Machine Learning for IoT – HW1
Davide Napolitano, Edoardo Lardizzone – Group20

In the first exercise, we write a script that is able to read a csv file with some data on it and to copy them into a TensorFlow dataset, with the goal to minimize the storage space. Especially about data, the first three elements are the datatime, the temperature and the humidity respectively, instead the last data in every row of the csv is the name of an audio wav file inside the same folder.
First of all we instantiate a loop that iteratively read the dataset that we import as a pandas dataframe (the samples.csv), then, for every row, the date is converted to POSIX via the datetime class with the *.timestamp()* method, the temperature and humidity are not modified and the audio is opened with the *tf.io.readfile()* functions which returns a String Tensor that is not normalized in [-1,1] (it could have led to problems of quality loss), then casted to numpy to be passed to the list.
In order to create the TensorFlow dataset, we have to choose between three different datatype in order to minimize the size of the created file. We run several attempts, with a csv of length 10, obtaining:

Date	Temperature	Humidity	Audio	Size(bytes)
Int64	Int64	Int64	Int64	2667938
Int64	Int64	Int64	Float32	1921070
Int64	**Int64**	**Int64**	**Bytes**	**961510**
Int64	Float32	Float32	Bytes	961570
Bytes	Int64	Int64	Bytes	961630
Float32	Int64	Int64	Bytes	961500
Int64	Bytes	Bytes	Bytes	961530

We check that using the Float type and the Bytes type for the datetime, the number was casted in a wrong way due to approximation, so we excluded them. Generally, we notice that using a Bytes list to memorize the audio led to smaller sizes so the final combination we used was: **Int64, Int64, Int64, Bytes**. We also tried to cast the first three data to Int32 and to force them in the *tf.train()* as indicated in the documentation, but the results were always the same.
The final action was to check if the audio quality was untouched, in order to do that we use the *tf.equal()* method that confront two tensors, especially we confront the output of *tf.io.read_file()* with what was saved into TFRecord noticing that using a Byteslist the audio was exactly the same. The output here is a TensorFlow dataset with three integers and a list of bytes for every row of the original dataset.

For the second exercise, the request is to record and pre-process an audio, optimizing the power consumption in order to not exceed a time constraint of 80ms. We start by defining every constant variable before the loop of num_samples. Between these variables there are: num_spectogram_bins = 321, since we always have the same size, the linear_to_mel_matrix, the definition of the audio, of the stream and of the BytesIO() buffer.
The first Popen is used just before the main loop in order to reset the time statistic for each frequency and to do this we put the command to *.wait()* to be sure that the rest of the code is not executed before in parallel.
For each iteration we start the stream, than we call the Popen in order to put the governor to powersave, we set the pointer of the buffer at the beginning with *.seek(0)* and the we start the loop of recording, in which the read data are written inside the buffer to have them only on the RAM.
Inside this last cycle, we set the governor to performance before the read of the last chunk, to be sure that the preprocessing operations will be computed inside the 80ms.
We apply the two Popen inside the recording since the change of governor doesn't interfere with the read of each chunk. We also try to use the Popen outside the recording, since ideally, by analysing the time for each operation both in performance and in powersave, the resample could be done in powersave by staying inside the margin of 80ms, however the time to execute the Popen would be too long (60ms, about 4 time longer than the time to resample in powersave, so the stft executional time would be huge) and moreover added to the time to compute the whole preprocessing.
About the preprocessing phase, for the resample we take the data from the buffer with the *.get value()* method and we convert them to numpy with *np.frombuffer* (dtype=int16), for the operations instead the only difference from the code in the Lab2 is that the audio is not decoded/encoded to have data in [-1,1] but it is before converted to tensor (dtype=float32) and then normalized with a TensorFlow operation to be inside the same range ($2 \cdot \frac{value - float\_min}{float_{max} - float\_min} - 1$). Here the output are a .bin files where we have the first ten components of the MFCC of a recorded audio saved as binary data. The average results for the time in each frequency are: VF_min 600MHz=476; VF_max 1.5GHz=61.