HW1: Sensor Fusion & TFRecord Datasets

Machine Learning for IOT

Group 2

Dri Emanuele s280133

Malan Erich s267475

Nicolì Alessandro s278091

Exercise 4

Objective: realize a Python script to generate a TFRecordDataset containing temperature, humidity, and one-second audio samples (Int16 resolution at 48KHz), collected at different datetimes and stored as raw data.

Execution: the script takes as input the *raw_data* folder and the name of the output file. Thus the first step is to read the *sample.csv* file inside the *raw_data* folder and to store it in a Pandas dataframe. Subsequently the TFRecordWriter class is used to write on the output file the data in the new format: for each dataframe's row the first two elements are joined together to obtain the input of the mktime function which outputs a timestamp in posix format, then the last element is used to specify the path of the audio file to read using tf.io.read_file(), the resulting string tensor is given to a function appositely defined in order to create the relative feature. Regarding this aspect, other than specify the structure of each record through the definition of a key-value based dictionary (Example) we obviously needed to define the value for each key of the record, i.e. the Feature and the data-type for each of these values. We identified as most suitable the 64-bit integer data type for the *posix* timestamp, the humidity and the temperature readings, while for the audio further experiments were conducted to determine which among tf.train classes: BytesList, FloatList and Int64List could be the less space-requiring alternative. For the FloatList option it was necessary to pass the audio string tensor to tf.audio.decode_wav() in order to obtain a float32 tensor, while for Int64List the wav file was read using wavfile.read (which outputs the rate and an integer tensor). The best option however was shown to be to directly use the audio string tensor to create a BytesList Feature (385 kB for 4 rows). Example is then serialized and finally, at the end of the loop, the output's size is assessed using os.path.getsize().

BytesList	FloatList	Int64List
384604	768428	422822

Exercise 5

Objective: Realize a Python script that given N as number of steps, records N-times 1 sec of audio, then processes it in order to extract the mfccs in the minimum possible time in high performance mode, subject to the hard constraint: processing_time<80ms.

Execution: The script takes as input the number of samples and the output folder where to store the mfccs, which in case of presence is erased. The import packages have been optimized to load only the necessary classes. In order to optimize the execution time every other possible constant is defined at the beginning of the script. The chunks size is one of the most meaningful optimization hyperparameters, as it must be a submultiple of the sample rate and should be the same size of the time needed by the system to switch from powersave to performance mode, so that before recording the program switches in low-profile mode, and before the last chunk reading operation it turns back to performance-mode in time to compute the intensive part of processing, in our case (kernel 5.4.72) the switch took 0.0519s (2490 frames for the sample rate of 48000), and the minimum optimum chunk has been declared as 4800 frames, even if 3000 frames is the min(submultiple) > 2490, but differently from k. 5.4.51 it involved errors so it was not feasible. The process begins in low performance mode by starting the stream of PyAudio class, then after having read (n-1) chunks it toggles high performances in non blocking mode. The optimal-point is chosen to start the processing in high performances and to minimize the high voltage over the record time where it is unnecessary. Then we stopped the stream, we saved the audio recorded into a buffer in order to avoid writing/reading onto the disk (using BytesIO), we performed the resampling and normalization and then we converted the result to compute the stft of tensorflow which is the most intensive part of the script (HV uses 40/50 ms over 65 of avg per cicle). The mfccs are extracted by the specific function of tensorflow that gets as input the product of the stft converted in decibels and then mapped into the mel scale, the reshaping functions have been removed because were not necessary. Finally it is stored into the specified folder through a serialization process and write_file function provided by tensor_flow, and the timings (end-start times of each iteration, and profile/time passed until the end) are printed.

AVG time (ms)	Max (ms)	Min (ms)
1064,83	1072,74	1062,32

	VFmin = 0.81V @ 0.6GHz	VFmax = 0.86V @ 1.5GHz	TOTAL
Time (ms)	4850	560	5410