Machine Learning for IoT - Homework 1

Luca Campana
Student id: s290065

Gianvito Liturri
Student id: s290464

Alberto Morcavallo
Student id: s290086

EXERCISE 1: TEMPERATURE AND HUMIDITY DATASET WITH TFRECORD

In the first exercise, our task is to convert a *csv* file into its corresponding *TFRecord*. This must be done in two ways: by applying a min-max normalization on records, and by simply copying data as it is.

For the first case, we choose to apply the normalization according to boundaries given by the specifications written on the sensor's data sheet. In this way, we avoid to iterate over data to find the maximum and minimum values, and furthermore we get coherent normalized values among different collections. After normalization, data is mapped into $[0, 1]$ interval, so it is handled as a floating point object. As a consequence, it gets stored with the tailored *FloatList* container. Another possible way to store data could have been to transform it into the equivalent byte string, to be stored in *BytesList* containers: we also tried this approach and we saw that the output file size did not change, while the time required for the operations increased. Thus, we choose to follow the former approach.

In this second case, we followed a slightly different reasoning. As we can see from the DHT-11 data sheet, both temperatures (Celsius grades) and humidities (percentual points) are recorded with a resolution equal to 1. As a consequence, all recordings are integer numbers, and can be stored in the optimized container *Int64List*. This last container has been also used to store, in both cases, the data referring to time in which records have been generated. In fact, we had to store them with the *POSIX* format, which can only assume integer values.

The script has been tested on a *csv* file containing 48 different recordings, with the following outcomes: for the normalized version, we obtained an average row size of $84B$, while for the other one we got an average row size of $78B$.

If normalization is required, to reduce the *TFR* storage requirements is useful to create the *Non-Normalized* version and then to apply normalization only during the reading phase.

EXERCISE 2: AUDIO PRE-PROCESSING OPTIMIZATION

The second exercise focused on the optimization process for the *MFCCs* extraction from 1-second length *wav* files.

After having decoded the *wav* files and having performed the slow extractions, for which we obtained an average processing time of $52ms$, we tried to change the transformations parameters, in order to obtain a faster processing.

The first things we tried to change have been the frequency boundaries used to build the matrix allowing to pass from spectrogram bins to Mel-frequency ones. Since those parameters does not change the matrix dimensions, their contribute in terms of time and *SNR* gain was negligible.

Then, we tried to vary the following set of parameters:

- *Sampling rate, windows length, windows step*. To respect the first given constraint, the fast execution had to produce the same number of splits of the slow one; moreover, splits had to refer to the same audio slices, so we had a set of possible combinations, as shown in Table I.
- *FFT length*. This had to be a power of 2, at least equal to *windows length*.
- *Number of Mel bins*. It defines the shape of the Mel-spectrogram matrix, so, since we want to extract the first 10 *MFCCs*, it had to be at least equal to 10.

TABLE I

Sampling Rate	*Window Length*	*Window Step*
16kHz	256	128
8kHz	128	64
4kHz	64	32

Modifying these parameters, we were not able to maintain both processing time and *SNR* at optimal levels: often, one of the two was good, but the other diverged. So, we decided to try another strategy: we casted samples to *float32* tensors before computing the *STFT*. We found out that, in this way, we reduced execution time by about the 40%. Despite using this, we still could not achieve good results: so, our last idea was to act directly on the spectrogram dimensions. Since its shape was $(124, 65)$, we thought that we could save some computation by resizing it to a more compact $(16, 16)$ version. In fact, since spectrograms are effectively images, we can exploit image-tailored methods to manage them. In order to match the final MFCC shape, we had to reconvert our samples to $(124, 32)$ matrices, after having applied the dot product. Finally, acting on the described parameters, we managed to obtain an average processing time of $17.4ms$ and an average *SNR* of $11.9dB$. These results were obtained with the following parameters:

- *Sampling rate*: $4kHz$;
- *Window length*: 64;
- *Window step*: 32;
- *FFT length*: 64;
- *Number of Mel bins*: 10.

Obviously, in order to avoid unneeded operations, we calculated the *linear_to_mel_weight_matrix* only for the 1^{st} sample, and exploit it in all the following conversions, such that the computation time is distributed over the whole dataset.