HW 1 Group 14

Scorca Francesco s288876 Dicataldo Michele s290091 Padovano Dario s291475

1. Temperature and Humidity Dataset with TFRecord

1.1. Data type selection

The objective is to find the best data type to use for each field (date,temperature,humidity), in order to minimize the storage requirements. For this purpose, we set the type of the three fields to $int64$, in the case the normalization is not required, and we set $float$ to temperature and humidity, and $int64$ to the date field. What we get on a dataset of 16 lines, is that the size of the output is $1392byte$ for the first case, and $1408byte$ for the second case. This because the float type needs more memory for the presence of an overhead.

1.2. Pipeline steps

Assuming that normalization is required, the best solution to minimize storage is to save the three values as $int64$ in a single TFRecord feature. Indeed, test have shown that when reading the saved dataset, values present no approximation error, and we have a memory storage improvement of about 30%, with respect to save the fields in three different features. Then, we need to map the examples to a single[1] tensor with 3 entries. Eventually, we can cast humidity and temperature values to float and perform normalization.

2. Audio pre-processing optimization

The objective is to define a preprocessing routine referred as $MFCC_{fast}$ returning tensor of shape [124, 10][2], minimizing execution time and maximizing signal-to-noise ratio:

$$SNR = \frac{\|MFCC_{fast}\|}{\|MFCC_{slow} - MFCC_{fast} + 10^{-6}\|} \quad (1)$$

Since this will drive the following choices, it is important to notice that this metric does not measure the absolute quality of the result, but how similar it is to the one obtained through $MFCC_{slow}$. Table 1 reports the tested parameters.

PARAMETER	VALUES TESTED
Number of mel bins	from 10 to 39
Mel lower frequency	0, 20, 40, 70, 100, 130
Mel upper frequency	1000, 2000, 3000, 4000
Resampling	True, False

Table 1: Tested parameters.

2.1. Number of Mel-bins filters

The number of filters applied to the log-spectrogram is supposed to decrease execution time, reducing mathematical computation, but to complete drop some information. In the tests, we find a monotonic increase of SNR, but some irregularities in the increase of execution time, which make us suppose that the algorithm is optimized for powers of 2. This allows us to choose 32 bins for $MFFC_{fast}$, whose computation takes less time than using less filters.

2.2. Re-sampling

Re-sampling the tracks, from 16kHz to 8kHz, is supposed to decrease execution time, sacrificing some quality. In spite of this, the tests do not show any speed increase, while the SNR drops[3] from 22.37dB to 15.94dB. We hypothesize that the time gained in the following computation is completely spent for the re-sampling process.

2.3. Frequency ranges

Fixing 32 filters, we have to tune from which frequency they are sampled, to try to make them catch as much information as the 40 of $MFCC_{slow}$. This is not supposed to impact on execution time, since we are just varying values, not dimensions. Independently from re-sampling, the best range always results from 0 Hz to 4kHz[4].

2.4. Conclusion

The final implementation does not re-sample the audio tracks and employs 32 filters, sampled from 0 Hz to 4kHz. The developed routine reduces of 32% the execution time, from 25ms to 17ms, returning an SNR = 22.37dB.

[1] The read dataset entries will be tensors of tensors, since we have saved the fields in a single dictionary

[2] the one returned by $MFCC_{slow}$

[3] considering the optimal frequency range, and using 32 filters

[4] Notice that in the re-sampling case this is the highest value testable, being the Nyquist frequency