Nelly Gaillard, s282613
Giuseppe Galilei, s295620
Beatrice Alessandra Motetti, s287618

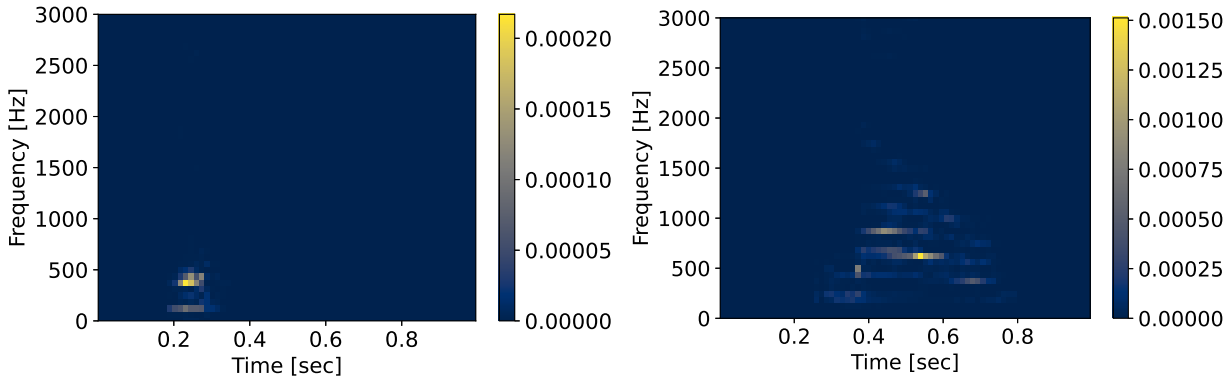
Exercise 1

Our goal is to reduce the storage requirements for an input data set. We take advantage of the TFRecord format to serialize the data. w We know from the DHT-11 datasheet that the resolution of the temperature measurement is $\pm 1°C$ and $1\%RH$ for the humidity. Thus we know that we can encode these values as integers without losing any information. Since the datetime has also to be stored as a POSIX timestamp, also this data can be represented by using an integer value. When we apply the min-max normalization on the temperature and humidity samples, we obtain as results floating point numbers. Thus encoding them as integer types would lead to quality loss. The aforementioned considerations led us to the choice of the *tf.train.Int64List* data-type for the temperature and the humidity value if normalization is not applied, else the *tf.train.FloatList* for the temperature and humidity samples. We can always represent the POSIX timestamp with the *tf.train.Int64List* data-type. As example, we consider a csv file containing the 4 samples reported in the text of the Homework. We obtain that the size in bytes of the TFRecord files are 332B and 356B, without and with normalization respectively. This can be explained by the fact that storing a float is more expensive than an integer value in terms of bytes. Thus we conclude that, to minimize the storage requirements, it is more convenient to apply normalization when loading the data than before storing it on disk.

Exercise 2

Given the input data set, the goal of this exercise is to build an audio pre-processing pipeline to extract the MFCCs, that optimizes the average execution time (i.e. < 18 ms) while preserving the quality of the input signal (SNR > 10.40 dB).

First of all we observed that the parameters which have an influence on the speed are the number of Mel bins (i.e. the number of filters of the MFCC) and the sampling rate, while to get a higher quality we can widen the frequency range or increase the number of filters. While the frequency range does not have any effect on the speed, for the number of bins we need to find a trade-off which allows to respect both the speed and quality constraints. In addition, by performing down-sampling on the audio signal we can reduce the time for the spectrogram and MFFCs computation, but we add a supplementary step in the pre-processing pipeline which is computationally expensive. We set up a grid search ({downsampling factor: {1, 2, 4}, lower frequency: {20, 50, 200, 300, 500, 1000, 1500}, upper frequency: {1000, 1500, 2000, 3000, 4000}, num mel bins: {15, 20, 25}}) to find out the best parameters combination to tune the fast pre-processing pipeline, while satisfying the time and SNR constraints. The best configuration turned out to be the following: {num mel bins = 15, lower frequency = 300, upper frequency = 1000, downsampling factor = 4}, that led to a computational time of 17 ms and a SNR of 11.31 dB. In support of this, the spectrograms in Fig. 1 show how the frequencies are actually mostly in the selected range. For what concerns the number of bins and the downsampling factor, the choice was driven by the requested boundaries. In fact, with a downsampling factor of 2, the average time is attested around 20ms and the same holds for higher number of bins too.



(a) A Spectrogram of a "yes".

(b) Spectrogram of a "no".

Figure 1: Spectrograms of two random audio files sampled randomly from the data set.