Machine Learning for IoT
Lab 2 – Pre-processing

<u>Exercise 1:</u> Timeseries pre-processing

1.1 In VS Code, continuously run the battery monitoring script developed in *LAB 1 – Exercise 2*.

1.2 In Deepnote, create a new script to set the following retention rules on your Redis TimeSeries database:

a. Set the retention period of the *mac_address:*battery and *mac_address:*power timeseries to 1 day.
b. Create two timeseries with a chunk size of 128 bytes, called *mac_address:*battery_avg and *mac_address:*power_avg, that store the average over every 30s of *mac_address:*battery and *mac_address:*power, respectively. Set the retention period of the two timeseries to 30 days.
c. Create a timeseries with a chunk size of 128 bytes, called *mac_address:*battery_min, that stores the minimum over every 1 minute of *mac_address:*battery. Set the retention period of the timeseries to 30 days.
d. Create a timeseries with a chunk size of 128 bytes, called *mac_address:*battery_max that stores the maximum over every 1 minute of *mac_address:*battery. Set the retention period of the timeseries to 30 days.
e. For all timeseries, check the number of samples and their memory size after at least 15 minutes of monitoring.
f. Repeat the steps a., b., c., and e., disabling compression (create new timeseries for each step adding "_uncompressed" to the original names). Report the results in the following table and comment the collected statistics:

Timeseries Name	**# of Samples**	**Compressed Size (KB)**	**Uncompres. Size (KB)**
*mac_address:*battery			
*mac_address:*power			
*mac_address:*battery_avg			
*mac_address:*power_avg			
*mac_address:*battery_min			
*mac_address:*battery_max			

g. For each timeseries, estimate the maximum number of samples and the maximum memory size, considering both the compressed and the uncompressed versions. Report the results in a table.
h. Plot the created timeseries in Deepnote.

<u>Exercise 2:</u> Audio Features Extraction & Visualization

2.1 In Deepnote, create a Python script (e.g., *preprocessing.py*) with the following Python classes:

Name	**AudioReader**
Description	Read audio data and its label from a WAV file.
Args	• **resolution**: *tf.dtype.Dtype*. Resolution of the WAV file (e.g., tf.int16). • **sampling_rate**: *int*. Sampling rate of the WAV file in Hz.
Methods	**get_audio** **Args** • **filename**: *str*. The path of a WAV file. **Returns** • **audio**: 1D *Tensor* of type *float32*. **get_label** **Args** • **filename**: *str*. The path of a WAV file. **Returns** • **label**: A *Tensor* of type *str*. **get_audio_and_label** **Args** • **filename**: *str*. The path of a WAV file. **Returns** A tuple of two tensors: • **audio**: 1D *Tensor* of type *float32*. • **label**: A *Tensor* of type *str*.

Name	**Spectrogram**
Description	Compute the magnitude of the STFT of an audio tensor.
Args	• **sampling_rate**: *int*. Sampling rate of the WAV file in Hz. • **frame_length_in_s**: *int*. Frame length (in seconds) of the STFT. • **frame_step_in_s**: *int*. Frame step (in seconds) of the STFT.
Methods	**get_spectrogram** **Args** • **audio**: 1D *Tensor* of type *float32*. **Returns** • **spectrogram**: 2D *Tensor* of type *float32*.

get_spectrogram_and_label

Args

- **audio**: 1D *Tensor* of type *float32*.

Returns A tuple of two tensors:

- **spectrogram**: 2D *Tensor* of type *float32*.
- **label**: A *Tensor* of type *str*.

Name **MelSpectrogram**

Description Compute log-Mel spectrogram of an audio tensor.

Args

- **sampling_rate**: *int*. Sampling rate of the WAV file in Hz.
- **frame_length_in_s**: *int*. Frame length (in seconds) of the STFT.
- **frame_step_in_s**: *int*. Frame step (in seconds) of the STFT.
- **num_mel_bins**: *int*. Number of Mel bins of the Mel spectrogram
- **lower_frequency**: *float*. Lower bound on the frequencies to be included in the Mel spectrum.
- **upper_frequency**: *float*. Upper bound on the frequencies to be included in the Mel spectrum.

Methods

get_mel_spec

Args

- **audio**: 1D *Tensor* of type *float32*.

Returns

- **log_mel_spectrogram**: 2D *Tensor* of type *float32*.

get_mel_spec_and_label

Args

- **audio**: 1D *Tensor* of type *float32*.

Returns A tuple of two tensors:

- **log_mel_spectrogram**: 2D *Tensor* of type *float32*.
- **label**: A *Tensor* of type *str*.

Name **MFCC**

Description Compute the MFCCs of an audio tensor.

Args

- **sampling_rate**: *int*. Sampling rate of the WAV file in Hz.
- **frame_length_in_s**: *int*. Frame length (in seconds) of the STFT.
- **frame_step_in_s**: *int*. Frame step (in seconds) of the STFT.
- **num_mel_bins**: *int*. Number of Mel bins of the Mel spectrogram
- **lower_frequency**: *float*. Lower bound on the frequencies to be included in the Mel spectrum.
- **upper_frequency**: *float*. Upper bound on the frequencies to be included in the Mel spectrum.

Methods	**get_mfccs** **Args** • **audio**: 1D *Tensor* of type *float32*. **Returns** • **mfccs**: 2D *Tensor* of type *float32*. **get_mfccs_and_label** **Args** • **audio**: 1D *Tensor* of type *float32*. **Returns** A tuple of two tensors: • **mfccs**: 2D *Tensor* of type *float32*. • **label**: A *Tensor* of type *str*.

2.2 In Deepnote, create a notebook for testing the functions of *preprocessing.py*.

a) Verify that the functions return the expected outputs.

b) Measure the latency (i.e., the execution time) using the *time* method from the *time* module and fill out the following table:

Function	**Args values**	**Latency (ms)**
Spectrogram		
MelSpectrogram		
MFCC		
…		

c) Try different args values in the following ranges

- STFT frame length ∈ [10, 50] ms. Try also with power-of-two values like 8ms, 16ms, 32ms, and comment the results.
- STFT frame step such that overlap ∈ {0%, 25%, 50%, 75%}.
- # of mel bins ∈ [10, 128].
- Mel lower frequency ∈ [0, 80] Hz.
- Mel upper frequency ∈ [2000, 8000] Hz.
- # of MFCCs ∈ [10, 40].

2.3 Use the *visualize* function from the *visualization.py* script (uploaded on Deepnote) to visualize the audio features of the files contained in the *examples* folder. Comment the results.

<u>Exercise 3:</u> Voice Activity Detection

3.1 In Deepnote, develop a Python function for Voice Activity Detection (VAD) based on log-Mel spectrogram features (see the table below for the specifications).

Name	**VAD**
Description	Classify an audio file as *silence* or *not silence* using log-Mel spectrogram features. The VAD algorithm to be implemented consists of the following steps: 1. Compute the log-Mel spectrogram. 2. Convert the amplitude to decibels relative to full scale (dbFS) 3. For each time frame of the log-Mel spectrogram, compute the average amplitude over the frequencies. 4. Count the number of frames with dbFS greater than a pre-defined threshold (*dbFSthres*) and mark them as *not silence*. 5. Compute the duration in second of *not silence* frames. 6. If the duration is greater than a pre-defined threshold (*duration_thres*), classify the audio as *not silent*, *silent* otherwise.
Args	• **sampling_rate**: *int*. Sampling rate of the WAV file in Hz. • **frame_length_in_s**: *int*. Frame length (in seconds) of the STFT. • **num_mel_bins**: *int*. Number of Mel bins of the Mel spectrogram • **lower_frequency**: *float*. Lower bound on the frequencies to be included in the Mel spectrum. • **upper_frequency**: *float*. Upper bound on the frequencies to be included in the Mel spectrum. • **dbFSthres**: *int*. dbFS threshold. • **duration_thres**: *int*. Voice duration threshold in seconds.
Methods	**is_silence** **Args** • **audio**: 1D *Tensor* of type *float32*. **Returns** • **is_silence**: An *int* equal to 1, if the audio has been classified as silence, 0 otherwise.

3.2 Use the *vad-dataset* uploaded on Deepnote to test the VAD function. Measure the accuracy and the average latency (in ms) of VAD at different values of the input arguments. Report the collected results in a table.