Machine Learning for IoT
Lab 2 – Pre-processing

Exercise 1: Audio Features Extraction & Visualization

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

Name	**get_audio_and_label**
Description	Read audio data and its label from a WAV file.
Args	**filename**: *str*. Path of an audio file.
Returns	A tuple of three *Tensor* objects: • **audio**: A *Tensor* of type *float32*. • **sampling_rate**: A *Tensor* of type *int32*. • **label**: A *Tensor* of type *string*.

Name	**get_spectrogram**
Description	Compute the magnitude of the STFT of a WAV file.
Args	• **filename**: *str*. The path of a WAV file. • **downsampling_rate**: A *Tensor* of type *int*. Sampling rate after downsampling. Set equal to the original sampling rate to skip downsampling. • **frame_length_in_s**: *int*. Frame length (in seconds) of the STFT. • **frame_step_in_s**: *int*. Frame step (in seconds) of the STFT.
Returns	A tuple of three *Tensor* objects: • **spectrogram**: A *Tensor* of type *float32*. • **sampling_rate**: A *Tensor* of type *int32*. • **label**: A *Tensor* of type *string*.

Name	**get_log_mel_spectrogram**
Description	Compute log-Mel spectrogram of a WAV file.
Args	• **filename**: *str*. The path of the file. • **downsampling_rate**: A *Tensor* of type *int*. Sampling rate after downsampling. Set equal to the original sampling rate to skip downsampling. • **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.

Returns A tuple of two *Tensor* objects:

- **log_mel_spectrogram**: A *Tensor* of type *float32*.
- **label**: A *Tensor* of type *string*.

Name **get_mfccs**

Description Compute the MFCCs of a WAV file.

Args

- **filename**: *str*. The path of the file.
- **downsampling_rate**: A *Tensor* of type *int*. Sampling rate after downsampling. Set equal to the original sampling rate to skip downsampling.
- **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.
- **num_coefficients**: *float*. Number of MFCCs.

Returns A tuple of two *Tensor* objects:

- **mfccs**: A *Tensor* of type *float32*.
- **label**: A *Tensor* of type *string*.

1.2 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)**
tfio.audio.resample		
get_spectrogram		
get_log_mel_spectrogram		
get_mfccs		
tf.image.resize		
…		

c) Try different args values in the following ranges
- Downsampling rate ∈ [4000, 16000] Hz.
- STFT frame length ∈ [10, 50] ms. Try also with power-of-two values like 8ms, 16 ms, 32 ms, and comment the results.
- STFT frame step such that overlap ∈ {0%, 25%, 50%, 75%}.
- # of mel bins ∈ [10, 40].
- Mel lower frequency ∈ [20, 80] Hz.
- Mel upper frequency ∈ [2000, 8000] Hz.
- # of MFCCs ∈ [10, 40].

1.3 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.

Exercise 2: Voice Activity Detection

1.1 In Deepnote, develop a Python function for Voice Activity Detection (VAD) based on STFT features (see the table below for the specifications).

Name	**is_silence**
Description	The function classifies an audio file as *silence* or *not silence* using STFT features. Specifically, the VAD algorithm to be implemented consists of the following steps 1. Compute the STFT. 2. For each time frame of the STFT, compute the average amplitude over the frequencies. 3. Convert to amplitude to decibels relative to full scale (dbFS) 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	• **filename**: *string*. Path of a WAV file. • **downsampling_rate**: *int*. Sampling rate after downsampling (in Hz). • **frame_length_in_s**: *int*. Frame length of the STFT in seconds. • **dbFSthres**: *int*. dbFS threshold. • **duration_thres**: *int*. Duration threshold in seconds.
Returns	An *int* equal to 1, if the audio has been classified as silence, 0 otherwise.

1.2 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 args. Report the collected results in a table, e.g.:

Rate (Hz)	**Frame Length (s)**	**dbFS thres.**	**Duration thres. (s)**	**Acc. (%)**	**Lat. (ms)**
16000	0.02	-100	0.1		
16000	0.02	-110	0.1		
…	…	…	…	…	…

<u>Exercise 3:</u> Timeseries pre-processing

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

3.2 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)**	**Uncompress. 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.