Machine Learning for IoT
Lab 3 – Training & Deployment

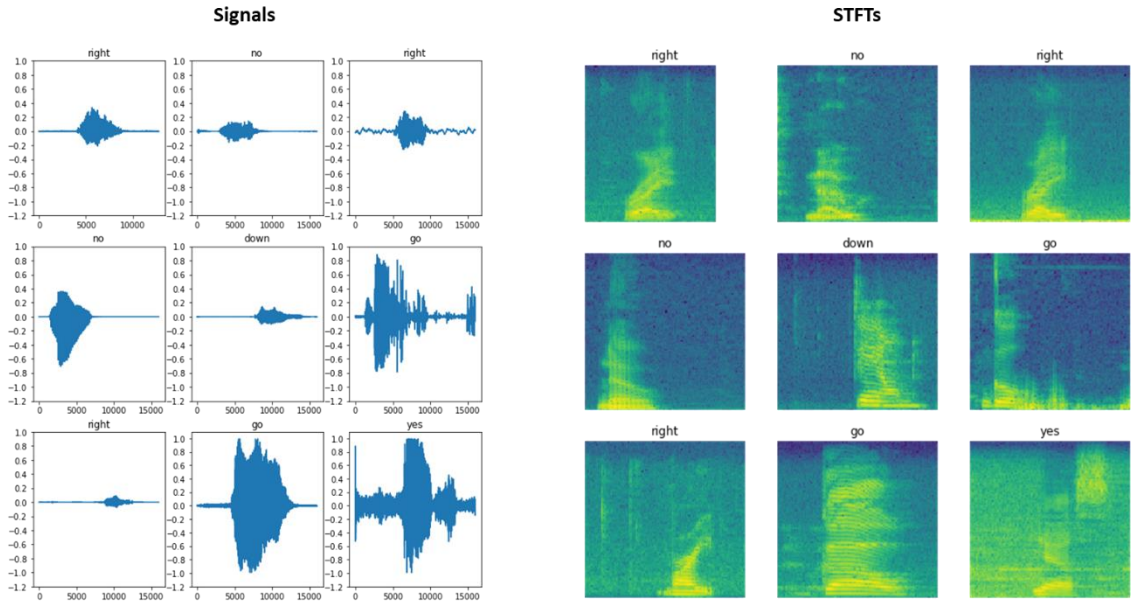
Dataset 1: Jena Climate dataset

The Jena Climate dataset is a weather time series dataset containing 14 features (e.g. temperature, atmospheric pressure, humidity) collected every 10 minutes from 2009 to 2016.

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
5	996.50	-8.05	265.38	-8.78	94.4	3.33	3.14	0.19	1.96	3.15	1307.86	0.21	0.63	192.7
11	996.62	-8.88	264.54	-9.77	93.2	3.12	2.90	0.21	1.81	2.91	1312.25	0.25	0.63	190.3
17	996.84	-8.81	264.59	-9.66	93.5	3.13	2.93	0.20	1.83	2.94	1312.18	0.18	0.63	167.2
23	996.99	-9.05	264.34	-10.02	92.6	3.07	2.85	0.23	1.78	2.85	1313.61	0.10	0.38	240.0
29	997.46	-9.63	263.72	-10.65	92.2	2.94	2.71	0.23	1.69	2.71	1317.19	0.40	0.88	157.0

Dataset 2: Mini Speech Command Dataset

The Mini Speech Command dataset collects 8000 samples of eight keywords ('down', 'no', 'go', 'yes', 'stop', 'up', 'right', 'left'), 1000 samples per label. Each sample is recorded at 16kHz with an Int16 resolution and has a variable duration (1s the longest).



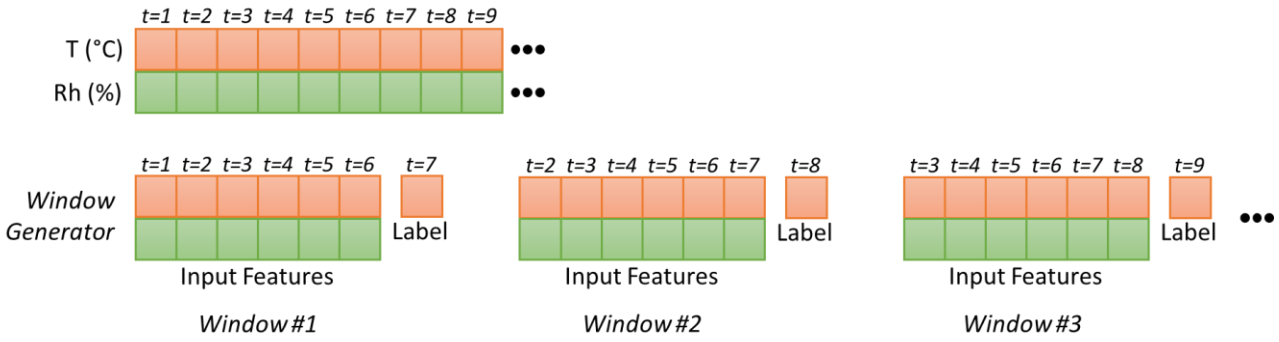
Exercise 1: Temperature and Humidity Forecasting

Write a Python script to train and evaluate different models for temperature forecasting on the Jena Climate dataset. Select as input features of your model six consecutive temperature and humidity measurements. The model should predict the temperature for the next time instant.

1.1. Download and load the dataset:

- Use the *get_file* method from *tf.keras.utils*. Link: https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip
- Load the dataset using the *read_csv* method from *pandas*.
- Select the temperature and humidity columns (#2 and #5).
- Convert the data into a 32-bit float *numpy* array.
- Split the data into three different sets: train (70%), validation (10%), test (20%)

1.2. Develop a Python class called *WindowGenerator* to implement the data preparation and pre-processing stages. The output is a *tf.data.Dataset* composed by six-value temperature and humidity windows (the shape of each window is 6×2) and the temperature labels for each window. Normalize each window with the mean and standard deviation of the training set. Use the *WindowGenerator* to create three instances of *tf.data.Dataset* for train, validation, and test, respectively.



1.3. Develop three different models (see the Table below) using the Keras API:

- A multilayer perceptron (MLP).
- A convolutional neural network (CNN-1D).
- An LSTM.

MLP	**CNN-1D**	**LSTM**

`Flatten()`
`Dense(units=128)`
`ReLU()`
`Dense(units=128)`
`ReLU()`
`Dense(units=1)`
	`Conv1D(filters=64, kernel_size=3)`
	`ReLU()`
	`Flatten()`
	`Dense(units=64)`
	`ReLU()`
	`Dense(units=1)`
		`LSTM(units=64)`
		`Flatten()`
		`Dense(units=1)`

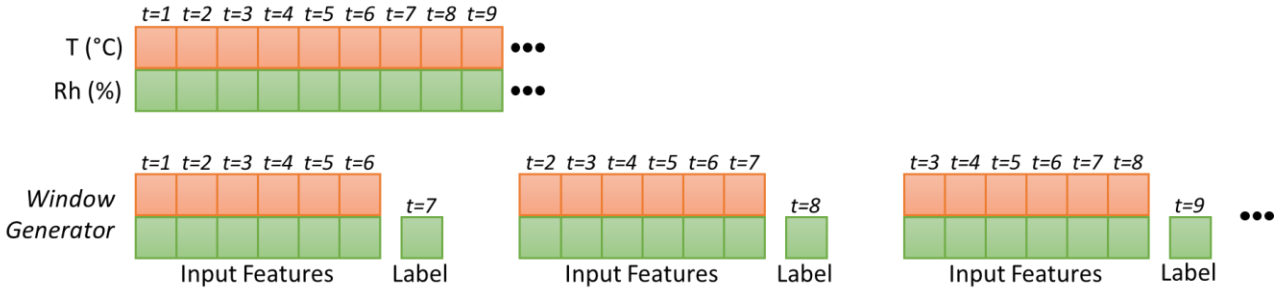
1.4. Train each model and evaluate the prediction error (use the Keras API):

- Select the *MeanSquaredError* (MSE) as loss function.
- Select the *MeanAbsoluteError* (MAE) as validation and test metric.
- Select *Adam* as optimizer (use default parameters).
- Train for 20 epochs.
- Measure the MAE on the test-set.
- Evaluate the memory requirements. Plot the number of parameters (#Params) vs. MAE for each model commenting the results.

1.5. Save the model on disk using the *save* method of the model object.

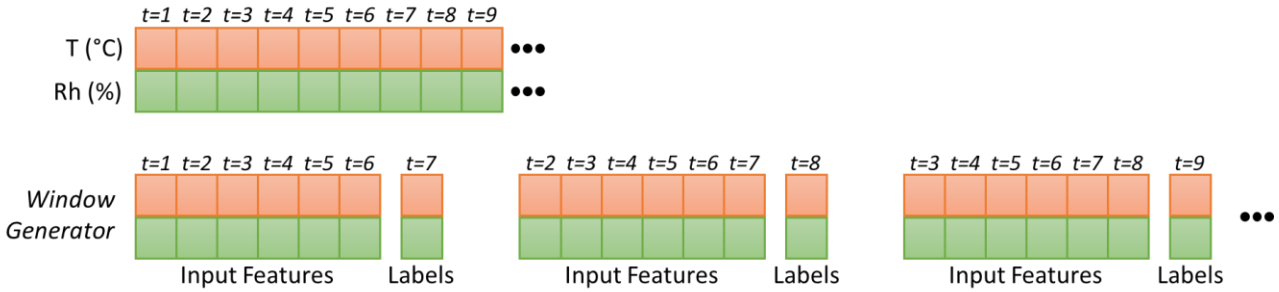
```
run_model = tf.function(lambda x: model(x))
concrete_func = run_model.get_concrete_function(tf.TensorSpec([1, 6, 2],
    tf.float32))
model.save(saved_model_dir, signatures=concrete_func)
```

1.6. Modify the script to train models for humidity forecasting. Repeat the experiments and the assessment. Comment the results.



1.7. Modify the script to train multi-output models that predict both temperature and humidity.

- Define a new class that inherits from *tf.keras.metrics.Metric* to measure the MAE on the two outputs separately. Use this class as test metric. Check the documentation: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Metric.
- Plot #Params vs. MAE of both single-output and multi-output models commenting the results.



1.8. Taken as reference the model that performs best, is there any modification you would suggest to reduce the memory footprint without affecting the prediction quality?

Exercise 2: Keyword Spotting

Write a Python script to train and evaluate different models for keyword spotting on the Mini Speech Command dataset.

2.1. Download the dataset from the following link: http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip

2.2. Develop a Python class called *SignalGenerator* to implement the data preparation and pre-processing steps.

- The *SignalGenerator* takes as input the list of keywords, the sampling rate, the STFT frame length, the STFT frame step, the number of Mel bins, the frequency range of the Mel spectrum, the number of MFCCs, and a binary variable called *mfcc* that indicates whether to use the STFT (if false) or the MFCCs (if true) as input features.
- The output is an instance of *tf.data.Dataset* that reads the audio samples from a list of paths, pads the signal to 1s (if needed), applies pre-processing (STFT or MFCC), and generates batches of tensors and related label IDs.
- Use the *SignalGenerator* to create three instances of *tf.data.Dataset*, for train, validation, and test. Use a random split of 80%/10%/10%, respectively.

2.3. Develop three different models (see the Tables below):

- A multilayer perceptron (MLP).
- A convolutional neural network (CNN).
- A depthwise-separable convolutional neural network (DS-CNN)

MLP
`Flatten()`
`Dense(units=256)`
`ReLU()`
`Dense(units=256)`
`ReLU()`
`Dense(units=256)`
`ReLU()`
`Dense(units=8)`

CNN
`Conv2D(filters=128, kernel_size=[3, 3], stride=strides, use_bias=False)`
`BatchNormalization(momentum=0.1)`
`ReLU()`
`Conv2D(filters=128, kernel_size=[3, 3], stride=[1, 1], use_bias=False)`
`BatchNormalization(momentum=0.1)`
`ReLU()`
`Conv2D(filters=128, kernel_size=[3, 3], stride=[1, 1], use_bias=False)`
`BatchNormalization(momentum=0.1)`
`ReLU()`
`GlobalAveragePooling2D()`
`Dense(units=8)`

`* strides = [2,2] if STFT, [2,1] if MFCC`

DS-CNN
`Conv2D(filters=256, kernel_size=[3, 3], stride=strides, use_bias=False)`
`BatchNormalization(momentum=0.1)`
`ReLU()`
`DepthwiseConv2D(kernel_size=[3, 3], stride=[1, 1], use_bias=False)`
`Conv2D(filters=256, kernel_size=[1, 1], stride=[1, 1], use_bias=False)`
`BatchNormalization(momentum=0.1)`
`ReLU()`
`DepthwiseConv2D(kernel_size=[3, 3], stride=[1, 1], use_bias=False)`
`Conv2D(filters=256, kernel_size=[1, 1], stride=[1, 1], use_bias=False)`
`BatchNormalization(momentum=0.1)`
`ReLU()`
`GlobalAveragePooling2D()`
`Dense(units=8)`

`* strides = [2,2] if STFT, [2,1] if MFCC`

2.4. Train each model and evaluate the prediction error using the STFT as input features.
- Set frame length=16ms, frame step=8ms.
- Resize the 124×129 STFT to 32×32 using *tf.image.resize*.
- Select the *SparseCategoricalCrossentropy* (with *from_logits*=True) as loss function.
- Select the *SparseCategoricalAccuracy* as validation and test metric.
- Select *Adam* as optimizer.
- Use the *ModelCheckpoint* callback to save the model with maximum accuracy on the validation set.
- Train for 20 epochs.

2.5. Train each model and evaluate the prediction error using the MFCCs as input features.
- Set frame length=40ms, frame step=20ms, #Mel bins=40, lower frequency=20Hz, upper frequency=4kHz, #MFCCs=10.
- Select the same training hyper-parameters as in 4.4.

2.6. Evaluate the memory requirements and the prediction quality of the different pre-processing strategies and models. Plot the results.

2.7. Save the models on disk.

2.8. Record 1000 samples of one-second audio signals of background noise (labelled as *silence*) at 48kHz and Int16 resolution. Resample the signals at 16kHz. Modify the training script to include the collected data and the new *silence* class. Repeat the experiments.

2.9. Taken as reference the model that performs best, is there any modification you would suggest to reduce the memory footprint without affecting the prediction quality?

Exercise 3: Deployment

3.1. Deploy and test the multi-output models for temperature and humidity forecasting on the Raspberry Pi:

- Generate the *tflite* models from the SavedModel

```
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_model = converter.convert()

with open(tflite_model_dir, 'wb') as fp:
    fp.write(tflite_model)
```

- Measure the *tflite* file size for each model (use *getsize* from *os.path*).
- Copy the *tflite* models on the board (use *scp*).
- Write a Python script that collect six consecutive values (sampling frequency = 1s) of temperature and humidity values with the DHT-11 sensor and make predictions with the *tflite* models.
- Compare measured vs. predicted values. Example output:

```
Measured: 20.0, 80.0 --- Predicted: 20.4,79.8 --- MAE: 0.4,0.2
```

- Write a Python script to measure the Latency (batch size=1) in ms for each model. Compute the average on 100 runs.

3.2. Deploy and test the models for keyword spotting.

- Measure the *tflite* file size for each model.
- Write a Python script that record one-second audio samples and classify the keyword.
- Measure the time (in ms) needed for pre-processing, inference, and the total execution time for the different strategies (with STFT vs. with MFCC) and models (MLP, CNN, DS-CNN).
- Compare the prediction quality of the models trained on the original dataset vs. that trained on the extended dataset.