Machine Learning for IoT

Lab 4 – Optimization

<u>Exercise 1:</u> Post-Training Quantization (PTQ)

1.1. Modify the training scripts (see Lab 3) to save the Tensorflow datasets on disk. For instance, to save the temperature and humidity datasets, use the following commands:

```
tf.data.experimental.save(train_ds, './th_train')
tf.data.experimental.save(val_ds, './th_val')
tf.data.experimental.save(test_ds, './th_test')
```

You can load the dataset in other scripts with the following commands:

```
tensor_specs = (tf.TensorSpec([None, 6, 2], dtype=tf.float32),
                tf.TensorSpec([None, 2]))
train_ds = tf.data.experimental.load('./th_train', tensor_specs)
val_ds = tf.data.experimental.load('./th_val', tensor_specs)
test_ds = tf.data.experimental.load('./th_test', tensor_specs)
```

1.2. On your notebook, write a Python script to evaluate the prediction quality of TFLite models. Verify that the FP32 TFLite models have the same prediction quality as the Keras models.

- Write an evaluation script to compute the MAE of TFLite multi-output models, using *numpy* methods. Run this script to evaluate temperature and humidity forecasting models in TFLite format.
- Write an evaluation script to compute the classification accuracy of TFLite models. Run this script to evaluate the keyword spotting models in TFLite format.

<u>N.B.:</u> Set the batch size to 1 when running inference with TFLite models:

```
test_ds = test_ds.unbatch().batch(1)
```

1.3. Apply Weights-Only PTQ to the multi-output models for temperature and humidity forecasting.

- Generate the TFLite models.

```
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter.convert()
```

- Save the TFLite model on disk and measure the file size.
- Evaluate the MAE of each model using the script of 1.1.
- Plot TFLite Size vs. Temperature MAE and TFLite Size vs Humidity MAE for each model (MLP, CNN-1D) and precision (FP32 and INT8).
- Deploy the quantized models on the Raspberry and measure the inference latency. Plot Latency vs. Temperature MAE and Latency vs Humidity MAE.

1.4. Apply Weights-Only PTQ to the keyword spotting models. Plot TFLite Size vs Accuracy for the different pre-processing strategies and models.

1.5. Apply PTQ (Weights + Activations) to the multi-output models for temperature and humidity forecasting.

```
def representative_dataset_gen():
    for x, _ in train_ds.take(1000):
        yield [x]

converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset_gen
tflite_quant_model = converter.convert()
```

- Plot TFLite Size vs. Temperature MAE and TFLite Size vs Humidity MAE for each model and precision.
- Plot Latency vs Accuracy for each model and precision.

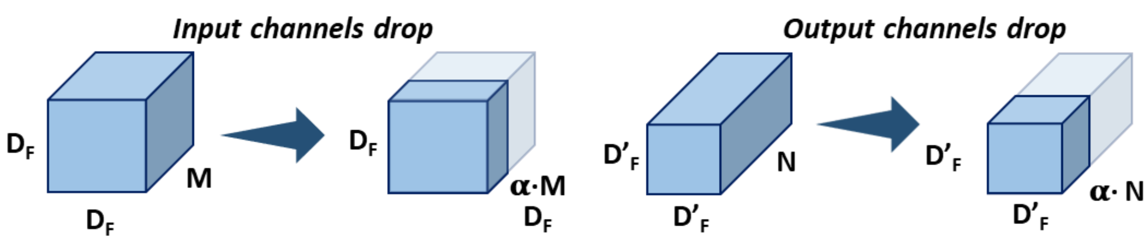
N.B.: PTQ (Weights + Activations) of CNN-1D is not supported.

1.6. Apply PTQ (Weights + Activations) to the keyword spotting models.
- Plot TFLite Size vs Accuracy.
- Plot Latency vs Accuracy.

1.7. For each task, which optimized model achieves the best accuracy-size-latency trade-off?

Exercise 2: Structured Pruning via Width Scaling

2.1. Modify the models for temperature and humidity forecasting to implement models with a parametric *width*:
- Multiply the number of filters (of convolutional layers) and the number of units (of dense and LSTM layers) by a parameter called the width multiplier $\alpha \epsilon (0, 1]$. All the layers share the same width multiplier.
- Train the models with different values of the width multiplier, i.e. 0.5 and 0.75. Evaluate the MAE, the TFLite Size, and Latency.



2.2. Repeat the same experiments with the models for keyword spotting.

2.3. Comment the results. Which is the best strategy for size optimization between quantization and structured pruning? Which is the best strategy for latency optimization?

Exercise 3: Magnitude-Based Pruning

3.1. Train the multi-output models for temperature and humidity forecasting with magnitude-based pruning

- Define the sparsity scheduler as follows:

```
import tensorflow_model_optimization as tfmot

pruning_params = {'pruning_schedule':
                        tfmot.sparsity.keras.PolynomialDecay(
                              initial_sparsity=0.30,
                              final_sparsity=0.8,
                              begin_step=len(train_ds)*5,
                              end_step=len(train_ds)*15)
                  }

prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude
model = prune_low_magnitude(model, **pruning_params)
```

- Define the pruning callback:

```
callbacks = [tfmot.sparsity.keras.UpdatePruningStep()]
```

- Train the model:

```
input_shape = [32, 6, 2]
model.build(input_shape)
model.fit(train_ds, epochs=20, validation_data=val_ds, callbacks=callbacks)
```

- Strip the model

```
model = tfmot.sparsity.keras.strip_pruning(model)
```

- Generate the TFLite model.
- Compress the model using the zlib and save the output on disk.

```
import zlib
tflite_model = converter.convert()
with open(tflite_model_dir, 'wb') as fp:
    tflite_compressed = zlib.compress(tflite_model)
    fp.write(tflite_compressed)
```

- Repeat the same experiment with *final_sparsity*=0.9

3.2. Repeat the same experiments with the keyword spotting models.

3.3. Evaluate the prediction quality, the TFLite size, and the latency. Comment the results.