AI on Chip
Lab2

TAs: course.aislab@gmail.com

Post-training integer quantization

- You can convert the trained model to TensorFlow Lite format using the *TFLiteConverter* API, and apply varying degrees of quantization.

```
[ ]  converter = tf.lite.TFLiteConverter.from_keras_model(model)
     tflite_model = converter.convert()
```

- Beware that some versions of quantization leave some of the data in float format. So the following sections show each option with increasing amounts of quantization, until we get a model that's entirely int8 or uint8 data.
- It's now a TensorFlow Lite model, but it's still using 32-bit float values for all parameter data.

Post-training integer quantization

- Now let's enable the default *optimizations* flag to quantize all fixed parameters (such as weights)

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite_model_quant = converter.convert()
```

using dynamic range quantization

- The model is now a bit smaller with quantized weights, but other variable data is still in float format.

Post-training integer quantization

- To quantize the variable data (such as model input/output and intermediates between layers), you need to provide a *RepresentativeDataset*. It allows the converter to estimate a dynamic range for all the variable data.

```
def representative_data_gen():
    for input_value in tf.data.Dataset.from_tensor_slices(train_images).batch(1).take(100):
        # Model has only one input so each data point has one element.
        yield [input_value]

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen

tflite_model_quant = converter.convert()
```

using float fallback quantization

- Now all weights and variable data are quantized, and the model is significantly smaller compared to the original TensorFlow Lite model.

Post-training integer quantization

- However, to maintain compatibility with applications that traditionally use float model input and output tensors, the TensorFlow Lite Converter leaves the model input and output tensors in float:

```
interpreter = tf.lite.Interpreter(model_content=tflite_model_quant)
input_type = interpreter.get_input_details()[0]['dtype']
print('input: ', input_type)
output_type = interpreter.get_output_details()[0]['dtype']
print('output: ', output_type)
```

```
input:  <class 'numpy.float32'>
output:  <class 'numpy.float32'>
```

- That's usually good for compatibility, but it won't be compatible with devices that perform only integer-based operations, such as the Edge TPU.

Post-training integer quantization

- So to ensure an end-to-end integer-only model, you need a couple more parameters...

```
def representative_data_gen():
    for input_value in tf.data.Dataset.from_tensor_slices(train_images).batch(1).take(100):
        yield [input_value]

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen
# Ensure that if any ops can't be quantized, the converter throws an error
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Set the input and output tensors to uint8 (APIs added in r2.3)
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8

tflite_model_quant = converter.convert()
```

using integer-only quantization

Post-training integer quantization

- The internal quantization remains the same as above, but you can see the input and output tensors are now integer format:

```
[ ]  interpreter = tf.lite.Interpreter(model_content=tflite_model_quant)
     input_type = interpreter.get_input_details()[0]['dtype']
     print('input: ', input_type)
     output_type = interpreter.get_output_details()[0]['dtype']
     print('output: ', output_type)

     input:  <class 'numpy.uint8'>
     output:  <class 'numpy.uint8'>
```

Save the models as files

- You'll need a *.tflite* file to deploy your model on other devices. So let's save the converted models to files and then load them when we run inferences below.

```
[ ]  import pathlib

     tflite_models_dir = pathlib.Path("/tmp/mnist_tflite_models/")
     tflite_models_dir.mkdir(exist_ok=True, parents=True)

     # Save the unquantized/float model:
     tflite_model_file = tflite_models_dir/"mnist_model.tflite"
     tflite_model_file.write_bytes(tflite_model)
     # Save the quantized model:
     tflite_model_quant_file = tflite_models_dir/"mnist_model_quant.tflite"
     tflite_model_quant_file.write_bytes(tflite_model_quant)
```

Evaluate the models on all images

- Now let's run both models using all the test images we loaded at the beginning :

```python
# Helper function to evaluate a TFLite model on all images
def evaluate_model(tflite_file, model_type):
    global test_images
    global test_labels

    test_image_indices = range(test_images.shape[0])
    predictions = run_tflite_model(tflite_file, test_image_indices)

    accuracy = (np.sum(test_labels== predictions) * 100) / len(test_images)

    print('%s model accuracy is %.4f%% (Number of test samples=%d)' % (
        model_type, accuracy, len(test_images)))
```

```
[16] evaluate_model(tflite_model_file, model_type="Float")

     Float model accuracy is 87.8900% (Number of test samples=10000)

[17] evaluate_model(tflite_model_quant_file, model_type="Quantized")

     Quantized model accuracy is 87.8300% (Number of test samples=10000)
```

Quantization aware training

- You will apply quantization aware training to the whole model and see this in the model summary. All layers are now prefixed by "quant".

```
import tensorflow_model_optimization as tfmot

quantize_model = tfmot.quantization.keras.quantize_model

# q_aware stands for for quantization aware.
q_aware_model = quantize_model(model)

# `quantize_model` requires a recompile.
q_aware_model.compile(optimizer='adam',
                      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=['accuracy'])

q_aware_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 quantize_layer (QuantizeLay  (None, 28, 28)           3
 er)

 quant_reshape (QuantizeWrap  (None, 28, 28, 1)        1
 perV2)

 quant_conv2d (QuantizeWrapp  (None, 26, 26, 12)       147
 erV2)

 quant_max_pooling2d (Quanti  (None, 13, 13, 12)       1
 zeWrapperV2)

 quant_flatten (QuantizeWrap  (None, 2028)             1
 perV2)

 quant_dense (QuantizeWrappe  (None, 10)               20295
 rV2)

=================================================================
Total params: 20,448
Trainable params: 20,410
Non-trainable params: 38
_________________________________________________________________
```

Quantization aware training

- For this example, there is minimal to no loss in test accuracy after quantization aware training, compared to the baseline.

```
_, baseline_model_accuracy = model.evaluate(
    test_images, test_labels, verbose=0)

_, q_aware_model_accuracy = q_aware_model.evaluate(
    test_images, test_labels, verbose=0)

print('Baseline test accuracy:', baseline_model_accuracy)
print('Quant test accuracy:', q_aware_model_accuracy)
```

```
Baseline test accuracy: 0.8942000269889832
Quant test accuracy: 0.8914999961853027
```

Quantization aware training

- Note that the resulting model is quantization aware but not quantized (e.g. the weights are float32 instead of int8).

```
converter = tf.lite.TFLiteConverter.from_keras_model(q_aware_model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

quantized_tflite_model = converter.convert()
```

- After this, you have an actually quantized model with int8 weights and uint8 activations.

Quantization aware training

- You create a float TFLite model and then see that the quantized TFLite model is 4x smaller.

```
# Create float TFLite model.
float_converter = tf.lite.TFLiteConverter.from_keras_model(model)
float_tflite_model = float_converter.convert()

# Measure sizes of models.
_, float_file = tempfile.mkstemp('.tflite')
_, quant_file = tempfile.mkstemp('.tflite')

with open(quant_file, 'wb') as f:
  f.write(quantized_tflite_model)

with open(float_file, 'wb') as f:
  f.write(float_tflite_model)

print("Float model in Mb:", os.path.getsize(float_file) / float(2**20))
print("Quantized model in Mb:", os.path.getsize(quant_file) / float(2**20))
```

```
INFO:tensorflow:Assets written to: /tmp/tmpv_evofke/assets
INFO:tensorflow:Assets written to: /tmp/tmpv_evofke/assets
WARNING:absl:Buffer deduplication procedure will be skipped when flatbuffer library is not properly loaded
Float model in Mb: 0.08062362670898438
Quantized model in Mb: 0.023468017578125
```

Assignment

- Requirement :
  1. Choose one CNN model from Lab1
  2. Do post-training integer quantization
  3. Do quantization aware training
  4. Analyzing two types of quantization results(i.e. size and accuracy) and make it into a report
- File format:
  - StudentID_Name_ post-training_integer_quantization.ipynb(15%)
  - StudentID_Name_ quantization_aware_training.ipynb(15%)
  - StudentID_Name_ Lab2.pdf(70%)

Deadline: 4/1(Fri.) 23:55

Reference

- [Post-training quantization]
- [Quantization aware training]