# Quantization and Pruning

## Imports

```
import tensorflow as tf
import numpy as np
import os
import tempfile
import zipfile
```

## Utilities and constants

```
# GLOBAL VARIABLES

# String constants for model filenames
FILE_WEIGHTS = 'baseline_weights.h5'
FILE_NON_QUANTIZED_H5 = 'non_quantized.h5'
FILE_NON_QUANTIZED_TFLITE = 'non_quantized.tflite'
FILE_PT_QUANTIZED = 'post_training_quantized.tflite'
FILE_QAT_QUANTIZED = 'quant_aware_quantized.tflite'
FILE_PRUNED_MODEL_H5 = 'pruned_model.h5'
FILE_PRUNED_QUANTIZED_TFLITE = 'pruned_quantized.tflite'
FILE_PRUNED_NON_QUANTIZED_TFLITE = 'pruned_non_quantized.tflite'

# Dictionaries to hold measurements
MODEL_SIZE = {}
ACCURACY = {}
```

```
# UTILITY FUNCTIONS

def print_metric(metric_dict, metric_name):
  '''Prints key and values stored in a dictionary'''
  for metric, value in metric_dict.items():
    print(f'{metric_name} for {metric}: {value}')


def model_builder():
  '''Returns a shallow CNN for training on the MNIST dataset'''

  keras = tf.keras

  # Define the model architecture.
  model = keras.Sequential([
    keras.layers.InputLayer(input_shape=(28, 28)),
    keras.layers.Reshape(target_shape=(28, 28, 1)),
    keras.layers.Conv2D(filters=12, kernel_size=(3, 3), activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(10, activation='softmax')
  ])

  return model


def evaluate_tflite_model(filename, x_test, y_test):
  '''
  Measures the accuracy of a given TF Lite model and test set

  Args:
    filename (string) - filename of the model to load
    x_test (numpy array) - test images
    y_test (numpy array) - test labels

  Returns
    float showing the accuracy against the test set
  '''

  # Initialize the TF Lite Interpreter and allocate tensors
  interpreter = tf.lite.Interpreter(model_path=filename)
  interpreter.allocate_tensors()

  # Get input and output index
  input_index = interpreter.get_input_details()[0]["index"]
  output_index = interpreter.get_output_details()[0]["index"]

  # Initialize empty predictions list
  prediction_digits = []

  # Run predictions on every image in the "test" dataset.
  for i, test_image in enumerate(x_test):
```

```
    # Pre-processing: add batch dimension and convert to float32 to match with
    # the model's input data format.
    test_image = np.expand_dims(test_image, axis=0).astype(np.float32)
    interpreter.set_tensor(input_index, test_image)

    # Run inference.
    interpreter.invoke()

    # Post-processing: remove batch dimension and find the digit with highest
    # probability.
    output = interpreter.tensor(output_index)
    digit = np.argmax(output()[0])
    prediction_digits.append(digit)

  # Compare prediction results with ground truth labels to calculate accuracy.
  prediction_digits = np.array(prediction_digits)
  accuracy = (prediction_digits == y_test).mean()

  return accuracy


def get_gzipped_model_size(file):
  '''Returns size of gzipped model, in bytes.'''
  _, zipped_file = tempfile.mkstemp('.zip')
  with zipfile.ZipFile(zipped_file, 'w', compression=zipfile.ZIP_DEFLATED) as f:
    f.write(file)

  return os.path.getsize(zipped_file)
```

## Download and Prepare the Dataset

```
# Load MNIST dataset
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the input image so that each pixel value is between 0 to 1.
train_images = train_images / 255.0
test_images = test_images / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

## Baseline Model

```
# Create the baseline model
baseline_model = model_builder()

# Save the initial weights for use later
baseline_model.save_weights(FILE_WEIGHTS)

# Print the model summary
baseline_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape (Reshape)           (None, 28, 28, 1)         0

 conv2d (Conv2D)             (None, 26, 26, 12)        120

 max_pooling2d (MaxPooling2D  (None, 13, 13, 12)       0
 )

 flatten (Flatten)           (None, 2028)              0

 dense (Dense)               (None, 10)                20290

=================================================================
Total params: 20,410
Trainable params: 20,410
Non-trainable params: 0
_________________________________________________________________
```

```
# Setup the model for training
baseline_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
baseline_model.fit(train_images, train_labels, epochs=1, shuffle=False)
```

```
1875/1875 [==============================] - 30s 16ms/step - loss: 0.2972 - accuracy: 0.9167
<keras.callbacks.History at 0x7f8c9b262a30>
```

```
# Get the baseline accuracy
_, ACCURACY['baseline Keras model'] = baseline_model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.1469 - accuracy: 0.9584
```

```
# Save the Keras model
baseline_model.save(FILE_NON_QUANTIZED_H5, include_optimizer=False)

# Save and get the model size
MODEL_SIZE['baseline h5'] = os.path.getsize(FILE_NON_QUANTIZED_H5)

# Print records so far
print_metric(ACCURACY, "test accuracy")
print_metric(MODEL_SIZE, "model size in bytes")
```

```
test accuracy for baseline Keras model: 0.9584000110626221
model size in bytes for baseline h5: 98968
```

## Convert the model to TF Lite format

```
def convert_tflite(model, filename, quantize=False):
  '''
  Converts the model to TF Lite format and writes to a file

  Args:
    model (Keras model) - model to convert to TF Lite
    filename (string) - string to use when saving the file
    quantize (bool) - flag to indicate quantization

  Returns:
    None
  '''

  # Initialize the converter
  converter = tf.lite.TFLiteConverter.from_keras_model(model)

  # Set for quantization if flag is set to True
  if quantize:
    converter.optimizations = [tf.lite.Optimize.DEFAULT]

  # Convert the model
  tflite_model = converter.convert()

  # Save the model.
  with open(filename, 'wb') as f:
    f.write(tflite_model)
```

This is not yet quantized

```
# Convert baseline model
convert_tflite(baseline_model, FILE_NON_QUANTIZED_TFLITE)
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _update_step_xla while saving (showing 2 of 2). These functions will n
```

Slight decrease in model size when converting to `.tflite` format.

```
MODEL_SIZE['non quantized tflite'] = os.path.getsize(FILE_NON_QUANTIZED_TFLITE)

print_metric(MODEL_SIZE, 'model size in bytes')
```

```
model size in bytes for baseline h5: 98968
model size in bytes for non quantized tflite: 85012
```

*If there is a `Runtime Error: There is at least 1 reference to internal data in the interpreter in the form of a numpy array or slice.`, try re-running the cell.*

```
ACCURACY['non quantized tflite'] = evaluate_tflite_model(FILE_NON_QUANTIZED_TFLITE, test_images, test_labels)
```

```
print_metric(ACCURACY, 'test accuracy')
```

```
test accuracy for baseline Keras model: 0.9584000110626221
test accuracy for non quantized tflite: 0.9584
```

## Post-Training Quantization

Convert 32 bit representations (float) into 8 bits (integer) to reduce model size and achieve faster computation

```
# Convert and quantize the baseline model
convert_tflite(baseline_model, FILE_PT_QUANTIZED, quantize=True)
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _update_step_xla while saving (showing 2 of 2). These functions will n
```

```
# Get the model size
MODEL_SIZE['post training quantized tflite'] = os.path.getsize(FILE_PT_QUANTIZED)

print_metric(MODEL_SIZE, 'model size')
```

```
model size for baseline h5: 98968
model size for non quantized tflite: 85012
model size for post training quantized tflite: 24256
```

About 4X reduction in model size in the quantized version

```
ACCURACY['post training quantized tflite'] = evaluate_tflite_model(FILE_PT_QUANTIZED, test_images, test_labels)
```

```
print_metric(ACCURACY, 'test accuracy')
```

```
test accuracy for baseline Keras model: 0.9584000110626221
test accuracy for non quantized tflite: 0.9584
test accuracy for post training quantized tflite: 0.9582
```

## Quantization Aware Training

Doing quantization aware training before quantizing the model in order to preserve more accuracy. It simulates the loss of precision by inserting fake quant nodes in the model during training, so the model will learn to adapt with the loss of precision to get more accurate predictions.

```
# Install the toolkit
!pip install tensorflow_model_optimization
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow_model_optimization
  Downloading tensorflow_model_optimization-0.7.3-py2.py3-none-any.whl (238 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 238.9/238.9 KB 9.6 MB/s eta 0:00:00
Requirement already satisfied: numpy~=1.14 in /usr/local/lib/python3.9/dist-packages (from tensorflow_model_optimization) (1.22.4)
Requirement already satisfied: six~=1.10 in /usr/local/lib/python3.9/dist-packages (from tensorflow_model_optimization) (1.15.0)
Requirement already satisfied: dm-tree~=0.1.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow_model_optimization) (0.1.8)
Installing collected packages: tensorflow_model_optimization
Successfully installed tensorflow_model_optimization-0.7.3
```

If we need to pass in a model that is already trained, need to recompile before continue training.

```
import tensorflow_model_optimization as tfmot

# method to quantize a Keras model
quantize_model = tfmot.quantization.keras.quantize_model

# Define the model architecture. Still the baseline model
model_to_quantize = model_builder()

# Reinitialize weights with saved file
model_to_quantize.load_weights(FILE_WEIGHTS)

# Quantize the model
q_aware_model = quantize_model(model_to_quantize)

# `quantize_model` requires a recompile.
q_aware_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

q_aware_model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.9/dist-packages/tensorflow/python/autograph/pyct/static_analysis/liveness.py:83: Analyzer.lamba_che
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at least in the same block. https://github.com/tensorf
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 quantize_layer (QuantizeLay  (None, 28, 28)           3
 er)

 quant_reshape_1 (QuantizeWr  (None, 28, 28, 1)        1
 apperV2)

 quant_conv2d_1 (QuantizeWra  (None, 26, 26, 12)       147
 pperV2)
```

```
 quant_max_pooling2d_1 (Quan  (None, 13, 13, 12)       1
 tizeWrapperV2)

 quant_flatten_1 (QuantizeWr  (None, 2028)             1
 apperV2)

 quant_dense_1 (QuantizeWrap  (None, 10)               20295
 perV2)

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

The number of model params increased because of the nodes added by the `quantize_model()` method.

```
# Train the model
q_aware_model.fit(train_images, train_labels, epochs=1, shuffle=False)
```

```
1875/1875 [==============================] - 42s 22ms/step - loss: 0.2997 - accuracy: 0.9158
<keras.callbacks.History at 0x7f8c9b5f3ee0>
```

```
# Reinitialize the dictionary
ACCURACY = {}

# Get the accuracy of the quantization aware trained model (not yet quantized)
_, ACCURACY['quantization aware non-quantized'] = q_aware_model.evaluate(test_images, test_labels, verbose=0)
print_metric(ACCURACY, 'test accuracy')
```

```
test accuracy for quantization aware non-quantized: 0.95660001039505
```

```
# Convert and quantize the model.
convert_tflite(q_aware_model, FILE_QAT_QUANTIZED, quantize=True)

# Get the accuracy of the quantized model
ACCURACY['quantization aware quantized'] = evaluate_tflite_model(FILE_QAT_QUANTIZED, test_images, test_labels)
print_metric(ACCURACY, 'test accuracy')
```

```
WARNING:absl:Found untraced functions such as _update_step_xla, reshape_1_layer_call_fn, reshape_1_layer_call_and_return_conditional_losses, conv2
/usr/local/lib/python3.9/dist-packages/tensorflow/lite/python/convert.py:765: UserWarning: Statistics for quantized inputs were expected, but not
  warnings.warn("Statistics for quantized inputs were expected, but not "
test accuracy for quantization aware non-quantized: 0.95660001039505
test accuracy for quantization aware quantized: 0.9566
```

## Pruning

Zero out insignificant low magnitude weights. Making the weights sparse helps in compressing the model.

Pass in the baseline model trained earlier. The number of model params increased because of the wrapper layers added by the pruning method.

```
# Get the pruning method
prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

# Compute end step to finish pruning after 2 epochs.
batch_size = 128
epochs = 2
validation_split = 0.1 # 10% of training set will be used for validation set.

num_images = train_images.shape[0] * (1 - validation_split)
end_step = np.ceil(num_images / batch_size).astype(np.int32) * epochs

# Define pruning schedule.
pruning_params = {
      'pruning_schedule': tfmot.sparsity.keras.PolynomialDecay(initial_sparsity=0.50,
                                                               final_sparsity=0.80,
                                                               begin_step=0,
                                                               end_step=end_step)
}

# Pass in the trained baseline model
model_for_pruning = prune_low_magnitude(baseline_model, **pruning_params)

# `prune_low_magnitude` requires a recompile.
model_for_pruning.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model_for_pruning.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 prune_low_magnitude_reshape  (None, 28, 28, 1)        1
  (PruneLowMagnitude)

 prune_low_magnitude_conv2d   (None, 26, 26, 12)       230
 (PruneLowMagnitude)

 prune_low_magnitude_max_poo  (None, 13, 13, 12)       1
 ling2d (PruneLowMagnitude)

 prune_low_magnitude_flatten  (None, 2028)             1
  (PruneLowMagnitude)

 prune_low_magnitude_dense (  (None, 10)               40572
 PruneLowMagnitude)

=================================================================
Total params: 40,805
Trainable params: 20,410
Non-trainable params: 20,395
_________________________________________________________________
```

Peek at the weights of one layer before pruning. After pruning, many of these will be zeroed out.

```
# Preview model weights
model_for_pruning.weights[1]
```

```
<tf.Variable 'conv2d/kernel:0' shape=(3, 3, 1, 12) dtype=float32, numpy=
array([[[[ 2.47235626e-01,  5.76701641e-01,  2.76040822e-01,
           2.62139052e-01, -9.30636972e-02, -7.03000367e-01,
           2.99865752e-02,  6.39910251e-02,  9.76387486e-02,
           3.87996733e-01,  1.93972990e-01,  1.74673781e-01]],

         [[ 3.47311616e-01,  1.14257313e-01,  3.11344743e-01,
           -5.71893789e-02,  2.46381015e-01, -2.44362615e-02,
           4.20328856e-01,  1.63339734e-01, -3.54331639e-03,
           3.63262981e-01, -4.57135076e-03,  3.47524248e-02]],

         [[ 2.74401277e-01, -3.29056650e-01,  1.31493434e-01,
           1.03725806e-01,  2.75407702e-01,  4.61605430e-01,
           4.39918816e-01, -2.28615195e-01,  4.99208719e-02,
           4.23490375e-01,  3.27305704e-01,  2.61756539e-01]]],


        [[[-1.01836033e-01,  1.59417003e-01,  2.63394415e-01,
           -3.29185352e-02, -6.02039583e-02, -6.28425002e-01,
           1.58554018e-01,  8.28374252e-02,  1.13733545e-01,
           6.51831564e-04,  6.28367215e-02,  2.47349605e-01]],

         [[ 5.00846468e-02,  9.12962854e-03, -7.20648840e-02,
           2.82972395e-01,  1.45532548e-01,  4.64076996e-02,
           3.52211624e-01,  8.61134380e-02,  6.99764192e-02,
           2.56031990e-01,  1.11236624e-01,  1.81771517e-01]],

         [[ 2.96588391e-01, -5.84539890e-01,  1.99352771e-01,
           1.58643275e-01,  1.32288009e-01,  5.00420153e-01,
           2.15587333e-01,  4.39853072e-02,  3.63908261e-01,
           3.20478864e-02,  3.41913760e-01, -1.06462672e-01]]],


        [[[-3.41926754e-01, -2.43972480e-01, -7.93728828e-02,
           -3.64903286e-02,  2.08510309e-01, -3.35873514e-01,
           -6.44597471e-01,  2.44223505e-01, -1.37151316e-01,
           -5.31413734e-01, -5.46651959e-01,  6.22910298e-02]],

         [[-2.16374658e-02, -7.21738279e-01, -1.07215911e-01,
           -1.00222647e-01,  7.22561404e-02,  2.43830845e-01,
           -8.15873981e-01,  3.35721135e-01, -1.46215498e-01,
           -7.23250091e-01, -4.82220113e-01, -1.93895221e-01]],

         [[ 1.98041335e-01, -2.80745715e-01,  1.21470988e-01,
            2.58320123e-01,  1.93520635e-01,  3.85316700e-01,
           -4.50816154e-01,  5.45709729e-02,  2.51890153e-01,
           -6.77445531e-01,  1.72831848e-01, -5.32421321e-02]]]],
       dtype=float32)>
```

Start re-training

```
# Callback to update pruning wrappers at each step
callbacks = [
  tfmot.sparsity.keras.UpdatePruningStep(),
]

# Train and prune the model
model_for_pruning.fit(train_images, train_labels,
                  epochs=epochs, validation_split=validation_split,
                  callbacks=callbacks)
```

```
Epoch 1/2
1688/1688 [==============================] - 23s 12ms/step - loss: 0.1610 - accuracy: 0.9579 - val_loss: 0.1054 - val_accuracy: 0.9715
Epoch 2/2
```

```
1688/1688 [==============================] - 19s 11ms/step - loss: 0.1191 - accuracy: 0.9661 - val_loss: 0.0930 - val_accuracy: 0.9740
<keras.callbacks.History at 0x7f8c9a8049a0>
```

Weights in the same layer after pruning

```
# Preview model weights
model_for_pruning.weights[1]
```

```
<tf.Variable 'conv2d/kernel:0' shape=(3, 3, 1, 12) dtype=float32, numpy=
array([[[[-0.        ,  0.9986974 ,  0.        ,  0.        ,
           0.        , -1.0993321 ,  0.        ,  0.        ,
           0.        ,  0.        , -0.        ,  0.        ]],

        [[ 0.8603713 ,  0.        ,  0.        ,  0.        ,
           0.        , -0.        ,  0.9159232 , -0.        ,
           0.        ,  0.89622444, -0.        ,  0.        ]],

        [[-0.        , -0.        ,  0.        ,  0.        ,
           0.        ,  0.82024074,  0.80412585, -0.        ,
           0.        ,  0.7941427 ,  0.63745815,  0.        ]]],


       [[[ 0.        ,  0.        ,  0.        ,  0.        ,
           0.        , -0.81527257,  0.        , -0.        ,
           0.        ,  0.        , -0.        ,  0.        ]],

        [[-0.        ,  0.        ,  0.        ,  0.        ,
           0.        , -0.        , -0.        , -0.        ,
           0.        ,  0.        , -0.        ,  0.        ]],

        [[-0.        , -1.2665261 ,  0.        ,  0.        ,
           0.        ,  0.9397773 , -0.        , -0.        ,
           1.0304172 ,  0.        ,  0.6701804 ,  0.        ]]],


       [[[-0.        ,  0.        ,  0.        ,  0.        ,
           0.        , -0.        , -0.8565529 , -0.        ,
          -0.        , -0.5979682 , -1.095686  ,  0.        ]],

        [[-0.        , -1.2964823 ,  0.        ,  0.        ,
           0.        ,  0.        , -1.239559  ,  0.94420266,
          -0.        , -0.81302065, -0.        ,  0.        ]],

        [[-0.        ,  0.        ,  0.        ,  0.        ,
           0.        ,  0.        , -0.        , -0.        ,
          -0.        , -1.2485639 , -0.        ,  0.        ]]]],
      dtype=float32)>
```

After pruning, remove the wrapper layers to have the same layers and params as the baseline model.

```
# Remove pruning wrappers
model_for_export = tfmot.sparsity.keras.strip_pruning(model_for_pruning)
model_for_export.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape (Reshape)           (None, 28, 28, 1)         0

 conv2d (Conv2D)             (None, 26, 26, 12)        120

 max_pooling2d (MaxPooling2D  (None, 13, 13, 12)       0
 )

 flatten (Flatten)           (None, 2028)              0

 dense (Dense)               (None, 10)                20290

=================================================================
Total params: 20,410
Trainable params: 20,410
Non-trainable params: 0
_________________________________________________________________
```

For the same model weights, the index is now different, because the wrappers were removed.

```
# Preview model weights (index 1 earlier is now 0 because pruning wrappers were removed)
model_for_export.weights[0]
```

```
<tf.Variable 'conv2d/kernel:0' shape=(3, 3, 1, 12) dtype=float32, numpy=
array([[[[-0.        ,  0.9986974 ,  0.        ,  0.        ,
           0.        , -1.0993321 ,  0.        ,  0.        ,
           0.        ,  0.        , -0.        ,  0.        ]],

        [[ 0.8603713 ,  0.        ,  0.        ,  0.        ,
           0.        , -0.        ,  0.9159232 , -0.        ,
           0.        ,  0.89622444, -0.        ,  0.        ]],
```

```
          [[-0.        , -0.        ,  0.        ,  0.        ,
             0.        ,  0.82024074,  0.80412585, -0.        ,
             0.        ,  0.7941427 ,  0.63745815,  0.        ]]],


        [[[ 0.        ,  0.        ,  0.        ,  0.        ,
             0.        , -0.81527257,  0.        , -0.        ,
             0.        ,  0.        , -0.        ,  0.        ]],

          [[-0.        ,  0.        ,  0.        ,  0.        ,
             0.        , -0.        , -0.        , -0.        ,
             0.        ,  0.        , -0.        ,  0.        ]],

          [[-0.        , -1.2665261 ,  0.        ,  0.        ,
             0.        ,  0.9397773 , -0.        , -0.        ,
             1.0304172 ,  0.        ,  0.6701804 ,  0.        ]]],


        [[[-0.        ,  0.        ,  0.        ,  0.        ,
             0.        , -0.        , -0.8565529 , -0.        ,
            -0.        , -0.5979682 , -1.095686  ,  0.        ]],

          [[-0.        , -1.2964823 ,  0.        ,  0.        ,
             0.        ,  0.        , -1.239559  ,  0.94420266,
            -0.        , -0.81302065, -0.        ,  0.        ]],

          [[-0.        ,  0.        ,  0.        ,  0.        ,
             0.        ,  0.        , -0.        , -0.        ,
            -0.        , -1.2485639 , -0.        ,  0.        ]]]],
        dtype=float32)>
```

The pruned model has the same file size as the baseline_model when saved as H5, which is to be expected. The improvement will be noticeable after compressing.

```
# Save Keras model
model_for_export.save(FILE_PRUNED_MODEL_H5, include_optimizer=False)

# Get uncompressed model size of baseline and pruned models
MODEL_SIZE = {}
MODEL_SIZE['baseline h5'] = os.path.getsize(FILE_NON_QUANTIZED_H5)
MODEL_SIZE['pruned non quantized h5'] = os.path.getsize(FILE_PRUNED_MODEL_H5)

print_metric(MODEL_SIZE, 'model_size in bytes')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train
model_size in bytes for baseline h5: 98968
model_size in bytes for pruned non quantized h5: 98968
```

The pruned model is about 3 times smaller, because the zeros can be compressed much more efficiently than the low magnitude weights before pruning.

```
# Get compressed size of baseline and pruned models
MODEL_SIZE = {}
MODEL_SIZE['baseline h5'] = get_gzipped_model_size(FILE_NON_QUANTIZED_H5)
MODEL_SIZE['pruned non quantized h5'] = get_gzipped_model_size(FILE_PRUNED_MODEL_H5)

print_metric(MODEL_SIZE, "gzipped model size in bytes")
```

```
gzipped model size in bytes for baseline h5: 78058
gzipped model size in bytes for pruned non quantized h5: 25799
```

Can make the model even more lightweight by quantizing the pruned model. About 10X reduction in compressed model size as compared to the baseline.

```
# Convert and quantize the pruned model.
pruned_quantized_tflite = convert_tflite(model_for_export, FILE_PRUNED_QUANTIZED_TFLITE, quantize=True)

# Compress and get the model size
MODEL_SIZE['pruned quantized tflite'] = get_gzipped_model_size(FILE_PRUNED_QUANTIZED_TFLITE)
print_metric(MODEL_SIZE, "gzipped model size in bytes")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op while saving (showing 1 of 1). These functions will not be directly cal
gzipped model size in bytes for baseline h5: 78058
gzipped model size in bytes for pruned non quantized h5: 25799
gzipped model size in bytes for pruned quantized tflite: 8247
```

Accuracy remains good

```
# Get accuracy of pruned Keras and TF Lite models
ACCURACY = {}

_, ACCURACY['pruned model h5'] = model_for_pruning.evaluate(test_images, test_labels)
```

```
ACCURACY['pruned and quantized tflite'] = evaluate_tflite_model(FILE_PRUNED_QUANTIZED_TFLITE, test_images, test_labels)

print_metric(ACCURACY, 'accuracy')
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.1059 - accuracy: 0.9686
accuracy for pruned model h5: 0.9685999751091003
accuracy for pruned and quantized tflite: 0.9683
```

✓ 4 秒　完成时间：02:10