TensorFlow Lite 8-bit quantization specification

Specification summary

We are providing a specification, and we can only provide some guarantees on behaviour if the spec is followed. We also understand different hardware may have preferences and restrictions that may cause slight deviations when implementing the spec that result in implementations that are not bit-exact. Whereas that may be acceptable in most cases (and we will provide a suite of tests that to the best of our knowledge include per-operation tolerances that we gathered from several models), the nature of machine learning (and deep learning in the most common case) makes it impossible to provide any hard guarantees.

8-bit quantization approximates floating point values using the following formula.

$$real\_value = (int8\_value - zero\_point) \times scale$$

Per-axis (aka per-channel in Conv ops) or per-tensor weights are represented by `int8` two's complement values in the range `[-127, 127]` with zero-point equal to 0. Per-tensor activations/inputs are represented by `int8` two's complement values in the range `[-128, 127]`, with a zero-point in range `[-128, 127]`.

There are other exceptions for particular operations that are documented below.

> **Note:** In the past our quantized tooling used per-tensor, asymmetric, `uint8` quantization. New tooling, reference kernels, and optimized kernels for 8-bit quantization will use this spec.

Signed integer vs unsigned integer

TensorFlow Lite quantization will primarily prioritize tooling and kernels for `int8` quantization for 8-bit. This is for the convenience of symmetric quantization being represented by zero-point equal to 0. Additionally many backends have additional optimizations for `int8xint8` accumulation.

Per-axis vs per-tensor

Per-tensor quantization means that there will be one scale and/or zero-point per entire tensor. Per-axis quantization means that there will be one scale and/or `zero_point` per slice in the `quantized_dimension`. The quantized dimension specifies the dimension of the Tensor's shape that the scales and zero-points correspond to. For example, a tensor `t`, with `dims=[4, 3, 2, 1]` with quantization params: `scale=[1.0, 2.0, 3.0]`, `zero_point=[1, 2, 3]`,`quantization_dimension=1` will be quantized across the second dimension of t:

```
t[:, 0, :, :] will have scale[0]=1.0, zero_point[0]=1
t[:, 1, :, :] will have scale[1]=2.0, zero_point[1]=2
t[:, 2, :, :] will have scale[2]=3.0, zero_point[2]=3
```

Often, the quantized_dimension is the output_channel of the weights of convolutions, but in theory it can be the dimension that corresponds to each dot-product in the kernel implementation, allowing more quantization granularity without performance implications. This has large improvements to accuracy.

TFLite has per-axis support for a growing number of operations. At the time of this document support exists for Conv2d and DepthwiseConv2d.

Symmetric vs asymmetric

Activations are asymmetric: they can have their zero-point anywhere within the signed `int8` range `[-128, 127]`. Many activations are asymmetric in nature and a zero-point is an relatively inexpensive way to effectively get up to an extra binary bit of precision. Since activations are only multiplied by constant weights, the constant zero-point value can be optimized pretty heavily.

Weights are symmetric: forced to have zero-point equal to 0. Weight values are multiplied by dynamic input and activation values. This means that there is a unavoidable runtime cost of multiplying the zero-point of the weight with the activation value. By enforcing that zero-point is 0 we can avoid this cost.

Explanation of the math:

A is a $m \times n$ matrix of quantized activations.
B is a $n \times p$ matrix of quantized weights.
Consider multiplying the jth row of A, a_j by the kth column of B, b_k, both of length n. The quantized integer values and zero-points values are q_a, z_a and q_b, z_b respectively.

$$a_j \cdot b_k = \sum_{i=0}^{n} a_j^{(i)} b_k^{(i)} = \sum_{i=0}^{n} (q_a^{(i)} - z_a)(q_b^{(i)} - z_b) = \sum_{i=0}^{n} q_a^{(i)} q_b^{(i)} - \sum_{i=0}^{n} q_a^{(i)} z_b - \sum_{i=0}^{n} q_b^{(i)} z_a + \sum$$

The $\sum_{i=0}^{n} q_a^{(i)} q_b^{(i)}$ term is unavoidable since it's performing the dot product of the input value and the weight value.

The

$\sum_{i=0}^{n} q_b^{(i)} z_a$
and
$\sum_{i=0}^{n} z_a z_b$
terms are made up of constants that remain the same per inference invocation, and thus can be pre-calculated.

The $\sum_{i=0}^{n} q_a^{(i)} z_b$ term needs to be computed every inference since the activation changes every inference. By enforcing weights to be symmetric we can remove the cost of this term.

int8 quantized operator specifications

Below we describe the quantization requirements for our int8 tflite kernels:

```
ADD
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

AVERAGE_POOL_2D
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
```

```
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

CONCATENATION
  Input ...:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

CONV_2D
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1 (Weight):
    data_type  : int8
    range      : [-127, 127]
    granularity: per-axis (dim = 0)
    restriction: zero_point = 0
  Input 2 (Bias):
    data_type  : int32
    range      : [int32_min, int32_max]
    granularity: per-axis
    restriction: (scale, zero_point) = (input0_scale * input1_scale[...], 0)
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

DEPTHWISE_CONV_2D
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1 (Weight):
    data_type  : int8
    range      : [-127, 127]
    granularity: per-axis (dim = 3)
    restriction: zero_point = 0
  Input 2 (Bias):
    data_type  : int32
    range      : [int32_min, int32_max]
    granularity: per-axis
    restriction: (scale, zero_point) = (input0_scale * input1_scale[...], 0)
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

FULLY_CONNECTED
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1 (Weight):
    data_type  : int8
    range      : [-127, 127]
    granularity: per-tensor
    restriction: zero_point = 0
  Input 2 (Bias):
    data_type  : int32
    range      : [int32_min, int32_max]
    granularity: per-tensor
    restriction: (scale, zero_point) = (input0_scale * input1_scale[...], 0)
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

L2_NORMALIZATION
```

```
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
    restriction: (scale, zero_point) = (1.0 / 128.0, 0)

LOGISTIC
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
    restriction: (scale, zero_point) = (1.0 / 256.0, -128)

MAX_POOL_2D
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

MUL
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

RESHAPE
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

RESIZE_BILINEAR
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

SOFTMAX
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
    restriction: (scale, zero_point) = (1.0 / 256.0, -128)
```

```
SPACE_TO_DEPTH
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

TANH
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
    restriction: (scale, zero_point) = (1.0 / 128.0, 0)

PAD
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

GATHER
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

BATCH_TO_SPACE_ND
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

SPACE_TO_BATCH_ND
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

TRANSPOSE
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point
```

```
MEAN
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

SUB
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

SUM
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

SQUEEZE
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

LOG_SOFTMAX
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
    restriction: (scale, zero_point) = (16.0 / 256.0, 127)

MAXIMUM
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

ARG_MAX
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

MINIMUM
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
```

```
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

LESS
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

PADV2
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

GREATER
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

GREATER_EQUAL
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

LESS_EQUAL
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

SLICE
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  restriction: Input and outputs must all have same scale/zero_point

EQUAL
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
```

```
NOT_EQUAL
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Input 1:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

SHAPE
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor

QUANTIZE (Requantization)
  Input 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
  Output 0:
    data_type  : int8
    range      : [-128, 127]
    granularity: per-tensor
```