**TensorFlow** Blog          Q Search the Blog                    ← **Return to TensorFlow**

| A | TensorFlo w Core | Tensor Flow.js | TensorFl ow Lite | T F X | S w i | Com munit y | | | |

## TensorFlow Lite

# Quantization Aware Training with TensorFlow Model Optimization Toolkit - Performance with Accuracy

April 08, 2020

*Posted by the TensorFlow Model Optimization team*

We are excited to release the Quantization Aware Training (QAT) API as part of the [TensorFlow Model Optimization Toolkit](). QAT enables you to train and deploy models with the performance and size benefits of quantization, while retaining close to their original accuracy. This work is part of our [roadmap]() to support the development of smaller and faster ML models. For more background, you can see previous posts on [post-training quantization](), [float16 quantization]() and [sparsity]().

execution performance and efficiency. For example, [TensorFlow Lite](#) 8-bit integer quantization results in models that are up to 4x smaller in size, 1.5x-4x faster in computations, and lower power consumption on CPUs. Additionally, it allows model execution on specialized neural accelerators, such as Edge TPU in [Coral](#), which often has a restricted set of data types.

However, the process of going from higher to lower precision is lossy in nature. As seen in the image below, quantization squeezes a small range of floating-point values into a fixed number of information buckets.
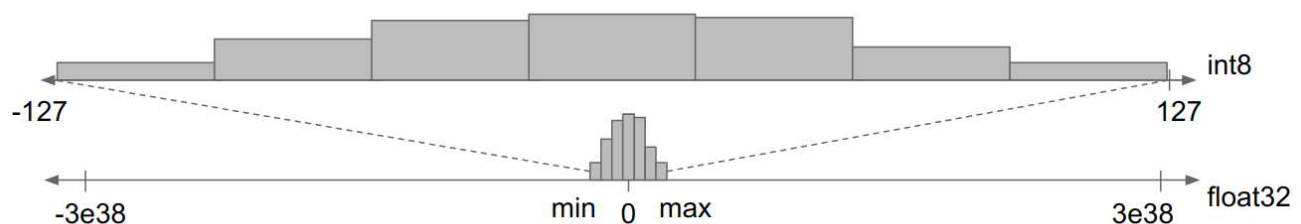


Small range of float32 values mapped to int8 is a lossy conversion since int8 only has 255 information channels

This leads to information loss. The parameters (or weights) of a model can now only take a small set of values and the minute differences between them are lost. For example, all values in range [2.0, 2.3] may now be represented in one single bucket. This is similar to rounding errors when fractional values are represented as integers.

There are also other sources of loss. When these lossy numbers are used in several multiply-add computations, these losses accumulate. Further, int8 values, which accumulate into int32 integers, need to be rescaled back to int8 values for the next computation, thus introducing more computational error.

# Quantization Aware Training

The core idea is that QAT simulates low-precision inference-time computation in the forward pass of the training process. This [work](#) is credited to the original innovations by Skirmantas

If training is not an option, please check out post-training quantization, which works as part of TensorFlow Lite model conversion. QAT is also useful for researchers and hardware designers who may want to experiment with various quantization strategies (beyond what is supported by TensorFlow Lite) and / or simulate how quantization affects accuracy for different hardware backends.

# QAT-trained models have comparable accuracy to floating-point

| Model | Floating-point baseline model | QAT model | Delta | Post-training full integer quantized model |
| --- | --- | --- | --- | --- |
| MobileNet v1 1.0 224 | 71.03% | 71.06% | 0.04% | 69.57% |
| MobileNet v2 1.0 224 | 70.77% | 70.01% | -1.07% | 70.2% |
| ResNet v1 50 | 76.3% | 76.1% | -0.26% | 75.95% |

In the table above, QAT accuracy numbers were trained with the default TensorFlow Lite configuration and contrasted with the floating-point baseline and post-training quantized models.

## Emulating low-precision computation

The training graph itself operates in floating-point (e.g. float32), but it has to emulate low-precision computation, which is fixed-point (e.g. int8 in the case of TensorFlow Lite). To do so, we insert special operations into the graph (tensorflow::ops::FakeQuantWithMinMaxVars) that convert the floating-point tensors into low-precision values and then convert the low-precision values back into floating-point. This ensures that losses from quantization are

☰  **TensorFlow** Blog    🔍 Search the Blog              ← **Return to TensorFlow** ▌

| A | TensorFlo | Tensor | TensorFl | T | S | Com | | |
| I | w Core | Flow.js | ow Lite | F | w | munit | | |
| I | | | | X | i | | | |

computation with similarly mapped tensors won't introduce any further loss and mimics low-precision computations exactly.

## Placing the quantization emulation operations

The quantization emulation operations need to be placed in the training graph such that they are consistent with the way that the quantized graph will be computed. This means that, for our API to be able to execute in TensorFlow Lite, we needed to follow the TensorFlow Lite quantization spec precisely.
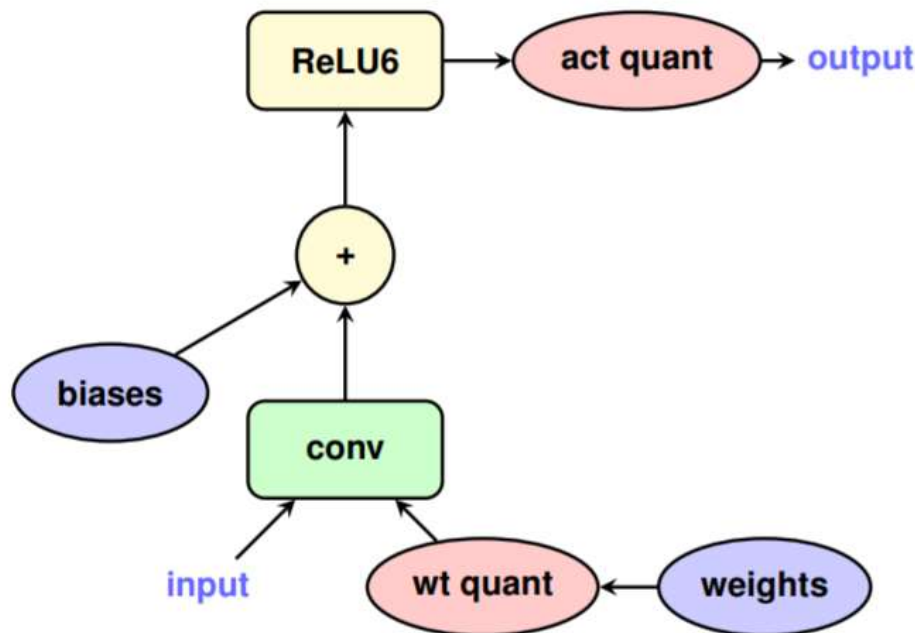


The 'wt quant' and 'act quant' ops introduce losses in the forward pass of the model to simulate actual quantization loss during inference. Note how there is no Quant operation between Conv and ReLU6. This is because ReLUs get fused in TensorFlow Lite.

The API, built upon the Keras layers and model abstractions, hides the complexities mentioned above, so that you can quantize your entire model with a few lines of code.

## Logging computation statistics

# How to use the API with only few lines of code

The QAT API provides a simple and highly flexible way to quantize your TensorFlow Keras model. It makes it really easy to train with "quantization awareness" for an entire model or only parts of it, then export it for deployment withTensorFlow Lite.

## Quantize the entire Keras model

```python
import tensorflow_model_optimization as tfmot

model = tf.keras.Sequential([
    ...
])
# Quantize the entire model.
quantized_model = tfmot.quantization.keras.quantize_model(model)

# Continue with training as usual.
quantized_model.compile(...)
quantized_model.fit(...)
```

## Quantize part(s) of a Keras model

```python
import tensorflow_model_optimization as tfmot
quantize_annotate_layer = tfmot.quantization.keras.quantize_annota

model = tf.keras.Sequential([
```

```
  quantize_annotate_layer(ReLU()),
  Dense(),
  ...
])

# Quantize the model.
quantized_model = tfmot.quantization.keras.quantize_apply(model)
```
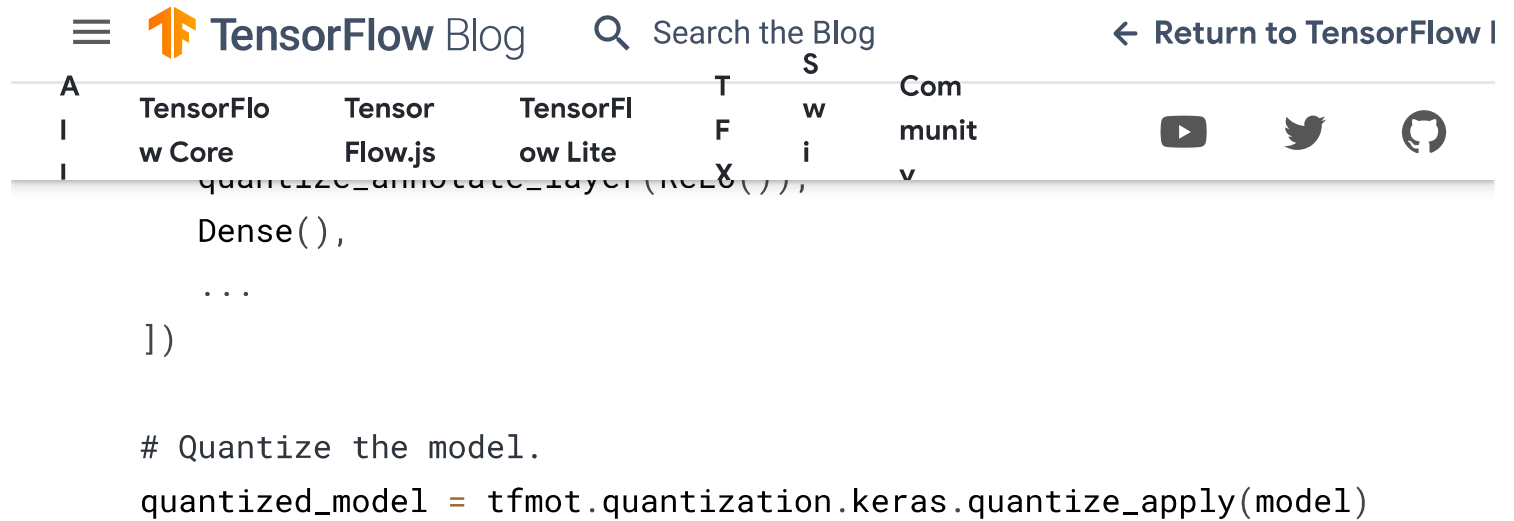
By default, our API is configured to work with the quantized execution support available in TensorFlow Lite. A detailed Colab with an end-to-end training example is located here.

The API is quite flexible and capable of handling far more complicated use cases. For example, it allows you to control quantization precisely within a layer, create custom quantization algorithms, and handle any custom layers that you may have written.

To learn more about how to use the API, please try this Colab. These sections of the Colab provide examples of how users can experiment with different quantization algorithms using the API. You can also check out this recent talk from the TensorFlow Developer Summit.

We are very excited to see how the QAT API further enables TensorFlow users to push the boundaries of efficient execution in their TensorFlow Lite-powered products as well as how it opens the door to researching new quantization algorithms and further developing new hardware platforms with different levels of precision.

If you want to learn more, check out this video from the TensorFlow DevSummit which introduces the Model Optimization Toolkit and explains QAT.

▶

## Acknowledgements

*Thanks to Pulkit Bhuwalka, Alan Chiao, Suharsh Sivakumar, Raziel Alvarez, Feng Liu, Lawrence Chan, Skirmantas Kligys, Yunlu Li, Khanh LeViet, Billy Lambert, Mark Daoust, Tim Davis, Sarah Sirajuddin, and François Chollet*

◆

TensorFlow Lite

# Next post

TensorFlow.js

**TensorFlow** Blog

Q Search the Blog

← **Return to TensorFlow**

A
I
I

**TensorFlo
w Core**

**Tensor
Flow.js**

**TensorFl
ow Lite**

T
F
X

S
w
i

**Com
munit
y**

2.0

April 07, 2020 — Posted by Yannick Assogba,
Software Engineer, Google Research

As TensorFlow.js is used more and more in
production environments, our team recognize...

**TensorFlow**