Post-training Quantization (PTQ)

```python
converter = tf.lite.TFLiteConverter.from_saved_model(CATS_VS_DOGS_SAVED_MODEL)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite_model = converter.convert()
tflite_model_file = 'converted_model.tflite'

with open(tflite_model_file, "wb") as f:
    f.write(tflite_model)
```

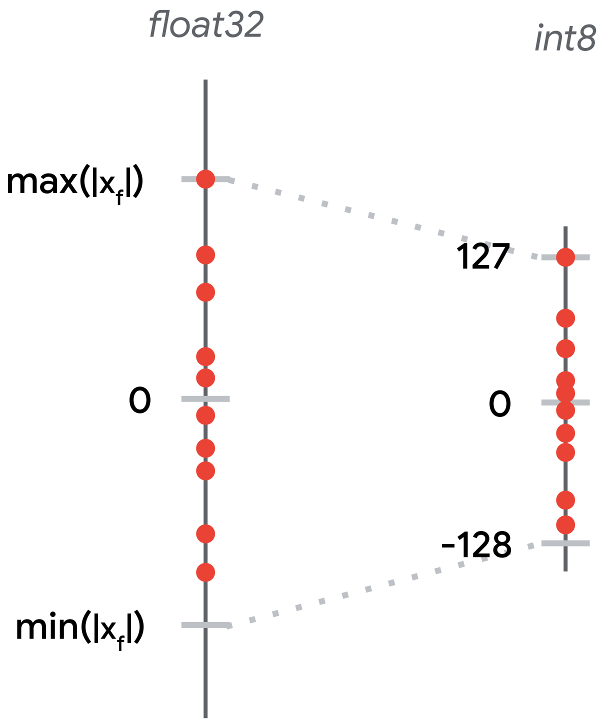
```python
converter = tf.lite.TFLiteConverter.from_saved_model(CATS_VS_DOGS_SAVED_MODEL)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite_model = converter.convert()
tflite_model_file = 'converted_model.tflite'

with open(tflite_model_file, "wb") as f:
    f.write(tflite_model)
```

```python
[tf.lite.Optimize.DEFAULT]

[tf.lite.Optimize.OPTIMIZE_FOR_SIZE]

[tf.lite.Optimize.OPTIMIZE_FOR_LATENCY]
```

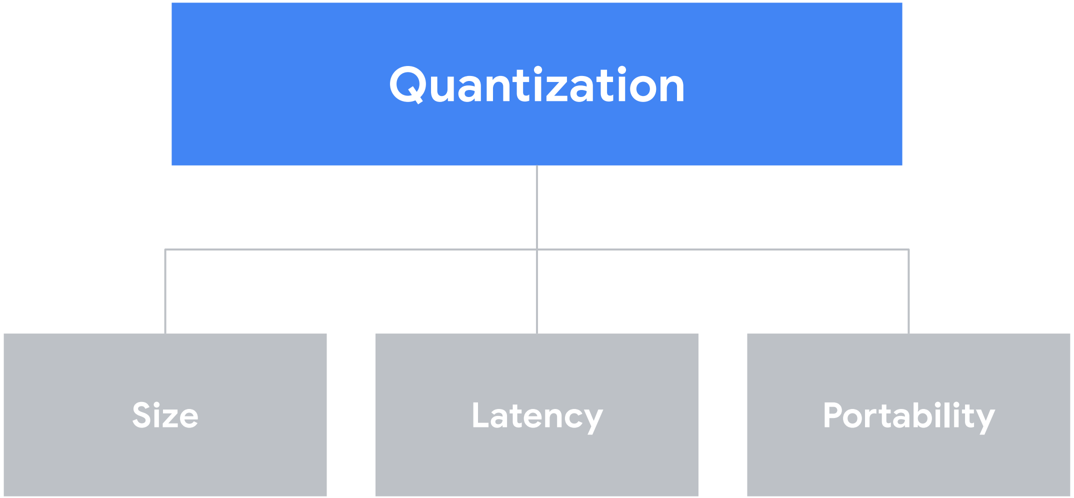
Quantization

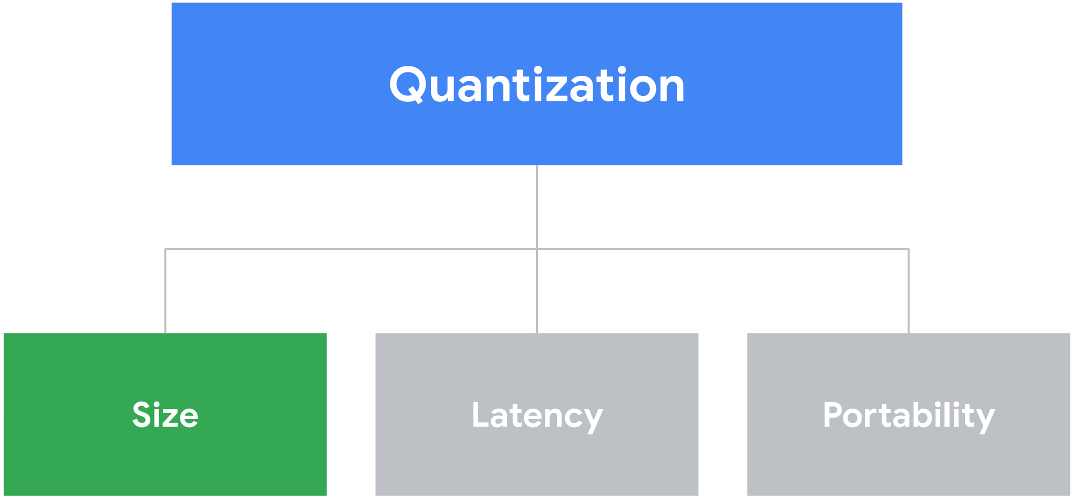
Quantization is an optimization that works by **reducing the precision** of the numbers used to represent a model's parameters, which by default are 32-bit floating point numbers. This results in a **smaller model size**, **better portability** and **faster computation**.

Reducing the Precision



Why do we Quantize?


Quantization
Size
Latency
Portability


Quantization
Size
Latency
Portability

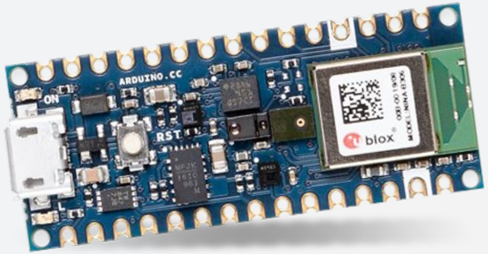
Size

Storage size: Smaller neural network models occupy less storage space on your device.

Storage & RAM Size

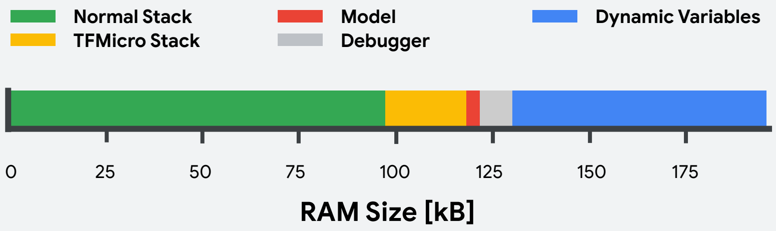
Storage size: Smaller neural network models occupy less storage space on your device, and in moving from 32-bits to 8-bits we readily get **4x** reduction in memory.

Our board (in your kit for Course 3) only has **256KB** of RAM (memory) and **1MB** of Flash (storage)



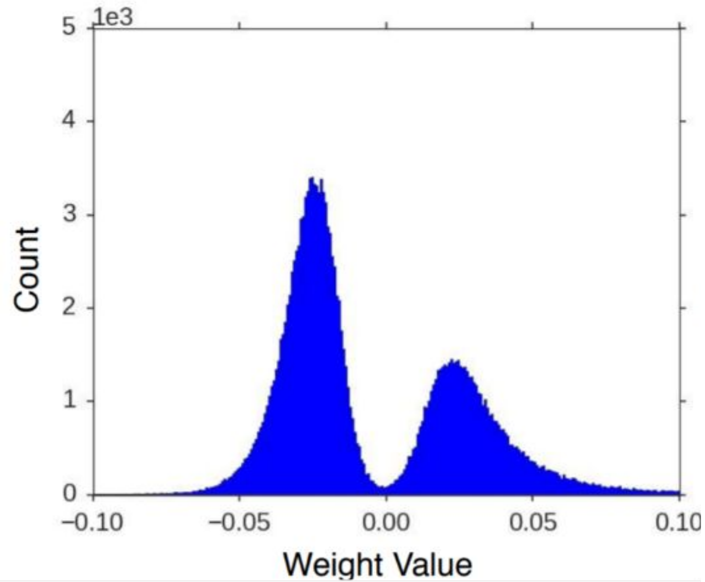
Storage & RAM Size

Less memory usage: Smaller models use less RAM when they are run, which frees up memory for other parts of your application to use, and can translate to better performance and stability.

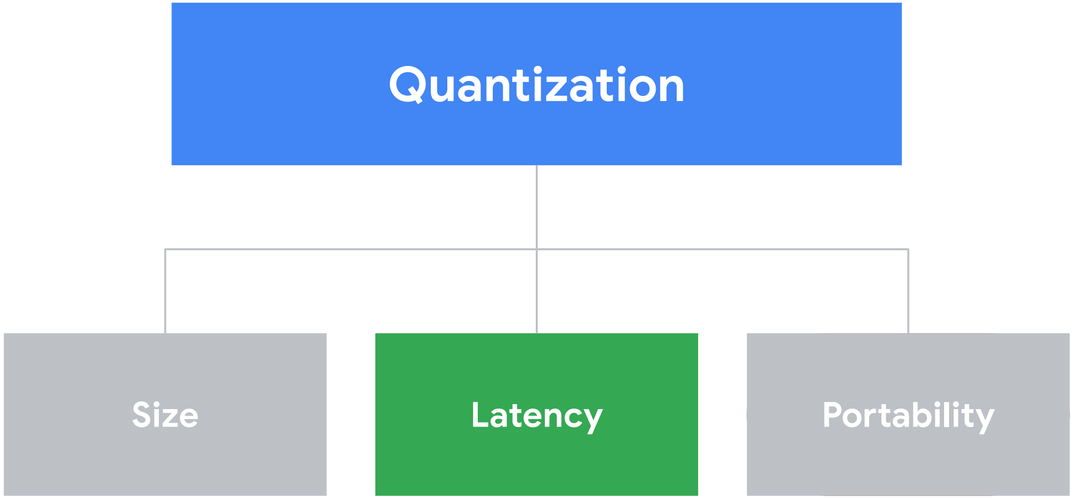


Weight Ranges

Weight distribution for AlexNet shows how most weight values are **concentrated** in a small range.

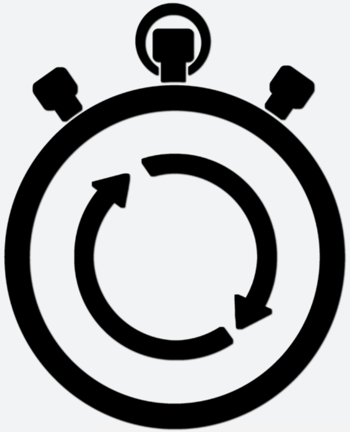


Source: "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding" by Song Han, Huizi Mao, William J. Dally


Quantization
Size
Latency
Portability

Latency

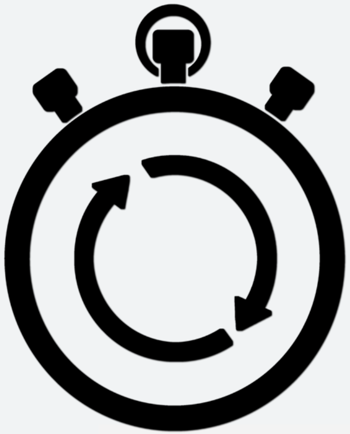
- ***Int8*** (v. fp32) format severely **reduces the computation** to run inference using a model, resulting in lower latency



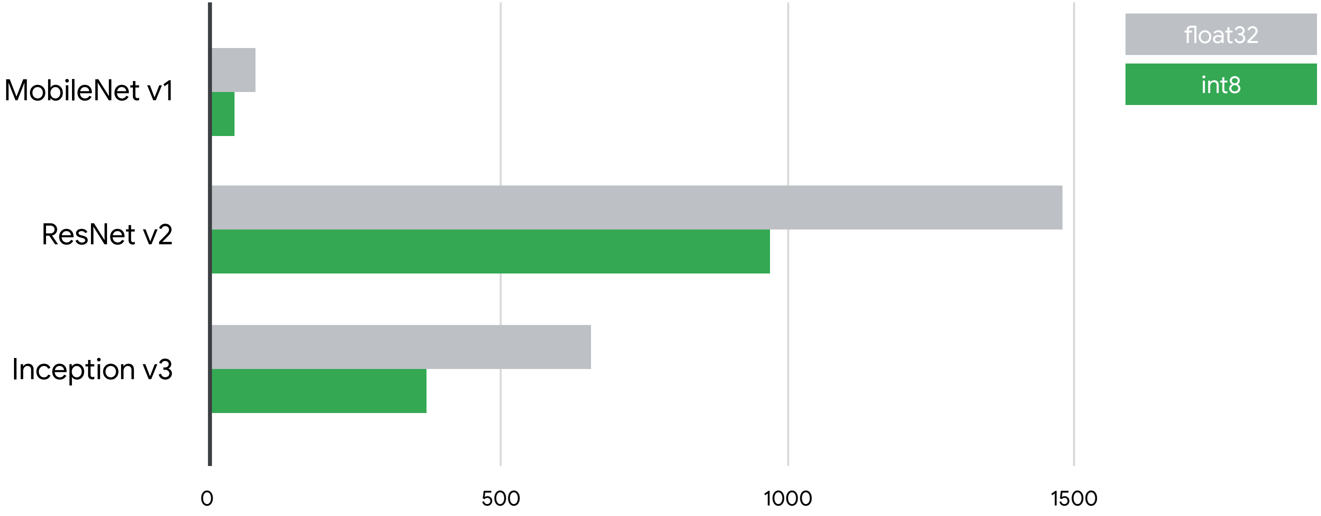
Latency

- ***Int8*** (v. fp32) format severely **reduces the computation** to run inference using a model, resulting in lower latency

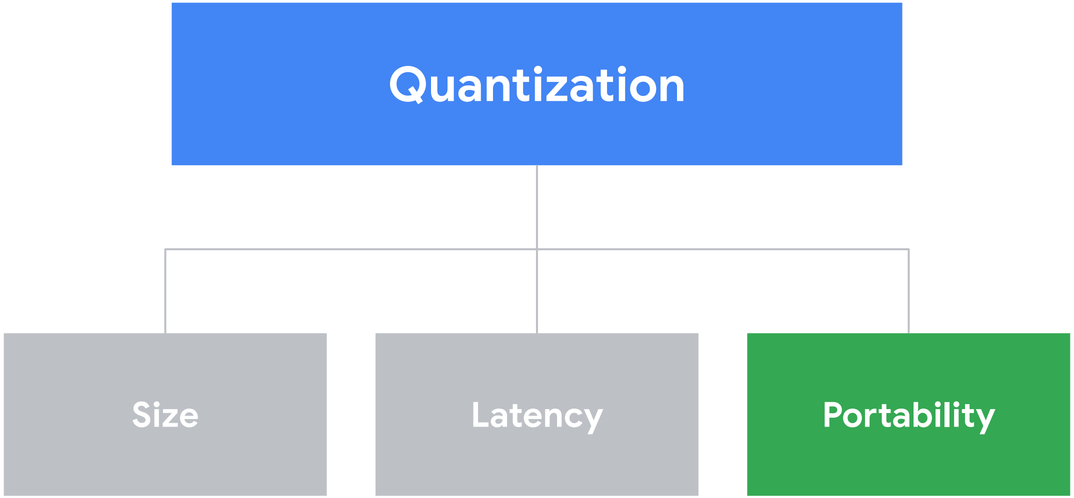
- Latency optimizations can also have a notable impact on **power consumption**.



Int8 v. Float (CPU time per inference)

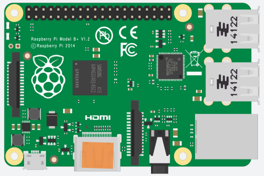


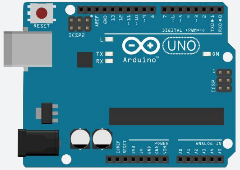
Quantized models are up to 2–4x faster on CPU and 4x smaller.


Quantization
Size
Latency
Portability

Portability **Trade-offs**

Not all embedded systems are created equal. Sacrifice **portability** across systems for **efficiency**.

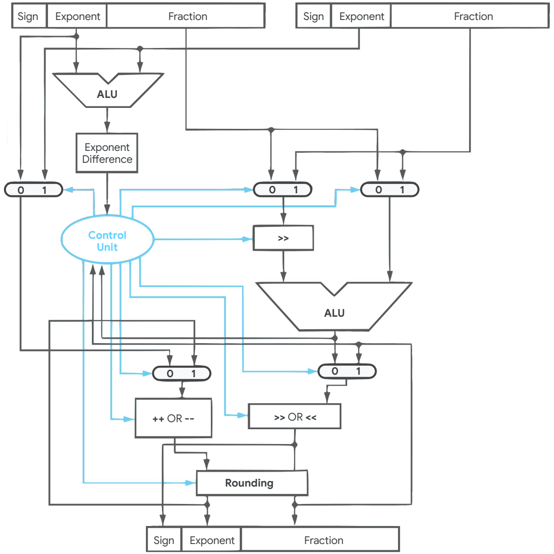




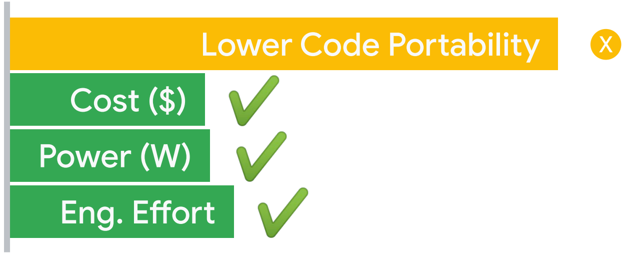
X	✓

Specific HW Implementation of a Library

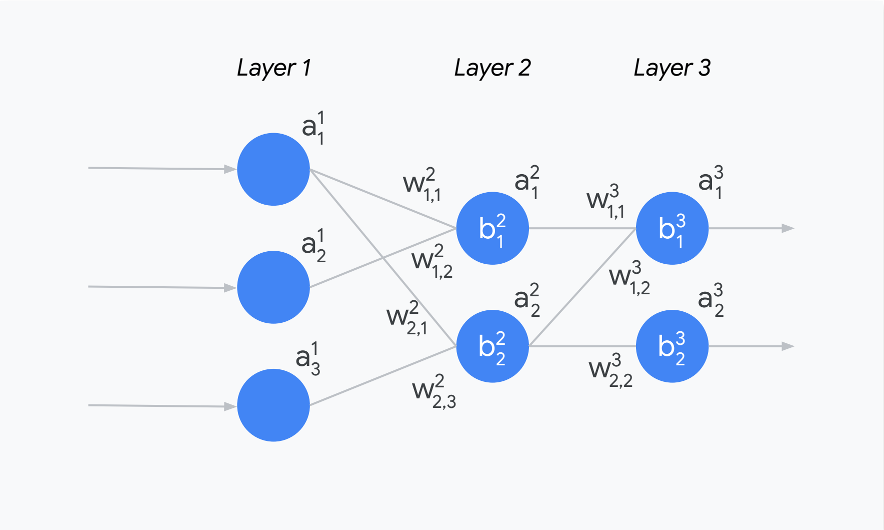
Single Precision
IEEE 754 Floating-Point Standard



Option 2

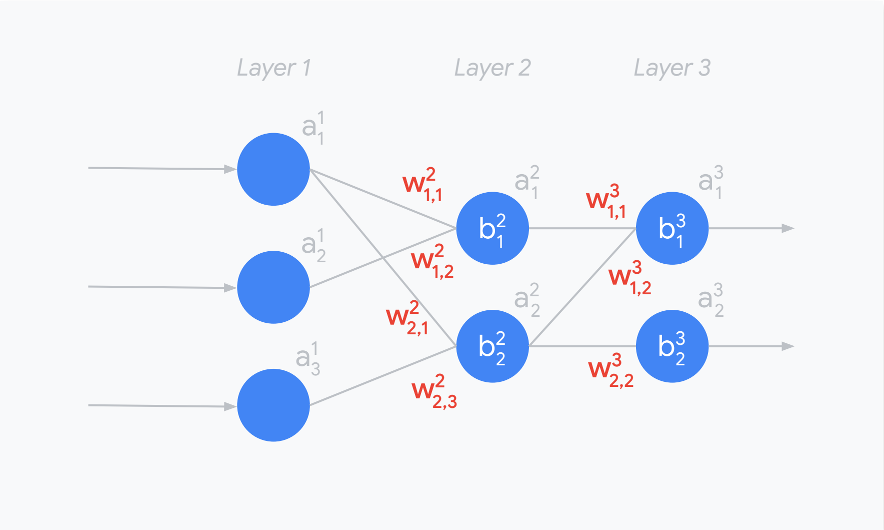


How do we Quantize?



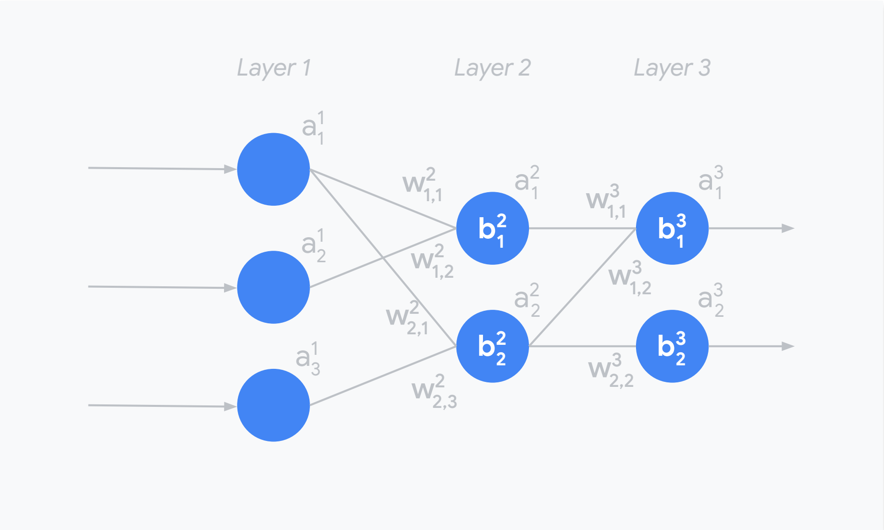
Layer 1
Layer 2
Layer 3
a11
a12
a13
w21,1
w21,2
w22,1
w22,3
b21
b22
a21
a22
w31,1
w31,2
w32,2
b31
b32
a31
a32

Weights


Layer 1
Layer 2
Layer 3
a^1_1
a^1_2
a^1_3
$w^2_{1,1}$
$w^2_{1,2}$
$w^2_{2,1}$
$w^2_{2,3}$
b^2_1
b^2_2
a^2_1
a^2_2
$w^3_{1,1}$
$w^3_{1,2}$
$w^3_{2,2}$
b^3_1
b^3_2
a^3_1
a^3_2

Weights

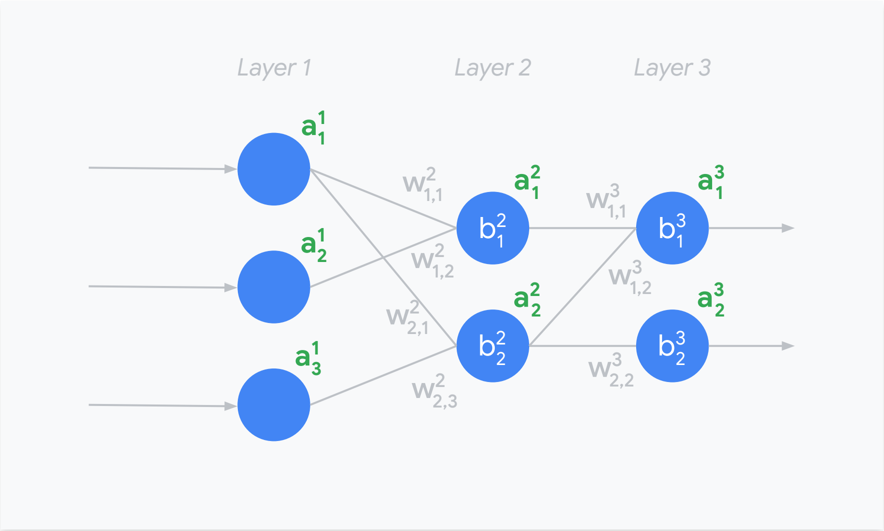
Biases



Weights

Biases

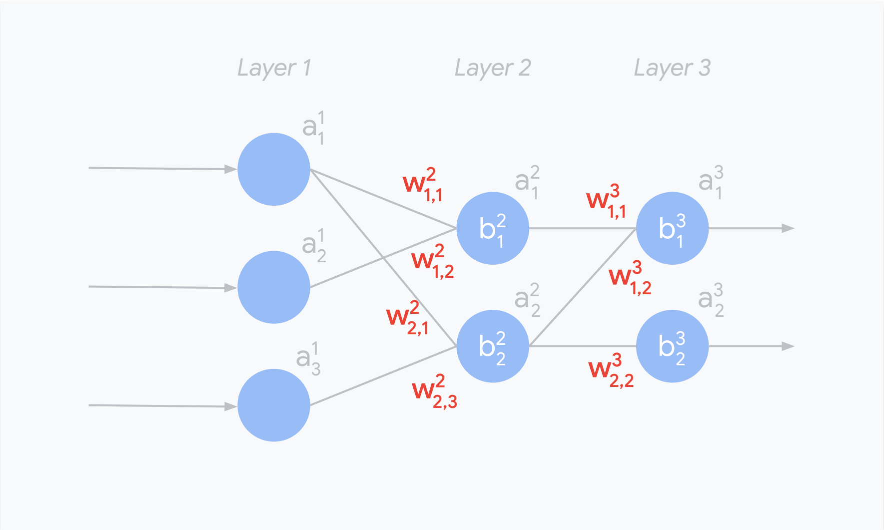
Activations



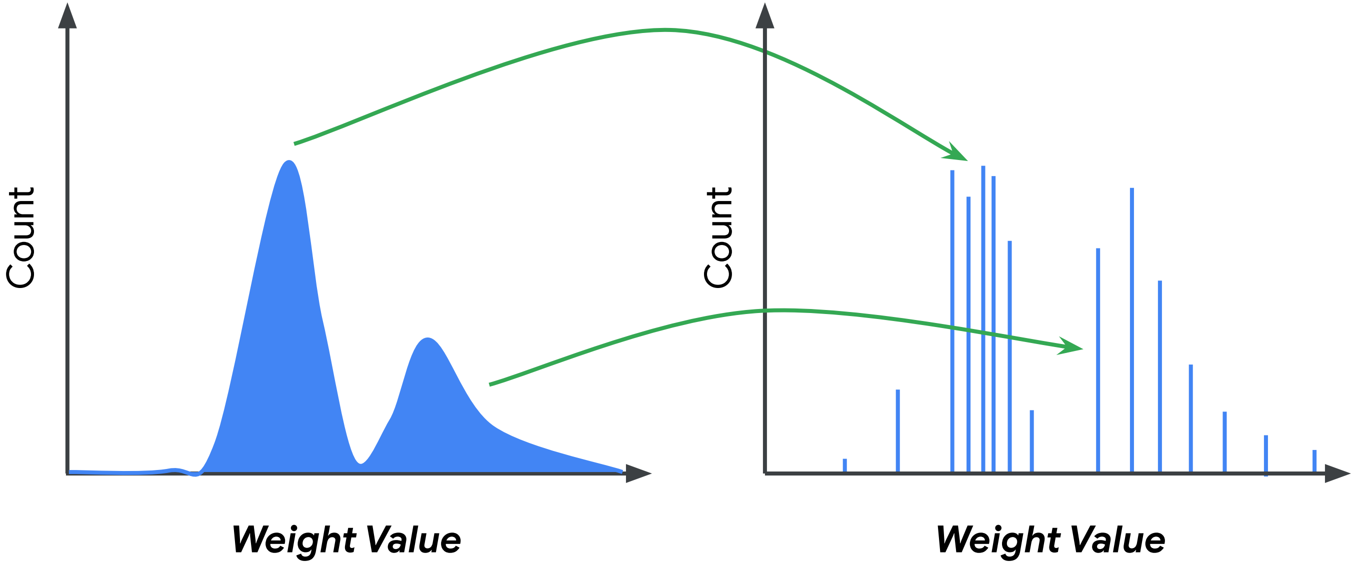
Weights

Biases

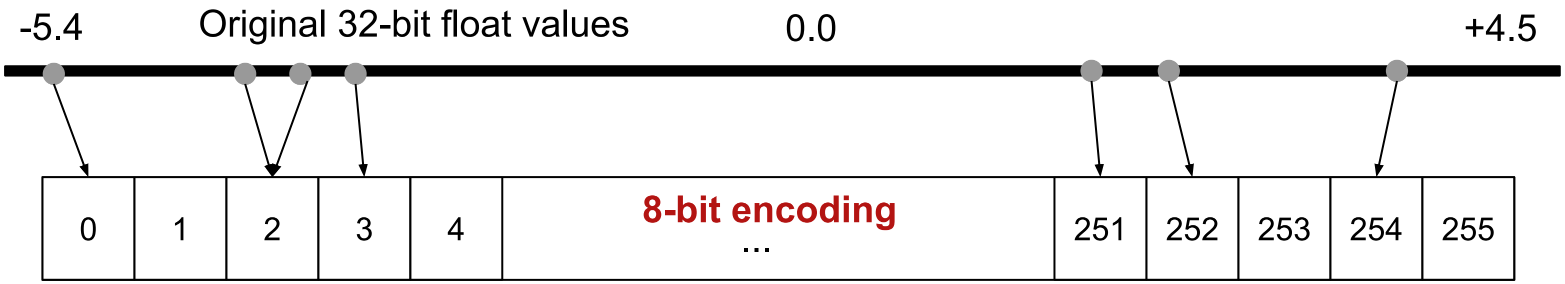
Activations

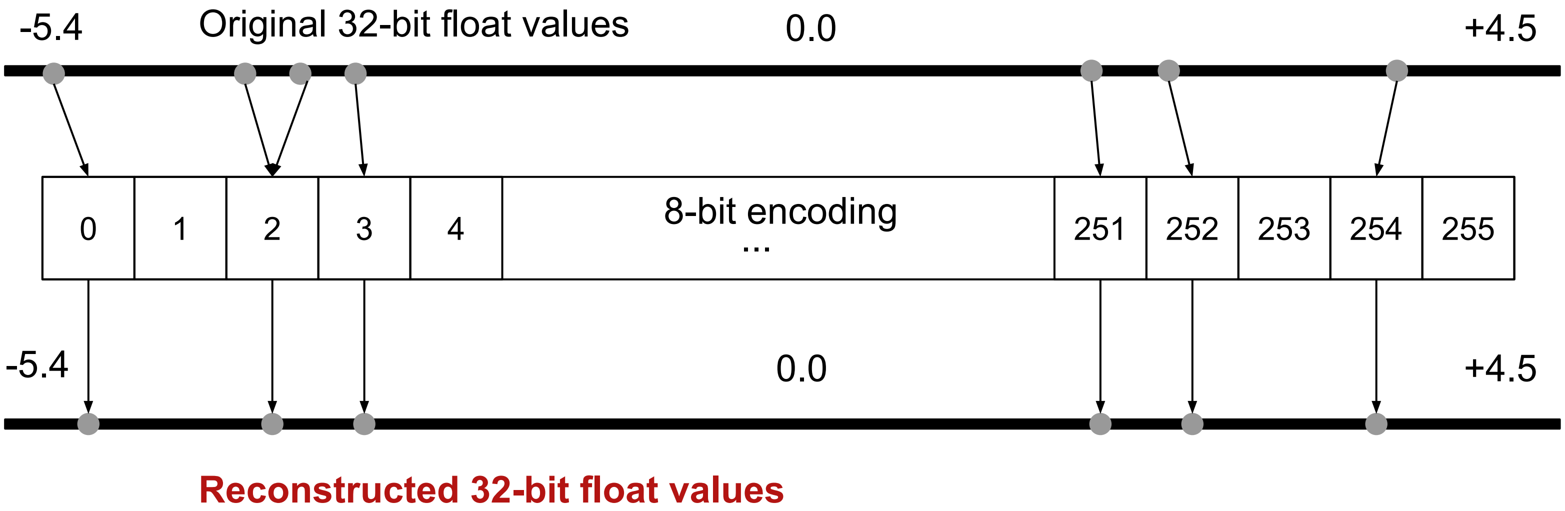


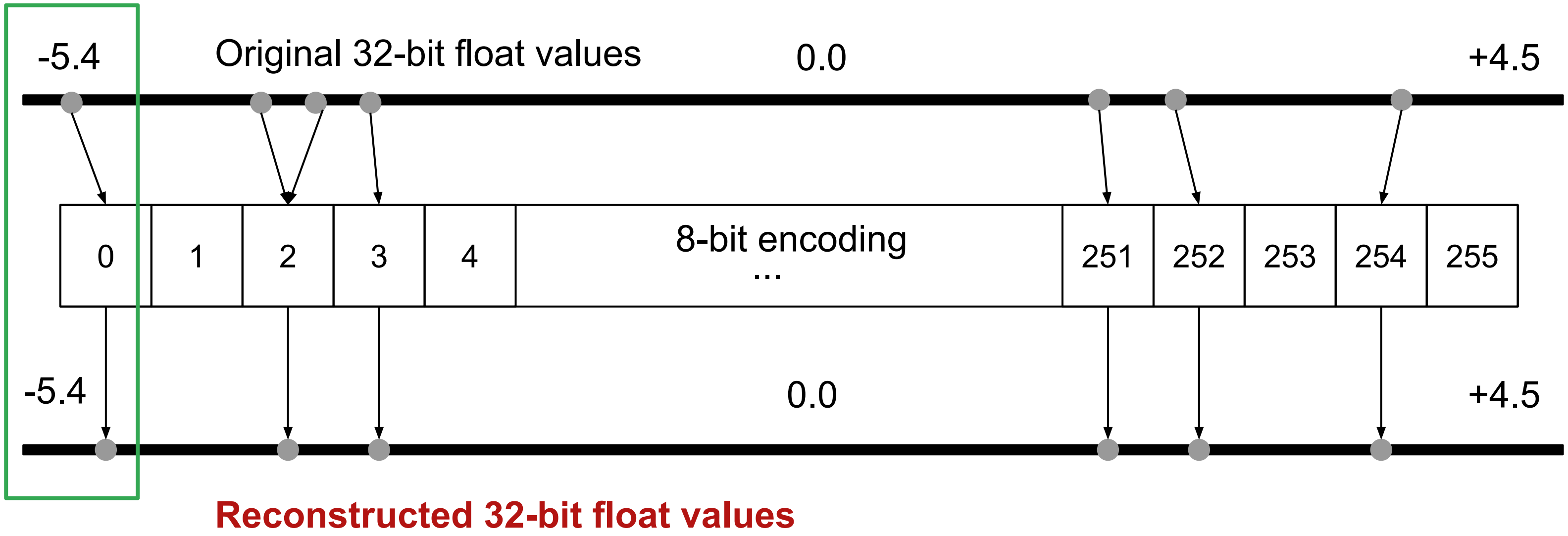
Reduce Precision (**Discretize**)

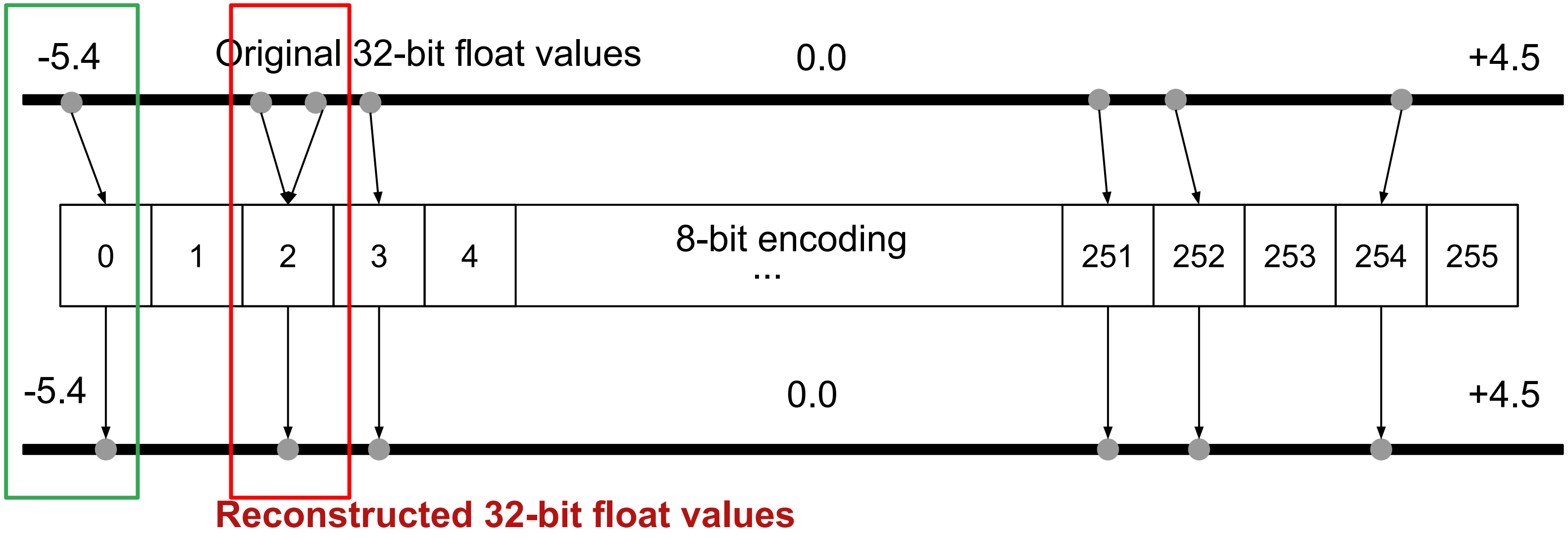


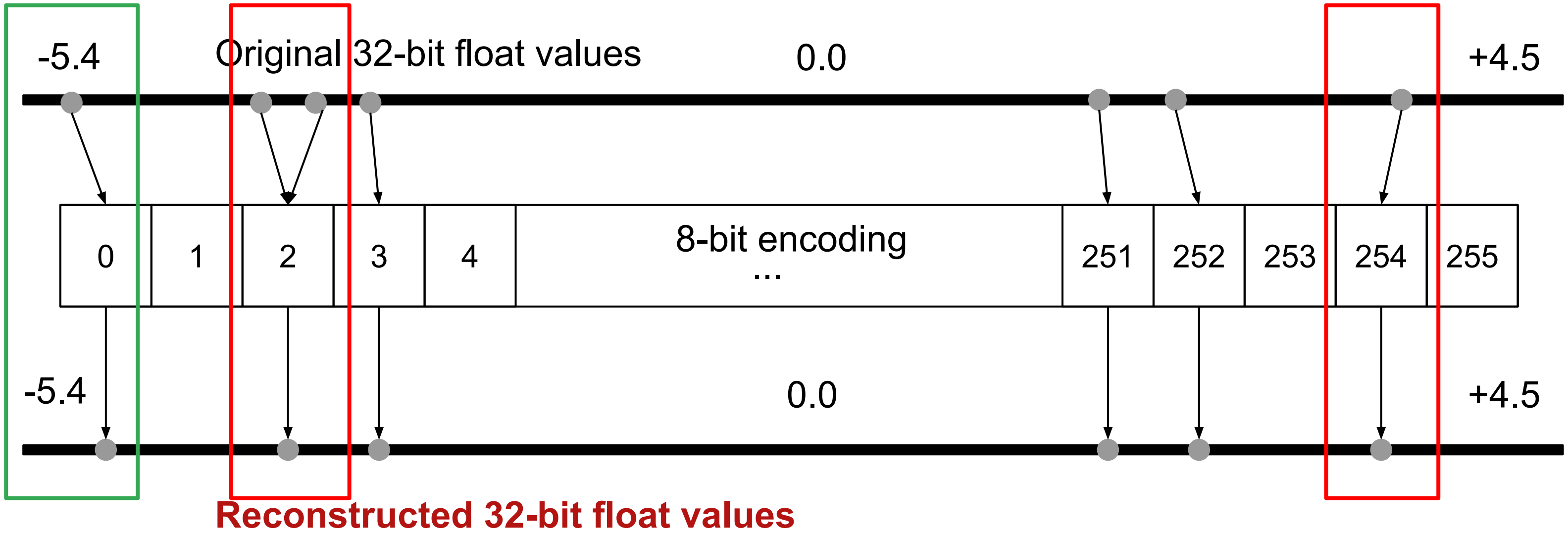

-5.4
Original 32-bit float values
0.0
+4.5


-5.4
Original 32-bit float values
0.0
+4.5
0
1
2
3
4
8-bit encoding
...
251
252
253
254
255


-5.4
Original 32-bit float values
0.0
+4.5
0
1
2
3
4
8-bit encoding
...
251
252
253
254
255
-5.4
0.0
+4.5
Reconstructed 32-bit float values

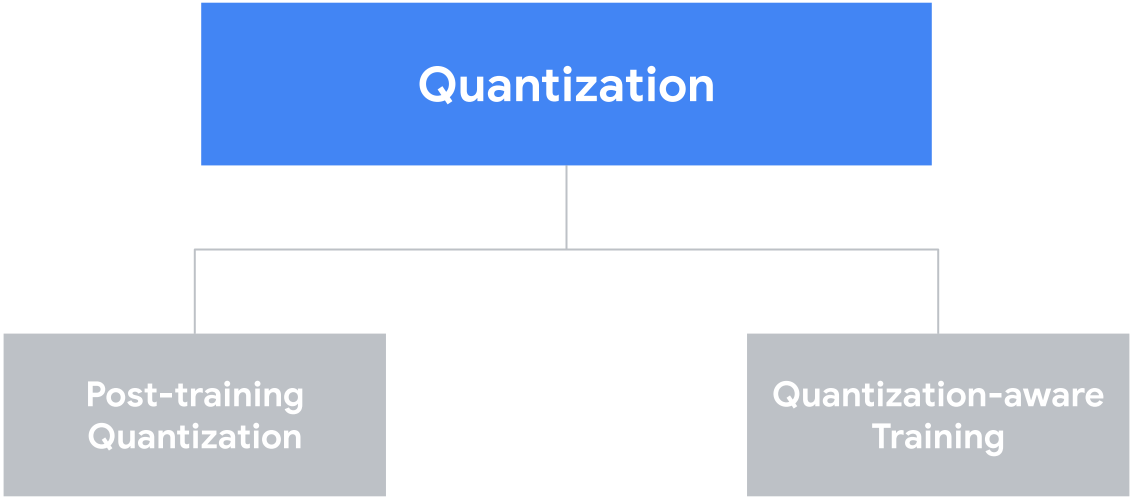

-5.4
Original 32-bit float values
0.0
+4.5
0
1
2
3
4
8-bit encoding
...
251
252
253
254
255
-5.4
0.0
+4.5
Reconstructed 32-bit float values

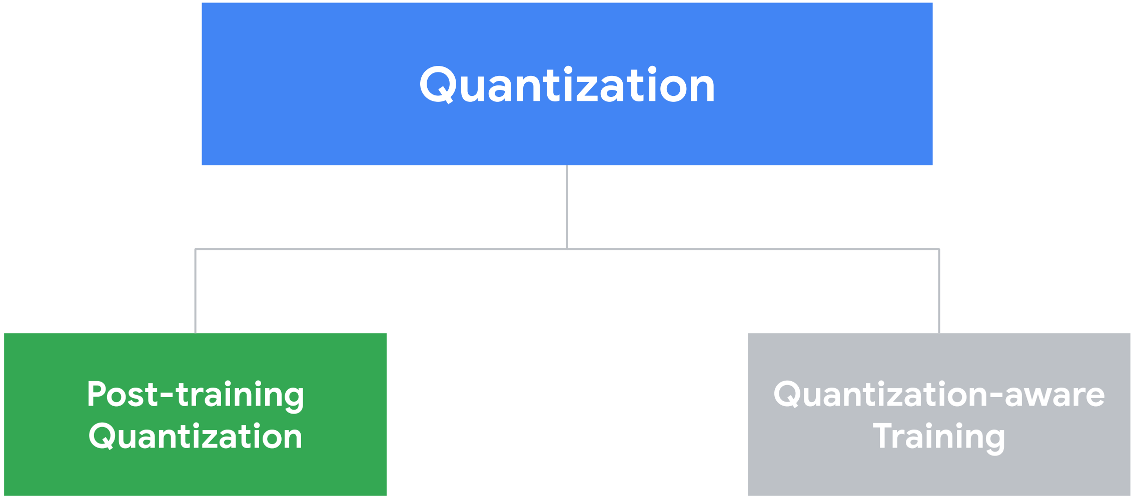

-5.4
Original 32-bit float values
0.0
+4.5
0
1
2
3
4
8-bit encoding
...
251
252
253
254
255
-5.4
0.0
+4.5
Reconstructed 32-bit float values

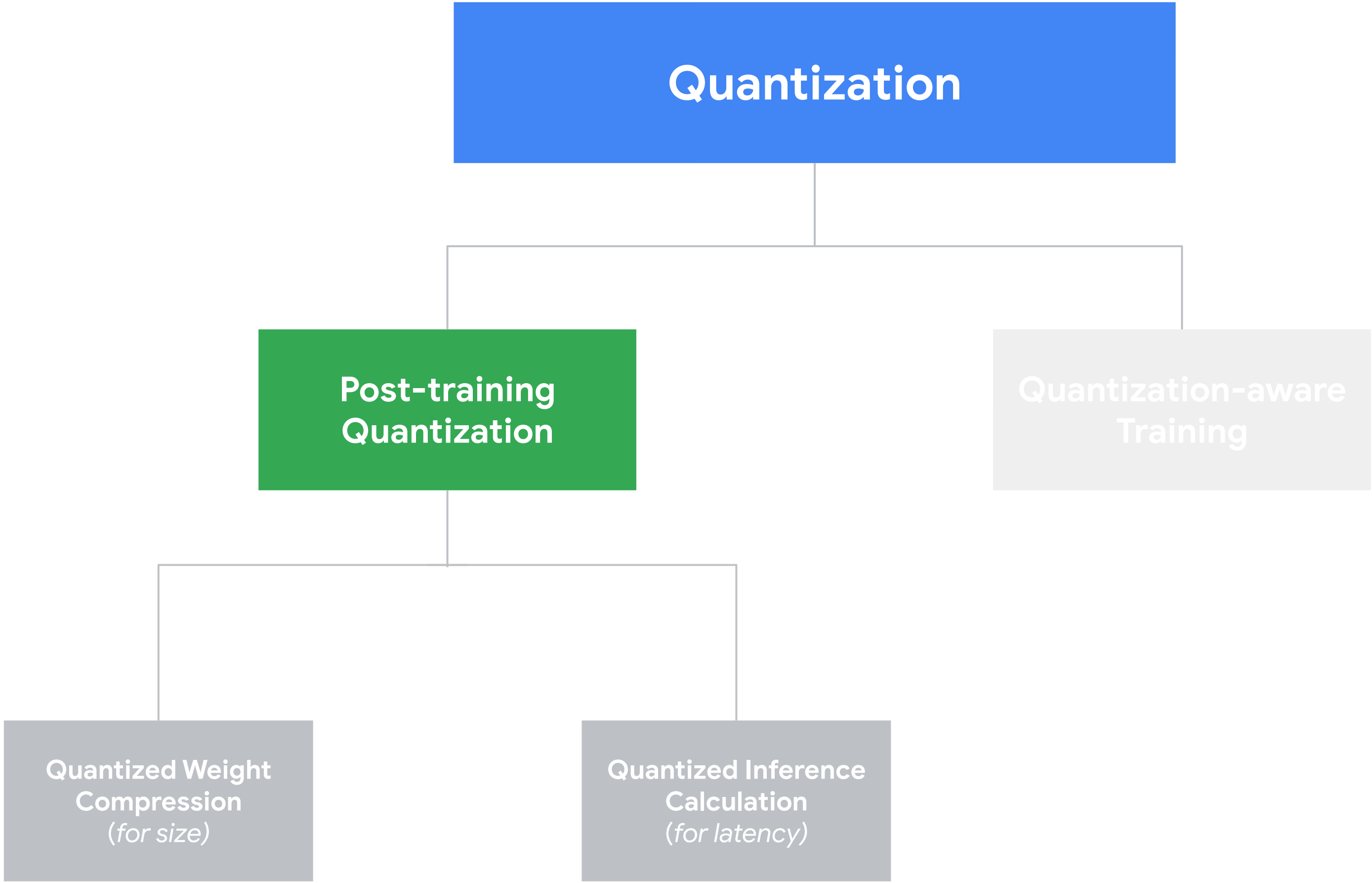


-5.4
Original 32-bit float values
0.0
+4.5
0
1
2
3
4
8-bit encoding
...
251
252
253
254
255
-5.4
0.0
+4.5
Reconstructed 32-bit float values

Quantization


Quantization
Post-training Quantization
Quantization-aware Training


Quantization
Post-training Quantization
Quantization-aware Training


Quantization
Post-training Quantization
Quantization-aware Training
Quantized Weight Compression
(for size)
Quantized Inference Calculation
(for latency)

Quantized Weight Compression
(for size)

Quantized Inference Calculation
(for latency)

Decompress each weight value from **8-bit integer** into a **fp32 floating-point** value before multiplying it with the input value:

```
output = … inputn * decompress(q_weightn)
```

Where:

```
decompress(quantized_code) {
    return float((quantized_code / 255.0) * (max - min)) + min;
}
```

Quantized Weight Compression
(for size)

Quantized Inference Calculation
(for latency)

Imagine that we artificially **reduce the precision** of every input to the ***dot product***, so that they're no longer using the full range of a 32-bit float:

```
output = … quantize(inputn, step) * quantize(weightn, step)
```

Where:

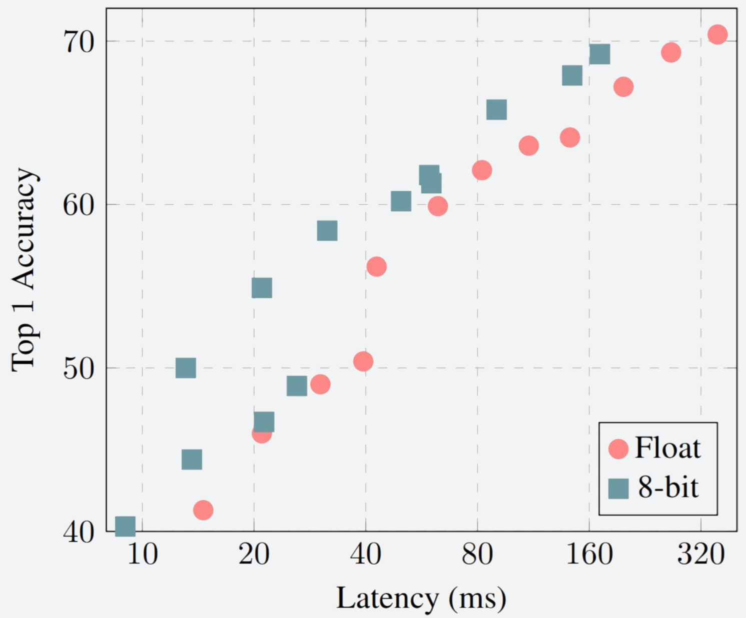
```
quantize(x, step) {
 return round(x*step) / step;
}
```

e.g., quantize(3.14, 1.0) = 3.0 *(rounding to nearest whole number)* and quantize(3.14, 0.1) = 3.1 **(rounding to nearest 1/10)**

What are the trade-offs?

Accuracy-Latency **Trade-off**

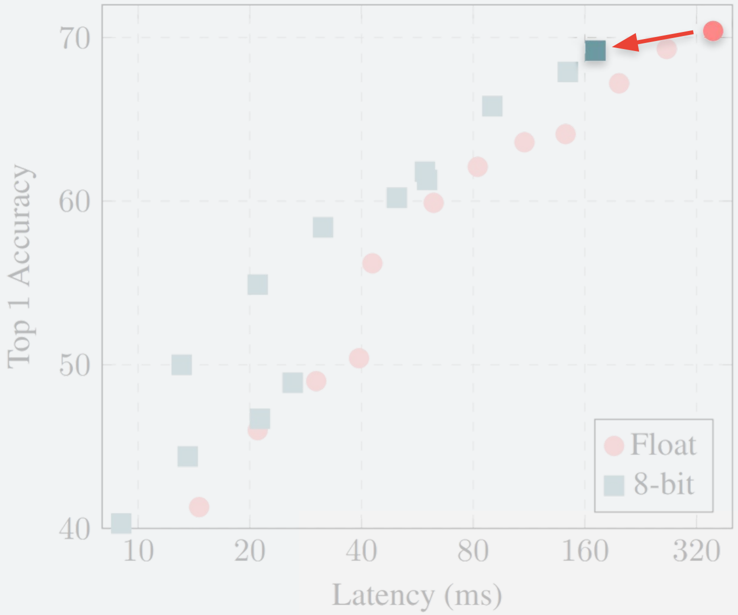
Quantization works well but performance but can suffer from **accuracy loss** during ***inference***.



Source: "Accuracy-latency tradeoff with MobileNets" Benoit et al.

Accuracy-Latency **Trade-off**

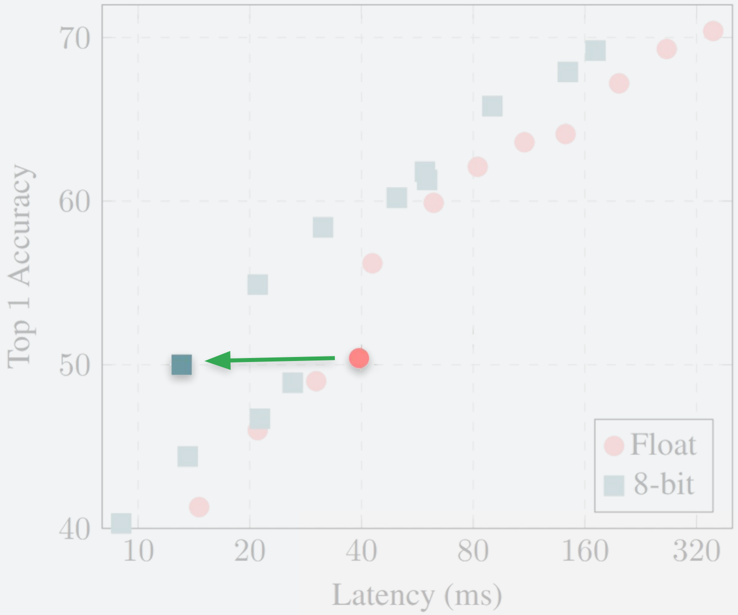
Quantization works well but performance but can suffer from **accuracy loss** during ***inference***.



Source: "Accuracy-latency tradeoff with MobileNets" Benoit et al.

Accuracy-Latency **Trade-off**

Quantization works well but performance but can suffer from **accuracy loss** during ***inference***.



Source: "Accuracy-latency tradeoff with MobileNets" Benoit et al.

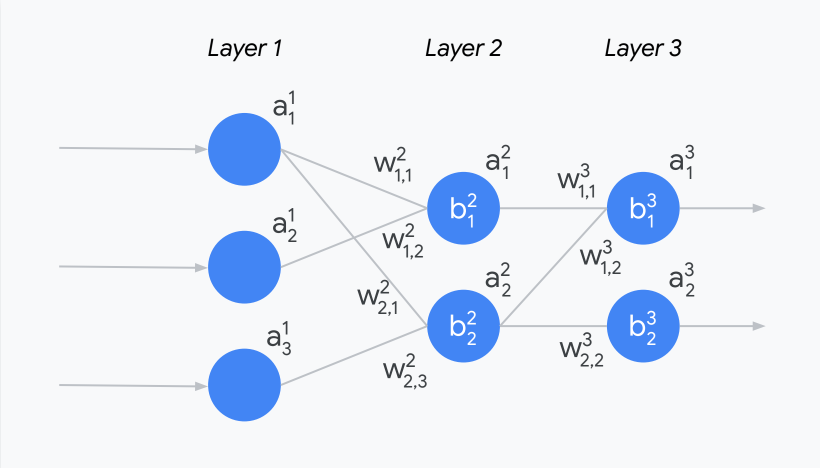
About Quantizing **Other NN Parts?**

- Weights
- Activations
- Channels
- Tensors
- Layers
- ...

Every network has **something unique** for it, so the degree to which you can quantize (e.g., weights, activations) ***will vary***.

About Quantizing **Other NN Parts?**

- Weights
- Activations
- Channels
- Tensors
- Layers
- ...



In Summary...

Summary

Doing all calculations in eight-bit integers offers some compelling advantages:

- **Faster arithmetic.** You need a lot fewer gates to implement an eight-bit integer multiply-add than a 32-bit floating point operation.

Summary

Doing all calculations in eight-bit integers offers some compelling advantages:

- **Faster arithmetic.** You need a lot fewer gates to implement an eight-bit integer multiply-add than a 32-bit floating point operation.
- **Lower memory demands.** We're only accessing eight bits instead of thirty-two, which reduces the load on the memory system by 75%.

Summary

Doing all calculations in eight-bit integers offers some compelling advantages:

- **Faster arithmetic.** You need a lot fewer gates to implement an eight-bit integer multiply-add than a 32-bit floating point operation.
- **Lower memory demands.** We're only accessing eight bits instead of thirty-two, which reduces the load on the memory system by 75%.
- **Reduced resource requirements.** Many low-end microcontrollers and DSPs lack floating-point hardware, so avoiding floats increases portability.