2020 embedded VISION summit

Practical DNN Quantization Techniques and Tools

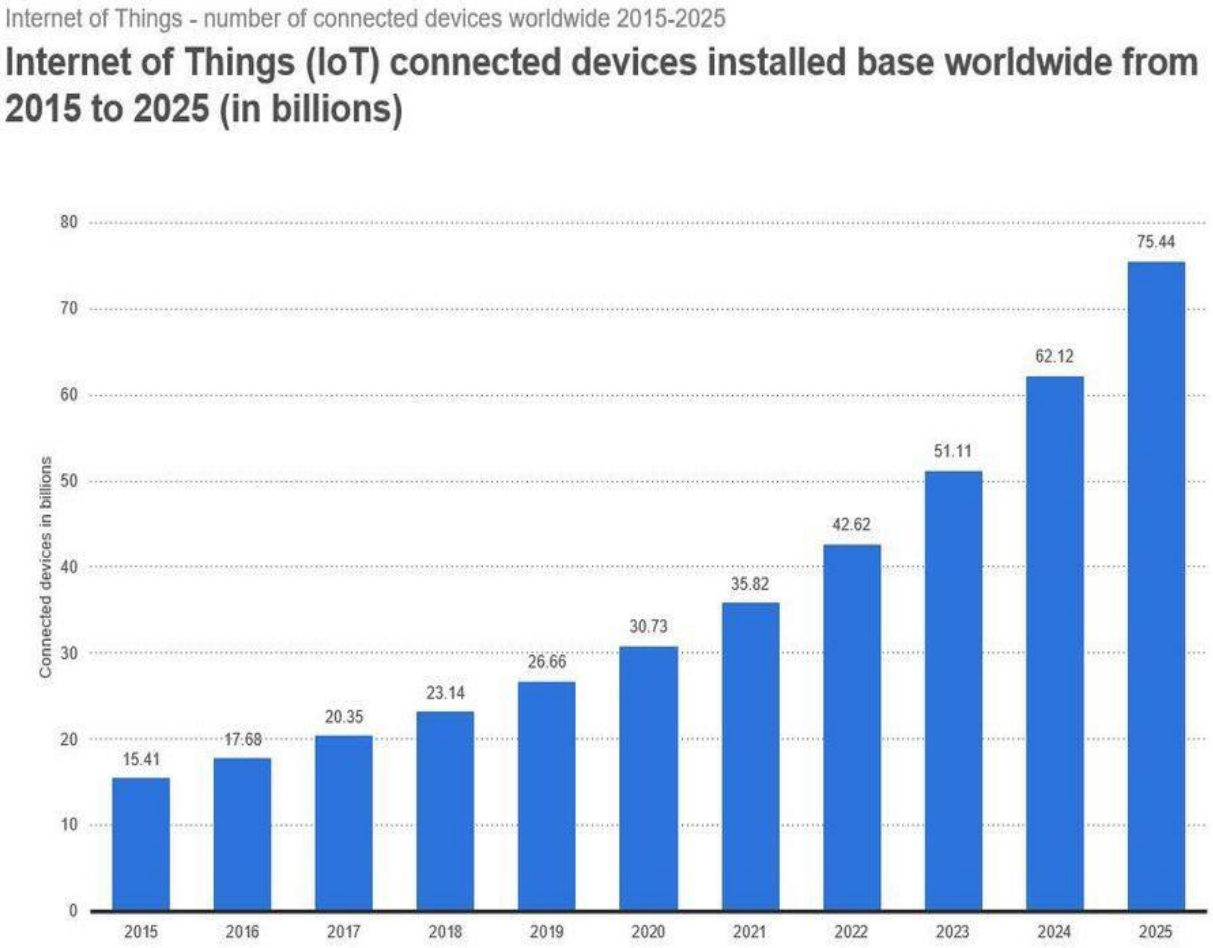
Raghuraman Krishnamoorthi
Software Engineer, Facebook
Sept 2020

Outline

- Motivation
- Quantization: Overview
- Quantizing deep networks
  - Post Training quantization
  - Quantization aware training
- Best Practices
- Conclusions

Motivation(1)

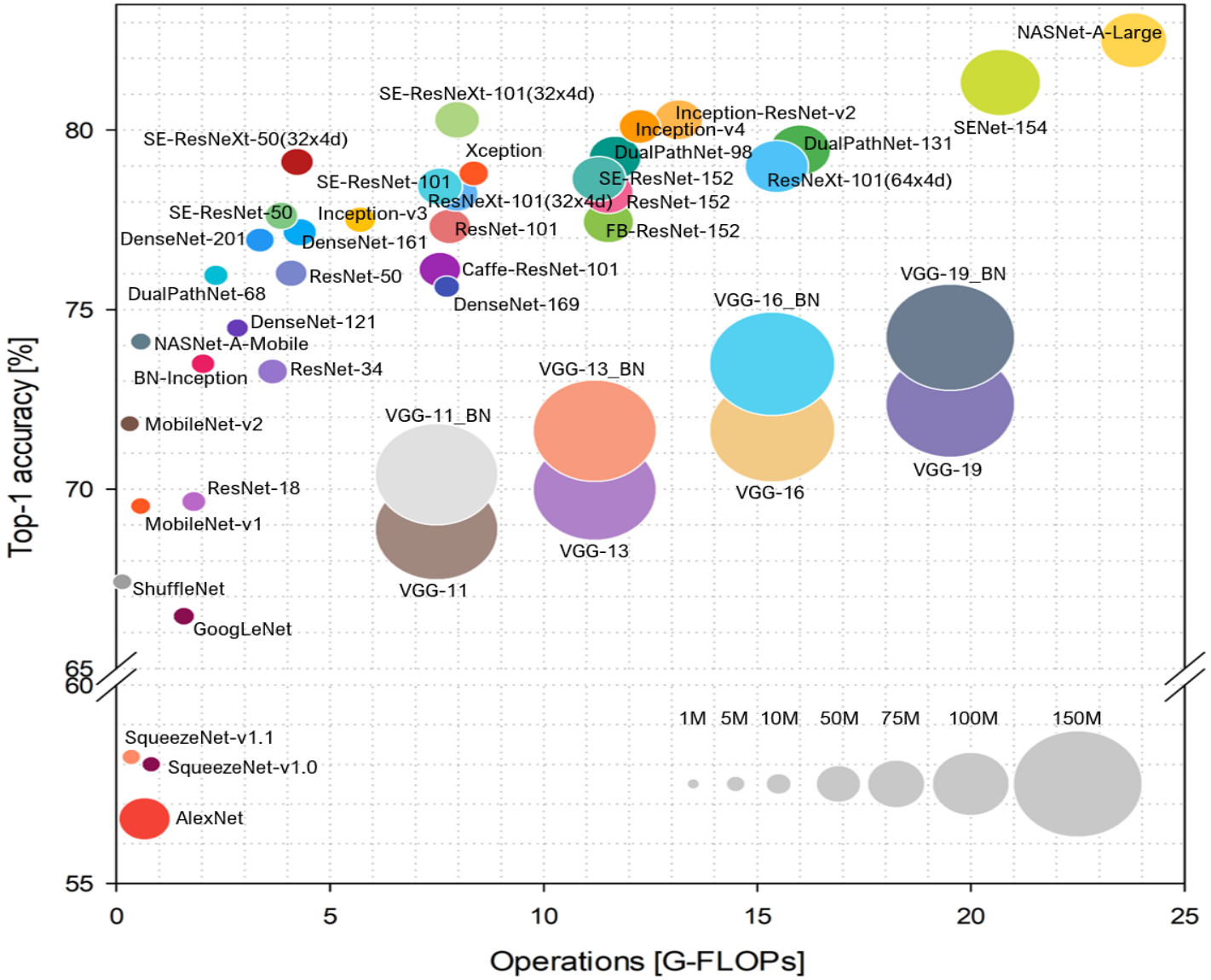
- Number of edge devices is growing rapidly, lots of these devices are resource constrained.
- Gartner predicts that 80% of mobile devices shipped in 2022 will have on device AI.



Source: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

Motivation(2)

- While models are becoming more efficient, high accuracy still implies high complexity

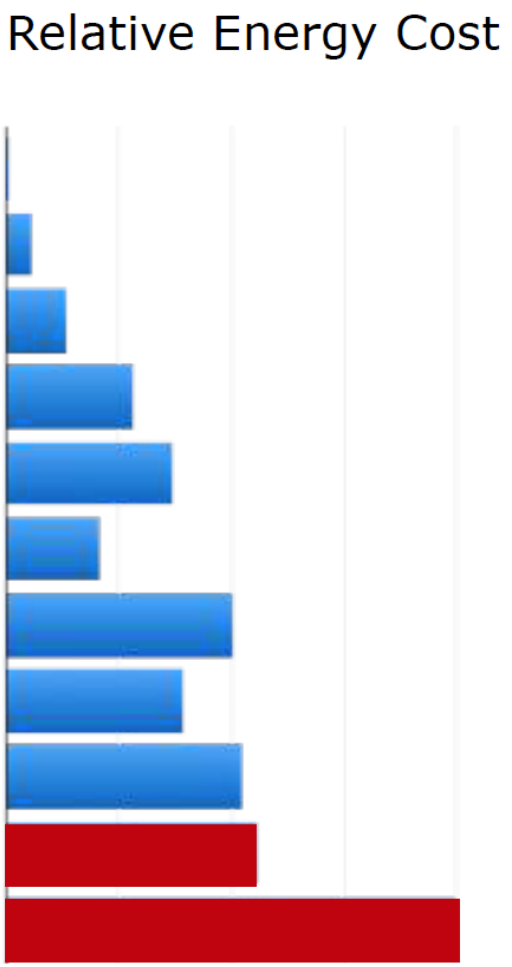


From: [Benchmark Analysis of Representative Deep Neural Network Architectures](), Simone Bianco et al,

Motivation(3)

- Energy consumption is dominated by memory accesses
  - Reduced precision is critical to save power

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Multiply	0.2
32b Multiply	3.1
16b FP Multiply	1.1
32b FP Multiply	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640



[**Horowitz**, *ISSCC* 2014]

Quantization

- Many approaches to solve the problems outlined here:
  - Better hardware accelerators: DSPs, NPUs
    - Requires new custom hardware
  - Efficient deep network architectures: Mnasnet, Mobilenet, FBNet
    - Requires new model architectures
- A simpler approach that does not require re-design of models/new hardware is quantization.
  - Quantization refers to techniques to perform computation and storage at reduced precision
  - Works in combination with above approaches
  - Table stakes for low power custom hardware

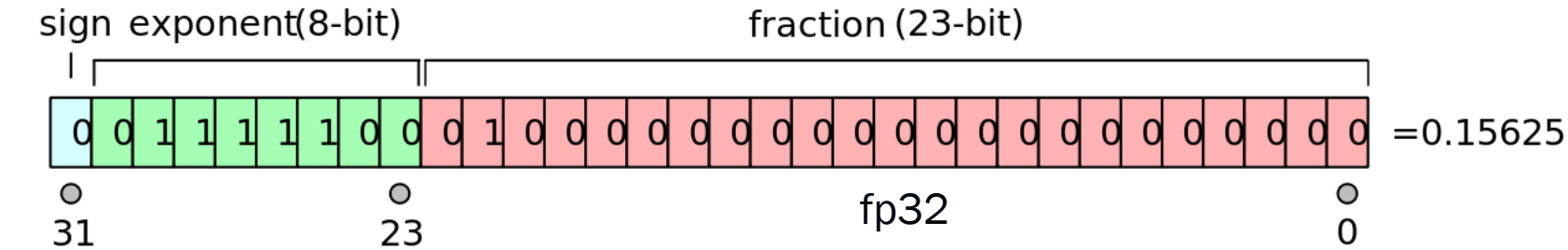
Quantization: Benefits

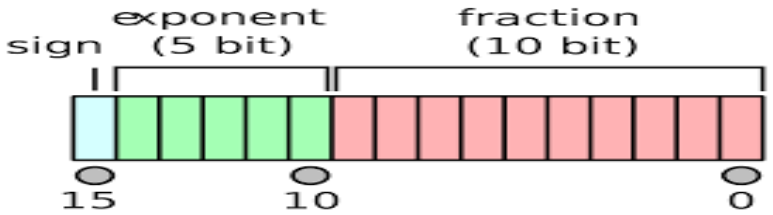
Benefits	Quantization
Applicability	Broad applicability across models and use cases
Support	Supported by x86, Nvidia Volta, ARM, Mali, Hexagon
Software Support	Kernel libraries widely available
Memory Size	4x reduction
Memory Bandwidth/Cache	4x reduction
Compute	2x to 4x speedup, depending on ISA
Power	~4x*

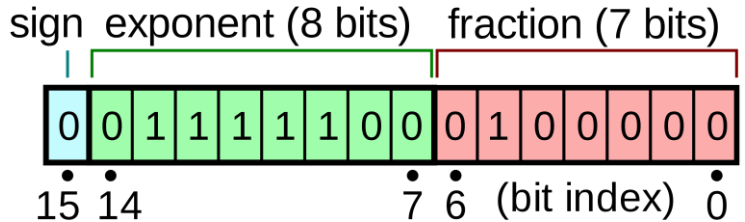
Note: Comparing float32 implementations with 8 bit inference on CPU
* Higher power savings are possible due to cache effects

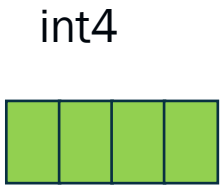
Background: Quantization(1)

- Quantization refers to mapping values from fp32 to a lower precision format.
  - Specified by
    - Format
    - Mapping type
    - Granularity



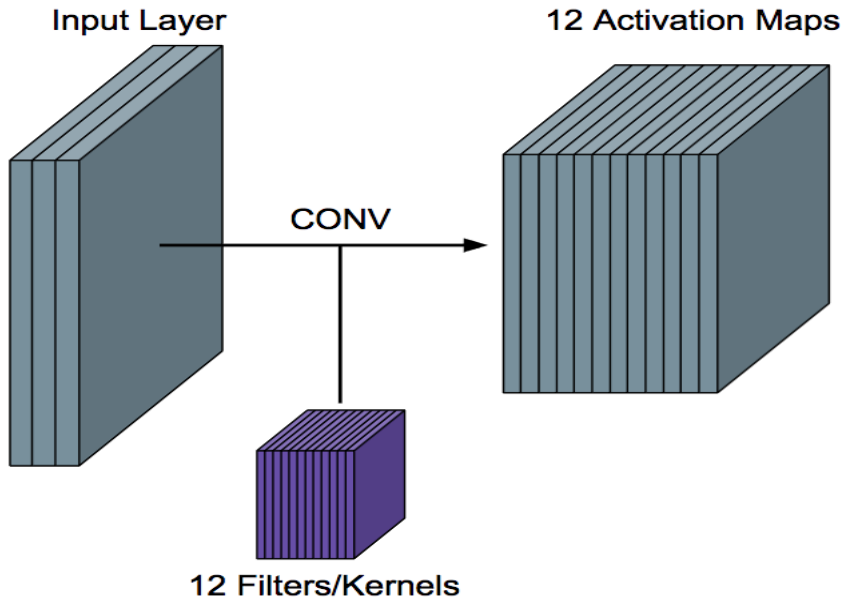






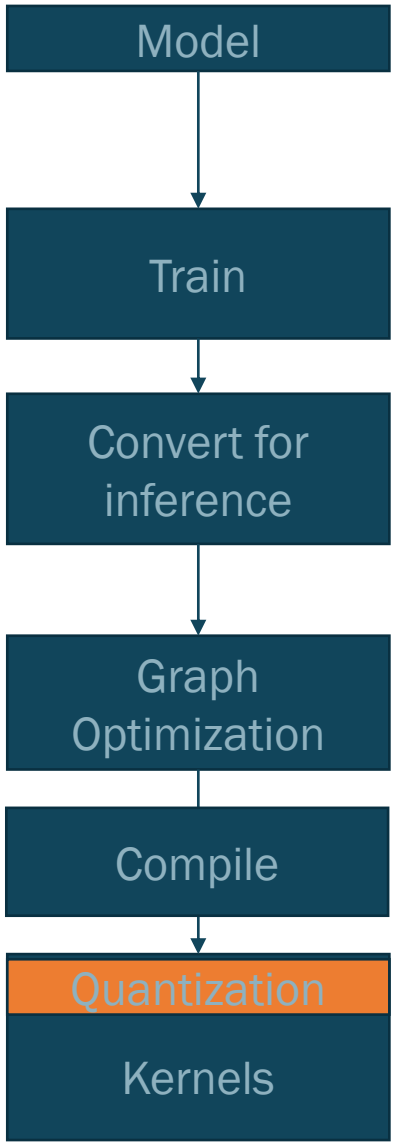
Background: Quantization(2)

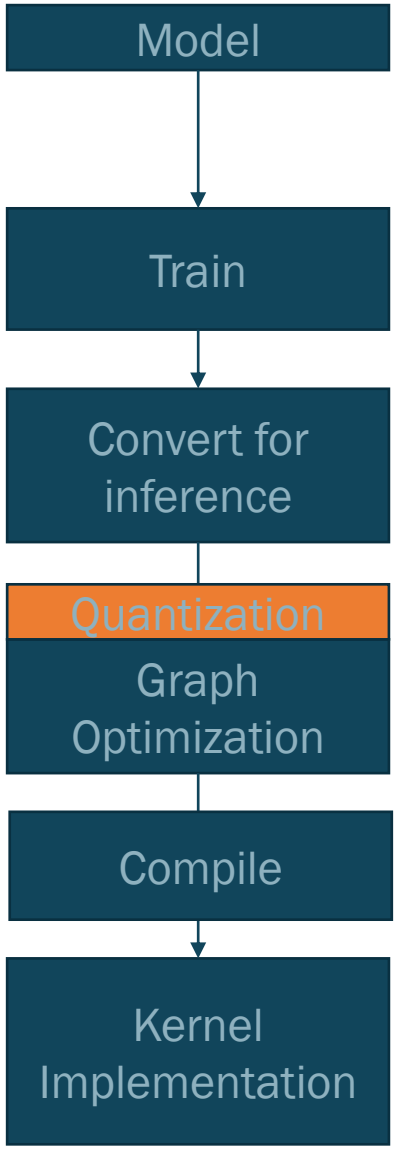
- We also consider different granularities of quantization:
  - Per tensor quantization
    - Same mapping for all elements in a tensor.
  - Per-axis/per-channel quantization:
    - Choose quantizer parameters independently for each row (fc layers) or for each conv kernel (conv layers) for weights

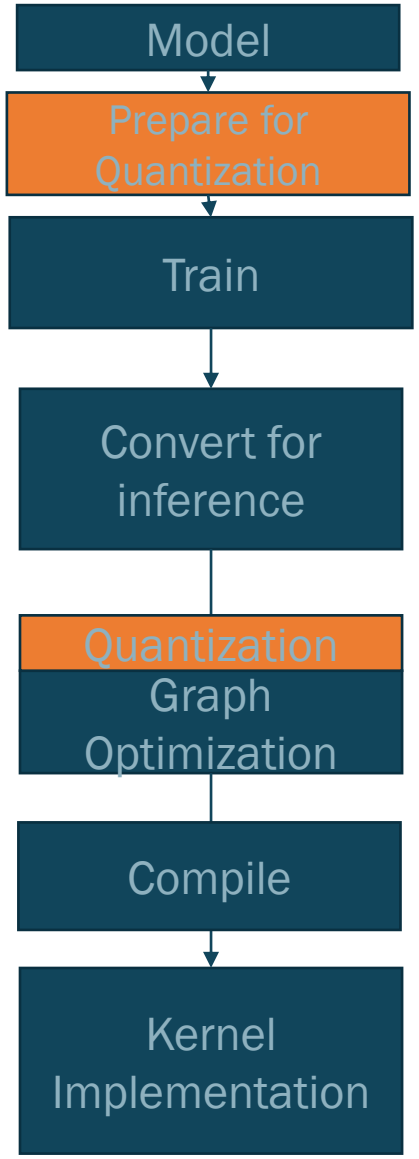


Quantizing Deep Neural Networks

Model Quantization: Overview







Post Training Quantization: Weight Compression

- Simplest quantization scheme is to compress the weights to lower precision
  - Requires no input data and can be done statically as part of preparing a model for inference
  - Hardware accelerators can benefit if de-compression is done after memory access
    - Trivial for case of fp16/int8 quantization of weights.
  - K-means compression is also supported in select platforms and is amenable to simple de-compression
    - Scatter-Gather operation in select processors
    - Supported in CoreML

Dynamic Quantization

- Dynamic quantization refers to schemes where the activations are read/written in fp32 and are dynamically quantized to lower precisions for compute.
- Requires no calibration data
- Data exchanged between operations is in floating point, so no need to worry about format conversion.
- Provides performance improvements close to static quantization as long as the latency is compute bound.
  - Suitable for inference for transformer/LSTM models
- Supported by:
  - Pytorch
  - Tensorflow Lite

Quantizing Weights and Activations

- Post training quantization refers to quantizing both weights and activations to reduced precision, typically int8.
- Requires estimation of statistics of activations for determining quantizer parameters.
- Quantizer parameters are determined by minimizing an error metric:
  - KL Divergence: TensorRT
  - Saturation error: Tensorflow Lite
  - Expected quantization error: Pytorch

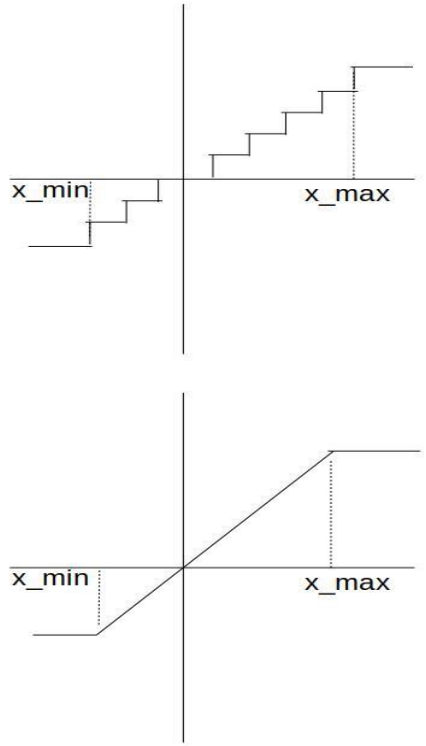
Modeling Quantization During Training

- Emulate quantization by quantizing and de-quantizing in succession
  - Values are still in floating point, but with reduced precision
- $x_{out} = FakeQuant(x)$

$$= s.\left(Clamp\left(round\left(\frac{x}{s}\right) - z\right) + z\right)$$

$$= DeQuant(Quant(x))$$

- Can also model quantization as a stochastic rounding operation



Fake Quantizer (top), showing the quantization of output values. Approximation for purposes of derivative calculation (bottom).

Comparison of Quantization Techniques

Multiple ways to quantize a network with different impact:

Quantization scheme	Memory bandwidth reduction (Weights)	Memory bandwidth reduction (Activations)	Compute Speedup	Notes
Weight only quantization to int8	4x	1x	1x	Suitable for embedding lookups
Dynamic quantization	4x	1x	2x	Suitable for compute bound layers
Static quantization (int32 accumulators)	4x	4x	2x	Suited for all layers, important for convolutions
Static quantization (int16 accumulators)	4x	4x	4x	Requires lower precision weights/activations

Results

Post Training Quantization Results

	fp32 accuracy	int8 accuracy change	Technique	CPU inference speed up
ResNet50	76.1 Imagenet	-0.2 75.9	Post Training	2x 214ms →102ms, Intel Skylake-DE
BERT	90.2 F1 (GLUE MRPC)	0.0 90.2	Dynamic Quantization with per-channel quantization	1.6x 581ms →313ms, Intel Skylake-DE, Batch size=1

Quantization Aware Training

- Quantization aware training provides the best accuracy and allows for simpler quantization schemes.

	fp32 accuracy	int8 accuracy change	Technique	CPU Inference speedup
MobileNetV2	71.9 Imagenet	-6.3 65.6	Post Training: Per Tensor quantization	4x 75ms →18ms OnePlus 5, Snapdragon 835
MobileNetV2	71.9 Imagenet	-4.8 67.1	Post-Training: Per-channel quantization	
MobileNetV2	71.9 Imagenet	-0.4 71.5	Quantization-Aware Training: Per Tensor Quantization	

Recipe for Quantizing a Model

Recipe(1)

- Profile your model first:
    - Latency: Dynamic quantization (CPU)
    - Model size: Dynamic/Static quantization
    - Code size: Choose appropriate framework
    - Power: Static quantization, custom hardware (NPU)
    - Accuracy: Quantization aware training (if needed)
- Iterate on architecture: Exponential tradeoff between complexity and accuracy

Recipe(2)

- Fuse: Optimize your model and kernels
  - Fuse operations as much as possible:
    - Conv-Relu, Conv-Batch norm and more
- Quantize and check the accuracy
    - If accuracy is not sufficient
      - Try Quantization aware training
      - Selectively quantize parts of model

Learn More

Resources

https://pytorch.org/docs/stable/quantization.html

https://www.tensorflow.org/model_optimization

Tutorials

https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html

Quantized models for download

https://github.com/pytorch/vision/tree/master/torchvision/models/quantization

2020 Embedded Vision Summit

“Practical DNN Quantization Techniques and Tools”

9/15/2020