Introduction to Quantized Neural Network

A Software-Hardware Codesign Perspective

zsc@megvii.com

Jul. 2022

周舒畅

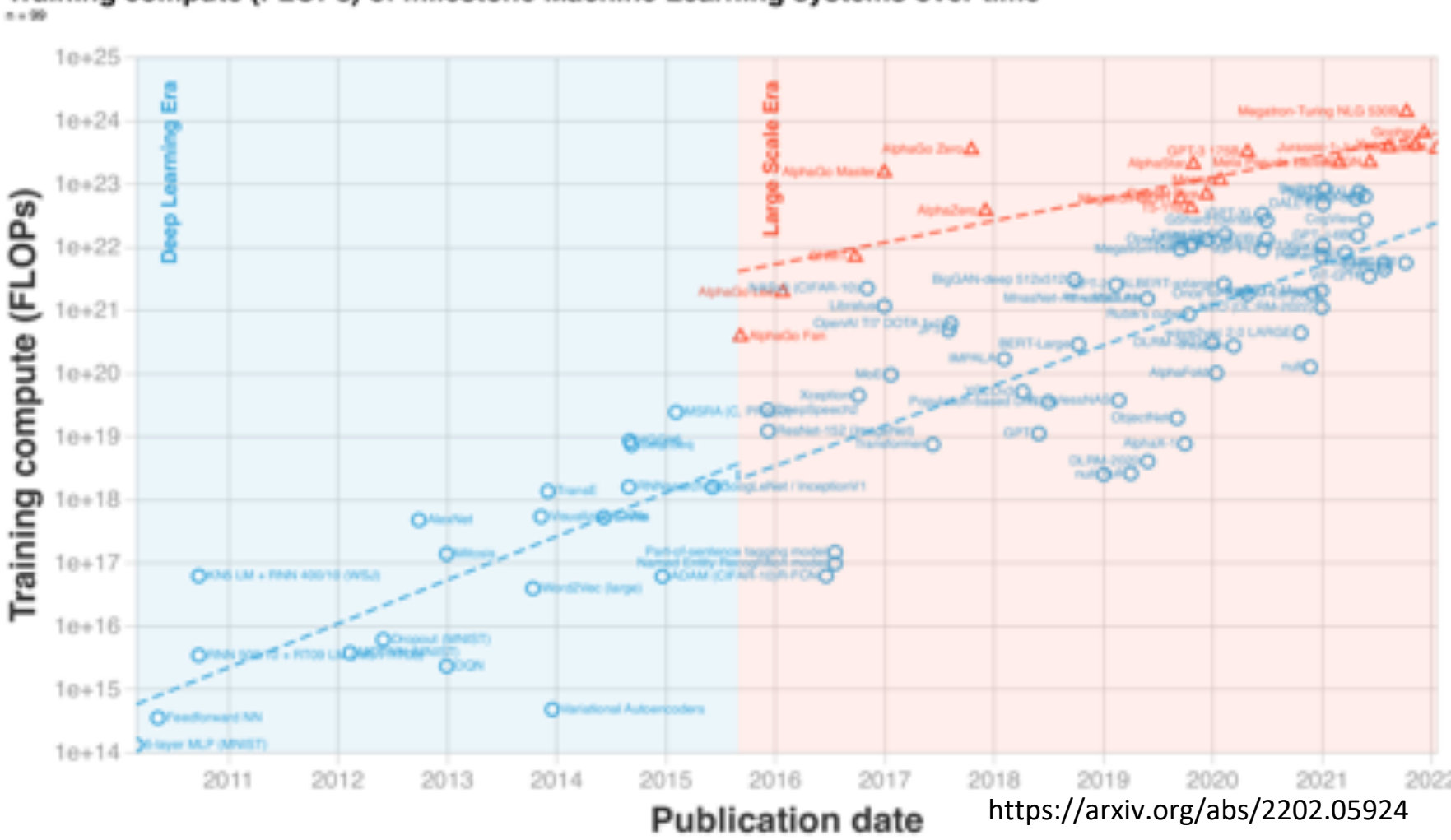
旷视研究院 AI 计算组组长；2000 年入学清华电子系，博士毕业于中科院计算所。主要工作包括 Dorefa-Net、 EAST、 RIFE。

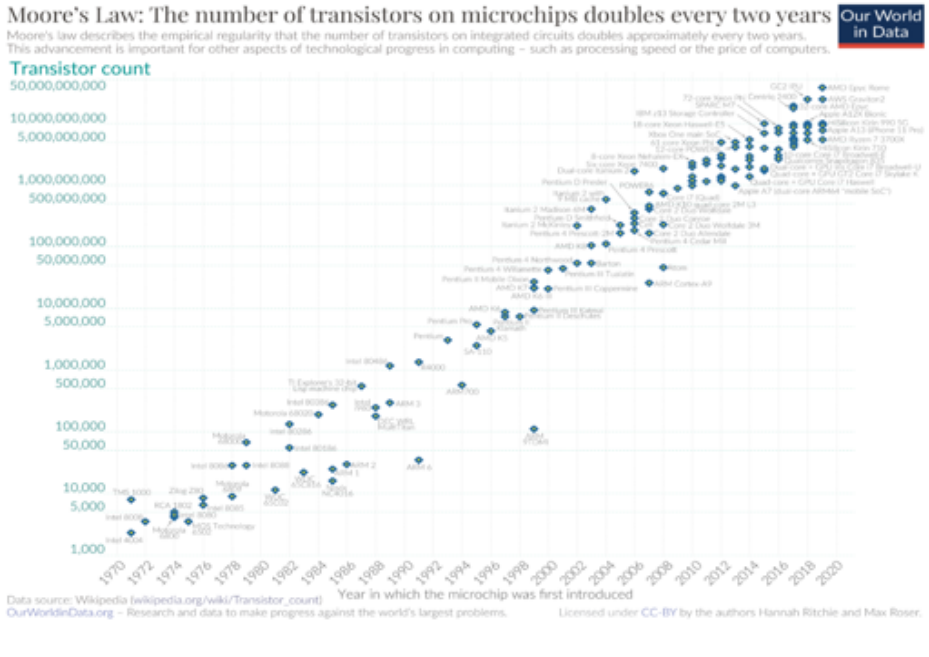
曾获得

NeurIPS2021 ML4CO DualTrack 第一名，

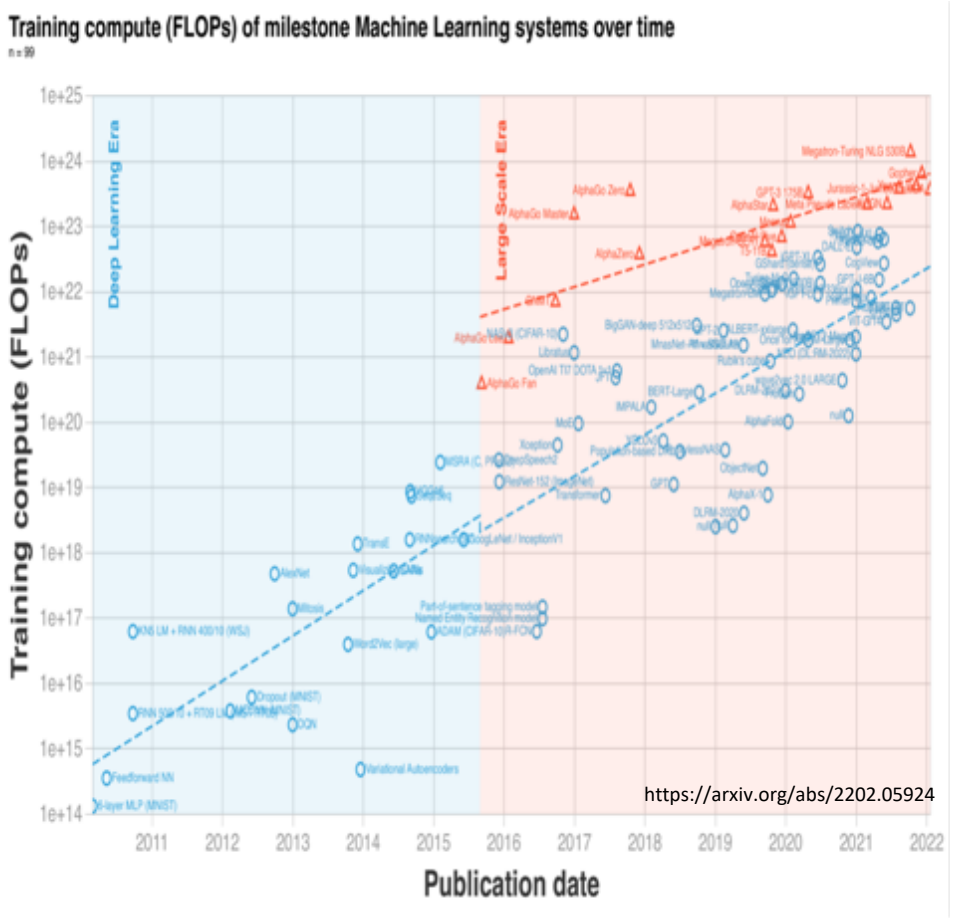
NeurIPS2017 Learning to Run Challenge 第二名，

美国国家标准技术研究院 TRAIT 2016 OCR 冠军。


Training compute (FLOPs) of milestone Machine Learning systems over time
Deep Learning Era
Large Scale Era
Training compute (FLOPs)
1e+25
1e+24
1e+23
1e+22
1e+21
1e+20
1e+19
1e+18
1e+17
1e+16
1e+15
1e+14
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
Publication date
https://arxiv.org/abs/2202.05924



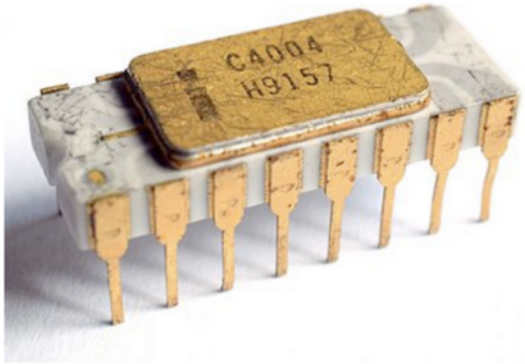
50 years of computing power growth



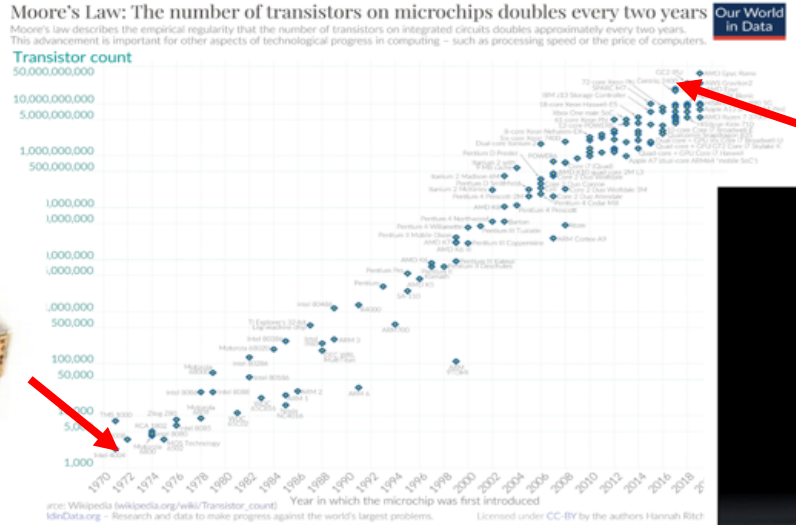
12 years of computing power demand growth

Better Algorithm = More Computing Power

- Many techniques exist, but will focus on Quantized Neural Network in this talk.



Intel® 4004, released 1971
1.1 Kops @ ??



Both supports 4-bit

Tesla® T4, released 2018
260 Tops @ 75 Watt

Quantized Neural Network can help

- Faster Inference
  - Meet latency requirements
    - Augmented Reality
    - Auto Driving
  - No dropping frames
- Faster Training
  - Higher algorithm evolution speed
  - In-time incorporation of new data
- Lower Resource Requirements
  - saves storage size
  - reduces memory footprint (external memory)

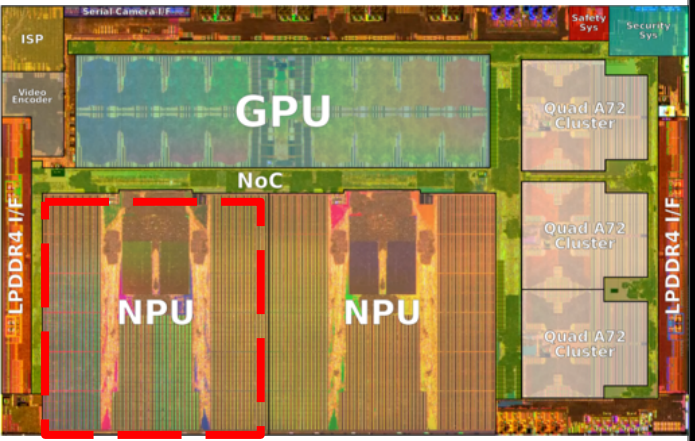
Software

- Lower Power
  - Allows more compact packaging and less peripherals
  - Easier cooling
- Higher Performance
  - Simpler pipelines allows higher frequency
- Lower Cost (smaller chip area)
  - Reduce #MAC
  - Reduce cache/on-chip memory
  - Be P&R friendly

Hardware

Most AI-Chip's are now equipped with int8 capability

Tesla FSD chip,
260 mm^2 die size



36T int8
@2GHz

350 mm^2 die size

16 Lane CSI
109 Gbps CPHY 1.1
1Gb Ethernet

DLA
5.7 TFLOPS FP16
11.4 TOPS INT8

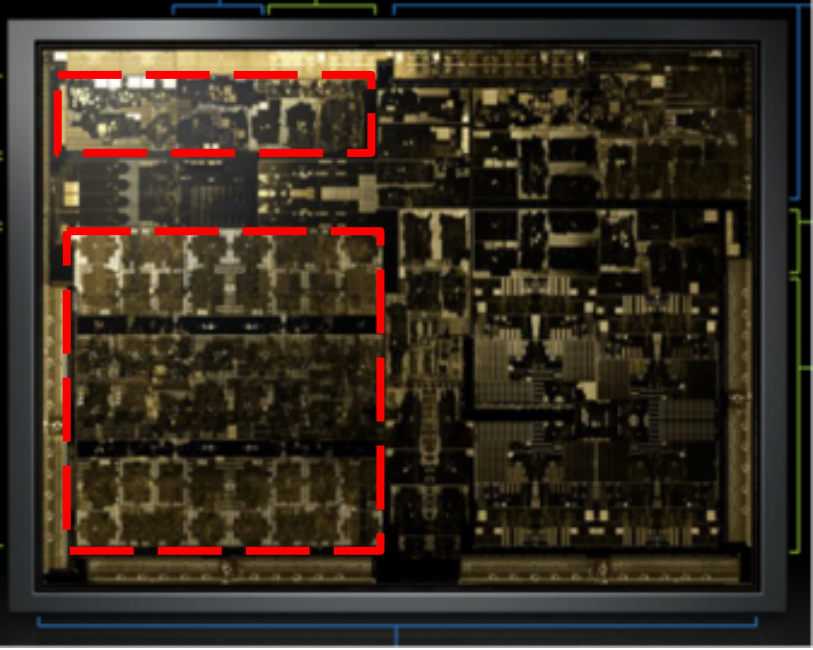
Multimedia Engines
1.2 GPIX/s Encode
1.8 GPIX/s Decode
4 GPIX/s Video Image Compositor

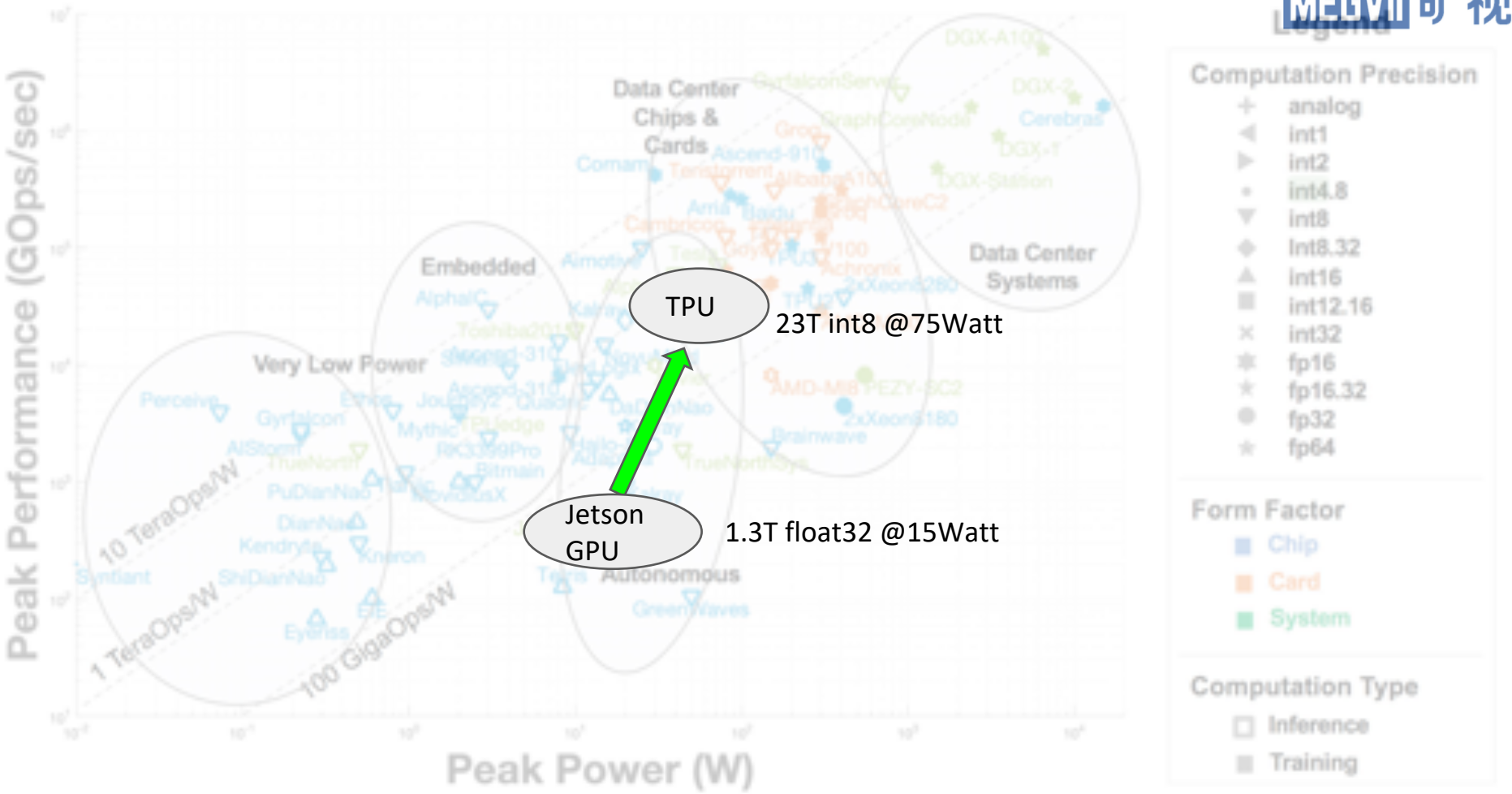
Vision Accelerator
1.7 TOPS

Stereo & Optical Flow Engine
2x 3.1 TOPS

Volta Tensor Core GPU
FP32 / FP16 / INT8 Multi-Precision
512 CUDA Tensor Cores
2.8 CUDA TFLOPS (FP16)
22.6 Tensor Core DL TOPS

23T int8 @1.1GHz





https://arxiv.org/pdf/2009.00993.pdf

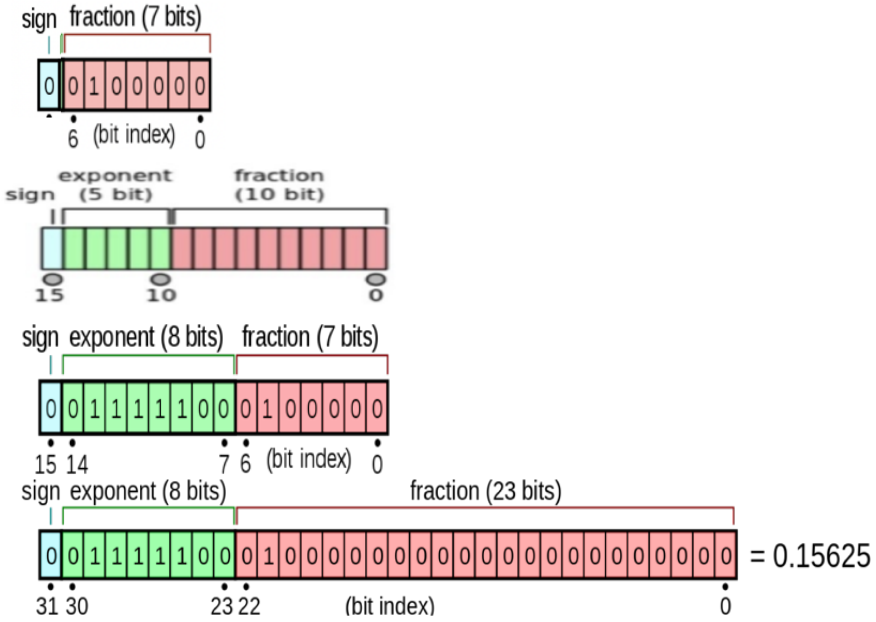
Benefits of going from float32 to int8

int8

float16

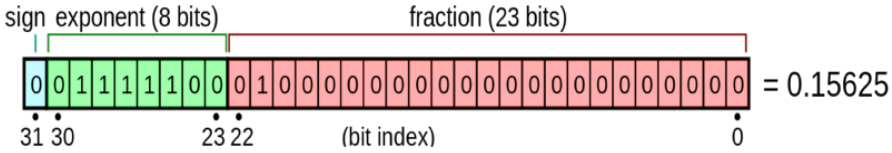
bfloat16

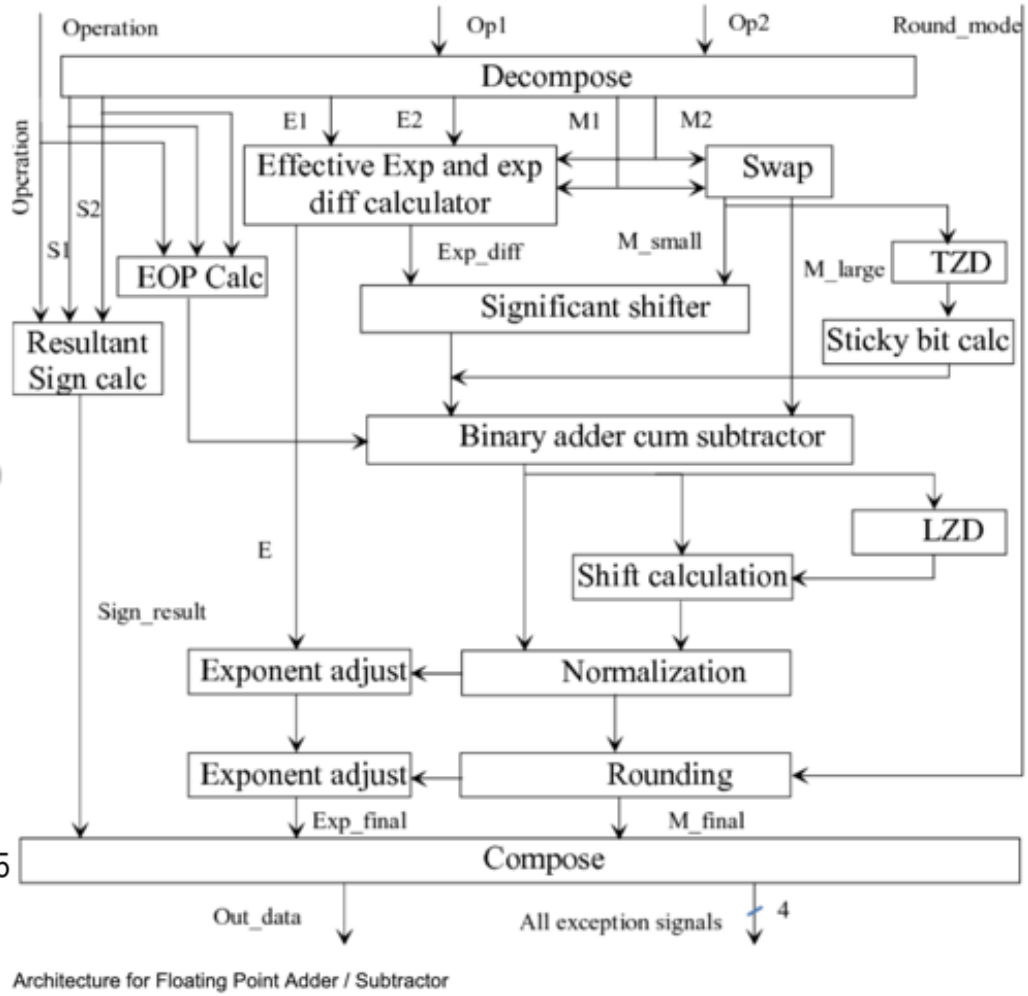
float32



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 Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

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 Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640





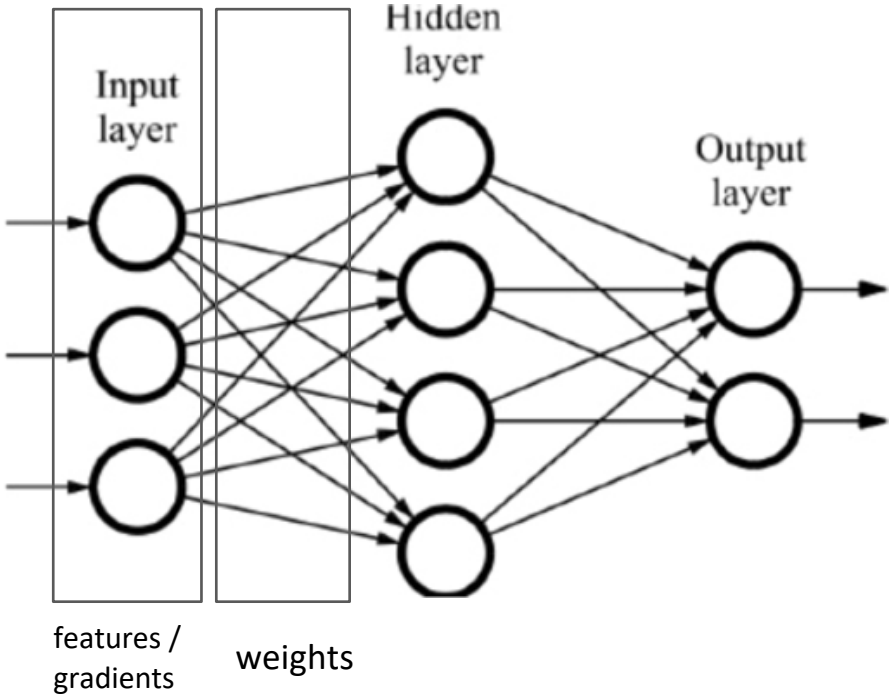
Architecture for Floating Point Adder / Subtractor

Floating point numbers are complex

https://ieeexplore.ieee.org/document/7208116

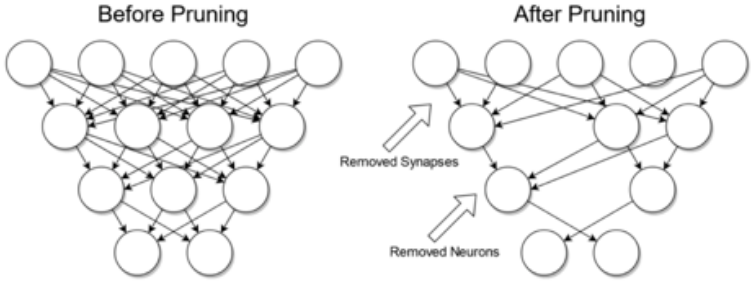
Quantized Neural Network

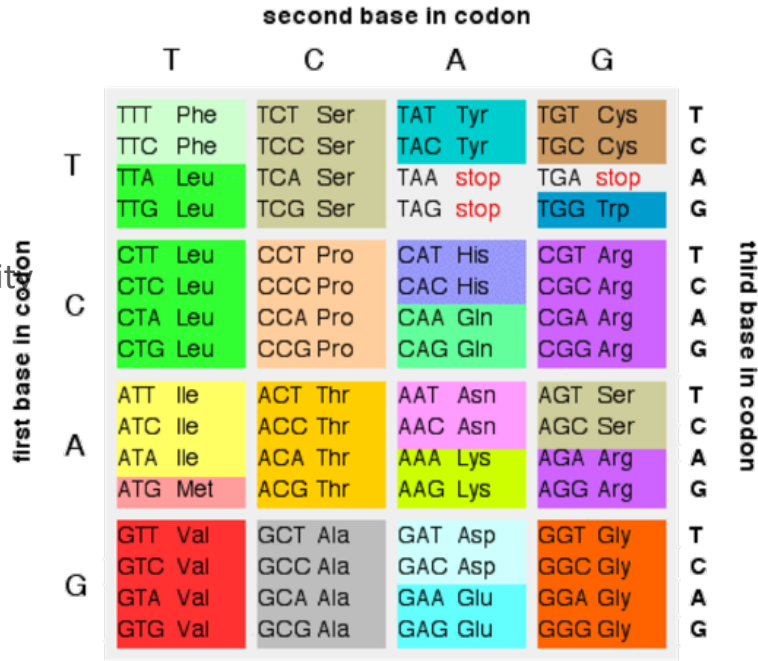
- Reduce computation / IO by quantization of weights, features and gradients



Beyond Int8: Quantization is not only about speed

- Quantization is a special form of Discretization
  - DNA is loosely using 6-bit encoding (compressible)
- For NN, Quantization is a soft Pruning
  - Quantization can help understand model redundancy
  - Pruning model (like a Decision Tree) helps its interpretability
- Need to go below 8-bit quantization!



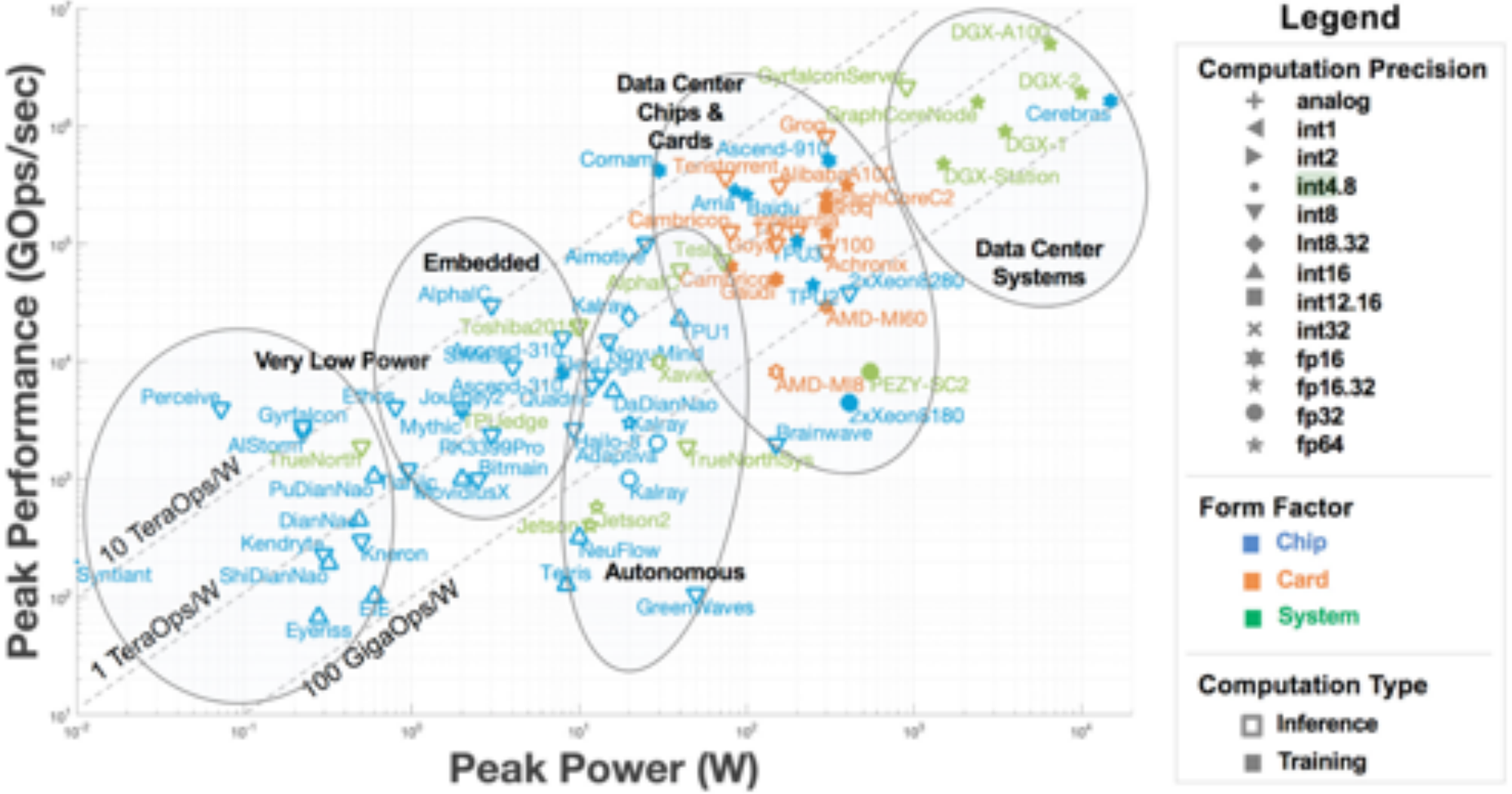


Go below int8 to gain more

- int4:
  - symmetric: {-15, -13 ... , 15}, asymmetric: {-8, -7, ... 7}
- int2:
  - symmetric: {-3, -1, 1, 3}, asymmetric: {-2, -1, 0, 1}
- Ternary: {-1, 0, 1}
- Binary
  - weight: {-1, 1}, feature {0, 1}
- Float8

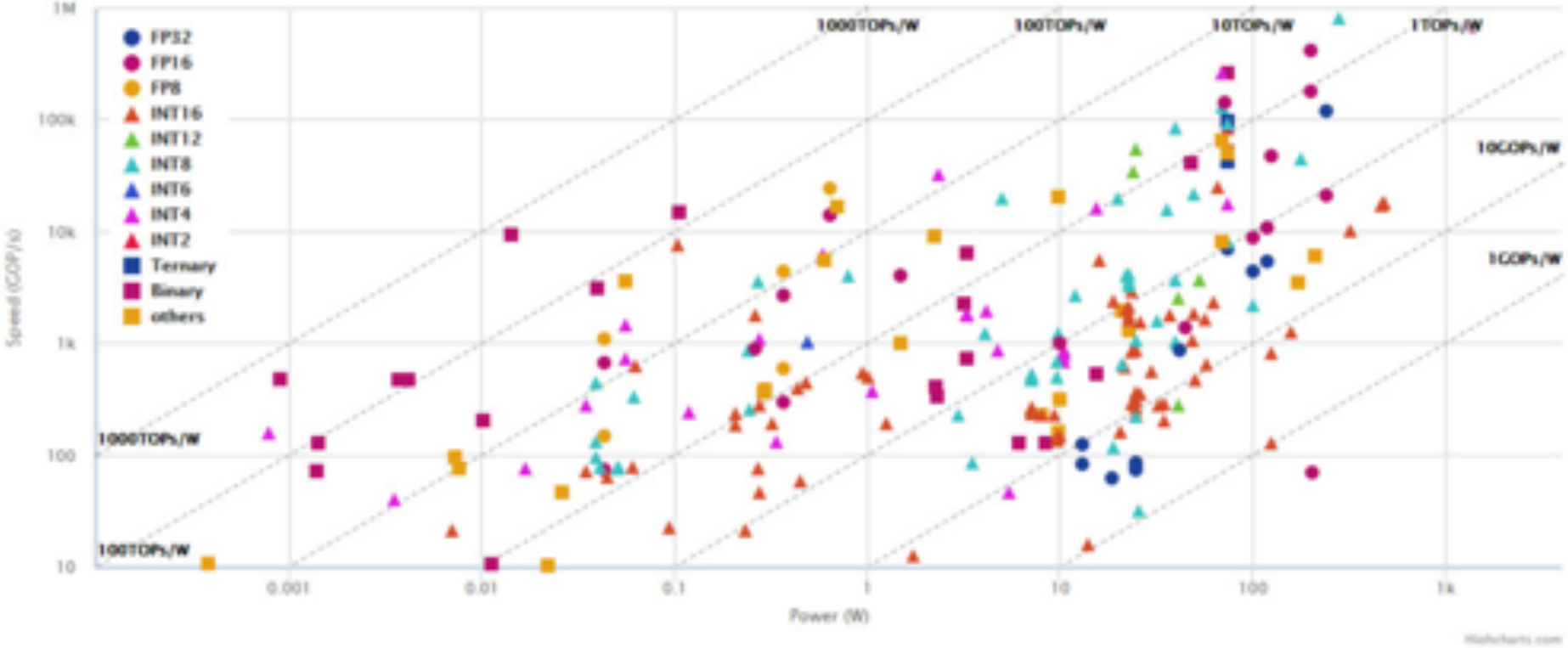
Table 3: Tesla configurable 8-bit Floating Point formats

Format	Sign bit?	No. of Mantissa bits	No. of Exponent bits	Exponent Bias Value
CFloat8_1_4_3	Yes	1 + 3	4	Unsigned 6-bit integer
CFloat8_1_5_2	Yes	1 + 2	5	Unsigned 6-bit integer



https://arxiv.org/pdf/2009.00993.pdf

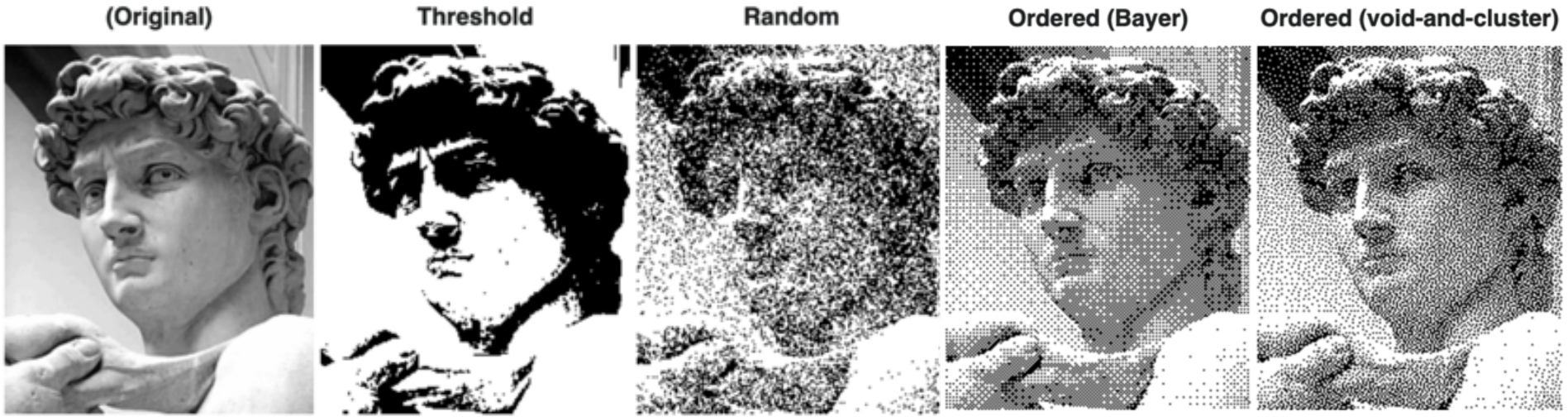
High TOPS/W usually due to low-bit number support



Source: https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/

But going below int8 is hard

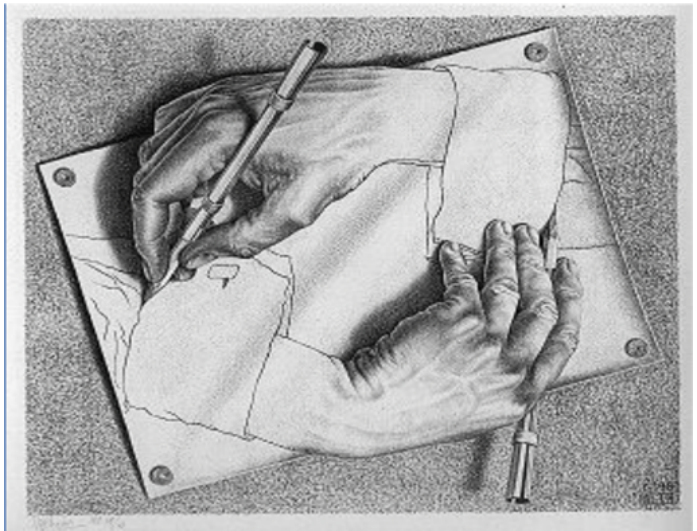
- Severe numerical errors leads to Accuracy Degradation



https://en.wikipedia.org/wiki/Dither

Challenges of Quantized Neural Network

- Algorithmic Challenge: maintaining accuracy with decreasing bit-width
  - Dynamic Quantization (DQ)
  - Post Training Quantization (PTQ)
  - Quantization Aware Training (QAT)
- Hardware Challenge: design efficient architecture to accelerate QNN
  - Multiple bit-width support
  - Toolchain support
  - User adoption scheme



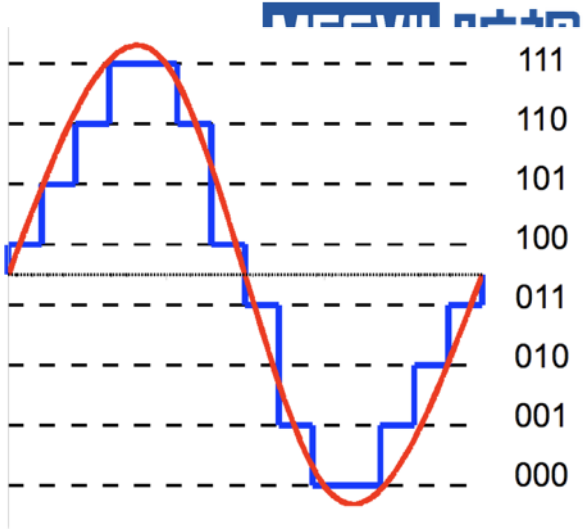
Quantized Neural Network: Algorithm

- Basics of Quantization
- Dynamic Quantization (DQ) and Post Training Quantization (PTQ)
- Quantization Aware Training (QAT)
  - Zero Gradient Problem and Straight Through Estimators
  - Non-uniform Quantization (fp8 and LUT)
  - Balanced Quantization and Maximum Entropy
  - Quantization of Gradients and Stochastic Rounding
- Power law of Neural Network Capacity and BitOp metric of QNN

Quantization

- Linear Quantization
  - Essentially rounding
  - Con: dynamic range of floating point numbers are problematic

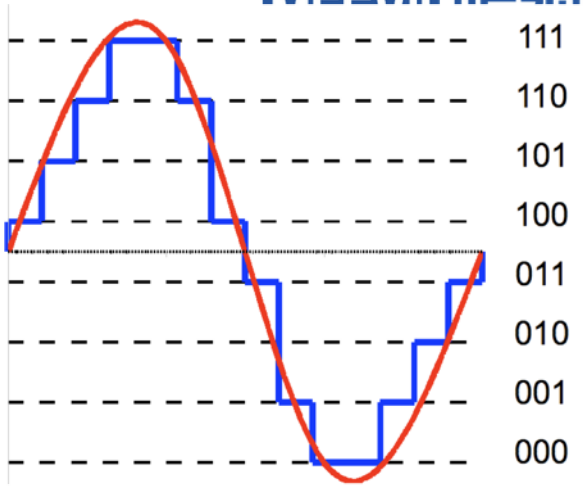
$$Q(x) = \Delta \cdot \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor$$



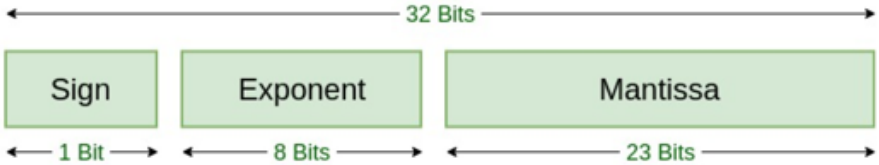
Quantization

- Linear Quantization
  - Essentially rounding
  - Con: dynamic range of floating point numbers are problematic

$$Q(x) = \Delta \cdot \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor$$



IEEE float32



$$\mathbf{quant}_k : [-1, 1] \rightarrow [-1, 1]$$

$$\text{uniform-quant}_k(\mathbf{W}) \stackrel{\text{def}}{=\!=} \max(|\mathbf{W}|)\, \mathbf{quant}_k\left(\frac{\mathbf{W}}{\max(|\mathbf{W}|)}\right)$$

$$\approx \mathbf{W}$$

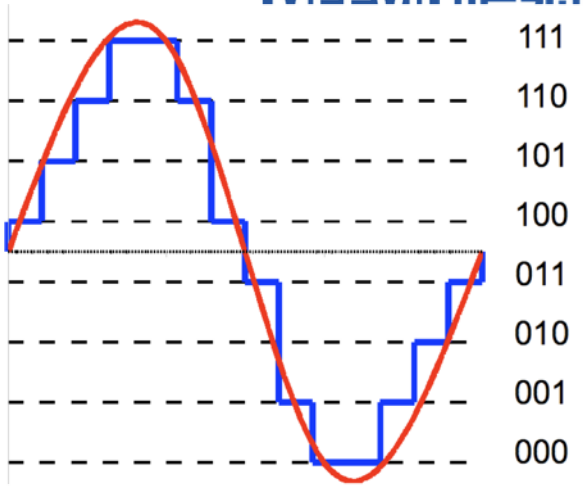
Floating Point Scaling Factor

Integer

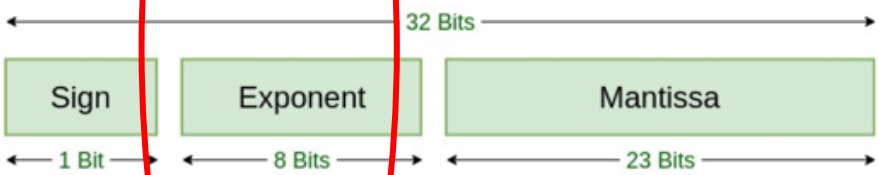
Quantization

- Linear Quantization
  - Essentially rounding
  - Con: dynamic range of floating point numbers are problematic

$$Q(x) = \Delta \cdot \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor$$



IEEE float32



$$\mathbf{quant}_k : [-1, 1] \rightarrow [-1, 1]$$

$$\text{uniform-quant}_k(\mathbf{W}) \stackrel{\text{def}}{=} \max(|\mathbf{W}|)\, \mathbf{quant}_k\left(\frac{\mathbf{W}}{\max(|\mathbf{W}|)}\right)$$

$$\approx \mathbf{W}$$

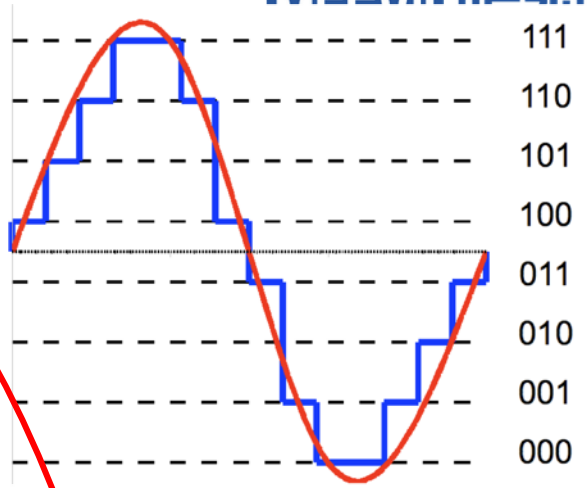
Floating Point Scaling Factor

Integer

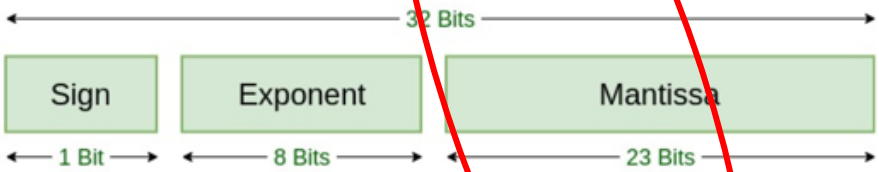
Quantization

- Linear Quantization
  - Essentially rounding
  - Con: dynamic range of floating point numbers are problematic

$$Q(x) = \Delta \cdot \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor$$



IEEE float32



$$\mathbf{quant}_k : [-1, 1] \rightarrow [-1, 1]$$

$$\text{uniform-quant}_k(\mathbf{W}) \stackrel{\text{def}}{=\!=} \max(|\mathbf{W}|)\, \mathbf{quant}_k\left(\frac{\mathbf{W}}{\max(|\mathbf{W}|)}\right)$$

$$\approx \mathbf{W}$$

Floating Point Scaling Factor

Integer

Scale stats and Symmetric Quantization of Neural Network

$\boldsymbol{W}$: weight matrix

$\boldsymbol{X}$: input of a layer

Symmetric quantization

$$\boldsymbol{WX} \approx s_W(\boldsymbol{W}_{\text{int}})\, s_X(\boldsymbol{X}_{\text{int}})$$
$$= s_W s_X(\boldsymbol{W}_{\text{int}}\, \boldsymbol{X}_{\text{int}})$$

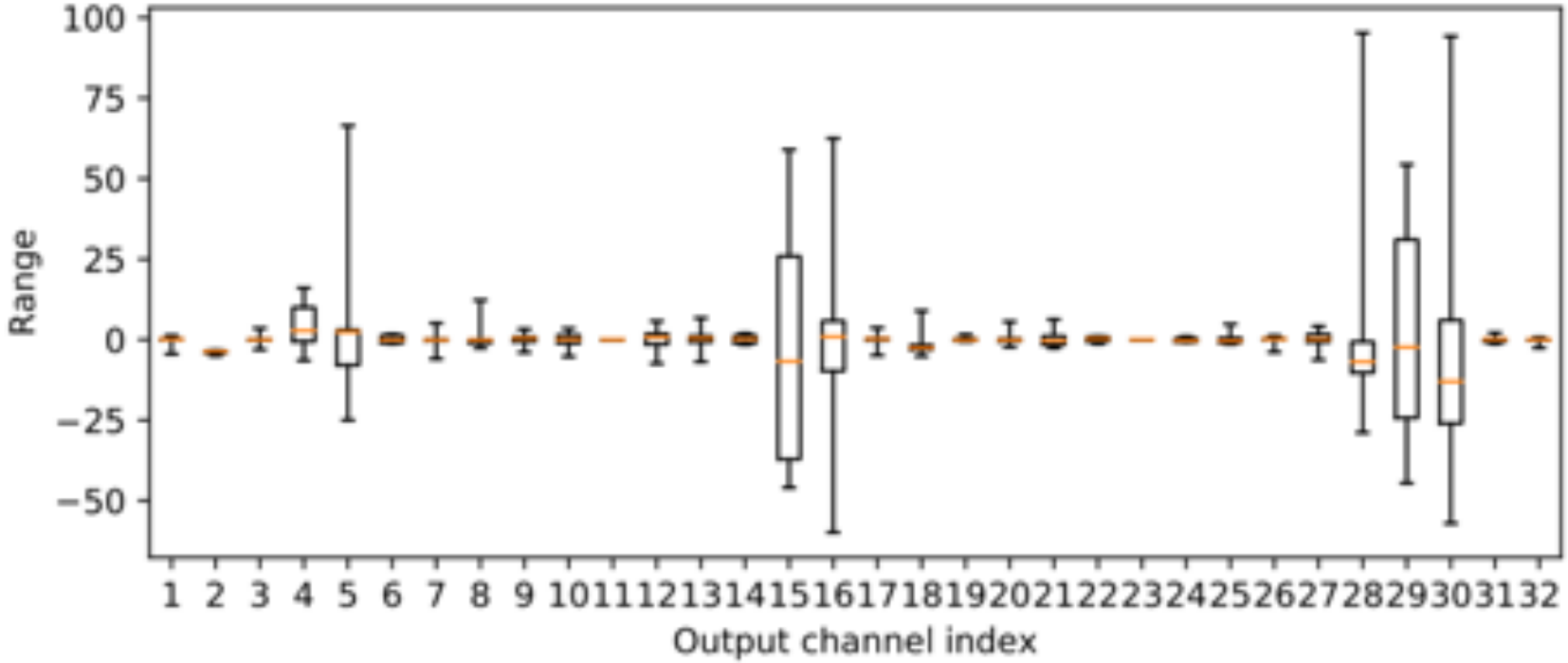
- Since weight scale can be absorbed with feature scale, will refer generally as a combined "scale" hereafter.

- Weight stats can be computed offline for normal layers like convolutions.
- Feature stats (stats of X) may be collected on-the fly (DQ), or in "calibration" stage (PTQ) on the "calibration" dataset (discuss later).

https://www.youtube.com/watch?v=KASuxB3XoYQ

Per-channel asymmetric quantization is necessary when required by data distribution

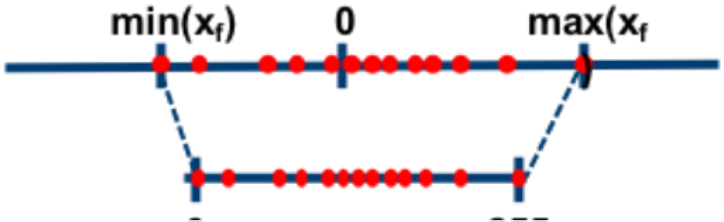
https://arxiv.org/abs/1906.04721 a layer in mobilenet-v2



Asymmetric uniform quantization

- Two equivalent forms
  - min, max
  - scale, zero-point

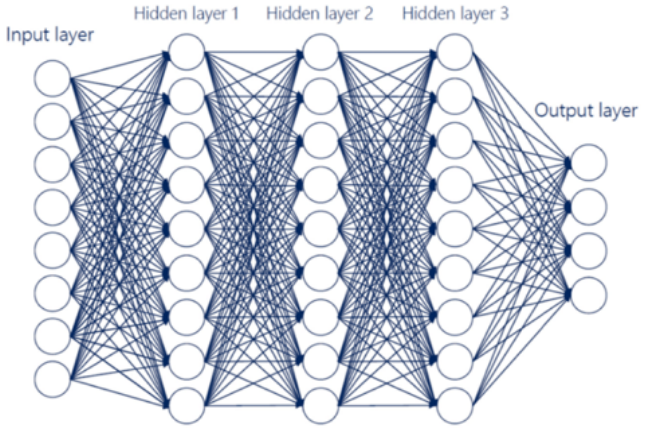
asymmetric (with zero-point):



Asymmetric quantization

$$\boldsymbol{WX} \approx s_W(\boldsymbol{W}_{\text{int}} - z_W)\, s_X(X_{\text{int}} - z_X)$$
$$= s_W s_X(\boldsymbol{W}_{\text{int}}\boldsymbol{X}_{\text{int}}) + s_W s_X z_X \boldsymbol{W}_{\text{int}} + s_W z_W s_X z_X + s_W s_X z_W \boldsymbol{X}_{\text{int}}$$

Quantization workflows for whole Neural Networks (PyTorch)



WORKFLOWS

	Quantization	Dataset Requirements	Works Best For	Accuracy
Dynamic Quantization	weights only		small batch LSTMs and MLPs	good
Post Training Quantization	weights and activations	calibration	all	good
Quantization-Aware Training	weights and activations	fine-tuning	all	best

Maybe Little speedup

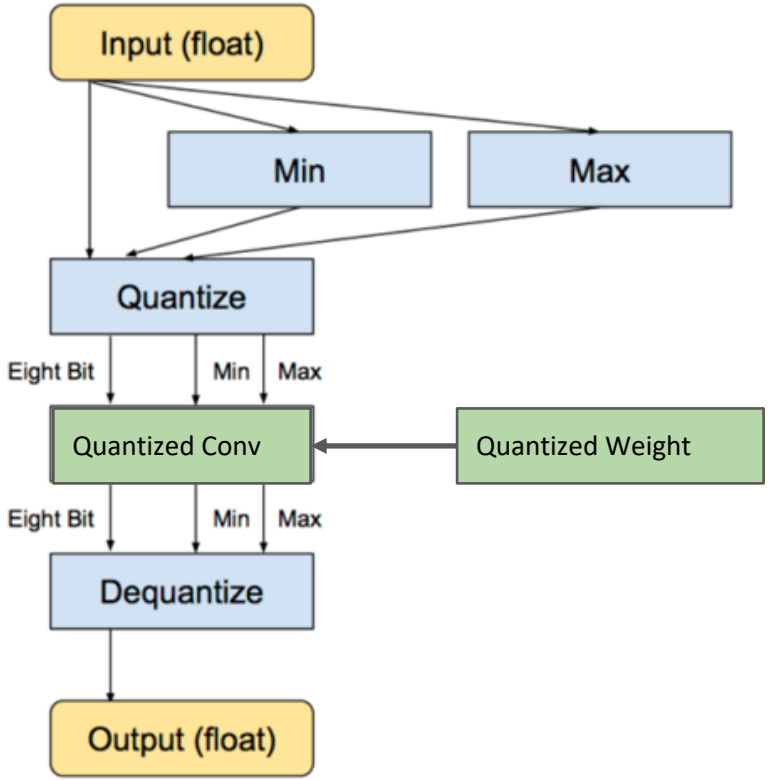
Generally at least int8 bitwidth

Tuning with data labels required

Dynamic Quantization

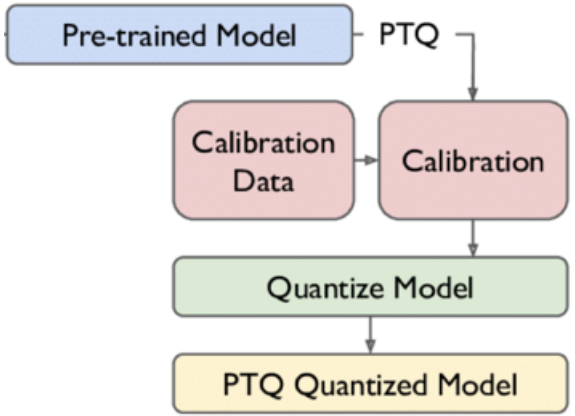
- Weights are quantized offline
- For features, compute stats and quant/dequant at runtime
  - Can exploit dedicated MAC for quantized numbers
  - Saves storage, memory footprint and memory bandwidth

https://leimao.github.io/blog/PyTorch-Dynamic-Quantization/
FP32 Model Size: 411.00 MB
INT8 Model Size: 168.05 MB



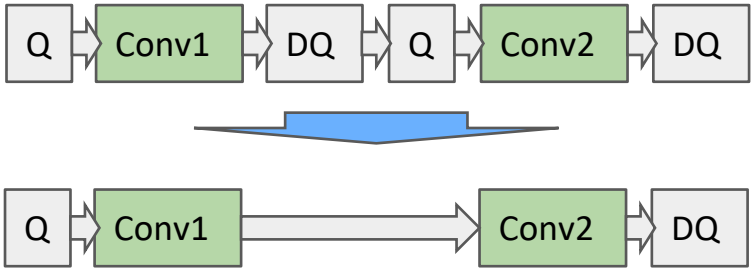
Post Training Quantization (PTQ)

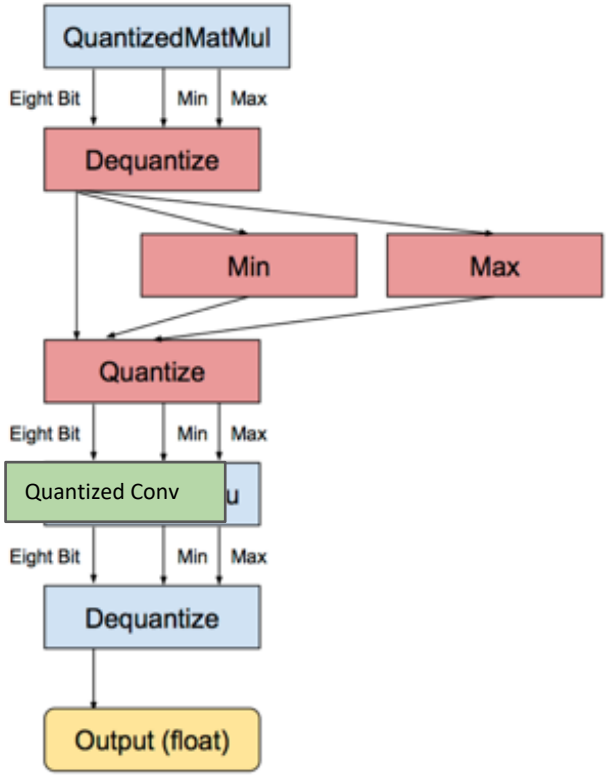
- Introduce a calibration stage for collecting stats of features.
- The calibration set need be I.I.D. with the test set.
  - Need weak generalization between calibration set and the test set.
  - Having 100's of images covering different classes of an ImageNet model is beneficial.



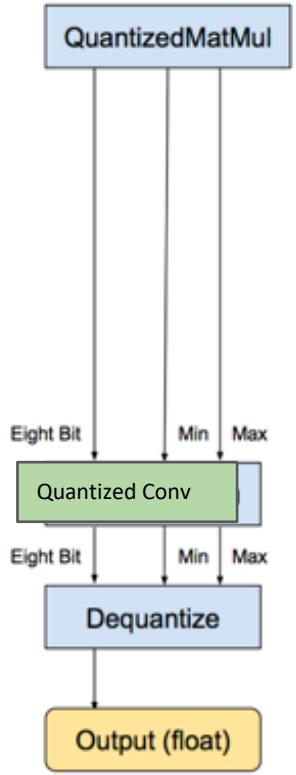
PTQ and graph partitioning

- Caveat: delay Dequantize as much as possible





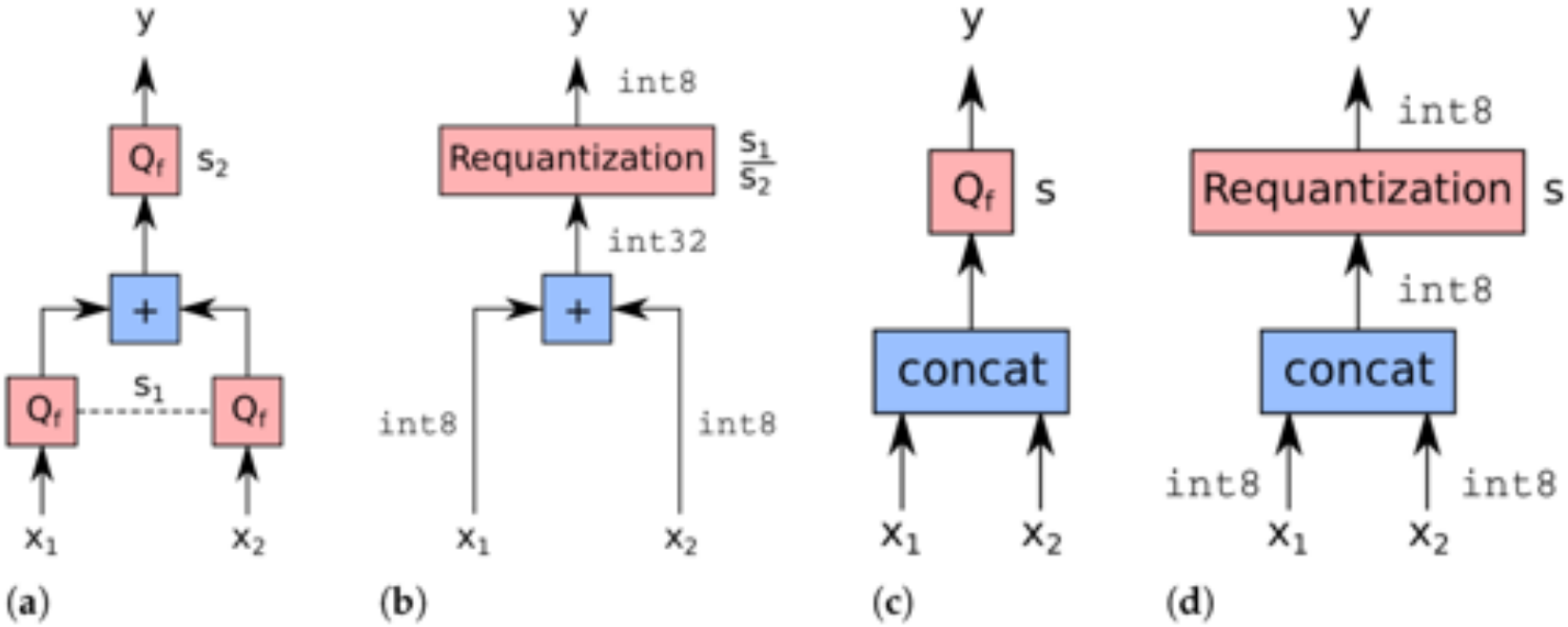
During Calibration



Inference

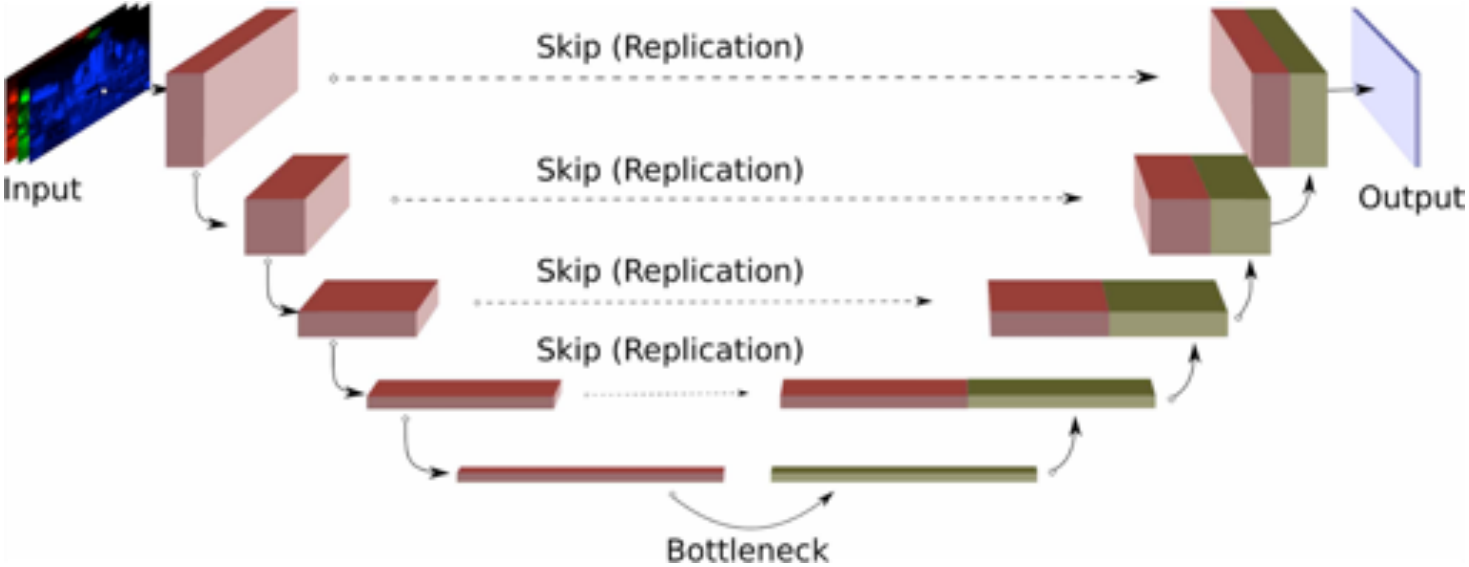
Multi-layer quantization is more difficult

Change of scale requires requantization.



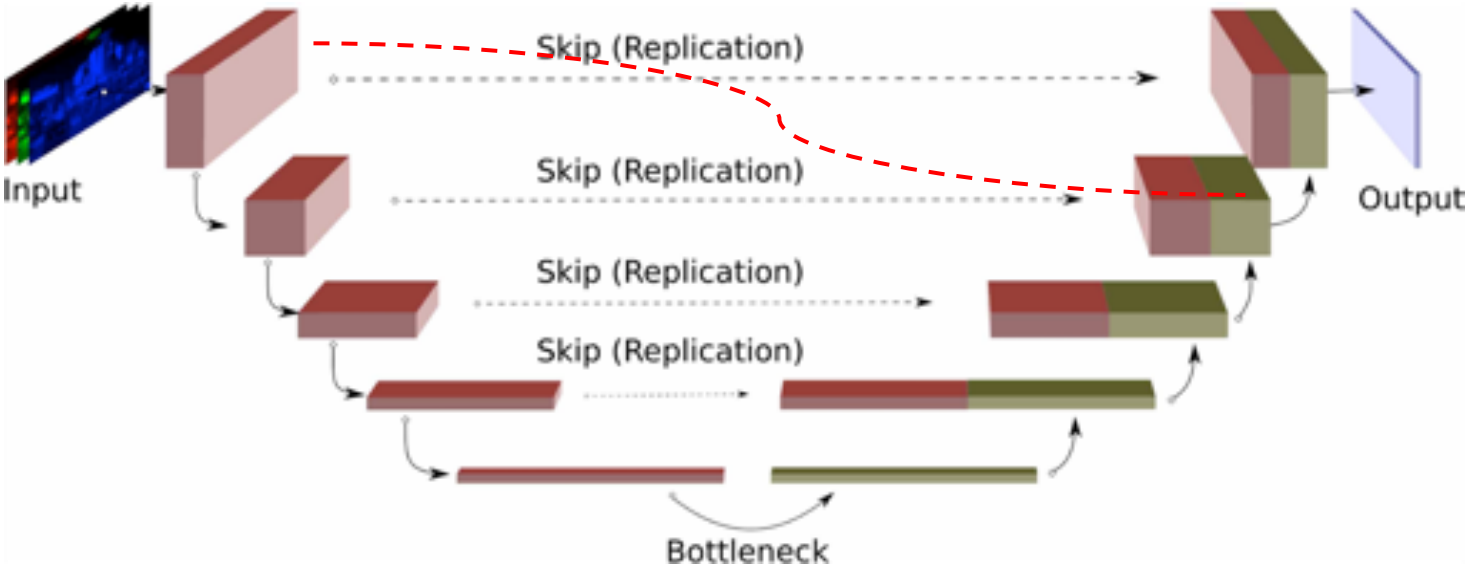
Forcing the same (scale, zero-point) is difficult

Will introduce dependencies that propagate in the computation graph.



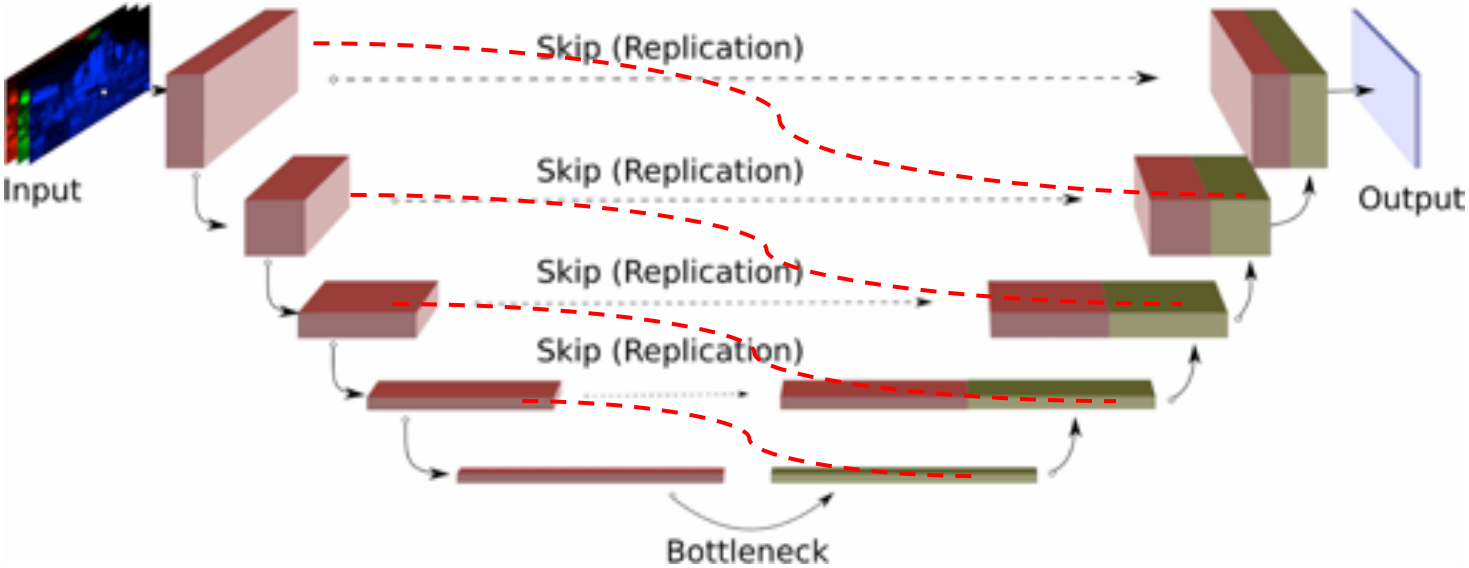
Forcing the same (scale, zero-point) is difficult

Will introduce dependencies that propagate in the computation graph.

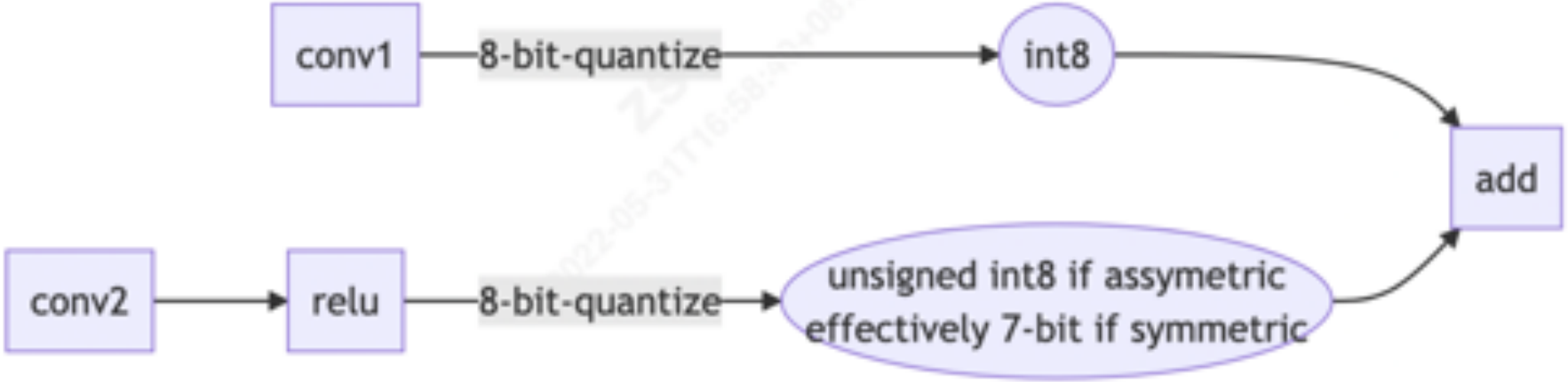


Forcing the same (scale, zero-point) is difficult

Will introduce dependencies that propagate in the computation graph.



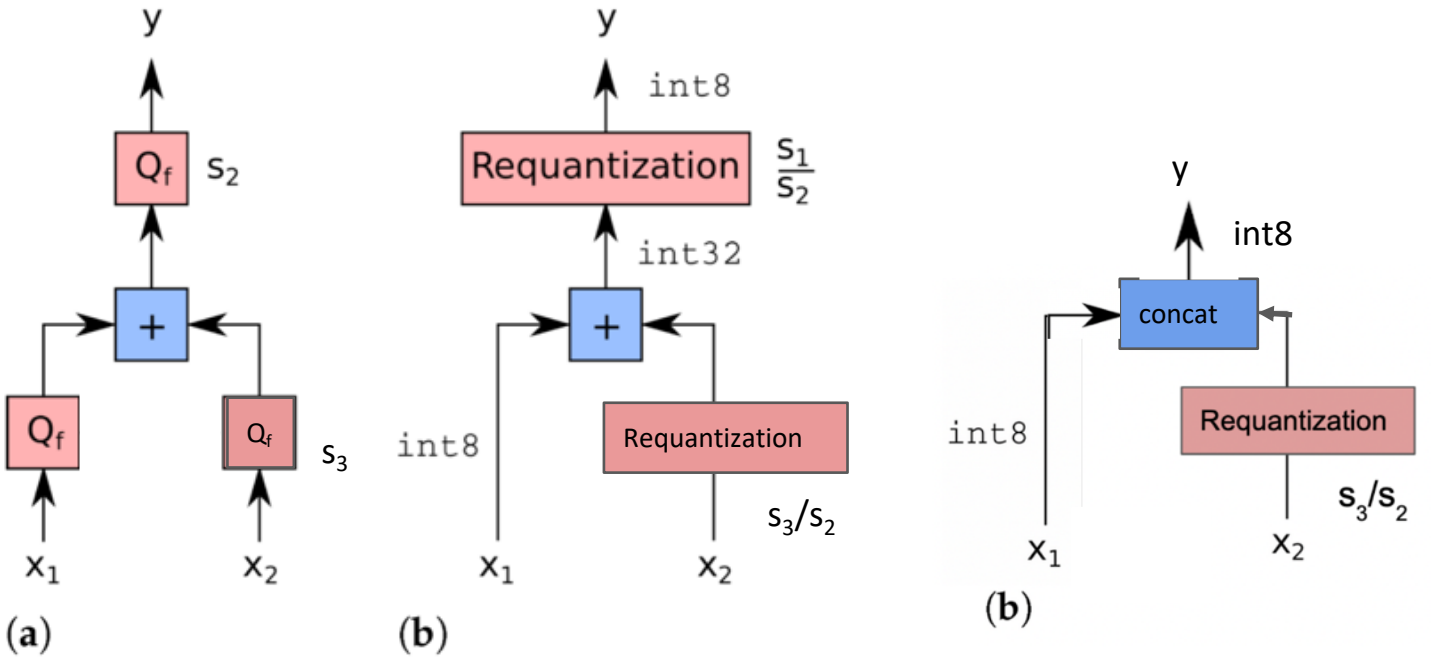
Need insert requantization or increase bit-width to unify (scale, zero-point) when adding / concatenating



addition may need be done with int16/fp32

Need insert requantization or increase bit-width to unify (scale, zero-point) when adding / concatenating

Complex scale requires lots of lossy requantization.



PTQ of a pretrained model *inevitably* degrades accuracy

- Quantization errors are inevitable, and accumulates throughout the whole network, as no knowledge of the training data is assumed.
- Requantization caused by addition/concatenation introduces additional errors and computations.
  - Concatenation / addition is ubiquitous
    - E.g. zero-padding is a form of concatenation.
    - E.g. zero-skipping is hard when zero-point ≠ 0
  - E.g. conversion between int8 and fp32 is expensive (may involve CPU, or causes intermediate data blowup)
  - Forcing the same (scale, zero-point) is difficult