Introduction to Quantized Neural Network

A Software-Hardware Codesign Perspective

zsc@megvii.com

Jul. 2022

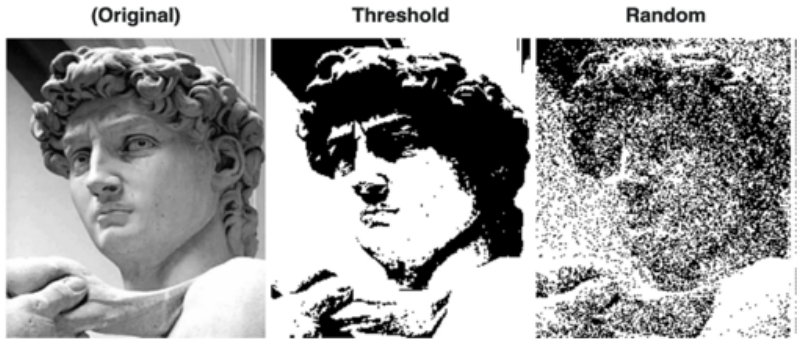
PTQ of a pretrained model inevitably degrades accuracy

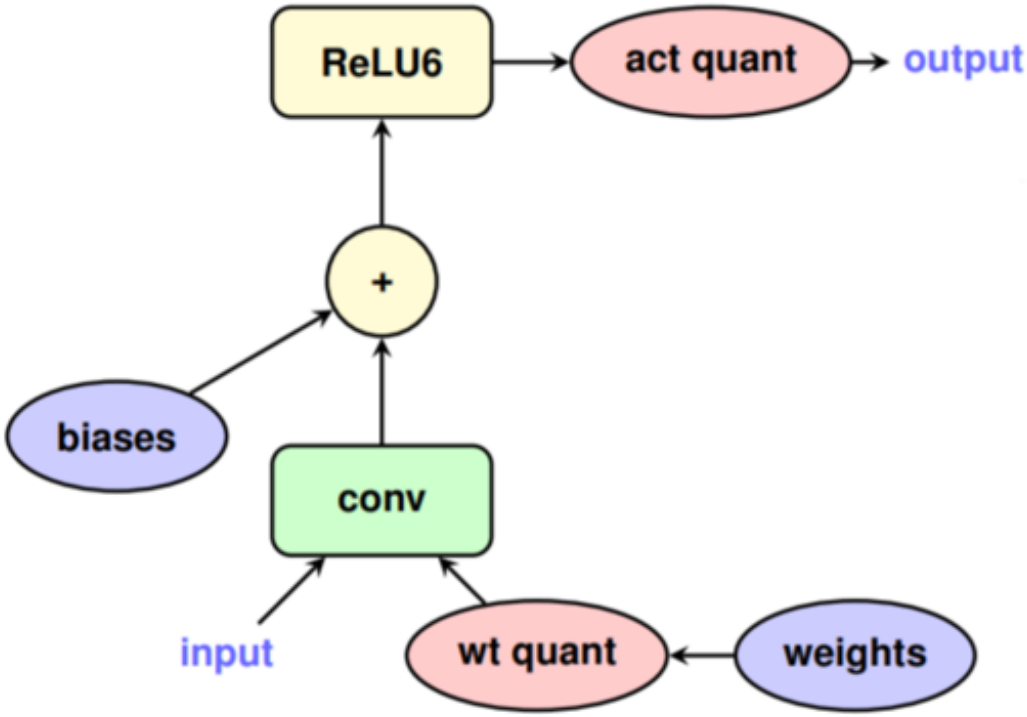
- Quantization errors are inevitable, and accumulates throughout the whole network.
- Requantization caused by addition/concatenation introduces additional errors and computations.
  - Concatenation / addition is ubiquitous
    - E.g. zero-padding is a form of concatenation.
    - E.g. zero-skipping is hard when zero-point $\neq 0$
  - 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

Compensate for the quantization error with Quantization Aware Training (QAT)

Incorporates quantization into the training process, so that model can learn to deal with it

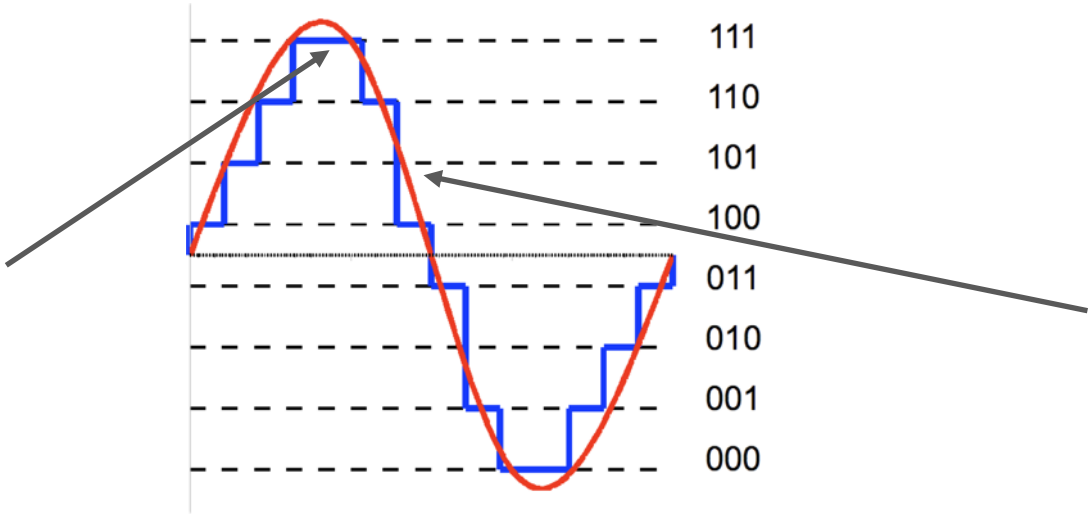
- In the spirit of Adversarial Learni
- Quantization errors are like pepper-salt noise, except with many "flats".





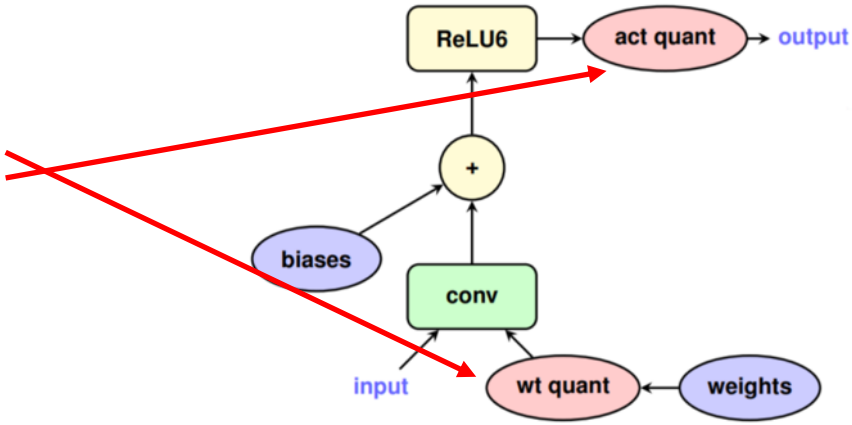
Challenges for QAT: no well defined gradients for rounding

Gradient is zero for constant values.



Gradient is undefined for the discontinuous jump.

Unable to train a Neural network with a "round" function with vanilla gradient descent.



"Differentiable" Quantization

- Bengio 13: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation
  - REINFORCE algorithm $\mathbf{g}_{\mathrm{RF}} = \left\langle \left(\sum_{k=0}^{H} \nabla_\theta \log \pi_\theta(\mathbf{u}_k|\mathbf{x}_k) \right) \left(\sum_{l=0}^{H} a_l r_l - b \right) \right\rangle$
  - Decompose binary stochastic neuron into stochastic and differentiable part

$$G(\mu, \sigma) = \mu + G(0, \sigma)$$

  - Injection of additive/multiplicative noise to emulate quantization errors
  - Straight-through estimator

Straight-through estimator

- QAT exploits “Straight-through estimator” (Hinton, Coursera lecture, 2012)

$$x \approx \hat{x}$$

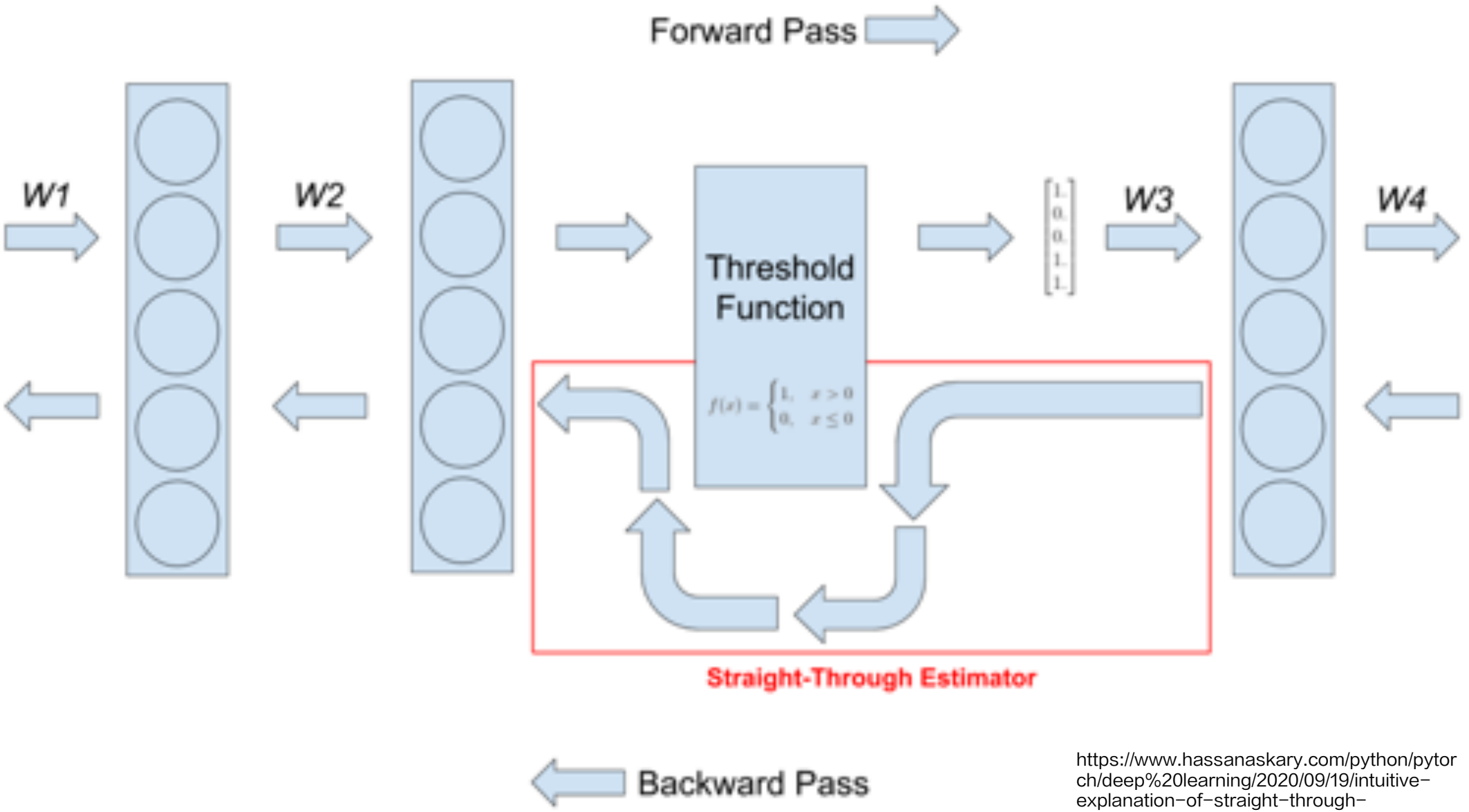
$$\Rightarrow$$

$$\frac{\partial}{\partial x} \approx \frac{\partial}{\partial \hat{x}}$$

- Example: guessing the true probability p of a "biased" coin facing up

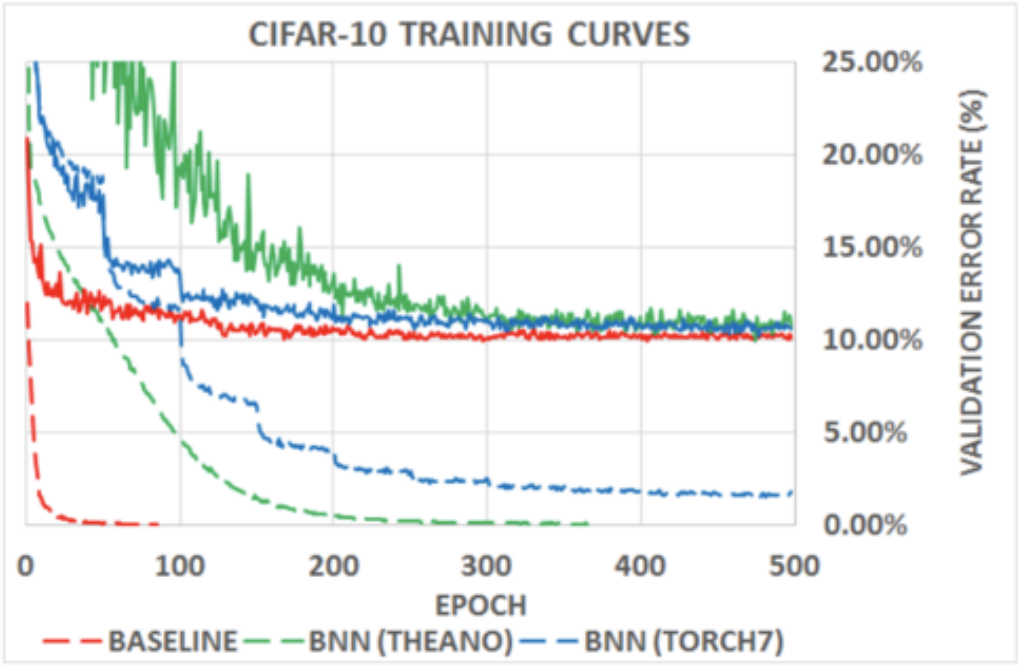
Forward: $q \sim Bernoulli(p)$ $\quad q \approx \mathbf{E}[q] = p$

Backward: $\dfrac{\partial c}{\partial p} = \dfrac{\partial c}{\partial q}.$

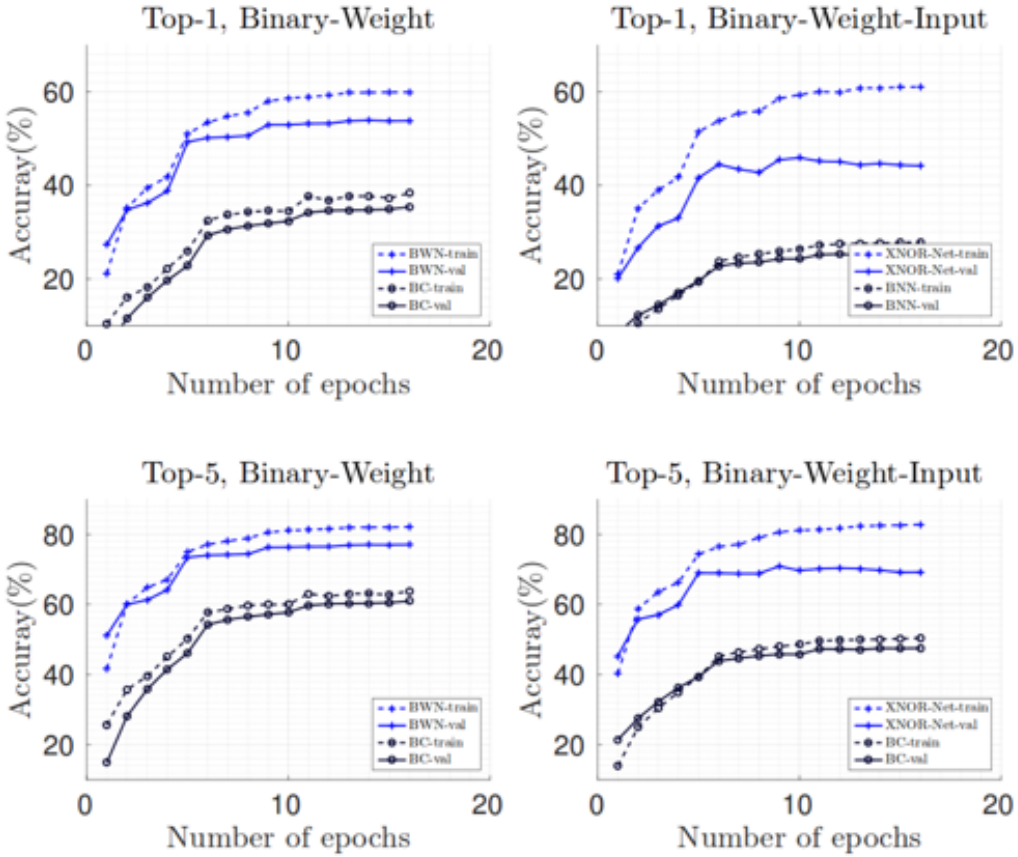


https://www.hassanaskary.com/python/pytorch/deep%20learning/2020/09/19/intuitive-explanation-of-straight-through-estimators.html

Quantized Neural Network　XNOR-Net



https://jmlr.org/papers/volume18/16-456/16-456.pdf



https://arxiv.org/abs/1603.05279

General QAT trick: make more params learnable

- Learned Step Size Quantization (arxiv/1902.08153)

$$\bar{x} = \lfloor \mathrm{clip}(x/s, -Q_N, Q_P) \rceil$$

$$\hat{x} = s \cdot \bar{x}$$

$$\frac{\partial \hat{x}}{\partial s} = \bar{x} + s \cdot \frac{\partial \bar{x}}{\partial s} = \bar{x} + s \cdot (-\frac{x}{s^2}) = \lfloor x/s \rceil - \frac{x}{s}$$

- PACT：PArameterized Clipping Activation for Quantized Neural Networks (arxiv/1805.06085)

$$y = PACT(x) = 0.5(|x| - |x - \alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases} \quad (1)$$

Closed-form solution for Scaled binarization with STE

$$\min_{\Lambda \in \text{diagonal}, B \in \{1,-1\}^{m \times n}} \|\Lambda B - W\|_F^2$$

- Solution:

$$\|\Lambda B - W\|_F^2 = \|(\Lambda B) \circ B - W \circ B\|_F^2$$
$$= \|\Lambda (B \circ B) - W \circ B\|_F^2$$
$$= \|\Lambda \mathbf{1} - W \circ B\|_F^2$$

= Variance of rows of W o B

Minimize $\|\Lambda B - W\|_F^2 \Rightarrow B = \text{sgn}(W)$

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks https://arxiv.org/pdf/1603.05279.pdf

Iterative Solution for optimal scale of multi-bit quantization

No closed-form solution, but STE works for a for-loop!

for $i = 1 \rightarrow I$ **do**

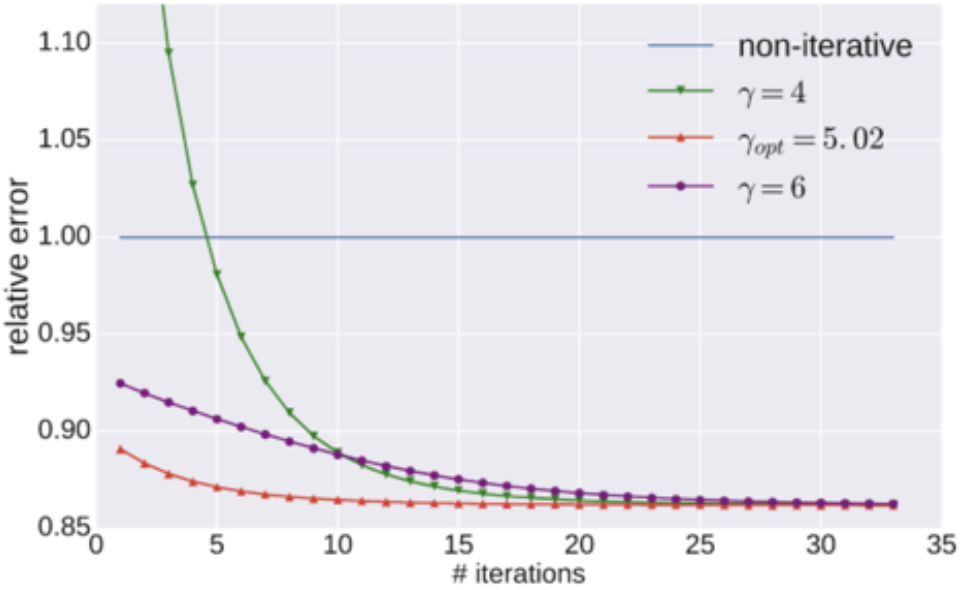
$(\mathbf{\Lambda}_t)_i \leftarrow \frac{<(\mathbf{W})_i,(\mathbf{Q}_t)_i>}{\epsilon+<(\mathbf{Q}_t)_i,(\mathbf{Q}_t)_i>}$;

// $< \cdot, \cdot >$ computes the inner product.

// $(\mathbf{W})_i, (\mathbf{Q}_t)_i$ are the i-th row of $\mathbf{W}$ and $\mathbf{Q}_t$ respectively. $(\mathbf{\Lambda}_t)_i$ is the i-th diagonal entry of $\mathbf{\Lambda}_t$.

// ϵ is a small constant thas is used to avoid division by zero.

end



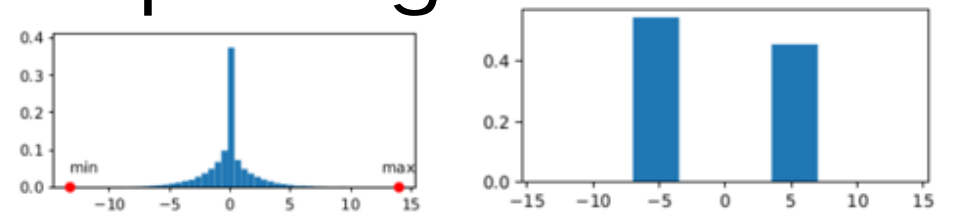
IQNN: Training Quantized Neural Networks with Iterative Optimizations

https://www.researchgate.net/profile/Taihong-Xiao-2/publication/320589547_IQNN_Training_Quantized_Neural_Networks_with_Iterative_Optimizations/links/5be506484585150b2ba90073/IQNN-Training-Quantized-Neural-Networks-with-Iterative-Optimizations.pdf?origin=publication_detail

Balanced Quantization for exploiting limited bit−width

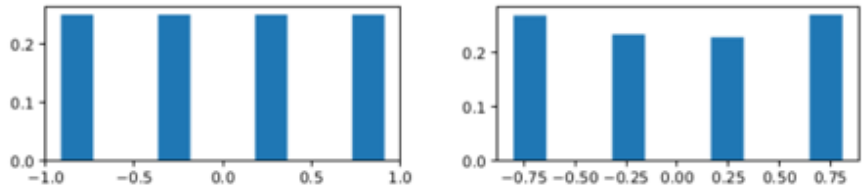
Model	weight-bits	activation-bits	PPW	
			balanced	unbalanced
LSTM	2	2	152	164
LSTM	2	3	142	155
LSTM (Hubara et al., 2016a)	2	3		220
LSTM (Hubara et al., 2016a)	4	4		100

* Our FP baseline is worse than that of Hubara.



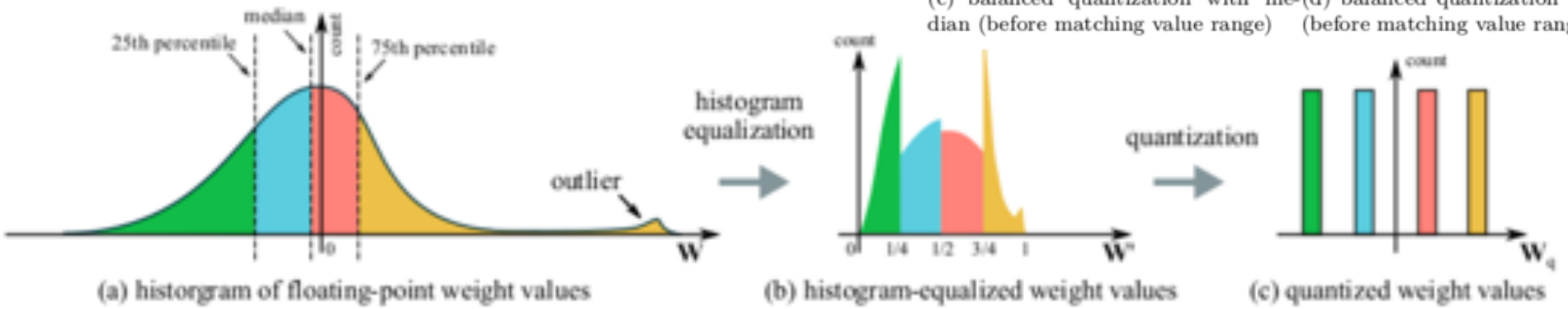
(a) floating point copy of weights in QNN after 60 epochs

(b) imbalanced quantization (no equalization)



(c) balanced quantization with median (before matching value range)

(d) balanced quantization with mean (before matching value range)



(a) histogram of floating-point weight values

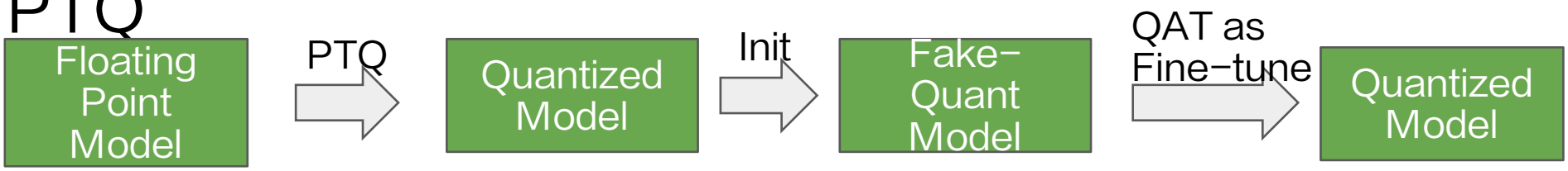
(b) histogram-equalized weight values

(c) quantized weight values

Balanced quantization https://arxiv.org/abs/1706.07145

Effective Quantization Methods for Recurrent Neural Networks https://arxiv.org/abs/1611.10176

Best of both worlds: QAT as a Fine-tuning step after PTQ



	Bit-width	Dataset	Workflow and impacts to model-trainer
QAT from scratch	<8W8A	Full training dataset	One stage: train from random initialization
PTQ	8W8A, 8W16A	Small (1~1000) calibration set, **even Synthetic Data**	Two stages: first train a float model, then a fast calibration stage
PTQ +Fine-tuning	<8W8A	Small (1~1000) calibration set, **even Synthetic Data**	Two stages of PTQ **+ QAT fine-tuning**

Benefits of QAT fine-tuning

Difference from FP accuracy for W4A8 quantization

Models	FP32	Per-tensor		Per-channel	
		PTQ	QAT	PTQ	QAT
ResNet18	69.68	-1.06	+0.08	-0.77	+0.33
ResNet50	76.07	-0.92	-0.18	-0.64	+0.45
MobileNetV2	71.72	-2.51	-1.55	-1.93	-1.24
InceptionV3	77.40	-0.92	+0.44	-0.58	+0.72
EfficientNet lite	75.42	-4.18	-3.87	-1.41	-1.50
DeepLabV3	72.94	-2.14	-2.04	-1.27	+0.07
EfficientDet-D1	40.08	-39.77	-4.74	-5.00	-3.33
BERT-base	83.06	-1.30	-0.42	-1.04	-0.67

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

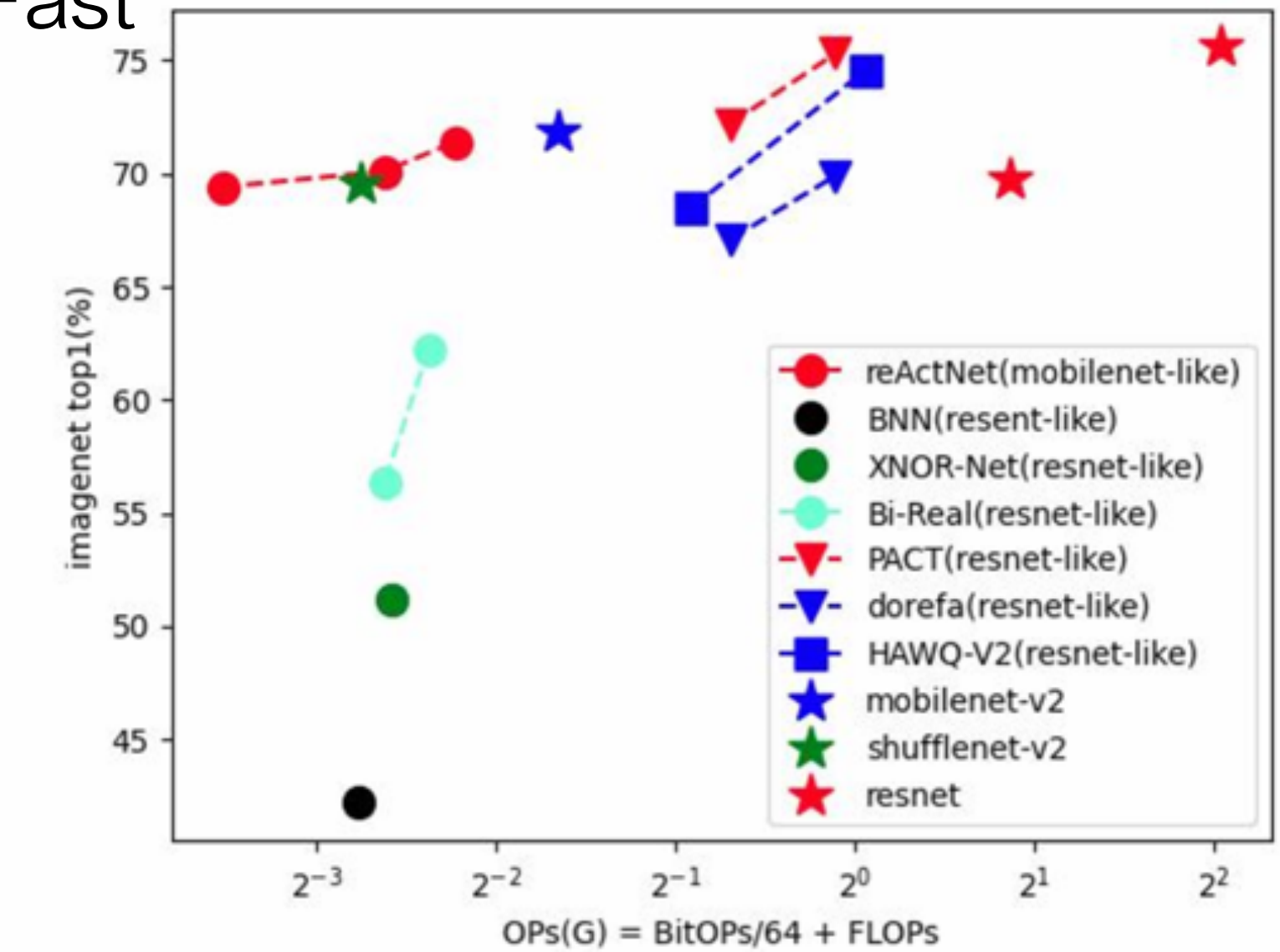
Designing Quantization friendly models

- When a QAT model is trained from scratch, we can redesign the network architecture.
- Guide for deciding #OP and bit-width
  - Focus on real speed as low #OP $\neq$ Fast
  - The power law of accuracy for Neural Networks
  - #BitOP complexity metric for QNN
  - Neural Architecture Search and Self-SL for QNN

Low #OP ≠ Fast

Inference speed (FPS) is only weakly correlated with #OP

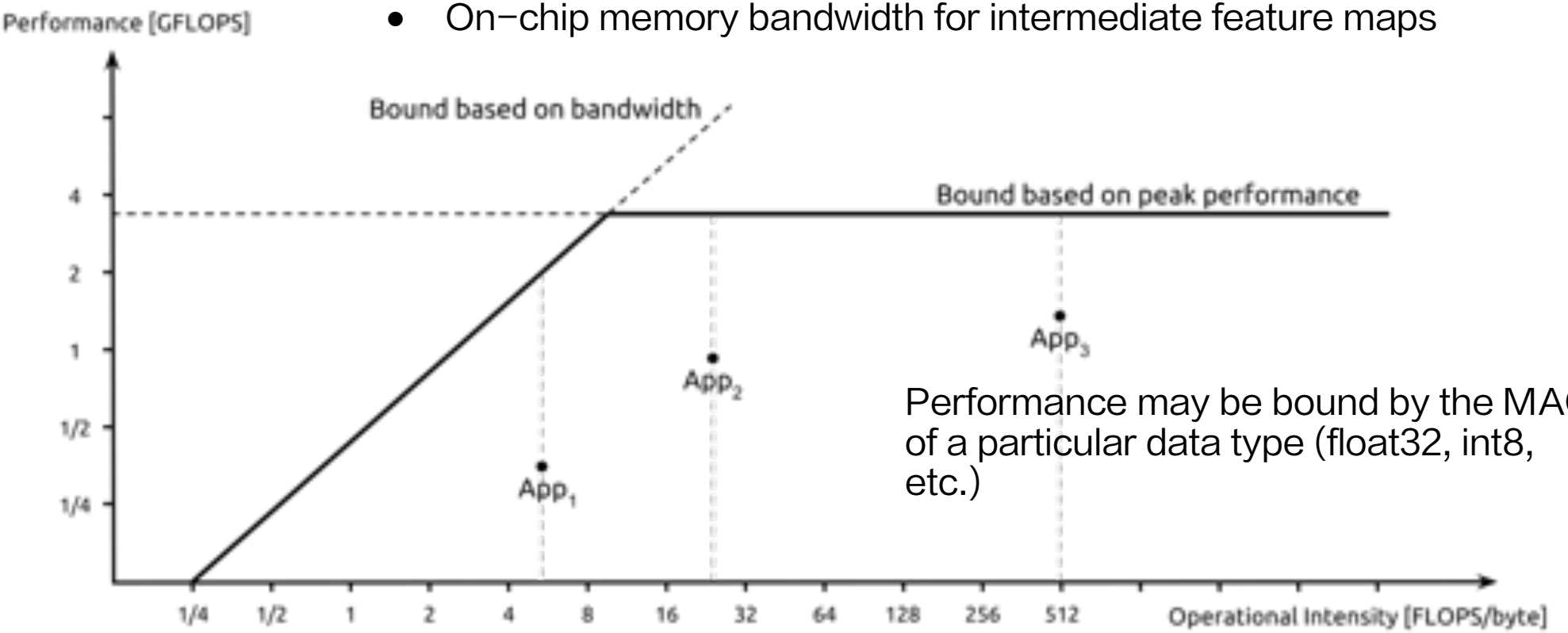
- E.g. Depthwise conv is IO-bound
- E.g. MAC may require a lower bound for channel number



Roofline model for inference FPS

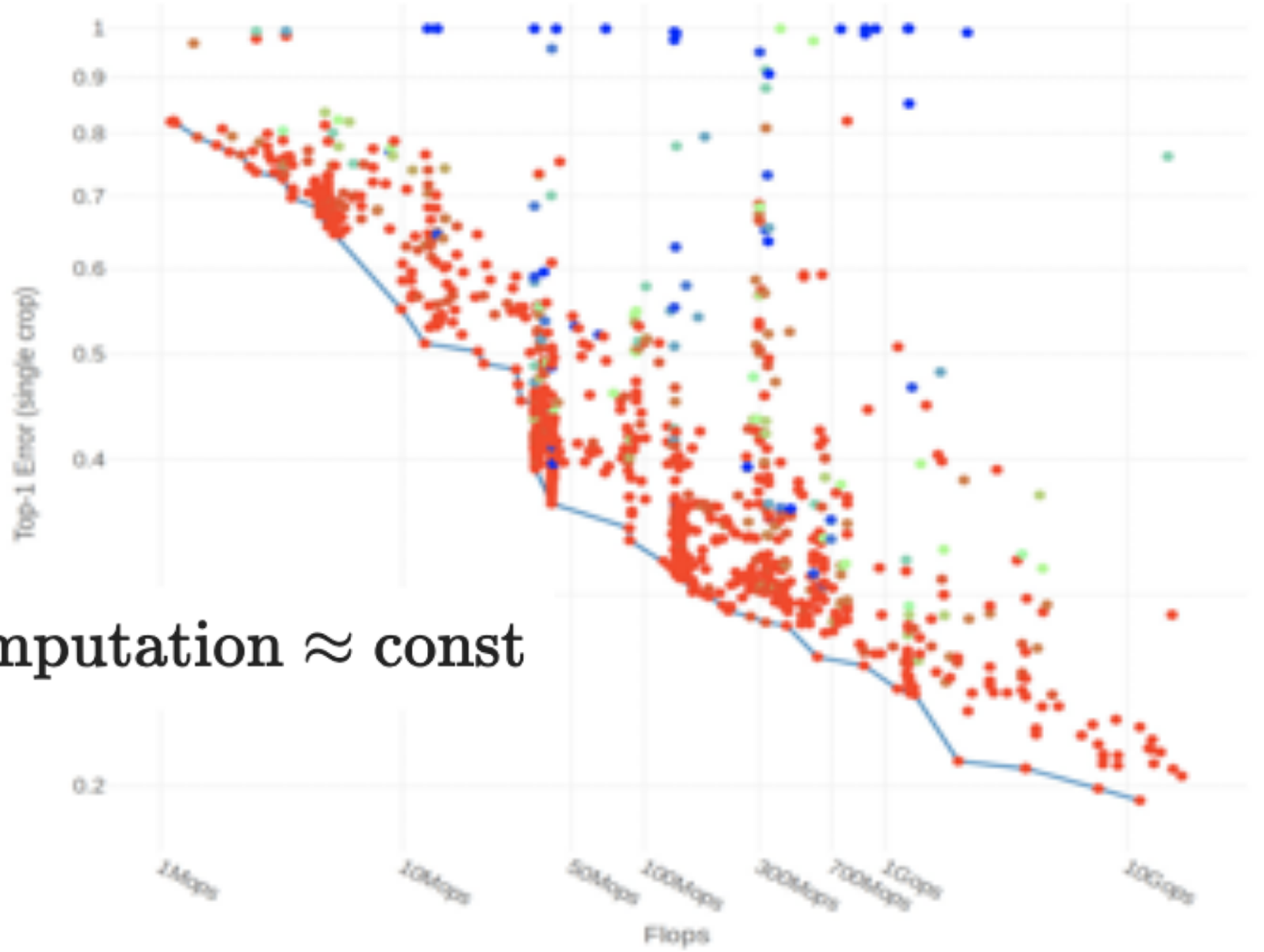
Bandwidth can be

- Off-chip memory bandwidth for loading model weights and intermediate feature maps, and instructions
- On-chip memory bandwidth for intermediate feature maps



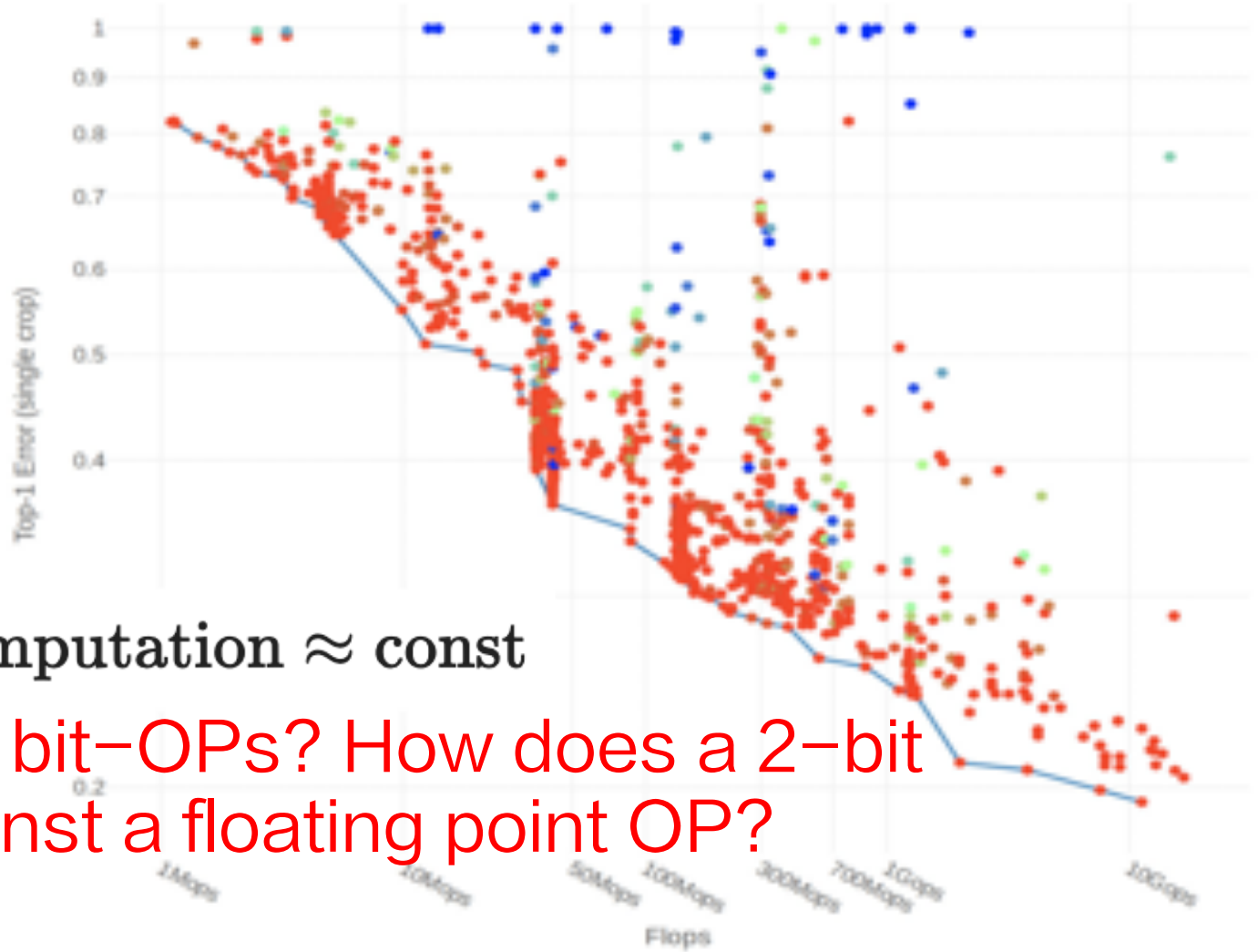
Performance may be bound by the MAC of a particular data type (float32, int8, etc.)

Accuracy-Computation Power Law



$$\log \text{Error} + \lambda \log \text{Computation} \approx \text{const}$$

Accuracy-Computation Power Law

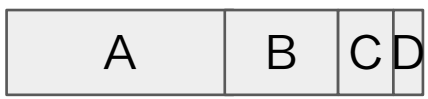


$$\log \text{Error} + \lambda \log \text{Computation} \approx \text{const}$$

But what about bit-OPs? How does a 2-bit OP counts against a floating point OP?

Table 1: Comparison of prediction accuracy for SVHN with different choices of Bit-width in a DoReFa-Net. W, A, G are bitwidths of weights, activations and gradients respectively. When bitwidth is 32, we simply remove the quantization functions.

W	A	G	Training Complexity	Inference Complexity	Storage Relative Size	Model A Accuracy	Model B Accuracy	Model C Accuracy	Model D Accuracy
1	1	2	3	1	1	0.934	0.924	0.910	0.803
1	1	4	5	1	1	0.968	0.961	0.916	0.846
1	1	8	9	1	1	0.970	0.962	0.902	0.828
1	1	32	-	-	1	0.971	0.963	0.921	0.841
1	2	2	4	2	1	0.909	0.930	0.900	0.808
1	2	3	5	2	1	0.968	0.964	0.934	0.878
1	2	4	6	2	1	0.975	0.969	0.939	0.878
2	1	2	6	2	2	0.927	0.928	0.909	0.846
2	1	4	10	2	2	0.969	0.957	0.904	0.827
1	2	8	10	2	1	0.975	0.971	0.946	0.866
1	2	32	-	-	1	0.976	0.970	0.950	0.865
1	3	3	6	3	1	0.968	0.964	0.946	0.887
1	3	4	7	3	1	0.974	0.974	0.959	0.897
1	3	6	9	3	1	0.977	0.974	0.949	0.916
1	4	2	6	4	1	0.815	0.898	0.911	0.868
1	4	4	8	4	1	0.975	0.974	0.962	0.915
1	4	8	12	4	1	0.977	0.975	0.955	0.895
2	2	2	8	4	1	0.900	0.919	0.856	0.842
8	8	8	-	-	8			0.970	0.955
32	32	32	-	-	32	0.975	0.975	0.972	0.950



A has two times as many channels as B. B has two times as many channels as C.

...

dorefa-net
https://arxiv.org/abs/1606.06160

Table 1: Comparison of prediction accuracy for SVHN with different choices of Bit-width in a DoReFa-Net. W, A, G are bitwidths of weights, activations and gradients respectively. When bitwidth is 32, we simply remove the quantization functions.

W	A	G	Training Complexity	Inference Complexity	Storage Relative Size	Model A Accuracy	Model B Accuracy	Model C Accuracy	Model D Accuracy
1	1	2	3	1	1	0.934	0.924	0.910	0.803
1	1	4	5	1	1	0.968	0.961	0.916	0.846
1	1	8	9	1	1	0.970	0.962	0.902	0.828
1	1	32	-	-	1	0.971	0.963	0.921	0.841
1	2	2	4	2	1	0.909	0.930	0.900	0.808
1	2	3	5	2	1	0.968	0.964	0.934	0.878
1									
2									
2									
1									
1									
1									
1	3	4	7	3	1	0.974	0.974	0.959	0.897
1	3	6	9	3	1	0.977	0.974	0.949	0.916
1	4	2	6	4	1	0.815	0.898	0.911	0.868
1	4	4	8	4	1	0.975	0.974	0.962	0.915
1	4	8	12	4	1	0.977	0.975	0.955	0.895
2	2	2	8	4	1	0.900	0.919	0.856	0.842
8	8	8	-	-	8			0.970	0.955
32	32	32	-	-	32	0.975	0.975	0.972	0.950

log Error ∝ log #bit operations

= #input−channel * #output−channel * input−bitwidth * weight−bitwidth

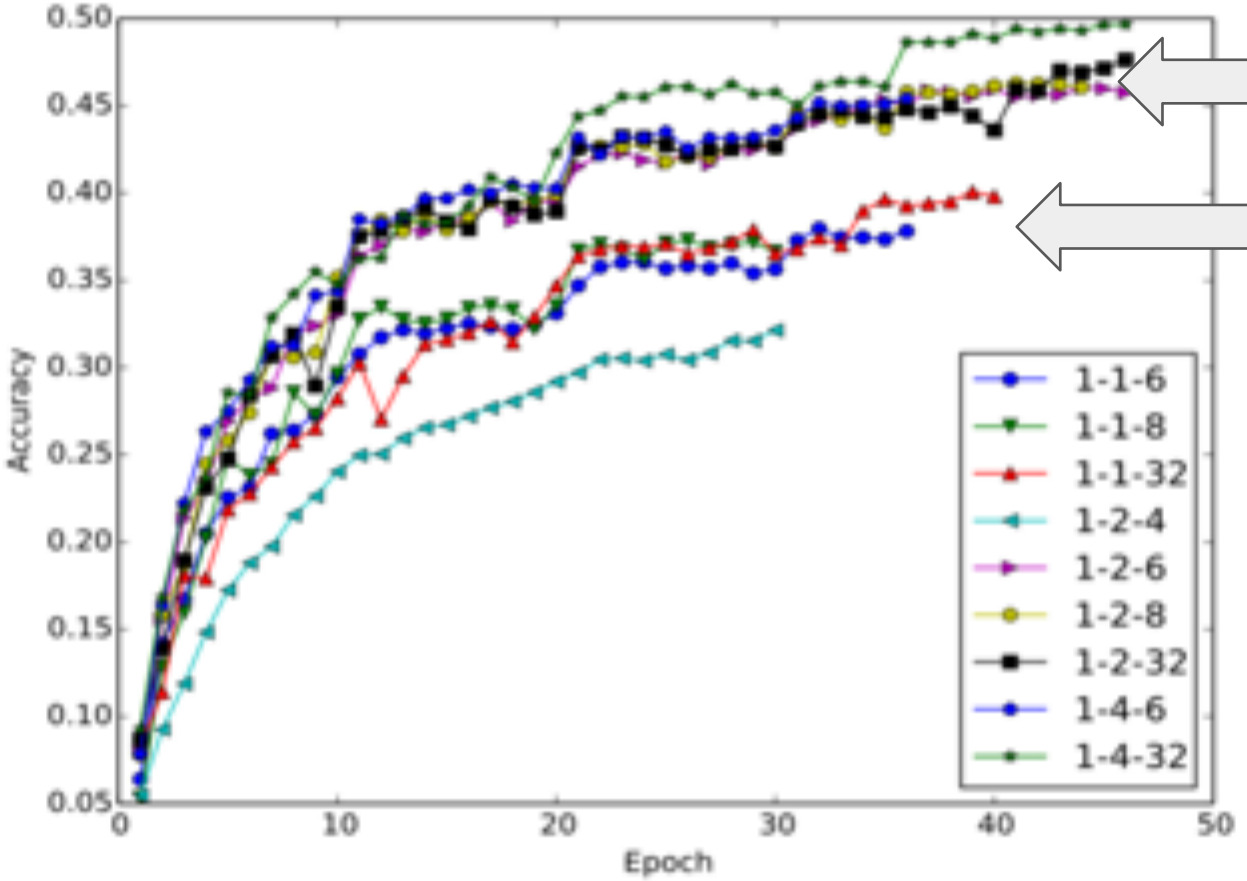
A has two times as many channels as B. B has two times as many channels as C.

...

dorefa−net

https://arxiv.org/abs/1606.06160

When Gradient bit-width is large enough, accuracy only depends on W-Bit and A-Bit



W-Bit * A-Bit = 4

W-Bit * A-Bit = 1

log Error $\propto$ log #bit operations

= #input-channel * #output-channel * input-bitwidth * weight-bitwidth

https://www.arxiv.org/pdf/1612.00212v1

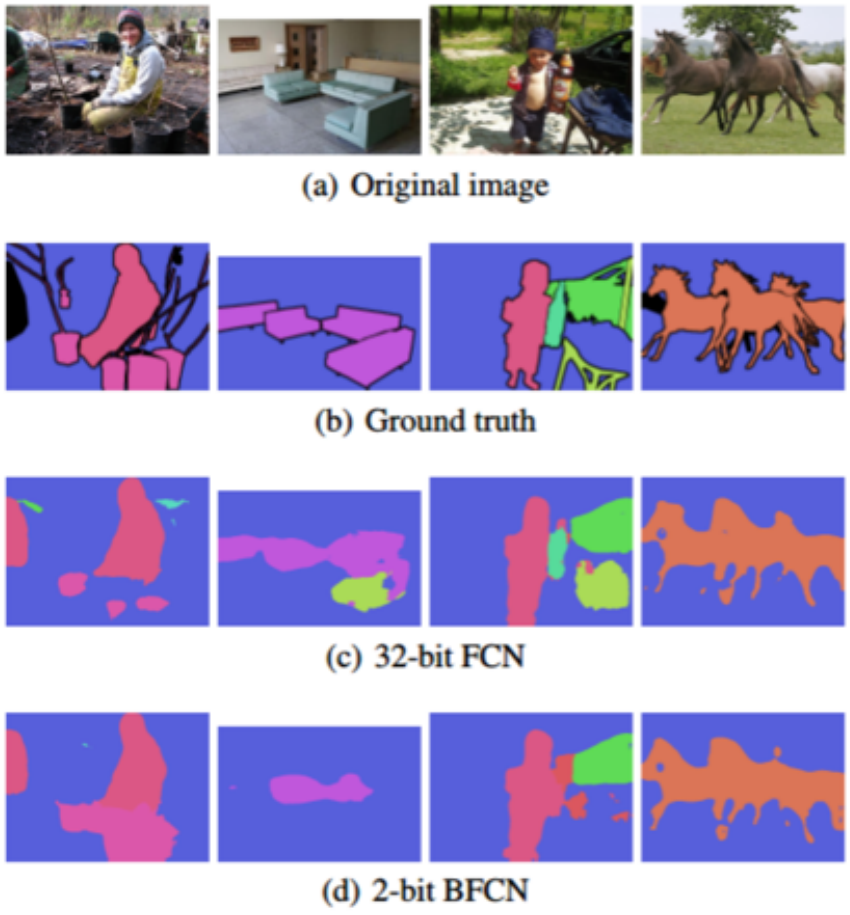
MEGVII 旷视

bit-width (W / A)	mean IoU	Complexity
32 / 32	69.8%	-
8 / 8	69.8%	64
4 / 4	68.6%	16
3 / 3	67.4%	9
2 / 2	65.7%	4
1 / 4	64.4%	4
4 / 1	diverge	4
1 / 2	62.8%	2

Table 5: Results of different bit-width allocated to weight and activation on PASCAL VOC 2012 val set. W represents

$\log \text{Error} \propto \log \#\text{bit operations}$

$= \#\text{input-channel} * \#\text{output-channel} * \text{input-bitwidth} * \text{weight-bitwidth}$



(a) Original image

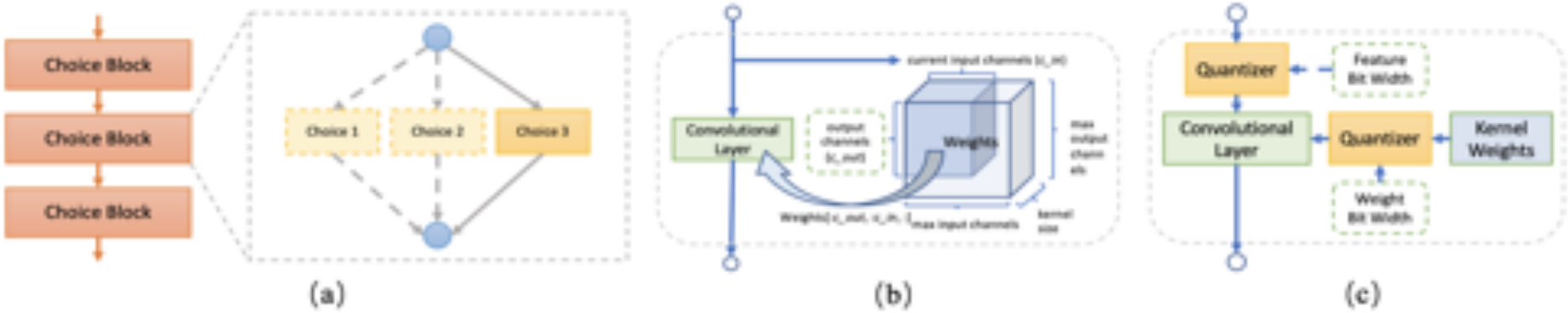
(b) Ground truth

(c) 32-bit FCN

(d) 2-bit BFCN

Figure 4: Examples on PASCAL VOC 2012.

Neural Architecture Search for QNN



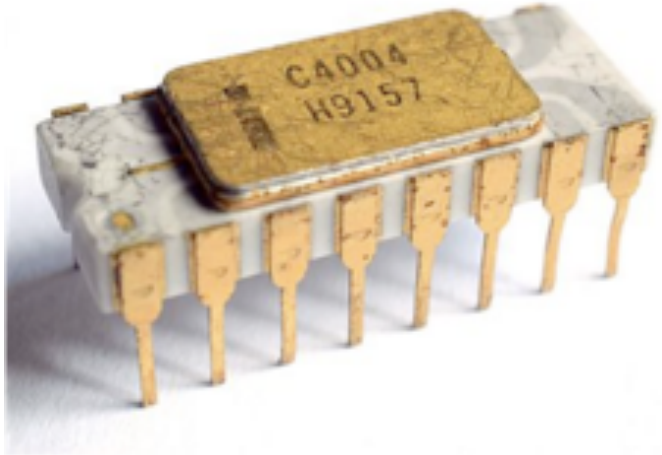
Single Path One-Shot Neural Architecture Search with Uniform Sampling https://arxiv.org/pdf/1904.00420.pdf

Fig. 3. *Choice blocks* for (a) our single path supernet (b) channel number search (c) mixed-precision quantization search

And Self-supervised Learning for training supernet model that allows Accuracy-Quantization tradeoff: SSQL https://arxiv.org/abs/2207.05432

Quantized Neural Network: Hardware

- Gifts from hardware to be exploited
- Implementing Real Time Video Processing on an FPGA
  - Multiple bit-width support
  - A minimal toolchain



Intel® 4004, released 1971
1.1 Kops @ ??

Tesla® T4, released 2018
260 Tops @ 75 Watt

Gifts from hardware: Non-uniform Quantization

- Logarithmic Quantization (Non-uniform)
  - Quantize log(x) instead of x
  - Achieve 12 bit quality with 8 bits
  - Con: emulation not efficient on GPU/CPU
- General: Non-uniform quantization with Lookup-Table

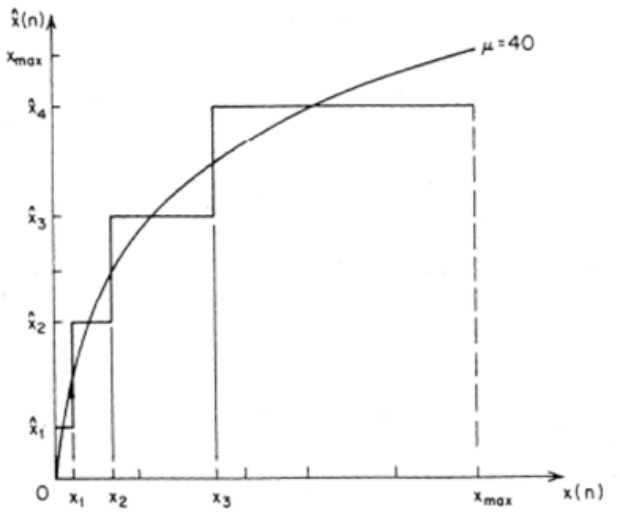
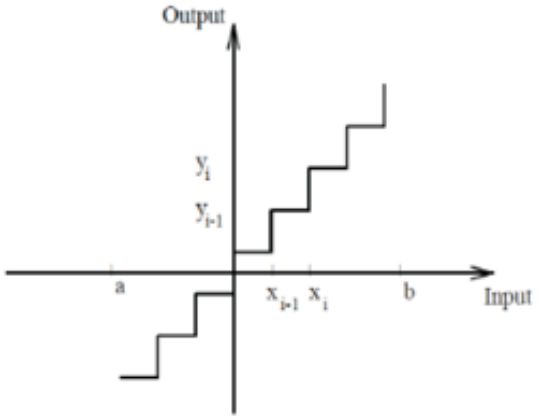
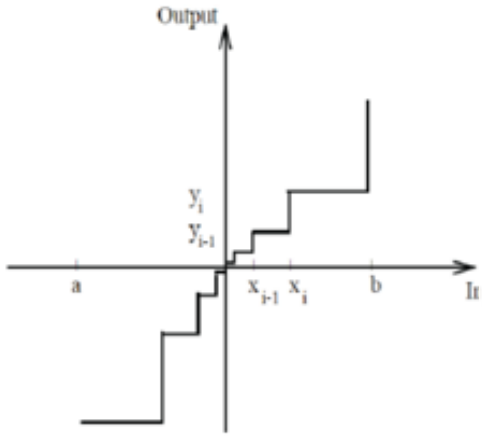
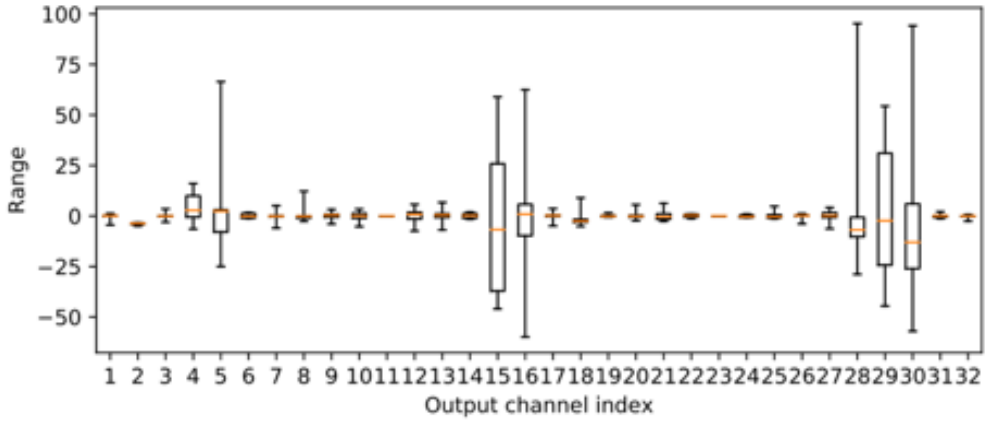


Fig. 5.16 Distribution of quantization levels for a μ-law 3-bit quantizer with $\mu = 40$.

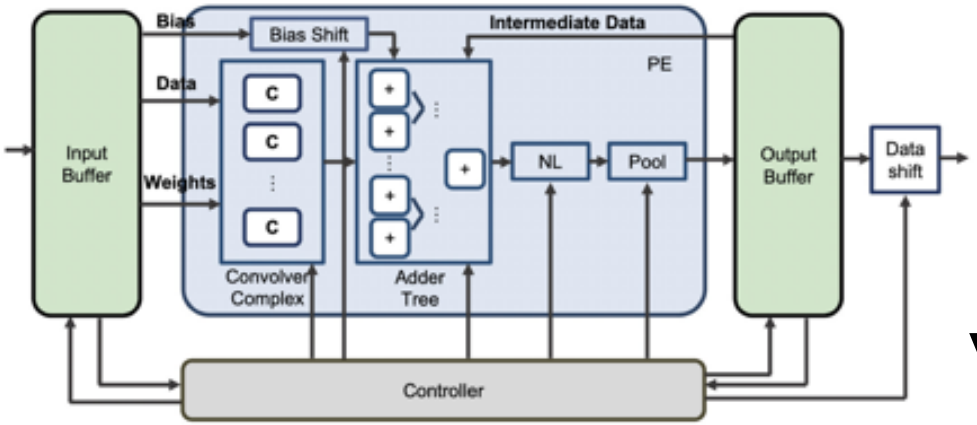






Hardware Support for Quantization

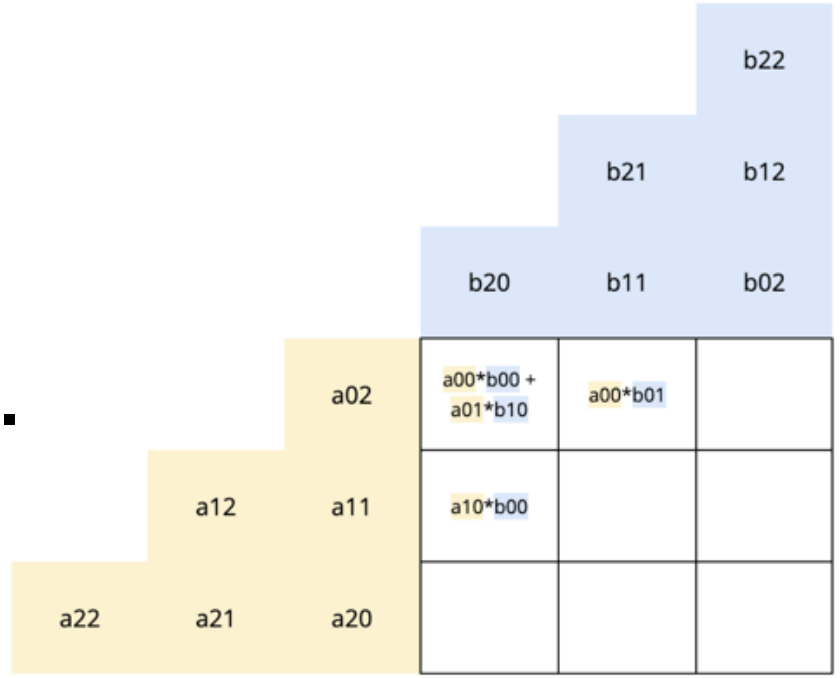
- Processing Engine Architecture



- Achieve intra-output parallelism by placing multiple Convolvers
- Convolver: optimized for 3x3 convolution operation
- Adder Tree: sum up results of one convolution operation
- NL: supports non-linear function (ReLU)
- Pool: supports max-pooling
- Bias Shift & Data Shift: support dynamic-precision fixed-point numbers

Going Deeper with Embedded FPGA Platform for Convolutional Neural Network
https://dl.acm.org/doi/10.1145/2847263.2847265

V.S.



Systolic array for matrix multiplication

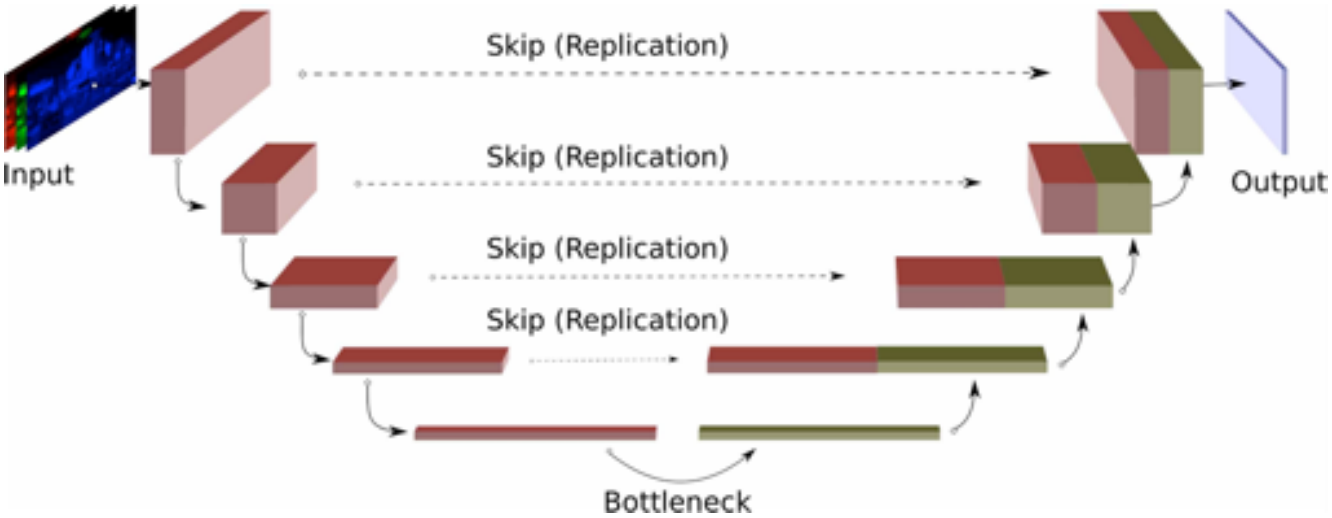
Case study: Build a Real-time Vision Pipeline with a low end FPGA

- Zynq 7020 FPGA
- costs 18 USD, ~20Gops float16 ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network (2020)
- FHD (1080p) is ~40 times the 224x224 resolution
- Need a 1000 FPS backbone for Real-time (25FPS) processing!

	Z-7020
Logic Cells (K)	85
Block RAM (Mb)	4.9
DSP Slices	220
Maximum I/O Pins	200
Maximum Transceiver Count	-

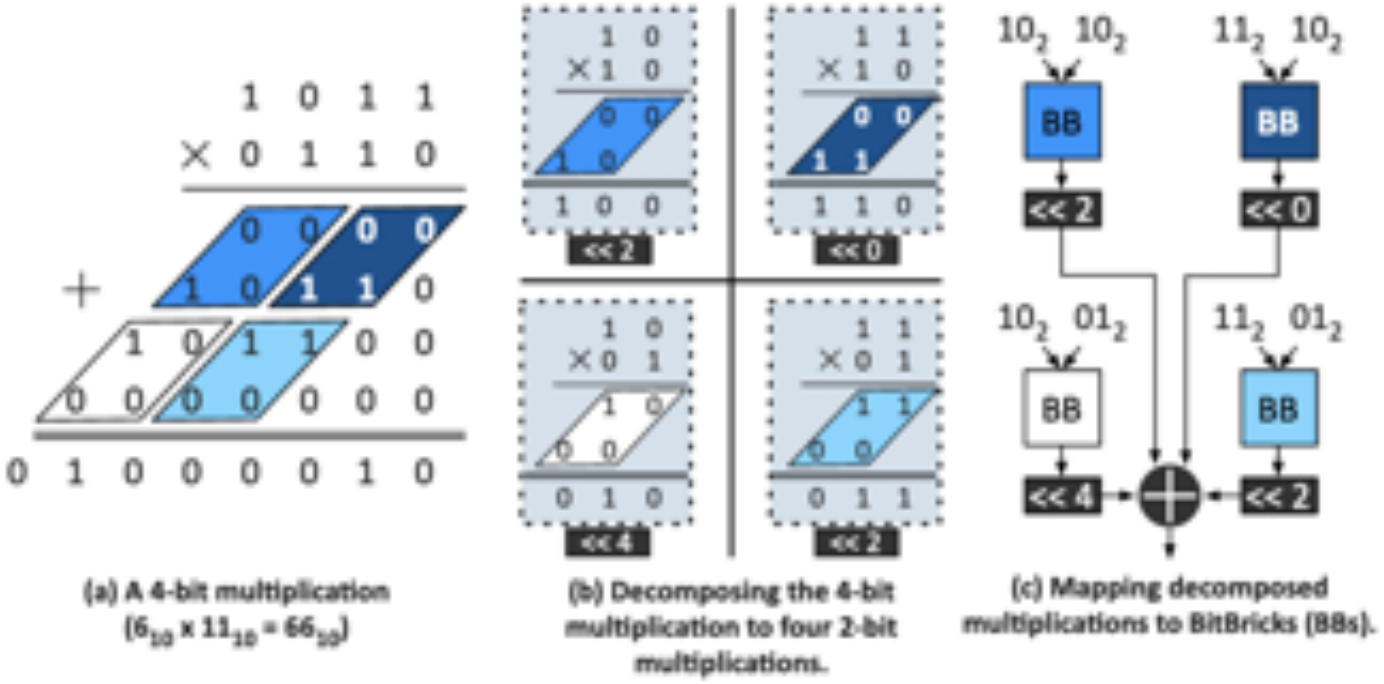
Multiple Bit-width Support

- RGB decoded from JPEG and video: 8-bit YUV
- RAW input from sensor, 10~14-bit
- Precomputed features: 8-bit, 16-bit, or even floating point



An AI-ISP model may need 16-bit at the two end,
but can use 4-bit in intermediate layers.

Multiple Bit-width Support



(a) A 4-bit multiplication ($6_{10} \times 11_{10} = 66_{10}$)

(b) Decomposing the 4-bit multiplication to four 2-bit multiplications.

(c) Mapping decomposed multiplications to BitBricks (BBs).

Fig. 6: Using BitBricks to execute 4-bit multiplications.

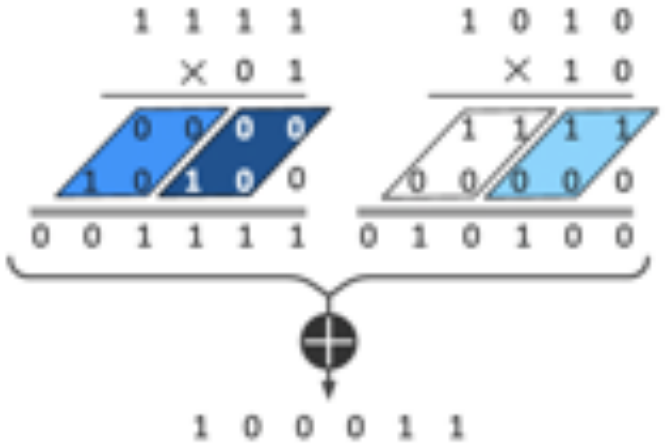
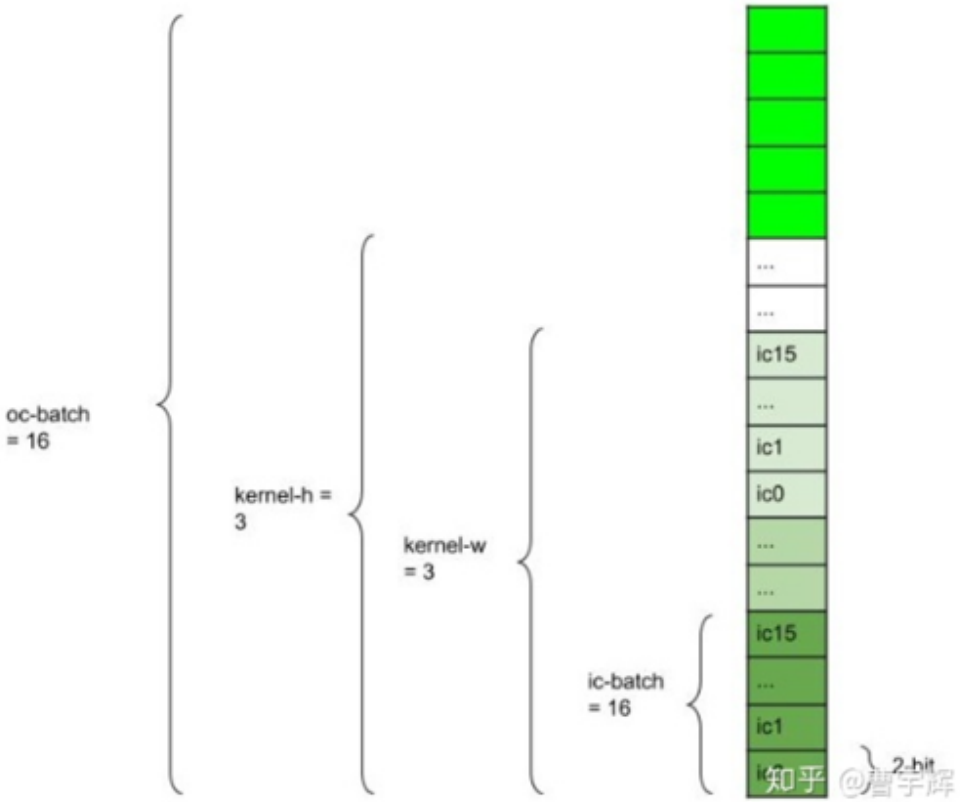


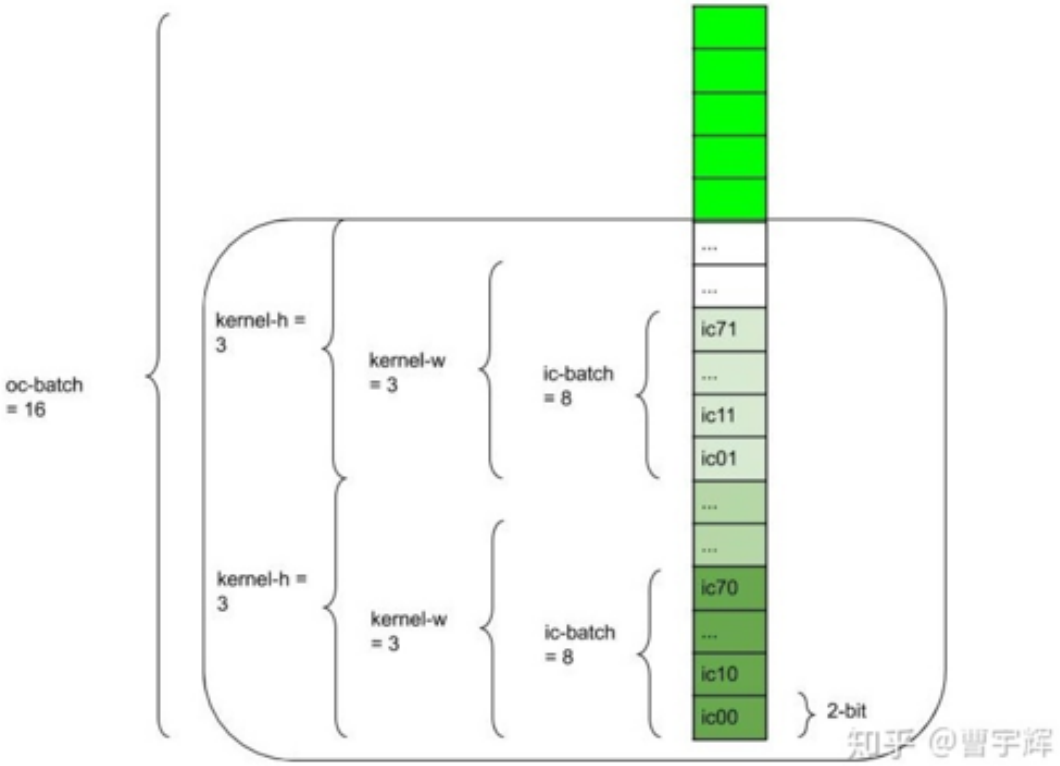
Fig. 7: Two 4-bit × 2-bit multiplications decomposed to four 2-bit multiplications followed by the accumulation (summation) logic.

https://arxiv.org/pdf/1712.01507.pdf

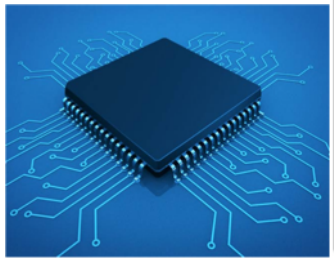
Our work (parallel work to BitFusion)

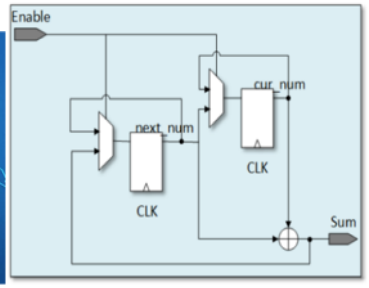
4-bit computation is implemented as shifted-add.





Quantization impacts across the Computation Stack

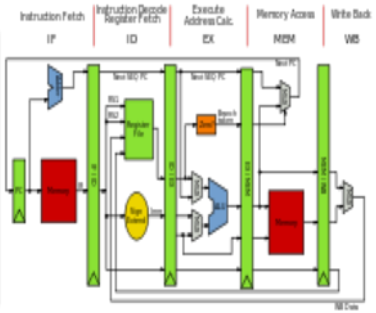




Circuit to Generate Fibonacci Series *(Fig. 13)*

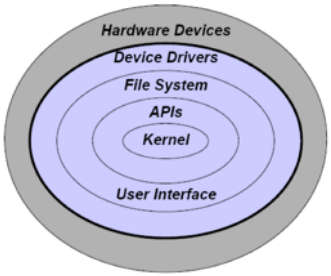
Verilog

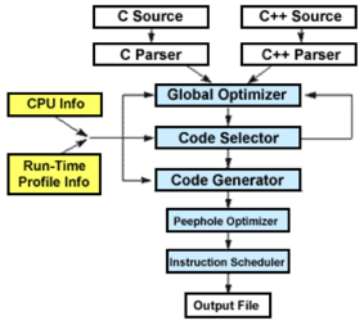
- Precise bit-width



Architecture

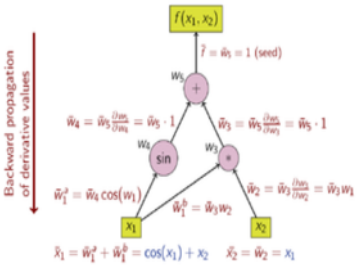
- Multi bit-width compliance





Compiler

- Quantization domain partitioning (different bit-width)
- Programming Interface



Computation Graph Engine

- Quantized Kernels
- Transformations

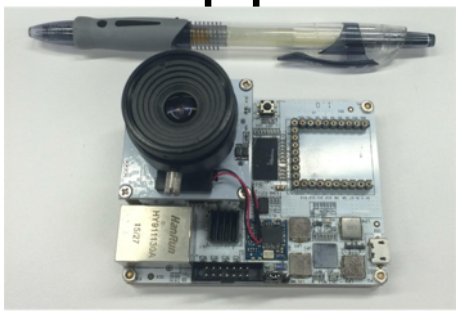
Benefits of Quantized Neural Networks

- Reduce weight size
- Reduce feature size
  - larger feature library size
- Speedup computation (usually require hardware support)
  - O(Bit-width) if SIMD, O(Bit-width^2) if special hardware
  - Saves bandwidth (intra-chip and off-chip), eases P & R
  - Dense computation

$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} \, bitcount[and(c_m(\boldsymbol{x}), c_k(\boldsymbol{y}))]$$

	INT8	INT4	Binary
Tesla T4	130 T	260 T	1104 T
Megvii Zynq 7020	0.05 T	0.2 T	3.2 T

Real-time Object Detection on a low-end FPGA with QNN support



Zynq 7020

Conclusion

- Quantization is an effective method for enabling larger neural network with the same hardware budget (Power, Performance, Area)
- Post Training Quantization can be applied to trained models to gain significant hardware speedup.
- Quantization Errors can be effectively reduced with Quantization Aware Training.
- Codesigning model allows running powerful vision pipelines on very resource-limited hardware.