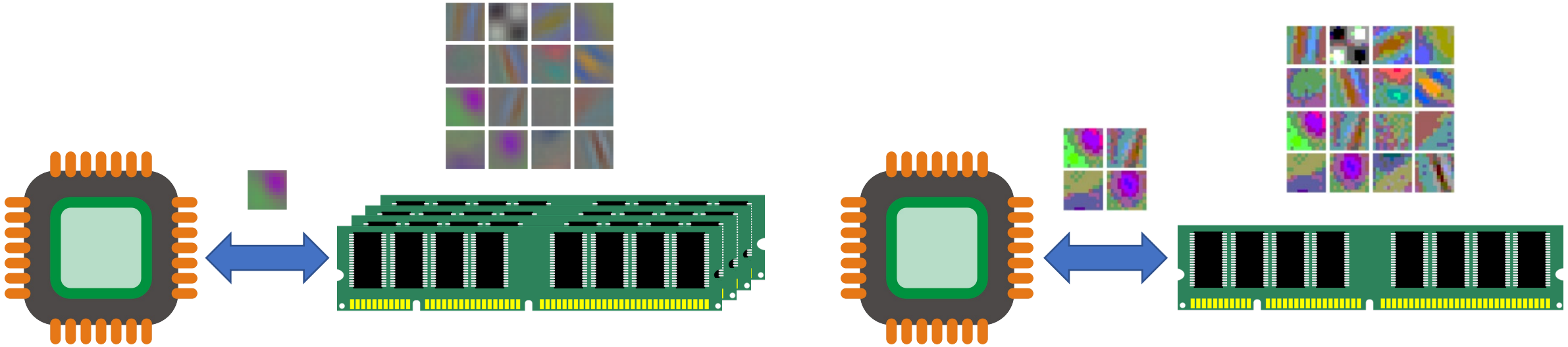
Deep Learning Optimization - Quantization

May 1, 2023

Eunhyeok Park

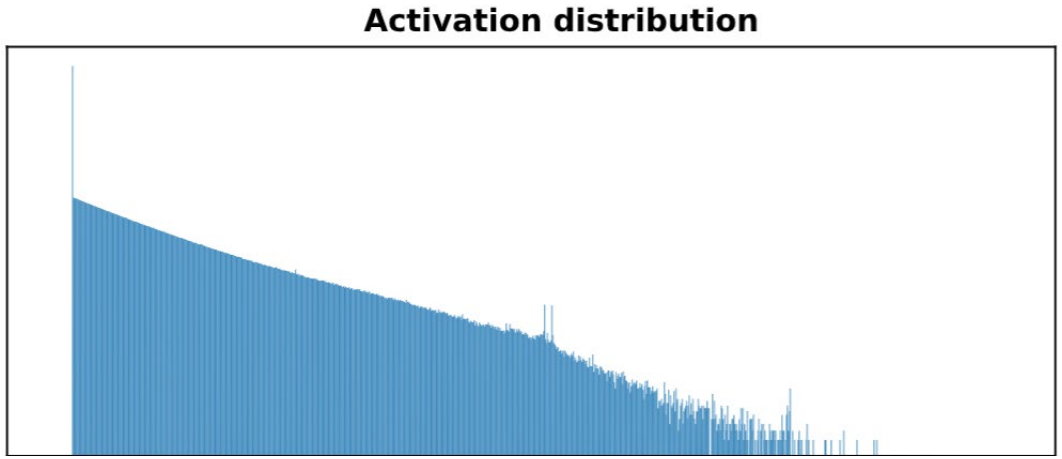
Benefit of Quantization

- Quantization is beneficial to both energy consumption and performance
  - Data can be fetched quickly with limited bandwidth
  - The same number of data can be stored in the smaller storage space

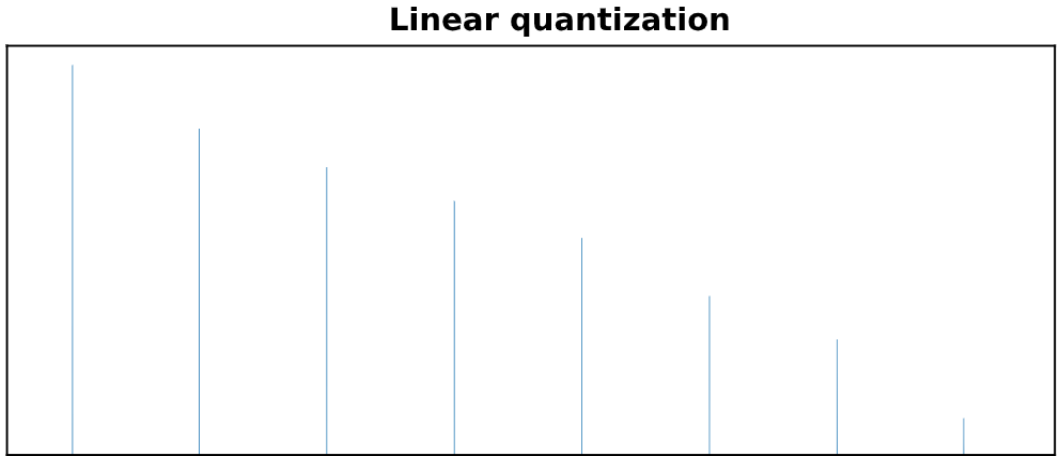


Quantization

- Quantization is the task of restricting the data representation
  - Convert high-precision data to low-precision one
  - Map data to discontinuous values based on clustering



32-bit floating point



3-bit integer with constant scale

Quantization - 2

- Quantization is the task of restricting the data representation
  - Convert high-precision data to low-precision one
  - Map data to discontinuous values based on clustering
  - Assign the integer index to discrete value
    - Store index of data + mapping table

0.40	0.05	0.01	1.16
0.26	0.57	0.19	0
0.09	0.75	1.99	0.46
0.37	1.39	1.16	1.69

32-bit floating point

0.51	0.10	0.10	1.27
0.10	0.51	0.10	0.10
0.10	0.51	1.84	0.51
0.51	1.27	1.27	1.84

4-level quantization

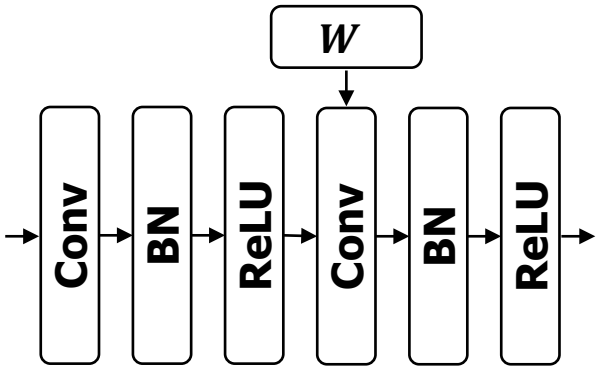
1	0	0	2
0	1	0	0
0	1	3	1
1	2	2	3

0	0.10
1	0.51
2	1.27
3	1.84

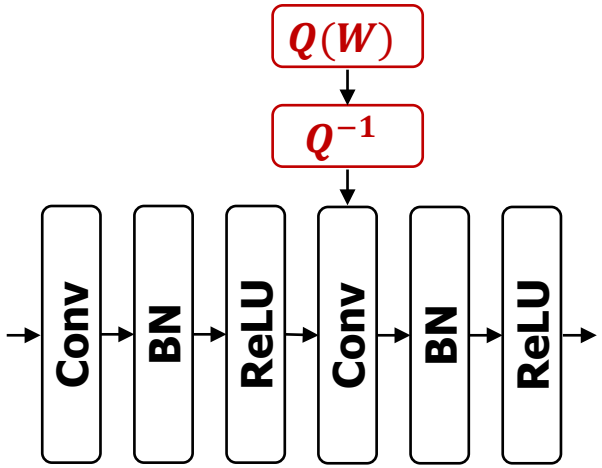
2-bit quantization

Neural Network Quantization

- Insert quantization operator within DNN
  - Change data representation while preserving network structure
  - By default, apply full-precision computation after de-quantization
  - Take advantage of computation benefit when HW acceleration is available



Full-precision

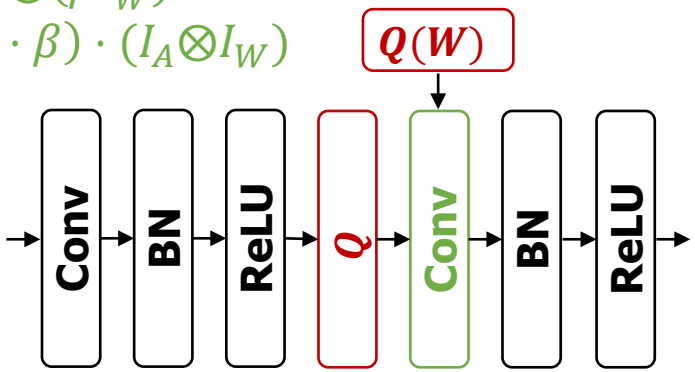


Quantization

Ex) Integer op. w/ scaling

$(\alpha I_A) \otimes (\beta I_W)$

$= (\alpha \cdot \beta) \cdot (I_A \otimes I_W)$



Quantization + Acceleration

Linear vs Non-linear Quantization

1	0	0	2
0	1	0	0
0	1	3	1
1	2	2	3

Index Tensor

0	0.10
1	0.51
2	1.27
3	1.84

Mapping Table

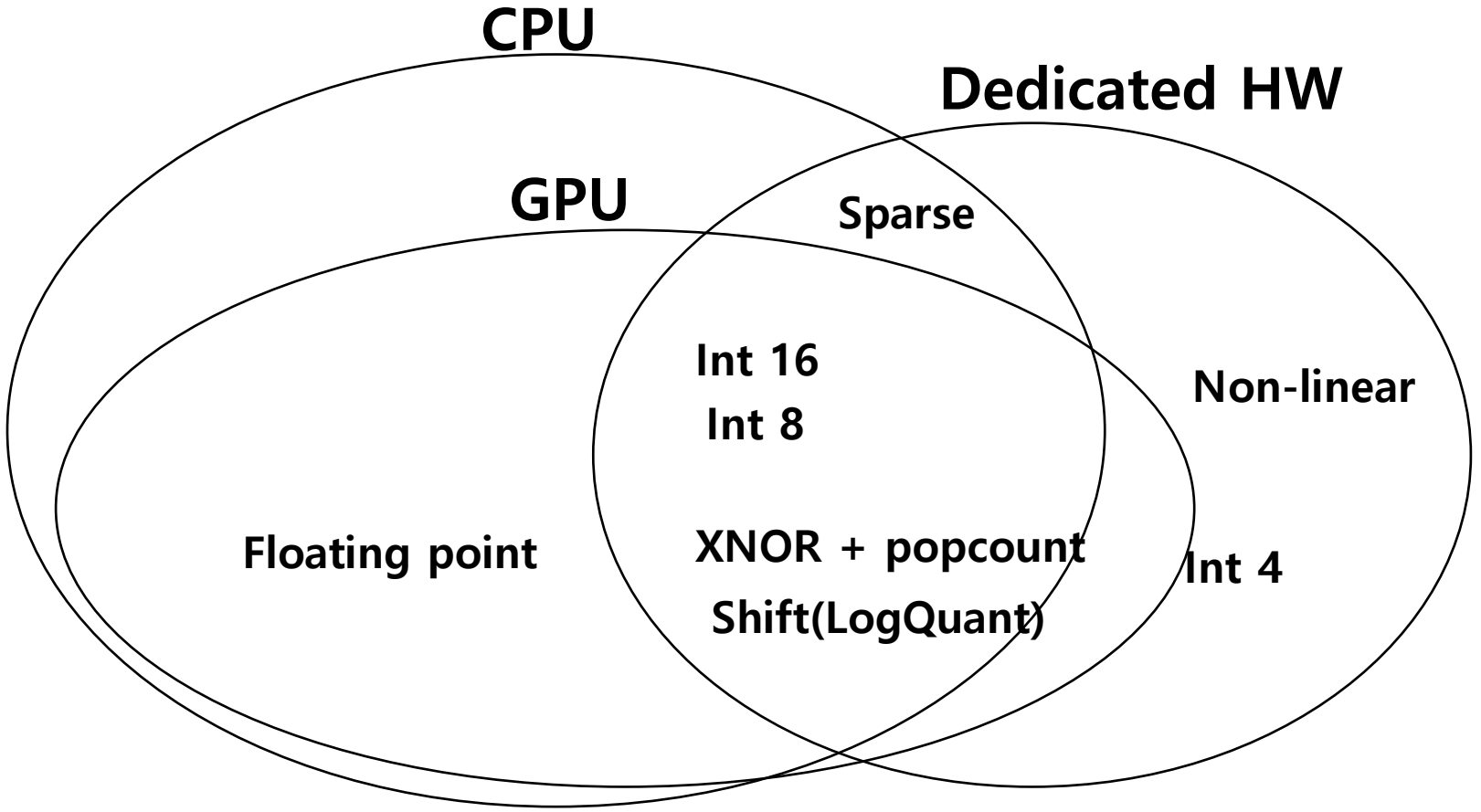
- Non-linear Quantization
  - High bit-width efficiency
  - High-precision computation is required (not always)

- Linear Quantization
  - Every quantization levels have equal space
  - Mapping table is not necessary
    - High-precision value can be computed from the index tensor with scaling/shift values
  - Implemented based on fixed point computation with element-wise floating-point scale/shift

- Binary Quantization?

HW-Friendly Computation

- Quantization is the task of restricting the data representation
  - Quantization could be beneficial to computation



XNOR-Net

Why XNOR?

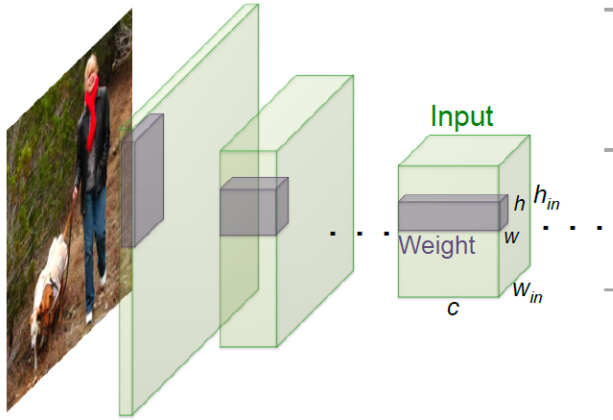
- { 1, -1, 1, 1} dot_prod { -1, -1, 1, -1} = 1*(-1)+(-1)*(-1)+1*1+1*(-1) =-1+1+1-1=0
- XNOR operation

A	B	A XNOR B (1/0 vs 1/-1)
0 (-1)	0 (-1)	1 (=(-1)*(-1)=1)
0 (-1)	1 (1)	0 (=(-1)*1=-1)
1 (1)	0 (-1)	0 (=1*(-1)=-1)
1 (1)	1 (1)	1 (=1*1=1)

- Multiplications + accumulation → XNOR + bit count + addition (# 1's - # (-1)'s)

XNOR-Net

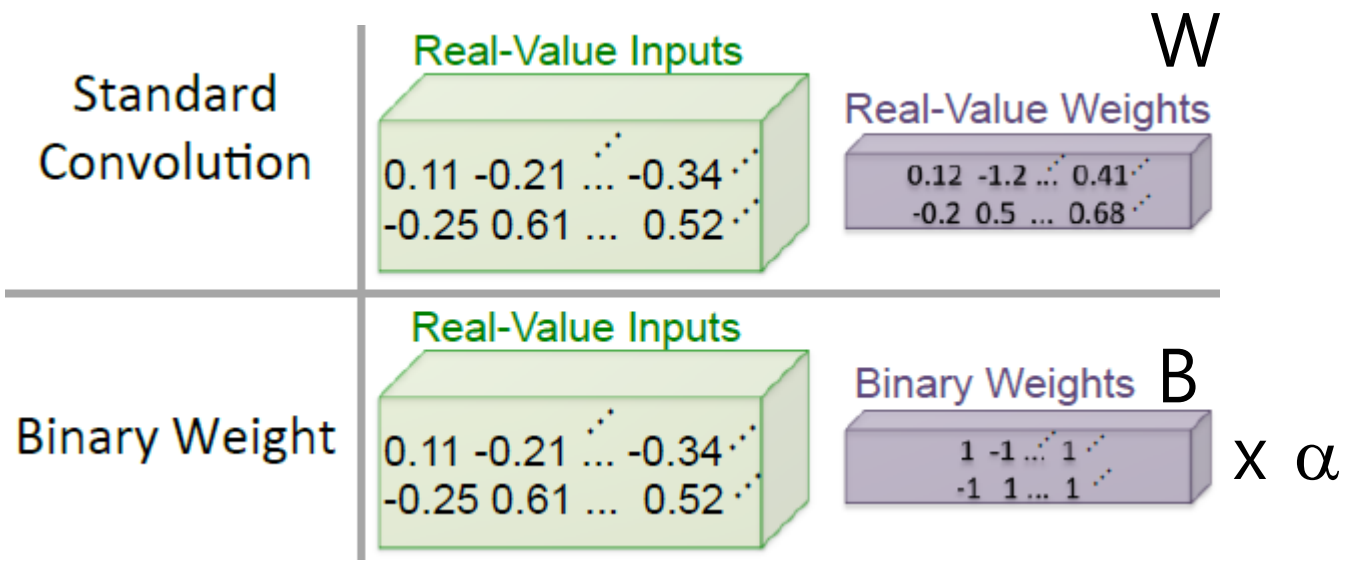
- Two types of network
  - Binary-Weight network
    - Only weights are binary, activations are 32-bit floating point values.
    - It does not hurt accuracy while reducing weight size significantly
  - XNOR-Net
    - Both weight and activation are binary



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs 1 -1 ... -1 / -1 1 ... 1; Binary Weights 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Binary-Weight-Network

we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$.



$$\mathbf{W} \approx \alpha \mathbf{B}$$

$$\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$$

$$\mathbf{I} * \mathbf{W} \approx (\mathbf{I} \oplus \mathbf{B})\, \alpha$$

$\oplus$ indicates a convolution without any multiplication.

How to Obtain B and α ?

we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$.

Solve this

$$J(\mathbf{B}, \alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$$

$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha)$$

$$\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$$

$$J(\mathbf{B}, \alpha) = \alpha^2 \mathbf{B}^\mathsf{T}\mathbf{B} - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{W}^\mathsf{T}\mathbf{W}$$

$\mathbf{B}^\mathsf{T}\mathbf{B} = n$ is a constant

$\mathbf{W}^\mathsf{T}\mathbf{W}$ is also a constant

$$J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{c}$$

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}}\{\mathbf{W}^\mathsf{T}\mathbf{B}\} \quad s.t. \ \mathbf{B} \in \{+1, -1\}^n$$

How to Obtain B and α ?

we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$.

Solve this

$$J(\mathbf{B}, \alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$$

$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha)$$

$$\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$$

$$J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{c}$$

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}}\{\mathbf{W}^\mathsf{T}\mathbf{B}\} \quad s.t. \ \mathbf{B} \in \{+1, -1\}^n$$

$$\mathbf{B}^* = \operatorname{sign}(\mathbf{W})$$

$$J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B}^* + \mathbf{c}$$

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T}\mathbf{B}^*}{n}$$

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T}\operatorname{sign}(\mathbf{W})}{n} = \frac{\sum|\mathbf{W}_i|}{n} = \frac{1}{n}\|\mathbf{W}\|_{\ell 1}$$

Training for Binary-Weight-Network

Algorithm 1 Training an L-layers CNN with binary weights:

Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t. 32bit

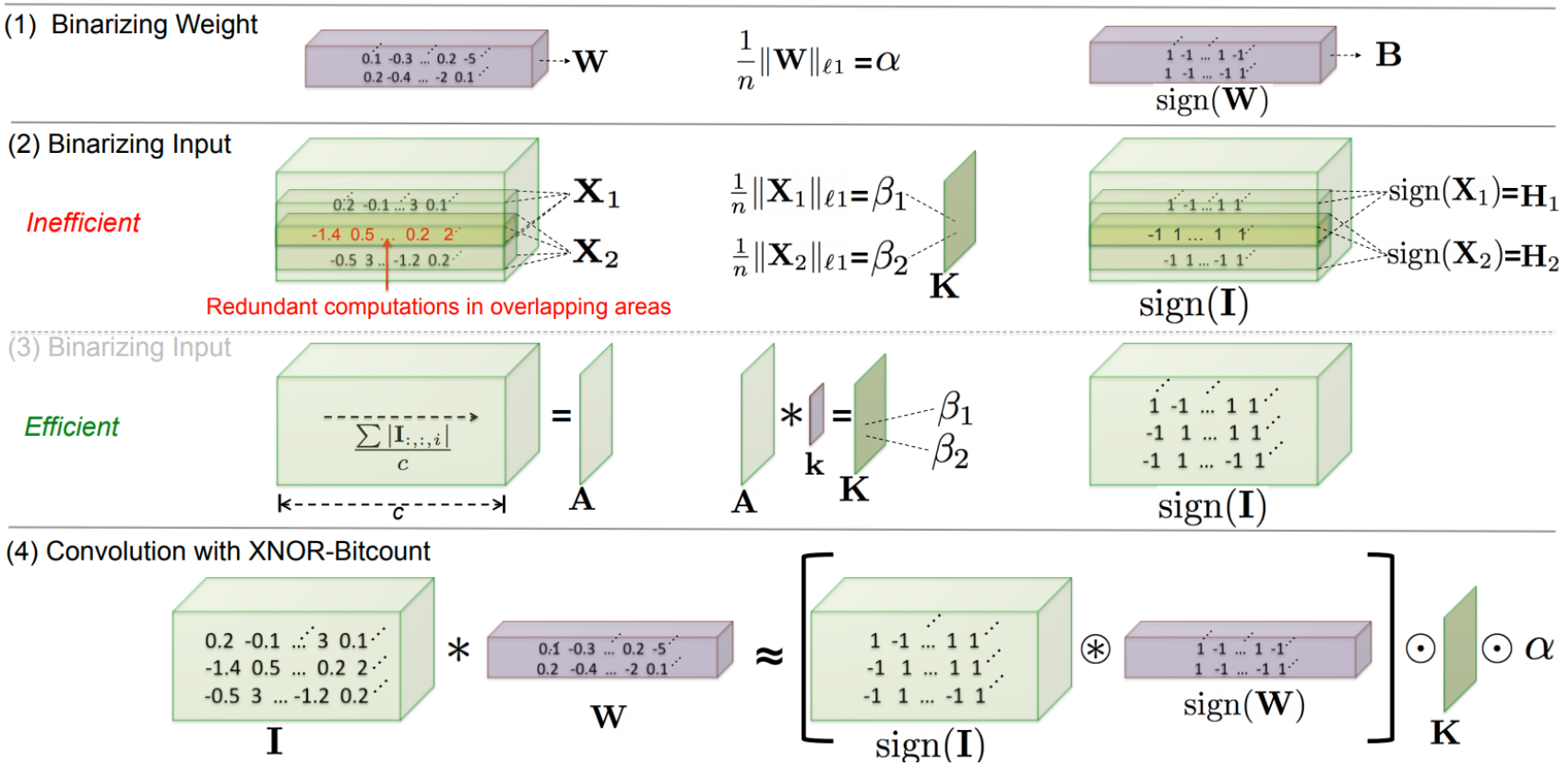
Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

Binarizing weight filters:
for $l = 1$ to L **do**
  for k^{th} filter in l^{th} layer **do**
    $\mathcal{A}_{lk} = \frac{1}{n}\|\mathcal{W}^t_{lk}\|_{\ell 1}$
    $\mathcal{B}_{lk} = \text{sign}(\mathcal{W}^t_{lk})$
    $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk}\mathcal{B}_{lk}$ Forward pass with binary weights
$\hat{\mathbf{Y}} =$ **BinaryForward**$(\mathbf{I}, \mathcal{B}, \mathcal{A})$ // standard forward propagation except that convolutions are computed using equation 1 or 11 Backward pass with binary weights
$\frac{\partial C}{\partial \widetilde{\mathcal{W}}} =$ **BinaryBackward**$(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}})$ // standard backward propagation except that gradients are computed using $\widetilde{\mathcal{W}}$ instead of $\mathcal{W}^t$ Update full-precision weights
$\mathcal{W}^{t+1} =$ **UpdateParameters**$(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t)$ // Any update rules (*e.g.*,SGD or ADAM)
$\eta^{t+1} =$ **UpdateLearningrate**(η^t, t) // Any learning rate scheduling function

XNOR-Network (Binary Weight and Binary Activation Network)

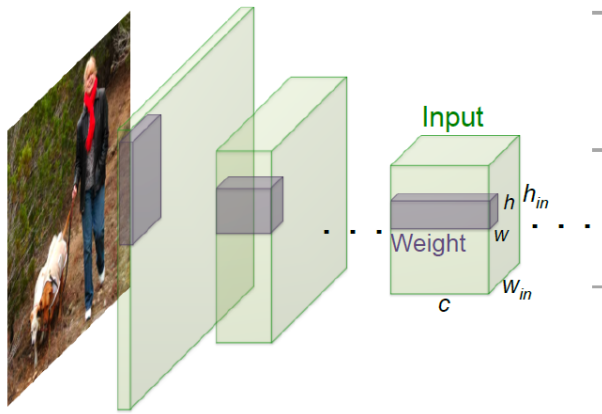
- We approximate both X and B with binary vectors

$$\mathbf{X}^\top \mathbf{W} \approx \beta \mathbf{H}^\top \alpha \mathbf{B}, \text{ where } \mathbf{H}, \mathbf{B} \in \{+1, -1\}^n \text{ and } \beta, \alpha \in \mathbb{R}^+$$



AlexNet Results

- Binary-Weight network:
  - Only weights are binary, activations are 32-bit floating point values
  - It does not hurt accuracy while reducing weight size significantly
- XNOR-Net:
  - Both weight and activation are binary. 12.6% top-1 loss
  - Note that additional full precision scaling computation is required



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

ResNet-18 and GoogLeNet

- Binary weight
  - 8.5% (ResNet-18) and 5.8% top-1 loss (GoogLeNet)
- XNOR
  - 18.1% loss in ResNet-18
- Shallower models tend to require more bits
  - ResNet-18 is more difficult to quantize than ResNet-50
  - Deeper models have more parameters which can compensate for the accuracy loss due to quantization

	ResNet-18		GoogLenet	
Network Variations	top-1	top-5	top-1	top-5
Binary-Weight-Network	60.8	83.0	65.5	86.1
XNOR-Network	51.2	73.2	N/A	N/A
Full-Precision-Network	69.3	89.2	71.3	90.0

DoReFa-Net

Computation of Low-Precision DNN

- Multi-bit linear quantization for DNN
  - Instead of using scaling, quantize with multi-bit representation

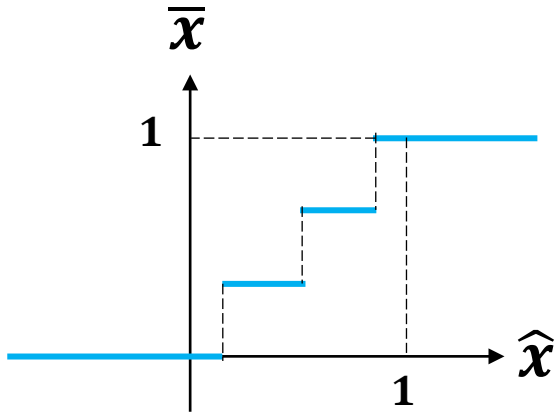
- Vector of M bit $\boldsymbol{x}$ & K bit $\boldsymbol{y}$

- $\boldsymbol{x} = \Sigma_{\mathrm{m}=0}^{\mathrm{M}-1} \mathrm{c}_{\mathrm{m}}(\mathbf{x}) \cdot 2^{\mathrm{m}},\ \mathbf{y} = \Sigma_{k=0}^{K-1} c_k(\boldsymbol{y}) \cdot 2^k$
  - $\boldsymbol{x} \cdot \boldsymbol{y} = \Sigma_{\mathrm{m}=0}^{\mathrm{M}-1} \Sigma_{k=0}^{K-1} 2^{m+k} bit\_count\left[and\left(c_m(\boldsymbol{x}), c_k(\boldsymbol{y})\right)\right]$

  - Replace multiplication by binary operation and pop-count(bit-count) with shift

Straight-Through Estimator for Quantization

- Problem of the quantized network
  - How can we estimate the gradient of the rounding/discontinuous function?
  - Bypass gradient as if there is no rounding function
  - K-bit activation quantization
    - Forward:
      - $\bar{x} = clip(x, 0, 1)$
      - $\hat{x} = \frac{1}{2^k - 1} round\left(\left(2^k - 1\right) \cdot \bar{x}\right)$
    - Backward:
      - $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \bar{x}} I_{|x| \leq 1}$
      - $\frac{\partial L}{\partial \bar{x}} = \frac{\partial L}{\partial \hat{x}}$



DoReFa-Net Weight Quantization

- Use tanh to limit the value range of weights to [-1, 1]
  - K-bit weight quantization
    - Forward:
      - $\bar{x} = \frac{\tanh(x)}{2*\max|\tanh(x)|} + \frac{1}{2}$
      - $\hat{x} = \frac{2}{2^k-1} round\left(\left(2^k-1\right)\cdot \bar{x}\right) - 1$
    - Backward:
      - $\frac{\partial L}{\partial x} = \frac{\partial \bar{x}}{\partial x}\frac{\partial L}{\partial \bar{x}}$
      - $\frac{\partial L}{\partial \bar{x}} = \frac{1}{2}\frac{\partial L}{\partial \hat{x}}$

Results

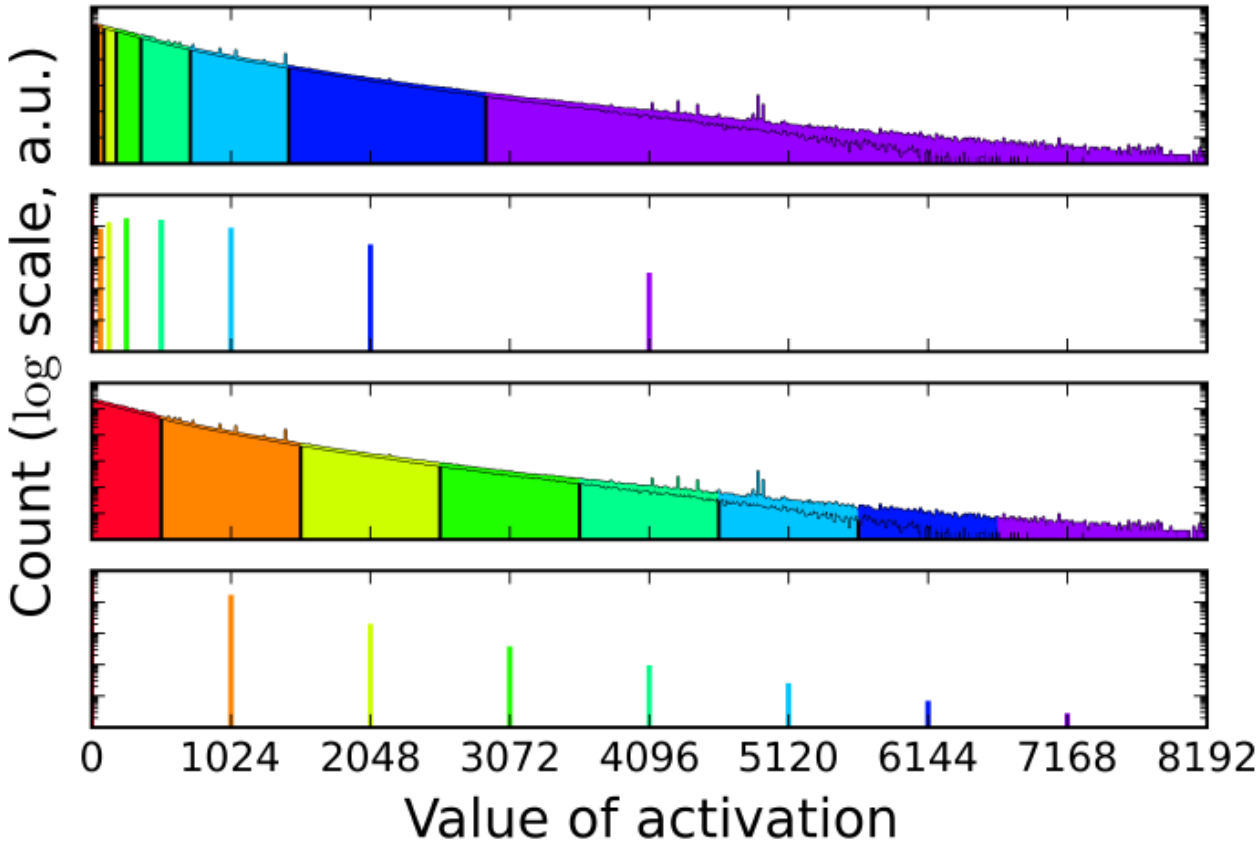
W	A	G	Training Complexity	Inference Complexity	Storage Relative Size	AlexNet Accuracy
1	1	6	7	1	1	0.395
1	1	8	9	1	1	0.395
1	1	32	-	1	1	0.279 (BNN)
1	1	32	-	1	1	0.442 (XNOR-Net)
1	1	32	-	1	1	0.401
1	1	32	-	1	1	0.436 (initialized)
1	2	6	8	2	1	0.461
1	2	8	10	2	1	0.463
1	2	32	-	2	1	0.477
1	2	32	-	2	1	0.498 (initialized)
1	3	6	9	3	1	0.471
1	3	32	-	3	1	0.484
1	4	6	-	4	1	0.482
1	4	32	-	4	1	0.503
1	4	32	-	4	1	0.530 (initialized)
8	8	8	-	-	8	0.530
32	32	32	-	-	32	0.559

Logarithm-based Quantization

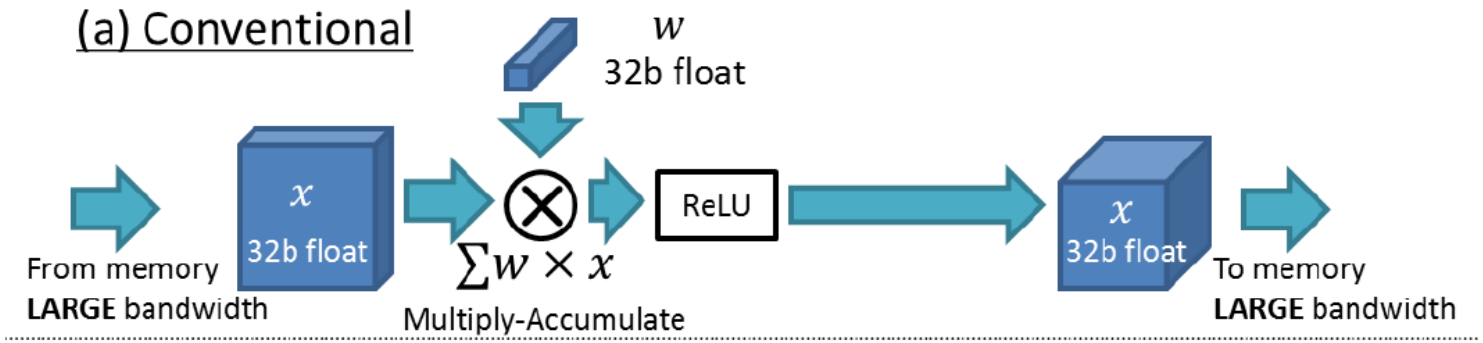
Logarithm-based Quantization (LogQuant)

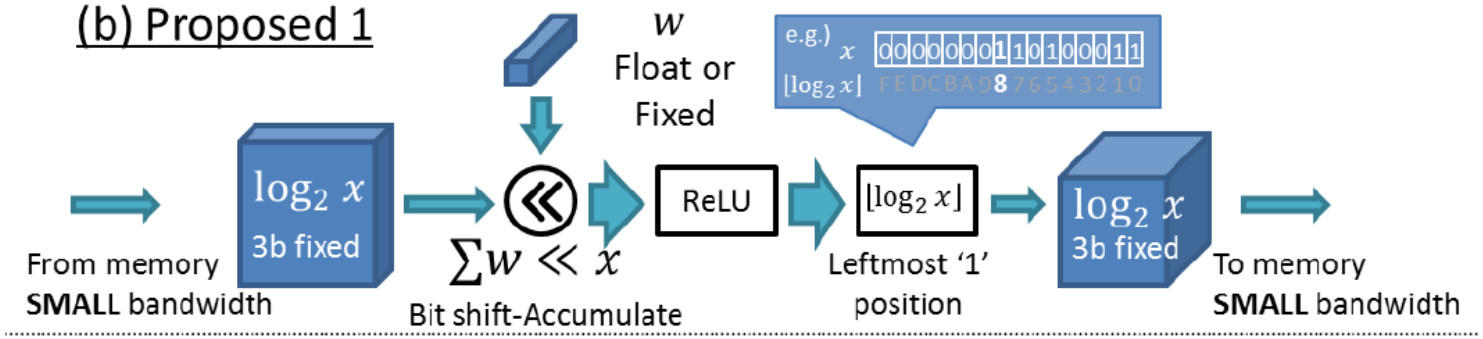
- Motivation
- Less quantization errors for small values

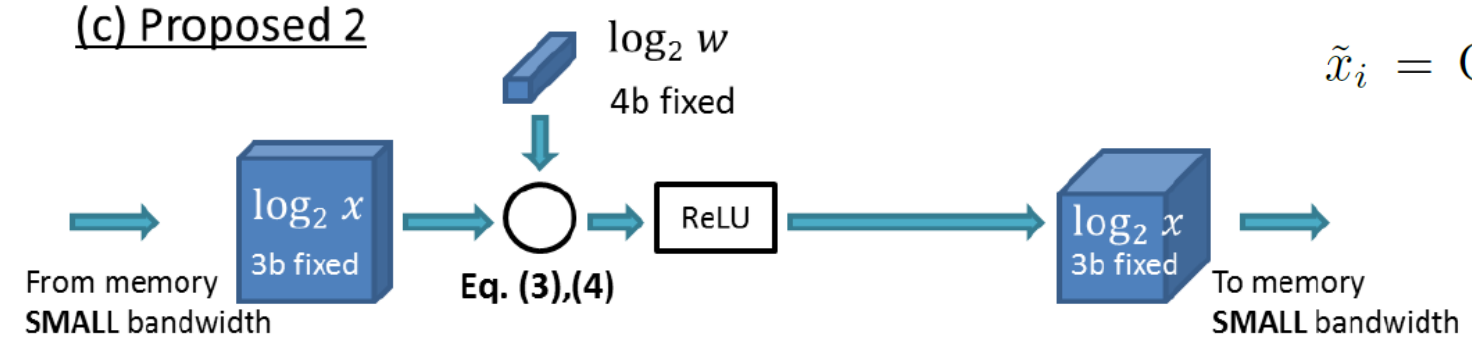
- No need of multiplication
  - b1101 x b0100 = shift(b1101, 2)
  - Shift instead of multiplication



Logarithm-based Quantization (LogQuant)





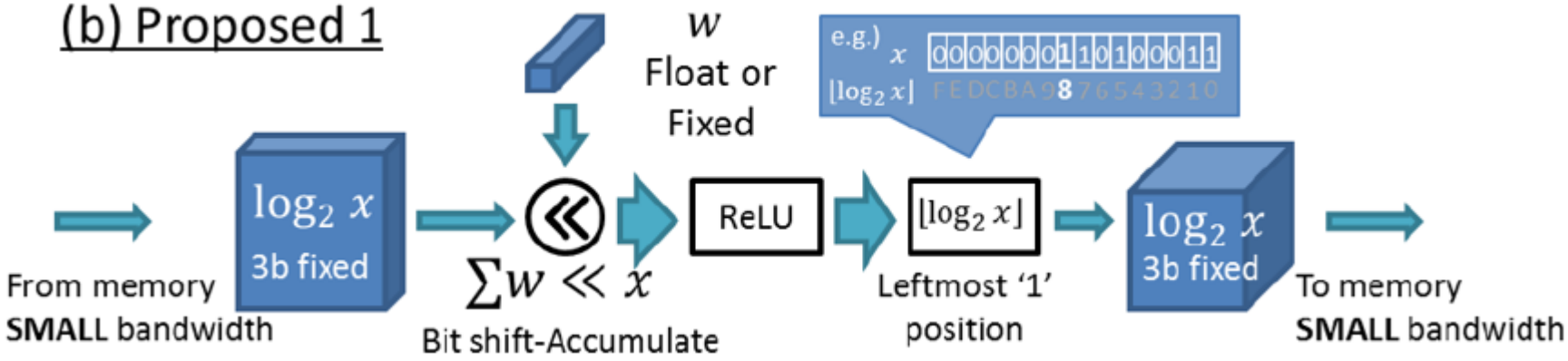


$$w^T x \simeq \sum_{i=1}^{n} w_i \times 2^{\tilde{x}_i}$$

$$= \sum_{i=1}^{n} \text{Bitshift}(w_i, \tilde{x}_i),$$

$$\tilde{x}_i = \text{Quantize}(\log_2(x_i)), \text{ Quantize}(\bullet) \text{ quantizes}$$

Logarithm-based Quantization (LogQuant)



$$w^T x \simeq \sum_{i=1}^{n} w_i \times 2^{\tilde{x}_i}$$

$$= \sum_{i=1}^{n} \text{Bitshift}(w_i, \tilde{x}_i),$$

5.5 * 4.5 = b101.1 * b100.1

~ b101.1 * b100.0 = Bitshift(b101.1, 2)

= b10110 (22 in decimal)

= b10000 (leftmost '1')

24.75 ~ 5.5*4=22 (error = 2.75)

After activation quantization (truncation case)

22 → 16 (b10000)

Fractional Exponent

- $Base^{exponent}$
  - 10^2, 2^3, $2^{1/2}$, …
- Fractional exponent (a.k.a radical or rational exponent) is often used in LogQuant
- How to represent $2^{1/2}$-base number?
  - 2.5 in decimal (10-base) number = $2*10^0 + 5*10^{-1}$
  - 2.5 in binary (2-base) number = $1*2^1 + 0*2^0 + 1*2^{-1} = 10.1$
  - 2.5 in $2^{1/2}$-base number = $1*(2^{1/2})^2+0*(2^{1/2})^1+0*(2^{1/2})^0+0*(2^{-1/2})^1+1*(2^{-1/2})^2 = 100.01$
- How to perform quantization?
  - When truncation is adopted, only leading '1' remains.
  - 2.5 = 100.01 (base $2^{1/2}$) → 100.00 (after quantization)

LogQuant for Weight (5b) and Activation (4b)

- Activations are in 4b log
- LogQuant applied to weights of convolutional layers
- 1.7% (AlexNet) and 0.5% (VGG) loss
  - Better than linear quantization
- Conv weights are sensitive
- Smaller base, $2^{1/2}$ or $2^{1/4}$ works well

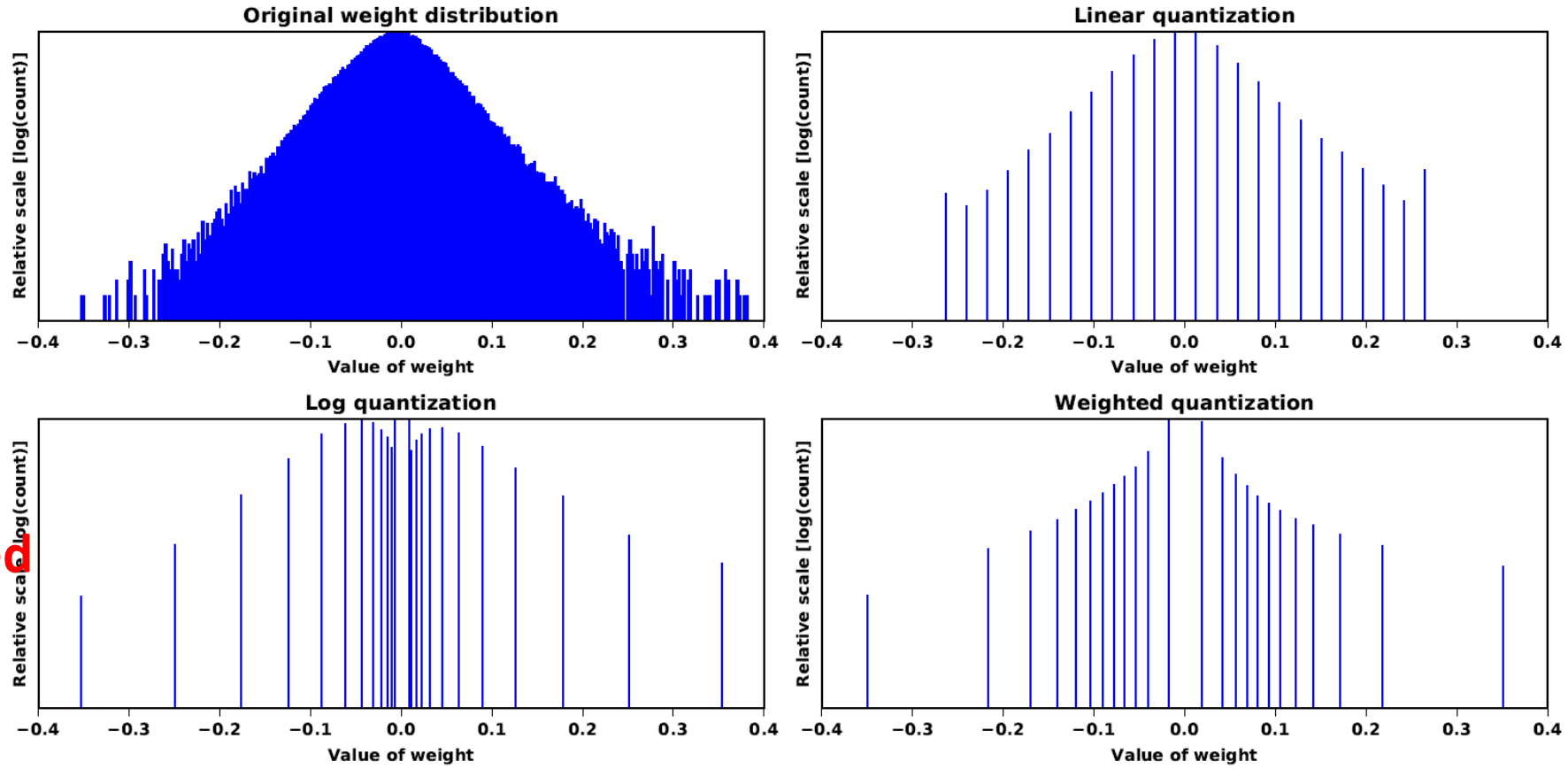
We first observe that the weights of the convolutional layers for AlexNet and VGG16 are more sensitive to quantization than are FC weights. Each FC weight is used only once per image (batch size of 1) whereas convolutional weights are reused many times across the layer's input activation map. Because of this, the quantization error of each weight

Model	Float 32b	Linear 5b	Base-2 Log 5b	Base-$\sqrt{2}$ Log 5b
AlexNet	76.8%	73.6%	70.6%	75.1%
VGG16	89.5%	85.1%	83.4%	89.0%

Weighted-entropy-based Quantization

Weighted Entropy-based Quantization

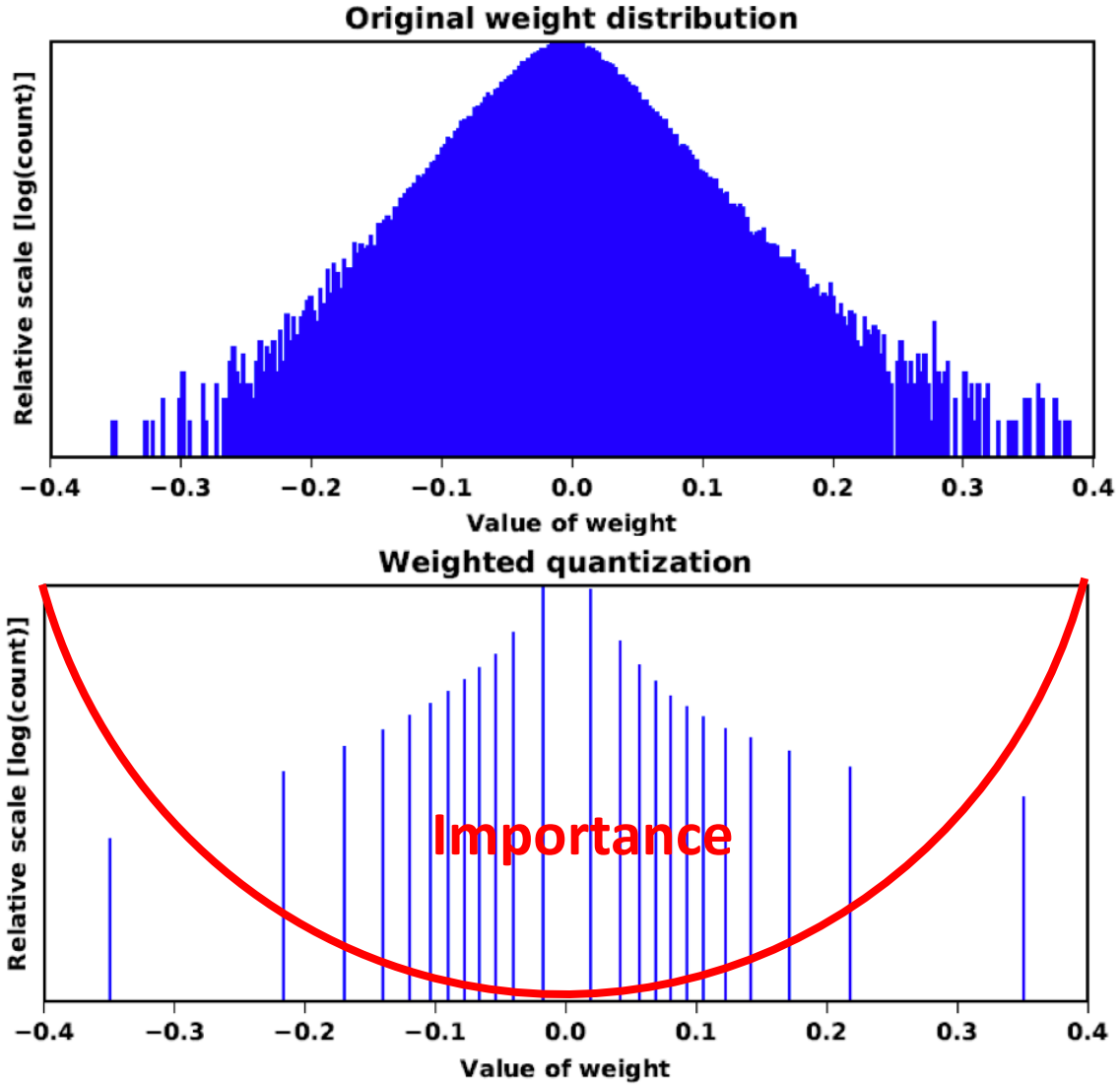
Activation in logarithm-based quantization



Clustered weights

Figure 1. Comparison of various quantization schemes. The weights are extracted from the second 3×3 convolution layer of GoogLeNet [21]. Each quantization scheme is given to assign 24 levels. We use $2^{0.5}$ as the base of LogQuant and optimize both linear quantization and LogQuant towards minimizing the L2 norm of overall activations.

Finding Weight Clustering
Large Weights are Important → Weighted Entropy



Importance ~ Weight magnitude

$$S = -\sum_n I_n P_n \log P_n$$

Weighted entropy

where

$$P_n = \frac{|C_n|}{\sum_k |C_k|} \quad \text{(relative frequency)}$$

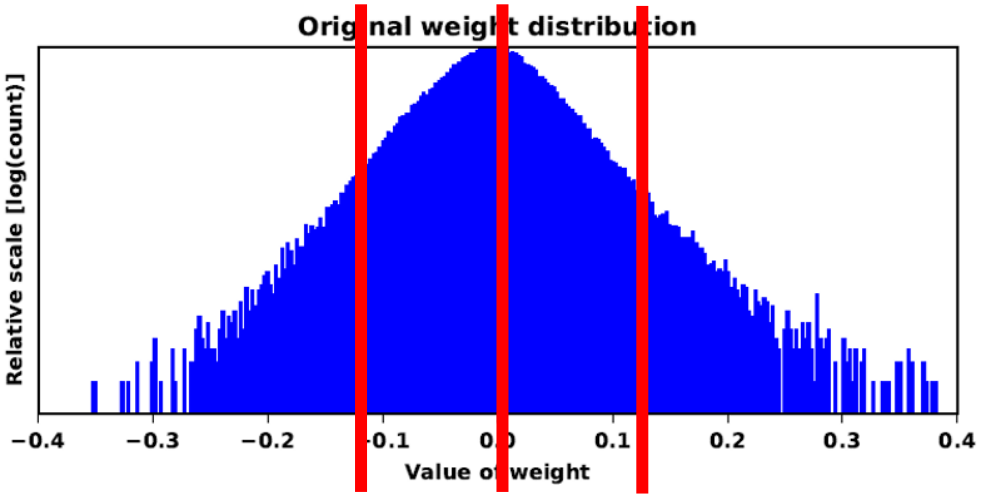
$$I_n = \frac{\sum_m i_{(n,m)}}{|C_n|} \quad \text{(representative importance)}$$

$$i_{(n,m)} = w_{(n,m)}{}^2$$

Algorithm 1 Weight Quantization

function OPTSEARCH(N, w)
  for $k = 0$ to $N_w - 1$ **do**
    $i_k \leftarrow f_i(w_k)$
  $s \leftarrow sort([i_0, \cdots, i_{N_w-1}])$
  $c_0, \cdots, c_N \leftarrow$ initial cluster boundary
  while S is increased **do**
    for $k = 1$ to $N - 1$ **do**
      for $c'_k \in [c_{k-1}, c_{k+1}]$ **do**
        $S' \leftarrow S$ with $c_0, \cdots, c'_k, \cdots, c_N$
        if $S' > S$ **then**
          $c_k \leftarrow c'_k$
  for $k = 0$ to $N - 1$ **do**
    $I_k \leftarrow \sum_{i=c_k}^{c_{k+1}-1} s[i]/(c_{k+1} - c_k)$
    $r_k \leftarrow f_i^{-1}(I_k)$
    $b_k \leftarrow f_i^{-1}(s[c_k])$
  $b_N \leftarrow \infty$
  return $[r_0 : r_{N-1}], [b_0 : b_N]$
function QUANTIZE($w_n, [r_0 : r_{N-1}], [b_0 : b_N]$)
  return r_k for k s.t. $b_k \leq w_n < b_{k+1}$

- N: The number of levels
- N_w: The number of weights
- w_n: Value of n-th weight
- i_n: Importance of n-th weight
- f_i: Importance mapping function
- c_i: Cluster boundary index
- S: Overall weighted entropy



clusters (N = 4)

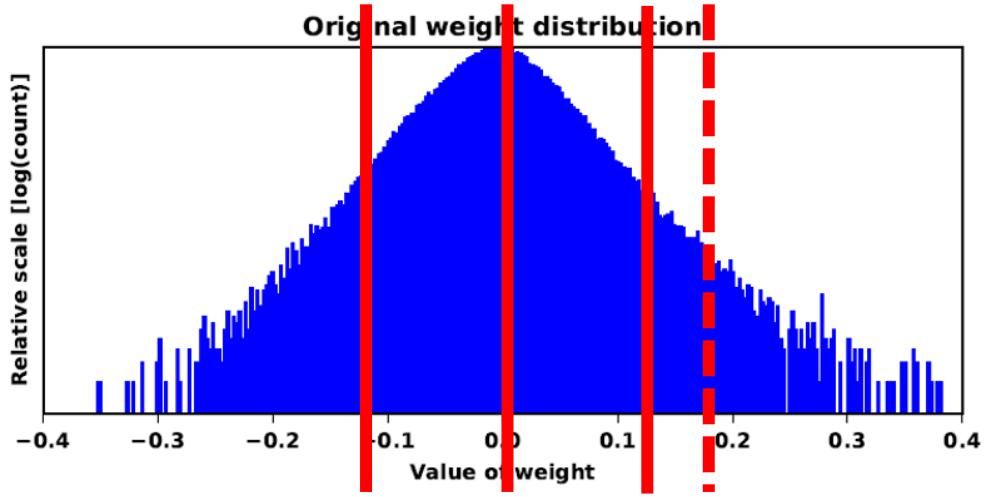
Algorithm 1 Weight Quantization

function OPTSEARCH(N, w)
  for $k = 0$ to $N_w - 1$ **do**
    $i_k \leftarrow f_i(w_k)$
  $s \leftarrow sort([i_0, \cdots, i_{N_w-1}])$
  $c_0, \cdots, c_N \leftarrow$ initial cluster boundary
  while S is increased **do**
    for $k = 1$ to $N - 1$ **do**
      for $c'_k \in [c_{k-1}, c_{k+1}]$ **do**
        $S' \leftarrow S$ with $c_0, \cdots, c'_k, \cdots, c_N$
        if $S' > S$ **then**
          $c_k \leftarrow c'_k$
  for $k = 0$ to $N - 1$ **do**
    $I_k \leftarrow \sum_{i=c_k}^{c_{k+1}-1} s[i]/(c_{k+1} - c_k)$
    $r_k \leftarrow f_i^{-1}(I_k)$
    $b_k \leftarrow f_i^{-1}(s[c_k])$
  $b_N \leftarrow \infty$
  return $[r_0 : r_{N-1}], [b_0 : b_N]$
function QUANTIZE($w_n, [r_0 : r_{N-1}], [b_0 : b_N]$)
  return r_k for k s.t. $b_k \leq w_n < b_{k+1}$

- N: The number of levels
- N_w: The number of weights
- w_n: Value of n-th weight
- i_n: Importance of n-th weight
- f_i: Importance mapping function
- c_i: Cluster boundary index
- S: Overall weighted entropy

Choose boundaries which maximize S

clusters (N = 4)



$$S = -\sum_n I_n P_n \log P_n$$

Weighted entropy

where

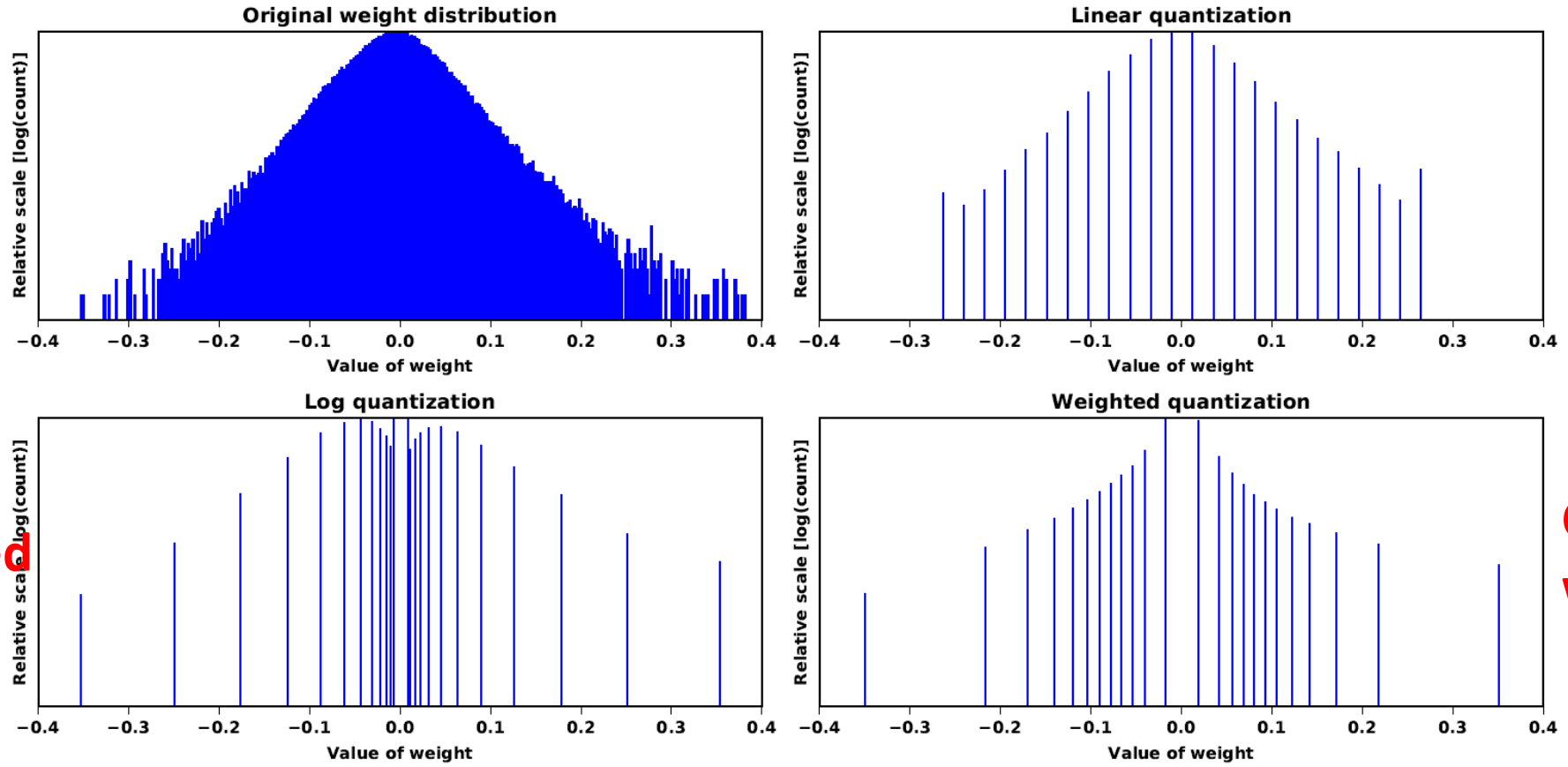
$$P_n = \frac{|C_n|}{\sum_k |C_k|} \qquad \text{(relative frequency)}$$

$$I_n = \frac{\sum_m i_{(n,m)}}{|C_n|} \qquad \text{(representative importance)}$$

$$i_{(n,m)} = w_{(n,m)}{}^2$$

Weighted Entropy-based Quantization

Activation in logarithm-based quantization



Clustered weights

Figure 1. Comparison of various quantization schemes. The weights are extracted from the second 3×3 convolution layer of GoogLeNet [21]. Each quantization scheme is given to assign 24 levels. We use $2^{0.5}$ as the base of LogQuant and optimize both linear quantization and LogQuant towards minimizing the L2 norm of overall activations.

Quantization Results: AlexNet and GoogLeNet

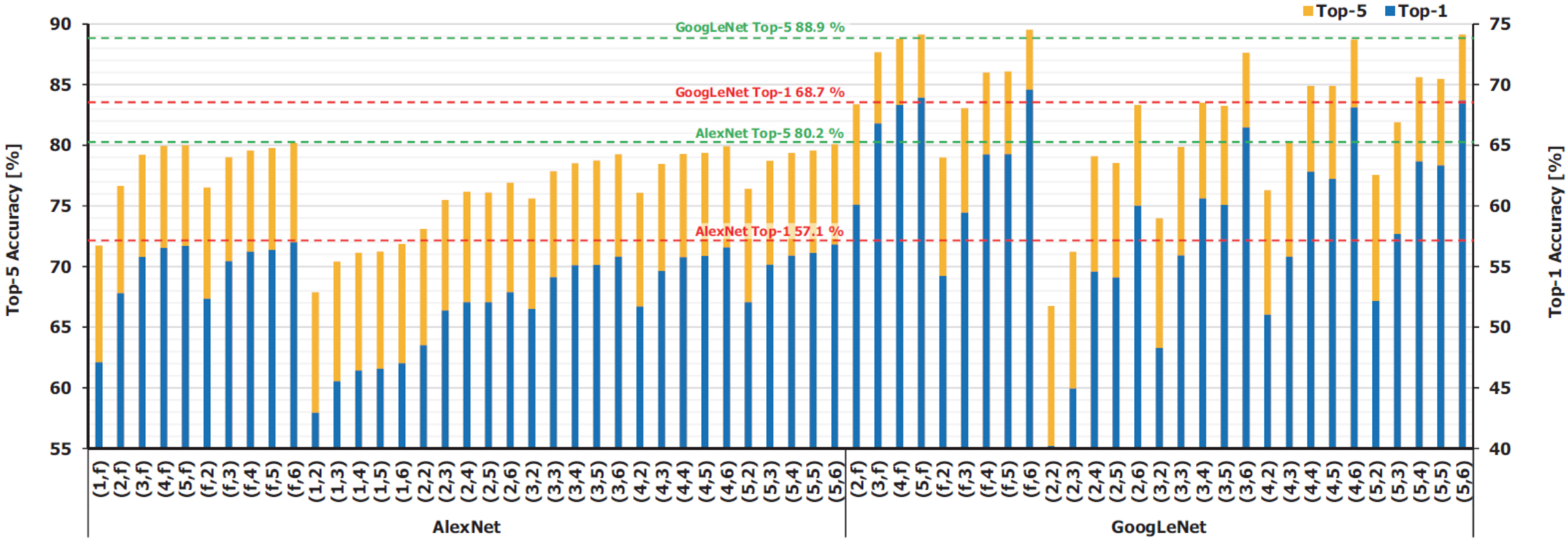
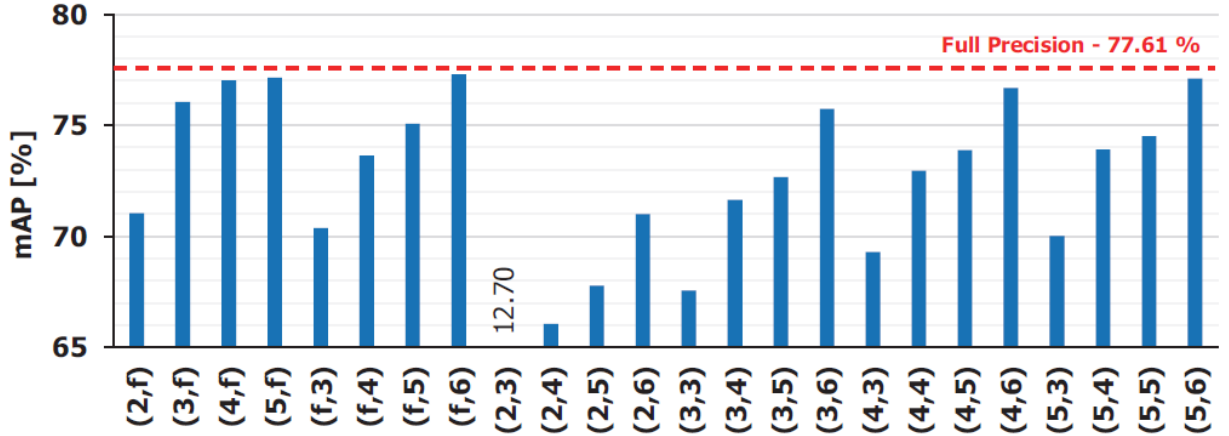


Figure 2. Top-1 and top-5 accuracy of quantized CNNs after fine-tuning. The dashed lines represent the accuracy of the baseline networks, which use full-precision arithmetic.

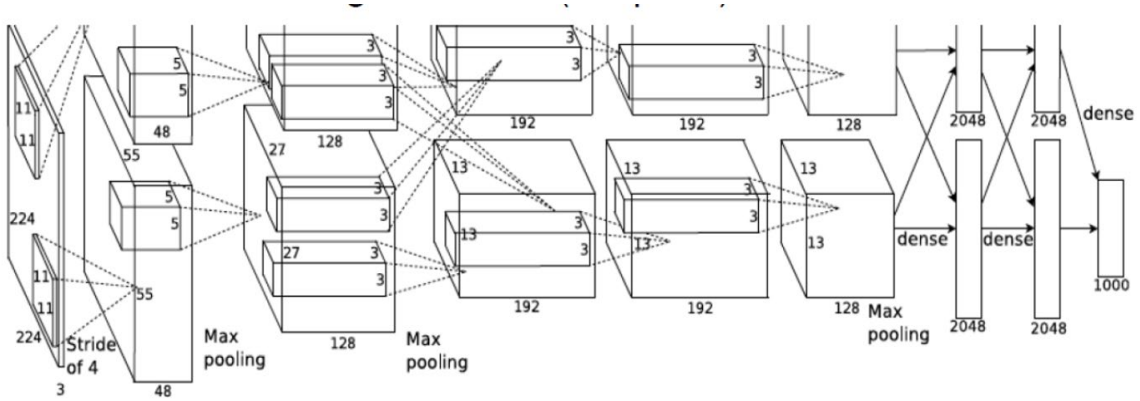
R-FCN and Per-layer Bitwidth

- R-FCN (ResNet-50)
  - (5,6) gives <1% loss in mAP

- Different per-layer bitwidths in AlexNet
  - Convex case, i.e., larger bitwidth near input and output gives the best accuracy



	DEC	INC	CONCAVE	CONVEX
Top-1 [%]	53.79	50.35	54.45	54.33
Top-5 [%]	77.59	74.89	76.43	78.20



6bit 5bit 4bit 5bit 6bit

Memory Benefits

- Ours can give significant reduction in the size of activation

	Weights			Activations	Top-1
	P [%]	**Q** [MB]	**+H**	**Q** [MB]	[%]
WQ(4,4)	-	30.5	18.1	0.47	55.8
WQ(2,3)	-	15.3	12.5	0.35	53.7
XNOR-Net [19]	-	23.7	-	0.72	44.2
DoReFa-Net [25]	-	23.6	-	0.47	53.0
Deep Compression [10]	11	8.9	6.9	3.75	57.2
[10] + WQ(4,6)	11	8.3	6.5	0.70	56.3

Table 1. Memory requirement comparison with AlexNet (P: Pruning ratio, Q: Quantization, H: Huffman encoding).

NVIDIA 8-bit Quantization

Low-precision Speed Up

Input Type	Accumulation Type	Relative math throughput	Bandwidth savings
FP16	FP16	8x	2x
INT8	INT32	16x	4x
INT4	INT32	32x	8x
INT1	INT32	128x	32x

Input size 224x224 for all, except 299x299 for Inception networks

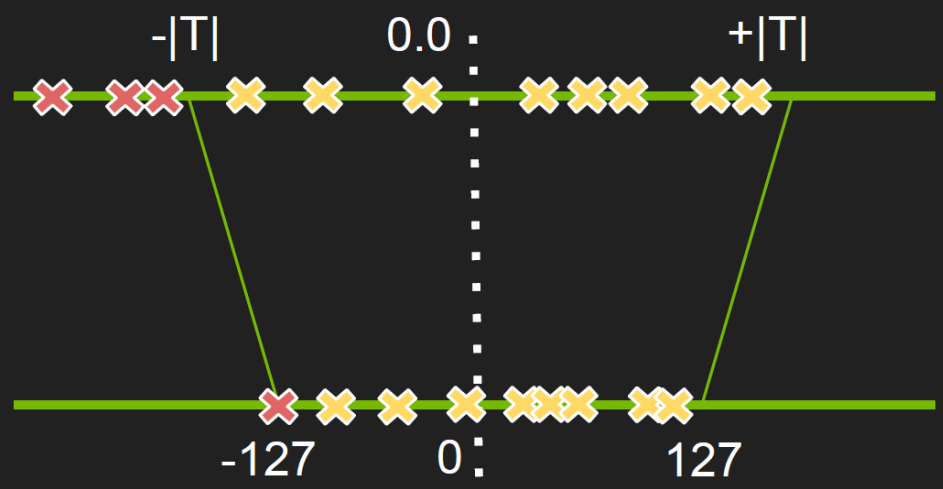
	Batch size 1			Batch size 8			Batch size 128		
	FP32	FP16	Int8	FP32	FP16	Int8	FP32	FP16	Int8
MobileNet v1	1	1.91	2.49	1	3.03	5.50	1	3.03	6.21
MobileNet v2	1	1.50	1.90	1	2.34	3.98	1	2.33	4.58
ResNet50 (v1.5)	1	2.07	3.52	1	4.09	7.25	1	4.27	7.95
VGG-16	1	2.63	2.71	1	4.14	6.44	1	3.88	8.00
VGG-19	1	2.88	3.09	1	4.25	6.95	1	4.01	8.30
Inception v3	1	2.38	3.95	1	3.76	6.36	1	3.91	6.65
Inception v4	1	2.99	4.42	1	4.44	7.05	1	4.59	7.20
ResNext101	1	2.49	3.55	1	3.58	6.26	1	3.85	7.39

Quantization

- No saturation: map |max| to 127

- Significant accuracy loss, in general

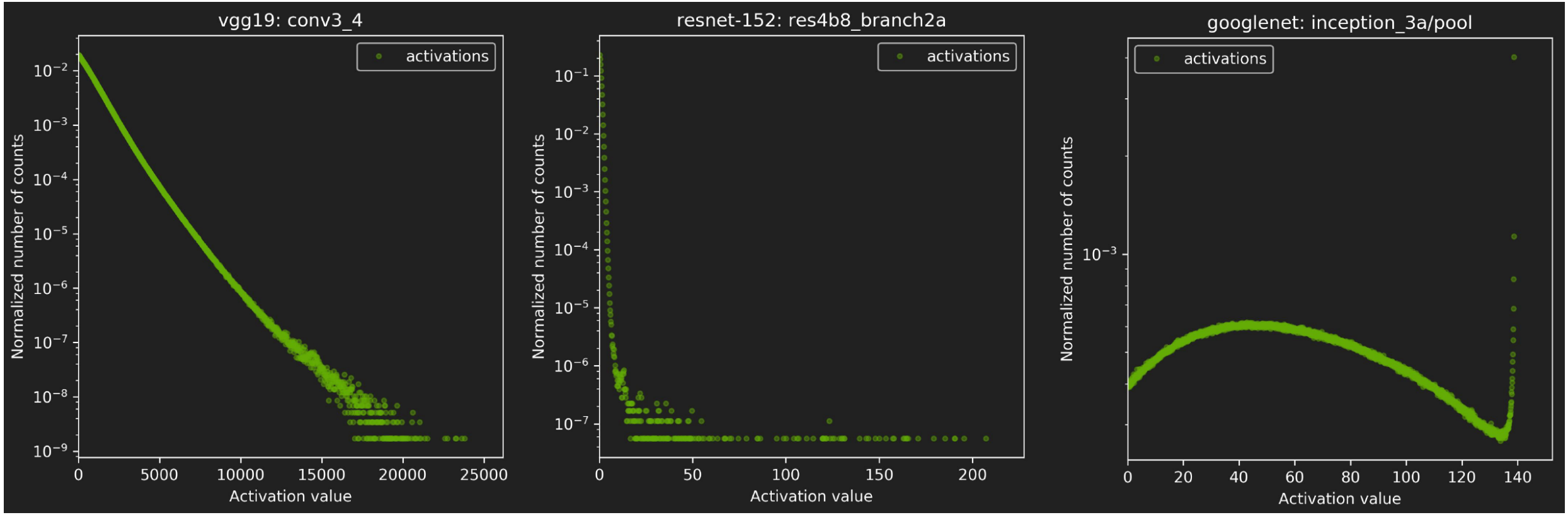
- Saturate above |threshold| to 127



- Weights: no accuracy improvement
- Activations: improved accuracy
- Which |threshold| is optimal?

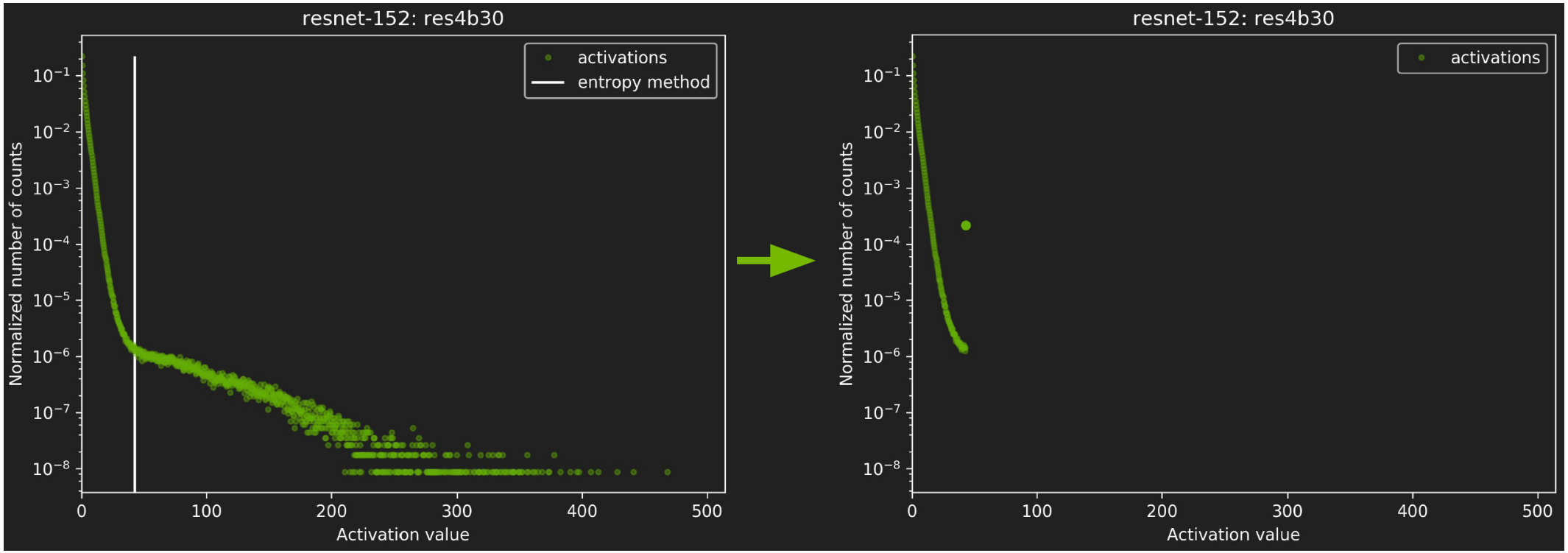
Quantization for int8 in NVIDIA TensorRT

- Examples of the distributions of activation



Quantization for int8 in NVIDIA TensorRT

- Basic idea of determining the threshold
  - Minimize information loss in the distribution of quantized data
  - KL divergence is used as the metric of information loss



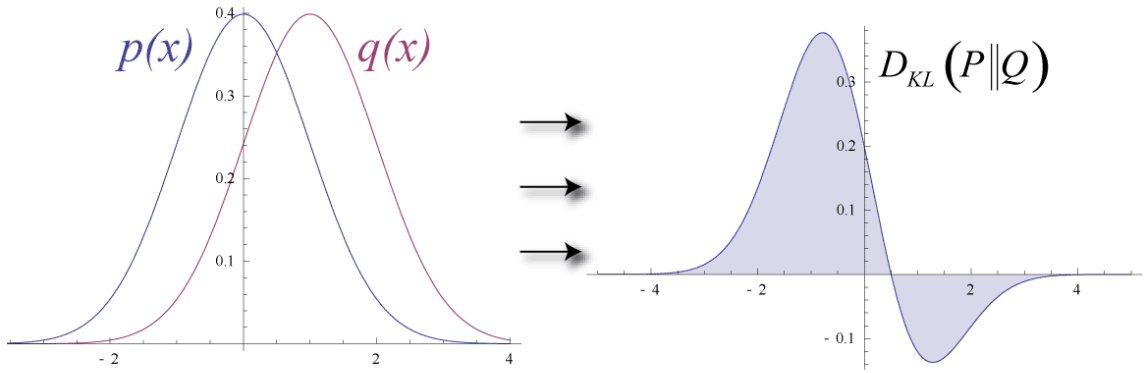
Kullback-Leibler (KL) Divergence

- A measure of how one probability distribution diverges from a second expected probability distribution
- The amount of information lost when *Q* is used to approximate *P*

$$D_{\mathrm{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x$$

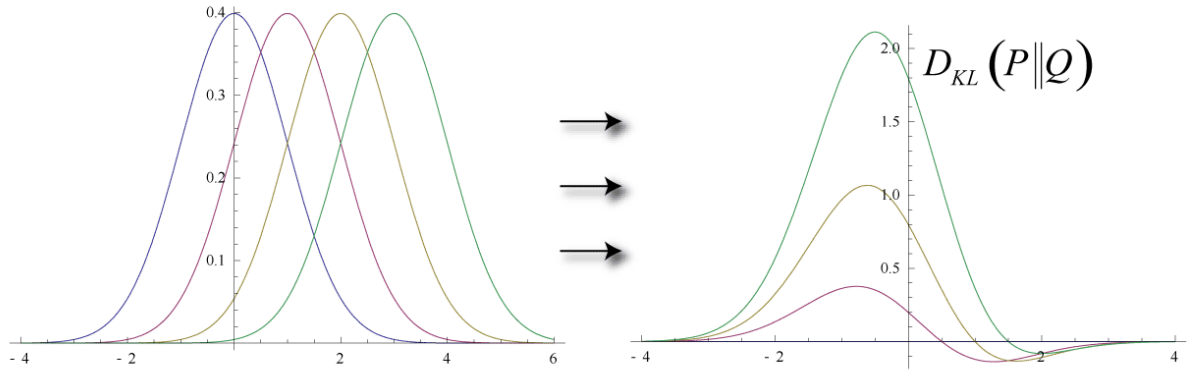
$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= -\sum_x p(x) \log q(x) + \sum_x p(x) \log p(x) \\ &= H(P,Q) - H(P) \end{aligned}$$

- The more similar distributions, the smaller KL divergence



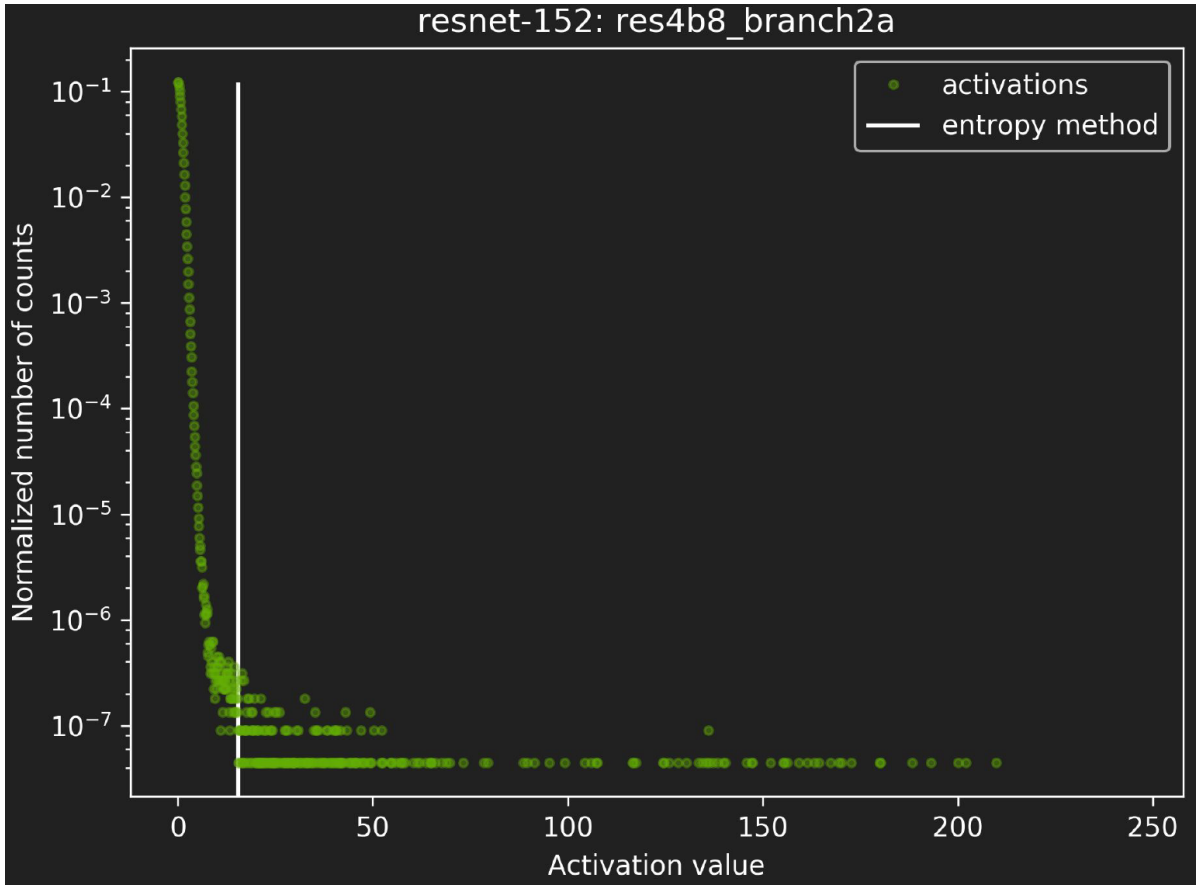
Original Gaussian PDF's

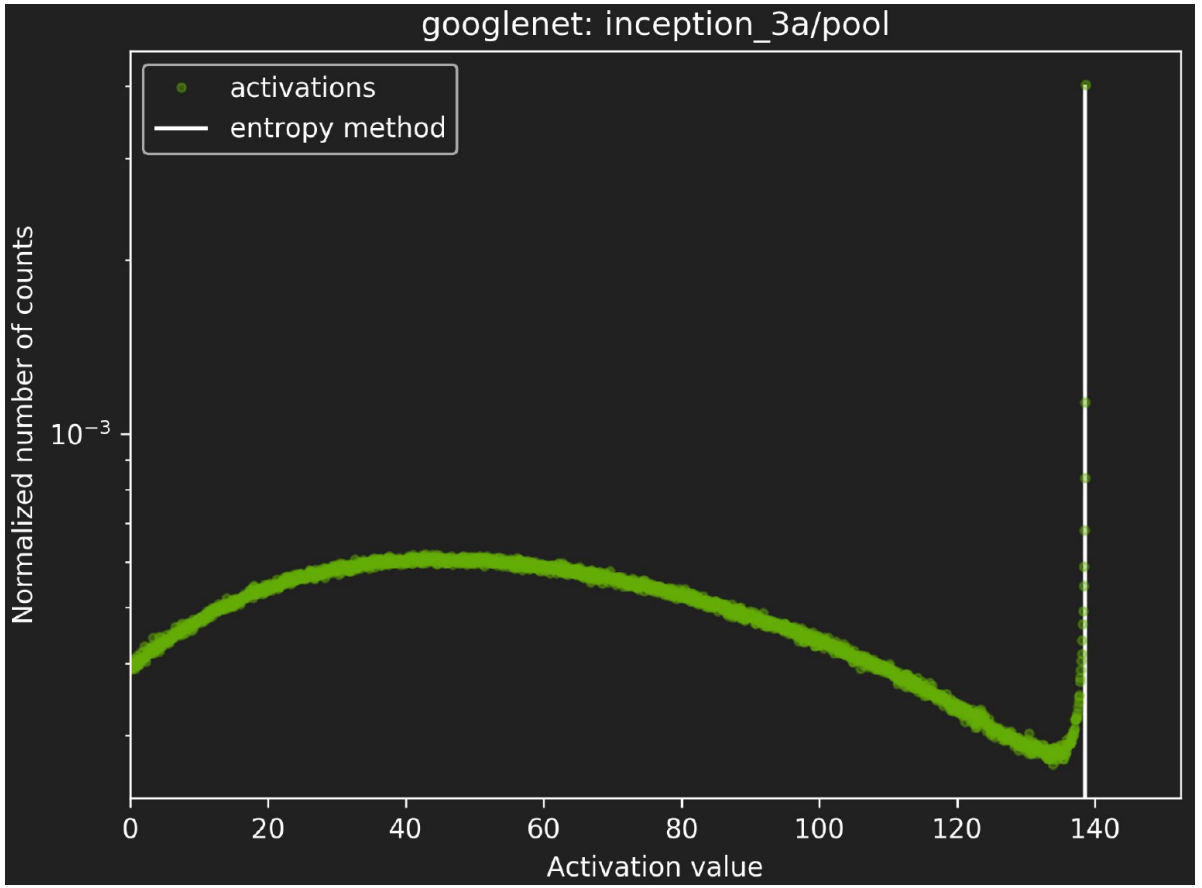
KL Area to be Integrated



Quantization for int8 in NVIDIA TensorRT

- Quantization examples





Quantization for int8 in NVIDIA TensorRT

	FP32		INT8					
			Calibration using 5 batches		Calibration using 10 batches		Calibration using 50 batches	
NETWORK	Top1	Top5	Top1	Top5	Top1	Top5	Top1	Top5
Resnet-50	73.23%	91.18%	73.03%	91.15%	73.02%	91.06%	73.10%	91.06%
Resnet-101	74.39%	91.78%	74.52%	91.64%	74.38%	91.70%	74.40%	91.73%
Resnet-152	74.78%	91.82%	74.62%	91.82%	74.66%	91.82%	74.70%	91.78%
VGG-19	68.41%	88.78%	68.42%	88.69%	68.42%	88.67%	68.38%	88.70%
Googlenet	68.57%	88.83%	68.21%	88.67%	68.10%	88.58%	68.12%	88.64%
Alexnet	57.08%	80.06%	57.00%	79.98%	57.00%	79.98%	57.05%	80.06%
NETWORK	Top1	Top5	Diff Top1	Diff Top5	Diff Top1	Diff Top5	Diff Top1	Diff Top5
Resnet-50	73.23%	91.18%	0.20%	0.03%	0.22%	0.13%	0.13%	0.12%
Resnet-101	74.39%	91.78%	-0.13%	0.14%	0.01%	0.09%	-0.01%	0.06%
Resnet-152	74.78%	91.82%	0.15%	0.01%	0.11%	0.01%	0.08%	0.05%
VGG-19	68.41%	88.78%	-0.02%	0.09%	-0.01%	0.10%	0.03%	0.07%
Googlenet	68.57%	88.83%	0.36%	0.16%	0.46%	0.25%	0.45%	0.19%
Alexnet	57.08%	80.06%	0.08%	0.08%	0.08%	0.07%	0.03%	-0.01%

TensorRT 2.1, all optimizations enabled. ILSVRC2012 validation dataset, batch = 25 images.
Accuracy was measured on 500 batches which were not used for the calibration.