Review

- BinaryConnect (1-bit weight)
  - Update real value weights by using gradients of binary weights

- Binarized Neural Network (1-bit weight and 1-bit activation)
  - Use **straight-through estimator (STE)** for back propagation
  - Update real value weights by using gradients of binary weights

- XNOR-NET (1-bit weight + real value scalar and 1-bit activation + real value scalar)
  - Quantize CNN that works well on ImageNet
  - Real value scaling factor becomes L1 norm
  - Update real value weights by using gradients of real value weights (STE!)
  - Computation overhead reduction for input value scaling factors

DNN Quantization (2)

Lecture 7

Hyung-Sin Kim

SNU Graduate School of Data Science

Let’s step forward!

Not just for heavy redundant models (AlexNet, VGG…)
but for ***already efficient models*** *(MobileNet)!*

Integer-arithmetic-only inference! (edge TPU)

Integer-only [CVPR'18] – Concept

- 1-bit quantization is attractive but degrades performance a lot…
  - Instead, let's use 8-bit integers mainly and a few 32-bit integers

- No real values at all during **inference**
  - Outputs of conv and batchnorm are all integers!
  - It can be executed on integer-arithmetic-only hardware, such as edge TPU

Integer-only [CVPR'18] – Quantization Scheme

$$r = \mathbf{S}(q - \mathbf{Z})$$

Real value (32-bit real)

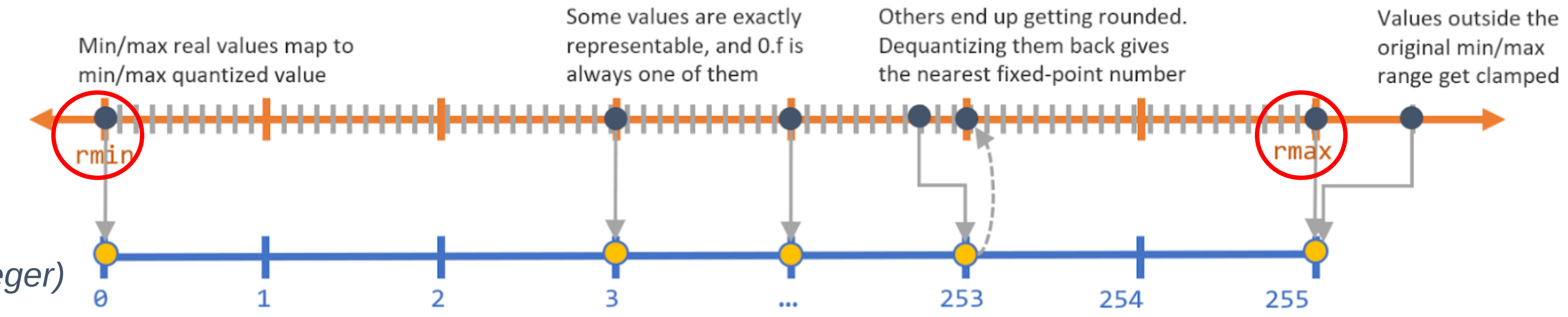
Scaling parameter (32-bit real)

Quantized value (B-bit integer)

Offset parameter (B-bit integer)

$$S = \frac{r_{max} - r_{min}}{2^B - 1},\ Z = \left\lfloor -\frac{r_{min}}{S} \right\rceil_B$$

r (real)



q (8-bit integer)

*We want to represent a **real value r** by using an **integer q!***

*Impossible to represent $[-\infty, \infty]$… So a real value is first **clipped** $[r_{min}, r_{max}]$*

Integer-only [CVPR’18] – Quantization Scheme

- Each weight or activation **array** at each layer is represented as below
  - To quantize an array, we need to have <u>one real value</u> and <u>one integer value</u>
  - A different parameter set is needed for each weight array and activation array

$$\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix} = \boldsymbol{S}\left(\begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{bmatrix} - \boldsymbol{Z}\right)$$

Real array
(32-bit real values)

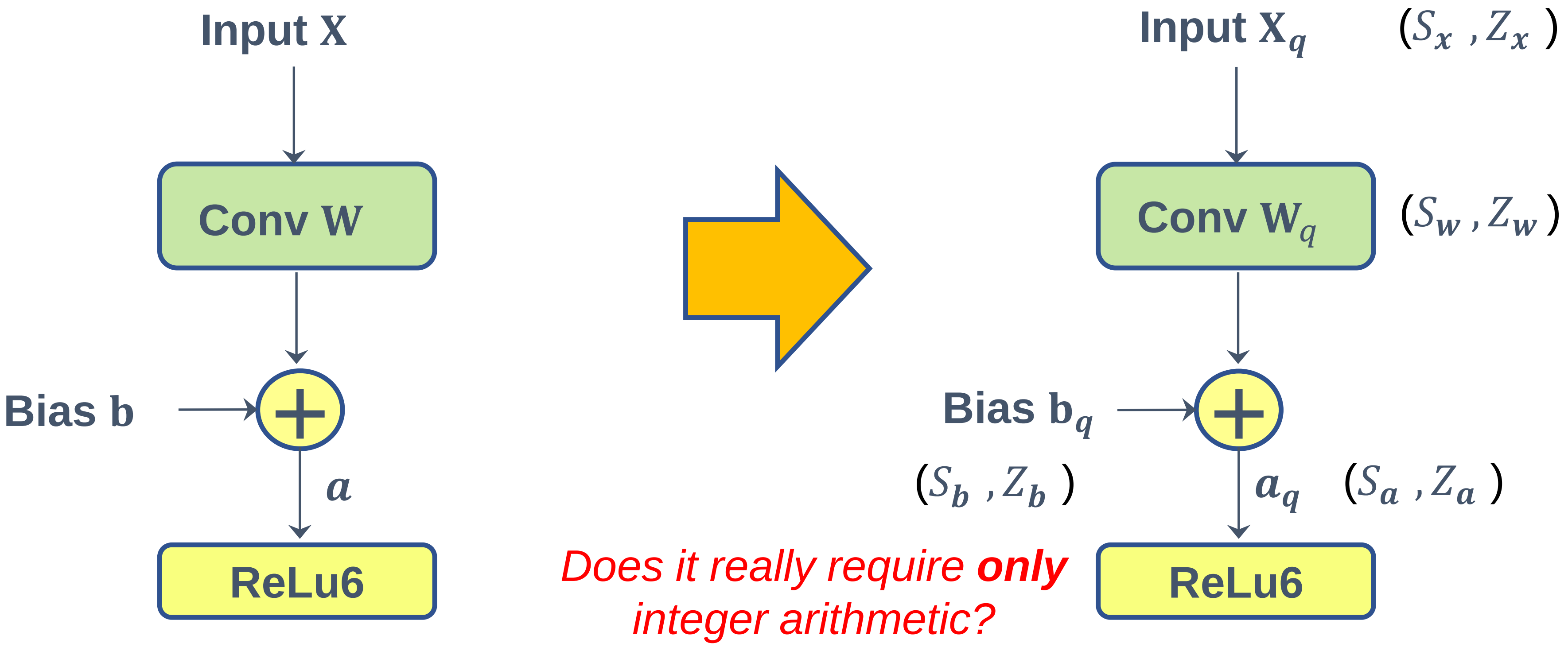
Scaling parameter
*(a **single** 32-bit real)*

Quantized array
(B-bit integers)

Offset parameter
*(a **single** B-bit integer)*

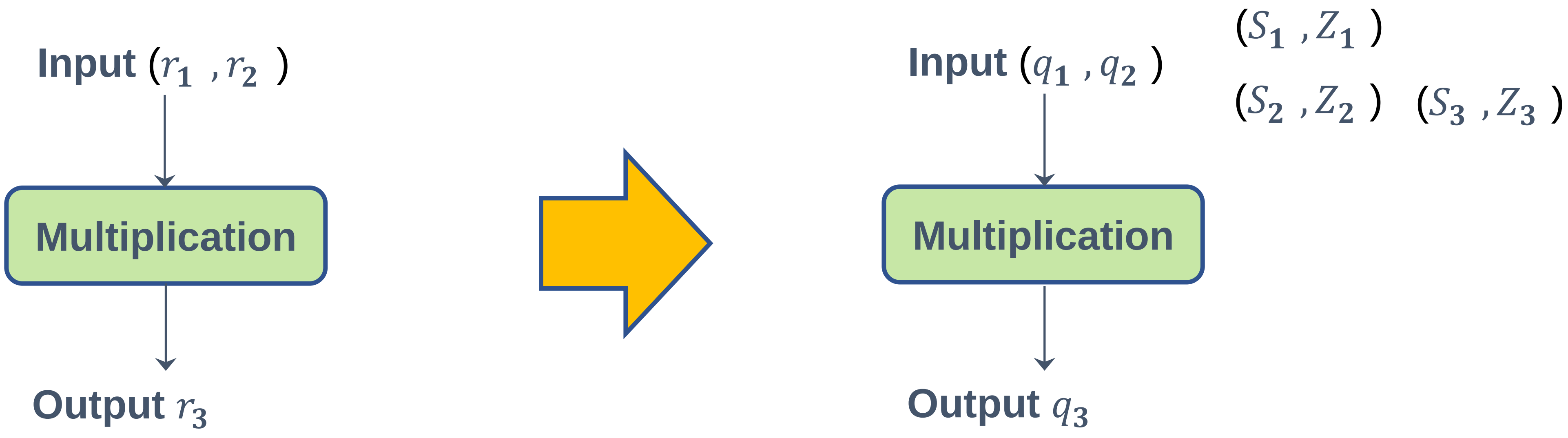
Integer-only [CVPR'18] – Quantized Inference

- We first focus on inference framework
  - Assume that quantization parameters $\boldsymbol{S}$ and $\boldsymbol{Z}$ are already trained for each weight array and activation array
  - The goal is to use **zero** real value calculation



Integer-only [CVPR'18] – Quantized Inference

- For $r_3 = r_1\, r_2$ where $r_\alpha \in \mathbf{R}^{N\times N}$ (matrix multiplication), what is $q_3^{(i,j)}$ for $r_3^{(i,j)}$?
  - Recall that we already have $(S_1\ , Z_1\ )$, $(S_2\ , Z_2\ )$, and $(S_3\ , Z_3\ )$



Integer-only [CVPR'18] – Quantized Inference

- For $r_3 = r_1 \, r_2$ where $r_\alpha \in \mathbf{R}^{N \times N}$ (matrix multiplication), what is $q_3^{(i,j)}$ for $r_3^{(i,j)}$?
  - Recall that we already have $(S_1\ , Z_1\ )$, $(S_2\ , Z_2\ )$, and $(S_3\ , Z_3\ )$
  - $r_\alpha^{(i,j)} = S_\alpha(q_\alpha^{(i,j)} - Z_\alpha)$
  - $S_3(q_3^{(i,k)} - Z_3) = \sum_{j=1}^{N} S_1(q_1^{(i,j)} - Z_1) S_2(q_2^{(j,k)} - Z_2)$

$$\begin{bmatrix} \boxed{r_3^{(1,1)}} & \cdots & r_3^{(1,N)} \\ \vdots & \ddots & \vdots \\ r_3^{(N,1)} & \cdots & r_3^{(N,N)} \end{bmatrix} = \begin{bmatrix} r_1^{(1,1)} & \cdots & r_1^{(1,N)} \\ \vdots & \ddots & \vdots \\ r_1^{(N,1)} & \cdots & r_1^{(N,N)} \end{bmatrix} \begin{bmatrix} r_2^{(1,1)} & \cdots & r_2^{(1,N)} \\ \vdots & \ddots & \vdots \\ r_2^{(N,1)} & \cdots & r_2^{(N,N)} \end{bmatrix}$$

  - $q_3^{(i,k)} = Z_3 + M \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2)$ where $\boxed{M := \frac{S_1 S_2}{S_3}}$ (computed offline)
    - **Problem**: Empirically found that ***M*** is always in the interval (0,1) – **not** represented as an integer
    - **Solution**: Scale up -> calculation -> scale down
  - $q_3^{(i,k)} = Z_3 \ \underbrace{+\ 2^{-n}}_{\text{bit shift}} \left\{ \underbrace{(2^n\ M)}_{\text{int32}}\ \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1\ )(q_2^{(j,k)} - Z_2\ ) \right\}$

Integer only multiplication!

Integer-only [CVPR'18] – Quantized Inference

- **Full convolution**: $a^{(i,k)} = \sum_{j=1}^{N} x^{(i,j)} w^{(j,k)} + b^{(i,k)}$
- $a_q^{(i,k)} = Z_a + \frac{S_x S_w}{S_a} \sum_{j=1}^{N} (x_q^{(i,j)} - Z_x)(w_q^{(j,k)} - Z_w) + \frac{\overbrace{S_b}^{= S_x S_w}}{S_a} (b_q^{(i,k)} - \overbrace{Z_b}^{= 0})$

Pre-compute and store as an integer

*A single multiplication output needs only int16, but **int32** is needed for accumulation*

$$\underbrace{= Z_a + \underbrace{2^{-n} \left\{ \underbrace{\boxed{\left(2^n \frac{S_x S_w}{S_a}\right)}}_{\text{int32}} \left(\underbrace{b_q^{(i,k)}}_{\text{int32}} + \underbrace{\sum_{j=1}^{N} (\underbrace{x_q^{(i,j)}}_{\text{uint8}} - Z_x)(\underbrace{w_q^{(j,k)}}_{\text{uint8}} - Z_w)}_{\text{int32}} \right) \right\}}_{\text{int8}}}_{\text{uint8}}$$

Integer-only [CVPR'18] – Quantized Inference

- Computation overhead due to zero-points
  - $a_q^{(i,k)} = M \sum_{j=1}^{N} \boxed{x_q^{(i,j)} w_q^{(j,k)}}$ vs. $a_q^{(i,k)} = Z_a + M \sum_{j=1}^{N} \boxed{(x_q^{(i,j)} - Z_x)(w_q^{(j,k)} - Z_w)}$
  - We need to calculate N×N $a_q^{(i,k)}$ for all i and k
  - 2N subtractions (j) × # of output elements (N × N, i and k) = $2N^3$ more subtractions?

- Handling zero-points by **unfolding**
  - $a_q^{(i,k)} = Z_a + M\left(NZ_x Z_w - Z_x \boxed{\sum_{j=1}^{N} w_q^{(j,k)}} - Z_w \boxed{\sum_{j=1}^{N} x_q^{(i,j)}} + \boxed{\sum_{j=1}^{N} x_q^{(i,j)} w_q^{(j,k)}}\right)$

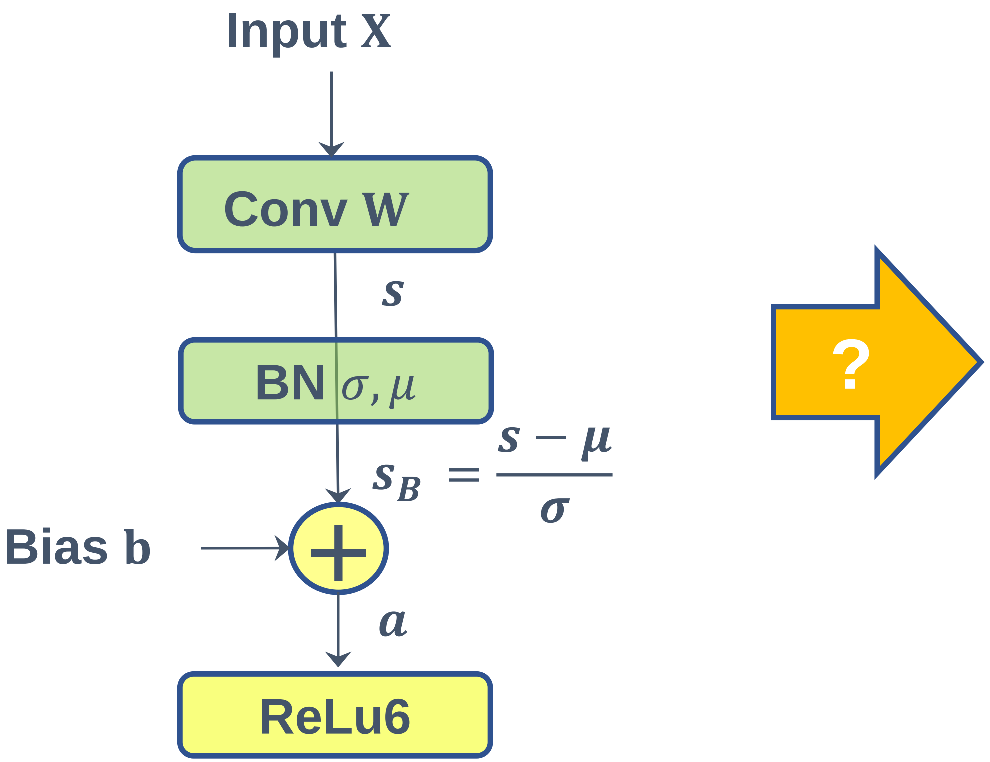
Independent from i
(can be reused for all i)
N^2 *additions*

Independent from k
(can be reused for all k)
N^2 *additions*

Main computation burden (just same as without zero points!)

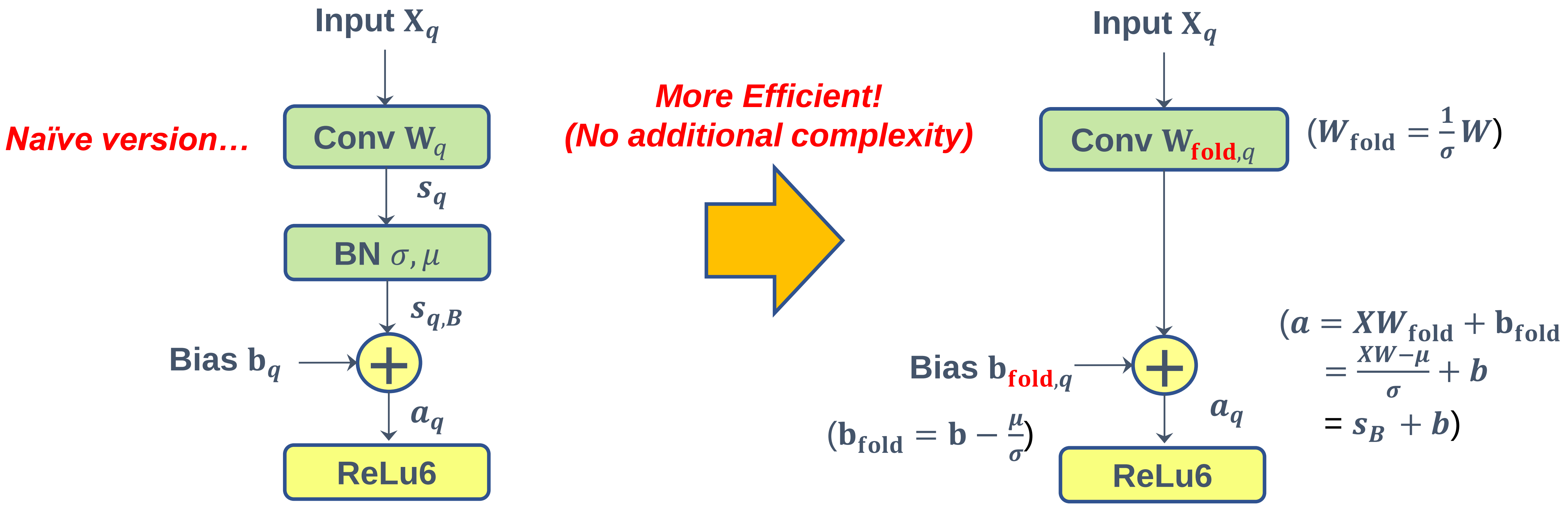
Integer-only [CVPR'18] – Quantized Inference

- Include batch normalization
  - Assume that Bnorm parameters $\boldsymbol{\sigma}$ and μ are already trained for each activation array
  - Again, the goal is to use **zero** real value calculation



Integer-only [CVPR'18] – Quantized Inference

- Include batch normalization
  - Assume that Bnorm parameters $\boldsymbol{\sigma}$ and μ are already trained for each activation array
  - Again, the goal is to use **zero** real value calculation

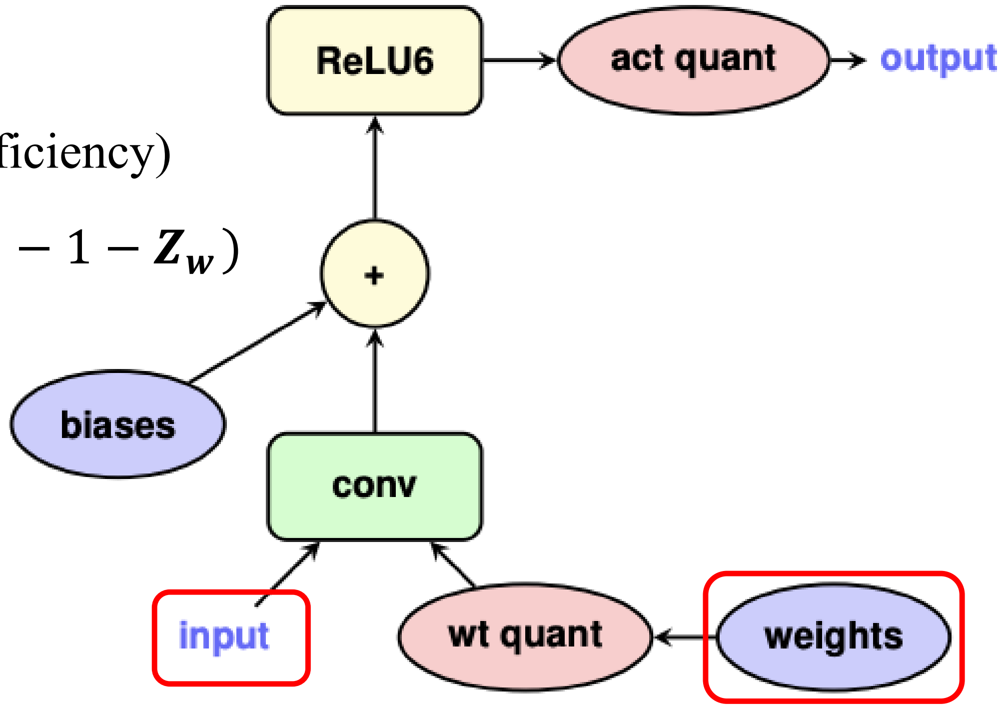


How can we train a DNN
so that the integer-arithmetic-only inference is possible?

Need to train ***weights*** *and also* ***quantization parameters****!*

Integer-only [CVPR'18] – Quant-aware Training

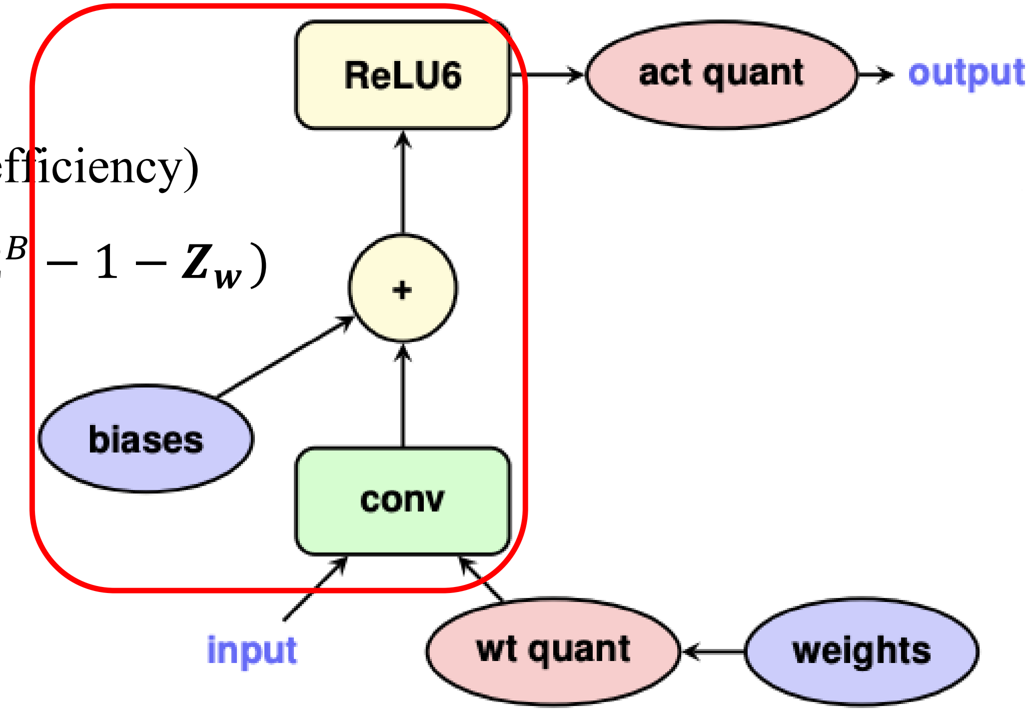
- Forward propagation
  - For t-th iteration, based on the current weight $\mathbf{W}^t$ and input $\mathbf{X}^t$, update quantization parameters as below:
    - EWMA filter: $\boldsymbol{r}^t_{\boldsymbol{w,min}} = \alpha \cdot \boldsymbol{r}^{t-1}_{\boldsymbol{w,min}} + (1-\alpha) \cdot \min(\mathbf{W}^t)$, $\boldsymbol{r}^t_{\boldsymbol{w,max}} = \alpha \cdot \boldsymbol{r}^{t-1}_{\boldsymbol{w,max}} + (1-\alpha) \cdot max(\mathbf{W}^t)$
    - Scaling factor: $\boldsymbol{S}^t_{\boldsymbol{w}} = \frac{\boldsymbol{r}^t_{\boldsymbol{w,max}} - \boldsymbol{r}^t_{\boldsymbol{w,min}}}{2^B - 1}$
    - Zero point: $\boldsymbol{Z}^t_{\boldsymbol{w}} = \left\lfloor \frac{-\boldsymbol{r}^t_{\boldsymbol{w,min}}}{S_w} \right\rceil$ (must be an integer for efficiency)
    - Tune the range: $\boldsymbol{r}^t_{\boldsymbol{w,min}} = -\boldsymbol{S}_{\boldsymbol{w}} \boldsymbol{Z}_{\boldsymbol{w}}$, $\boldsymbol{r}^t_{\boldsymbol{w,max}} = \boldsymbol{S}_{\boldsymbol{w}} (2^B - 1 - \boldsymbol{Z}_{\boldsymbol{w}})$



[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Quant-aware Training

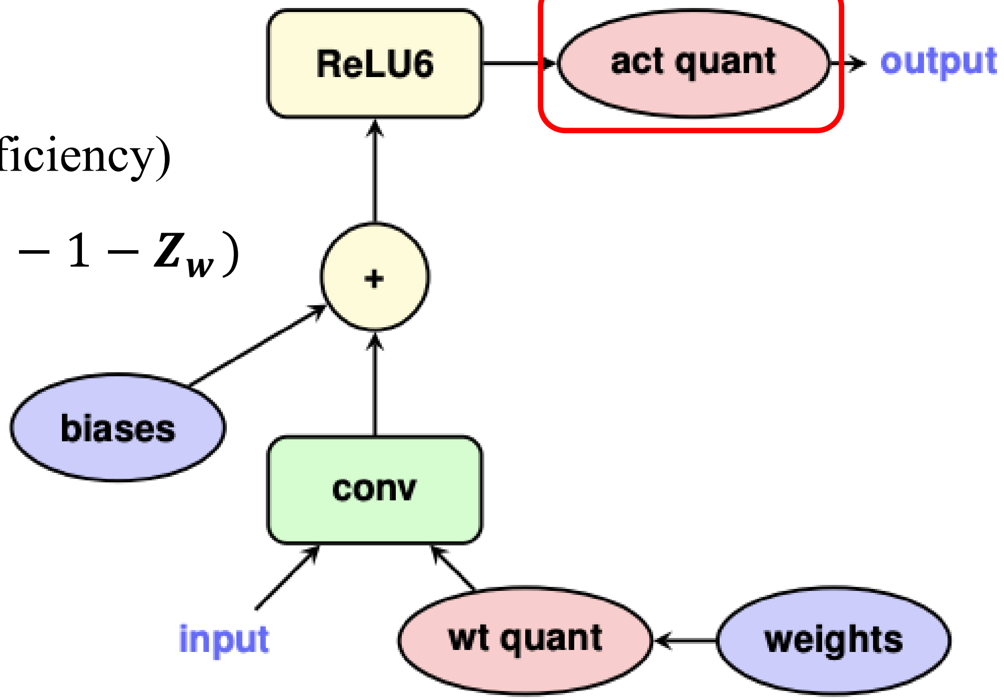
- Forward propagation
  - For t-th iteration, based on the current weight $\mathbf{W}^t$ and input $\mathbf{X}^t$, update quantization parameters as below:
    - EWMA filter: $\boldsymbol{r}_{w,min}^{t} = \alpha \cdot \boldsymbol{r}_{w,min}^{t-1} + (1-\alpha) \cdot \min(\mathbf{W}^t), \boldsymbol{r}_{w,max}^{t} = \alpha \cdot \boldsymbol{r}_{w,max}^{t-1} + (1-\alpha) \cdot max(\mathbf{W}^t)$
    - Scaling factor: $\boldsymbol{S}_w^t = \frac{\boldsymbol{r}_{w,max}^t - \boldsymbol{r}_{w,min}^t}{2^B - 1}$
    - Zero point: $\boldsymbol{Z}_w^t = \left\lfloor \frac{-\boldsymbol{r}_{w,min}^t}{S_w} \right\rceil$ (must be an integer for efficiency)
    - Tune the range: $\boldsymbol{r}_{w,min}^t = -\boldsymbol{S}_w \boldsymbol{Z}_w, \boldsymbol{r}_{w,max}^t = \boldsymbol{S}_w (2^B - 1 - \boldsymbol{Z}_w)$
  - Conv with quantized weights



[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Quant-aware Training

- Forward propagation
  - For t-th iteration, based on the current weight $\mathbf{W}^t$ and input $\mathbf{X}^t$, update quantization parameters as below:
    - EWMA filter: $\boldsymbol{r}_{w,min}^{t} = \alpha \cdot \boldsymbol{r}_{w,min}^{t-1} + (1-\alpha) \cdot \min(\mathbf{W}^t)$, $\boldsymbol{r}_{w,max}^{t} = \alpha \cdot \boldsymbol{r}_{w,max}^{t-1} + (1-\alpha) \cdot max(\mathbf{W}^t)$
    - Scaling factor: $\boldsymbol{S}_w^t = \frac{\boldsymbol{r}_{w,max}^t - \boldsymbol{r}_{w,min}^t}{2^B - 1}$
    - Zero point: $\boldsymbol{Z}_w^t = \left\lfloor \frac{-\boldsymbol{r}_{w,min}^t}{S_w} \right\rceil$ (must be an integer for efficiency)
    - Tune the range: $\boldsymbol{r}_{w,min}^t = -\boldsymbol{S}_w \boldsymbol{Z}_w$, $\boldsymbol{r}_{w,max}^t = \boldsymbol{S}_w (2^B - 1 - \boldsymbol{Z}_w)$
  - Conv with quantized weights
  - Update quantization parameters for activations
  - Quantize activations



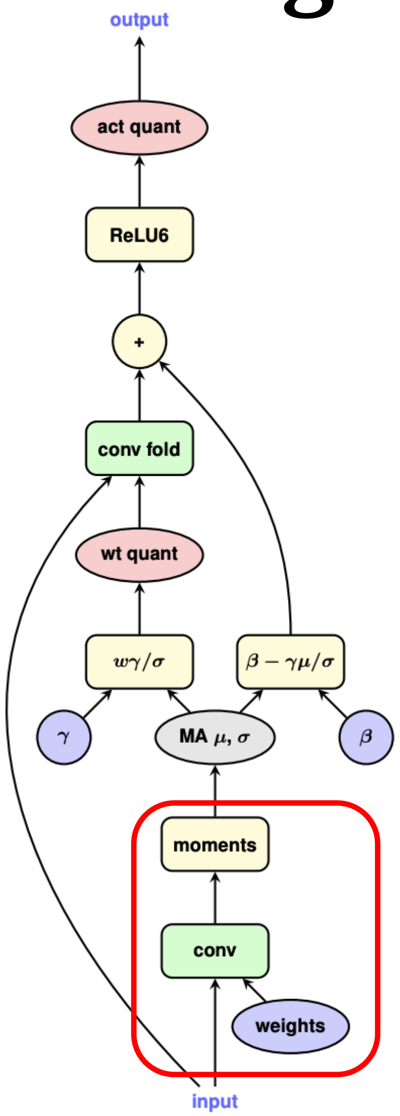
[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Quant-aware Training

- Backward propagation
  - As if none of them are quantized… (no gradients for quantized weights/activations!)
  - All of weights, activations, and gradients are real values!
  - Although forward propagation uses quantized weights and activations, their real value versions are **stored** to calculate gradients of the real value versions!

Integer-only [CVPR'18] – Quant-aware Training

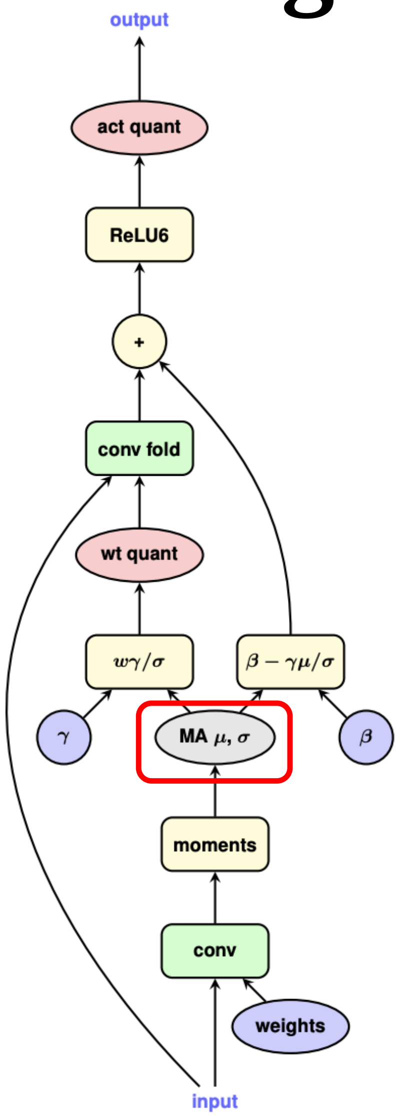
- Include batch normalization for forward propagation
  - Gain real value (temporary) output s^t $(= X \cdot W)$ by using a real weight conv filter
    - This is not a real output, **only** for BNorm parameter update



[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Quant-aware Training

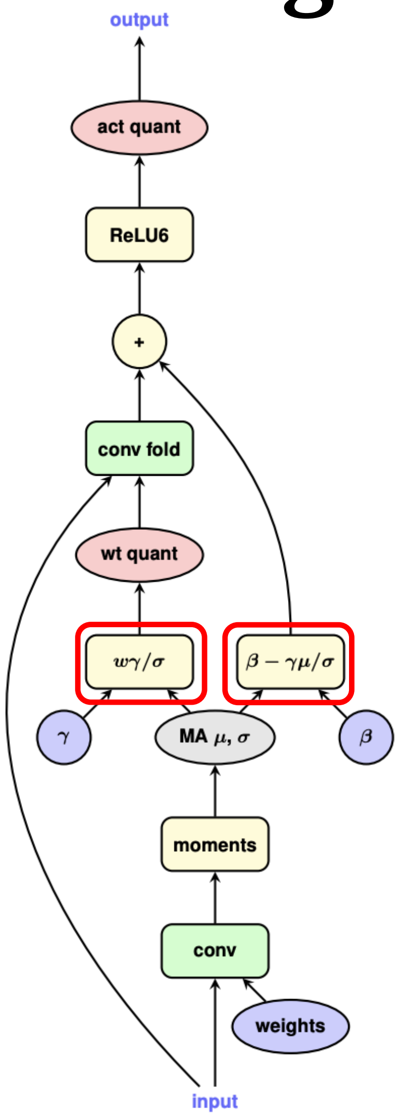
- Include batch normalization for forward propagation
  - Gain real value (temporary) output $\boldsymbol{s}^t$ $(= \boldsymbol{X} \cdot \boldsymbol{W})$ by using a real weight conv filter
    - This is not a real output, **only** for BNorm parameter update
  - Based on $\boldsymbol{s}^t$, update BNorm parameters
    - $\boldsymbol{\sigma}^2_{\boldsymbol{a},\boldsymbol{t}} = \alpha \cdot \boldsymbol{\sigma}^2_{\boldsymbol{s},\boldsymbol{t-1}} + (1-\alpha) \cdot VAR(\boldsymbol{s}^t)$
    - $\mu_{\boldsymbol{a},\boldsymbol{t}} = \alpha \cdot \mu_{\boldsymbol{s},\boldsymbol{t-1}} + (1-\alpha) \cdot E\,(\boldsymbol{s}^t)$



[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Quant-aware Training

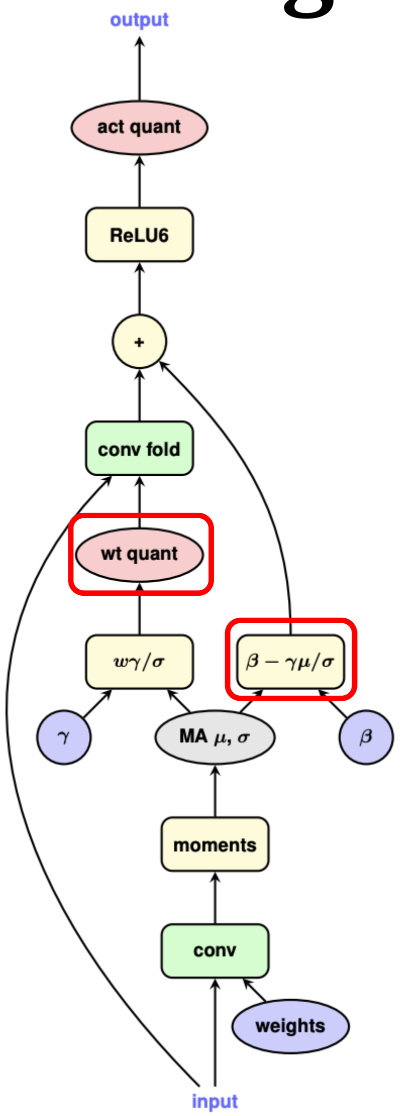
- Include batch normalization for forward propagation
  - Gain real value (temporary) output $\boldsymbol{s^t}$ $(= \boldsymbol{X} \cdot \boldsymbol{W})$ by using a real weight conv filter
    - This is not a real output, **only** for BNorm parameter update
  - Based on $\boldsymbol{s^t}$, update BNorm parameters
    - $\boldsymbol{\sigma^2_{a,t}} = \alpha \cdot \boldsymbol{\sigma^2_{s,t-1}} + (1-\alpha) \cdot VAR(\boldsymbol{s^t})$
    - $\mu_{\boldsymbol{a,t}} = \alpha \cdot \mu_{\boldsymbol{s,t-1}} + (1-\alpha) \cdot E(\boldsymbol{s^t})$
  - Folding BNorm parameters into conv weights and bias



[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Quant-aware Training

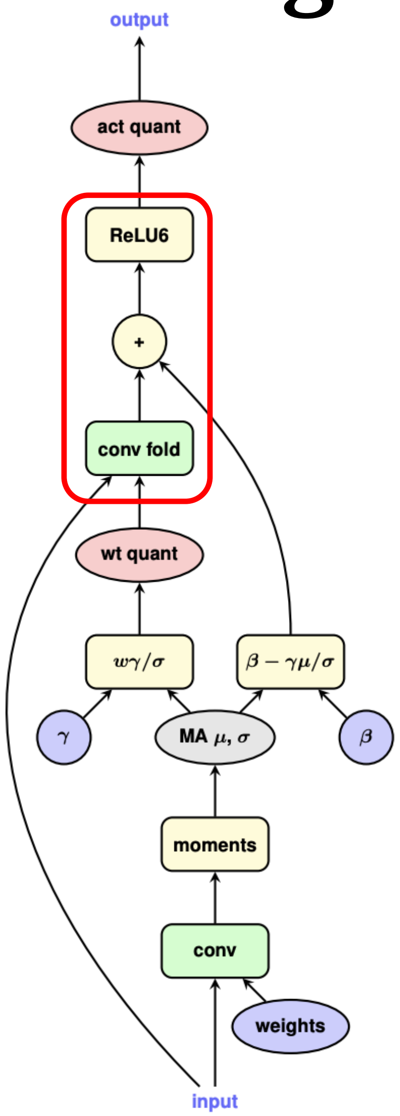
- Include batch normalization for forward propagation
  - Gain real value (temporary) output $\boldsymbol{s}^{\boldsymbol{t}}$ $(= \boldsymbol{X} \cdot \boldsymbol{W})$ by using a real weight conv filter
    - This is not a real output, **only** for BNorm parameter update
  - Based on $\boldsymbol{s}^{\boldsymbol{t}}$, update BNorm parameters
    - $\boldsymbol{\sigma}^{\mathbf{2}}_{\boldsymbol{a},\boldsymbol{t}} = \alpha \cdot \boldsymbol{\sigma}^{\mathbf{2}}_{\boldsymbol{s},\boldsymbol{t}-\mathbf{1}} + (1-\alpha) \cdot VAR(\boldsymbol{s}^{\boldsymbol{t}})$
    - $\mu_{\boldsymbol{a},\boldsymbol{t}} = \alpha \cdot \mu_{\boldsymbol{s},\boldsymbol{t}-\mathbf{1}} + (1-\alpha) \cdot E\,(\boldsymbol{s}^{\boldsymbol{t}})$
  - Folding BNorm parameters into conv weights and bias
  - Quantize the weights of the **folded** conv filter



[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Quant-aware Training

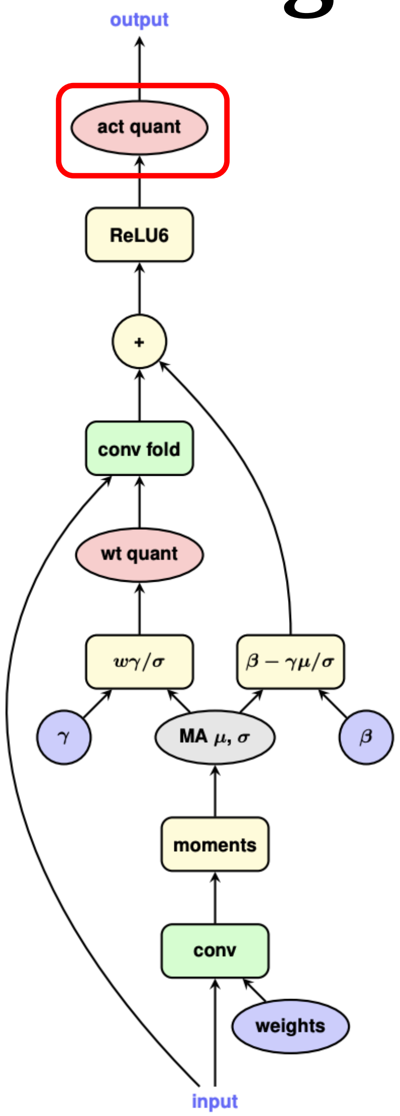
- Include batch normalization for forward propagation
  - Gain real value (temporary) output $\boldsymbol{s^t}$ $(= \boldsymbol{X} \cdot \boldsymbol{W})$ by using a real weight conv filter
    - This is not a real output, **<u>only</u>** for BNorm parameter update
  - Based on $\boldsymbol{s^t}$, update BNorm parameters
    - $\boldsymbol{\sigma^2_{a,t}} = \alpha \cdot \boldsymbol{\sigma^2_{s,t-1}} + (1-\alpha) \cdot VAR(\boldsymbol{s^t})$
    - $\mu_{\boldsymbol{a,t}} = \alpha \cdot \mu_{\boldsymbol{s,t-1}} + (1-\alpha) \cdot E\,(\boldsymbol{s^t})$
  - Folding BNorm parameters into conv weights and bias
  - Quantize the weights of the **folded** conv filter
  - Gain a BNormed output by using a **<u>quantized</u> conv fold** filter and a folded bias



[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Quant-aware Training

- Include batch normalization for forward propagation
  - Gain real value (temporary) output $\boldsymbol{s^t}$ $(= \boldsymbol{X} \cdot \boldsymbol{W})$ by using a real weight conv filter
    - This is not a real output, **only** for BNorm parameter update
  - Based on $\boldsymbol{s^t}$, update BNorm parameters
    - $\boldsymbol{\sigma^2_{a,t}} = \alpha \cdot \boldsymbol{\sigma^2_{s,t-1}} + (1 - \alpha) \cdot VAR(\boldsymbol{s^t})$
    - $\mu_{\boldsymbol{a,t}} = \alpha \cdot \mu_{\boldsymbol{s,t-1}} + (1 - \alpha) \cdot E\,(\boldsymbol{s^t})$
  - Folding BNorm parameters into conv weights and bias
  - Quantize the weights of the **folded** conv filter
  - Gain a BNormed output by using a **quantized conv fold** filter and a folded bias
  - After ReLU6, **quantize** the final output for the next layer



[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Results

- ResNet on ImageNet

ResNet depth	50	100	150
Floating-point accuracy	76.4%	78.0%	78.8%
Integer-quantized accuracy	74.9%	76.6%	76.7%

Table 4.1: ResNet on ImageNet: Floating-point vs quantized network accuracy for various network depths.

Scheme	BWN	TWN	INQ	FGQ	Ours
Weight bits	1	2	5	2	8
Activation bits	float32	float32	float32	8	8
Accuracy	68.7%	72.5%	74.8%	70.8%	74.9%

Table 4.2: ResNet on ImageNet: Accuracy under various quantization schemes, including binary weight networks (BWN [21, 15]), ternary weight networks (TWN [21, 22]), incremental network quantization (INQ [33]) and fine-grained quantization (FGQ [26])

[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR’18] – Results

- Inception on ImageNet

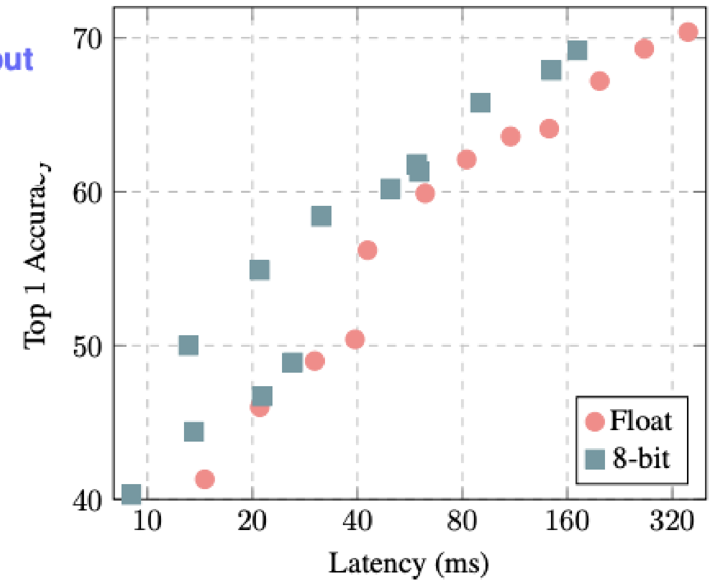
Act.	type	accuracy		recall 5	
		mean	std. dev.	mean	std.dev.
ReLU6	floats	78.4%	0.1%	94.1%	0.1%
	8 bits	75.4%	0.1%	92.5%	0.1%
	7 bits	75.0%	0.3%	92.4%	0.2%
ReLU	floats	78.3%	0.1%	94.2%	0.1%
	8 bits	74.2%	0.2%	92.2%	0.1%
	7 bits	73.7%	0.3%	92.0%	0.1%

Table 4.3: Inception v3 on ImageNet: Accuracy and recall 5 comparison of floating point and quantized models.

[The tables are from B. Jacob et al., “Quantization and training of neural networks for efficient integer-arithmetic-only inference.”]

Integer-only [CVPR'18] – Results

- **MobileNet** on ImageNet
  - For three CPU cores on Pixel 1 and 2
  - Different resolutions and depth multipliers



(c) ImageNet latency-vs-accuracy tradeoff

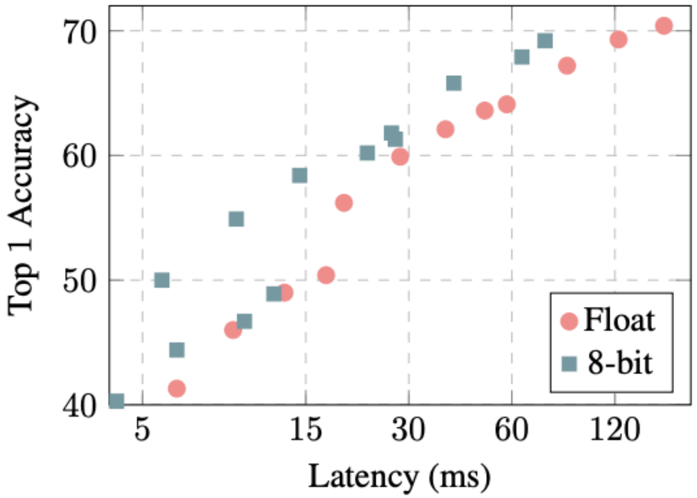


Figure 4.1: ImageNet classifier on Qualcomm Snapdragon 835 big cores: Latency-vs-accuracy tradeoff of floating-point and integer-only MobileNets.

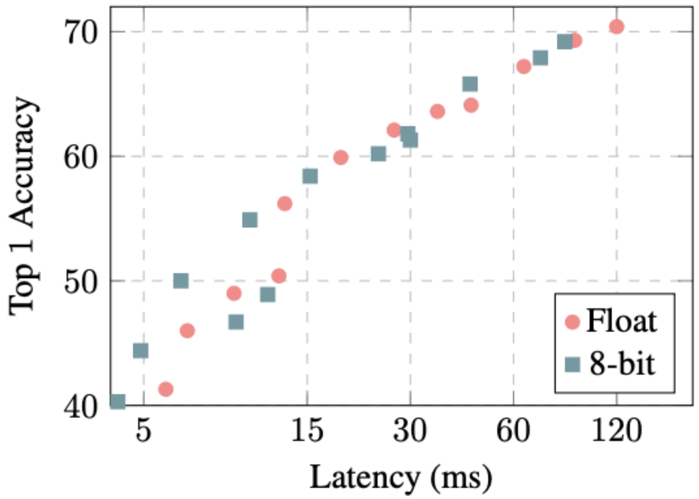


Figure 4.2: ImageNet classifier on Qualcomm Snapdragon 821: Latency-vs-accuracy tradeoff of floating-point and integer-only MobileNets.

[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

Integer-only [CVPR'18] – Results

- SSD + MobileNet

DM	Type	mAP	LITTLE (ms)	big (ms)
100%	floats	22.1	778	370
	8 bits	21.7	687	272
50%	floats	16.7	270	121
	8 bits	16.6	146	61

Table 4.4: Object detection speed and accuracy on COCO dataset of floating point and integer-only quantized models. Latency (ms) is measured on Qualcomm Snapdragon 835.

DM	Type	Precision	Recall	LITTLE (ms)	big (ms)
100%	floats	68%	76%	711	337
	8 bits	66%	75%	372	154
50%	floats	65%	70%	233	106
	8 bits	62%	70%	134	56
25%	floats	56%	64%	100	44
	8 bits	54%	63%	67	28

Table 4.5: Face detection accuracy of floating point and integer-only quantized models. The reported precision / recall is averaged over different precision / recall values where an IOU of x between the groundtruth and predicted windows is considered a correct detection, for x in $\{0.5, 0.55, \ldots, 0.95\}$. Latency (ms) of floating point and quantized models are reported on Qualcomm Snapdragon 835 using a single LITTLE and big core, respectively.

[The tables are from B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference."]

This is the reason why 8-bit integer-only edge TPU can perform DNN-based inference

Thanks!