On-Device AI 실습
Quantization for CNN

Youngmin Jeon

Overview

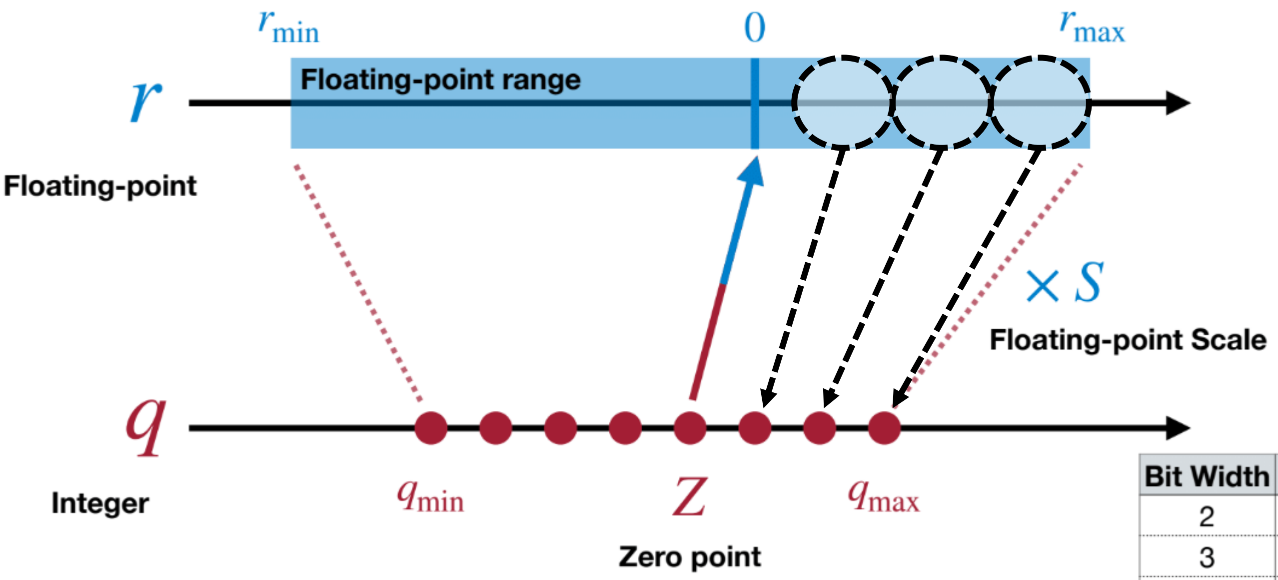
1. **Uniform Quantization**
   - Linear Quantization
   - Per-tensor / Per-channel Quantization
   - Quantized inference

2. **Non-uniform Quantization**
   - K-Means Quantization, Quantization-aware Training

3. **Quantization with PyTorch API**
   - Post-Training Quantization, Quantization Aware Training

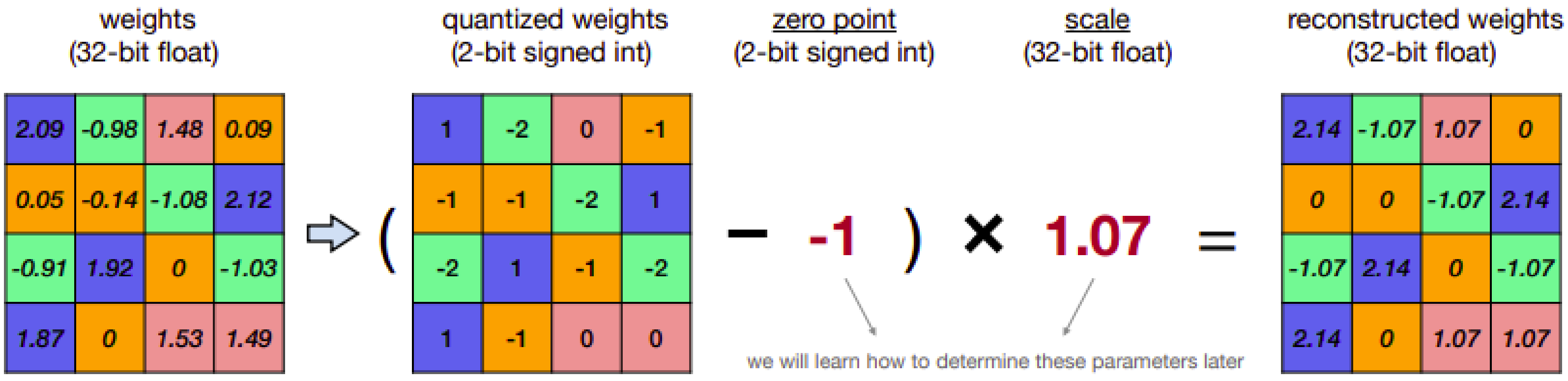
Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$



Bit Width	$q_{\min}$	$q_{\max}$
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

Linear Quantization



weights
(32-bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

⇨ (

quantized weights
(2-bit signed int)

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

− -1) × 1.07 =

zero point
(2-bit signed int)

scale
(32-bit float)

we will learn how to determine these parameters later

reconstructed weights
(32-bit float)

2.14	-1.07	1.07	0
0	0	-1.07	2.14
-1.07	2.14	0	-1.07
2.14	0	1.07	1.07

$$r = (q - Z) \times S$$

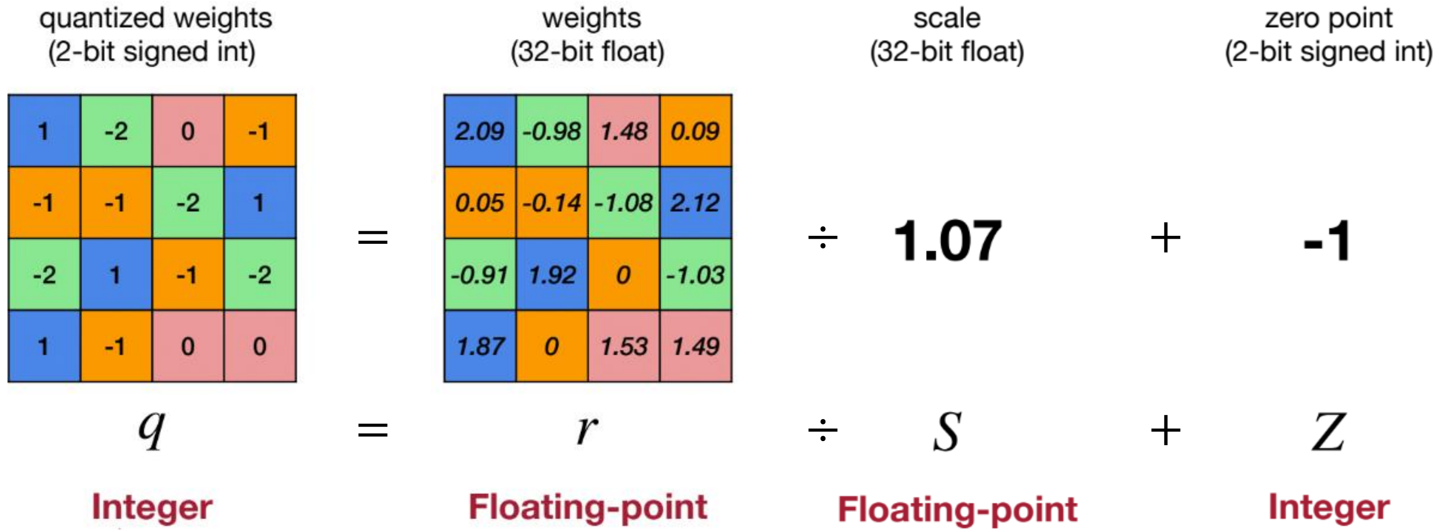
Binary	Decimal
01	1
00	0
11	-1
10	-2

quantization error

-0.05	0.09	0.41	0.09
0.05	-0.14	-0.01	-0.02
0.16	-0.22	0	0.04
-0.27	0	0.46	0.42

[실습1] Linear Quantization 함수 구현

Given the zero point and scale, how is a tensor quantized?



quantized weights (2-bit signed int) | weights (32-bit float) | scale (32-bit float) | zero point (2-bit signed int)

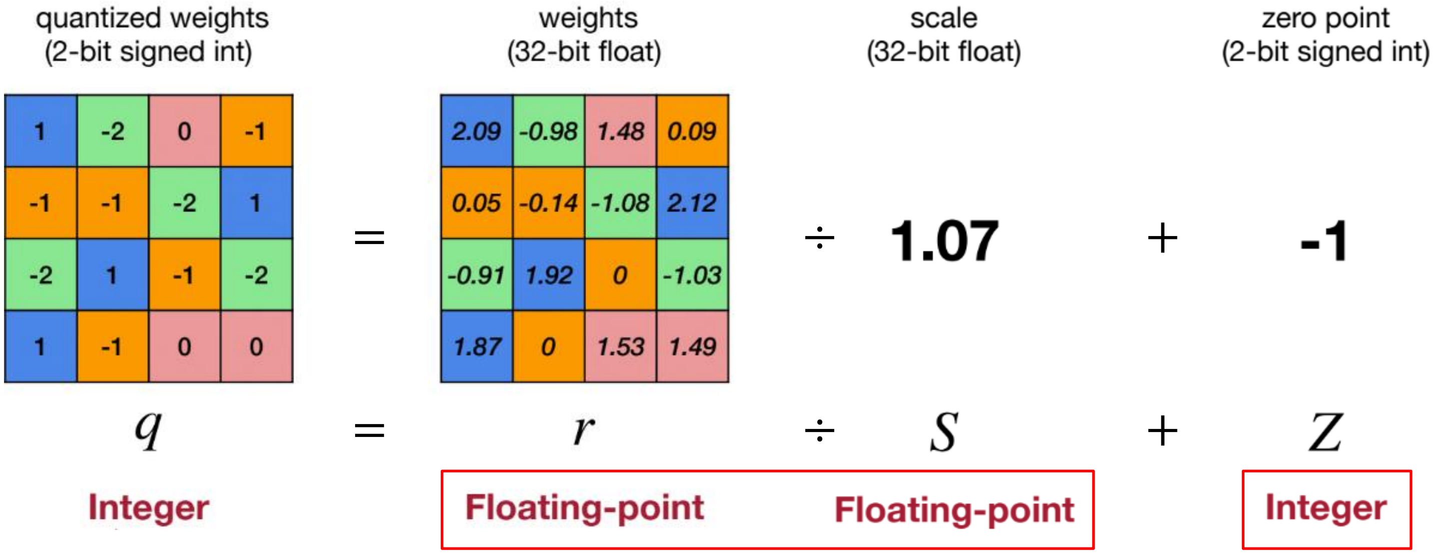
$$\begin{pmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{pmatrix} \div 1.07 + -1$$

$$q = r \div S + Z$$

Integer | **Floating-point** | **Floating-point** | **Integer**

[실습1] Linear Quantization 함수 구현

$$q = r/s + z$$



quantized weights (2-bit signed int)

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

=

weights (32-bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

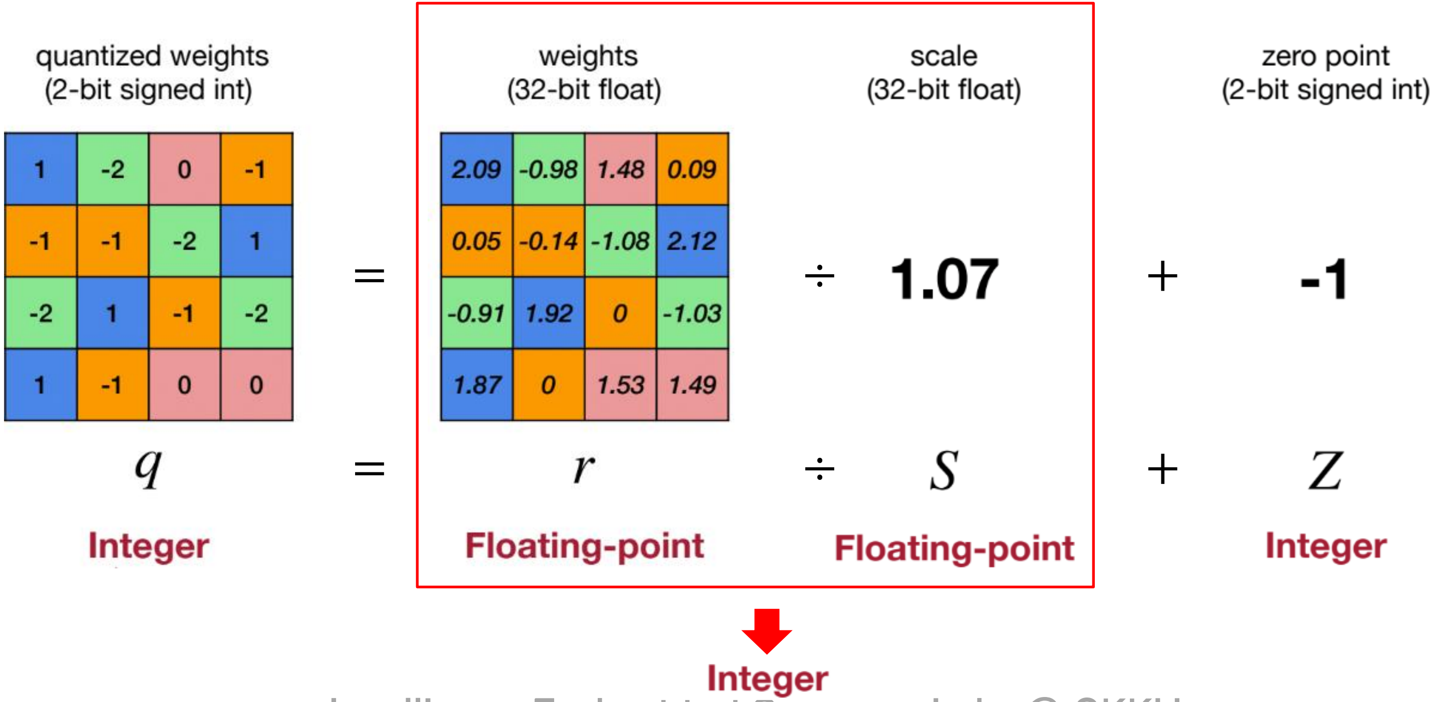
÷ scale (32-bit float) **1.07** + zero point (2-bit signed int) **-1**

$$q = r \div S + Z$$

Integer **Floating-point** **Floating-point** **Integer**

[실습1] Linear Quantization 함수 구현

$$q = int(round(r/s)) + z$$



quantized weights
(2-bit signed int)

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

=

weights
(32-bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

÷

scale
(32-bit float)

1.07

+

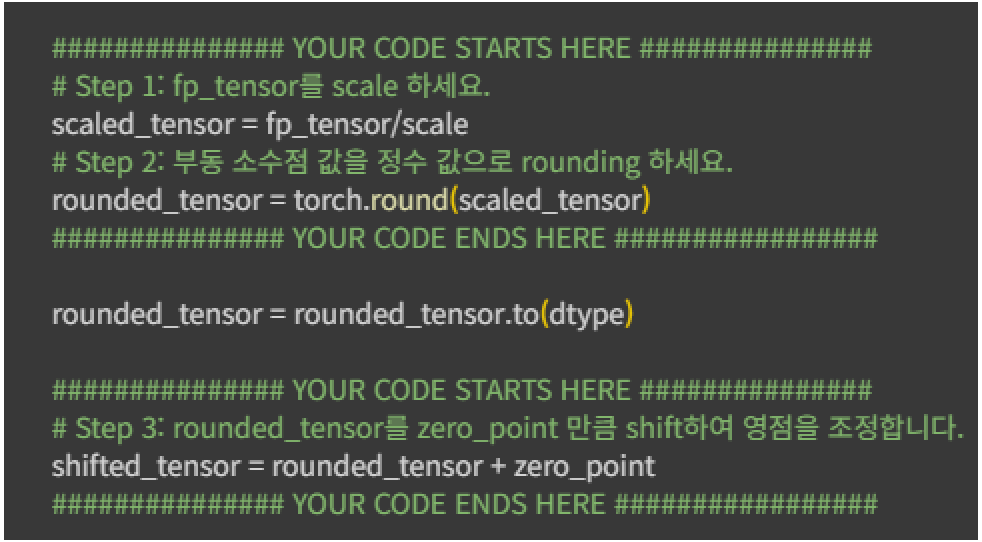
zero point
(2-bit signed int)

-1

$$q = r \div S + Z$$

Integer — **Floating-point** — **Floating-point** — **Integer**

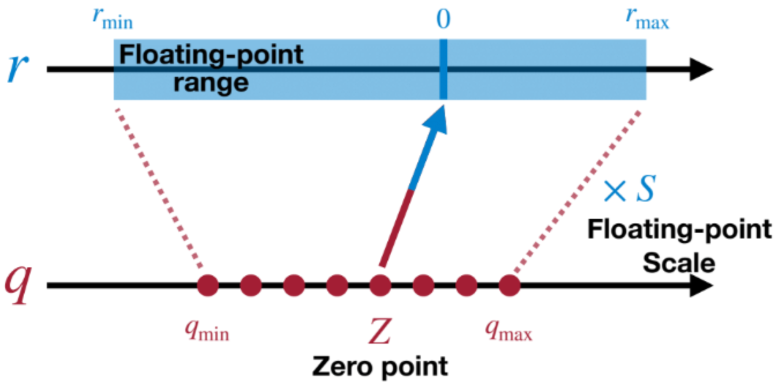
[실습1] Linear Quantization 함수 구현



```
############### YOUR CODE STARTS HERE ###############
# Step 1: fp_tensor를 scale 하세요.
scaled_tensor = fp_tensor/scale
# Step 2: 부동 소수점 값을 정수 값으로 rounding 하세요.
rounded_tensor = torch.round(scaled_tensor)
############### YOUR CODE ENDS HERE #################

rounded_tensor = rounded_tensor.to(dtype)

############### YOUR CODE STARTS HERE ###############
# Step 3: rounded_tensor를 zero_point 만큼 shift하여 영점을 조정합니다.
shifted_tensor = rounded_tensor + zero_point
############### YOUR CODE ENDS HERE #################
```

Linear Quantization – Scale

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



$$r_{\max} = S\left(q_{\max} - Z\right)$$

$$r_{\min} = S\left(q_{\min} - Z\right)$$

⊖

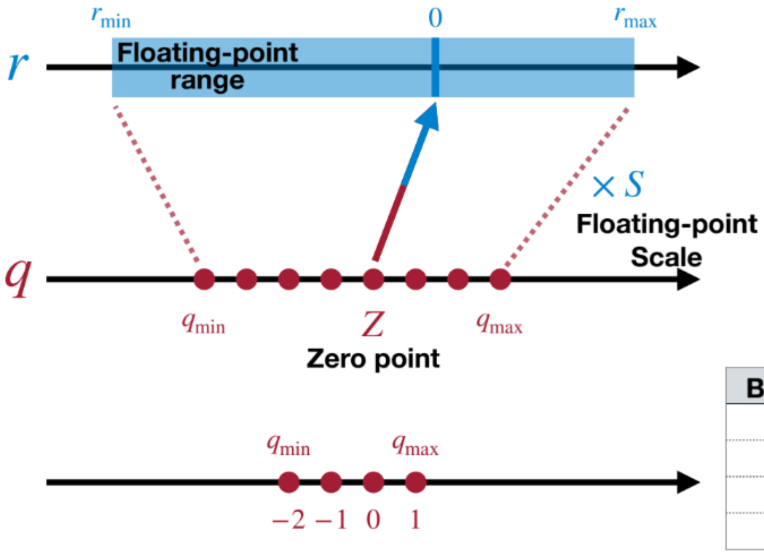
$$\downarrow$$

$$r_{\max} - r_{\min} = S\left(q_{\max} - q_{\min}\right)$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

Linear Quantization – Scale

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Min & Max value of fp tensor

Binary	Decimal
01	1
00	0
11	-1
10	-2

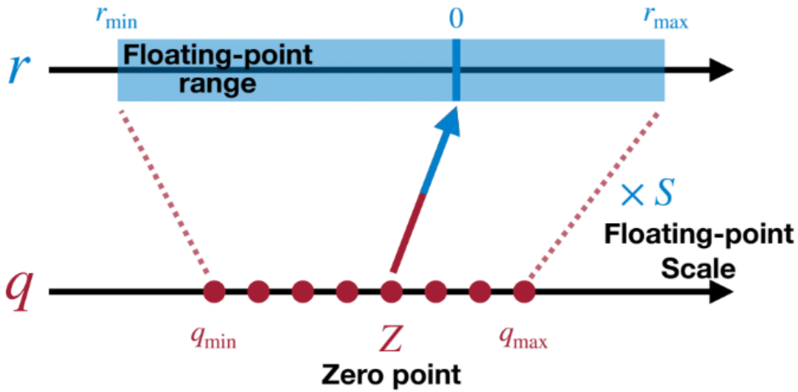
$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

$$= \frac{2.12 - (-1.08)}{1 - (-2)}$$

$$= 1.07$$

Linear Quantization – Zero Point

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



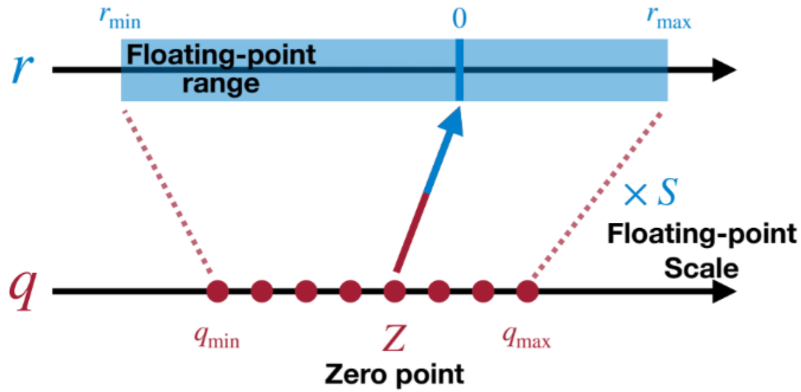
$$r_{\min} = S\left(q_{\min} - Z\right)$$

$$\downarrow$$

$$Z = q_{\min} - \frac{r_{\min}}{S}$$

Linear Quantization – Zero Point

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



$$r_{\min} = S\left(q_{\min} - Z\right)$$

$$\downarrow$$

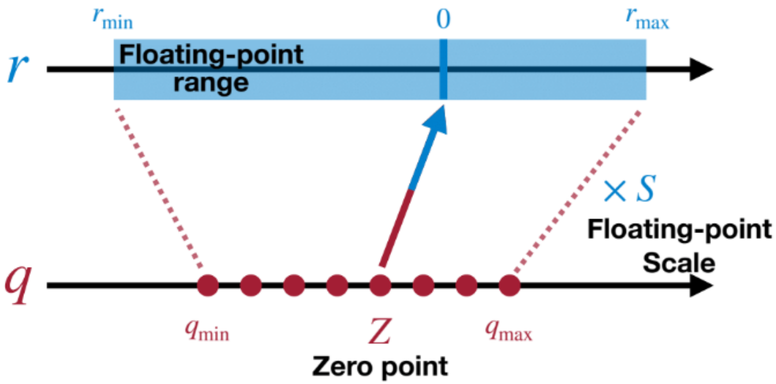
$$\underset{\text{Integer}}{Z} = \boxed{\underset{\text{Integer}}{q_{\min}} - \overset{\text{Floating-point}}{\underset{\text{Floating-point}}{\frac{r_{\min}}{S}}}}$$

$$\downarrow$$

$$Z = \mathrm{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

[실습2] Scale and Zero Point 계산

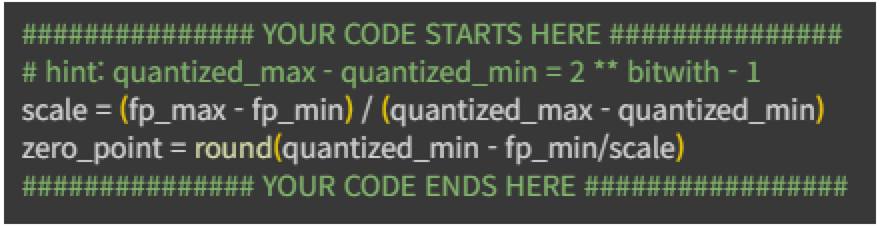
Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

$$Z = \text{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

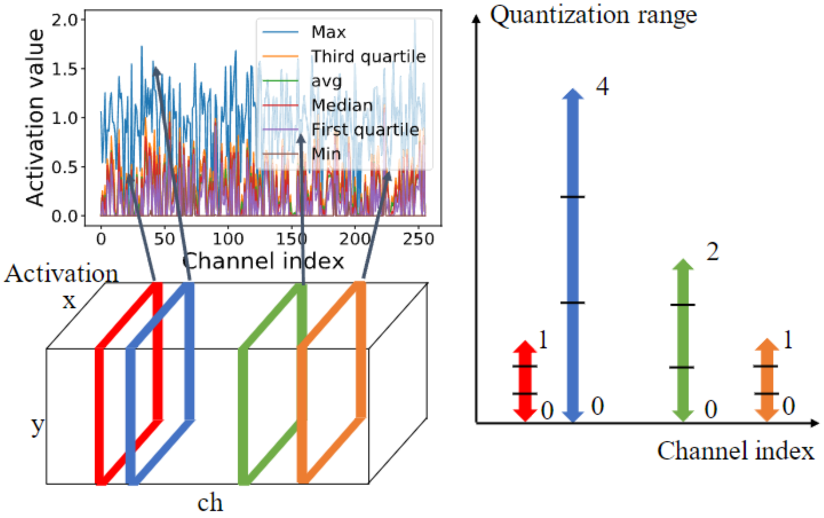
[실습2] Scale and Zero Point 계산

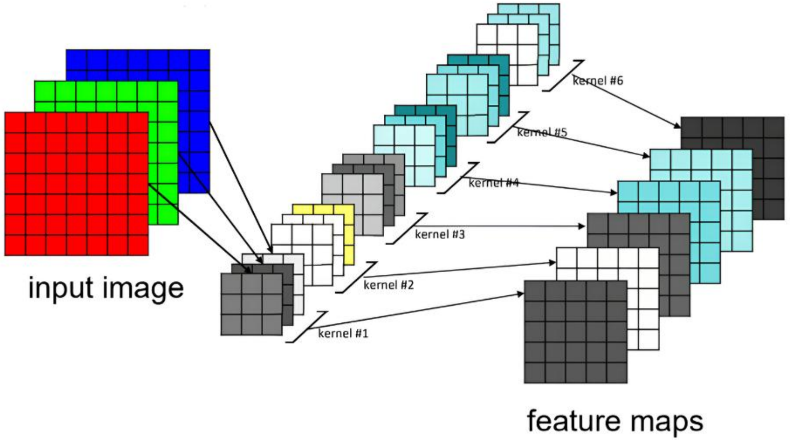


```
############### YOUR CODE STARTS HERE ###############
# hint: quantized_max - quantized_min = 2 ** bitwith - 1
scale = (fp_max - fp_min) / (quantized_max - quantized_min)
zero_point = round(quantized_min - fp_min/scale)
############### YOUR CODE ENDS HERE #################
```

Per-Channel Linear Quantization

- Different scaling factors **S** and zero points **Z** for **different output channels** will **perform better**



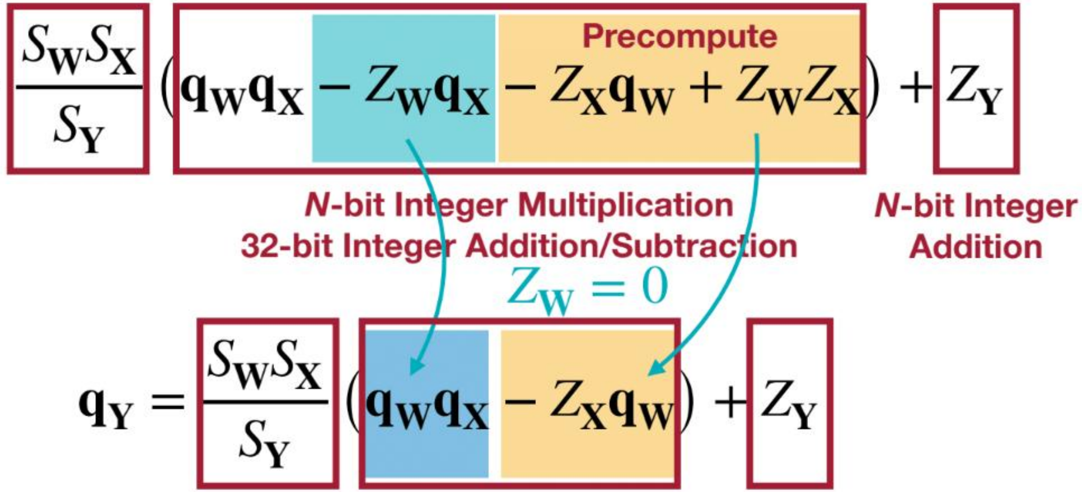


Quantized Matrix Multiplication

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following matrix multiplication, when Zw=0.

$$\mathbf{Y} = \mathbf{WX}$$



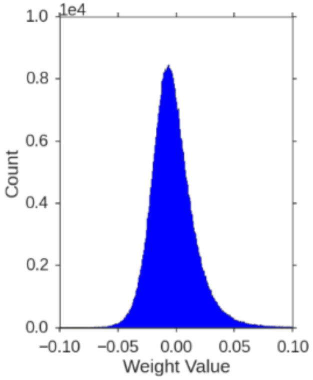
$$q_{\mathbf{Y}} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \boxed{\left(q_{\mathbf{W}} q_{\mathbf{X}} - Z_{\mathbf{W}} q_{\mathbf{X}} \overbrace{- Z_{\mathbf{X}} q_{\mathbf{W}} + Z_{\mathbf{W}} Z_{\mathbf{X}}}^{\text{Precompute}}\right)} + \boxed{Z_{\mathbf{Y}}}$$

N-bit Integer Multiplication
32-bit Integer Addition/Subtraction

N-bit Integer
Addition

$$Z_{\mathbf{W}} = 0$$

$$q_{\mathbf{Y}} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \boxed{\left(q_{\mathbf{W}} q_{\mathbf{X}} - Z_{\mathbf{X}} q_{\mathbf{W}}\right)} + \boxed{Z_{\mathbf{Y}}}$$



Quantized Inference

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}}\left(\mathbf{q}_{\mathbf{W}} - Z_{\mathbf{W}}\right) \cdot S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\right) + S_{\mathbf{b}}\left(\mathbf{q}_{\mathbf{b}} - Z_{\mathbf{b}}\right)$$

$$\downarrow \; Z_{\mathbf{W}} = 0$$

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}}S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}}\right) + S_{\mathbf{b}}\left(\mathbf{q}_{\mathbf{b}} - Z_{\mathbf{b}}\right)$$

$$\downarrow \; Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}}S_{\mathbf{X}}$$

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}}S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}} + \mathbf{q}_{\mathbf{b}}\right)$$

Quantized Inference

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

$$Z_{\mathbf{W}} = 0 \ \downarrow \ Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$$

$$S_{\mathbf{Y}} \left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}} \right) = S_{\mathbf{W}} S_{\mathbf{X}} \left(\mathbf{q}_{\mathbf{W}} \mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}} \mathbf{q}_{\mathbf{W}} + \mathbf{q}_{\mathbf{b}} \right)$$

$$\downarrow$$

$$\mathbf{q}_{\mathbf{Y}} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}} \left(\mathbf{q}_{\mathbf{W}} \mathbf{q}_{\mathbf{X}} + \overset{\textbf{Precompute}}{\mathbf{q}_{\mathbf{b}} - Z_{\mathbf{X}} \mathbf{q}_{\mathbf{W}}} \right) + Z_{\mathbf{Y}}$$

$$\downarrow \ \mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - Z_{\mathbf{X}} \mathbf{q}_{\mathbf{W}}$$

$$\mathbf{q}_{\mathbf{Y}} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}} \left(\mathbf{q}_{\mathbf{W}} \mathbf{q}_{\mathbf{X}} + \mathbf{q}_{bias} \right) + Z_{\mathbf{Y}}$$

[실습3] Quantized Fully-Connected Layer

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

$$\Big\downarrow$$

$Z_{\mathbf{W}} = 0$

$Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$

$\mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - Z_{\mathbf{X}} \mathbf{q}_{\mathbf{W}}$

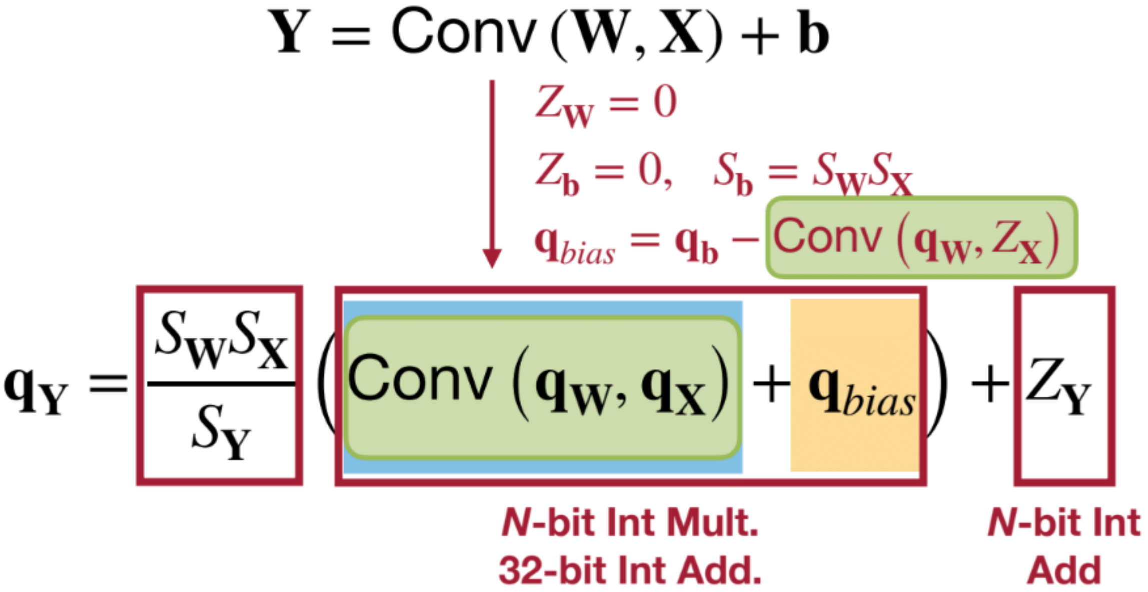
$$\mathbf{q}_{\mathbf{Y}} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \left(\boxed{\mathbf{q}_{\mathbf{W}} \mathbf{q}_{\mathbf{X}} + \mathbf{q}_{bias}} \right) + \boxed{Z_{\mathbf{Y}}}$$

_N_-bit Int Mult.
32-bit Int Add.

_N_-bit Int
Add

Note: both $\mathbf{q}_{\mathbf{b}}$ and $\mathbf{q}_{bias}$ are 32 bits.

[실습3] Quantized Convolution Layer



$$\mathbf{Y} = \text{Conv}\left(\mathbf{W}, \mathbf{X}\right) + \mathbf{b}$$

$$Z_{\mathbf{W}} = 0$$

$$Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$$

$$\mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - \boxed{\text{Conv}\left(\mathbf{q}_{\mathbf{W}}, Z_{\mathbf{X}}\right)}$$

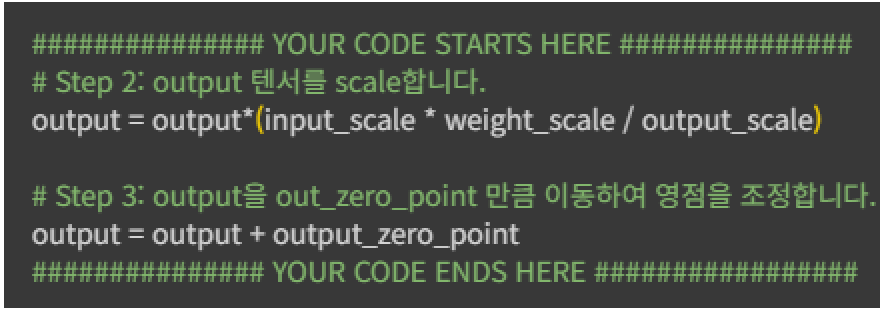
$$\mathbf{q}_{\mathbf{Y}} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \left(\boxed{\boxed{\text{Conv}\left(\mathbf{q}_{\mathbf{W}}, \mathbf{q}_{\mathbf{X}}\right)} + \mathbf{q}_{bias}} \right) + \boxed{Z_{\mathbf{Y}}}$$

***N*-bit Int Mult.**
32-bit Int Add.

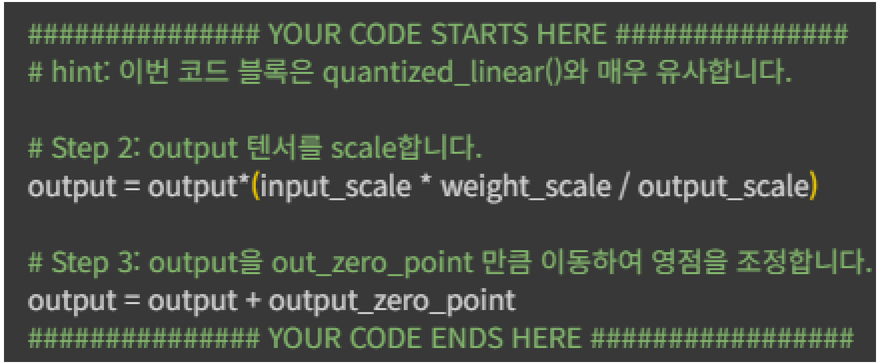
***N*-bit Int**
Add

Note: both $\mathbf{q}_{\mathbf{b}}$ and $\mathbf{q}_{bias}$ are 32 bits.

[실습3] Quantized Layer



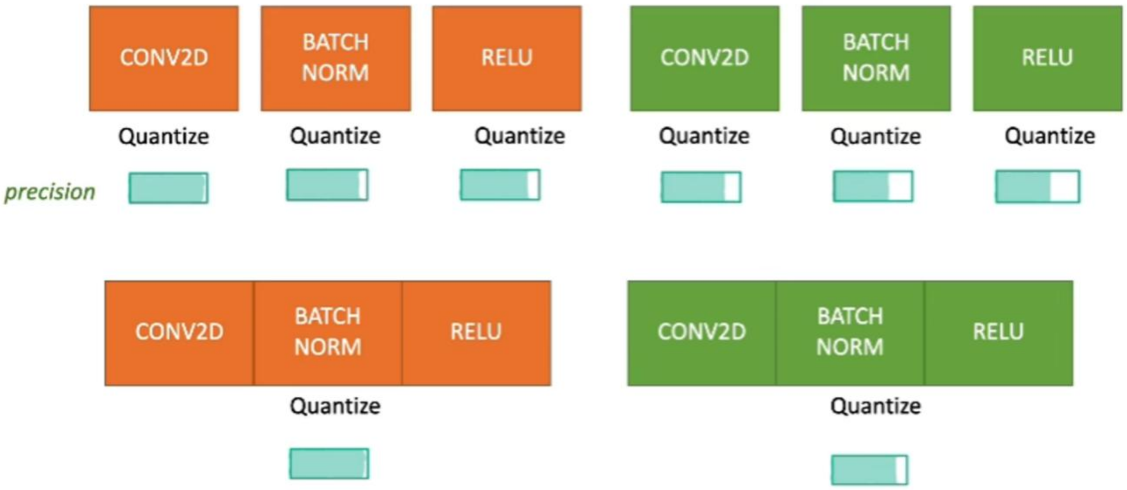
```
################ YOUR CODE STARTS HERE ################
# Step 2: output 텐서를 scale합니다.
output = output*(input_scale * weight_scale / output_scale)

# Step 3: output을 out_zero_point 만큼 이동하여 영점을 조정합니다.
output = output + output_zero_point
################ YOUR CODE ENDS HERE ##################
```

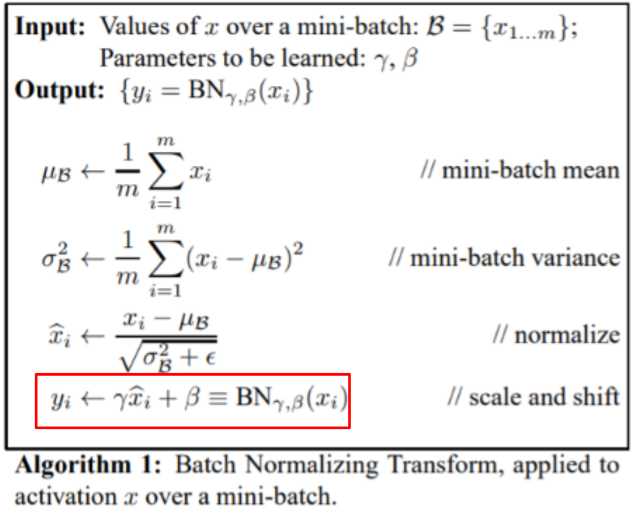


```
################ YOUR CODE STARTS HERE ################
# hint: 이번 코드 블록은 quantized_linear()와 매우 유사합니다.

# Step 2: output 텐서를 scale합니다.
output = output*(input_scale * weight_scale / output_scale)

# Step 3: output을 out_zero_point 만큼 이동하여 영점을 조정합니다.
output = output + output_zero_point
################ YOUR CODE ENDS HERE ##################
```

Model Fusion

- We can **fuse** a **BatchNorm** layer into its previous **convolutional layer**
- Fusing model can reduce the extra multiplication during inference





Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

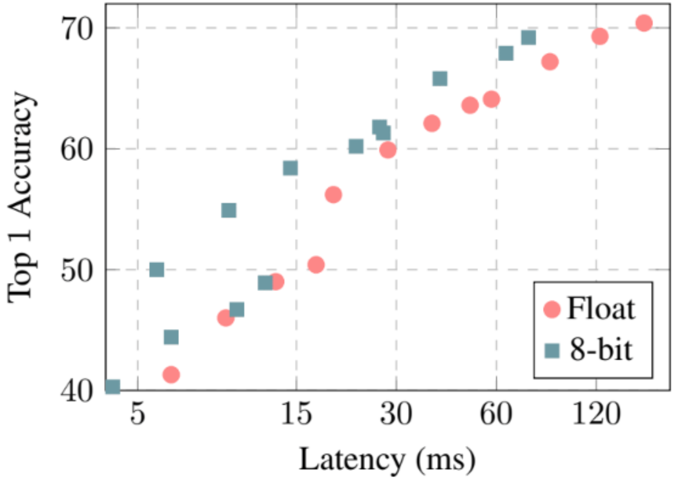
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$\boxed{y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)} \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Inference with quantized model

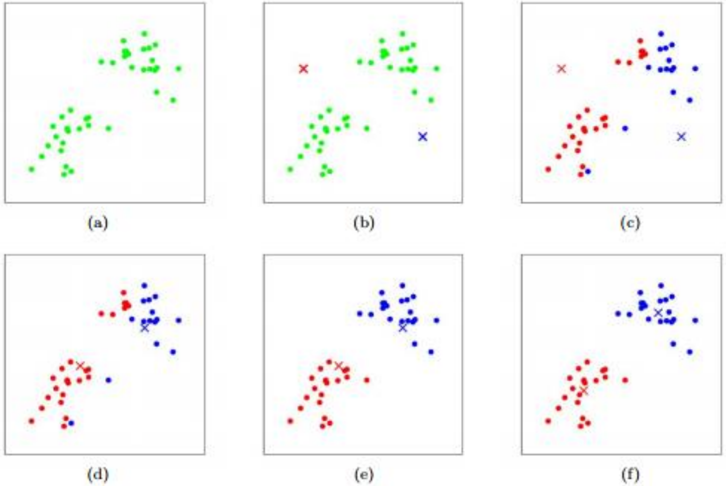
Neural Network	ResNet-50	Inception-V3
Floating-point Accuracy	76.4%	78.4%
8-bit Integer-quantized Acurracy	74.9%	75.4%



Latency-vs-accuracy tradeoff of float vs. integer-only MobileNets on ImageNet using Snapdragon 835 big cores.

K-Means Clustering

- Popular **unsupervised** learning algorithm to group data into K clusters
- Select K initial centroids and assign each data point into nearest centroids
- Update centroids by calculating the mean of the assigned points



K-Means-based Weight Quantization

weights
(32-bit float)

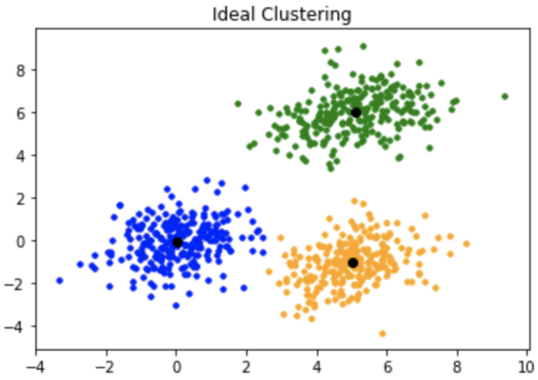
2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

~~2.09, 2.12, 1.92, 1.87~~

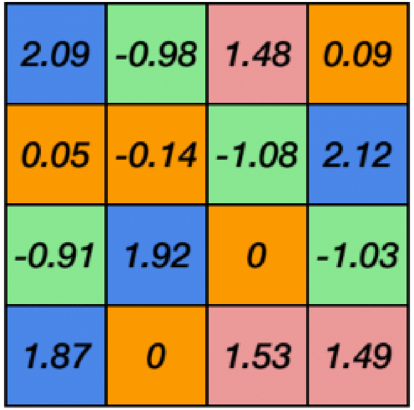
↓

2.0

Cluster

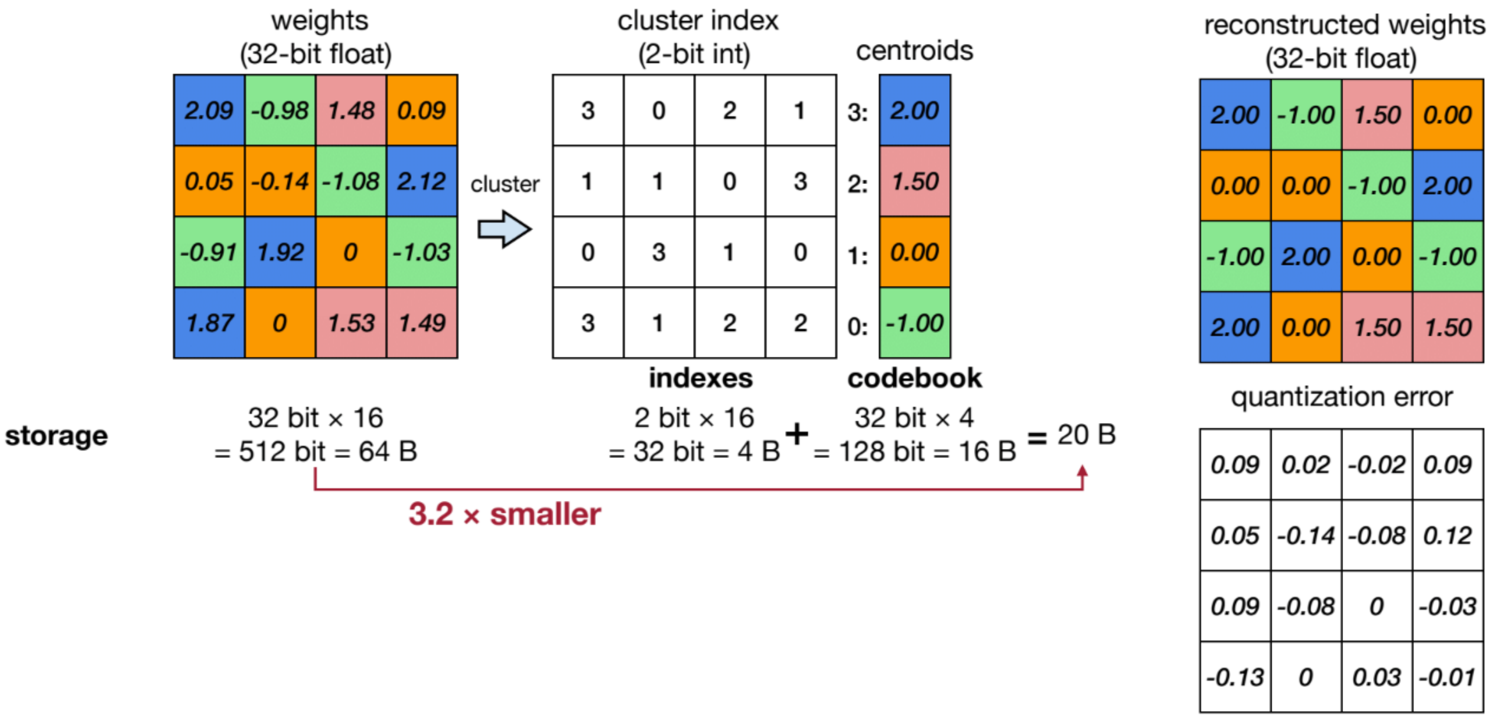

Ideal Clustering

weights
(32-bit float)



2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

K-Means-based Weight Quantization



weights
(32-bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

cluster ⇨

cluster index
(2-bit int)

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

indexes

centroids

3:	2.00
2:	1.50
1:	0.00
0:	-1.00

codebook

reconstructed weights
(32-bit float)

2.00	-1.00	1.50	0.00
0.00	0.00	-1.00	2.00
-1.00	2.00	0.00	-1.00
2.00	0.00	1.50	1.50

quantization error

0.09	0.02	-0.02	0.09
0.05	-0.14	-0.08	0.12
0.09	-0.08	0	-0.03
-0.13	0	0.03	-0.01

storage

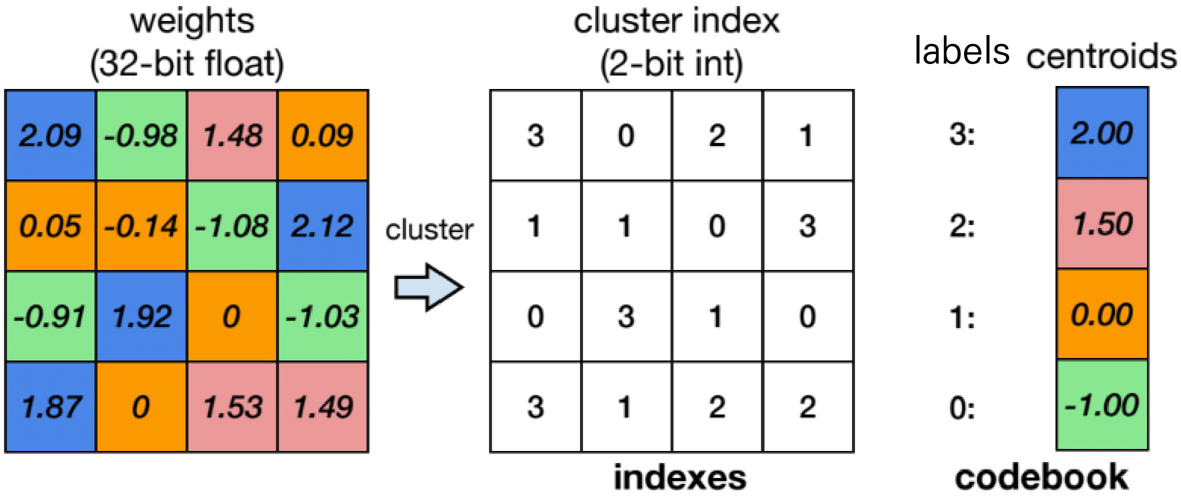
32 bit × 16
= 512 bit = 64 B

2 bit × 16
= 32 bit = 4 B **+** 32 bit × 4
= 128 bit = 16 B **=** 20 B

3.2 × smaller

[실습4] K-Means Quantization 함수 구현

- Get number of clusters (2^n) based on the quantization precision
- **Decode** the codebook into k-means quantized tensor for inference



weights (32-bit float):

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

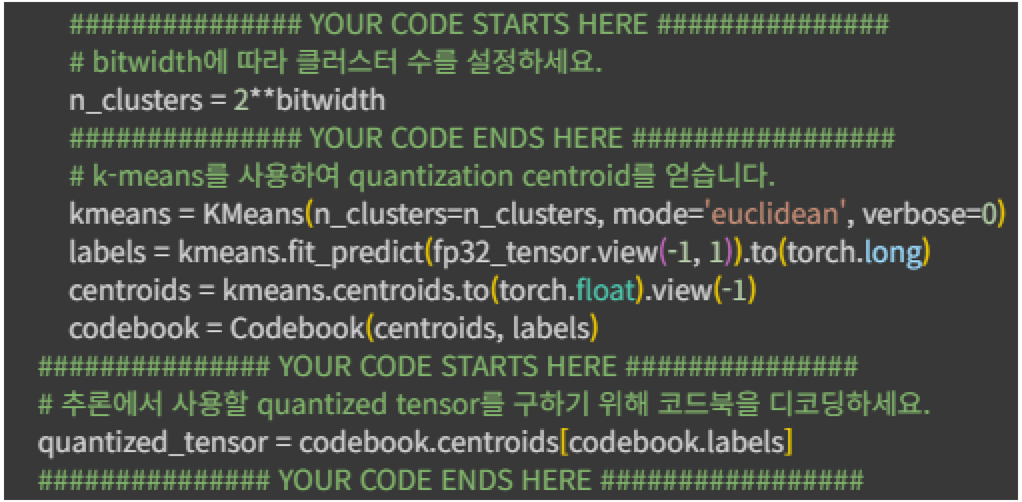
cluster index (2-bit int) — indexes:

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

codebook:

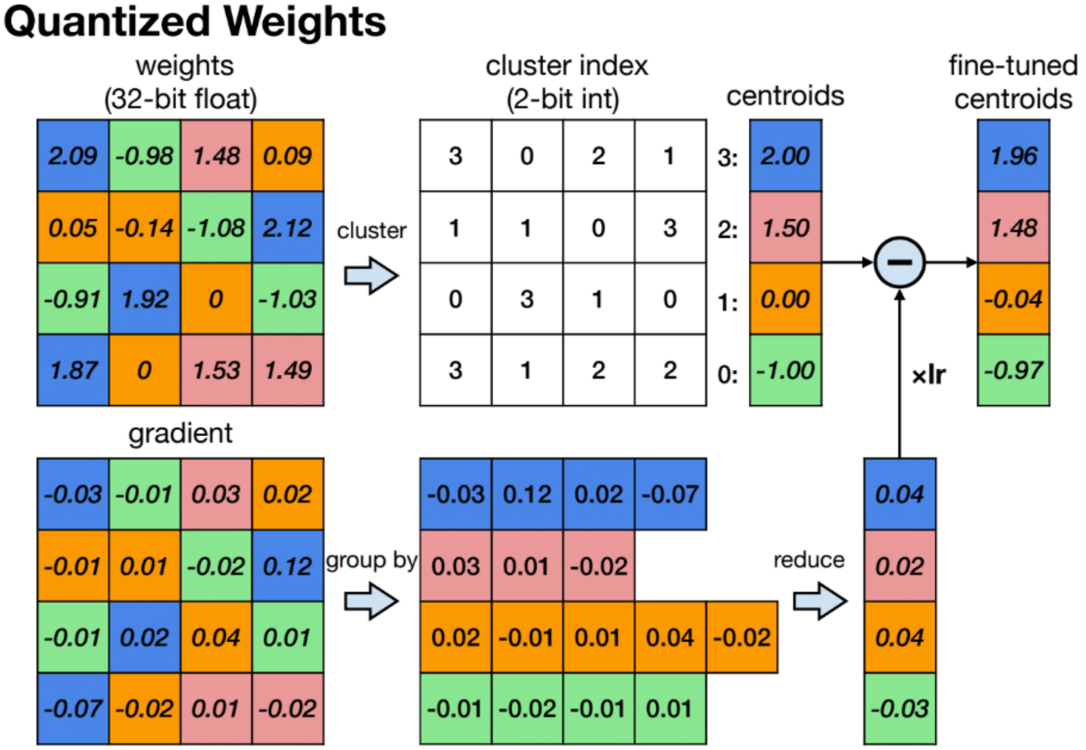
labels	centroids
3:	2.00
2:	1.50
1:	0.00
0:	-1.00

[실습4] K-Means Quantization 함수 구현



```
    ################ YOUR CODE STARTS HERE ################
    # bitwidth에 따라 클러스터 수를 설정하세요.
    n_clusters = 2**bitwidth
    ################ YOUR CODE ENDS HERE ##################
    # k-means를 사용하여 quantization centroid를 얻습니다.
    kmeans = KMeans(n_clusters=n_clusters, mode='euclidean', verbose=0)
    labels = kmeans.fit_predict(fp32_tensor.view(-1, 1)).to(torch.long)
    centroids = kmeans.centroids.to(torch.float).view(-1)
    codebook = Codebook(centroids, labels)
################ YOUR CODE STARTS HERE ################
# 추론에서 사용할 quantized tensor를 구하기 위해 코드북을 디코딩하세요.
quantized_tensor = codebook.centroids[codebook.labels]
################ YOUR CODE ENDS HERE ##################
```

QAT with K-means quantization



Fine-tuning Quantized Weights

weights (32-bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

cluster

cluster index (2-bit int)

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

centroids

3:	2.00
2:	1.50
1:	0.00
0:	-1.00

fine-tuned centroids

1.96
1.48
-0.04
-0.97

gradient

-0.03	-0.01	0.03	0.02
-0.01	0.01	-0.02	0.12
-0.01	0.02	0.04	0.01
-0.07	-0.02	0.01	-0.02

group by

-0.03	0.12	0.02	-0.07	
0.03	0.01	-0.02		
0.02	-0.01	0.01	0.04	-0.02
-0.01	-0.02	-0.01	0.01	

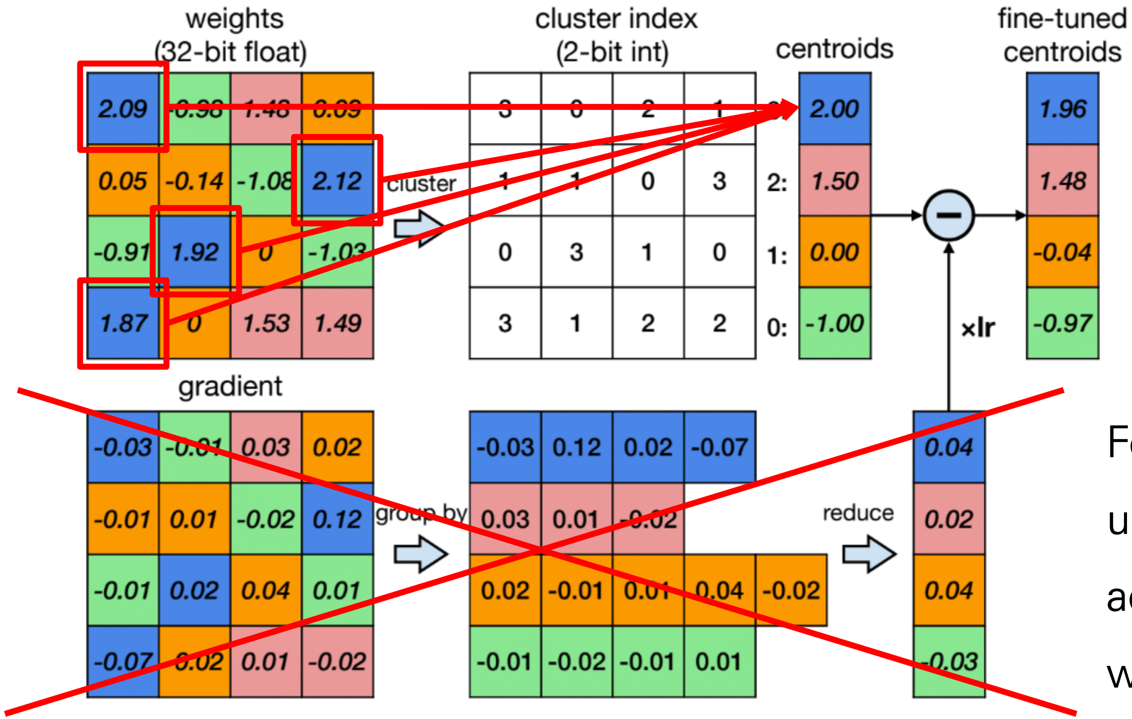
reduce

0.04
0.02
0.04
-0.03

×lr

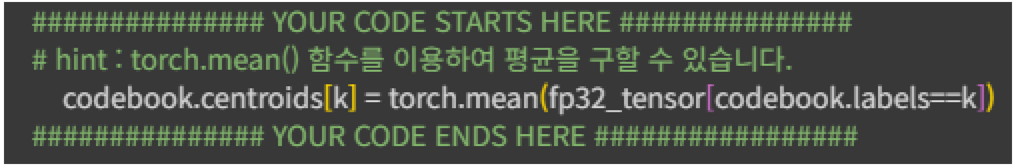
[실습5] Fine-tuning K-means quantization

Fine-tuning Quantized Weights



For simplicity, **directly** update the centroids according to the latest weights

[실습5] Fine-tuning K-means quantization

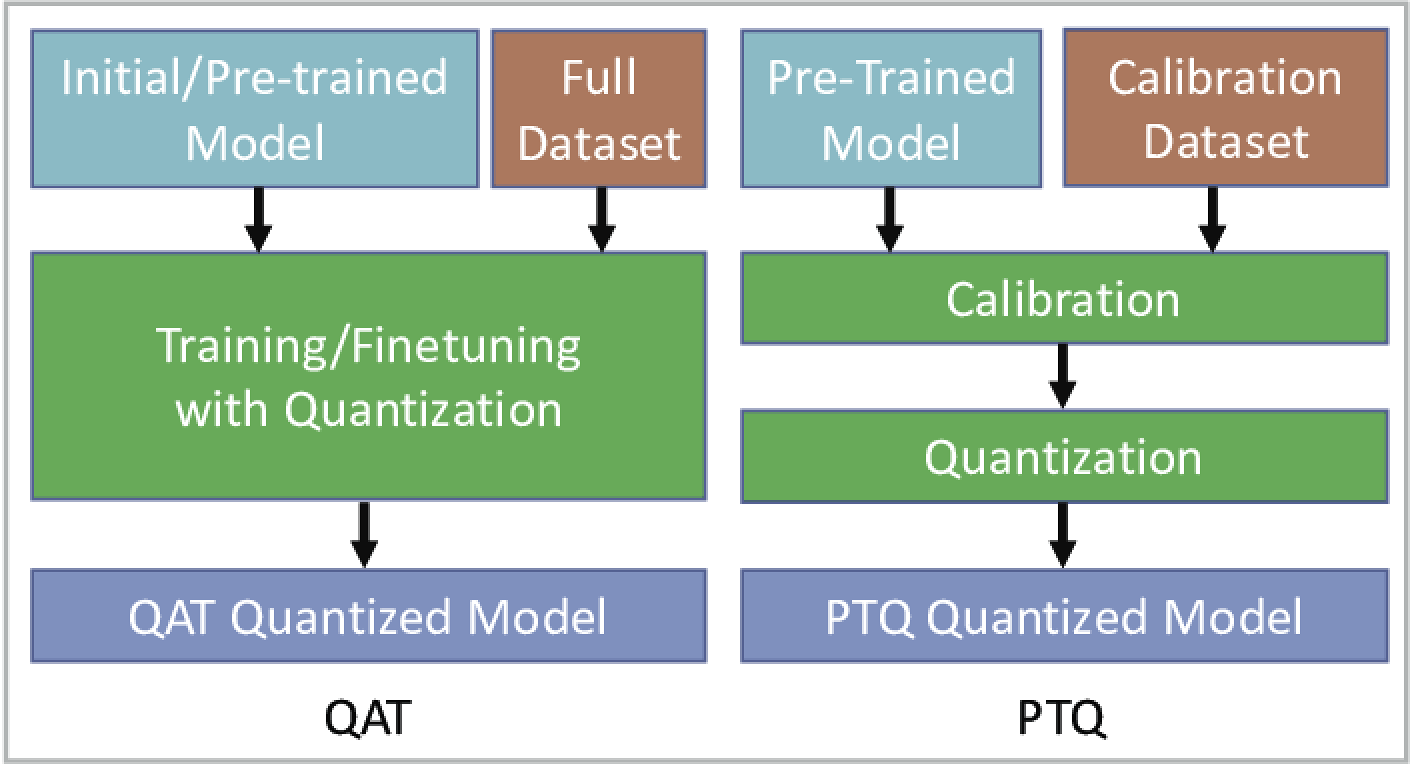


```
################ YOUR CODE STARTS HERE ################
# hint : torch.mean() 함수를 이용하여 평균을 구할 수 있습니다.
   codebook.centroids[k] = torch.mean(fp32_tensor[codebook.labels==k])
################ YOUR CODE ENDS HERE ##################
```

Quantization with PyTorch API

Machine Learning Libraries

Quantization with PyTorch API



Initial/Pre-trained Model

Full Dataset

Training/Finetuning with Quantization

QAT Quantized Model

QAT

Pre-Trained Model

Calibration Dataset

Calibration

Quantization

PTQ Quantized Model

PTQ