Neural Networks Speed-up and Compression

Lecture 4: Quantization

Lecturer: Dr. Julia Gusak, Senior Research Scientist, Skoltech
TAs: Stanislav Abukhovich, Dmitry Ermilov, Konstantin Sobolev; PhD students, Skoltech

When Quantization is needed?

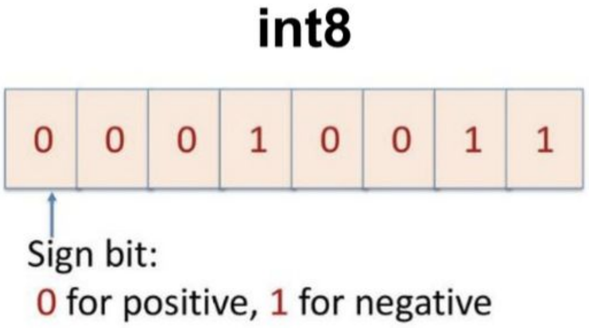
Outline

- Data types
- Quantization schemes
- Quantization of tensors
- Quantization of neural networks
  - without additional data
  - with additional data without fine-tuning
  - with additional data with fine-tuning

Data Types

Integer Format

- Most significant bit: defines the **sign** of the number
  - for positive X: “0”
  - for negative X: “1”
- Rest of bits represent
  - for positive X**:** |X|_2, a bit representation of **absolute value** of X
  - for negative X: (2^(N-1) - |X|)_2, where N - number of bits in the data format



On the image: bit representation of number 19

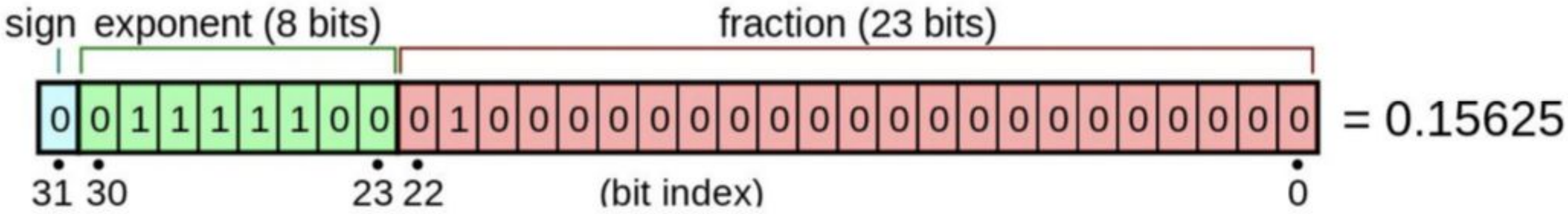
Integer Format

Data Types	Name in NumPy	Name in PyTorch	Bytes per number	Minimum representable number	Maximum representable number
int8	int8/byte	int8	1	-128	127
uint8	uint8/ubyte	uint8	1	0	255
int16	int16/short	int16/short	2	-32768	32767
uint16	uint16/ushort	-	2	0	65535
int32	int32/intc	int32/int	4	-2^{31}	$2^{31}-1$
uint32	uint32/uintc	-	4	0	$2^{32}-1$
int64	int64/int	int64/long	8	-2^{63}	$2^{63}-1$
uint64	uint64/uint	-	8	0	$2^{64}-1$

Names and characteristics of different integer types

Data Types: float

Bit representation:



Number corresponding to bit representation:

$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\cdots b_{23})_2-127} \times (1.(b_{22}b_{21}\cdots b_0)_2) = (-1)^{sign} \times 2^{E-127} \times \left(\sum_{i=1}^{23} b_{23-i}2^{-i}\right)$$

Data Type: float

Common name	Name in (IEEE 754)	Name in NumPy	Name in PyTorch	Bytes per number	Bound values	Bits in exponent	Bits in fraction
half-precision	binary16	float16, half	float16, half	2	from $\pm 6.10 \cdot 10^{-5}$ to $\pm 6.5 \cdot 10^{4}$	5	10
single-precision	binary32	float32, single	float32, float	4	from $\pm 1.18 \cdot 10^{-38}$ to $\pm 3.4 \cdot 10^{38}$	8	23
double-precision	binary64	float64, double	float64, double	8	from $\pm 2.23 \cdot 10^{-308}$ to $\pm 1.80 \cdot 10^{308}$	11	52

Names and characteristics of different floating-point types

Big-endian and Little-endian

- **Big-endian**: most significant bytes are stored at the smallest memory address
- **Little-endian**: least significant bytes are stored at the smallest memory address

```
Type:           int16
Digit:          19
Numpy:          00010011 00000000
PyTorch:        00010011 00000000
Little-endian:  00010011 00000000
Big-endian:     00000000 00010011
```

```
Type:           float32
Digit:          0.15625
Numpy:          0 00000000 00000000010000000111110
PyTorch:        0 00000000 00000000010000000111110
Little-endian:  0 00000000 00000000010000000111110
Big-endian:     0 01111100 01000000000000000000000
```

Remark: 1 byte = 8 bits

Integer Format: Benefits & Drawbacks

- Benefits
  - Arithmetic operations are performed much faster
  - The result of arithmetic operations in the valid range of values is absolutely accurate

- Drawbacks
  - Operations are performed only on integers
  - Limited range of values

Floating-point Format: Benefits & Drawbacks

- Benefits
  - Ability to work with real numbers
  - Wide range of possible values
- Drawbacks
  - The result of operations is calculated for more clock cycles
  - Loss of accuracy in calculations

Loss of accuracy due to rounding error

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

```
N = 1000000
items = np.array([1. / n**2 for n in range(1, N+1)], dtype=np.float32)
print(items.sum(), np.pi**2 / 6.)
print(np.abs(items.sum() - np.pi**2 / 6.))
```

```
1.6449317 1.6449340668482264
2.392844625331847e-06
```

```
val1 = np.float32(0.0)
for i in range(items.shape[0]):
    val1 += items[i]

print(val1, np.abs(val1 - np.pi**2 / 6.))

val2 = np.float32(0.0)
for i in range(items.shape[0]-1, -1, -1):
    val2 += items[i]

print(val2, np.abs(val2 - np.pi**2 / 6.))
```

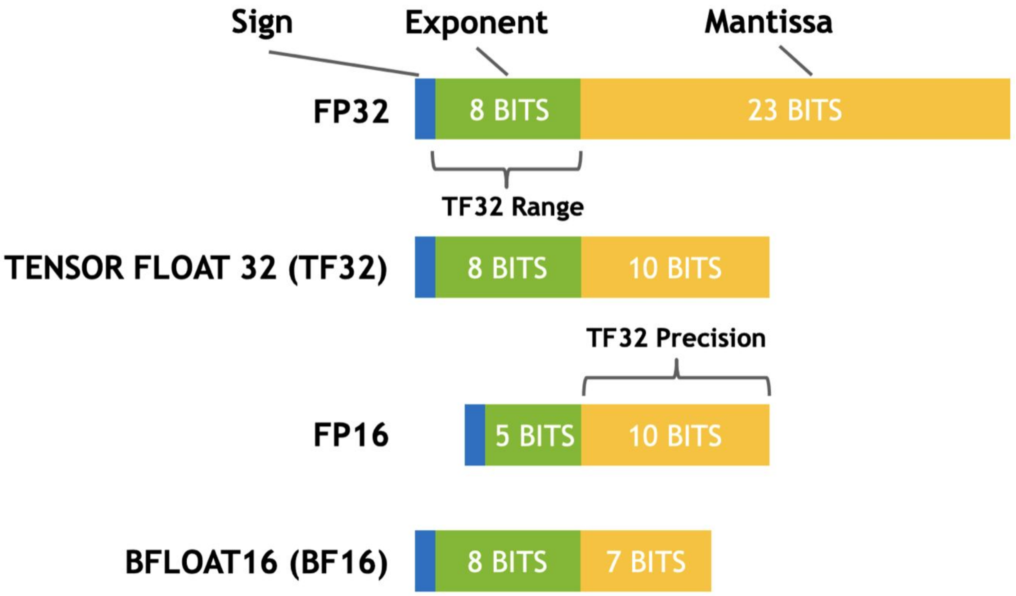
```
1.6447253 0.00020874412483773342
1.644933 1.0815424402732532e-06
```

Best strategy: start summation from smaller terms.

In that case the error does not accumulate so quickly, because terms have comparable orders

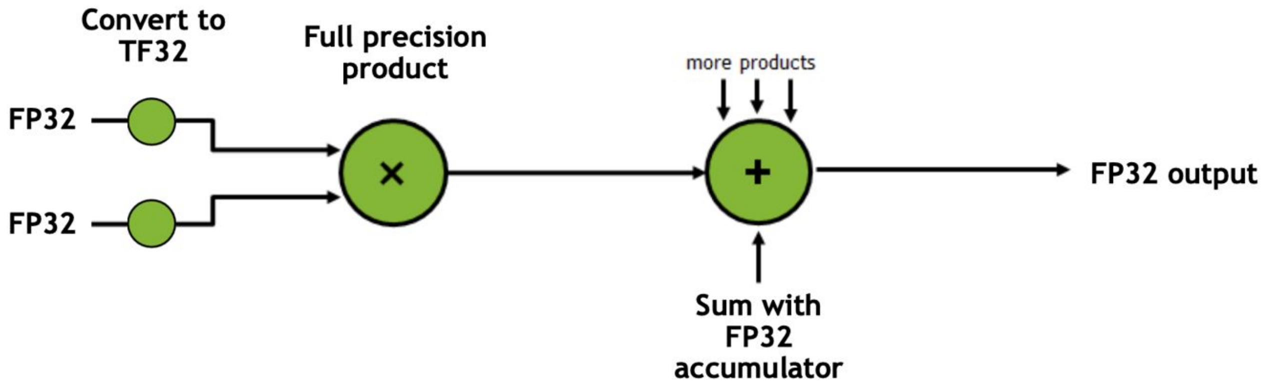
Tensor Float 32 (TF32) Format

NVIDIA A100 GPUs: supports TF32 format, which accelerates single-precision convolutions and matrix multiply layers



Source: https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

Tensor Core and TF32



- Benefits: fast hardware implementation of matrix and convolution operations, with comparable accuracy when using the float32 format.

- Drawbacks: limited application, can't be applied to
  - layers other than convolution/matrix-multiply operations (e.g., BN)
  - tensors in the format other than float32
  - optimizer or solver operations

Source: https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

NVIDIA V100 vs. NVIDIA A100

```
layer = torch.nn.Linear(10000,1001).to('cuda')
inp = torch.randn(990,10000).to('cuda')

with torch.autograd.profiler.profile(use_cuda=True) as prof:
    for i in range(10000):
        output_c = layer(inp)
```

	CUDA time, sec	CUDA average time, msec
V100 (no TF32 support)	14.83	1.483
A100 (TF32 support)	3.79	0.379

Matrix by vector multiplication,
comparison for NVIDIA V100 and NVIDIA A100

Quantization Schemes

Uniform Affine Quantization

- x - floating-point number in a range (x_{min}, x_{max})
- Quantize to one of the number from $(0, 2^b - 1)$
- s - scale factor, z - zero-point, b - bit-width

Quantization

$$\mathbf{x}_{\text{int}} = clamp\left(\left\lceil \frac{\mathbf{x}}{s} \right\rfloor + z, 0, 2^b - 1\right)$$

$$\text{clamp}(\mathbf{x}; a, c) = \begin{cases} a, & \mathbf{x} < a \\ \mathbf{x}, & a \leq \mathbf{x} \leq c \\ c, & \mathbf{x} > c \end{cases}$$

De-quantization

$$\mathbf{x} \approx \widehat{\mathbf{x}} = s\left(\mathbf{x}_{\text{int}} - z\right)$$

- quantization grid limits $q_{\text{min}} = -sz$ and $q_{\text{max}} = s(2^b - 1 - z)$

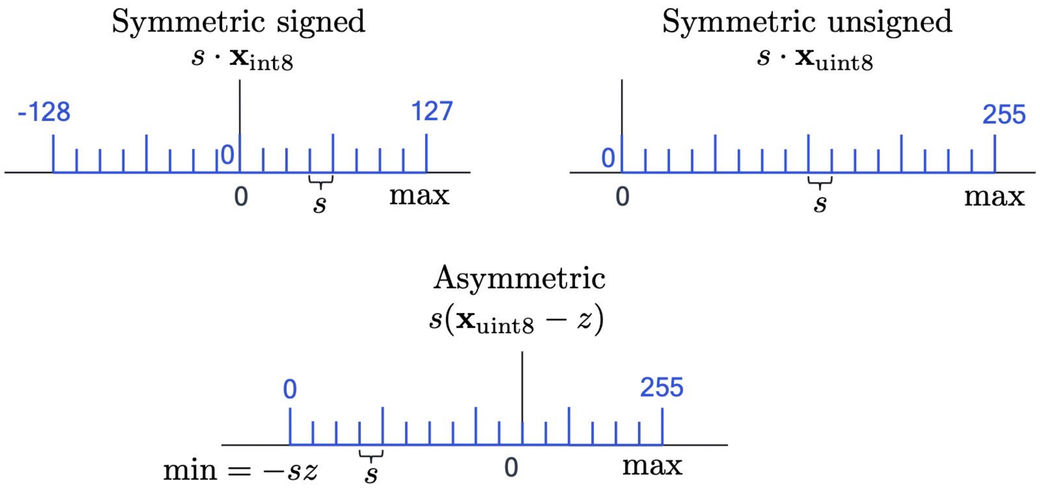
Sources:

- https://arxiv.org/pdf/1806.08342.pdf
- https://arxiv.org/pdf/2106.08295.pdf

Uniform Affine Quantization

$$\widehat{\mathbf{x}} = q(\mathbf{x}; s, z, b) = s\left[\text{clamp}\left(\left\lfloor\frac{\mathbf{x}}{s}\right\rceil + z; 0, 2^b - 1\right) - z\right]$$

- Symmetric: z = 0
- Asymmetric: z != 0



Source: https://arxiv.org/pdf/2106.08295.pdf

Other Quantization Schemes

- MinMax Quantization

$$\mathbf{x}_{\text{int}} = \left\lfloor \frac{\mathbf{x} - \min \mathbf{x}}{\max \mathbf{x} - \min \mathbf{x}} N + 0.5 \right\rfloor,$$
$$\widehat{\mathbf{x}} = \mathbf{x}_{\text{int}} \frac{\max \mathbf{x} - \min \mathbf{x}}{N} + \min \mathbf{x}$$

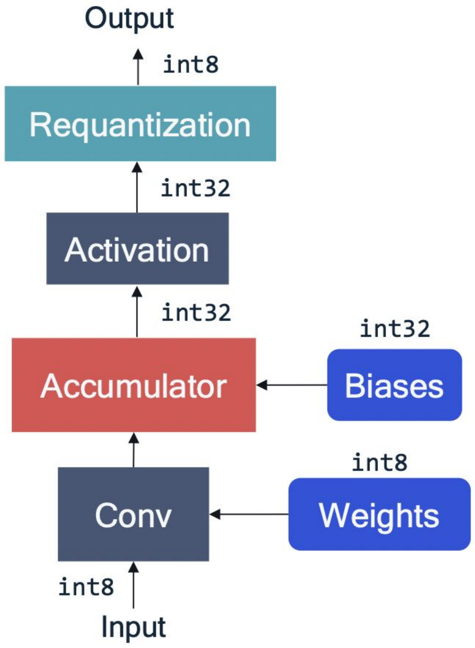
  - $N = 2^b, \; s = (\max \mathbf{x} - \min \mathbf{x})$
  - z = $-N \min \mathbf{x} \left(\max \mathbf{x} - \min \mathbf{x}\right)^{-1} + 0.5$

- Logarithmic

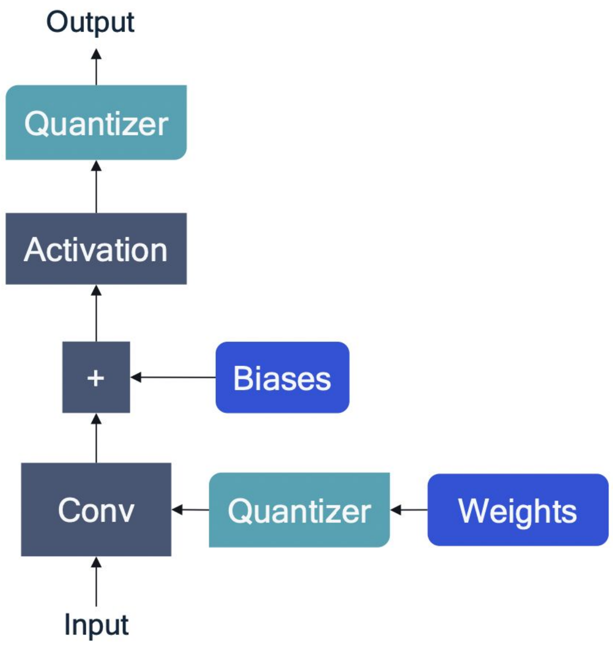
$$\widehat{\mathbf{x}} = \operatorname{sign}(\mathbf{x}) \exp\left(q(\log \|\mathbf{x}\|_{\infty})\right)$$

Quantization of Neural Networks

Quantization Simulation



(a) Diagram for quantized on-device inference with fixed-point operations.



(b) Simulated quantization using floating-point operations.

Source: https://arxiv.org/pdf/2106.08295.pdf

Batch Normalization Folding

$$\mathbf{y} = \mathrm{BatchNorm}(\mathbf{W}\mathbf{x})$$

$$\mathrm{BatchNorm}(\mathbf{x}) = \gamma \left(\frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta$$

$$\begin{aligned}
\mathbf{y}_k &= \mathrm{BatchNorm}(\mathbf{W}_{k,:}\,\mathbf{x}) \\
&= \boldsymbol{\gamma}_k \left(\frac{\mathbf{W}_{k,:}\,\mathbf{x} - \boldsymbol{\mu}_k}{\sqrt{\boldsymbol{\sigma}_k^2 + \epsilon}} \right) + \boldsymbol{\beta}_k \\
&= \frac{\boldsymbol{\gamma}_k \mathbf{W}_{k,:}}{\sqrt{\boldsymbol{\sigma}_k^2 + \epsilon}} \mathbf{x} + \left(\boldsymbol{\beta}_k - \frac{\boldsymbol{\gamma}_k \boldsymbol{\mu}_k}{\sqrt{\boldsymbol{\sigma}_k^2 + \epsilon}} \right) \\
&= \widetilde{\mathbf{W}}_{k,:}\,\mathbf{x} + \widetilde{\mathbf{b}}_k
\end{aligned}$$

where:

$$\begin{aligned}
\widetilde{\mathbf{W}}_{k,:} &= \frac{\boldsymbol{\gamma}_k \mathbf{W}_{k,:}}{\sqrt{\boldsymbol{\sigma}_k^2 + \epsilon}}, \\
\widetilde{\mathbf{b}}_k &= \boldsymbol{\beta}_k - \frac{\boldsymbol{\gamma}_k \boldsymbol{\mu}_k}{\sqrt{\boldsymbol{\sigma}_k^2 + \epsilon}}.
\end{aligned}$$

Activation Function Fusion

- **y = Activation(Wx)**
- It is wasteful
  - to write linear layer's activations to memory
  - and load them back into a compute core to apply a non-linearity.
- **Modern hardware solutions come with a hardware unit** that
  - **applies the non-linearity before the requantization step.**
  - In this case we only have to simulate requantization that happens after the non-linearity.
- Examples:
  - **ReLU** non-linearities are readily modelled by the requantization block (by setting the minimum representable value of that activation quantization to 0)
  - More complex activation functions (e.g., **Sigmoid or Swish**), require more dedicated support

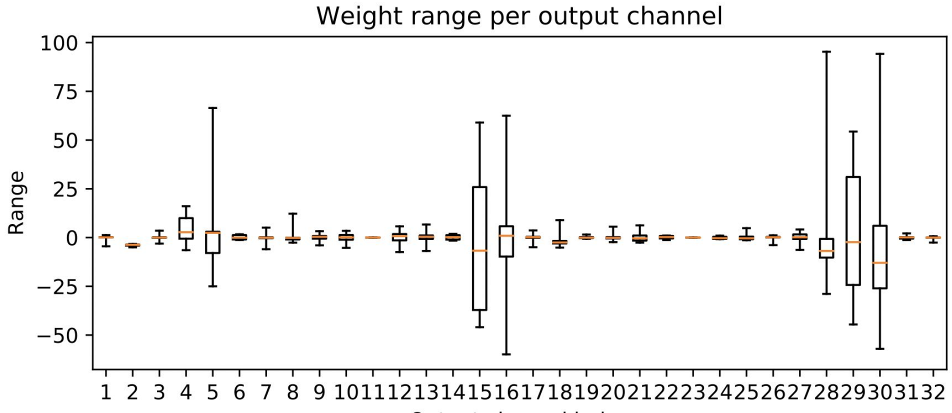
Source: https://arxiv.org/pdf/2106.08295.pdf

Per-tensor vs. Per-channel Quantization

- Per-tensor $$\widehat{\mathbf{W}}_{ij} = s\left[\text{clamp}\left(\left\lfloor \frac{\mathbf{W}_{ij}}{s^{\mathbf{W}}} \right\rceil + z^{\mathbf{W}}; 0, 2^b - 1\right) - z^{\mathbf{W}}\right]$$

- Per-channel $$\widehat{\mathbf{W}}_{ij} = s_i\left[\text{clamp}\left(\left\lfloor \frac{\mathbf{W}_{ij}}{s_i^{\mathbf{W}}} \right\rceil + z_i^{\mathbf{W}}; 0, 2^b - 1\right) - z_i^{\mathbf{W}}\right]$$

Per-channel quantization is useful, when different weight channels have significantly different range of values



Layer Quantizatization

$$\begin{aligned}\widehat{\mathbf{W}}\widehat{\mathbf{x}} &= s_{\mathbf{w}}(\mathbf{W}_{\text{int}} - z_{\mathbf{w}})s_{\mathbf{x}}(\mathbf{x}_{\text{int}} - z_{\mathbf{x}}) \\ &= s_{\mathbf{w}}s_{\mathbf{x}}\mathbf{W}_{\text{int}}\mathbf{x}_{\text{int}} - {\color{red}s_{\mathbf{w}}z_{\mathbf{w}}s_{\mathbf{x}}\mathbf{x}_{\text{int}}} - {\color{blue}s_{\mathbf{w}}s_{\mathbf{x}}z_{\mathbf{x}}\mathbf{W}_{\text{int}}} + {\color{blue}s_{\mathbf{w}}z_{\mathbf{w}}s_{\mathbf{x}}z_{\mathbf{x}}}\end{aligned}$$

Quantization of Neural Networks: without additional data

Quantization Range for Weight Quantization

Quantization range selection: trade off between clipping and rounding errors

- **Min-Max approach**

$$q_{\min} = \min \mathbf{W}$$

$$q_{\max} = \max \mathbf{W}$$

  - no clipping error
  - outliers can lead to large rounding error

- **MSE**

$$\underset{q_{\min}, q_{\max}}{\arg\min} \left\| \mathbf{W} - \widehat{\mathbf{W}}\left(q_{\min}, q_{\max}\right) \right\|_F^2$$

  - helps to reduce rounding error caused by outliers

Range Settings for Weight Quantization

Model (FP32 accuracy)	ResNet18 (69.68)			MobileNetV2 (71.72)		
Bit-width	W8	W6	W4	W8	W6	W4
Min-Max	**69.57**	63.90	0.12	**71.16**	64.48	0.59
MSE	69.45	**64.64**	**18.82**	71.15	**65.43**	**13.77**
Min-Max (Per-channel)	69.60	69.08	44.49	71.21	68.52	18.40
MSE (Per-channel)	**69.66**	**69.24**	**54.67**	**71.46**	**68.89**	**27.17**

Table 1: Ablation study for different methods of range setting of (symmetric uniform) weight quantizers while keeping the activations in FP32. Average ImageNet validation accuracy (%) over 5 runs.

Source: https://arxiv.org/pdf/2106.08295.pdf

Quantization Range for Activation Quantization

For some layers not all elements might be equally important (e.g., when quantizing logits in the last layer of classification NN)

- **Cross-entropy based**: $$\underset{q_{\min}, q_{\max}}{\arg\min} \quad H\left(\psi(\mathbf{v}), \psi\left(\hat{\mathbf{v}}\left(q_{\min}, q_{\max}\right)\right)\right),$$

  where $H(\cdot, \cdot)$ denotes the cross-entropy function, ψ is the softmax function, and $\mathbf{v}$ is the logits vector.

- **Batch Normalization based**: $$\begin{aligned} q_{\min} &= \min(\boldsymbol{\beta} - \alpha\boldsymbol{\gamma}) \\ q_{\max} &= \max(\boldsymbol{\beta} + \alpha\boldsymbol{\gamma}) \end{aligned},$$

  where $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are vectors of per-channel learned shift and scale parameters, and $\alpha > 0$.

Note: Batch Normalization based approach doesn't require additional calibration data

Range Setting for Activation Quantization

Model (FP32 accuracy)	ResNet18 (69.68)			MobileNetV2 (71.72)		
Bit-width	A8	A6	A4	A8	A6	A4
Min-Max	69.60	68.19	18.82	70.96	64.58	0.53
MSE	69.59	67.84	31.40	71.35	67.55	13.57
MSE + Xent	**69.60**	**68.91**	**59.07**	**71.36**	**68.85**	**30.94**
BN ($\alpha = 6$)	69.54	68.73	23.83	71.32	65.20	0.66

Table 2: Ablation study for different methods of range setting of (asymmetric uniform) activation quantizers while keeping the weights in FP32. Average ImageNet validation accuracy (%) over 5 runs.

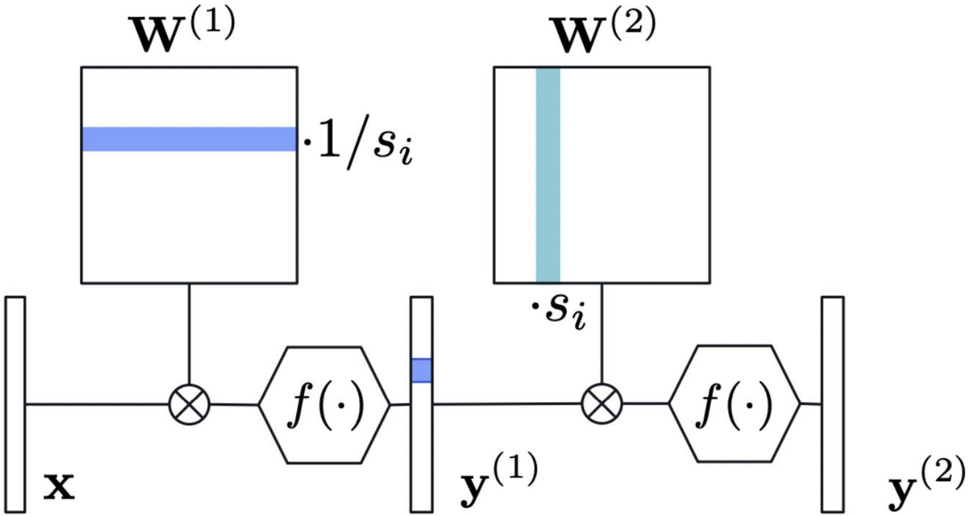
Source: https://arxiv.org/pdf/2106.08295.pdf

Quantization of Neural Networks:
with additional data, without fine-tuning

Cross-layer Equalization (CLE)

$$\mathbf{h} = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{y} = f(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

$$\begin{aligned}\mathbf{y} &= f(\mathbf{W}^{(2)} f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \\ &= f(\mathbf{W}^{(2)}\mathbf{S}\hat{f}(\mathbf{S}^{-1}\mathbf{W}^{(1)}\mathbf{x} + \mathbf{S}^{-1}\mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \\ &= f(\widetilde{\mathbf{W}}^{(2)}\hat{f}(\widetilde{\mathbf{W}}^{(1)}\mathbf{x} + \widetilde{\mathbf{b}}^{(1)}) + \mathbf{b}^{(2)})\end{aligned}$$



$$\widetilde{\mathbf{W}}^{(2)} = \mathbf{W}^{(2)}\mathbf{S}$$
$$\widetilde{\mathbf{W}}^{(1)} = \mathbf{S}^{-1}\mathbf{W}^{(1)}$$
$$\widetilde{\mathbf{b}}^{(1)} = \mathbf{S}^{-1}\mathbf{b}^{(1)}$$

S=diag(s) is a diagonal matrix,
S_ii denotes scaling factor s_i

Note: CLE helps to solve the problem with different ranges in weight channels (e.g., depth-wise layers) without per-channel quantization

High Biases Absorbtion

- High biases can lead to the differences in dynamic ranges of activations (especially after CLE)
- High biases can be absorbed into the next layer:

$$
\begin{aligned}
\mathbf{y} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \\
&= \mathbf{W}^{(2)}(f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{c} - \mathbf{c}) + \mathbf{b}^{(2)} \\
&= \mathbf{W}^{(2)}(f(\mathbf{W}^{(1)}\mathbf{x} + \widetilde{\mathbf{b}}^{(1)}) + \mathbf{c}) + \mathbf{b}^{(2)} \\
&= \mathbf{W}^{(2)}\widetilde{\mathbf{h}} + \widetilde{\mathbf{b}}^{(2)}
\end{aligned}
$$

Model	FP32	INT8
Original model	71.72	0.12
+ CLE	71.70	69.91
+ absorbing bias	71.57	**70.92**
Per-channel quantization	71.72	70.65

Table 3: Impact of cross-layer equalization (CLE) for MobileNetV2. ImageNet validation accuracy (%), evaluated at full precision and 8-bit quantization.

Source: https://arxiv.org/pdf/2106.08295.pdf

Quantization Bias Correction

- Quantization error is biased, that is $\left(\mathbb{E}[\mathbf{Wx}] \neq \mathbb{E}[\widehat{\mathbf{W}}\mathbf{x}]\right)$
    - Clipping error is the main contributor: strongly clipped outliers will likely lead to a shift in the expected distribution

$$\begin{aligned}\mathbb{E}[\widehat{\mathbf{y}}] &= \mathbb{E}[\widehat{\mathbf{W}}\mathbf{x}] \\ &= \mathbb{E}[(\mathbf{W} + \Delta\mathbf{W})\mathbf{x}] \\ &= \mathbb{E}[\mathbf{Wx}] + \mathbb{E}[\Delta\mathbf{Wx}] \\ &= \mathbb{E}[\mathbf{y}] + \Delta\mathbf{W}\mathbb{E}[\mathbf{x}].\end{aligned}$$

- Corrected output: $\mathbb{E}\left[\mathbf{y}_{\text{corr}}\right] = \mathbb{E}\left[\widehat{\mathbf{W}}\mathbf{x}\right] - \Delta\mathbf{W}\,\mathbb{E}\left[\mathbf{x}\right] = \mathbb{E}\left[\mathbf{y}\right]$
- Empirical correction (requires calibration data): $\Delta\mathbf{W}\mathbb{E}[\mathbf{x}] = \mathbb{E}[\widehat{\mathbf{W}}\mathbf{x}] - \mathbb{E}[\mathbf{Wx}].$
- Analytical correction (use BN statistics):

$$\begin{aligned}\mathbb{E}[\mathbf{x}] &= \mathbb{E}\left[\mathrm{ReLU}\left(\mathbf{x}^{\text{pre}}\right)\right] \\ &= \gamma\mathcal{N}\left(\frac{-\boldsymbol{\beta}}{\boldsymbol{\gamma}}\right) + \boldsymbol{\beta}\left[\mathbf{1} - \Phi\left(\frac{-\boldsymbol{\beta}}{\boldsymbol{\gamma}}\right)\right]\end{aligned}$$

Source: https://arxiv.org/pdf/1906.04721.pdf

Model	FP32	INT8
Original Model	71.72	0.12
+ bias correction	71.72	**52.02**
CLE + bias absorption	71.57	70.92
+ bias correction	71.57	**71.19**

Table 4: Impact of bias correction for MobileNetV2. ImageNet validation accuracy (%) evaluated at full precision and 8-bit quantization.

Source: https://arxiv.org/pdf/2106.08295.pdf

NN Quantization: w/ data, w/o fine-tuning

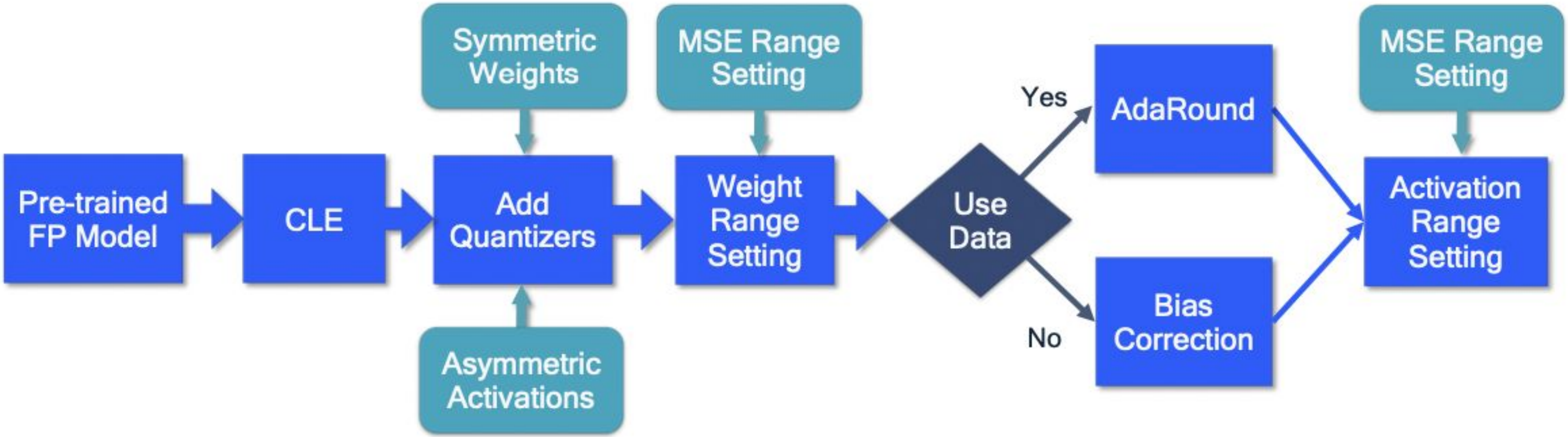


Figure 8: Standard PTQ pipeline. Blue boxes represent required steps and the turquoise boxes recommended choices.

Source: https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

Models	FP32	Per-tensor W8A8	Per-tensor W4A8	Per-channel W8A8	Per-channel W4A8
ResNet18	69.68	69.60	68.62	69.56	68.91
ResNet50	76.07	75.87	75.15	75.88	75.43
MobileNetV2	71.72	70.99	69.21	71.16	69.79
InceptionV3	77.40	77.68	76.48	77.71	76.82
EfficientNet lite	75.42	75.25	71.24	75.39	74.01
DeeplabV3	72.94	72.44	70.80	72.27	71.67
EfficientDet-D1	40.08	38.29	0.31	38.67	35.08
BERT-base†	83.06	82.43	81.76	82.77	82.02

Table 6: Performance (average over 5 runs) of our standard PTQ pipeline for various models and tasks. DeeplabV3 (MobileNetV2 backbone) is evaluated on Pascal VOC (mean intersection over union), EfficientDet-D1 on COCO 2017 (mean average precision), BERT-base on the GLUE benchmark and other models on ImageNet (accuracy). We evaluate all models on the respective validation sets. Higher is better in all cases. †A few quantized activations are kept in higher precision (16 bits).

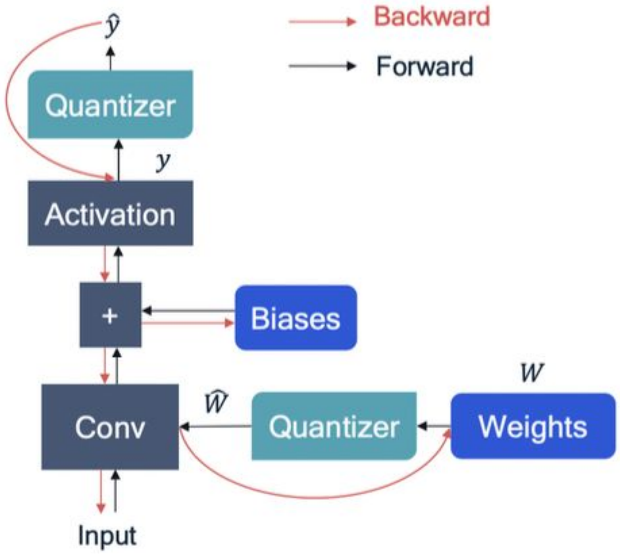
Source: https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

Quantization of Neural Networks:
with additional data,
with fine-tuning

Back-propagation for Quantization Block

$$\frac{\partial \lfloor y \rceil}{\partial y} = 1$$

$$\begin{aligned}
\frac{\partial \widehat{\mathbf{x}}_i}{\partial \mathbf{x}_i} &= \frac{\partial q(\mathbf{x}_i)}{\partial \mathbf{x}_i} \\
&= s \cdot \frac{\partial}{\partial \mathbf{x}_i} \operatorname{clamp}\left(\left\lfloor \frac{\mathbf{x}_i}{s} \right\rceil; n, p\right) + 0 \\
&= \begin{cases} s \cdot \frac{\partial \lfloor \mathbf{x}_i / s \rceil}{\partial (\mathbf{x}_i / s)} \frac{\partial (\mathbf{x}_i / s)}{\partial \mathbf{x}_i} & \text{if } q_{\min} \leq \mathbf{x}_i \leq q_{\max}, \\ s \cdot \frac{\partial n}{\partial \mathbf{x}_i} & \text{if } \mathbf{x}_i < q_{\min}, \\ s \cdot \frac{\partial p}{\partial \mathbf{x}_i} & \text{if } \mathbf{x}_i > q_{\max}, \end{cases} \\
&= \begin{cases} 1 & \text{if } q_{\min} \leq \mathbf{x}_i \leq q_{\max}, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$



NN Quantization: w/ data, w/ fine-tuning

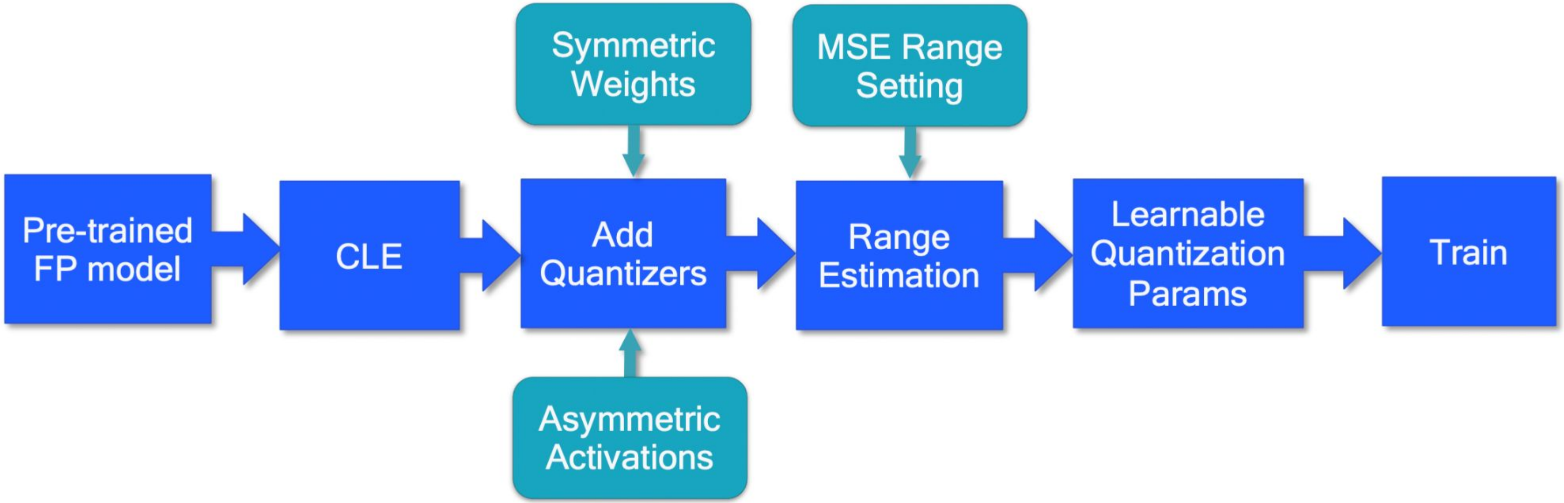


Figure 12: Standard quantization-aware training pipeline. The blue boxes represent the steps and the turquoise boxes recommended choices.

Source: https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

Models	FP32	Per-tensor			Per-channel		
		W8A8	W4A8	W4A4	W8A8	W4A8	W4A4
ResNet18	69.68	70.38	69.76	68.32	70.43	70.01	68.83
ResNet50	76.07	76.21	75.89	75.10	76.58	76.52	75.53
InceptionV3	77.40	78.33	77.84	77.49	78.45	78.12	77.74
MobileNetV2	71.72	71.76	70.17	66.43	71.82	70.48	66.89
EfficientNet lite	75.42	75.17	71.55	70.22	74.75	73.92	71.55
DeeplabV3	72.94	73.99	70.90	66.78	72.87	73.01	68.90
EfficientDet-D1	40.08	38.94	35.34	24.70	38.97	36.75	28.68
BERT-base	83.06	83.26	82.64	78.83	82.44	82.39	77.63

Table 10: Performance (average over 3 runs) of our standard QAT pipeline for various models and tasks. DeeplabV3 (MobileNetV2 backbone) is evaluated on Pascal VOC (mean intersection over union), EfficientDet-D1 on COCO 2017 (mean average precision), BERT-base on the GLUE benchmark and all other models on ImageNet (accuracy). We evaluate all models on the respective validation sets. Higher is better in all cases.

Source: https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/