April 2020

@qualcomm_tech

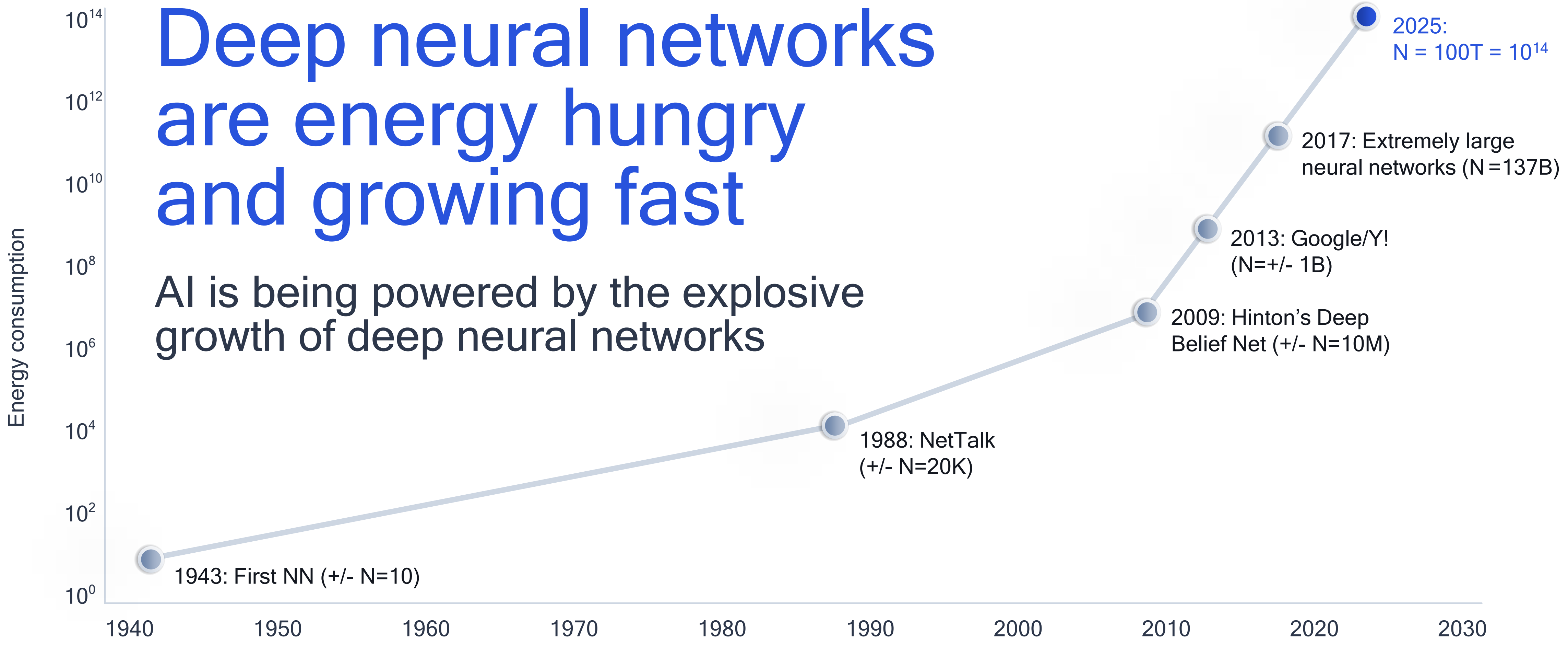
Qualcomm

Enabling power-efficient AI through quantization

Qualcomm Technologies Inc.

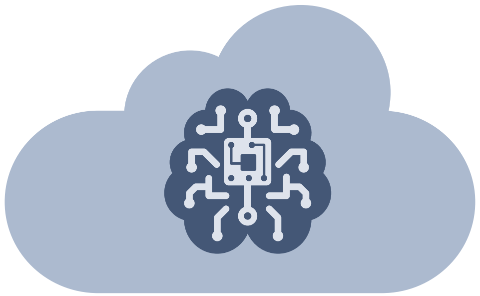
Deep neural networks are energy hungry and growing fast

AI is being powered by the explosive growth of deep neural networks



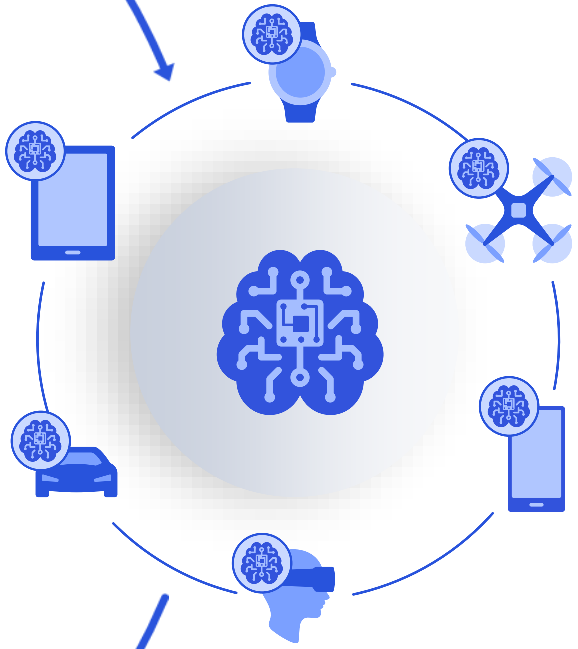
Source: Welling

2025 | Will we have reached the capacity of the human brain?
Energy efficiency of a brain is 100x better than current hardware



On-device intelligence is paramount

Process data closest to the source, complement the cloud

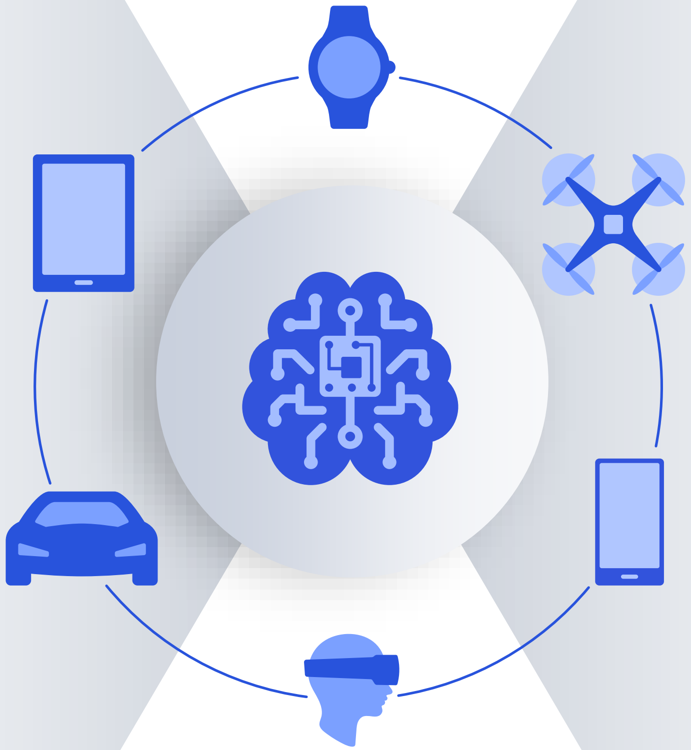


- Privacy
- Reliability
- Low latency
- Efficient use of network bandwidth

The AI power and thermal ceiling

The challenge of AI workloads

- Very compute intensive
- Complex concurrencies
- Real-time
- Always-on

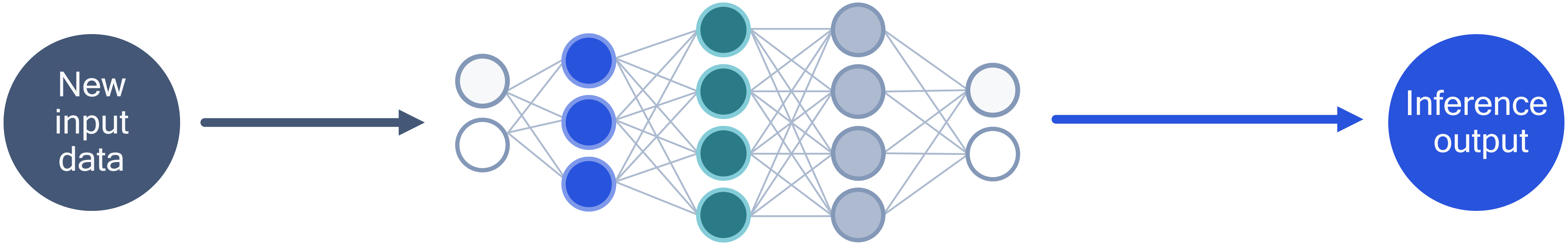


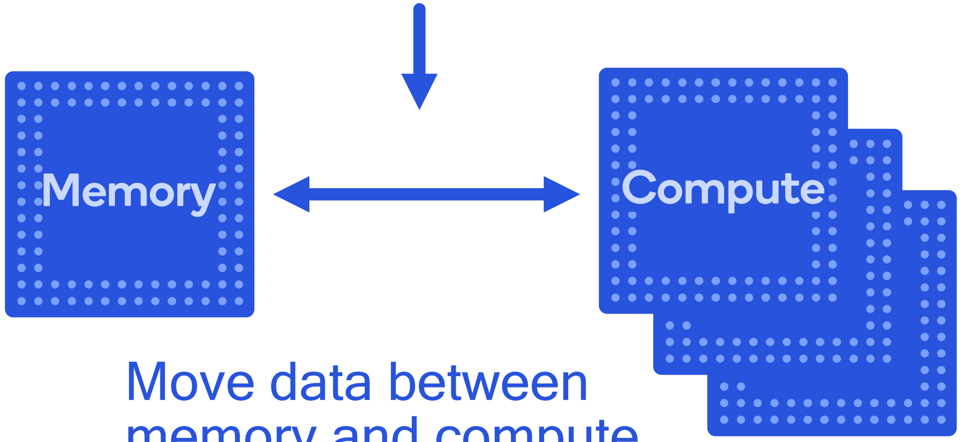
Constrained mobile environment

- Must be thermally efficient for sleek, ultra-light designs
- Requires long battery life for all-day use
- Storage/memory bandwidth limitations

Advancing AI research to increase power efficiency

Trained neural network model



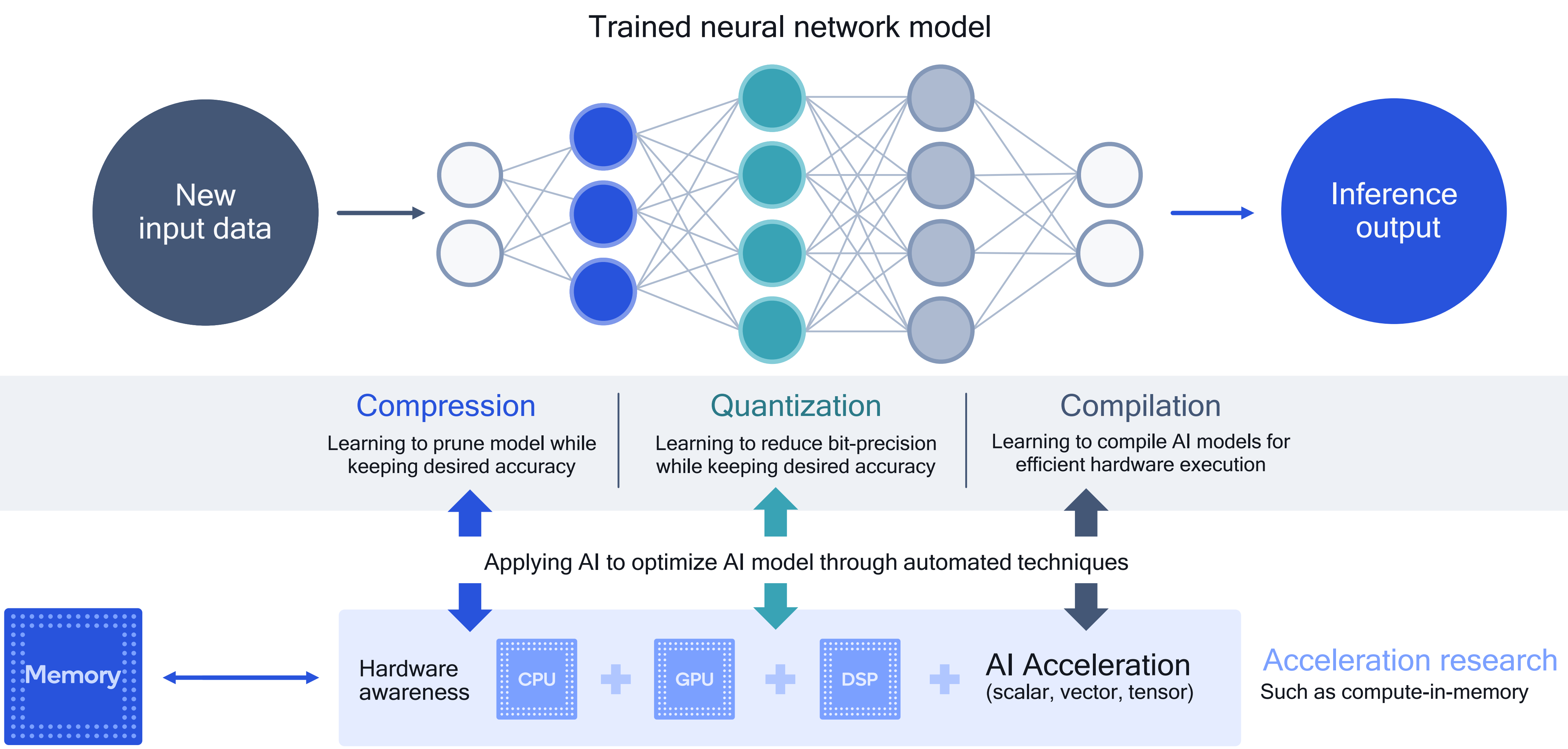


Compute operations

- Vector and matrix manipulations
- CPU, GPU, DSP, and AI acceleration

Move data between memory and compute

- Move pieces of input data and AI model from memory to compute
- Send partial results back to memory



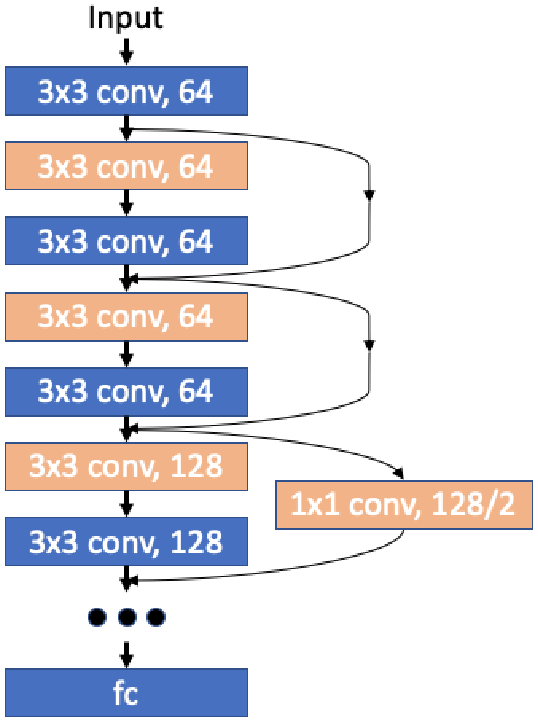
Advancing AI research to increase power efficiency

What is quantization?

What is neural network quantization?

For any given trained neural network:

- Store weights in n bits
- Compute calculations in n bits



Quantization analogy
Similar to representing the pixels of an image with less bits

Quantizing AI models offers significant benefits

Memory usage

8-bit versus 32-bit weights and activations stored in memory

01010101	01010101	01010101	01010101

↓

01010101

Power consumption

Significant reduction in energy for both computations and memory access

Add energy (pJ)	
INT8	FP32
0.03	0.9
30X energy reduction	

Mult energy (pJ)	
INT8	FP32
0.2	3.7
18.5X energy reduction	

Mem access energy (pJ)	
Cache (64-bit)	
8KB	10
32KB	20
1MB	100
DRAM	1300-2600
Up to 4X energy reduction	

Latency

With less memory access and simpler computations, latency can be reduced

Silicon area

Integer math or less bits require less silicon area compared to floating point math and more bits

Add area (μm^2)	
INT8	FP32
36	4184
116X area reduction	

Mult area (μm^2)	
INT8	FP32
282	7700
27X area reduction	

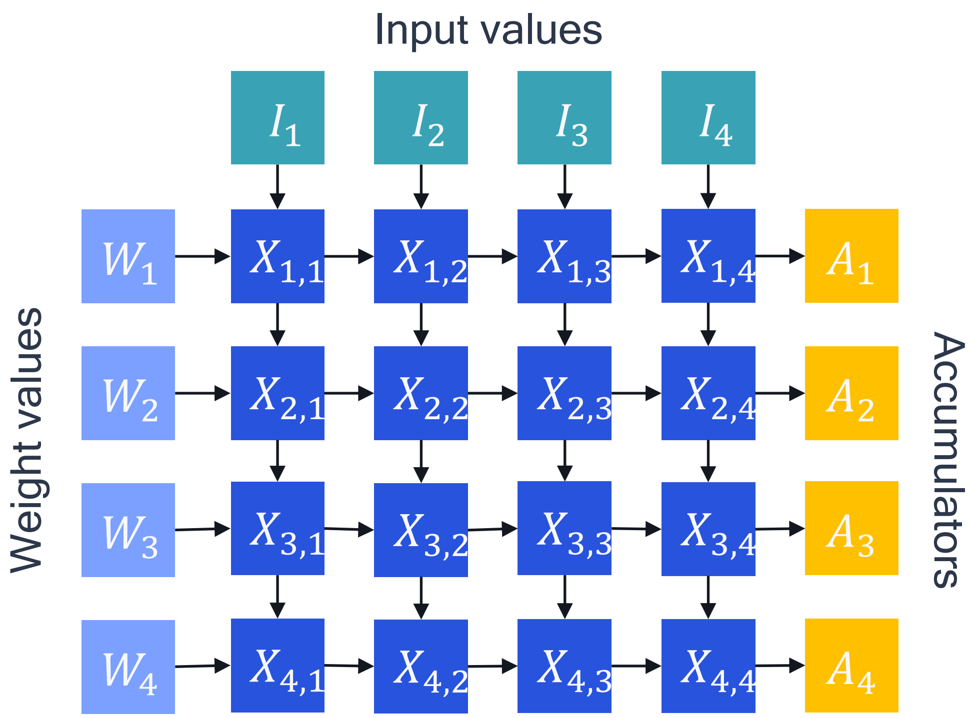
Matrix math is the primary operation of neural nets

A running example to showcase how to make these operations more efficient

$$A^T = \begin{pmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix} \quad B = \begin{pmatrix} 0.41 & 0.25 & 0.73 & 0.66 \\ 0.00 & 0.41 & 0.41 & 0.57 \\ 0.42 & 0.24 & 0.71 & 1.00 \\ 0.39 & 0.82 & 0.17 & 0.35 \end{pmatrix} \quad b = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{pmatrix}$$

How to most efficiently calculate $AB+b$?

A schematic mac array for efficient computation



The array efficiently calculates the dot product between multiple vectors

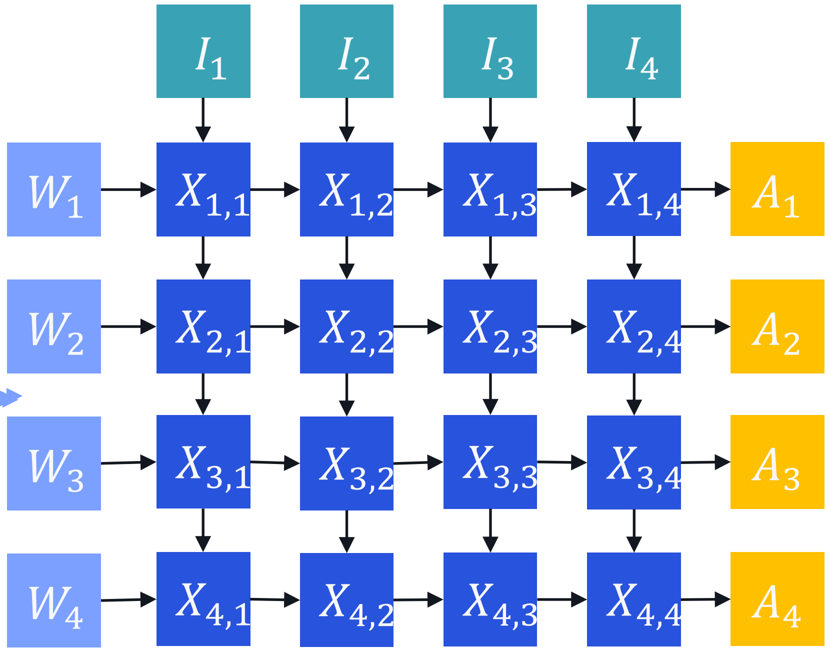
$$A_i = W_i \cdot I_1 + W_i \cdot I_2 + W_i \cdot I_3 + W_i \cdot I_4$$

Step-by-step matrix multiply calculation on mac array

$$B = \begin{pmatrix} 0.41 & 0.25 & 0.73 & 0.66 \\ 0.00 & 0.41 & 0.41 & 0.57 \\ 0.42 & 0.24 & 0.71 & 1.00 \\ 0.39 & 0.82 & 0.17 & 0.35 \end{pmatrix}$$

Load full matrix into each individual block

$$A^T = \begin{pmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix}$$



$$Out^T = \begin{pmatrix} \quad \end{pmatrix}$$

Quantization comes at a cost of lost precision

Instead of storing weights as floating point values, store them as integers with a scale factor:

$$A^T = \begin{pmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix} \approx \frac{1}{255} \begin{pmatrix} 247 & 163 & 189 & 255 \\ 148 & 214 & 214 & 207 \\ 0 & 46 & 229 & 71 \\ 145 & 245 & 204 & 207 \end{pmatrix} = s \cdot \boldsymbol{Z}$$

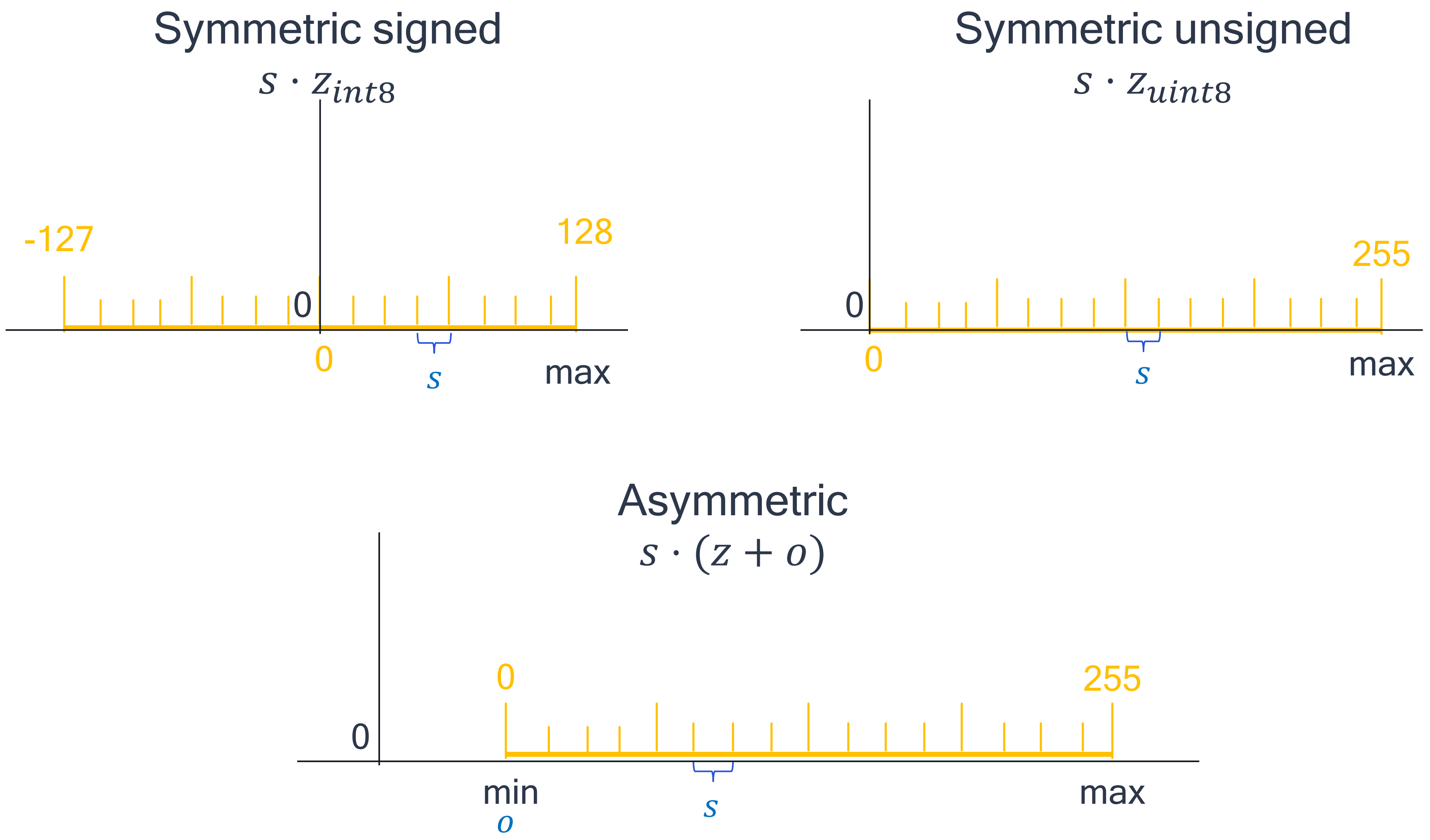
This means that for every weight tensor or activation tensor, we only have to store an INT8 weight matrix and 1 scaling factor, instead of a FP32 weight matrix.

However, quantization is not free:

$$A^T - s \cdot \boldsymbol{Z} = \frac{1}{255} \begin{pmatrix} 0.35 & 0.20 & -0.3 & 0 \\ -0.1 & 0.20 & 0.20 & -0.45 \\ 0.00 & -0.1 & -0.5 & 0.40 \\ 0.35 & -0.2 & 0 & -0.45 \end{pmatrix}$$

Different types of quantization have pros and cons

Symmetric, asymmetric, signed, and unsigned quantization



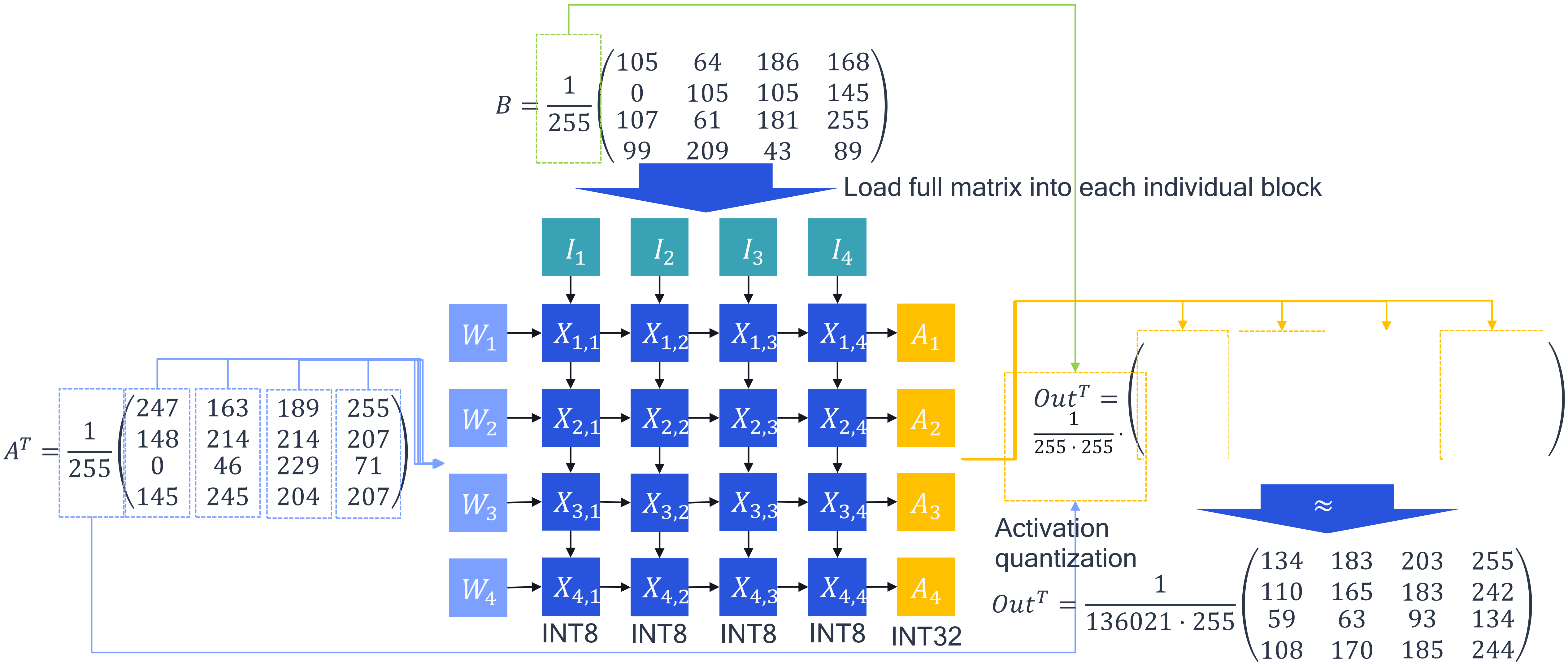
Fixed point grid

Floating point grid

s: scale factor

o: offset

An example calculation using symmetric quantization



What type of quantization should you use?

W is the weight matrix

X is the input of a layer

Symmetric quantization

$$\begin{aligned} W \cdot X &\approx s_1(W_{int}) \cdot s_2(X_{int}) \\ &= s_1 s_2(W_{int} \cdot X_{int}) \end{aligned}$$

Asymmetric quantization

$$\begin{aligned} W \cdot X &\approx s_1(W_{int} + o_1) \cdot s_2(X_{int} + o_2) \\ &= s_1 s_2(W_{int} \cdot X_{int}) + \underbrace{s_1 s_2 o_1 X_{int}}_{\text{Unavoidable overhead}} + \underbrace{s_1 s_2 o_2 W_{int} + s_1 o_1 s_2 o_2}_{\text{Precompute, add to layer bias}} \end{aligned}$$

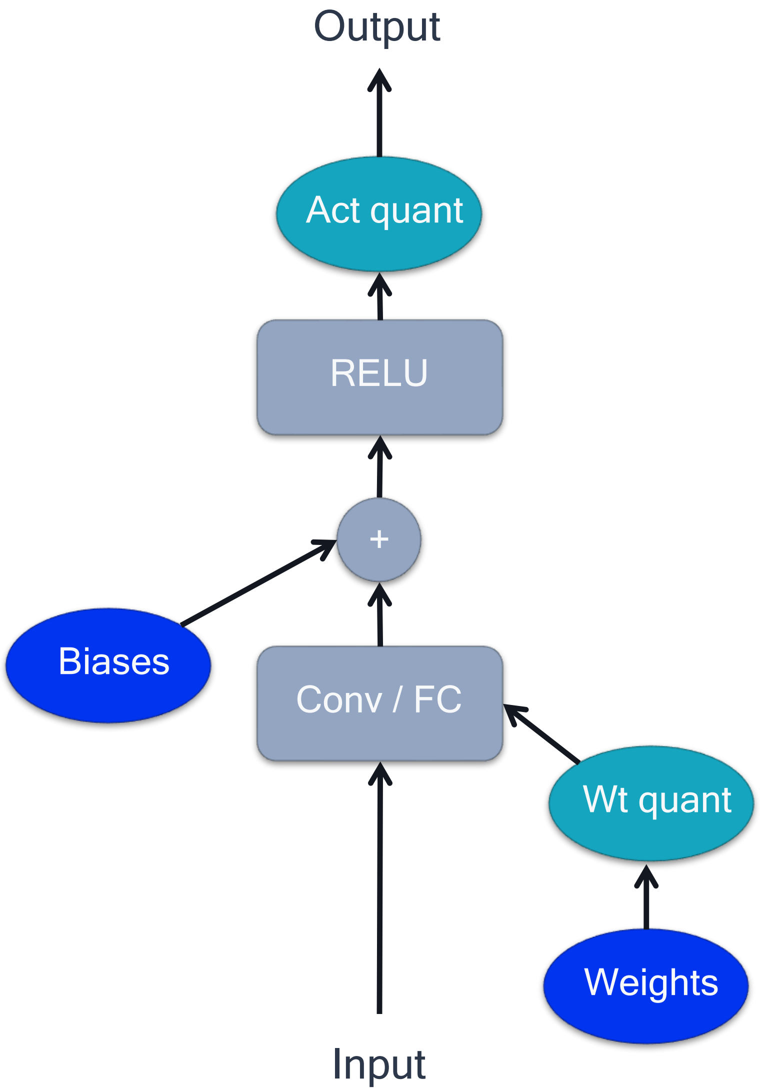
Same calculation

Asymmetric weight calculations incur 10-15% extra energy consumption

Symmetric weights and asymmetric activations are the best option

Simulating quantization

How to accurately simulate quantization

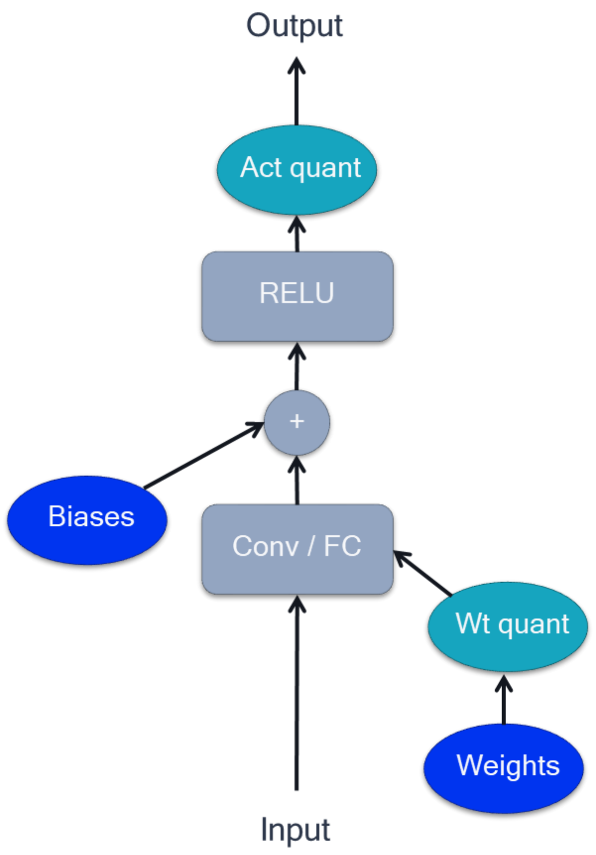


Quantization is generally simulated in floating point instead of actually running in integer math

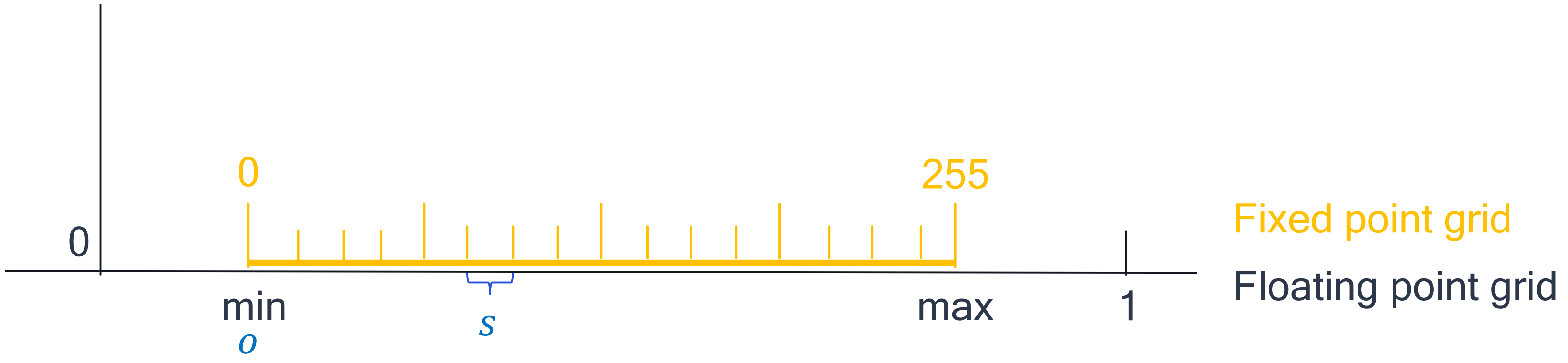
Simulated quantization ops are added in the neural network after each usage of weights, and after every ‘operation’

- No dedicated kernels are necessary
- This easily allows for flexible bit-widths 1,2,3,4,…
- Makes GPU speed-up easy
- Biases are not quantized

How to simulate asymmetric quantization with b bits



What happens in those simulated quantization blocks?



Given a floating point value x, we quantize:

$$x_{int} = round\left(\frac{x - o}{s}\right)$$
$$x_Q = clamp(x_{int}, min = 0, max = 2^b - 1)$$
$$x_{float} = x_Q \cdot s + 0$$

The procedure turns any value into a '8-bit quantized' value, while all calculations are done in float32

Definitely slower than training without quantization operations

These operations are added everywhere in the network

min, max are set for activations based on passing of batches of data through the whole network

How accurate is the quantization simulation?

Very accurate – the rounding errors are tiny

Model	Top1 simulated	Top1 on-device
Resnet 50	75.76%	75.67%
MobileNetV2	70.12%	70.01%

Hardly any difference between quantization simulation and real hardware

How well does quantizing a model work?

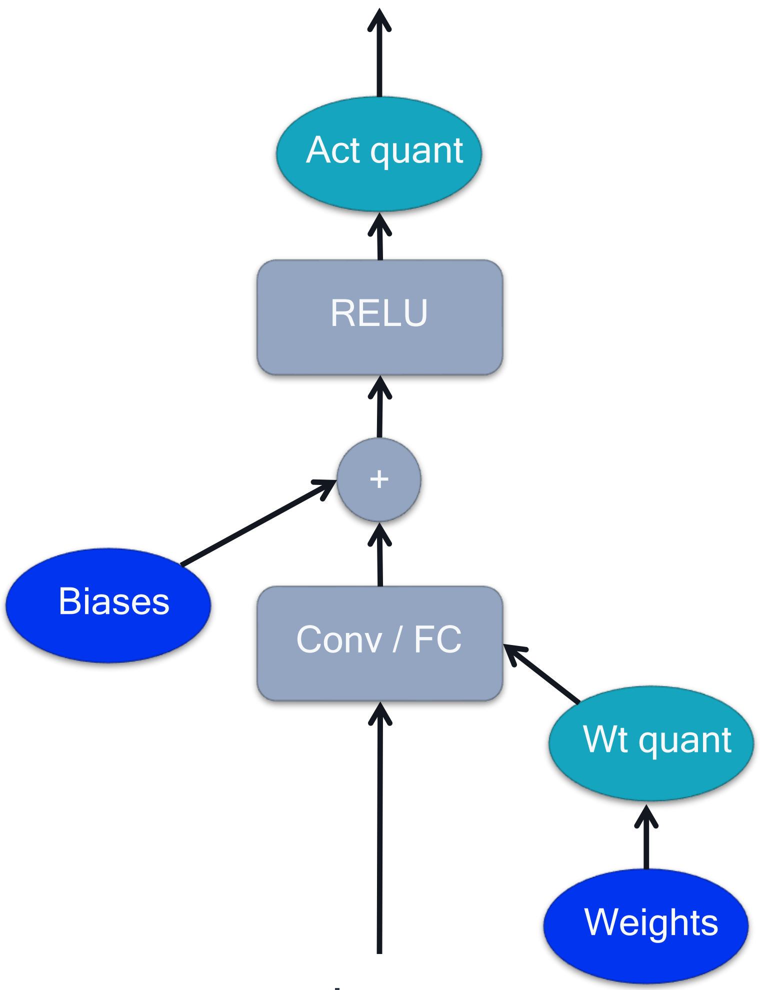
Quantizing some computer vision models to 8-bit weights and activations

Model	Top1 accuracy	Top1 quantized
InceptionV3	0.78	0.78
NasnetMobile	0.74	0.722
Resnet 50	0.756	0.75
MobileNetV2	0.749	0.004
DeepLabV3	0.729	0.41
FastDVDNet	0.862	0.696

16-bit fixed-point quantization is always fine

Quantization-aware training

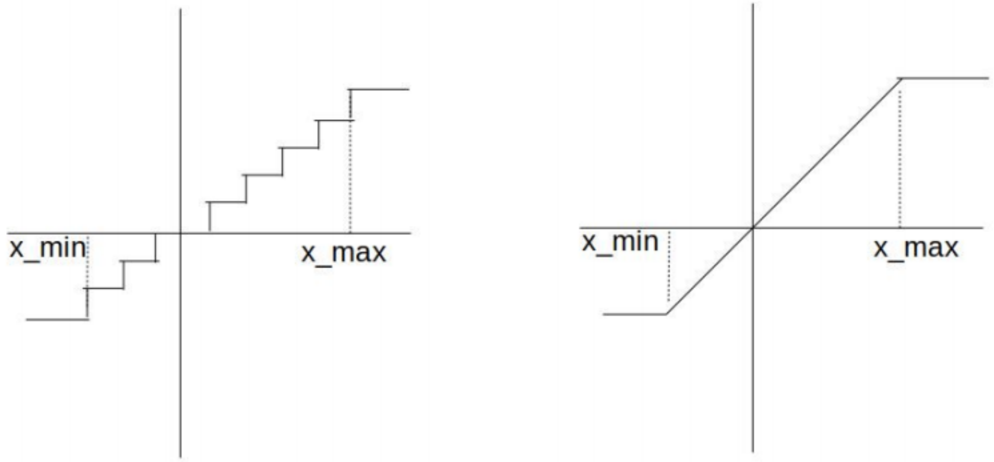
Overcoming the challenges of quantized training



Quantized training challenges:

- ”Round” doesn’t have a proper gradient.
- “Clamp” kills the gradient

Solution: Redefine gradient op as ”straight-through”*

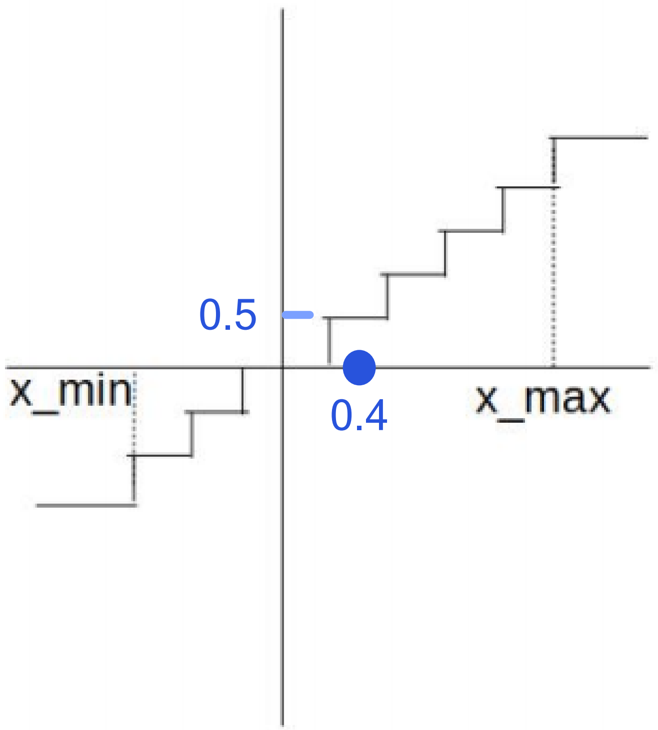


Forward pass Backward pass

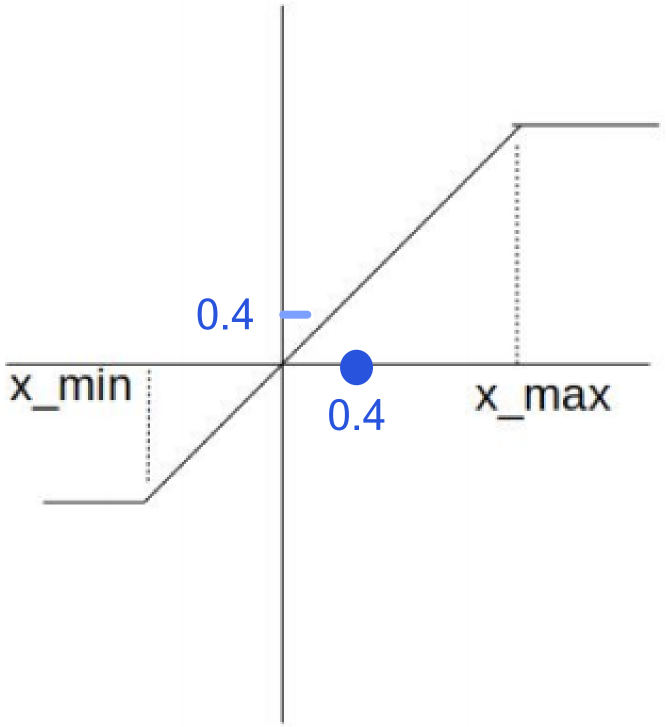
Then train with gradient descent as usual

*Bengio et al. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation

Problem: A mismatch between real and quantized values



Forward pass

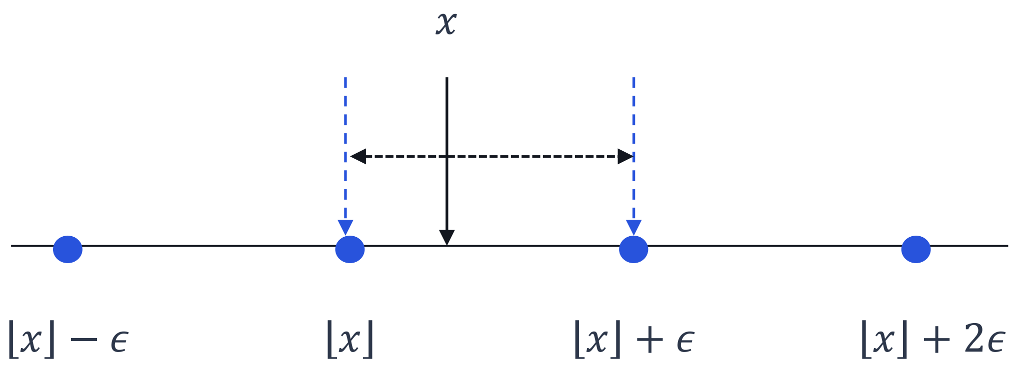


Backward pass

Each calculated gradient is a little bit ’wrong’.
This compounds over the whole network and makes training deep networks difficult.

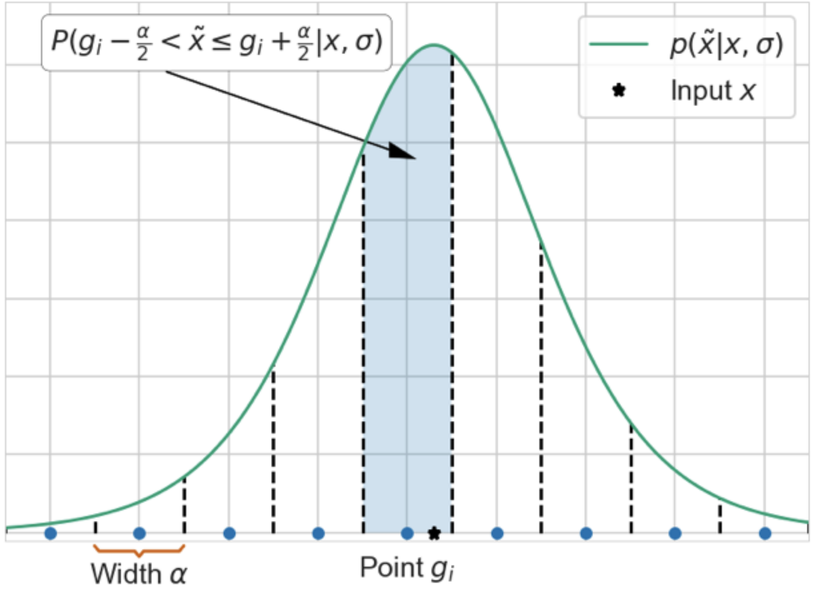
Addressing biased gradients in quantized training

Through stochastic rounding and relaxed quantization



$$q(x)=\begin{cases} \lfloor x \rfloor & p:\ 1-\frac{x-\lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & p:\frac{x-\lfloor x \rfloor}{\epsilon} \end{cases}$$

Stochastic rounding[1]



Relaxed quantization[2]

Not a big problem for 8-bit quantization

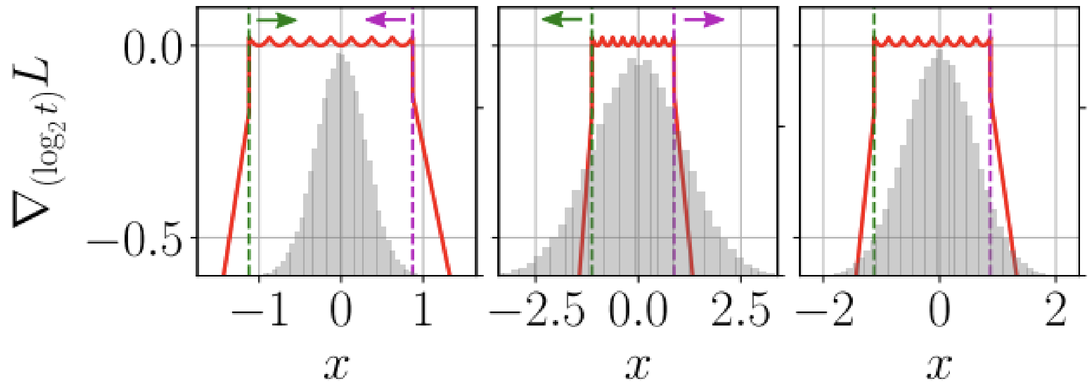
1) Gupta et al. 2015 Deep Learning with Limited Numerical Precision
2) Louizos et al. 2019. Relaxed Quantization for discretized neural networks

Major improvements from learning the min and max values

Dynamic ranges[1]

Adjust [min, max] on the fly while training,
such as when overflow occurs

Fully trainable min, max[2]



Parametrize min and max,
and train them alongside full network

With learned min and max values,
ImageNet models trained to 4-bit weights and 4-bit activations have hardly any loss

1) Wu et al. 2018 Training and Inference with Integers in Deep Neural Networks
2) Jain et al. 2019 Trained Uniform Quantization for Accurate and Efficient Neural Network Inference on Fixed-Point Hardware

Quantized training solves a lot of accuracy problems

Model	Top1 accuracy	Top1 quantized	Fine-tuned
InceptionV3	0.78	0.78	0.78
NasnetMobile	0.74	0.722	0.73
Resnet 50	0.756	0.75	0.752
MobileNetV2	0.749	0.004	0.735
DeeplabV3	0.729	0.414	0.725

If possible, fine-tune after compression and quantization for optimal performance

Results from Jacob et al. 2018 Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only inference

Making quantization practical for the masses

Our research focuses on quantization techniques that maintain accuracy while reducing engineering effort

Ideal properties for simple quantization

No data

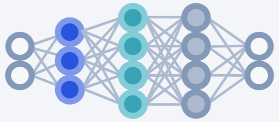
Data might not be available

No back propagation

Retraining is time intensive and sensitive to hyper parameters

No architecture changes

Requires training from scratch with quantization in mind



DFQ:
Data-Free Quantization

A better method for no-data quantization

Data-Free Quantization through Weight Equalization and Bias Correction

Markus Nagel
Qualcomm Technologies Netherlands B.V.

Mart van Baalen
Qualcomm Technologies Netherlands B.V.

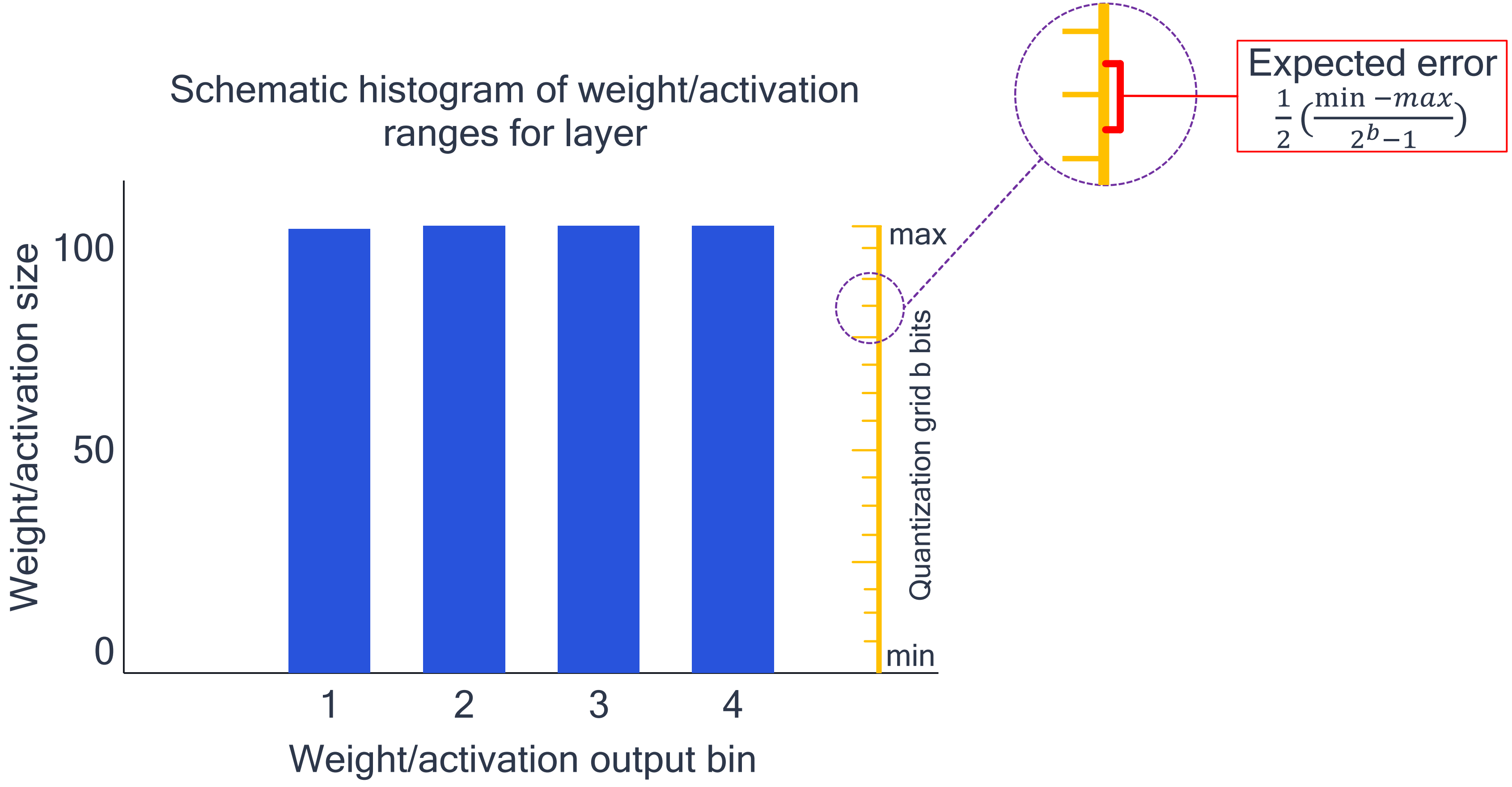
Tijmen Blankevoort
Qualcomm Technologies Netherlands B.V.

Max Welling
Qualcomm Technologies Netherlands B.V.

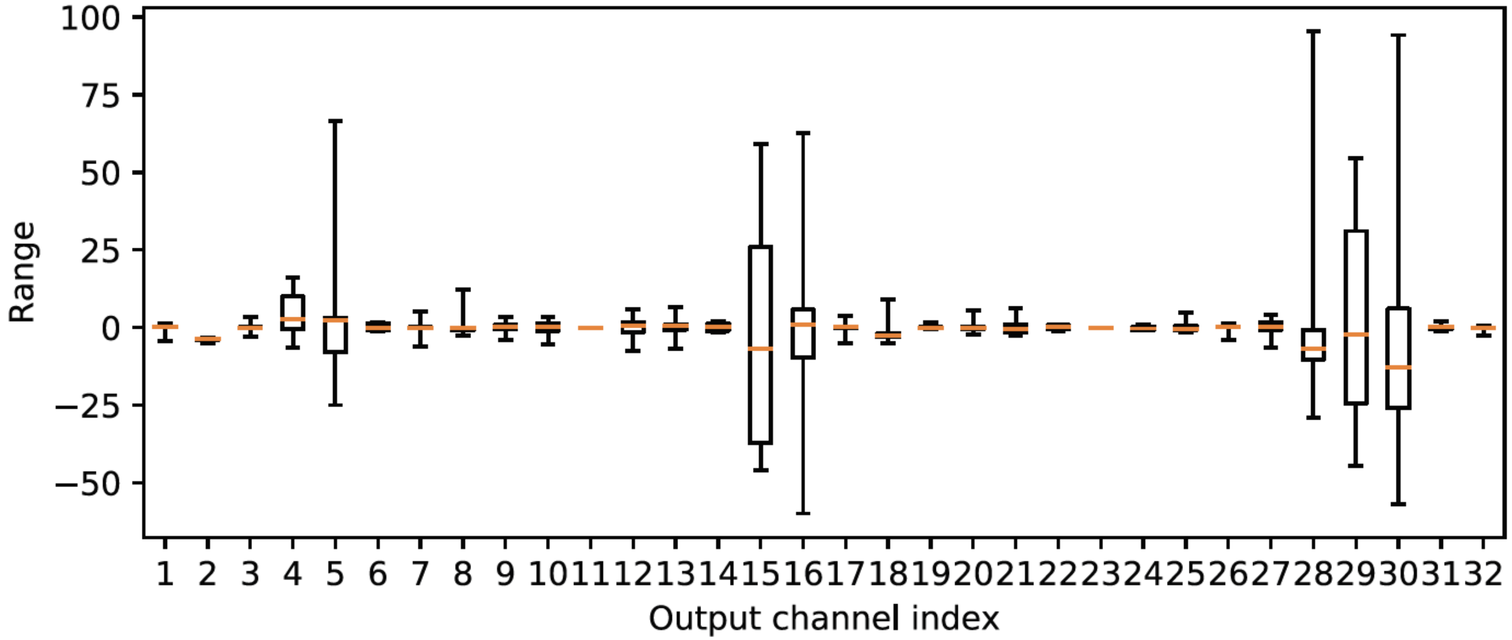
- Paper introduces a method for quantization without the use of data
- Excellent results without any training

Nagel et al. 2019 Data-Free Quantization Through Weight Equalization and Bias Correction

Visualizing quantization rounding error

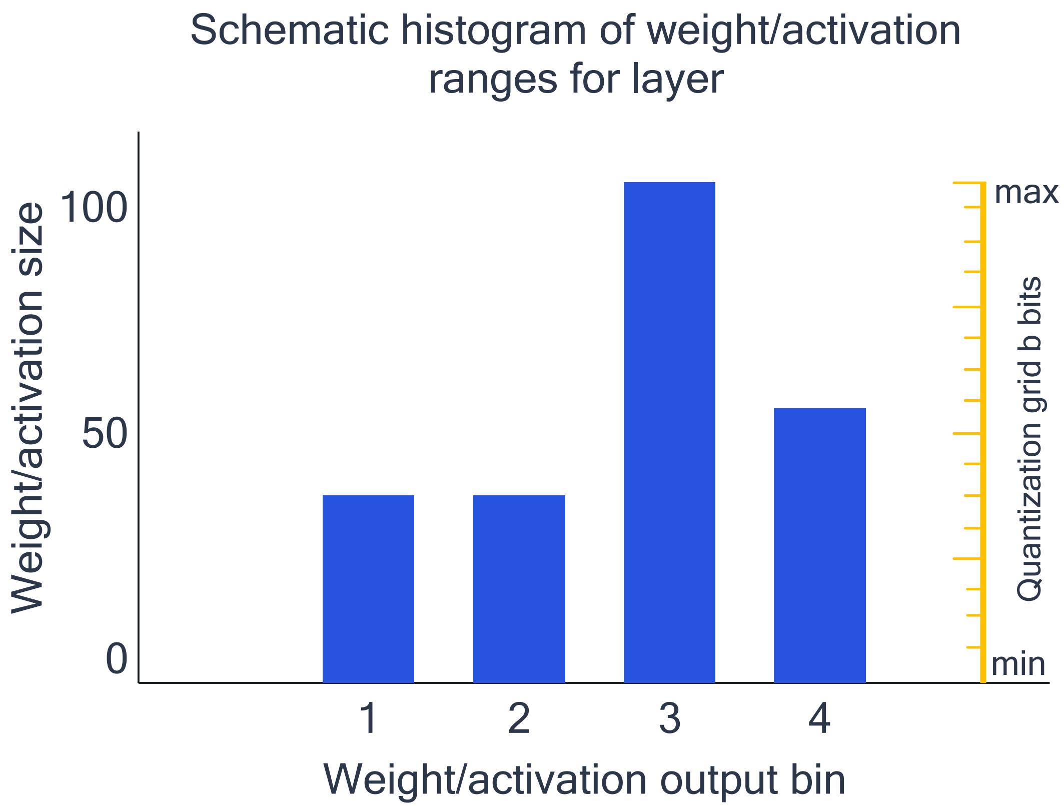


Imbalanced weights is a common problem in practice



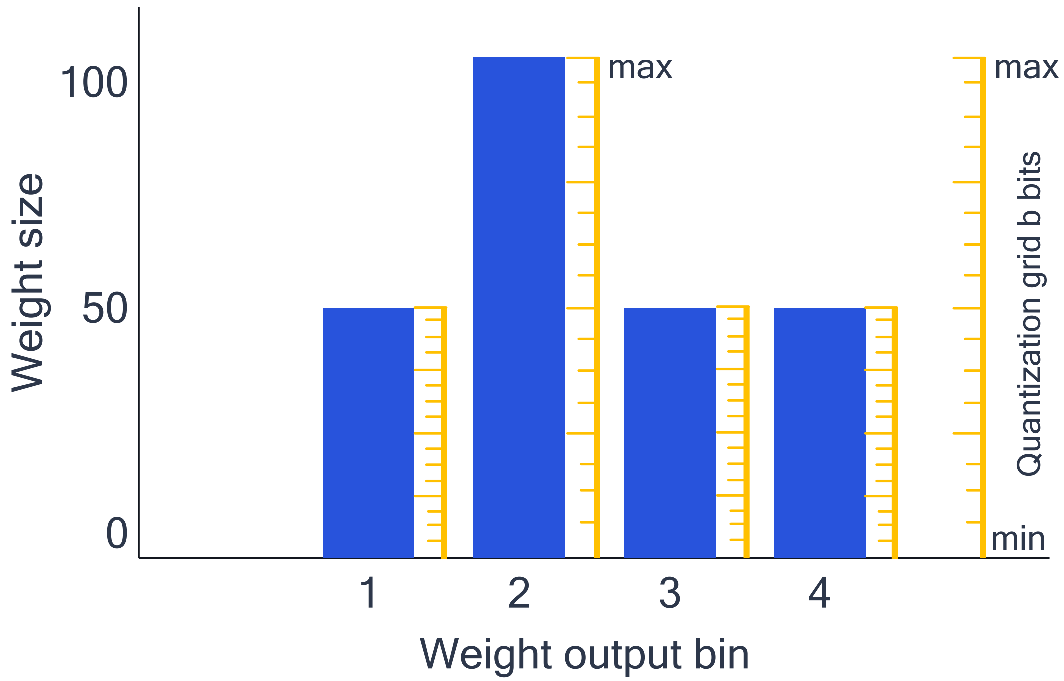
Distributions of weights in 2nd layer of MobileNetV2 (ImageNet)

The problem occurs because of mismatched ranges



Per-channel quantization

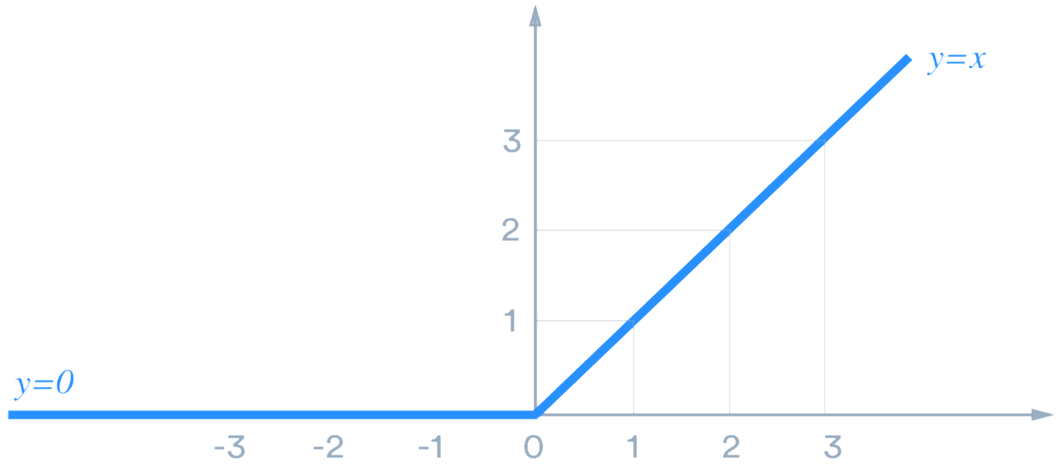
Schematic histogram of weight ranges for layer



- Per-channel quantization[1] keeps a scale *s_i* (and offset *o_i*) for each output *i*
- Not all hardware supports this

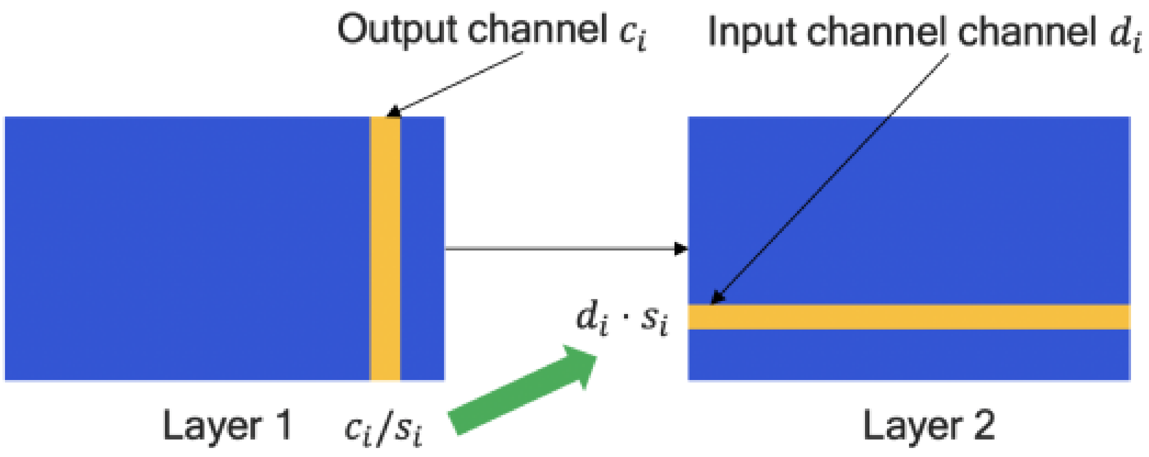
[1] Krishnamoorthi 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper

Cross-layer equalization scales weights in neighboring layers for better quantization



$$relu(x) = \max(0, x)$$

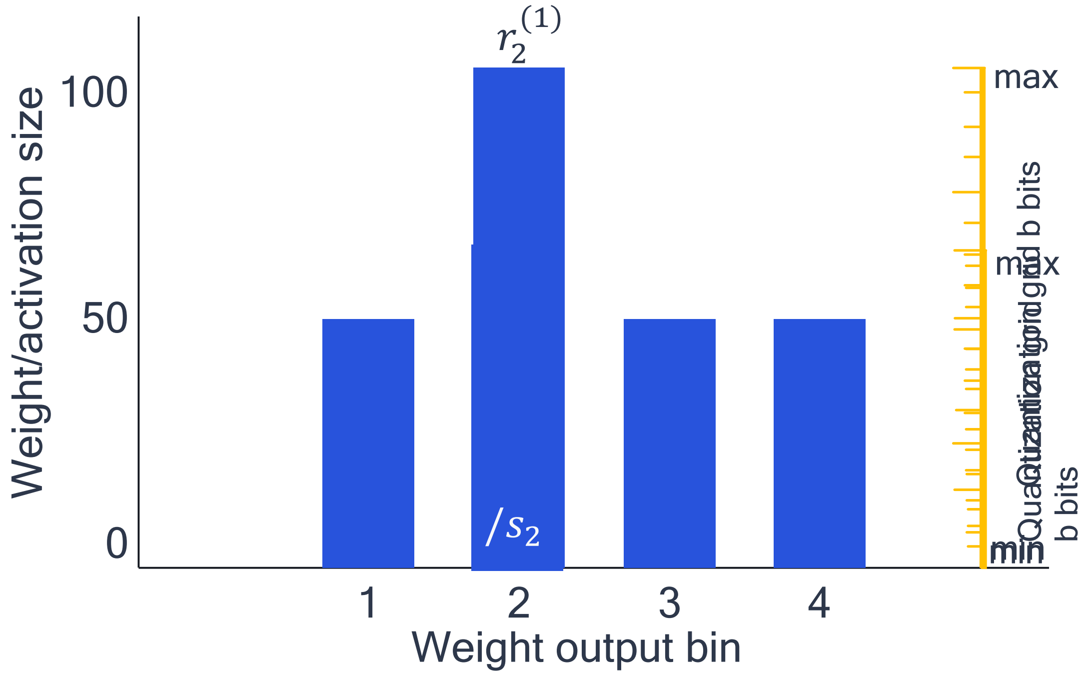
We have that
$$relu(sx) = s \cdot relu(x)$$



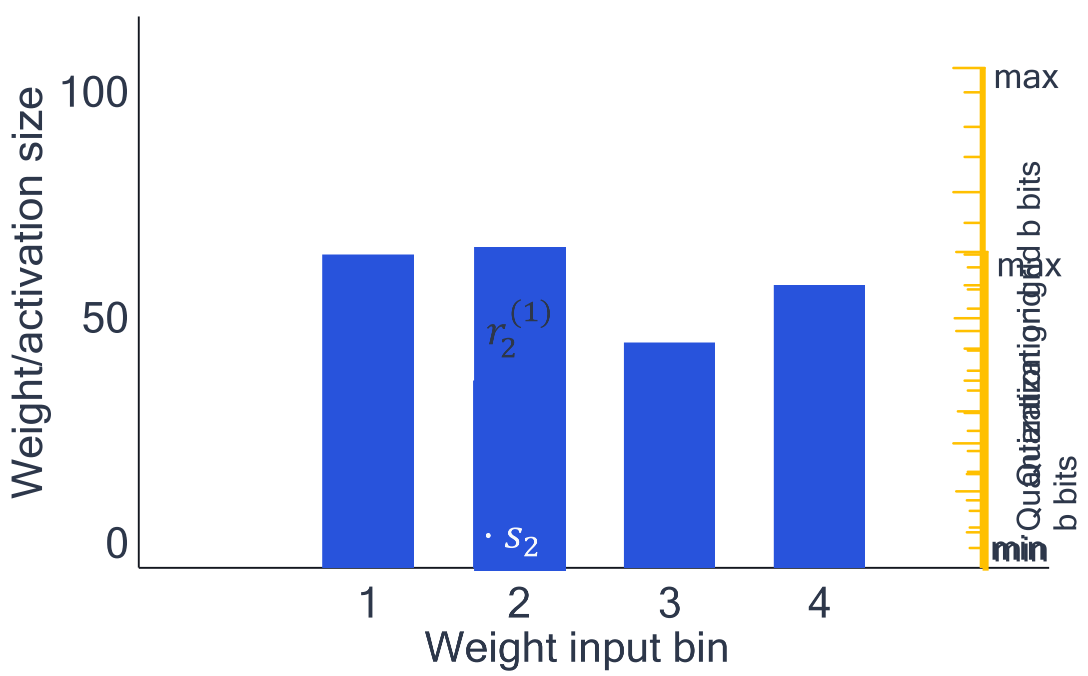
We can scale two layers with a (P)Relu together to optimize it for quantization

Finding the scaling factors for cross-layer equalization

Schematic histogram of weight ranges for layer 1



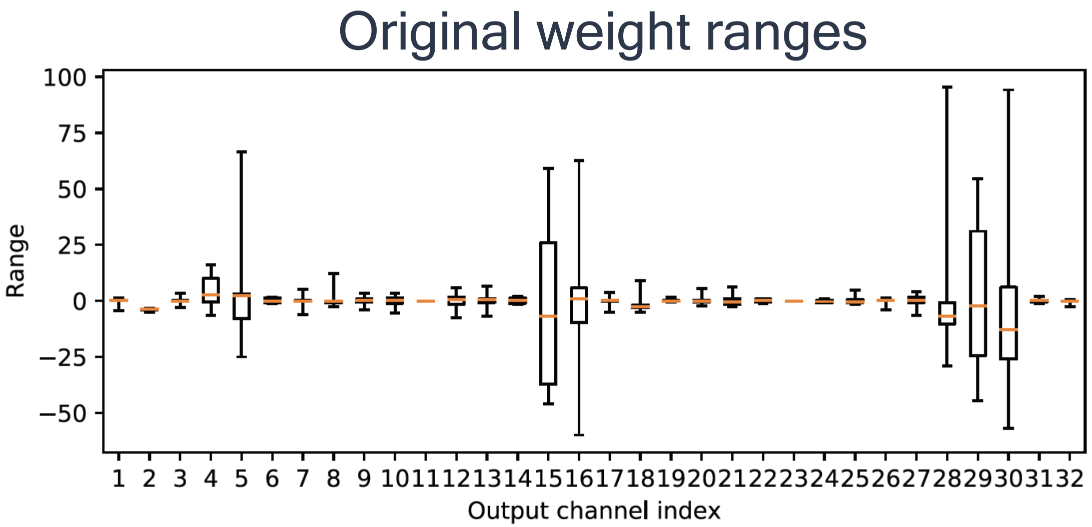
Schematic histogram of weight ranges for layer 2

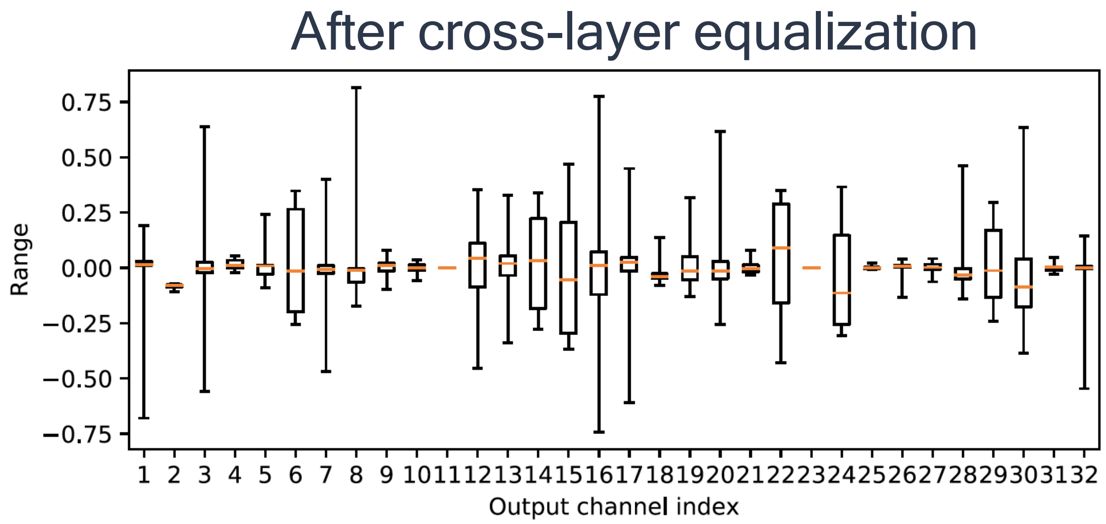


Equalize the outputs of layer 1 with the inputs of layer 2

by setting $s_i = \frac{1}{r_i}\sqrt{r_i^{(1)} r_i^{(2)}}$

Cross-layer equalization significantly improves accuracy

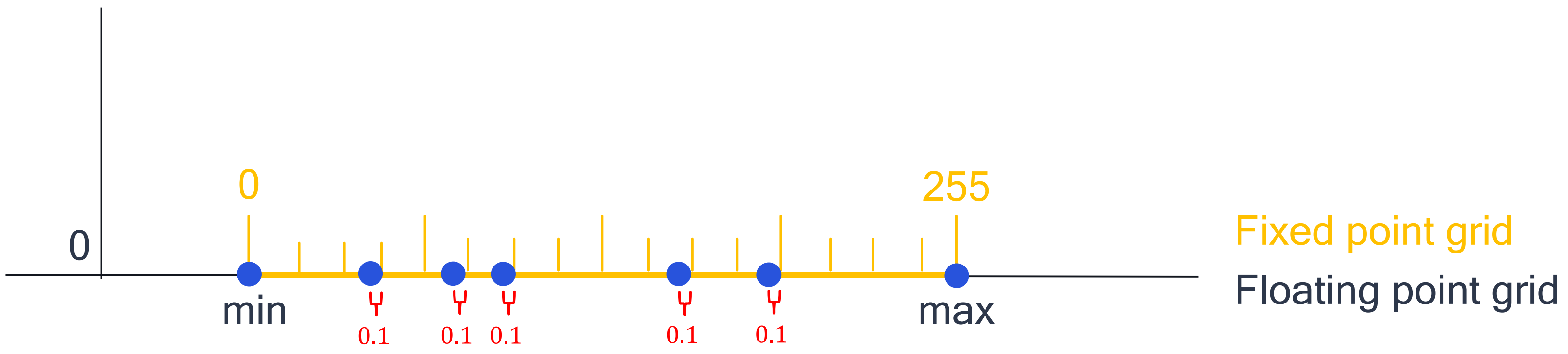




	Top-1 accuracy Float32	Top-1 accuracy INT8 (best)	Difference Top-1 accuracy
Asymmetric quant	71.9%	0.1%	**-71.8%**
Equalization	71.72%	69.91%	**-1.99%**

MobileNetV2 results for ImageNet

Biased quantization is a result of rounding errors



By sheer coincidence, it could be that

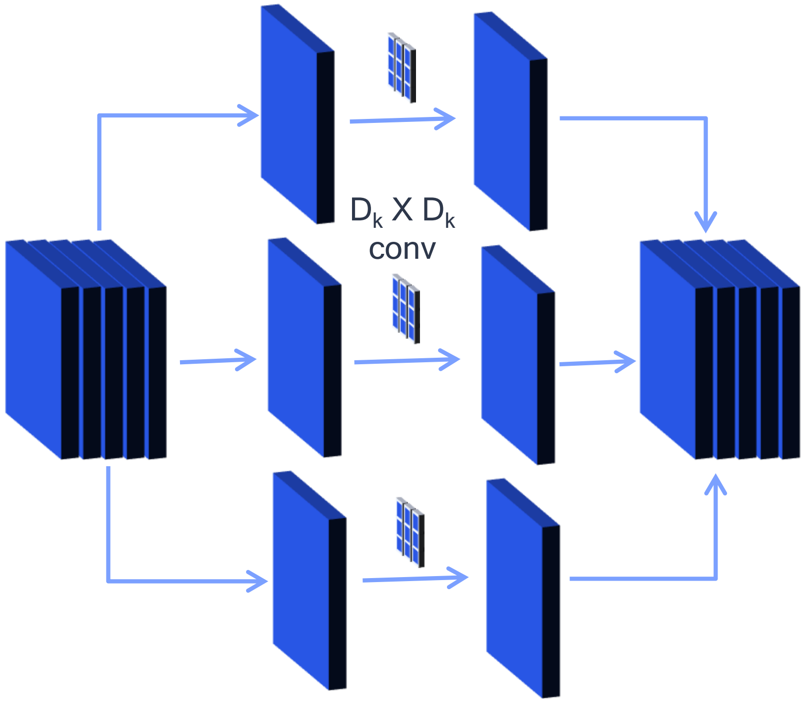
$$\boldsymbol{w} \cdot \boldsymbol{x} \approx \widetilde{\boldsymbol{w}} \cdot \boldsymbol{x} \approx \boldsymbol{w} \cdot \boldsymbol{x} + 0.1\boldsymbol{x}$$

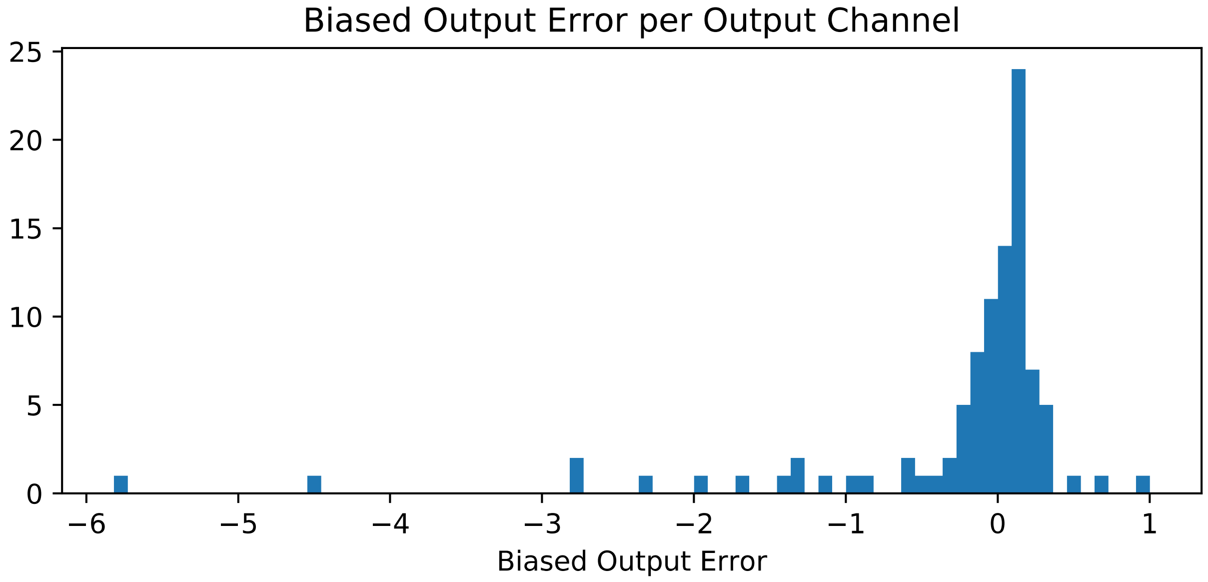
$\widetilde{w}$ = quantized w

Since most values are rounded up, the average output of the quantized model is now bigger than the original

i.e. $\mathbb{E}[\boldsymbol{y}] \neq \mathbb{E}[\widetilde{\boldsymbol{y}}]$

Biased errors are very detrimental to network performance





This bias is especially strong for networks with depth-wise separable convolutions!

Each output only has 3x3 associated parameters

MobileNet v2 layer 2 biased error histogram per output

Calculating bias correction to address biased quantization

Given $\boldsymbol{W}$, a weight matrix, and a quantized approximation $\widetilde{W}$, we can write in closed form:

$$\boldsymbol{W} = \widetilde{\boldsymbol{W}} + \boldsymbol{\epsilon}$$

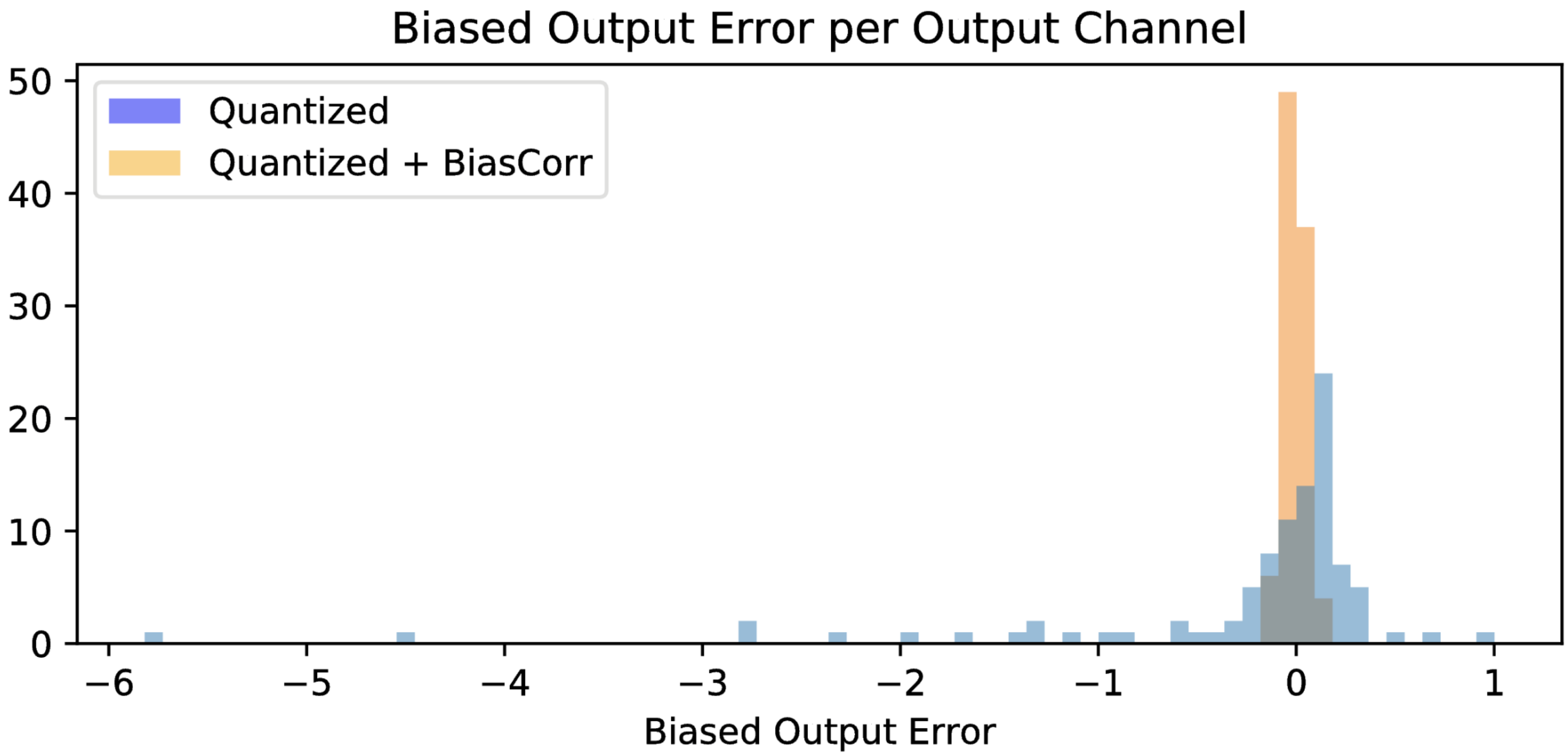
The bias of an output is given as:

$$\mathbb{E}[\boldsymbol{y}] - \mathbb{E}[\widetilde{\boldsymbol{y}}] =$$
$$\mathbb{E}[\boldsymbol{W}\boldsymbol{x}] - \mathbb{E}\left[\widetilde{\boldsymbol{W}}\boldsymbol{x}\right] =$$
$$\mathbf{W}\mathbb{E}[\boldsymbol{x}] - \widetilde{\boldsymbol{W}}\mathbb{E}[\boldsymbol{x}] =$$
$$\boldsymbol{\epsilon}\mathbb{E}[\boldsymbol{x}]$$

Key idea: Bias correction

We find $\boldsymbol{\epsilon}\mathbb{E}[\boldsymbol{x}]$ and subtract it from the output after quantization to correct for the bias effect!

Bias correction removes the biased output error



MobileNetv2 2nd layer

DFQ offers state-of-the-art results

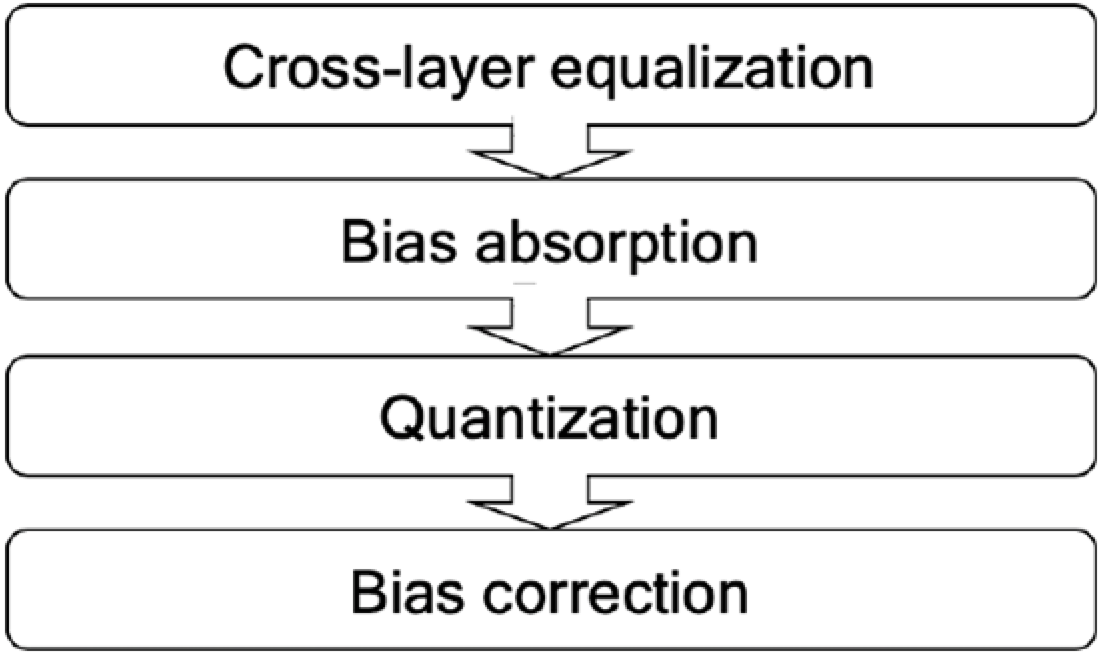
		~D	~BP	~AC	MobileNetV2 FP32	MobileNetV2 INT8	MobileNetV1 FP32	MobileNetV1 INT8	ResNet18 FP32	ResNet18 INT8	ResNet18 INT6
Data-free	DFQ (ours)	✓	✓	✓	71.7%	**71.2%**	70.8%	**70.5%**	69.7%	**69.7%**	66.3%
	Per-layer [18]	✓	✓	✓	71.9%	0.1%	70.9%	0.1%	69.7%	69.2%*	63.8%*
	Per-channel [18]	✓	✓	✓	71.9%	69.7%	70.9%	70.3%	69.7%	69.6%*	**67.5%***
Requires training	QT [16] ^	✗	✗	✓	71.9%	70.9%	70.9%	70.0%	-	**70.3%**†	67.3%†
	SR+DR†	✗	✗	✓	-	-	-	**71.3%**	-	68.2%	59.3%
	QMN [31]	✗	✗	✗	-	-	70.8%	68.0%	-	-	-
	RQ [21]	✗	✗	✗	-	-	-	70.4%	-	69.9%	**68.6%**

ImageNet top 1 accuracy

Per-channel is per channel quantization (Krishnamoorthi 2018)
QT is quantization aware training (Jacob et al. CVPR 2018)
SR+DR is stochastic rounding + dynamic ranges (results from Louizos et al. ICLR 2019)
QMN is Quantization friendly MobileNets (Sheng et al. EMC^2 2018)
RQ is Relaxed quantization (Louizos et al. ICLR 2019)

~D: no data needed
~BP: no backprop needed
~AC: no architecture changes needed

Data-free quantization recap



Flow diagram of the data-free quantization method

- A simple API call results in better quantization performance
- A fully-automatic pipeline gives near-original model performance without fine-tuning
- No (P)ReLU activations? Use smart clipping and bias correction

AdaRound

AdaRound

Adaptive rounding for neural network quantization

Markus Nagel
Qualcomm Technologies Netherlands B.V.

Rana Amjad
Qualcomm Technologies Netherlands B.V.

Mart van Baalen
Qualcomm Technologies Netherlands B.V.

Christos Louizos
Qualcomm Technologies Netherlands B.V.

Tijmen Blankevoort
Qualcomm Technologies Netherlands B.V.

- Introduces a new way of rounding
- Achieves excellent 4-bit weight results

Nagel et al. 2020 Up or Down? Adaptive Rounding for Post-Training Quantization

Rounding can be done better for quantization

Introducing AdaRound to optimize for rounding

Normally, we round our weights to the nearest grid-point

$$\widehat{\mathbf{w}} \in \left\{\mathbf{w}^{floor}, \mathbf{w}^{ceil}\right\}$$

$$\widehat{\mathbf{w}} = \mathbf{w} - \Delta\mathbf{w} = s \cdot clip\left(\left\lfloor \frac{\mathbf{w}}{s} \right\rceil, \mathrm{n}, \mathrm{p}\right)$$

Rounding-to-the-nearest is not optimal

Consider the accuracy of various rounding schemes for 4-bit quantization of Resnet18 first layer

Rounding scheme	Acc(%)
Nearest	52.29
Ceil	0.10
Floor	0.10
Stochastic	52.06± 5.52
Stochastic (best)	63.06

Rounding all values up or down gives 0 performance. Drawing 100 samples and picking the best one increases performance by 10%

By optimizing the layer-wise objective, AdaRound optimizes the network weights in minutes without fine-tuning or hyperparameters

$$\underset{\mathbf{V}}{\arg\min} \left\| \mathbf{W}\mathbf{x} - \widetilde{\mathbf{W}}\mathbf{x} \right\|_F^2 + \lambda f_{reg}(\mathbf{V})$$

AdaRound makes 4-bit weights possible

Without any fine-tuning or hyperparameter tweaking

Optimization	#bits W/A	Resnet18	Resnet50	InceptionV3	MobilenetV2
Full precision	32/32	69.68	76.07	77.40	71.72
DFQ (Nagel et al., 2019)	8/8	69.7	-	-	71.2
Nearest	4/32	23.99	35.60	1.67	8.09
OMSE+opt(Choukroun et al., 2019)	4*/32	67.12	74.67	73.66	-
OCS (Zhao et al., 2019)	4/32	-	66.2	4.8	-
AdaRound	4/32	**68.71±0.06**	**75.23±0.04**	**75.76±0.09**	**69.78±0.05**†
DFQ (our impl.)	4/8	38.98	52.84	-	46.57
Bias corr (Banner et al., 2019)	4*/8	67.4	74.8	59.5	-
AdaRound w/ act quant	4/8	**68.55±0.01**	**75.01±0.05**	**75.72±0.09**	**69.25±0.06**†

Table 7. Comparison among different post-training quantization strategies in the literature. We report results for various models in terms of ImageNet validation accuracy (%). *Uses per channel quantization. †Using CLE (Nagel et al., 2019) as preprocessing.

For several models, we can now have 4-bit weights while only dropping 1-2% accuracy

Compared to networks quantized to 8-bit weight and 8-bit activation, 4-bit weight and 8-bit activation speeds up execution by 2x and reduces energy consumption by 2x – with virtually no additional work

Bayesian Bits

Bayesian Bits

Unifying quantization and pruning

Mart van Baalen
Qualcomm Technologies Netherlands B.V.

Christos Louizos
Qualcomm Technologies Netherlands B.V.

Rana Amjad
Qualcomm Technologies Netherlands B.V.

Markus Nagel
Qualcomm Technologies Netherlands B.V.

Tijmen Blankevoort
Qualcomm Technologies Netherlands B.V.

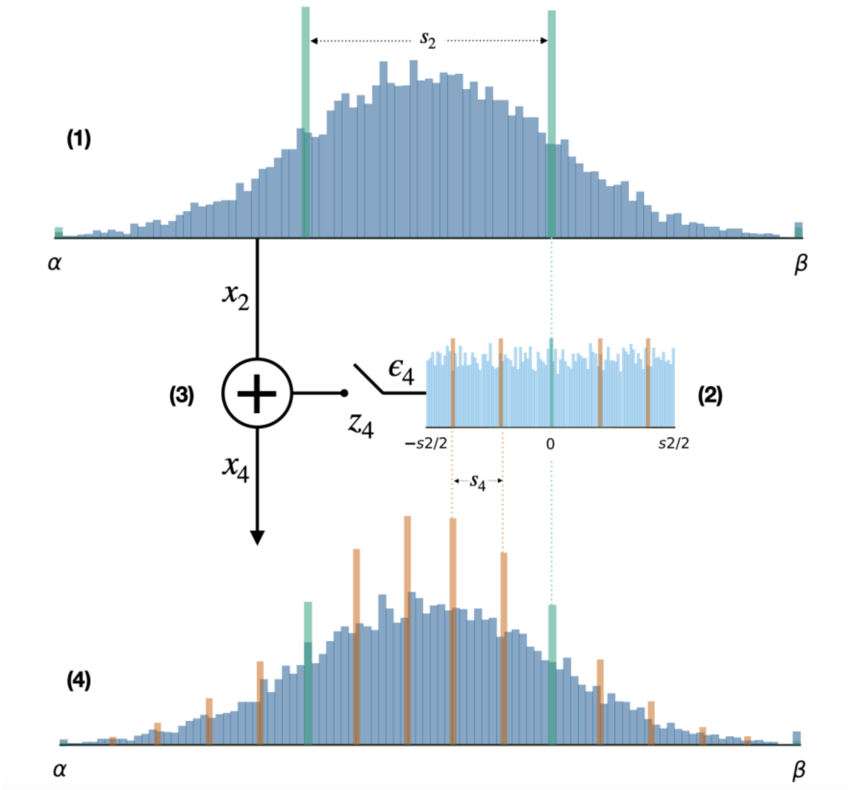
Max Welling
Qualcomm Technologies Netherlands B.V.

Introduces one training scheme to automatically find mixed-precision quantization networks and pruning

Baalen et al. 2020 Bayesian Bits: Unifying Quantization and Pruning

Bayesian Bits

Automatically learning quantization bit-width and pruning during training



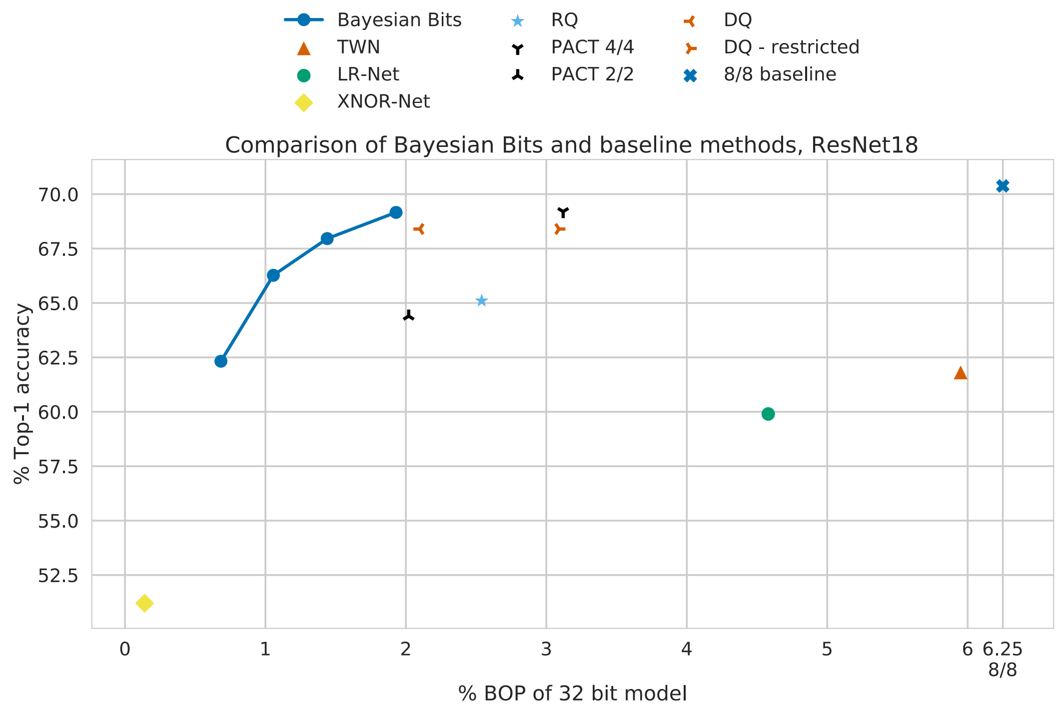
We decompose quantization grids into multiple smaller grids

$$x_q = x_2 + z_4\left(\epsilon_4 + z_8\left(\epsilon_8 + z_{16}(\epsilon_{16} + z_{32}\epsilon_{32})\right)\right)$$

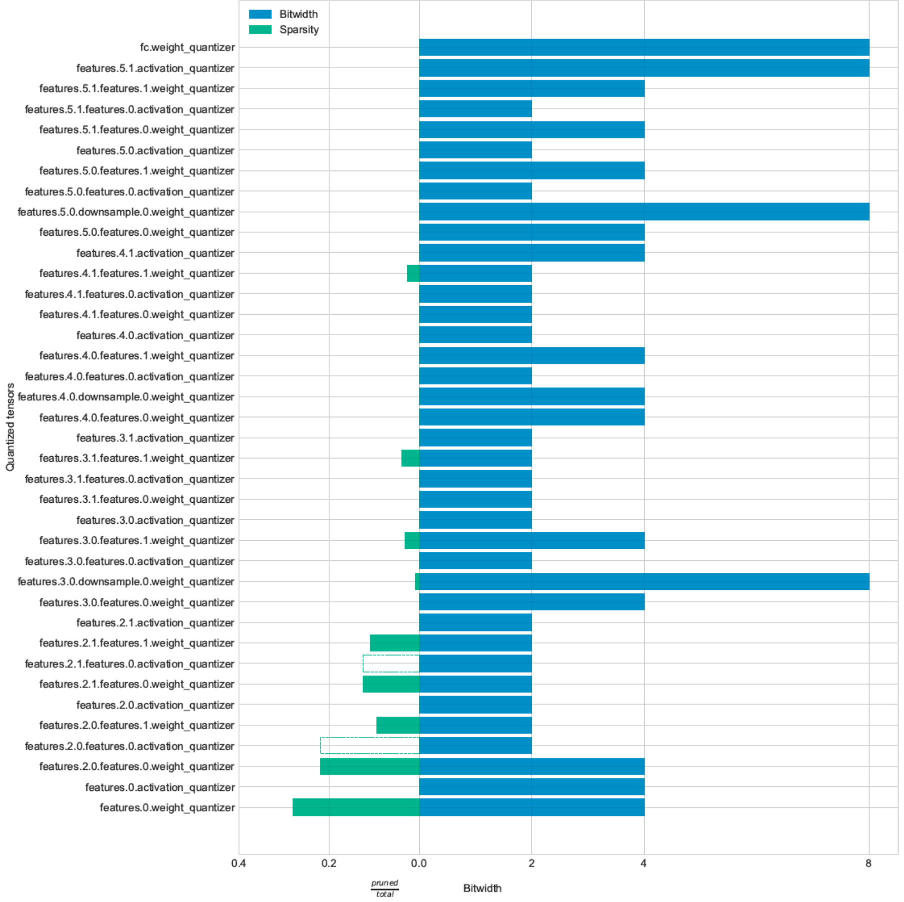
This allows us to introduce gating variables z that toggle higher-bit-width quantization on/off

State-of-the-art performance for mixed-precision quantization

Systematically selecting the appropriate amount of precision



During training, the network automatically finds the optimal trade-off between network complexity and accuracy



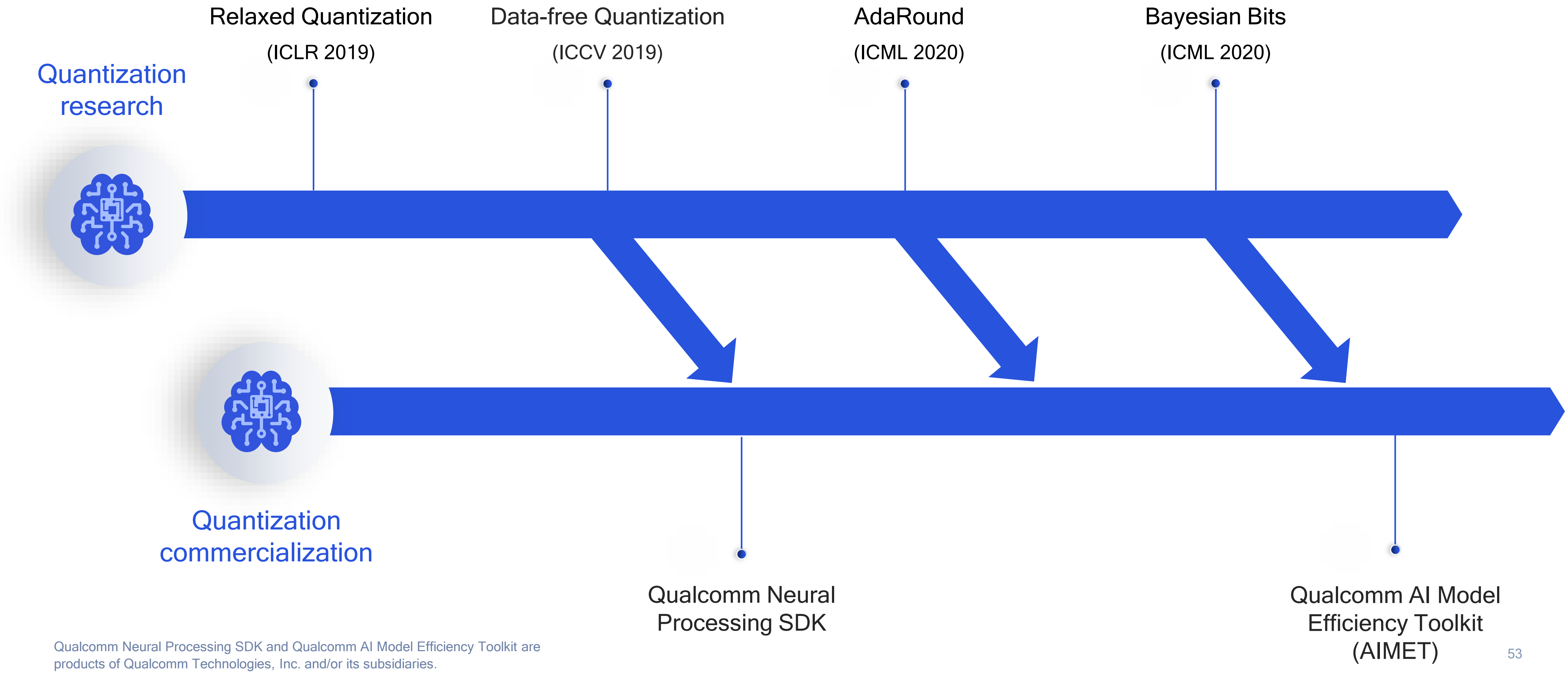
The result: Some layers are fine with 8 bits, while others are fine with 2 bits. And some layers are pruned (green)

Develop

How developers and the research community can take advantage of our quantization tools

Leading quantization research and fast commercialization

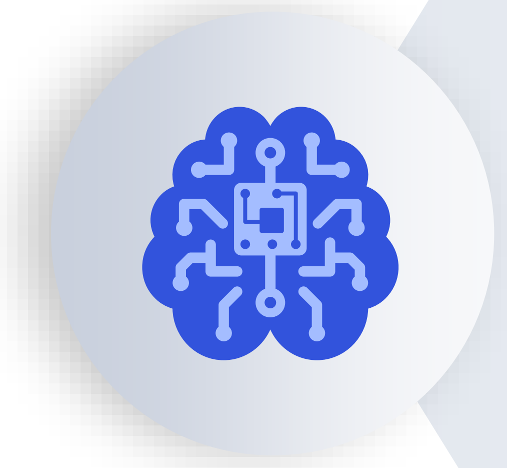
Driving the industry towards integer inference and power-efficient AI



Qualcomm Neural Processing SDK and Qualcomm AI Model Efficiency Toolkit are products of Qualcomm Technologies, Inc. and/or its subsidiaries.

Qualcomm® Neural Processing SDK

Software accelerated runtime for the execution of deep neural networks on device



Qualcomm snapdragon

CPU — Qualcomm® Kryo™ CPU

GPU — Qualcomm® Adreno™ GPU

DSP — Qualcomm® Hexagon™ DSP

Efficient execution on Qualcomm® Snapdragon™ Mobile Platform

- Takes advantage of Snapdragon heterogeneous computing capabilities
- Runtime and libraries accelerate deep neural net processing on all engines: CPU, GPU, and DSP with Hexagon Vector eXtensions (HVX) and Hexagon Tensor Accelerator (HTA)

ONNX · TensorFlow · Caffe · PyTorch · Caffe2

Model framework/Network support

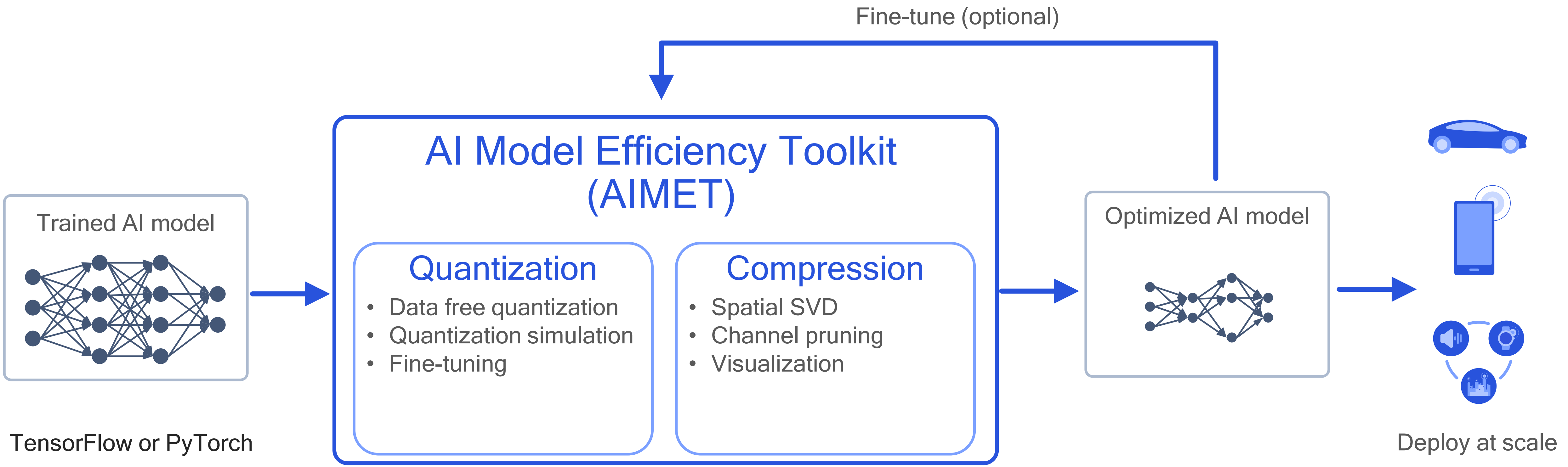
- Convolutional neural networks and Long short Term Memory (LSTM) Networks
- Support for Caffe/Caffe2, TensorFlow, and user/developer defined layers

Offline conversion tools · Analyze performance · Sample code · Ease of integration

Optimization/Debugging tools

- Offline network conversion tools
- Debug and analyze network performance
- API and SDK documentation with sample code
- Ease of integration into customer applications

Available at: developer.qualcomm.com



AIMET plugs in seamlessly to the developer workflow

AIMET makes AI models small

Includes state-of-the-art quantization and compression techniques from Qualcomm AI Research

Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

Features

- State-of-the-art network compression tools
- State-of-the-art quantization tools
- Support for both TensorFlow and Pytorch
- Benchmarks and tests for many models
- Developed by professional software developers

If interested, email: aimet.support@qti.qualcomm.com

Coming soon…

AIMET open source

Quantization improves power efficiency, performance, and memory usage

Our research addresses the limitations of traditional quantization approaches

Our user-friendly tools allow developers to develop power-efficient AI apps on Qualcomm Snapdragon

Thank you

Follow us on:

For more information, visit us at:

www.qualcomm.com & www.qualcomm.com/blog