中国科学院自动化研究所
INSTITUTE OF AUTOMATION CHINESE ACADEMY OF SCIENCES

www.ia.ac.cn

S9713:

Quantized Neural Networks and QEngine

Yifan Zhang

yfzhang@nlpr.ia.ac.cn

Institute of Automation, Chinese Academy of Sciences

2019.3.18

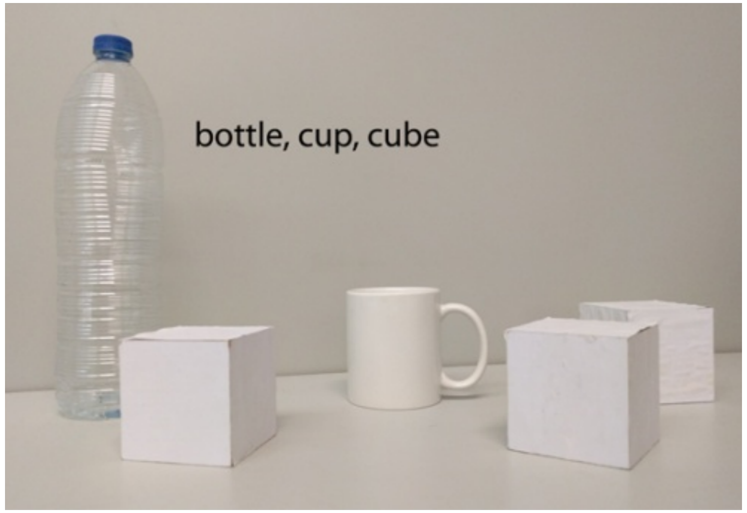
Outline

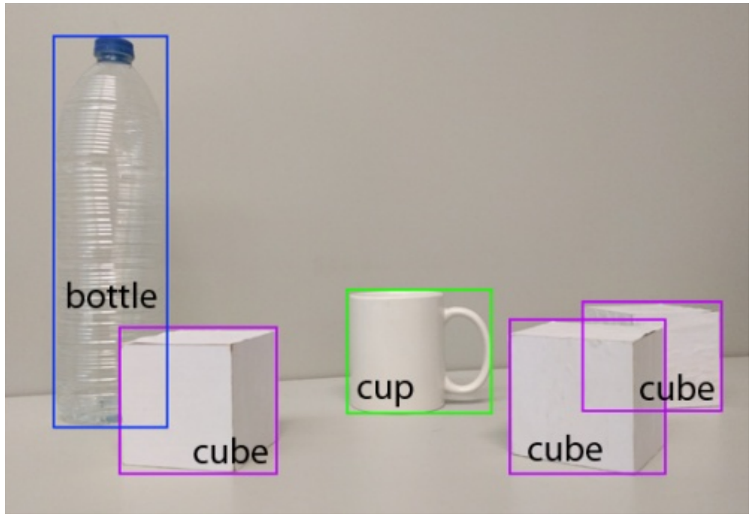
- Background
- Preliminary
- Network Quantization
- QEngine
- Summary

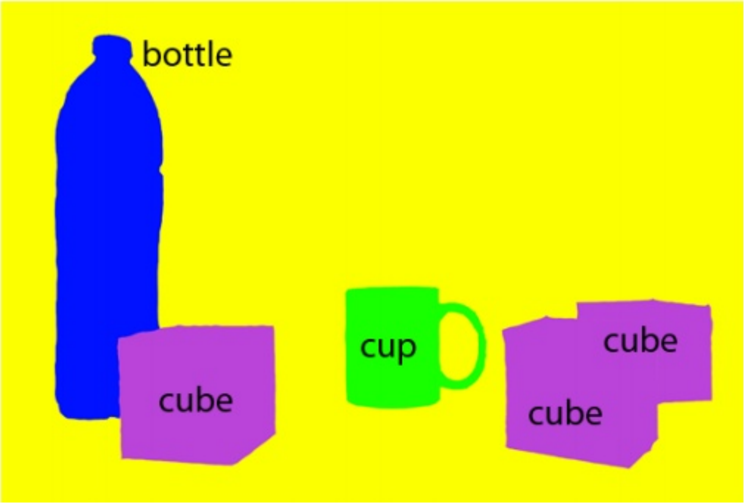
Outline


AI

Background: Applications of CNNs







Classification

Detection

Segmentation

Convolutional Neural Networks

Background: Real-World Applications

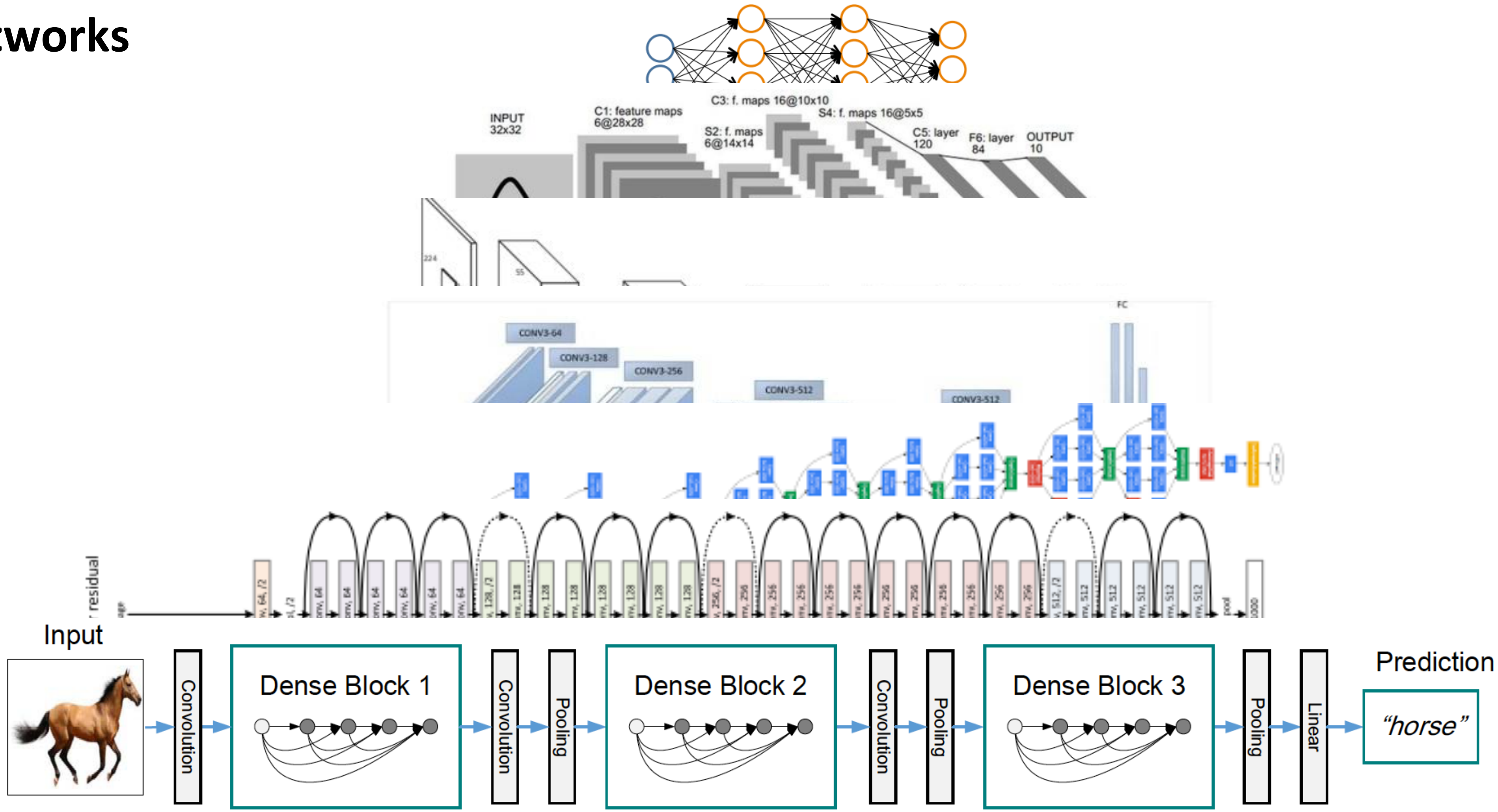
Self-Driving Car

Intelligent Robot

Face Unlock

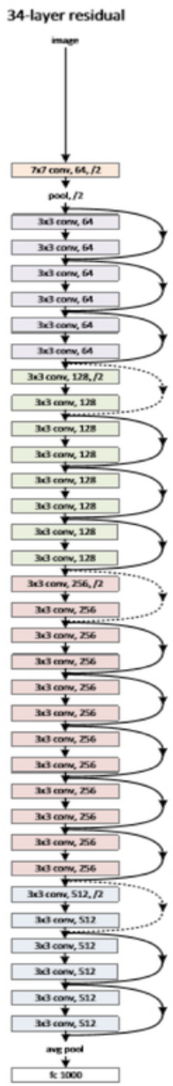
Revolution of Deep Neural Networks

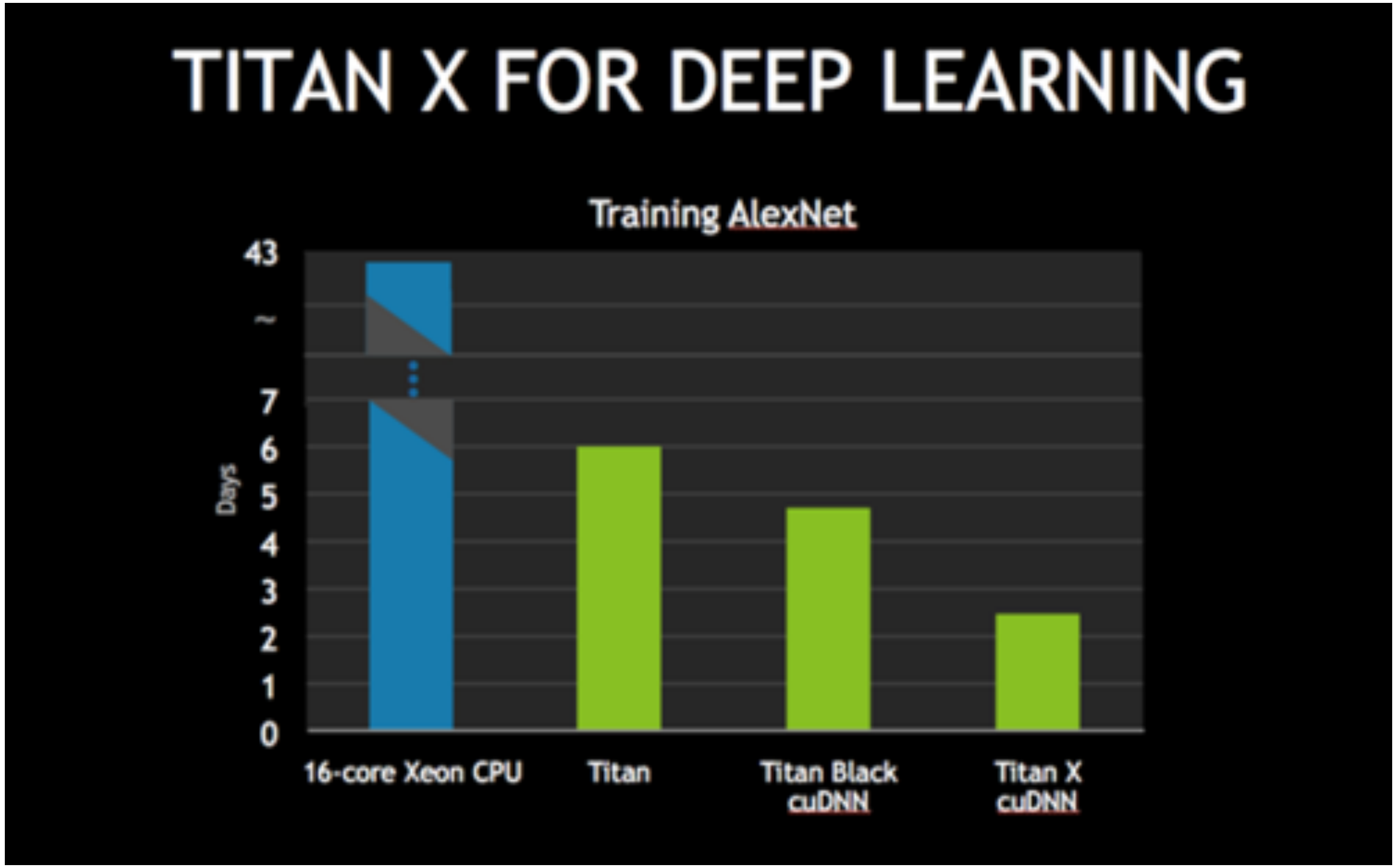
- ❑ Shallow Networks
- ❑ LeNet
- ❑ AlexNet
- ❑ VGG
- ❑ GoogLeNet
- ❑ ResNet
- ❑ DenseNet



Year	Model	Layers	Parameter	FLOPs	ImageNet Top-5 error
2012	AlexNet	5+3	60M	725M	16.4%
2013	Clarifai	5+3	60M	1.17B	11.7%
2014	VGG-19	16+3	143M	19.6B	7.32%
2014	GoogLeNet	22	6.8M	1.566B	6.67%
2015	Inception-V3	42	23.8M	5.7B	3.6%
2015	ResNet	152	19.4M	11.3B	3.57%
2016	Inception-V4	112	42.6M	12.25B	3.08%

Training : Big Data + Large Model + Powerful Resources

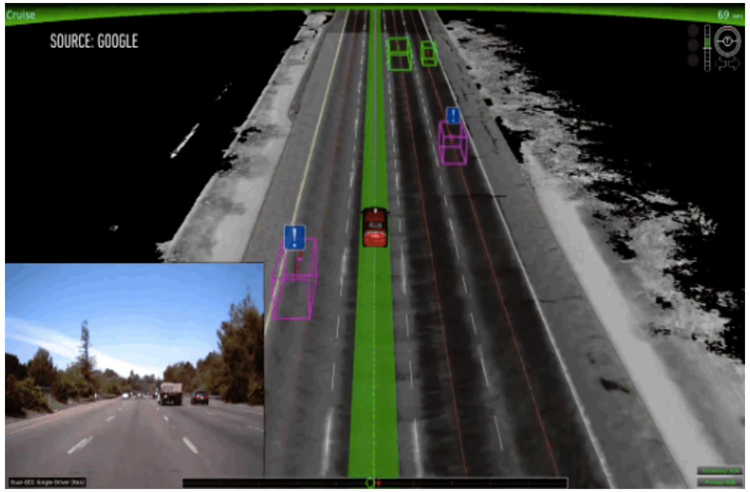




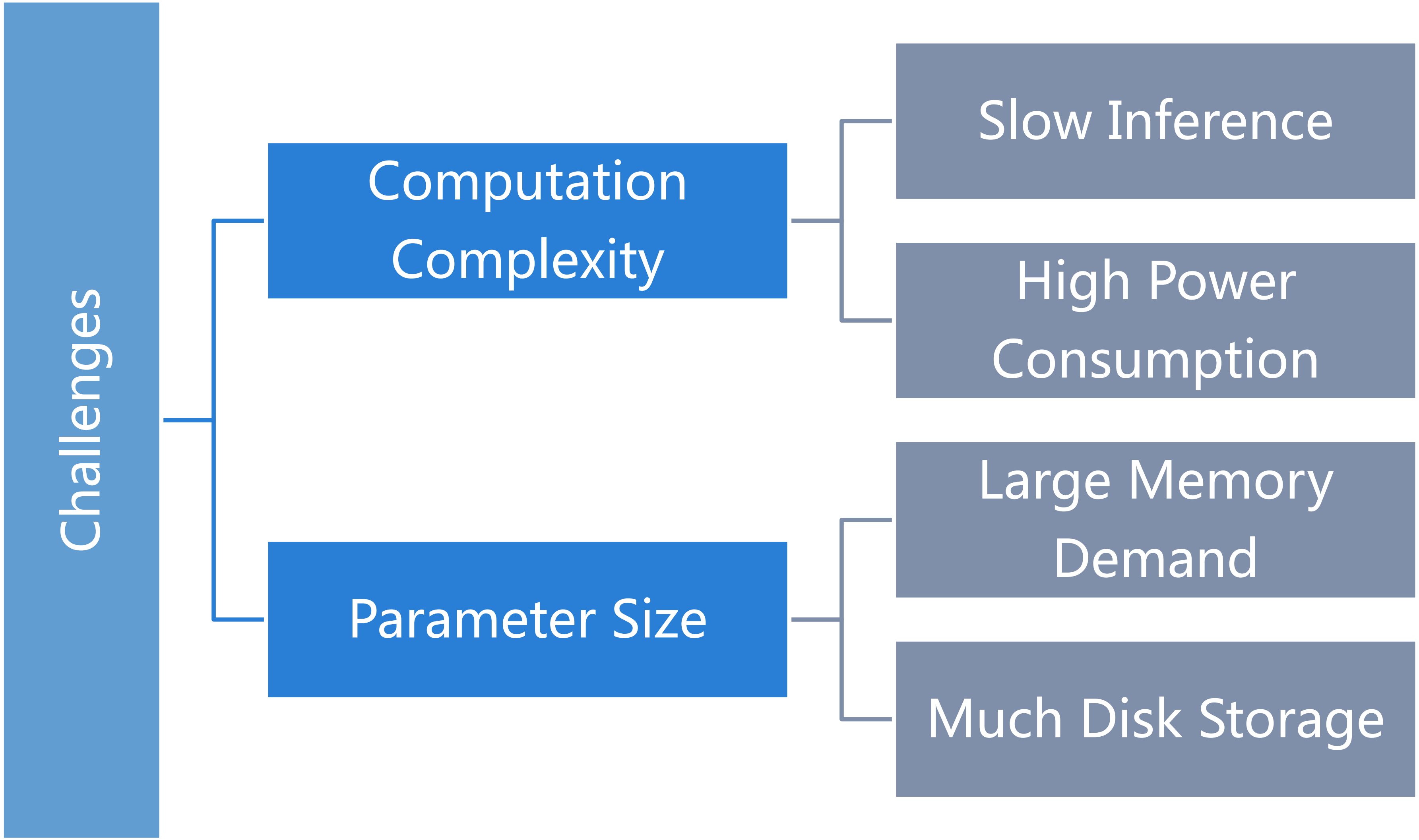
Train ResNet50 from many days to:

- Facebook: 1 hour
- Fast.ai: 18 min
- Tencent: 6.6 min
- Sony: 3.7 min
- Google: 2.2 min
- SenseTime: 1.5 min

Inference : Big Data + Large Model + Limited Resources



We need Network Compression and Acceleration



Outline

Background

Preliminary

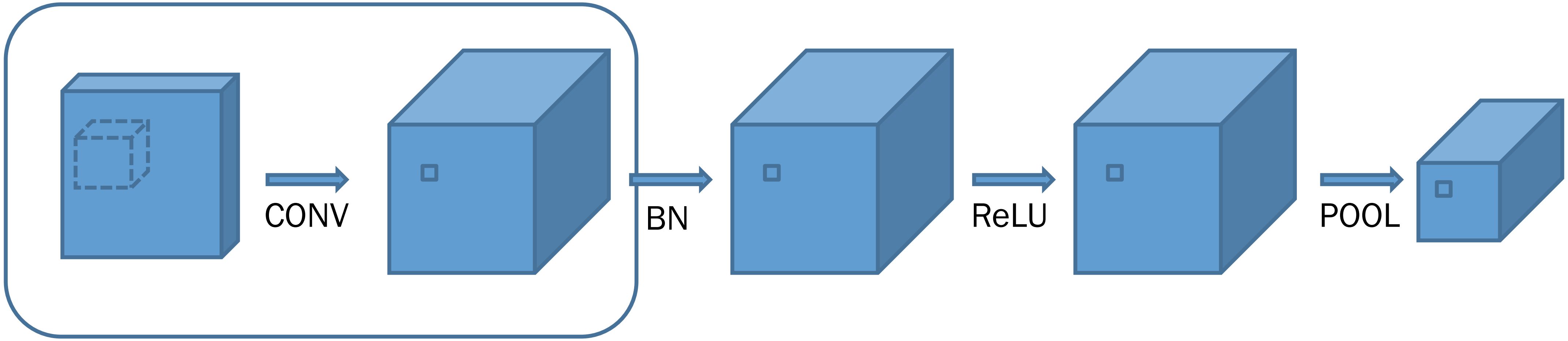
Network Quantization

QEngine

Summary

Preliminary

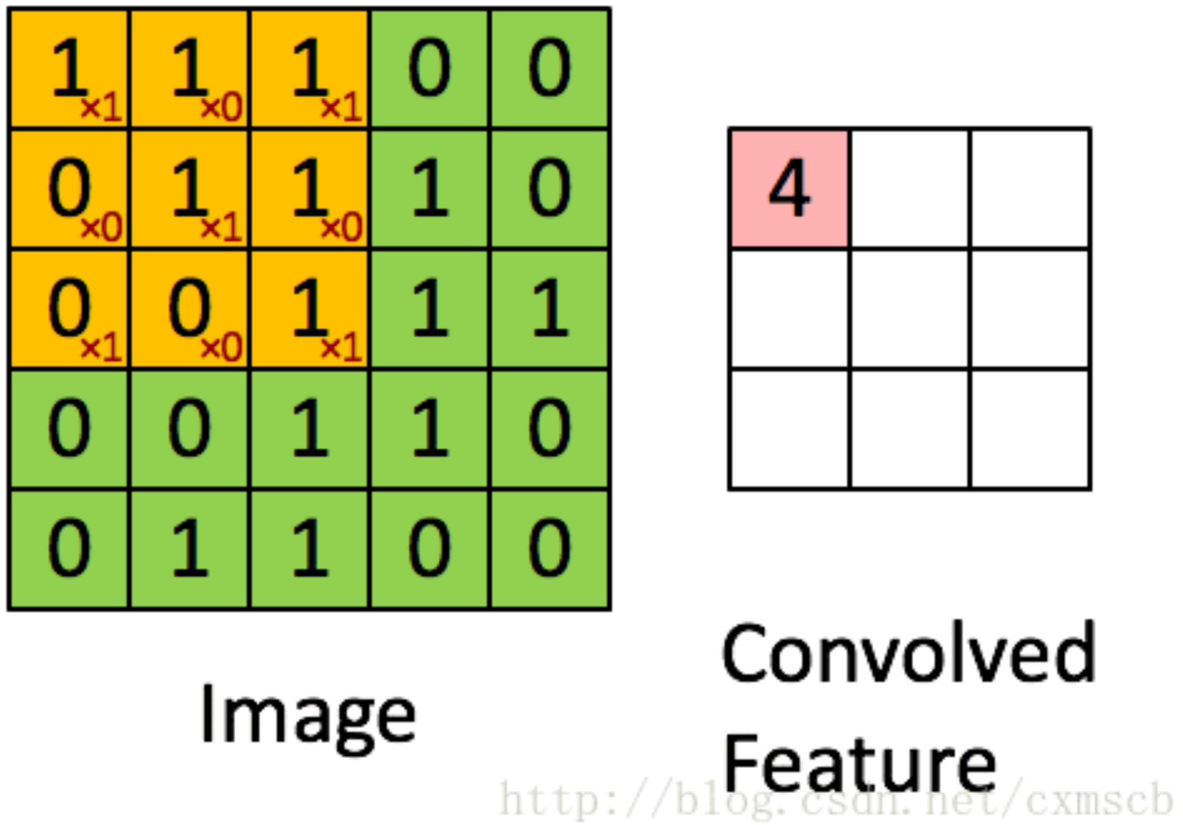
- Basic building block of CNN



Most of computation and parameters

Preliminary

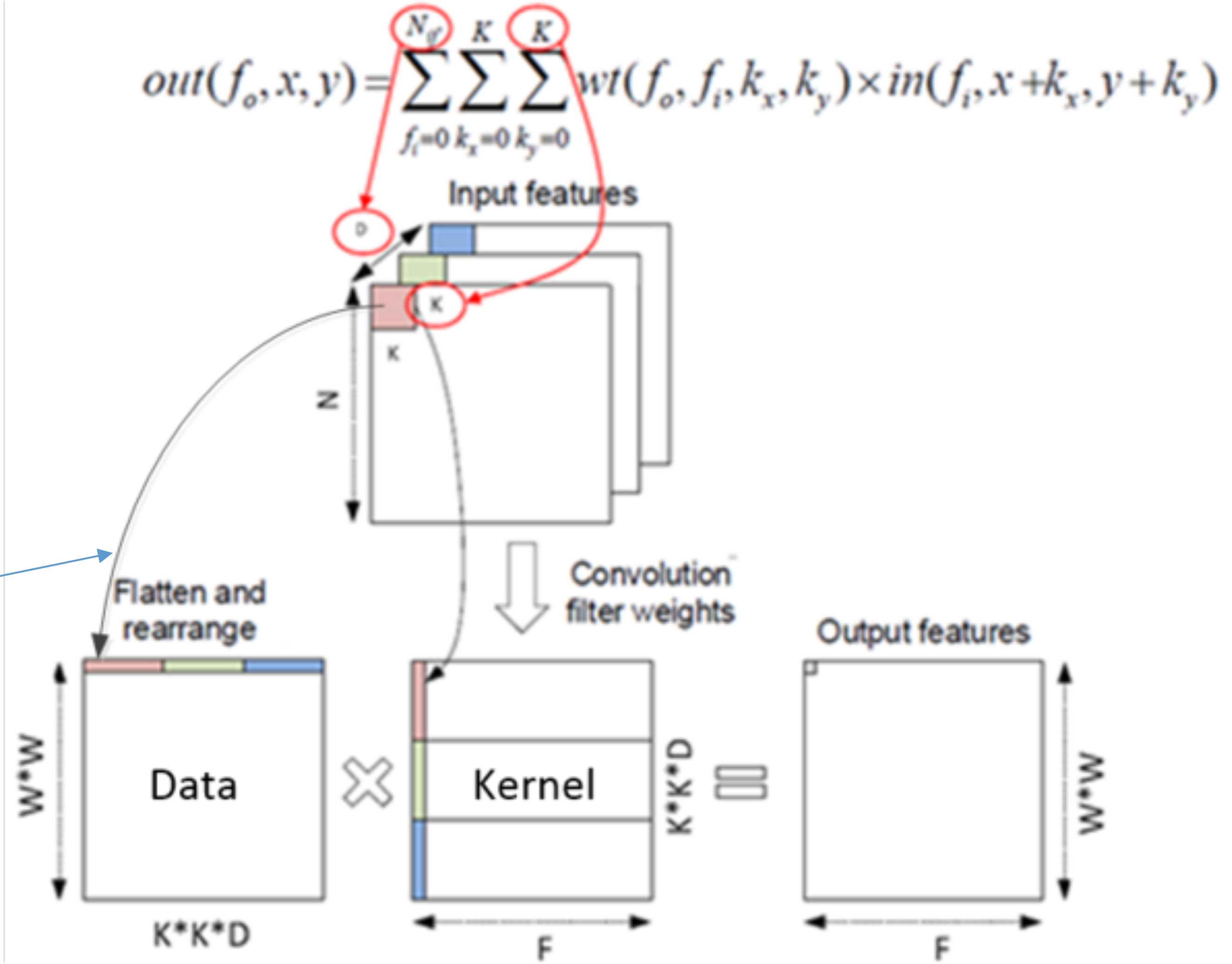
- 2D convolution



Preliminary

- 3D convolution

Im2col operation of Caffe

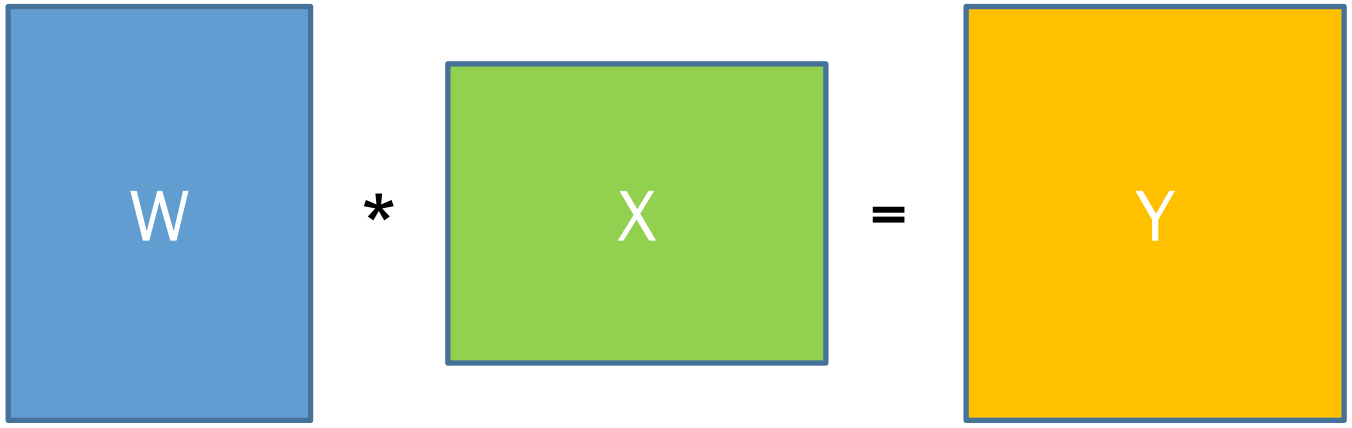


Outline

Background
Preliminary
Network Quantization
QEngine
Summary

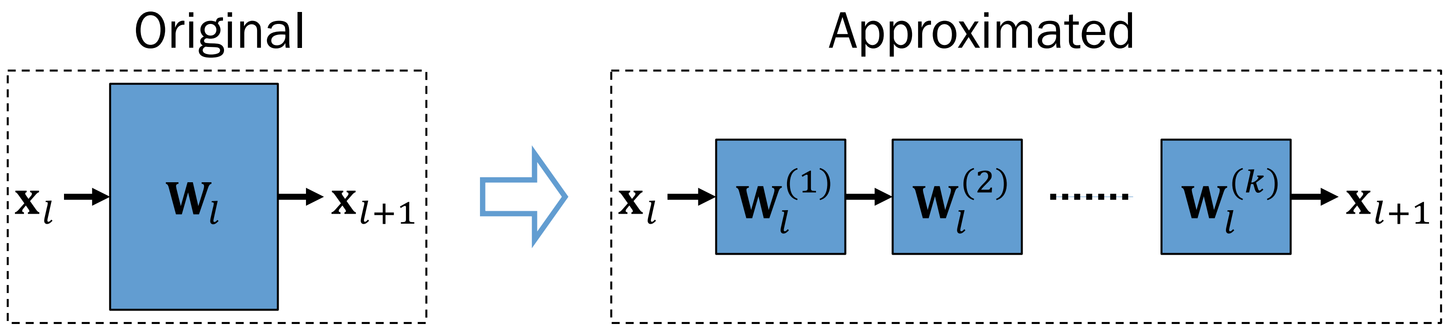
Network Compression and Acceleration

- Low-rank Decomposition
- Sparse/Pruning
- Quantization
- ……



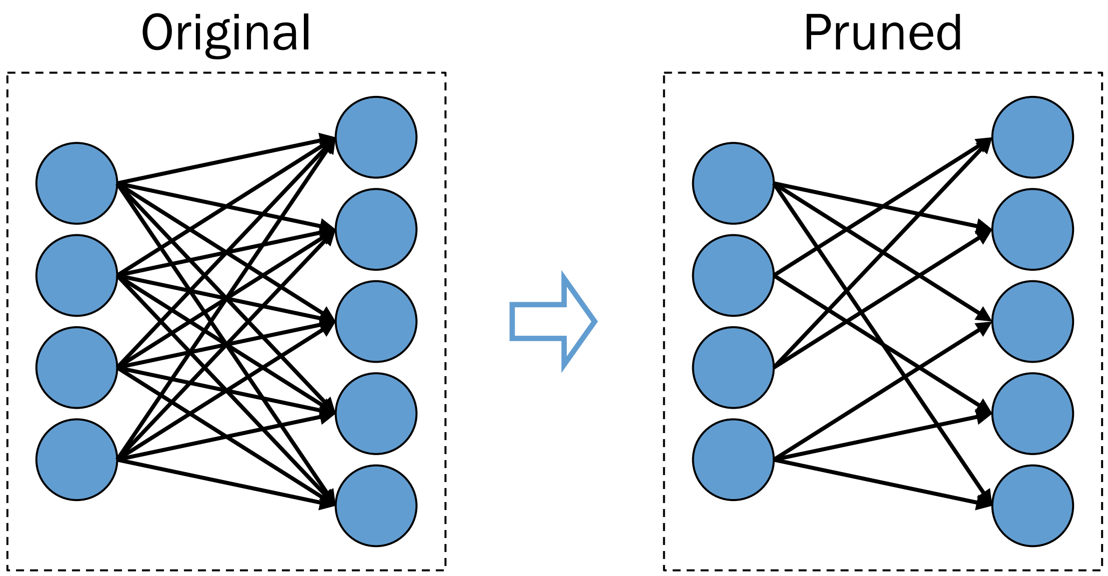
Low-Rank

- Single conv./fc. layer -> multiple layers
  - Faster computation
  - Fewer connection weights



Pruning

- Remove network connections
  - – Fewer FLOPs (may not be faster)
  - – Fewer connection weights

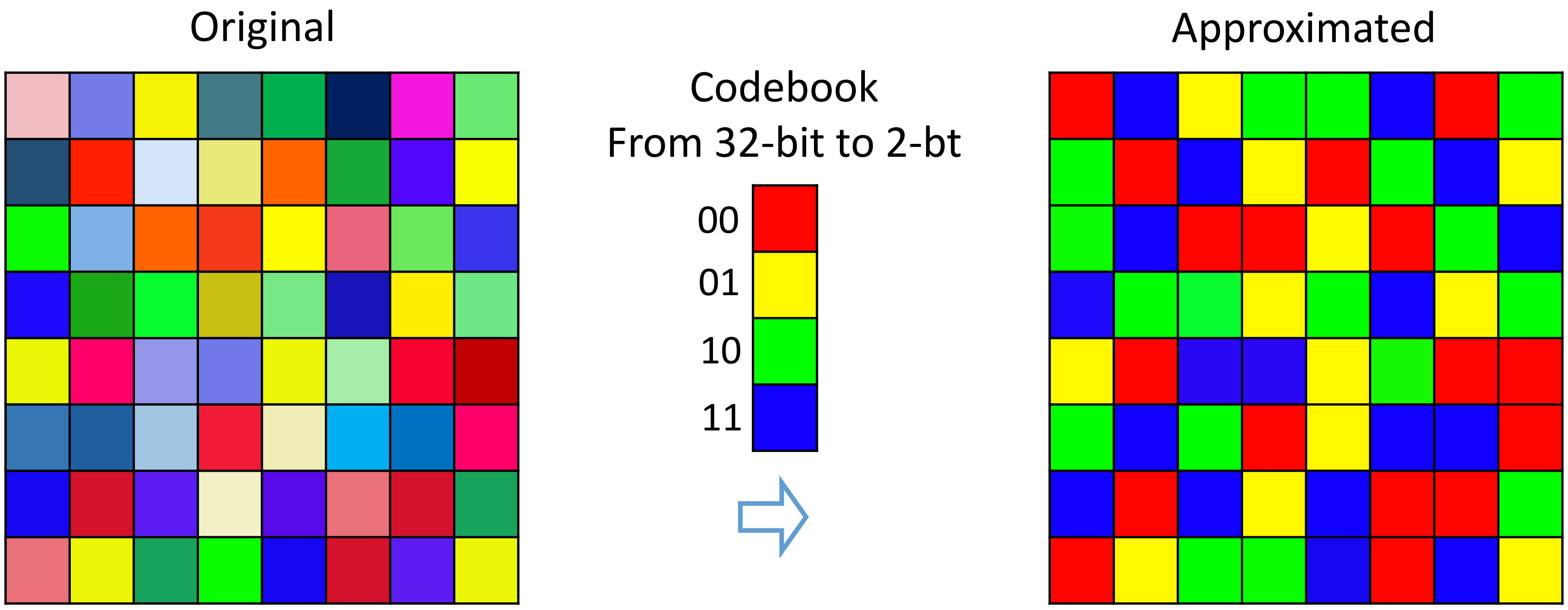


Network Compression and Acceleration

- Quantization
  - Code-based Quantization
    - Scalar Quantization
    - Vector Quantization
    - Product Quantization
  - Fixed point Quantization

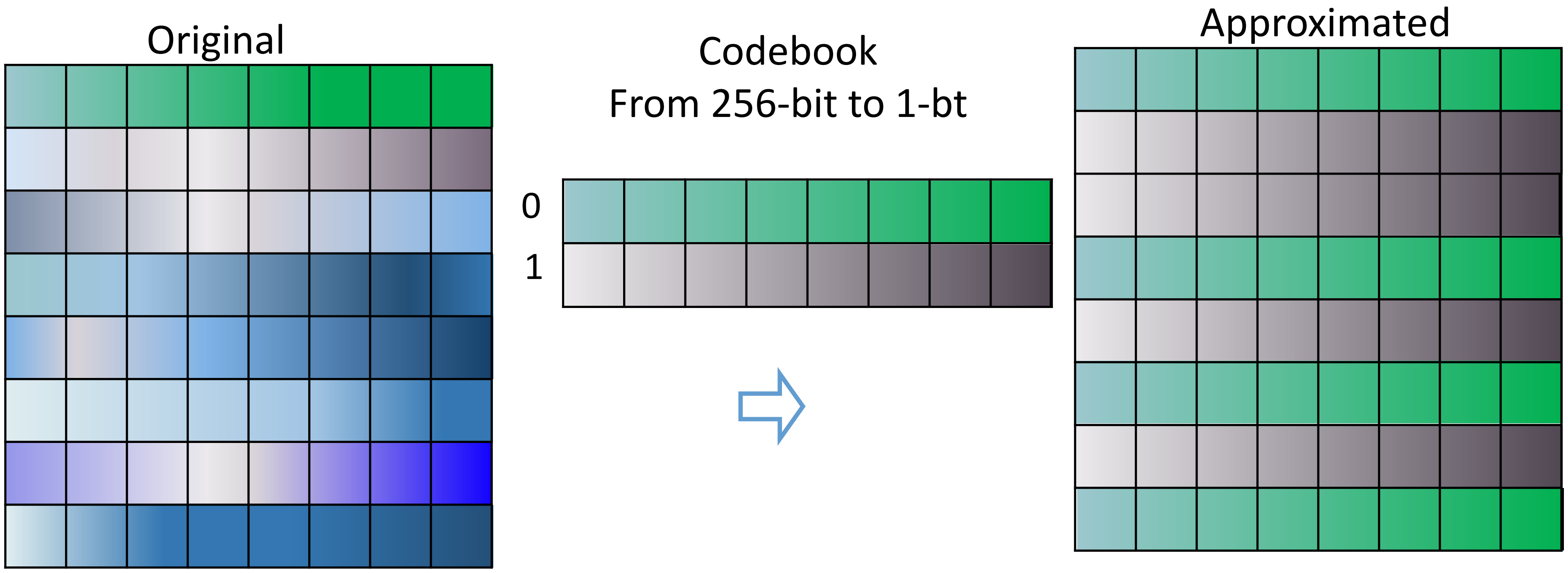
Code-based Quantization

- Scalar quantization
  - Element level



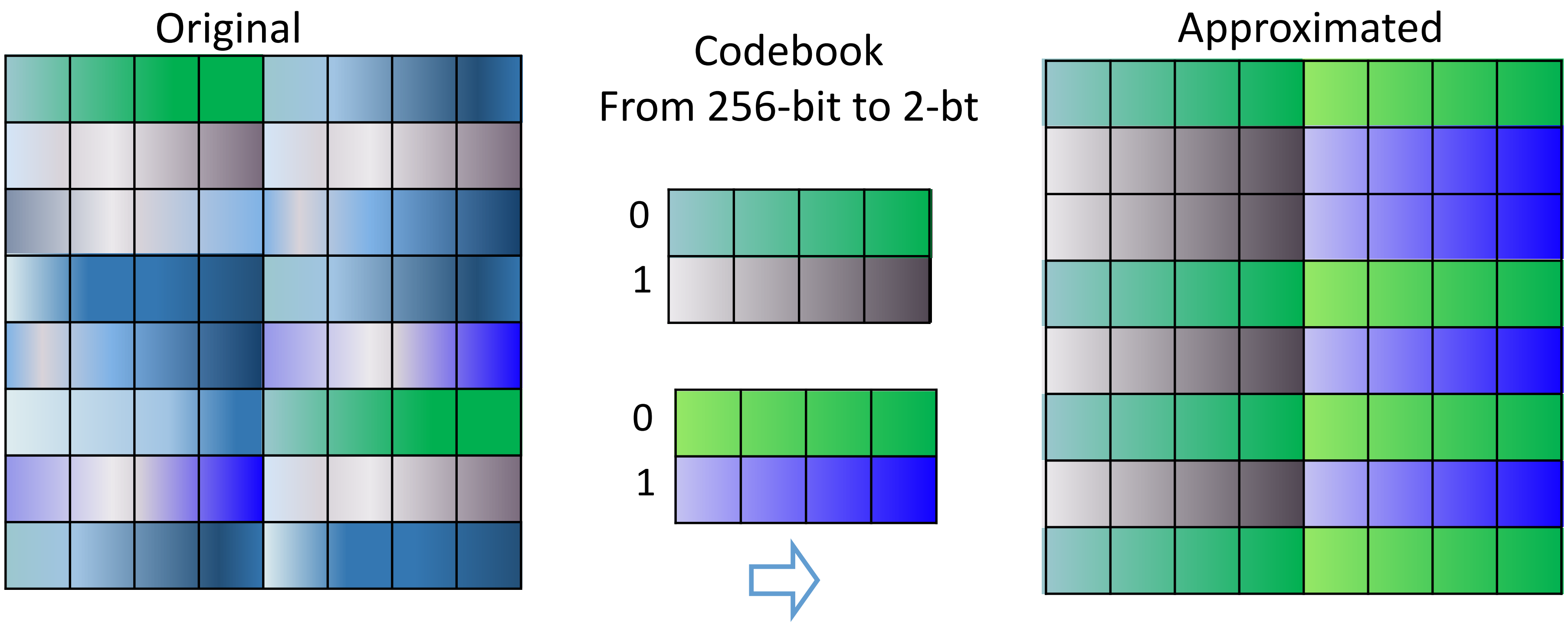
Code-based Quantization

- Vector quantization
  - Row/column level



Code-based Quantization

- Product quantization
  - Sub-Row/Sub-column level

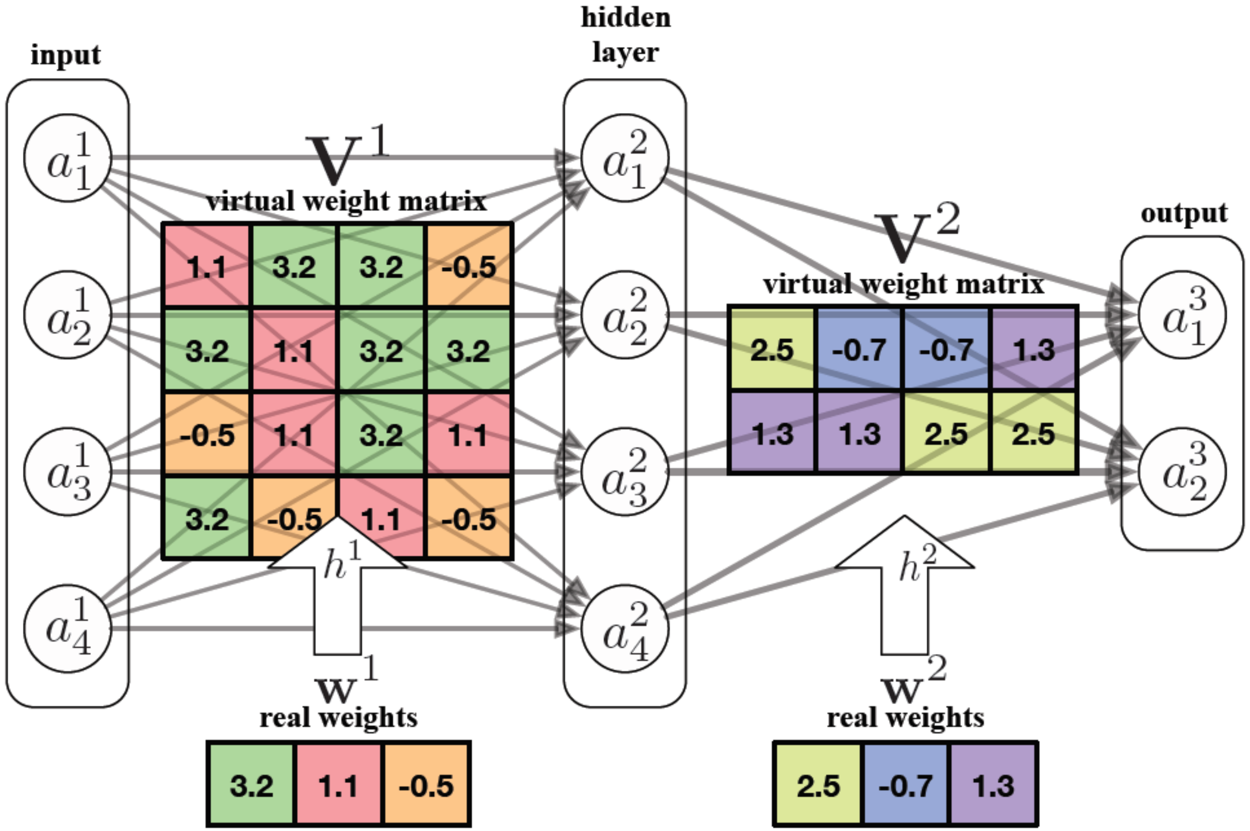


Hashed-Net

- Determine the correspondence via hashing

Random Weight sharing

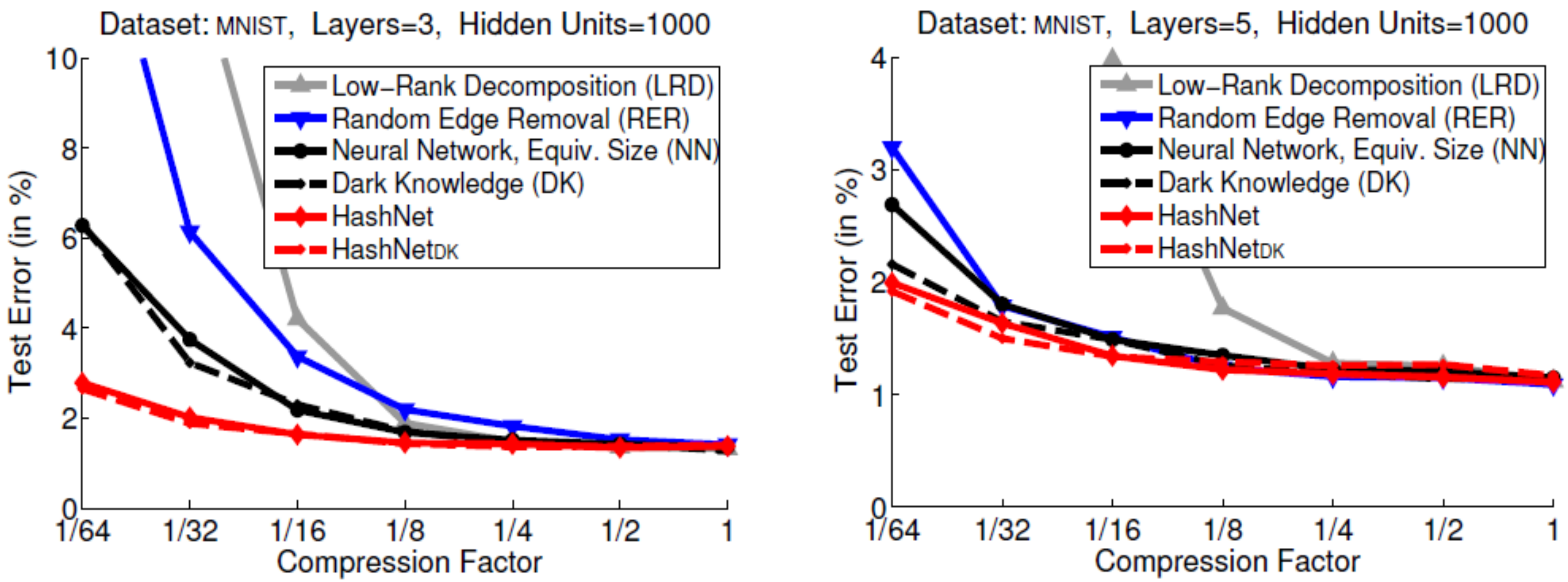
Hashing



W. Chen et al., “Compressing Neural Networks with the Hashing Trick”. WUSTL&NVIDIA, ICML 2015.

Hashed-Net

- Test error vs. Compression factor



W. Chen et al., “Compressing Neural Networks with the Hashing Trick”. WUSTL&NVIDIA , ICML 2015.

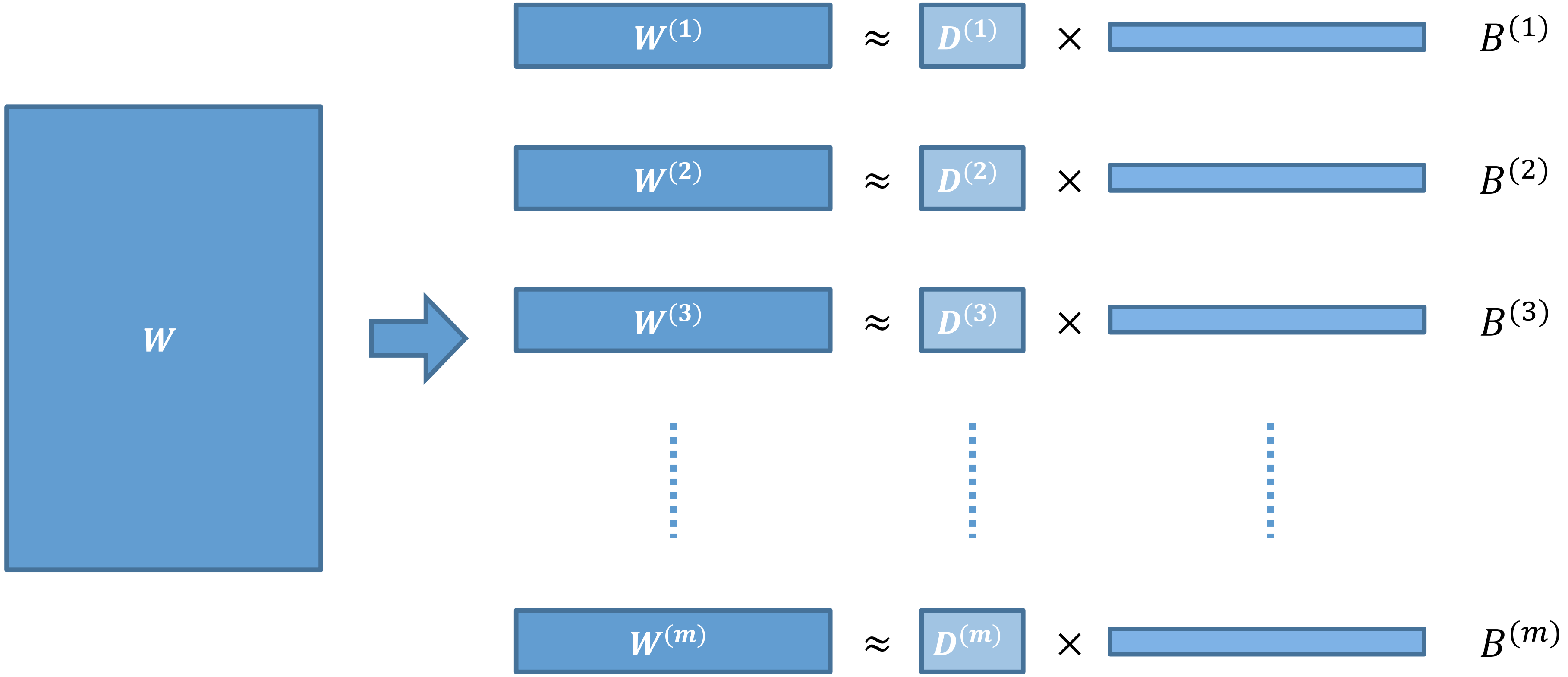
PQ for FC. layer

- Apply product quantization in fc. Layer
  - Split weighting matrix into sub-matrices
  - K-means clustering on each sub-matrix
  - Re-construct weighting matrix with clustering results

Y. Gong et al., “Compressing Deep Convolutional Networks using Vector Quantization”. Facebook, arXiv 2014.

PQ for FC. layer

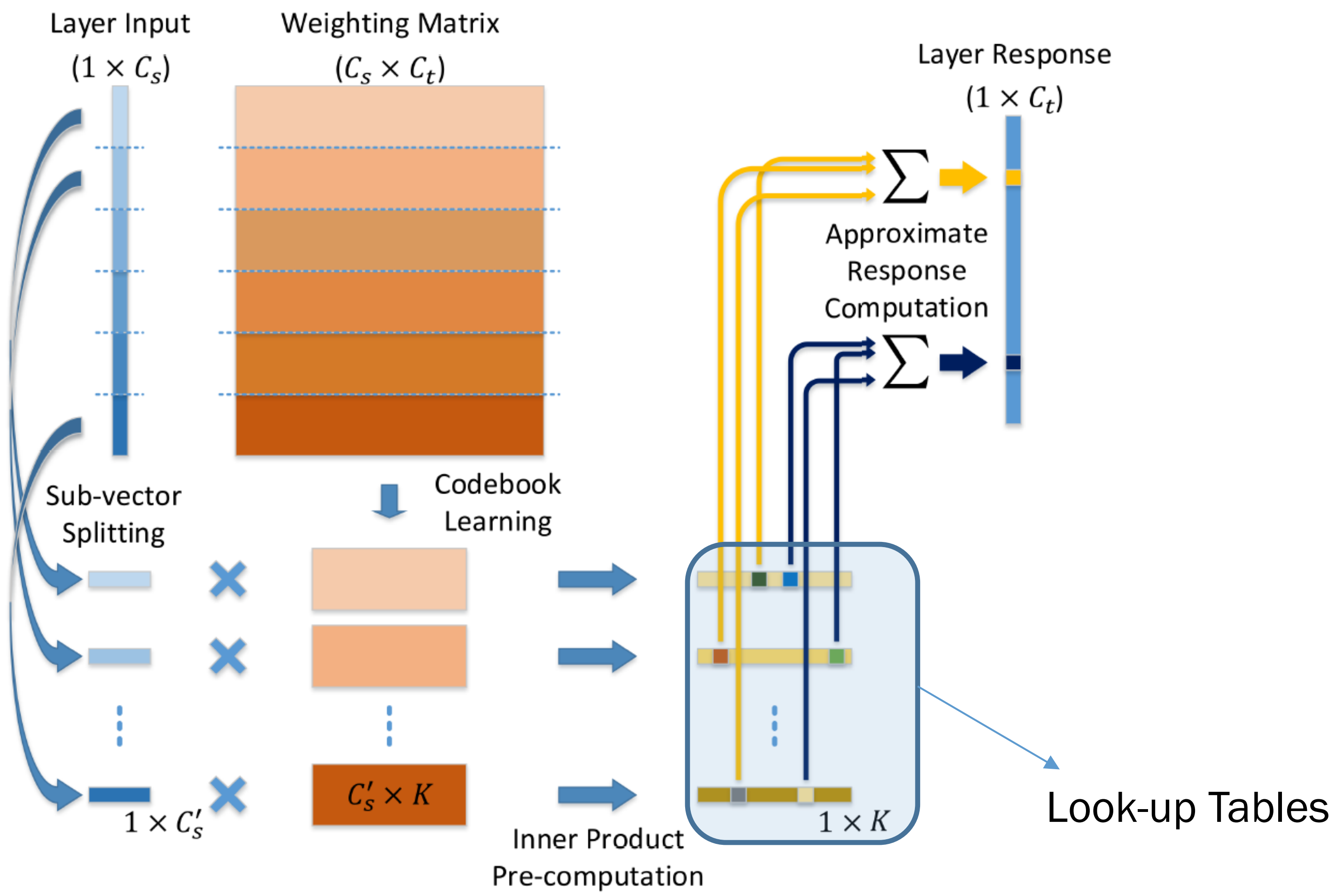
- Apply product quantization in fc. layer



Y. Gong et al., “Compressing Deep Convolutional Networks using Vector Quantization”. Facebook, arXiv 2014.

Quantized-CNN

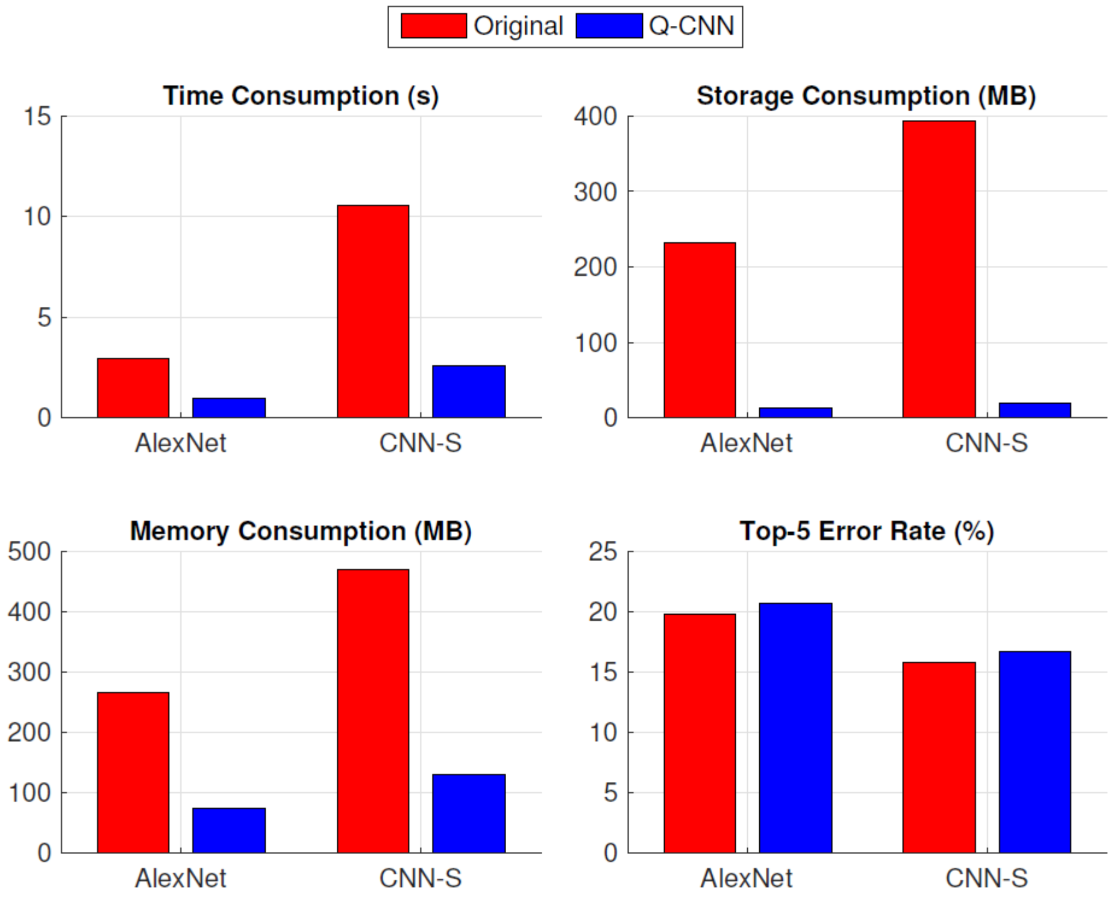
- Accelerate fc. layer with pre-computation and lookup tables



J. Wu et al., “Quantized Convolutional Neural Networks for Mobile Devices”. CVPR 2016.

Quantized-CNN

- Results on mobile devices



J. Wu et al., “Quantized Convolutional Neural Networks for Mobile Devices”. CVPR 2016.

Summary of Code-based Quantization

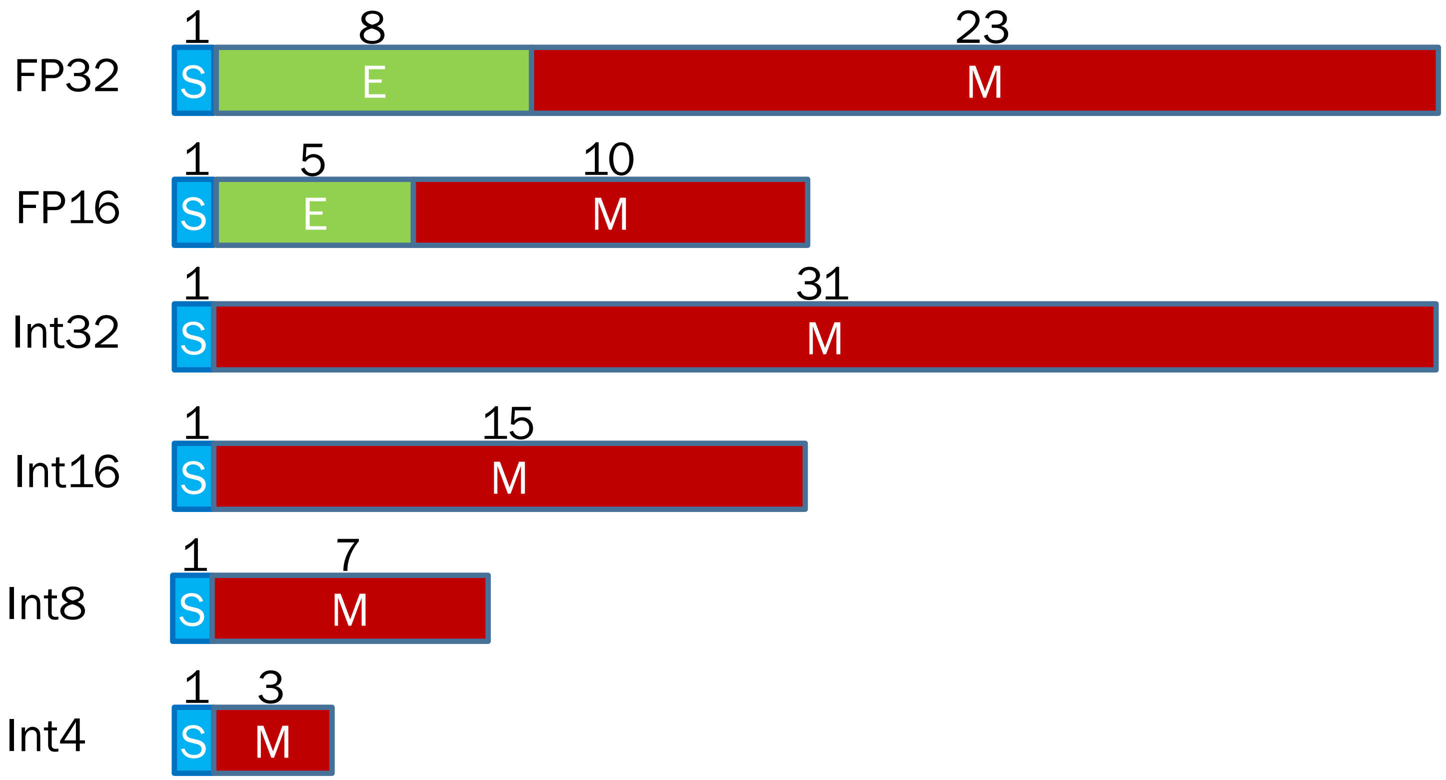
- Quantize weights with codebook(s)
  - Compression: floating-point numbers -> indices
  - Acceleration: via pre-computation and lookup tables

Fixed point Quantization

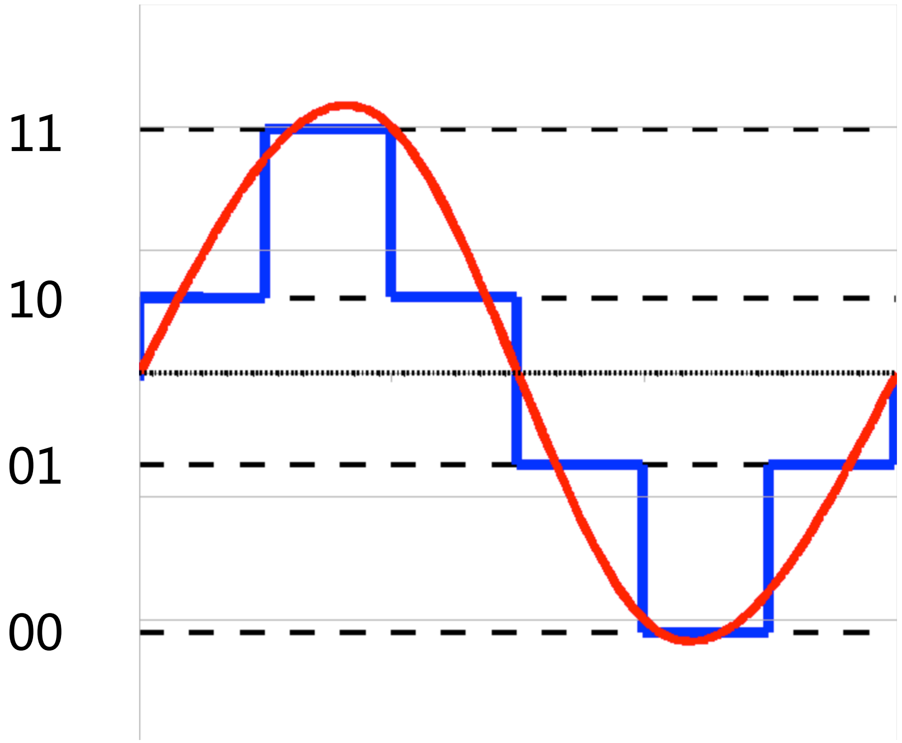
$\boldsymbol{S}$: sign
$\mathbf{M}$: Mantissa
$\boldsymbol{E}$: Exponent

$$(-\mathbf{1})^{S}\times\mathbf{1.M}\times\mathbf{2}^{E}$$

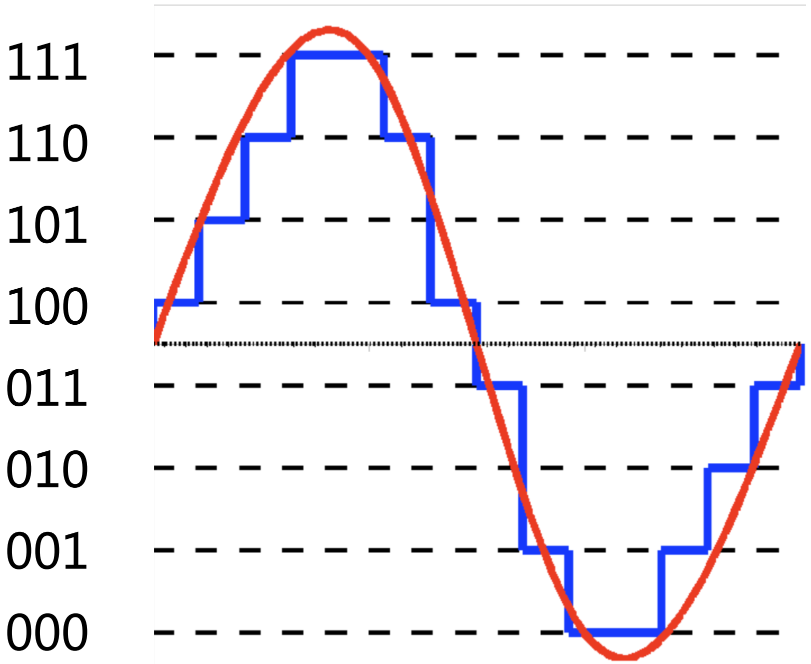
$$(-\mathbf{1})^{S}\times\mathbf{M}$$



Fixed point Quantization: WHAT?



2-bit fixed-point quantization
of sine function

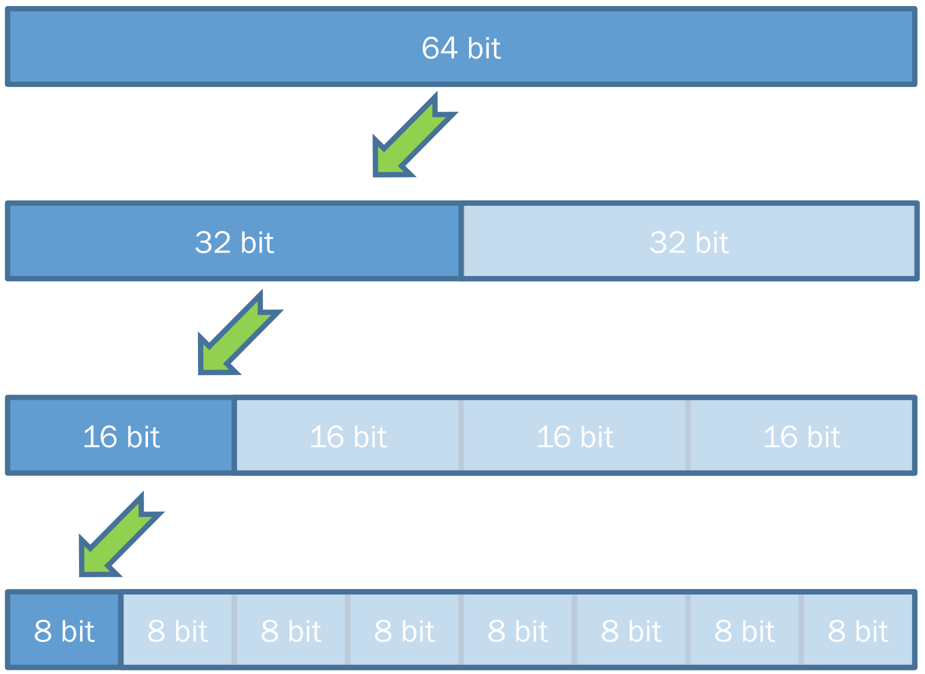


3-bit fixed-point quantization
of sine function.

Fixed point Quantization: WHY?

Fixed point Quantization: WHY?

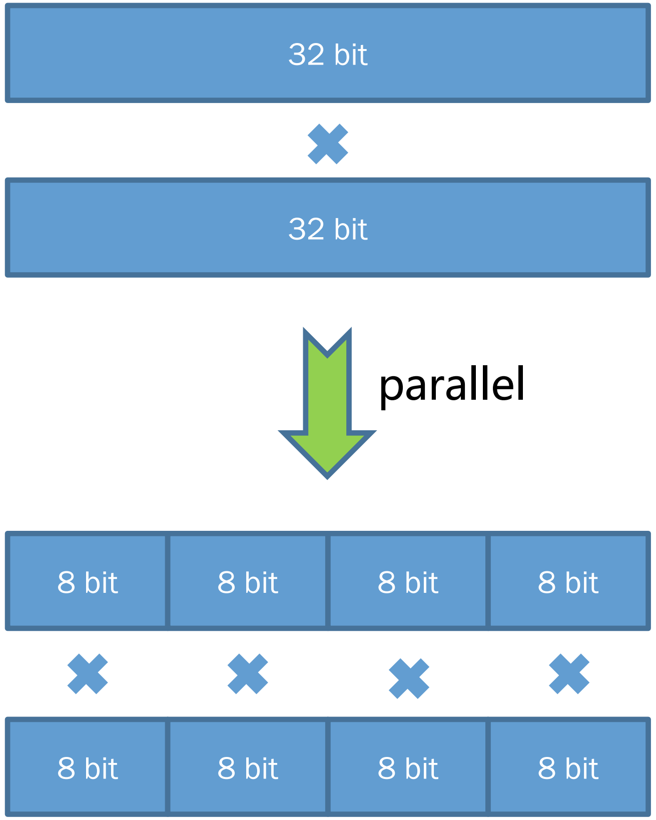
Reducing Storage/Memory



Fixed point Quantization: WHY?

Reducing Storage/Memory

Higher Throughput



Fixed point Quantization: WHY?

Reducing Storage/Memory

Higher Throughput

More Energy Efficient

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

Mark Horowitz , “Computing’s Energy Problem”. ISSCC 2014.

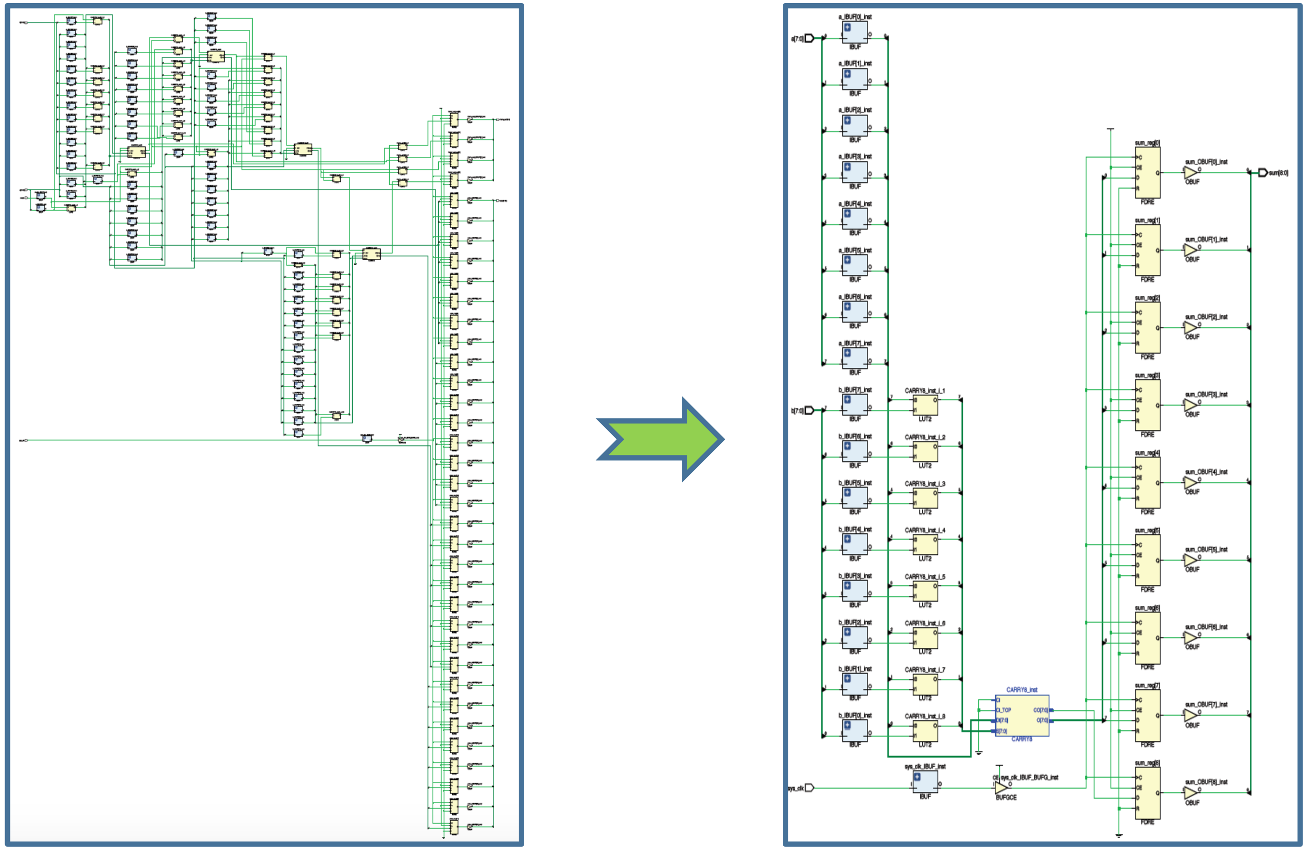
Fixed point Quantization: WHY?

Reducing Storage/Memory

Higher Throughput

More Energy Efficient

Simplify Hardware Design



N-bit Fixed point Quantization: HOW?

2^N *values*: $\underbrace{000\ldots000}_{N-\text{bit}} \sim 111\ldots111$

Non-uniform Quantization

Uniform Quantization

Logarithmic Quantization

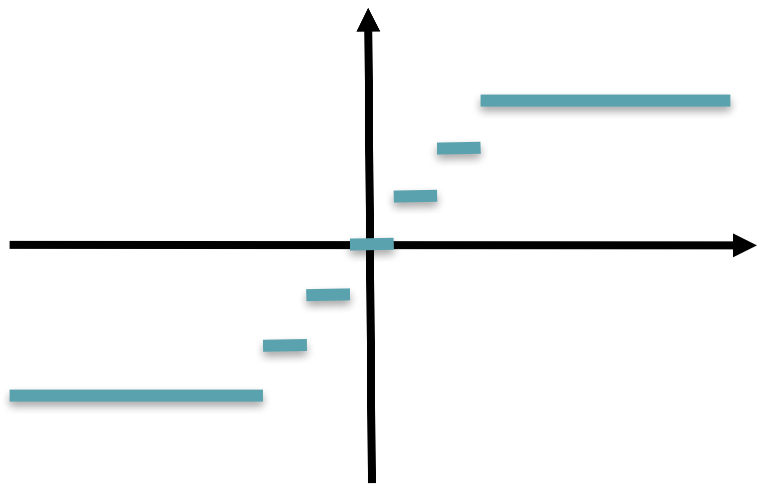
Scalar quantization with/without constrains

	Non-uniform	Uniform	Logarithmic
0...000	C_0	0	0
0...001	C_1	1	1
0...010	C_2	2	2
0...011	C_3	3	4
0...100	C_4	4	8
0...101	C_5	5	16
0...110	C_6	6	32
.	.	.	.
.	.	.	.
.	.	.	.
1...111	$C\_(2^N - 1)$	$2^N - 1$	$2^{2^N - 2}$

N-bit Fixed point Quantization: HOW?

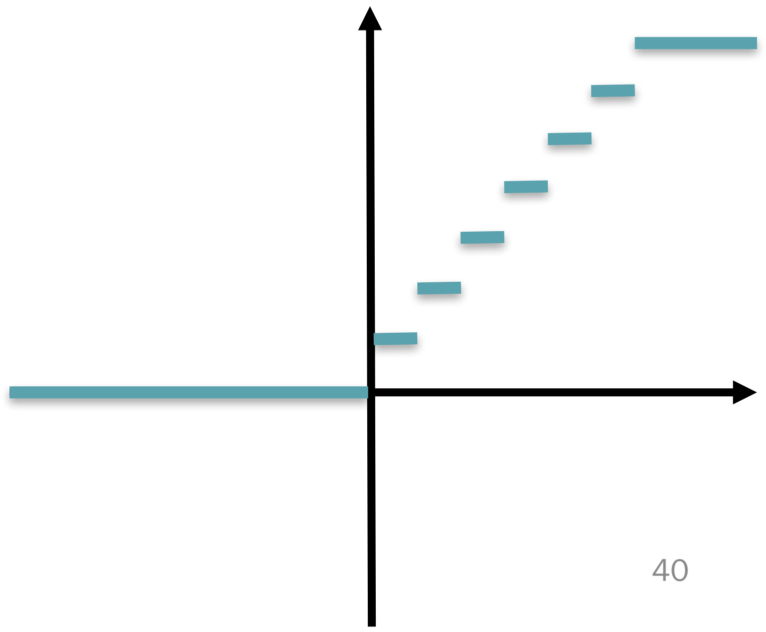
Signed Quantization (weights/activations):

$$\{-(2^{n-1}-1)\Delta, \cdots, -2\Delta, -1\Delta, 0, \Delta, 2\Delta, \cdots, (2^{n-1}-1)\Delta\}$$



Unsigned Quantization (activations):

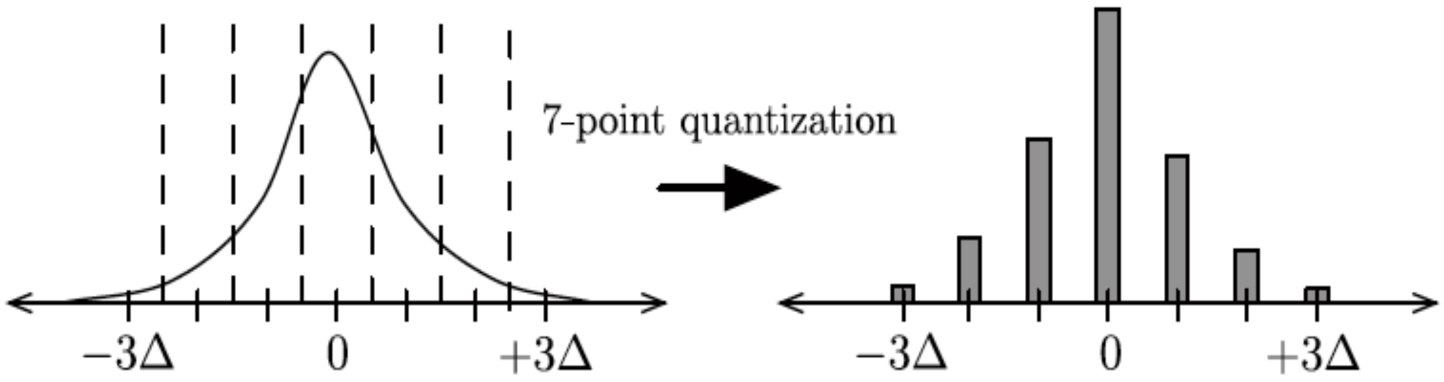
$$\{0,\ \Delta, 2\Delta, \cdots, (2^n-1)\Delta\}$$



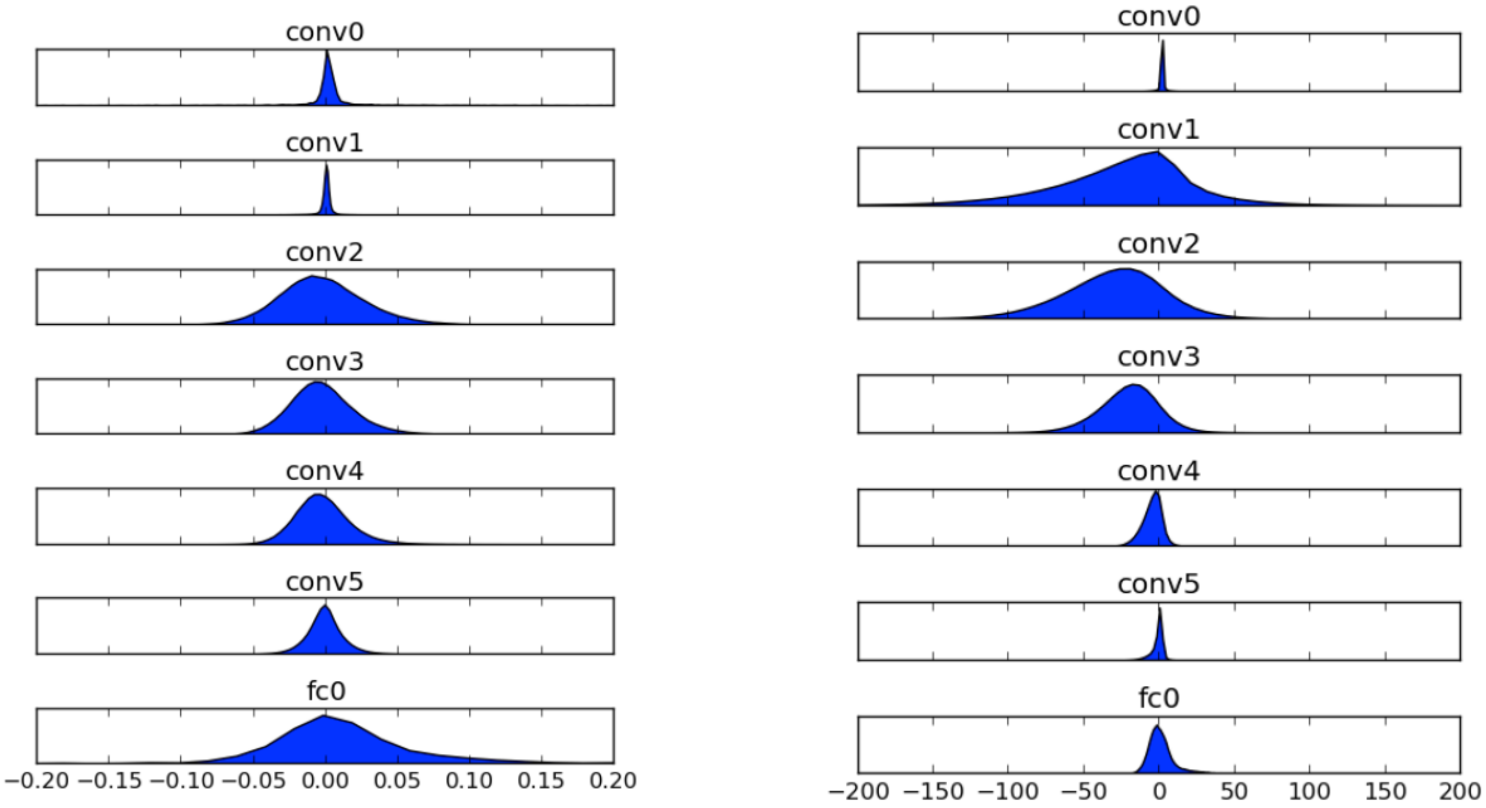
N-bit Fixed point Quantization: HOW?

3-bit Quantization for Normal Distribution

Signed: $\{-(2^{n-1}-1)\Delta, \cdots, -2\Delta, -1\Delta, 0, \Delta, 2\Delta, \cdots, (2^{n-1}-1)\Delta\}$



Weight/Activation Distribution for CNN



(a) Histogram of weight values (b) Histogram of output activation values

Figure 2: Distribution of weights & activations in a DCN design for CIFAR-10 benchmark.

Lin, Darryl, Sachin Talathi, and Sreekanth Annapureddy. "Fixed point quantization of deep convolutional networks." International Conference on Machine Learning. 2016.

Quantization

- Problem: $W \approx \alpha B$
- Optimization: $\min \left|\left|W - \alpha B\right|\right|^2$
- Solution(Iterative optimization):
  - Given alpha, optimize B:

$$B = sgn(w)\min\left\{\left\lfloor\frac{|w|}{\alpha} + 0.5\right\rfloor, 127\right\}$$

Round() + Clip()

  - Given B, optimize alpha:

$$\alpha = \frac{\sum_i w_i b_i}{\sum_i b_i^2}$$

Performance of Quantization

Direct Quantization of Weights of AlexNet, on ImageNet Benchmark

acceptable

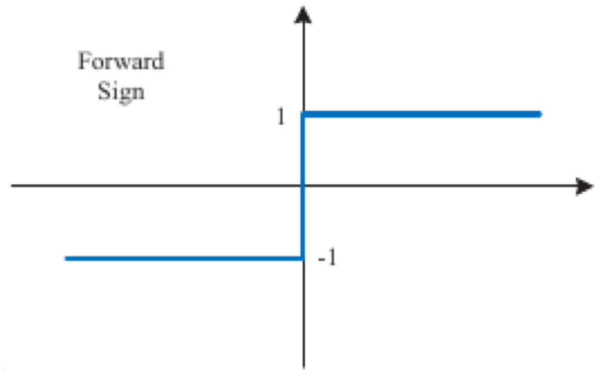
inacceptable

AlexNet		Full	8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Uniform	Acc@Top-1	60.92%	60.85%	60.91%	60.42%	56.55%	49.17%	32.70%	5.61%	0.10%
	Acc@Top-5	81.84%	81.83%	81.77%	81.51%	78.35%	71.48%	52.33%	12.25%	0.50%
Logarithmic	Acc@Top-1	60.92%	59.10%	59.03%	59.80%	59.28%	59.40%	46.18%	4.17%	0.14%
	Acc@Top-5	81.84%	80.29%	80.40%	80.52%	80.50%	80.42%	69.27%	9.77%	0.60%

Finetune

- Problems of Quantization:
  - Non-differentiable
  - Not continuous (not learn anything)

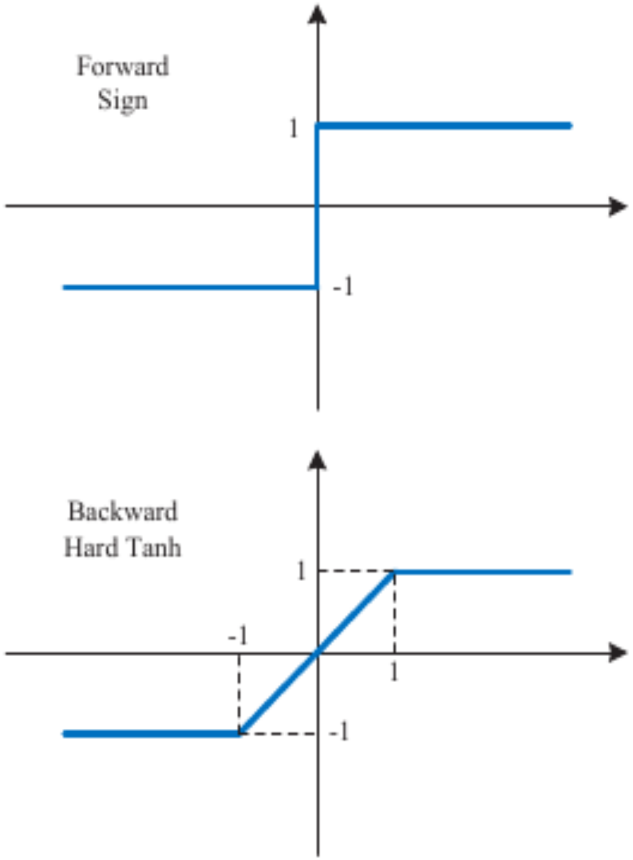
$$\text{ROUND}(w + \delta) = w$$



Binary Quantization

Finetune

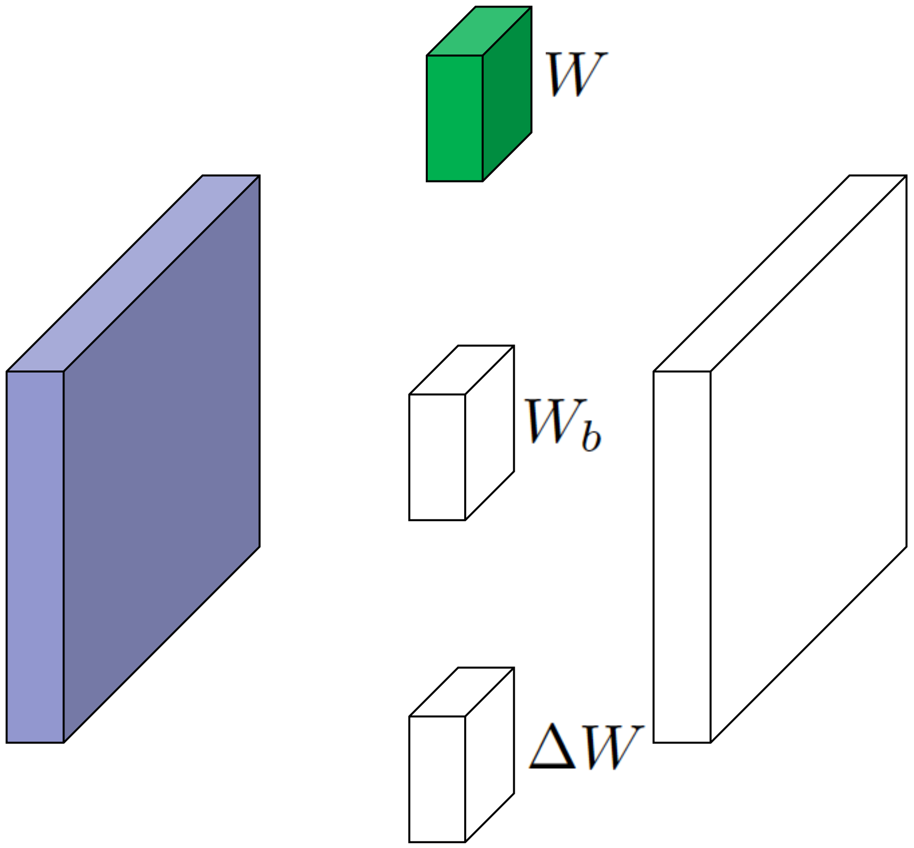
- Problems of Quantization:
  - Non-differentiable
    - Straight-through estimation
  - Not continuous
    - Two copies of weights
    - Fixed-point: for gradient computation
    - Floating-point: for gradient accumulation



Courbariaux M, Bengio Y, David J P. BinaryConnect: training deep neural networks with binary weights during propagations. International Conference on Neural Information Processing Systems. 2015.

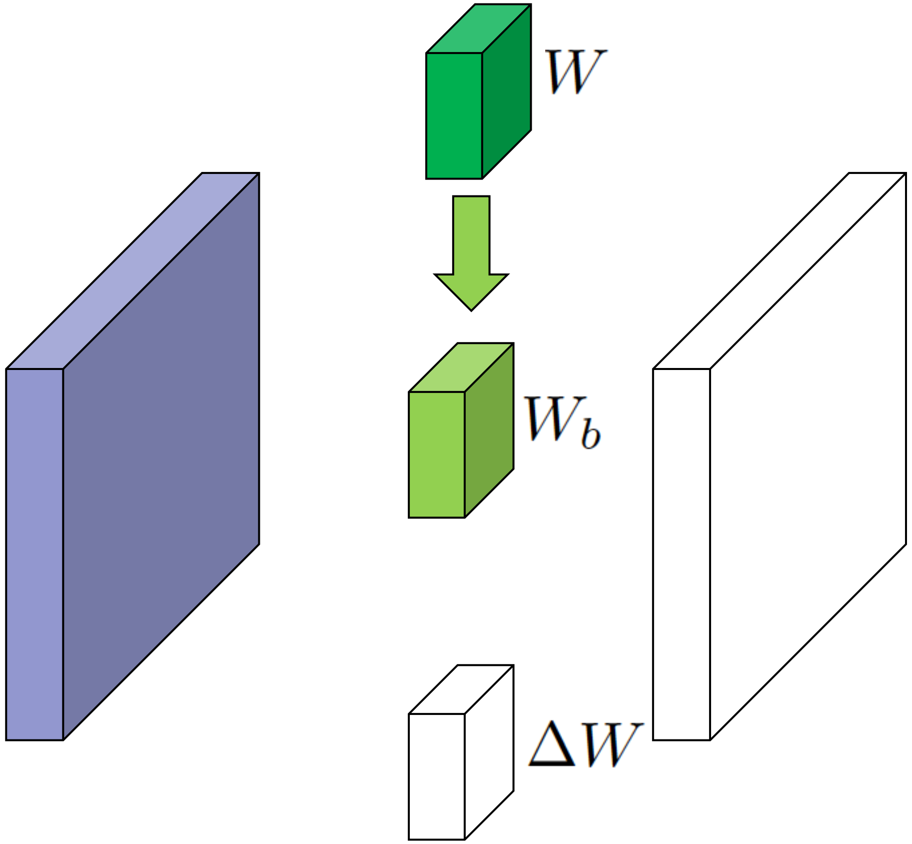
Finetune

```
Randomly initialize W
while not converge do
    W_b = q(W)
    Forward computation with input and W_b
    Backward computation with output and W_b
    Update W = W − ΔW
end while
```



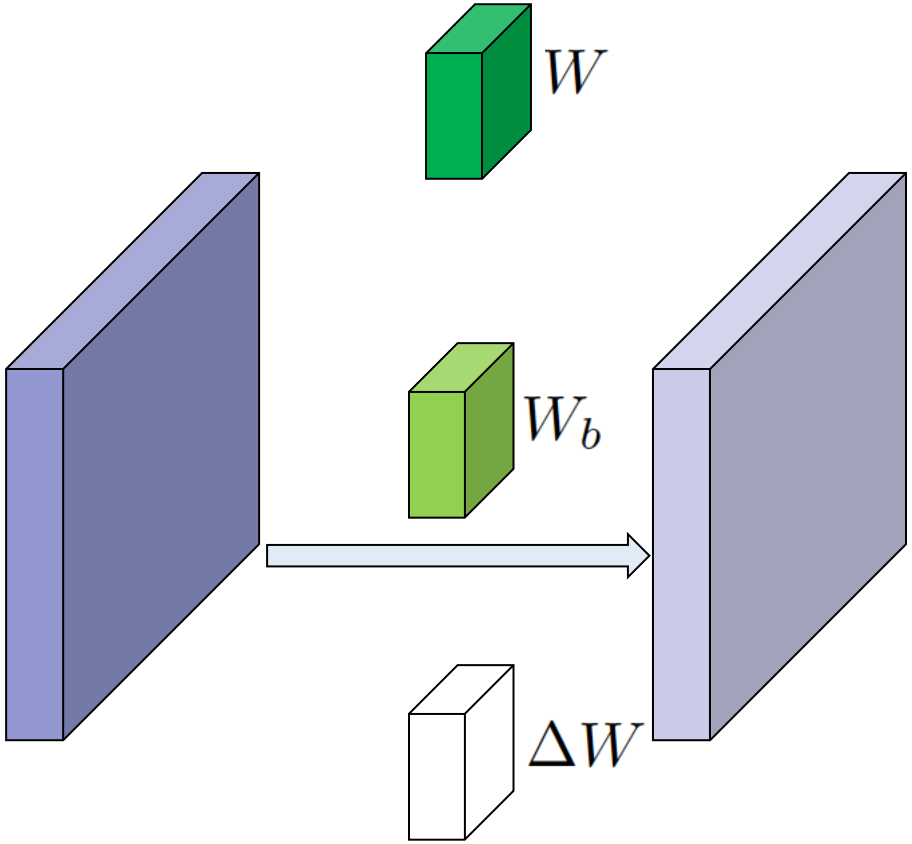
Finetune

```
Randomly initialize W
while not converge do
    W_b = q(W)
    Forward computation with input and W_b
    Backward computation with output and W_b
    Update W = W − ΔW
end while
```



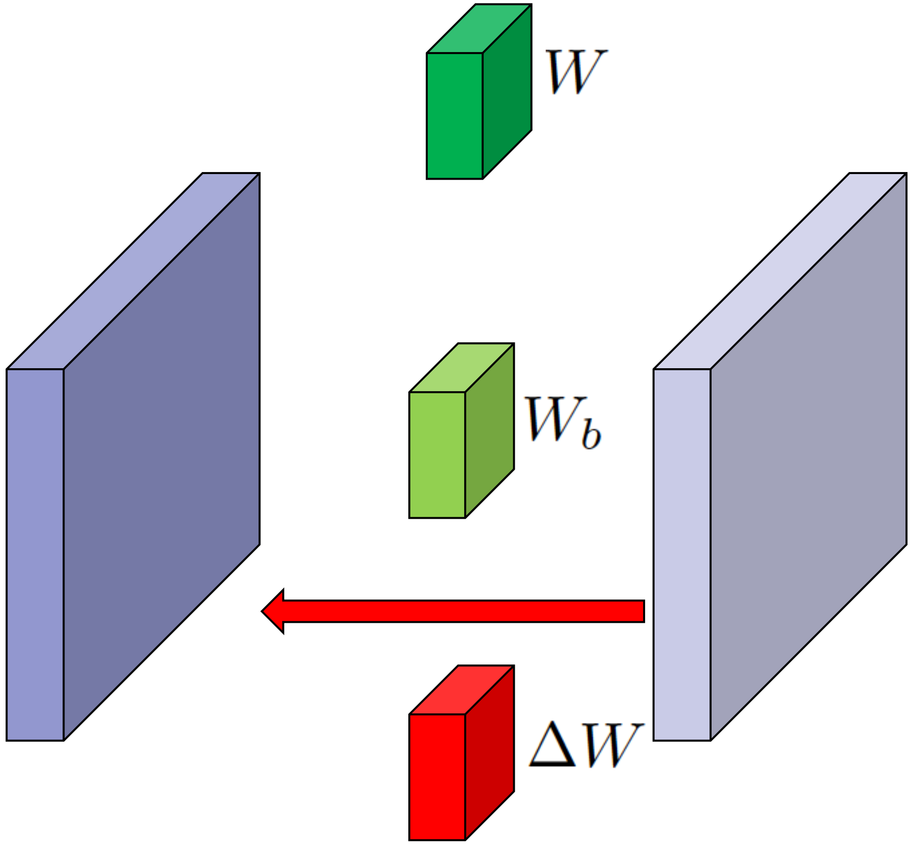
Finetune

```
Randomly initialize W
while not converge do
    W_b = q(W)
    Forward computation with input and W_b
    Backward computation with output and W_b
    Update W = W − ΔW
end while
```



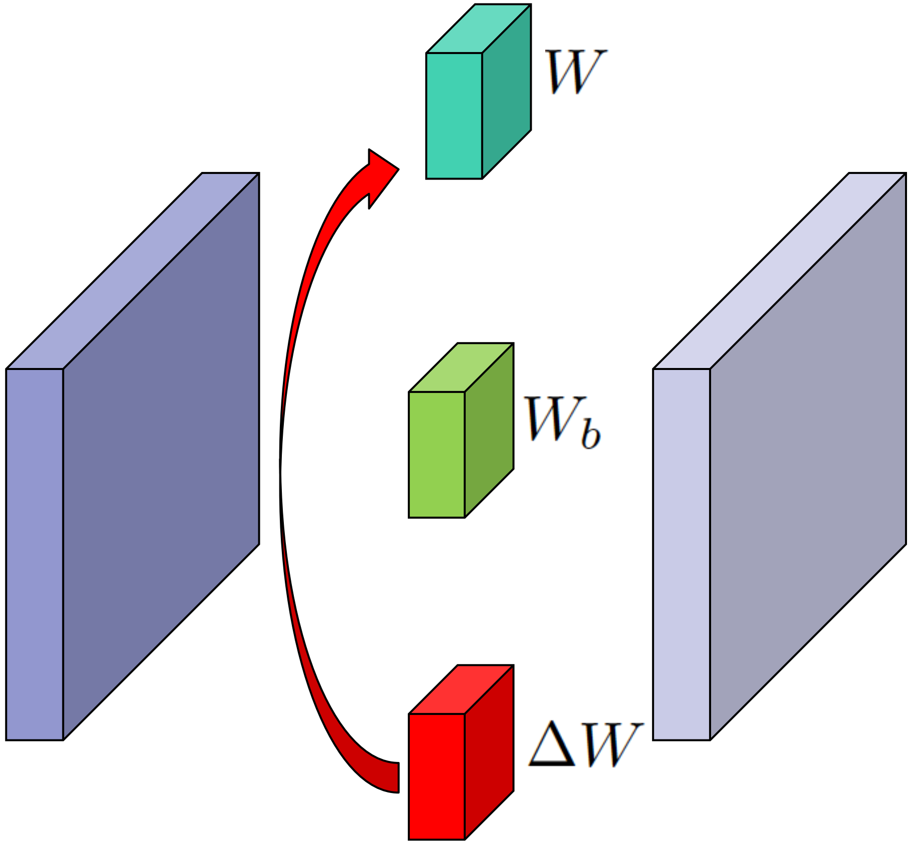
Finetune

1: Randomly initialize W
2: **while** not converge **do**
3: $W_b = q(W)$
4: Forward computation with input and W_b
5: Backward computation with output and W_b
6: Update $W = W - \Delta W$
7: **end while**



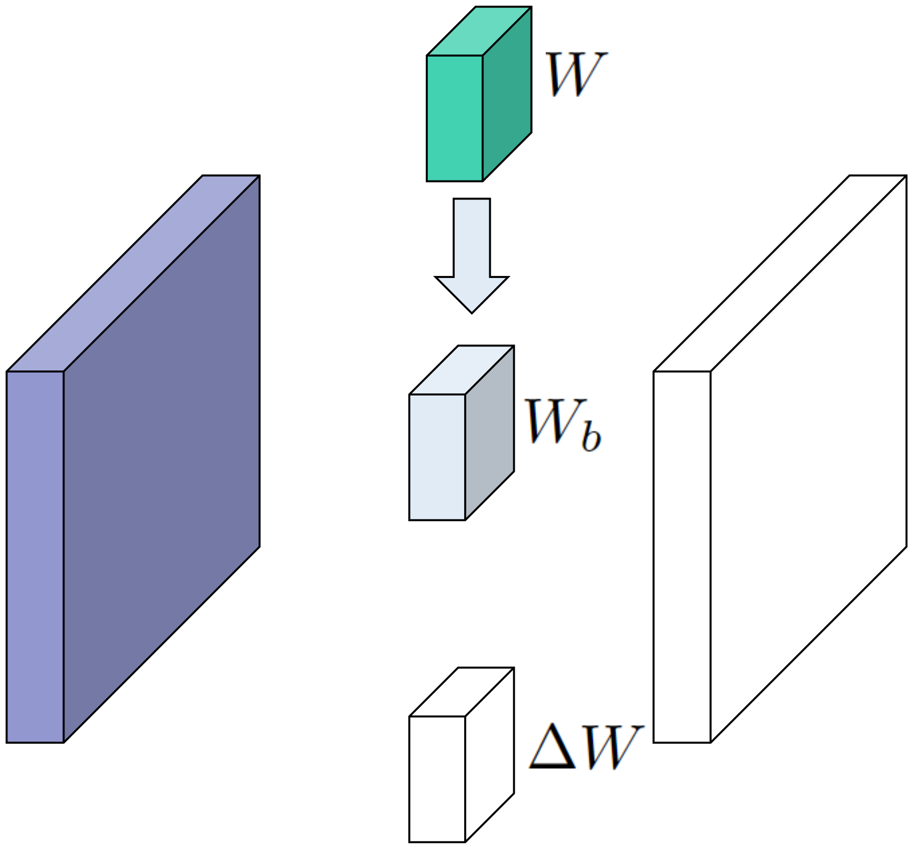
Finetune

Randomly initialize W
while not converge **do**
    $W_b = q(W)$
    Forward computation with input and W_b
    Backward computation with output and W_b
    Update $W = W - \Delta W$
end while



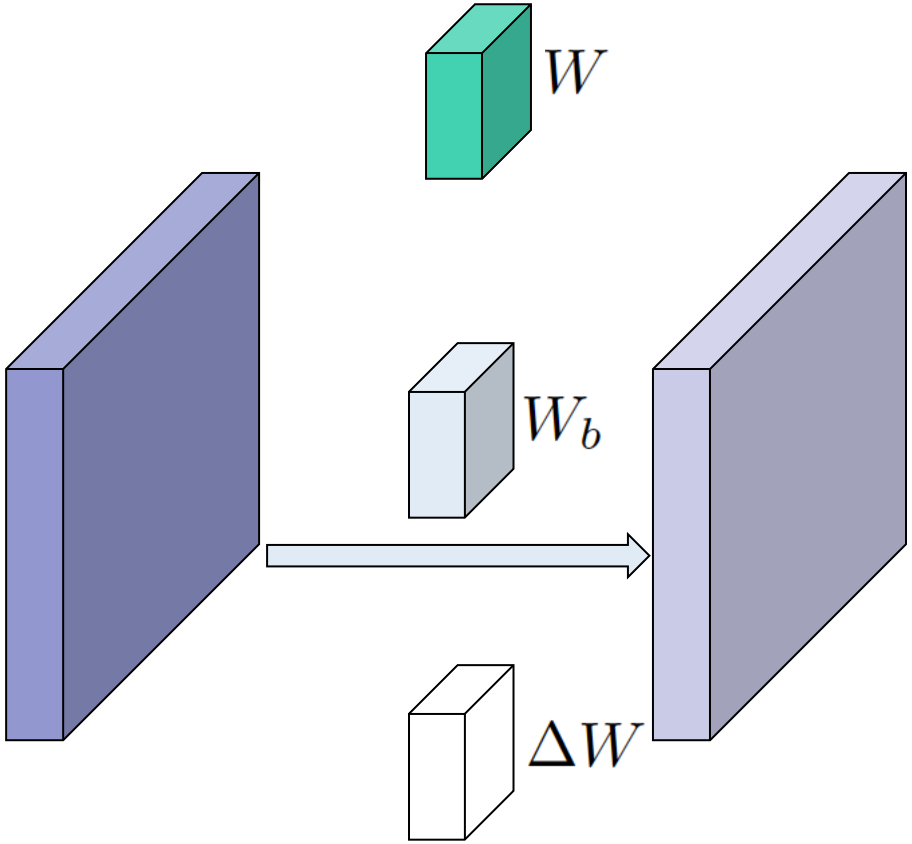
Finetune

1: Randomly initialize W
2: **while** not converge **do**
3: $\boxed{W_b = q(W)}$
4: Forward computation with input and W_b
5: Backward computation with output and W_b
6: Update $W = W - \Delta W$
7: **end while**



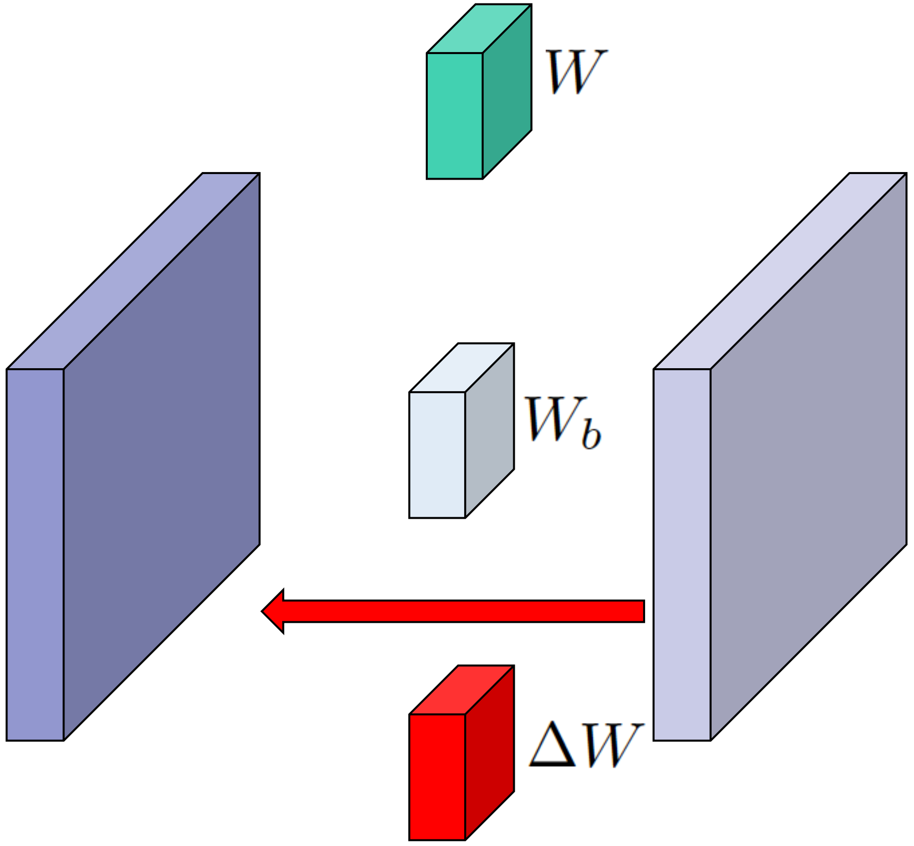
Finetune

Randomly initialize W
while not converge **do**
    $W_b = q(W)$
    Forward computation with input and W_b
    Backward computation with output and W_b
    Update $W = W - \Delta W$
end while



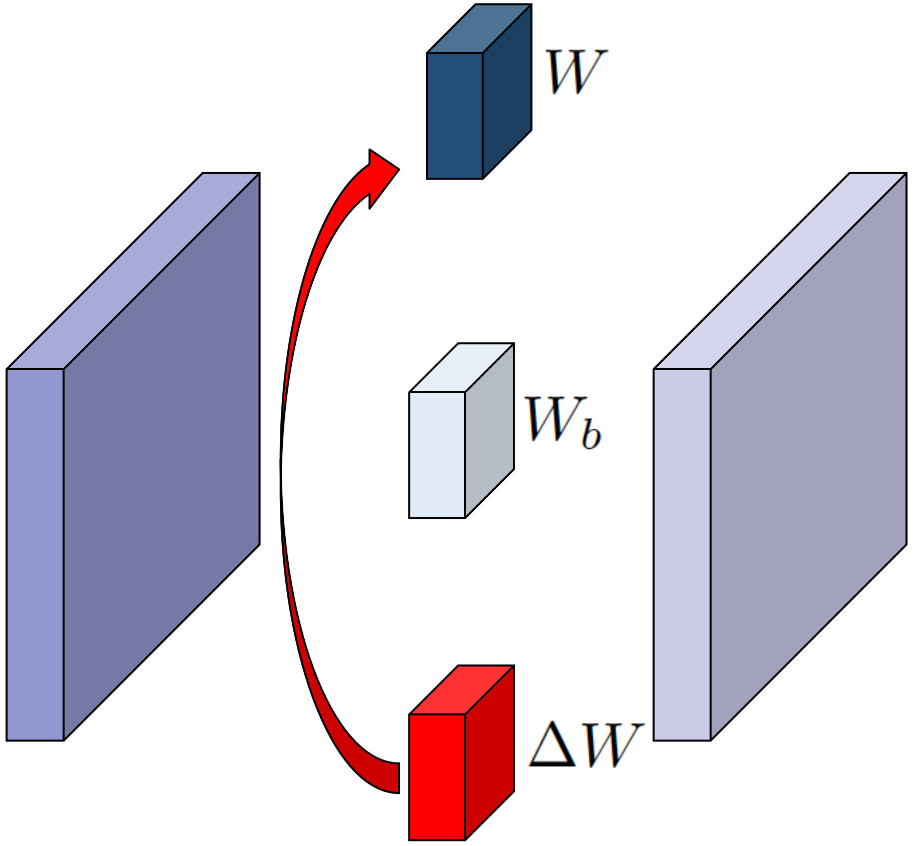
Finetune

```
Randomly initialize W
while not converge do
    W_b = q(W)
    Forward computation with input and W_b
    Backward computation with output and W_b
    Update W = W − ΔW
end while
```



Finetune

Randomly initialize W
while not converge **do**
  $W_b = q(W)$
  Forward computation with input and W_b
  Backward computation with output and W_b
  Update $W = W - \Delta W$
end while



BinaryConnect

$$\min \; L(B) = ||W - B||_F^2$$
$$s.t. \;\; B \in \{+1, -1\}^{C \times N}$$

Deterministic Binarization:

$$w_b = \begin{cases} +1 & if \; w \geq 0 \\ -1 & otherwise \end{cases}$$

Stochastic Binarization:

$$w_b = \begin{cases} +1 & with \; probability \; p = \sigma(w) \\ -1 & with \; probability \; 1 - p \end{cases}$$

$$\sigma(x) = clip(\frac{x+1}{2}, 0, 1) = max(0, min(1, \frac{x+1}{2}))$$

Courbariaux M, Bengio Y, David J P. BinaryConnect: training deep neural networks with binary weights during propagations. International Conference on Neural Information Processing Systems. 2015.

Binary-Weight Networks

BWN: $$\min\ L(B) = ||W - BA||_F^2$$
$$s.t.\ \ B \in \{+1, -1\}^{C\times N}$$

$$\underbrace{\begin{bmatrix} 0.2 & 0.1 & -0.2 & 0.4 \\ -0.5 & 0.6 & -0.3 & 0.7 \\ -0.4 & -0.7 & 0.8 & 0.3 \\ 0.9 & 0.4 & -0.6 & 0.7 \end{bmatrix}}_{W} \Rightarrow \underbrace{\begin{bmatrix} 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \end{bmatrix}}_{B} * \underbrace{\begin{bmatrix} 0.23 & 0 & 0 & 0 \\ 0 & 0.53 & 0 & 0 \\ 0 & 0 & 0.55 & 0 \\ 0 & 0 & 0 & 0.65 \end{bmatrix}}_{A}$$

Mohammad et.al, “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”. ECCV 16

Binary Weight Network via Hashing

Instead of minimizing the quantization error of weights:

$$\min \; L(B) = ||W - B||_F^2 \quad s.t. \; B \in \{+1, -1\}^{C \times N}$$

(1) BinaryConnect

$$\min \; L(B) = ||W - BA||_F^2 \quad s.t. \; B \in \{+1, -1\}^{C \times N}$$

(2) BWN

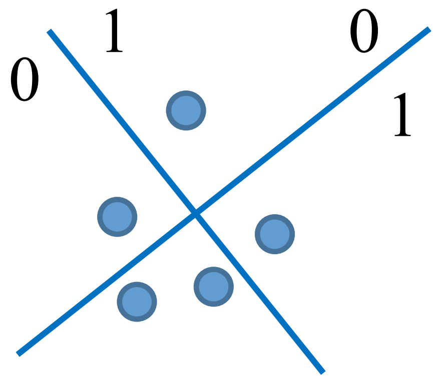
We minimize the quantization error of inner-product similarity:

$$\min \; L(B) = ||X^T W - X^T B||_F^2 \quad s.t. \; B \in \{+1, -1\}^{C \times N}$$

Hu et al. "From Hashing to CNNs: Training Binary Weight Networks via Hashing". AAAI，2018

Binary Weight Network via Hashing

Hashing: $g_r(w) = \begin{cases} 1, & r^T w \geq 0 \\ 0, & r^T w < 0 \end{cases}$



Inner-product Preserving Hashing

$\min ||S - h(X)^T g(W)||_F^2$ where $S = X^T W$

Hu et al. “From Hashing to CNNs: Training Binary Weight Networks via Hashing”. AAAI，2018

Binary Weight Network via Hashing

AlexNet

Method	Classification Accuracy	
	Top1	Top5
BinaryConnect	35.4	61.0
BWN	56.8	79.4
SQ-BWN	51.2	75.1
HWGQ-BWN	52.4	75.9
BWNH (Ours)	**58.5**	**80.9**

ResNet-18

Method	Classification Accuracy	
	Top1	Top5
Full-Precision	69.3	89.2
BWN	60.8	83.0
SQ-BWN	58.3	81.6
HWGQ-BWN	61.3	83.9
BWNH (Ours)	**64.3**	**85.9**

Hu et al. “From Hashing to CNNs: Training Binary Weight Networks via Hashing”. AAAI，2018

Fixed-point Factorized Networks

$$\underset{X,D,Y}{\text{minimize}} \quad \| W - XDY^T \|_F^2$$

$$= \underset{\{x_i\},\{d_i\},\{y_i\}}{\text{minimize}} \quad \| W - \sum_i^k d_i x_i y_i^T \|_F^2$$

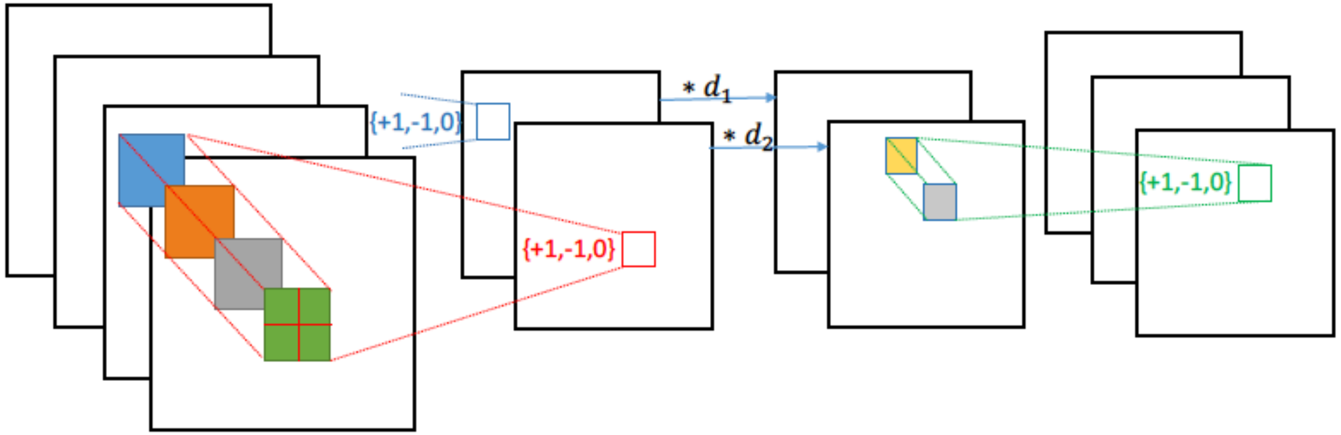
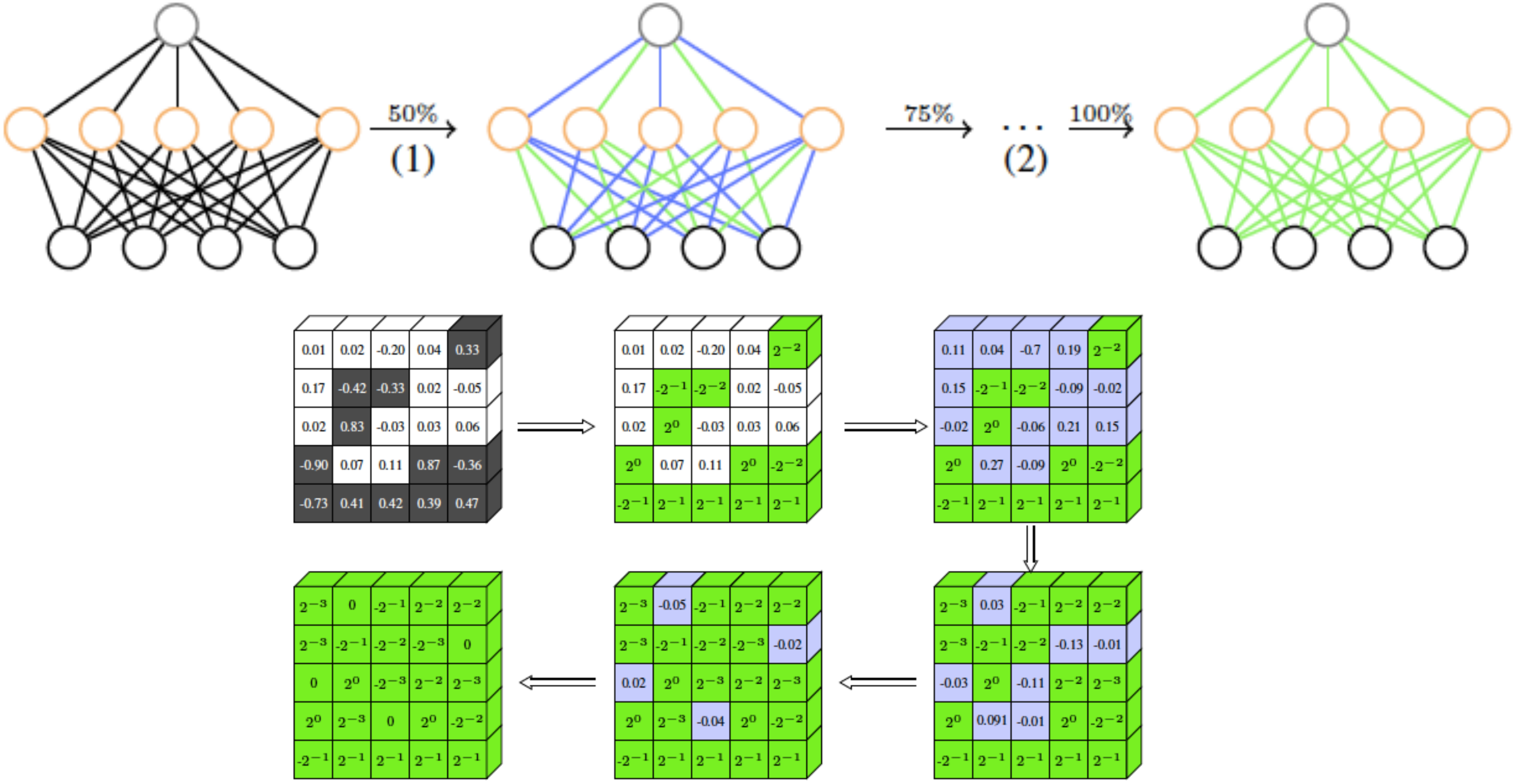


Figure 1. New layers used in our FFN architecture to replace the original convolutional layers.

P. Wang and J.Cheng, "Fixed-point Factorized Networks". CVPR 17

Incremental Quantization

- One-step quantization v.s. Incremental Quantization



Zhou, Aojun, et al. "Incremental network quantization: Towards lossless cnns with low-precision weights." arXiv preprint arXiv:1702.03044 (2017).

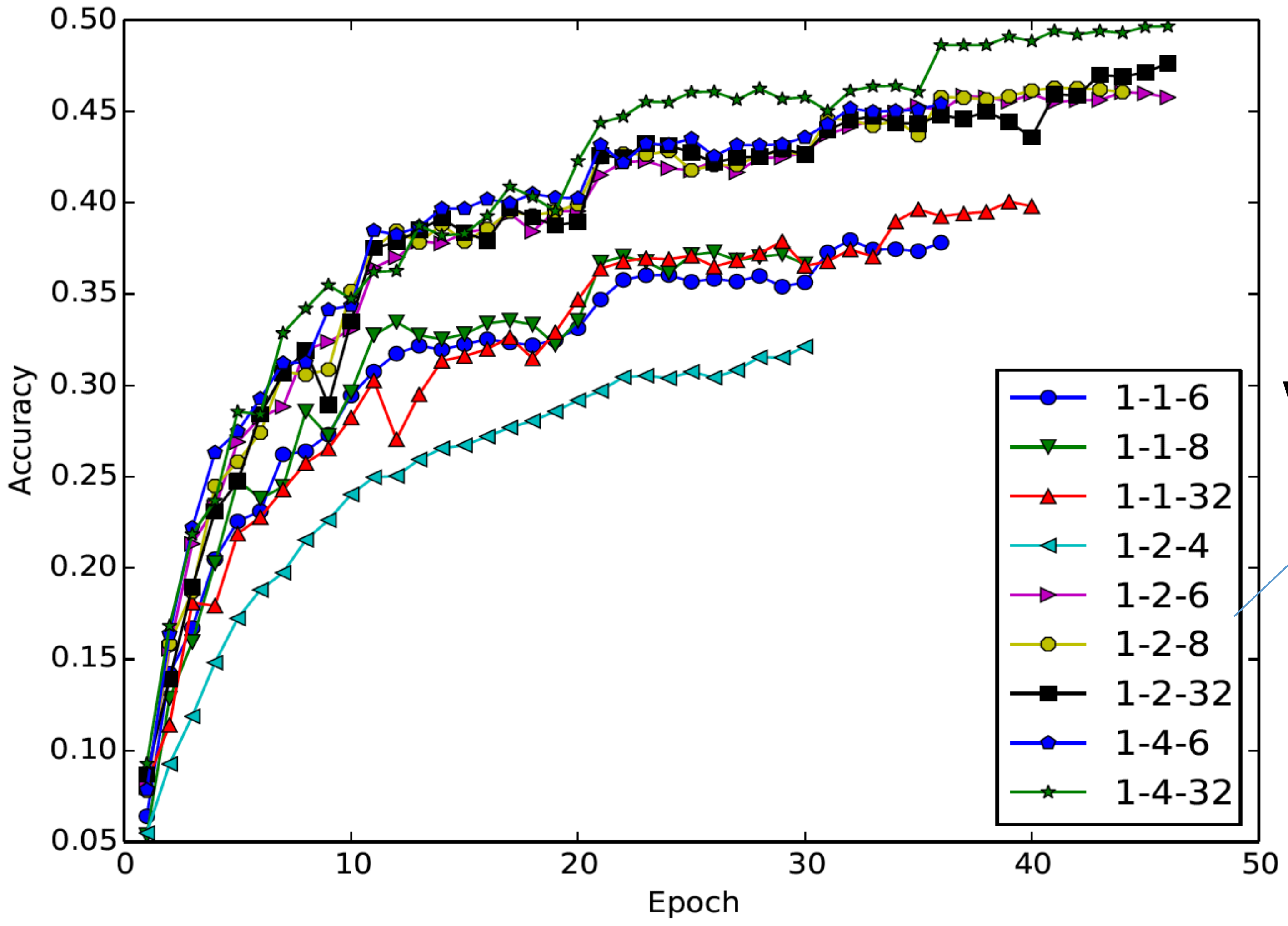
XNOR-Net

- XNOR-Net
- Main idea: binary values * scaling factor

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

Mohammad et.al, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". ECCV 16

DoReFa-Net



Weight-Activation-Gradient

Zhou et.al, “DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients”. arXiv:1606.06160, 2016

Summary of Fixed-point Quantization

- Quantize weights with low-bit representation
  - Compression: floating numbers -> fixed-point numbers
  - Accelerate via fixed-point operation
  - Energy efficiency

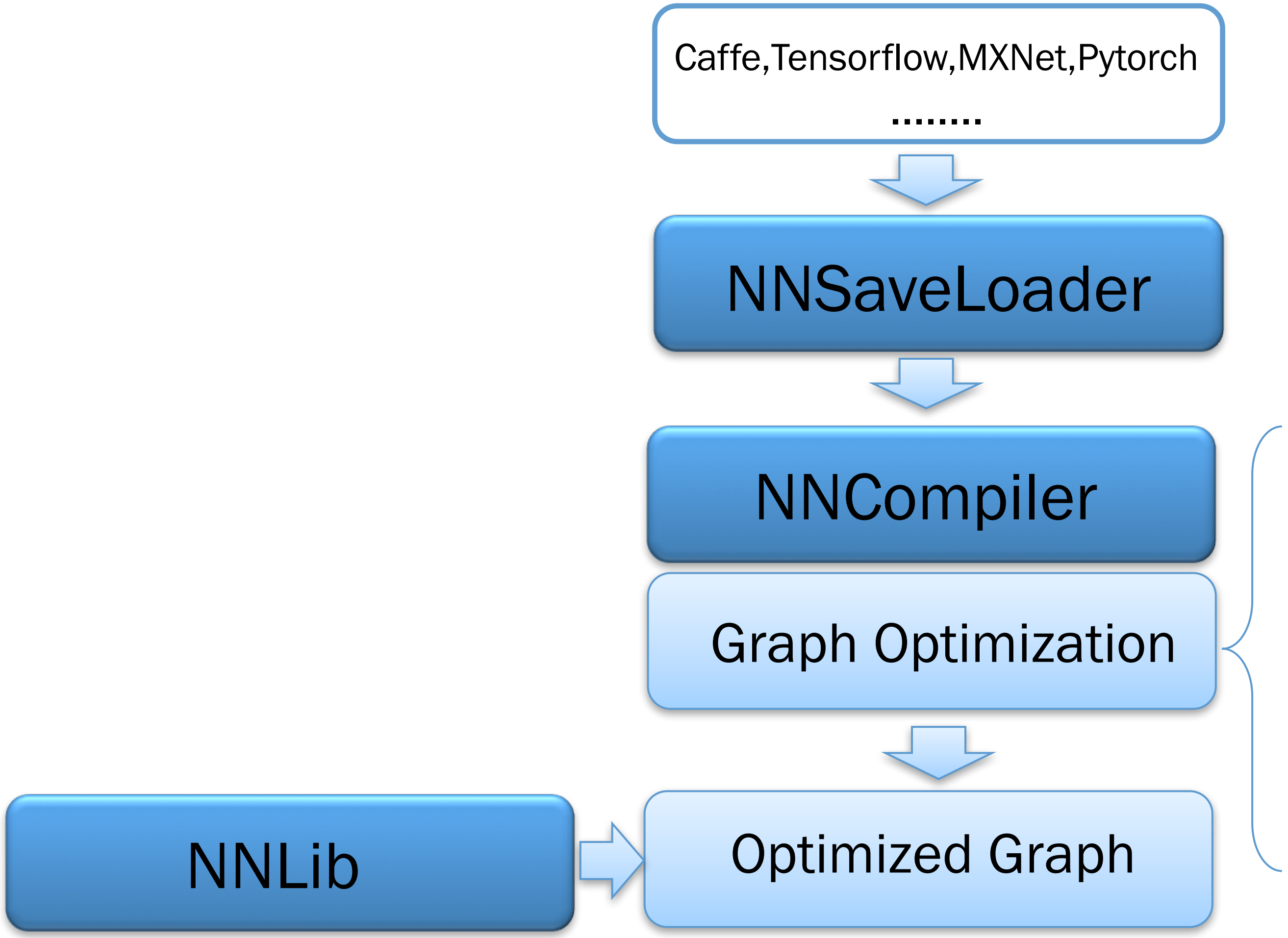
Outline

Q-Engine

- A lite, high-performance and modular deep learning inference engine for embedded devices.

- Designed and developed by Institution of Automation, Chinese Academy of Science.

Q-Engine Architecture



1. Layer Fusion and Combination

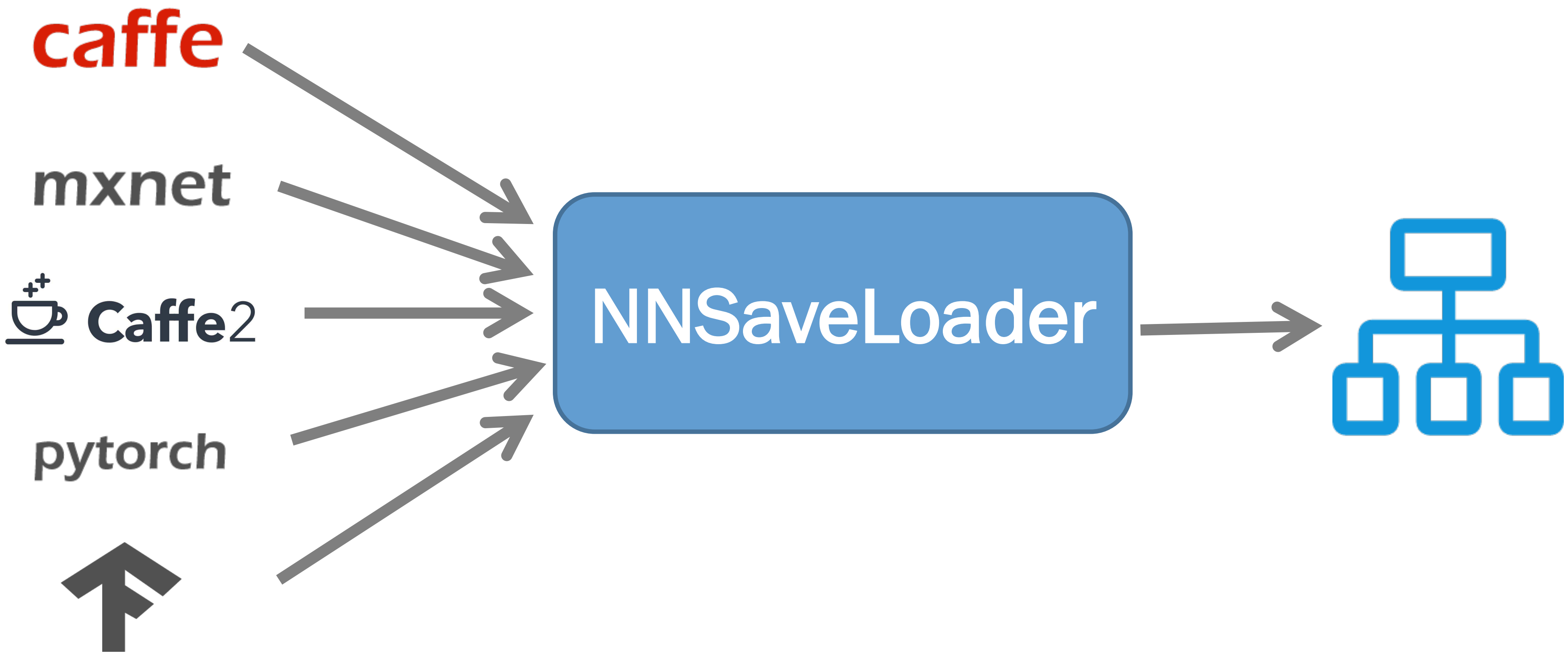
2. Sparse and Int8 Operater

3. Memory Optimization

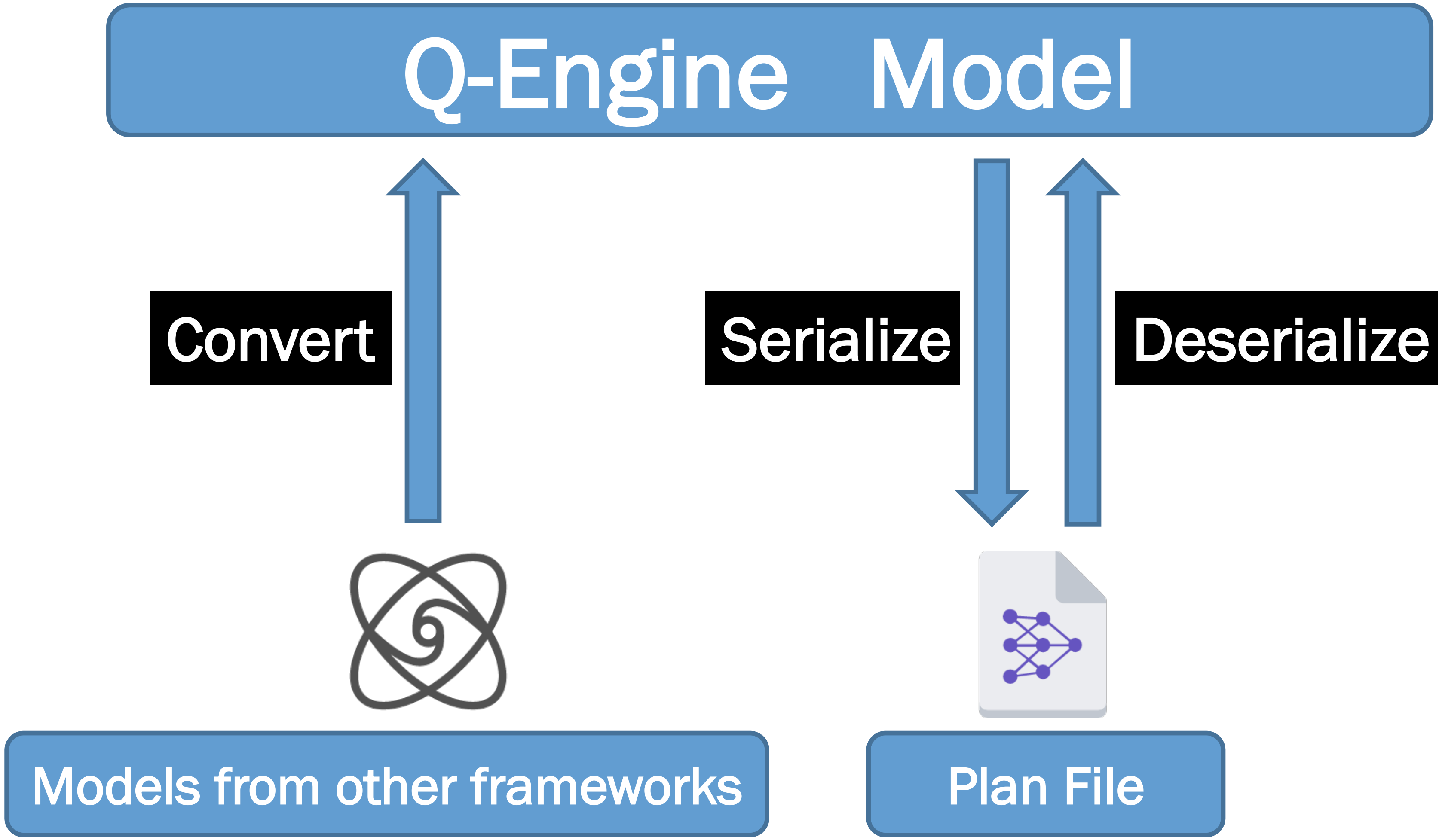
4. Automatic Algorithm Selection

......

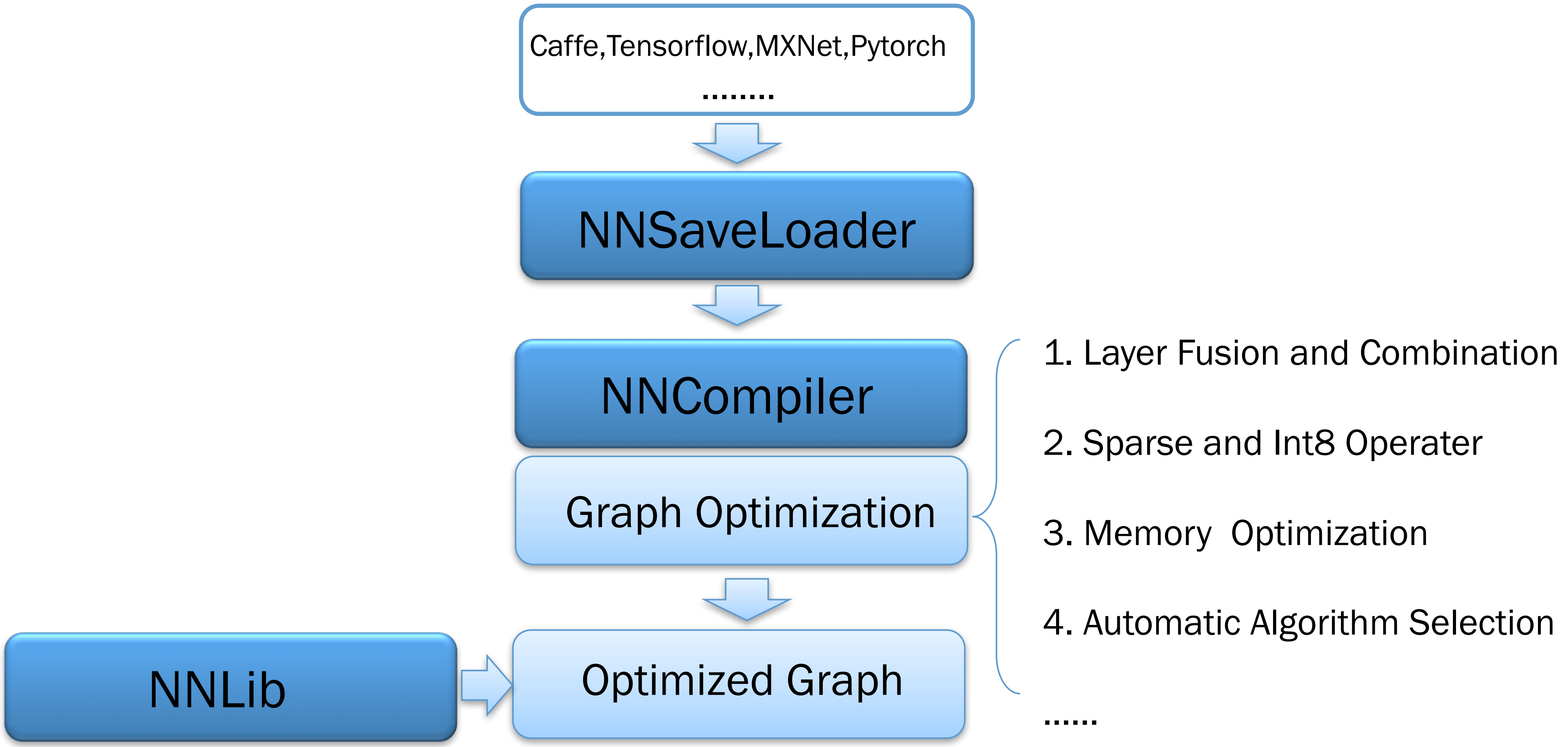
NNSaveLoader



NNSaveLoader



Q-Engine Architecture



NNCompiler

1. Combining and fusing layers

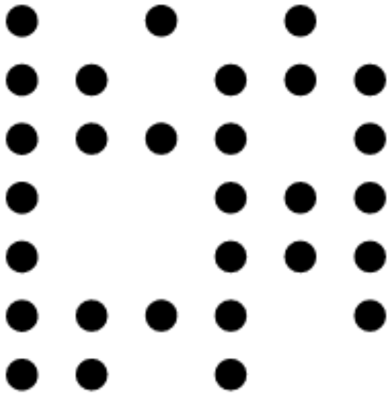
- Convolution and ReLU Activation
- FullyConnected and ReLU Activation
- Scale and Activation
- Convolution and Element-Wise Sum
- Convolution and Scale

…

NNCompiler

2. Sparse and Int8 Operator Supports

Sparse Representation



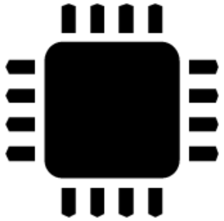
Higher Compression Rate

Low-Precision Approximation

Speed-up Computation

NNCompiler

3. Optimize Memory Management



– Dramatically reduce memory footprint

– Lower memory consumption

– Higher throughput

NNCompiler

4. Automatic Algorithm Selection

Convolution Operator
- Filter Size 3 x 3
  - Large input : Winograd algorithm with 6x6 output tiles
  - Small input : Winograd algorithm with 4x4 output tiles
- Otherwise : our proposed direct convolution

NNLib : Computing kernel library for mobile devices

- High performance direct convolution
- Sparse direct convolution / (sparse GEMM)
- int8 direct convolution / (int8 GEMM)

NNLib : Computing kernel library for mobile devices

Direct convolution: Compute convolution directly without reordering inputs

Runtime of different algorithms/ms

	conv1	conv2	conv3	conv4	conv5	conv6	conv7	conv8	conv9	conv10
Im2col	30.1	108.3	26.0	43.0	25.3	86.9	37.8	15.2	30.4	22.9
MEC	14.0	65.0	31.1	47.6	14.6	44.1	15.7	9.5	18.8	21.9
QEngine	**10.4**	**49.9**	**17.3**	**25.8**	**12.9**	**33.1**	**12.1**	**5.9**	**13.8**	**13.3**

Memory overhead of different algorithms/MB

	conv1	conv2	conv3	conv4	conv5	conv6	conv7	conv8	conv9	conv10
Im2col	6.31	10.48	5.52	8.16	11.23	10.88	8.98	3.31	5.01	4.57
MEC	3.57	4.88	4.49	6.61	5.96	6.06	4.16	2.70	2.59	3.36
QEngine	**1.92**	**3.48**	**3.97**	**5.83**	**3.85**	**3.65**	**1.75**	**1.50**	**1.39**	**2.76**

[1] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Gir-shick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.

[2] M. Cho and D. Brand. MEC: memory-efficient convolution for deep neural network. CoRR, abs/1706.06873, 2017.

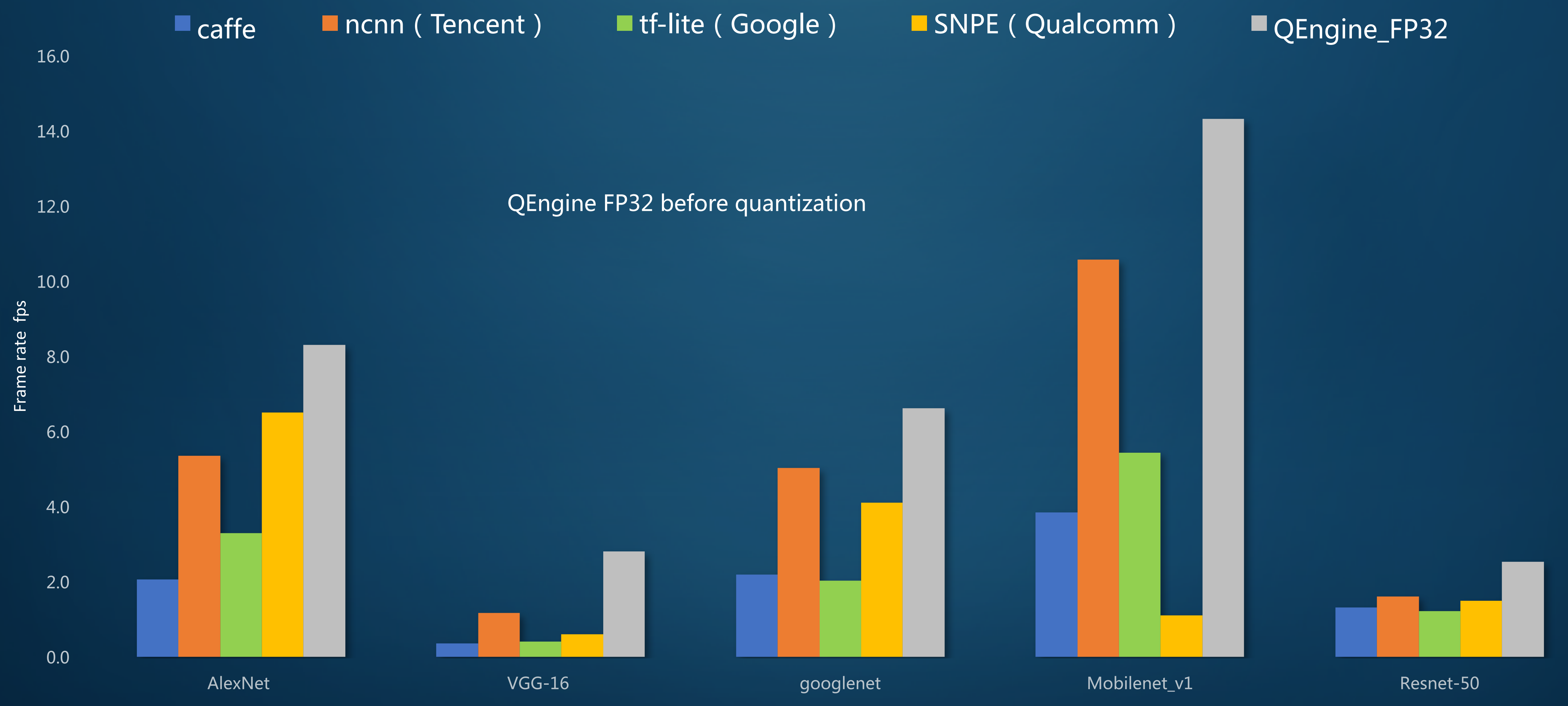
NNLib : Computing kernel library for mobile devices

Sparse direct convolution & Int8 direct convolution

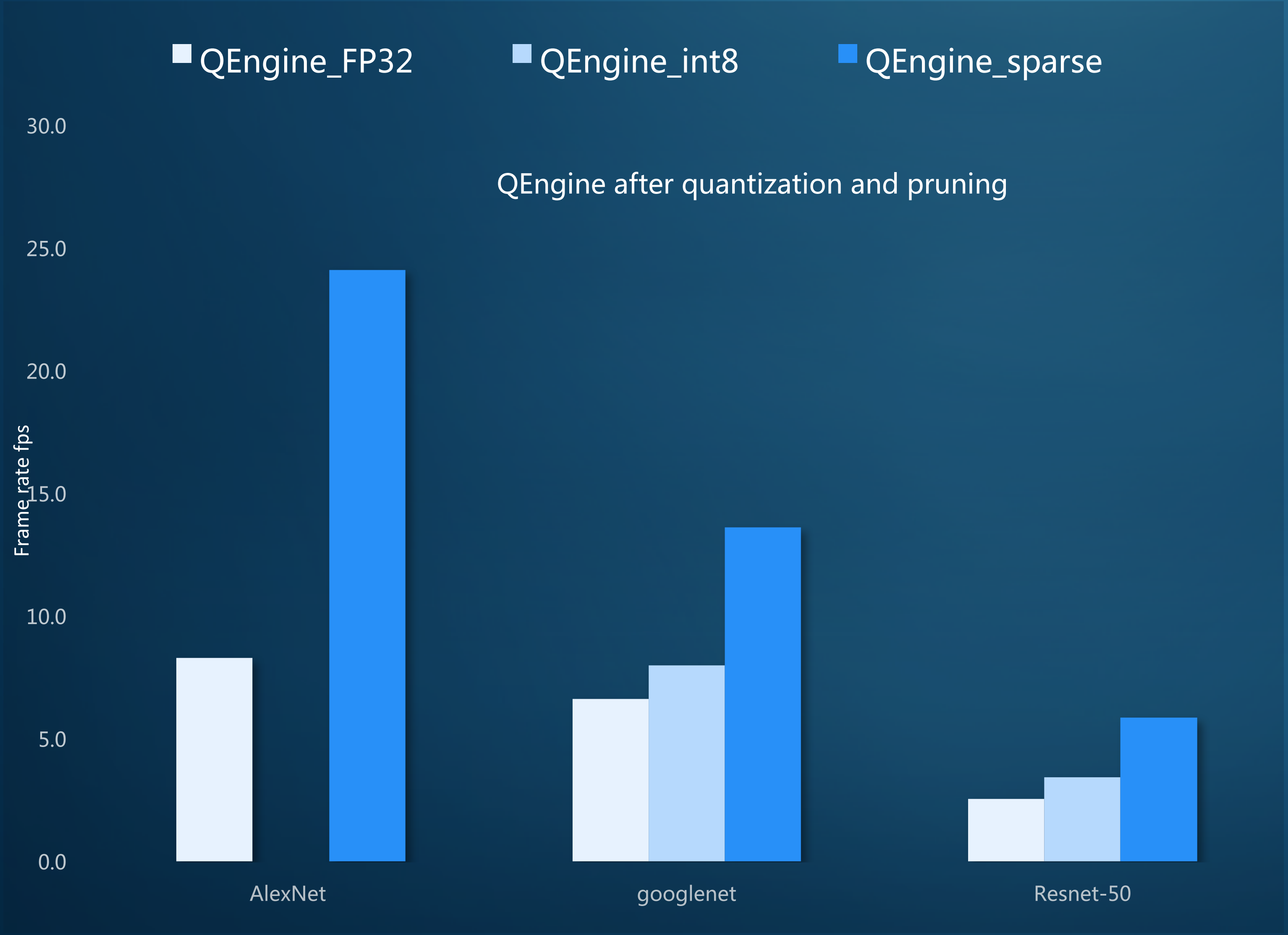
Runtime of different algorithms/ms

		conv1	conv2	conv3	conv4	conv5	conv6	conv7	conv8	conv9	conv10
Dense (direct)		10.4	49.9	17.3	25.8	12.9	33.1	12.1	5.9	13.8	13.3
Sparse (direct)	50%	9.12	36.42	12.62	18.99	10.08	27.38	10.65	5.16	9.90	9.55
	75%	6.38	23.44	8.20	11.93	6.96	17.47	7.45	3.42	6.48	5.99
	90%	4.01	14.09	4.11	6.22	4.57	10.12	4.81	2.27	4.06	3.37
Int8 (direct)		7.83	28.61	8.87	13.12	8.96	21.72	10.20	4.50	8.54	6.85

BenchMark@Snapdragon820E

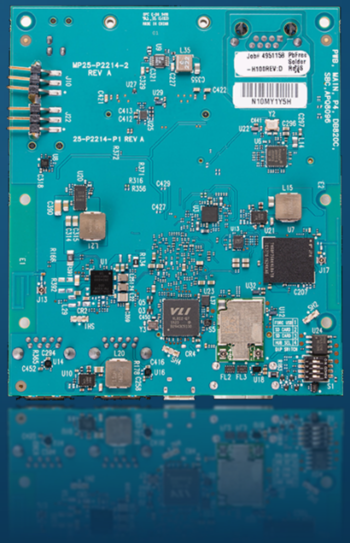


BenchMark@Snapdragon820E

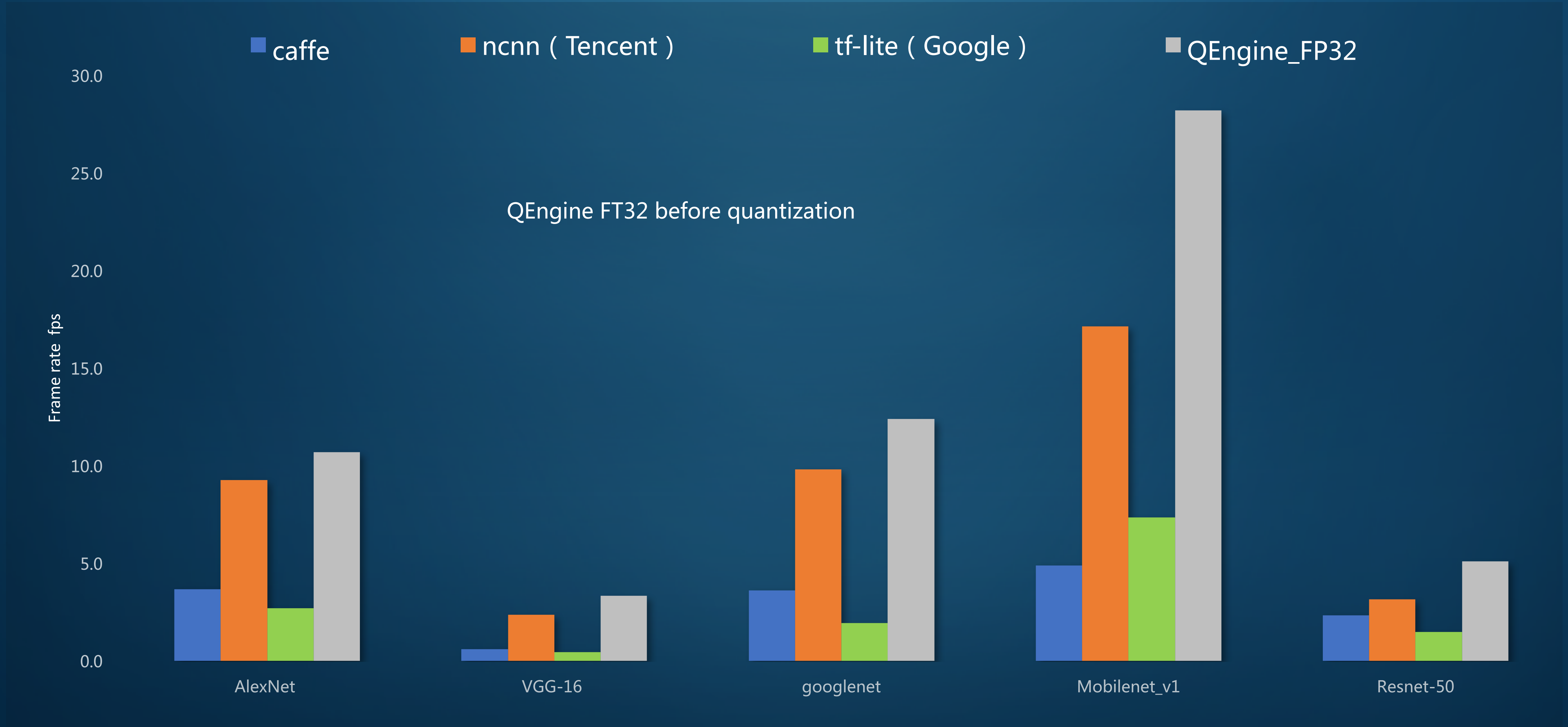


DragonBoard™ 820E (Arrow)

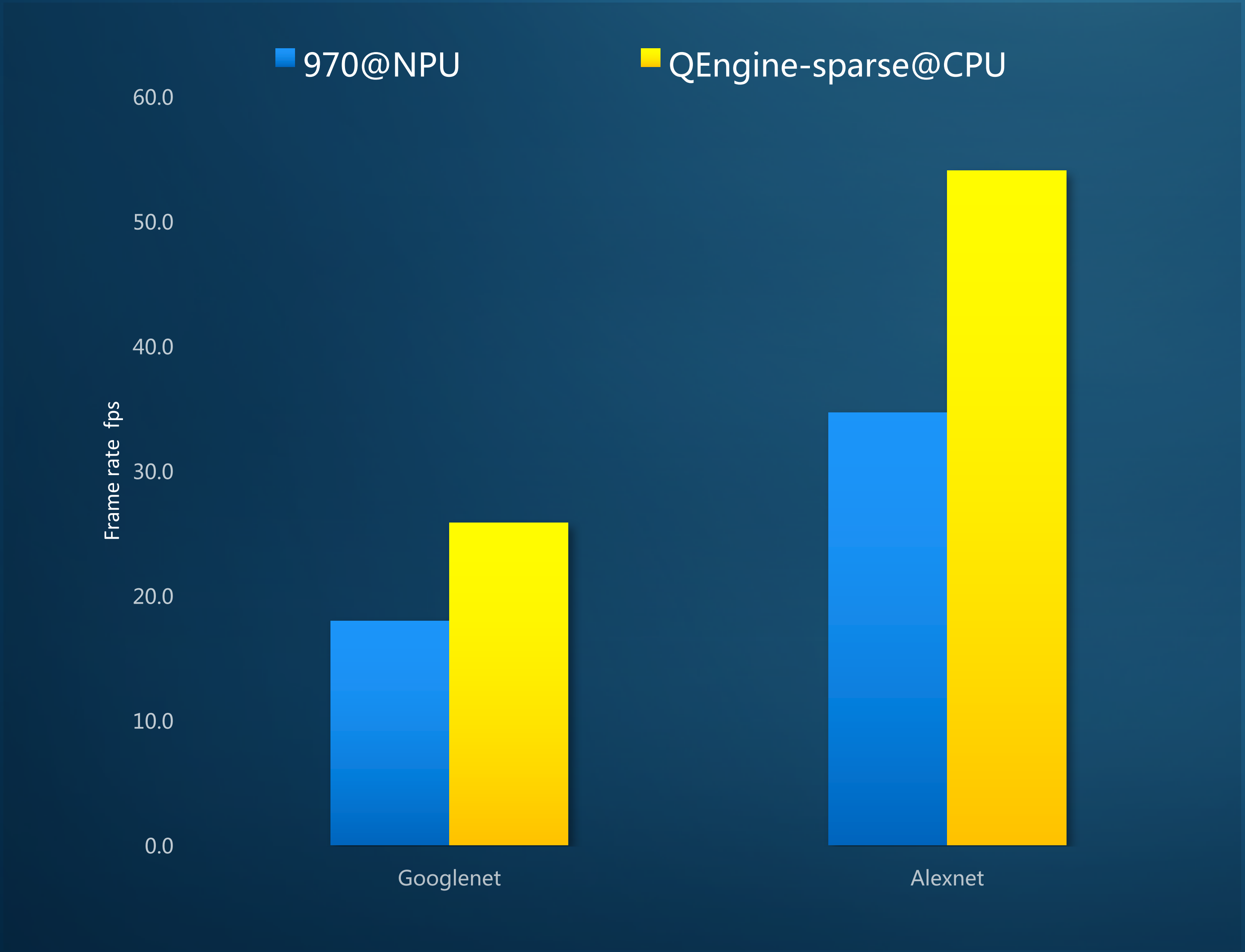
SoC	Qualcomm® Snapdragon™ 820E
CPU	custom 64-bit Kryo quad-core CPU up to 2.15GHz
RAM	Quad-channel, 16bit, 3GB PoP LPDDR4 SDRAM designed for 1866 MHz CLK

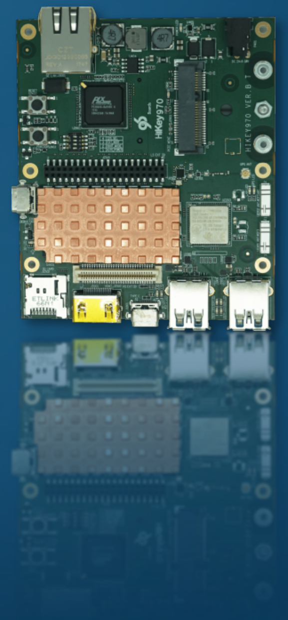


BenchMark@HUAWEI Hikey 970



QEngine@CPU VS Hikey970@NPU





HK970：

SOC Kirin970
ARM Cortex-A73 MPCore4 @up to2.36GHz, ARM Cortex-A53 MPCore4 @up to1.8GHz
NPU Cambricon1A single core

Summary

- Quantization
  - Code-based and fixed point quantization
    - Scalar, vector, product quantization
    - Uniform, non-uniform, logarithmic quantization
  - Weight, activation, gradient quantization
    - Storage, memory, computation, energy efficient
- QEngine
  - A light-weight framework for **Efficient Inference**

Thank you!

yfzhang@nlpr.ia.ac.cn