Efficient Methods and Hardware for Deep Learning

Song Han

Stanford University

May 25, 2017

Intro

Song Han
PhD Candidate
Stanford

Bill Dally
Chief Scientist
NVIDIA
Professor
Stanford

Deep Learning is Changing Our Lives

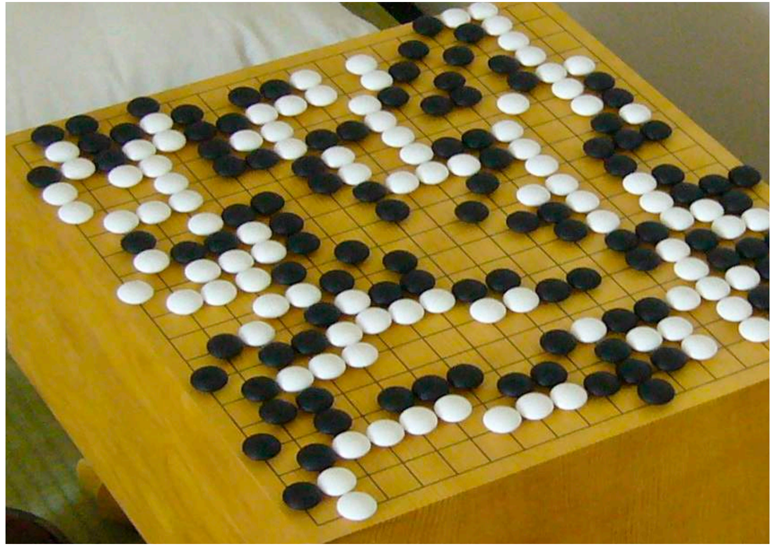
Self-Driving Car

This image is licensed under CC-BY 2.0

Machine Translation

This image is in the public domain

This image is in the public domain



3

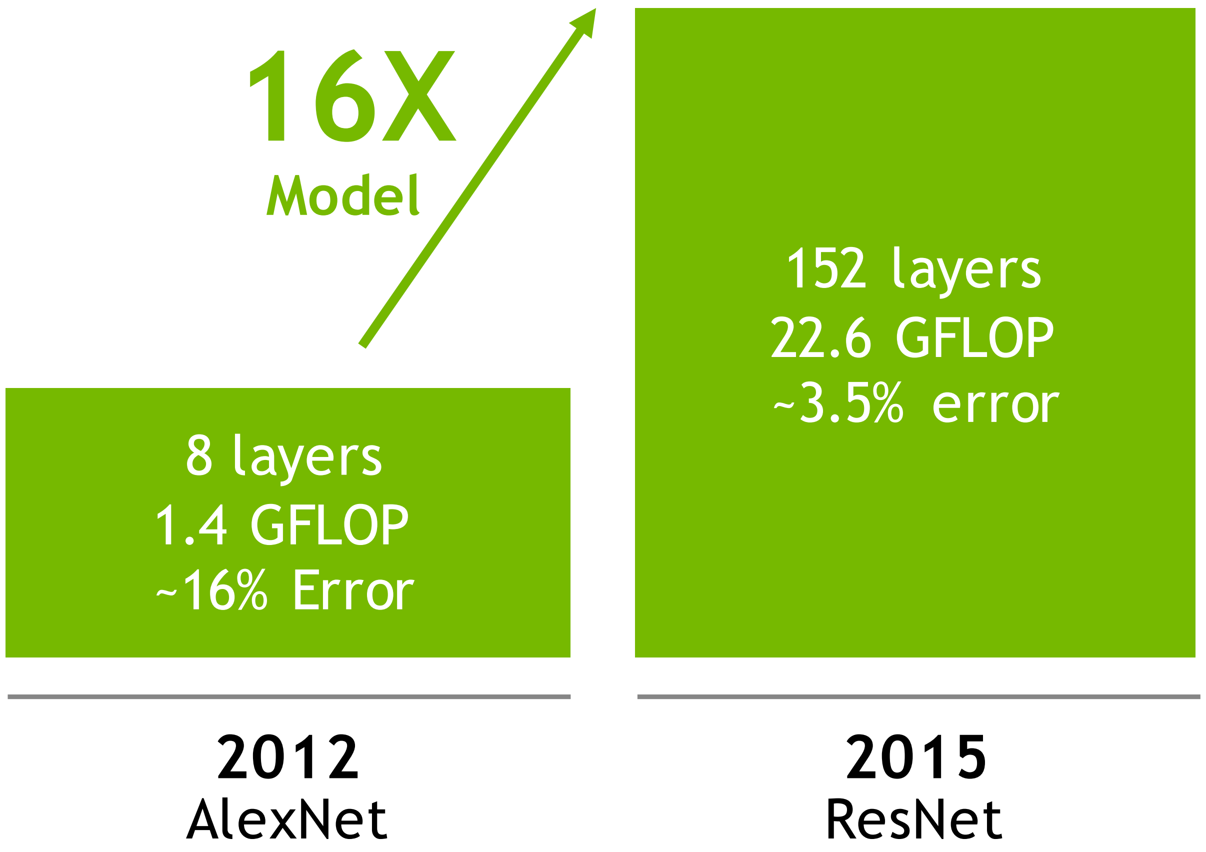
AlphaGo

This image is licensed under CC-BY 2.0

Smart Robots

Models are Getting Larger

IMAGE RECOGNITION



16X
Model

8 layers
1.4 GFLOP
~16% Error

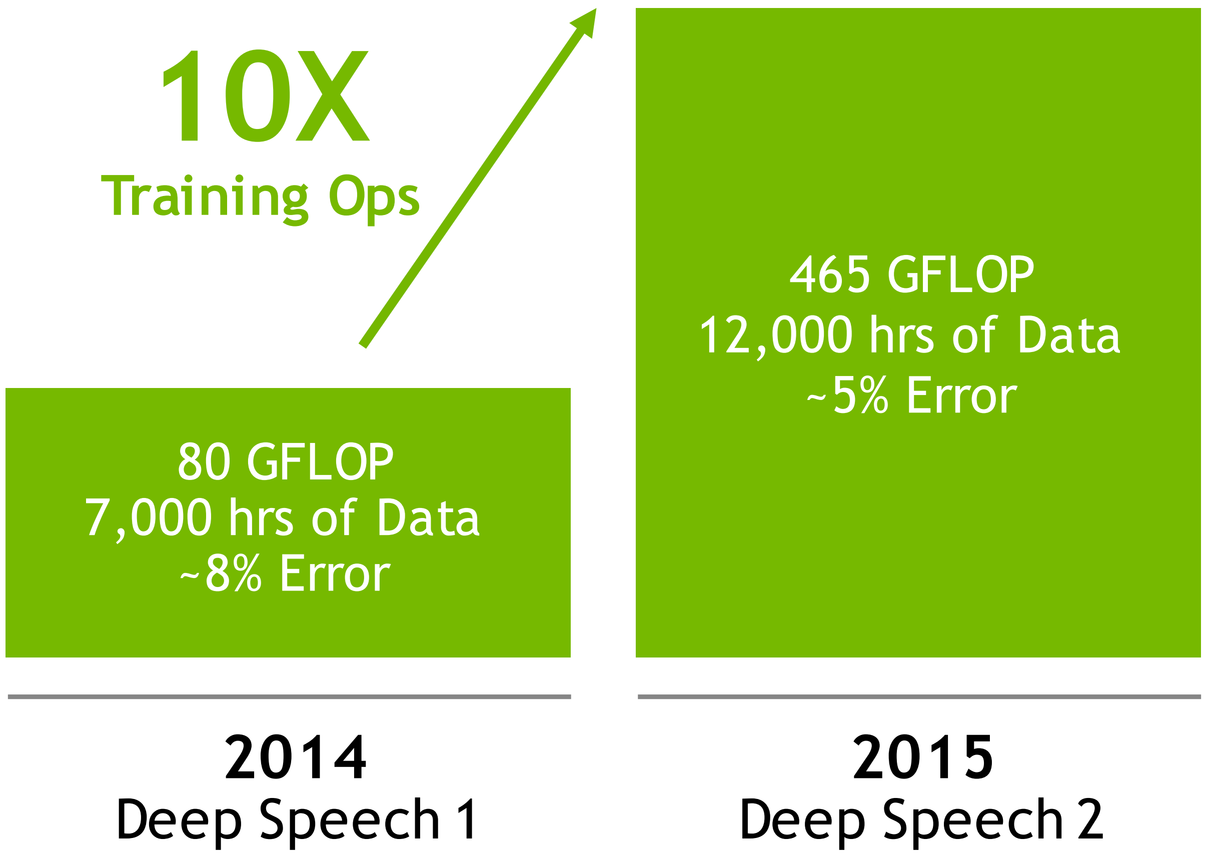
152 layers
22.6 GFLOP
~3.5% error

2012
AlexNet

2015
ResNet

Microsoft

SPEECH RECOGNITION



10X
Training Ops

80 GFLOP
7,000 hrs of Data
~8% Error

465 GFLOP
12,000 hrs of Data
~5% Error

2014
Deep Speech 1

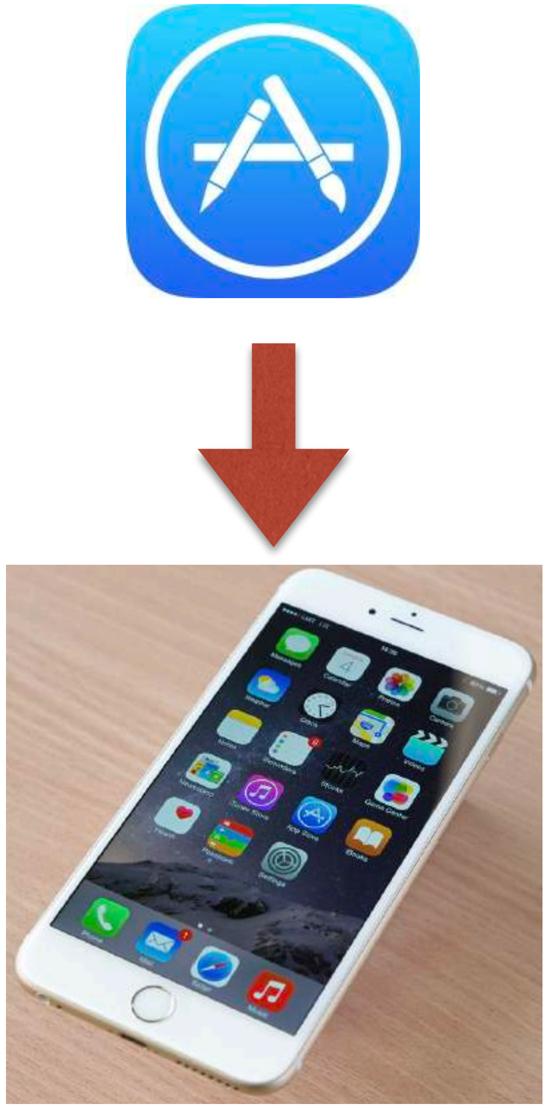
2015
Deep Speech 2

Baidu

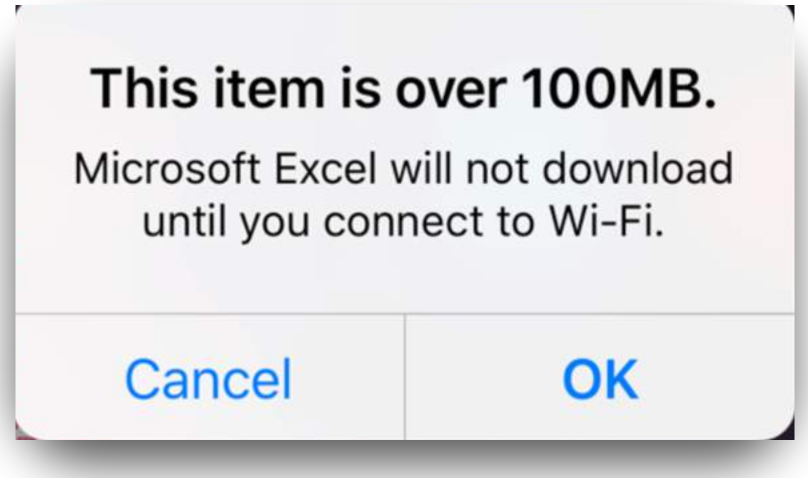
Dally, NIPS’2016 workshop on Efficient Methods for Deep Neural Networks

The first Challenge: Model Size

Hard to distribute large models through over-the-air update



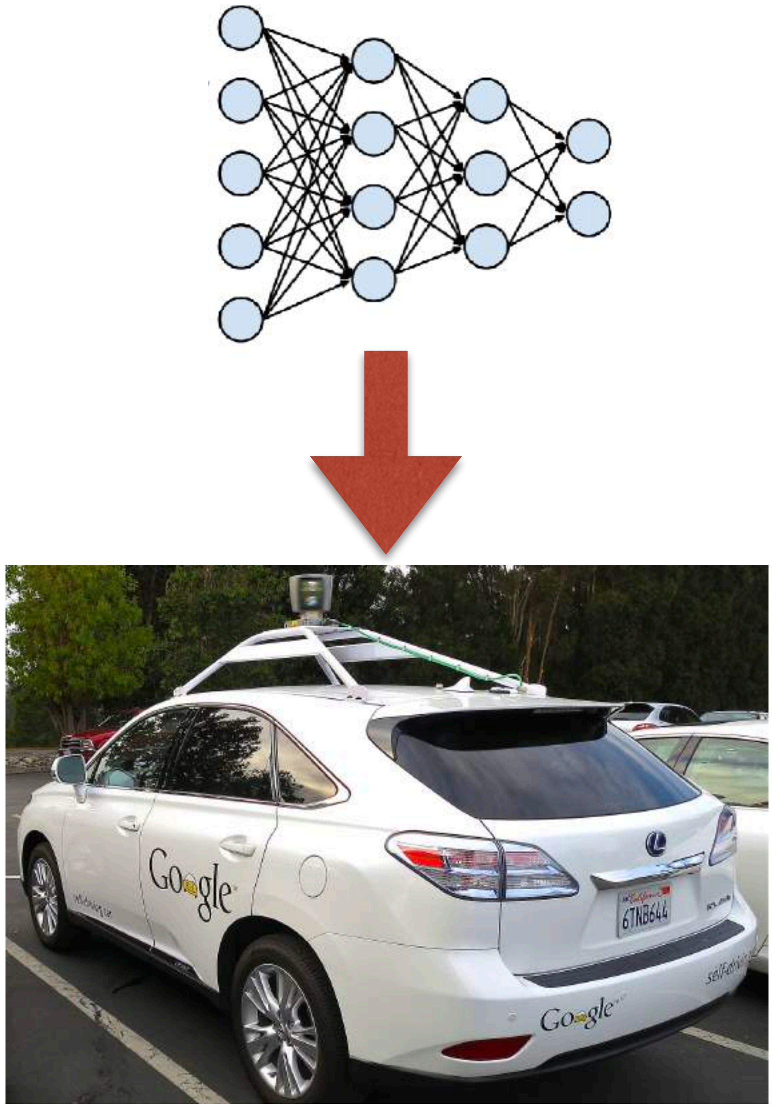
App icon is in the public domain
Phone image is licensed under CC-BY 2.0



This item is over 100MB.

Microsoft Excel will not download until you connect to Wi-Fi.

Cancel OK



This image is licensed under CC-BY 2.0

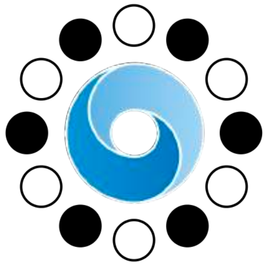
The Second Challenge: Speed

	Error rate	Training time
ResNet18:	10.76%	2.5 days
ResNet50:	7.02%	5 days
ResNet101:	6.21%	1 week
ResNet152:	6.16%	1.5 weeks

Such long training time limits ML researcher’s productivity

Training time benchmarked with fb.resnet.torch using four M40 GPUs

The Third Challenge: Energy Efficiency

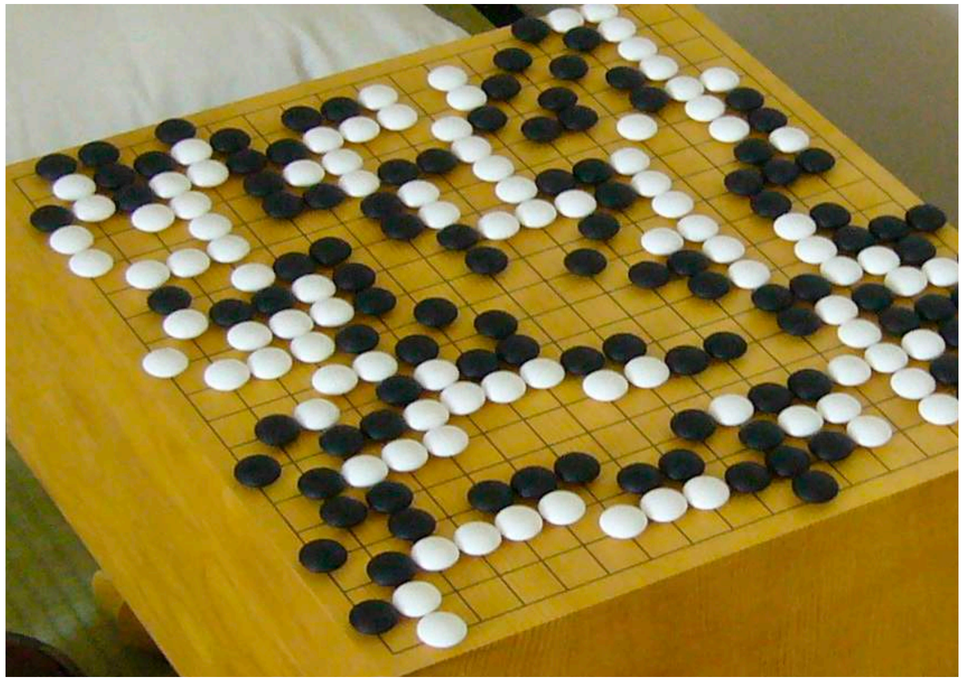


This image is in the public domain

AlphaGo: 1920 CPUs and 280 GPUs,

$3000 electric bill per game

This image is in the public domain



Phone image is licensed under CC-BY 2.0

on mobile: drains battery

on data-center: increases TCO

This image is licensed under CC-BY 2.0

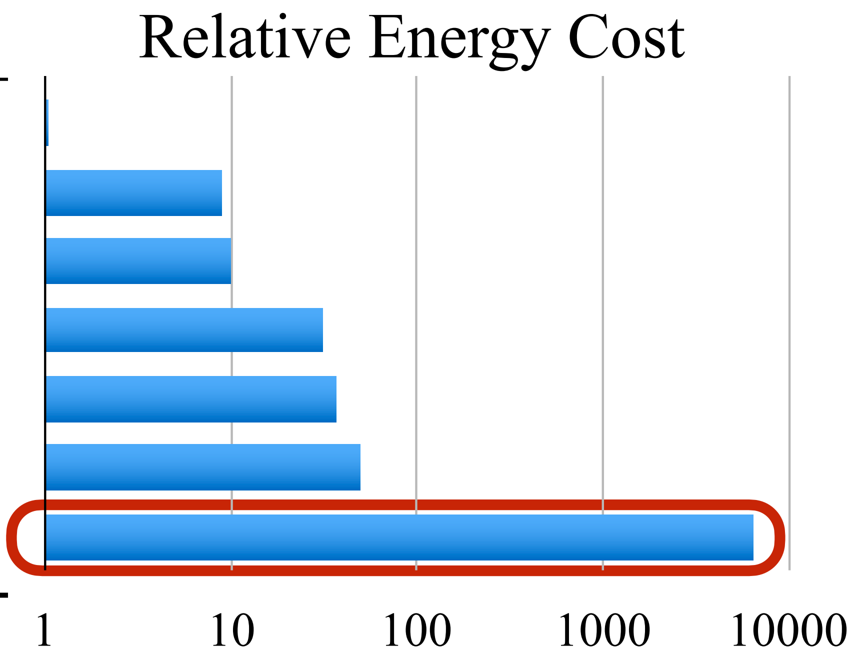
Where is the Energy Consumed?

larger model => more memory reference => more energy

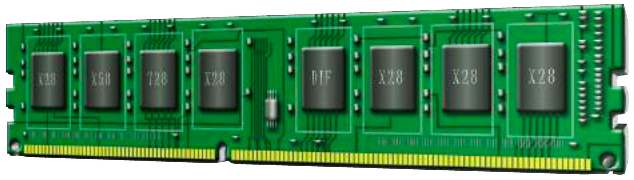
Where is the Energy Consumed?

larger model => more memory reference => more energy

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	**640**



1



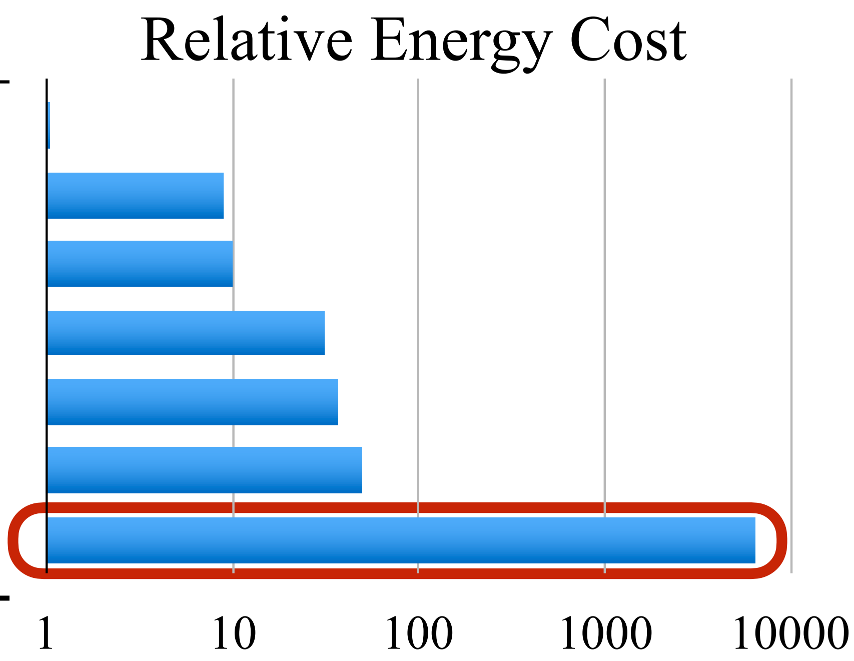
This image is in the public domain

= 1000 ✖ ✚

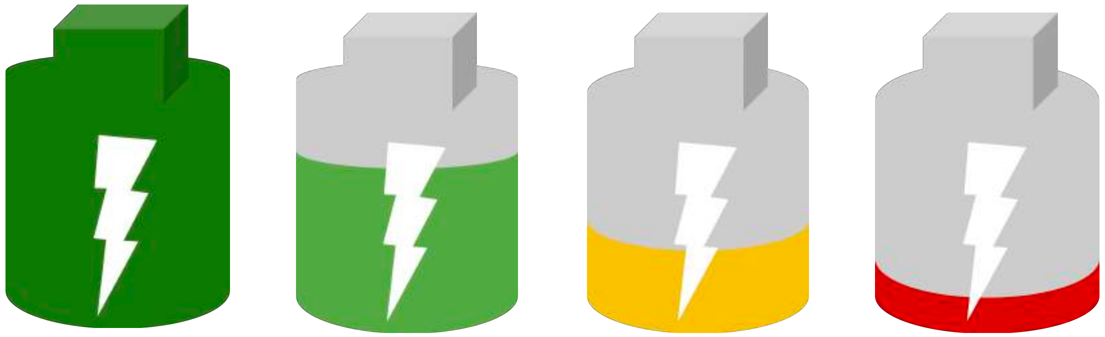
Where is the Energy Consumed?

larger model => more memory reference => more energy

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	**640**



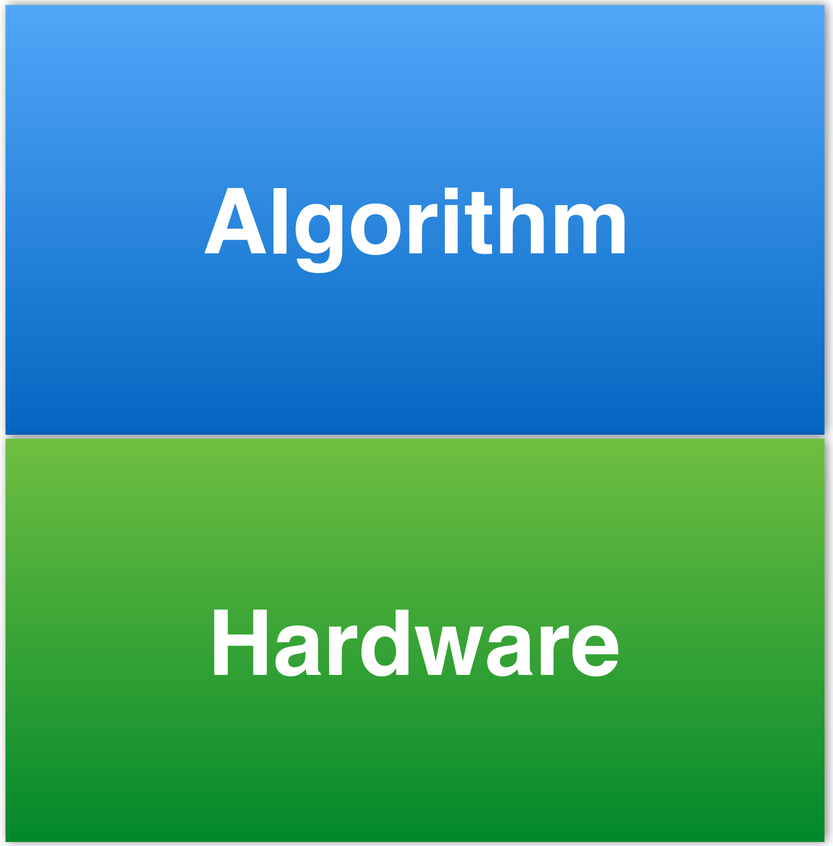
how to make deep learning more efficient?



Battery images are in the public domain
Image 1, image 2, image 2, image 4

Improve the Efficiency of Deep Learning by Algorithm-Hardware Co-Design

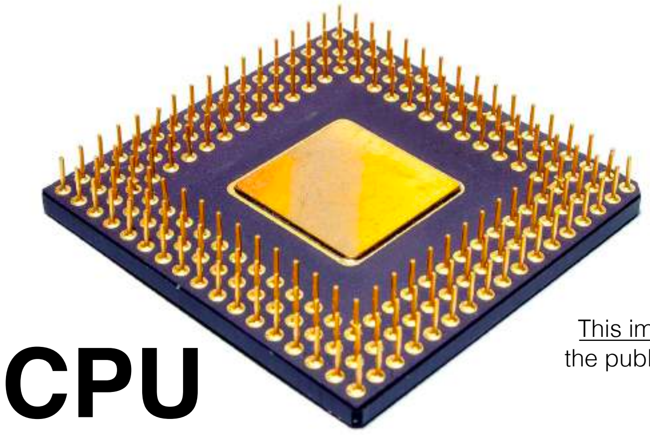
Application as a Black Box



Algorithm

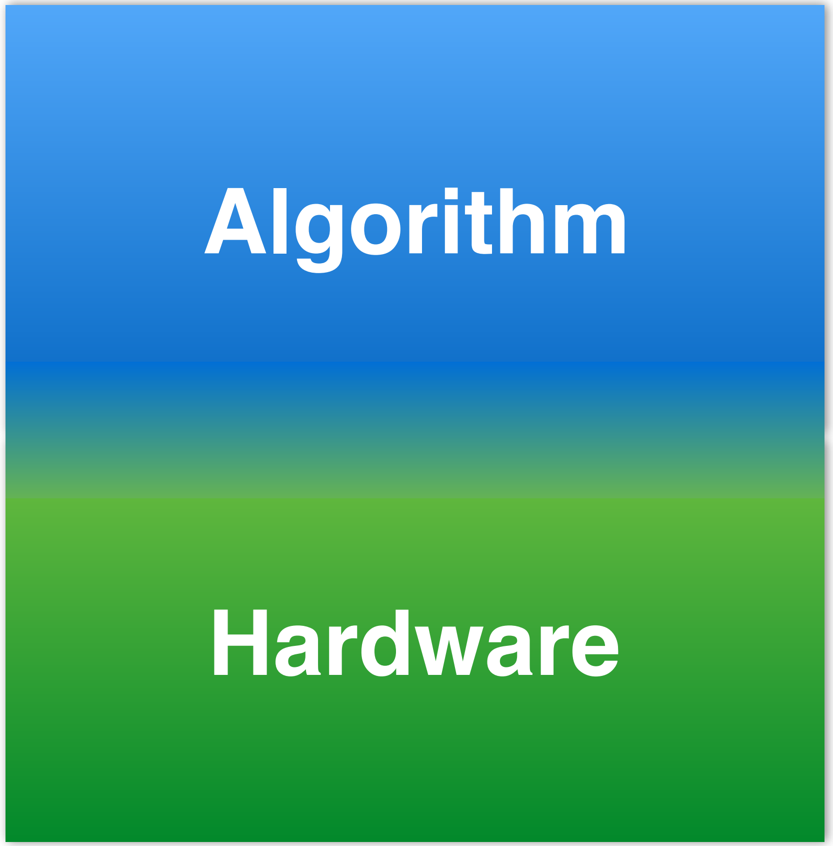
Hardware

This image is in the public domain

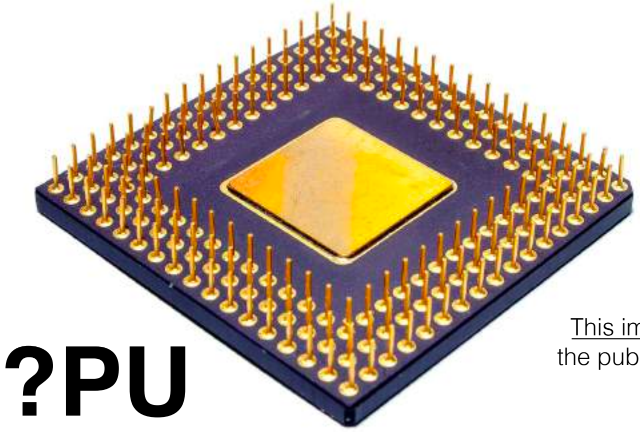


This image is in the public domain

Open the Box before Hardware Design



This image is in the public domain



This image is in the public domain

Breaks the boundary between algorithm and hardware

Agenda

Inference

Training

Agenda

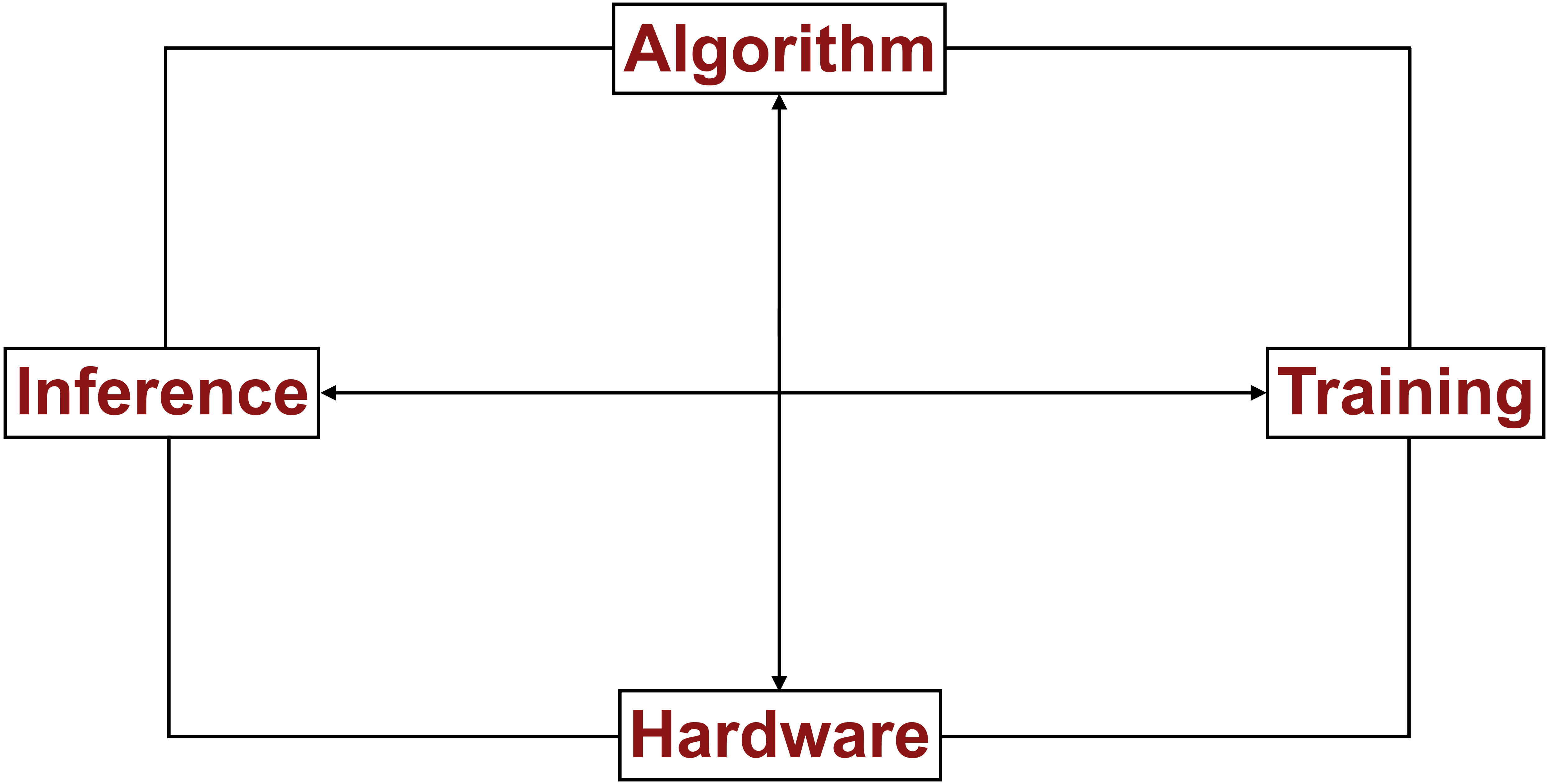
Algorithm

Inference

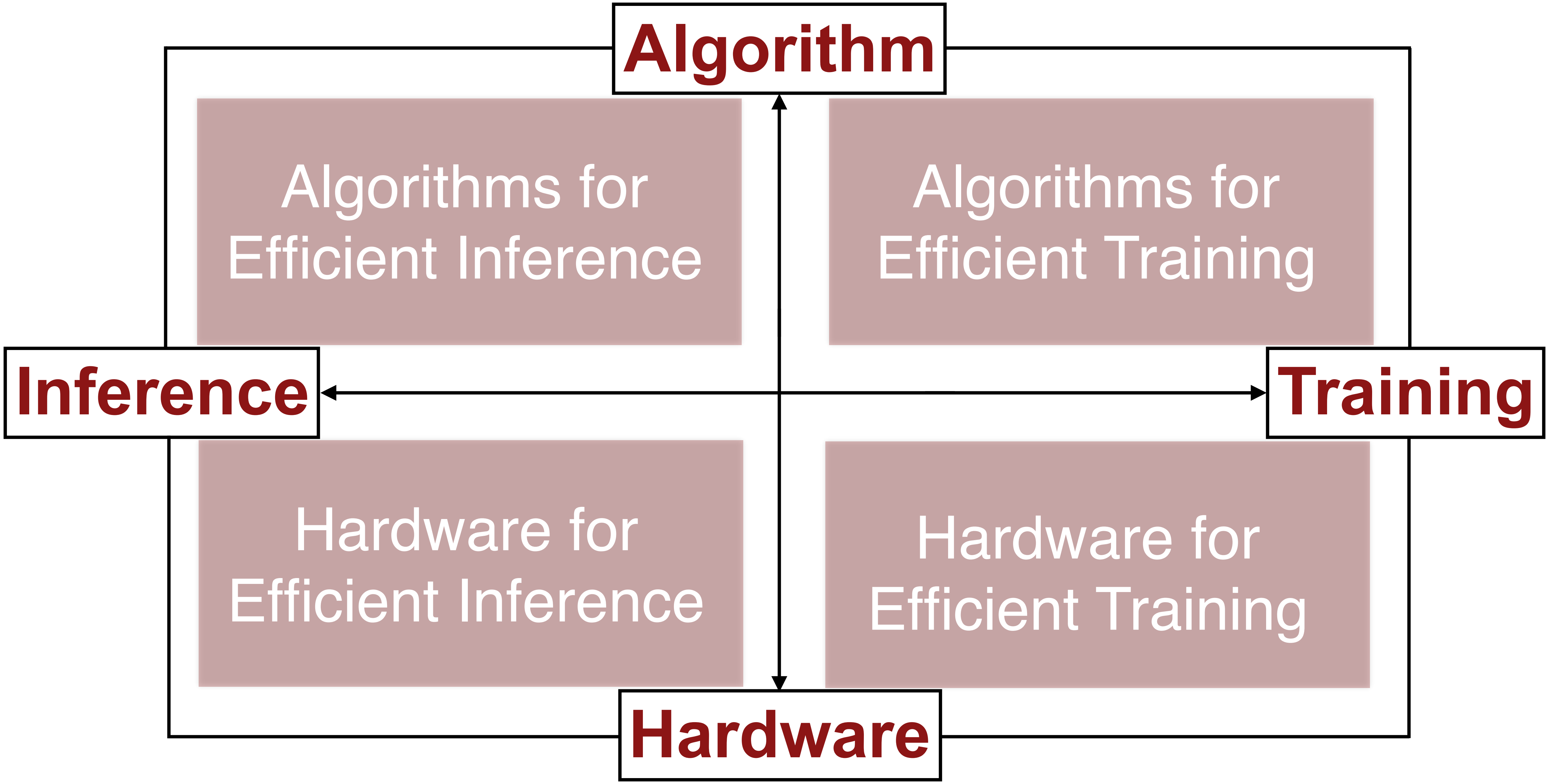
Training

Hardware

Agenda



Agenda



Algorithm

Algorithms for Efficient Inference

Algorithms for Efficient Training

Inference

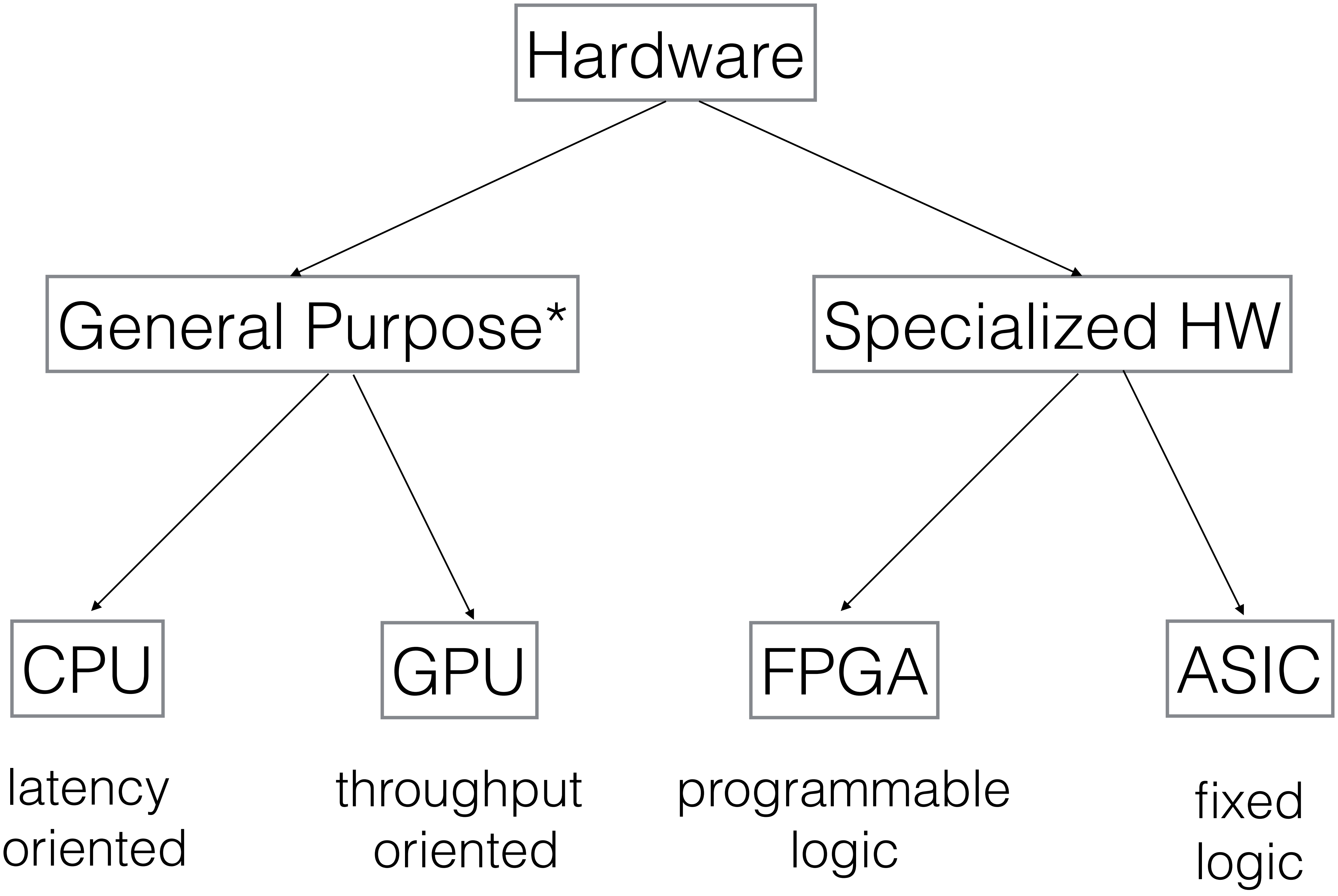
Training

Hardware for Efficient Inference

Hardware for Efficient Training

Hardware

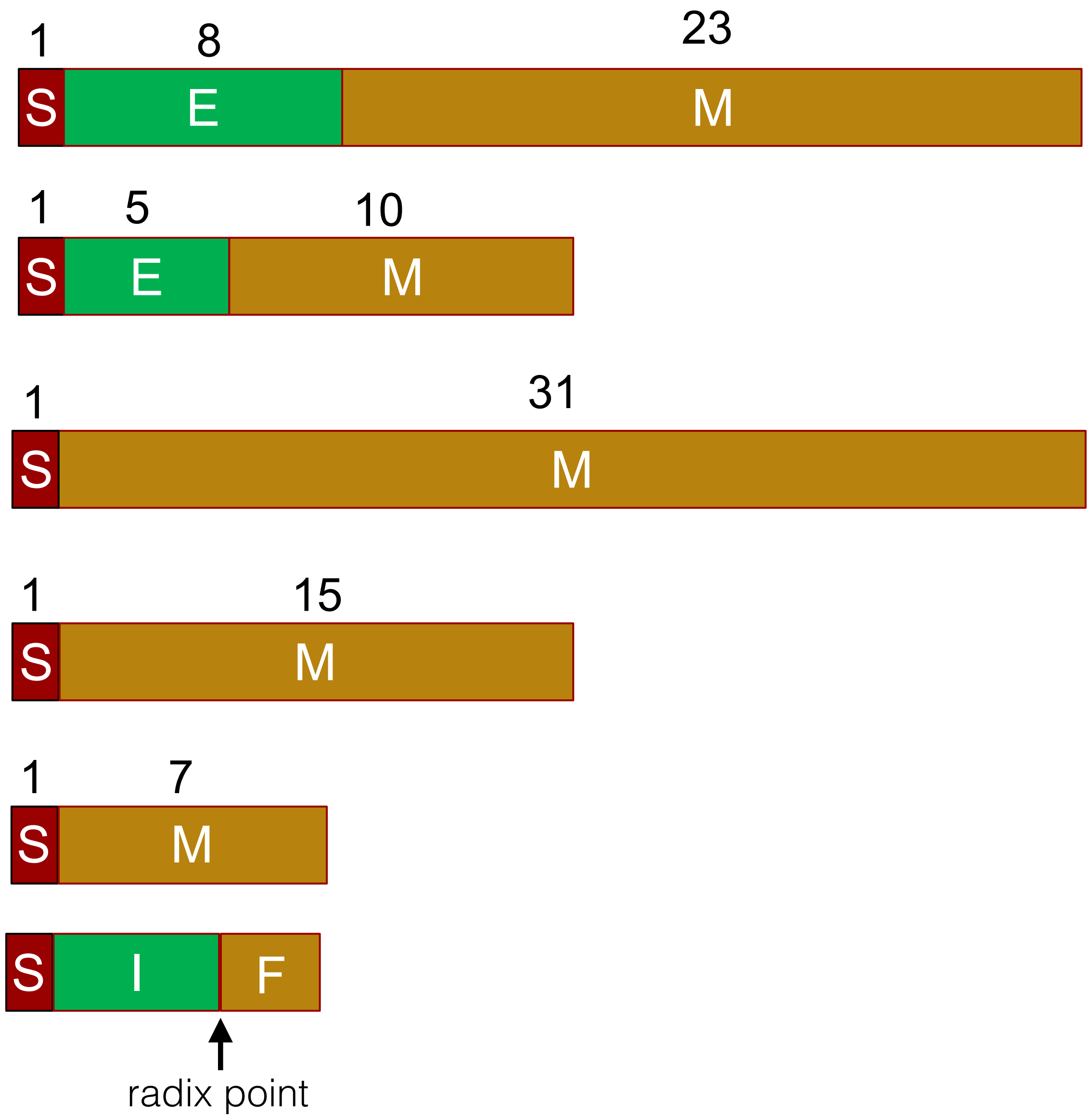
Hardware 101: the Family



* including GPGPU

Hardware 101: Number Representation

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

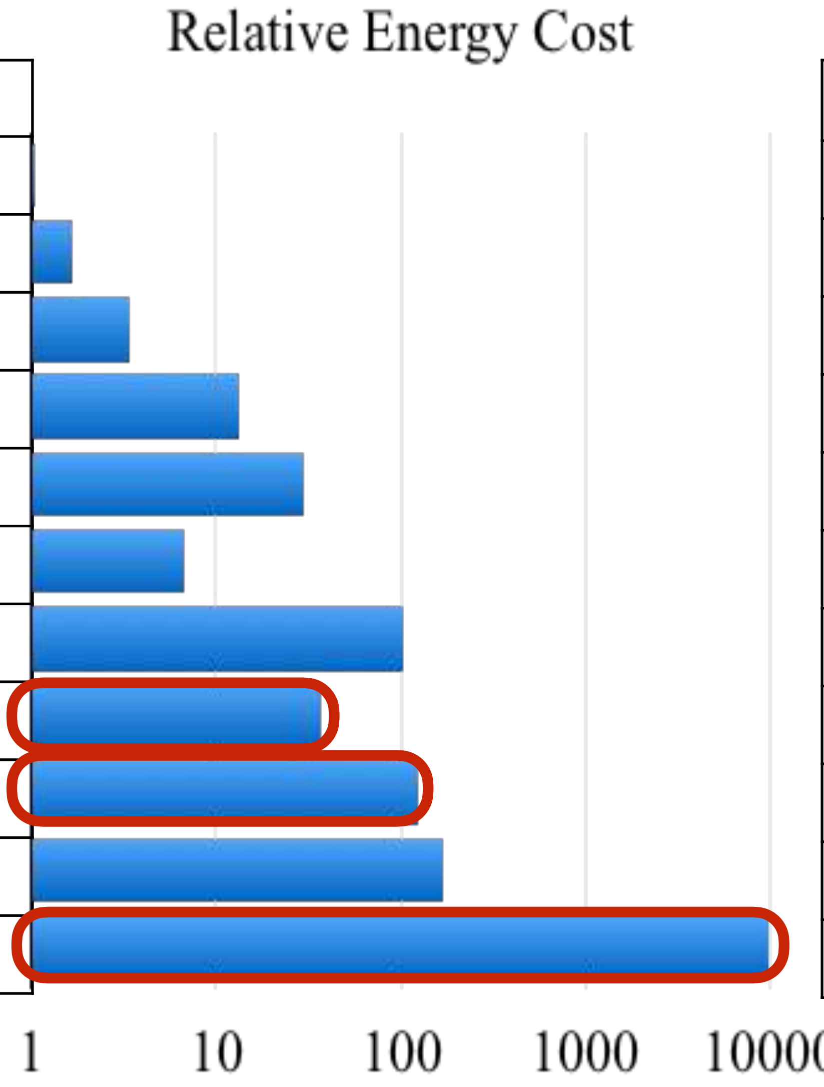


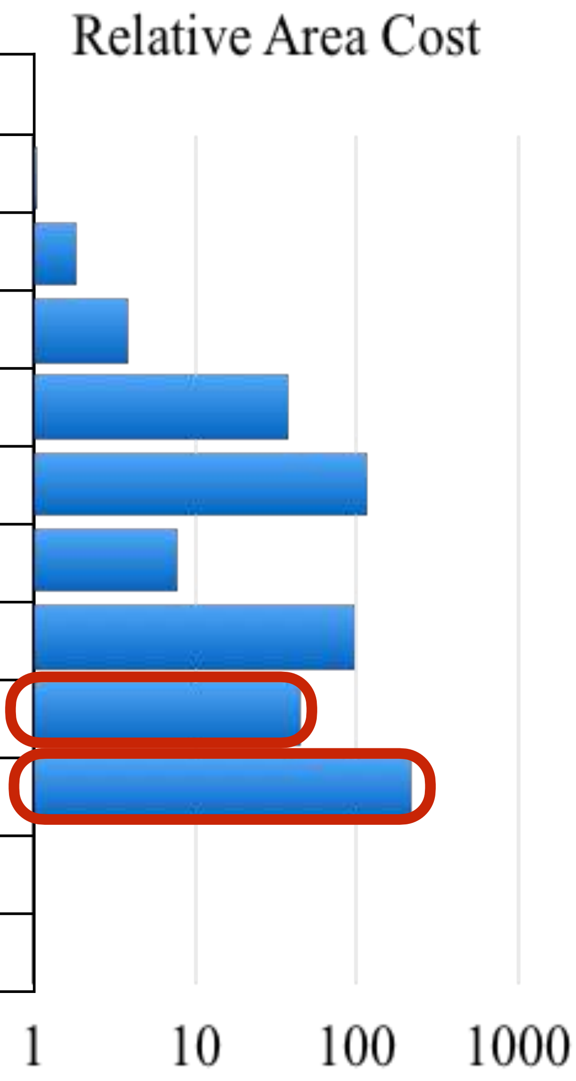
	Range	Accuracy
FP32	10^{-38} - 10^{38}	.000006%
FP16	$6x10^{-5}$ - $6x10^{4}$	.05%
Int32	0 – $2x10^{9}$	½
Int16	0 – $6x10^{4}$	½
Int8	0 – 127	½
Fixed point	-	-

Dally, High Performance Hardware for Machine Learning, NIPS’2015

Hardware 101: Number Representation

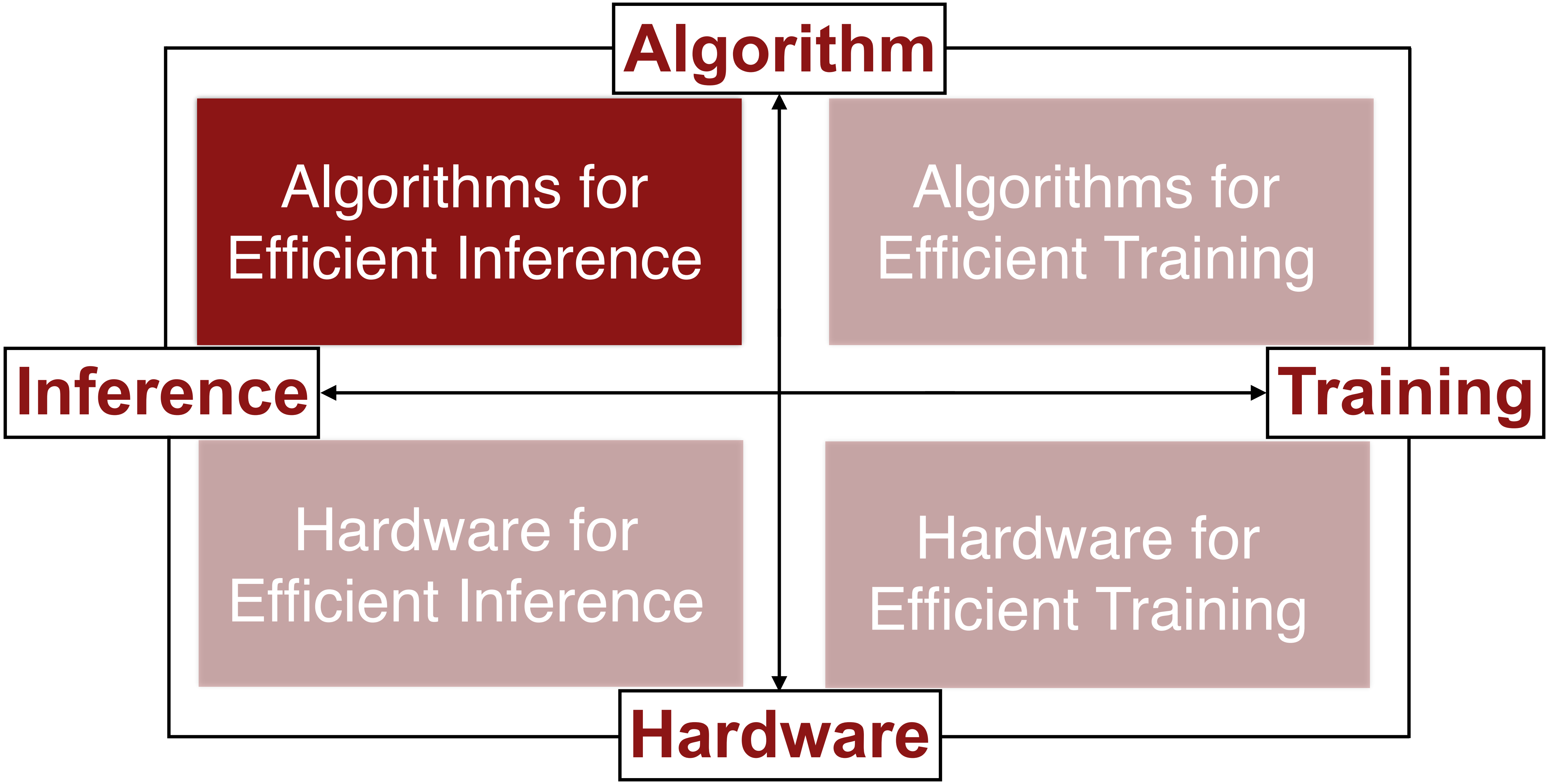
Operation:	Energy (pJ)	Area (µm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





Energy numbers are from Mark Horowitz “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014
Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.

Agenda


Algorithm
Algorithms for Efficient Inference
Algorithms for Efficient Training
Inference
Training
Hardware for Efficient Inference
Hardware for Efficient Training
Hardware

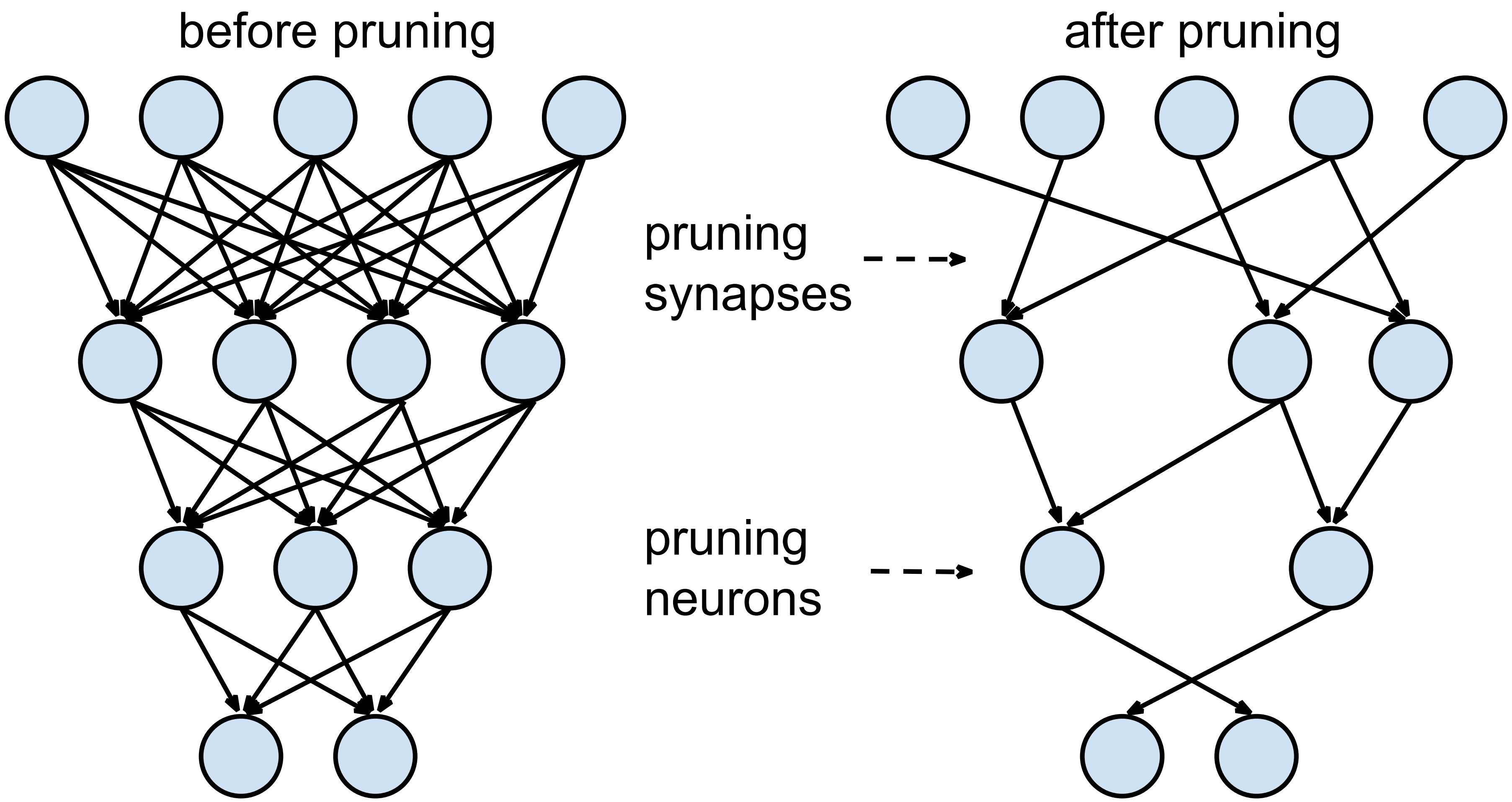
Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

Pruning Neural Networks

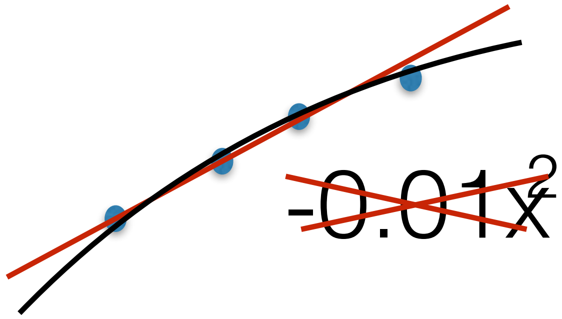


[Lecun et al. NIPS'89]

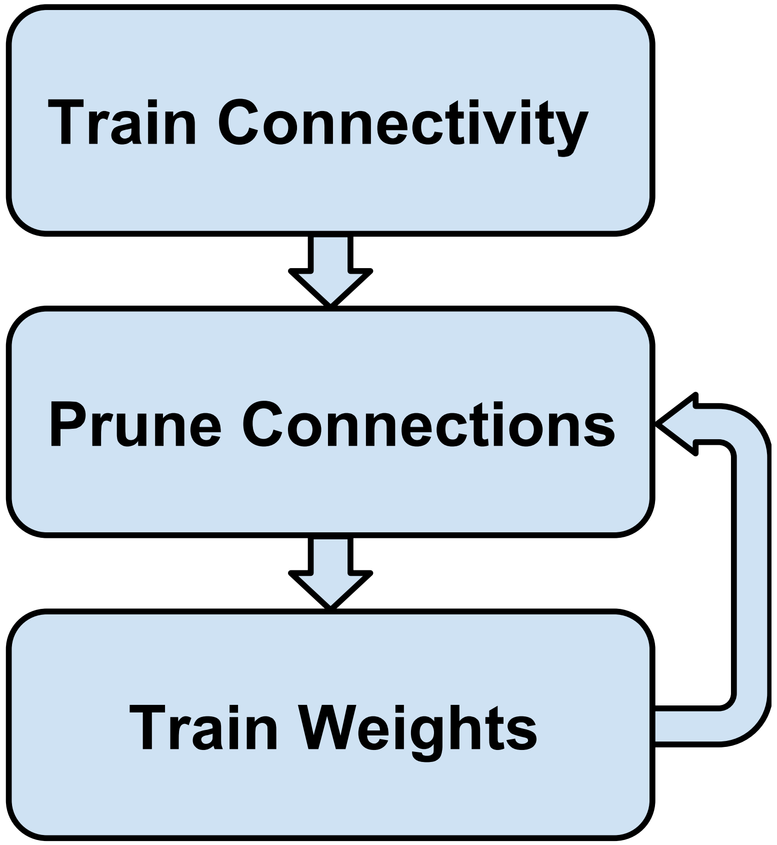
[Han et al. NIPS'15]

[Han et al. NIPS’15]

Pruning Neural Networks



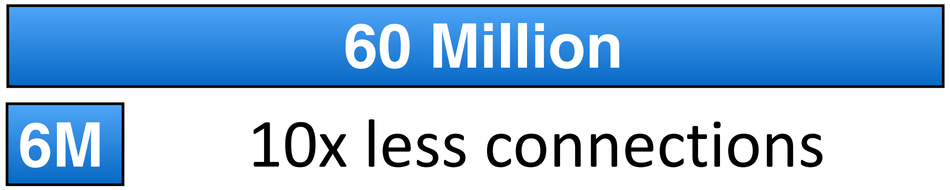
$\cancel{-0.01x^2} + x + 1$



Train Connectivity

Prune Connections

Train Weights

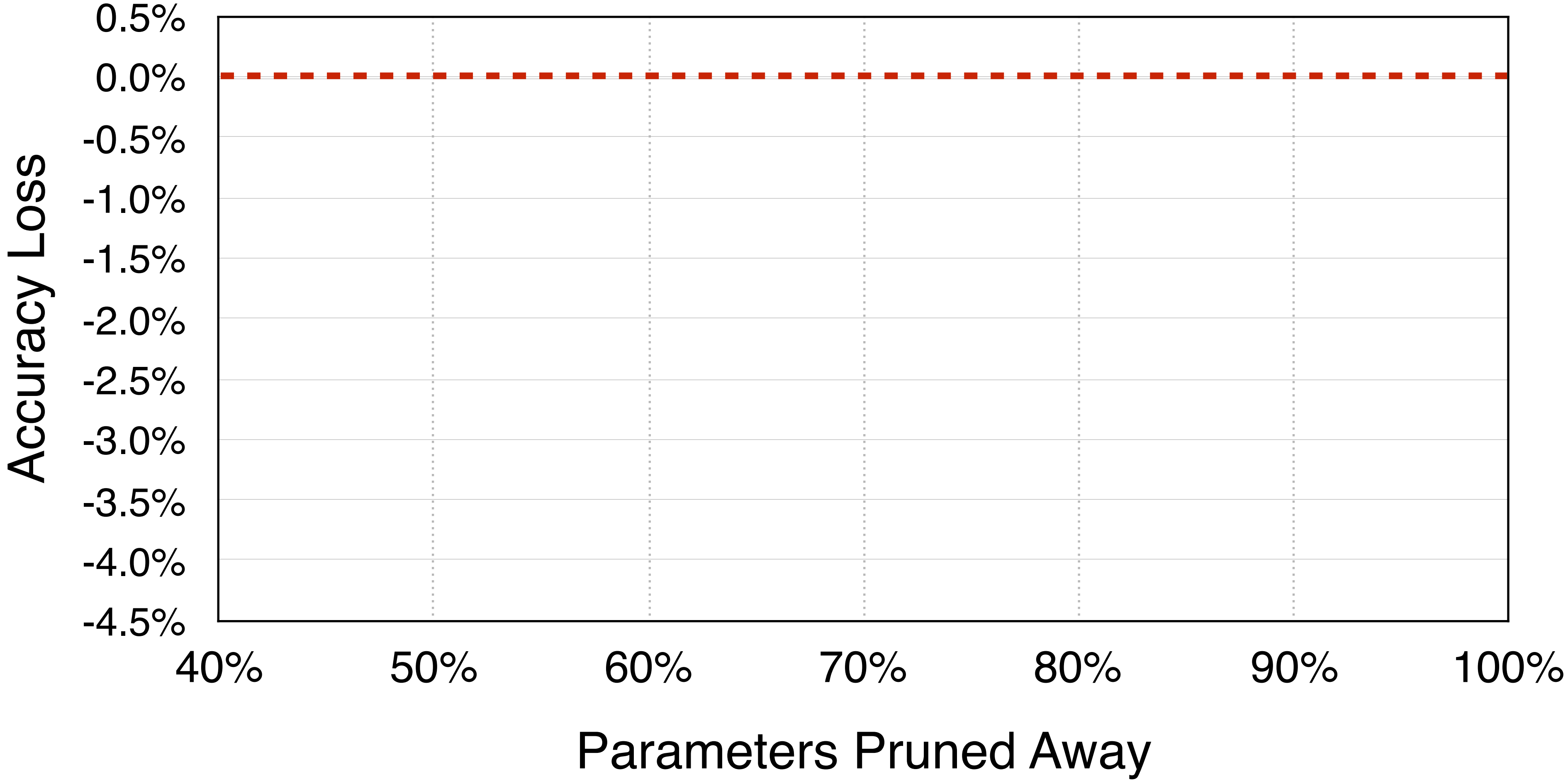


10x less connections

[Han et al. NIPS’15]

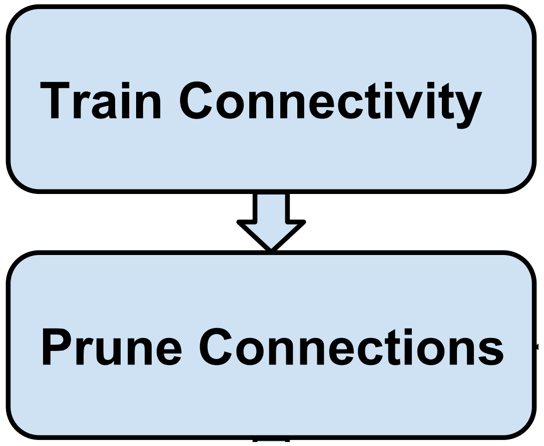
Pruning Neural Networks

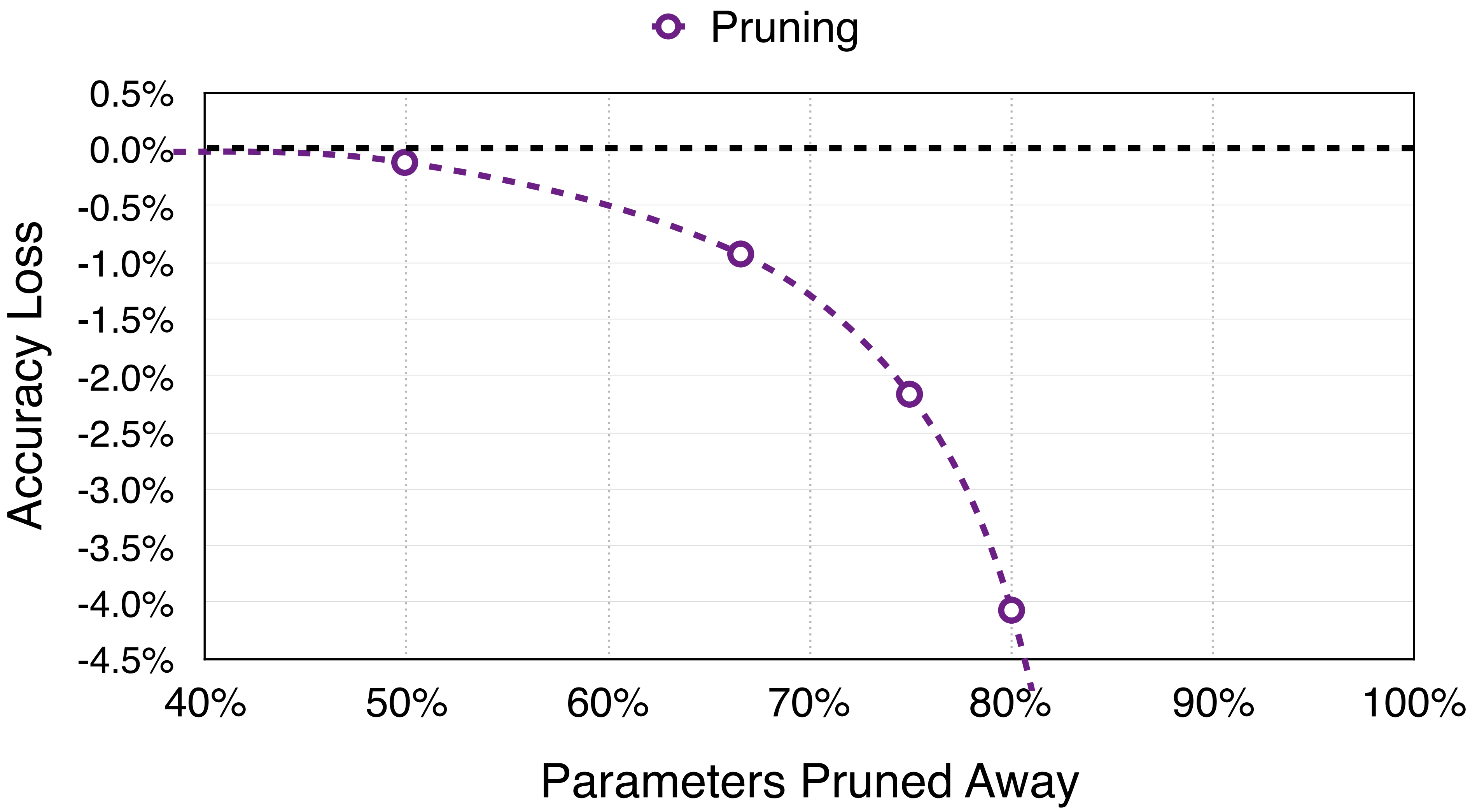
Train Connectivity



[Han et al. NIPS'15]

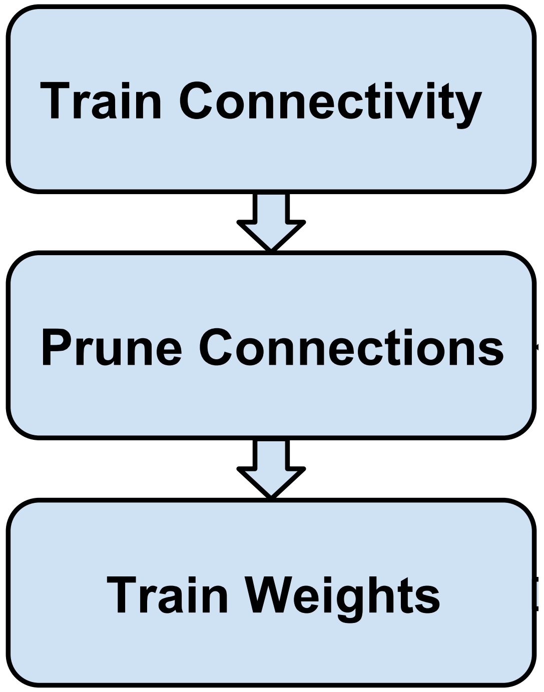
Pruning Neural Networks





[Han et al. NIPS’15]

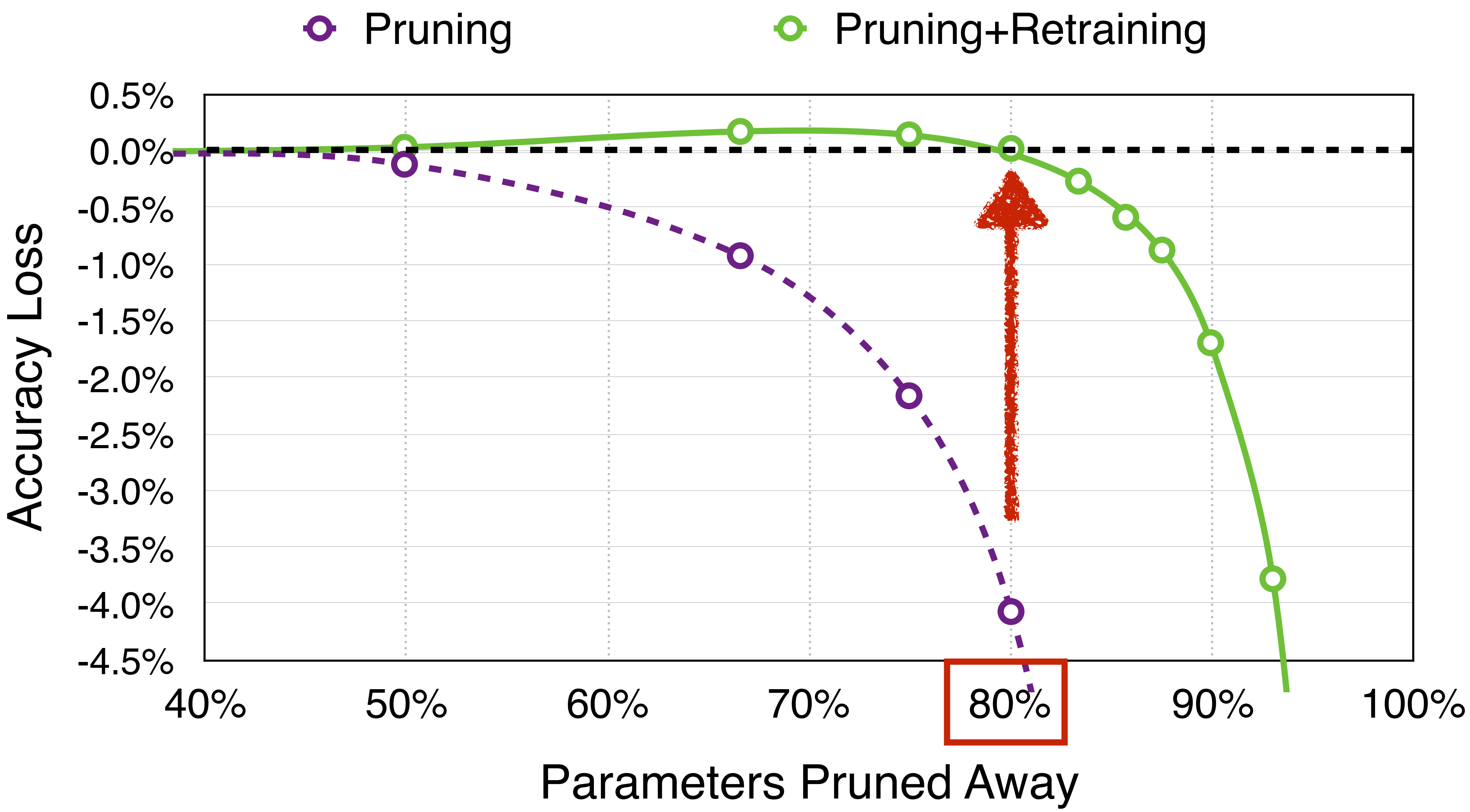
Retrain to Recover Accuracy



Train Connectivity

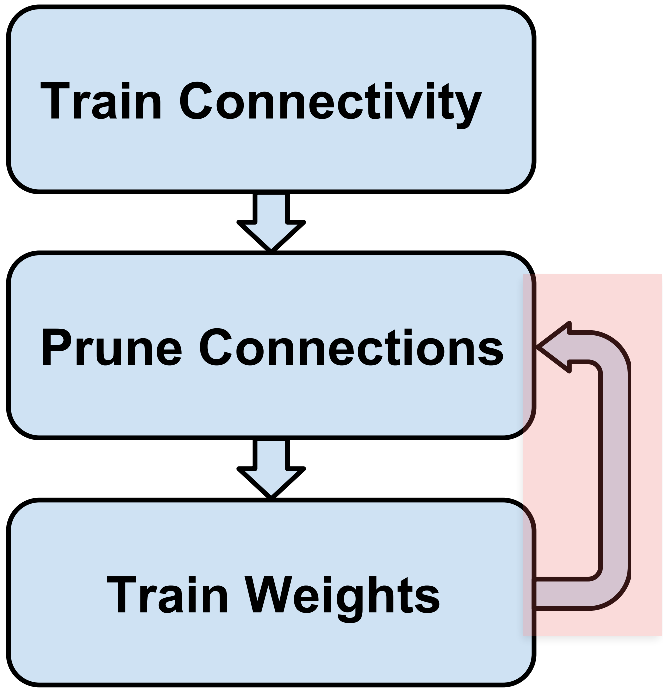
Prune Connections

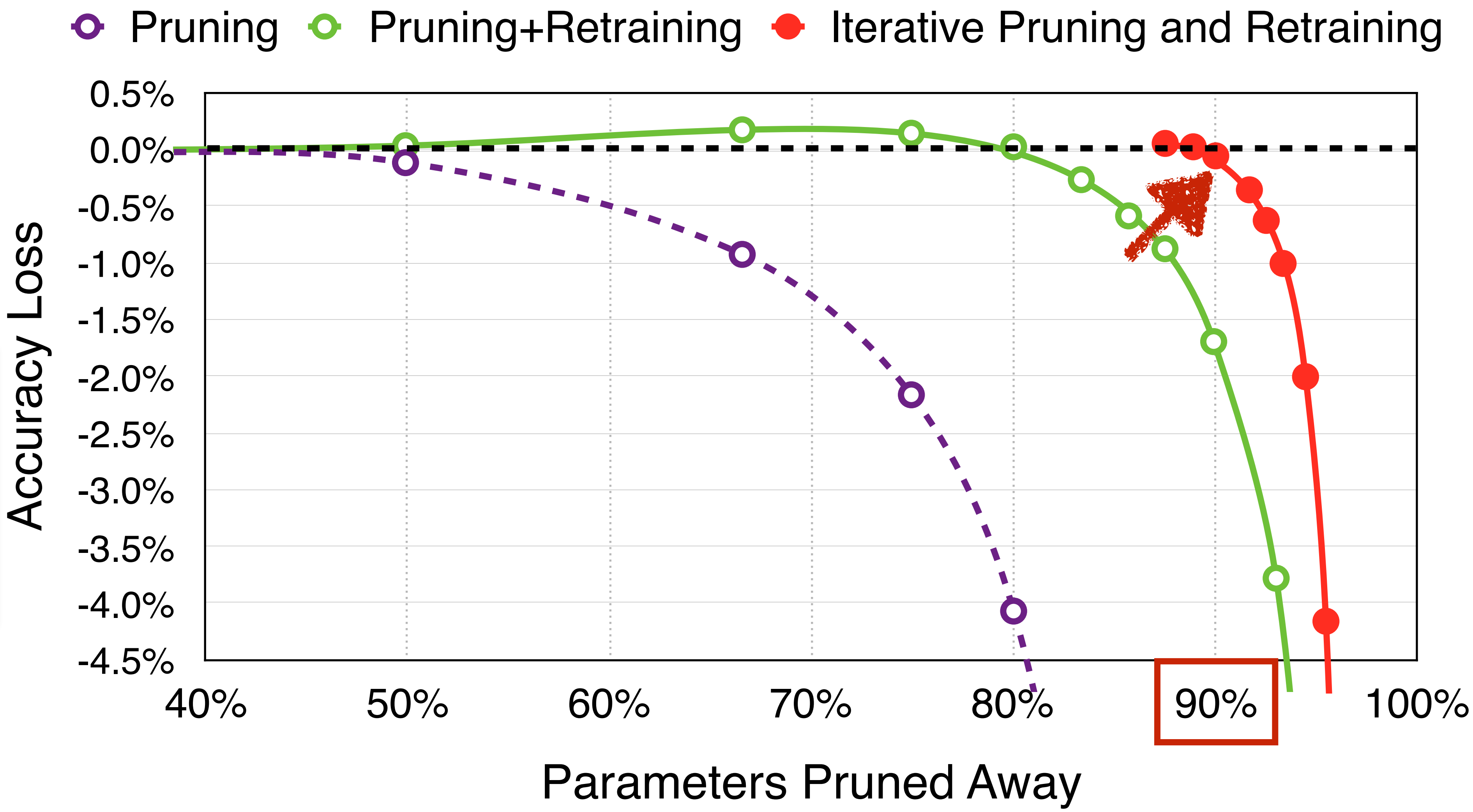
Train Weights



[Han et al. NIPS'15]

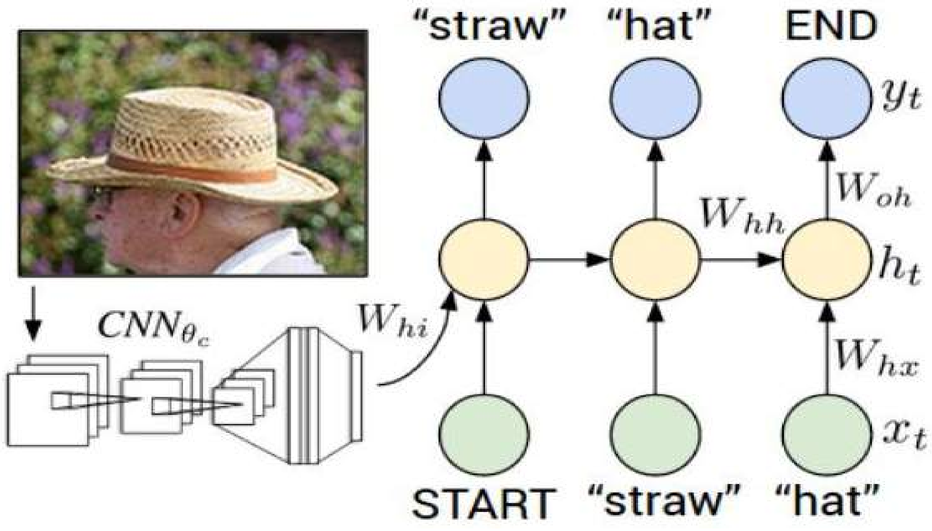
Iteratively Retrain to Recover Accuracy





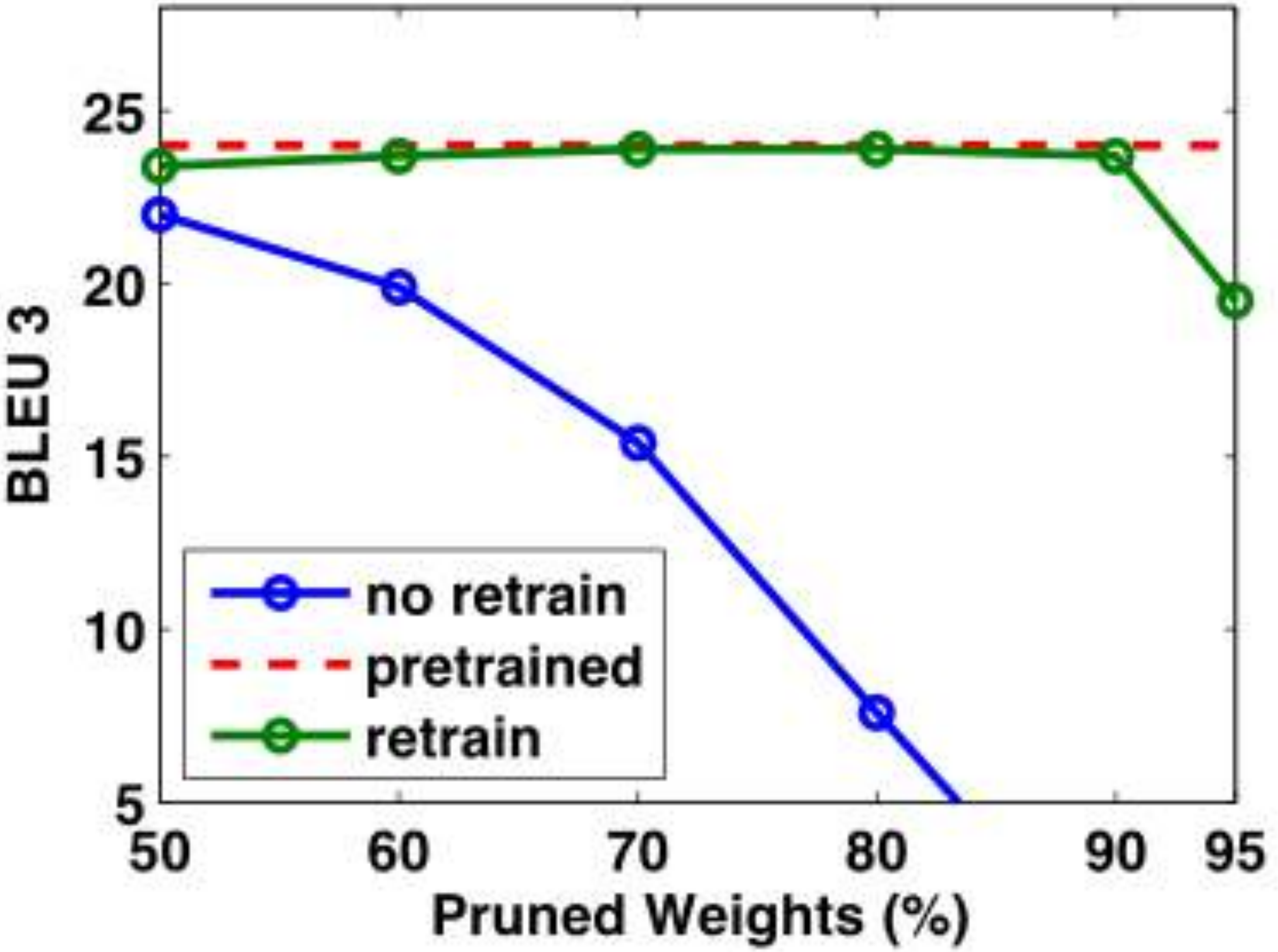
[Han et al. NIPS'15]

Pruning RNN and LSTM



*Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", 2015.

Figure copyright IEEE, 2015; reproduced for educational purposes.



[Han et al. NIPS'15]

Pruning RNN and LSTM

90%

- **Original**: a basketball player in a white uniform is playing with a ball
- **Pruned 90%**: a basketball player in a white uniform is playing with a basketball

90%

- **Original** : a brown dog is running through a grassy field
- **Pruned 90%**: a brown dog is running through a grassy area

90%

- **Original** : a man is riding a surfboard on a wave
- **Pruned 90%**: a man in a wetsuit is riding a wave on a beach

95%

- **Original** : a soccer player in red is running in the field
- **Pruned 95%**: a man in a red shirt and black and white black shirt is running through a field

Pruning Happens in Human Brain

1000 Trillion Synapses

50 Trillion Synapses

500 Trillion Synapses

This image is in the public domain

Newborn

This image is in the public domain

1 year old

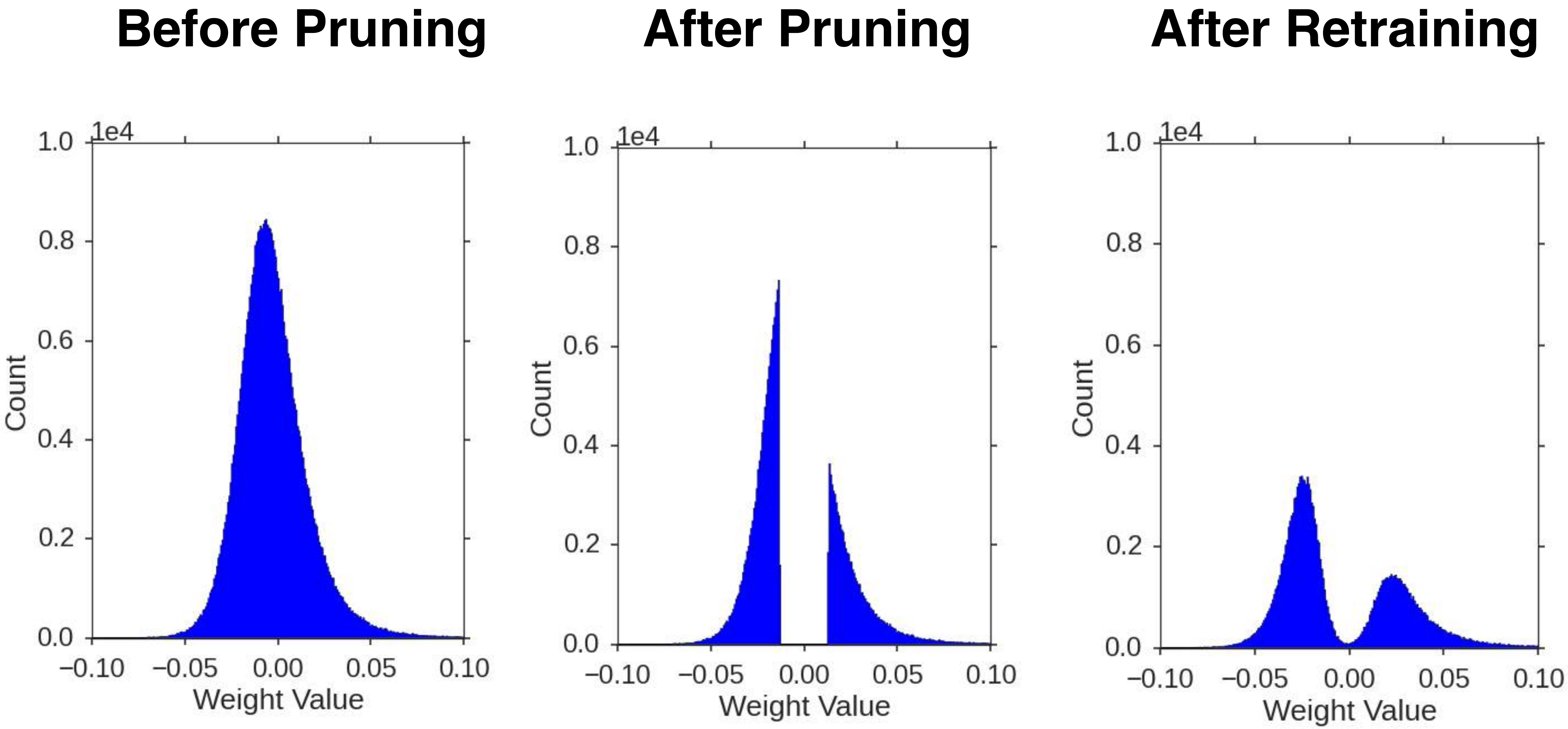
This image is in the public domain

Adolescent

Christopher A Walsh. Peter Huttenlocher (1931-2013). Nature, 502(7470):172–172, 2013.

[Han et al. NIPS’15]

Pruning Changes Weight Distribution



Before Pruning

After Pruning

After Retraining

Conv5 layer of Alexnet. Representative for other network layers as well.

Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

[Han et al. ICLR’16]

Trained Quantization

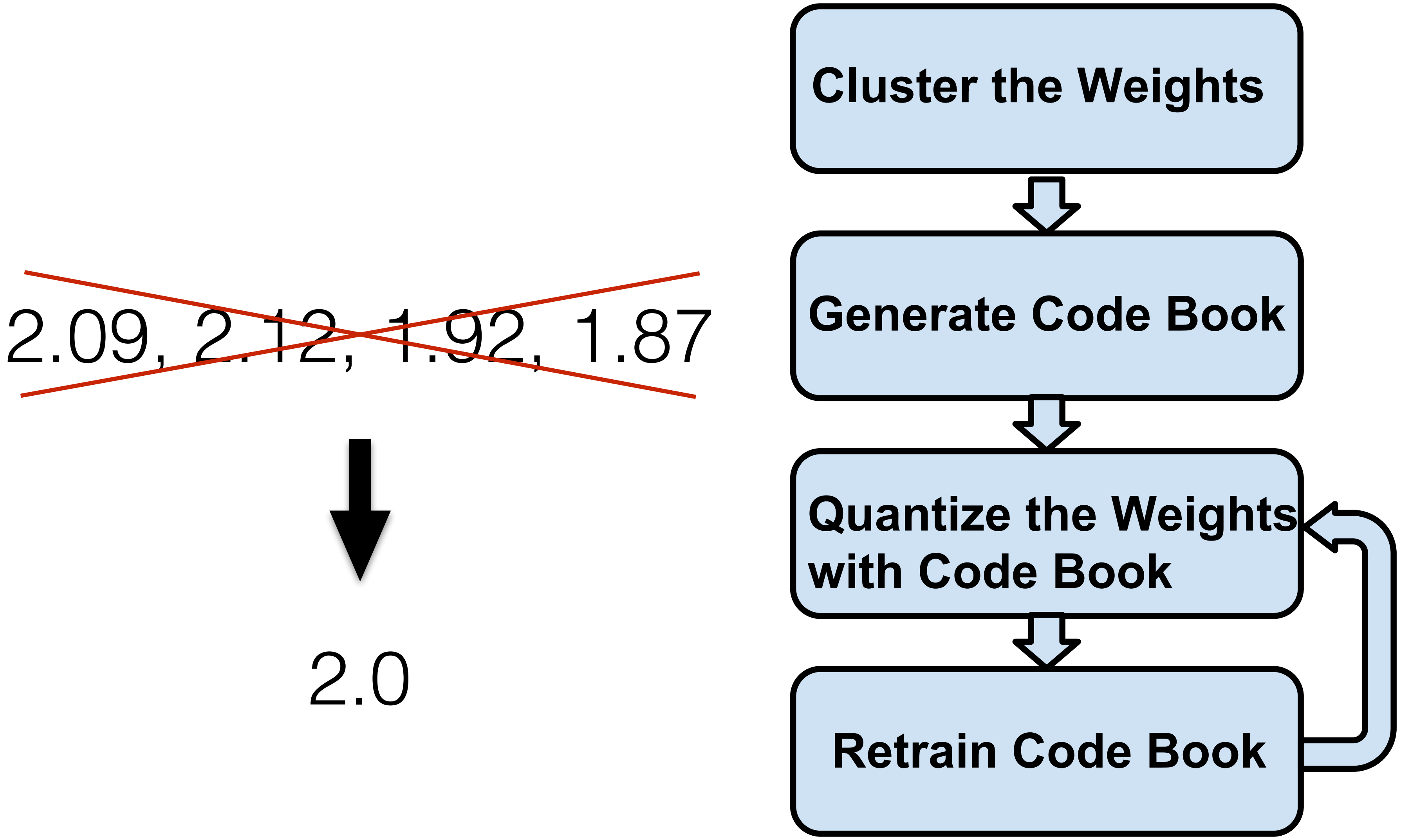
~~2.09, 2.12, 1.92, 1.87~~

↓

2.0

[Han et al. ICLR’16]

Trained Quantization



~~2.09, 2.12, 1.92, 1.87~~

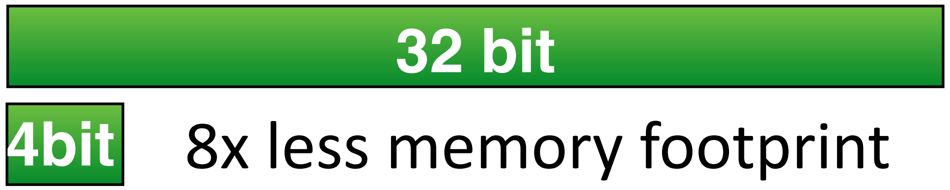
2.0

Cluster the Weights

Generate Code Book

Quantize the Weights with Code Book

Retrain Code Book

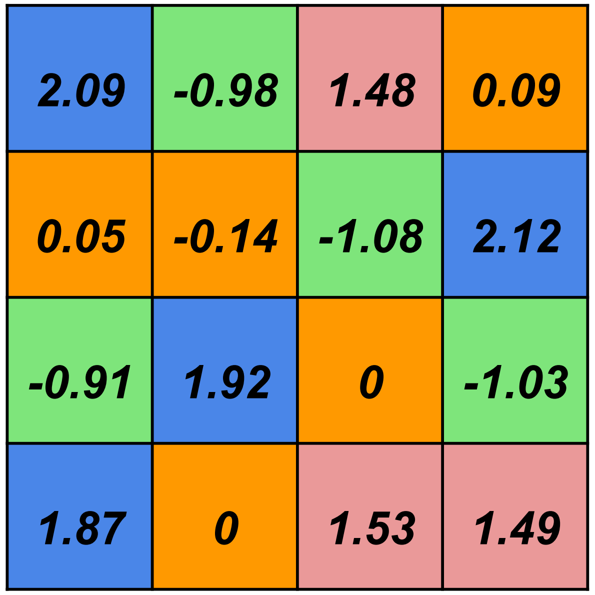


8x less memory footprint

[Han et al. ICLR’16]

Trained Quantization

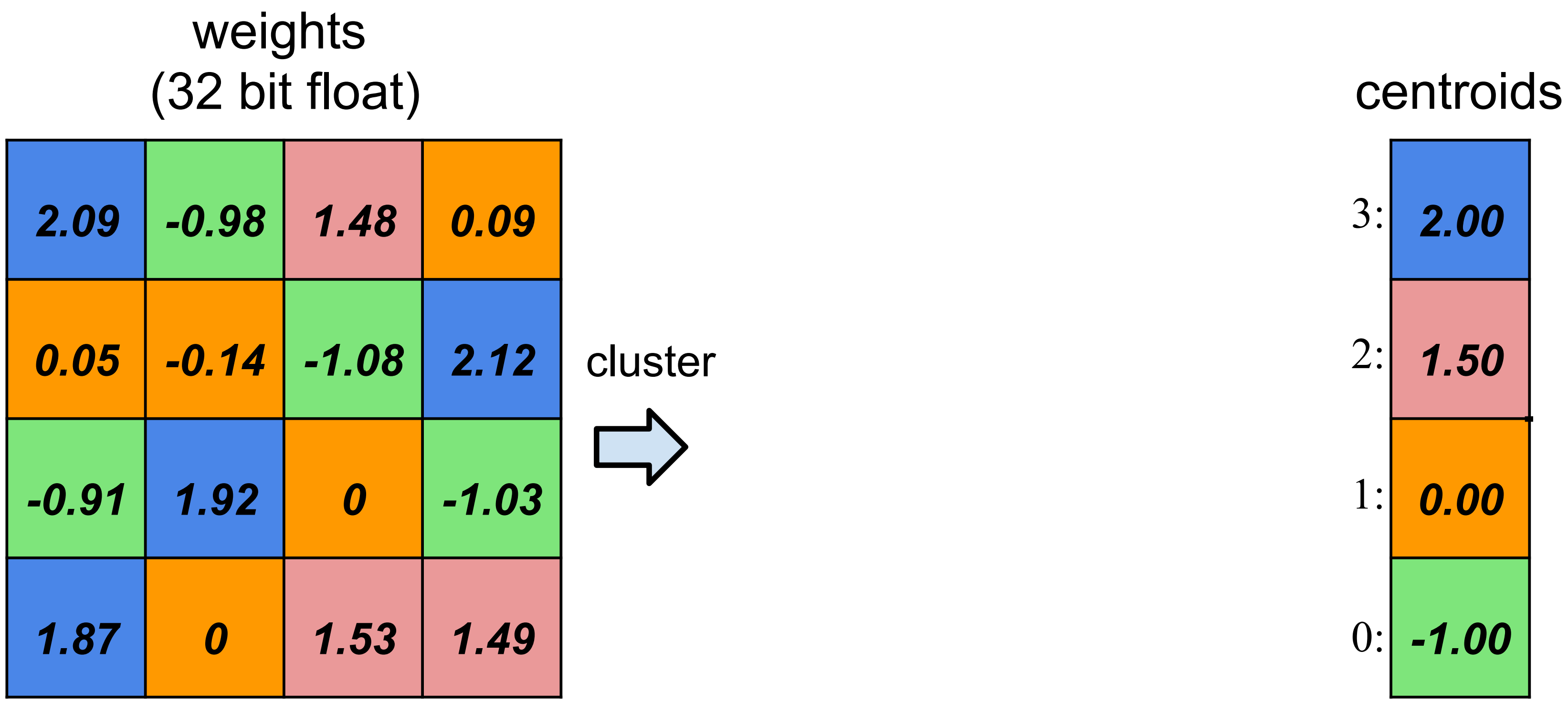
weights
(32 bit float)



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

[Han et al. ICLR’16]

Trained 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

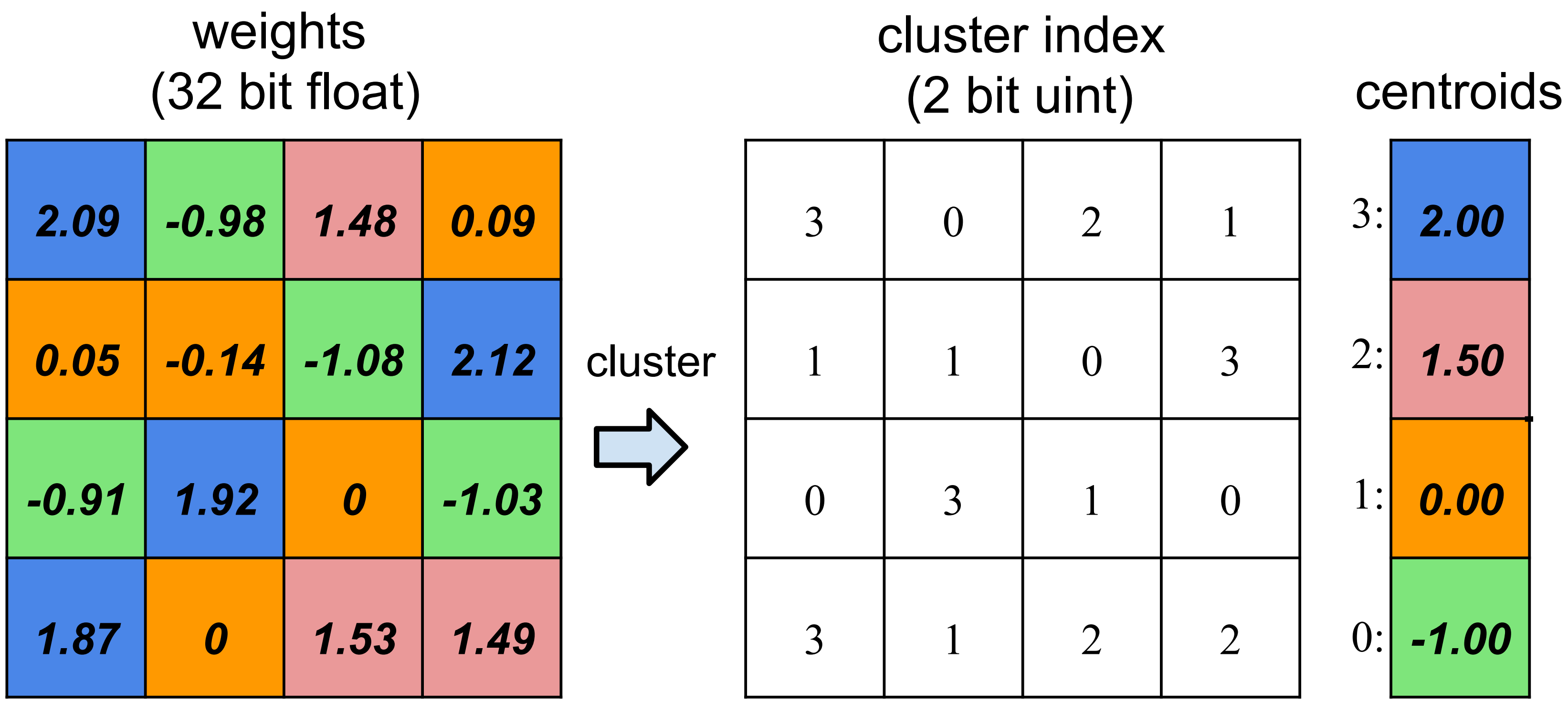
cluster

centroids

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

[Han et al. ICLR’16]

Trained 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 index
(2 bit uint)

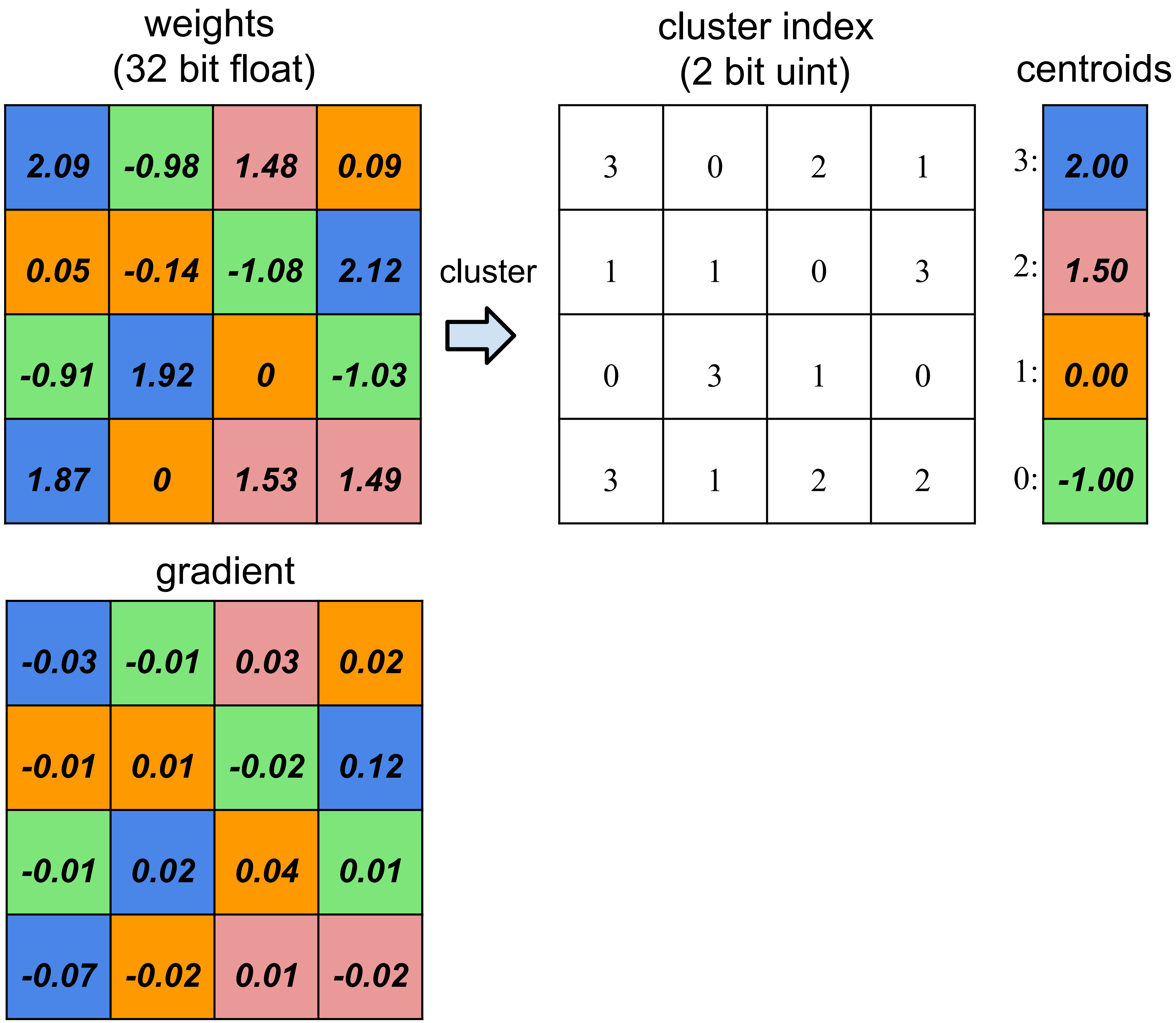
3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

centroids

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

[Han et al. ICLR'16]

Trained 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 uint)

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

centroids

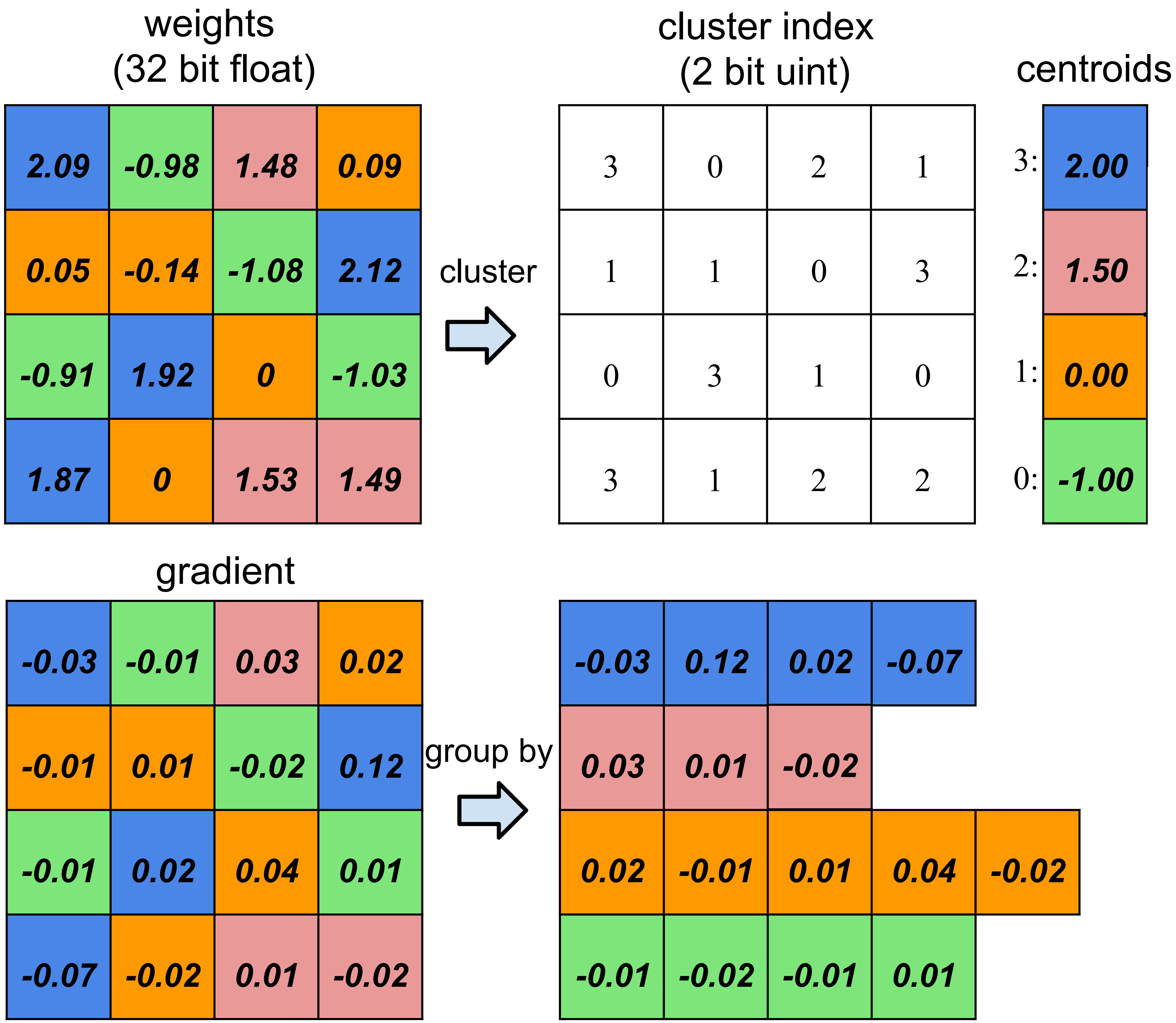
3:	2.00
2:	1.50
1:	0.00
0:	-1.00

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

[Han et al. ICLR'16]

Trained 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 uint)

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

gradient

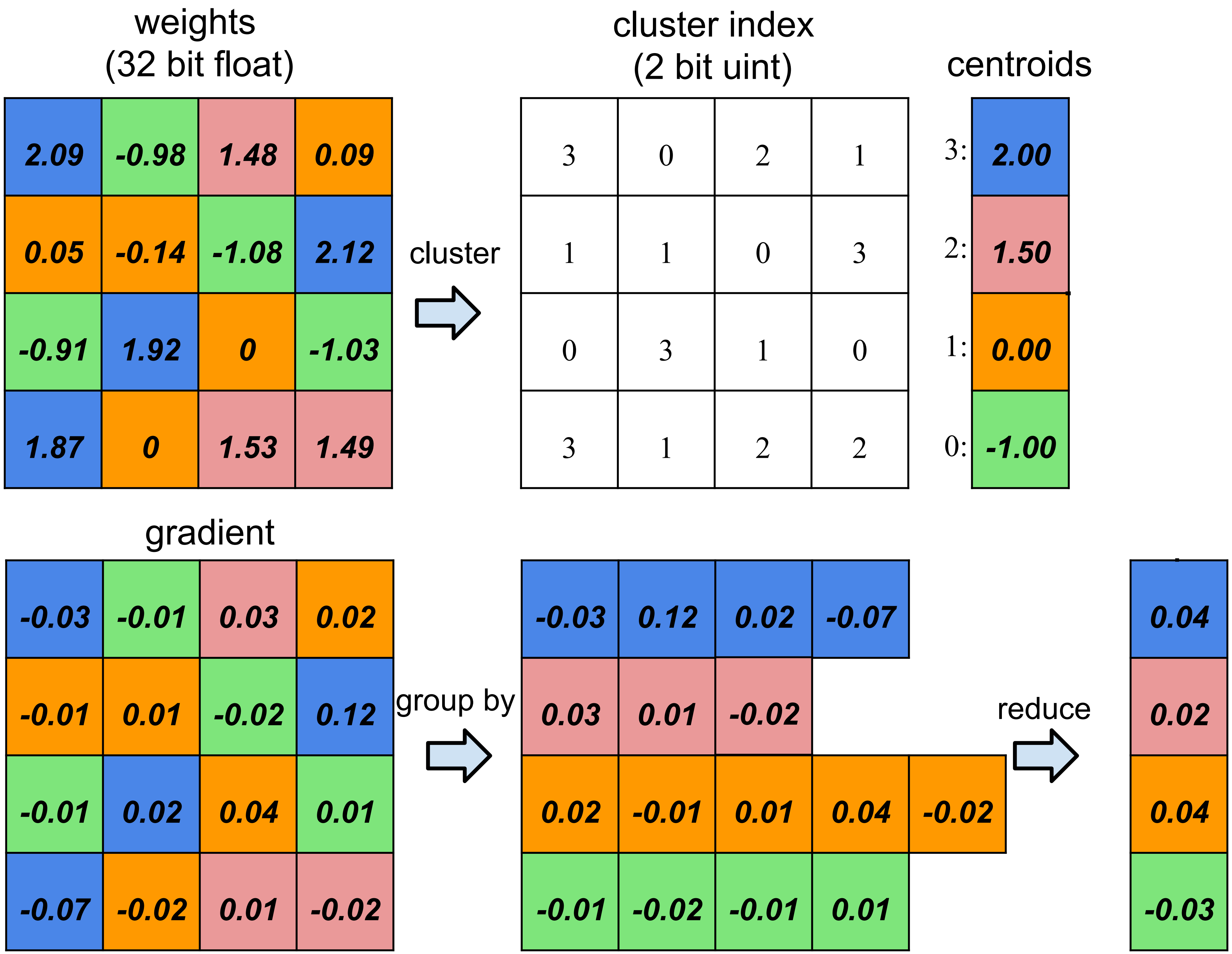
-0.03	-0.01	0.03	0.02
-0.01	0.01	-0.02	0.12
-0.01	0.02	0.04	0.01
-0.07	-0.02	0.01	-0.02

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	

[Han et al. ICLR'16]

Trained 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 uint)

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

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

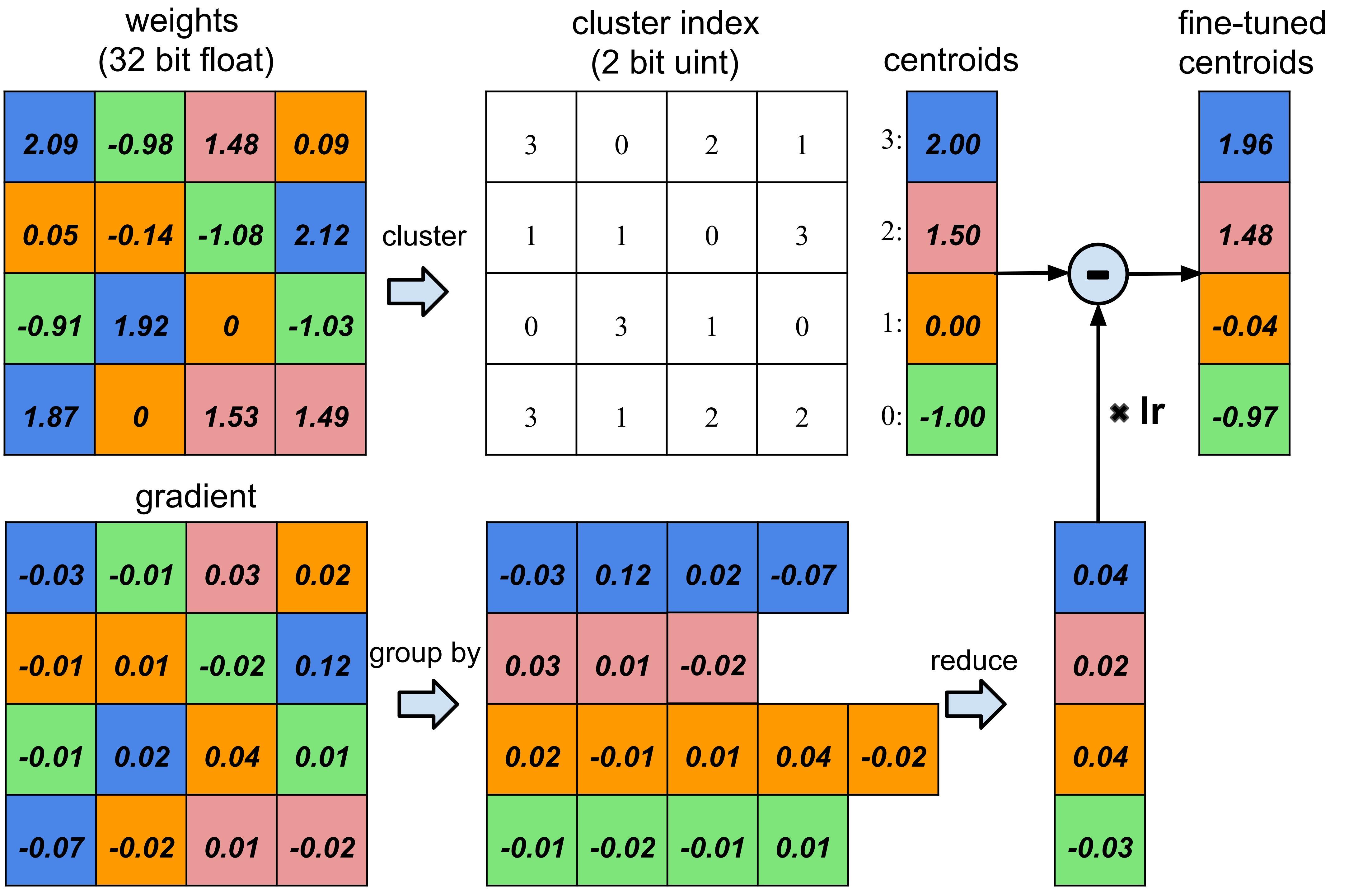
-0.03	0.12	0.02	-0.07	
0.03	0.01	-0.02		
0.02	-0.01	0.01	0.04	-0.02
-0.01	-0.02	-0.01	0.01	

reduce

0.04
0.02
0.04
-0.03

[Han et al. ICLR’16]

Trained 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 uint)

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

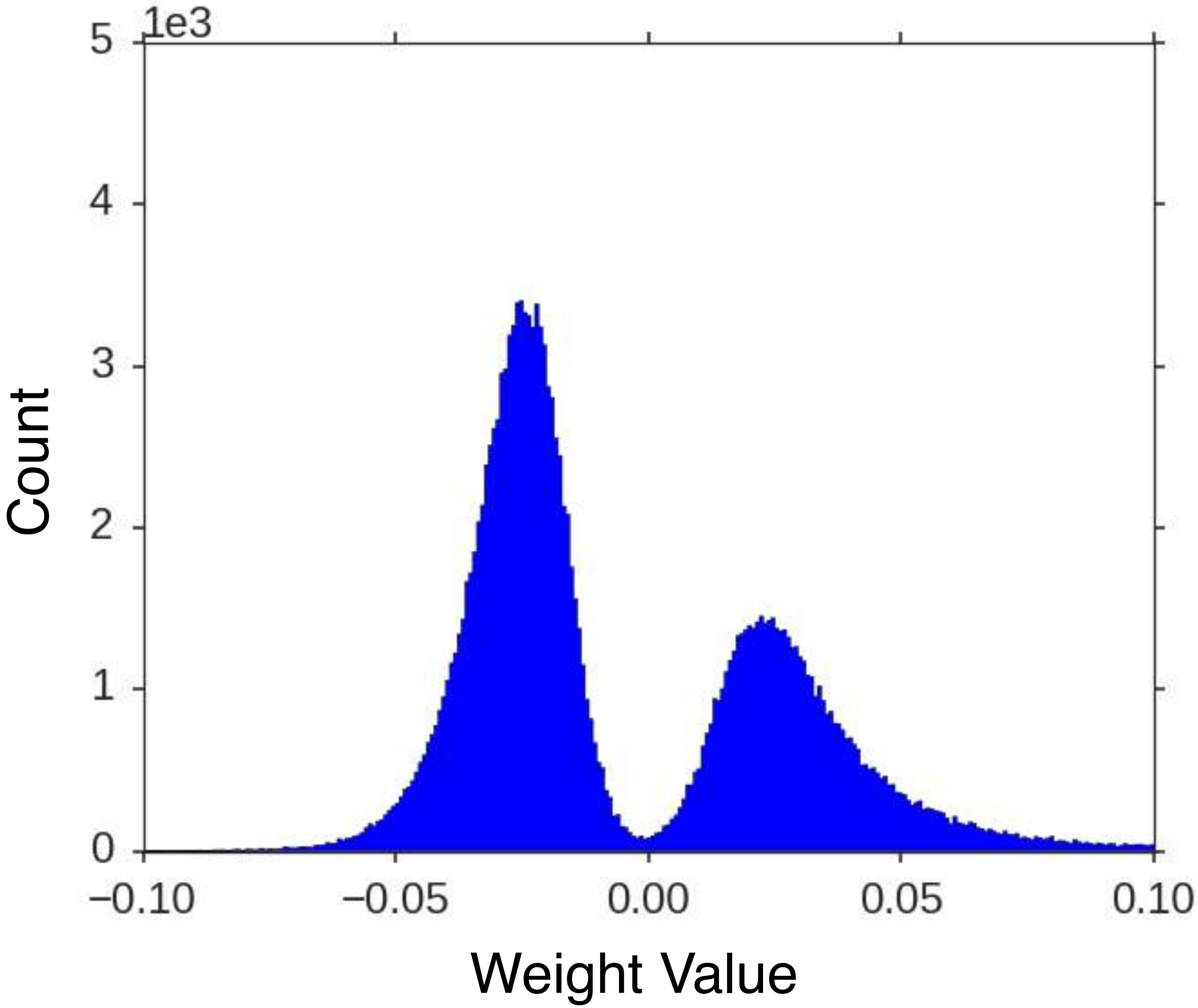
-0.03	0.12	0.02	-0.07	
0.03	0.01	-0.02		
0.02	-0.01	0.01	0.04	-0.02
-0.01	-0.02	-0.01	0.01	

reduce

0.04
0.02
0.04
-0.03

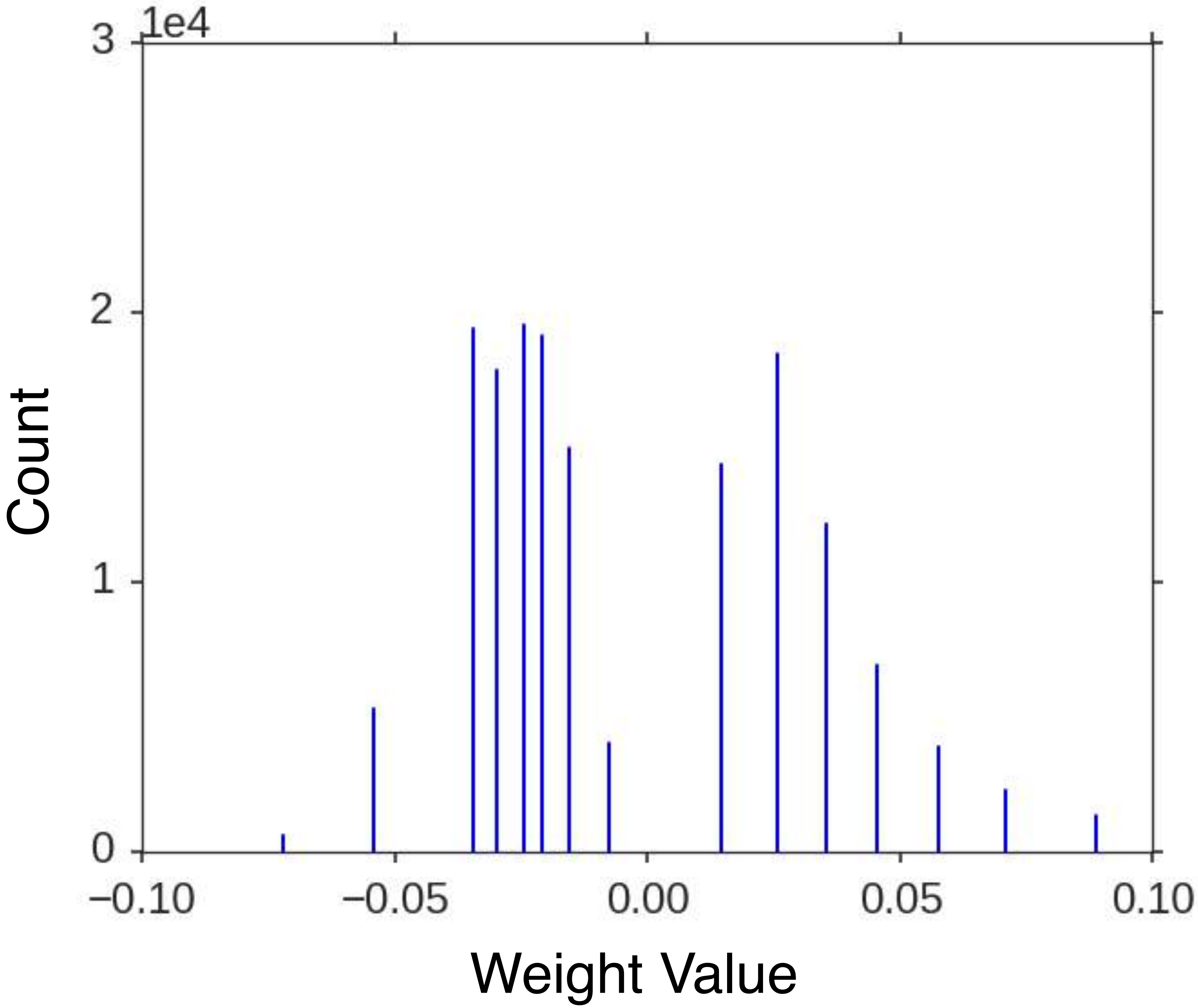
[Han et al. ICLR'16]

Before Trained Quantization: Continuous Weight

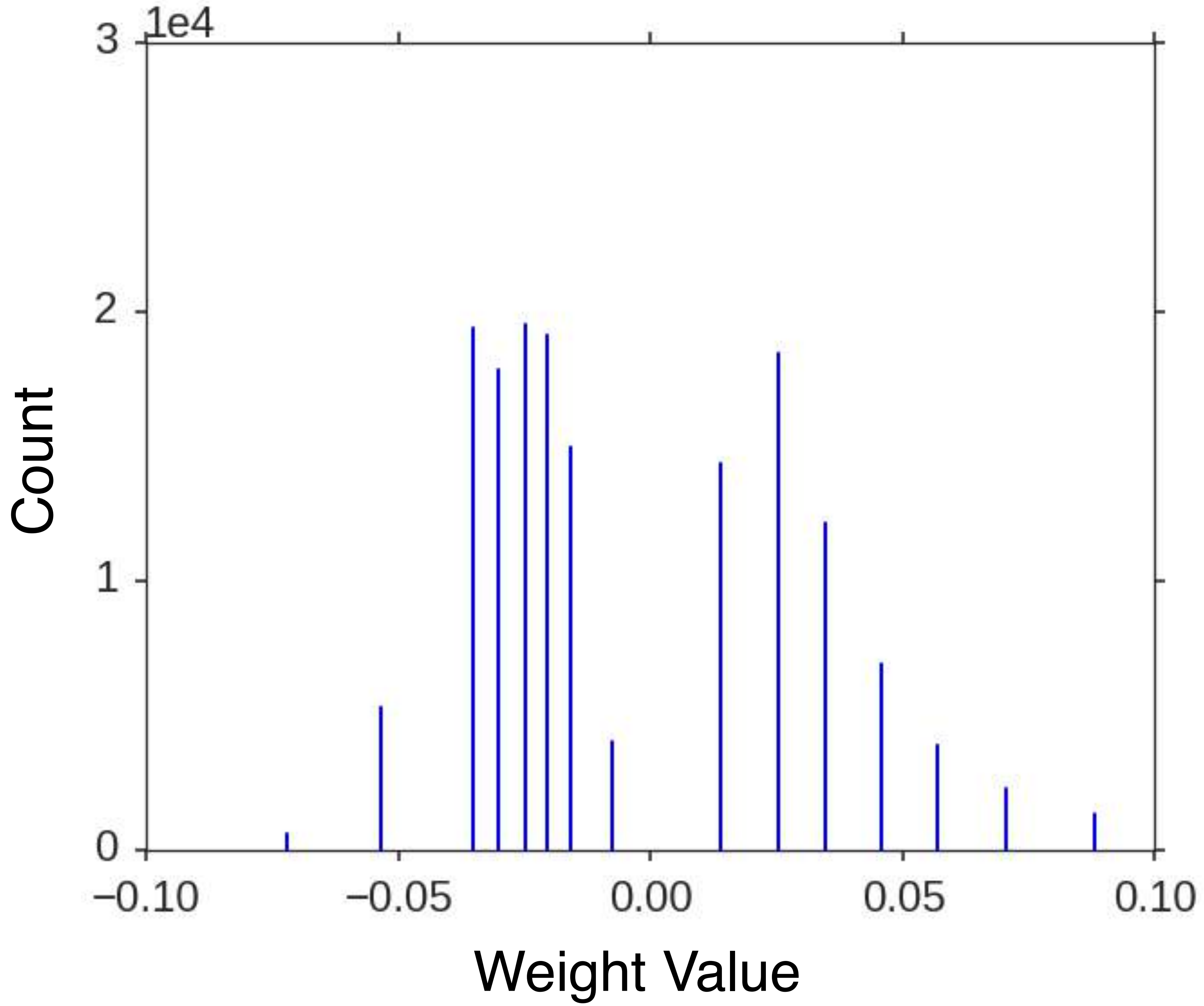


[Han et al. ICLR'16]

After Trained Quantization: Discrete Weight



After Trained Quantization: Discrete Weight after Training

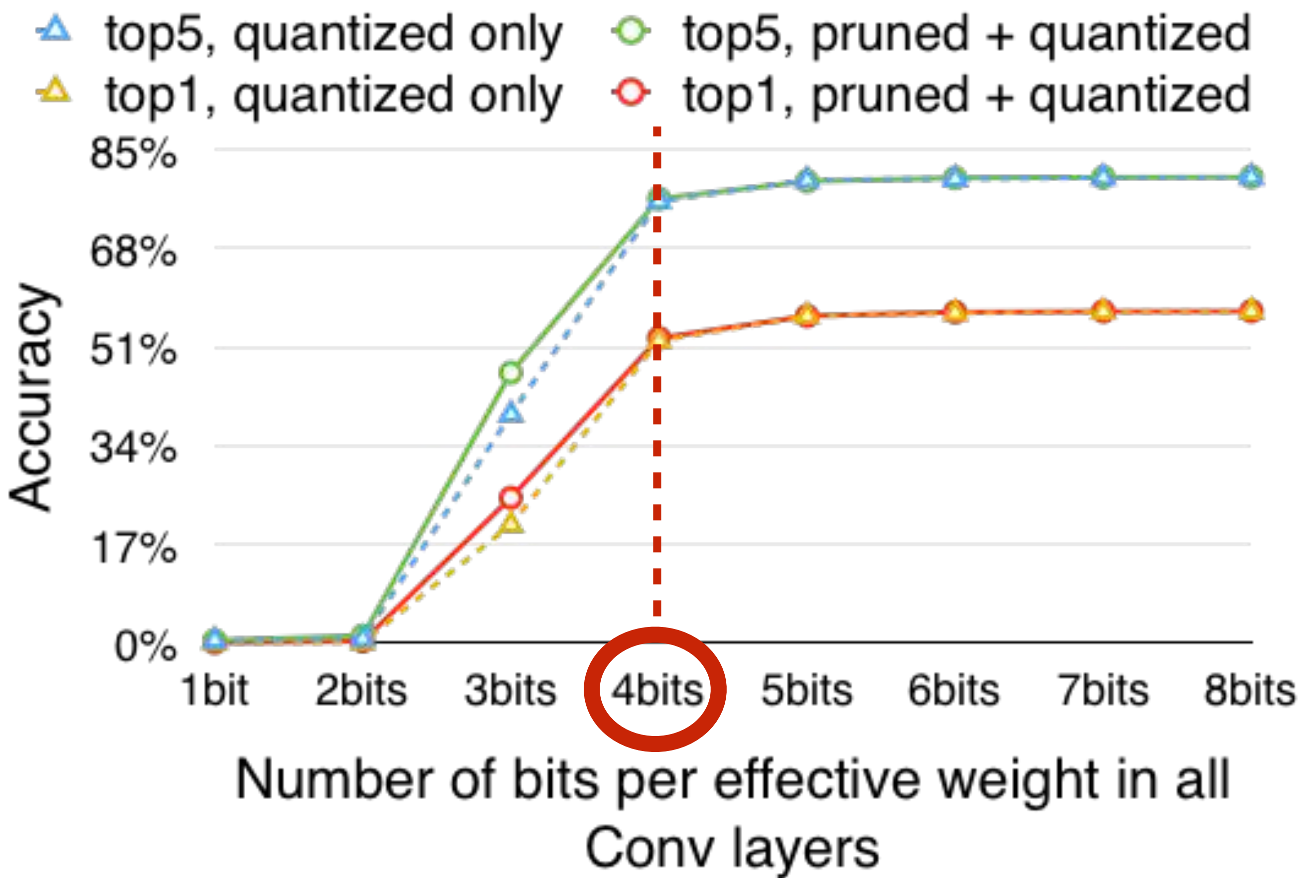


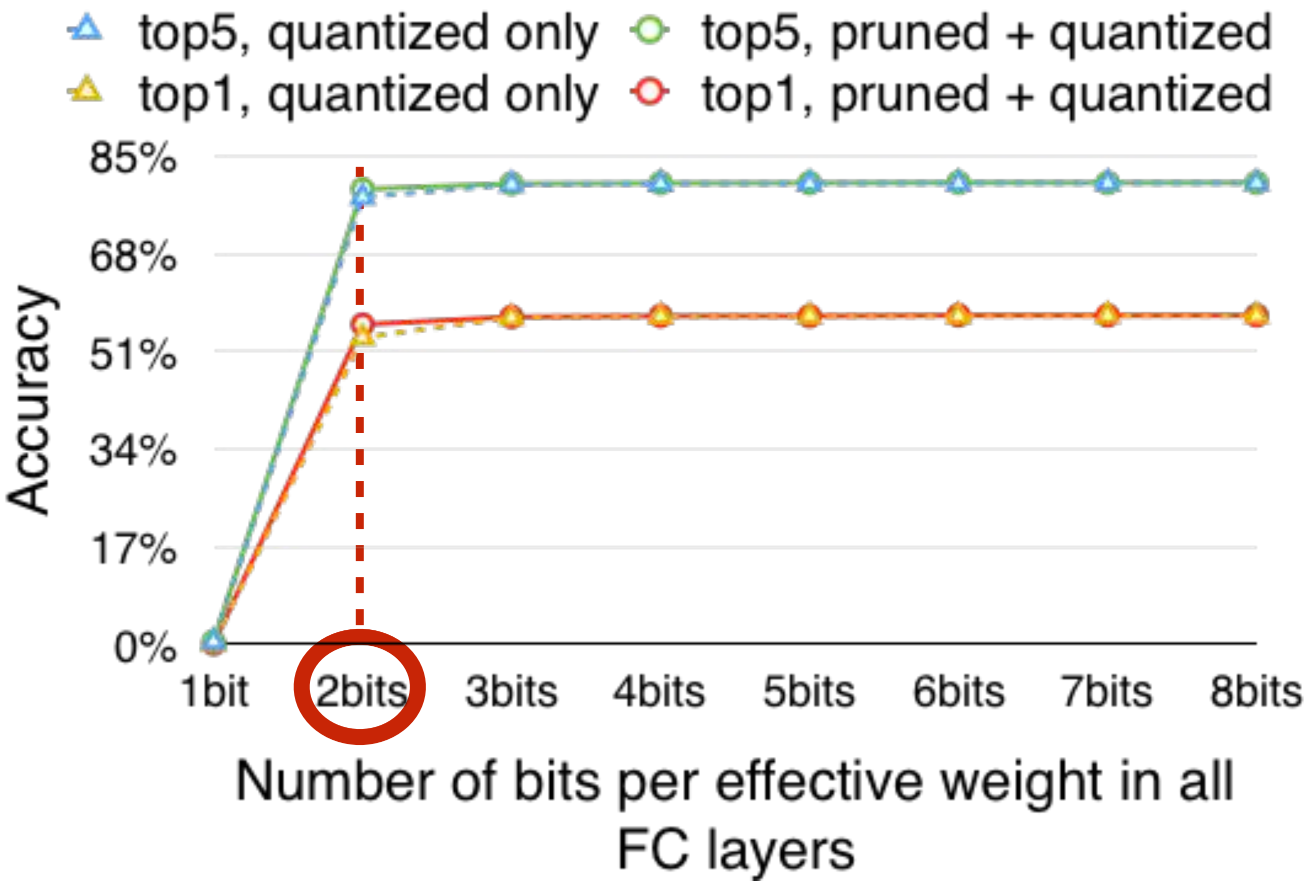
[Han et al. ICLR’16]

How Many Bits do We Need?

[Han et al. ICLR'16]

How Many Bits do We Need?



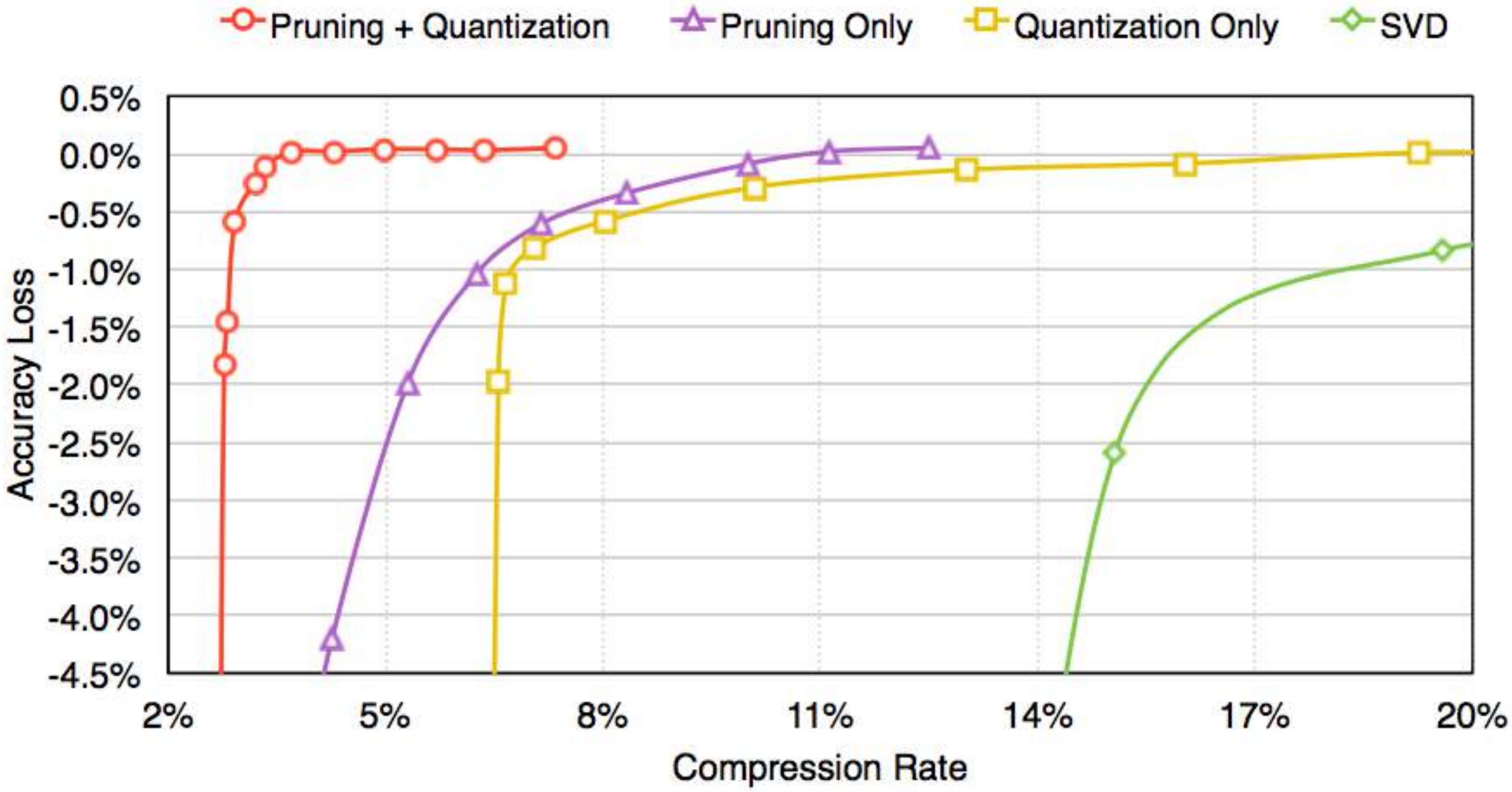


[Han et al. ICLR’16]

Pruning + Trained Quantization Work Together

[Han et al. ICLR’16]

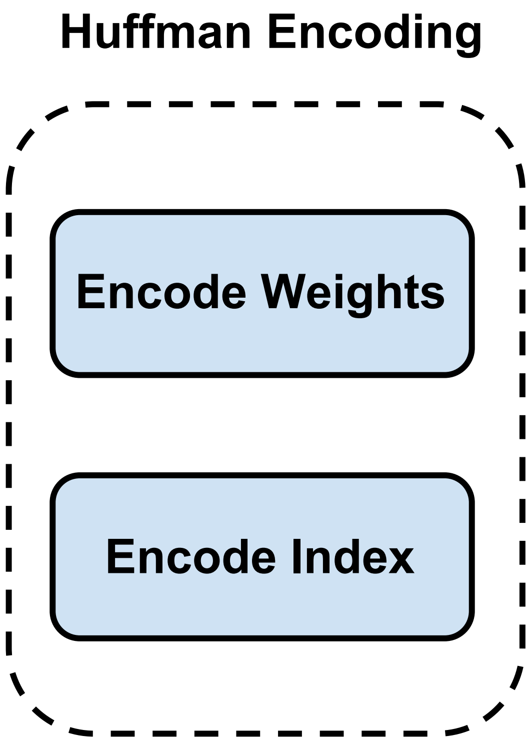
Pruning + Trained Quantization Work Together

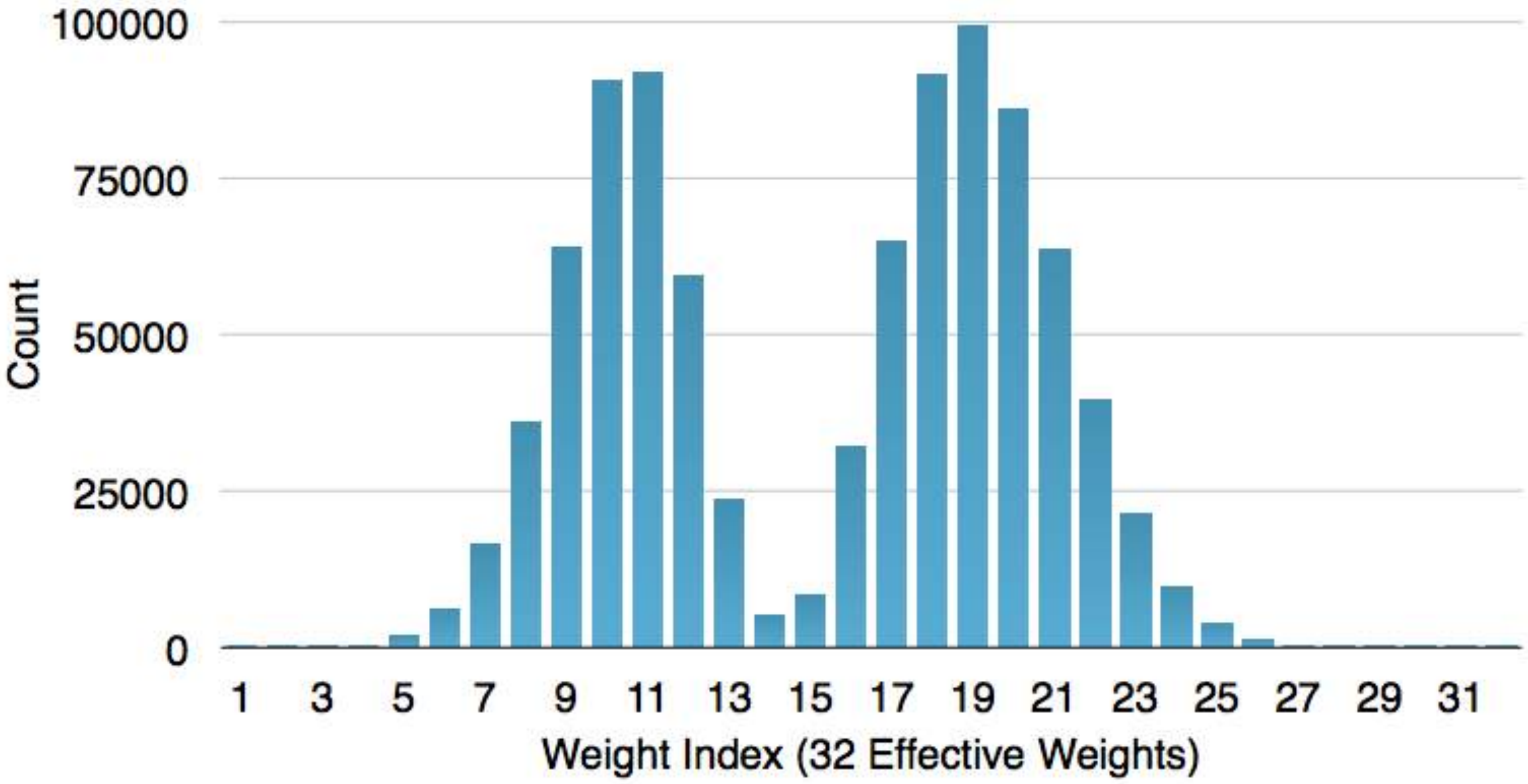


AlexNet on ImageNet

[Han et al. ICLR'16]

Huffman Coding

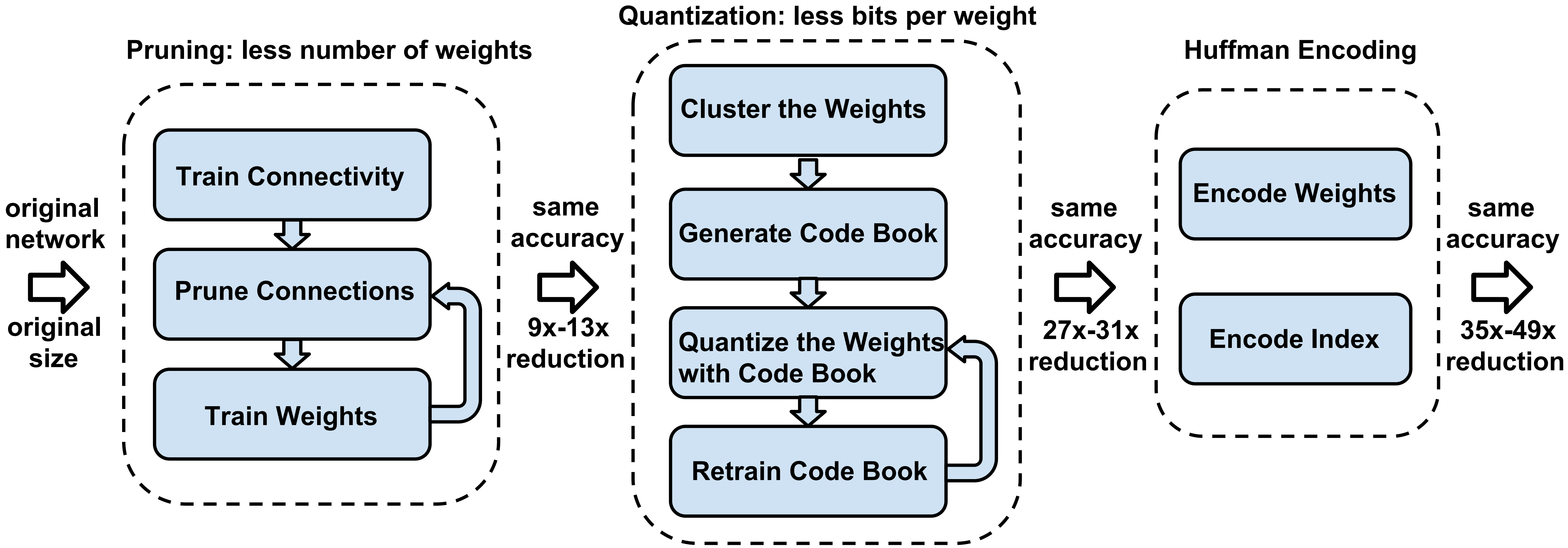




- In-frequent weights: use more bits to represent
- Frequent weights: use less bits to represent

[Han et al. ICLR'16]

Summary of Deep Compression



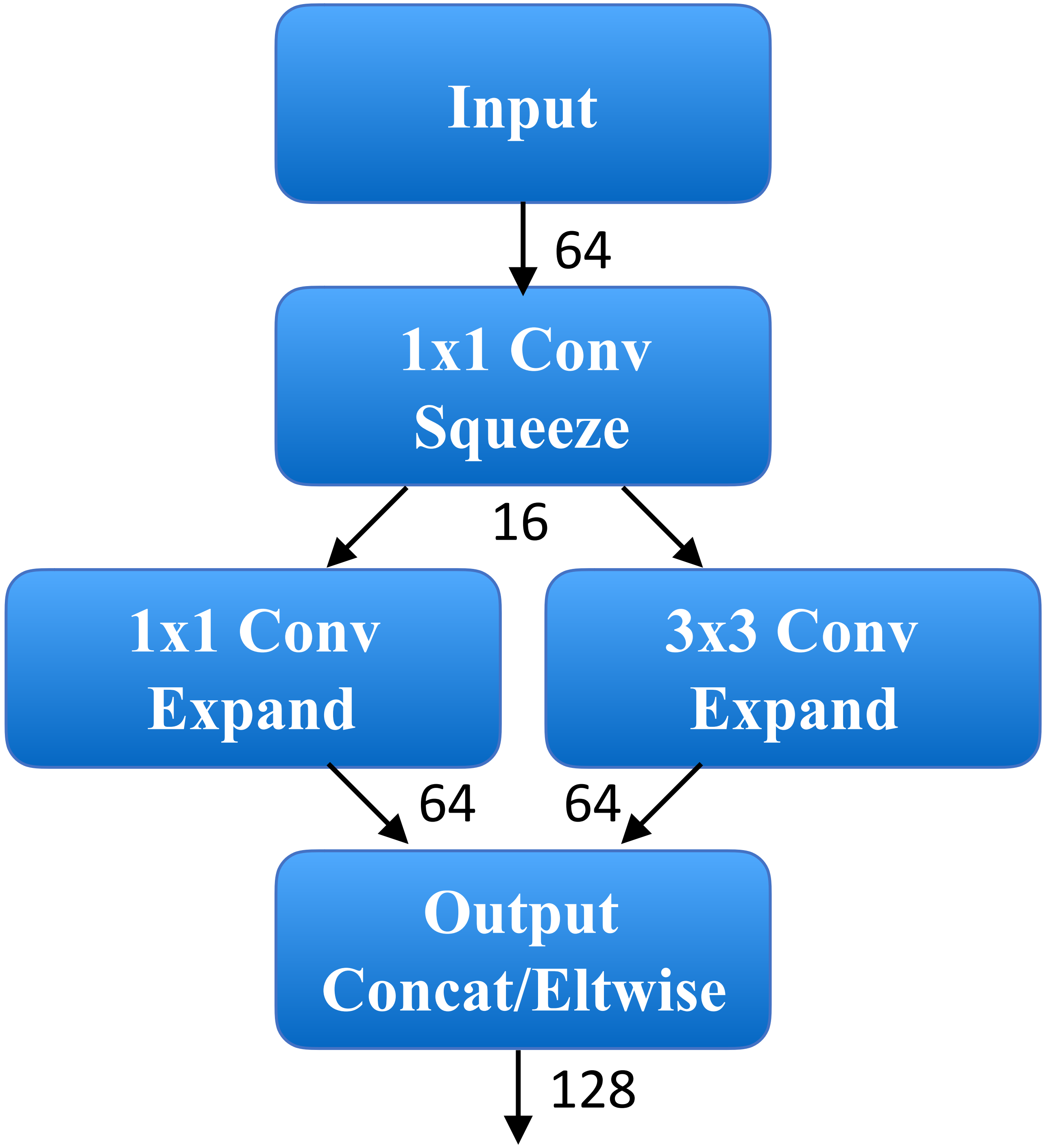
[Han et al. ICLR’16]

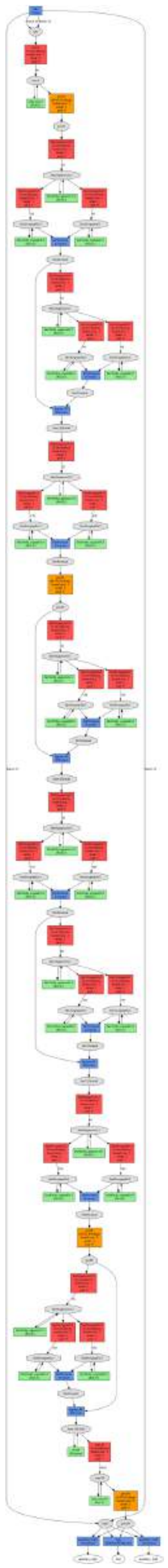
Results: Compression Ratio

Network	Original Size	Compressed Size	Compression Ratio	Original Accuracy	Compressed Accuracy
LeNet-300	1070KB →	27KB	**40x**	98.36% →	98.42%
LeNet-5	1720KB →	44KB	**39x**	99.20% →	99.26%
AlexNet	240MB →	6.9MB	**35x**	80.27% →	80.30%
VGGNet	550MB →	11.3MB	**49x**	88.68% →	89.09%
GoogleNet	28MB →	2.8MB	**10x**	88.90% →	88.92%
ResNet-18	44.6MB →	4.0MB	**11x**	89.24% →	89.28%

Can we make compact models to begin with?

SqueezeNet





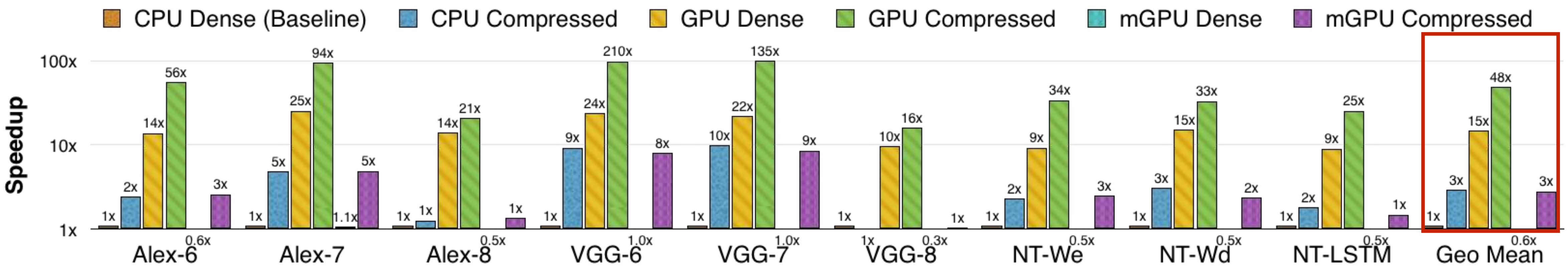
Iandola et al, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size”, arXiv 2016

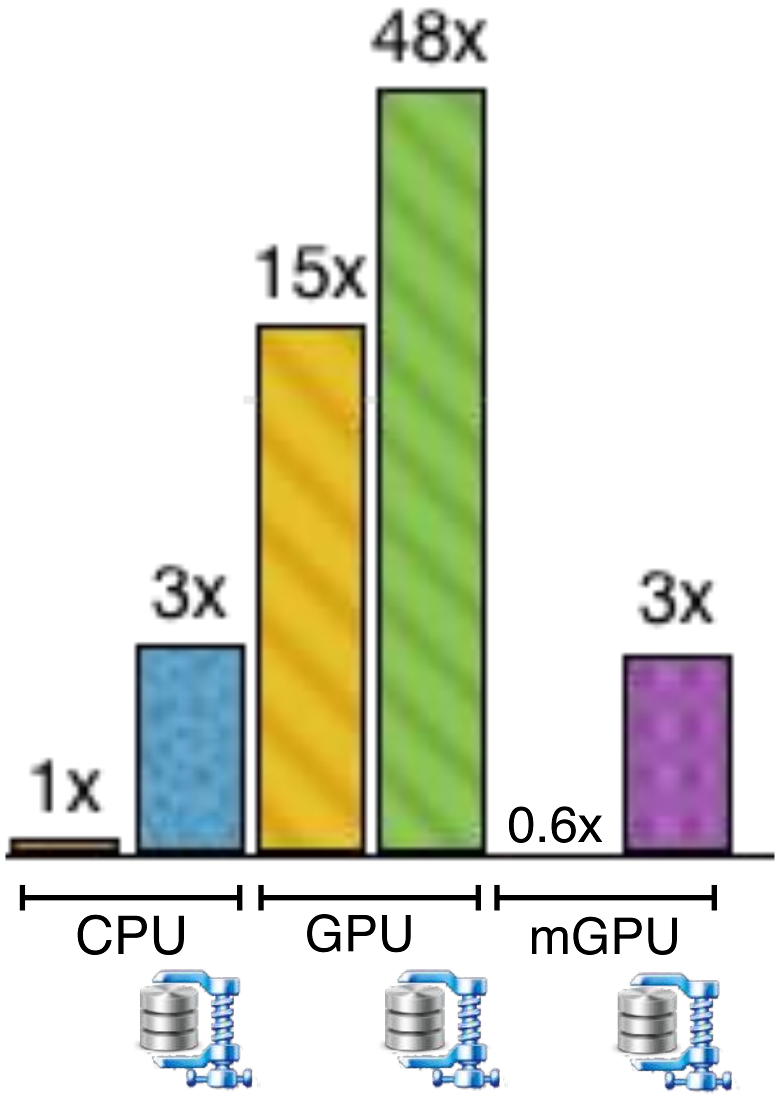
Compressing SqueezeNet

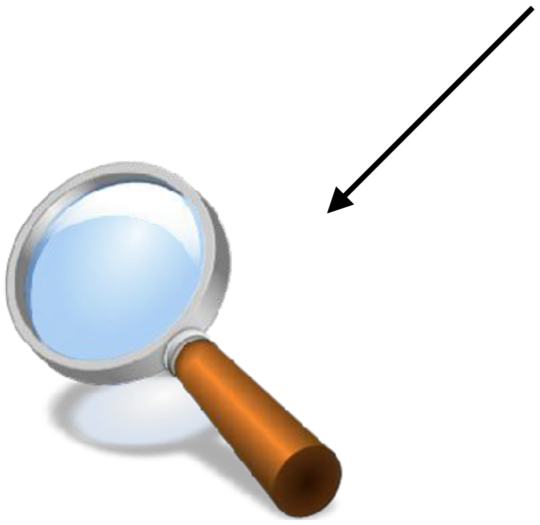
Network	Approach	Size	Ratio	Top-1 Accuracy	Top-5 Accuracy
AlexNet	-	240MB	**1x**	57.2%	80.3%
AlexNet	SVD	48MB	**5x**	56.0%	79.4%
AlexNet	Deep Compression	6.9MB	**35x**	57.2%	80.3%
SqueezeNet	-	4.8MB	**50x**	57.5%	80.3%
SqueezeNet	Deep Compression	0.47MB	**510x**	57.5%	80.3%

Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", arXiv 2016

Results: Speedup

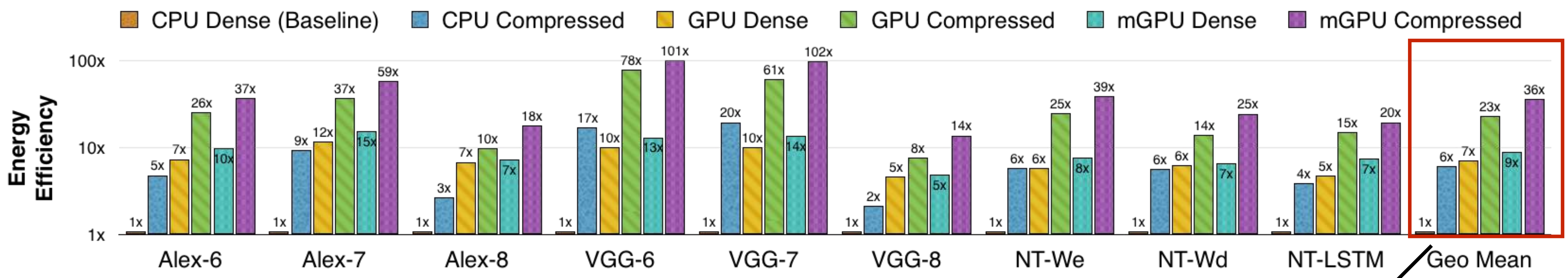


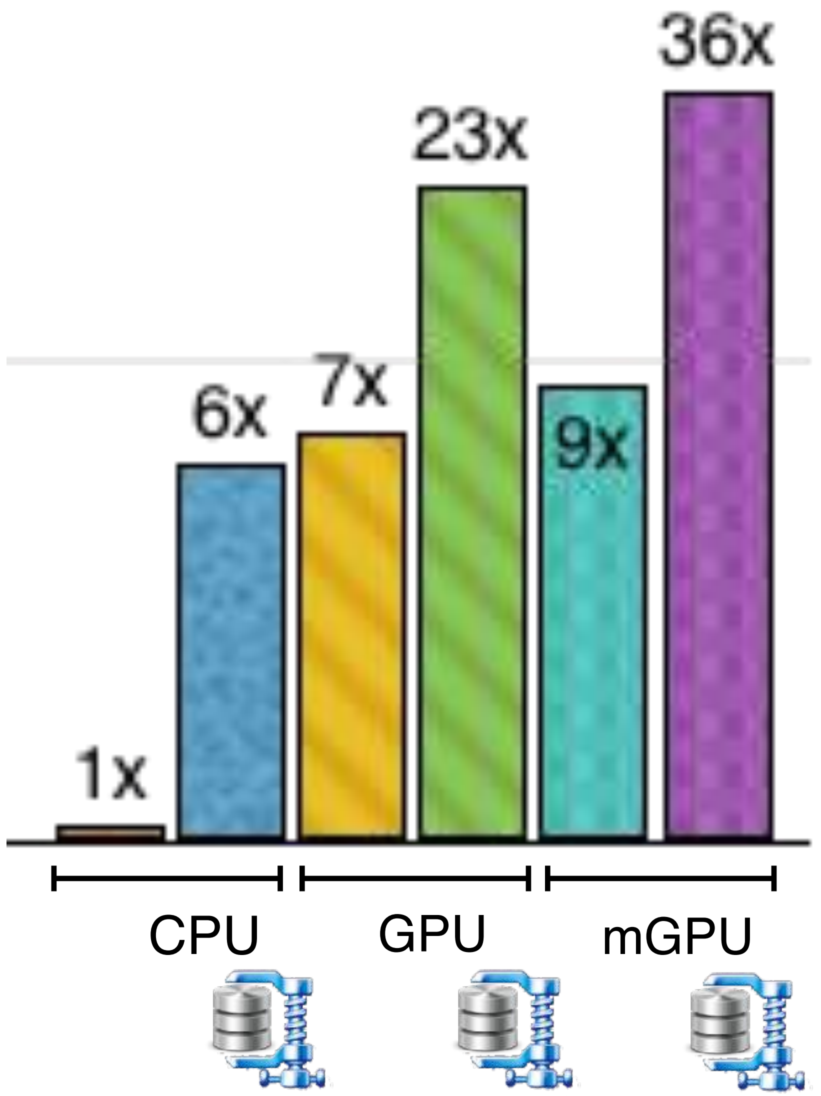




Average

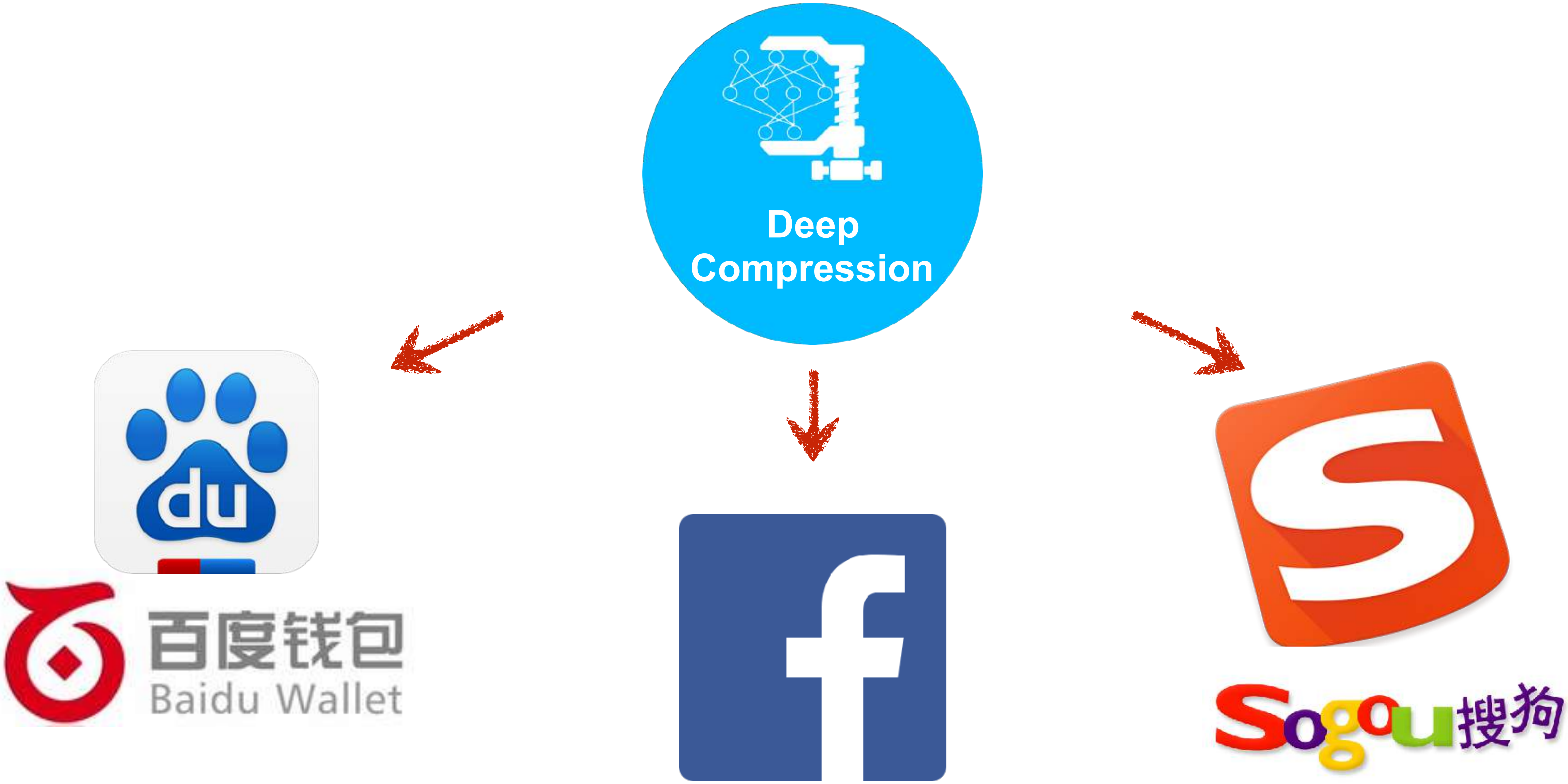
Results: Energy Efficiency





Average

Deep Compression Applied to Industry

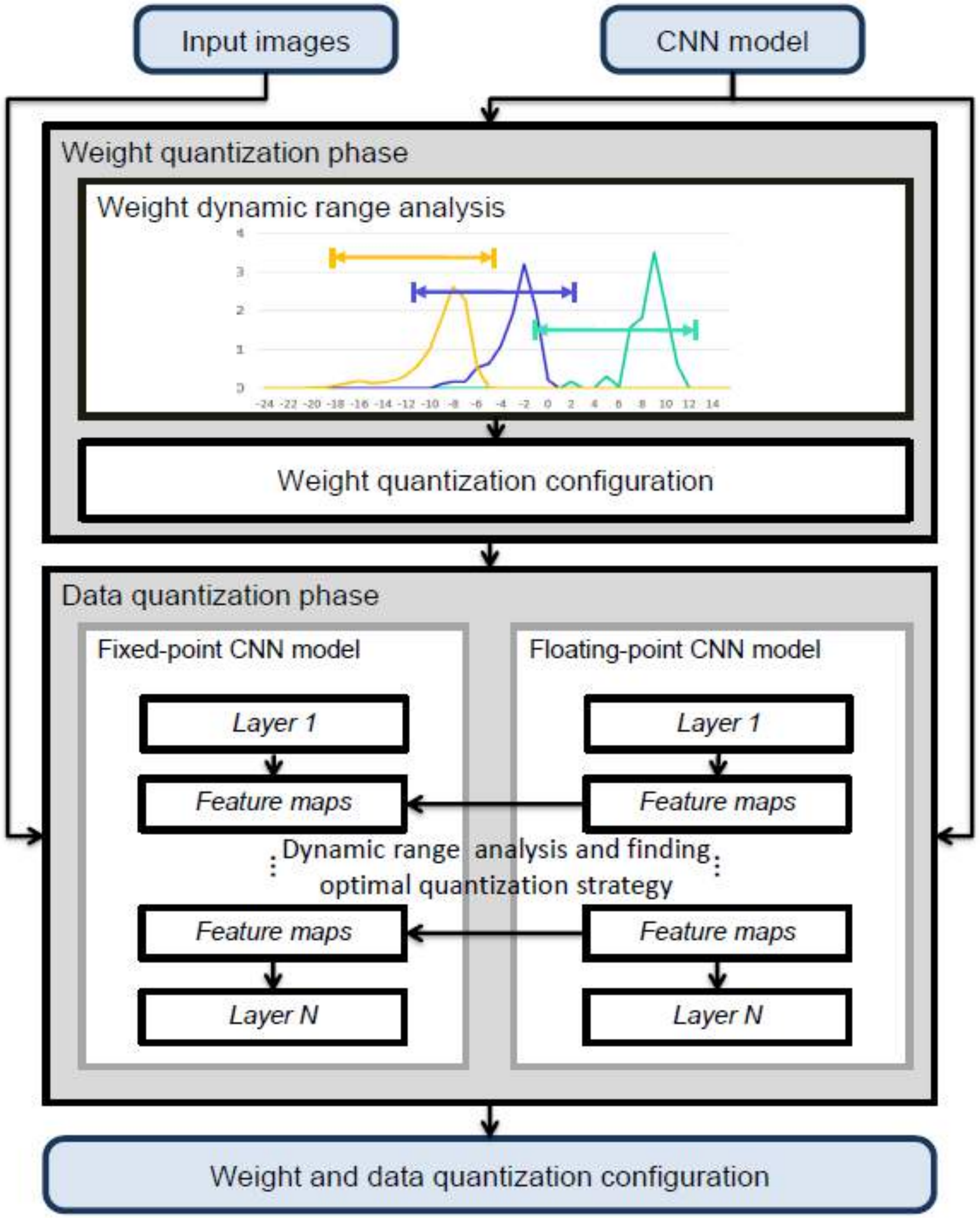


F8

Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

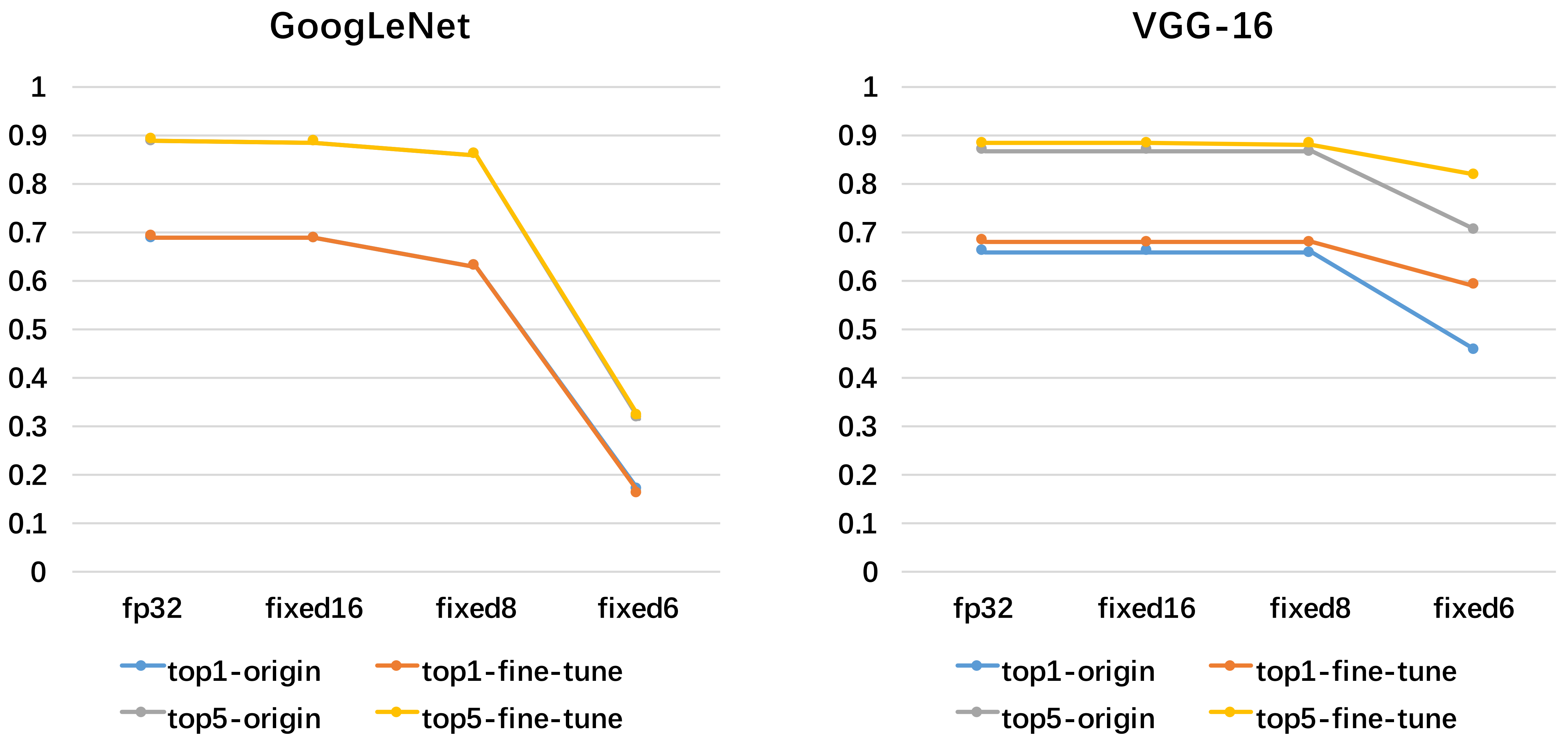
Quantizing the Weight and Activation



- Train with float
- Quantizing the weight and activation:
  - Gather the statistics for weight and activation
  - Choose proper radix point position
- Fine-tune in float format
- Convert to fixed-point format

Qiu et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network, FPGA'16

Quantization Result



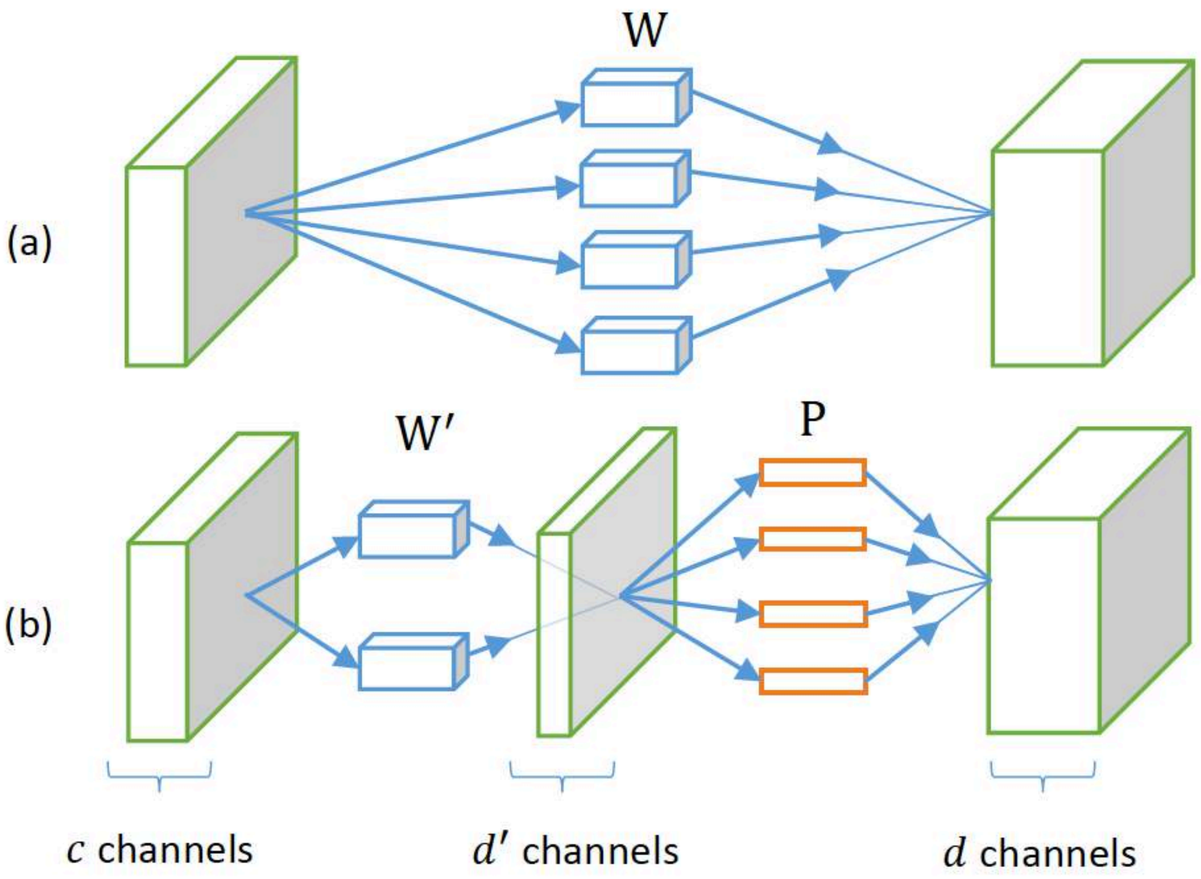
Qiu et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network, FPGA'16

Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

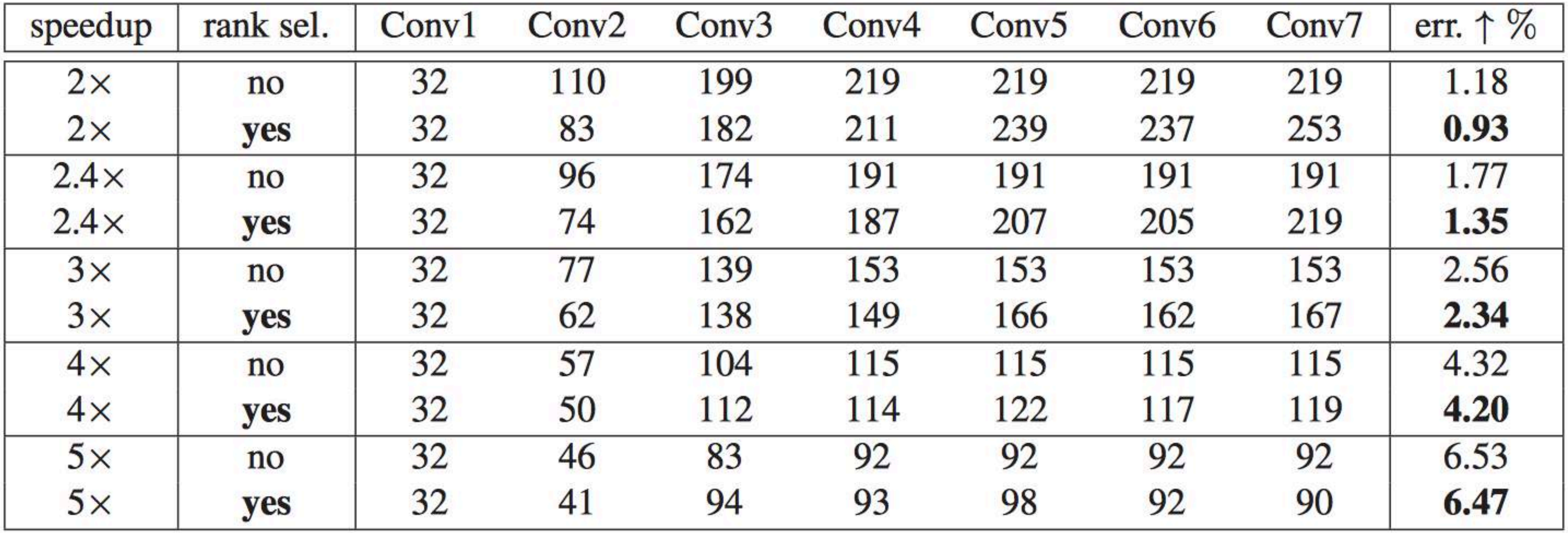
Low Rank Approximation for Conv

- Layer responses lie in a low-rank subspace
- Decompose a convolutional layer with d filters with filter size $k \times k \times c$ to
  - A layer with d' filters ($k \times k \times c$)
  - A layer with d filter ($1 \times 1 \times d'$)



Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

Low Rank Approximation for Conv



speedup	rank sel.	Conv1	Conv2	Conv3	Conv4	Conv5	Conv6	Conv7	err. ↑ %
2×	no	32	110	199	219	219	219	219	1.18
2×	**yes**	32	83	182	211	239	237	253	**0.93**
2.4×	no	32	96	174	191	191	191	191	1.77
2.4×	**yes**	32	74	162	187	207	205	219	**1.35**
3×	no	32	77	139	153	153	153	153	2.56
3×	**yes**	32	62	138	149	166	162	167	**2.34**
4×	no	32	57	104	115	115	115	115	4.32
4×	**yes**	32	50	112	114	122	117	119	**4.20**
5×	no	32	46	83	92	92	92	92	6.53
5×	**yes**	32	41	94	93	98	92	90	**6.47**

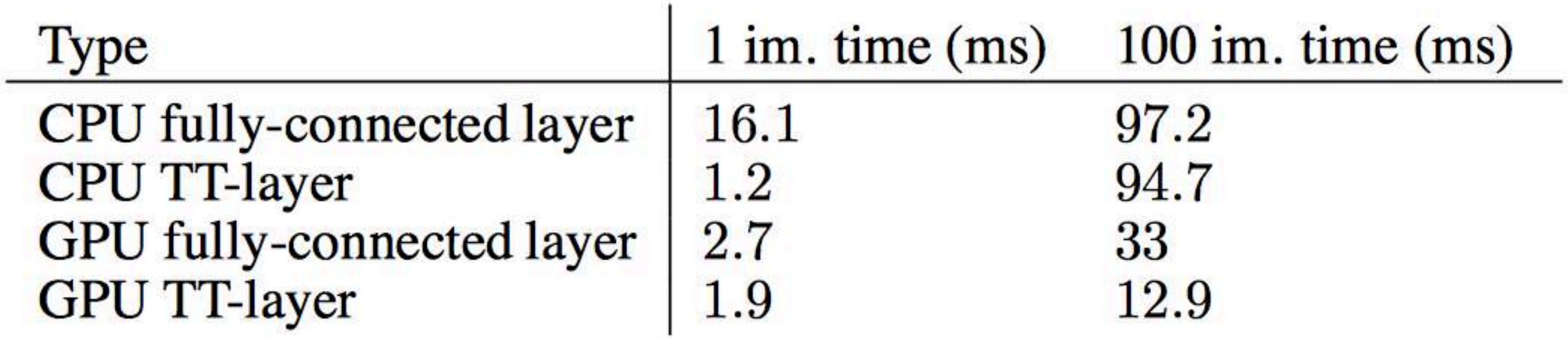
Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR’15

Low Rank Approximation for FC

Build a mapping from row / column indices of matrix $\boldsymbol{W} = [W(x, y)]$ to vectors $\boldsymbol{i}$ and $\boldsymbol{j}$: $x \leftrightarrow \boldsymbol{i} = (i_1, \dots, i_d)$ and $y \leftrightarrow \boldsymbol{j} = (j_1, \dots, j_d)$.

TT-format for matrix $\boldsymbol{W}$:

$$\boldsymbol{W}(i_1, \dots, i_d; j_1, \dots, j_d) = \boldsymbol{W}(x(\boldsymbol{i}), y(\boldsymbol{j})) = \underbrace{\boldsymbol{G}_1[i_1, j_1]}_{1\times r}\,\underbrace{\boldsymbol{G}_2[i_2, j_2]}_{r \times r} \dots \underbrace{\boldsymbol{G}_d[i_d, j_d]}_{r \times 1}$$



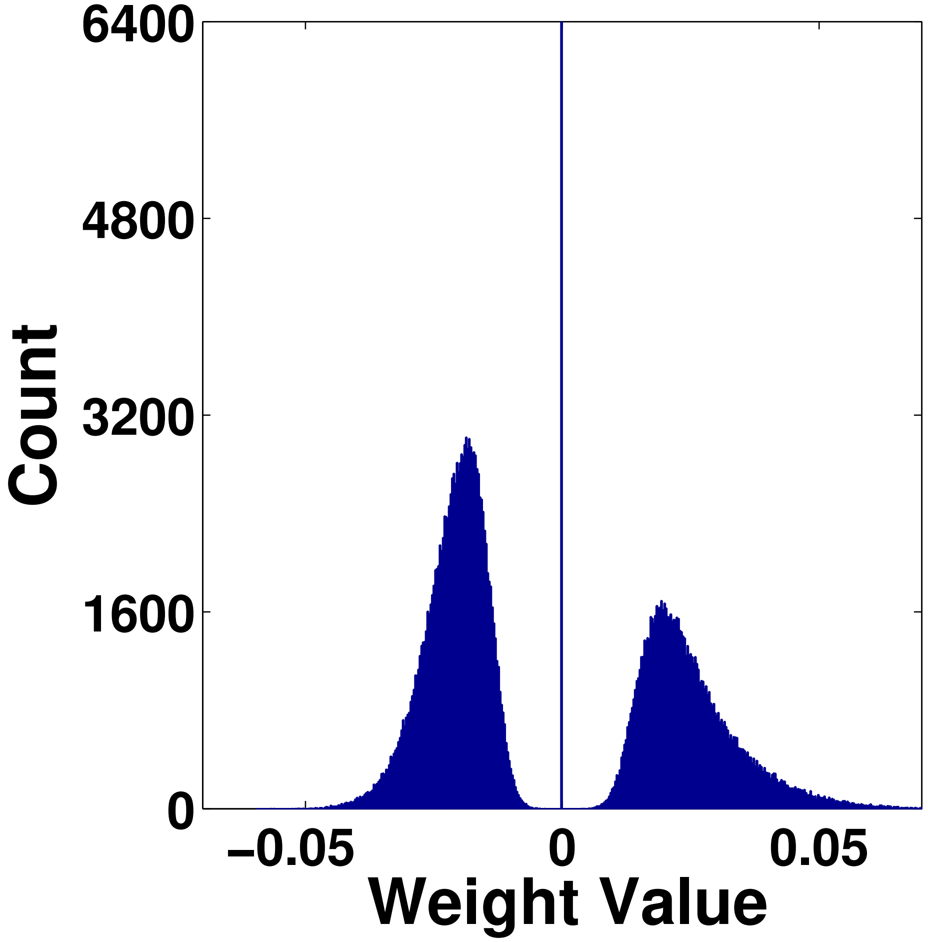
Type	1 im. time (ms)	100 im. time (ms)
CPU fully-connected layer	16.1	97.2
CPU TT-layer	1.2	94.7
GPU fully-connected layer	2.7	33
GPU TT-layer	1.9	12.9

Novikov et al Tensorizing Neural Networks, NIPS’15

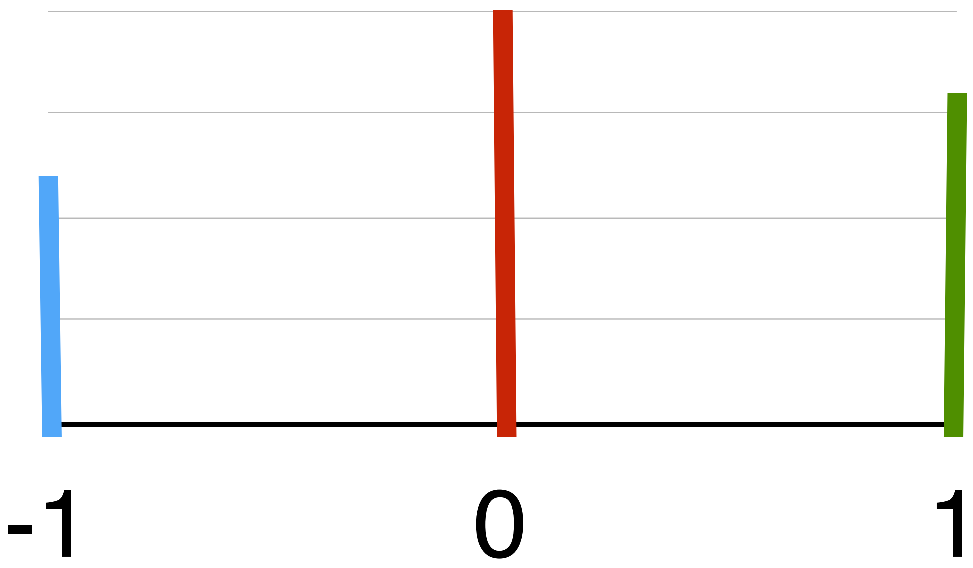
Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

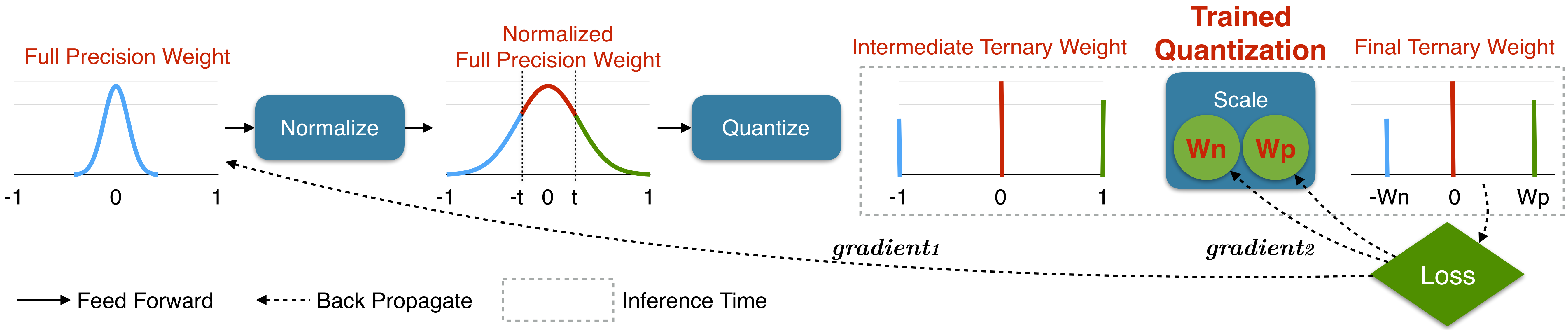
Binary / Ternary Net: Motivation



=>



Trained Ternary Quantization



Zhu, Han, Mao, Dally. Trained Ternary Quantization, ICLR’17

Weight Evolution during Training

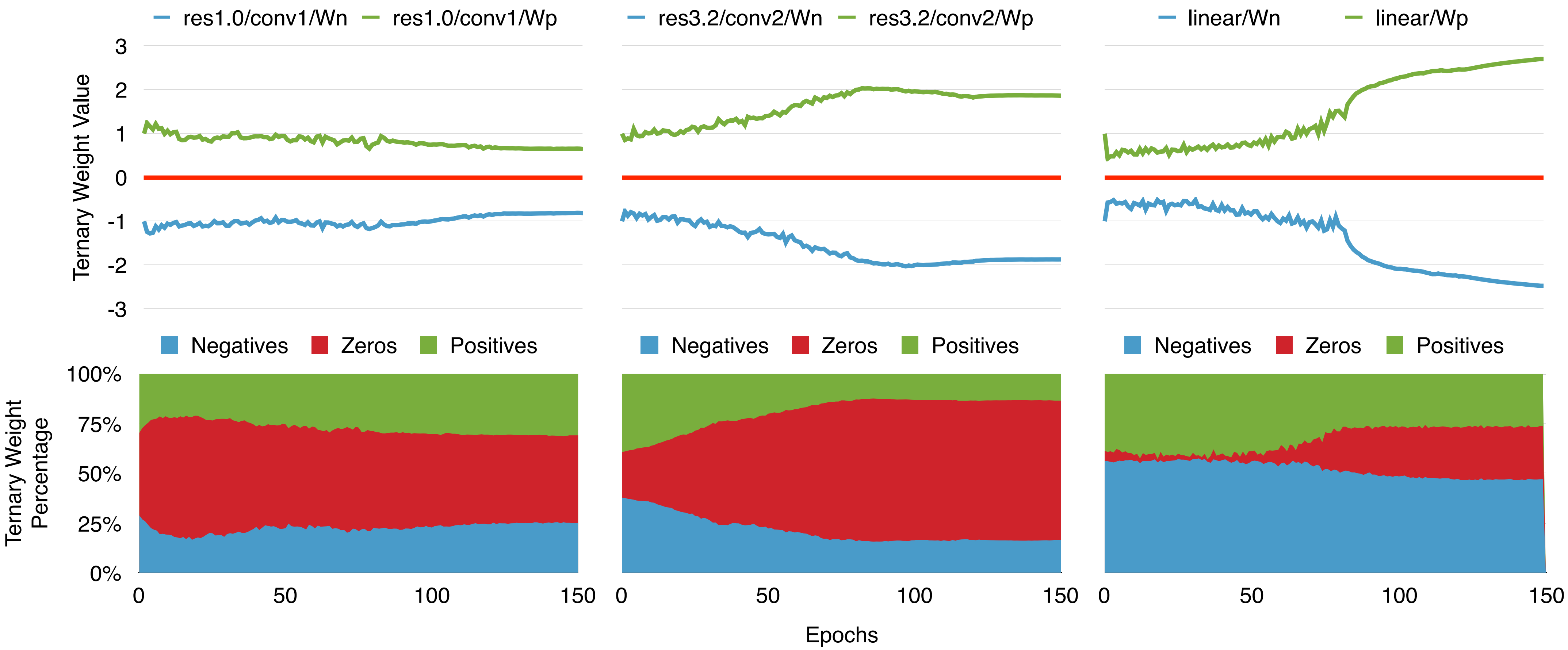
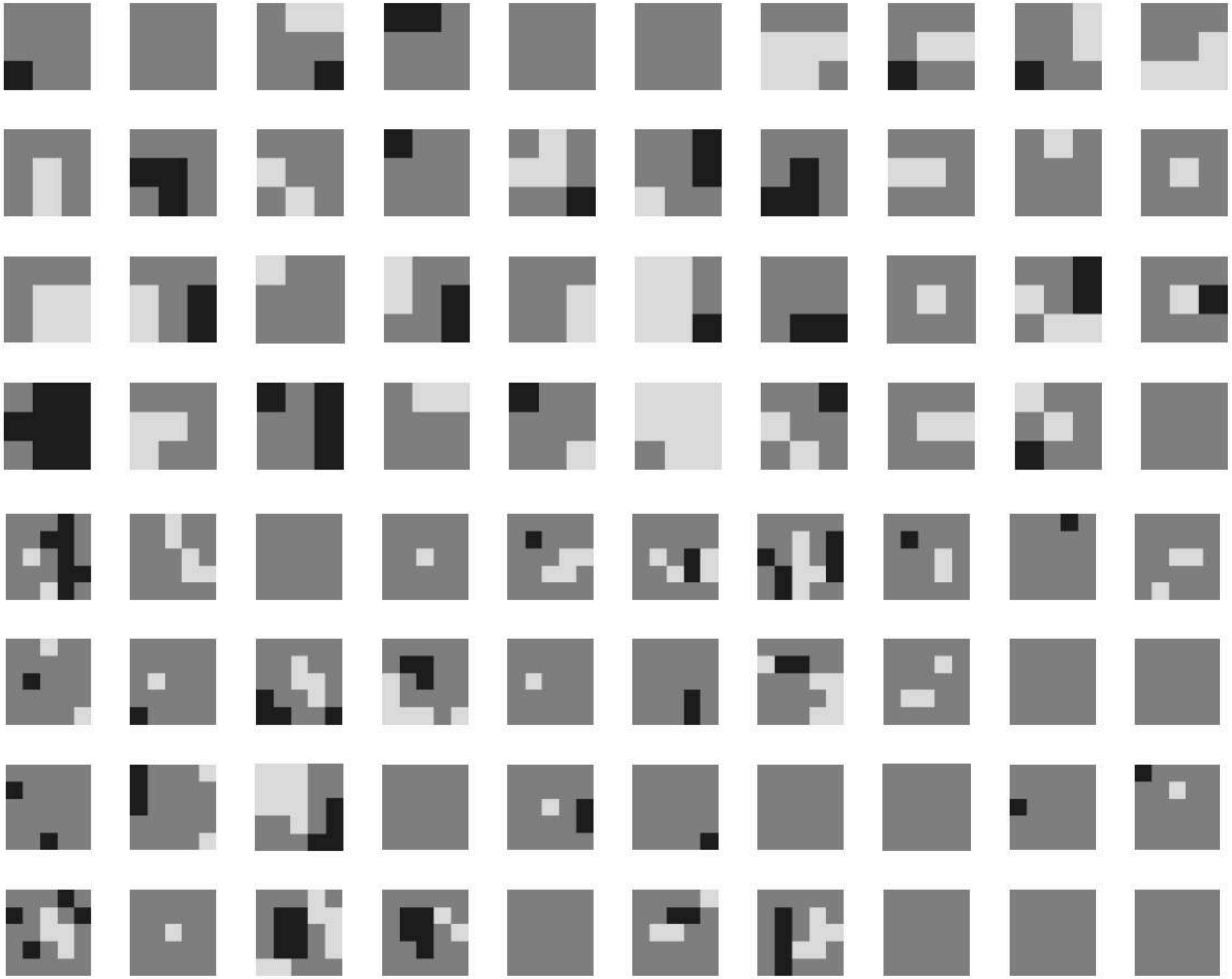


Figure 2: Ternary weights value (above) and distribution (below) with iterations for different layers of ResNet-20 on CIFAR-10.

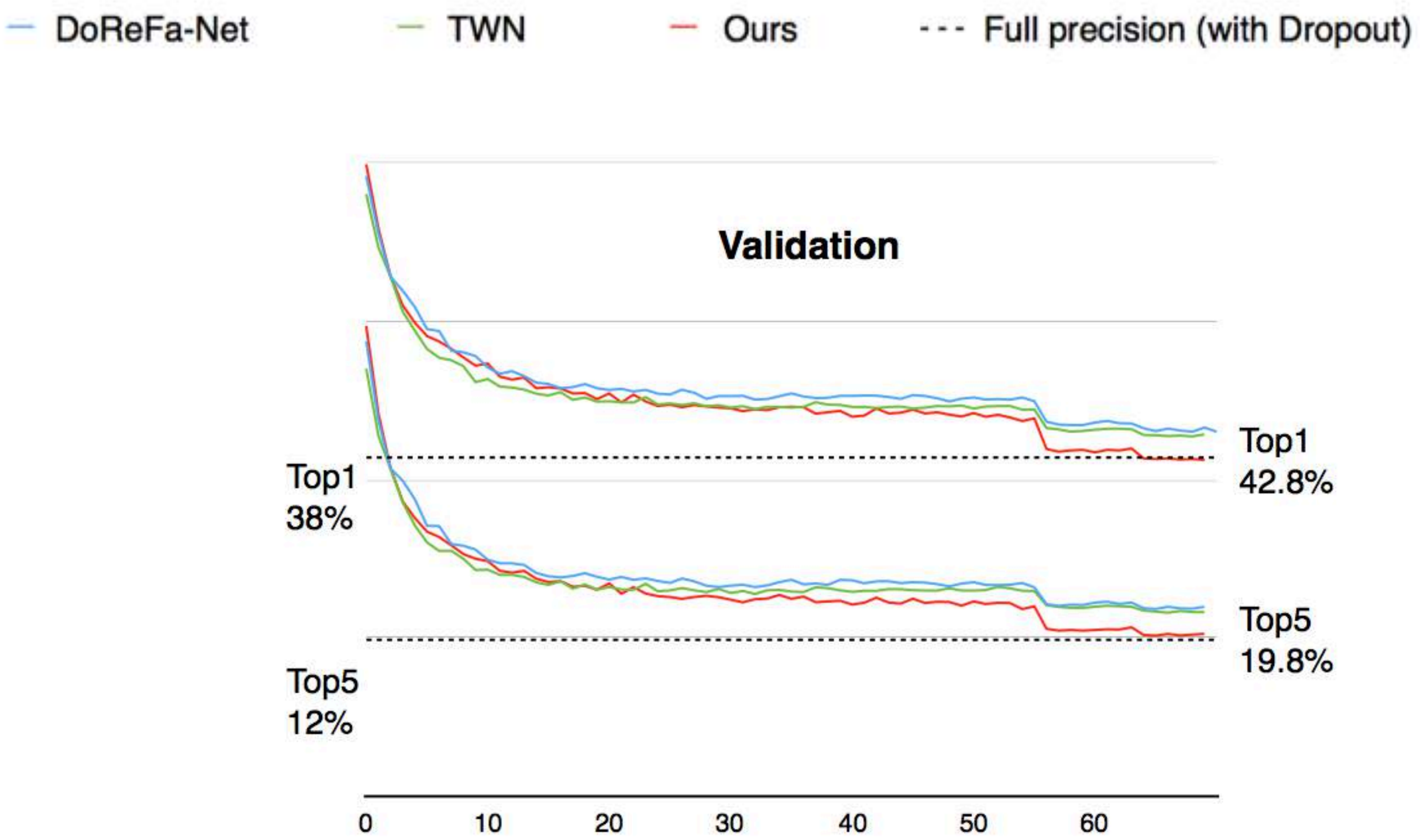
Zhu, Han, Mao, Dally. Trained Ternary Quantization, ICLR'17

Visualization of the TTQ Kernels



Zhu, Han, Mao, Dally. Trained Ternary Quantization, ICLR'17

Error Rate on ImageNet



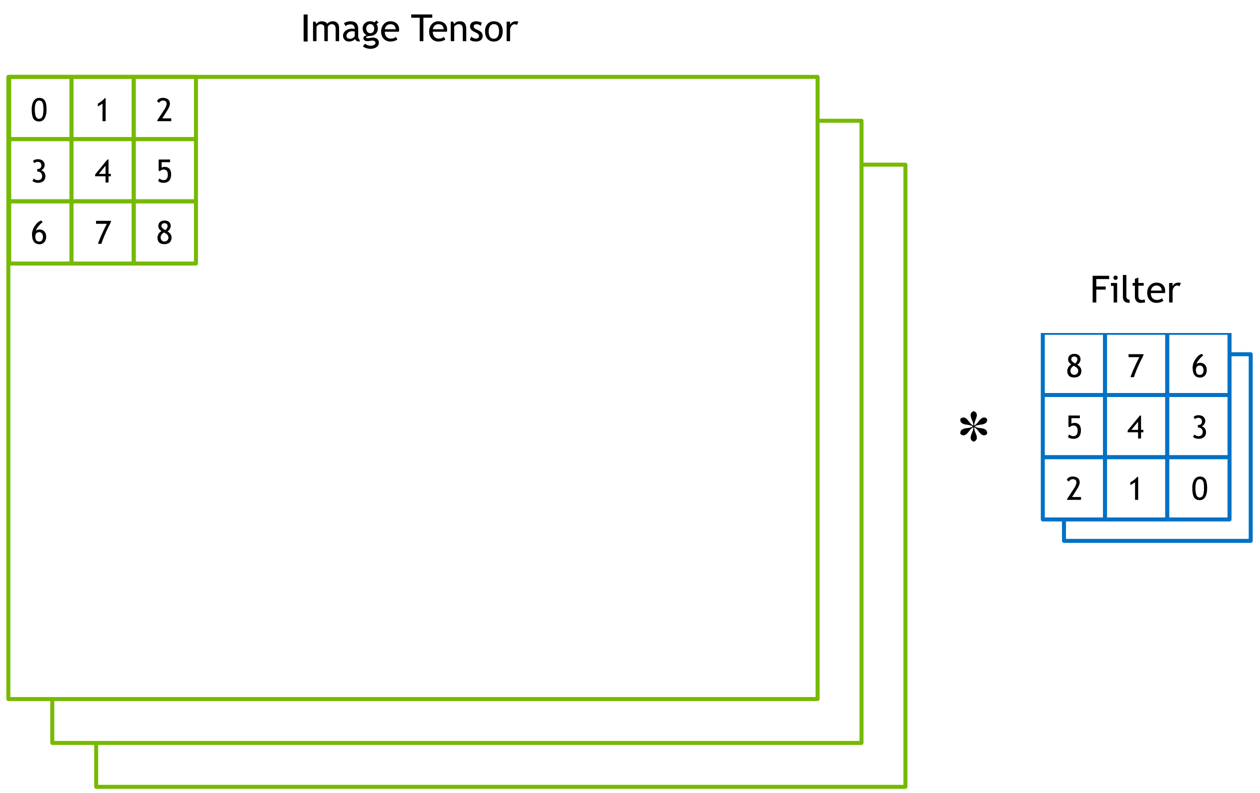
Zhu, Han, Mao, Dally. Trained Ternary Quantization, ICLR’17

Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

3x3 DIRECT Convolutions

Compute Bound



9xC FMAs/Output: Math Intensive

$$\sum \begin{array}{l} \boxed{0} \times \boxed{8} + \boxed{1} \times \boxed{7} + \boxed{2} \times \boxed{6} + \\ \boxed{3} \times \boxed{5} + \boxed{4} \times \boxed{4} + \boxed{5} \times \boxed{3} + \\ \boxed{6} \times \boxed{2} + \boxed{7} \times \boxed{1} + \boxed{8} \times \boxed{0} \end{array}$$

9xK FMAs/Input: Good Data Reuse

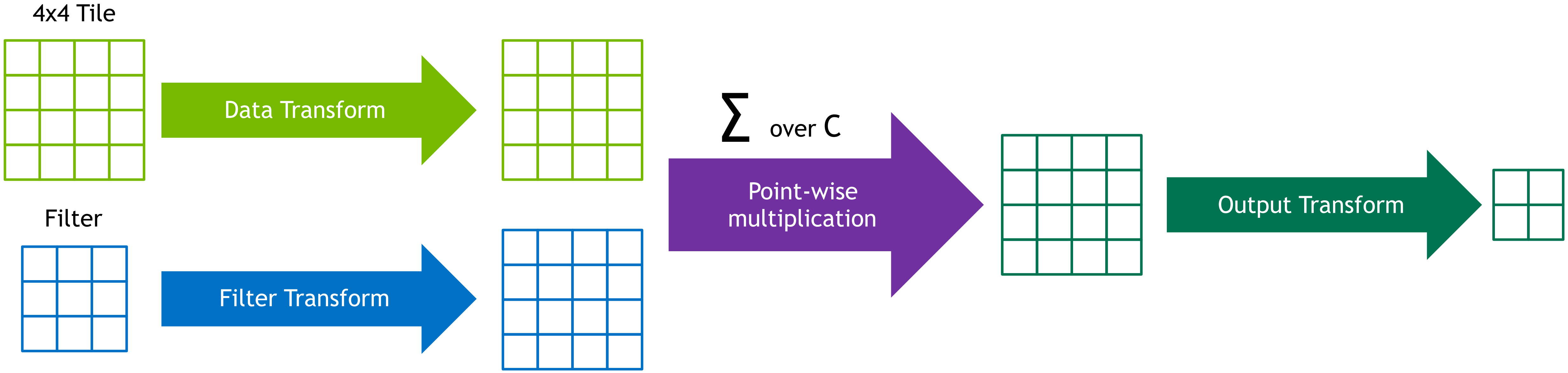
$$\begin{array}{lll} \boxed{4} \times \boxed{0} & \boxed{4} \times \boxed{1} & \boxed{4} \times \boxed{2} \\ \boxed{4} \times \boxed{3} & \boxed{4} \times \boxed{4} & \boxed{4} \times \boxed{5} \\ \boxed{4} \times \boxed{6} & \boxed{4} \times \boxed{7} & \boxed{4} \times \boxed{8} \end{array}$$

Direct convolution: we need 9xCx4 = 36xC FMAs for 4 outputs

Julien Demouth, Convolution OPTIMIZATION: Winograd, NVIDIA

3x3 WINOGRAD Convolutions

Transform Data to Reduce Math Intensity



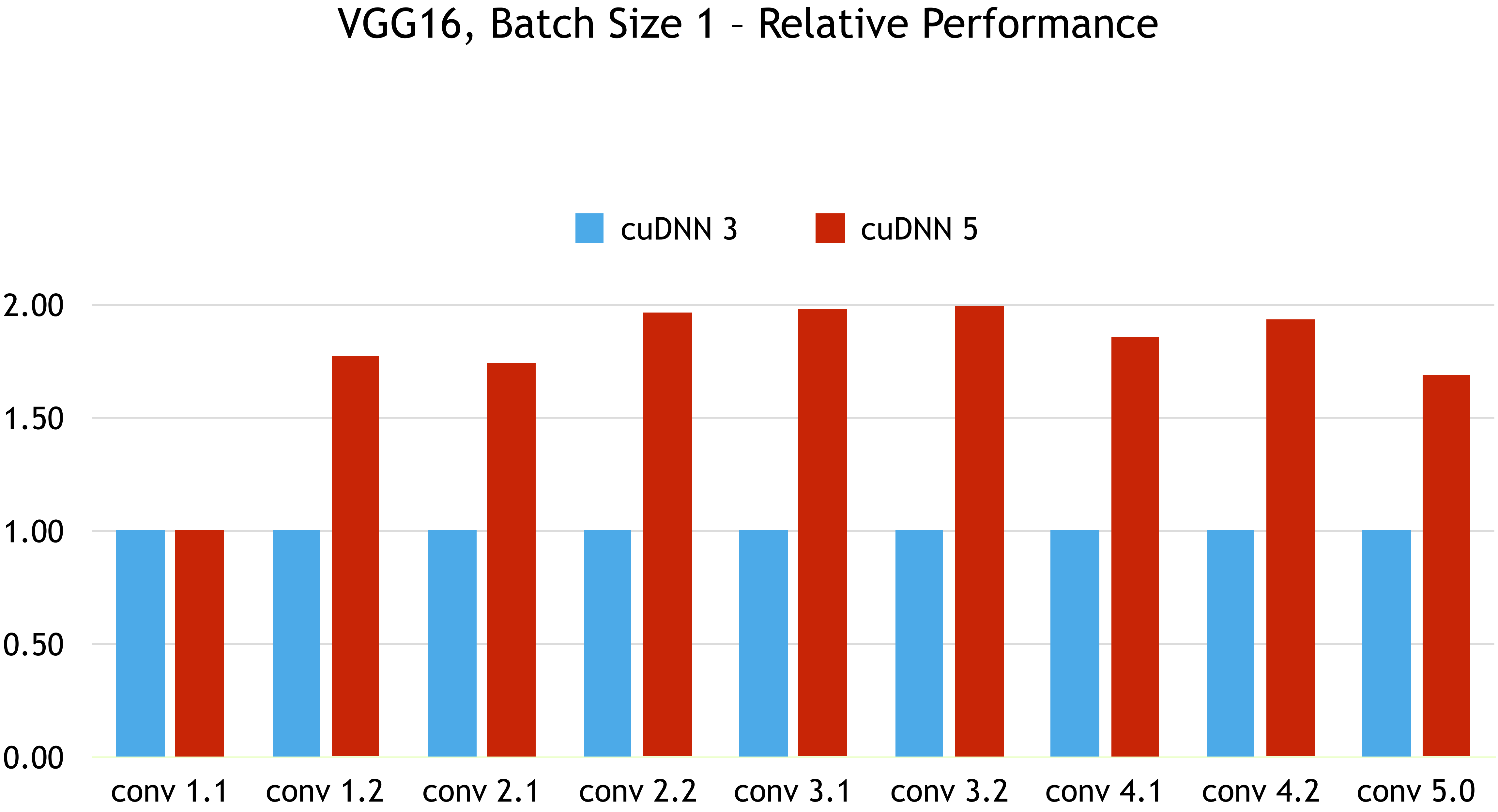
Direct convolution: we need 9xCx4 = 36xC FMAs for 4 outputs

Winograd convolution: we need 16xC FMAs for 4 outputs: **2.25x** fewer FMAs

See A. Lavin & S. Gray, "Fast Algorithms for Convolutional Neural Networks

Julien Demouth, Convolution OPTIMIZATION: Winograd, NVIDIA

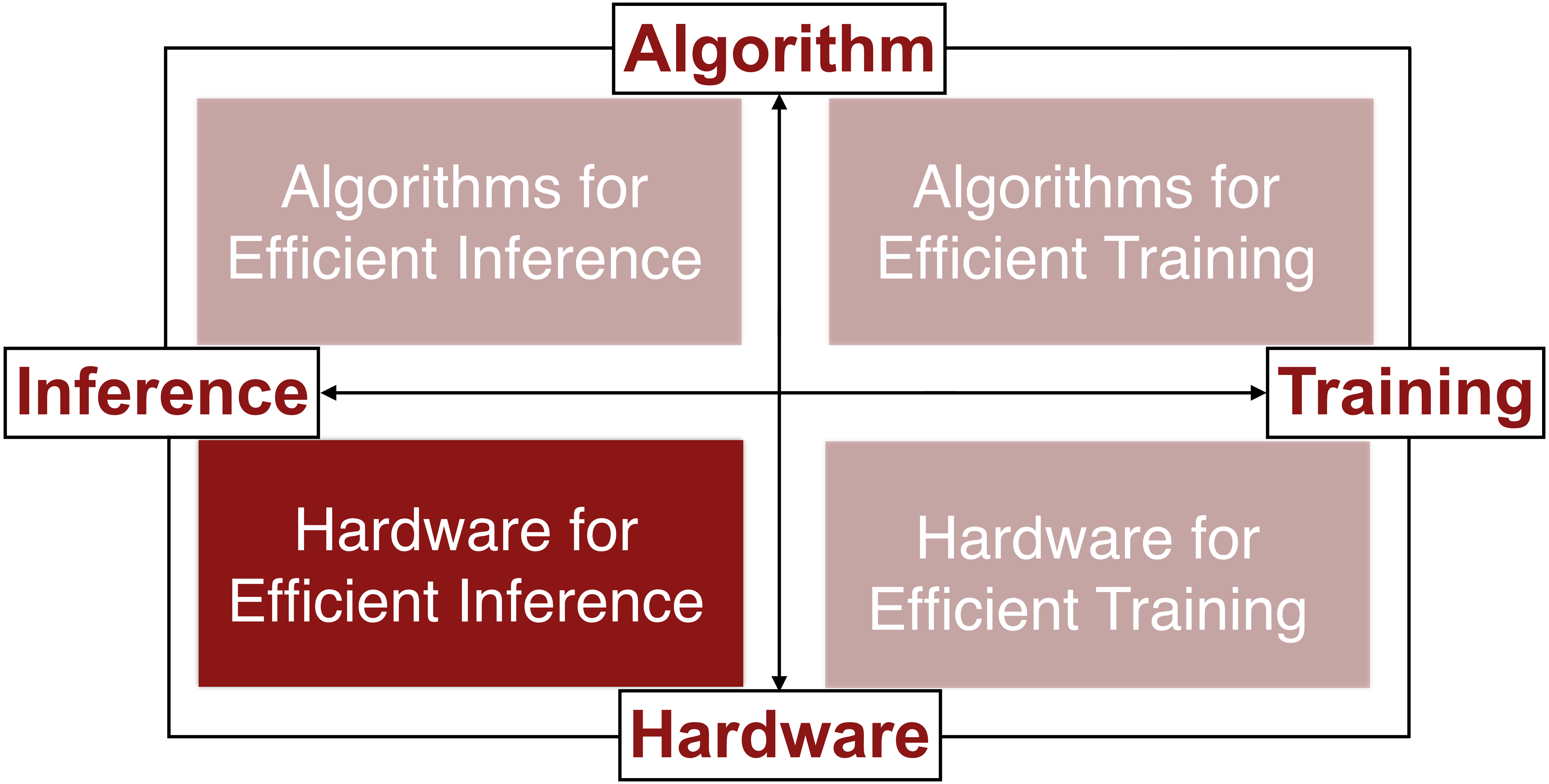
Speedup of Winograd Convolution



Measured on Maxwell TITAN X

Julien Demouth, Convolution OPTIMIZATION: Winograd, NVIDIA

Agenda



Algorithm

Algorithms for Efficient Inference

Algorithms for Efficient Training

Inference

Training

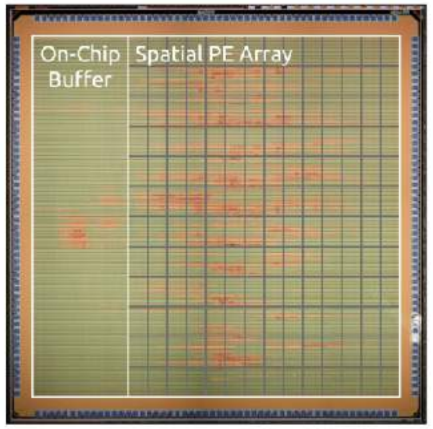
Hardware for Efficient Inference

Hardware for Efficient Training

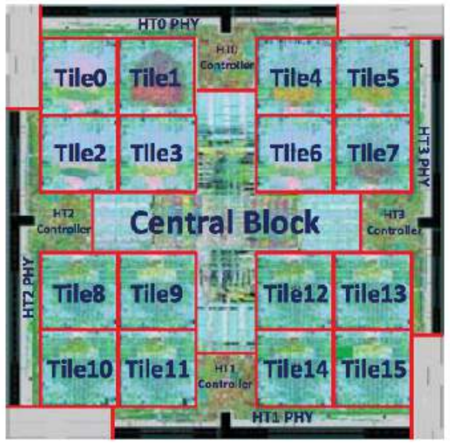
Hardware

Hardware for Efficient Inference

a common goal: minimize memory access



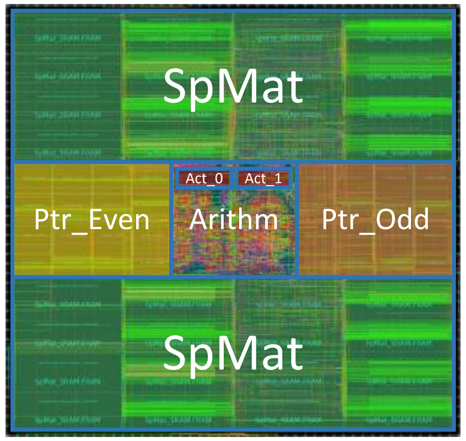
Eyeriss
MIT
RS Dataflow



DaDiannao
CAS
eDRAM

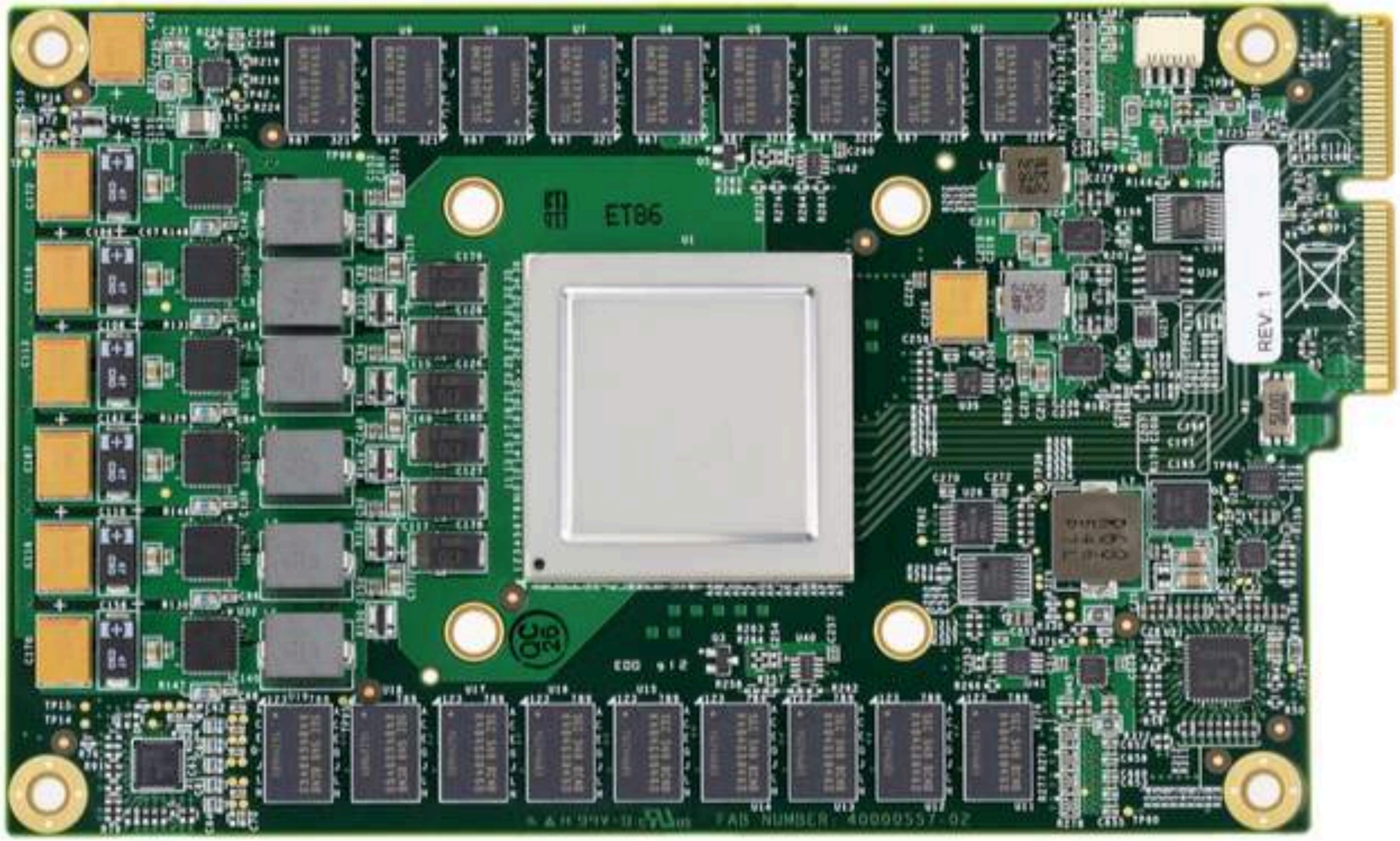
TPU
Google
8-bit Integer

"This unit is designed for dense matrices. Sparse architectural support was omitted for time-to-deploy reasons. Sparsity will have high priority in future designs"



EIE
Stanford
Compression/
Sparsity

Google TPU



TPU Card to replace a disk

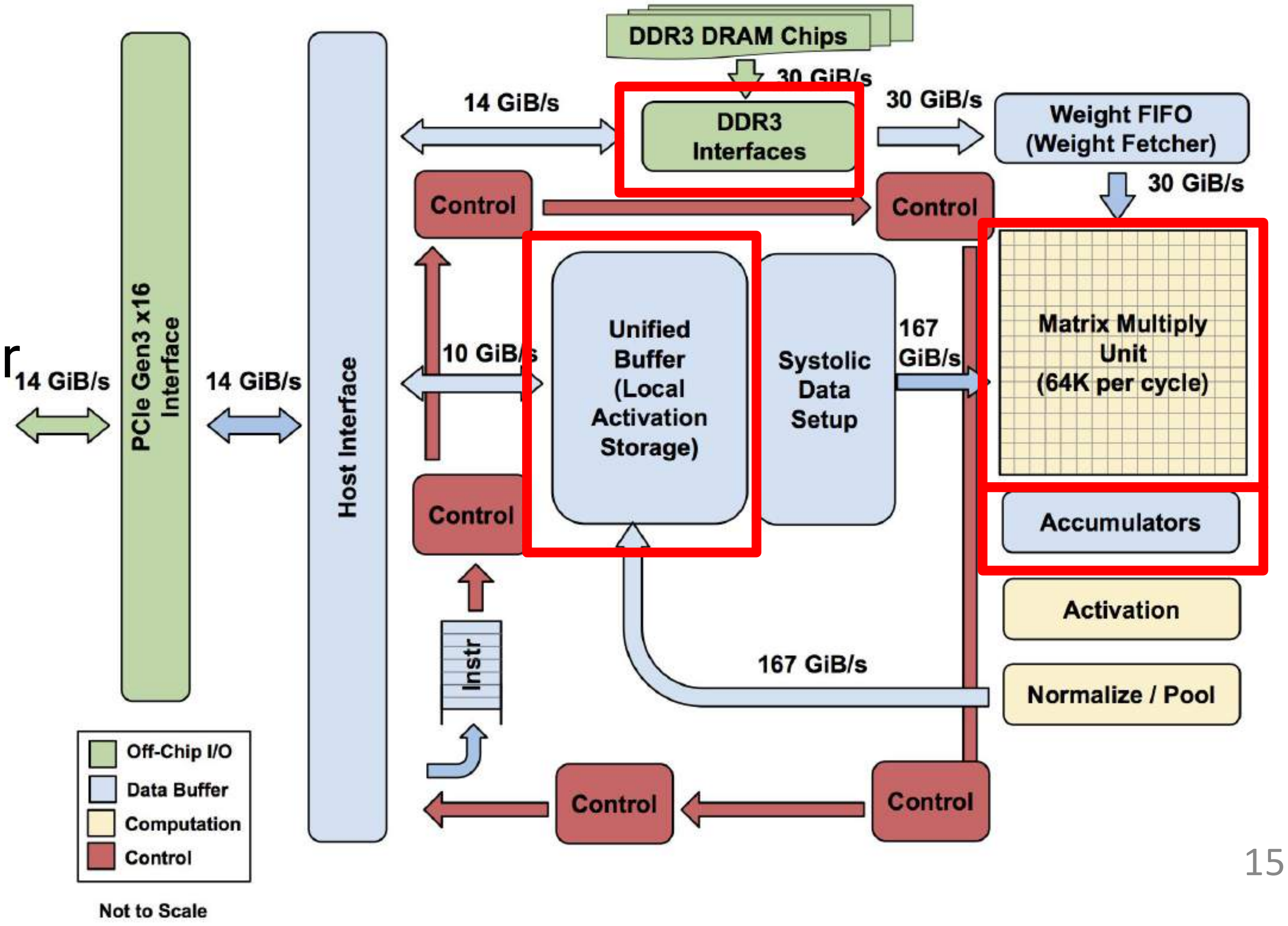
Up to 4 cards / server

David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

Google TPU

- The Matrix Unit: 65,536 (256x256) 8-bit multiply-accumulate units
- 700 MHz clock rate
- Peak: 92T operations/second
  - 65,536 * 2 * 700M
- >25X as many MACs vs GPU
- >100X as many MACs vs CPU
- 4 MiB of on-chip Accumulator memory
- 24 MiB of on-chip Unified Buffer (activation memory)
- 3.5X as much on-chip memory vs GPU
- Two 2133MHz DDR3 DRAM channels
- 8 GiB of off-chip weight DRAM memory

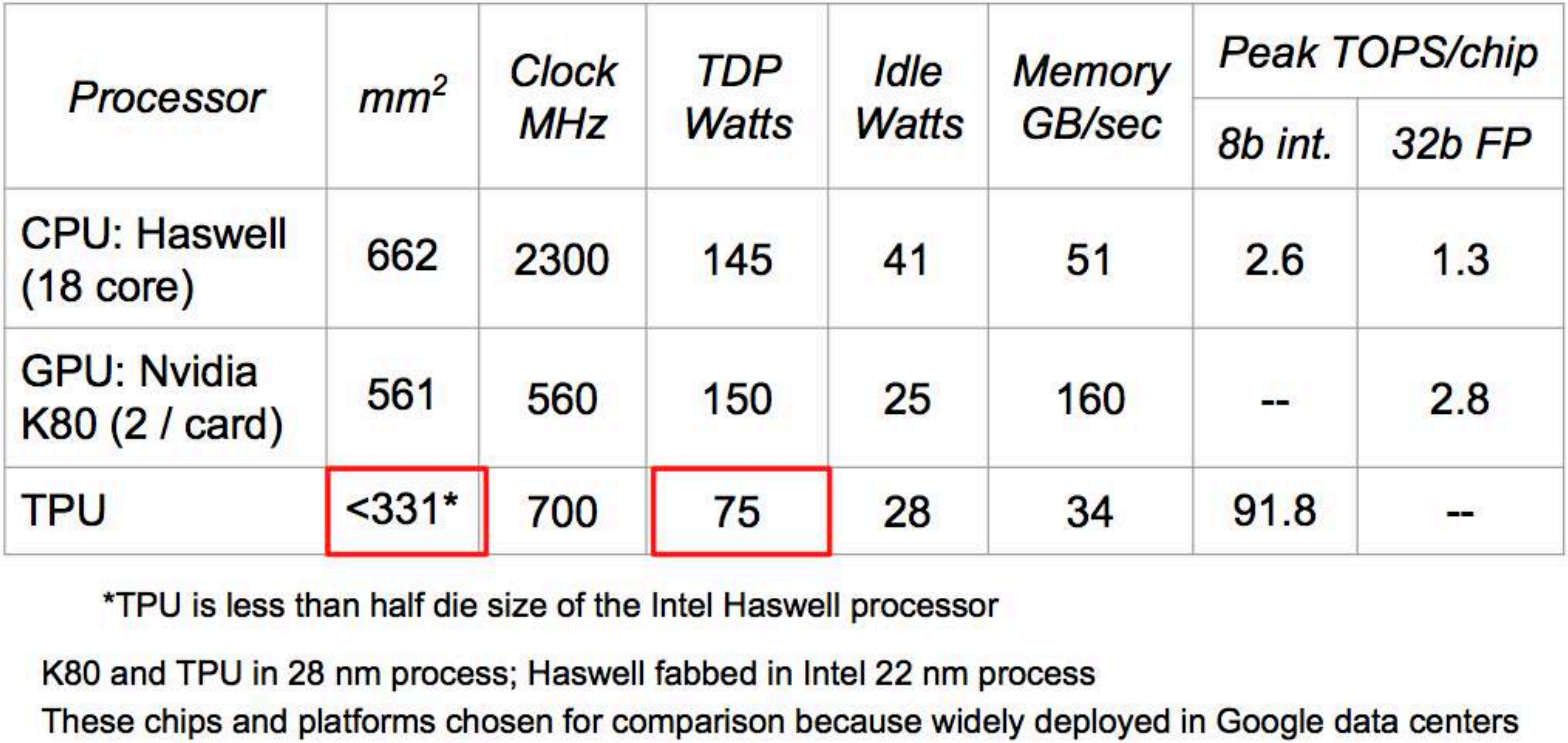
TPU: High-level Chip Architecture



15

David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

Google TPU



Processor	*mm^2*	*Clock MHz*	*TDP Watts*	*Idle Watts*	*Memory GB/sec*	*Peak TOPS/chip*	
						8b int.	*32b FP*
CPU: Haswell (18 core)	662	2300	145	41	51	2.6	1.3
GPU: Nvidia K80 (2 / card)	561	560	150	25	160	--	2.8
TPU	<331*	700	75	28	34	91.8	--

*TPU is less than half die size of the Intel Haswell processor

K80 and TPU in 28 nm process; Haswell fabbed in Intel 22 nm process

These chips and platforms chosen for comparison because widely deployed in Google data centers

David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

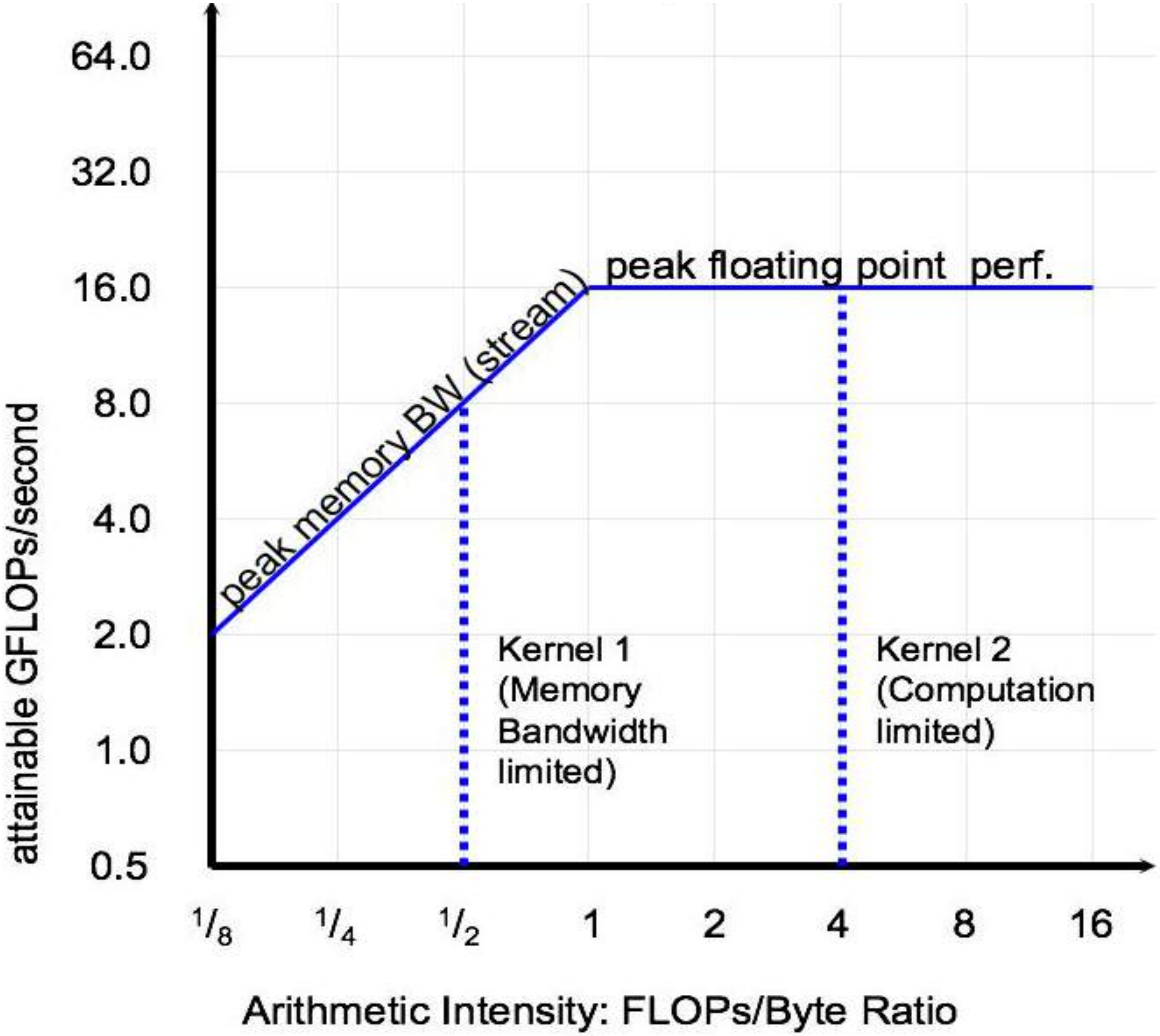
Inference Datacenter Workload

Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% Deployed*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	0.1k	5				5	ReLU	20M	200	200	61%
MLP1	1k	4				4	ReLU	5M	168	168	
LSTM0	1k	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1.5k	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1k		16			16	ReLU	8M	2888	8	5%
CNN1	1k	4	72		13	89	ReLU	100M	1750	32	

David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

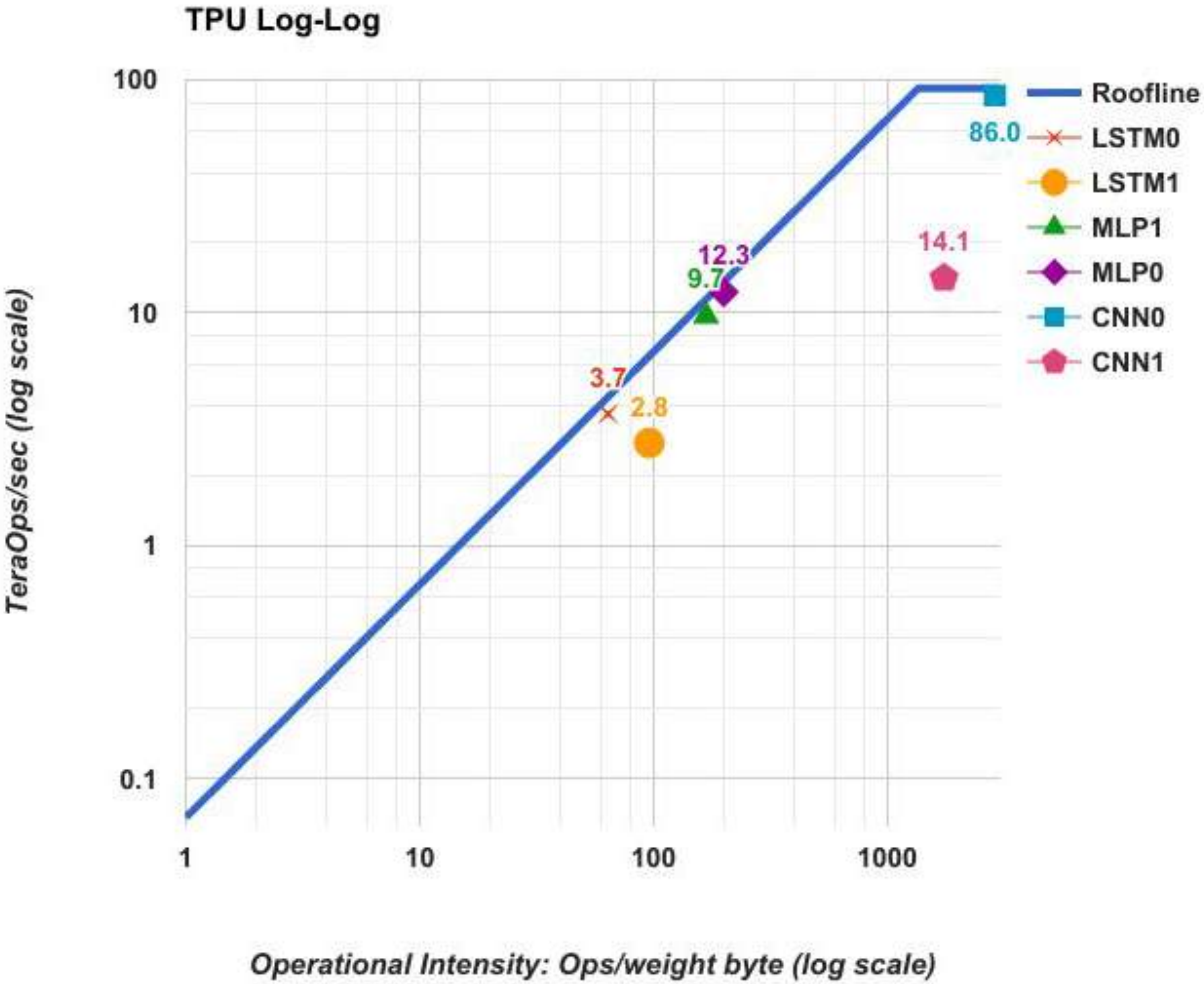
Roofline Model: Identify Performance Bottleneck

GFLOP/s = Min(Peak GFLOP/s, Peak GB/s x AI)



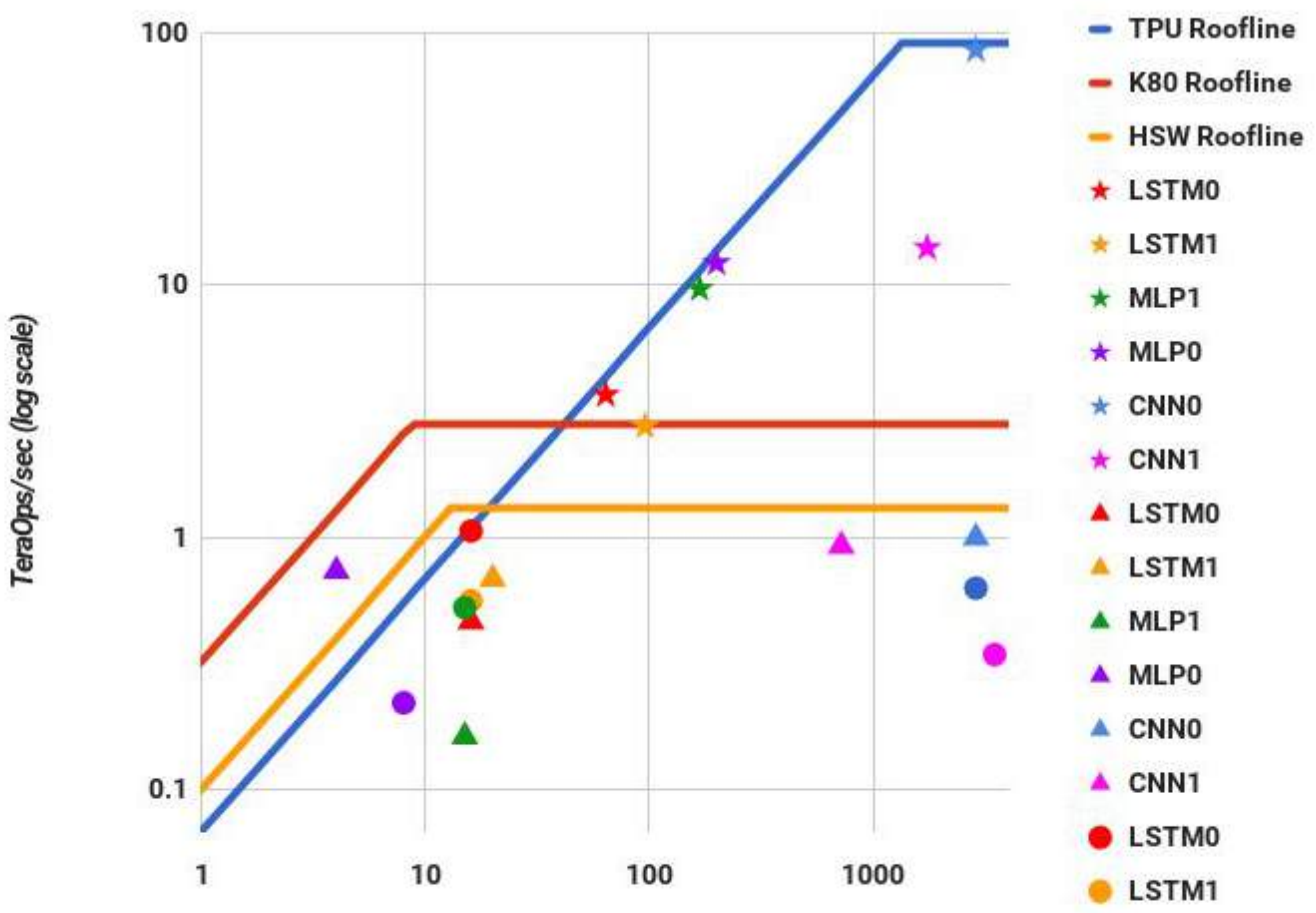
David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

TPU Roofline



David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

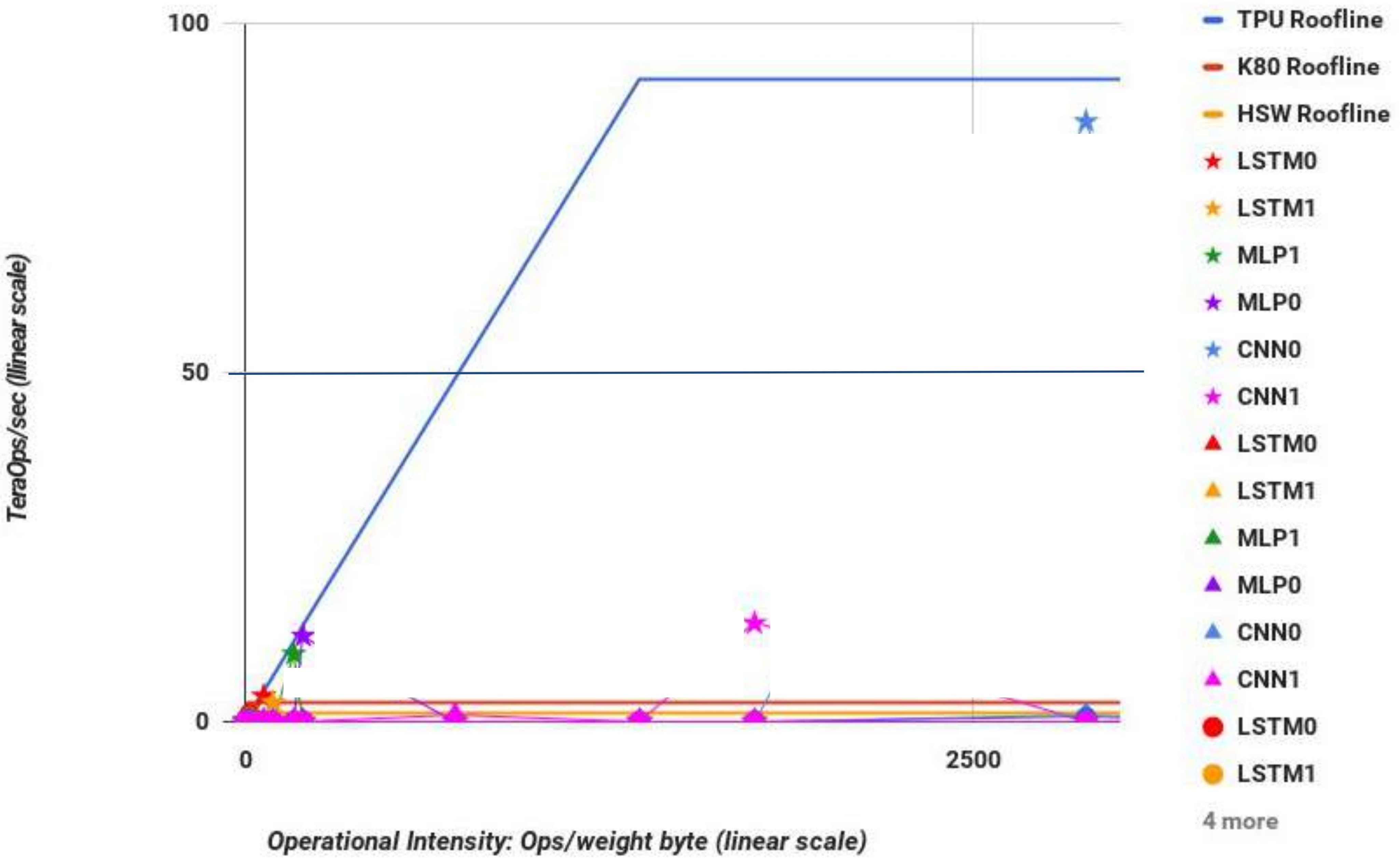
Log Rooflines for CPU, GPU, TPU



☆Star = TPU
△Triangle = GPU
○Circle = CPU

David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

Linear Rooflines for CPU, GPU, TPU



☆Star = TPU
△Triangle = GPU
○Circle = CPU

David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

Why so far below Rooflines?

Low latency requirement => Can’t batch more => low ops/byte

How to Solve this?

less memory footprint => need compress the model

Challenge:

Hardware that can infer on compressed model

[Han et al. ISCA'16]

EIE: the First DNN Accelerator for Sparse, Compressed Model

[Han et al. ISCA'16]

EIE: the First DNN Accelerator for Sparse, Compressed Model

0 * A = 0

Sparse Weight

90% *static* sparsity

W * 0 = 0

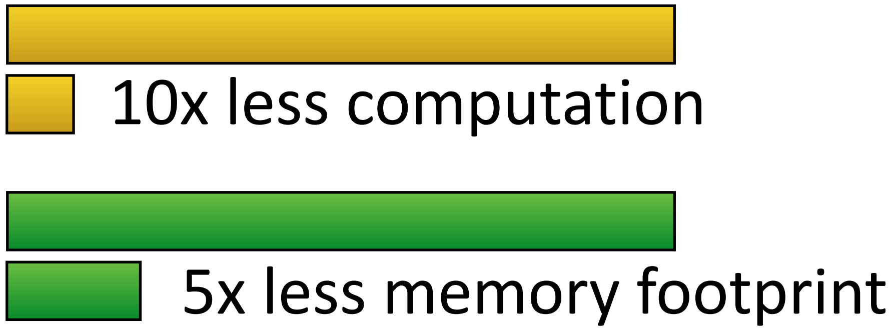
Sparse Activation

70% *dynamic* sparsity

~~2.09, 1.92~~=> 2

Weight Sharing

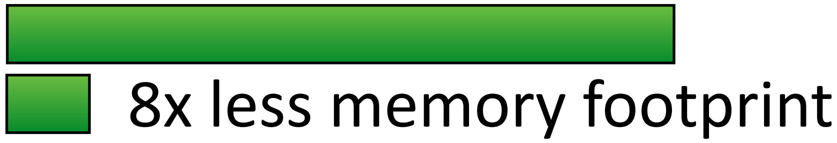
4-bit weights



10x less computation

5x less memory footprint

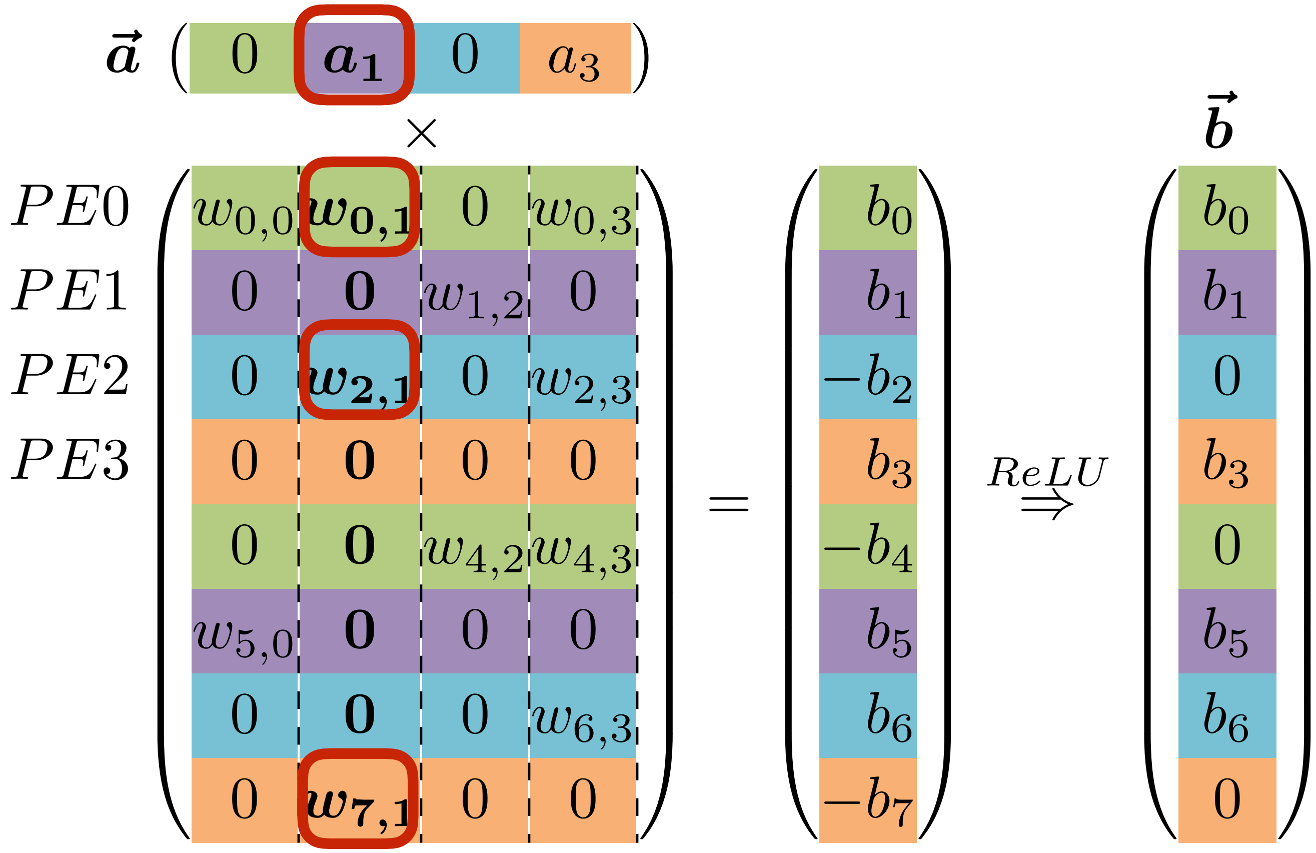
3x less computation



8x less memory footprint

EIE: Reduce Memory Access by Compression

logically



$$\vec{a} \; \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

$$\times$$

$$\begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \vec{b} \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

physically

Virtual Weight	$W_{0,0}$	$W_{0,1}$	$W_{4,2}$	$W_{0,3}$	$W_{4,3}$
Relative Index	0	1	2	0	0
Column Pointer	0	1	2	3	

Han et al. "EIE: Efficient Inference Engine on Compressed Deep Neural Network", ISCA 2016, Hotchips 2016

Dataflow

$$\vec{\boldsymbol{a}} \quad \begin{pmatrix} \boxed{0} & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

$$\times$$

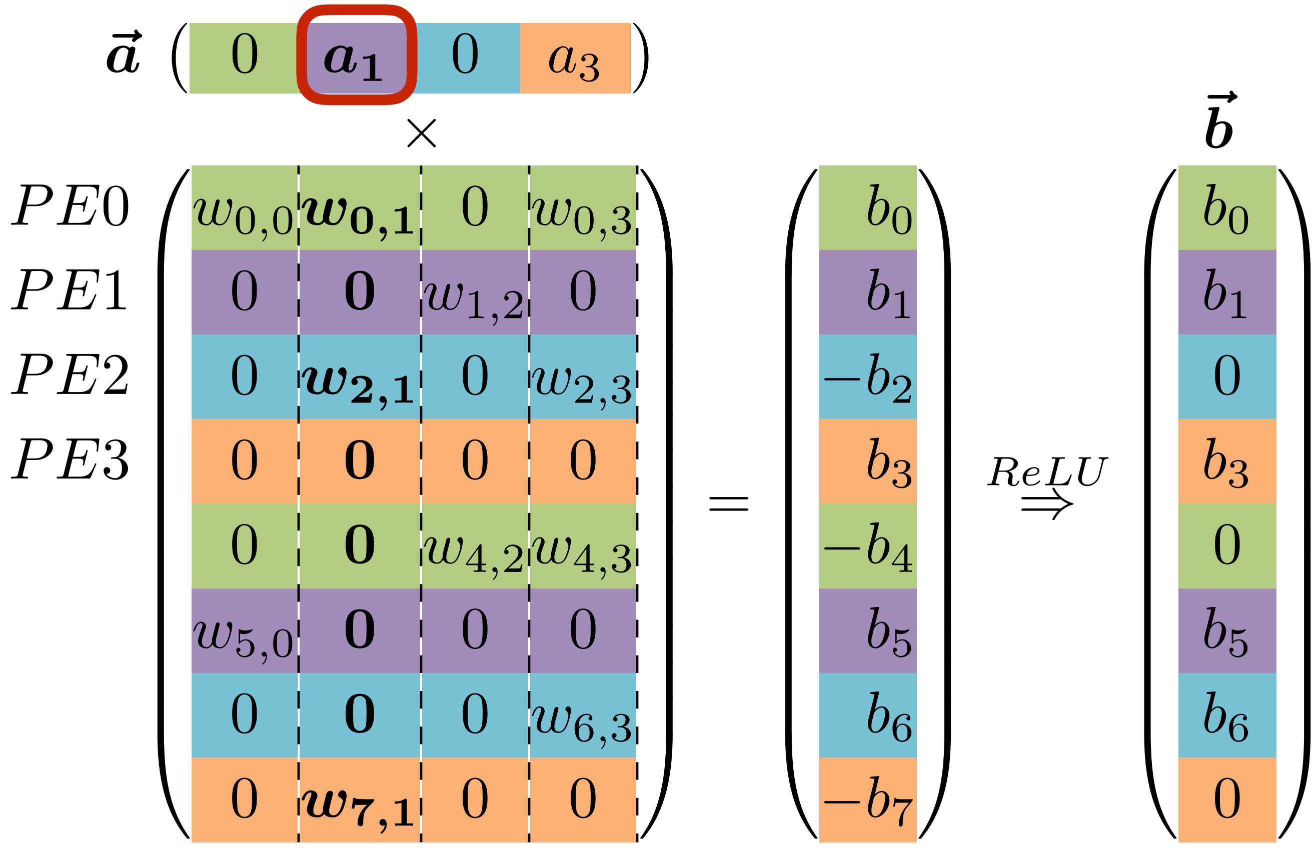
$$\begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \mathbf{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \mathbf{0} & 0 & 0 \\ 0 & \mathbf{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \mathbf{0} & 0 & 0 \\ 0 & \mathbf{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \overset{\vec{\boldsymbol{b}}}{\begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}}$$

rule of thumb:

0 * A = 0 W * 0 = 0

[Han et al. ISCA'16]

Dataflow



$$\vec{a} \; \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

$$\times$$

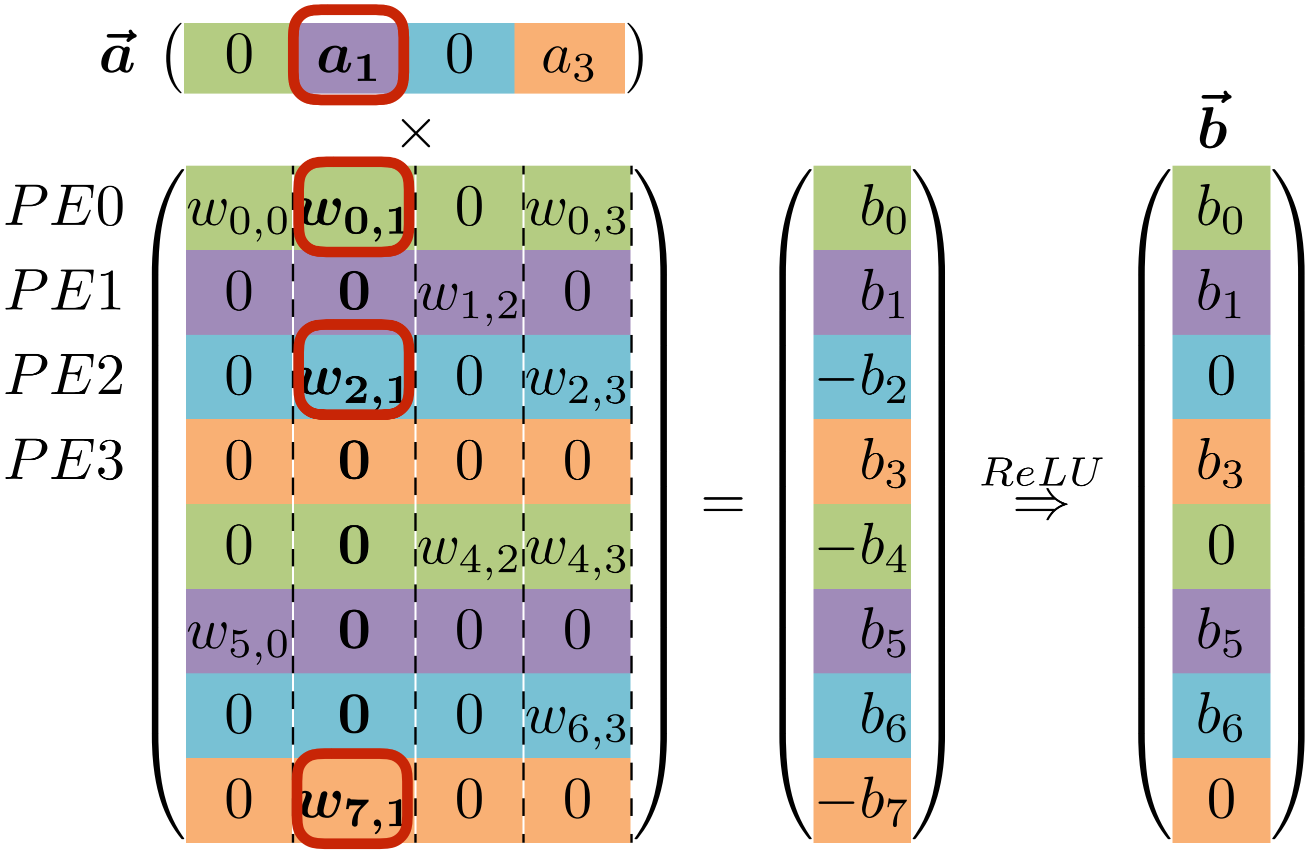
$$\begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \overset{\vec{b}}{\begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA'16]

Dataflow



$$\vec{a}\ \begin{pmatrix} 0 & \boxed{\boldsymbol{a_1}} & 0 & a_3 \end{pmatrix}$$

$$\times$$

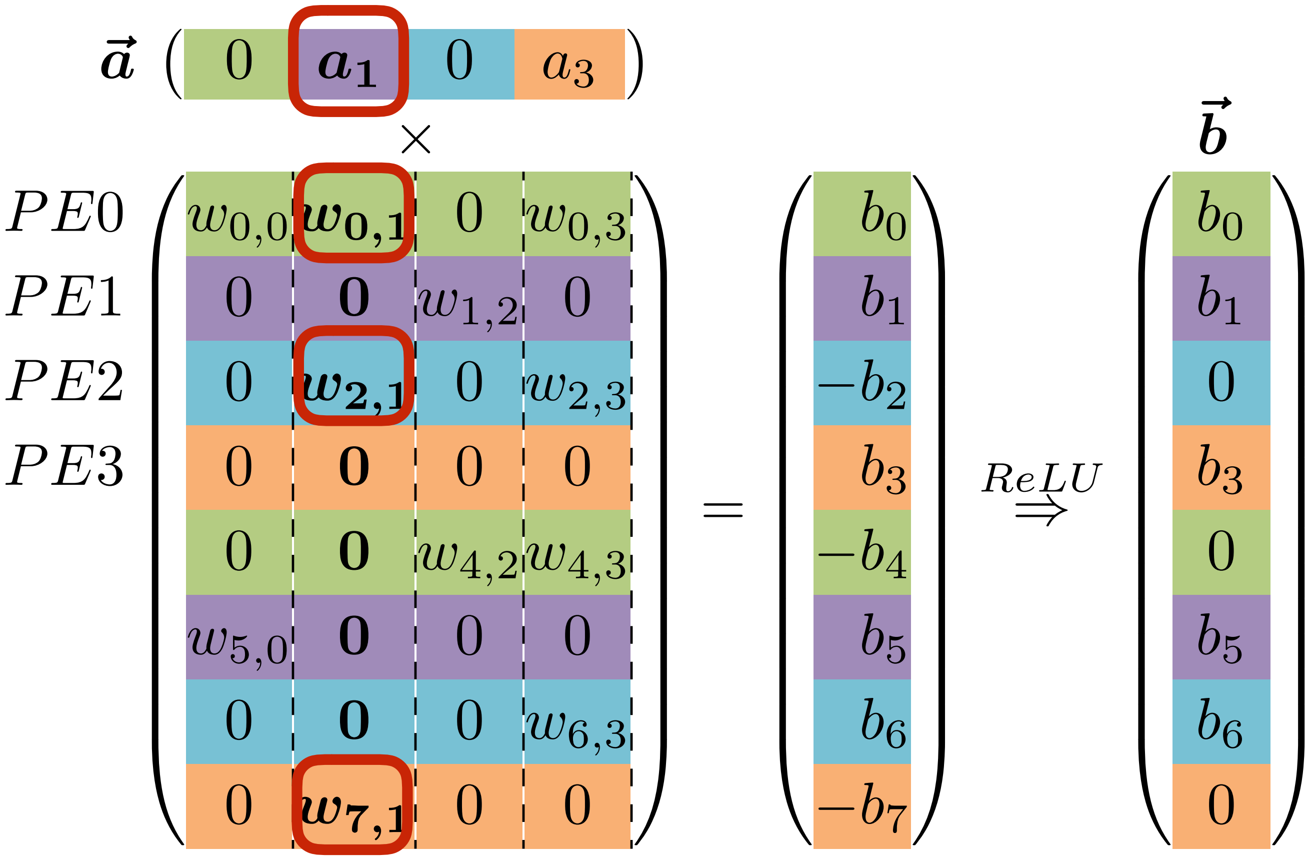
$$\begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \end{array}\begin{pmatrix} w_{0,0} & \boxed{\boldsymbol{w_{0,1}}} & 0 & w_{0,3} \\ 0 & \mathbf{0} & w_{1,2} & 0 \\ 0 & \boxed{\boldsymbol{w_{2,1}}} & 0 & w_{2,3} \\ 0 & \mathbf{0} & 0 & 0 \\ 0 & \mathbf{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \mathbf{0} & 0 & 0 \\ 0 & \mathbf{0} & 0 & w_{6,3} \\ 0 & \boxed{\boldsymbol{w_{7,1}}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \vec{b}\ \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA'16]

Dataflow



$$\vec{\boldsymbol{a}} \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

$$\times$$

$$\begin{array}{c|cccc} PE0 & w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ PE1 & 0 & \boldsymbol{0} & w_{1,2} & 0 \\ PE2 & 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ PE3 & 0 & \boldsymbol{0} & 0 & 0 \\ & 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ & w_{5,0} & \boldsymbol{0} & 0 & 0 \\ & 0 & \boldsymbol{0} & 0 & w_{6,3} \\ & 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{array} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \overset{\vec{\boldsymbol{b}}}{\begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA'16]

Dataflow

$$\vec{\boldsymbol{a}} \quad \begin{pmatrix} 0 & \boldsymbol{a_1} & \boxed{0} & a_3 \end{pmatrix}$$

$$\times$$

$$\begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \overset{\vec{\boldsymbol{b}}}{\begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA'16]

Dataflow

$$\vec{\boldsymbol{a}} \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & \boxed{a_3} \end{pmatrix}$$

$$\times$$

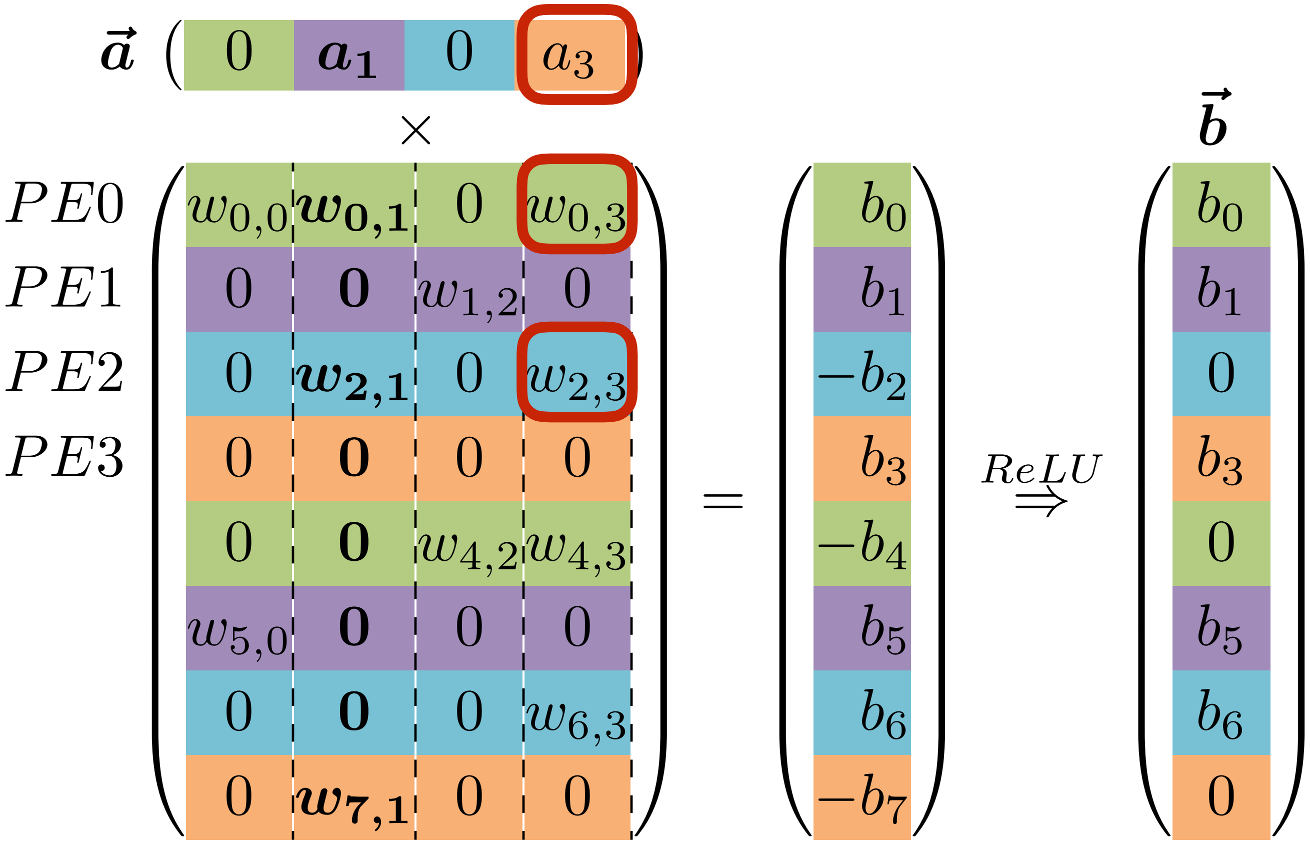
$$\begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \overset{\vec{\boldsymbol{b}}}{\begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA'16]

Dataflow



$$\vec{\boldsymbol{a}} \left(\begin{array}{cccc} 0 & \boldsymbol{a_1} & 0 & \boxed{a_3} \end{array} \right)$$

$$\times$$

$$\begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & \boxed{w_{0,3}} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & \boxed{w_{2,3}} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \begin{array}{c} \vec{\boldsymbol{b}} \\ \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix} \end{array}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA'16]

Dataflow

$$\vec{\boldsymbol{a}} \quad \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & \boxed{a_3} \end{pmatrix}$$

$$\times$$

$$\begin{array}{r|cccc} PE0 & w_{0,0} & \boldsymbol{w_{0,1}} & 0 & \boxed{w_{0,3}} \\ PE1 & 0 & \boldsymbol{0} & w_{1,2} & 0 \\ PE2 & 0 & \boldsymbol{w_{2,1}} & 0 & \boxed{w_{2,3}} \\ PE3 & 0 & \boldsymbol{0} & 0 & 0 \\ & 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ & w_{5,0} & \boldsymbol{0} & 0 & 0 \\ & 0 & \boldsymbol{0} & 0 & w_{6,3} \\ & 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{array} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \overset{\vec{\boldsymbol{b}}}{\begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA'16]

Dataflow

$$\vec{\boldsymbol{a}} \; \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & \boxed{a_3} \end{pmatrix}$$

$$\times$$

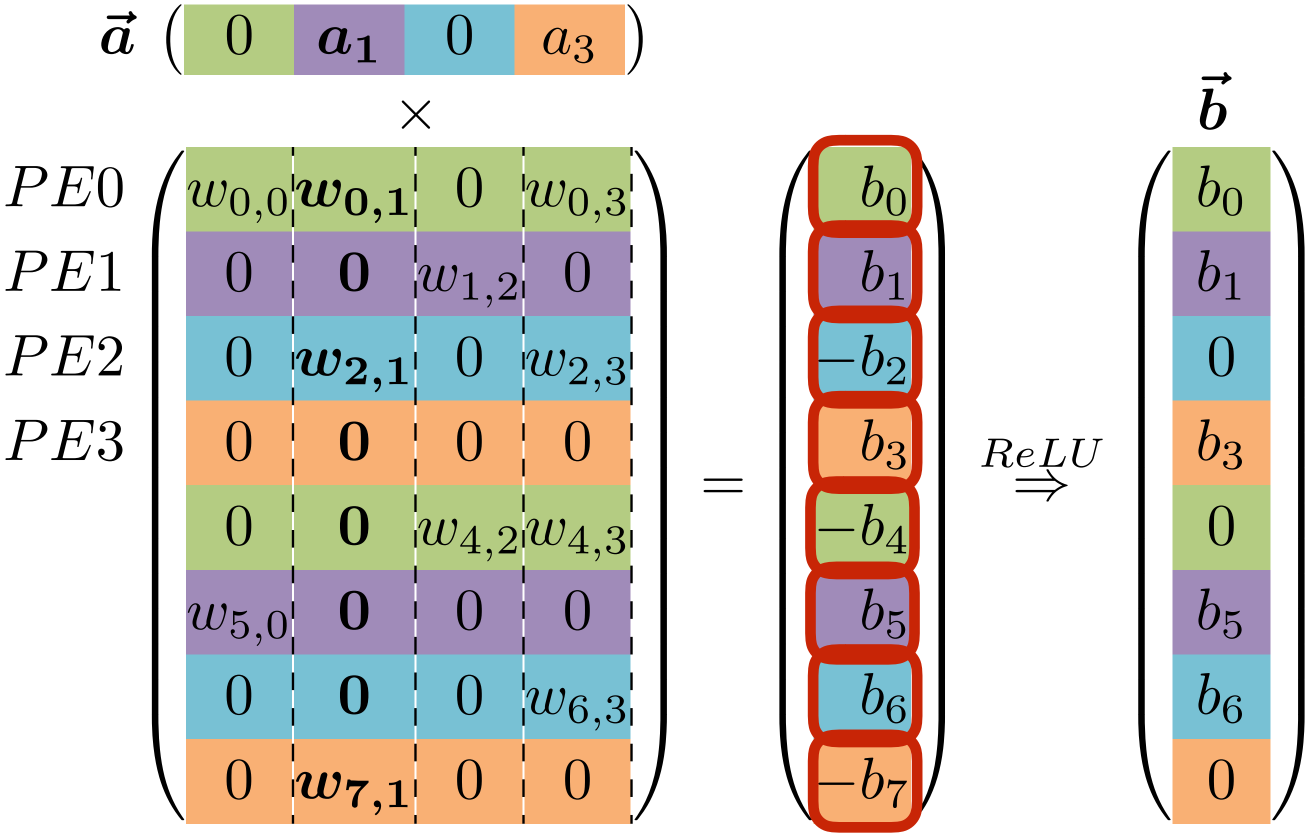
$$\begin{array}{cc}
PE0 & \\
PE1 & \\
PE2 & \\
PE3 & \\
 & \\
 & \\
 & \\
 &
\end{array}
\begin{pmatrix}
w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\
0 & \mathbf{0} & w_{1,2} & 0 \\
0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\
0 & \mathbf{0} & 0 & 0 \\
0 & \mathbf{0} & w_{4,2} & \boxed{w_{4,3}} \\
w_{5,0} & \mathbf{0} & 0 & 0 \\
0 & \mathbf{0} & 0 & \boxed{w_{6,3}} \\
0 & \boldsymbol{w_{7,1}} & 0 & 0
\end{pmatrix}
=
\begin{pmatrix}
b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7
\end{pmatrix}
\overset{ReLU}{\Longrightarrow}
\overset{\vec{\boldsymbol{b}}}{\begin{pmatrix}
b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0
\end{pmatrix}}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA’16]

Dataflow



$$\vec{a} \; \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

$$\times$$

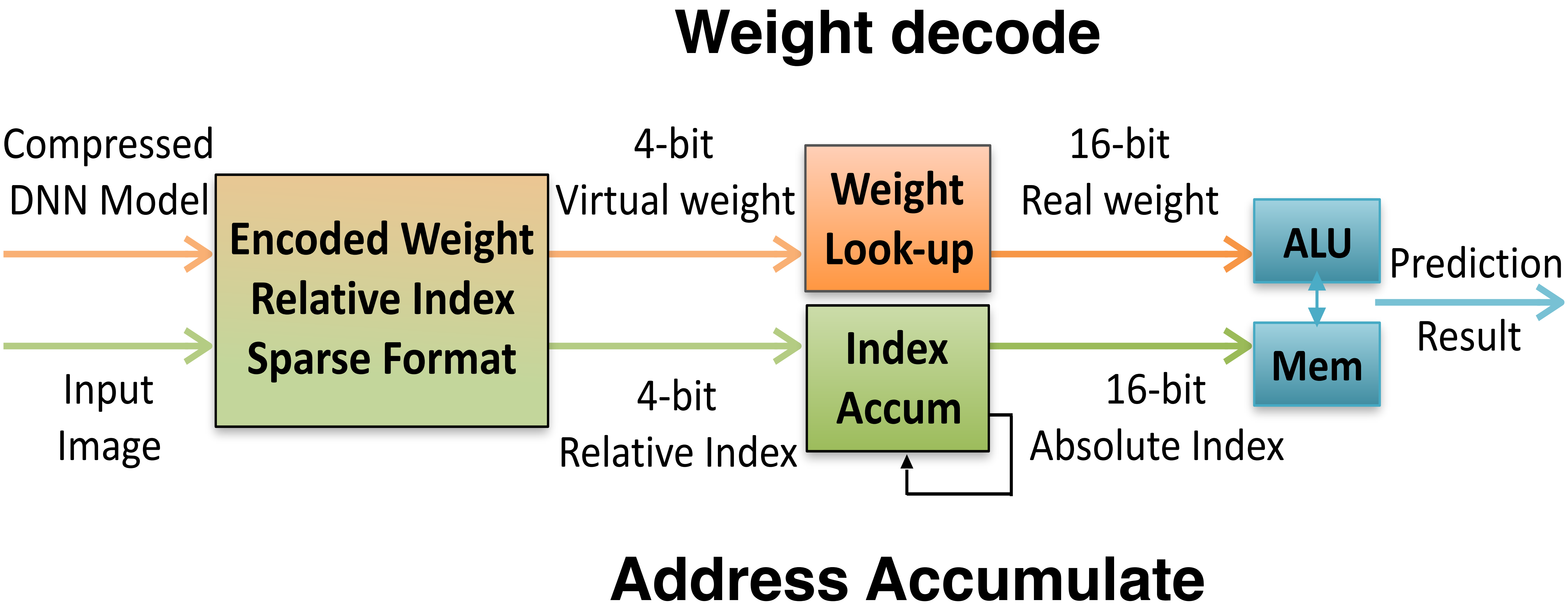
$$\begin{array}{cc} PE0 & \\ PE1 & \\ PE2 & \\ PE3 & \\ & \\ & \\ & \\ & \end{array} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \overset{\vec{b}}{\begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}}$$

rule of thumb:

0 * A = 0   W * 0 = 0

[Han et al. ISCA'16]

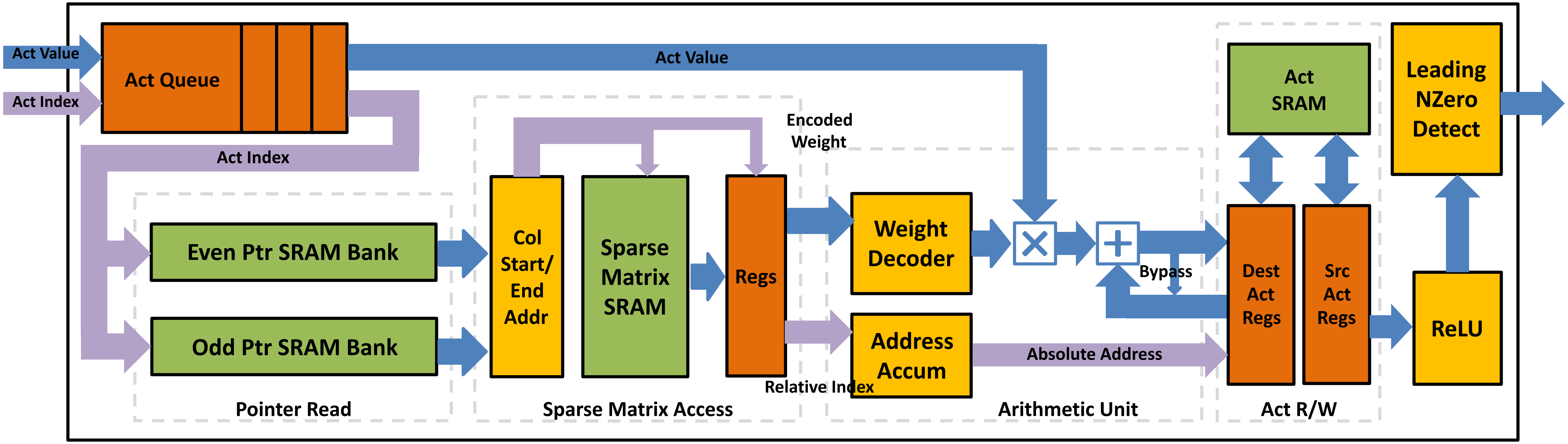
EIE Architecture

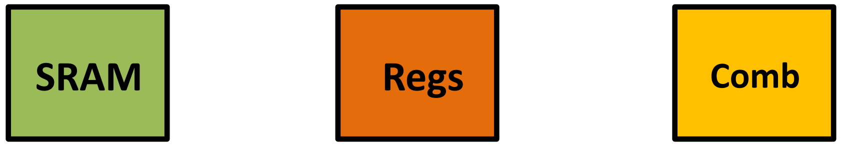


rule of thumb: 0 * A = 0 W * 0 = 0 ~~2.09, 1.92~~=> 2

[Han et al. ISCA'16]

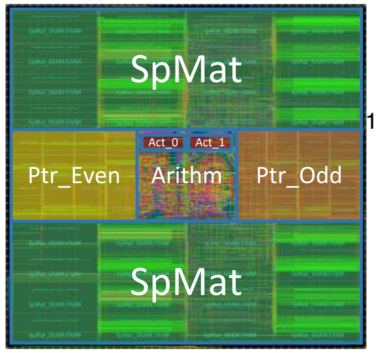
Micro Architecture for each PE

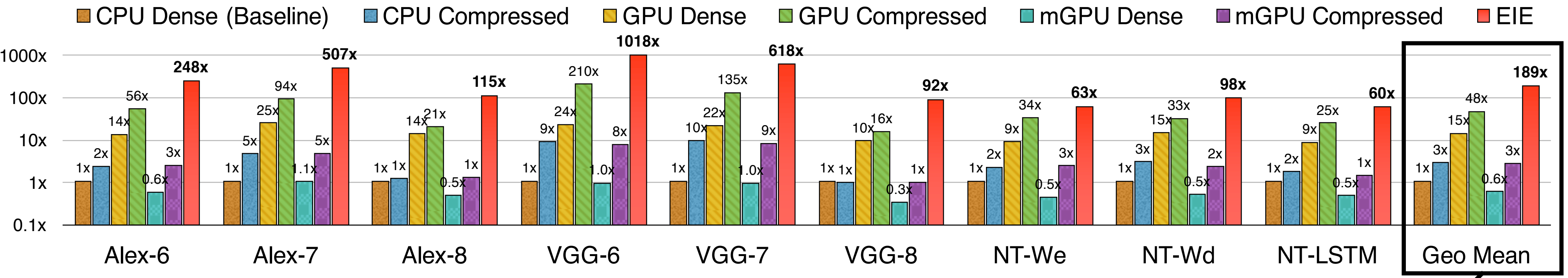


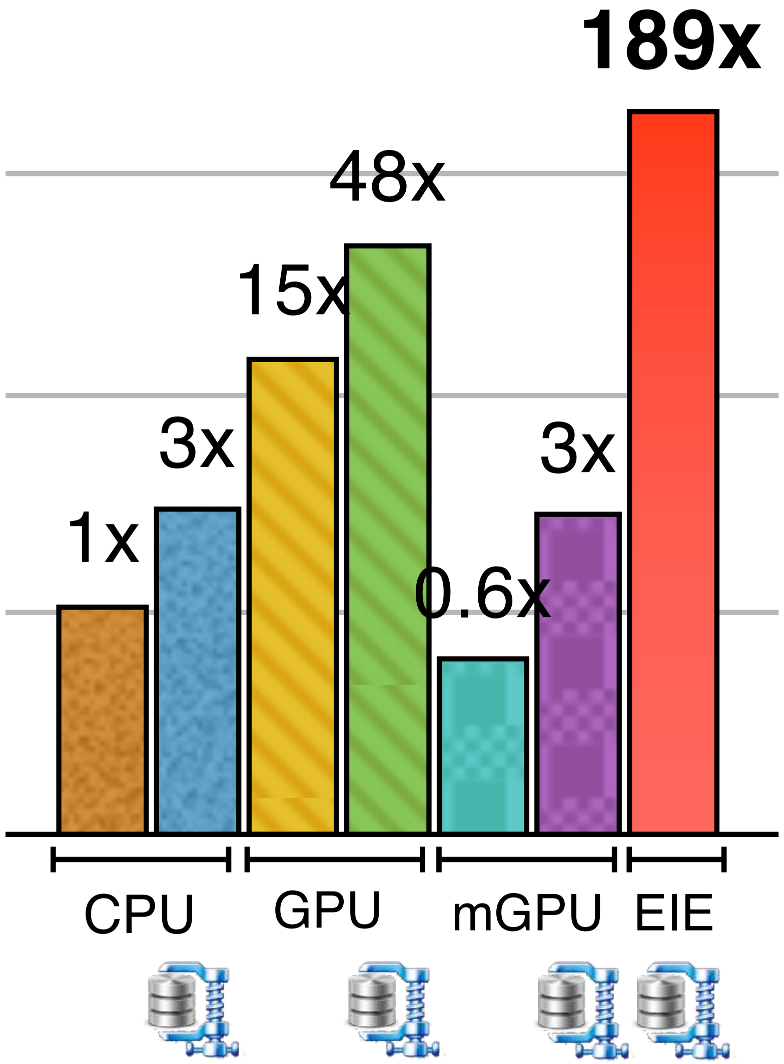


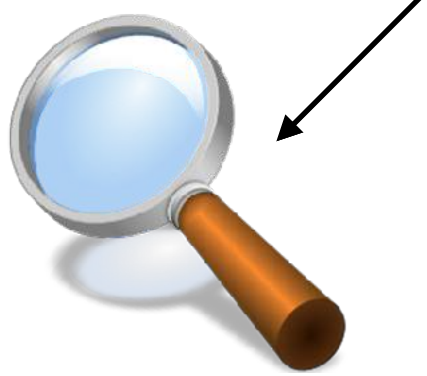
[Han et al. ISCA'16]

Speedup on EIE





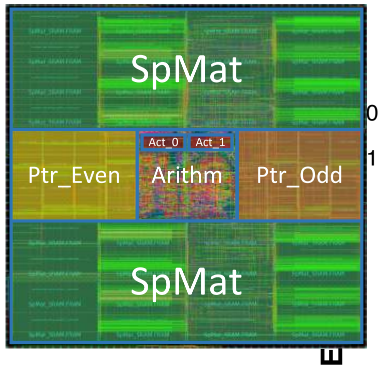


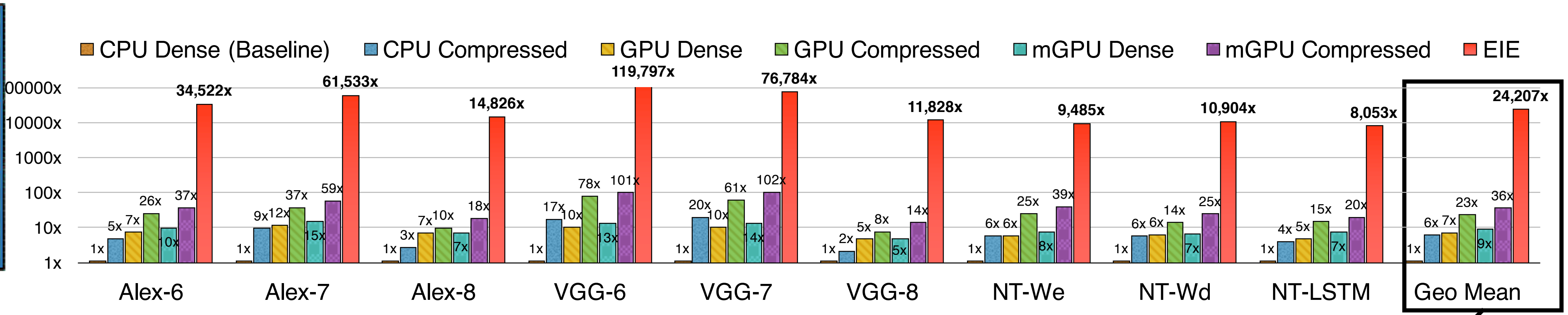


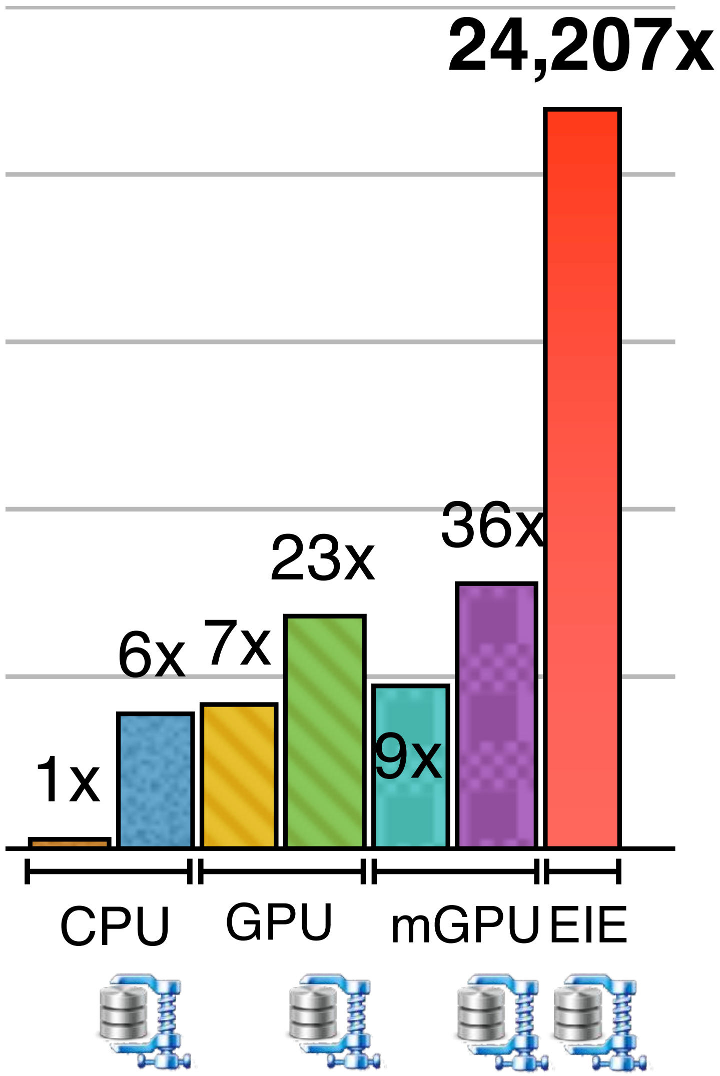
Geo Mean

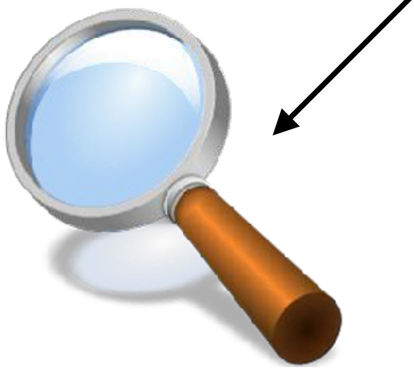
[Han et al. ISCA'16]

Energy Efficiency on EIE





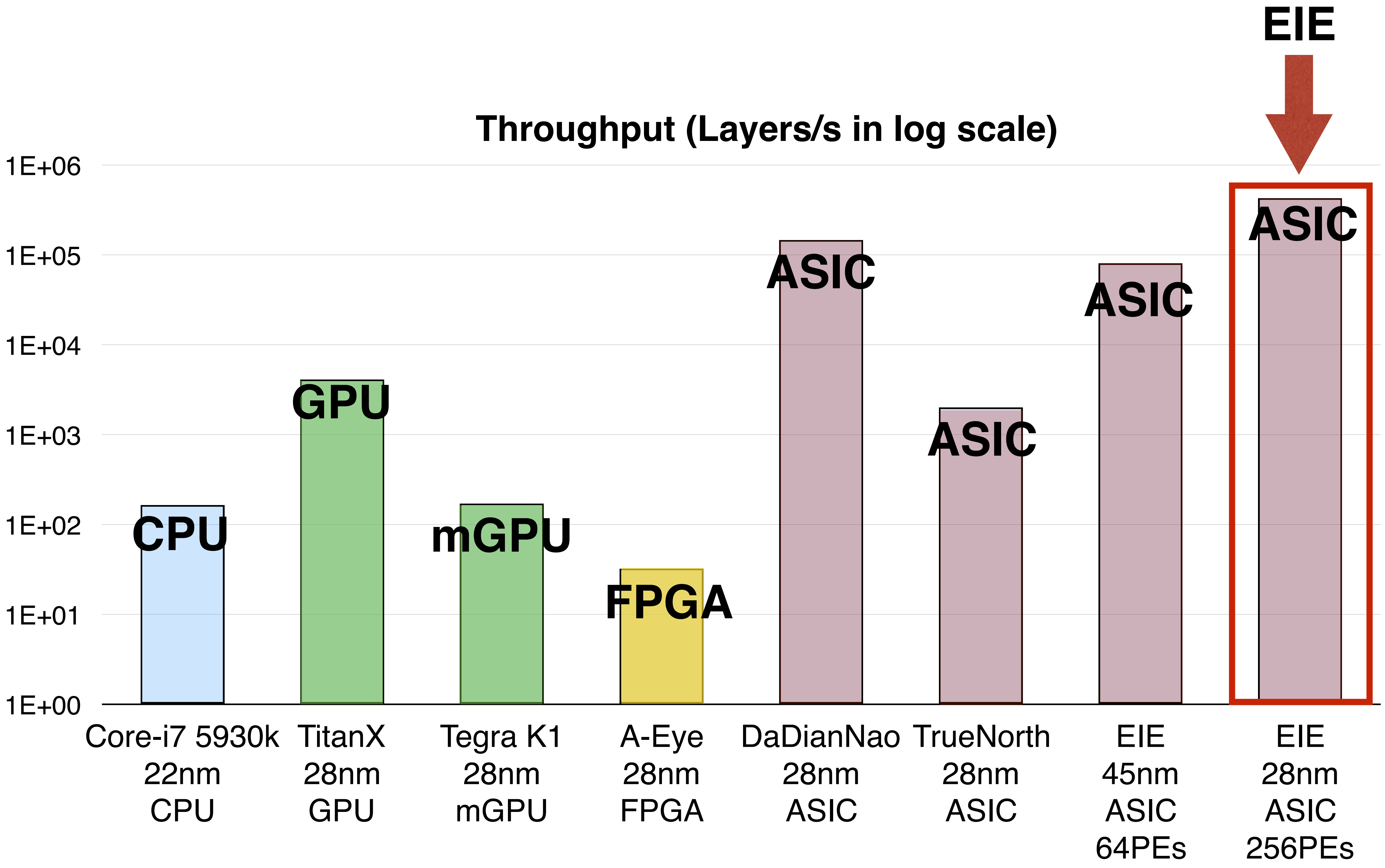




Geo Mean

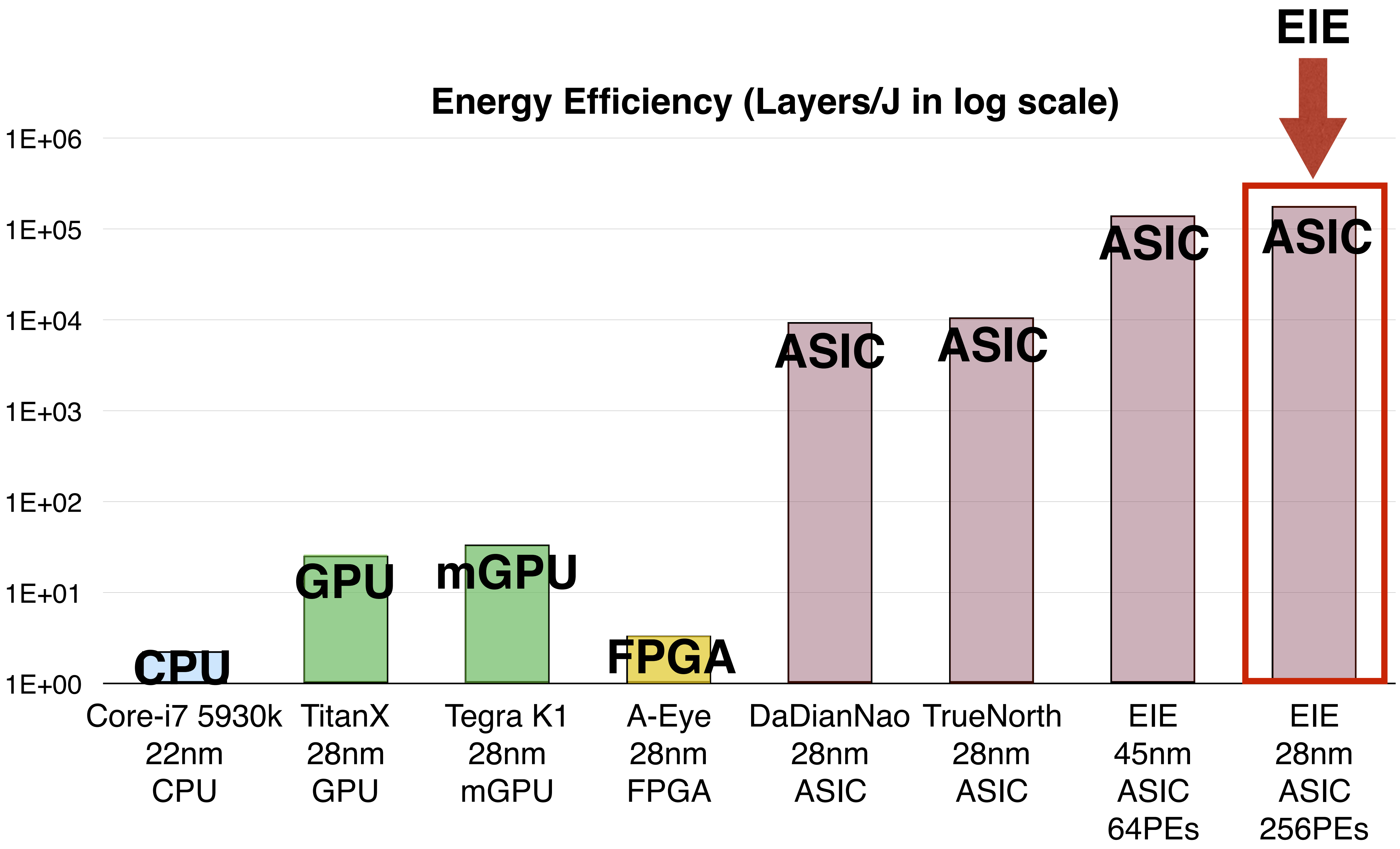
[Han et al. ISCA'16]

Comparison: Throughput

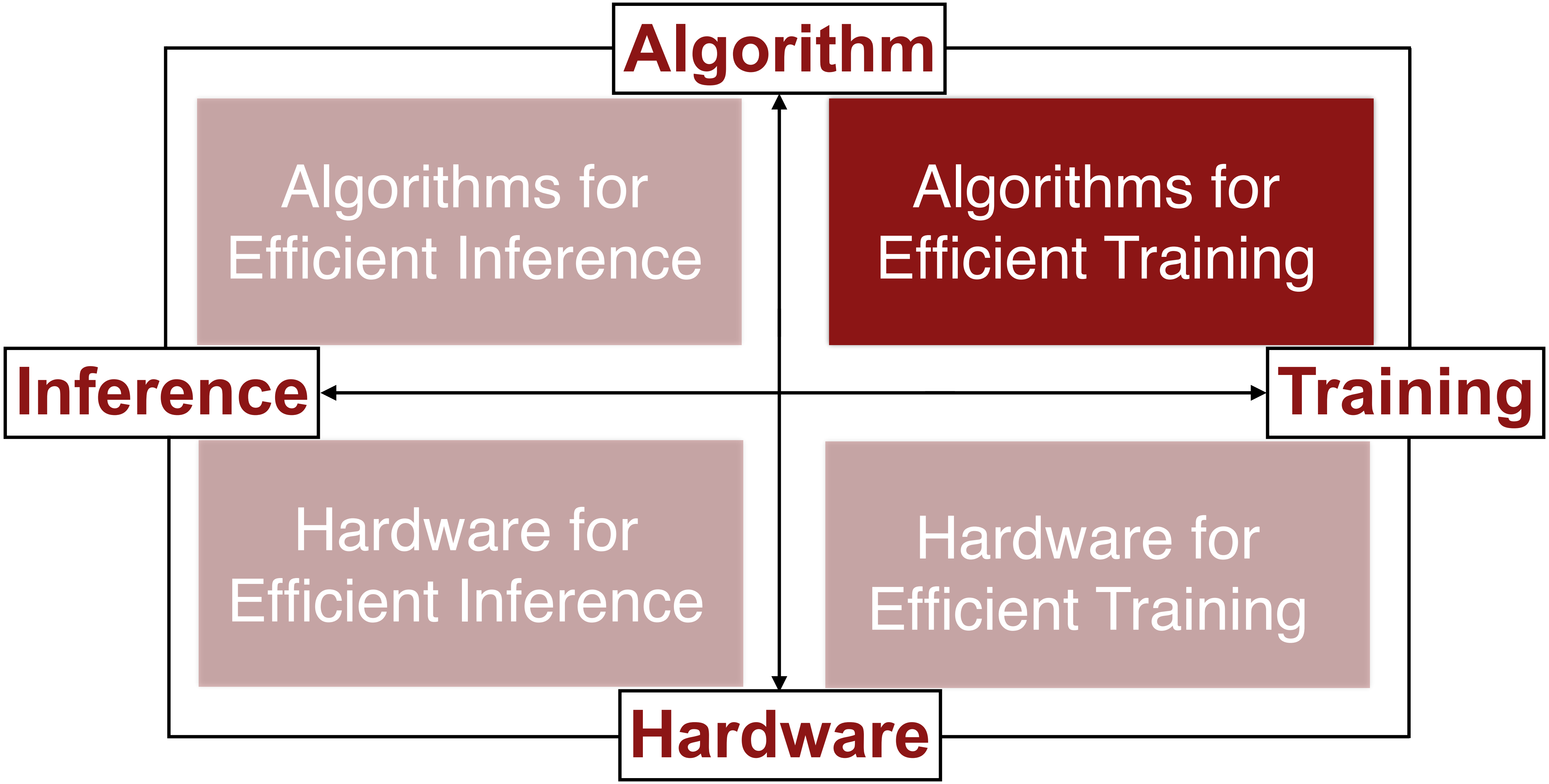


[Han et al. ISCA'16]

Comparison: Energy Efficiency



Agenda



Algorithm

Algorithms for Efficient Inference

Algorithms for Efficient Training

Inference

Training

Hardware for Efficient Inference

Hardware for Efficient Training

Hardware

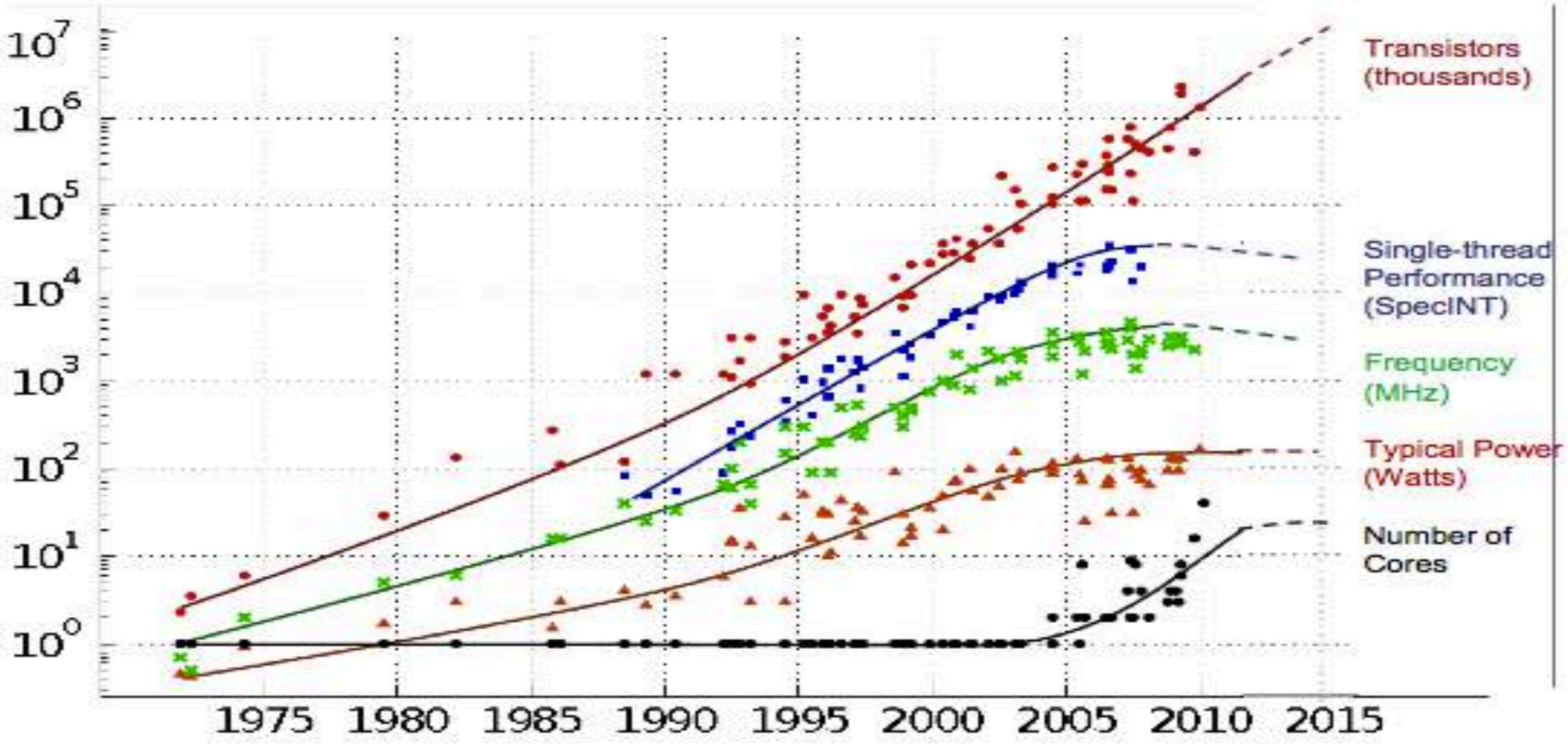
Part 3: Efficient Training — Algorithms

- 1. Parallelization
- 2. Mixed Precision with FP16 and FP32
- 3. Model Distillation
- 4. DSD: Dense-Sparse-Dense Training

Part 3: Efficient Training — Algorithms

- 1. Parallelization
- 2. Mixed Precision with FP16 and FP32
- 3. Model Distillation
- 4. DSD: Dense-Sparse-Dense Training

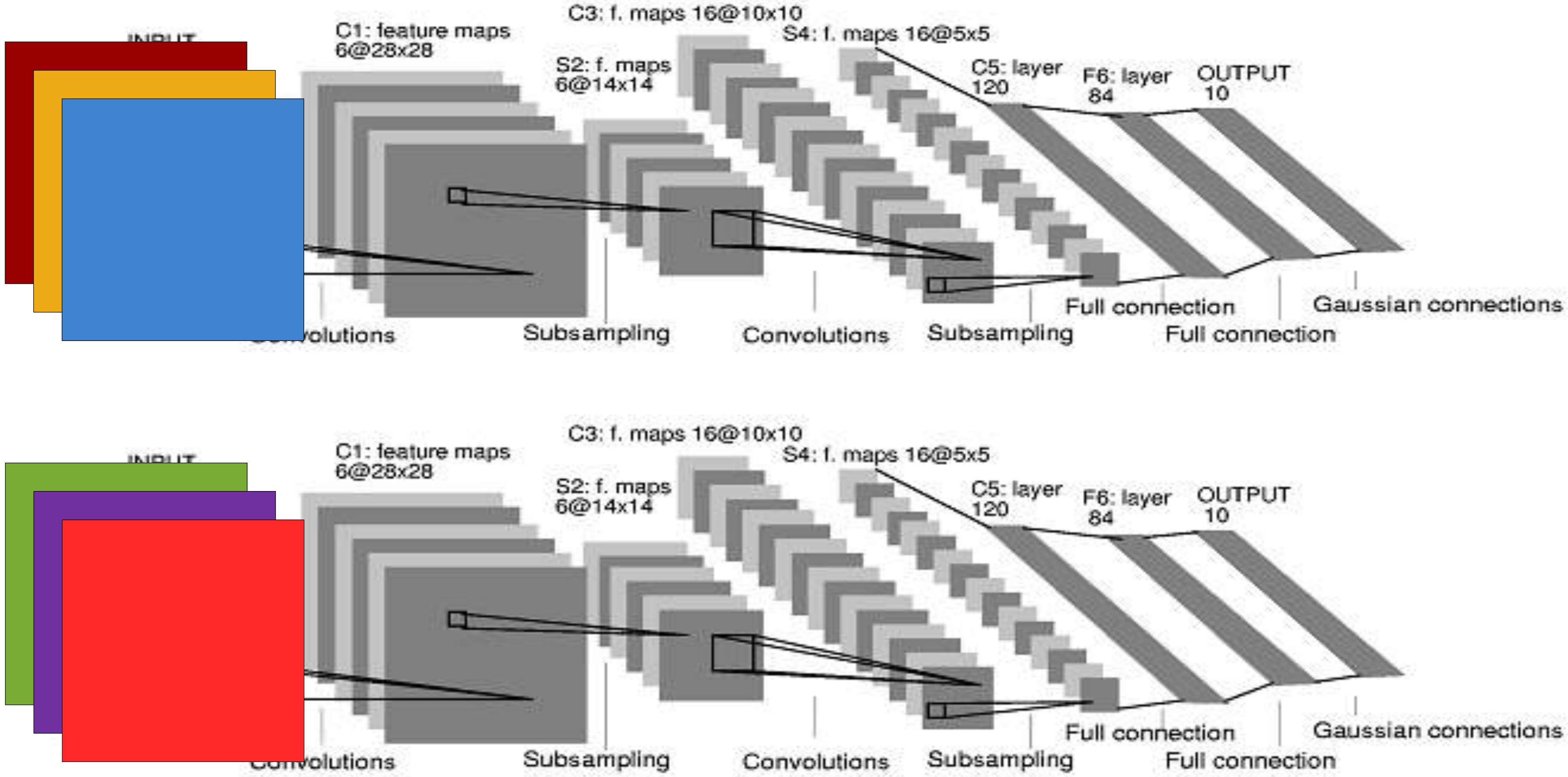
Moore's law made CPUs 300x faster than in 1990
But its over…



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

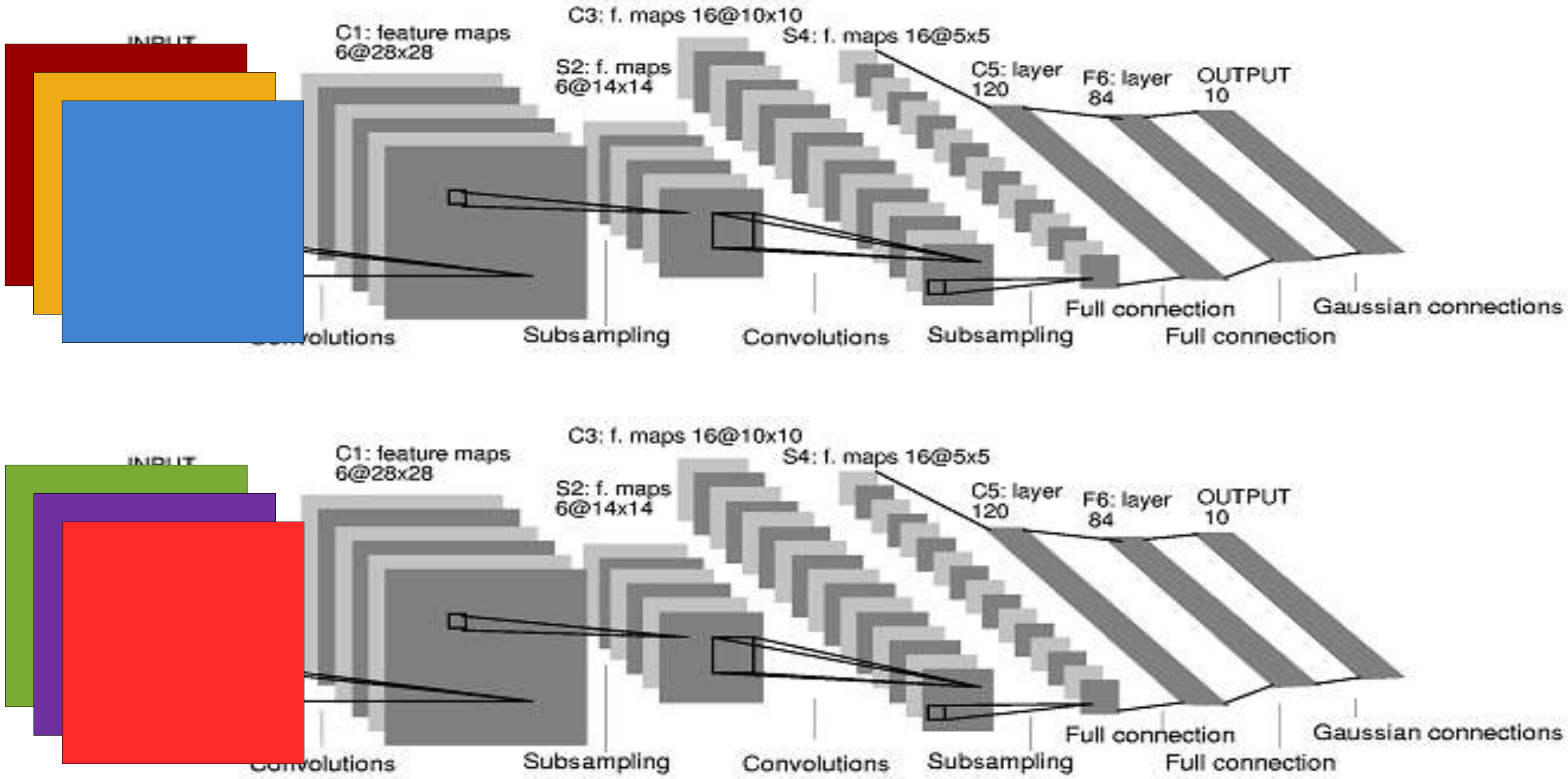
C Moore, Data Processing in ExaScale-ClassComputer Systems, Salishan, April 2011

Data Parallel – Run multiple inputs in parallel



Dally, High Performance Hardware for Machine Learning, NIPS'2015

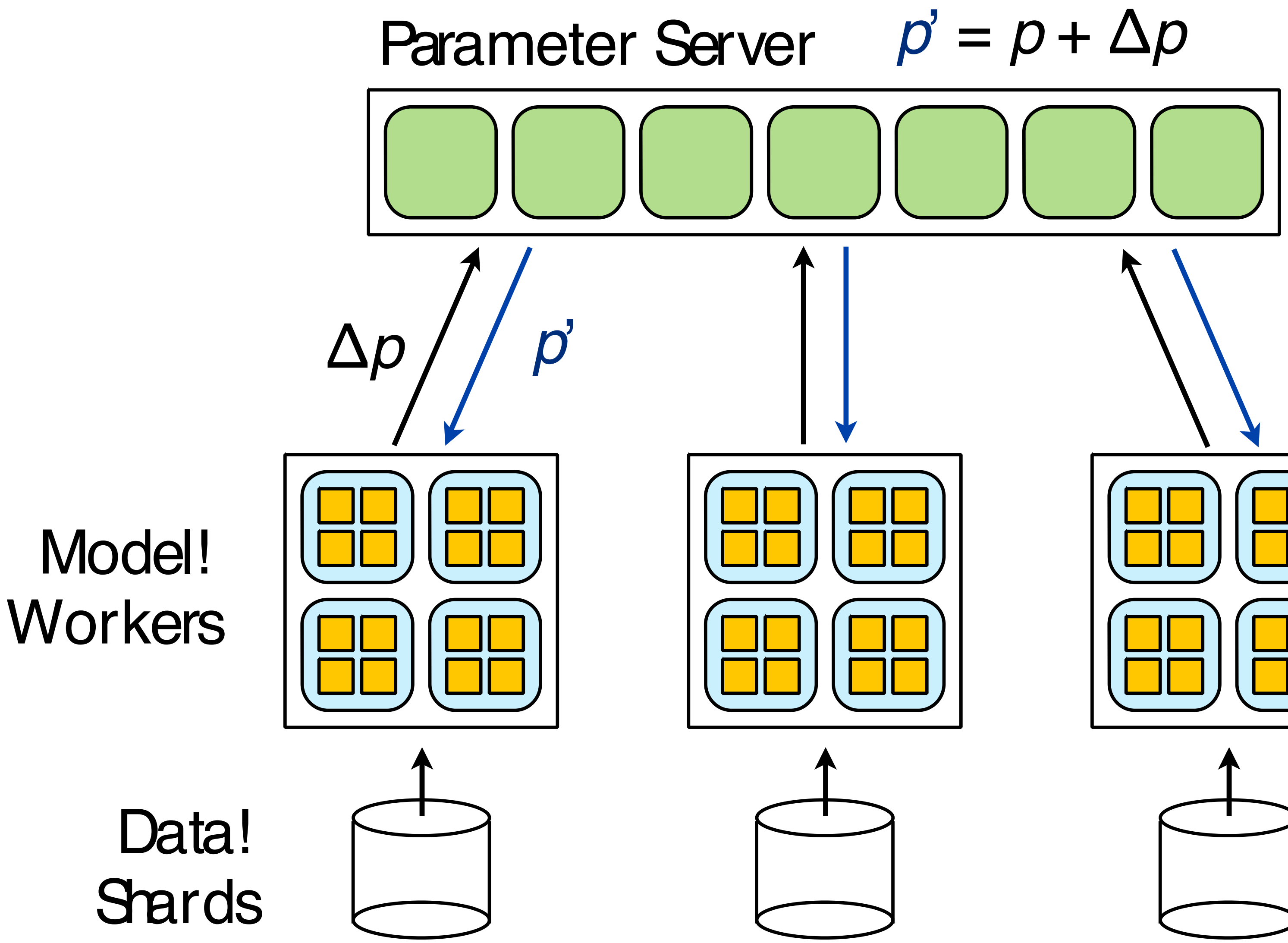
Data Parallel – Run multiple inputs in parallel



- Doesn’t affect latency for one input
- Requires P-fold larger batch size
- For training requires coordinated weight update

Dally, High Performance Hardware for Machine Learning, NIPS’2015

Parameter Update



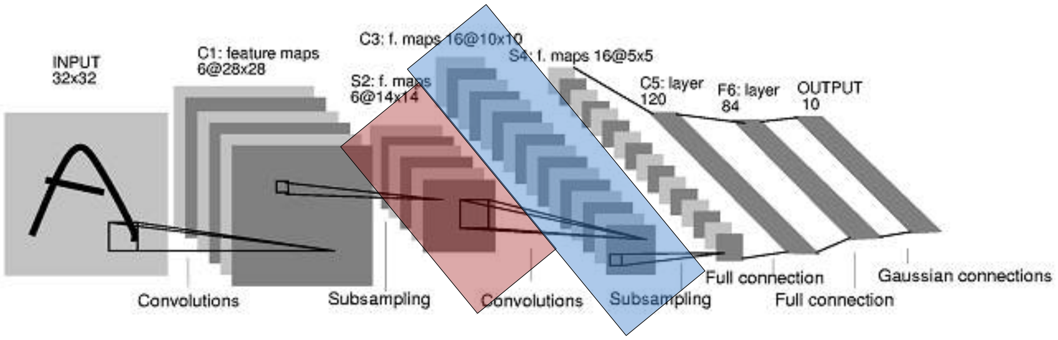
Large Scale Distributed Deep Networks, Jeff Dean et al., 2013

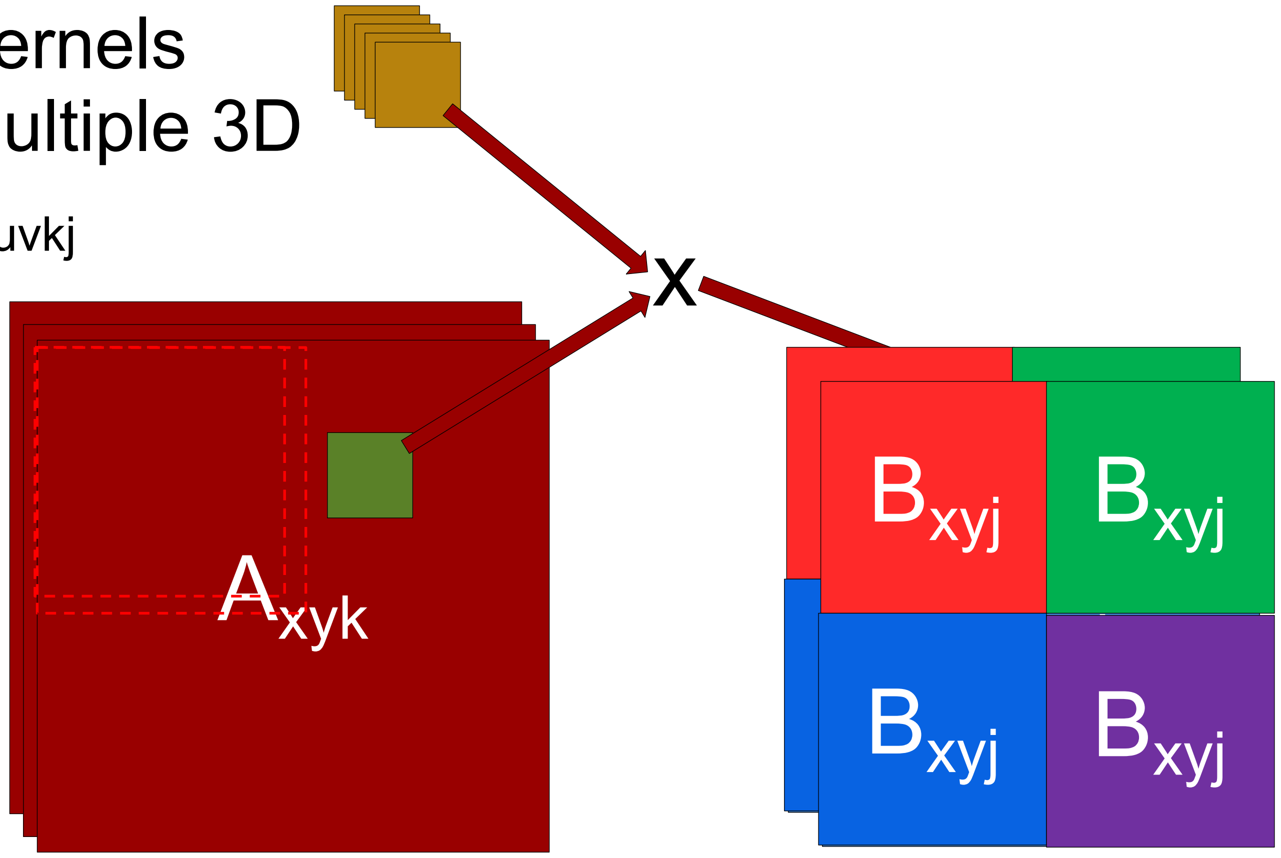
Model Parallel

Split up the Model – i.e. the network

Dally, High Performance Hardware for Machine Learning, NIPS’2015

Model-Parallel Convolution – by output region (x,y)





Kernels
Multiple 3D
K_{uvkj}

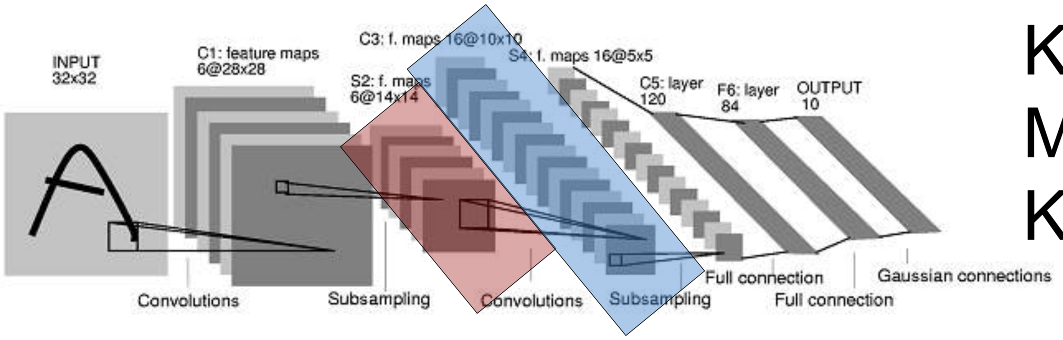
Input maps
A_{xyk}

Output maps
B_{xyj}

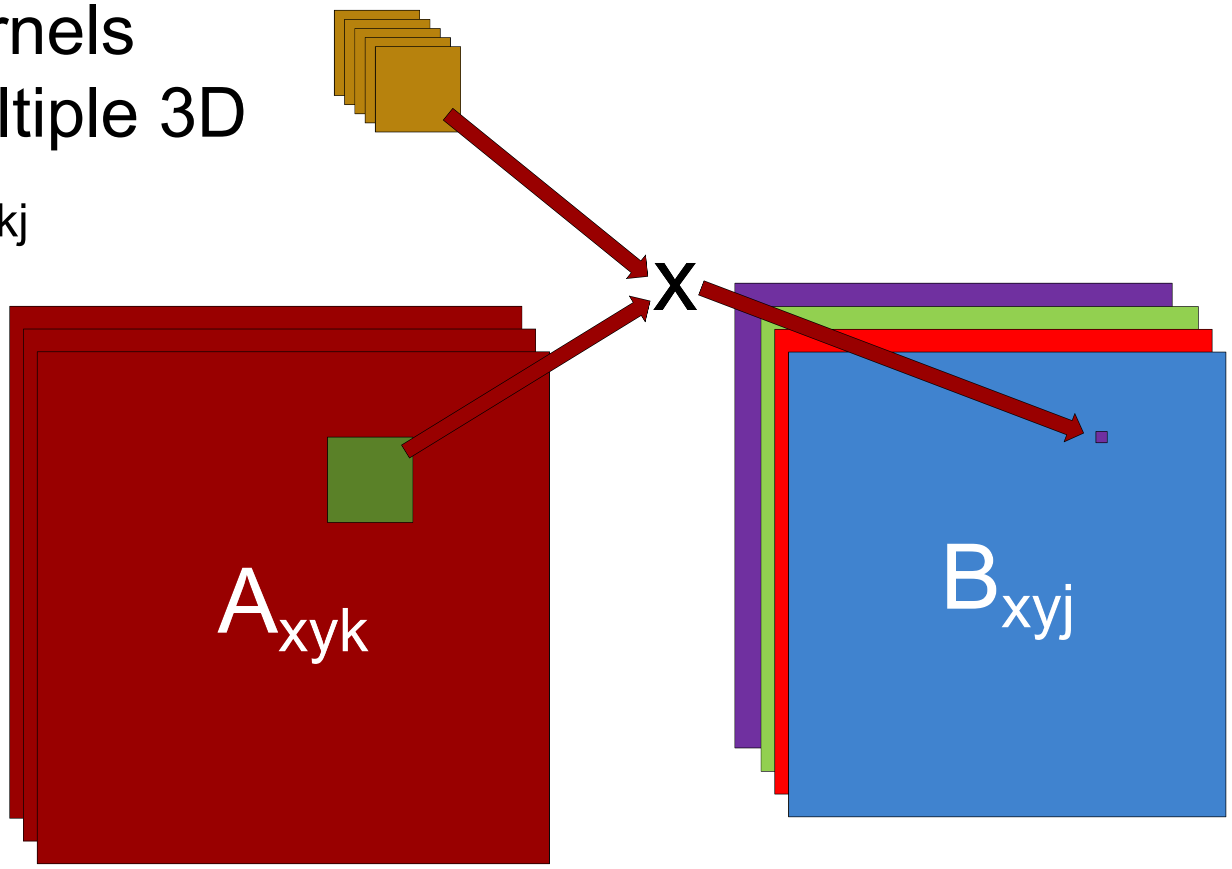
```
6D Loop
Forall output map j
    For each input map k
        For each pixel x,y
            For each kernel element u,v
                $B_{xyj} \mathrel{+}= A_{(x-u)(y-v)k} \times K_{uvkj}$
```

Dally, High Performance Hardware for Machine Learning, NIPS’2015

Model-Parallel Convolution – By output map j (filter)



Kernels
Multiple 3D
K_{uvkj}



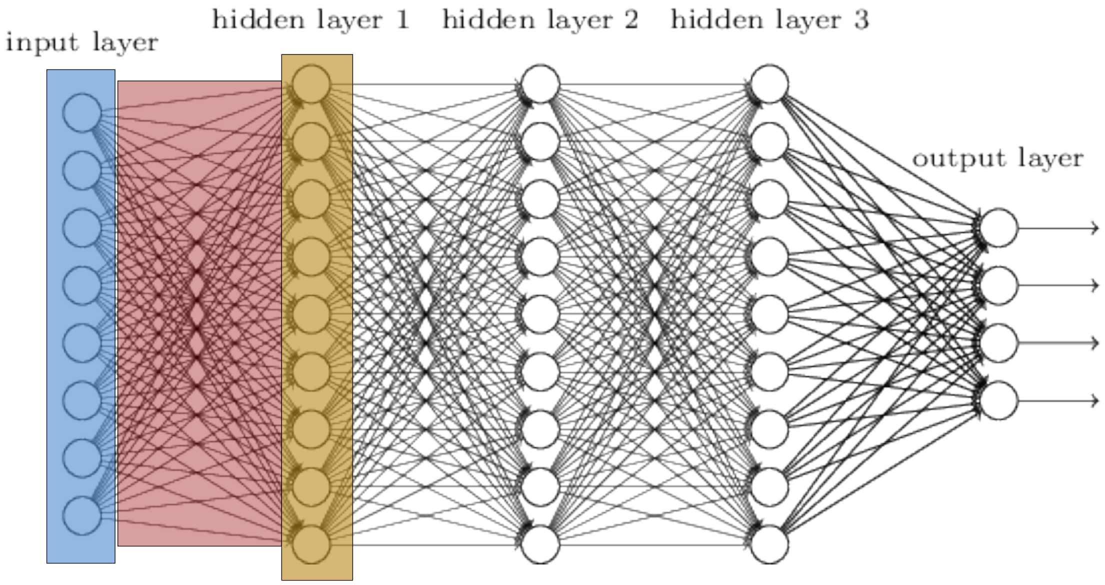
Input maps
A_{xyk}

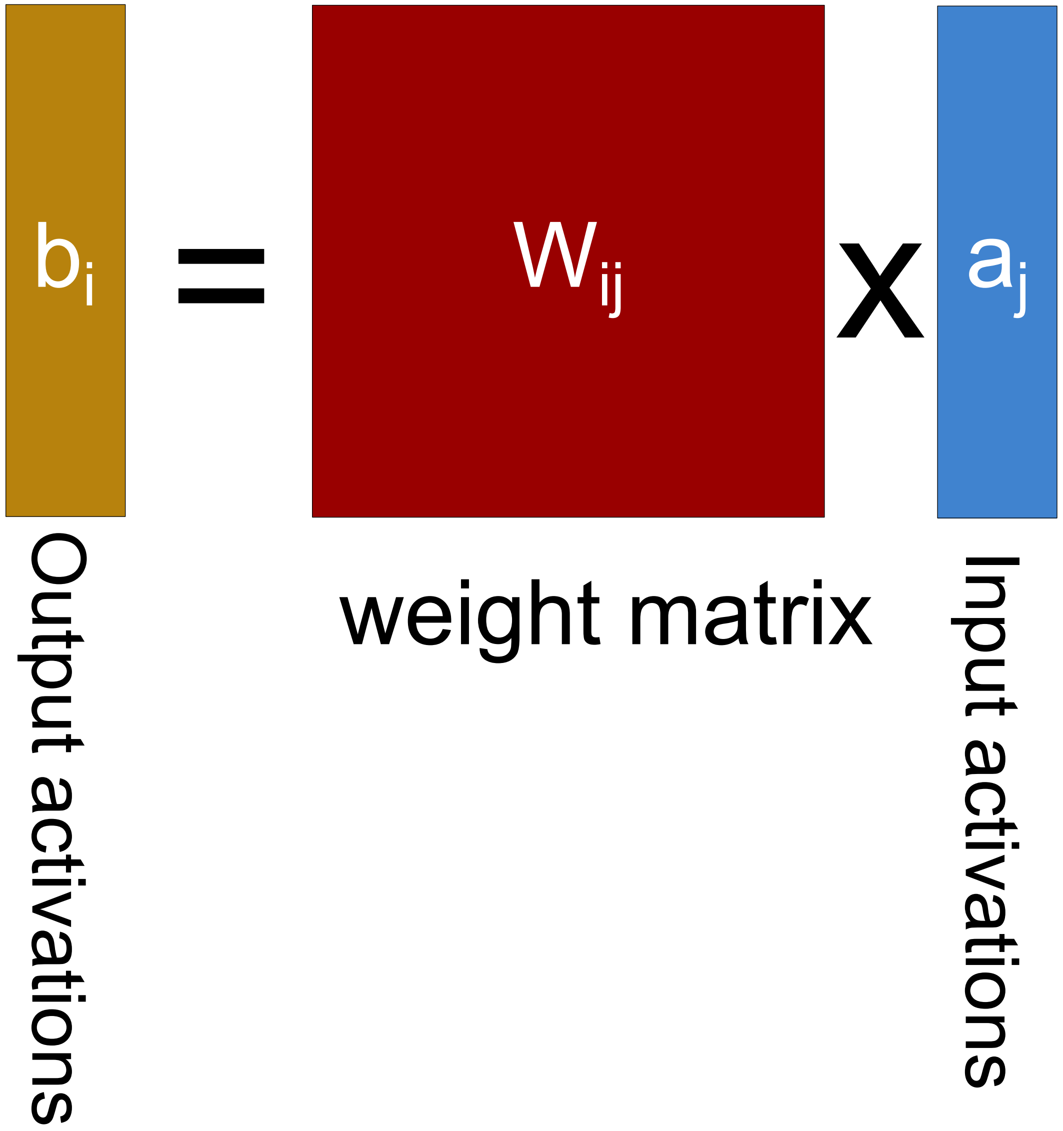
Output maps
B_{xyj}

```
6D Loop
Forall output map j
    For each input map k
        For each pixel x,y
            For each kernel element u,v
                $B_{xyj} \mathrel{+}= A_{(x-u)(y-v)k} \times K_{uvkj}$
```

Dally, High Performance Hardware for Machine Learning, NIPS’2015

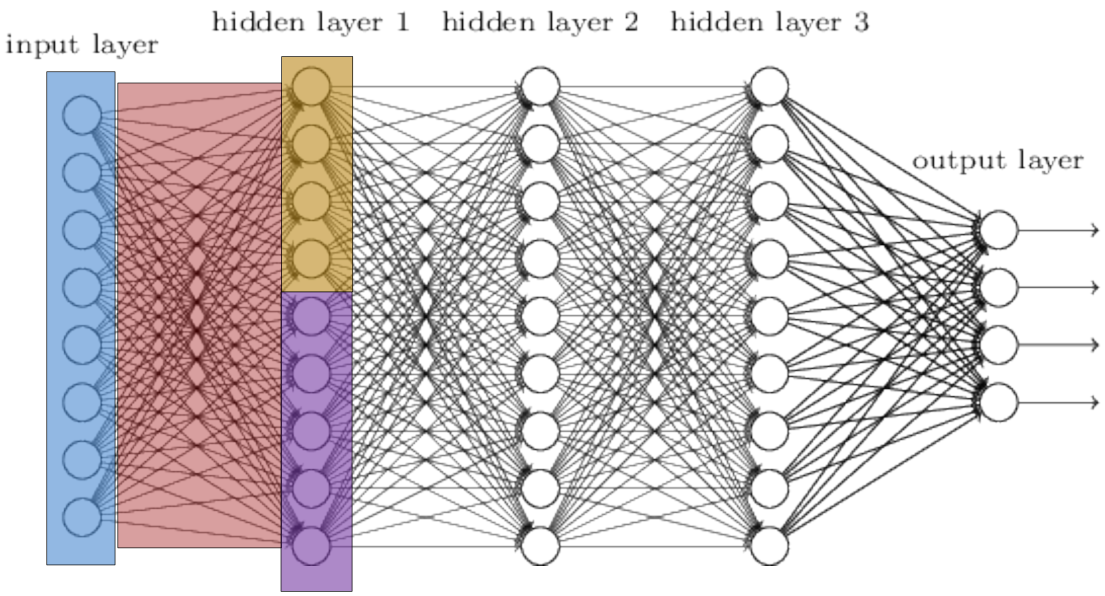
Model Parallel Fully-Connected Layer (M x V)

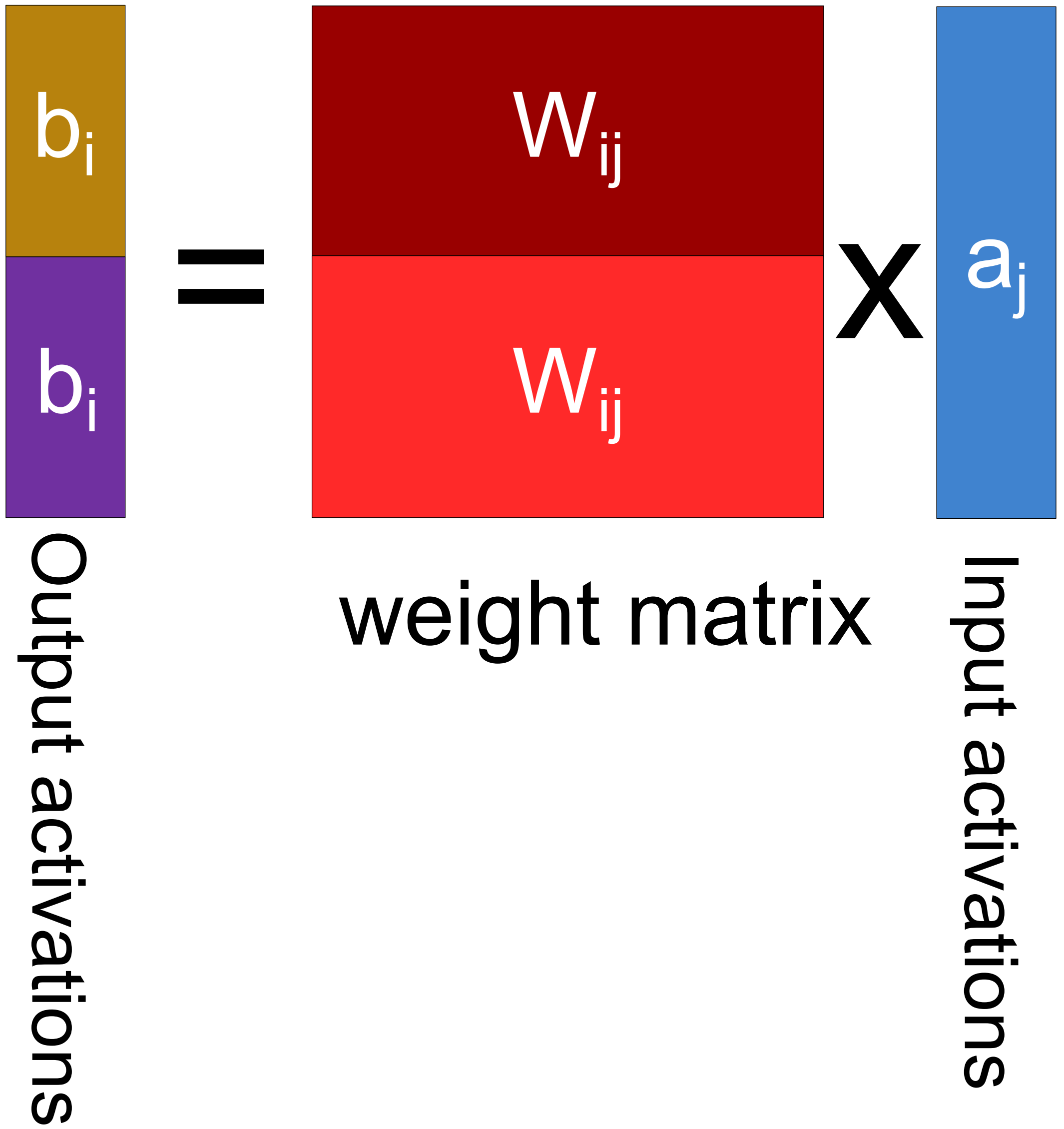




Dally, High Performance Hardware for Machine Learning, NIPS’2015

Model Parallel Fully-Connected Layer (M x V)





b_i

b_i

=

W_{ij}

W_{ij}

x

a_j

Output activations

weight matrix

Input activations

Dally, High Performance Hardware for Machine Learning, NIPS'2015

Hyper-Parameter Parallel

Try many alternative networks in parallel

Summary of Parallelism

- Lots of parallelism in DNNs
  - 16M independent multiplies in one FC layer
  - Limited by overhead to exploit a fraction of this

- Data parallel
  - Run multiple training examples in parallel
  - Limited by batch size

- Model parallel
  - Split model over multiple processors
  - By layer
  - Conv layers by map region
  - Fully connected layers by output activation

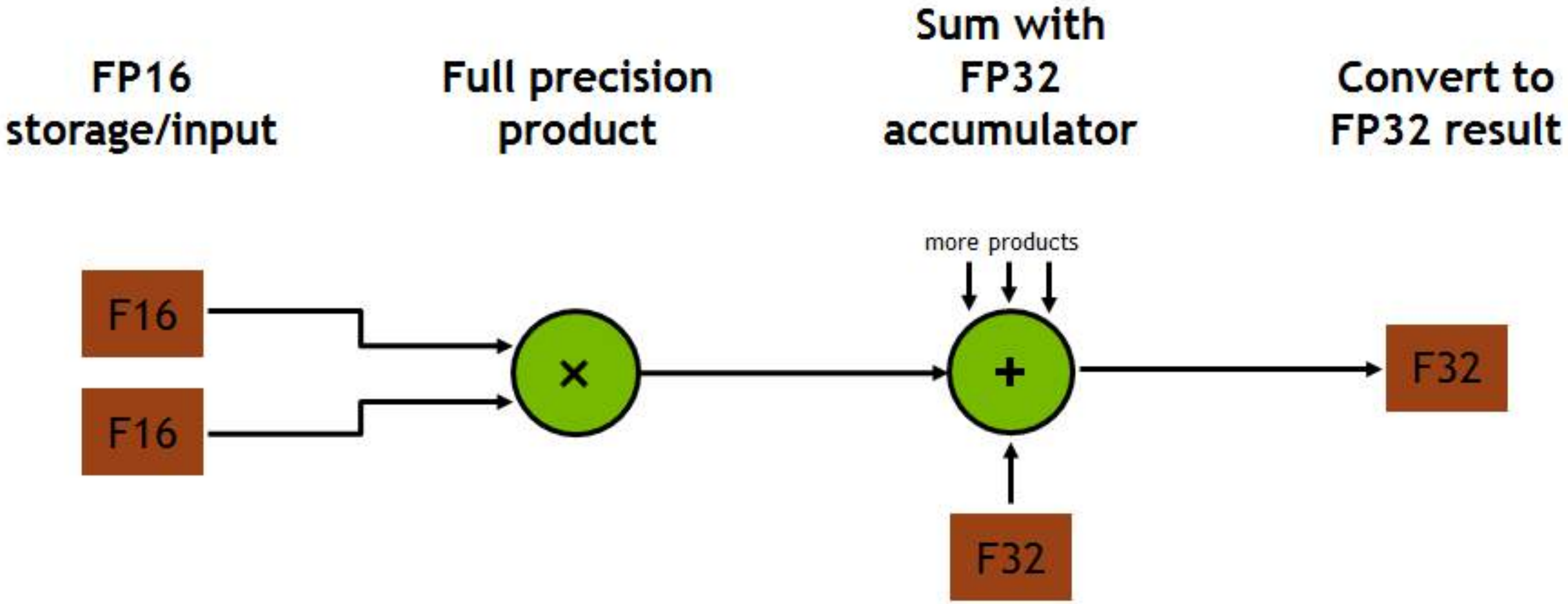
- Easy to get 16-64 GPUs training one model in parallel

Dally, High Performance Hardware for Machine Learning, NIPS'2015

Part 3: Efficient Training — Algorithms

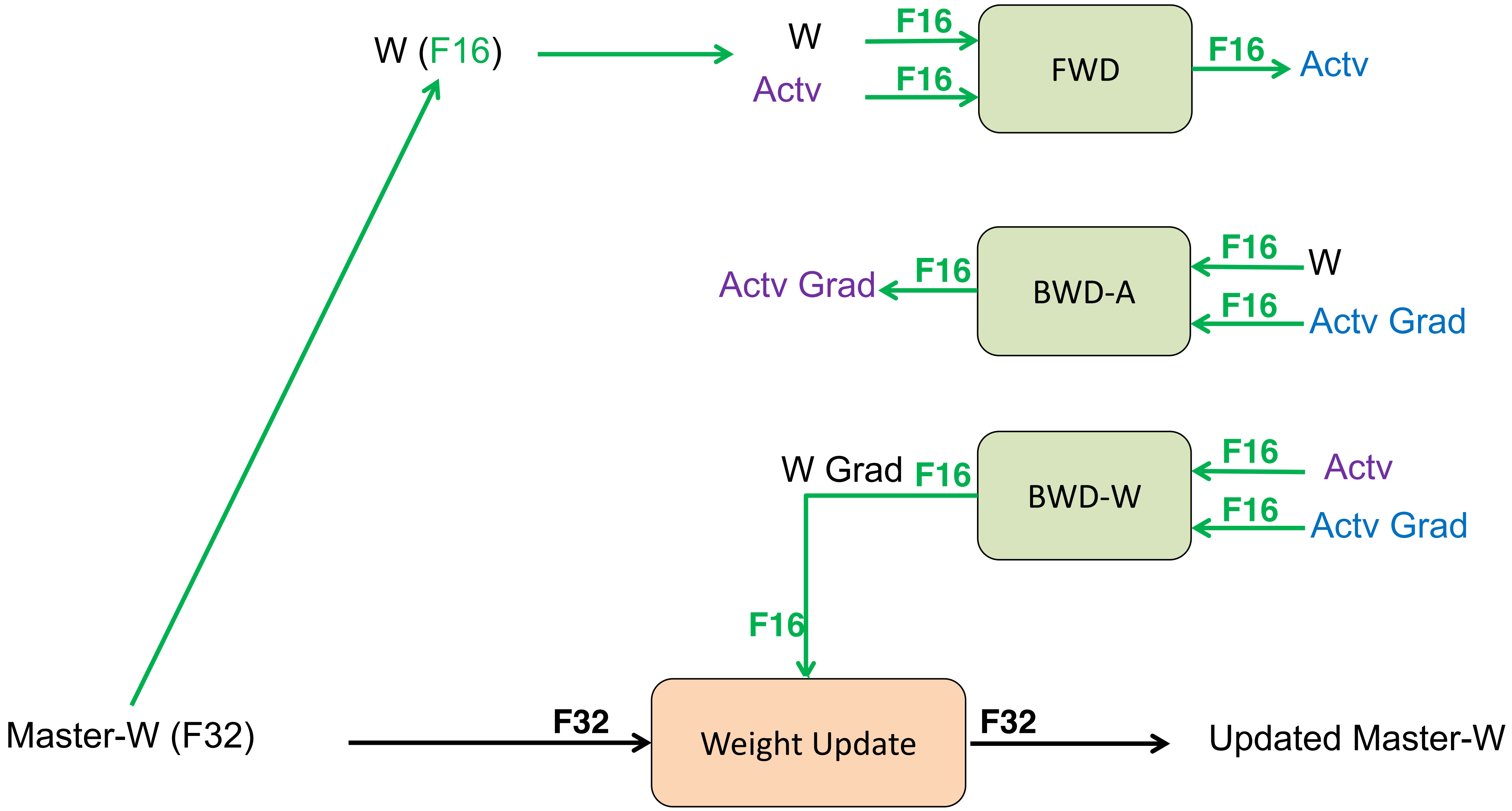
- 1. Parallelization
- 2. Mixed Precision with FP16 and FP32
- 3. Model Distillation
- 4. DSD: Dense-Sparse-Dense Training

Mixed Precision



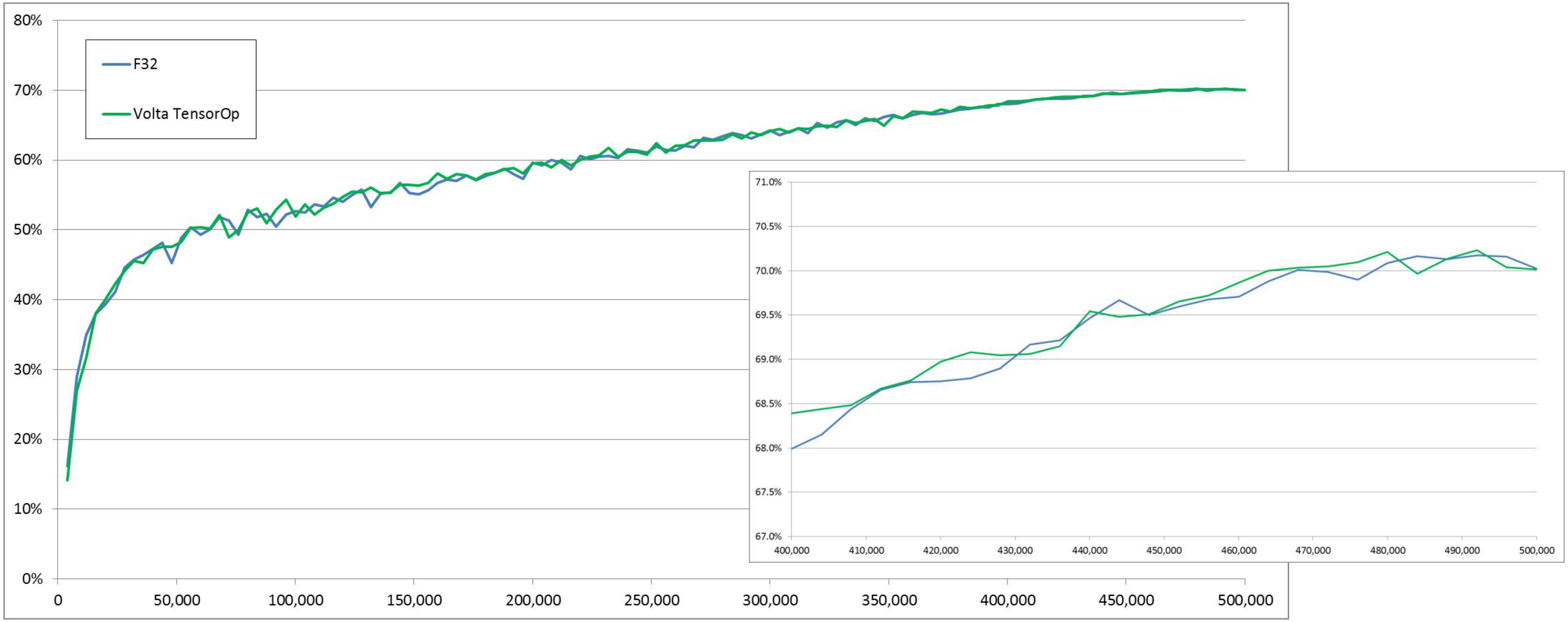
https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/

Mixed Precision Training



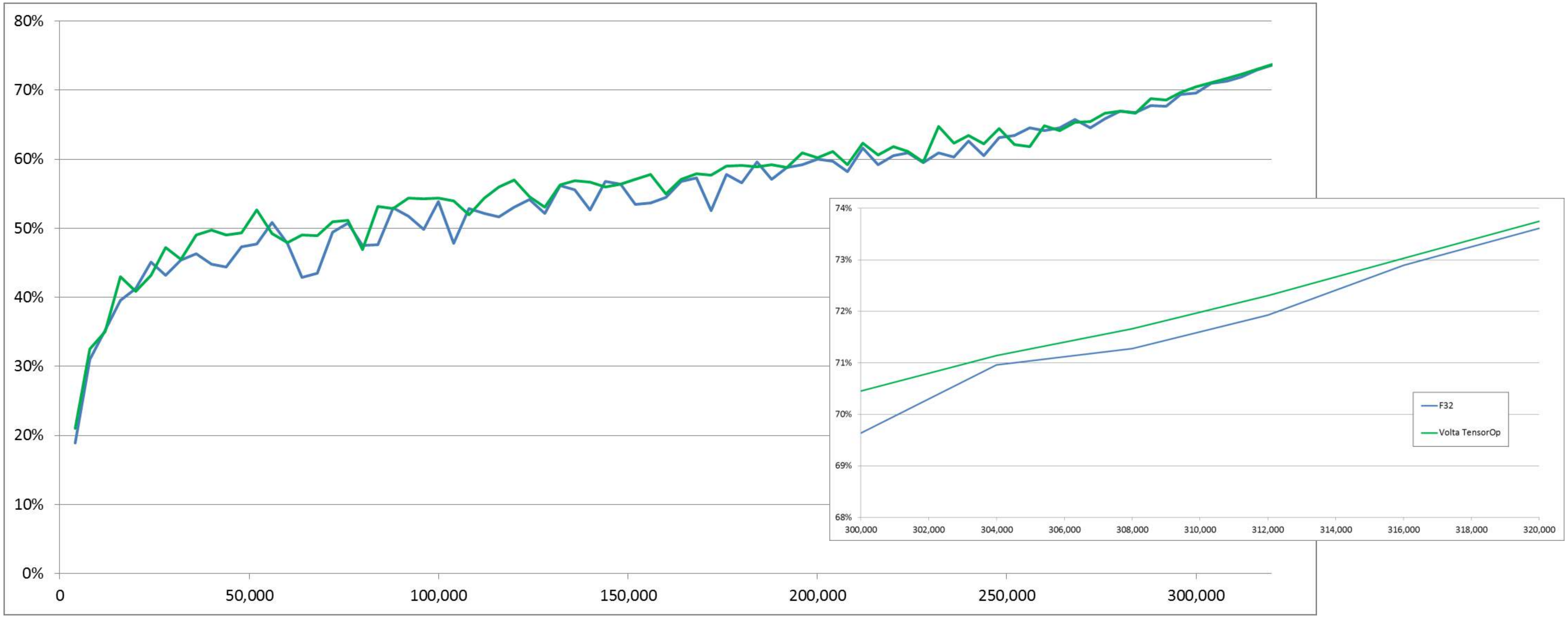
Boris Ginsburg, Sergei Nikolaev, Paulius Micikevicius, “Training with mixed precision”, NVIDIA GTC 2017

Inception V1



Boris Ginsburg, Sergei Nikolaev, Paulius Micikevicius, “Training with mixed precision”, NVIDIA GTC 2017

ResNet



Boris Ginsburg, Sergei Nikolaev, Paulius Micikevicius, “Training with mixed precision”, NVIDIA GTC 2017

AlexNet

Mode	Top1 accuracy, %	Top5 accuracy, %
Fp32	58.62	81.25
Mixed precision training	58.12	80.71

Inception V3

Mode	Top1 accuracy, %	Top5 accuracy, %
Fp32	71.75	90.52
Mixed precision training	71.17	90.10

ResNet-50

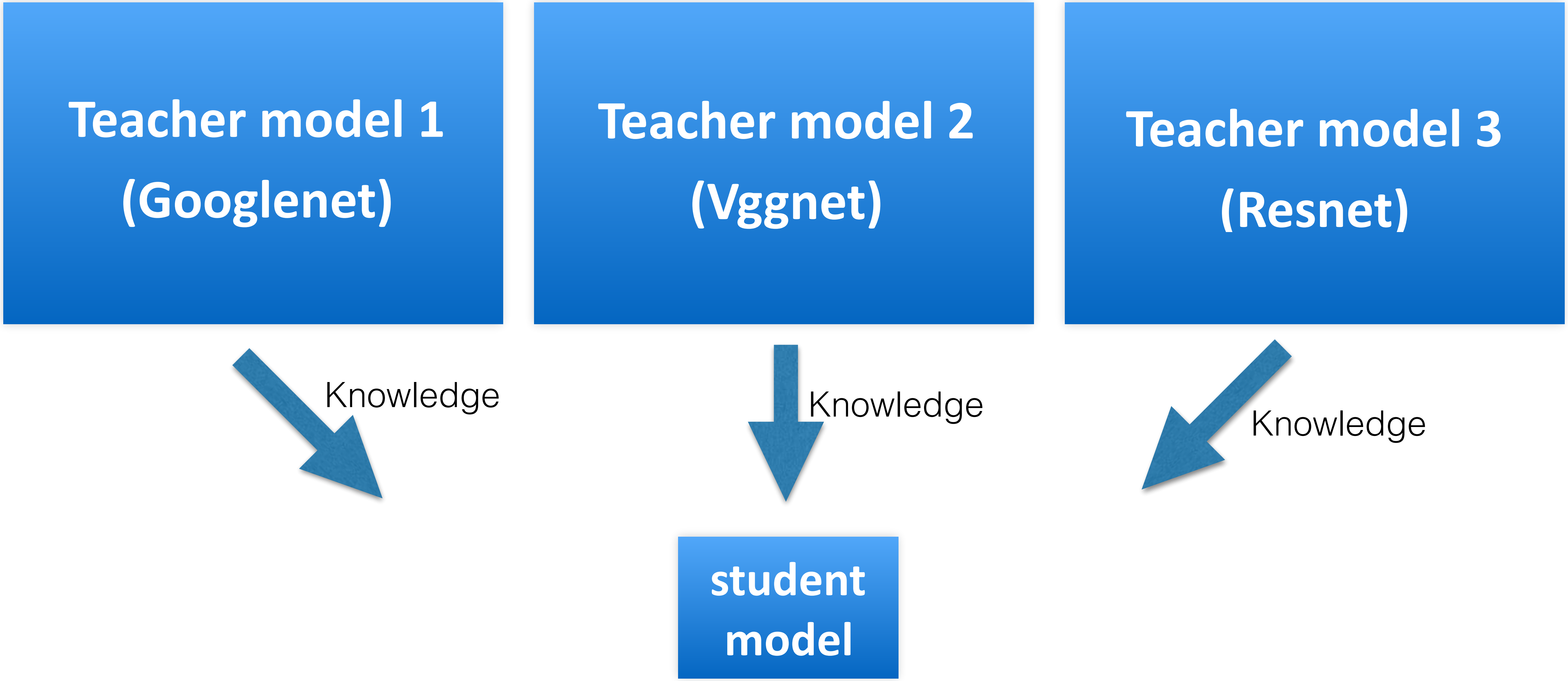
Mode	Top1 accuracy, %	Top5 accuracy, %
Fp32	73.85	91.44
Mixed precision training	73.6	91.11

Boris Ginsburg, Sergei Nikolaev, Paulius Micikevicius, “Training with mixed precision”, NVIDIA GTC 2017

Part 3: Efficient Training Algorithm

- 1. Parallelization
- 2. Mixed Precision with FP16 and FP32
- 3. Model Distillation
- 4. DSD: Dense-Sparse-Dense Training

Model Distillation



student model has much smaller model size

Softened outputs reveal the dark knowledge

cow	dog	cat	car	
0	1	0	0	original hard targets
10^{-6}	.9	.1	10^{-9}	output of geometric ensemble
.05	.3	.2	.005	softened output of ensemble

Hinton et al. Dark knowledge / Distilling the Knowledge in a Neural Network

Softened outputs reveal the dark knowledge

$$p_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

- Method: Divide score by a "temperature" to get a much softer distribution

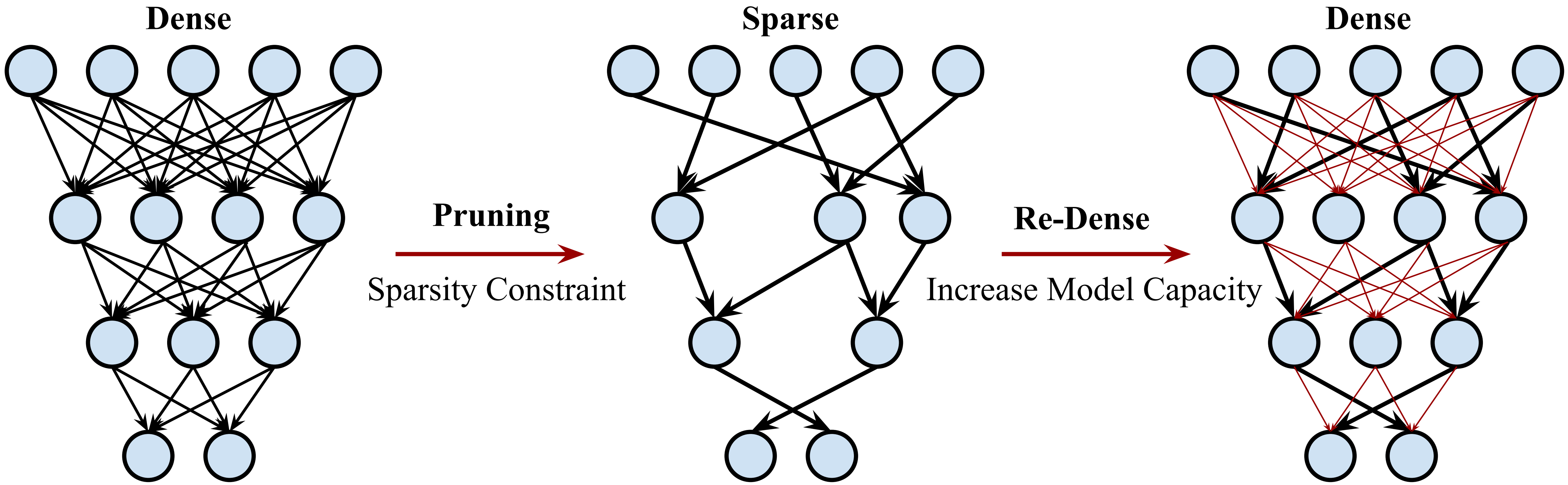
- Result: Start with a trained model that classifies 58.9% of the test frames correctly. The new model converges to 57.0% correct even when it is only trained on 3% of the data

Hinton et al. Dark knowledge / Distilling the Knowledge in a Neural Network

Part 3: Efficient Training Algorithm

- 1. Parallelization
- 2. Mixed Precision with FP16 and FP32
- 3. Model Distillation
- 4. DSD: Dense-Sparse-Dense Training

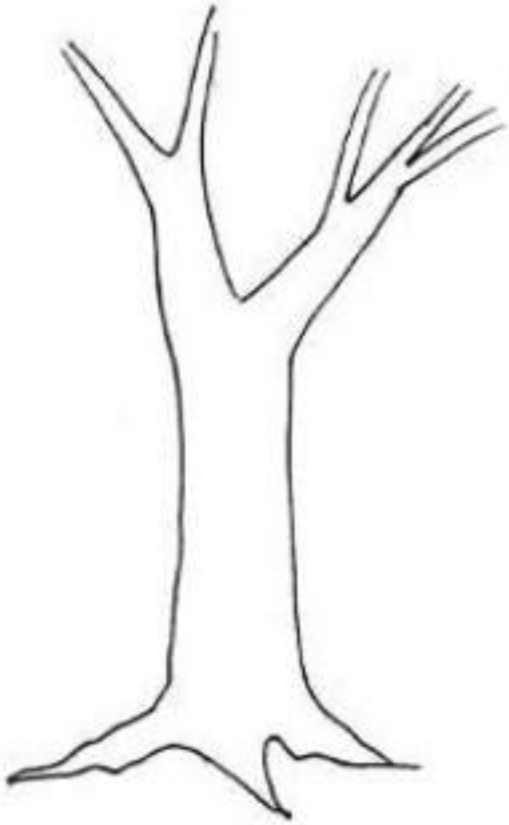
DSD: Dense Sparse Dense Training



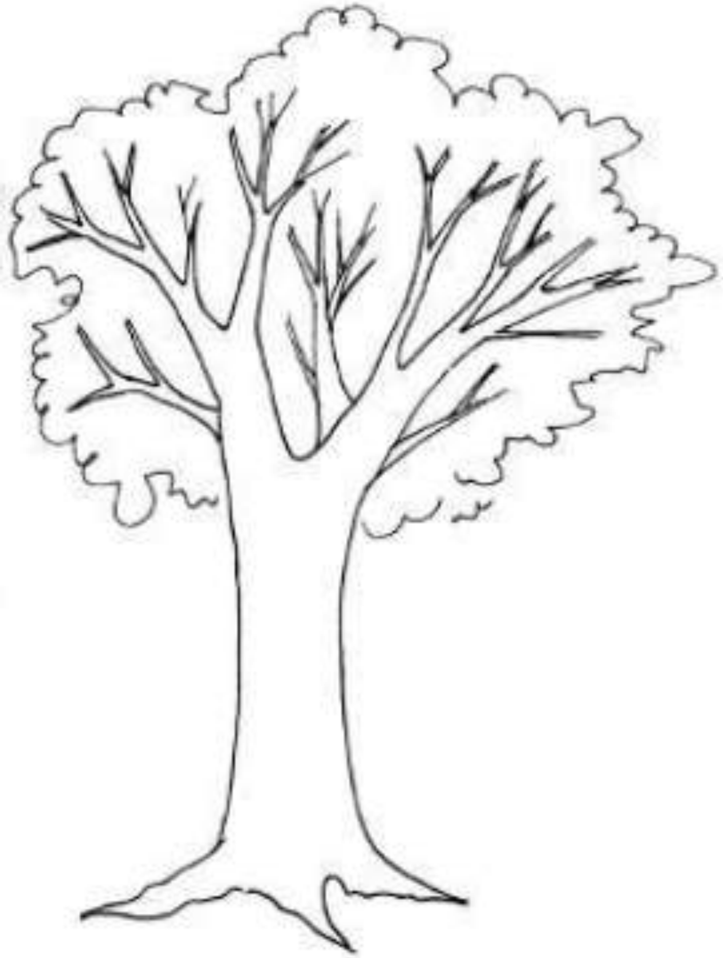
DSD produces same model architecture but can find better optimization solution, arrives at better local minima, and achieves higher prediction accuracy across a wide range of deep neural networks on CNNs / RNNs / LSTMs.

Han et al. "DSD: Dense-Sparse-Dense Training for Deep Neural Networks", ICLR 2017

DSD: Intuition



learn the trunk first



then learn the leaves

Han et al. “DSD: Dense-Sparse-Dense Training for Deep Neural Networks”, ICLR 2017

[Han et al. ICLR 2017]

DSD is General Purpose: Vision, Speech, Natural Language

Network	Domain	Dataset	Type	Baseline	DSD	Abs. Imp.	Rel. Imp.
GoogleNet	Vision	ImageNet	CNN	31.1% →	**30.0%**	1.1%	3.6%
VGG-16	Vision	ImageNet	CNN	31.5% →	**27.2%**	4.3%	13.7%
ResNet-18	Vision	ImageNet	CNN	30.4% →	**29.3%**	1.1%	3.7%
ResNet-50	Vision	ImageNet	CNN	24.0% →	**23.2%**	0.9%	3.5%

Open Sourced DSD Model Zoo: https://songhan.github.io/DSD

The beseline results of AlexNet, VGG16, GoogleNet, SqueezeNet are from Caffe Model Zoo. ResNet18, ResNet50 are from fb.resnet.torch.

[Han et al. ICLR 2017]

DSD is General Purpose: Vision, Speech, Natural Language

Network	Domain	Dataset	Type	Baseline		DSD	Abs. Imp.	Rel. Imp.
GoogleNet	Vision	ImageNet	CNN	31.1%	→	**30.0%**	1.1%	3.6%
VGG-16	Vision	ImageNet	CNN	31.5%	→	**27.2%**	4.3%	13.7%
ResNet-18	Vision	ImageNet	CNN	30.4%	→	**29.3%**	1.1%	3.7%
ResNet-50	Vision	ImageNet	CNN	24.0%	→	**23.2%**	0.9%	3.5%
NeuralTalk	Caption	Flickr-8K	LSTM	16.8	→	**18.5**	1.7	10.1%

Open Sourced DSD Model Zoo: https://songhan.github.io/DSD

The beseline results of AlexNet, VGG16, GoogleNet, SqueezeNet are from Caffe Model Zoo. ResNet18, ResNet50 are from fb.resnet.torch.

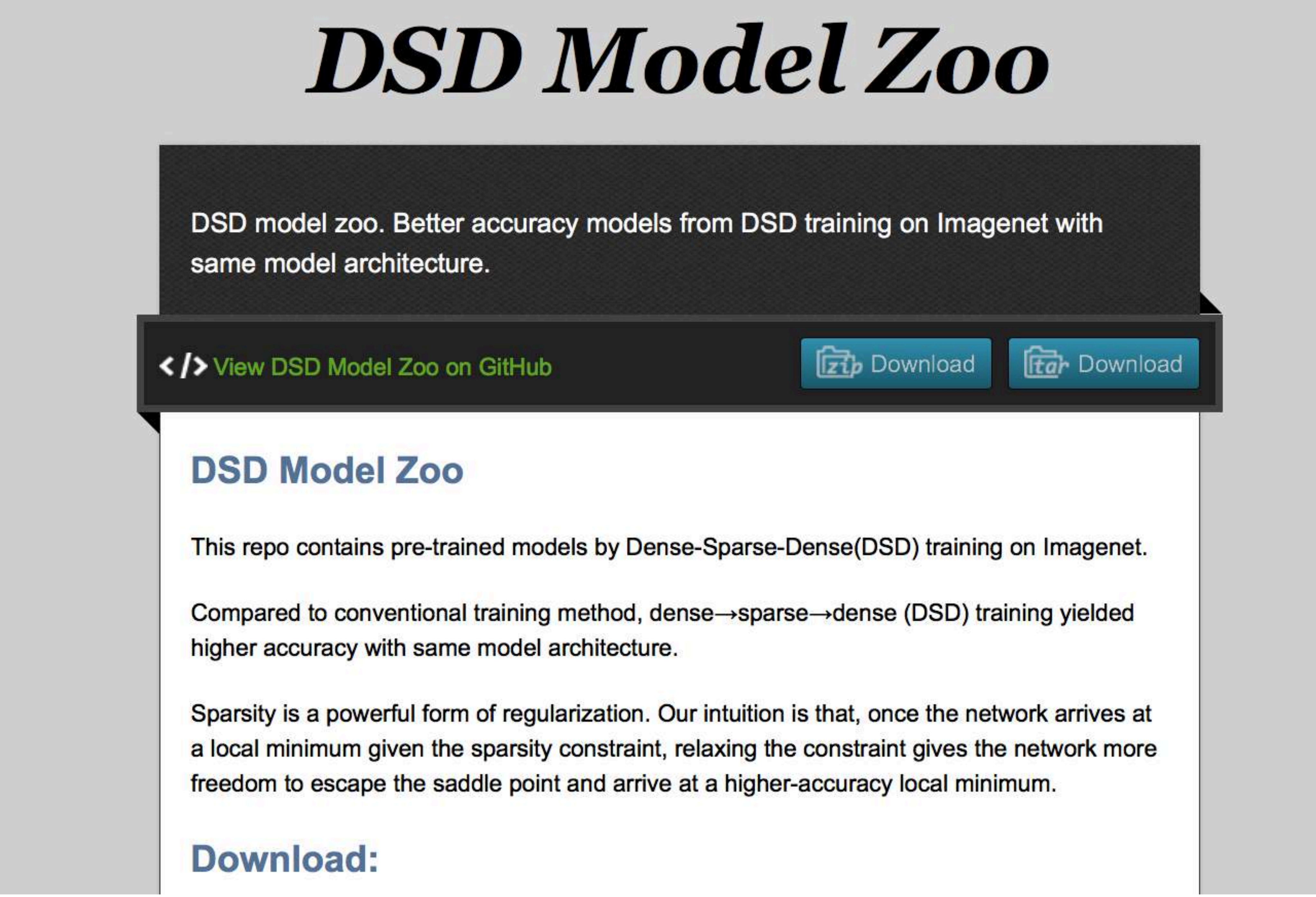
[Han et al. ICLR 2017]

DSD is General Purpose: Vision, Speech, Natural Language

Network	Domain	Dataset	Type	Baseline	DSD	Abs. Imp.	Rel. Imp.
GoogleNet	Vision	ImageNet	CNN	31.1% →	**30.0%**	1.1%	3.6%
VGG-16	Vision	ImageNet	CNN	31.5% →	**27.2%**	4.3%	13.7%
ResNet-18	Vision	ImageNet	CNN	30.4% →	**29.3%**	1.1%	3.7%
ResNet-50	Vision	ImageNet	CNN	24.0% →	**23.2%**	0.9%	3.5%
NeuralTalk	Caption	Flickr-8K	LSTM	16.8 →	**18.5**	1.7	10.1%
DeepSpeech	Speech	WSJ'93	RNN	33.6% →	**31.6%**	2.0%	5.8%
DeepSpeech-2	Speech	WSJ'93	RNN	14.5% →	**13.4%**	1.1%	7.4%

Open Sourced DSD Model Zoo: https://songhan.github.io/DSD

The beseline results of AlexNet, VGG16, GoogleNet, SqueezeNet are from Caffe Model Zoo. ResNet18, ResNet50 are from fb.resnet.torch.



DSD Model Zoo

DSD model zoo. Better accuracy models from DSD training on Imagenet with same model architecture.

DSD Model Zoo

This repo contains pre-trained models by Dense-Sparse-Dense(DSD) training on Imagenet.

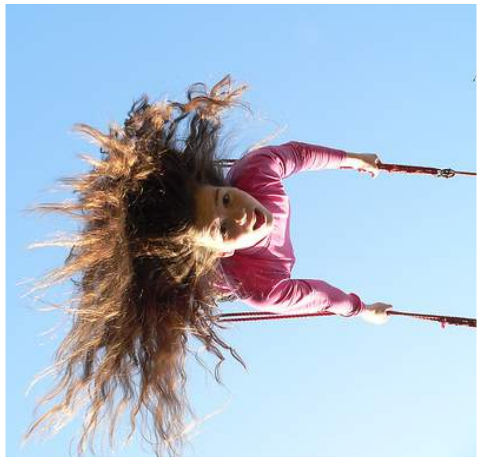
Compared to conventional training method, dense→sparse→dense (DSD) training yielded higher accuracy with same model architecture.

Sparsity is a powerful form of regularization. Our intuition is that, once the network arrives at a local minimum given the sparsity constraint, relaxing the constraint gives the network more freedom to escape the saddle point and arrive at a higher-accuracy local minimum.

Download:

https://songhan.github.io/DSD

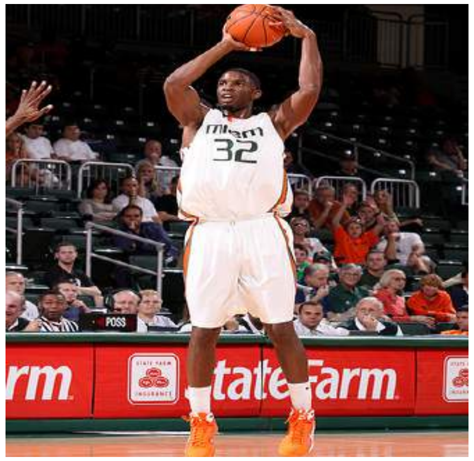
DSD on Caption Generation



✗ **Baseline**: a boy in a red shirt is climbing a rock wall.

✗ **Sparse**: a young girl is jumping off a tree.

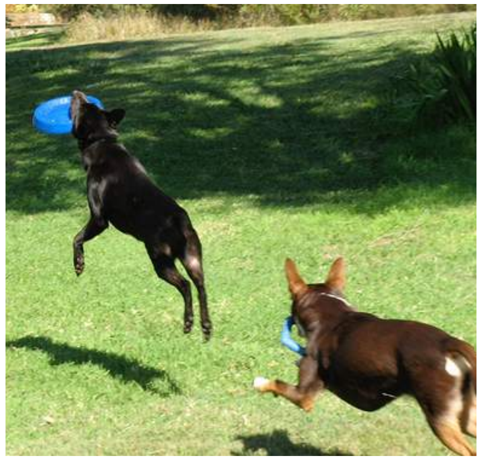
✓ **DSD**: a young girl in a pink shirt is swinging on a swing.



○ **Baseline**: a basketball player in a red uniform is playing with a ball.

○ **Sparse**: a basketball player in a blue uniform is jumping over the goal.

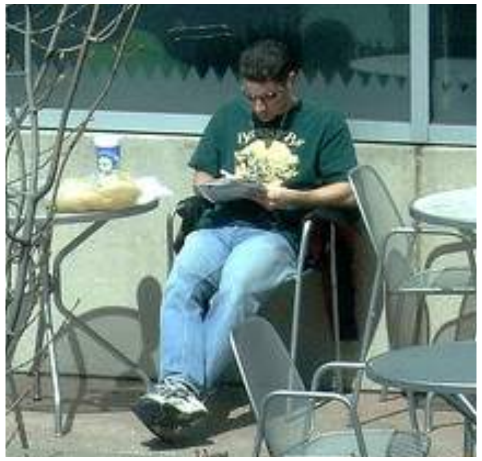
✓ **DSD**: a basketball player in a white uniform is trying to make a shot.



✓ **Baseline**: two dogs are playing together in a field.

✓ **Sparse**: two dogs are playing in a field.

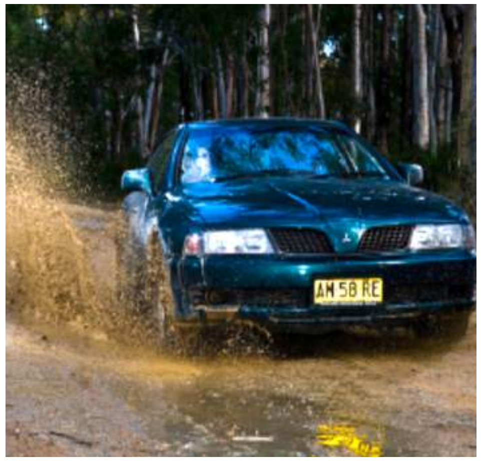
✓ **DSD**: two dogs are playing in the grass.



✗ **Baseline**: a man and a woman are sitting on a bench.

○ **Sparse**: a man is sitting on a bench with his hands in the air.

○ **DSD**: a man is sitting on a bench with his arms folded.



✗ **Baseline**: a person in a red jacket is riding a bike through the woods.

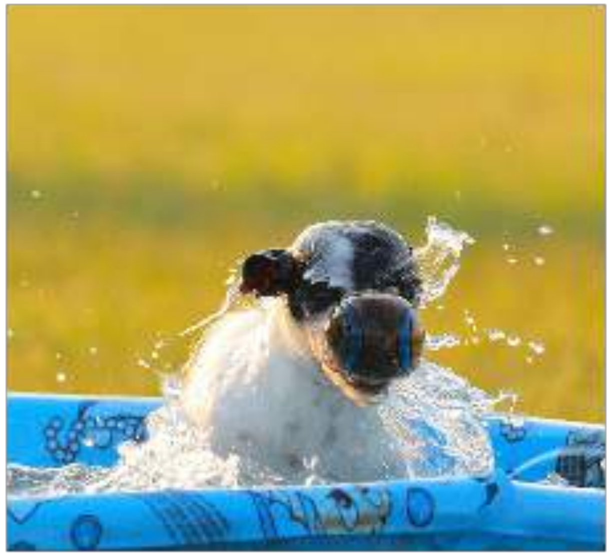
✓ **Sparse**: a car drives through a mud puddle.

✓ **DSD**: a car drives through a forest.

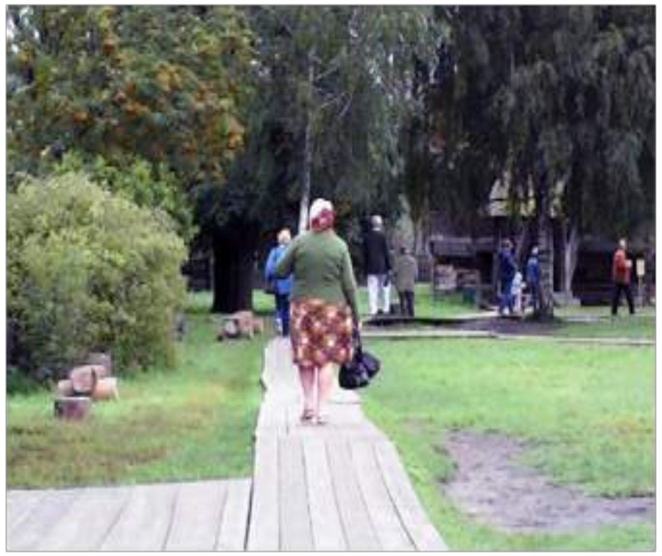
Baseline model: Andrej Karpathy, Neural Talk model zoo.

Han et al. "DSD: Dense-Sparse-Dense Training for Deep Neural Networks", ICLR 2017

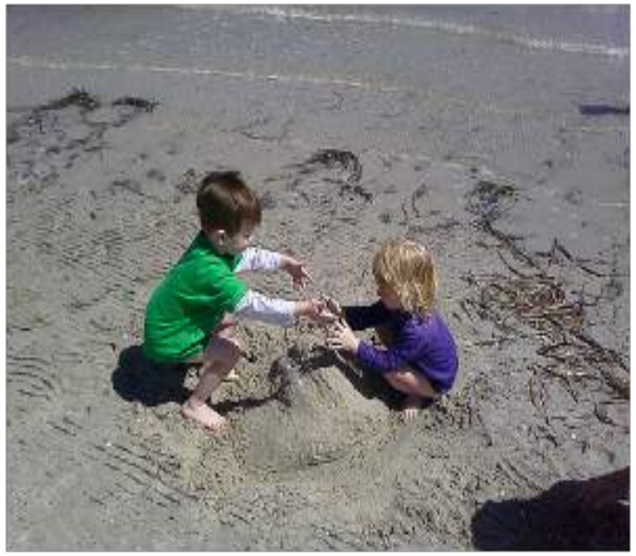
DSD on Caption Generation



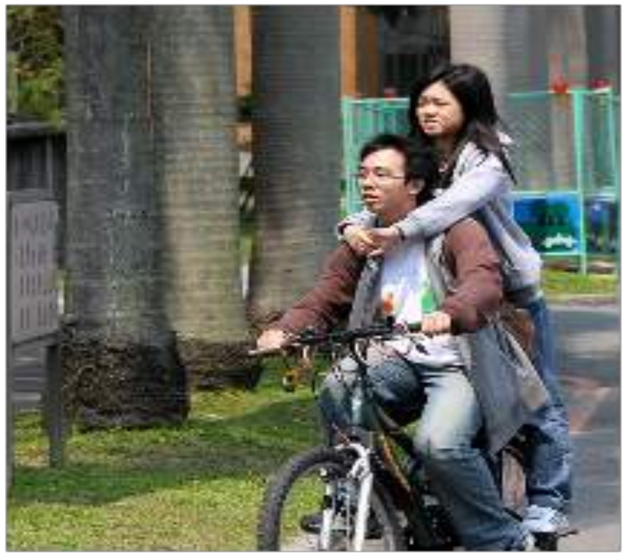
✗ **Baseline**: a boy is swimming in a pool.
○ **Sparse**: a small black dog is jumping into a pool.
✓ **DSD**: a black and white dog is swimming in a pool.



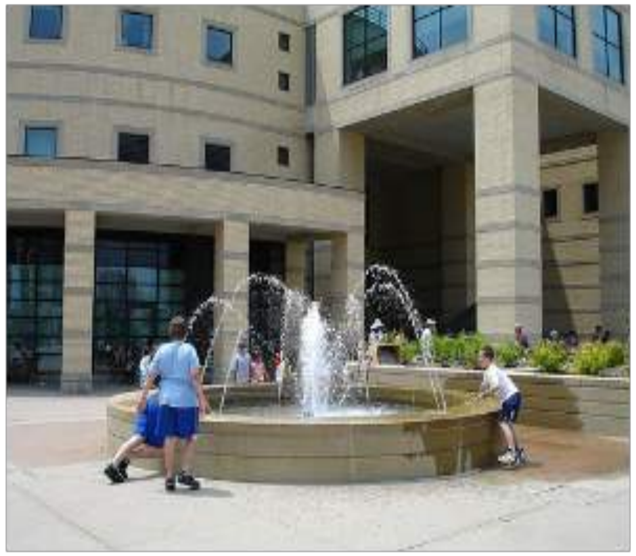
✗ **Baseline**: a group of people are standing in front of a building.
✗ **Sparse**: a group of people are standing in front of a building.
✓ **DSD**: a group of people are walking in a park.



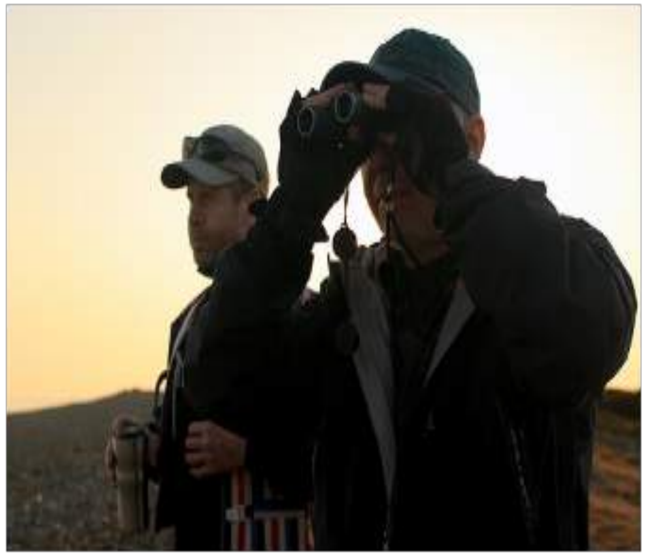
✗ **Baseline**: two girls in bathing suits are playing in the water.
✓ **Sparse**: two children are playing in the sand.
✓ **DSD**: two children are playing in the sand.



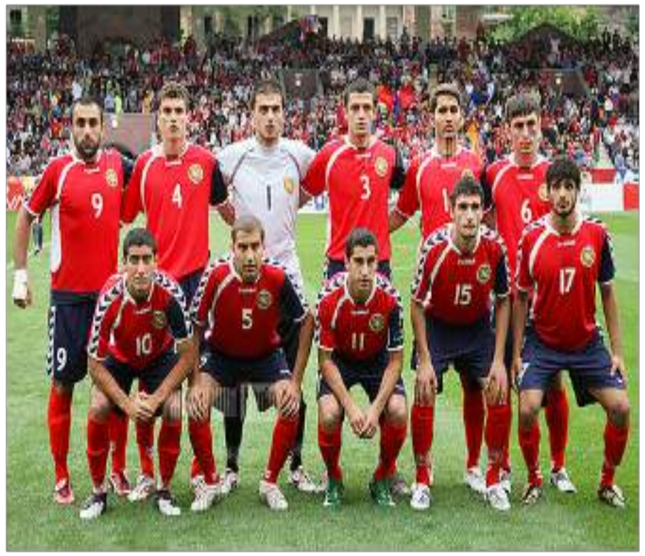
○ **Baseline**: a man in a red shirt and jeans is riding a bicycle down a street.
○ **Sparse**: a man in a red shirt and a woman in a wheelchair.
✓ **DSD**: a man and a woman are riding on a street.



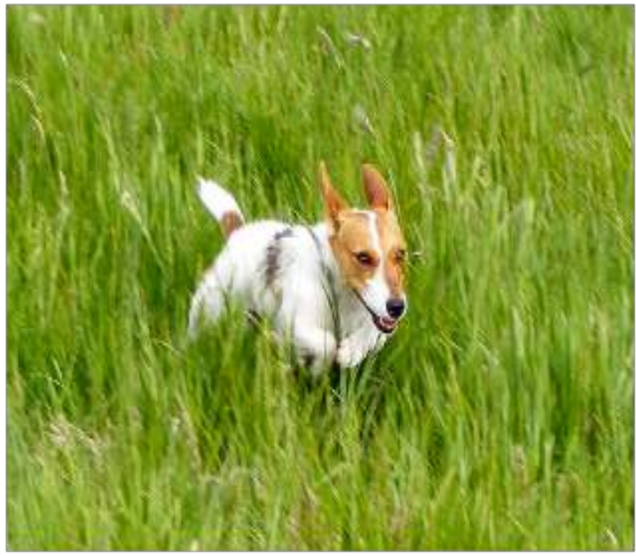
✗ **Baseline**: a group of people sit on a bench in front of a building.
○ **Sparse**: a group of people are standing in front of a building.
✓ **DSD**: a group of people are standing in a fountain.



✗ **Baseline**: a man in a black jacket and a black jacket is smiling.
✗ **Sparse**: a man and a woman are standing in front of a mountain.
✓ **DSD**: a man in a black jacket is standing next to a man in a black shirt.



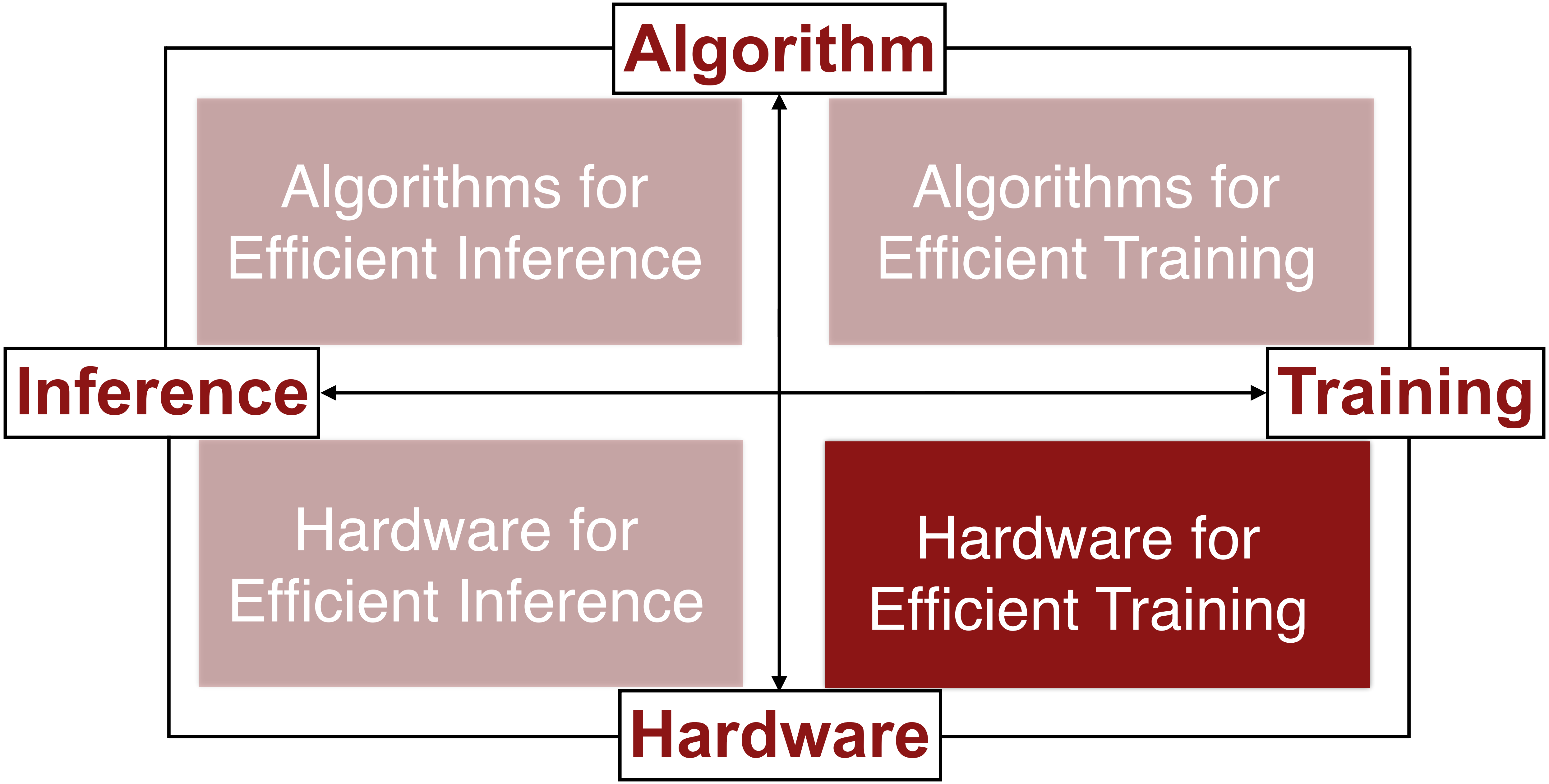
○ **Baseline**: a group of football players in red uniforms.
○ **Sparse**: a group of football players in a field.
✓ **DSD**: a group of football players in red and white uniforms.



○ **Baseline**: a dog runs through the grass.
○ **Sparse**: a dog runs through the grass.
✓ **DSD**: a white and brown dog is running through the grass.

Baseline model: Andrej Karpathy, Neural Talk model zoo.

Agenda



Algorithm

Algorithms for Efficient Inference

Algorithms for Efficient Training

Inference

Training

Hardware for Efficient Inference

Hardware for Efficient Training

Hardware

CPUs for Training

Intel Knights Landing (2016)

- 7 TFLOPS FP32
- 16GB MCDRAM– 400 GB/s
- 245W TDP
- 29 GFLOPS/W (FP32)
- 14nm process

Knights Mill: next gen Xeon Phi "optimized for deep learning"

Intel announced the addition of new vector instructions for deep learning (AVX512-4VNNIW and AVX512-4FMAPS), October 2016

Slide Source: Sze et al Survey of DNN Hardware, MICRO'16 Tutorial.
Image Source: Intel, Data Source: Next Platform

GPUs for Training

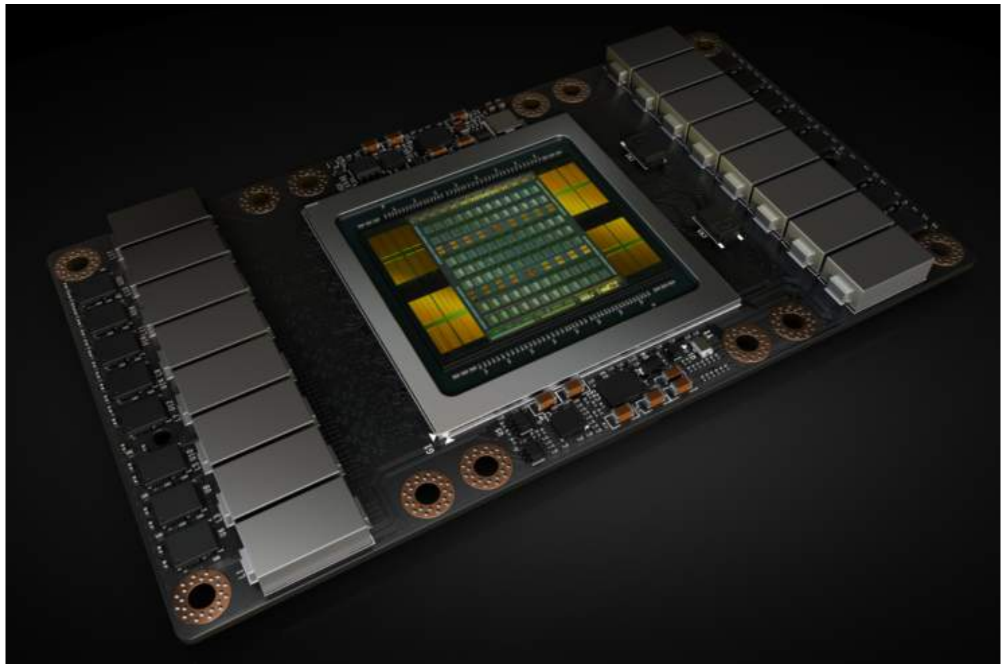
Nvidia PASCAL GP100 (2016)

- 10/20 TFLOPS FP32/FP16
- 16GB HBM – 750 GB/s
- 300W TDP
- 67 GFLOPS/W (FP16)
- 16nm process
- 160GB/s NV Link

Slide Source: Sze et al Survey of DNN Hardware, MICRO’16 Tutorial.
Data Source: NVIDIA

GPUs for Training

Nvidia Volta GV100 (2017)



- 15 FP32 TFLOPS
- 120 Tensor TFLOPS
- 16GB HBM2 @ 900GB/s
- 300W TDP
- 12nm process
- 21B Transistors
- die size: 815 mm2
- 300GB/s NVLink

Data Source: NVIDIA

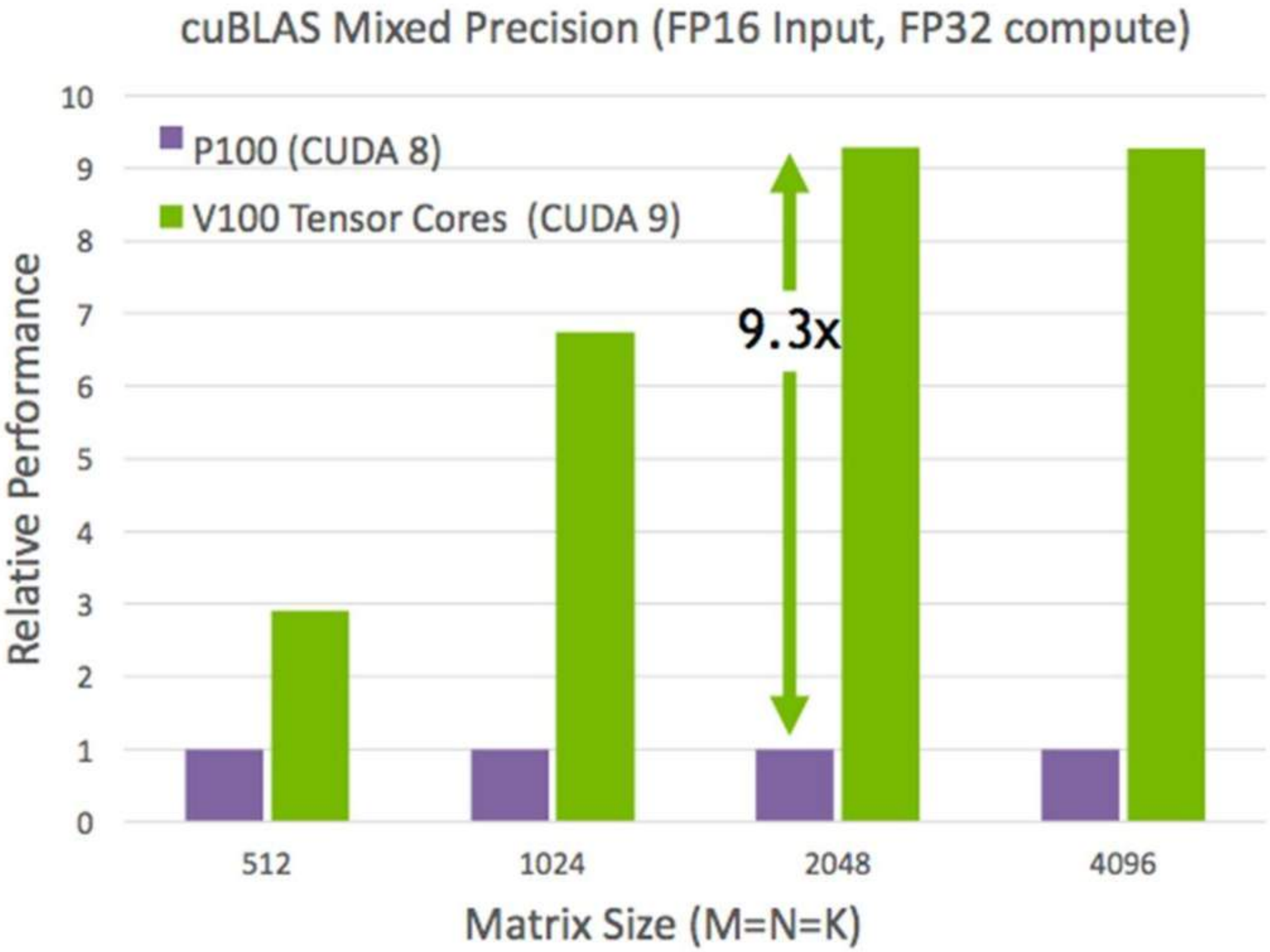
What’s new in Volta: Tensor Core

$$\underbrace{D}_{\text{FP16 or FP32}} = \underbrace{\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}}_{\text{FP16}} \underbrace{\begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}}_{\text{FP16}} + \underbrace{\begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}}_{\text{FP16 or FP32}}$$

a new instruction that performs 4x4x4 FMA mixed-precision operations per clock
12X increase in throughput for the Volta V100 compared to the Pascal P100

https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/

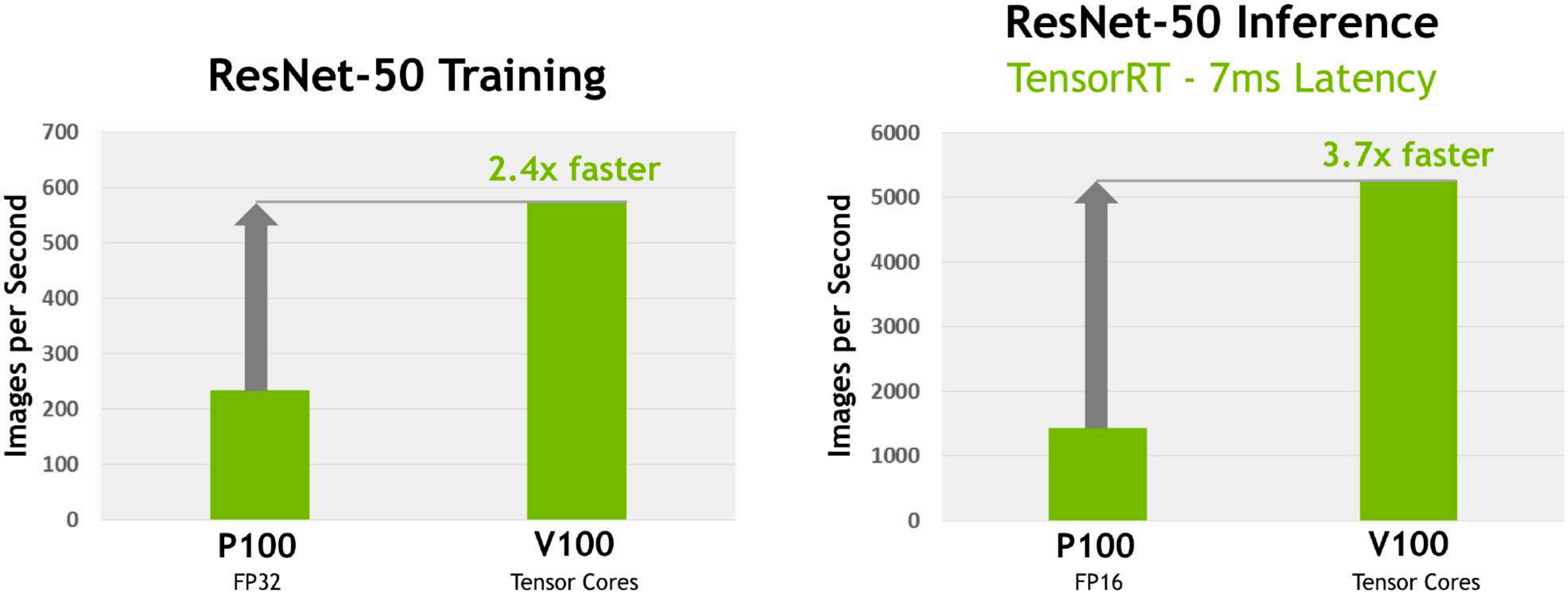
Pascal v.s. Volta



Tesla V100 Tensor Cores and CUDA 9 deliver up to 9x higher performance for GEMM operations.

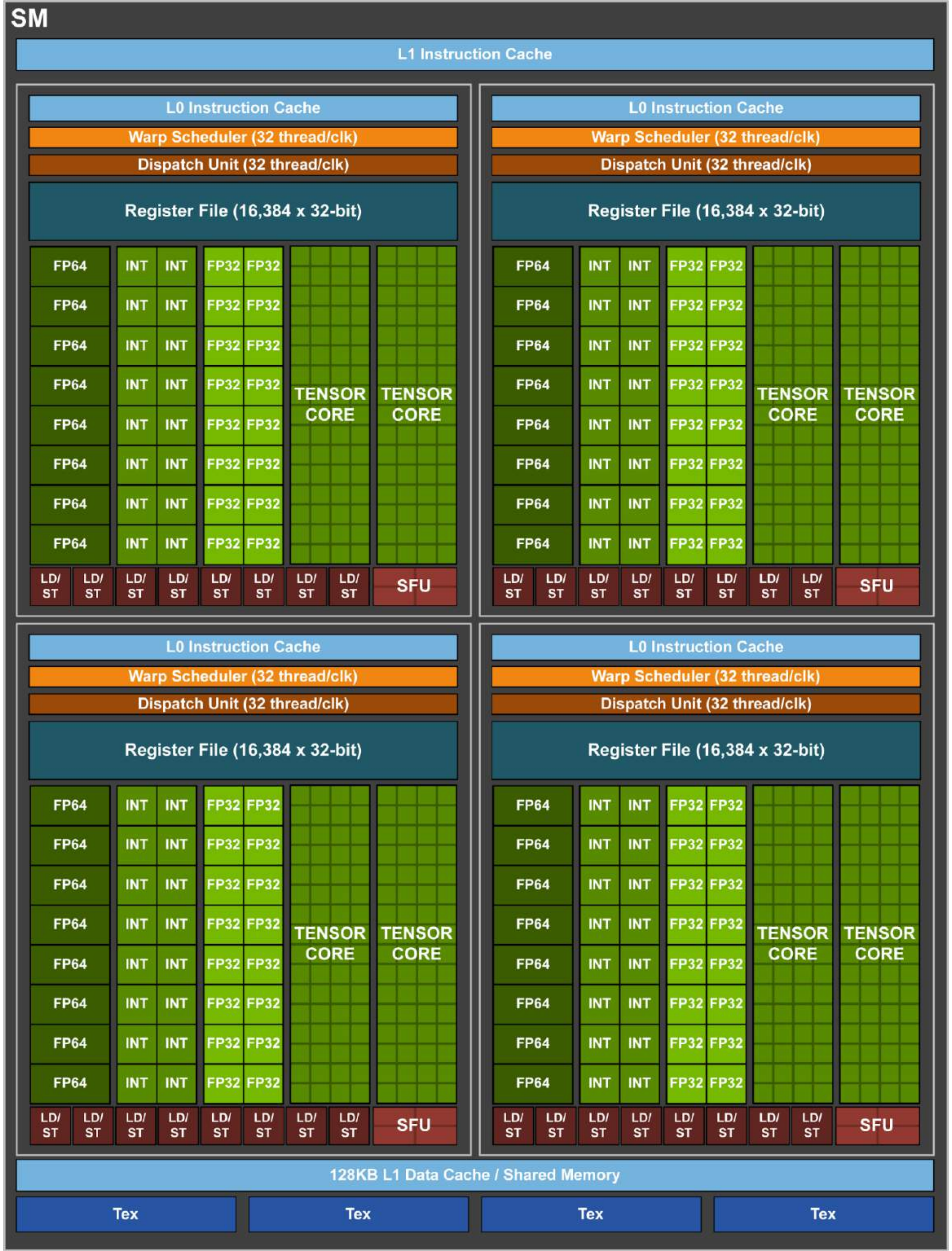
https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/

Pascal v.s. Volta



Left: Tesla V100 trains the ResNet-50 deep neural network 2.4x faster than Tesla P100.
Right: Given a target latency per image of 7ms, Tesla V100 is able to perform inference using the ResNet-50 deep neural network 3.7x faster than Tesla P100.

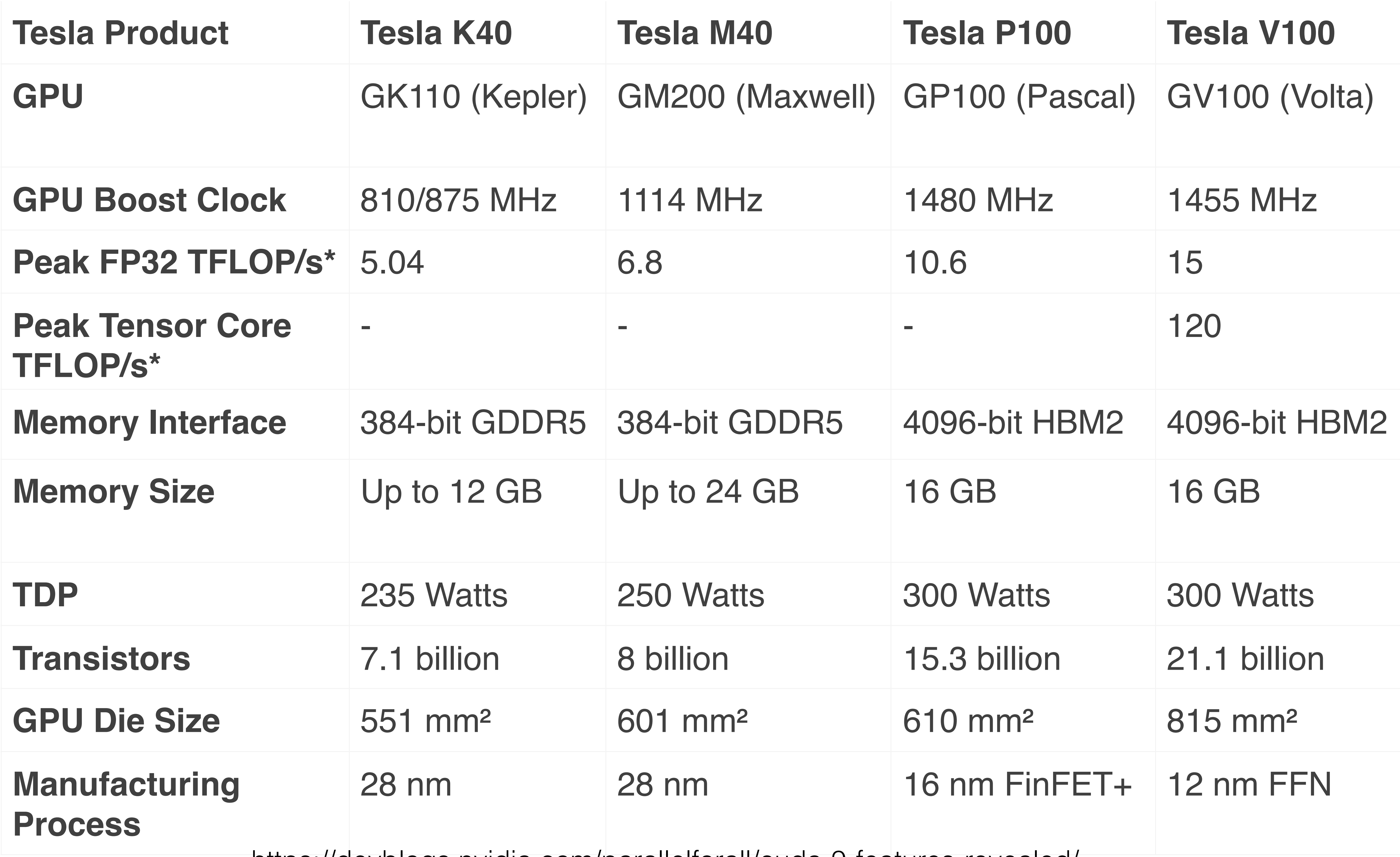
https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

The GV100 SM is partitioned into four processing blocks, each with:

- 8 FP64 Cores
- 16 FP32 Cores
- 16 INT32 Cores
- two of the new mixed-precision Tensor Cores for deep learning
- a new L0 instruction cache
- one warp scheduler
- one dispatch unit
- a 64 KB Register File.

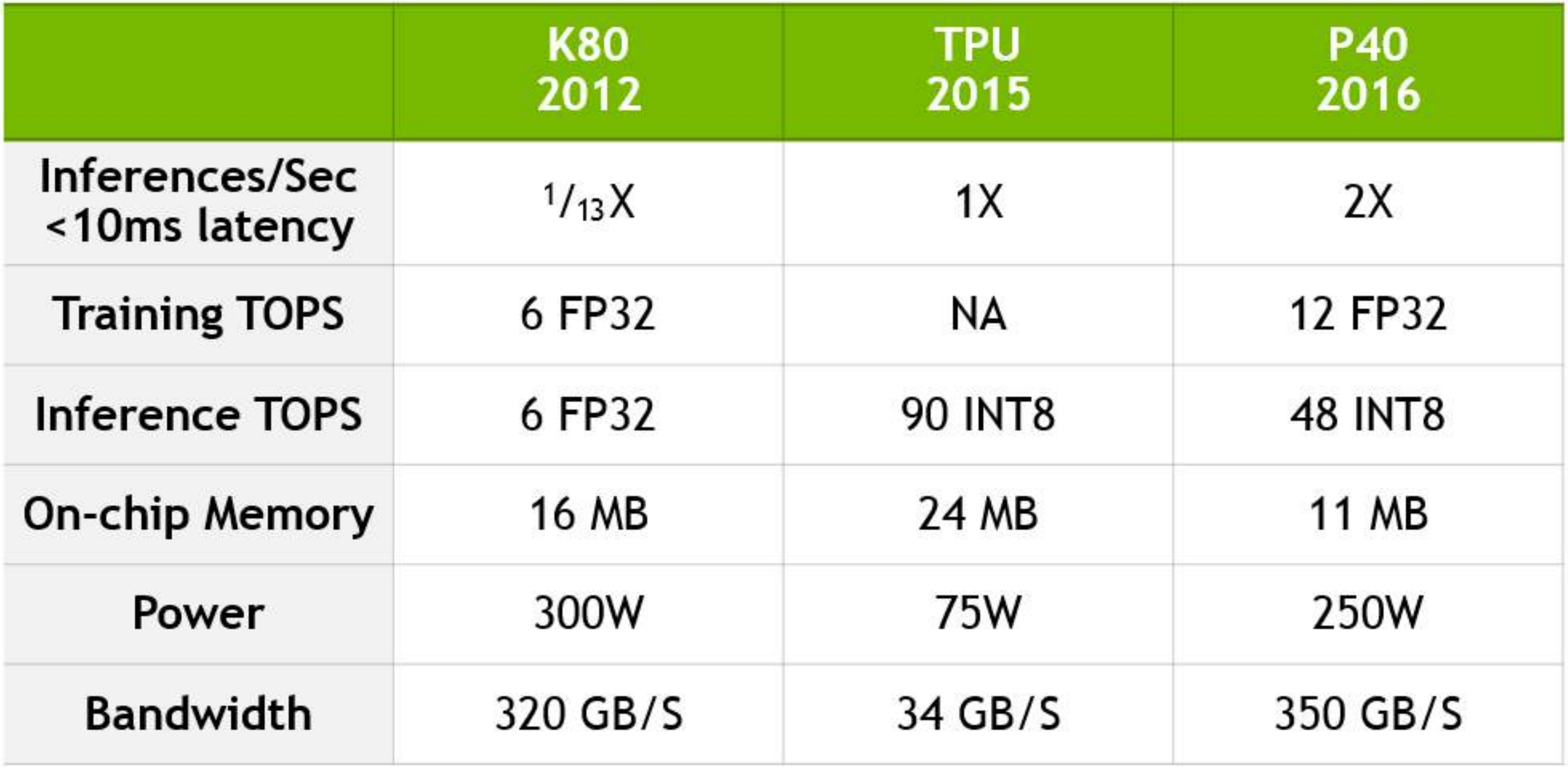
https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/



Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK110 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1455 MHz
Peak FP32 TFLOP/s*	5.04	6.8	10.6	15
Peak Tensor Core TFLOP/s*	-	-	-	120
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB	16 GB
TDP	235 Watts	250 Watts	300 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion	21.1 billion
GPU Die Size	551 mm^2	601 mm^2	610 mm^2	815 mm^2
Manufacturing Process	28 nm	28 nm	16 nm FinFET+	12 nm FFN

https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/

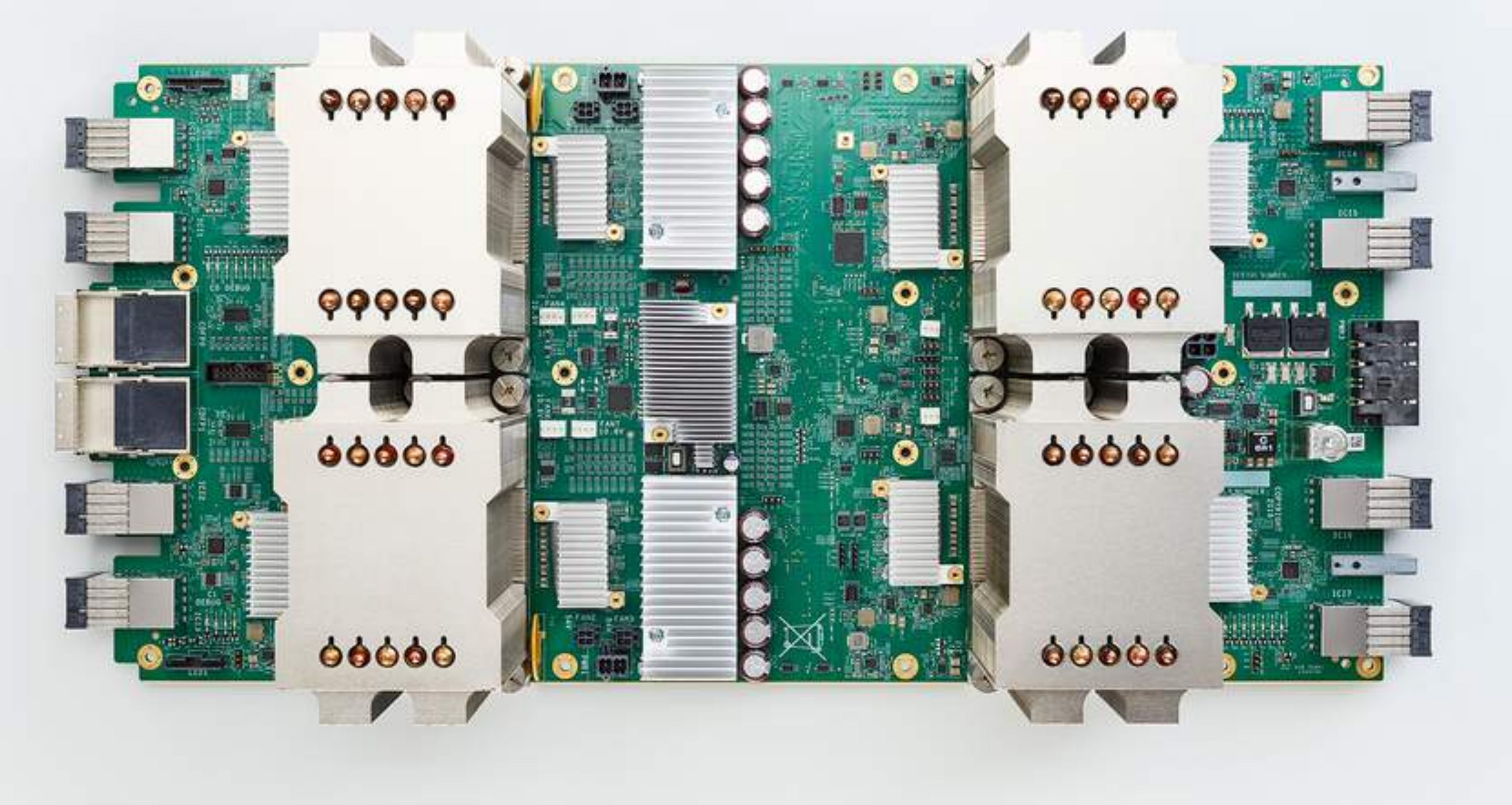
GPU / TPU



	K80 2012	TPU 2015	P40 2016
Inferences/Sec <10ms latency	$^{1}/_{13}$X	1X	2X
Training TOPS	6 FP32	NA	12 FP32
Inference TOPS	6 FP32	90 INT8	48 INT8
On-chip Memory	16 MB	24 MB	11 MB
Power	300W	75W	250W
Bandwidth	320 GB/S	34 GB/S	350 GB/S

https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/

Google Cloud TPU



Cloud TPU delivers up to 180 teraflops to train and run machine learning models.

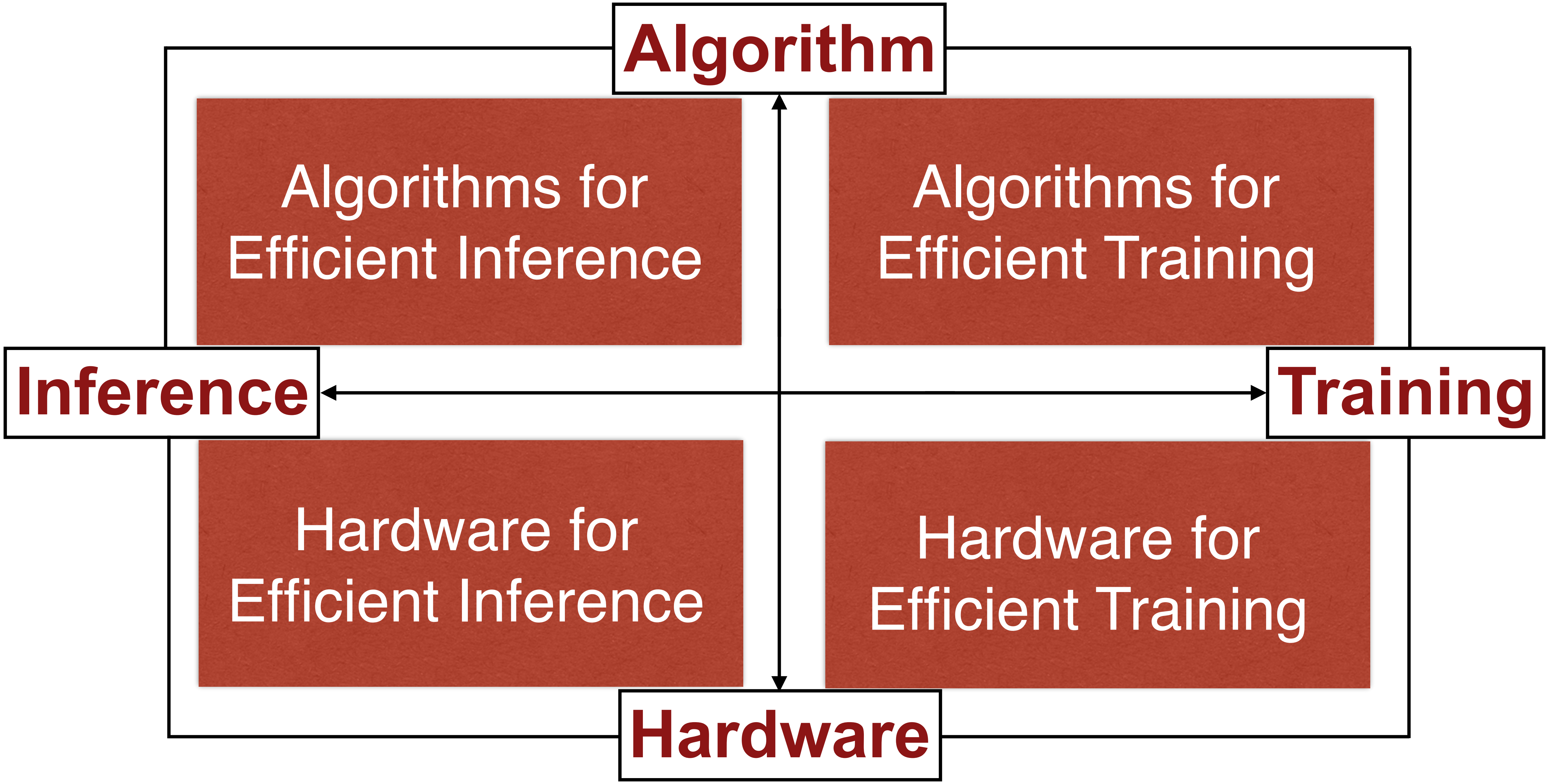
source: Google Blog

Google Cloud TPU

A "TPU pod" built with 64 second-generation TPUs delivers up to 11.5 petaflops of machine learning acceleration.

"One of our new large-scale translation models used to take a full day to train on 32 of the best commercially-available GPUs—now it trains to the same accuracy in an afternoon using just one eighth of a TPU pod."— Google Blog

Wrap-Up



Algorithm

Algorithms for Efficient Inference

Algorithms for Efficient Training

Inference

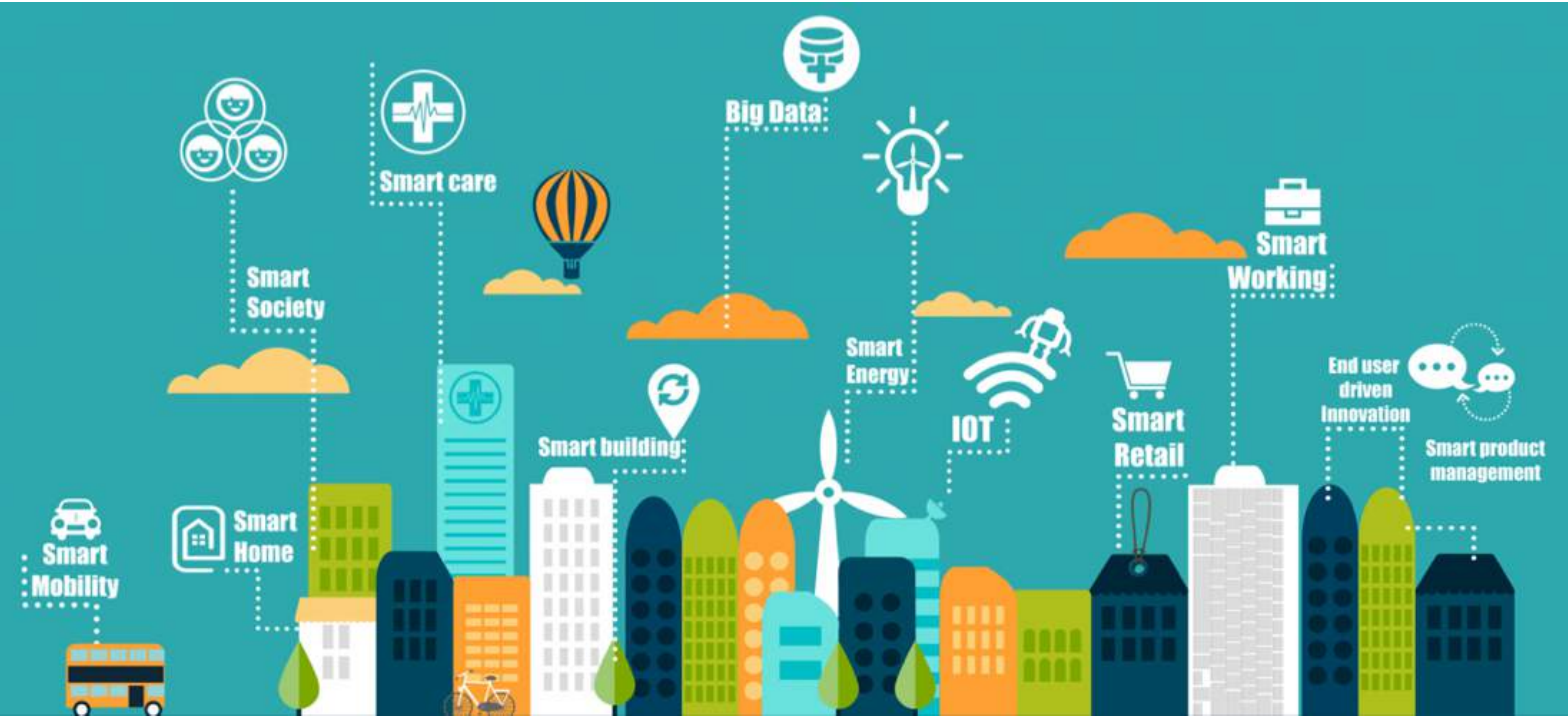
Training

Hardware for Efficient Inference

Hardware for Efficient Training

Hardware

Future

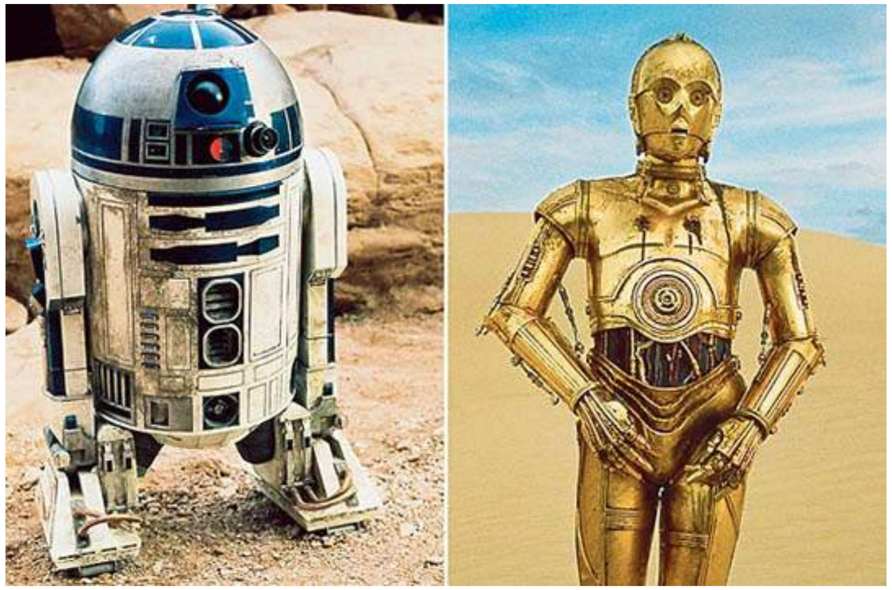


Smart   Low Latency   Privacy   Mobility   Energy-Efficient

Outlook: the Focus for Computation

PC Era

Mobile-First Era



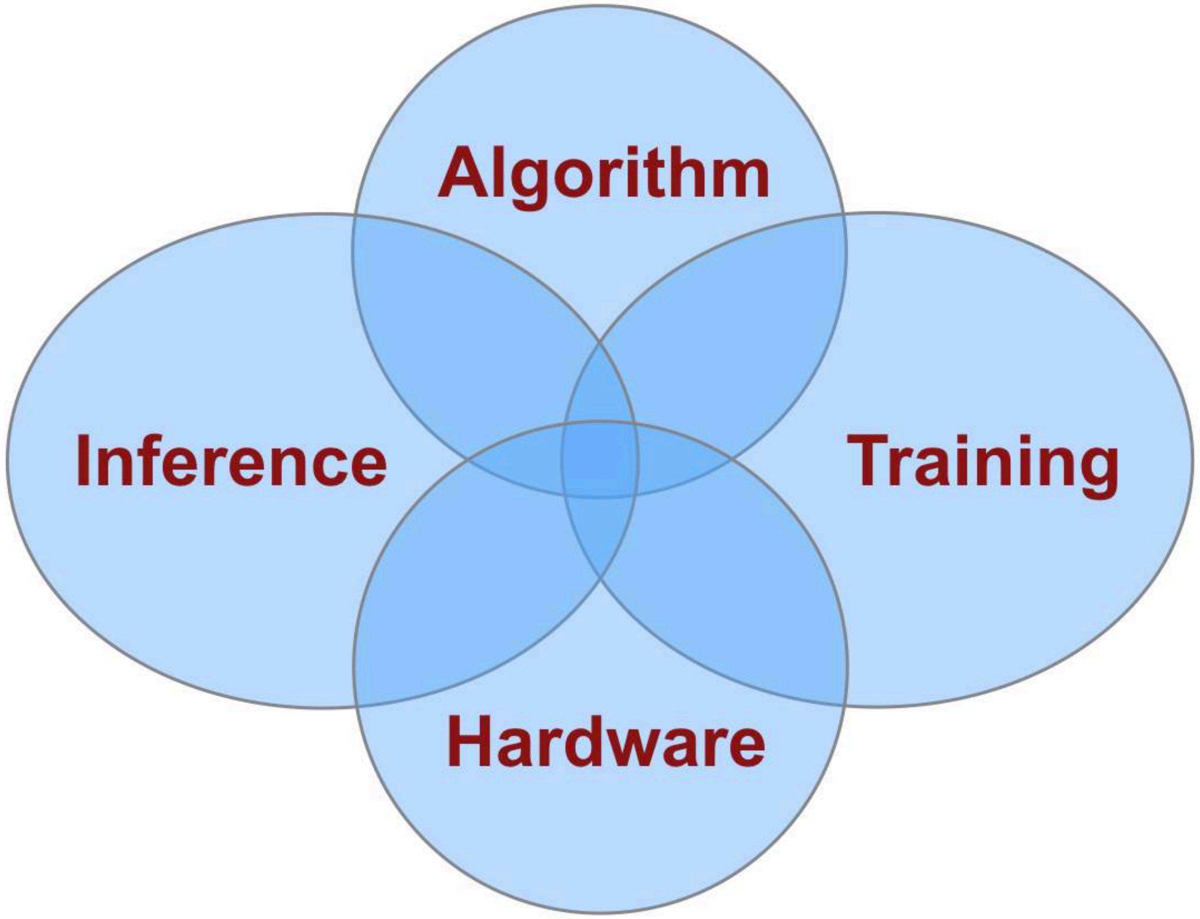
AI-First Era

Computing → Mobile Computing → Brain-Inspired Cognitive Computing

Sundar Pichai, Google IO, 2016

Thank you!

stanford.edu/~songhan


Algorithm
Inference
Training
Hardware