High-Performance Hardware for Machine Learning

NIPS Tutorial

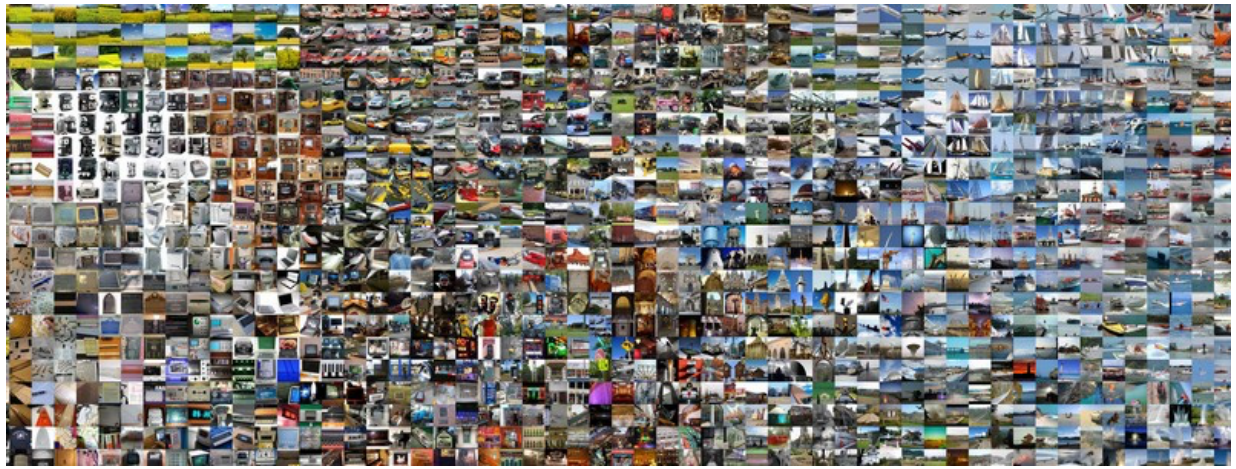
12/7/2015

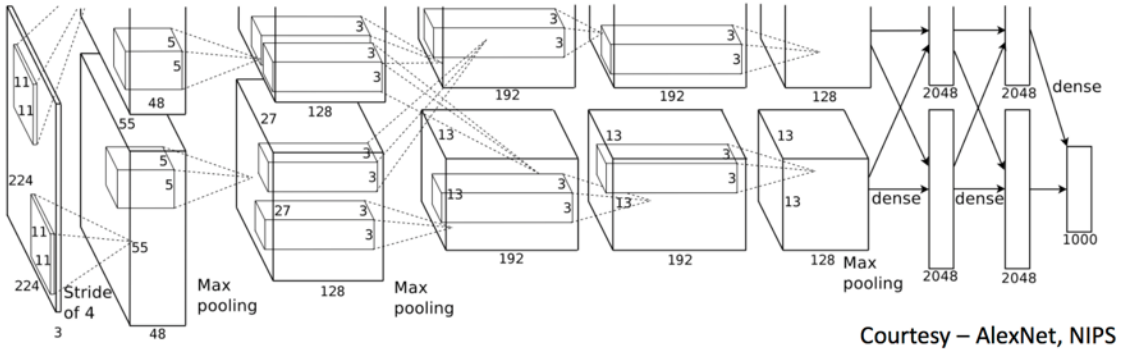
Prof. William Dally

Stanford University

NVIDIA Corporation

Hardware and Data enable DNNs





Courtesy – AlexNet, NIPS 2012

The Need for Speed

Larger data sets and models lead to better accuracy but also increase computation time. Therefore progress in deep neural networks is limited by how fast the networks can be computed.

Likewise the application of convnets to low latency inference problems, such as pedestrian detection in self driving car video imagery, is limited by how fast a small set of images, possibly a single image, can be classified.

More data ➔ Bigger Models ➔More Need for Compute

But Moore's law is no longer providing more compute…

Lavin & Gray, Fast Algorithms for Convolutional Neural Networks, 2015

Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

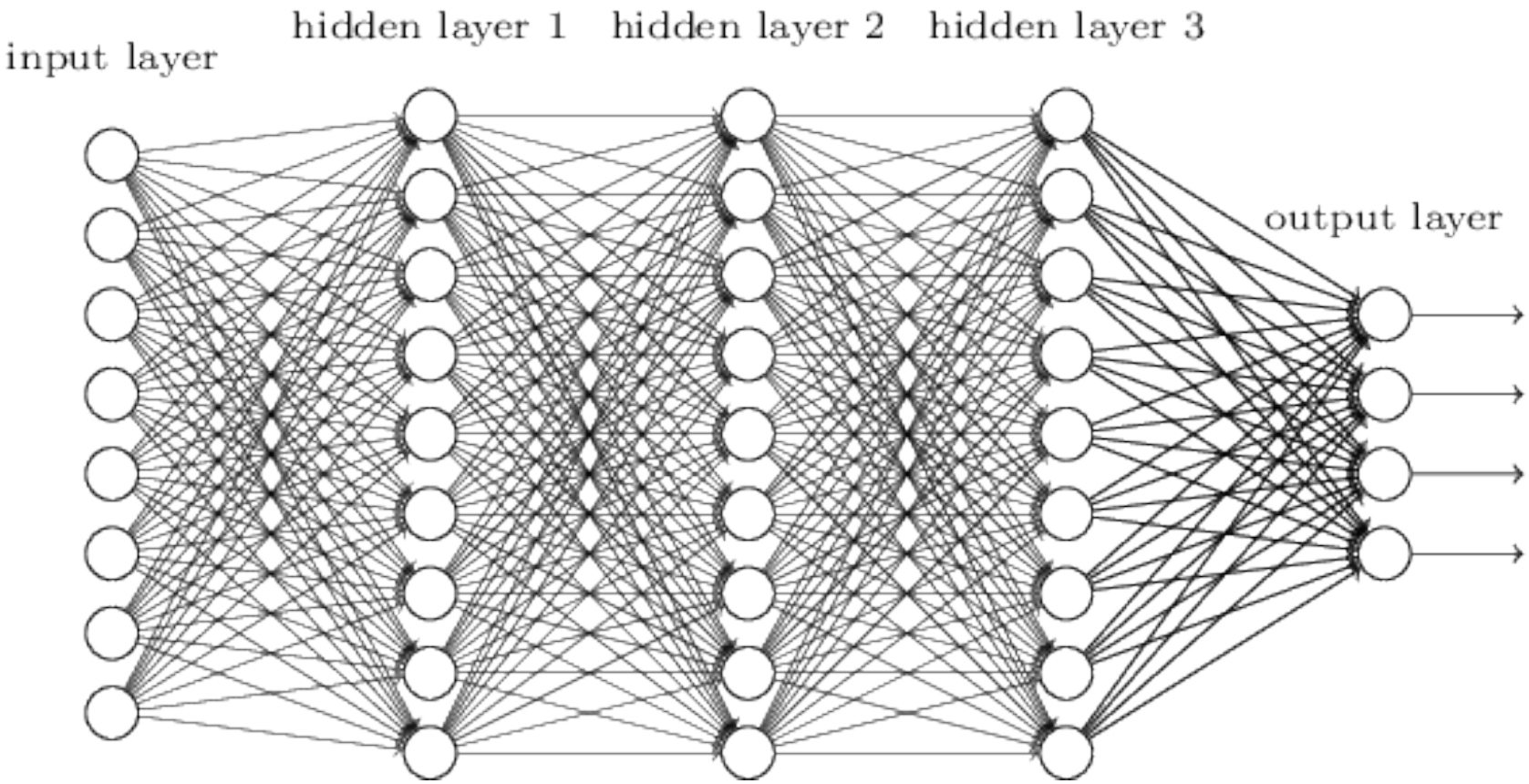
Acceleration

- Run a network faster (Performance, inf/s)
- Run a network more efficiently
  - Energy (inf/J)
  - Cost (inf/s$)

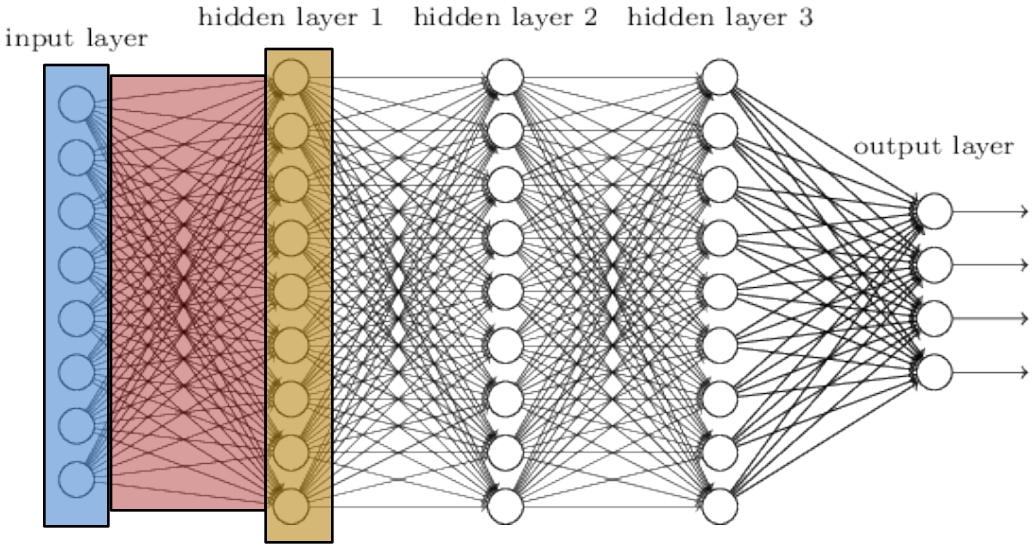
- Inference
  - Just running the network forward

- Training
  - Running the network forward
  - Back-propagation of gradient
  - Update of parameters

What Network? DNNs, CNNs, and RNNs



DNN, key operation is dense M x V



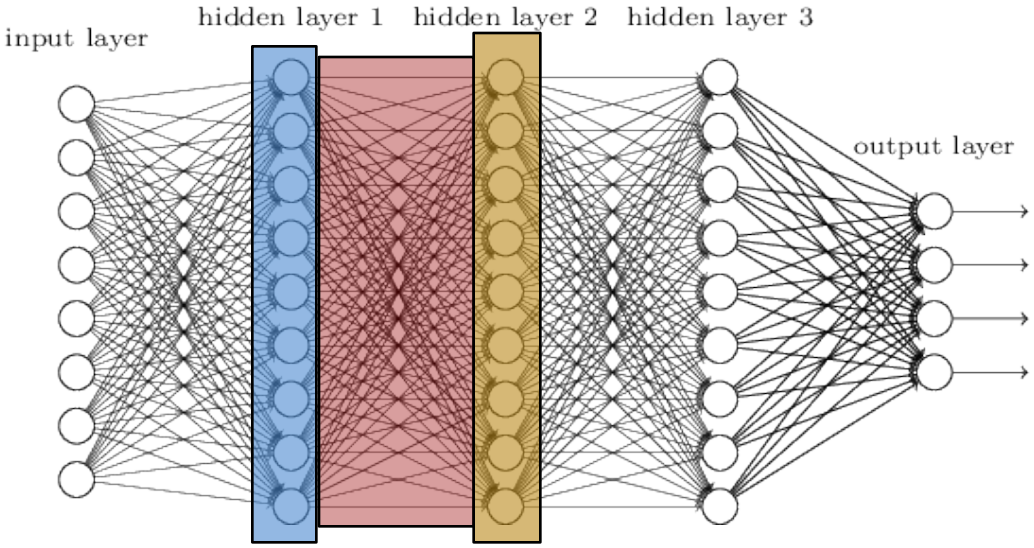
$b_i = W_{ij} \times a_j$

Output activations

weight matrix

Input activations

DNN, key operation is dense M x V



Repeat for each layer

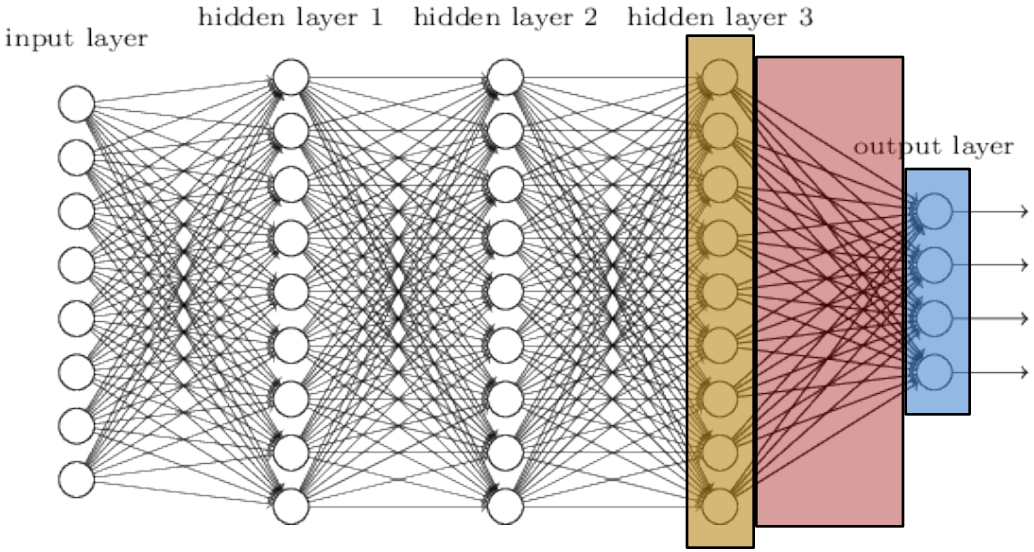
$b_i = W_{ij} \times a_j$

Output activations

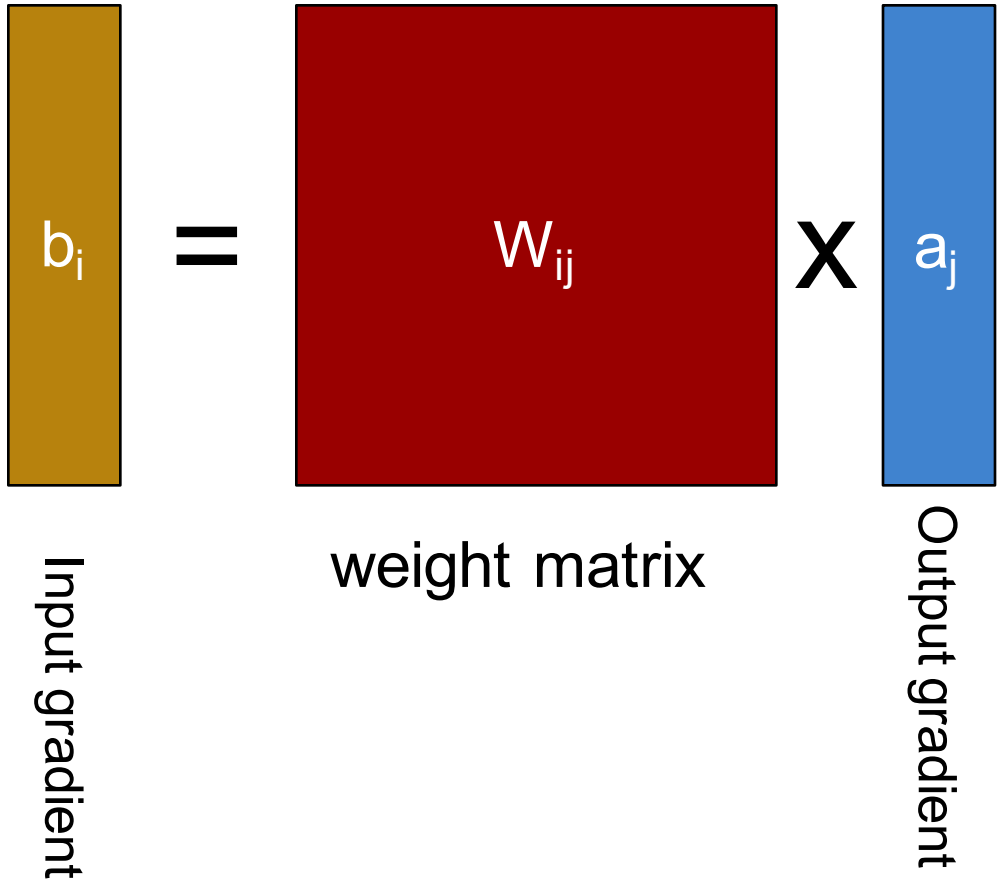
weight matrix

Input activations

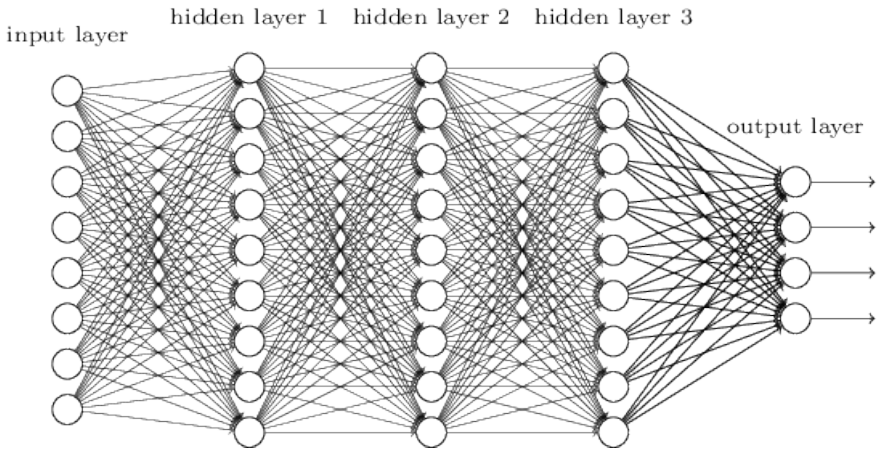
DNN, key operation is dense M x V



Backpropagation just does this backward



Training, and Latency Insensitive Networks can be Batched – operation is M x M – gives re-use of weights



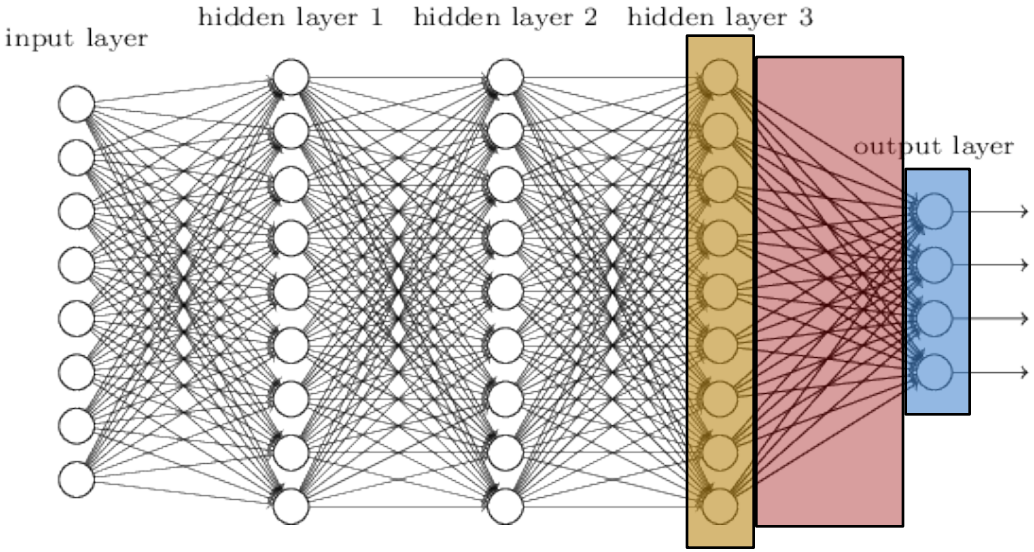
$b_{ik} = W_{ij} \times a_{jk}$

Output activations

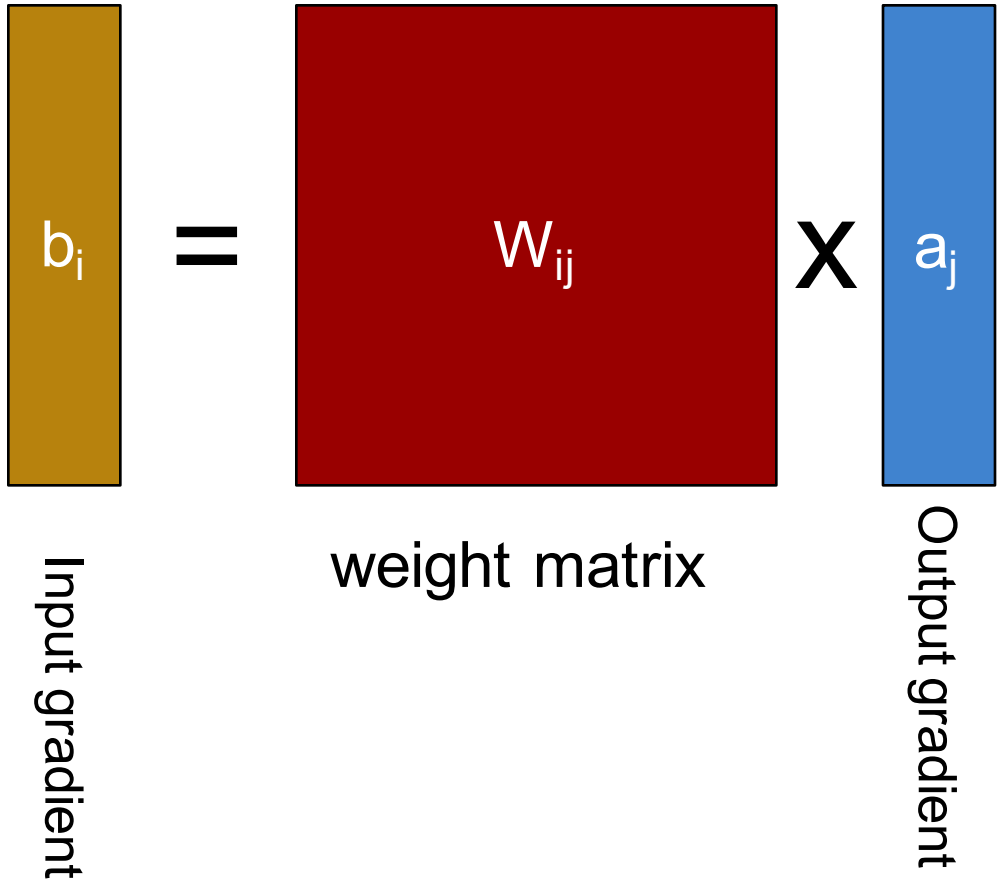
weight matrix

Input activations

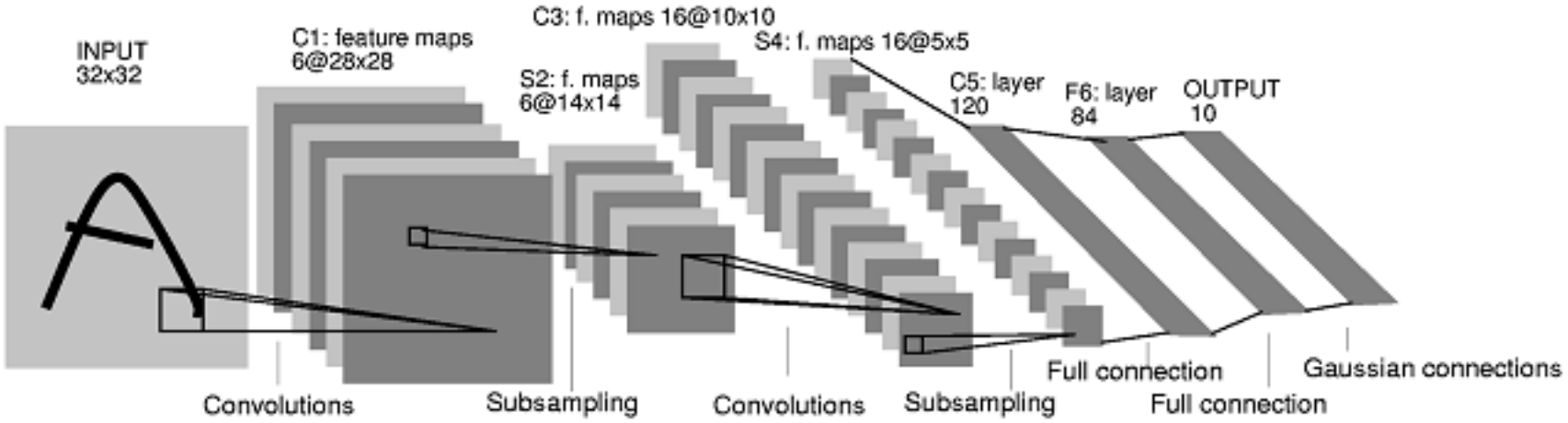
For real time you can't batch
And there is sparsity in both weights and activations
key operations is spM x spV



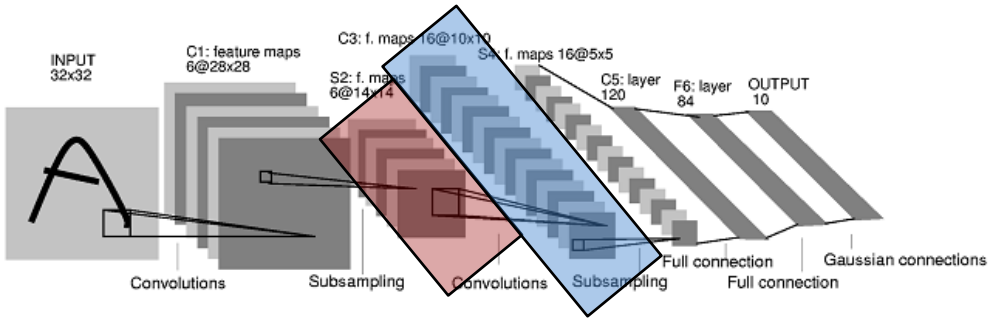
Backpropagation just does this backward



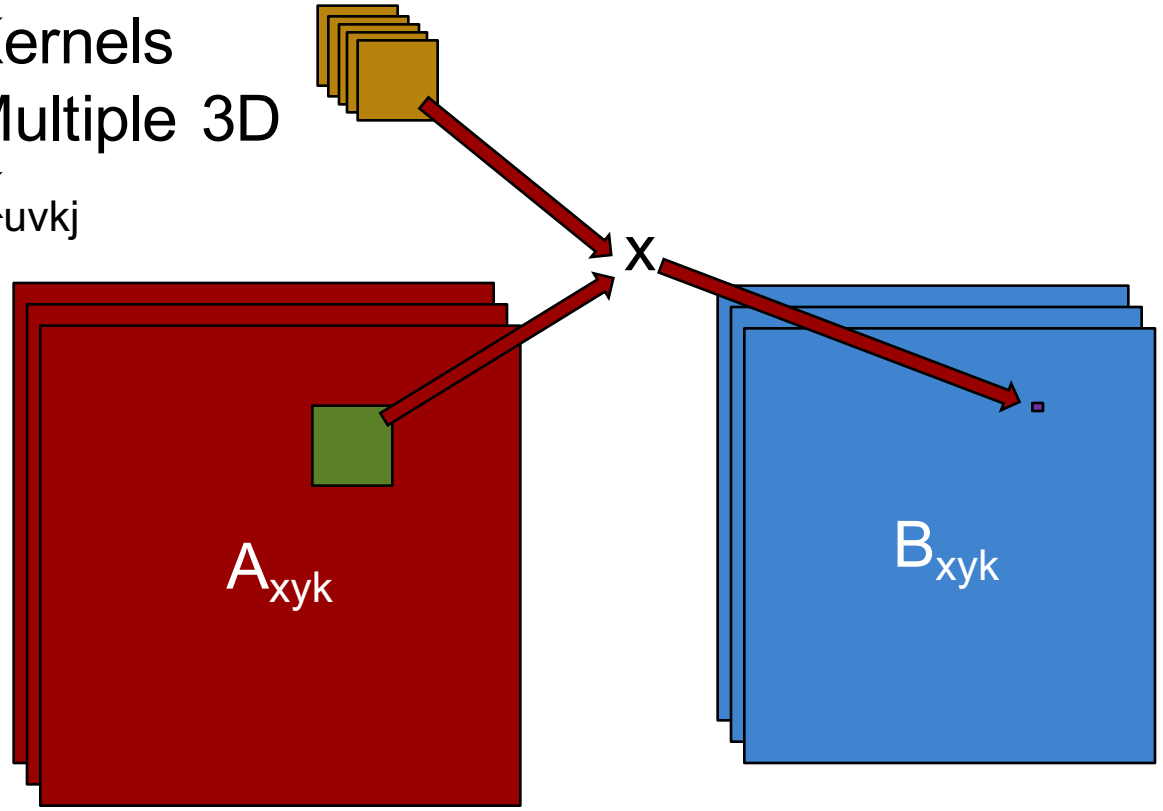
CNNs – For Image Inputs, Convolutional stages act as trained feature detectors



CNNs require Convolution in addition to M x V



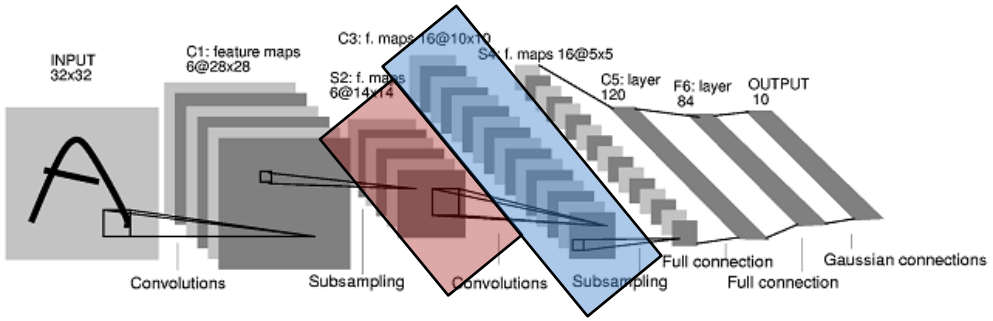
Kernels
Multiple 3D
K_{uvkj}



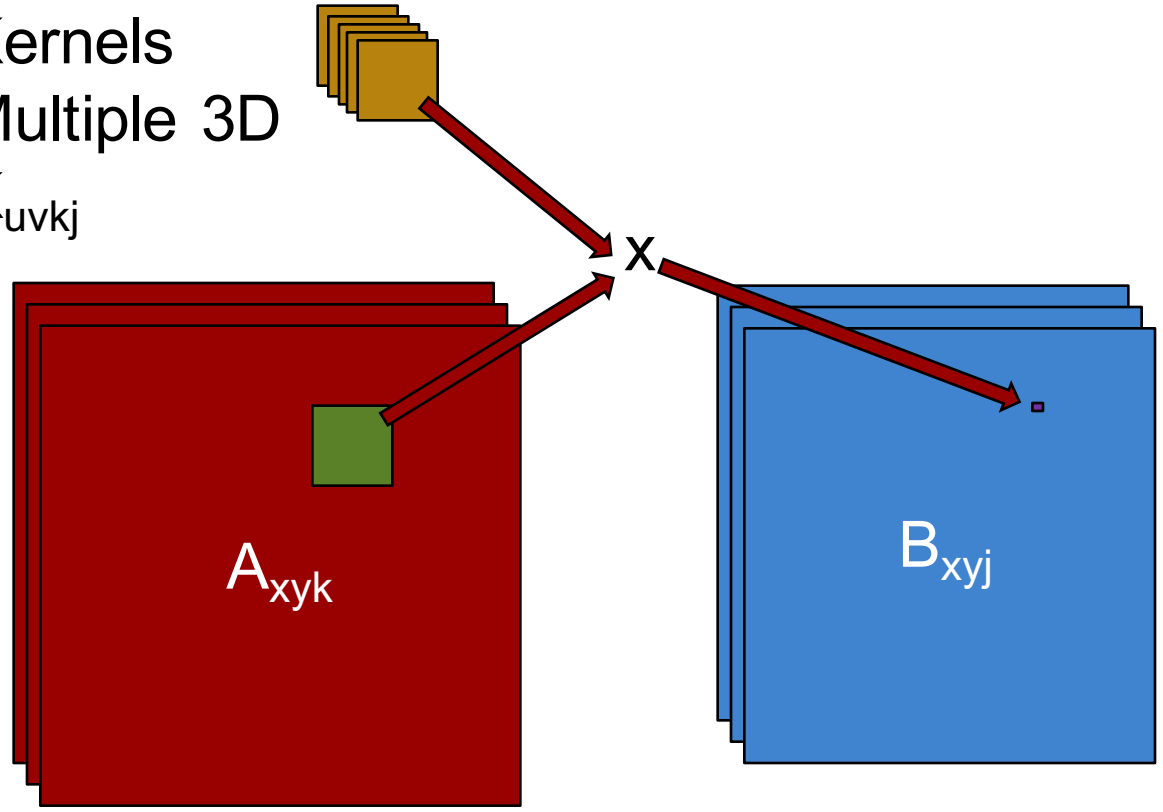
Input maps
A_{xyk}

Output maps
B_{xyj}

CNNs require Convolution in addition to M x V



Kernels
Multiple 3D
K_{uvkj}

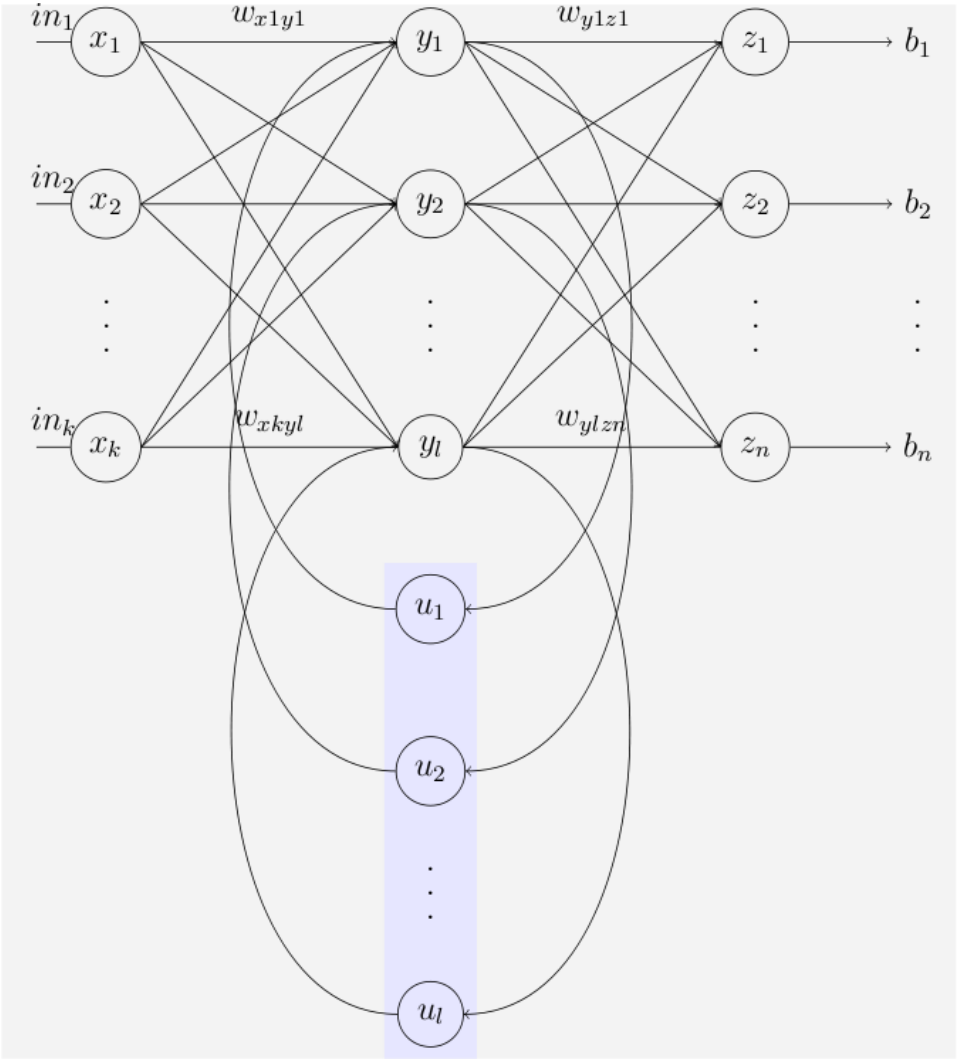


6D Loop
For each output map j
  For each input map k
    For each pixel x,y
      For each kernel element u,v
        B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}

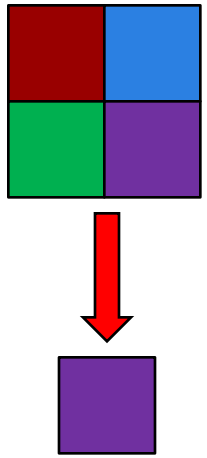
Input maps
A_{xyk}

Output maps
B_{xyj}

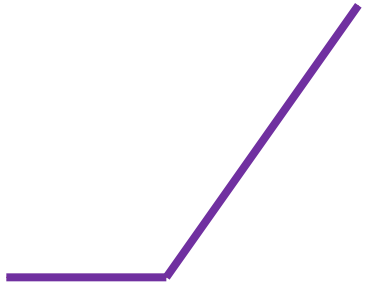
RNNs


in_1
x_1
w_{x1y1}
y_1
w_{y1z1}
z_1
b_1
in_2
x_2
y_2
z_2
b_2
in_k
x_k
w_{xkyl}
y_l
w_{ylzn}
z_n
b_n
u_1
u_2
u_l

Some Other Operations



Pooling

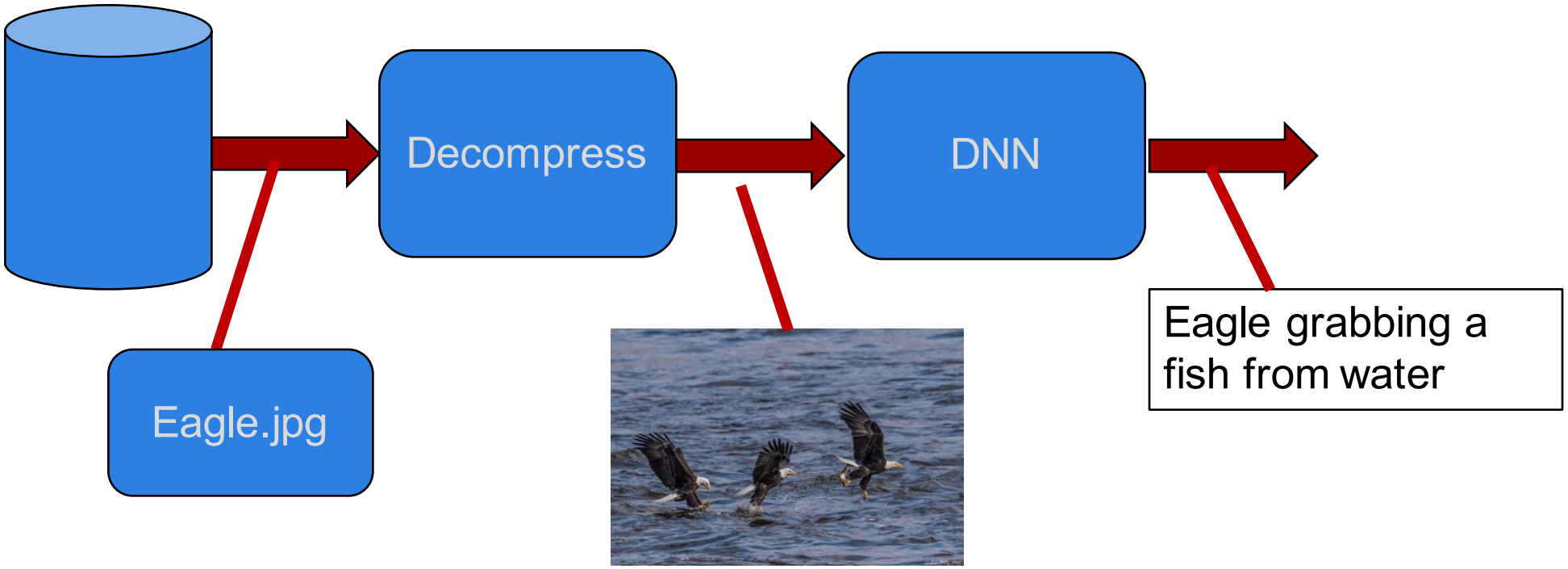


ReLU
(or other non-linear function)

$$w_{ij} \mathrel{+}= \alpha a_j g_i$$

Weight Update

Infrastructure



Summary of the Problem

- Run DNNs, CNNs, and RNNs
  - For training and inference
  - Can batch if not latency sensitive
- Optimize
  - Speed inf/s
  - Efficiency inf/J, inf/s$
- Key operations are
  - M x V
    - M x M if batched
    - May be sparse (spM x spV)
  - Convolution
- Also
  - Pooling, non-linear operator (ReLU), weight update

Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

Baseline Performance Xeon E5-2698 – Single Core

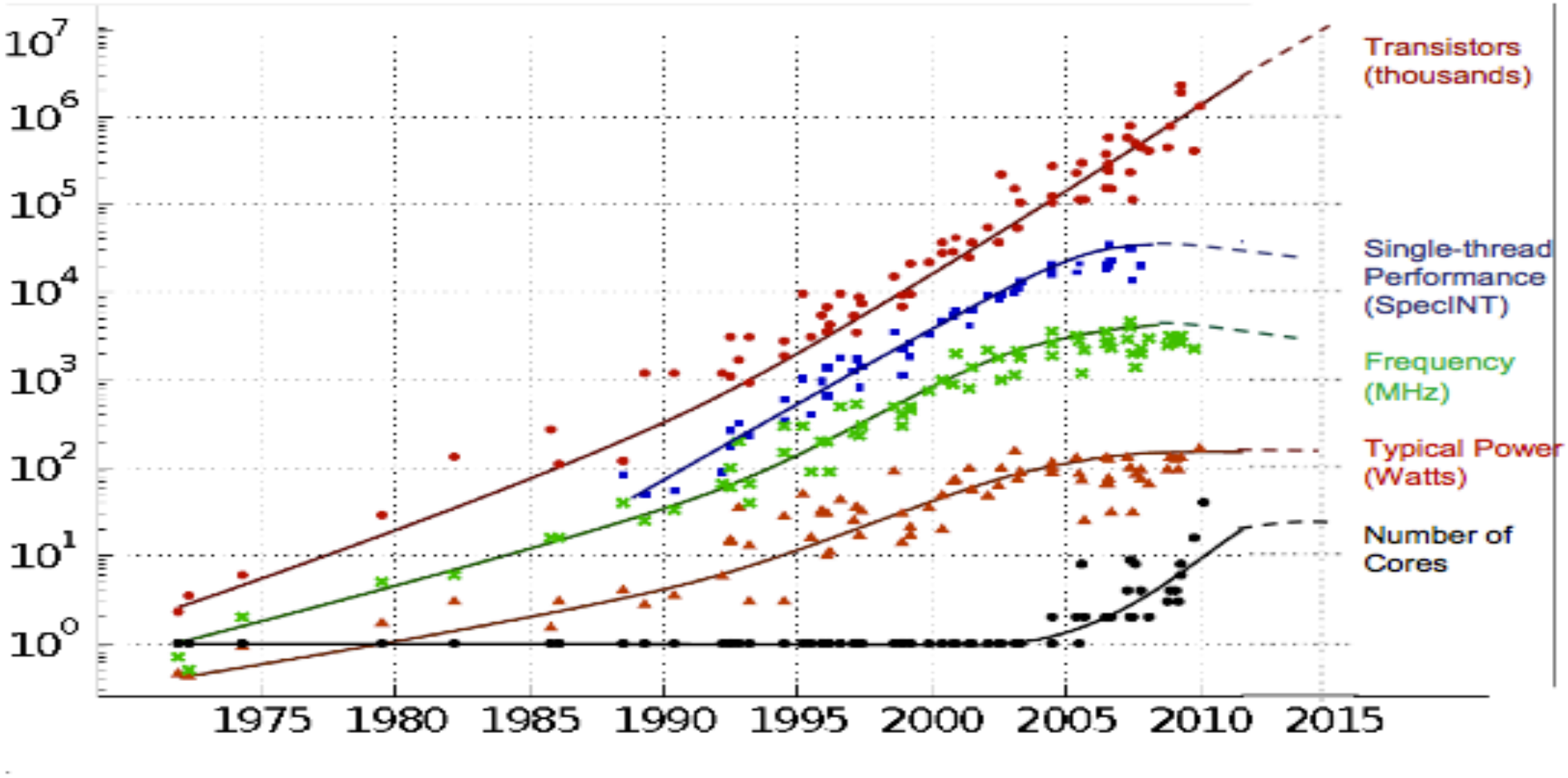
AlexNet – inference, batched

30 f/s

3.2 f/J

Most ops on AVX (SIMD) units

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

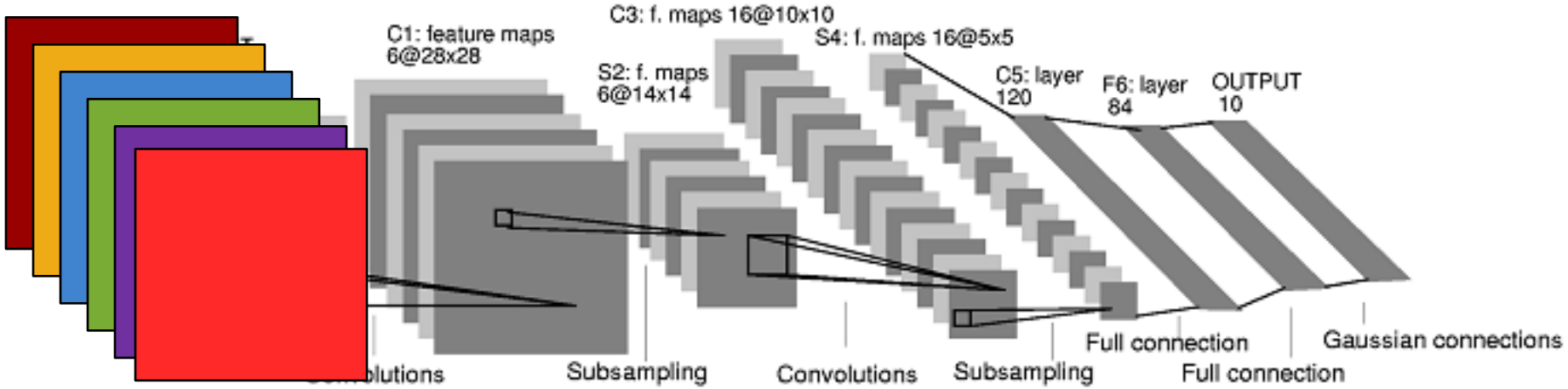
C Moore, *Data Processing in ExaScale-ClassComputer Systems,* Salishan, April 2011

Outline

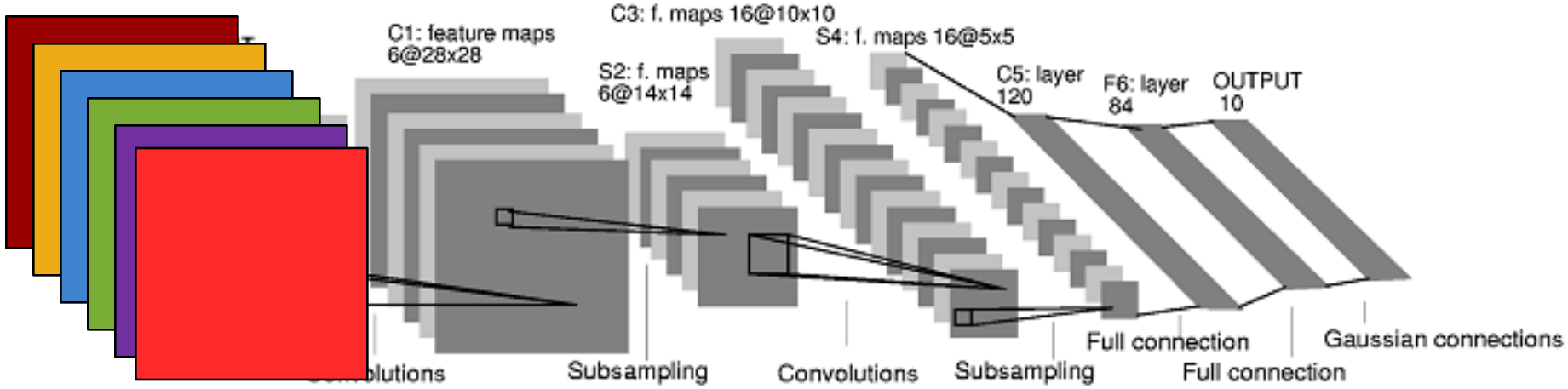
- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

To go faster, use more processors

Lots of parallelism in a DNN



Lots of parallelism in a DNN

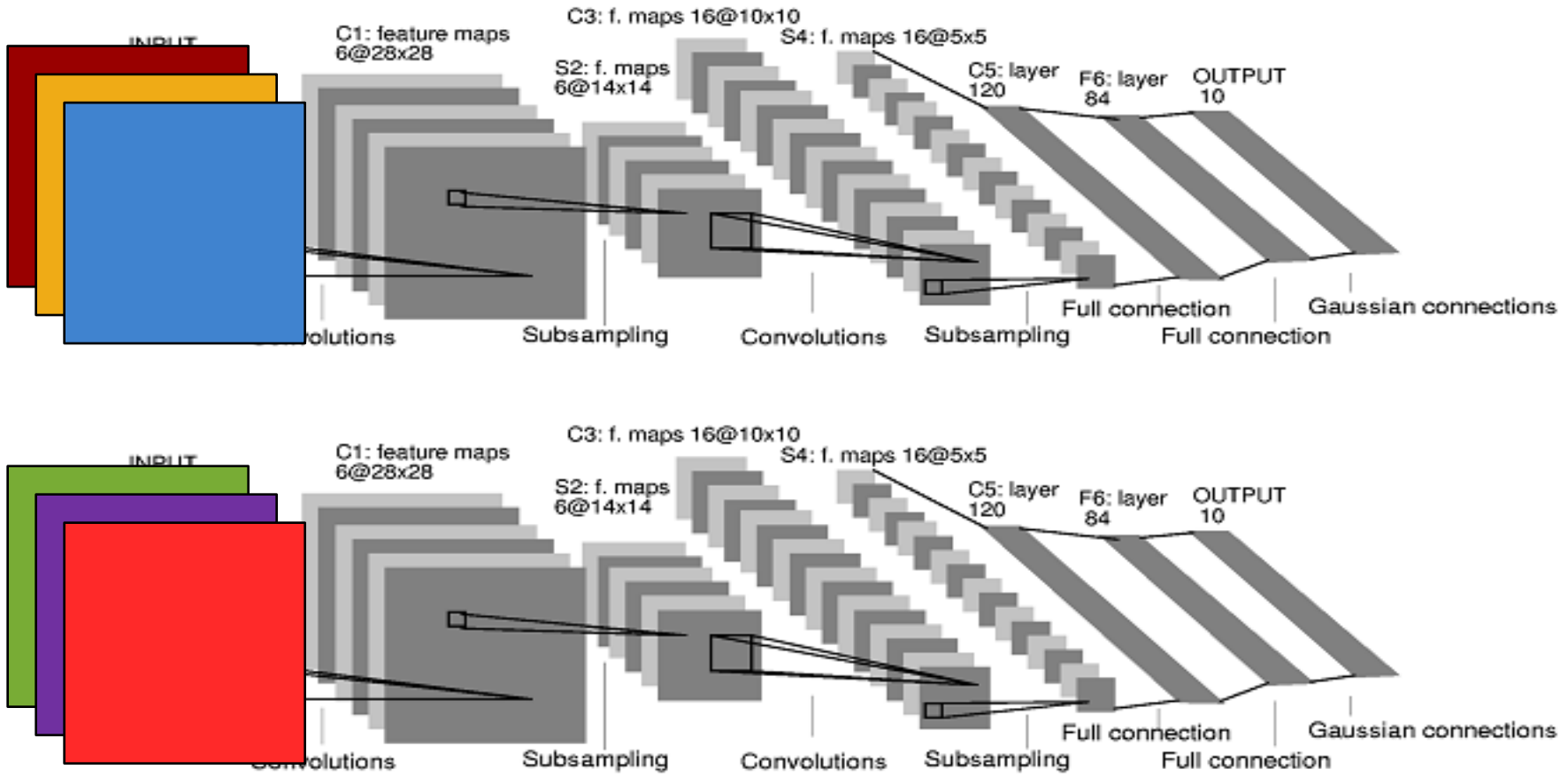


- Inputs
- Points of a feature map
- Filters
- Elements within a filter

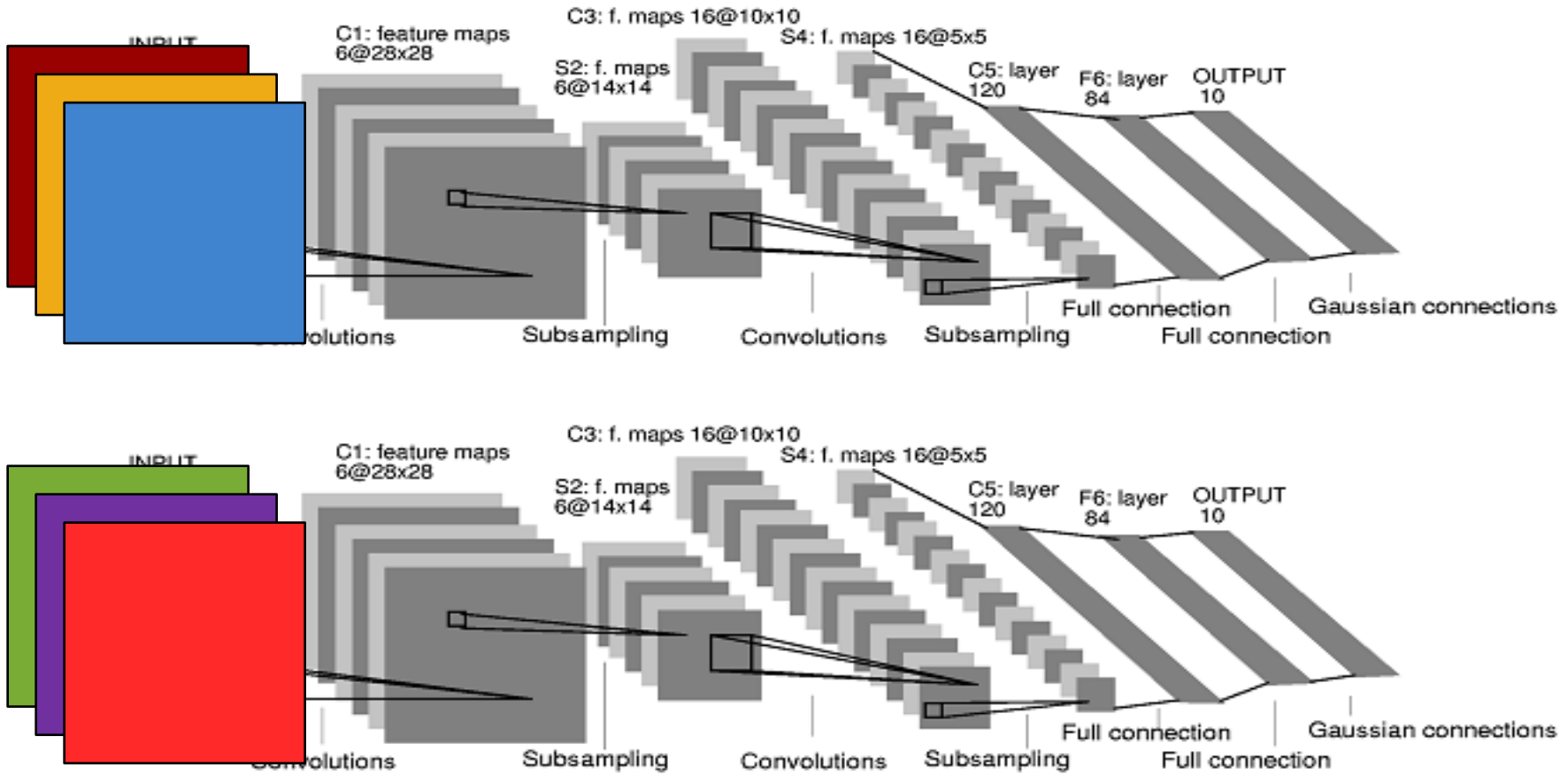
- Multiplies within layer are independent
- Sums are reductions
- Only layers are dependent
- No data dependent operations

  => can be statically scheduled

Data Parallel – Run multiple inputs in parallel

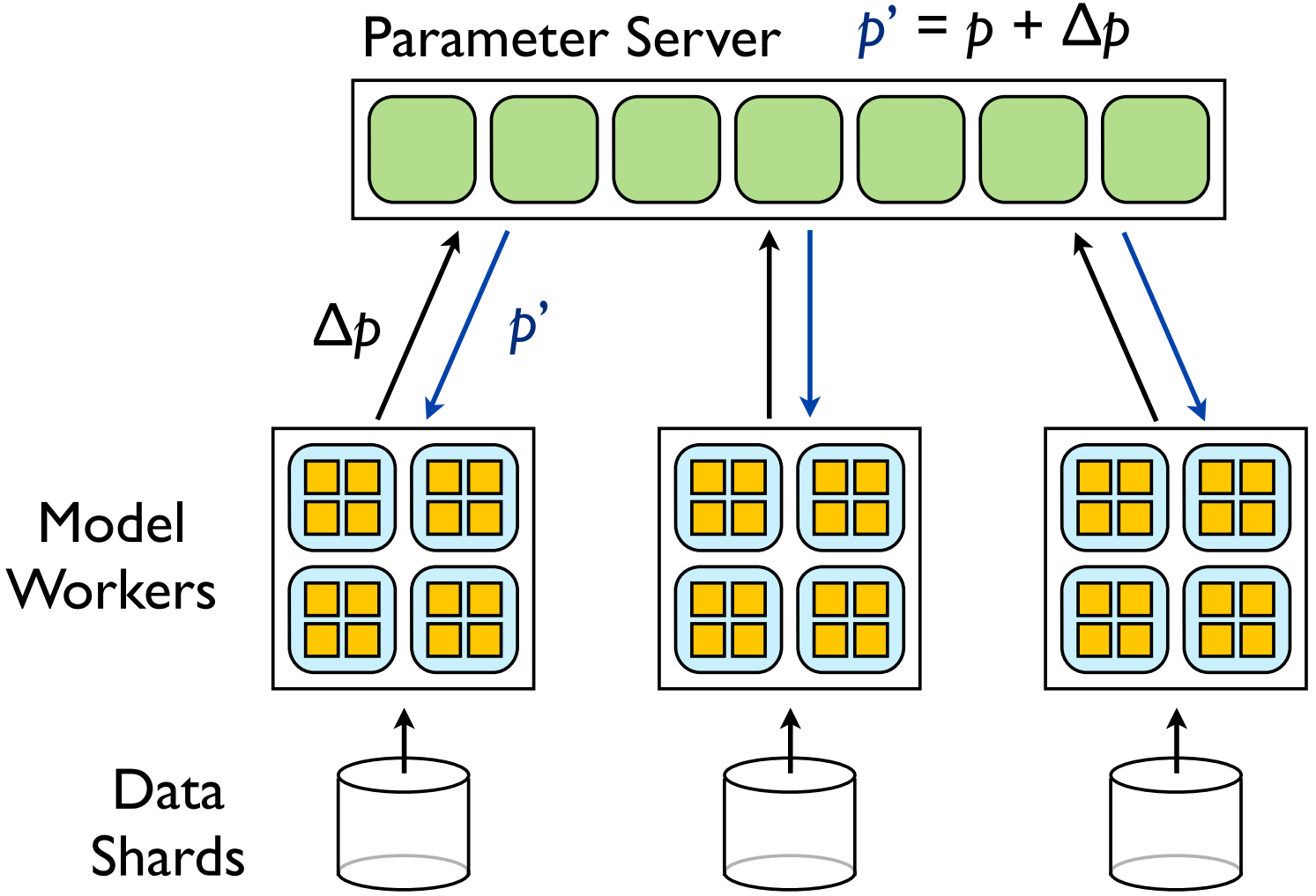


Data Parallel – Run multiple inputs in parallel



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

Parameter Update

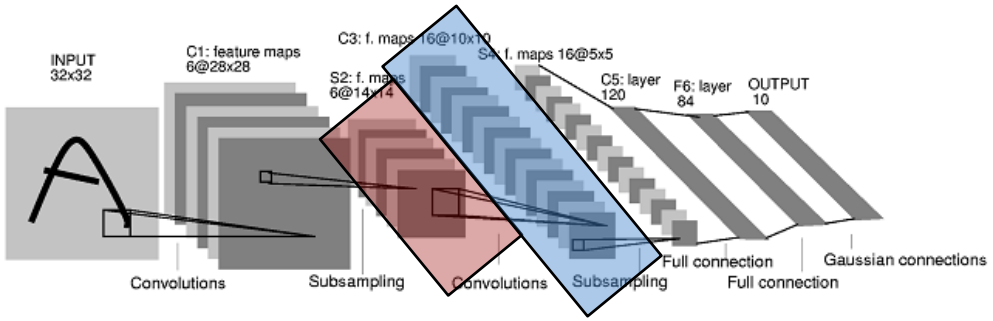


Large Scale Distributed Deep Networks, Jeff Dean et al., 2013

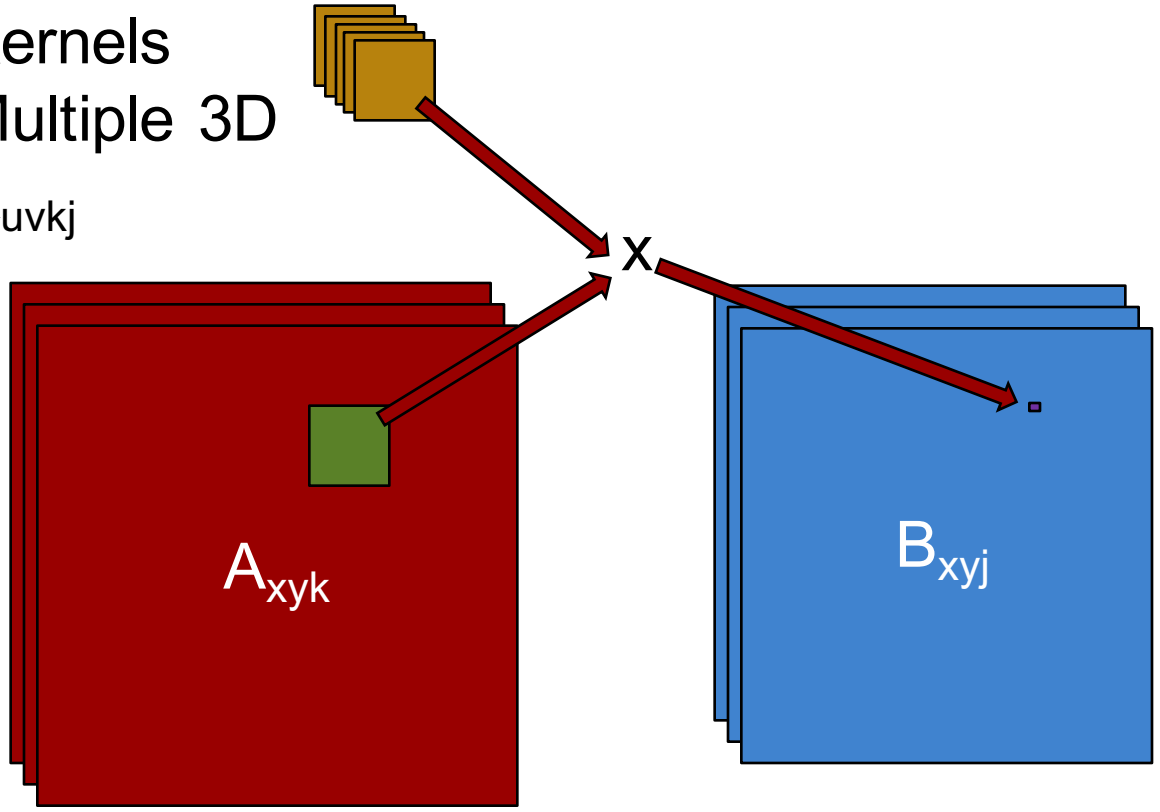
Model Parallel

Split up the Model – i.e. the network

Model-Parallel Convolution



Kernels
Multiple 3D
K_{uvkj}

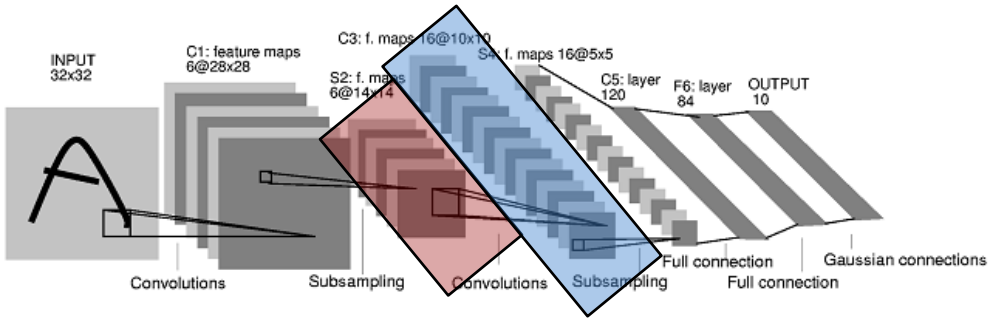


6D Loop
For each output map j
  For each input map k
    For each pixel x,y
      For each kernel element u,v
        B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}

Input maps
A_{xyk}

Output maps
B_{xyj}

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



6D Loop

Forall region XY

  For each output map j

    For each input map k

      For each pixel x,y in XY

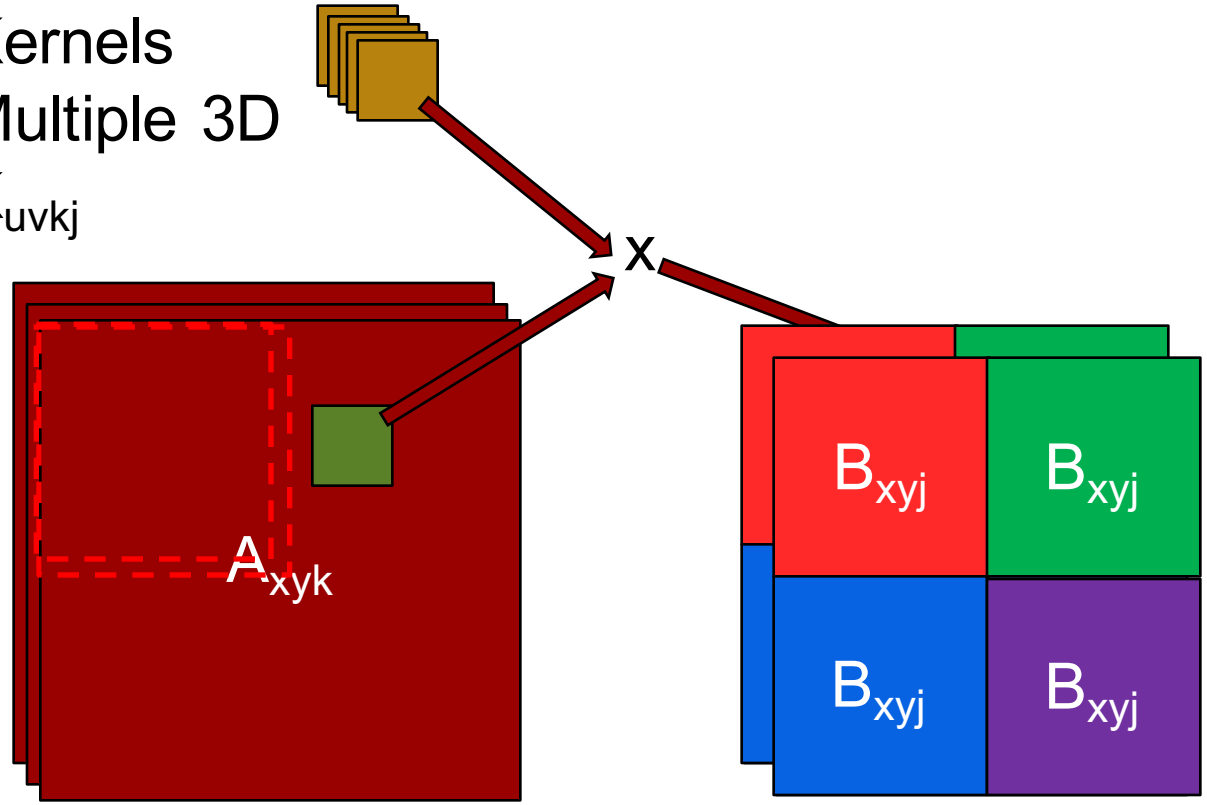
        For each kernel element u,v

$$B_{xyj} \mathrel{+}= A_{(x-u)(y-v)k} \times K_{uvkj}$$

Kernels

Multiple 3D

K_{uvkj}



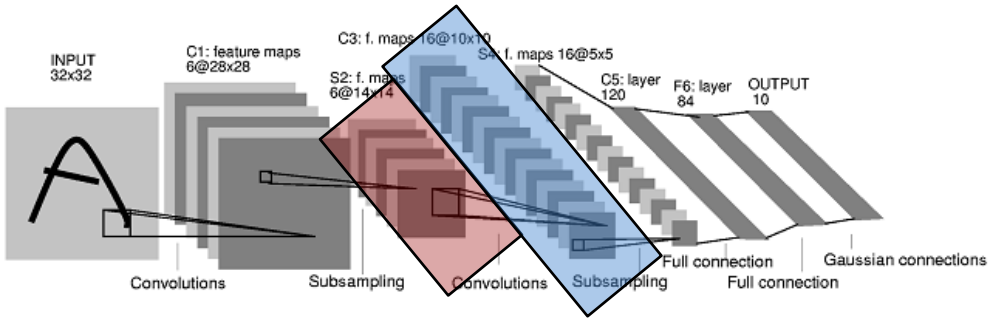
Input maps

A_{xyk}

Output maps

B_{xyj}

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



6D Loop

Forall output map j

  For each input map k

    For each pixel x,y

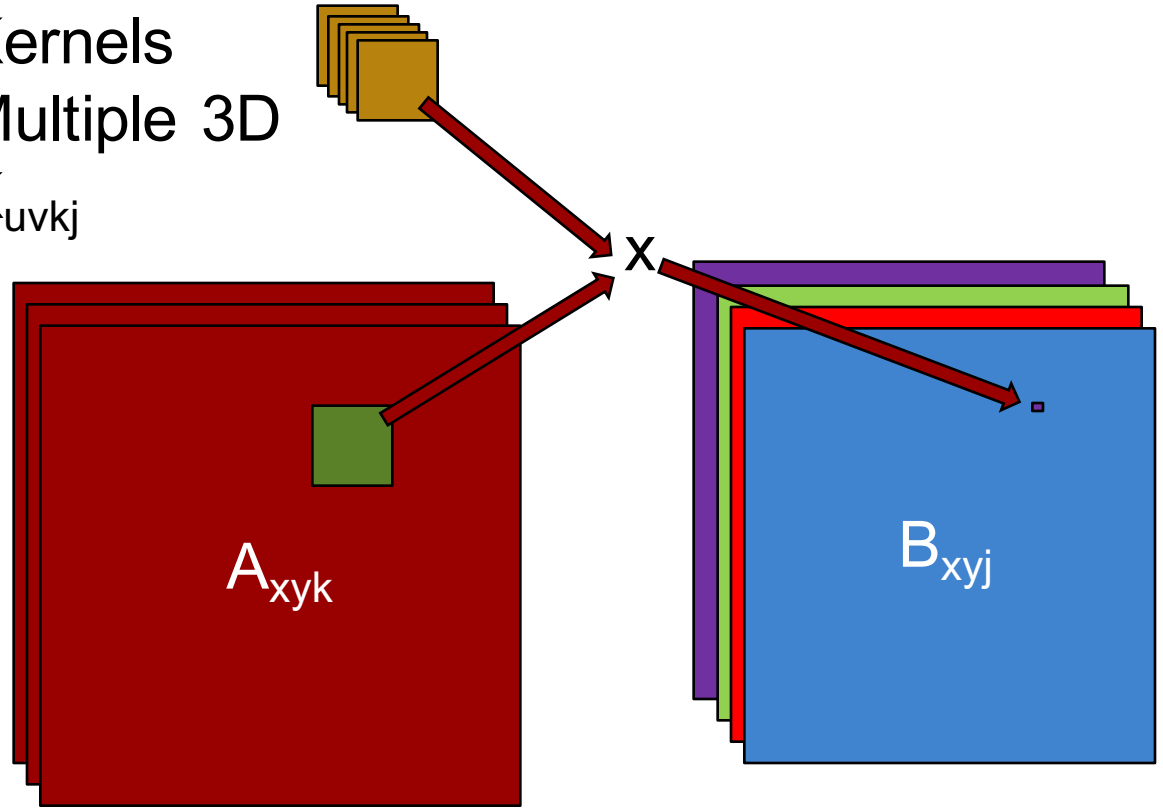
      For each kernel element u,v

        B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}

Kernels

Multiple 3D

K_{uvkj}



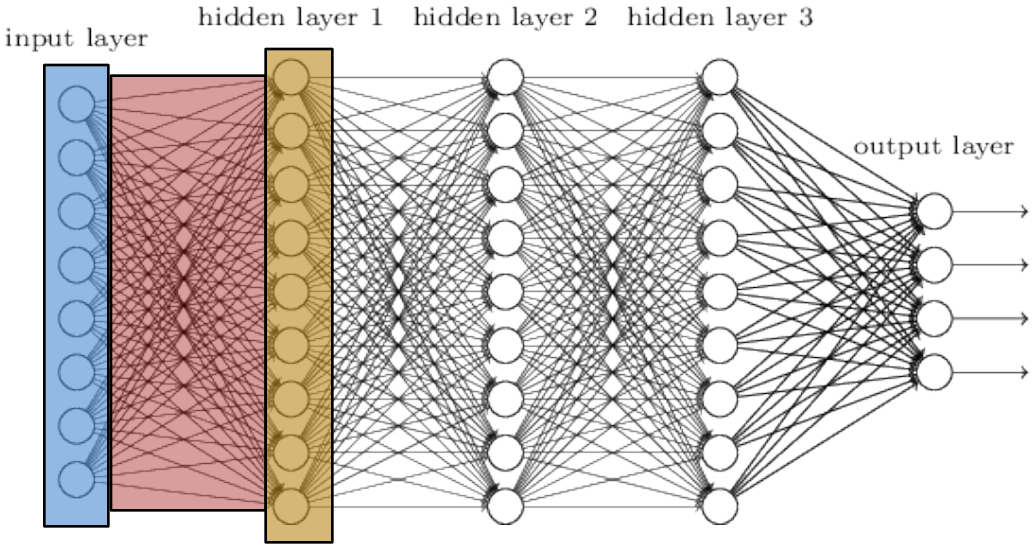
Input maps

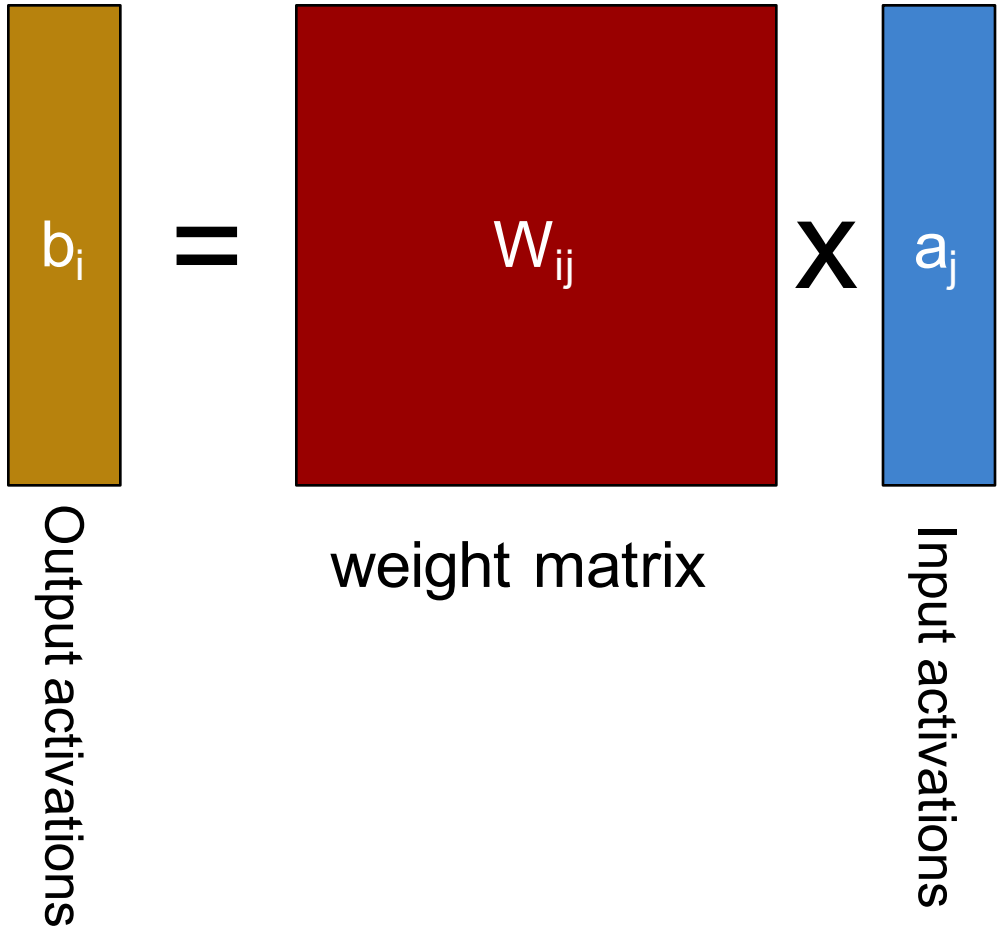
A_{xyk}

Output maps

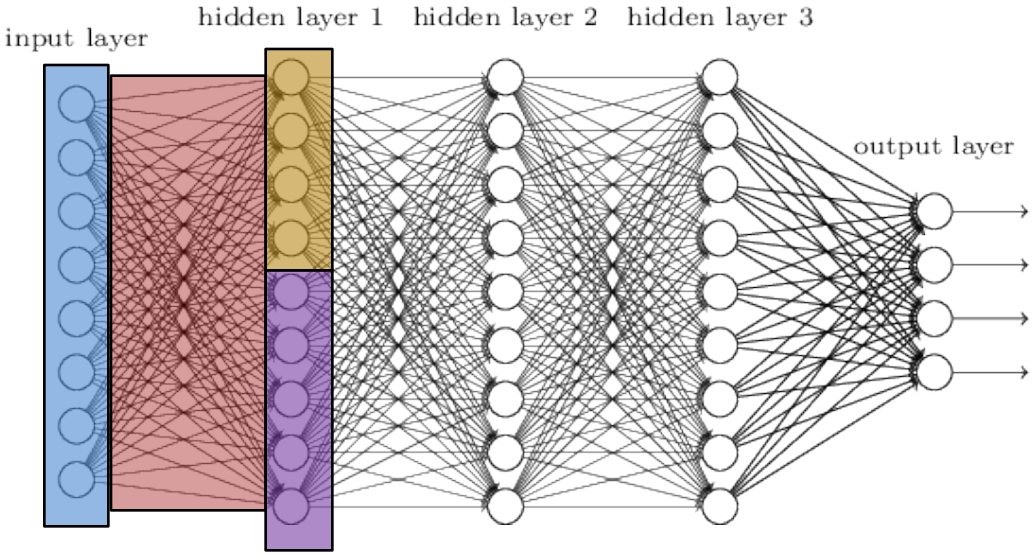
B_{xyj}

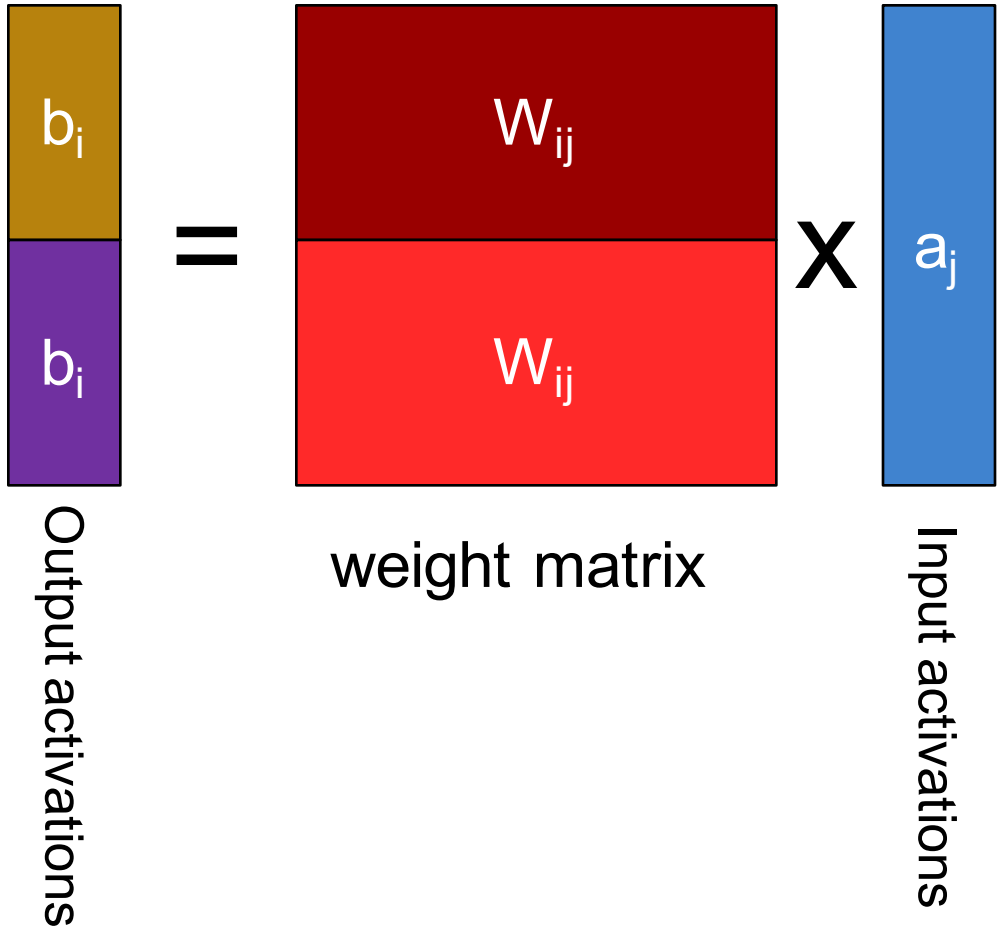
Model Parallel Fully-Connected Layer (M x V)





Model Parallel Fully-Connected Layer (M x V)

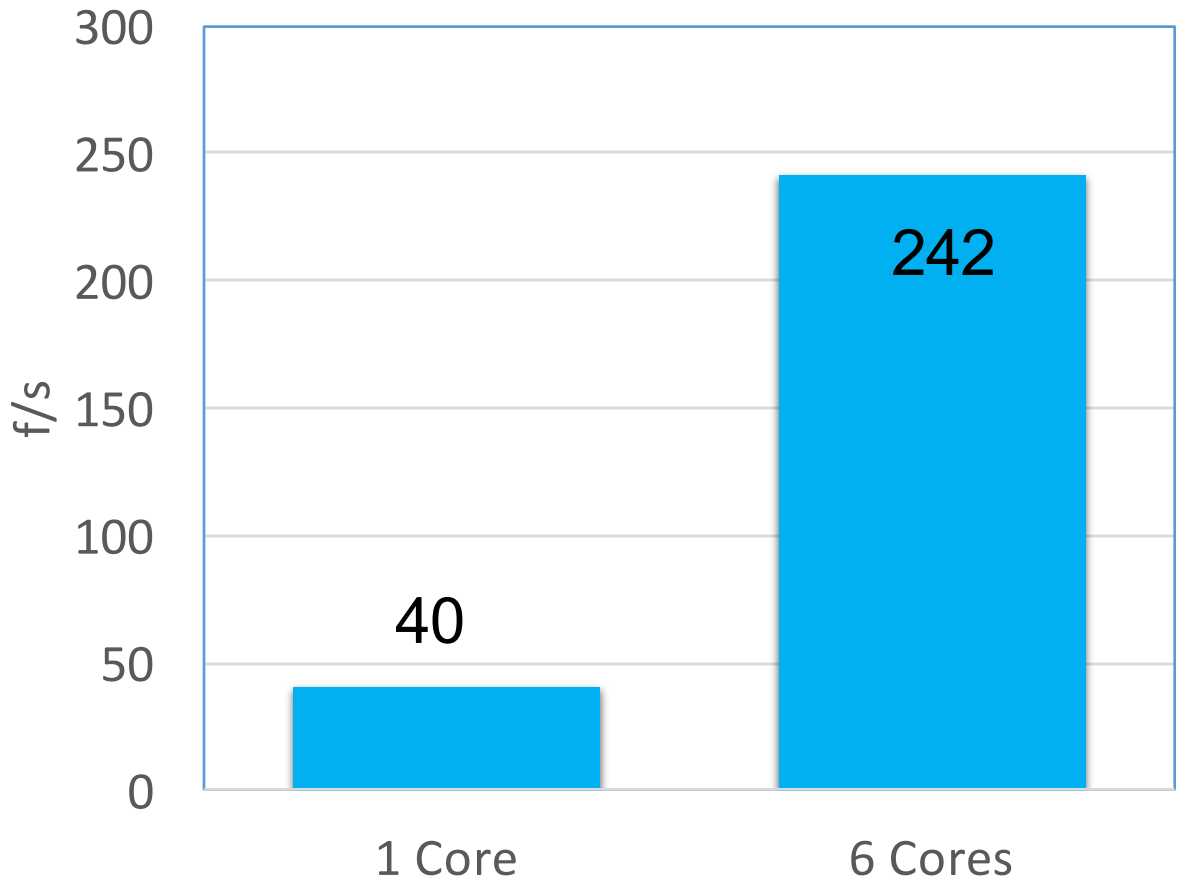




Hyper-Parameter Parallel

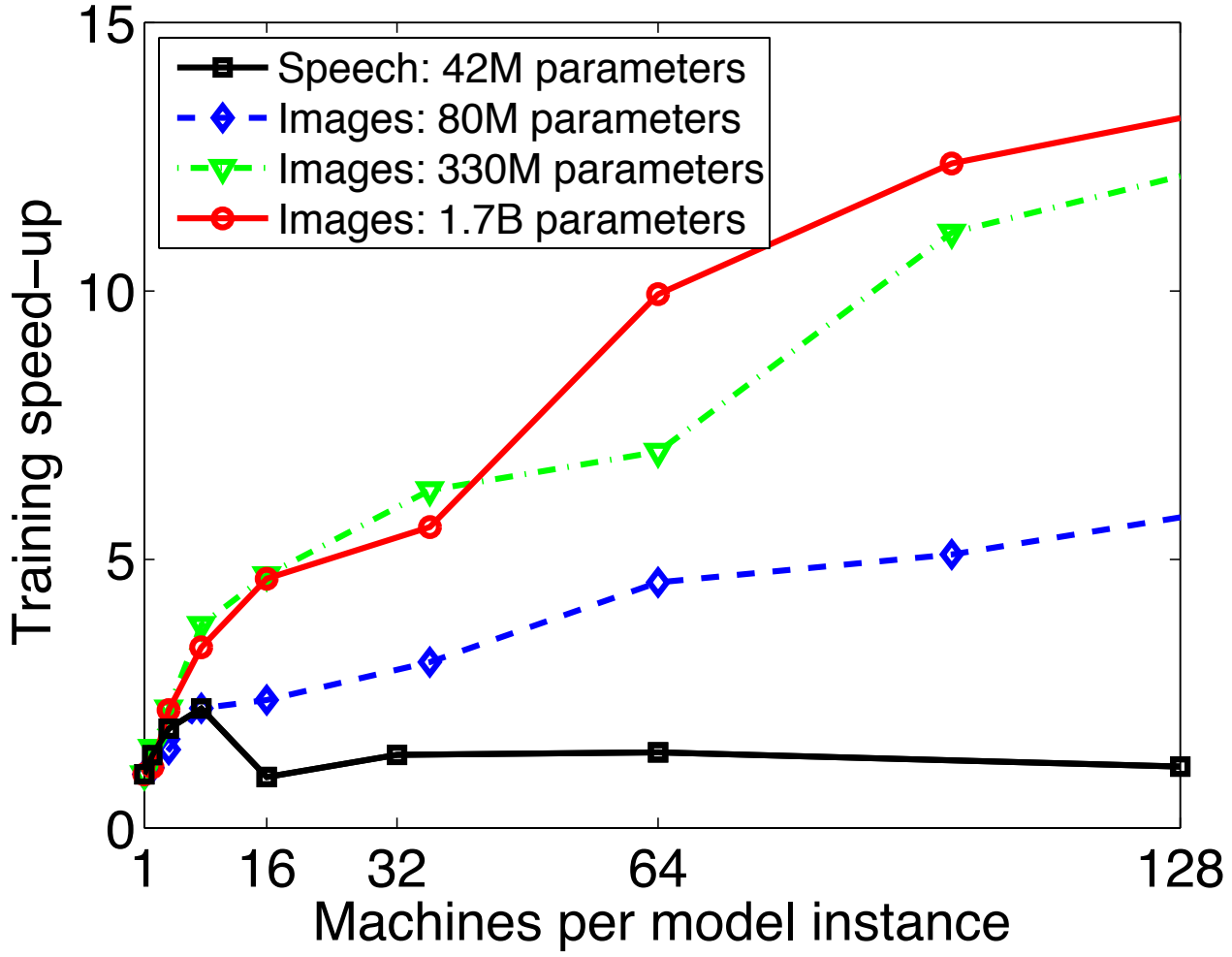
Try many alternative networks in parallel

CPU Parallelism – Core i7 – 1 core vs 6 cores



NVIDIA, “Whitepaper: GPU-based deep learning inference: A performance and power analysis.”

Data and Model Parallel Performance



Dean et al. Large Scale Distributed Deep Networks, NIPS 2012

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

Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

To go fast, use multiple processors

To go fast, use multiple processors

To be efficient and fast, use GPUs

To go fast, use multiple processors

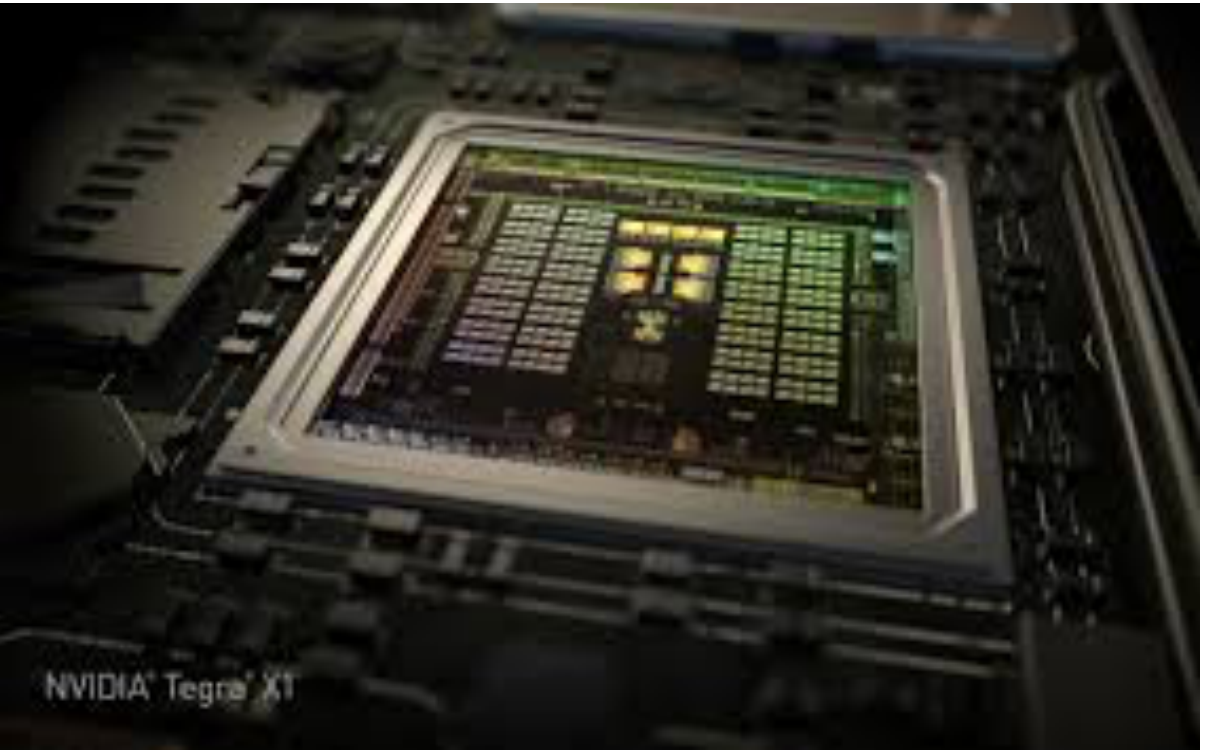
To be efficient and fast, use GPUs

To be efficient and go really fast, use multiple GPUs

Titan X

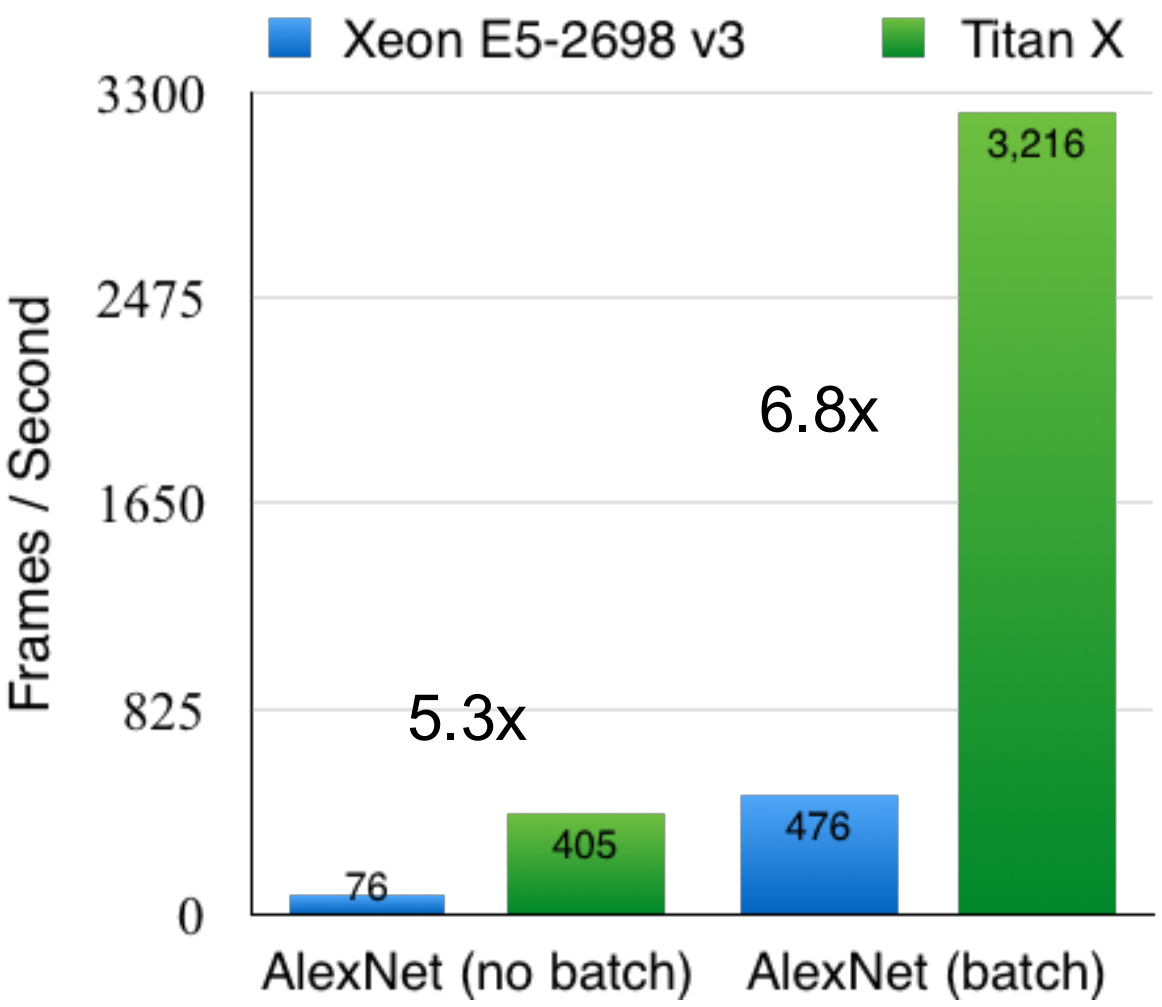
- 3072 CUDA cores @ 1 GHz
- 6 Teraflops FP32
- 12GB of GDDR5 @ 336 GB/sec
- 250W TDP
- 24GFLOPS/W
- 28nm process

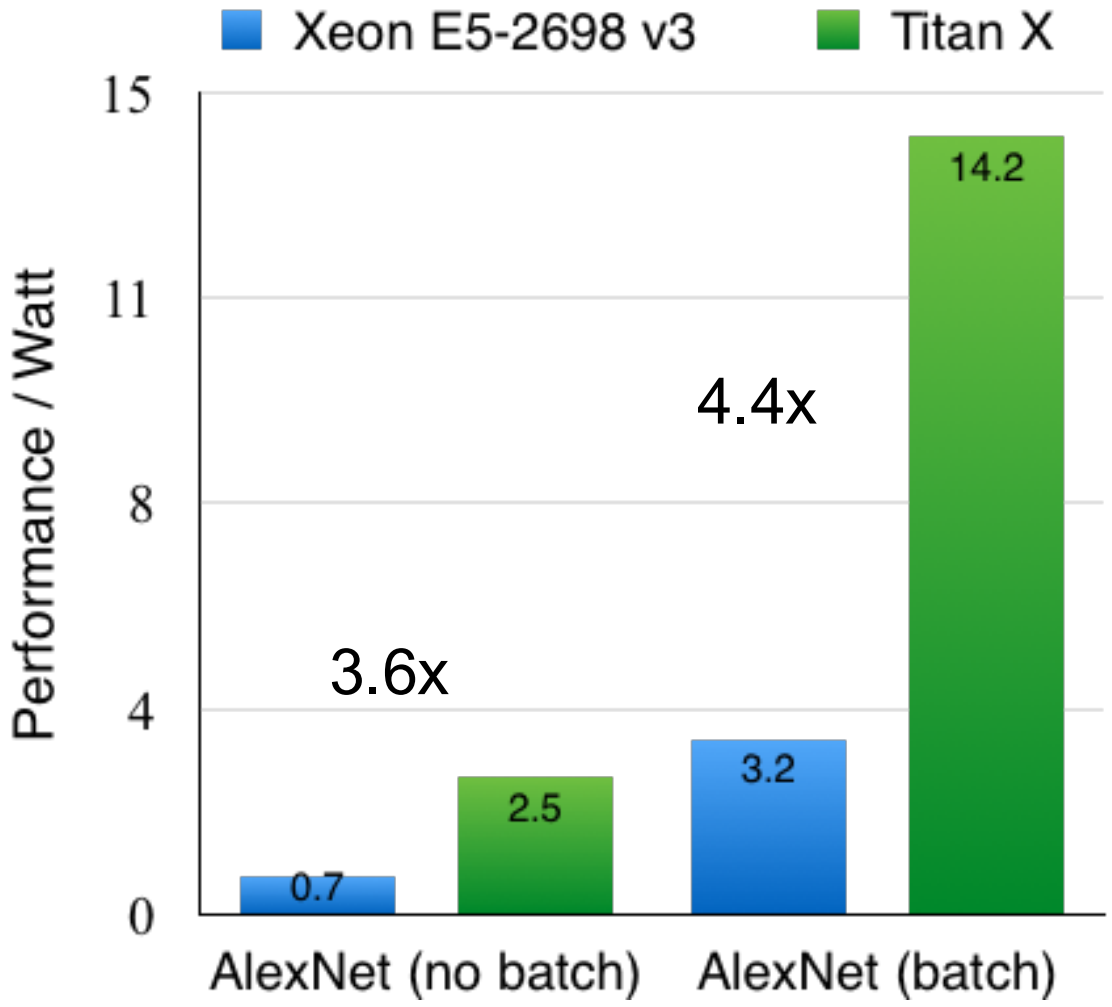
Tegra X1



- 256 CUDA cores @ ~1 GHz
- 1 Teraflop FP16
- 4GB of LPDDR4 @ 25.6 GB/s
- 15 W TDP (1W idle, <10W typical)
- 100GFLOPS/W (FP16)
- 20nm process

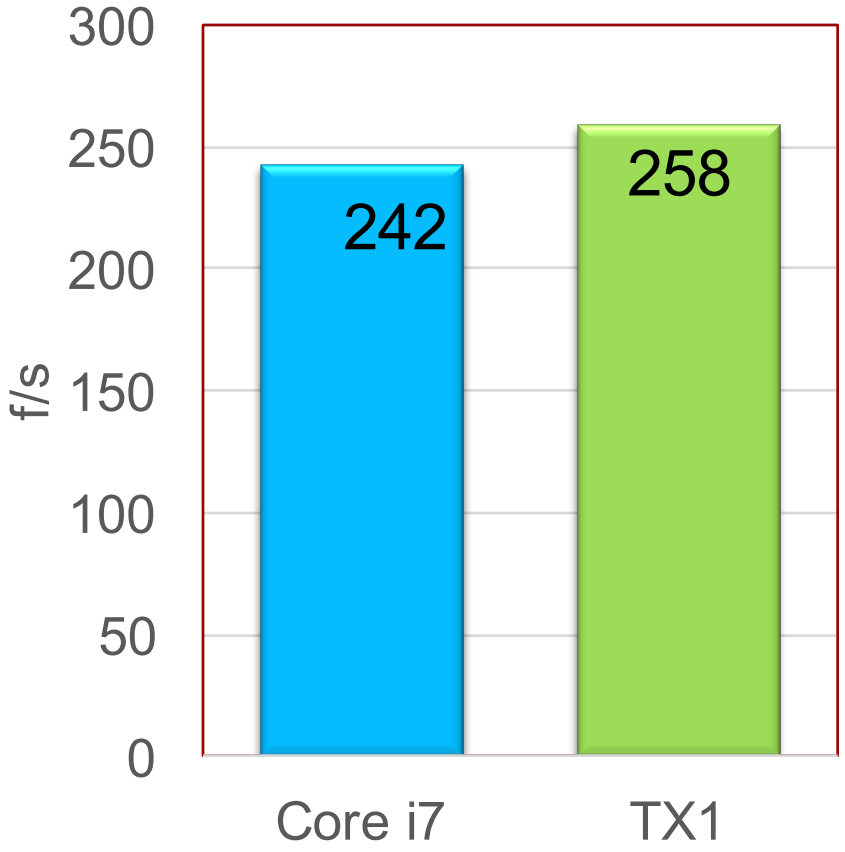
Xeon E5-2698 CPU v.s. TitanX GPU

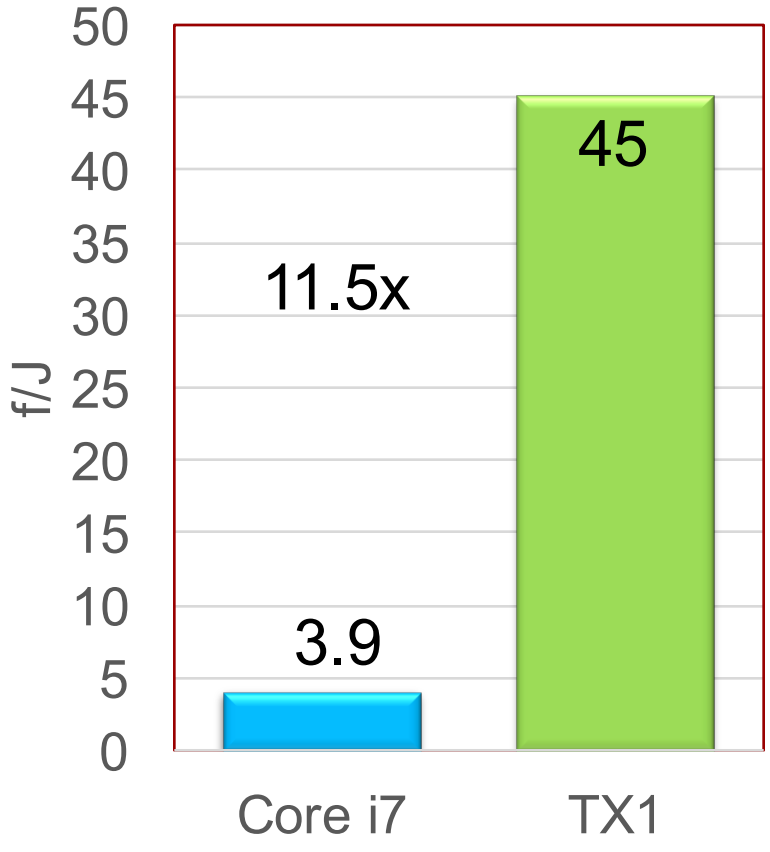




NVIDIA, “Whitepaper: GPU-based deep learning inference: A performance and power analysis.”

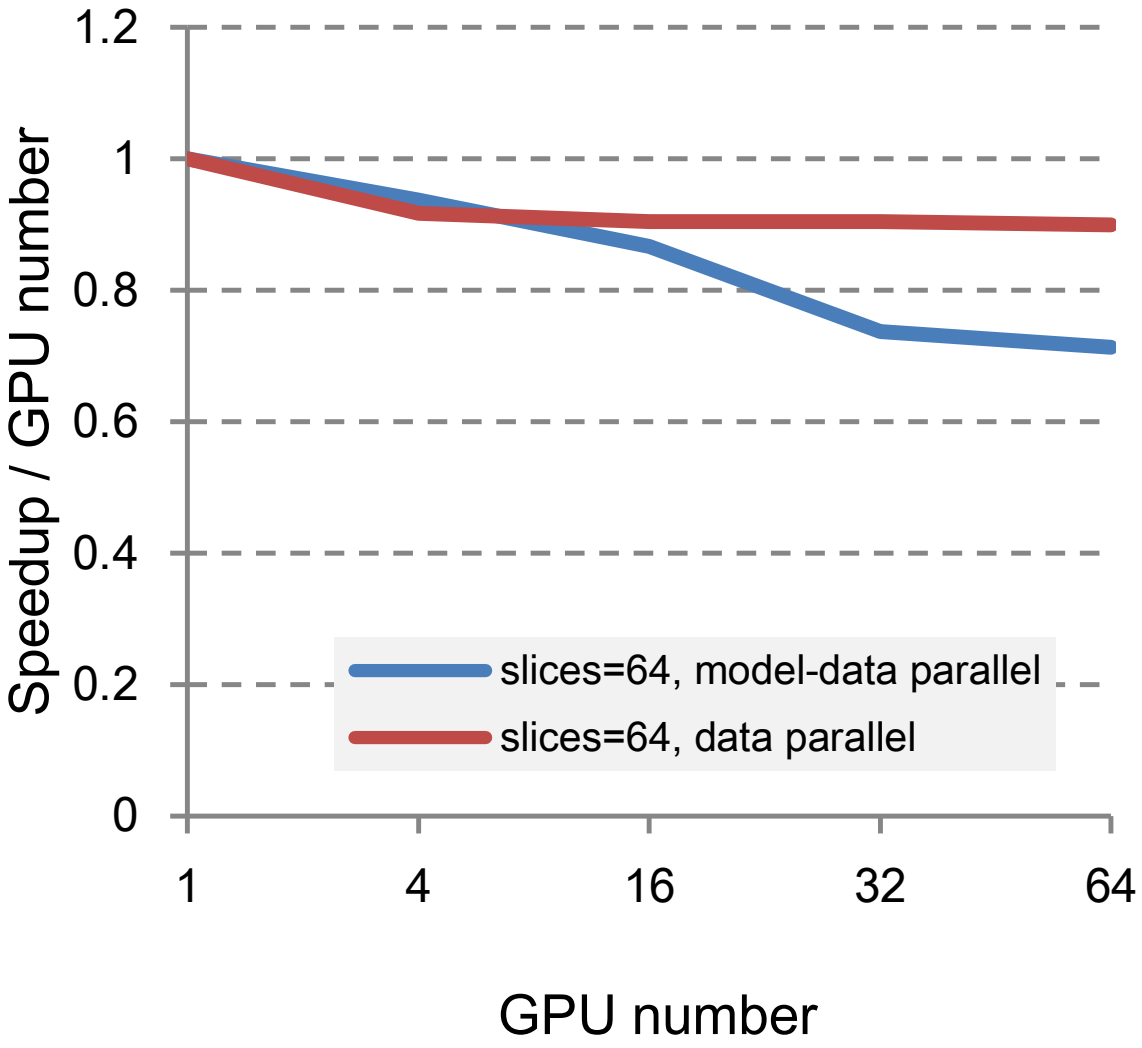
Tegra X1 vs Core i7

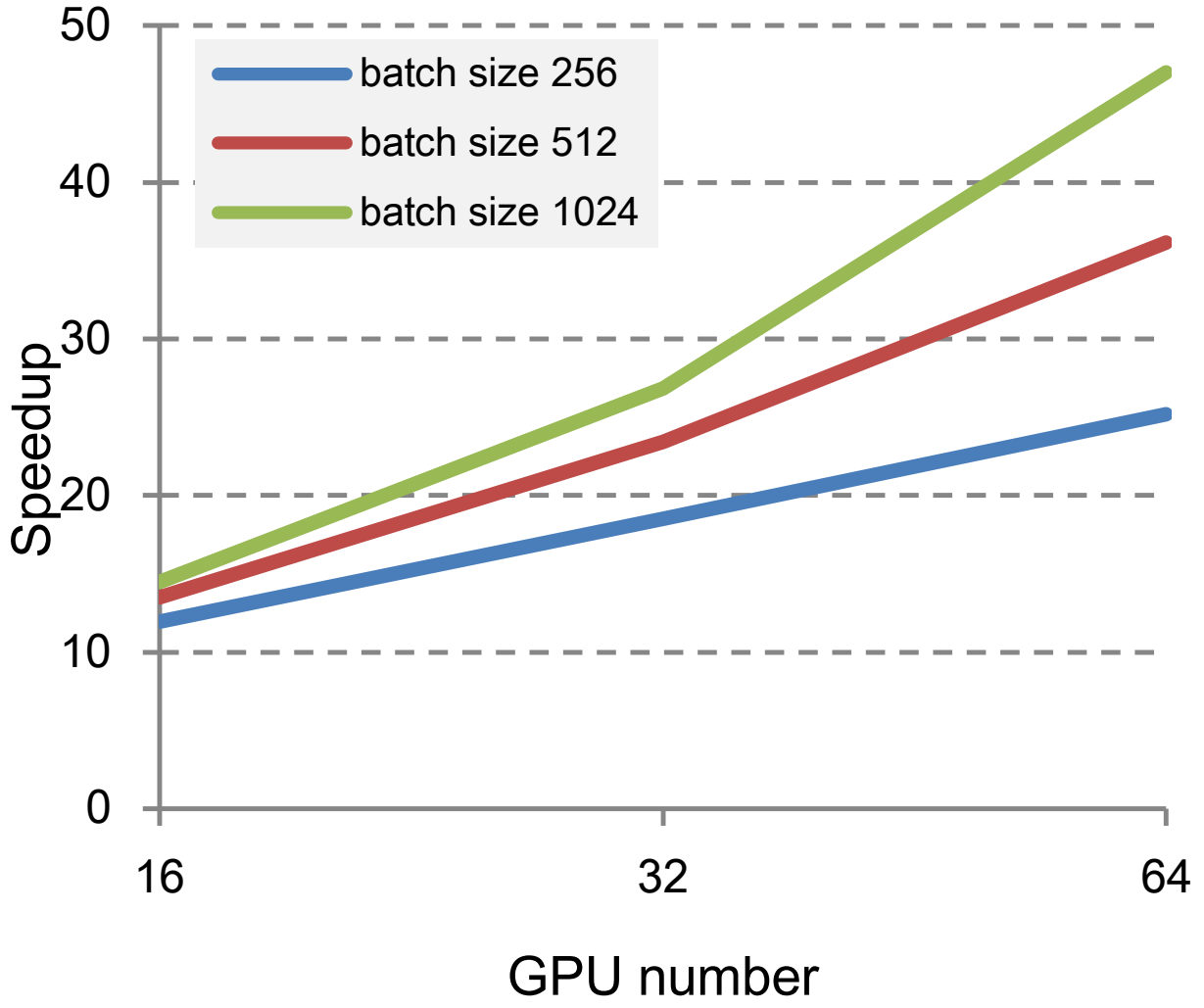




NVIDIA, “Whitepaper: GPU-based deep learning inference: A performance and power analysis.”

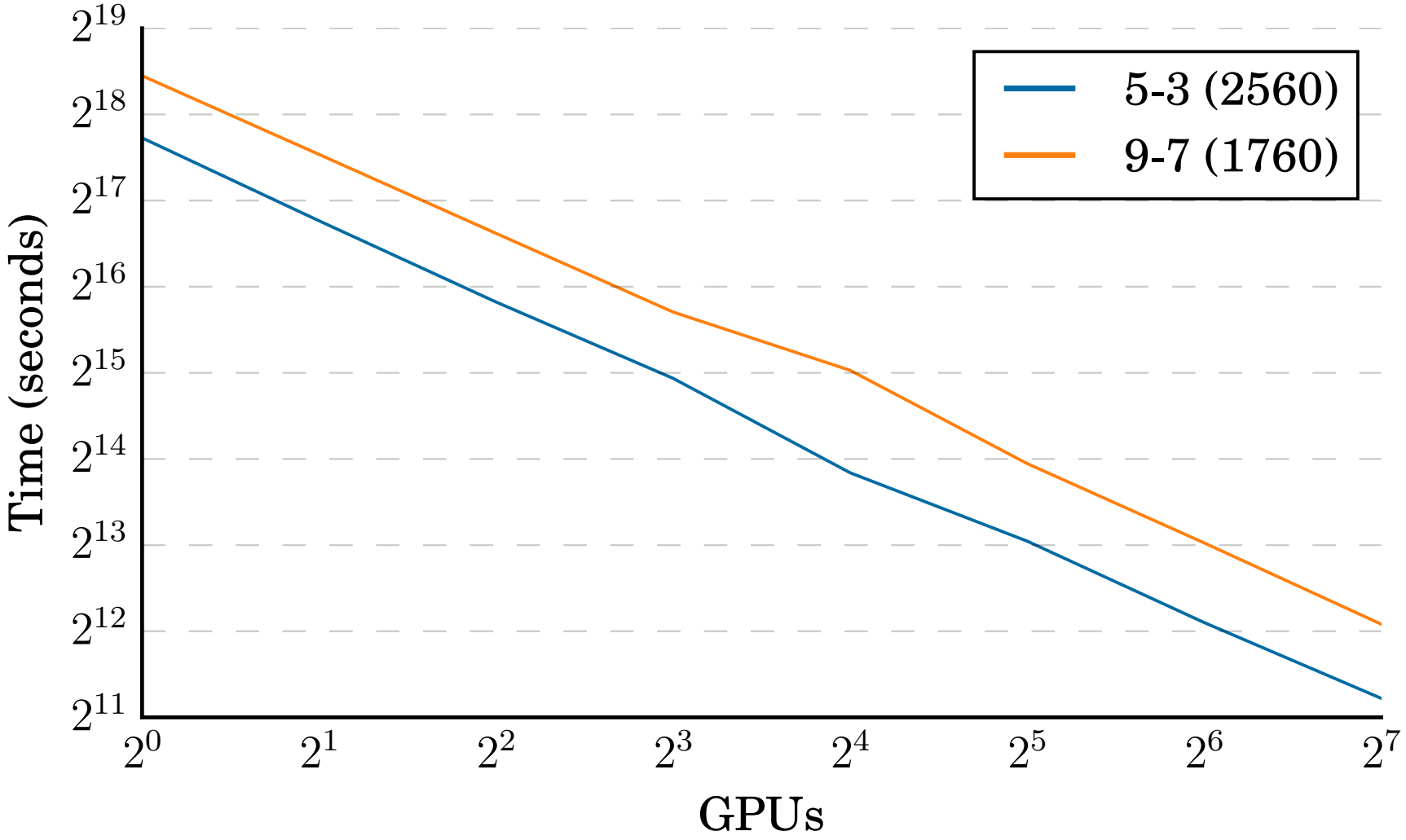
Parallel GPUs





Ren Wu et al., Deep Image: Scaling up Image Recognition, arXiv 2015

Parallel GPUs on Deep Speech 2



Baidu, Deep Speech 2: End-to-End Speech Recognition in English and Mandarin, 2015

Summary of GPUs

- Titan X ~6x faster, 4x more efficient than Xeon E5
- TX1 11.5x more efficient than Core i7
- On inference
- Larger gains on training

- Data parallelism scales easily to 16GPUs
- With some effort, linear speedup to 128GPUs

Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

Reducing precision

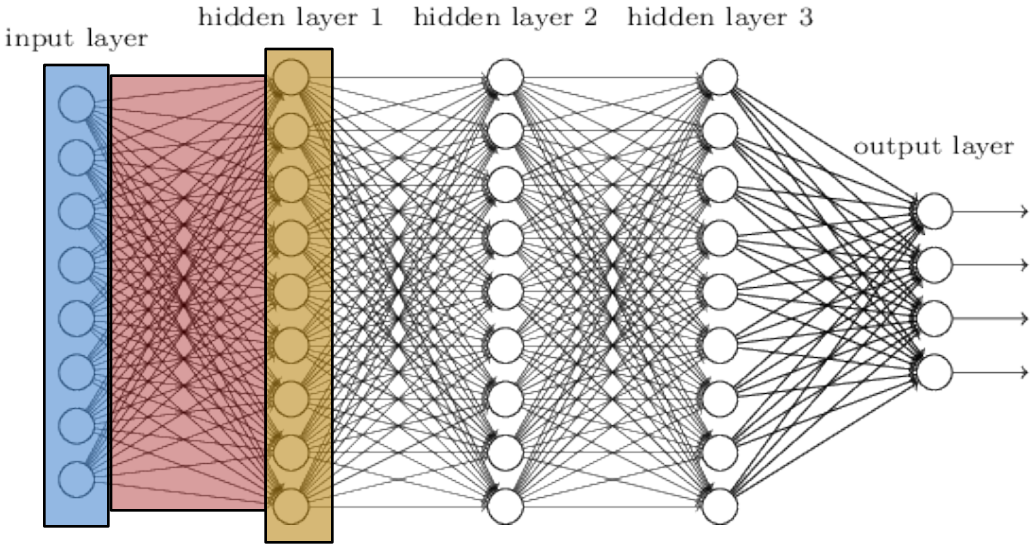
Reduces storage

Reduces energy

Improves performance

Has little effect on accuracy – to a point

DNN, key operation is dense M x V



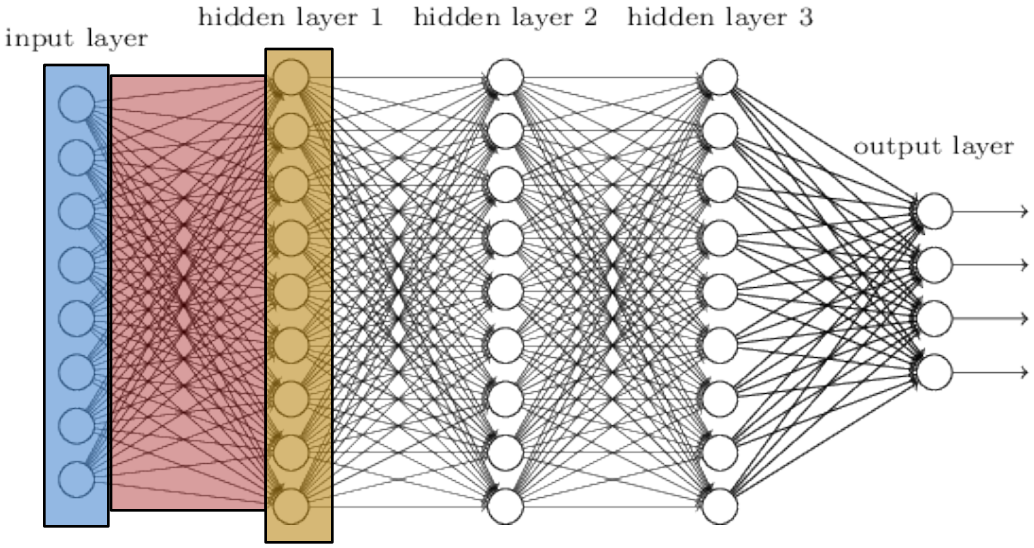
$b_i = W_{ij} \times a_j$

Output activations

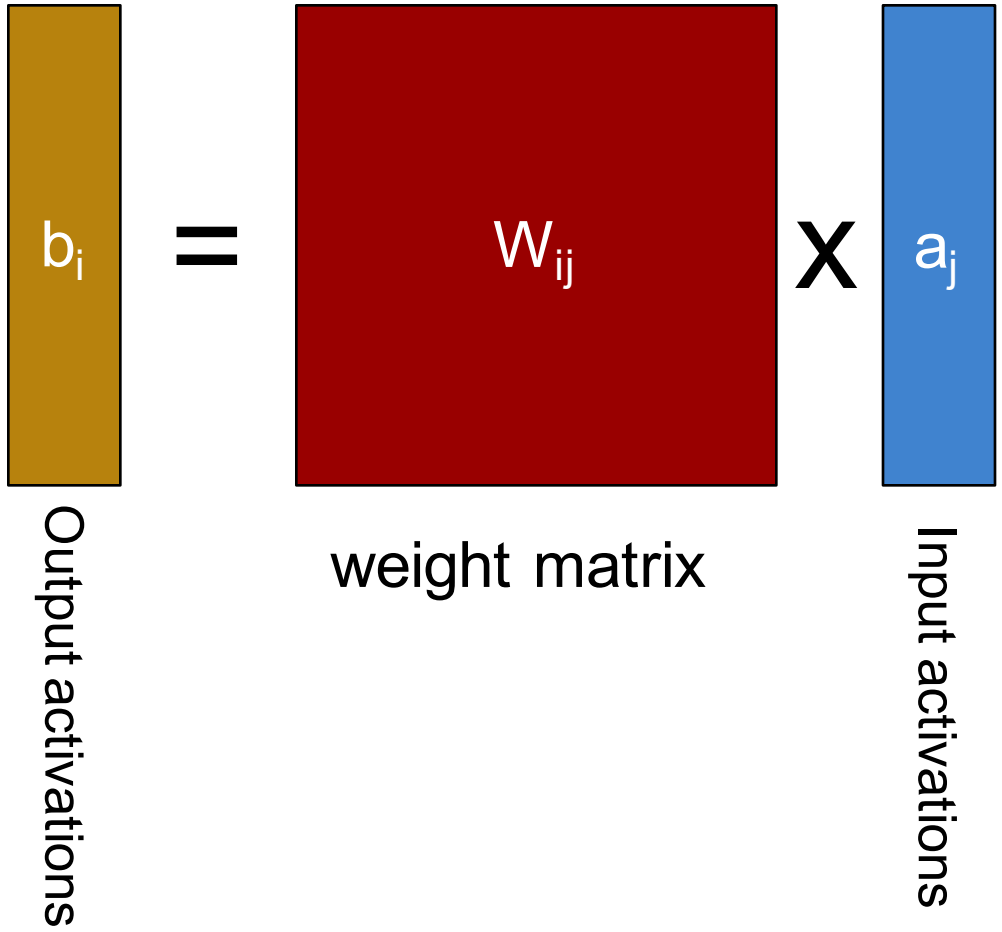
weight matrix

Input activations

DNN, key operation is dense M x V



$$b_i = f\left(\sum_j w_{ij}\, a_i\right)$$

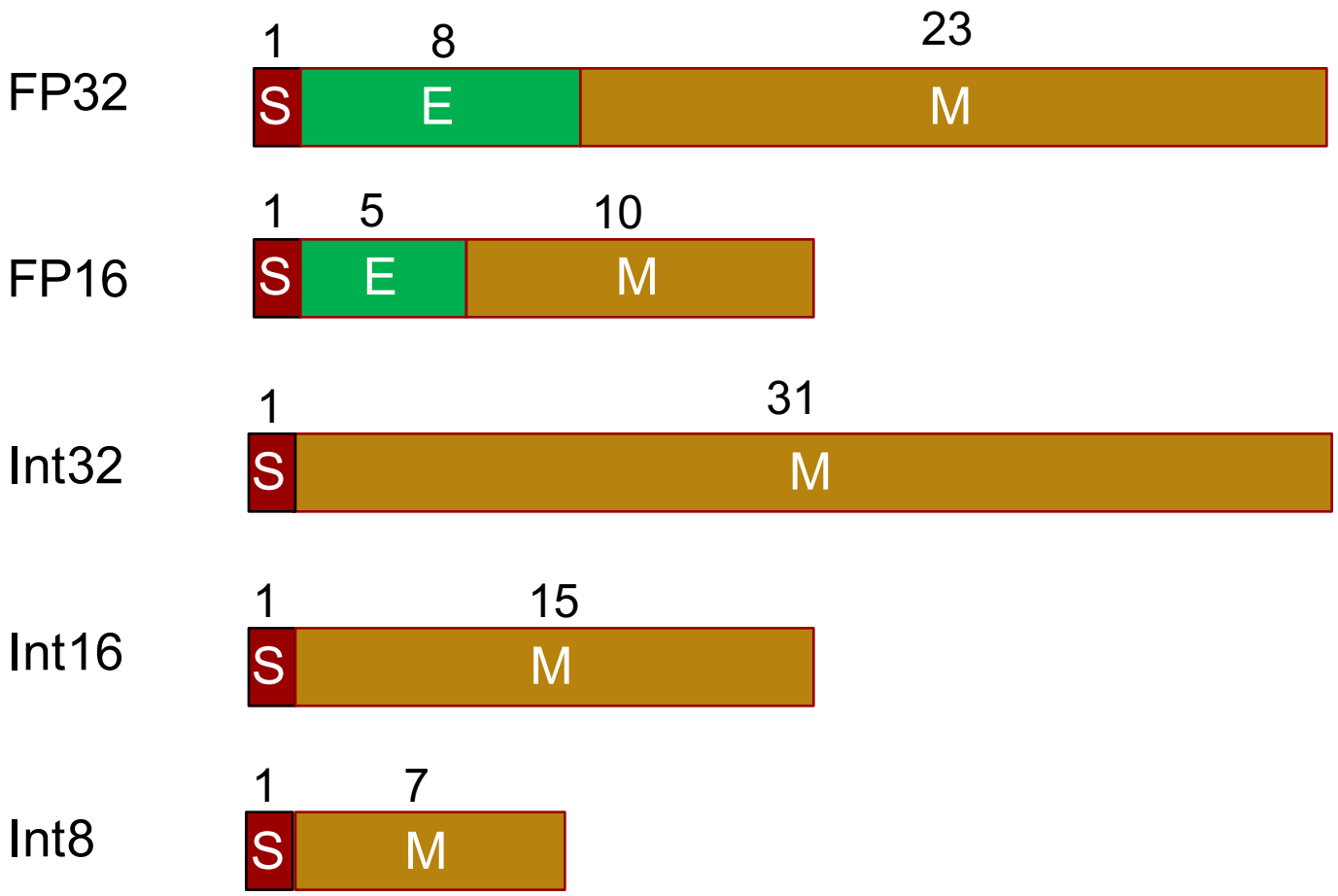


How much accuracy do we need in the computations:

$$b_i = f\left(\sum_j w_{ij}\, a_i\right)$$

$$w_{ij} = w_{ij} + \alpha a_i g_j$$

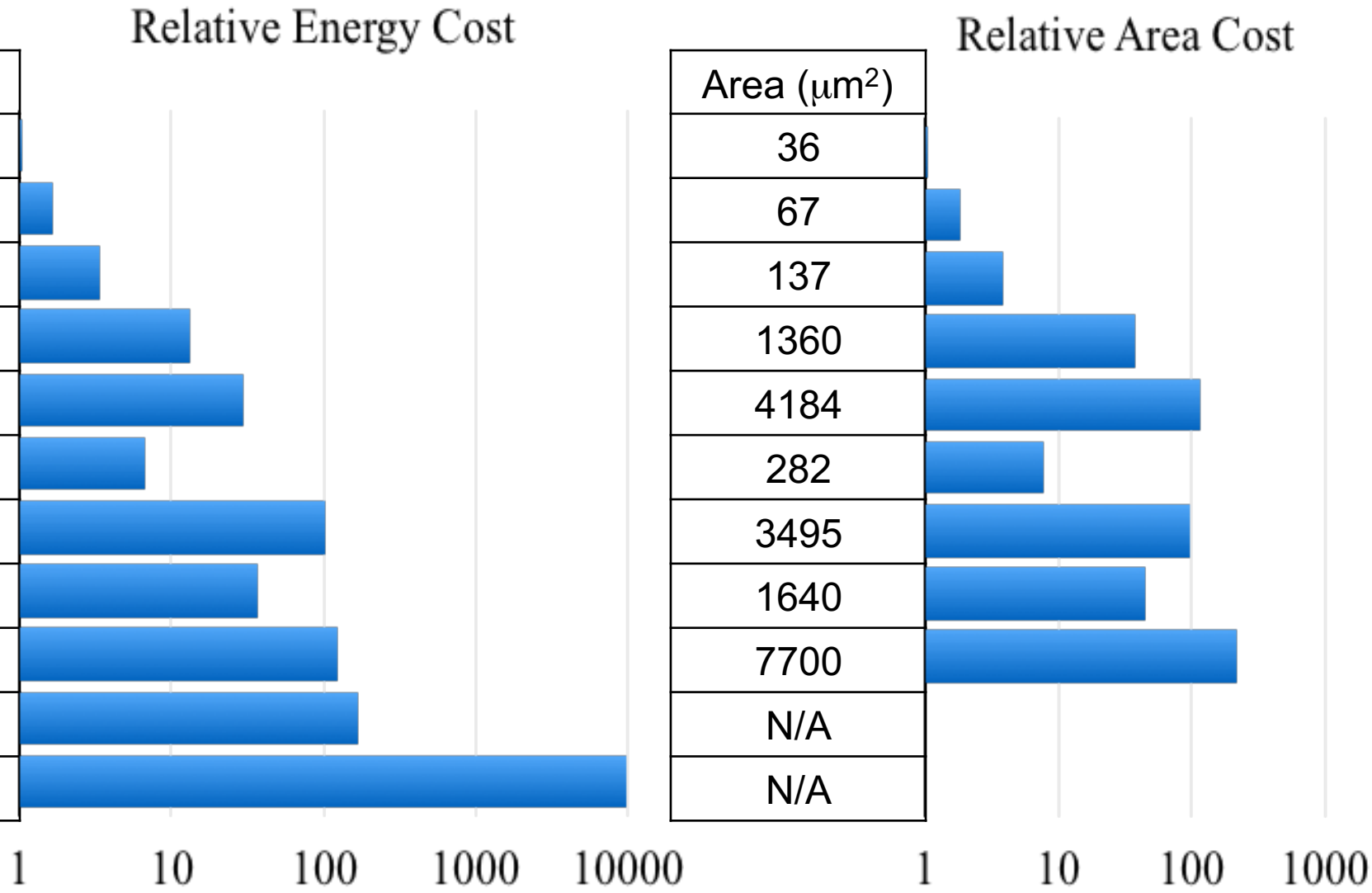
Number Representation



	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

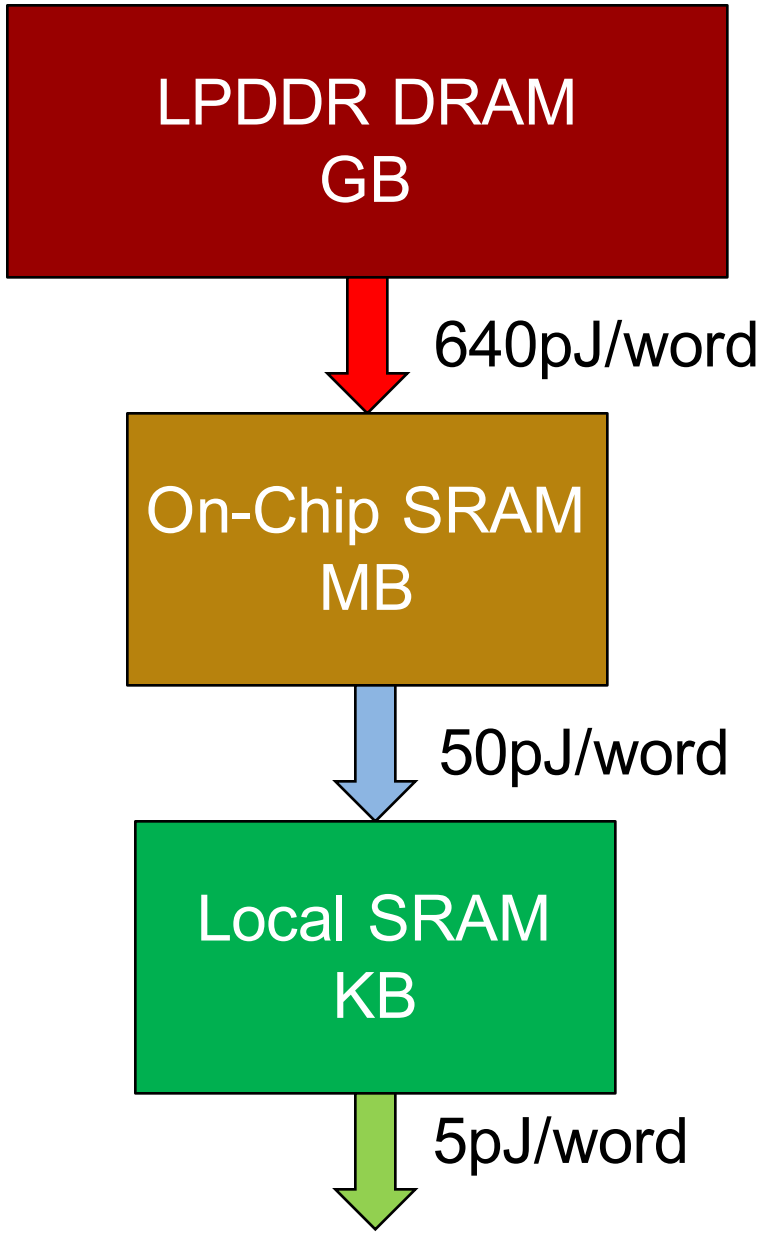
Cost of Operations

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

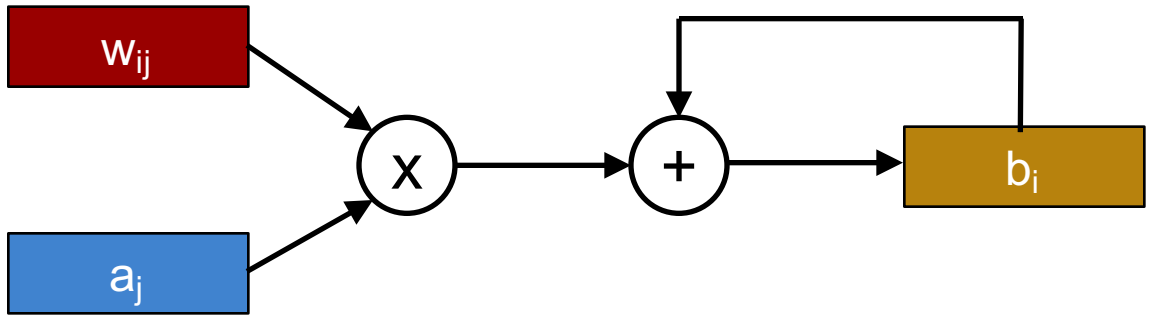


Energy numbers are from Mark Horowitz "Computing's Energy Problem (and what we can do about it)", ISSCC 2014
Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.

The Importance of Staying Local



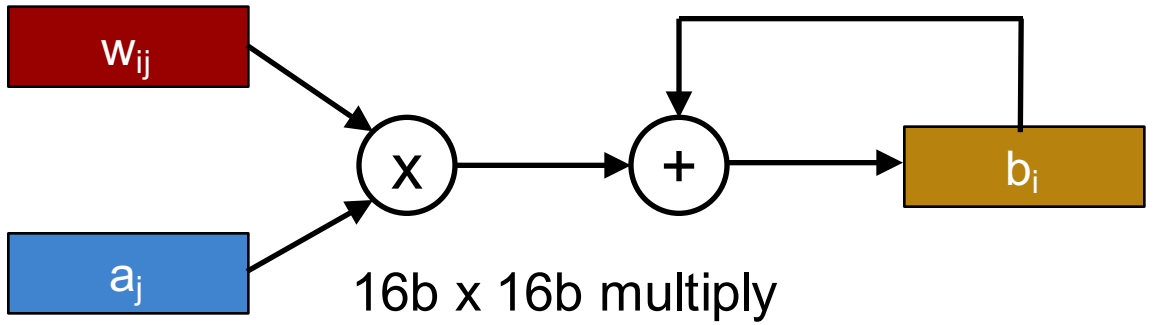
Mixed Precision



Mixed Precision

Store weights as 4b using Trained quantization, decode to 16b

Store activations as 16b



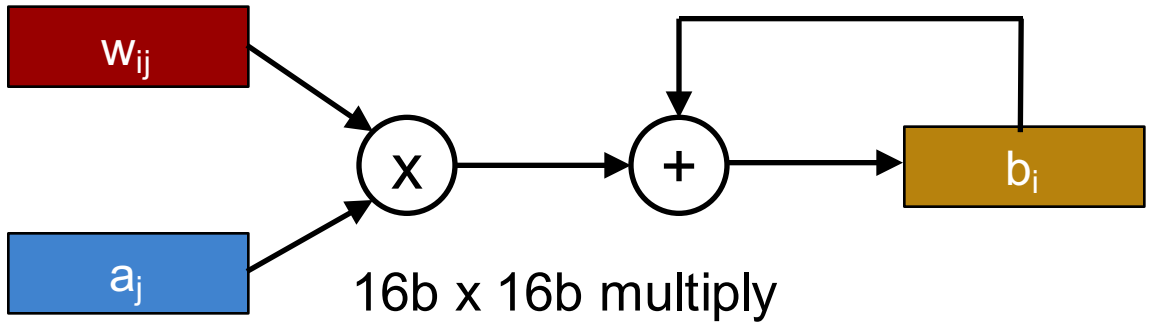
16b x 16b multiply
round result to 16b

accumulate 24b or 32b to avoid saturation

Mixed Precision

Store weights as 4b using Trained quantization, decode to 16b

Store activations as 16b

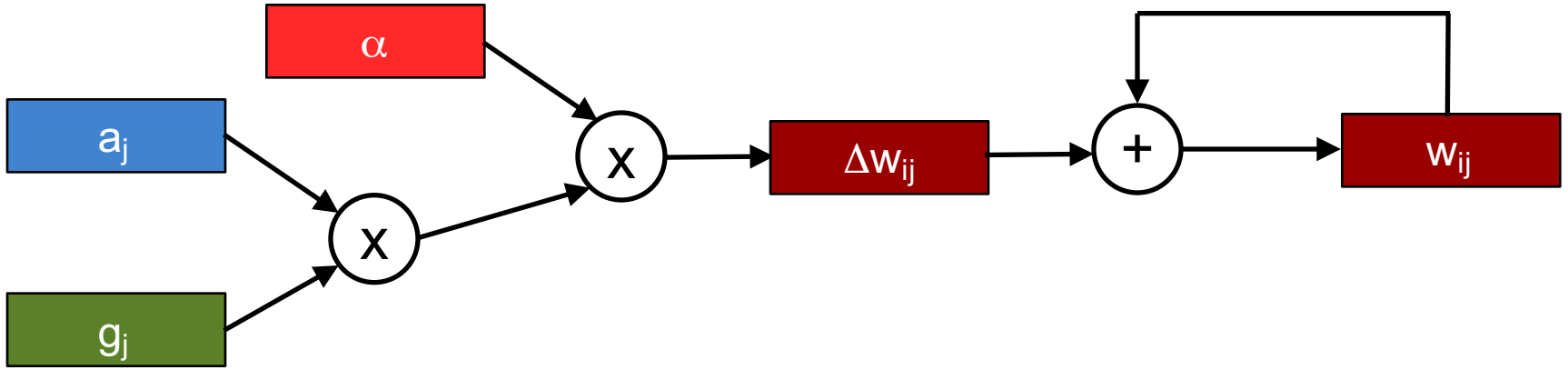


16b x 16b multiply
round result to 16b

accumulate 24b or 32b to avoid saturation

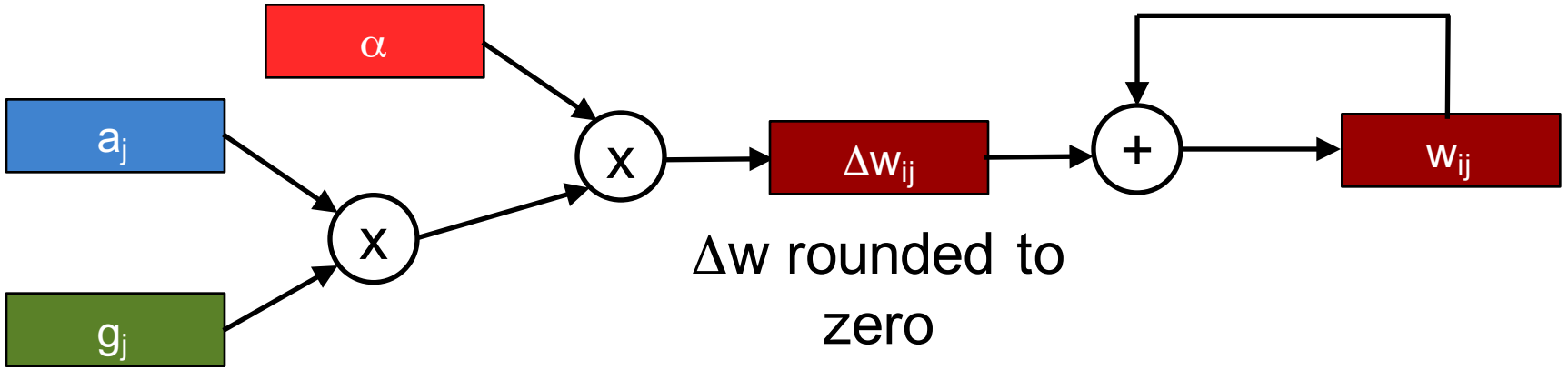
Batch normalization important to ‘center’ dynamic range

Weight Update



Weight Update

Learning rate may be very small (10^{-5} or less)

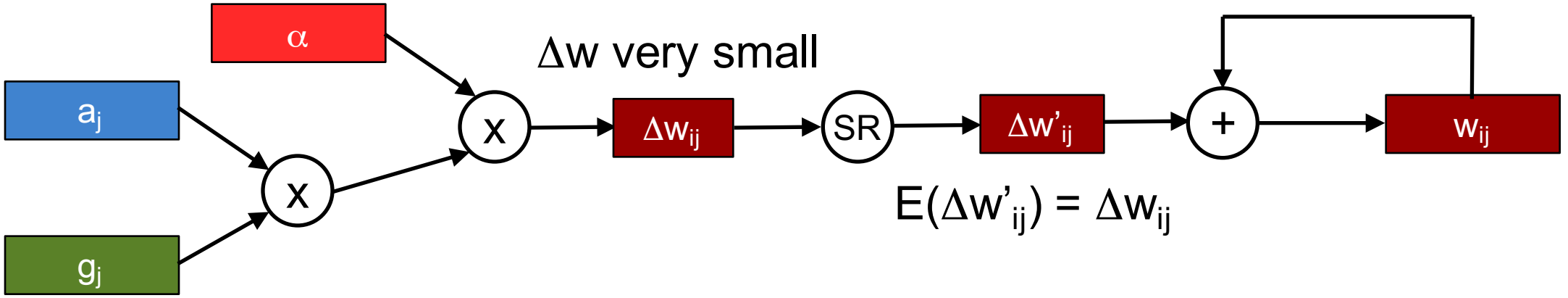


Δw rounded to zero

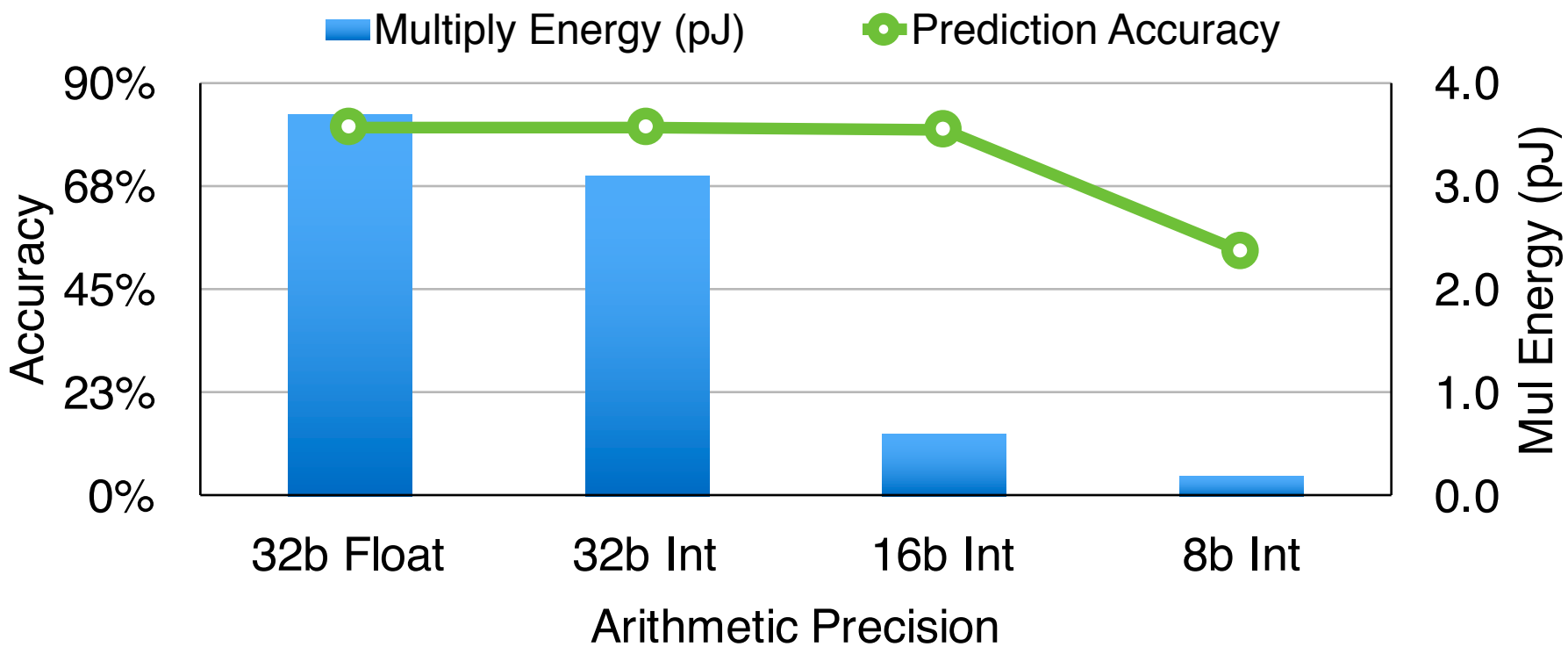
No learning!

Stochastic Rounding

Learning rate may be very small (10^{-5} or less)



Reduced Precision for Inference



Reduced Precision For Training

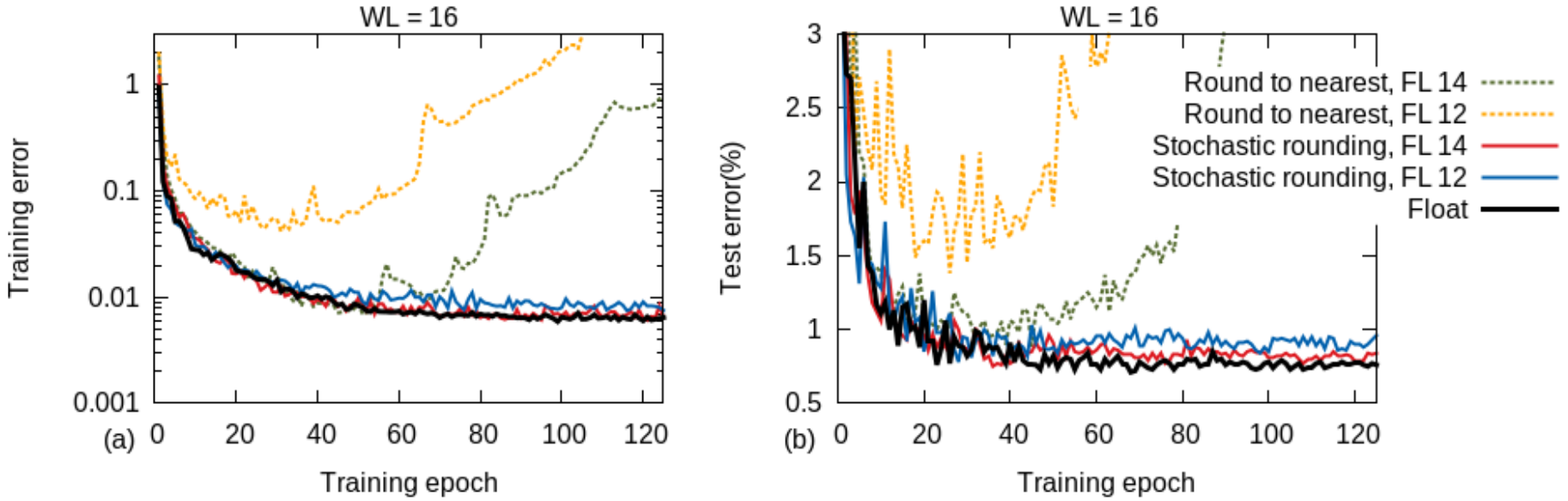


Figure 2. MNIST dataset using CNNs: Training error (a) and the test error (b) for training using fixed-point number representation and rounding mode set to either "Round to nearest" or "Stochastic rounding". The word length for fixed-point numbers `WL` is kept fixed at 16 bits and results are shown for different fractional (integer) lengths for weights and weight updates: 12(4), and 14(2) bits. Layer outputs use $\langle$`6, 10`$\rangle$ format in all cases. Results using `float` are also shown for comparison.

S. Gupta et.al "Deep Learning with Limited Numerical Precision" ICML 15

Summary of Reduced Precision

- Can save memory capacity, memory bandwidth, memory power, and arithmetic power by using smaller numbers
- FP16 works with little effort
  - 2x gain in memory, 4x in multiply power
- With care, one can use
  - 8b for convolutions
  - 4b for fully-connected layers
- Batch normalization – important to ‘center’ ranges
- Stochastic rounding – important to retain small increments

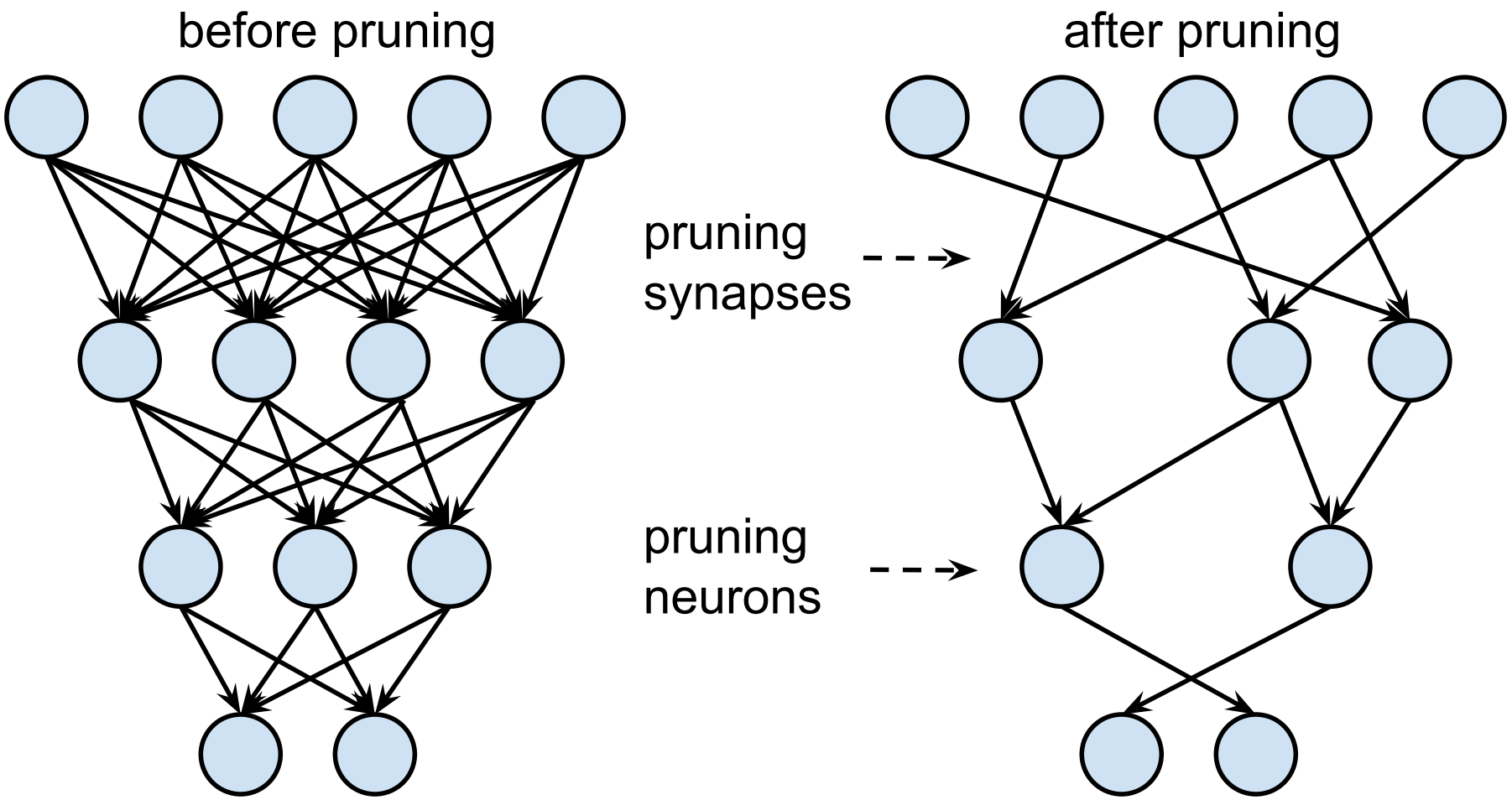
Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

Reducing Size of Network Reduces Work and Storage

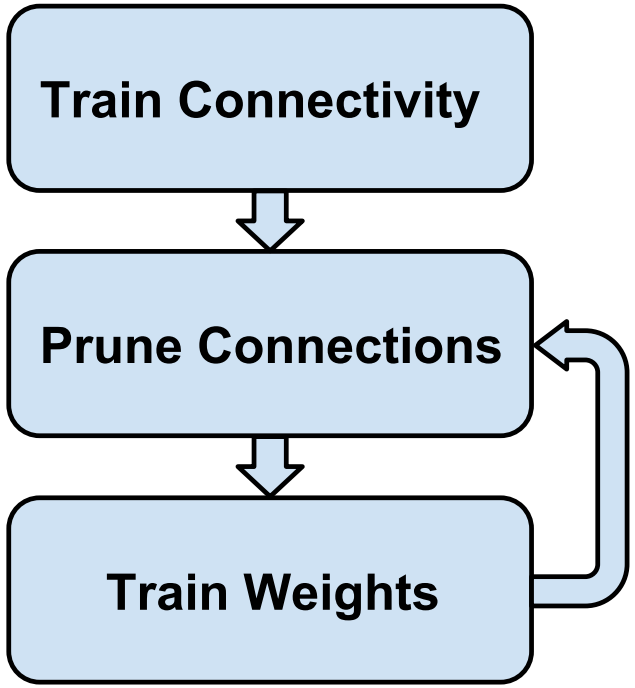
Prune Unneeded Connections

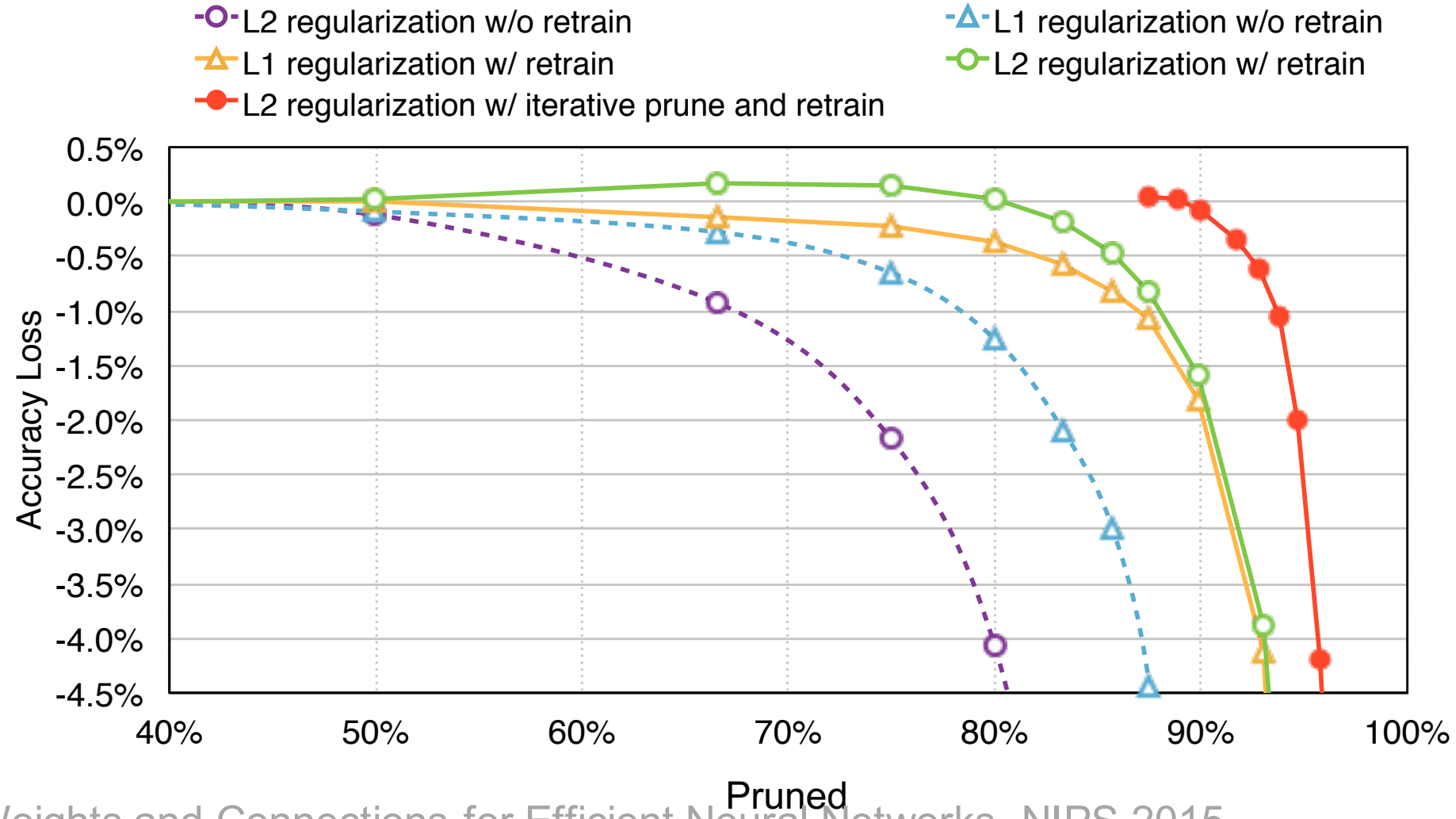
Pruning



Han et al. Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015

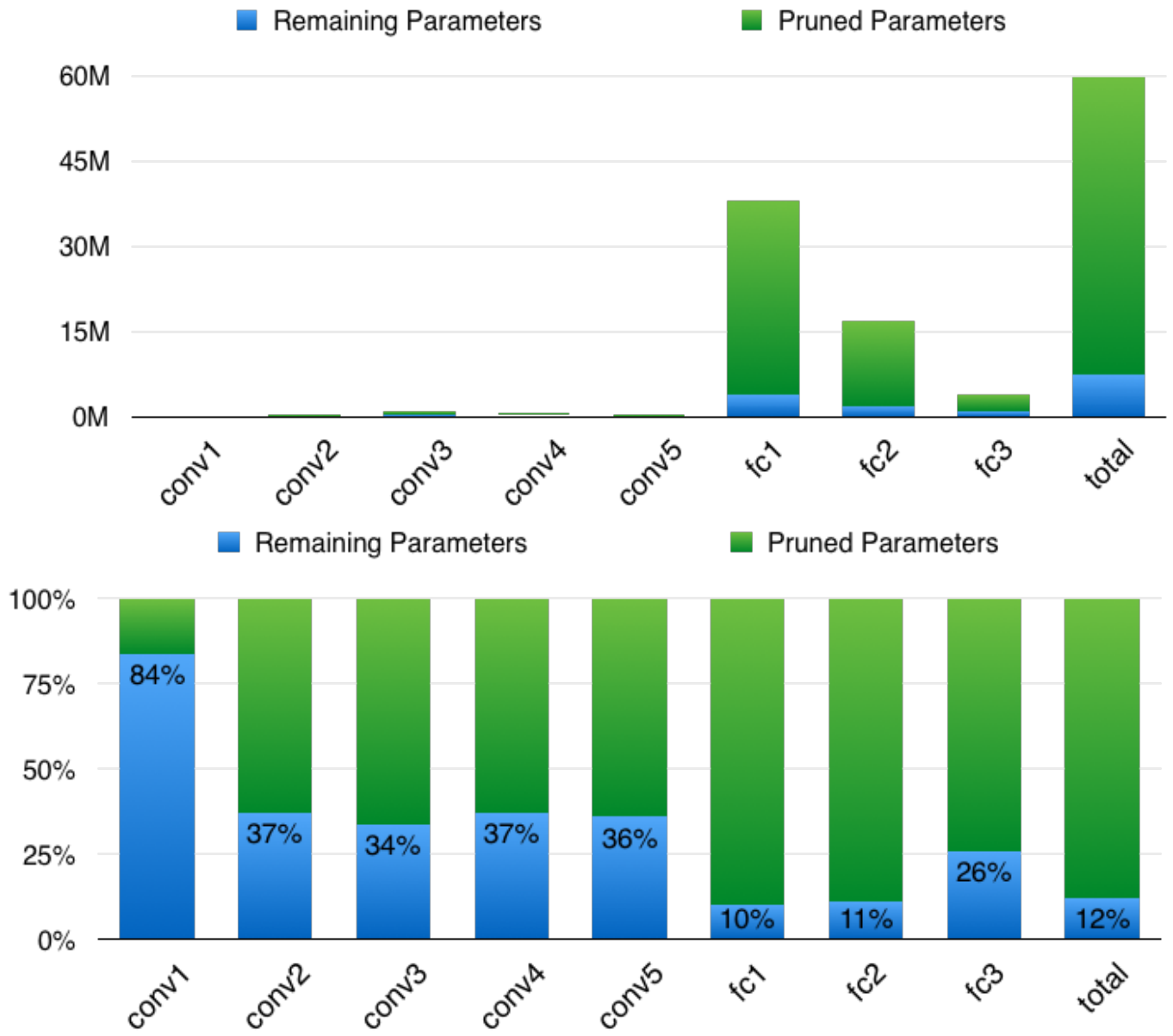
Retrain to Recover Accuracy



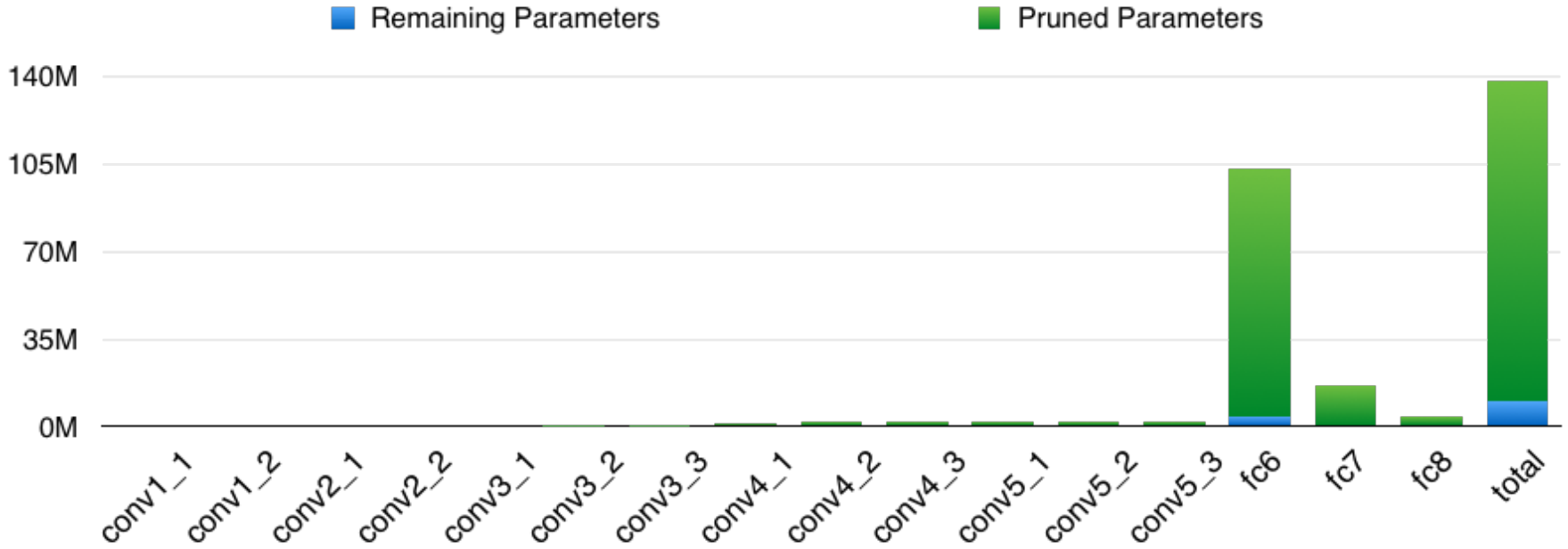


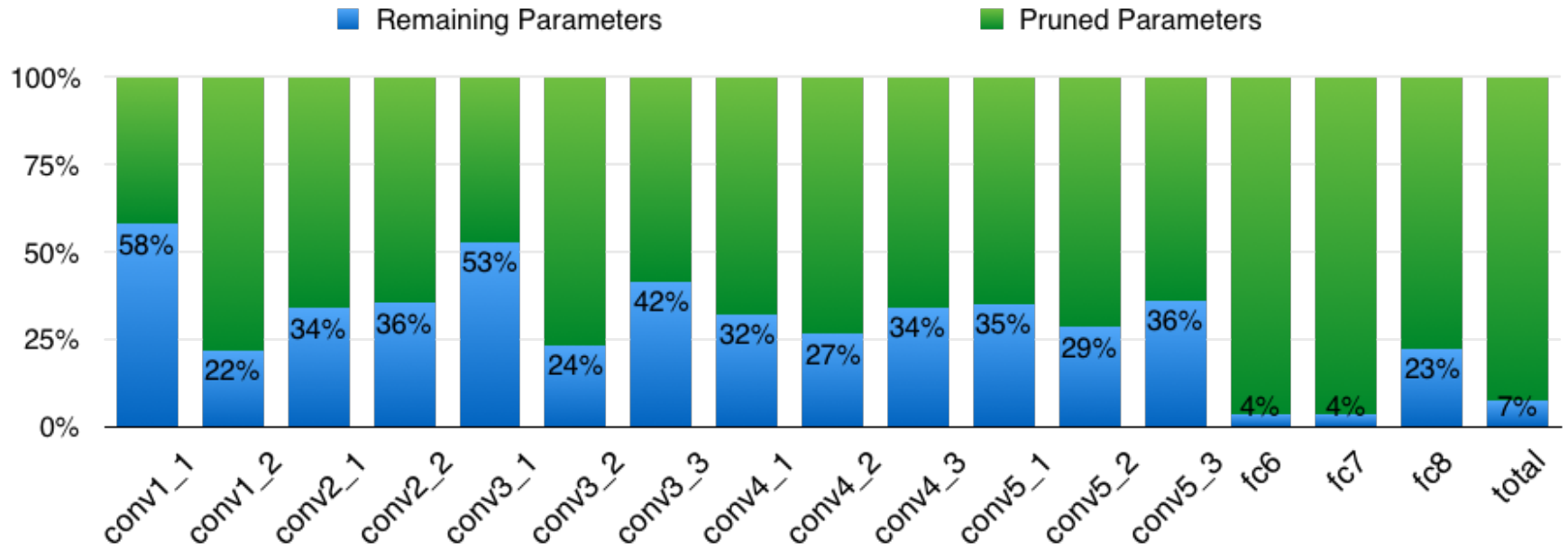
Han et al. Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015

Pruning of AlexNet

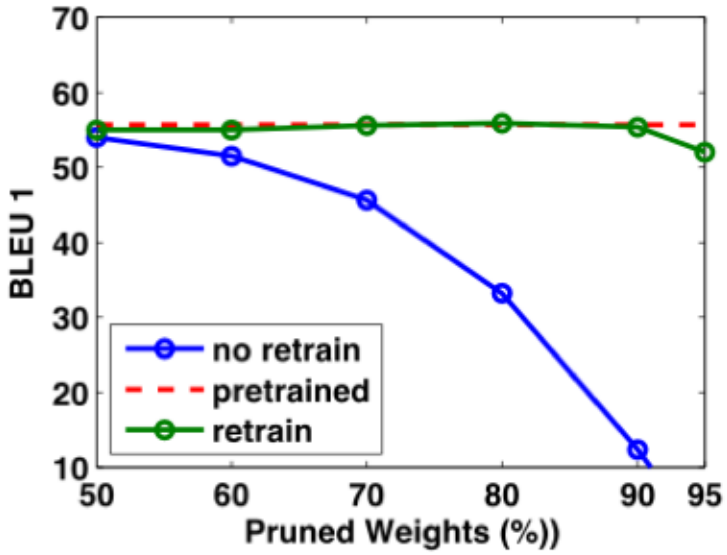


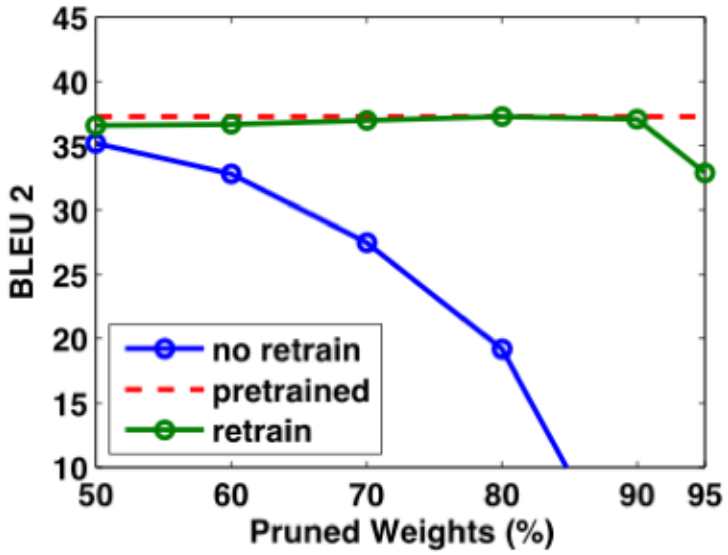
Pruning of VGG-16

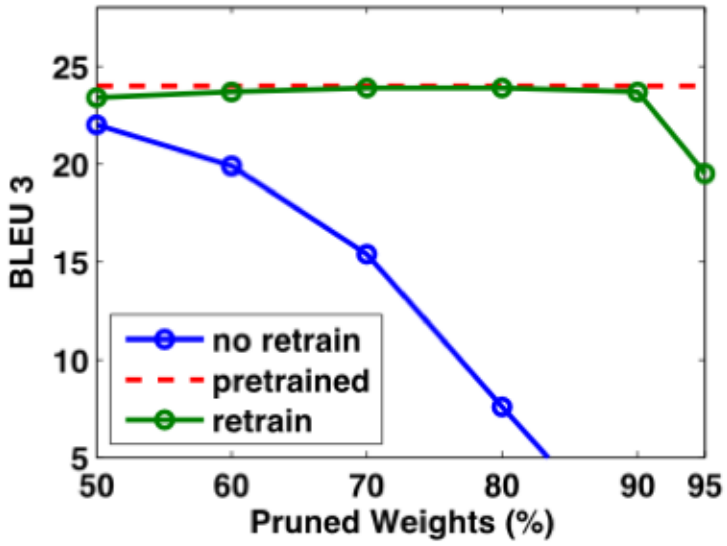


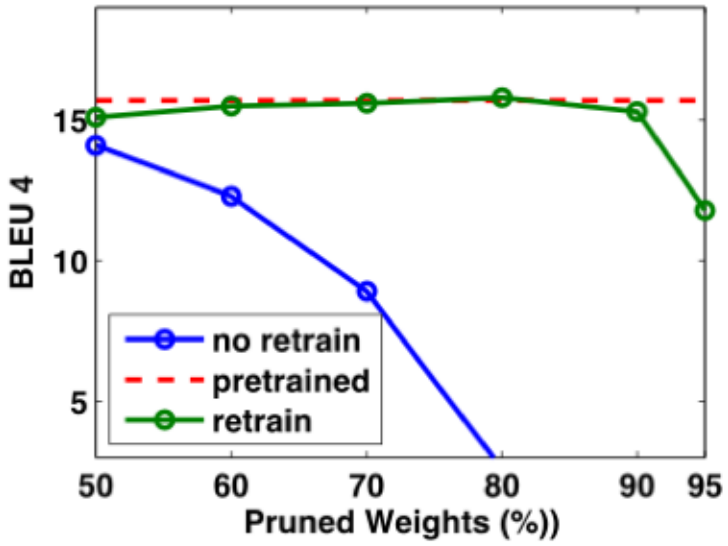


Pruning Neural Talk and LSTM









Pruning Neural Talk and LSTM

- **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

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

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

- **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

Speedup of Pruning on CPU/GPU

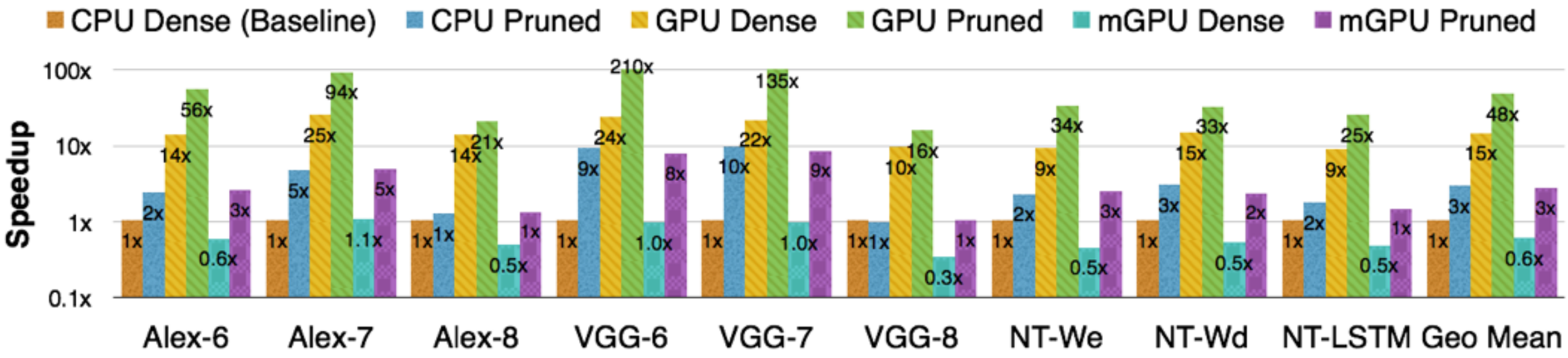


Figure 9: Compared with the original network, pruned network layer achieved 3× speedup on CPU, 3.5× on GPU and 4.2× on mobile GPU on average. Batch size = 1 targeting real time processing. Performance number normalized to CPU.

Intel Core i7 5930K: MKL CBLAS GEMV, MKL SPBLAS CSRMV
NVIDIA GeForce GTX Titan X: cuBLAS GEMV, cuSPARSE CSRMV
NVIDIA Tegra K1: cuBLAS GEMV, cuSPARSE CSRMV

History of Pruning

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990.

Babak Hassibi, David G Stork, et al. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, pages 164–164, 1993.

See Poster:

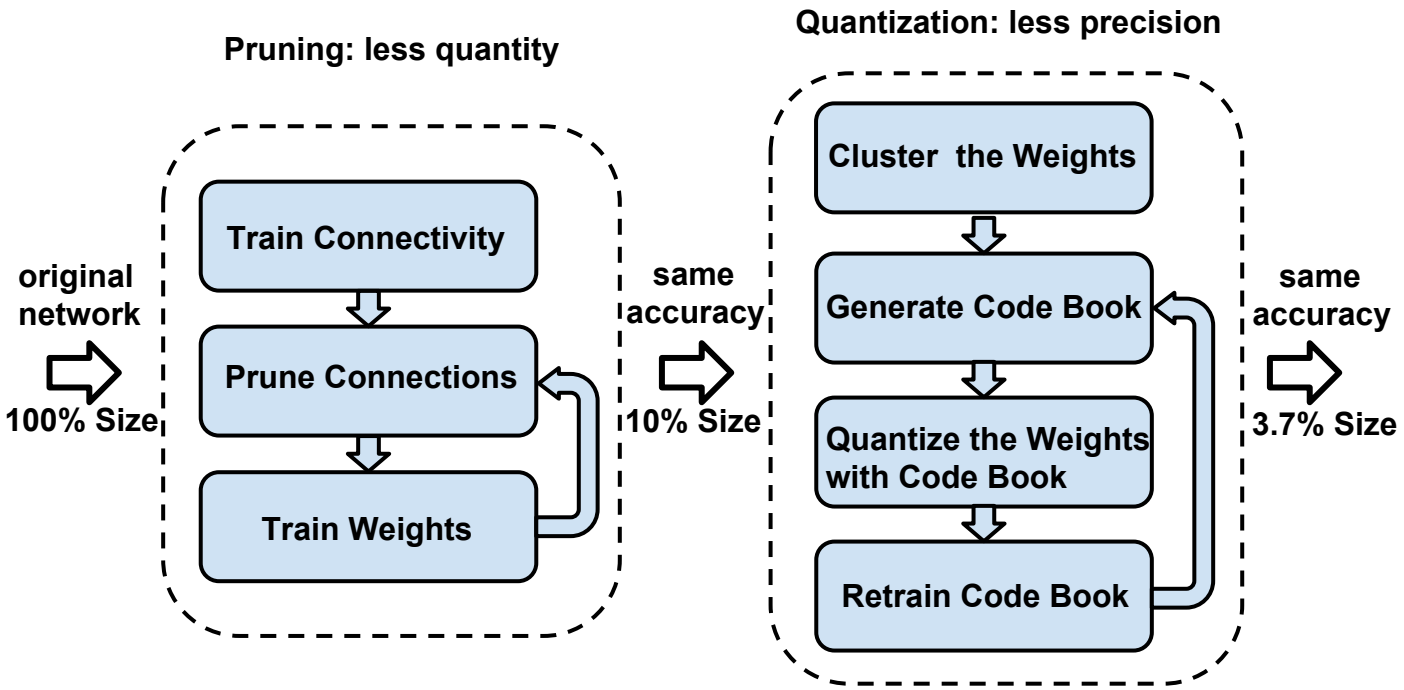
Tue Dec 8th 07:00 - 11:59 PM @ 210 C #12

Learning both Weights and Connections for Efficient Neural Network

Song Han · Jeff Pool · John Tran · Bill Dally

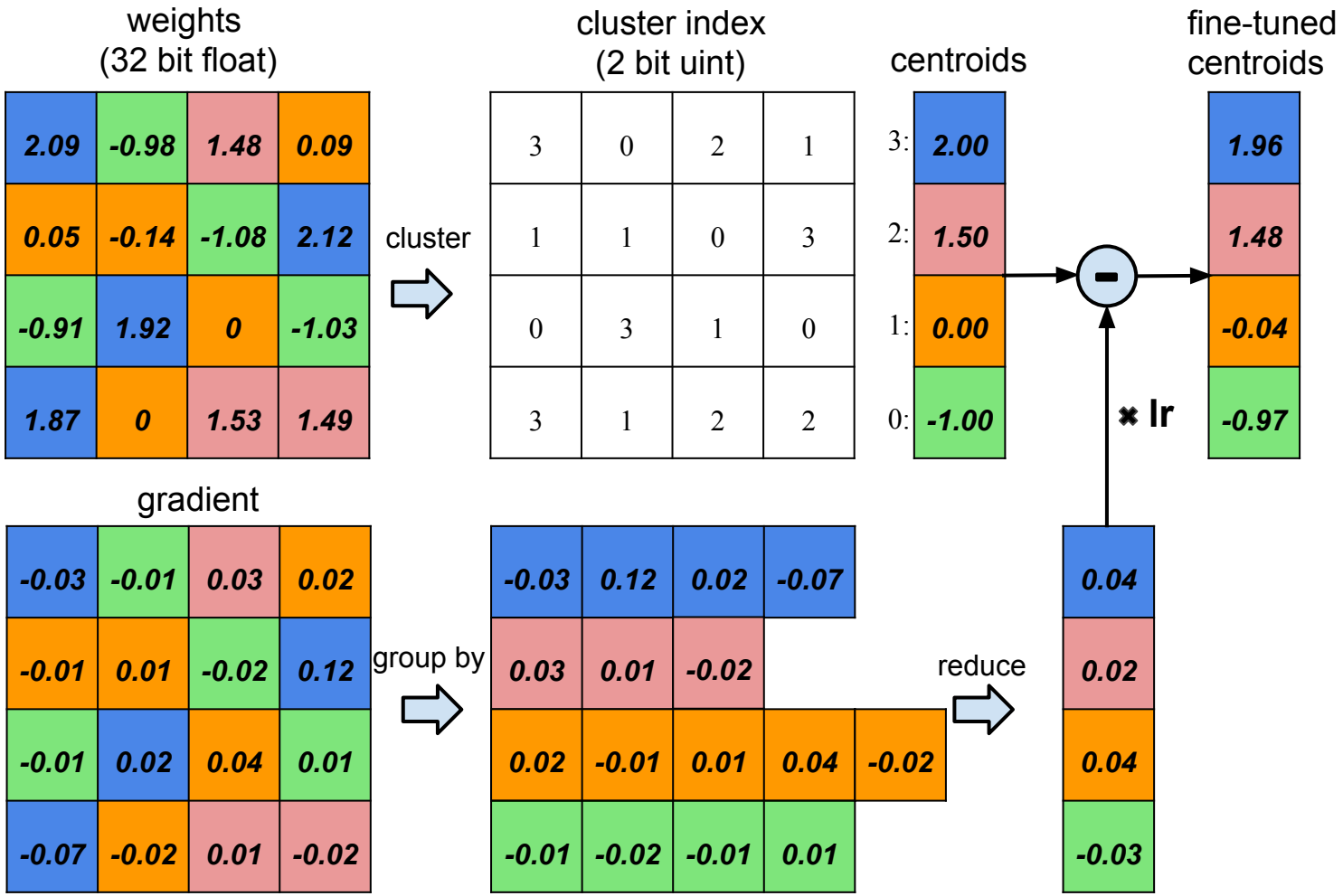
Reduce Storage for Each Remaining Weight

Trained Quantization
(Weight Sharing)



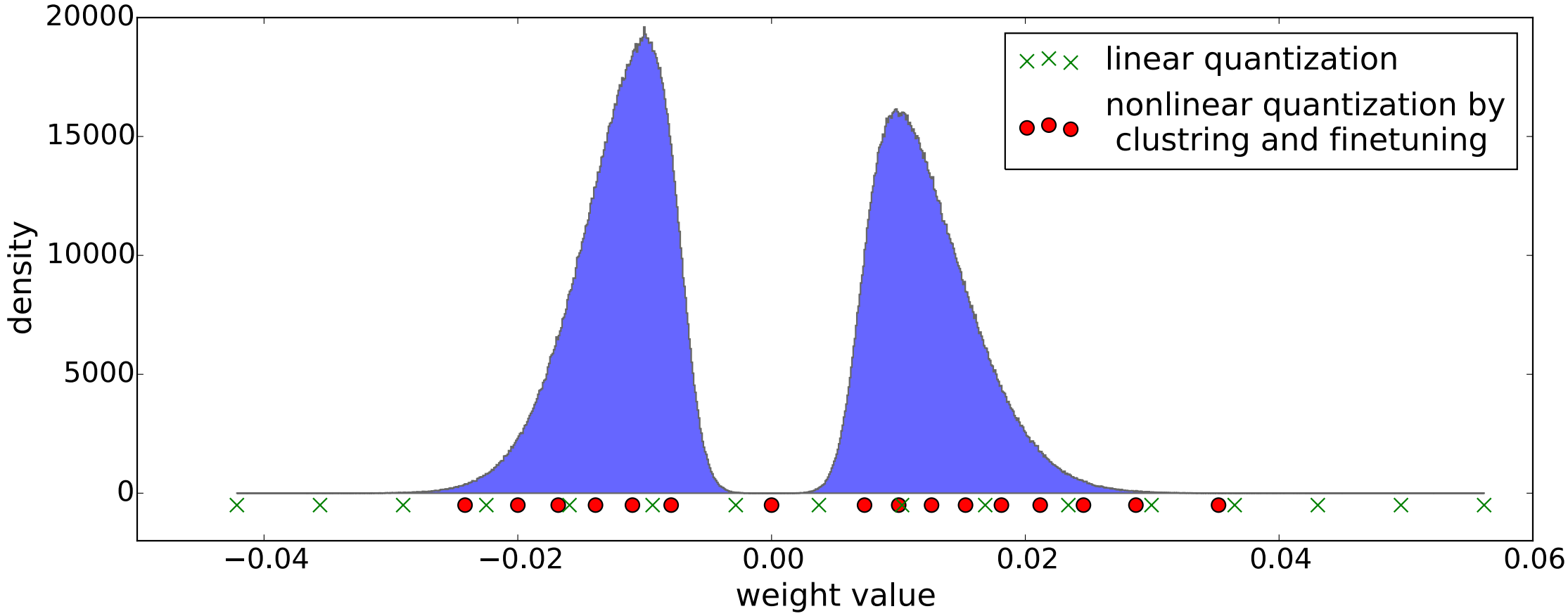
Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, arXiv 2015

Weight Sharing via K-Means



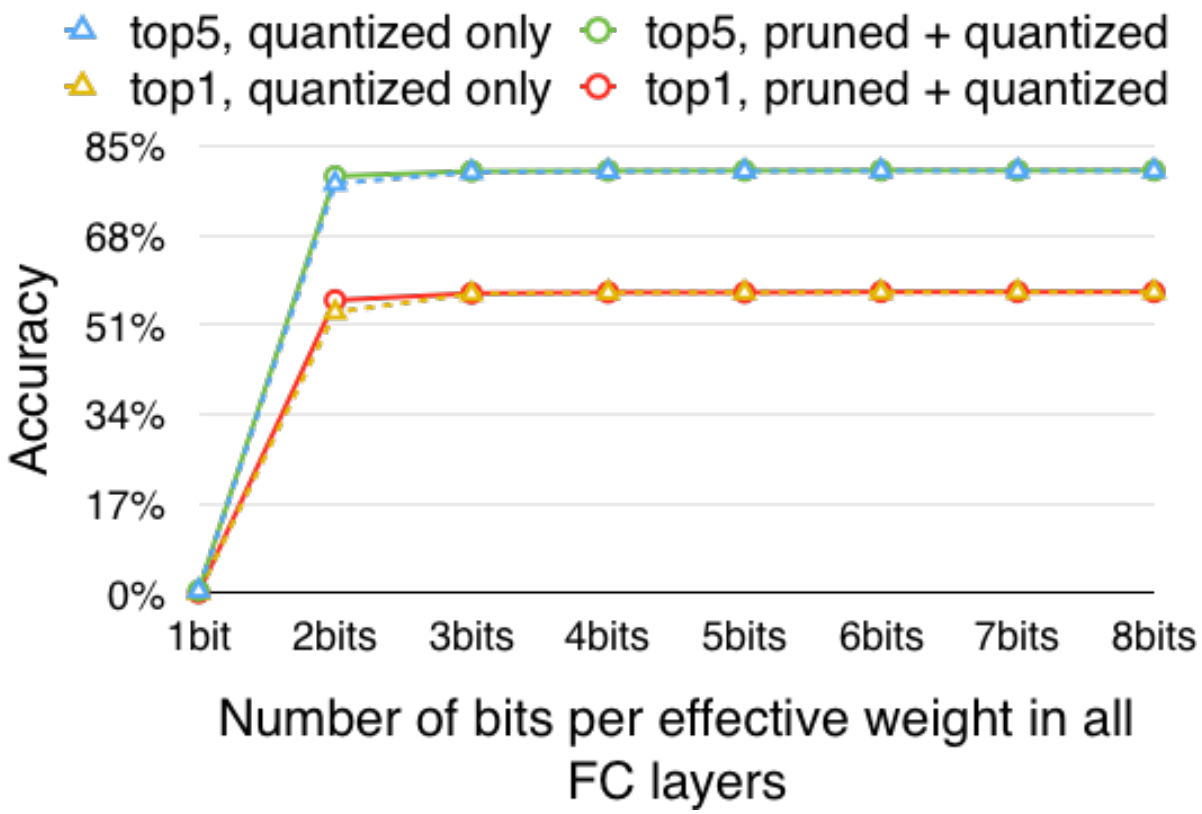
Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, arXiv 2015

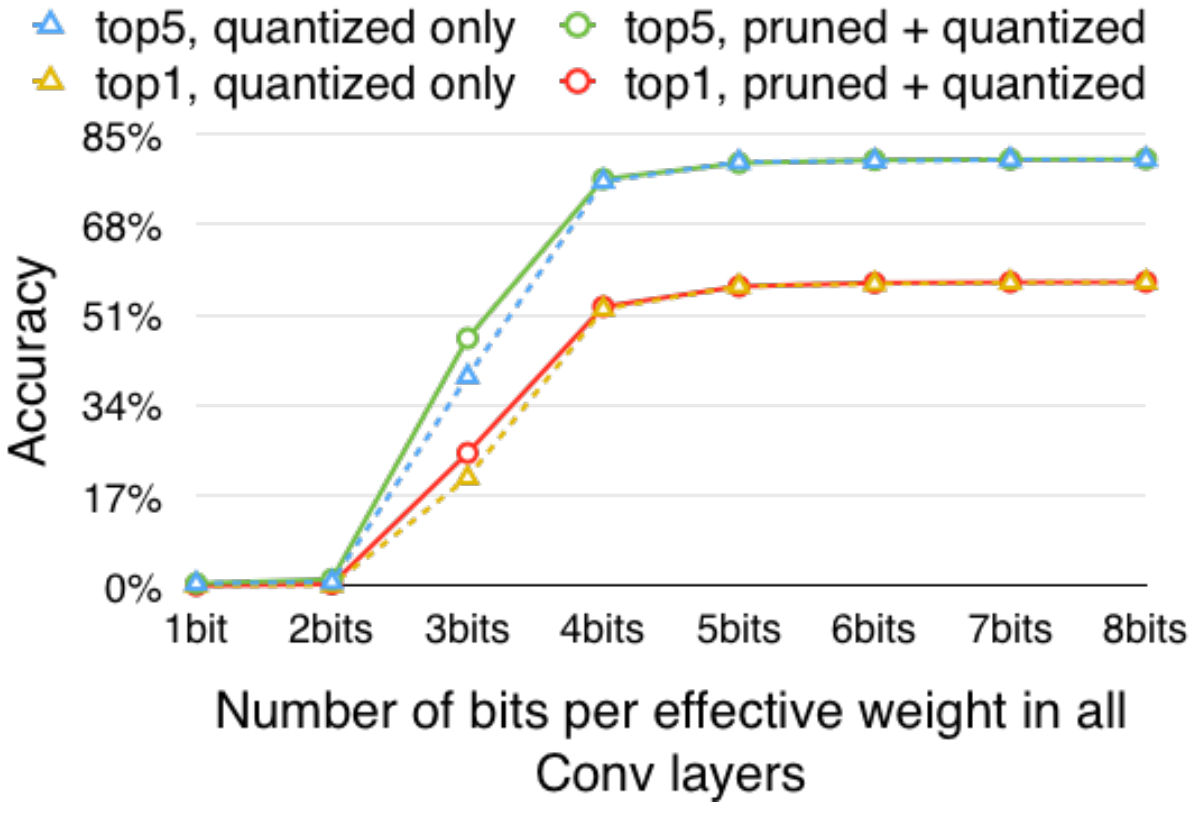
Trained Quantization



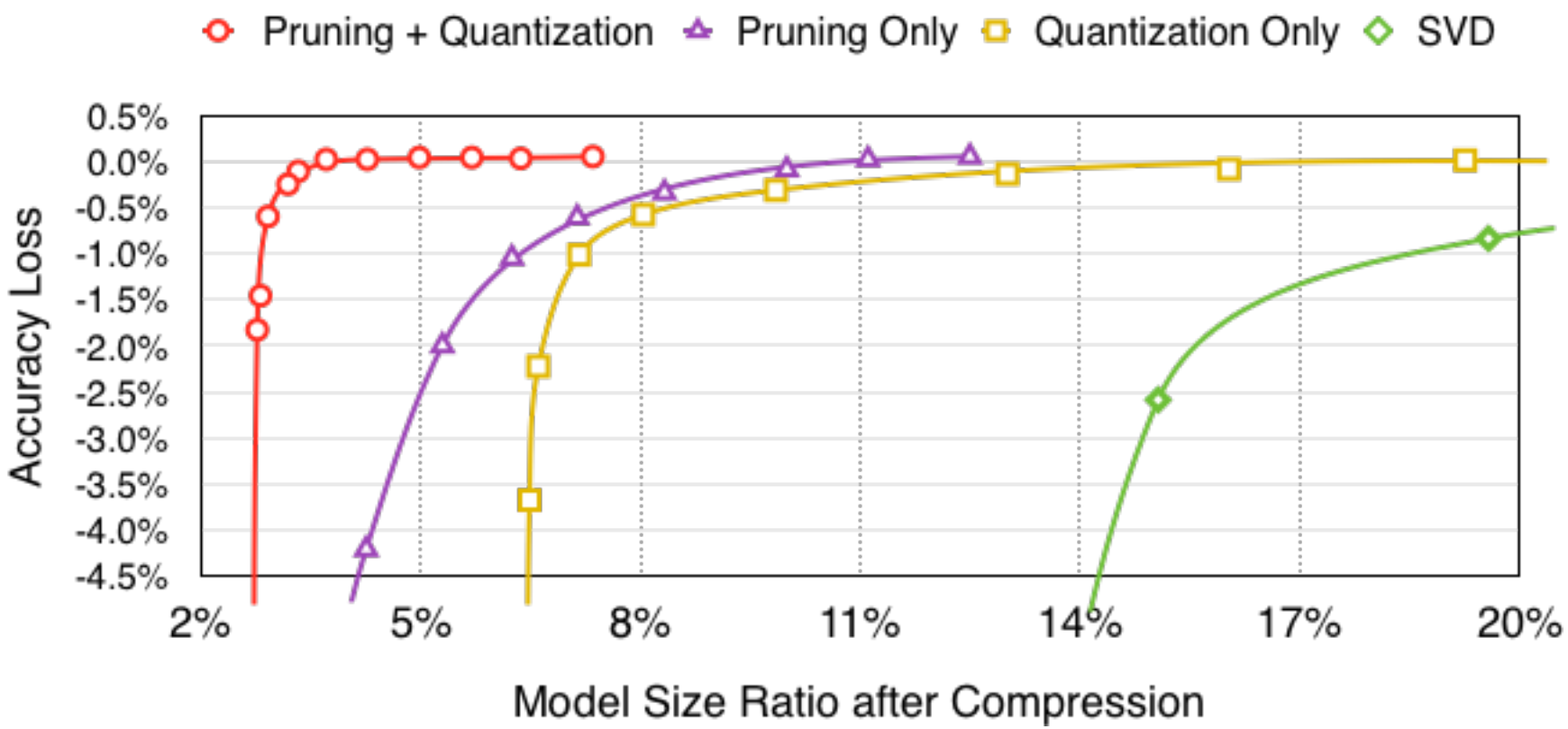
Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, arXiv 2015

Bits per Weight





Pruning + Trained Quantization



See Workshop Poster:

Thur Dec 10 3:00 - 7:00 PM *Deep Learning Symposium* @ 210 A, B Level 2

Deep Compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding

Song Han · Huizi Mao· Bill Dally

Paper

Demo:
Pocket AlexNet

Summary of Compression

Table 1: The compression pipeline can save 35× to 49× parameter storage with no loss of accuracy.

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	**40×**
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	**39×**
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	**35×**
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	**11.3 MB**	**49×**

Compress neural networks without affecting accuracy by:

1. Pruning the unimportant connections =>
2. Quantizing the network and enforce weight sharing =>
3. Apply Huffman encoding

30x – 50x Compression Means

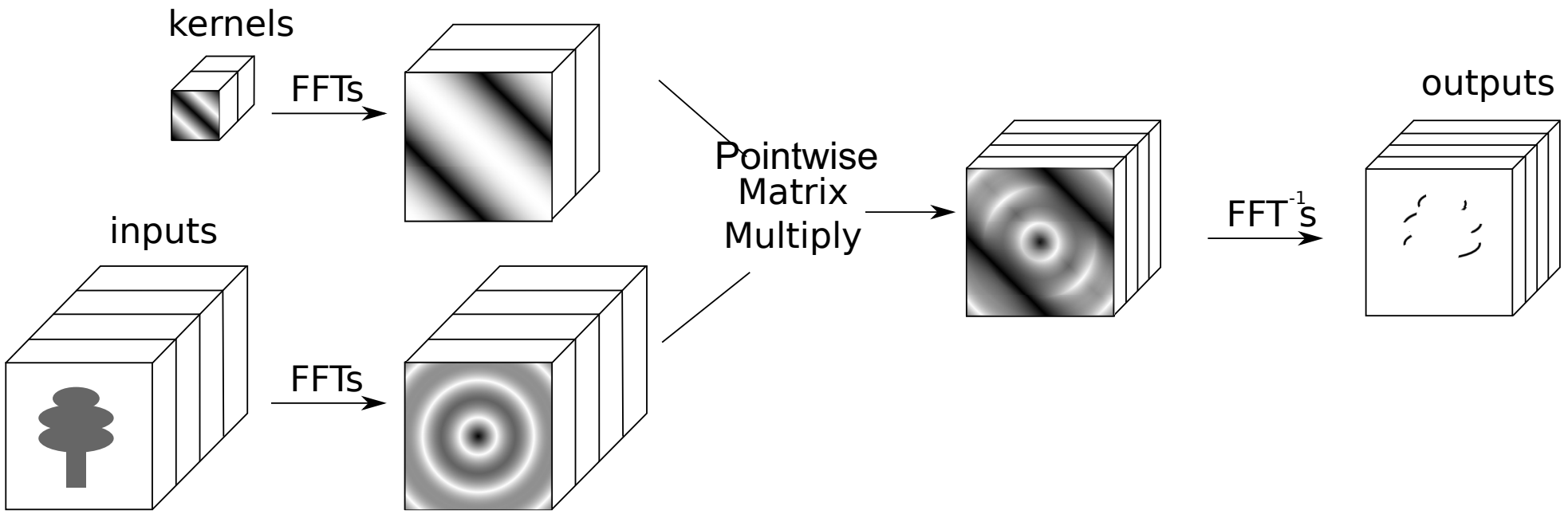
- Complex DNNs can be put in mobile applications (<100MB total)
  - 1GB network (250M weights) becomes 20-30MB
- Memory bandwidth reduced by 30-50x
  - Particuarly for FC layers in real-time applications with no reuse
- Memory working set fits in on-chip SRAM
  - 5pJ/word access vs 640pJ/word

Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

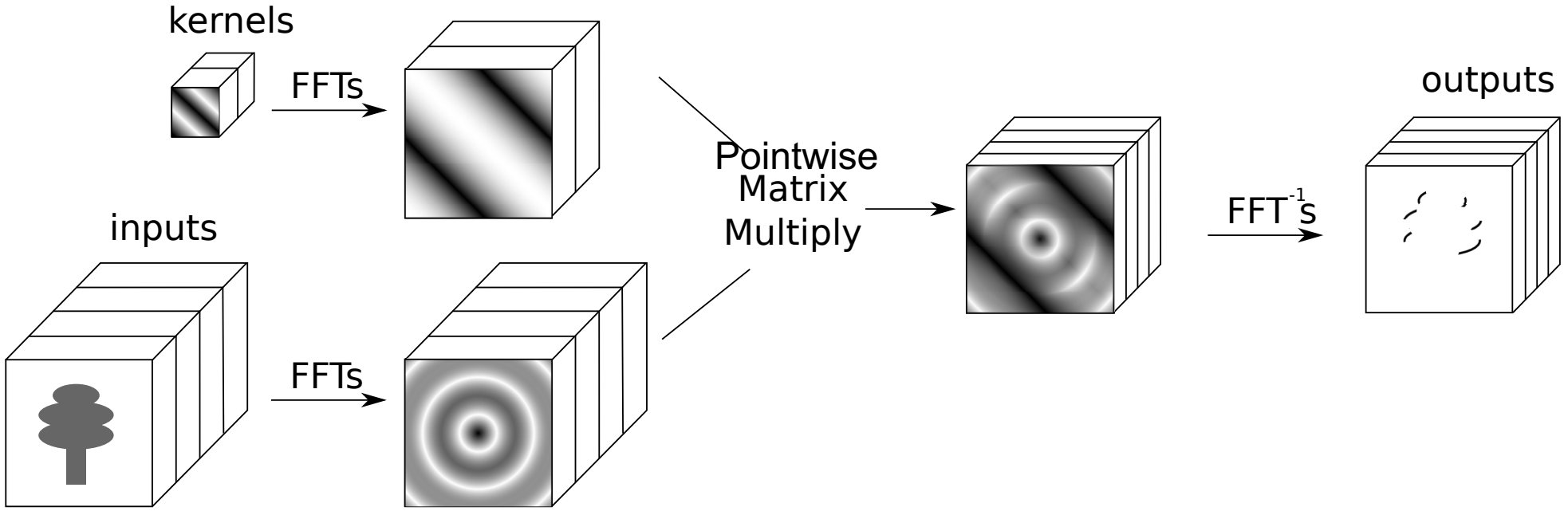
Before accelerating, make sure you have the fastest algorithm

FFT for Convolution



Mathieu, Henaff, & LeCunn, Fast Training of Convolutional Networks through FFTs, CVPR, 2014

FFT for Convolution



FFTs are amortized over K input maps and J output maps

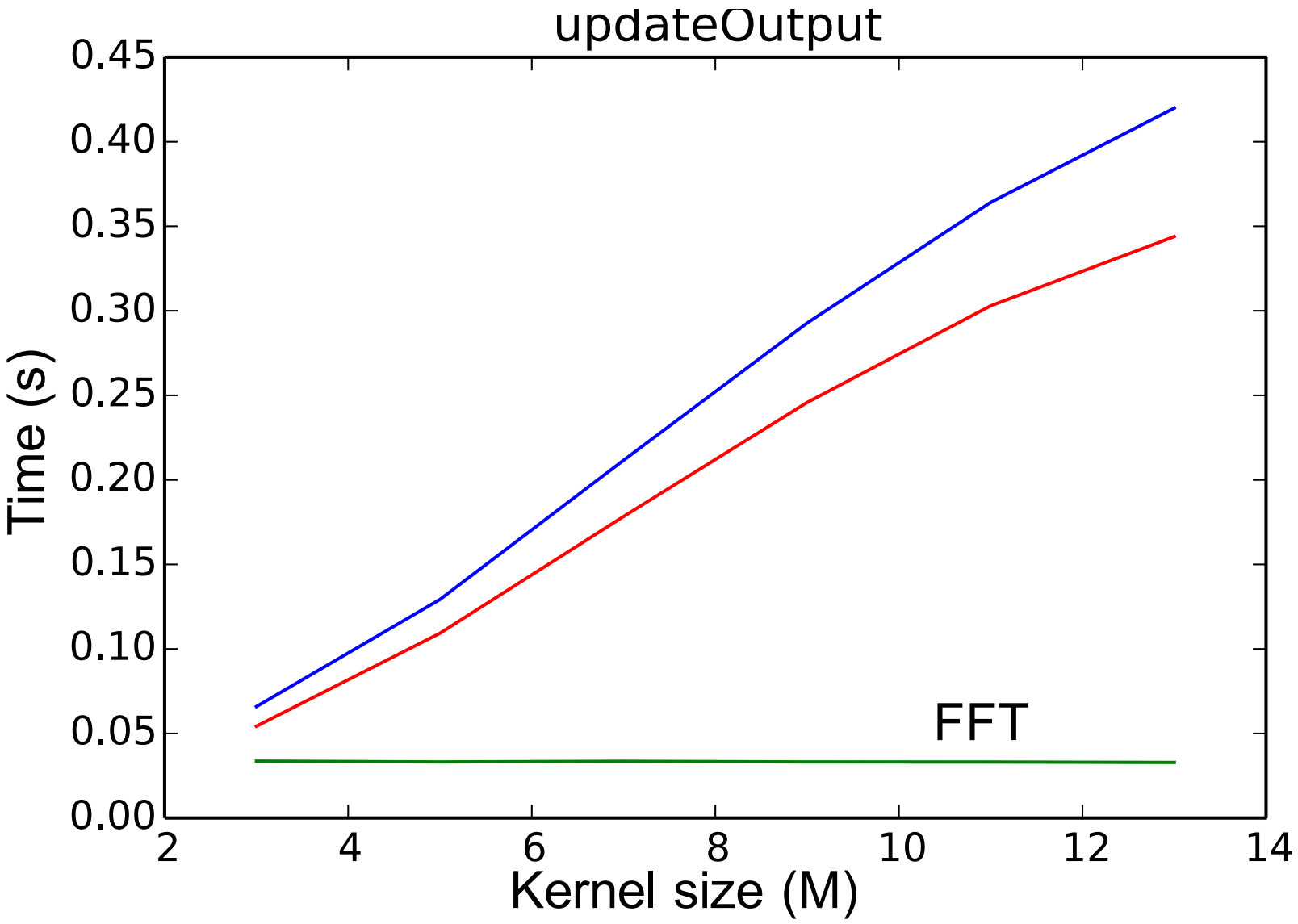
Conventional convolution is KJM^2N^2 ops for MxM kernel and NxN maps

FFT is $4KJN^2 + C(K+J)(2N^2 \log N)$

Faster – even for M=3 – with moderate sized K, J.

Mathieu, Henaff, & LeCunn, Fast Training of Convolutional Networks through FFTs, 2013

FFT Performance



Mathieu, Henaff, & LeCunn, Fast Training of Convolutional Networks through FFTs, 2013

Winograd Convolution

$$
\begin{aligned}
Y_{i,k,\widetilde{x},\widetilde{y}} &= \sum_{c=1}^{C} D_{i,c,\widetilde{x},\widetilde{y}} * G_{k,c} \\
&= \sum_{c=1}^{C} A^T \left[U_{k,c} \odot V_{c,i,\widetilde{x},\widetilde{y}} \right] A \\
&= A^T \left[\sum_{c=1}^{C} U_{k,c} \odot V_{c,i,\widetilde{x},\widetilde{y}} \right] A
\end{aligned}
$$

Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980

Lavin & Gray, Fast Algorithms for Convolutional Neural Networks, 2015

Summary of Algorithms

- FFT or Winograd convolution
  - ~2x faster for 3x3 convolutions
  - ~25x faster for 11x11 convolutions
- Special purpose hardware running brute-force convolution looses its advantage vs. GPU running FFT convolutions
- FFT convolution cost is independent of convolution size

Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

To be maximally efficient use special-purpose hardware

Unless you are memory limited

Diannao (Electric Brain)

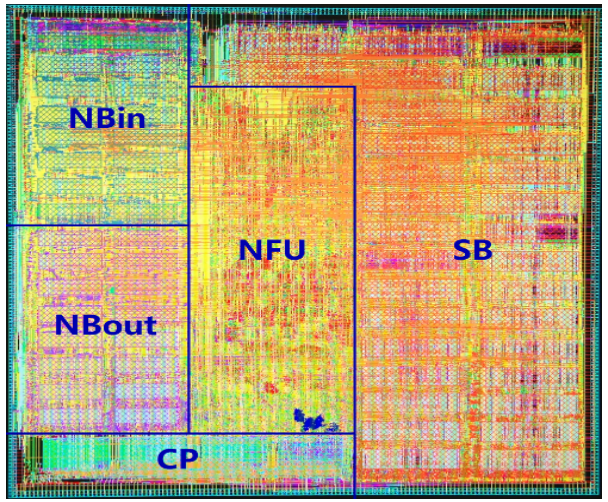


Figure 15. *Layout (65nm).*

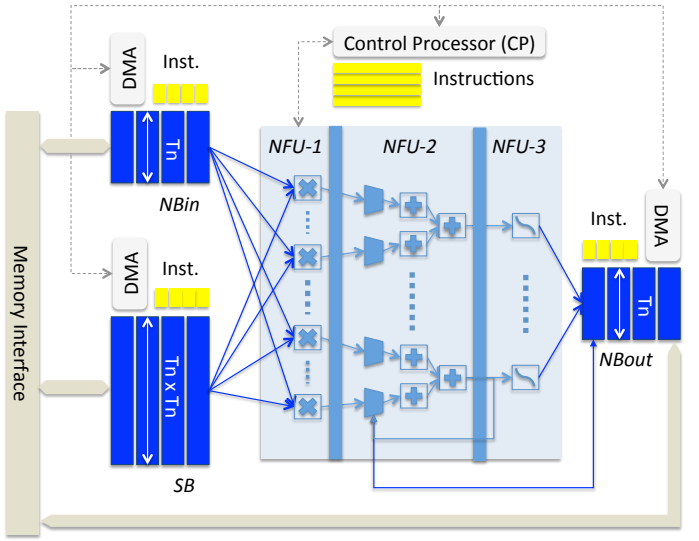


Figure 11. *Accelerator.*

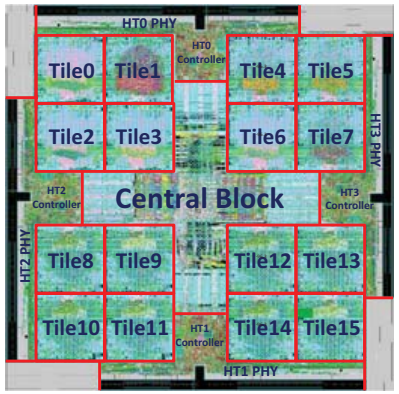
Component or Block	Area in μm^2	(%)	Power in mW	(%)	Critical path in ns
ACCELERATOR	3,023,077		485		1.02
Combinational	608,842	(20.14%)	89	(18.41%)	
Memory	1,158,000	(38.31%)	177	(36.59%)	
Registers	375,882	(12.43%)	86	(17.84%)	
Clock network	68,721	(2.27%)	132	(27.16%)	
Filler cell	811,632	(26.85%)			
SB	1,153,814	(38.17%)	105	(22.65%)	
NBin	427,992	(14.16%)	91	(19.76%)	
NBout	433,906	(14.35%)	92	(19.97%)	
NFU	846,563	(28.00%)	132	(27.22%)	
CP	141,809	(5.69%)	31	(6.39%)	
AXIMUX	9,767	(0.32%)	8	(2.65%)	
Other	9,226	(0.31%)	26	(5.36%)	

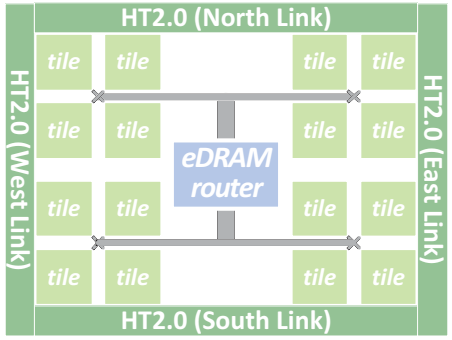
Table 6. *Characteristics of accelerator and breakdown by component type (first 5 lines), and functional block (last 7 lines).*

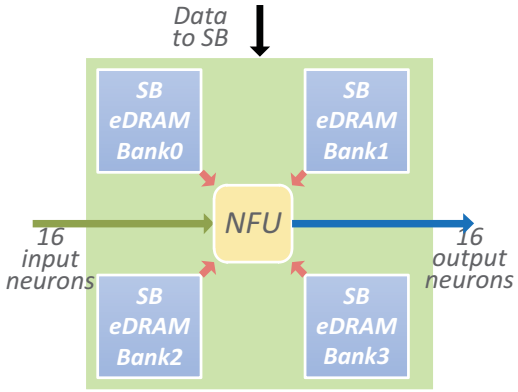
- Diannao improved CNN computation efficiency by using dedicated functional units and memory buffers optimized for the CNN workload.
- Multiplier + adder tree + shifter + non-linear lookup orchestrated by instructions
- Weights in off-chip DRAM
- 452 GOP/s, 3.02 mm^2 and 485 mW

Chen et al. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning, ASPLOS 2014

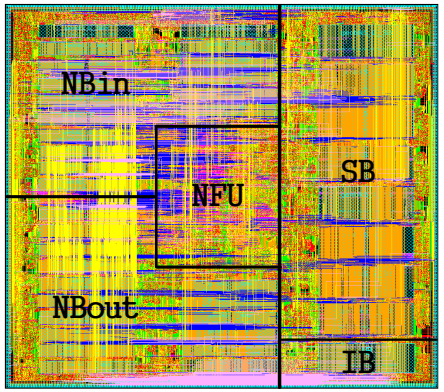
Diannao and Friends

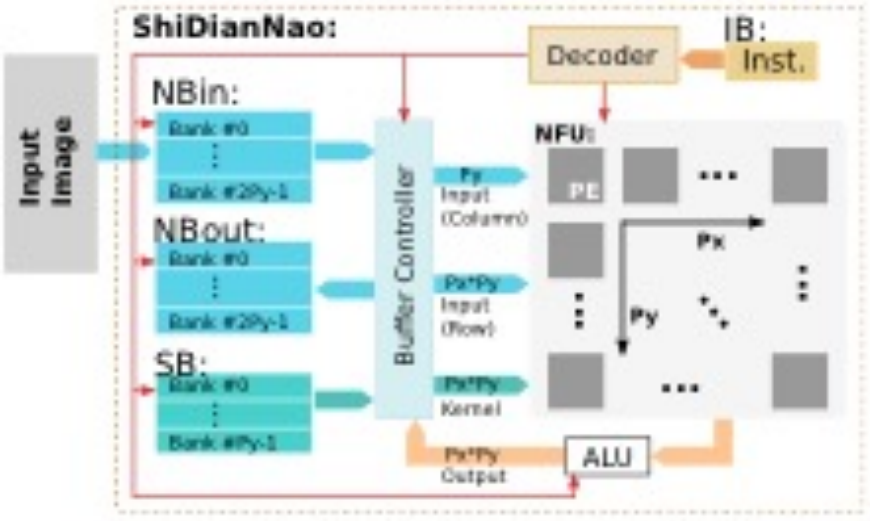






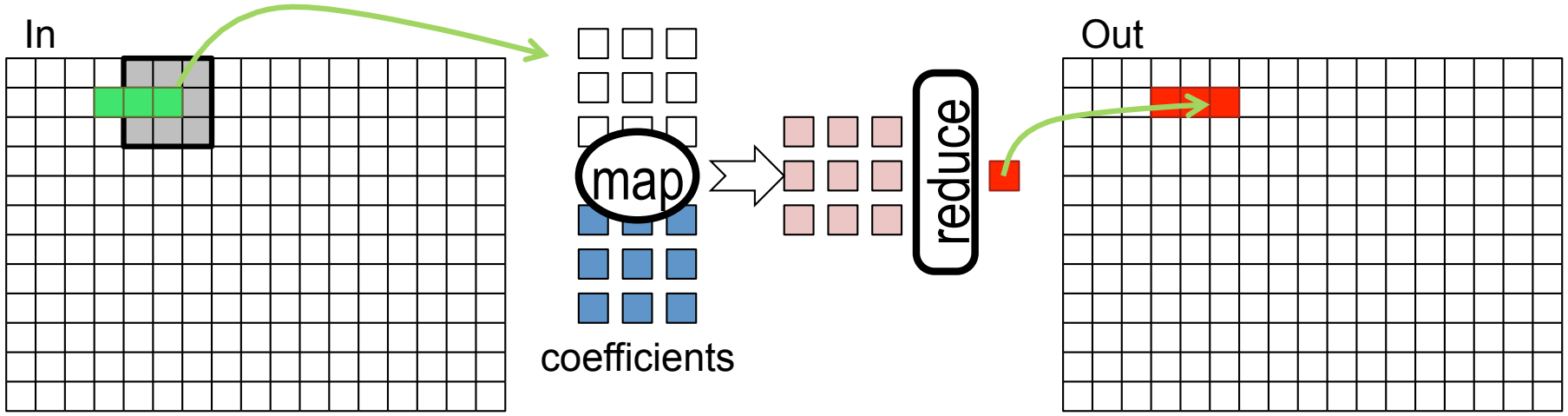
DaDiannao (Bigger Computer) uses multi-chip and EDRAM to fit larger models. Each chip is 68mm^2 fitting 12 Million parameters, consumes 16W





ShiDiannao (Vision Computer) It can fits small model (up-to 64K parameters) on-chip. It maps the computation on 2D PE array. The chip is 4.86 mm^2 and consumes 320 mW ,

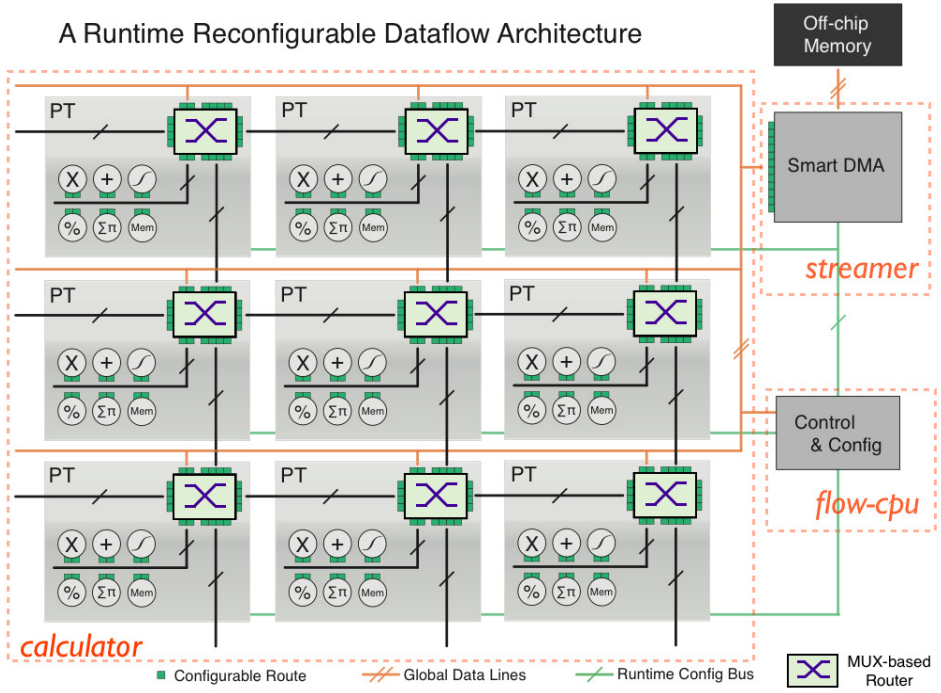
Convolution Engine

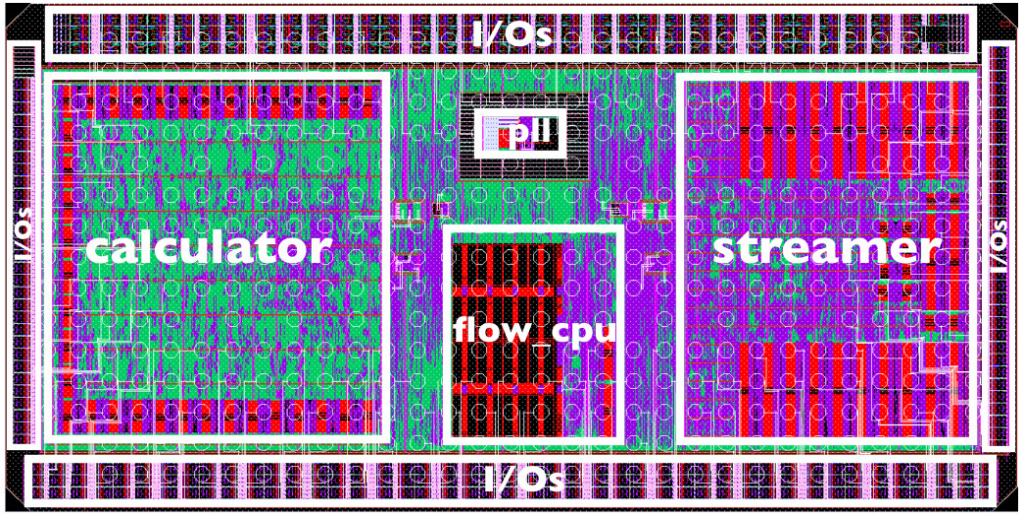


- Convolution Engine (CE), is specialized for the convolution-like data-flow that is common in image processing.
- CE achieves energy efficiency by capturing data reuse patterns, eliminating data transfer overheads, and enabling a large number of operations per memory access.
- With restricted the domain in image and video processing, flexible convolution engine improves improves energy and area efficiency by 8-15x over a SIMD engine.

Wajahat Qadeer et.al, Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing

NeuFlow



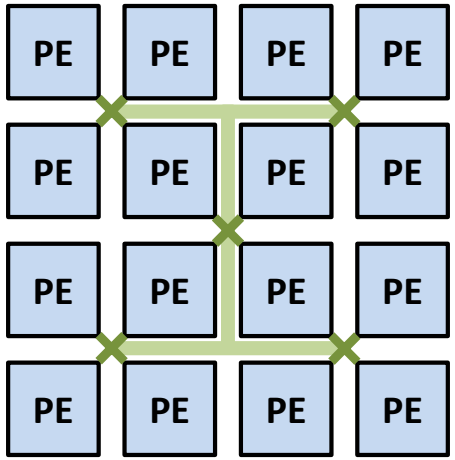


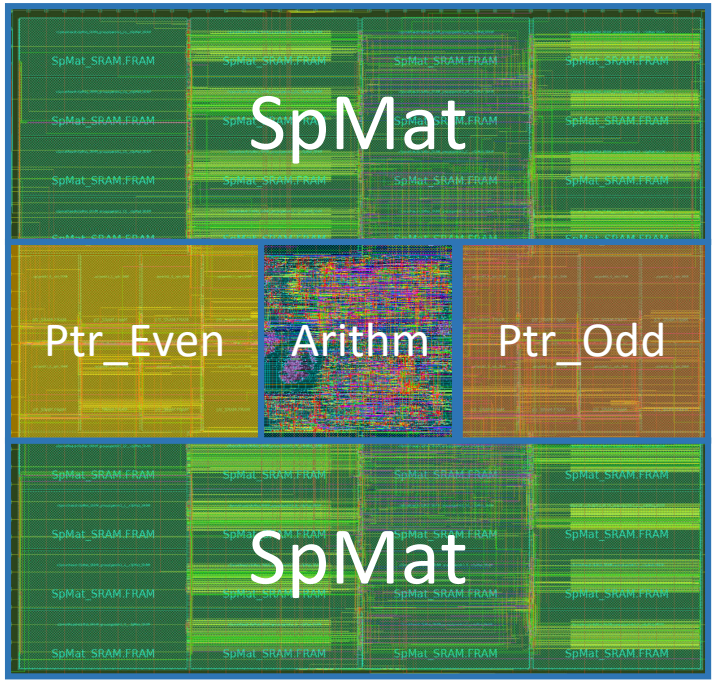
- An SoC designed to accelerate neural networks and other complex vision algorithms based on large numbers of convolutions and matrix-to-matrix operations.
- 160 GOPS, 570 mW, 12.5 mm^2 @ IBM SOI 45nm

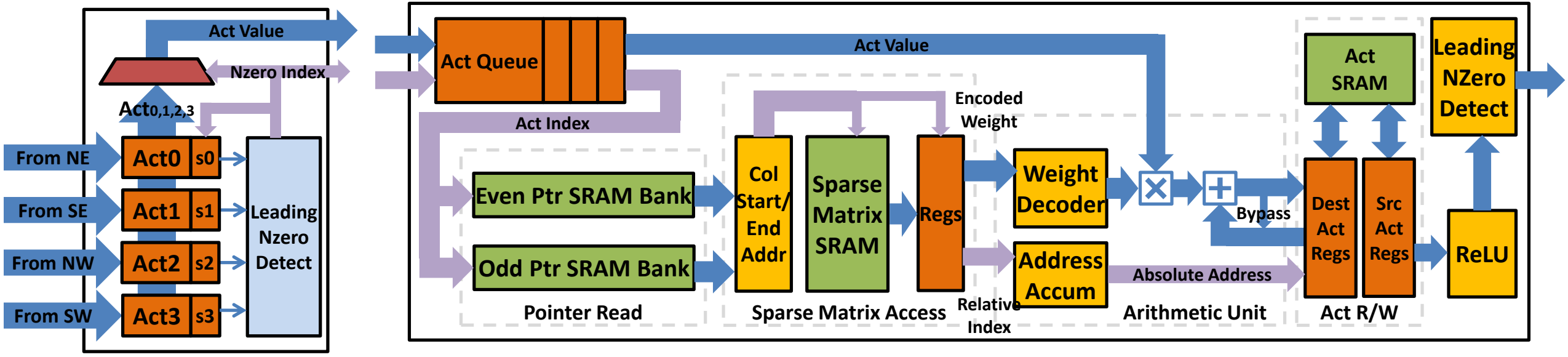
P. Pham et.al, NeuFlow: Dataflow Vision Processing System-on-a-Chip

Efficient Inference Engine

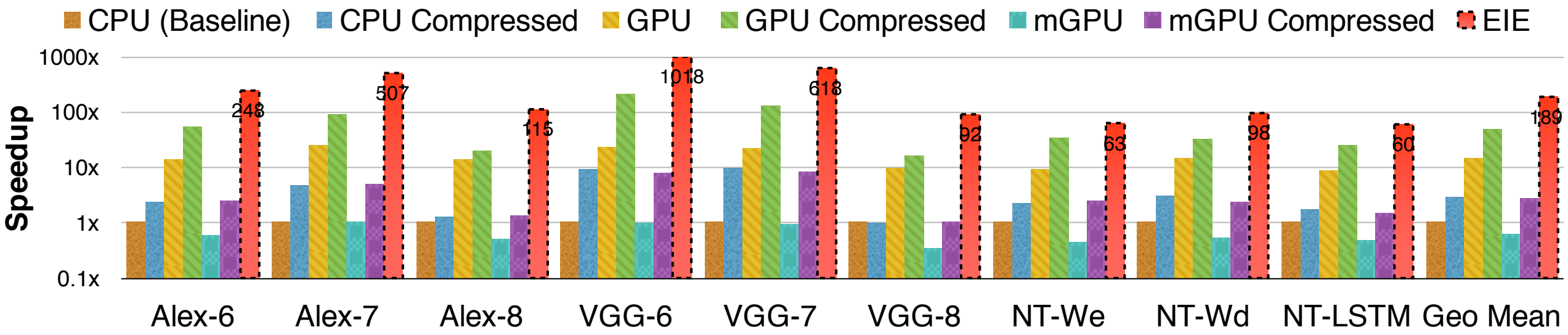
	Power (mW)	(%)	Area (μm^2)	(%)
Total	9.157		638,024	
memory	5.416	(59.15%)	594,786	(93.22%)
clock network	1.874	(20.46%)	866	(0.14%)
register	1.026	(11.20%)	9,465	(1.48%)
combinational	0.841	(9.18%)	8,946	(1.40%)
filler cell			23,961	(3.76%)
Act_queue	0.112	(1.23%)	758	(0.12%)
PtrRead	1.807	(19.73%)	121,849	(19.10%)
SpmatRead	4.955	(54.11%)	469,412	(73.57%)
ArithmUnit	1.162	(12.68%)	3,110	(0.49%)
ActRW	1.122	(12.25%)	18,934	(2.97%)
filler cell			23,961	(3.76%)



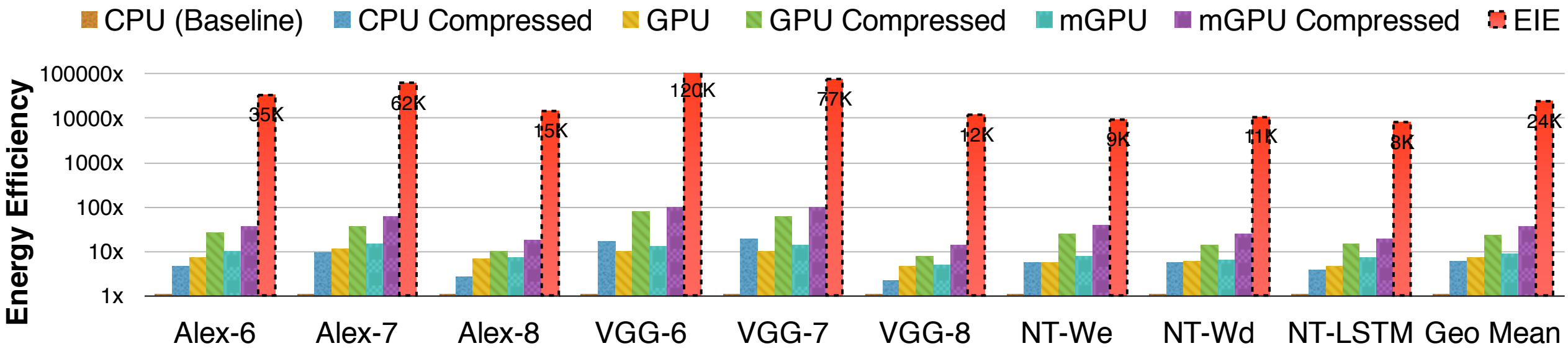




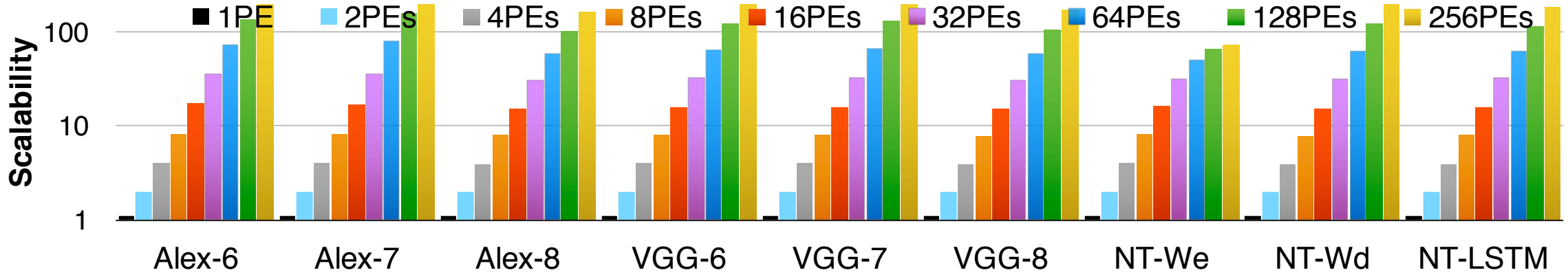
Speedup

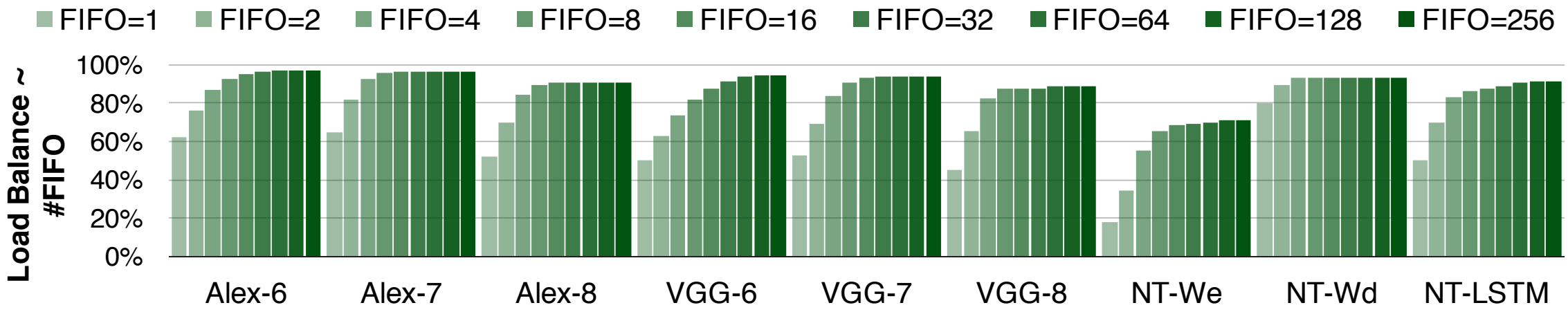

CPU (Baseline)
CPU Compressed
GPU
GPU Compressed
mGPU
mGPU Compressed
EIE
Speedup
1000x
100x
10x
1x
0.1x
248
507
115
1018
618
92
63
98
60
189
Alex-6
Alex-7
Alex-8
VGG-6
VGG-7
VGG-8
NT-We
NT-Wd
NT-LSTM
Geo Mean

Energy Efficiency



Scalability and load balancing





FPGAs

- A field-configurable ASIC
- Fixed-function units have good efficiency
  - Arithmetic units (int and FP)
  - RAMs
  - ARM cores
- Logic built from LUTs has poor efficiency
  - 30-100x worse ops/J than an ASIC

Microsoft Experience

ImageNet-1K Classification Performance

Platform	Library/OS	ImageNet 1K Inference Throughput	Peak TFLOPs	Effective TFLOPs	Estimated Peak Power with Server	Estimated GOPs/J (assuming peak power)
16-core, 2-socket Xeon E5-2450, 2.1GHz	Caffe + Intel MKL Ubuntu 14.04.1*	53 images/s	0.27T	0.074T (27%)	~225W	~0.3
Arria 10 GX1150	Windows Server 2012	369 images/s[1]	1.366T	0.51T (38%)	~265W	~1.9
NervanaSys-32 on NVIDIA Titan X	NervanaSys-32 on Ubuntu 14.0.4	4129 images/s[2]	6.1T	5.75T (94%)	~475W	~12.1

Includes server power; however, CPUs available to other jobs in the datacenter

[1]Dense layer time estimated
[2]https://github.com/soumith/convnet-benchmarks

Comparison of FPGAs

	[1]	[2]	[3]	[4]
Year	2010	2014	2015	2015
Platform	Virtex5 SX240t	Zynq XC7Z045	Virtex7 VX485t	Zynq XC7Z045
Clock(MHz)	120	150	100	150
Bandwidth (GB/s)	–	4.2	12.8	4.2
Quantization Strategy	48-bit fixed	16-bit fixed	32-bit float	16-bit fixed
Power (W)	14	8	18.61	9.63
Problem Complexity (GOP)	0.52	0.552	1.33	30.76
Performance (GOP/s)	16	23.18	61.62	187.80 (CONV) 136.97 (Overall)
Resource Efficiency (GOP/s/Slices)	4.30×10^{-4}	–	8.12×10^{-4}	3.58×10^{-3} (CONV) 2.61×10^{-3} (Overall)
Power Efficiency (GOP/s/W)	1.14	2.90	3.31	19.50 (CONV) 14.22 (Overall)

[1] S.Chakradhar, et.al,"A dynamically configurable coprocessor for convolutional neural networks," in ACM SIGARCH Computer Architecture News,
[2] V. Gokhale, et.al, "A 240 g-ops/s mobile coprocessor for deep neural networks," in Computer Vision and Pattern Recognition Workshops (CVPRW),
[3] C. Zhang et.al, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in FPGA 2015
[4] J. Qiu et.al, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network", to appear in FPGA 2016

Hardware Comparison

Platform	Titan X	Tegra K1	A-Eye [14]	DaDianNao[11]	EIE (ours)
Year	2015	2014	2015	2014	2015
Platform Type	GPU	mGPU	FPGA	ASIC	ASIC
Technology	28nm	28nm	-	28nm	45nm
Clock (MHz)	1075	852	150	606	800
Memory type	DRAM+ SRAM	DRAM+ SRAM	DRAM	eDRAM+ SRAM	SRAM
Max DNN model size	<3G	<500M	<500M	11.3M	84M
Quantization Stategy	32-bit float	32-bit float	16-bit fixed	16-bit fixed	4-bit → 16-bit fixed
Area (mm^2)	-	-	-	67.7	40.8
Peak Throughput (GOP/s)	3225	365	188	5580	102
Throughput for M×V (GOP/s)	138.1	5.8	1.2	205	94.6
Power(W)	250	8.0	9.63	15.97	0.59
Power Efficiency (GOP/s/W)	12.9	45.6	19.5	349.4	172.9
Power Efficiency for M×V (GOP/s/W)	**0.55**	**0.73**	**0.12**	**12.8**	**160.3**

[2] S. Han et.al, “EIE: Efficient Inference Engine on Compressed Deep Neural Network”, submitted to ISCA 2016

Summary of Special Purpose Hardware

- Diannao – 16 16b multiply-accumulators with buffers optimized for DNNs
  - All data stored in off-chip DRAM
- ShiDiannao – for conv layers – up to 64K parameters on chip
- DaDiannao – for FC layers – up to 12M parameters in on-chip EDRAM
- Convolution Engine – fast convolutions (brute-force algorithm)
- EIE – hardware for compressed networks
  - Trained quantization and pruning
  - No data movement – scalable to 256PEs

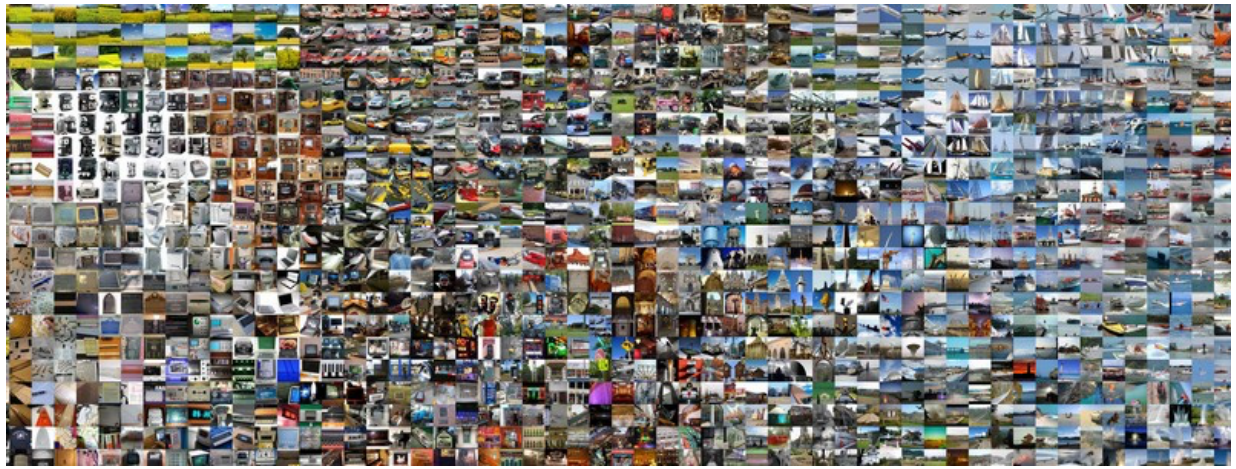
Bottom Line

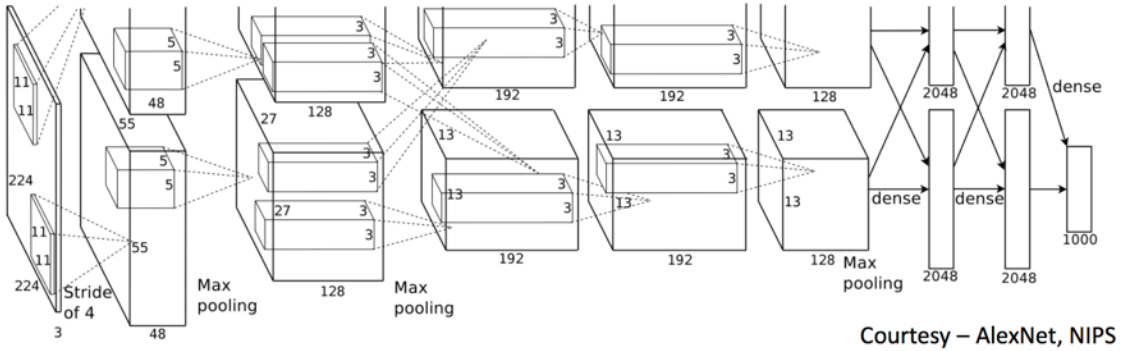
- Arithmetic perf/W of special purpose hardware is ~2x a GPU (FP16)
- Perf/W on memory limited layers (FC, not batch) is no better than GPU
- Big win from special-purpose hardware is
  - When entire network fits on chip
  - Decompressing highly-compressed networks
- FPGAs are just inefficient ASICs
  - Good arithmetic and on-chip memory
  - 30-100x less efficient elsewhere

Outline

- The Problem
- Baseline
- Parallelization
- GPUs
- Reduced Precision
- Compression
- Better Algorithms
- Hardware for DNNs
- Summary

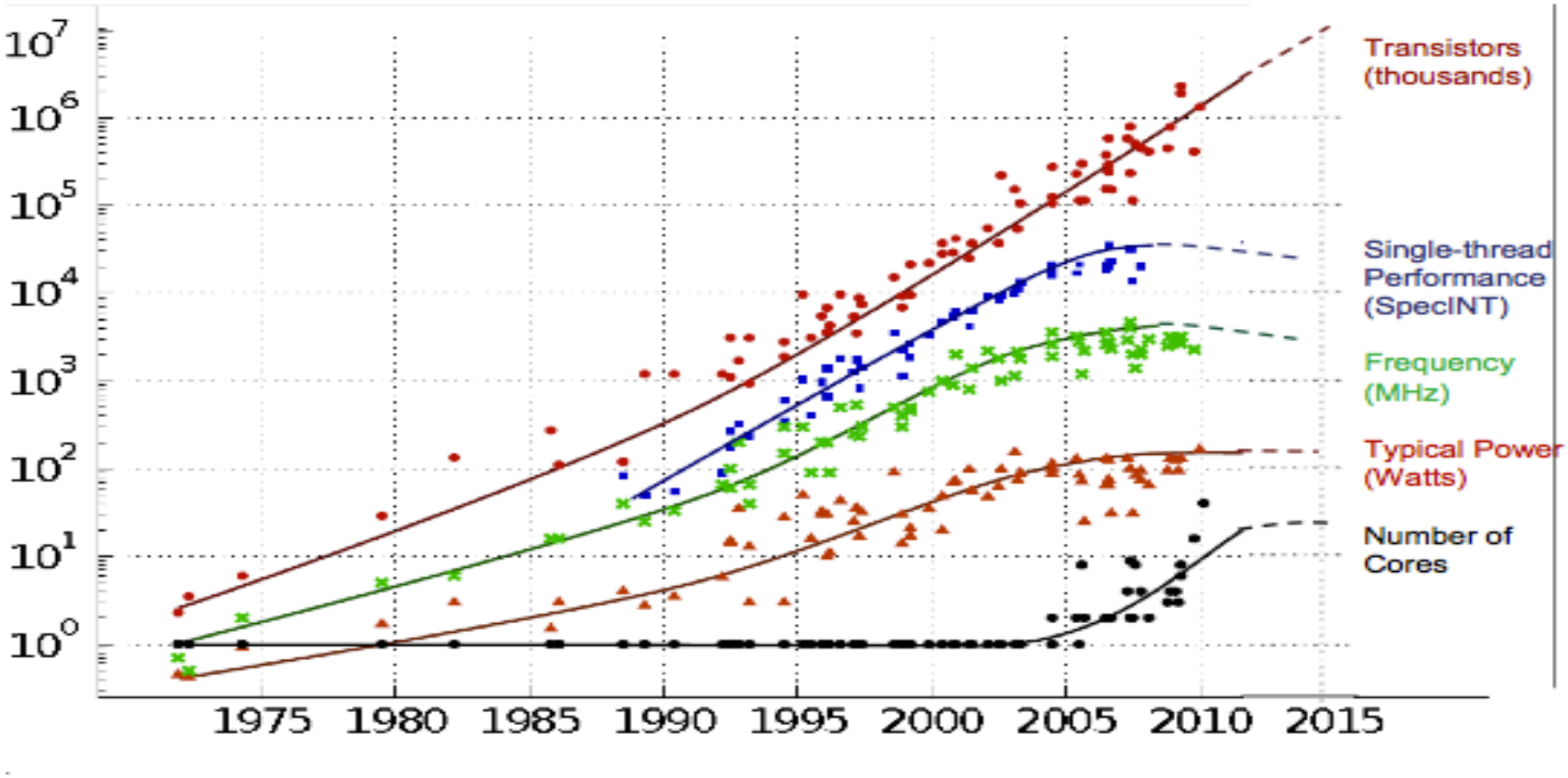
Hardware and Data enable DNNs





Courtesy – AlexNet, NIPS 2012

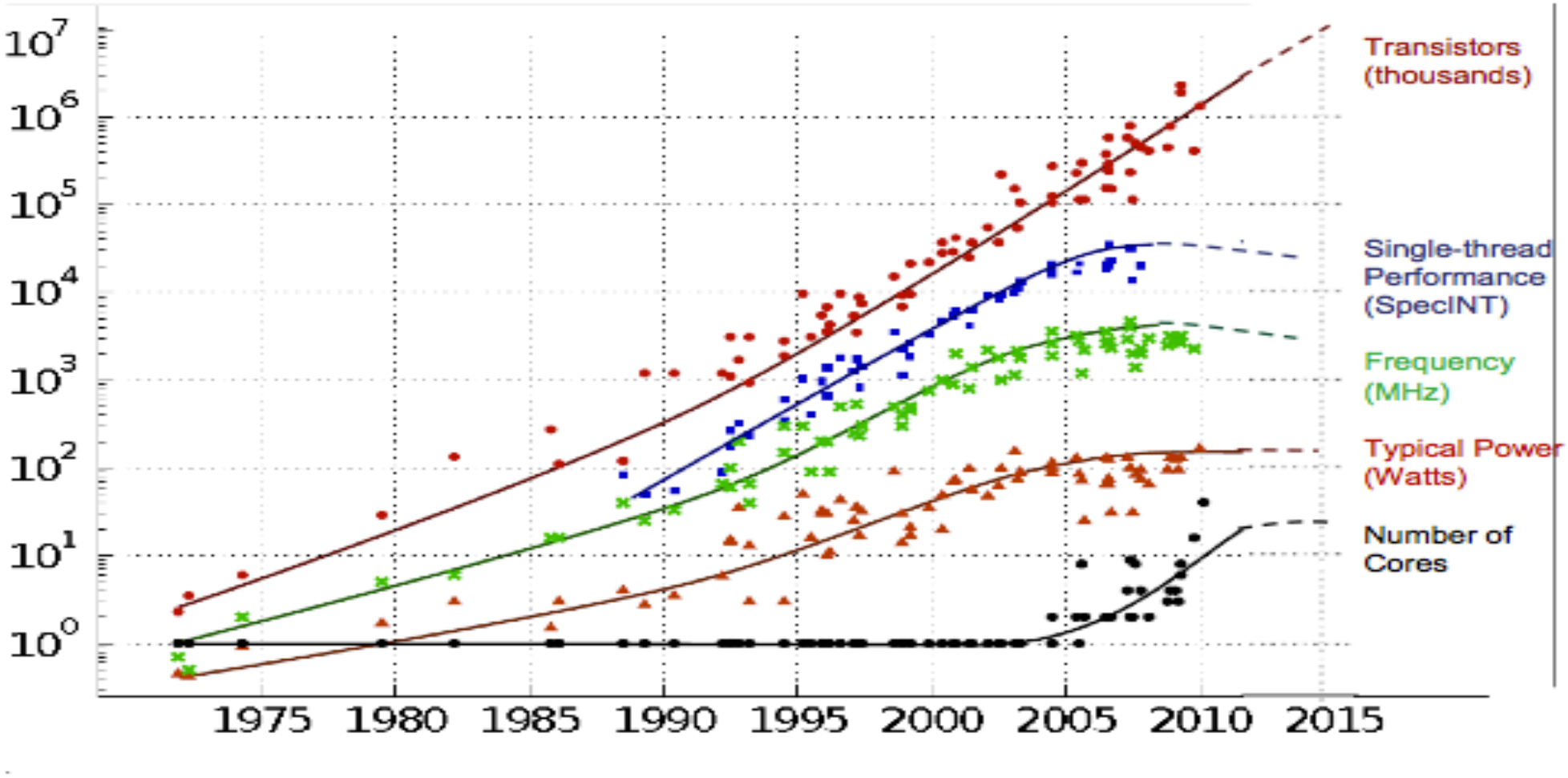
In 1990, CPUs had one 100 SpecINT Core



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

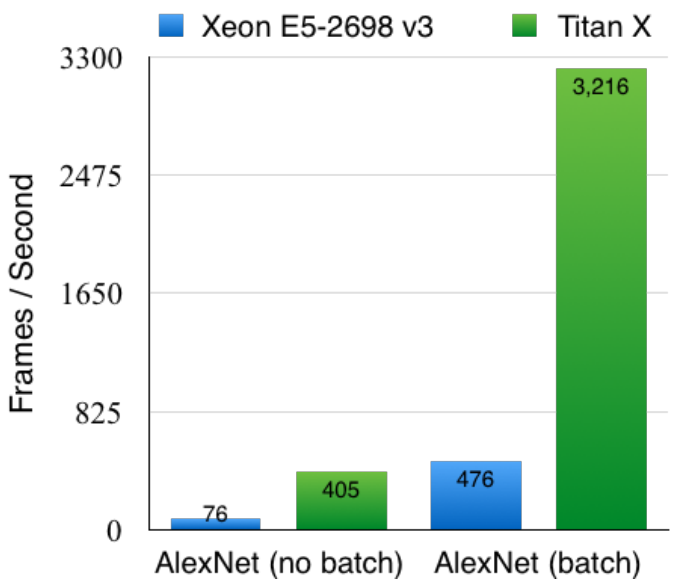
Today they have 6-8 30,000SpecINT cores (~200,000x) But Moore's Law is over…

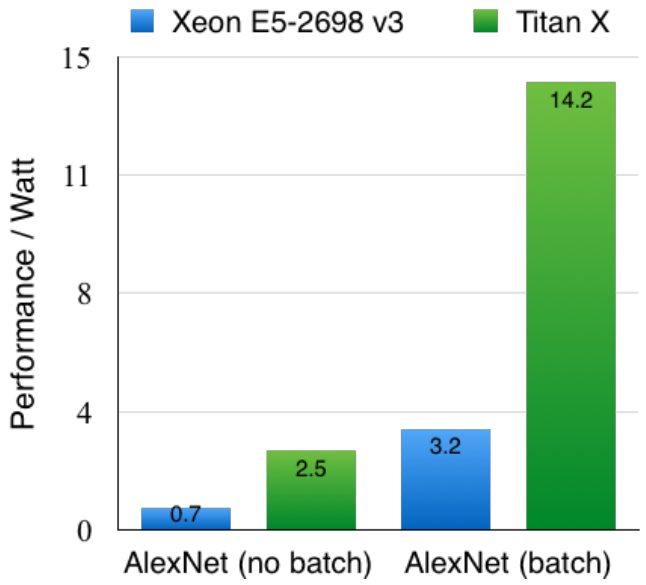


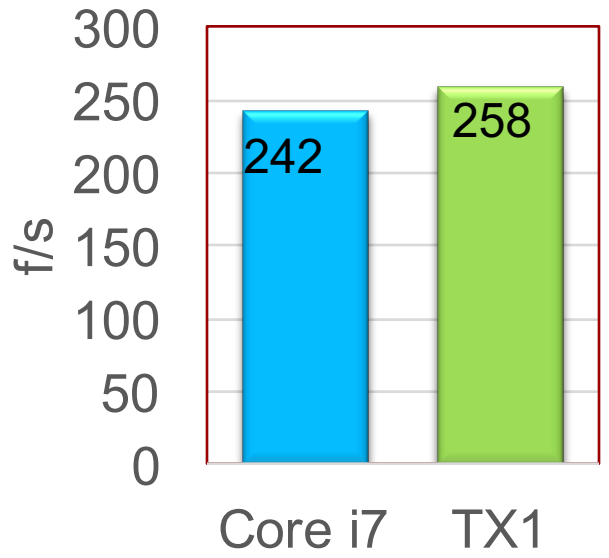
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

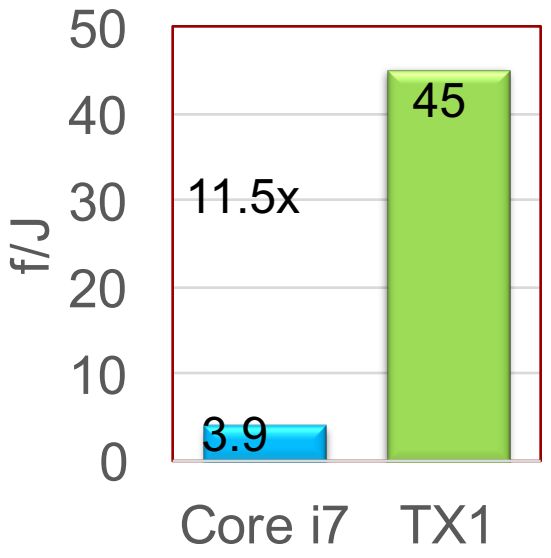
C Moore, *Data Processing in ExaScale-ClassComputer Systems,* Salishan, April 2011

GPUs give an additional 5-10x (2,000,000x)

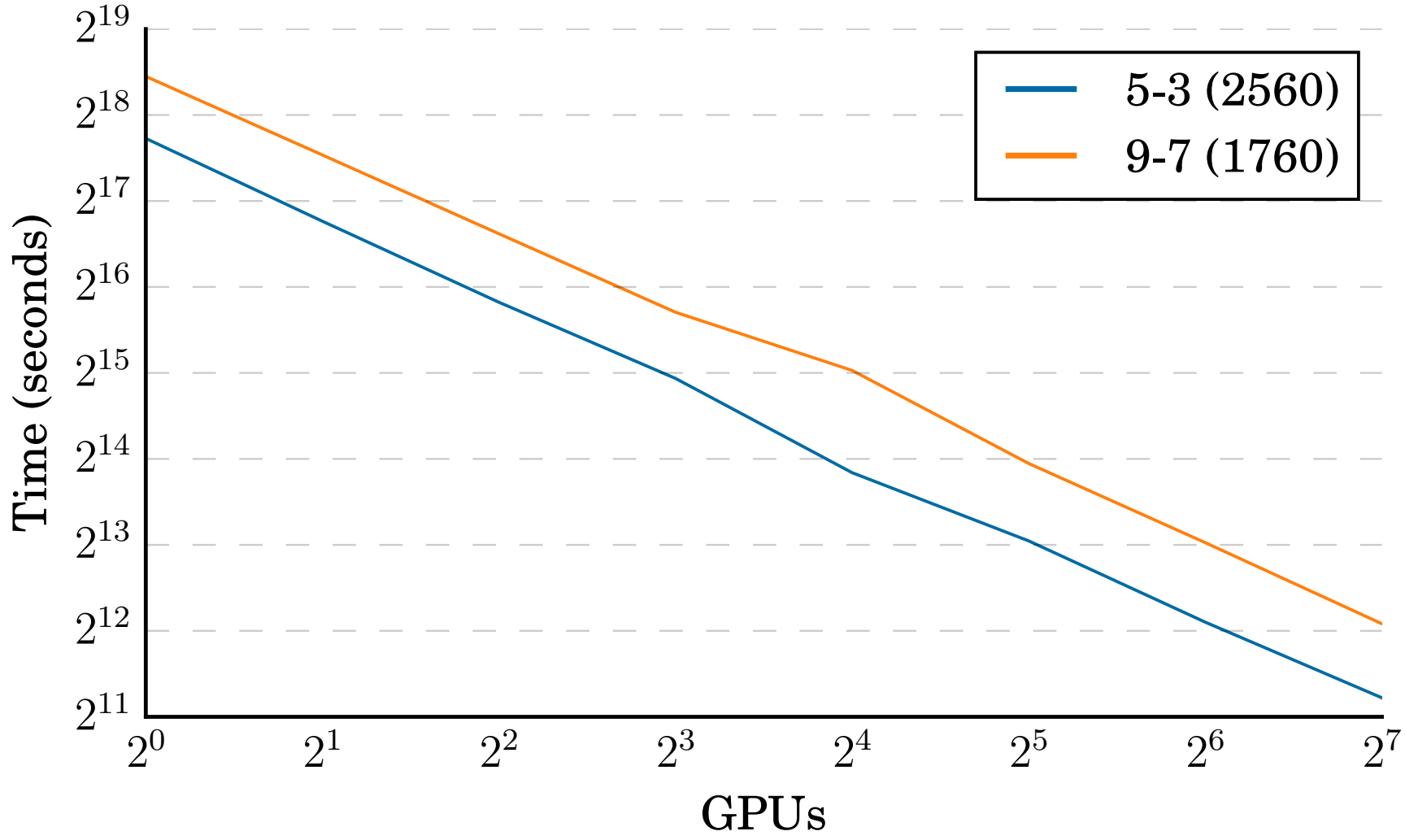






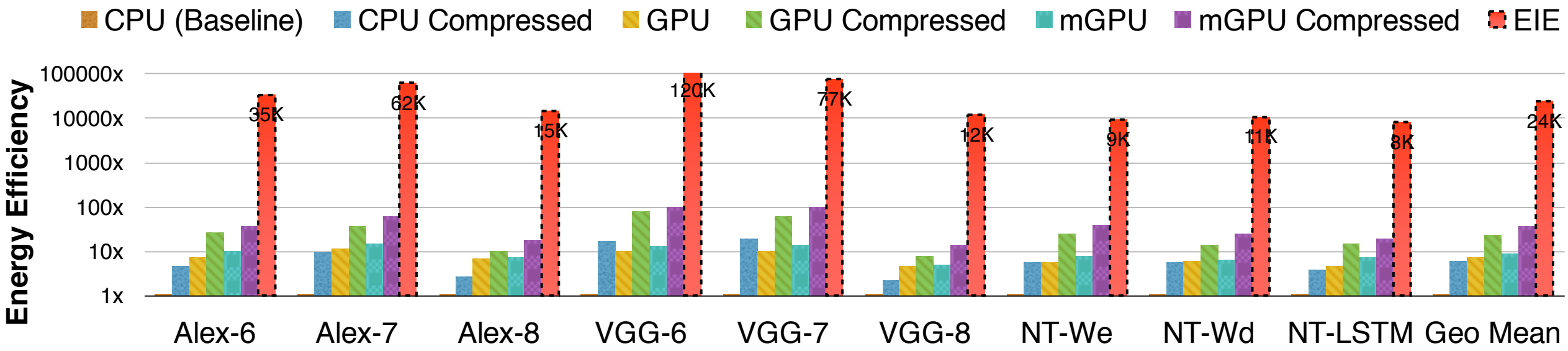


Data Parallelism Can get another 128x (256,000,000x) More with Model and Hyper-Parameter Parallelism



Baidu, Deep Speech 2: End-to-End Speech Recognition in English and Mandarin, 2015

Special-Purpose Hardware Can Give another 100x (25,000,000,000x) Mostly from localizing memory



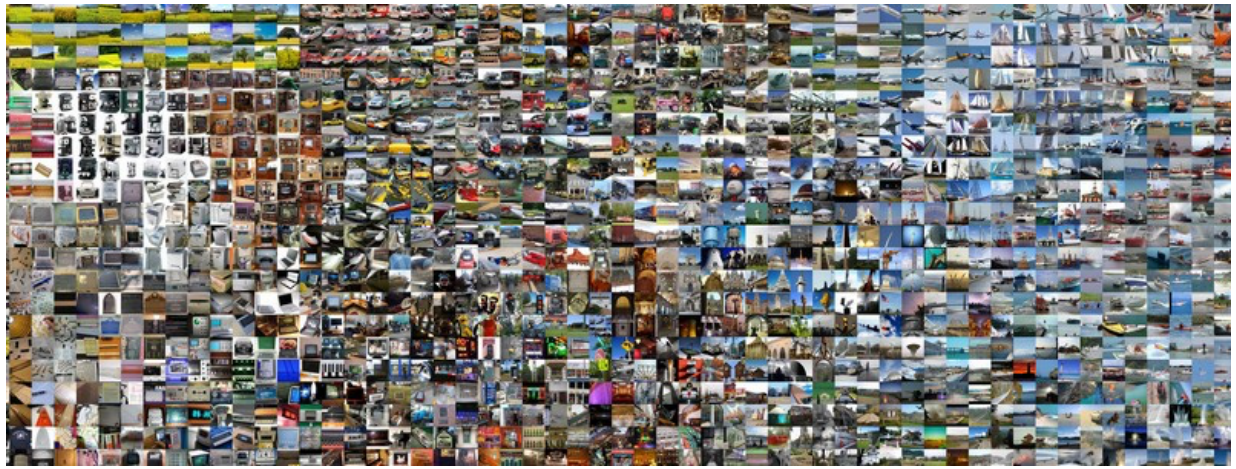
Accelerate the best algorithms
Prune the network
Compress the network
FFT convolutions

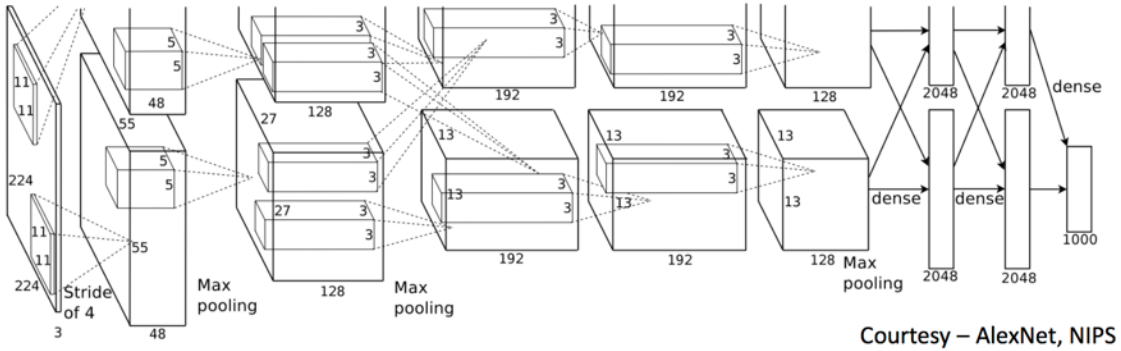
So what should you do?

- For training use clusters of 8-16GPUs
  - Best perf, perf/W, perf/$, and memory bandwidth
  - Easy parallelism
- For inference in the data center use single GPUs
  - Tesla M4 and M40
- For inference in mobile devices (Automotive, IoT)
  - Use a TX1 (11.5x perf/W of CPU)

- For the absolute best performance and efficiency use an ASIC
  - But make sure the model fits (memory limited ASICs no better than GPU)
  - And that your algorithm isn't going to change

Thank You





Courtesy – AlexNet, NIPS 2012