Network and Hardware Co-Design

MICRO Tutorial (2016)

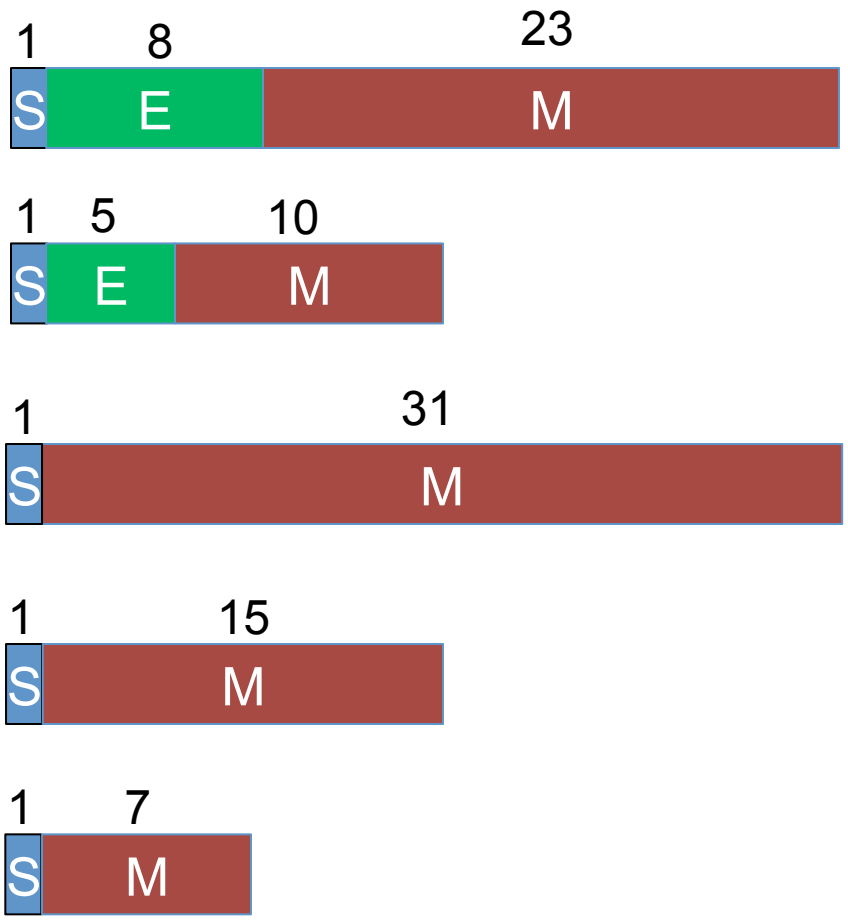
Website: http://eyeriss.mit.edu/tutorial.html

Joel Emer, Vivienne Sze, Yu-Hsin Chen

Network Optimization

- <u>Reduce precision</u> of operations and operands
  - Fixed and Floating point
  - Bit-width
- <u>Reduce number</u> of operations and storage of weights
  - Compression
  - Pruning
  - Network Architectures

Number Representation

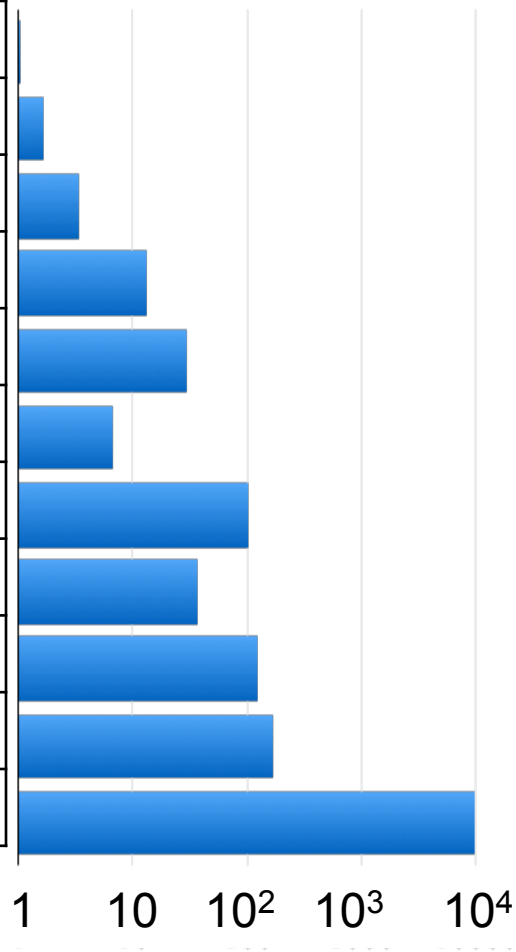


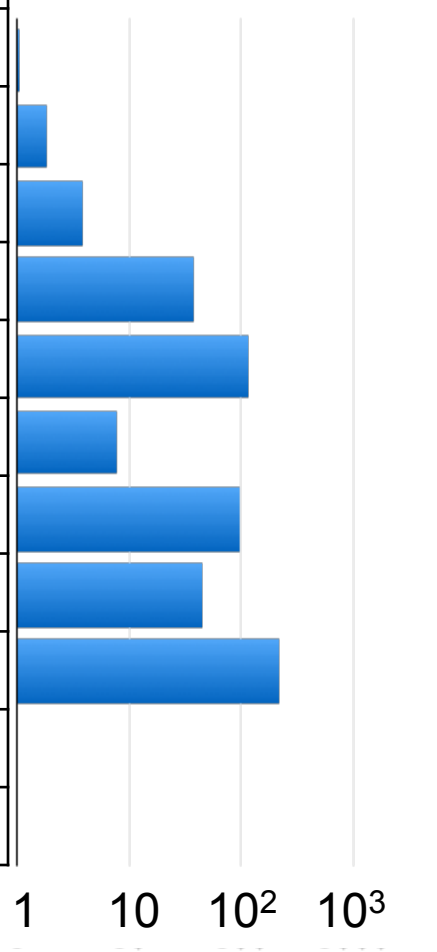
	Range	Accuracy
FP32	$10^{-38} – 10^{38}$	.000006%
FP16	$6x10^{-5}$ - $6x10^{4}$	.05%
Int32	$0 – 2x10^{9}$	½
Int16	$0 – 6x10^{4}$	½
Int8	0 – 127	½

Image Source: B. Dally

Cost of Operations

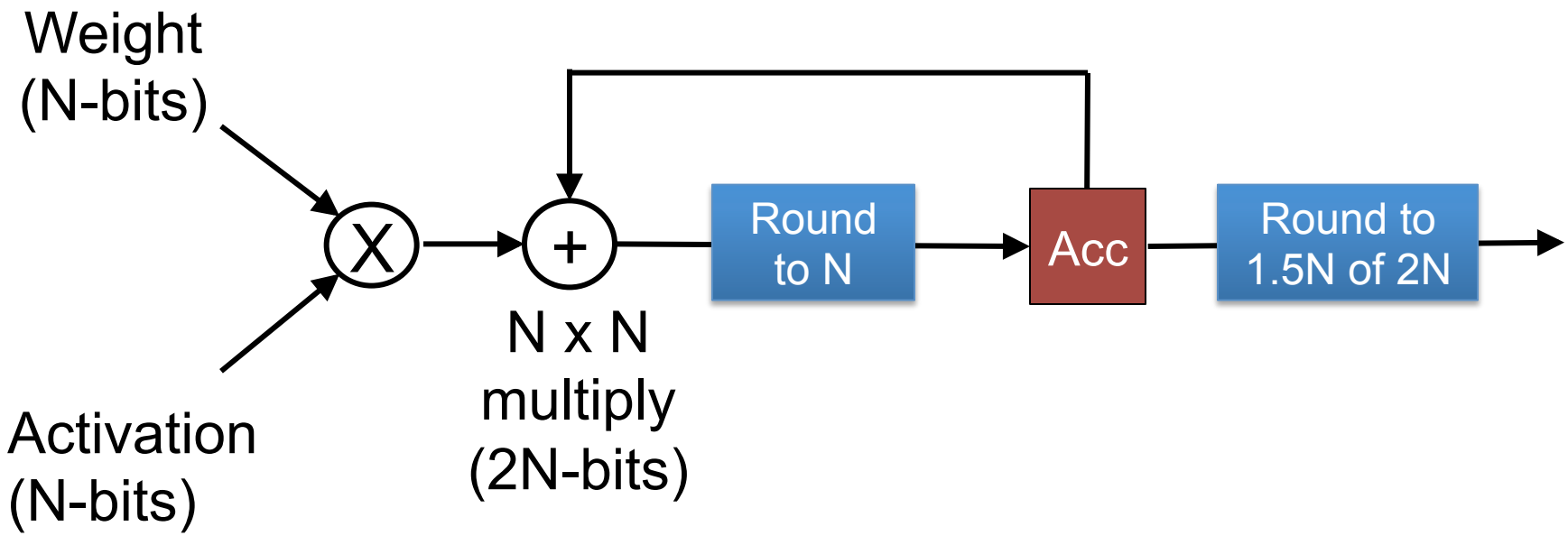
Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





[Horowitz, “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014]

N-bit Precision



Methods to Reduce Bits

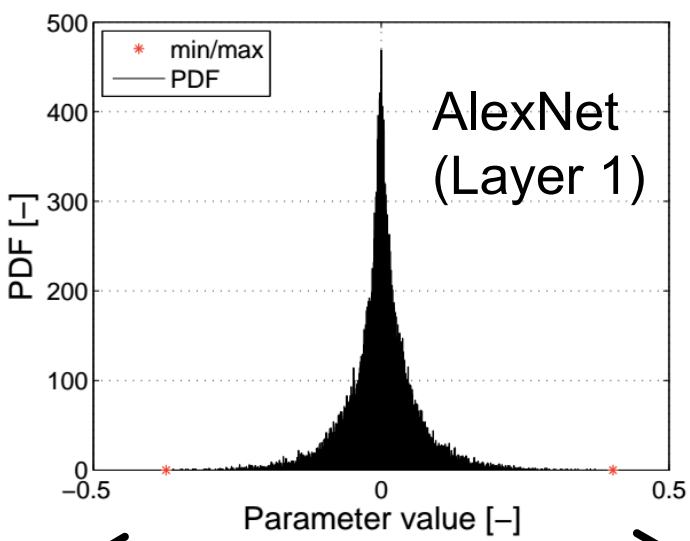
- **Quantization/Rounding**
- **Dynamic Fixed Point**
  - **Rescale and Reduce bits**
- **Fine-tuning: Retrain Weights**

Example: 16-bit → 8-bits

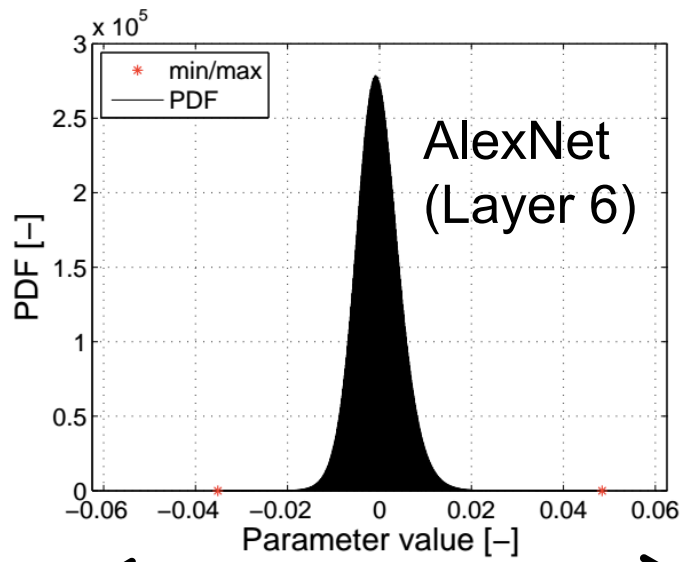
$2^{11} + 2^{9} + 2^{6} + 2^{1} = 2626$ (overflow)

0	1	0	0	1	0	1	0	0	1	0	0	0	0	1	0

$2^{10} + 2^{7} + 2^{5} + 2^{2} + 2^{0} = 1189$



Dynamic range = 1



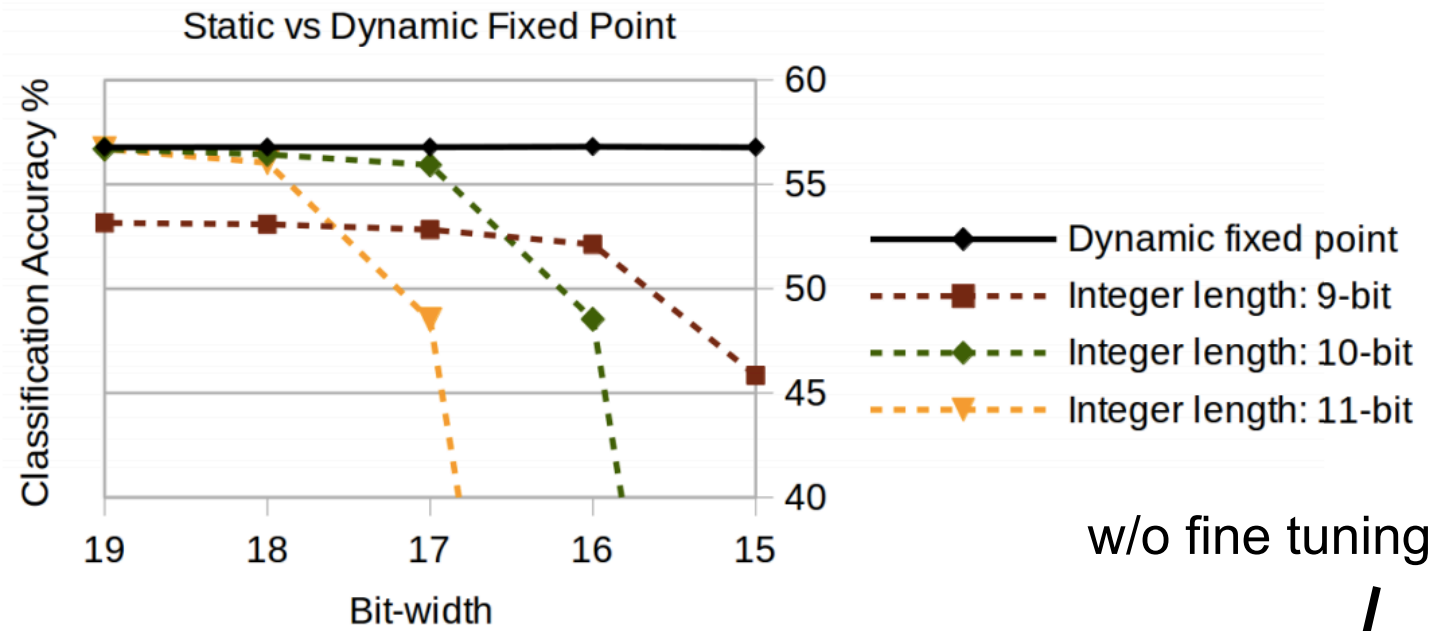
Dynamic range = 0.125

Image Source: Moons et al, WACV 2016

Batch normalization important to 'center' dynamic range

Impact on Accuracy

Top-1 accuracy on of CaffeNet on ImageNet



w/o fine tuning

	Layer outputs	CONV parameters	FC parameters	32-bit floating point baseline	Fixed point accuracy
LeNet (Exp 1)	4-bit	4-bit	4-bit	99.1%	99.0% (98.7%)
LeNet (Exp 2)	4-bit	2-bit	2-bit	99.1%	98.8% (98.0%)
Full CIFAR-10	8-bit	8-bit	8-bit	81.7%	81.4% (80.6%)
SqueezeNet top-1	8-bit	8-bit	8-bit	57.7%	57.1% (55.2%)
CaffeNet top-1	8-bit	8-bit	8-bit	56.9%	56.0% (55.8%)
GoogLeNet top-1	8-bit	8-bit	8-bit	68.9%	66.6% (66.1%)

[Gysel et al., Ristretto, ICLR 2016]

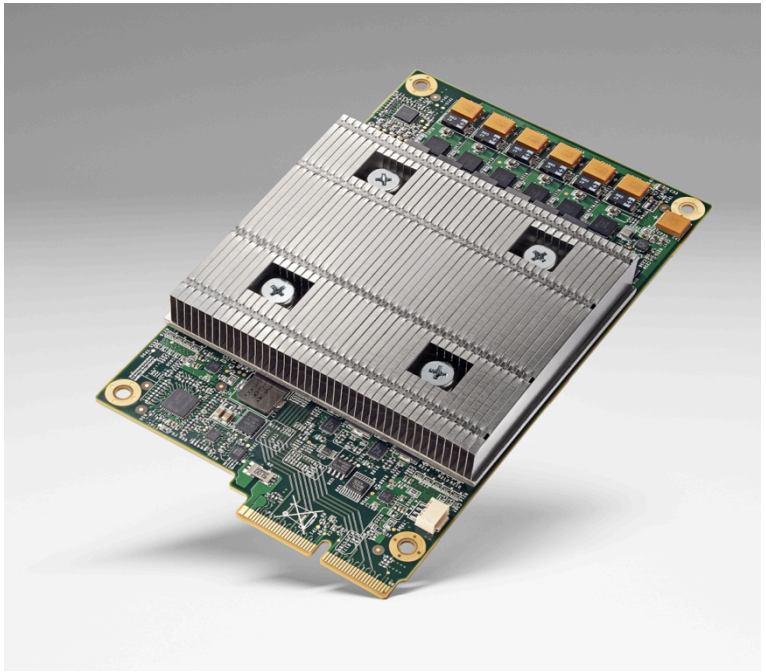
Google's Tensor Processing Unit (TPU)

" With its TPU Google has seemingly focused on delivering the data really quickly by **cutting down on precision**. Specifically, it doesn't rely **on floating point precision like a GPU**

….

Instead the chip uses integer math…TPU used **8-bit integer**."

- Next Platform (May 19, 2016)



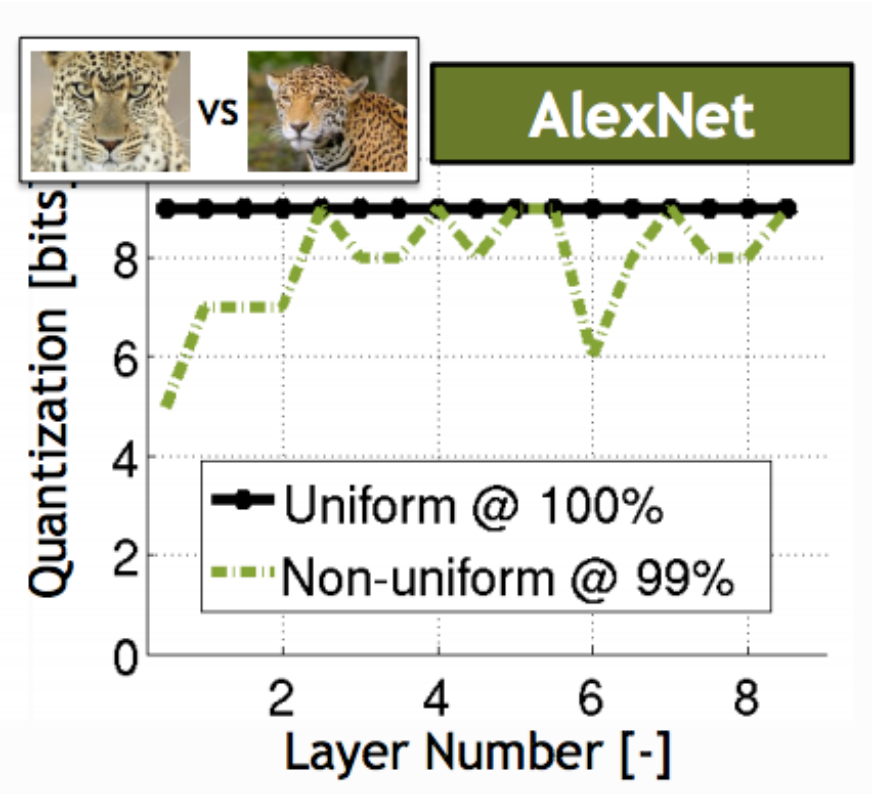
Nvidia PASCAL

"New half-precision, **16-bit floating point instructions deliver over 21 TeraFLOPS** for unprecedented training performance. **With 47 TOPS (tera-operations per second) of performance, new 8-bit integer instructions** in Pascal allow AI algorithms to deliver real-time responsiveness for deep learning inference."

– Nvidia.com (April 2016)

Precision Varies from Layer to Layer

Tolerance	Bits per layer (I+F)
AlexNet (F=0)	
1%	10-8-8-8-8-8-6-4
2%	10-8-8-8-8-8-5-4
5%	10-8-8-8-7-7-5-3
10%	9-8-8-8-7-7-5-3



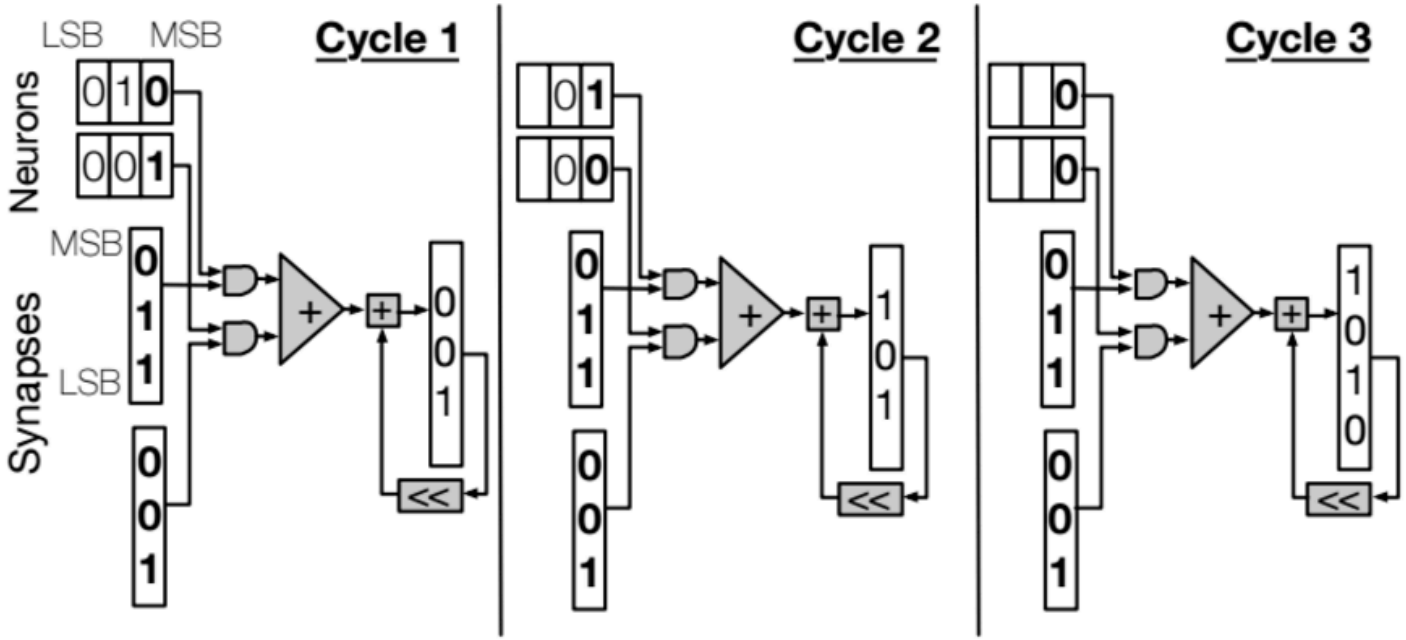
[Judd et al., ArXiv 2016]

[Moons et al., WACV 2016]

Bitwidth Scaling (Speed)

Bit-Serial Processing: Reduce Bit-width → Skip Cycles
Speed up of 2.24x vs. 16-bit fixed

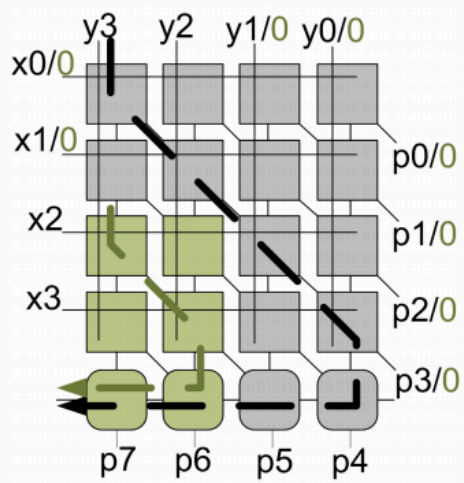
$$\sum_{i=0}^{N_i-1} s_i \times n_i = \sum_{i=0}^{N_i-1} s_i \times \sum_{b=0}^{P-1} n_i^b \times 2^b = \sum_{b=0}^{P-1} 2^b \times \sum_{i=0}^{N_i-1} n_i^b \times S_i$$

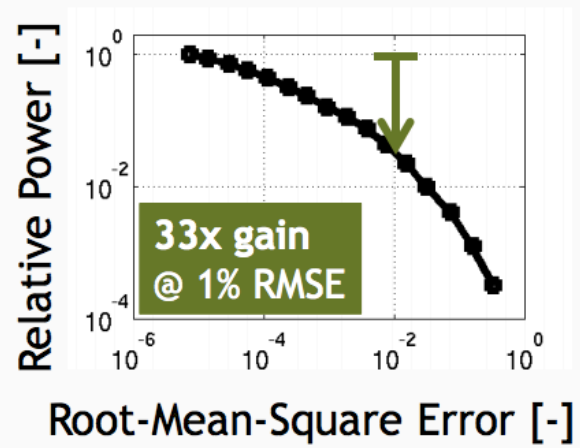


[Judd et al., Stripes, CAL 2016]

Bitwidth Scaling (Power)

Reduce Bit-width → Shorter Critical Path → Reduce Voltage

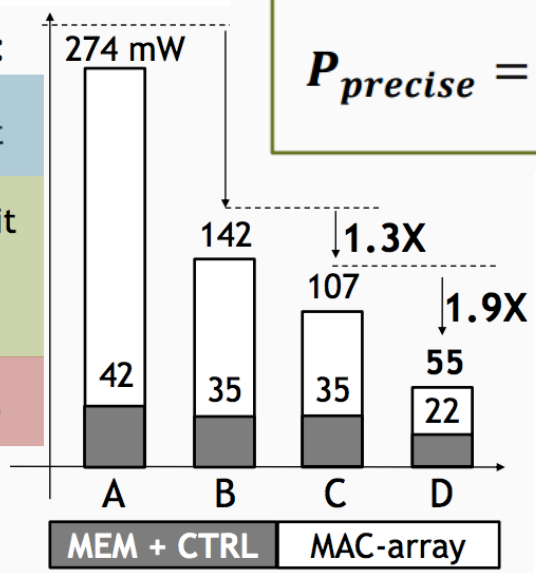




$$P_{precise} = \alpha C f V^2 \quad \Rightarrow \quad P_{imprecise} = \frac{\alpha}{k_1} C f \left(\frac{V}{k_2}\right)^2$$

AlexNet Layer 2 example:

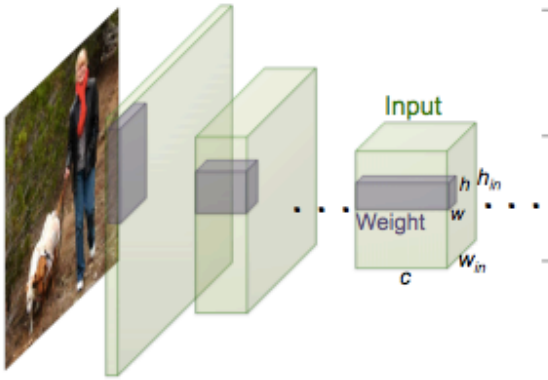
A. 2D-baseline	@ 16 bit
B. Precision-Scaling	@ 7-7 bit
C. Voltage-Scaling	@ 0.9 V
D. Sparse operation guarding	



Power reduction of 2.56x vs. 16-bit fixed On AlexNet Layer 2

[Moons et al., VLSI 2016]

Binary Nets



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

Classification Accuracy(%)									
Binary-Weight				Binary-Input-Binary-Weight				Full-Precision	
BWN		BC[11]		XNOR-Net		BNN[11]		AlexNet[1]	
Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
56.8	**79.4**	35.4	61.0	**44.2**	**69.2**	27.9	50.42	56.6	80.2

BinaryConnect (BC) = [Courbariaux et al., ArXiv 2015]
Binary Neural Networks (BNN) = [Courbariaux et al., ArXiv 2016]

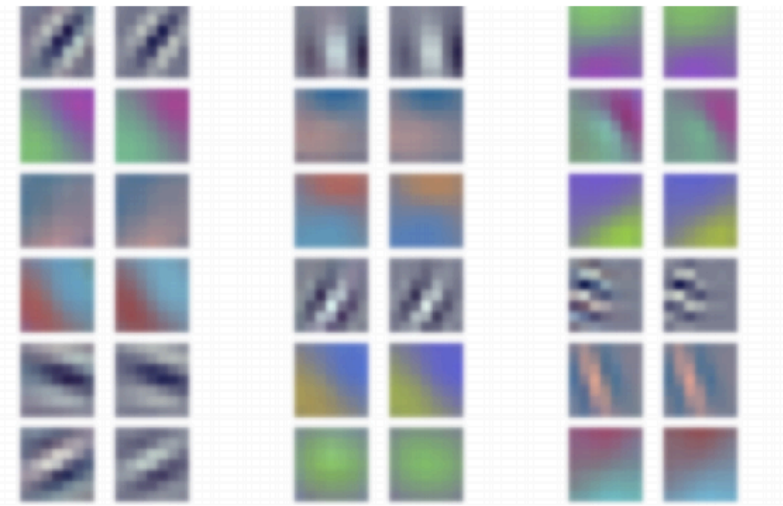
[Rastegari et al., BWN & XNOR-Net, ECCV 2016]

Reduce Number of Ops and Weights

- **Network Compression**
  - Low Rank Approximation
  - Weight Sharing and Vector Quantization
- **Pruning**
  - Weights
  - Activations
- **Network Architectures**

Low Rank Approximation

- **Low Rank approximation**
  - **Tensor decomposition based on singular value decomposition (SVD)**
  - **Filter Clustering with modified K-means**
  - **Fine Tuning**



- **Speed up by 1.6 – 2.7x on CPU/GPU for CONV1, CONV2 layers**
- **Reduce size by 5 - 13x for FC layer**
- **< 1% drop in accuracy**

[Denton et al., NIPS 2014]

Low Rank Approximation on Phone

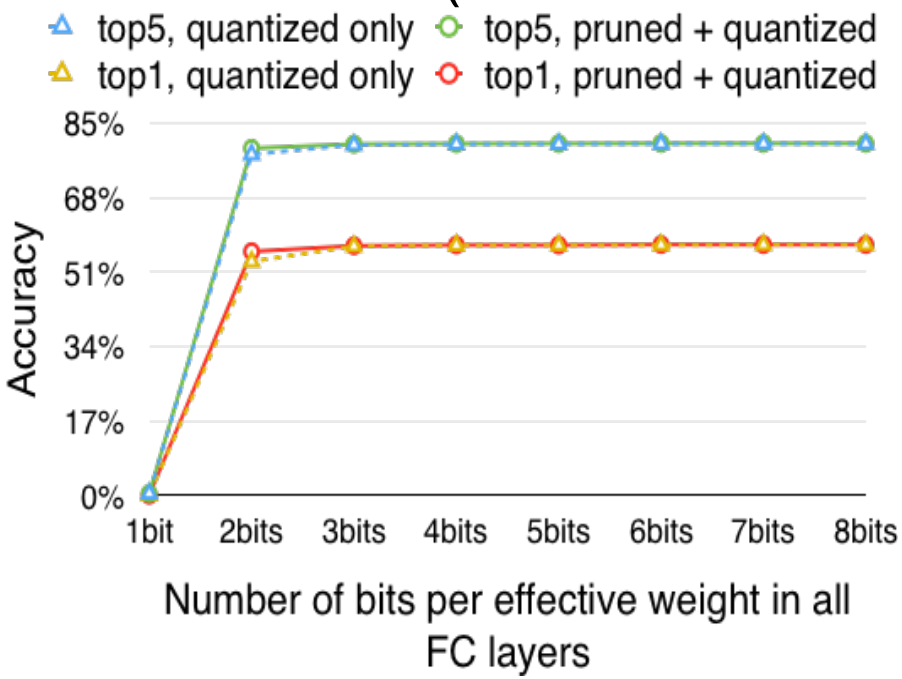
- **Rank selection per Layer**
- **Tucker Decomposition (extension of SVD)**
- **Fine tuning**

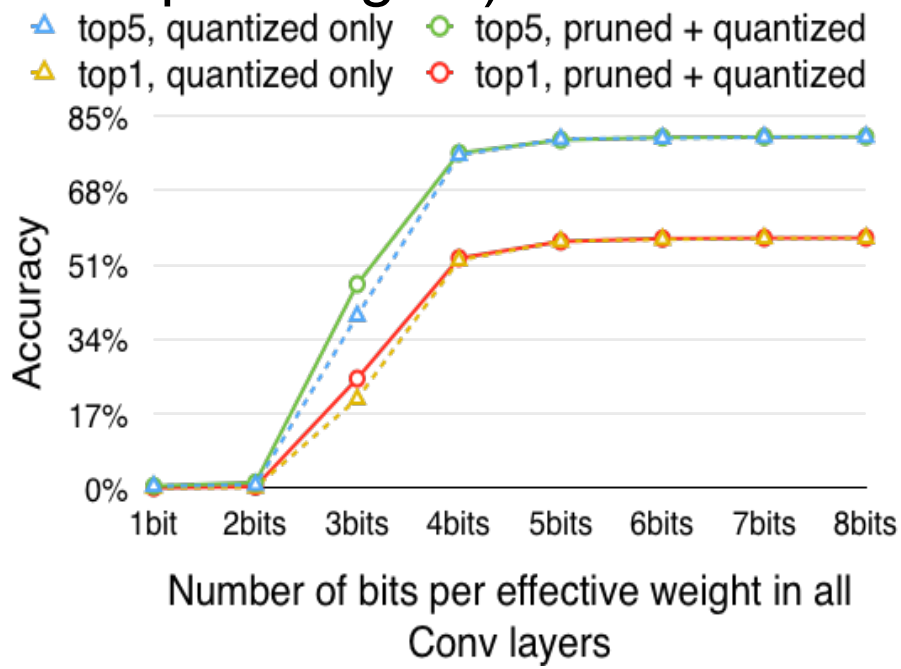
Model	Top-5	Weights	FLOPs	S6		Titan X
AlexNet	80.03	61M	725M	117ms	245mJ	0.54ms
*AlexNet**	78.33	11M	272M	43ms	72mJ	0.30ms
(imp.)	(-1.70)	(×5.46)	(×2.67)	(×2.72)	(×3.41)	(×1.81)
VGG-S	84.60	103M	2640M	357ms	825mJ	1.86ms
*VGG-S**	84.05	14M	549M	97ms	193mJ	0.92ms
(imp.)	(-0.55)	(×7.40)	(×4.80)	(×3.68)	(×4.26)	(×2.01)
GoogLeNet	88.90	6.9M	1566M	273ms	473mJ	1.83ms
*GoogLeNet**	88.66	4.7M	760M	192ms	296mJ	1.48ms
(imp.)	(-0.24)	(×1.28)	(×2.06)	(×1.42)	(×1.60)	(×1.23)
VGG-16	89.90	138M	15484M	1926ms	4757mJ	10.67ms
*VGG-16**	89.40	127M	3139M	576ms	1346mJ	4.58ms
(imp.)	(-0.50)	(×1.09)	(×4.93)	(×3.34)	(×3.53)	(×2.33)

[Kim et al., ICLR 2016]

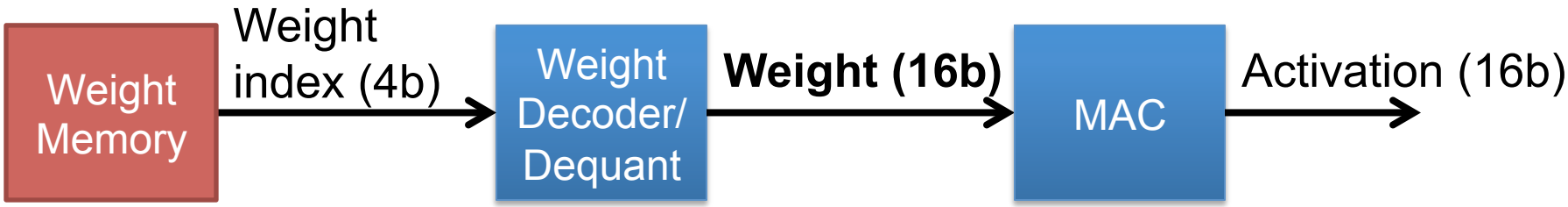
Weight Sharing + Vector Quantization

Trained Quantization: Weight Sharing via K-means clustering (reduce number of unique weights)





Reduce Bits for Storage (compute still 16-bits)

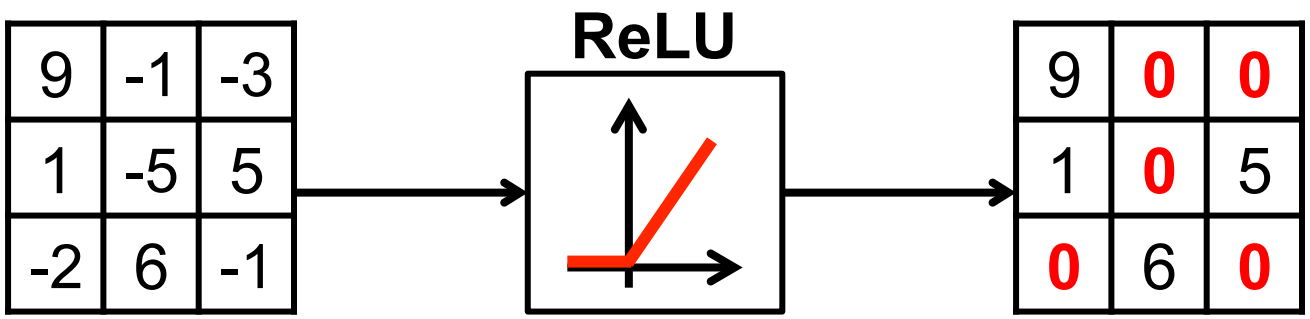


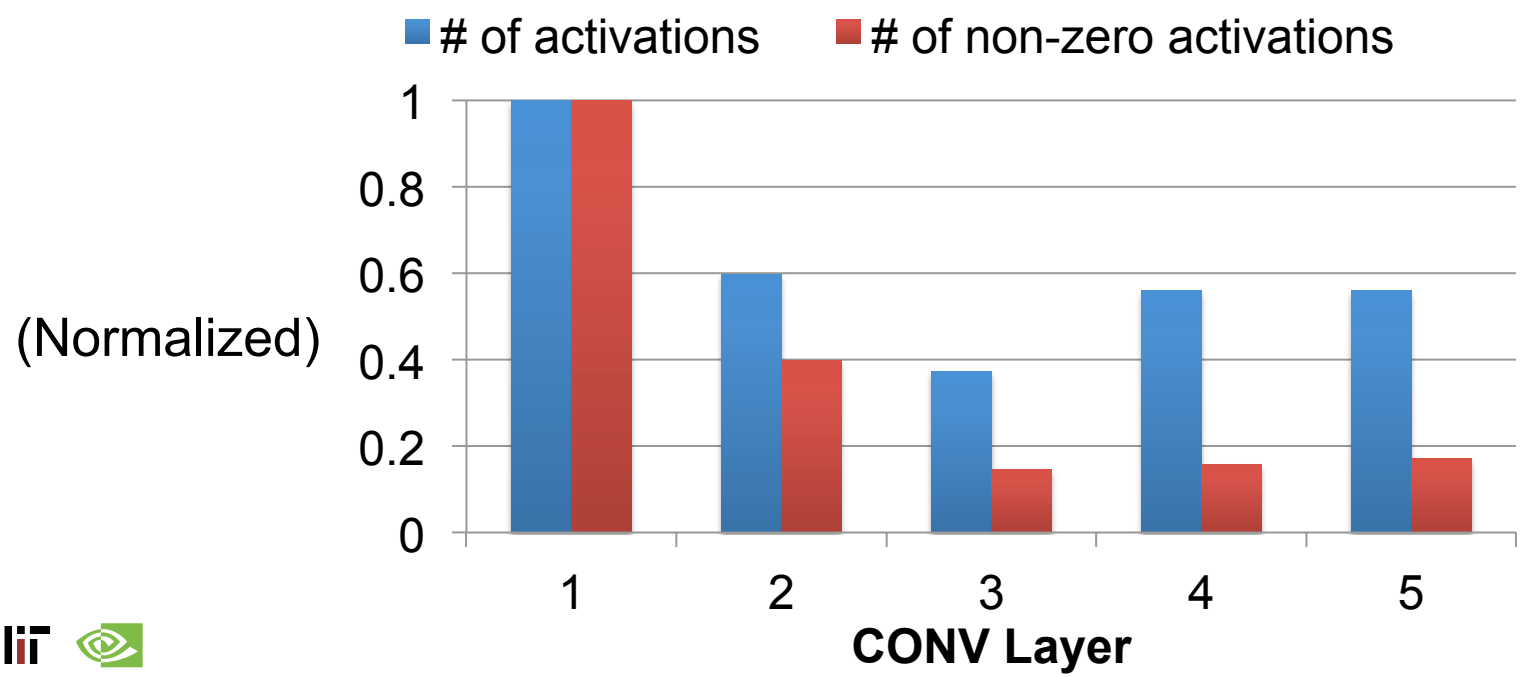
[Han et al., Deep Compression, ICLR 2016]

Exploit Data Statistics

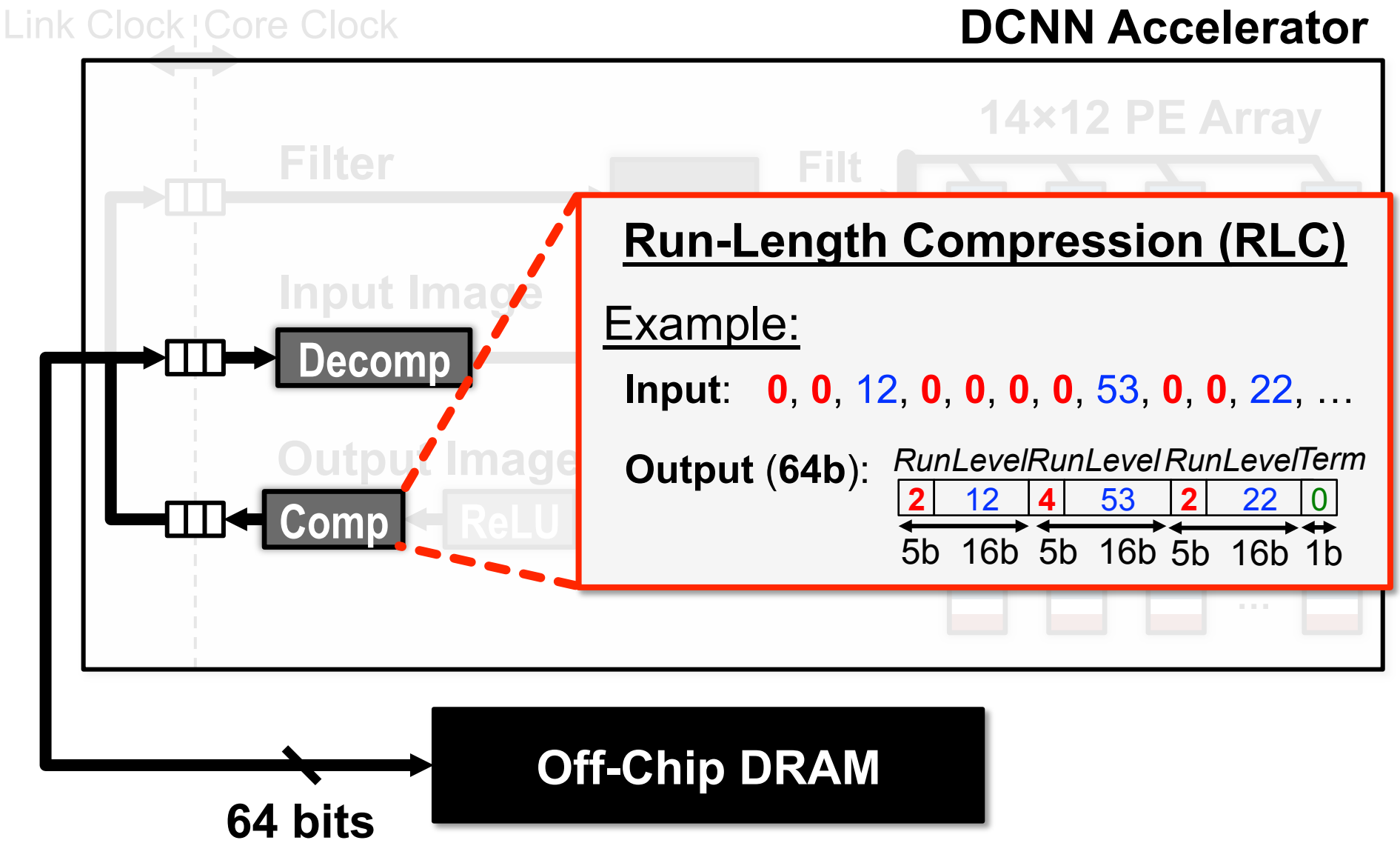
Sparsity in Fmaps

Many **zeros** in **output fmaps** after **ReLU**



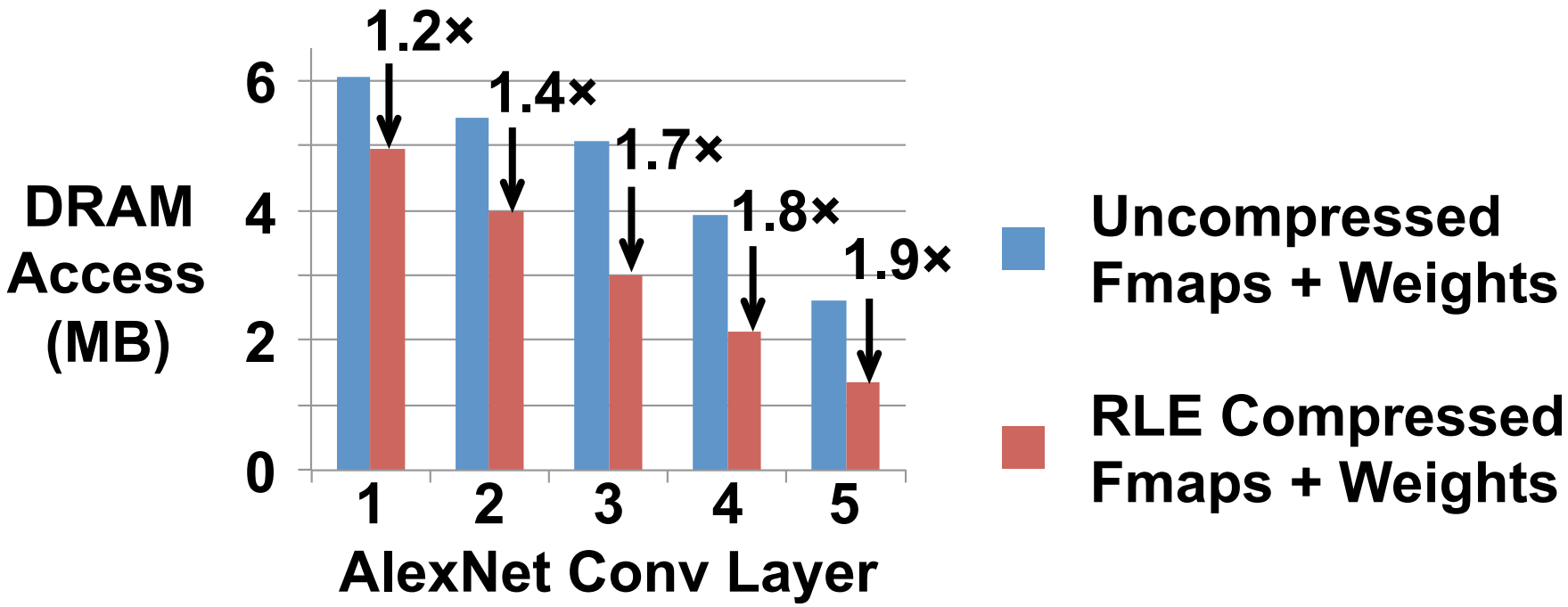


I/O Compression in Eyeriss



[Chen et al., ISSCC 2016]

Compression Reduces DRAM BW

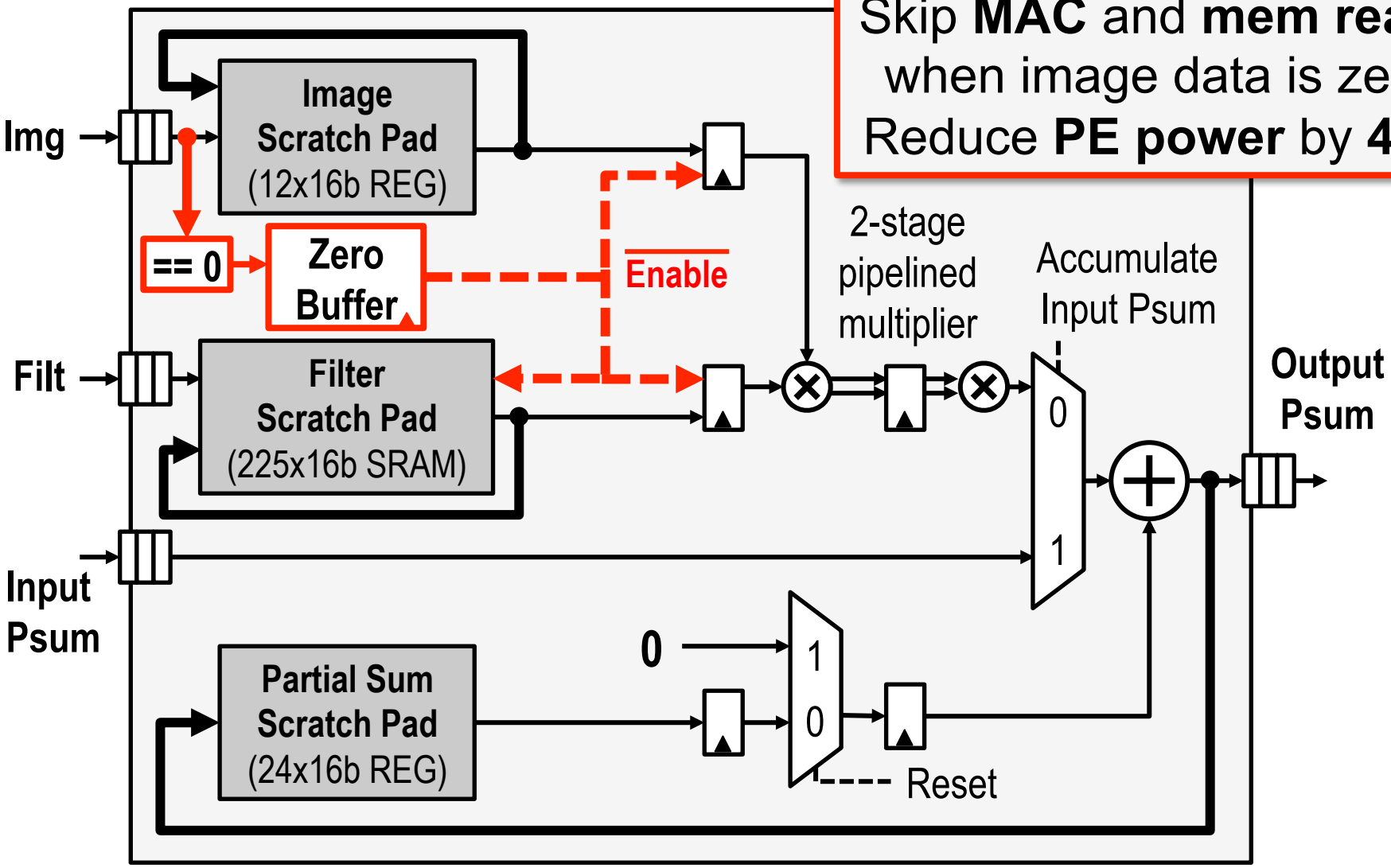


Simple RLC within 5% - 10% of theoretical entropy limit

[Chen et al., ISSCC 2016]

Data Gating / Zero Skipping in Eyeriss

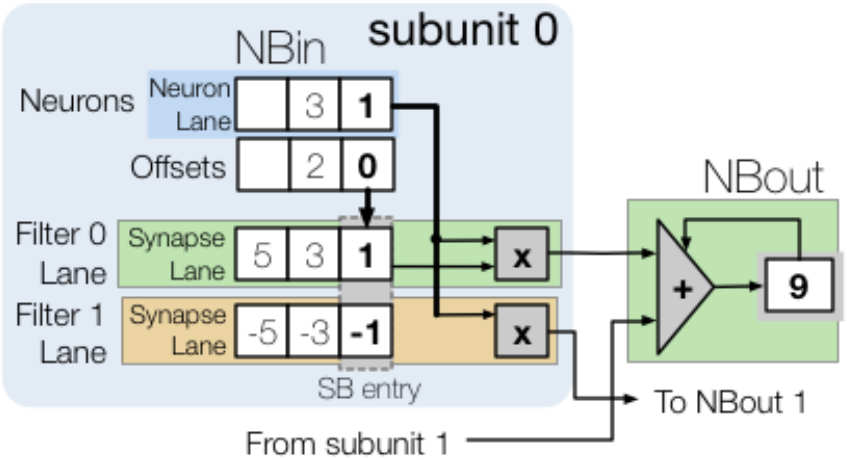
Skip **MAC** and **mem reads** when image data is zero. Reduce **PE power** by **45%**

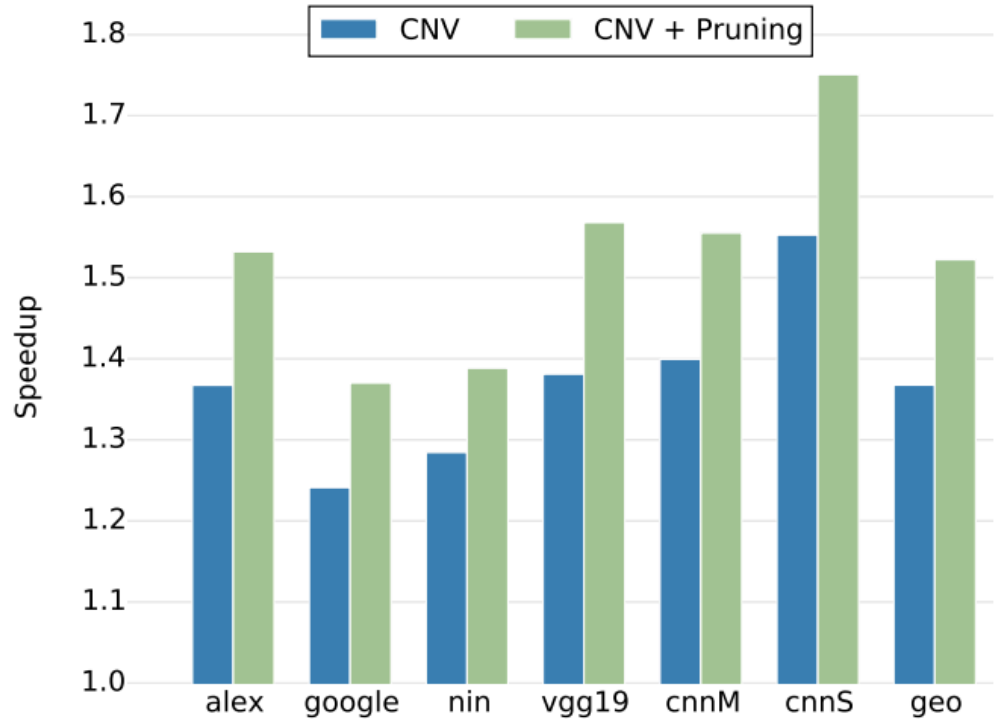


[Chen et al., ISSCC 2016]

Cnvlutin

- **Process Convolution Layers**
- **Built on top of DaDianNao (4.49% area overhead)**
- **Speed up of 1.37x (1.52x with activation pruning)**



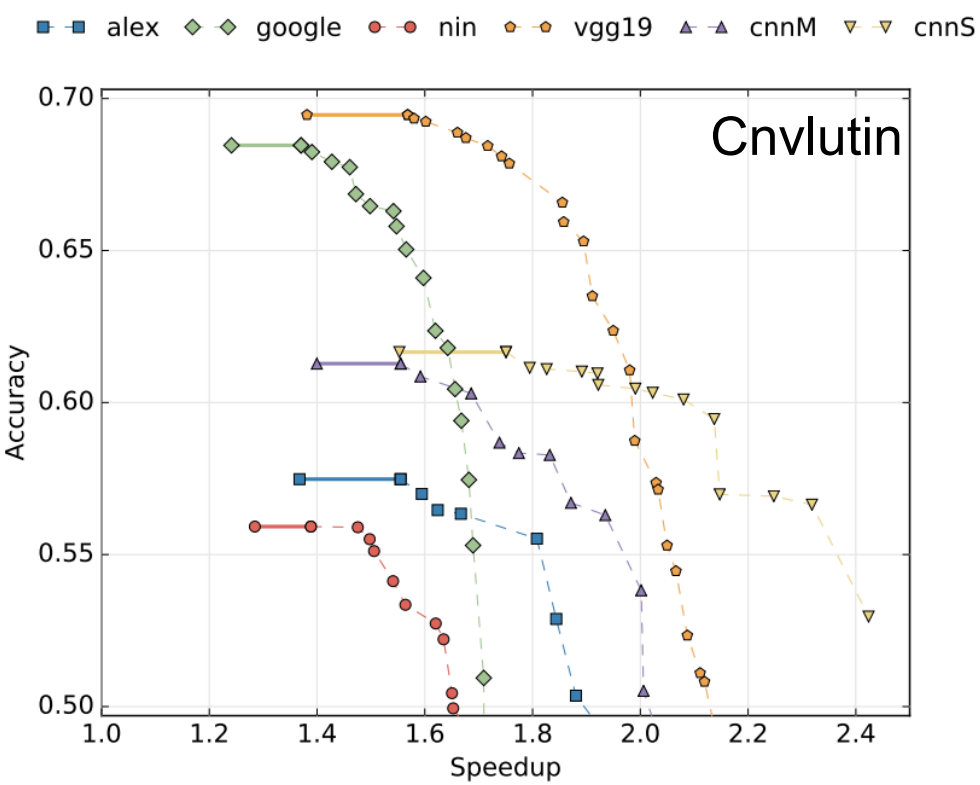


[Albericio et al., ISCA 2016]

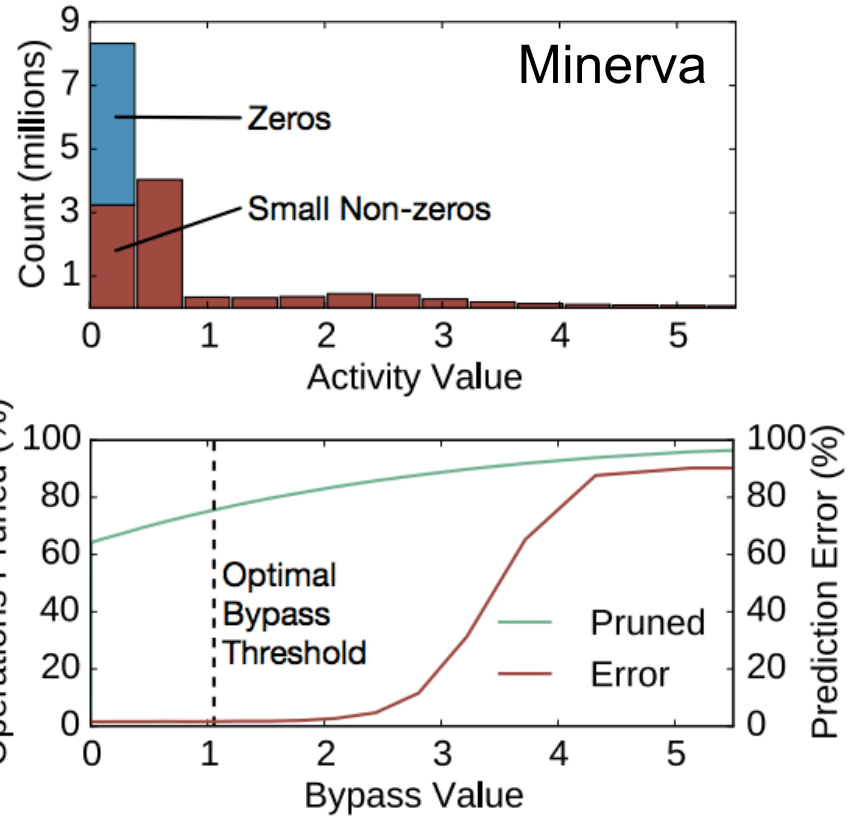
Pruning Activations

Remove small activation values

Speed up 11% (ImageNet)



Reduce power 2x (MNIST)



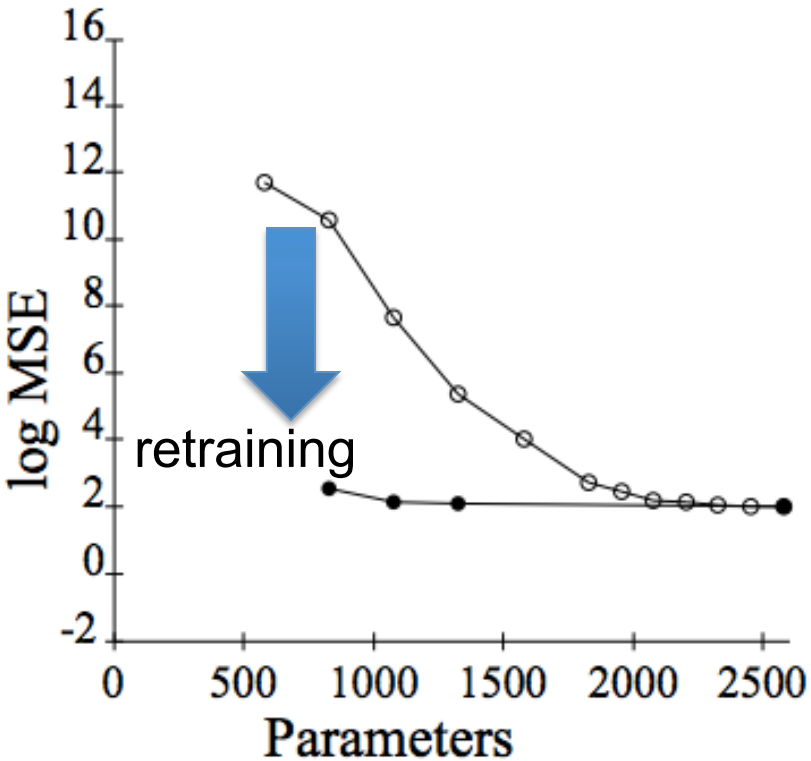
[Albericio et al., ISCA 2016]

[Reagen et al., ISCA 2016]

Pruning – Make Weights Sparse

- **Optimal Brain Damage**

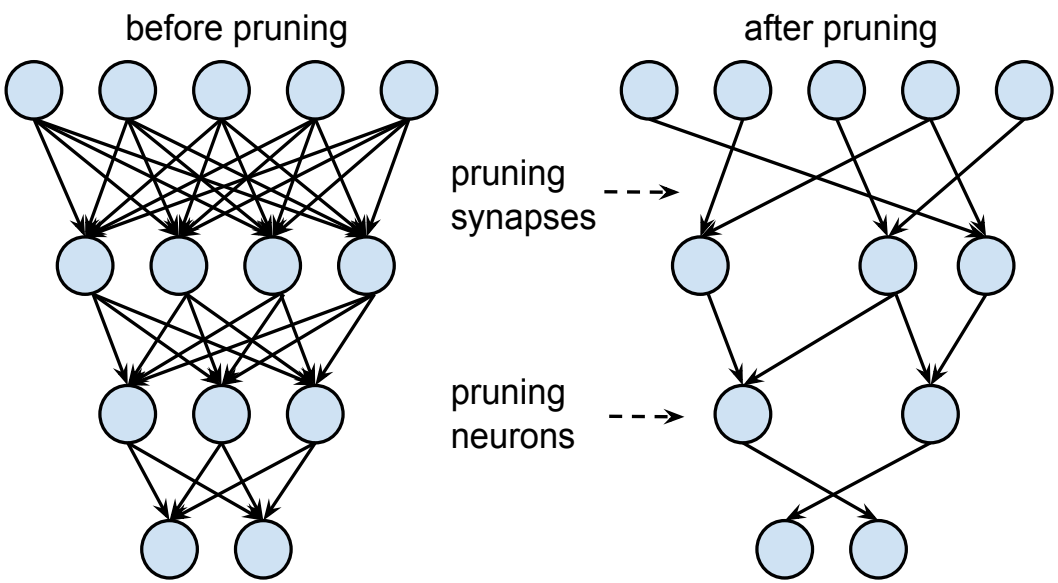
1. Choose a reasonable network architecture
2. Train network until reasonable solution obtained
3. Compute the second derivative for each weight
4. Compute saliencies (i.e. impact on training error) for each weight
5. Sort weights by saliency and delete low-saliency weights
6. Iterate to step 2

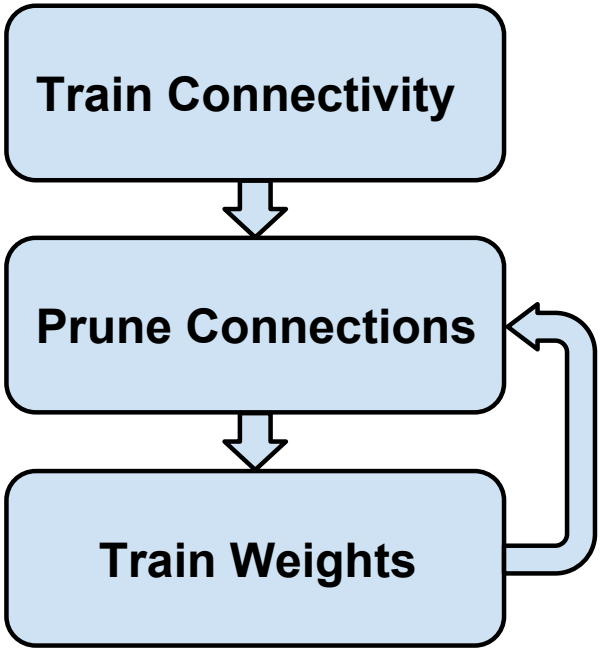


[Lecun et al., NIPS 1989]

Pruning – Make Weights Sparse

Prune based on magnitude of weights

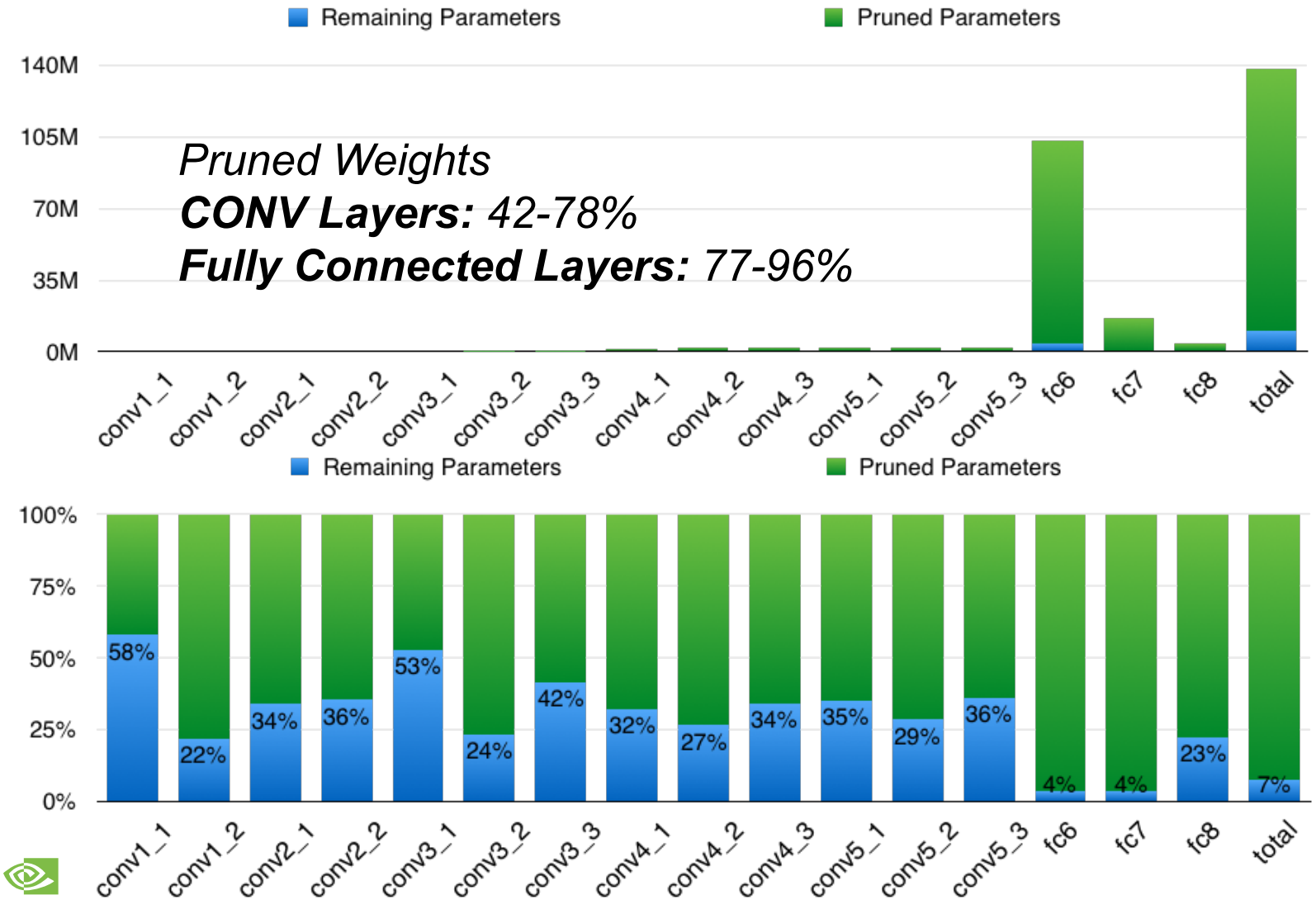




[Han et al., NIPS 2015]

Pruning of VGG-16

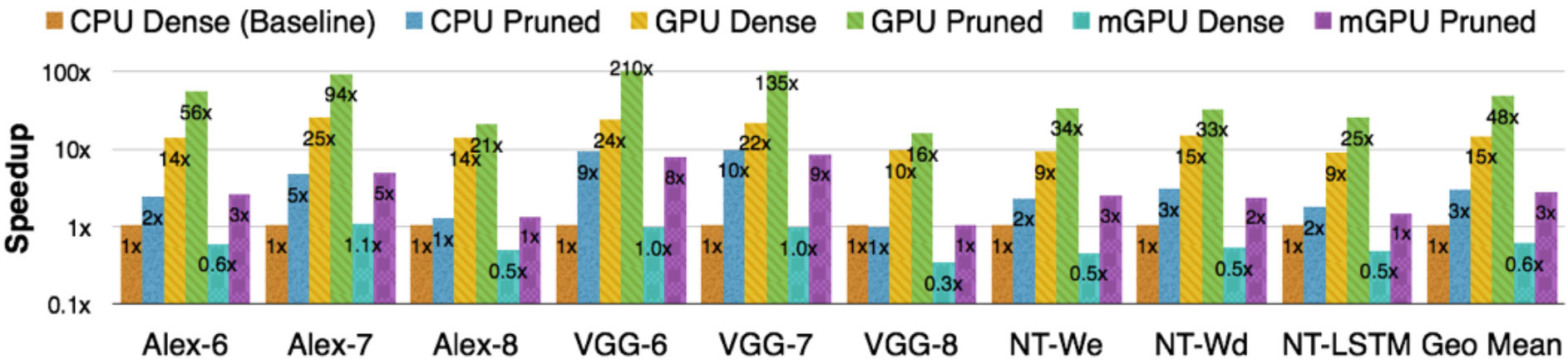
Pruning has most impact on Fully Connected Layers



Speed up of Weight Pruning on CPU/GPU

On Fully Connected Layers

Average Speed up of 3.2x on GPU, 3x on CPU, 5x on mGPU



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

Batch size = 1

[Han et al., NIPS 2015]

Energy-Aware Pruning

- # of Weights alone is not a good metric for energy
  - Example (AlexNet):
    - # of Weights (FC Layer) > # of Weights (CONV layer)
    - Energy (FC Layer) < Energy (CONV layer)
- Us energy evaluation method to estimate DNN energy
  - Account for data movement
- Prune based on energy rather than weights
  - Reduce <u>overall energy (ALL layers)</u> by 3.7x for AlexNet
    - 1.8x more efficient than previous magnitude-based approach
  - 1.6x energy reduction for GoogleNet

[Yang et al., ArXiv 2016]

Compression of Weights & Activations

- **Compress weights and fmaps between DRAM and accelerator**
- **Variable Length / Huffman Coding**

Example:

Value: **16'b0** → Compressed Code: {**1'b0**}

Value: **16'bx** → Compressed Code: {**1'b1**, **16'bx**}

- Tested on AlexNet → **2× overall BW Reduction**

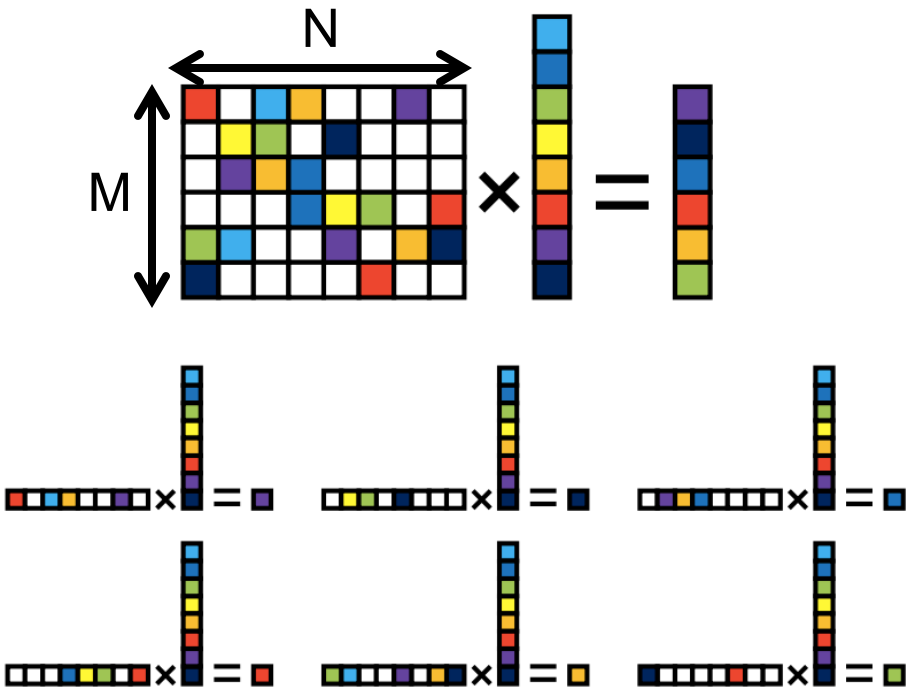
Layer	Filter / Image bits (0%)	Filter / Image BW Reduc.	IO / HuffIO (MB/frame)	Voltage (V)	MMACs/ Frame	Power (mW)	Real (TOPS/W)
General CNN	16 (0%) / 16 (0%)	1.0x		1.1	—	**288**	**0.3**
AlexNet l1	7 (21%) / 4 (29%)	1.17x / 1.3x	1 / 0.77	0.85	105	85	0.96
AlexNet l2	7 (19%) / 7 (89%)	1.15x / **5.8x**	3.2 / 1.1	0.9	224	55	1.4
AlexNet l3	8 (11%) / 9 (82%)	1.05x / 4.1x	6.5 / 2.8	0.92	150	77	0.7
AlexNet l4	9 (04%) / 8 (72%)	1.00x / 2.9x	5.4 / 3.2	0.92	112	95	0.56
AlexNet l5	9 (04%) / 8 (72%)	1.00x / 2.9x	3.7 / 2.1	0.92	75	95	0.56
Total / avg.	—	—	19.8 / **10**	—	—	**76**	**0.94**
LeNet-5 l1	3 (35%) / 1 (87%)	1.40x / 5.2x	0.003 / 0.001	0.7	0.3	25	1.07
LeNet-5 l2	4 (26%) / 6 (55%)	1.25x / 1.9x	0.050 / 0.042	0.8	1.6	35	1.75
Total / avg.	—	—	0.053 / **0.043**	—	—	**33**	**1.6**

[Moons et al., VLSI 2016; Han et al., ICLR 2016]

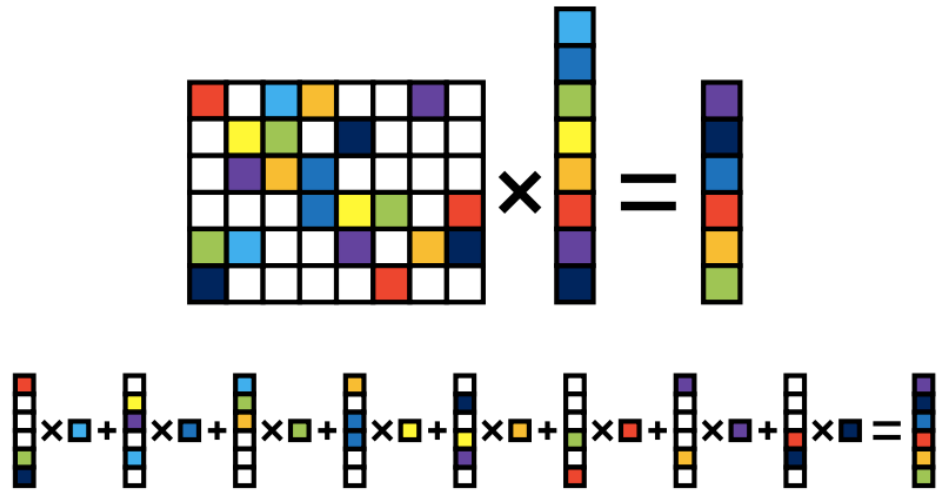
Sparse Matrix-Vector DSP

- **Use CSC rather than CSR for SpMxV**

Compressed Sparse Row (CSR)



Compressed Sparse Column (CSC)

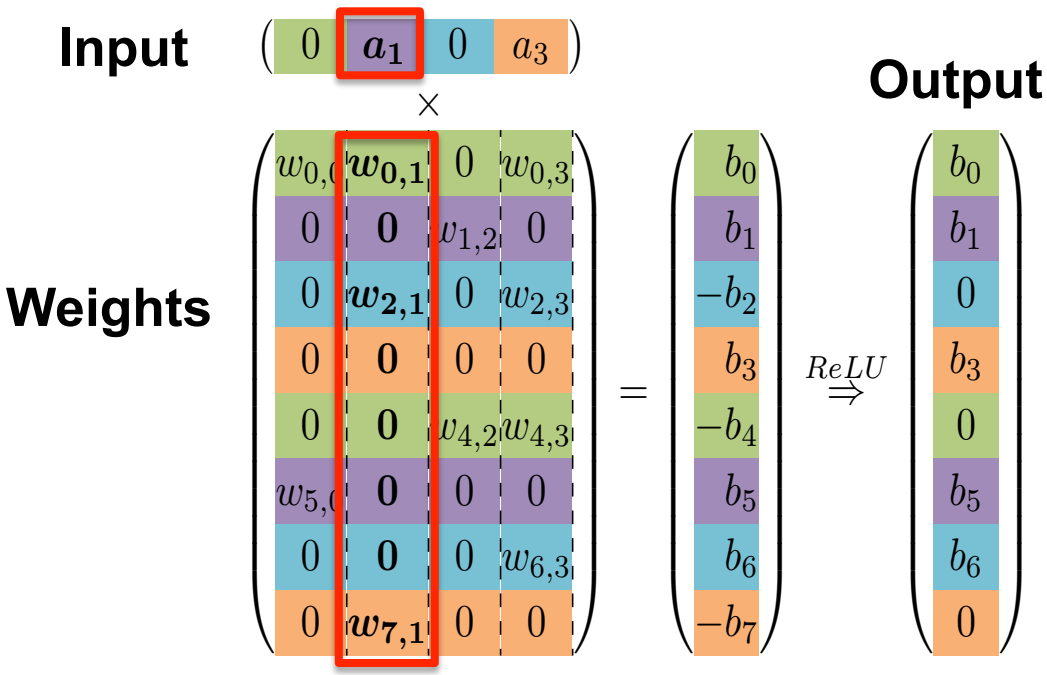


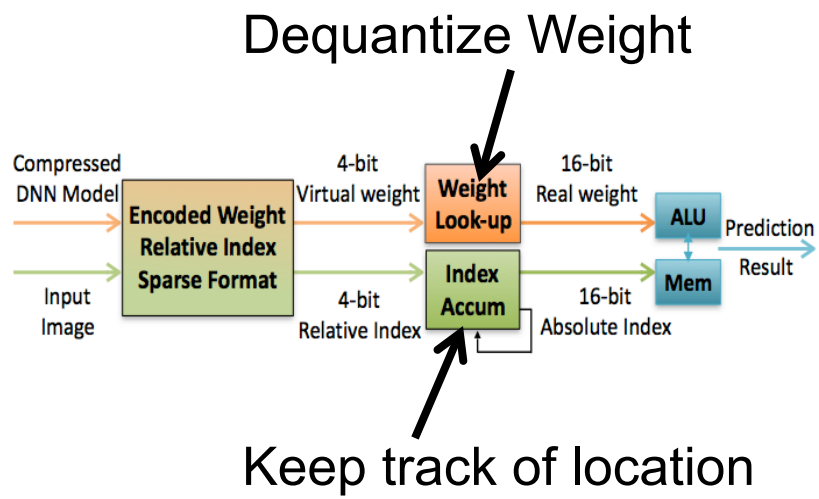
Reduce memory bandwidth by 2x (when not M >> N)

[Dorrance et al., FPGA 2014]

EIE: A Sparse Linear Algebra Engine

- **Process Fully Connected Layers (after Deep Compression)**
- **Store weights column-wise in Run Length format**
  - **Non-zero weights, Run-length of zeros**
  - **Start location of each column since variable length**
- **Read relative column when input is non-zero**





[Han et al., ISCA 2016]

Network Architecture

Reduce size and computation with 1x1 Filter

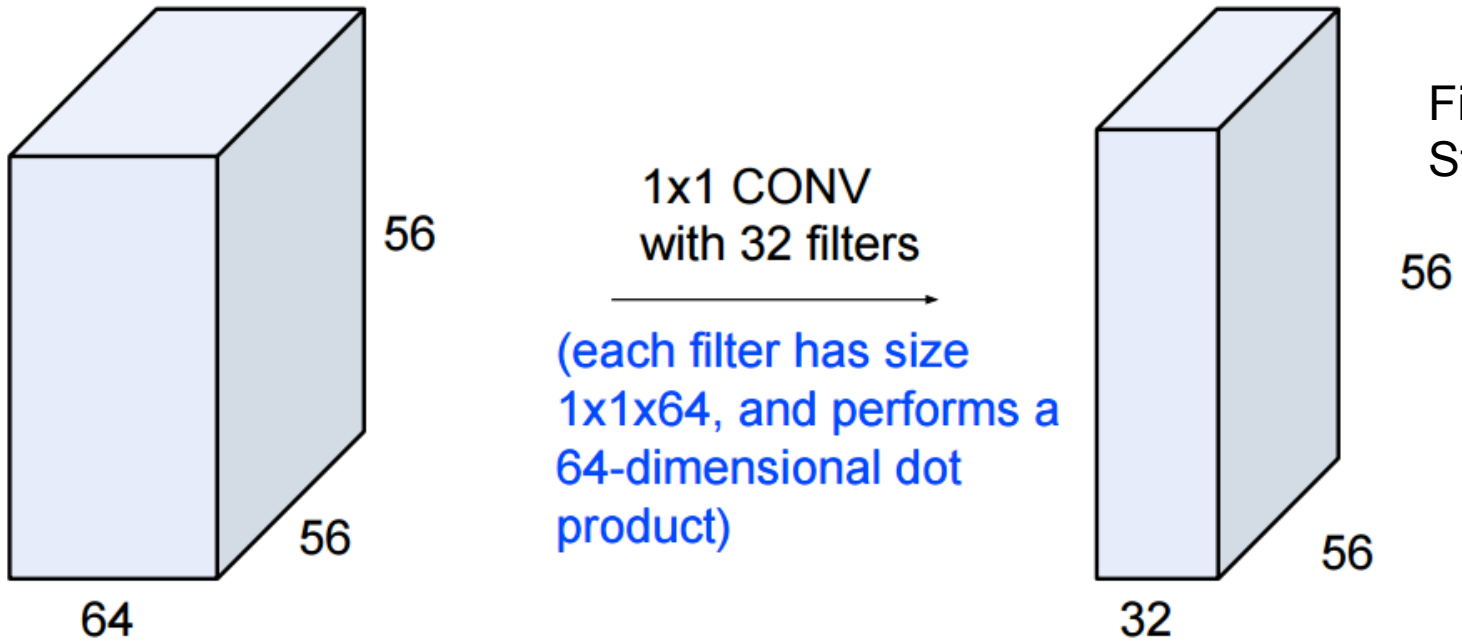


Figure Source:
Stanford cs231n

Used in Network In Network(NiN) and GoogLeNet

[Lin et al., ArXiV 2013 / ICLR 2014] [Szegedy et al., ArXiV 2014 / CVPR 2015]

Network Architecture

Reduce size and computation with 1x1 Filter

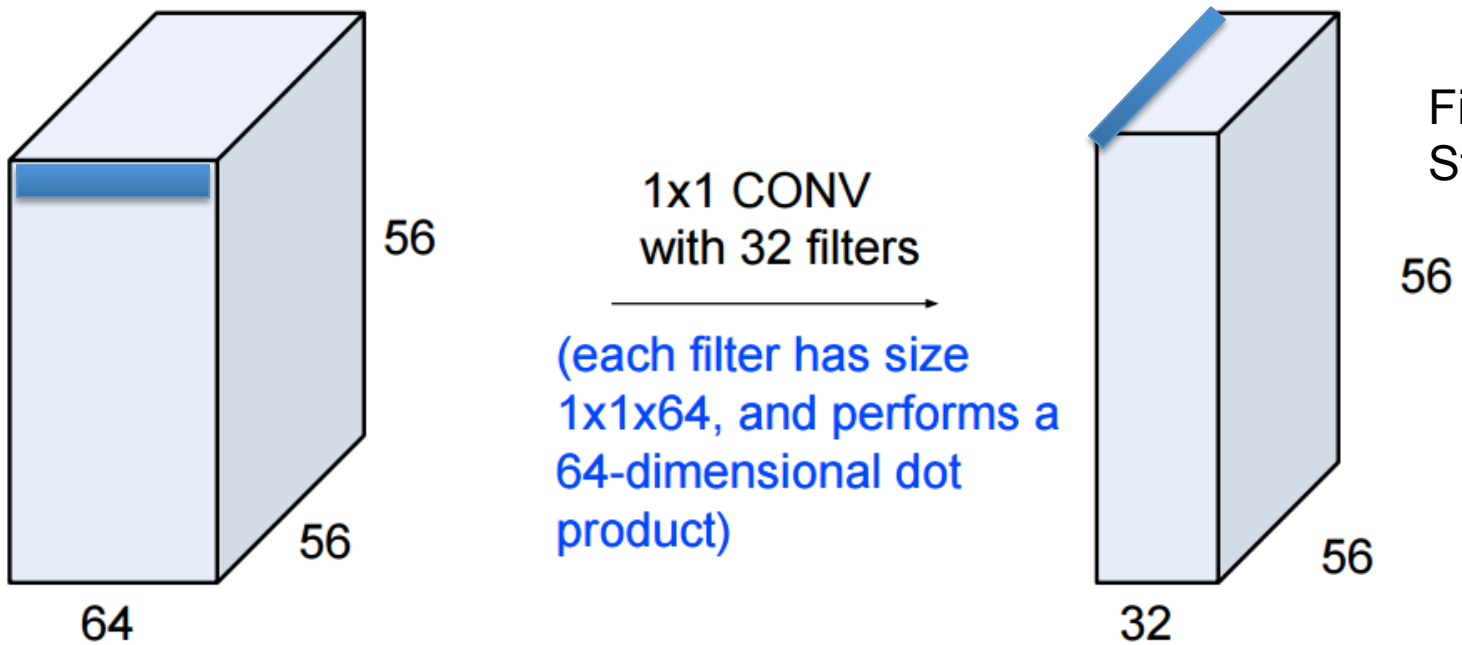


Figure Source: Stanford cs231n

Used in Network In Network(NiN) and GoogLeNet

[Lin et al., ArXiV 2013 / ICLR 2014] [Szegedy et al., ArXiV 2014 / CVPR 2015]

Network Architecture

Reduce size and computation with 1x1 Filter

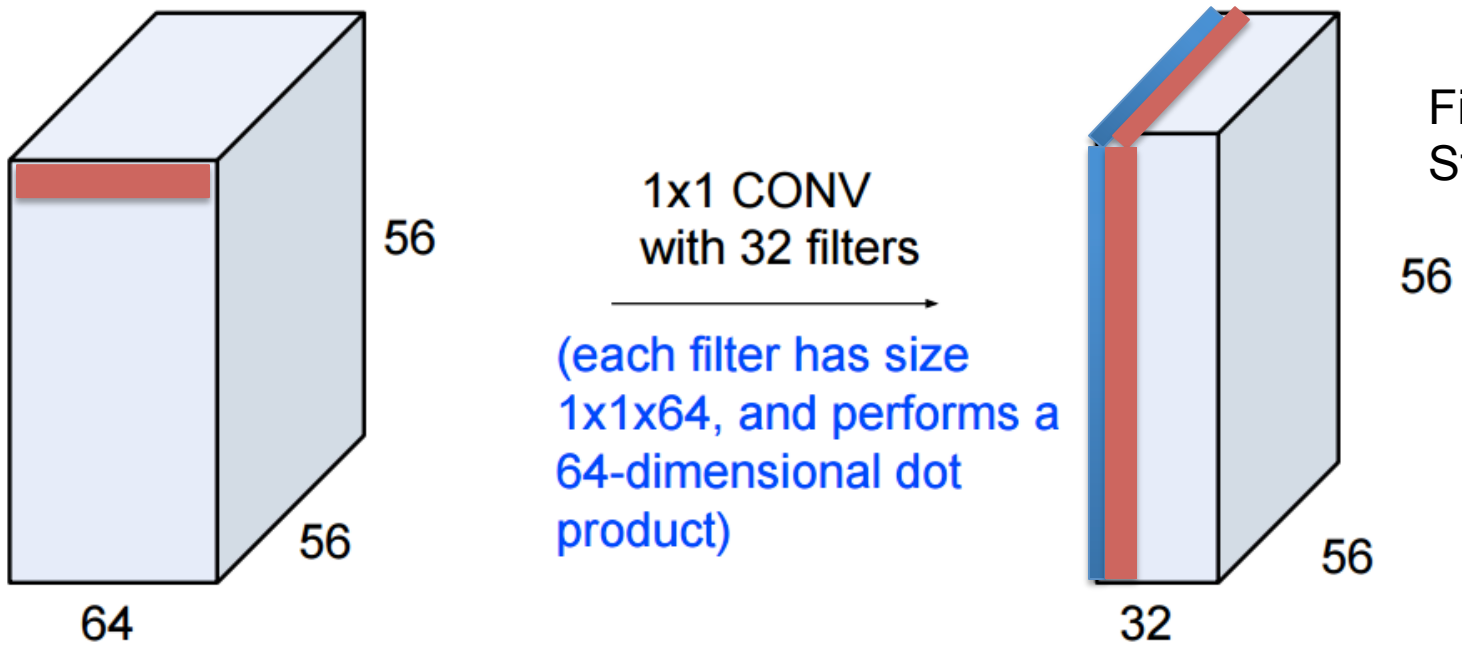
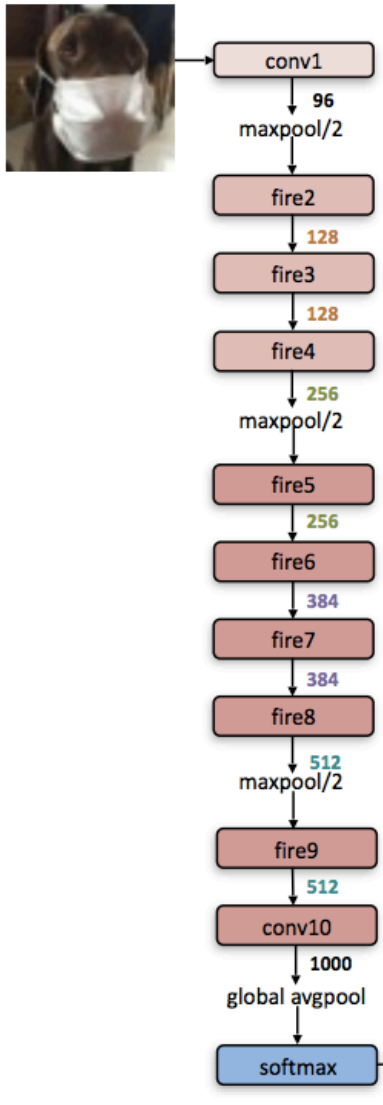


Figure Source: Stanford cs231n

Used in Network In Network(NiN) and GoogLeNet

[Lin et al., ArXiV 2013 / ICLR 2014] [Szegedy et al., ArXiV 2014 / CVPR 2015]

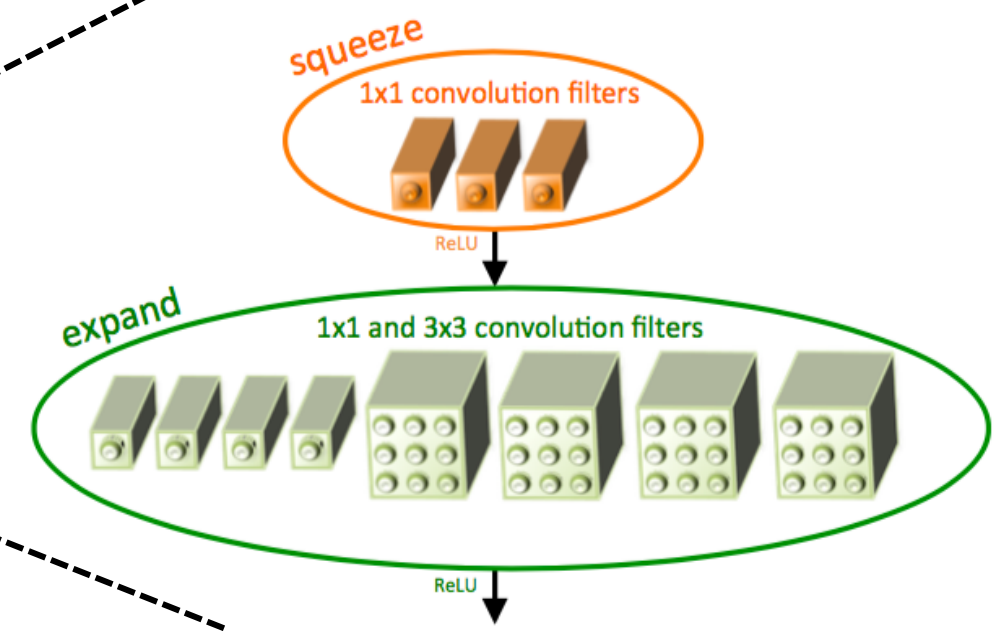
SqueezeNet



Reduce weights by reducing number of input channels by "squeezing" with 1x1

50x fewer weights than AlexNet

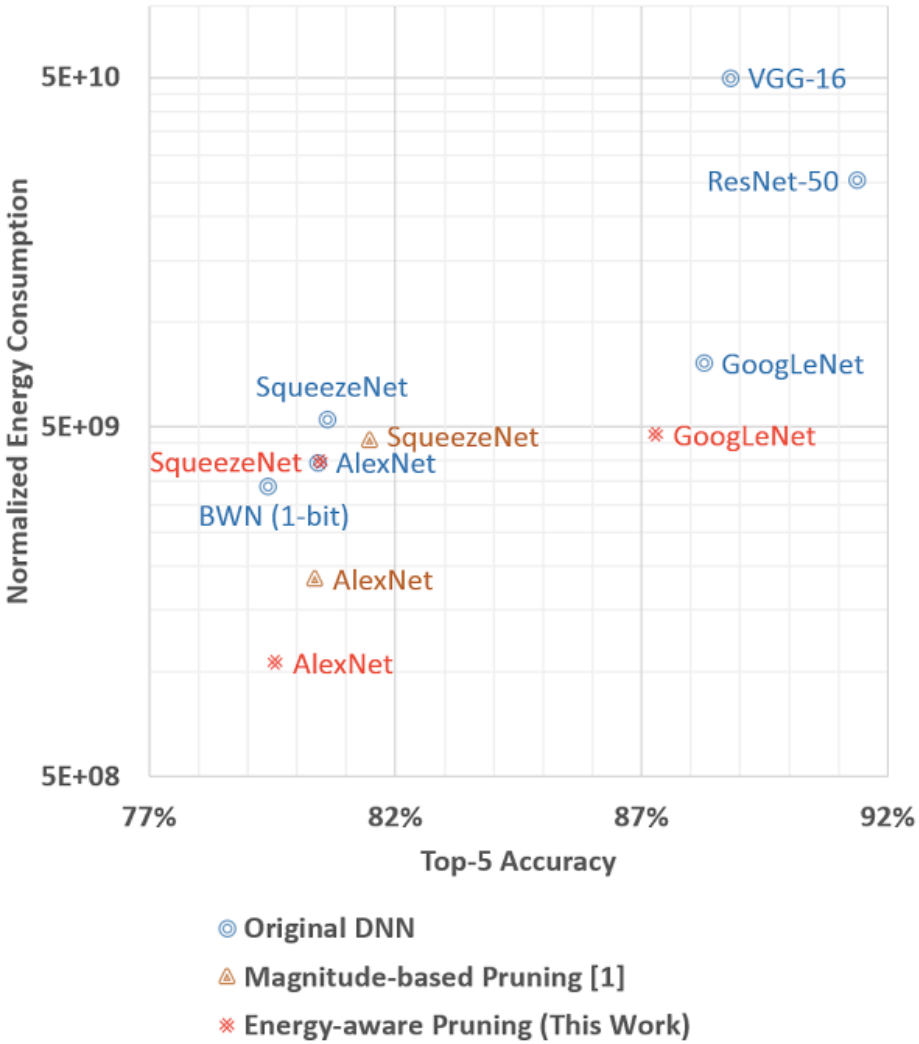
Fire Module



[F.N. Iandola et al., ArXiv, 2016]

Energy Consumption of Existing DNNs

- Maximally reducing # of weights does not necessarily result in optimized energy consumption
- Deeper CNNs with fewer weights (e.g. GoogleNet, SqueezeNet), do not necessarily consume less energy than shallower CNNs with more weights (e.g. AlexNet)
- Reducing # of weights can provide equal or more reduction than reducing the bitwidth of weights (e.g. BWN)



[Yang et al., ArXiv 2016]