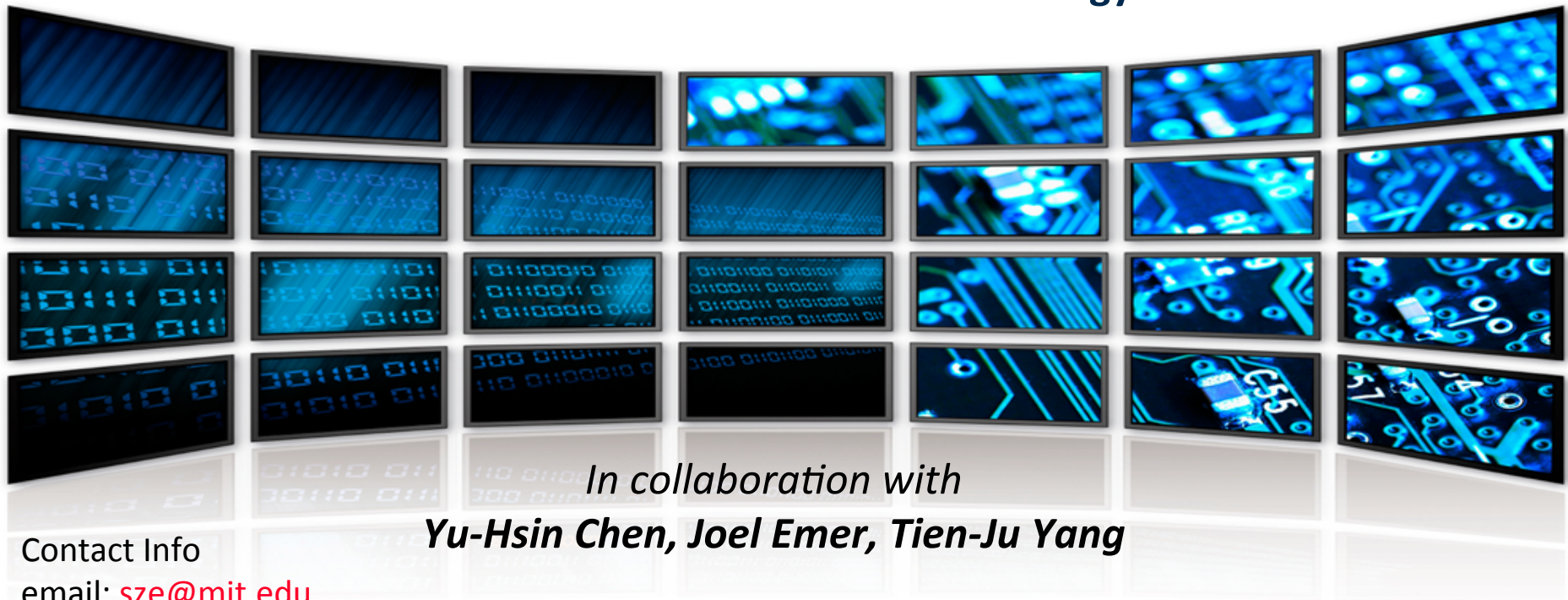


Efficient Processing for Deep Learning: Challenges and Opportunities

Vivienne Sze

Massachusetts Institute of Technology



In collaboration with
Yu-Hsin Chen, Joel Emer, Tien-Ju Yang

Contact Info

email: sze@mit.edu

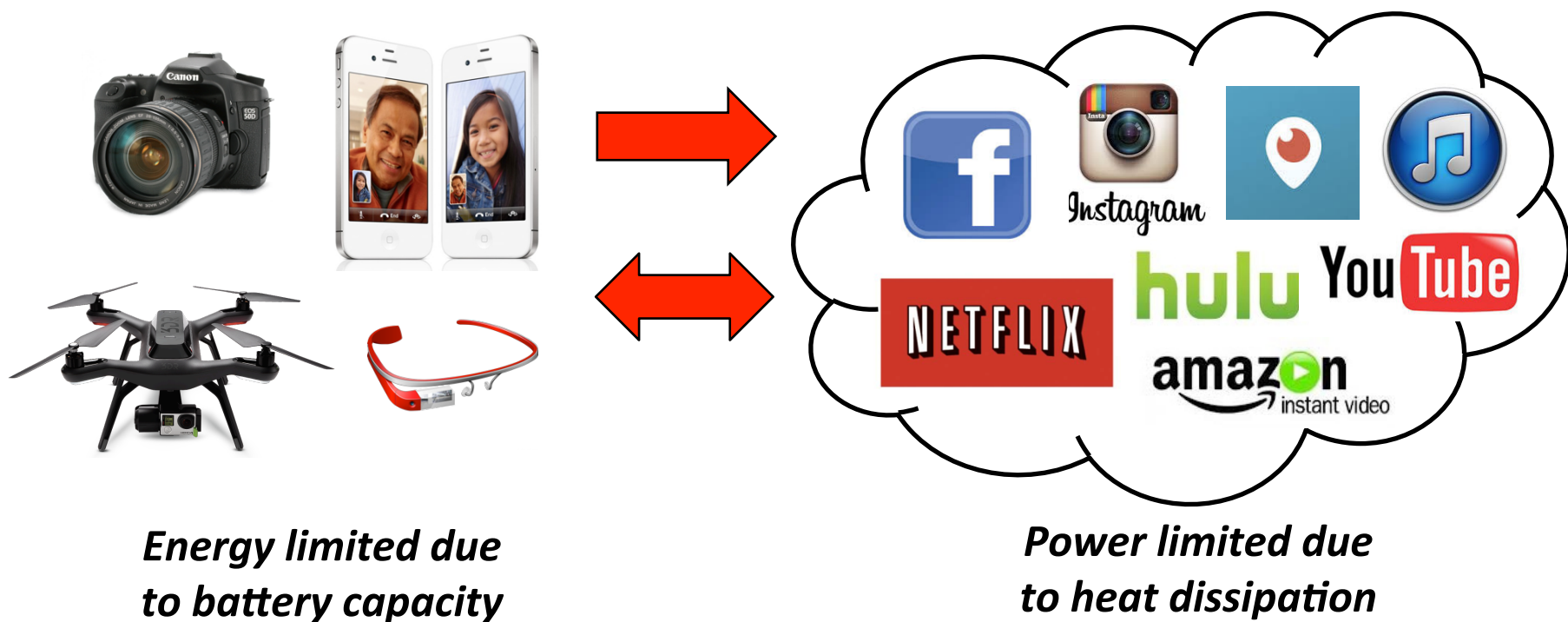
website: www.rle.mit.edu/eems

Video is the Biggest Big Data

Over 70% of today's Internet traffic is video

Over 300 hours of video uploaded to YouTube **every minute**

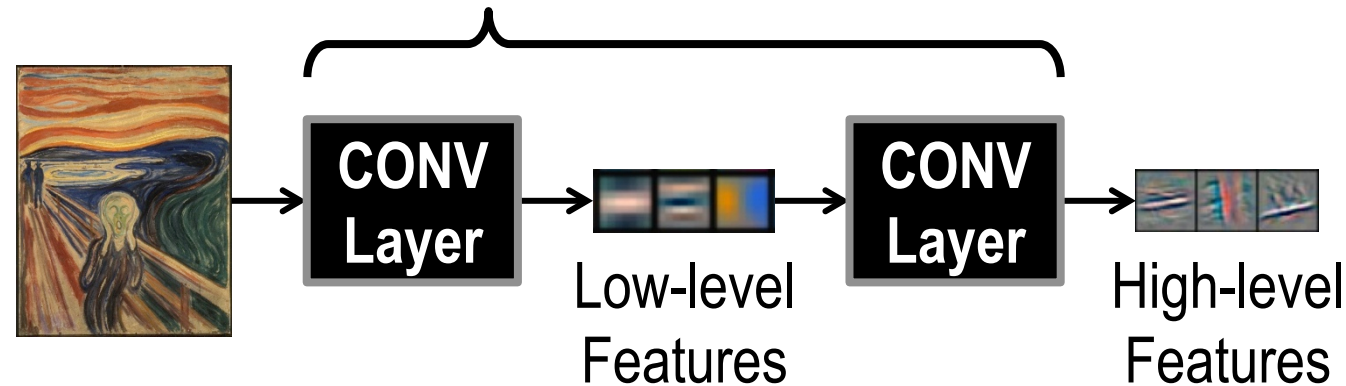
Over 500 million hours of video surveillance collected **every day**



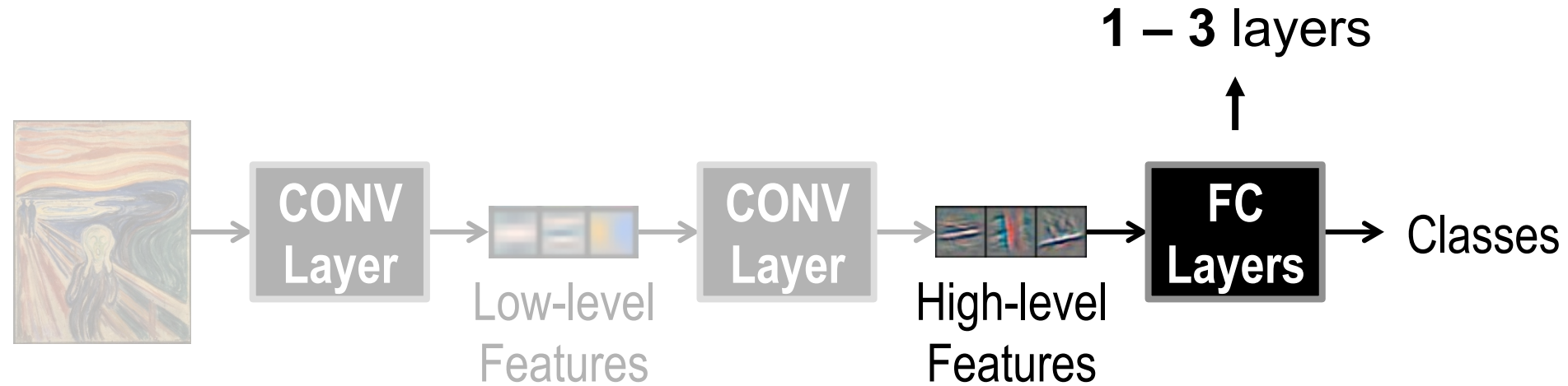
Need energy-efficient pixel processing!

Deep Convolutional Neural Networks

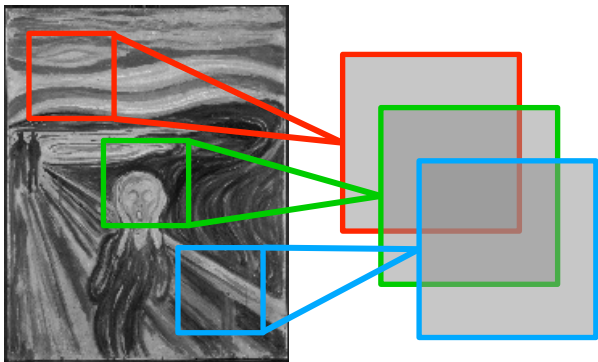
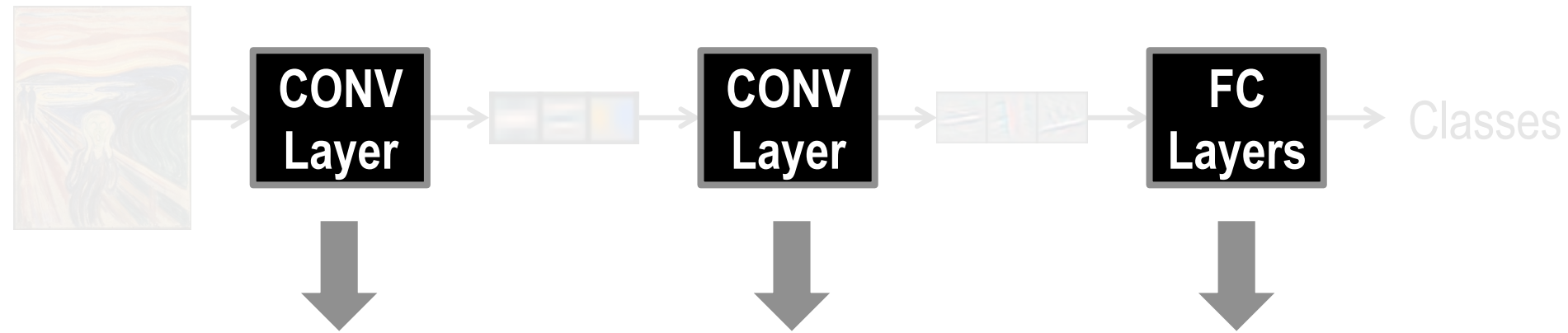
Modern *deep* CNN: up to **1000** CONV layers



Deep Convolutional Neural Networks



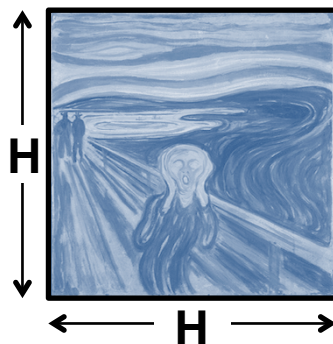
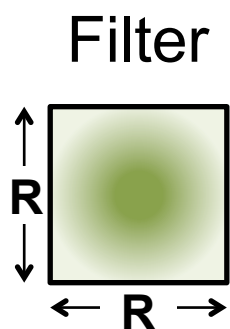
Deep Convolutional Neural Networks



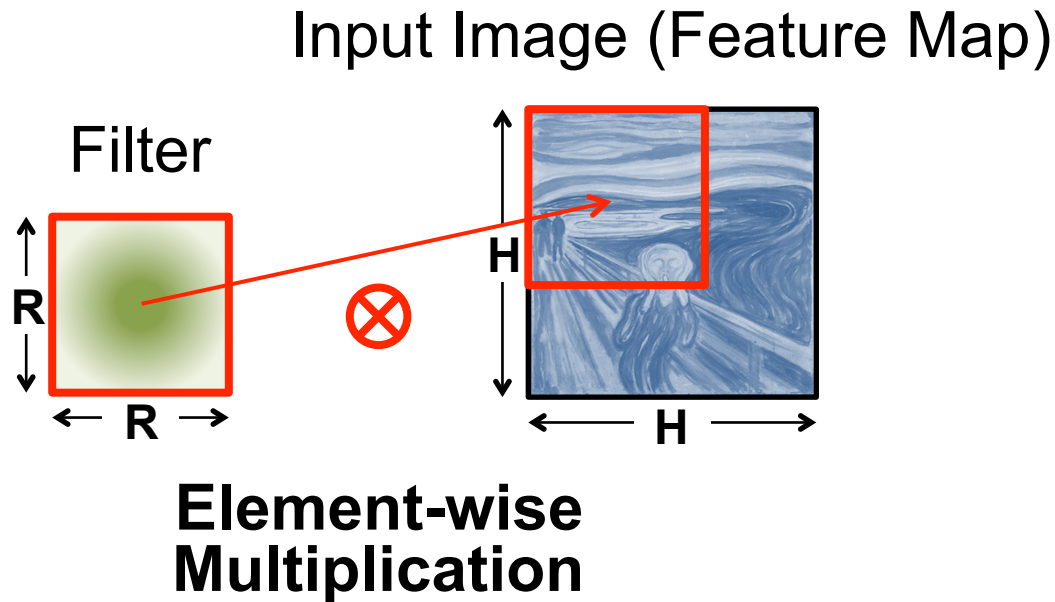
Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

High-Dimensional CNN Convolution

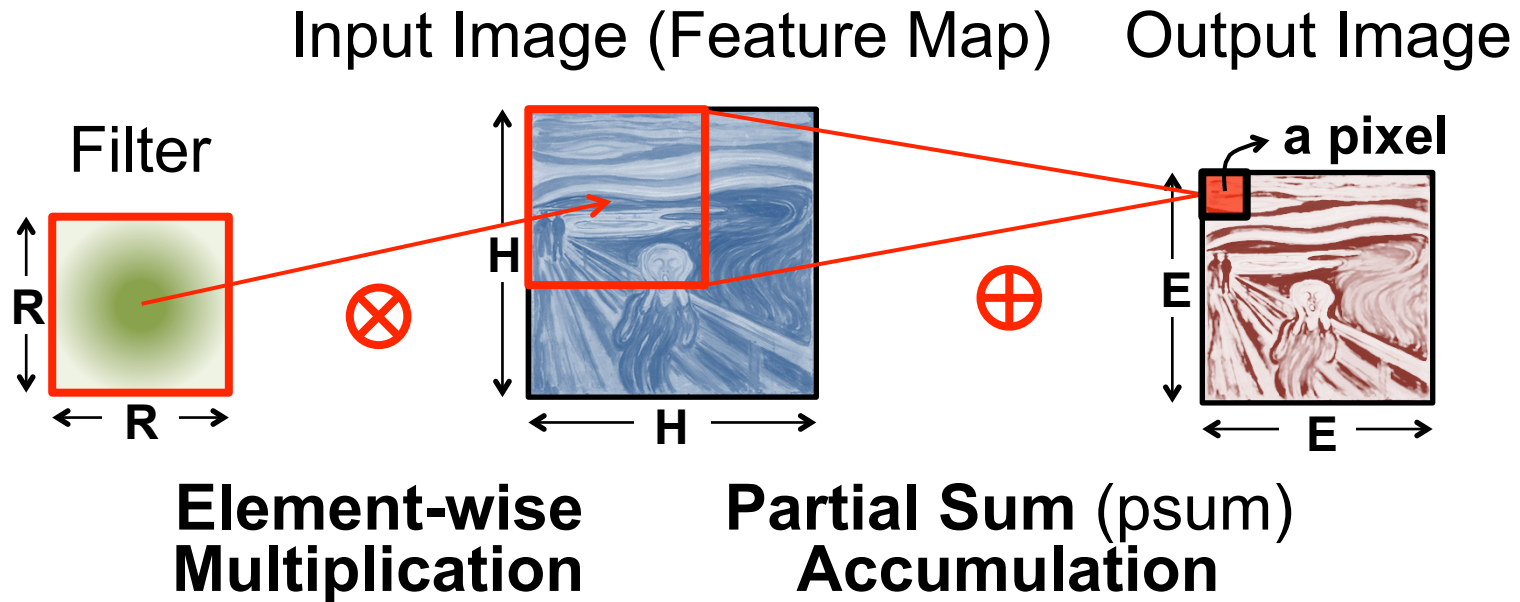
Input Image (Feature Map)



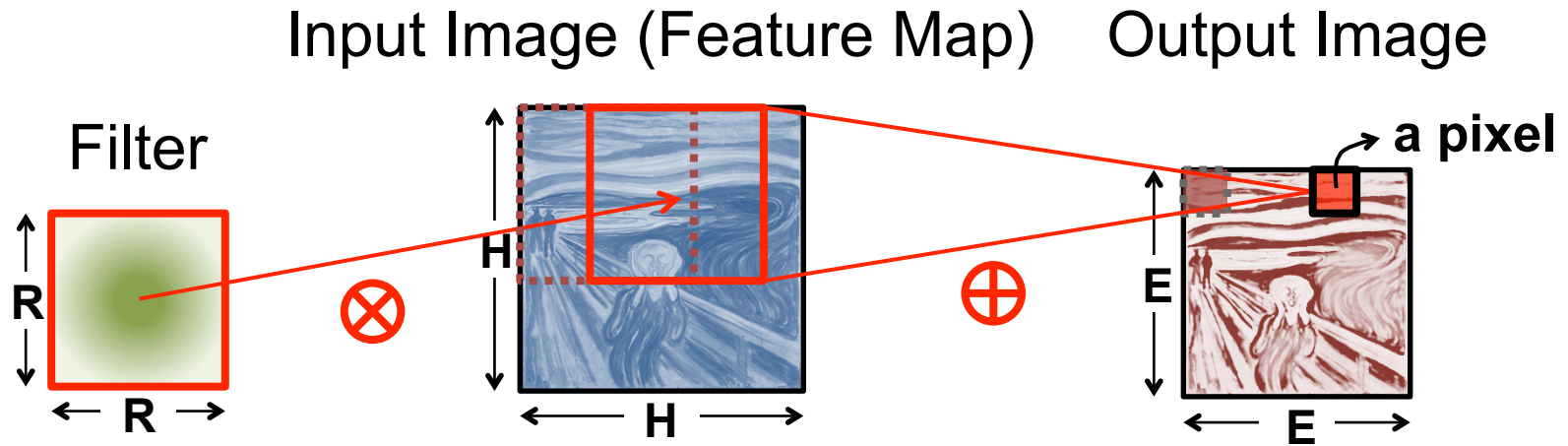
High-Dimensional CNN Convolution



High-Dimensional CNN Convolution

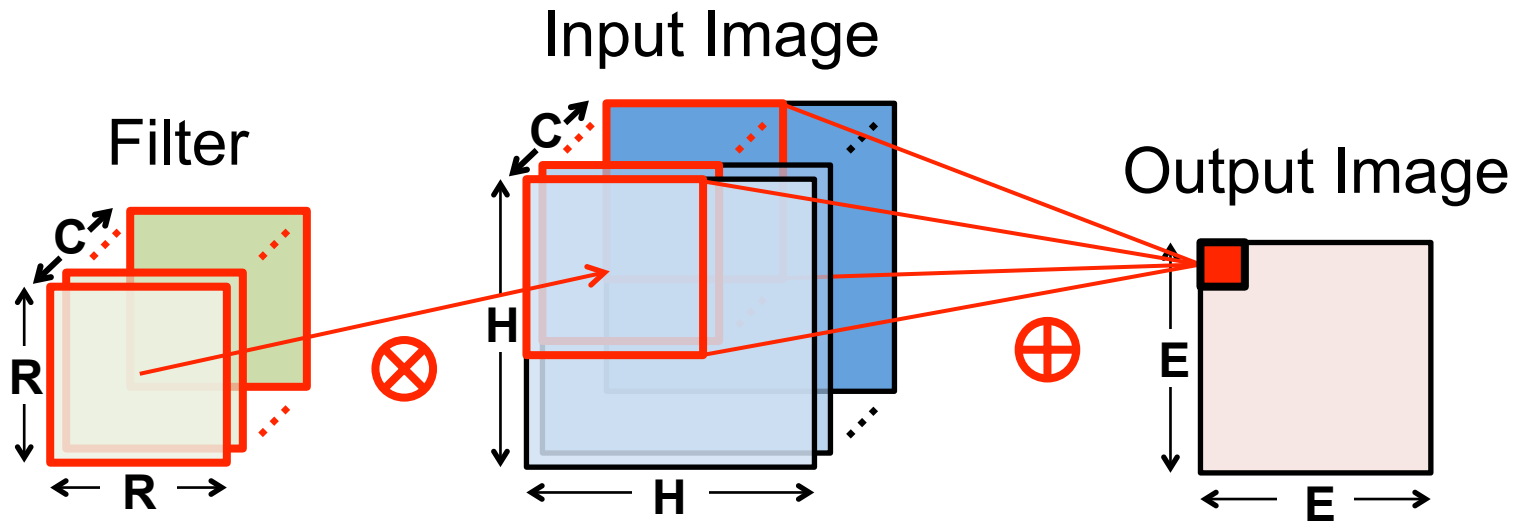


High-Dimensional CNN Convolution



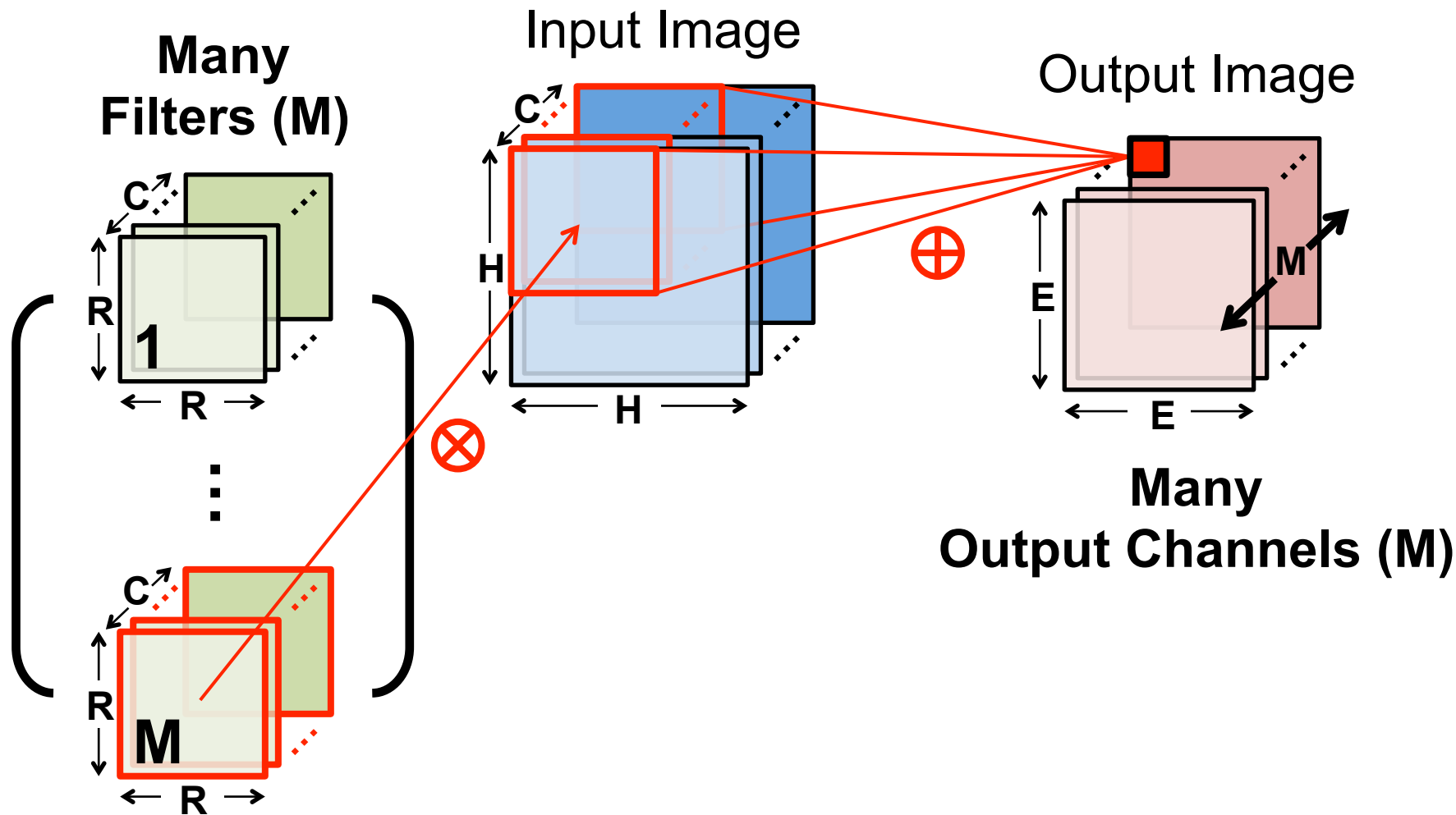
Sliding Window Processing

High-Dimensional CNN Convolution

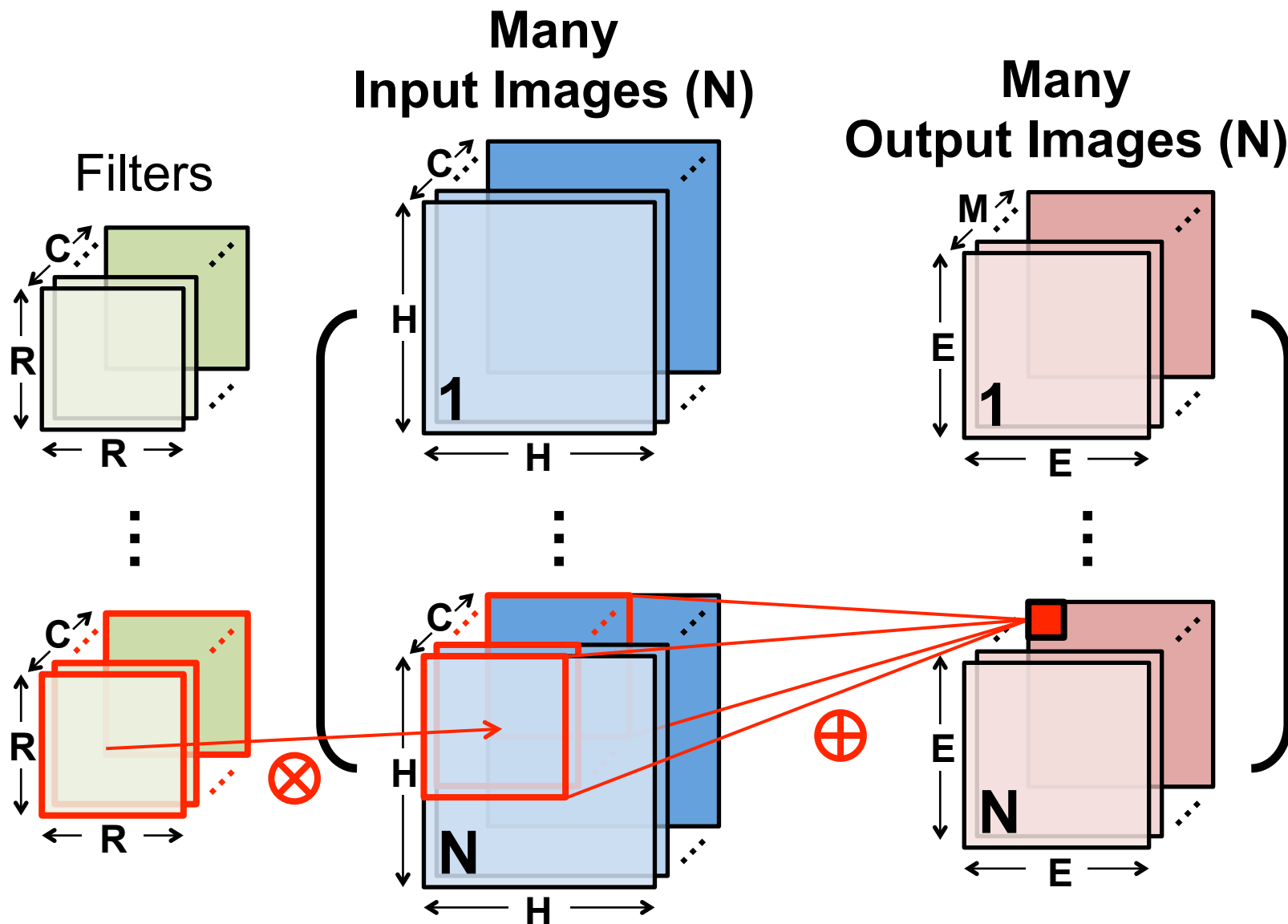


Many Input Channels (C)

High-Dimensional CNN Convolution



High-Dimensional CNN Convolution

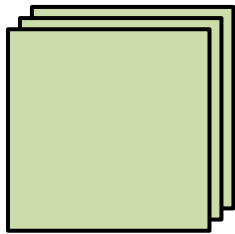


Large Sizes with Varying Shapes

AlexNet¹ Convolutional Layer Configurations

Layer	Filter Size (R)	# Filters (M)	# Channels (C)	Stride
1	11x11	96	3	4
2	5x5	256	48	1
3	3x3	384	256	1
4	3x3	384	192	1
5	3x3	256	192	1

Layer 1



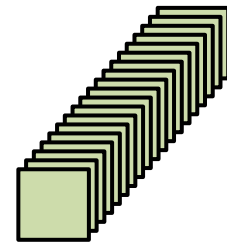
34k Params
105M MACs

Layer 2



307k Params
224M MACs

Layer 3

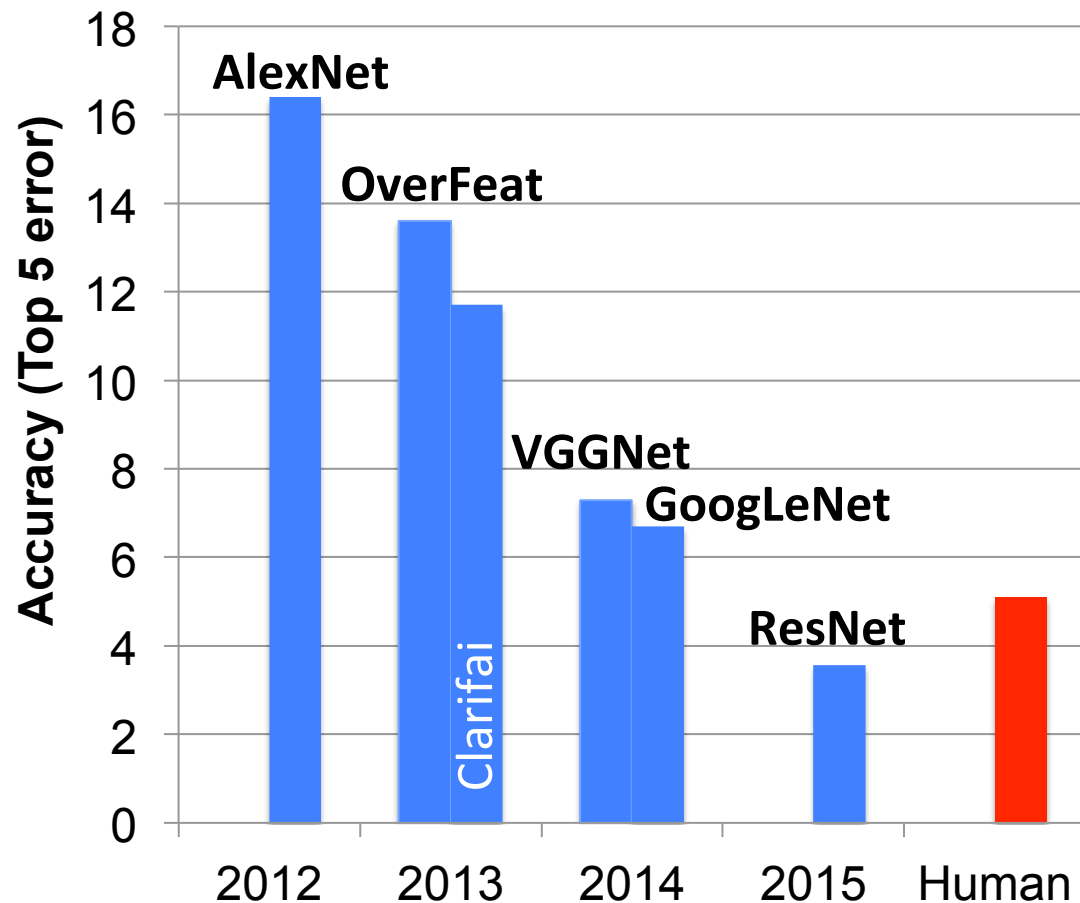


885k Params
150M MACs

Popular CNNs

- LeNet (1998)
- AlexNet (2012)
- OverFeat (2013)
- VGGNet (2014)
- GoogleNet (2014)
- ResNet (2015)

ImageNet: Large Scale Visual Recognition Challenge (ILSVRC)



[O. Russakovsky et al., IJCV 2015]

Summary of Popular CNNs

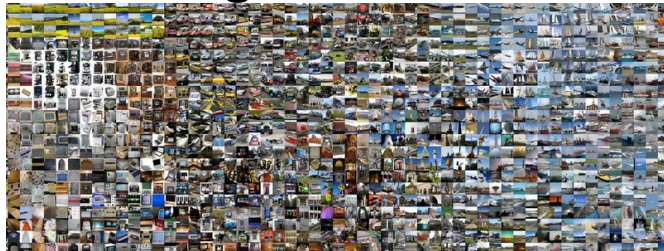
Metrics	LeNet-5	AlexNet	VGG-16	GoogLeNet (v1)	ResNet-50
Top-5 error	n/a	16.4	7.4	6.7	5.3
Input Size	28x28	227x227	224x224	224x224	224x224
# of CONV Layers	2	5	16	21 (depth)	49
Filter Sizes	5	3, 5, 11	3	1, 3, 5, 7	1, 3, 7
# of Channels	1, 6	3 - 256	3 - 512	3 - 1024	3 - 2048
# of Filters	6, 16	96 - 384	64 - 512	64 - 384	64 - 2048
Stride	1	1, 4	1	1, 2	1, 2
# of Weights	2.6k	2.3M	14.7M	6.0M	23.5M
# of MACs	283k	666M	15.3G	1.43G	3.86G
# of FC layers	2	3	3	1	1
# of Weights	58k	58.6M	124M	1M	2M
# of MACs	58k	58.6M	124M	1M	2M
Total Weights	60k	61M	138M	7M	25.5M
Total MACs	341k	724M	15.5G	1.43G	3.9G

CONV Layers increasingly important!

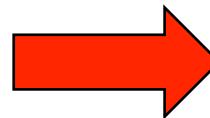
Training vs. Inference

Training
(determine weights)

Large Datasets



Weights



Inference
(use weights)



Challenges

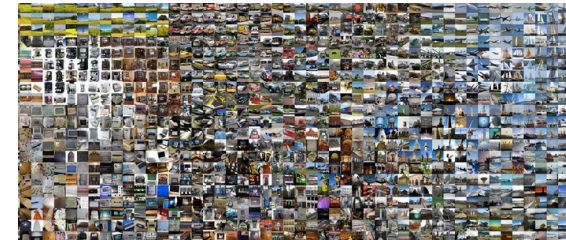
Key Metrics

- **Accuracy**
 - Evaluate hardware using the appropriate DNN model and dataset
- **Programmability**
 - Support multiple applications
 - Different weights
- **Energy/Power**
 - Energy per operation
 - DRAM Bandwidth
- **Throughput/Latency**
 - GOPS, frame rate, delay
- **Cost**
 - Area (size of memory and # of cores)

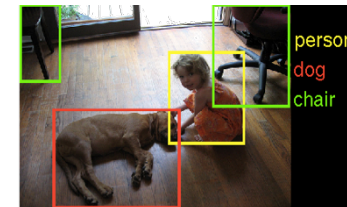
MNIST



ImageNet



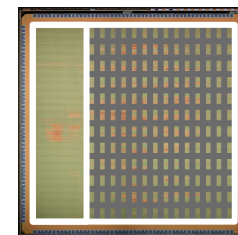
Computer Vision



Speech Recognition



Chip



Opportunities in Architecture

GPUs and CPUs Targeting Deep Learning

Intel Knights Landing (2016)

Nvidia PASCAL GP100 (2016)



Knights Mill: next gen Xeon
Phi “optimized for deep
learning”

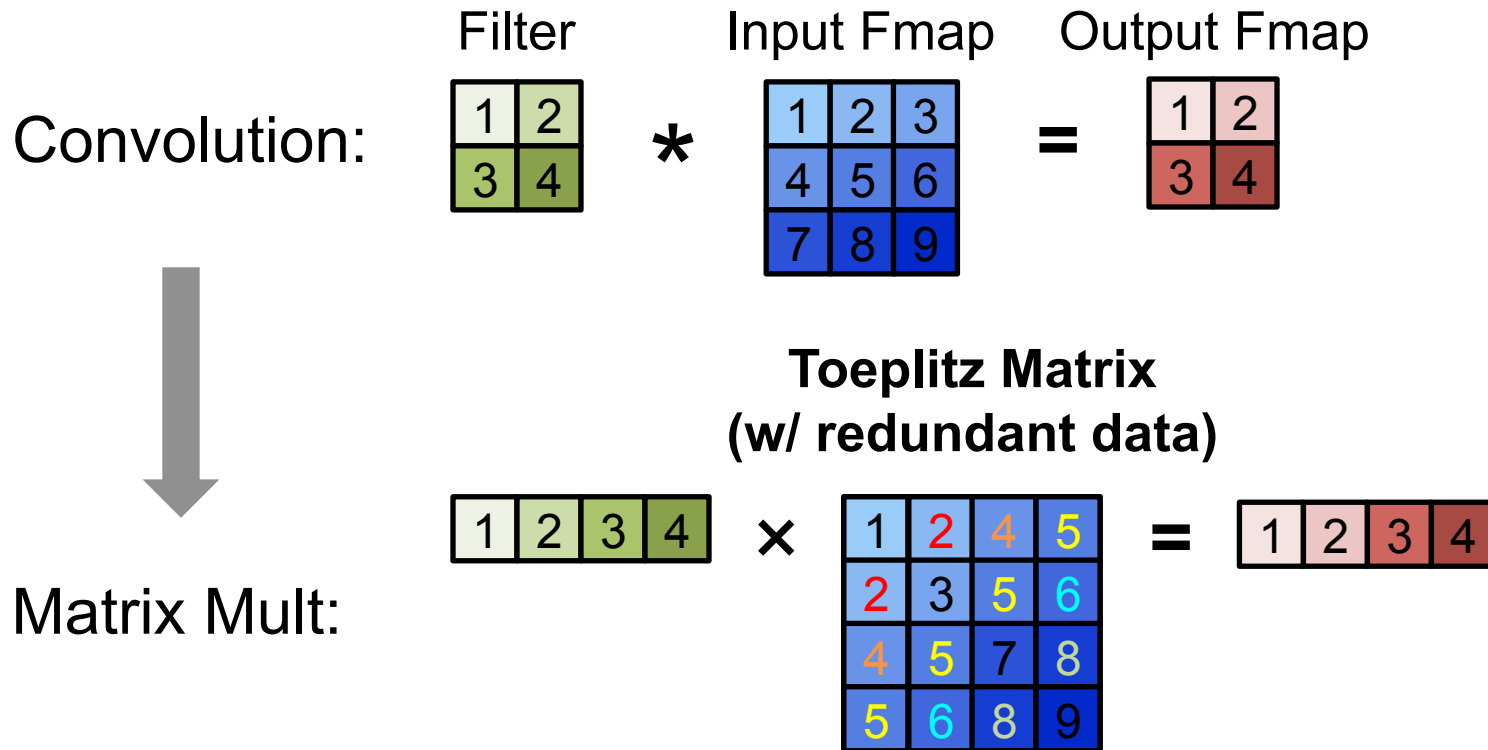
Use **matrix multiplication libraries** on CPUs and GPUs

Accelerate Matrix Multiplication

- Implementation: **Matrix Multiplication (GEMM)**
 - **CPU:** OpenBLAS, Intel MKL, etc
 - **GPU:** cuBLAS, cuDNN, etc
- Optimized by tiling to storage hierarchy

Map DNN to a Matrix Multiplication

- Convert to matrix mult. using the **Toeplitz Matrix**



Data is repeated

Goal: Reduced number of operations to **increase throughput**

Computation Transformations

- **Goal: Bitwise same result, but reduce number of operations**
- **Focuses mostly on compute**

Analogy: Gauss's Multiplication Algorithm

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i.$$

4 multiplications + 3 additions

$$k_1 = c \cdot (a + b)$$

$$k_2 = a \cdot (d - c)$$

$$k_3 = b \cdot (c + d)$$

$$\text{Real part} = k_1 - k_3$$

$$\text{Imaginary part} = k_1 + k_2.$$

3 multiplications + 5 additions

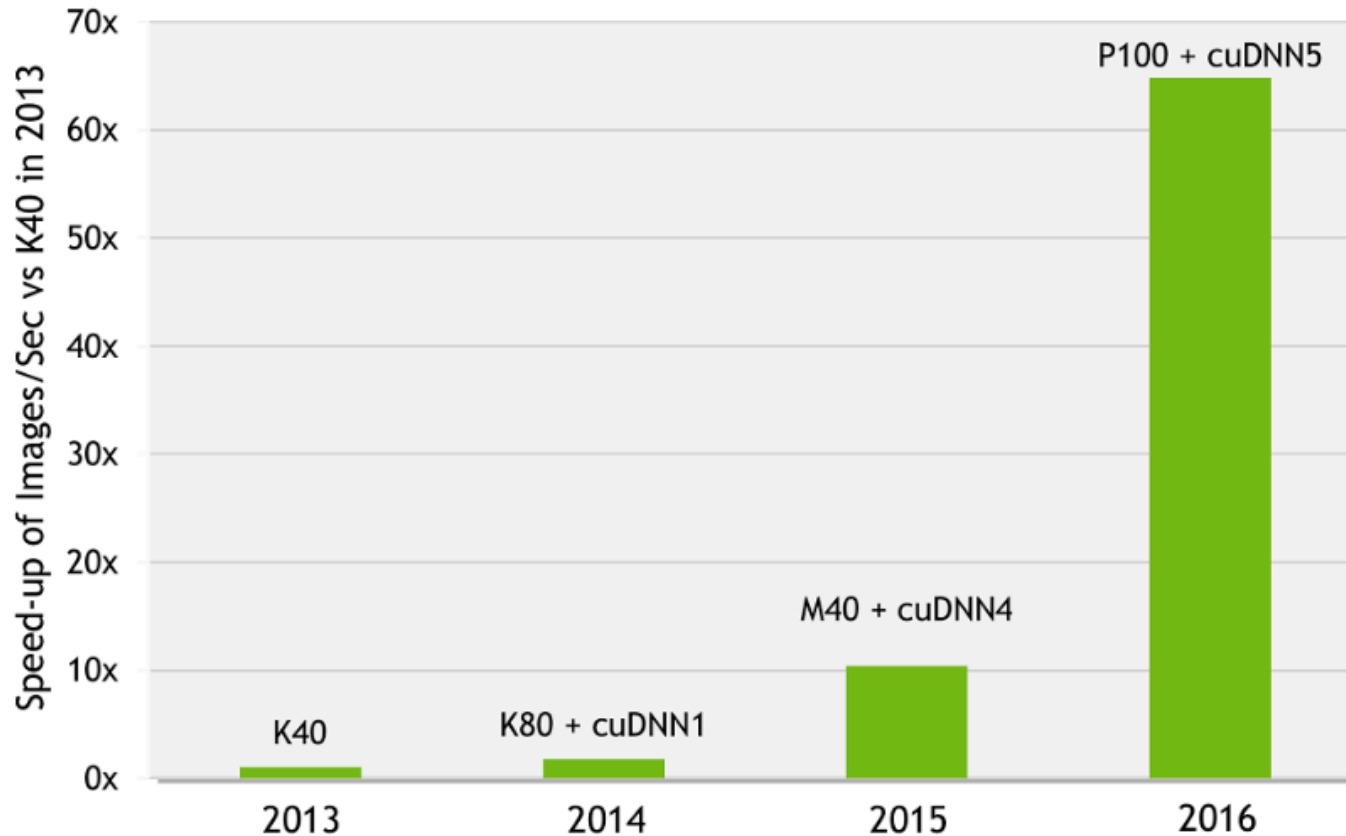
Reduce number of multiplications, but **increase** number of additions

Reduce Operations in Matrix Multiplication

- **Winograd** [Lavin, CVPR 2016]
 - **Pro:** 2.25x speed up for 3x3 filter
 - **Con:** Specialized processing depending on filter size
- **Fast Fourier Transform** [Mathieu, ICLR 2014]
 - **Pro:** Direct convolution $O(N_o^2 N_f^2)$ to $O(N_o^2 \log_2 N_o)$
 - **Con:** Increase storage requirements
- **Strassen** [Cong, ICANN 2014]
 - **Pro:** $O(N^3)$ to $(N^{2.807})$
 - **Con:** Numerical stability

cuDNN: Speed up with Transformations

60x Faster Training in 3 Years



AlexNet training throughput on:

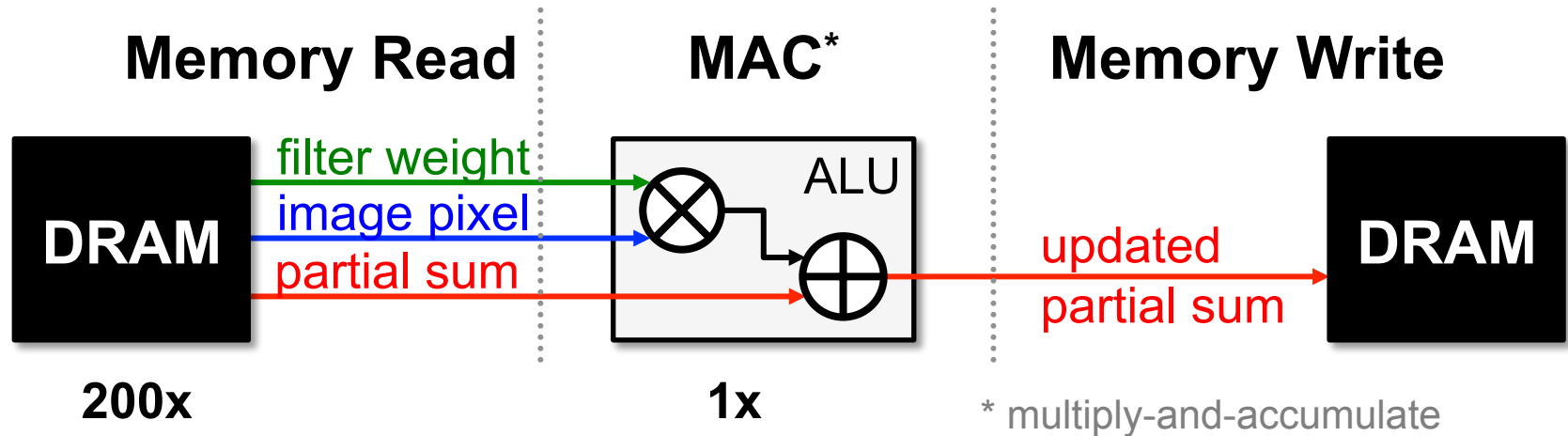
CPU: 1x E5-2680v3 12 Core 2.5GHz. 128GB System Memory, Ubuntu 14.04

M40 bar: 8x M40 GPUs in a node, P100: 8x P100 NVLink-enabled

Specialized Hardware (Accelerators)

Properties We Can Leverage

- Operations exhibit **high parallelism**
→ **high throughput** possible
- Memory Access is the Bottleneck

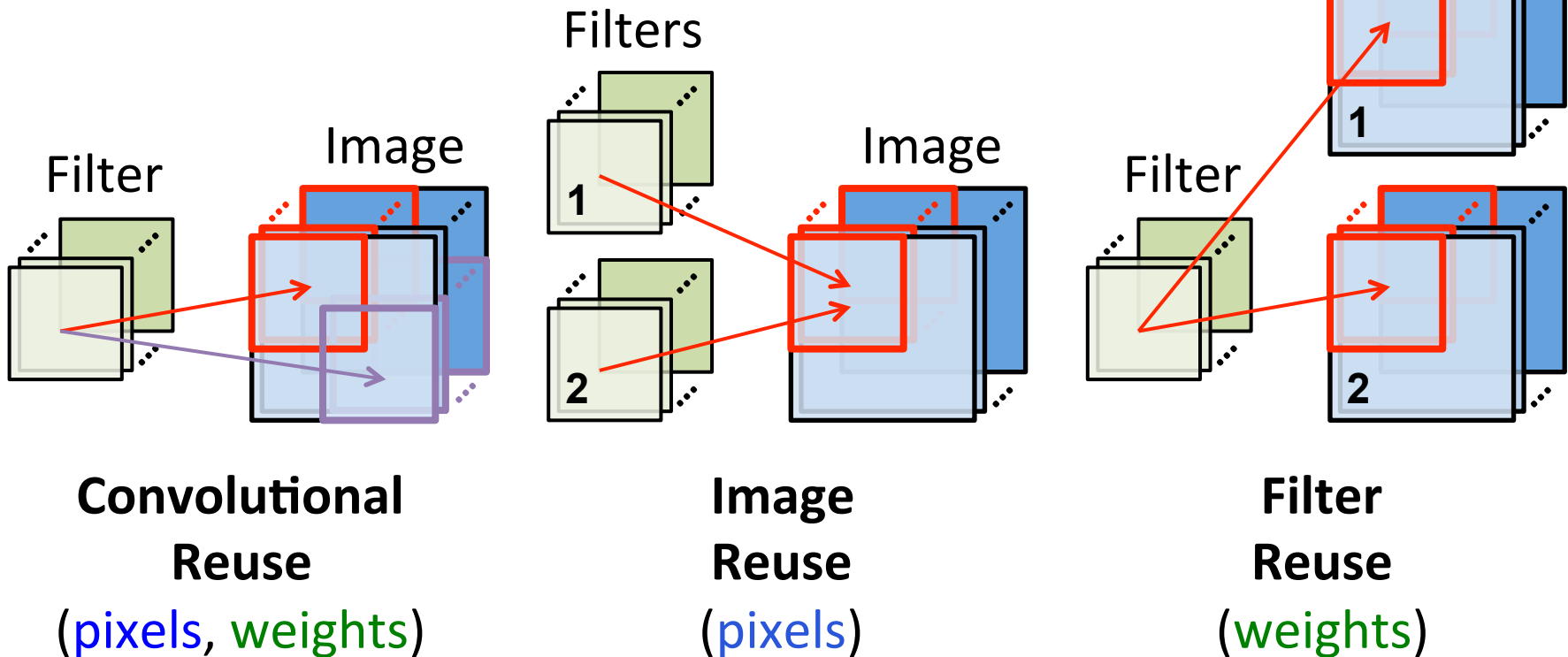


Worst Case: all memory R/W are **DRAM** accesses

- Example: AlexNet [NIPS 2012] has **724M** MACs
→ **2896M** DRAM accesses required

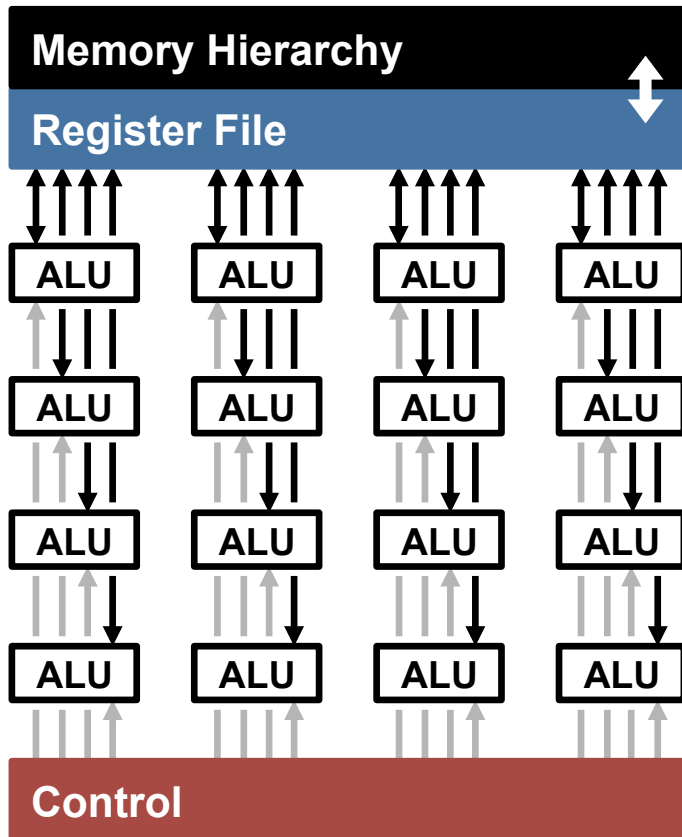
Properties We Can Leverage

- Operations exhibit **high parallelism**
→ **high throughput** possible
- Input data reuse** opportunities (up to 500x)
→ exploit **low-cost memory**

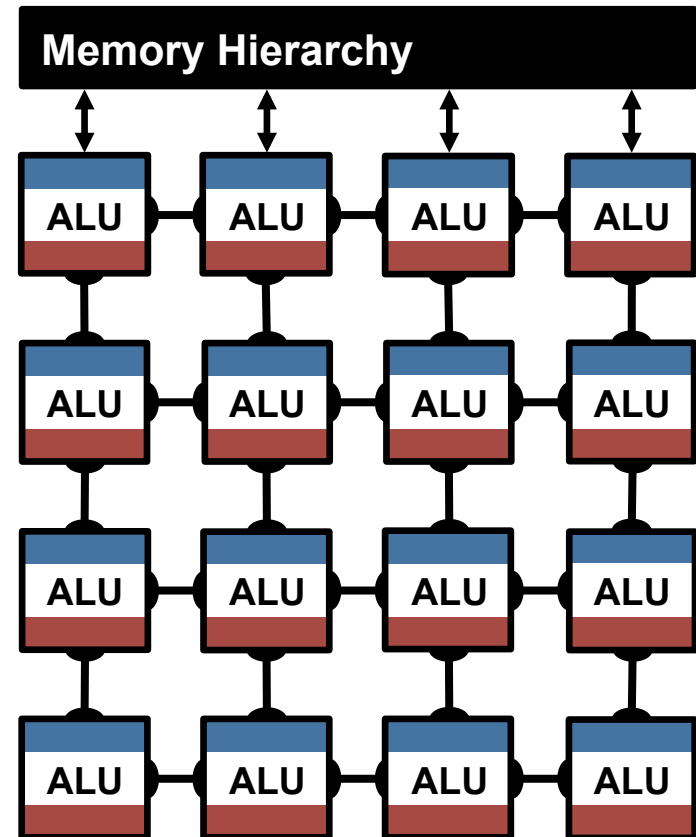


Highly-Parallel Compute Paradigms

Temporal Architecture (SIMD/SIMT)



Spatial Architecture (Dataflow Processing)



Advantages of Spatial Architecture

Temporal Architecture
(SIMD/SIMT)

Efficient Data Reuse

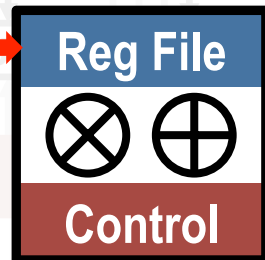
Distributed local storage (RF)

Inter-PE Communication

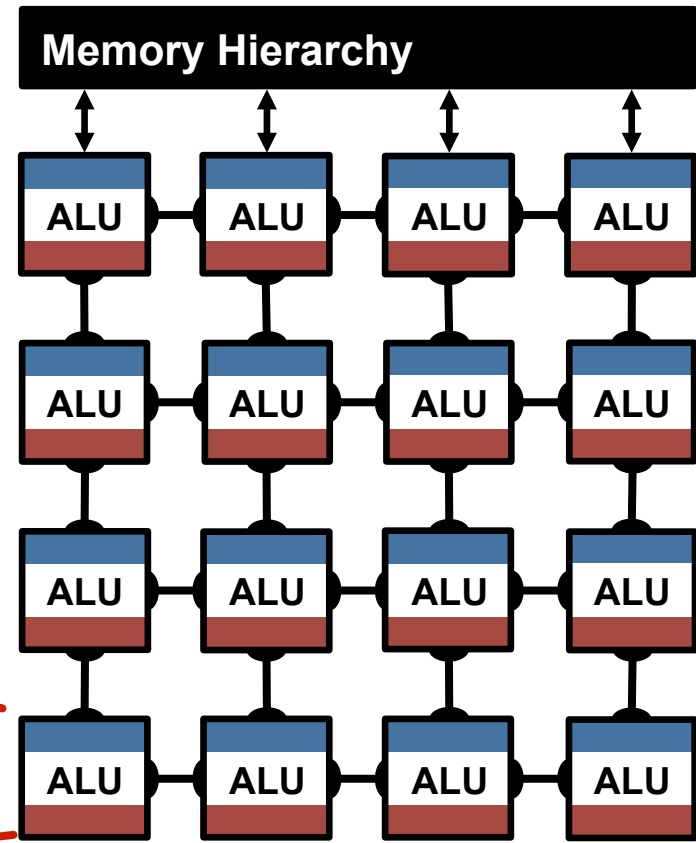
Sharing among regions of PEs

Processing
Element (PE)

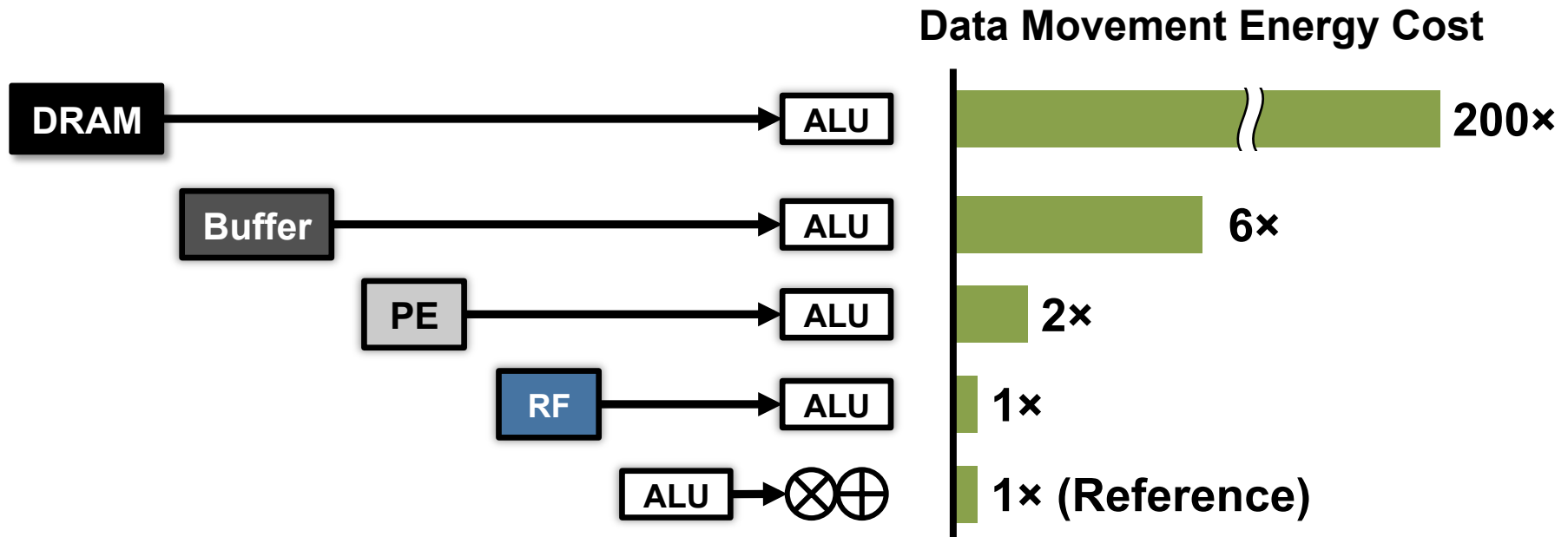
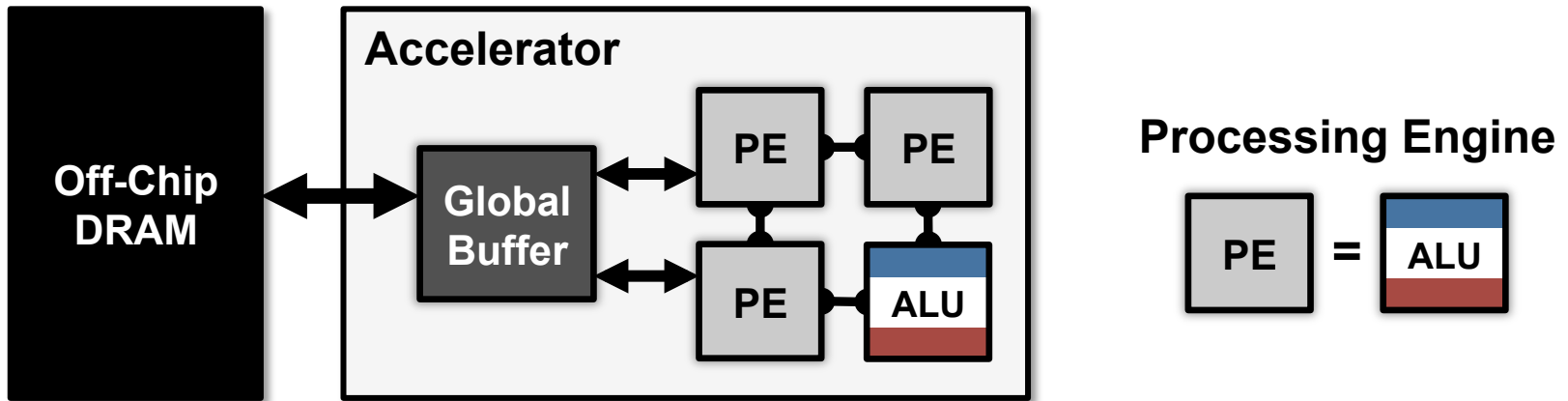
0.5 – 1.0 kB



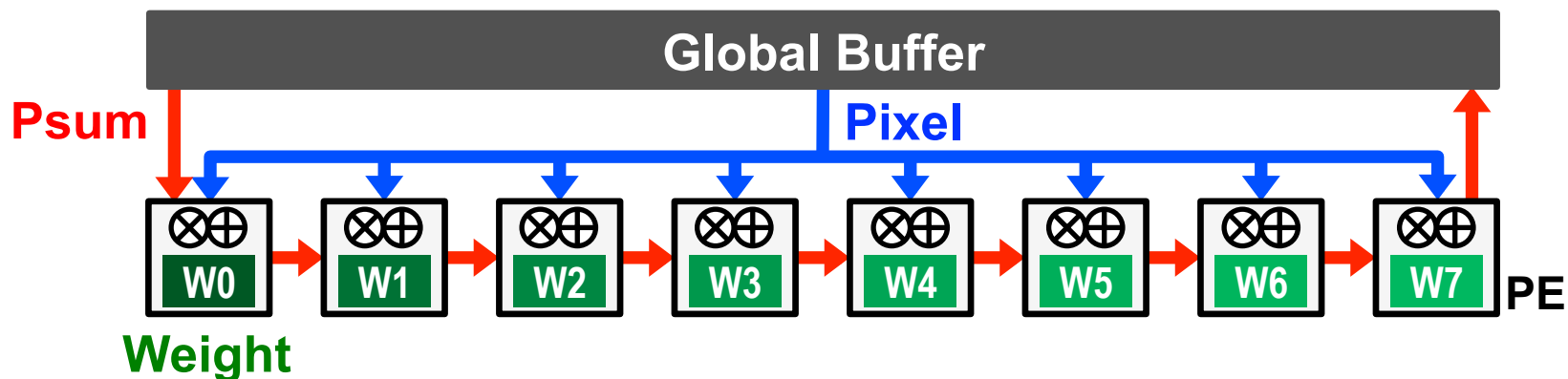
Spatial Architecture
(Dataflow Processing)



Data Movement is Expensive



Maximize data reuse at lower levels of hierarchy

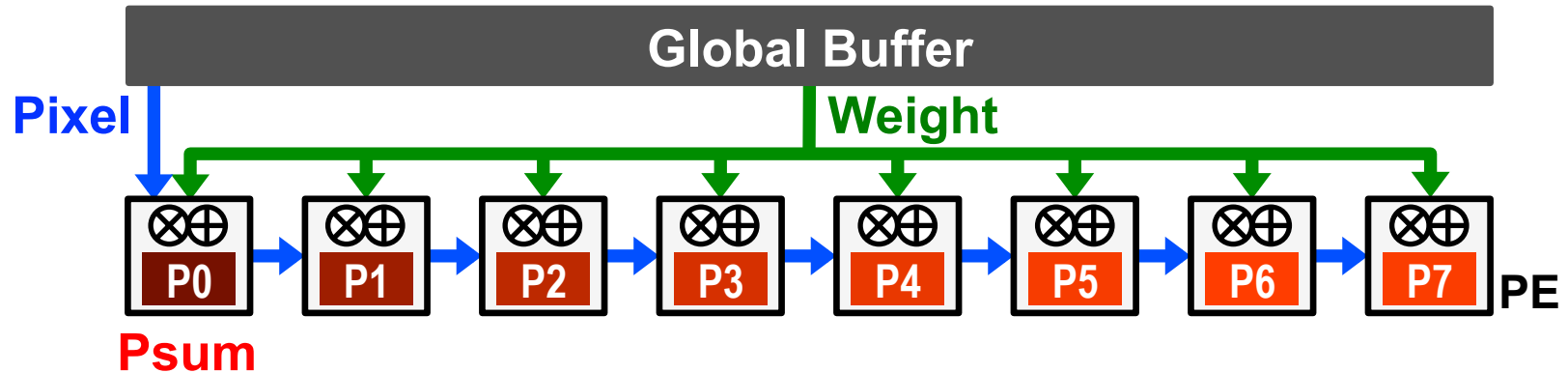


- **Minimize **weight**** read energy consumption
 - maximize convolutional and filter reuse of weights

- **Examples:**

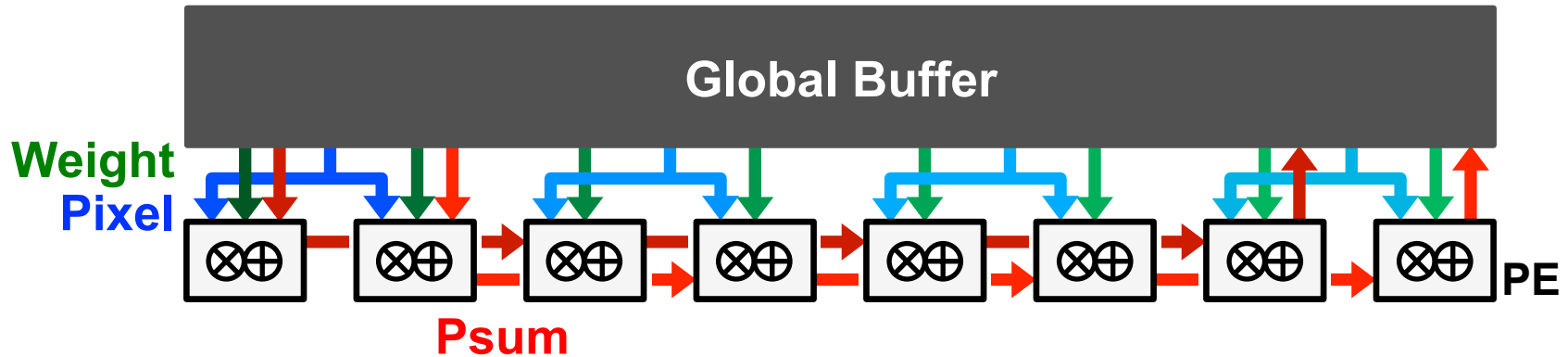
[Chakradhar, *ISCA* 2010] [nn-X (NeuFlow), *CVPRW* 2014]
[Park, *ISSCC* 2015] [Origami, *GLSVLSI* 2015]

Output Stationary (OS)



- Minimize **partial sum** R/W energy consumption
 - maximize local accumulation
- Examples:
 - [Gupta, *ICML* 2015]
 - [ShiDianNao, *ISCA* 2015]
 - [Peemen, *ICCD* 2013]

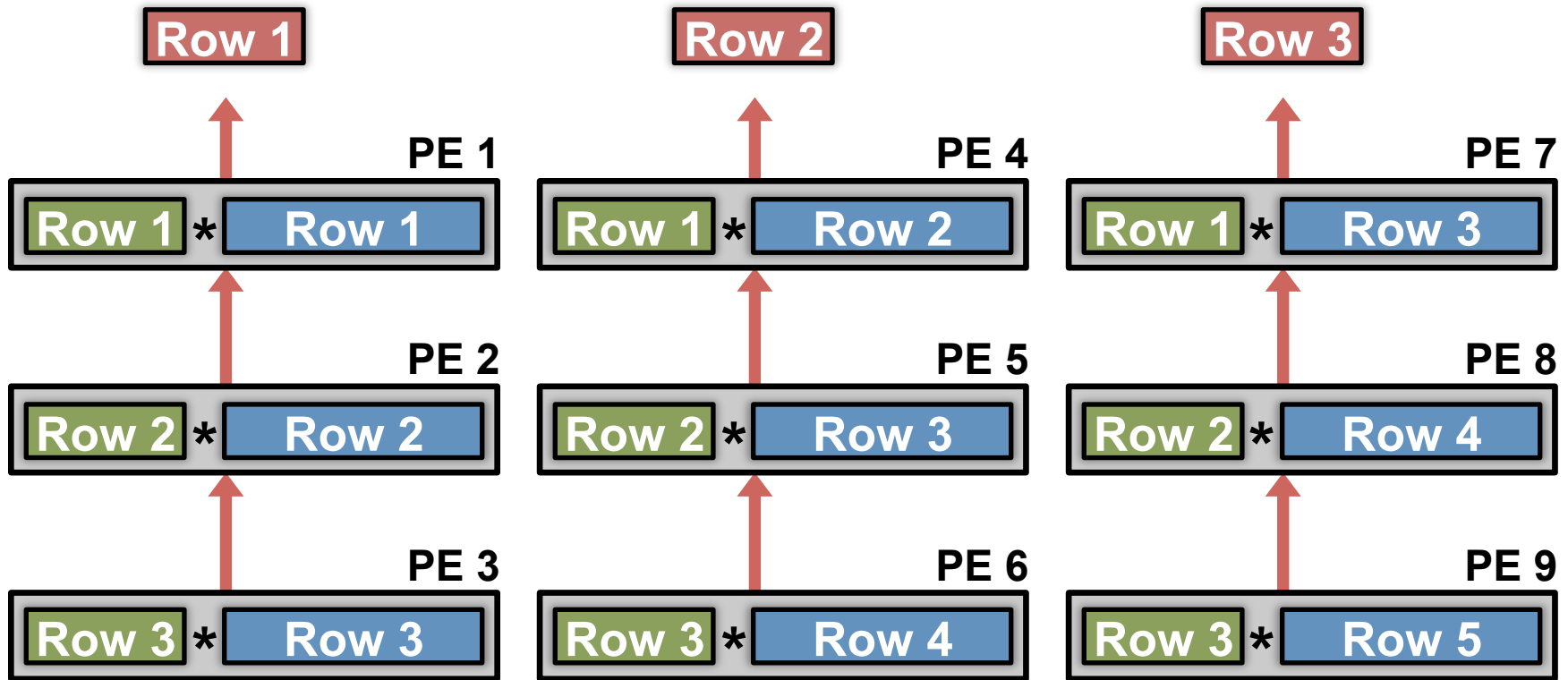
No Local Reuse (NLR)



- Use a **large global buffer** as shared storage
 - Reduce **DRAM** access energy consumption
- **Examples:**

[DianNao, *ASPLOS* 2014] [DaDianNao, *MICRO* 2014]
 [Zhang, *FPGA* 2015]

Row Stationary Dataflow

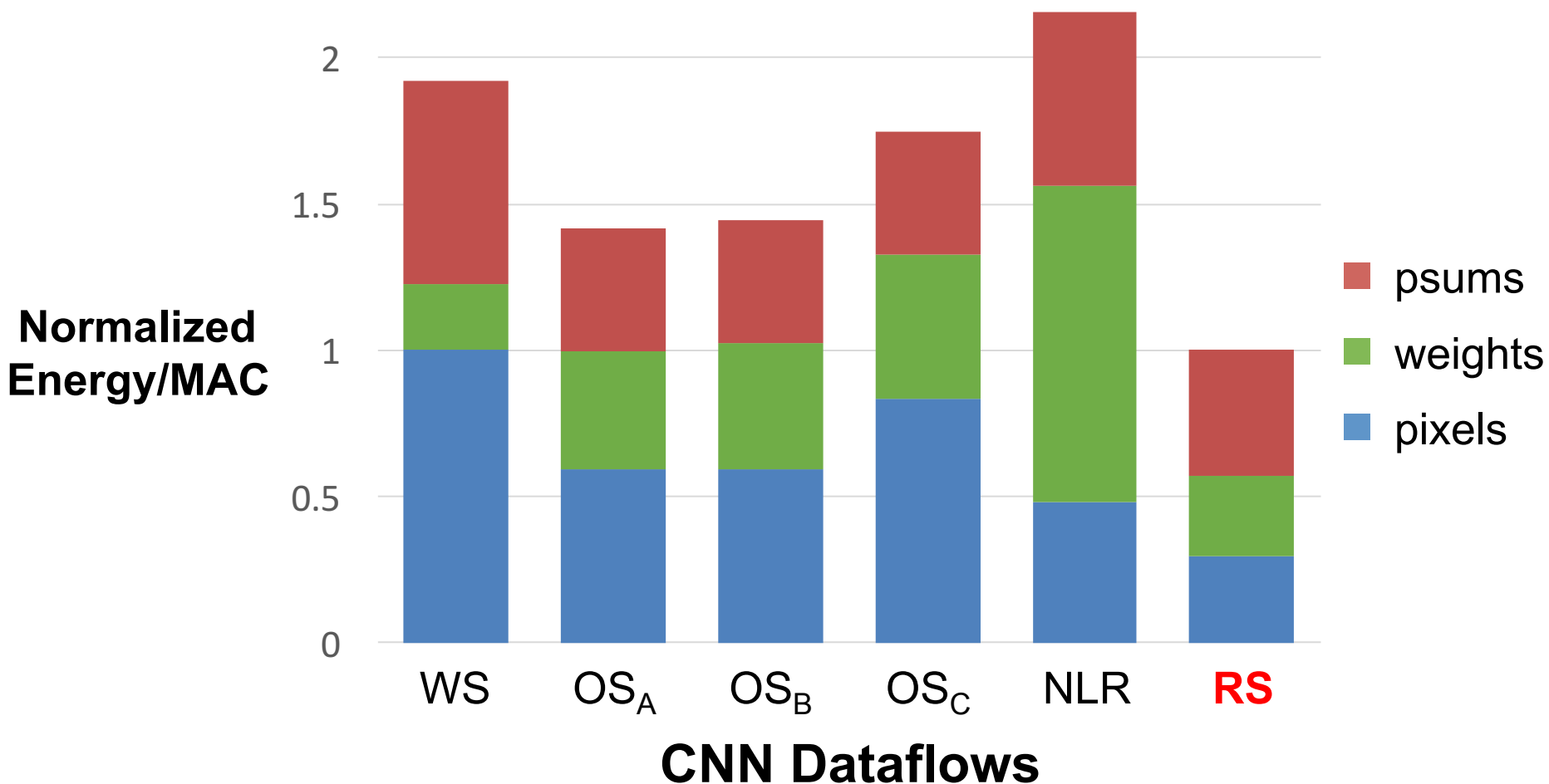


$$\begin{bmatrix} \text{Green} & \text{Green} & \text{Green} \\ \text{Green} & \text{Green} & \text{Green} \\ \text{Green} & \text{Green} & \text{Green} \end{bmatrix} * \begin{bmatrix} \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \\ \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \\ \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \end{bmatrix} = \begin{bmatrix} \text{Red} & \text{Red} & \text{Red} \\ \text{Red} & \text{Red} & \text{Red} \\ \text{Red} & \text{Red} & \text{Red} \end{bmatrix}$$

$$\begin{bmatrix} \text{Green} & \text{Green} & \text{Green} \\ \text{Green} & \text{Green} & \text{Green} \\ \text{Green} & \text{Green} & \text{Green} \end{bmatrix} * \begin{bmatrix} \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \\ \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \\ \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \end{bmatrix} = \begin{bmatrix} \text{Red} & \text{Red} & \text{Red} \\ \text{Red} & \text{Red} & \text{Red} \\ \text{Red} & \text{Red} & \text{Red} \end{bmatrix}$$

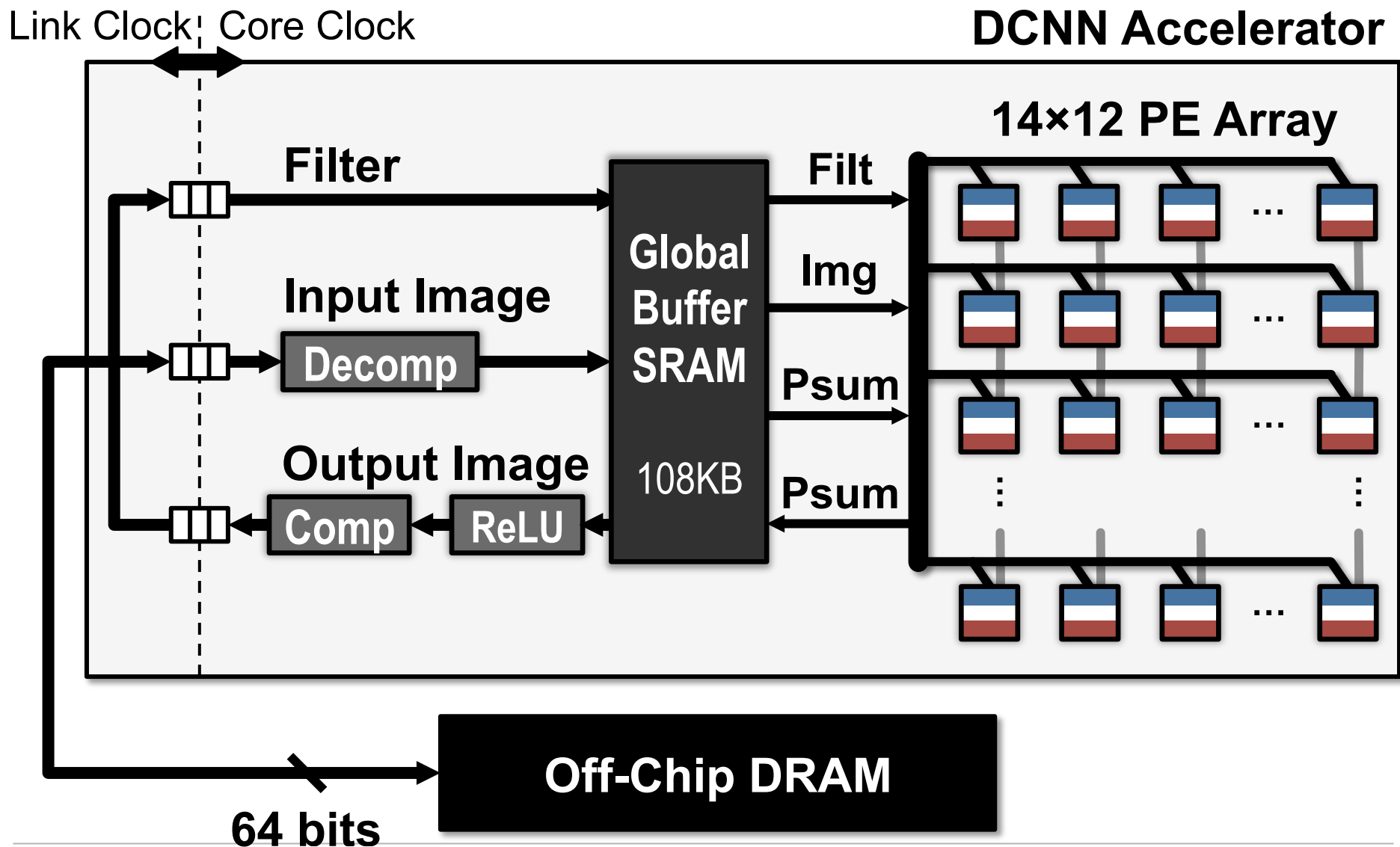
$$\begin{bmatrix} \text{Green} & \text{Green} & \text{Green} \\ \text{Green} & \text{Green} & \text{Green} \\ \text{Green} & \text{Green} & \text{Green} \end{bmatrix} * \begin{bmatrix} \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \\ \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \\ \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} & \text{Blue} \end{bmatrix} = \begin{bmatrix} \text{Red} & \text{Red} & \text{Red} \\ \text{Red} & \text{Red} & \text{Red} \\ \text{Red} & \text{Red} & \text{Red} \end{bmatrix}$$

Dataflow Comparison: CONV Layers



RS uses **1.4× – 2.5× lower** energy than other dataflows

Eyeriss Deep CNN Accelerator



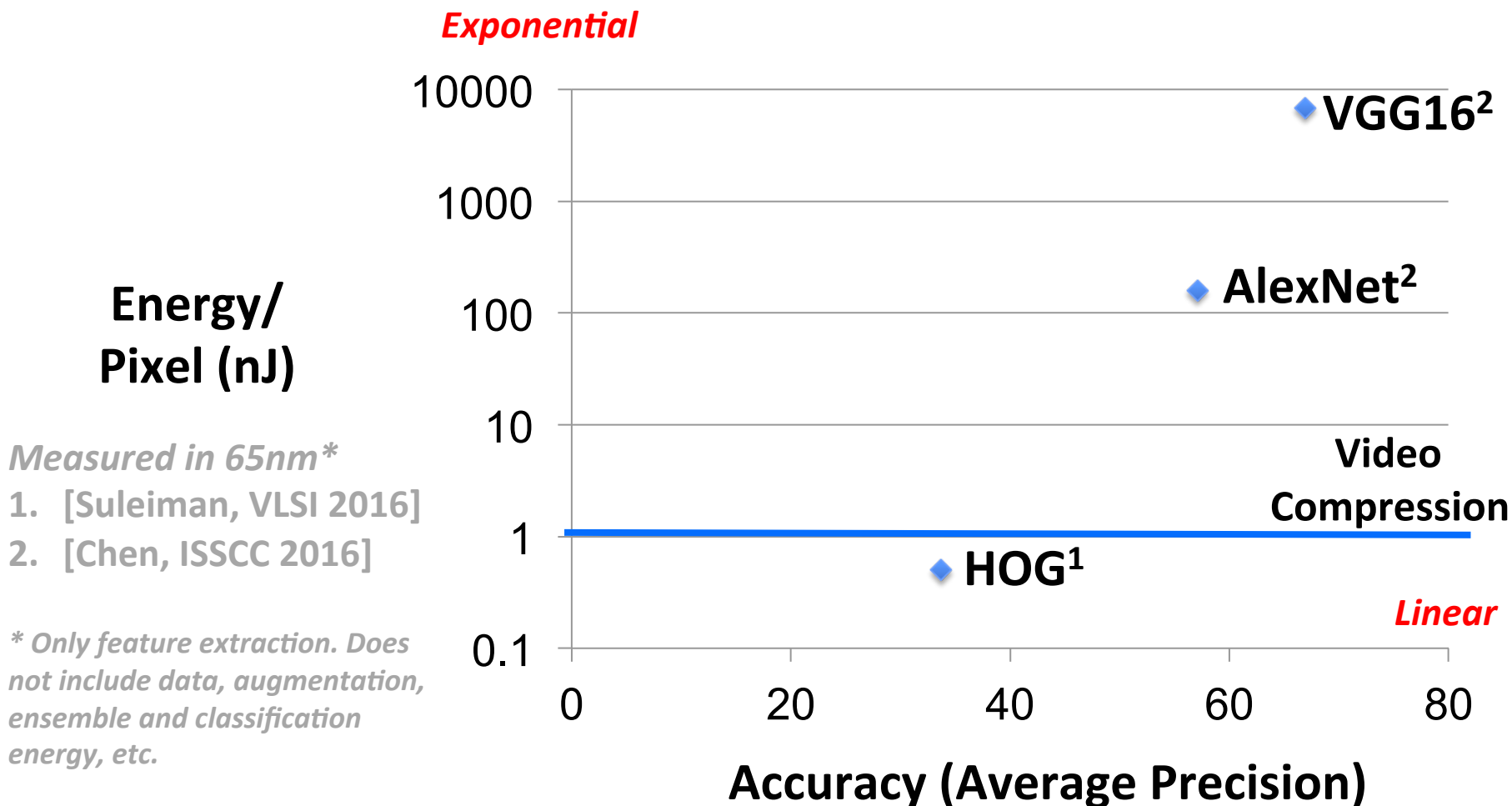
Comparison with GPU

	<i>Eyeriss</i>	NVIDIA TK1 (Jetson Kit)
Technology	65nm	28nm
Clock Rate	200MHz	852MHz
# Multipliers	168	192
On-Chip Storage	Buffer: 108KB Spad: 75.3KB	Shared Mem: 64KB Reg File: 256KB
Word Bit-Width	16b Fixed	32b Float
Throughput¹	34.7 fps	68 fps
Measured Power	278 mW	Idle/Active ² : 3.7W/10.2W
DRAM Bandwidth	127 MB/s	1120 MB/s ³

1. AlexNet Convolutional Layers Only
2. Board Power
3. Modeled from [Tan, SC11]

<http://eyeriss.mit.edu>

Features: Energy vs. Accuracy



[Suleiman et al., ISCAS 2017]

Opportunities in Joint Algorithm Hardware Design

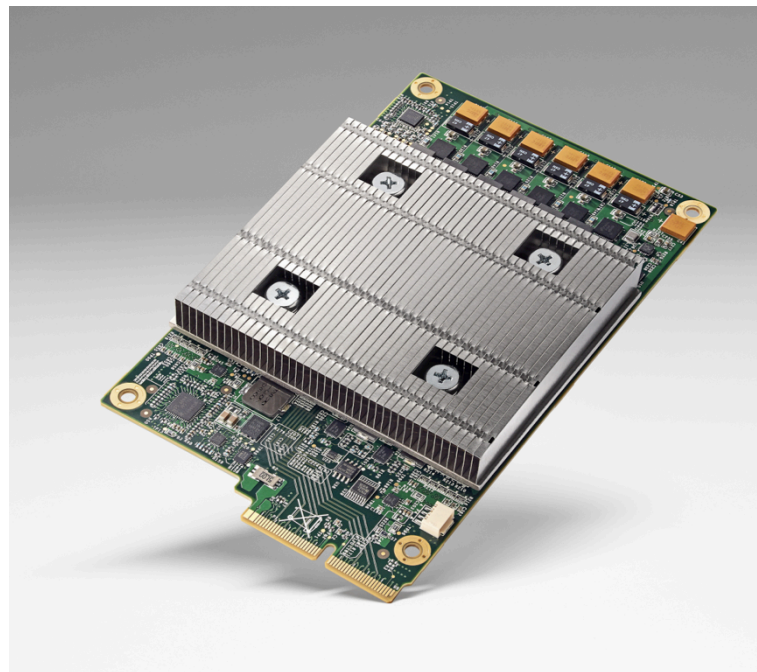
Approaches

- **Reduce size of operands for storage/compute**
 - Floating point → Fixed point
 - Bit-width reduction
 - Non-linear quantization
- **Reduce number of operations for storage/compute**
 - Exploit Activation Statistics (Compression)
 - Network Pruning
 - Compact Network Architectures

Commercial Products using 8-bit Integer



Nvidia's Pascal (2016)

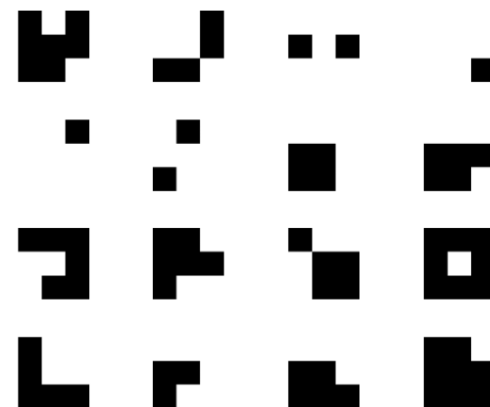


Google's TPU (2016)

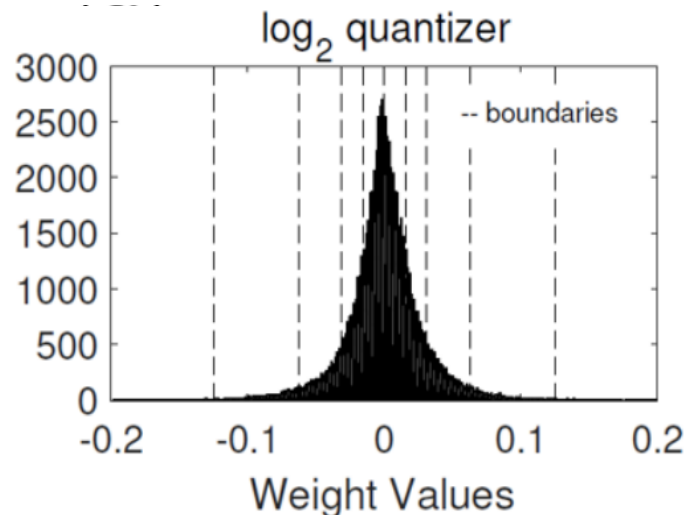
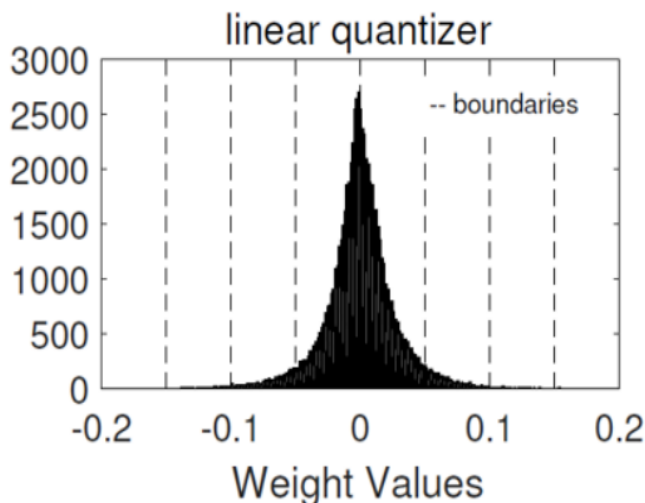
Reduced Precision in Research

- **Reduce number of bits**
 - Binary Nets [Courbariaux, NIPS 2015]
- **Reduce number of unique weights**
 - Ternary Weight Nets [Li, arXiv 2016]
 - XNOR-Net [Rategari, ECCV 2016]
- **Non-Linear Quantization**
 - LogNet [Lee, ICASSP 2017]

Binary Filters

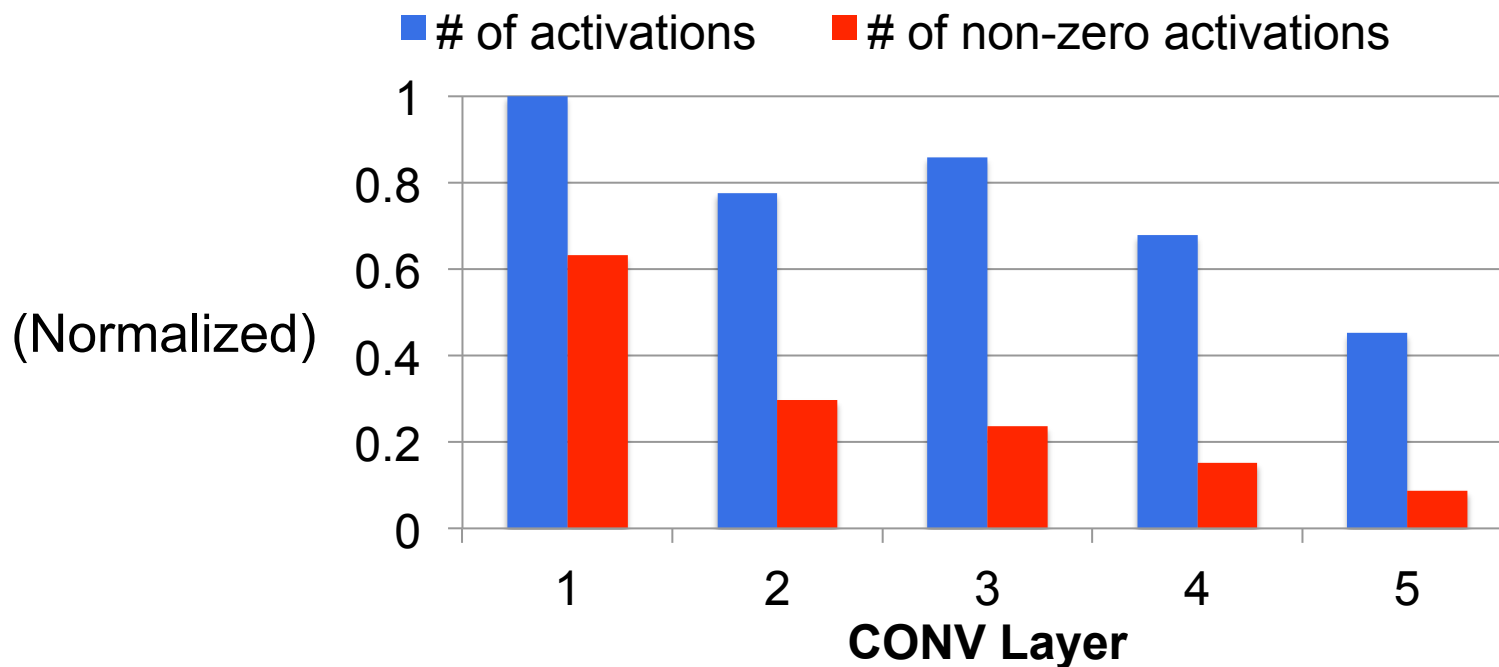
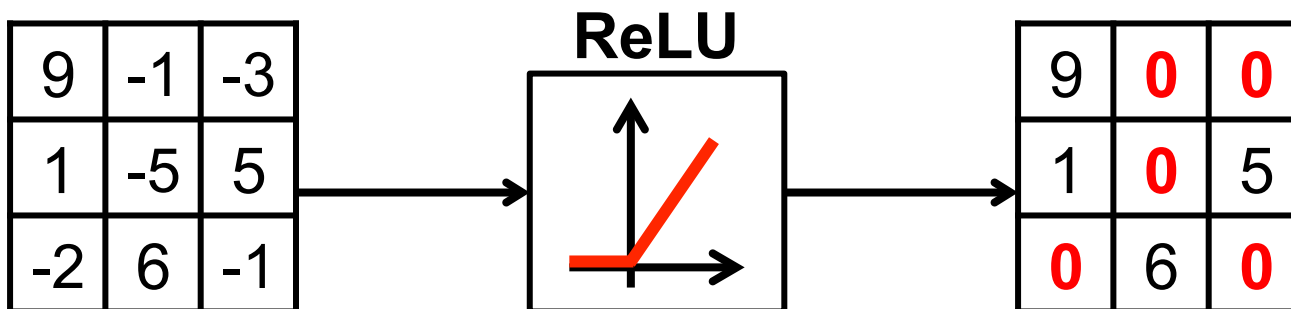


Log Domain Quantization



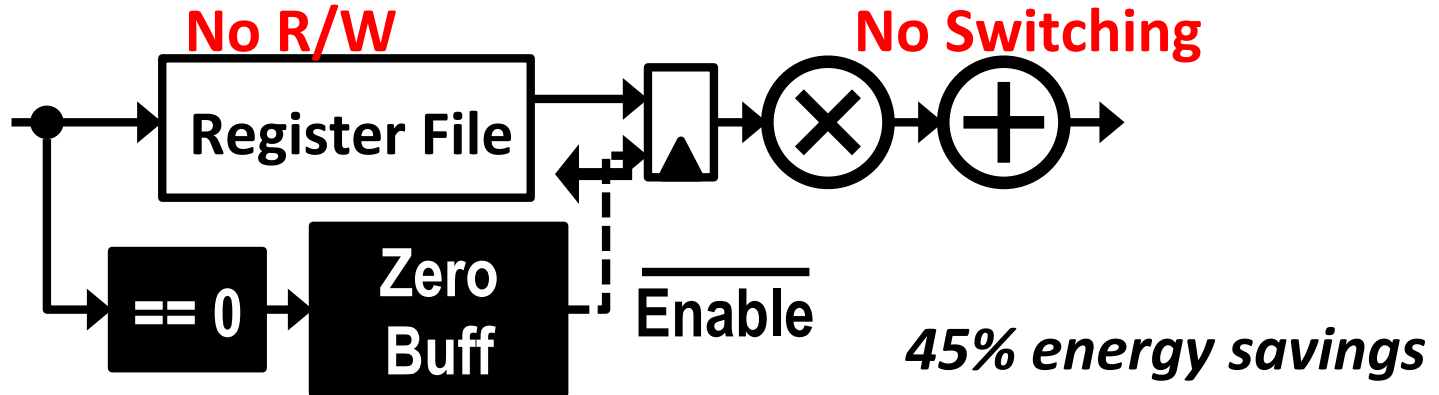
Sparsity in Feature Maps

Many **zeros** in output fmaps after ReLU

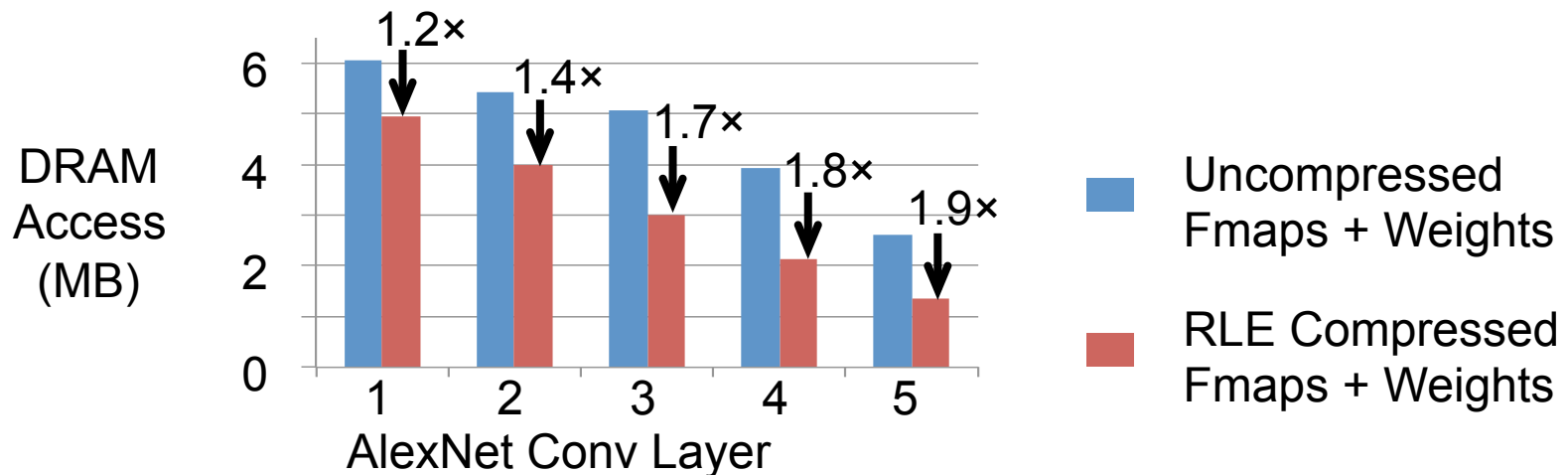


Exploit Sparsity

Method 1: Skip memory access and computation



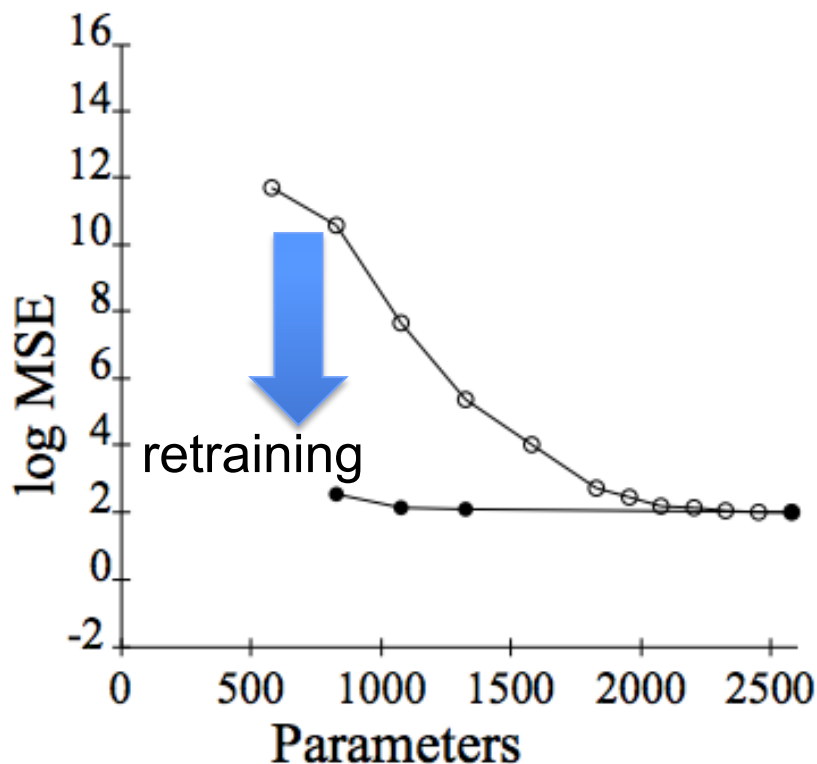
Method 2: Compress data to reduce storage and data movement



Pruning – Make Weights Sparse

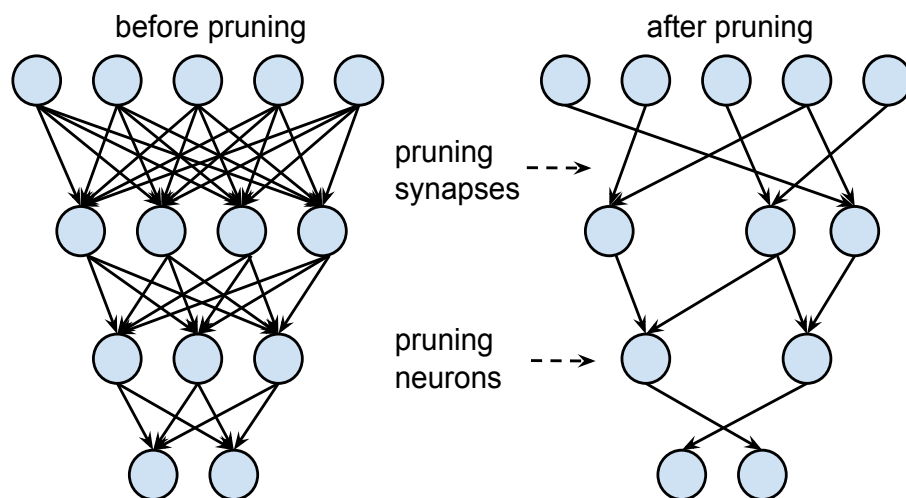
Optimal Brain Damage

[Lecun et al., NIPS 1989]



Prune DNN based on ***magnitude*** of weights

[Han et al., NIPS 2015]



Example: AlexNet

Weight Reduction:

CONV layers 2.7x, **FC layers 9.9x**

Overall Reduction:

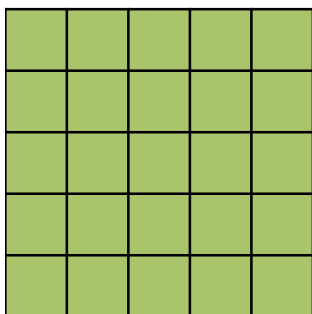
Weights 9x, MACs 3x

Network Architecture Design

Build Network with series of Small Filters

GoogleNet/Inception v3

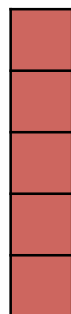
5x5 filter



decompose



5x1 filter

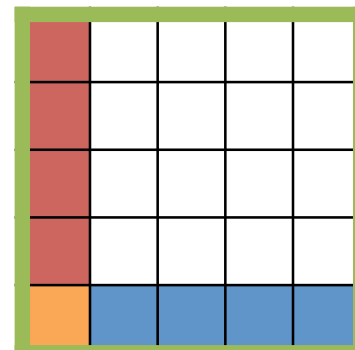


1x5 filter



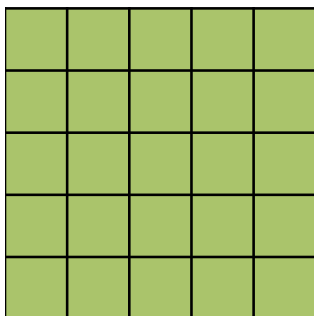
*separable
filters*

Apply sequentially



VGG-16

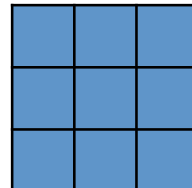
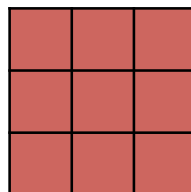
5x5 filter



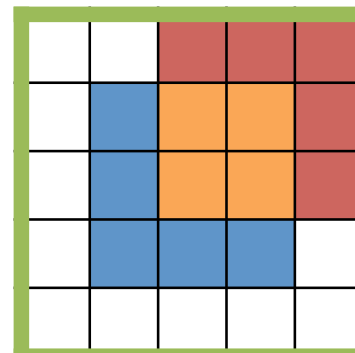
decompose



Two 3x3 filters

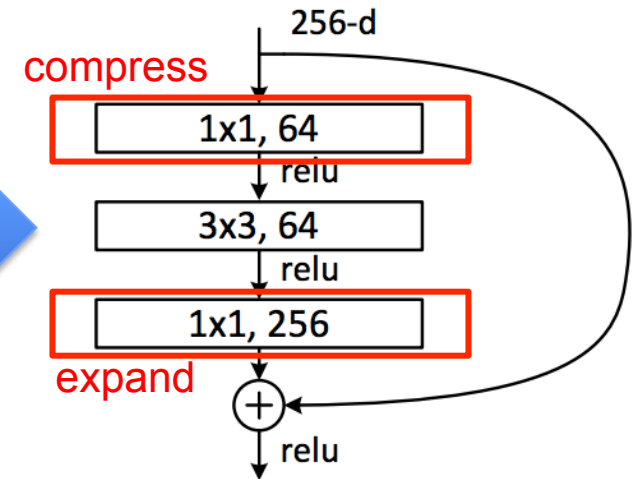
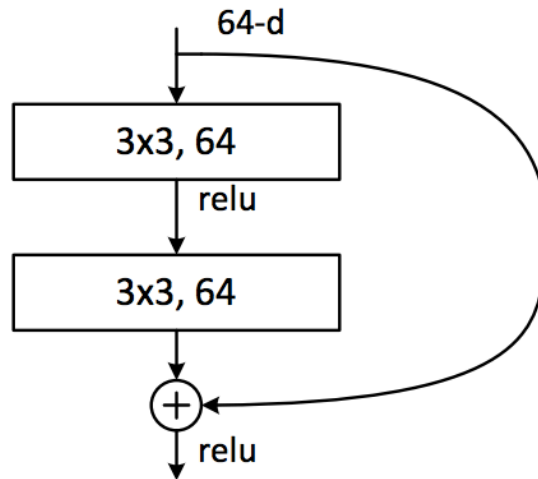


Apply sequentially

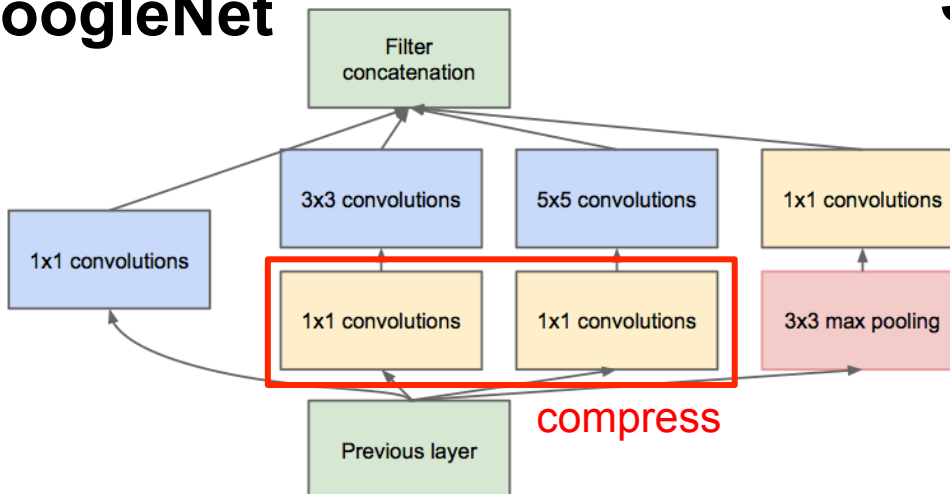


1x1 Bottleneck in Popular DNN models

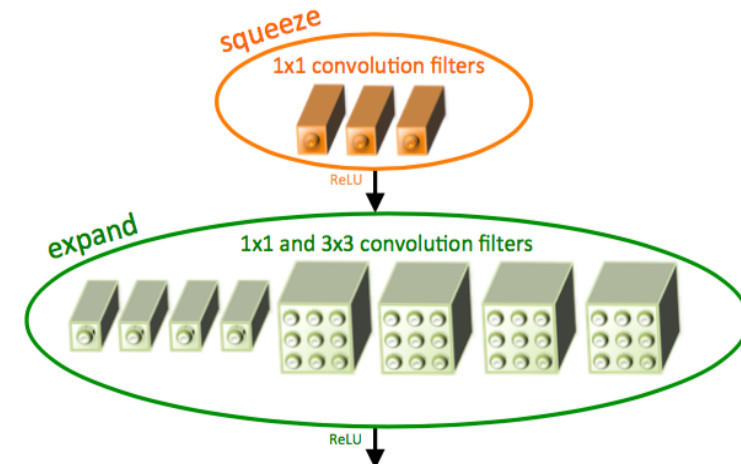
ResNet



GoogleNet



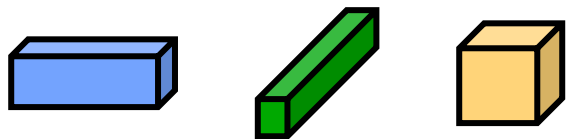
SqueezeNet



Key Metrics for Embedded DNN

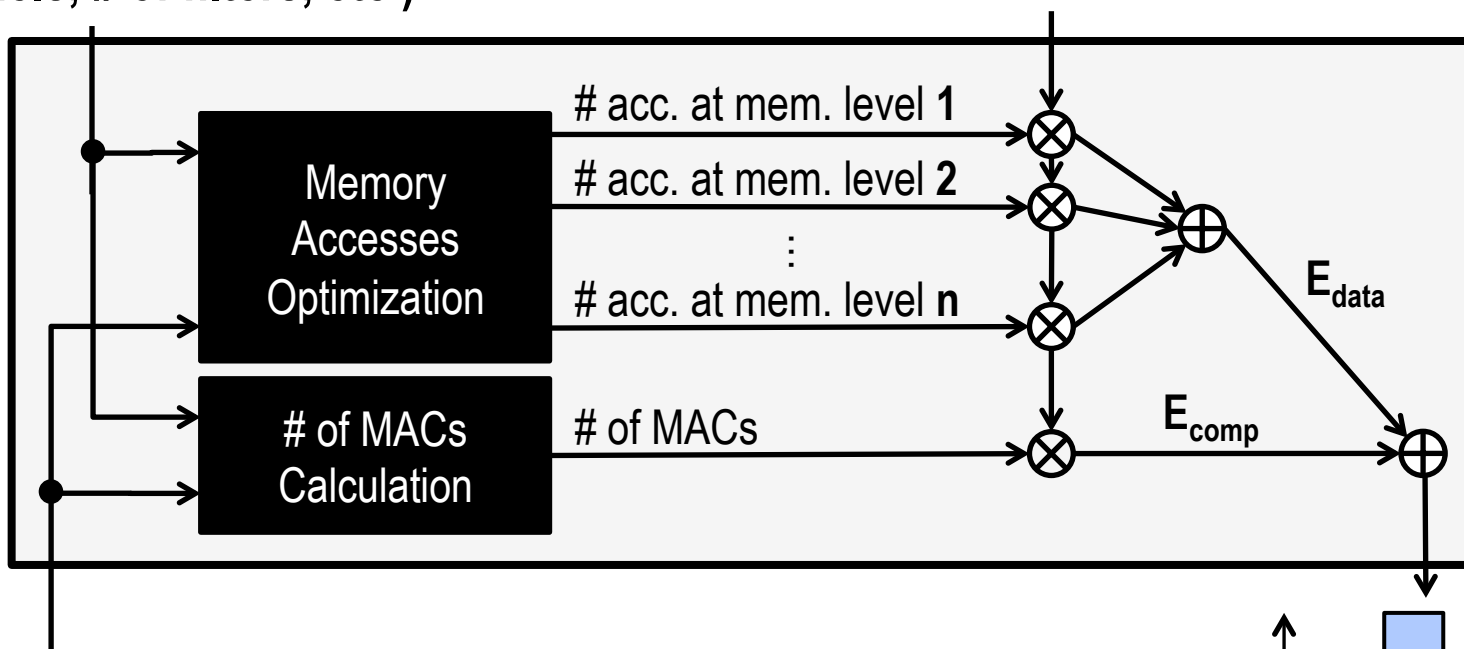
- Accuracy → Measured on Dataset
- Speed → Number of MACs
- Storage Footprint → Number of Weights
- Energy → ?

Energy-Evaluation Methodology



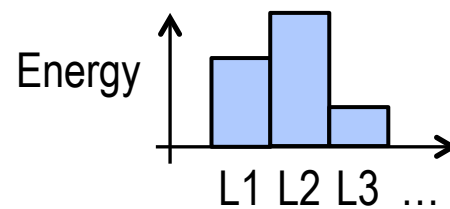
CNN Shape Configuration
(# of channels, # of filters, etc.)

**Hardware Energy Costs of each
MAC and Memory Access**



CNN Weights and Input Data

[0.3, 0, -0.4, 0.7, 0, 0, 0.1, ...]

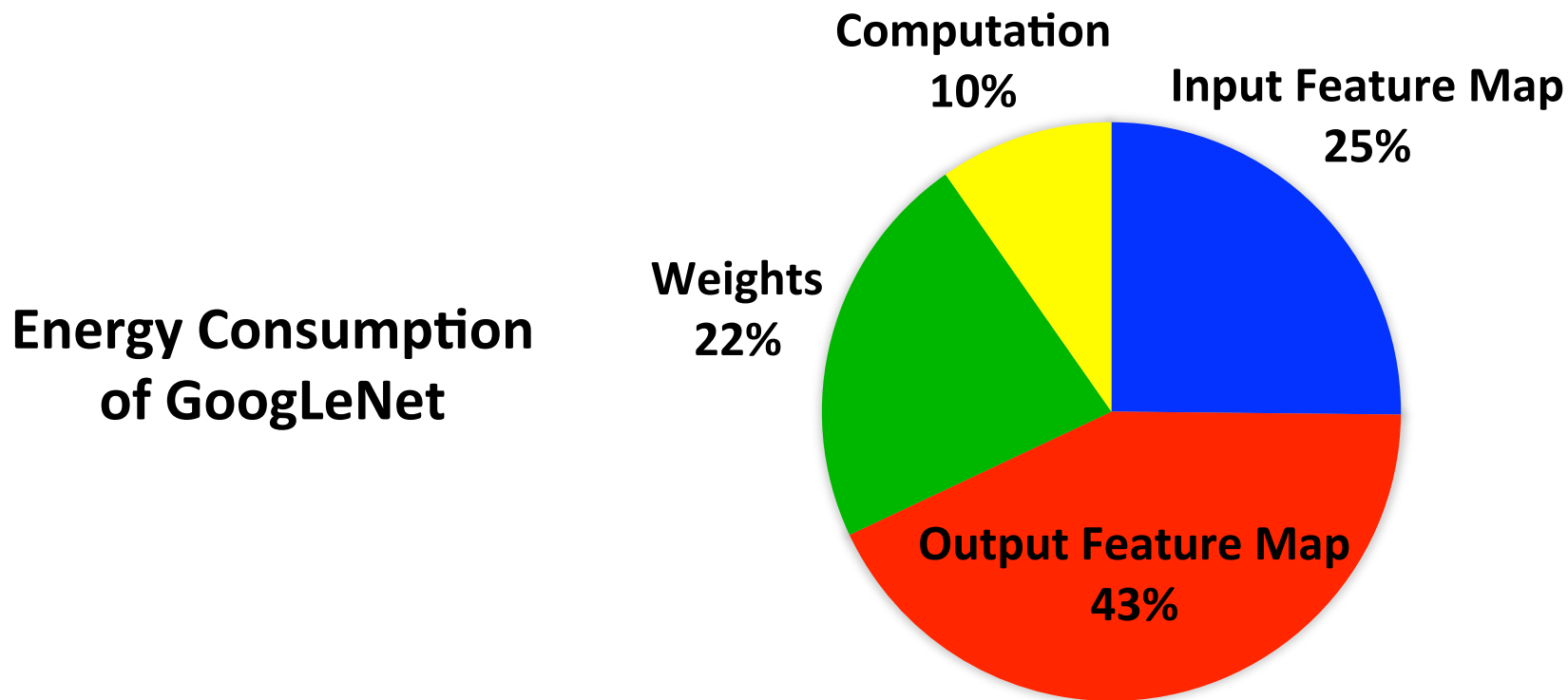


[Yang et al., CVPR 2017]

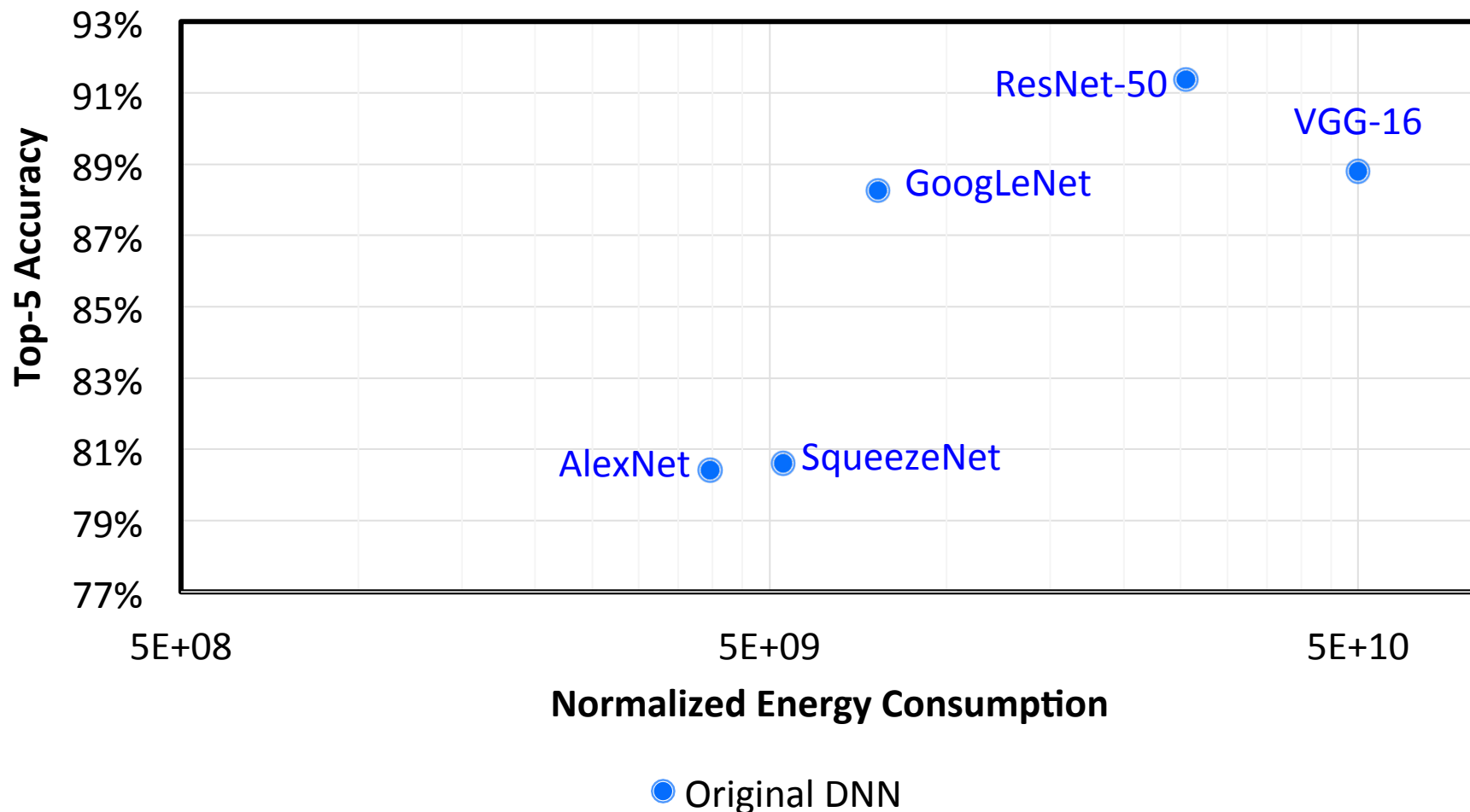
CNN Energy Consumption

Key Observations

- Number of weights *alone* is not a good metric for energy
- **All data types** should be considered

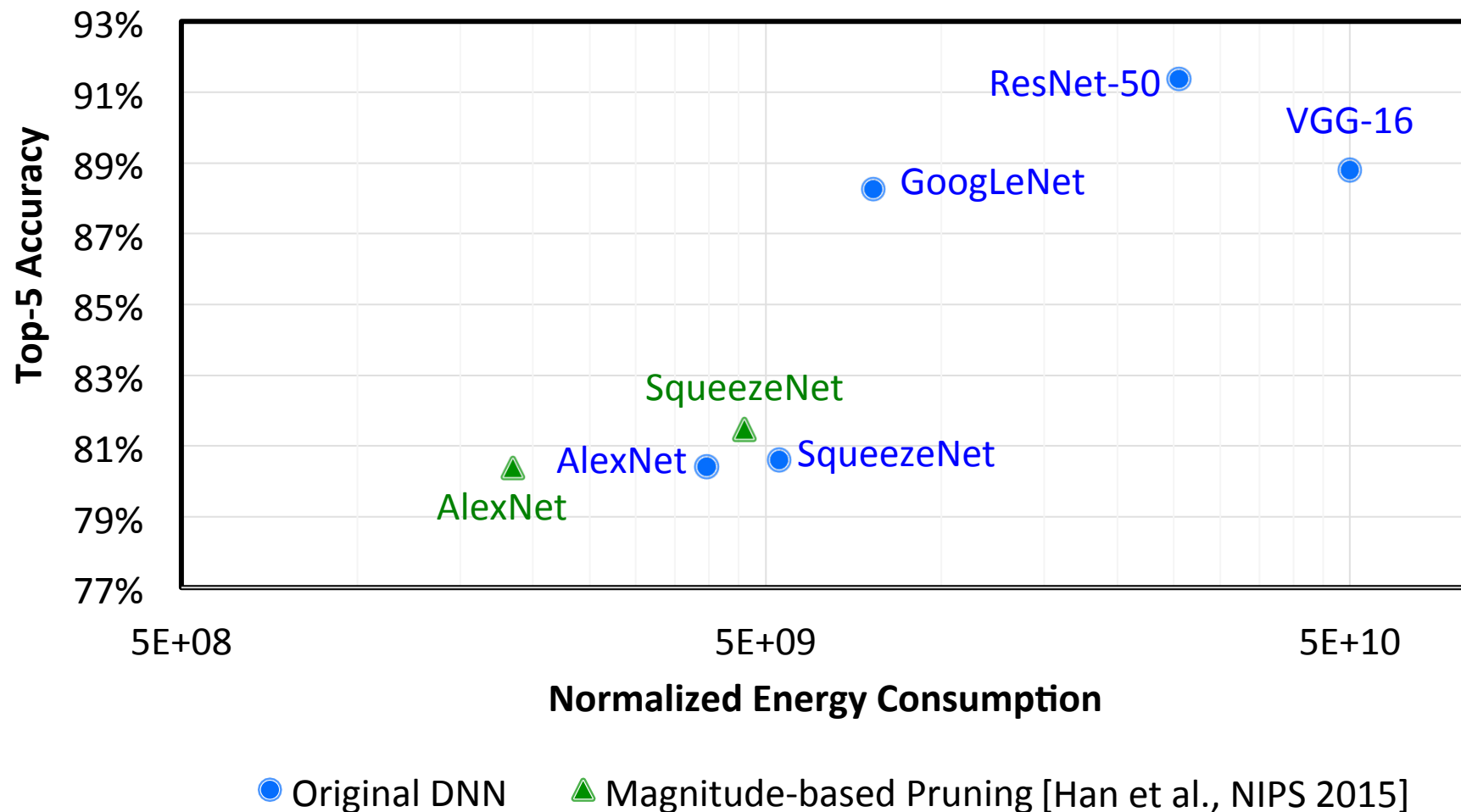


Energy Consumption of Existing DNNs



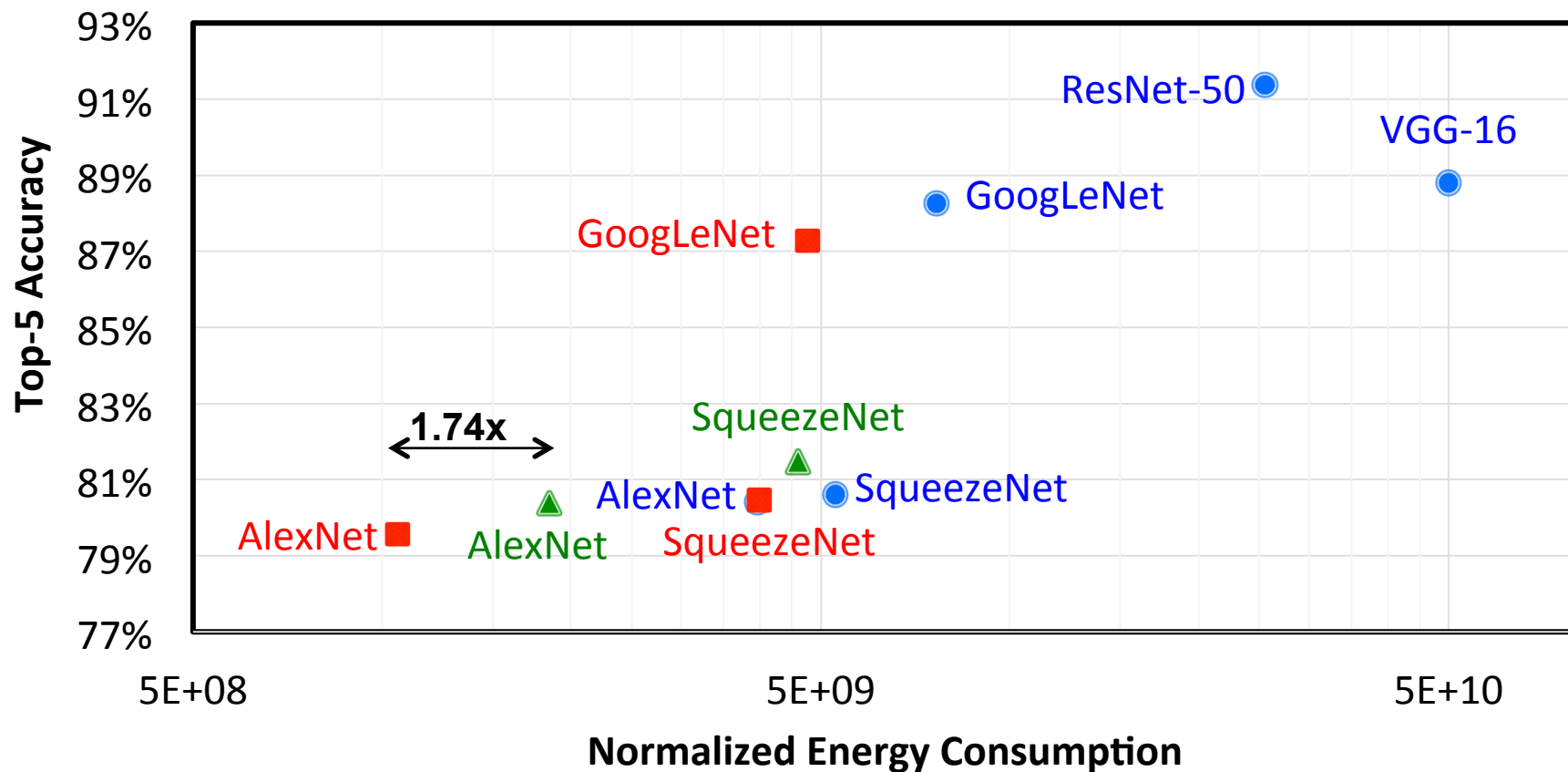
Deeper CNNs with fewer weights do not necessarily consume less energy than shallower CNNs with more weights

Magnitude-based Weight Pruning



Reduce number of weights by **removing small magnitude weights**

Energy-Aware Pruning



● Original DNN ▲ Magnitude-based Pruning ■ Energy-aware Pruning (This Work)

Remove weights from layers **in order of highest to lowest energy**
3.7x reduction in AlexNet / 1.6x reduction in GoogLeNet

Summary

- **Energy-Efficient Approaches**
 - Minimize data movement
 - Balance flexibility and energy-efficiency
 - Exploit sparsity with joint algorithm and hardware design
- **Joint algorithm and hardware design** can deliver additional energy savings (directly target energy)
- **Linear increase in accuracy** requires **exponential increase in energy**

Acknowledgements: This work is funded by the DARPA YFA grant, MIT Center for Integrated Circuits & Systems, and gifts from Intel, Nvidia and Google.

References

Overview Paper

V. Sze, Y.-H. Chen, T-J. Yang, J. Emer, “*Efficient Processing of Deep Neural Networks: A Tutorial and Survey*”, arXiv, 2017

<https://arxiv.org/pdf/1703.09039.pdf>

More info about **Eyeriss** and **Tutorial on DNN Architectures**

<http://eyeriss.mit.edu>

MIT Professional Education Course on
“**Designing Efficient Deep Learning Systems**”

March 26 – 27, 2018 in Mountain View, CA

<http://professional-education.mit.edu/deeplearning>

For updates



Follow @eems_mit

<http://mailman.mit.edu/mailman/listinfo/eems-news>