How to Understand and Evaluate Deep Learning Processors

Vivienne Sze
Massachusetts Institute of Technology

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

February 16, 2020

Contact Info
email: sze@mit.edu
website: http://sze.mit.edu
https://twitter.com/eems_mit

Goals of this Tutorial

- Many existing Deep Learning Processors. **Too many to cover!**

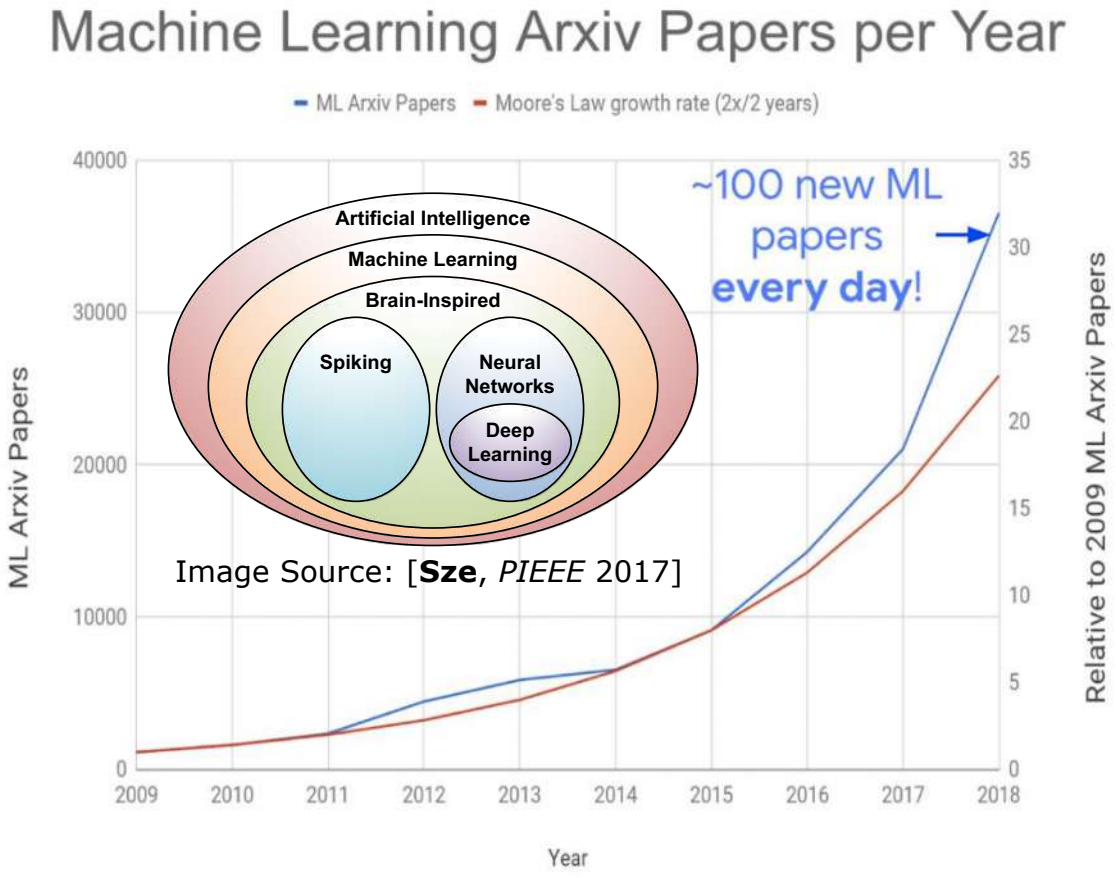


Image Source: [**Sze**, *PIEEE* 2017]

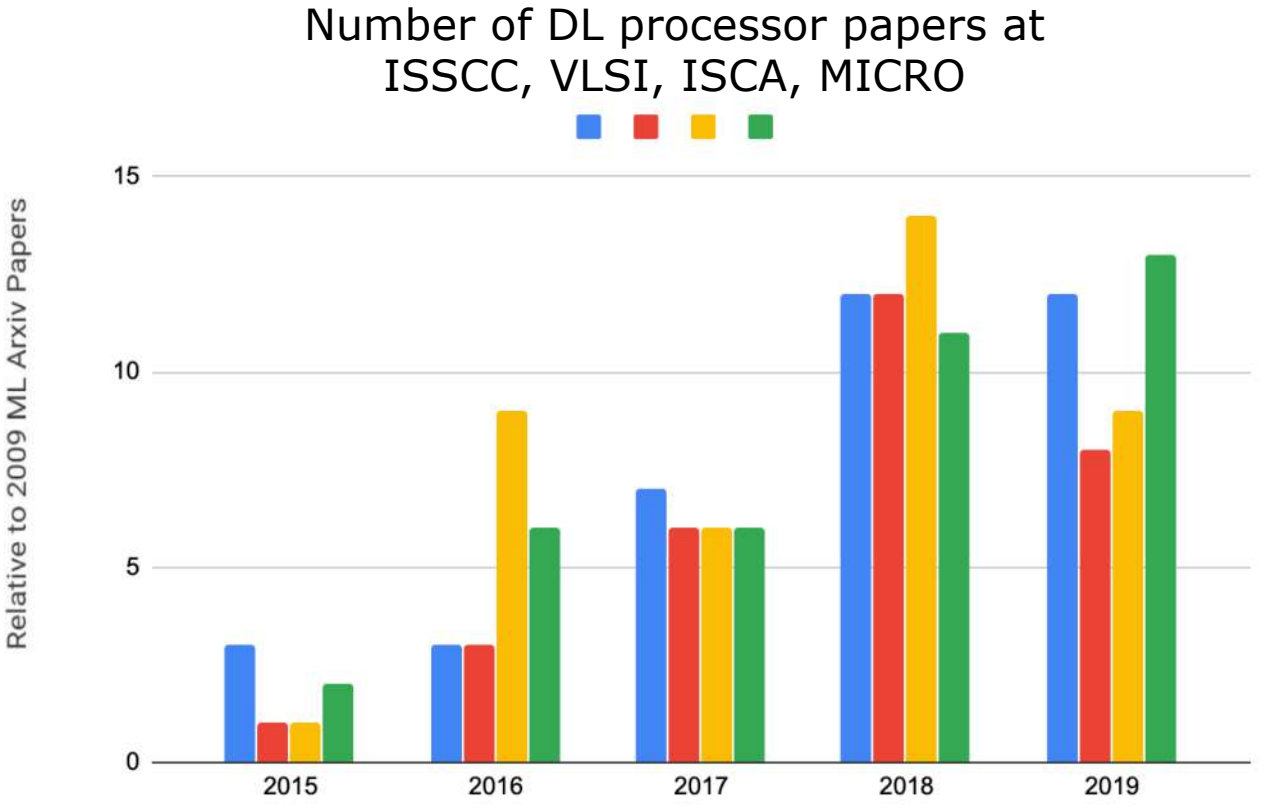


Image Source: [**Dean**, *ISSCC* 2020]

Goals of this Tutorial

- Many existing Deep Learning Processors. **Too many to cover!**

- In this tutorial, we focus on how to **evaluate** DL processors
  - What are the key questions to ask?

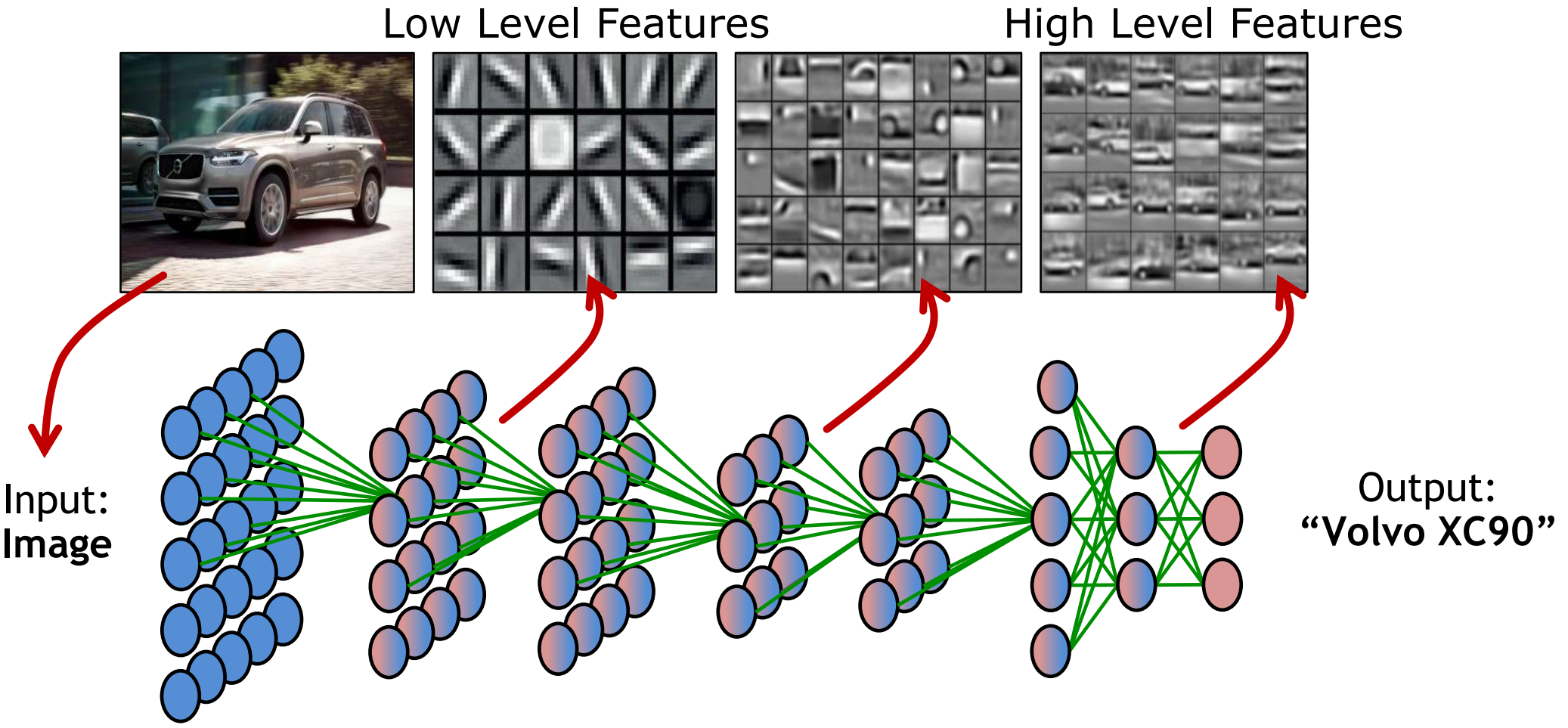
- Specifically, we will discuss
  - What are the **key metrics** that should be measured and compared?
  - What are the **challenges** towards achieving these metrics?
  - What are the **design considerations** and tradeoffs?
  - How do these challenges and design considerations **differ across platforms** (e.g., CPU, GPU, ASIC, PIM, FPGA)?

- We will focus on inference, but many concepts covered also apply to training

Tutorial Overview

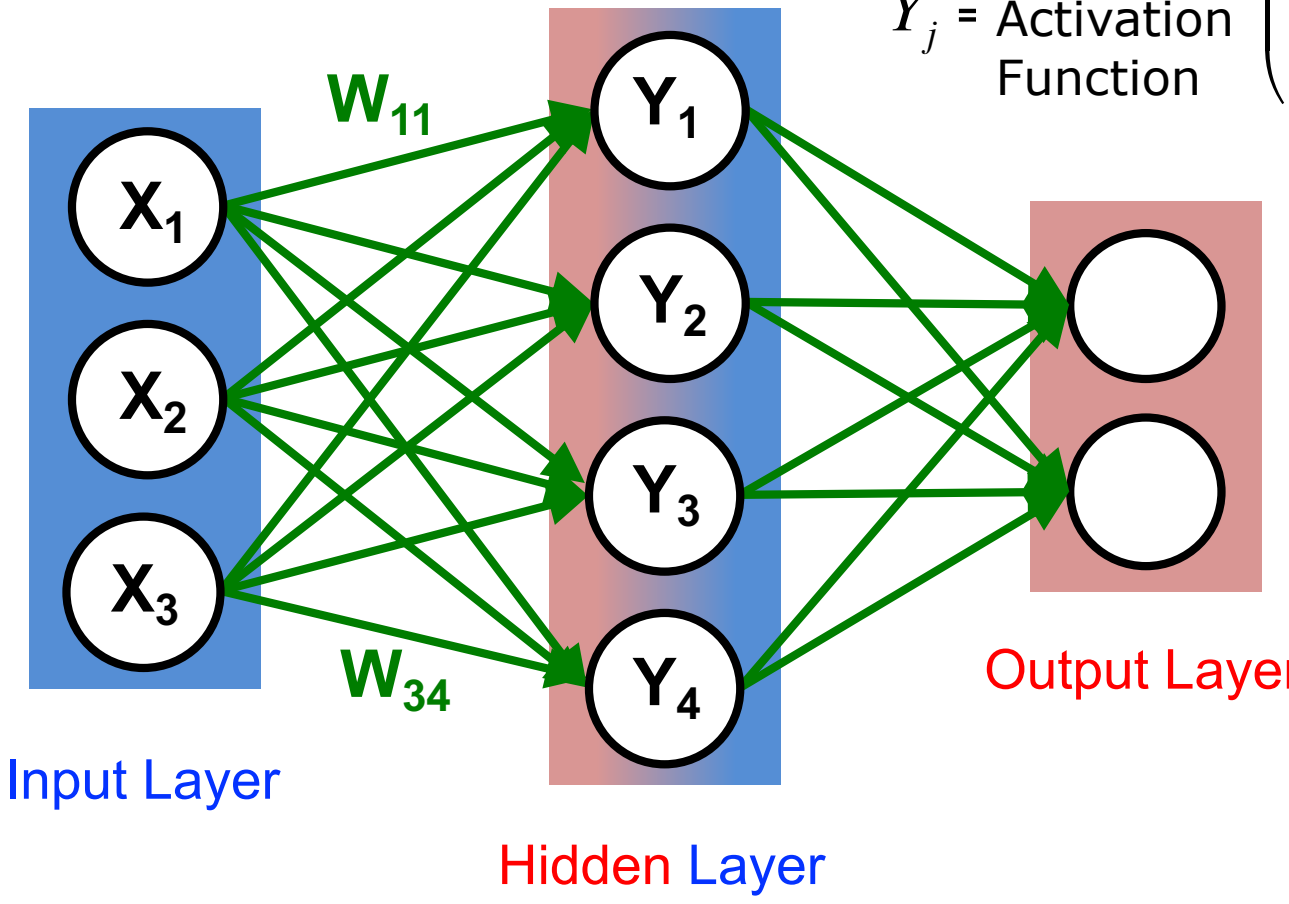
- Deep Learning Overview
- Key Metrics and Design Objectives
- Design Considerations
  - CPU and GPU Platforms
  - Specialized / Domain Specific Hardware (ASICs)
    - Efficient Dataflows
    - Algorithm (DNN Model) and Hardware Co-Design
    - Flexibility and Scalability
  - Other Platforms
    - Processing In Memory / In Memory Computing
    - Field Programmable Gate Arrays (FPGAs)
- Tools for Systematic Evaluation of DL Processors
- Should I Use Deep Learning for a Given Task?

What is Deep Learning (aka Deep Neural Networks)?

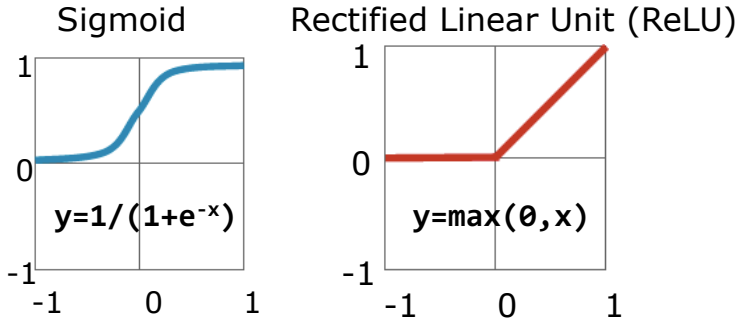


Modified Image Source: [**Lee**, *CACM* 2011]

Weighted Sums



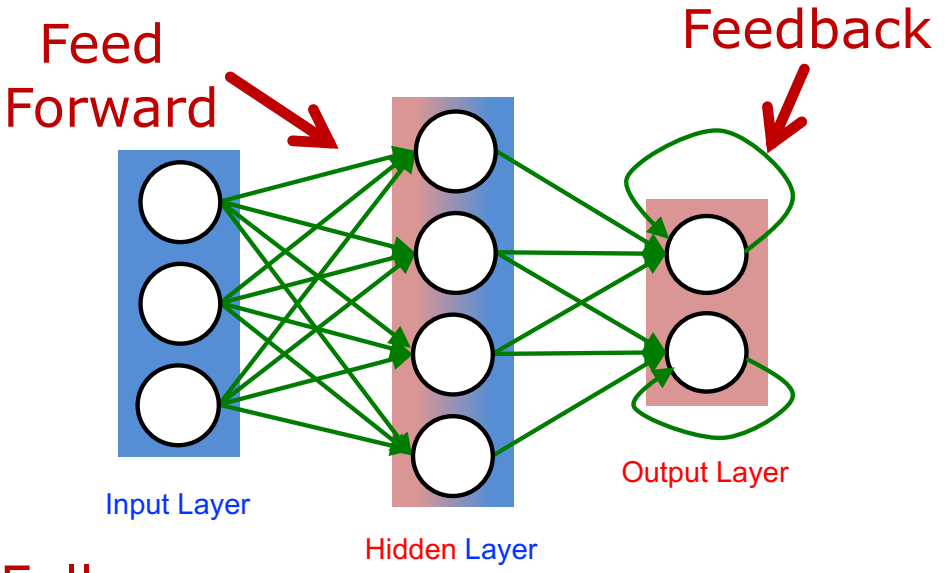
$$Y_j = \text{Nonlinear Activation Function}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$

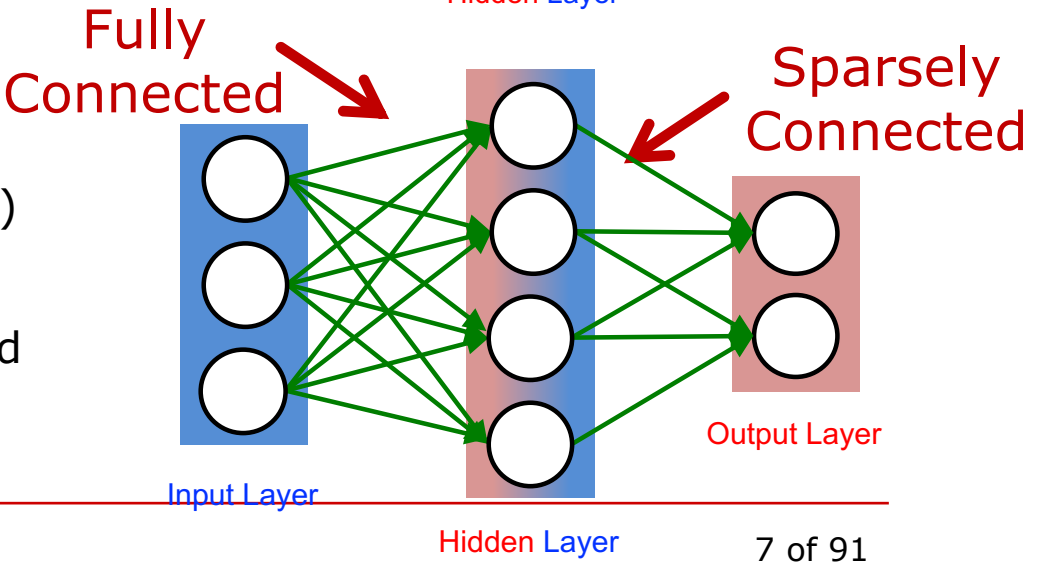


Key operation is
multiply and accumulate (MAC)
Accounts for > 90% of computation

Popular Types of Layers in DNNs

- **Fully Connected Layer**
  - Feed forward, fully connected
  - Multilayer Perceptron (MLP)
- **Convolutional Layer**
  - Feed forward, sparsely-connected w/ weight sharing
  - Convolutional Neural Network (CNN)
  - Typically used for images
- **Recurrent Layer**
  - Feedback
  - Recurrent Neural Network (RNN)
  - Typically used for sequential data (e.g., speech, language)
- **Attention Layer/Mechanism**
  - Attention (matrix multiply) + feed forward, fully connected
  - Transformer [**Vaswani**, *NeurIPS* 2017]

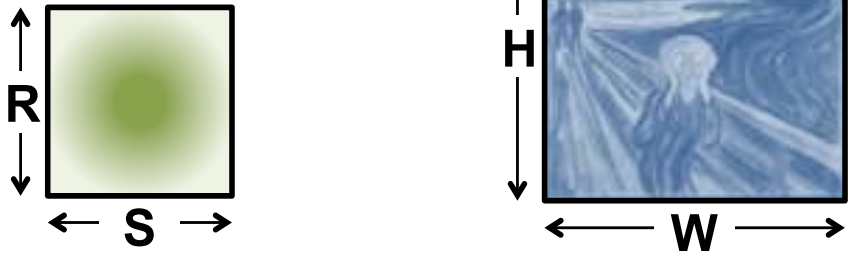




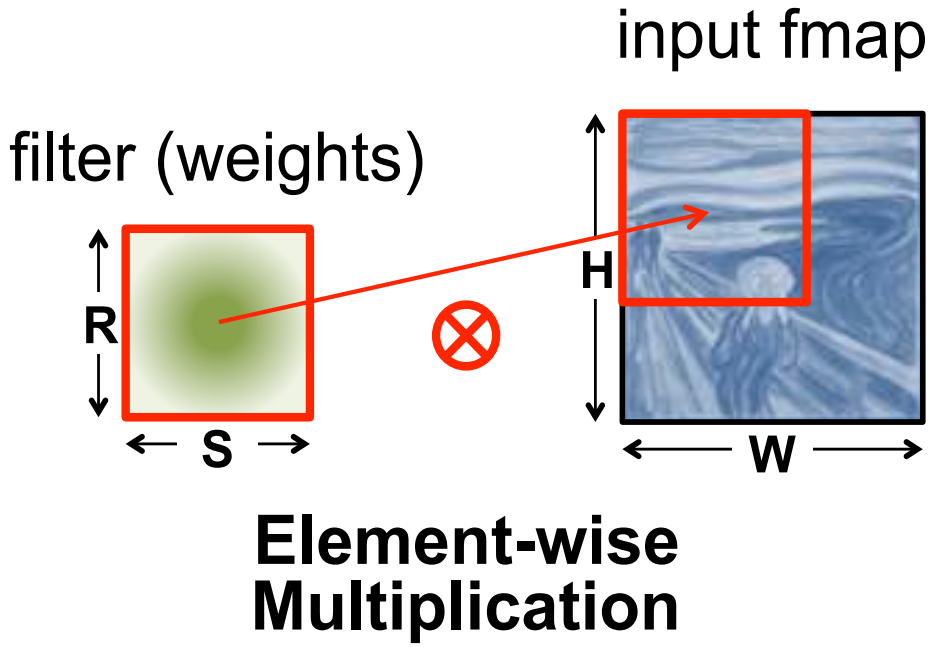
High-Dimensional Convolution in CNN

a plane of input activations
a.k.a. **input feature map (fmap)**

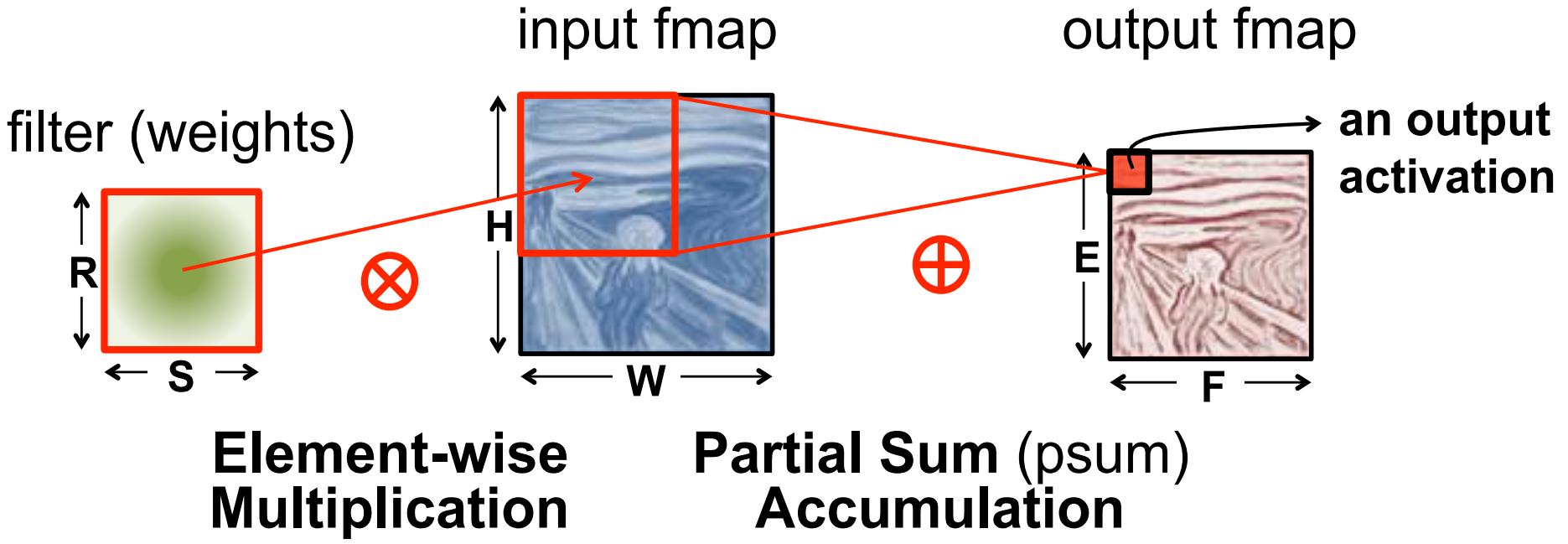
filter (weights)



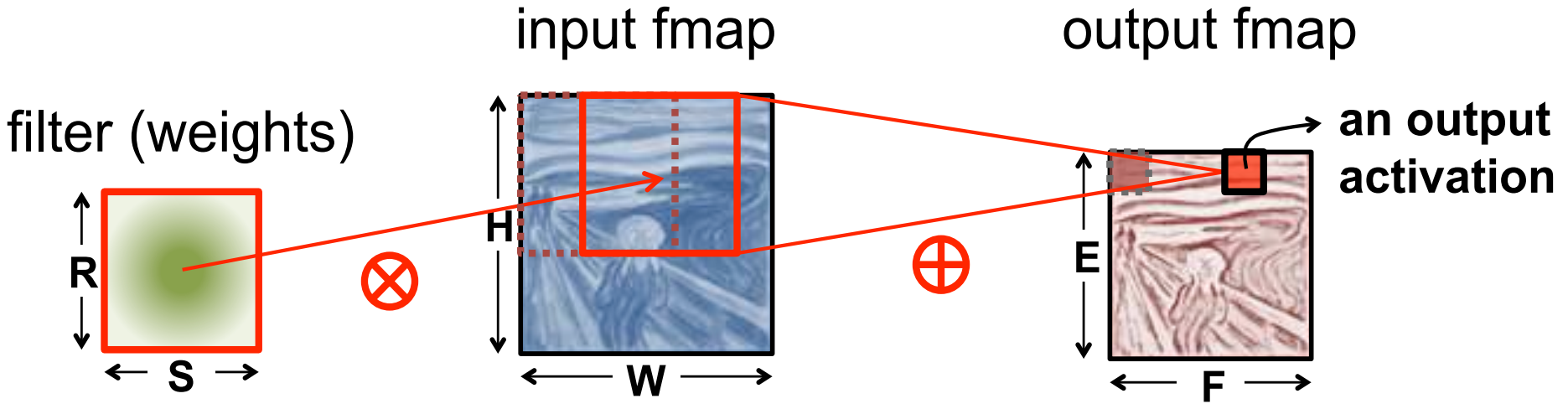
High-Dimensional Convolution in CNN



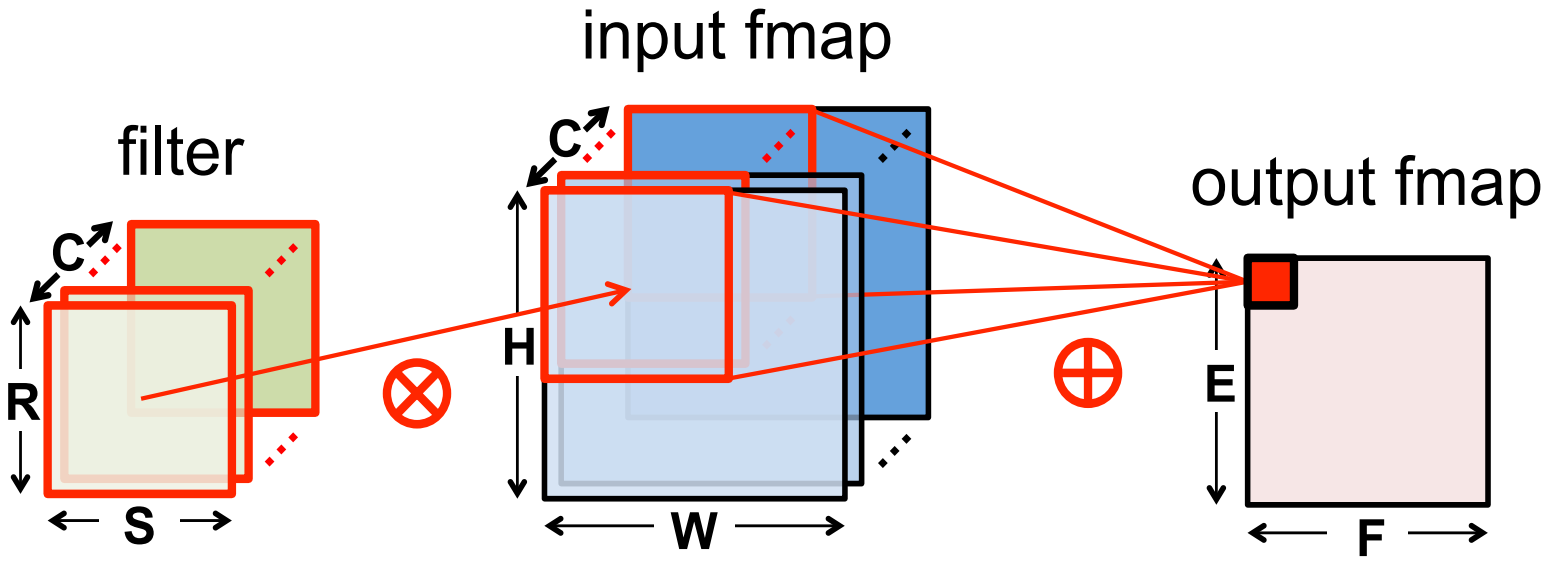
High-Dimensional Convolution in CNN



High-Dimensional Convolution in CNN



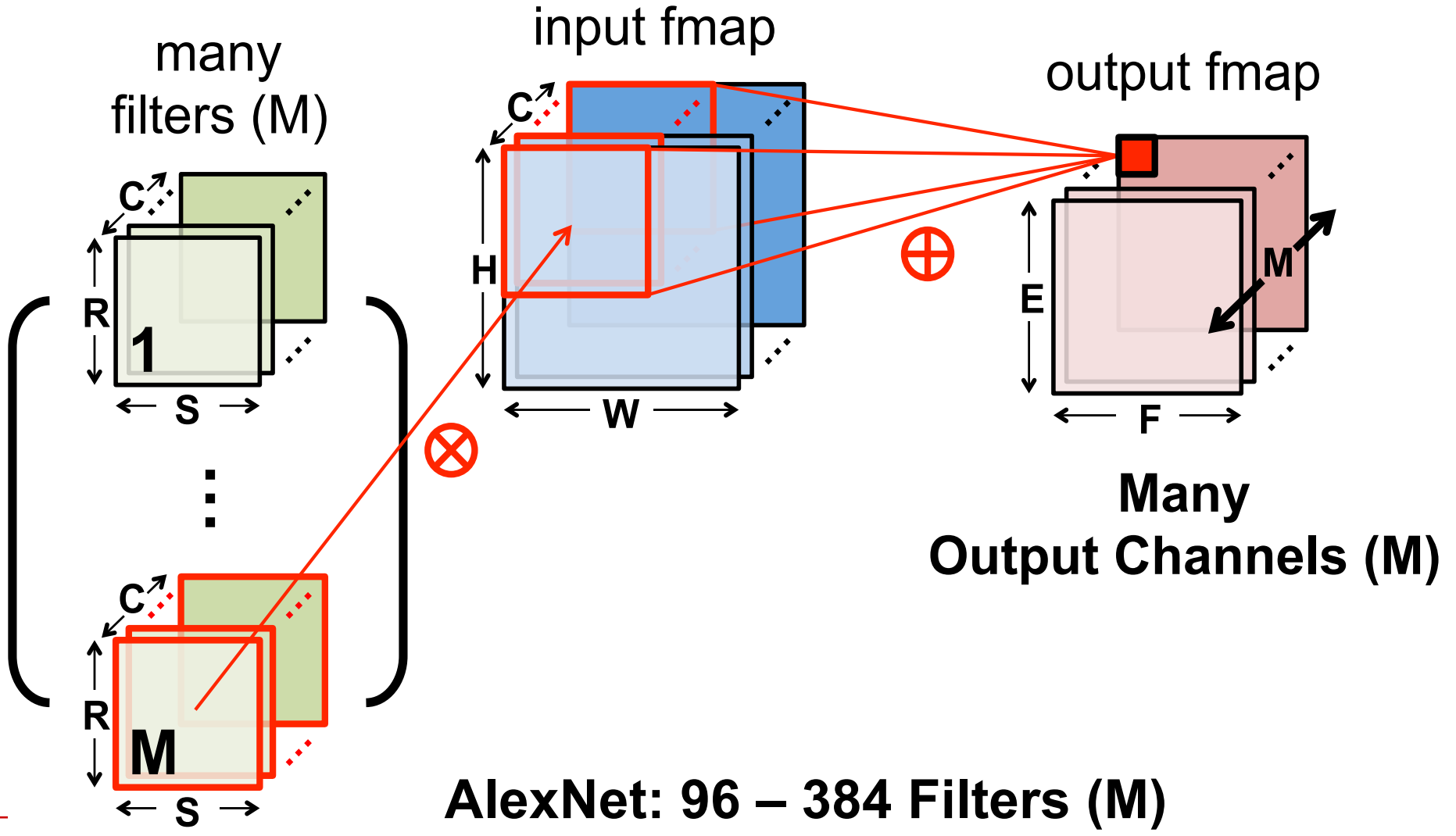
High-Dimensional Convolution in CNN



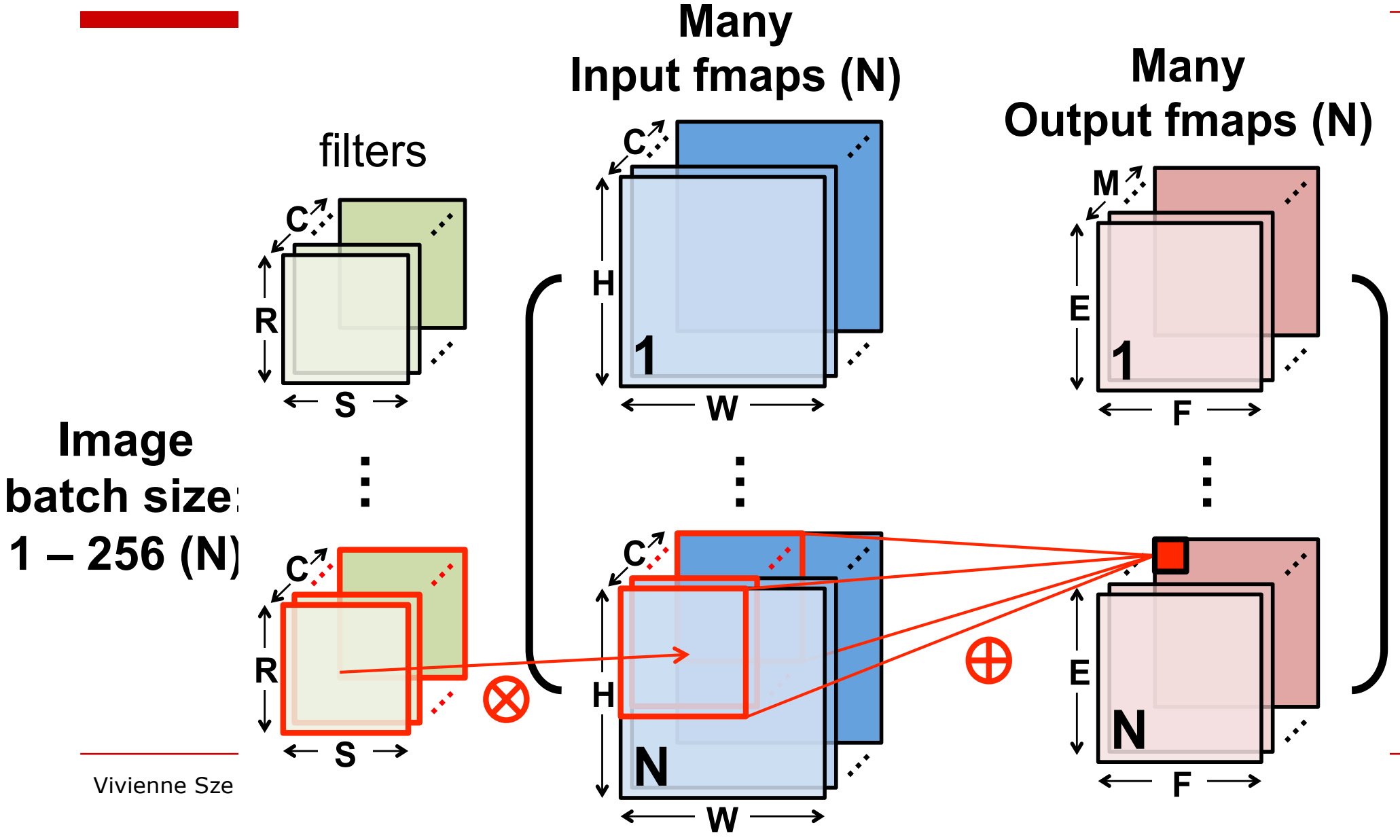
Many Input Channels (C)

AlexNet: 3 – 192 Channels (C)

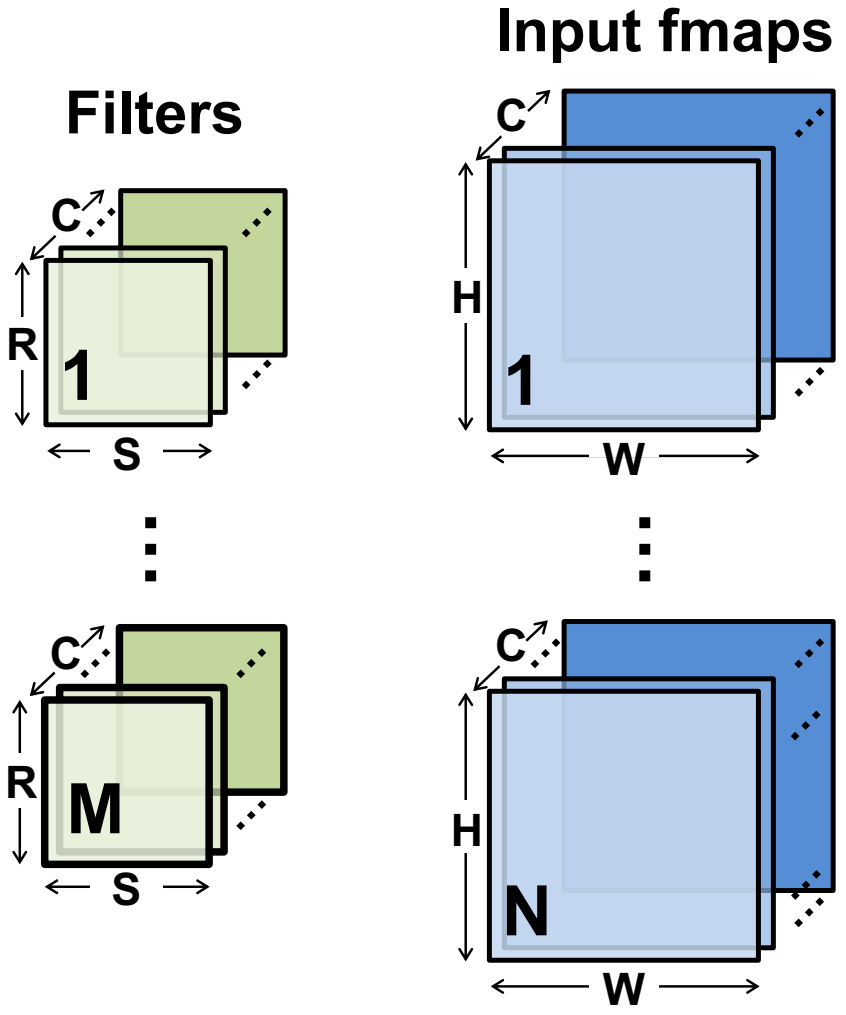
High-Dimensional Convolution in CNN



High-Dimensional Convolution in CNN



Define Shape for Each Layer



Shape **varies** across layers

H – Height of input fmap (activations)
W – Width of input fmap (activations)
C – Number of 2-D input fmaps /filters (channels)
R – Height of 2-D filter (weights)
S – Width of 2-D filter (weights)
M – Number of 2-D output fmaps (channels)
E – Height of output fmap (activations)
F – Width of output fmap (activations)
N – Number of input fmaps/output fmaps (batch size)

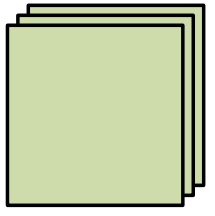
Large Sizes with Varying Shapes

AlexNet 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

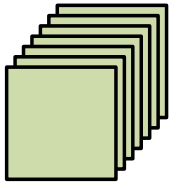
[**Krizhevsky**, *NeurIPS* 2012]

Layer 1



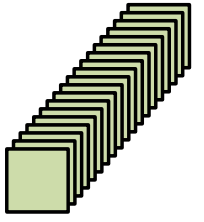
34k Params
105M MACs

Layer 2



307k Params
224M MACs

Layer 3



885k Params
150M MACs

Popular DNNs

Metrics	LeNet-5	AlexNet	VGG-16	GoogLeNet (v1)	ResNet-50	EfficientNet-B4
Top-5 error (ImageNet)	n/a	16.4	7.4	6.7	5.3	3.7*
Input Size	28x28	227x227	224x224	224x224	224x224	380x380
# of CONV Layers	**2**	**5**	**16**	**21 (depth)**	**49**	**96**
# of Weights	2.6k	2.3M	14.7M	6.0M	23.5M	14M
# of MACs	283k	666M	15.3G	1.43G	3.86G	4.4G
# of FC layers	**2**	**3**	**3**	**1**	**1**	**65****
# of Weights	58k	58.6M	124M	1M	2M	4.9M
# of MACs	58k	58.6M	124M	1M	2M	4.9M
Total Weights	**60k**	**61M**	**138M**	**7M**	**25.5M**	**19M**
Total MACs	**341k**	**724M**	**15.5G**	**1.43G**	**3.9G**	**4.4G**
Reference	**Lecun**, *PIEEE* 1998	**Krizhevsky**, *NeurIPS* 2012	**Simonyan**, *ICLR* 2015	**Szegedy**, *CVPR* 2015	**He**, *CVPR* 2016	**Tan**, *ICML* 2019

DNN models getting **larger** and **deeper**

* Does not include multi-crop and ensemble
** Increase in FC layers due to squeeze-and-excitation layers (much smaller than FC layers for classification)

Tutorial Overview

- Deep Learning Overview
- **Key Metrics and Design Objectives**
- Design Considerations
  - CPU and GPU Platforms
  - Specialized / Domain Specific Hardware (ASICs)
    - Efficient Dataflows
    - Algorithm (DNN Model) and Hardware Co-Design
    - Flexibility and Scalability
  - Other Platforms
    - Processing In Memory / In Memory Computing
    - Field Programmable Gate Arrays (FPGAs)
- Tools for Systematic Evaluation of DL Processors
- Should I Use Deep Learning for a Given Task?

Key Metrics: Much more than TOPS/W!

- **Accuracy**
  - Quality of result
- **Throughput**
  - Analytics on high volume data
  - Real-time performance (e.g., video at 30 fps)
- **Latency**
  - For interactive applications (e.g., autonomous navigation)
- **Energy and Power**
  - Embedded devices have limited battery capacity
  - Data centers have power ceiling due to cooling cost
- **Hardware Cost**
  - $$$
- **Flexibility**
  - Range of DNN models and tasks
- **Scalability**
  - Scaling of performance with amount of resources

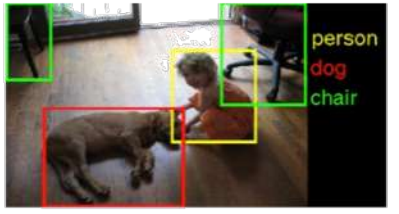
MNIST

ImageNet

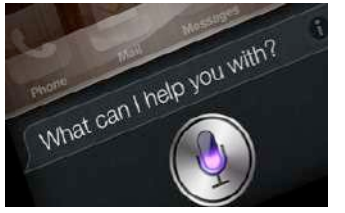
Embedded Device / Low Latency

Data Center / High Throughput

Computer Vision



Speech Recognition



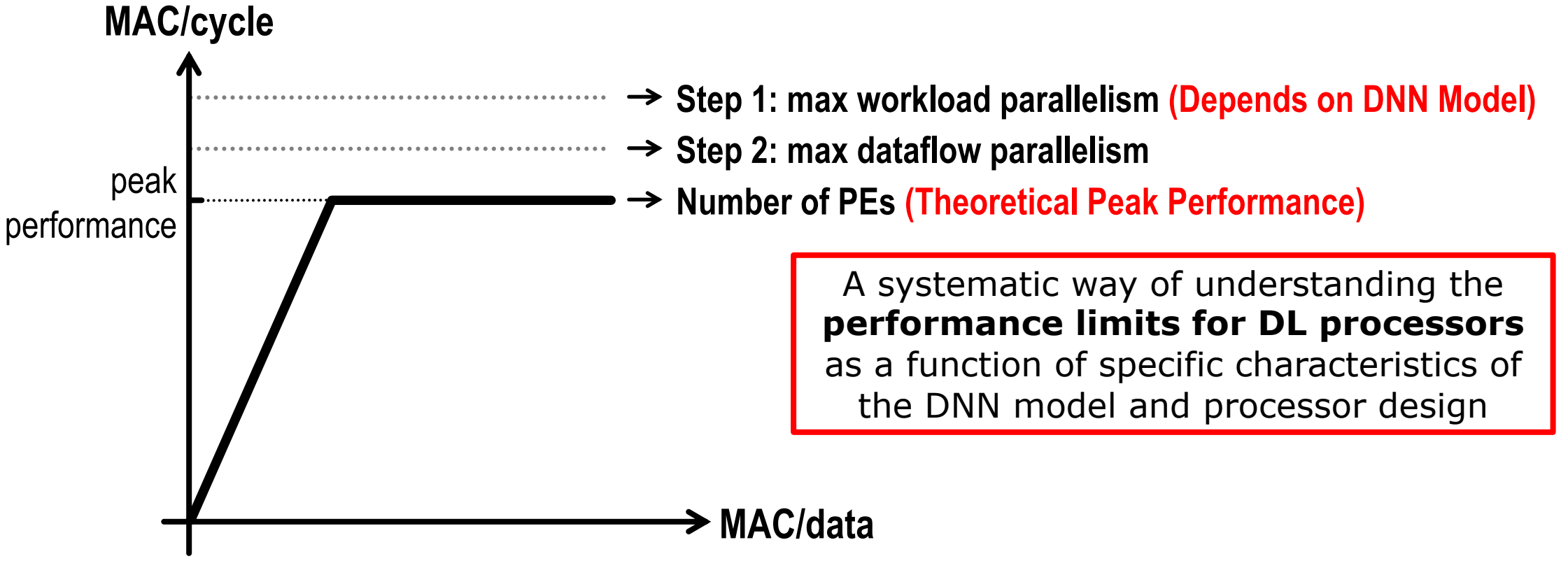
[**Sze**, *CICC* 2017]

Key Design Objectives of DL Processors

- **Increase Throughput and Reduce Latency**
  - Reduce time per MAC
    - Reduce critical path → increase clock frequency
    - Reduce instruction overhead
  - Avoid unnecessary MACs (save cycles)
  - Increase number of processing elements (PE) → more MACs in parallel
    - Increase area density of PE or area cost of system
  - Increase PE utilization* → keep PEs busy
    - Distribute workload to as many PEs as possible
    - Balance the workload across PEs
    - Sufficient memory BW to deliver workload to PEs (reduce idle cycles)
- Low latency has an additional constraint of **small batch size**

*(100% = peak performance)

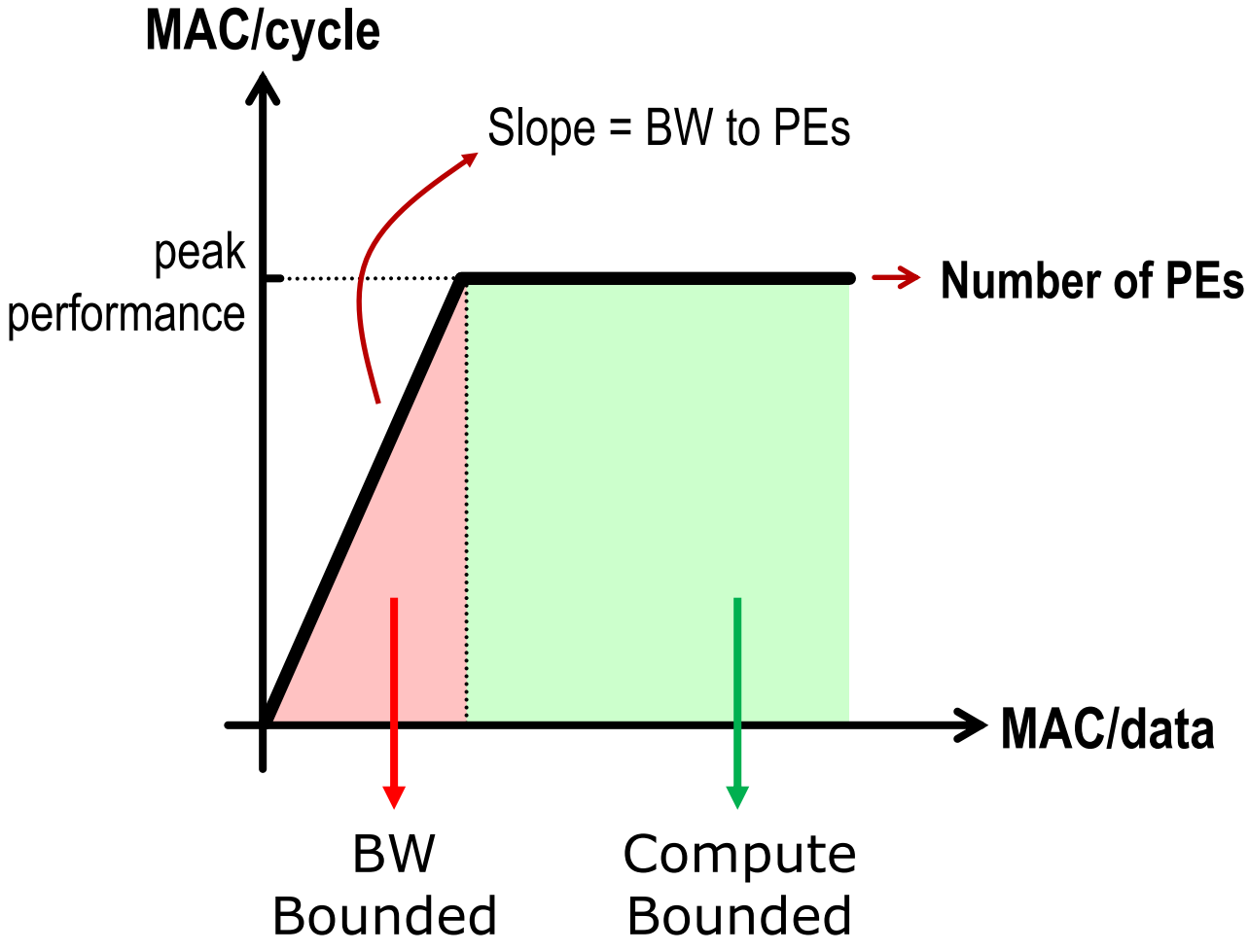
Eyexam: Performance Evaluation Framework



A systematic way of understanding the **performance limits for DL processors** as a function of specific characteristics of the DNN model and processor design

[**Chen**, *arXiv* 2019: https://arxiv.org/abs/1807.07928]

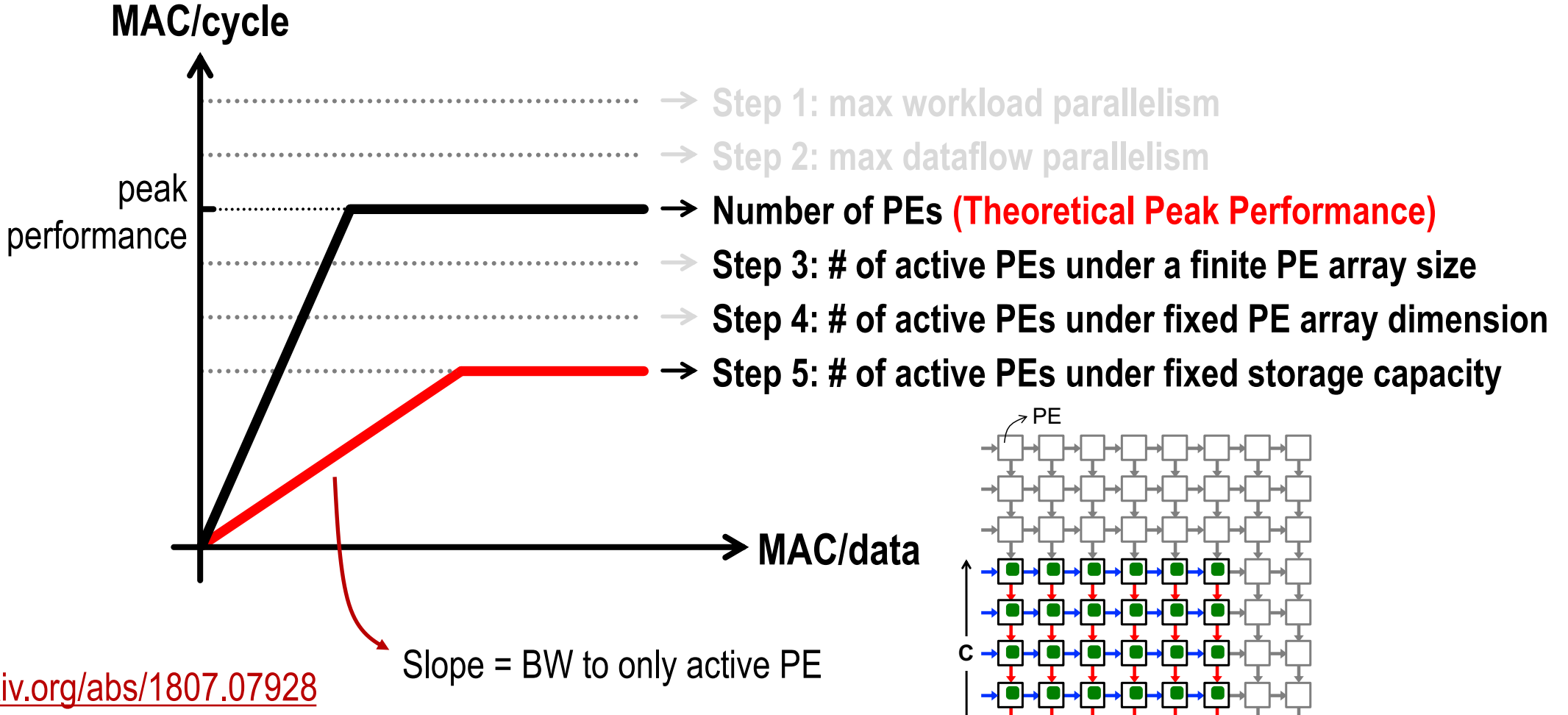
Eyexam: Performance Evaluation Framework



Based on Roofline Model

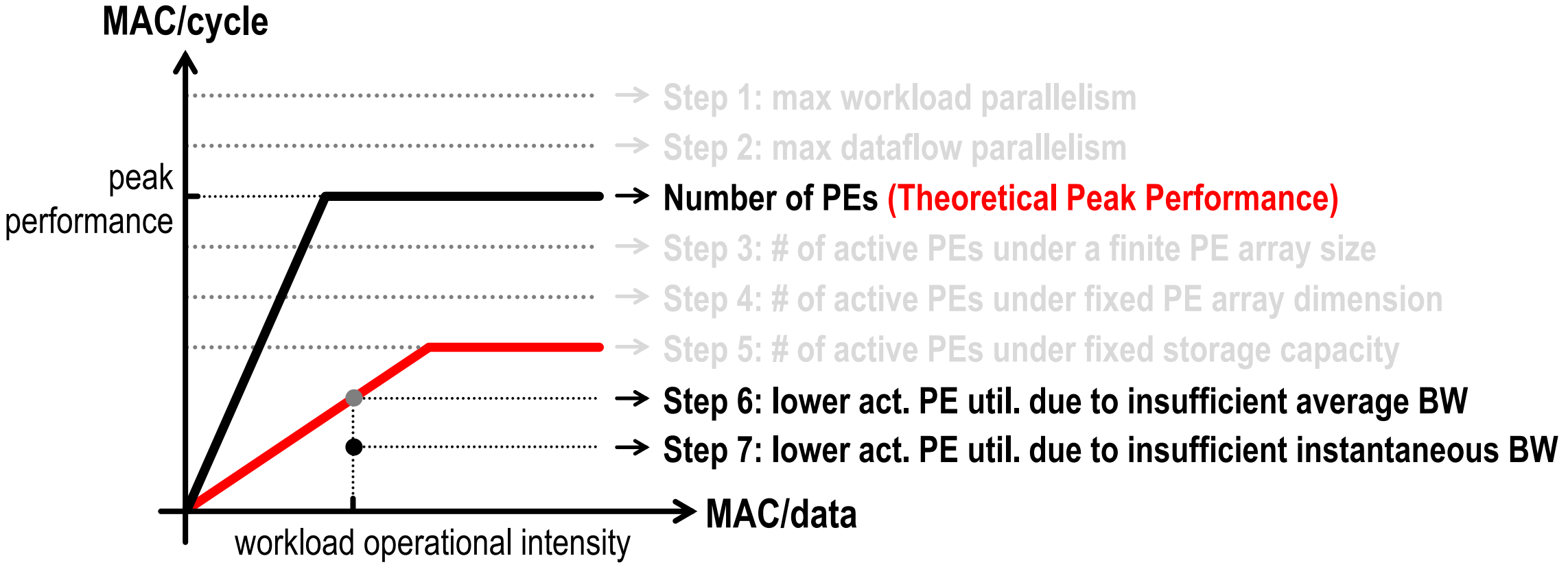
[**Williams**, *CACM* 2009]

Eyexam: Performance Evaluation Framework



https://arxiv.org/abs/1807.07928

Eyexam: Performance Evaluation Framework



https://arxiv.org/abs/1807.07928

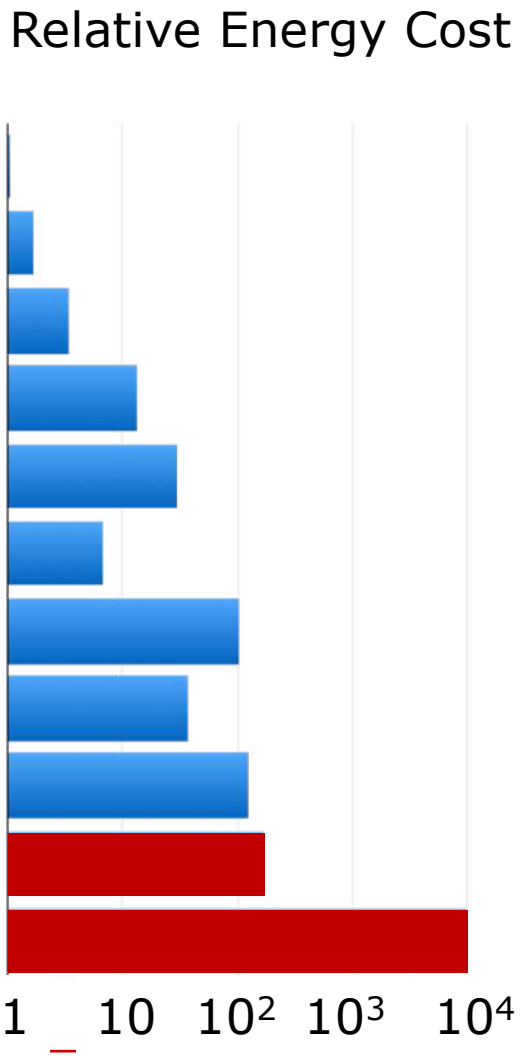
Key Design Objectives of DL Processors

- **Reduce Energy and Power Consumption**
  - Reduce data movement as it dominates energy consumption
    - Exploit data reuse
  - Reduce energy per MAC
    - Reduce switching activity and/or capacitance
    - Reduce instruction overhead
  - Avoid unnecessary MACs

- Power consumption is limited by heat dissipation, which limits the **maximum # of MACs in parallel** (i.e., throughput)

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

[**Horowitz**, *ISSCC* 2014]



Key Design Objectives of DL Processors

- **Flexibility**
  - Reduce overhead of supporting flexibility
  - Maintain efficiency across wide range of DNN workloads
    - Different layer shapes impact the amount of
      - Required storage and compute
      - Available data reuse that can be exploited
    - Different precision across layers & data types (weight, activation, partial sum)
    - Different degrees of sparsity (number of zeros in weights or activations)
    - Types of DNN layers and compute beyond MACs (e.g., activation functions)
- **Scalability**
  - Increase how performance (i.e., throughput, latency, energy, power) scales with increase in amount of resources (e.g., number of PEs, amount of memory, etc.)

Specifications to Evaluate Metrics

- **Accuracy**
  - Difficulty of dataset and/or task should be considered
  - Difficult tasks typically require more complex DNN models
- **Throughput**
  - Number of PEs with utilization (not just peak performance)
  - Runtime for running specific DNN models
- **Latency**
  - Batch size used in evaluation
- **Energy and Power**
  - Power consumption for running specific DNN models
  - Off-chip memory access (e.g., DRAM)
- **Hardware Cost**
  - On-chip storage, # of PEs, chip area + process technology
- **Flexibility**
  - Report performance across a wide range of DNN models
  - Define range of DNN models that are efficiently supported

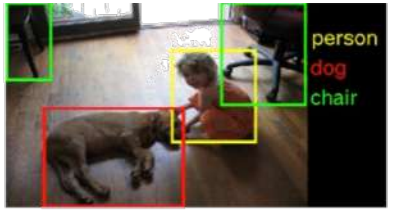
MNIST

ImageNet

Chip

Off-chip memory access

Computer Vision



Speech Recognition

[**Sze**, *CICC* 2017]

Comprehensive Coverage for Evaluation

- All metrics should be reported for fair evaluation of design tradeoffs

- Examples of what can happen if a certain metric is omitted:
  - **Without the accuracy** given for a specific dataset and task, one could run a simple DNN and claim low power, high throughput, and low cost – however, the processor might not be usable for a meaningful task
  - **Without reporting the off-chip memory access**, one could build a processor with *only* MACs and claim low cost, high throughput, high accuracy, and low chip power – however, when evaluating system power, the off-chip memory access would be substantial

- Are results measured or simulated? On what test data?

Example Evaluation Process

The evaluation process for whether a DL processor is a viable solution for a given application might go as follows:

1. **Accuracy** determines if it can perform the given task
2. **Latency and throughput** determine if it can run fast enough and in real-time
3. **Energy and power consumption** will primarily dictate the form factor of the device where the processing can operate
4. **Cost**, which is primarily dictated by the chip area, determines how much one would pay for this solution
5. **Flexibility** determines the range of tasks it can support

Interim Q&A Session

Please ask questions that you feel are essential to follow the rest of this tutorial

Tutorial Overview

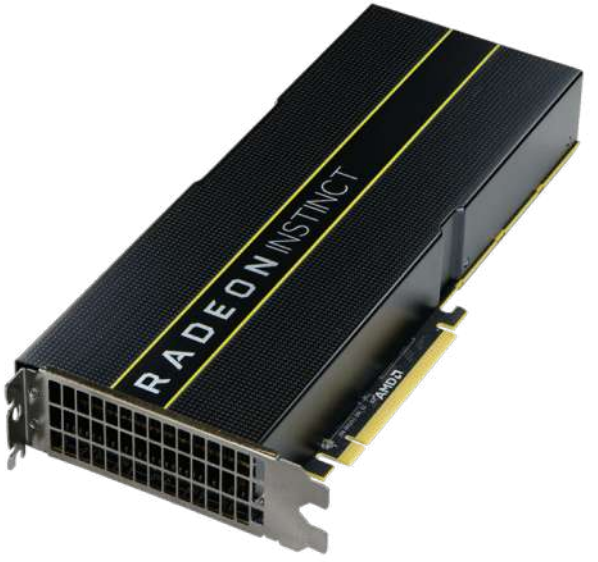
- Deep Learning Overview
- Key Metrics and Design Objectives
- Design Considerations
  - **CPU and GPU Platforms**
  - Specialized / Domain Specific Hardware (ASICs)
    - Efficient Dataflows
    - Algorithm (DNN Model) and Hardware Co-Design
    - Flexibility and Scalability
  - Other Platforms
    - Processing In Memory / In Memory Computing
    - Field Programmable Gate Arrays (FPGAs)
- Tools for Systematic Evaluation of DL Processors
- Should I Use Deep Learning for a Given Task?

CPUs and GPUs Targeting Deep Learning

Intel Xeon (Cascade Lake)

Nvidia Tesla (Volta)

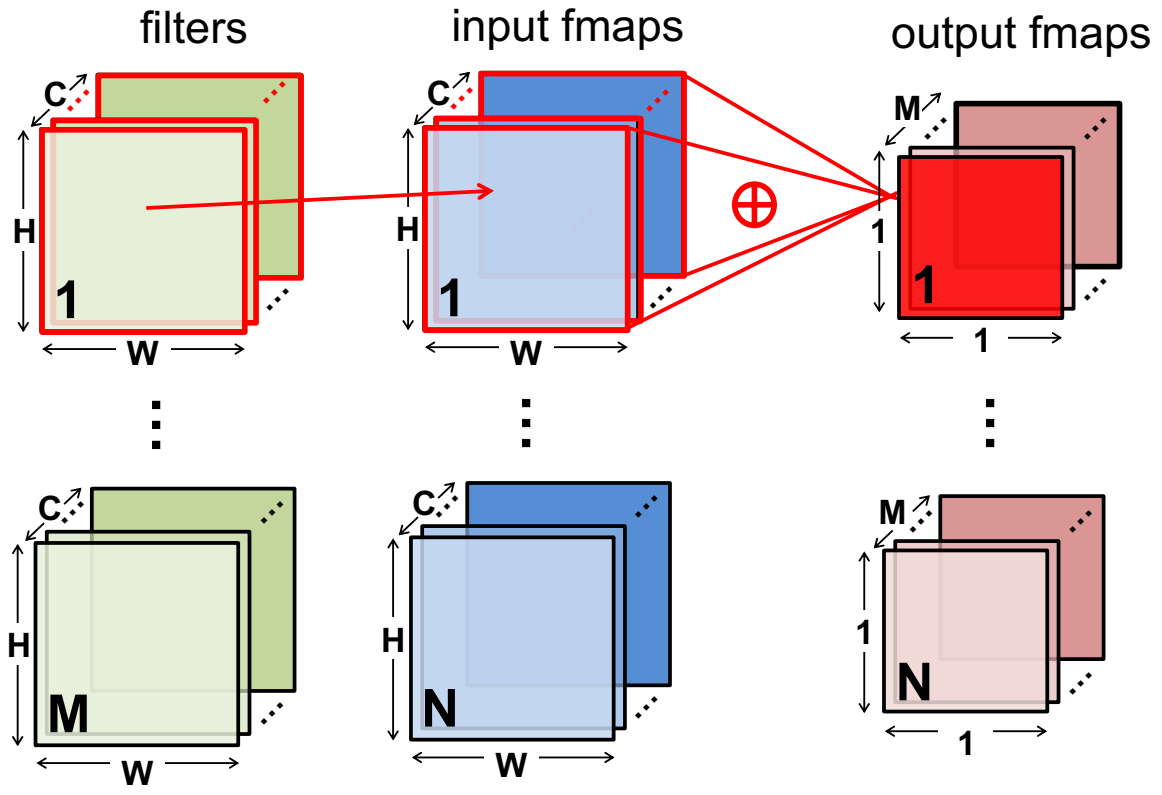
AMD Radeon (Instinct)

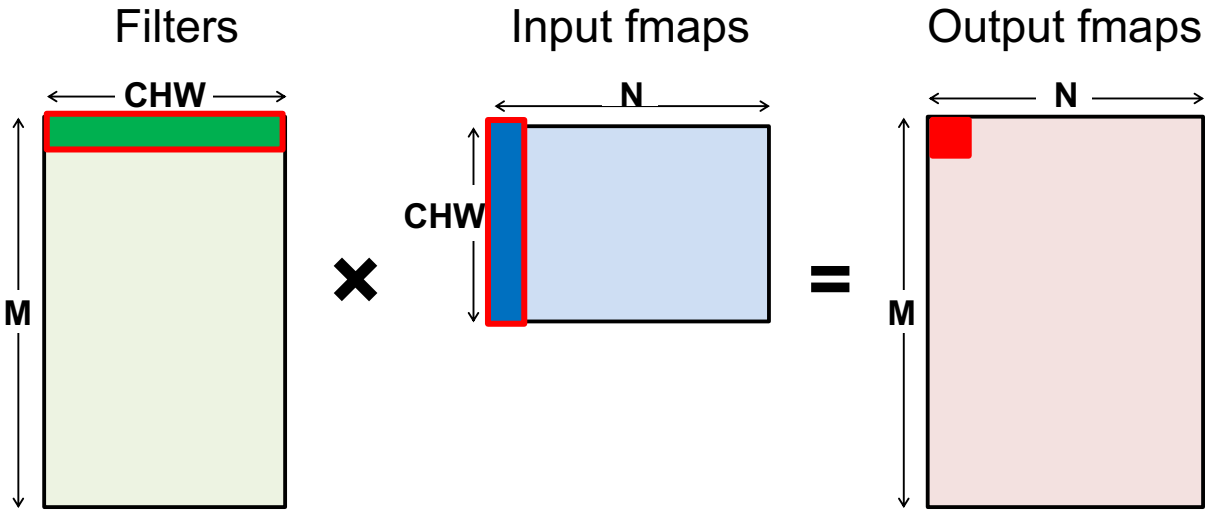


Use **matrix multiplication libraries** on CPUs and GPUs

Map DNN to a Matrix Multiplication

Fully connected layer can be directly represented as matrix multiplication



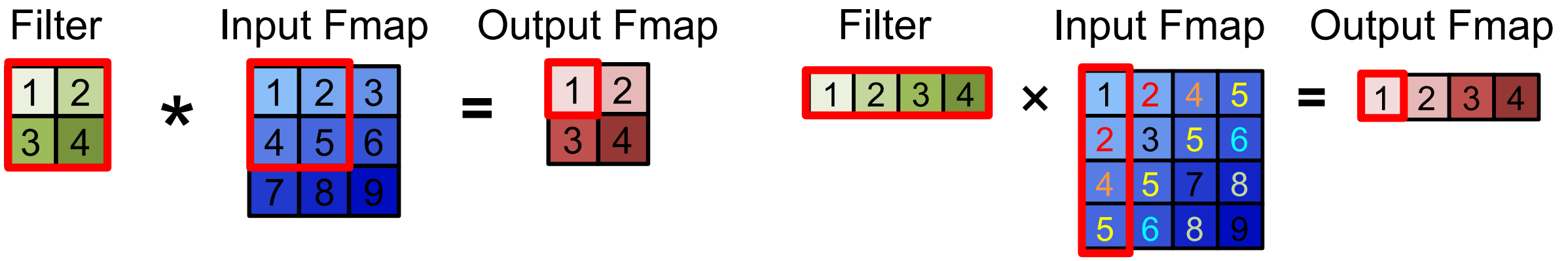


In fully connected layer, filter size (R, S) same as input size (H, W)

Note: Matrix multiplication also heavily used by recurrent and attention layers

Map DNN to a Matrix Multiplication

Convolutional layer can be converted to Toeplitz Matrix



Convolution

Matrix Multiply (by Toeplitz Matrix)

Data is repeated

CPU, GPU Libraries for Matrix Multiplication

- Implementation: Matrix Multiplication (GEMM)
  - CPU: OpenBLAS, Intel MKL, etc
  - GPU: cuBLAS, cuDNN, etc
- Library will note shape of the matrix multiply and select implementation optimized for that shape
- Optimization usually involves proper tiling to memory hierarchy

Analogy: Gauss's Multiplication Algorithm

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

4 multiplications + 3 additions

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

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

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

Real part $= k_1 - k_3$

Imaginary part $= k_1 + k_2$.

3 multiplications + 5 additions

Reduce number of multiplications to **increase** throughput

Reduce Operations in Matrix Multiplication

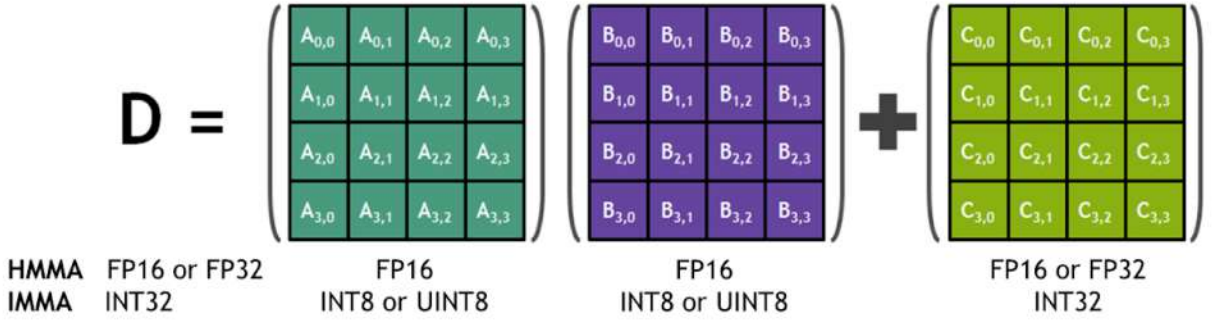
- **Fast Fourier Transform** [**Mathieu**, *ICLR* 2014]
  - Pro: Direct convolution $O(N_o^2 N_f^2)$ to $O(N_o^2 \log_2 N_o)$
  - Con: Increase storage requirements
- **Strassen** [**Cong**, *ICANN* 2014]
  - Pro: $O(N^3)$ to $(N^{2.807})$
  - Con: Numerical stability
- **Winograd** [**Lavin**, *CVPR*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

Compiler selects transform based on filter size

Reduce Instruction Overhead

- **Perform more operations per instruction**
  - **CPU:** SIMD / Vector Instructions
    - e.g., Specialized Vector Neural Network Instructions (VNNI) fuses separate multiply and add instructions into single MAC instruction and avoids storing intermediate values in memory
  - **GPU:** SIMT / Tensor Instructions
    - e.g., New opcode Matrix Multiply Accumulate (HMMA) performs 64 MACs with Tensor Core
- **Perform more MACs per cycle without increasing memory bandwidth by adding support for reduced precision**
  - e.g., If access 512 bits per cycle, can perform **64** 8-bit MACs vs. **16** 32-bit MACs

Tensor Core
Image Source: Nvidia



Design Considerations for CPU and GPU

- **Software (compiler)**
  - **Reduce unnecessary MACs**: Apply transforms
  - **Increase PE utilization**: Schedule loop order and tile data to increase data reuse in memory hierarchy
- **Hardware**
  - **Reduce time per MAC**
    - Increase speed of PEs
    - Increase MACs per instructions using large aggregate instructions (e.g., SIMD, tensor core) → requires additional hardware
  - **Increase number of parallel MACs**
    - Increase number of PEs on chip → area cost
    - Support reduced precision in PEs
  - **Increase PE utilization**
    - Increase on-chip storage → area cost
    - External memory BW → system cost

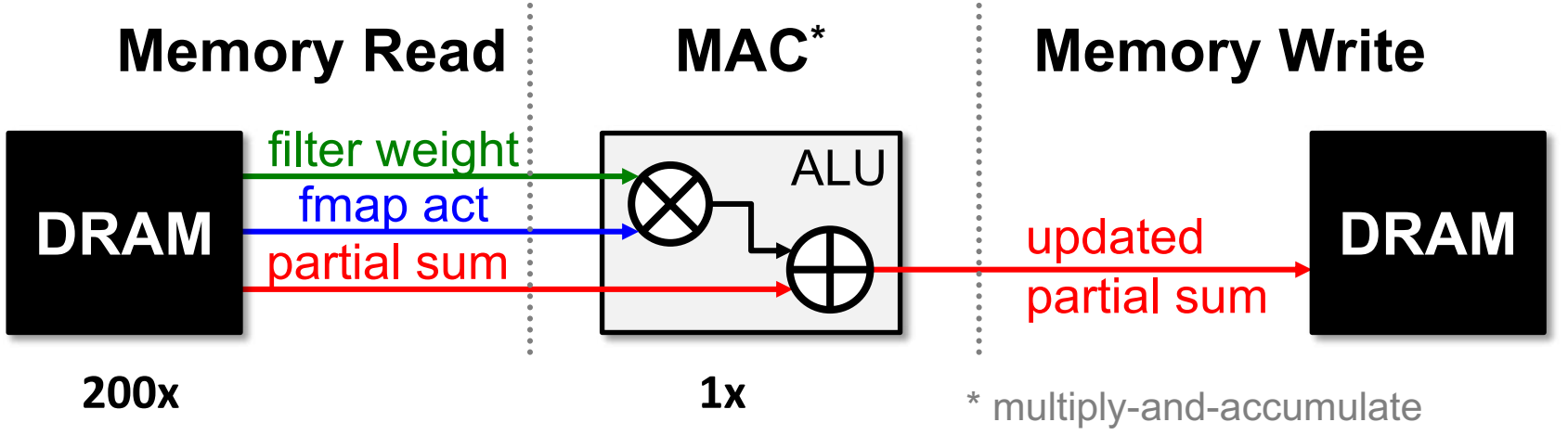
Tutorial Overview

- Deep Learning Overview
- Key Metrics and Design Objectives
- Design Considerations
  - CPU and GPU Platforms
  - **Specialized / Domain Specific Hardware (ASICs)**
    - Efficient Dataflows
    - Algorithm (DNN Model) and Hardware Co-Design
    - Flexibility and Scalability
  - Other Platforms
    - Processing In Memory / In Memory Computing
    - Field Programmable Gate Arrays (FPGAs)
- Tools for Systematic Evaluation of DL Processors
- Should I Use Deep Learning for a Given Task?

Properties We Can Leverage

- Operations exhibit **high parallelism**
  → **high throughput** possible

- Memory Access is the Bottleneck



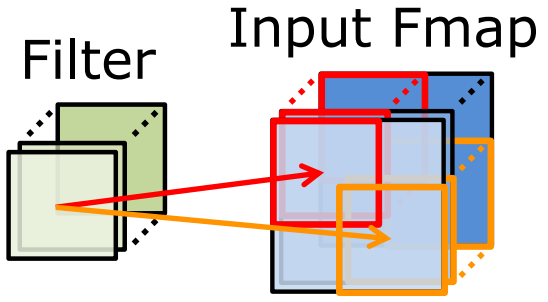
Worst Case: all memory R/W are **DRAM** accesses

Example: AlexNet has **724M** MACs → **2896M** DRAM accesses required

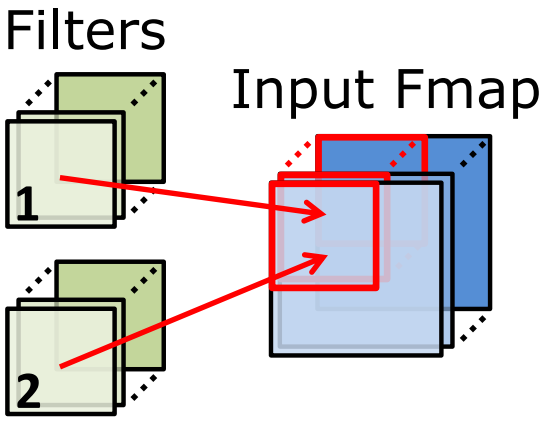
Properties We Can Leverage

- Operations exhibit **high parallelism**
  → **high throughput** possible

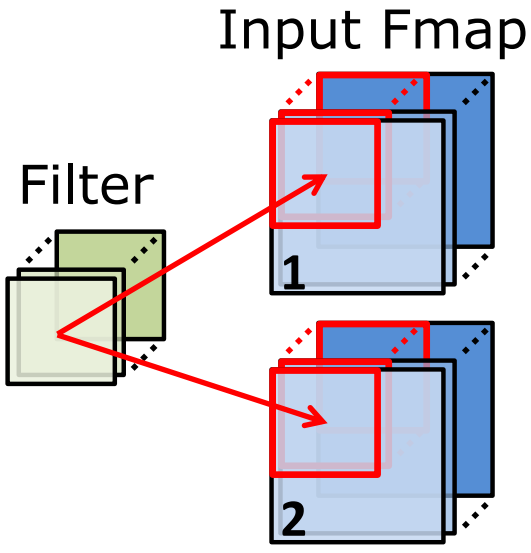
- **Input data reuse** opportunities (e.g., up to 500x for AlexNet)
  → exploit **low-cost memory**



Convolutional Reuse
(Activations, Weights)
CONV layers only
(sliding window)



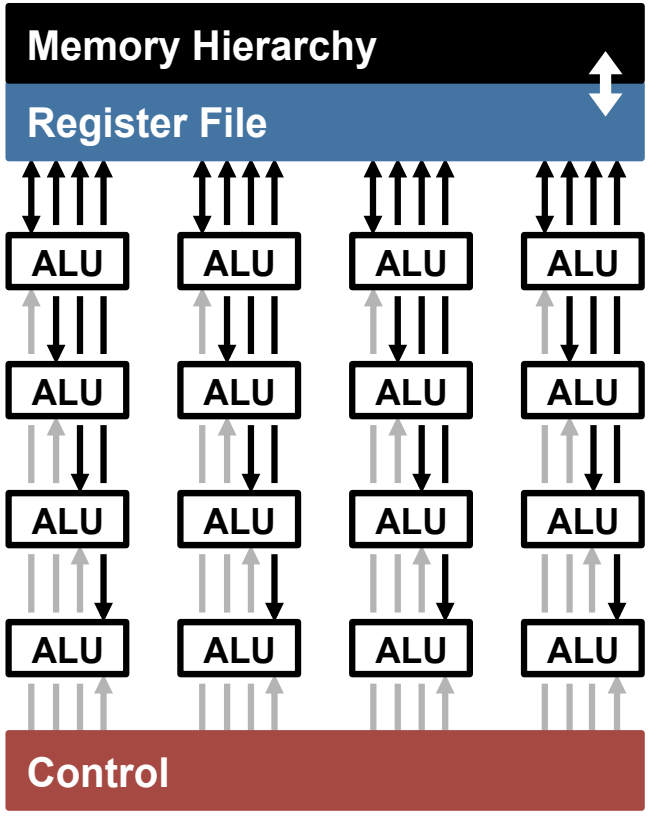
Fmap Reuse
(Activations)
CONV and FC layers



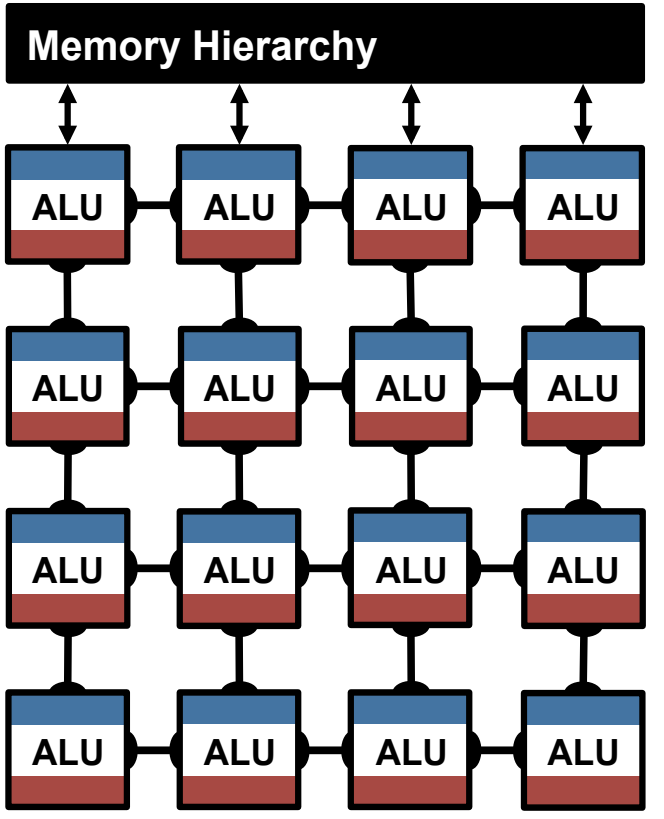
Filter Reuse
(Weights)
CONV and FC layers
(batch size > 1)

Highly-Parallel Compute Paradigms

Temporal Architecture (SIMD/SIMT)



Spatial Architecture (Dataflow Processing)

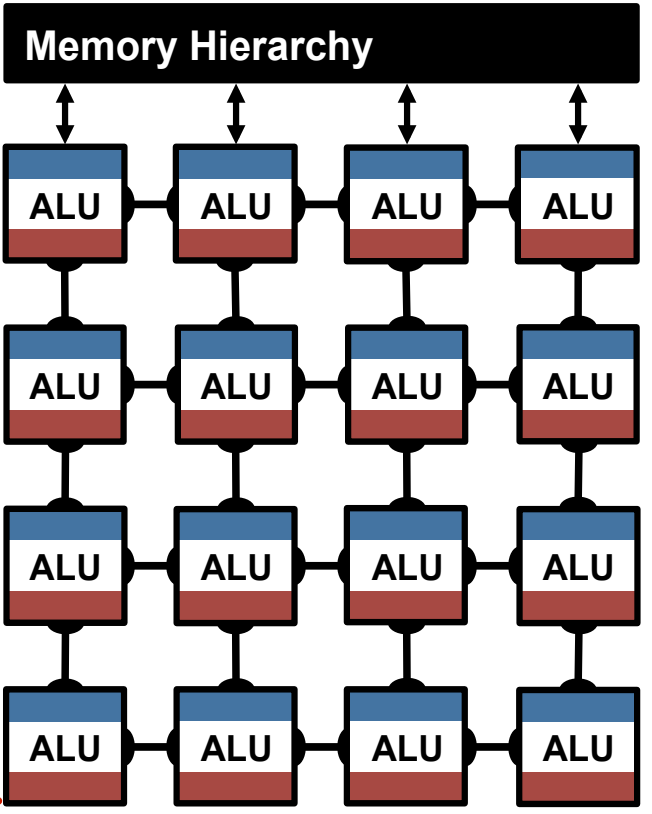


Advantages of Spatial Architecture

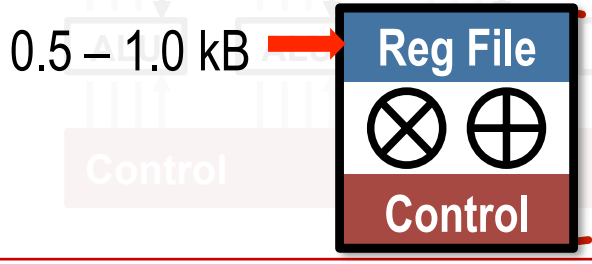
Efficient Data Reuse
Distributed local storage (RF)

Inter-PE Communication
Sharing among regions of PEs

Spatial Architecture (Dataflow Processing)

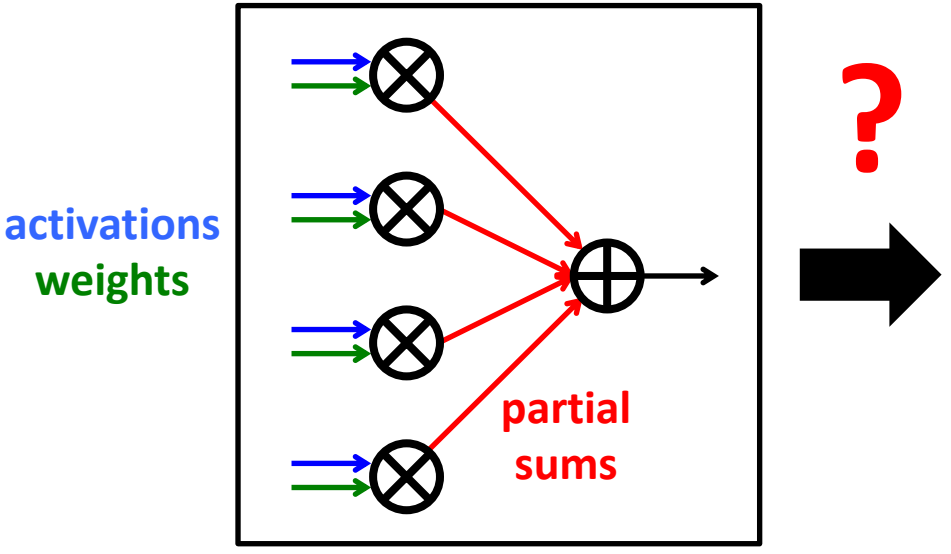


Processing Element (PE)



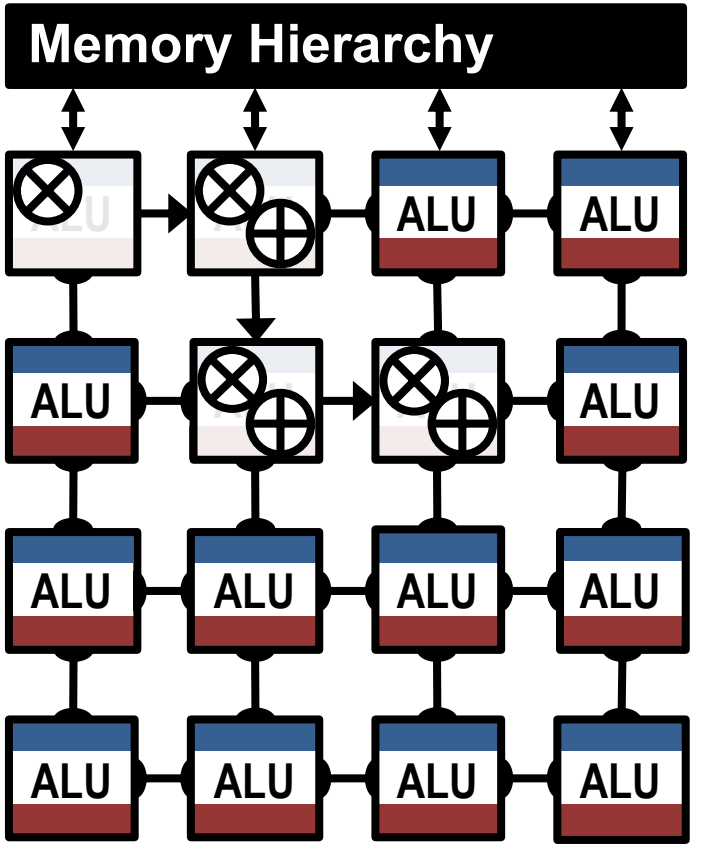
How to Map the Dataflow?

CNN Convolution



?

Spatial Architecture (Dataflow Processing)



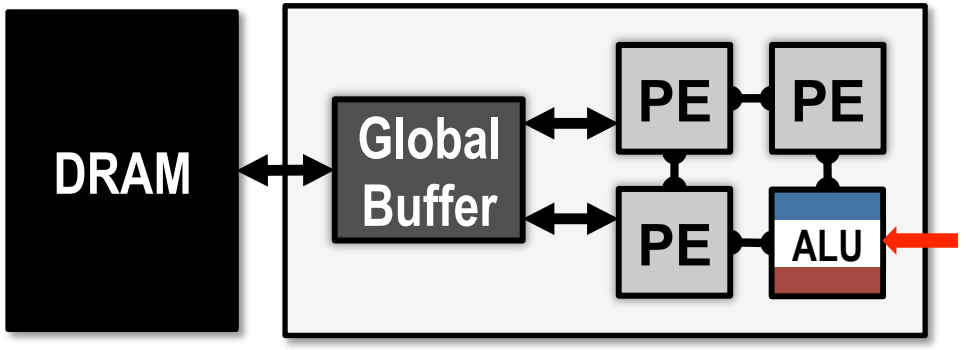
Goal: Increase reuse of input data (**weights** and **activations**) and local **partial sums** accumulation

Tutorial Overview

- Deep Learning Overview
- Key Metrics and Design Objectives
- Design Considerations
  - CPU and GPU Platforms
  - Specialized / Domain Specific Hardware (ASICs)
    - **Efficient Dataflows**
    - Algorithm (DNN Model) and Hardware Co-Design
    - Flexibility and Scalability
  - Other Platforms
    - Processing In Memory / In Memory Computing
    - Field Programmable Gate Arrays (FPGAs)
- Tools for Systematic Evaluation of DL Processors
- Should I Use Deep Learning for a Given Task?

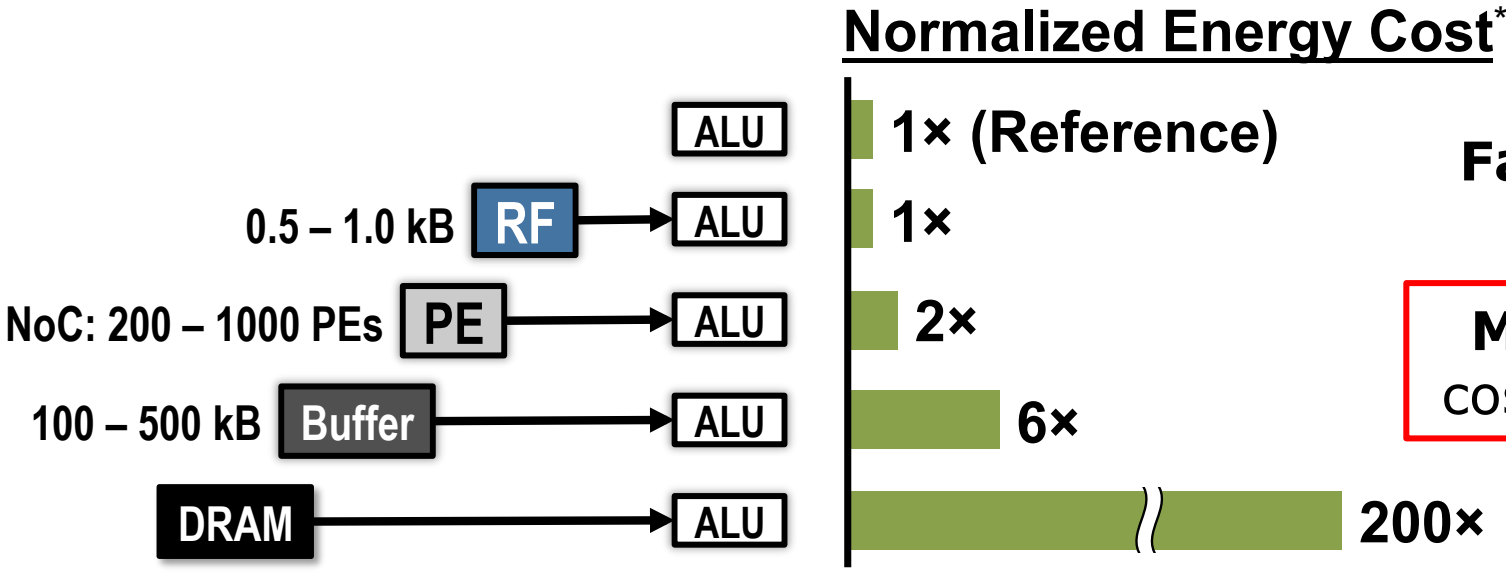
Y.-H. Chen, J. Emer, V. Sze, "**Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks**," *International Symposium on Computer Architecture (ISCA)*, June 2016.

Data Movement is Expensive



Specialized hardware with small (< 1kB) low cost memory near compute

fetch data to run a MAC here



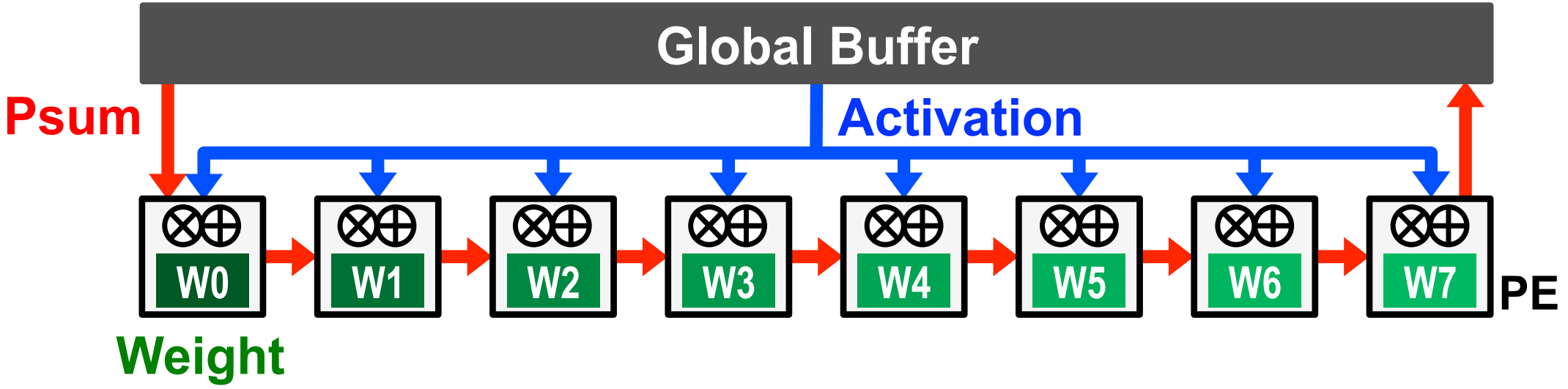
Farther and **larger** memories consume more power

Maximize data reuse at low cost levels of memory hierarchy

* measured from a commercial 65nm process

Weight Stationary (WS)

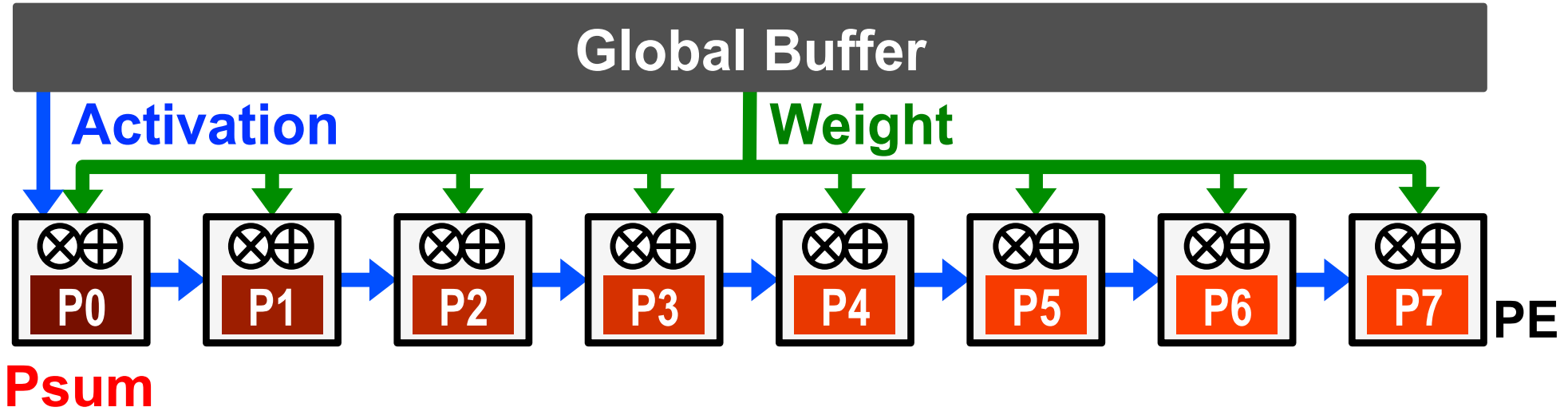
[**Chen**, *ISCA* 2016]



- **Minimize weight** read energy consumption
  - maximize convolutional and filter reuse of weights
- **Broadcast activations** and **accumulate partial sums spatially** across the PE array
- Examples: [**TPU**, *ISCA* 2017], **NVDLA**

Output Stationary (OS)

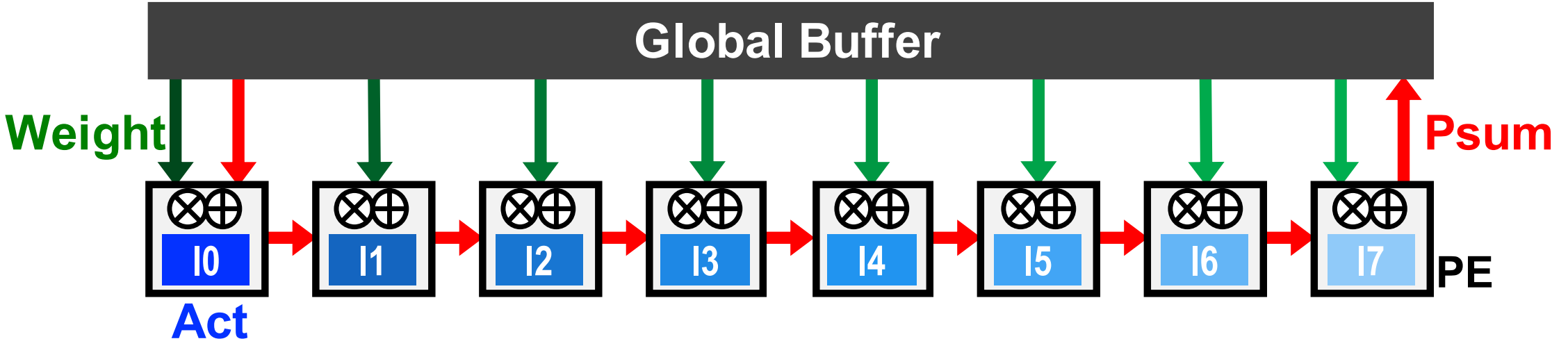
[**Chen**, *ISCA* 2016]



- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Broadcast/Multicast filter weights** and **reuse activations spatially** across the PE array
- Examples: [**Moons**, *VLSI* 2016], [**Thinker**, *VLSI* 2017]

Input Stationary (IS)

[**Chen**, *ISCA* 2016]

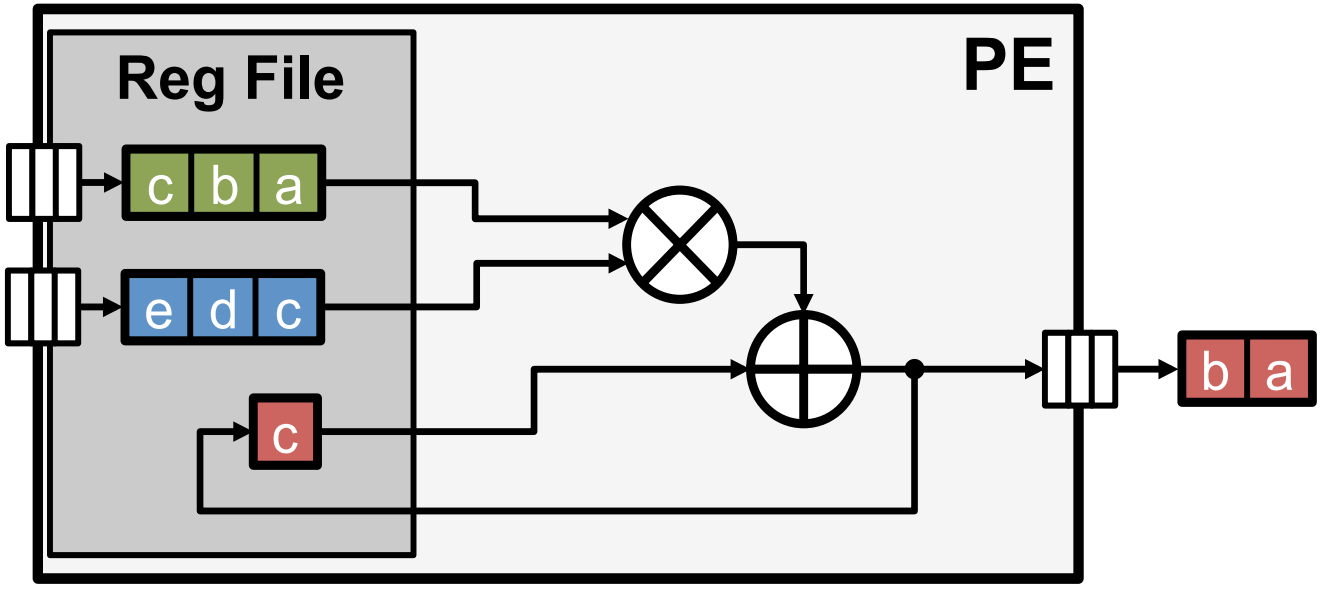


- **Minimize activation** read energy consumption
  - maximize convolutional and fmap reuse of activations
- **Unicast weights** and **accumulate partial sums spatially** across the PE array
- Example: [**SCNN**, ISCA 2017]

Row Stationary Dataflow

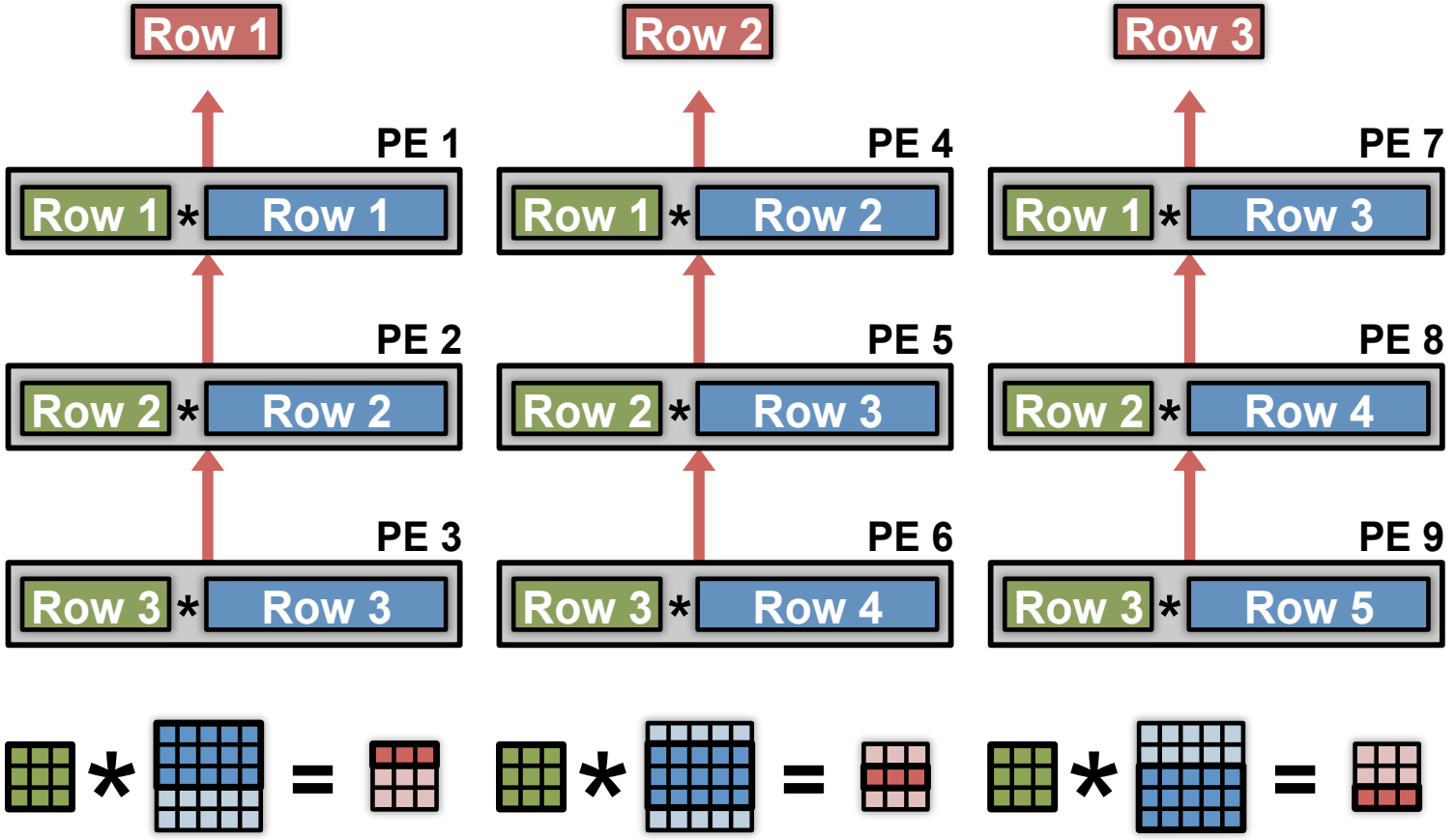
- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **fmap** sliding window in RF
- Maximize row **psum accumulation** in RF

1D convolution within PE



[**Chen**, *ISCA* 2016]

Row Stationary Dataflow



Optimize for **overall energy efficiency** instead for only a certain data type

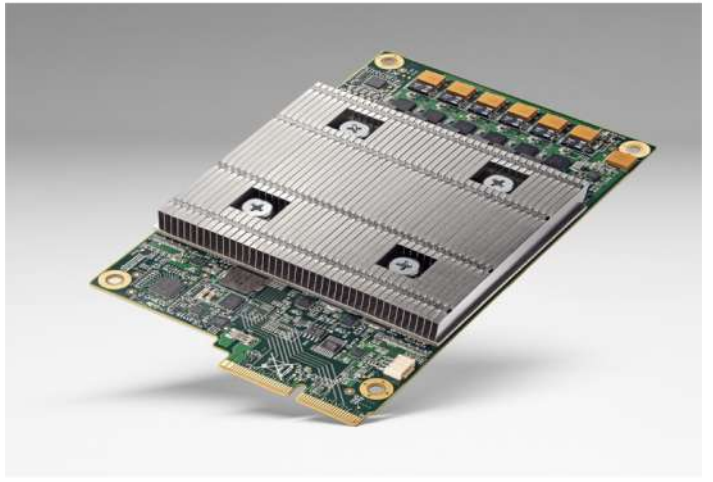
[**Chen**, *ISCA* 2016]

Tutorial Overview

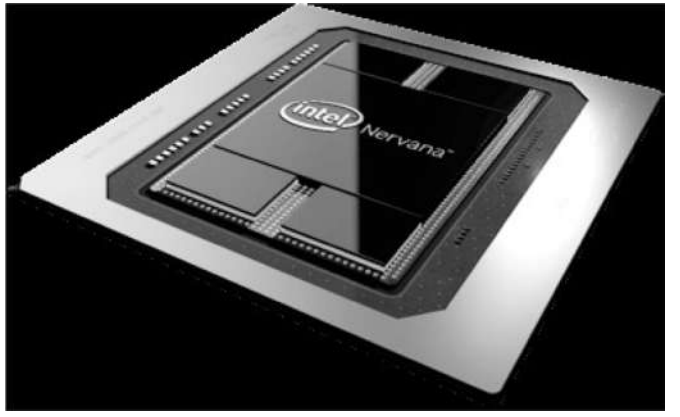
- Deep Learning Overview
- Key Metrics and Design Objectives
- Design Considerations
  - CPU and GPU Platforms
  - Specialized / Domain Specific Hardware (ASICs)
    - Efficient Dataflows
    - **Algorithm (DNN Model) and Hardware Co-Design**
    - Flexibility and Scalability
  - Other Platforms
    - Processing In Memory / In Memory Computing
    - Field Programmable Gate Arrays (FPGAs)
- Tools for Systematic Evaluation of DL Processors
- Should I Use Deep Learning for a Given Task?

Commercial Products Support Reduced Precision

Nvidia's Pascal (2016)



Google's TPU (2016)
TPU v2 & v3 (2019)



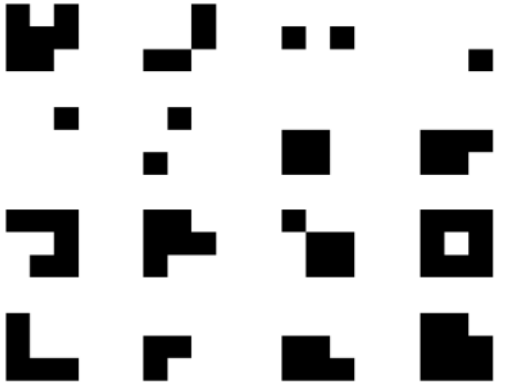
Intel's NNP-L (2019)

8-bit fixed for Inference & **16-bit float** for Training

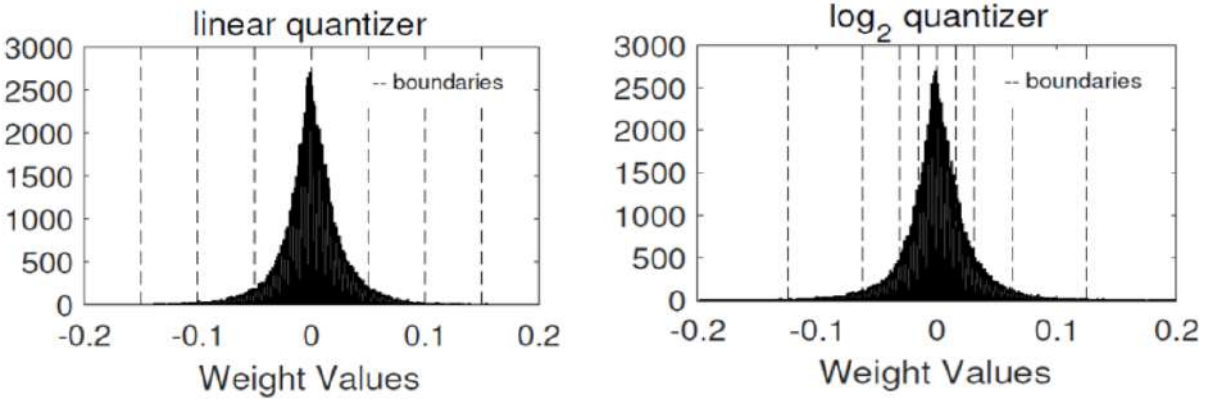
Reduced Precision in Research

- Reduce number of bits
  - Binary Nets [**Courbariaux**, *NeurIPS* 2015]
- Reduce number of unique weights
  - Ternary Weight Nets [**Li**, *NeurIPS Workshop* 2016]
  - XNOR-Net [**Rategari**, *ECCV* 2016]
- Non-Linear Quantization
  - LogNet [**Lee**, *ICASSP* 2017]
- Training
  - 8-bit with stochastic rounding [**Wang**, *NeurIPS* 2018]

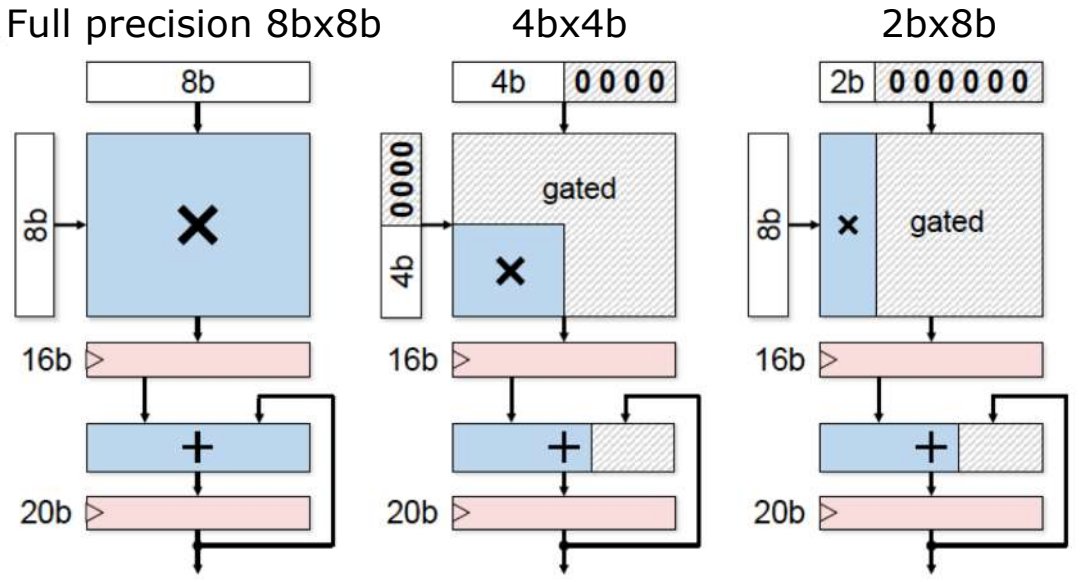
Binary Filters



Log Domain Quantization

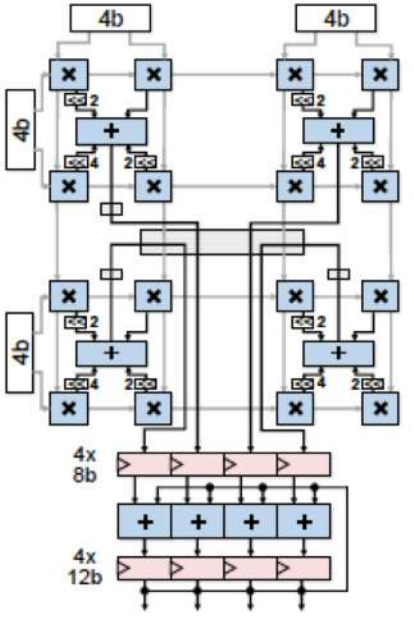


Precision Scalable MACs for Varying Precision

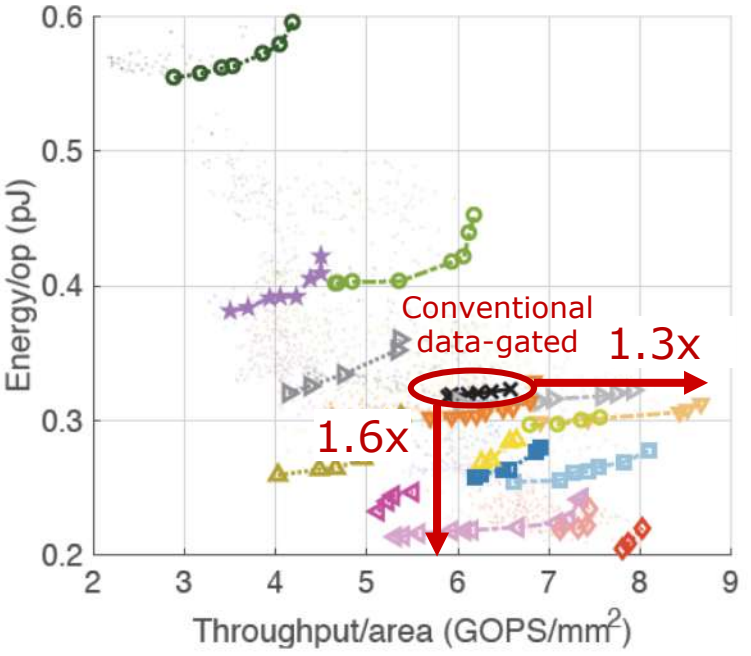


Conventional data-gated MAC
Gate unused logic (e.g., full adders) to reduce energy consumption

[**Camus**, *JETCAS* 2019]



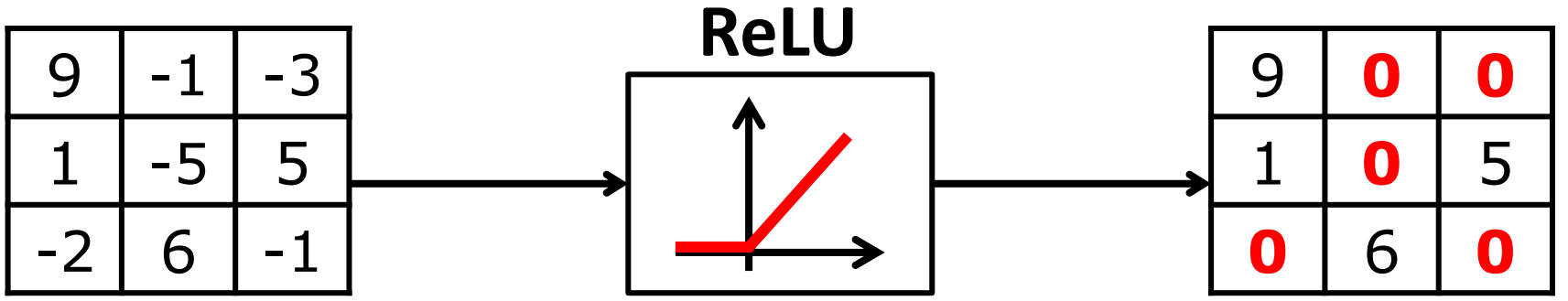
Evaluation of **19 precision scalable MAC designs**
5% of values 8bx8b
47.5% of values at 2bx2b and 4bx4b

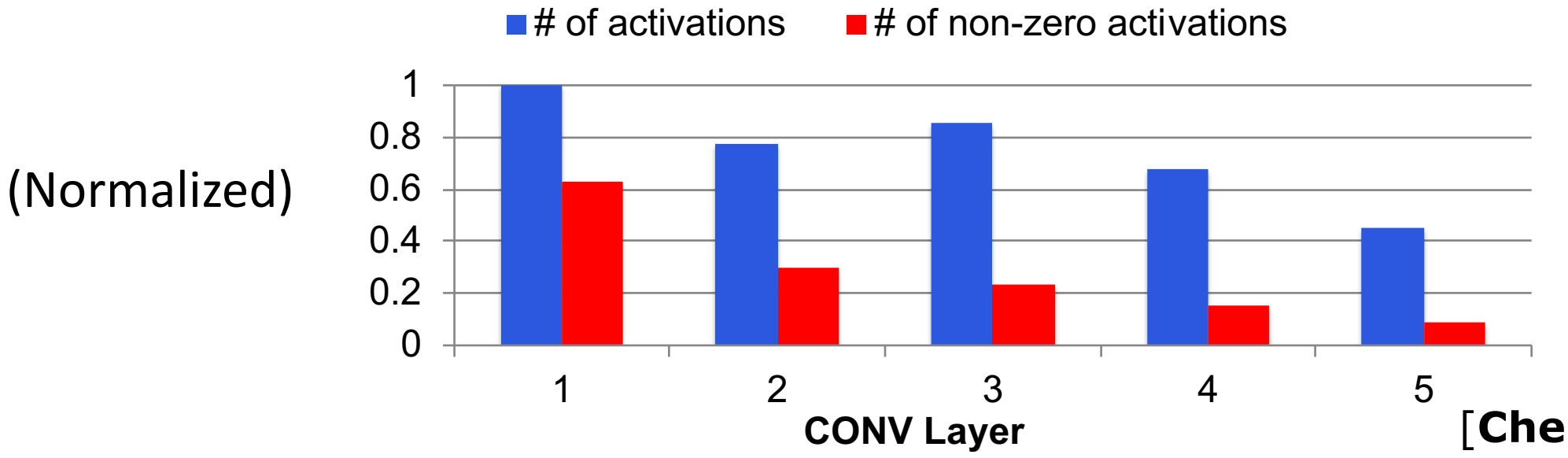


Many approaches **increase utilization of logic to increase throughput/area**; however, area and energy overhead can reduce benefits

Sparsity in Activation Data

Many zeros in output fmaps after ReLU

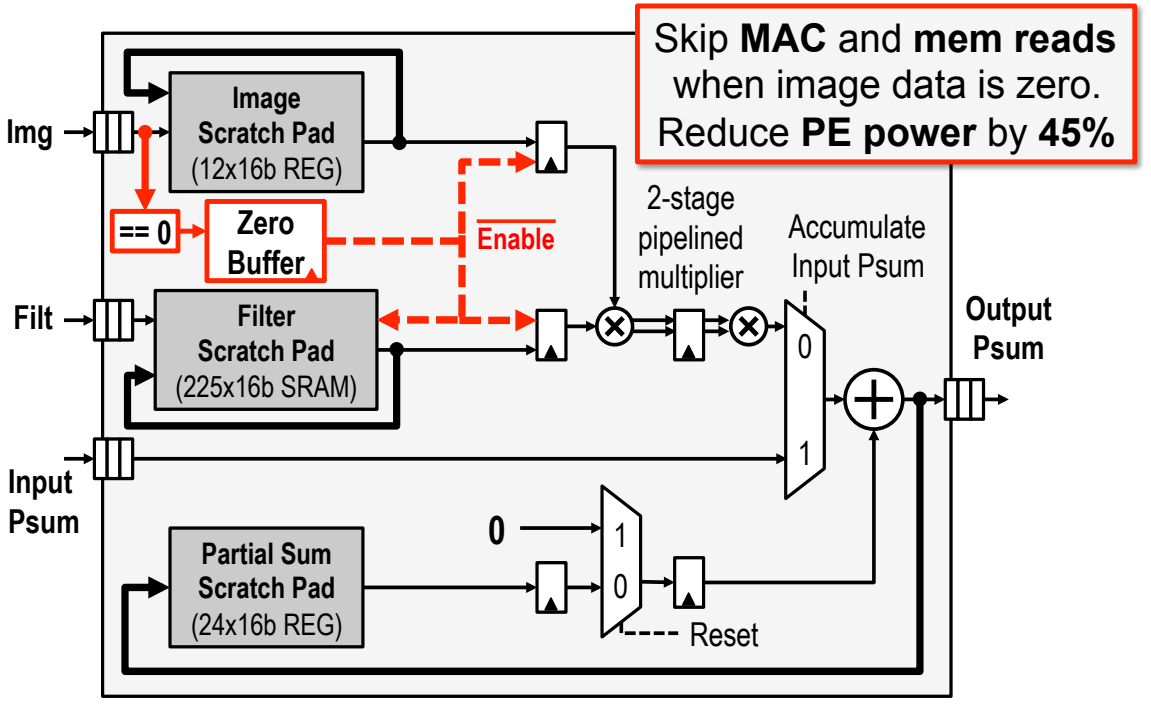




[**Chen**, *ISSCC* 2016]

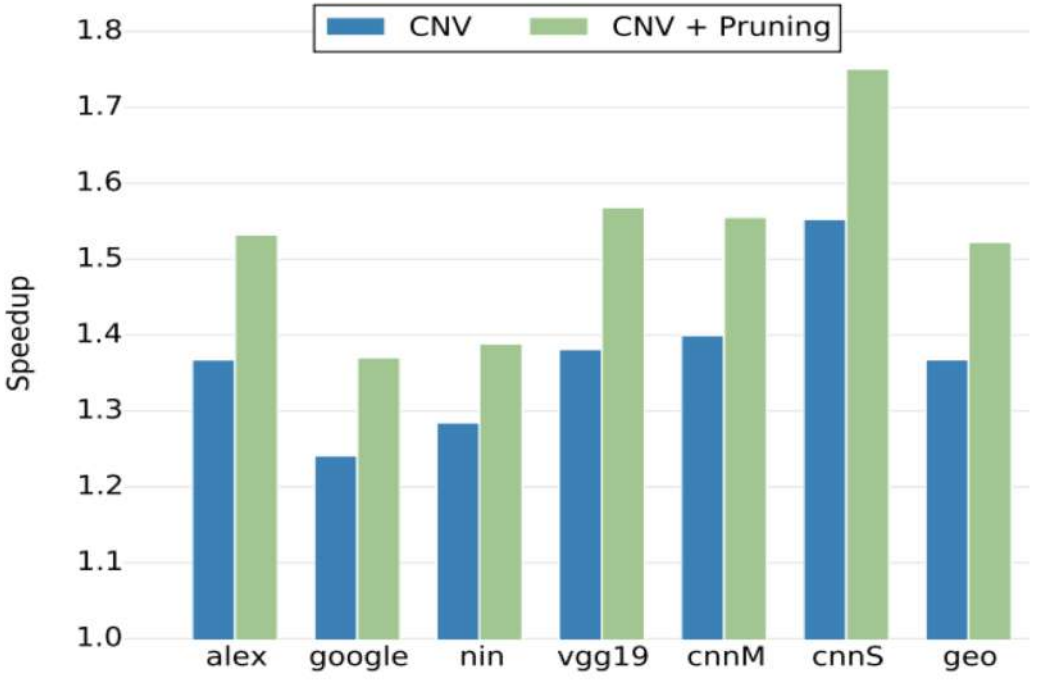
Data Gating / Zero Skipping

Gate operations
(reduce power consumption)



[**Chen**, Eyeriss, *ISSCC* 2016]

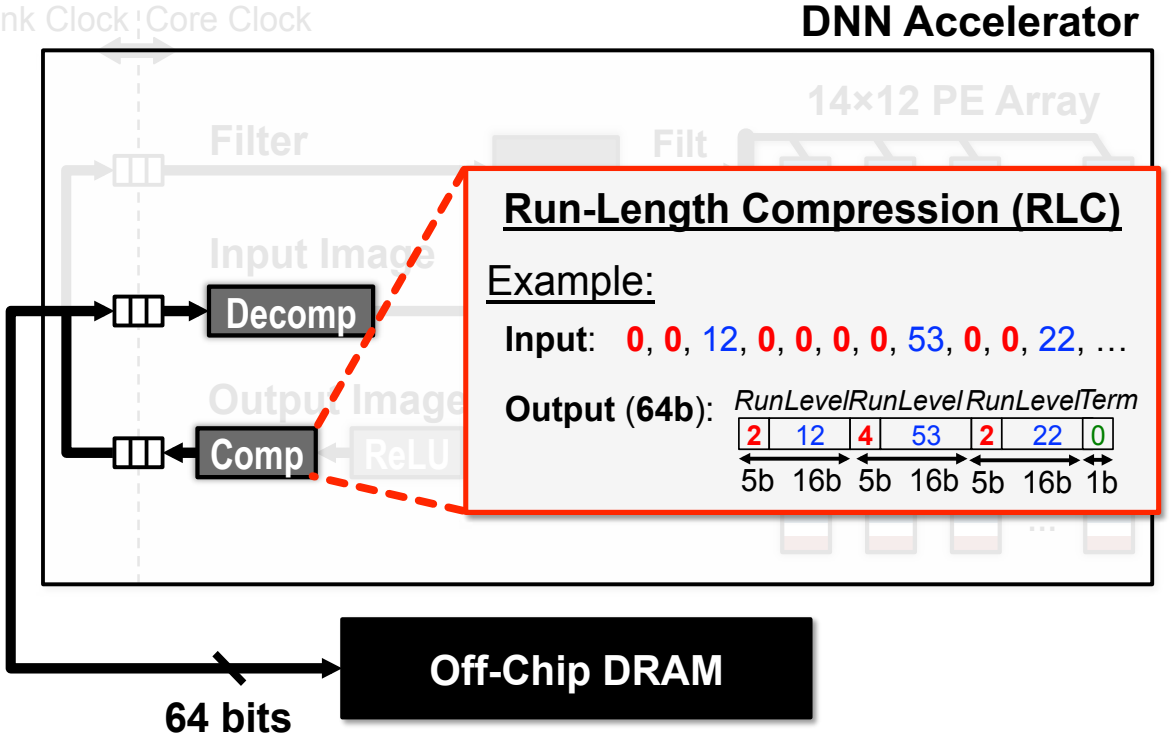
Skip operations
(increase throughput)

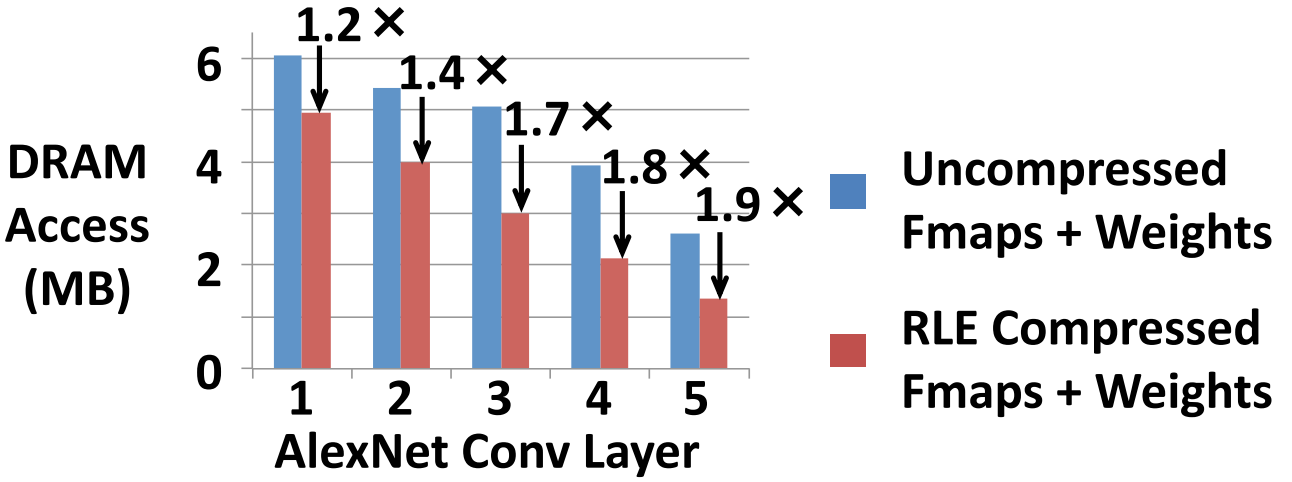


[**Albericio**, Cnvlutin, *ISCA* 2016]

Apply Compression to Reduce Data Movement

Example: Eyeriss compresses activations to reduce DRAM BW



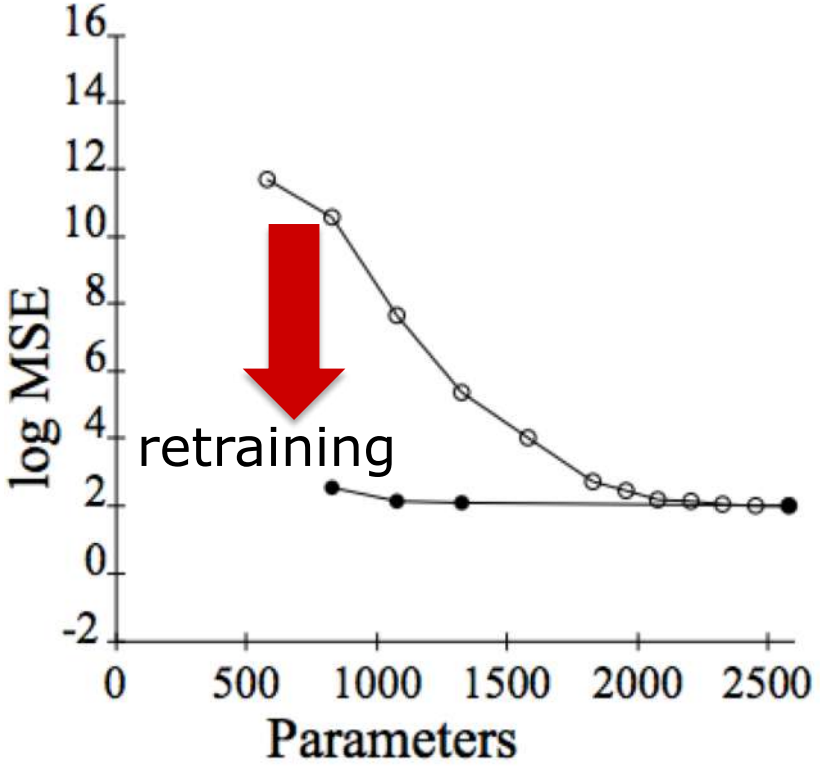


Simple RLC within 5% - 10% of theoretical entropy limit

[**Chen**, *ISSCC* 2016]

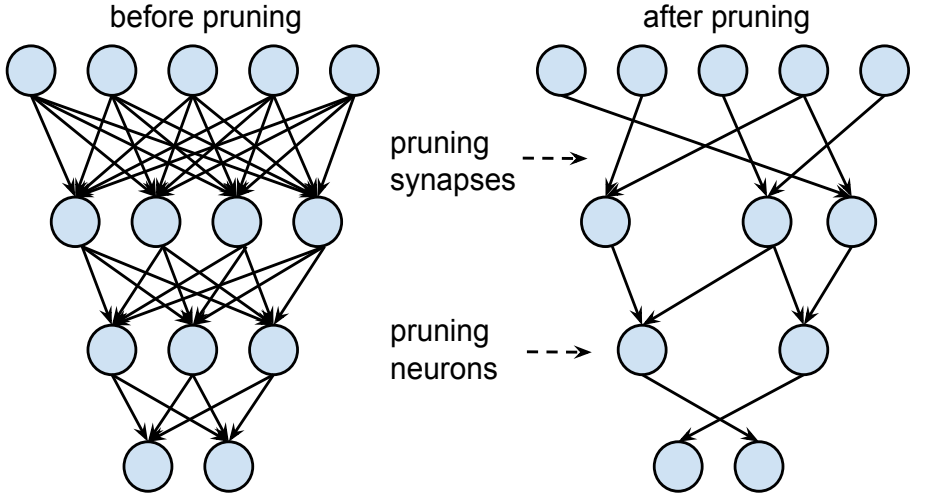
Pruning – Make Weights Sparse

Optimal Brain Damage



[**Lecun**, *NeurIPS* 1990]

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



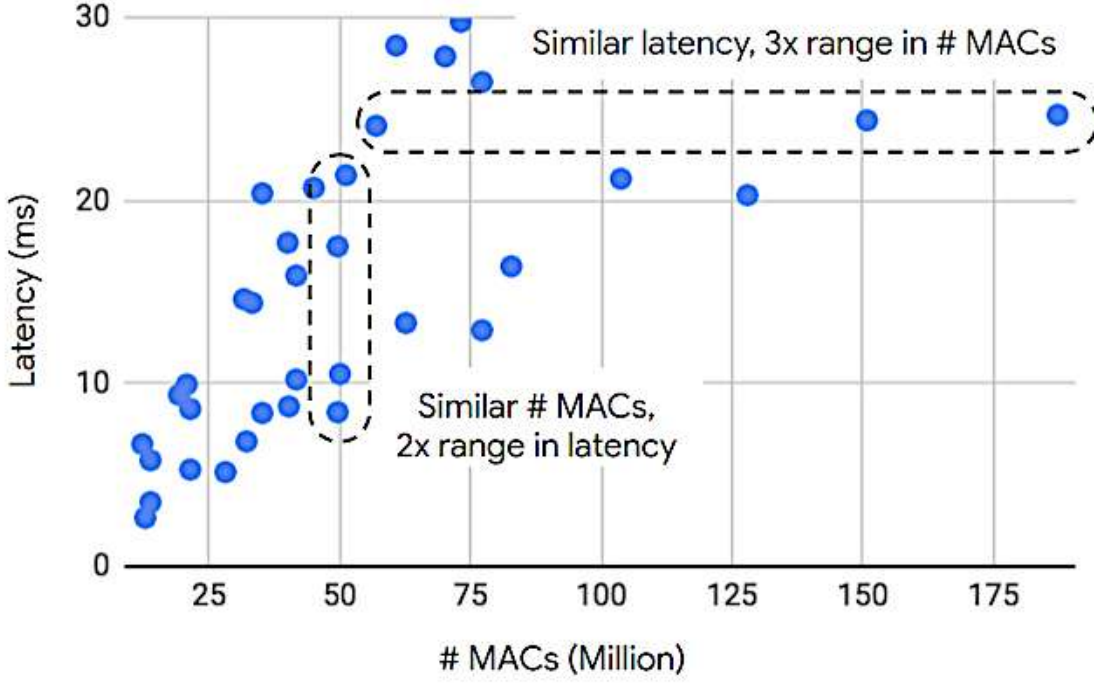
Example: *AlexNet*
Weight Reduction: *CONV layers 2.7x,* ***FC layers 9.9x***
Overall Reduction: *Weights 9x, MACs 3x*

[**Han**, *NeurIPS* 2015]

How to Evaluate Complexity of DNN Model?

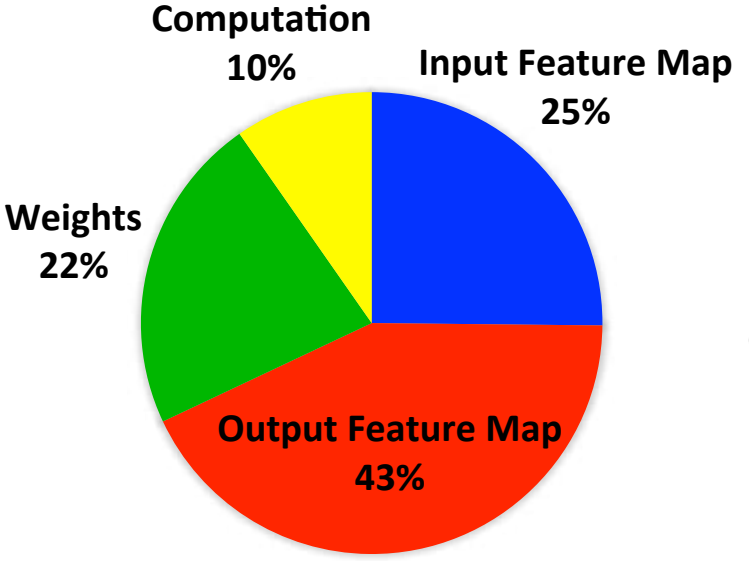
Number of MACs and weights are not good proxies for latency and energy

# of operations (MACs) does not approximate latency well



Source: Google (https://ai.googleblog.com/2018/04/introducing-cvpr-2018-on-device-visual.html)

# of weights ***alone*** is not a good metric for energy (**All data types** should be considered)



Energy breakdown of GoogLeNet

https://energyestimation.mit.edu/

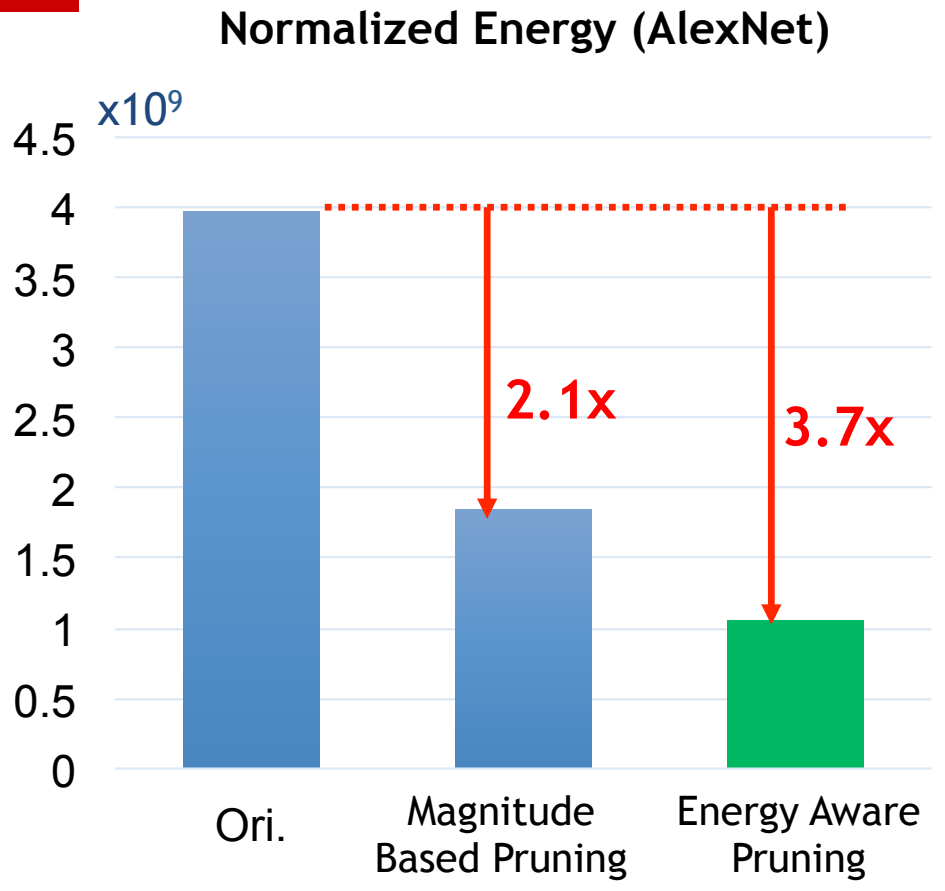
[**Yang**, *CVPR* 2017]

Energy-Aware Pruning

Directly target energy and incorporate it into the optimization of DNNs to provide greater energy savings

- Sort layers based on energy and prune layers that consume the most energy first
- Energy-aware pruning reduces AlexNet energy by **3.7x** and outperforms the previous work that uses magnitude-based pruning by **1.7x**

[**Yang**, *CVPR* 2017]



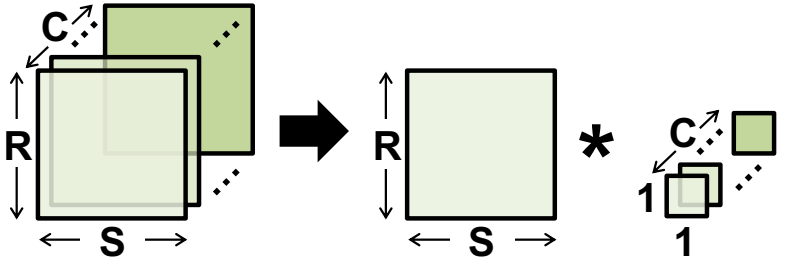
Pruned models available at
http://eyeriss.mit.edu/energy.html

Compact DNN Models

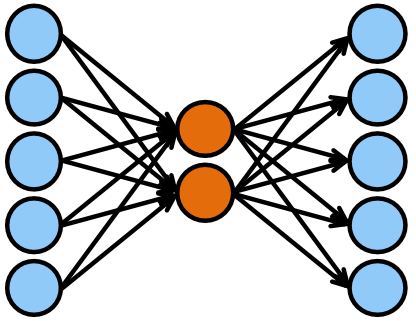
- Design compact DNN Models
  - Tends to increase range of layer shapes (e.g., R, S, C) that need to be supported
  - Can be handcrafted or learned using Network/Neural Architecture Search (NAS)

Filter Decomposition

Decompose large filters into smaller filters



1x1 convolutions



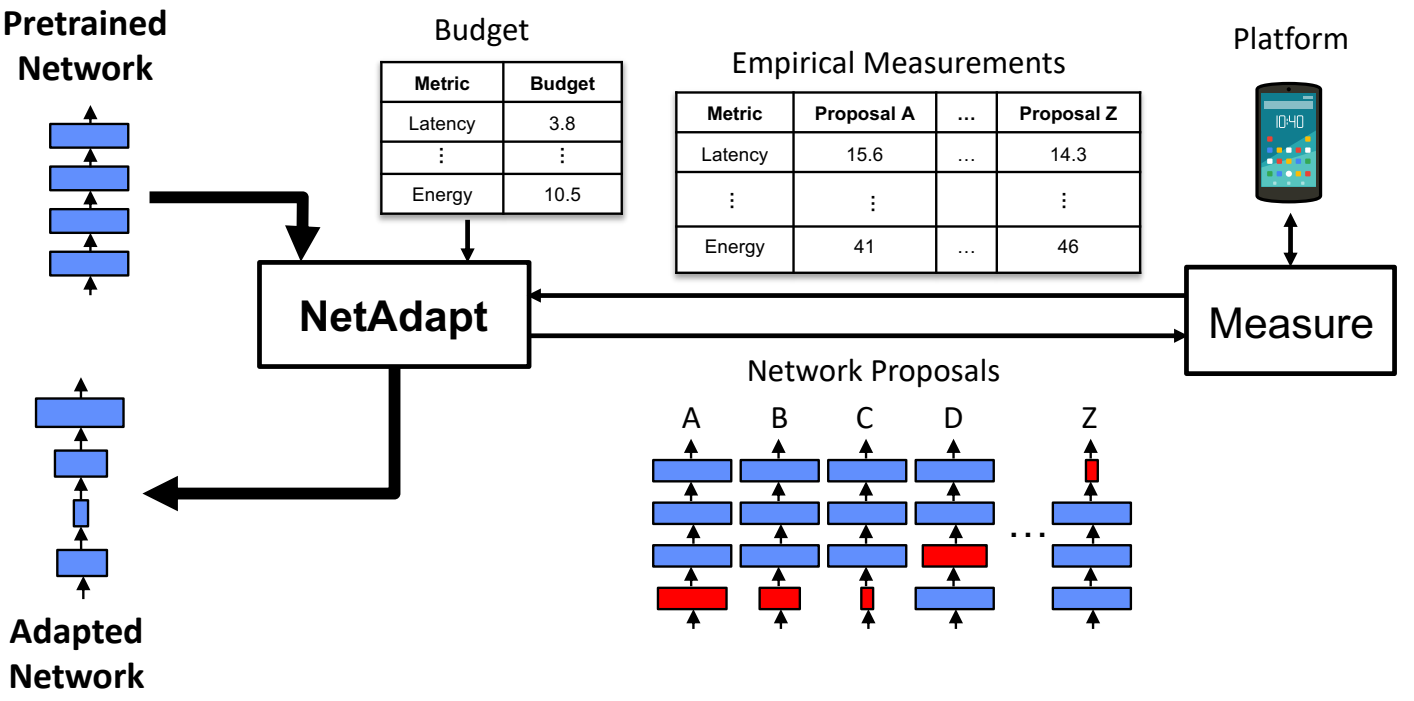
Reduce number of channels before large filter convolution

	Year	Accuracy*	# Layers	# Weights	# MACs
AlexNet	2012	80.4%	8	**61M**	**724M**
MobileNet	2017	**89.5%**	28	**4M**	**569M**

* ImageNet Classification Top-5

NetAdapt: Platform-Aware DNN Adaptation

- **Automatically adapt DNN** to a mobile platform to reach a target latency or energy budget
- Use **empirical measurements** to guide optimization (avoid modeling of tool chain or platform architecture)
- Requires **very few hyperparameters** to tune



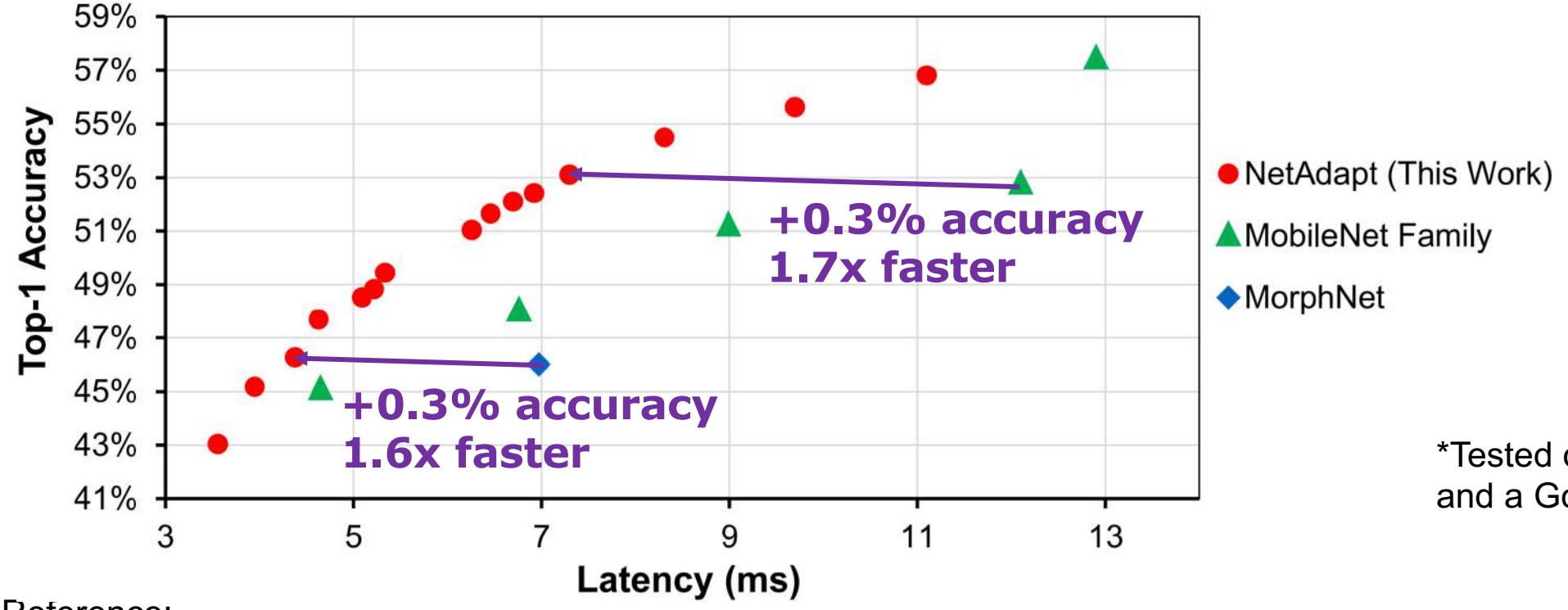
Code available at http://netadapt.mit.edu

[**Yang**, *ECCV* 2018]

In collaboration with Google's Mobile Vision Team

Improved Latency vs. Accuracy Tradeoff

- NetAdapt boosts the real inference speed of MobileNet by up to 1.7x with higher accuracy



*Tested on the ImageNet dataset and a Google Pixel 1 CPU

Reference:
MobileNet: Howard et al, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", arXiv 2017
MorphNet: Gordon et al., "Morphnet: Fast & simple resource-constrained structure learning of deep networks", CVPR 2018

Design Considerations for Co-Design

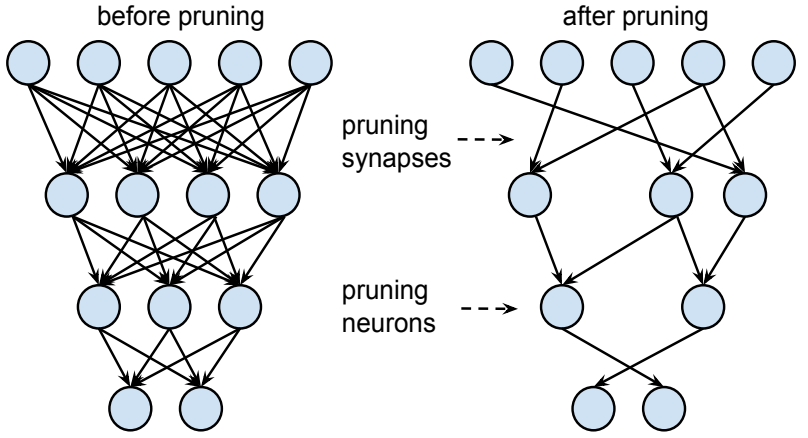
- **Impact on accuracy**
  - Must consider difficulty of dataset, task, and DNN model
    - e.g., Easy to reduce precision for an easy task (e.g., digit classification) → does method work for a more difficult task?
    - e.g., Easy to prune weights from over parameterized DNN models (e.g., AlexNet, VGG) → does method work on compact DNN models?
- **Does hardware cost exceed benefits?**
  - Need extra hardware to support variable precision and shapes or identify sparsity
    - e.g., Additional shift-and-add logic and registers for varying precision
  - Granularity impacts hardware overhead as well as accuracy
    - e.g., Fine-grained or coarse-grained (structured) sparsity
- **Evaluation**
  - Avoid evaluating impact based on number of weights or MACs as they may not be sufficient for evaluating energy consumption and latency
  - Baseline for precision: 8-bit for inference and 16-bit float for training
    - 32-bit float is a weak baseline

Tutorial Overview

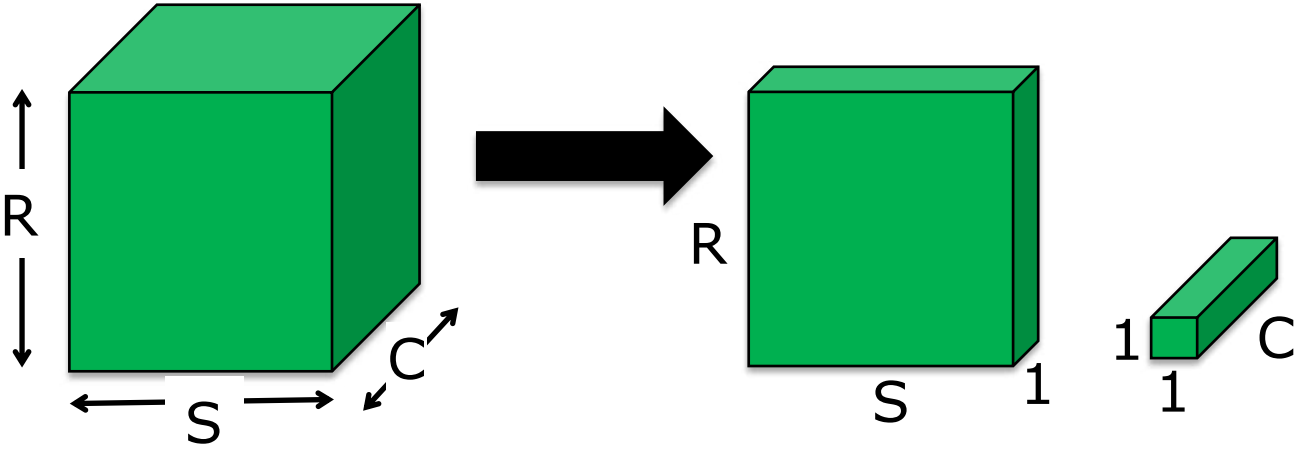
- Deep Learning Overview
- Key Metrics and Design Objectives
- Design Considerations
  - CPU and GPU Platforms
  - Specialized / Domain Specific Hardware (ASICs)
    - Efficient Dataflows
    - Algorithm (DNN Model) and Hardware Co-Design
    - **Flexibility and Scalability**
  - Other Platforms
    - Processing In Memory / In Memory Computing
    - Field Programmable Gate Arrays (FPGAs)
- Tools for Systematic Evaluation of DL Processors
- Should I Use Deep Learning for a Given Task?

Many Efficient DNN Design Approaches

Network Pruning



Compact Network Architectures



Reduce Precision

32-bit float

8-bit fixed

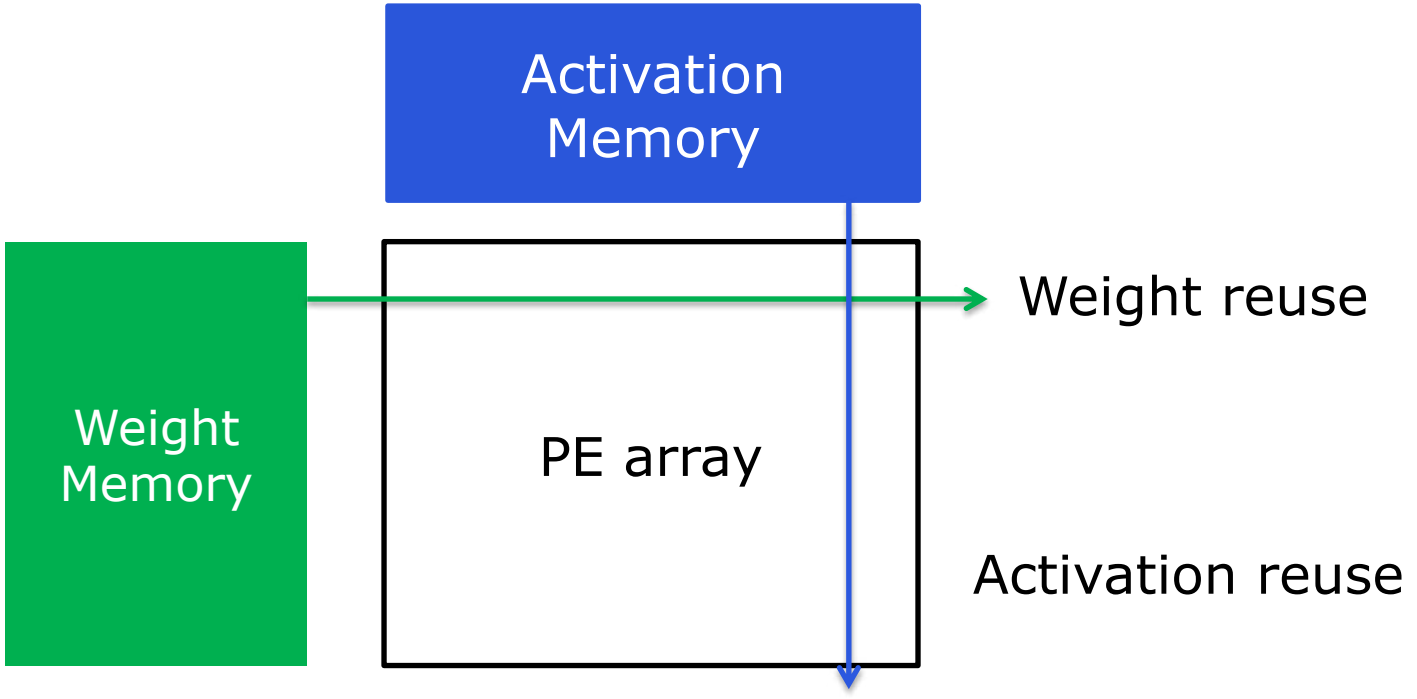
Binary

No guarantee that DNN algorithm designer will use a given approach. **Need flexible hardware!**

[**Chen**, *SysML* 2018]

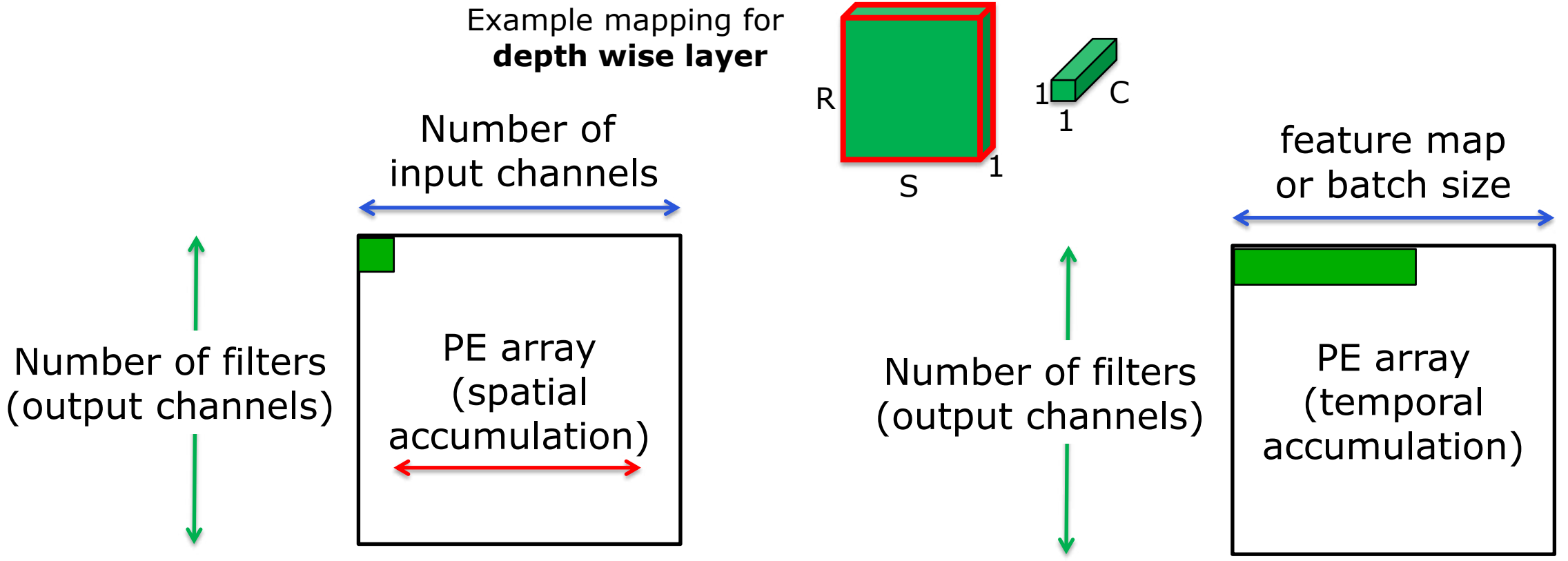
Limitations of Existing DNN Architectures

- Specialized DNN hardware often rely on certain properties of the DNN model in order to achieve high energy-efficiency
- Example: Reduce memory access by amortizing across PE array



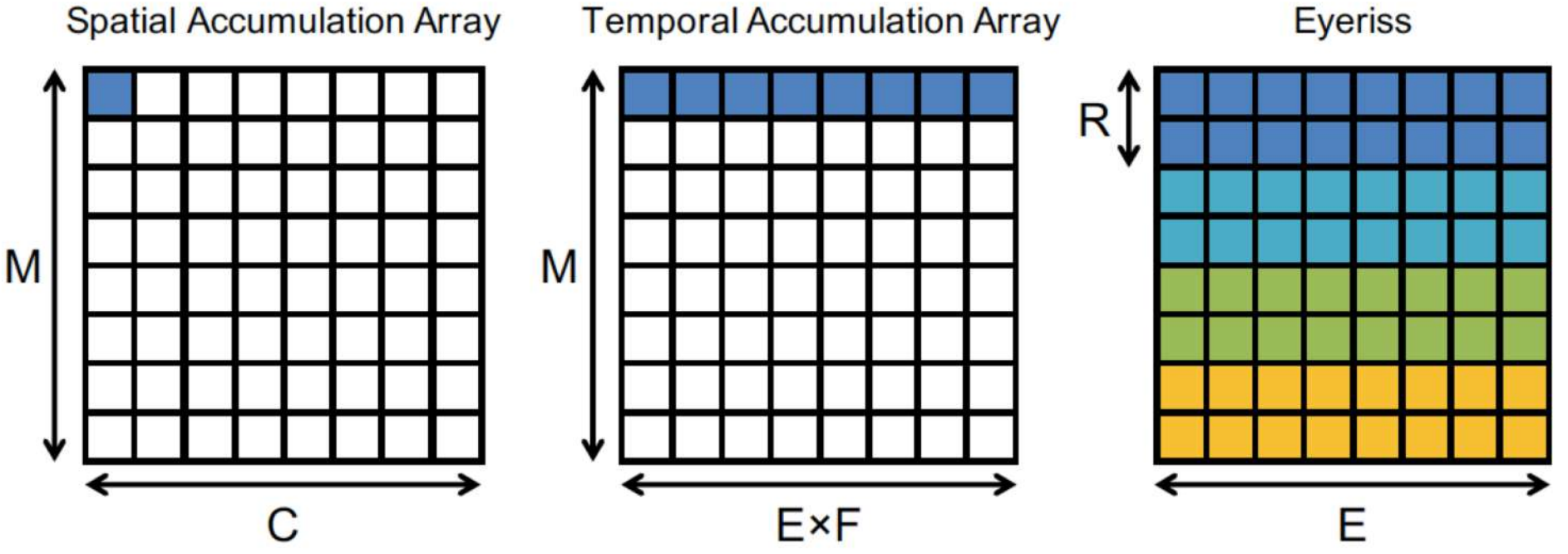
Limitations of Existing DNN Architectures

- Reuse depends on # of channels, feature map/batch size
  - Not efficient across all DNN models (e.g., compact DNNs)



Need Flexible Dataflow

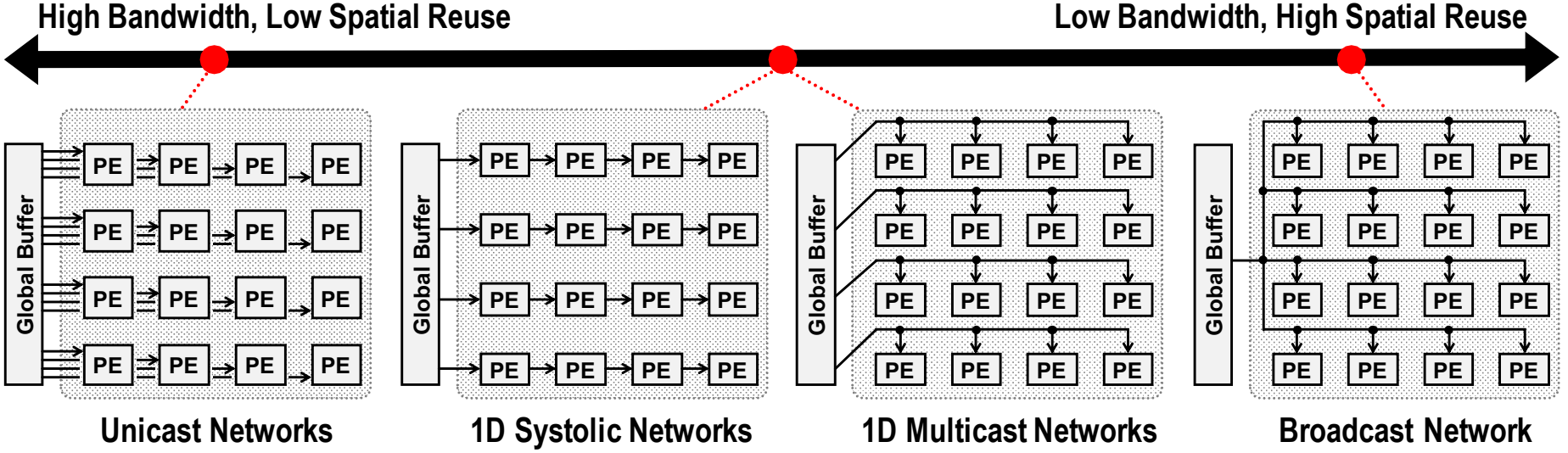
Use flexible dataflow (Row Stationary) to exploit reuse in any dimension of DNN to increase energy efficiency and array utilization



Example: Depth-wise layer

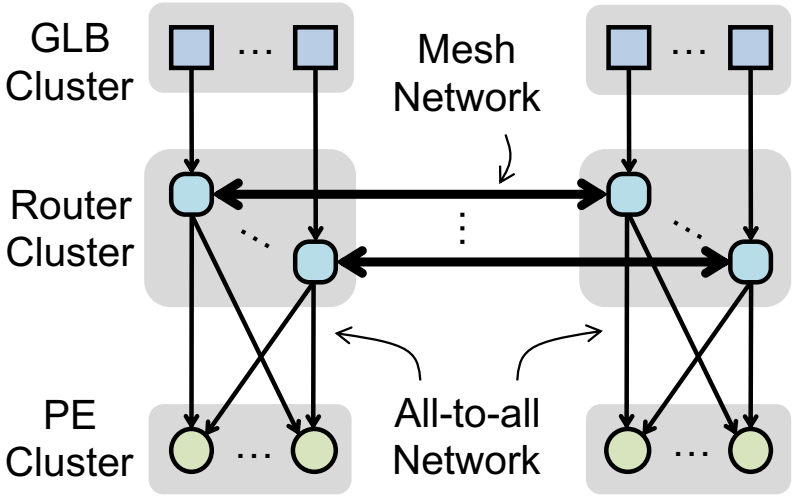
Need Flexible On-Chip Network for Varying Reuse

- When reuse available, need multicast to exploit spatial data reuse for energy efficiency and high array utilization
- When reuse not available, need unicast for high BW for weights for FC and weights & activations for high PE utilization
- An all-to-all on-chip network satisfies above but too expensive and not scalable

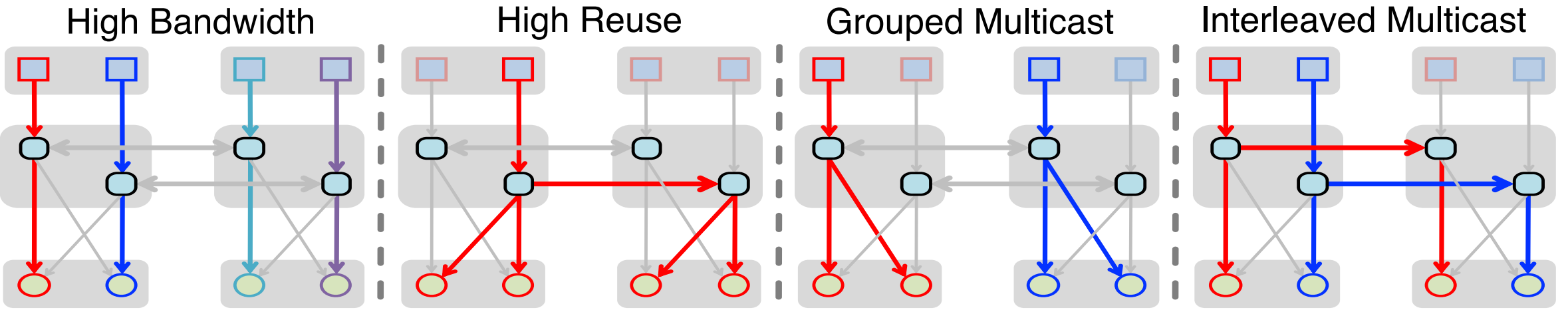


[**Chen**, *JETCAS* 2019]

Hierarchical Mesh



[**Chen**, *JETCAS* 2019]

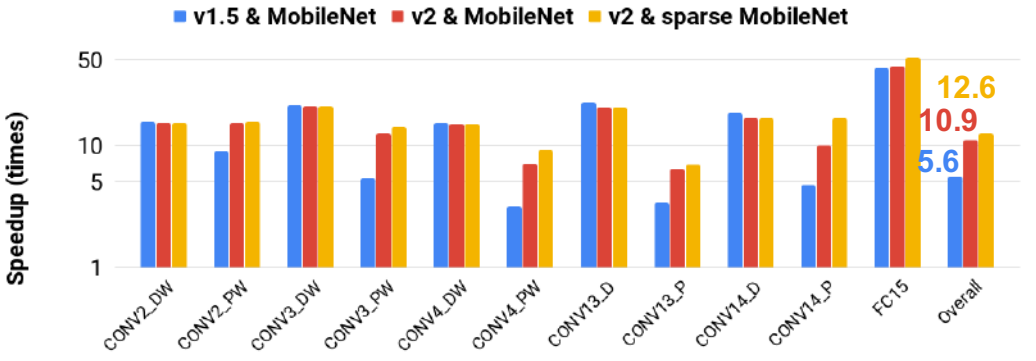


Eyeriss v2: Balancing Flexibility and Efficiency

Efficiently supports

- Wide range of filter shapes
  - Large and Compact
- Different Layers
  - CONV, FC, depth wise, etc.
- Wide range of sparsity
  - Dense and Sparse
- Scalable architecture

Over an order of magnitude faster and more energy efficient than Eyeriss v1



Speed up over Eyeriss v1 scales with number of PEs

# of PEs	256	1024	16384
AlexNet	17.9x	71.5x	1086.7x
GoogLeNet	10.4x	37.8x	448.8x
MobileNet	15.7x	57.9x	873.0x

[**Chen**, *JETCAS* 2019]

Design Considerations for Flexibility and Scalability

- Many of the existing DL processors rely on certain properties of the DNN Model
  - **Properties cannot be guaranteed** as the wide range techniques used for efficient DNN model design has resulted in a more diverse set of DNNs
  - DL processors should be sufficiently flexible to efficiently support a wide range of techniques

- Evaluate DL processors on a comprehensive set of benchmarks
  - MLPerf benchmark is a start, but may need more (e.g., sparsity, reduced precision, compact network architectures)

- Evaluate improvement in performance as resources scales up!
  - Multiple chips modules [**Zimmer**, *VLSI* 2019] and Wafer Scale [**Lie**, *HotChips* 2019]

Design Considerations for ASIC

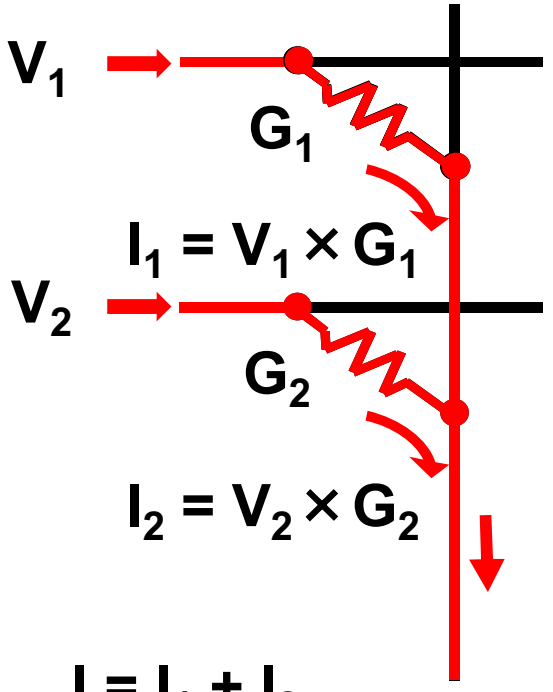
- **Increase PE utilization**
  - Flexible mapping and on-chip network for different DNN workloads → requires additional hardware
- **Reduce data movement**
  - Custom memory hierarchy and dataflows that exploit data reuse
  - Apply compression to exploit redundancy in data → requires additional hardware
- **Reduce time and energy per MAC**
  - Reduce precision → if precision varies requires additional hardware; impact on accuracy
- **Reduce unnecessary MACs**
  - Exploit sparsity → requires additional hardware; impact on accuracy
  - Exploit redundant operations → requires additional hardware

Tutorial Overview

- Deep Learning Overview
- Key Metrics and Design Objectives
- Design Considerations
  - CPU and GPU Platforms
  - Specialized / Domain Specific Hardware (ASICs)
    - Efficient Dataflows
    - Algorithm (DNN Model) and Hardware Co-Design
    - Flexibility and Scalability
  - **Other Platforms**
    - Processing In Memory / In Memory Computing
    - Field Programmable Gate Arrays (FPGAs)
- Tools for Systematic Evaluation of DL Processors
- Should I Use Deep Learning for a Given Task?

Performing MAC with Memory Storage Element

Activation is input voltage (V_i)
Weight is resistor conductance (G_i)



Psum is output current

$$I = I_1 + I_2 = V_1 \times G_1 + V_2 \times G_2$$

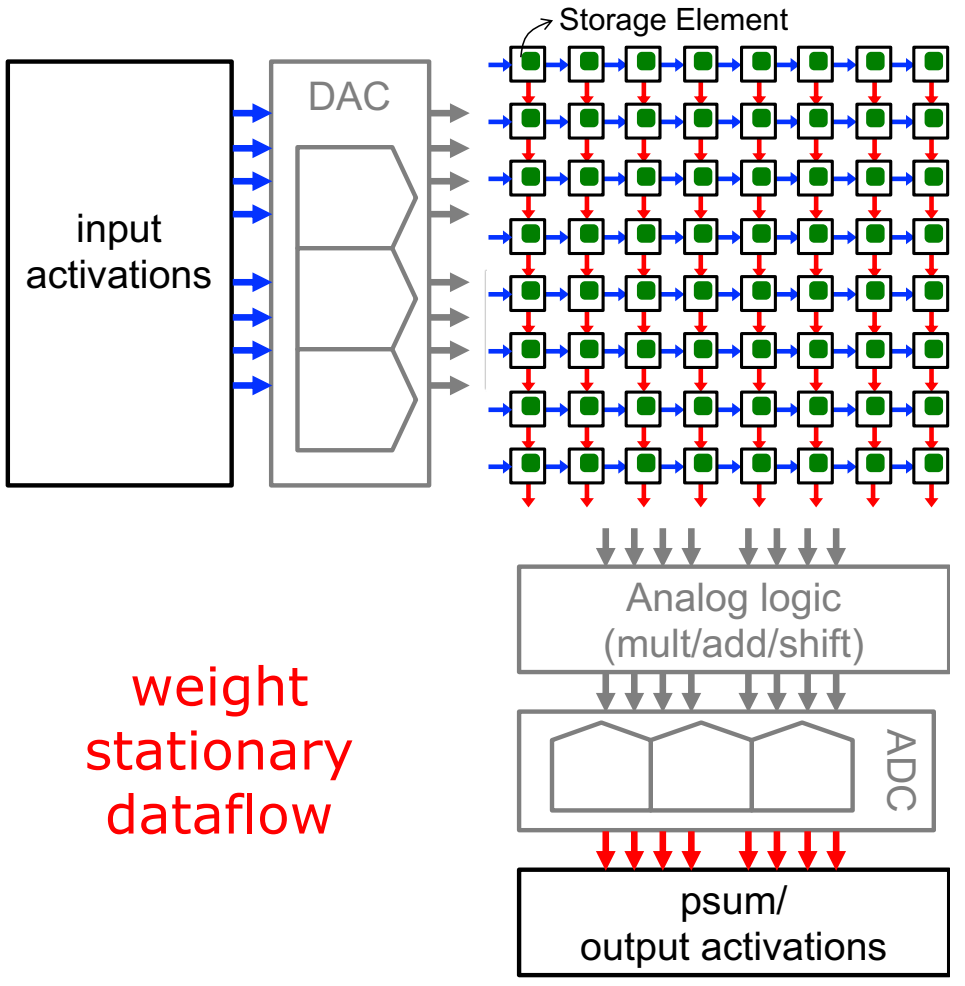
Image Source: [**Shafiee**, *ISCA* 2016]

- **Analog Compute**
  - Activations, weights and/or partial sums are encoded with analog voltage, current, or resistance
  - **Increased sensitivity** to circuit non-idealities: non-linearities, process, voltage, and temperature variations
  - Require A/D and D/A peripheral circuits to interface with digital domain
- **Multiplication**
  - eNVM (RRAM, STT-RAM, PCM) use **resistive device**
  - Flash and SRAM use **transistor** (I-V curve) or **local cap**
- **Accumulation**
  - Current summing
  - Charge sharing

Processing In Memory (PIM*)

* a.k.a. In-Memory Computing (IMC)

- Implement as **matrix-vector multiply**
  - Typically, matrix composed of stored weights and vector composed of input activations
- **Reduce weight data movement by moving compute into the memory**
  - Perform MAC with storage element or in peripheral circuits
  - Read out partial sums rather than weights → fewer accesses through peripheral circuits
- **Increase weight bandwidth**
  - Multiple weights accessed in parallel to keep MACs busy (high utilization)
- **Increase amount of parallel MACs**
  - Storage element can be higher area density than digital MAC
  - Reduce routing capacitance



weight stationary dataflow

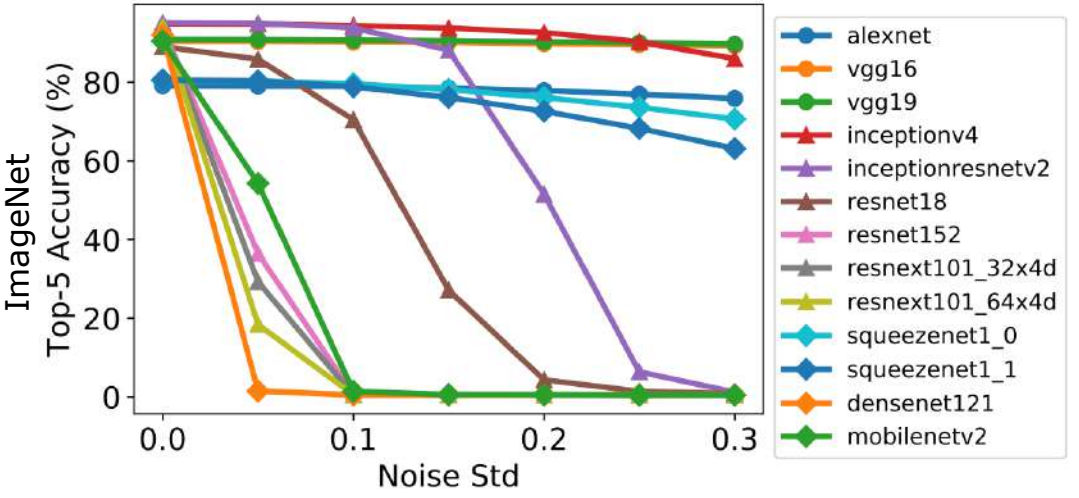
eNVM:[**Yu**, *PIEEE* 2018], SRAM:[**Verma**, *SSCS* 2019]

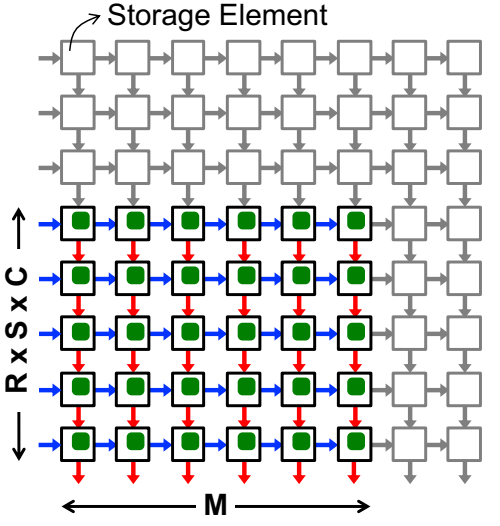
Design Considerations for PIM

- **Reduced precision due to non-idealities of analog compute**
  - For higher precision use
    - multiple storage elements per MAC → reduce area density and number of MACs
    - bit serial processing → increase cycles per MAC
  - **Note:** Per chip training may address variability but expensive in practice
- **A/D and D/A conversion increase energy consumption and reduce area density**
  - Large array size can amortize conversion cost → increase area density and data reuse
  - Array size limited by capacitance and resistance of bitline and wordline
- **PE Utilization**
  - Number of rows or columns read in parallel from array limited by peripheral circuits
- **Flexibility**
  - Limited mapping of weights due to sensitivity of analog compute and density requirements → affects utilization of array
  - Can be challenging to support sparsity
  - If expensive to write (e.g., eNVM), need to store entire DNN model on-chip

Design Considerations for DNNs on PIM

- Designing DNNs for PIM may differ from DNNs for digital processors
- Highest accuracy DNN on digital processor may be different on PIM
  - Accuracy drops based on robustness to non-idealities
- Reducing number of weights is less desirable
  - Since PIM is weight stationary, may be better to reduce number of activations
  - PIM tend to have larger arrays → fewer weights may lead to low utilization on PIM
- Current trend is deeper and smaller filters
  - For PIM, may be preferable to do shallower and larger filters





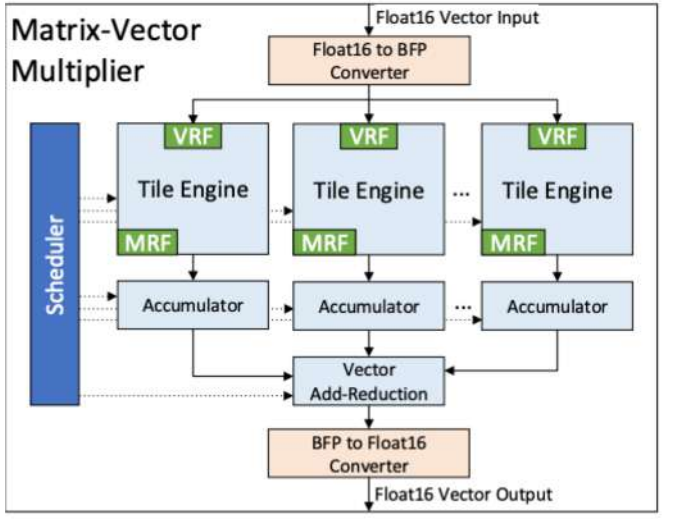
[**Yang**, *IEDM* 2019]

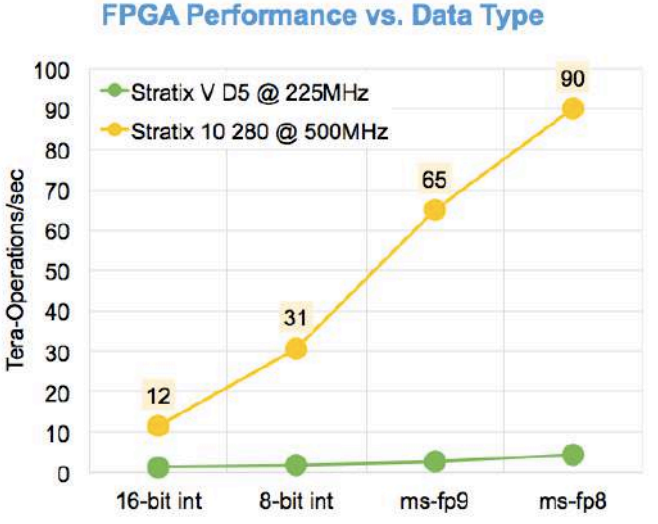
Field Programmable Gate Array (FPGA)

- Often implemented as **matrix-vector multiply**
  - e.g., Microsoft Brainwave NPU [**Fowers**, *ISCA* 2018]

- A popular approach uses **weight stationary dataflow** and stores all weights on FPGA for low latency (batch size of 1)
  - Referred to as *pinning/persistent weights*
  - Scale to larger DNNs with multiple FPGAs due to high external bandwidth capabilities

- Reduced precision to fit more weights and MACs on FPGA
  - e.g., custom ms-fp: 5-bit exponent (shared across vector), 2-3 bit mantissa → 96,000 MACs on Stratix





Design Considerations for FPGAs

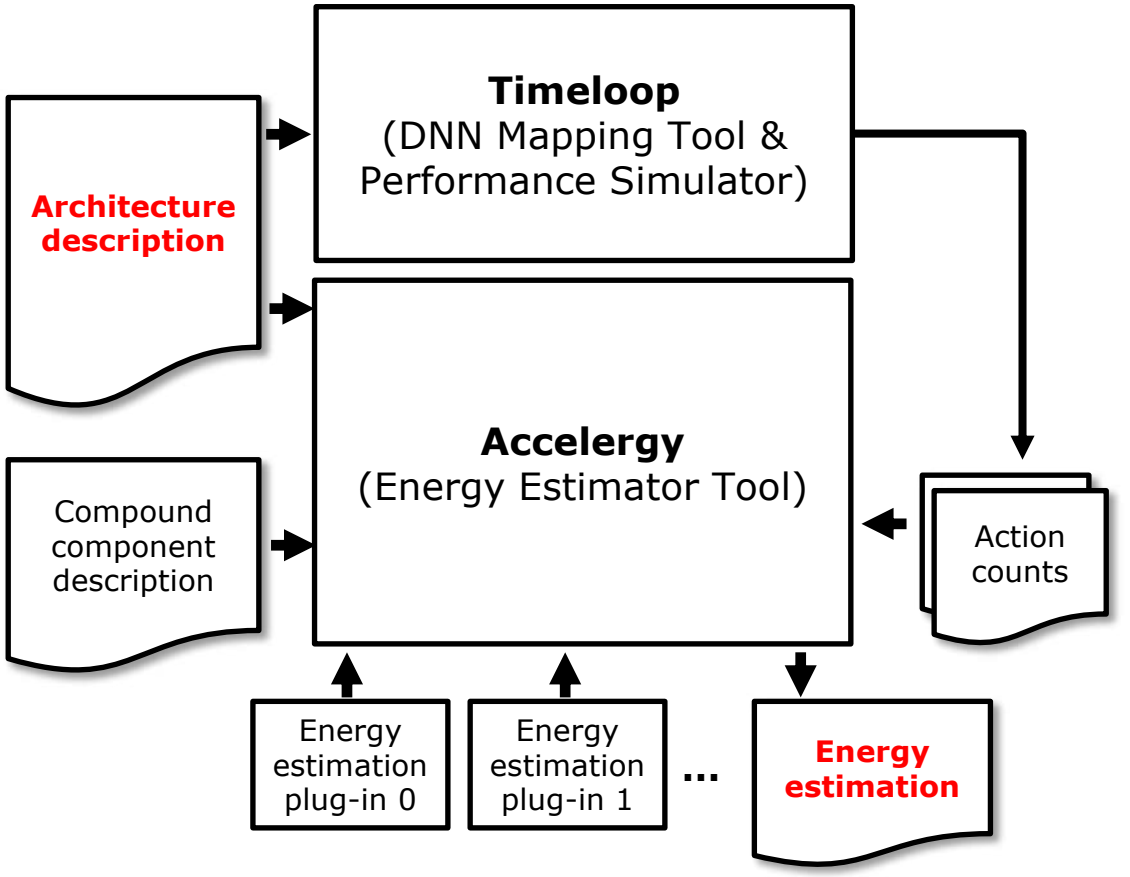
- **Increase number of PEs**
  - Use custom reduced precision → check impact on accuracy
  - Use Look up tables (LUTs) for MACs, but slower and more area than DSPs
    - Tradeoff between storage and compute → number of PEs versus PE utilization

- **Increase PE utilization**
  - Design memory hierarchy using on-chip memory resources
    - Store all weights on-chip when feasible (good for batch size 1)
  - Customized architecture and mapping for each layer
    - Tailor architecture (e.g., PE array dimension) at synthesis time to DNN model and increase efficiency (e.g., PE utilization) of underlying FPGA hardware

- **Specialized PE array added as co-processor to FPGA**
  - Example: Xilinx Alveo platform has a xDNN systolic array accelerator

DL Processor Evaluation Tools

- Require systematic way to
  - Evaluate and compare wide range of DL processor designs
  - Rapidly explore design space
- **Accelergy** [**Wu**, *ICCAD* 2019]
  - Early stage energy estimation tool at the architecture level
    - Estimate energy consumption based on architecture level components (e.g., # of PEs, memory size, on-chip network)
  - Evaluate architecture level energy impact of emerging devices
    - Plug-ins for different technologies
- **Timeloop** [**Parashar**, *ISPASS* 2019]
  - DNN mapping tool
  - Performance Simulator → Action counts



Open-source code available at:
http://accelergy.mit.edu

Summary

- **DNNs are a critical component in the AI revolution**, delivering record breaking accuracy on many important AI tasks for a wide range of applications; however, it comes at the cost of **high computational complexity**
- Efficient processing of DNNs is an important area of research with many promising **opportunities for innovation at various levels** of hardware design, including algorithm co-design
- When considering different DNN solutions it is important to **evaluate with the appropriate workload** in term of both input and model, and recognize that they are evolving rapidly
- It is important to consider **a comprehensive set of metrics when evaluating different DNN solutions**: accuracy, throughput, latency, power, energy, flexibility, scalability and cost

Should I Use Deep Learning for a Given Task?

- While deep learning gives state-of-the-art accuracy on many tasks, it may not be the best approach for all tasks. Some factors to consider include
- **How much training data is available?**
  - Deep learning requires a significant amount of data → in particular, labelled data (current state-of-the-art results rely on supervised learning)
- **How much computing resource is available?**
  - Despite the progress in the area of efficient deep learning, it can still require orders of magnitude more complexity than other machine learning based approaches
- **How critical is interpretability?**
  - Understanding why the DNN makes a certain decision is still an open area of research
  - DNN models can be fooled – increasing robustness is still an open area of research
  - In general, debugging what happens within a DNN can be challenging
- **Does a known model already exist?**
  - Many things in the world are based on known models or laws (e.g., Ohm's Law V=IR); it may be unnecessary to re-learn this from the data

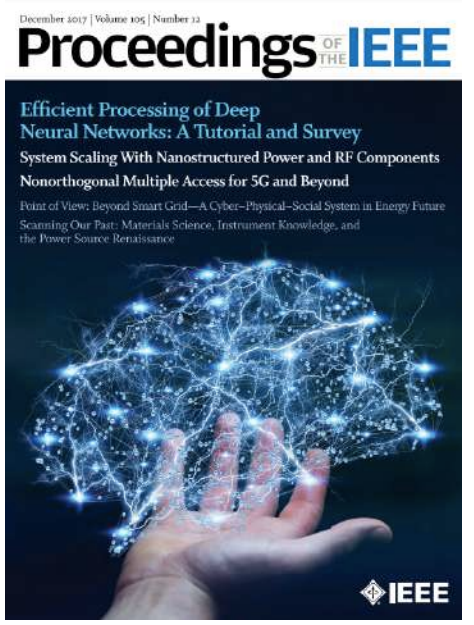
Papers at this year's ISSCC

- 7.1 A 3.4-to-13.3TOPS/W 3.6TOPS Dual-Core Deep-Learning Accelerator for Versatile AI Applications in 7nm 5G Smartphone SoC
- 7.2 A 12nm Programmable Convolution-Efficient Neural-Processing-Unit Chip Achieving 825TOPS
- 7.4 GANPU: A 135TFLOPS/W Multi-DNN Training Processor for GANs with Speculative Dual-Sparsity Exploitation

- 14.1 A 510nW 0.41V Low-Memory Low-Computation Keyword-Spotting Chip Using Serial FFT-Based MFCC and Binarized Depthwise Separable Convolutional Neural Network in 28nm CMOS
- 14.2 A 65nm 24.7μJ/Frame 12.3mW Activation-Similarity-Aware Convolutional Neural Network Video Processor Using Hybrid Precision, Inter-Frame Data Reuse and Mixed-Bit-Width Difference-Frame Data Codec
- 14.3 A 65nm Computing-in-Memory-Based CNN Processor with 2.9-to-35.8TOPS/W System Energy Efficiency Using Dynamic-Sparsity Performance-Scaling Architecture and Energy-Efficient Inter/Intra-Macro Data Reuse

- 15.2 A 28nm 64Kb Inference-Training Two-Way Transpose Multibit 6T SRAM Compute-in-Memory Macro for AI Edge Chips
- 15.3 A 351TOPS/W and 372.4GOPS Compute-in-Memory SRAM Macro in 7nm FinFET CMOS for Machine-Learning Applications
- 15.4 A 22nm 2Mb ReRAM Compute-in-Memory Macro with 121-28TOPS/W for Multibit MAC Computing for Tiny AI Edge Devices
- 15.5 A 28nm 64Kb 6T SRAM Computing-in-Memory Macro with 8b MAC Operation for AI Edge Chips

Additional Resources



V. Sze, Y.-H. Chen, T-J. Yang, J. Emer, "***Efficient Processing of Deep Neural Networks: A Tutorial and Survey***," Proceedings of the IEEE, Dec. 2017

Book Coming Soon!

DNN Tutorial website
http://eyeriss.mit.edu/tutorial.html

MIT Professional Education Course on
"Designing Efficient Deep Learning Systems"
http://professional-education.mit.edu/deeplearning

For updates

Follow @eems_mit

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

References (1 of 3)

Deep Learning Overview

- Transformer: Vaswani et al, "Attention is all you need," NeurIPS 2017
- LeNet: LeCun, Yann, et al. "Gradient-based learning applied to document recognition," Proc. IEEE 1998
- AlexNet: Krizhevsky et al. "Imagenet classification with deep convolutional neural networks," NeurIPS 2012
- VGGNet: Simonyan et al.. "Very deep convolutional networks for large-scale image recognition," ICLR 2015
- GoogleNet: Szegedy et al. "Going deeper with convolutions," CVPR 2015
- ResNet: He et al. "Deep residual learning for image recognition," CVPR 2016
- EfficientNet: Tan et al., "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019

Key Metrics and Design Objectives

- Sze et al., "Hardware for machine learning: Challenges and opportunities," CICC 2017, https://www.youtube.com/watch?v=8Qa0E1jdkrE&feature=youtu.be
- Sze et al., "Efficient processing of deep neural networks: A tutorial and survey," Proc. IEEE 2017
- Williams et al., "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," CACM 2009
- Chen et al., Eyexam, https://arxiv.org/abs/1807.07928

CPU and GPU Platforms

- Mathieu et al., "Fast training of convolutional networks through FFTs," ICLR 2014
- Cong et al., "Minimizing computation in convolutional neural networks," ICANN 2014
- Lavin et al., "Fast algorithms for convolutional neural networks," CVPR 2016

References (2 of 3)

Specialized Hardware (ASICs): Dataflow, Reduced Precision, Sparsity and Compact Model

- Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," ISCA 2016, http://eyeriss.mit.edu
- Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations," NeurIPS 2015
- Li et al., "Ternary weight networks," NeurIPS Workshop 2016
- Rategari et al., "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," ECCV 2016
- Lee et al., "LogNet: Energy-efficient neural networks using logarithmic computation," ICASSP 2017
- Lecun et al., Optimal Brain Damage," NeurIPS 1990
- Han et al., "Learning both weights and connections for efficient neural networks," NeurIPS 2015
- Chen et al., "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," ISSCC 2016, http://eyeriss.mit.edu
- Albericio et al., "Cnvlutin: Ineffectual-neuron-free deep neural network computing", ISCA 2016
- Yang et al., "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," CVPR 2017, http://eyeriss.mit.edu/energy.html
- Yang et al., "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," ECCV 2018, http://netadapt.mit.edu
- Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications", arXiv 2017
- Camus et al., "Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," JETCAS 2019

References (3 of 3)

Specialized Hardware (ASICs): Flexibility and Scalability

- Chen et al., "Understanding the Limitations of Existing Energy-Efficient Design Approaches for Deep Neural Networks," SysML 2018, https://www.youtube.com/watch?v=XCdy5egmvaU
- Chen et al., "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," JETCAS 2019
- Zimmer et al., "A 0.11pJ/Op, 0.32-128 TOPS, Scalable Multi-Chip-Module-based Deep Neural Network Accelerator with Ground-Reference Signaling in 16nm," VLSI 2019
- Lie (Cerebras), "Wafer Scale Deep Learning," Hot Chips 2019

Other Platforms: Processing In Memory / In-Memory Computing and FPGAs

- Verma et al., "In-Memory Computing: Advances and prospects," ISSCC Tutorial 2018 / SSCS Magazine 2019
- Yu, "Neuro-Inspired Computing with Emerging Nonvolatile Memorys," Proc. IEEE 2018
- Yang et al., "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators," IEDM 2019
- Fowers et al., "A configurable cloud-scale DNN processor for real-time AI," ISCA 2018

DL Processor Evaluation Tools

- Wu et al., "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," ICCAD 2019, http://accelergy.mit.edu
- Parashar et al., "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," ISPASS 2019