DNN Model and Hardware Co-Design

ISCA Tutorial (2019)

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

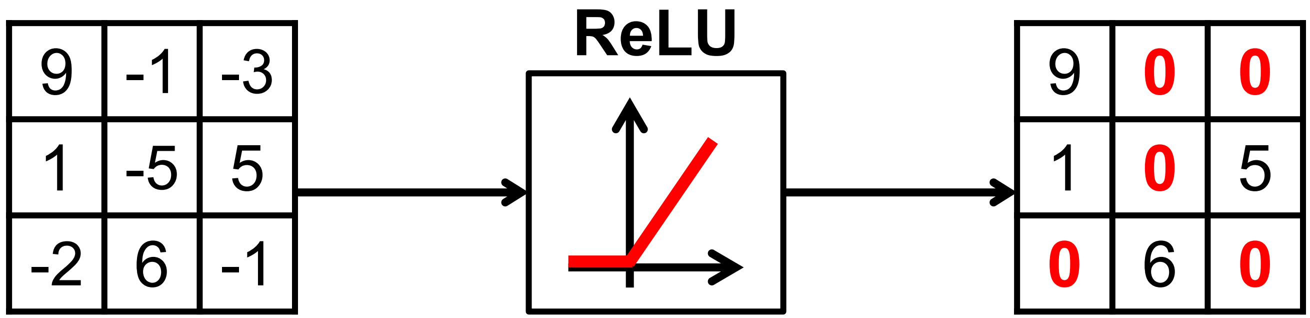
Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang

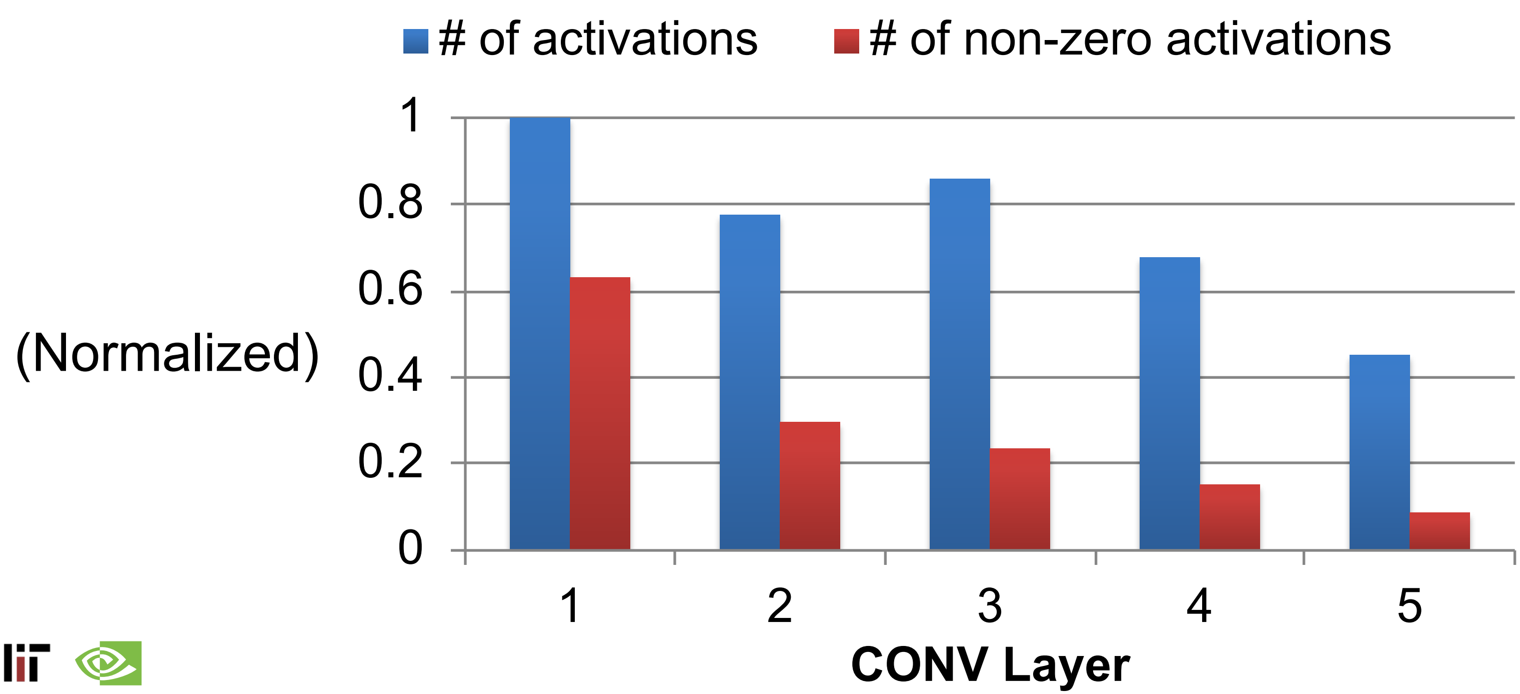
Reduce Number of Ops and Weights

- Exploit Activation Statistics
- Exploit Weight Statistics
- Exploit Dot Product Computation
- Decomposed Trained Filters
- Knowledge Distillation

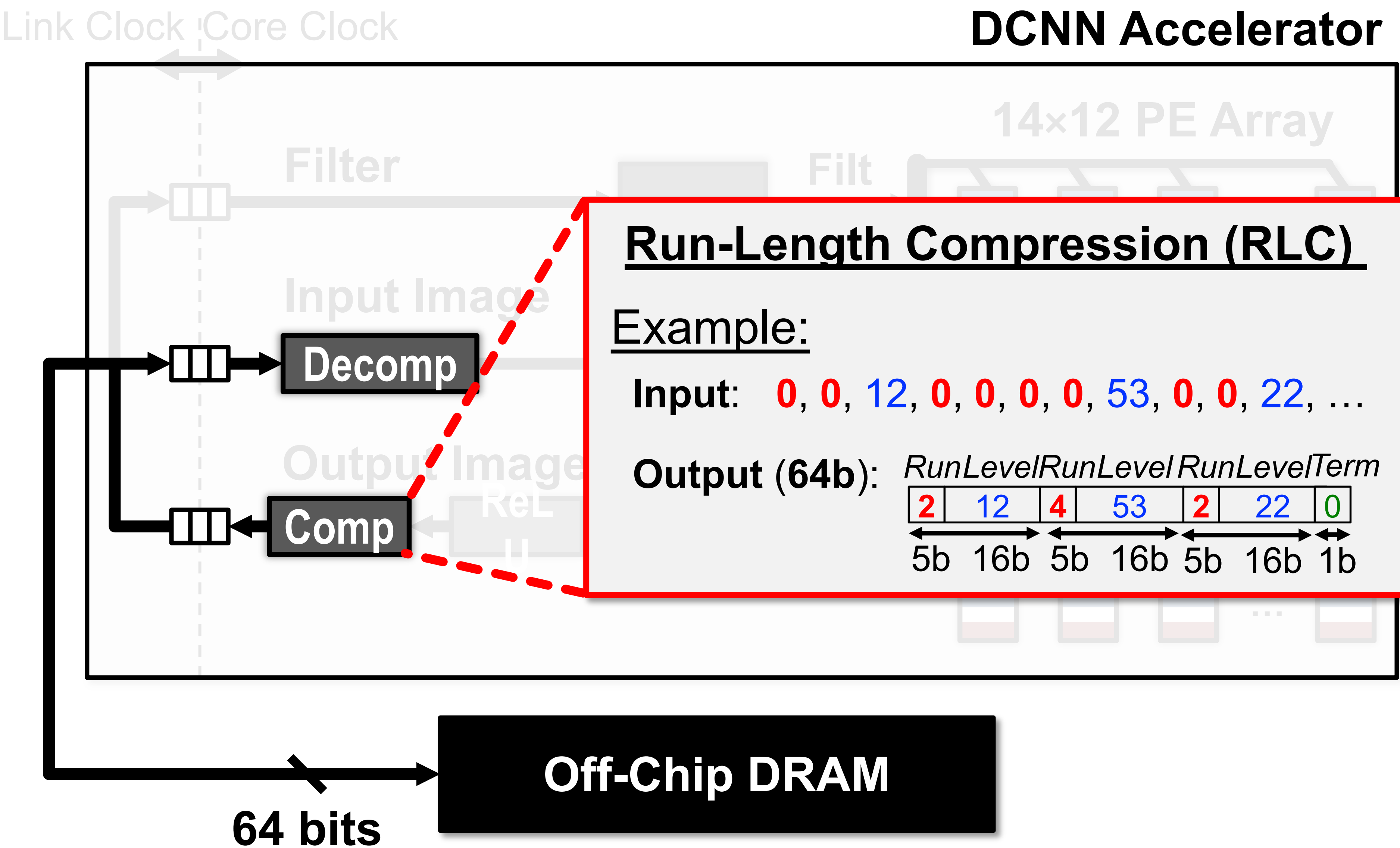
Sparsity in Fmaps

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



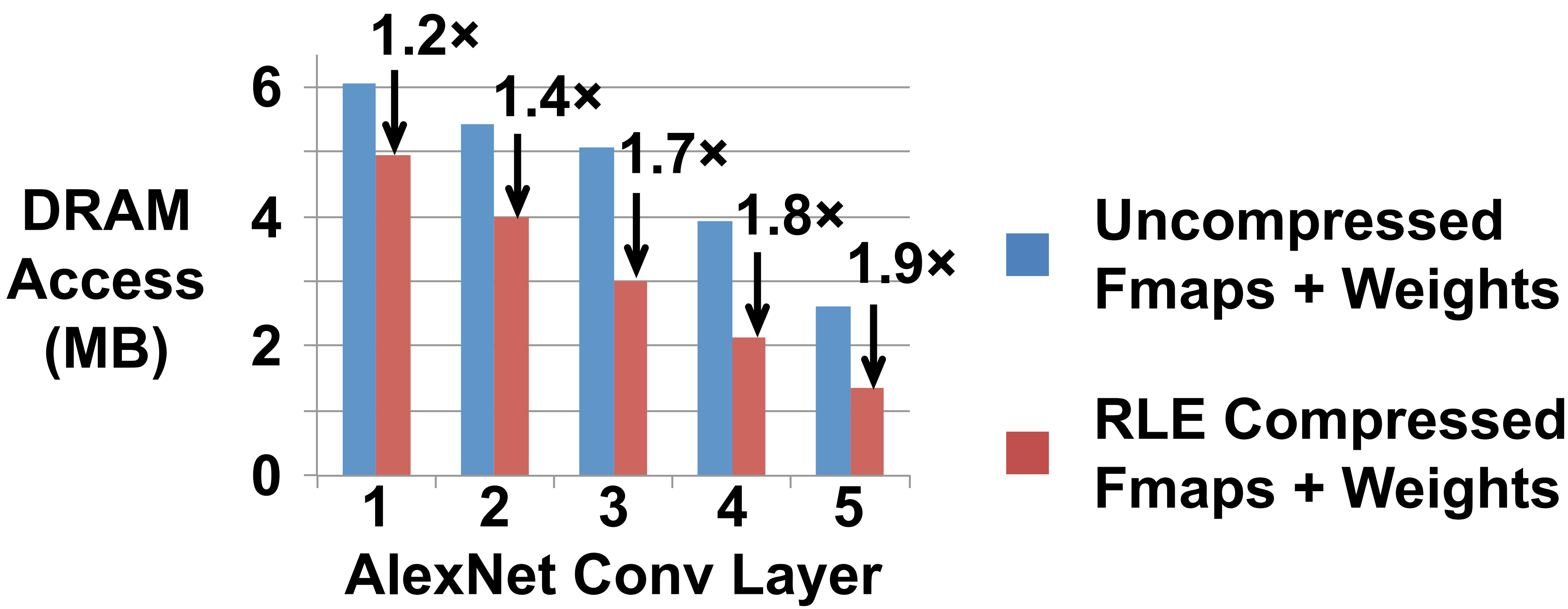


I/O Compression in Eyeriss



[Chen et al., ISSCC 2016]

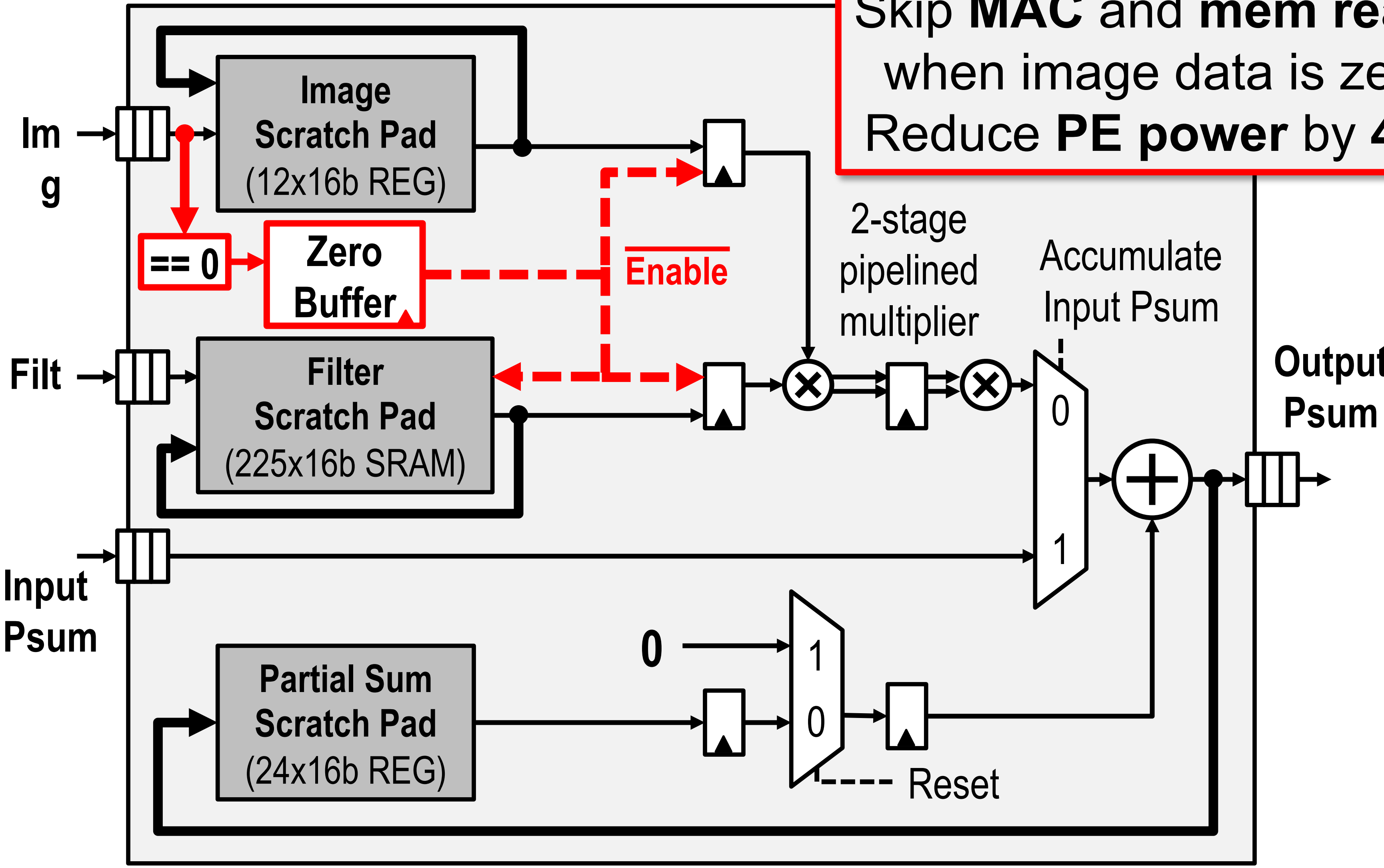
Compression Reduces DRAM BW



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

[Chen et al., ISSCC 2016]

Data Gating / Zero Skipping in Eyeriss

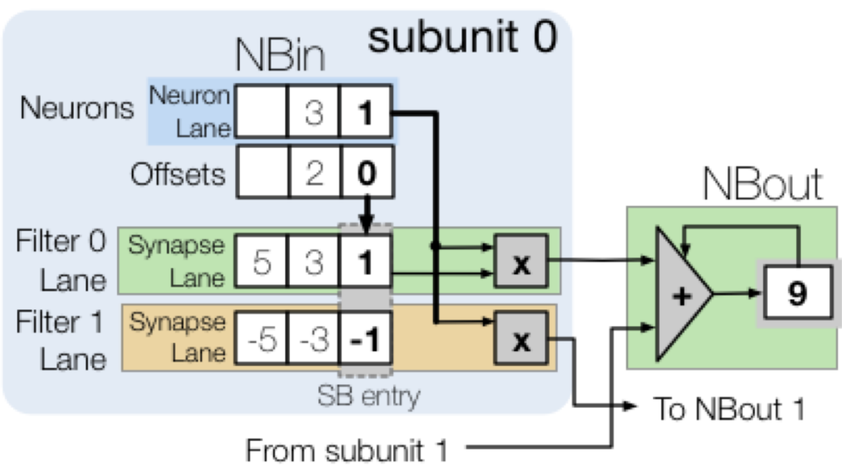


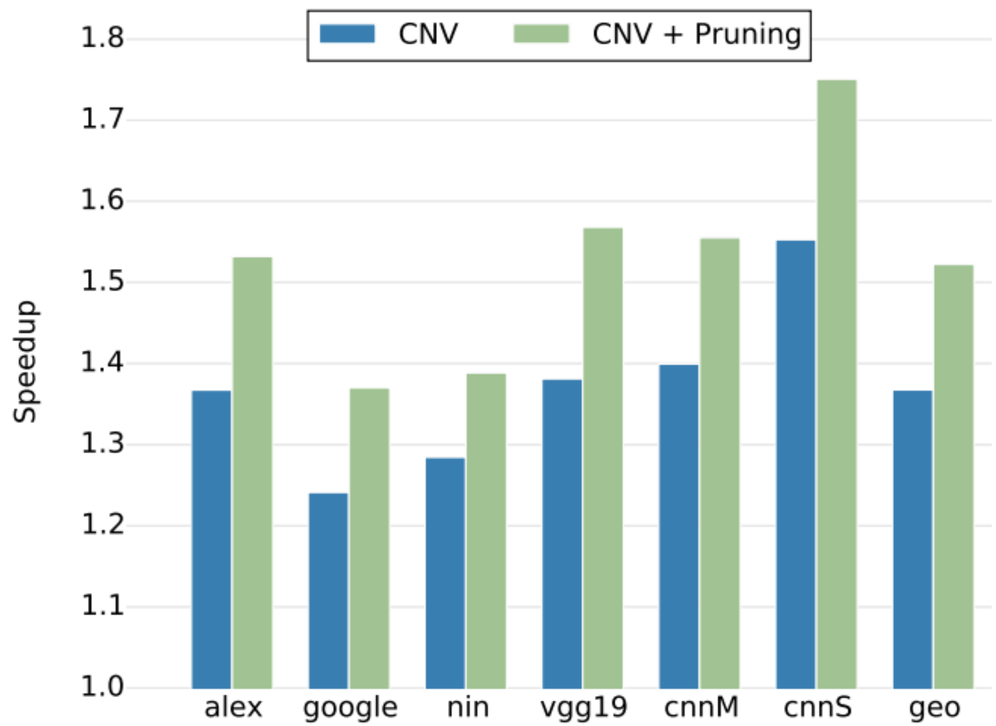
Skip **MAC** and **mem reads** when image data is zero.
Reduce **PE power** by **45%**

[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)**



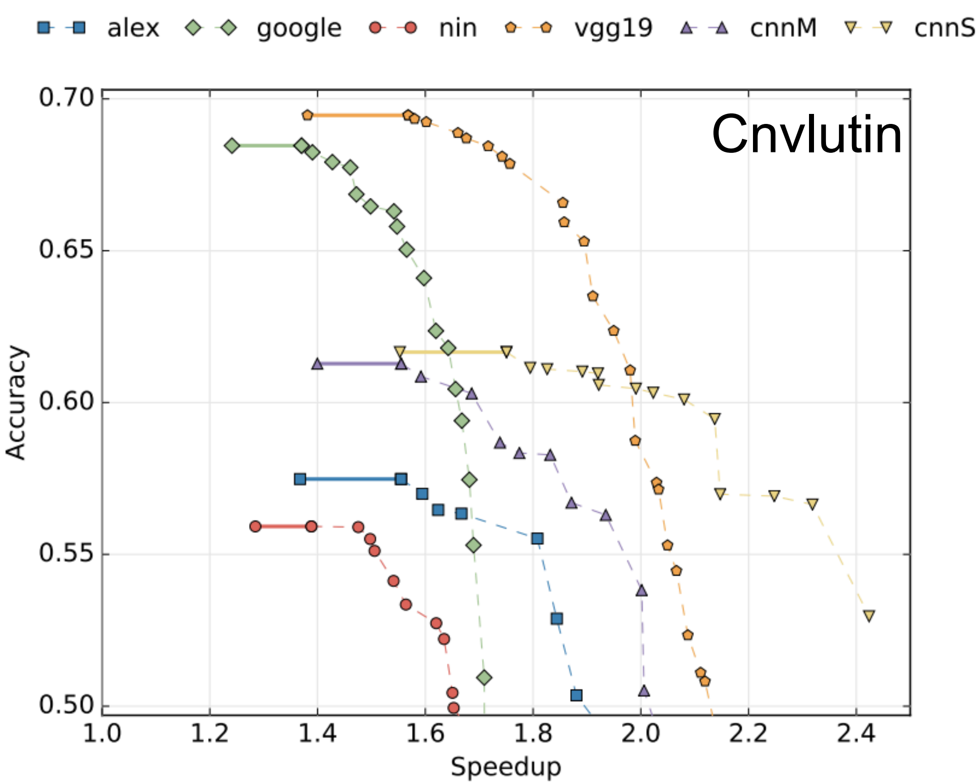


[Albericio et al., ISCA 2016]

Pruning Activations

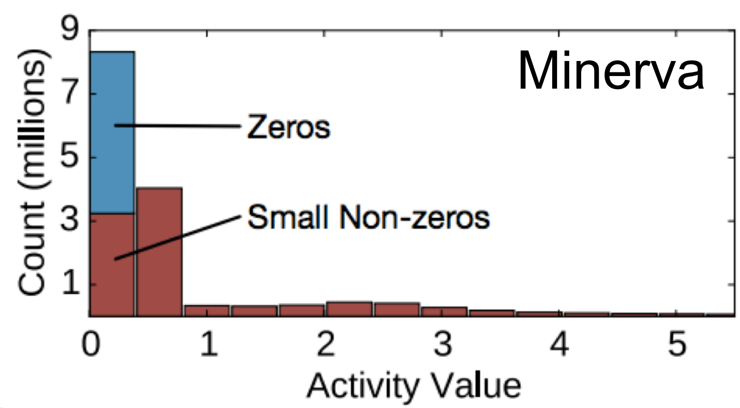
Remove small activation values

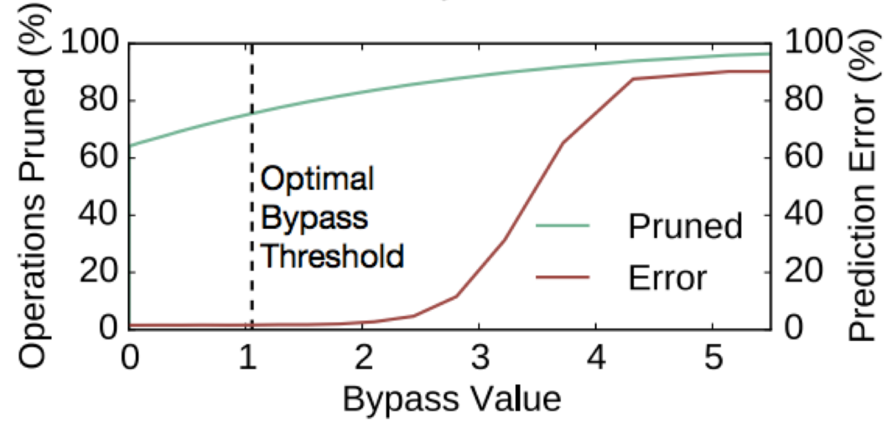
Speed up 11% (ImageNet)



[Albericio et al., ISCA 2016]

Reduce power 2x (MNIST)

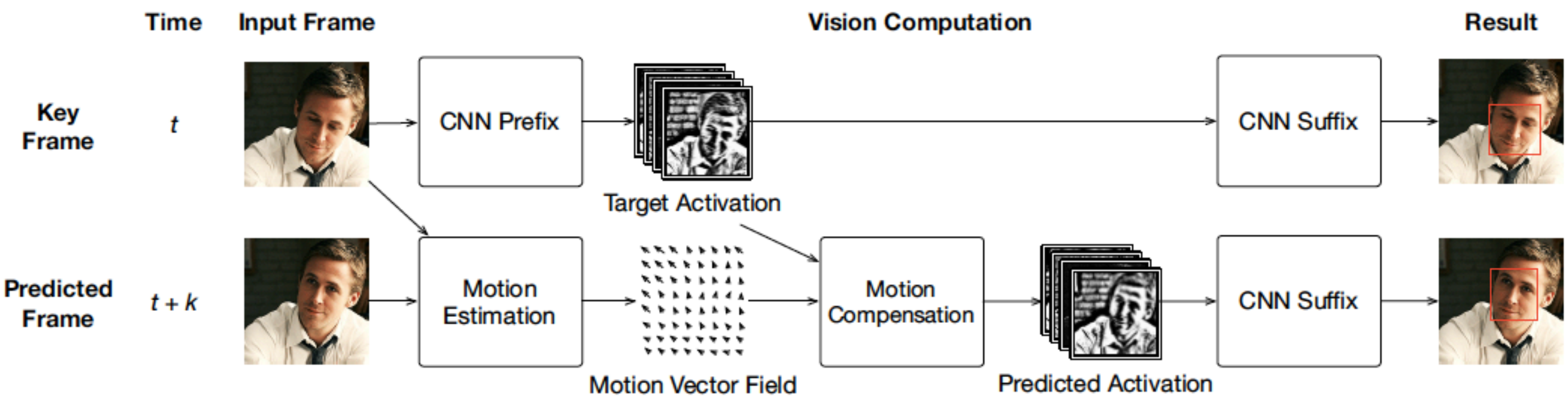




[Reagen et al., ISCA 2016]

Exploit Correlation in Input Data

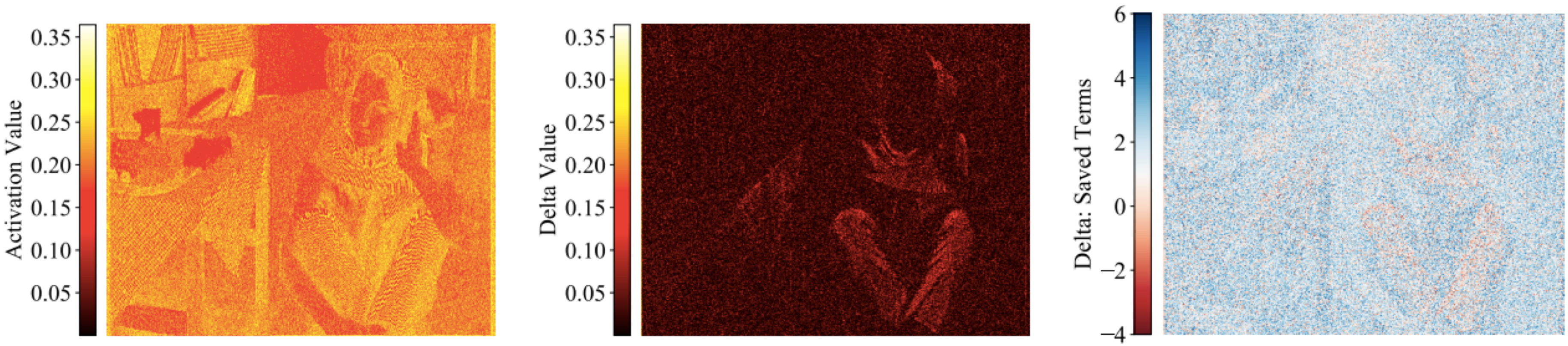
- **Exploit Temporal Correlation of Inputs**
  - Reduce amount of computation if there is temporal correlation between frames
  - Requires additional storage and need to measure redundancy (e.g. motion vector for videos)
  - Application specific (e.g. videos) – requires that the same operation is done for each frame (not always the case)



[Zhang et al., FAST, CVPRW 2017], [EVA2, ISCA 2018], [Euphrates, ISCA 2018], [Riera et al., ISCA 2018]

Exploit Correlation in Input Data

- **Exploit Spatial Correlation of Inputs**
  - Delta code neighboring values (activation) resulting in sparse inputs to each layer
  - Reduces storage cost and data movement for improvement in energy-efficiency and throughput



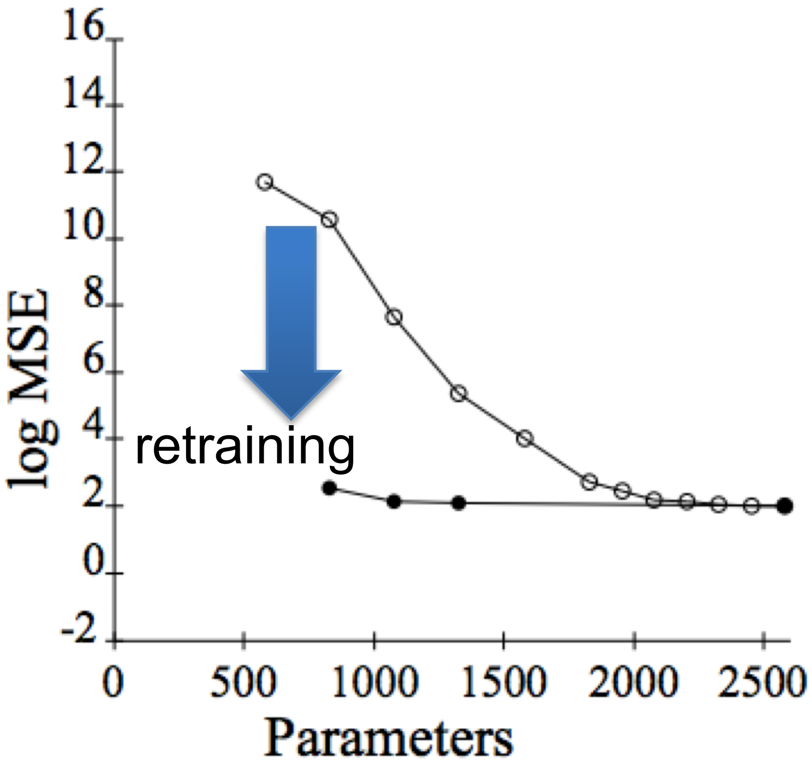
(a) Activation values (b) Deltas along the X-Axis (c) Reduction in effectual terms per imap activation

[Diffy, MICRO 2018]

Pruning – Make Weights Sparse

- **Optimal Brain Damage**

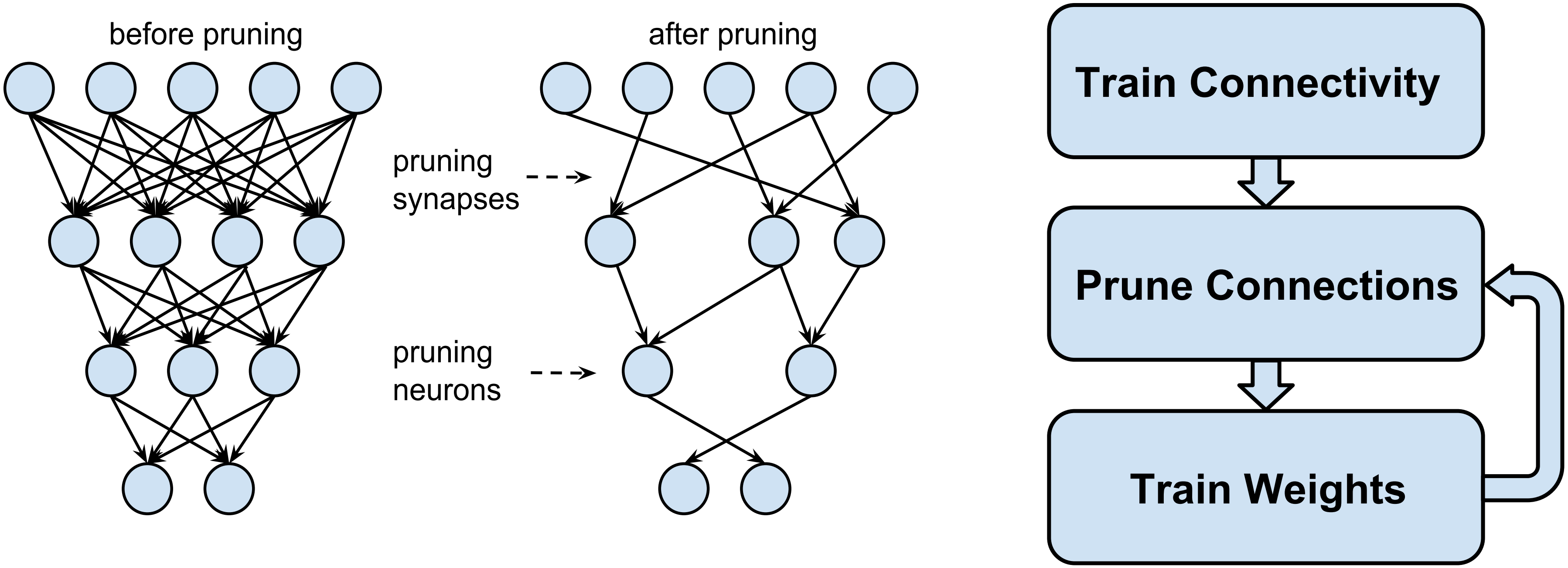
1. **Choose a reasonable network architecture**
2. **Train network until reasonable solution obtained**
3. **Compute the second derivative for each weight**
4. **Compute saliencies (i.e. impact on training error) for each weight**
5. **Sort weights by saliency and delete low-saliency weights**
6. **Iterate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



[Lecun et al., NeurIPS 1989]

Pruning – Make Weights Sparse

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



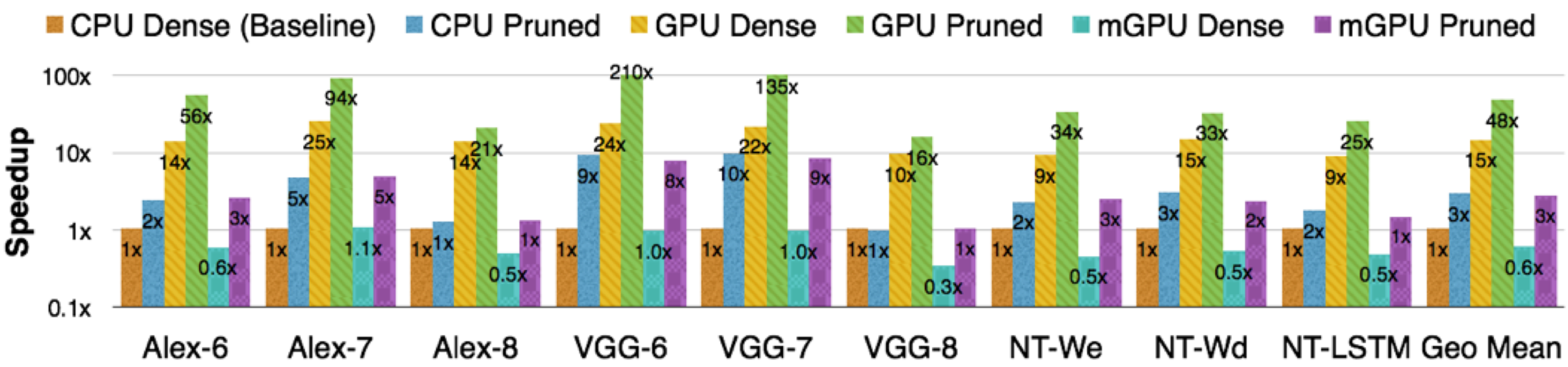
Example: *AlexNet*
Weight Reduction: *CONV layers 2.7x, FC layers 9.9x*
(Most reduction on fully connected layers)
Overall: *9x weight reduction, 3x MAC reduction*

[Han et al., NeurIPS 2015]

Speed up of Weight Pruning on CPU/GPU

On Fully Connected Layers Only

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

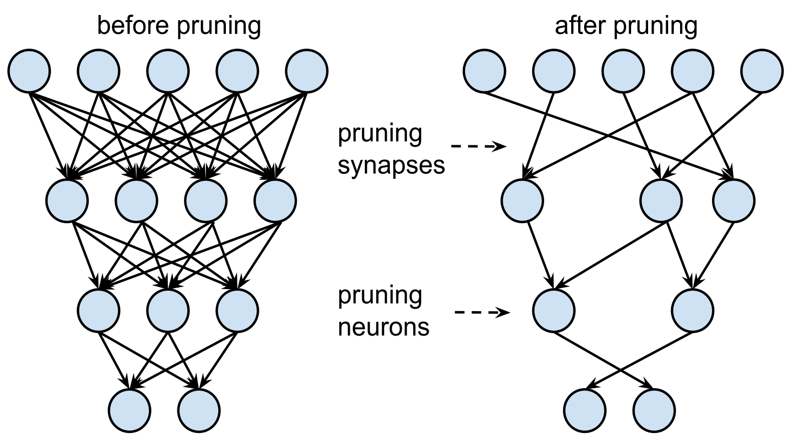
Batch size = 1

[Han et al., NeurIPS 2015]

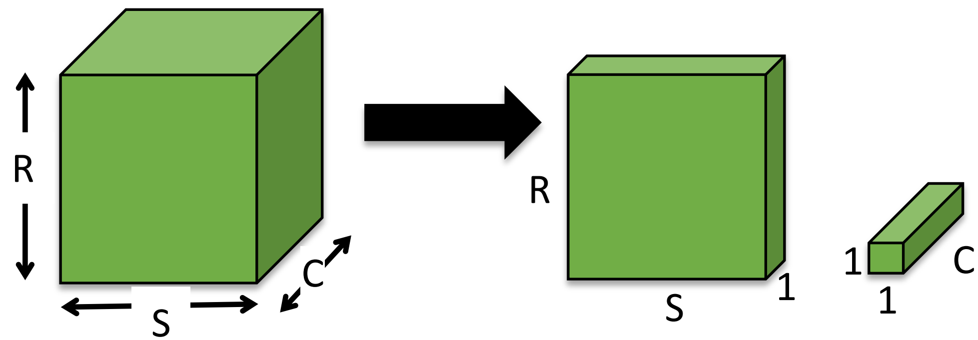
Design of Efficient DNN Algorithms

- Popular efficient DNN algorithm approaches

Network Pruning



Compact Network Architectures

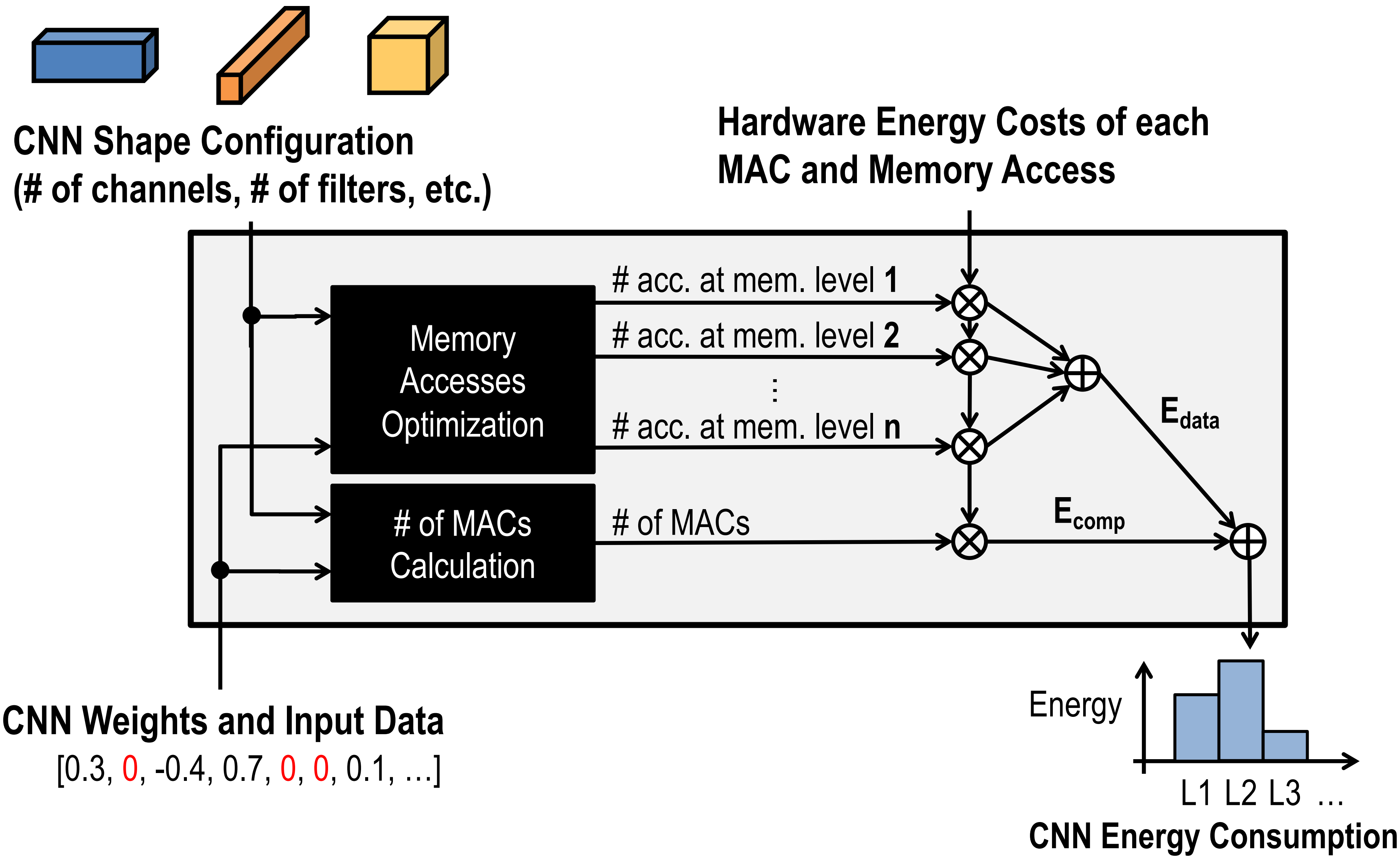


Examples: SqueezeNet, MobileNet

... also reduced precision

- Focus on reducing **number of MACs and weights**
- **Does it translate to energy savings and reduced latency?**

Energy-Evaluation Methodology

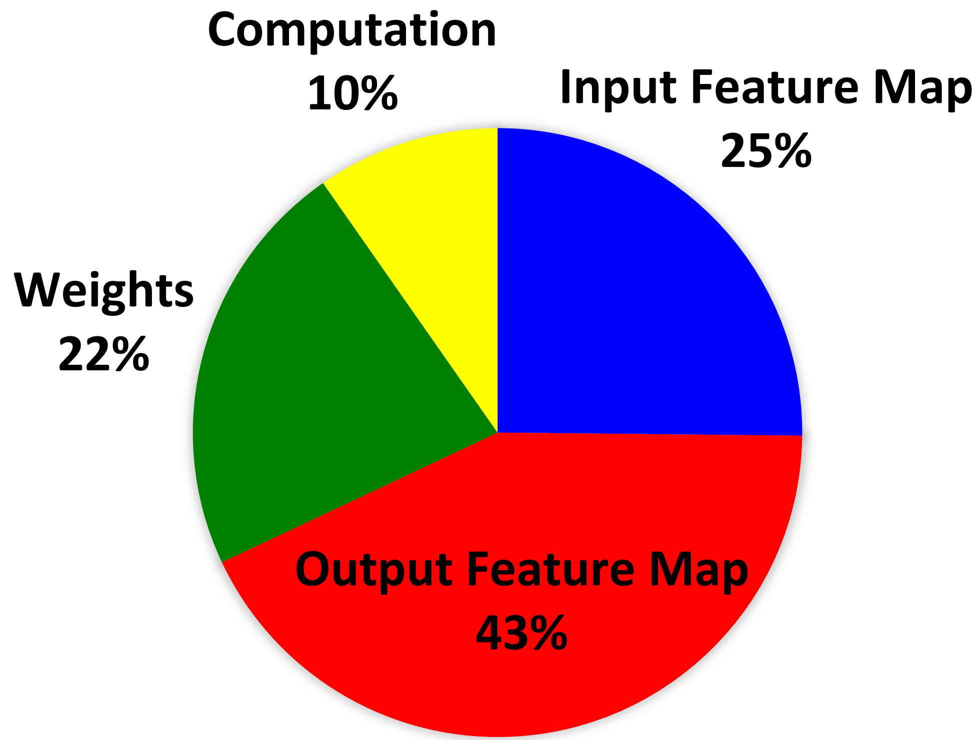


Evaluation tool available at http://eyeriss.mit.edu/energy.html

Key Observations

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

Energy Consumption of GoogLeNet

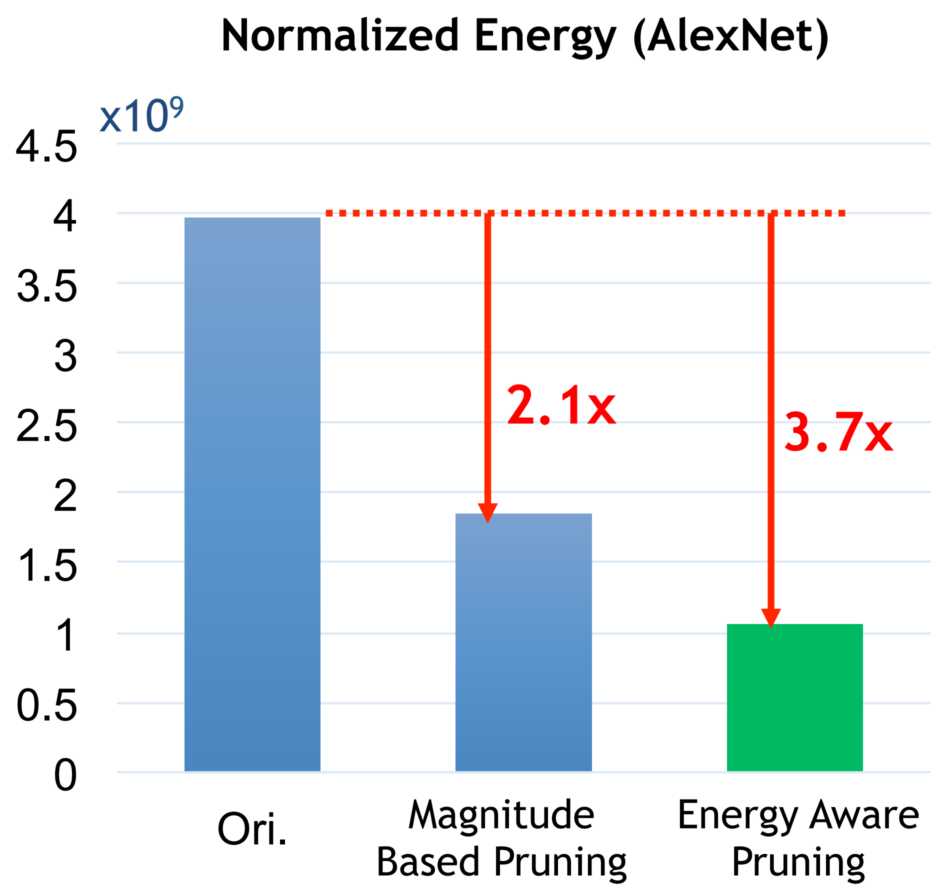


[Yang et al., CVPR 2017]

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 most energy first
- EAP reduces AlexNet energy by **3.7x** and outperforms the previous work that uses magnitude-based pruning by **1.7x**

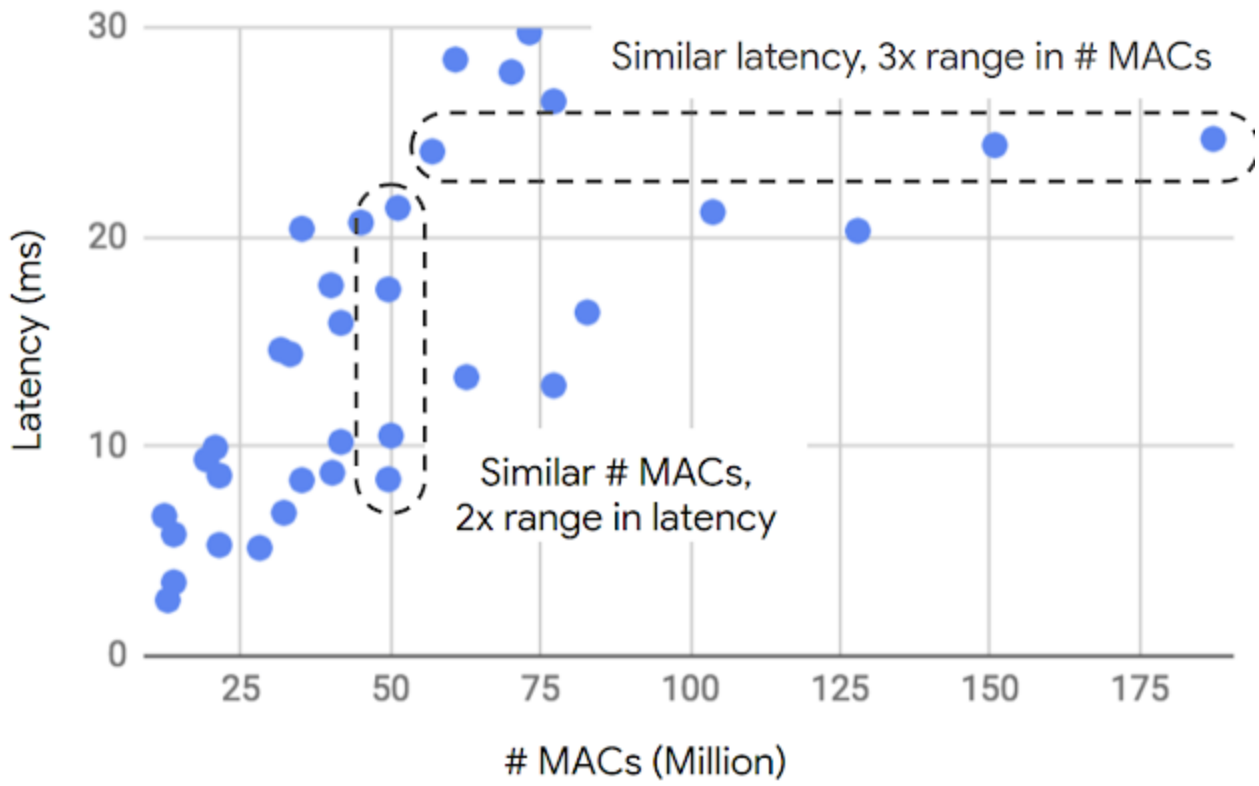


Pruned models available at http://eyeriss.mit.edu/energy.html

[Yang et al., CVPR 2017]

of Operations vs. Latency

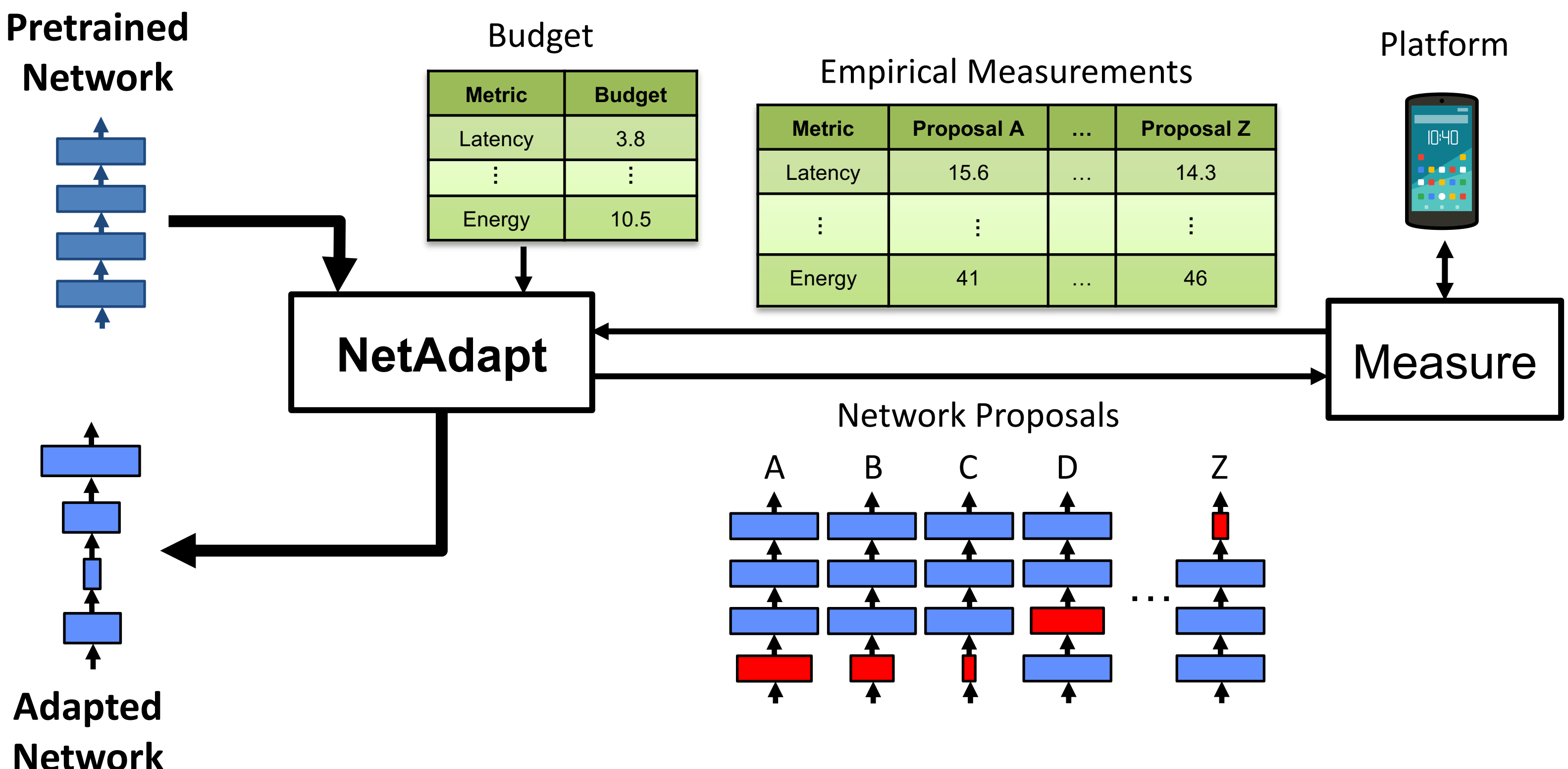
- # of operations (MACs) does not approximate latency well



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

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)

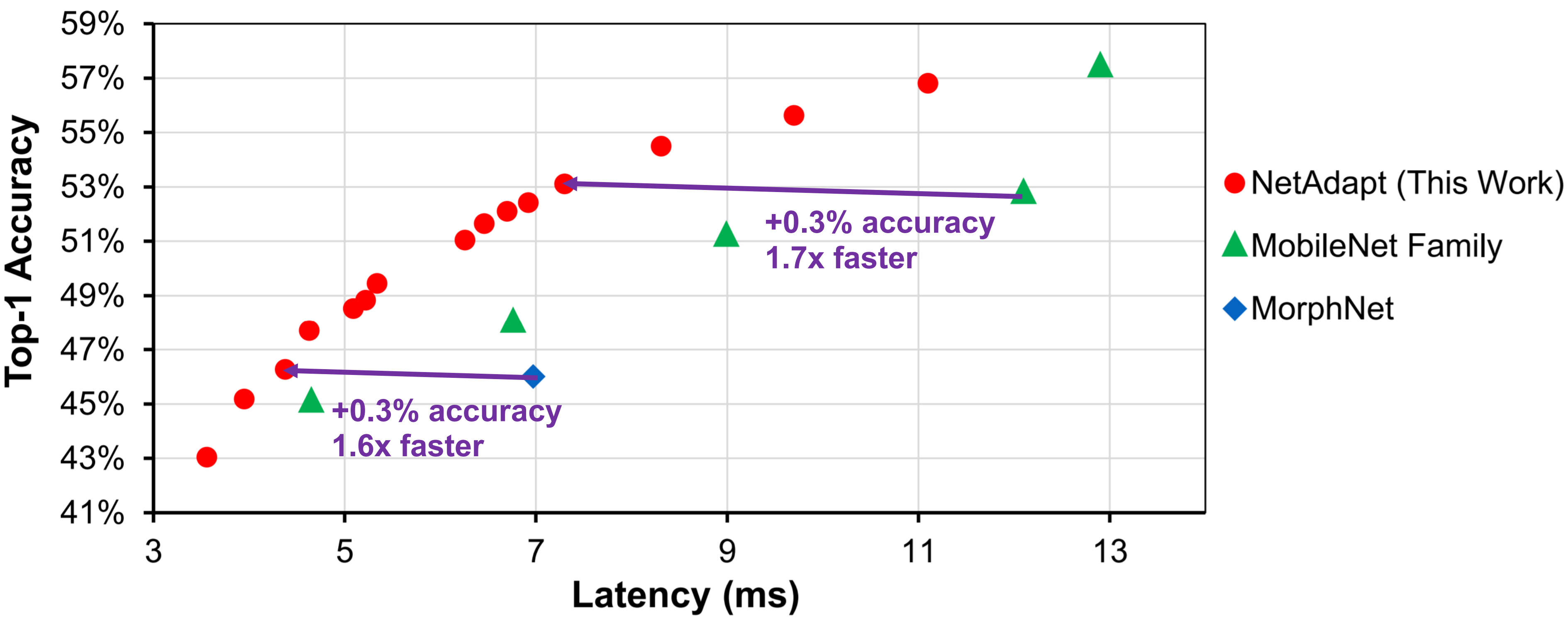


Code to be released at http://netadapt.mit.edu

[Yang et al., ECCV 2018]

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

[Yang et al., ECCV 2018]

Compression of Weights & Activations

- **Compress weights and activations 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**}

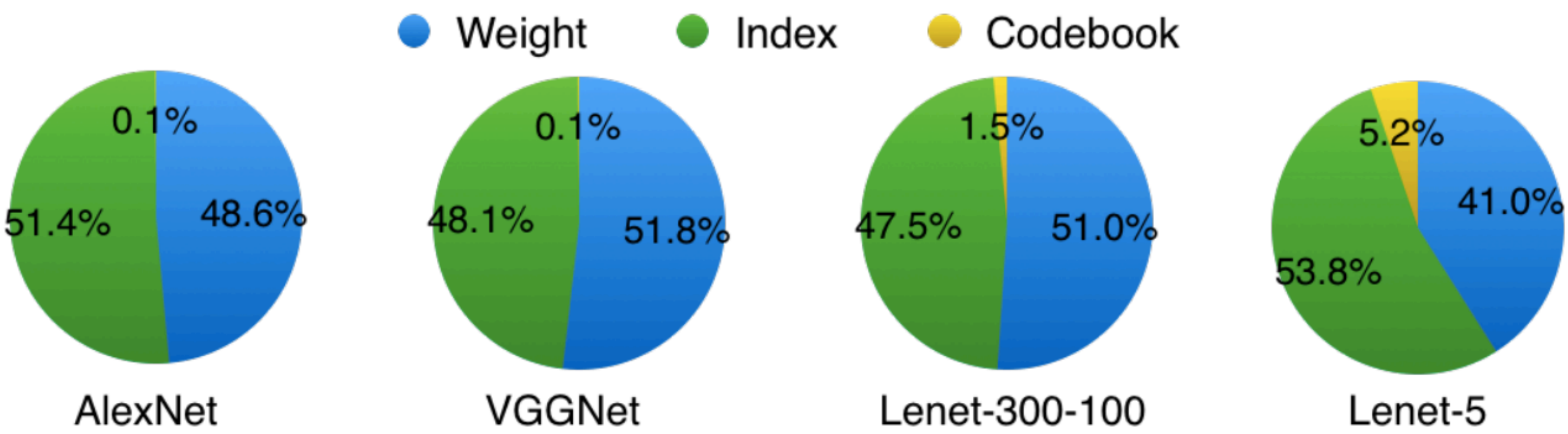
- Tested on AlexNet → **2× overall BW Reduction**

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]

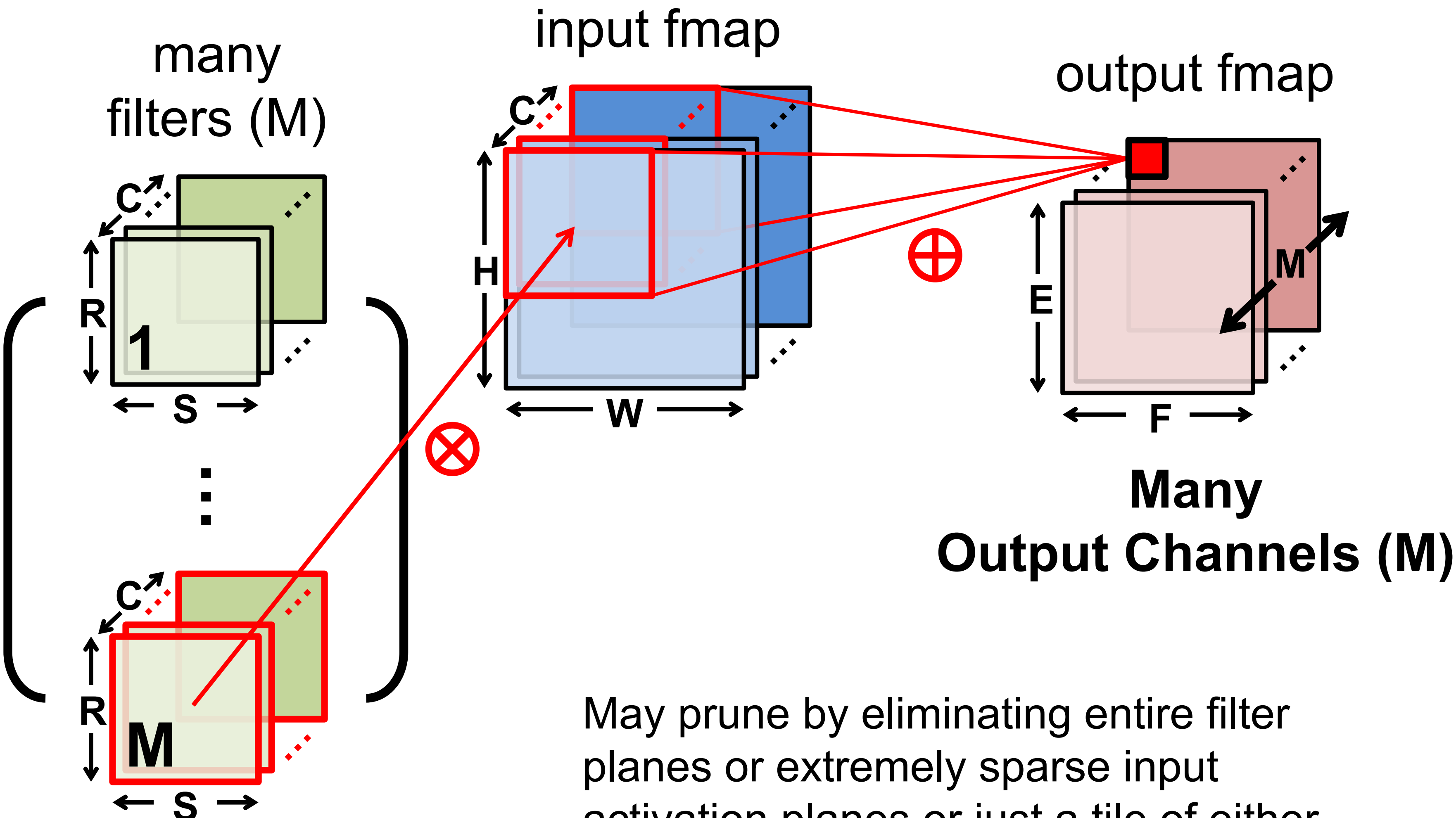
Compression Overhead

*Index (non-zero position info – e.g., **IA** and **JA** for CSR) accounts for approximately half of storage for fine grained pruning*



[Han et al., ICLR 2016]

Coarse-Grained Pruning

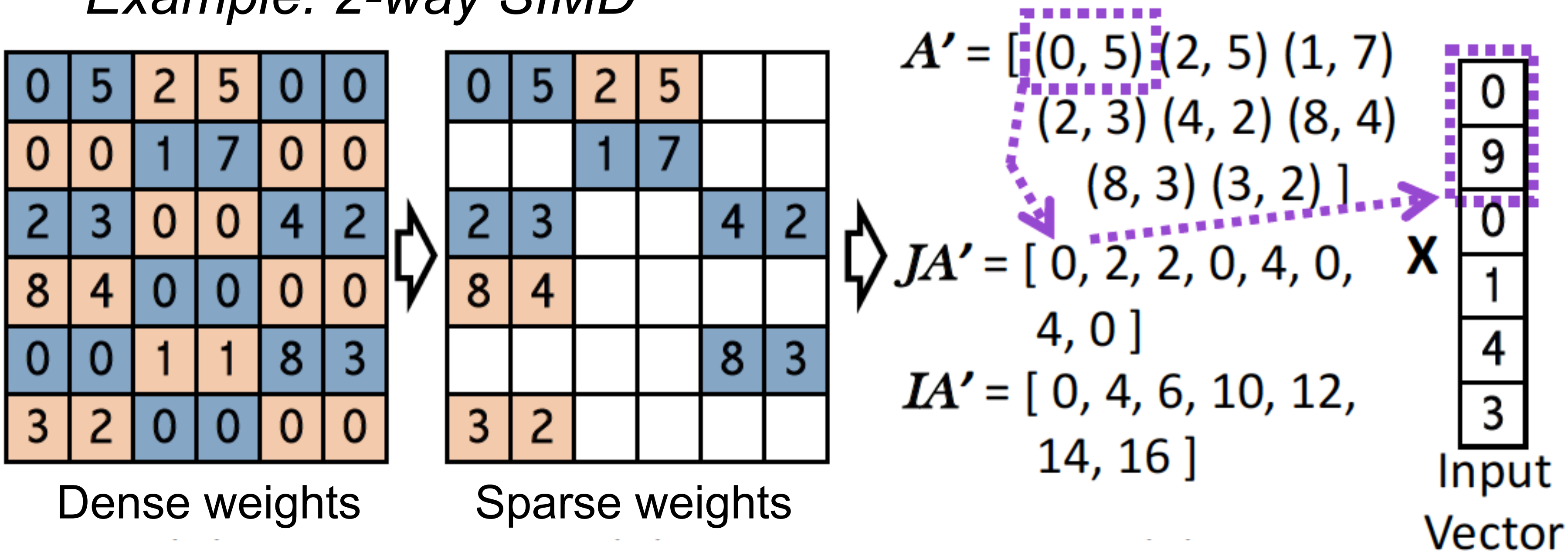


May prune by eliminating entire filter planes or extremely sparse input activation planes or just a tile of either.

Structured/Coarse-Grained Pruning

- **Scalpel**
  - Prune to match the underlying data-parallel hardware organization for speed up

Example: 2-way SIMD



[Yu et al., ISCA 2017]

Exploit Redundant Weights

- Preprocessing to reorder weights (ok since weights are known)
- Perform addition before multiplication to reduce number of multiplies and reads of weights
- **Example:** Input = [1 2 3] and filter [A B A]

Typical processing: Output = A*1+B*2+A*3

3 multiplies and 3 weight reads

If reorder as [A A B]: Output = A*(1+3)+B*1

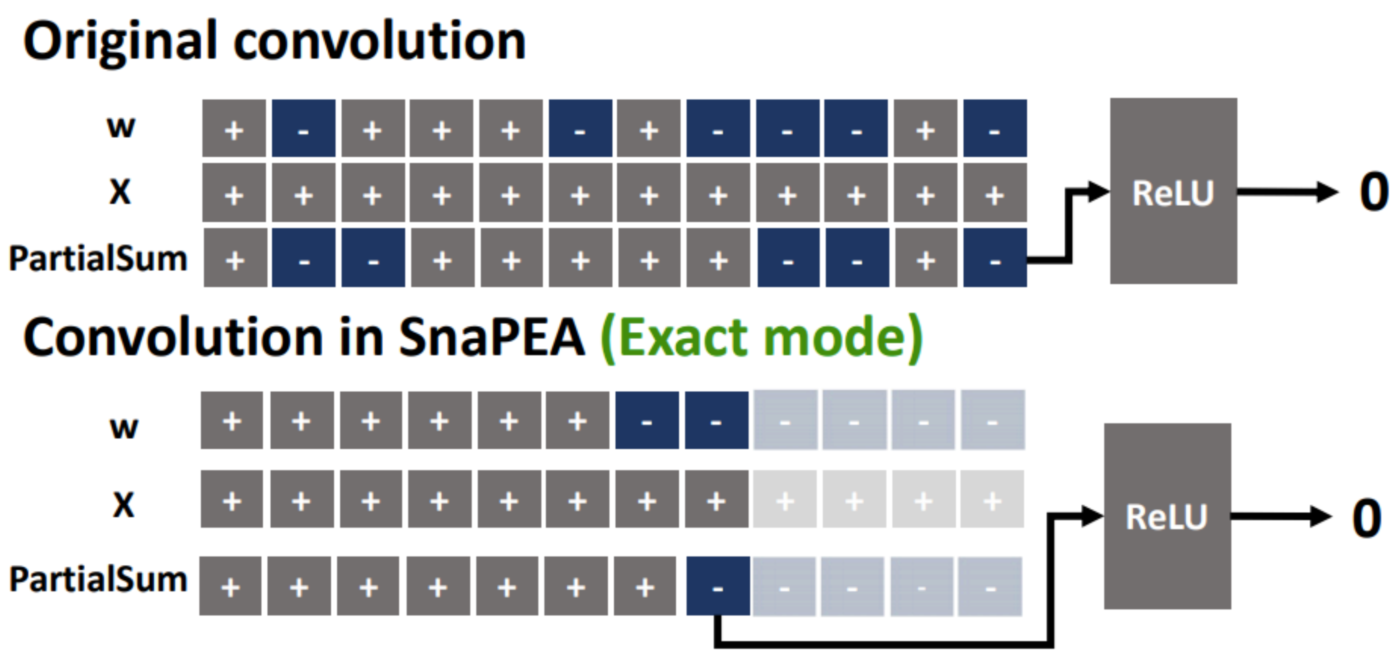
2 multiplies and 2 weight reads

Note: Bitwidth of multiplication may need to increase

[UCNN, ISCA 2018]

Exploit ReLU

- Reduce number operations when if resulting activation will be negative as ReLU will set to zero
- Need to either perform preprocess (sort weights) or minimize prediction overhead and error



[PredictiveNet, ISCAS 2017], [SnaPEA, ISCA 2018], [Song et al., ISCA 2018]

Compact Network Architectures

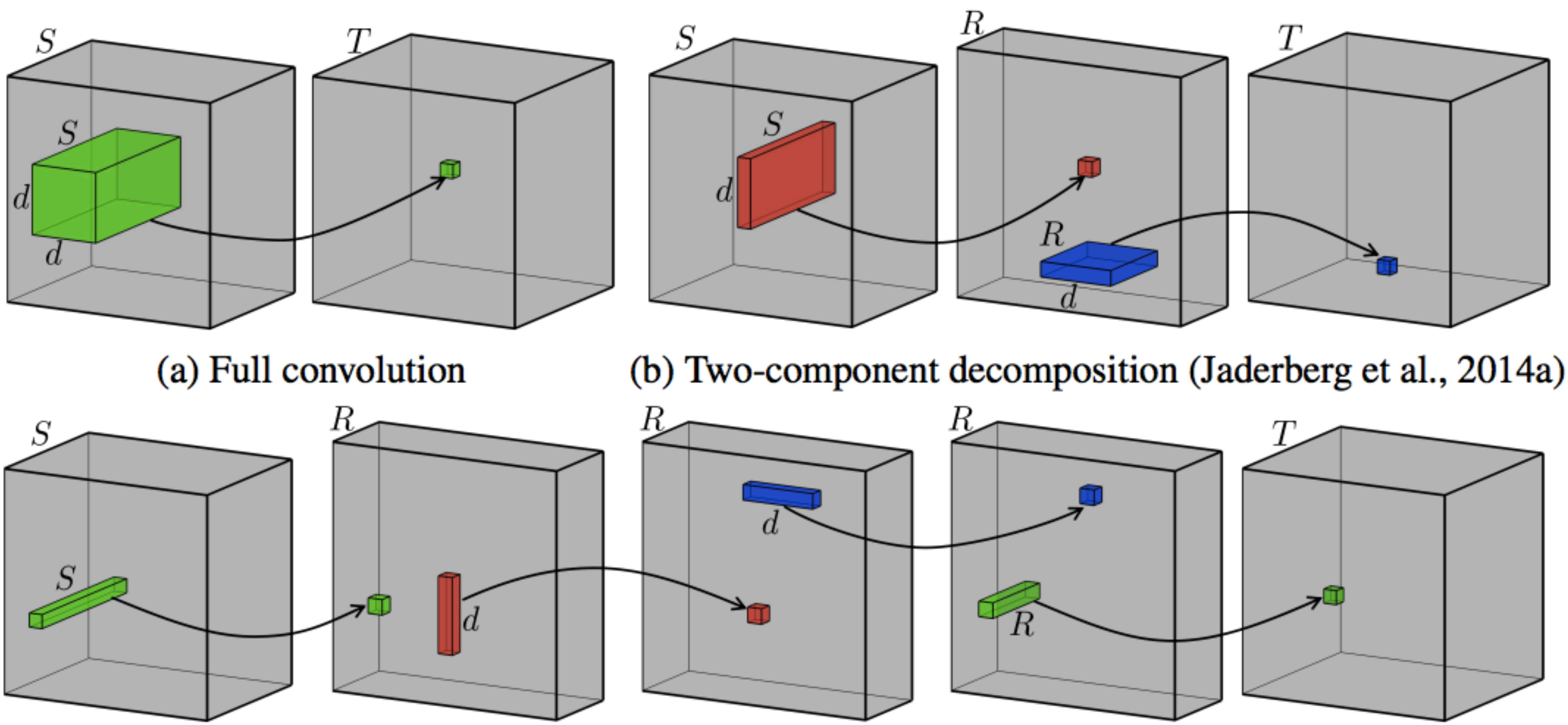
- Break large convolutional layers into a series of smaller convolutional layers
  - Fewer weights, but same effective receptive field

- **Before Training:** Network Architecture Design *(already discussed this morning; e.g., MobileNet)*

- **After Training**: Decompose Trained Filters

Decompose Trained Filters

After training, perform **low-rank approximation by applying tensor decomposition** to weight kernel; then **fine-tune** weights for accuracy



(a) Full convolution

(b) Two-component decomposition (Jaderberg et al., 2014a)

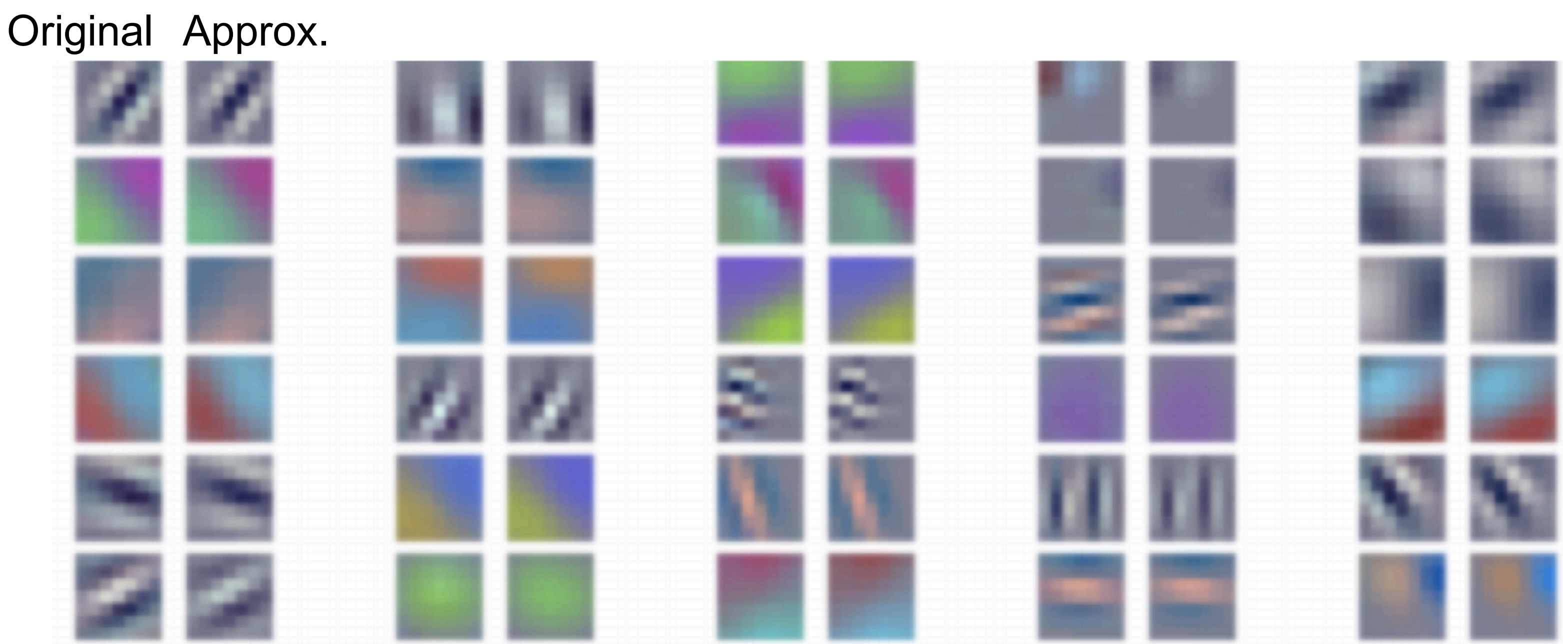
(c) CP-decomposition

R = canonical rank

[Lebedev et al., ICLR 2015]

Decompose Trained Filters

Visualization of Filters

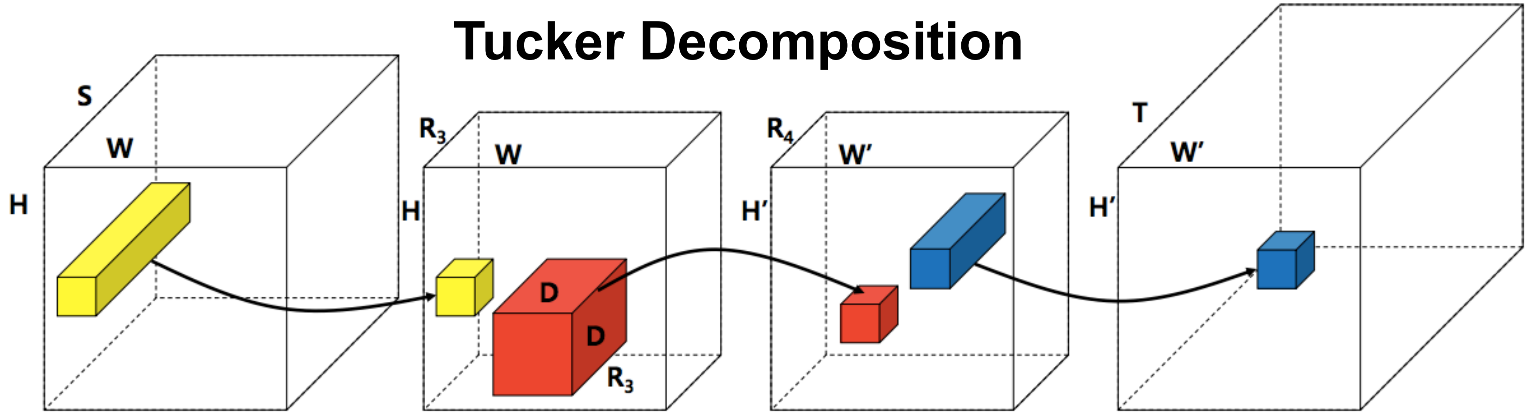


- **Speed up by 1.6 – 2.7x on CPU/GPU for CONV1, CONV2 layers**
- **Reduce size by 5 - 13x for FC layer**
- **< 1% drop in accuracy**

[Denton et al., NeurIPS 2014]

Decompose Trained Filters on Phone

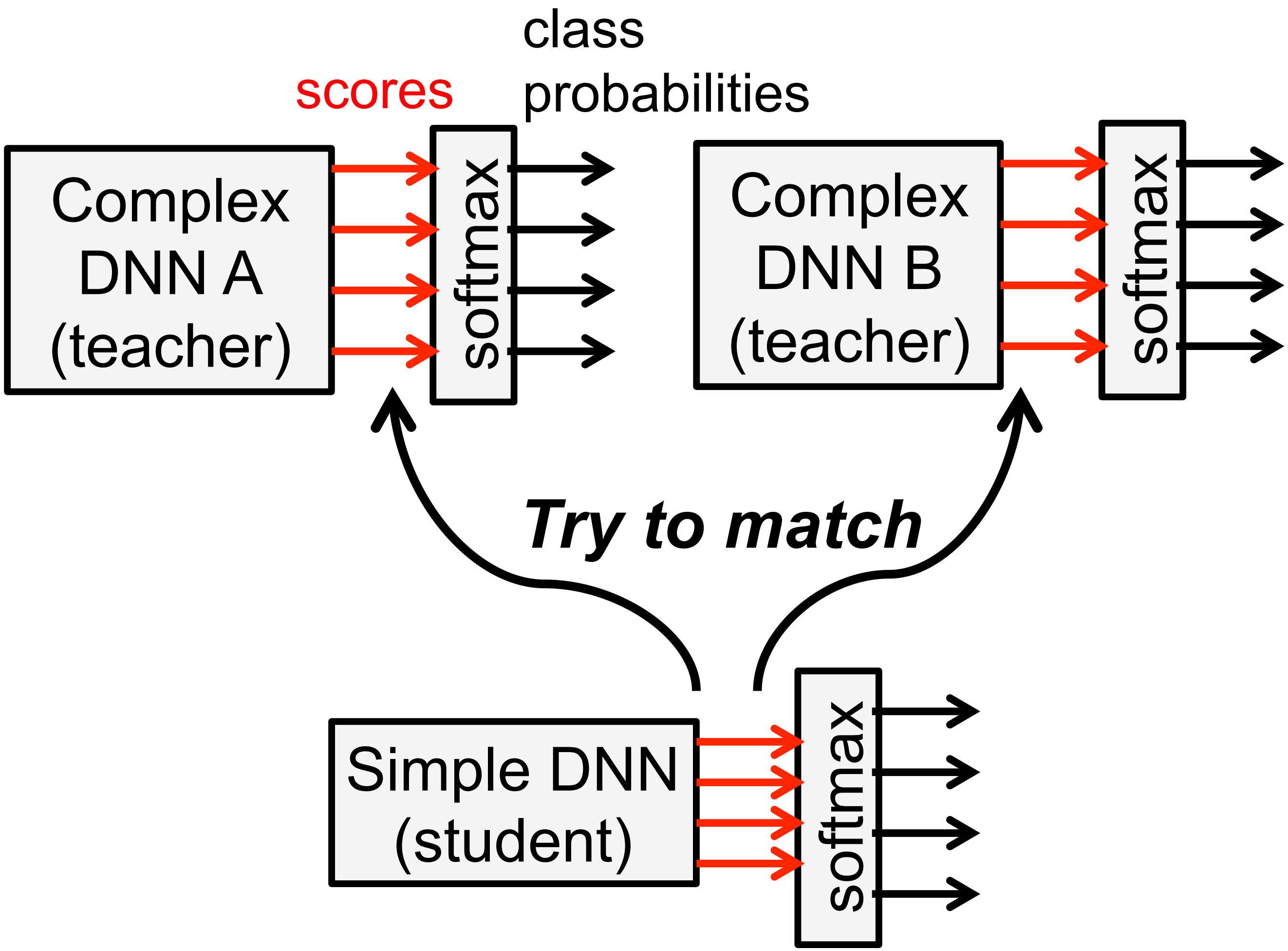
Tucker Decomposition



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]

Knowledge Distillation



[Bucilu et al., KDD 2006],[Hinton et al., arXiv 2015]