Efficient Image Processing with Deep Neural Networks

Vivienne Sze, Tien-Ju Yang

Massachusetts Institute of Technology

In collaboration with

Yu-Hsin Chen, Joel Emer

Contact Info
email: sze@mit.edu
website: www.rle.mit.edu/eems

Follow @eems_mit

Contributors

Vivienne Sze

Professor

MIT

Tien-Ju Yang

PhD Candidate

MIT

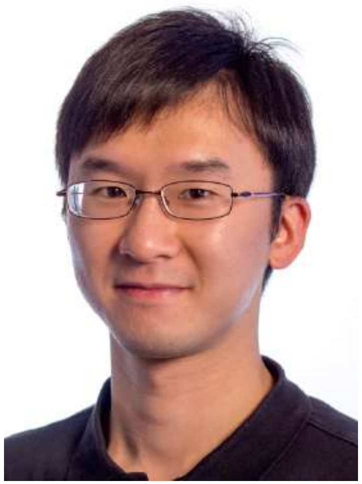
Joel Emer

Senior Distinguished Research Scientist

NVIDIA

Professor

MIT



Yu-Hsin Chen

PhD Graduate

MIT

Outline of Tutorial

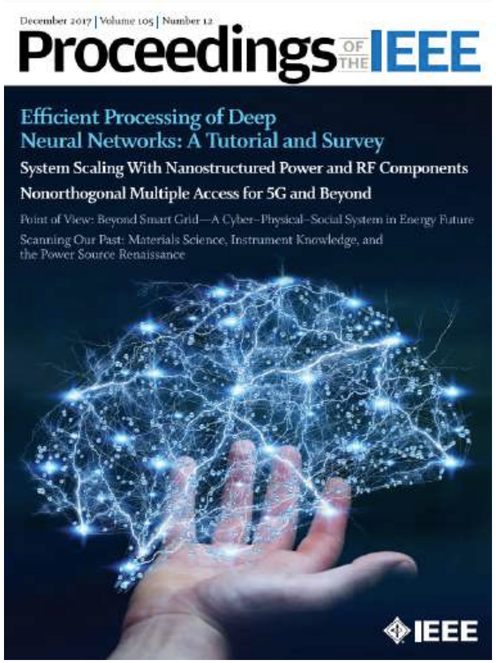
- Brief overview of Deep Neural Networks (DNN)
- **Part 1: Hardware Platforms for DNNs** (e.g., CPU, GPU, FPGA, ASIC) and **metrics for evaluating the efficiency of DNNs**
- **Part 2: Co-design algorithms and hardware** for efficient DNNs (e.g., precision, sparsity, network architecture design, network architecture search, designing networks with hardware in the loop)
- **Part 3: Application of efficient DNNs** on a wide range of image processing and computer vision tasks (e.g., image classification, depth estimation, image segmentation, super-resolution)

Additional Resources

Overview Paper

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!



More info about **Tutorial on DNN Architectures**
http://eyeriss.mit.edu/tutorial.html

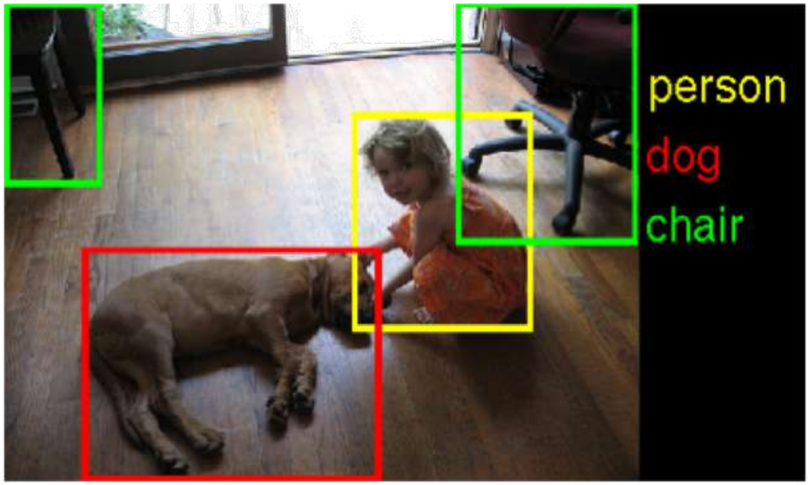
MIT Professional Education Course on
"Designing Efficient Deep Learning Systems"
http://professional-education.mit.edu/deeplearning

For updates Follow @eems_mit

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

Example Applications of Deep Learning

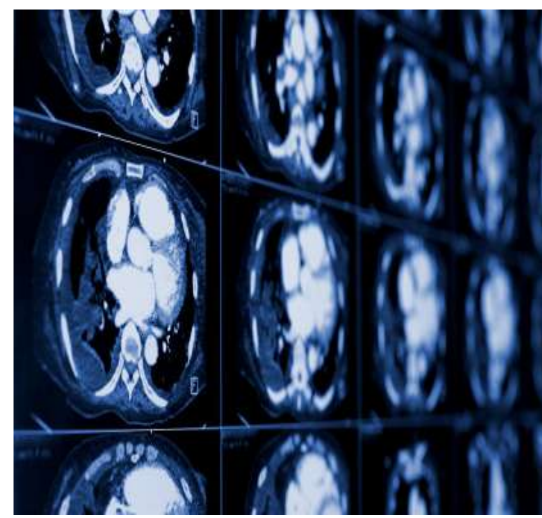
Computer Vision



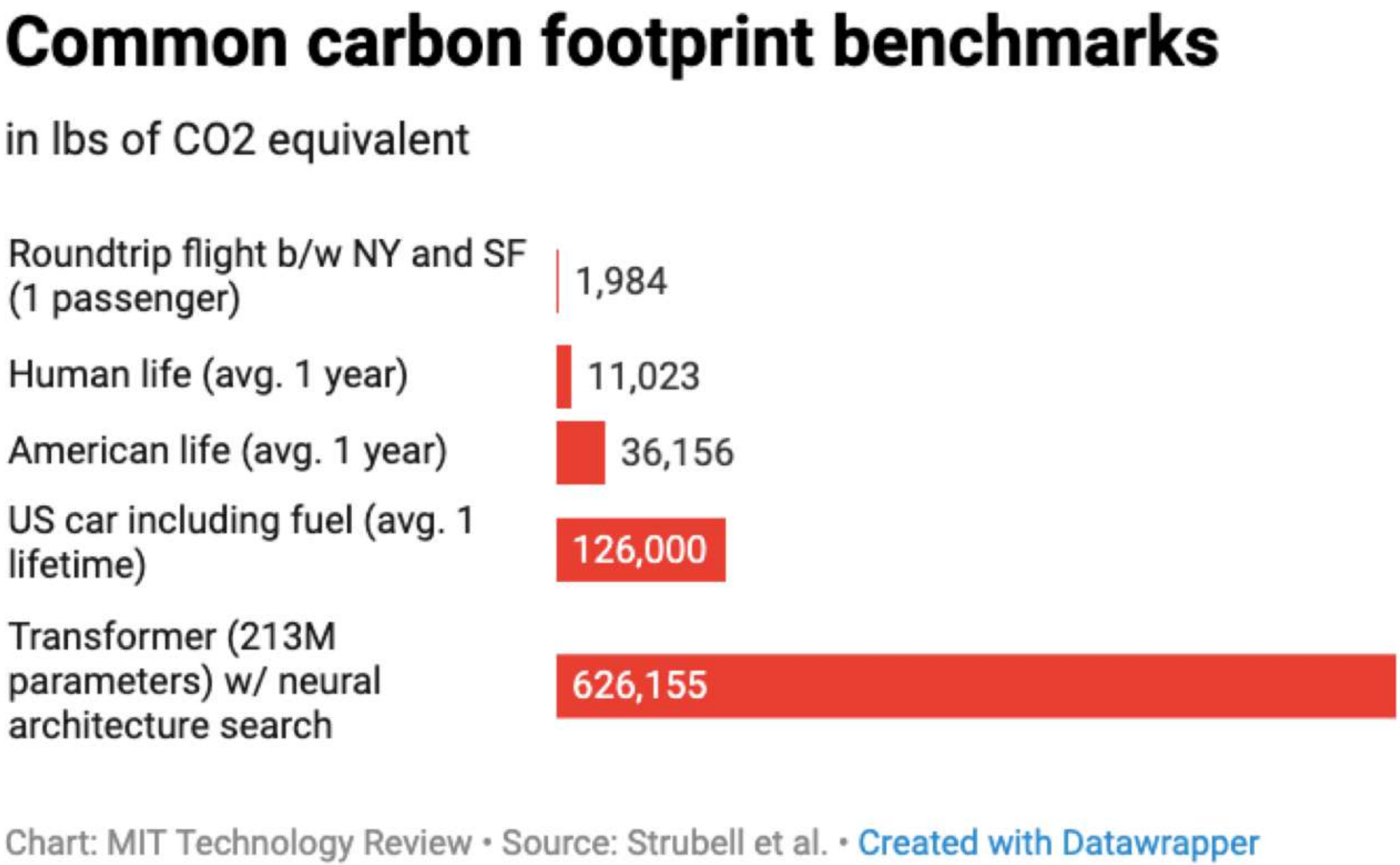
Speech Recognition

Game Play

Medical

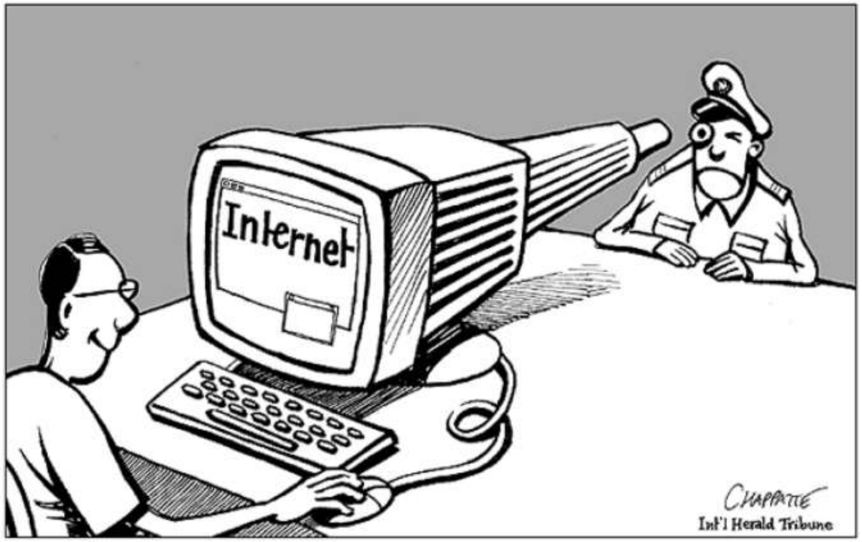


Compute Demands for Deep Learning



Processing at "Edge" instead of the "Cloud"

Privacy



Communication

Image source: www.theregister.co.uk

Latency

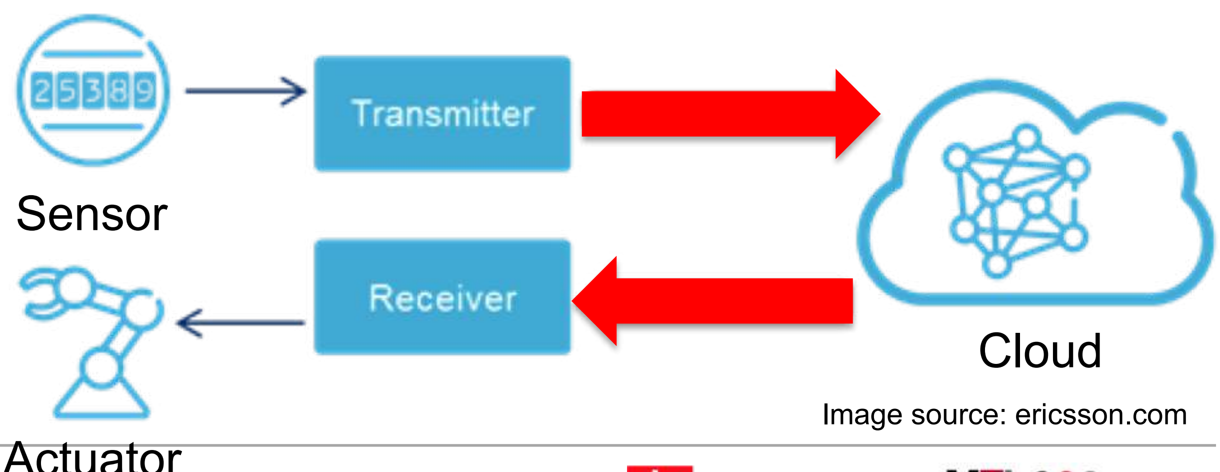


Image source: ericsson.com

Deep Learning for Self-Driving Cars

JACK STEWART TRANSPORTATION 02.06.18 08:00 AM

SELF-DRIVING CARS USE CRAZY AMOUNTS OF POWER, AND IT'S BECOMING A PROBLEM

Shelley, a self-driving Audi TT developed by Stanford University, uses the brains in the trunk to speed around a racetrack autonomously.

NIKKI KAHN/THE WASHINGTON POST/GETTY IMAGES

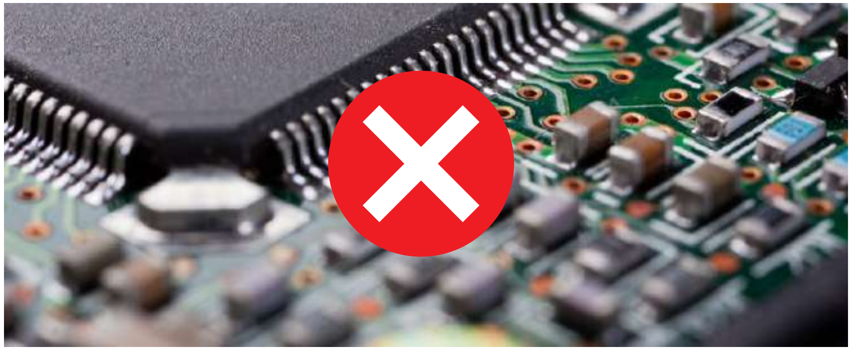
WIRED

(Feb 2018)

Cameras and radar generate ~6 gigabytes of data every 30 seconds.

Prototypes use around 2,500 Watts. Generates wasted heat and some prototypes need water-cooling!

Existing Processors Consume Too Much Power

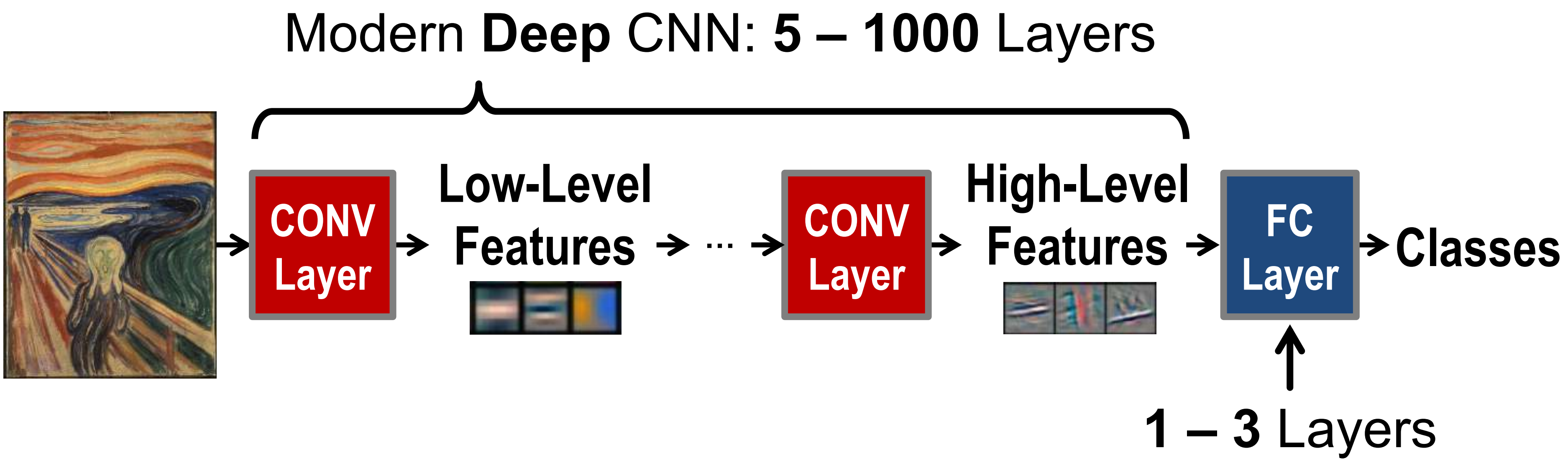


< 1 Watt

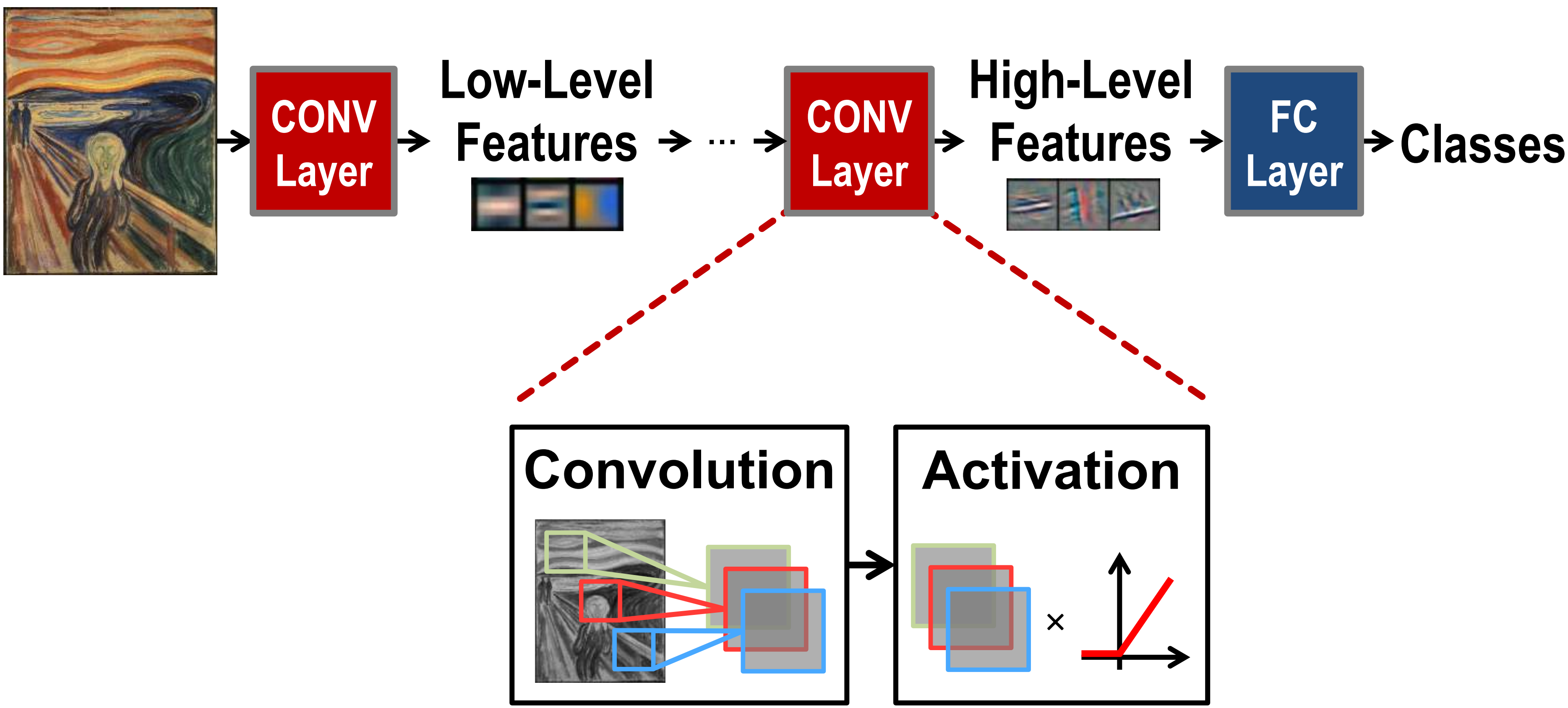
> 10 Watts

Overview of Deep Neural Networks

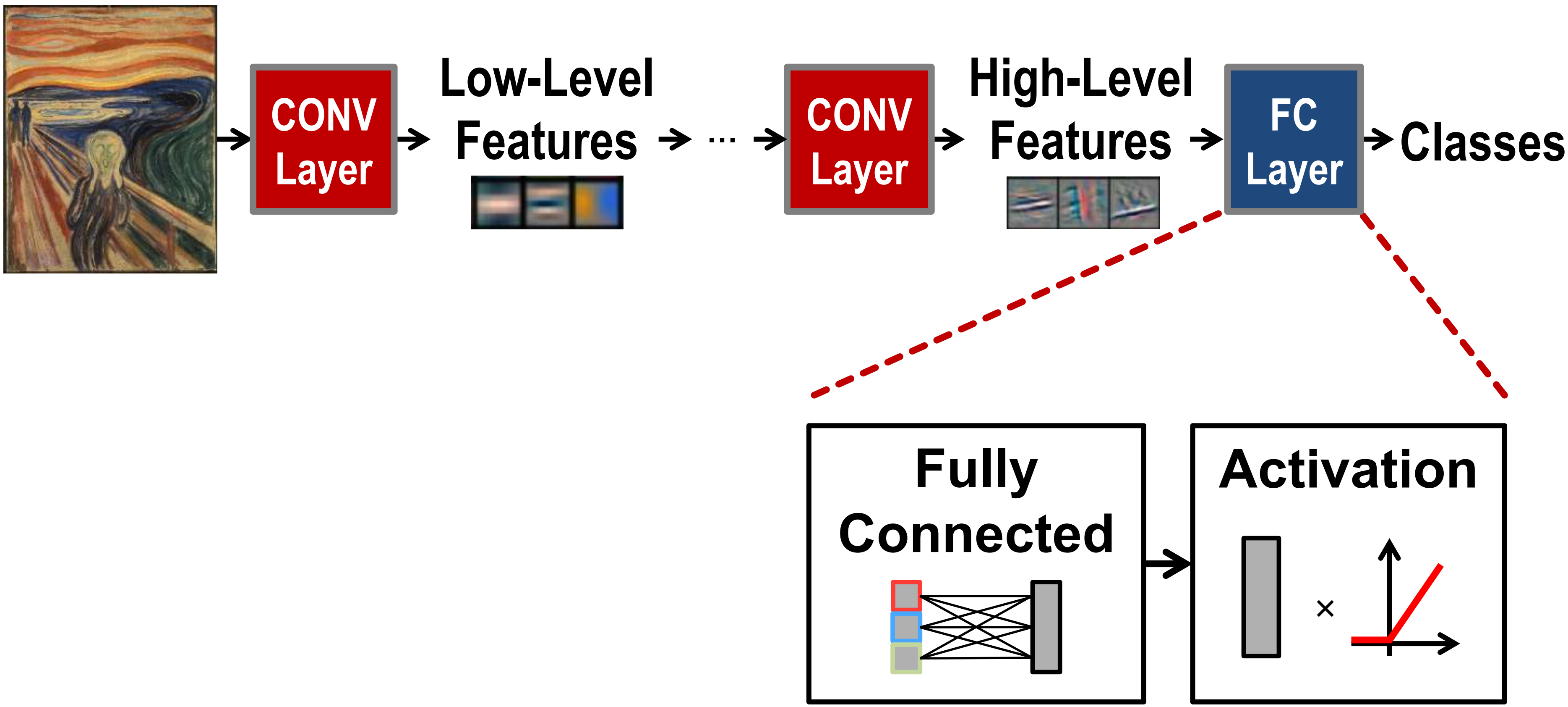
Deep Convolutional Neural Networks



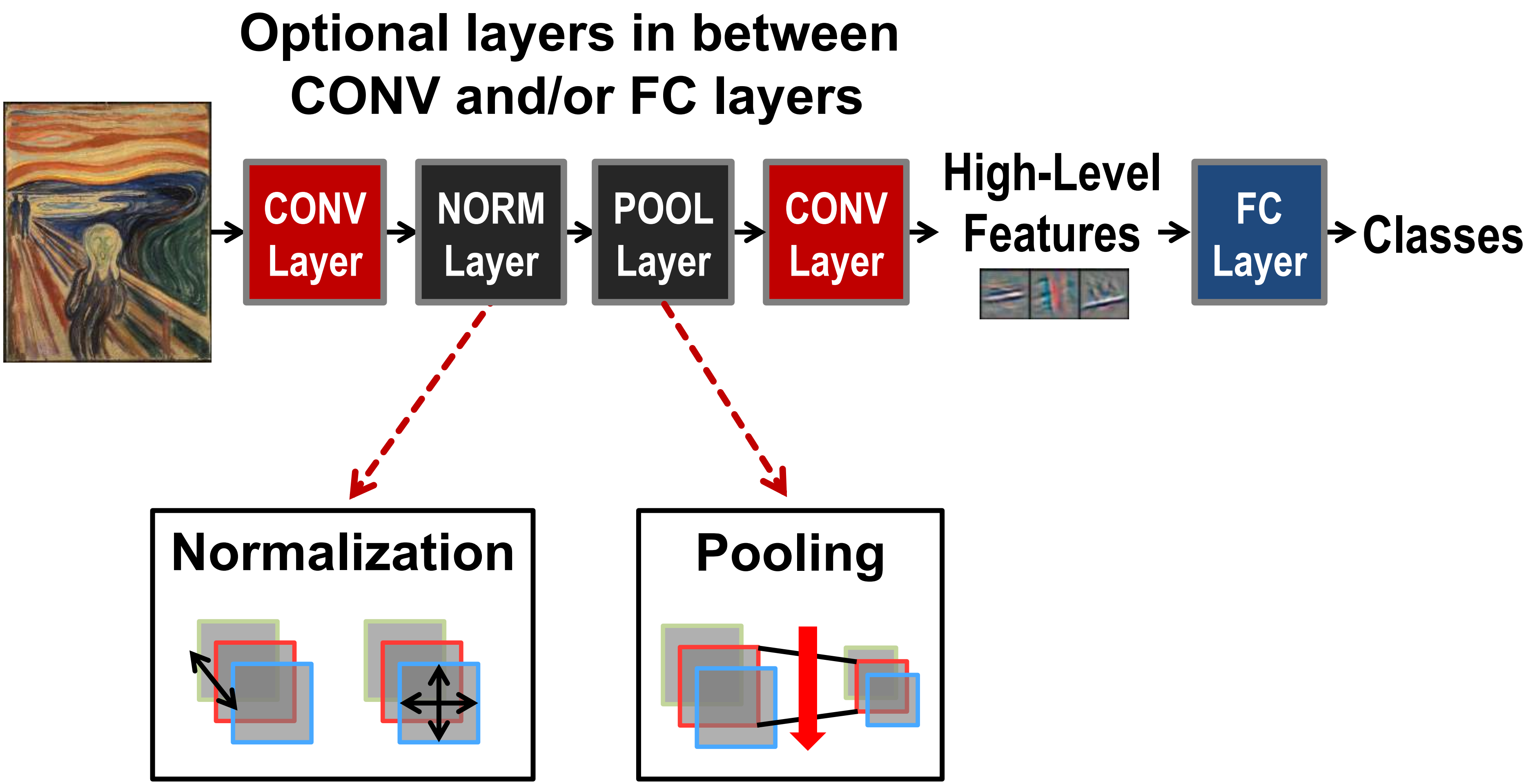
Deep Convolutional Neural Networks



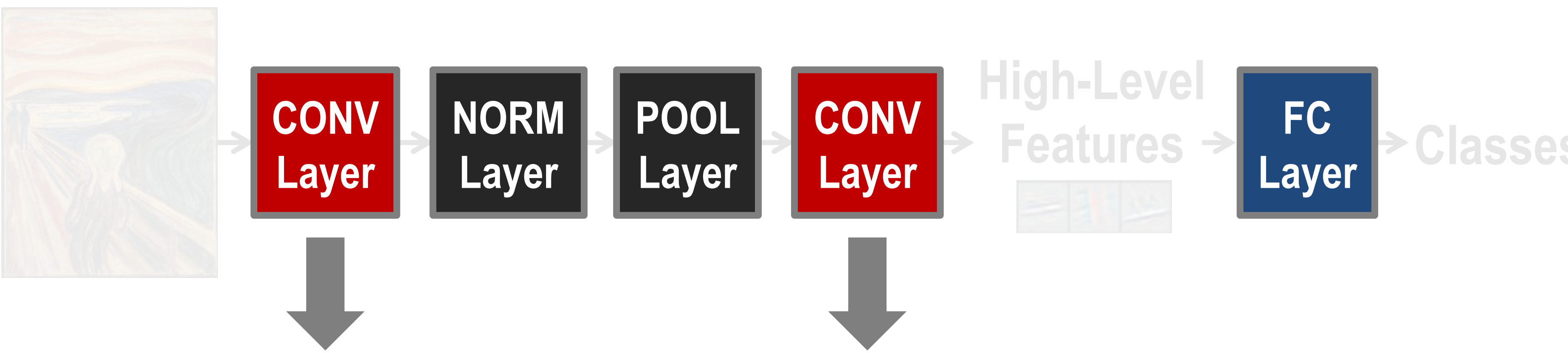
Deep Convolutional Neural Networks

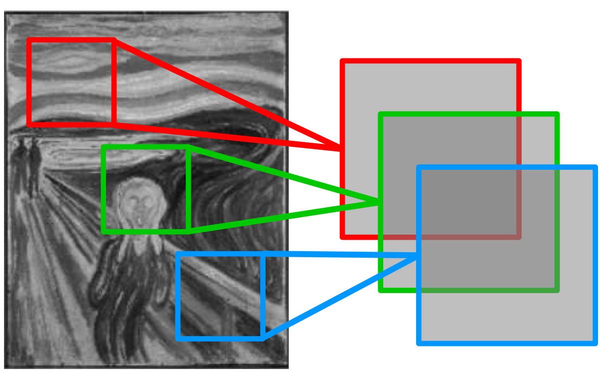


Deep Convolutional Neural Networks



Deep Convolutional Neural Networks



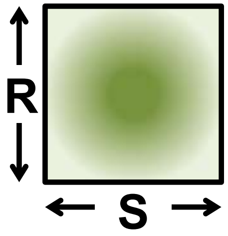


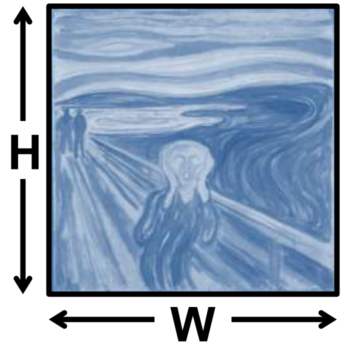
Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

Convolution (CONV) Layer

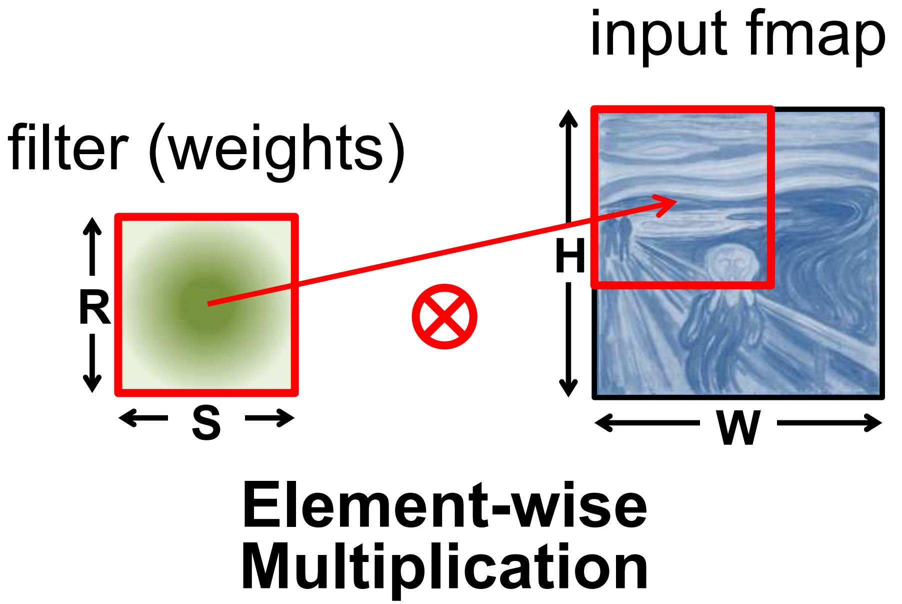
a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)

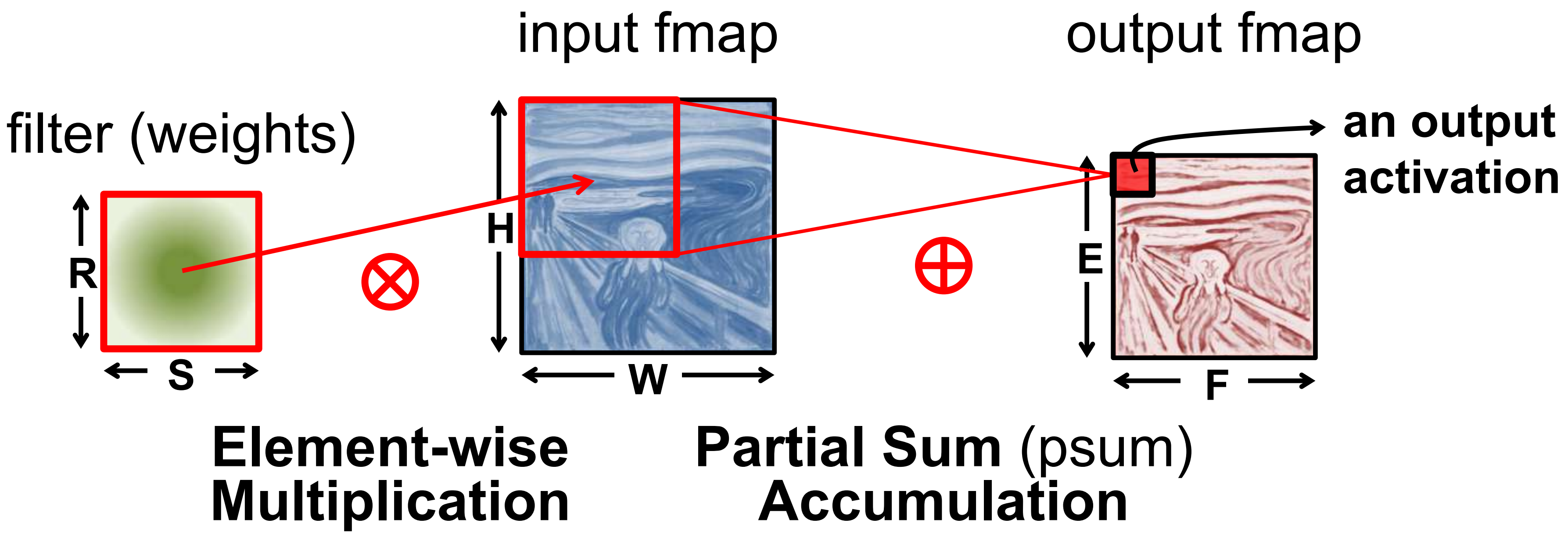




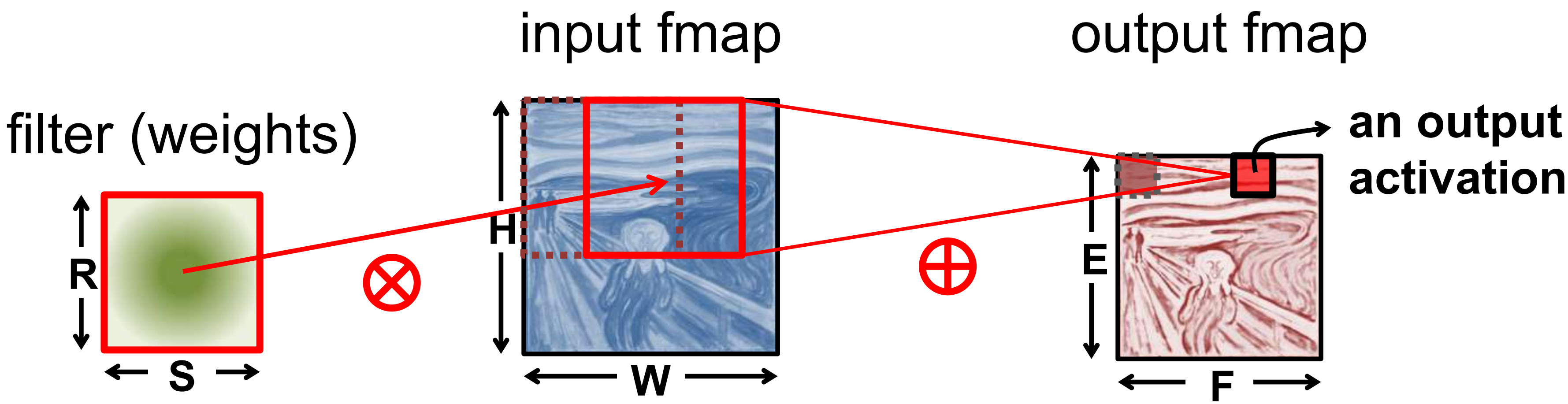
Convolution (CONV) Layer



Convolution (CONV) Layer

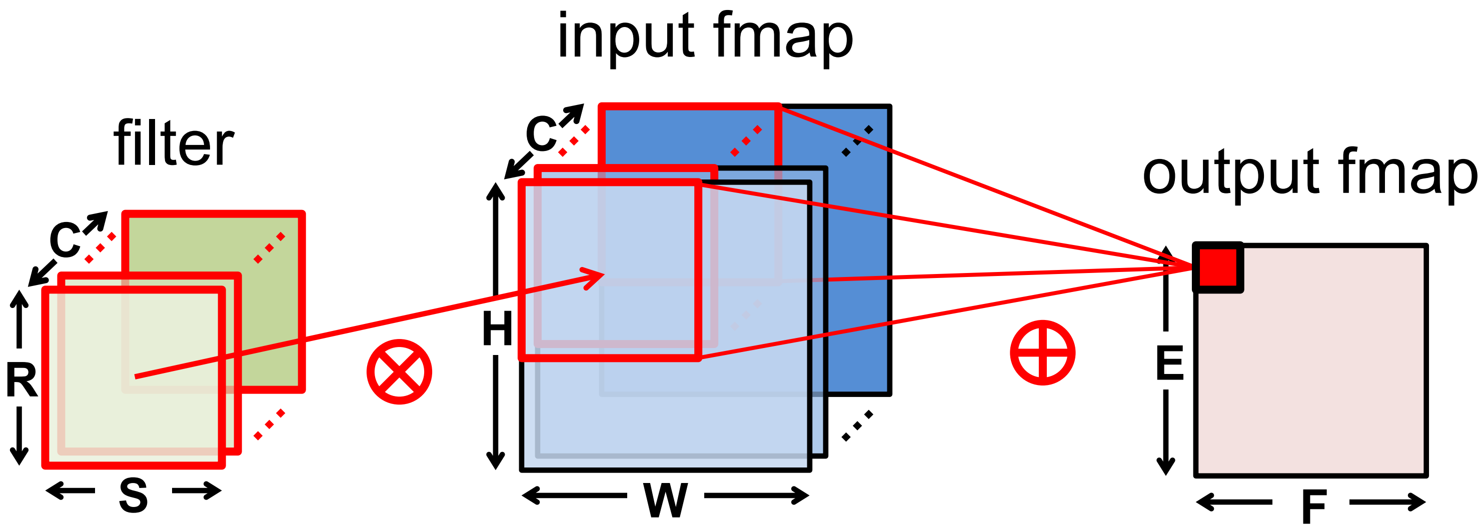


Convolution (CONV) Layer



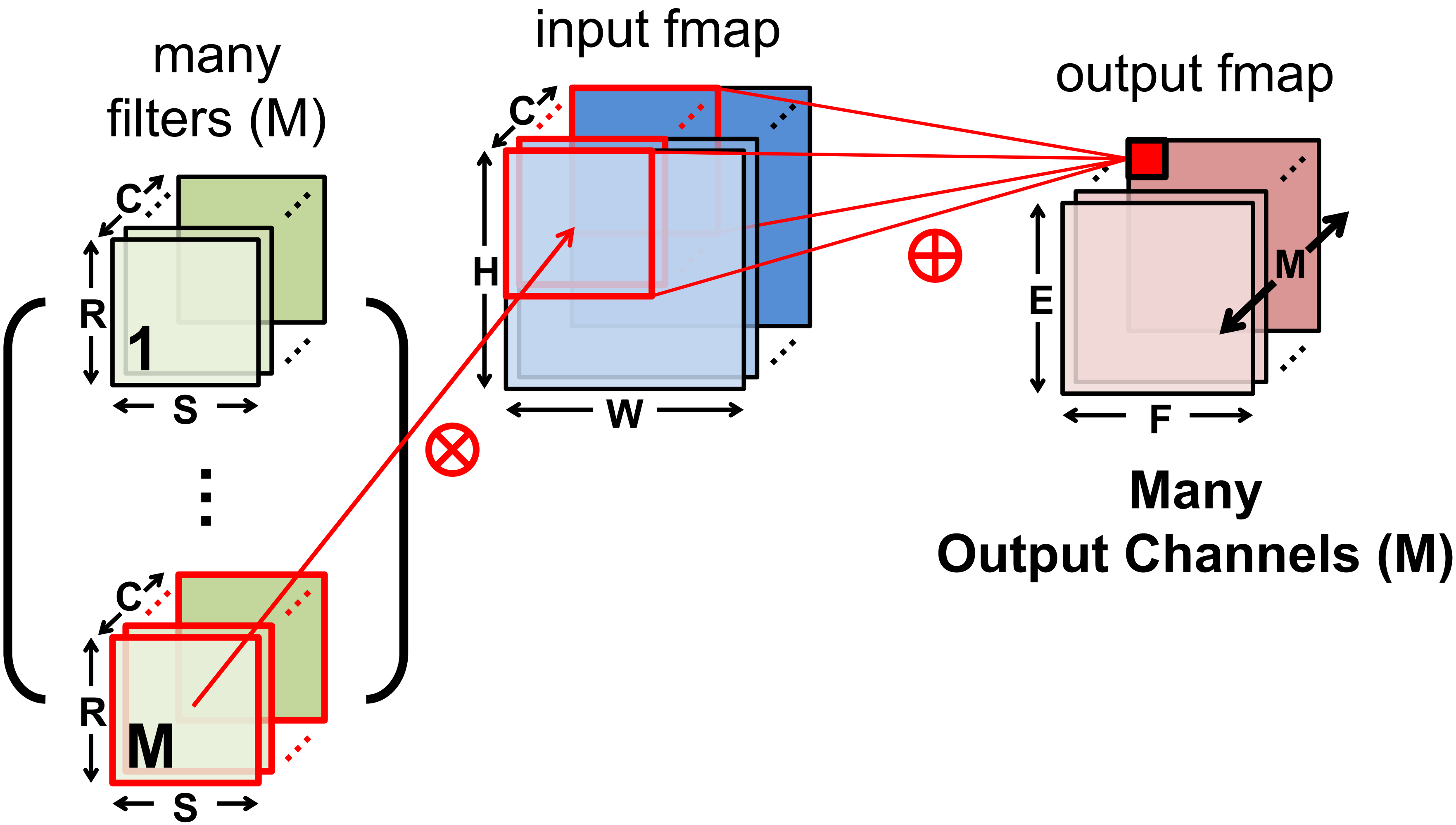
Sliding Window Processing

Convolution (CONV) Layer

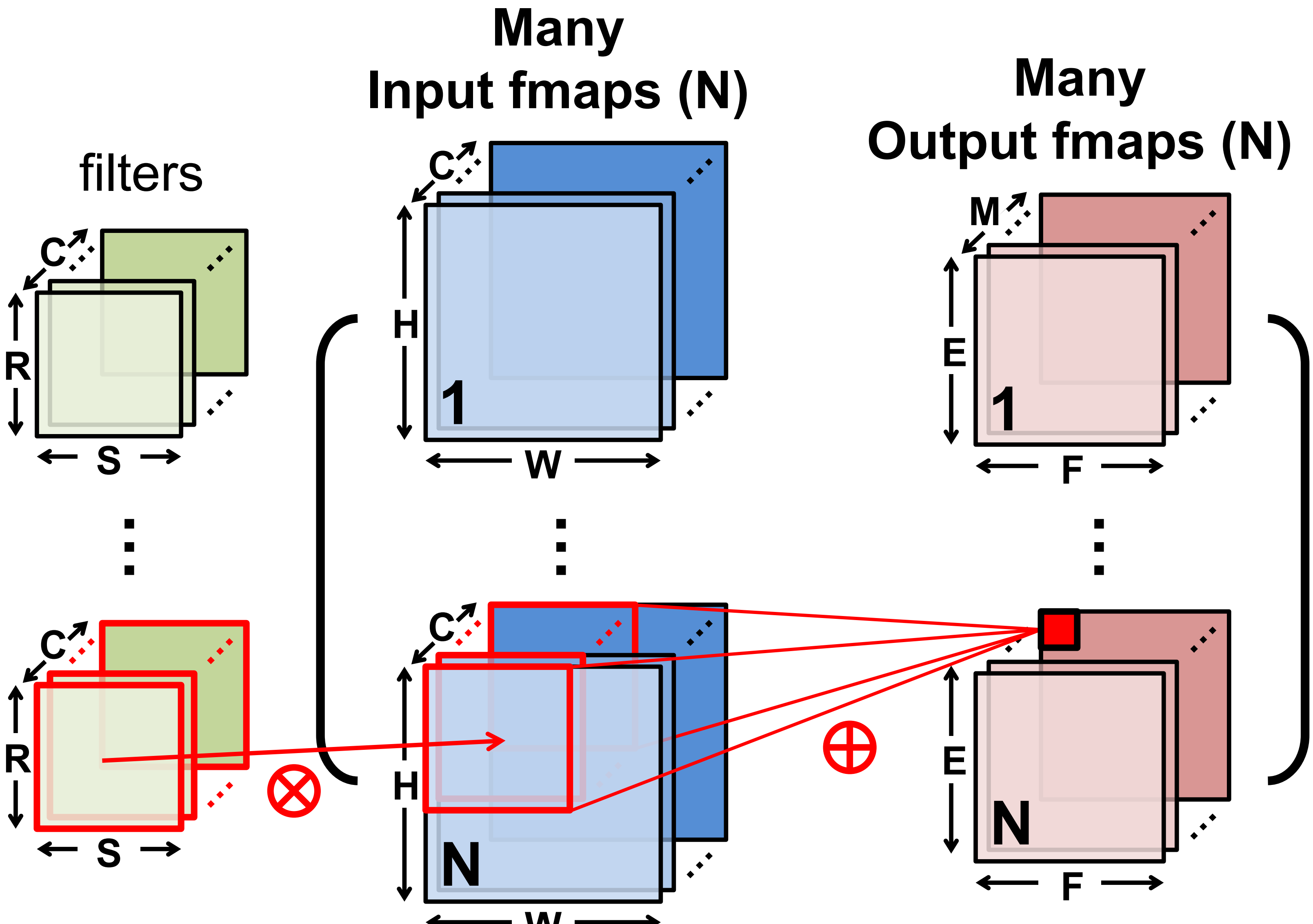


Many Input Channels (C)

Convolution (CONV) Layer



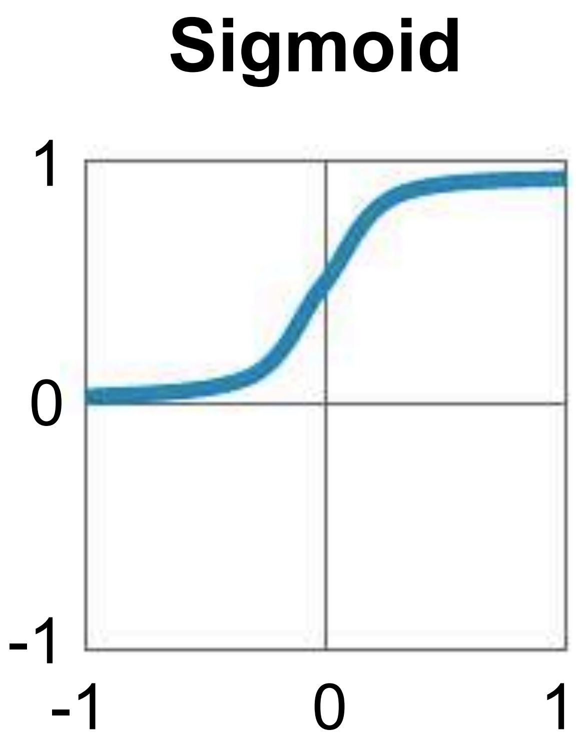
Convolution (CONV) Layer



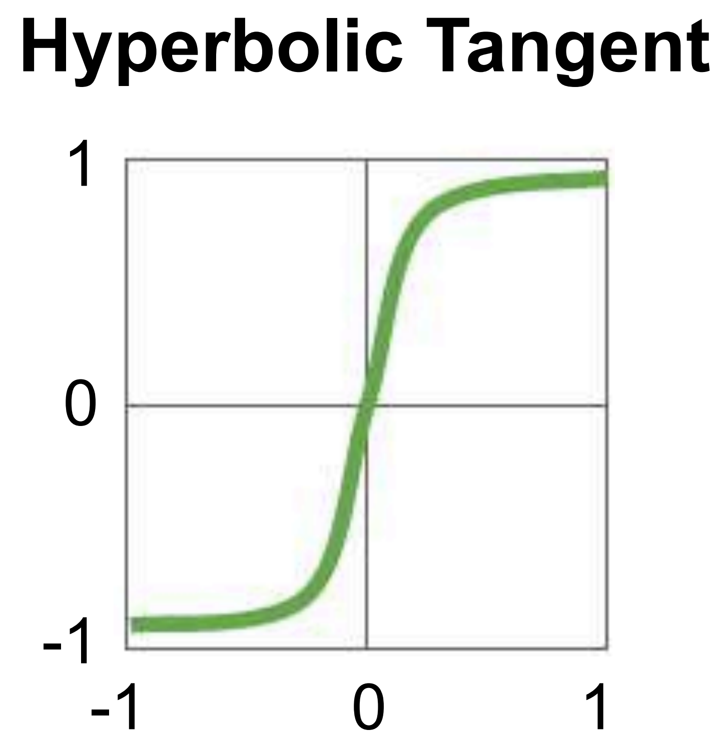
CNN Decoder Ring

- N – Number of input fmaps/output fmaps (batch size)
- C – Number of 2-D input fmaps /filters (channels)
- H – Height of input fmap (activations)
- W – Width of input fmap (activations)
- 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)

Traditional Activation Functions



$y=1/(1+e^{-x})$

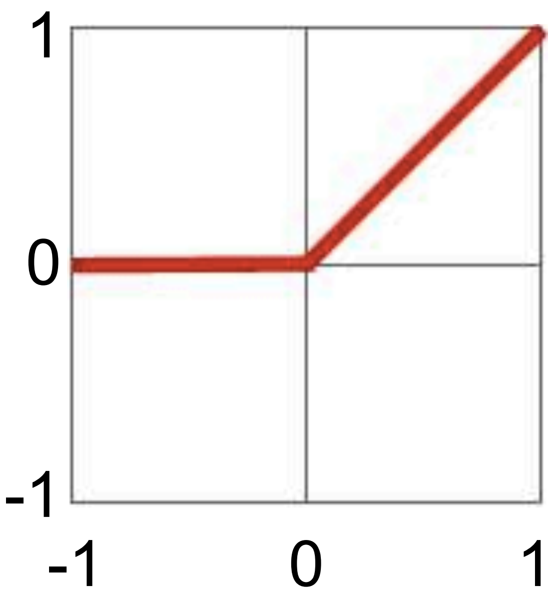


$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$

Image Source: Caffe Tutorial

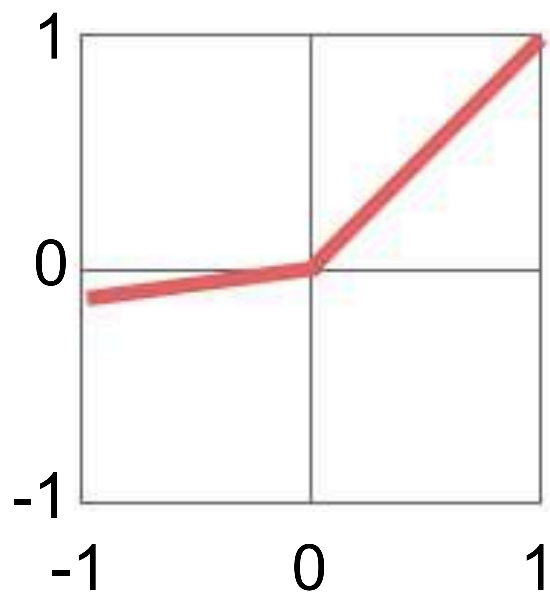
Modern Activation Functions

Rectified Linear Unit (ReLU)



$$y=\max(0,x)$$

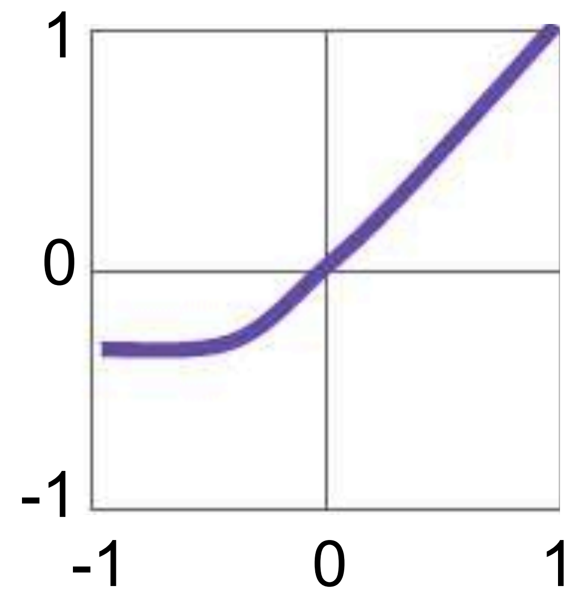
Leaky ReLU



$$y=\max(\alpha x,x)$$

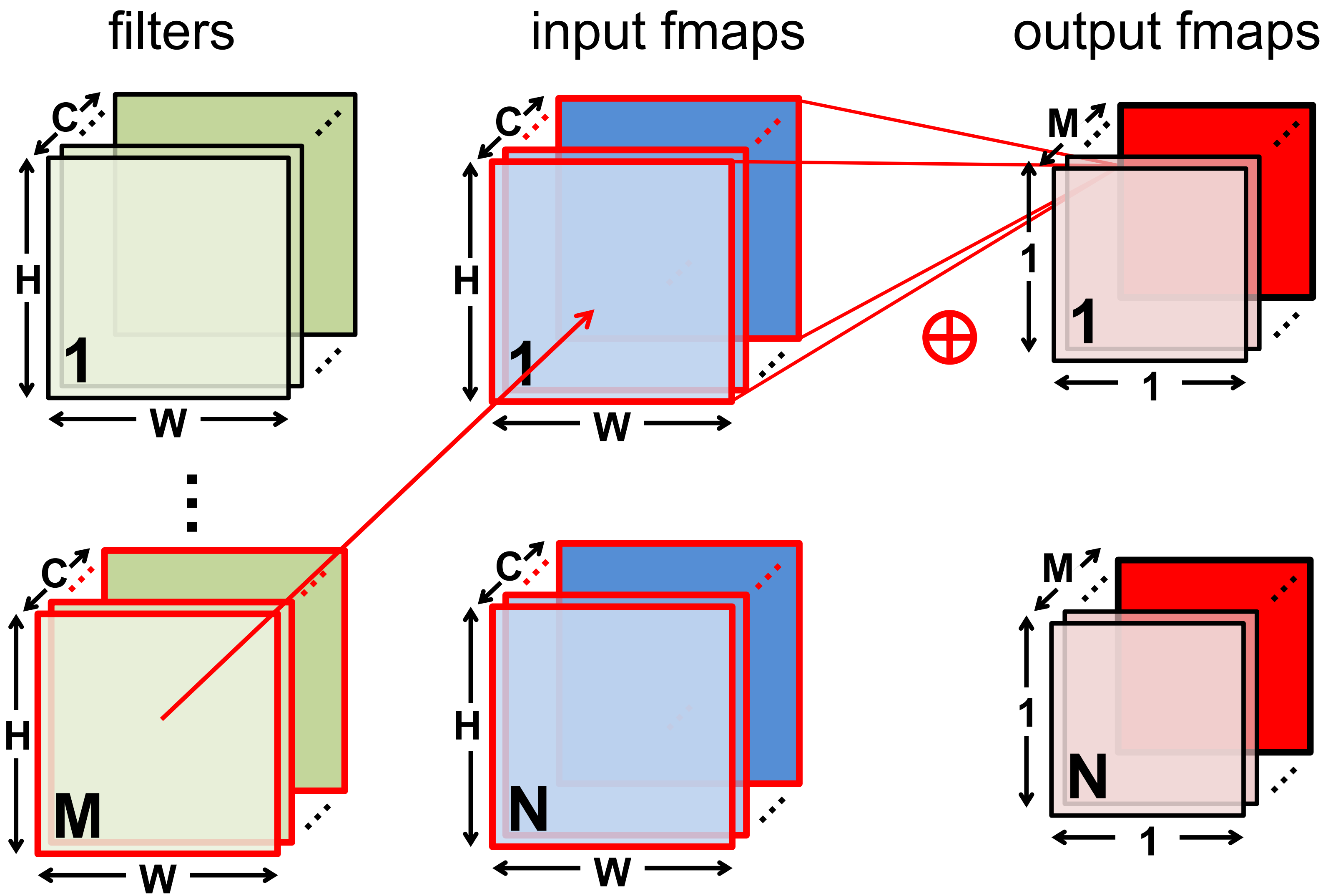
α = small const. (e.g. 0.1)

Exponential LU



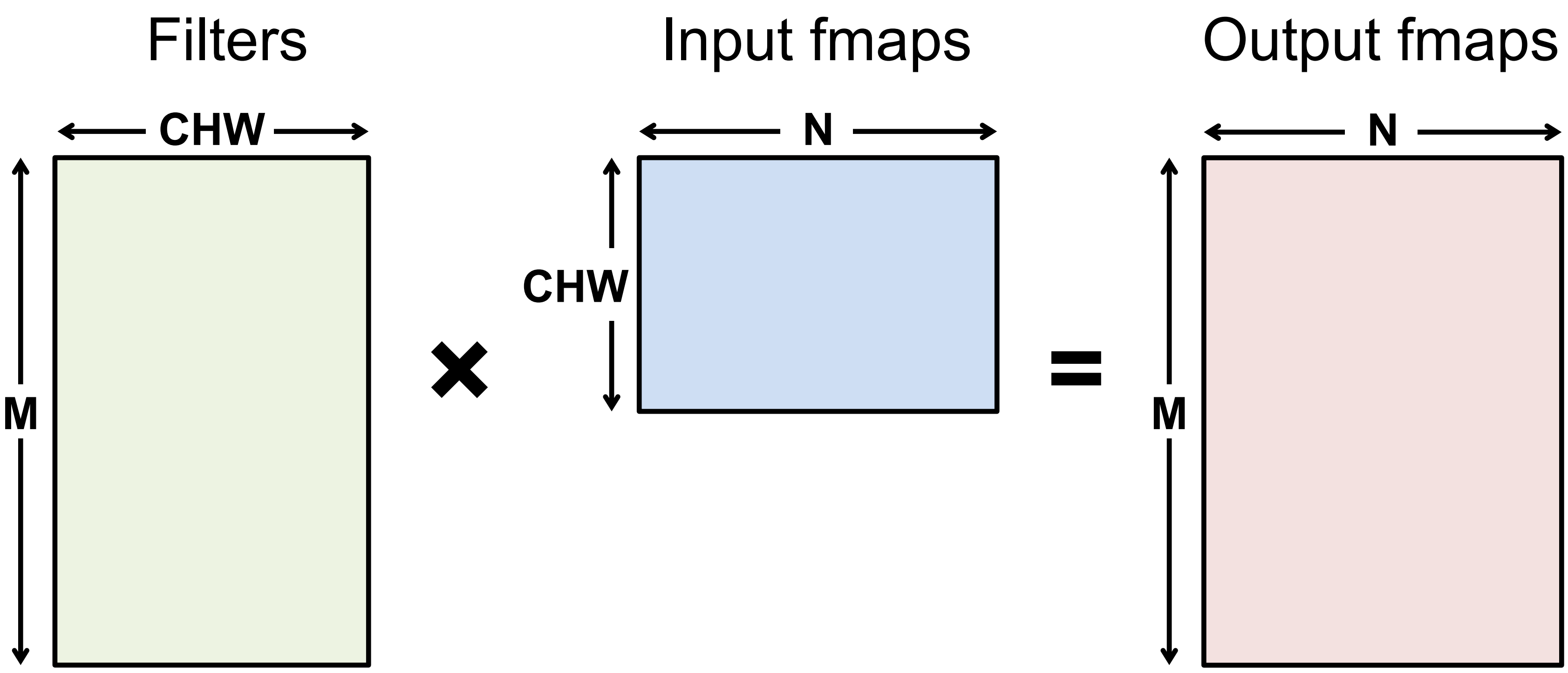
$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$$

FC Layer – from CONV Layer POV



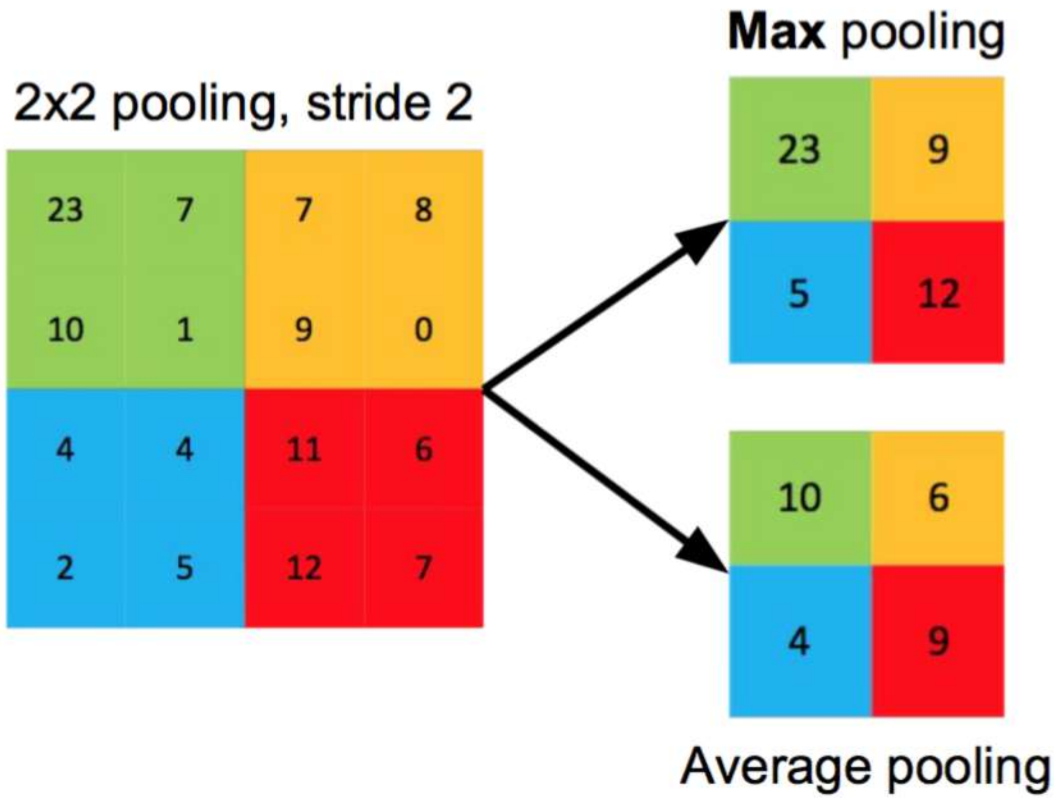
Fully-Connected (FC) Layer

- Height and width of output fmaps are 1 (E = F = 1)
- Filters as large as input fmaps (R = H, S = W)
- Implementation: **Matrix Multiplication**



Pooling (POOL) Layer

- Reduce resolution of each channel independently
- Overlapping or non-overlapping → depending on stride

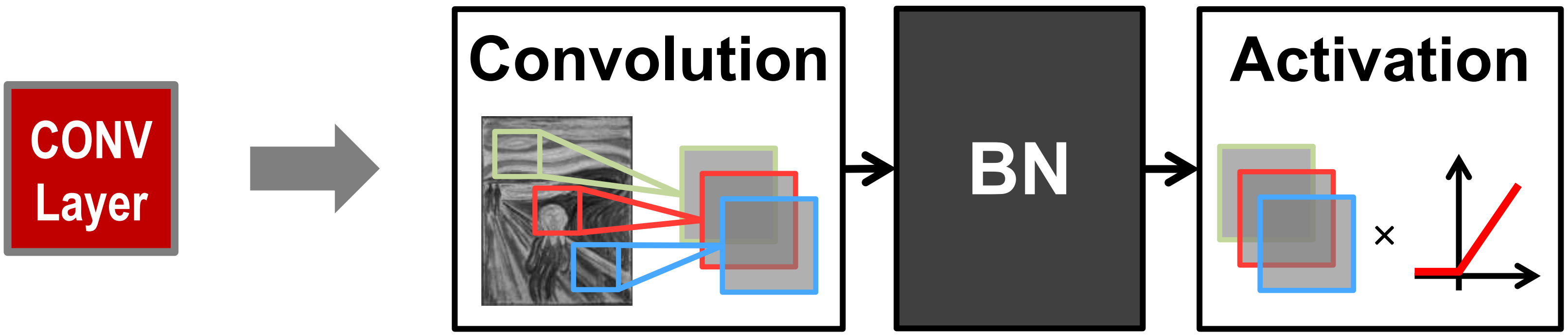


Increases translation-invariance and noise-resilience

Image Source: Caffe Tutorial

Normalization (NORM) Layer

- **Batch Normalization (BN)**
  - Normalize activations towards mean=0 and std. dev.=1 based on the statistics of the training dataset
  - put **in between** **CONV/FC** and **Activation function**



Believed to be key to getting high accuracy and faster training on very deep neural networks.

[Ioffe et al., ICML 2015]

BN Layer Implementation

- The normalized value is further scaled and shifted, the parameters of which are learned from training

data mean

learned scale factor

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \gamma + \beta$$

learned shift factor

data std. dev.

small const. to avoid numerical problems

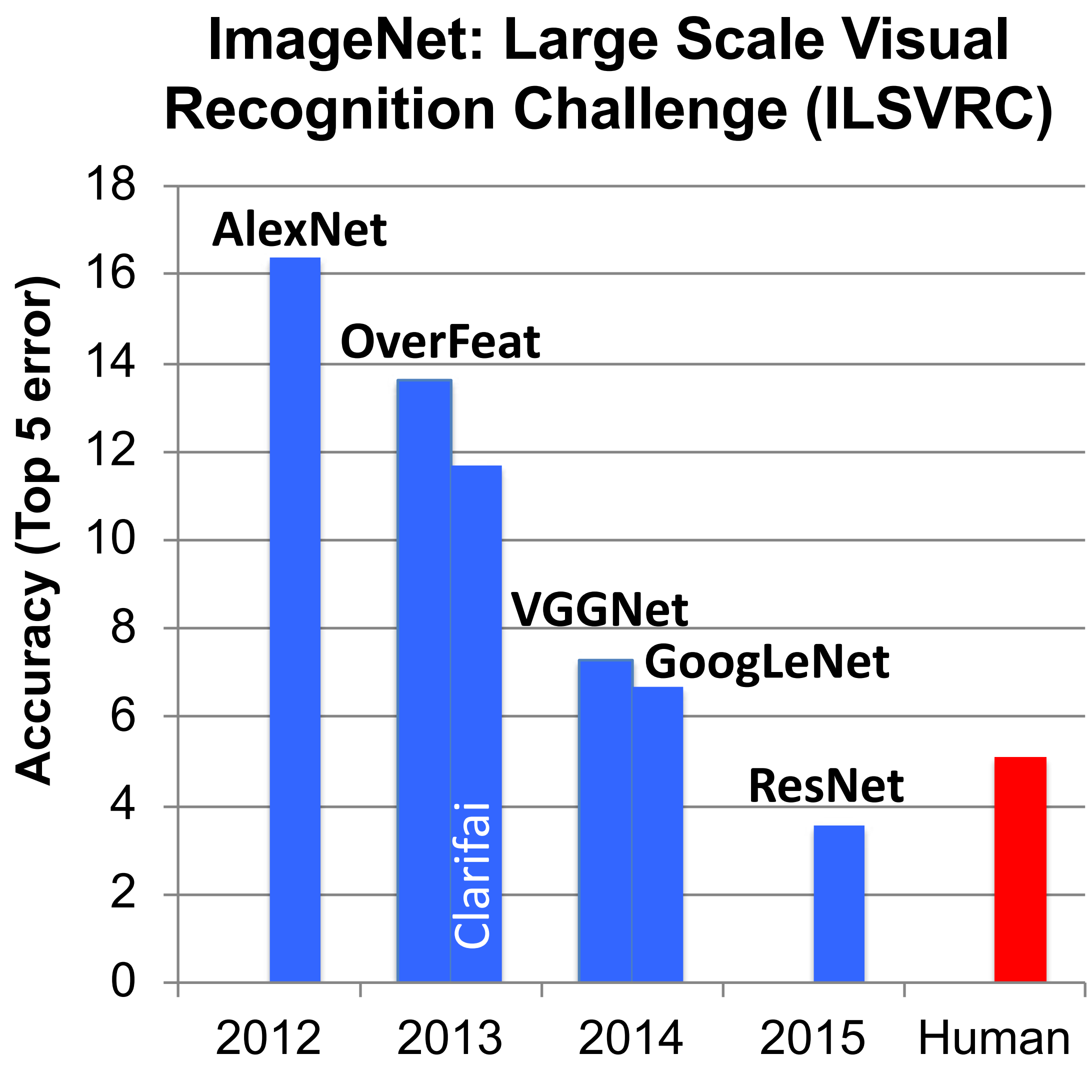
Relevant Components for this Tutorial

- Typical operations that we will discuss:
  - Convolution (CONV)
  - Fully-Connected (FC)
  - Max Pooling
  - ReLU

Popular DNN Models

Popular DNNs

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



[O. Russakovsky et al., IJCV 2015]

ImageNet

IMAGENET

Image Classification
~256x256 pixels (color)
1000 Classes
1.3M Training
100,000 Testing (50,000 Validation)

For **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**
accuracy of classification task reported based on top-1 and top-5 error

Image Source: http://karpathy.github.io/

http://www.image-net.org/challenges/LSVRC/

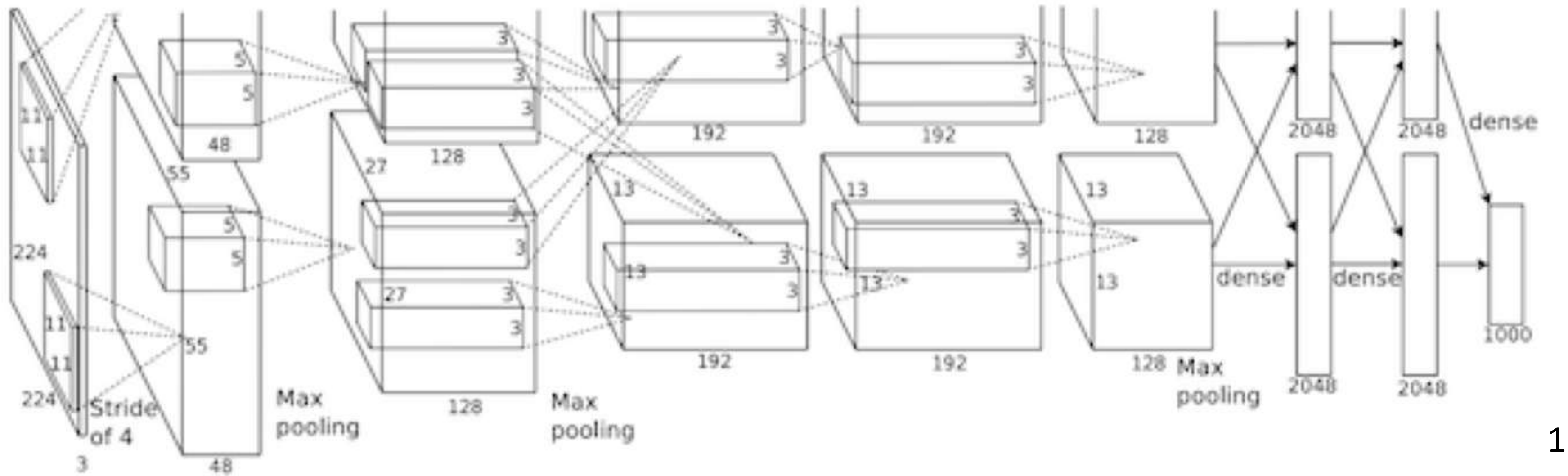
AlexNet

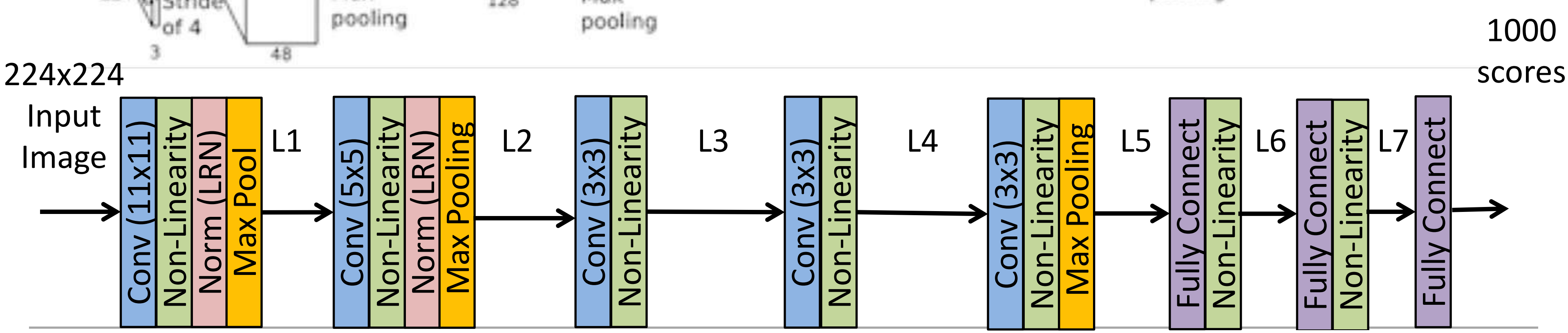
CONV Layers: 5
Fully Connected Layers: 3
Weights: 61M
MACs: 724M
ReLU used for non-linearity

ILSCVR12 Winner

Uses Local Response Normalization (LRN)

[Krizhevsky et al., NeurIPS 2012]





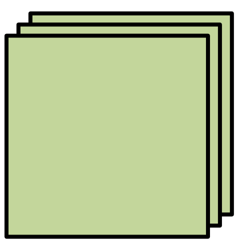
# of weights 34k 307k 885k 664k 442k 37.7M 16.8M 4.1M

Large Sizes with Varying Shapes

AlexNet Convolutional Layer Configurations

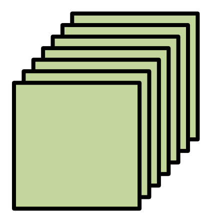
Layer	Filter Size (RxS)	# Filters (M)	# Channels (C)	Stride
1	11x11	96	3	4
2	5x5	256	48	1
3	3x3	384	256	1
4	3x3	384	192	1
5	3x3	256	192	1

Layer 1



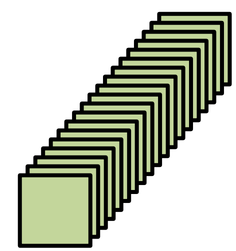
34k Params
105M MACs

Layer 2



307k Params
224M MACs

Layer 3



885k Params
150M MACs

[Krizhevsky et al., NeurIPS 2012]

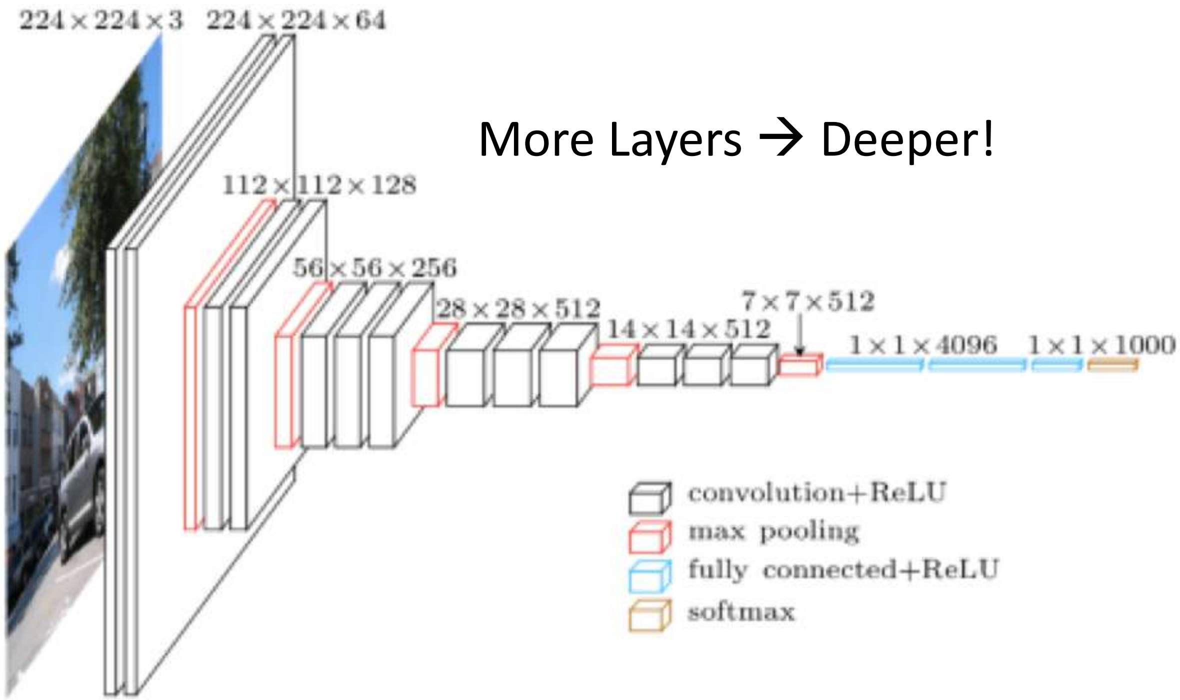
VGG-16

CONV Layers: 13
Fully Connected Layers: 3
Weights: 138M
MACs: 15.5G

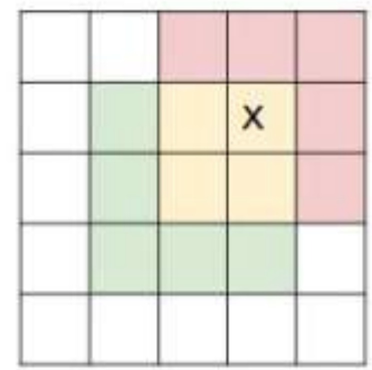
Also, 19 layer version

Reduce # of weights

More Layers → Deeper!



stack 2
3x3 conv



for a 5x5
receptive field

*[figure credit
A. Karpathy]*

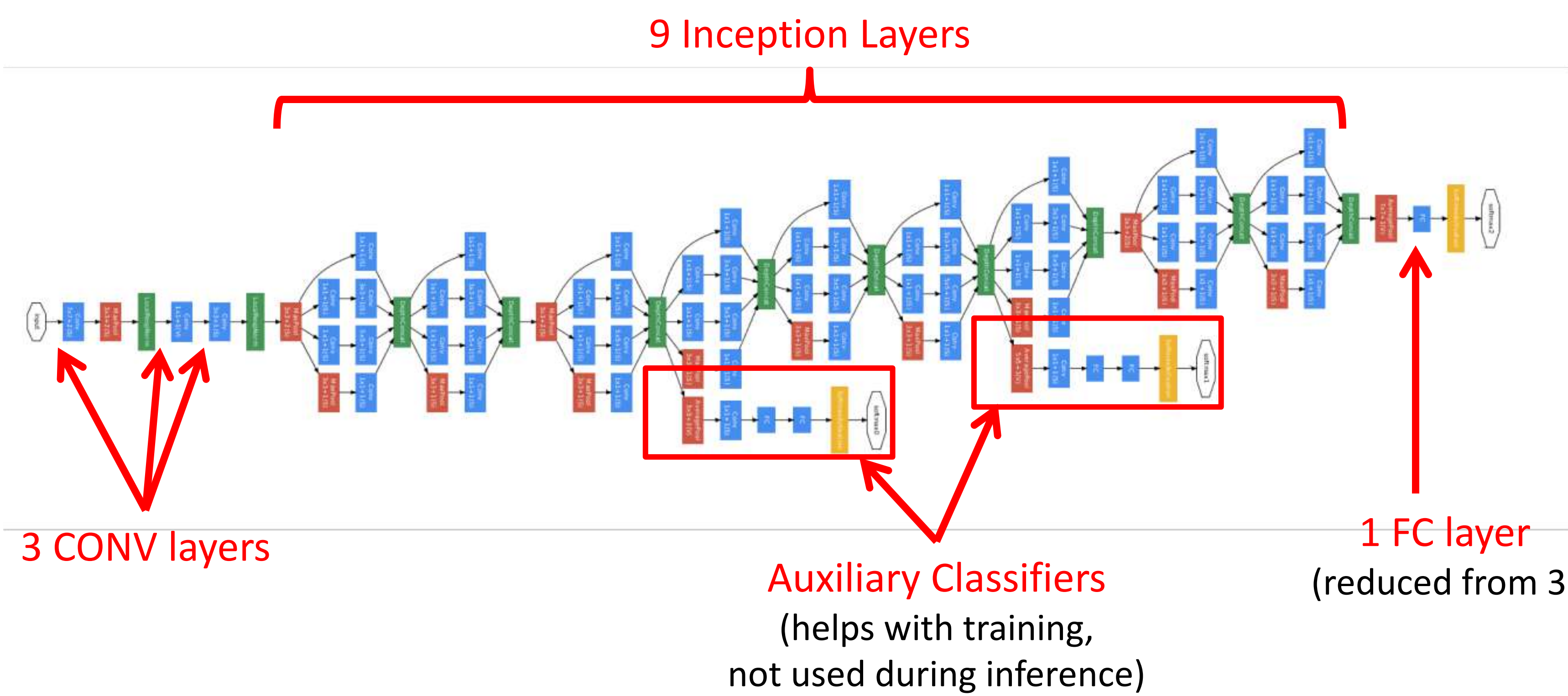
Image Source: http://www.cs.toronto.edu/~frossard/post/vgg16/

[Simonyan et al., arXiv 2014, ICLR 2015]

GoogLeNet/Inception (v1)

CONV Layers: 21 (depth), 57 (total)
Fully Connected Layers: 1
Weights: 7.0M
MACs: 1.43G

Also, v2, v3 and v4
ILSVRC14 Winner



[Szegedy et al., arXiv 2014, CVPR 2015]

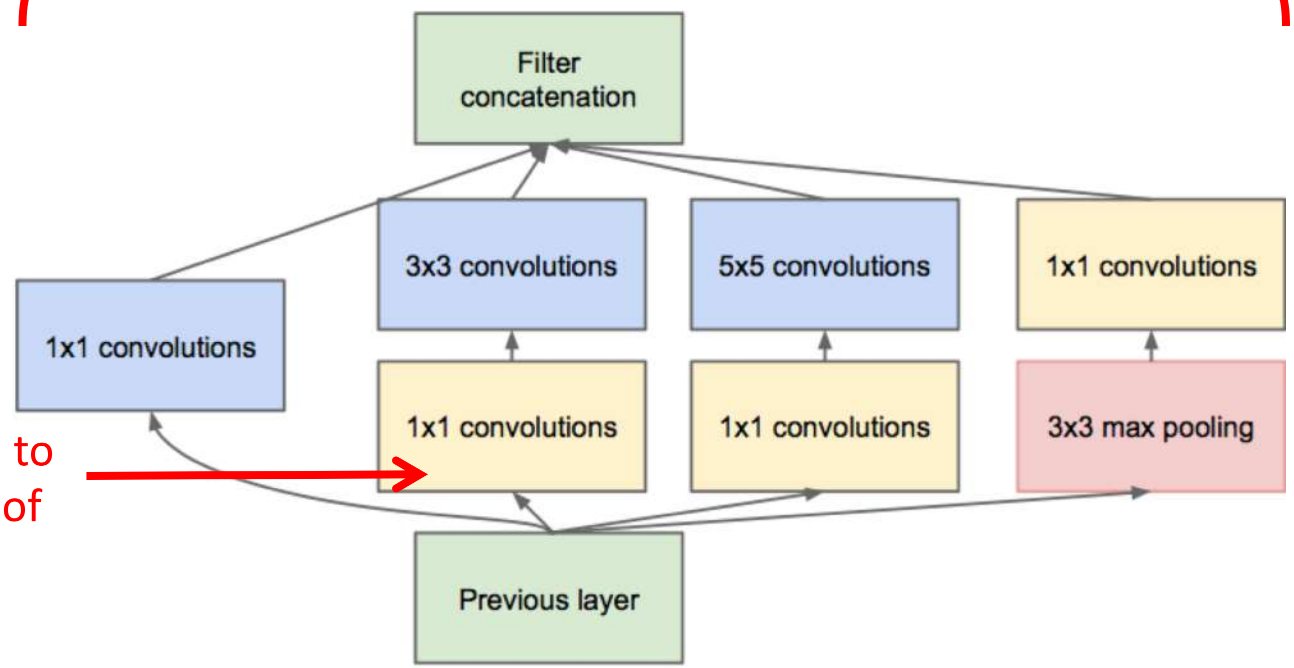
GoogLeNet/Inception (v1)

CONV Layers: 21 (depth), 57 (total)
Fully Connected Layers: 1
Weights: 7.0M
MACs: 1.43G

Also, v2, v3 and v4
ILSVRC14 Winner

parallel filters of different size have the effect of processing image at different scales

Inception Module



1x1 'bottleneck' to reduce number of weights and multiplications

[Szegedy et al., arXiv 2014, CVPR 2015]

ResNet

ILSVRC15 Winner
(better than human level accuracy!)

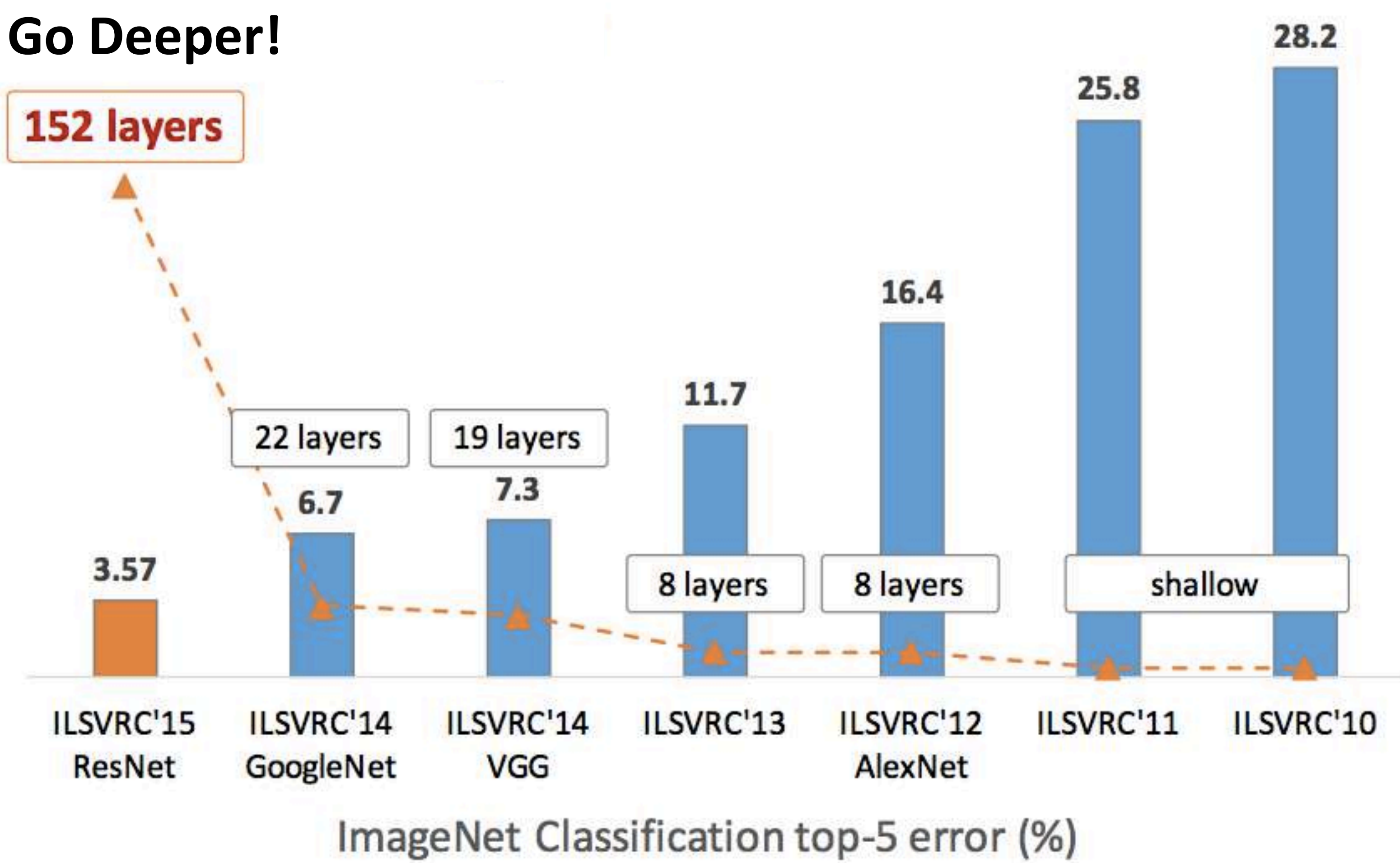


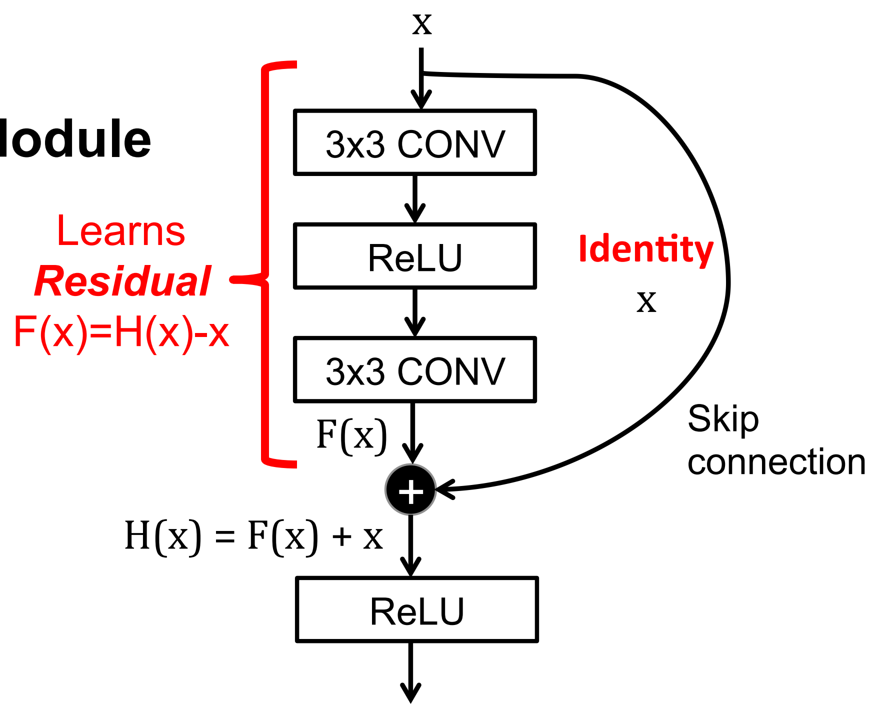
Image Source: http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

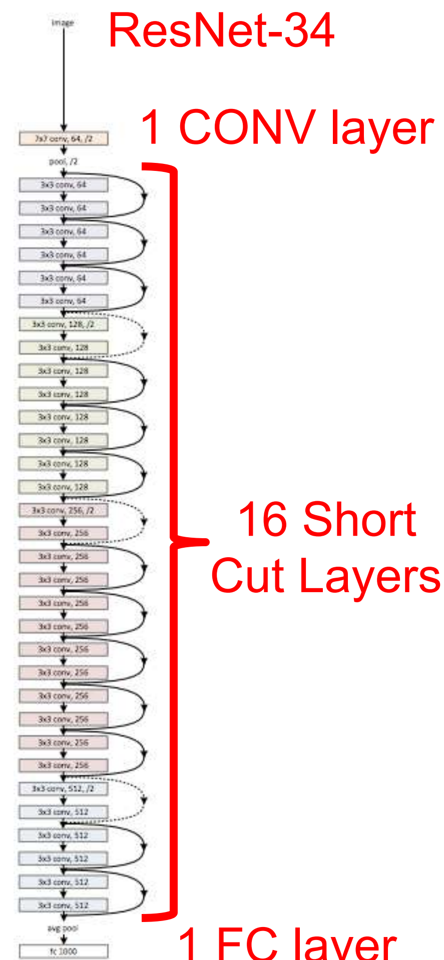
ResNet-50

CONV Layers: 49
Fully Connected Layers: 1
Weights: 25.5M
MACs: 3.9G

Also, 34,**152** and 1202 layer versions
ILSVRC15 Winner

Short Cut Module





Helps address the vanishing gradient challenge for training very deep networks

[He et al., arXiv 2015, CVPR 2016]

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**

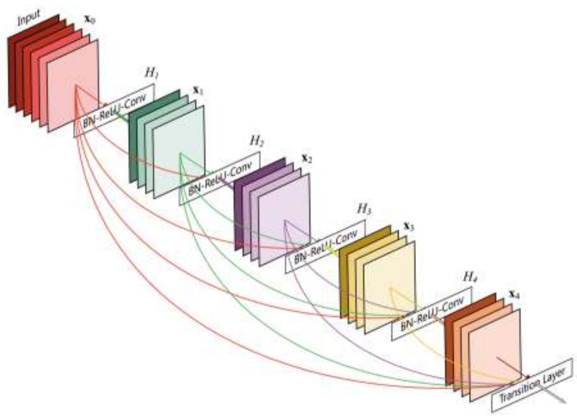
CONV Layers increasingly important!

Summary of Popular CNNs

- **AlexNet**
  - First CNN Winner of ILSVRC
  - Uses LRN (deprecated after this)
- **VGG-16**
  - Goes Deeper (16+ layers)
  - Uses only 3x3 filters (stack for larger filters)
- **GoogLeNet (v1)**
  - Reduces weights with Inception and only one FC layer
  - Inception: 1x1 and DAG (parallel connections)
  - Batch Normalization
- **ResNet**
  - Goes Deeper (24+ layers)
  - Shortcut connections

Beyond ResNet

DenseNet



[Huang et al., CVPR 2017]

Wide ResNet

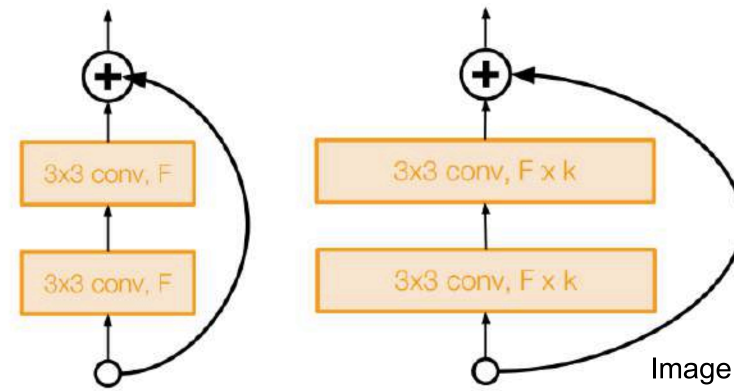
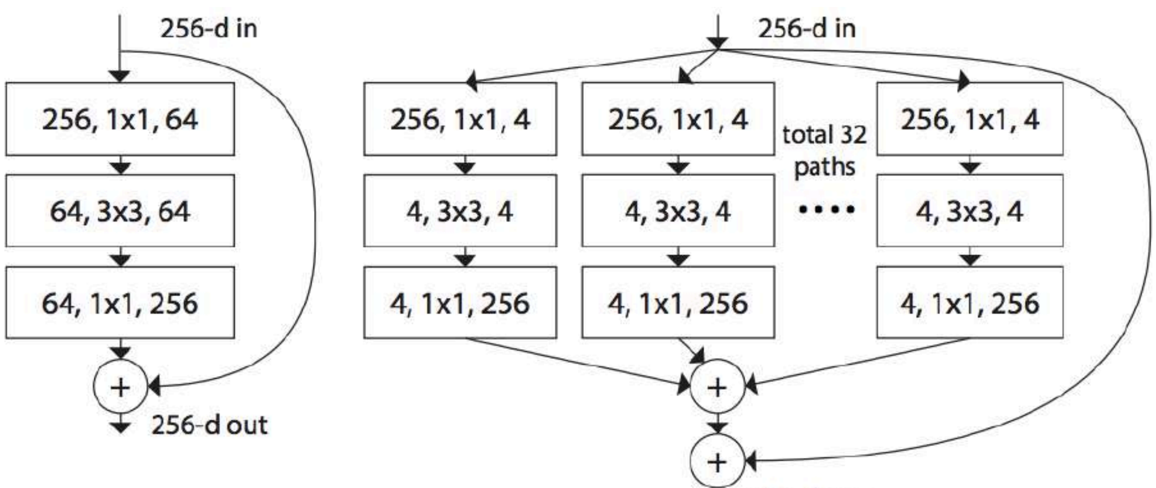


Image Source: Stanford cs231n

Basic residual block   Wide residual block

[Zagoruyko et al., BMVC 2016]

ResNeXt



[Xie et al., CVPR 2017]

Increase accuracy ***without*** going deeper!

Part 1: Hardware Platforms for DNN Processing

GPUs and CPUs Targeting Deep Learning

Intel Xeon Scalable CPU (2019)

Nvidia's V100 GPU (2018)

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

Matrix Multiplication Libraries

- 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 storage hierarchy

Map DNN to a Matrix Multiplication

Convolution:

$$\underset{\text{Filter}}{\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}} * \underset{\text{Input Fmap}}{\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}} = \underset{\text{Output Fmap}}{\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}}$$

Matrix Mult:

Toeplitz Matrix (w/ redundant data)

$$\begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix} \times \begin{bmatrix}1 & 2 & 4 & 5\\ 2 & 3 & 5 & 6\\ 4 & 5 & 7 & 8\\ 5 & 6 & 8 & 9\end{bmatrix} = \begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix}$$

Data is repeated

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

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

- **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

Specialized Hardware (Accelerators)

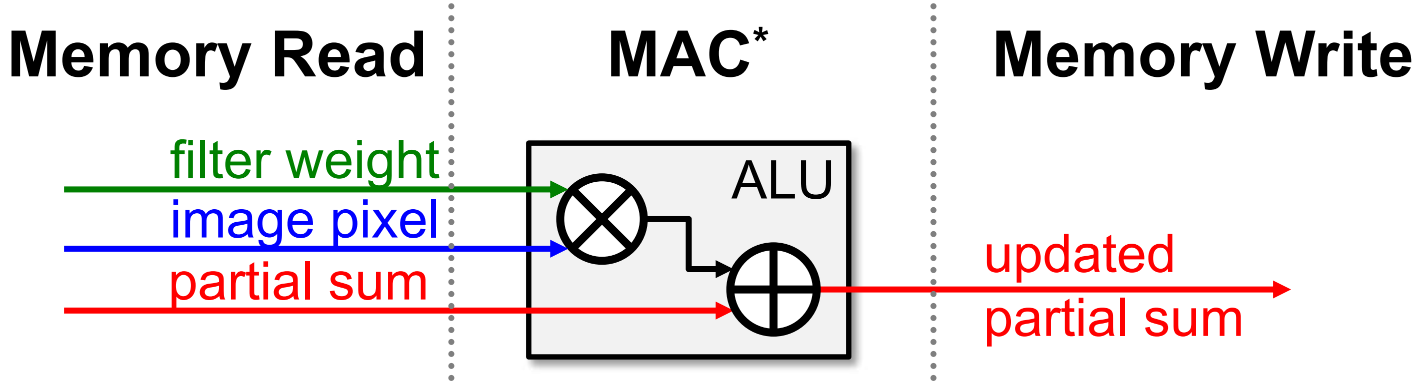
Properties We Can Leverage

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

Properties We Can Leverage

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

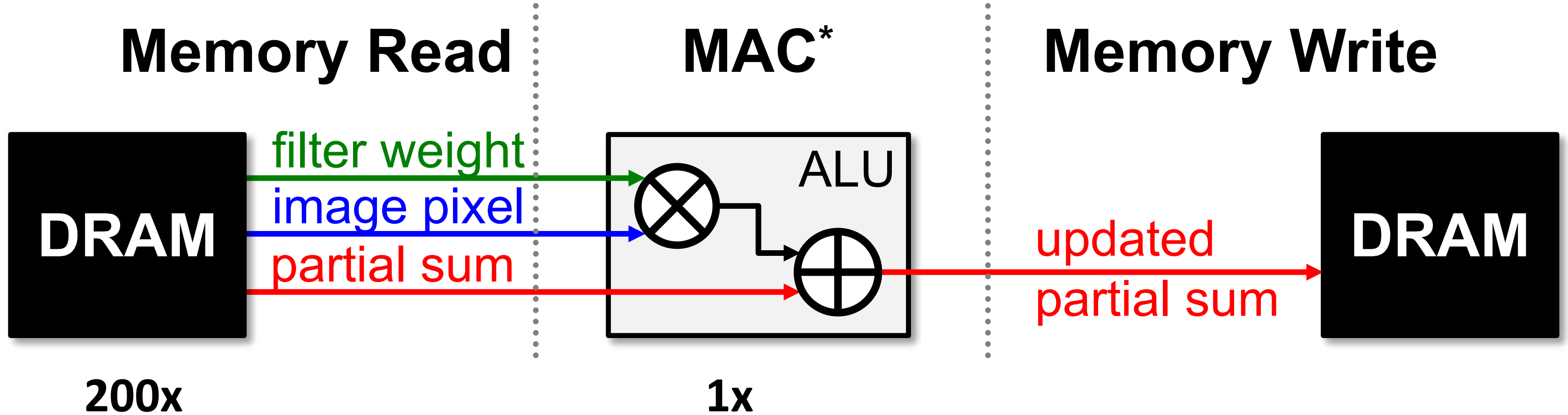
- Memory Access is the Bottleneck



Properties We Can Leverage

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

- Memory Access is the Bottleneck



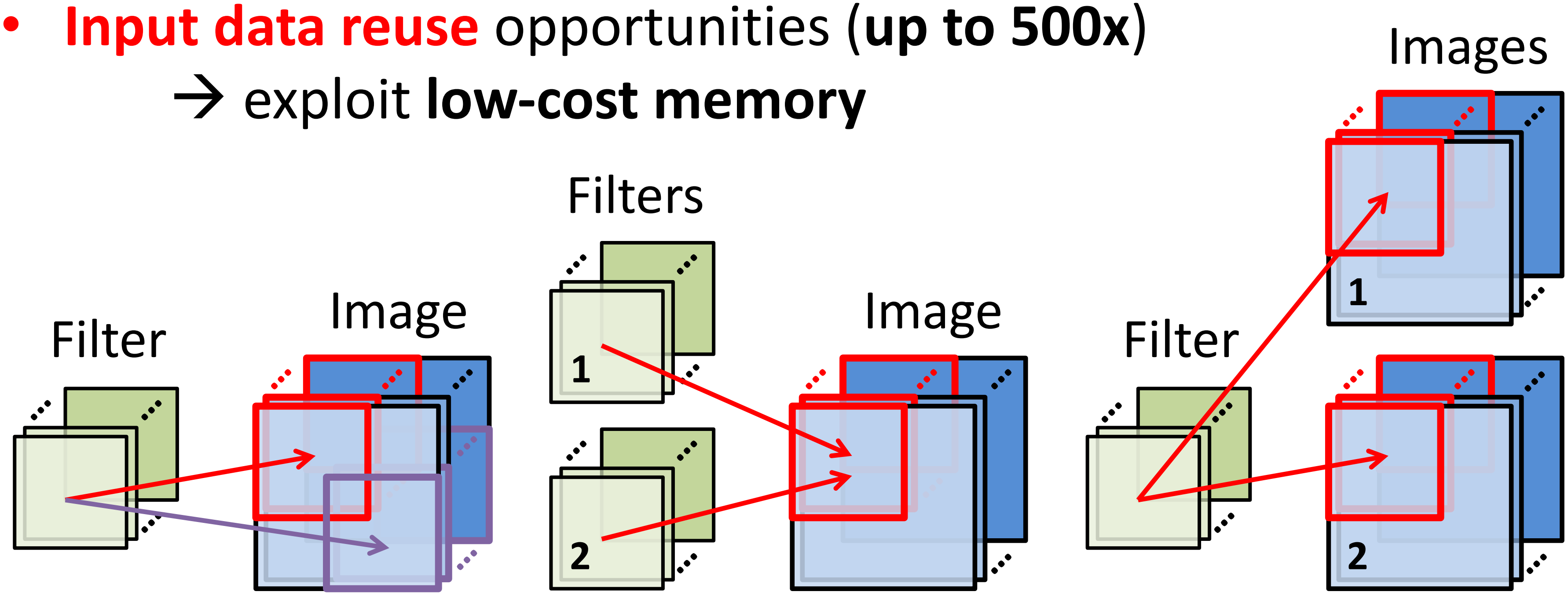
Worst Case: all memory R/W are **DRAM** accesses

- Example: AlexNet [NeurIPS 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**



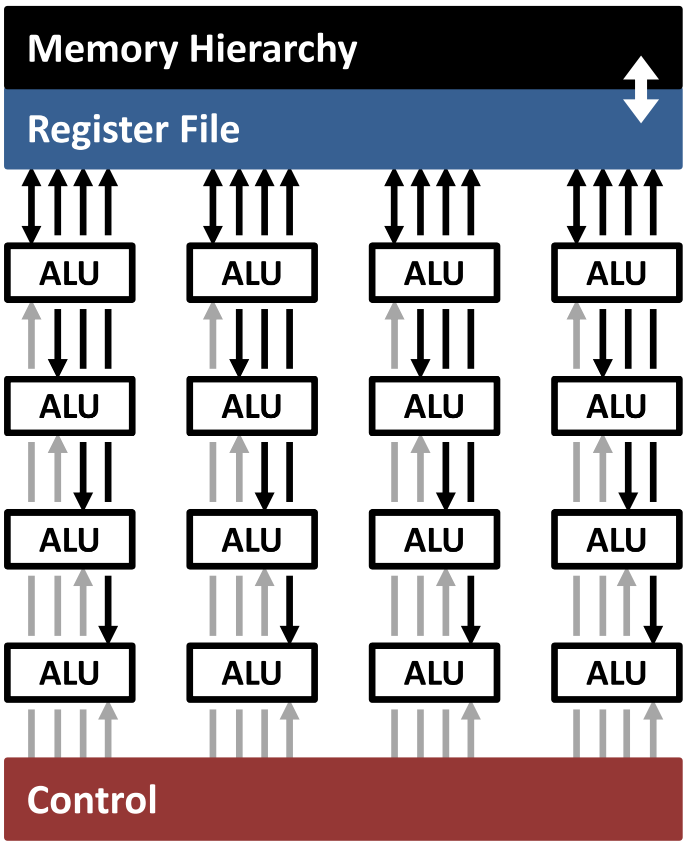
Convolutional Reuse (pixels, weights)

Image Reuse (pixels)

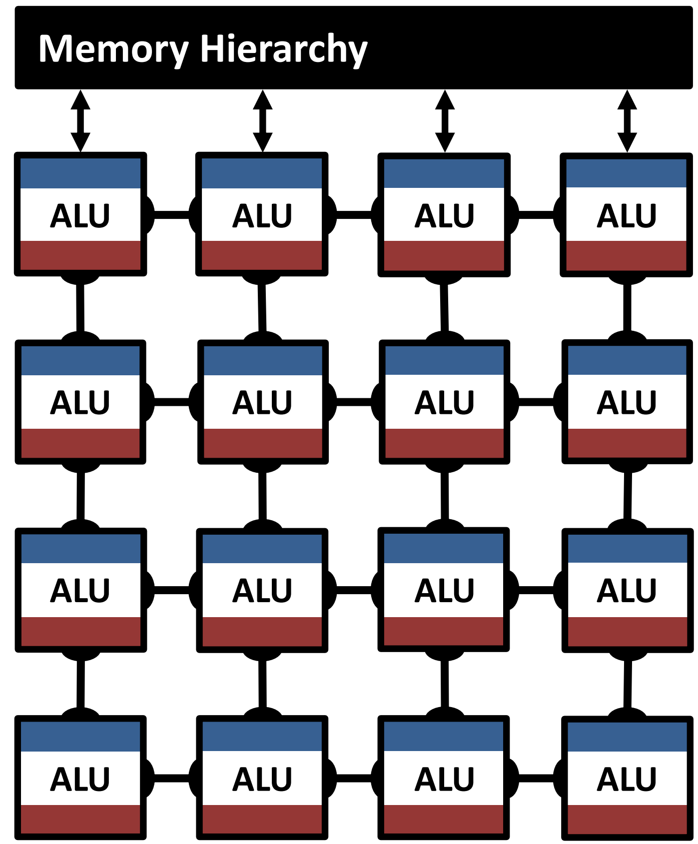
Filter Reuse (weights)

Highly-Parallel Compute Paradigms

Temporal Architecture (SIMD/SIMT)



Spatial Architecture (Dataflow Processing)



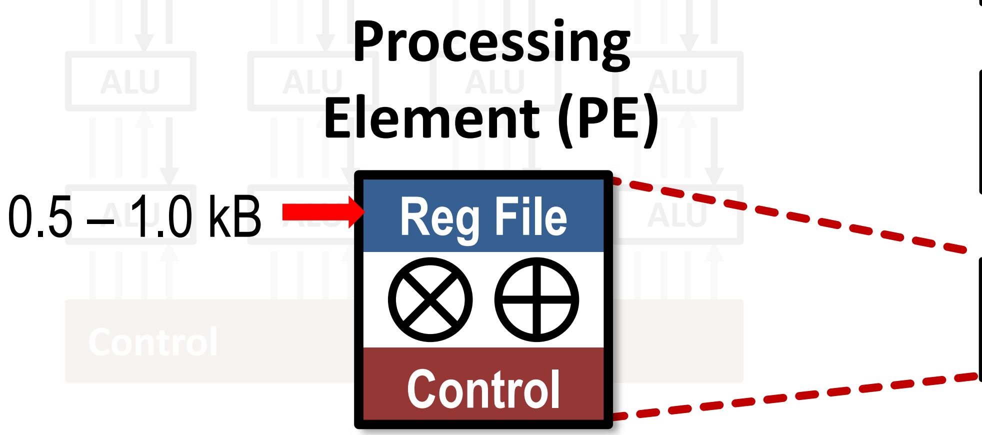
Advantages of Spatial Architecture

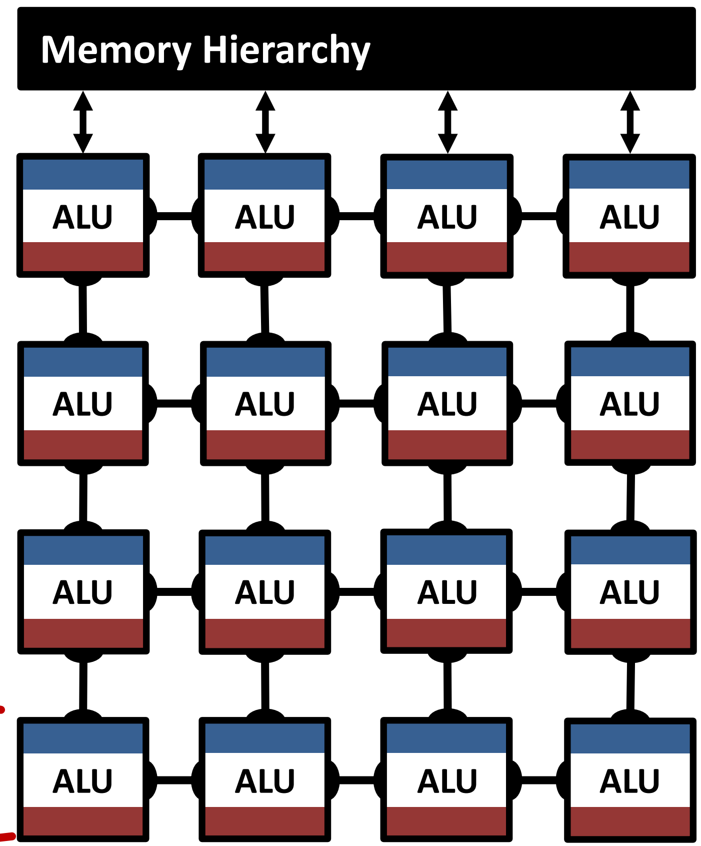
Temporal Architecture (SIMD/SIMT)

Spatial Architecture (Dataflow Processing)

Efficient Data Reuse
Distributed local storage (RF)

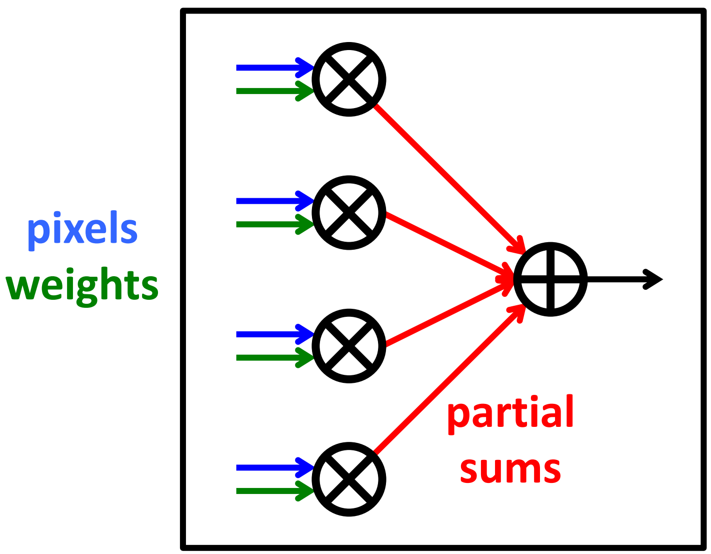
Inter-PE Communication
Sharing among regions of PEs

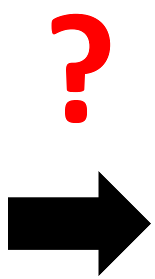




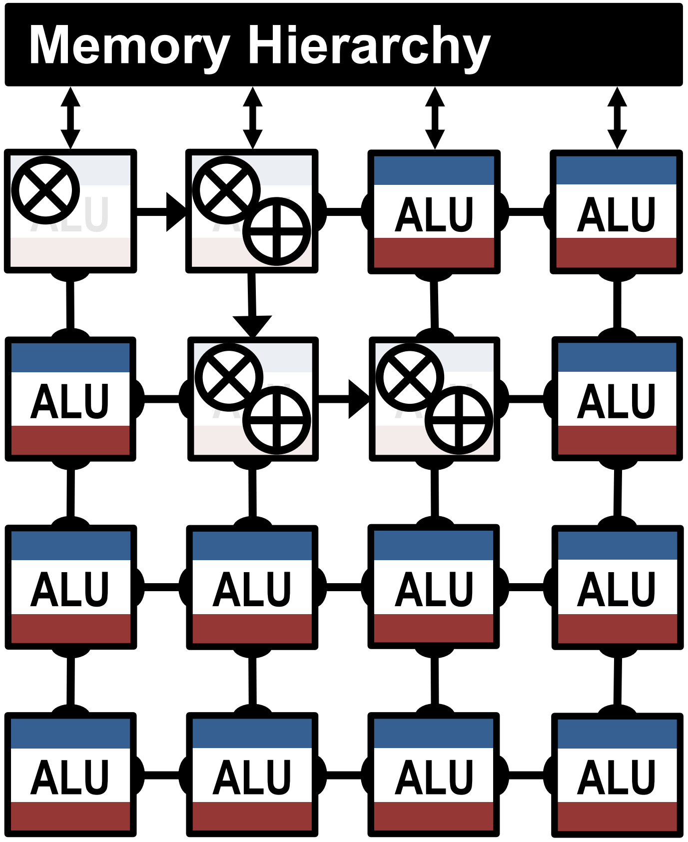
How to Map the Dataflow?

CNN Convolution





Spatial Architecture (Dataflow Processing)

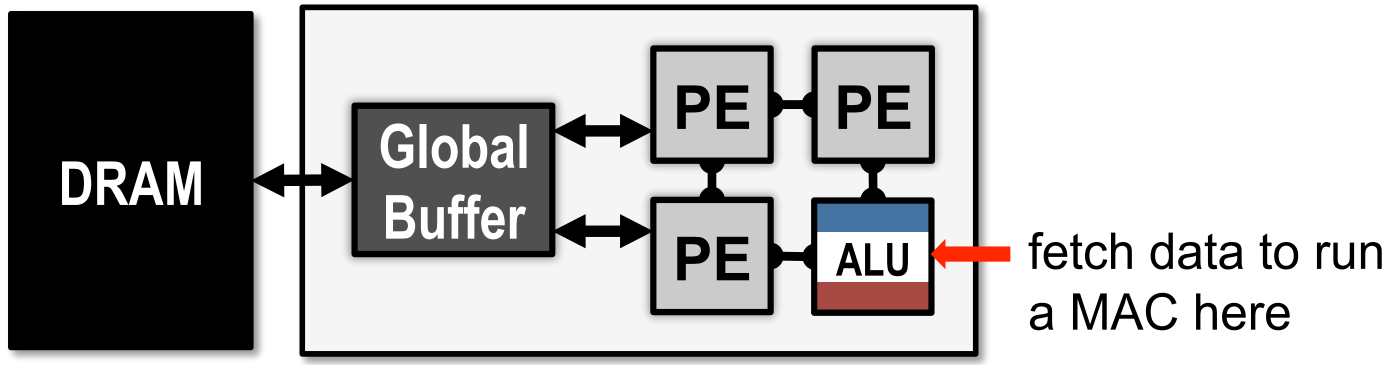


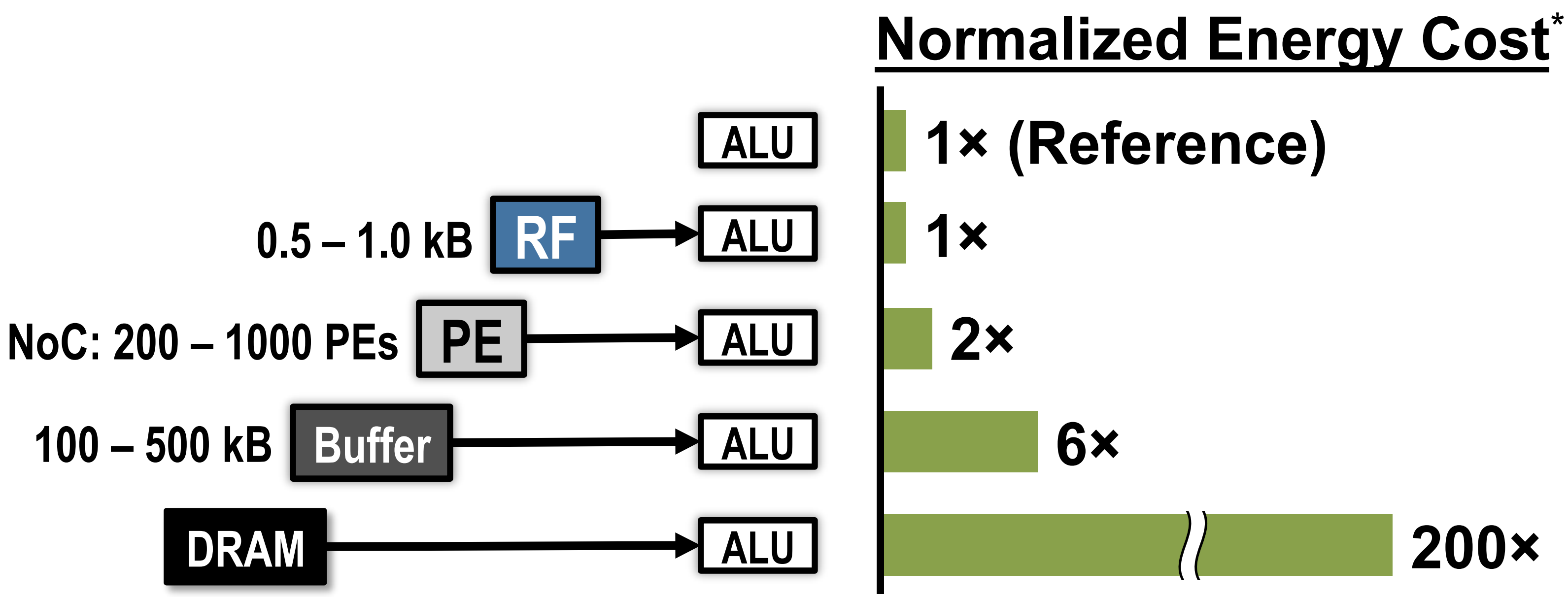
Goal: Increase reuse of input data (**weights** and **pixels**) and local **partial sums** accumulation

Energy-Efficient Dataflow

Y.-H. Chen, J. Emer, V. Sze, “Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks,” ISCA 2016

Data Movement is Expensive

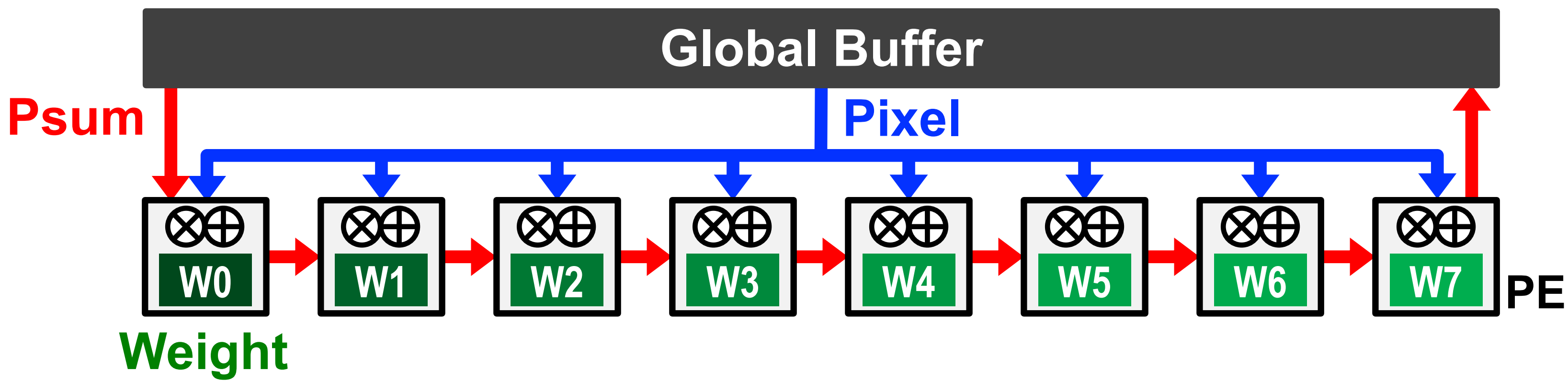




* measured from a commercial 65nm process

Maximize data reuse at low cost levels of hierarchy

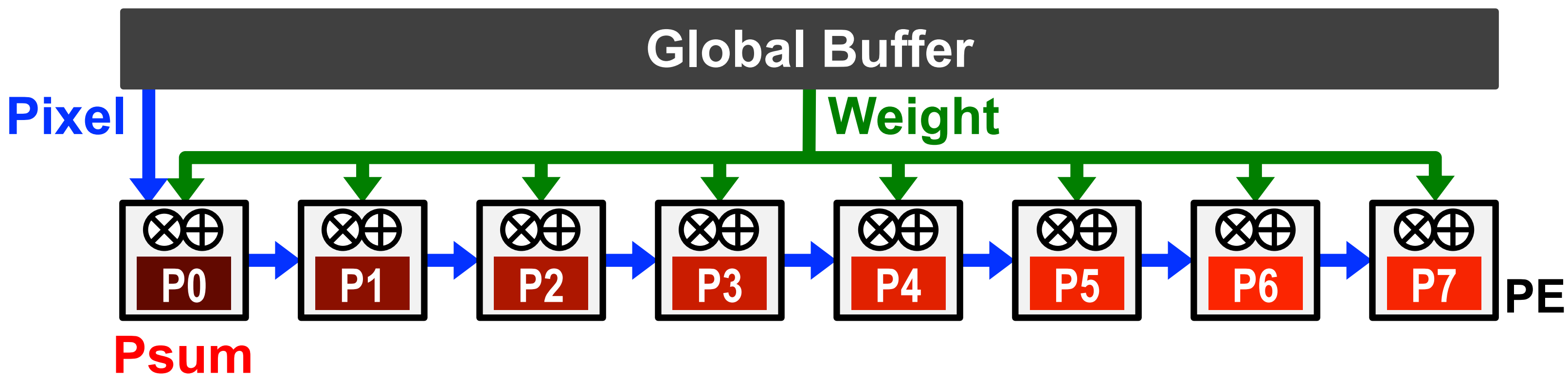
Weight Stationary (WS)



- **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)

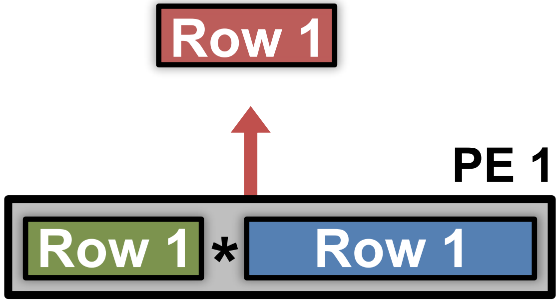


- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Examples:**

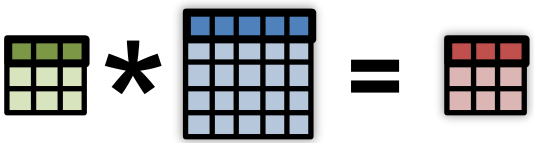
  [**Gupta**, *ICML* 2015] [**ShiDianNao**, *ISCA* 2015]

  [**Peemen**, *ICCD* 2013]

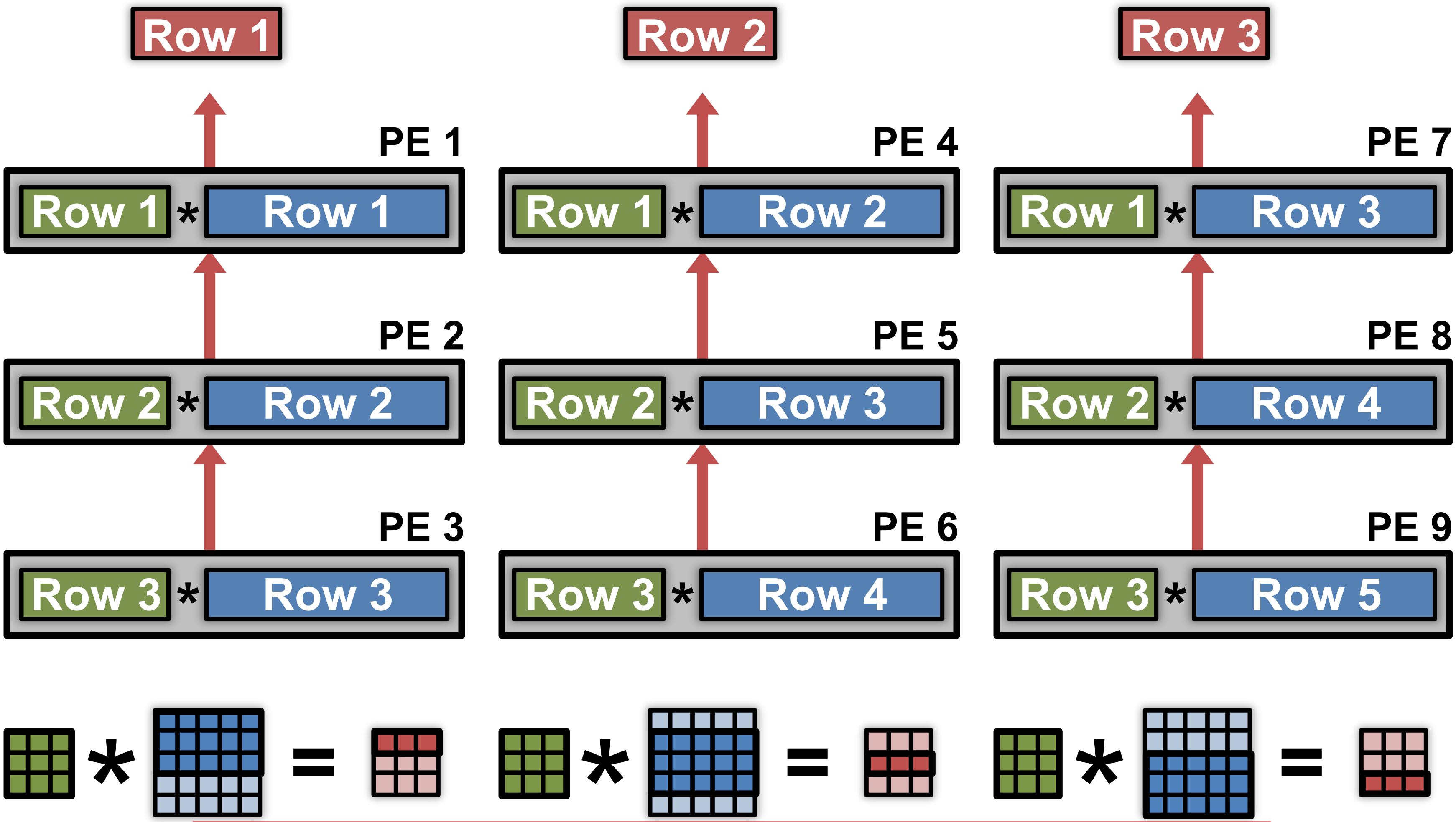
Row Stationary Dataflow



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



Row Stationary Dataflow



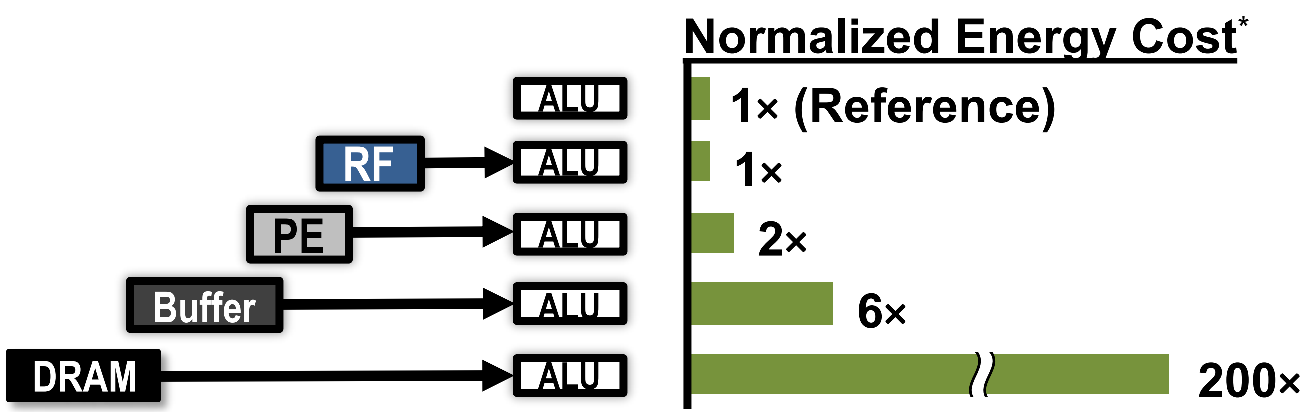
Optimize for **overall energy efficiency** instead for only a certain data type

Evaluate Reuse in Different Dataflows

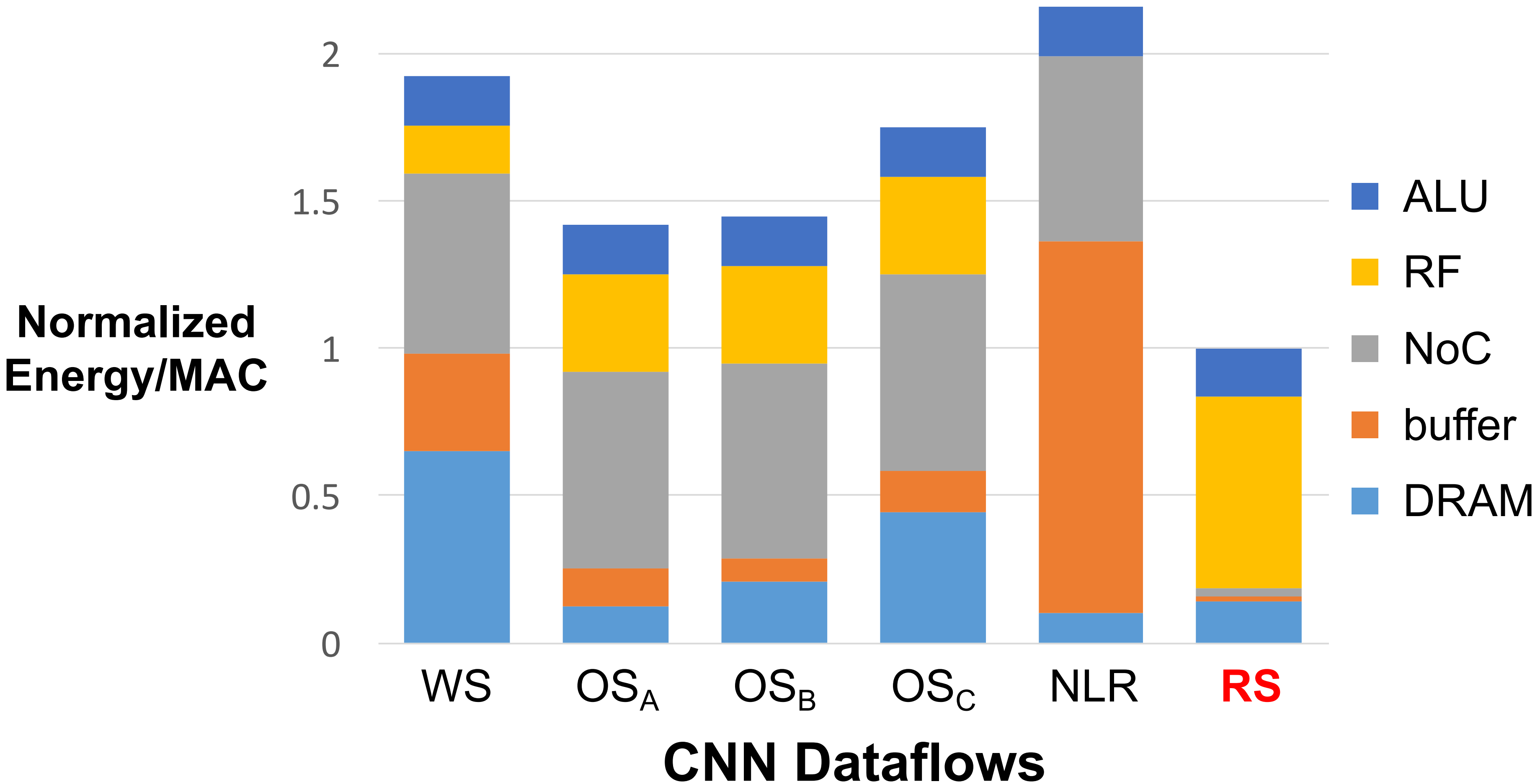
- **Weight Stationary**
  - Minimize movement of filter weights
- **Output Stationary**
  - Minimize movement of partial sums
- **No Local Reuse**
  - Don't use any local PE storage. Maximize global buffer size.
- **Row Stationary**

<u>Evaluation Setup</u>

- Same Total Area
- AlexNet
- 256 PEs
- Batch size = 16



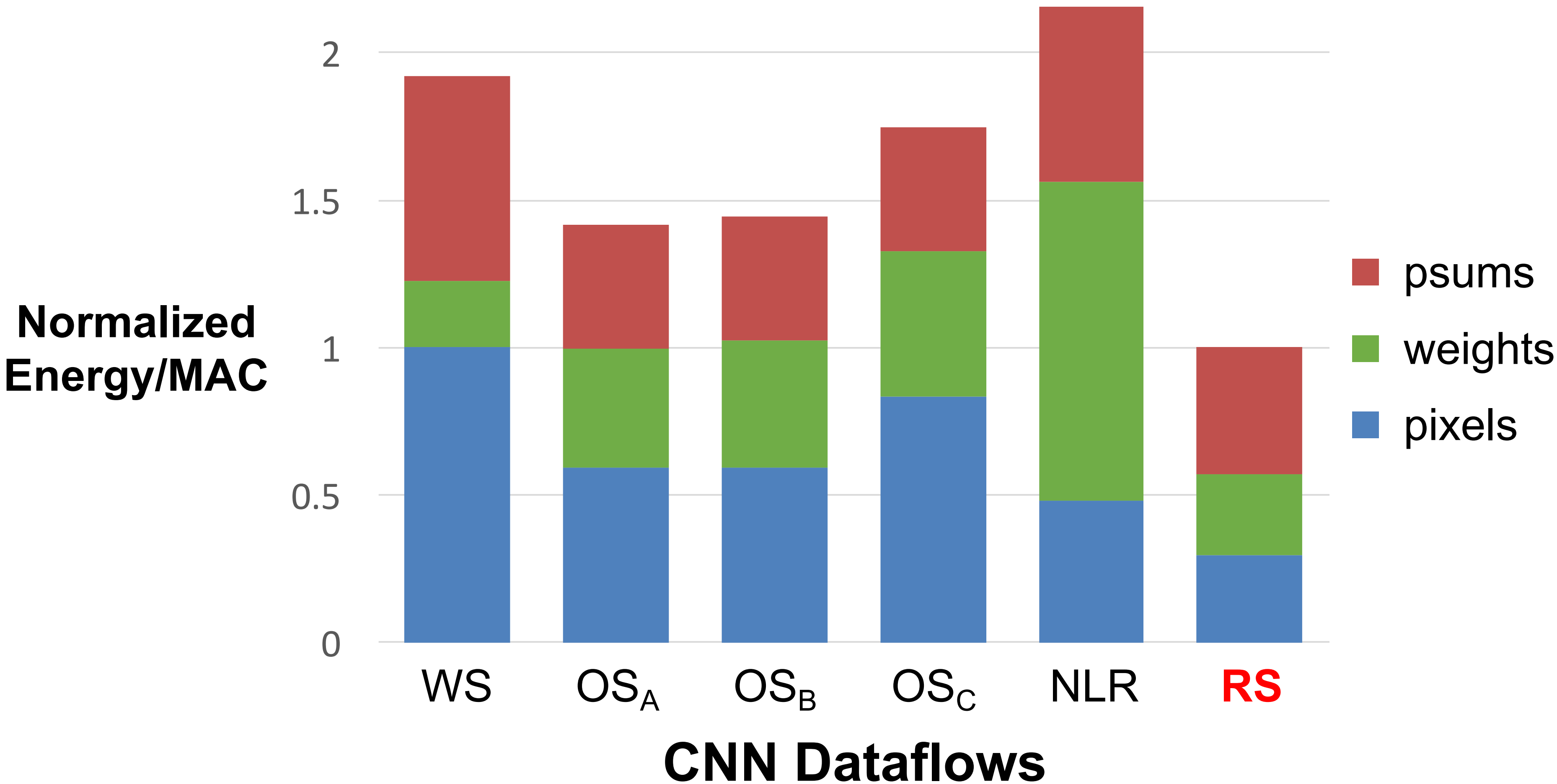
Dataflow Comparison: CONV Layers



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

[Chen et al., ISCA 2016]

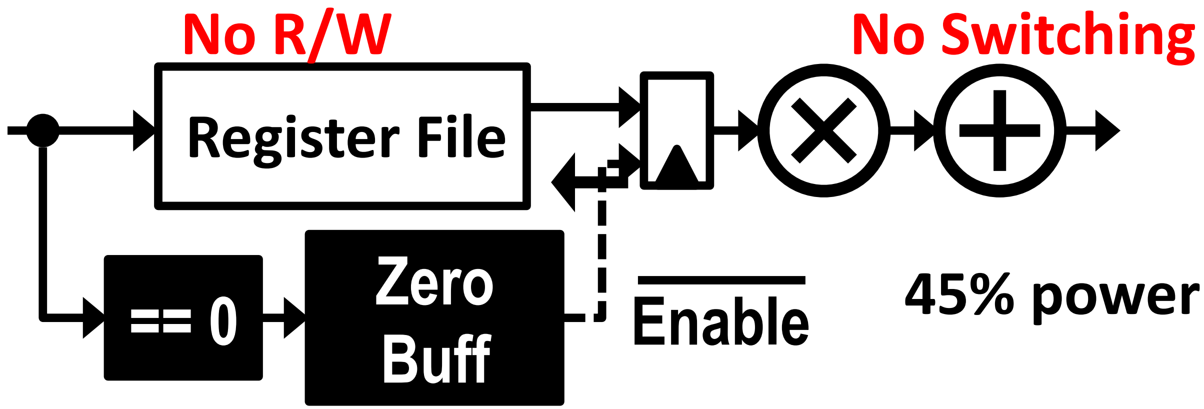
Dataflow Comparison: CONV Layers



RS optimizes for the best **overall** energy efficiency

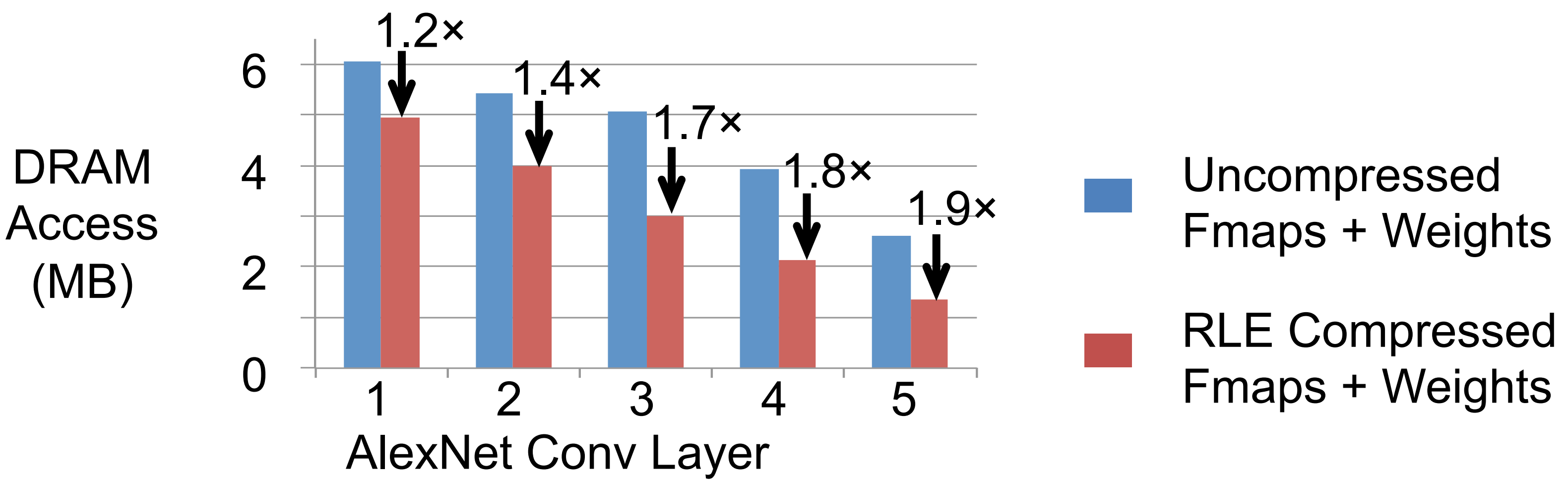
Exploit Sparsity

Method 1. Skip memory access and computation

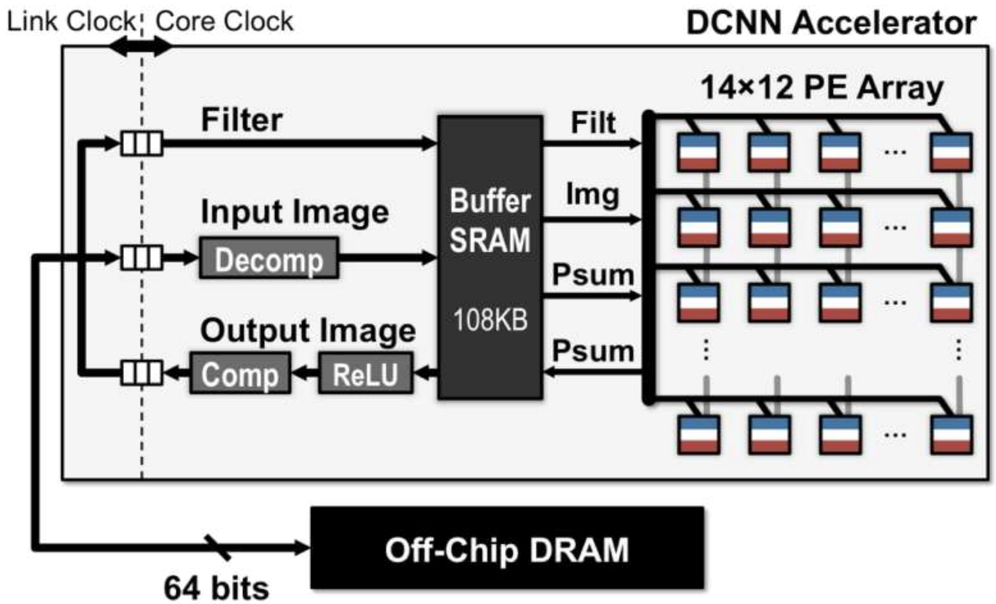


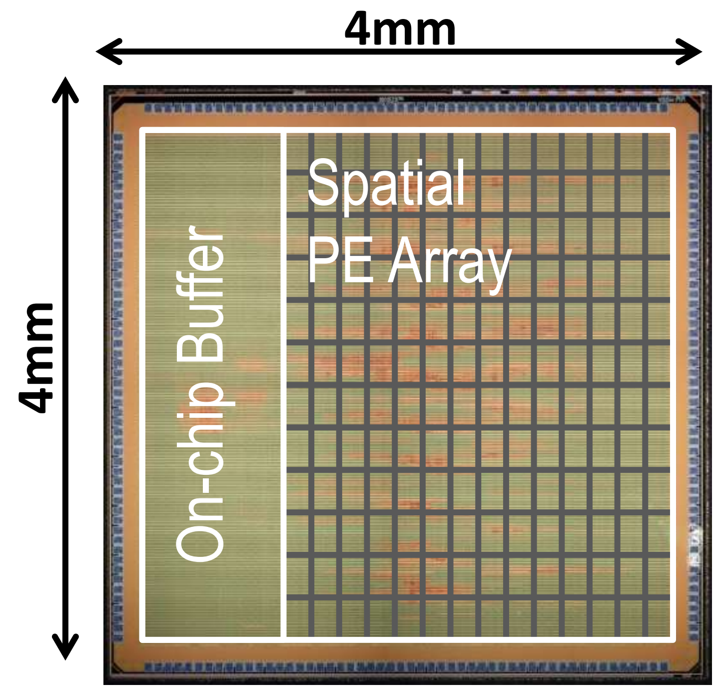
45% power reduction

Method 2. Compress data to reduce storage and data movement



Eyeriss: Deep Neural Network Accelerator





[Chen et al., ISSCC 2016, ISCA 2016]

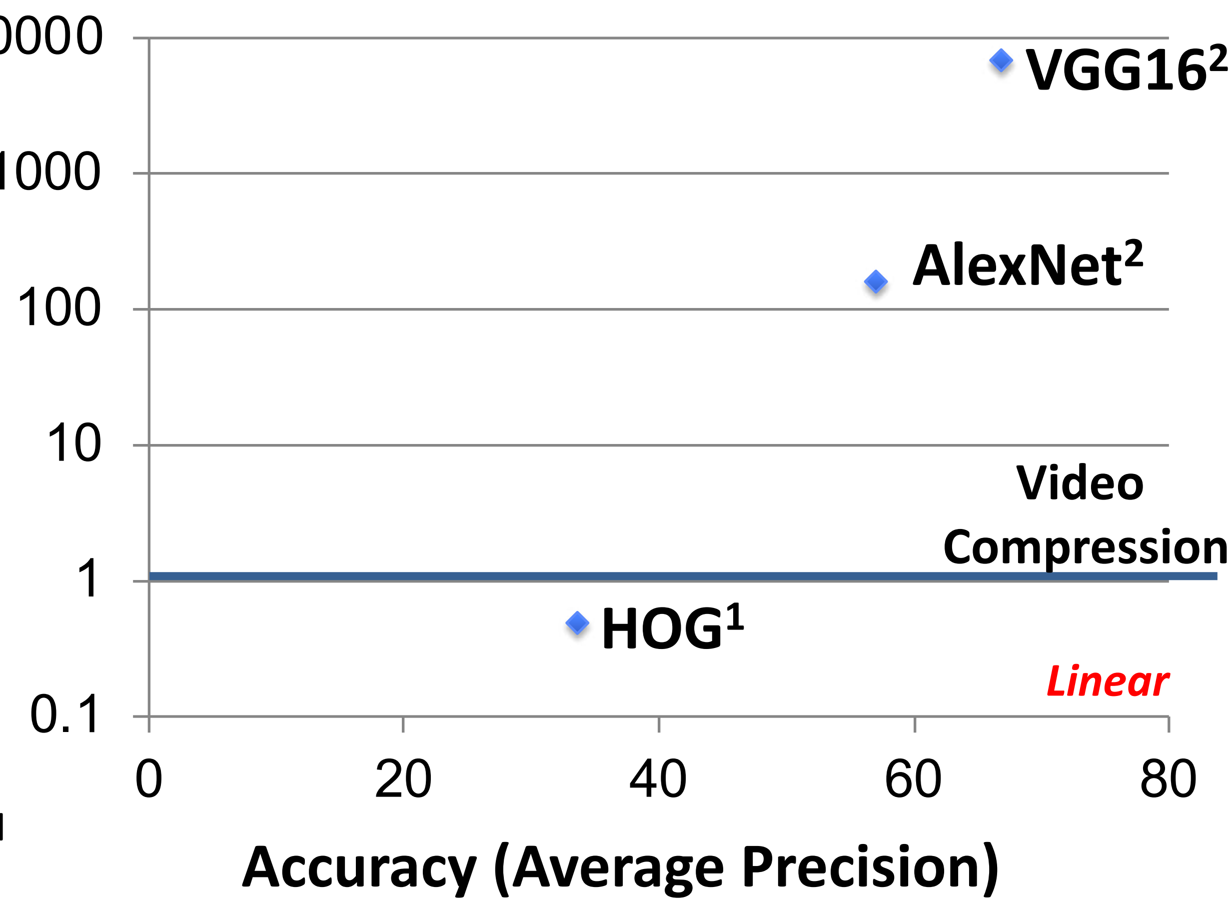
Exploits data reuse for **100x** reduction in memory accesses from global buffer and **1400x** reduction in memory accesses from off-chip DRAM

Overall **>10x energy reduction** compared to a mobile GPU (Nvidia TK1)

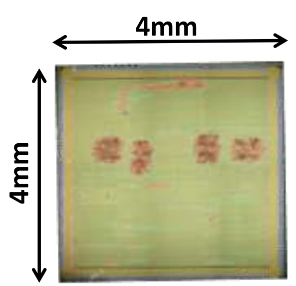
Results for AlexNet

[Joint work with Joel Emer]

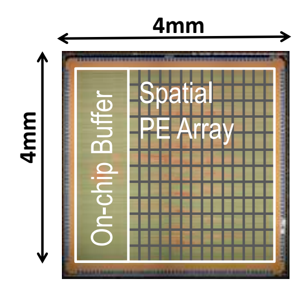
Features: Energy vs. Accuracy



*Measured in 65nm**



❶ [Suleiman, VLSI 2016]



❷ [Chen, ISSCC 2016]

** Only feature extraction. Does not include data, classification energy, augmentation and ensemble, etc.*

Measured in on VOC 2007 Dataset

1. DPM v5 [Girshick, 2012]
2. Fast R-CNN [Girshick, CVPR 2015]

Benchmarking Metrics for DNN Hardware

How can we compare designs?

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

Metrics for DNN Hardware

- **Accuracy**
  - Quality of result for a given task
- **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**
  - Edge and embedded devices have limited battery capacity
  - Data centers have stringent power ceilings due to cooling costs
- **Hardware Cost**
  - $$$

Specifications to Evaluate Metrics

- **Accuracy**
  - Difficulty of dataset and/or task should be considered
- **Throughput**
  - Number of cores (include utilization along with peak performance)
  - Runtime for running specific DNN models
- **Latency**
  - Include batch size used in evaluation
- **Energy and Power**
  - Power consumption for running specific DNN models
  - Include external memory access
- **Hardware Cost**
  - On-chip storage, number of cores, chip area + process technology

Example: Metrics of Eyeriss Chip

ASIC Specs	Input
Process Technology	65nm LP TSMC (1.0V)
Total Core Area (mm^2)	12.25
Total On-Chip Memory (kB)	192
Number of Multipliers	168
Clock Frequency (MHz)	200
Core area (mm^2) /multiplier	0.073
On-Chip memory (kB) / multiplier	1.14
Measured or Simulated	Measured

Metric	Units	Input
Name of CNN Model	Text	AlexNet
Top-5 error classification on ImageNet	#	19.8
Supported Layers		All CONV
Bits per weight	#	16
Bits per input activation	#	16
Batch Size	#	4
Runtime	ms	115.3
Power	mW	278
Off-chip Access per Image Inference	MBytes	3.85
Number of Images Tested	#	100

Comprehensive Coverage

- **All metrics** should be reported for fair evaluation of design tradeoffs
- Examples of what can happen if 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 bandwidth**, one could build a processor with only multipliers 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?

Evaluation Process

The evaluation process for whether a DNN system 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

Part 2: Co-Design of Algorithms and Hardware for DNNs

Approaches

- **<u>Reduce size</u> of operands for storage/compute**
  - Floating point → Fixed point
  - Bit-width reduction
  - Non-linear quantization

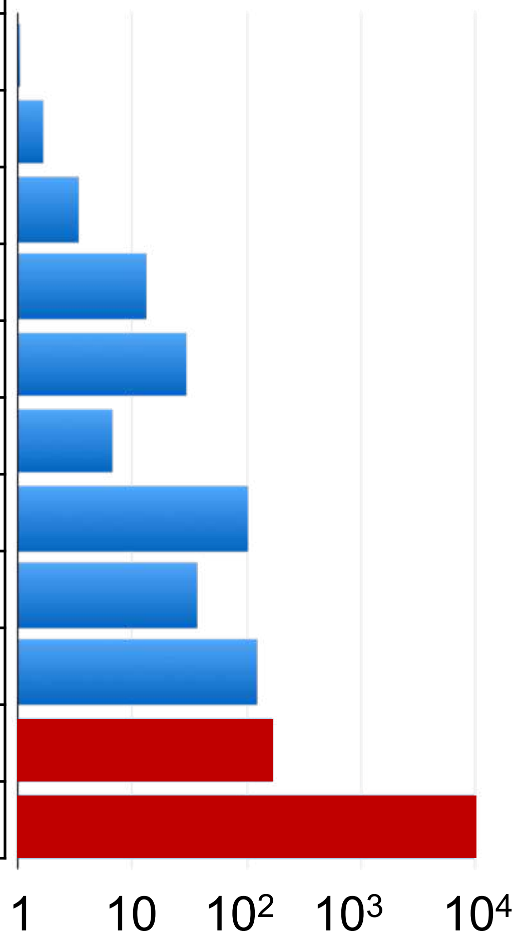
- **<u>Reduce number</u> of operations for storage/compute**
  - Exploit Activation Statistics (Compression)
  - Network Pruning
  - Compact Network Architectures

Reduced Precision

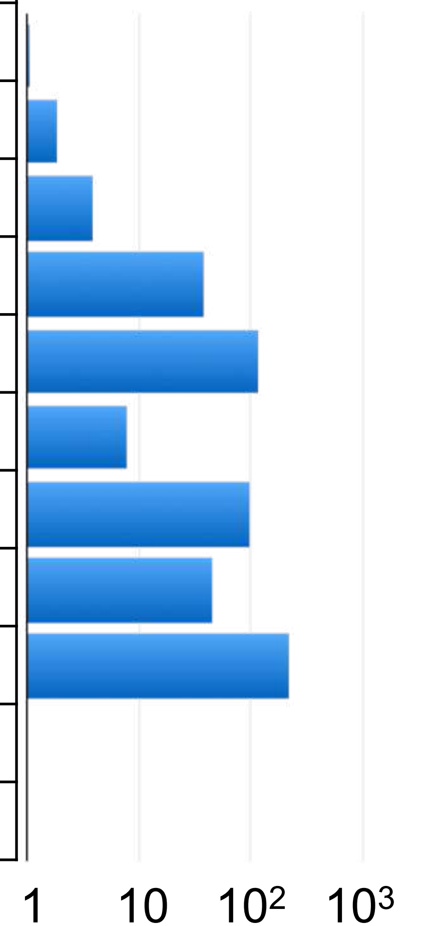
Cost Per Operation

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

Relative Energy Cost



Relative Area Cost



[Horowitz, ISSCC 2014]

Floating Point → Fixed Point

Mantissa (m): number of levels
Exponent (e): scale to a target range
Sign (s): indicates if number is positive or negative

Floating Point $(-1)^s \times m \times 2^{(e-127)}$

	sign	exponent (8-bits)	mantissa (23-bits)
32-bit float	1	0 1 0 0 1 0 1 0	0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0
-1.42122425 x 10^{-13}	***s*** = 1	***e*** = 70	***m*** = 20482

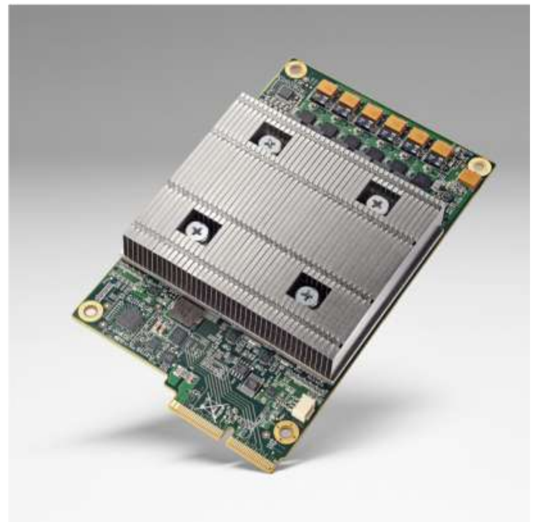
Fixed Point $(-1)^s \times m$

sign mantissa (7-bits)

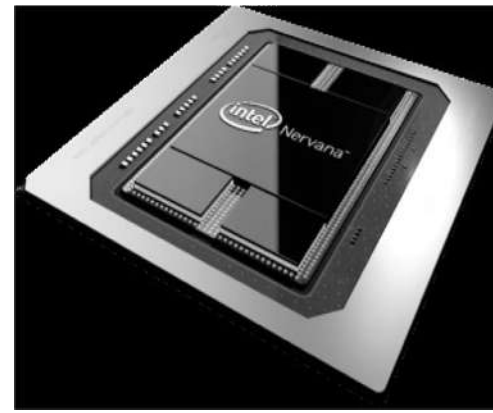
	sign	integer (4-bits)	fractional (3-bits)
8-bit fixed	0	1 1 0 0	1 1 0
12.75	***s*** = 0	***m***=*102*	

Commercial Products Support Reduced Precision

Nvidia's Pascal (2016)

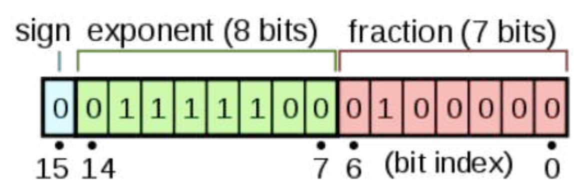


Google's TPU (2016)



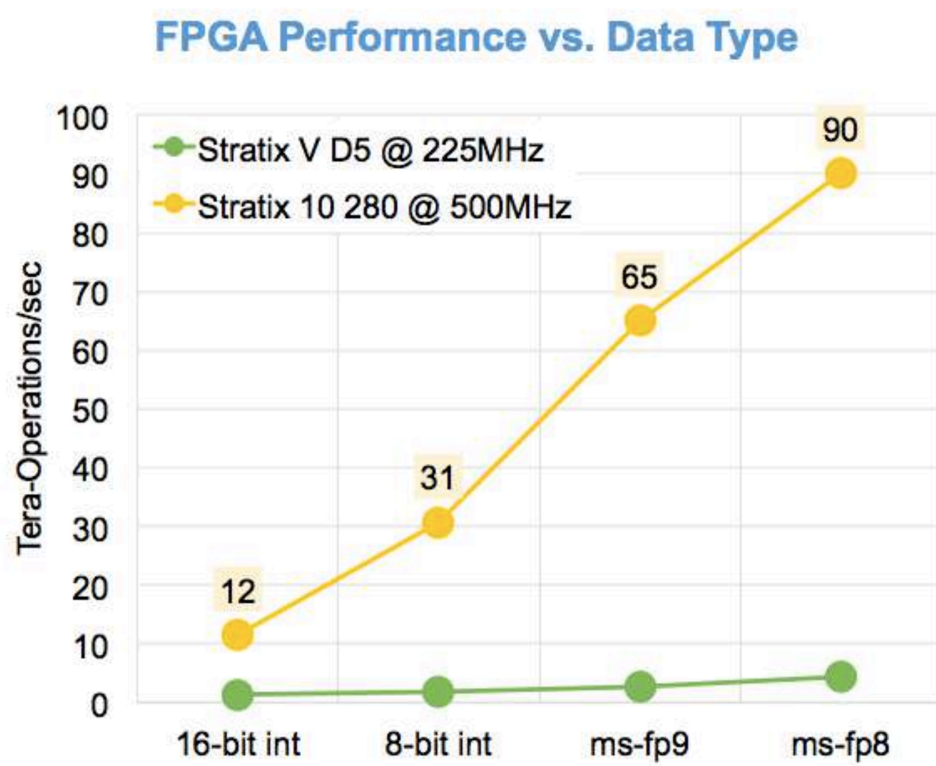
Intel's NNP-L (2019)

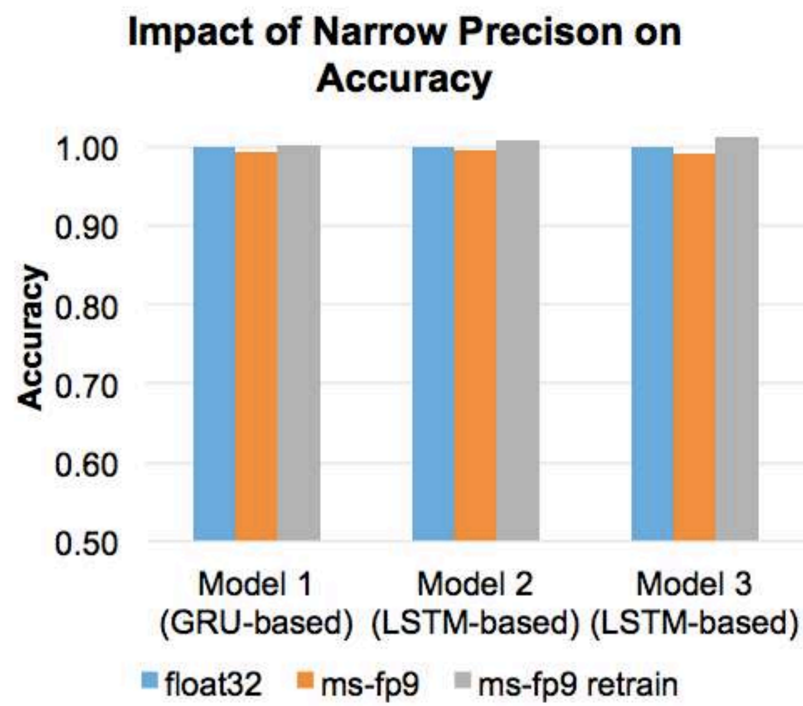
8-bit Inference & bfloat16 for Training



Microsoft BrainWave

Narrow Precision for Inference





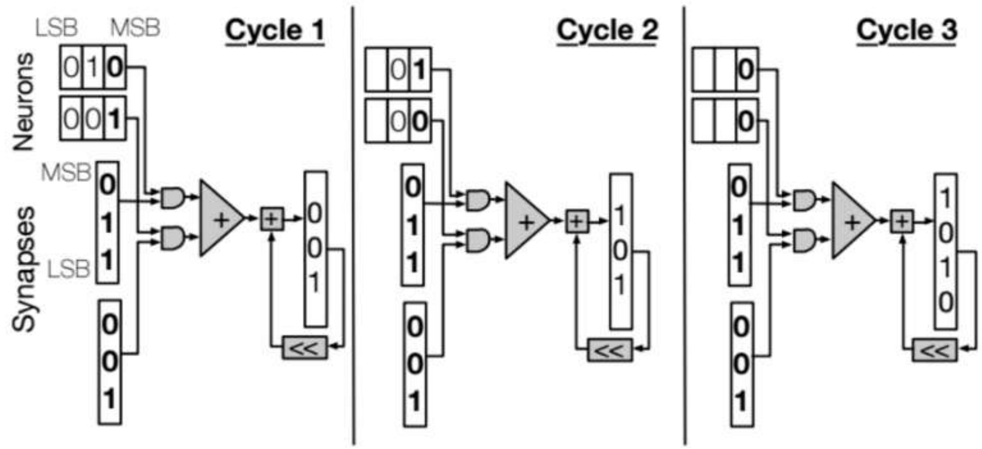
Custom 8-bit floating point format ("ms-fp8")

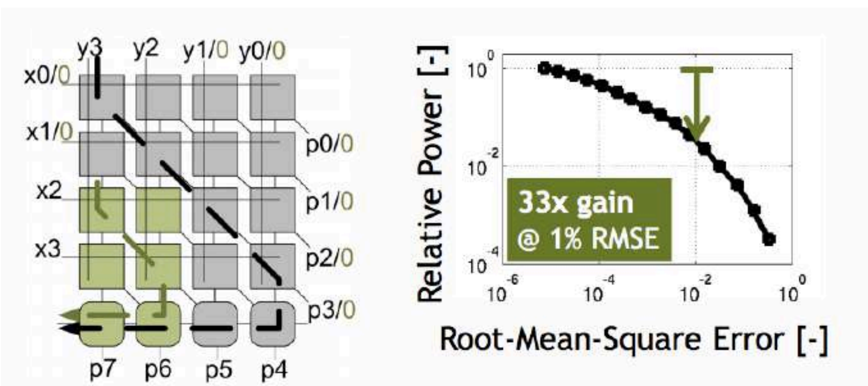
Reduced Precision Hardware

Stripes

[Judd et al., MICRO 2016]

Bit-serial processing for speed





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

KU Leuven

[Moons et al., VLSI 2016]

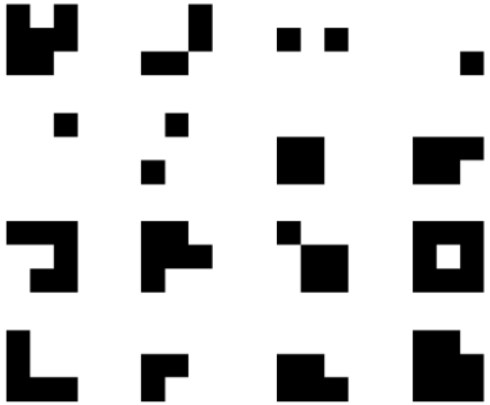
Voltage scaling for energy savings

Binary Nets

- **Binary Connect (BC)**
  - Weights {-1,1}, Activations 32-bit float
  - MAC → addition/subtraction
  - Accuracy loss: **19%** on AlexNet

[Courbariaux, NeurIPS 2015]

Binary Filters



- **Binarized Neural Networks (BNN)**
  - Weights {-1,1}, Activations {-1,1}
  - MAC → XNOR
  - Accuracy loss: **29.8%** on AlexNet

[Courbariaux, arXiv 2016]

Scale the Weights and Activations

- **Binary Weight Nets (BWN)**
  - Weights {-α, α} → except first and last layers are 32-bit float
  - Activations: 32-bit float
  - α determined by the l_1-norm of all weights in a filter
  - Accuracy loss: **0.8%** on AlexNet
- **XNOR-Net**
  - Weights {-α, α}
  - Activations $\{-\beta_i, \beta_i\}$ → except first and last layers are 32-bit float
  - β_i determined by the l_1-norm of all activations across channels ***for given position i*** of the input feature map
  - Accuracy loss: **11%** on AlexNet

Hardware needs to support both activation precisions

Scale factors (α, β_i) can change per filter or position in filter

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

Ternary Nets

- **Allow for weights to be zero**
  - Increase sparsity, but also increase number of bits (2-bits)

- **Ternary Weight Nets (TWN)** [Li et al., arXiv 2016]
  - Weights {-w, 0, w} → except first and last layers are 32-bit float
  - Activations: 32-bit float
  - Accuracy loss: 3.7% on AlexNet

- **Trained Ternary Quantization (TTQ)** [Zhu et al., ICLR 2017]
  - Weights $\{-w_1, 0, w_2\}$ → except first and last layers are 32-bit float
  - Activations: 32-bit float
  - Accuracy loss: 0.6% on AlexNet

Non-Linear Quantization

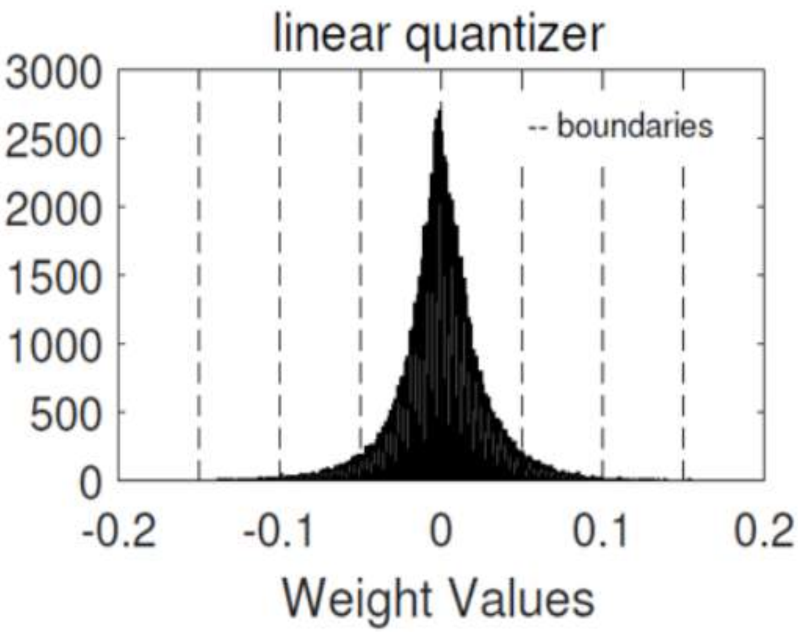
- **Precision** refers to the **number of levels**
  - Number of bits = $\log_2$ (number of levels)

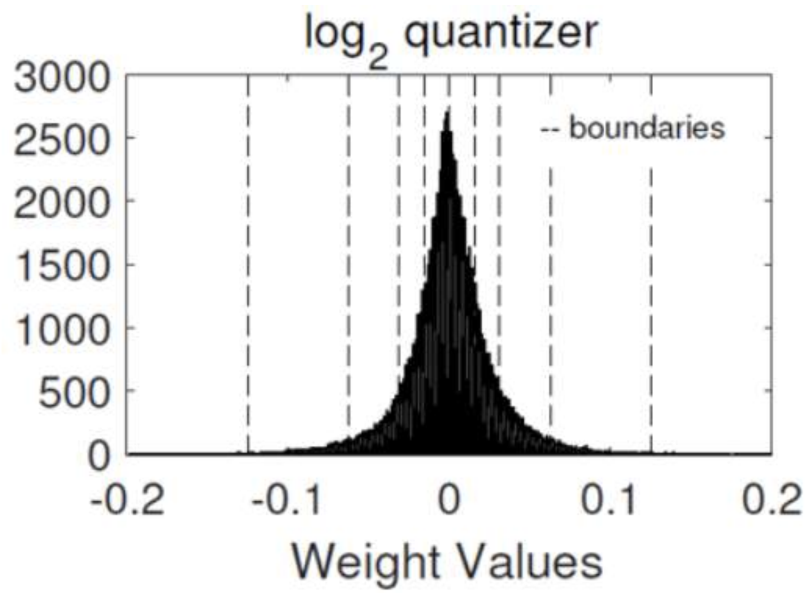
- **Quantization:** mapping data to a smaller set of **levels**
  - Linear, e.g., fixed-point
  - Non-linear
    - Computed
    - Table lookup

Objective: Reduce size to improve speed and/or reduce energy while preserving accuracy

Computed Non-linear Quantization

Log Domain Quantization





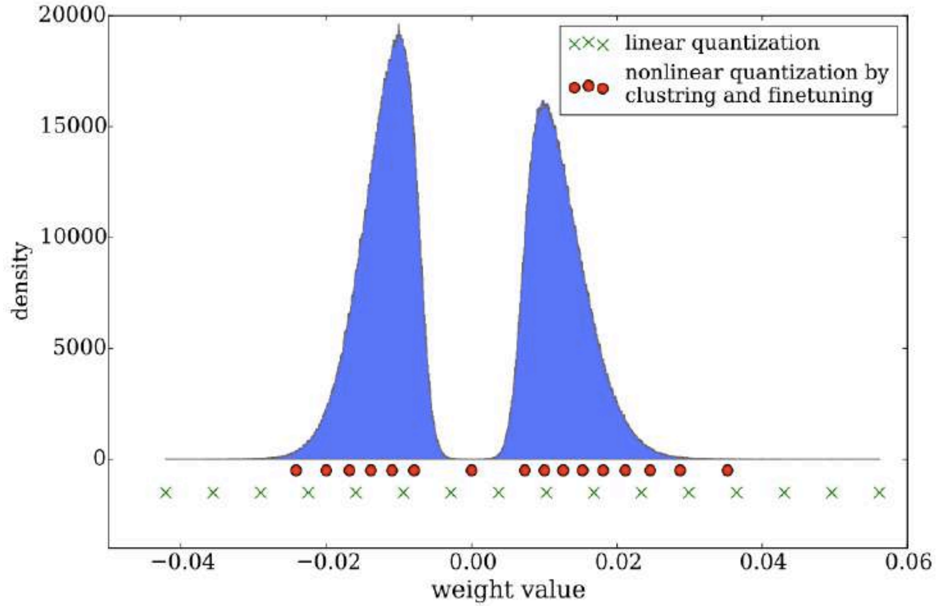
Product = X * W

Product = X << W

[Lee et al., LogNet, ICASSP 2017]

Reduce Precision Overview

- Learned mapping of data to quantization levels (e.g., k-means)



Implement with look up table

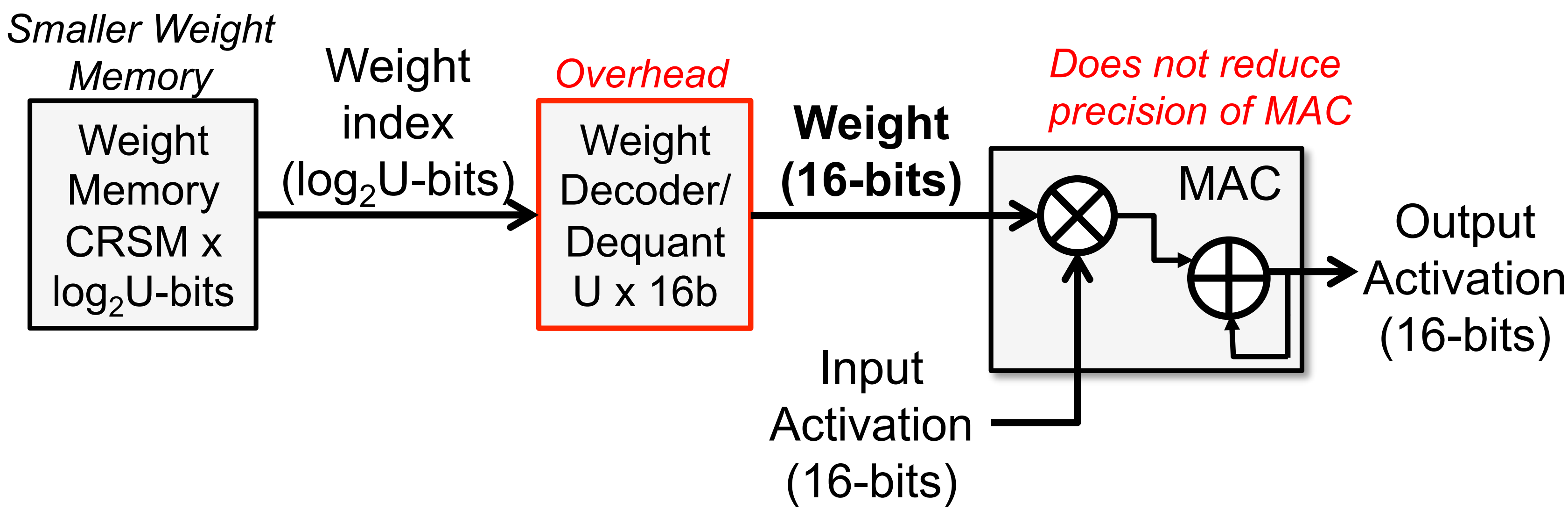
[Han et al., ICLR 2016]

- Additional Properties
  - Fixed or Variable (across data types, layers, channels, etc.)

Non-Linear Quantization Table Lookup

Trained Quantization: Find K weights via K-means clustering to reduce number of unique weights ***per layer*** (weight sharing)

Example: AlexNet (no accuracy loss)
256 unique weights for CONV layer
16 unique weights for FC layer



Consequences: Narrow weight memory and second access from (small) table

Summary of Reduce Precision

Category	Method	Weights (# of bits)	Activations (# of bits)	Accuracy Loss vs. 32-bit float (%)
Dynamic Fixed Point	w/o fine-tuning	8	10	0.4
	w/ fine-tuning	8	8	0.6
Reduce weight	Ternary weights Networks (TWN)	2*	32	3.7
	Trained Ternary Quantization (TTQ)	2*	32	0.6
	Binary Connect (BC)	1	32	19.2
	Binary Weight Net (BWN)	1*	32	0.8
Reduce weight and activation	Binarized Neural Net (BNN)	1	1	29.8
	XNOR-Net	1*	1	11
Non-Linear	LogNet	5(conv), 4(fc)	4	3.2
	Weight Sharing	8(conv), 4(fc)	16	0

* first and last layers are 32-bit float

Approaches

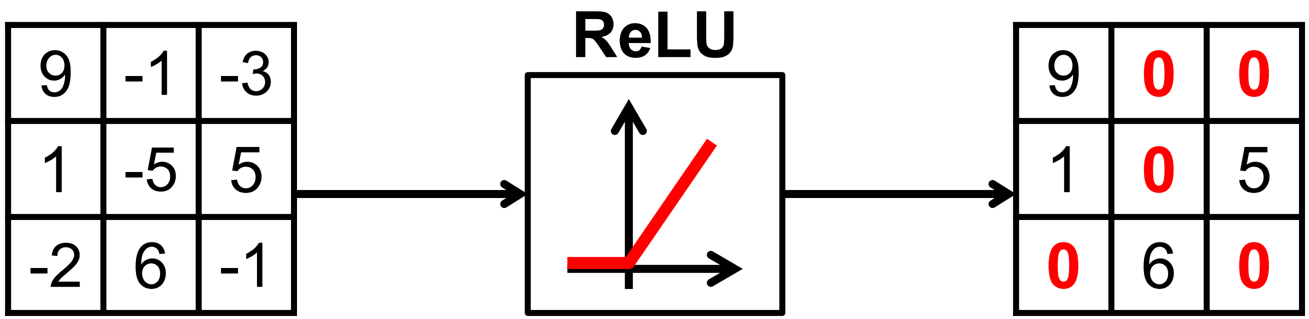
- **Reduce size of operands for storage/compute**
  - Floating point → Fixed point
  - Bit-width reduction
  - Non-linear quantization

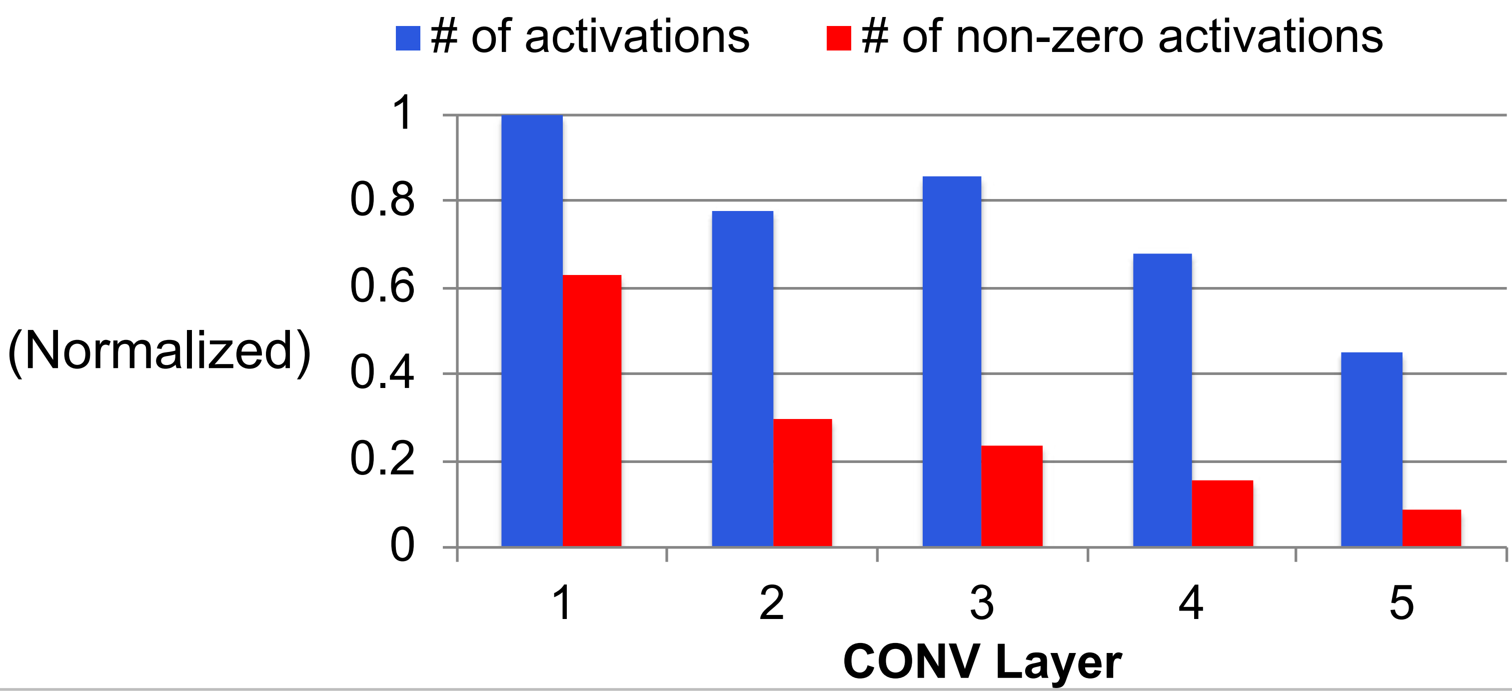
- **Reduce number of operations for storage/compute**
  - **Exploit Activation Statistics (Compression)**
  - **Network Pruning**
  - **Compact Network Architectures**

Exploit Sparsity

Sparsity in Feature Maps

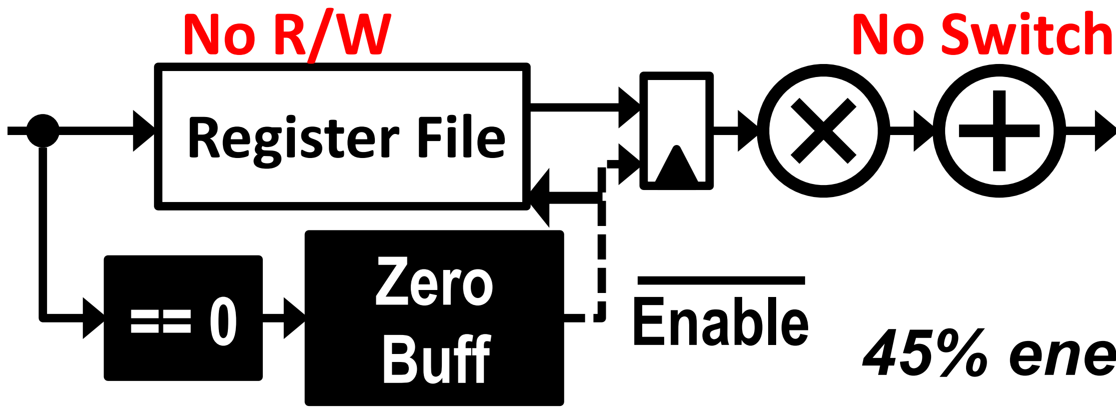
Many **zeros** in **output fmaps** after **ReLU**





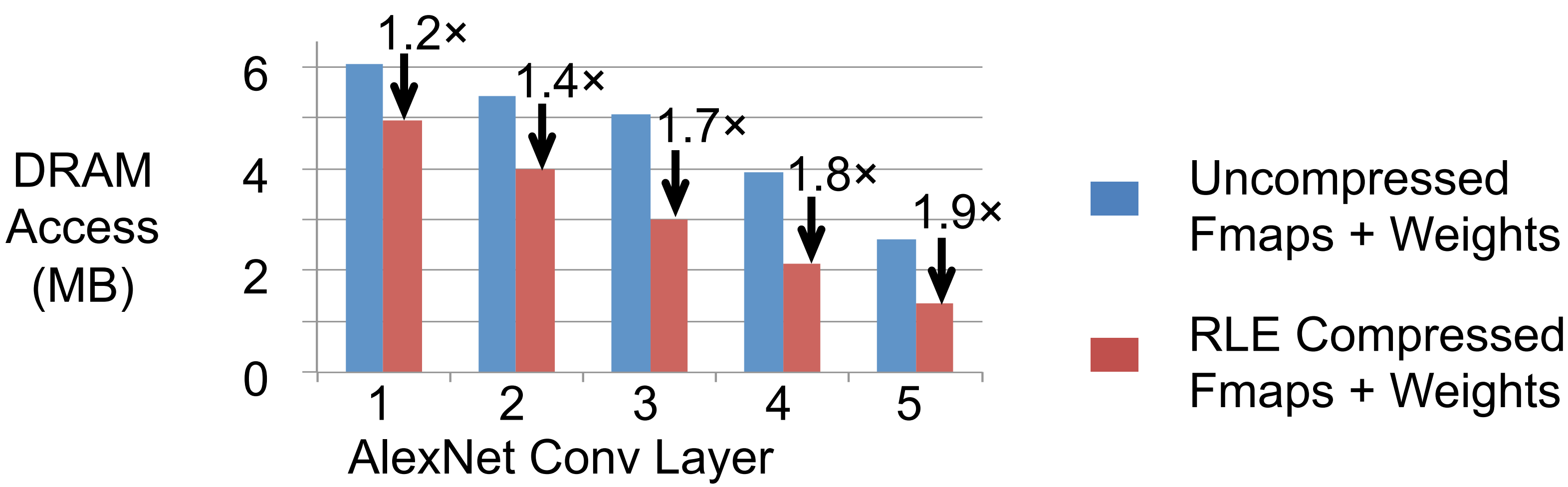
Exploit Sparsity

Method 1: Skip memory access and computation



45% energy savings

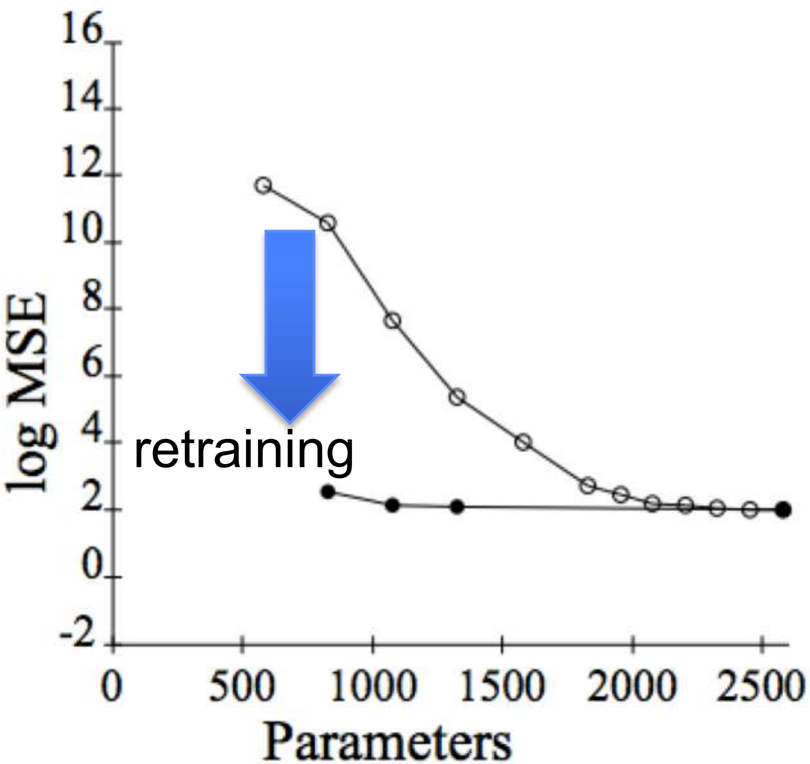
Method 2: Compress data to reduce storage and data movement



Pruning – Make Weights Sparse

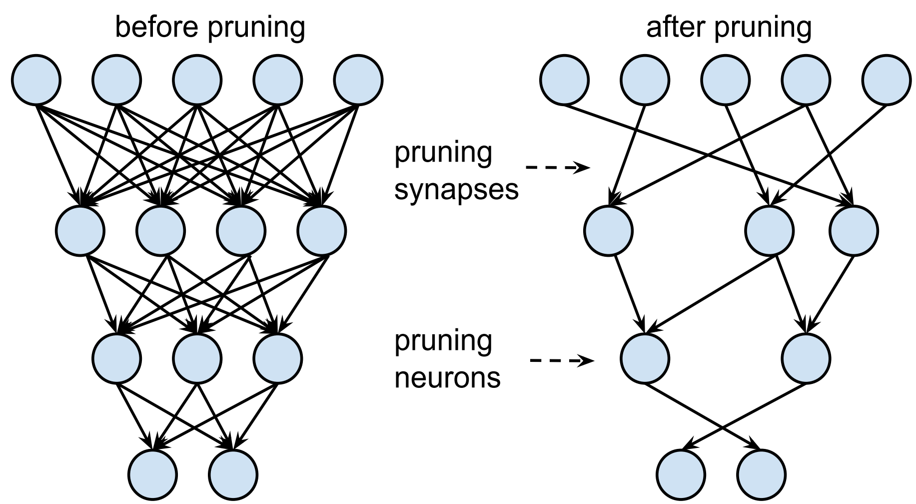
Optimal Brain Damage

[Lecun et al., NeurIPS 1989]



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

[Han et al., NeurIPS 2015]



Example: *AlexNet*

Weight Reduction:

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

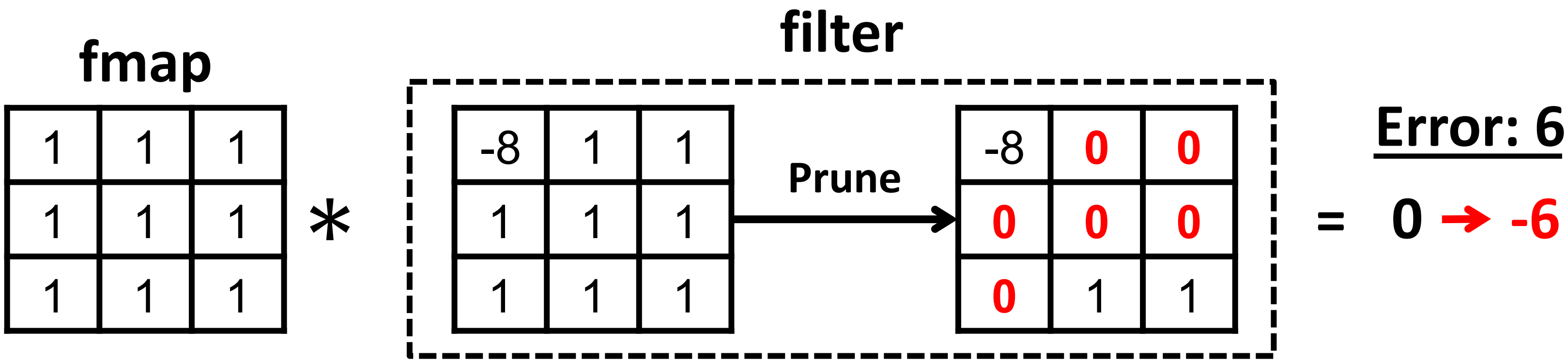
Overall Reduction:

Weights 9x, MACs 3x

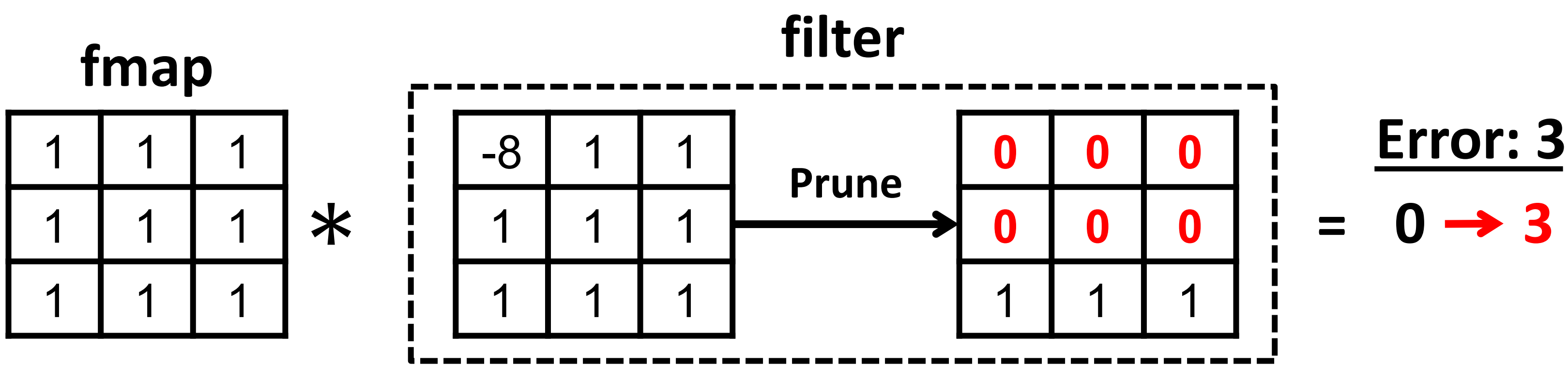
Pruning – Make Weights Sparse

Remove the weights with the **smallest joint impact** on the output feature map instead of that with the **smallest magnitude**

Magnitude-Based Method



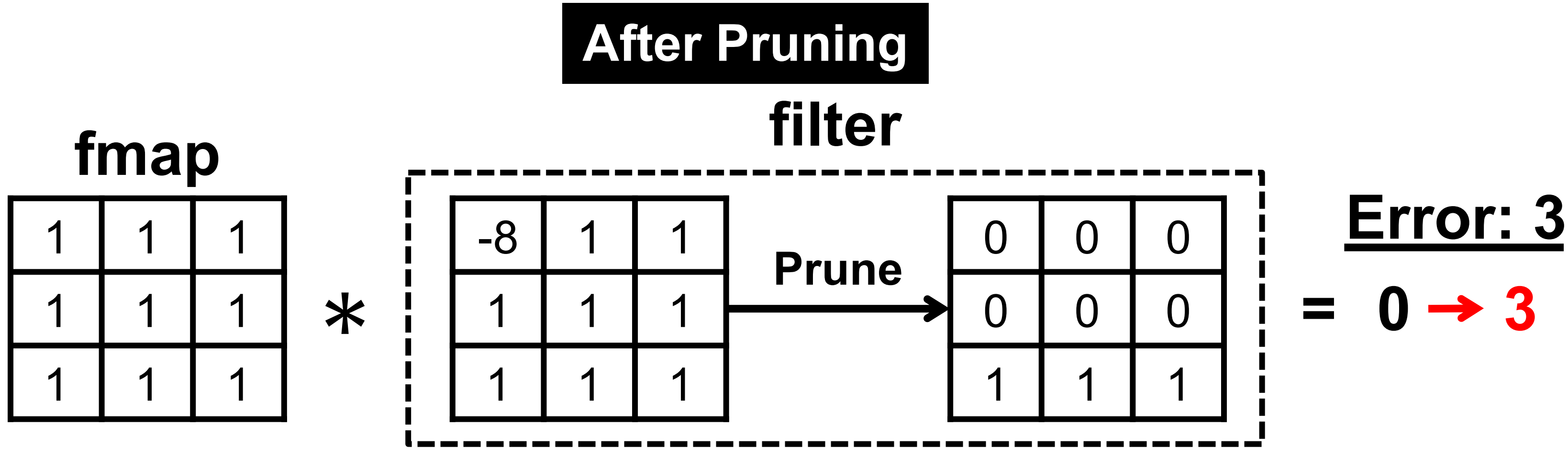
Feature-Map-Based Method

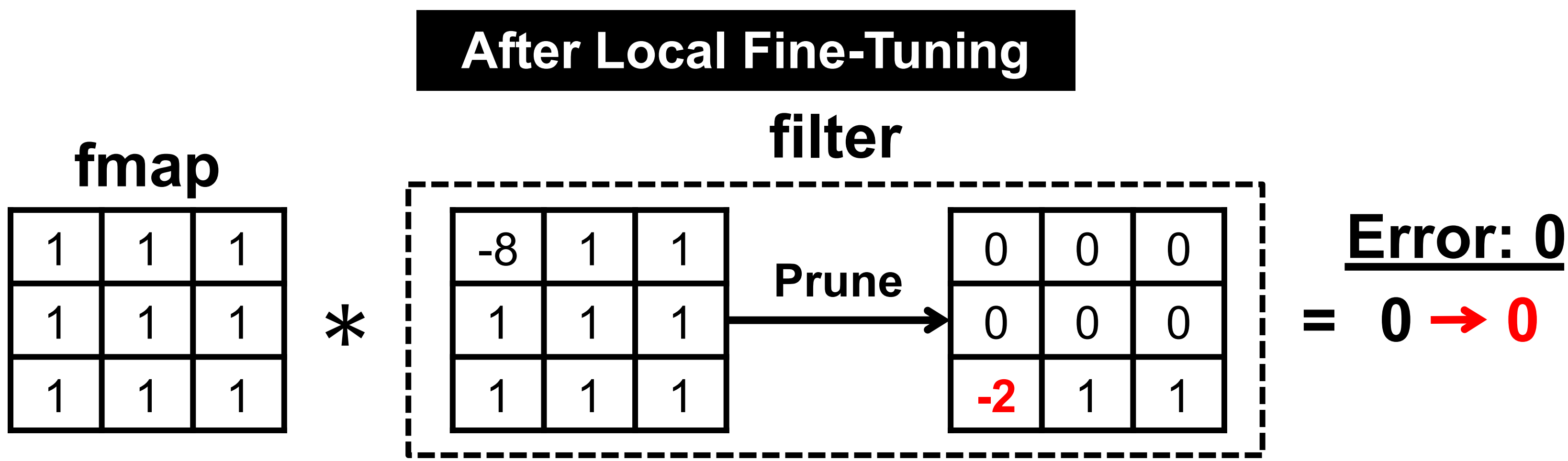


[Yang et al., Energy-Aware Pruning, CVPR 2017]

Fast Local Fine-Tuning

We then **locally fine-tune** the remaining weights, which is much faster than performing end-to-end training





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

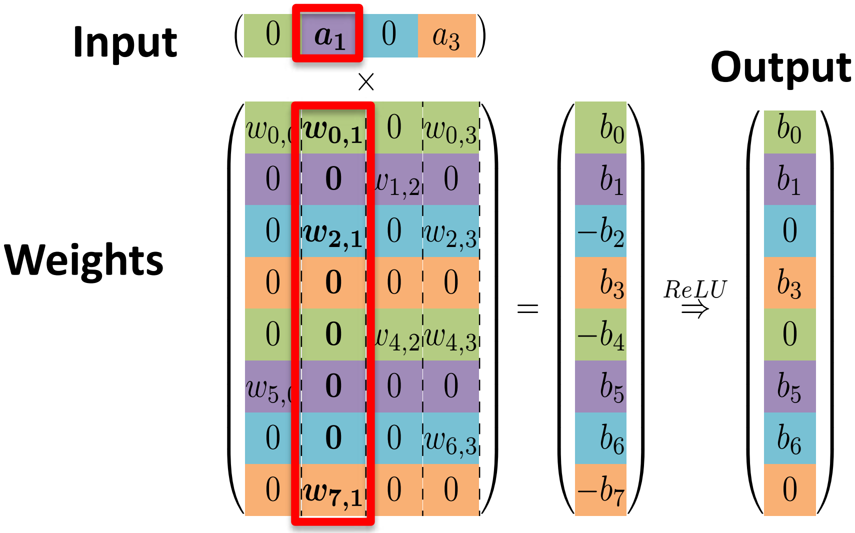
Layer	Filter / Image bits (0%)	Filter / Image BW Reduc.	IO / HuffIO (MB/frame)	Voltage (V)	MMACs/ Frame	Power (mW)	Real (TOPS/W)
General CNN	16 (0%) / 16 (0%)	1.0x		1.1	–	**288**	**0.3**
AlexNet l1	7 (21%) / 4 (29%)	1.17x / 1.3x	1 / 0.77	0.85	105	85	0.96
AlexNet l2	7 (19%) / 7 (89%)	1.15x / **5.8x**	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**

Sparse Hardware

EIE

[Han et al., ISCA 2016]

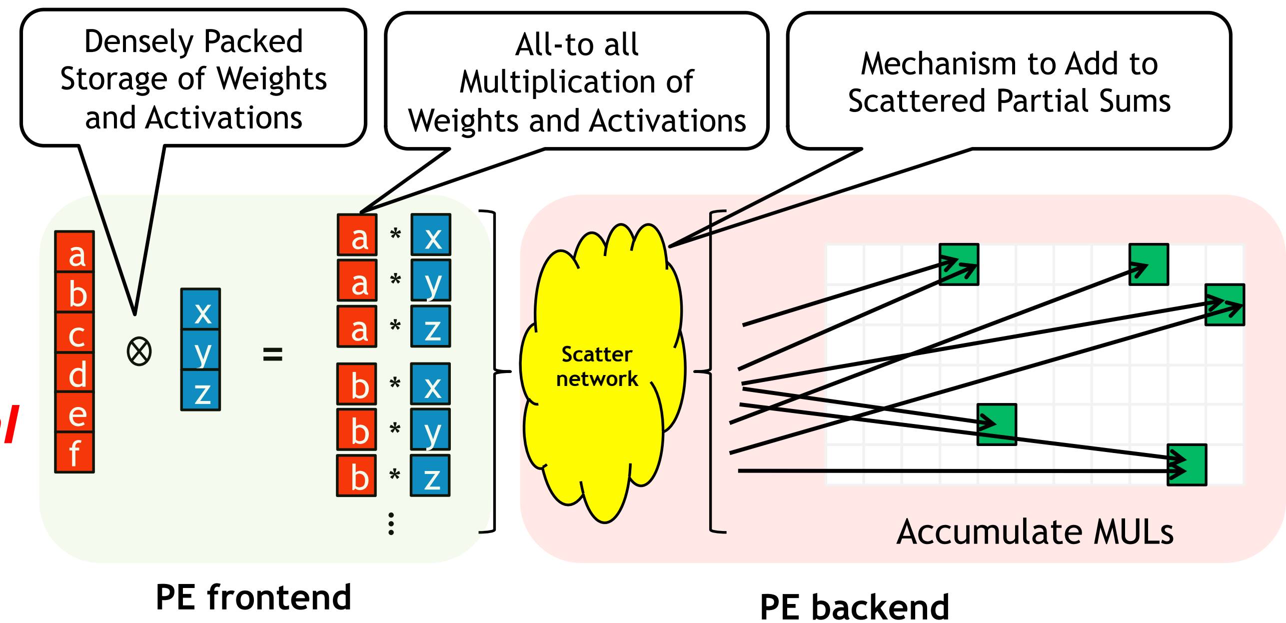
Supports Fully Connected Layers Only



SCNN

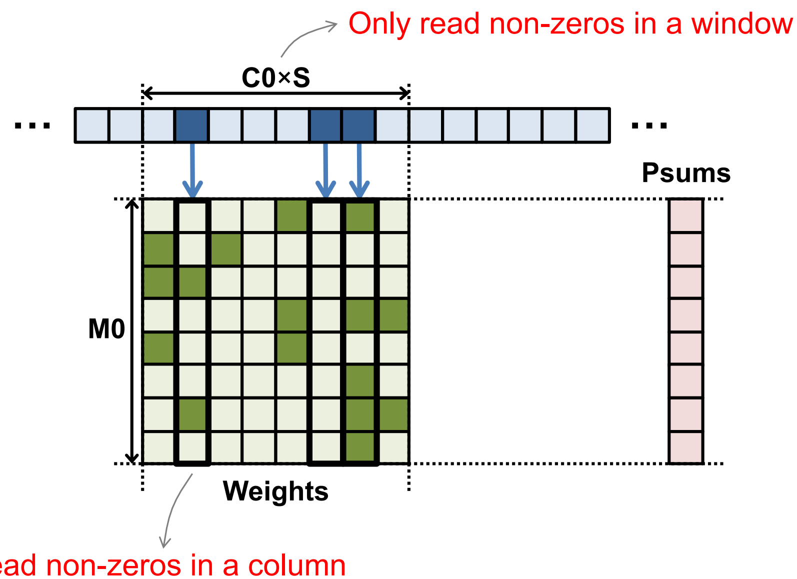
[Parashar et al., ISCA 2017]

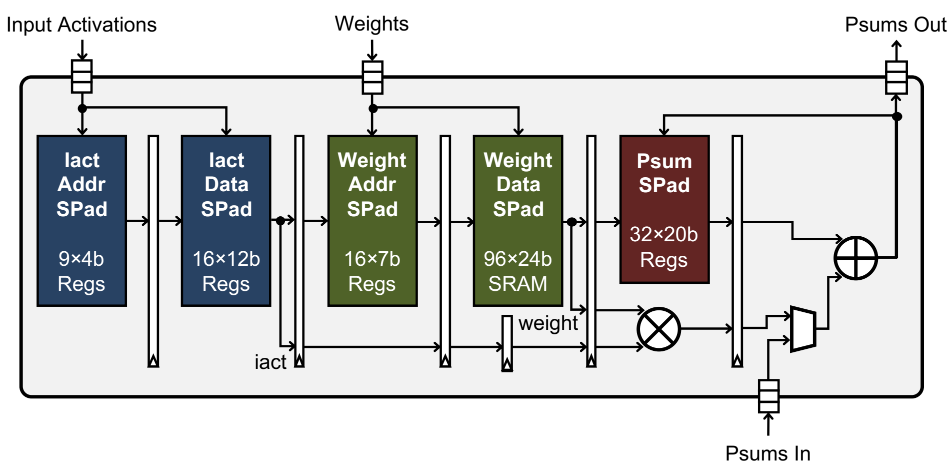
Supports Convolutional Layers Only



Sparse Hardware – Eyeriss v2

Supports both Convolutional and Fully Connected Layers



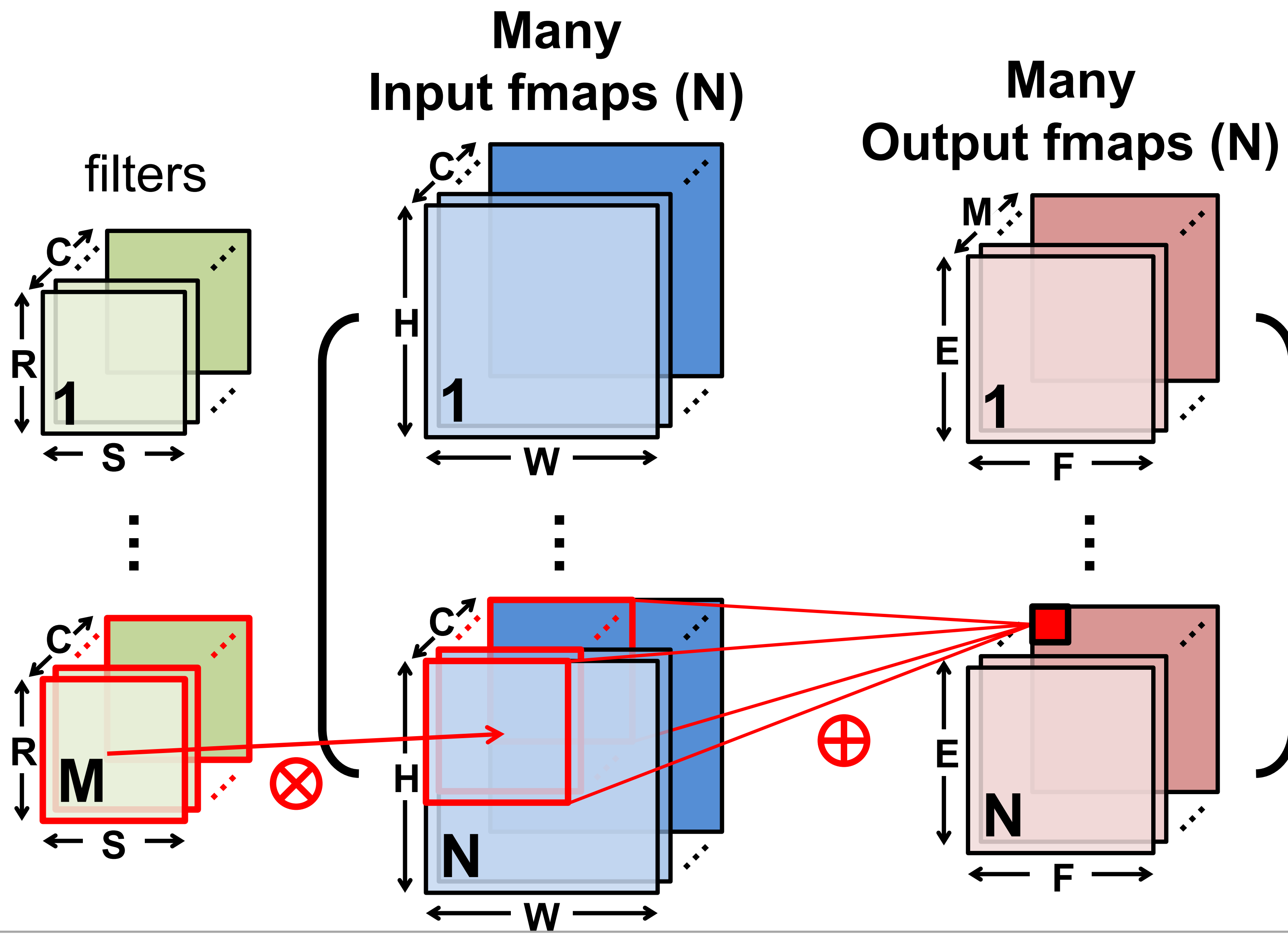


	AlexNet	sparse-AlexNet
GOPS	148.3	405.8
fps	102.4	280.1
Over v1	**15.5×**	**42.5×**
GOPS/W	277.9	1028.1
Inferences/J	191.8	709.7
Over v1	**3.0×**	**11.3×**

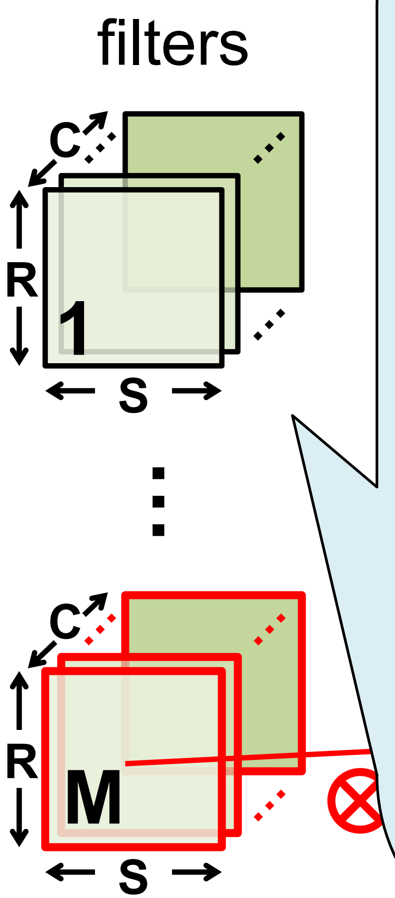
[Chen et al., JETCAS 2019]

Manual Network Architecture Design

Simplify CONV Layers



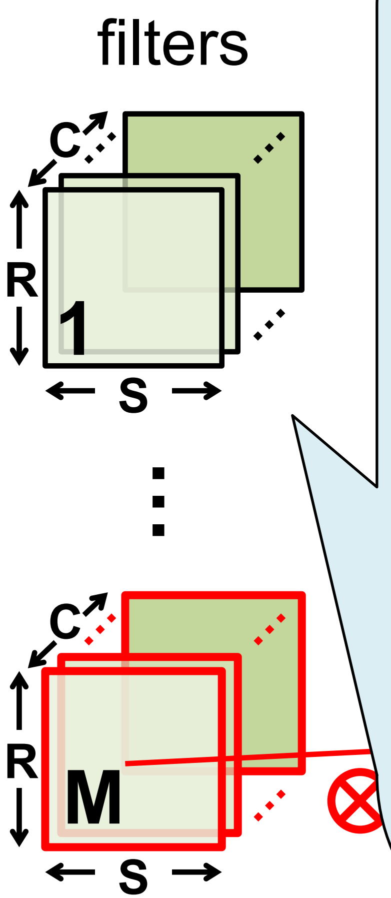
Simplify CONV Layers



Methods can be roughly categorized by how the filters are simplified:

- Reduce spatial size (R, S): stacked filters
- Reduce channels (C): 1x1 convolution, group of filters
- Reduce filters (M): feature map reuse

Simplify CONV Layers

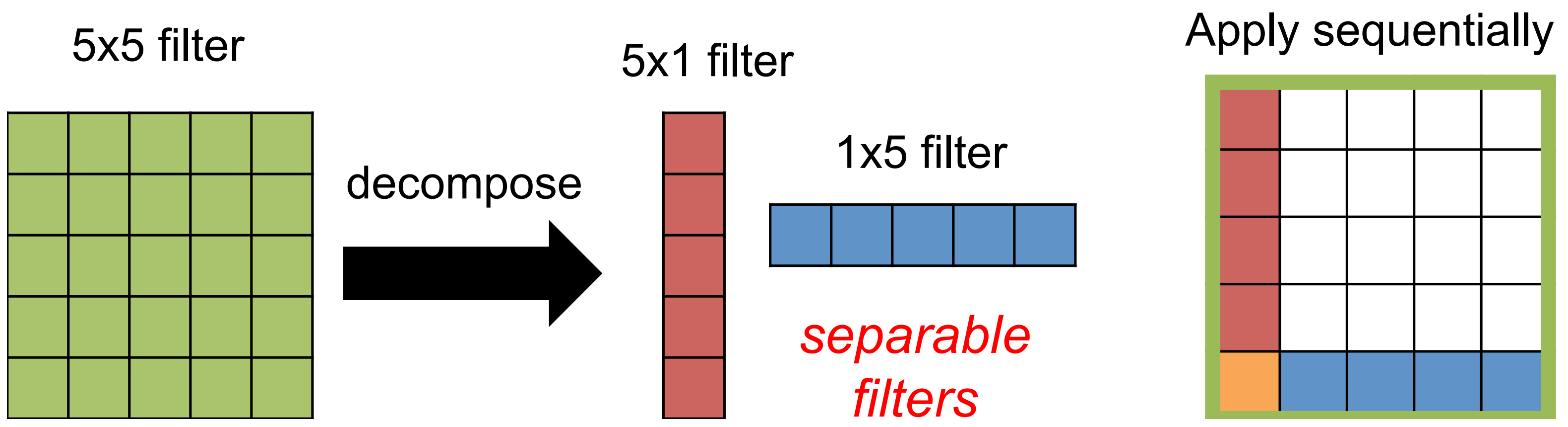


Methods can be roughly categorized by how the filters are simplified:

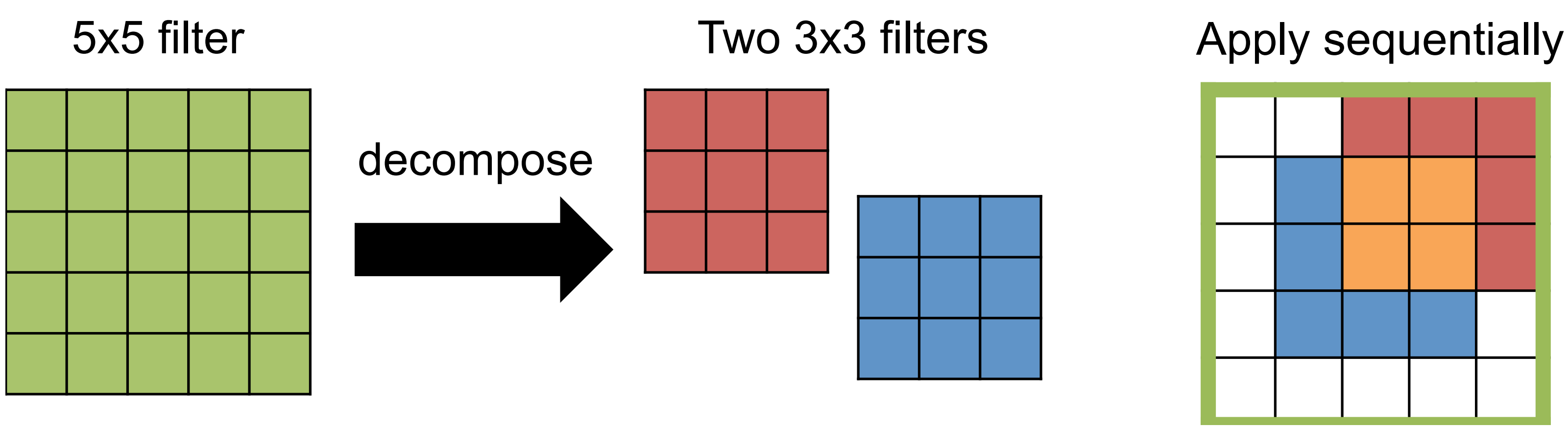
- Reduce spatial size (R, S): stacked filters
- Reduce channels (C): 1x1 convolution, group of filters
- Reduce filters (M): feature map reuse

Stacked Filters

GoogleNet/Inception v3



VGG-16

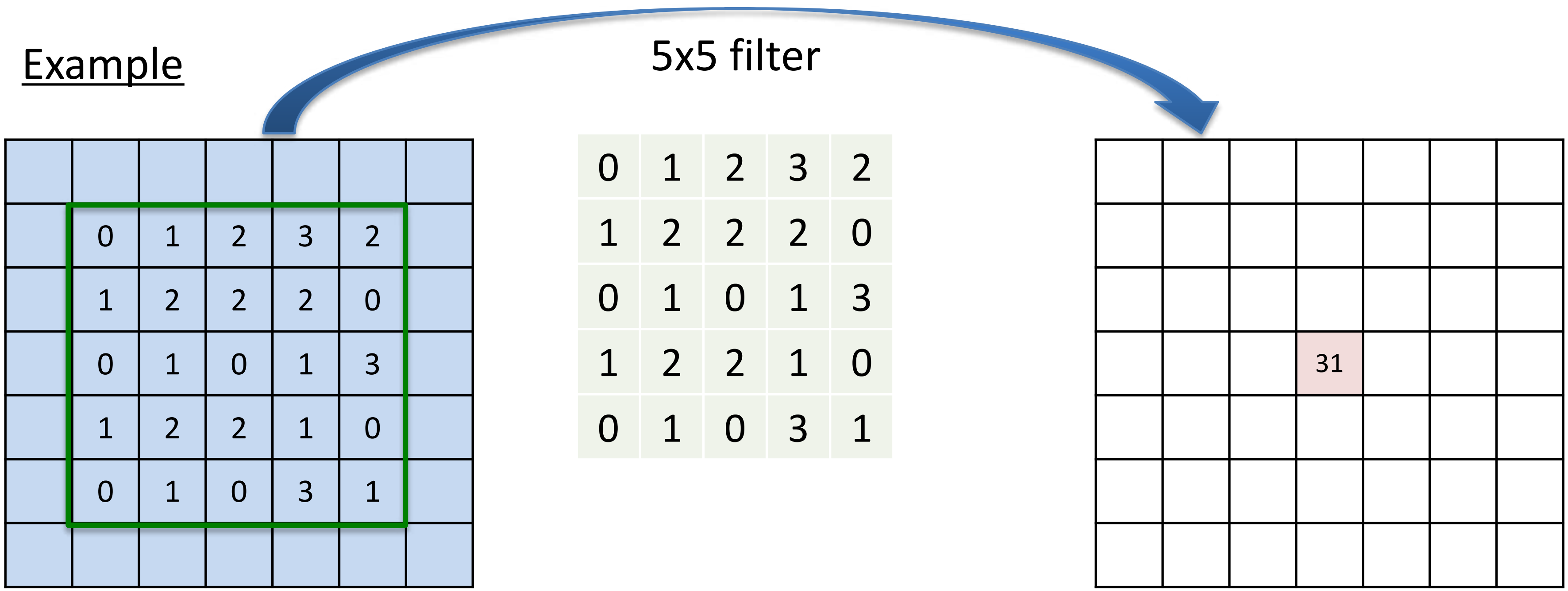


Replace a large filter with a series of smaller filters

Stacked Filters

- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights

<u>Example</u>



0	1	2	3	2
1	2	2	2	0
0	1	0	1	3
1	2	2	1	0
0	1	0	3	1

Stacked Filters

- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights

filter (3x3)

0	1	0
1	1	1
0	1	0

<u>Example</u>

0	0	0	0	0	0	0
0	0	1	2	3	2	0
0	1	2	2	2	0	0
0	0	1	0	1	3	0
0	1	2	2	1	0	0
0	0	1	0	3	1	0
0	0	0	0	0	0	0

		7	8	8		
		5	6	7		
		6	5	7		

3x3 filter$_1$

Stacked Filters

- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights

filter (3x3)

0	1	0
1	1	1
0	1	0

Example: 5x5 filter (25 weights) → two 3x3 filters (18 weights)

0	0	0	0	0	0	0
0	0	1	2	3	2	0
0	1	2	2	2	0	0
0	0	1	0	1	3	0
0	1	2	2	1	0	0
0	0	1	0	3	1	0
0	0	0	0	0	0	0

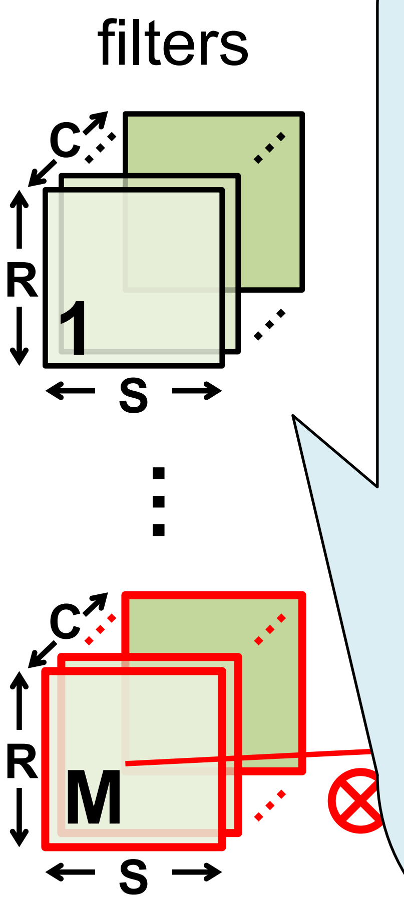
3x3 filter_1

0	0	0	0	0	0	0
0	0	1	2	3	2	0
0	1	7	8	8	0	0
0	0	5	6	7	3	0
0	1	6	5	7	0	0
0	0	1	0	3	1	0
0	0	0	0	0	0	0

3x3 filter_2

			31			

Simplify CONV Layers

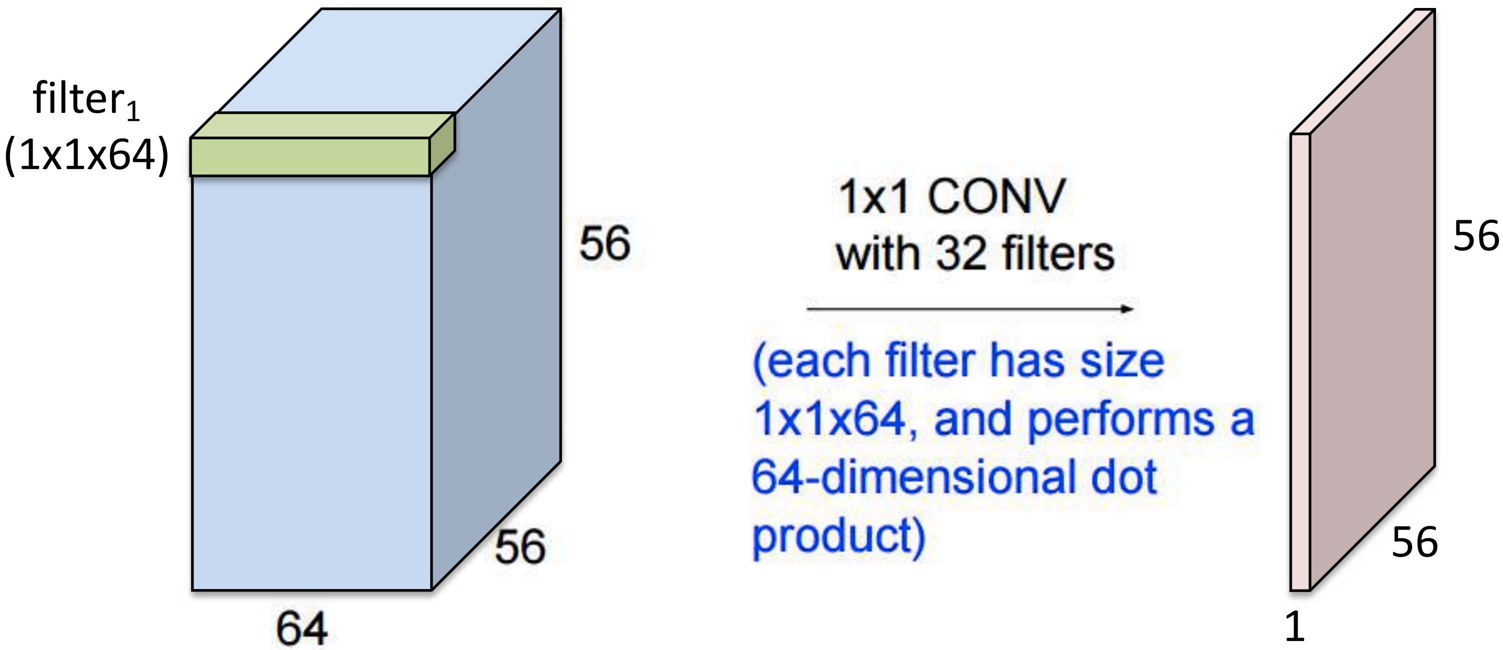


Methods can be roughly categorized by how the filters are simplified:

- Reduce spatial size (R, S): stacked filters
- Reduce channels (C): 1x1 convolution, group of filters
- Reduce filters (M): feature map reuse

1x1 Convolution

Use **1x1 filter** to condense the cross-channel information.

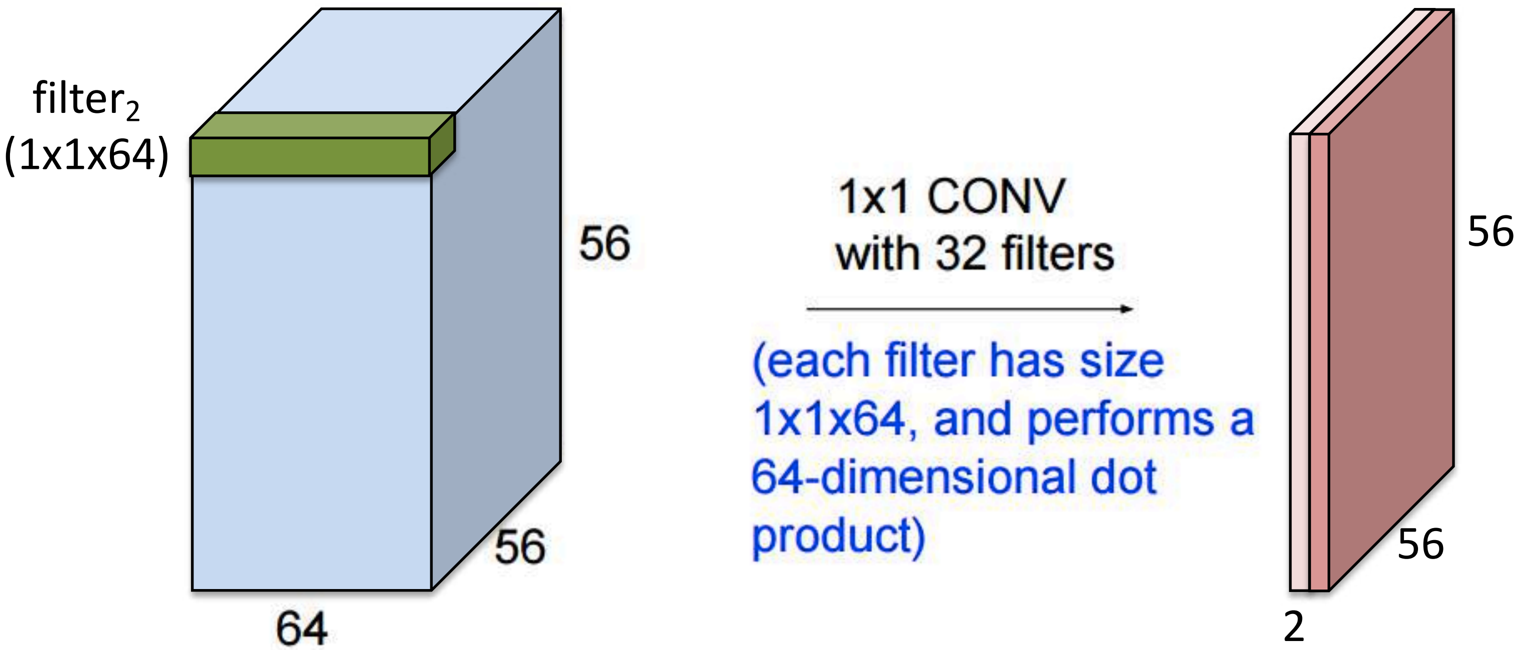


Modified image from source:
Stanford cs231n

[Lin et al., Network in Network, arXiv 2013, ICLR 2014]

1x1 Convolution

Use **1x1 filter** to condense the cross-channel information.

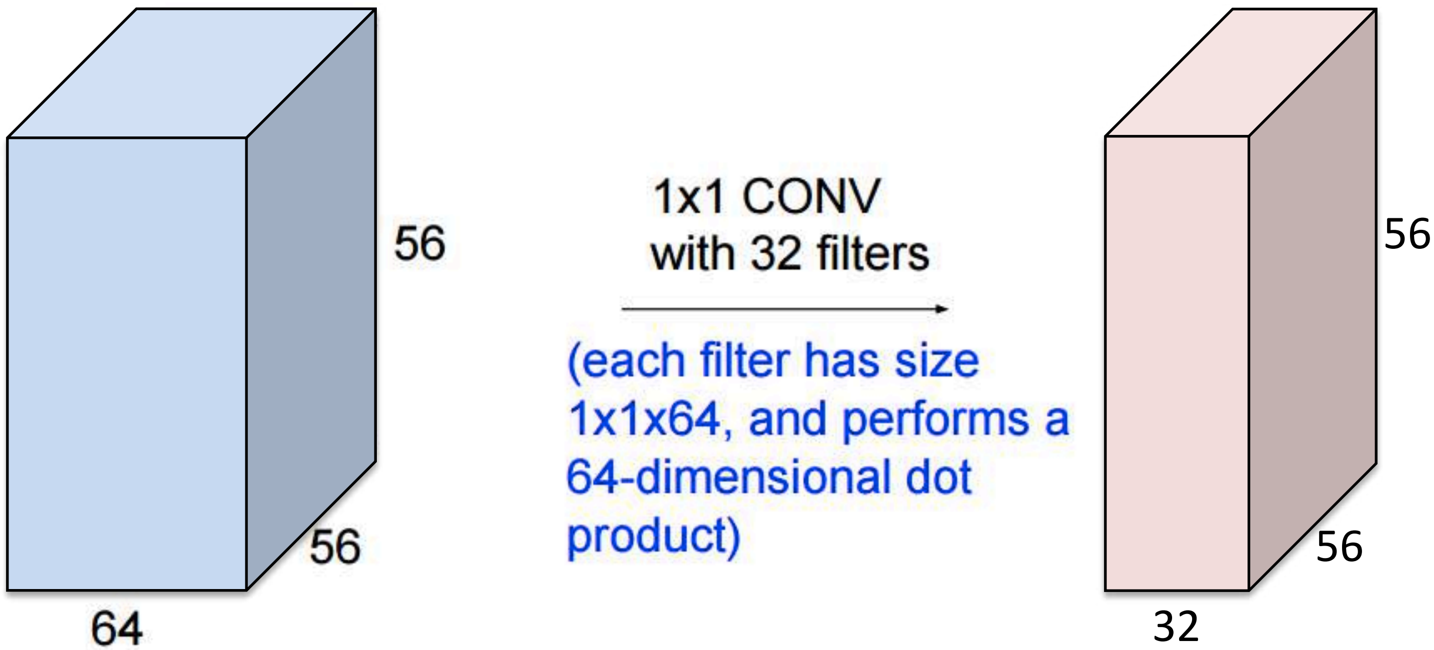


Modified image from source:
Stanford cs231n

[Lin et al., Network in Network, arXiv 2013, ICLR 2014]

1x1 Convolution

Use **1x1 filter** to condense the cross-channel information.



Modified image from source:
Stanford cs231n

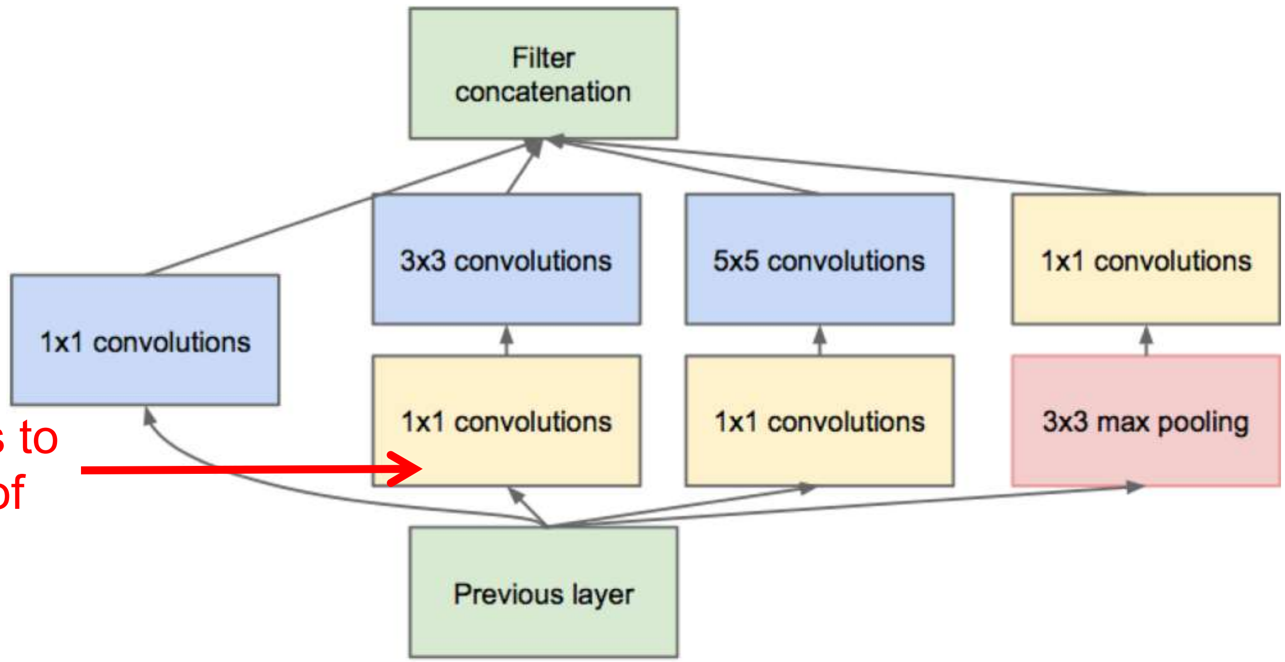
[Lin et al., Network in Network, arXiv 2013, ICLR 2014]

GoogLeNet:1x1 Convolution

Apply 1x1 convolution before 'large' convolution filters.
Reduce weights such that **entire CNN can be trained on one GPU.**
Number of multiplications reduced from 854M → 358M

Inception Module

1x1 convolutions to reduce number of weights and multiplications

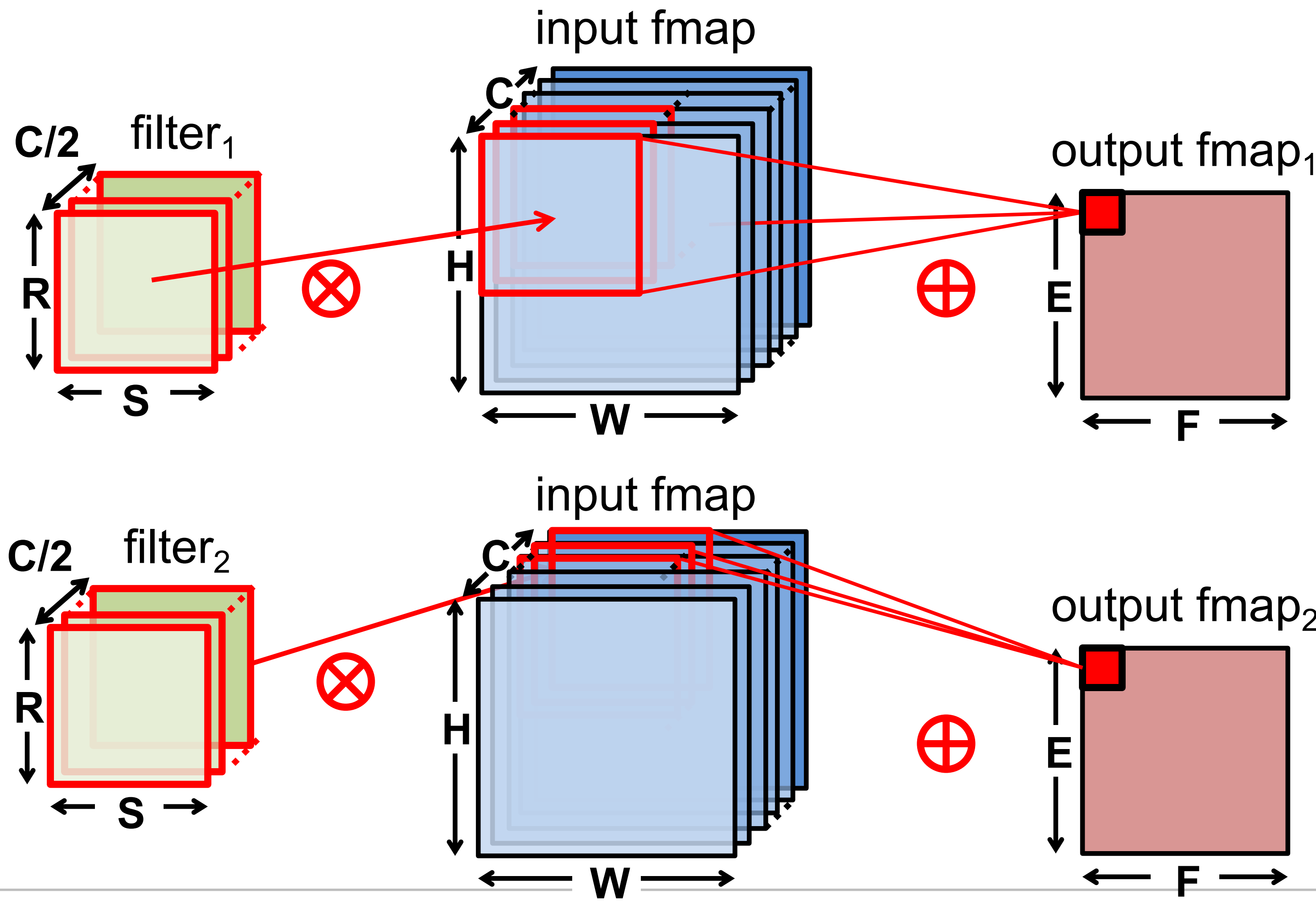


[Szegedy et al., arXiv 2014, CVPR 2015]

Group of Filters

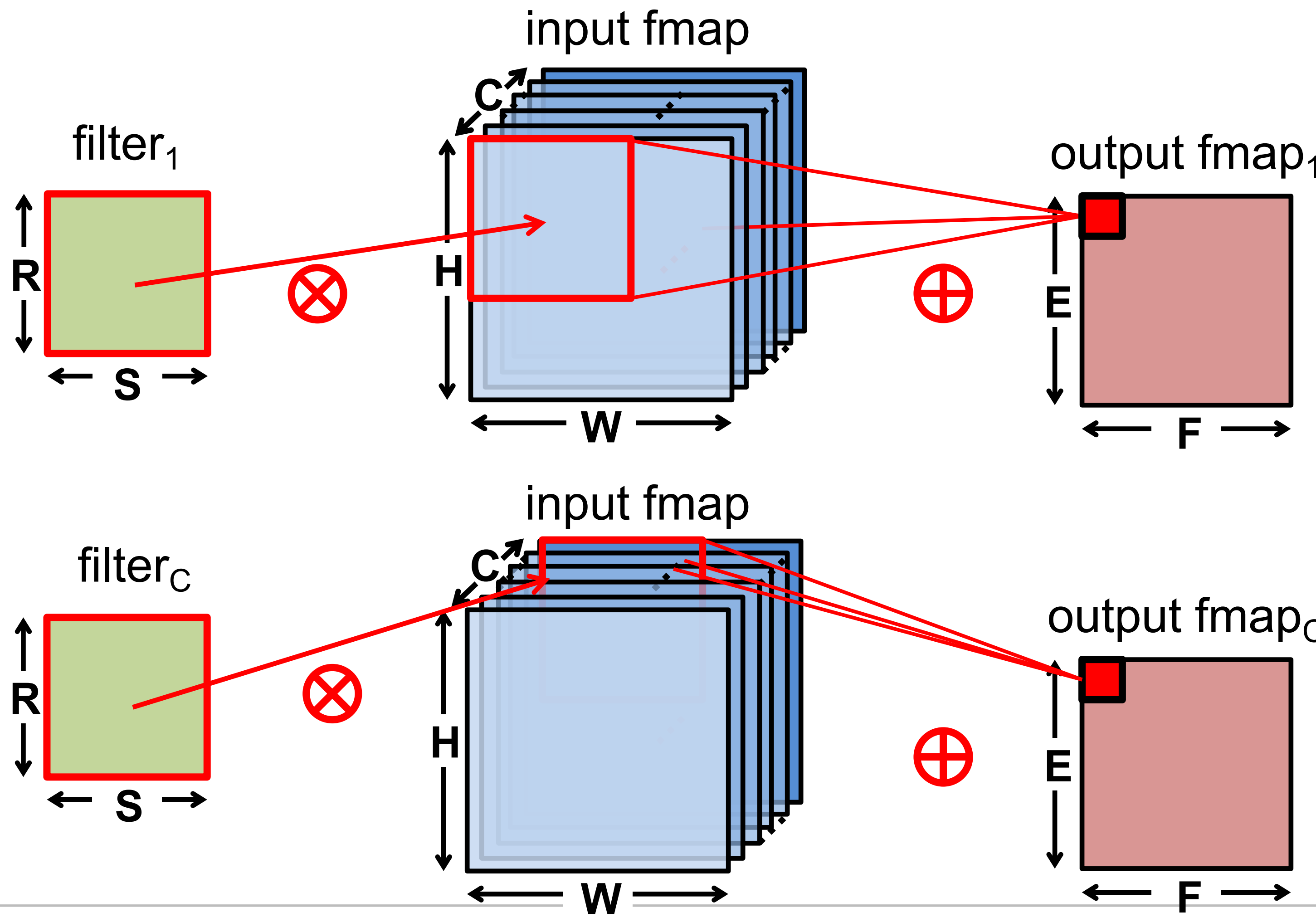
Idea: split filters and channels of feature map into different groups

Example: 2 groups, each filter requires **2x fewer weights and multiplications**.



Group of Filters

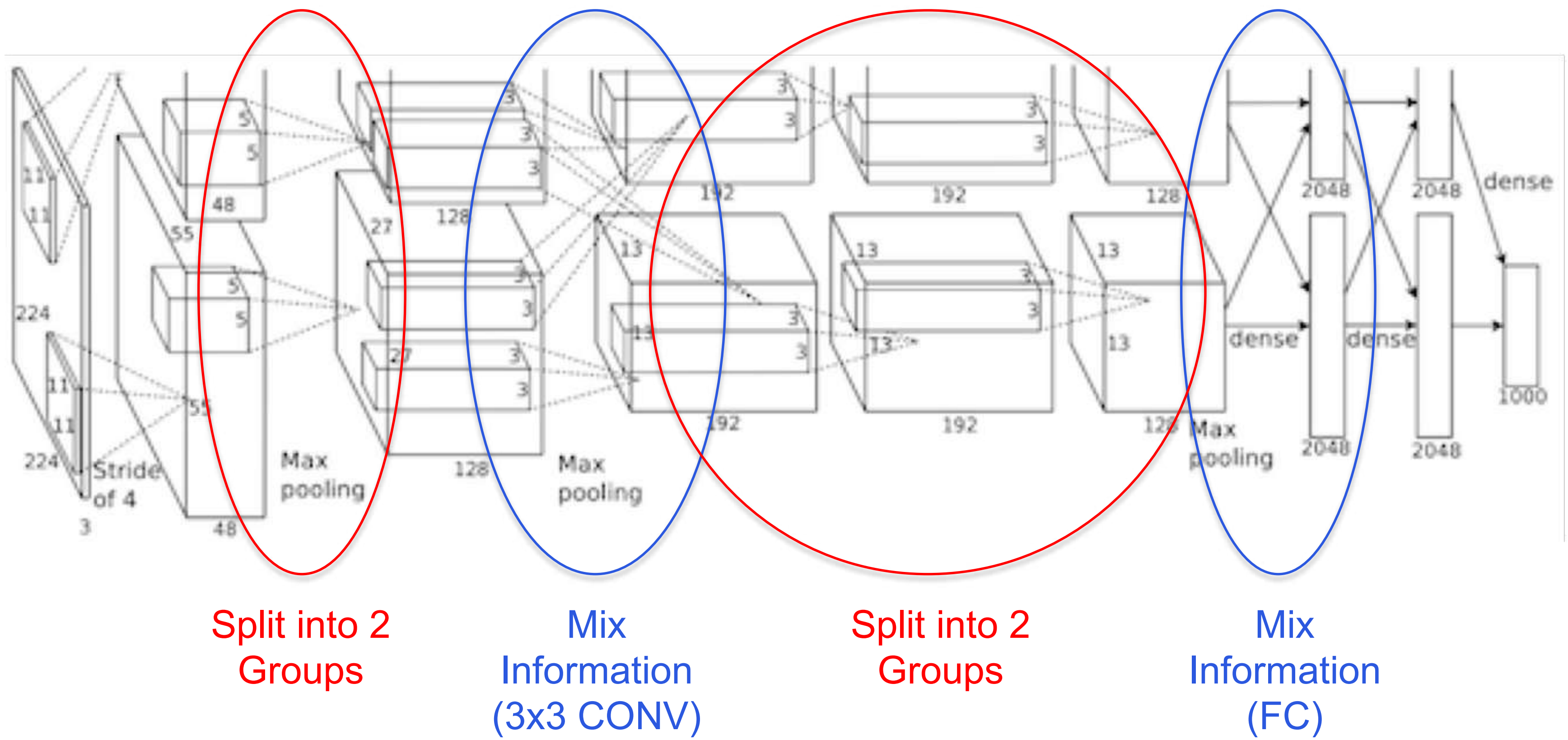
The extreme case is **depthwise convolution** – each group contains only one channel.



Group of Filters

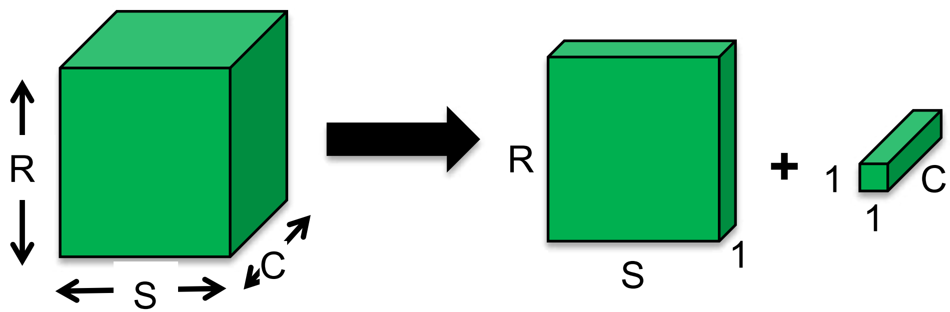
AlexNet uses group of filters to train on two separate GPUs

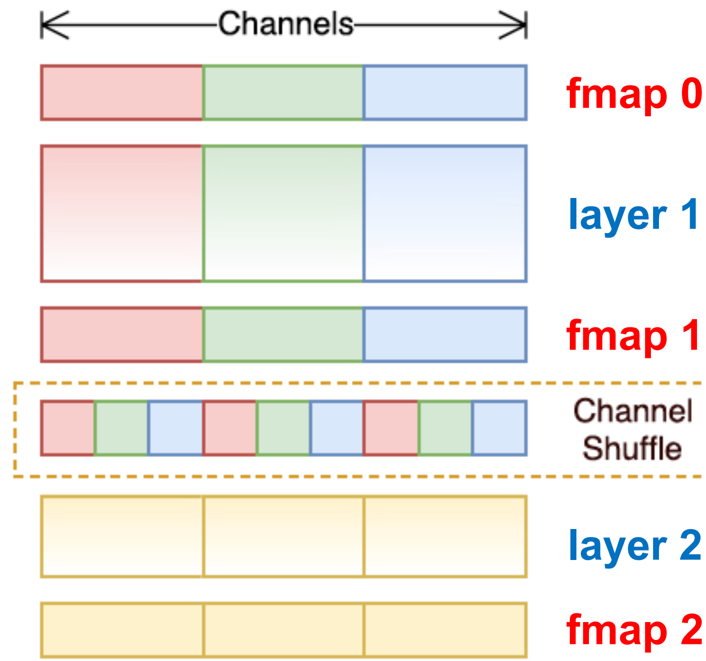
(Drawback: correlation between channels of different groups is not used)



Group of Filters

Two ways of mixing information from groups





Pointwise (1x1) Convolution
(Mix in one step)
MobileNet

Shuffle Operation
(Mix in multiple steps)
ShuffleNet

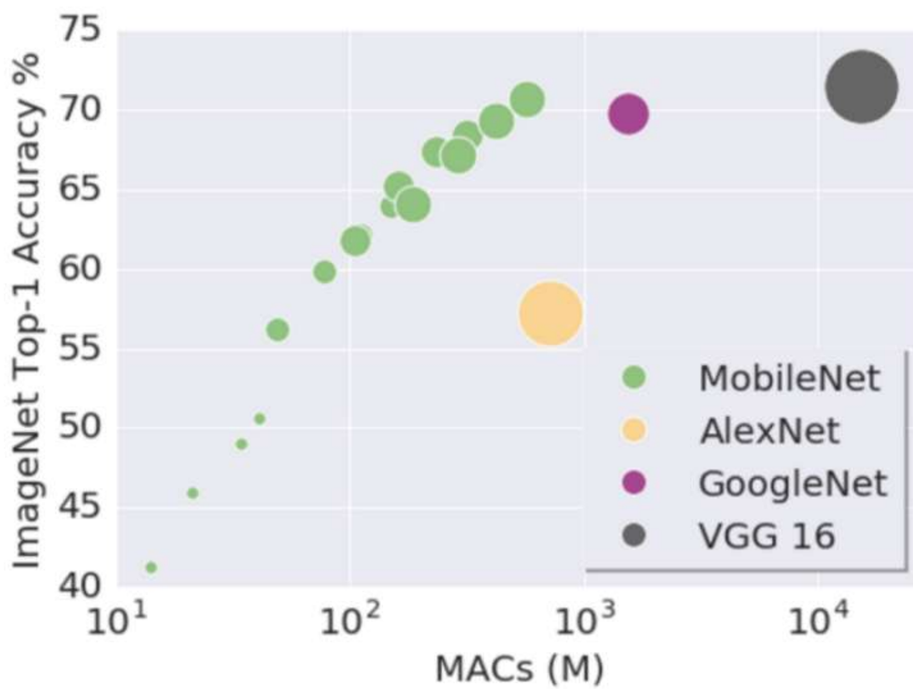
MobileNets: Comparison

Table 8. MobileNet Comparison to Popular Models

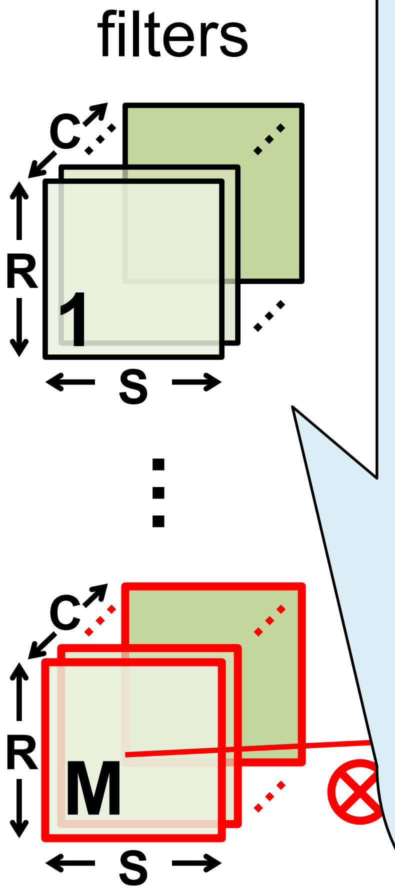
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameter
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameter
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60



Simplify CONV Layers

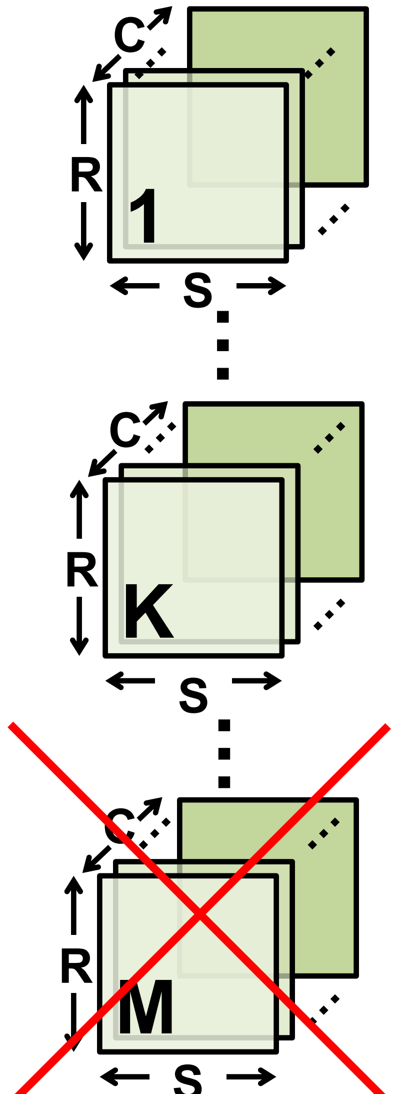


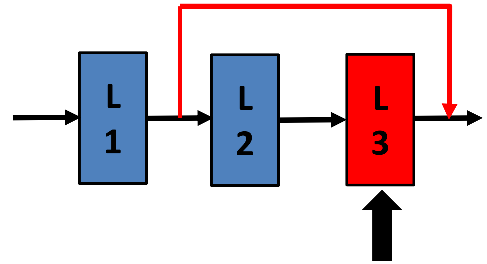
Methods can be roughly categorized by how the filters are simplified:

- Reduce spatial size (R, S): stacked filters
- Reduce channels (C): 1x1 convolution, group of filters
- Reduce filters (M): feature map reuse

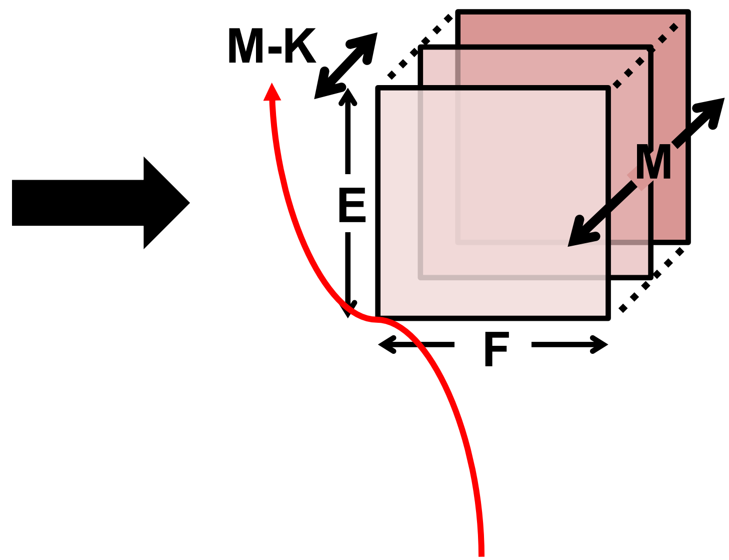
Feature Map Reuse

M Filters





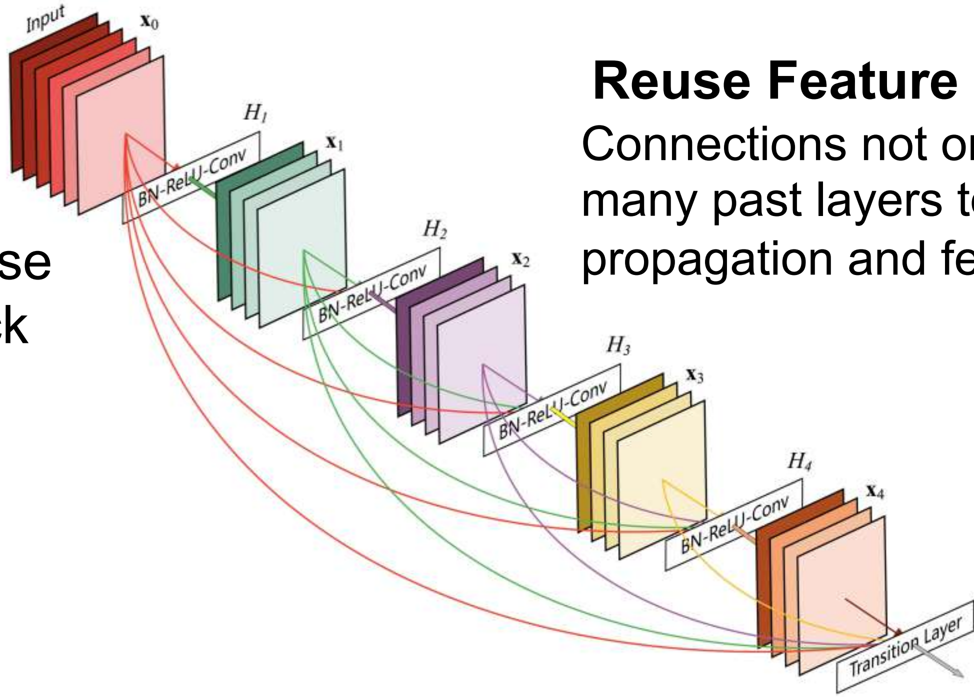
output fmap with M channels



Reuse (M-K) channels in feature maps from previously processed layers

Feature Map Reuse

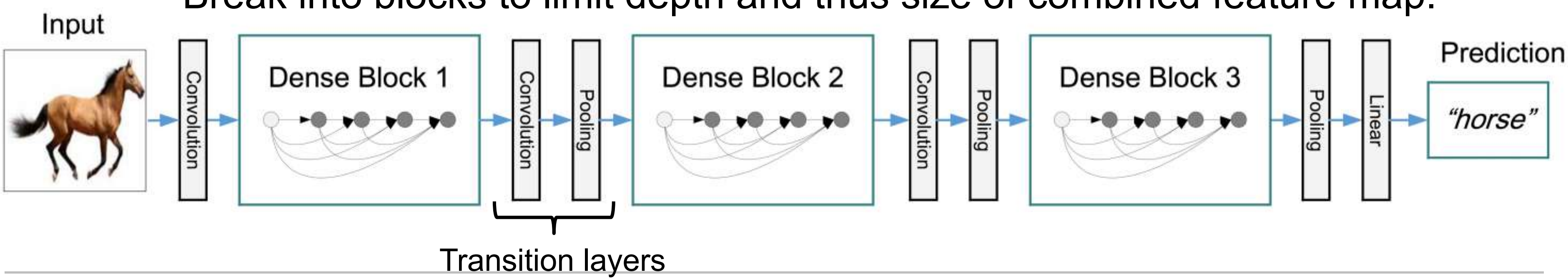
Dense
Block



Reuse Feature Maps from Multi. Layers!

Connections not only from previous layer, but many past layers to strengthen feature map propagation and feature reuse.

Feature maps are concatenated rather than added.
Break into blocks to limit depth and thus size of combined feature map.

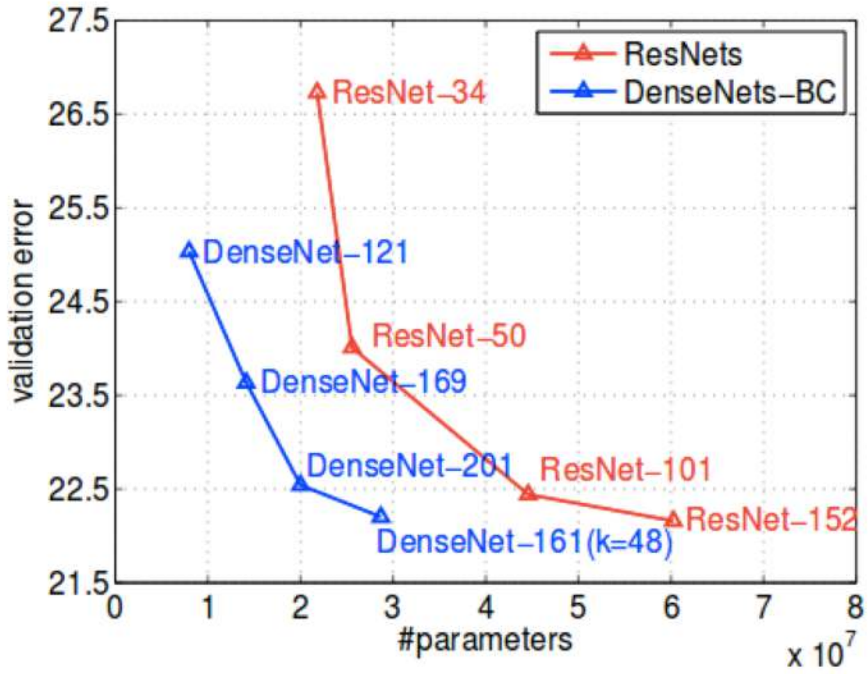


[Huang et al., CVPR 2017]

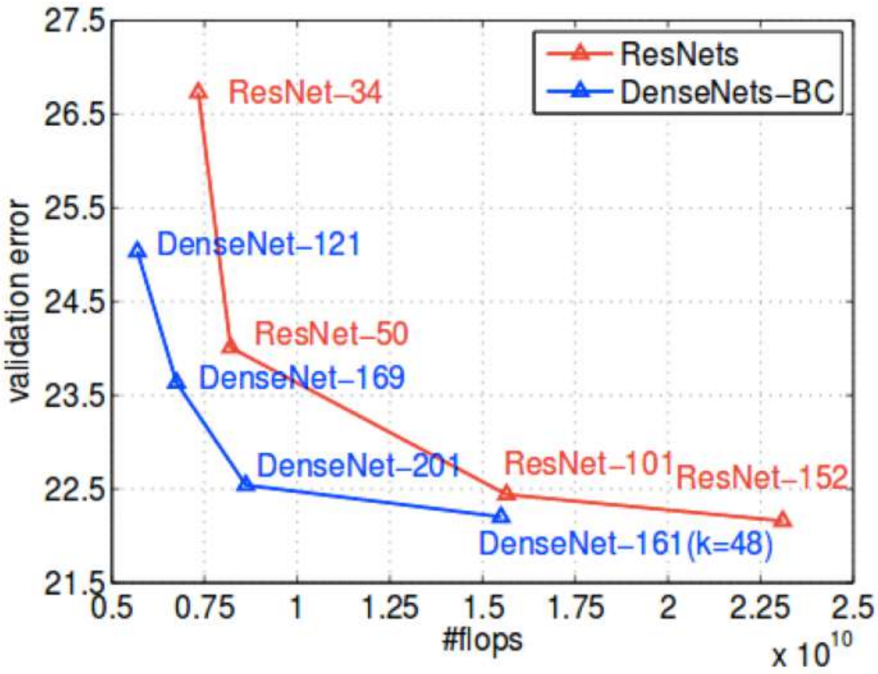
DenseNet

Higher accuracy than ResNet with fewer weights and multiplications

Top-1 error



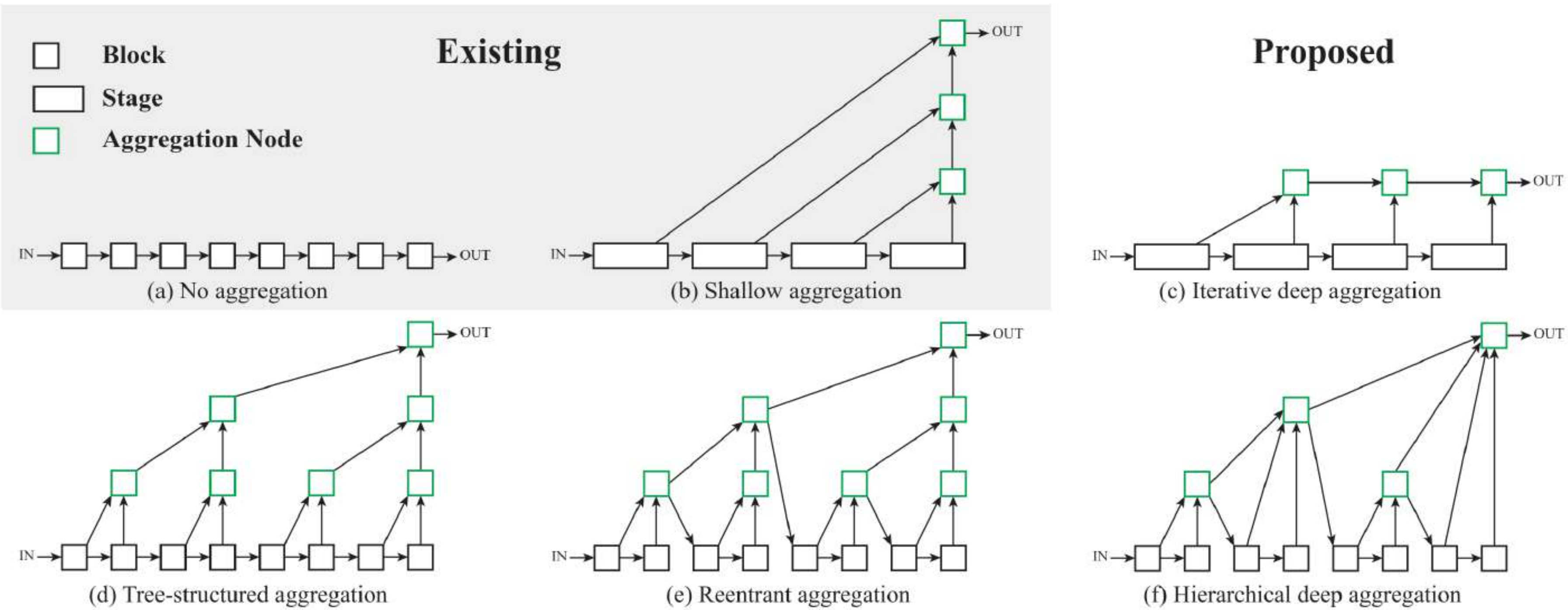
Top-1 error



Note: 1 MAC = 2 FLOPS

Feature Map Reuse

- **More complicated layer aggregation**



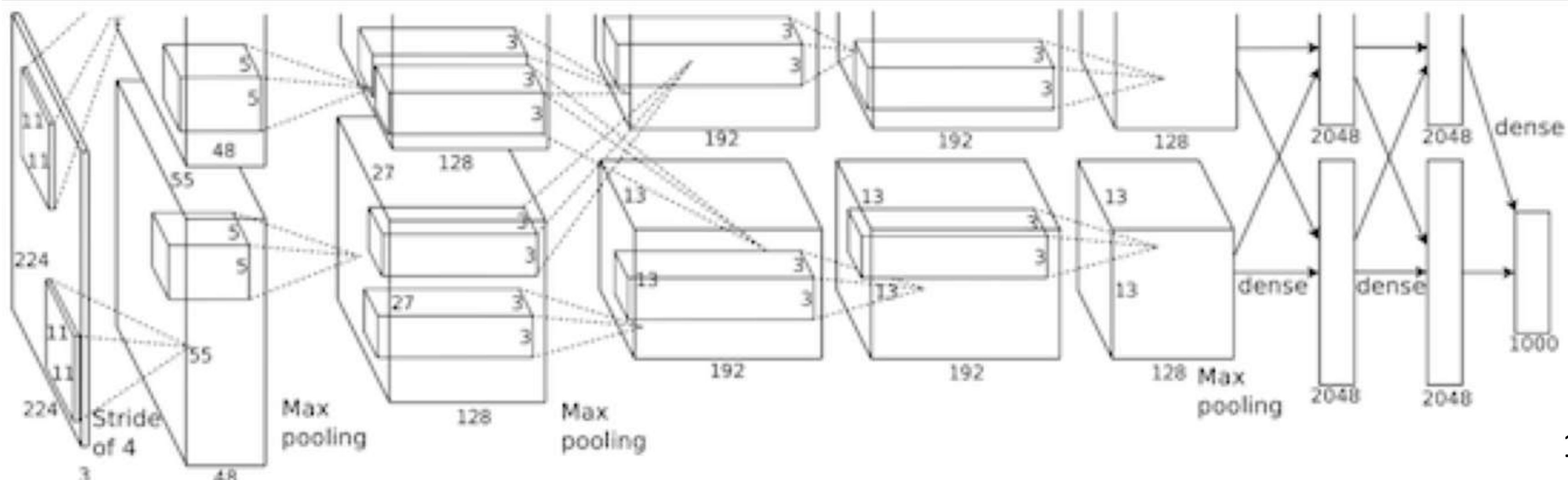
[Yu et al., CVPR 2018]

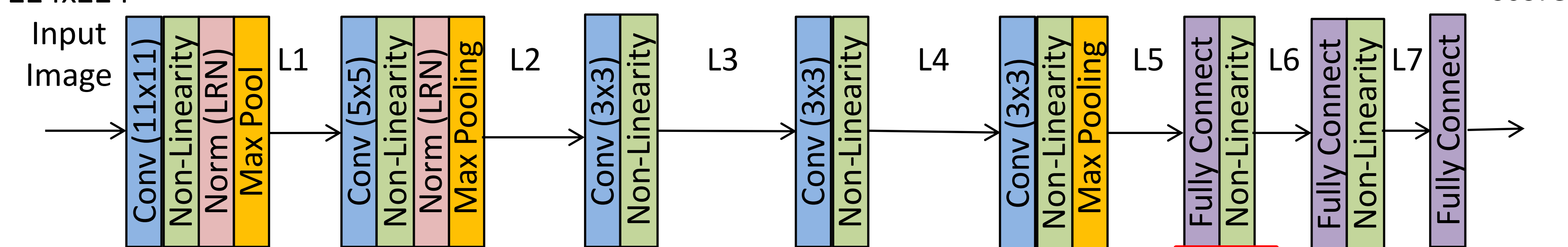
Simplify FC Layers

CONV Layers: 5
Fully Connected Layers: 3
Weights: 61M
MACs: 724M

ILSCVR12 Winner

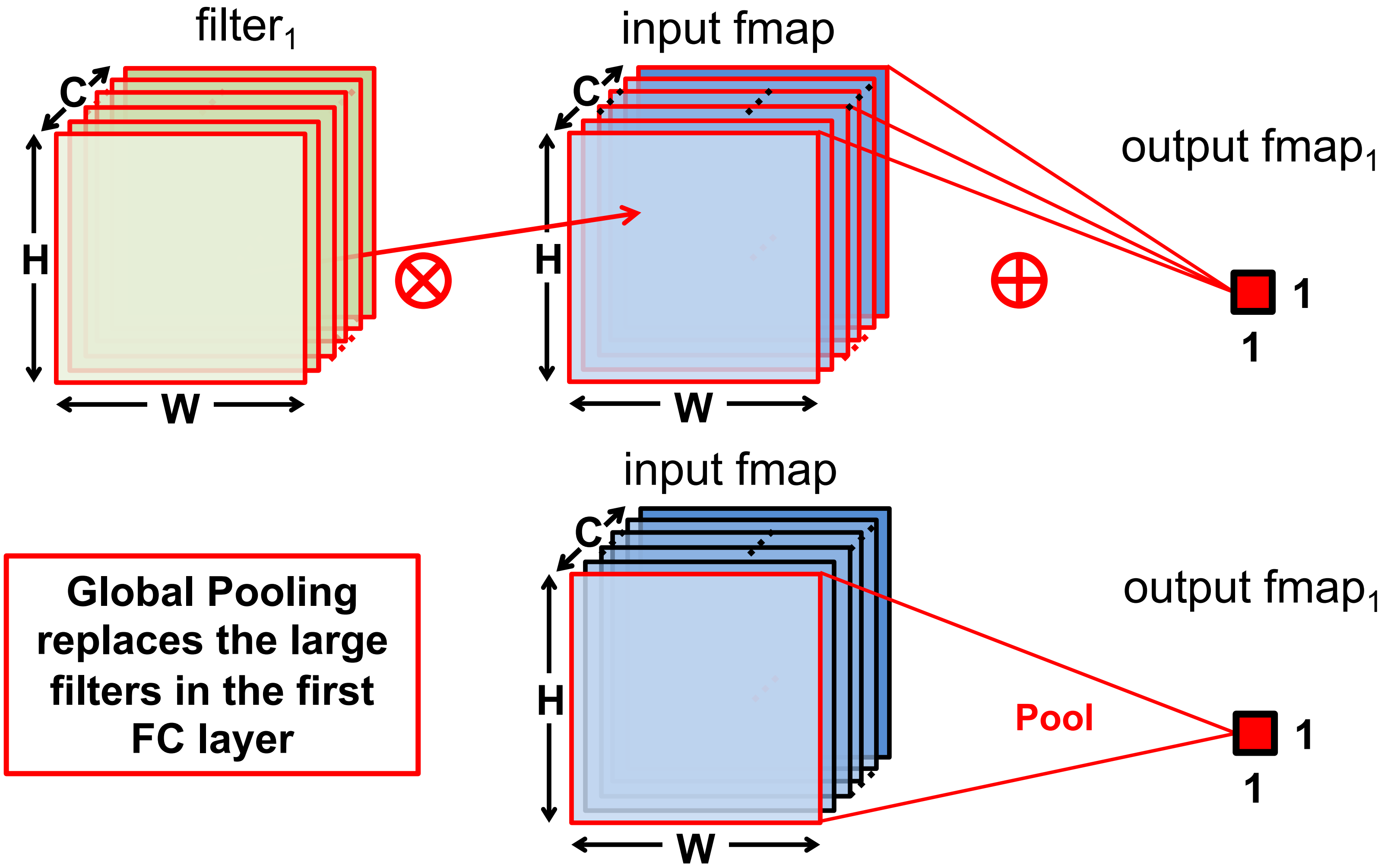
[Krizhevsky et al., NIPS 2012]





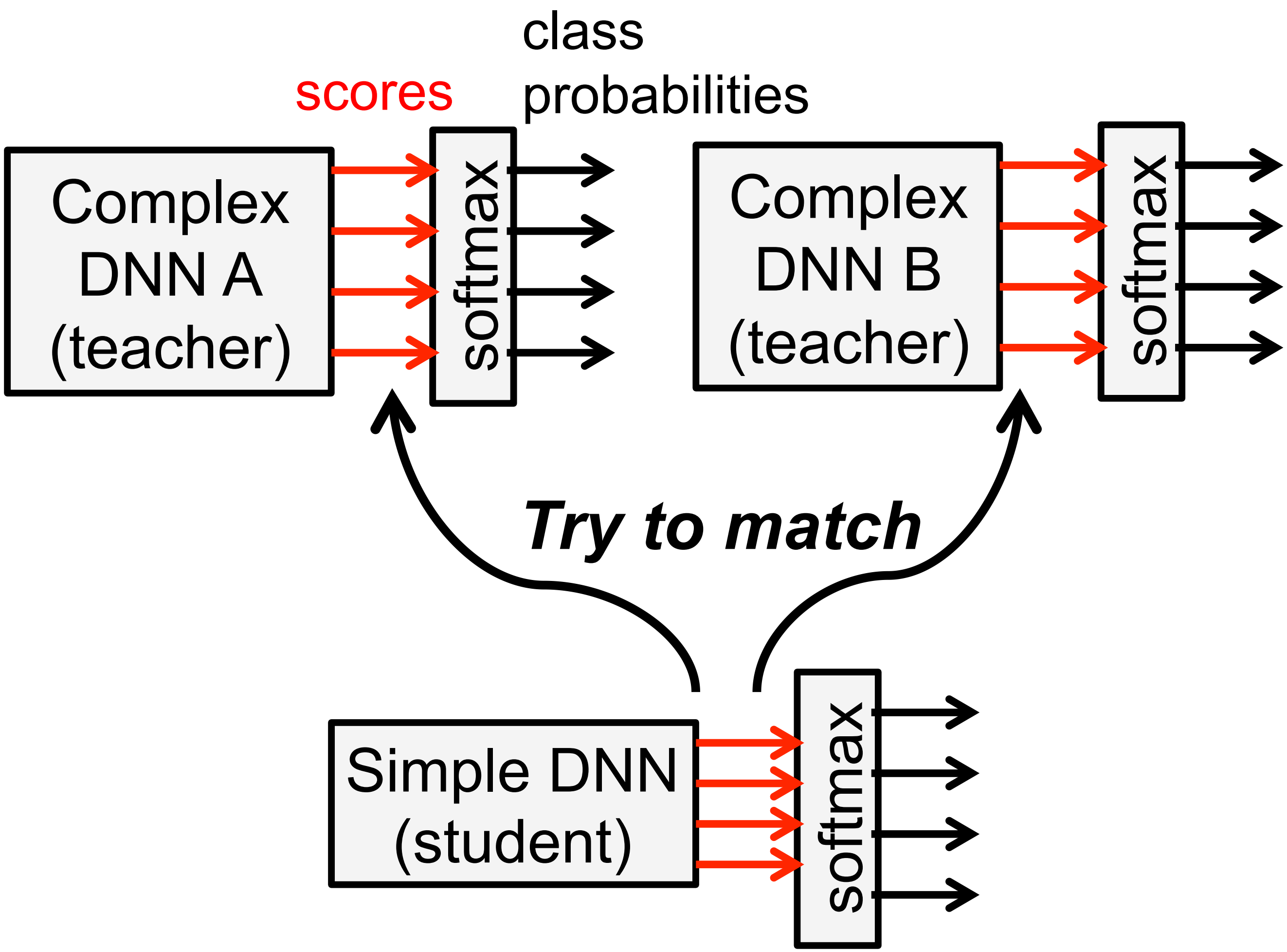
	L1	L2	L3	L4	L5	L6	L7	
# of weights	34k	307k	885k	664k	442k	38M	16.8M	4.1M
# of MACs	105M	224M	150M	112M	75M	38M	17M	4M

Simplify FC Layers



Global Pooling replaces the large filters in the first FC layer

Knowledge Distillation

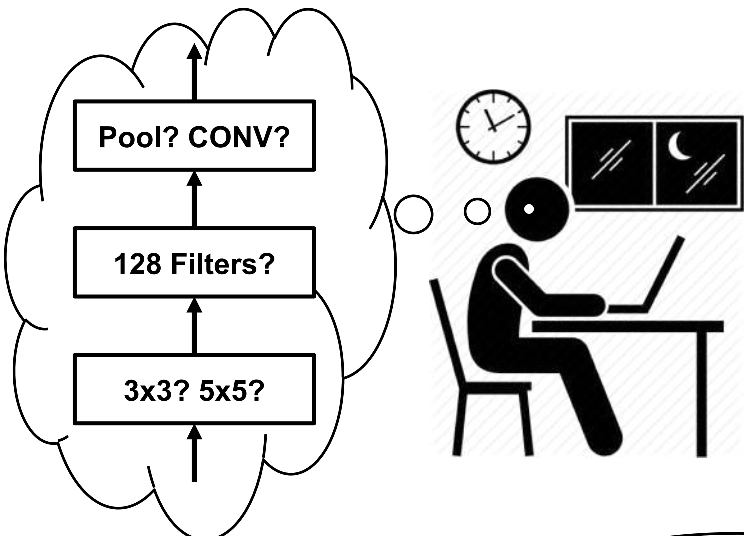


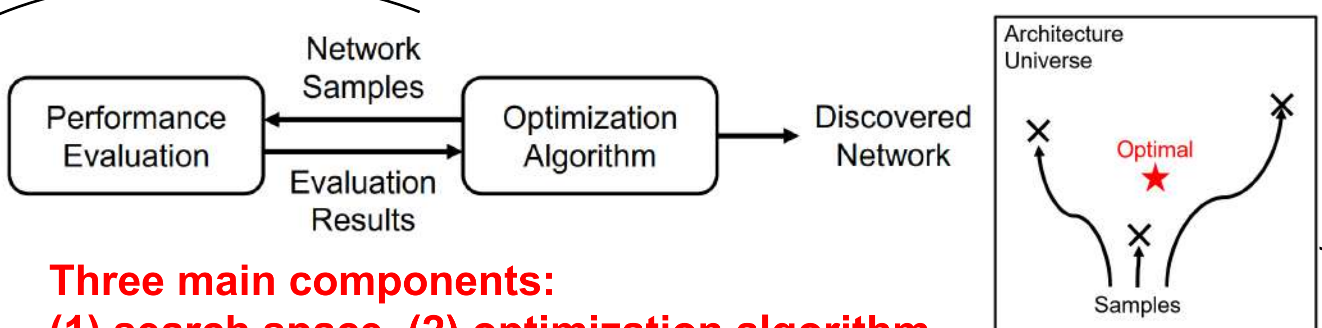
[Bucilu et al., KDD 2006],[Hinton et al., arXiv 2015]

Network Architecture Search (NAS)

Learn Network Architecture

Rather than handcrafting the architecture, automatically search for it





Three main components:
(1) search space, (2) optimization algorithm, and (3) performance evaluation.

Evaluate NAS Performance

- **Key Metrics**
  - Achievable DNN accuracy
  - Required search time

$$time_{nas} = num_{samples} \times time_{per\_sample}$$

$$time_{nas} \propto \left(\frac{\boxed{num_{nas\_tuning}} \times \boxed{size_{search\_space}}}{\boxed{efficiency_{alg}}}\right) \times \boxed{(time_{train} + time_{eval})}$$

(2) Improve the optimization algorithm

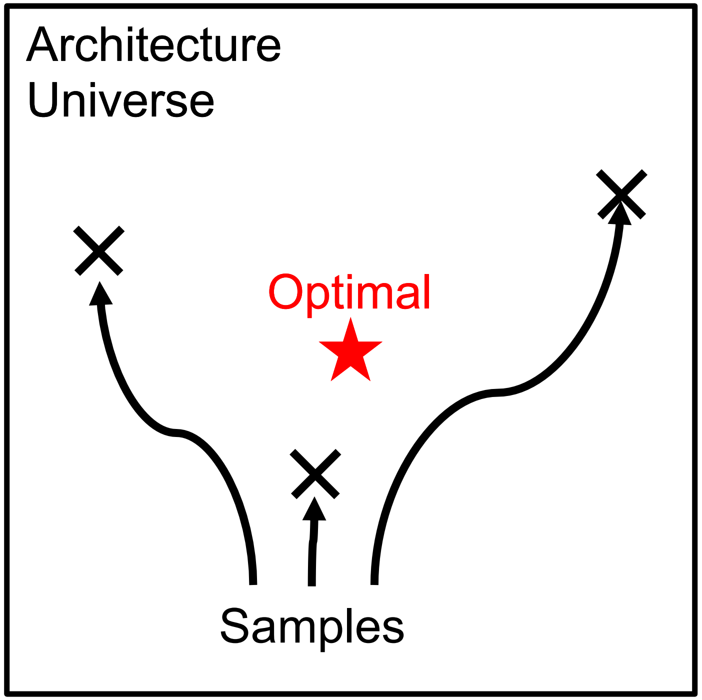
(1) Shrink the search space

(3) Simplify the performance evaluation

Researchers improve the efficiency of NAS in 3 main components

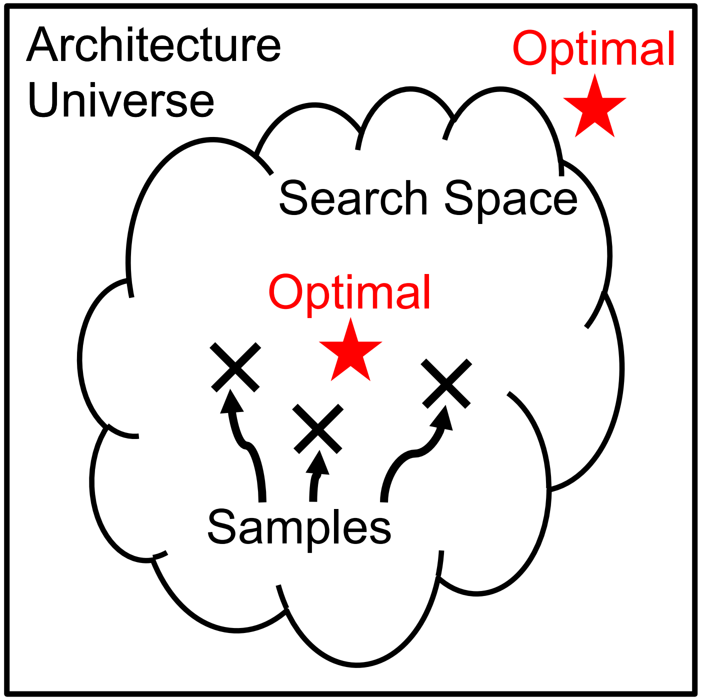
(1) Shrink the Search Space

- **Trade the discoverable architectures for search speed**



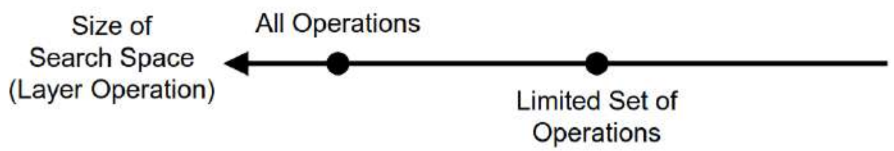
(1) Shrink the Search Space

- **Trade the discoverable architectures for search speed**
- **May irrecoverably limit the achievable network performance**
  - **Domain knowledge learned in manual network design provides guidance**



(1) Shrink the Search Space

- **Search space = layer operations + connections between layers**

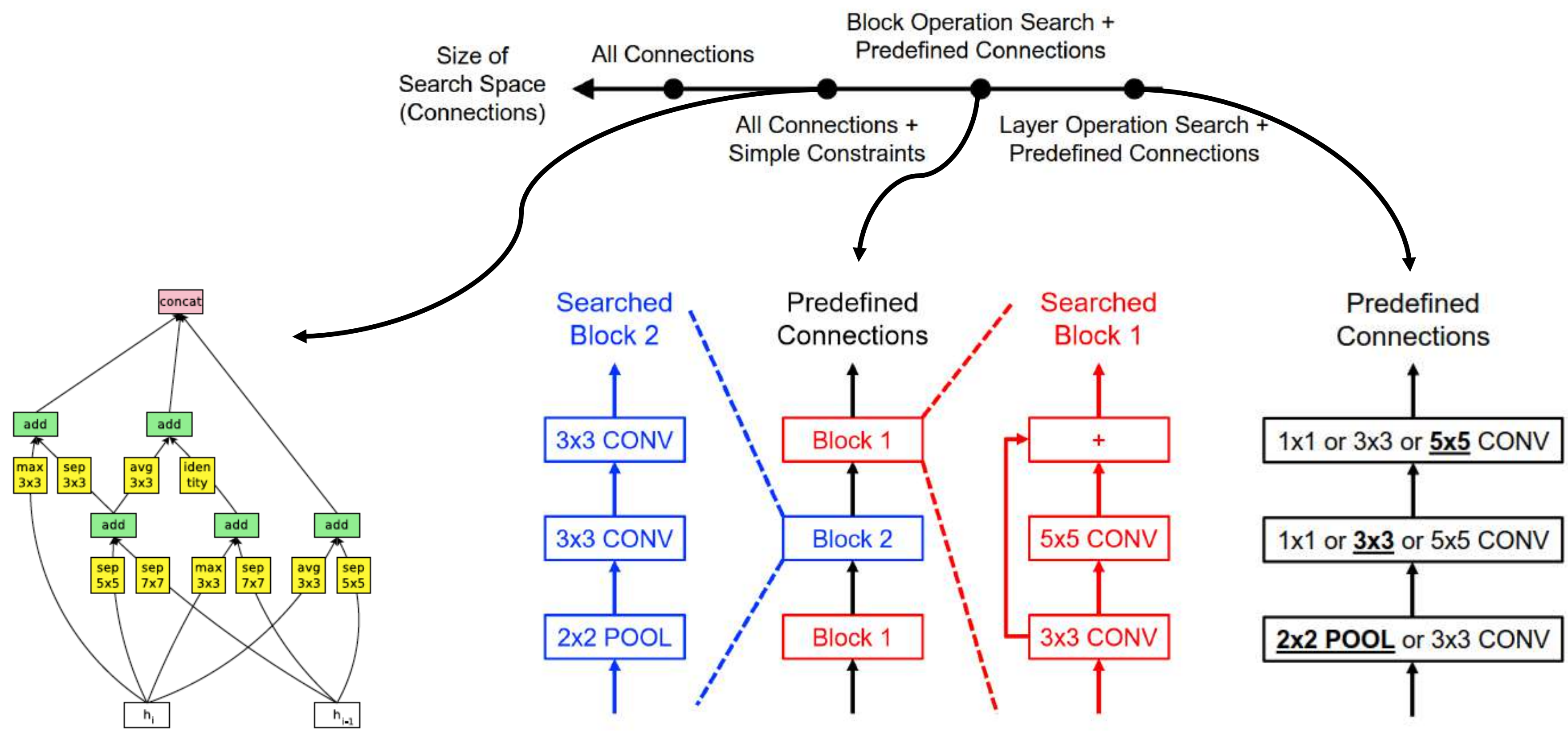


Common layer operations:

- Identity
- 1x3 then 3x1 convolution
- 1x7 then 7x1 convolution
- 3x3 dilated convolution
- 1x1 convolution
- 3x3 convolution
- 3x3 separable convolution
- 5x5 separable convolution
- 3x3 average pooling
- 3x3 max pooling
- 5x5 max pooling
- 7x7 max pooling

(1) Shrink the Search Space

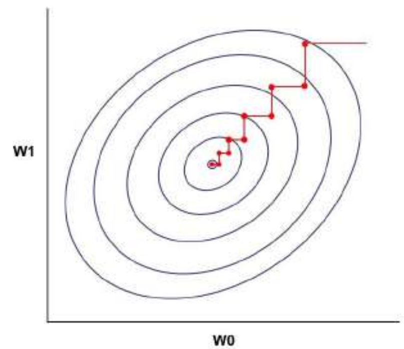
- **Search space = layer operations + <u>connections between layers</u>**



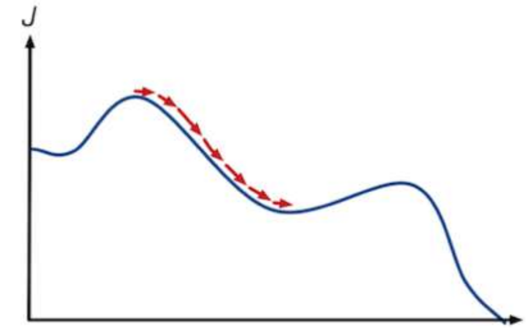
(2) Improve Optimization Algorithm

Random

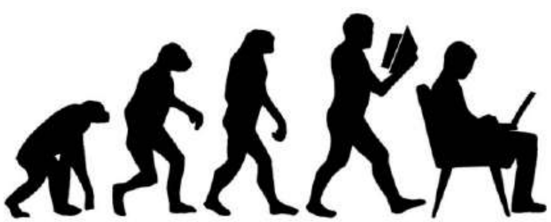
Coordinate Descent



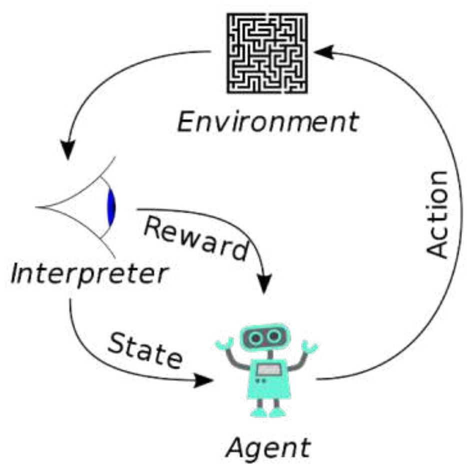
Gradient Descent



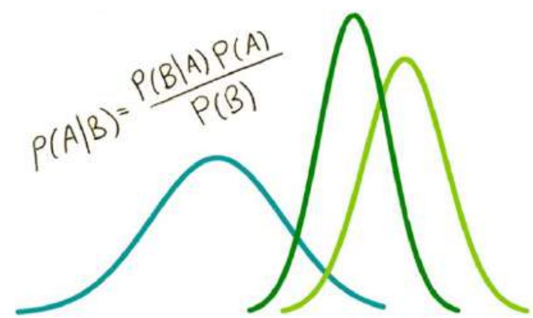
Evolutionary



Reinforcement Learning



Bayesian



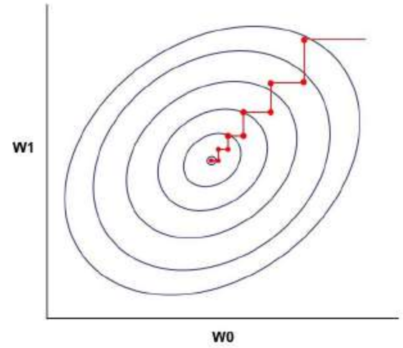
(2) Improve Optimization Algorithm

Random

Randomly samples the entire space

- Simple
- Does not use previous results

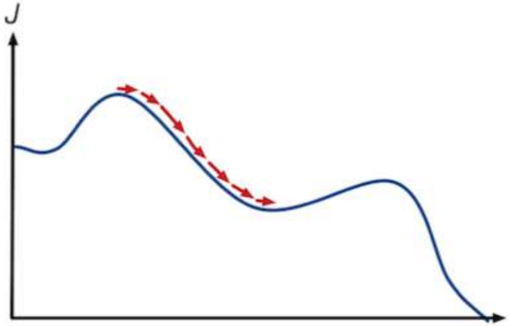
Coordinate Descent



Starts from the previous best sample and greedily finds the best direction to move

- Uses previous results
- Simple
- Limited number of directions

Gradient Descent



Starts from the previous best sample and goes in the direction that has the largest gradient

- Explores more directions
- The metric should be differentiable

(2) Improve Optimization Algorithm

Starts from the previous best sample and goes in the best randomly-sampled direction

- The metric does not need to be differentiable
- More complicated

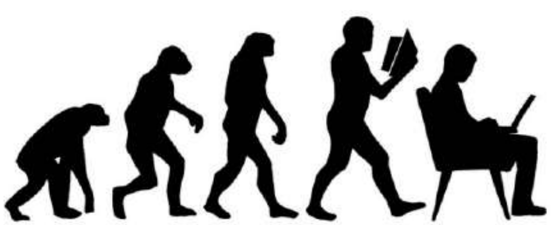
Learns from the previous samples and infers the best sample

- Better uses the previous samples
- Needs to design and train the agent

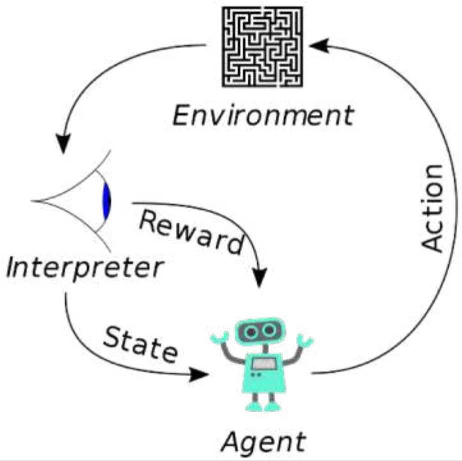
Models the entire surface of the search space and picks the best sample

- Gets rid of the iterative process
- Hard to model a large search space

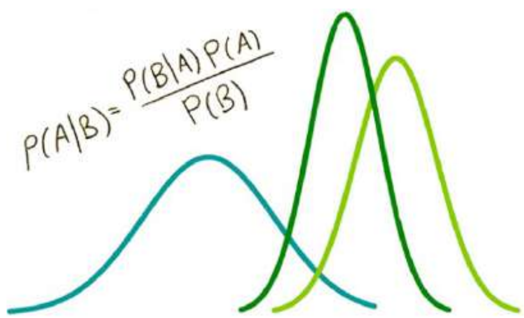
Evolutionary



Reinforcement Learning



Bayesian



(3) Simplify the Performance Evaluation

- **NAS needs only the rank of the performance values**
- **Method 1: approximate accuracy**
- **Method 2: approximate weights**
- **Method 3: approximate metrics (e.g., latency, energy)**

(3) Simplify the Performance Evaluation

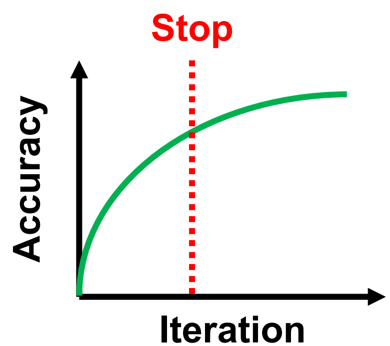
- NAS needs only the rank of the performance values
- Method 1: approximate accuracy

Proxy Task

E.g., Smaller resolution, simpler tasks

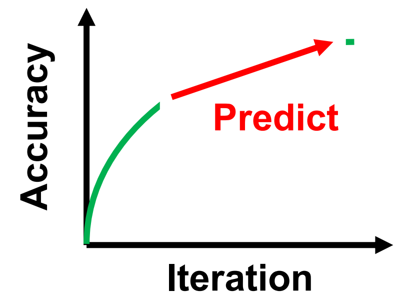
Early Termination

Stop training earlier



Accuracy Prediction

Extrapolate accuracy



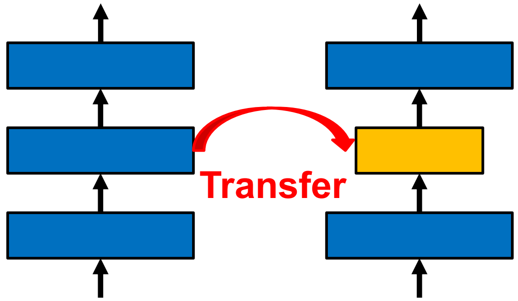
- Method 2: approximate weights
- Method 3: approximate metrics

(3) Simplify the Performance Evaluation

- NAS needs only the rank of the performance values
- Method 1: approximate accuracy
- Method 2: approximate weights

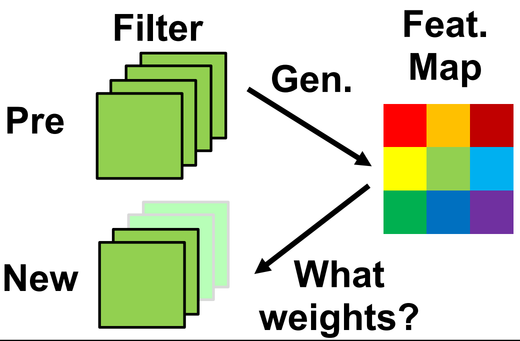
Transfer Weights

Reuse weights from other similar networks



Estimate Weights

Infer the weights from the previous feature maps



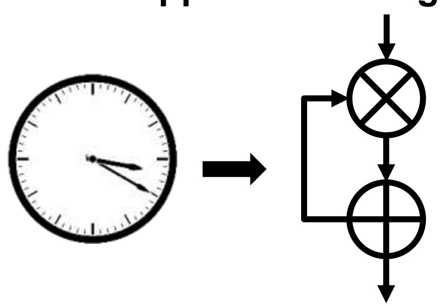
- Method 3: approximate metrics

(3) Simplify the Performance Evaluation

- **NAS needs only the <u>rank</u> of the performance values**
- Method 1: approximate accuracy
- Method 2: approximate weights
- **Method 3: approximate metrics (e.g., latency, energy)**

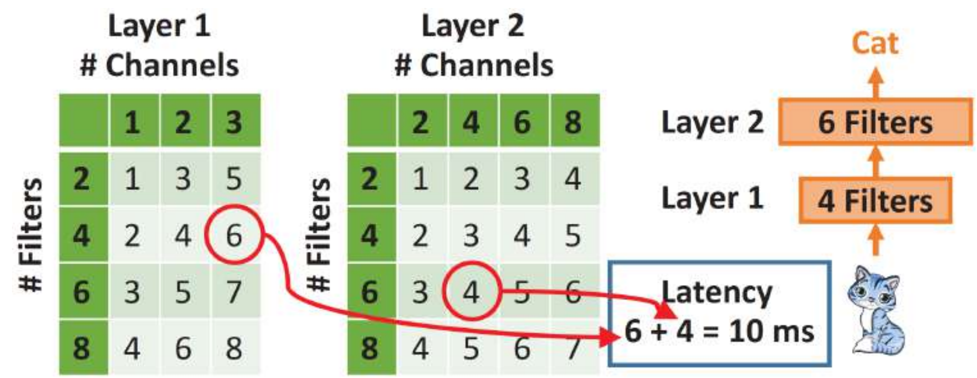
Proxy Metric

Use an easy-to-compute metric to approximate target



Look-Up Table

Use table lookup



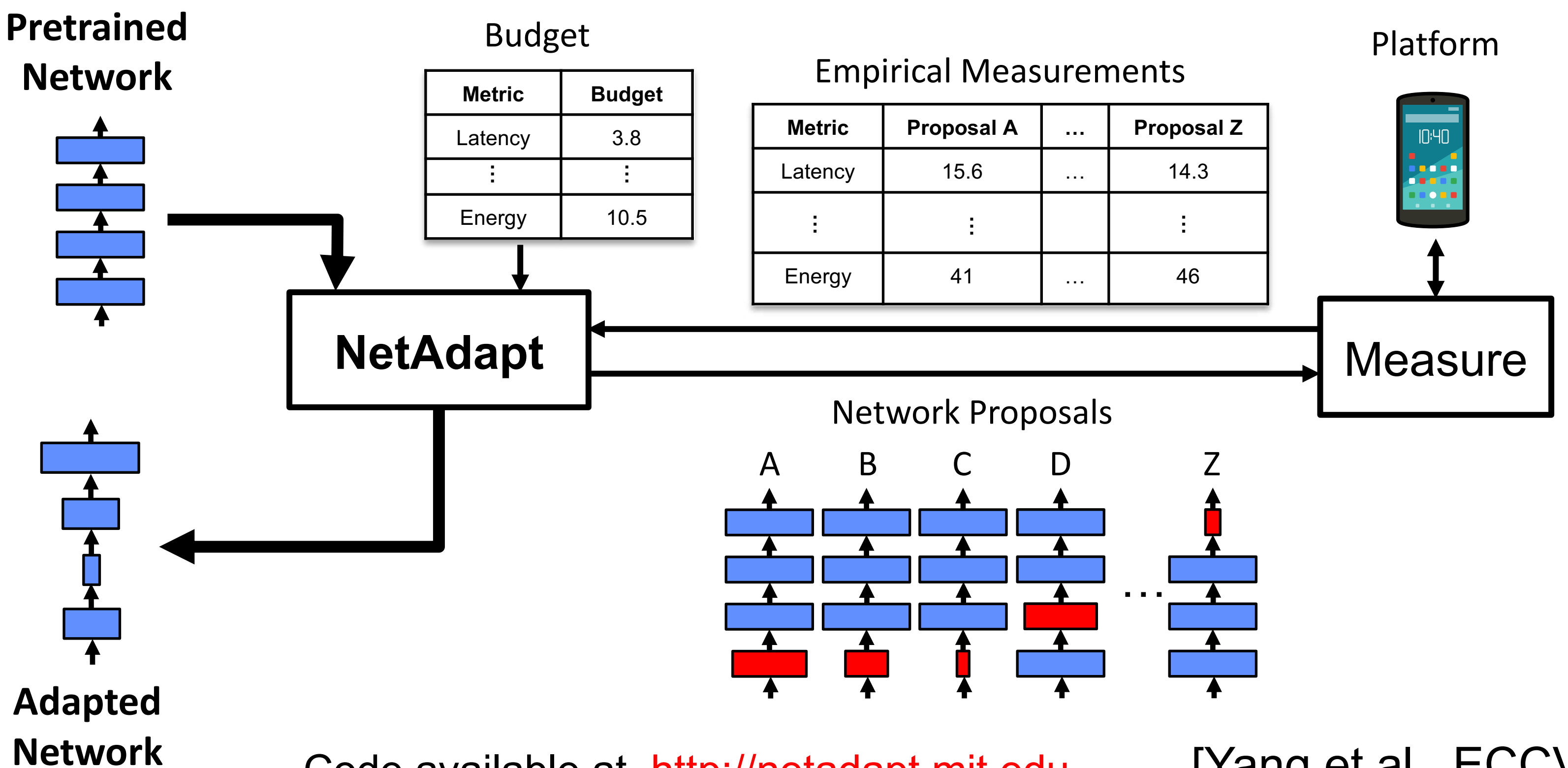
Other Things to Know

- The components may not be chosen individually
  - Some optimization algorithms limit the search space
  - Using direct hardware metrics may limit the selection of the optimization algorithms

- Commonly overlooked properties
  - The complexity of implementation and usage
  - The ease of tuning
  - The probability of convergence to a good architecture

NetAdapt: Platform-Aware DNN Adaptation

- **Automatically adapt DNN** to a mobile platform to reach a target latency or energy budget
- An example of **coordinate descent NAS**



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

[Yang et al., ECCV 2018]

Problem Formulation

$$\max_{Net} Acc(Net) \;\; subject\ to\ Res_j(Net) \leq Bud_j, j = 1, \cdots, m$$

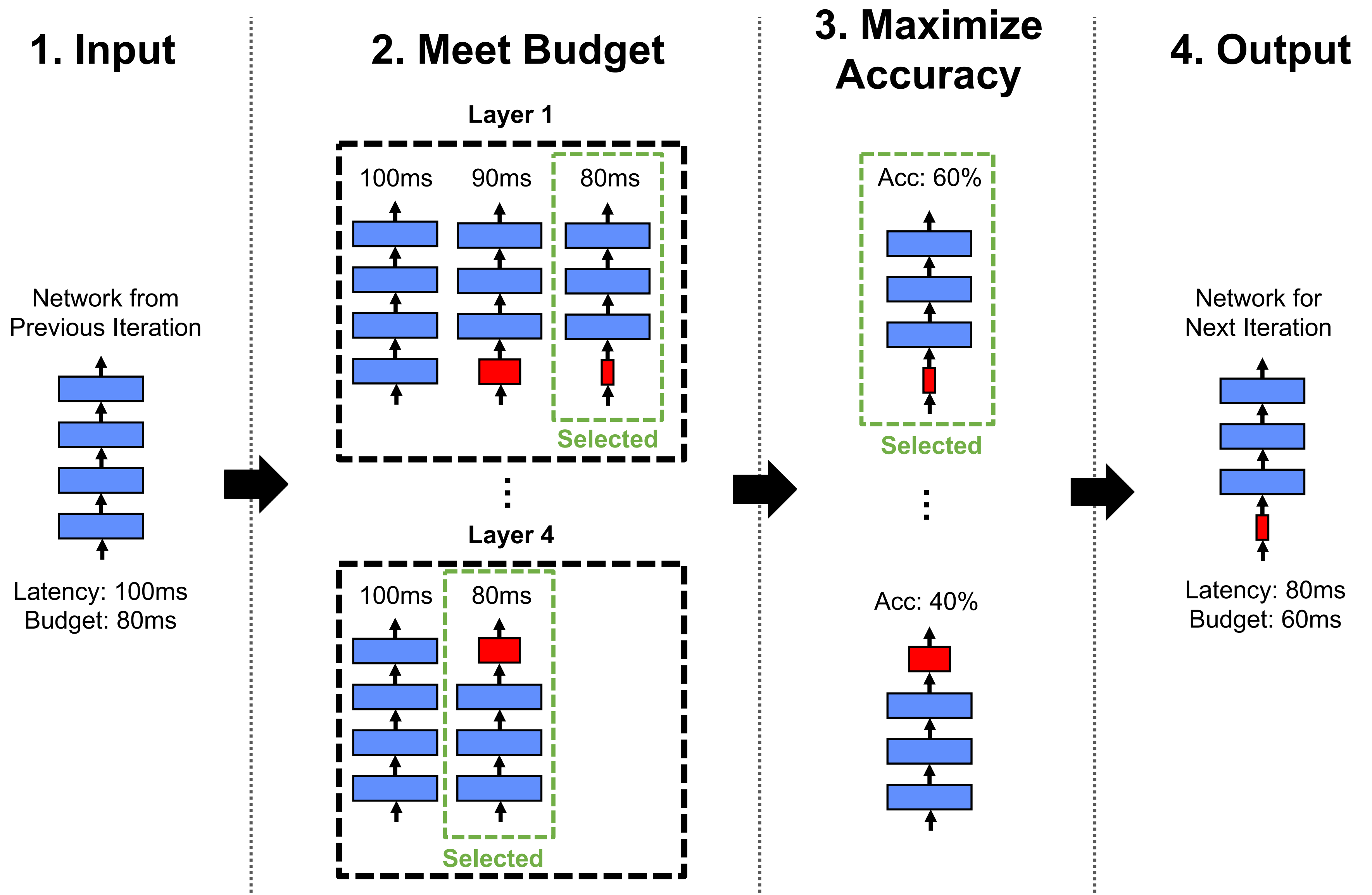
Break into a set of simpler problems and solve iteratively

$$\max_{Net_i} Acc(Net_i) \;\; subject\ to\ Res_j(Net_i) \leq Res_j(Net_{i-1}) - \Delta R_{i,j}, j = 1, \cdots, m$$

**Acc*: accuracy function, *Res*: resource evaluation function,
***ΔR*: resource reduction**, *Bud*: given budget

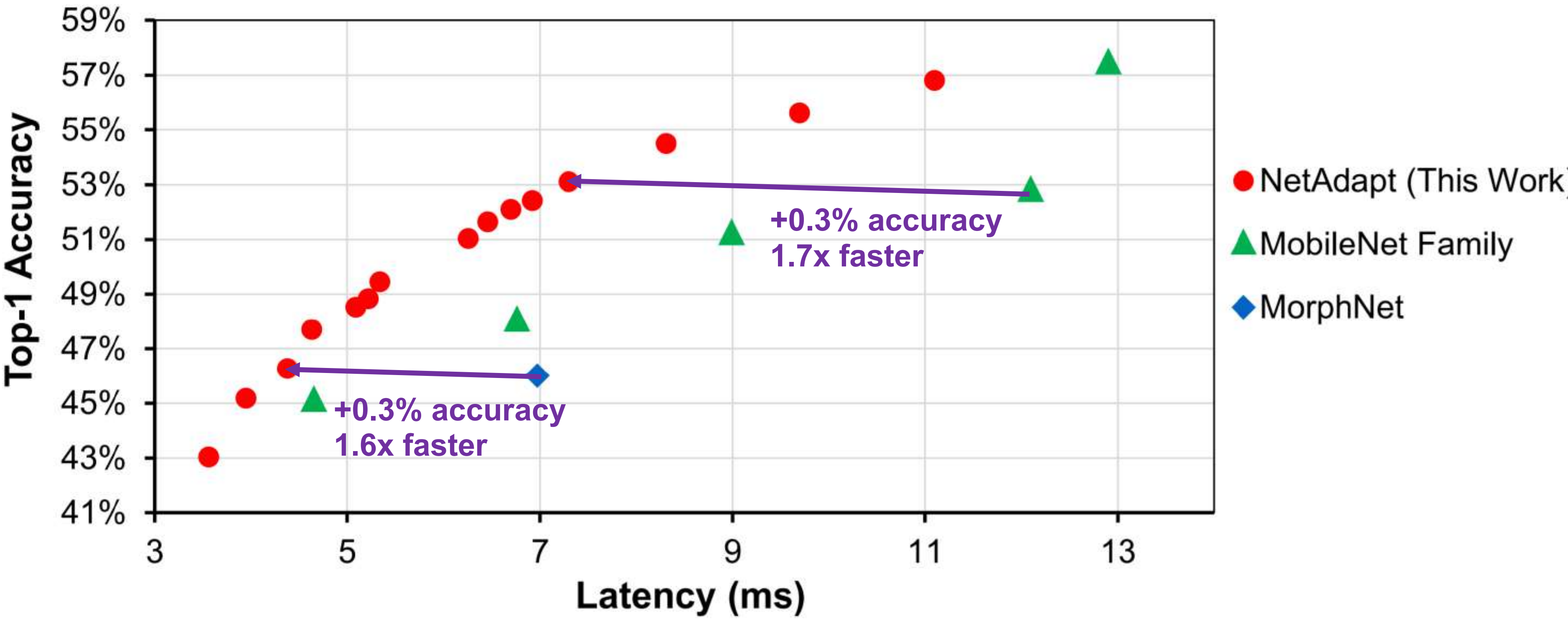
- **Advantages**
  - Supports multiple resource budgets at the same time
  - Guarantees that the budgets will be satisfied because the resource consumption decreases monotonically
  - Generates a family of networks (from each iteration) with different resource versus accuracy trade-offs

Simplified Example of One Iteration



Improved Latency vs. Accuracy Tradeoff

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

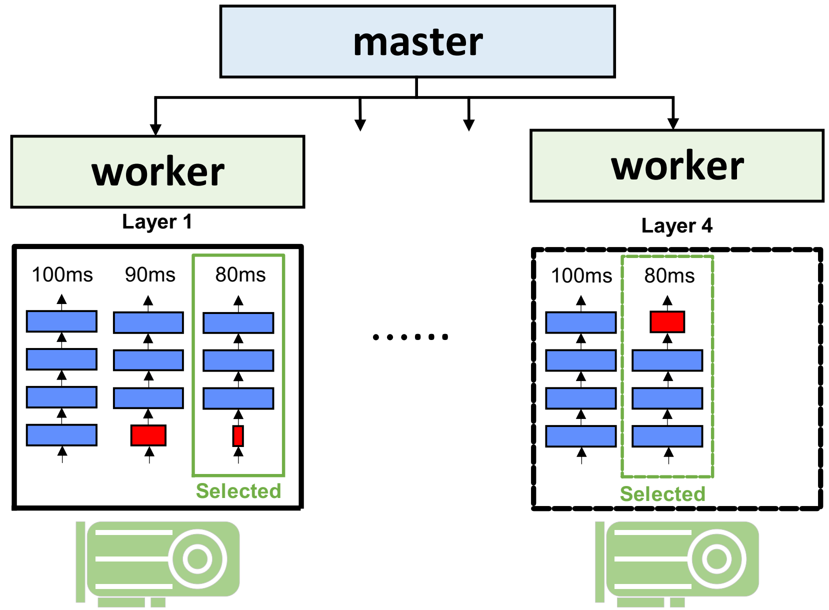


*Tested on the ImageNet dataset and a Google Pixel 1 CPU

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

Code of NetAdapt

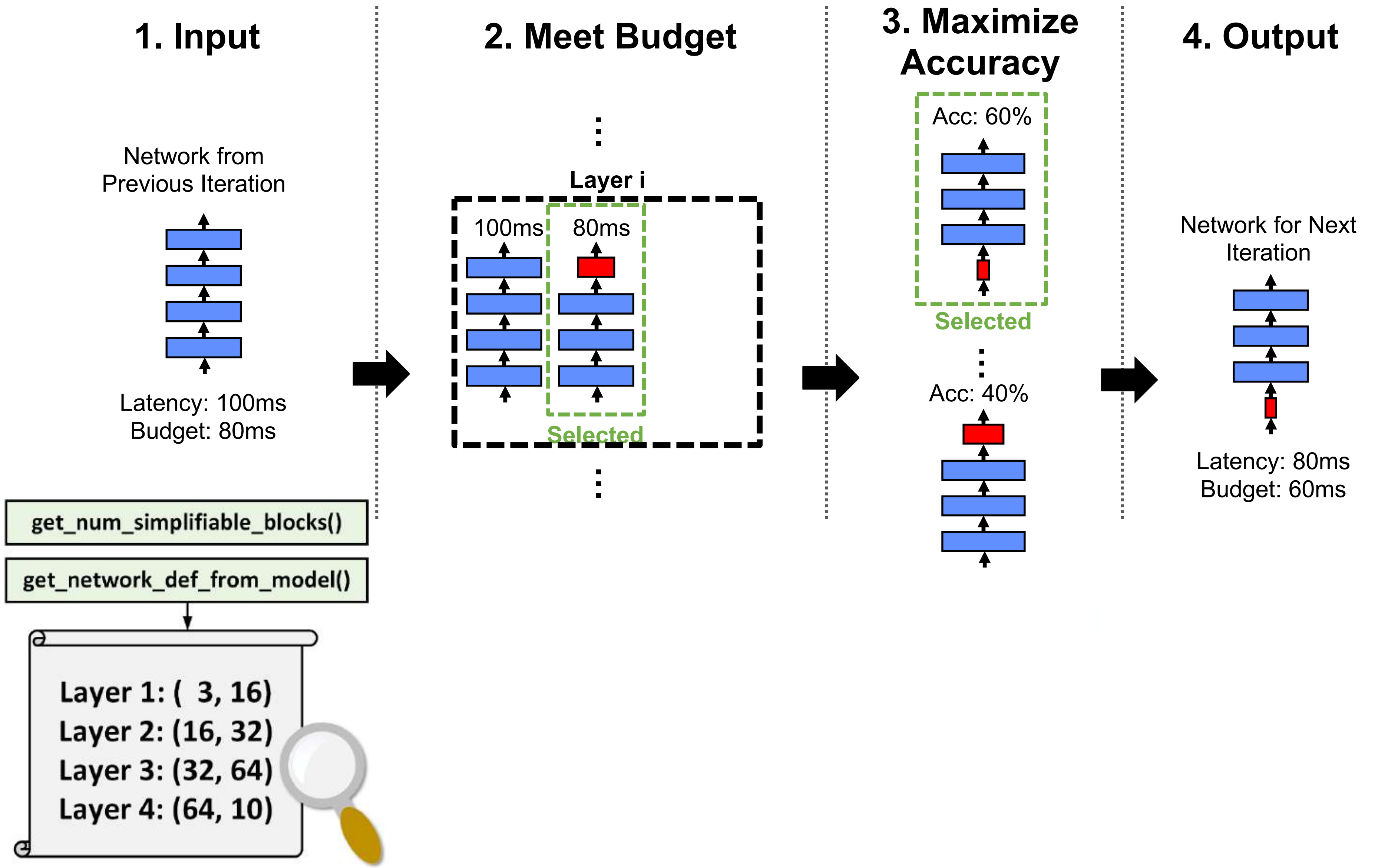
- Reimplemented framework on PyTorch
- **Flexible**: can support **different networks and tasks**
- **Scalable**: spawn multiple workers to simplify networks in parallel



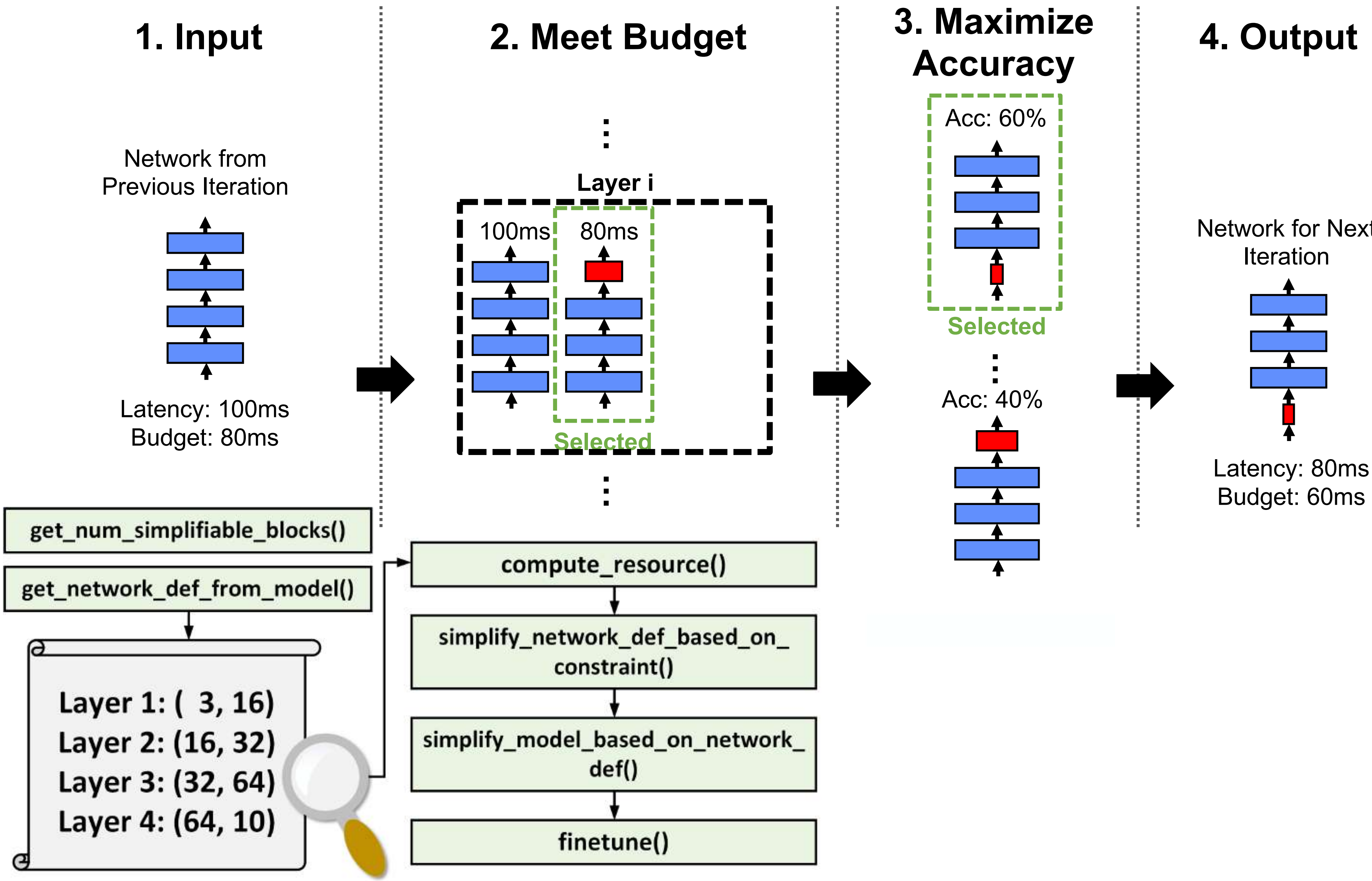
- **Easy-to-use**: require implementing only **one file (8 functions)**

Code available at https://github.com/denru01/netadapt

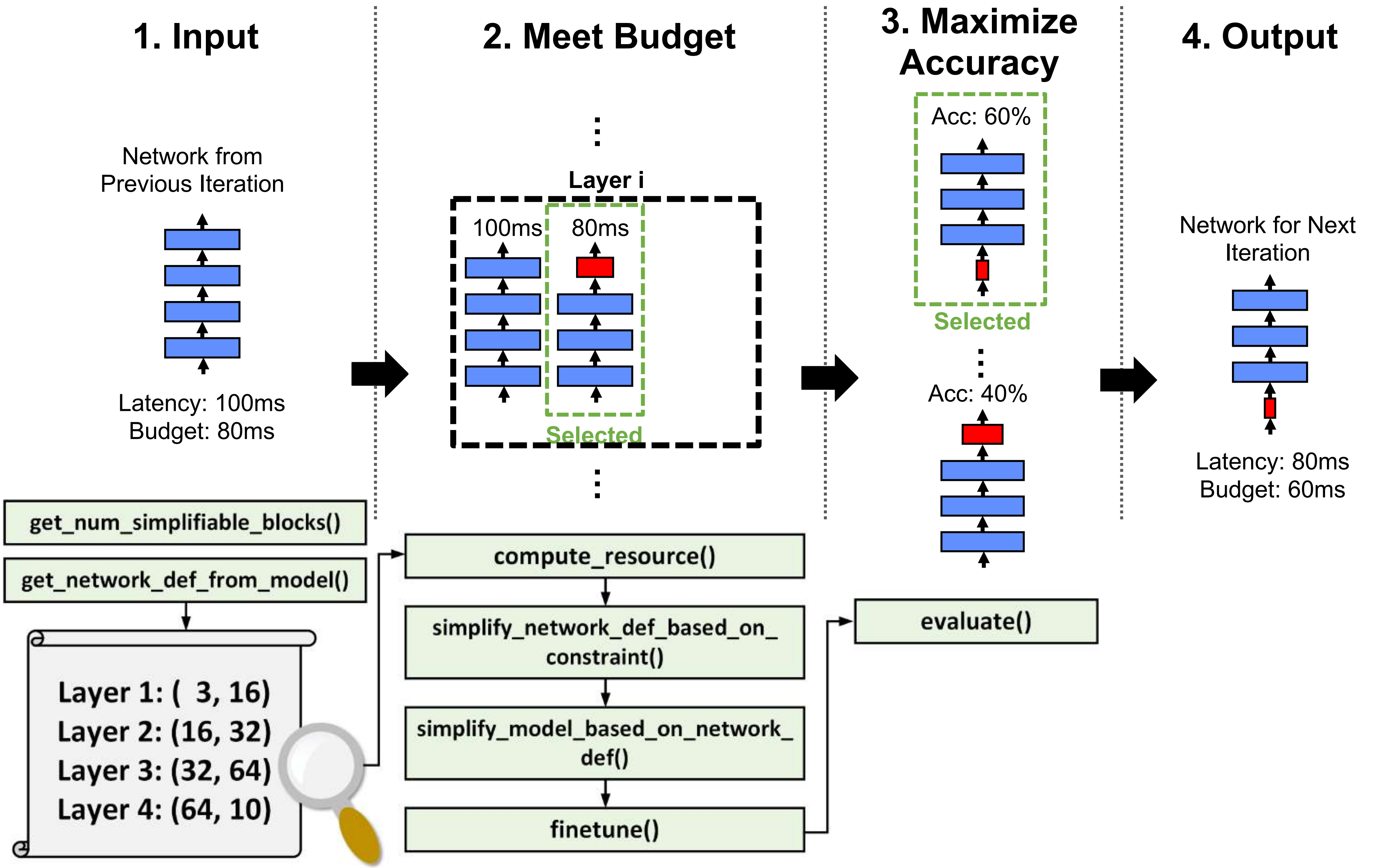
Code of NetAdapt



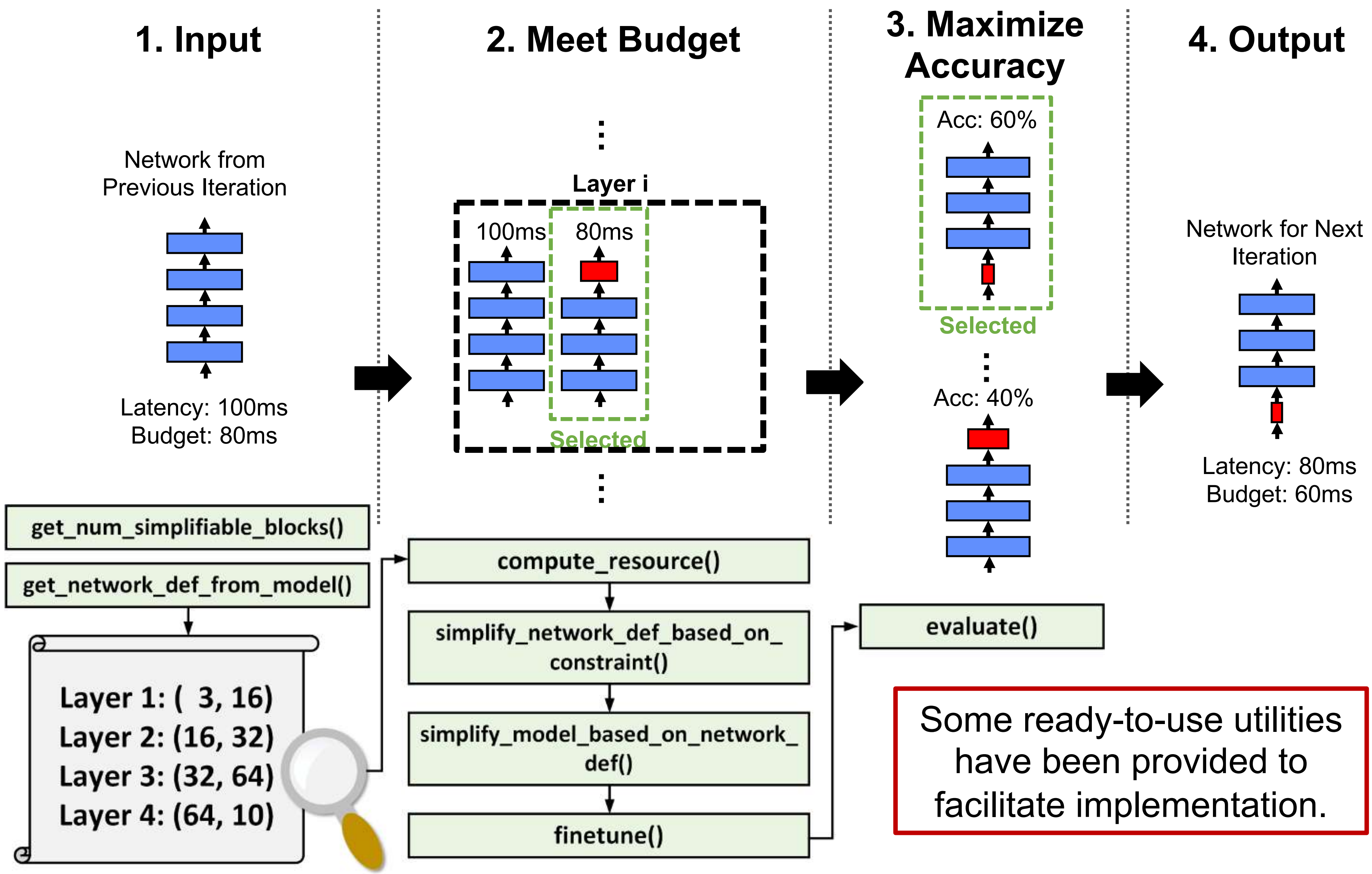
Code of NetAdapt



Code of NetAdapt



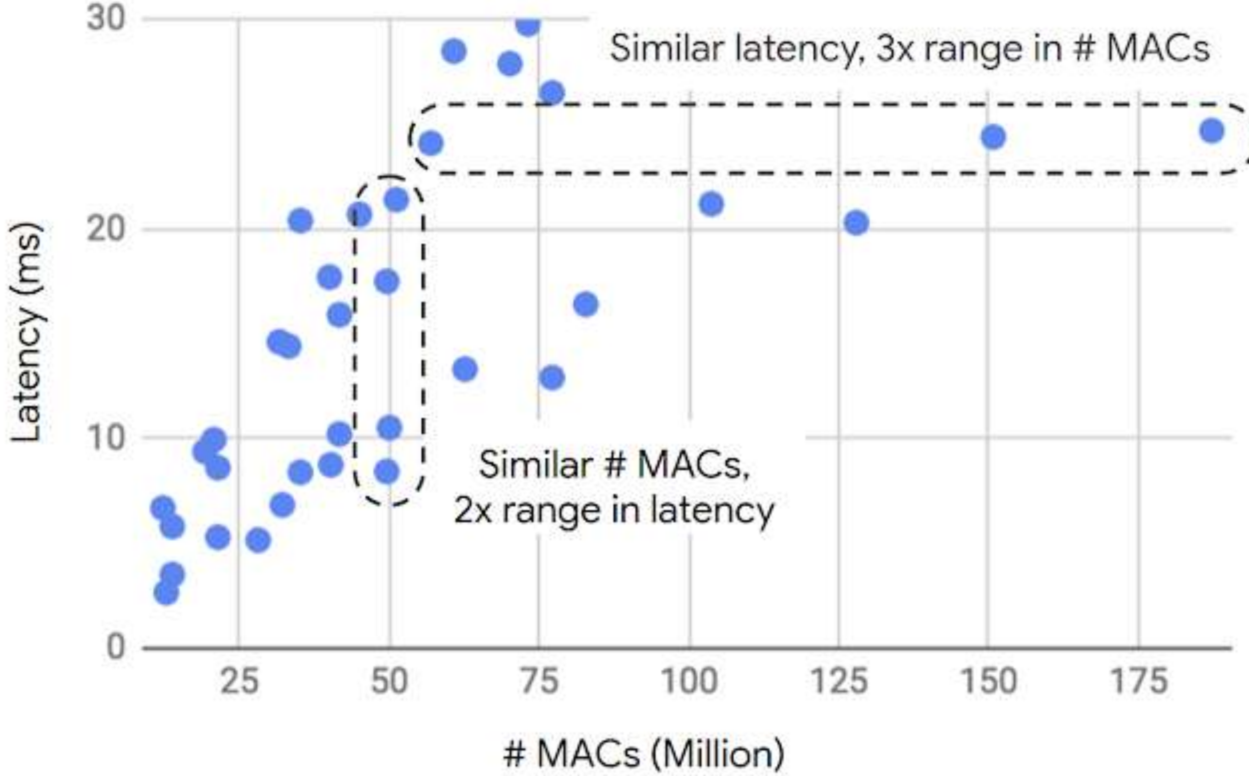
Code of NetAdapt



Hardware In the Loop

of Operations vs. Latency

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

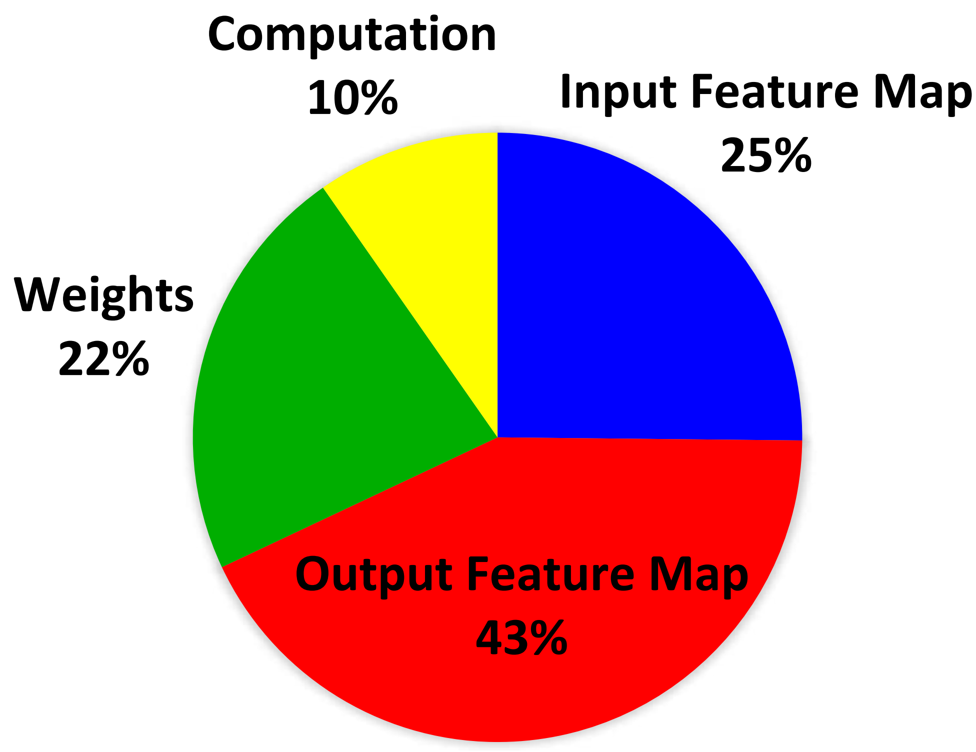


Source: Google (https://ai.googleblog.com/2018/04/introducing-cvpr-2018-on-device-visual.html)

of Weights vs. Energy

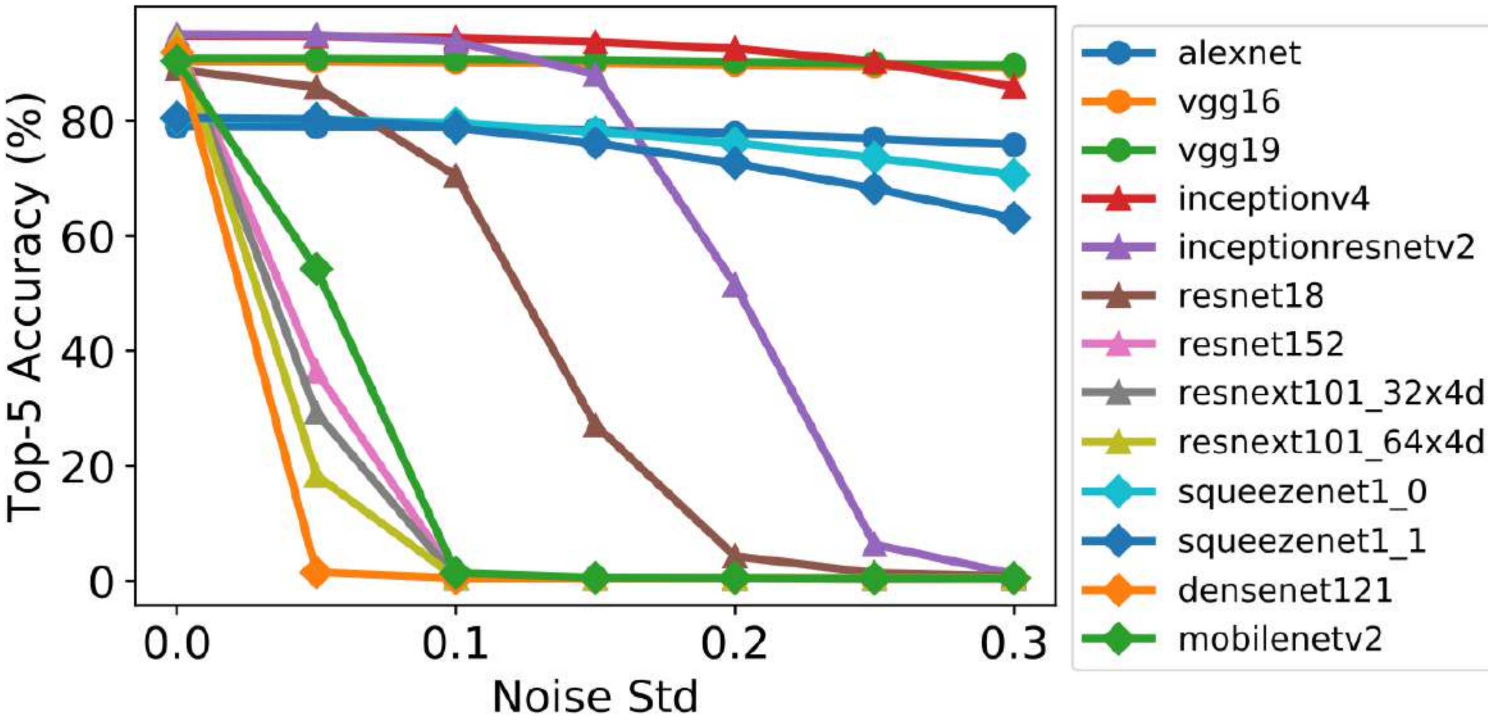
- Number of weights ***alone*** is not a good metric for energy
- **All data types** should be considered

Energy Consumption of GoogLeNet



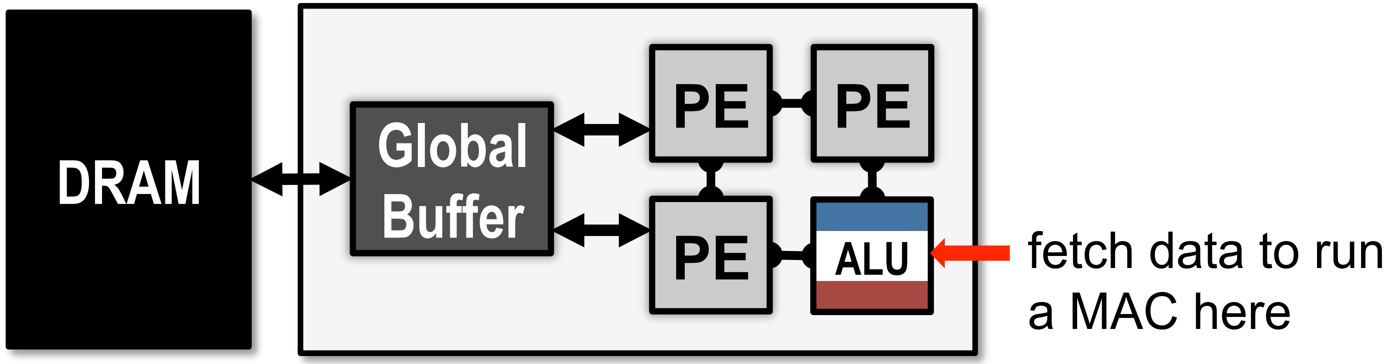
Other Hardware Metrics

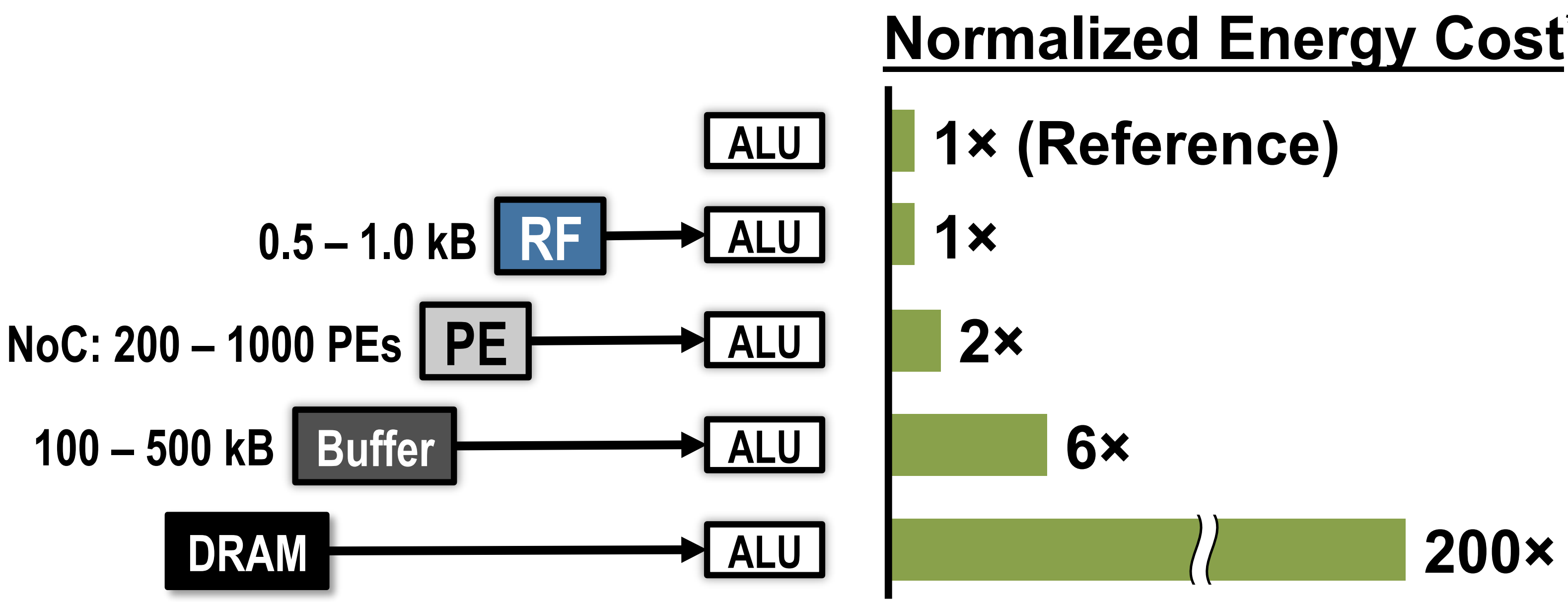
- **E.g., noise resilience in analog accelerators**



DNN model that gives highest accuracy on a digital processor may not be the best for an analog processor

Data Movement is Expensive

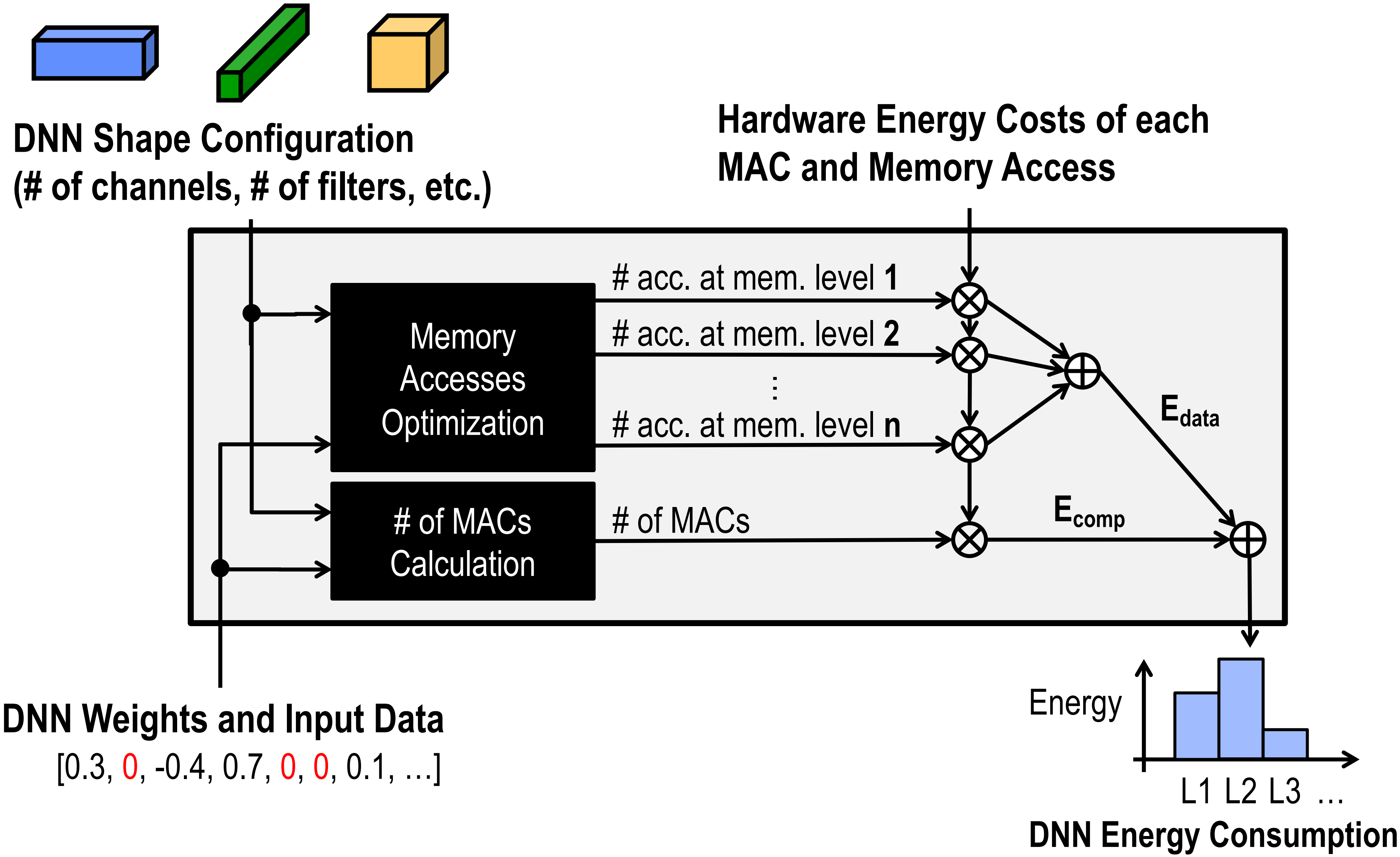




* measured from a commercial 65nm process

Energy of weight depends on **memory hierarchy** and **dataflow**

Energy Estimation Methodology



[Yang et al., Asilomar 2017]

Energy Estimation Tool V1

Website: https://energyestimation.mit.edu/

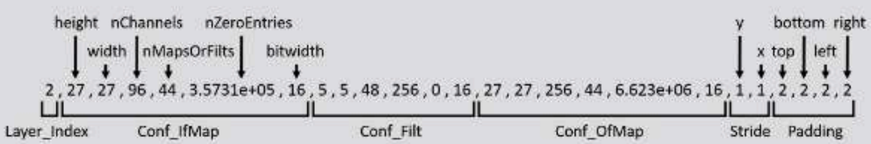
Deep Neural Network Energy Estimation Tool

Overview

This Deep Neural Network Energy Estimation Tool is used for evaluating and designing energy-efficient deep neural networks that are critical for embedded deep learning processing. Energy estimation was used in the development of the energy-aware pruning method (Yang et al., CVPR 2017), which reduced the energy consumption of AlexNet and GoogLeNet by 3.7x and 1.6x, respectively, with less than 1% top-5 accuracy loss. This website provides a simplified version of the energy estimation tool for shorter runtime (around 10 seconds).

Input

To support the variety of toolboxes, this tool takes a single network configuration file. The network configuration file is a txt file, where each line denotes the configuration of a CONV/FC layer. The format of each line is:



- Layer Index: the index of the layer, from 1 to the number of layers. It should be the same as the line number.
- Conf_IfMap, Conf_Filt, Conf_OfMap: the configuration of the input feature maps, the filters and the output feature maps. The configuration of each of the three data types is in the format of "height width number_of_channels number_of_maps_or_filts number_of_zero_entries bitwidth_in_bits".
- Stride: the stride of this layer. It is in the format of "stride_y stride_x".
- Pad: the amount of input padding. It is in the format of "pad_top pad_bottom pad_left pad_right".

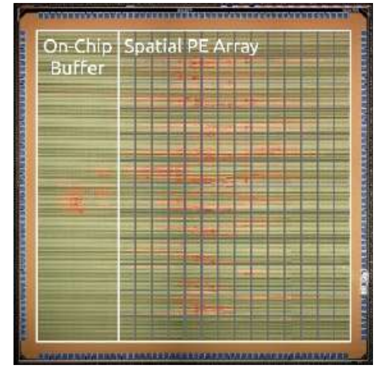
Therefore, there will be 25 entries separated by commas in each line.

Running the Estimation Model

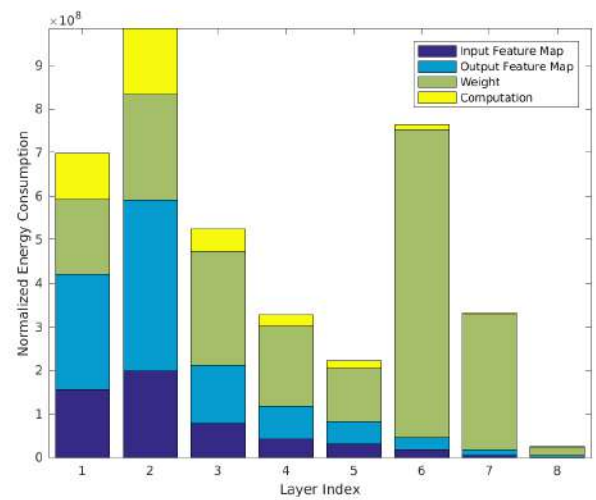
After creating your text file, follow these steps to upload your text file and run the estimation model:

1. Check the "I am not a robot" checkbox and complete the Google reCAPTCHA challenge. Help us prevent spam.
2. Click the "Choose File" button below to choose your text file from your computer.
3. Click the "Run Estimation Model" button below to upload your text file and run the estimation model.

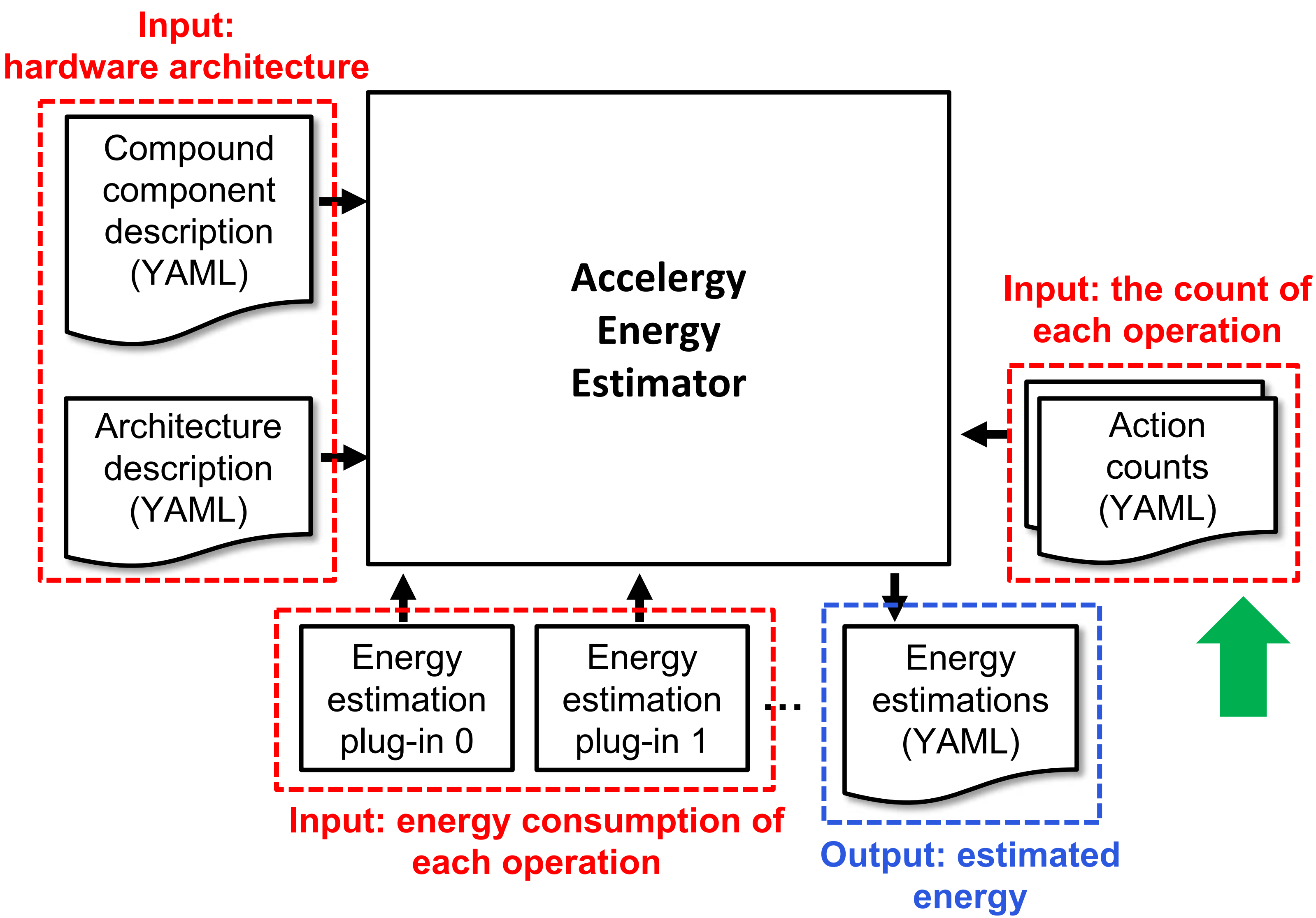
Eyeriss V1



Output DNN energy breakdown across layers



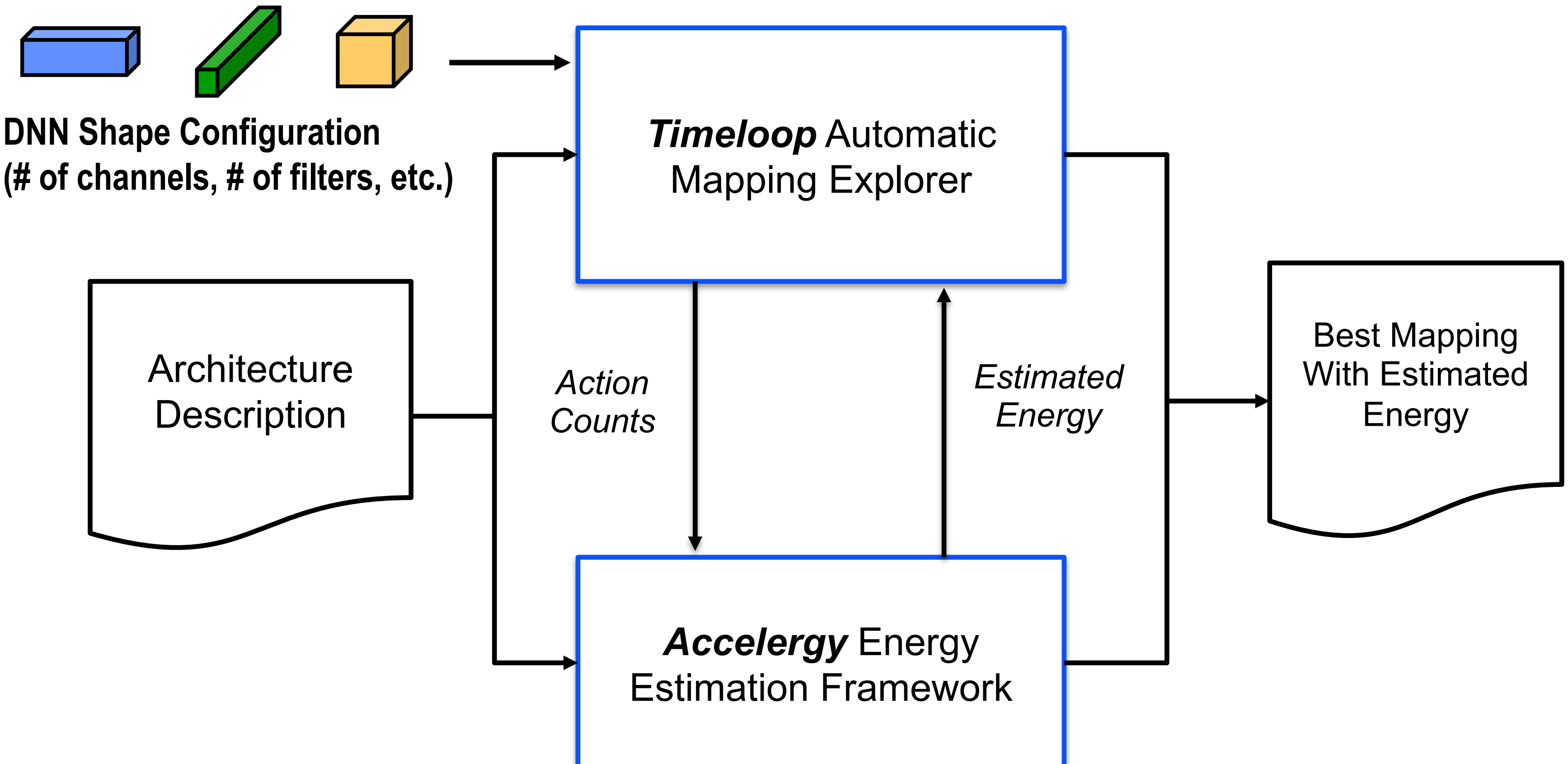
Energy Estimation Tool V2 - Accelergy



[Wu et al., ICCAD 2019]

Energy Estimation Tool V2 - Accelergy

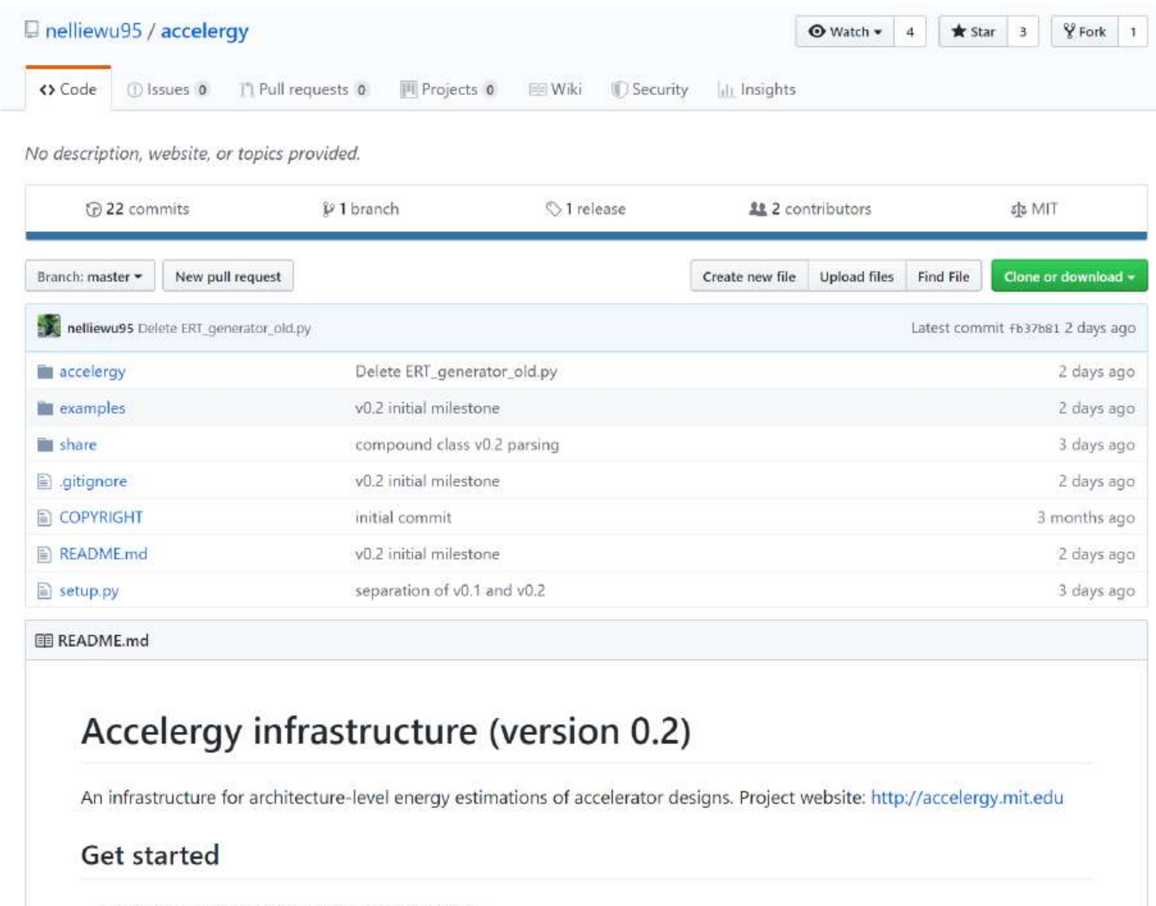
Integrate 3rd-party tools to generate the count of each operation



Tutorial at MICRO 2019: http://accelergy.mit.edu/tutorial.html

Energy Estimation Tool V2 - Accelergy

Website: https://accelergy.mit.edu/

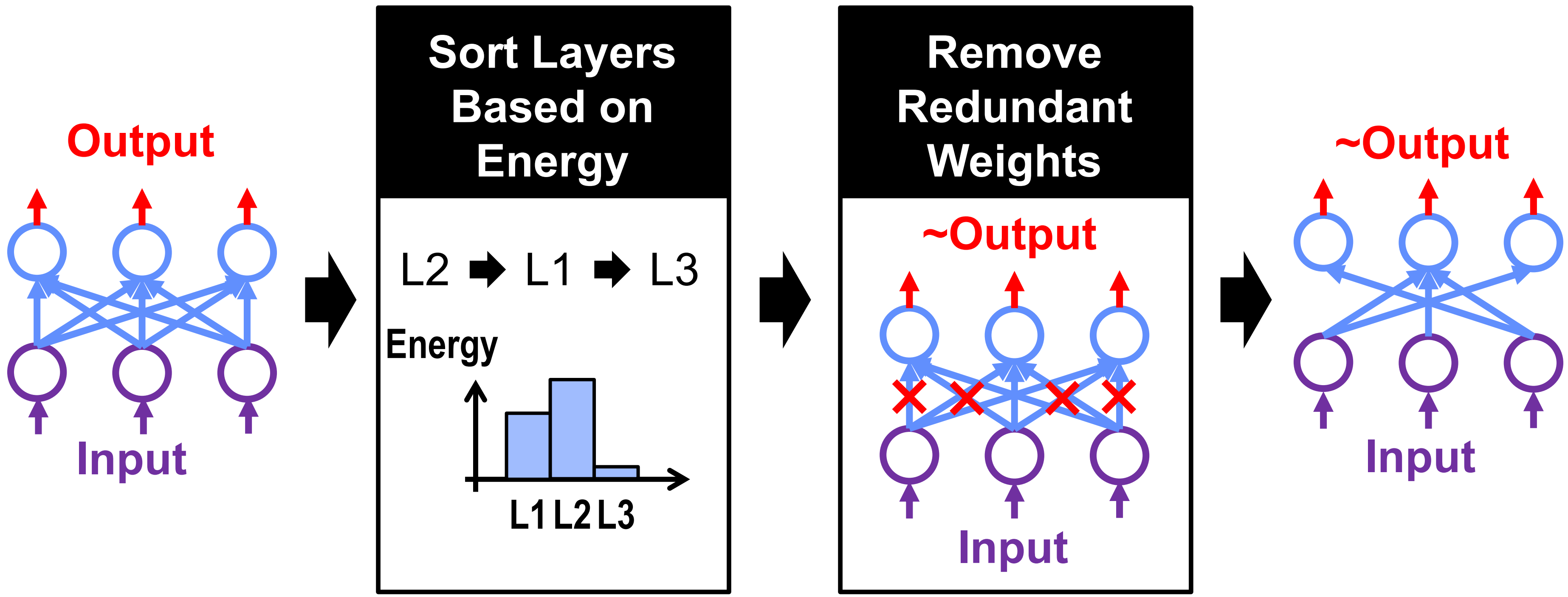


Output DNN energy breakdown across components

```
hierarchy.PE[0].ifmap_sp: 140.0
hierarchy.PE[0].mac[0]: 70.0
hierarchy.PE[0].mac[1]: 70.0
hierarchy.PE[1].ifmap_sp: 180.0
hierarchy.PE[1].mac[0]: 70.0
hierarchy.PE[1].mac[1]: 70.0
hierarchy.weights_glb: 5400.0
```

Energy-Aware Pruning

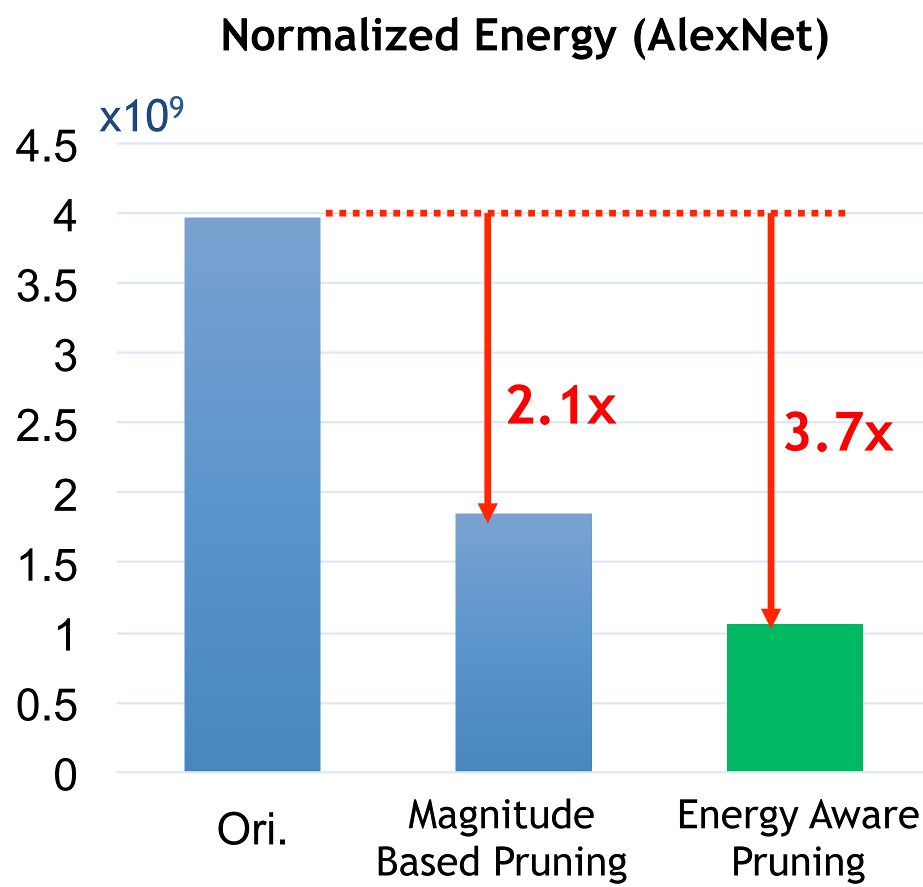
- Problem formulation: $\min_{Net} Erg(Net) \;\; subject\ to\ Acc(Net) \geq Th$
- Reduces energy by **removing redundant weights**
- Uses **estimated energy** to guide the layer-by-layer pruning
  - Prunes the layer that consume the most energy first



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**

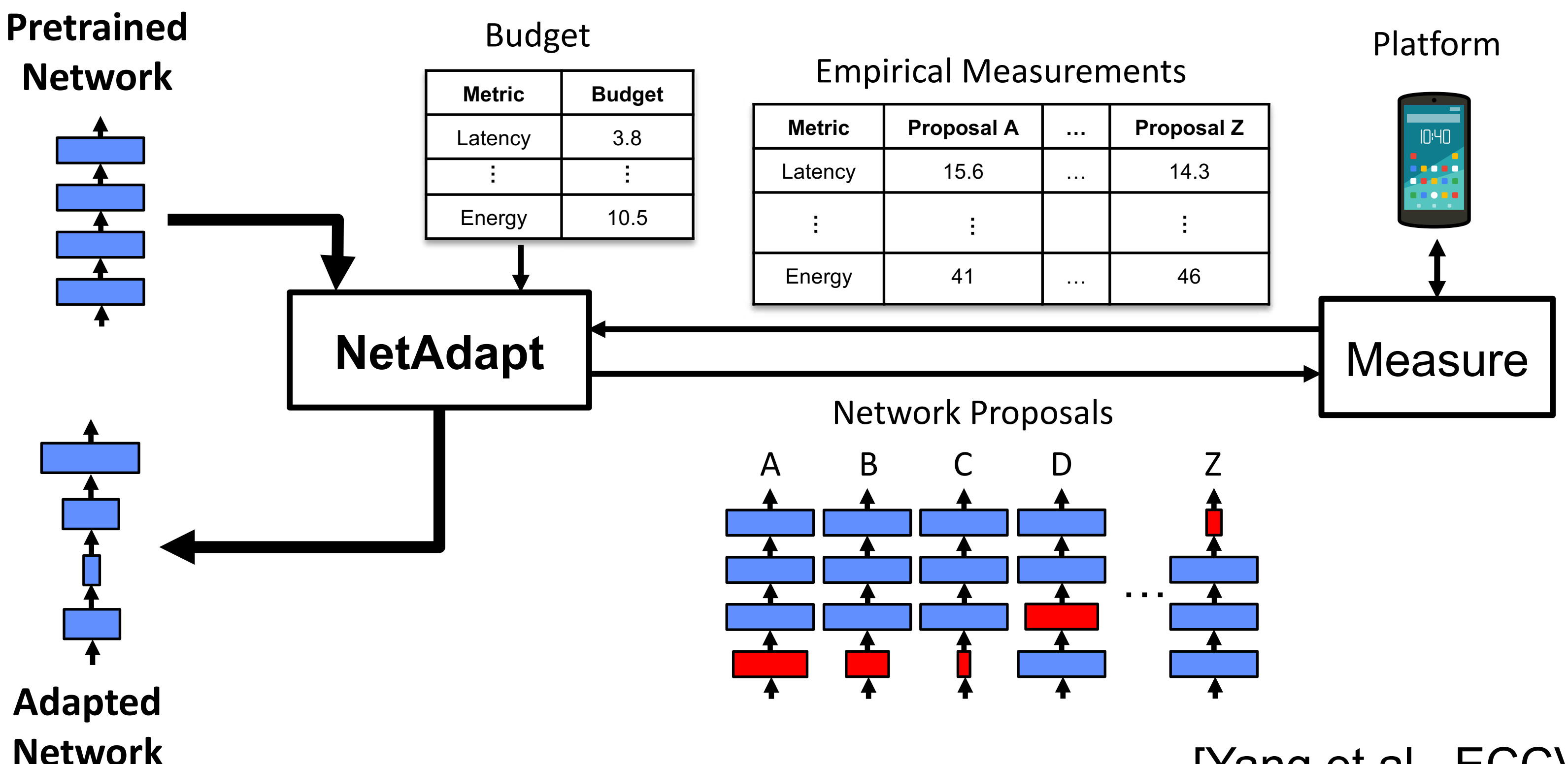


Pruned models available at
http://eyeriss.mit.edu/energy.html

[Yang et al., CVPR 2017]

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)



Metric	Budget
Latency	3.8
⋮	⋮
Energy	10.5

Metric	Proposal A	…	Proposal Z
Latency	15.6	…	14.3
⋮	⋮		⋮
Energy	41	…	46

[Yang et al., ECCV 2018]

In collaboration with Google's Mobile Vision Team

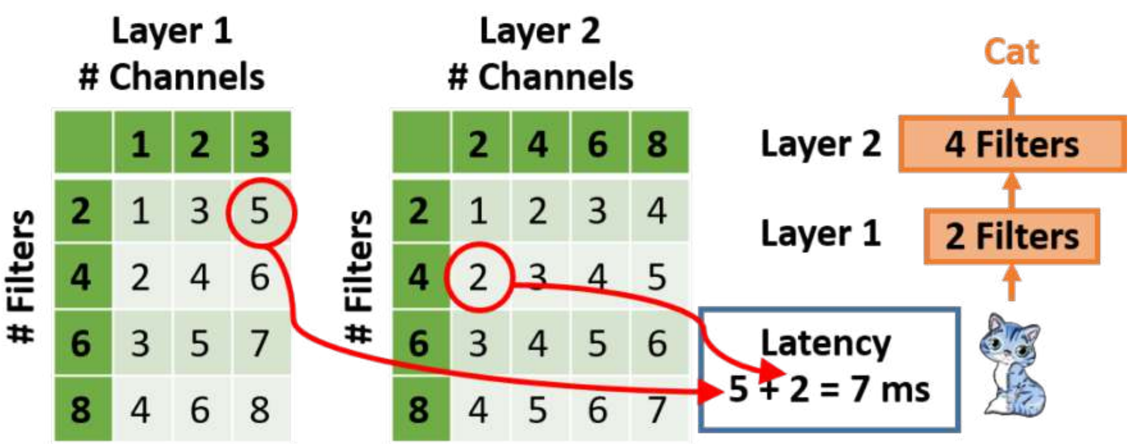
NetAdapt: Using Direct Metrics is Important

- If NetAdapt was guided by the number of MACs, it would also achieve a better accuracy-MAC trade-off
- However, it does not mean lower latency
- It is important to incorporate direct metrics rather than indirect metrics into the design of DNNs

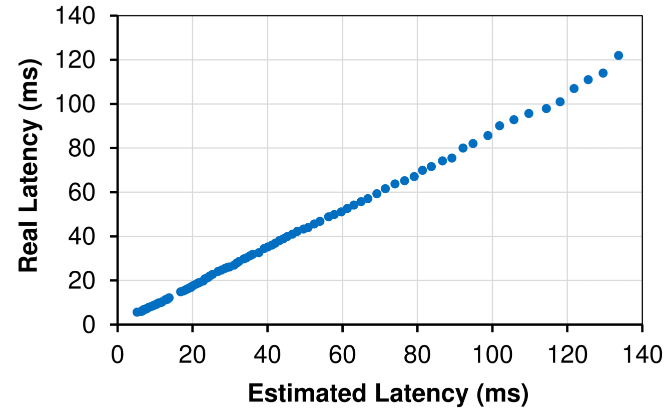
Network	Top-1 Accuracy	# of MACs (M)	Latency (ms)
Small MobileNet V1	45.1 (+0)	13.6 (100%)	4.65 (100%)
NetAdapt	46.3 (+1.2)	11.0 (81%)	6.01 (129%)
Large MobileNet V1	68.8 (+0)	325.4 (100%)	69.3 (100%)
NetAdapt	69.1 (+0.3)	284.3 (87%)	74.9 (108%)

NetAdapt: Fast Resource Consumption Estimation

- Taking measurements can be slow due to the long turn-around time and the limited number of platforms
- Solution: use per-layer lookup tables
    - The network latency can be estimated by the sum of the latency of each layer
    - The layers with the same configuration only need to be measured once
    - The network-wise lookup table grows exponentially with the number of layers



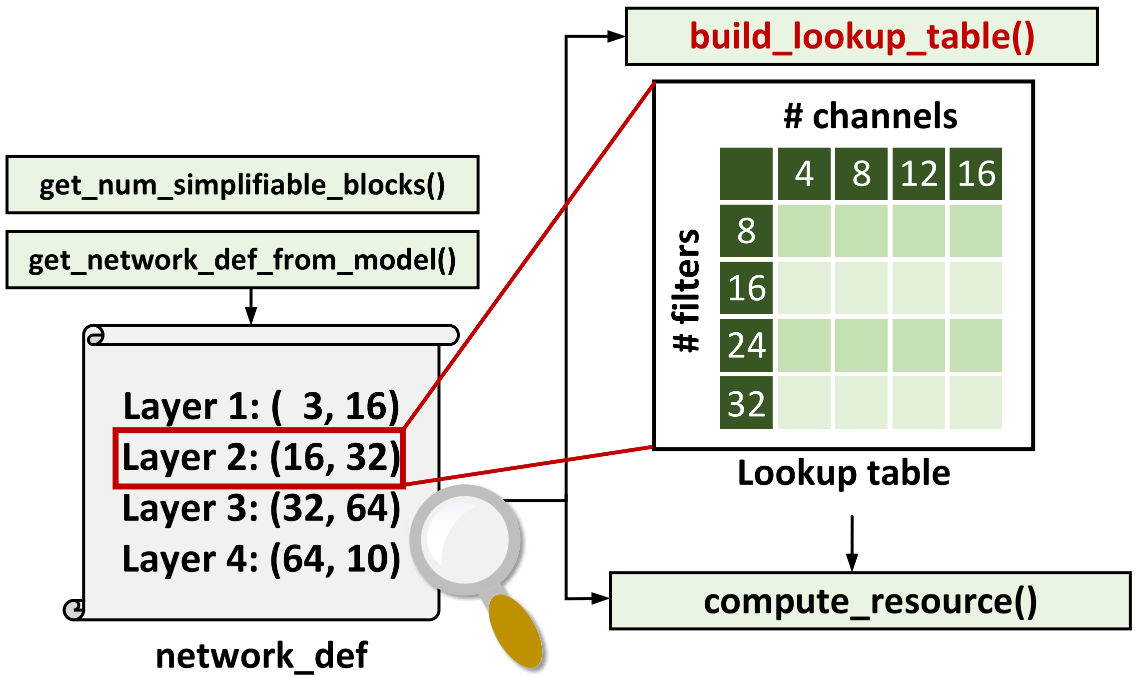
Fast Resource Consumption Estimation



Real Latency vs. Estimated Latency

NetAdapt: Code

- Support building and using lookup tables



Code available at https://github.com/denru01/netadapt

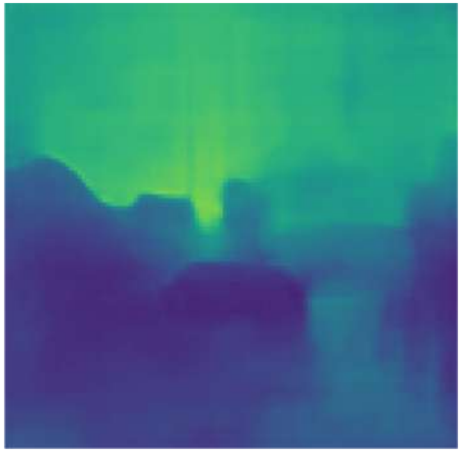
Part 3: Applications (Beyond Image Classification)

FastDepth: Fast Monocular Depth Estimation

- Real-time low-power depth sensing is critical for navigation of small robotic vehicles.
- Depth estimation from a single RGB image desirable, due to the relatively low cost and size of monocular cameras.

RGB

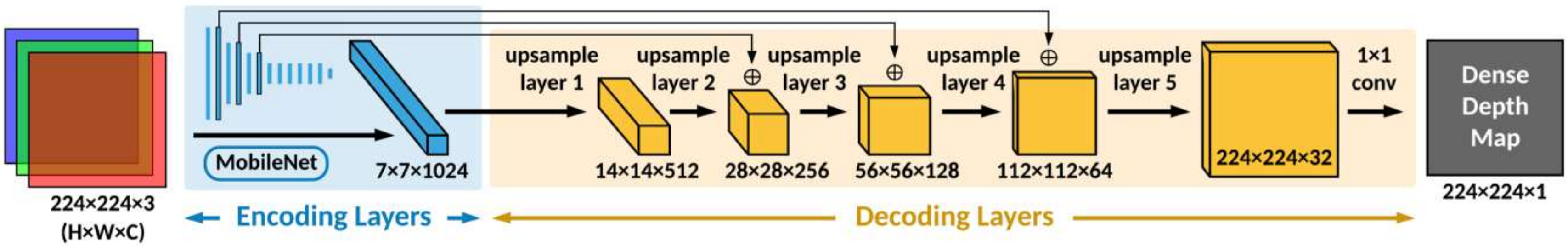
Prediction



Our goal is to enable **high accuracy, low latency, high throughput monocular depth estimation** on a **deployable embedded system**.

[Wofk*, Ma* et al., ICRA 2019]

Efficient Network Design for FastDepth



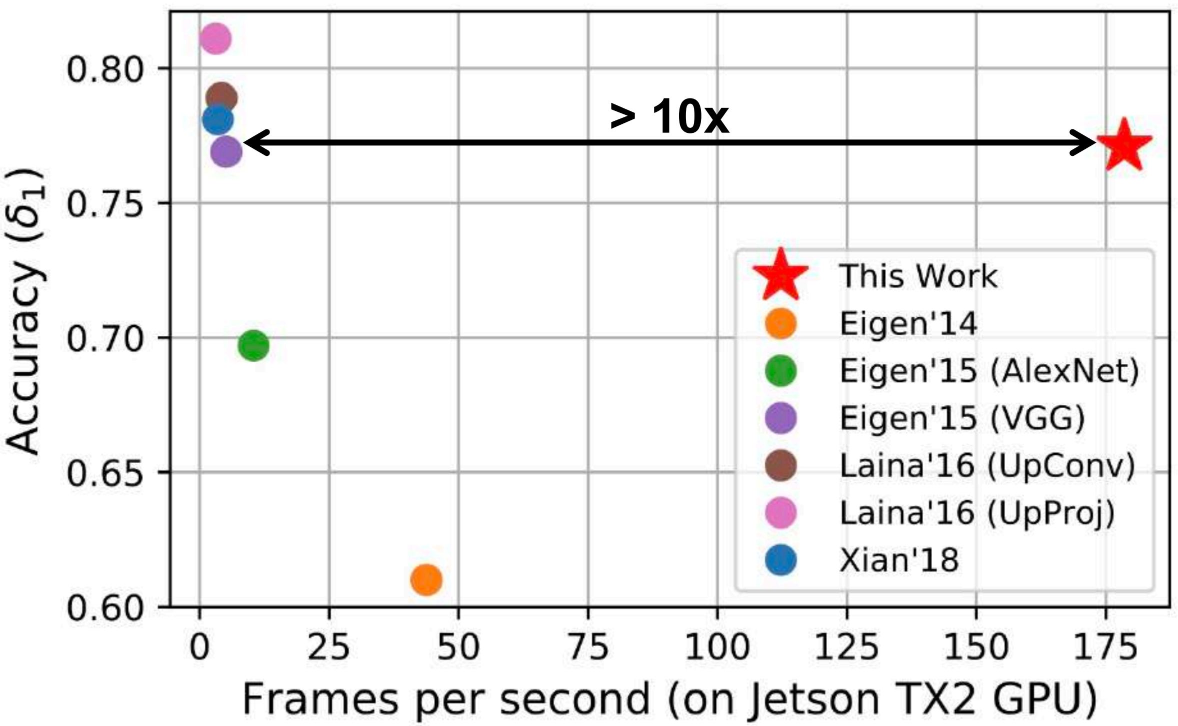
FastDepth achieves high frame rates through

- An efficient and lightweight encoder-decoder network architecture with a low-latency decoder design incorporating depthwise separable layers and additive skip connections
- Network pruning (NetAdapt) applied to whole encoder-decoder network
- Platform-specific compilation (TVM) targeting embedded systems

[Wofk*, Ma* et al., ICRA 2019]

FastDepth: Fast Monocular Depth Estimation

Depth estimation at **high frame rates on an embedded platform** (an order of magnitude faster than previous approaches) while still maintaining accuracy



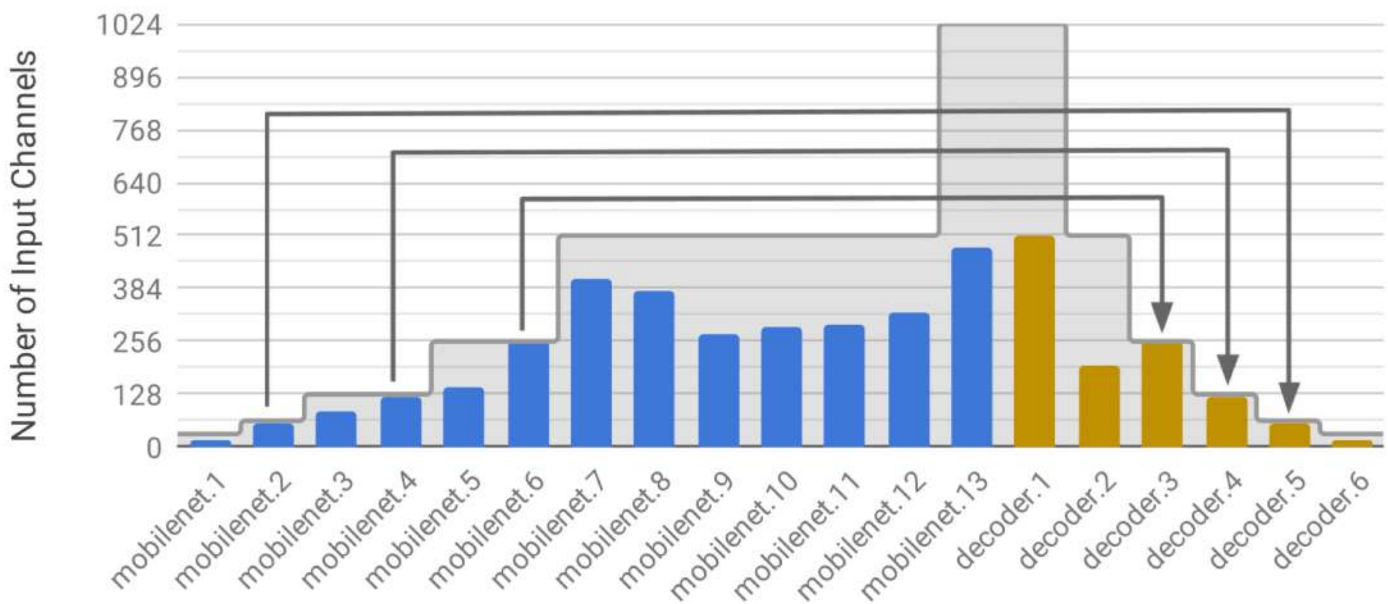
Configuration: Batch size of one (32-bit float)

~40fps on an iPhone

Models available at *http://fastdepth.mit.edu*

Simplify Network by NetAdapt

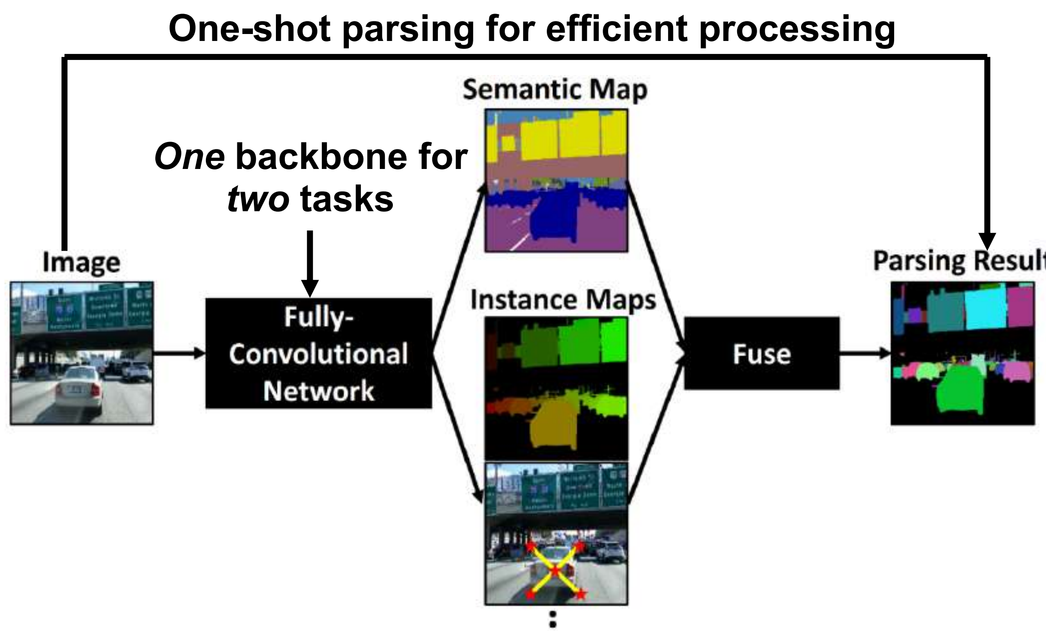
	Before Pruning	After Pruning	Reduction
Weights	3.93M	1.34M	2.9×
MACs	0.74G	0.37G	2.0×
RMSE	0.599	0.604	-
δ_1	0.775	0.771	-
CPU [ms]	66	37	1.8×
GPU [ms]	8.2	5.6	1.5×



DeeperLab: Single-Shot Image Parser

Results from Xception

Joint Semantic and Instance Segmentation (high resolution input image)



Fully convolutional, one-shot parsing
(bottom-up approach)

http://deeperlab.mit.edu/

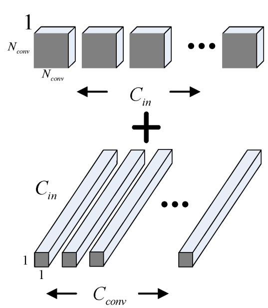
[Yang et al., arXiv 2019]

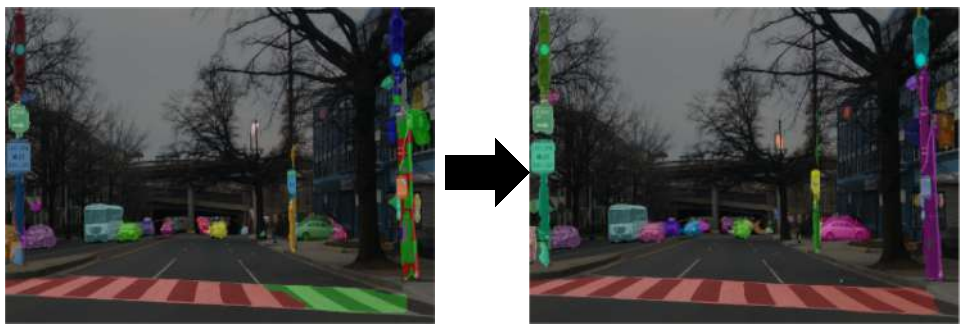
DeeperLab: Efficient Image Parsing

Address memory requirement for large feature map

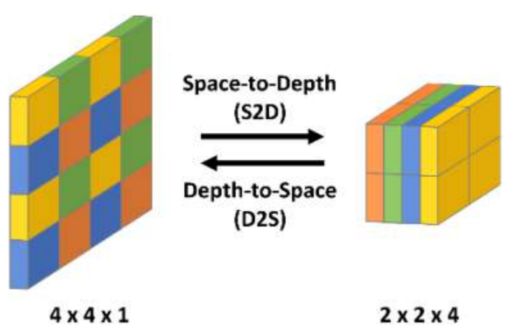
1 Wide MobileNet: Increase kernel size rather than depth

3x3 → 5x5





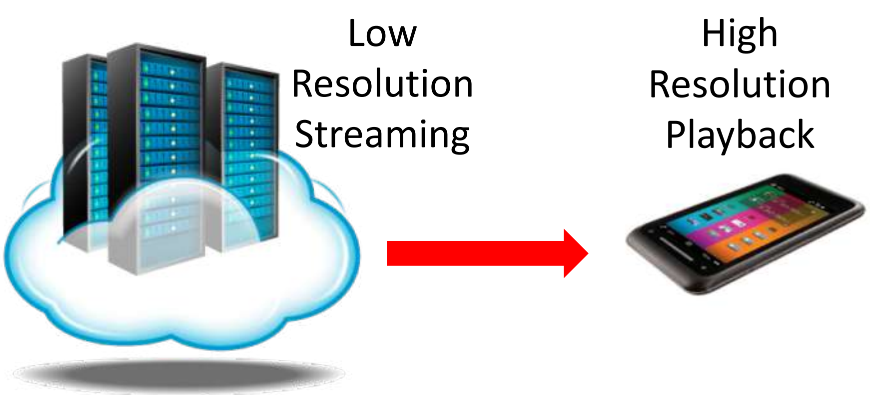
2 Space-to-depth/depth-to-space: Avoid upsampling

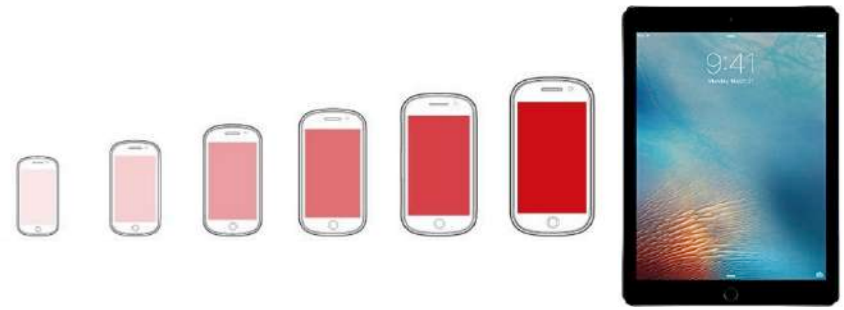


Achieves near real-time 6.19 fps on GPU (V100) with 25.2% PQ and 49.8% PC on Mapillary Vistas dataset

Applications (Beyond DNN Acceleration)

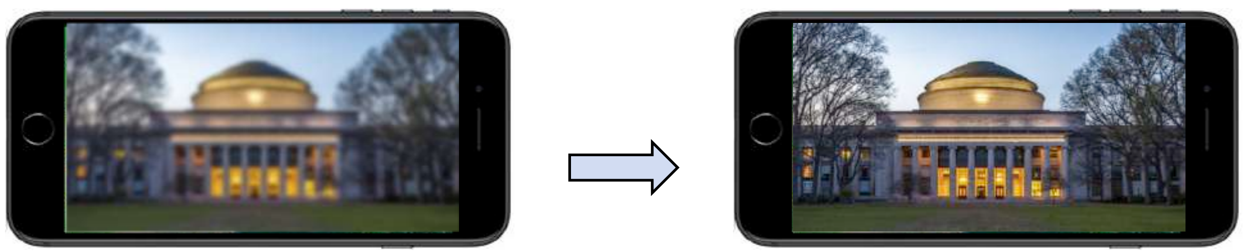
Super-Resolution on Mobile Devices





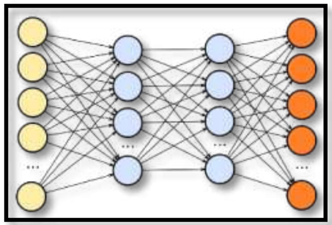
Transmit low resolution for lower bandwidth

Screens are getting larger



Use **super-resolution** to improve the viewing experience of lower-resolution content (*reduce communication bandwidth*)

FAST: A Framework to Accelerate SuperRes



SR algorithm

Compressed video

FAST

SR
15x faster

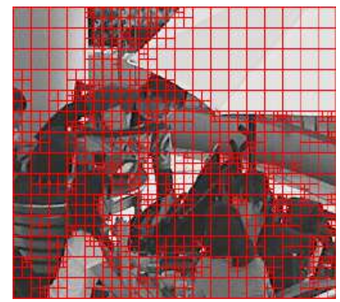
Real-time

A framework that accelerates **any SR** algorithm by up to **15x** when running on compressed videos

[Zhang et al., CVPRW 2017]

Free Information in Compressed Videos

Decode



Compressed video

Pixels

Block-structure

Motion-compensation

Video as a stack of pixels

Representation in compressed video

This representation can help **accelerate** super-resolution

Transfer is Lightweight

Transfer allows SR to run on only **a subset of frames**

Fractional Interpolation **Bicubic Interpolation**

Skip Flag

The complexity of the transfer is comparable to bicubic interpolation.
Transfer **N** frames, accelerate by **N**

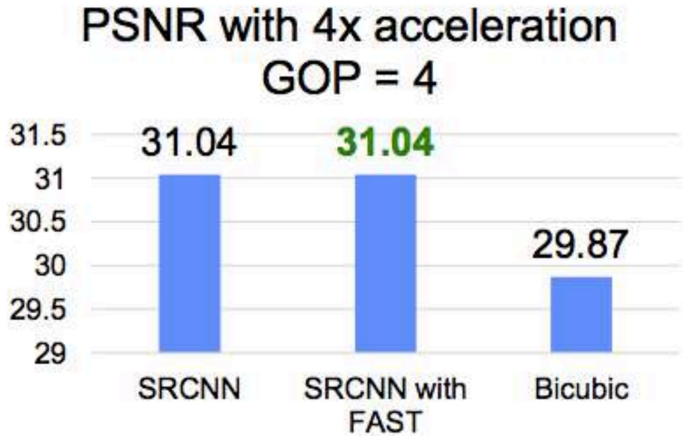
Evaluation: Accelerating SRCNN

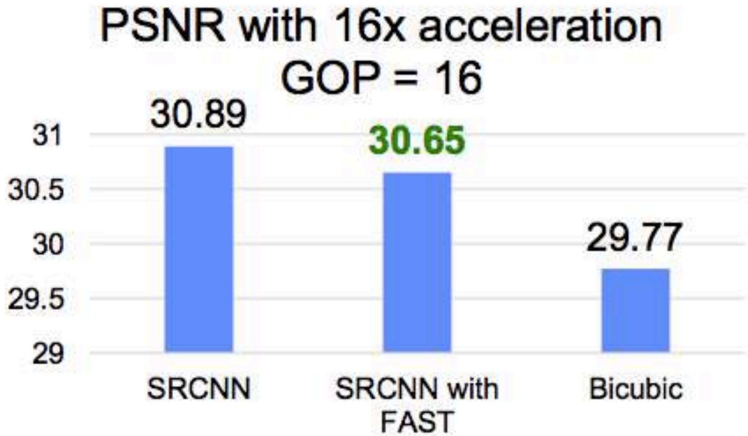

PartyScene


RaceHorse


BasketballPass

Examples of videos in the test set (20 videos for HEVC development)





4 × acceleration with NO PSNR LOSS. 16 × acceleration with 0.2 dB loss of PSNR

Visual Evaluation

SRCNN **FAST + SRCNN** **Bicubic**

Look ***beyond*** the DNN accelerator for opportunities to accelerate DNN processing (e.g., structure of data and temporal correlation)

Code released at www.rle.mit.edu/eems/fast

[Zhang et al., CVPRW 2017]

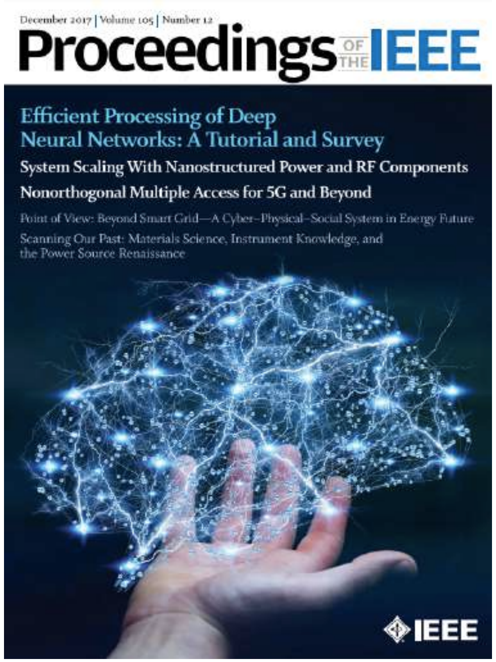
Summary

- **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's important to consider a **comprehensive set of metrics** when evaluating different DNN solutions: **accuracy, speed, energy, and cost**

Additional Resources

Overview Paper
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!



More info about **Tutorial on DNN Architectures**
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

- **Overview on DNN and Popular DNN Models**

  - *Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift," ICML 2015.*
  - ***LeNet**: LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proc. IEEE 1998.*
  - ***AlexNet**: Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS. 2012.*
  - ***VGGNet**: Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR 2015.*
  - ***GoogleNet**: Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR 2015.*
  - ***ResNet**: He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR 2016.*
  - ***DenseNet**: Huang, Gao, et al. "Densely connected convolutional networks." CVPR 2017*
  - ***Wide ResNet**: Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." BMVC 2017.*
  - ***ResNext**: Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." CVPR 2017.*

References

- **Part 1: Energy-Efficient Hardware for Deep Neural Networks**
  - **Project website:** http://eyeriss.mit.edu
  - *Y.-H. Chen, T. Krishna, J. Emer, V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE Journal of Solid State Circuits (JSSC), ISSCC Special Issue, Vol. 52, No. 1, pp. 127-138, January 2017.*
  - *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), pp. 367-379, June 2016.*
  - *Y.-H. Chen, T.-J. Yang, J. Emer, V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS), June 2019.*
  - *Eyexam:* https://arxiv.org/abs/1807.07928
- **Limitations of Existing Efficient DNN Approaches**
  - *Y.-H. Chen*, T.-J. Yang*, J. Emer, V. Sze, "Understanding the Limitations of Existing Energy-Efficient Design Approaches for Deep Neural Networks," SysML Conference, February 2018.*
  - *V. Sze, Y.-H. Chen, T.-J. Yang, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, December 2017.*
  - *Hardware Architecture for Deep Neural Networks:* http://eyeriss.mit.edu/tutorial.html

References

- **Transforms for processing on GPU and CPUs**

  - *Lavin, Andrew, and Gray, Scott, "Fast Algorithms for Convolutional Neural Networks," arXiv preprint arXiv:1509.09308 (2015)*
  - *Mathieu, Michael, Mikael Henaff, and Yann LeCun. "Fast training of convolutional networks through FFTs." arXiv preprint arXiv:1312.5851 (2013).*
  - *Cong, Jason, and Bingjun Xiao. "Minimizing computation in convolutional neural networks." International Conference on Artificial Neural Networks. Springer International Publishing, 2014.*

- **Part 2: Co-Design of Algorithms and Hardware for Deep Neural Networks**

  - *T.-J. Yang, Y.-H. Chen, V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.*
  - *Energy estimation tool:* http://eyeriss.mit.edu/energy.html
  - *T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, V. Sze, H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," European Conference on Computer Vision (ECCV), 2018. http://netadapt.mit.edu*
  - *T.-J. Yang, V. Sze, "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators," IEEE International Electron Devices Meeting (IEDM), Invited Paper, December 2019.*
  - *Y. N. Wu, J. S. Emer, V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," International Conference on Computer Aided Design (ICCAD), November 2019. http://accelergy.mit.edu*
  - *T.-J. Yang, Y.-H. Chen, J. Emer, V. Sze, "A Method to Estimate the Energy Consumption of Deep Neural Networks," Asilomar Conference on Signals, Systems and Computers, Invited Paper, October 2017.*

References

- **Reduced Precision**
  - *Courbariaux, Matthieu, and Yoshua Bengio. "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).*
  - *Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations," NeurIPS, 2015*
  - *Rastegari, Mohammad, et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," ECCV, 2016*
  - *Judd, Patrick, Jorge Albericio, and Andreas Moshovos. "Stripes: Bit-serial deep neural network computing." IEEE Computer Architecture Letters (2016).*
  - *Lee, Edward H., et al. "LogNet: Energy-efficient neural networks using logarithmic computation." 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2017.*
  - *Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," ICLR, 2016.*

References

- **Exploit Sparsity**
  - *LeCun, Yann, et al. "Optimal brain damage," NIPS, 1989.*
  - *Han, Song, et al. "Learning both weights and connections for efficient neural network," NeurIPS, 2015.*
  - *T.-J. Yang, Y.-H. Chen, V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.*
  - *Parashar, Angshuman, et al. "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks." ISCA, 2017*
  - *Han, Song, et al. "EIE: efficient inference engine on compressed deep neural network," ISCA, 2016.*
  - *Y.-H. Chen, T.-J. Yang, J. Emer, V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS), June 2019.*
- **Manual Network Design**
  - *Network in Network: Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." ICLR 2014*
  - ***MobileNet**: Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).*
  - ***ShuffleNet**: Zhang, Xiangyu, et al. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices." arXiv preprint arXiv:1707.01083 (2017).*
  - *Yu, Fisher, et al. "Deep layer aggregation." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.*

Reference

- **Neural Architecture Search**
  - *Learning Network Architecture: Zoph, Barret, et al. "Learning Transferable Architectures for Scalable Image Recognition." arXiv preprint arXiv:1707.07012 (2017).*
  - *T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, V. Sze, H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," European Conference on Computer Vision (ECCV), 2018. http://netadapt.mit.edu*
- **Hardware In the Loop**
  - *T.-J. Yang, Y.-H. Chen, V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.*
  - *Energy estimation tool:* http://eyeriss.mit.edu/energy.html
  - *T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, V. Sze, H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," European Conference on Computer Vision (ECCV), 2018. http://netadapt.mit.edu*
  - *T.-J. Yang, V. Sze, "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators," IEEE International Electron Devices Meeting (IEDM), Invited Paper, December 2019.*
  - *Y. N. Wu, J. S. Emer, V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," International Conference on Computer Aided Design (ICCAD), November 2019. http://accelergy.mit.edu*
  - *T.-J. Yang, Y.-H. Chen, J. Emer, V. Sze, "A Method to Estimate the Energy Consumption of Deep Neural Networks," Asilomar Conference on Signals, Systems and Computers, Invited Paper, October 2017.*

References

- **Part 3: Applications Beyond Image Classification**

  – *D. Wofk*, F. Ma*, T.-J. Yang, S. Karaman, V. Sze, "FastDepth: Fast Monocular Depth Estimation on Embedded Systems," IEEE International Conference on Robotics and Automation (ICRA), May 2019.* *http://fastdepth.mit.edu/*

  – *T.-J. Yang, M. D. Collins, Y. Zhu, J.-J. Hwang, T. Liu, X. Zhang, V. Sze, G. Papandreou, L.-C. Chen, "DeeperLab: Single-Shot Image Parser," arXiv, February 2019.* *http://deeperlab.mit.edu*

  – Z. Zhang, V. Sze, "FAST: A Framework to Accelerate Super-Resolution Processing on Compressed Videos," *CVPR Workshop on New Trends in Image Restoration and Enhancement*, July 2017. *www.rle.mit.edu/eems/fast*