# DNN Accelerator Design Optimization and NoC-based DNN

*Kun-Chih (Jimmy) Chen 陳坤志*

kcchen@nycu.edu.tw

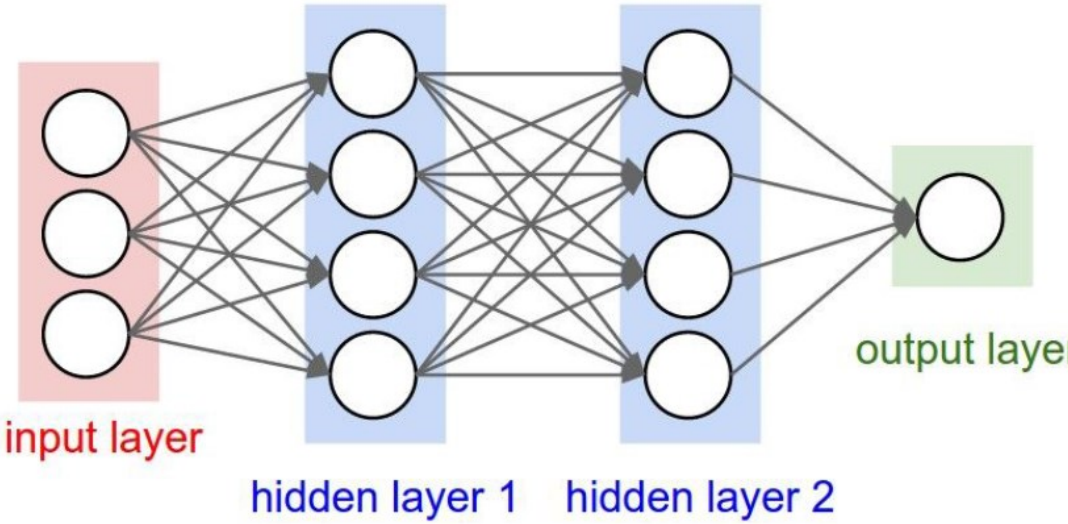
*Institute of Electronics,*

*National Yang Ming Chiao Tung University*

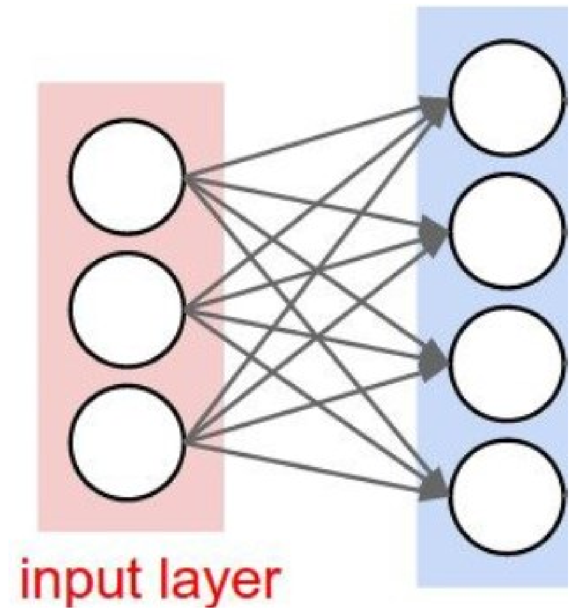
# Complexity in ANN (1/2)

- One layer of fully-connected (FC) ANN layer can be represented by an n x m matrix containing weights in links from m input neurons to n output neurons
  - e.g., 3 input neurons, 4 output neurons => 4x3 weight matrix plus 4x1 vector of biases
- In general, there are hundreds (thousands) input/output neurons, and several FC ANN layers (for classification)
  - FC-1 in VGG-16: input neurons: 7*7*512=25088; output neurons: 4090 => 102,760,000 parameters
  - FC-2 in VGG-16: input neurons: 4096; output neurons: 4090 => 16,777,216 parameters
  - FC3 in VGG-16: input neurons: 4090; output neurons: 1000 => 4,096,000 parameters
  - Total number of parameters in VGG-16: 138M => about 90% of parameters in FC layers

# Complexity in ANN (2/2)




total # of weights (links): 3x4+4x4+4x1=32
total # of biases: 4+4+1= 9
total # of operations (MAC): 3x4+4x4+4x1=32

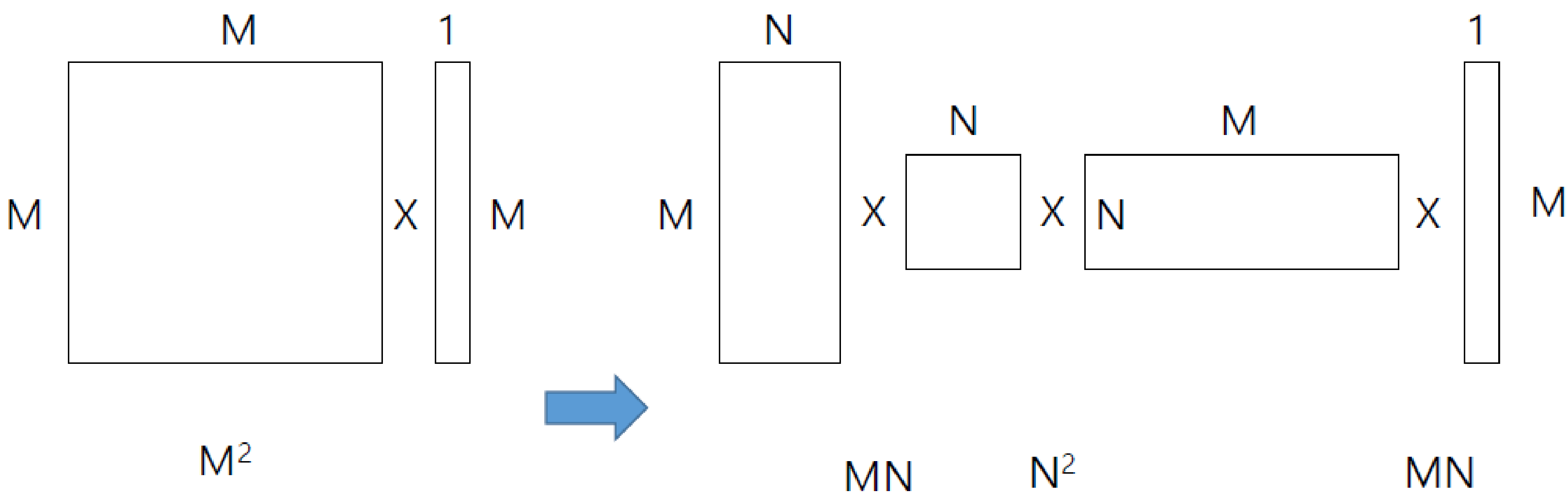



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \underbrace{\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}}_{M} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

total # of weights (links): 3x4=12
total # of biases: 4
total # of operations (MAC): 3x4=12

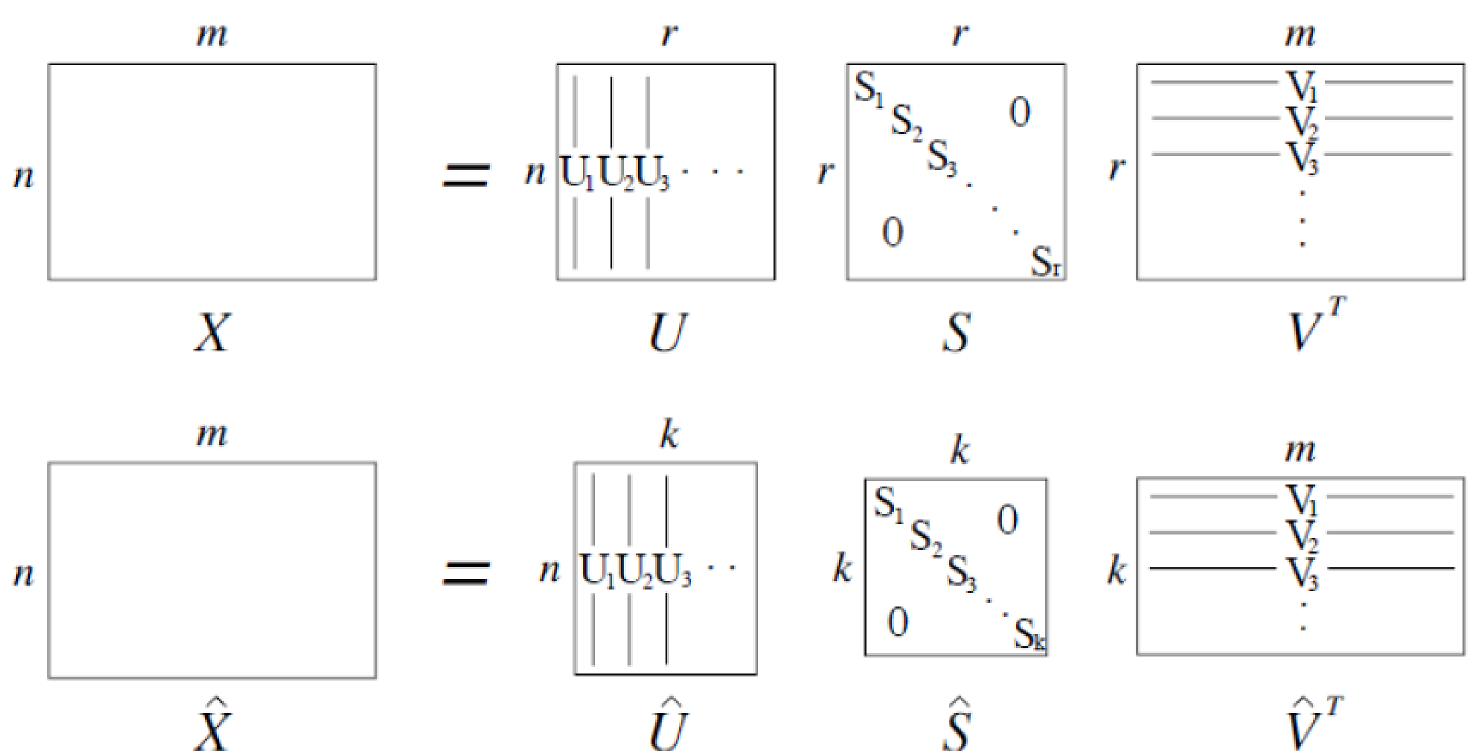
# Reduced Computations

- MM x M → MN x NN x NM x M
- # of multiplications: $M^2 \rightarrow 2MN + N^2$
  - i.e., M=100, N=20
  - 10000 → 4400 (56% reduction)



# Low-rank approximation: truncated SVD

- Fully connected (FC) layers can be represented by matrix-vector product
  - Matrix containing weights can be decomposed using SVD
  - Reduce # of weights by selecting only the largest singular values



# Neural Network Model Compression

- Selecting first 500 (out of 4096) largest singular values for Fully-Connected layer FC6 with 14x14x128 input neurons and 4096 output neurons
- 2/3 of total weights are cut off after SVD decomposition

$$f^{out} = W f^{in} + b \Rightarrow W \approx U_d S_d V_d \quad U_d \in \mathbb{R}^{n_{out} \times d},\ V_d \in \mathbb{R}^{d \times n_{in}} \quad S_d \in \mathbb{R}^{d \times d}$$

$$O(n_{in} n_{out}) \Rightarrow O(d n_{in} + d n_{out}) \quad d \ll n_{in}, n_{out}$$

| Network | FC6 | # of total weights | # of operations | Top-5 accuracy |
|---|---|---|---|---|
| **VGG-16** | 25088×4096 | 138.36M | 30.94G | 88.00% |
| **VGG-16-SVD** | 25088×500 + 500×4096 | 50.18M | 30.76G | 87.96% |


VGG16

# Why do we need DNN model compression?

- **Efficient deployment**
  - Efficient deployment of neural networks on devices with limited resources
  - Faster inference and reduced energy consumption
- **Reduce cost**
  - Smaller models can reduce the cost of hardware and infrastructure needed to train and deploy neural networks
- **Energy efficiency**
  - Model compression can reduce the power consumption of DNN models, which is essential for power resource-limited edge devices.

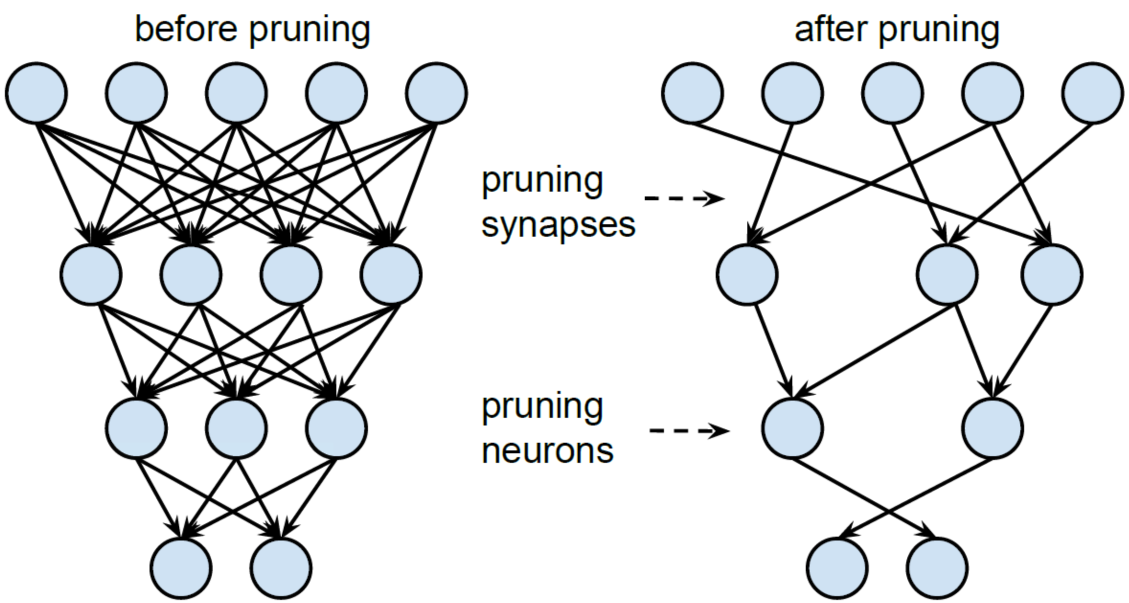
# Techniques of DNN Model Compression

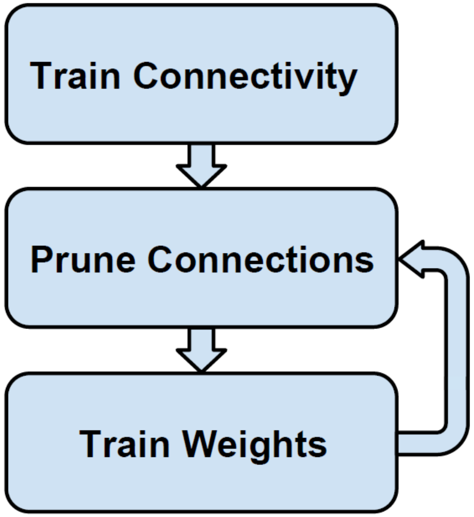
- **Pruning**
  - Removing the least important connections, neurons, or filters from the network → Making it smaller and faster.
- **Quantization**
  - Reducing the precision of the weights and activations in the network. → Reducing the memory requirements and computational complexity of the model.
- **Weight Sharing**
  - Sharing weights between multiple layers or filters in the network. → Reducing the number of unique weights in the model, making it smaller and faster.
- **Depthwise Separable Convolution**
  - Each channel is convolved with its own set of filters, rather than all channels being convolved with the same filter → Reducing the number of parameters needed in convolution layers.

# Pruning

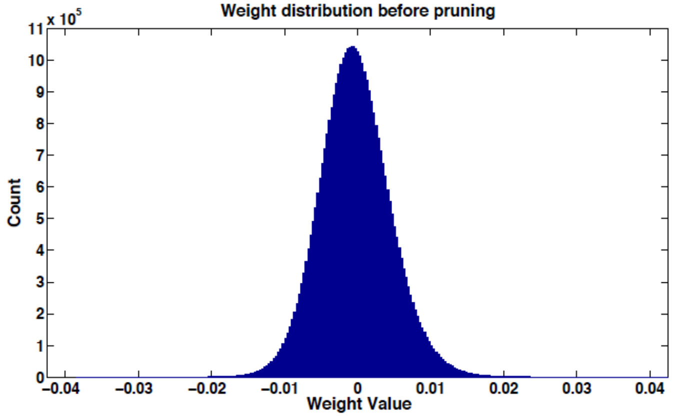
# Pruning Neural Network

- Removing weights of small magnitudes recursively
  - Remove connections with weight below a per-layer adjustable threshold
    - threshold chosen as a quantity parameter multiplied by standard deviation of a layer's weights
  - Retrain dropout ratio should be smaller

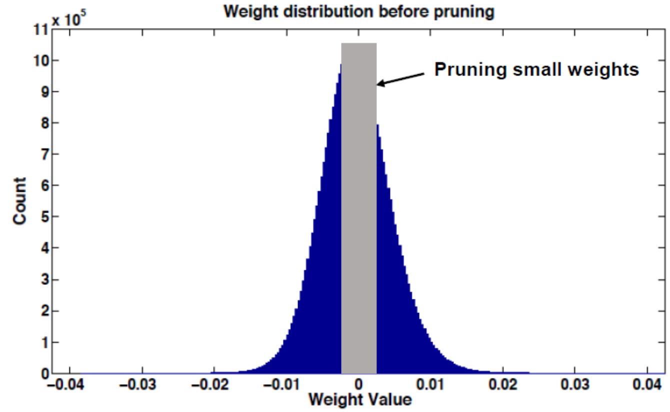


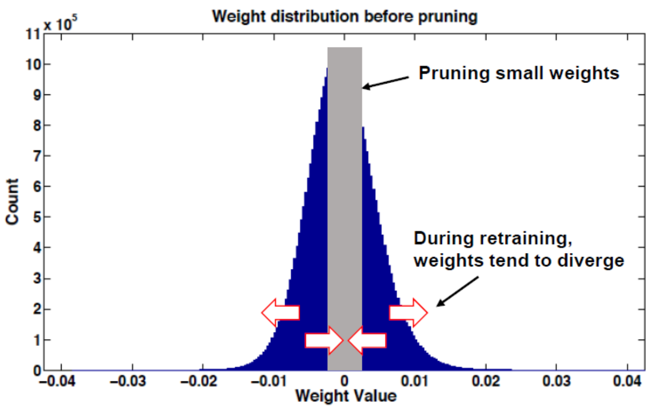
# Illustration of Iterative Pruning-Retraining



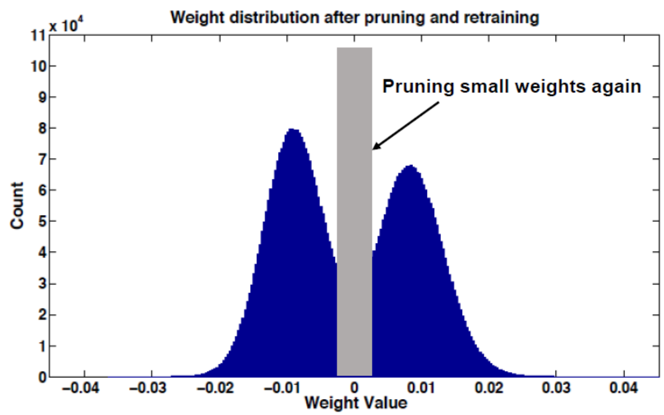

weight distribution before pruning




small weights are set to zero (pruned)




during training, weights are re-distributed




**10X smaller scale**

after a step of training, new small weights are obtained

[Han, 2015]

# Accuracy, Compression Ratio and # Bits (Quantization Levels)

| Network | Top-1 Error | Top-5 Error | Parameters | Compress Rate |
|---|---|---|---|---|
| LeNet-300-100 Ref | 1.64% | - | 1070 KB | |
| LeNet-300-100 Compressed | 1.58% | - | **27 KB** | **40**× |
| LeNet-5 Ref | 0.80% | - | 1720 KB | |
| LeNet-5 Compressed | 0.74% | - | **44 KB** | **39**× |
| AlexNet Ref | 42.78% | 19.73% | 240 MB | |
| AlexNet Compressed | 42.78% | 19.70% | **6.9 MB** | **35**× |
| VGG-16 Ref | 31.50% | 11.32% | 552 MB | |
| VGG-16 Compressed | 31.17% | 10.91% | **11.3 MB** | **49**× |

AlexNet

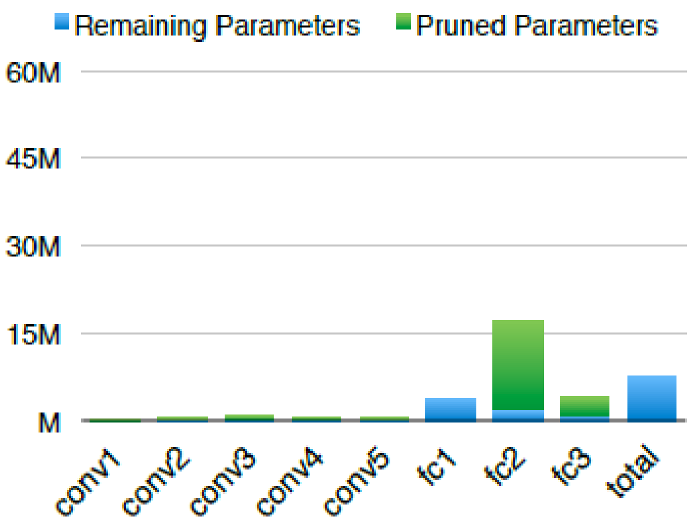
| #CONV bits / #FC bits | Top-1 Error | Top-5 Error | Top-1 Error Increase | Top-5 Error Increase |
|---|---|---|---|---|
| 32bits / 32bits | 42.78% | 19.73% | - | - |
| 8 bits / 5 bits | 42.78% | 19.70% | 0.00% | -0.03% |
| 8 bits / 4 bits | 42.79% | 19.73% | 0.01% | 0.00% |
| 4 bits / 2 bits | 44.77% | 22.33% | 1.99% | 2.60% |

# AlexNet Pruning

- # of parameters from 61M to 6.7M (11%)
- # of operations from 1.5Gop to 0.45Gop (30%)
- More FC-layer weights removed cp. Conv layers
- Retrain with 1/100 of the original network's original learning rate
  - cp. with 1/10 learning rate in LeNet

**some colors missing below**

| Layer | Weights | FLOP | Act% | Weights% | FLOP% |
|---|---|---|---|---|---|
| conv1 | 35K | 211M | 88% | 84% | 84% |
| conv2 | 307K | 448M | 52% | 38% | 33% |
| conv3 | 885K | 299M | 37% | 35% | 18% |
| conv4 | 663K | 224M | 40% | 37% | 14% |
| conv5 | 442K | 150M | 34% | 37% | 14% |
| fc1 | 38M | 75M | 36% | 9% | 3% |
| fc2 | 17M | 34M | 40% | 9% | 3% |
| fc3 | 4M | 8M | 100% | 25% | 10% |
| Total | 61M | 1.5B | 54% | **11%** | **30%** |



# VGG-16 Pruning

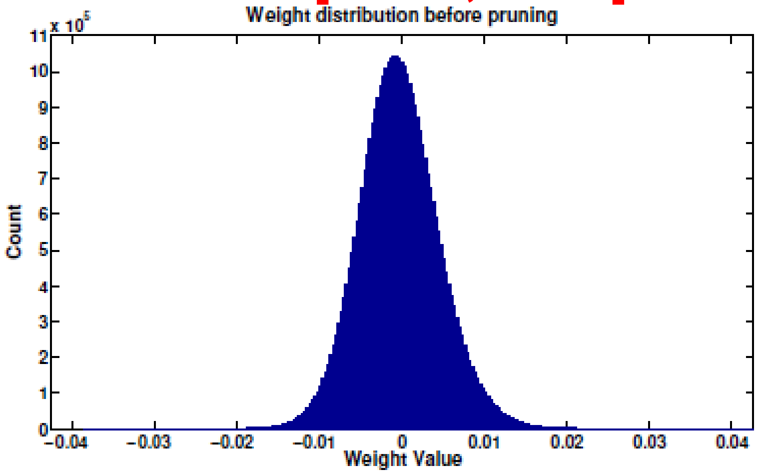
- # of parameters from 138M to 10.35M (7.5%)
- Operation count from 30.9 GOP to 6.5 GOP (21%)
- After pruning of AlexNet and VGG, weights can be stored on chip

| Layer | Weights | FLOP | Act% | Weights% | FLOP% |
|---|---|---|---|---|---|
| conv1_1 | 2K | 0.2B | 53% | 58% | 58% |
| conv1_2 | 37K | 3.7B | 89% | 22% | 12% |
| conv2_1 | 74K | 1.8B | 80% | 34% | 30% |
| conv2_2 | 148K | 3.7B | 81% | 36% | 29% |
| conv3_1 | 295K | 1.8B | 68% | 53% | 43% |
| conv3_2 | 590K | 3.7B | 70% | 24% | 16% |
| conv3_3 | 590K | 3.7B | 64% | 42% | 29% |
| conv4_1 | 1M | 1.8B | 51% | 32% | 21% |
| conv4_2 | 2M | 3.7B | 45% | 27% | 14% |
| conv4_3 | 2M | 3.7B | 34% | 34% | 15% |
| conv5_1 | 2M | 925M | 32% | 35% | 12% |
| conv5_2 | 2M | 925M | 29% | 29% | 9% |
| conv5_3 | 2M | 925M | 19% | 36% | 11% |
| fc6 | 103M | 206M | 38% | 4% | 1% |
| fc7 | 17M | 34M | 42% | 4% | 2% |
| fc8 | 4M | 8M | 100% | 23% | 9% |
| total | 138M | 30.9B | 64% | **7.5%** | **21%** |

# Comparison (AlexNet)

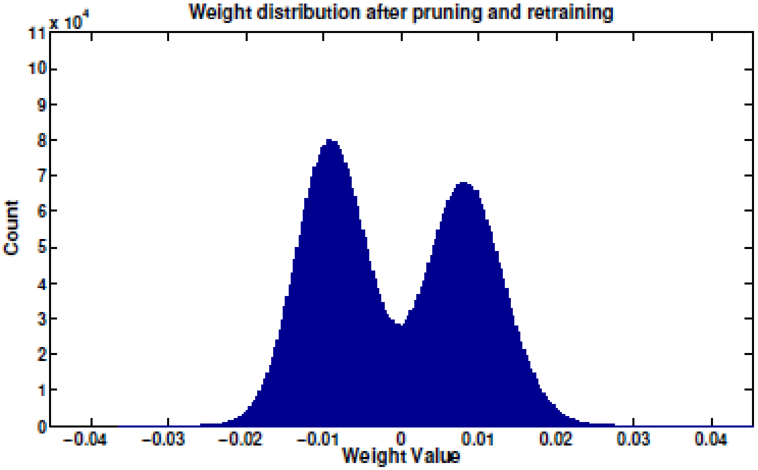
| Network | Top-1 Error | Top-5 Error | Parameters | Compression Rate |
|---|---|---|---|---|
| Baseline Caffemodel [26] | 42.78% | 19.73% | 61.0M | 1× |
| Data-free pruning [28] | 44.40% | - | 39.6M | 1.5× |
| Fastfood-32-AD [29] | 41.93% | - | 32.8M | 2× |
| Fastfood-16-AD [29] | 42.90% | - | 16.4M | 3.7× |
| Collins & Kohli [30] | 44.40% | - | 15.2M | 4× |
| Naive Cut | 47.18% | 23.23% | 13.8M | 4.4× |
| SVD [12] | 44.02% | 20.56% | 11.9M | 5× |
| **Network Pruning** | **42.77%** | **19.67%** | **6.7M** | 9× |

most in [-0.015, 0.015]




Weight distribution before parameter pruning

most in [-0.025, 0.025]



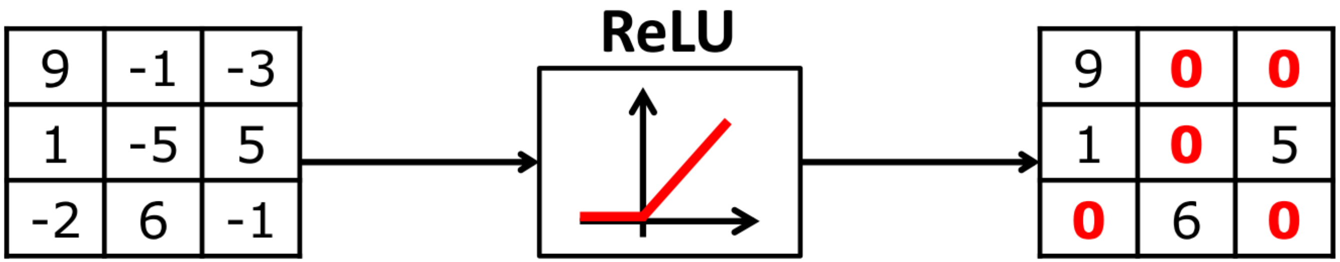

Weight distribution after parameter pruning

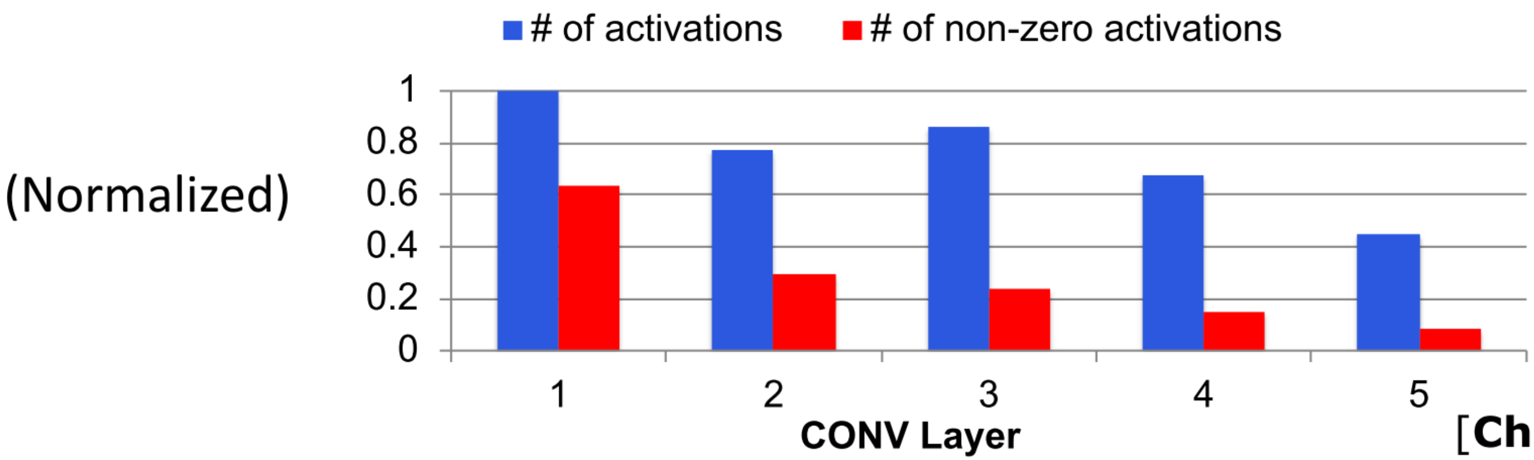
# Why Increase Sparsity?

- Reduce number of MACs
  - Anything multiplied by zero is zero → avoid performing unnecessary MACs
  - Reduce energy consumption and latency
- Reduce data movement
  - If one of the inputs to MAC is zero, can avoid reading the other input
  - Compress data by only sending non-zero values
- CPU/GPU libraries typically only support really high sparsity (> 99%) due to the overhead
  - Sparsity for DNNs typically much lower → need specialized hardware

# Sparsity in Activation Data

Many zeros in output fmaps after ReLU

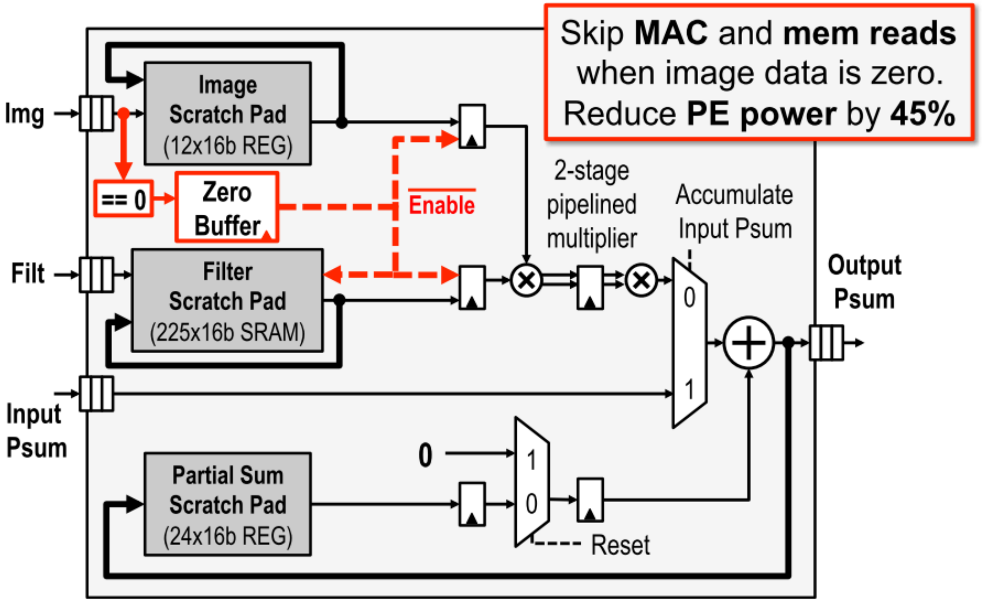



[**Chen**, *ISSCC* 2016]

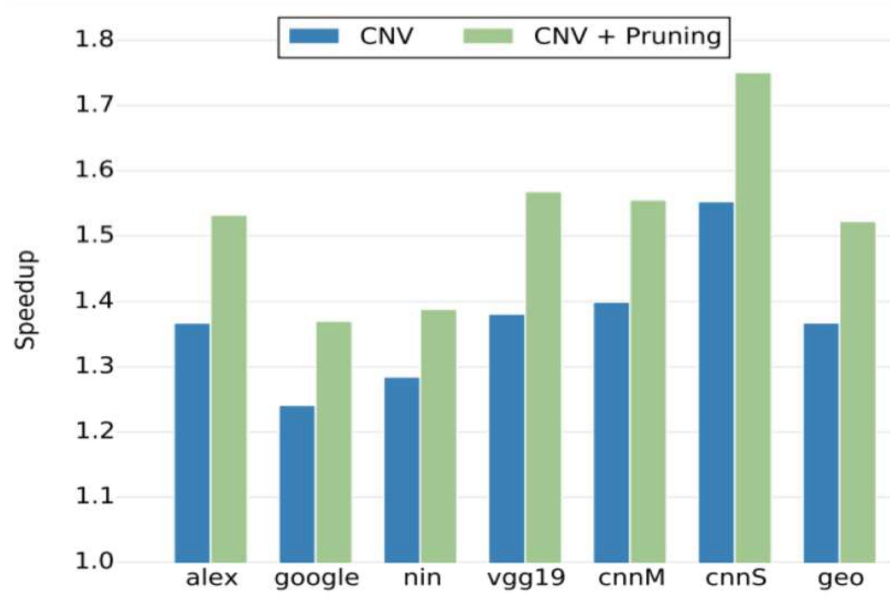
# Data Gating / Zero Skipping

## Gate operations
(reduce power consumption)




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

## Skip operations
(increase throughput)




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

# Unstructured or Structured Sparsity

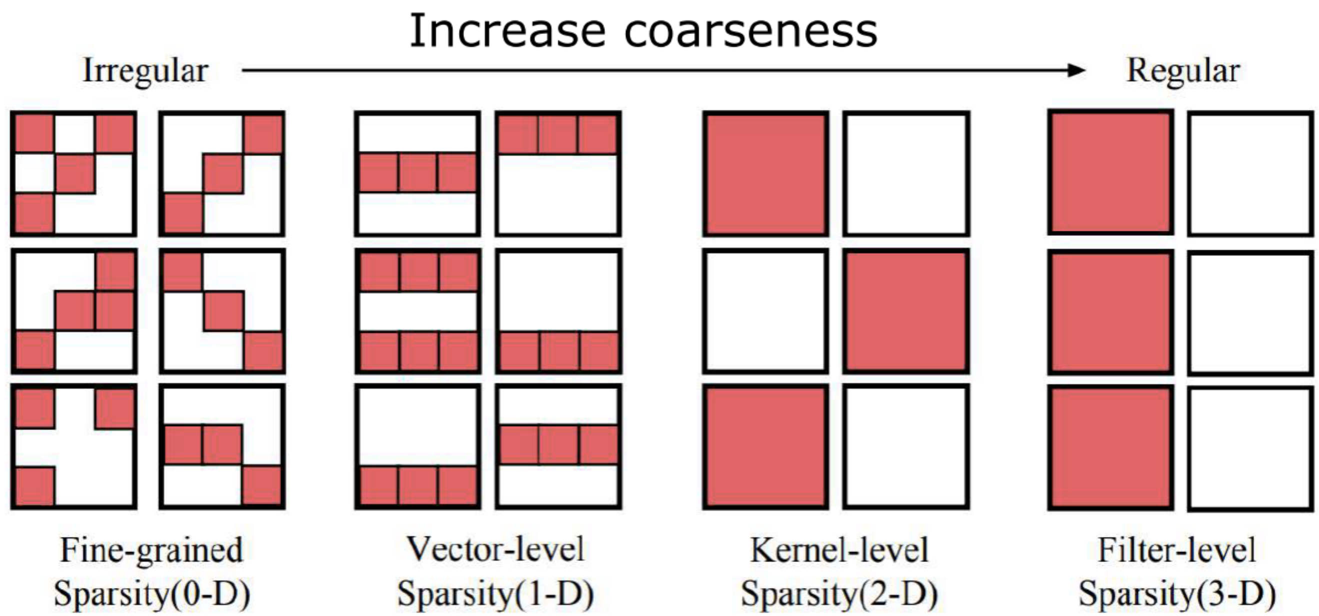



Image Source: [**Mao**, *CVPR Workshop* 2017]

- Benefits
  - Increase coarseness → more structure in sparsity (easier for hardware)
  - Less signaling overhead for location of zeros → better compression

# Design Considerations for Sparsity

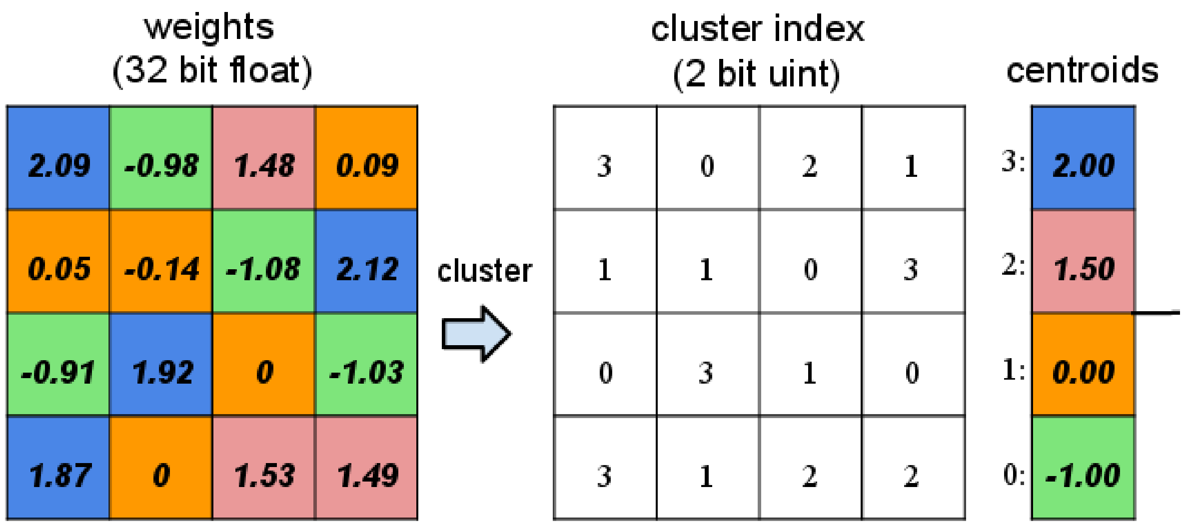
- **Impact on accuracy**
  - Must consider difficulty of dataset, task, and DNN model
    - e.g., AlexNet and VGG known to be over parameterized and thus easy to prune weights; does method work on efficient DNN models?

- **Does hardware cost exceed benefits?**
  - Need extra hardware to identify sparsity
    - e.g., Additional logic to identify non-zeros and store non-zero locations
  - Accounting for sparsity in both weights and activations is challenging
    - Need to compute intersection of two data streams rather than find next non-zero in one
  - Granularity impacts hardware overhead as well as accuracy
    - e.g., Fine-grained or coarse-grained (structured) sparsity
  - Compressed data will be variable length
    - Reduced flexibility in access order → random access will have significant overhead

# Quantization and Weight Sharing

# Quantization and Weight Sharing

- Quantize to fixed number of distinct values at no accuracy loss
- AlexNet conv layers quantized using 8 bits (256 16-bit weights) results in zero accuracy loss



# Quantization and Weight Sharing

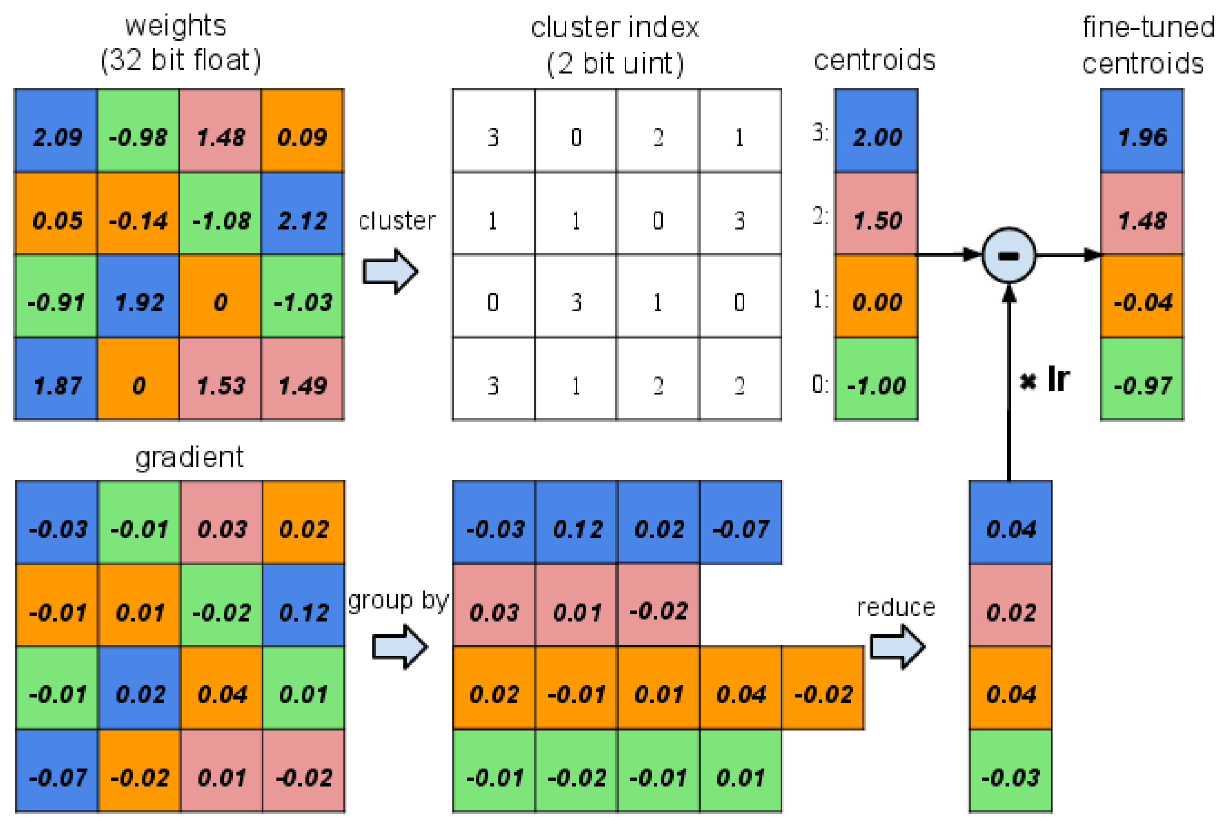
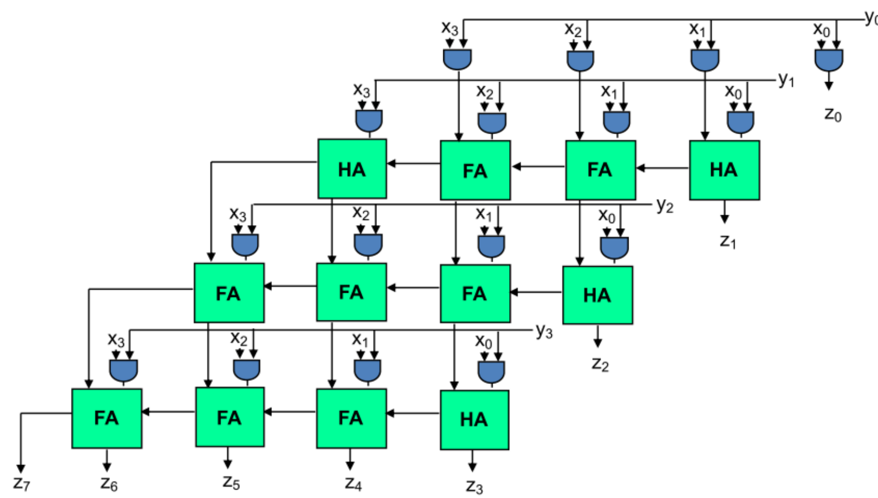



Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

# Why Reduce Precision (i.e., Reduce Bit Width)?

- Reduce data movement and storage cost for inputs and outputs of MAC
  - Smaller memory → lower energy
- Reduce cost of MAC
  - Cost of multiply increases with bit width (n) → energy and area by $O(n^2)$; delay by $O(n)$

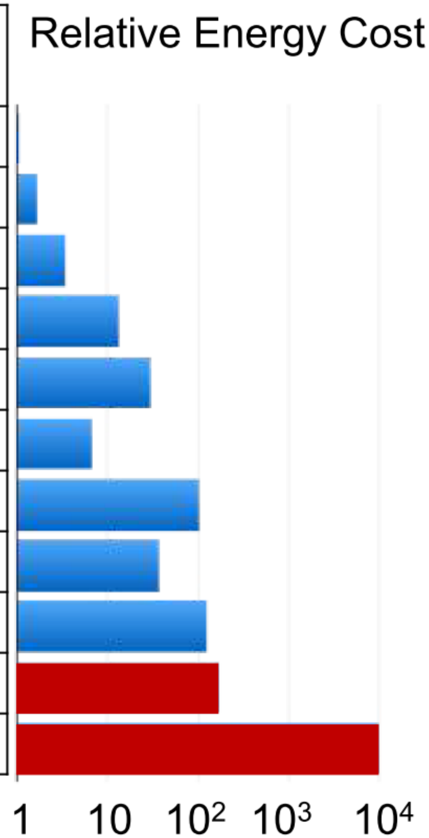
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | $x_3$ | $x_2$ | $x_1$ | $x_0$ | Multiplicand |
| | | | | $\times$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ | Multiplier |
| | | | | | $x_3y_0$ | $x_2y_0$ | $x_1y_0$ | $x_0y_0$ | |
| | | | | $x_3y_1$ | $x_2y_1$ | $x_1y_1$ | $x_0y_1$ | | Partial Product |
| | | | $x_3y_2$ | $x_2y_2$ | $x_1y_2$ | $x_0y_2$ | | | |
| + | | $x_3y_3$ | $x_2y_3$ | $x_1y_3$ | $x_0y_3$ | | | | |
| | $z_7$ | $z_6$ | $z_5$ | $z_4$ | $z_3$ | $z_2$ | $z_1$ | $z_0$ | Result |

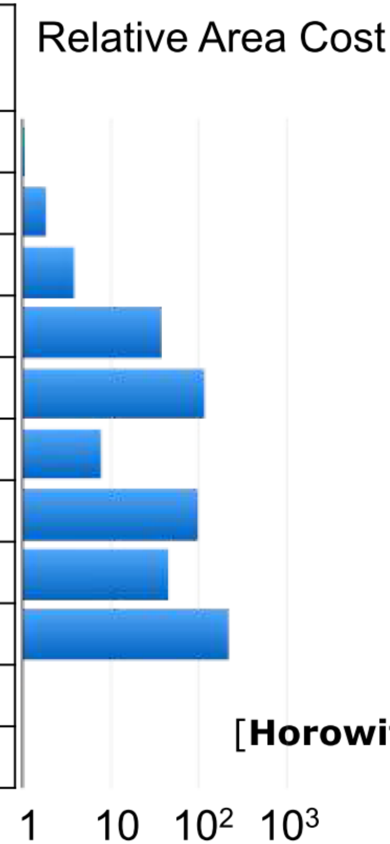



**Note:** Bit width for multiplication and accumulation in a MAC are different

# Impact of Reduced Precision on Energy & Area

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

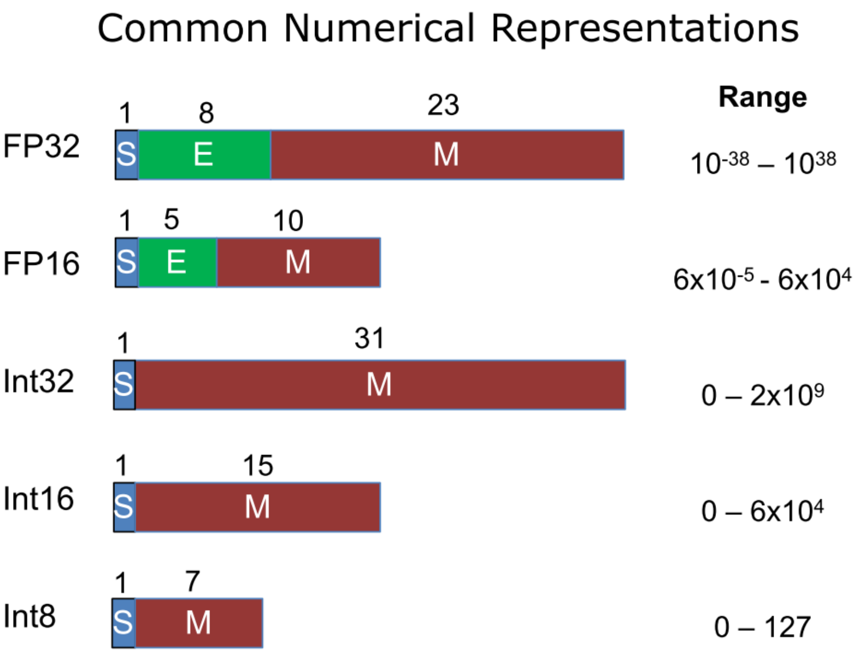



[**Horowitz**, *ISSCC* 2014]

# What Determines Bit Width? (1/4)

- Number of unique values
  - e.g., **M-bits** to represent $2^M$ values
- Dynamic range of values
  - e.g., **E-bits** to scale values by $2^{(E-127)}$
- Signed or unsigned values
  - e.g., signed requires one extra bit (S)
- **Total bits = S+E+M**
- **Floating point (FP)** allows range to change for each value (E-bits)
- **Fixed point (Int)** has fixed range
- Default CPU/GPU is 32-bit float (**FP32**)



# What Determines Bit Width? (2/4)

- For accuracy, require sufficient precision to represent different data types
  - For inference: weights, activations, and partial sums
  - For training: weights, activations, partial sums, gradients, and weight update
  - Required precision can vary across data types
  - Referred to as mixed precision

# What Determines Bit Width? (3/4)

- **Reduce number of unique values** (**M-bits**, a.k.a. mantissa)
  - **Default:** Uniform quantization (values are equally spaced out)
  - Non-uniform quantization (spacing can be computed, e.g., logarithmic, or with look-up-table)
  - Fewer unique values can make transforms and compression more effective

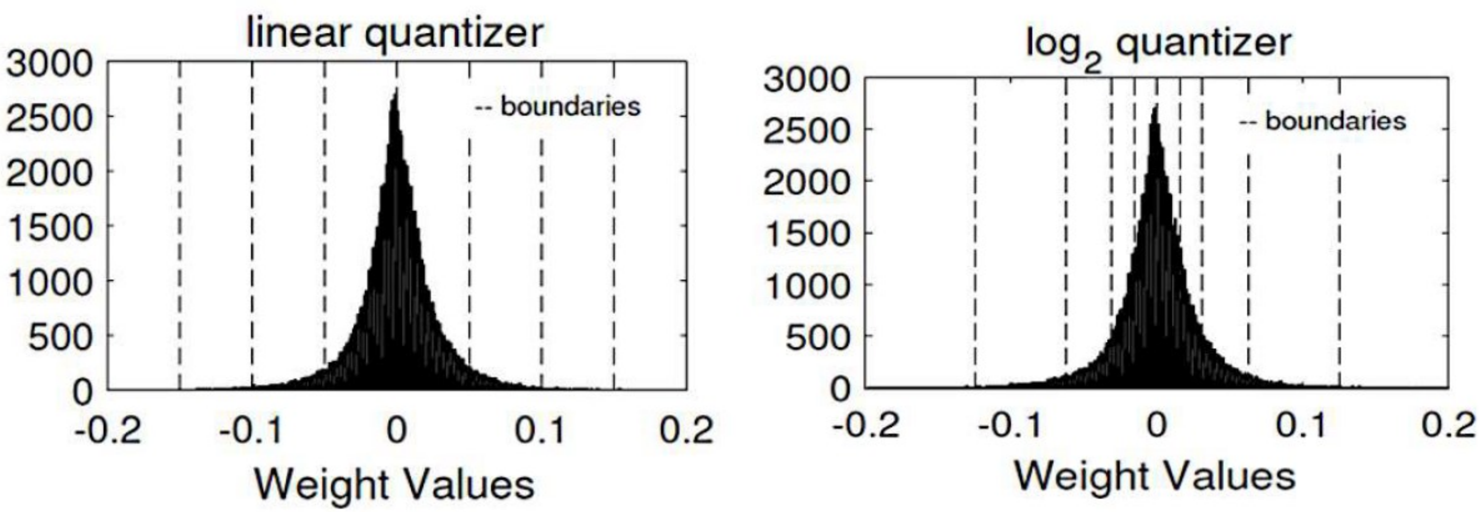



Image Source:[**Lee**, *ICASSP* 2017]

# What Determines Bit Width? (4/4)

- **Reduce number of unique values** (**M-bits**, a.k.a. mantissa)
  - **Default:** Uniform quantization (values are equally spaced out)
  - Non-uniform quantization (spacing can be computed, e.g., logarithmic, or with look-up-table)
  - Fewer unique values can make transforms and compression more effective
- **Reduce dynamic range** (**E-bits**, a.k.a., exponent)
  - If possible, fix range (i.e., used fixed point, E=0)
  - Share range across group of values (e.g., weights for a layer or channel)
- Tradeoff between number of bits allocated to **M-bits** and **E-bits**

| | | |
|---|---|---|
| **fp16** (S=1, E=5, M=10) | S E E E E E M M M M M M M M M M | range: ~$5.9e^{-8}$ to ~$6.5e^{4}$ |
| **bfloat16** (S=1, E=8, M=7) | S E E E E E E E E M M M M M M M | range: ~$1e^{-38}$ to ~$3e^{38}$ |

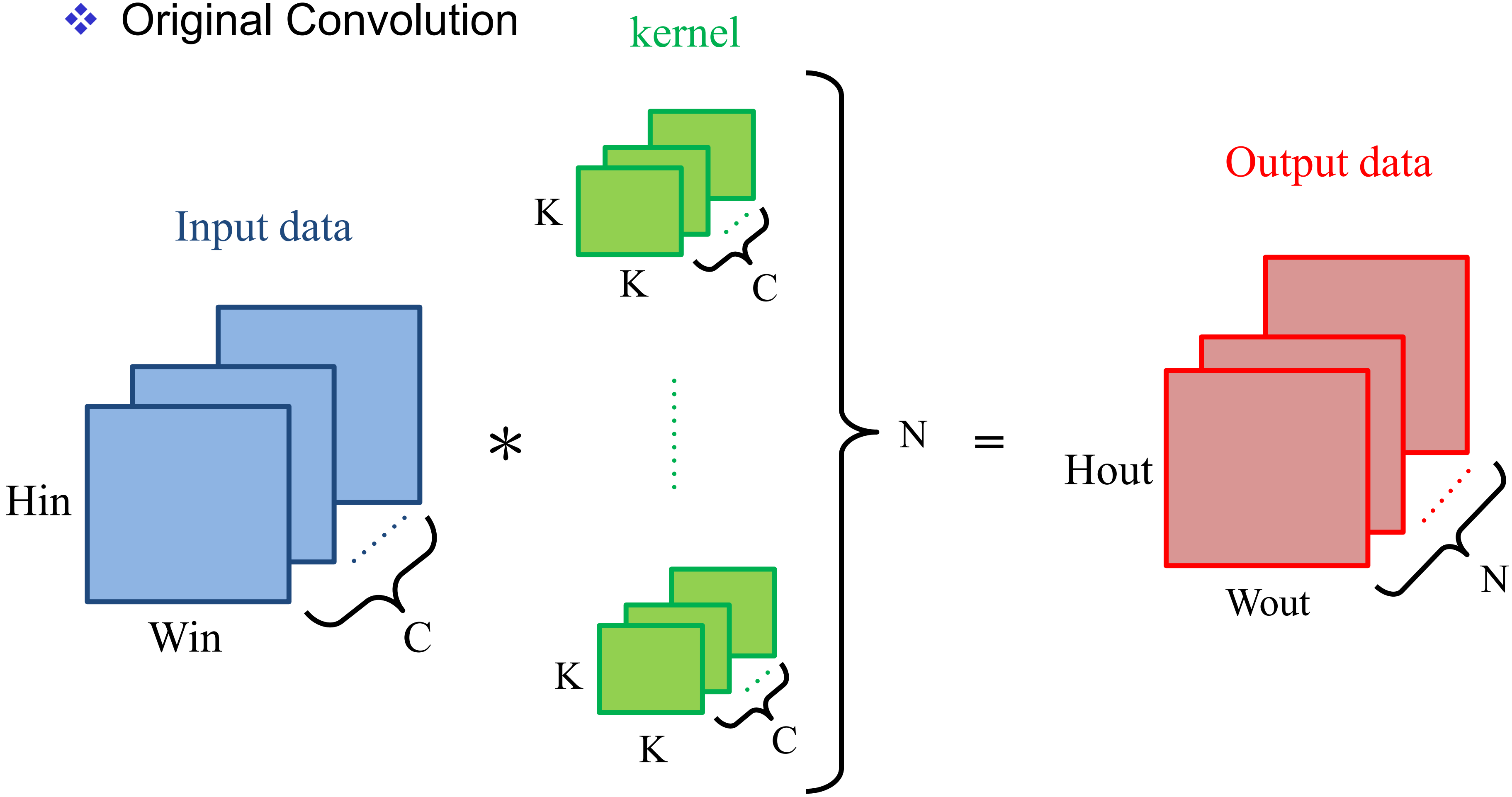
# Design Considerations for Reduced Precision

- **Impact on accuracy**
  - Must consider difficulty of dataset, task, and DNN model
    - e.g., Easy to reduce precision for an easy task (e.g., digit classification); does method work for a more difficult task?
- **Does hardware cost exceed benefits?**
  - Need extra hardware to support variable precision
    - e.g., Additional shift-and-add logic and registers for variable precision
  - Granularity impacts hardware overhead as well as accuracy
    - e.g., More overhead to support (1b, 2b, 3b … 16b) than (2b, 4b, 8b, 16b)
- **Evaluation**
  - Use 8-bit for inference and 16-bit float for training for baseline
  - 32-bit float is a weak baseline

# Depthwise Separable Convolution (DSC)

# Depthwise Separable Convolution (1/7)

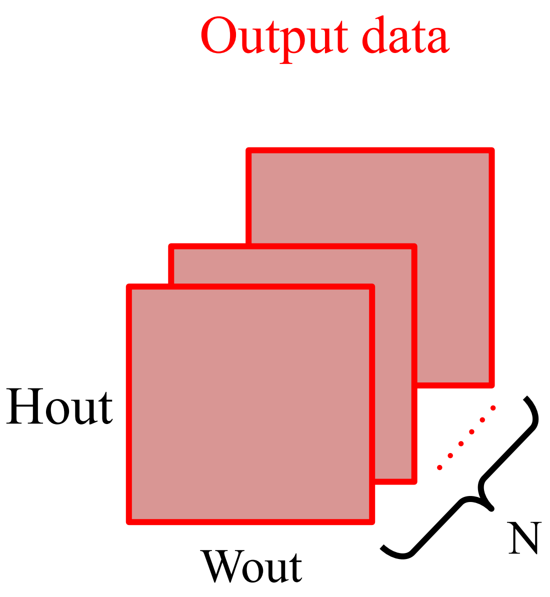
- Original Convolution



# Depthwise Separable Convolution (2/7)

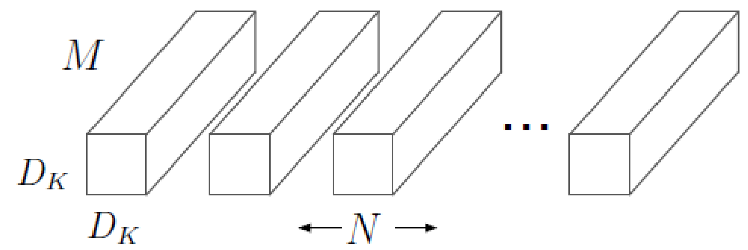
- Depthwise separable convolution is proposed to reduce the amount of computation without affecting the output structure.

- It can be split into two parts:
  - Depthwise Convolution
  - Pointwise Convolution

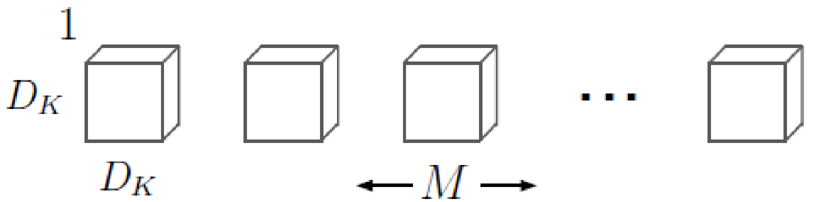


# Depthwise Separable Convolution (3/7)

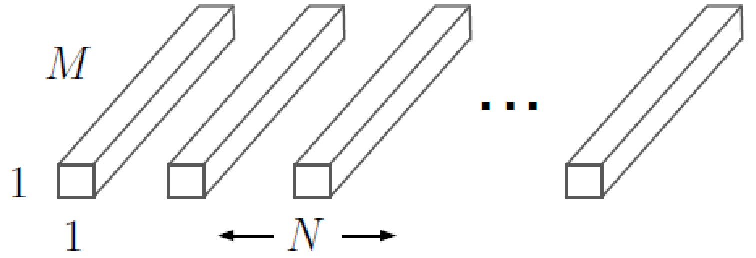
- Standard convolution
  - M input channels, N output channels
  - DK x DK filters




- Depthwise convolution (DWC)
  - Convolution of one input channel, generating one output channel
  - # of output channels = # of input channels



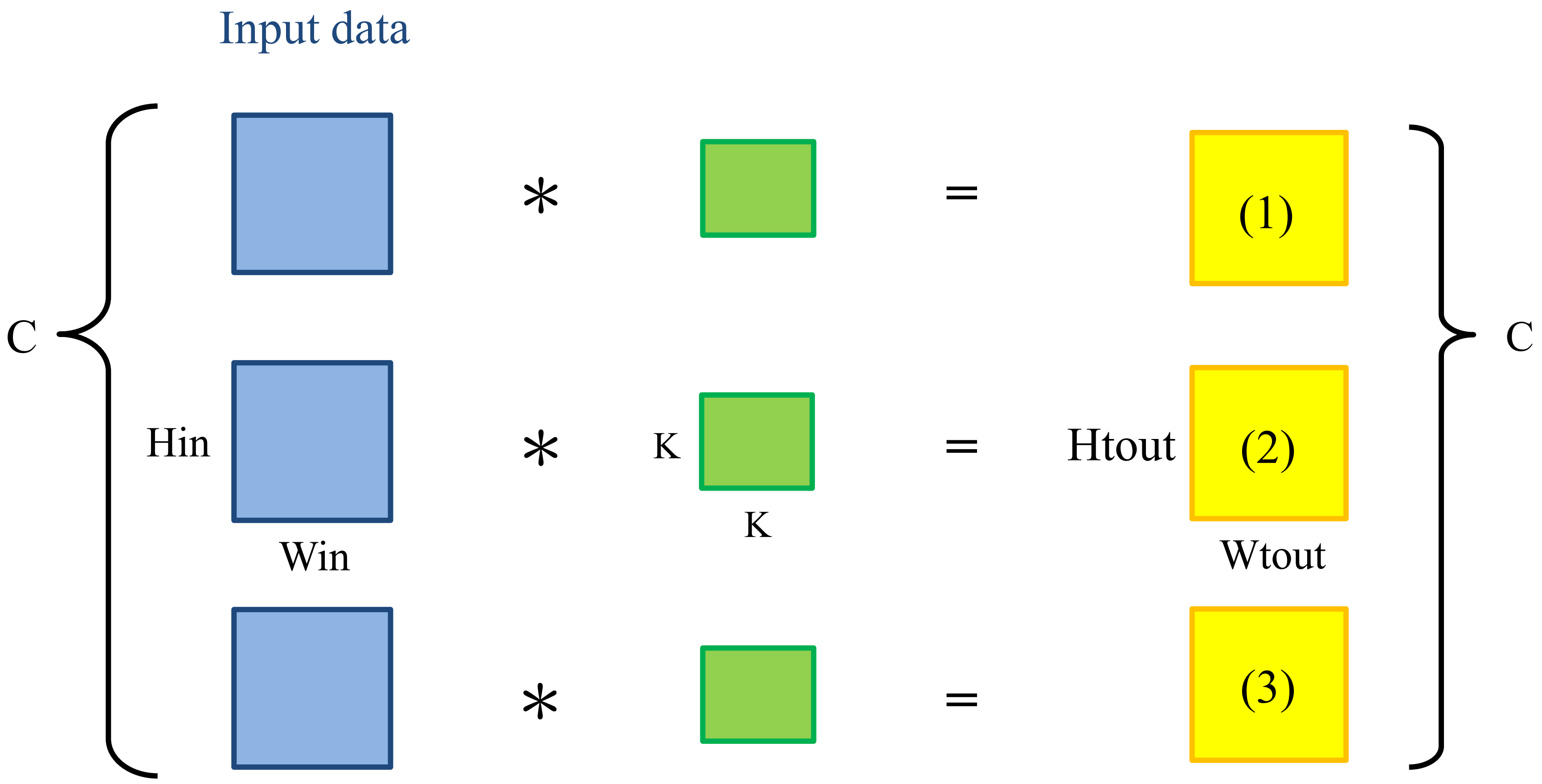

- Pointwise convolution (PWC)
  - 1x1 filters combining M input channels, generating N output channels




- DSC = DWC + PWC

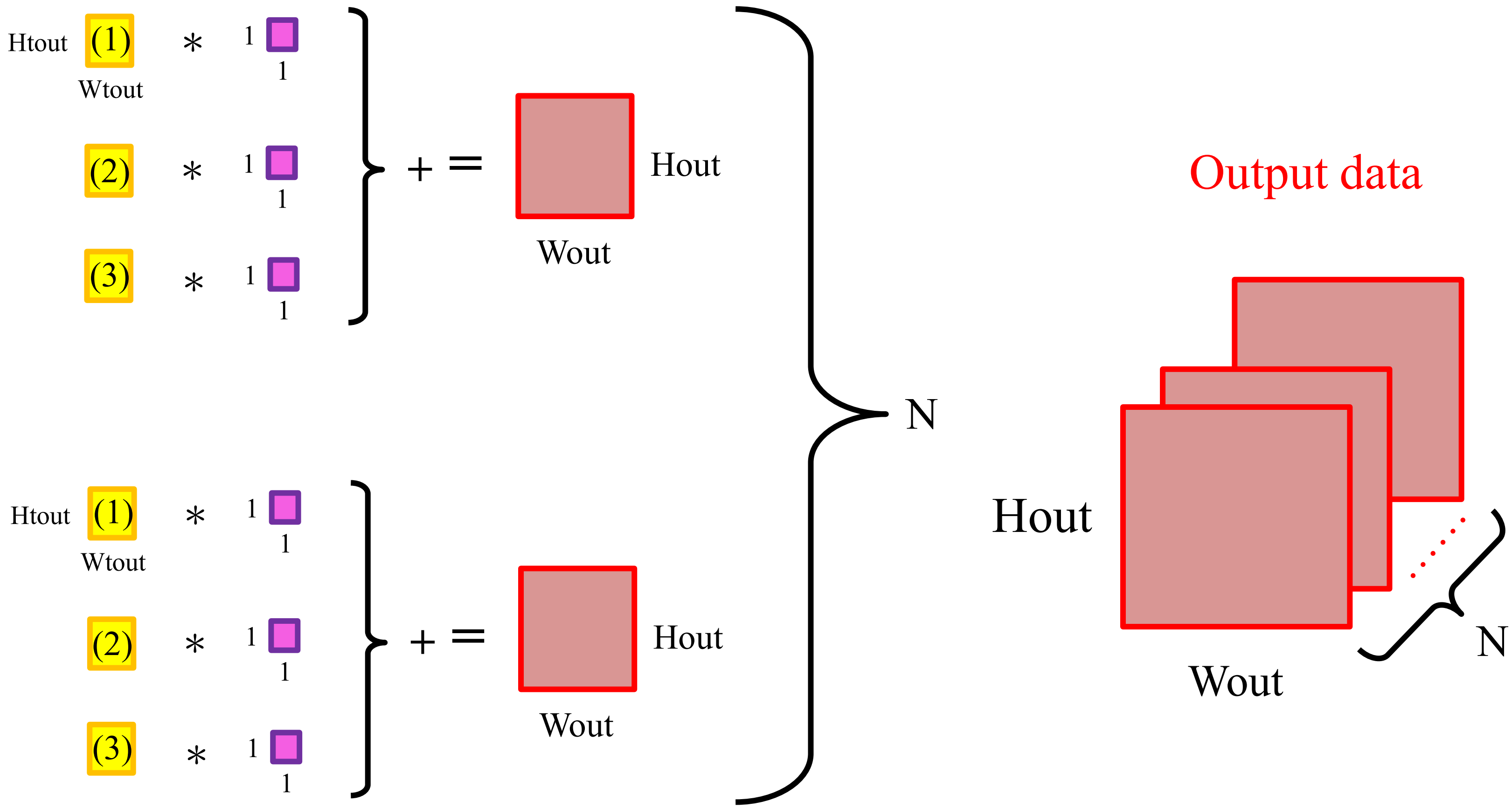
# Depthwise Separable Convolution (4/7)

- Depthwise Convolution



# Depthwise Separable Convolution (5/7)

- Pointwise Convolution



# Depthwise Separable Convolution (6/7)

- # Original Convolution operations can be approximated to:

$$Hout \times Wout \times C \times K \times K \times N$$

- # Depthwise Separable Convolution operations can be approximated to:

$$\underbrace{\boxed{Hout \times Wout \times C \times K \times K}}_{\text{Depthwise}} + \underbrace{\boxed{Hout \times Wout \times C \times N}}_{\text{Pointwise}}$$

Op. reduction =

$$\frac{Hout \times Wout \times C \times K \times K \ + \ Hout \times Wout \times C \times N}{Hout \times Wout \times C \times K \times K \times N}$$

$$= \frac{1}{K \times K} + \frac{1}{N}$$

# Depthwise Separable Convolution (7/7)

$$\frac{Hout \times Wout \times C \times K \times K \; + \; Hout \times Wout \times C \times N}{Hout \times Wout \times C \times K \times K \times N}$$

$$= \frac{1}{K \times K} + \frac{1}{N}$$

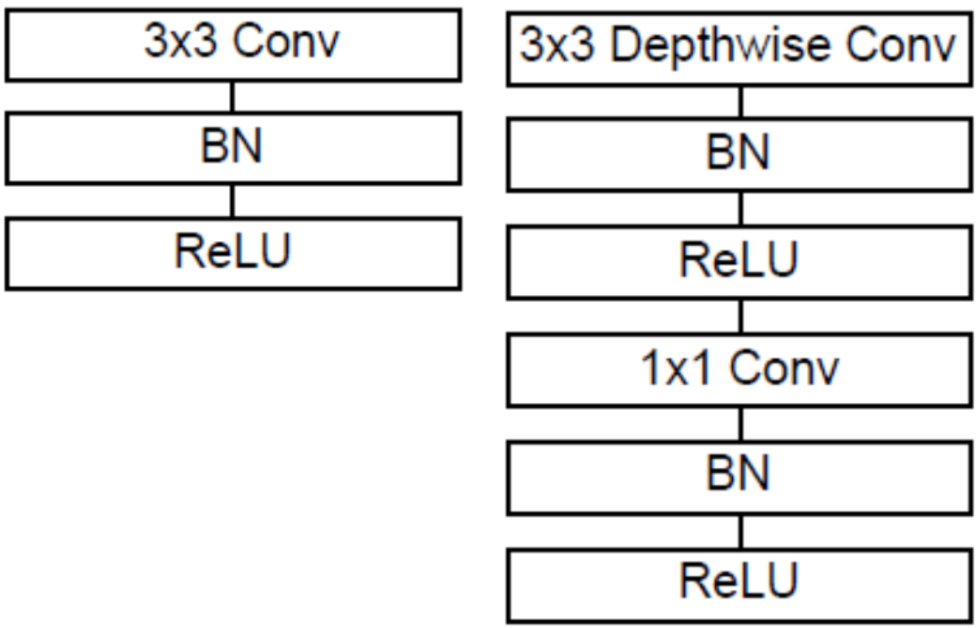
```
=====================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_____________________________________
```

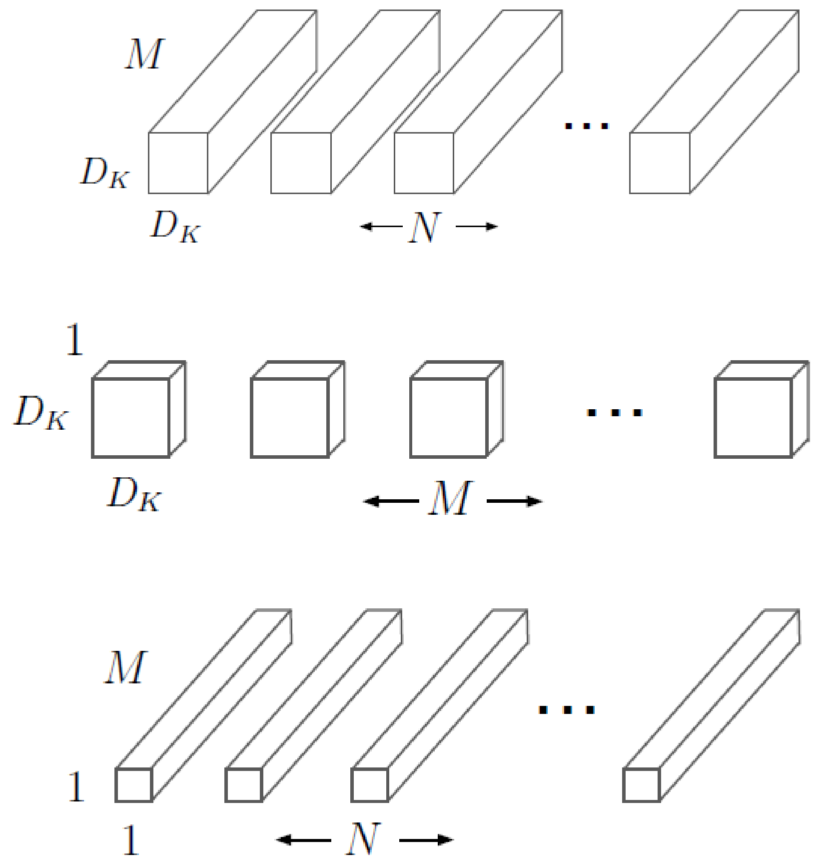
```
=====================================
Total params: 4,253,864
Trainable params: 4,231,976
Non-trainable params: 21,888
```

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

# Light CNN Models: MobileNet

- Standard 3D Convolution = Depthwise Convolution + 1x1 Convolution

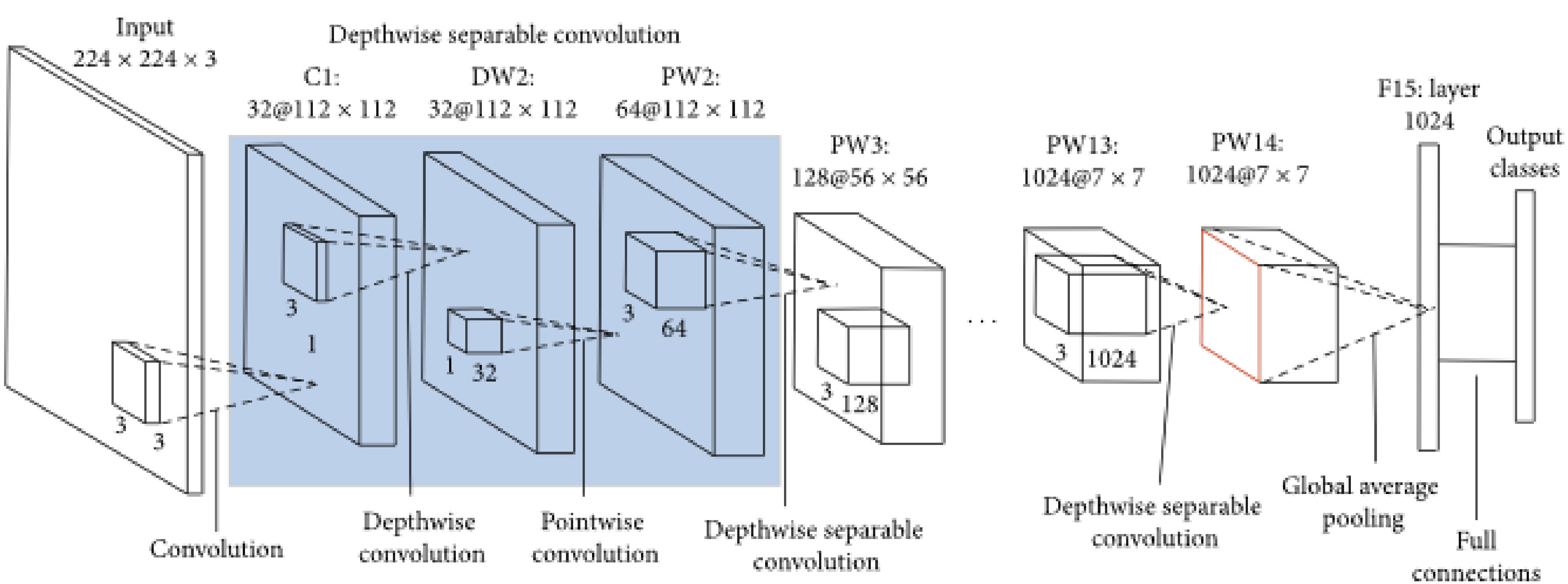


# MobileNet

- Width multiplier
  - Reduced channel number
- Resolution multiplier
  - Reduced channel size

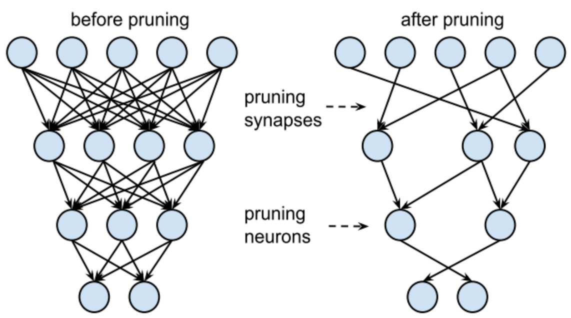
| Table 8. MobileNet Comparison to Popular Models | | | |
|---|---|---|---|
| Model | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
| 1.0MobileNet-224 | 70.6% | 569 | 4.2 |
| GoogleNet | 69.8% | 1550 | 6.8 |
| VGG 16 | 71.5% | 15300 | 138 |



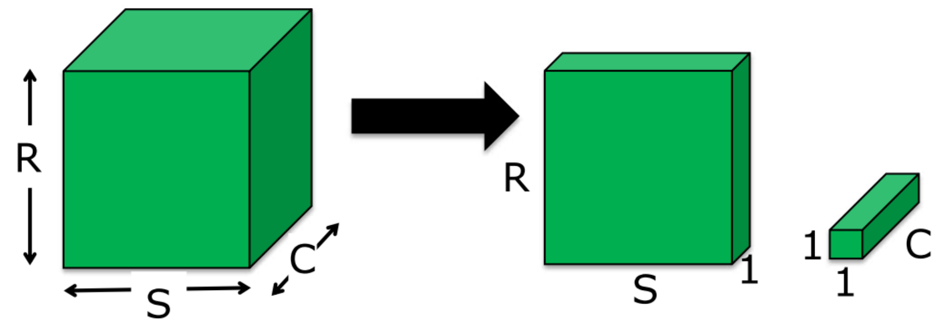
# Flexible and Scalability

# Many Efficient DNN Design Approaches

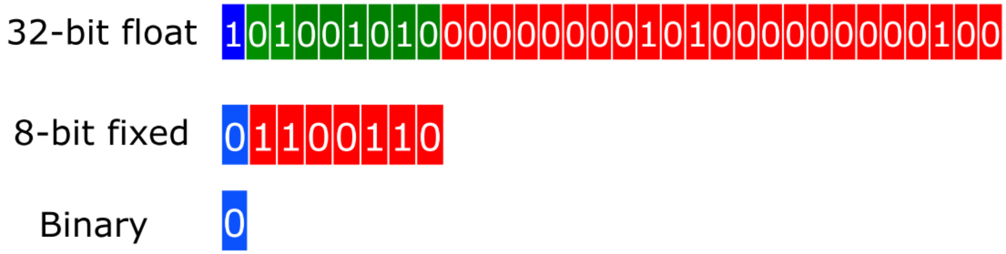
## Network Pruning




## Efficient Network Architectures




## Reduce Precision

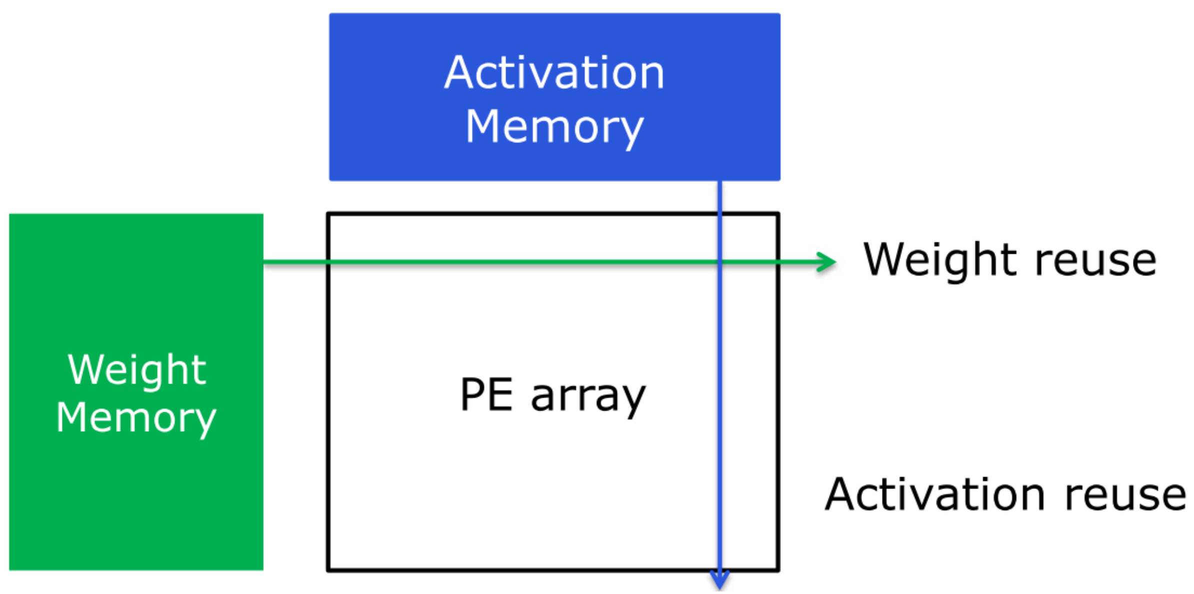



No guarantee that DNN algorithm designer will use a given approach.
**Need flexible DNN processor!**

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

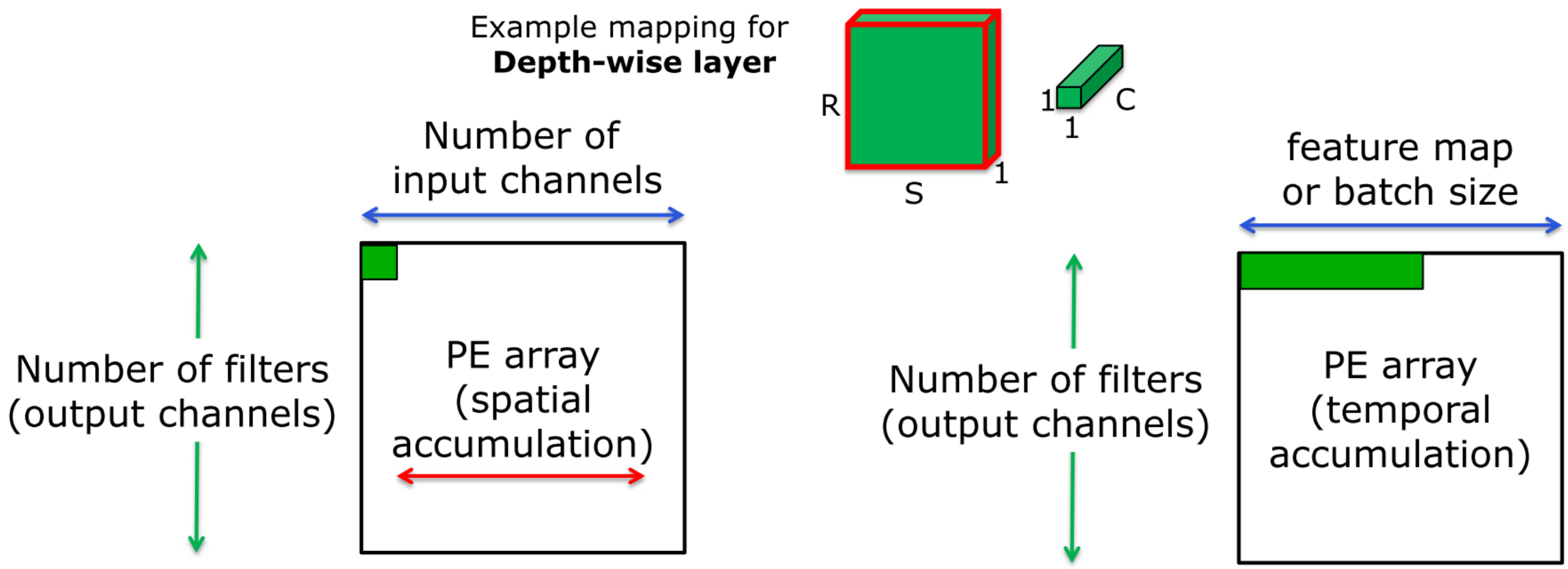
# Limitations of Existing DNN Processors (1/2)

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



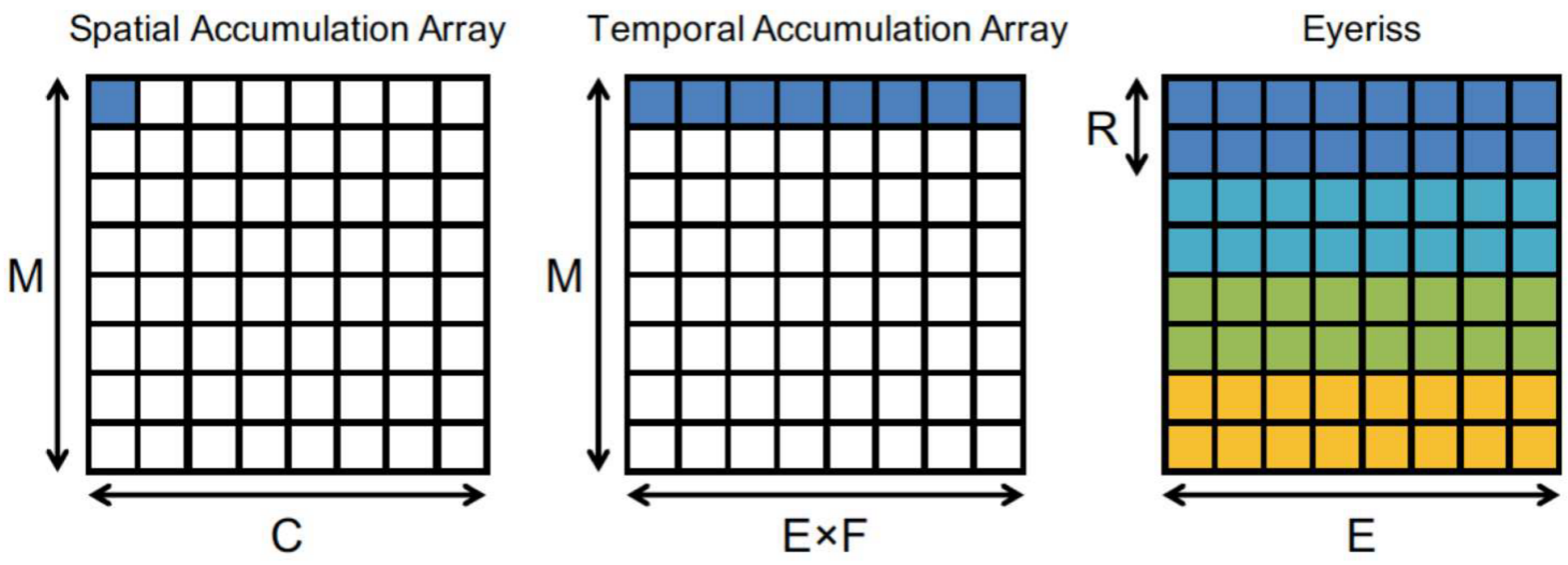
# Limitations of Existing DNN Processors (2/2)

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



# Need Flexible Dataflow

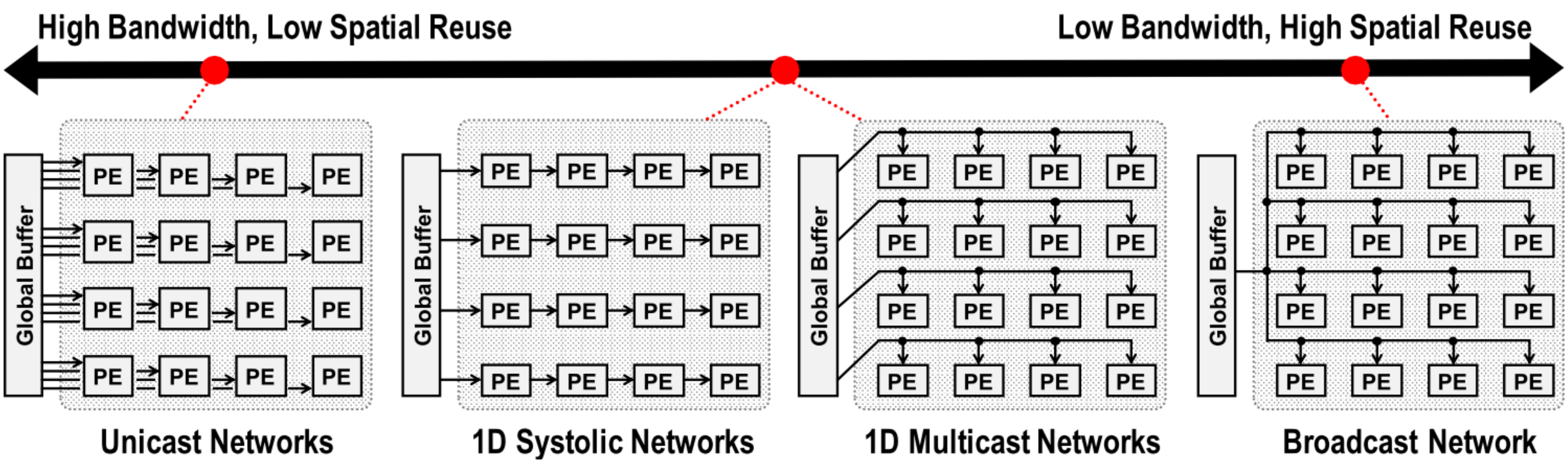
- Use flexible dataflow (Row Stationary) to exploit reuse in any dimension of DNN to increase energy efficiency and array utilization



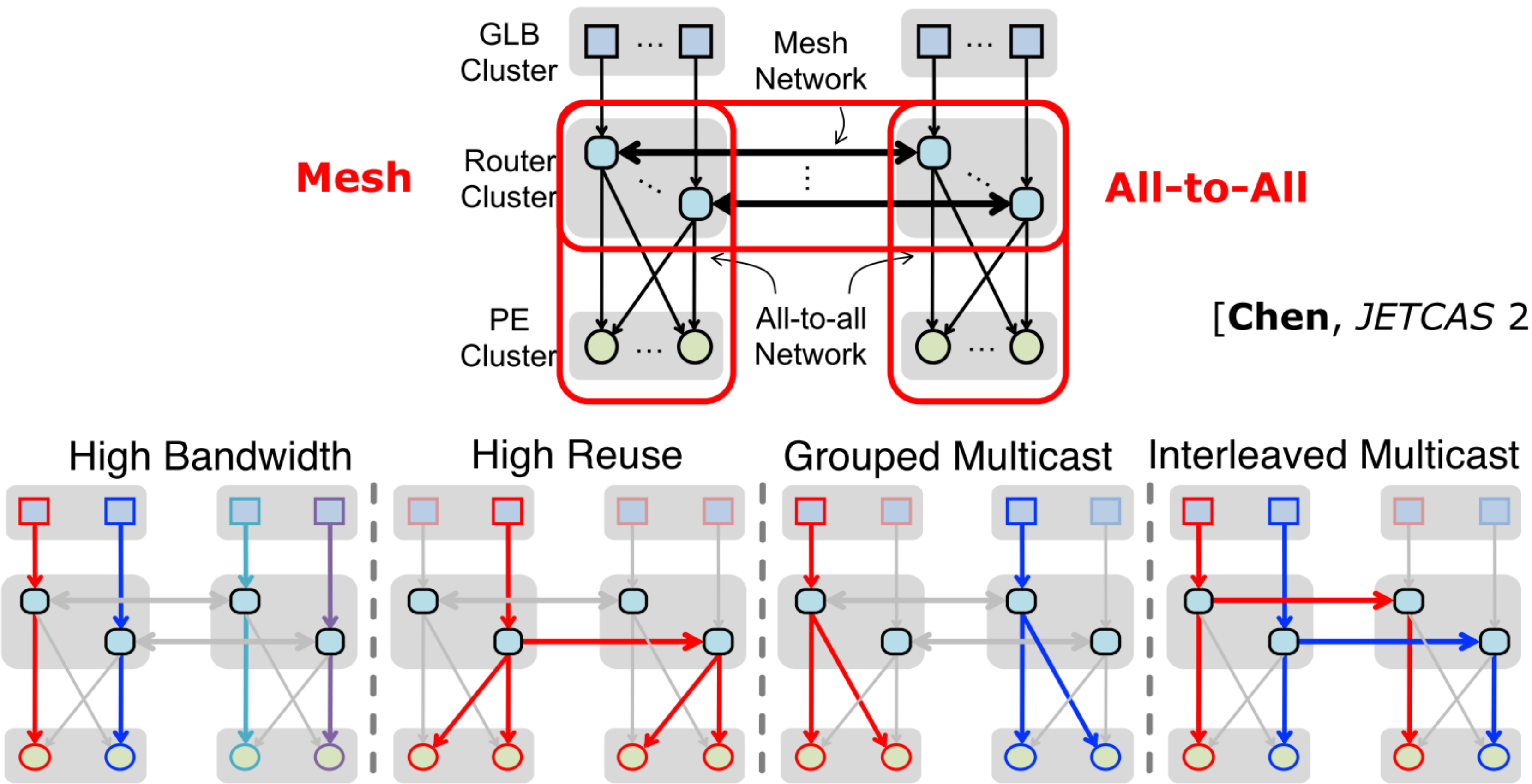

**Example: Depth-wise layer**

# Need Flexible On-Chip Network for Varying Reuse

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



# Hierarchical Mesh

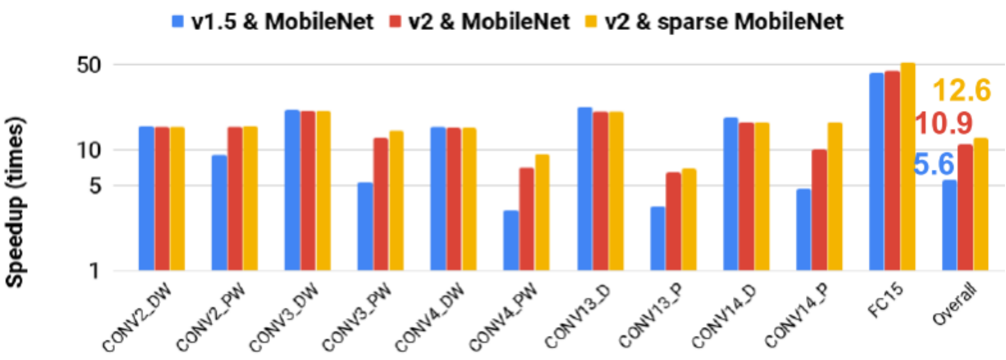



[**Chen**, *JETCAS* 2019]

# Eyeriss v2: Balancing Flexibility and Efficiency

Efficiently supports

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




*Speed up over Eyeriss v1 scales with number of PEs*

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

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

# Design Considerations for ASIC

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