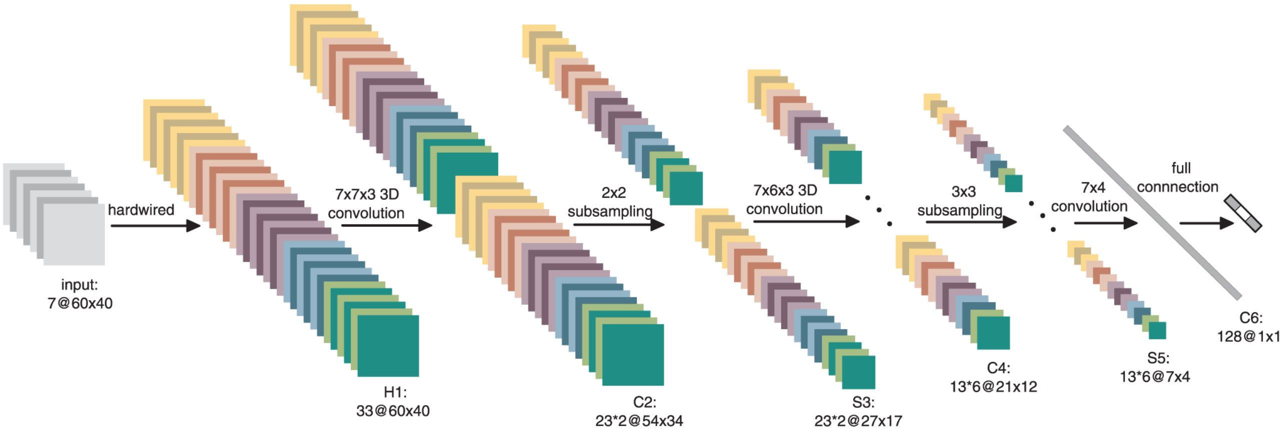
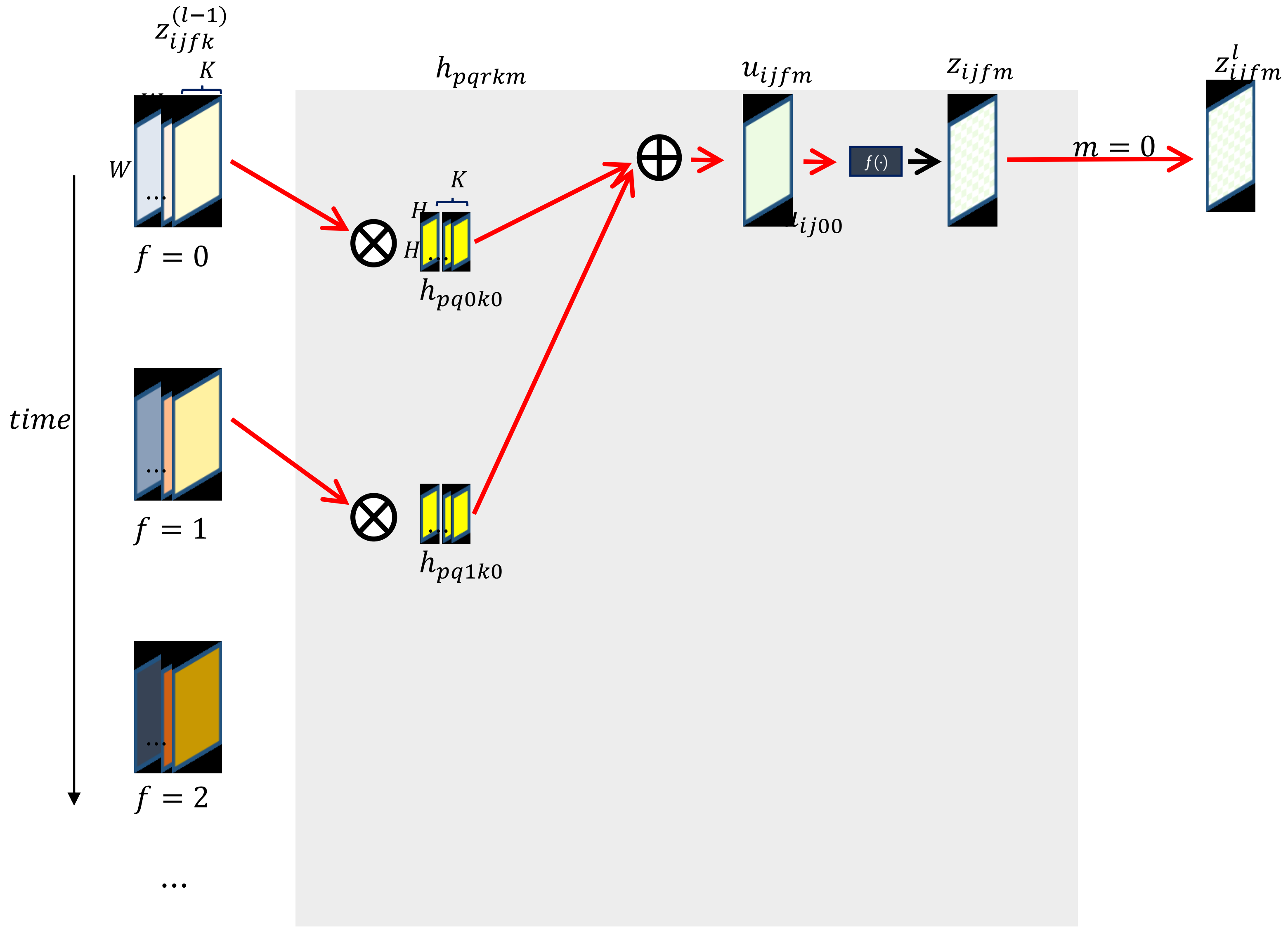
3D Convolutional Neural Networks

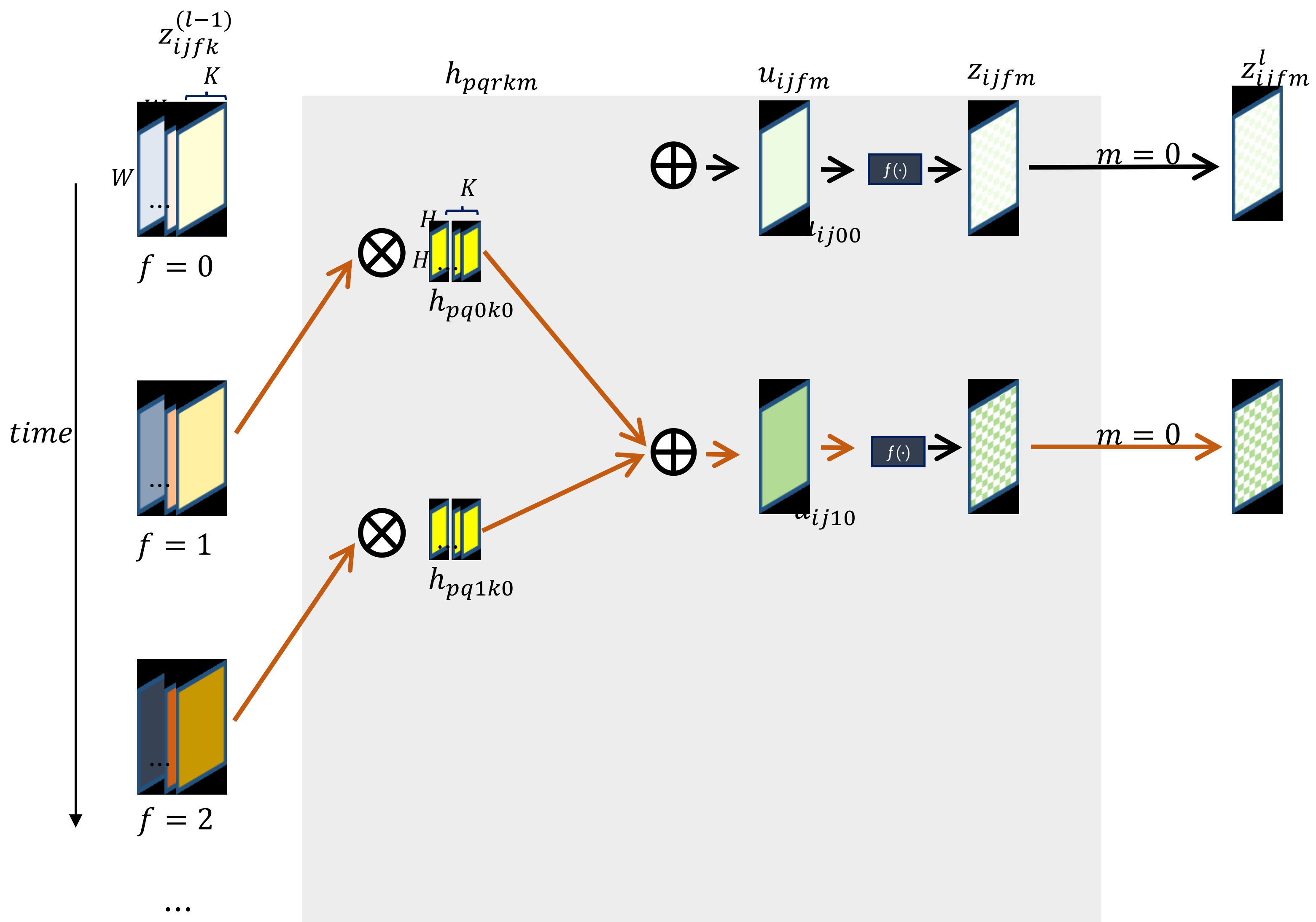


Fast Campus
Start Deep Learning with Tensorflow

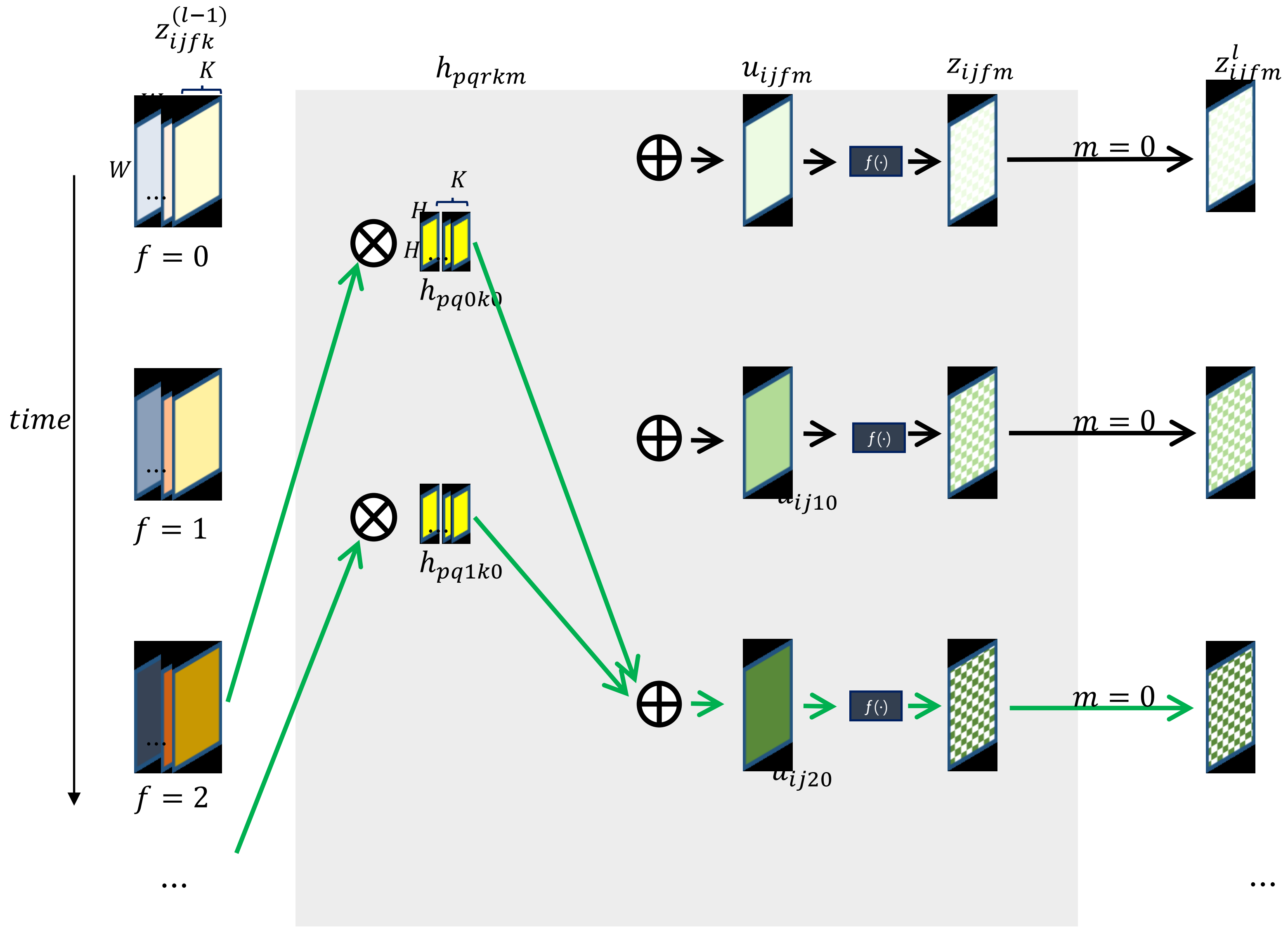
3D Convolution



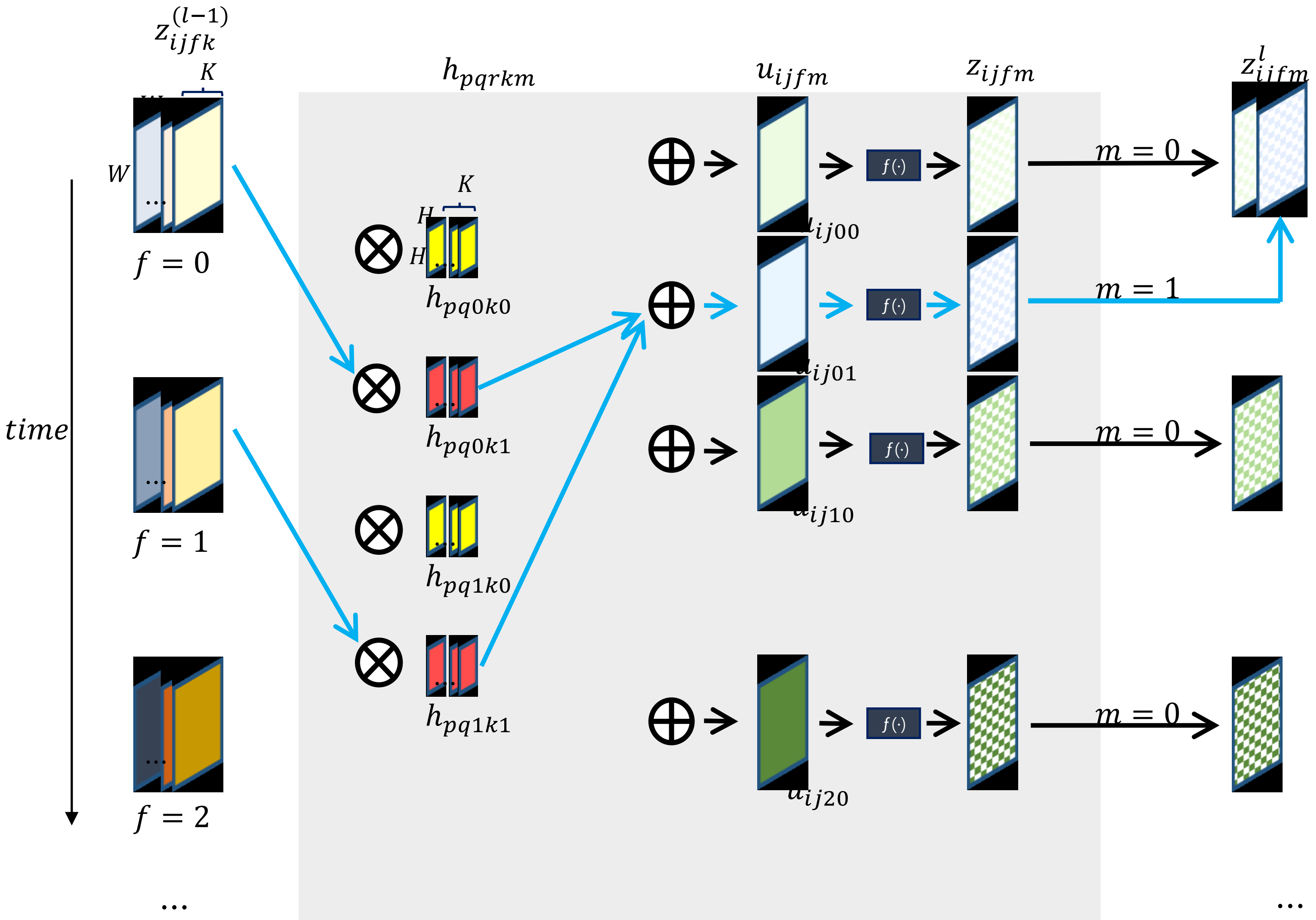
3D Convolution



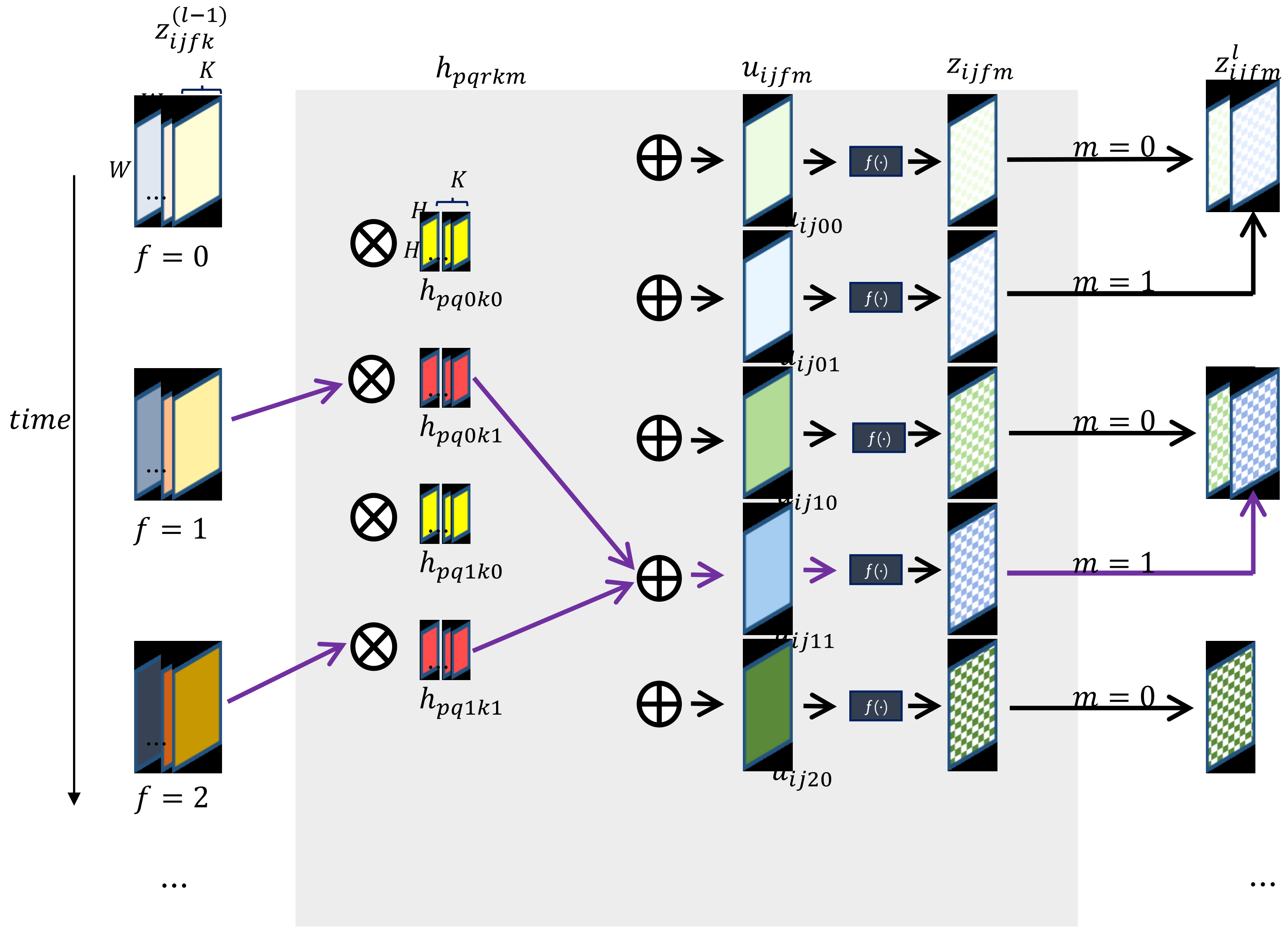
3D Convolution



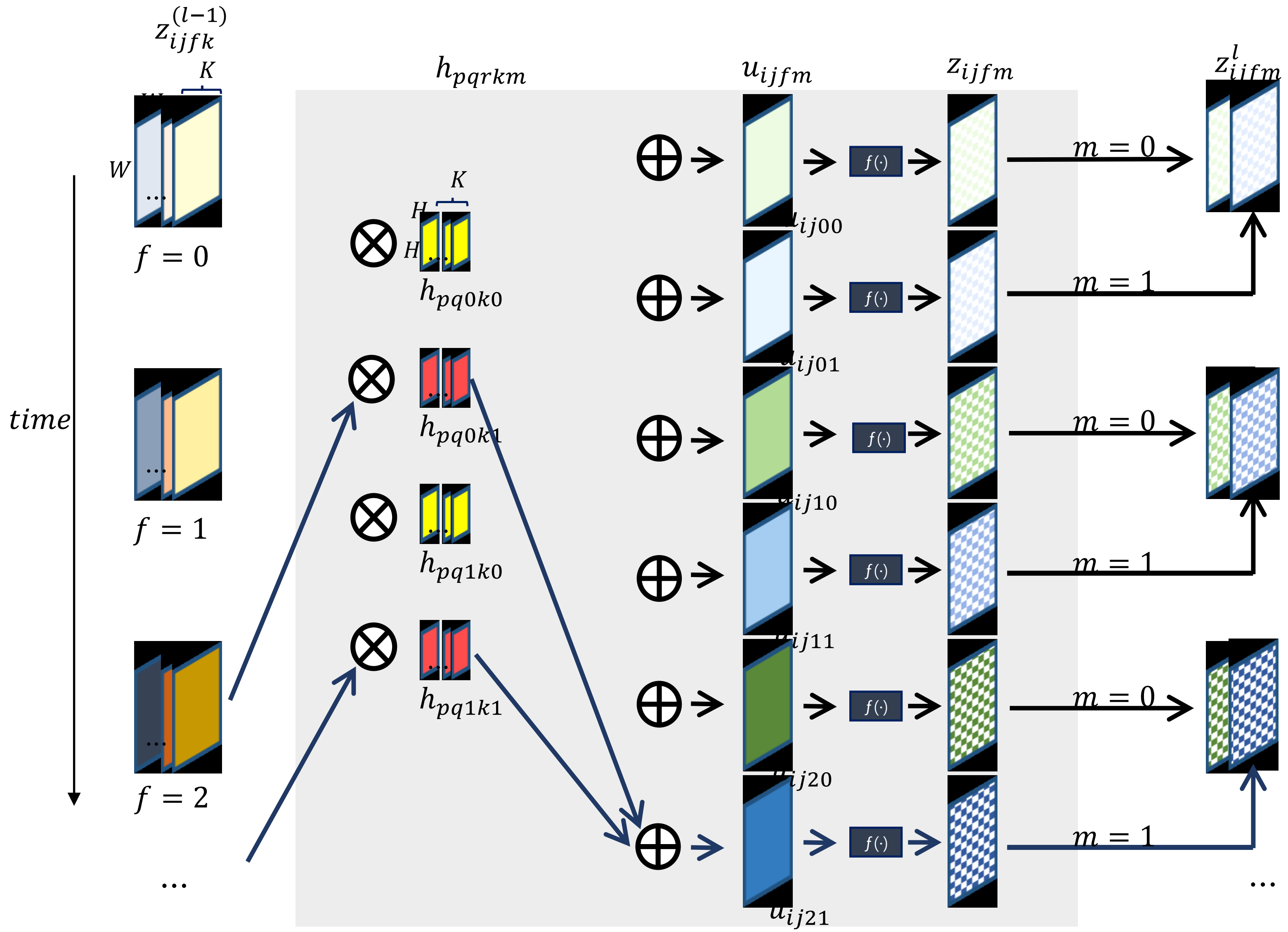
3D Convolution



3D Convolution

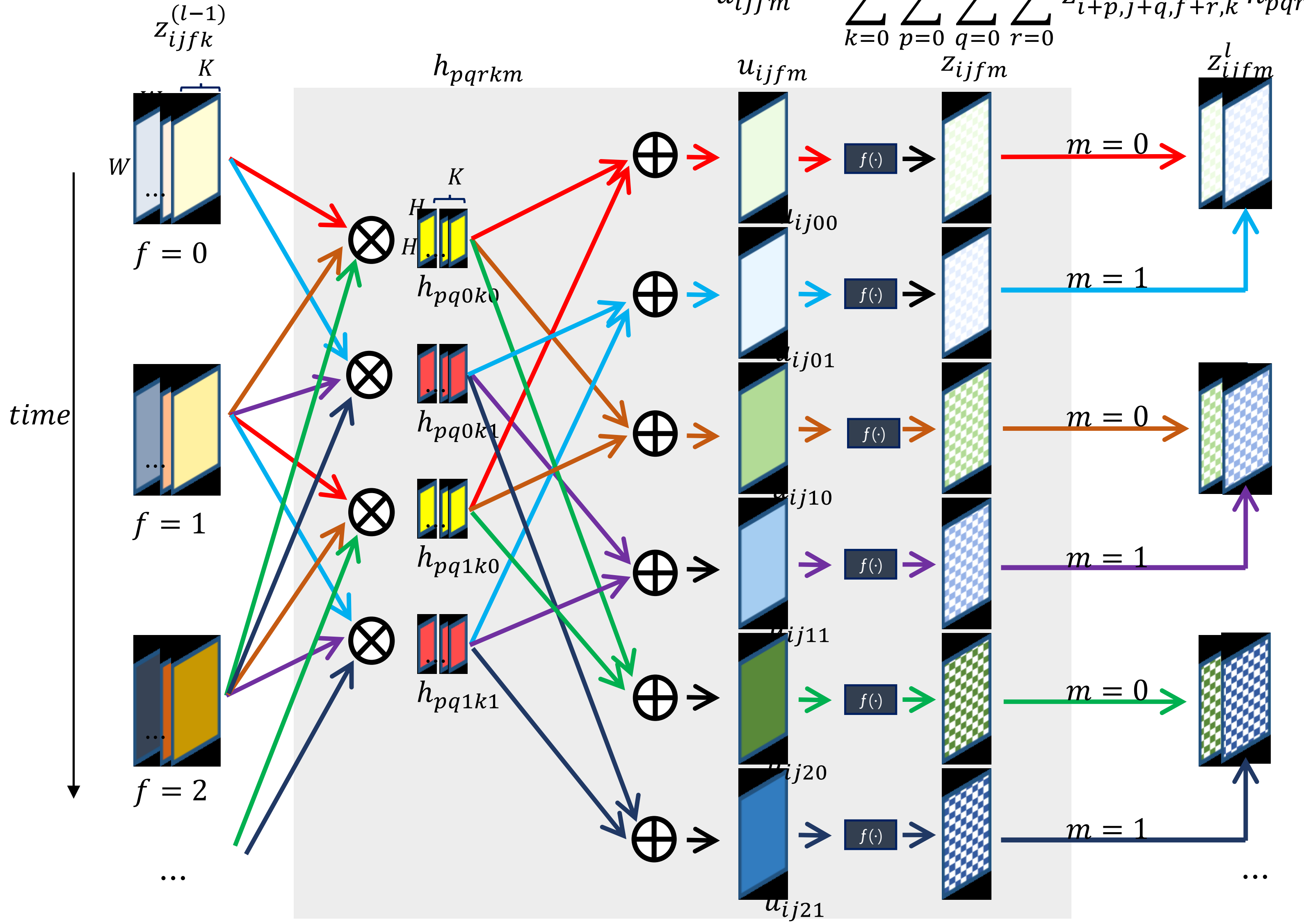


3D Convolution



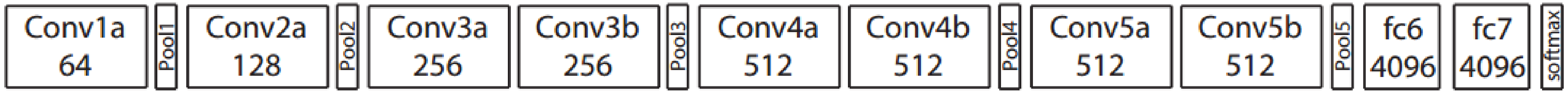
3D Convolution

$$u_{ijfm} = \sum_{k=0}^{K-1} \sum_{p=0}^{H-1} \sum_{q=0}^{H-1} \sum_{r=0}^{R-1} z_{i+p,j+q,f+r,k}^{(l-1)} h_{pqrkm} + b_{ijfm}$$



Problems

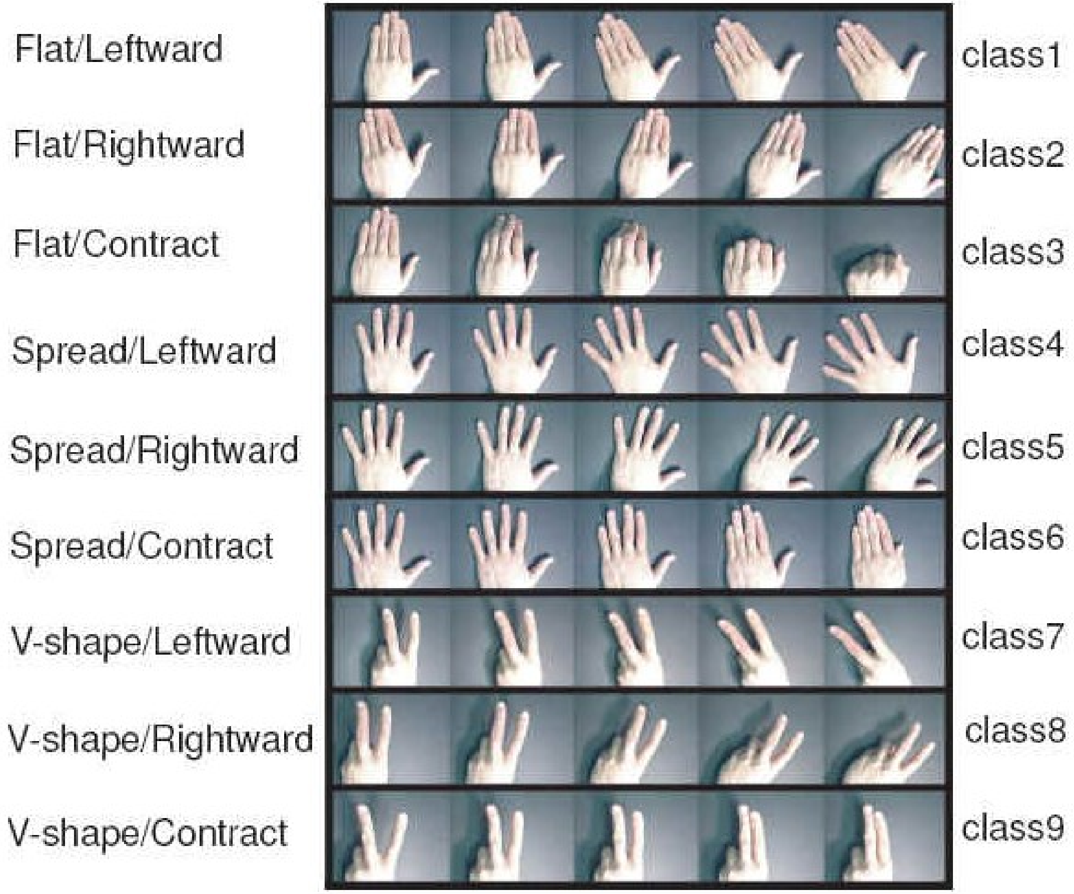
- Complex!
  - # of Multiplications
    - ➢ C3D : 50,332M
    - ➢ GoogLeNet : 1,566M
    - ➢ VGG-19 : 19,647M
    - ➢ AlexNet : 725M
- Too many parameters!
  - # of Parameters
    - ➢ C3D : 77.9M
    - ➢ GoogLeNet : 6.9M
    - ➢ VGG-19 : 144M
    - ➢ AlexNet : 61M
  - C3D requires about 312Mbyte memory space for saving parameters
  - More parameters need more data



C3D architecture

Dataset

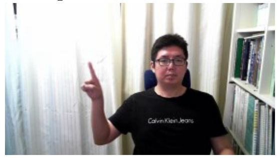
- Cambridge Hand Gesture Dataset
  - 900 image sequences of 9 gesture classes
  - Defined by 3 primitive hand shapes and 3 primitive motions
  - QVGA(320x240)

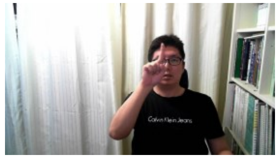


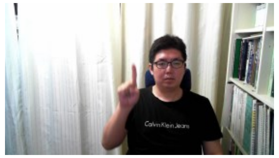
Dataset

- CAPP Hand Gesture Dataset
  - 20 gesture classes, 100 image sequences of each class
  - 720p(1280x720), 30fps

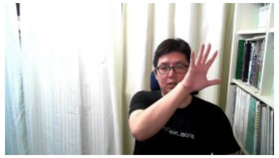

click


finger left


finger right


finger up


finger down


grab

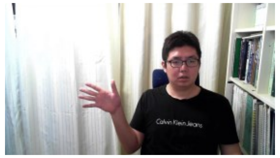

grab left


grab right


grab up


grab down


push


flip left

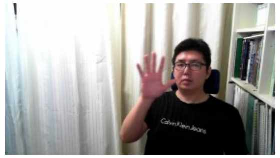

flip right

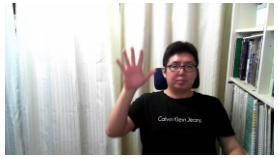

flip up

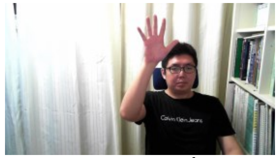

flip down


pull

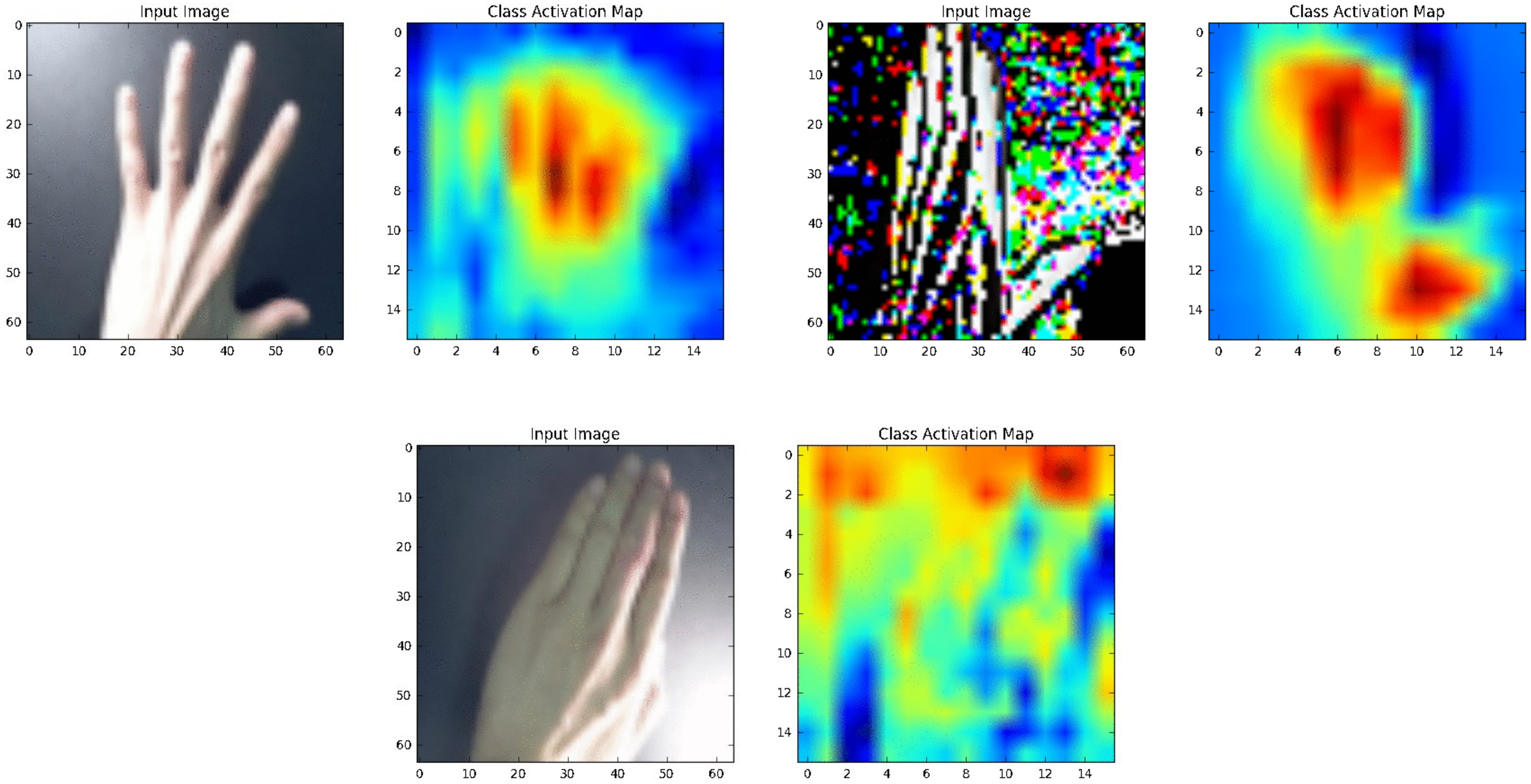

move left


move right

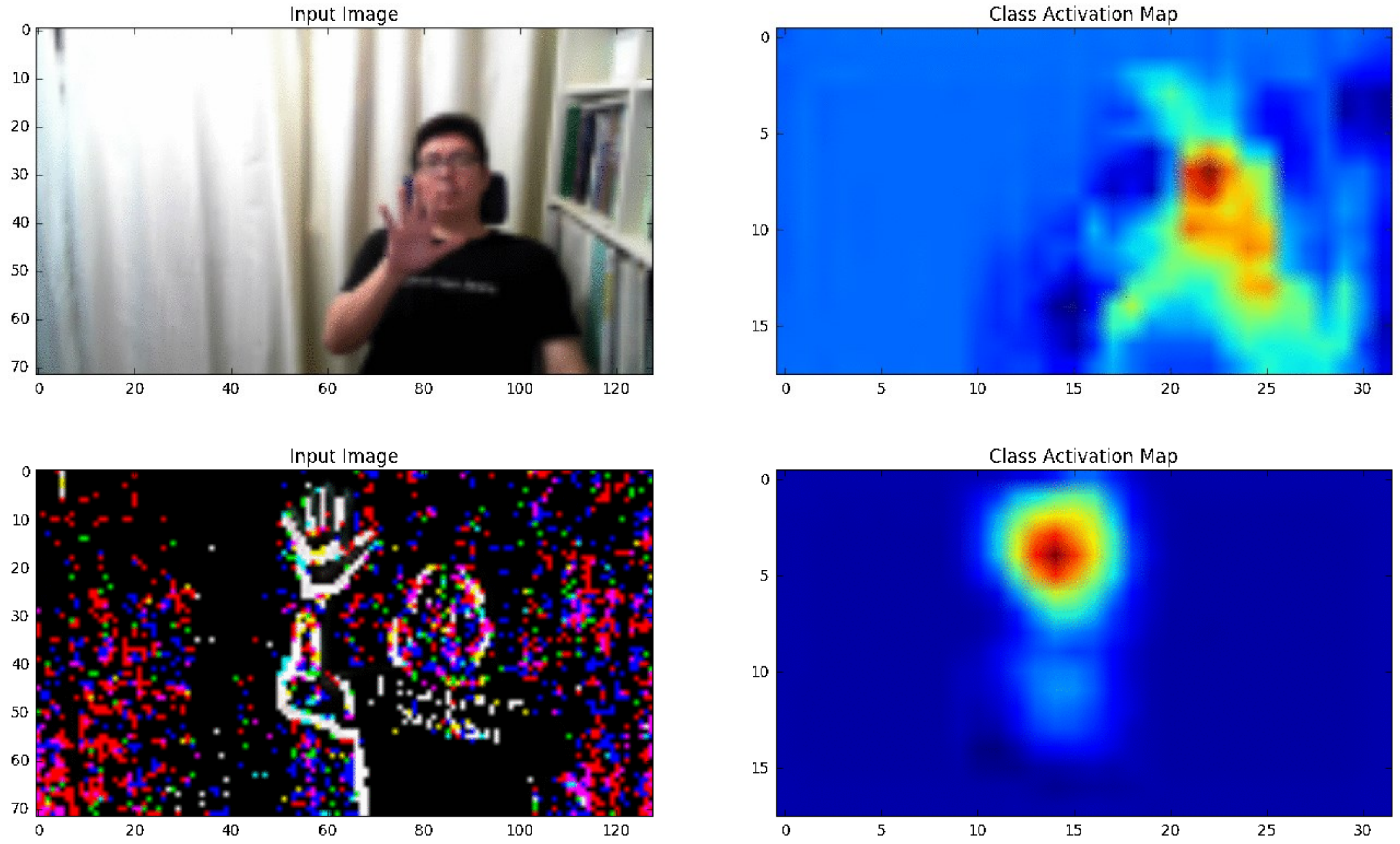

move up


move down

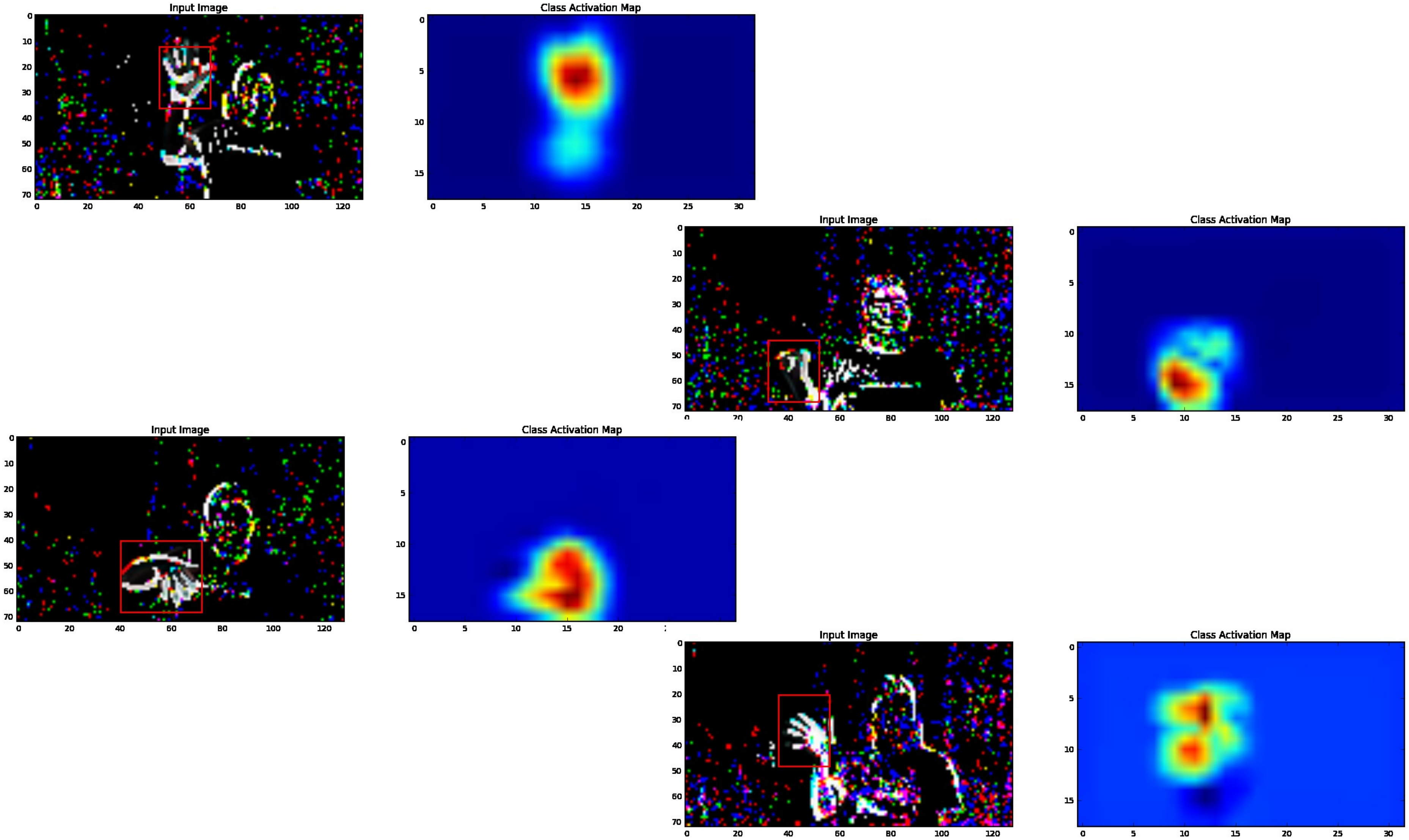
CAM Results – Cambridge Dataset



CAM Results – CAPP Dataset



Hand Detection using CAM



Optimization of CNN

Fast Campus
Start Deep Learning with Tensorflow

Key Requirements for Commercial Computer Vision Usage

- Data-centers(Clouds)
  - Rarely safety-critical
  - Low power is nice to have
  - Real-time is preferable
- Gadgets – Smartphones, Self-driving cars, Drones, etc.
  - Usually safety-critical(except smartphones)
  - Low power is must-have
  - Real-time is required

What’s the “Right” Neural Network for Use in a Gadget?

- Desirable Properties
  - Sufficiently high accuracy
  - Low computational complexity
  - Low energy usage
  - Small model size

Why Small Deep Neural Networks?

- Small DNNs train faster on distributed hardware
- Small DNNs are more deployable on embedded processors
- Small DNNs are easily updatable Over-The-Air(OTA)

Example - Vision Processing in Self-Driving Car in 2020

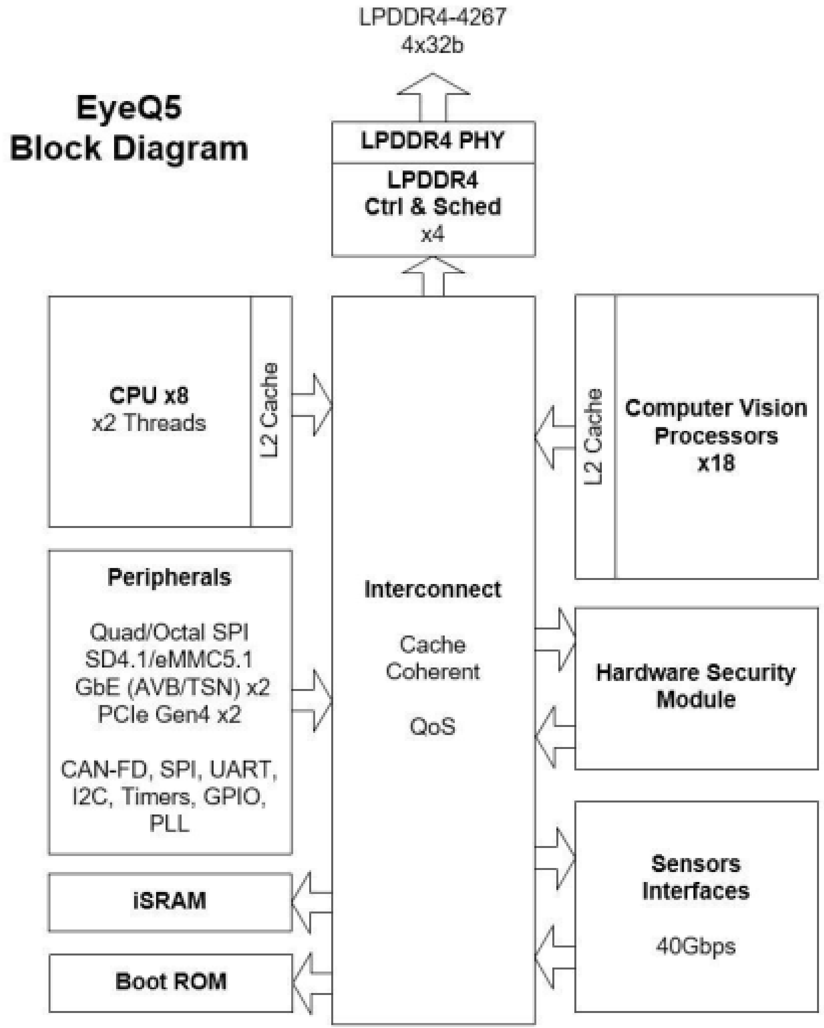
- High Precision
  - 4K images @30~60fps
  - Mobileye's keynote at CVPR2016

- Many Cameras
  - >10 cameras at least
  - 14 cameras in ZooX's prototype car

ZooX's prototype car at CVPR 2016 Exhibition

Mobileye EyeQ5

- 12TOPS@5W, 10nm, 2018
  - NVIDIA P100: 10TOPS@300W, 16nm, 2016
  - 10TOPS requires 5,000 MACs running at 1GHz
- EyeQ5 consists of
  - VMP(Vector Microcode Processors): wide(10s way) MAC array – FFT, …
  - PMA(Programmable Macro Array): very wide(100s way) MAC array – Convolution, …
  - MPCs(Multithreaded Processing Clusters): multi-threaded MAC array



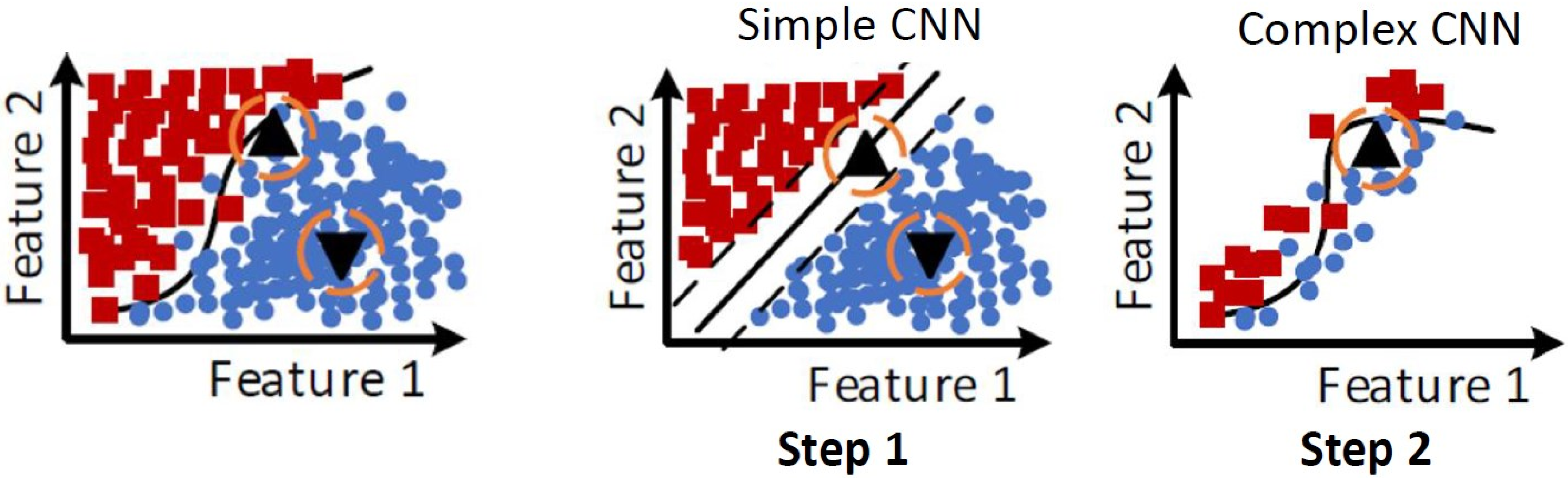
http://ir.mobileye.com

Computation Demand of Neural Network for Object Detection

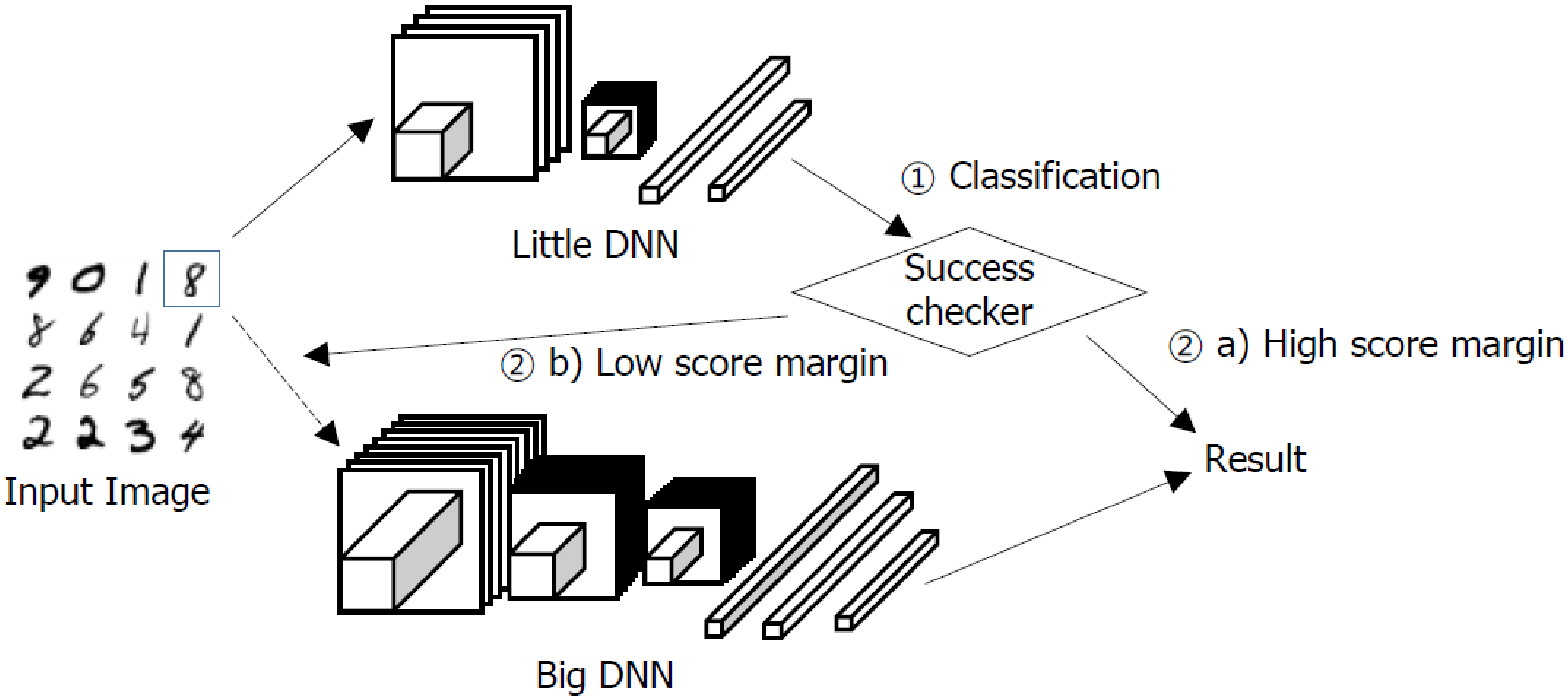
- Assumption
  - Object detection in full HD requires ~20TOPS(fp16)
    - Based on Faster R-CNN(GoogLeNet)
- Estimation of compute demand in 2020
  - 4K image → >4X increase in compute demand
  - ~10 cameras(assuming 4 cameras support 4K) are needed → >5X increase
  - >400TOPS(=20TOPSx4x5) only for object detection
  - Other tasks, e.g., image segmentation, also need to be performed → ~1.5X increase
  - **>600TOPS** for all vision-based processing
- Do we really need 60 EyeQ5 chips for a single car?
  - 12TOPS, fp16, 80% utilization

Simple Example: Two-Way Classification

- Classification – draw a surface between two groups
- Complex(high order) surface – high cost
- Basic idea – classify simple ones first at low cost

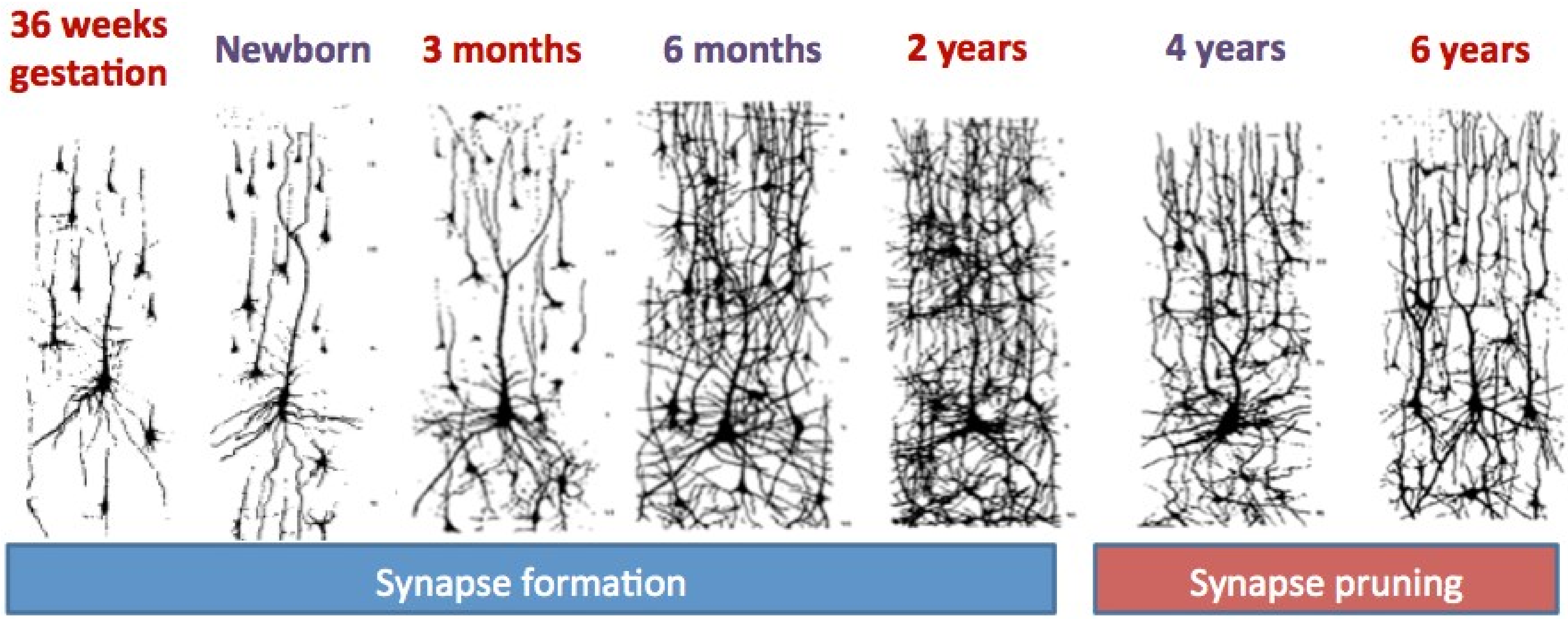


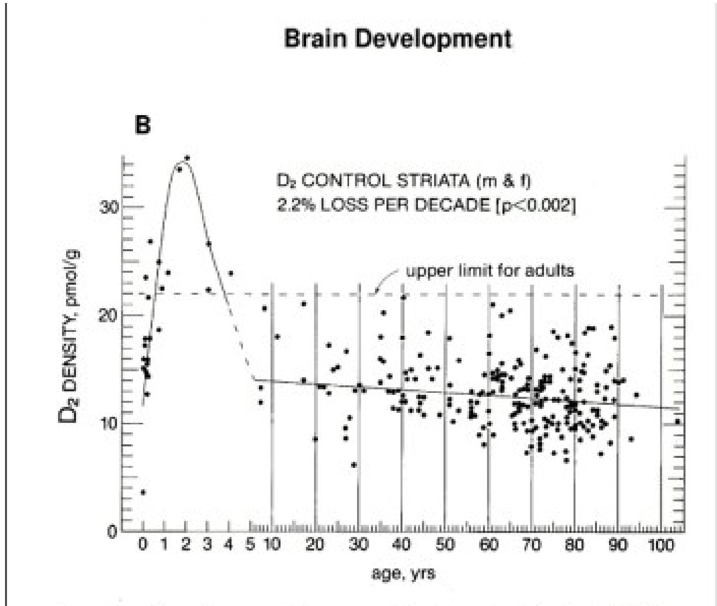
Big/Little DNN



Neuron Pruning is Natural in Biological System

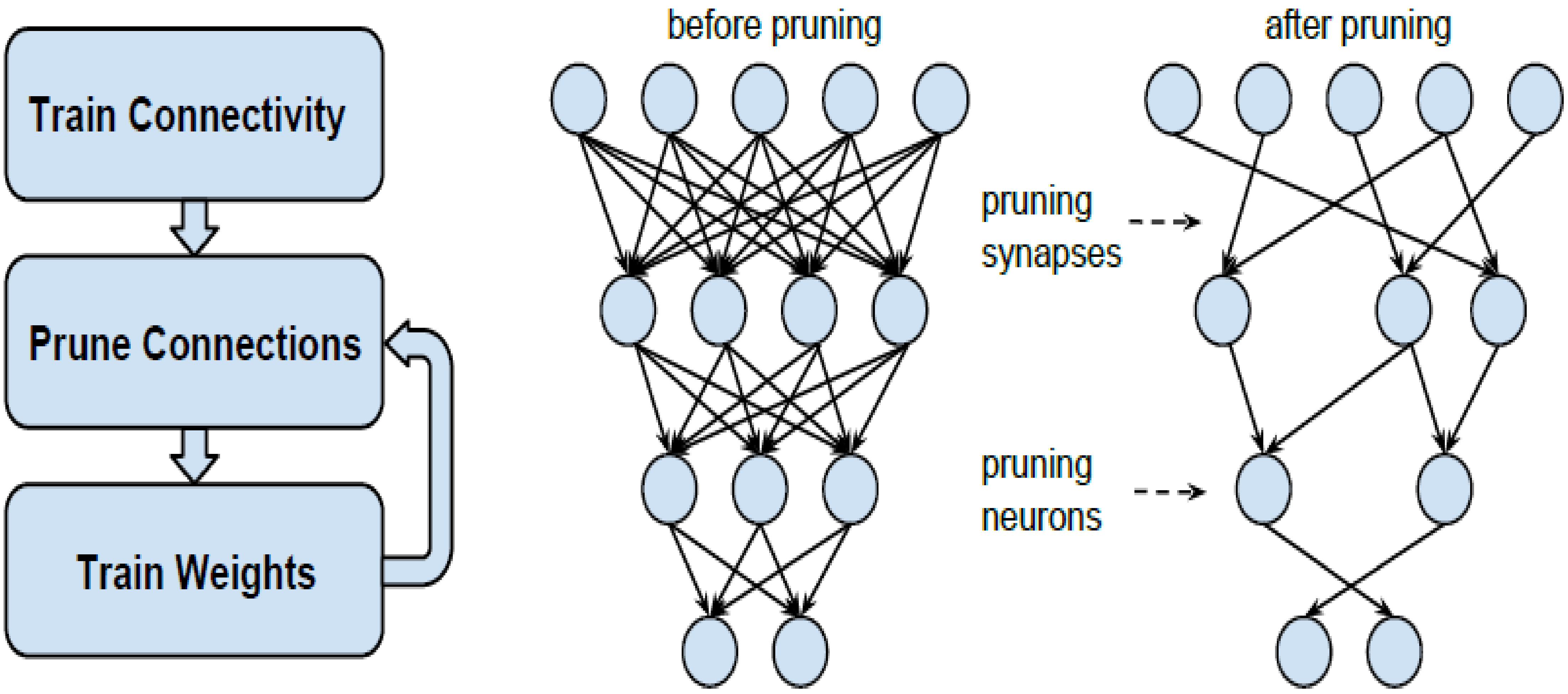
- The number of synapses increases before 2 years old and, then decrease due to pruning, possibly to reduce resources(e.g., energy) usage





Reproduced from Seeman et al.: Human Brain Dopamine Receptors in Children and Aging Adults. Synapse 1987; 1:399–404. Copyright © 1987, Wiley-Liss, Inc., a division of John Wiley and Sons, Inc. Reprinted by permission of John Wiley and Sons, Inc.

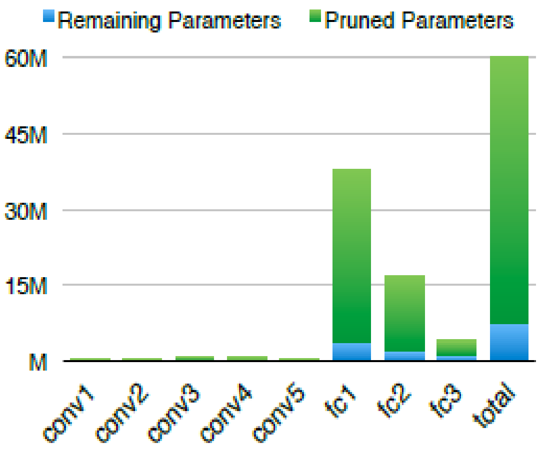
Pruning CNN


Train Connectivity
Prune Connections
Train Weights
before pruning
after pruning
pruning
synapses
pruning
neurons

Pruning CNN – Experimental Result

AlexNet

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%**

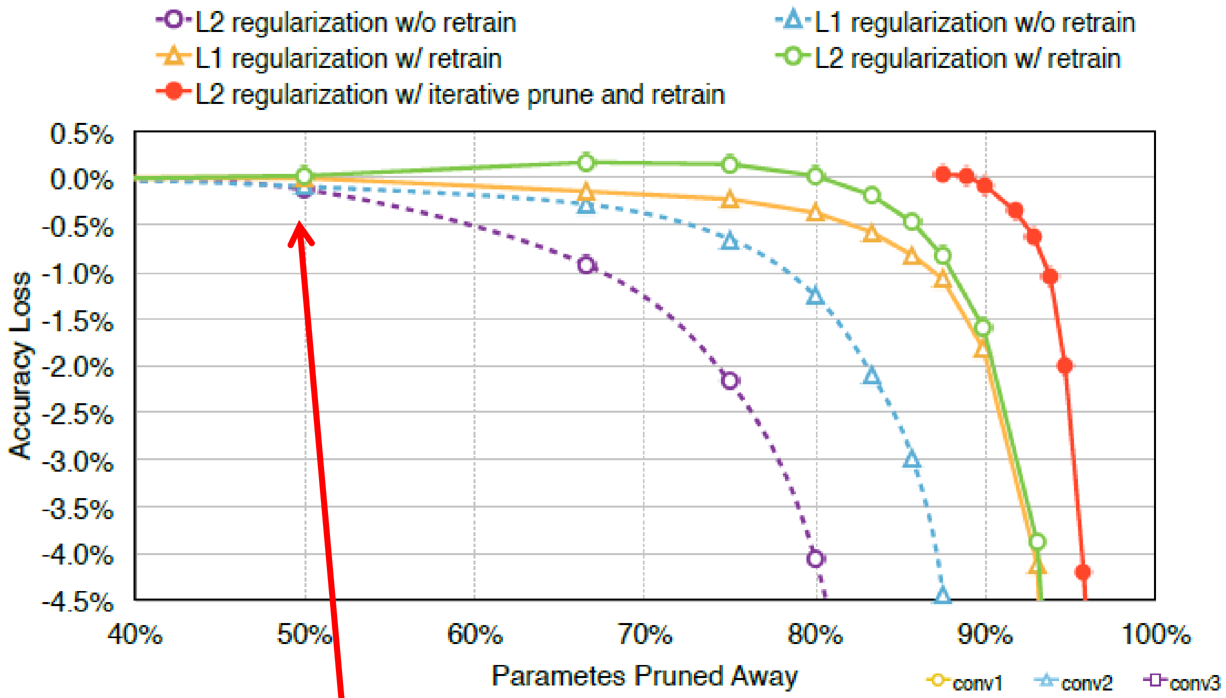


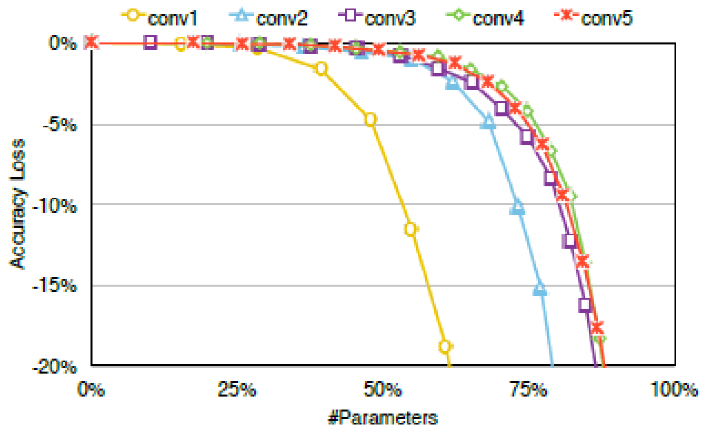
Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**×
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**×
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**×
VGG-16 Ref	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13**×

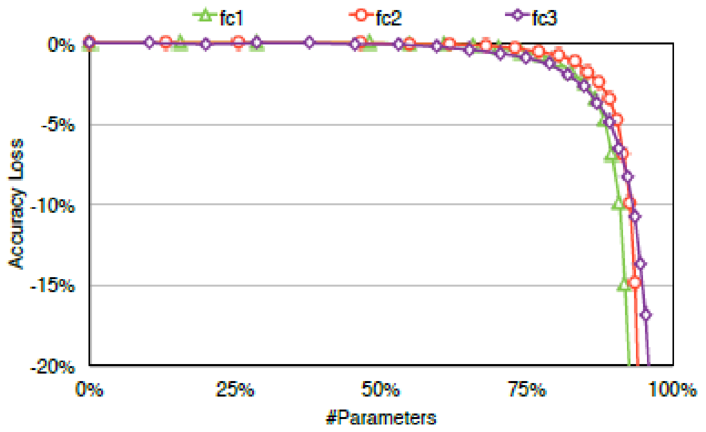
VGG-16

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%**

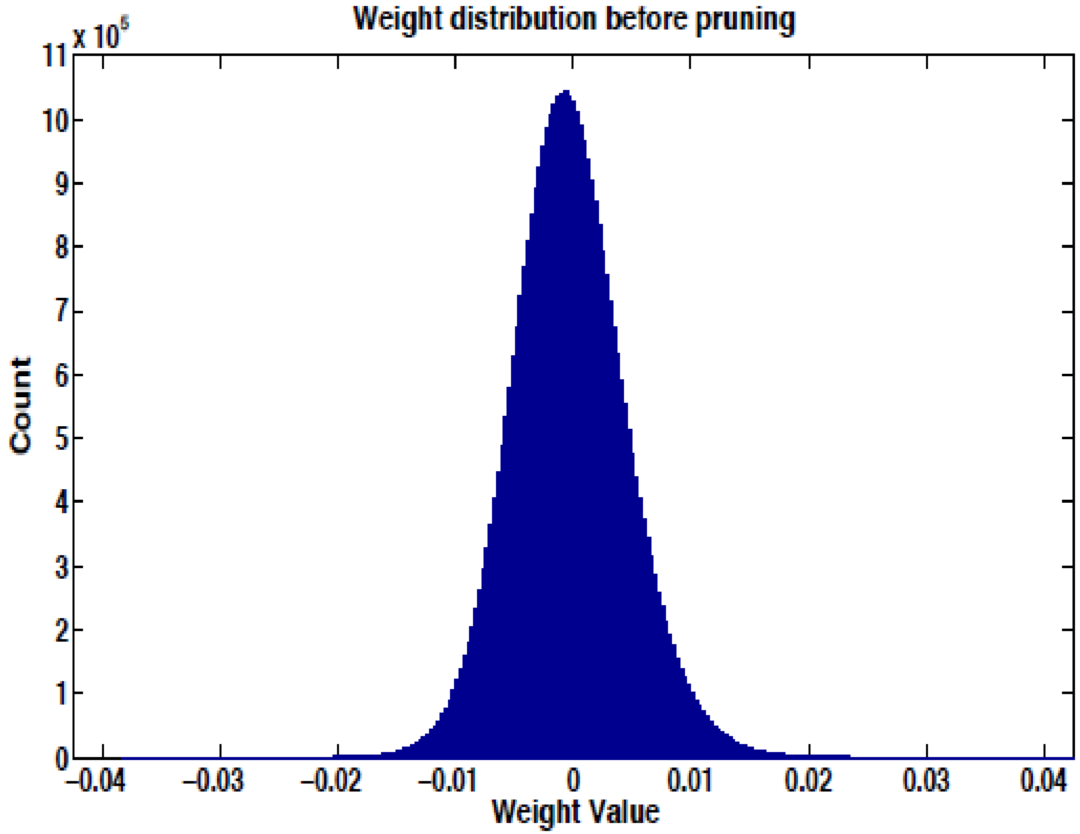
Pruning CNN – Experimental Result

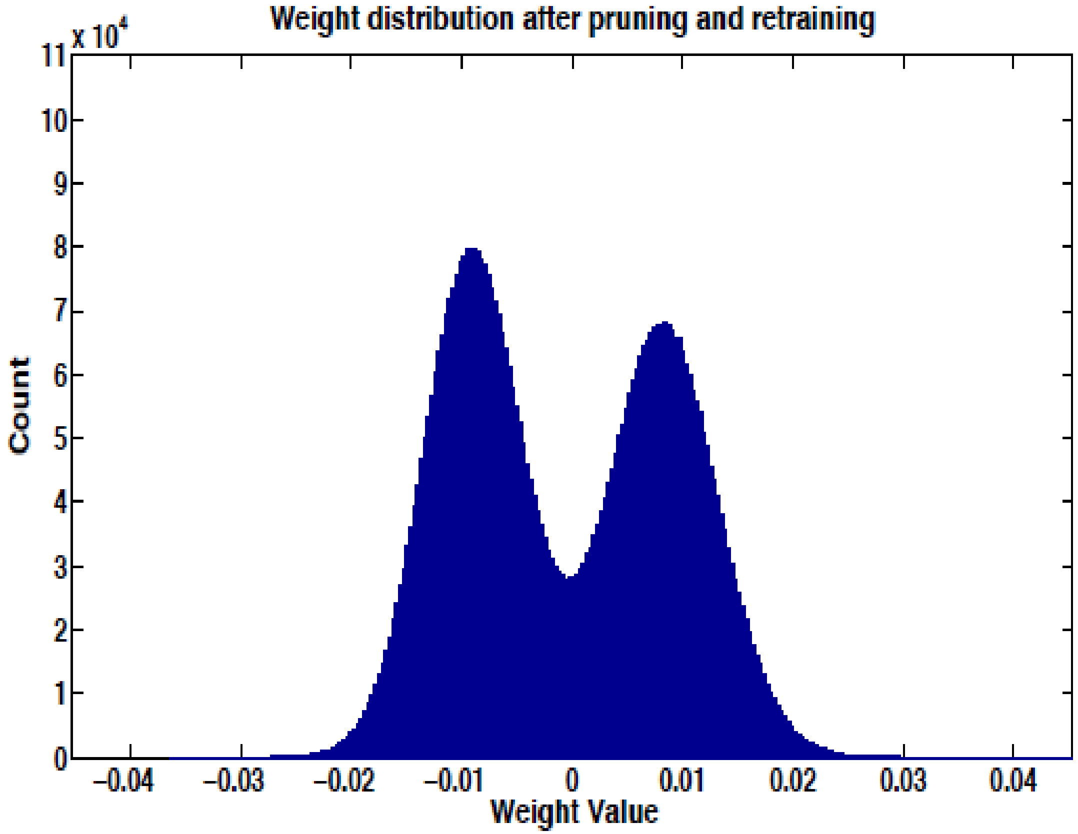






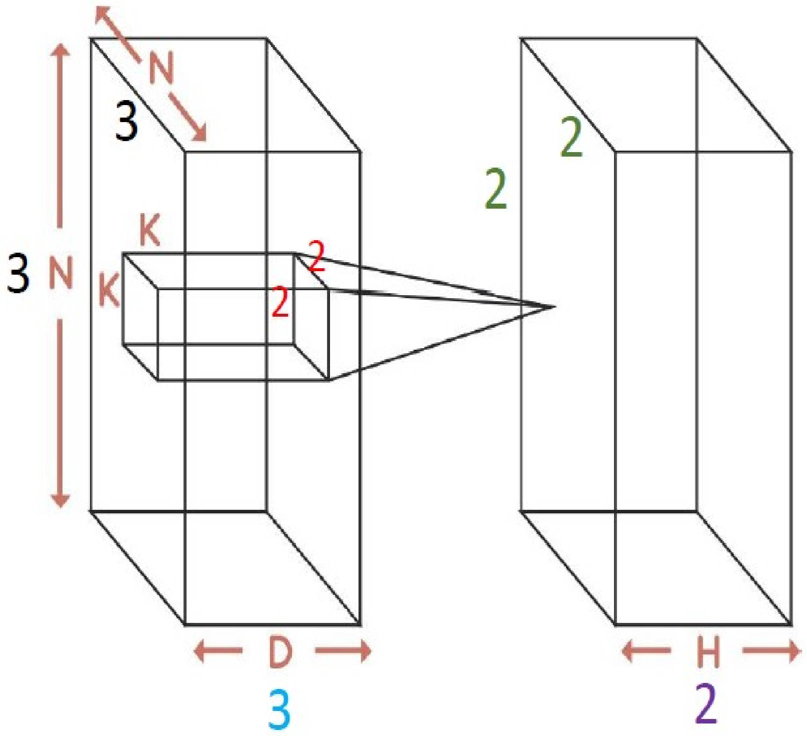
Pruning CNN – Weight Distribution

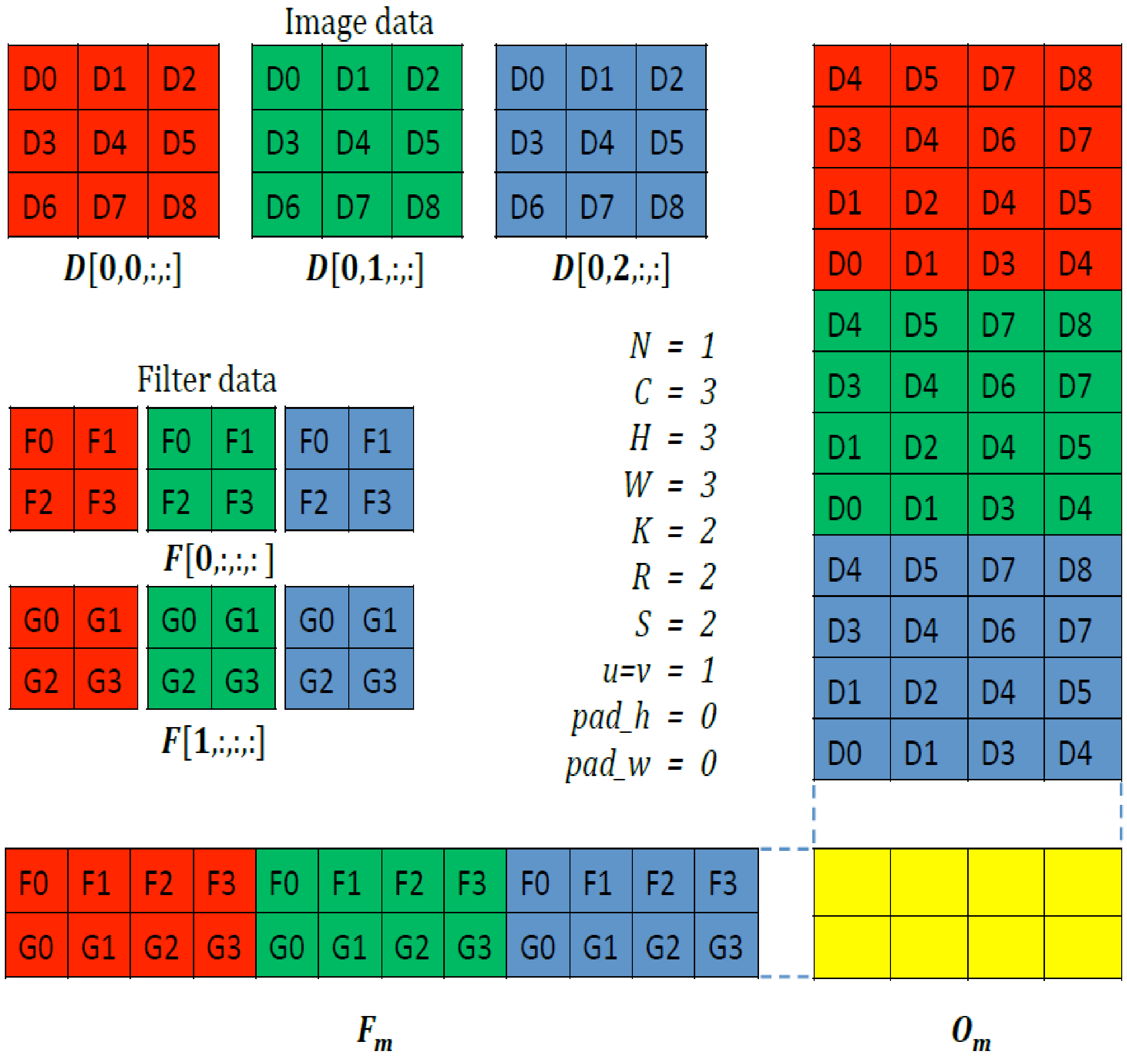




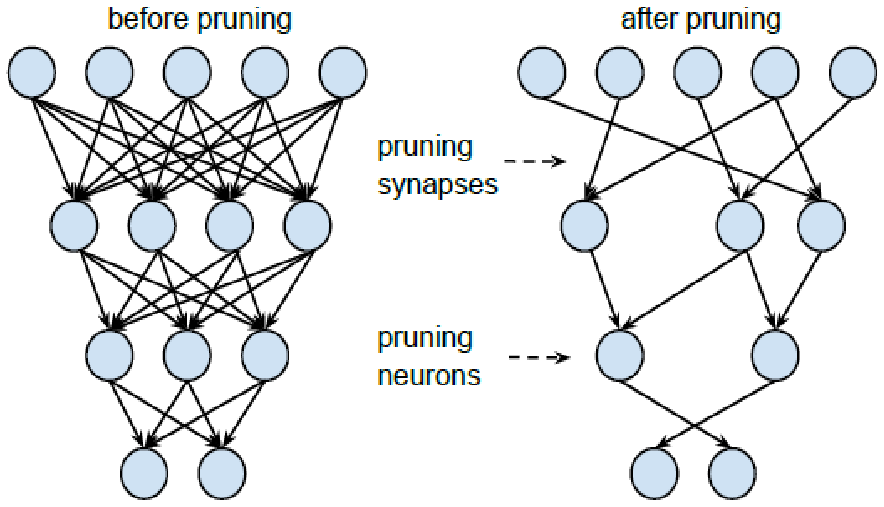
Convolution with Matrix Multiplication

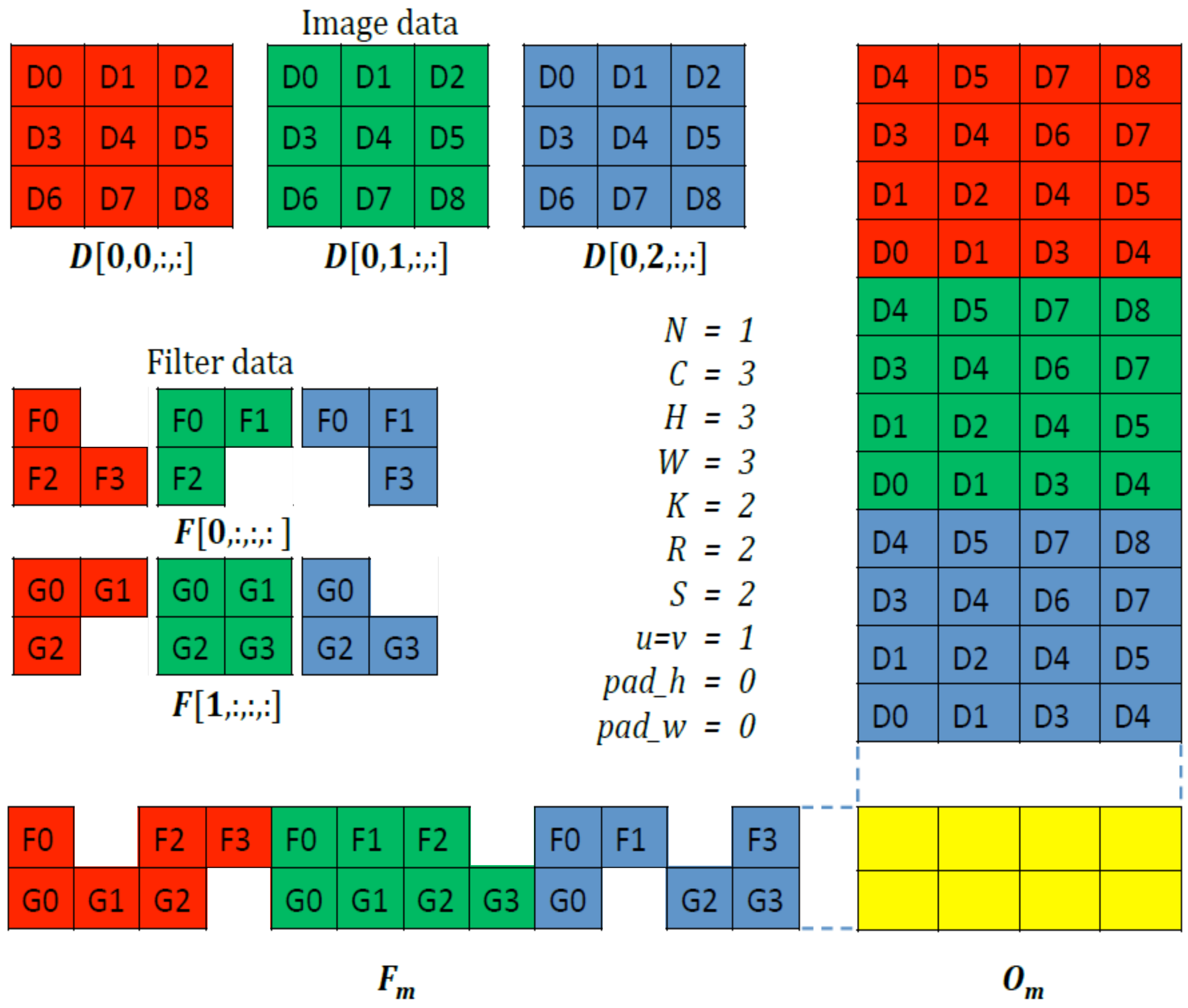
- Input: 3x3x3
- Output: 2x2x2
- Convolutional kernel: 3x2x2





Pruning Hardly Reduces the Runtime of Convolution on GPU

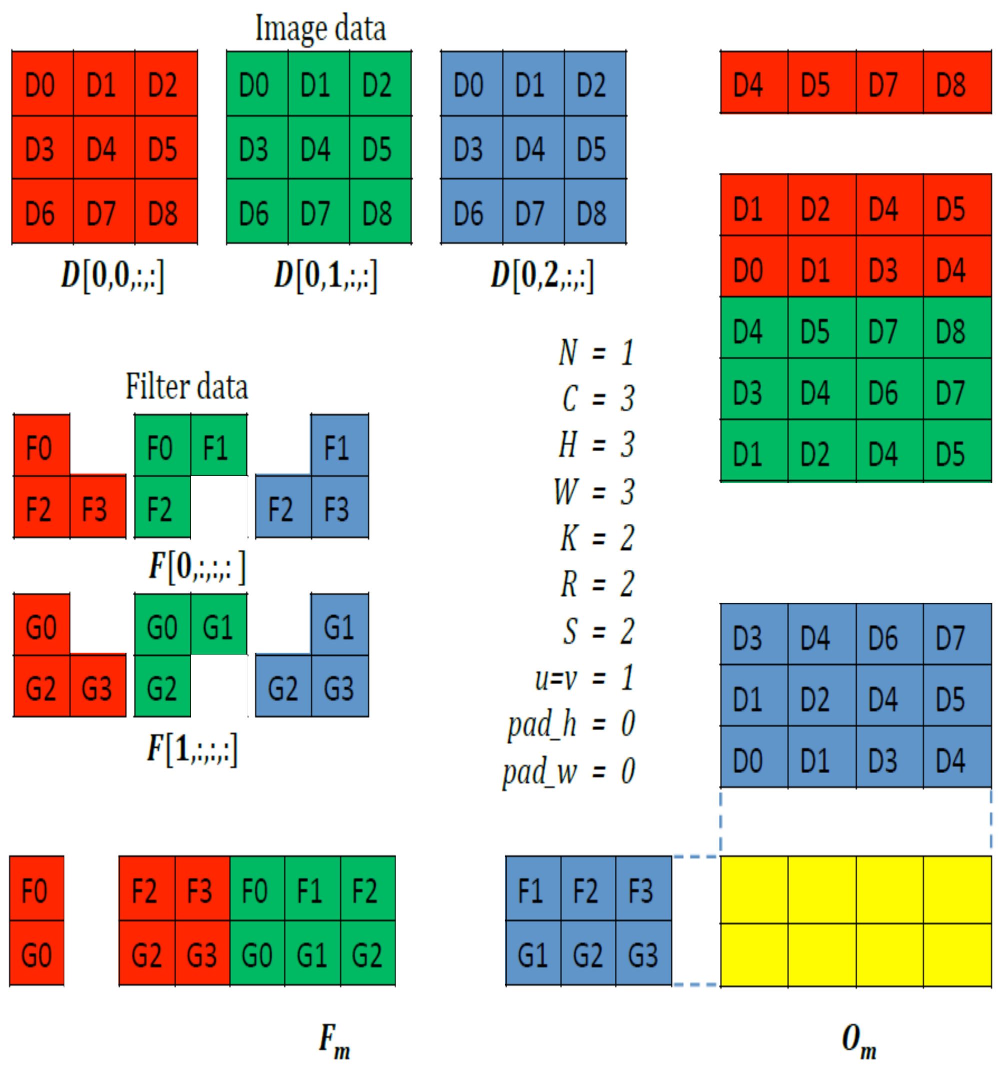




- Sparse matrix multiplication is effective when >90% values are zero
- Pruning makes 70% of weights (in convolution) zero averagely

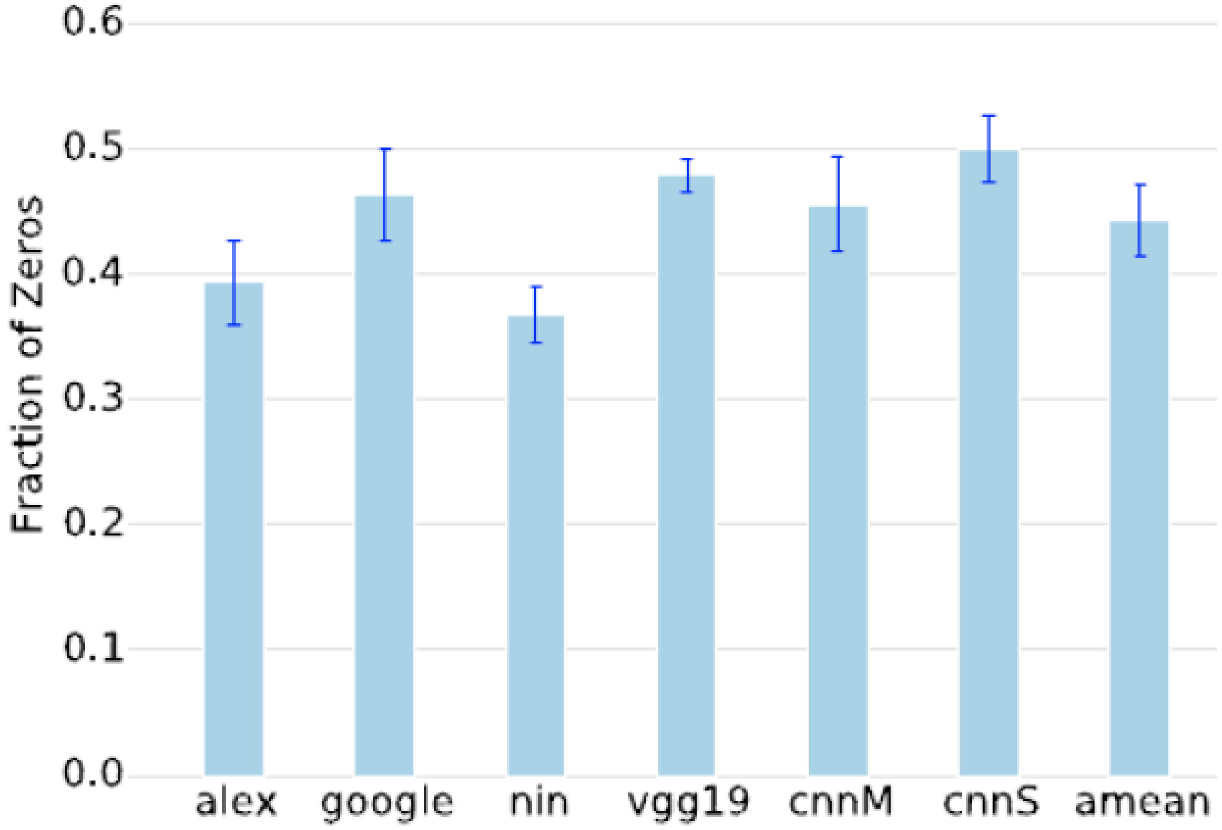
A Systematic Pruning Method – Group-wise Brain Damage

- For each input feature map, the same location of 2D filter elements is pruned
- Pruning is performed in an incremental manner
  - Repeat the followings until no more pruning candidate
    - ➢ Prune a column in F matrix and train the network to recover from accuracy loss
- Result
  - 3X reduction in # multiplications for AlexNet



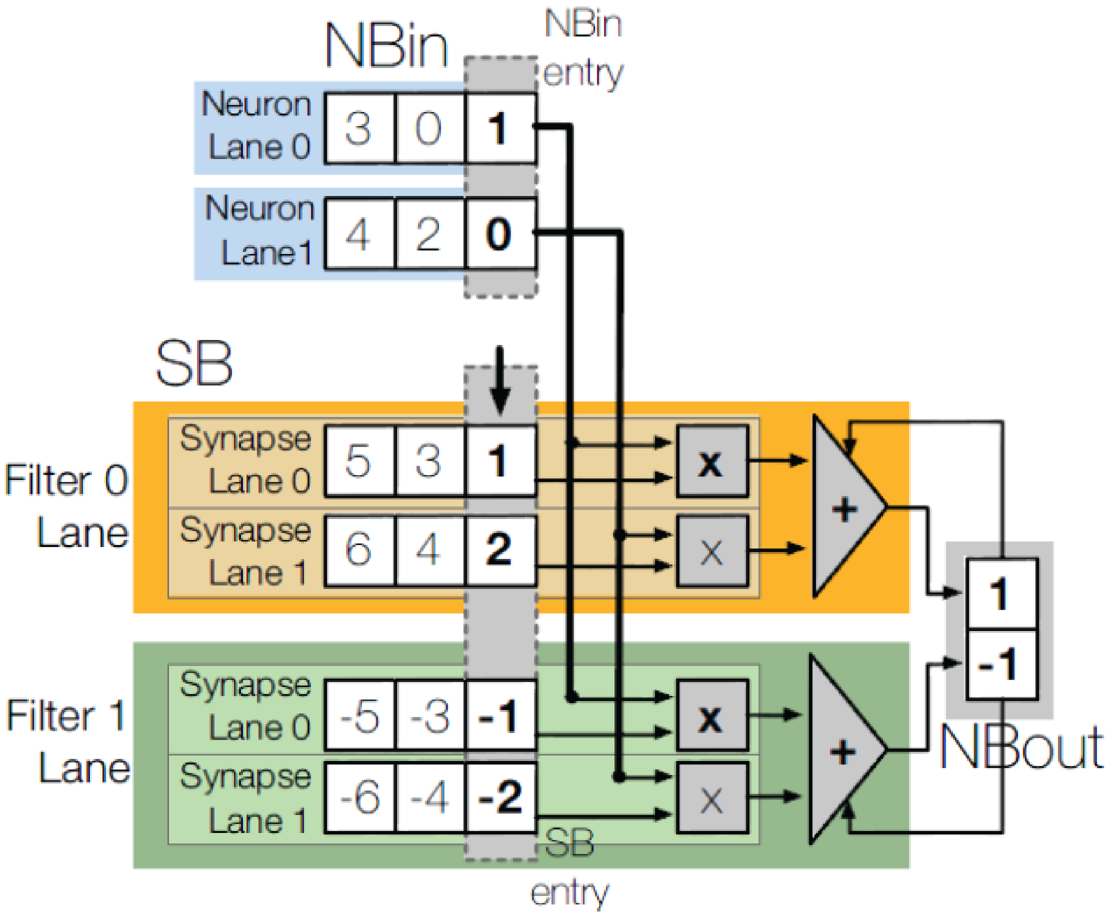
Cnvlutin

- Observation: average 44% of input neuron values are zero!

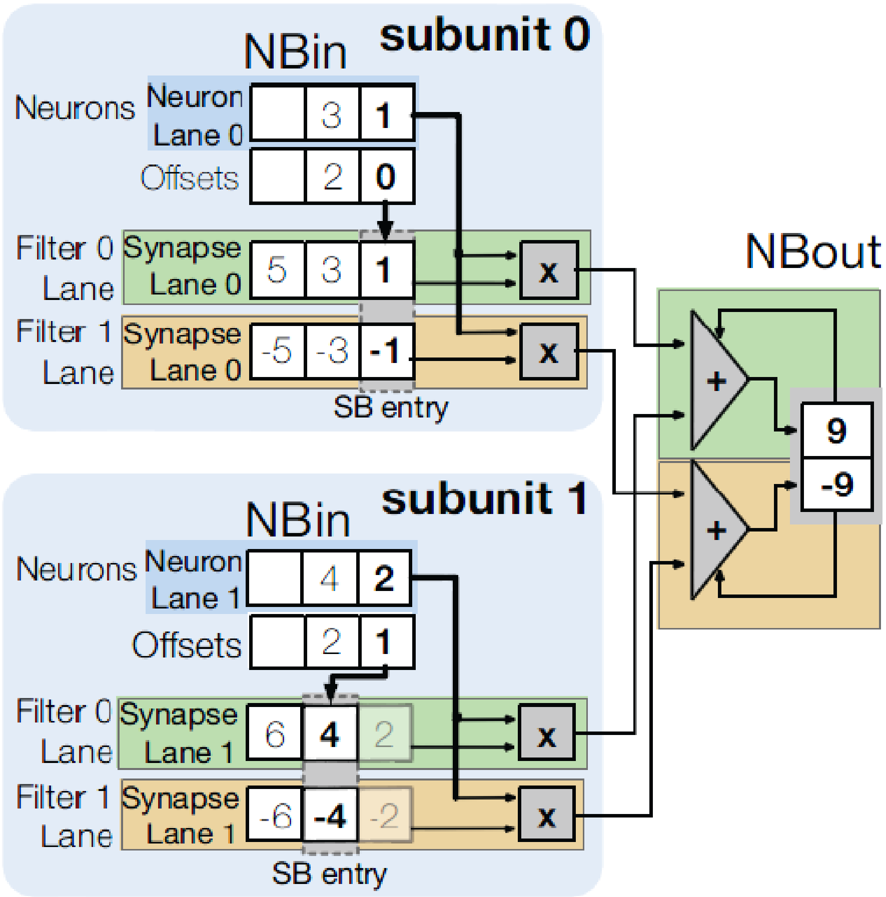


Average Fraction of Input Neuron Values are Zero

Cnvlutin

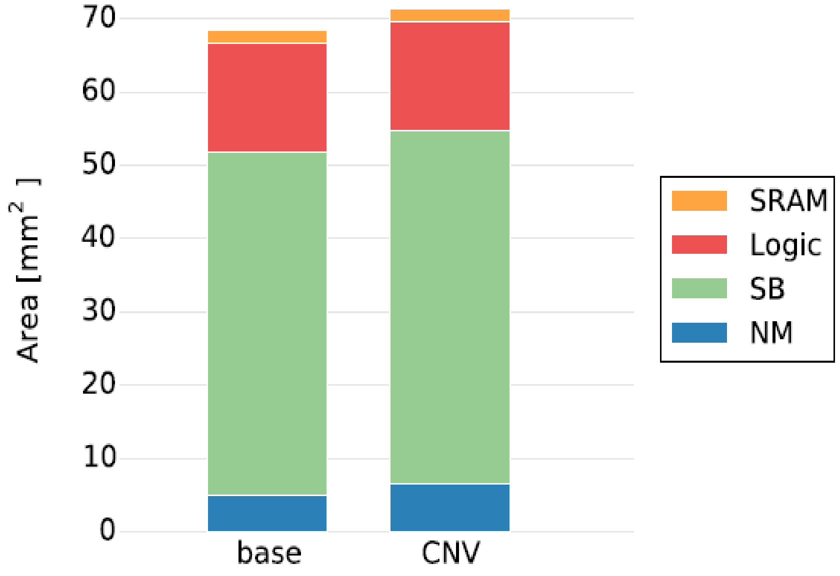


DaDianNao (Baseline) Architecture

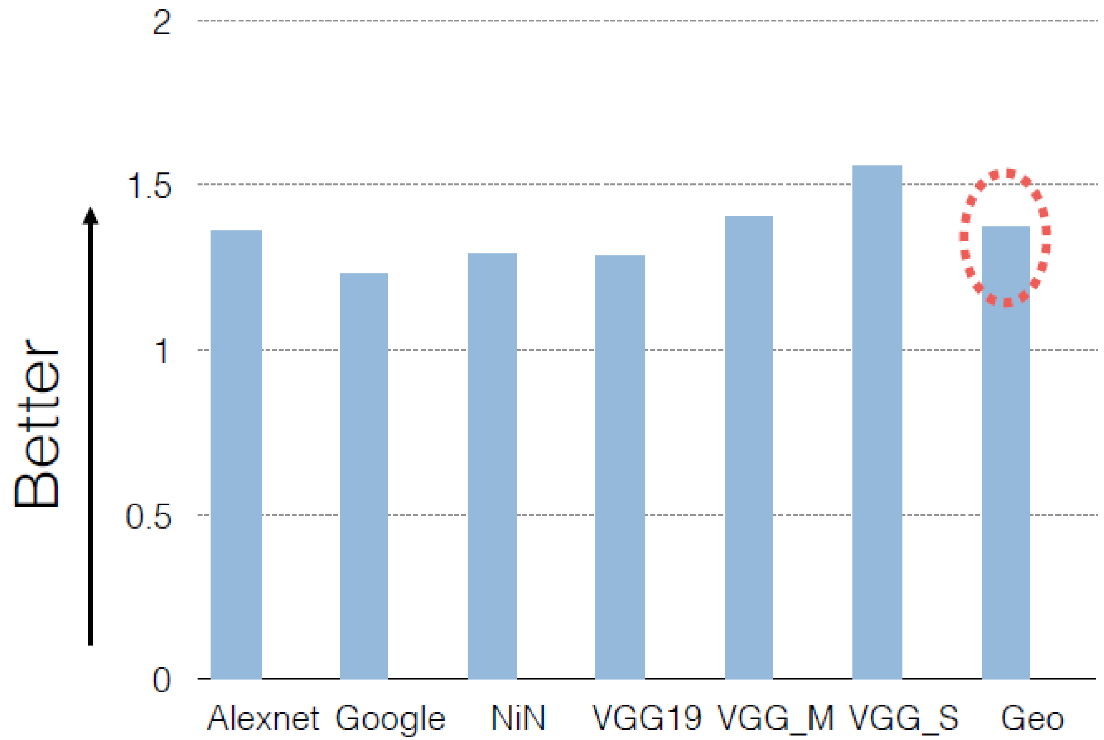


Cnvlutin Architecture

Cnvlutin – Exprimental Result



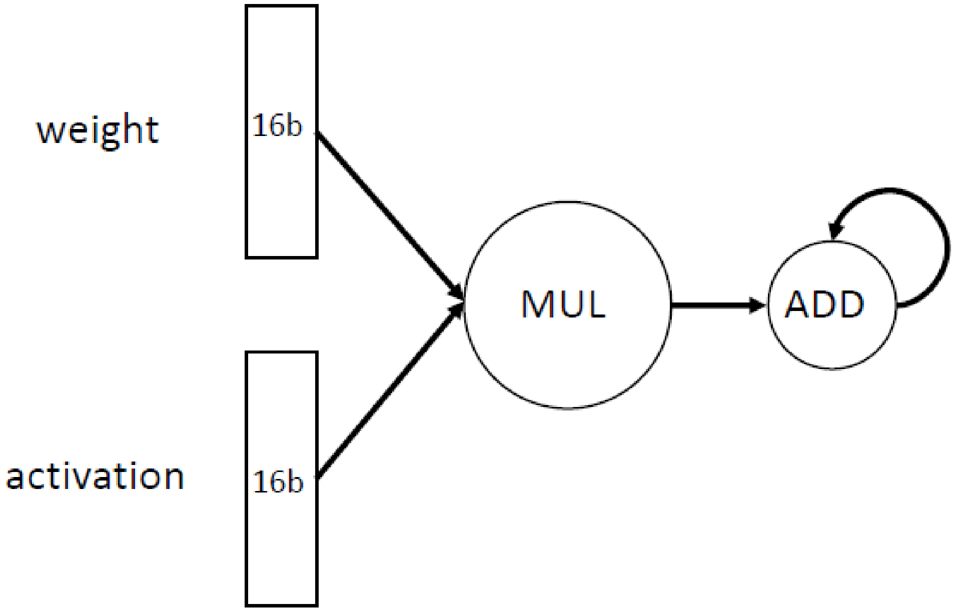
Only +4.5% in area overhead



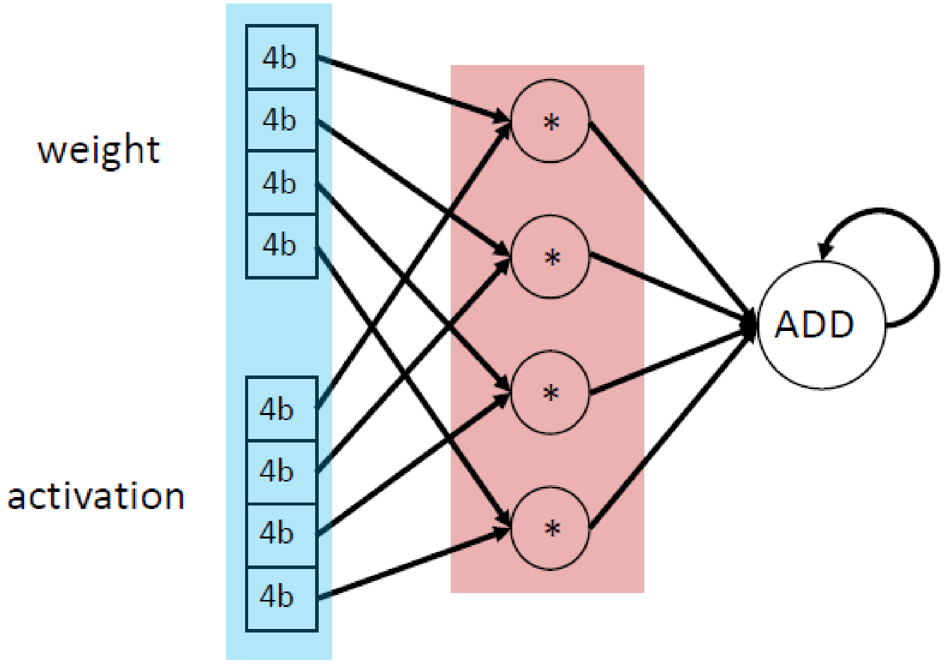
1.37x Performance on average

Narrow Data

- Performance improvement due to narrow data
  - E.g., 16bit → 4bit data, 4X speedup with the same memory bandwidth consumption



Conventional convolution

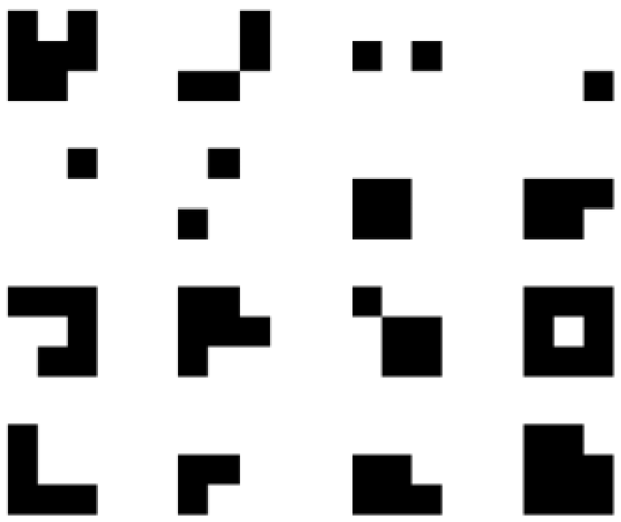


Convolution with narrow data

Binary Nets

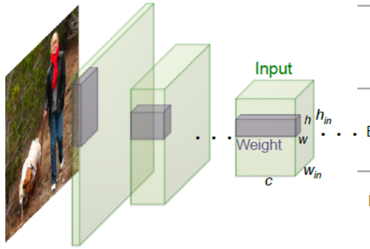
- Binary Connect(BC)
  - Weights {-1, 1}, Activations 32-bit float
  - MAC → addition/subtraction
  - Accuracy loss: 19% on AlexNet
- Binarized Neural Networks(BNN)
  - Weights {-1, 1}, Activations {-1, 1}
  - MAC → XNOR
  - Accuracy loss : 29.8% on AlexNet

Binary Filters



Scale the Weights and Activations

- Binary Weight Nets(BWN)
  - Weight {-α, α} → except first and last layers are 32-bit flaot
  - Activations: 32-bit float
  - α determined by the l1-norm of all weights in a layer
  - Accuracy loss: 0.8% on AlexNet
- XNOR-Net
  - Weights {- α, α}
  - Activations {-βi, βi} → except first and last layers are 32-bit float
  - βi determined by the l1-norm of all activations across channels for given position i of the input feature map
  - Accuracy loss: 11% on AlexNet



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Ternary Nets

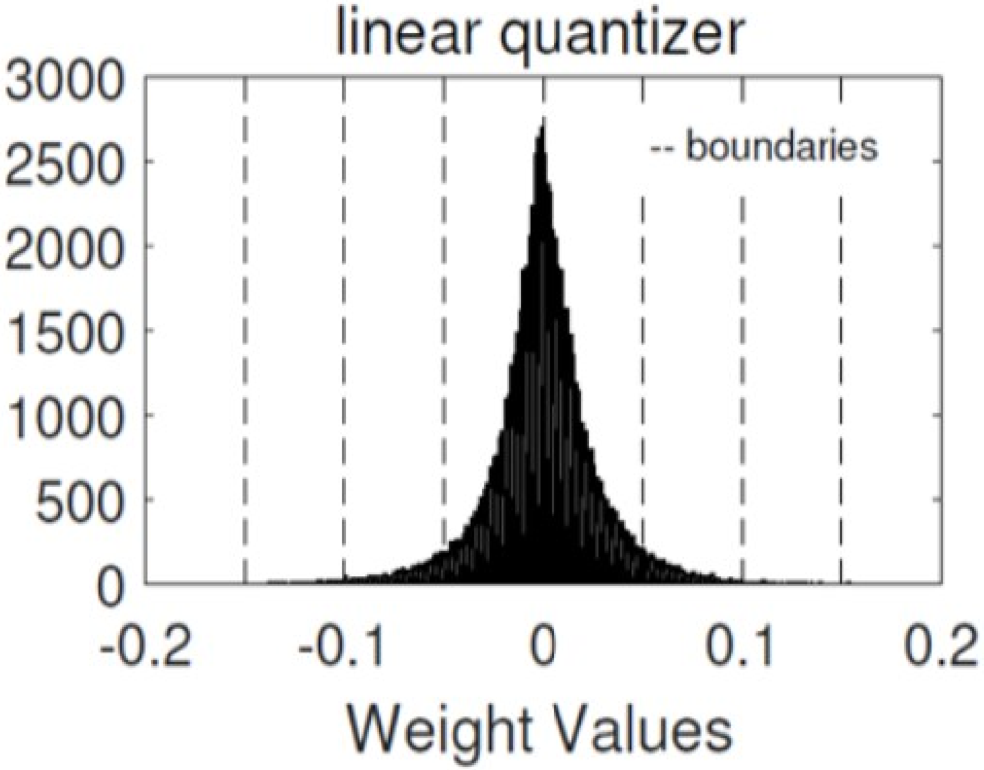
- Allow for weights to be zero
  - Increase sparsity, but also increase number of bits (2-bits)

- Ternary Weight Nets(TWN)
  - Weights {-w, 0, w} → except first and last layers are 32-bit float
  - Activations: 32-bit float
  - Accuracy loss: 3.7% on AlexNet

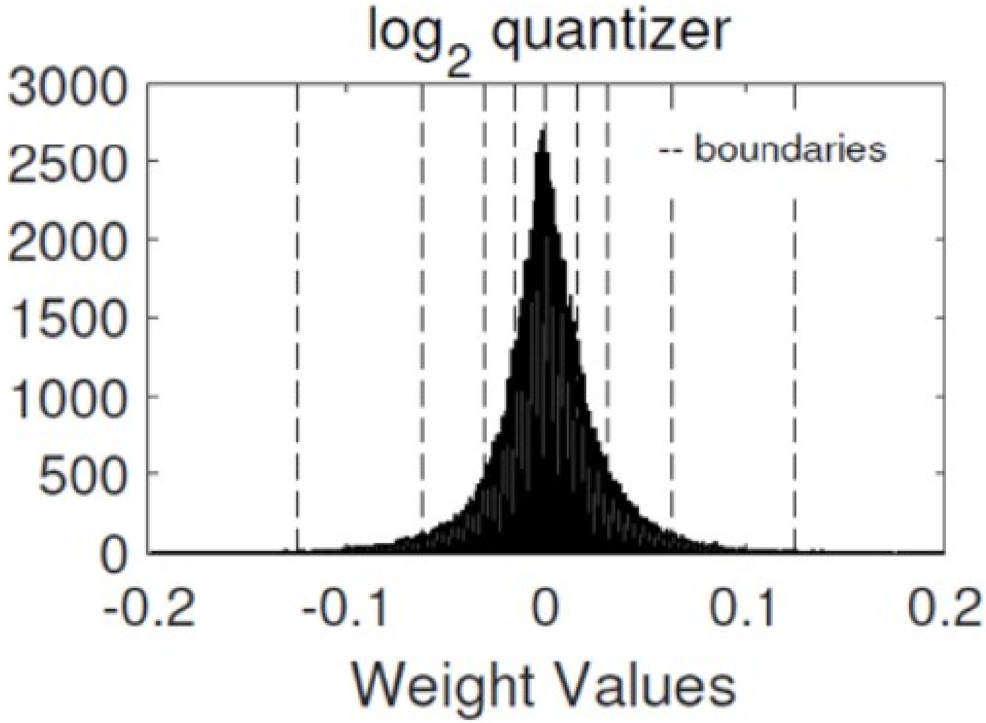
- Trained Ternary Quantization(TTQ)
  - Weights {-w1, 0, w2} → except first and last layers are 32-bit float
  - Activations: 32-bit float
  - Accuracy loss: 0.6% on AlexNet

Computed Non-linear Quantization

Log Domain Quantization



Product = X * W



Product = X << W