Hardware Design for Deep Learning

Lei Jiang

Jiang60@iu.edu

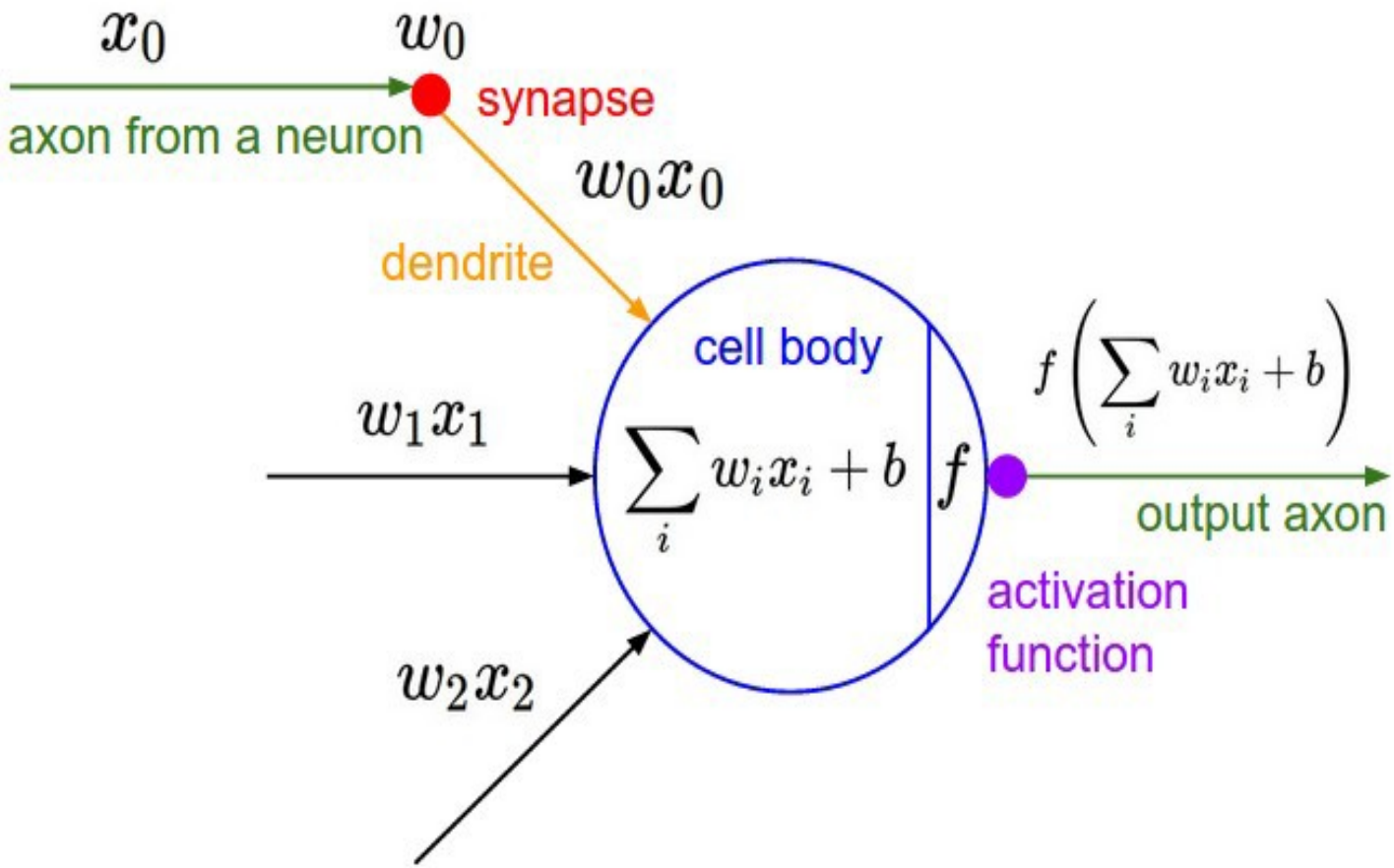
Indiana University Bloomington

Part of slide is developed based on the Tutorial on Hardware Architectures for Deep Neural Networks, Joel Emer, Vivienne Sze, Yu-Hsin Chen

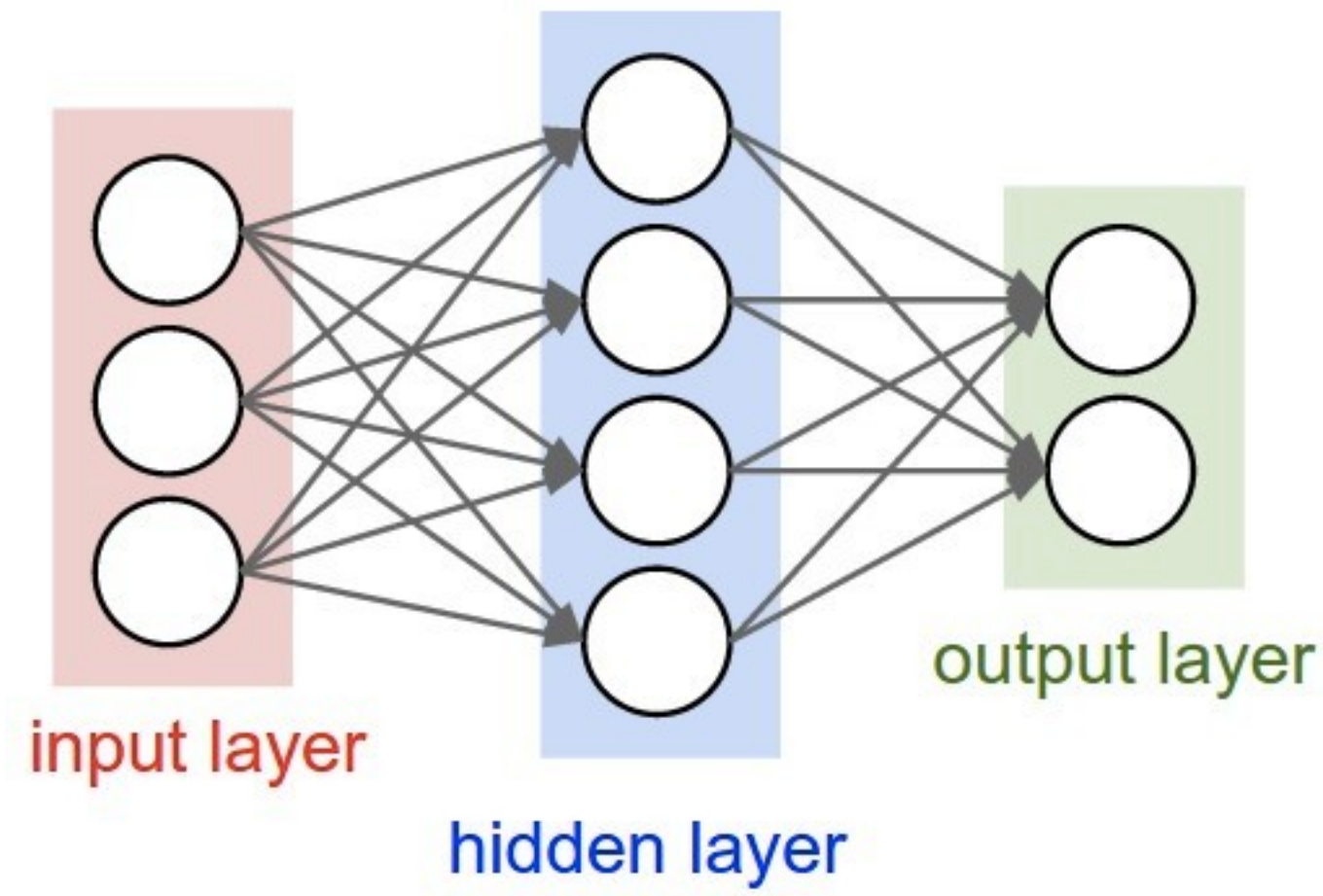
Disclaimer

1. You understand neural networks **better** than me.
2. We will **NOT** focus on hardware or algorithm **details**.
3. We will focus on the **interplay** between hardware and algorithm, e.g., XNOR is cheap from the hardware perspective, then algorithm needs to convert multiplication to XNOR. But **why** XNOR can replace multiplication is **beyond** today's content.
4. We will think from the **hardware side** temporarily.
5. Most of content today are **ONLY** about inference, but **NOT** related to training.

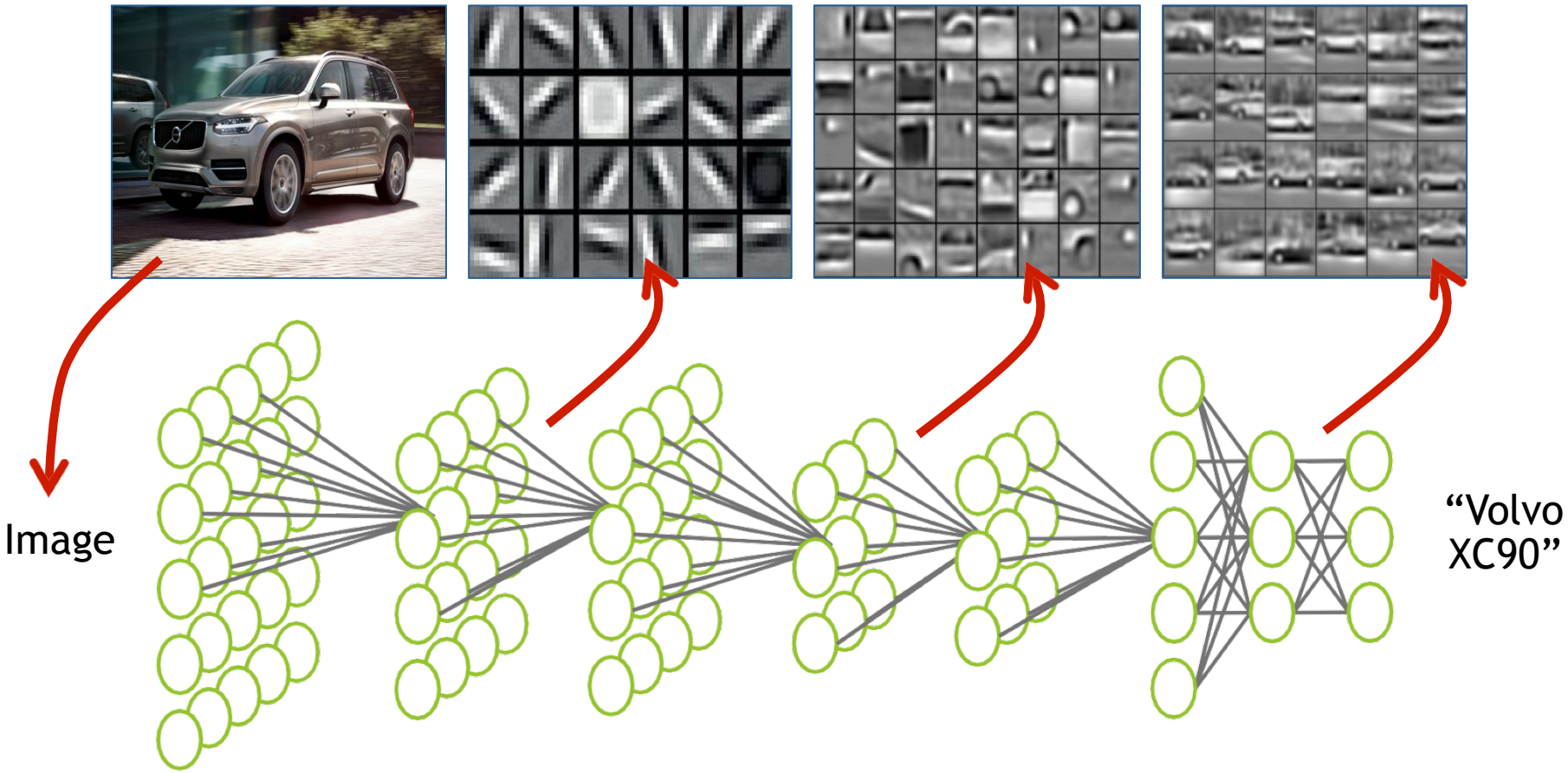
Natural Networks: Weighted Sum



Natural Networks: Weighted Sum



Deep Learning



ImageNet Challenge

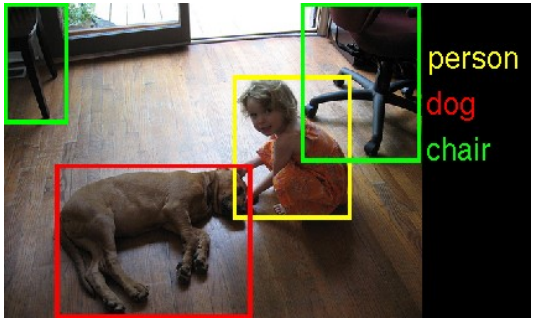
IMAGENET

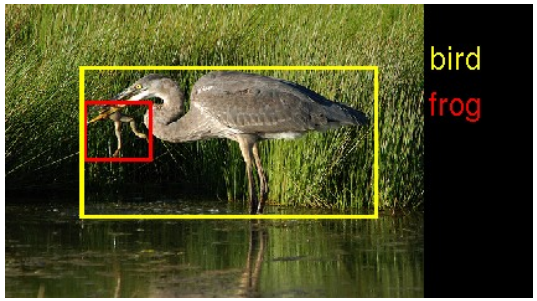
Image Classification Task:

1.2M training images • 1000 object categories

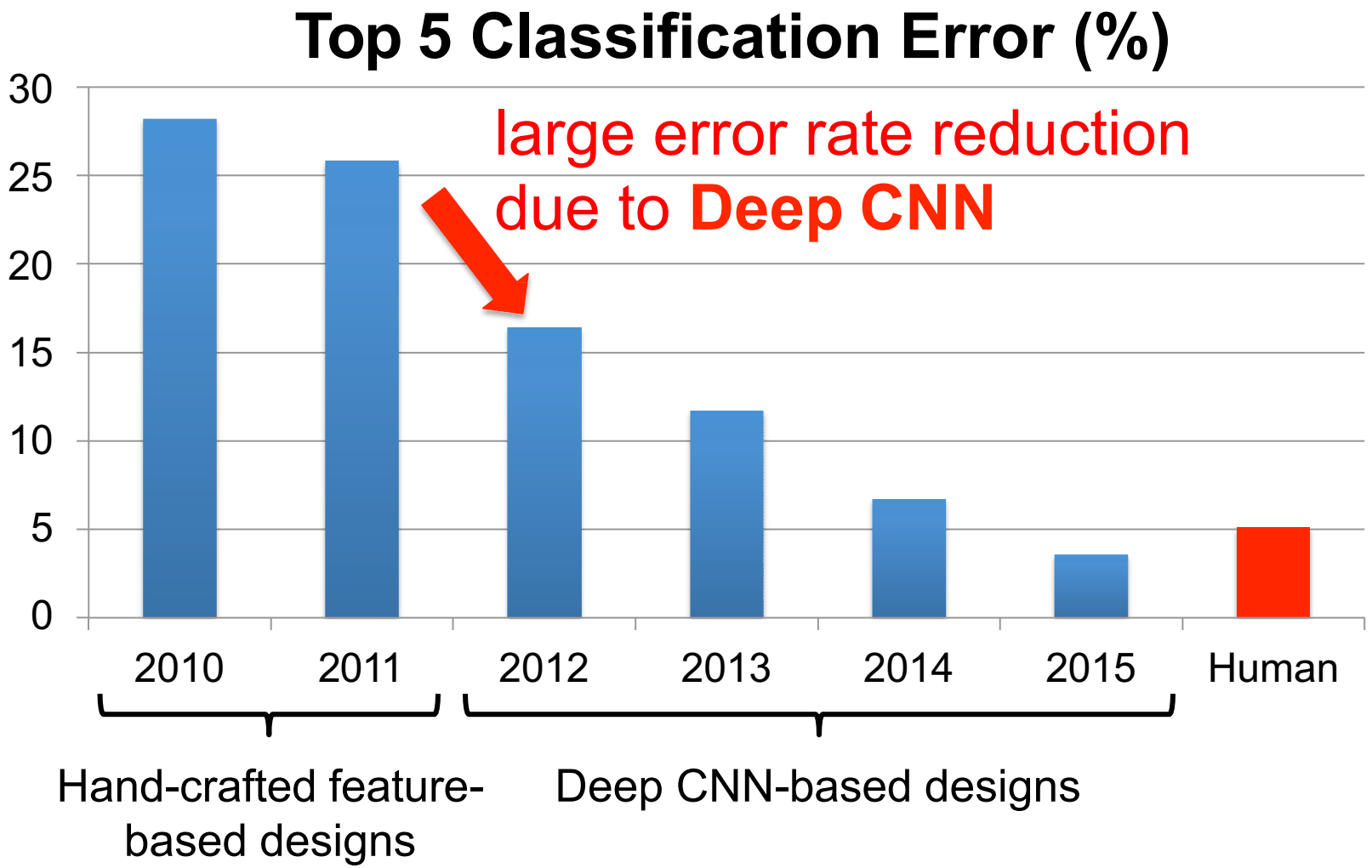
Object Detection Task:

456k training images • *200 object categories*

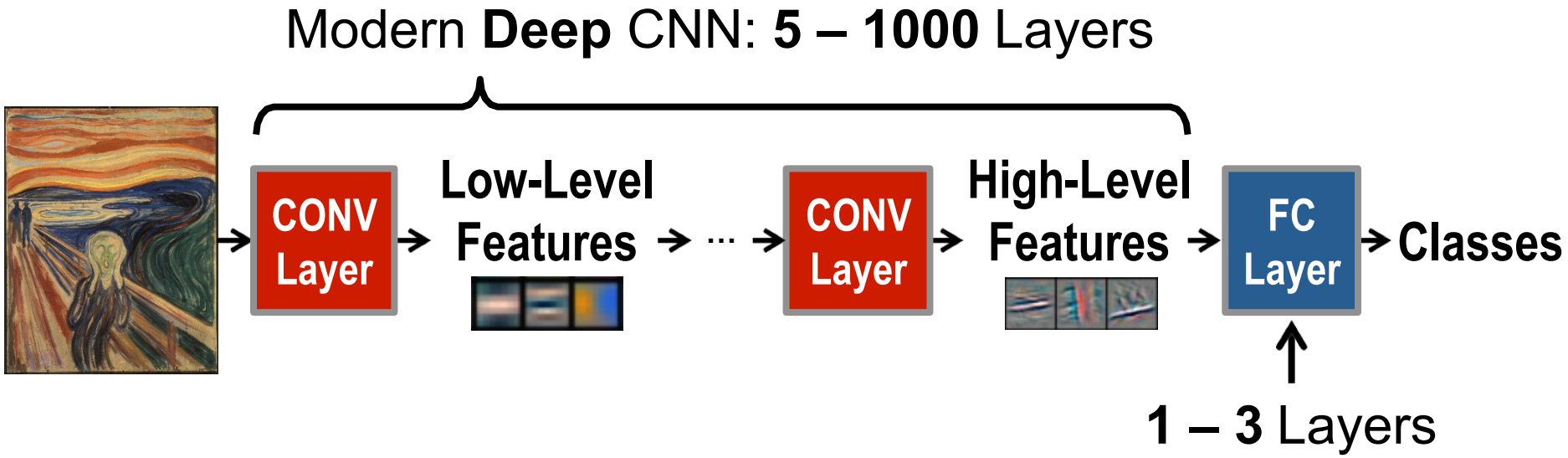




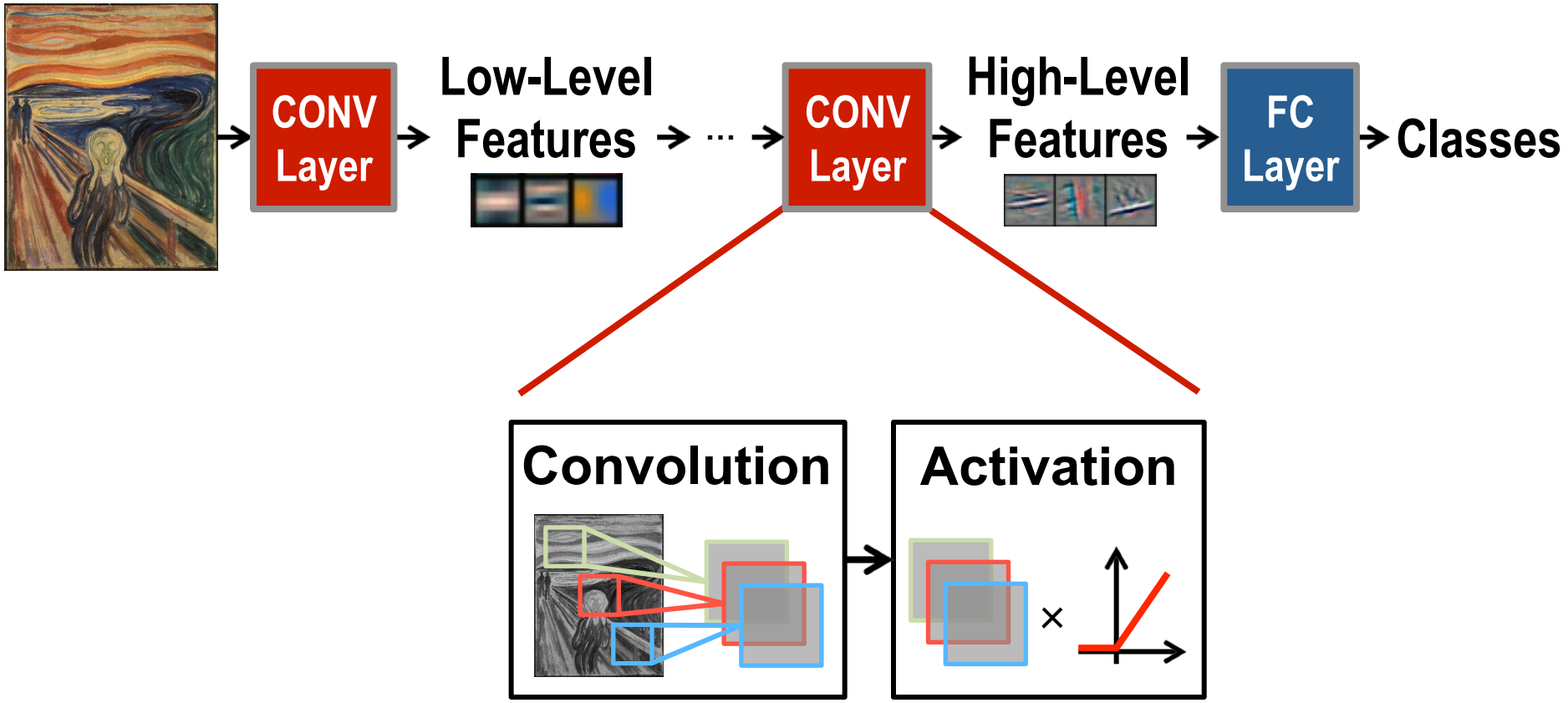
ImageNet Challenge



Convolutional Neural Network

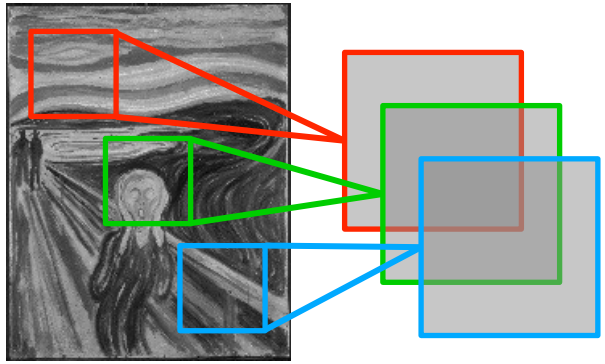


Convolutional Neural Network



Convolutional Neural Network

CONV Layer → NORM Layer → POOL Layer → CONV Layer → High-Level Features → FC Layer → Classes



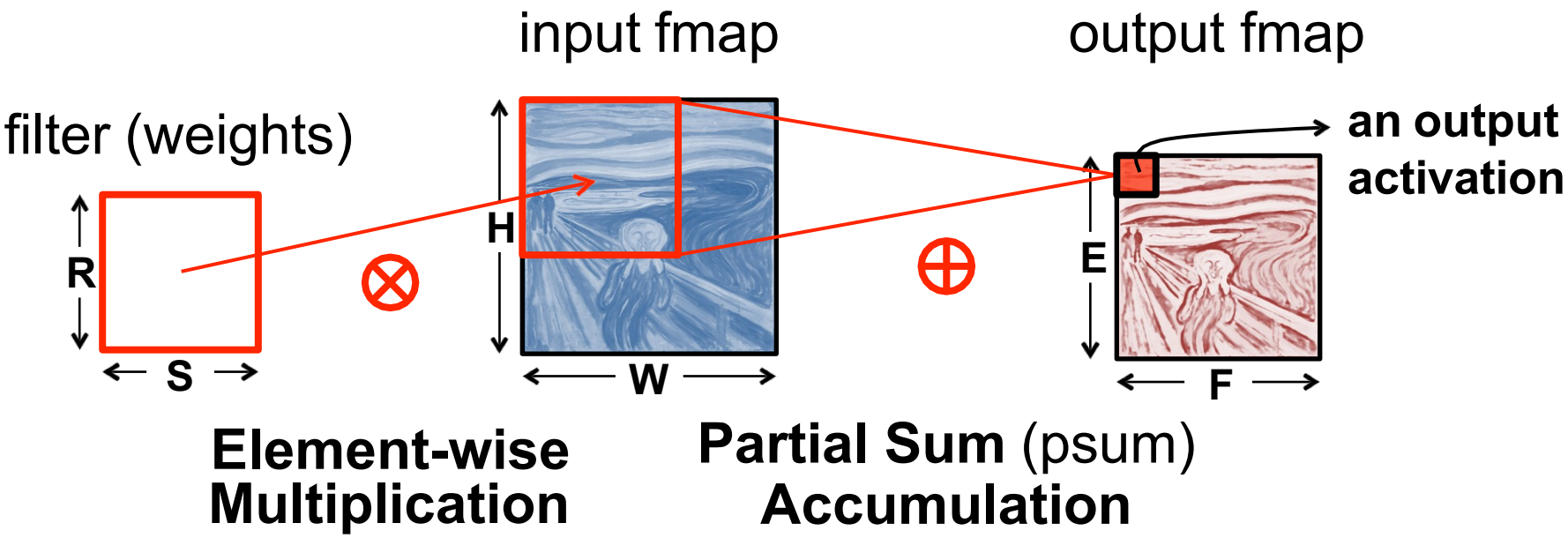
Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

CNN Summary

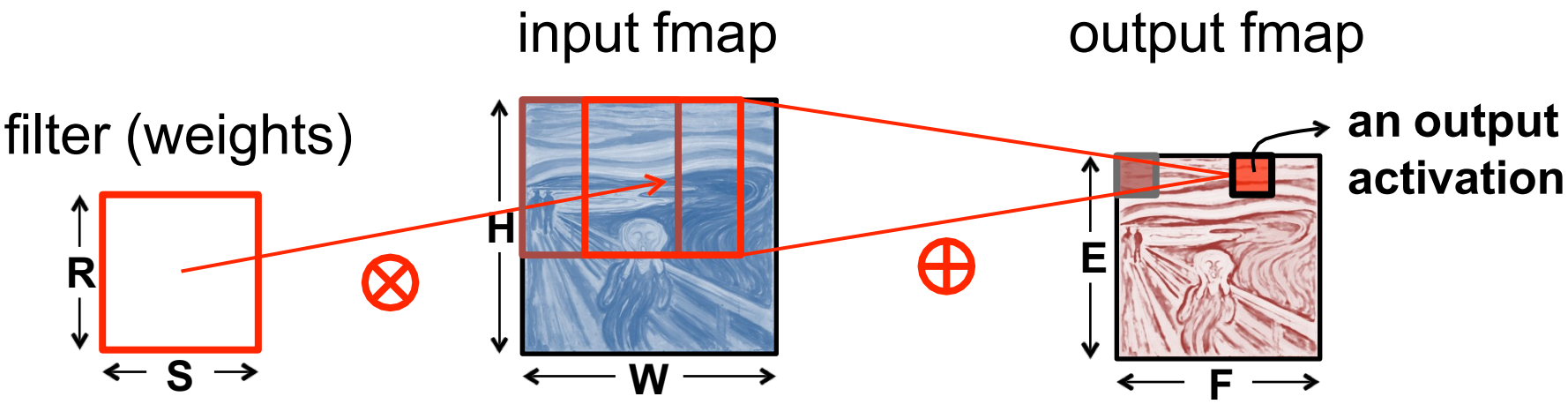
Metrics	LeNet-5	AlexNet	VGG-16	GoogLeNet (v1)	ResNet-50
Top-5 error	n/a	16.4	7.4	6.7	5.3
Input Size	28x28	227x227	224x224	224x224	224x224
# of CONV Layers	**2**	**5**	**16**	**21 (depth)**	**49**
Filter Sizes	5	3, 5,11	3	1, 3 , 5, 7	1, 3, 7
# of Channels	1, 6	3 - 256	3 - 512	3 - 1024	3 - 2048
# of Filters	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!

Convolutional Layer

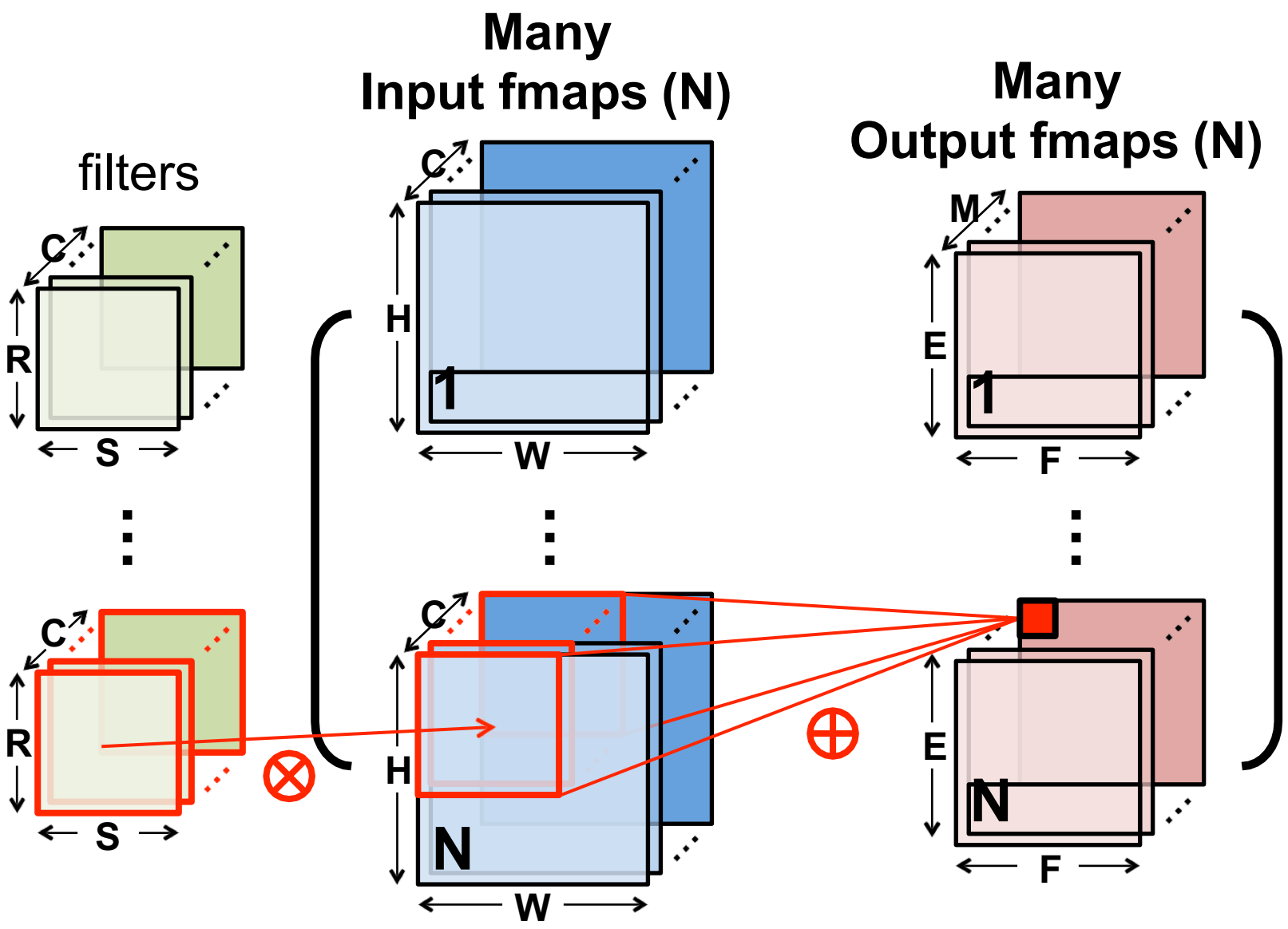


Convolutional Layer

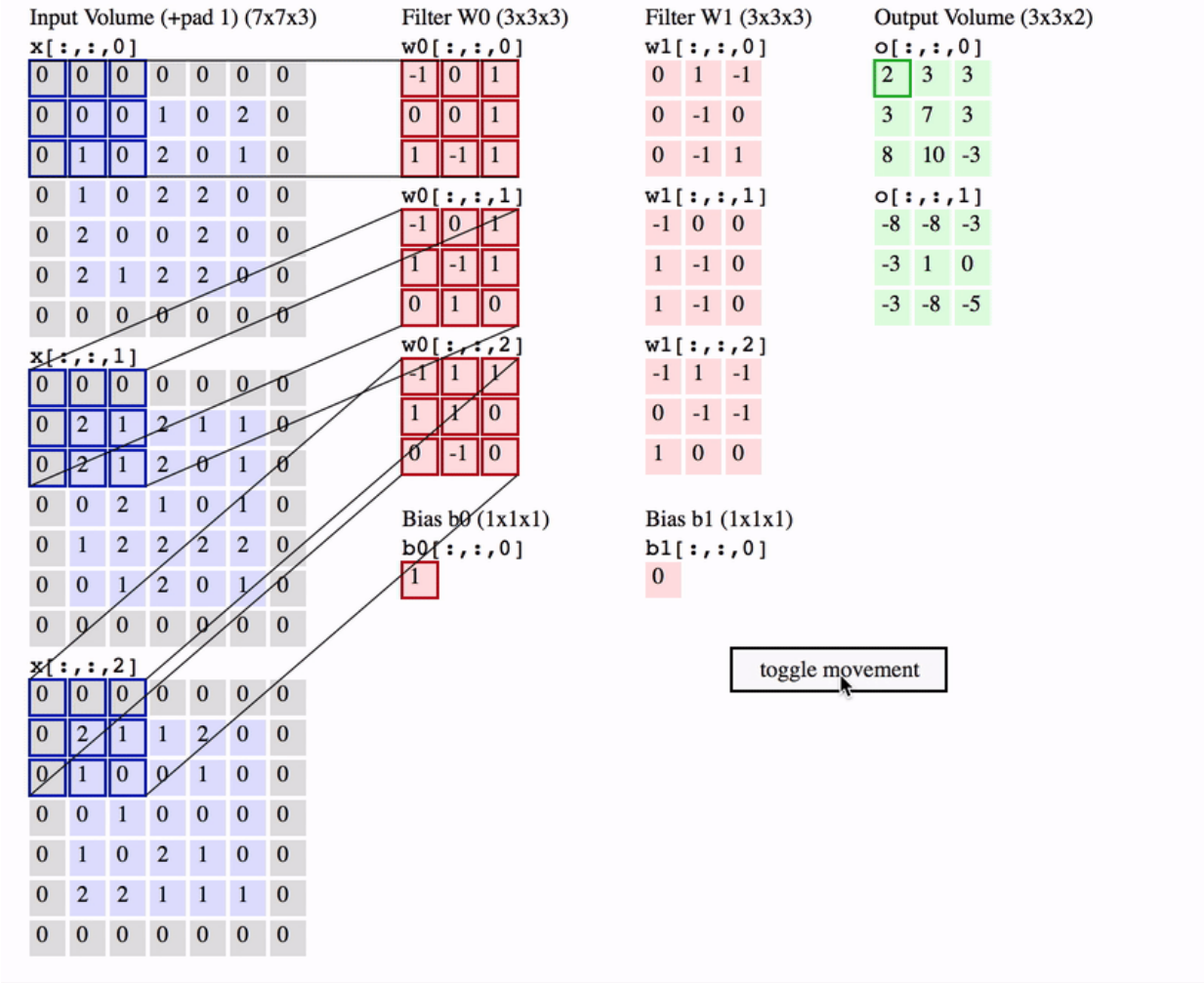


Sliding Window Processing

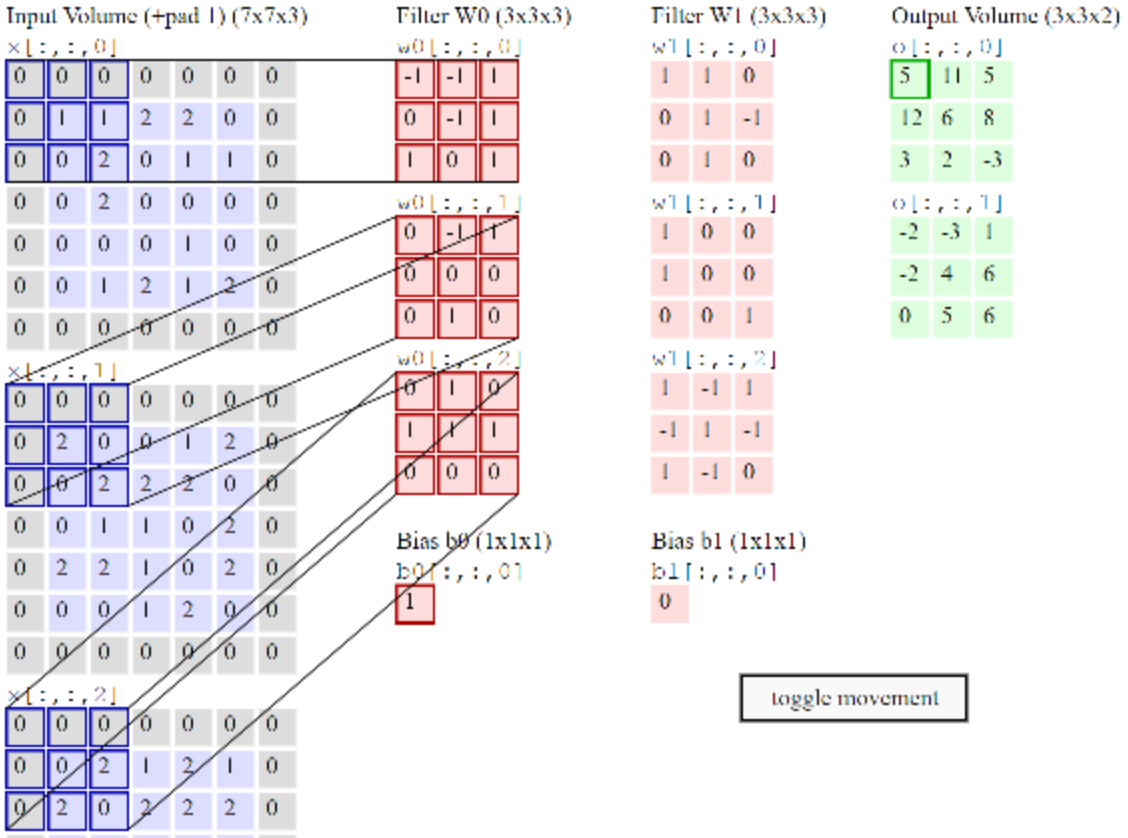
Convolutional Layer



Convolutional Layer



Convolutional Layer



0*-1+0*-1+0*1+0*0+1*-1+1*1+0*1+0*0+2*1=2

0*0+0*-1+0*1+0*0+2*0+0*0+0*0+0*1+2*0=0

0*0+0*1+0*0+0*1+0*1+2*1+0*0+2*0+0*0=2

Convolutional Layer

Output fmaps (O)

Biases (B)

Input fmaps (I)

Filter weights (W)

$$\mathbf{O}[n][m][x][y] = \text{Activation}(\mathbf{B}[m] + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \mathbf{I}[n][k][Ux+i][Uy+j] \times \mathbf{W}[m][k][i][j]),$$

$$0 \le n < N, 0 \le m < M, 0 \le y < E, 0 \le x < F,$$

$$E = (H - R + U)/U, F = (W - S + U)/U.$$

Shape Parameter	Description
N	fmap batch size
M	# of filters / # of output fmap channels
C	# of input fmap/filter channels
H/W	input fmap height/width
R/S	filter height/width
E/F	output fmap height/width
U	convolution stride

Convolutional Layer

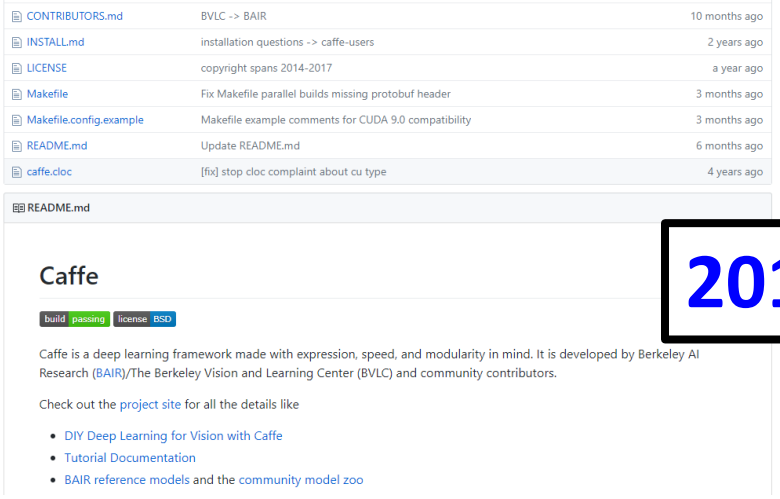
Naïve 7-layer for-loop implementation:

```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++) {
                            O[n][m][x][y] += I[n][k][Ux+i][Uy+j] ×
                            W[m][k][i][j];
                        }
                    }
                }
                O[n][m][x][y] =
                Activation(O[n][m][x][y]);
            }
        }
    }
}
```

for each output fmap value

convolve a window and apply activation

The Story of NVidia



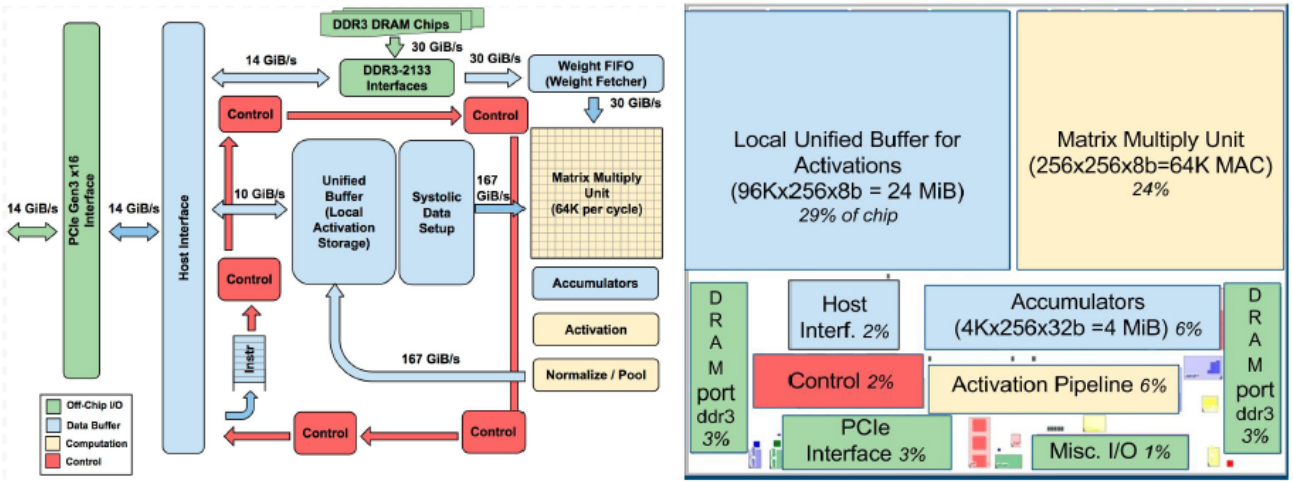
2014

Three Ways to Accelerate

1. GPU (CPU????)
2. ASIC application specific integrated circuit
3. FPGA

100K per year * 200 staffs * 3 years = $60M

ASIC: Google TPU



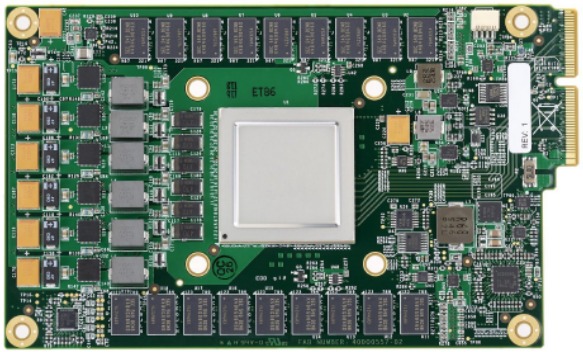


Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

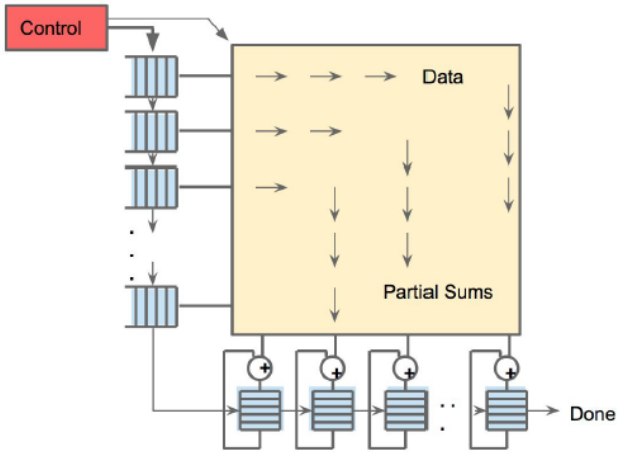


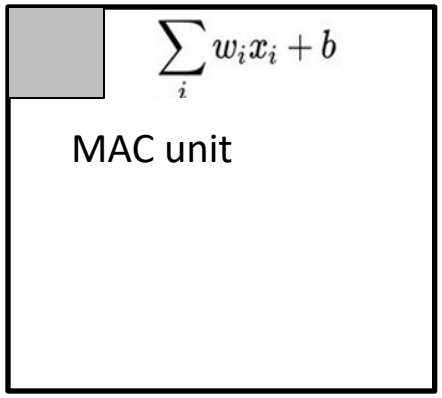
Figure 4. Systolic data flow of the Matrix Multiply Unit. Software has the illusion that each 256B input is read at once, and they instantly update one location of each of 256 accumulator RAMs.

ASIC: Google TPU

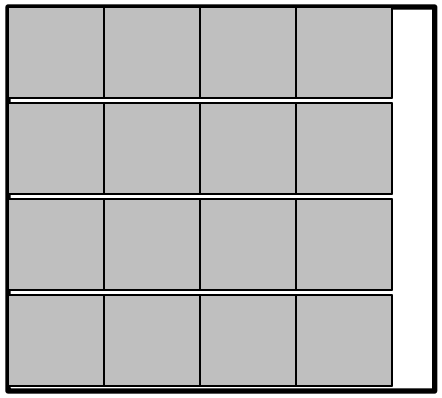
Name	LOC	Layers					Nonlinear function	Weights	TPU Ops / Weight Byte	TPU Batch Size	% of Deployed TPUs in July 2016
		FC	Conv	Vector	Pool	Total					
MLP0	100	5				5	ReLU	20M	200	200	61%
MLP1	1000	4				4	ReLU	5M	168	168	
LSTM0	1000	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1500	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1000		16			16	ReLU	8M	2888	8	5%
CNN1	1000	4	72		13	89	ReLU	100M	1750	32	

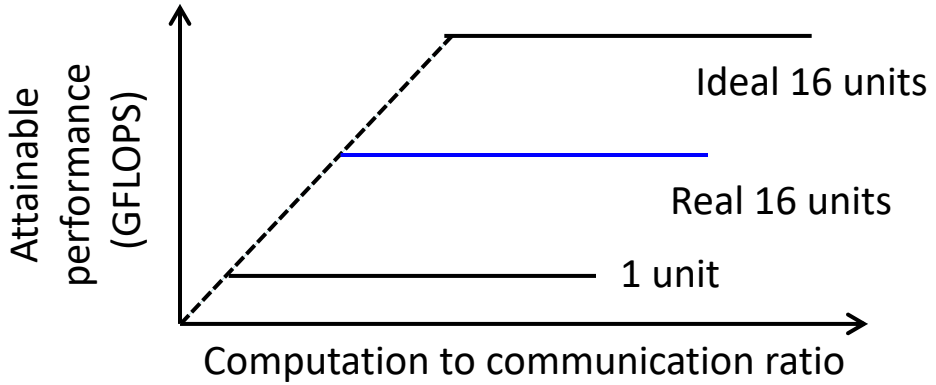
Table 1. Six NN applications (two per NN type) that represent 95% of the TPU's workload. The columns are the NN name; the number of lines of code; the types and number of layers in the NN (FC is fully connected, Conv is convolution, Vector is self-explanatory, Pool is pooling, which does nonlinear downsizing on the TPU; and TPU application popularity in July 2016. One DNN is RankBrain [Cla15]; one LSTM is a subset of GNM Translate [Wu16]; one CNN is Inception; and the other CNN is DeepMind AlphaGo [Sil16][Jou15].

FPGA Implementation



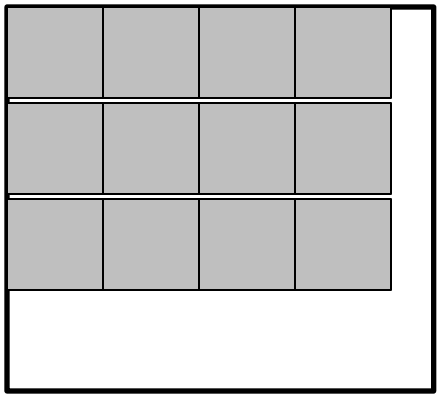
FPGA

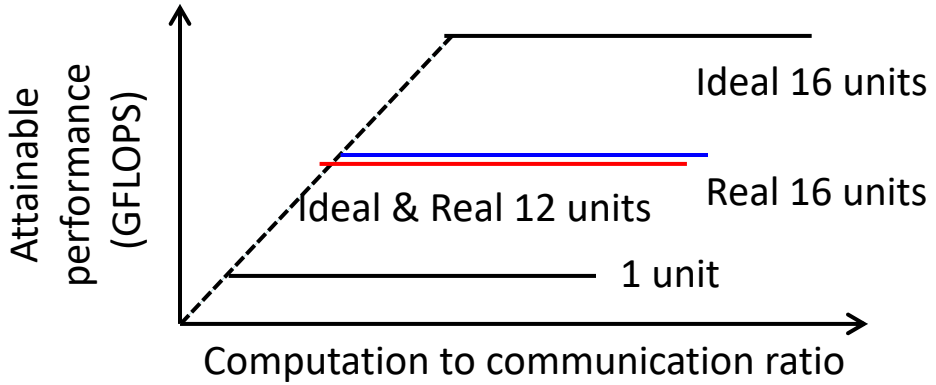




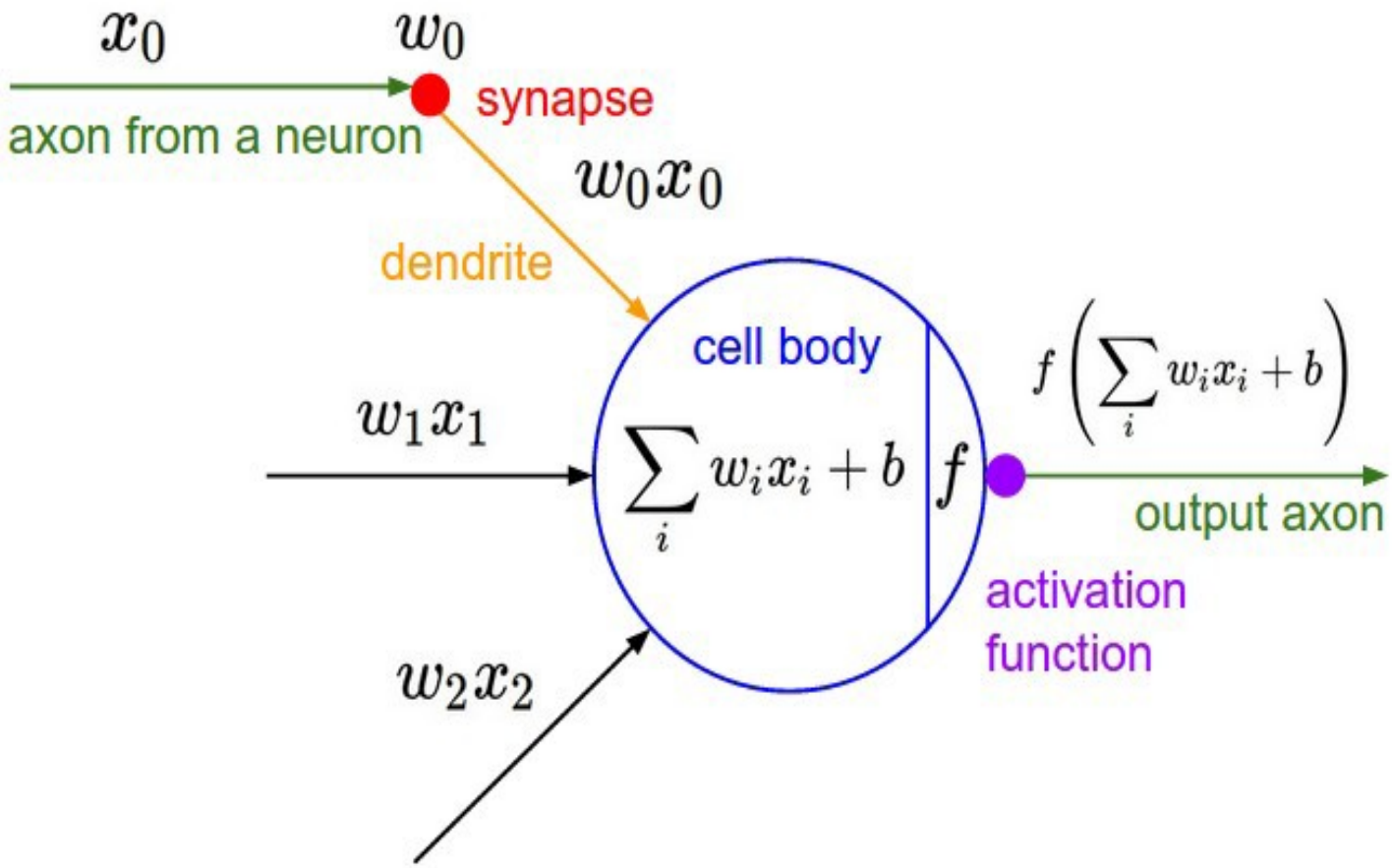

```
//on-chip data computation
for(i=0; i<K; i++) {
 for(j=0; j<K; j++) {
  for(trr=row; trr<min(row+Tr,R); trr++){
   for(tcc=col; tcc<min(col+Tc,C); tcc++){
    for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
     for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
    L:  output_fm[too][trr][tcc] +=
          weights[too][tii][i][j]*
          input_fm[tii][S*trr+i][S*tcc+j];
} } } } } }
```

FPGA Implementation





Error Tolerance of CNNs



Trade-off between
computing efficiency and **CNN accuracy**

Accelerator Type

$$\sum_i w_i x_i + b$$

Reducing the accuracy of both W and X

Reducing the accuracy of W

Outline of Hardware Solutions

1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

Outline of Hardware Solutions

1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

Limited Numerical Precision

32-bit float point =>
Naively implement 16-bit fixed-point convolution engines.

$$\sum_i w_i x_i + b$$

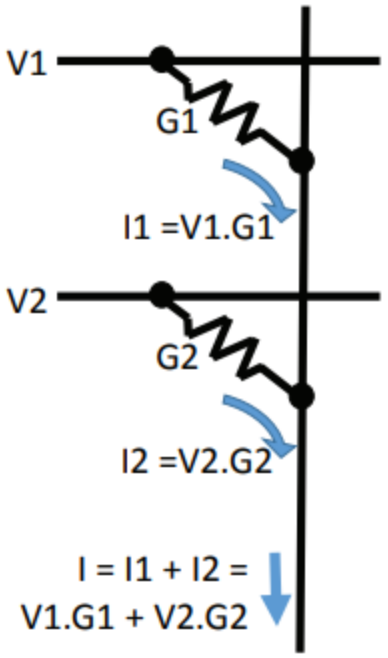
S. Gupta, et al., "Deep Learning with Limited Numerical Precision," CoRR, abs/1502.02551, 2015.

Outline of Hardware Solutions

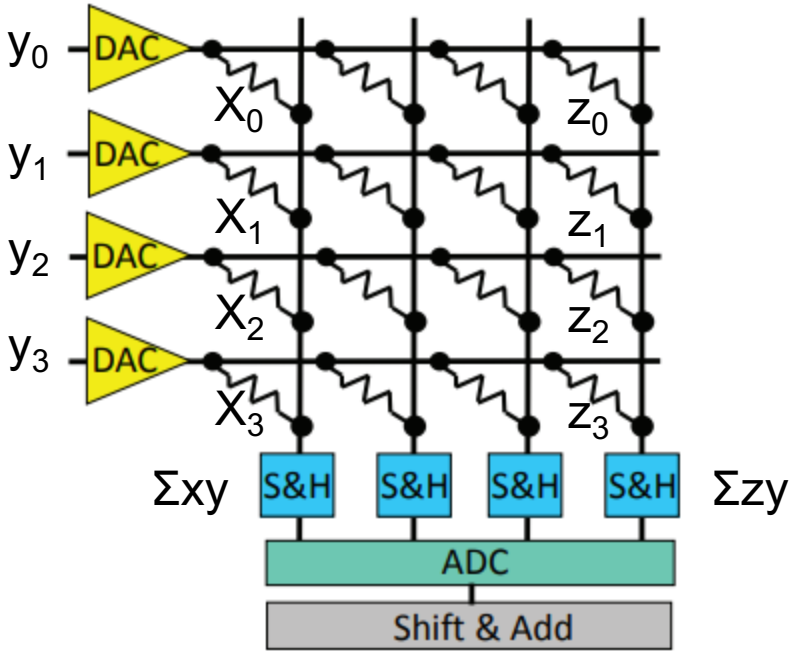
1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

ReRAM Convolution Engine

I=V/R
G=1/R



(a) Multiply-Accumulate operation



(b) Vector-Matrix Multiplier

$$\sum_i w_i x_i + b$$

Outline of Hardware Solutions

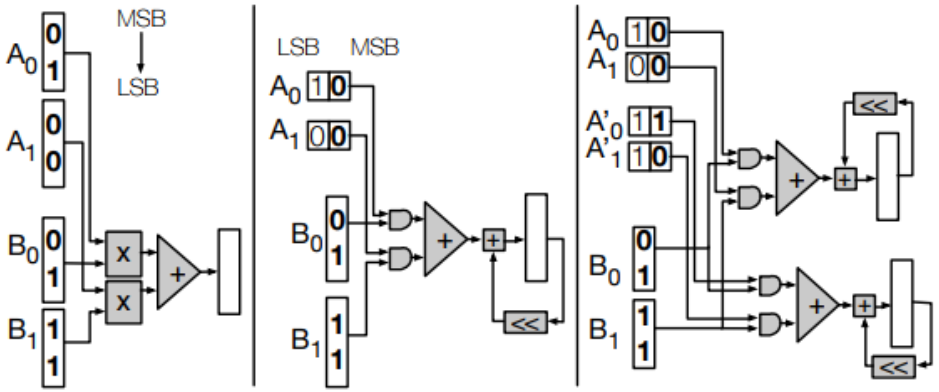
1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

Stripes

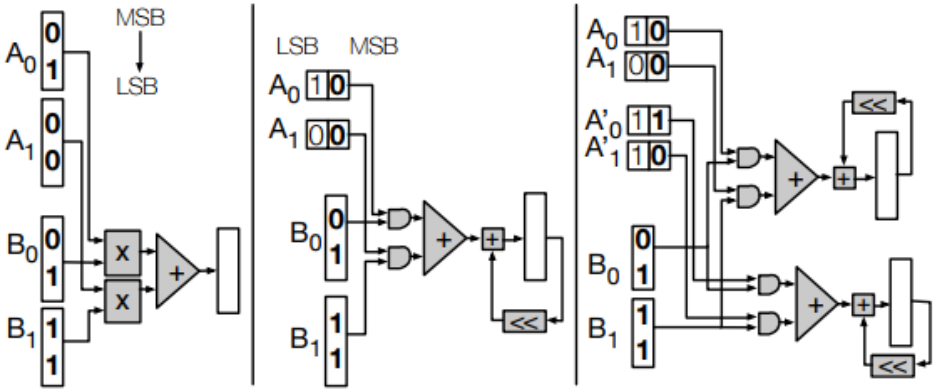
Reducing the accuracy of both W and X

Float point is 32-bit, but 16-bit fixed point is good for CNN

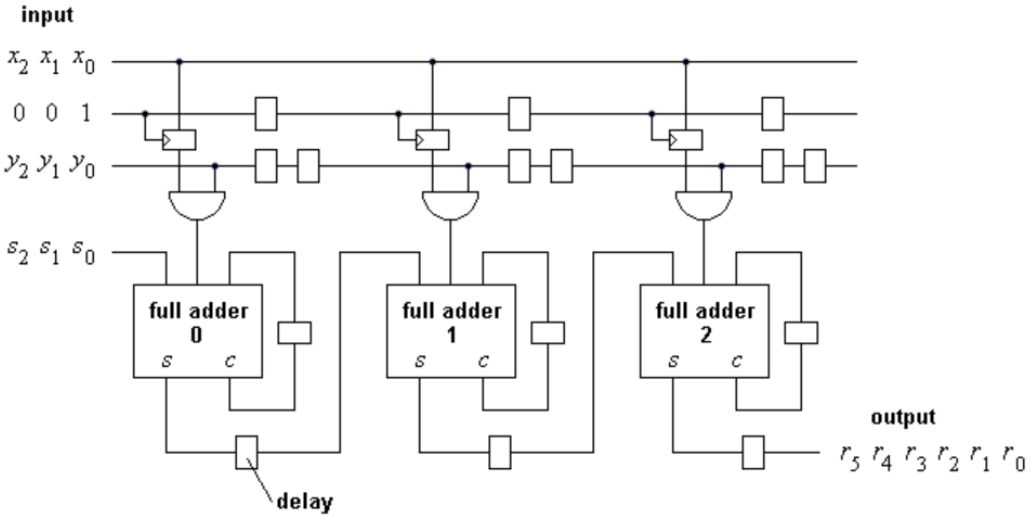
Network	Relative Accuracy 100% Per Layer Neuron Precision in Bits	100% Ideal Speedup	99% Per Layer Neuron Precision in Bits	99% Ideal Speedup
LeNet	3-3	5.33	2-3	7.33
Convnet	4-8-8	2.89	4-5-7	3.53
AlexNet	9-8-5-5-7	2.38	9-7-4-5-7	2.58
NiN	8-8-8-9-7-8-8-9-9-8-8-8	1.91	8-8-7-9-7-8-8-9-9-8-7-8	1.93
GoogLeNet	10-8-10-9-8-10-9-8-9-10-7	1.76	10-8-9-8-8-9-10-8-9-10-8	1.80
VGG_M	7-7-7-8-7	2.23	6-8-7-7-7	2.34
VGG_S	7-8-9-7-9	2.04	7-8-9-7-9	2.04
VGG_19	12-12-12-11-12-10-11-11-13-12-13-13-13-13-13-13	1.35	9-9-9-8-12-10-10-12-13-11-12-13-13-13-13-13	1.57



Stripes



$$
\begin{array}{rrrrrrr}
 & & & s_2 & s_1 & s_0 & \\
 & & y_2 & y_1 & y_0 & \cdot x_0 & \\
 & y_2 & y_1 & y_0 & & \cdot x_1 & \\
y_2 & y_1 & y_0 & & & \cdot x_2 & \\
\hline
r_5 & r_4 & r_3 & r_2 & r_1 & r_0 &
\end{array}
$$



Outline of Hardware Solutions

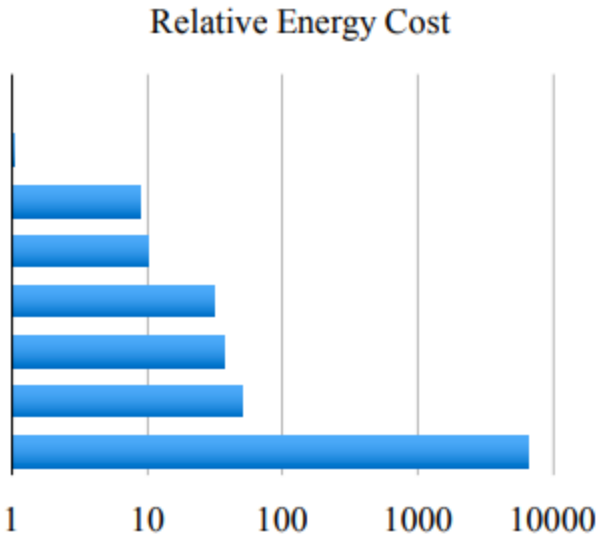
1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

Outline of Hardware Solutions

1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

Pruning

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**



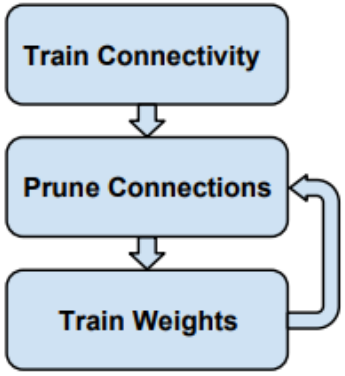


Figure 2: Three-Step Training Pipeline.

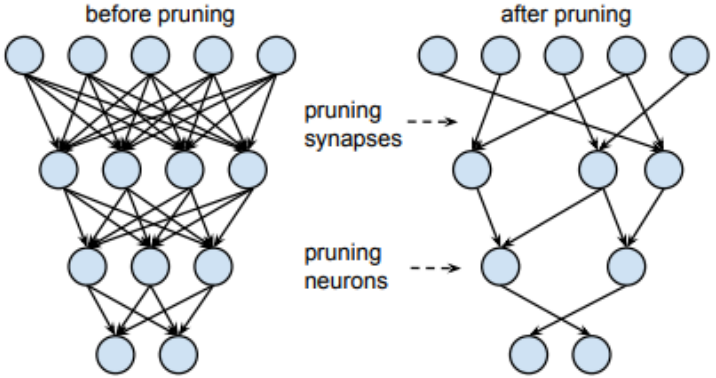


Figure 3: Synapses and neurons before and after pruning.

Outline of Hardware Solutions

1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

Binary, Ternary Weight CNNs

Reducing the accuracy of W

$$\sum_i w_i x_i + b$$ -> ΣW

Simplify W to 1 and -1, or 1, 0 and -1

$$w_b = \begin{cases} +1 & \text{with probability } p = \sigma(w), \\ -1 & \text{with probability } 1 - p. \end{cases}$$

Method	MNIST	CIFAR-10	SVHN
No regularizer	$1.30 \pm 0.04\%$	10.64%	2.44%
BinaryConnect (det.)	$1.29 \pm 0.08\%$	9.90%	2.30%
BinaryConnect (stoch.)	$1.18 \pm 0.04\%$	**8.27%**	2.15%
50% Dropout	$1.01 \pm 0.04\%$		
Maxout Networks [29]	0.94%	11.68%	2.47%
Deep L2-SVM [30]	**0.87%**		
Network in Network [31]		10.41%	2.35%
DropConnect [21]			1.94%
Deeply-Supervised Nets [32]		9.78%	**1.92%**

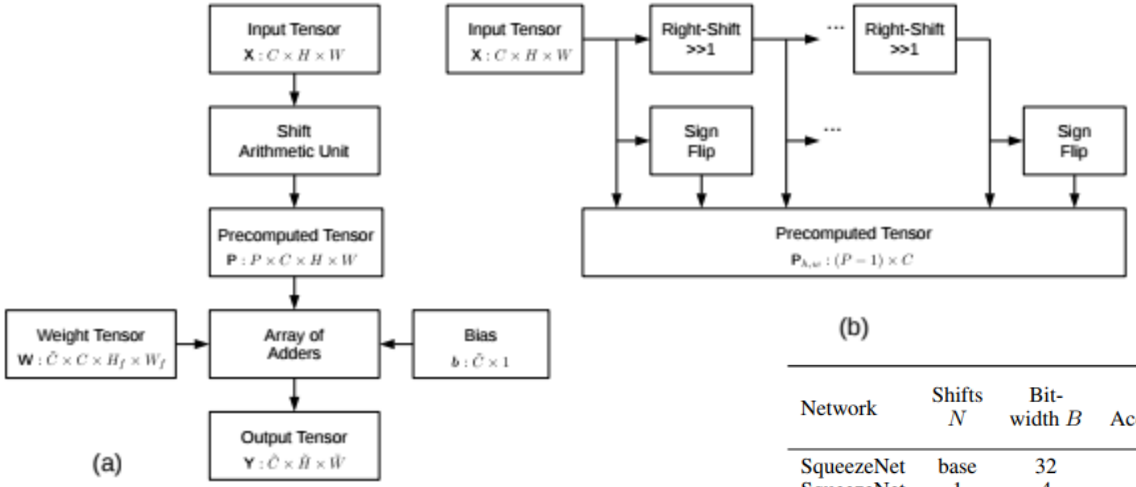
ShiftCNN

Reducing the accuracy of W

$$\sum_i w_i x_i + b$$

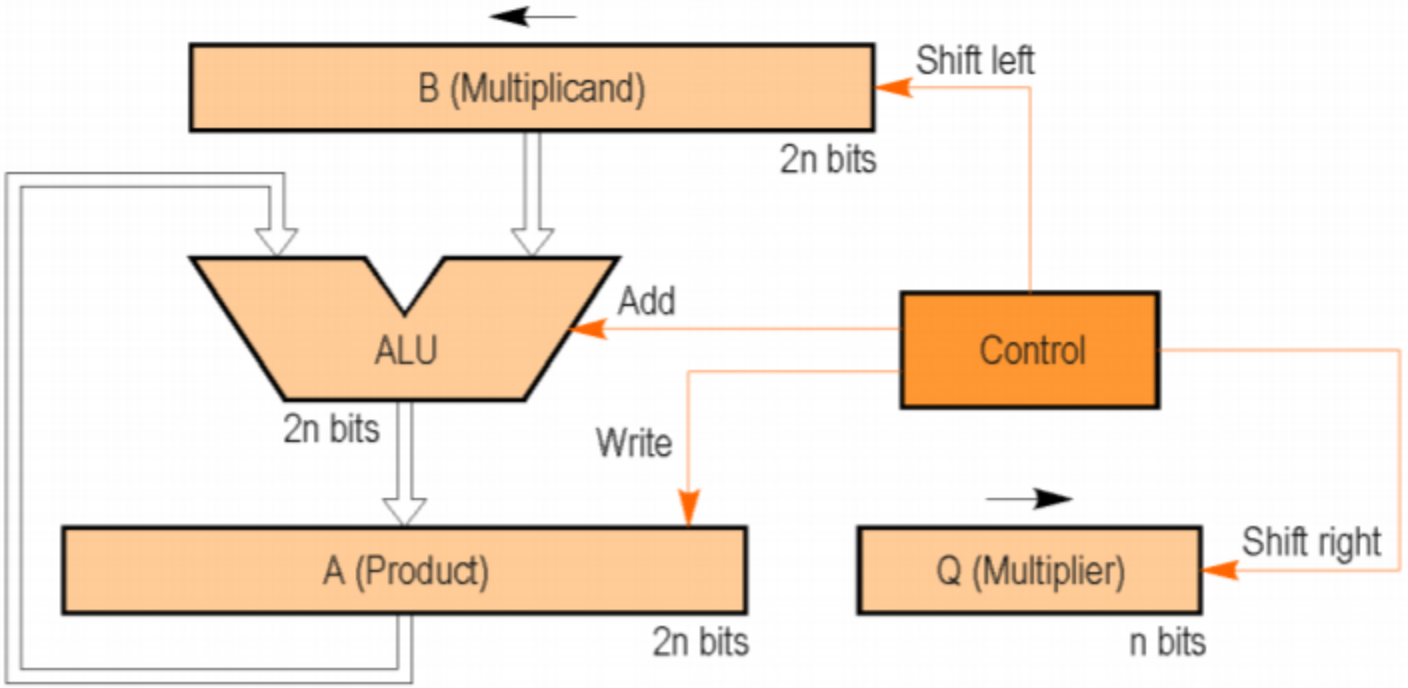
Transform w to 2^n

Simplify W to 2^n values



Network	Shifts N	Bit-width B	Top-1 Accuracy, %	Top-5 Accuracy, %	Decrease in Top-1 Accuracy, %	Decrease in Top-5 Accuracy, %
SqueezeNet	base	32	58.4	81.02		
SqueezeNet	1	4	23.01	45.93	35.39	35.09
SqueezeNet	2	4	57.39	80.31	1.01	0.71
SqueezeNet	3	4	58.39	81.01	0.01	0.01
GoogleNet	base	32	68.93	89.15		
GoogleNet	1	4	57.67	81.79	11.26	7.36
GoogleNet	2	4	68.54	88.86	0.39	0.29
GoogleNet	3	4	68.88	89.06	0.05	0.09
ResNet-18	base	32	64.78	86.13		
ResNet-18	1	4	24.61	47.02	40.17	39.11
ResNet-18	2	4	64.24(61.57)	85.79(84.08)	0.54(3.21)	0.34(2.05)
ResNet-18	3	4	64.75(64.69)	86.01(85.97)	0.03(0.09)	0.12(0.16)
ResNet-50	base	32	72.87	91.12		
ResNet-50	1	4	54.38	78.57	18.49	12.55
ResNet-50	2	4	72.20(70.38)	90.71(89.48)	0.67(2.49)	0.41(1.64)
ResNet-50	3	4	72.58(72.56)	90.97(90.96)	0.29(0.31)	0.15(0.16)

ShiftCNN



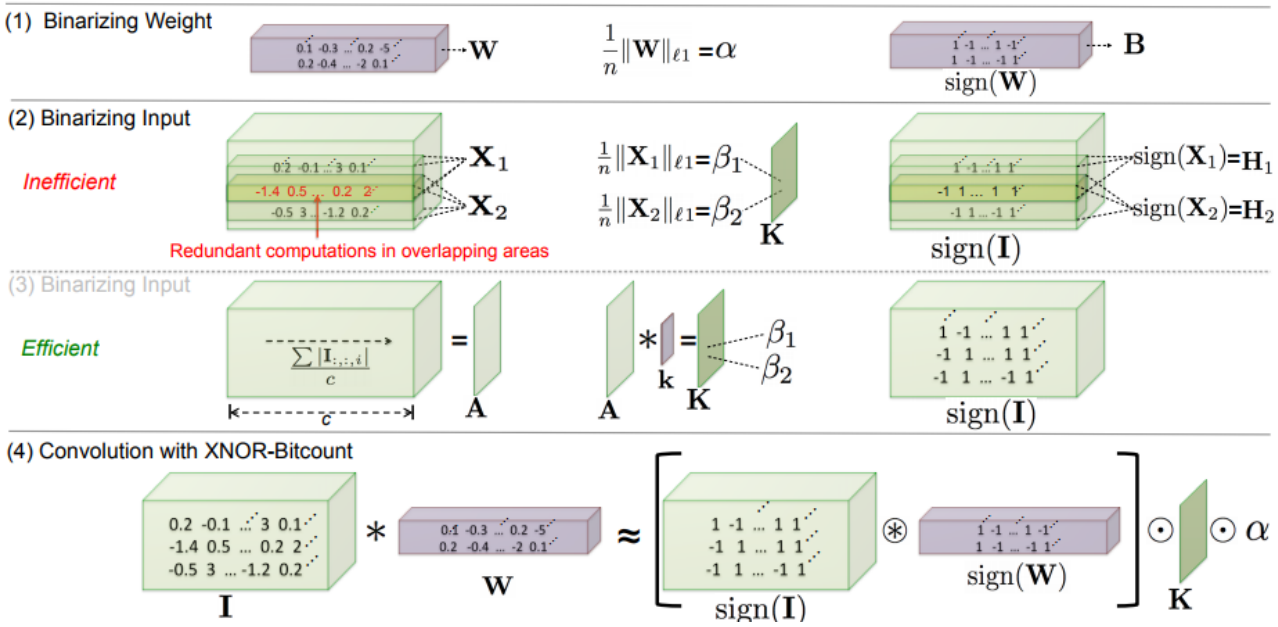
2^n -> shift n bits

Outline of Hardware Solutions

1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

XNOR-Net

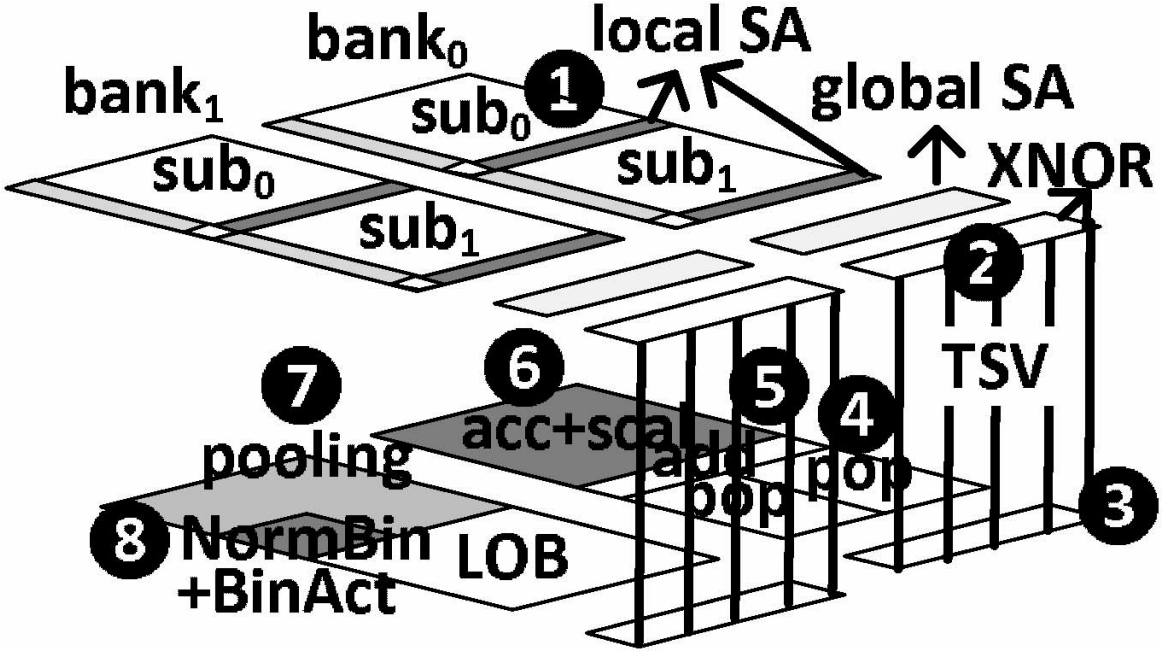
Reducing the accuracy of both W and X



XNOR, **Popcounting, how many 1s in 10111111 ???**

Classification Accuracy(%)									
Binary-Weight				Binary-Input-Binary-Weight				Full-Precision	
BWN		BC[11]		XNOR-Net		BNN[11]		AlexNet[1]	
Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
56.8	**79.4**	35.4	61.0	**44.2**	**69.2**	27.9	50.42	56.6	80.2

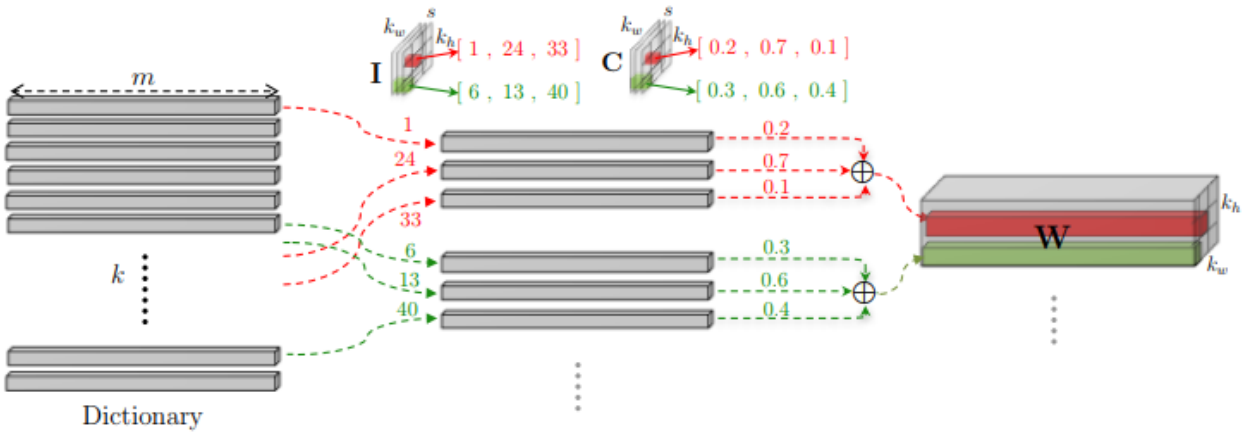
XNOR-POP


bank0
bank1
sub0
sub1
sub0
sub1
local SA
global SA
XNOR
TSV
1
2
3
4
5
6
7
8
pooling
acc+scal
add
pop
pop
NormBin
+BinAct
LOB

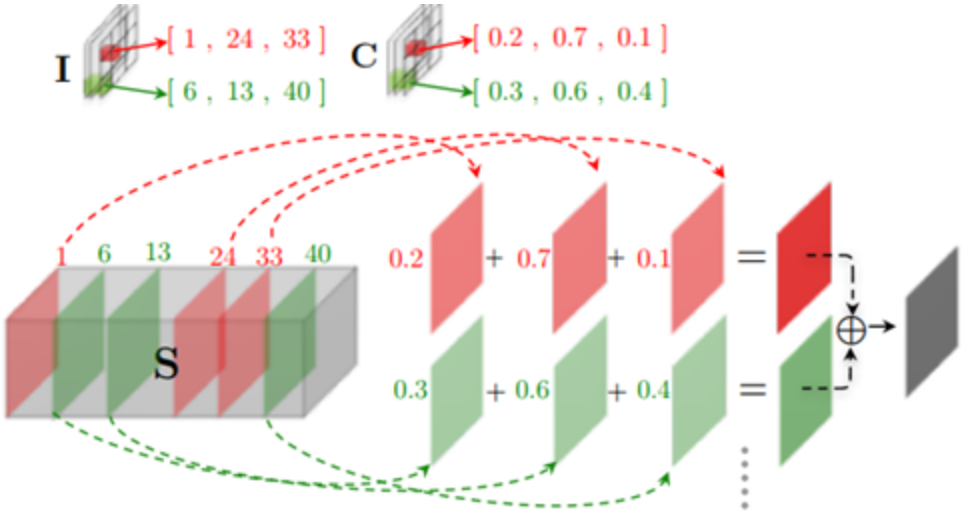
Outline of Hardware Solutions

1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

LCNN: Lookup Table-based CNN



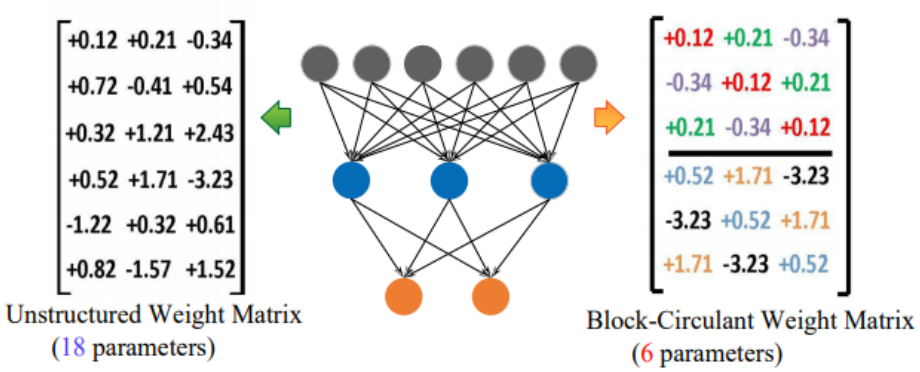
S is the convolution result between input and dictionary



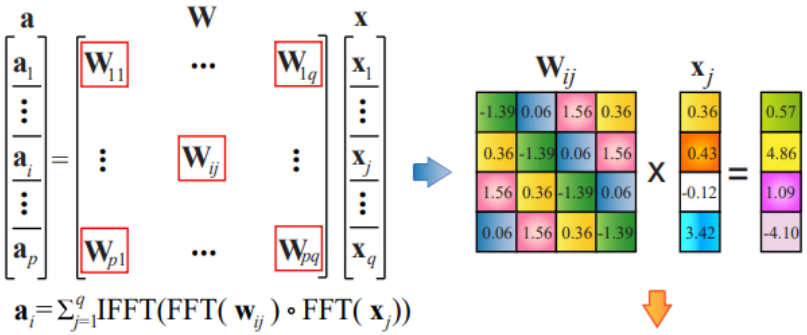
Outline of Hardware Solutions

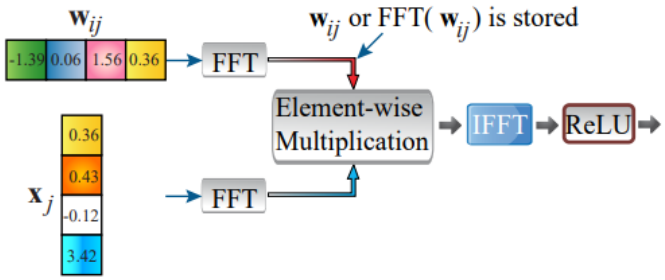
1. Limited numerical precision
2. Limited numerical precision hardware
3. Different Layers require different numerical precisions
4. Pruning: remove small weights
5. Extremely limited numerical precision on weights
   - Binary, ternary and 2^n
6. Extremely limited numerical precision on weights and inputs
7. Dictionary for weights
8. Block-Circulant matrix for weights

Block-Circulant Weight Matrices



storage complexity is reduced from $O(n^2)$ to $O(n)$.





computational complexity is reduced from $O(n^2)$ to $O(n\log n)$

Fast Fourier transform

Hardware Design for Deep Learning

Lei Jiang

Jiang60@iu.edu

Indiana University Bloomington

Part of slide is developed based on the Tutorial on Hardware Architectures for Deep Neural Networks, Joel Emer, Vivienne Sze, Yu-Hsin Chen

Hardware Design for Deep Learning

Lei Jiang

Jiang60@iu.edu

Indiana University Bloomington

Part of slide is developed based on the Tutorial on Hardware Architectures for Deep Neural Networks, Joel Emer, Vivienne Sze, Yu-Hsin Chen

Hardware Design for Deep Learning

Lei Jiang

Jiang60@iu.edu

Indiana University Bloomington

Part of slide is developed based on the Tutorial on Hardware Architectures for Deep Neural Networks, Joel Emer, Vivienne Sze, Yu-Hsin Chen

Hardware Design for Deep Learning

Lei Jiang

Jiang60@iu.edu

Indiana University Bloomington

Part of slide is developed based on the Tutorial on Hardware Architectures for Deep Neural Networks, Joel Emer, Vivienne Sze, Yu-Hsin Chen