Tutorial on Optimizing Machine Learning for Hardware

Prof. Warren Gross and Prof. Brett H. Meyer

Department of Electrical and Computer Engineering

McGill University

Montreal, Canada

EPEPS 2019

Montreal, QC

October 6, 2019

Acknowledgements

- Source materials from http://cs231n.stanford.edu/, http://www.rle.mit.edu/eems/publications/tutorials, and other sources

Arash Ardakani

Loren Lugosch

This tutorial

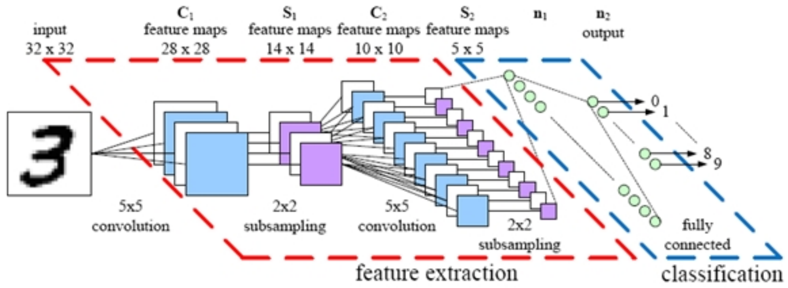
Part 1

- Introduction to neural networks
  - Convolutional neural networks
- Techniques for optimizing neural networks for hardware
  - Data flow
  - Model compression (quantization and pruning)

Part 2

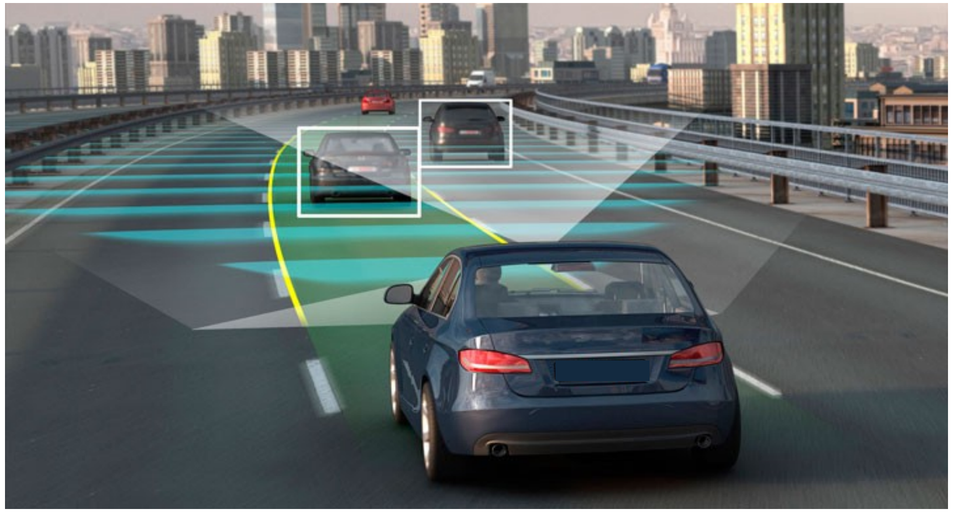
- Finding a good deep neural network model for a given AI processor
  - Design space exploration

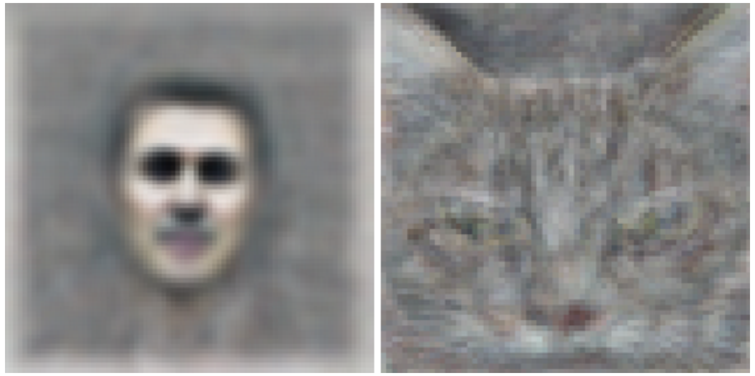
Deep Neural Networks



DNN Timeline

- 1940s - Neural networks were proposed
- 1960s - Deep neural networks were proposed
- 1989 - Neural net for recognizing digits (LeNet)
- 1990s - Hardware for shallow neural nets (Intel ETANN)
- 2011 - Breakthrough DNN-based speech recognition (Microsoft)
- 2012 - DNNs for vision start supplanting hand-crafted approaches (AlexNet)
- 2014+ - Rise of DNN accelerator research (Neuflow, DianNao...)

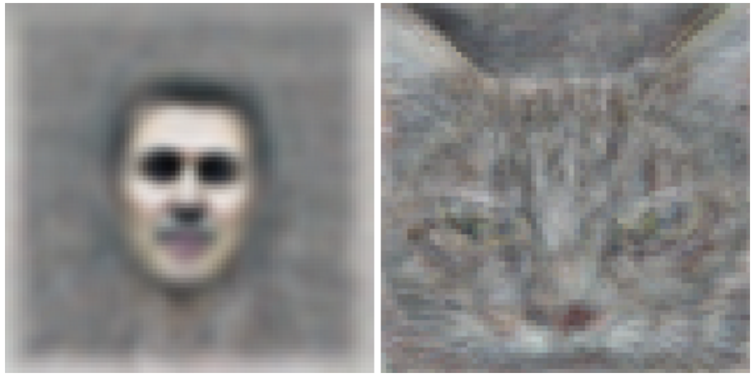




Google

Le at al, ICML 2012

Recognize human and cat faces in video
16,000 cores
100 kW



Le at al, ICML 2012

Recognize human and cat faces in video
16,000 cores
100 kW

What about machine learning at the edge?

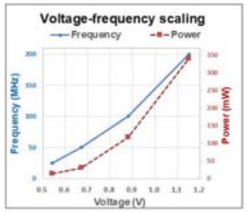
Privacy / regulation
Low-power
Latency

ISSSC 2019 chips – Deep Learning Processors

7.5 A 65nm 0.39-to-140.3TOPS/W 1-to-12b Unified Neural-Network Processor Using Block-Circulant-Enabled Transpose-Domain Acceleration with 8.1× Higher TOPS/mm² and 6T HBST-TRAM-Based 2D Data-Reuse Architecture

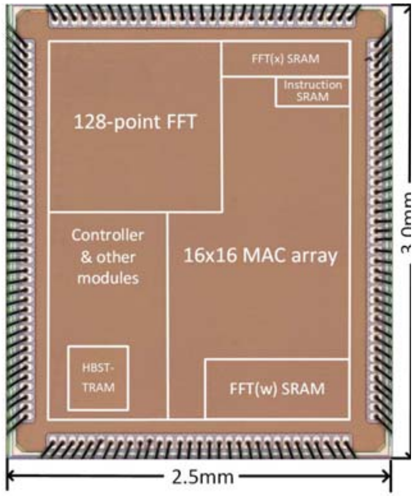
Jinshan Yue[1], Ruoyang Liu[1], Wenyu Sun[1], Zhe Yuan[1], Zhibo Wang[1], Yung-Ning Tu[2], Yi-Ju Chen[2], Ao Ren[3], Yanzhi Wang[3], Meng-Fan Chang[2], Xueqing Li[1], Huazhong Yang[1], Yongpan Liu[1]

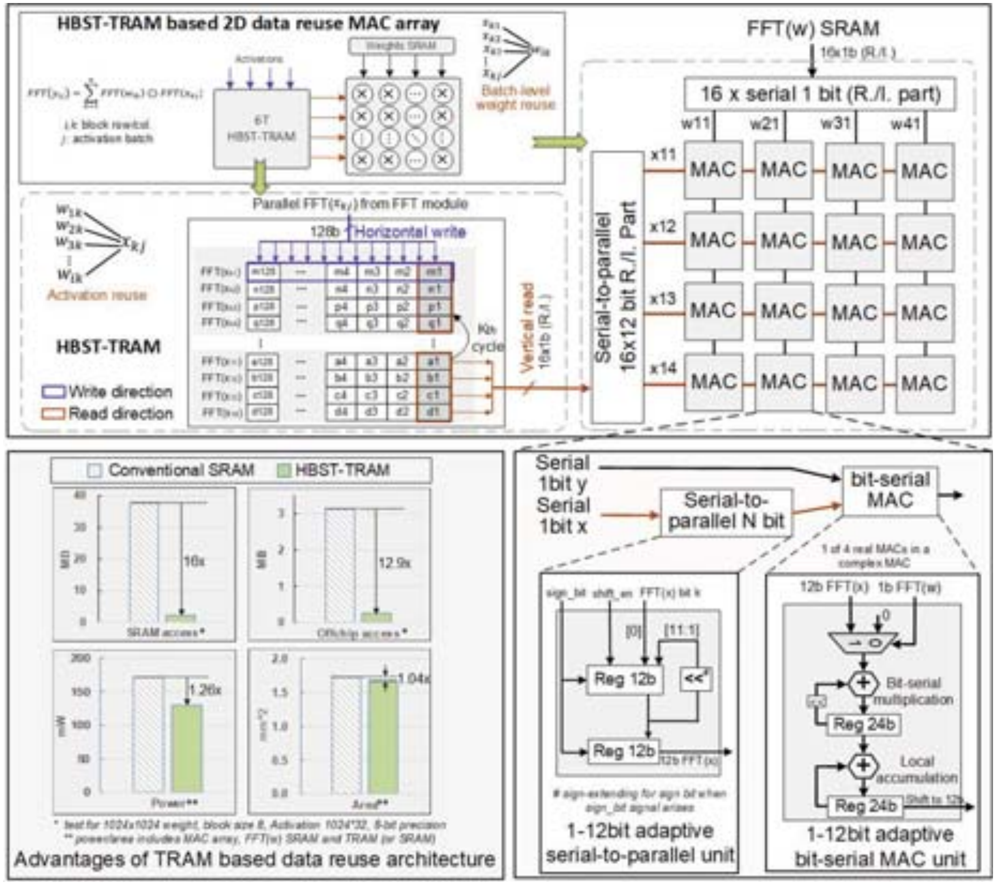
[1]Tsinghua University, Beijing, China
[2]National Tsing Hua University, Hsinchu, Taiwan
[3]Northeastern University, Boston, MA



Technology	TSMC 65nm GP
Chip Area	3.0mmx2.5mm
Core Area	2.46mmx1.96mm
Support NN type	CNN+FC+RNN
Weight Bit-precision	1-12 bit
Activition Bit-precision	1-12 bit
Total SRAM	100kB
HBST-TRAM	4kB
Supply Voltage	0.54-1.15V
Frequency	25-200MHz
Power	13.3-339.2mW
Performance	0.13-14.9TOPS
Energy Efficiency 1)	0.39-140.3TOPS/W

1) minimum @1.15V,200MHz,12bit, block size=8. Maximum @0.54V,25MHz ,3bit FFT, 1bit MAC, block size=128, on AlexNet FC layer 2.

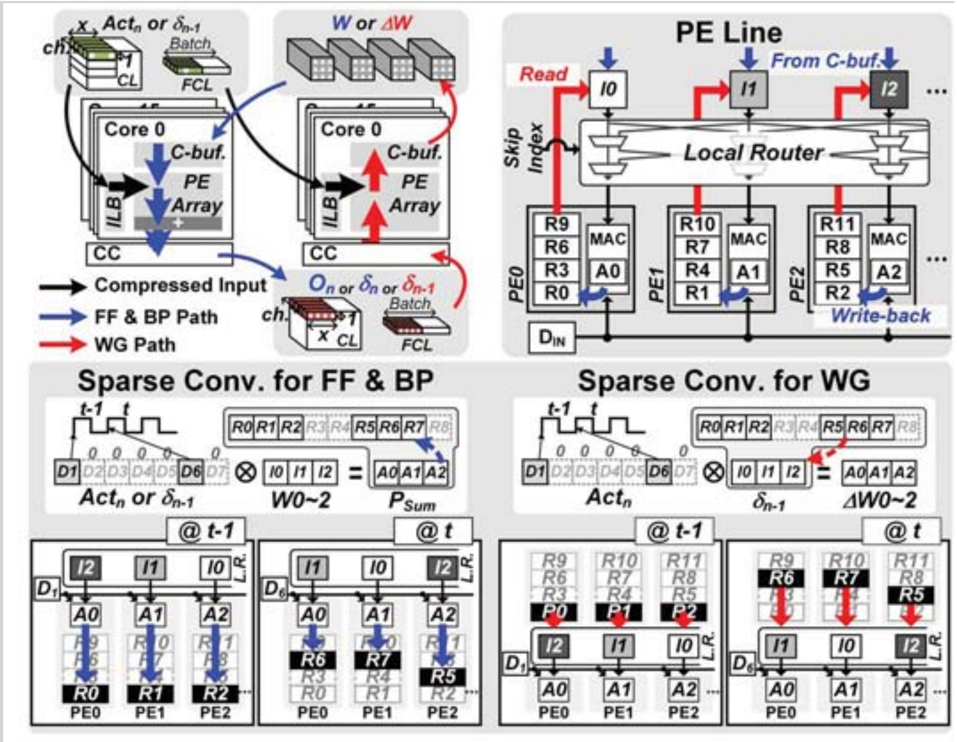


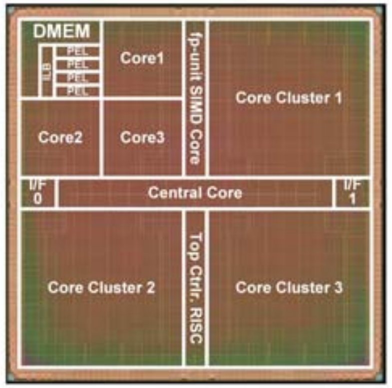


7.7 LNPU: A 25.3TFLOPS/W Sparse Deep-Neural-Network Learning Processor with Fine-Grained Mixed Precision of FP8-FP16

Jinsu Lee, Juhyoung Lee, Donghyeon Han, Jinmook Lee, Gwangtae Park, Hoi-Jun Yoo

KAIST, Daejeon, Korea





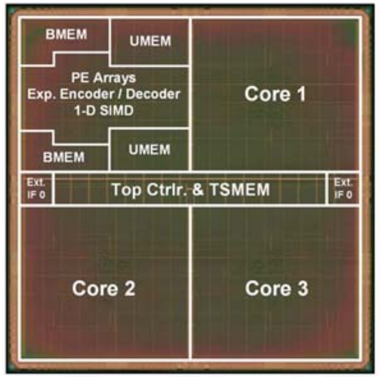
	Specifications		
Technology	65nm 1P8M CMOS		
Die Area	4mm × 4mm (16mm^2)		
SRAM	372 KB		
Supply Voltage	0.78V ~ 1.1V		
Frequency	~ 200MHz		
Data Type	FP8, FP16		
Power Consumption (mW)	43.1mW @ 0.78V, 50MHz		
	367mW @ 1.1V, 200MHz		
Power Efficiency [TFLOPS/W]		FP16	FP8
	50MHz, @0.78V	1.74 -15.6* TFLOPS/W	3.48-25.3* TFLOPS/W
	200MHz, @1.1V	0.817-7.32* TFLOPS/W	1.63-11.9* TFLOPS/W

*Effective TFLOPS/W with 90% Input Sparsity

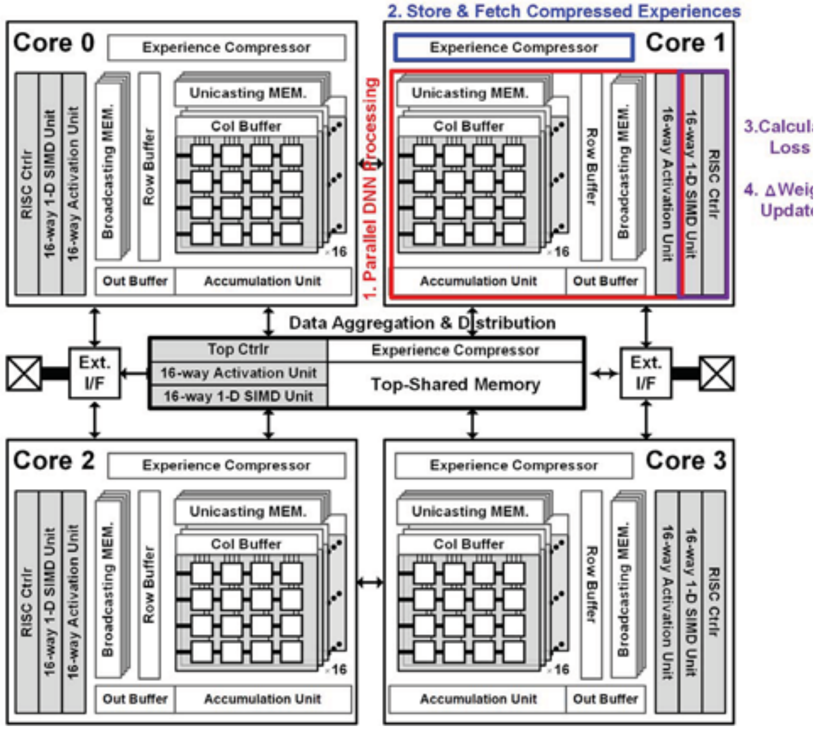
7.4 A 2.1TFLOPS/W Mobile Deep RL Accelerator with Transposable PE Array and Experience Compression

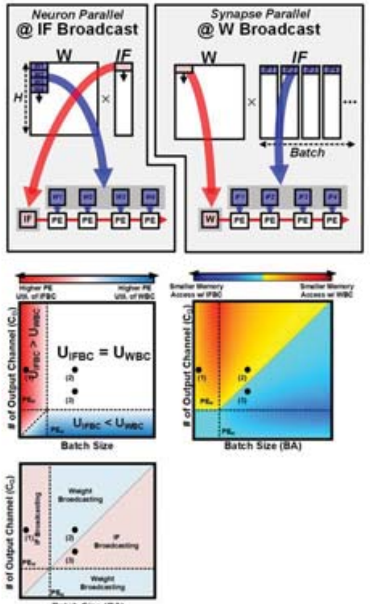
Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sungpill Choi, Youngwoo Kim, Hoi-Jun Yoo

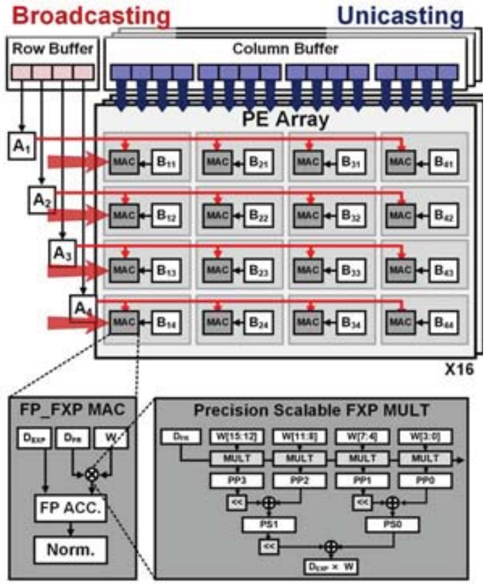
KAIST, Daejeon, Korea



Process	65nm 1P8M Logic CMOS
Area	4mm × 4mm
SRAM	448 KB
Supply	0.67V – 1.1V
Power	196 mW @ 200MHz, 1.1V
	2.4 mW @ 10MHz, 0.67V
Precision	Feature – bfloat16
	Weight – 16/8/4'b FXP
Peak Performance	204 GOPS @ 16b Weight

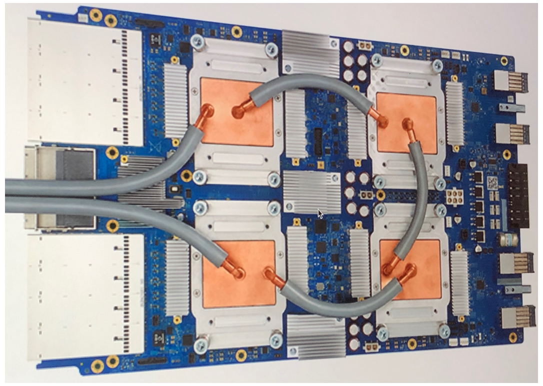






What role does hardware play in deep learning?

NVIDIA Tesla V100



Google TPU 3.0

Speed up training

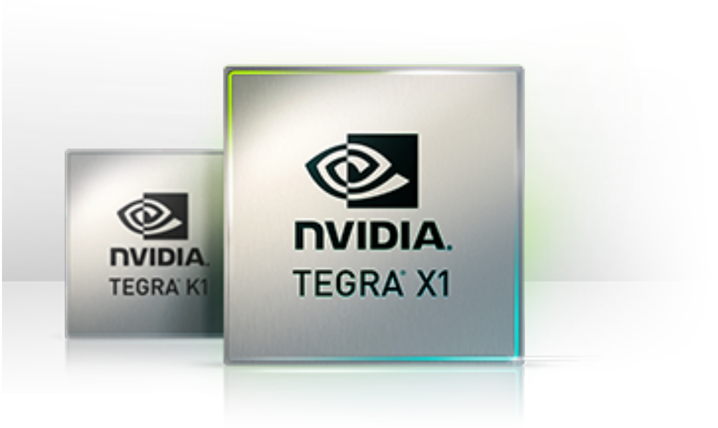
GPUs are optimized to do linear algebra on floating-point data

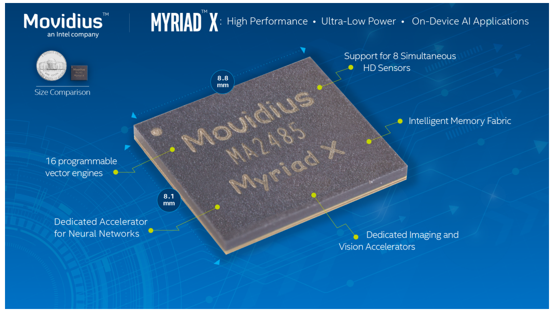
Huge memory bandwidth

Tensor processing unit (TPU)

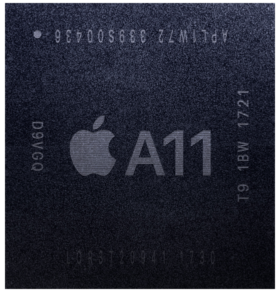
- 8-bit ASIC

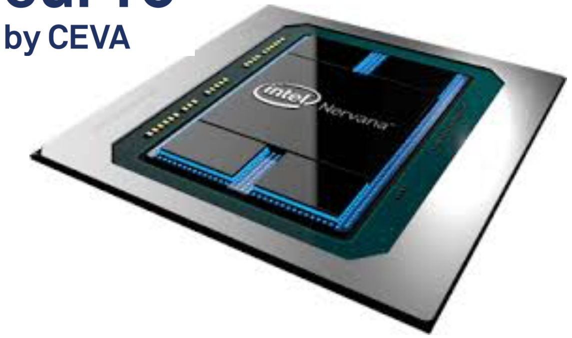
What role does hardware play in deep learning?





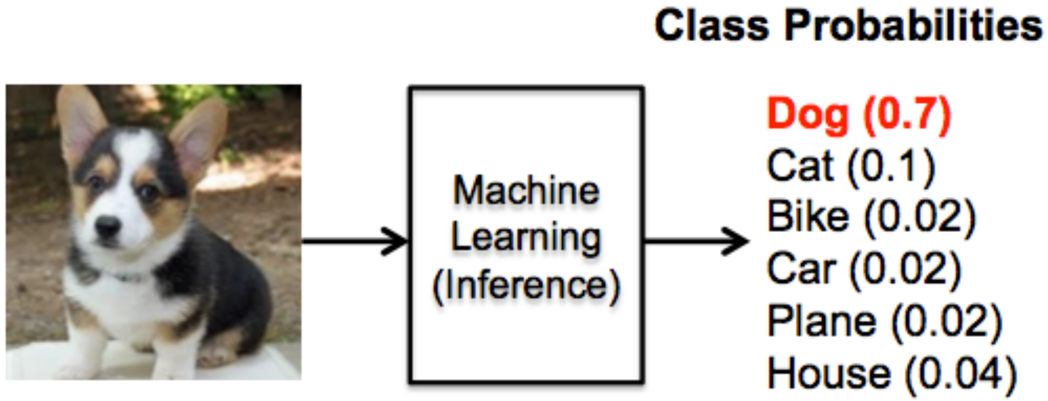
NeuPro™
by CEVA

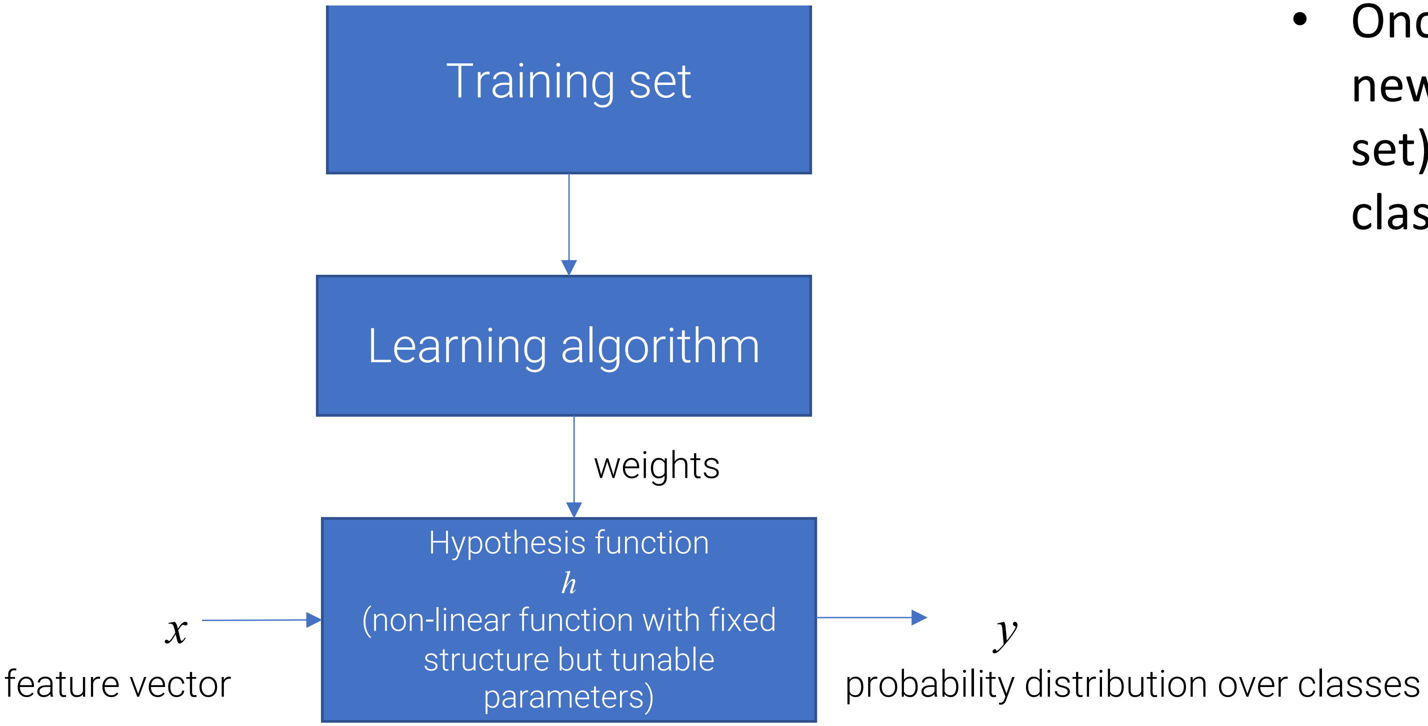




Low-energy inference

Mobile GPUs
Special-purpose ASICs
Microcontrollers (tinyML)
FPGAs

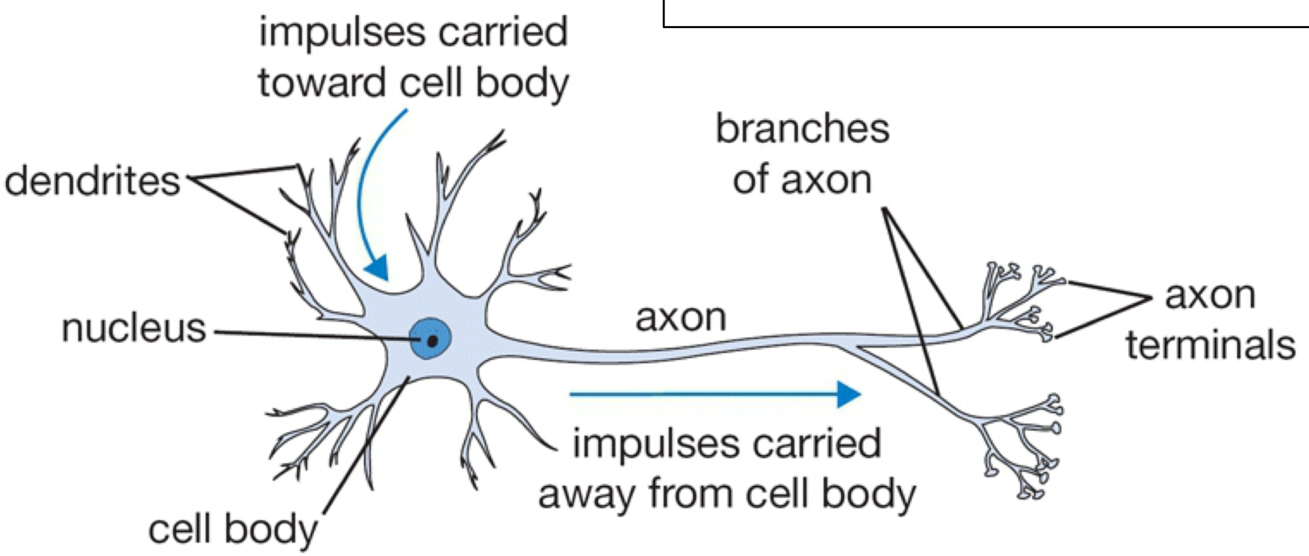


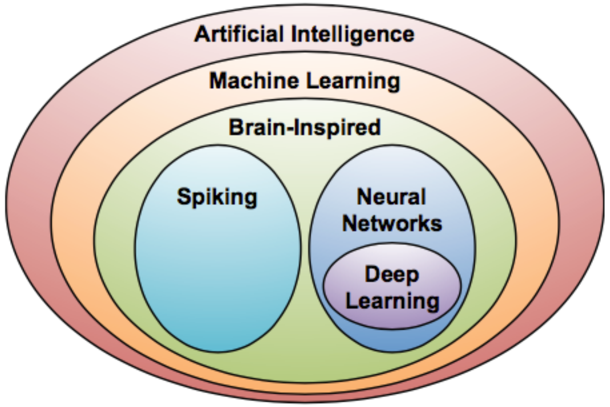


- Let’s look at the problem of classifying an input into one of several classes
- We first show the classifier many examples of input where we already know the class (training set)
    - The classifier “learns” how to classify the elements in the training set
    - Once the training is complete, you can present a new input to the classifier (not from the training set) and it should do a good job at correctly classifying it

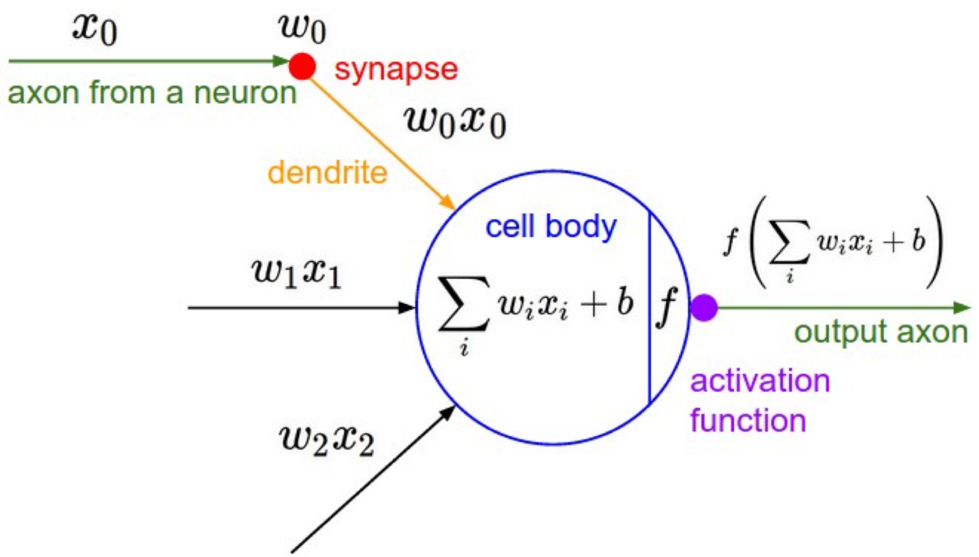
Neurons

Biological neuron



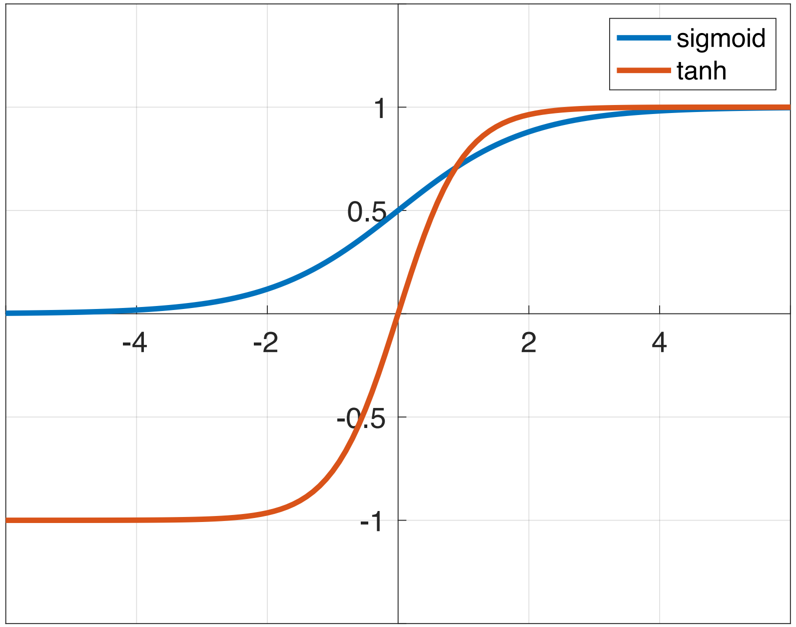


Artificial neuron

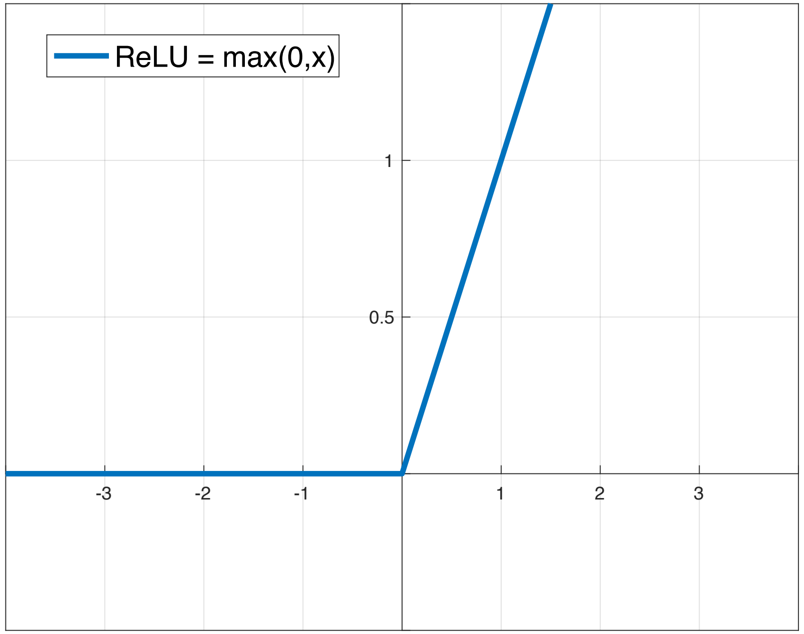


Nonlinear activation functions

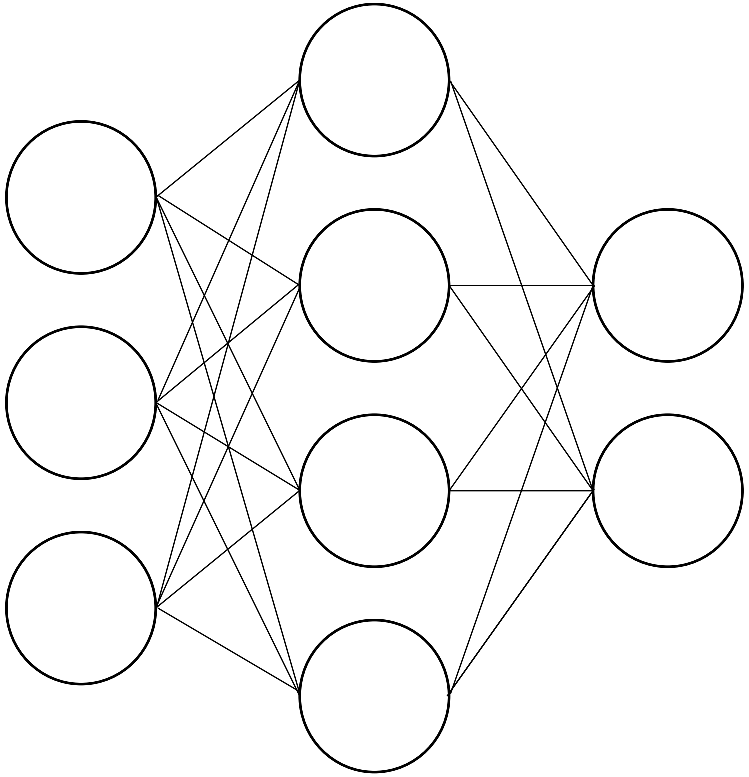
- Sigmoid, tanh()
  - Slow due to exp()
  - “vanishing gradient”



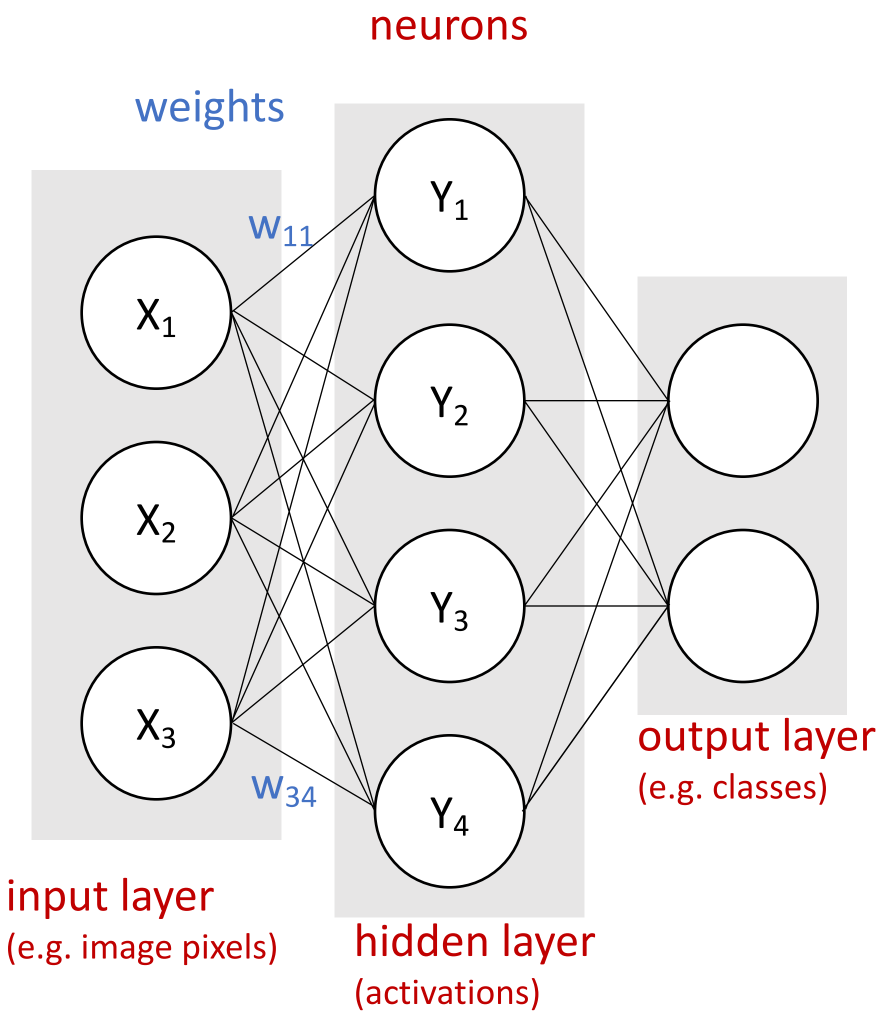
- Rectified linear unit (ReLU)



Fully connected neural networks

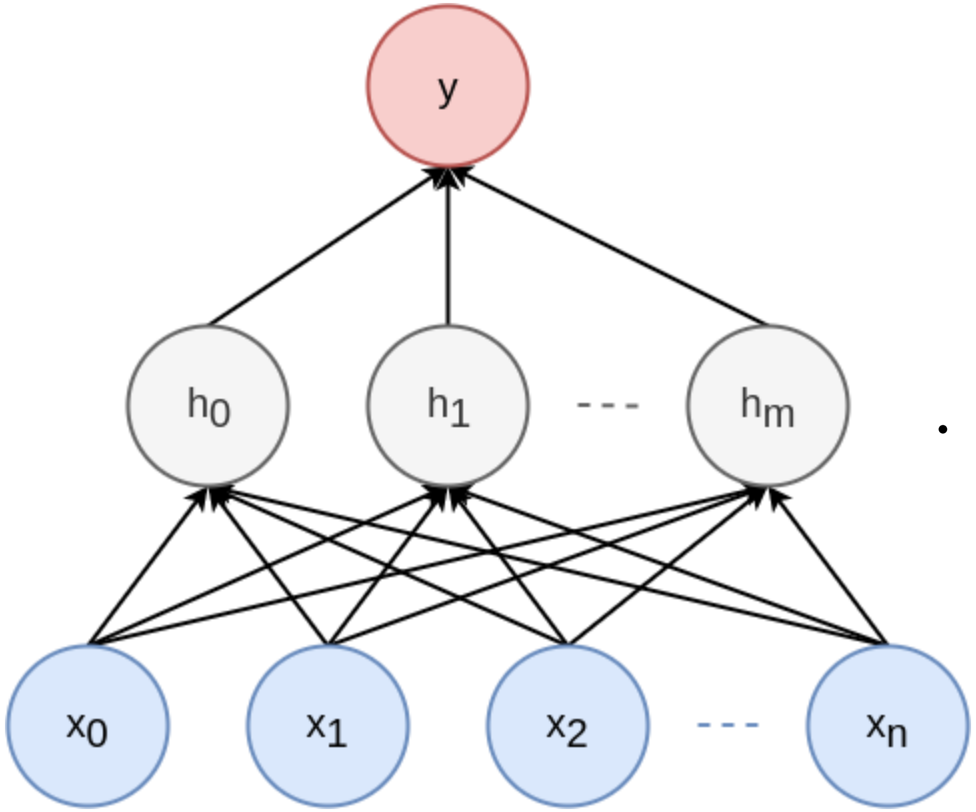


Fully connected neural networks



W. J. Gross and B. H. Meyer, EPEPS 2019

Back Propagation



- Goal: To find w' and w while minimizing the loss function L
- In other words, we want to compute the following equations:

$$w'^{t+1} = w'^{t} - \eta \frac{\partial L}{\partial w'}$$

$$w^{t+1} = w^{t} - \eta \frac{\partial L}{\partial w}$$

- Apply the chain rule:

$$\frac{\partial L}{\partial w'} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial b} \cdot \frac{\partial b}{\partial w'}$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial b} \cdot \frac{\partial b}{\partial h} \cdot \frac{\partial h}{\partial a} \cdot \frac{\partial a}{\partial w}$$

Given input x_i for i = {0, 1, …, n}

↓

Hidden (linear)
$a_j = \sum_{i=0}^{n} w_{ji} x_i$

↓

Hidden (non-linear)
$h_j = \sigma(a_j)$

↓

Output (linear)
$b = \sum_{j=0}^{m} w'_j h_j$

↓

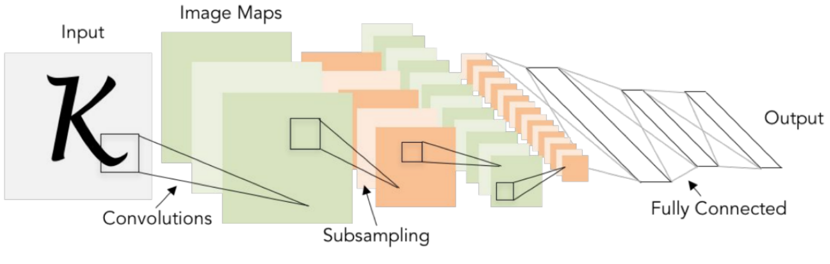
Output (non-linear)
$y = \sigma(b)$

↓

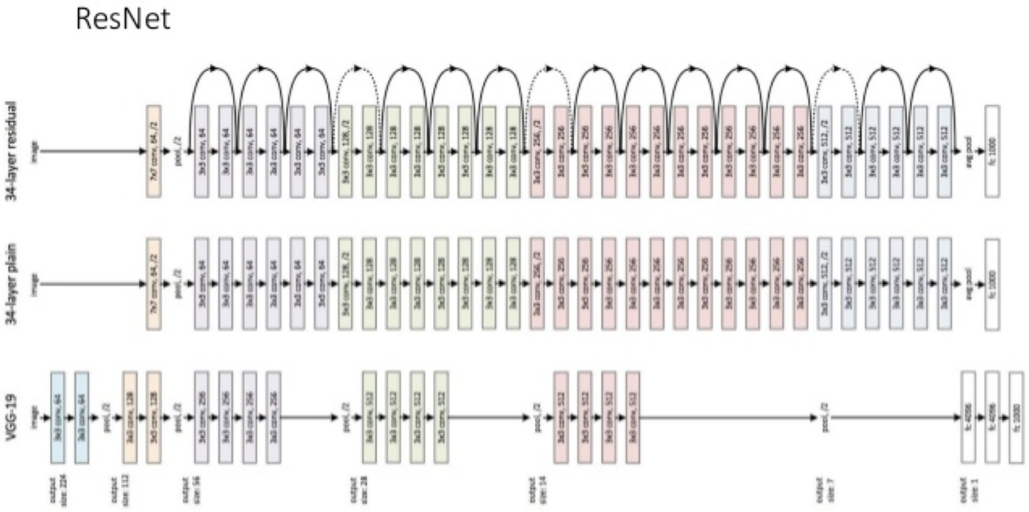
Loss function
$L = \frac{1}{2}(y - t)^2$

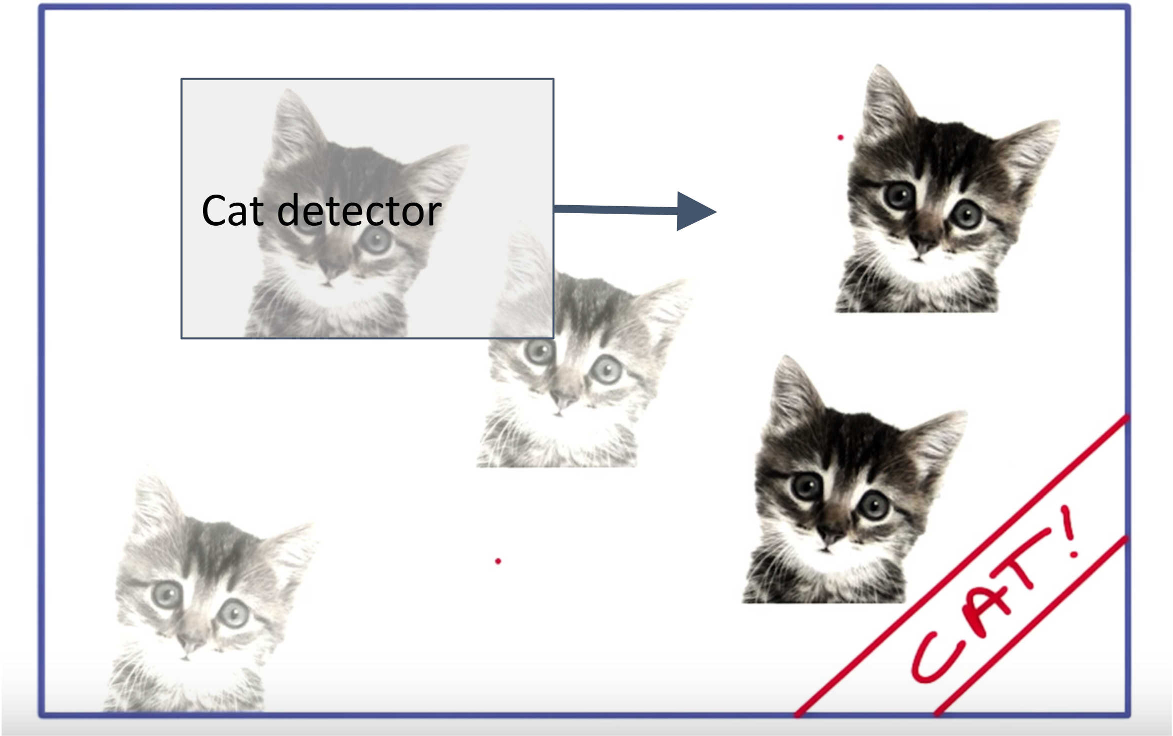
Convolutional Neural Networks

Scan (convolve) neural network over input to detect same feature in different places



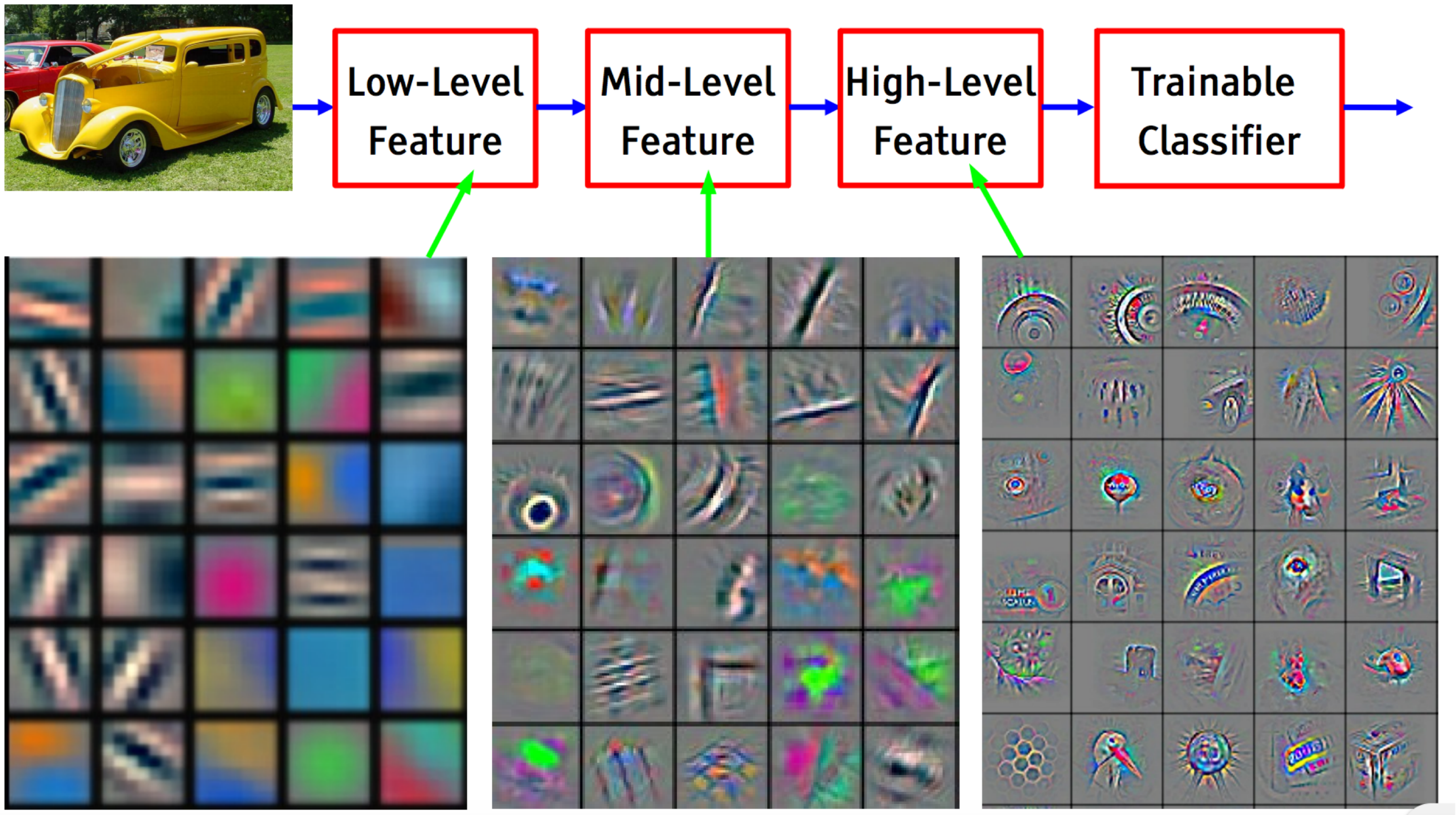
LeNet-5



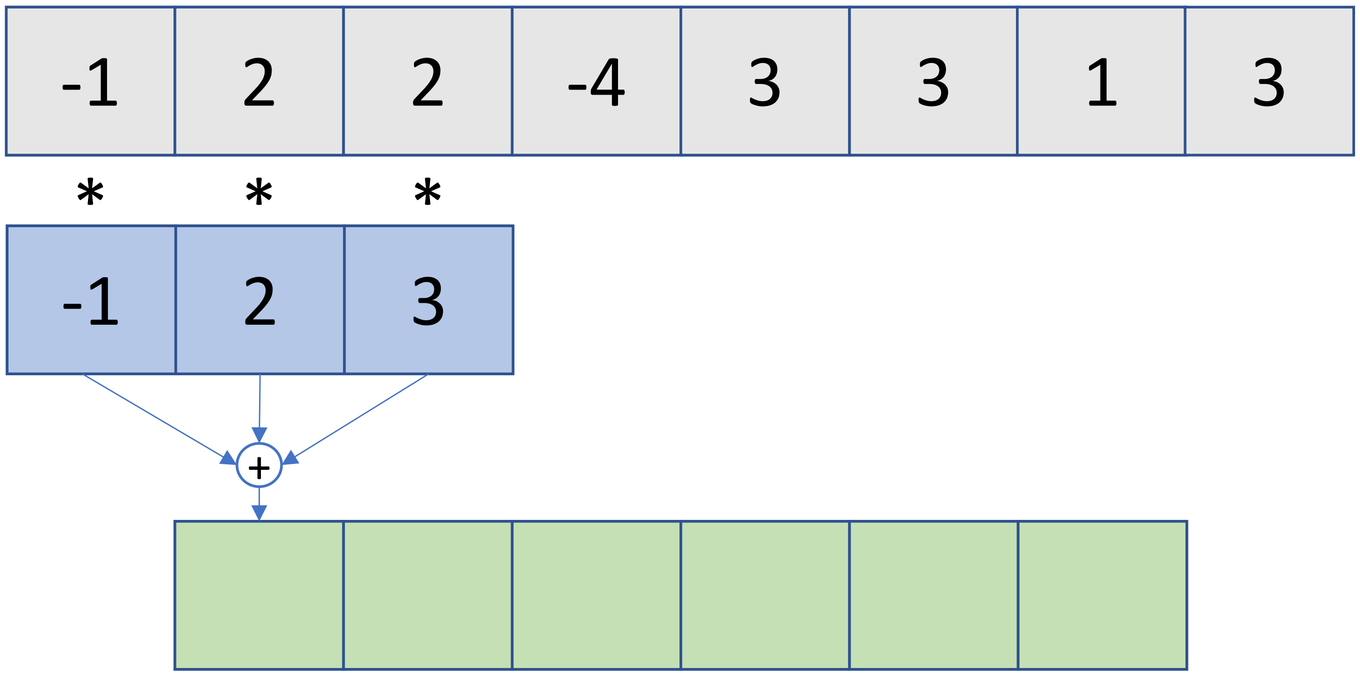


Deep networks

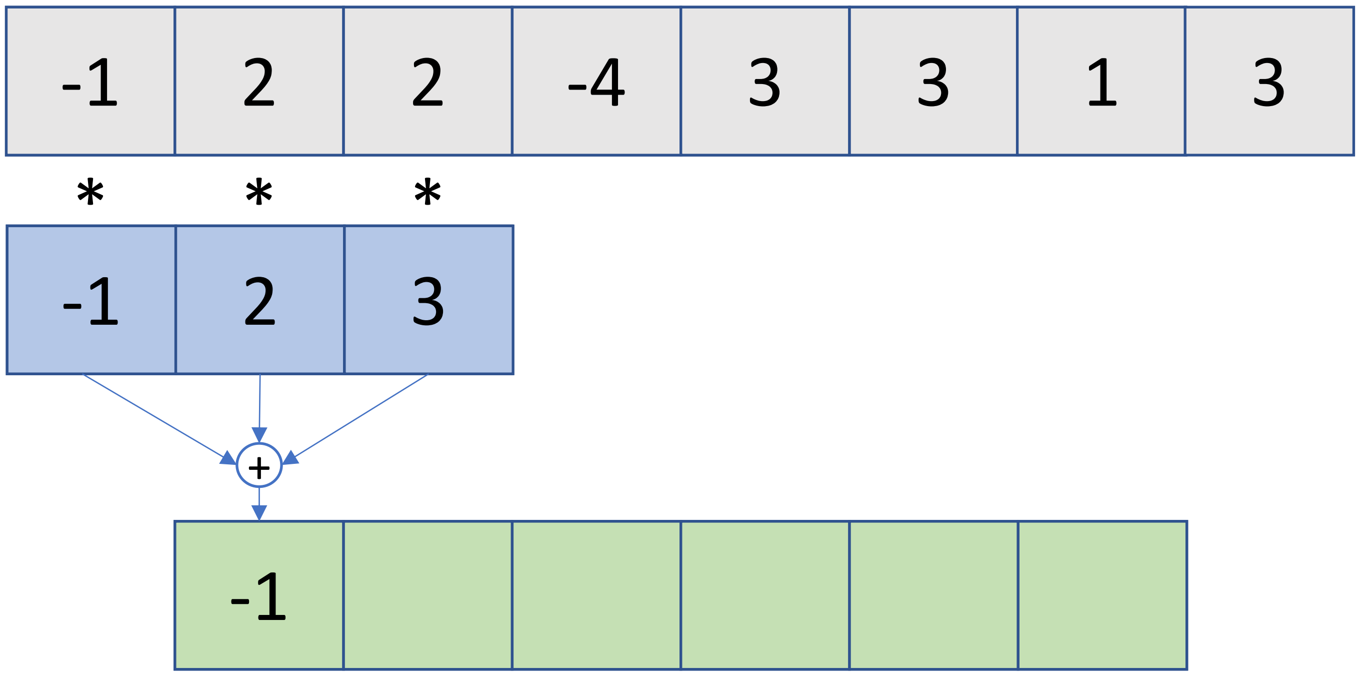
- Hidden layers can learn hierarchical features



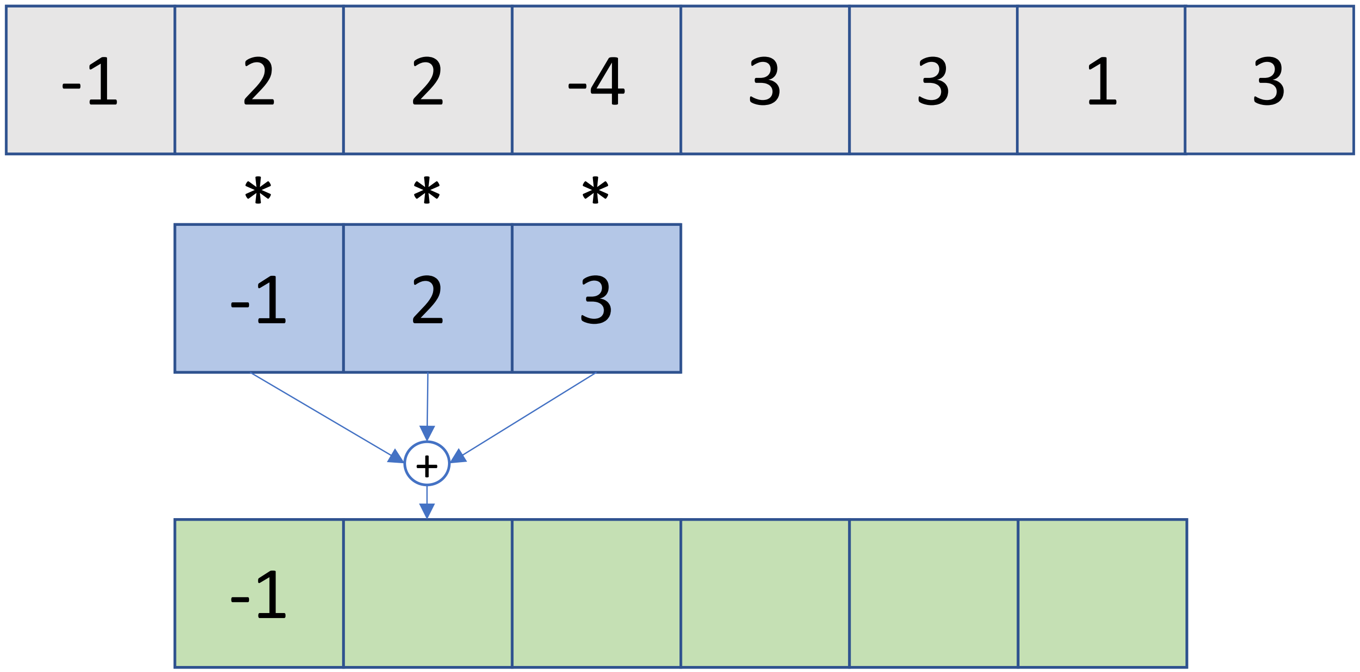
1-D input, 1-D convolution



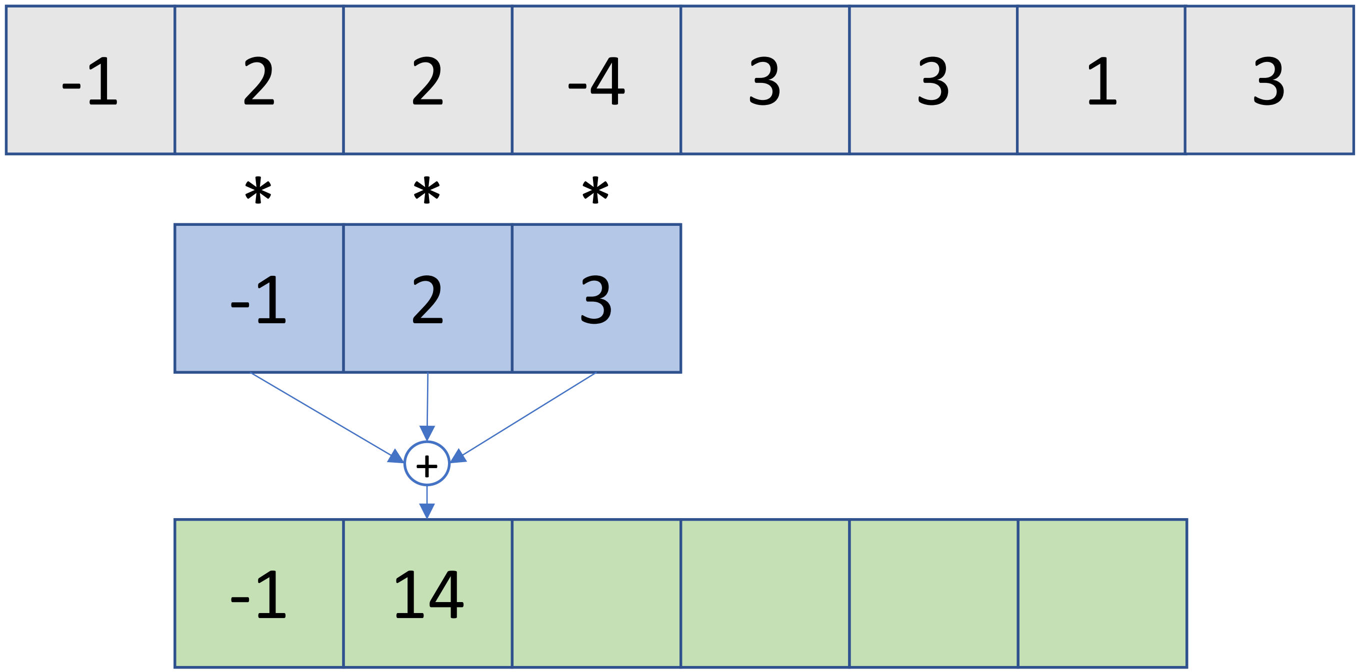
1-D input, 1-D convolution



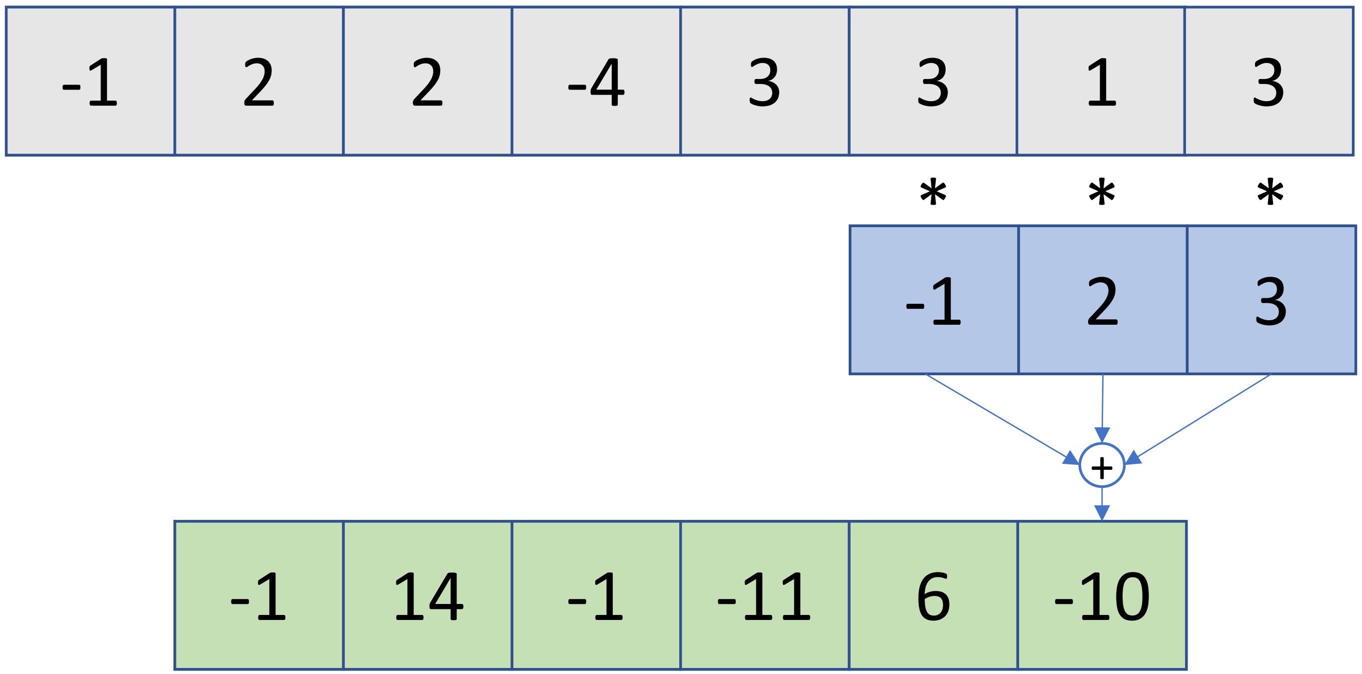
1-D input, 1-D convolution



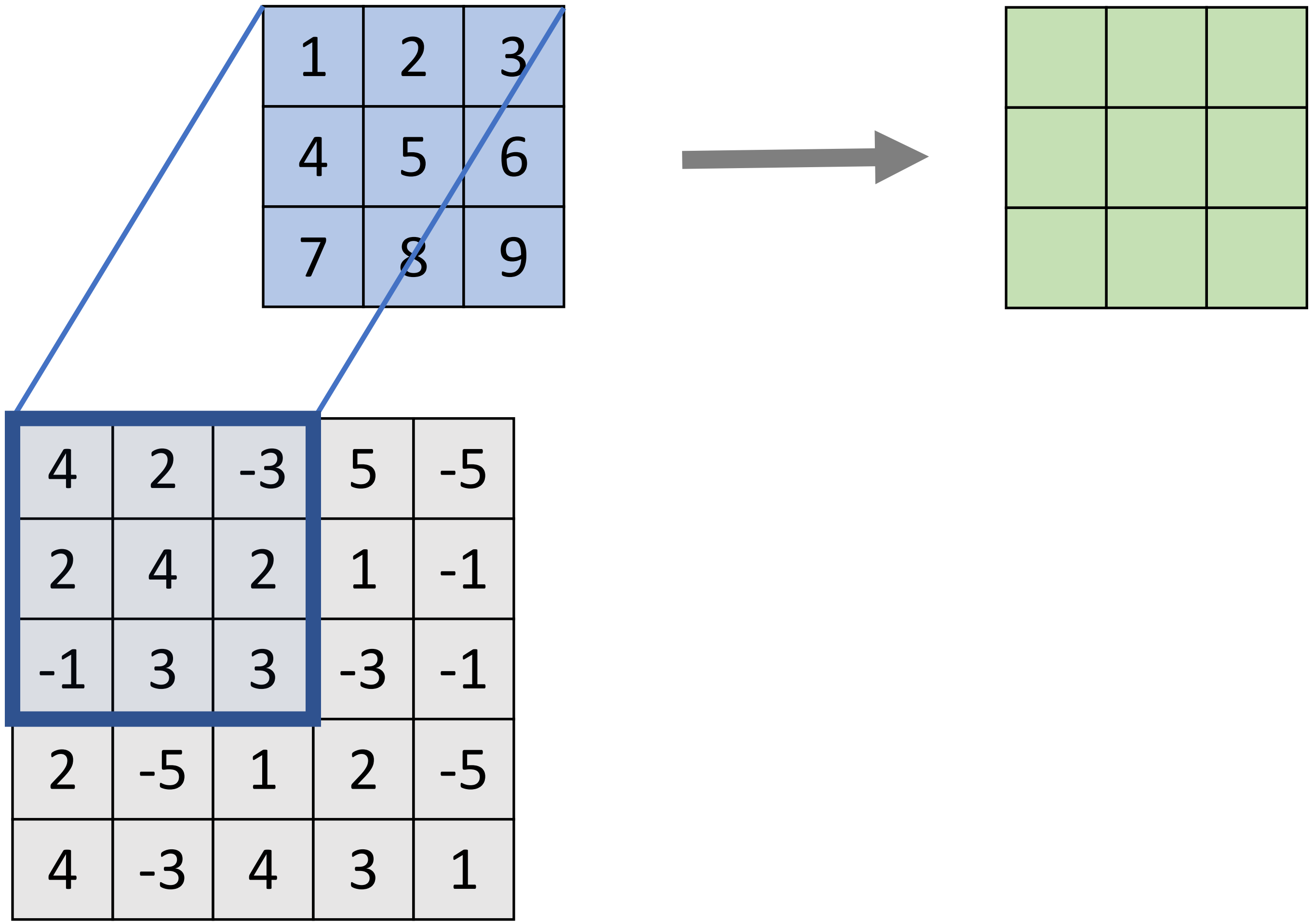
1-D input, 1-D convolution



1-D input, 1-D convolution

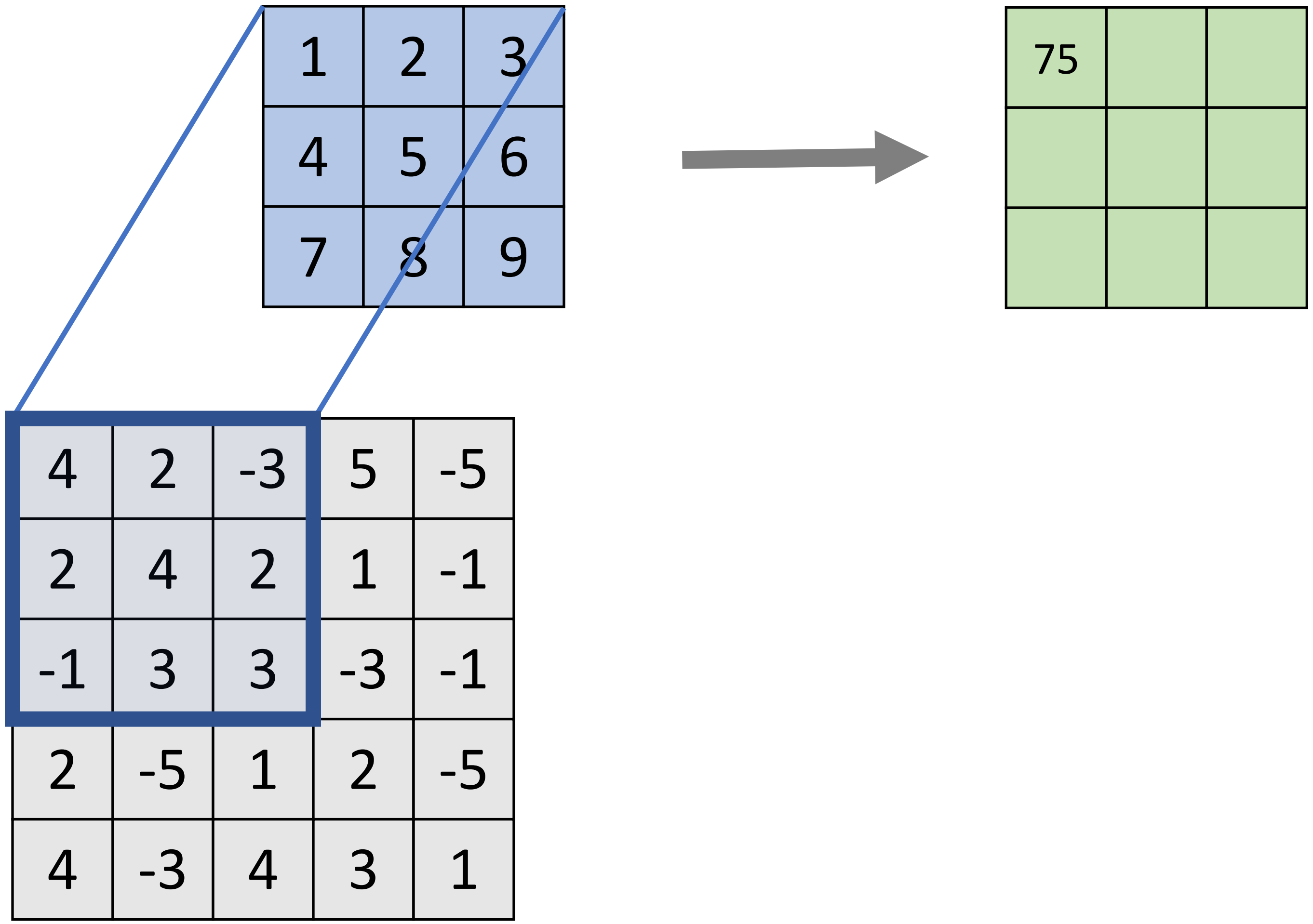


2-D input, 2-D convolution



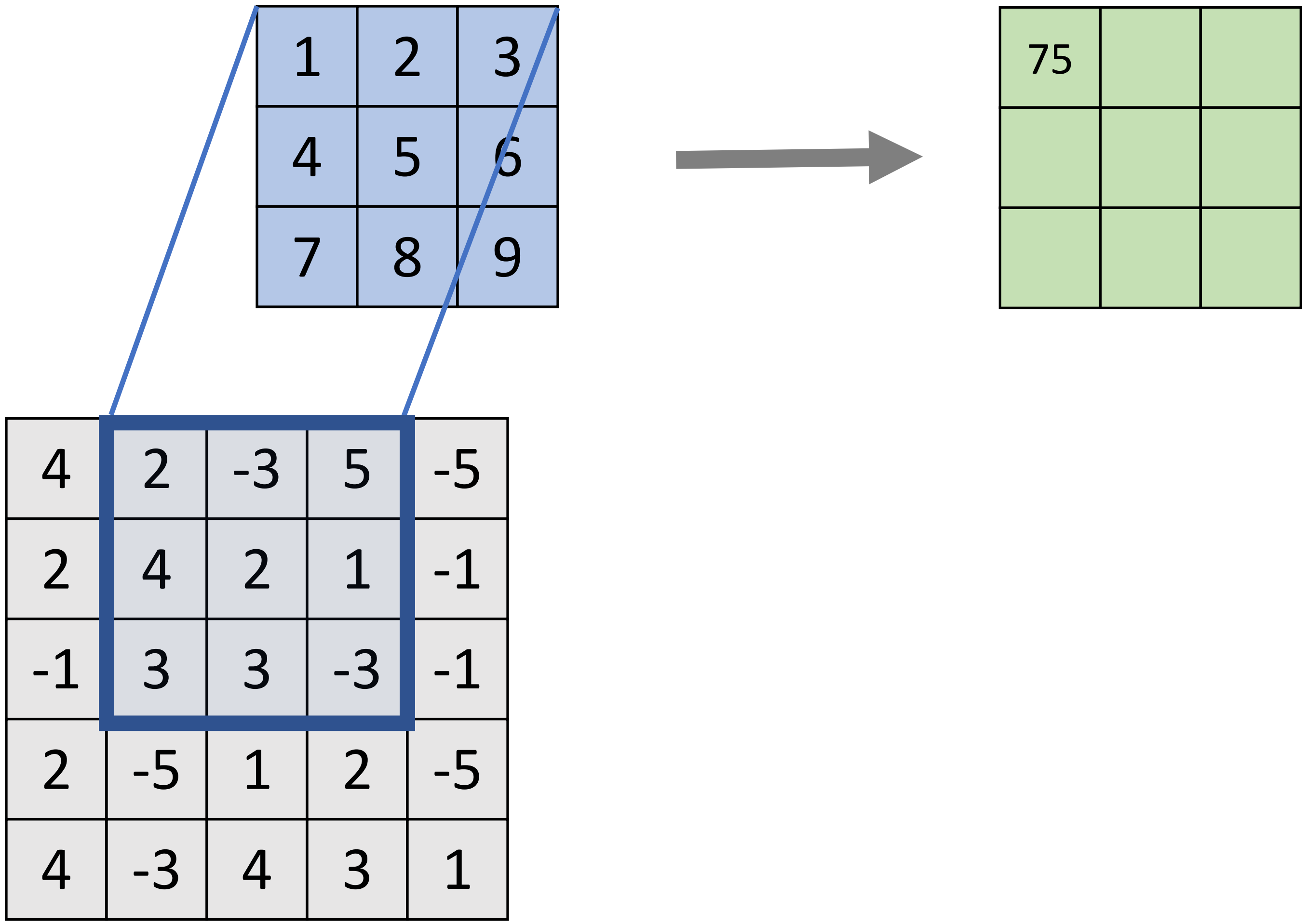
W. J. Gross and B. H. Meyer, EPEPS 2019

2-D input, 2-D convolution

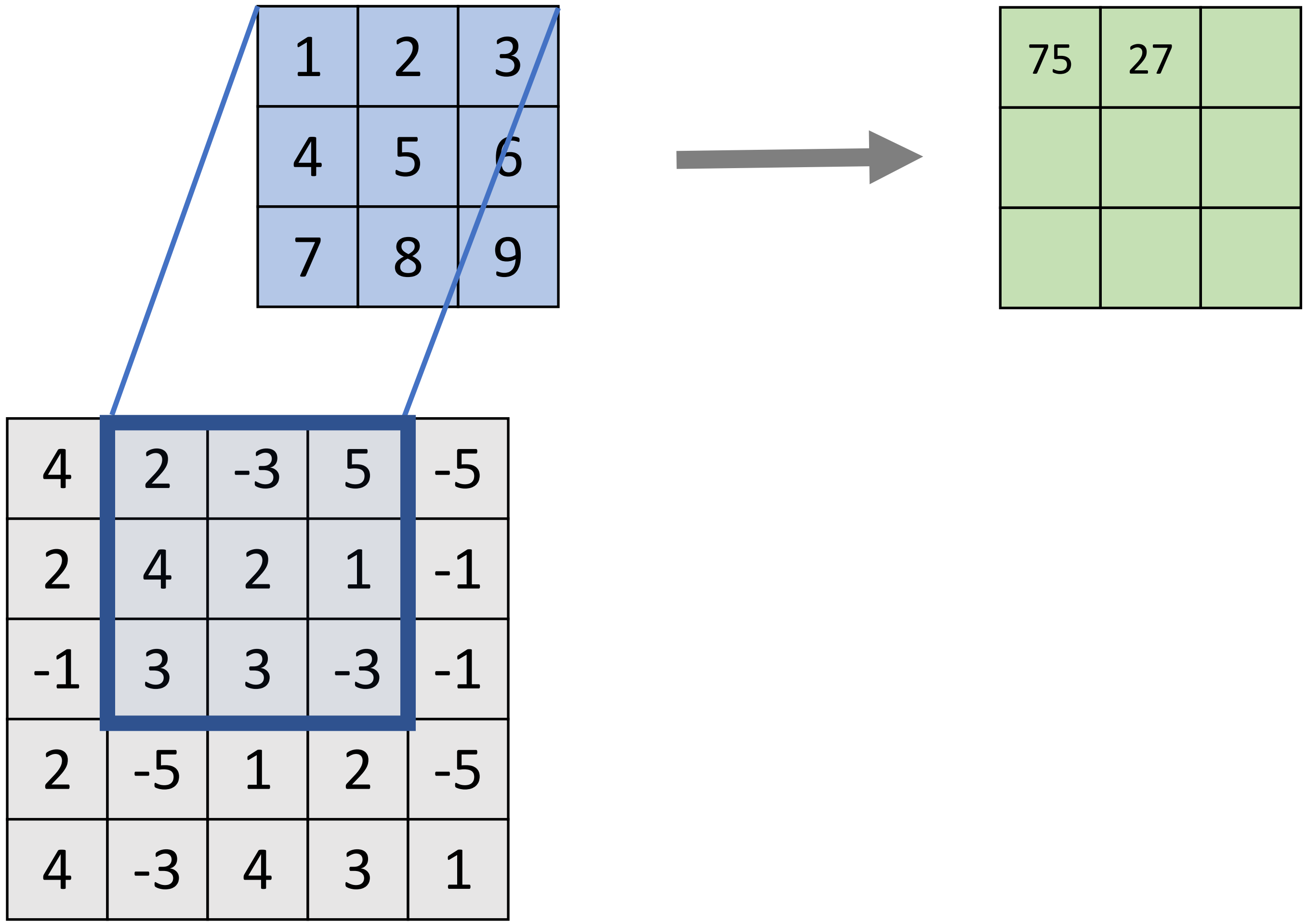


W. J. Gross and B. H. Meyer, EPEPS 2019

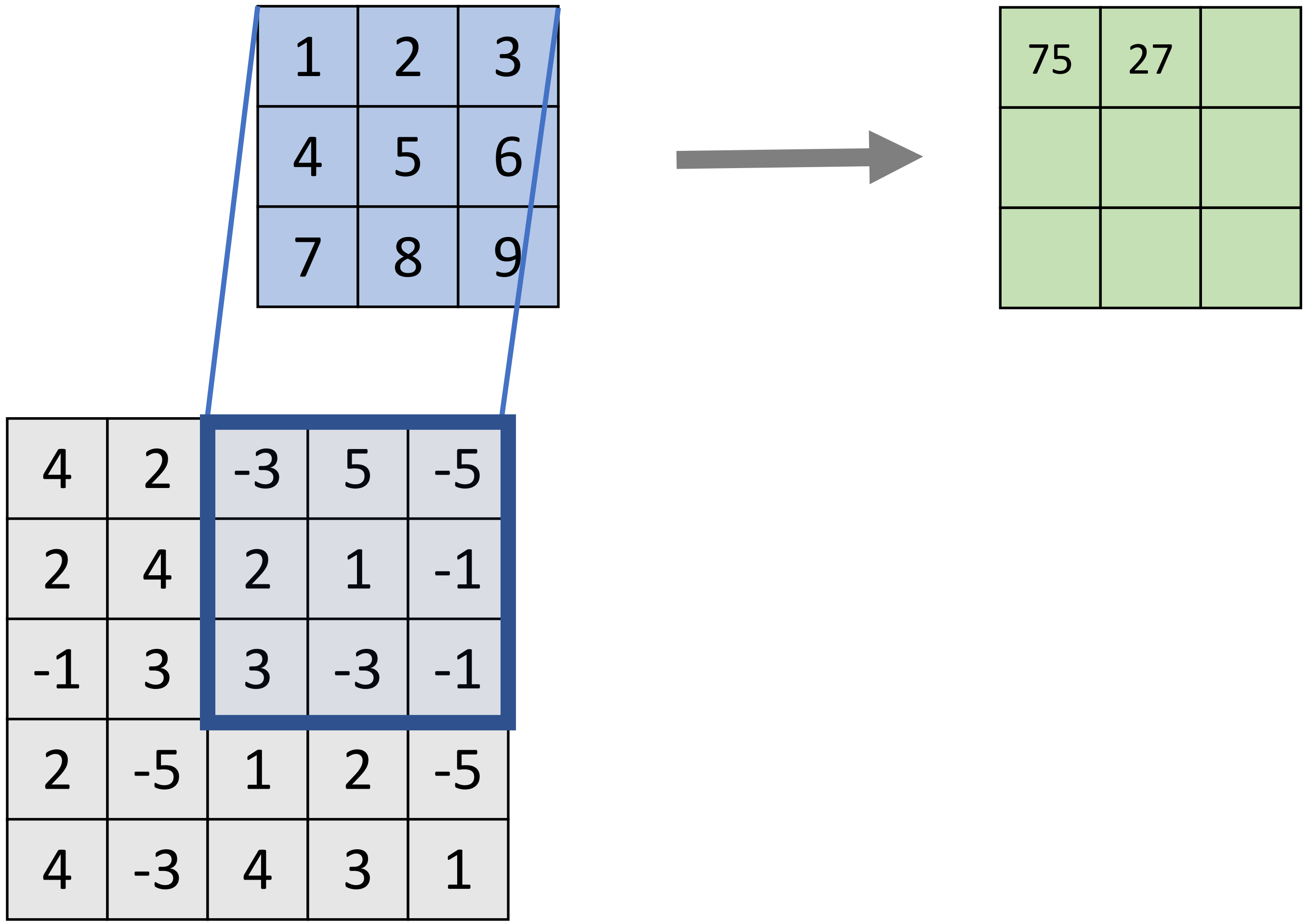
2-D input, 2-D convolution



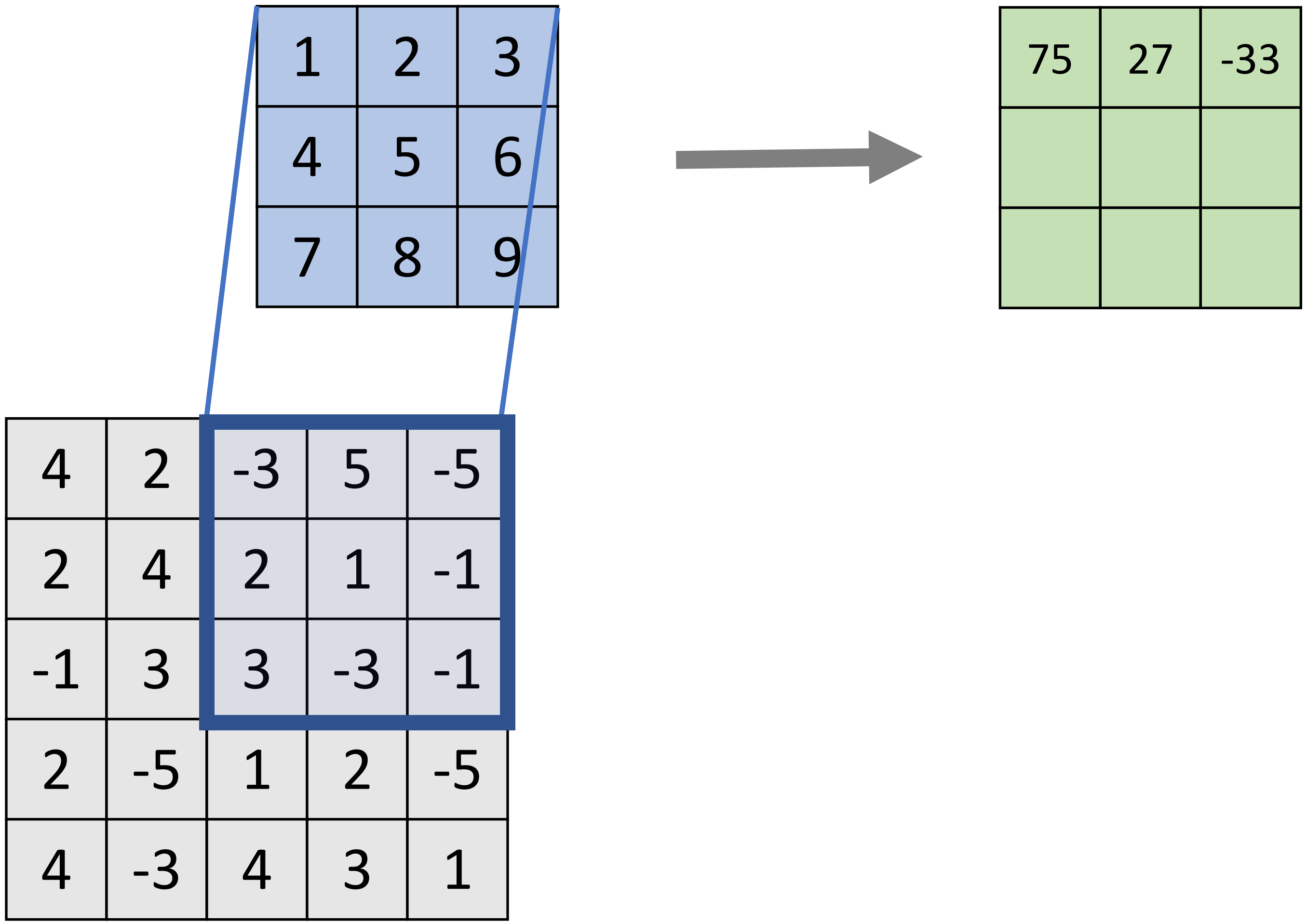
2-D input, 2-D convolution



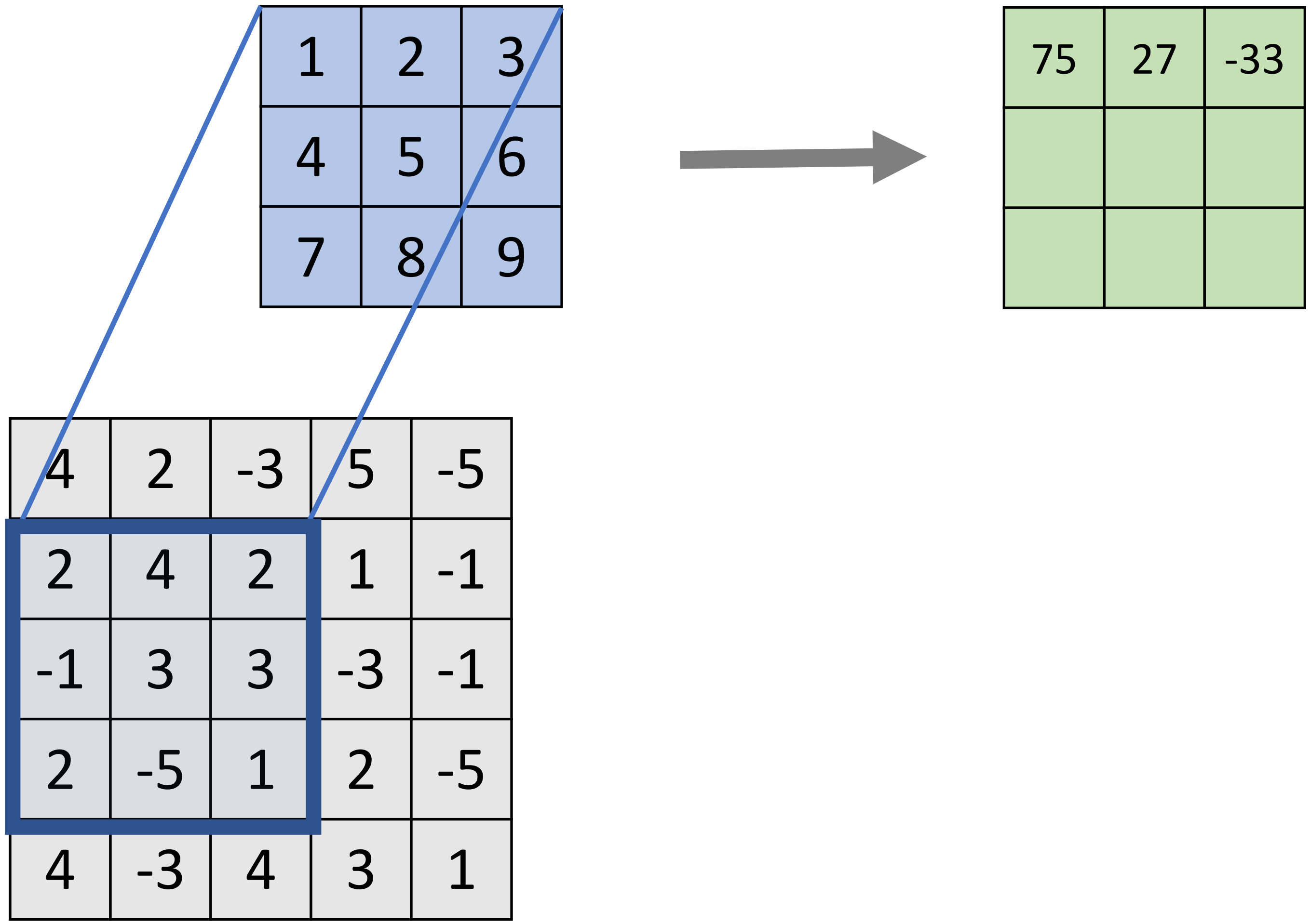
2-D input, 2-D convolution



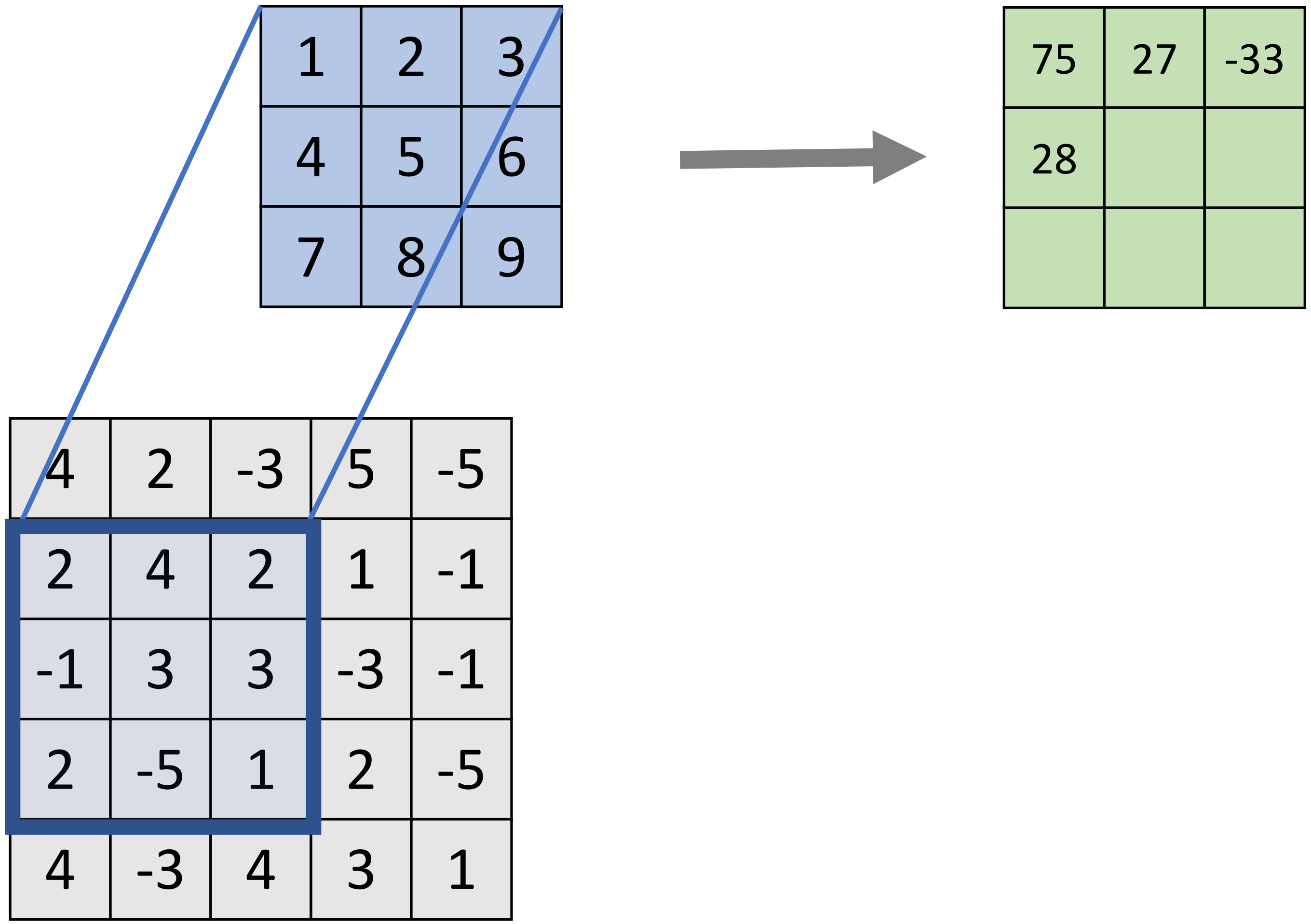
2-D input, 2-D convolution



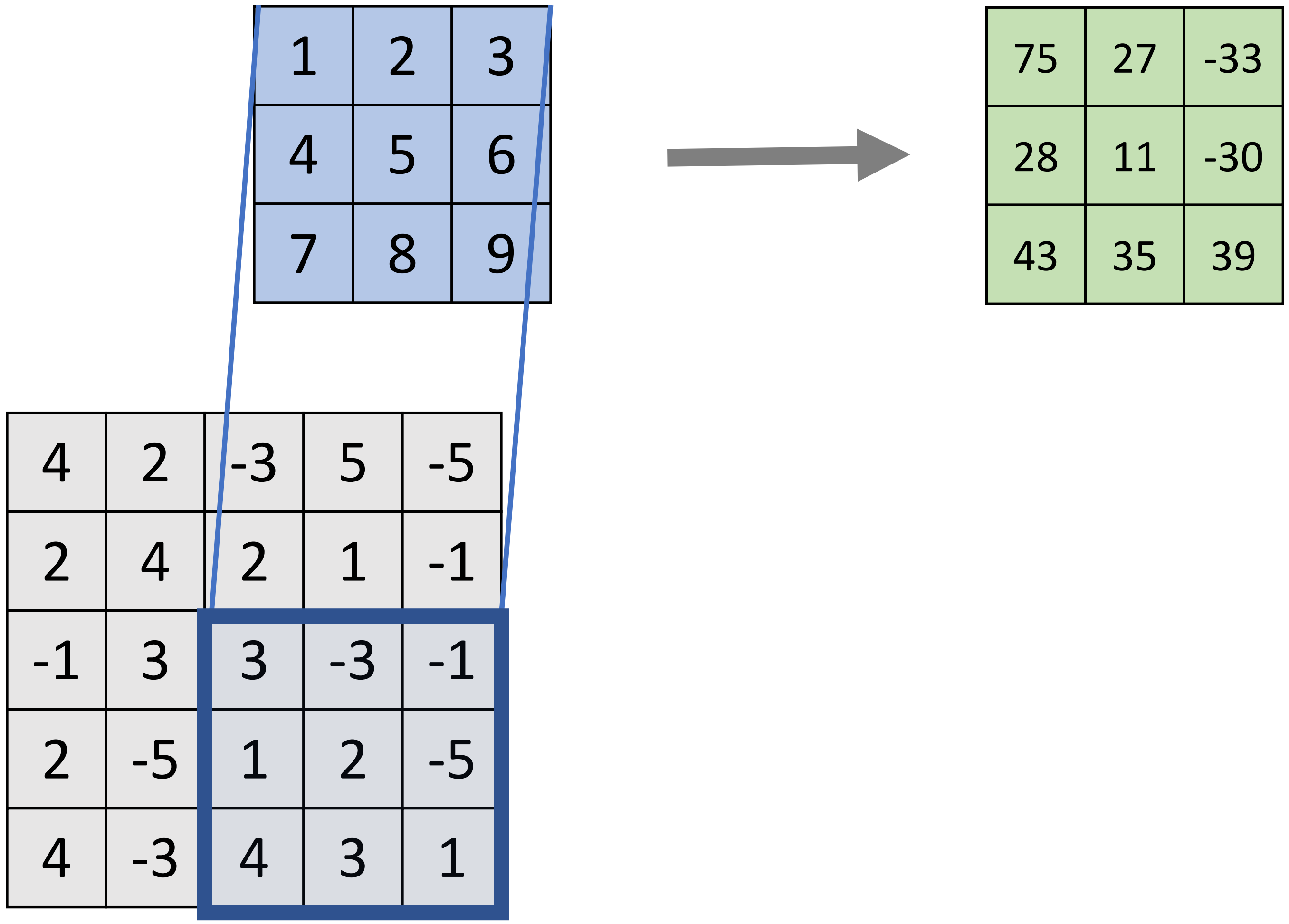
2-D input, 2-D convolution



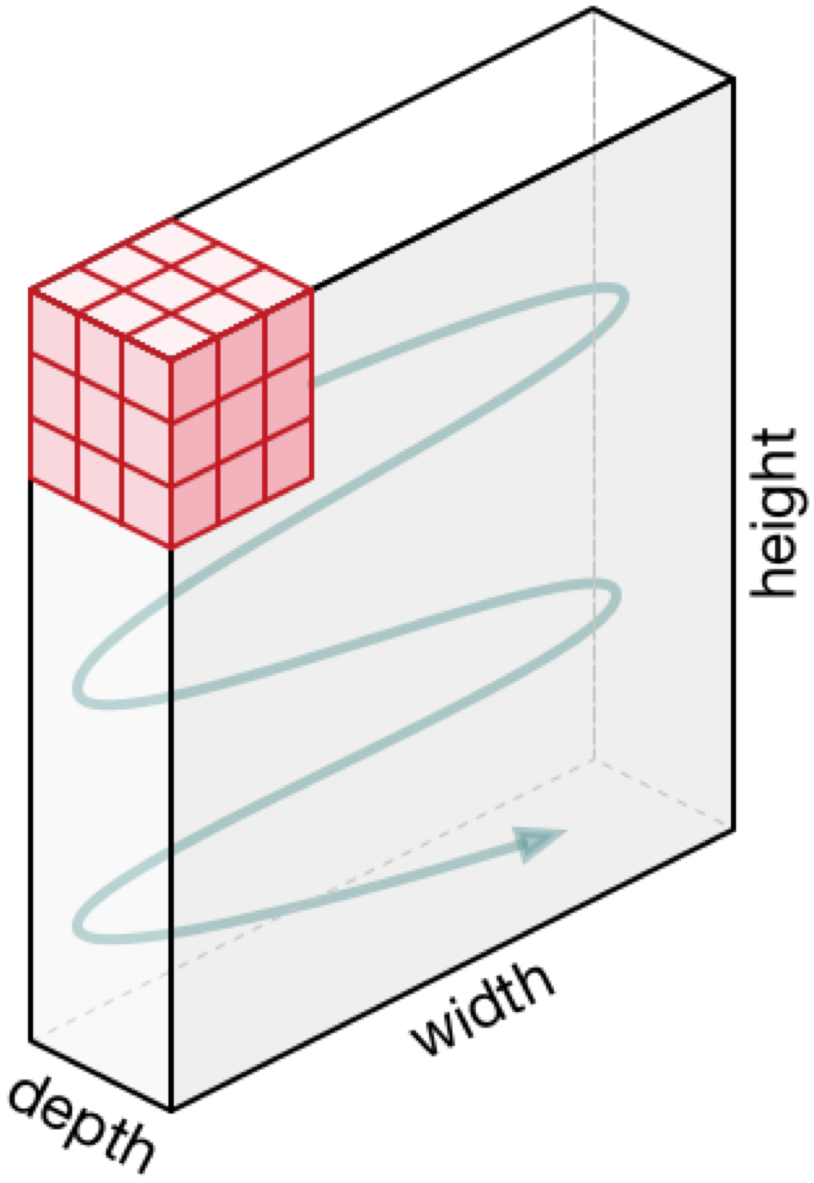
2-D input, 2-D convolution



2-D input, 2-D convolution



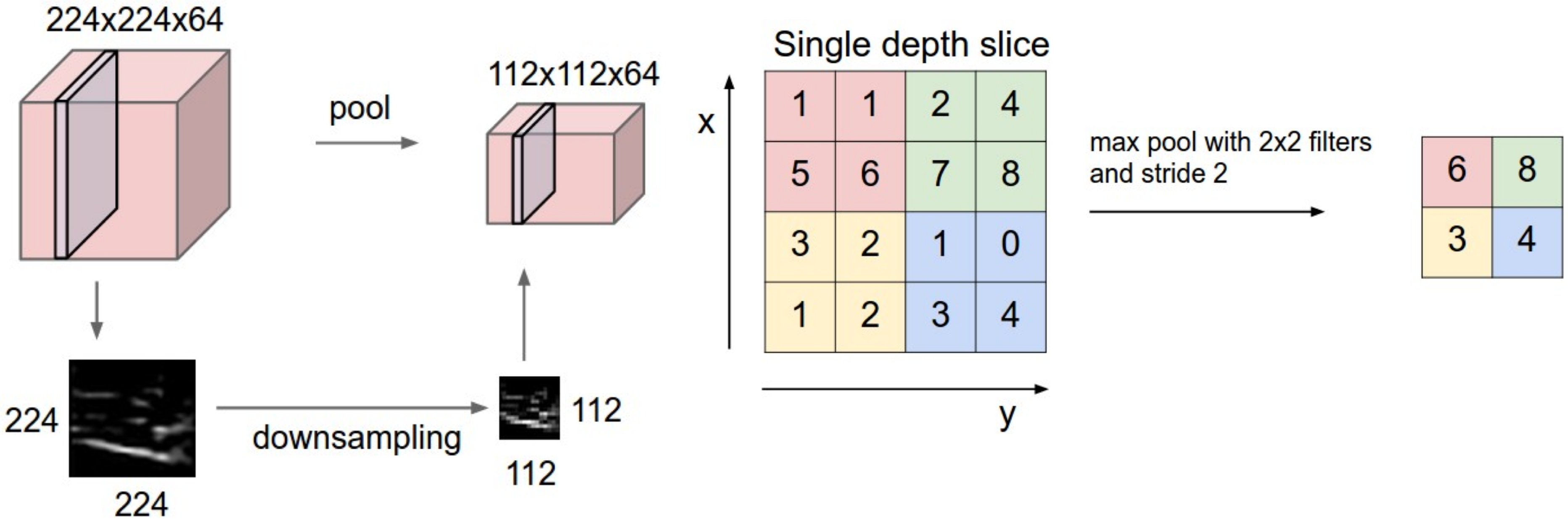
3-D input, 2-D convolution



Pooling

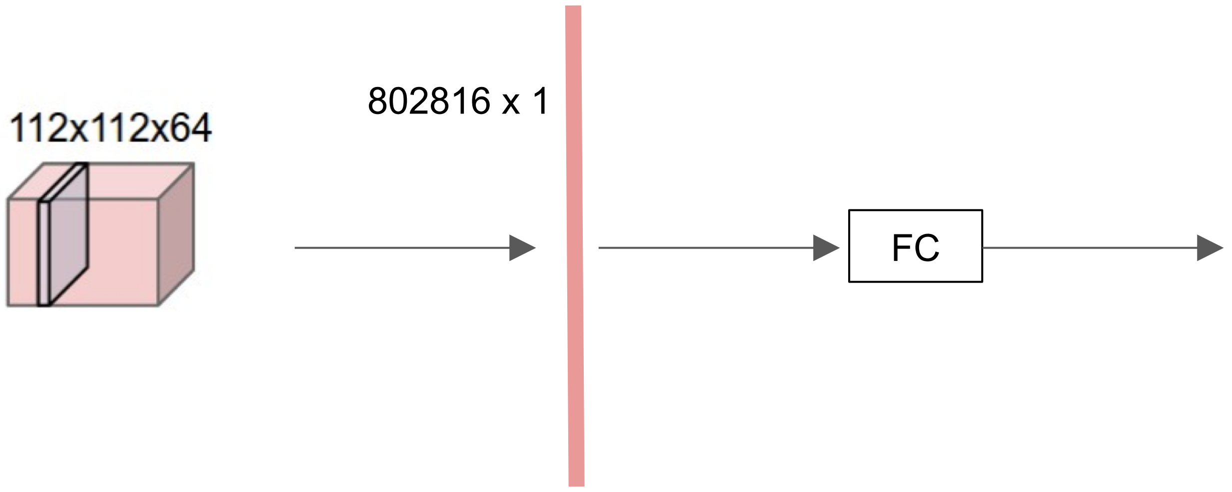
Use mean or max to downsample feature maps

- Reduces computation (without throwing away info)
- Improves translational invariance



Fully-connected layers

Reshape last CONV output into feature vector

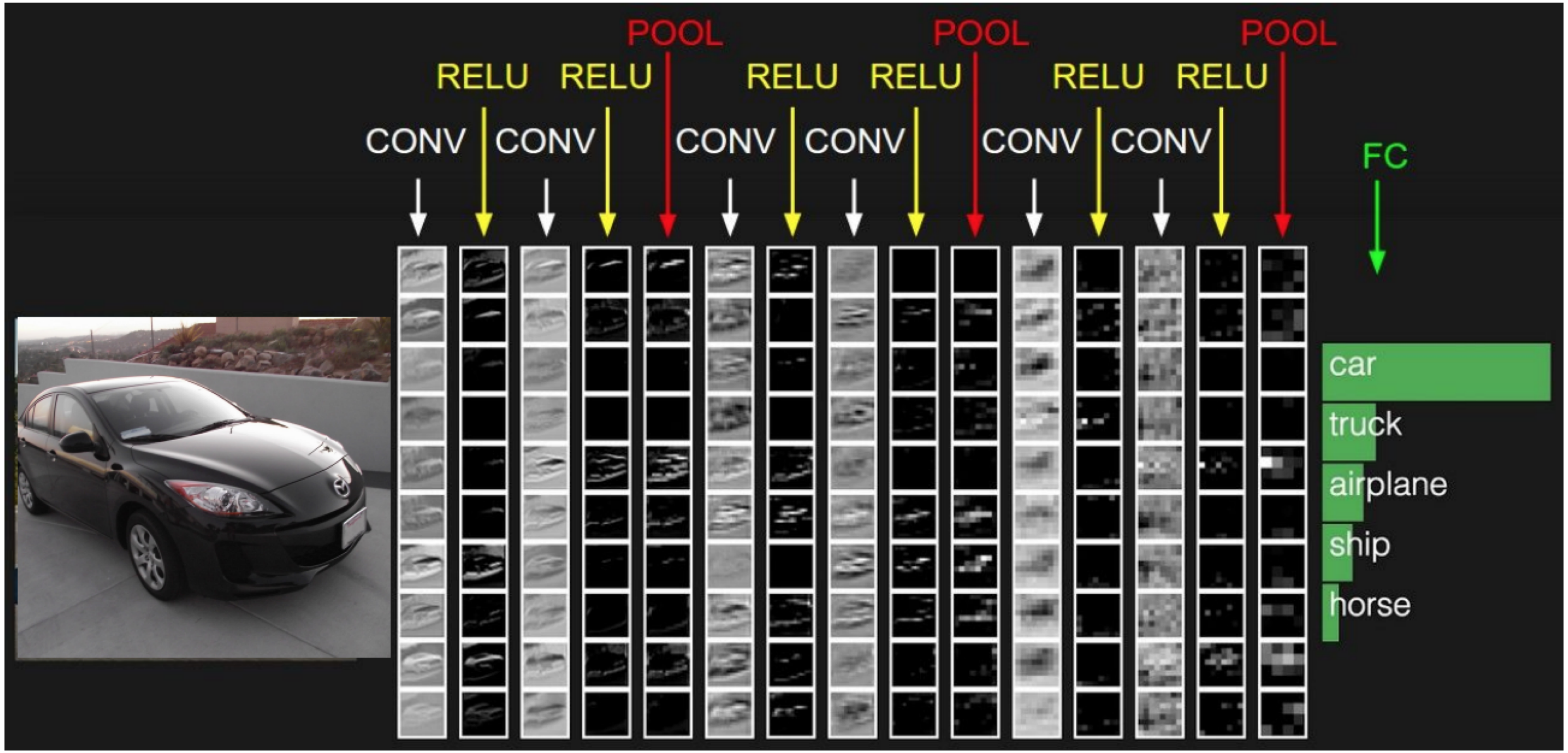


FC layer is equivalent to CONV layer, with

- filter with same size as the input
- no padding

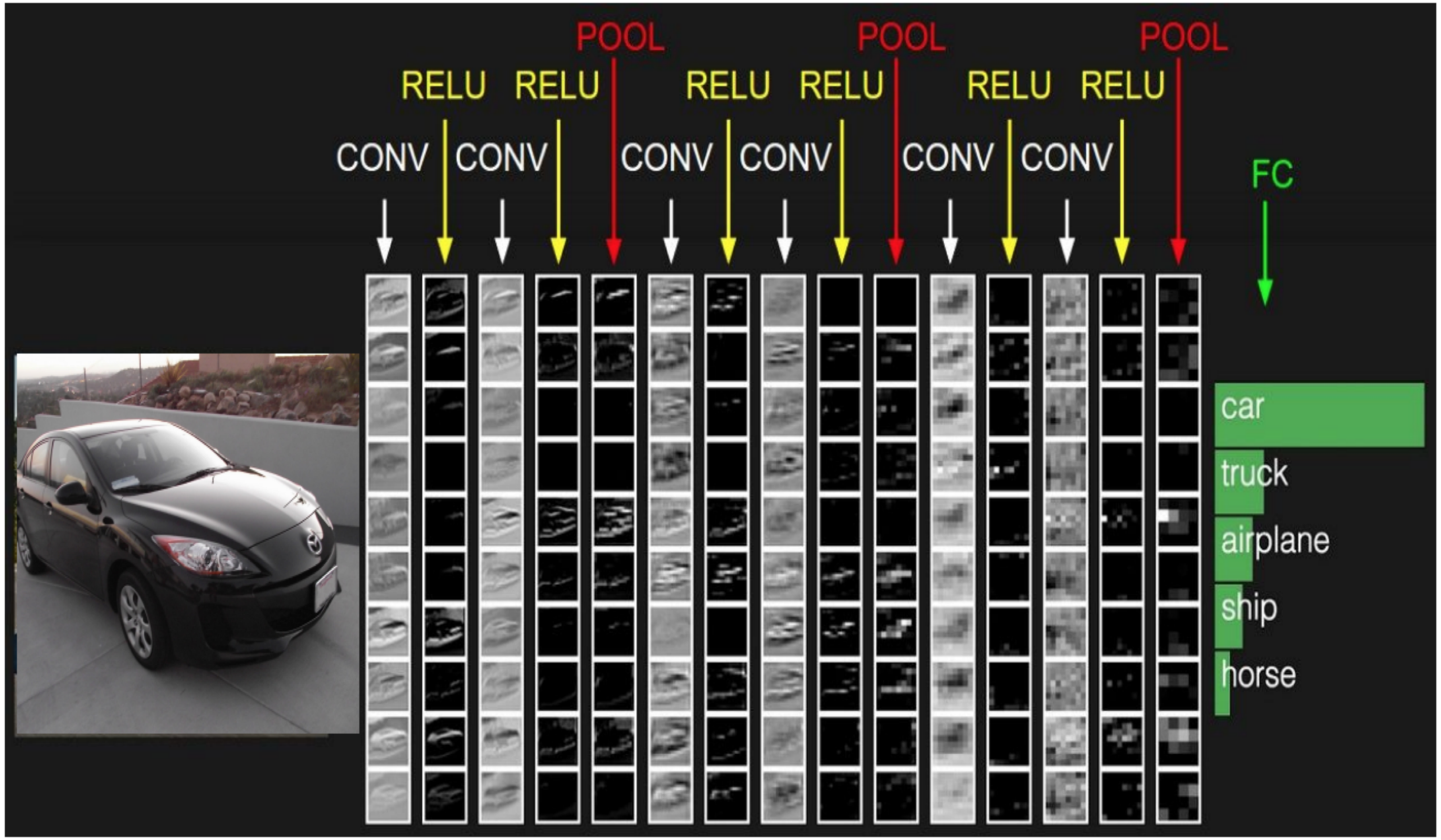
CONV + POOL + RELU + FC = ConvNet

- Train end-to-end using backpropagation + stochastic gradient descent



Typical hyperparameters

- 3x3 filters
- Stride 1
- 2x2 max pooling
- 64 filters



Where is the cost?

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Note:

Most memory is in early CONV

Most params are in late FC

Deep learning processors

Intel Knights Landing (2016)

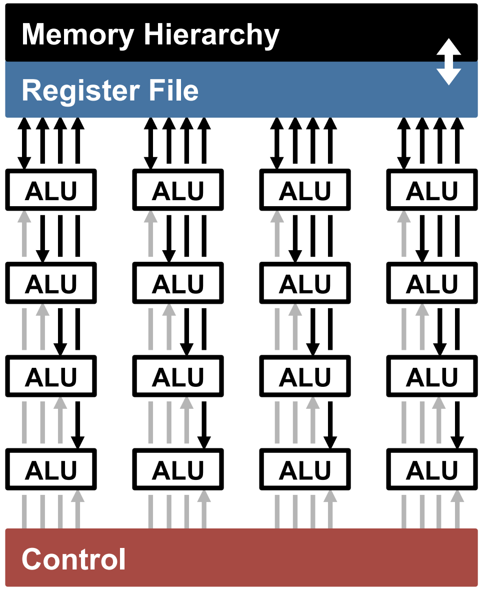
Knights Mill: next gen Xeon Phi "optimized for deep learning"

Nvidia PASCAL GP100 (2016)

- GPUs most commonly used compute engine for DNNs
- Specialized hardware can be designed for more efficient processing
  - e.g. Intel Knights Landing CPU has special vector instructions for deep learning
  - NVIDIA PASCAL GP100 GPU has 16-bit floating point arithmetic to perform two FP16 operations on a single-precision core
- Fundamental operation in DNN (both CONV and Fully-connected layers) is *multiply-and-accumulate* (MAC)

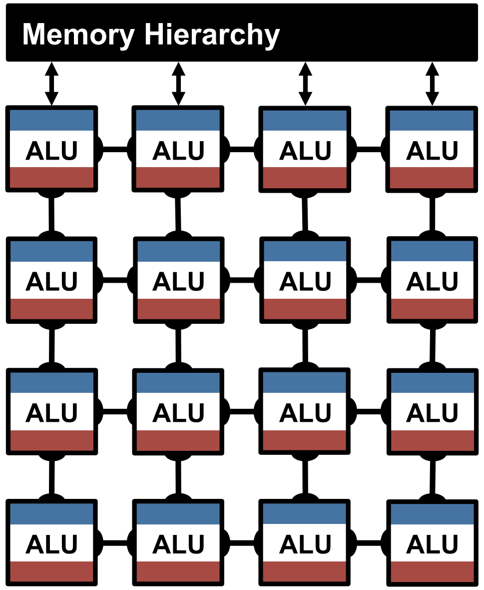
Parallelizing MACs

Temporal Architecture (SIMD/SIMT)



- CPUs, GPUs
- Vectors (SIMD) or parallel threads (SIMT)
- Centralized control -large number of ALUs
- ALUs only communicate with the memory hierarchy and not each other
- Goal: reduce multiplications to increase throughput

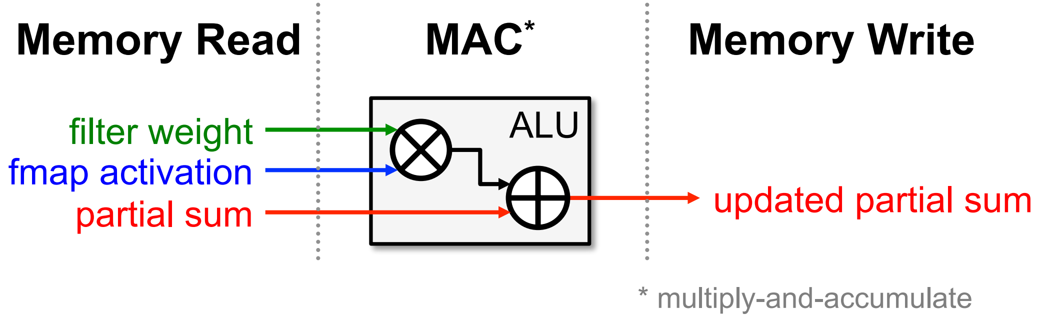
Spatial Architecture (Dataflow Processing)

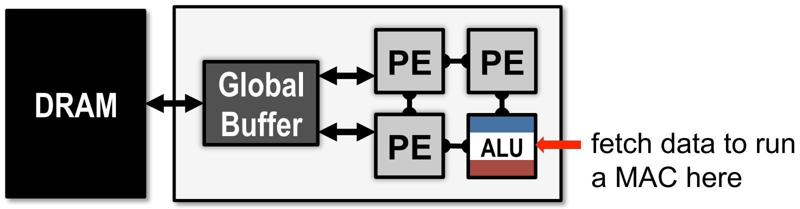


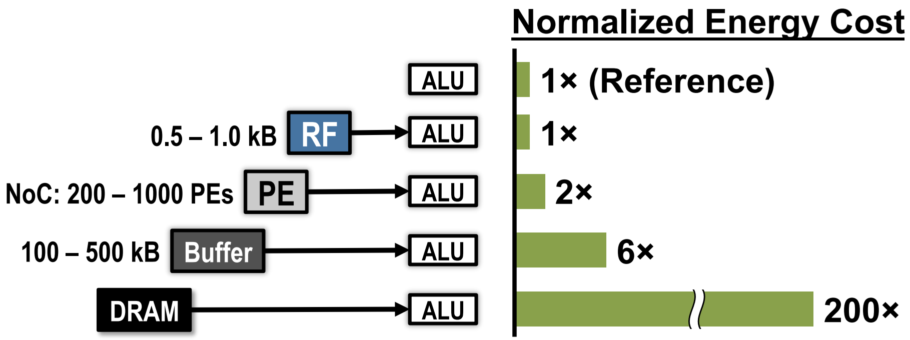
- ASIC / FPGA
- Processing chain with local interconnection
- ALU + local memory = PE
- Goal: reuse data from low-cost memories in hierarchy to reduce energy consumption

Energy-Efficient Dataflow for Accelerators

- 3 memory reads and 1 memory write per MAC
- E.g. AlexNet: 724M MACs -> 3000M DRAM accesses required
- DRAM access require several OOM higher energy than computation
- Spatial architectures reduce energy cost for data movement by using several memories of lower-cost local memory hierarchy

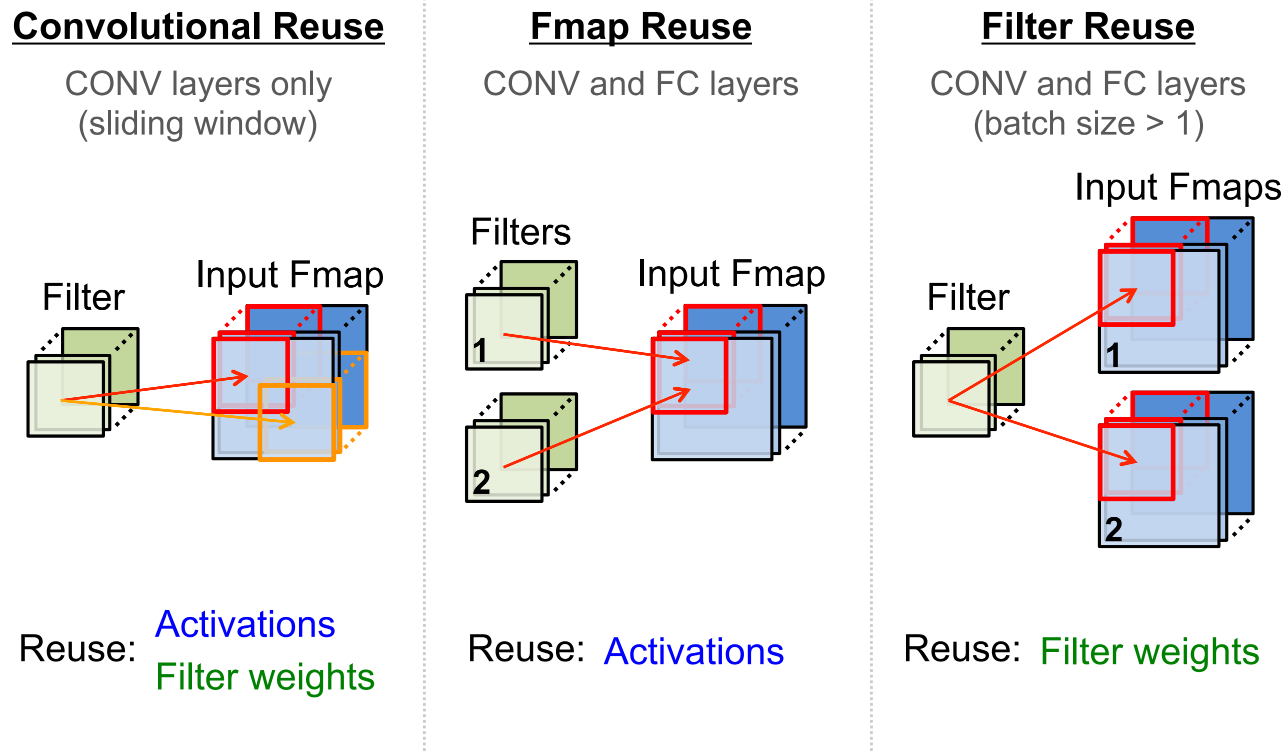




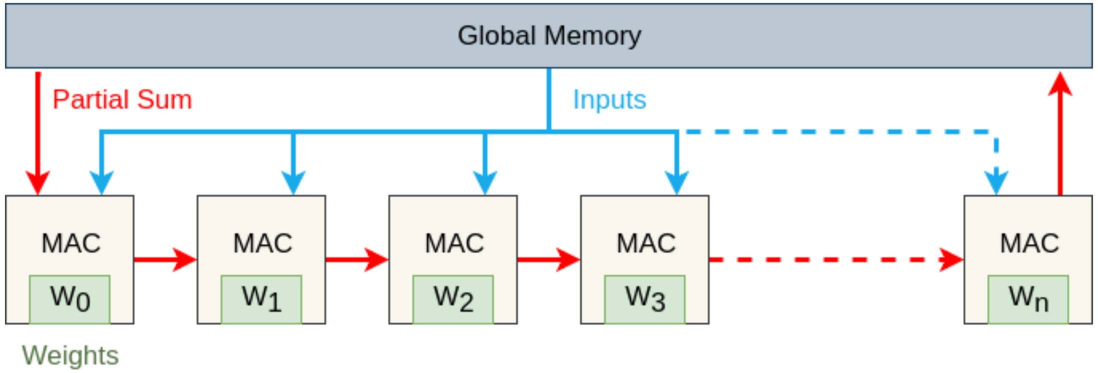


Input Data Reuse

- **Convolutional** – same input feature map activations and weights are used within a given channel – in different combinations for different weighted sums
- **Feature map** – multiple filters are applied to the same feature map so that input feature map filters are used multiple times across filters
- **Filter** – in batch processing (multiple input feature maps processed at once) – the same filter weights are used multiple times across input feature maps
- AlexNet: 3000M -> 61M DRAM accesses.
- Partial sum accumulation can be done in local memory.



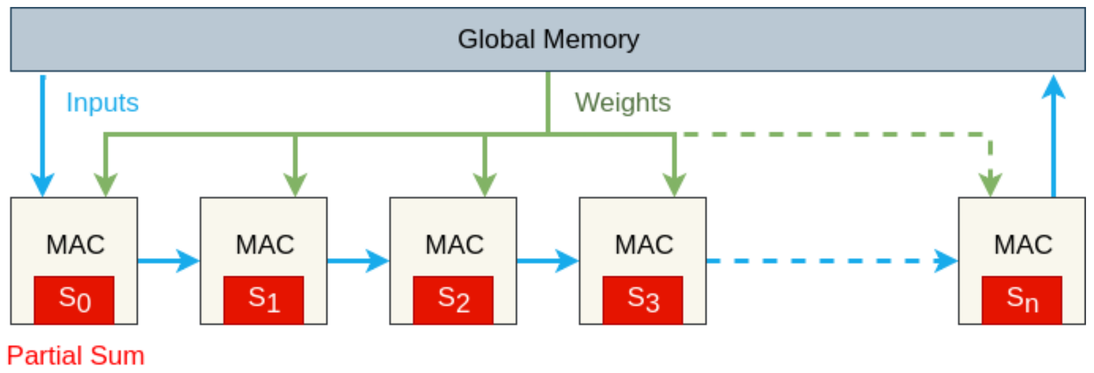
Weight Stationary (WS)



- Minimize the number of memory accesses to weights
- Maximize filter reuse of weights
- Requires parallel access to input pixels
- Examples:
    - NeuFlow[1]
    - Park[2]

[1] Farabet et al., "NeuFlow: A runtime reconfigurable dataflow processor for vision," CVPR 2011 WORKSHOPS, 2011.
[2] Park et al., "A 1.93TOPS/W Scalable Deep Learning/Inference Processor with Tetra-Parallel MIMD Architecture for Big-Data Applications," ISSCC 2015.

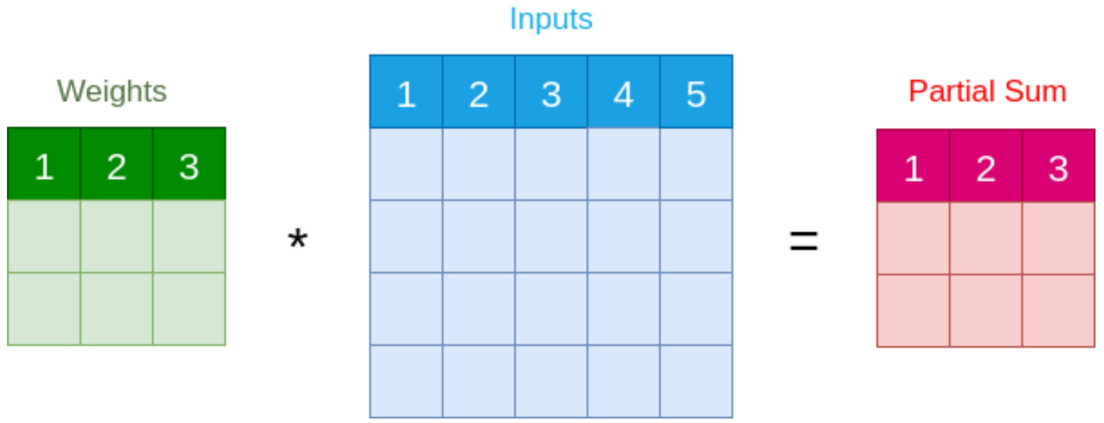
Output Stationary (OS)

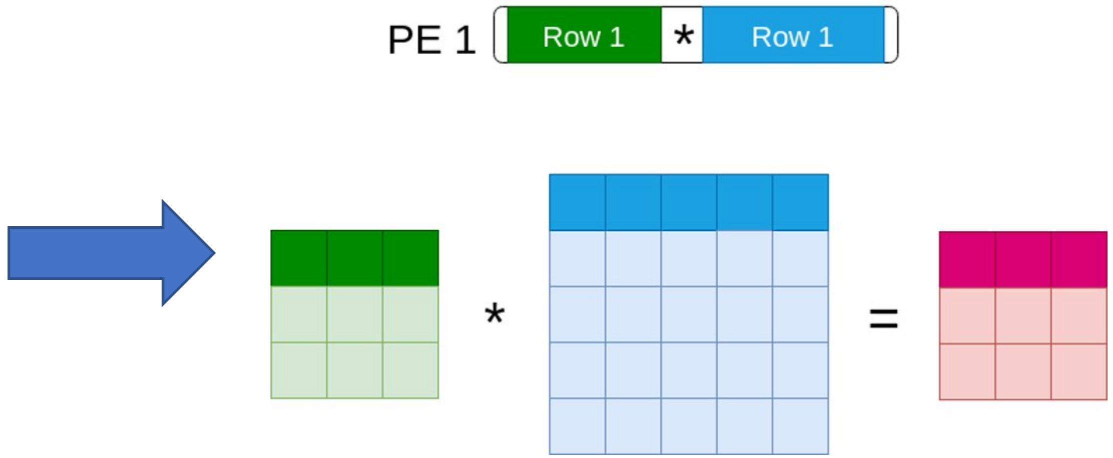


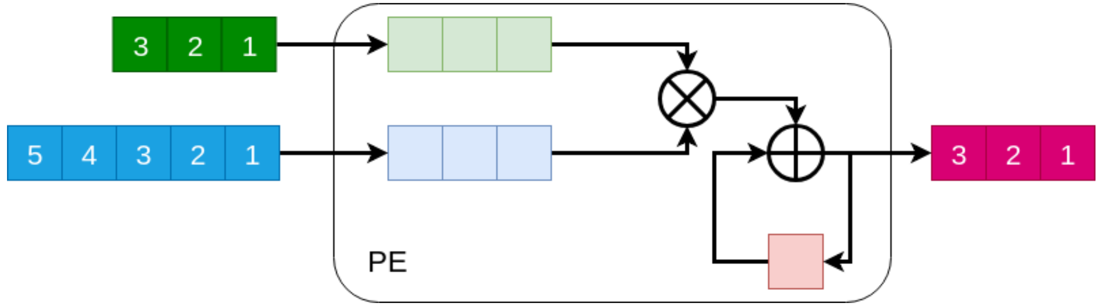
- Minimize read/write accesses for partial sum
- Maximize local accumulation
- Requires parallel access to weights
- Examples:
  - ShiDianNao[1]
  - Gupta[2]

[1] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," ISCA, 2015.
[2] S. Gupta et al., "Deep learning with limited numerical precision," ICML, 2015.

Row Stationary (RS): 1-D Convolution in PE



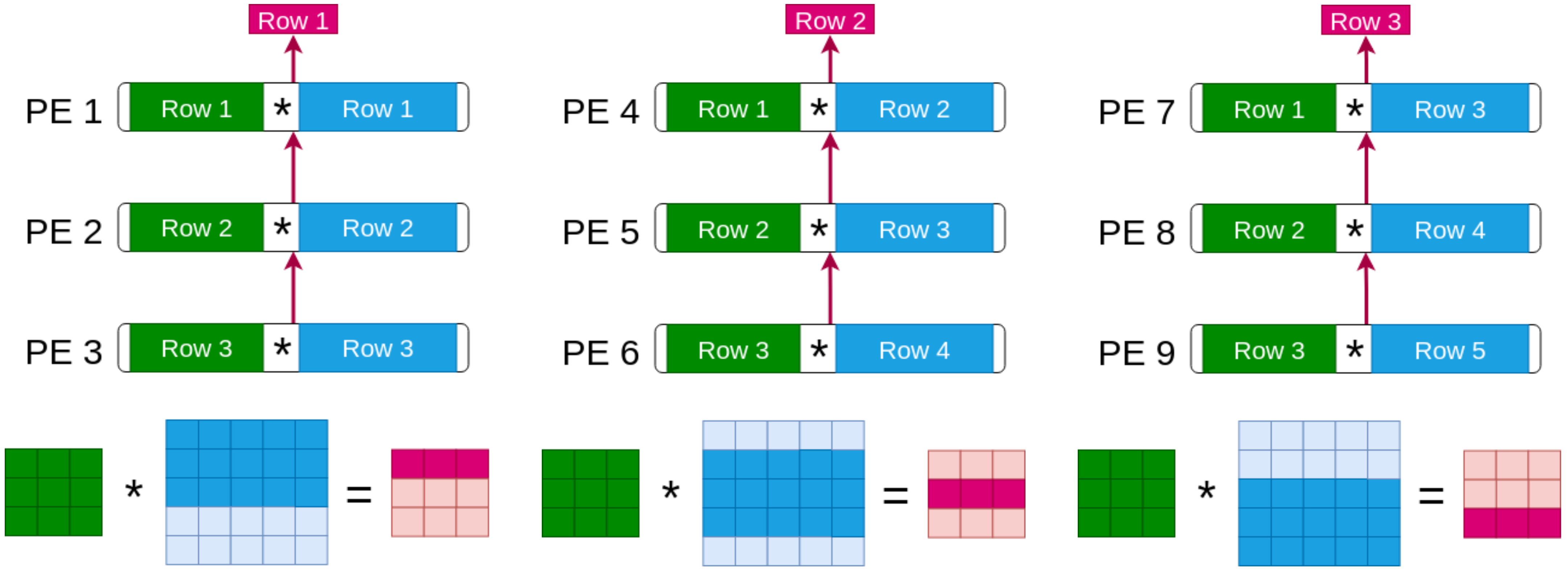




- Maximize filter/image row reuse in register file
- Maximize partial sum accumulation in register file
- Example: Eyeriss[1]

Y. Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ISCA*, 2016.

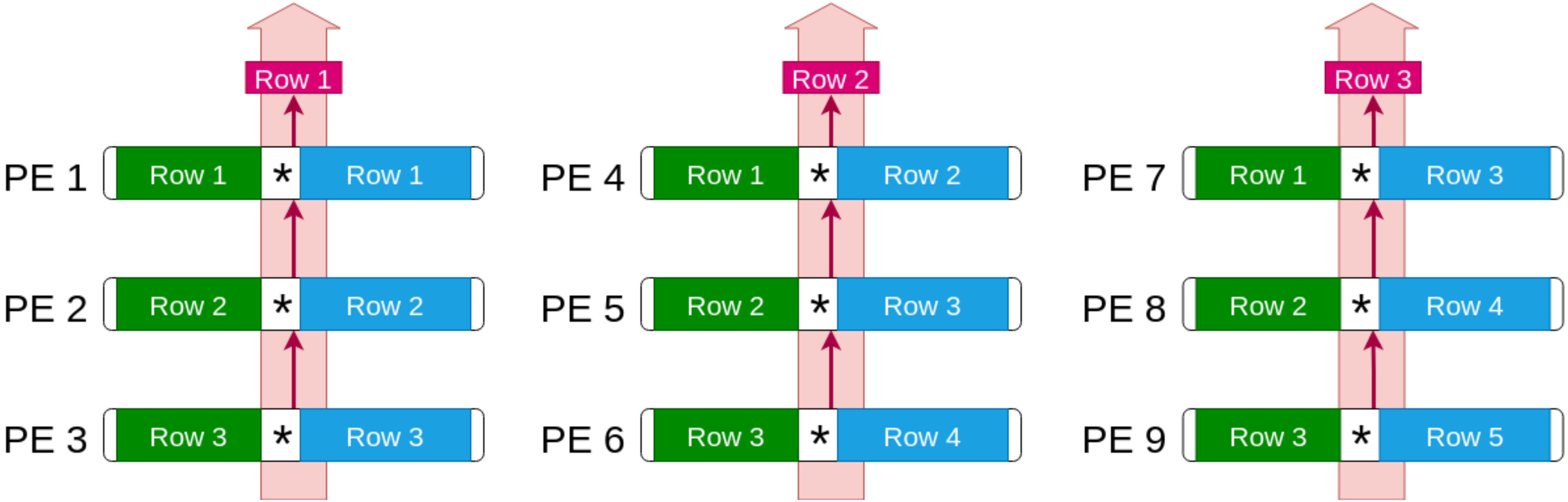
Row Stationary (RS): 2-D Convolution in PE



- To perform 2-d convolutions, arrange PEs in a 2-D form
- Each PE performs a row-wise convolution

Y. Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ISCA*, 2016.

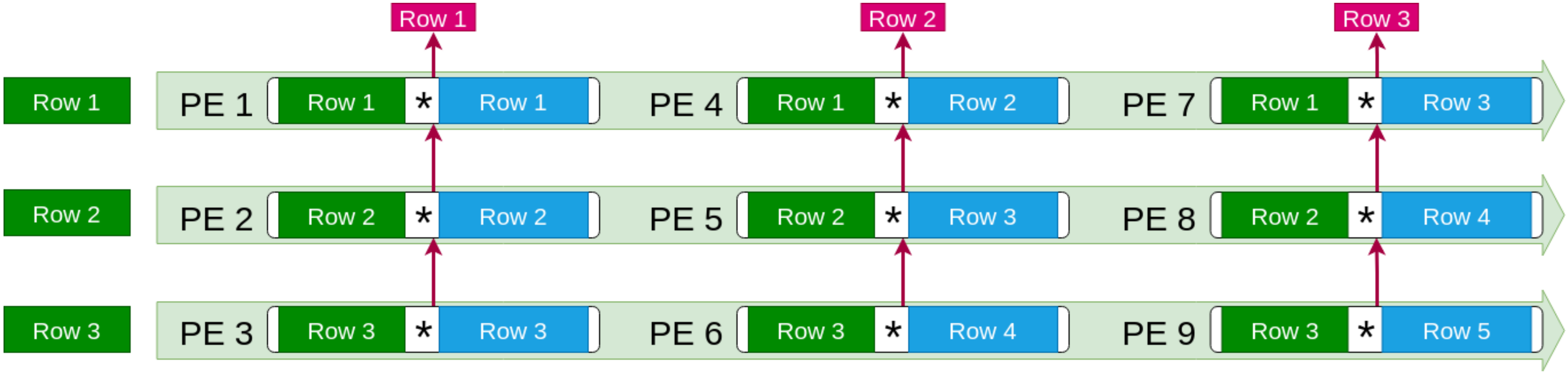
Row Stationary (RS)



- Veritical partial sum accumulation across PEs

Y. Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ISCA*, 2016.

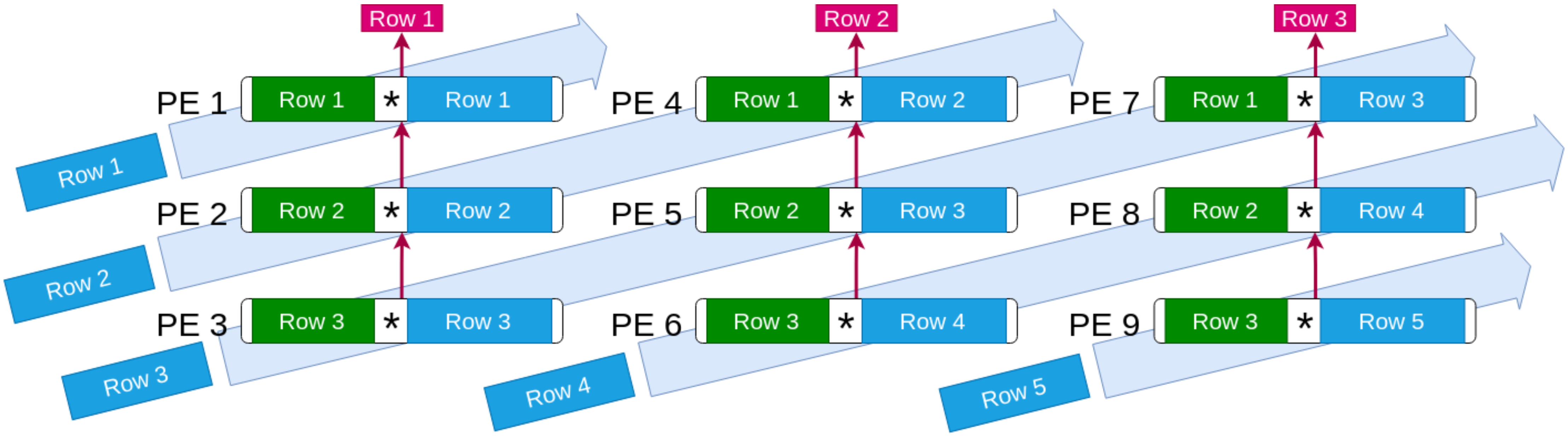
Row Stationary (RS)



- Horizontal filter row reuse across PEs

Y. Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ISCA*, 2016.

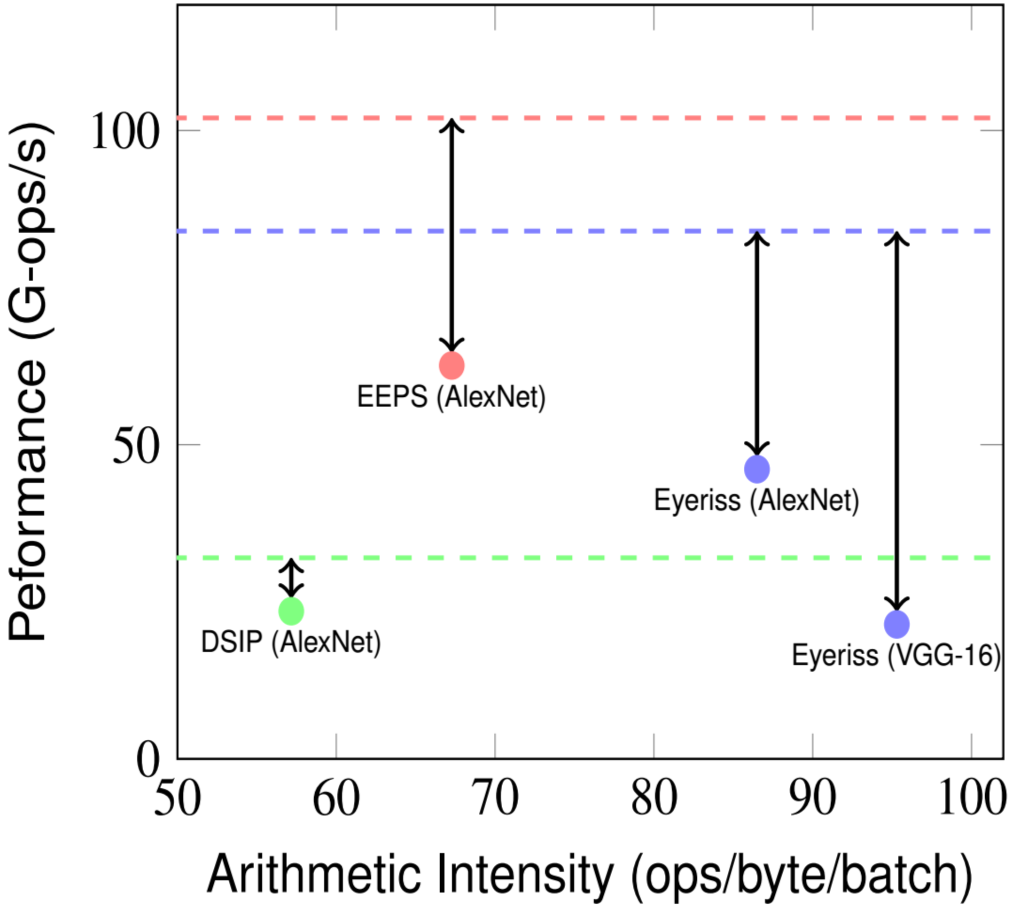
Row Stationary (RS)



- Diagonal image row reuse across PEs

Y. Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ISCA*, 2016.

Fast Efficient Inference Engine (FEIE)



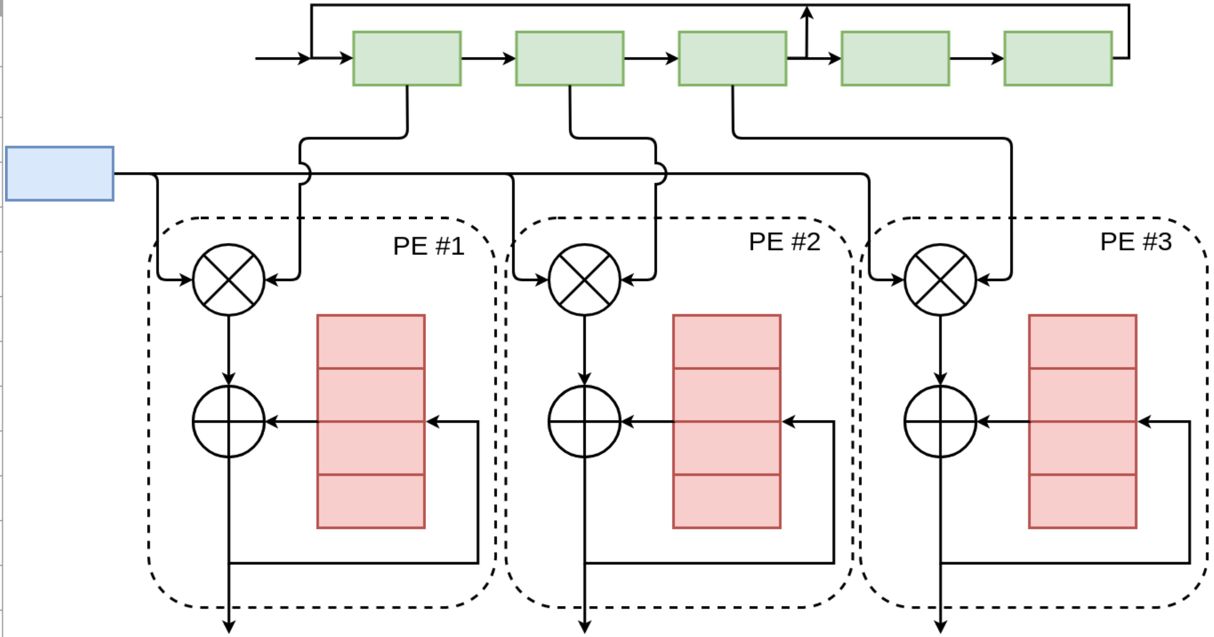
- Motivation:
  - Reduce the gap between the peak performance and run-time performance of state-of-the-art accelerators
  - Maximize arithmetic intensity

FEIE

- Reduces gap between peak and actual performance on a wide variety of models
- Maximize filter reuse
- Maximize image reuse
- First architecture that allows skipping noncontributory computations in edge computing
  - Uses very low memory bandwidth (e.g. 64-bits memory interface).

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3



- Exploit time division multiplexing to perform convolutions
- The width of weight vector and the stride denote the number of required PEs (which is 3 in this example)
- Inputs are shared among all PEs
- Weights are multiplexed and passed to each PE using shift registers

X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

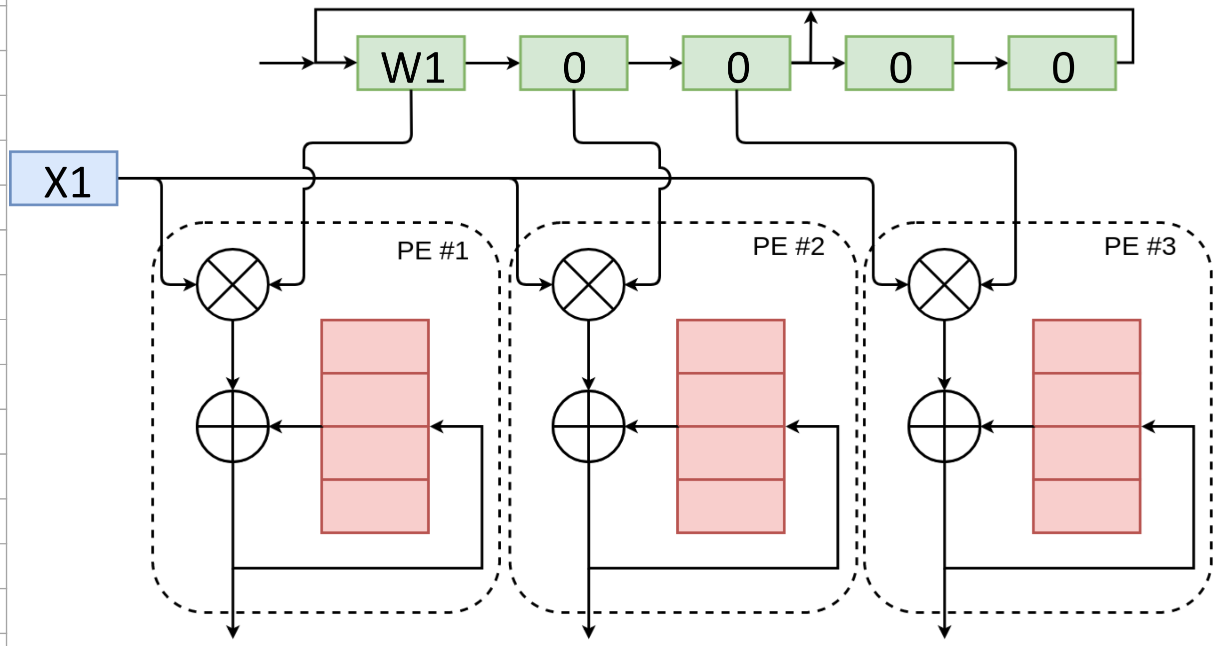
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3



- Input pixels are read sequentially

X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

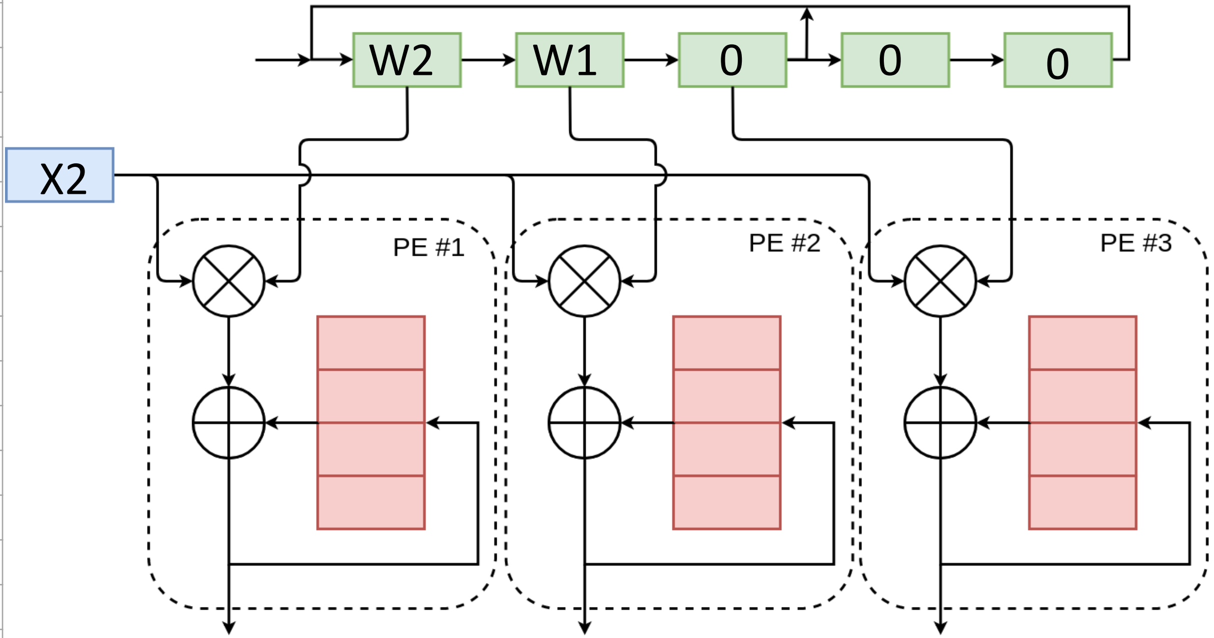
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

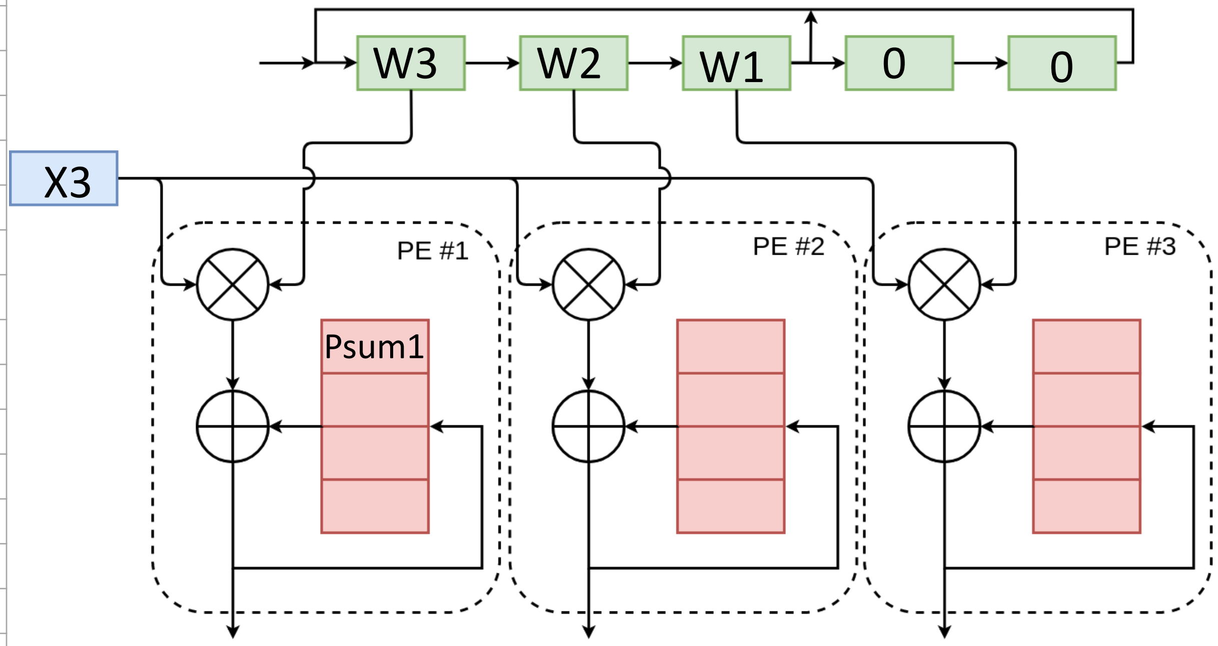
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1											



- The first partial sum is ready after 3 clock cycles.
- Next partial sums are generated after the third clock cycle in a pipelined manner.

X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

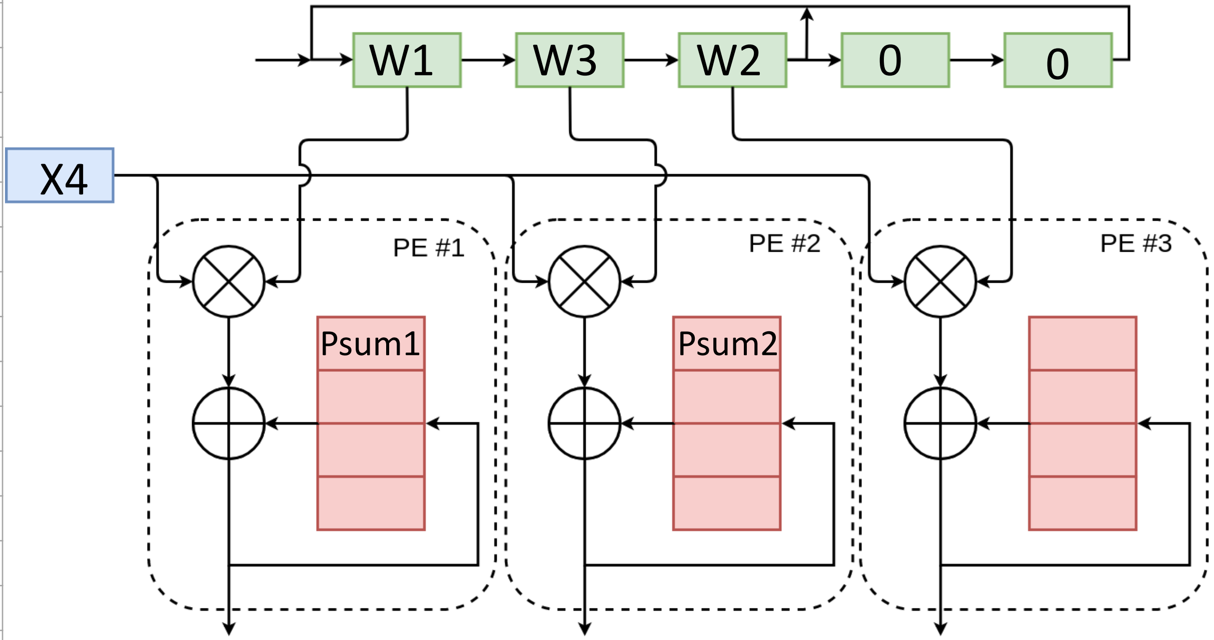
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2										



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

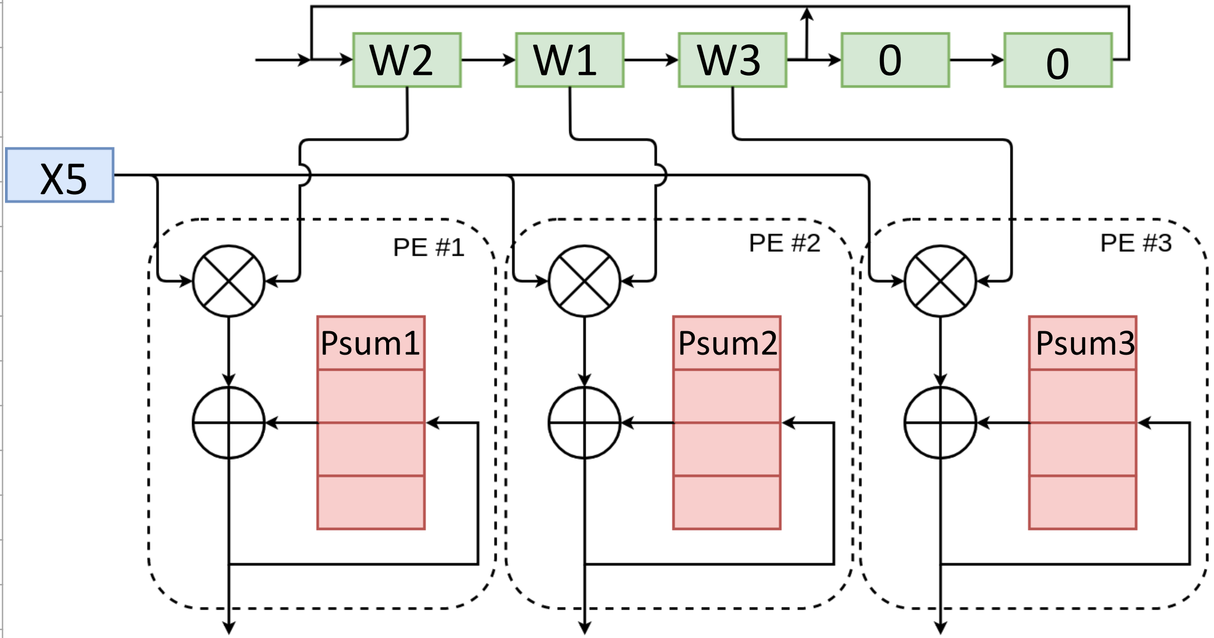
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3									



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

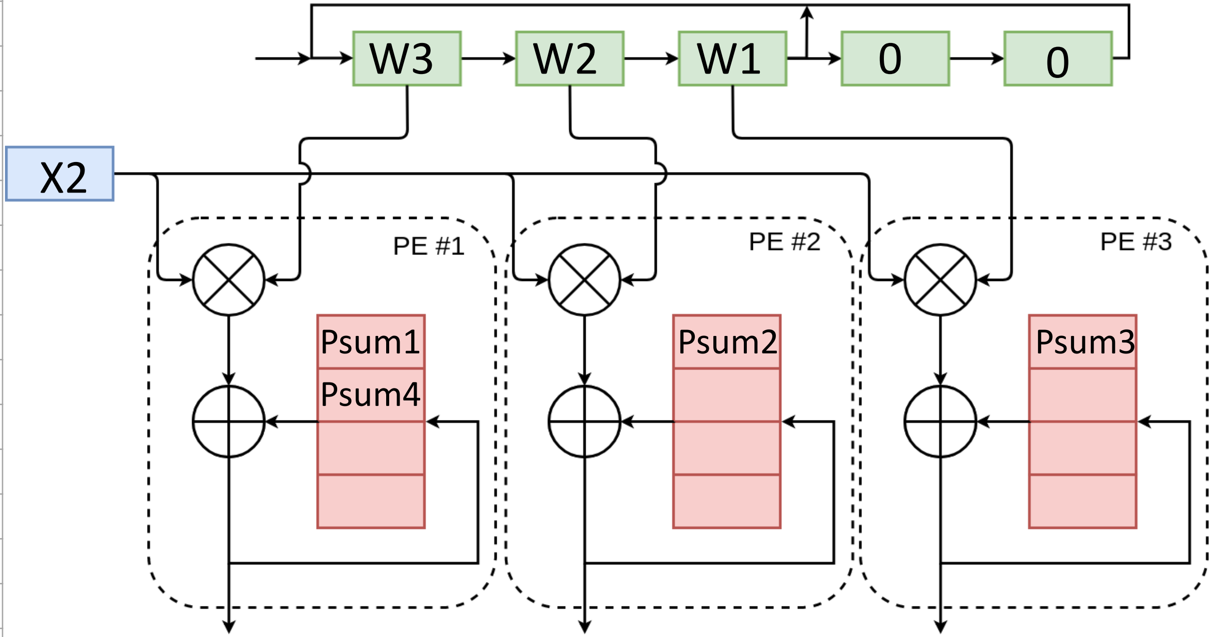
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4								



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

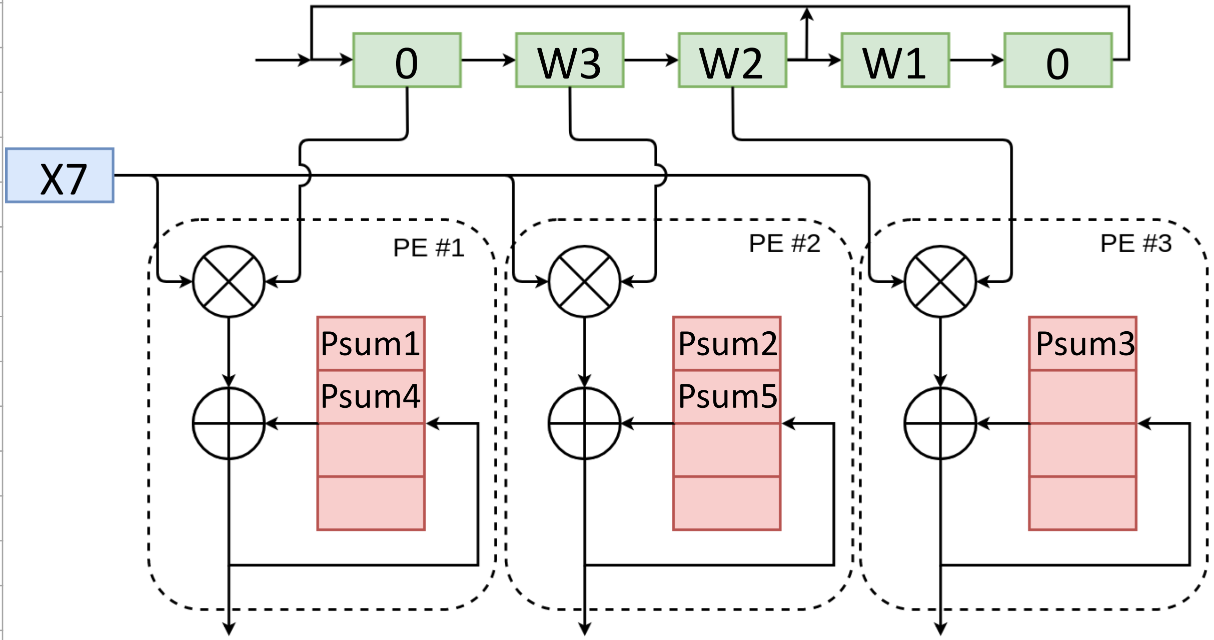
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5							



At this point, there will be two pipeline stalls to accommodate the input row change.

X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

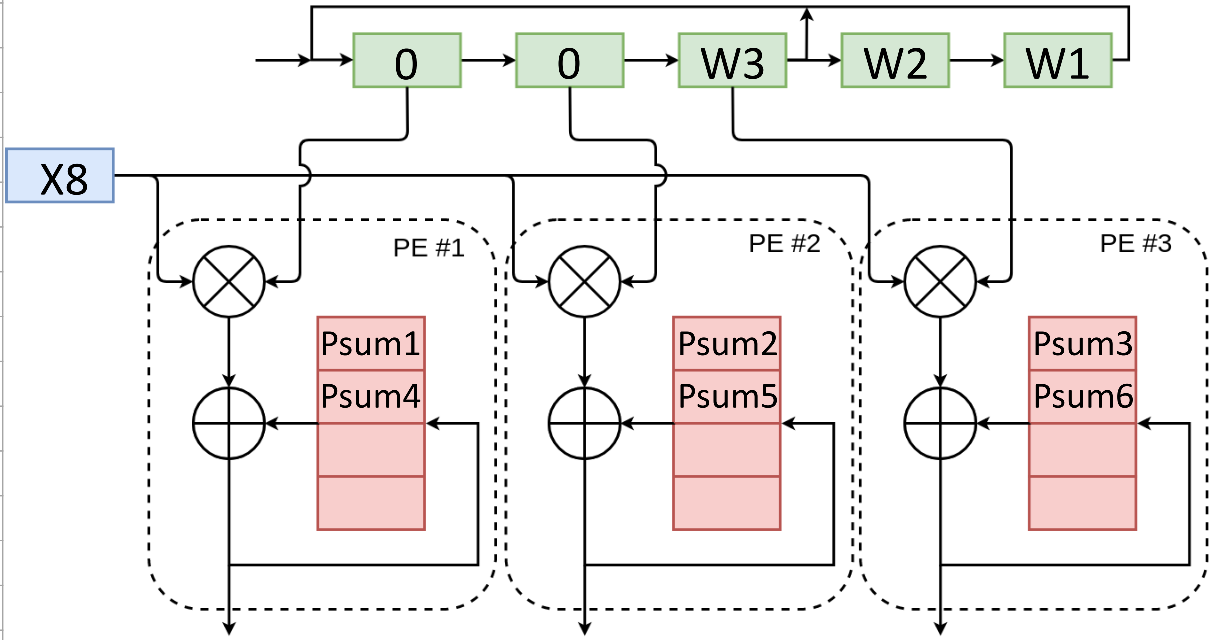
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6						



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

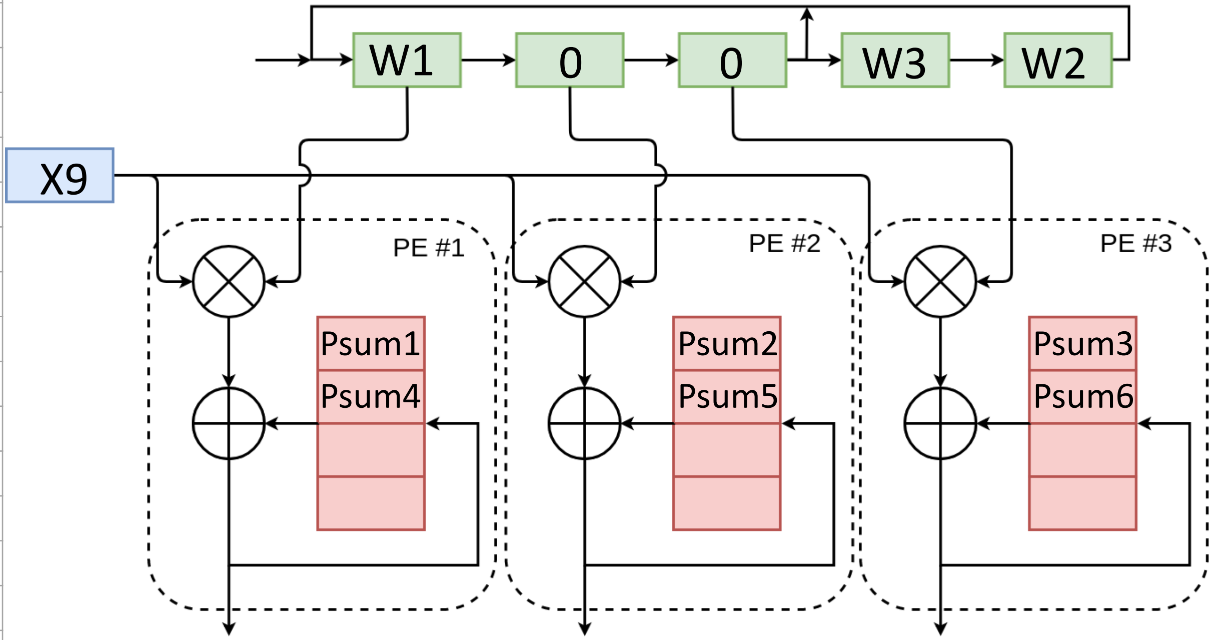
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6						



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

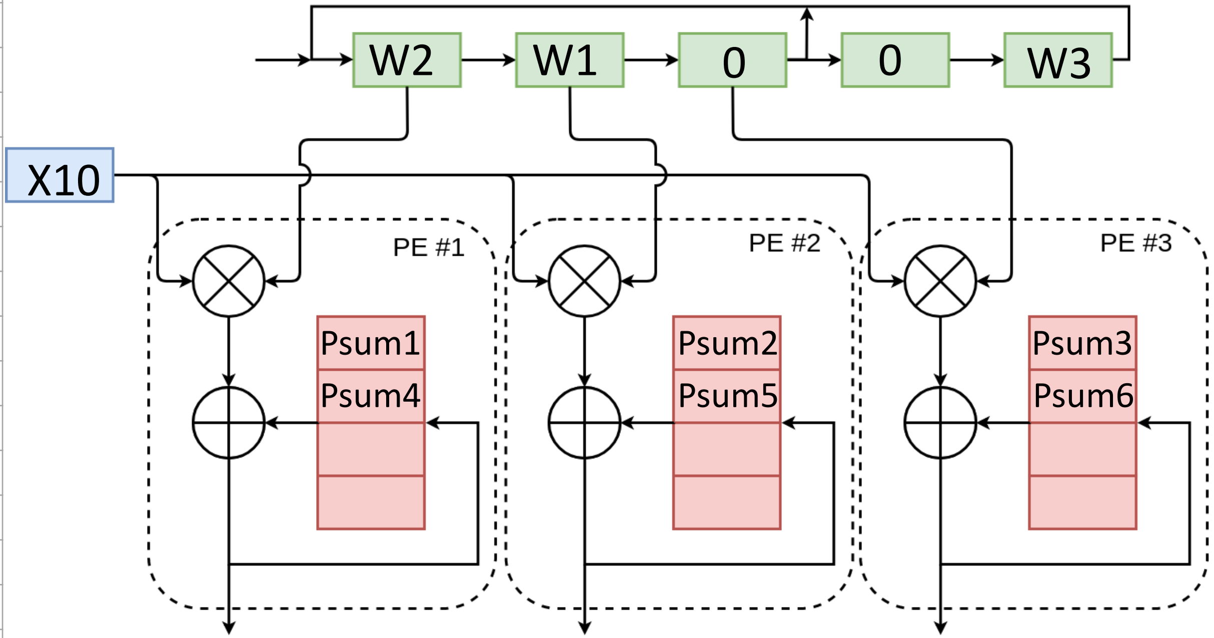
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6						



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

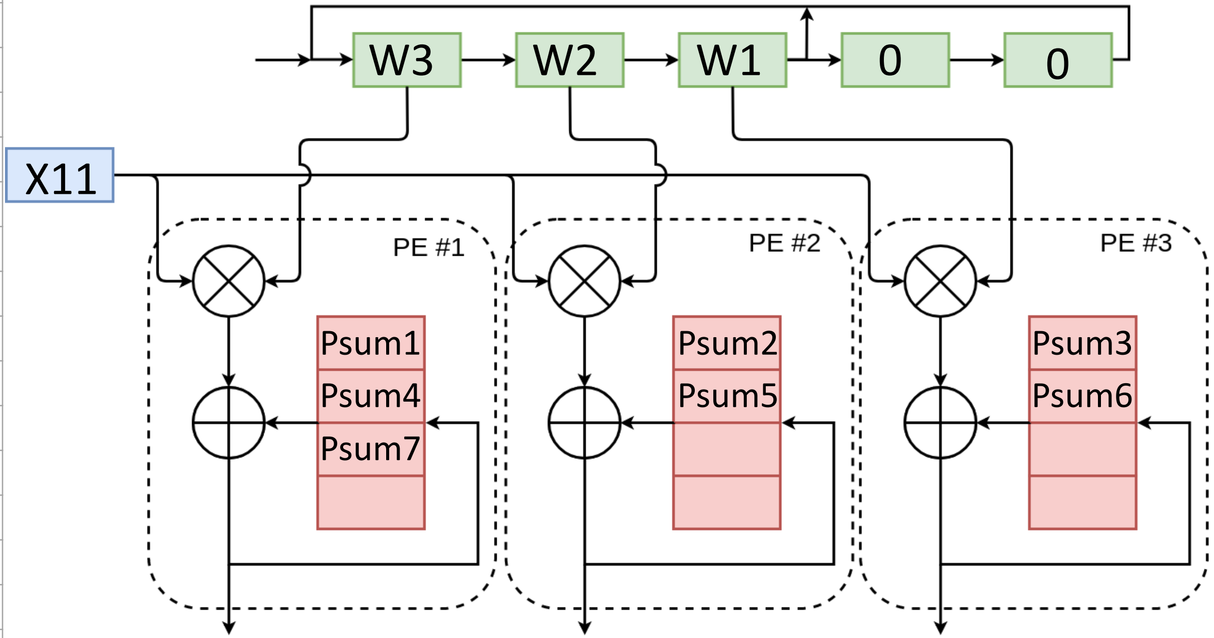
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6	Psum7					



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

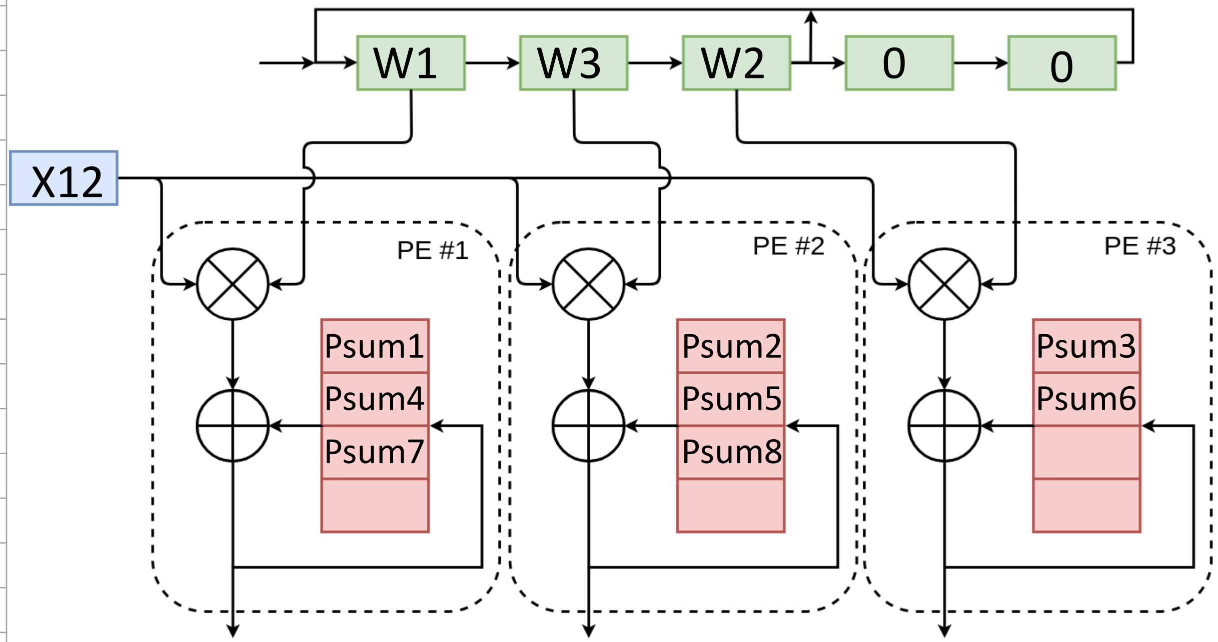
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map							2nd Row of Output Map				
		Psum							Psum				
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6	Psum7	Psum8				



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

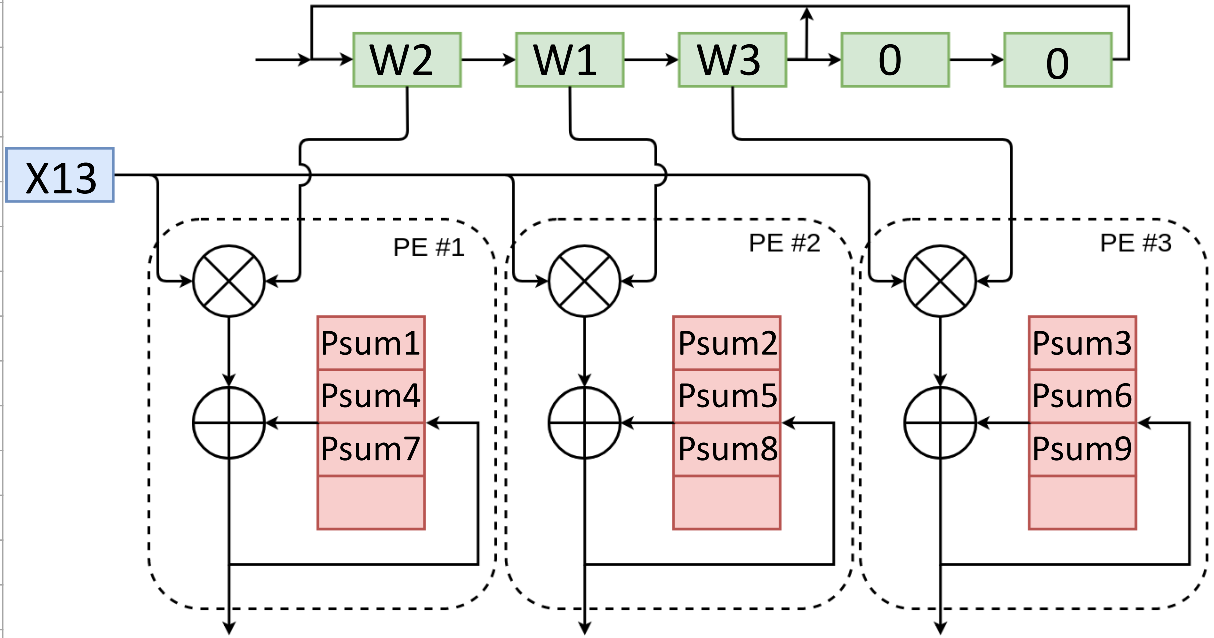
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6	Psum7	Psum8	Psum9			



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

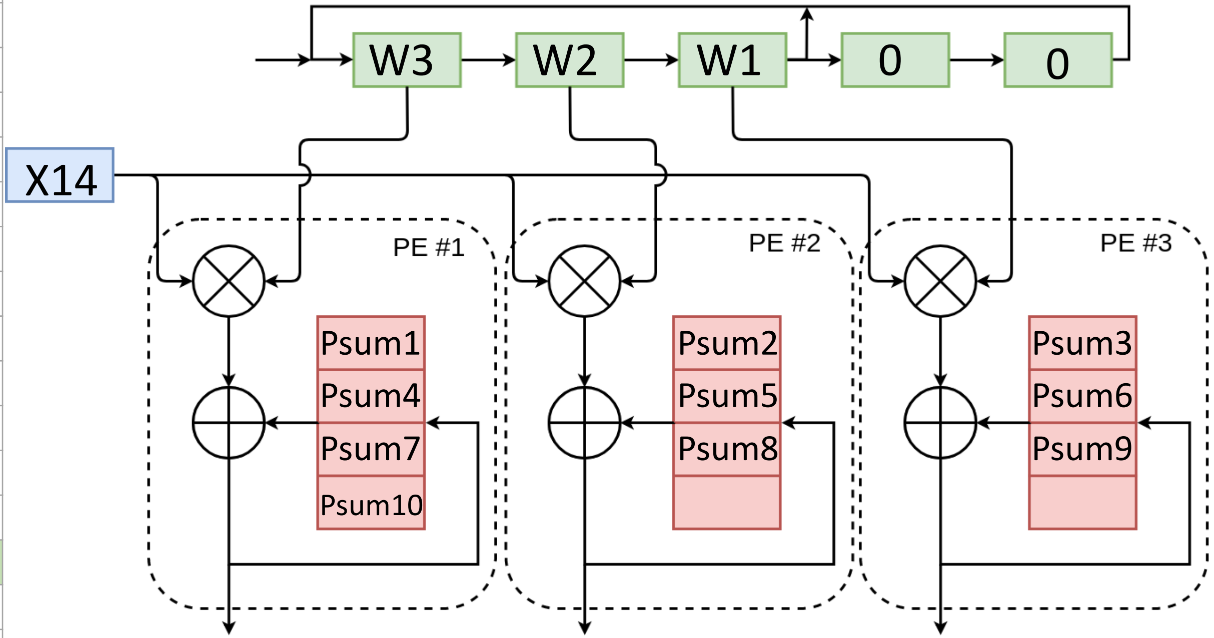
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6	Psum7	Psum8	Psum9	Psum10		



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

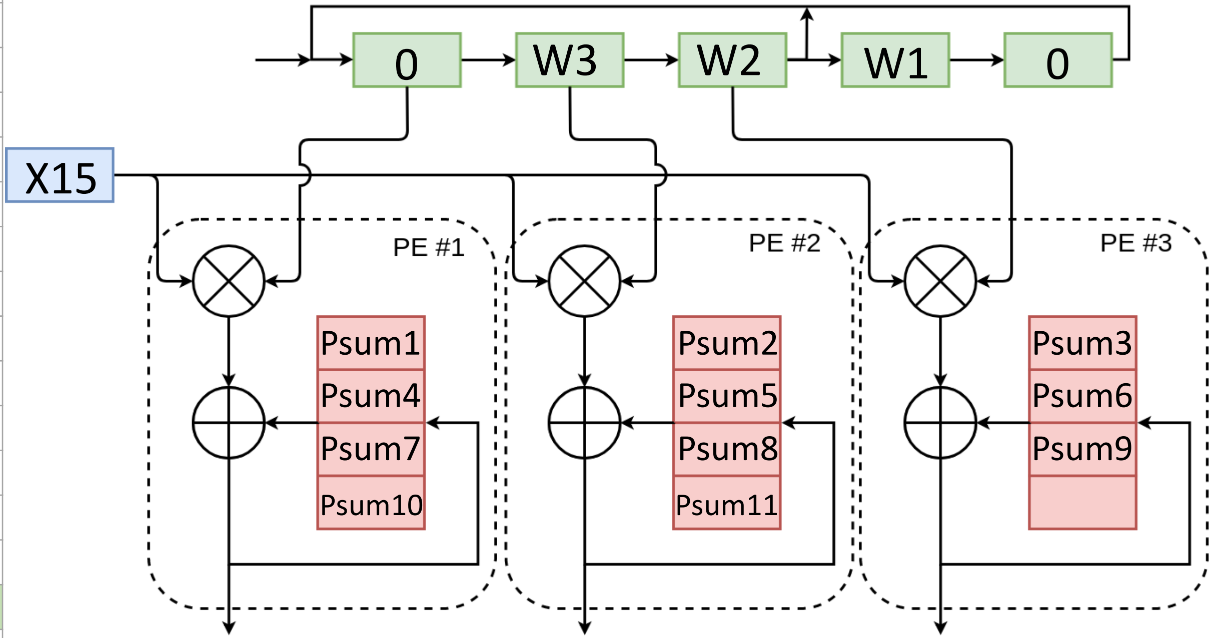
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6	Psum7	Psum8	Psum9	Psum10	Psum11	



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

W1	W2	W3

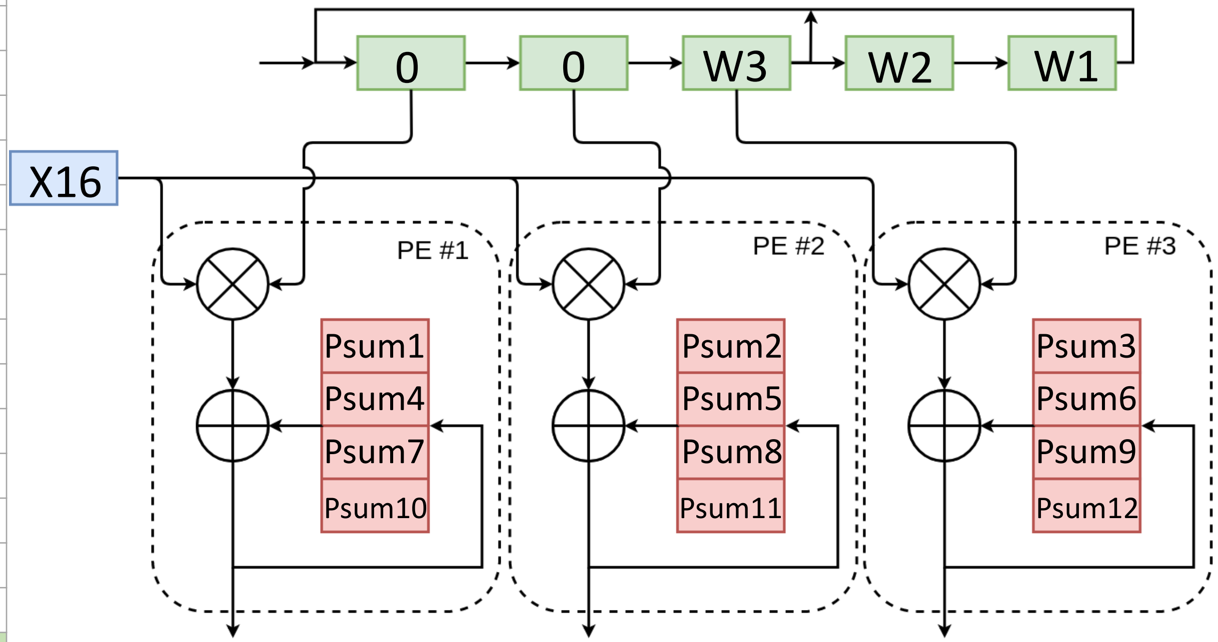
=

Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

Fast Efficient Inference Engine (FEIE)

		1st Row of Output Map						2nd Row of Output Map					
		Psum						Psum					
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	X6				W3	W2	W1						
CC #7	X7					W3	W2						
CC #8	X8						W3						
CC #9	X9							W1					
CC #10	X10							W2	W1				
CC #11	X11							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
		Psum1	Psum2	Psum3	Psum4	Psum5	Psum6	Psum7	Psum8	Psum9	Psum10	Psum11	Psum12



X1	X2	X3	X4	X5	X6	X7	X8
X9	X10	X11	X12	X13	X14	X15	X16

*

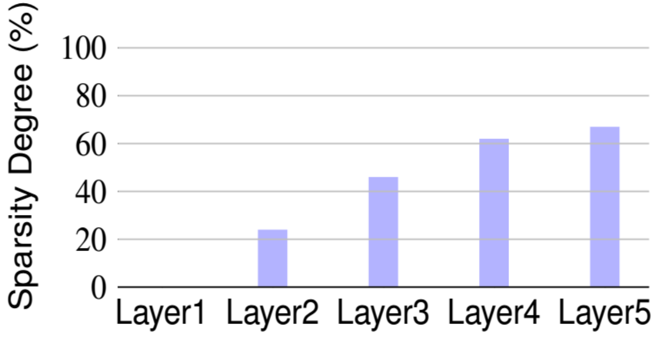
W1	W2	W3

=

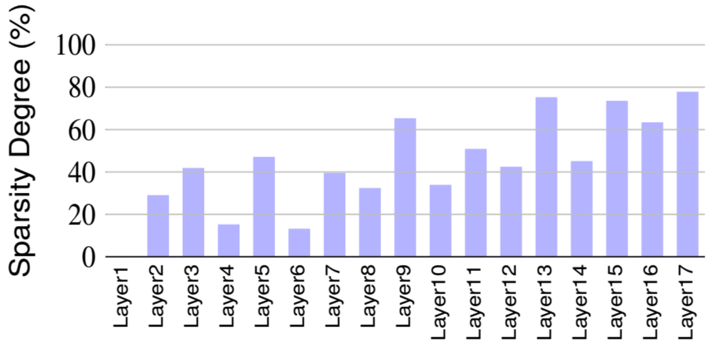
Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," IEEE TCAS I, 2018.

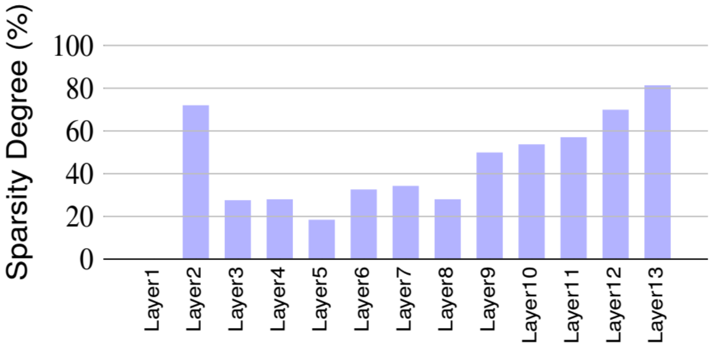
Sparsity in Activations



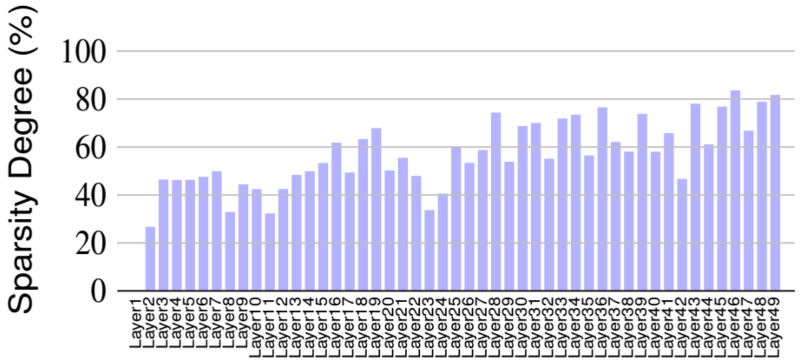
AlexNet



ResNet-18



VGGNet-16



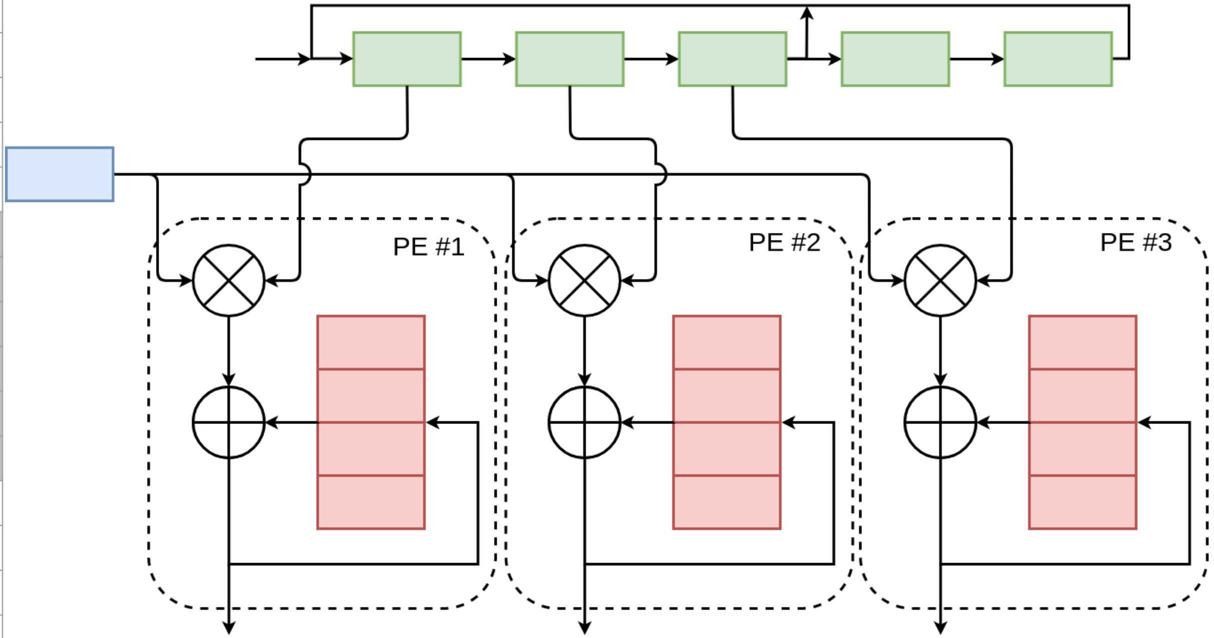
ResNet-50

- The use of ReLU as an activation function is a common choice in state-of-the-art CNNs
- ReLU layer lets positive values pass through while converting any negative input to zero.
- Avoiding the computations of zero-valued activations significantly speed up the process
- Examples of zero-skipping accelerators specialized for the inference computations in the cloud (large memory bandwidth):
  - Cnvlutin[1]
  - SCNN[2]

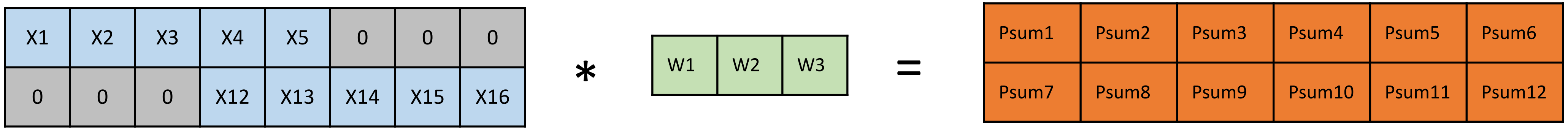
[1] J. Albericio et al., "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing," *ISCA,* 2016.
[2] A. Parashar et al., "SCNN: An accelerator for compressed-sparse convolutional neural networks," ISCA 2017.

Skipping Noncontributory Computations in FEIE

				1st Row of Output Map						2nd Row of Output Map			
				Psum						Psum			
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
CC #1	X1	W1											
CC #2	X2	W2	W1										
CC #3	X3	W3	W2	W1									
CC #4	X4		W3	W2	W1								
CC #5	X5			W3	W2	W1							
CC #6	0				W3	W2	W1						
CC #7	0					W3	W2						
CC #8	0						W3						
CC #9	0							W1					
CC #10	0							W2	W1				
CC #11	0							W3	W2	W1			
CC #12	X12								W3	W2	W1		
CC #13	X13									W3	W2	W1	
CC #14	X14										W3	W2	W1
CC #15	X15											W3	W2
CC #16	X16												W3
							0	0					



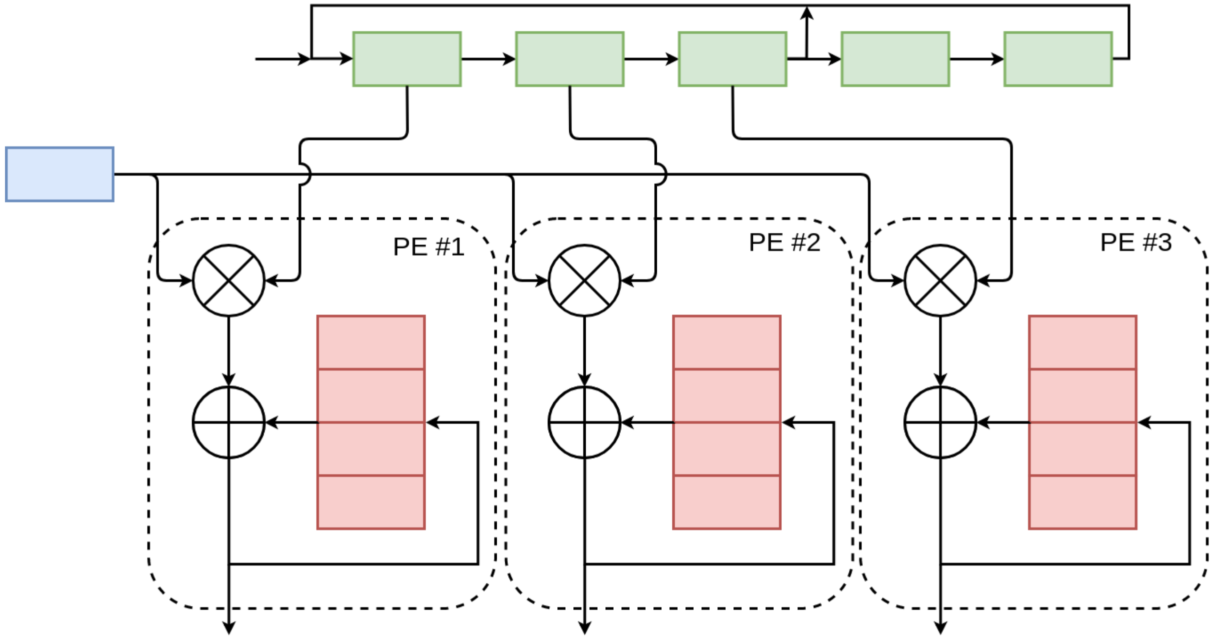
Performing computations on dense model requires 16 clock cycles!



A. Ardakani, C. Condo and W. J. Gross, "Fast and Efficient Convolutional Accelerator for Edge Computing," in *IEEE Transactions on Computers, 2019*.

Skipping Noncontributory Computations in FEIE

		1st Row of Output Map					2nd Row of Output Map				
		Psum									
Clcok Cycles	Inputs	#1	#2	#3	#4	#5	#8	#9	#10	#11	#12
CC #1	X1	W1									
CC #2	X2	W2	W1								
CC #3	X3	W3	W2	W1							
CC #4	X4		W3	W2	W1						
CC #5	X5			W3	W2	W1					
CC #6	X12						W3	W2	W1		
CC #7	X13							W3	W2	W1	
CC #8	X14								W3	W2	W1
CC #9	X15									W3	W2
CC #10	X16										W3



Performing computations on dense model requires 10 clock cycles only!

X1	X2	X3	X4	X5	0	0	0
0	0	0	X12	X13	X14	X15	X16

*

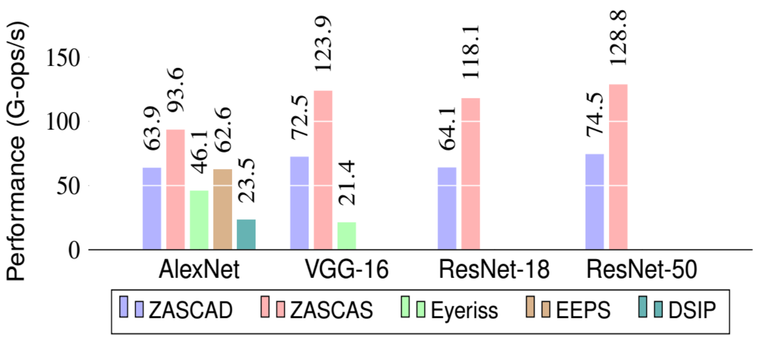
W1	W2	W3

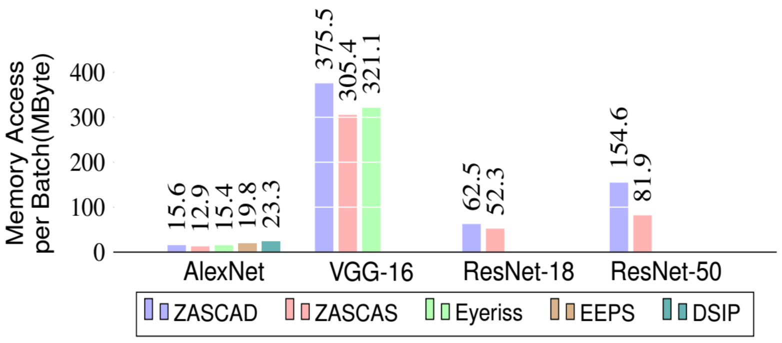
=

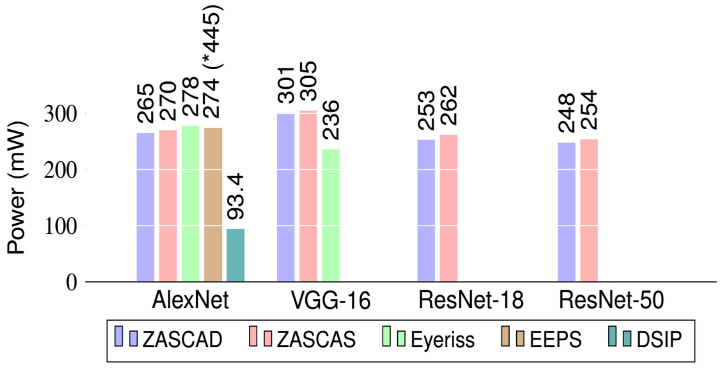
Psum1	Psum2	Psum3	Psum4	Psum5	Psum6
Psum7	Psum8	Psum9	Psum10	Psum11	Psum12

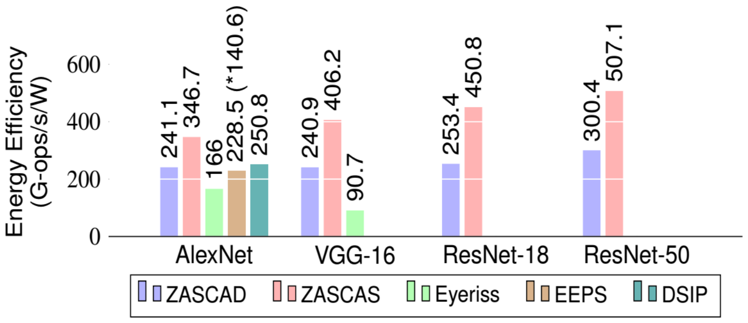
A. Ardakani, C. Condo and W. J. Gross, "Fast and Efficient Convolutional Accelerator for Edge Computing," in *IEEE Transactions on Computers, 2019*.

Performance of Convolutional Accelerators for Edge Computing





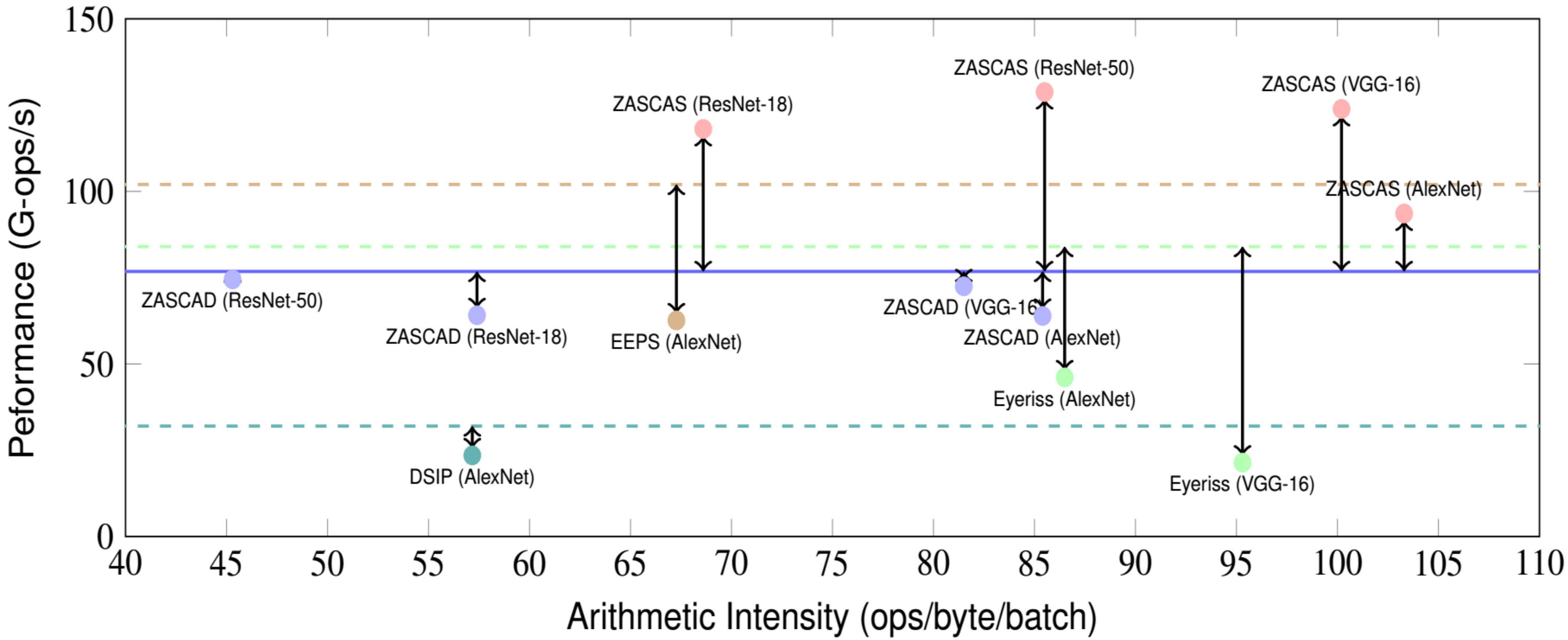




- Examples of accelerators specialized for the inference computations at the edge:
  - Eyeriss[1]
  - ZASCAS (zero-skipping FEIE)[2]
  - ZASCAD (non-zero-skipping FEIE)[2]
  - DSIP[3]
  - EEPS[4]
- Among all accelerators, ZASCAS significantly stands out in terms of performance, energy efficiency and memory accesses.

[1] Y. Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ISCA*, 2016.
[2] A. Ardakani, C. Condo and W. J. Gross, "Fast and Efficient Convolutional Accelerator for Edge Computing," in *IEEE Transactions on Computers, 2019*.
[3] B. Moons et al., “An energy-efficient precision-scalable convnet processor in a 40-nm CMOS,” IEEE Journal of Solid-State Circuits, 2016.
[4] J. Jo et al., “DSIP: A scalable inference accelerator for convolutional neural networks,” IEEE Journal of Solid-State Circuits, 2018.

Roofline Model of Convolutional Accelerators



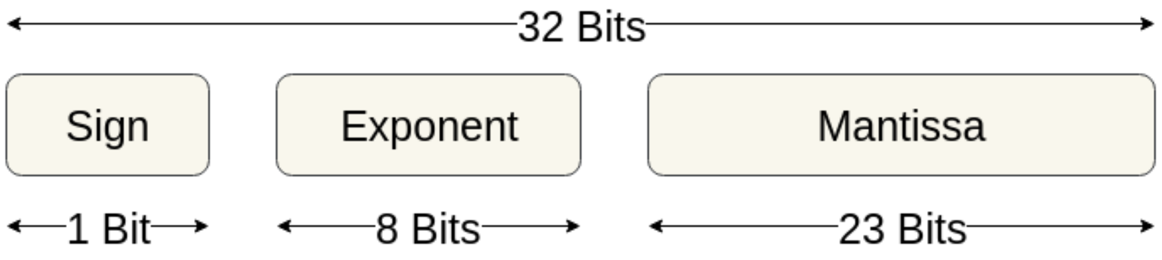
- In terms of arithmetic intensity, ZASCAS on AlexNet performs more operations per byte among all accelerators.
- In terms of performance, ZASCAS on ResNet-50 performs more operations regardless of memory accesses among all accelerators.

Model Compression

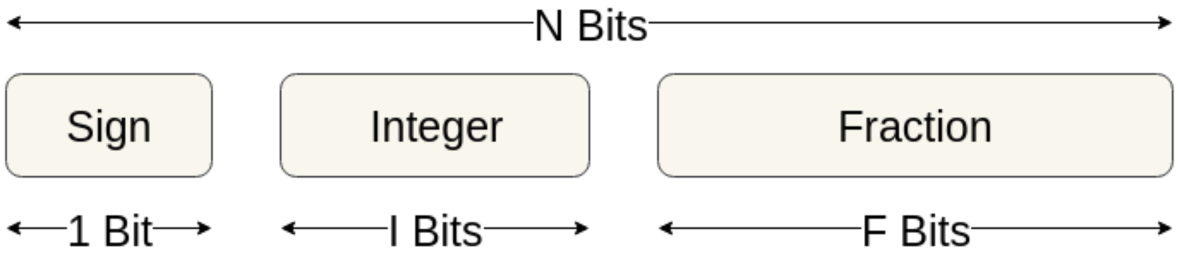
- Quantization
  - Reduce bitwidth of weights and activations
  - Simpler computational logic
  - Reduce memory footprint
- Pruning
  - Reduce number of operations
  - Reduce memory footprint

Quantization

- Full-precision representation:



- Fixed-point representation:



- To reduce the total number of bits for representation of weights and activations
  - Compression rate: 32/N
- Representing weights/activations by their sign values (i.e., N = 1) results in compression rate of 32x.

A General Quantization Method

- During training phase:

Represent **w** and **x** vectors using N bits in fixed-point format → Perform forward computations of **y** = Σ **wx** with quantized vectors → Perform Backward computations in full-precision format

During inference phase:

Perform forward computations of **y** = Σ **wx** with quantized vectors

M. Courbariaux et al., "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," NIPS 2015.

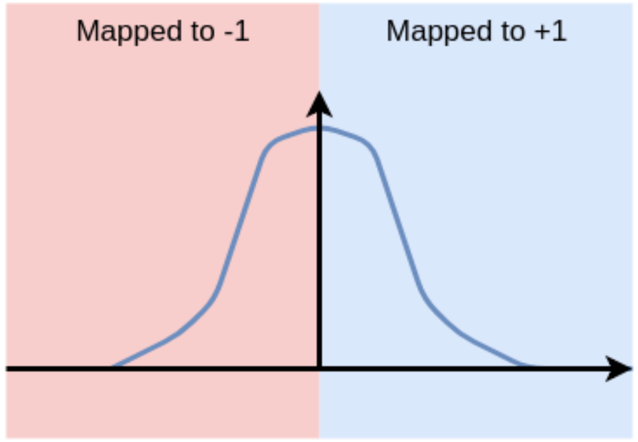
Binarization and Ternarization of CNNs/FCNs

Binary Networks

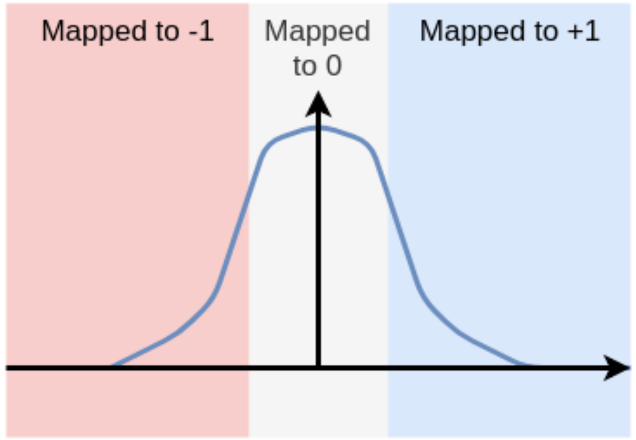
Use two possible values for weights: either +1 or -1

Replace multiplications with simple XNOR operations

Fixed-point adders are also much less expensive both in terms of area and energy



Ternary Networks



Use three possible values for weights: either +1, 0, or -1

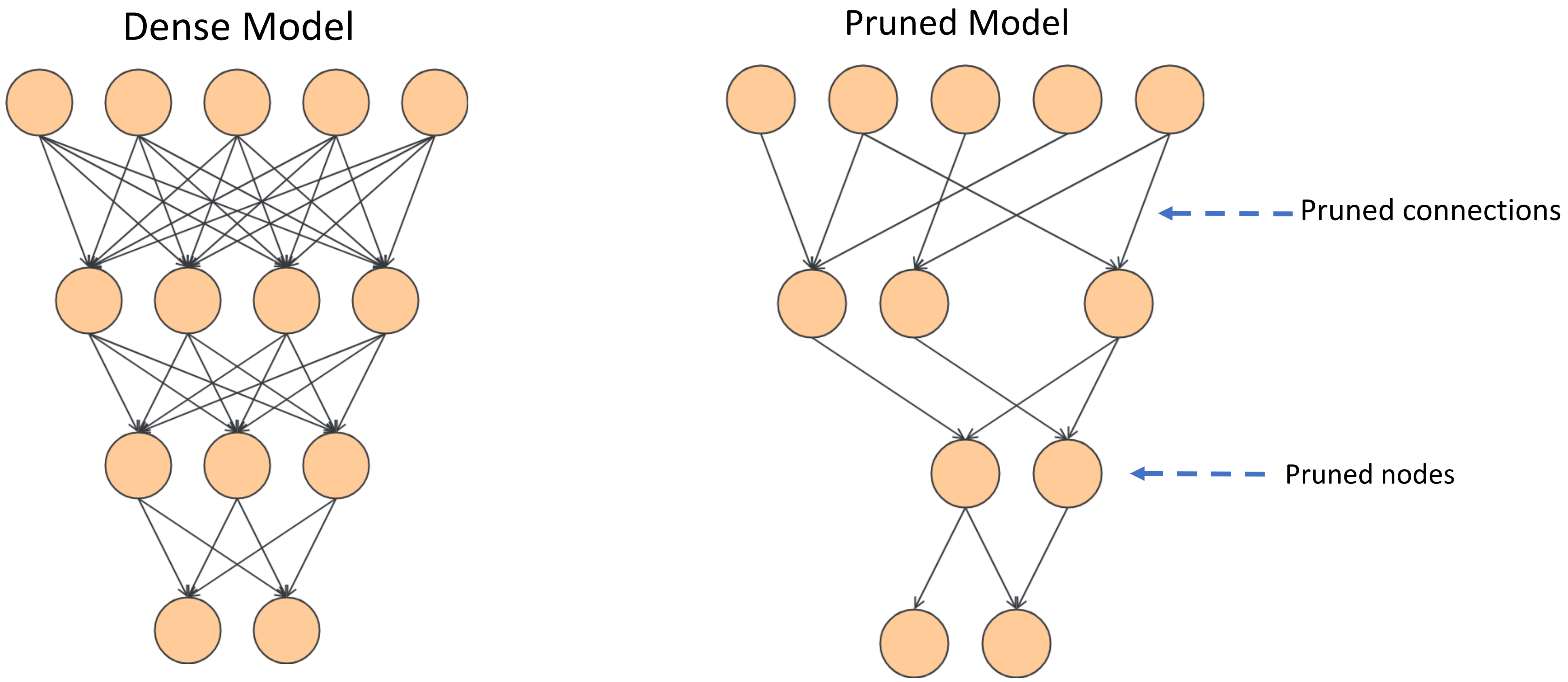
Replace multipliers with simple multiplexers

Fixed-point adders are also much less expensive both in terms of area and energy

Ternary networks are usually more accurate than binary networks

Z. Lin et al., "Neural Networks with Few Multiplications," ICLR, 2016.

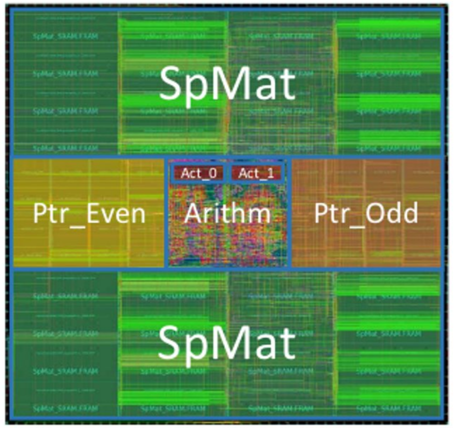
Model Pruning



- Pruning connections
  - Connections with weight value close to zero can be removed from the network.
- Pruning nodes
  - Activations with values close to zero can be removed.

S. Han et al., "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," NIPS 2015.

Pruning Method



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

Train fully connected model

Prune the weights of which magnitudes are less than a threshold

Retrain the network

S. Han et al., "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," NIPS, 2015.
S. Han et al., "EIE: efficient inference engine on compressed deep neural network," ISCA, 2016.