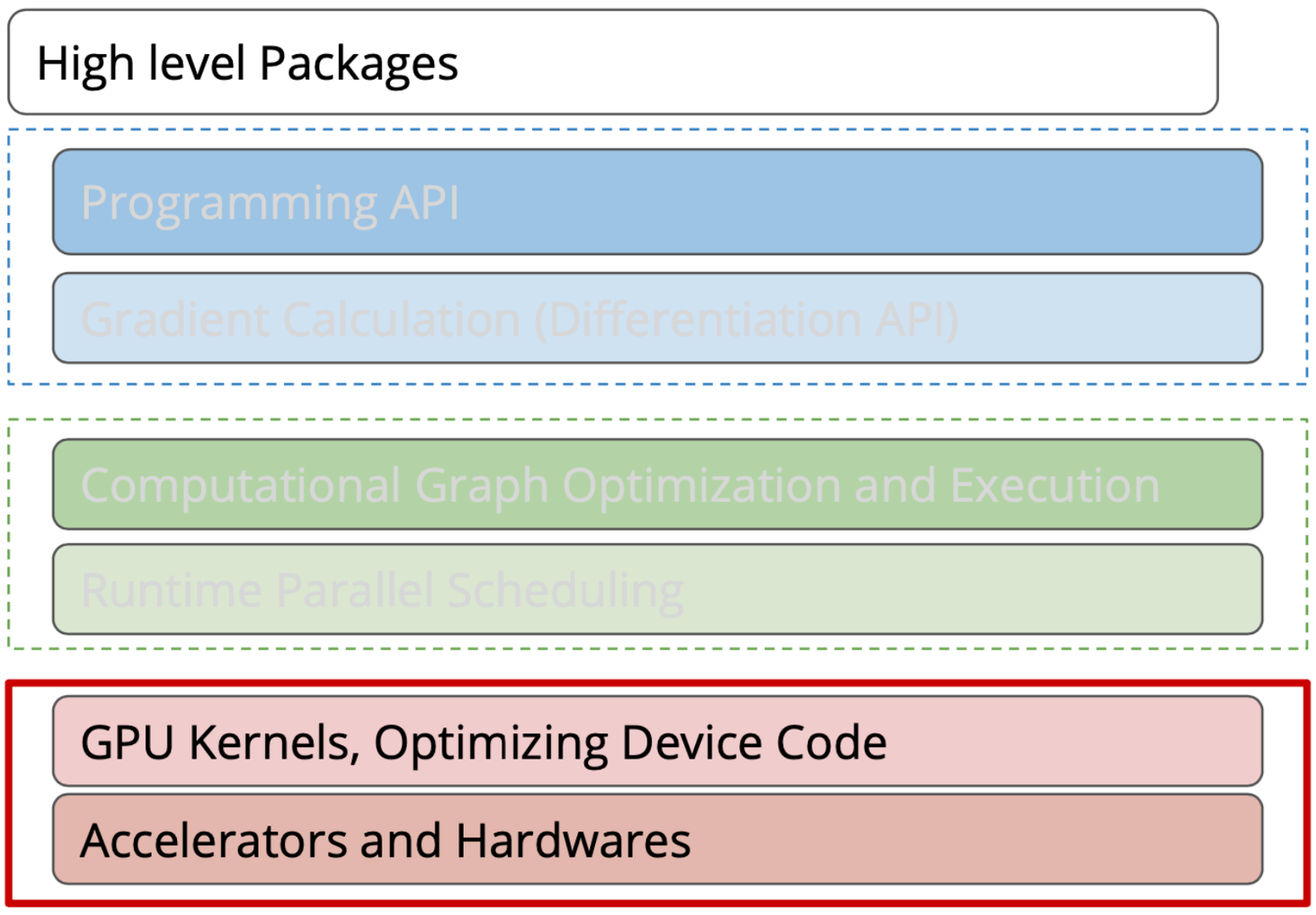
High-Performance Hardware for Deep Learning

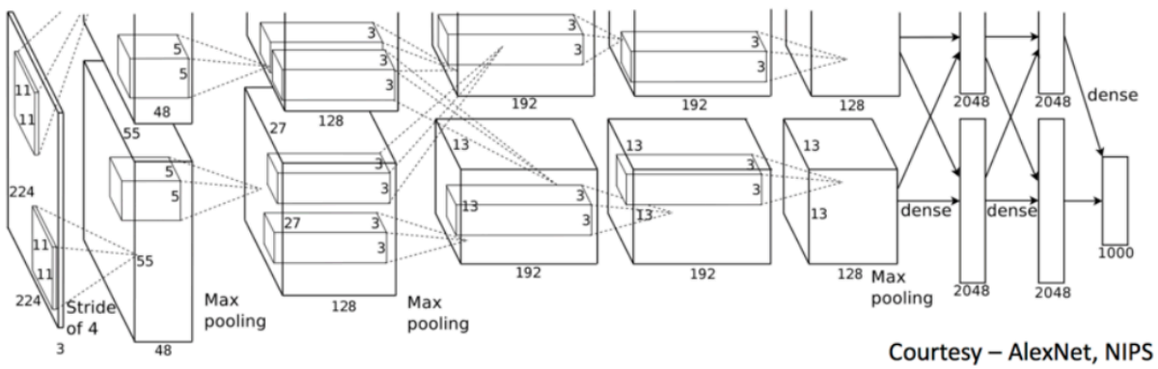
Pooyan Jamshidi
UofSC

Typical Deep Learning System Stack



Architecture

Hardware and Data enable DNNs



Courtesy – AlexNet, NIPS 2012

The need for Speed

- Larger data sets and models lead to better accuracy but also increase computation time. Therefore progress in deep neural networks is limited by how fast the networks can be computed.
- Likewise the application of convnets to low latency inference problems, such as pedestrian detection in self driving car video imagery, is limited by how fast a small set of images, possibly a single image, can be classified.

More data → Bigger Models →More Need for Compute

But Moore's law is no longer providing more compute…

Lavin & Gray, Fast Algorithms for Convolutional Neural Networks, 2015

Problem

Acceleration

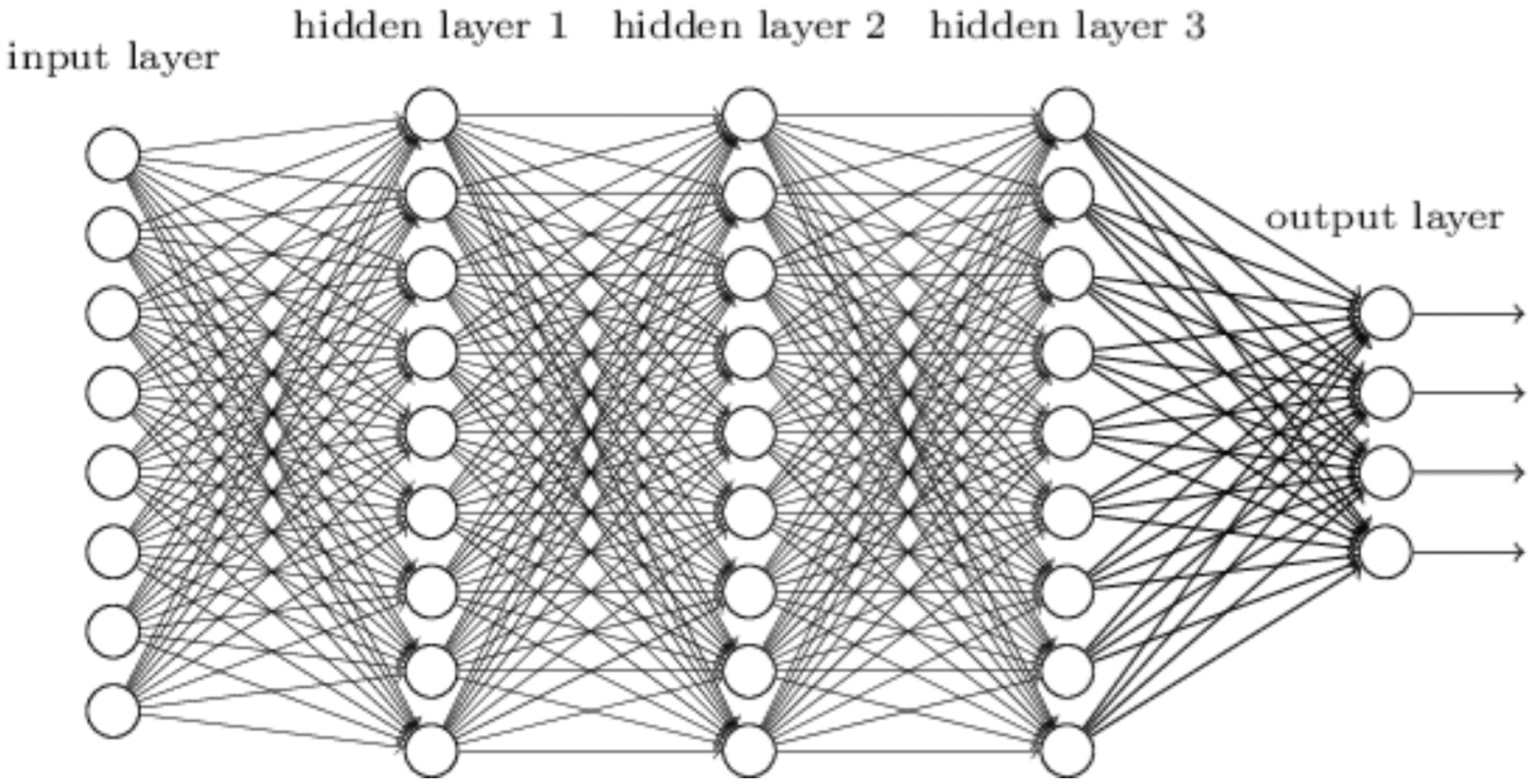
- Run a network faster (Performance, inf/s)

- Run a network more efficiently
  - Energy (inf/J)
  - Cost (inf/s$)

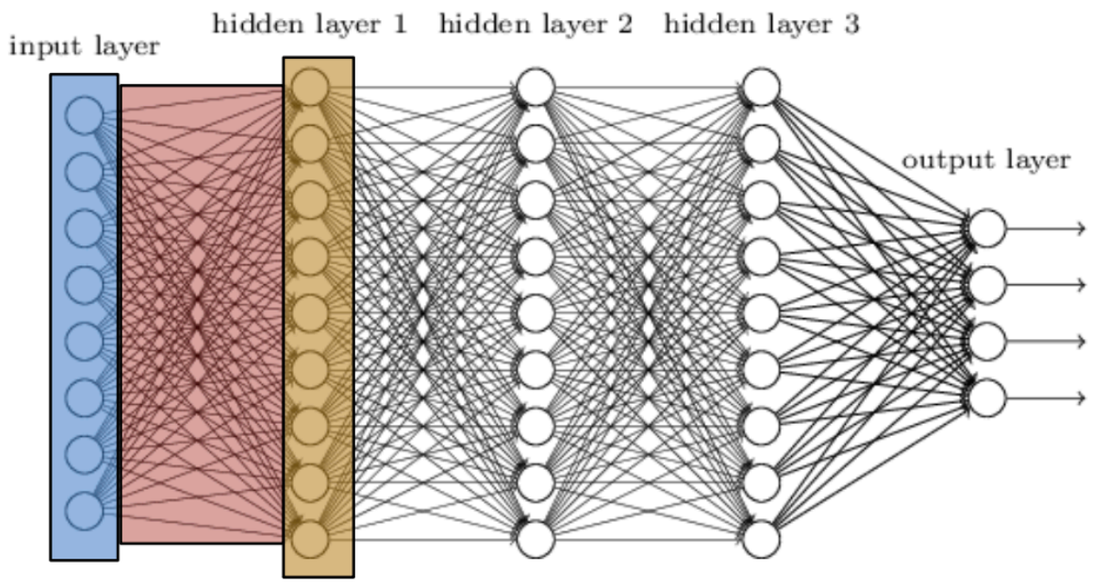
- Inference
  - Just running the network forward

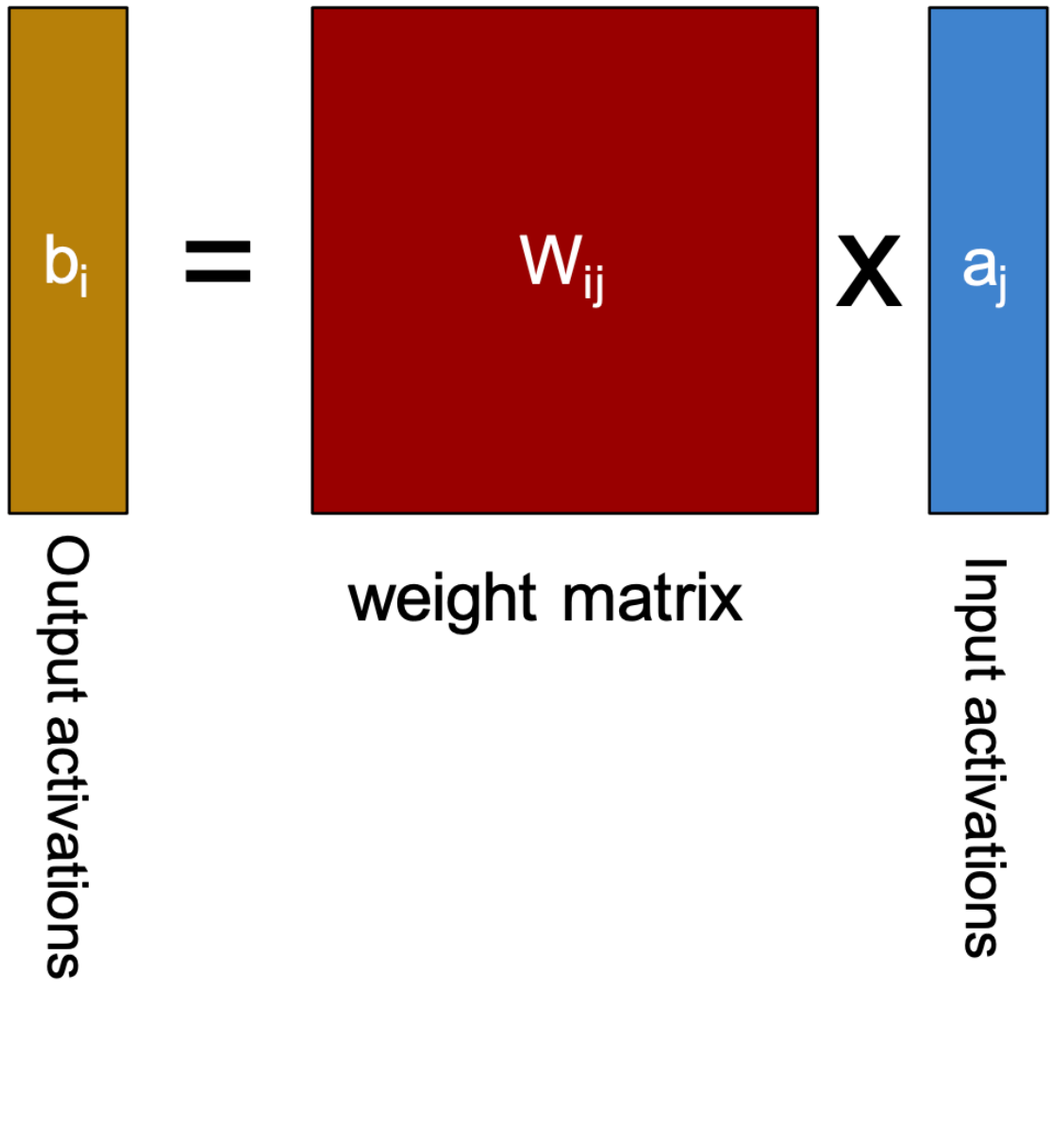
- Training
  - Running the network forward
  - Back-propagation of gradient
  - Update of parameters

What Network? DNNs, CNNs, and RNNs

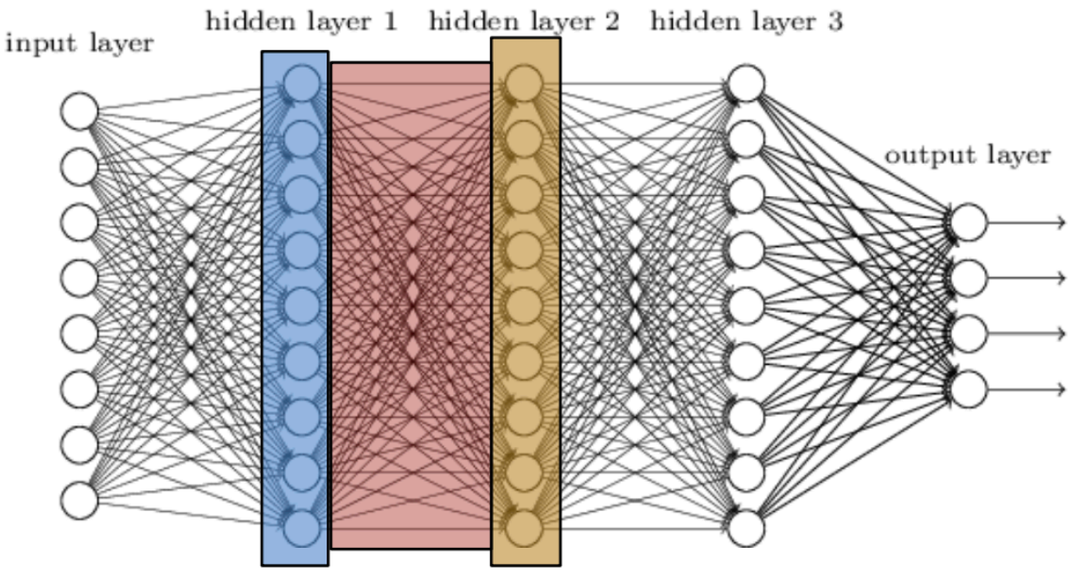


DNN, key operation is dense M x V

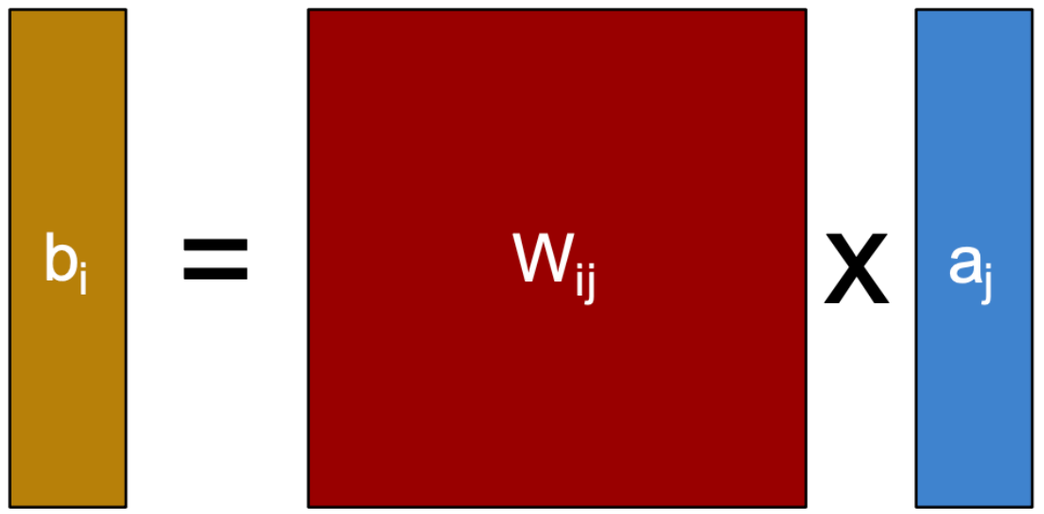




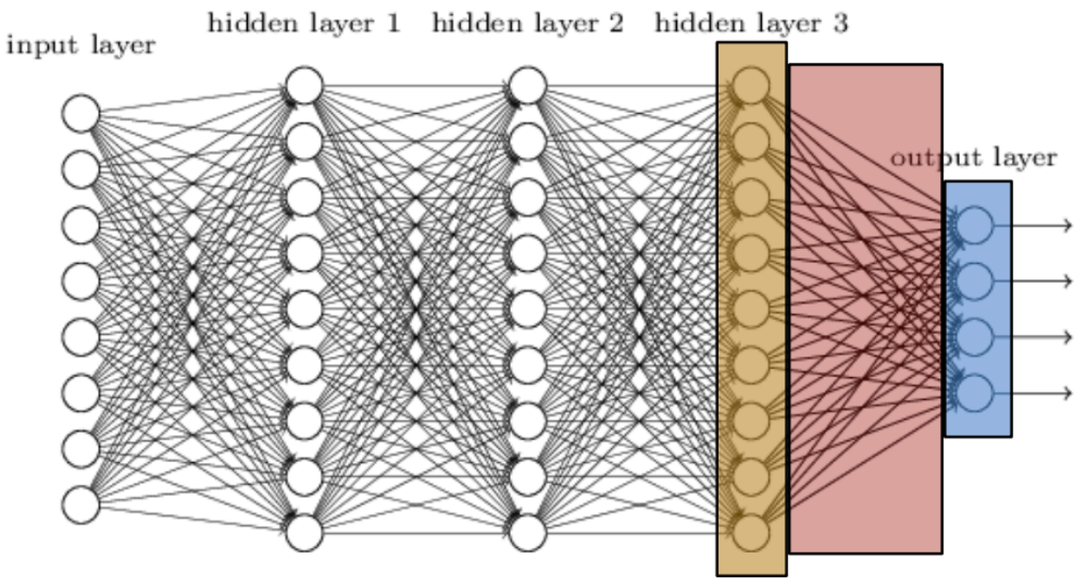
DNN, key operation is dense M x V



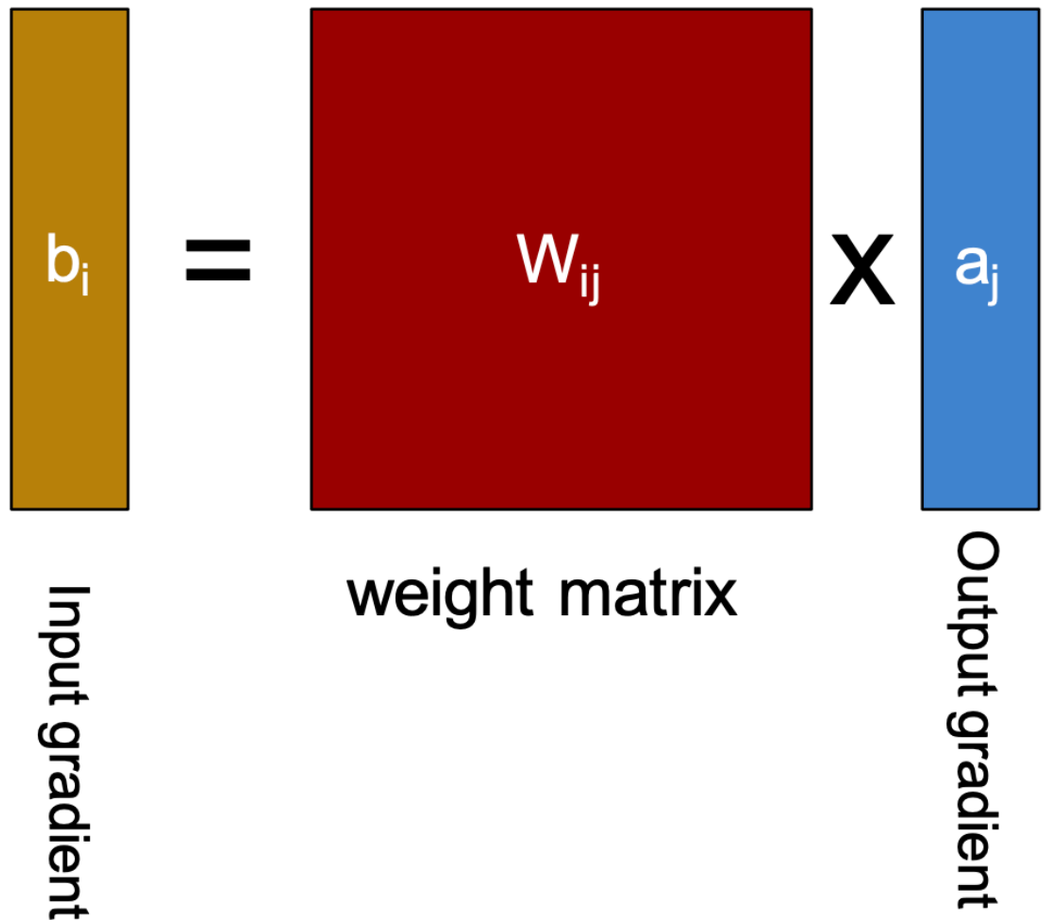
Repeat for each layer



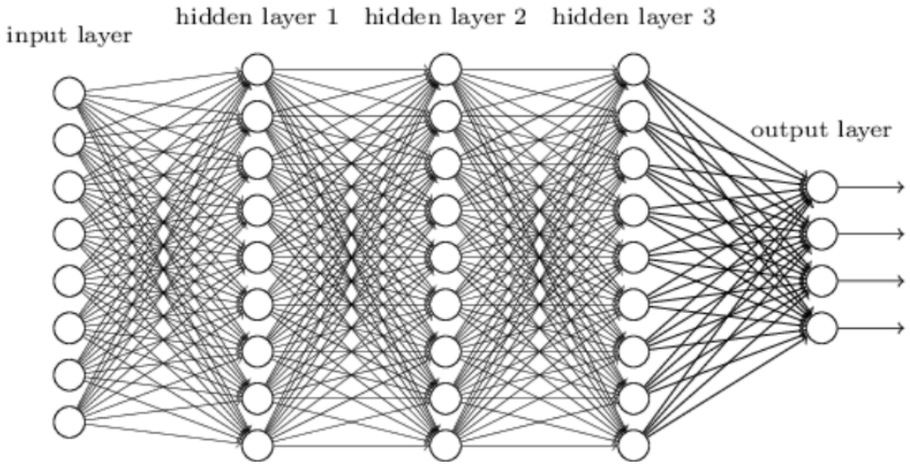
DNN, key operation is dense M x V

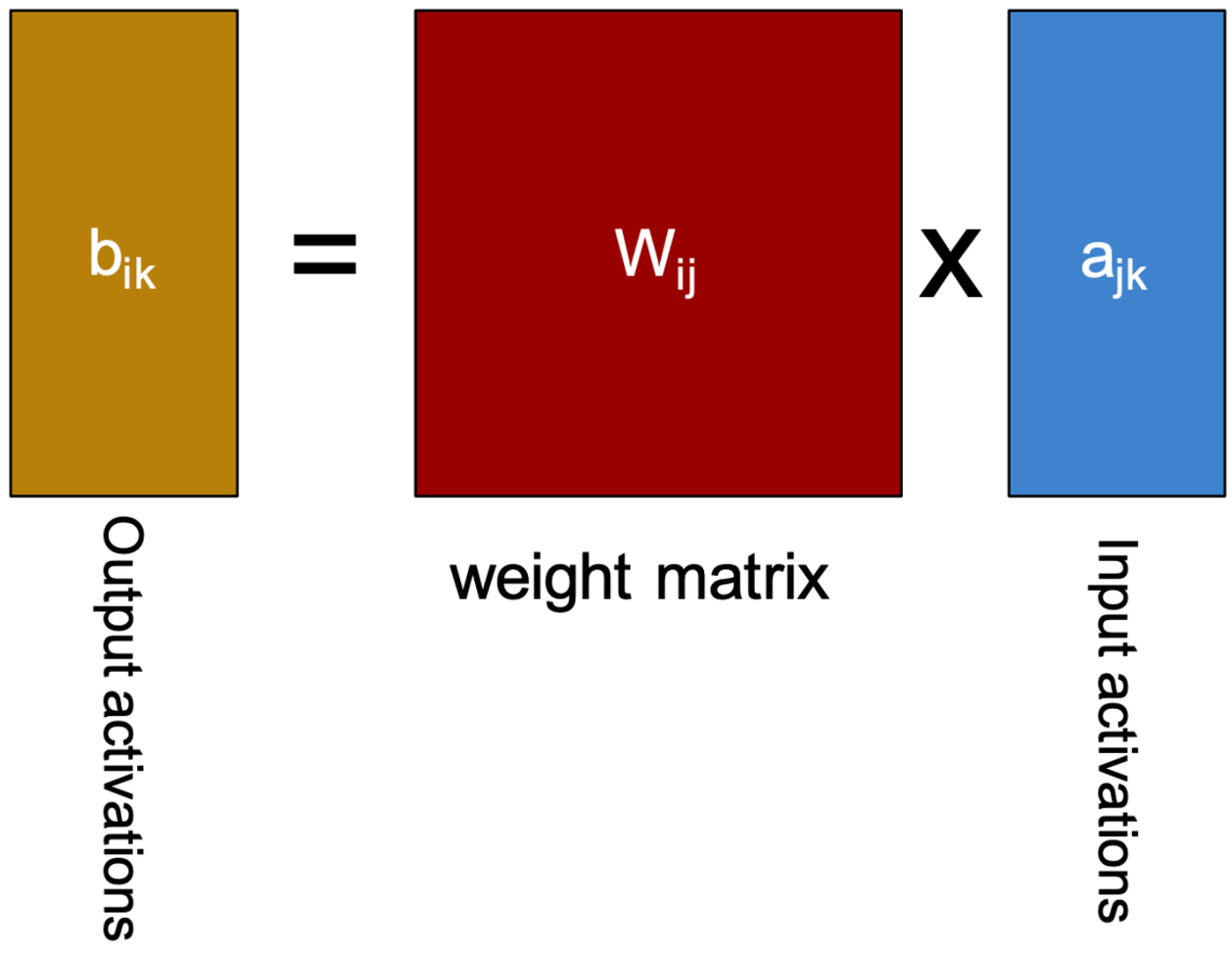


Backpropagation just does this backward

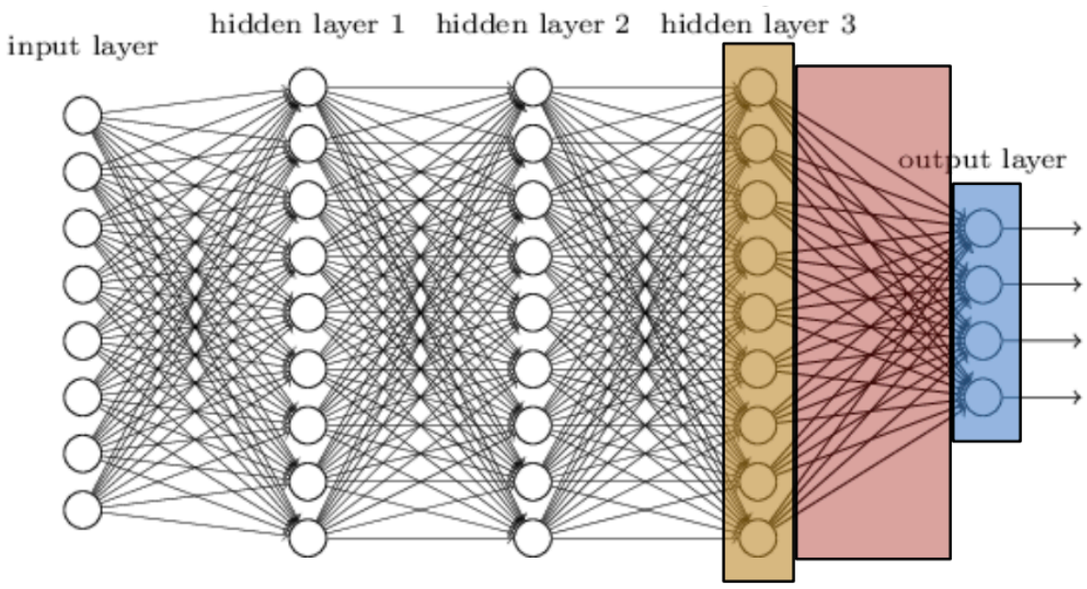


Training, and Latency Insensitive Networks can be Batched – operation is M x M – gives re-use of weights

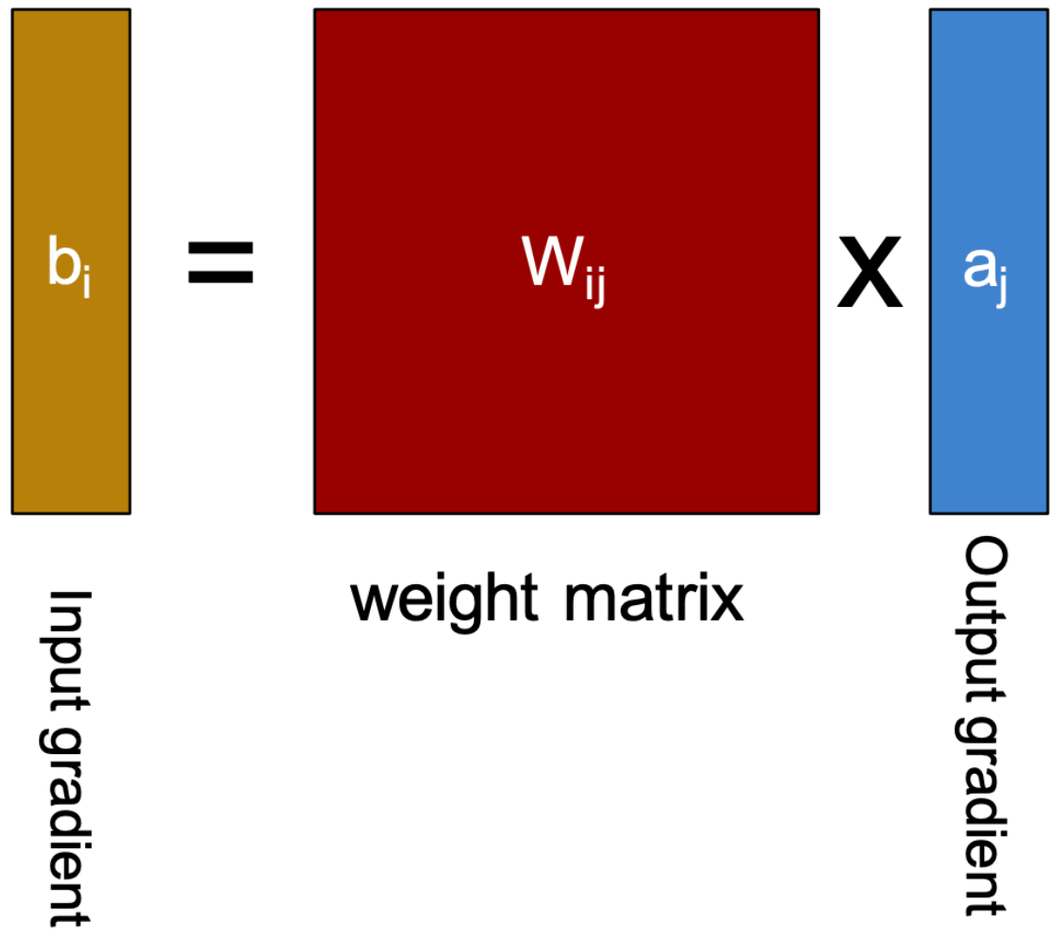




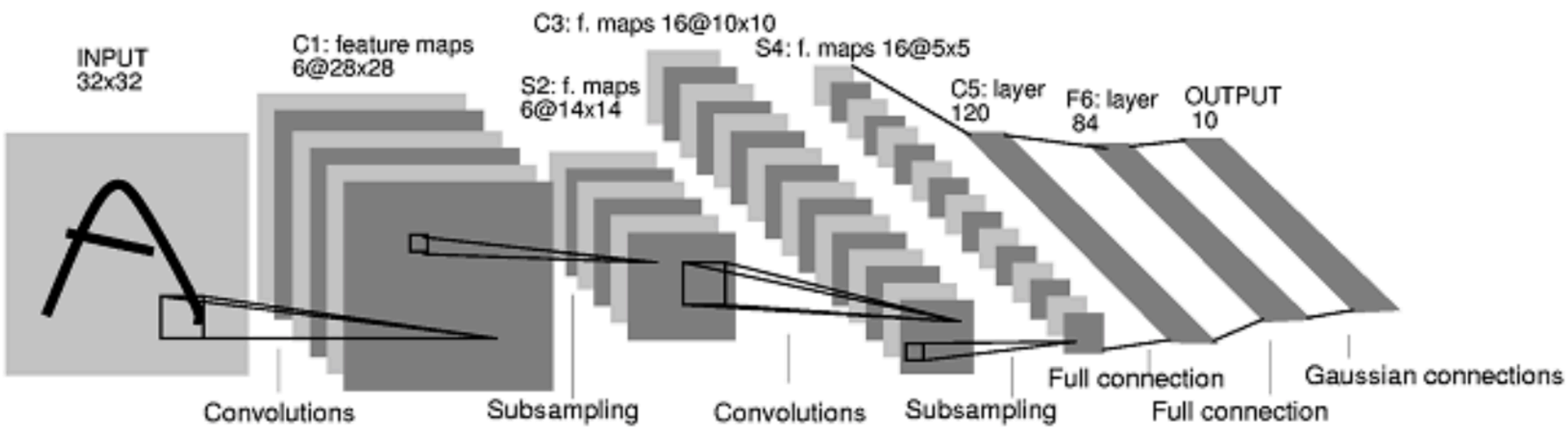
For real time you can’t batch And there is sparsity in both weights and activations key operations is spM x spV



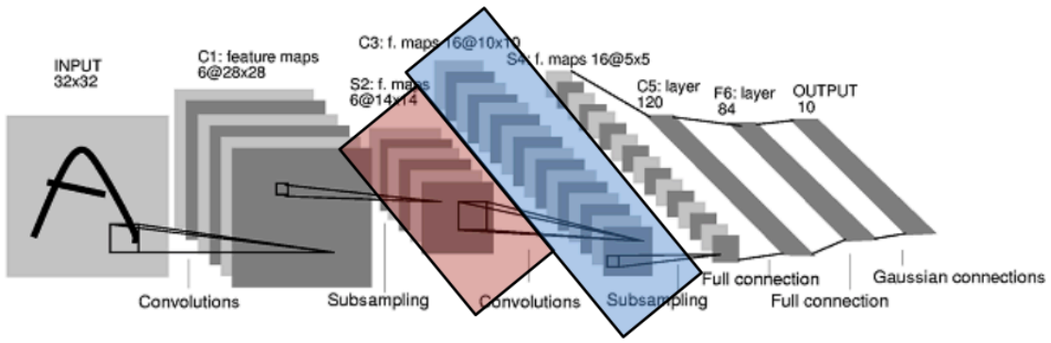
Backpropagation just does this backward

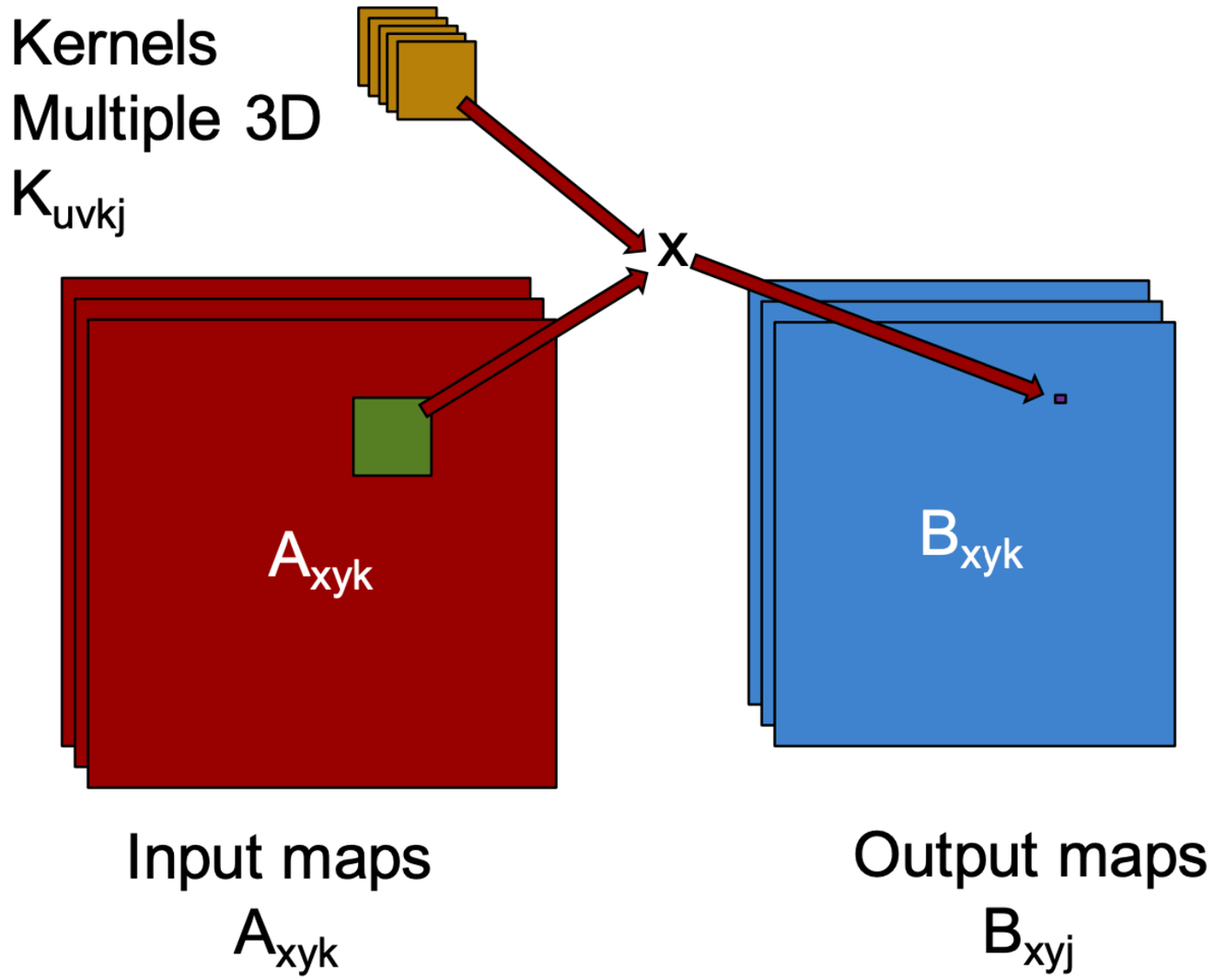


CNNs – For Image Inputs, Convolutional stages act as trained feature detectors

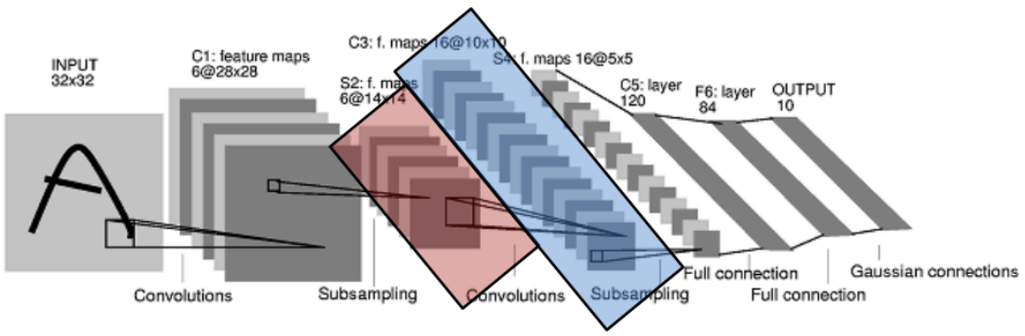


CNNs require Convolution in addition to M x V

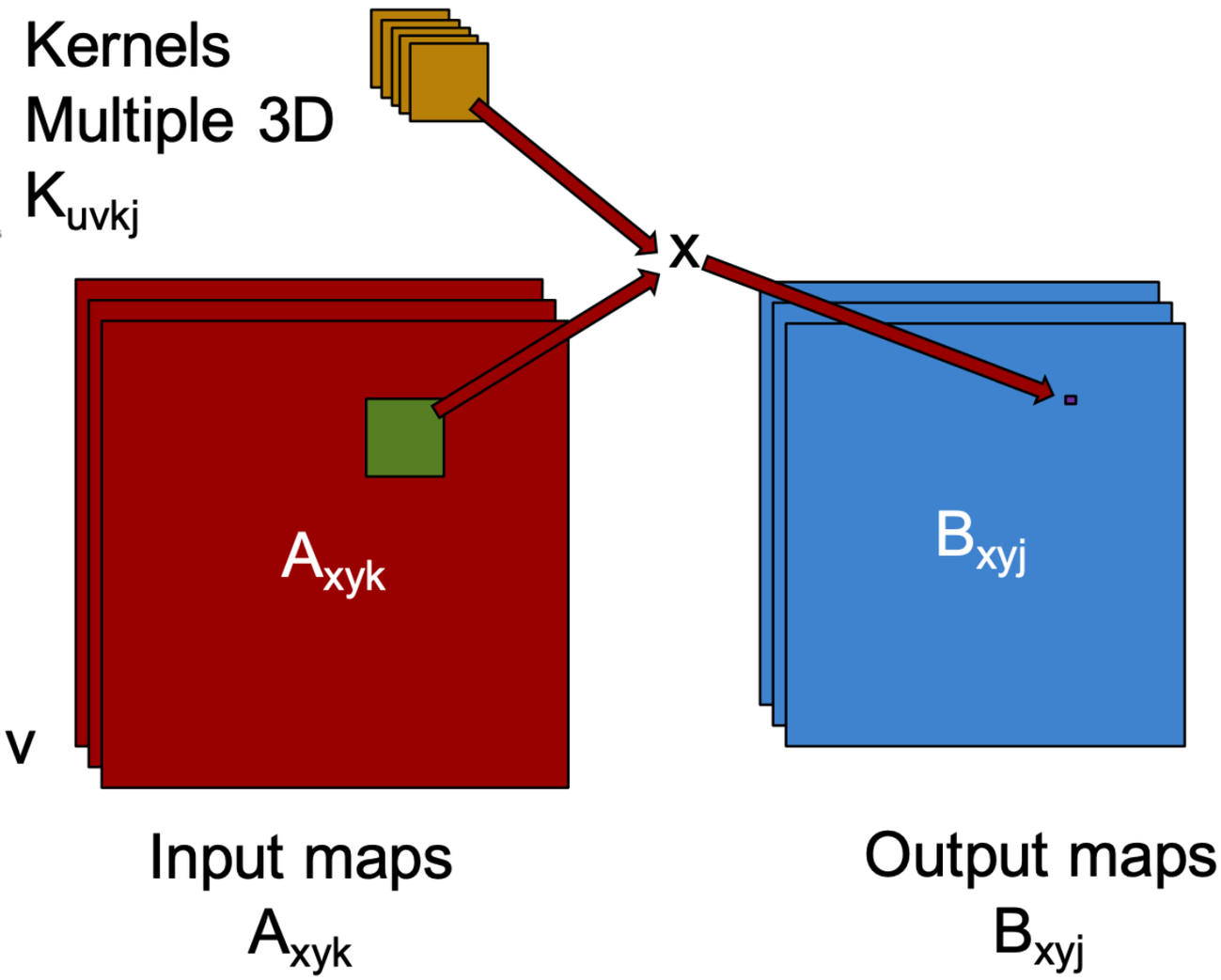




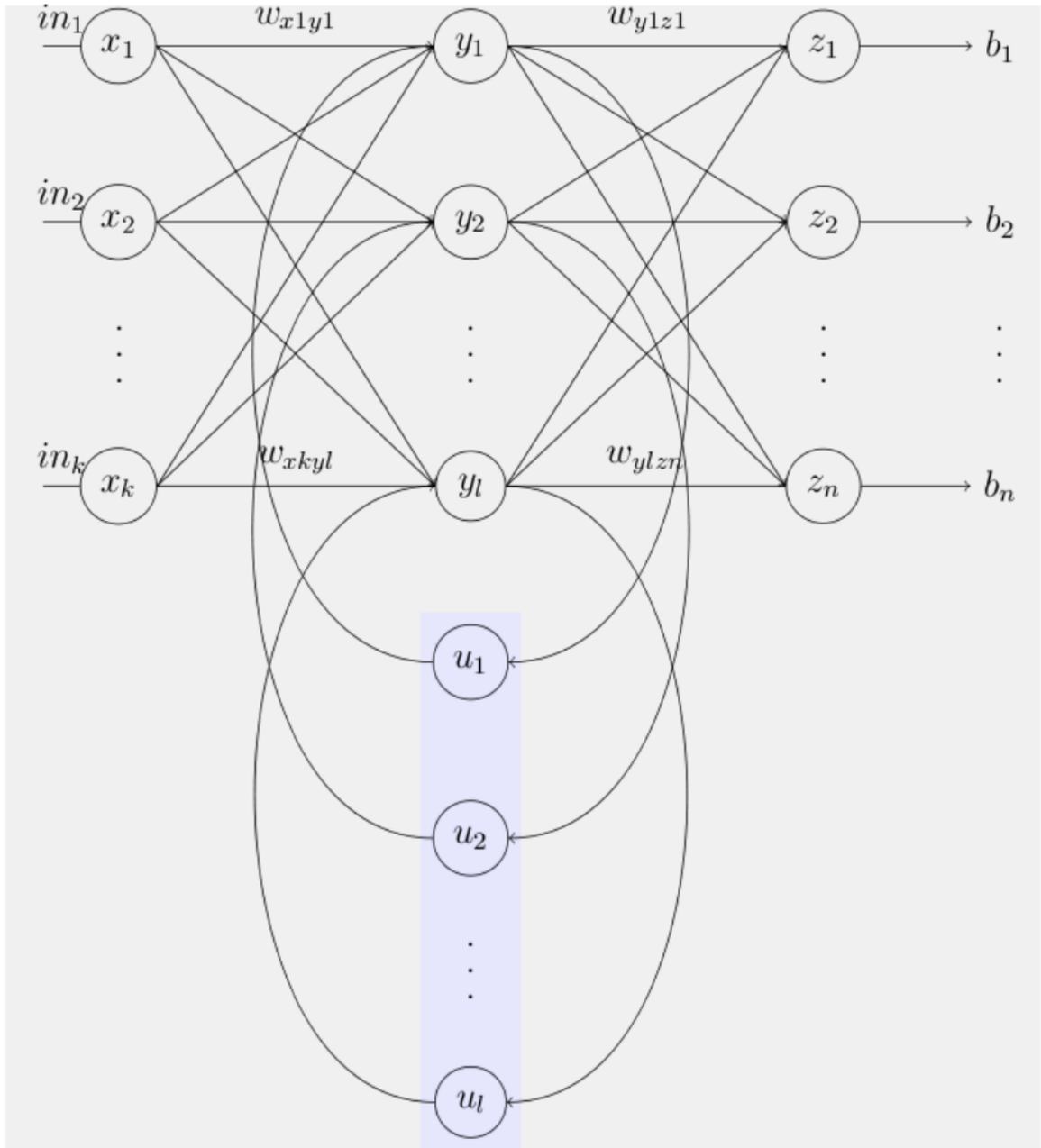
CNNs require Convolution in addition to M x V



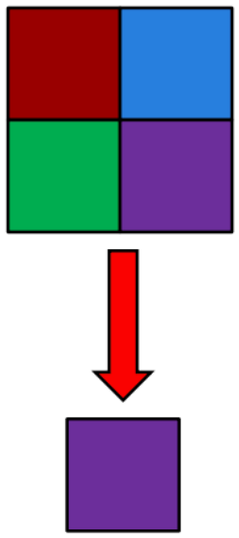
6D Loop
For each output map j
  For each input map k
    For each pixel x,y
      For each kernel element u,v
        B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}



RNNs


in_1
x_1
w_{x1y1}
y_1
w_{y1z1}
z_1
b_1
in_2
x_2
y_2
z_2
b_2
in_k
x_k
w_{xkyl}
y_l
w_{ylzn}
z_n
b_n
u_1
u_2
u_l

Some Other Operations



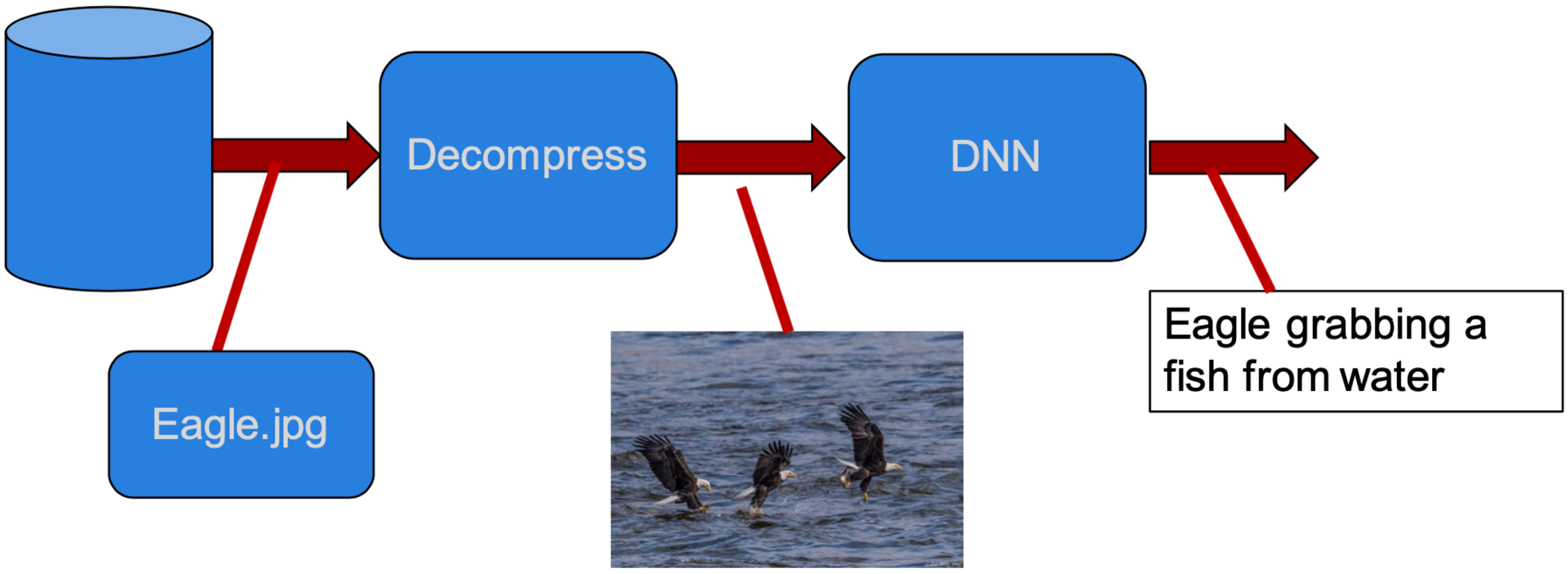
Pooling

ReLU
(or other non-linear function)

$w_{ij} \mathrel{+}= \alpha a_j g_i$

Weight Update

Infrastructure



Summary of the Problem

- Run DNNs, CNNs, and RNNs
  - For training and inference
  - Can batch if not latency sensitive

- Optimize
  - Speed inf/s
  - Efficiency inf/J, inf/s$

- Key operations are
  - MxV
    - MxM if batched
    - May be sparse (spM x spV)
  - Convolution

- Also
  - Pooling, non-linear operator (ReLU), weight update

Baseline

Baseline Performance Xeon E5-2698 – Single Core

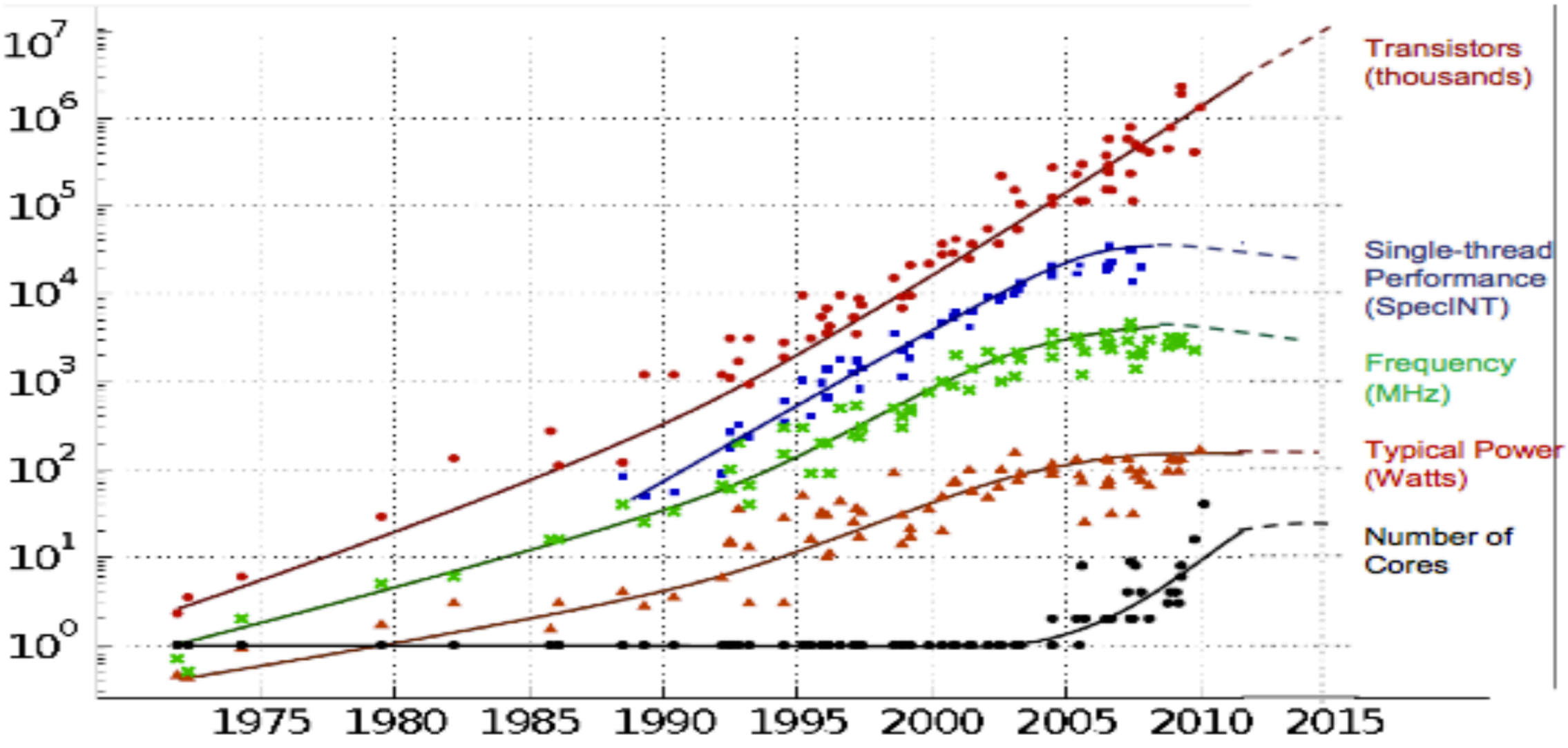
AlexNet – inference, batched

30 f/s

3.2 f/J

Most ops on AVX (SIMD) units

Moore’s law made CPUs 300x faster than in 1990 But its over...

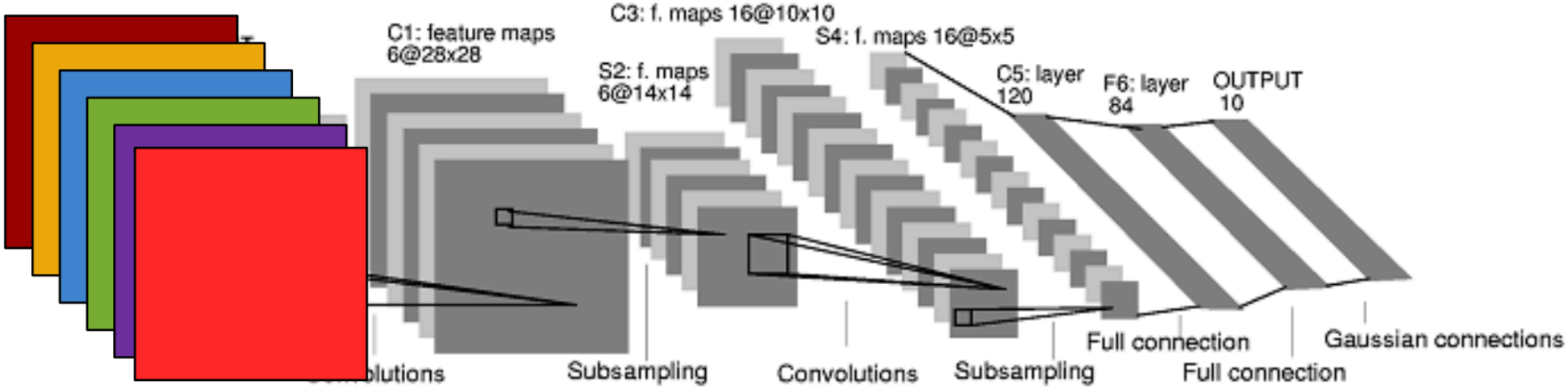


Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

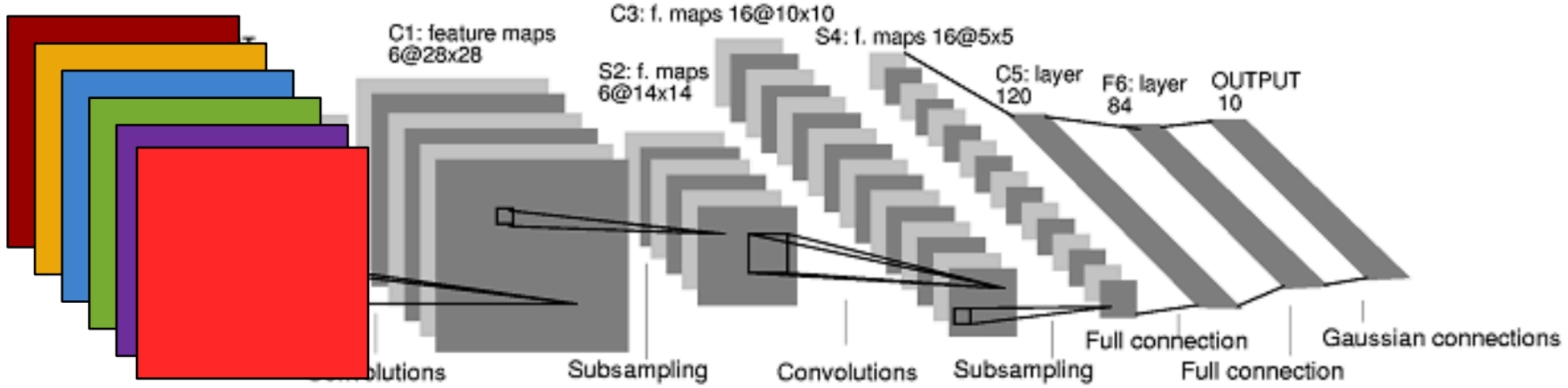
Parallelization

To go faster, use more processors

Lots of parallelism in a DNN



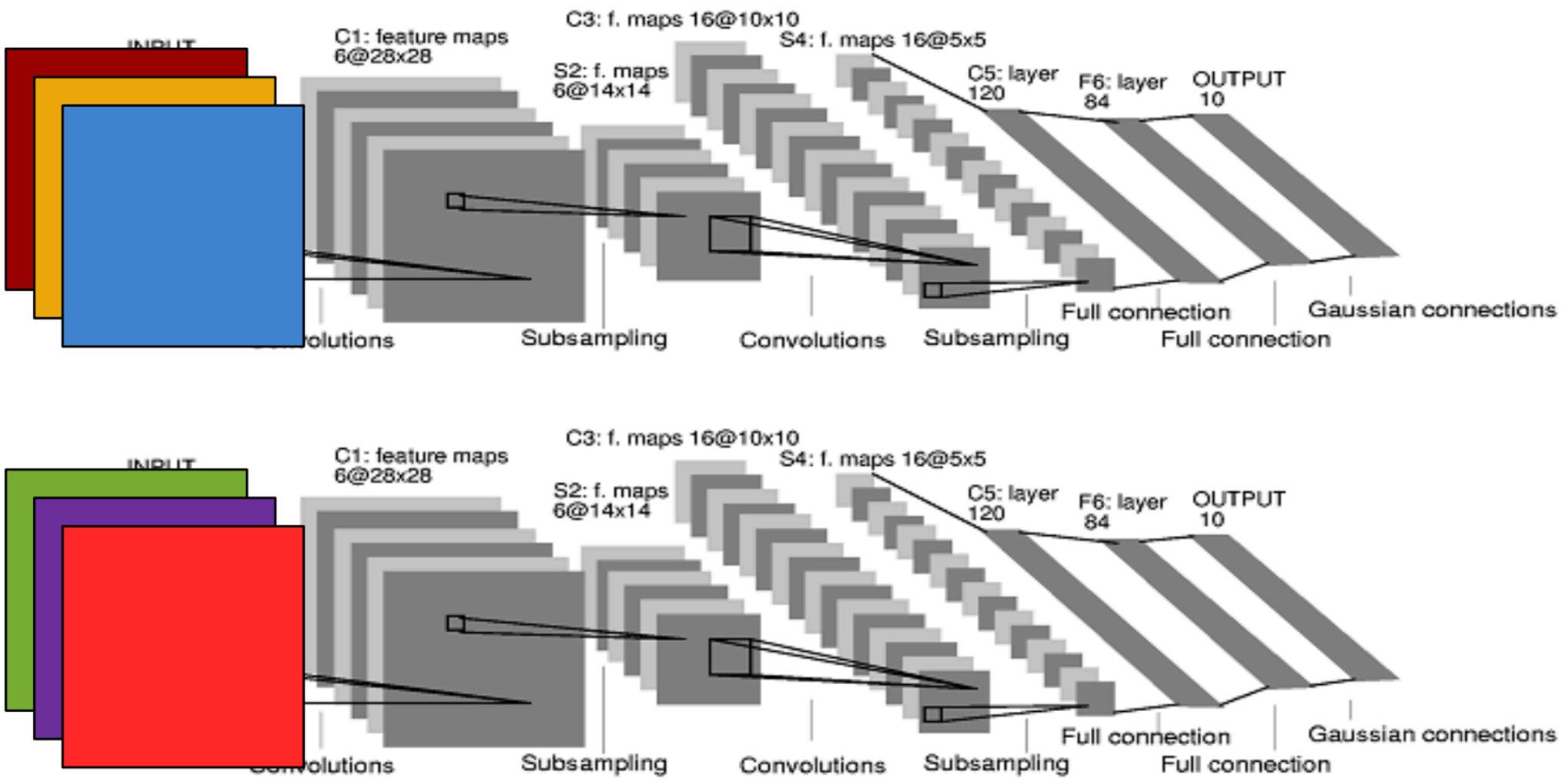
Lots of parallelism in a DNN



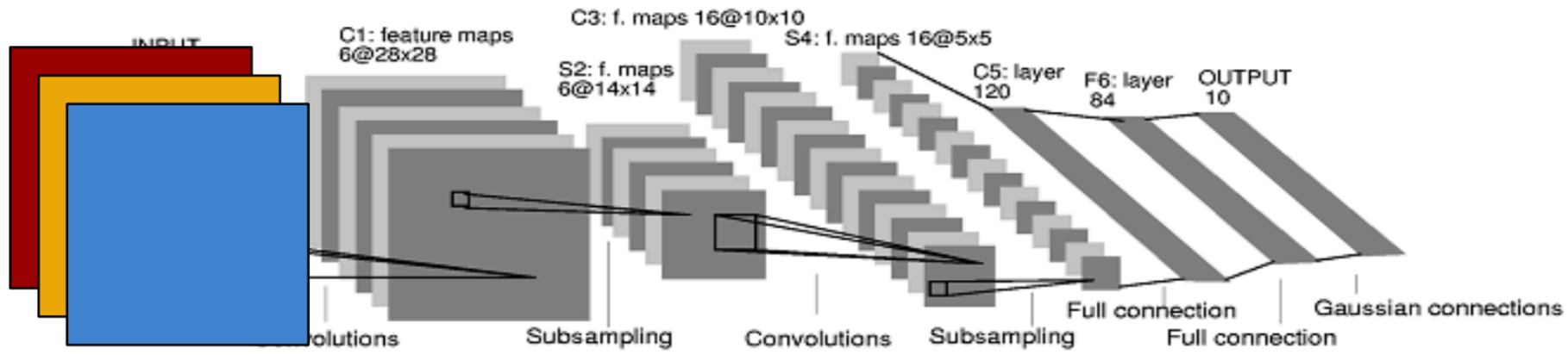
- Inputs
- Points of a feature map
- Filters
- Elements within a filter

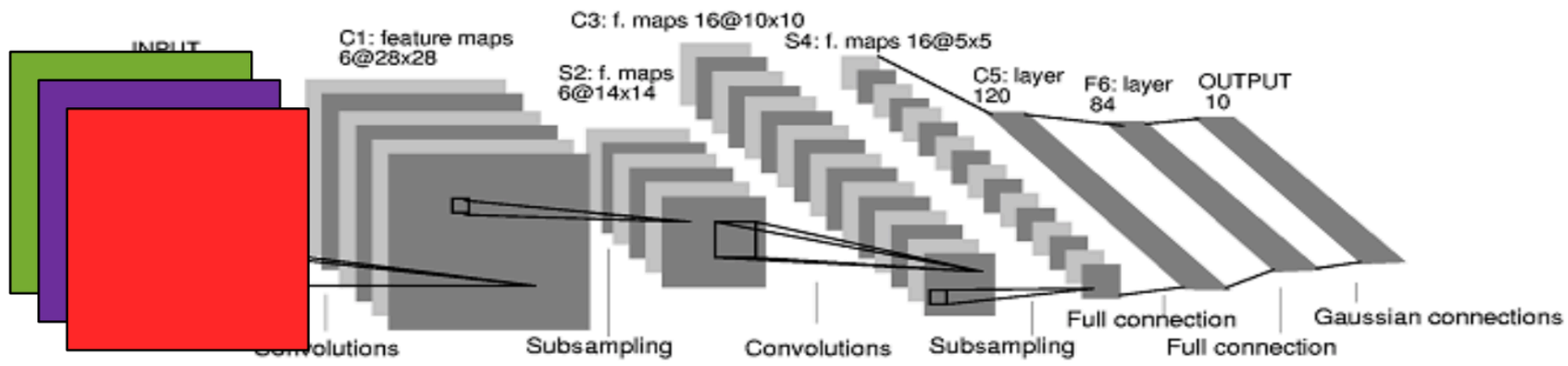
- Multiplies within layer are independent
- Sums are reductions
- Only layers are dependent
- No data dependent operations
  => can be statically scheduled

Data Parallel – Run multiple inputs in parallel



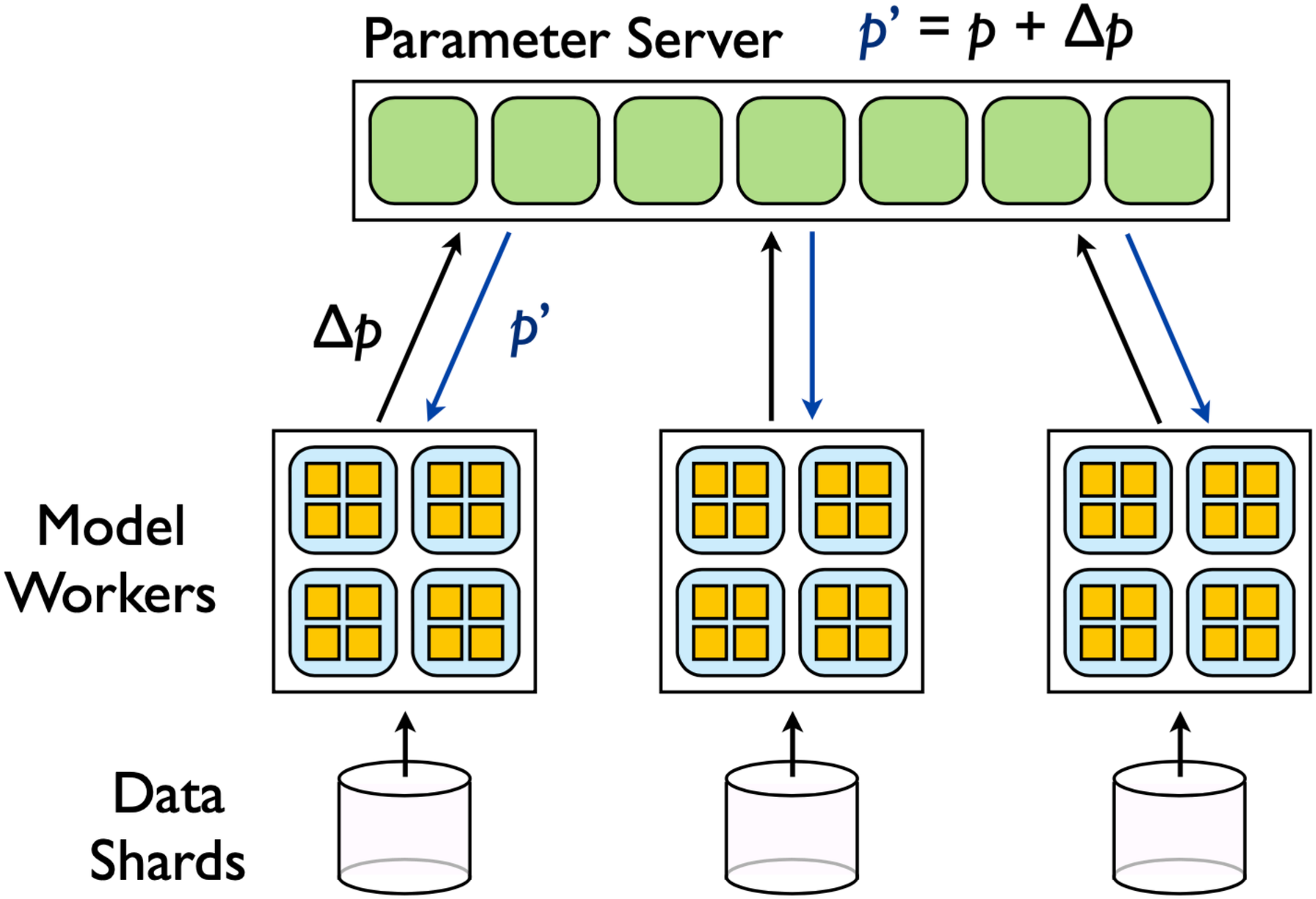
Data Parallel – Run multiple inputs in parallel





- Doesn't affect latency for one input
- Requires P-fold larger batch size
- For training requires coordinated weight update

Parameter Update

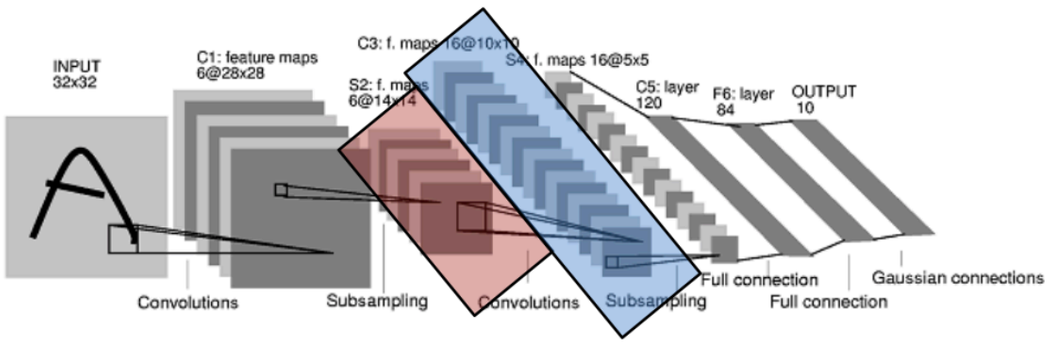


Large Scale Distributed Deep Networks, Jeff Dean et al., 2013

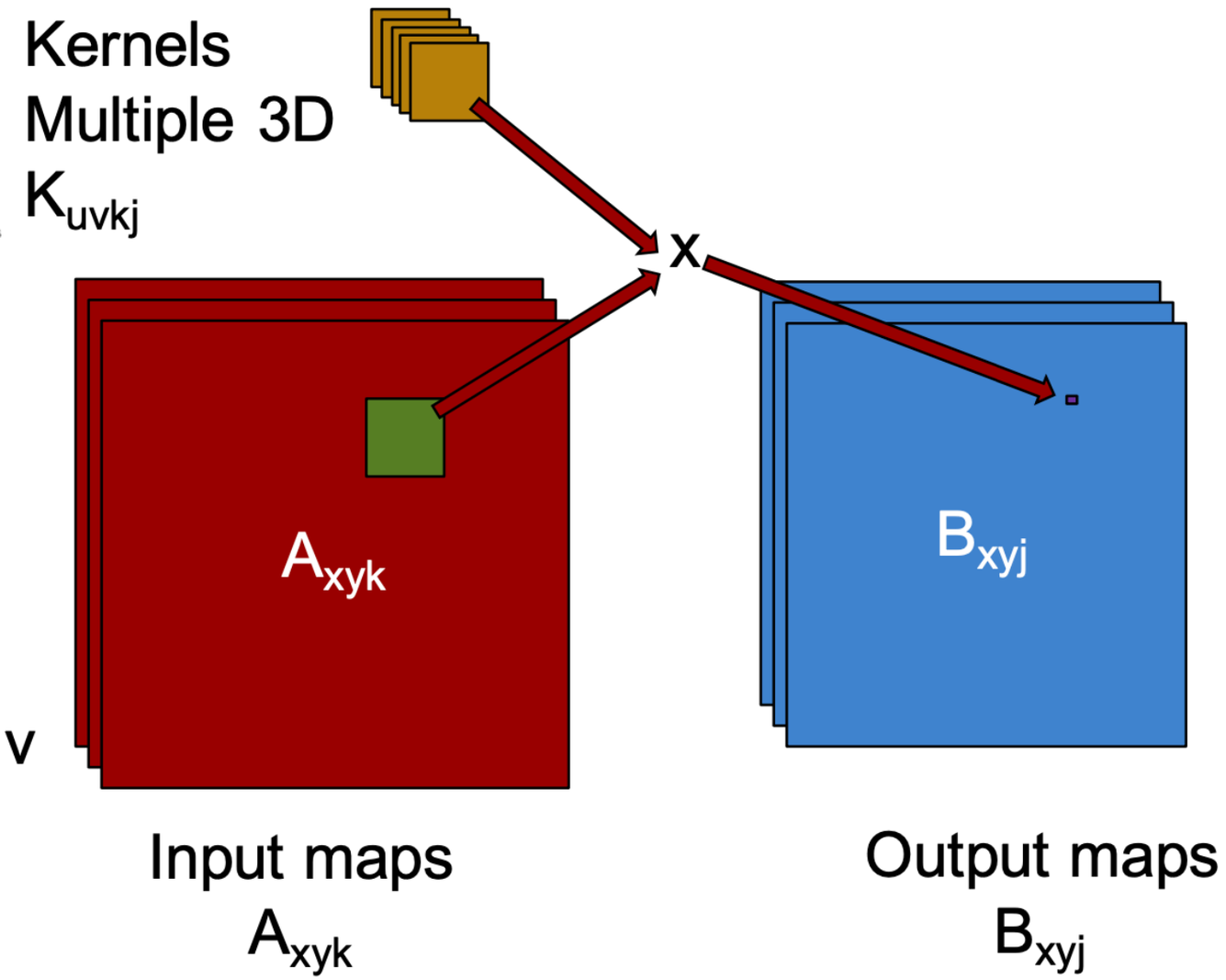
Model Parallel

Split up the Model – i.e. the network

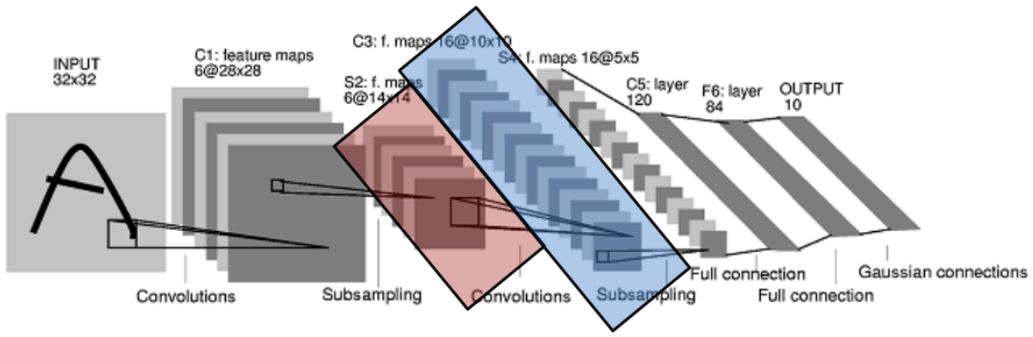
Model-Parallel Convolution

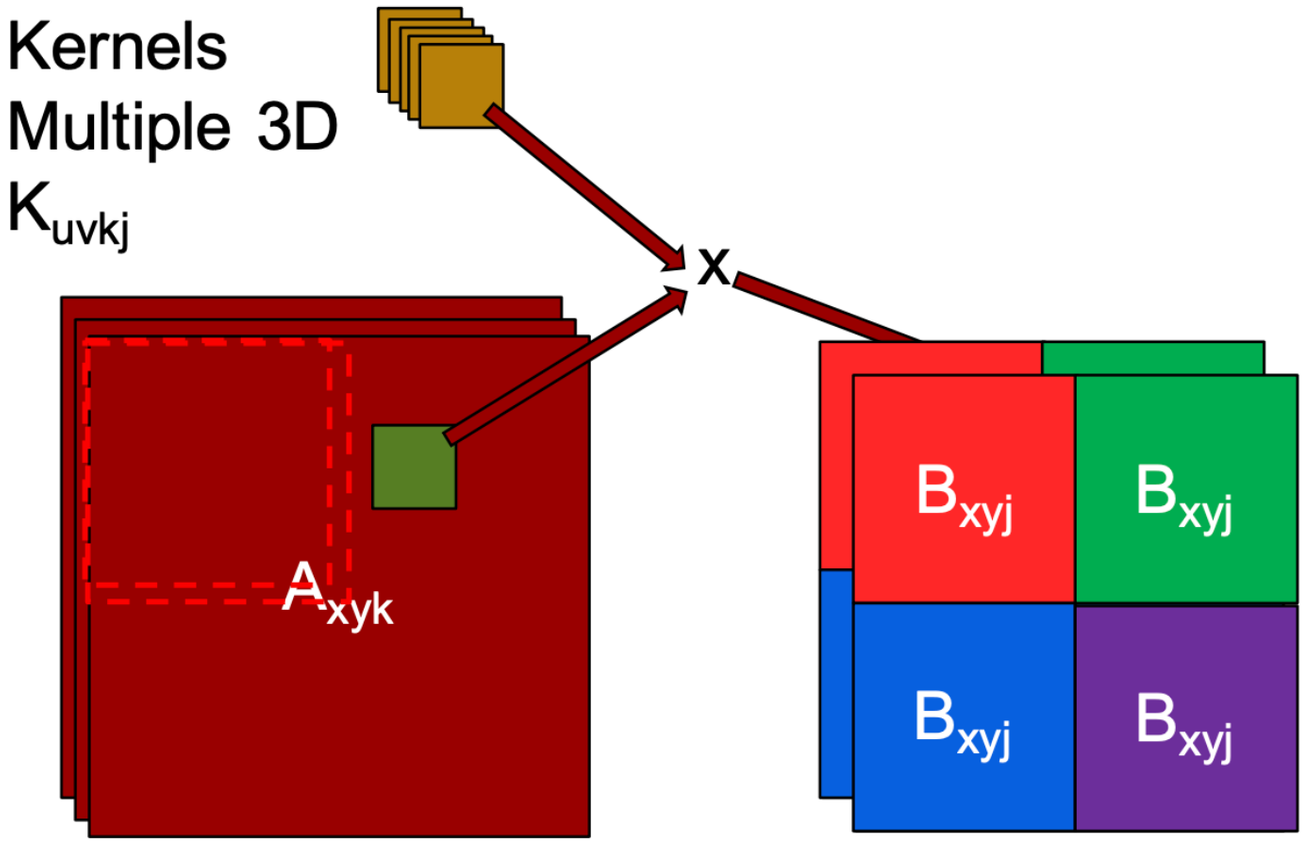


6D Loop
For each output map j
  For each input map k
    For each pixel x,y
      For each kernel element u,v
        B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}



Model-Parallel Convolution – by output region (x,y)





6D Loop

Forall region XY

  For each output map j

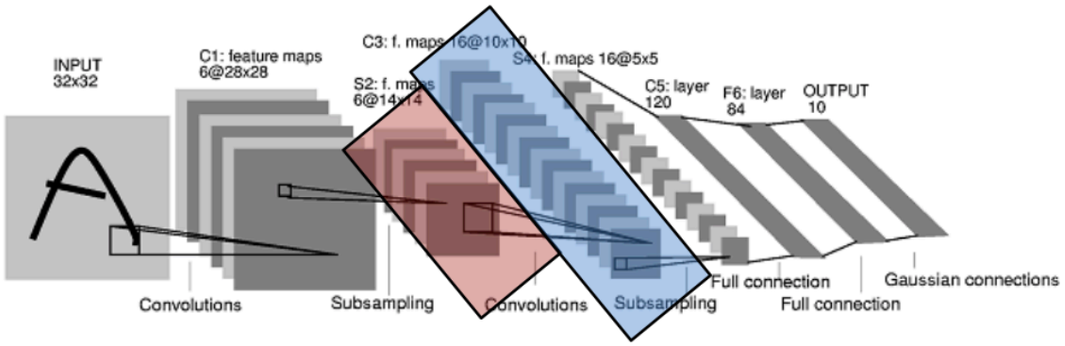
    For each input map k

      For each pixel x,y in XY

        For each kernel element u,v

          B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}

Model-Parallel Convolution – By output map j (filter)



6D Loop

Forall output map j

  For each input map k

    For each pixel x,y

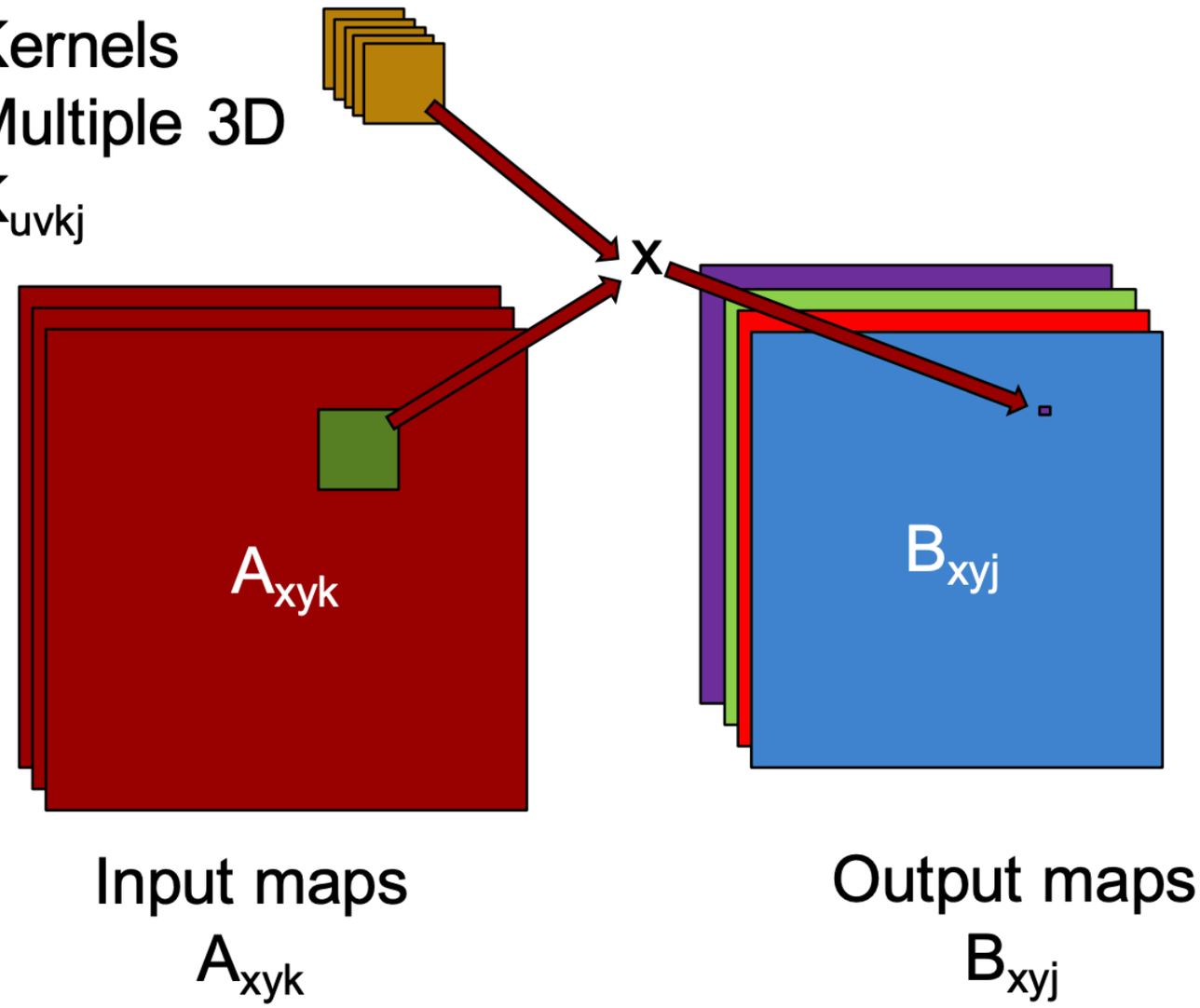
      For each kernel element u,v

        B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}

Kernels

Multiple 3D

K_{uvkj}



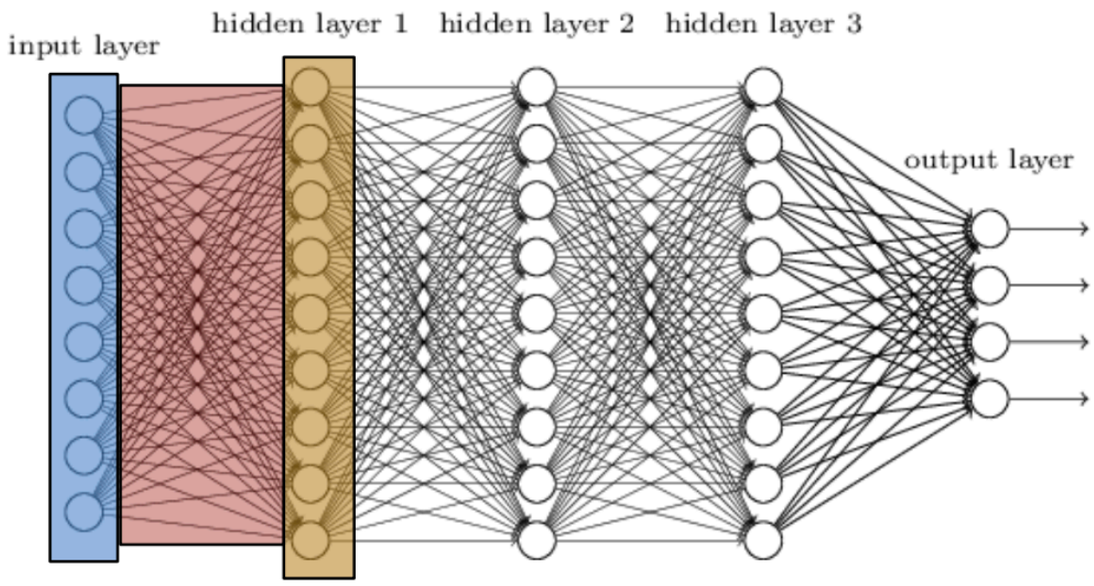
Input maps

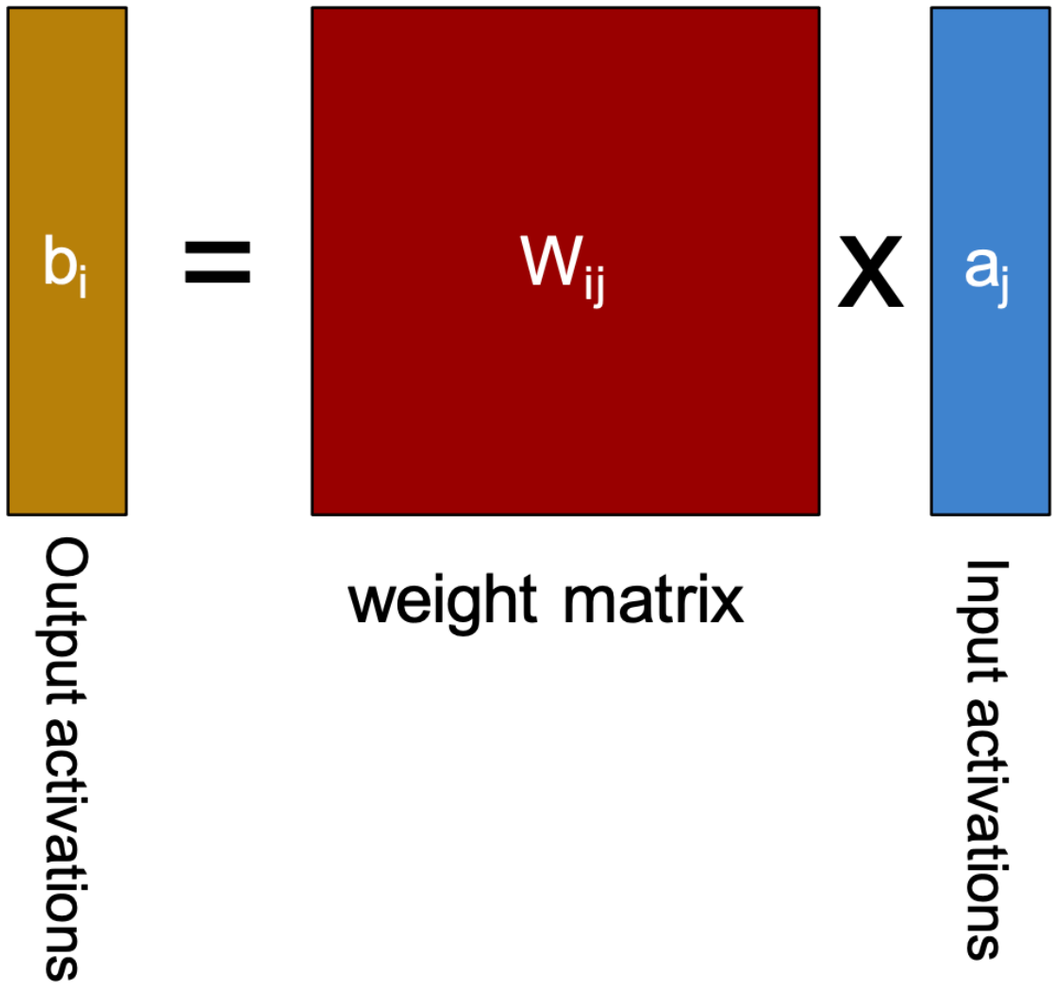
A_{xyk}

Output maps

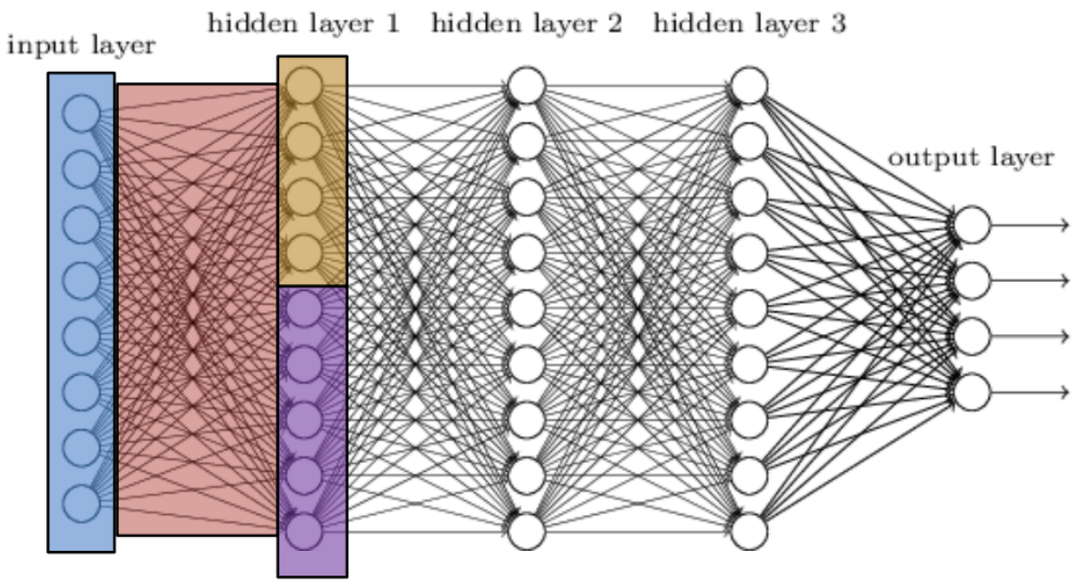
B_{xyj}

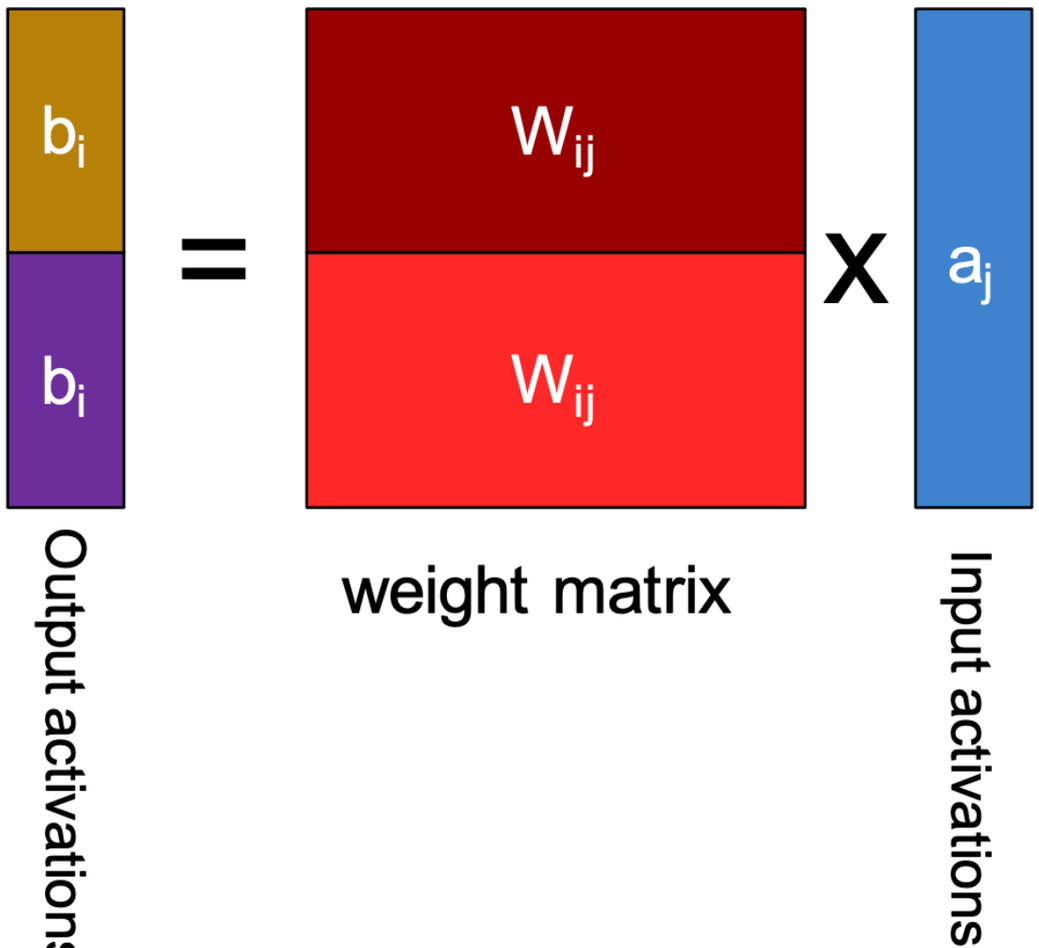
Model Parallel Fully-Connected Layer (M x V)





Model Parallel Fully-Connected Layer (M x V)

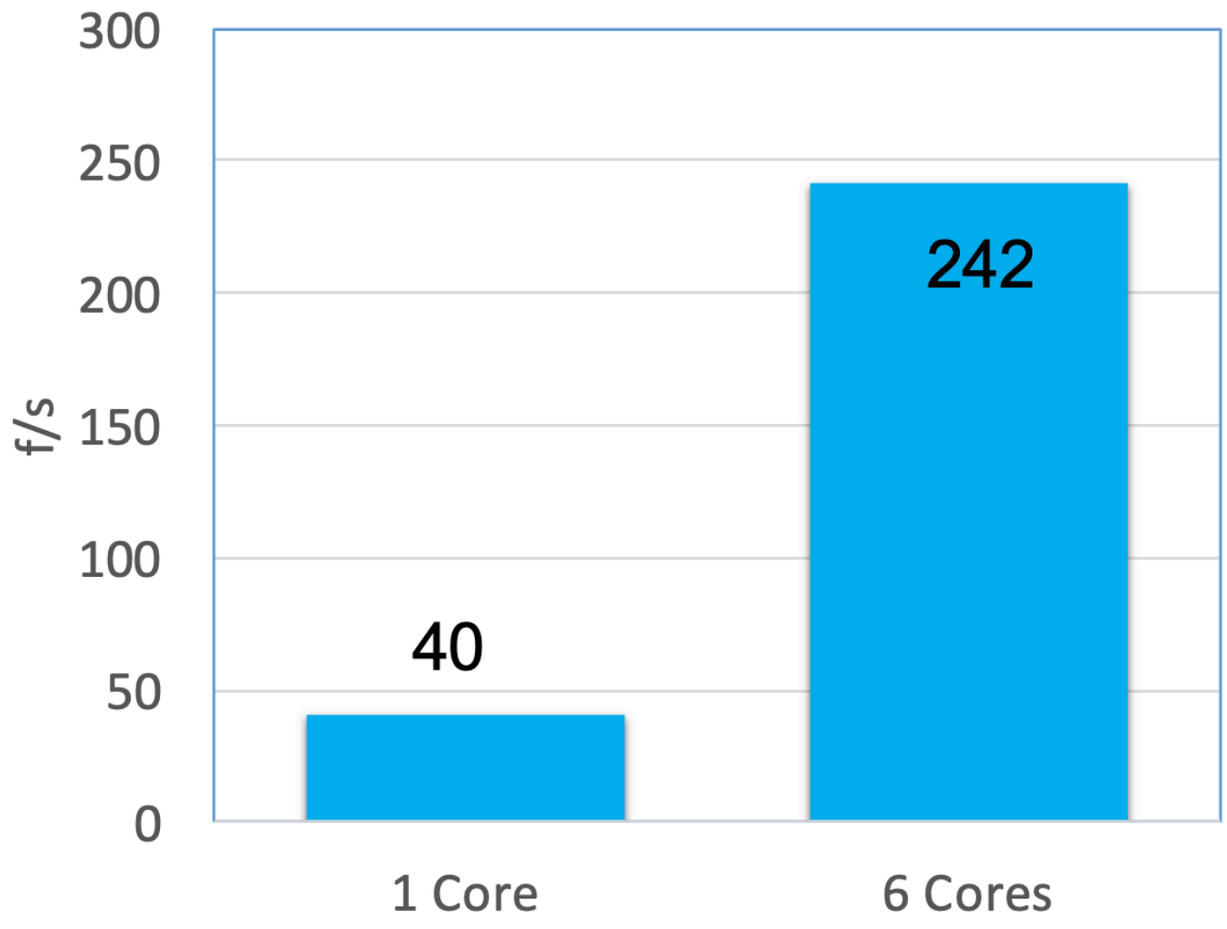




Hyper-Parameter Parallel

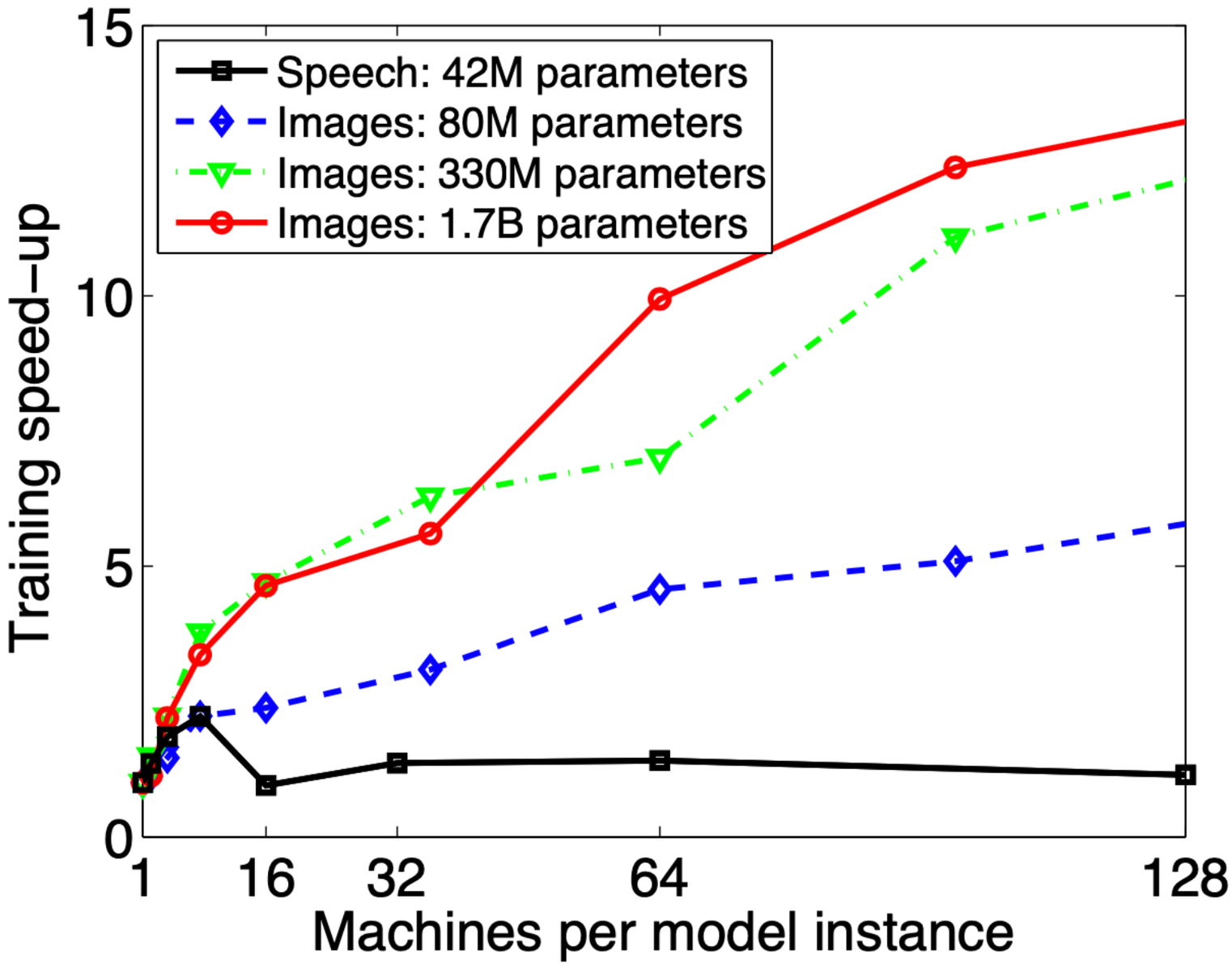
Try many alternative networks in parallel

CPU Parallelism – Core i7 – 1 core vs 6 cores



NVIDIA, “Whitepaper: GPU-based deep learning inference: A performance and power analysis.”

Data and Model Parallel Performance



Dean et al. Large Scale Distributed Deep Networks, NIPS 2012

Summary of Parallelism

- Lots of parallelism in DNNs
  - 16M independent multiplies in one FC layer
  - Limited by overhead to exploit a fraction of this

- Data parallel
  - Run multiple training examples in parallel
  - Limited by batch size

- Modelparallel
  - Split model over multiple processors
  - By layer
  - Conv layers by map region
  - Fully connected layers by output activation

- Easy to get 16-64 GPUs training one model in parallel

GPUs

- To go fast, use multiple processors

- To go fast, use multiple processors

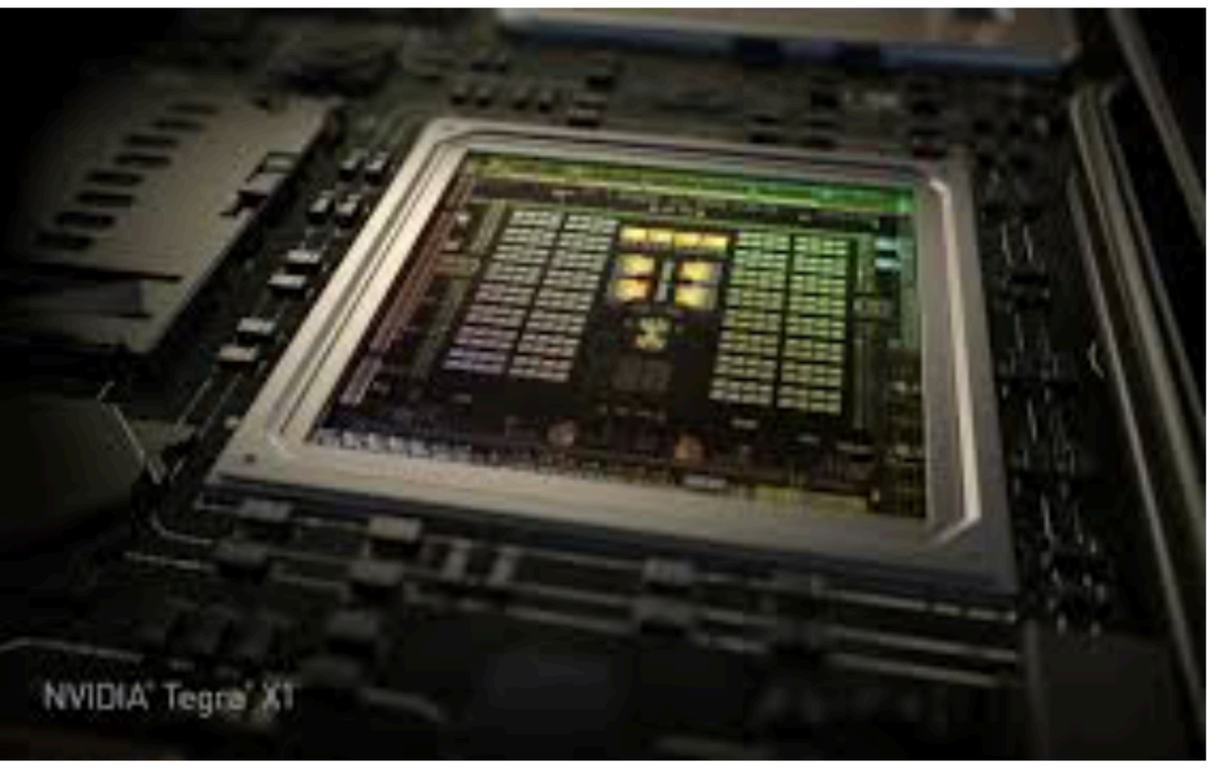
- To be efficient and fast, use GPUs

- To go fast, use multiple processors
- To be efficient and fast, use GPUs
- To be efficient and go really fast, use multiple GPUs

Titan X

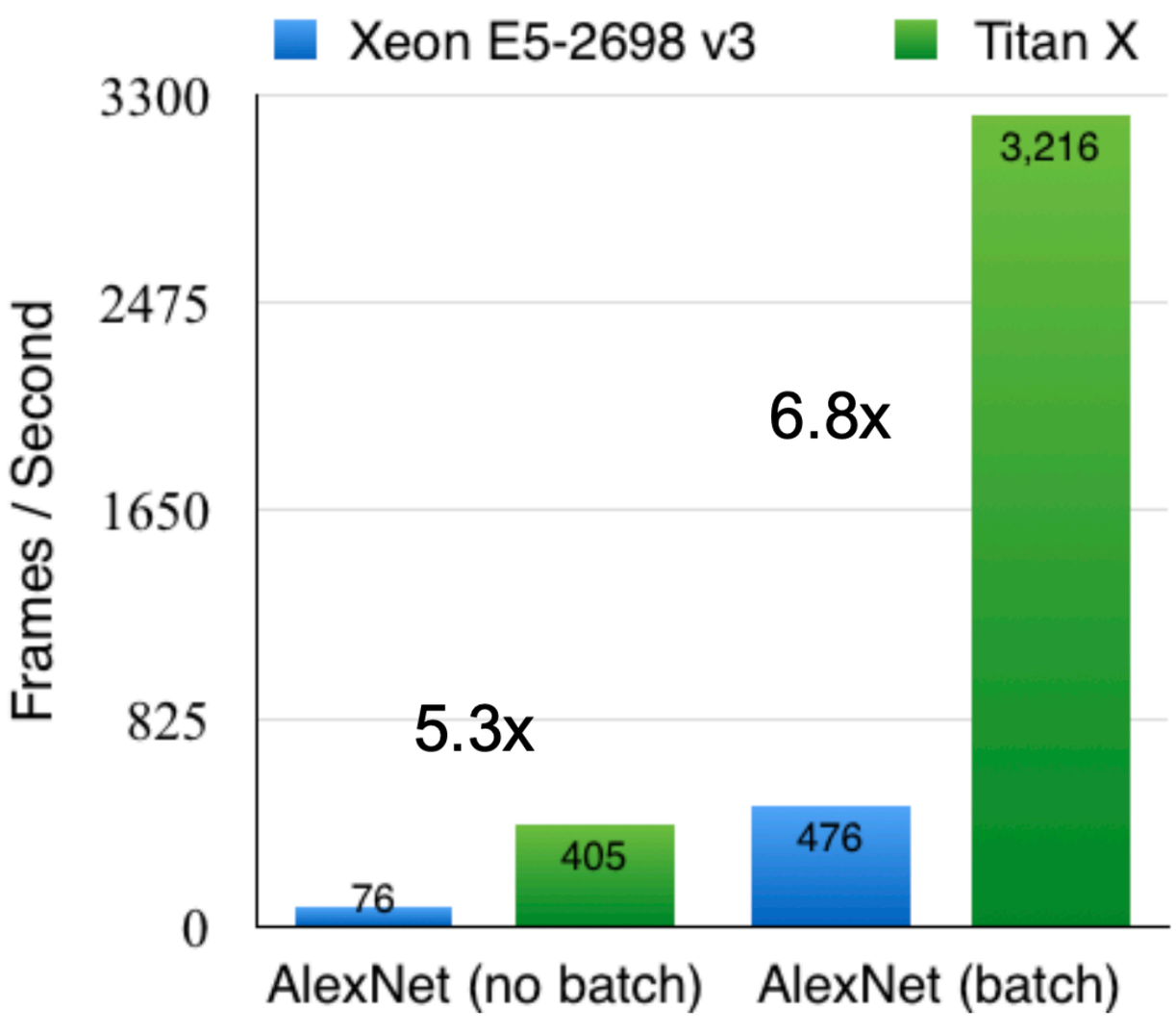
- 3072 CUDA cores @ 1 GHz
- 6 Teraflops FP32
- 12GB of GDDR5 @ 336 GB/sec
- 250W TDP
- 24GFLOPS/W
- 28nm process

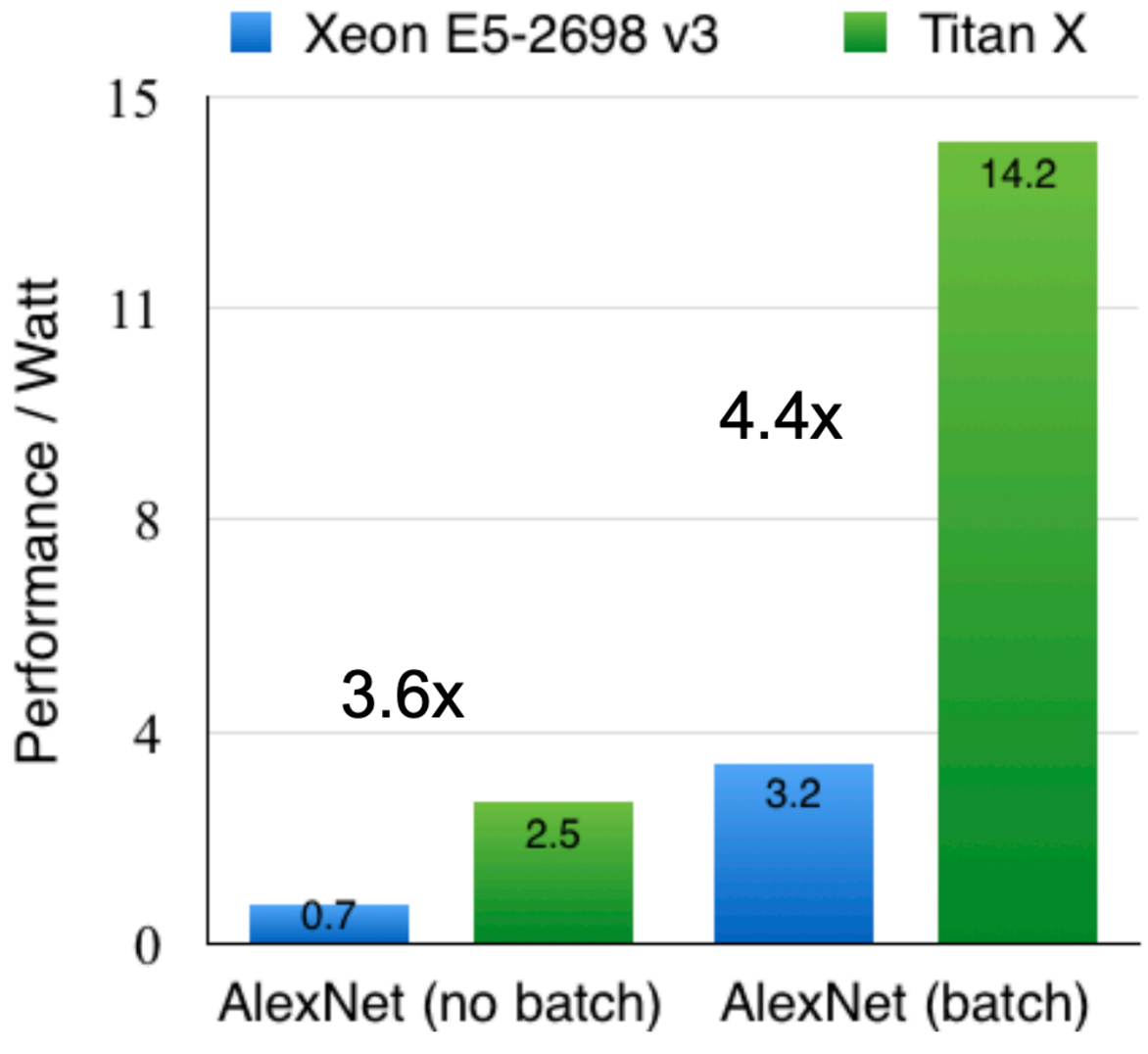
Tegra X1



- 256 CUDA cores @ ~1 GHz
- 1 Teraflop FP16
- 4GB of LPDDR4 @ 25.6 GB/s
- 15 W TDP (1W idle, <10W typical)
- 100GFLOPS/W (FP16)
- 20nm process

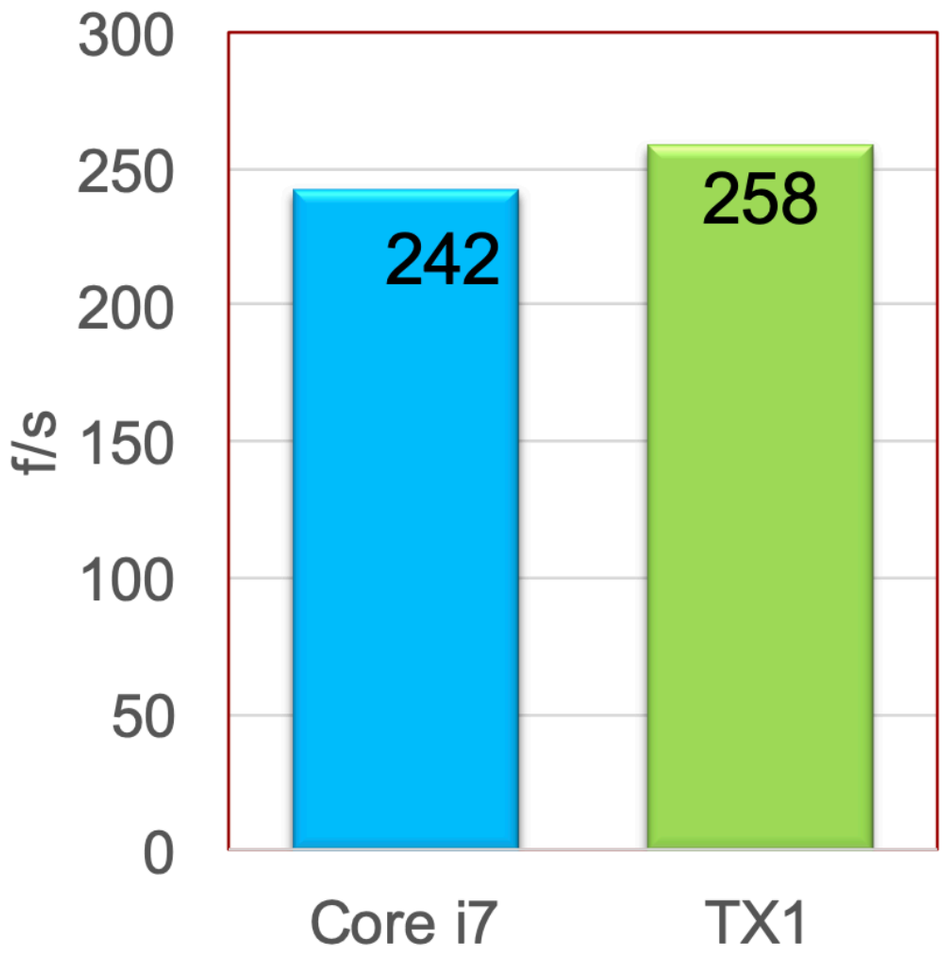
Xeon E5-2698 CPU v.s. TitanX GPU

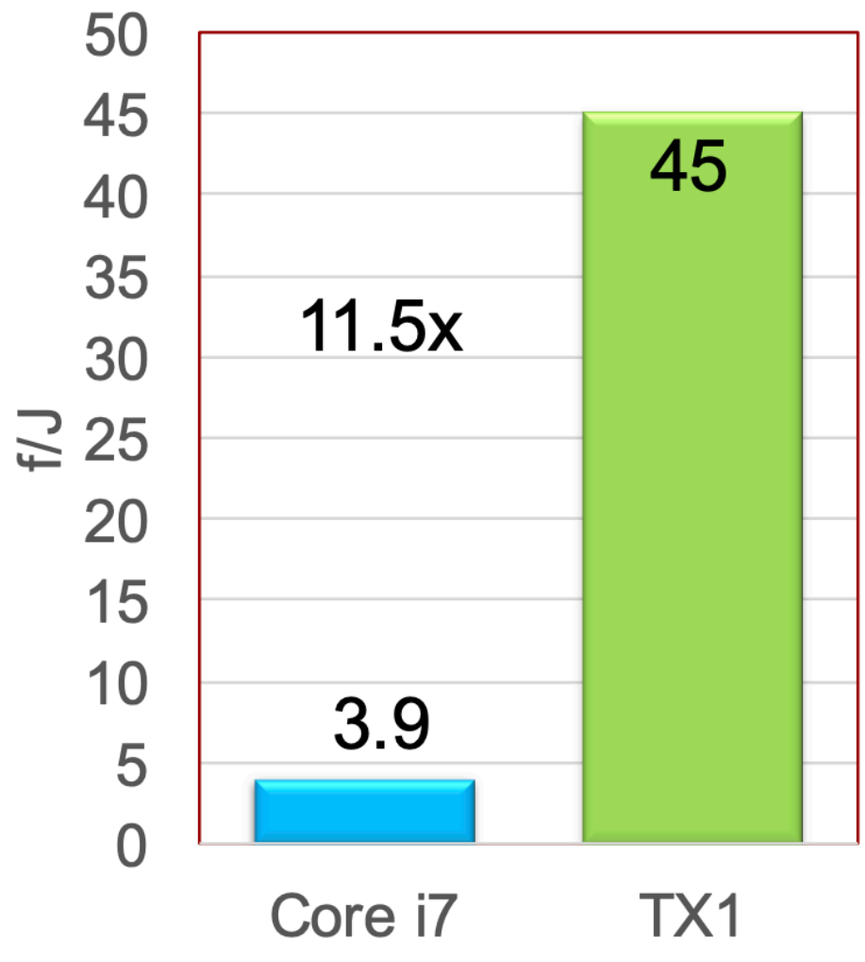




NVIDIA, “Whitepaper: GPU-based deep learning inference: A performance and power analysis.”

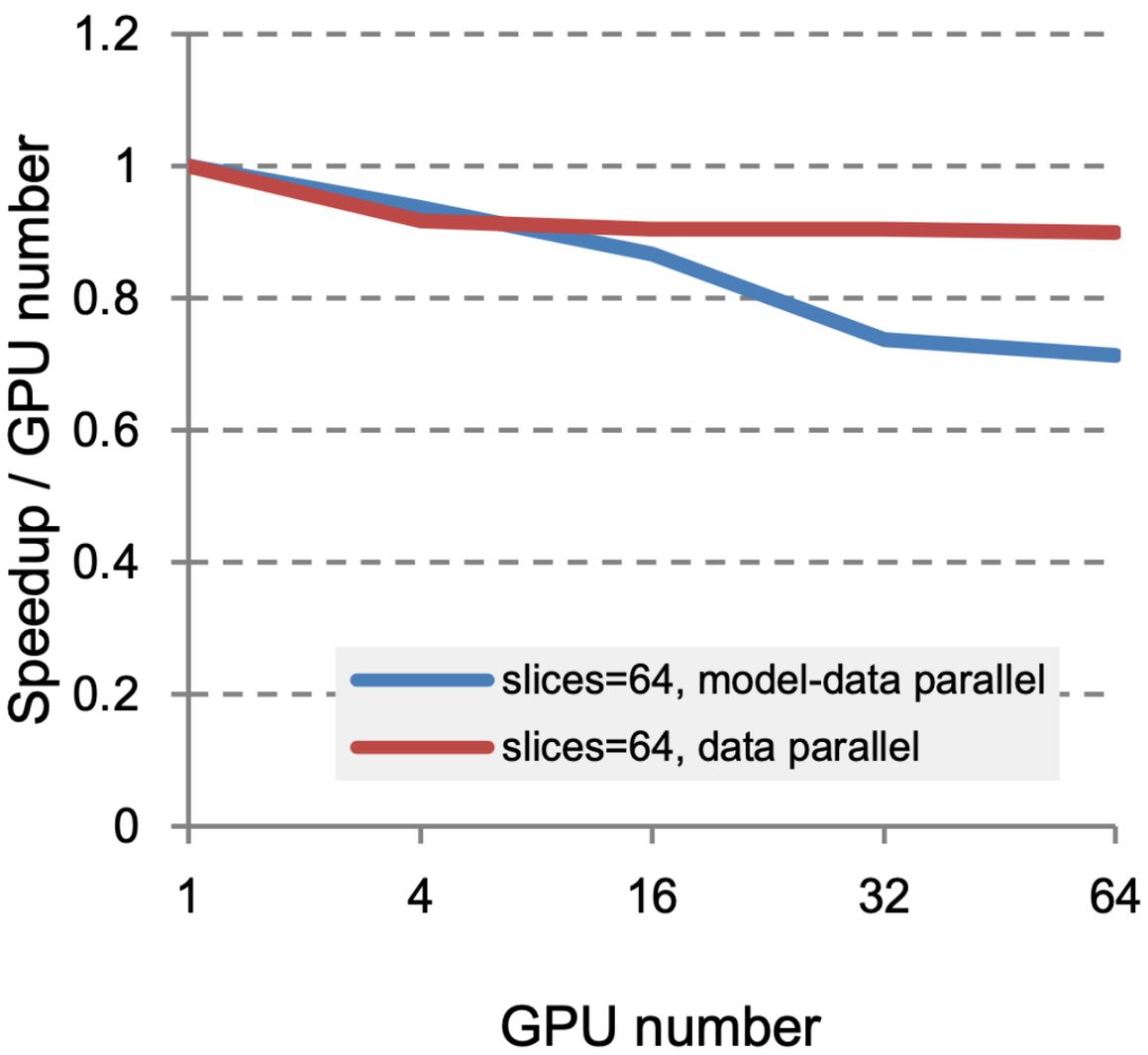
Tegra X1 vs Core i7

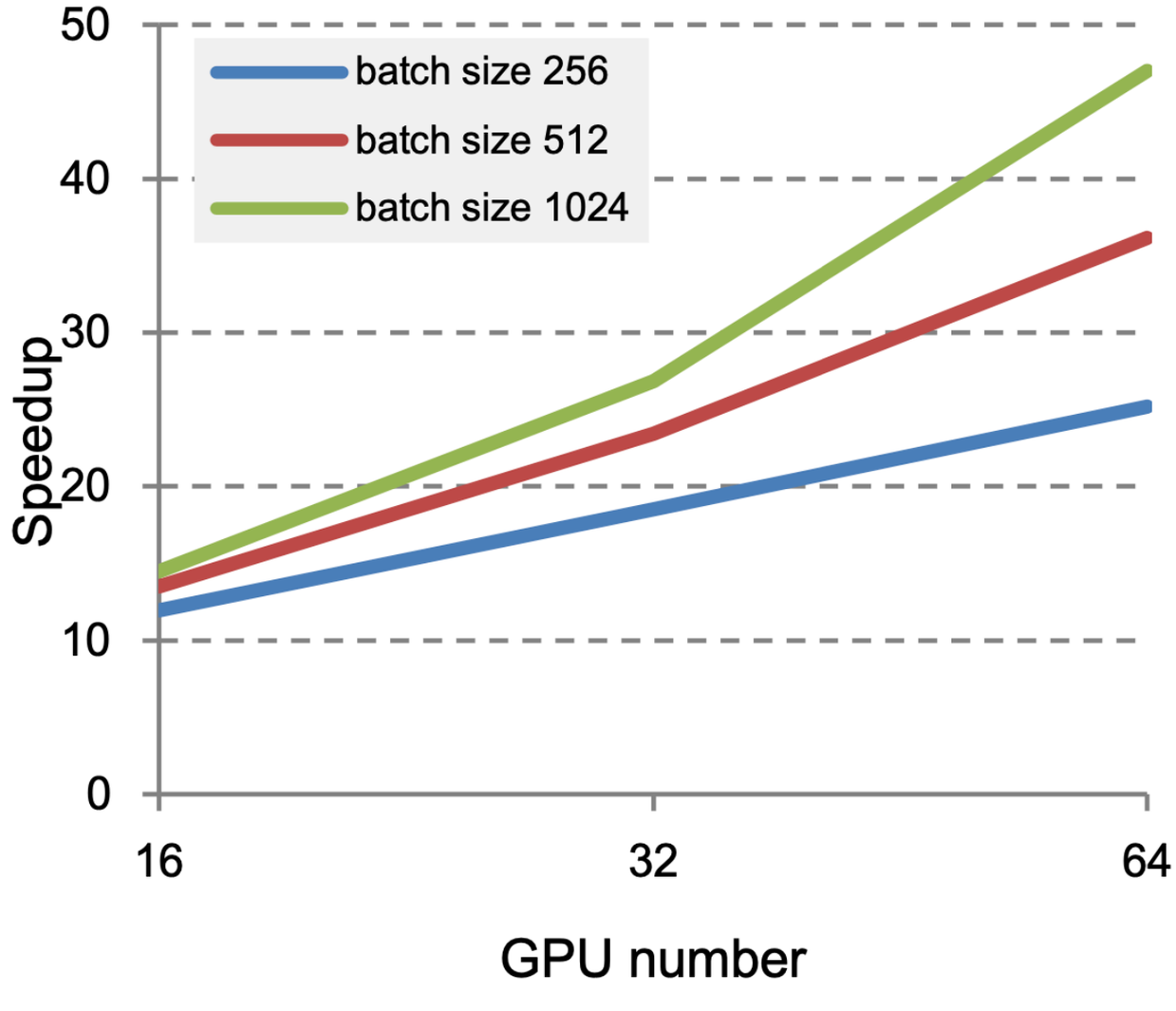




NVIDIA, “Whitepaper: GPU-based deep learning inference: A performance and power analysis.”

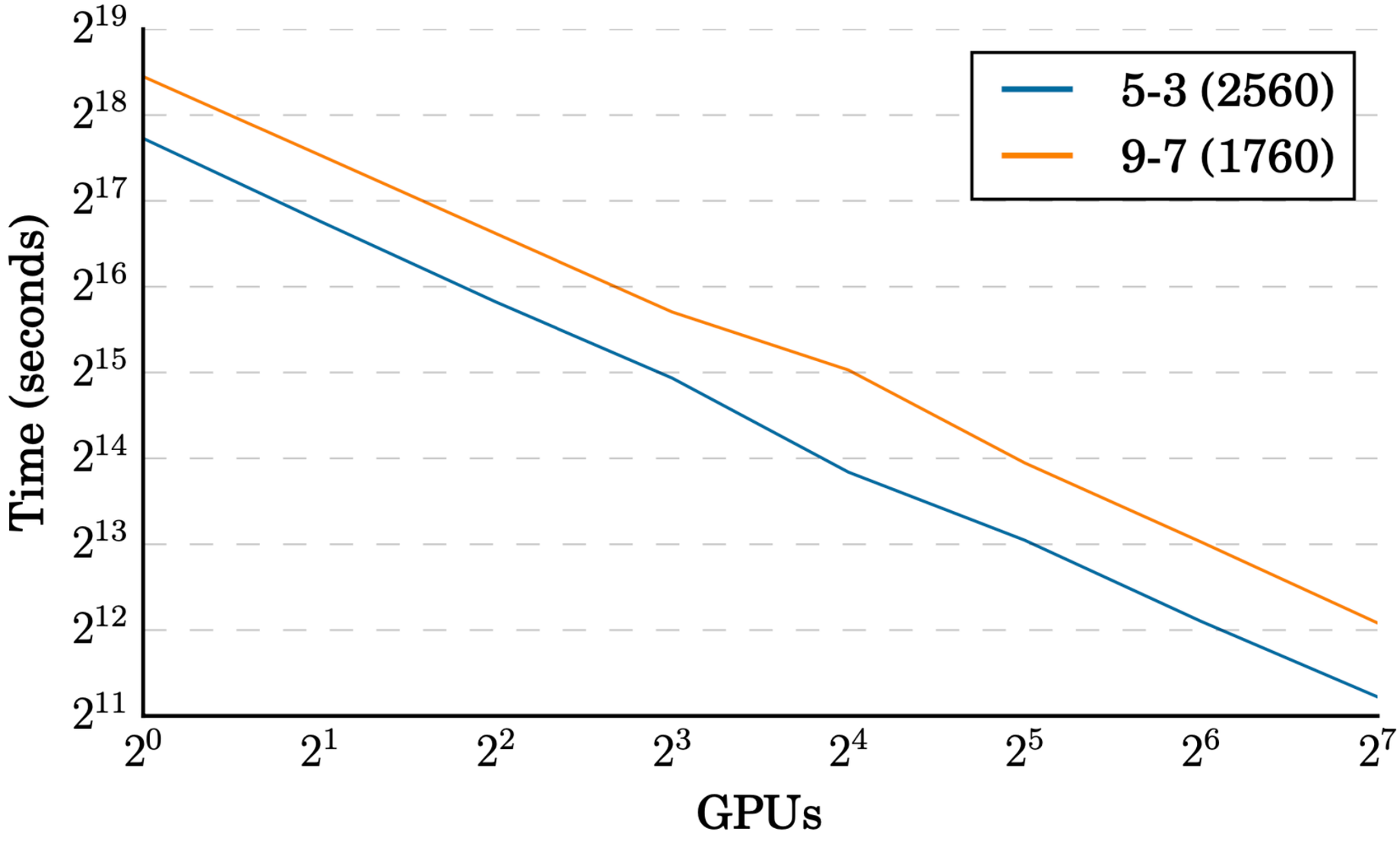
Parallel GPUs





Ren Wu et al., Deep Image: Scaling up Image Recognition, arXiv 2015

Parallel GPUs on Deep Speech 2



Baidu, Deep Speech 2: End-to-End Speech Recognition in English and Mandarin, 2015

Summary of GPUs

- Titan X ~6x faster, 4x more efficient than Xeon E5
- TX1 11.5x more efficient than Core i7
- On inference
- Larger gains on training

- Data parallelism scales easily to 16GPUs
- With some effort, linear speedup to 128GPUs