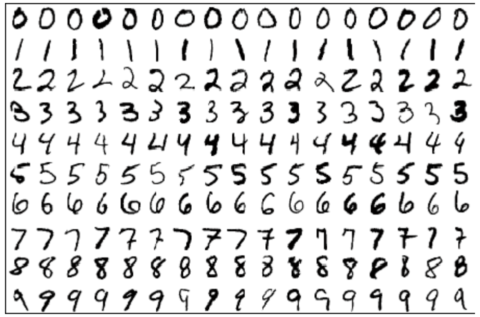
Tal Ben-Nun

Parallel and Distributed Deep Learning

Where is Deep Learning used?

Digit Recognition



Object Classification
Segmentation

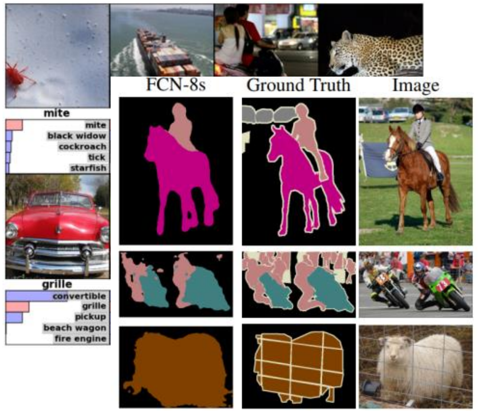
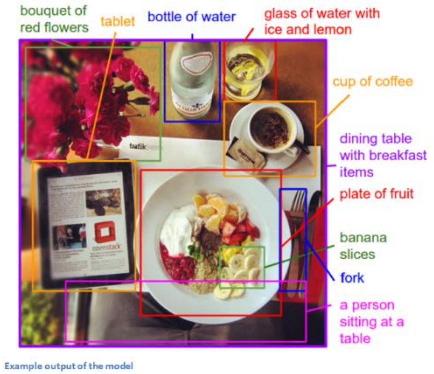
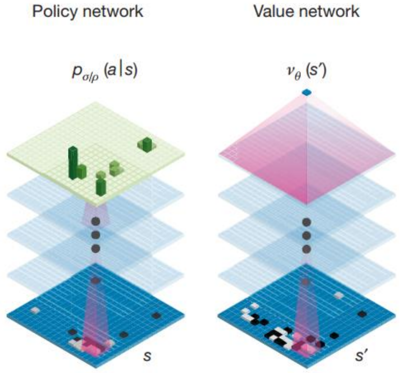


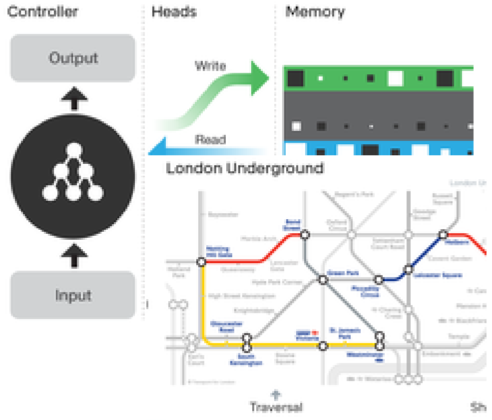
Image Captioning



Gameplay AI
Translation



Neural Computers
Routing



1989 — 2012 — 2013 — 2014 — 2016 — 2017

Why Scale Up?

- Enormous amounts of data
  - MSCOCO: 19 GB
  - ImageNet (1k): 180 GB
  - ImageNet (22k): A few TB
  - Industry: Much larger

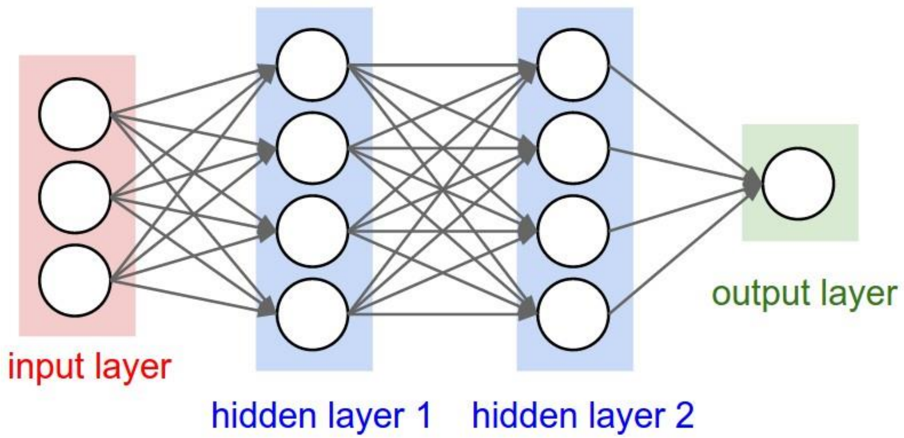
- Large neural network architectures
  - 100-200 layers deep today, ~100M-2B parameters

- Faster prototyping
  - Training time: 10s of hours to days (and weeks)

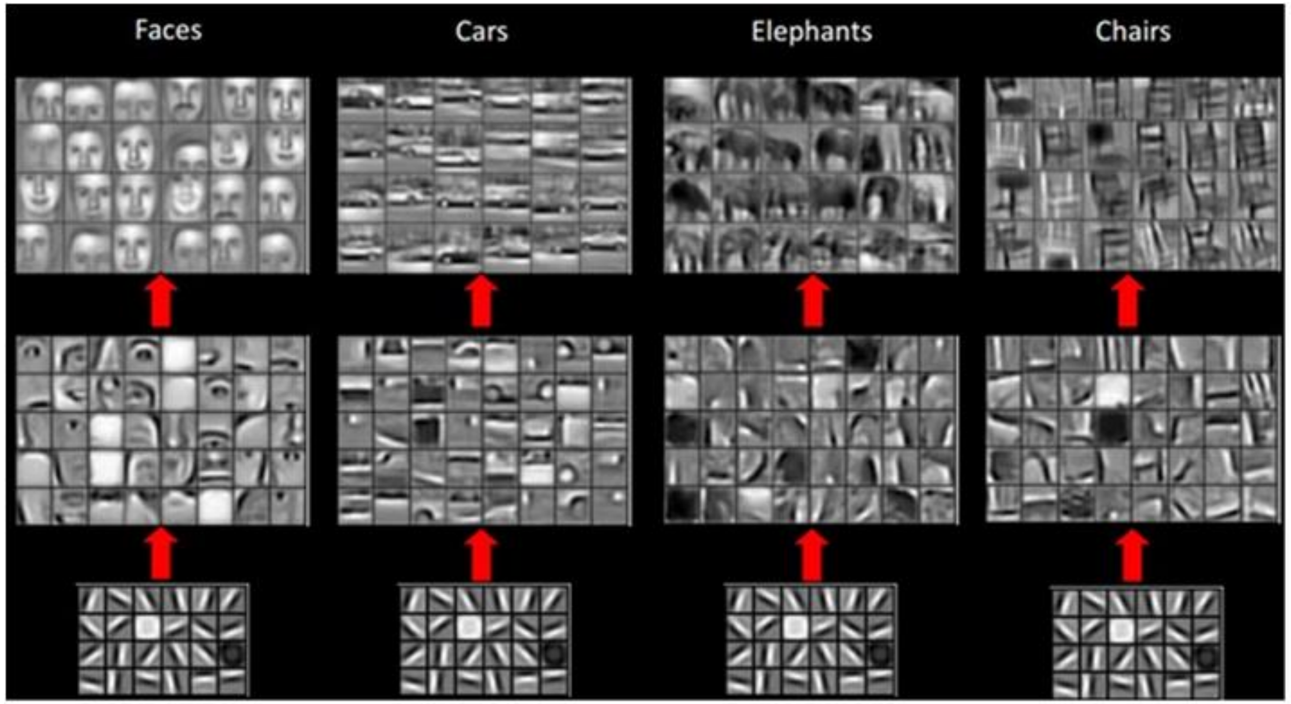
Neural Networks

Neural Networks

- Modeled after the human brain
- CNNs repeatedly perform convolutions and nonlinearity operations

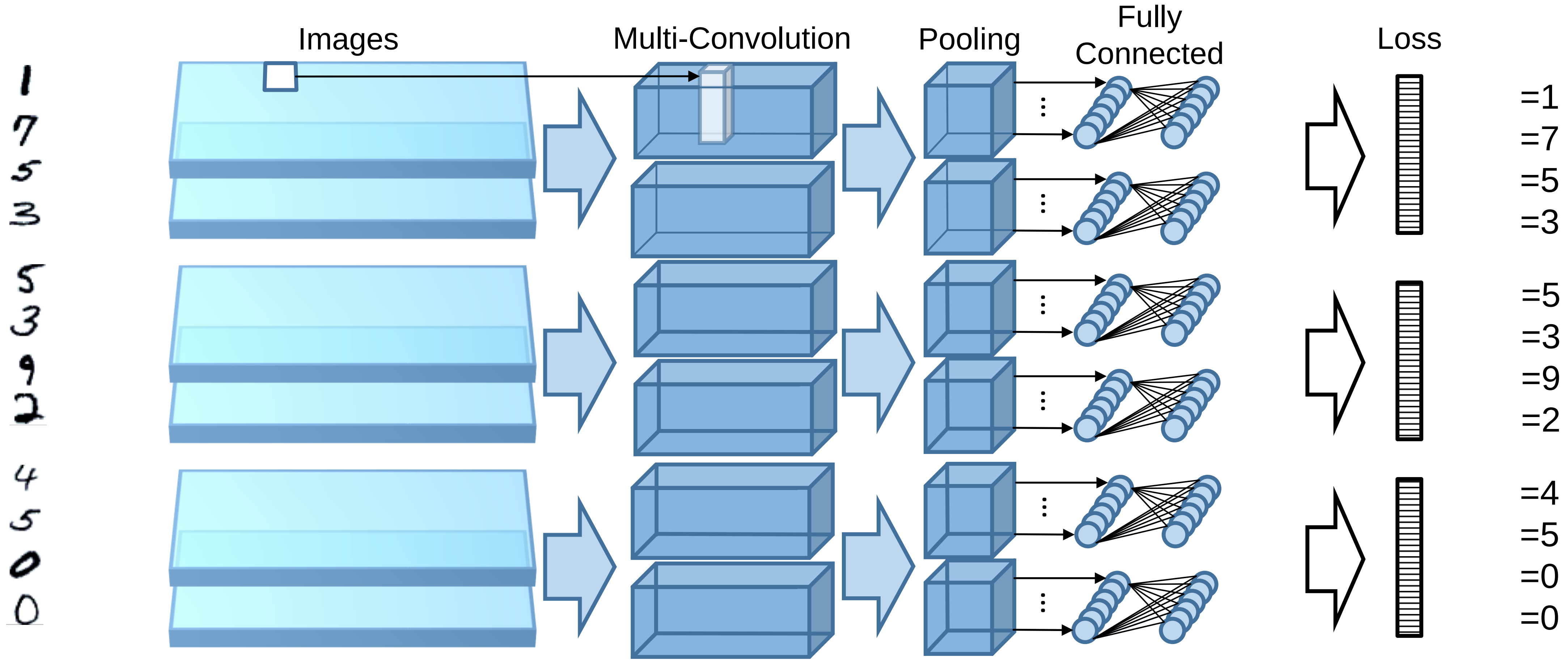


Source: CS231n



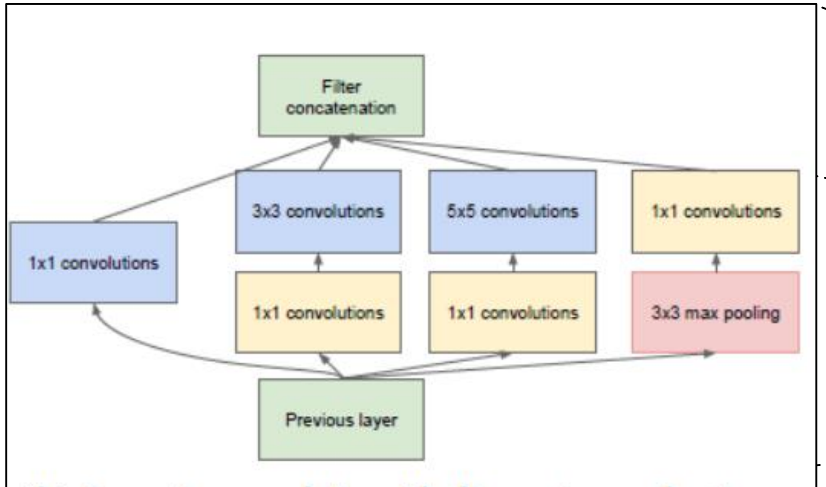
Source: Lee et al. "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks" (CACM 2011)

Simple CNN Architecture

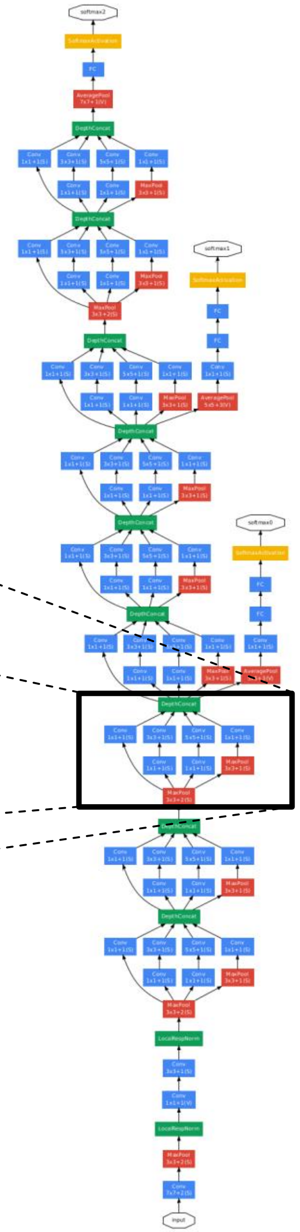


GoogLeNet [Szegedy et al., 2014]

- ~6.8M parameters
- 22 layers deep



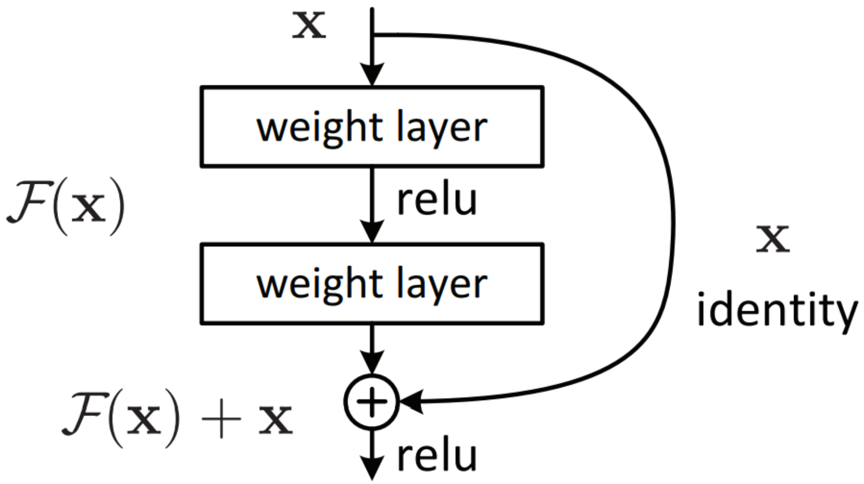
(b) Inception module with dimension reductions

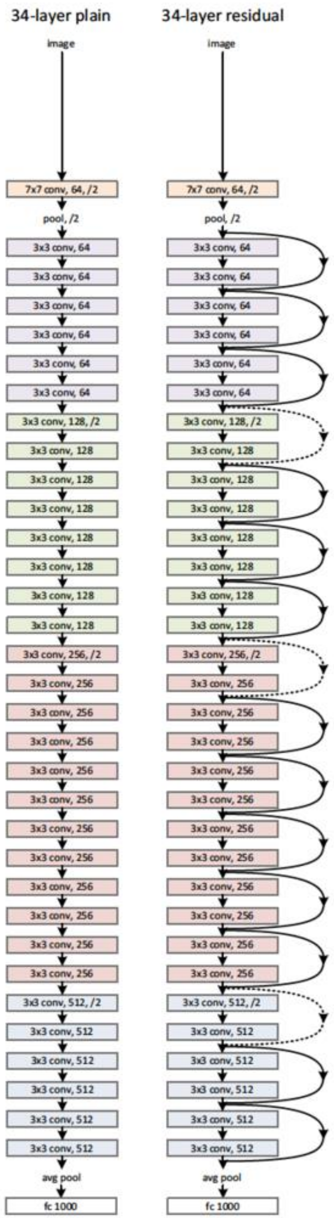


[Szegedy et al. "Going deeper with convolutions." (2014).]()

ResNet [He et al., 2016]

- ~2.35M parameters
- 152 layers deep





He et al. "Deep Residual Learning for Image Recognition." CVPR 2016.

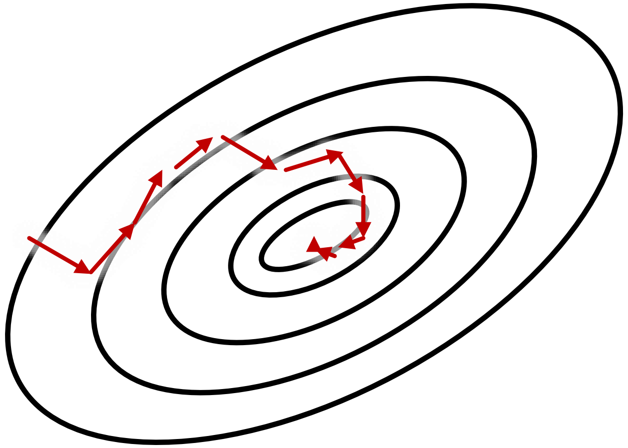
Stochastic Gradient Descent

- Gist: Improve network weights using samples from a labeled dataset

- Algorithm:
  - Initialize neural network weights (W_0)
  - For t in iterations:
    - **Sample** b images from dataset (B)
    - **Compute** loss $L_t(W_{t-1}, B)$
    - **Update** weights using gradients and update rule g:
      $W_t = g(W_{t-1}, \nabla L_t(W_{t-1}, B), [hyperparameters \dots])$

- $\nabla L_t(W, B)$ is an average direction of the gradient over a mini-batch of size b:

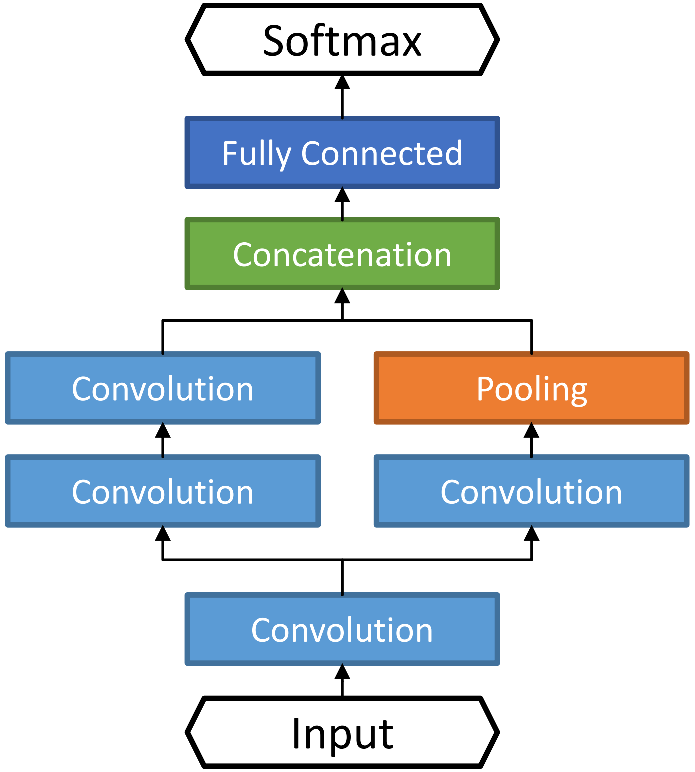
$$\nabla L_t(W, B) = \frac{1}{b} \sum_{i=1}^{b} \nabla \ell(W; (x_i, y_i))$$



Backpropagation Algorithm

- A CNN is a Directed Acyclic Graph (DAG)

- At each layer in backpropagation, derivatives are estimated w.r.t.:
  - Layer parameters (if necessary)
  - Data (chain rule)

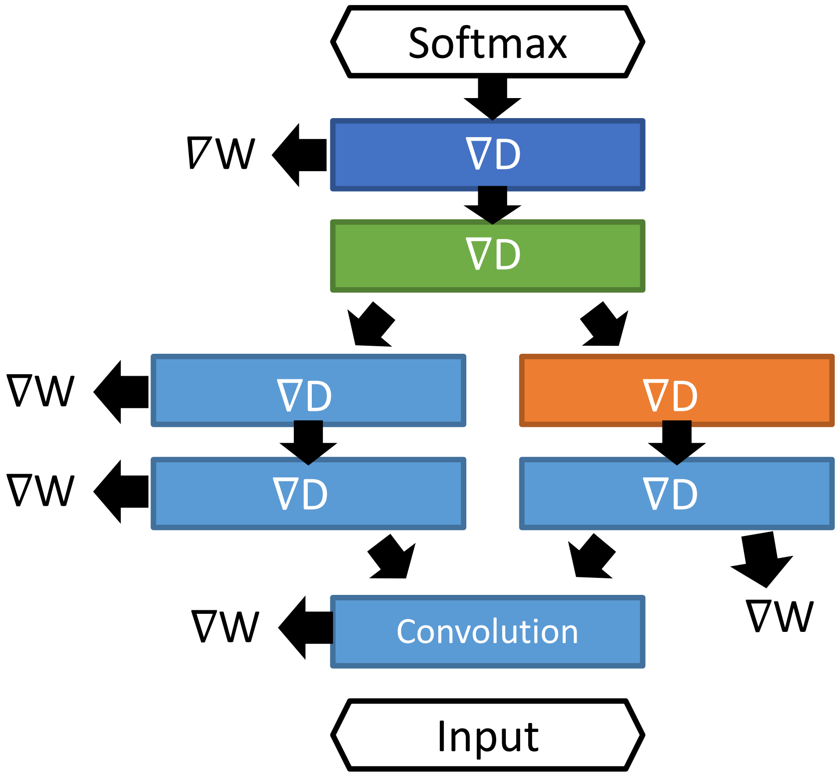


Backpropagation Algorithm

- A CNN is a Directed Acyclic Graph (DAG)

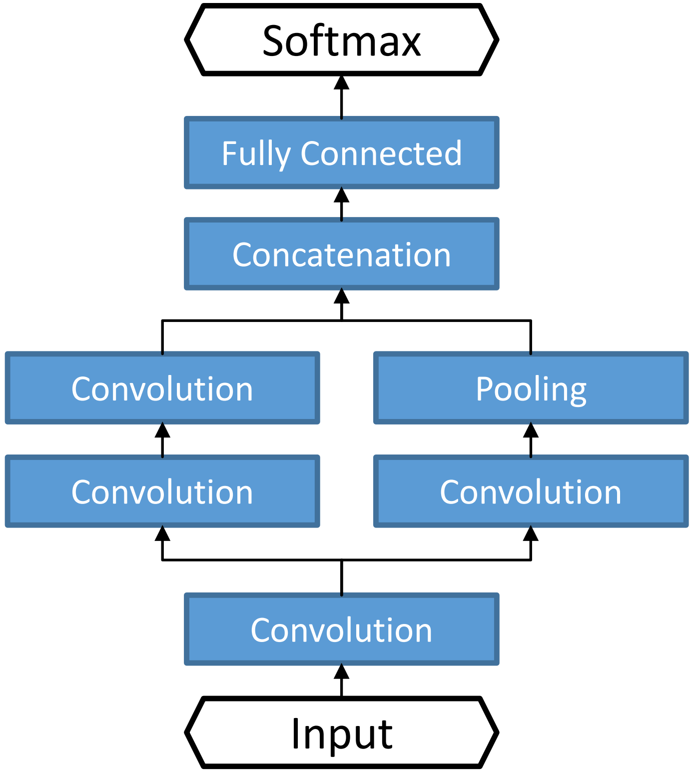
- At each layer in backpropagation, derivatives are estimated w.r.t.:
  - Layer parameters (if necessary)
  - Data (chain rule)

- Additional memory storage required for training:
  - D+W+ **∇D+ ∇W**



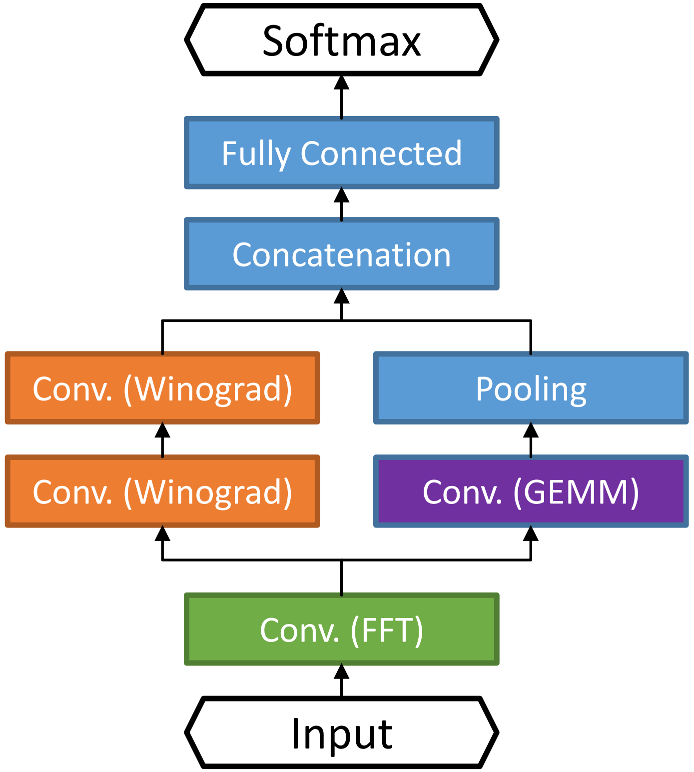
The World of Neural Network Acceleration

- Choice of Algorithm
- Parallelism
- Distributed Computing
- Hardware Architectures



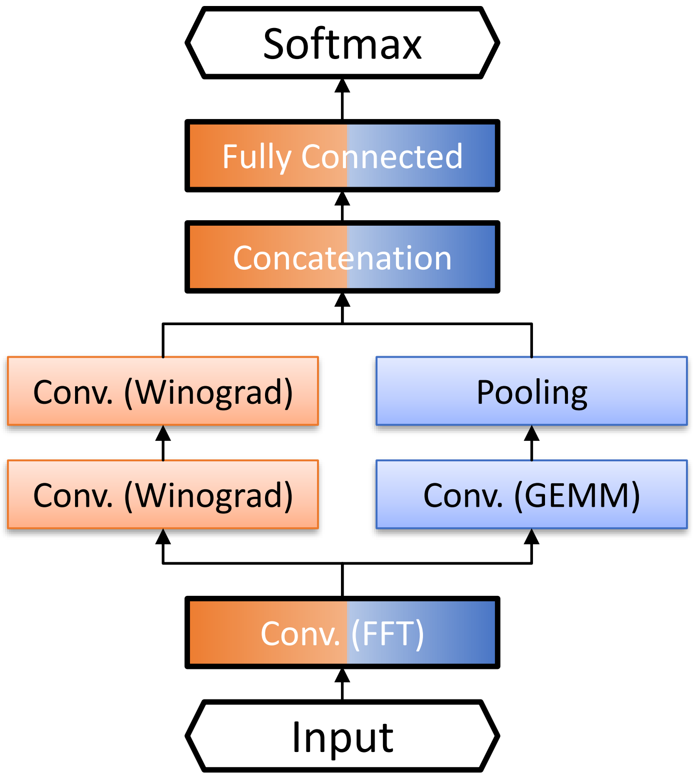
The World of Neural Network Acceleration

- **Choice of Algorithm**
- Parallelism
- Distributed Computing
- Hardware Architectures



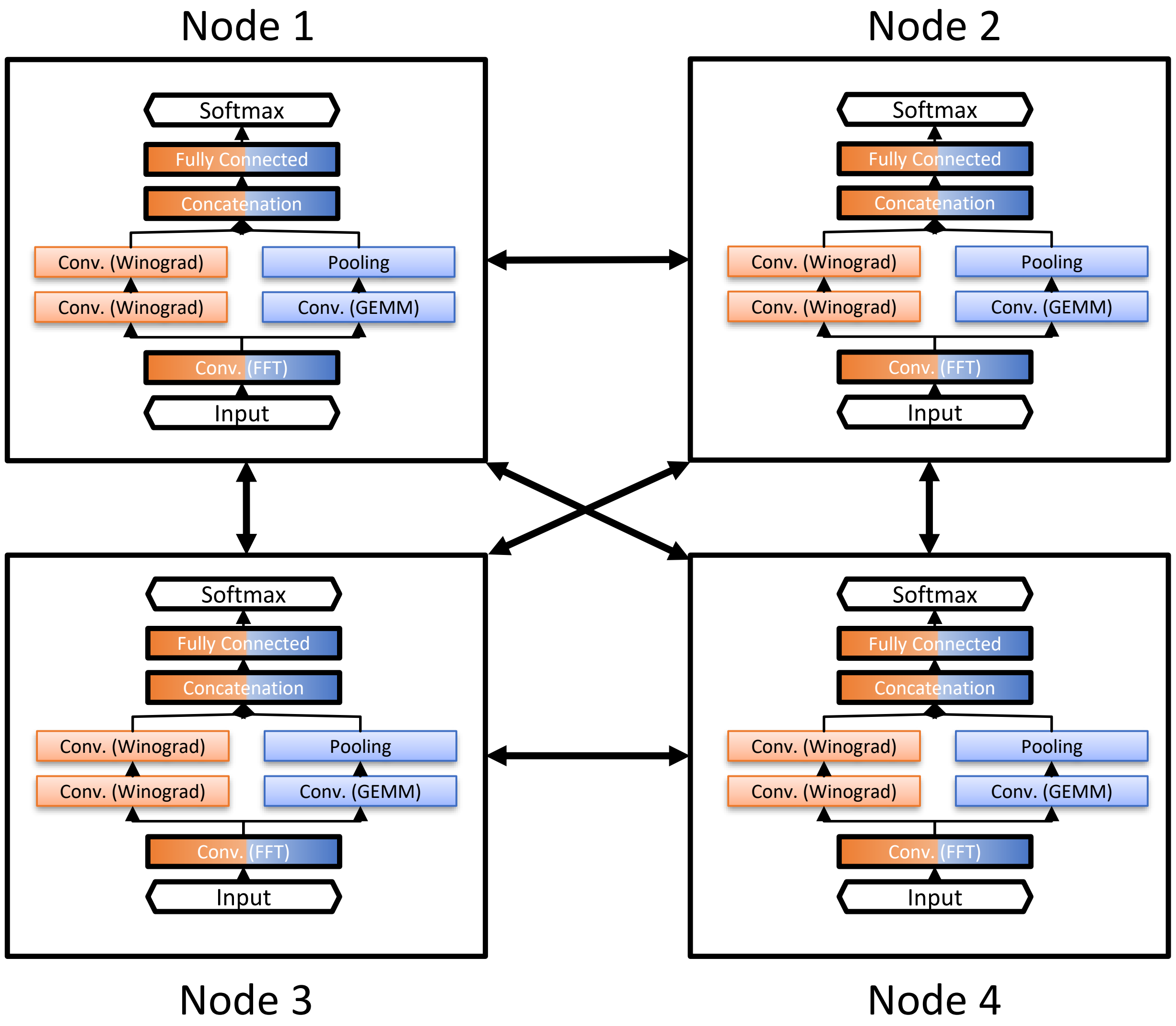
The World of Neural Network Acceleration

- Choice of Algorithm
- **Parallelism**
- Distributed Computing
- Hardware Architectures



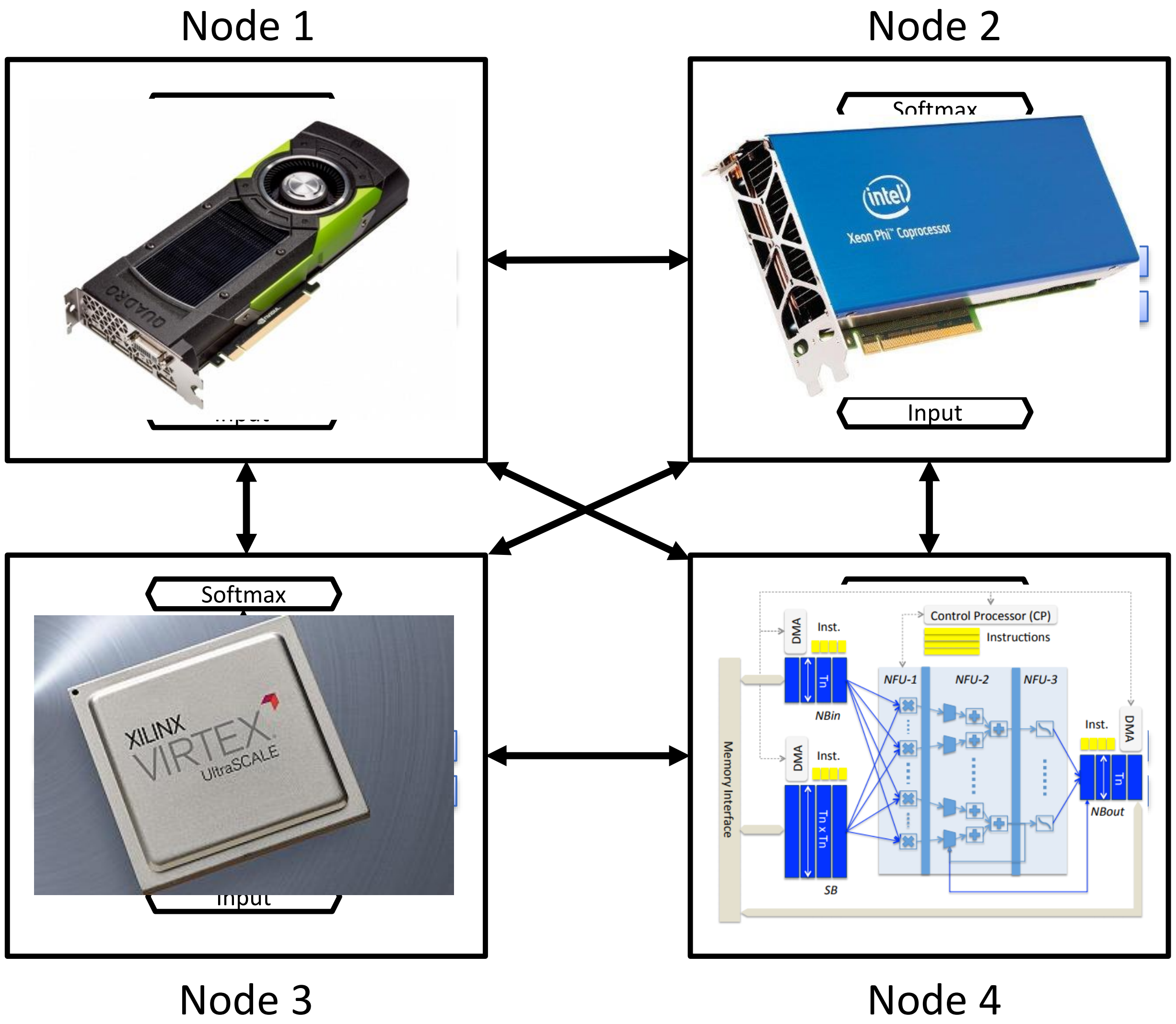
The World of Neural Network Acceleration

- Choice of Algorithm
- Parallelism
- **Distributed Computing**
- Hardware Architectures

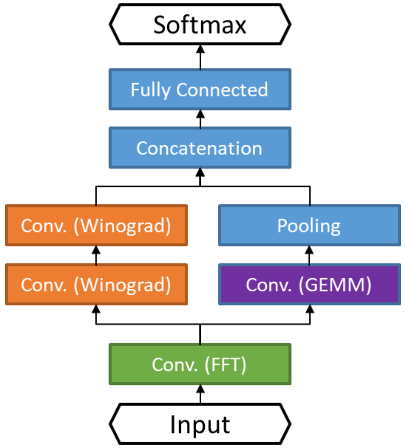


The World of Neural Network Acceleration

- Choice of Algorithm
- Parallelism
- Distributed Computing
- **Hardware Architectures**



Algorithms

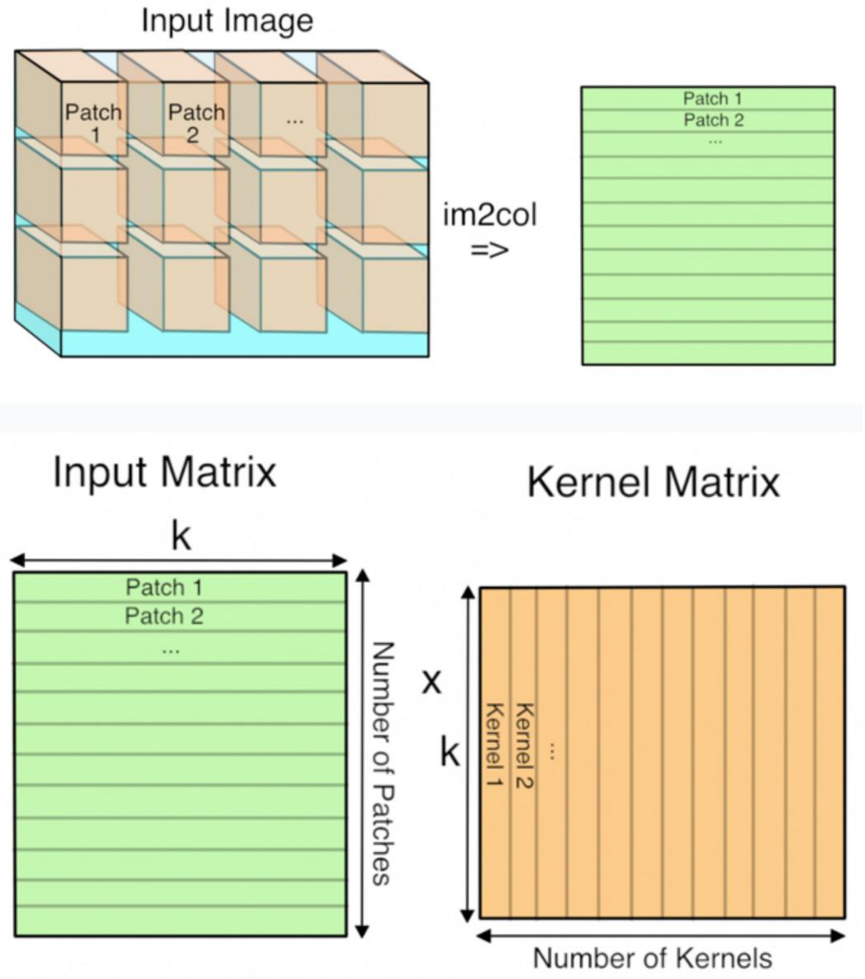

Softmax
Fully Connected
Concatenation
Conv. (Winograd)
Pooling
Conv. (Winograd)
Conv. (GEMM)
Conv. (FFT)
Input

Convolution Algorithms

- Most computationally-intensive layer

$$out(x,y)^{f_o} = \sum_{f_i=0}^{N_{if}} \sum_{k_x=0}^{K_x} \sum_{k_y=0}^{K_y} w_{f_i,f_o}(k_x,k_y) * in(x+k_x, y+k_y)^{f_i}$$

- Can be performed directly, or:
  - Via matrix multiplication (im2col) [Chellapilla et al., 2006]
  - Via Winograd convolution [Lavin and Gray, 2016]
  - In Fourier domain



Source: Pete Warden's Blog

im2col convolution

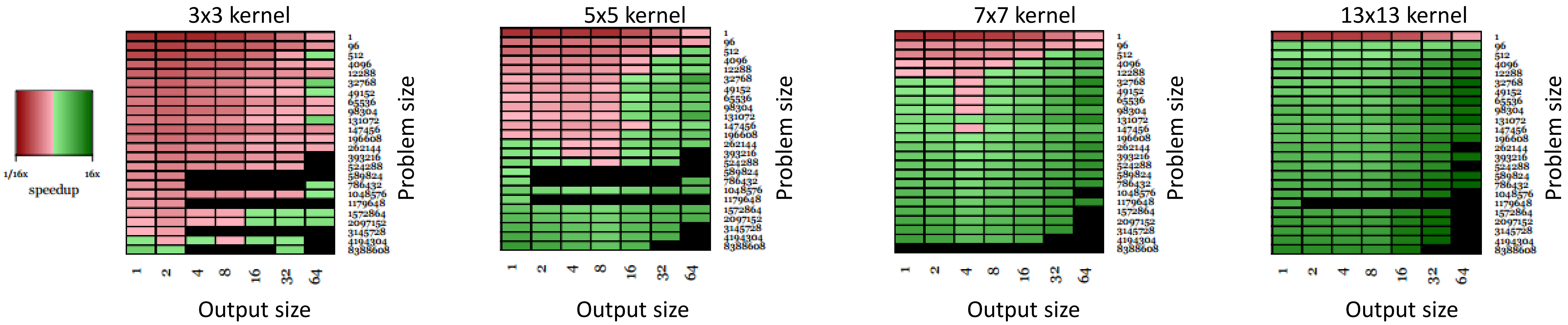
Chellapilla et al. "High Performance Convolutional Neural Networks for Document Processing." (2006)
Lavin and Gray. "Fast Algorithms for Convolutional Neural Networks." CVPR 2016.

Convolution in Fourier Domain

- Convolution can be computed using FFT [Mathieu et al., 2014]:

$$y_{(s,j)} = \sum_{i \in f} x_{(s,i)} \star w_{(j,i)} = \sum_{i \in f} \mathcal{F}^{-1}\left(\mathcal{F}(x_{(s,i)}) \circ \mathcal{F}(w_{(j,i)})^{*}\right)$$

- The larger the convolution kernel, the better the performance [Vasilache et al., 2015]:



Mathieu et al. “Fast Training of Convolutional Networks through FFTs”, ICLR 2014
Vasilache et al. "Fast Convolutional Nets With fbfft: A GPU Performance Evaluation", ICLR 2015

Sacrificing Accuracy for Performance

- Half-precision (16-bit floating point) [Gupta et al., 2015]
  - Memory is stored in 16-bit format
  - Computations are performed in 32-bits
  - Uses Stochastic Rounding:

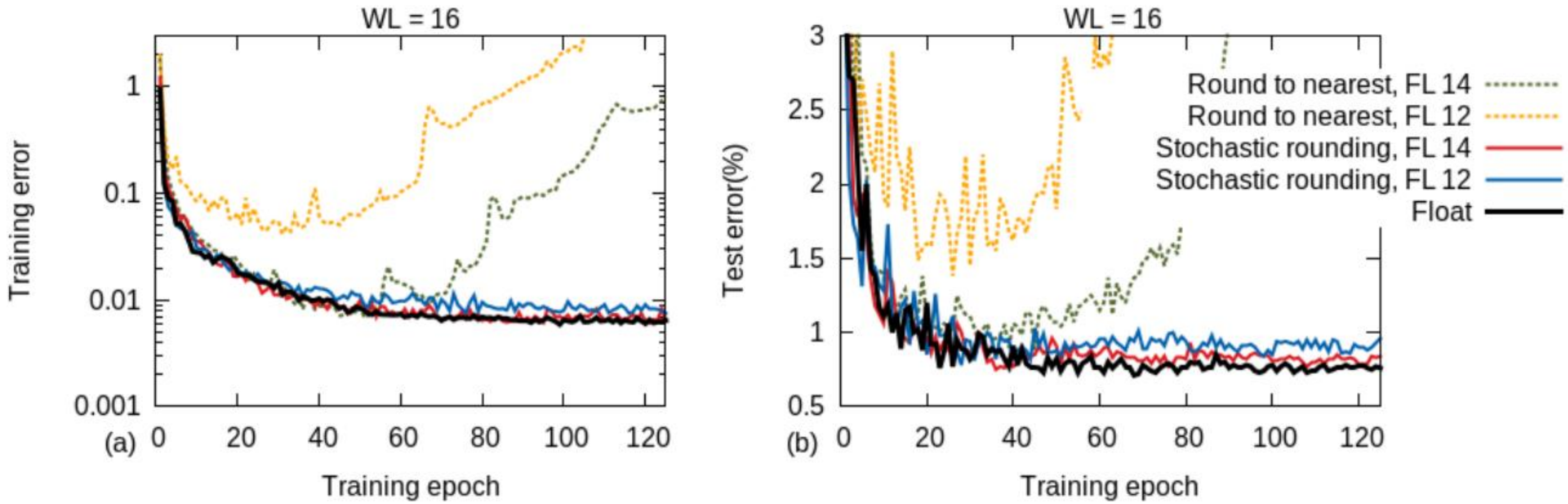
$$Round\left(x, \langle \mathrm{IL}, \mathrm{FL} \rangle\right) = \begin{cases} \lfloor x \rfloor & \text{w.p. } 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & \text{w.p. } \frac{x - \lfloor x \rfloor}{\epsilon} \end{cases}$$

Goal: *Preserve* $\mathbb{E}\left(Round\left(x, \langle \mathrm{IL}, \mathrm{FL} \rangle\right)\right) = x$

Gupta et al. "Deep Learning with Limited Numerical Precision." NIPS 2015.

Sacrificing Accuracy for Performance

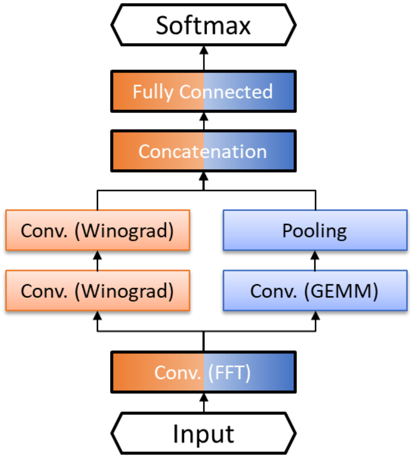
- Results on MNIST with LeNet:



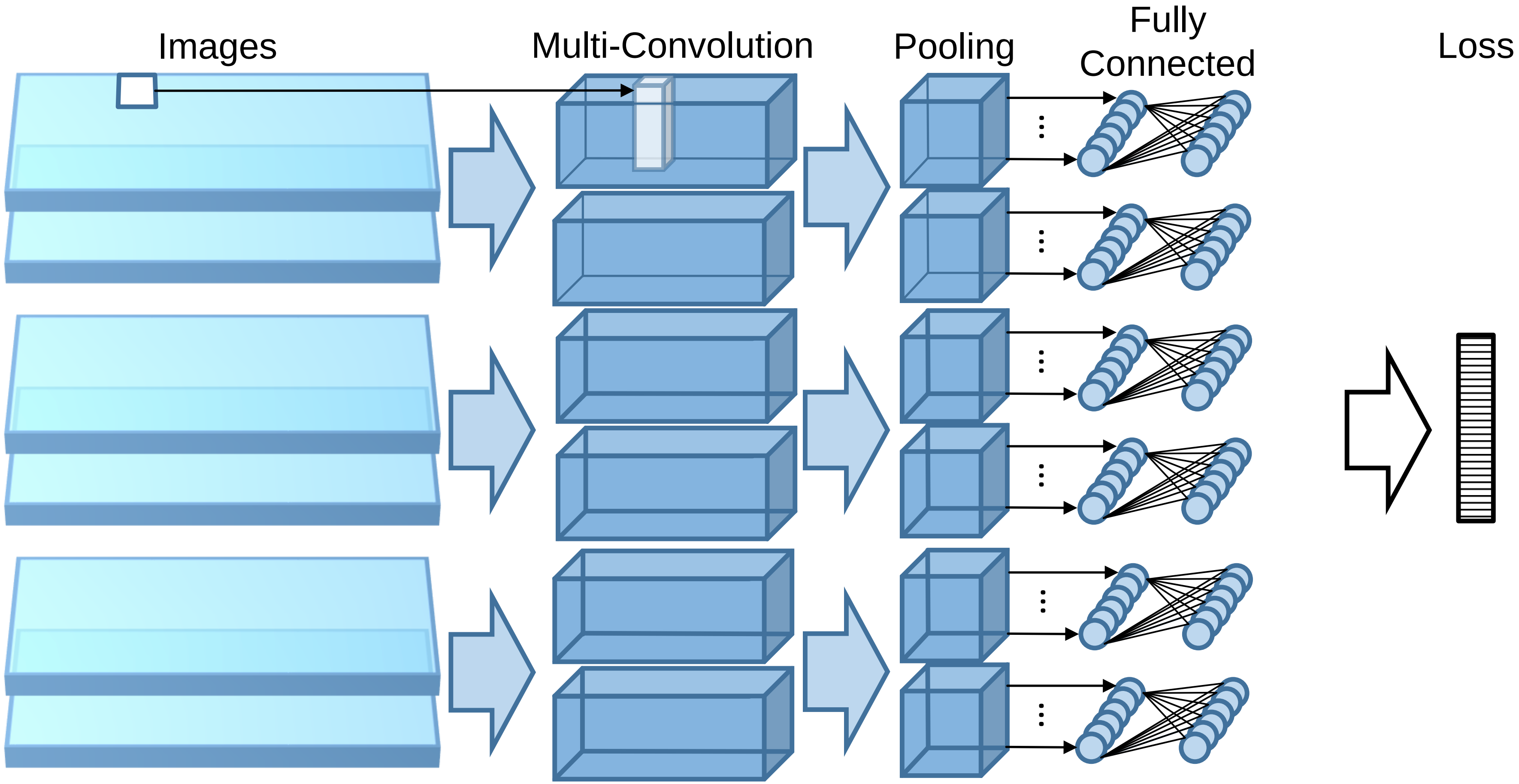
WL=Word Length (bits)
FL=Fractional Length (bits)

Gupta et al. "Deep Learning with Limited Numerical Precision." NIPS 2015.

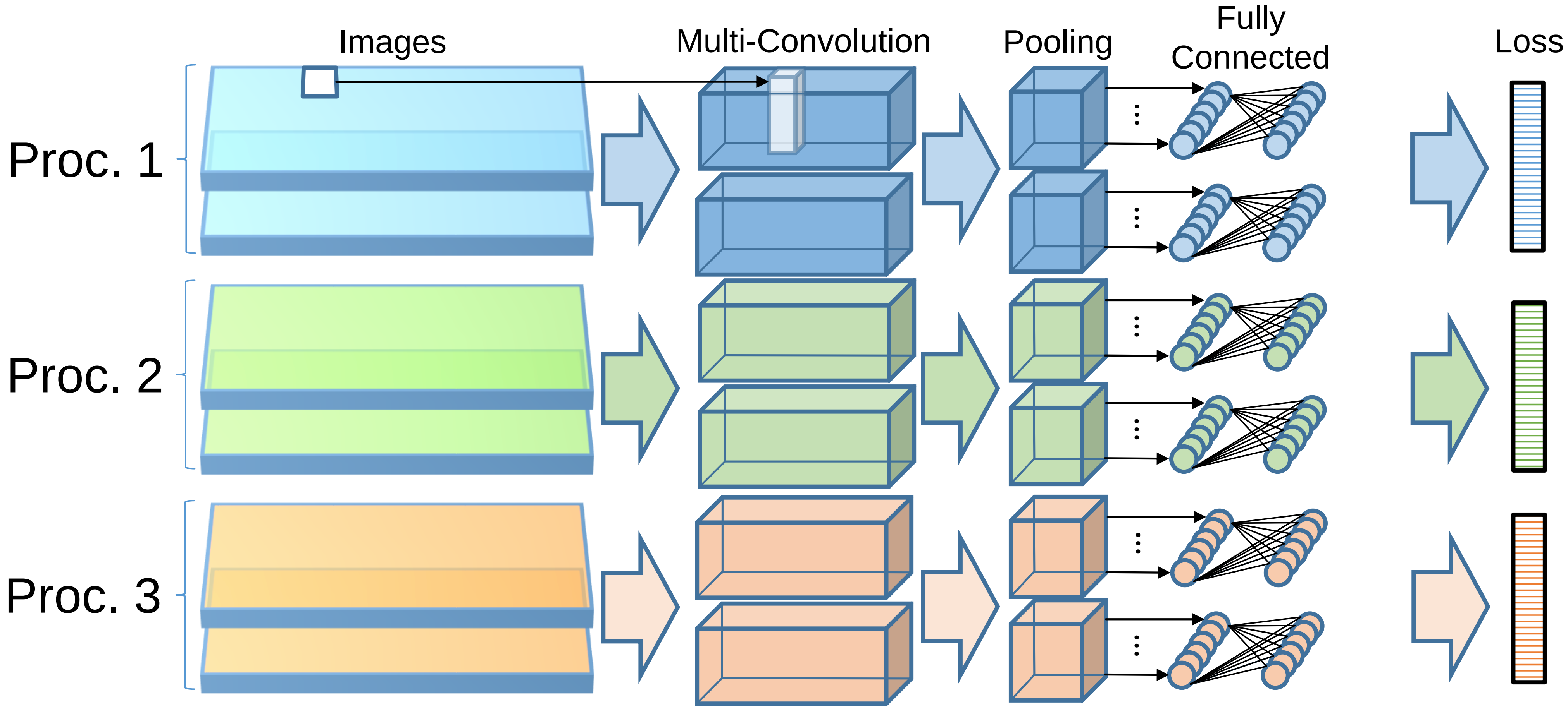
Parallelism



Data Parallelism



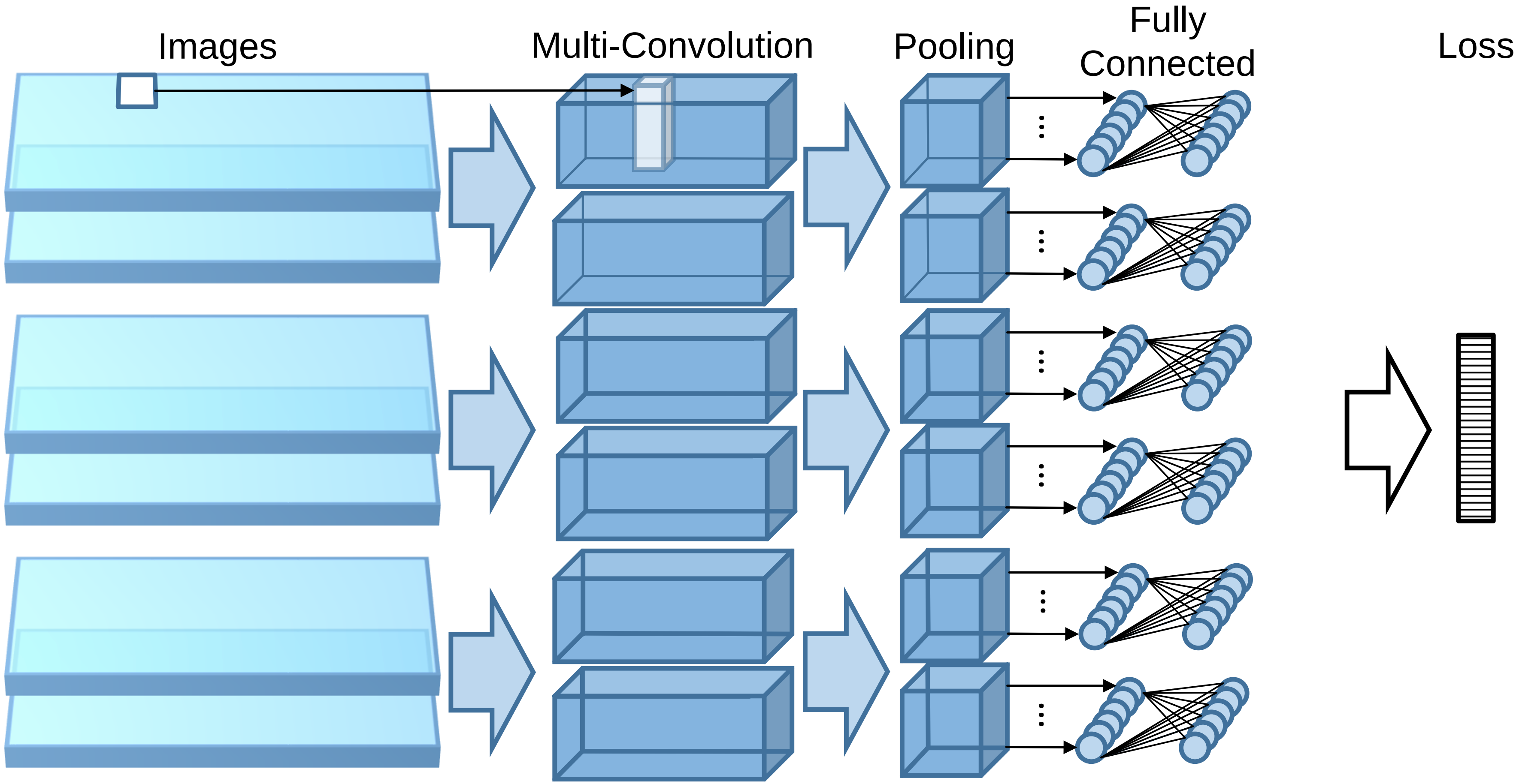
Data Parallelism



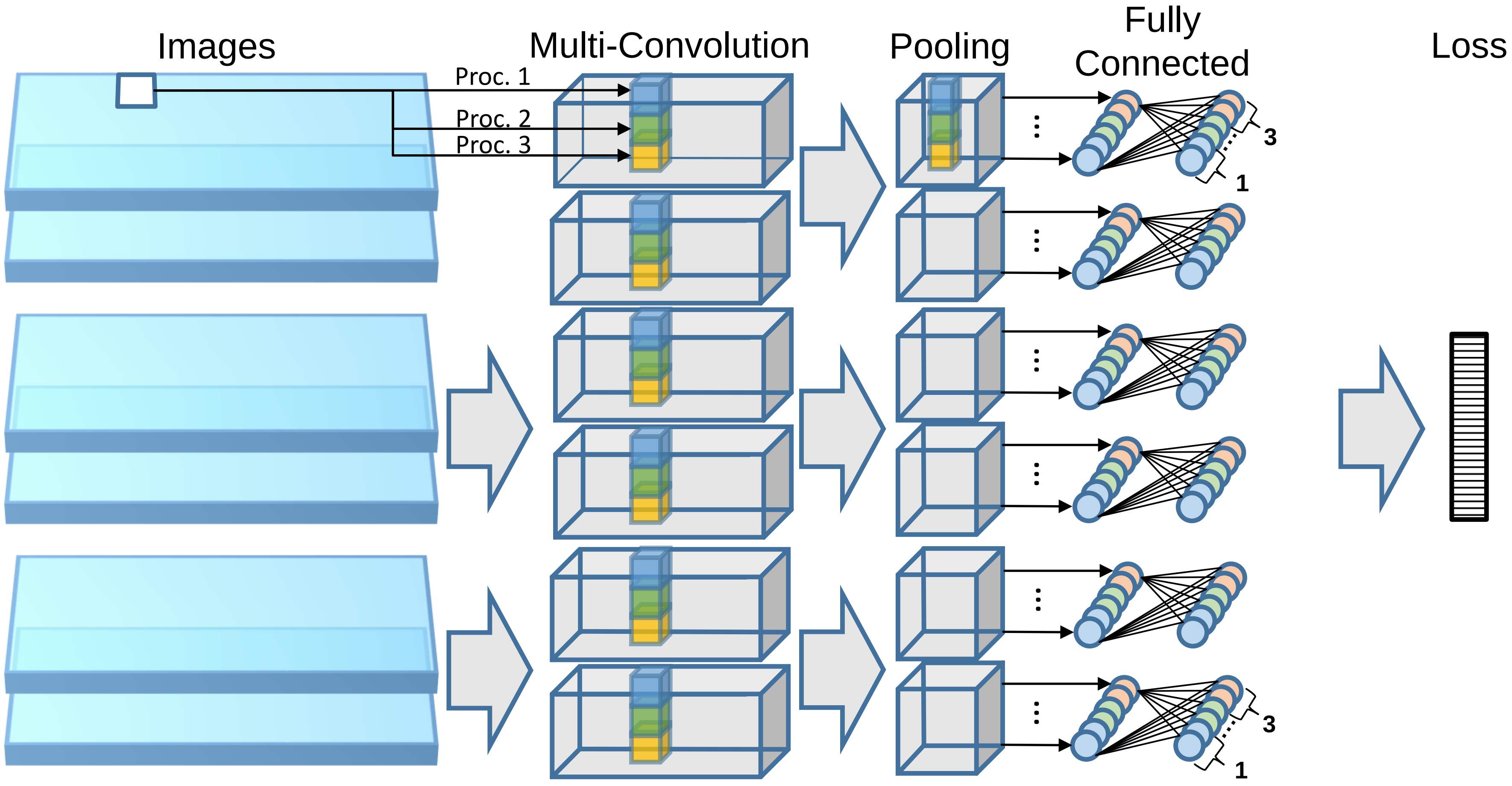
Data Parallelism

- ✓ Good for forward pass (independent)
- ✓ Backpropagation requires all-to-all communication only when accumulating results
- × Requires allocation of all parameters on each processor

Model Parallelism



Model Parallelism



Model Parallelism

- ✓ Parameters can be divided across processors
- × Mini-batch has to be copied to all processors
- × Backpropagation requires all-to-all communication every layer

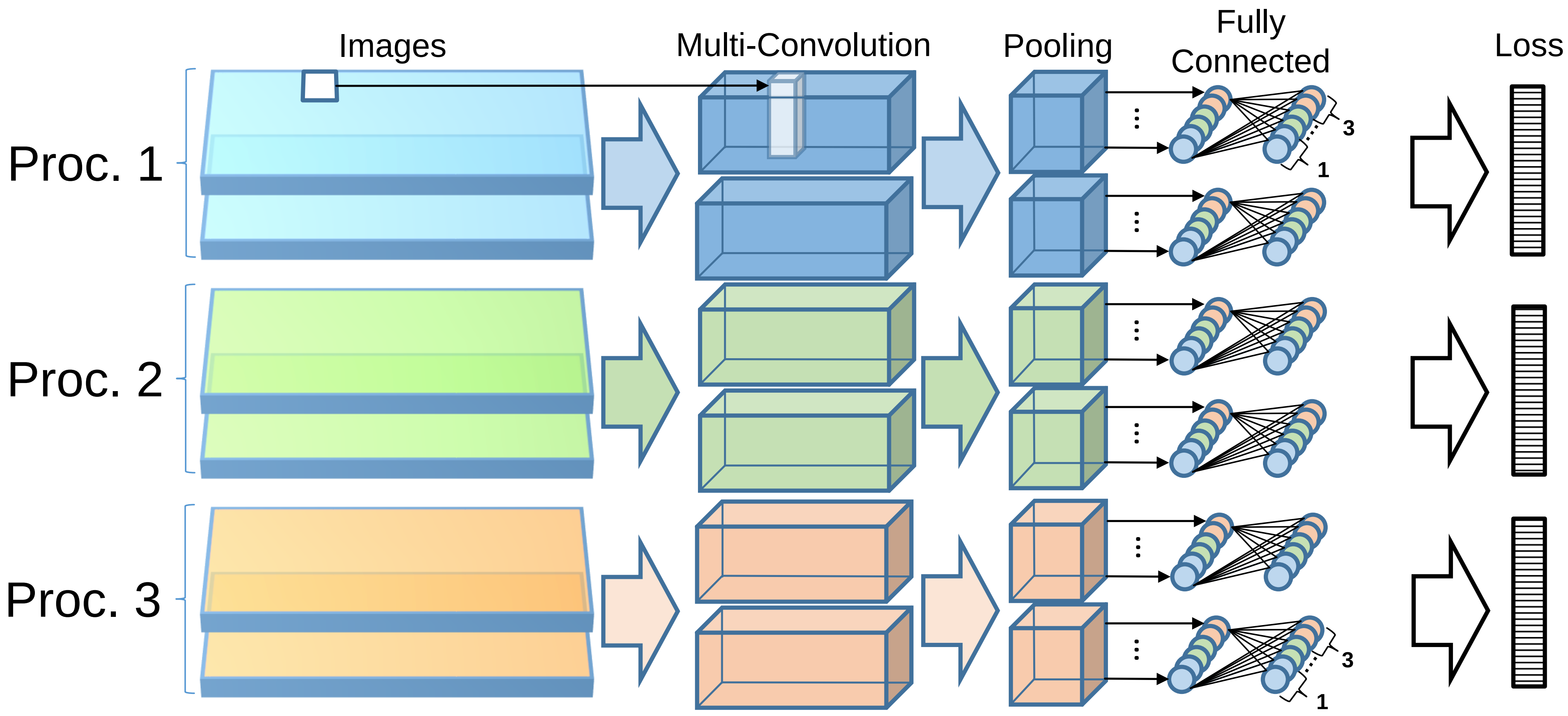
Hybrid Data/Model Parallelism

- Conjecture[Krizhevsky, 2014]: Most of the **computations** are performed in the convolutional portion, most of the **parameters** are stored in the fully connected portion

- Proposed Solution: Use data parallelism on convolutional portion and model parallelism on the FC portion

Krizhevsky. "One weird trick for parallelizing convolutional neural networks." (2014).

Hybrid Data/Model Parallelism [Krizhevsky, 2014]



Krizhevsky. "One weird trick for parallelizing convolutional neural networks." (2014).

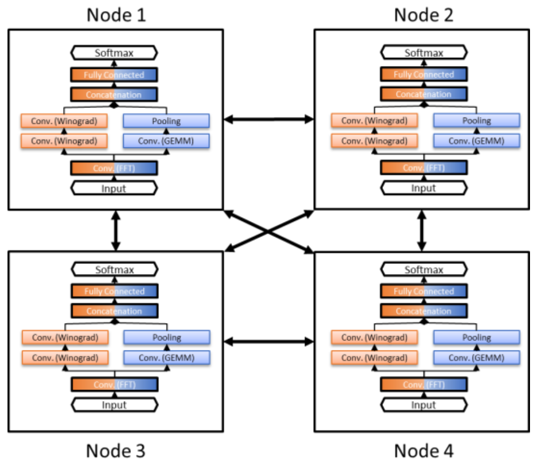
Hybrid Data/Model Parallelism Results

- AlexNet, ILSVRC 2012:

GPUs	Batch size	Top-1 error	Time	Speedup
1	(128, 128)	42.33%	98.05h	1x
2	(256, 256)	42.63%	50.24h	1.95x
2	(256, 128)	42.27%	50.90h	1.93x
4	(512, 512)	42.59%	26.20h	3.74x
4	(512, 128)	42.44%	26.78h	3.66x
8	(1024, 1024)	43.28%	15.68h	6.25x
8	(1024, 128)	42.86%	15.91h	6.16x

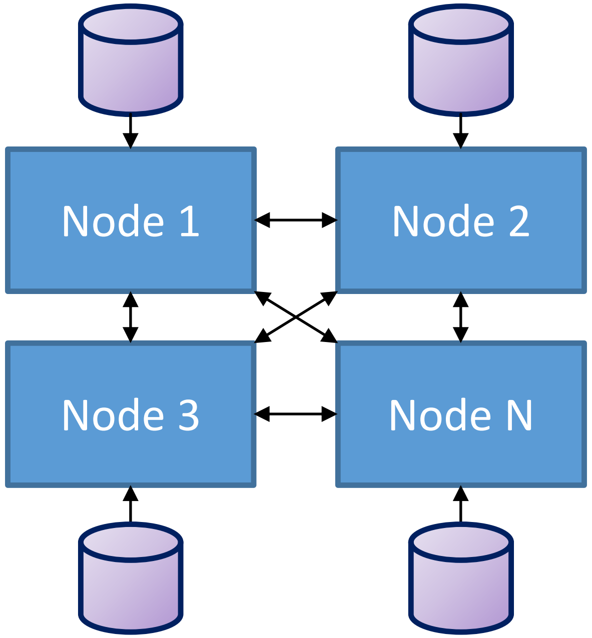
Krizhevsky. "One weird trick for parallelizing convolutional neural networks." (2014).

Distributed Computing



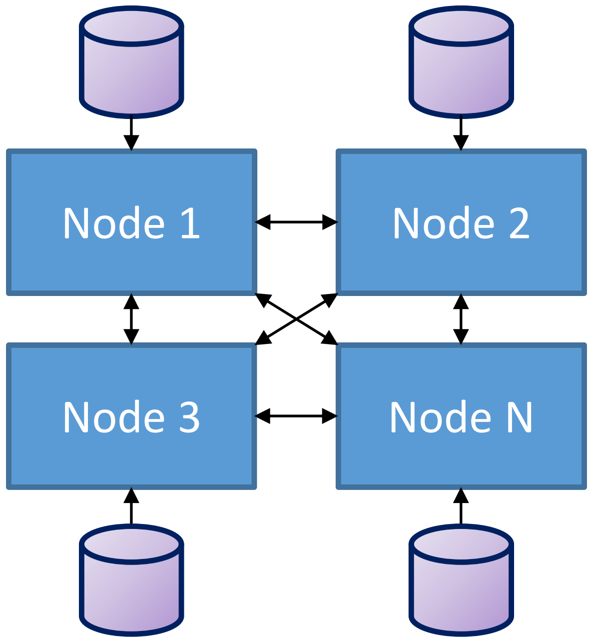
Distributed Deep Learning

- Runs on a computer cluster
- Each node runs partially autonomously
- Inter-node communication from time to time
- Best result is gathered from the nodes
- Training data can be split to per-node “shards”



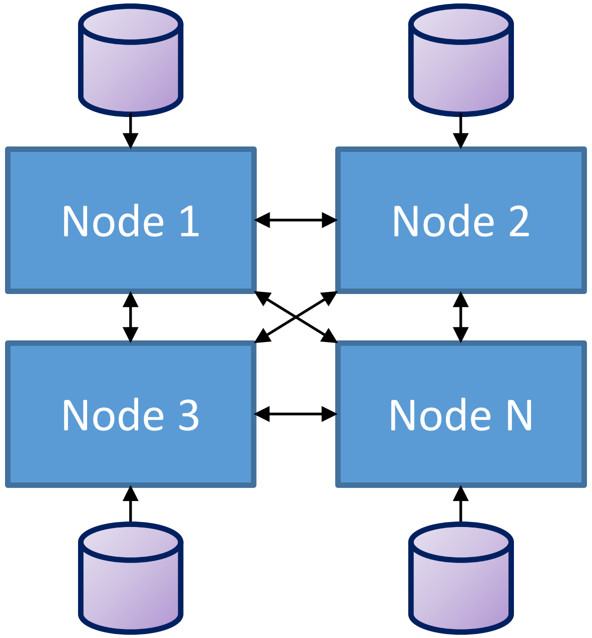
Distributed Deep Learning – Opportunities

- Increased memory:
  - More data
  - More parameters
- Fault tolerance
  - Protection against node crashes
- Improved stochasticity



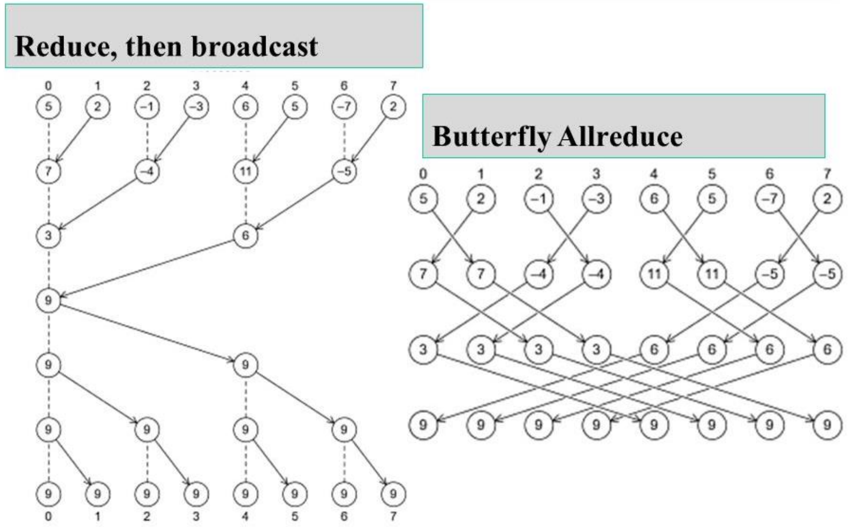
Distributed Deep Learning – Determining Factors

- Computational independence
- Communication efficiency
- Network congestion
- Load balancing
- Points of failure



Distributed Synchronous SGD

- Communication step is added to the algorithm:
  - Initialize neural network weights (W_0)
  - For t in iterations:
    - Sample *b* images from dataset (B)
    - Compute loss $L_t(W_{t-1}, B)$
    - **Synchronize weights across workers**
    - Update weights using gradients and update rule *g*



Source: [Basics of Message-Passing]

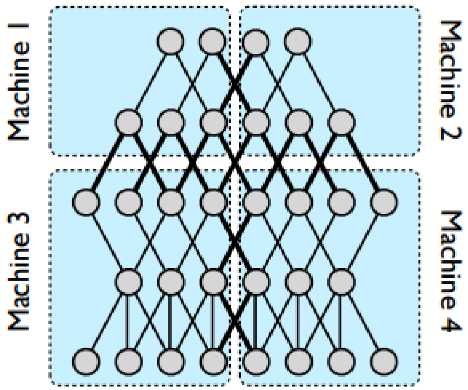
- The step requires all nodes to have the same data ($\sum \nabla W$)
  - This collective operation is also called AllReduce

- Different ways to implement, depending on message size and network topology

Distributed Deep Learning – DistBelief

- Distributed learning infrastructure used at Google [Dean et al., 2012]

- Each model **replica** has the same parameters, but optimizes different data
  - Replicas are divided among several machines



- Two distributed optimization schemes for training:
  - Online – Downpour SGD
  - Batch – Sandblaster LBFGS

- Uses a centralized parameter server (several machines, sharded)

- Handles slow and faulty replicas

Dean et al. "Large scale distributed deep networks." *NIPS* 2012.

Asynchronous SGD – HOGWILD!

- To achieve coherency in distributed SGD, nodes must synchronize w.r.t. parameters:

  - Each thread draws a random example i from the training data.
    - Acquire a lock on the current state of parameters θ.
    - Thread reads θ.
    - Thread updates $\theta \leftarrow (\theta - \alpha \nabla L(f_\theta(x_i), y_i))$.
    - Release lock on θ.

- *HOGWILD!* [Niu et al., 2011] removes this synchronization:

  Each thread draws a random example i from the training data.

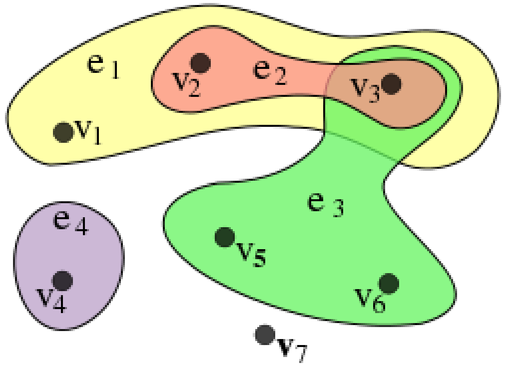
  - Thread reads current state of θ.
  - Thread updates $\theta \leftarrow (\theta - \alpha \nabla L(f_\theta(x_i), y_i))$.

Niu et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent." *NIPS* 2011.

Asynchronous SGD – HOGWILD!

- *HOGWILD!*:
  - Proven to converge in sparse problems
  - Provides near-linear scaling
  - Assumes shared-memory architecture (e.g., multicore CPUs)

- Formulates ML problems as hypergraphs $G = (V, E)$ where:
  - $w^* = \text{argmin}_w f(w) = \text{argmin}_w \sum_{e \in E} f_e(w_e)$
  - Each hyperedge $e \in \mathrm{E}$ represents subsets of $[n]$
  - w_e is reduced to coordinates in e



Source: Wikipedia

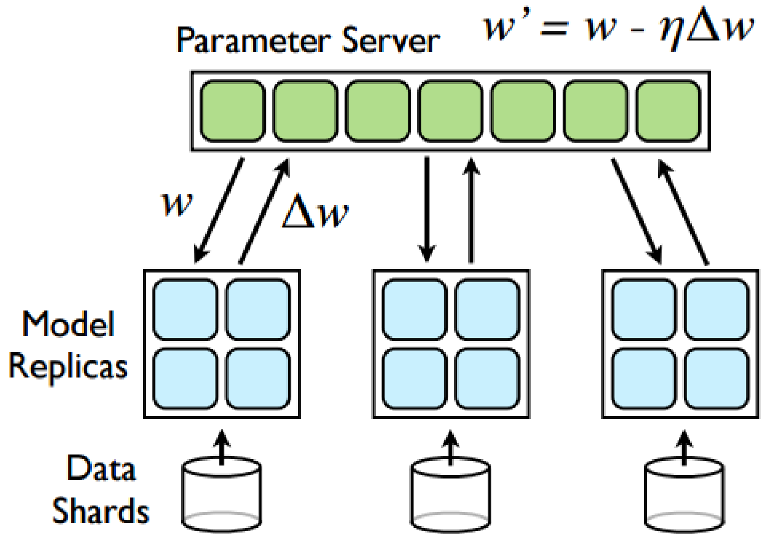
Algorithm 1 HOGWILD! update for individual processors

loop
  Sample e uniformly at random from E
  Read current state x_e and evaluate $G_e(x_e)$
  for $v \in e$ **do** $x_v \leftarrow x_v - \gamma G_{ev}(x_e)$
end loop

Niu et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent." *NIPS* 2011.

Distributed Deep Learning – Downpour SGD

- Relaxation of *HOGWILD!* for distributed systems
- Algorithm:
  - Divide training data into subsets and run a replica on each subset
  - Every n_{fetch} iterations, fetch up-to-date parameters from server
  - Every n_{push} iterations, push local gradients to server
- Note that parameter shards may be "out-of-sync"



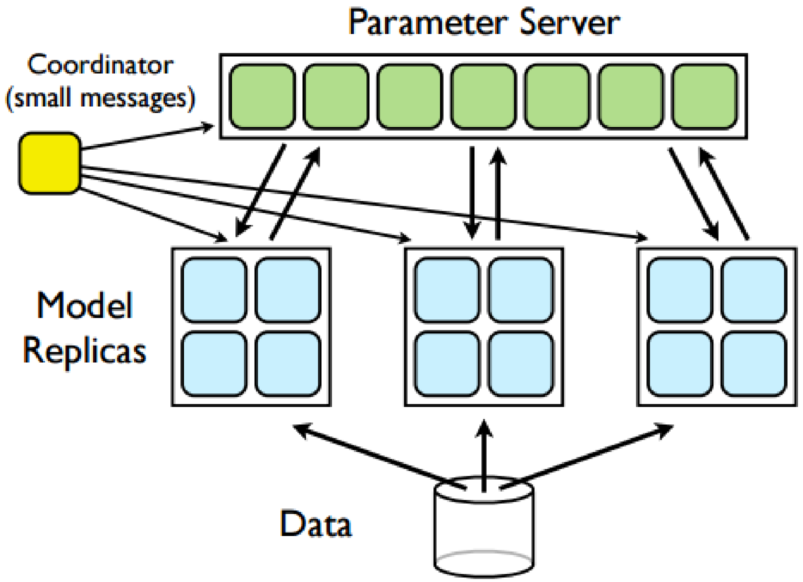
Dean et al. "Large scale distributed deep networks." *NIPS* 2012.

Distributed Deep Learning – Sandblaster LBFGS

- Coordinator process issues commands (dot product, scaling, multiplication, etc.) to slave nodes, each processing a different parameter shard

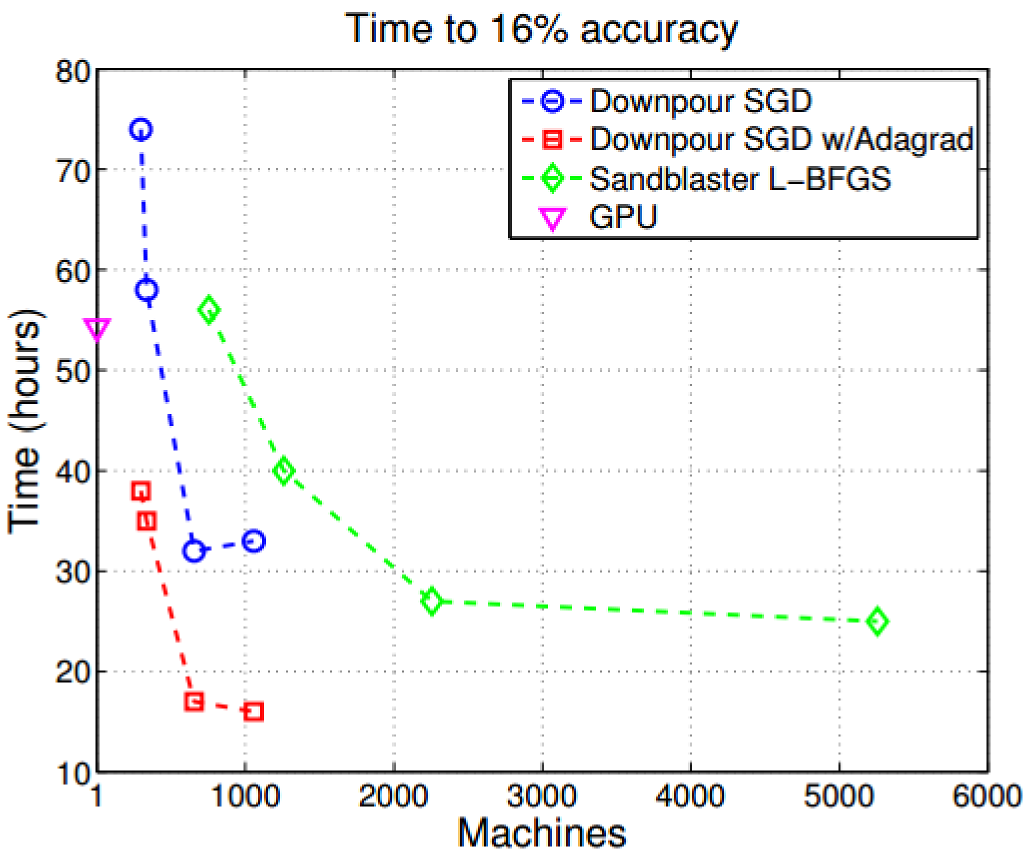
- Communication is sparser
  - Most of the information is stored locally
  - Coordinator messages are small
  - Slaves fetch parameters at the beginning of each batch, send gradients once in a while for fault tolerance

- Employs computation replication and load balancing
  - Nodes that finish their job get more jobs
  - If one node is slow, additional nodes get the same job



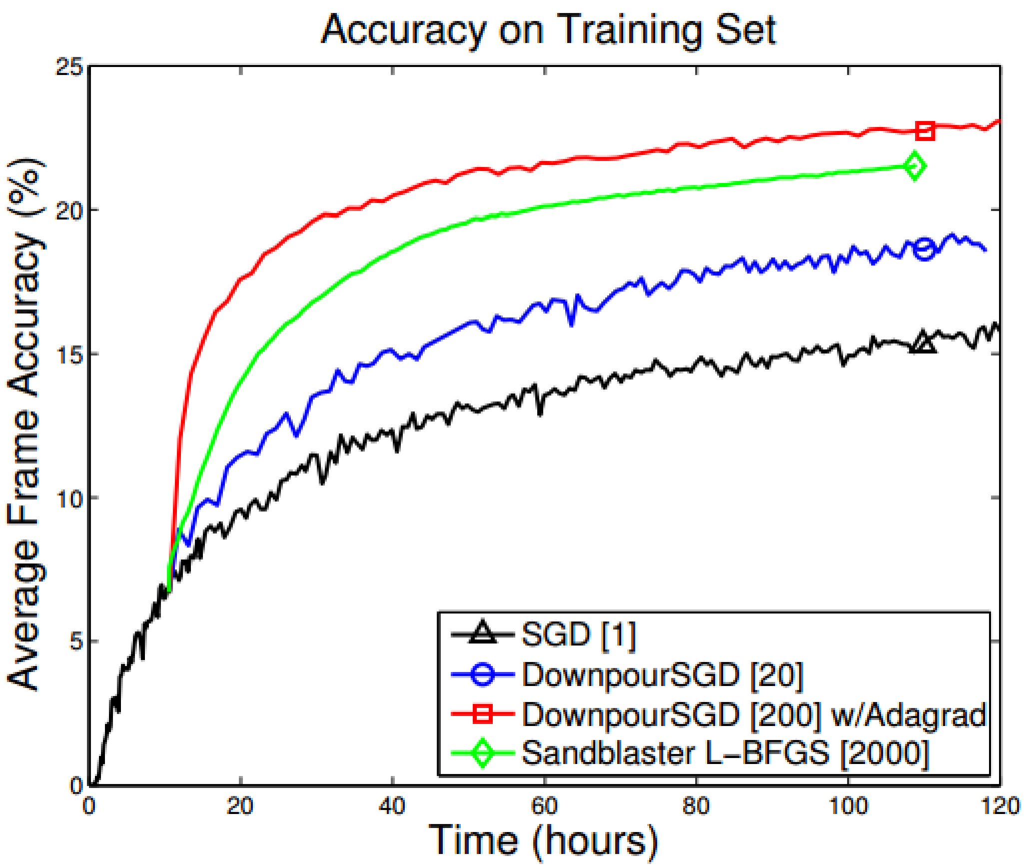
Dean et al. "Large scale distributed deep networks." *NIPS* 2012.

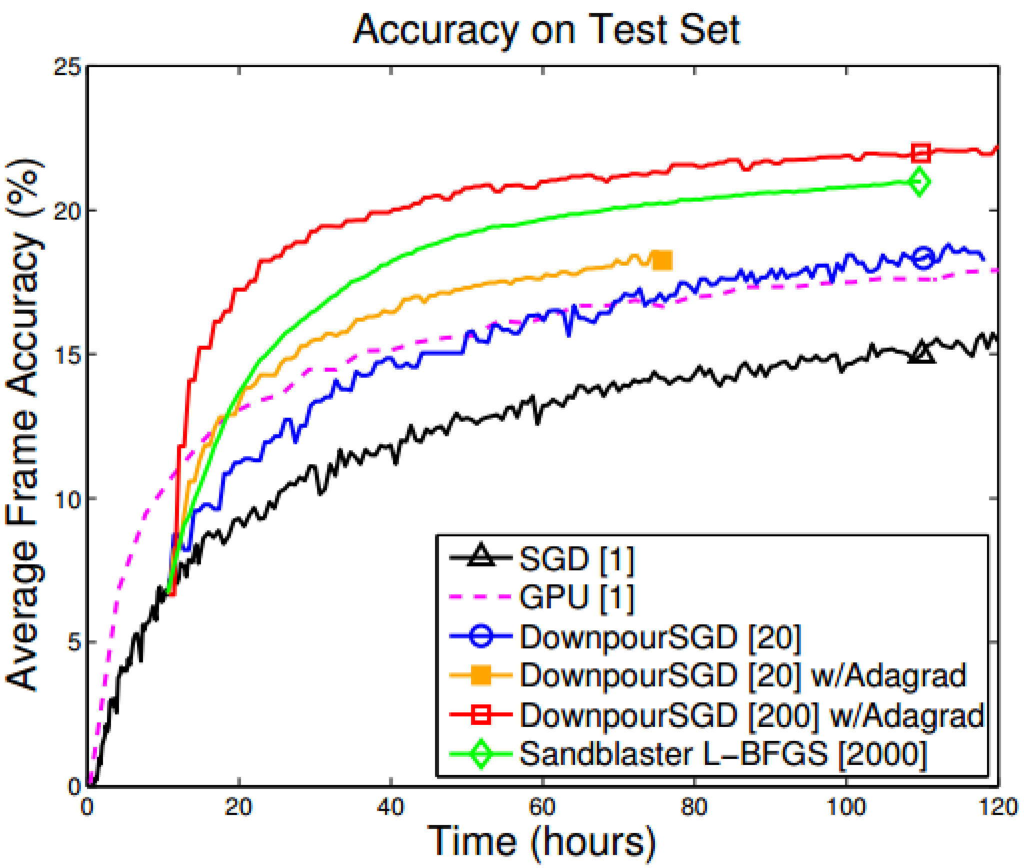
DistBelief Results – Time



Dean et al. "Large scale distributed deep networks." *NIPS* 2012.

DistBelief Results – Accuracy

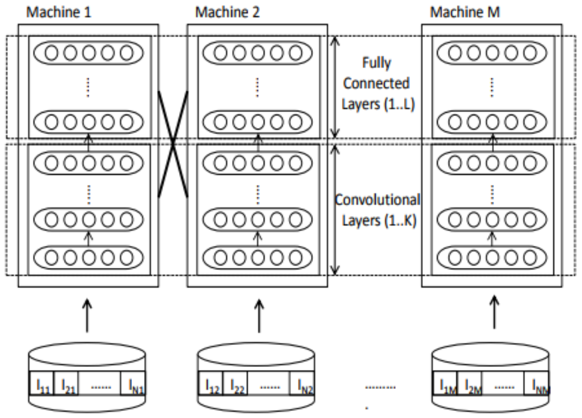




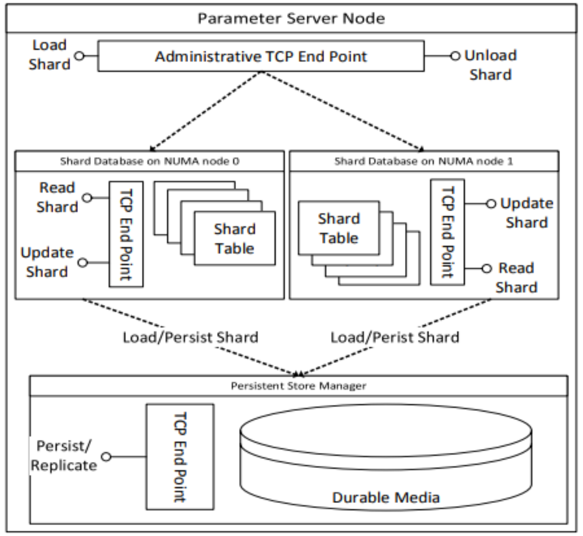
Dean et al. "Large scale distributed deep networks." *NIPS* 2012.

Project Adam

- Extends DistBelief with system-level support
  - Fast data serving mechanisms (e.g., with augmentation)
  - Better heterogeneous system management
  - Parameter server node optimization



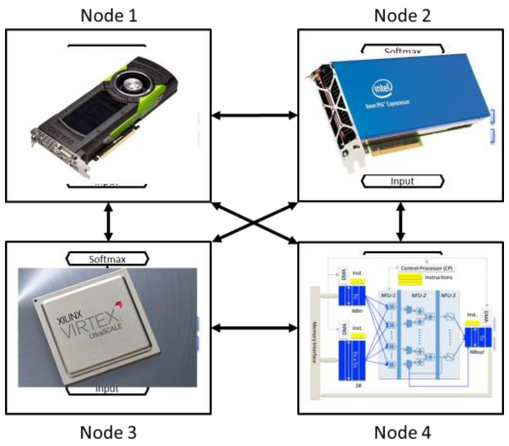
- Bottom-up communication redesign
  - Control message, data message separation
  - Inter-node communication reduction
  - Weight differences are sent instead of weights



- Only system to train ImageNet22k

Chilimbi et al. "Project Adam: Building an Efficient and Scalable Deep Learning Training System." OSDI 2014

Hardware

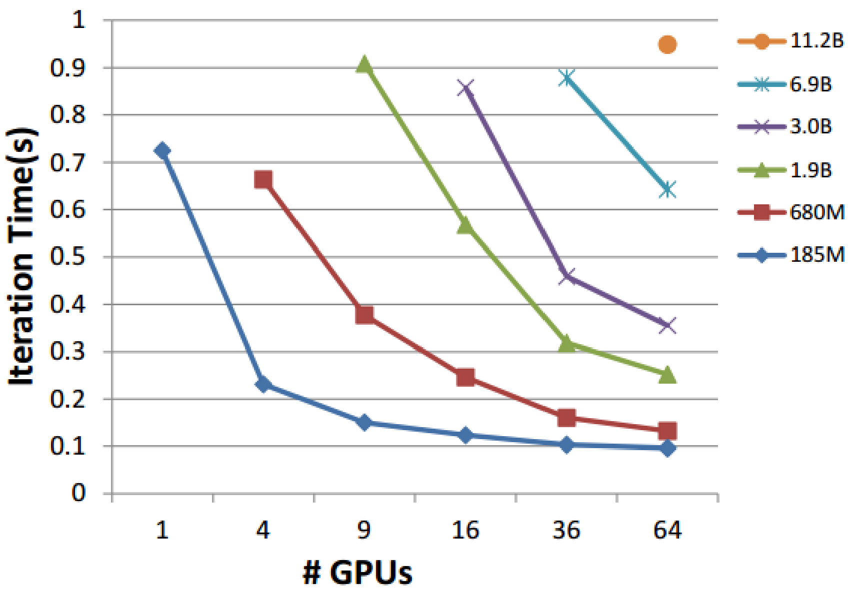


Specialized Hardware

- GPU
  - Thousands of cores, massively parallel (5-14 TFLOP/s per card)
  - Multi-GPU nodes further increase training performance (using data/model parallelism)
  - Drawback: Hard to program efficiently. Solution: Specialized libraries (CUDNN)
- FPGA
  - Specialized for certain operations (e.g. convolutions)
  - Drawbacks: Even harder to program
- Convolutional Processing Units

Deep Learning with GPUs

- A distributed GPU-based system[Coates et al., 2013] was shown to run DistBelief-scale problems (1000 machines) with 3 multi-GPU nodes
- 3 tiers of concurrency: GPU, model parallelism, nodes

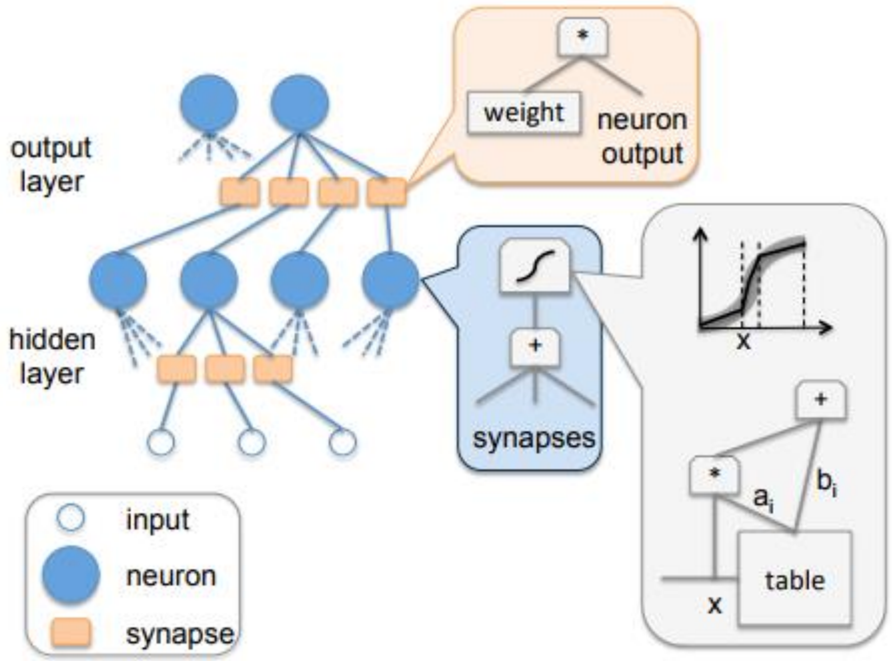


Time for a single mini-batch gradient update of a sparse autoencoder

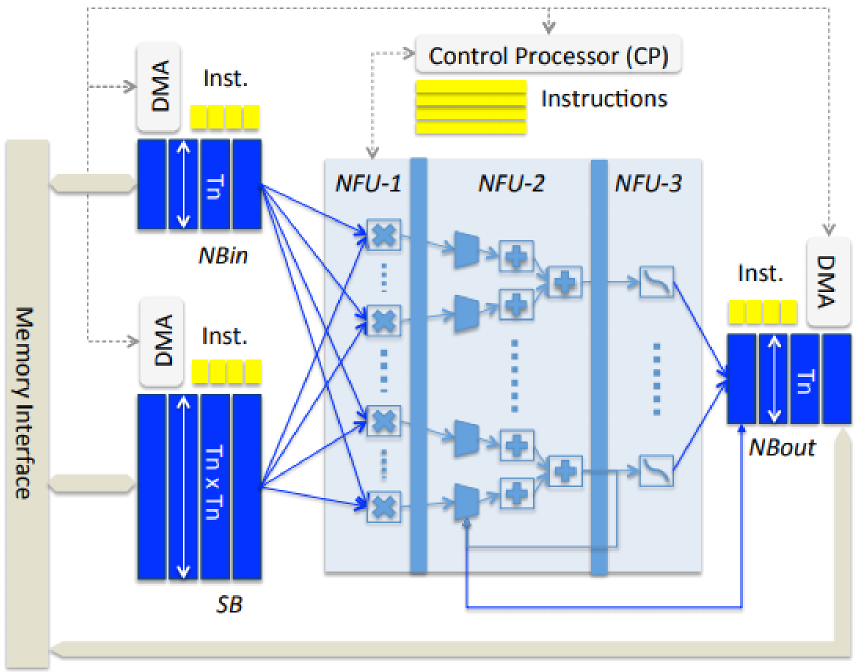
Coates et al. "Deep learning with COTS HPC systems." *ICML* 2013.

Specialized Hardware

- Two approaches:



Mapping neurons to hardware



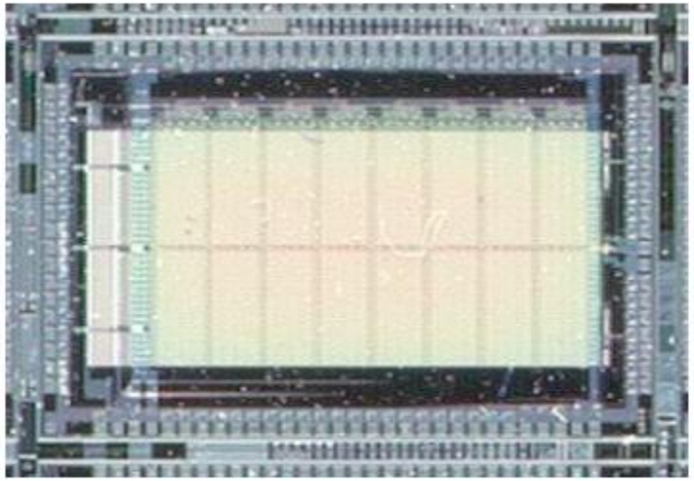
Custom processing elements

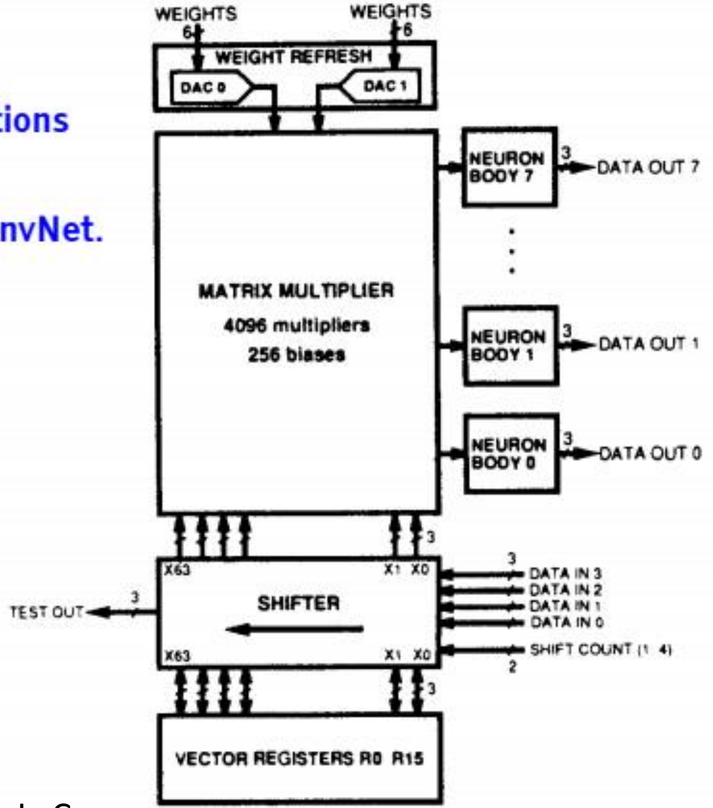
Specialized Hardware – ANNA

ANNA: Analog-Digital ConvNet Accelerator Chip (Bell Labs)

Y LeCun

- [Boser, Säckinger, Bromley, LeCun, Jackel, IEEE J. SSC 26(12), 1991]
- 4096 Multiply-Accumulate operators
- 6 bit weights, 4 bit states
- 20 MHz clock
- Shift registers for efficient I/O with convolutions
- 4 GOPS (peak)
- 1000 characters per second for OCR with ConvNet.

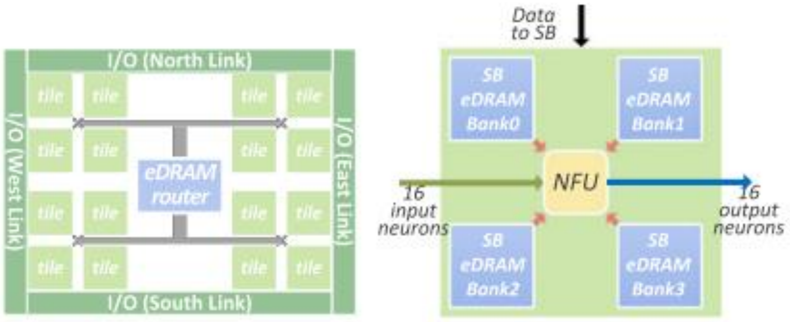




Source: Yann LeCun

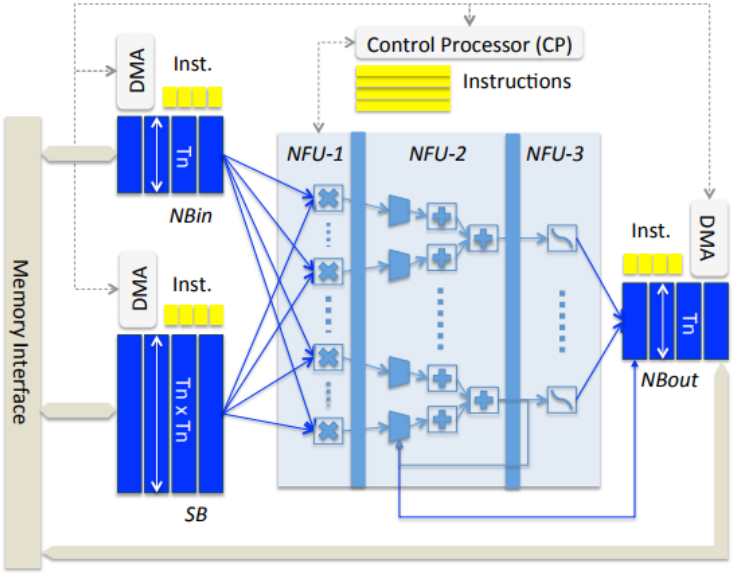
Specialized Hardware – DianNao

- Instead of building the neural network on a circuit, creates a Neural Function Unit (NFU)



- Operations reflect the different layers

- Each operation (conv., pooling, FC) takes up to three stages at computations (or less)
  - Computation
  - Reduction
  - Activation



[Chen et al. “DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine Learning.” ASPLOS 2014]

Conclusions

- Many acceleration opportunities
- Architectures keep changing, and with them new techniques arise
- Algorithms can be modified (to some extent)
  - Proven “shortcuts” can be taken

Questions?