Modern DNNs and TPUs

2024 NSF-sponsored Workshop on Deep Learning Systems in Advanced GPU Cyberinfrastructure

Dr. Iraklis Anagnostopoulos

Southern Illinois University Carbondale

Table of contents

1. Modern Convolutional Neural Networks
2. AlexNet
3. Networks Using Blocks (VGG)
4. Network in Network (NiN)
5. Multi-Branch Networks (GoogLeNet)
6. Residual Networks
7. TPUs

Modern CNNs

Introduction

- The idea of deep neural networks is quite simple ⇒ stack together a bunch of layers
- But, the performance can vary wildly across architectures and hyperparameter choices
- Modern CNNs are the product of
  - intuition
  - a few mathematical insights
  - a lot of trial and error
- Let's name a few:
  - AlexNet
  - VGG
  - NiN
  - Resnet/Resnext
  - etc.

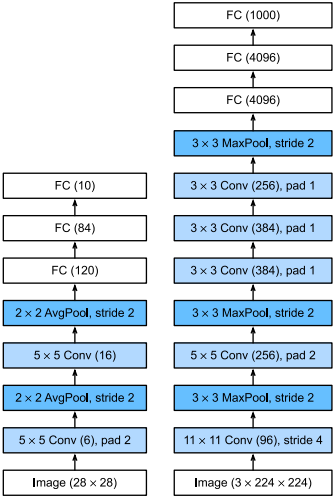
AlexNet

AlexNet

- Although LeNet achieved good results on early small datasets, the performance and feasibility of training CNNs on larger, more realistic datasets had yet to be established
- Until 2012, neural networks were often surpassed by other machine learning methods, such as
  - kernel methods
  - ensemble methods
  - structured estimation
- 2012 Alex Krizhevsky cane up with AlexNet
  - Evolutionary improvement over LeNet
  - It achieved excellent performance in the 2012 ImageNet challenge.
  - **It could run on GPUs**

AlexNet

- AlexNet: 8-layer CNN
- Features obtained by learning can transcend manually-designed features
- AlexNet is much deeper than the comparatively small LeNet5
- AlexNet used the ReLU instead of the sigmoid as its activation function



Networks Using Blocks (VGG)

Networks Using Blocks (VGG)

- While AlexNet was good, it did not provide a general template to guide subsequent researchers in designing new networks
- In 2014, Visual Geometry Group (VGG) at Oxford University presented **VGG**
- Easy to implement repeated structures
- The basic building block of CNNs is a sequence of the following
  - a convolutional layer with padding to maintain the resolution
  - a nonlinearity such as a ReLU
  - a pooling layer such as max-pooling to reduce the resolution

Networks Using Blocks (VGG)

- While AlexNet was good, it did not provide a general template to guide subsequent researchers in designing new networks
- In 2014, Visual Geometry Group (VGG) at Oxford University presented **VGG**
- Easy to implement repeated structures
- The basic building block of CNNs is a sequence of the following
  - a convolutional layer with padding to maintain the resolution
  - a nonlinearity such as a ReLU
  - a pooling layer such as max-pooling to reduce the resolution
- One of the problems with this approach is that the spatial resolution decreases quite rapidly

Networks Using Blocks (VGG)

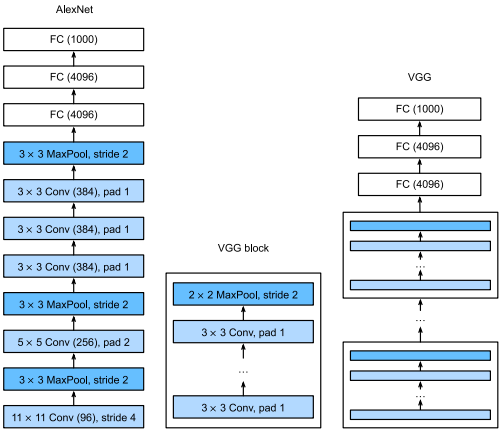
- The key idea of VGG was to use multiple convolutions in between downsampling via max-pooling in the form of a block
- They were primarily interested in whether deep or wide networks perform better
  - two 3×3 convolutions vs single 5×5

Networks Using Blocks (VGG)

- The key idea of VGG was to use multiple convolutions in between downsampling via max-pooling in the form of a block
- They were primarily interested in whether deep or wide networks perform better
  - two 3×3 convolutions vs single 5×5
- A VGG block consists of a sequence of
  - convolutions with 3×3 kernels with padding of 1
  - followed by a 2×2 max-pooling layer with stride of 2

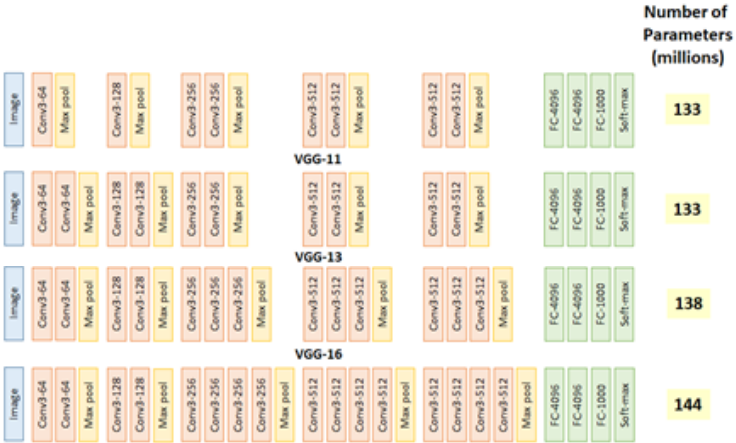
VGG

- From AlexNet to VGG
  - The key difference is that VGG consists of blocks of layers, whereas AlexNet's layers are all designed individually.



VGG variations

- There are multiple VGG variations such as VGG-11, VGG-13, Vgg-16, and VGG-19



Network in Network (NiN)

Network in Network (NiN)

- LeNet, AlexNet, and VGG all share a common design pattern:
  - Extract features exploiting spatial structure via a sequence of convolutions and pooling layers and
  - post-process the representations via fully connected layers

Network in Network (NiN)

- LeNet, AlexNet, and VGG all share a common design pattern:
  - Extract features exploiting spatial structure via a sequence of convolutions and pooling layers and
  - post-process the representations via fully connected layers
- However, this poses two major challenges

Network in Network (NiN)

- LeNet, AlexNet, and VGG all share a common design pattern:
  - Extract features exploiting spatial structure via a sequence of convolutions and pooling layers and
  - post-process the representations via fully connected layers
- However, this poses two major challenges
  - The fully connected layers at the end of the architecture consume tremendous numbers of parameters
    - VGG-11 requires 400MB of RAM in single precision (FP32) !

Network in Network (NiN)

- LeNet, AlexNet, and VGG all share a common design pattern:
  - Extract features exploiting spatial structure via a sequence of convolutions and pooling layers and
  - post-process the representations via fully connected layers
- However, this poses two major challenges
  - The fully connected layers at the end of the architecture consume tremendous numbers of parameters
    - VGG-11 requires 400MB of RAM in single precision (FP32) !
  - It is impossible to add fully connected layers earlier in the network to increase the degree of nonlinearity

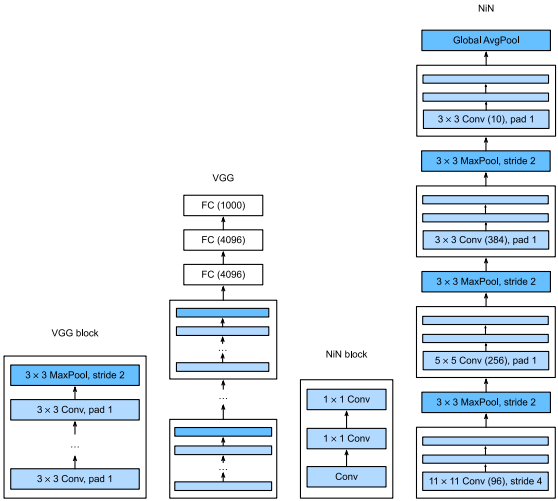
Network in Network (NiN)

- The network in network (NiN) blocks (1993) offer an alternative
- They are based on a very simple insight
  - use 1×1 convolutions to add local nonlinearities across the channel activations and
  - use global average pooling to integrate across all locations in the last representation layer

Network in Network (NiN)

- The network in network (NiN) blocks (1993) offer an alternative
- They are based on a very simple insight
  - use 1×1 convolutions to add local nonlinearities across the channel activations and
  - use global average pooling to integrate across all locations in the last representation layer
- The idea behind NiN is to apply a fully connected layer at each pixel location
- The resulting 1×1 convolution can be thought as a fully connected layer acting independently on each pixel location

Network in Network (NiN)



GoogLeNet

GoogLeNet

- In 2014, GoogLeNet won the ImageNet Challenge using a structure that combined
  - the strengths of NiN
  - repeated blocks
  - a cocktail of convolution kernels

GoogLeNet

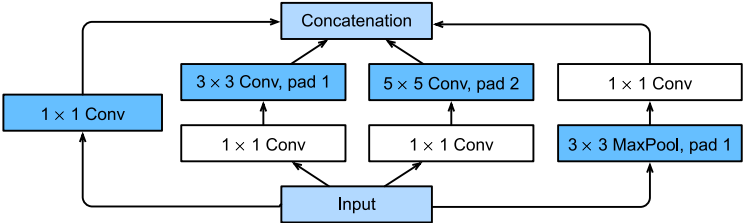
- In 2014, GoogLeNet won the ImageNet Challenge using a structure that combined
  - the strengths of NiN
  - repeated blocks
  - a cocktail of convolution kernels
- It exhibits a clear distinction among the stem (data ingest), body (data processing), and head (prediction) in a CNN
  - stem $\Rightarrow$ extract low-level features
  - head $\Rightarrow$ maps the features obtained to the required classification/segmentation/detection/tracking problem

GoogLeNet

- In 2014, GoogLeNet won the ImageNet Challenge using a structure that combined
  - the strengths of NiN
  - repeated blocks
  - a cocktail of convolution kernels
- It exhibits a clear distinction among the stem (data ingest), body (data processing), and head (prediction) in a CNN
  - stem $\Rightarrow$ extract low-level features
  - head $\Rightarrow$ maps the features obtained to the required classification/segmentation/detection/tracking problem
- Innovation of GoogLenet: Instead of identifying which kernel to use (1×1 to 11×11)
  - concatenate multi-branch convolutions

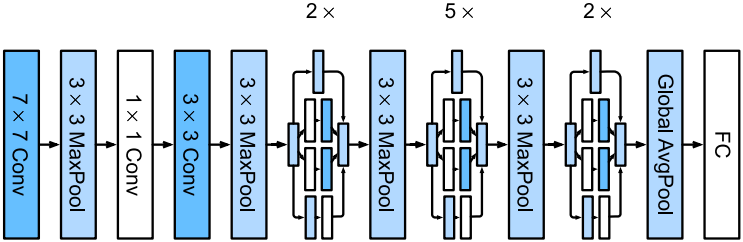
GoogLeNet

- The basic convolutional block in GoogLeNet is called **Inception block**
- The inception block consists of four parallel branches
  - The first three branches use convolutional layers with sizes of 1×1, 3×3, and 5×5
  - The middle two branches add a convolution of 1×1 to reduce the model's complexity
  - The fourth branch uses a 3×3 max-pooling layer and a 1×1 convolutional layer
  - Finally, the outputs along each branch are concatenated along the channel dimension and comprise the block's output



GoogLeNet

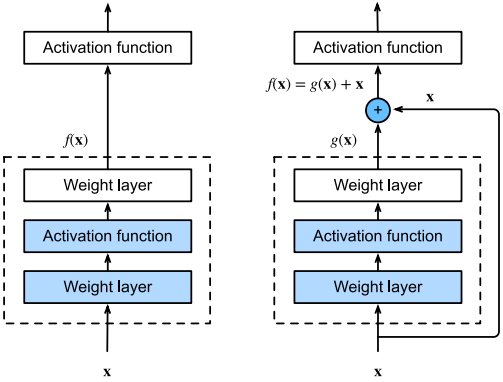
- GoogLeNet uses a stack of a total of 9 inception blocks, arranged into 3 groups
- Max-pooling in between
- Global average pooling in its head
- At its stem, the first module is similar to AlexNet and LeNet



Residual Networks

Residual Networks

- On the left, the portion within the dotted-line box must directly learn the mapping $f(\mathbf{x})$
- On the right, the portion within the dotted-line box needs to learn the residual mapping $g(\mathbf{x}) = f(\mathbf{x}) - \mathbf{x}$
- With residual blocks, inputs can forward propagate faster through the residual connections across layers.
- The residual block can be thought of as a special case of the Inception block

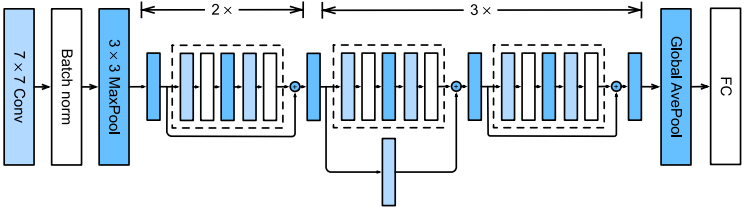


Residual Networks

- ResNet follows VGG's full 3×3 convolutional layer design
- The residual block has two 3×3 convolutional layers
- Each convolutional layer is followed by a ReLU activation function

ResNet

- The first two layers of ResNet are the same as those of the GoogLeNet (7×7 convolutional layer)
- Followed by a 3×3 max-pooling layer
- Below you can see the ResNet-18
- By configuring different numbers of channels and residual blocks in the module, we can create different ResNet models, such as the deeper 152-layer ResNet-152



TPUs

Systolic Arrays: Motivation

- **Goal**: design an accelerator that has
  - **Simple, regular design** (keep # unique parts small and regular)
  - **High concurrency** ⇒ high performance
  - **Balanced computation and I/O (memory) bandwidth**

Systolic Arrays: Motivation

- **Goal**: design an accelerator that has
  - **Simple, regular design** (keep # unique parts small and regular)
  - **High concurrency** ⇒ high performance
  - **Balanced computation and I/O (memory) bandwidth**
- Idea: Replace a single processing element (PE) with a **regular array of PEs** and carefully orchestrate flow of data between the PEs
  - such that they collectively transform a piece of input data before outputting it to memory

Systolic Arrays: Motivation

- **Goal**: design an accelerator that has
  - **Simple, regular design** (keep # unique parts small and regular)
  - **High concurrency** ⇒ high performance
  - **Balanced computation and I/O (memory) bandwidth**
- Idea: Replace a single processing element (PE) with a **regular array of PEs** and carefully orchestrate flow of data between the PEs
  - such that they collectively transform a piece of input data before outputting it to memory
- Benefit: **Maximizes computation done on a single piece of data element brought from memory**

Systolic Arrays

- Memory: heart
- Data: blood
- PEs: cells

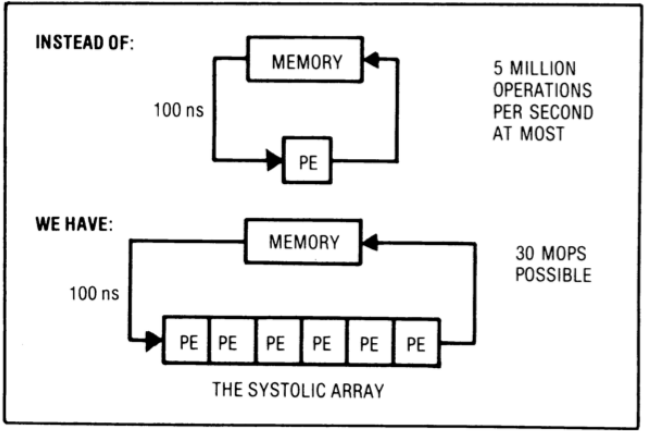


Figure 1. Basic principle of a systolic system.

Why Systolic Architectures ?

- Idea: Data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory

Why Systolic Architectures ?

- Idea: Data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory

- Similar to **blood flow: heart ⇒ many cells ⇒ heart**
  - Different cells "process" the blood
  - Many veins operate simultaneously
  - Can be many-dimensional

Why Systolic Architectures ?

- Idea: Data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory

- Similar to **blood flow: heart ⇒ many cells ⇒ heart**
  - Different cells "process" the blood
  - Many veins operate simultaneously
  - Can be many-dimensional

- Why? Special purpose accelerators/architectures need
  - **Simple, regular design** (keep # unique parts small and regular)
  - **High concurrency** ⇒ high performance
  - **Balanced computation and I/O (memory) bandwidth**

Systolic Architectures

- Basic principle: Replace a single PE with a **regular array of PEs** and **carefully orchestrate flow of data** between the PEs
  - Balance computation and memory bandwidth

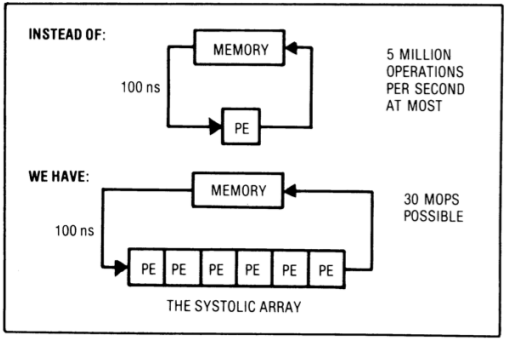
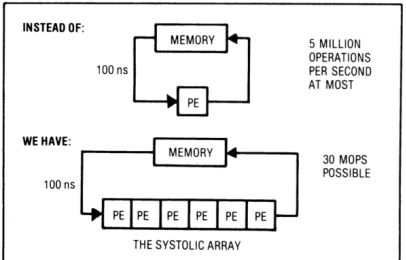


Figure 1. Basic principle of a systolic system.

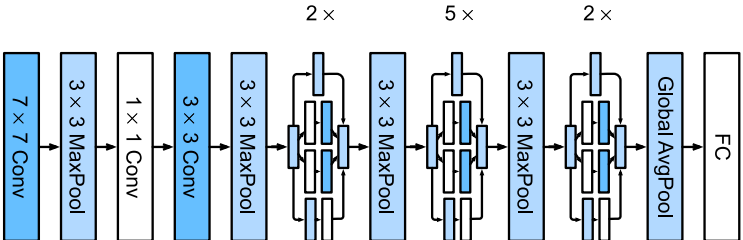
Systolic Architectures

- Differences from pipelining:
  - These are individual PEs
  - Array structure can be non-linear and multi-dimensional
  - PE connections can be multidirectional (and different speed)
  - PEs can have local memory and execute kernels (rather than a piece of the instruction)



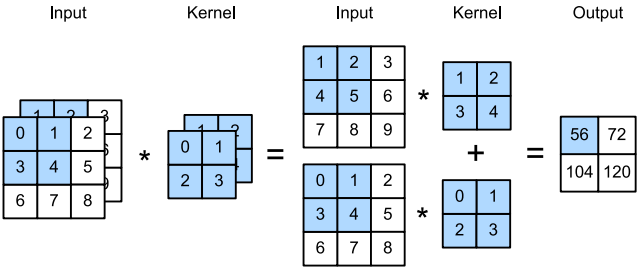
Systolic Computation Example

- Convolution:
  - Used in filtering, pattern matching, correlation, polynomial evaluation, etc . . .
  - Many image processing tasks
  - Machine learning: up to hundreds of convolutional layers in Convolutional Neural Networks (CNN)



Convolution example

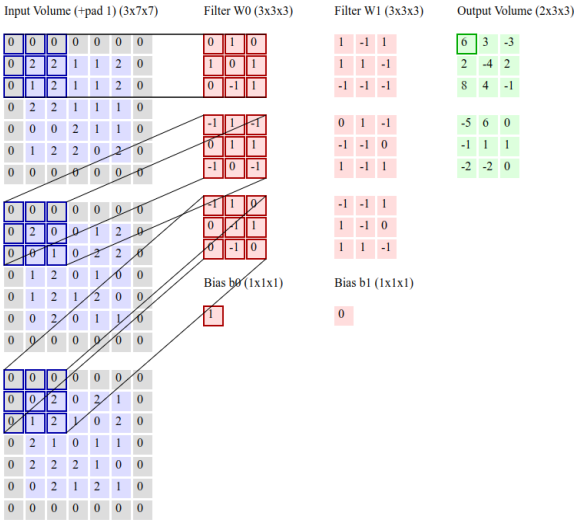
- Example of a two-dimensional cross-correlation with two input channels
- The shaded portions are the first output element as well as the input and kernel tensor elements used for the output computation



$$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$$

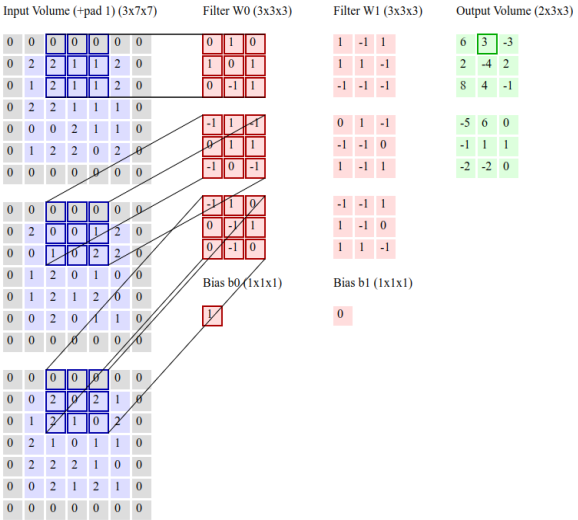
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



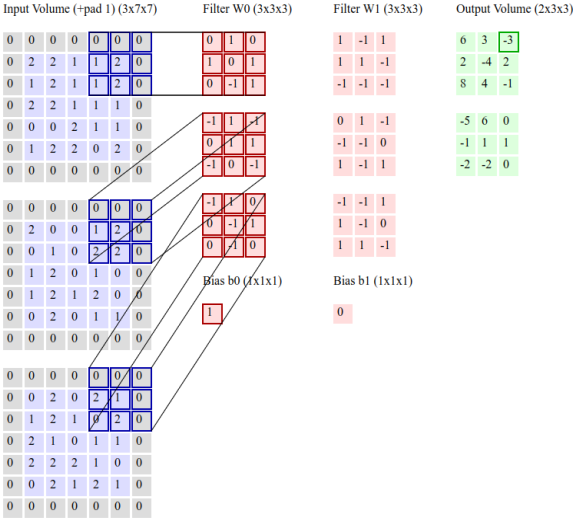
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



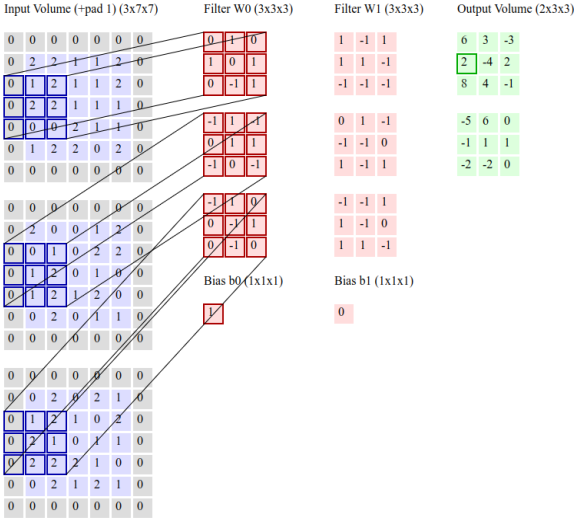
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



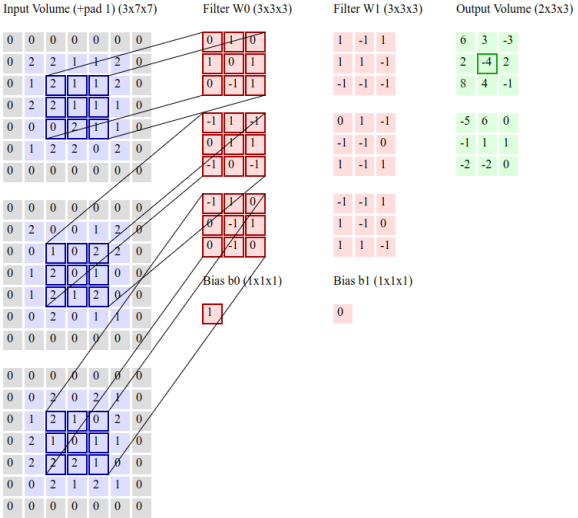
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



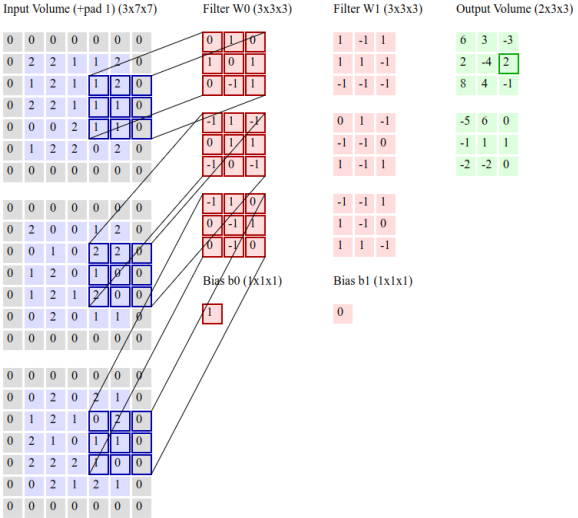
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



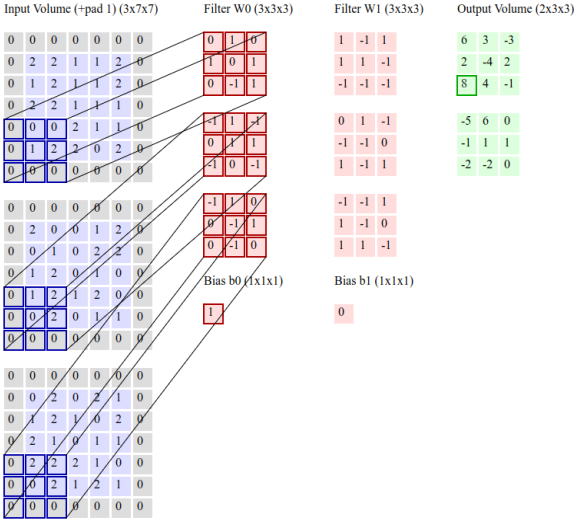
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



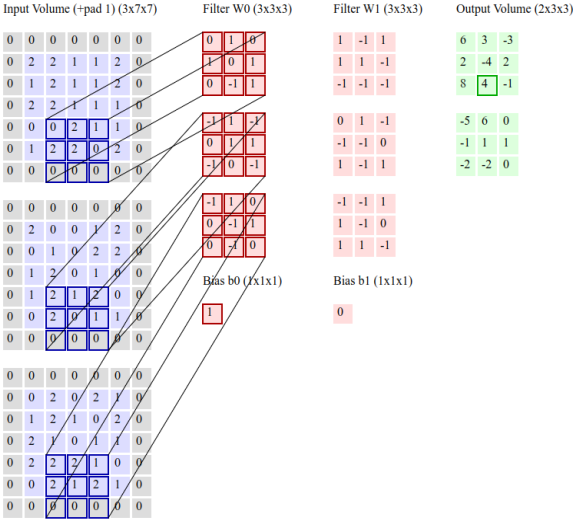
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



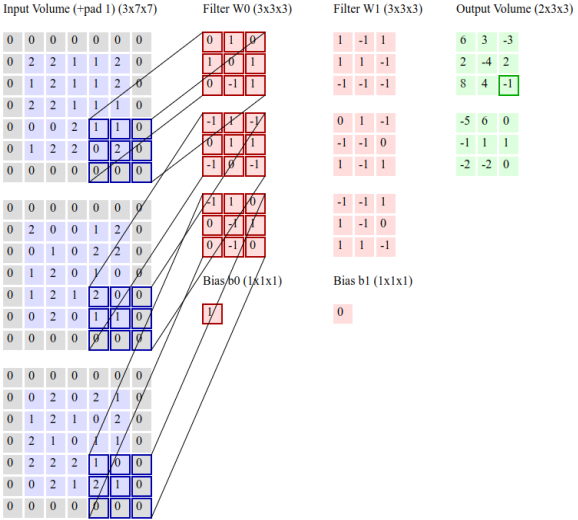
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



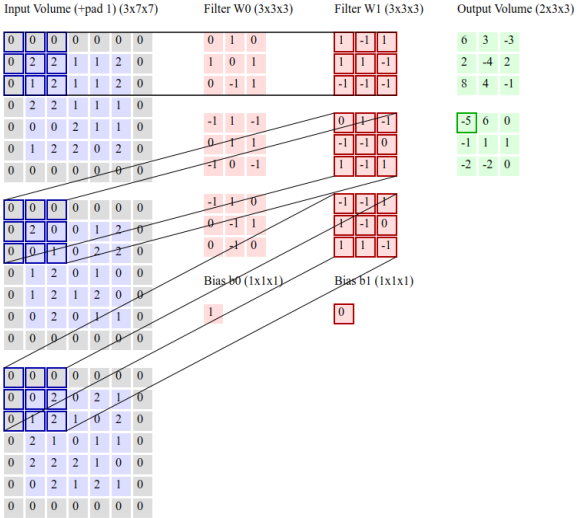
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



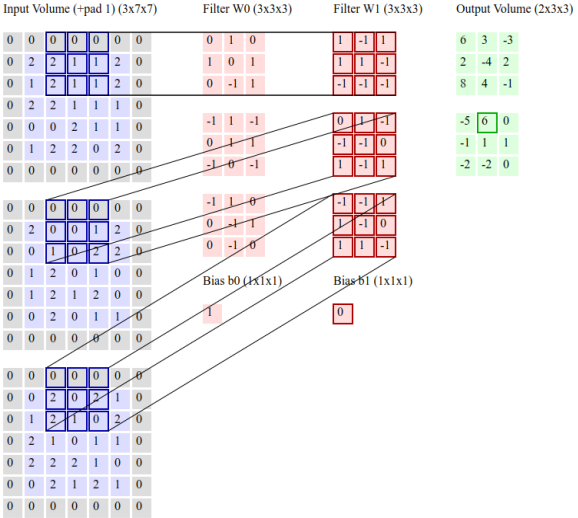
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



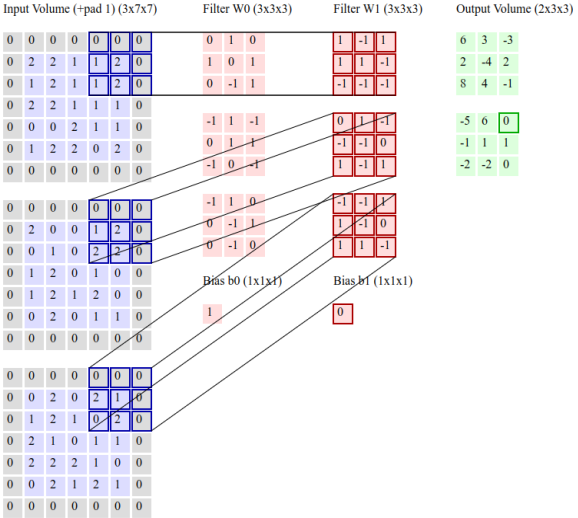
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



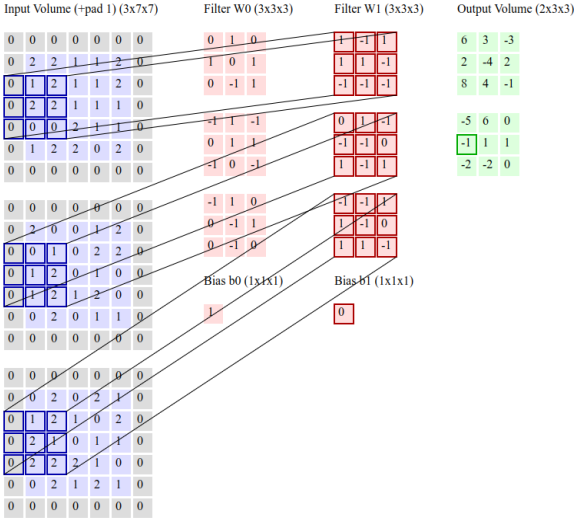
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



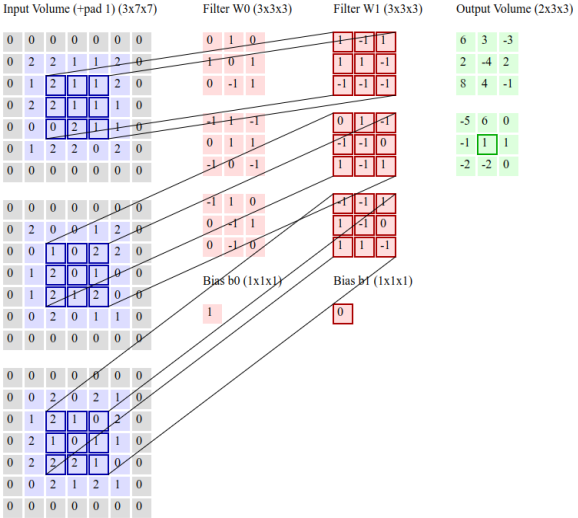
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



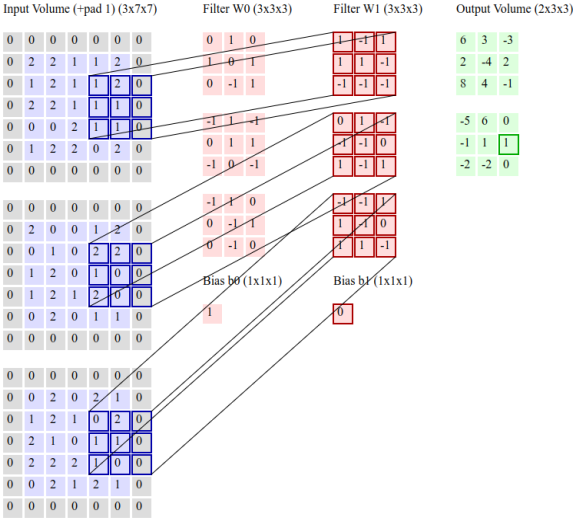
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



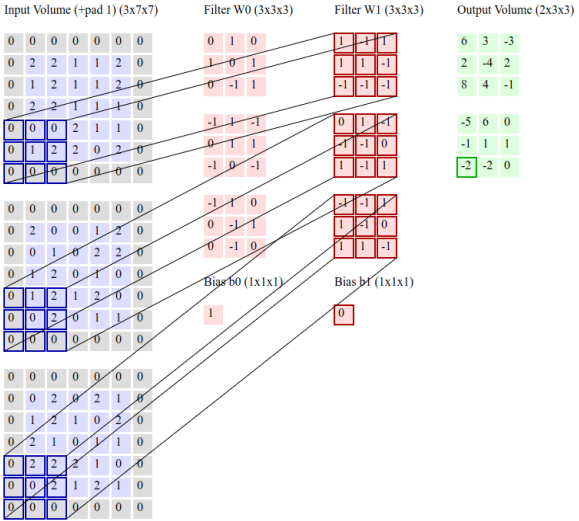
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



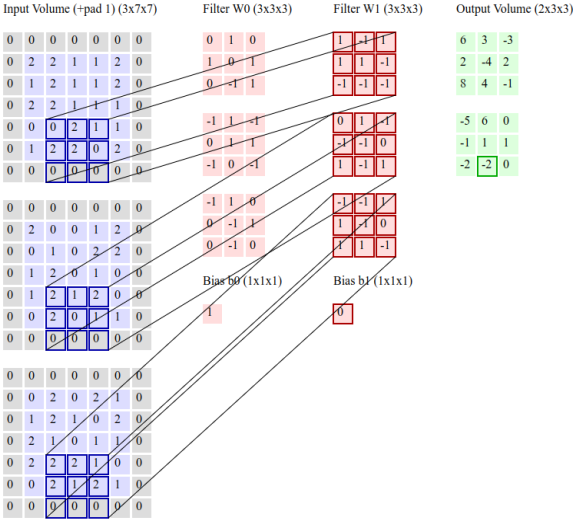
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



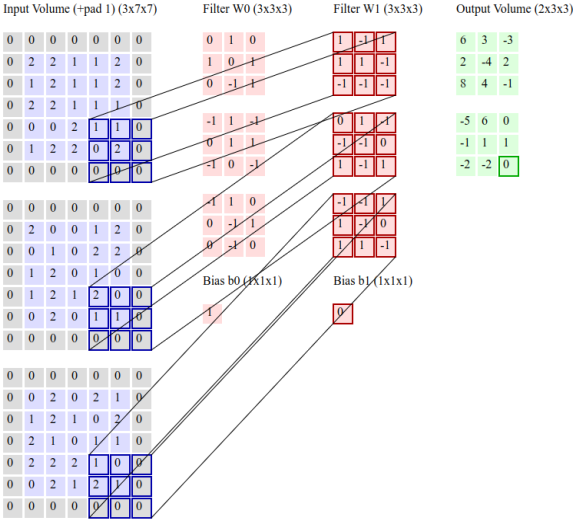
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2

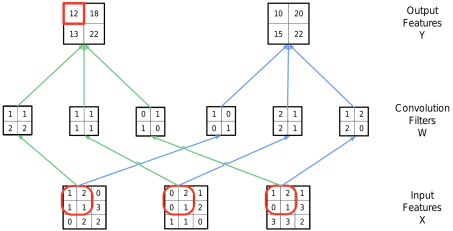


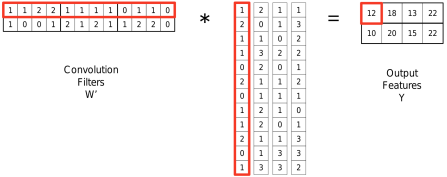
Convolution demo

- Input volume:
  - 3 channels, 5×5 image
- Padding 1
  - $\Rightarrow$ makes the image 7×7
- 2 filters
  - 3×3 and 3 channels each
- Stride 2



Convolutional Layer with Matrix Multiplication





Matrix-Matrix Multiplication

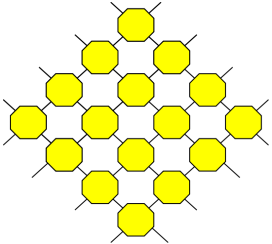
$$\begin{bmatrix} 2 & 7 & 3 \\ 1 & 5 & 8 \\ 0 & 4 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 0 & 1 \\ 2 & 1 & 0 \\ 1 & 2 & 4 \end{bmatrix} = \begin{bmatrix} 23 & 13 & 14 \\ 21 & 21 & 33 \\ 9 & 6 & 4 \end{bmatrix}$$

Example Systolic Algorithm: Matrix Multiplication

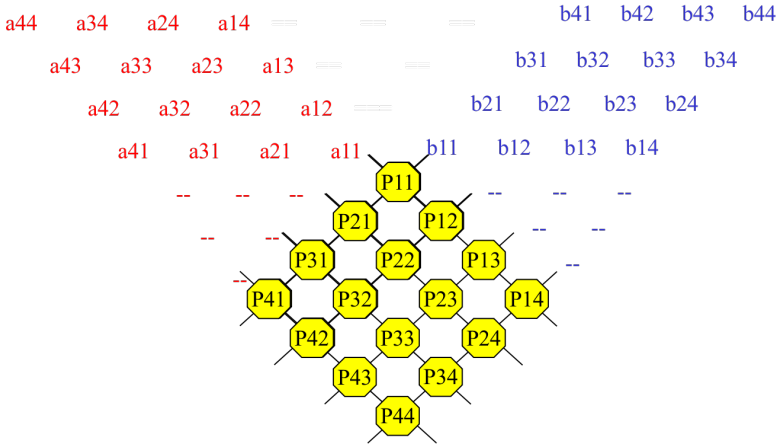
- Problem: multiply two $n \times n$ matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \end{bmatrix}$$

- Systolic solution uses 2D array with $N \times N$ cells, 2 input streams and 2 output streams

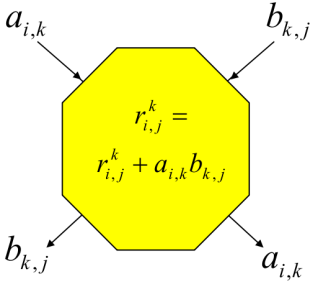


Systolic Matrix Multiplication



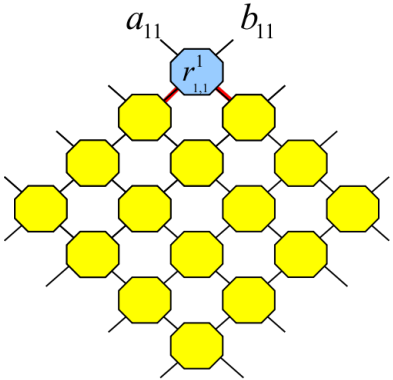
Operation at each cell

- Each $r_{i,j}^k$ cell updates at each time step as shown below
- $r_{i,j}^0$ initialized to 0

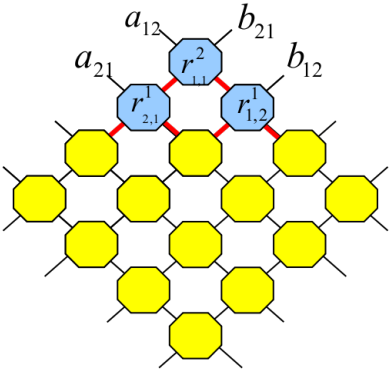


Data Flow for Systolic MM

- Beat 1

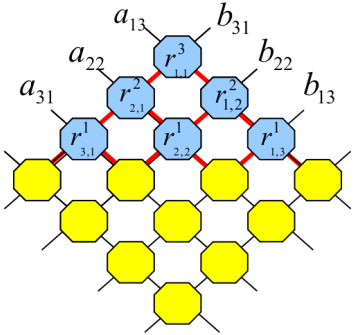


- Beat 2

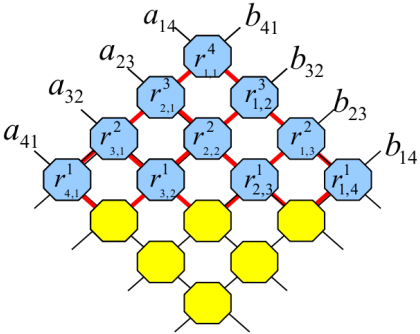


Data Flow for Systolic MM

- Beat 3

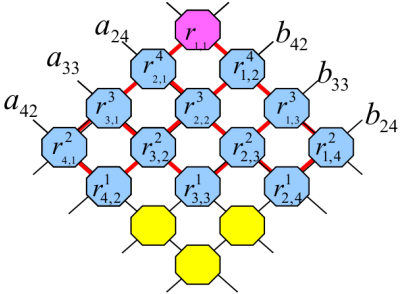


- Beat 4

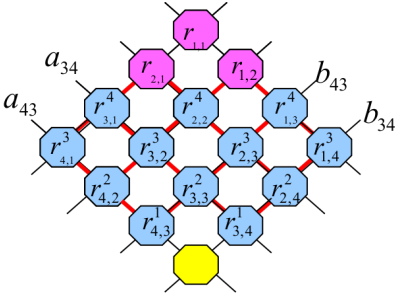


Data Flow for Systolic MM

- Beat 5

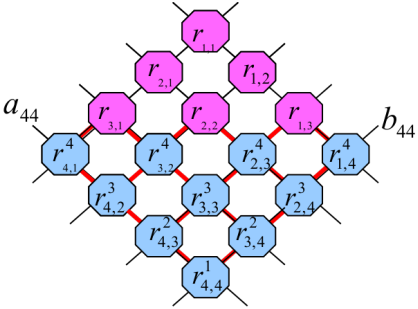


- Beat 6

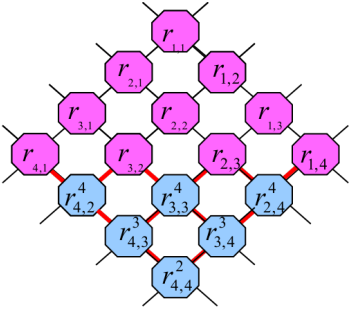


Data Flow for Systolic MM

- Beat 7

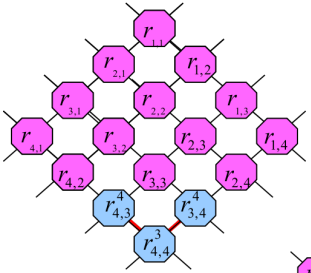


- Beat 8

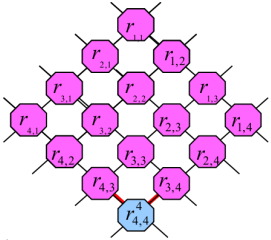


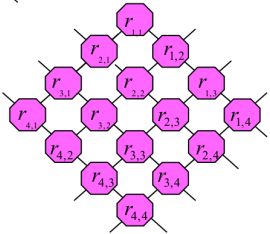
Data Flow for Systolic MM

- Beat 9



- Beat2 10 and 11





Systolic Arrays: Pros and Cons

- Advantages:
  - Principled: Efficiently makes use of limited memory bandwidth, balances computation to I/O bandwidth availability
  - Specialized (computation needs to fit PE organization/functions)
    - improved efficiency, simple design, high concurrency/ performance
    - good to do more with less memory bandwidth requirement

Systolic Arrays: Pros and Cons

- Advantages:
  - Principled: Efficiently makes use of limited memory bandwidth, balances computation to I/O bandwidth availability
  - Specialized (computation needs to fit PE organization/functions)
    - improved efficiency, simple design, high concurrency/ performance
    - good to do more with less memory bandwidth requirement
- Downside:
  - Specialized $\Rightarrow$ not generally applicable because computation needs to fit the PE functions/organization
  - Not good at exploiting irregular parallelism

An Example Modern Systolic Array: TPU

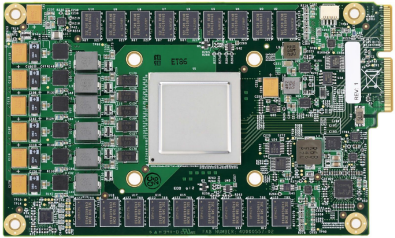


Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

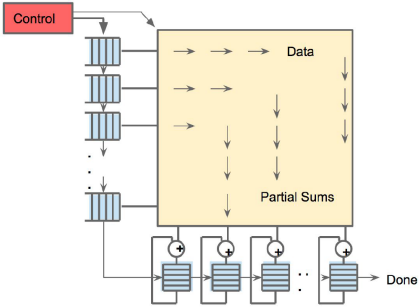


Figure 4. Systolic data flow of the Matrix Multiply Unit. Software has the illusion that each 256B input is read at once, and they instantly update one location of each of 256 accumulator RAMs.

Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA 2017.

An Example Modern Systolic Array: TPU

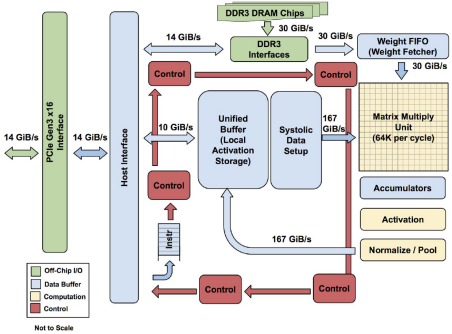
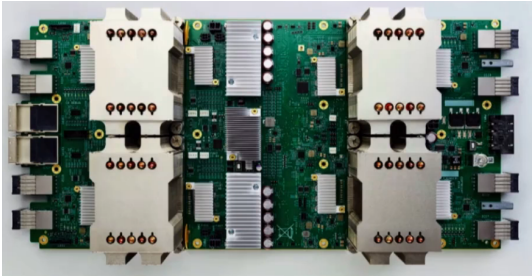


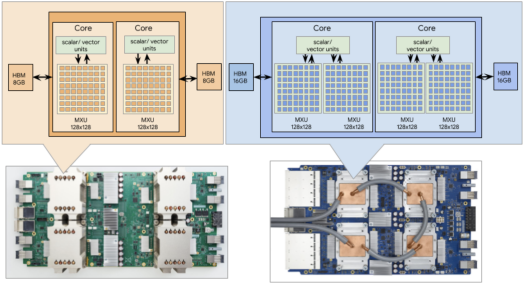
Figure 1. TPU Block Diagram. The main computation part is the yellow Matrix Multiply unit in the upper right hand corner. Its inputs are the blue Weight FIFO and the blue Unified Buffer (UB) and its output is the blue Accumulators (Acc). The yellow Activation Unit performs the nonlinear functions on the Acc, which go to the UB.

An Example Modern Systolic Array: TPU2



- **4 TPU chips** vs 1 chip in TPU1
- **High Bandwidth Memory** vs DDR3
- **Floating point operations** vs FP16
- **45 TFLOPS per chip** vs 23 TOPS
- Designed for **training and inference** vs only inference

An Example Modern Systolic Array: TPU3

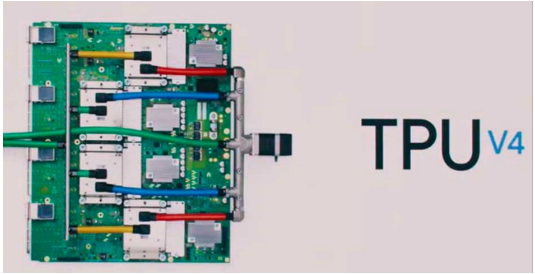


TPU v2 - 4 chips, 2 cores per chip

TPU v3 - 4 chips, 2 cores per chip

- **32GB HBM per chip** vs 16GB HBM in TPU2
- **4 Matrix Units per chip** vs 2 Matrix Units in TPU2
- **90 TFLOPS per chip** vs vs 45 TFLOPS in TPU2

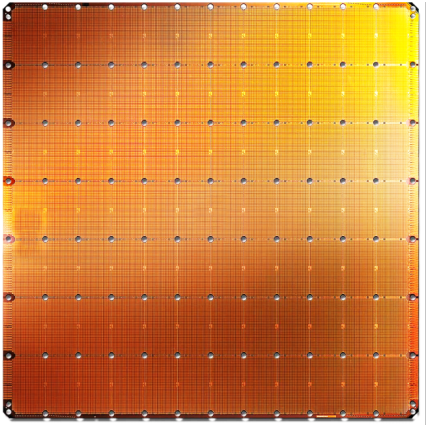
An Example Modern Systolic Array: TPU4



- **250 TFLOPS per chip in 2021** vs 90 TFLOPS in TPU3
- **1 ExaFLOPS per board**
- **New ML applications (vs. TPU3):**
  - Computer vision
  - Natural Language Processing (NLP)
  - Recommender system
  - Reinforcement learning that plays Go
  - 250 TFLOPS per chip in 2021

https://spectrum.ieee.org/tech-talk/computing/hardware/heres-how-googles-tpu-v4-ai-chip-stacked-up-in-training-tests

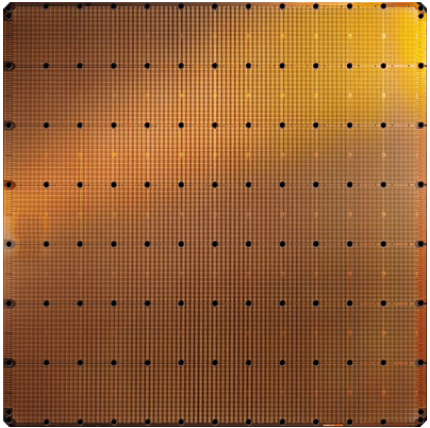
Cerebras's Wafer Scale Engine (2019)



- Cerebras WSE 1.2 Trillion transistors $46,225\ mm^2$
  - 400,000 cores
- NVIDIA TITAN V
  - 21.1 Billion transistors 815 mm^2

https://www.anandtech.com/show/14758/hot-chips-31-live-blogs-cerebras-wafer-scale-deep-learning

Cerebras's Wafer Scale Engine (2021)



- Cerebras WSE-2 2.6 Trillion transistors $46,225\ mm^2$
  - 850,000 cores
- NVIDIA Ampere GA100
  - 54.2 Billion transistors 826 mm^2

https://www.cerebras.net/cerebras-wafer-scale-engine-why-we-need-big-chips-for-deep-learning/

Questions?