

```
>>> c++ deep-learning.cpp
```

```
>>> ./a.out data.csv
```

```
>>> 42
```

Strategy



OSI Model of Machine Learning

Task Layer

Task Layer

$$f \left(\text{Teapot} \right) = \text{Teapot}$$

Classification*

Task Layer

$$f \left(\text{Teapot} \right) = \text{Teapot}$$

Classification*

$$f(\text{文内容}) = \begin{matrix} \circ & \circ & \circ \\ \circ & \circ & \circ \\ \circ & \circ & \circ \end{matrix}$$

Translation*

Task Layer

$$f \left(\text{Teapot} \right) = \text{Teapot}$$

Classification*

$$f(\text{文内容}) = \begin{matrix} \circ \\ \vdots \\ \circ \\ \circ \\ \vdots \\ \circ \end{matrix}$$

Translation*

$$f \left(\text{Clouds} \right) =$$



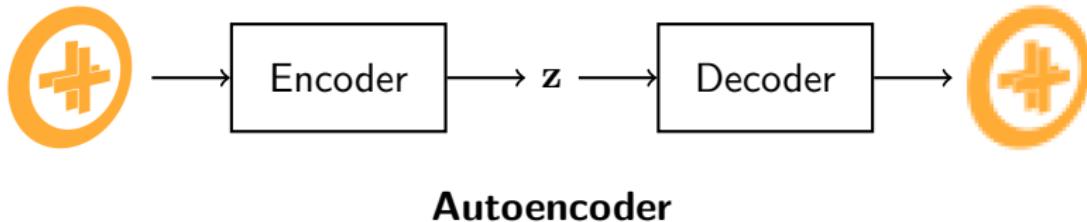
Generative[†]

* Supervised

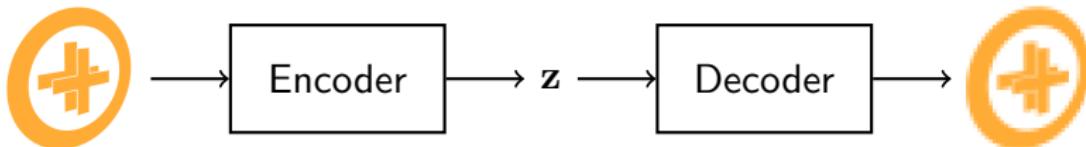
† Unsupervised

Model Layer

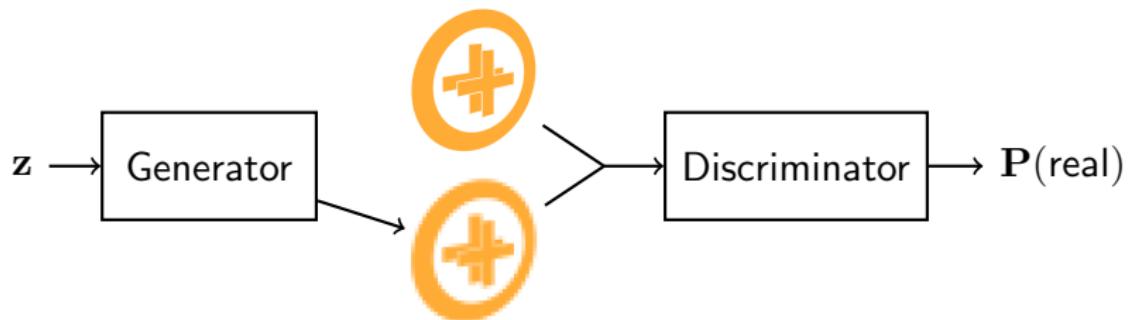
Model Layer



Model Layer

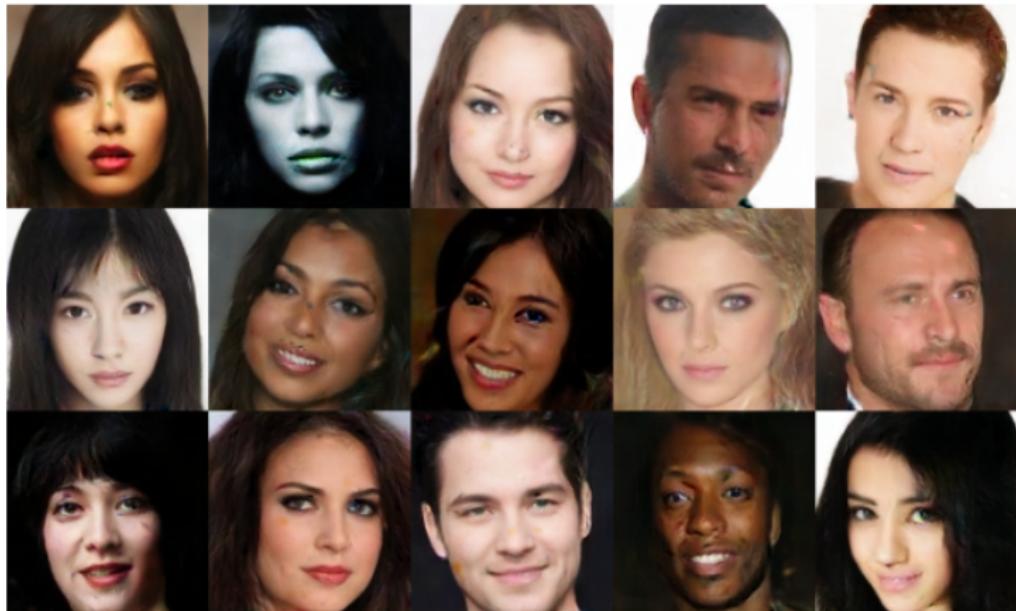


Autoencoder



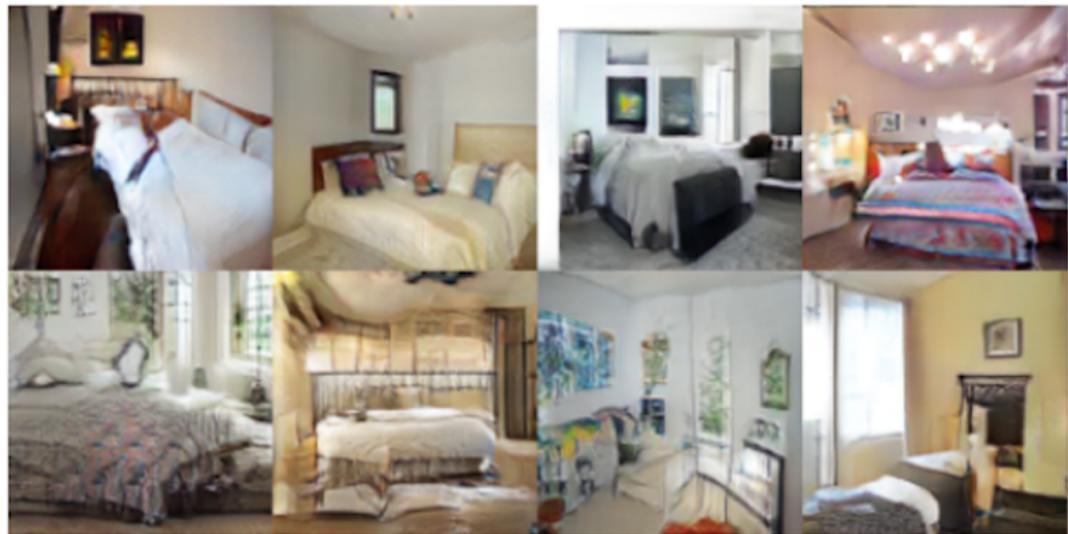
Generative Adversarial Networks (GAN)

Generative Adversarial Networks



BEGAN: Berthelot et al. (2017)

Generative Adversarial Networks



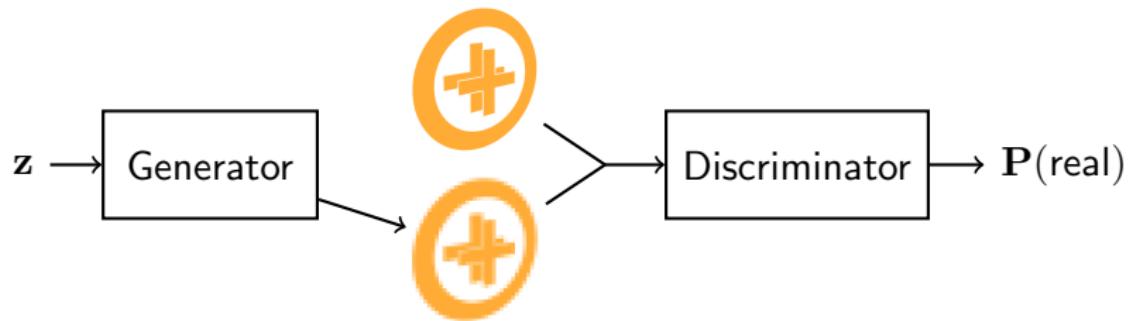
LSGAN: Mao et al. (2016)

Generative Adversarial Networks



BEGAN: Berthelot et al. (2017)

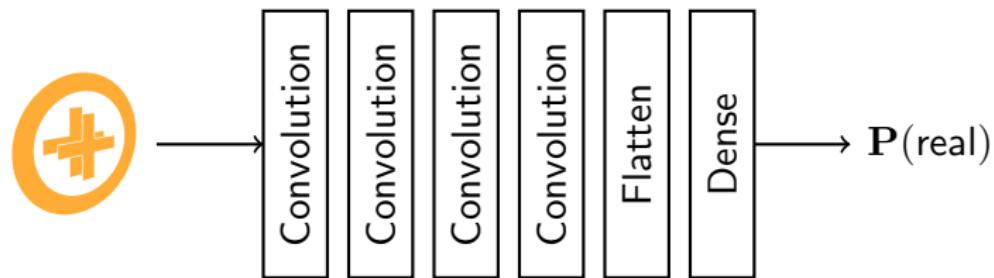
Layer Layer



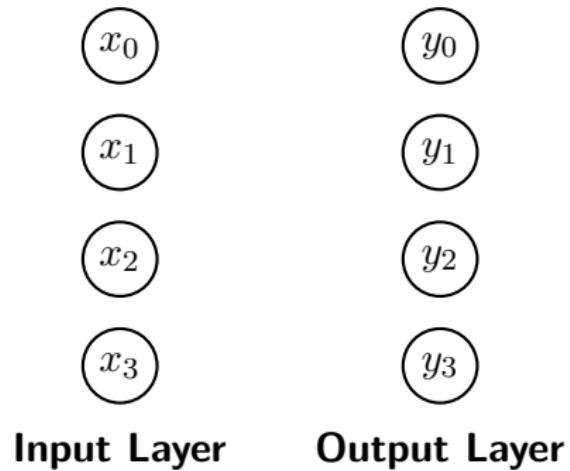
Layer Layer



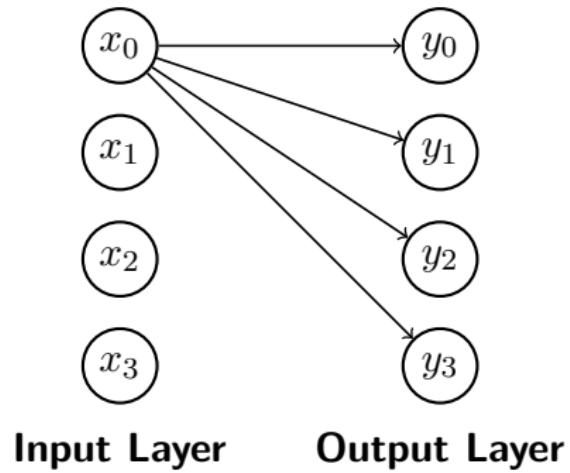
Layer Layer



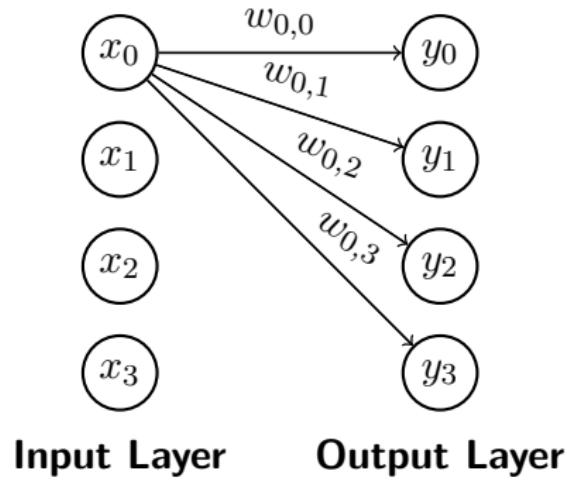
Layers: Dense



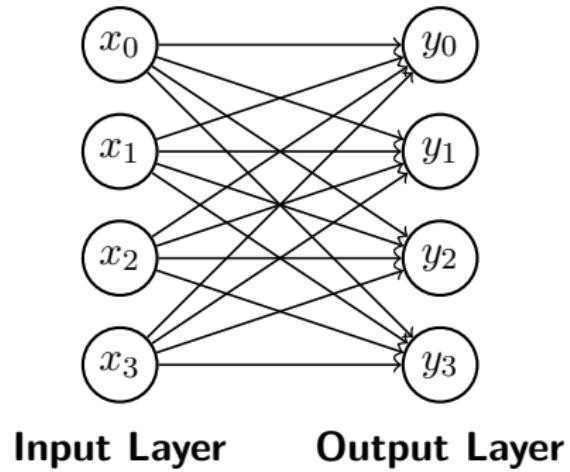
Layers: Dense



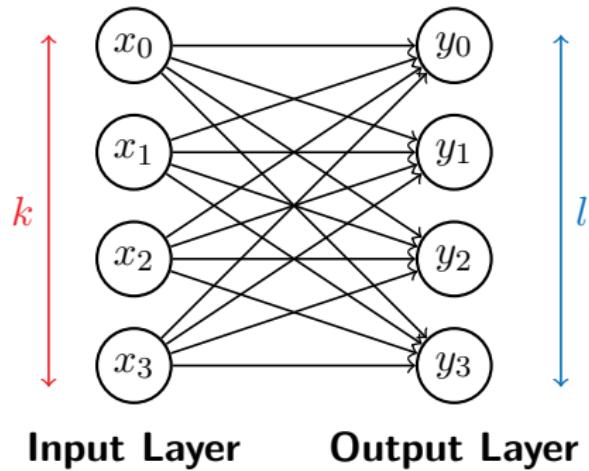
Layers: Dense



Layers: Dense



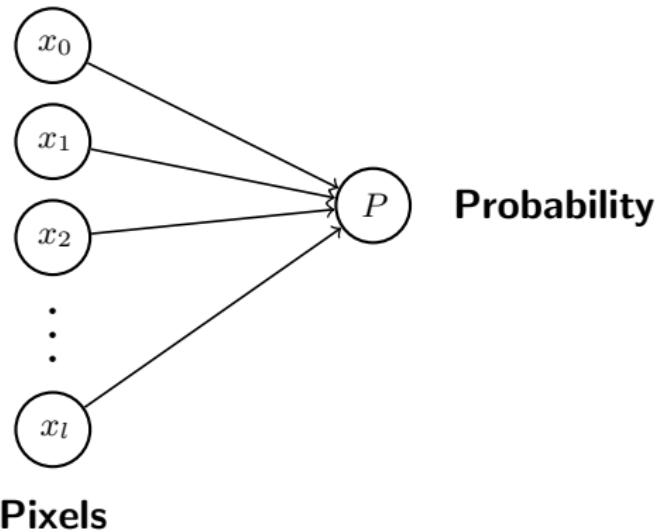
Layers: Dense



$$n \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & x_{n,3} \end{bmatrix} \times \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,0} & w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} & y_{0,3} \\ y_{1,0} & y_{1,1} & y_{1,2} & y_{1,3} \\ \vdots & \vdots & \vdots & \vdots \\ y_{n,0} & y_{n,1} & y_{n,2} & y_{n,3} \end{bmatrix}$$

The diagram shows a matrix multiplication operation. The input layer is represented as a column vector of size n (number of input nodes). The output layer is represented as a row vector of size l (number of output nodes). The weight matrix has dimensions $n \times l$. The result is a column vector of size l .

Layers: Dense



Layers: Convolution



Image

Layers: Convolution

0.4	0.9	0.1
0.7	0.2	0.6
0.8	0.3	0.5

Image

Layers: Convolution

0.4	0.9	0.1
0.7	0.2	0.6
0.8	0.3	0.5

Image

5.7	2.4
3.1	0.9

Kernel

Layers: Convolution

$5.7 \cdot 0.4$	$2.4 \cdot 0.9$	0.1
$3.1 \cdot 0.7$	$0.9 \cdot 0.2$	0.6
0.8	0.3	0.5

Image

Layers: Convolution

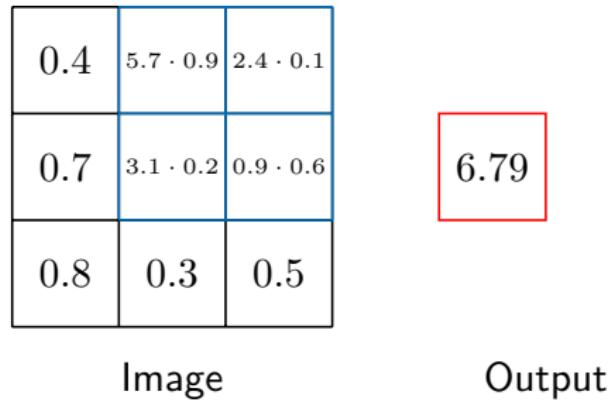
$5.7 \cdot 0.4$	$2.4 \cdot 0.9$	0.1
$3.1 \cdot 0.7$	$0.9 \cdot 0.2$	0.6
0.8	0.3	0.5

Image

6.79

Output

Layers: Convolution



Layers: Convolution

0.4	$5.7 \cdot 0.9$	$2.4 \cdot 0.1$
0.7	$3.1 \cdot 0.2$	$0.9 \cdot 0.6$
0.8	0.3	0.5

Image

6.79	6.53
------	------

Output

Layers: Convolution

0.4	0.9	0.1
$5.7 \cdot 0.7$	$2.4 \cdot 0.2$	0.6
$3.1 \cdot 0.8$	$0.9 \cdot 0.3$	0.5

Image

6.79	6.53
------	------

Output

Layers: Convolution

0.4	0.9	0.1
$5.7 \cdot 0.7$	$2.4 \cdot 0.2$	0.6
$3.1 \cdot 0.8$	$0.9 \cdot 0.3$	0.5

Image

6.79	6.53
7.67	

Output

Layers: Convolution

0.4	0.9	0.1
0.7	$5.7 \cdot 0.2$	$2.4 \cdot 0.6$
0.8	$3.1 \cdot 0.3$	$0.9 \cdot 0.5$

Image

6.79	6.53
7.67	

Output

Layers: Convolution

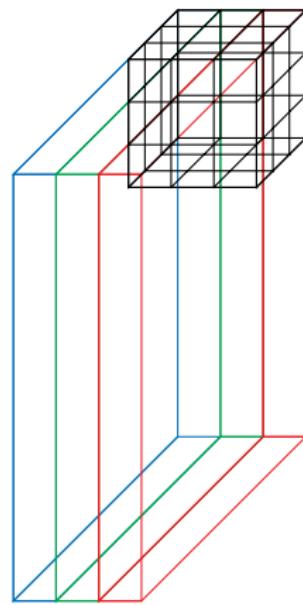
0.4	0.9	0.1
0.7	$5.7 \cdot 0.2$	$2.4 \cdot 0.6$
0.8	$3.1 \cdot 0.3$	$0.9 \cdot 0.5$

Image

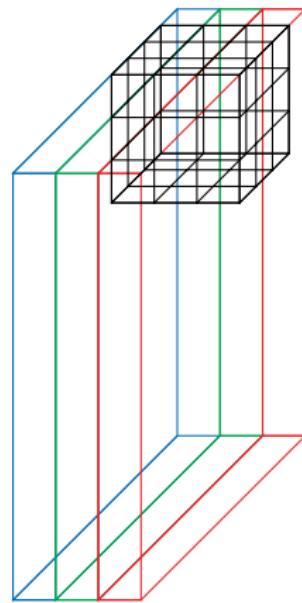
6.79	6.53
7.67	3.96

Output

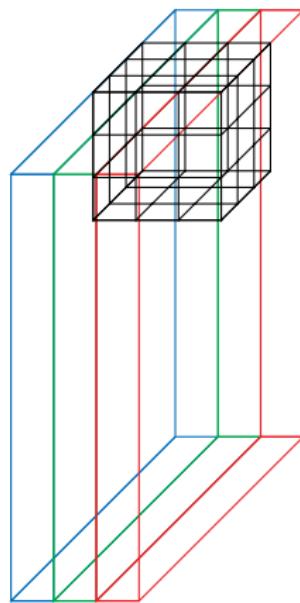
Layers: Convolution



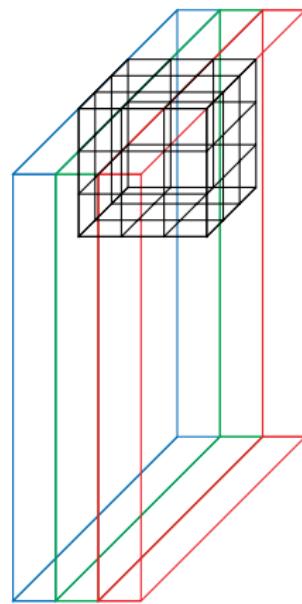
Layers: Convolution



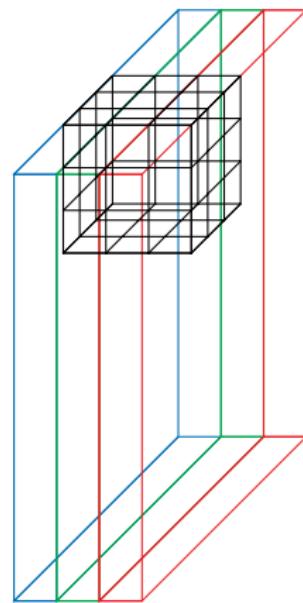
Layers: Convolution



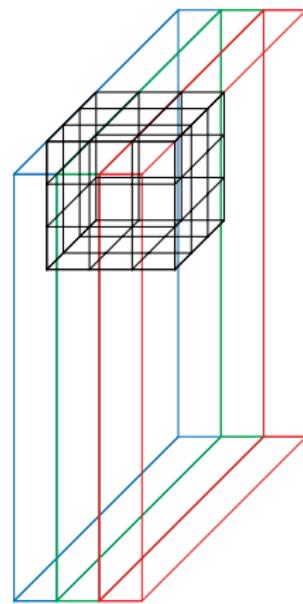
Layers: Convolution



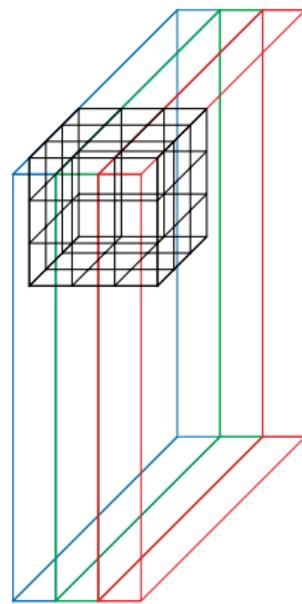
Layers: Convolution



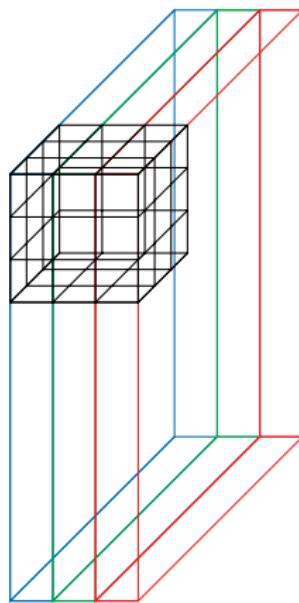
Layers: Convolution



Layers: Convolution



Layers: Convolution



Layers: Convolution

0	1	0
0	1	0
0	1	0

0	0	0
-1	1	0
0	0	0

Layers: Convolution

0	1	0
0	1	0
0	1	0

0	0	0
-1	1	0
0	0	0

Patterns

Layers: Convolution

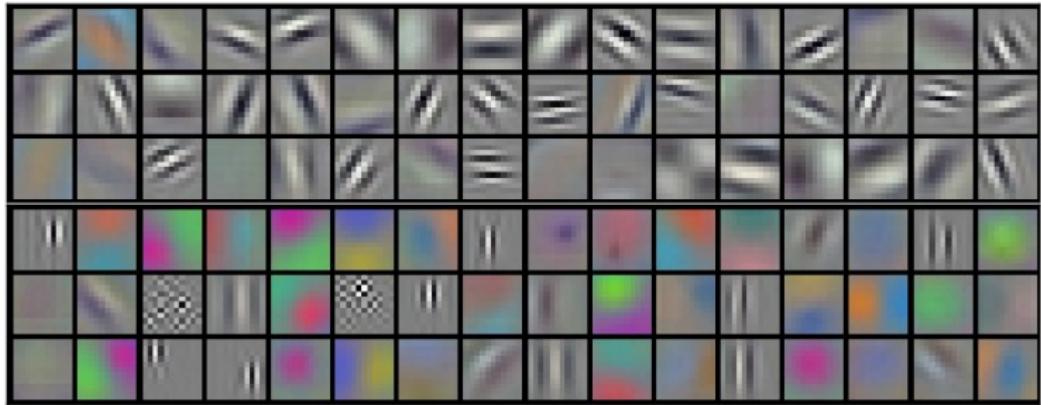
0	1	0
0	1	0
0	1	0

Patterns

0	0	0
-1	1	0
0	0	0

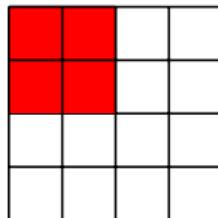
Edges

Layers: Convolution

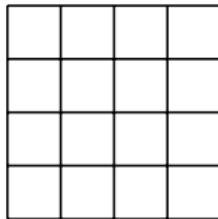


AlexNet: Krizhevsky et al. (2012)

Layer: Convolution

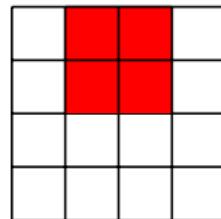


Stride 1

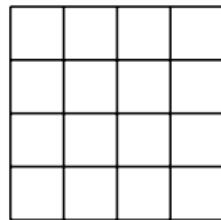


Stride 2

Layer: Convolution

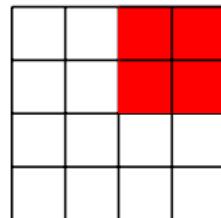


Stride 1

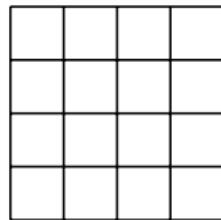


Stride 2

Layer: Convolution

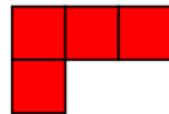
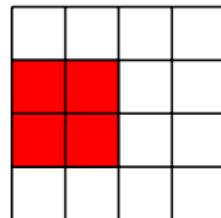


Stride 1

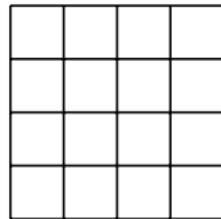


Stride 2

Layer: Convolution

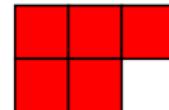
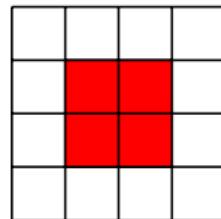


Stride 1

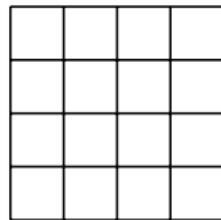


Stride 2

Layer: Convolution

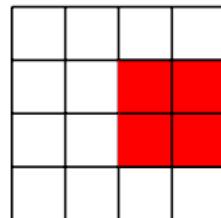


Stride 1

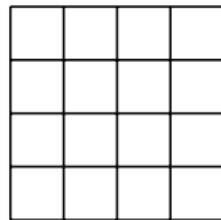


Stride 2

Layer: Convolution

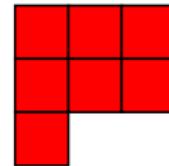
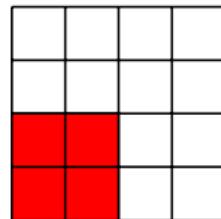


Stride 1

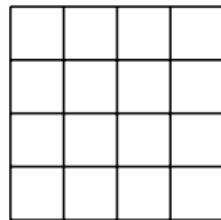


Stride 2

Layer: Convolution

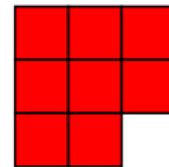
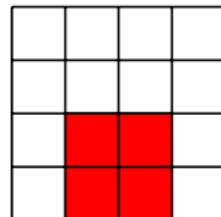


Stride 1

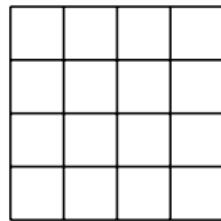


Stride 2

Layer: Convolution

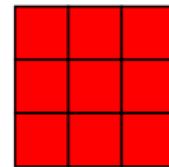
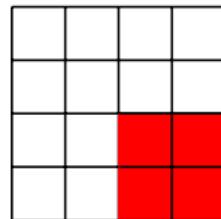


Stride 1

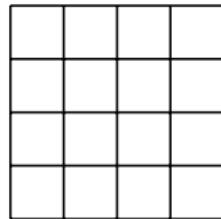


Stride 2

Layer: Convolution

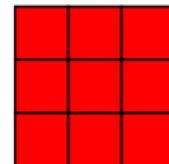
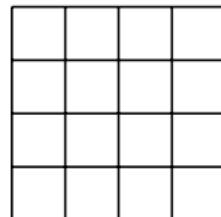


Stride 1

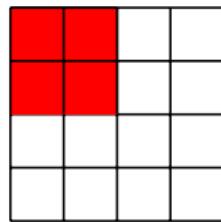


Stride 2

Layer: Convolution

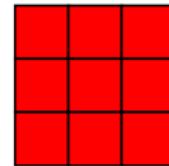
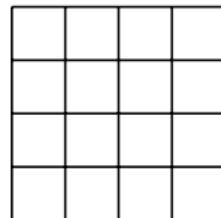


Stride 1

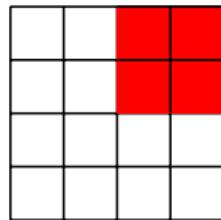


Stride 2

Layer: Convolution

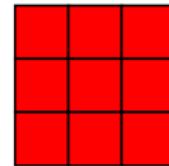
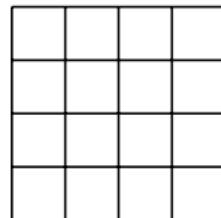


Stride 1

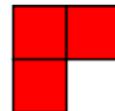
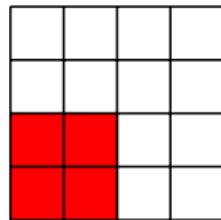


Stride 2

Layer: Convolution

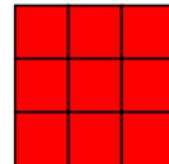
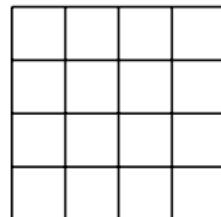


Stride 1

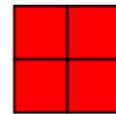
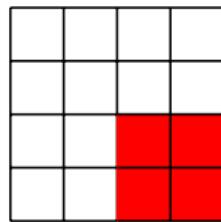


Stride 2

Layer: Convolution

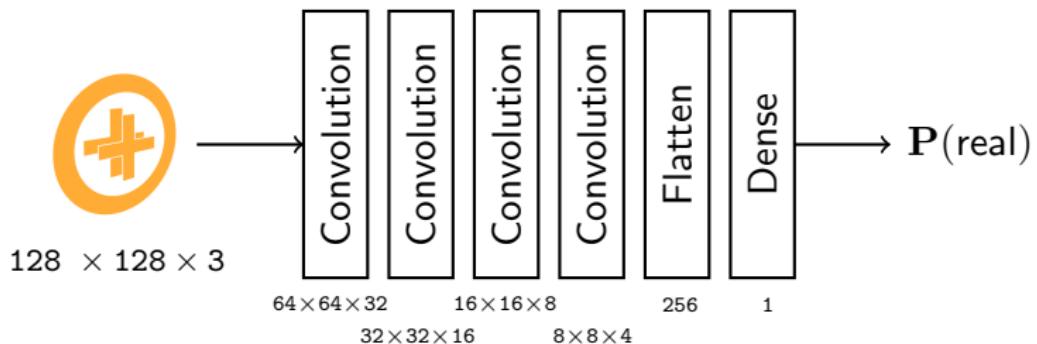


Stride 1



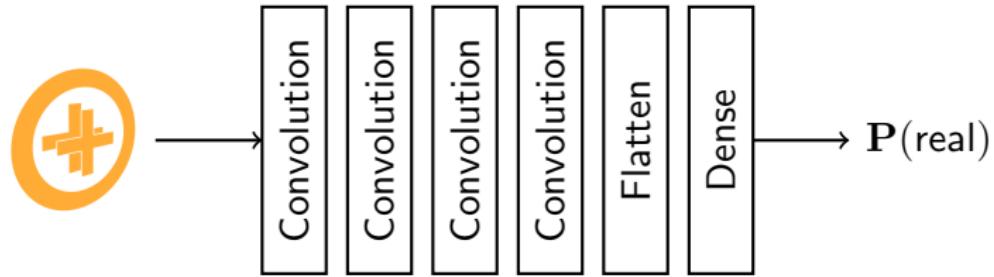
Stride 2

Layer: Convolution

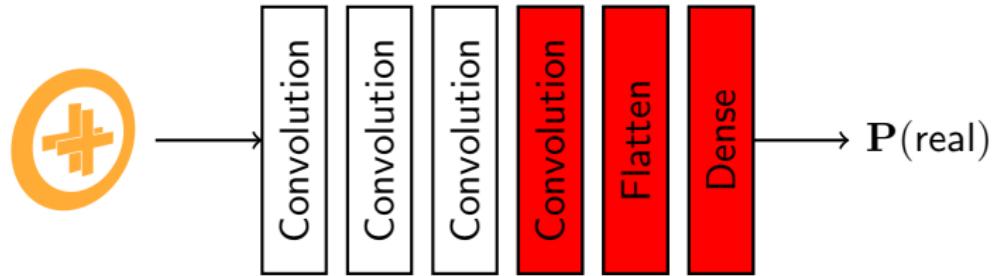


Kernel: 5×5
Stride: 2

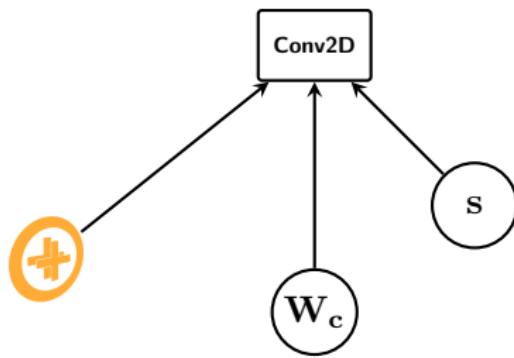
Graph Layer



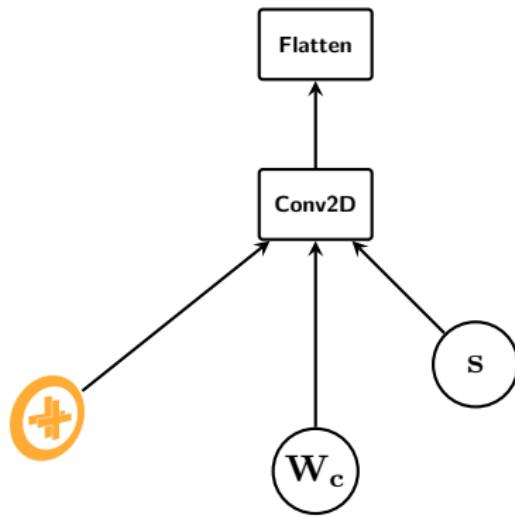
Graph Layer



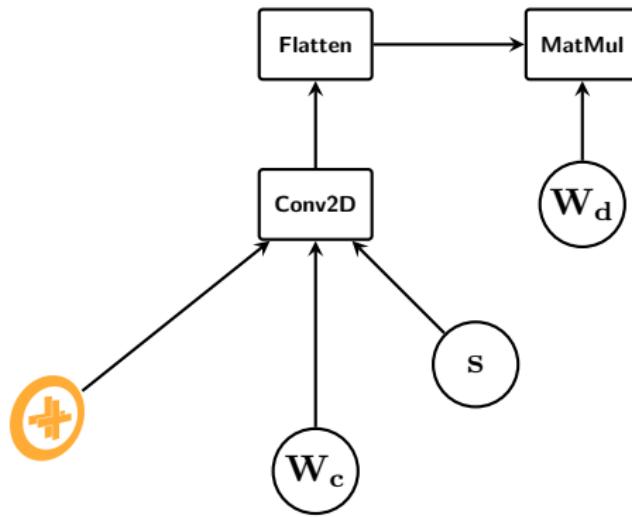
Graph Layer



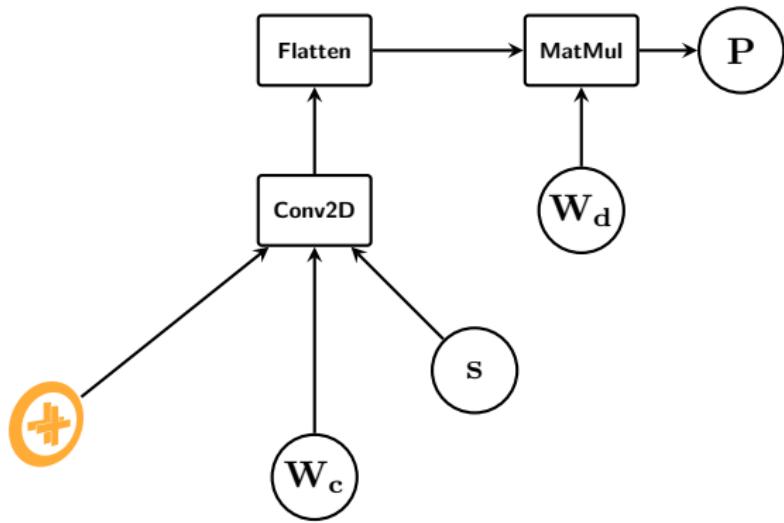
Graph Layer



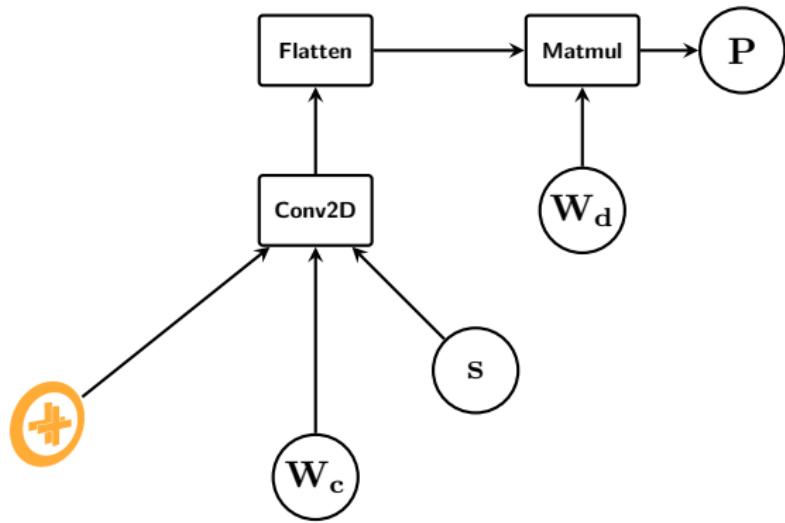
Graph Layer



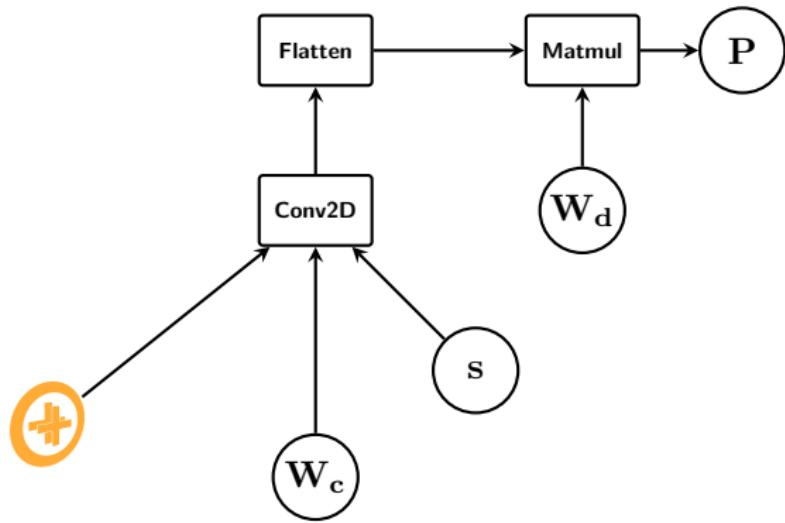
Graph Layer



Graph Layer

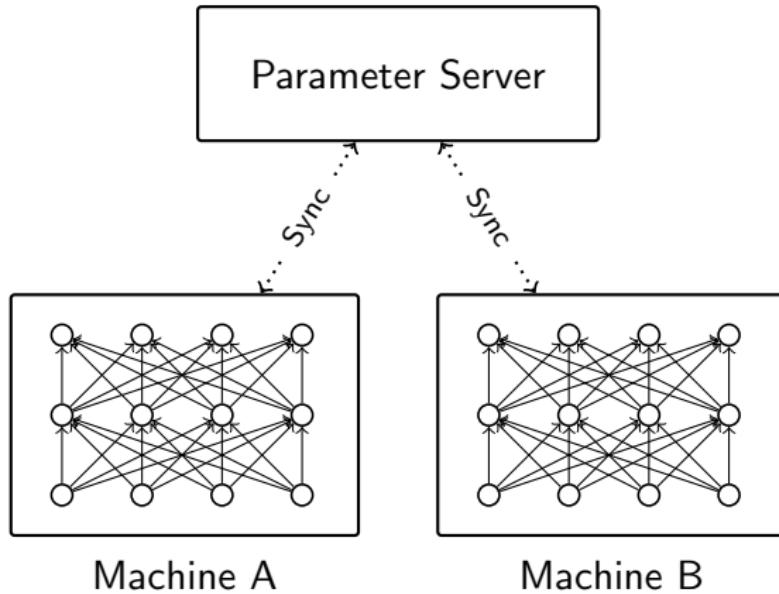


Graph Layer



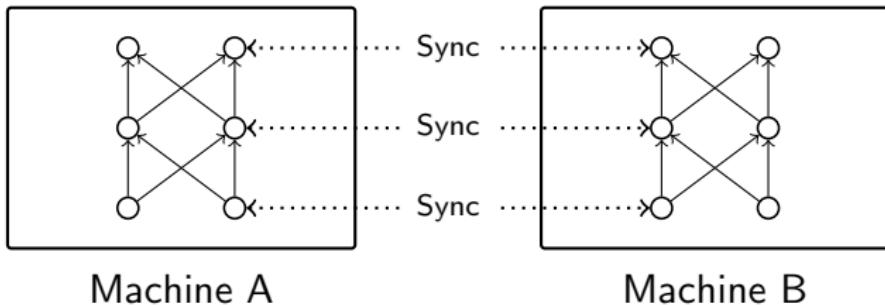
Graph Layer Parallelism

Graph Layer Parallelism



Data Parallelism

Graph Layer Parallelism



Model Parallelism

Static Graphs

Static Graphs

```
Matrix<6, 9> a = ...;
Matrix<6, 9> b = ...;
Matrix<16, 6> c = ...;
scalar s = 7;

d := s * a + b;
e := matmul(c, d);

result := evaluate(e);
```

Static Graphs

```
x = if_clause(condition,  
               then_value,  
               else_value);  
  
while_loop([a, b],  
           condition_function,  
           body_function);
```

Dynamic Graphs

```
Matrix a = ...;
Matrix b = ...;
Matrix c = ...;
scalar s = 7;

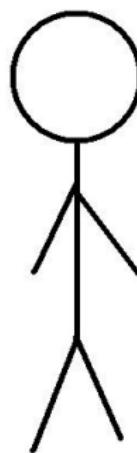
if s > 0 {
    d := s * a + b;
} else {
    d := a - b;
}

while s > 0 {
    x = input();
    c = matmul(c, x);
}

result := c;
```

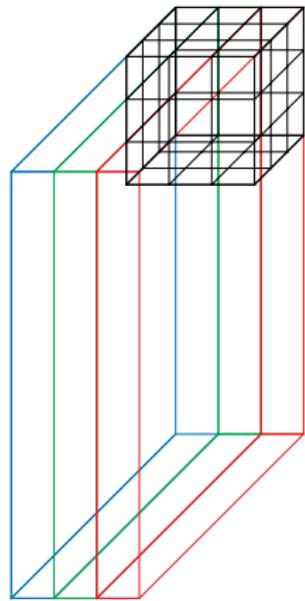
Op Layer

Op Layer

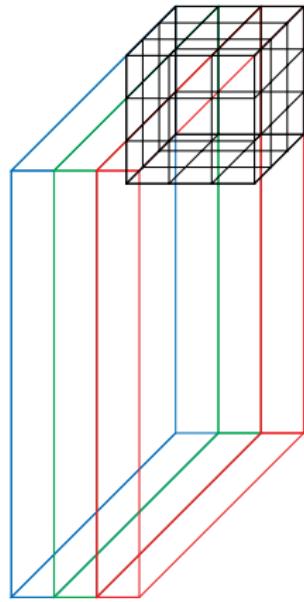


Bob

im2col

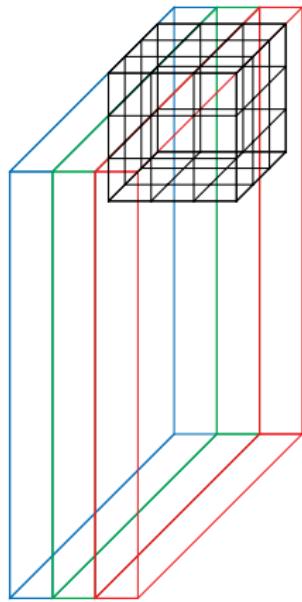


im2col

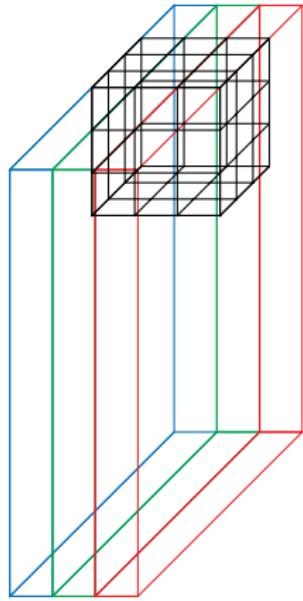


$$\begin{matrix} x_0 & x_1 & x_2 \\ \dots & & \\ x_{24} & x_{25} & x_{26} \end{matrix}$$

im2col

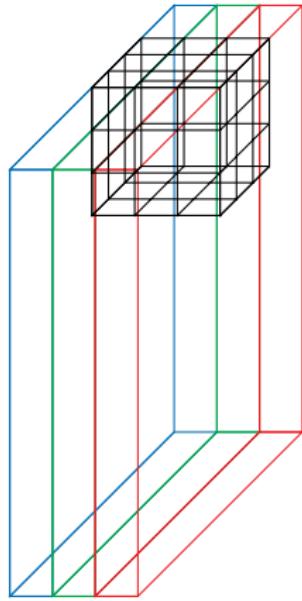

$$\begin{array}{|c|c|c|} \hline x_0 & x_1 & x_2 \\ \hline x_0 & x_1 & x_2 \\ \hline \end{array} \quad \cdots \quad \begin{array}{|c|c|c|} \hline x_{24} & x_{25} & x_{26} \\ \hline x_{24} & x_{25} & x_{26} \\ \hline \end{array}$$

im2col

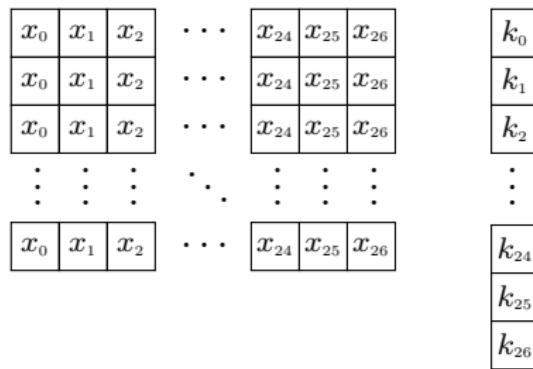
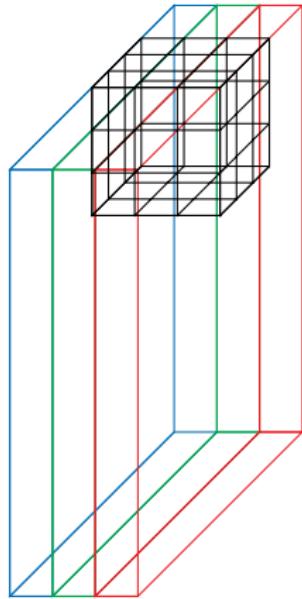


x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}

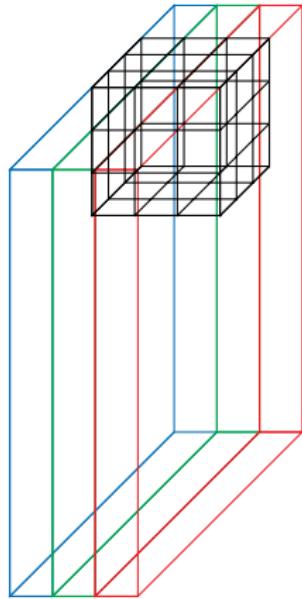
im2col


$$\begin{array}{|c|c|c|} \hline x_0 & x_1 & x_2 \\ \hline x_0 & x_1 & x_2 \\ \hline x_0 & x_1 & x_2 \\ \hline \vdots & \vdots & \vdots \\ \hline x_0 & x_1 & x_2 \\ \hline \end{array} \quad \cdots \quad \begin{array}{|c|c|c|} \hline x_{24} & x_{25} & x_{26} \\ \hline x_{24} & x_{25} & x_{26} \\ \hline x_{24} & x_{25} & x_{26} \\ \hline \vdots & \vdots & \vdots \\ \hline x_{24} & x_{25} & x_{26} \\ \hline \end{array}$$

im2col

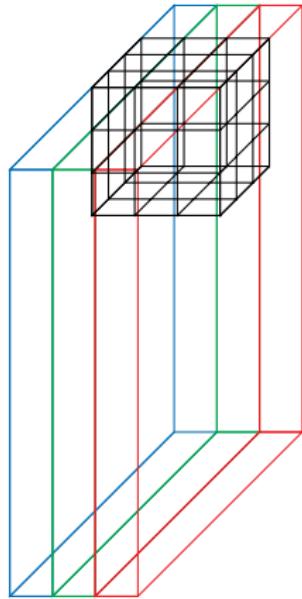


im2col



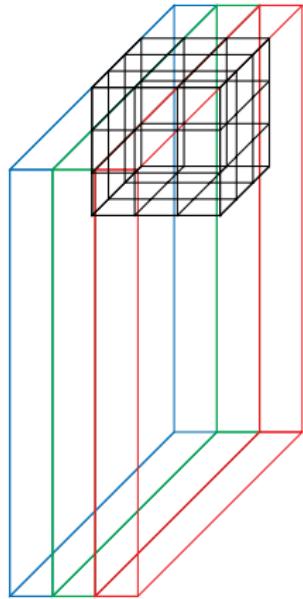
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}	k_0	k_0
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}	k_1	k_1
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}	k_2	k_2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}	k_{24}	k_{24}
							k_{25}	k_{25}
							k_{26}	k_{26}

im2col



x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}	k_0	k_0	k_0
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}	k_1	k_1	k_1
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}	k_2	k_2	k_2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
x_0	x_1	x_2	⋮	x_{24}	x_{25}	x_{26}	k_{24}	k_{24}	k_{24}
							k_{25}	k_{25}	k_{25}
							k_{26}	k_{26}	k_{26}

im2col



$$\begin{matrix} x_0 & x_1 & x_2 \\ x_0 & x_1 & x_2 \\ x_0 & x_1 & x_2 \end{matrix}$$

⋮

$$\begin{matrix} x_0 & x_1 & x_2 \\ x_0 & x_1 & x_2 \\ x_0 & x_1 & x_2 \end{matrix}$$

⋮

$$\begin{matrix} \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{matrix}$$

$$\begin{matrix} x_0 & x_1 & x_2 \\ x_0 & x_1 & x_2 \\ x_0 & x_1 & x_2 \end{matrix}$$

⋮

$$\begin{matrix} x_{24} & x_{25} & x_{26} \\ x_{24} & x_{25} & x_{26} \\ x_{24} & x_{25} & x_{26} \end{matrix}$$

⋮

$$\begin{matrix} x_{24} & x_{25} & x_{26} \\ x_{24} & x_{25} & x_{26} \\ x_{24} & x_{25} & x_{26} \end{matrix}$$

×

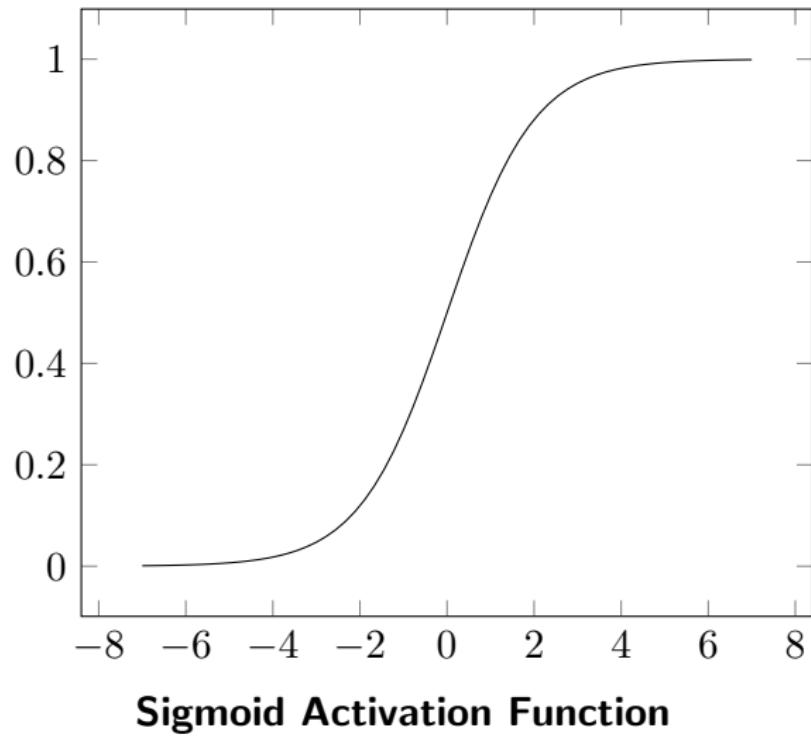
$$\begin{matrix} k_0 & k_0 & k_0 \\ k_1 & k_1 & k_1 \\ k_2 & k_2 & k_2 \end{matrix}$$

$$\begin{matrix} \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{matrix}$$

$$\begin{matrix} k_{24} & k_{24} & k_{24} \\ k_{25} & k_{25} & k_{25} \\ k_{26} & k_{26} & k_{26} \end{matrix}$$

Kernel Layer

Kernel Layer



Kernel Layer

Kernel Layer

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Forward Pass

Kernel Layer

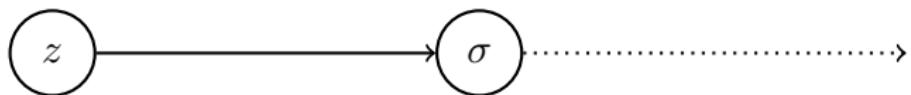
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Forward Pass

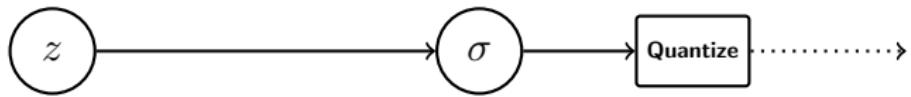
$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Backward Pass

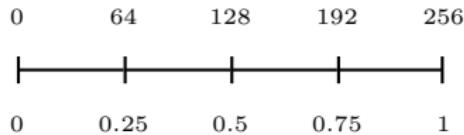
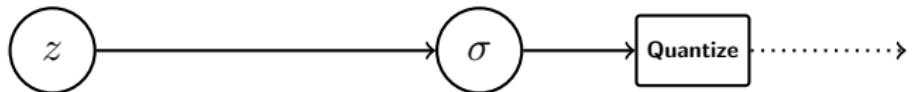
Quantization



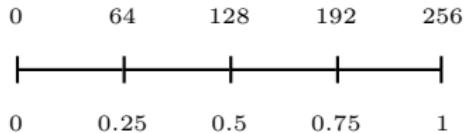
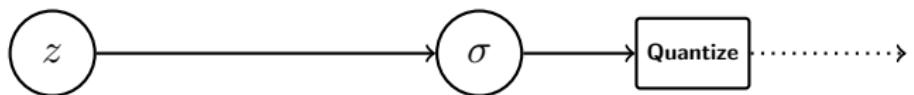
Quantization



Quantization



Quantization



github.com/google/gemmlowp

Hardware Layer

Hardware Layer



CPU

Hardware Layer



CPU



GPU

Hardware Layer



CPU

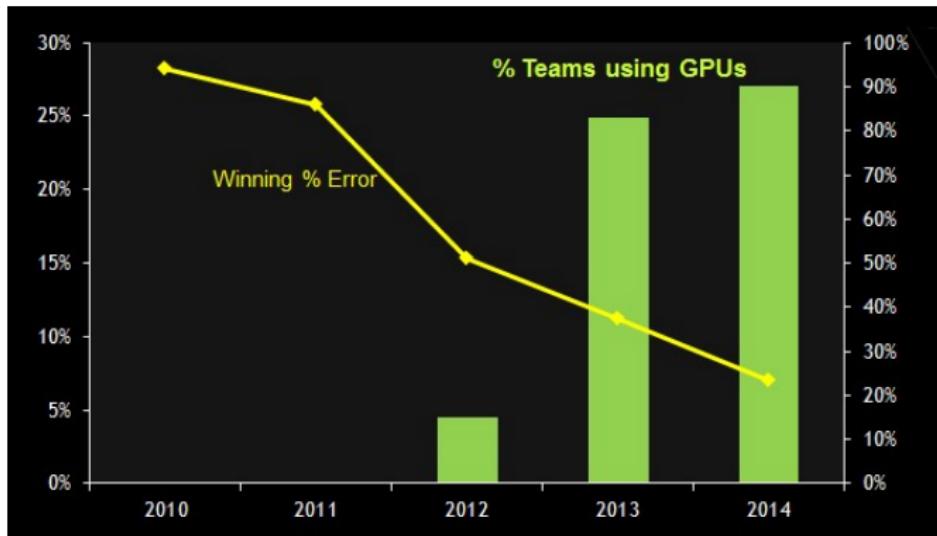


GPU



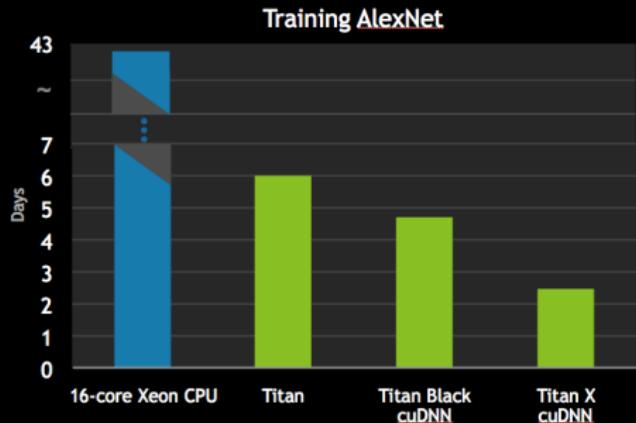
ASIC

GPUs



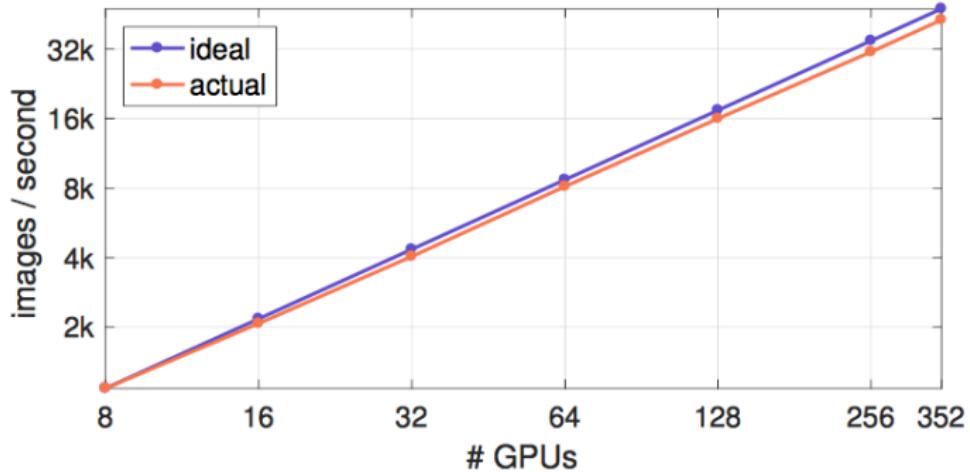
Teams using GPUs at ImageNet Competition

TITAN X FOR DEEP LEARNING



Training Time for ImageNet

GPGUs



Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
Goyal et al. (2017)

THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG
PILE OF LINEAR ALGEBRA, THEN COLLECT
THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

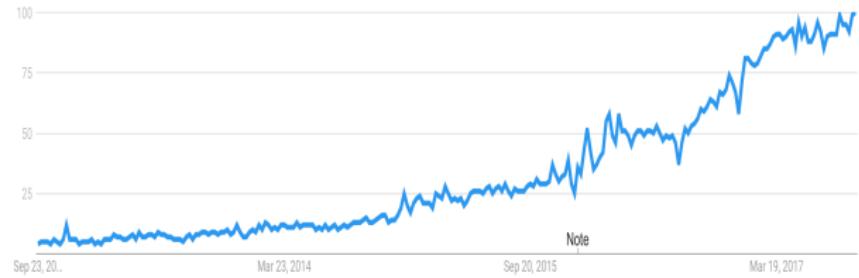
JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.



Hardware: GPU



NVIDIA Stock



Deep Learning Trend

Hardware: GPU

Hardware: GPU



Titan X

Hardware: GPU



Titan X



Jetson TX2

Hardware: GPU



Titan X



Jetson TX2



DGX-1

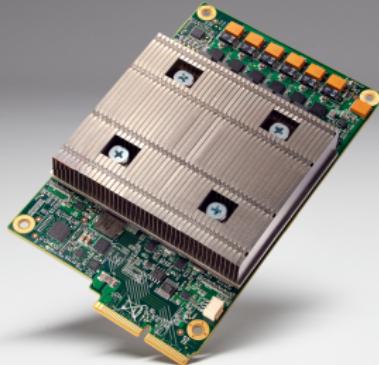
Hardware: Big Basin



Big Basin

- ▶ 8 NVIDIA Tesla P100 GPUs
- ▶ NVLink (12x faster than PCIe)
- ▶ 16 GB RAM
- ▶ 10.6 TFLOPS
- ▶ Reduced Precision Computing

Hardware: TPU

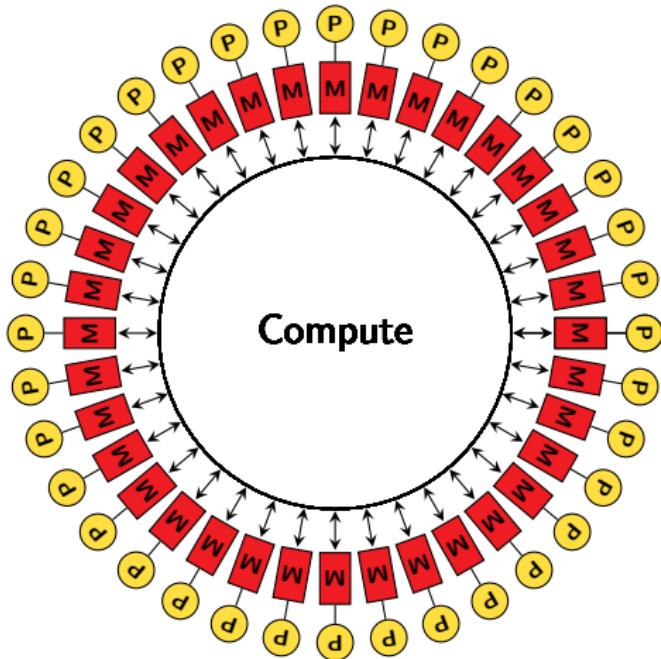


TPU

- ▶ *Coprocessor*
- ▶ 92 TOPS for 8-bit int
- ▶ 42 TFLOPS (TPU2)
- ▶ 24 MB Memory

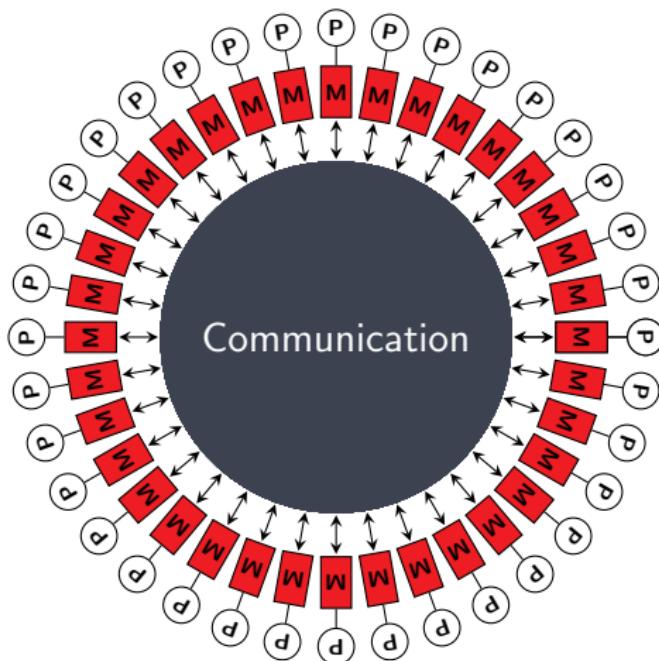
Hardware: IPU

Hardware: IPU



Bulk Synchronous Parallelism

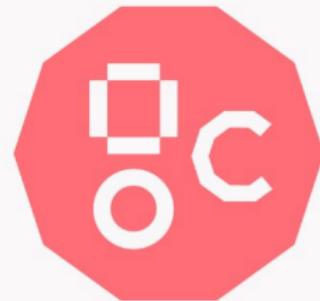
Hardware: IPU



Bulk Synchronous Parallelism

Hardware: IPU

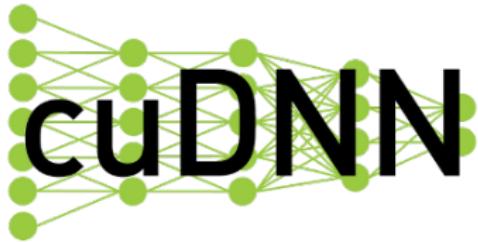
- ▶ Startup from Bristol (UK)
- ▶ Graphcore Colossus
- ▶ TBR later this year
- ▶ 1000 Processors/Chip
- ▶ Mixed Precision Arithmetic



Graphcore

Kernel Layer

Kernel Layer

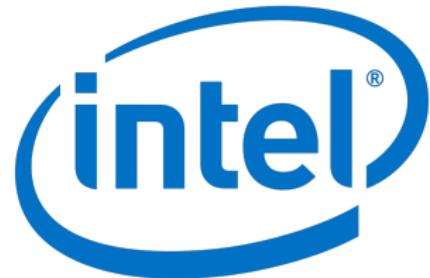


cuDNN/cuBLAS

Kernel Layer



cuDNN/cuBLAS



MKL

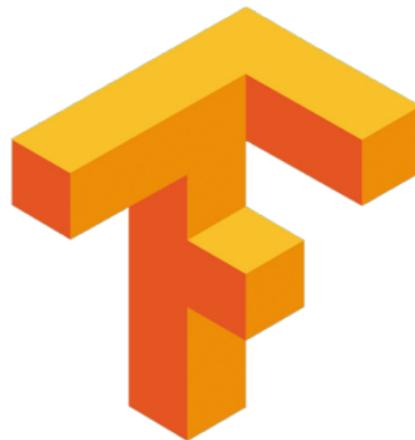
Demo: Convolution w/ cuDNN

Demo: Custom TensorFlow Op

Graph Layer

Graph Layer: TensorFlow

- ▶ Google (2015)
- ▶ Static Graphs
- ▶ GPUs ✓
- ▶ Distributed ✓
- ▶ C++ API ✗
- ▶ Very “Deployable”



Graph Layer: PyTorch/Caffe2



- ▶ Facebook (2016)
- ▶ Dynamic Graphs
- ▶ For Research
- ▶ GPUs ✓
- ▶ Distributed ✓
- ▶ Pure Python

- ▶ Facebook (2017)
- ▶ Static Graphs
- ▶ For Deployment
- ▶ GPUs ✓
- ▶ Distributed ✓
- ▶ C++ API ✓

Graph Layer: MXNet

- ▶ Community, then Amazon
- ▶ Apache Incubator
- ▶ Highly Modular
- ▶ GPUs ✓
- ▶ Distributed ✓
- ▶ APIs for many languages
(C++ ✓)



Graph Layer: Others

Microsoft
CNTK **theano**



Layer Layer

Demo: MXNet Classifier

Demo: Loading a TensorFlow graph

How do I continue?

Resources

- ▶ Machine Learning @ Coursera
- ▶ Deep Learning @ Coursera
- ▶ Deep Learning (Google) @ Udacity

Resources

- ▶ Machine Learning @ Coursera
- ▶ Deep Learning @ Coursera
- ▶ Deep Learning (Google) @ Udacity

- ▶ mxnet.incubator.apache.org/architecture
- ▶ github.com/dmlc/nnvm
- ▶ Intro to Parallel Programming (NVIDIA) @ Udacity
- ▶ A Tour of TensorFlow, Goldsborough (2016)

Resources

- ▶ Machine Learning @ Coursera
- ▶ Deep Learning @ Coursera
- ▶ Deep Learning (Google) @ Udacity

- ▶ mxnet.incubator.apache.org/architecture
- ▶ github.com/dmlc/nnvm
- ▶ Intro to Parallel Programming (NVIDIA) @ Udacity
- ▶ A Tour of TensorFlow, Goldsborough (2016)

github.com/peter-can-talk/cppcon-2017

Q & A