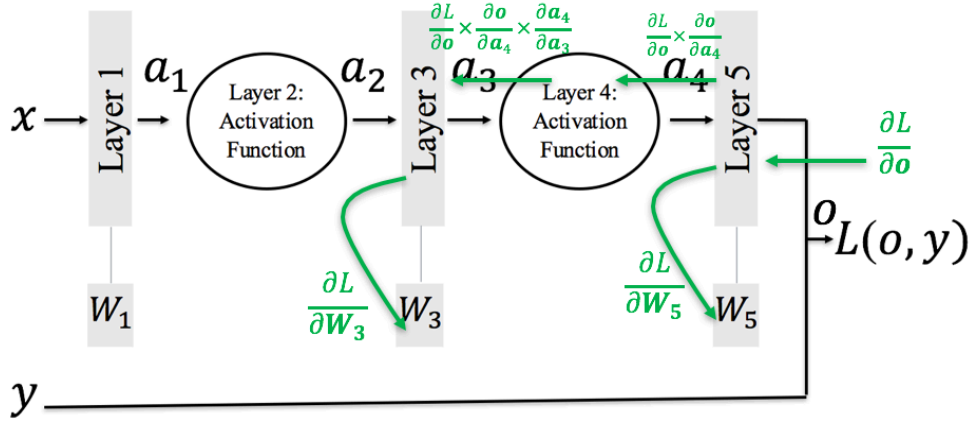
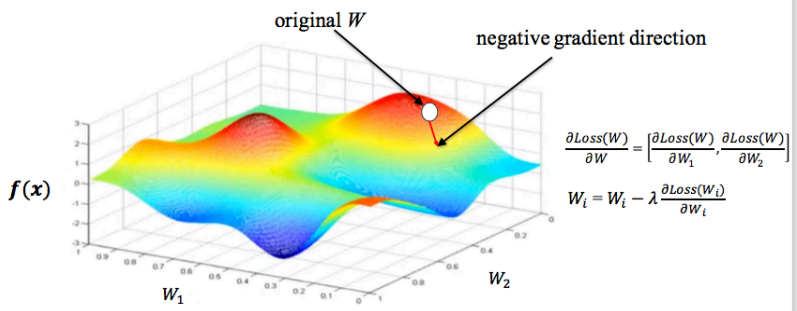
Deep Learning (for Computer Vision)

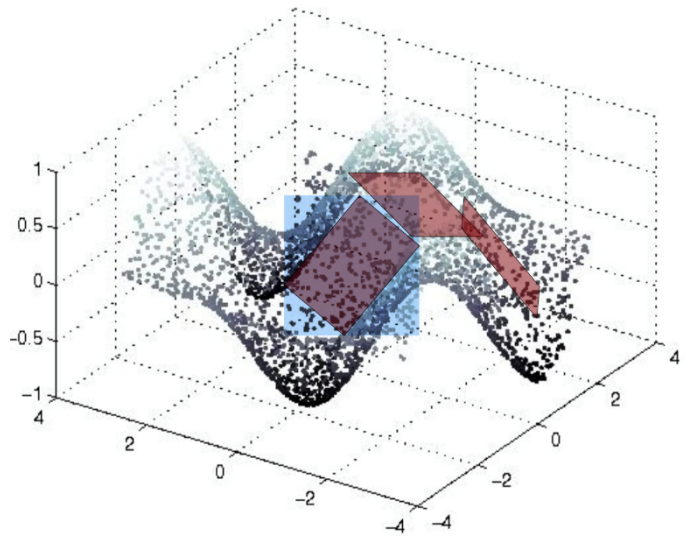
Arjun Jain

Recap

- Data-driven paradigm, CIFAR-10 dataset
- NN classifier, Linear Classifier
- Loss functions, Optimization
- Feed forward networks
- Back-propagation and chain rule
- Activation Layers
  - Sigmoid
  - ReLU
  - …







Agenda Today

- Building blocks
  - Linear
  - Convolution
  - Max Pooling
  - Cross Entropy
  - Dropout
  - Batch Normalization
- Weight initializations, data preprocessing
- Choosing hyperparameters, baby sitting the learning process

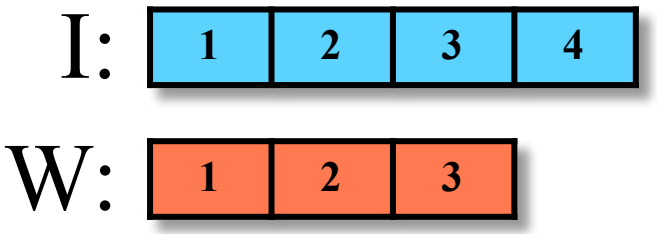
Sources

A lot of the material has been shamelessly and gratefully collected from:

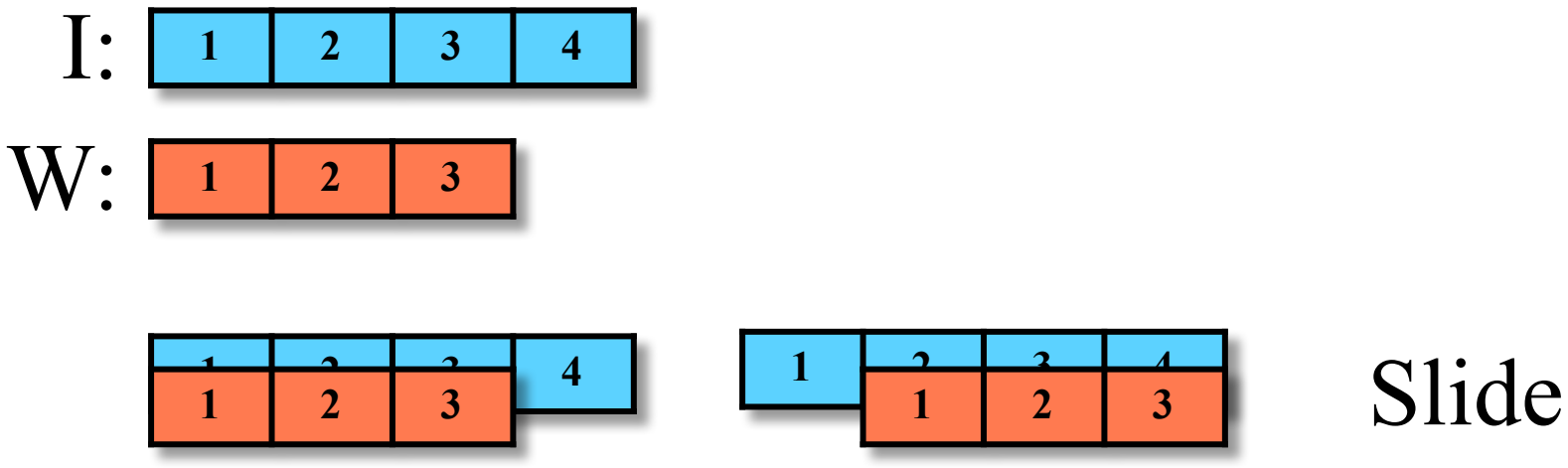
- **http://cs231n.stanford.edu/**
- http://cs231n.github.io/convolutional-networks/
- http://peterroelants.github.io/posts/neural_network_implementation_part04/

Building Blocks:
Convolution

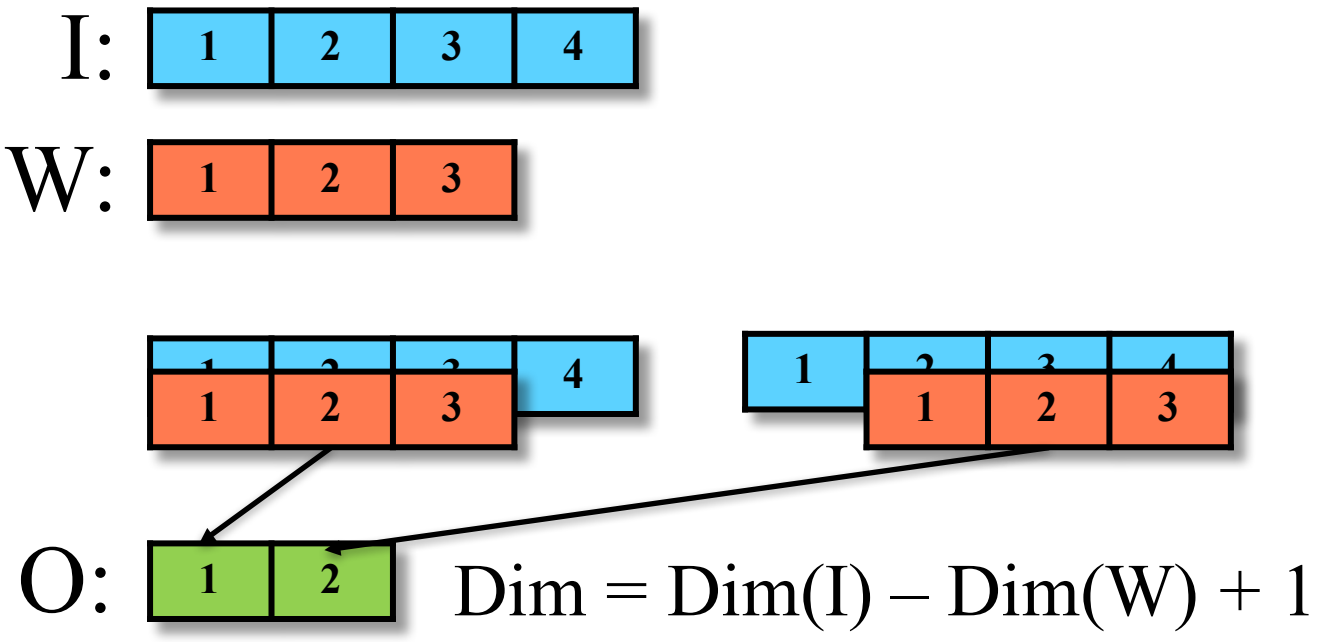
Building Blocks – Convolution (Discrete 1D)



Building Blocks – Convolution (Discrete 1D)



Building Blocks – Convolution (Discrete 1D)



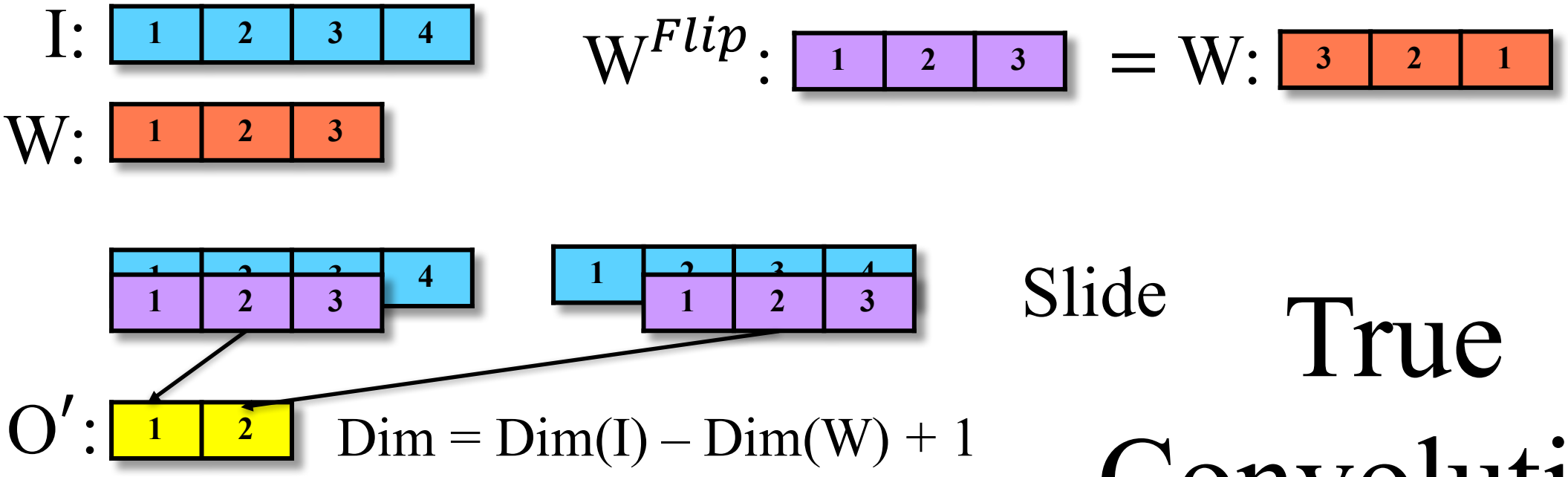
$$O_1 = I_1 W_1 + I_2 W_2 + I_3 W_3$$

$$O_2 = I_2 W_1 + I_3 W_2 + I_4 W_3$$

Correlation

$$O_i = \sum_{j=1}^{Dim(W)} I_{j+i-1} W_j$$

Building Blocks – Convolution (Discrete 1D)



Slide

$$O'_1 = I_1 W_1^{Flip} + I_2 W_2^{Flip} + I_3 W_3^{Flip}$$

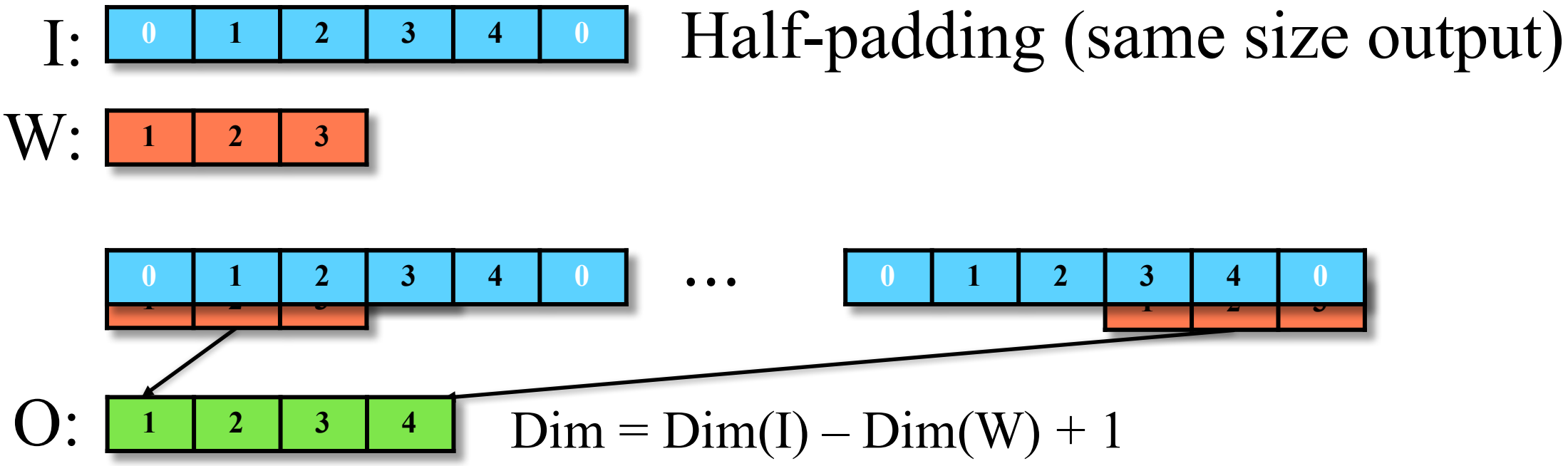
$$O'_2 = I_2 W_1^{Flip} + I_3 W_2^{Flip} + I_4 W_3^{Flip}$$

Dim = Dim(I) – Dim(W) + 1

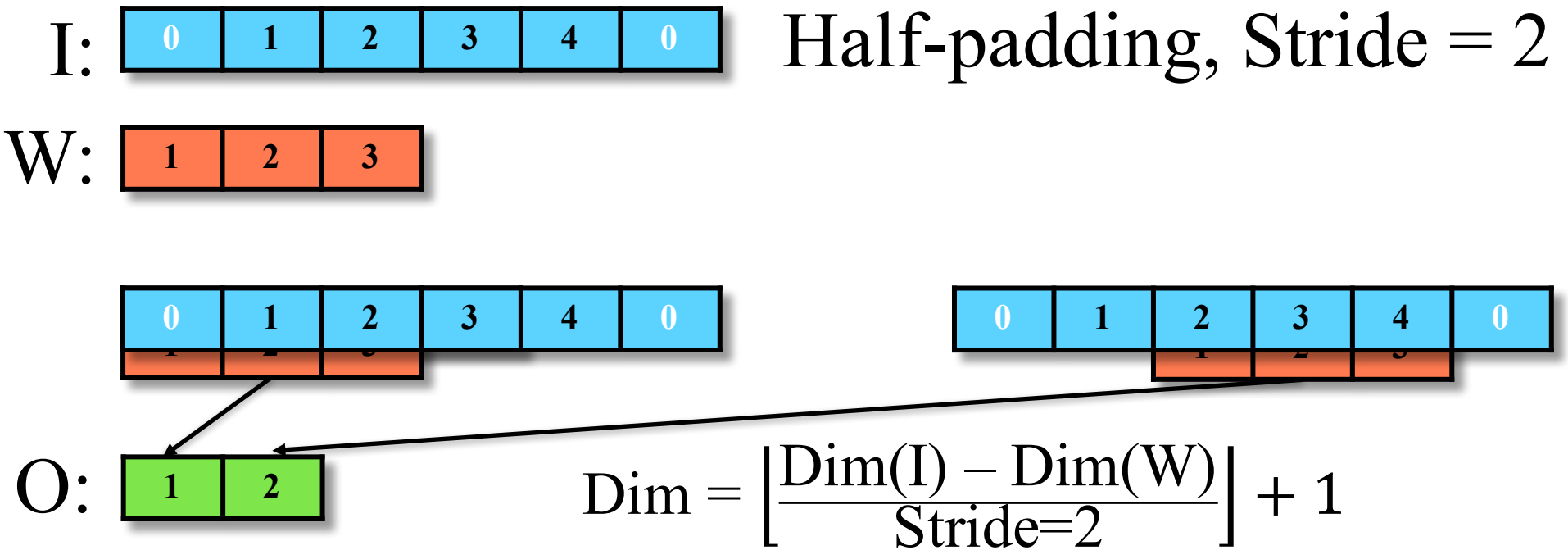
True Convolution

$$O_i = \sum_{j=1}^{Dim(W)} I_{j+i-1} W_{Dim(W)-j+1}$$

Building Blocks – Convolution (Discrete 1D)

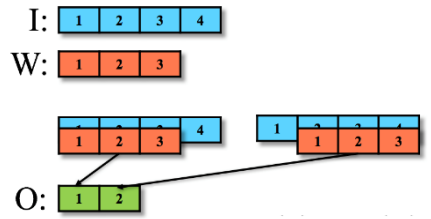


Building Blocks – Convolution (Discrete 1D)



Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$



$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

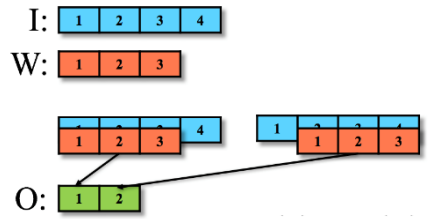
$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

Correlation

Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$



$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

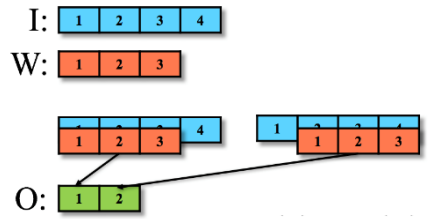
Correlation

Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{O}} = [\partial L O_1 \quad \partial L O_2]$$



$O_1 = I_1 W_1 + I_2 W_2 + I_3 W_3$

$O_2 = I_2 W_1 + I_3 W_2 + I_4 W_3$

Correlation

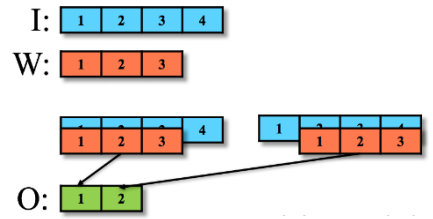
Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{O}} = [\partial LO_1 \quad \partial LO_2]$$

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = [\partial LO_1 \times I_1 + \partial LO_2 \times I_2 \quad \partial LO_1 \times I_2 + \partial LO_2 \times I_3 \quad \partial LO_1 \times I_3 + \partial LO_2 \times I_4]$$



$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

Correlation

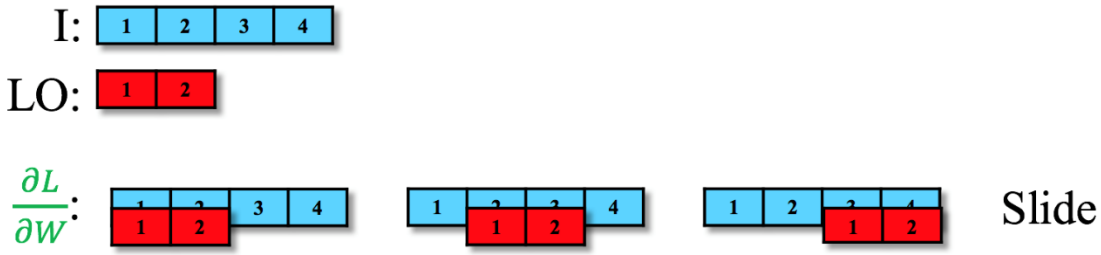
Building Blocks – Convolution – Backward

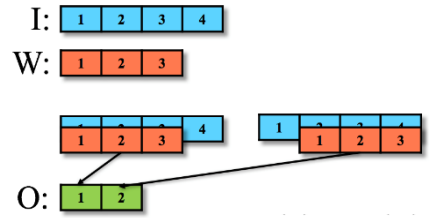
$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{O}} = [\partial LO_1 \quad \partial LO_2]$$

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = [\partial LO_1 \times I_1 + \partial LO_2 \times I_2 \quad \partial LO_1 \times I_2 + \partial LO_2 \times I_3 \quad \partial LO_1 \times I_3 + \partial LO_2 \times I_4]$$





$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

Correlation

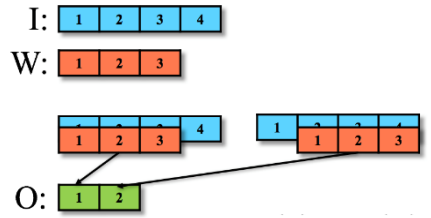
Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{O}} = [\partial LO_1 \quad \partial LO_2]$$

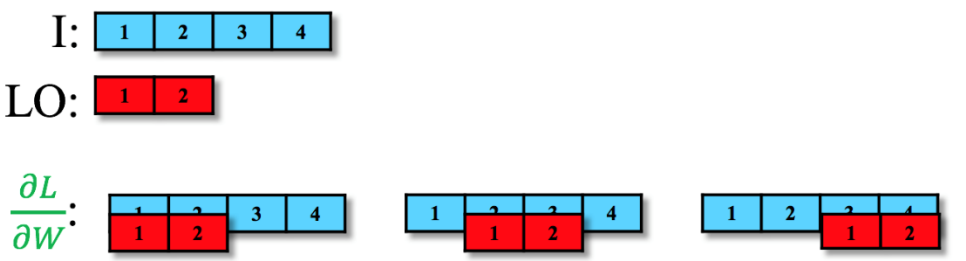
$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = [\partial LO_1 \times I_1 + \partial LO_2 \times I_2 \quad \partial LO_1 \times I_2 + \partial LO_2 \times I_3 \quad \partial LO_1 \times I_3 + \partial LO_2 \times I_4]$$



$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

Correlation



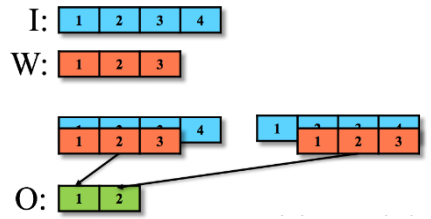
$$\frac{\partial L}{\partial \boldsymbol{W}} = \text{Correlation(I, LO)}$$

Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{O}} = [\partial L O_1 \quad \partial L O_2]$$



$O_1 = I_1 W_1 + I_2 W_2 + I_3 W_3$

$O_2 = I_2 W_1 + I_3 W_2 + I_4 W_3$

Correlation

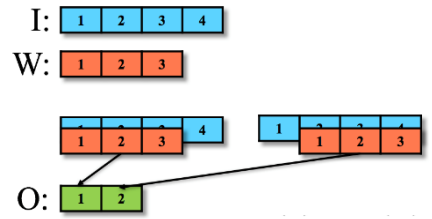
Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{O}} = [\partial LO_1 \quad \partial LO_2]$$

$$\frac{\partial L}{\partial \boldsymbol{I}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = [\partial LO_1 \times W_1 \quad \partial LO_1 \times W_2 + \partial LO_2 \times W_1 \quad \partial LO_1 \times W_3 + \partial LO_2 \times W_2 \quad \partial LO_2 \times W_3]$$



$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

Correlation

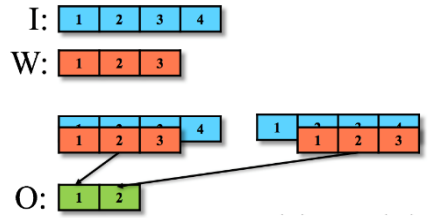
Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{O}} = [\partial LO_1 \quad \partial LO_2]$$

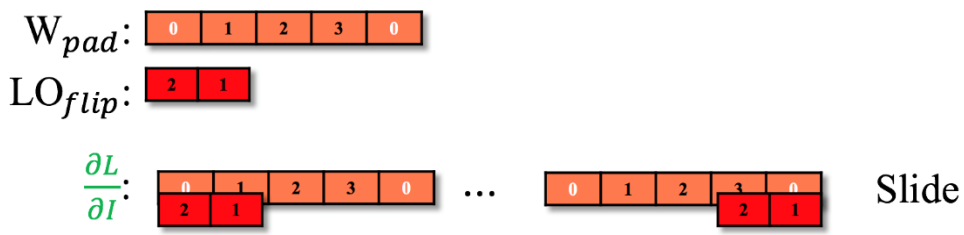
$$\frac{\partial L}{\partial \boldsymbol{I}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = [\partial LO_1 \times W_1 \quad \partial LO_1 \times W_2 + \partial LO_2 \times W_1 \quad \partial LO_1 \times W_3 + \partial LO_2 \times W_2 \quad \partial LO_2 \times W_3]$$



$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

Correlation



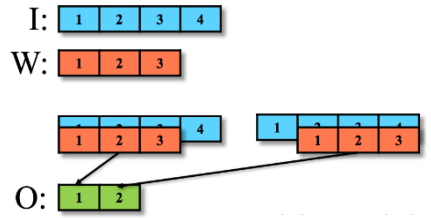
Building Blocks – Convolution – Backward

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

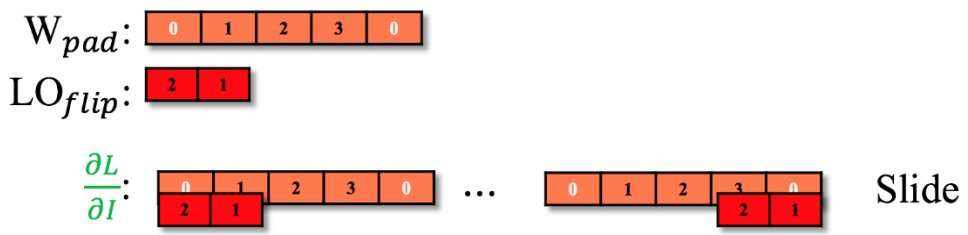
$$\frac{\partial L}{\partial \boldsymbol{O}} = [\partial LO_1 \quad \partial LO_2]$$

$$\frac{\partial L}{\partial \boldsymbol{I}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = [\partial LO_1 \times W_1 \quad \partial LO_1 \times W_2 + \partial LO_2 \times W_1 \quad \partial LO_1 \times W_3 + \partial LO_2 \times W_2 \quad \partial LO_2 \times W_3]$$



$$O_1 = I_1 W_1 + I_2 W_2 + I_3 W_3$$
$$O_2 = I_2 W_1 + I_3 W_2 + I_4 W_3$$

Correlation



$$\frac{\partial L}{\partial \boldsymbol{I}} = \text{Correlation}(W_{pad}, \text{LO}_{flip})$$

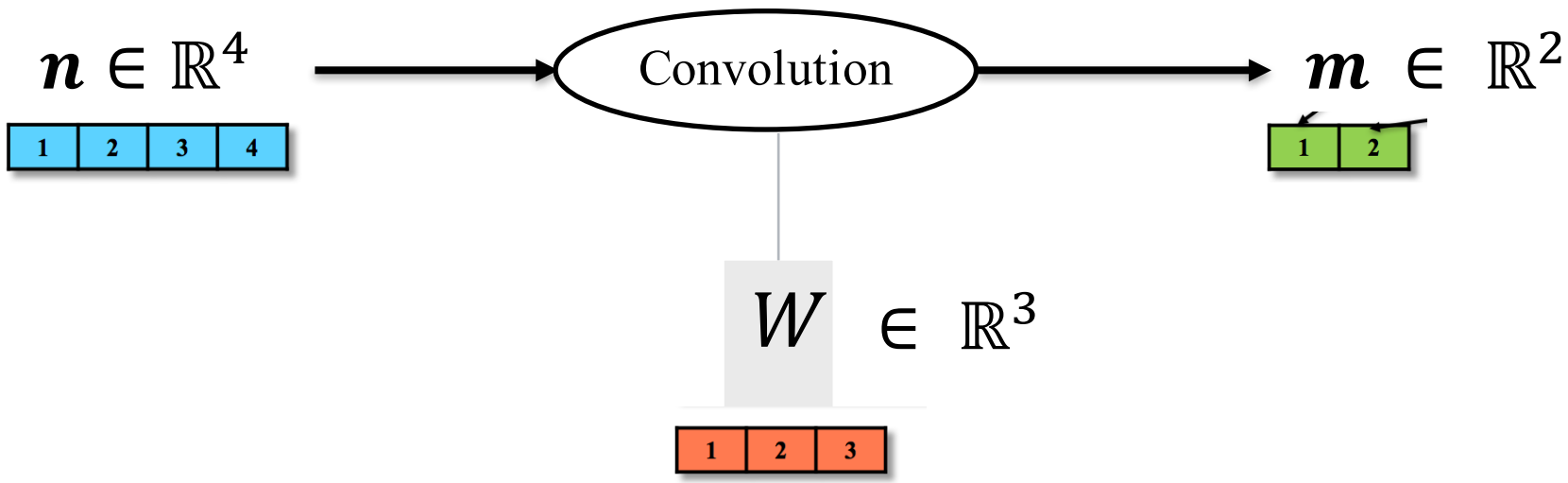
Building Blocks – Convolution – Forward

```
require 'nn';
n = torch.rand(4):reshape(1,1,4)
print(n)
```

```
(1,.,.) =
  0.3347  0.5901  0.7132  0.3187
[torch.DoubleTensor of size 1x1x4]
```

```
conv = nn.SpatialConvolutionMM(1,1,3,1)
conv.bias:fill(0)
m = conv:forward(n)
print(m)
```

```
(1,.,.) =
  0.1897  0.1130
[torch.DoubleTensor of size 1x1x2]
```



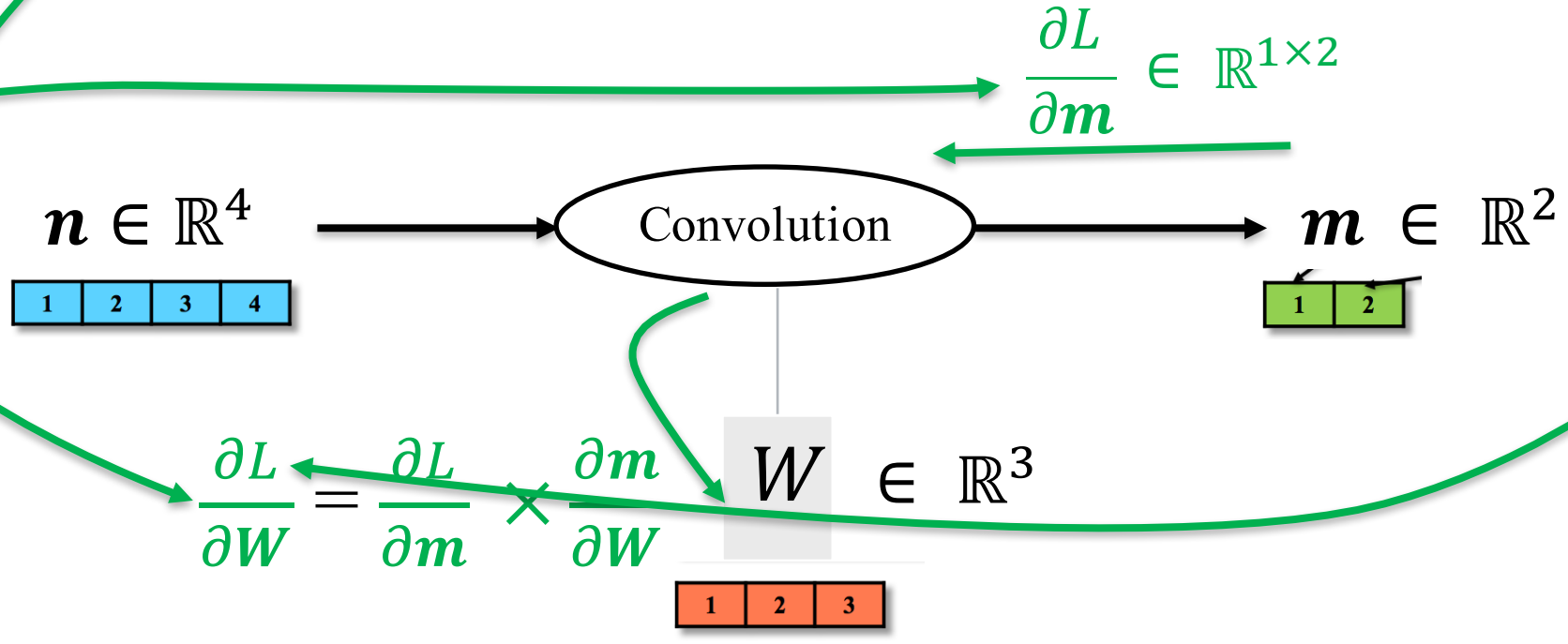
Building Blocks – Convolution – Backward

```
nextgrad=torch.rand(2):reshape(1,1,2)
conv:backward(n, nextgrad)
print(conv.gradWeight)
```

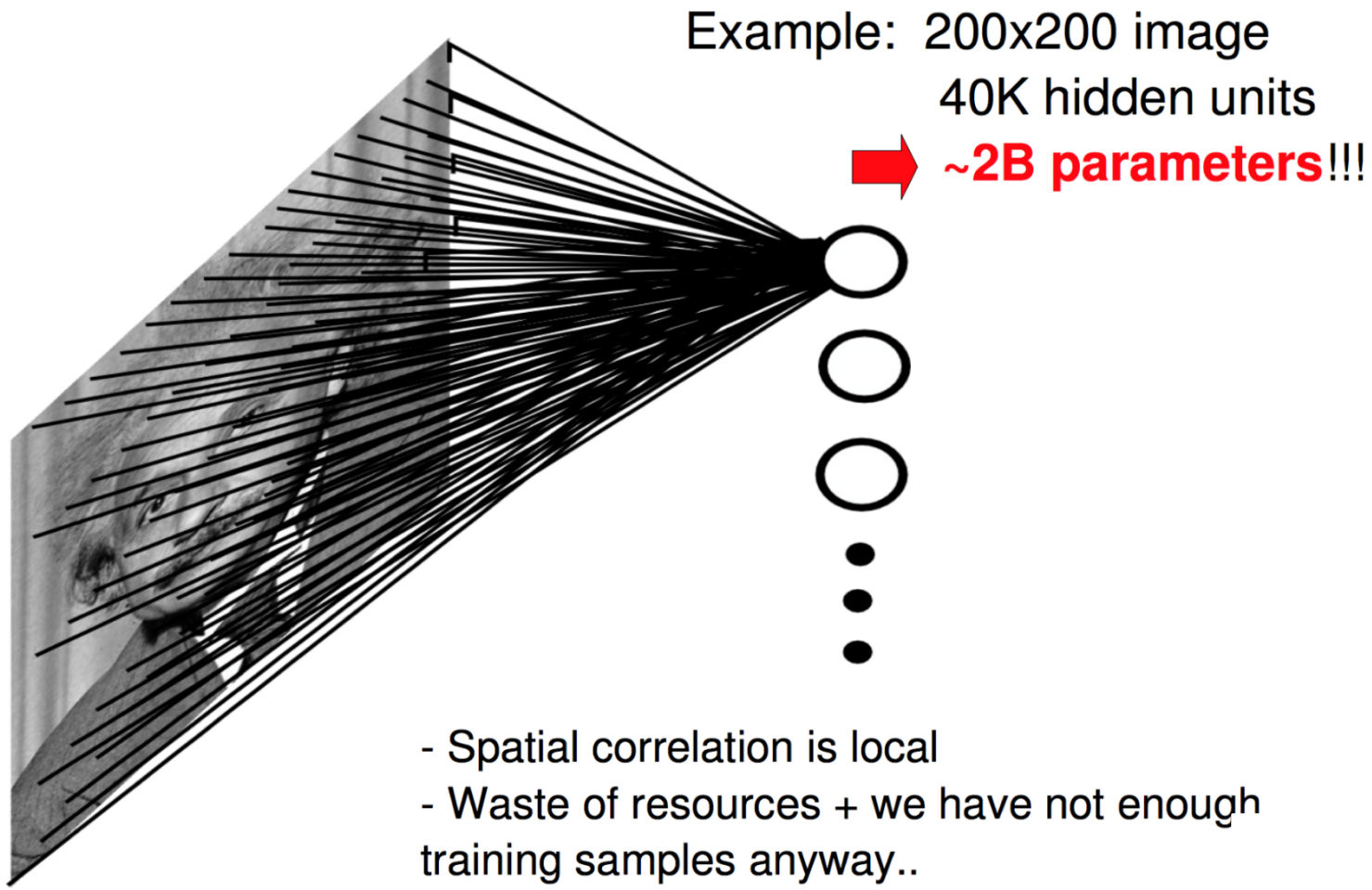
```
 0.6464   0.8428   0.5257
[torch.DoubleTensor of size 1x3]
```

```
convback = nn.SpatialConvolutionMM(1,1,2,1)
convback.bias:fill(0)
convback.weight:copy(nextgrad:reshape(1,2))
gradWeight = convback:forward(n)
print(gradWeight)
```

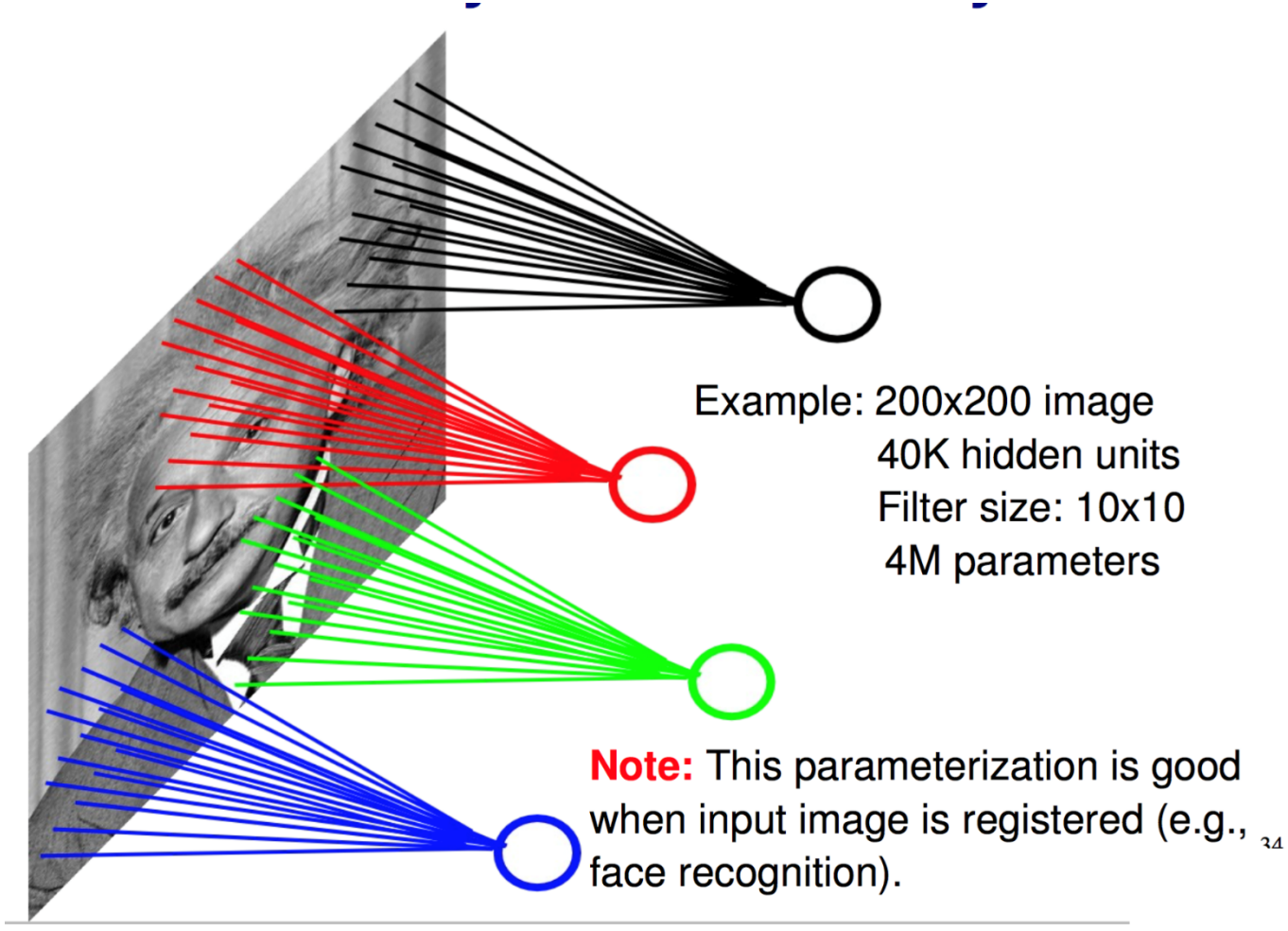
```
(1,.,.) =
  0.6464   0.8428   0.5257
[torch.DoubleTensor of size 1x1x3]
```



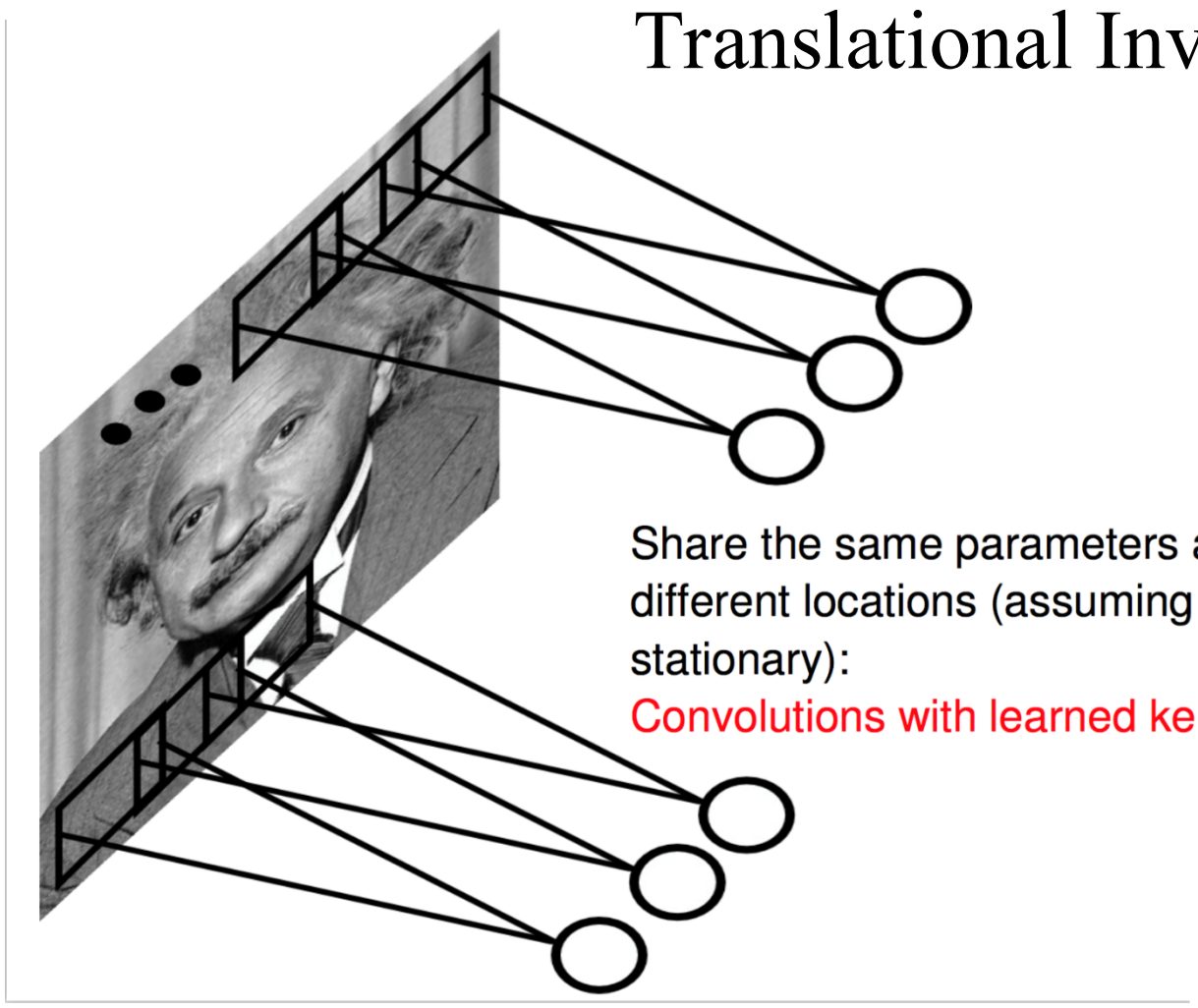
Building Blocks - Convolution



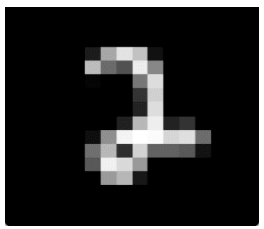
Building Blocks - Convolution



Building Blocks - Convolution



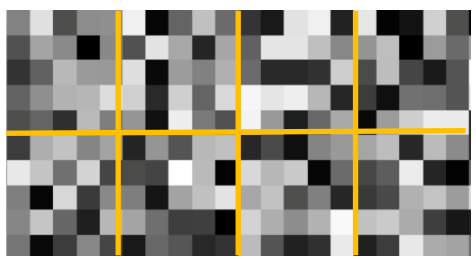
Building Blocks - Convolution



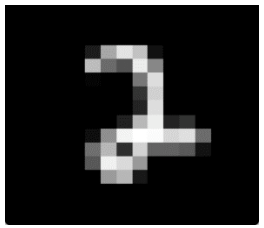
$\boldsymbol{n} \in \mathbb{R}^{16\times16}$ → Convolution → $\boldsymbol{m} \in \mathbb{R}^{8\times12\times12}$

$W \in \mathbb{R}^{1\times8\times5\times5}$

$=$



Building Blocks - Convolution



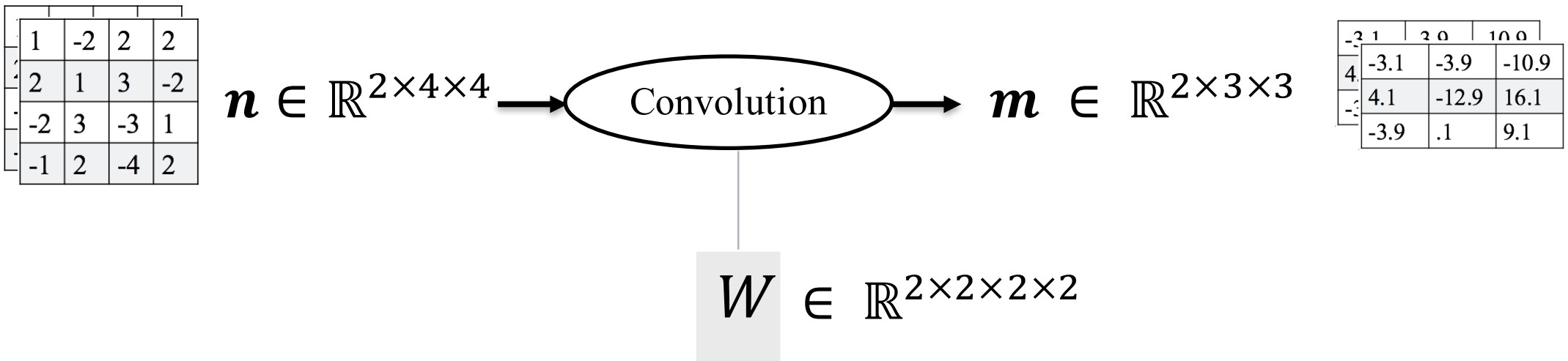
$$\boldsymbol{n} \in \mathbb{R}^{16\times16} \rightarrow \text{Convolution} \rightarrow \boldsymbol{m} \in \mathbb{R}^{8\times12\times12}$$

$$W$$

```
> conv = nn.SpatialConvolutionMM(1, 8, 5, 5)
> m = conv:forward(n)
> =n:size()
  1
 16
 16
[torch.LongStorage of size 3]

> =m:size()
  8
 12
 12
[torch.LongStorage of size 3]
```

```
> =conv.weight:size()
  8
 25
[torch.LongStorage of size 2]

> =conv.bias:size()
 8
[torch.LongStorage of size 1]
```

Building Blocks - Convolution



Building Blocks - Convolution

Image I = 2 x 4 x 4

I[1, :, :]

1	-2	2	2
2	1	3	-2
-2	3	-3	1
-1	2	-4	2

I[2, :, :]

3	0	0	0
-2	-2	1	-1
2	-1	3	1
5	-2	0	1

Weights W = 2 x 2 x 2 x 2
(nOutputPlane x nInputPlane x kH x kW)

W[1, 1, :, :]

1	-2
-2	1

W[2, 1, :, :]

3	1
2	2

W[1, 2, :, :]

1	0
0	1

W[2, 2, :, :]

0	0
0	4

0.1
0.2

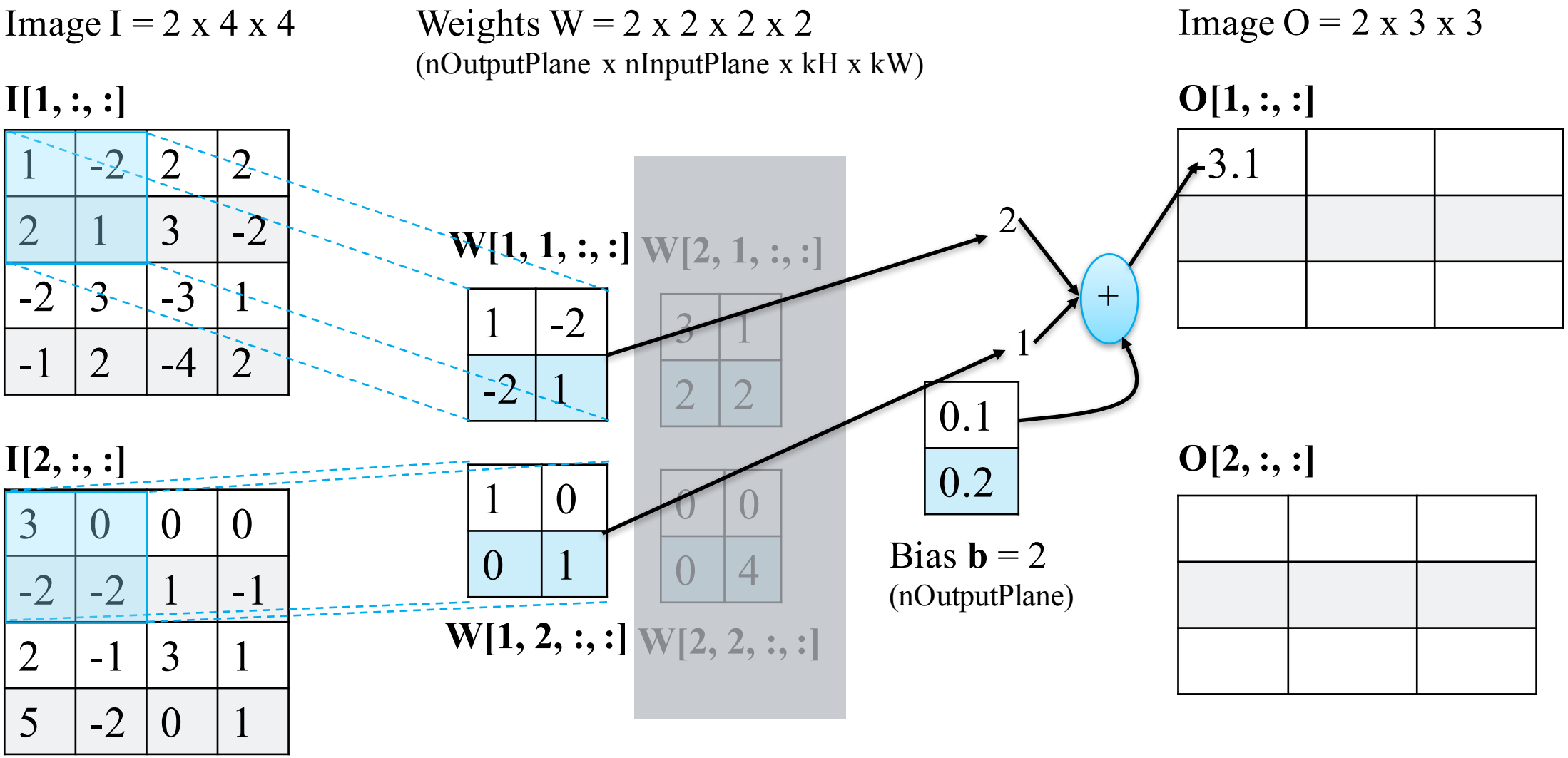
Bias **b** = 2
(nOutputPlane)

Image O = 2 x 3 x 3

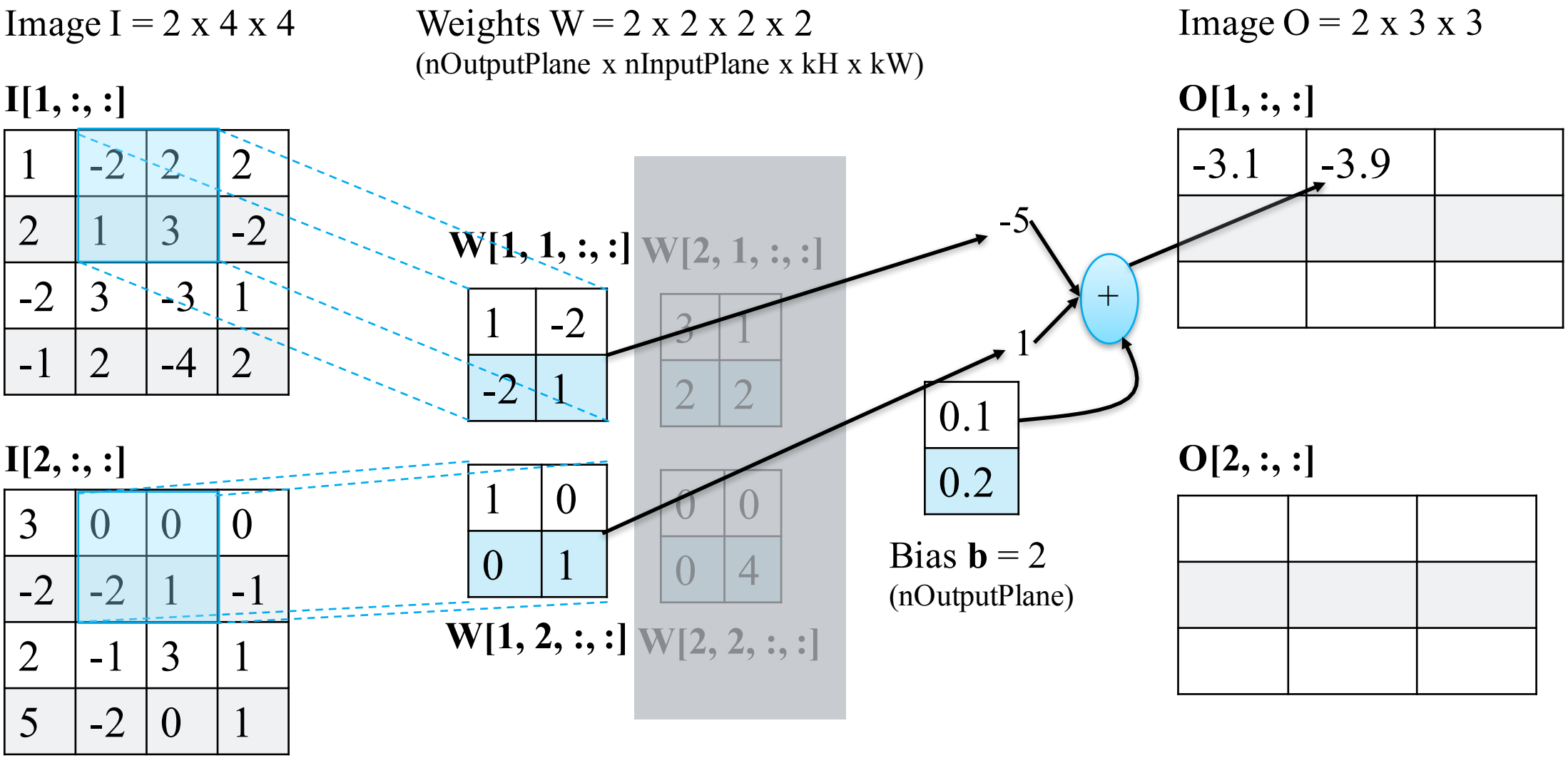
O[1, :, :]

O[2, :, :]

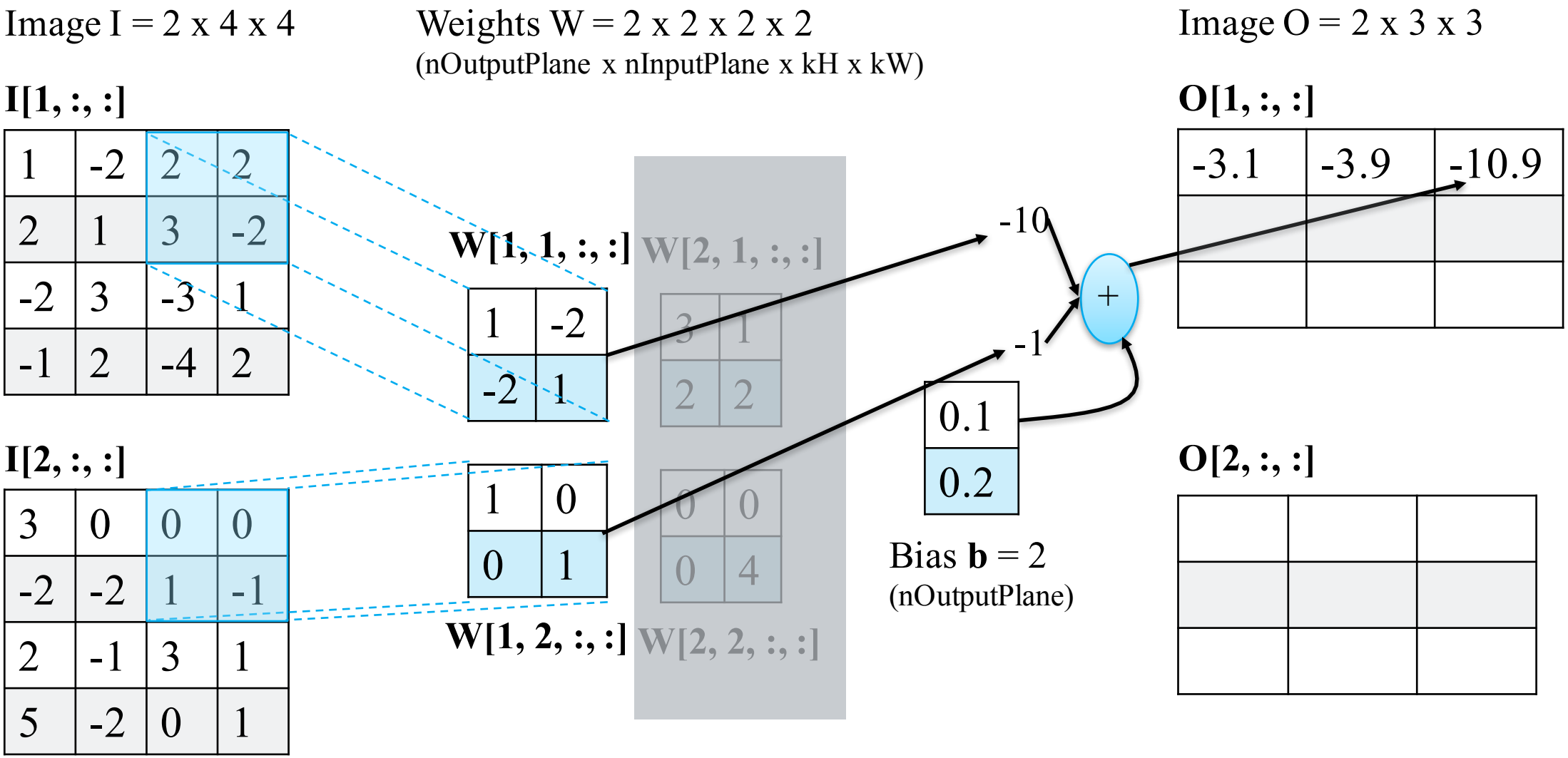
Building Blocks - Convolution



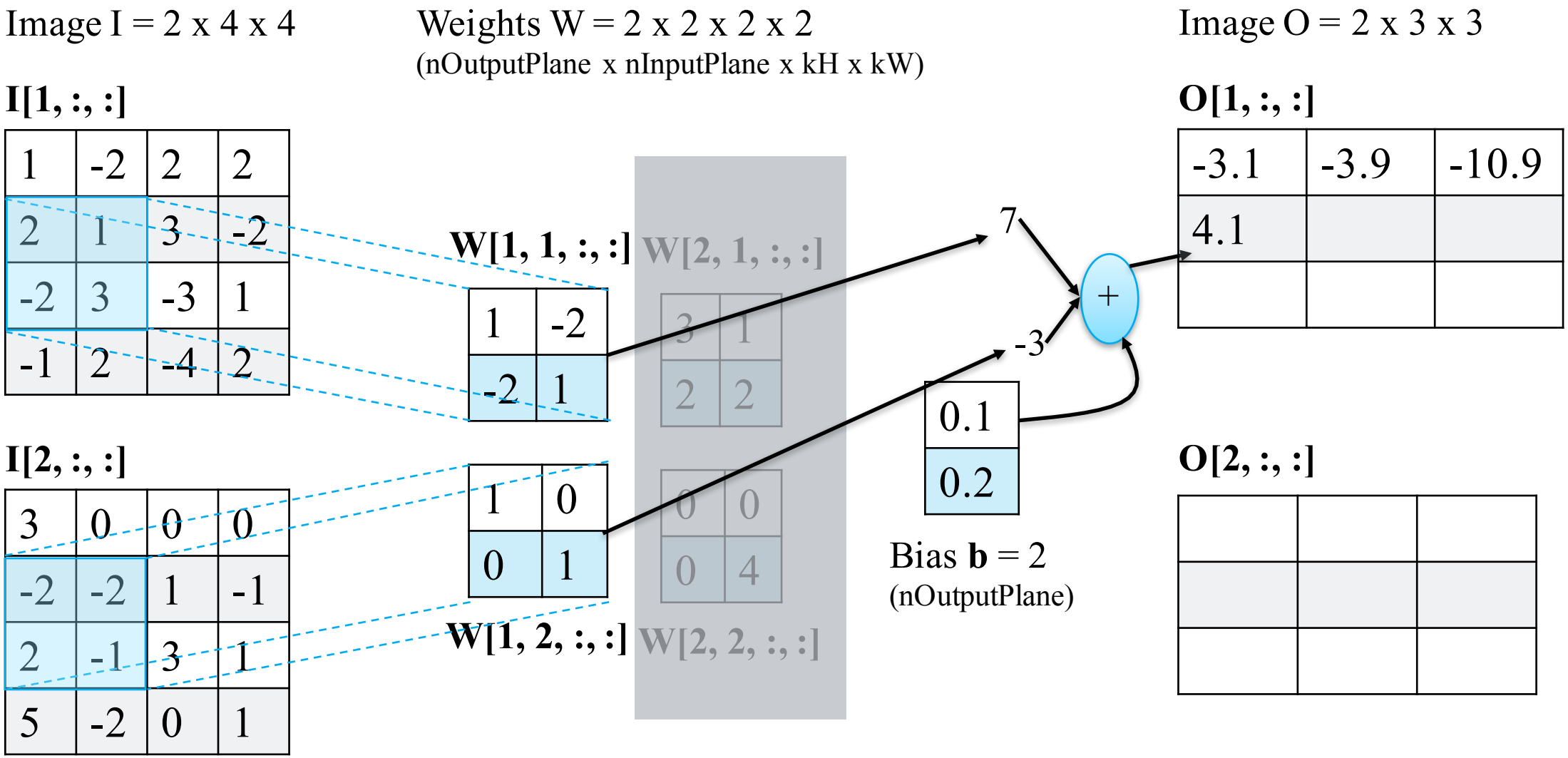
Building Blocks - Convolution



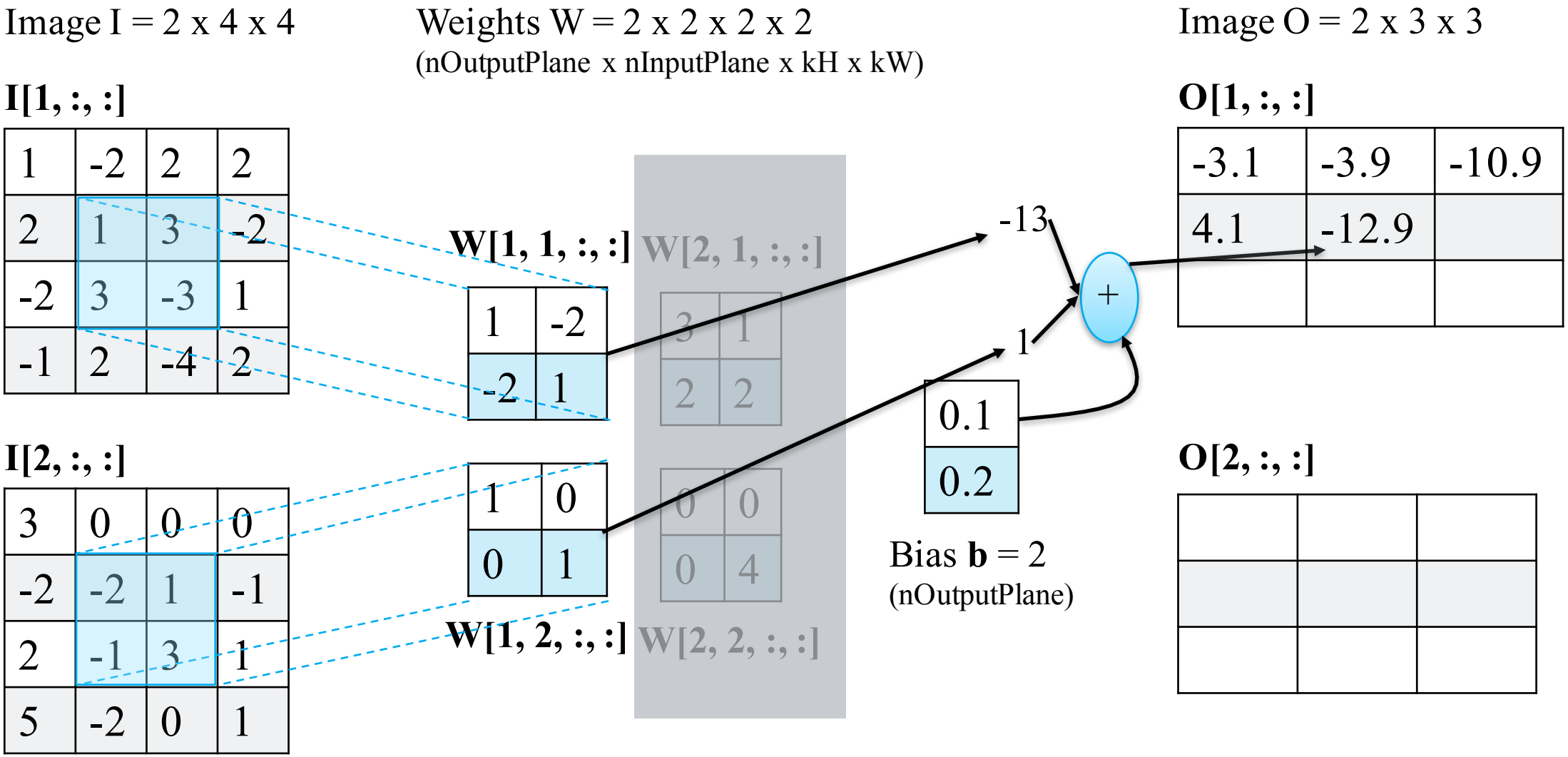
Building Blocks - Convolution



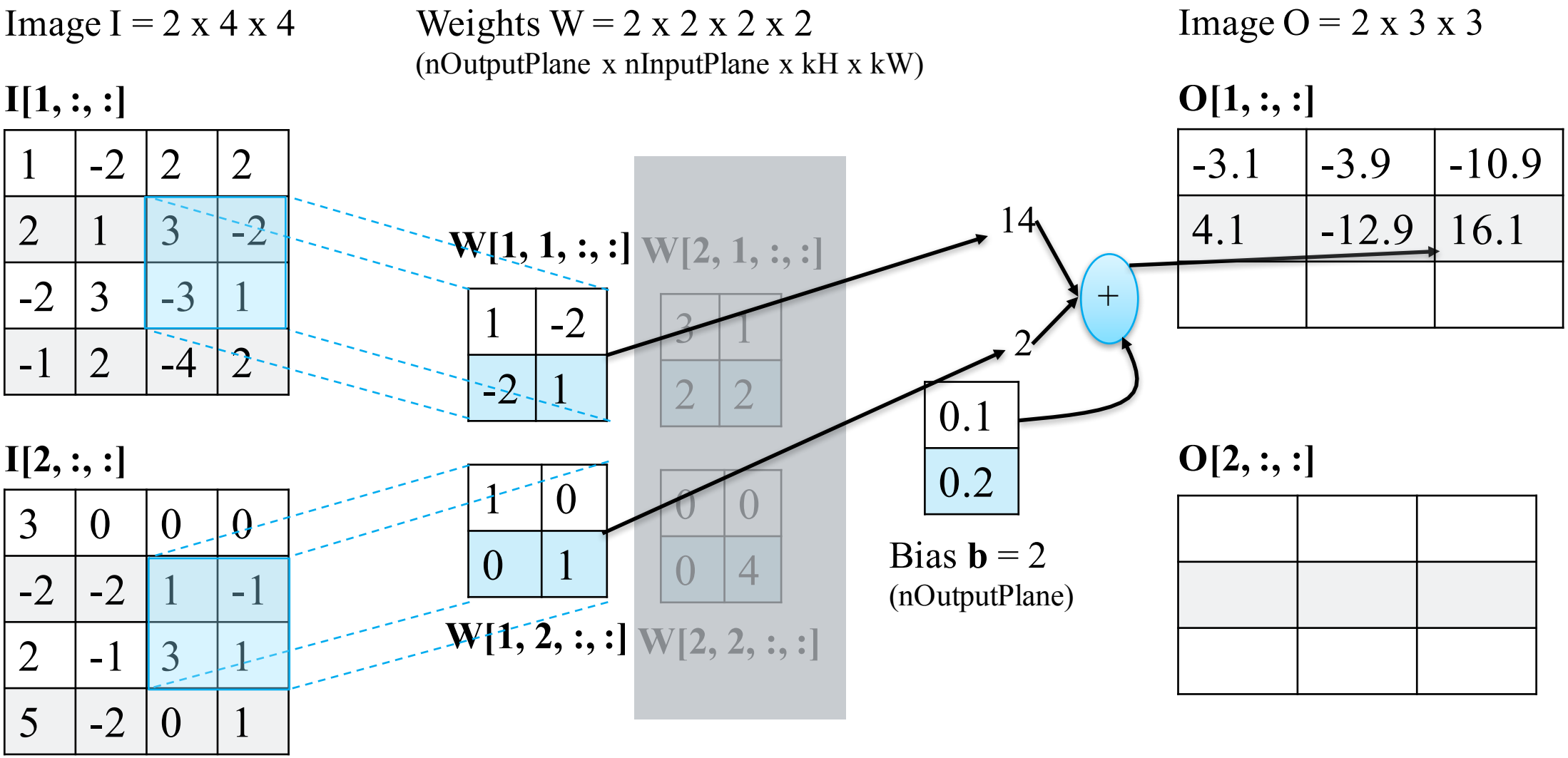
Building Blocks - Convolution



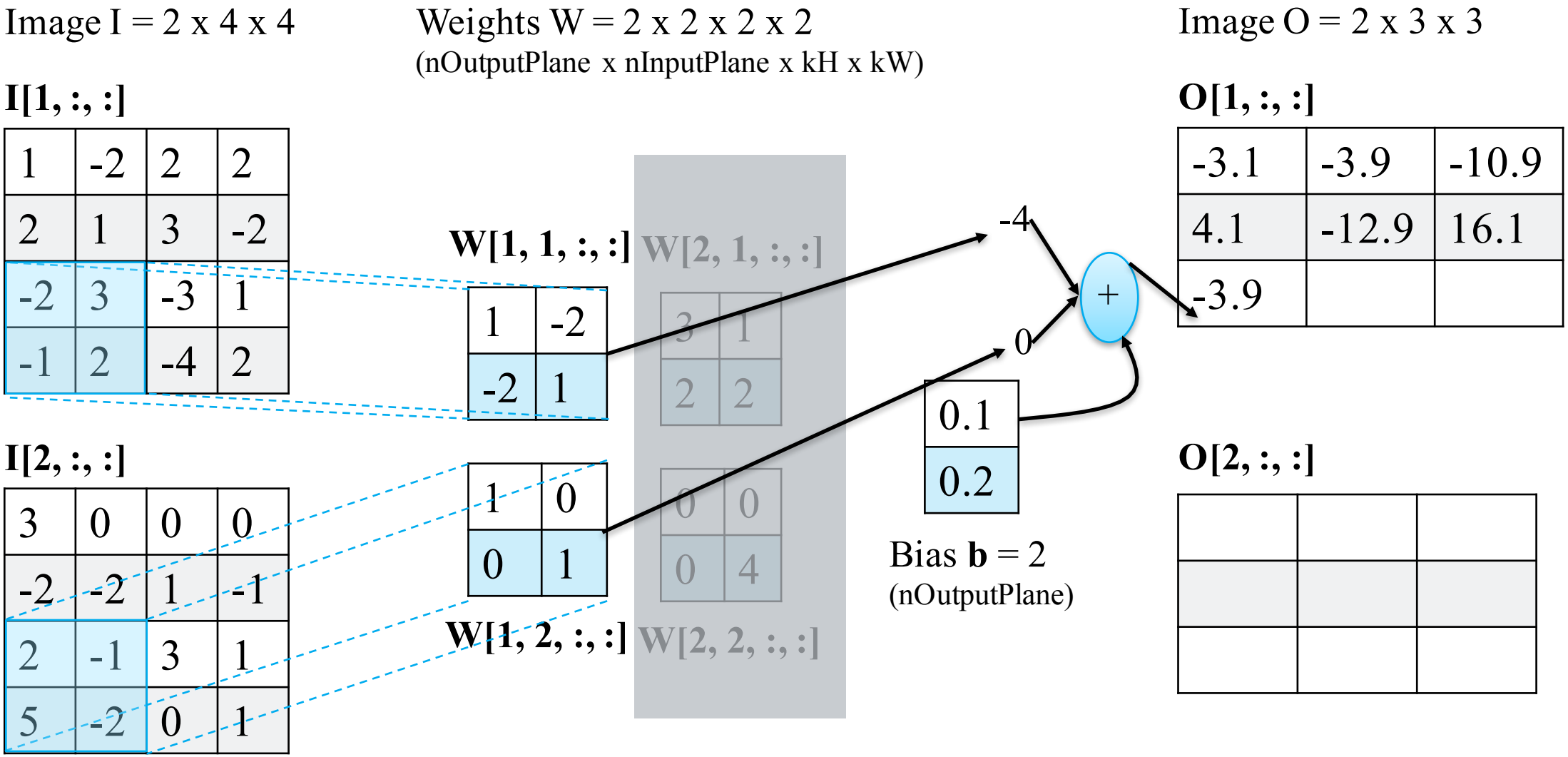
Building Blocks - Convolution



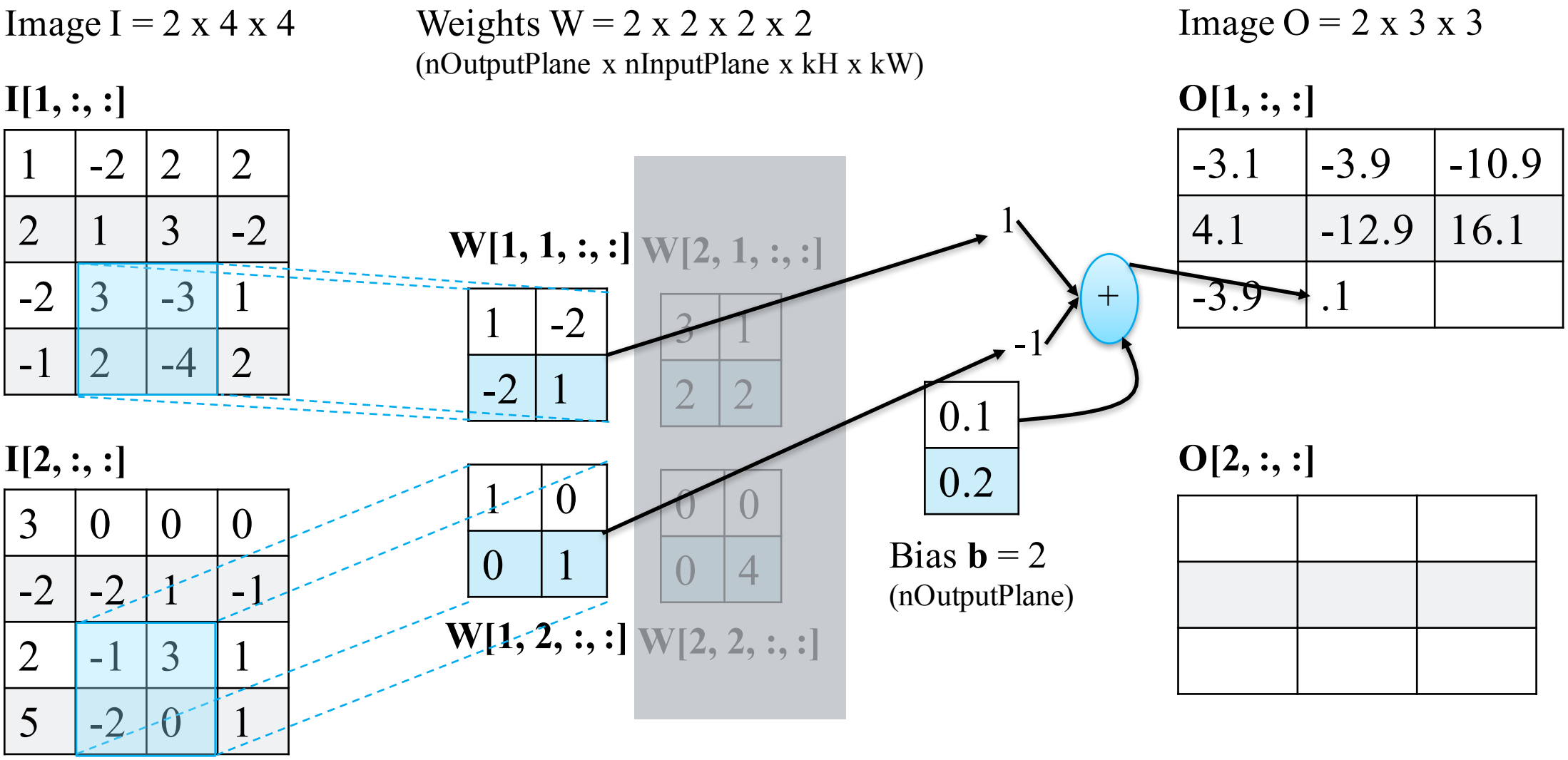
Building Blocks - Convolution



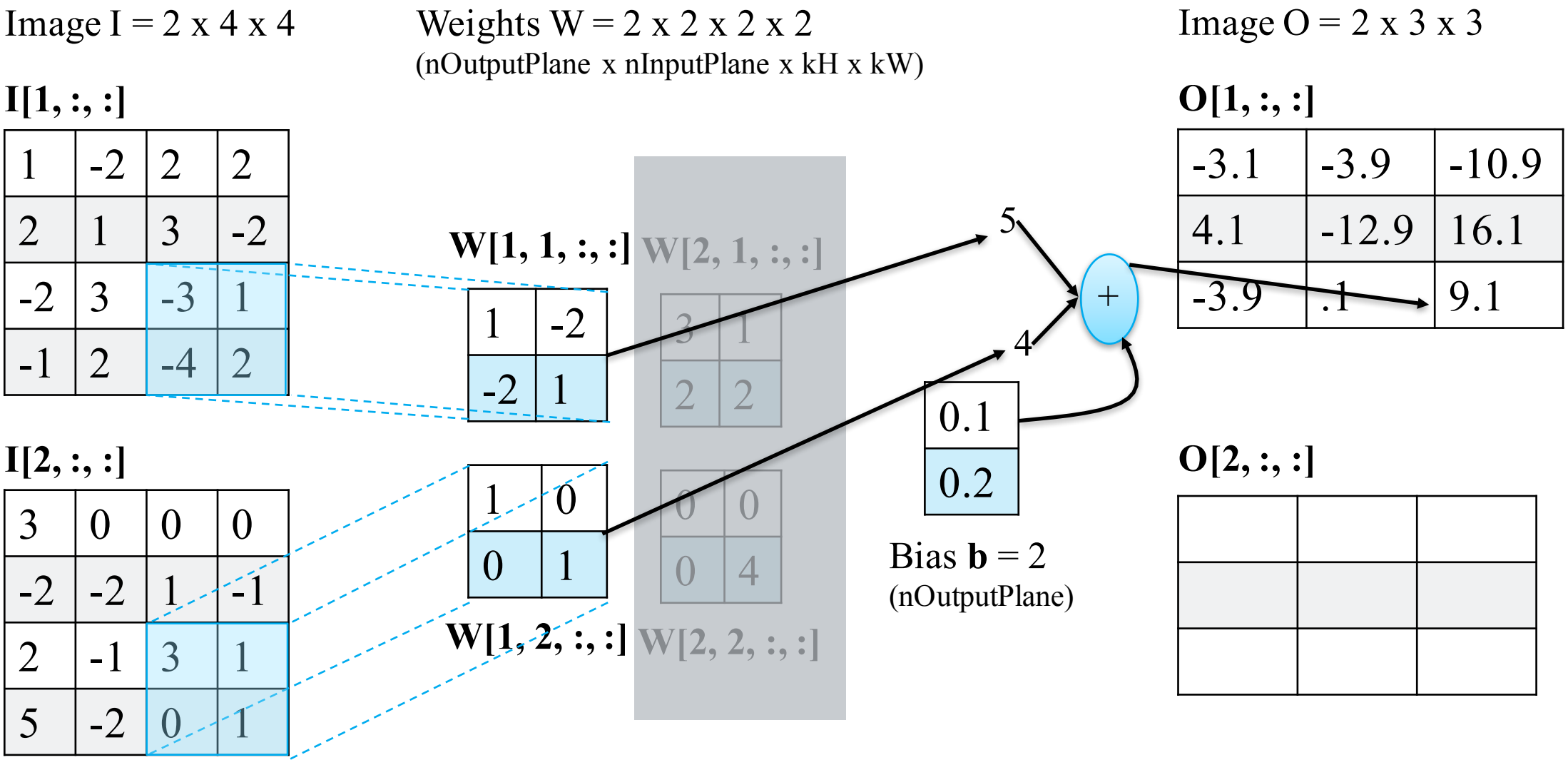
Building Blocks - Convolution



Building Blocks - Convolution



Building Blocks - Convolution



Building Blocks - Convolution

Image I = 2 x 4 x 4

I[1, :, :]

1	-2	2	2
2	1	3	-2
-2	3	-3	1
-1	2	-4	2

I[2, :, :]

3	0	0	0
-2	-2	1	-1
2	-1	3	1
5	-2	0	1

Weights W = 2 x 2 x 2 x 2
(nOutputPlane x nInputPlane x kH x kW)

W[1, 1, :, :]

1	-2
-2	1

W[2, 1, :, :]

3	1
2	2

W[1, 2, :, :]

1	0
0	1

W[2, 2, :, :]

0	0
0	4

7

-8

Bias **b** = 2
(nOutputPlane)

0.1
0.2

+

Image O = 2 x 3 x 3

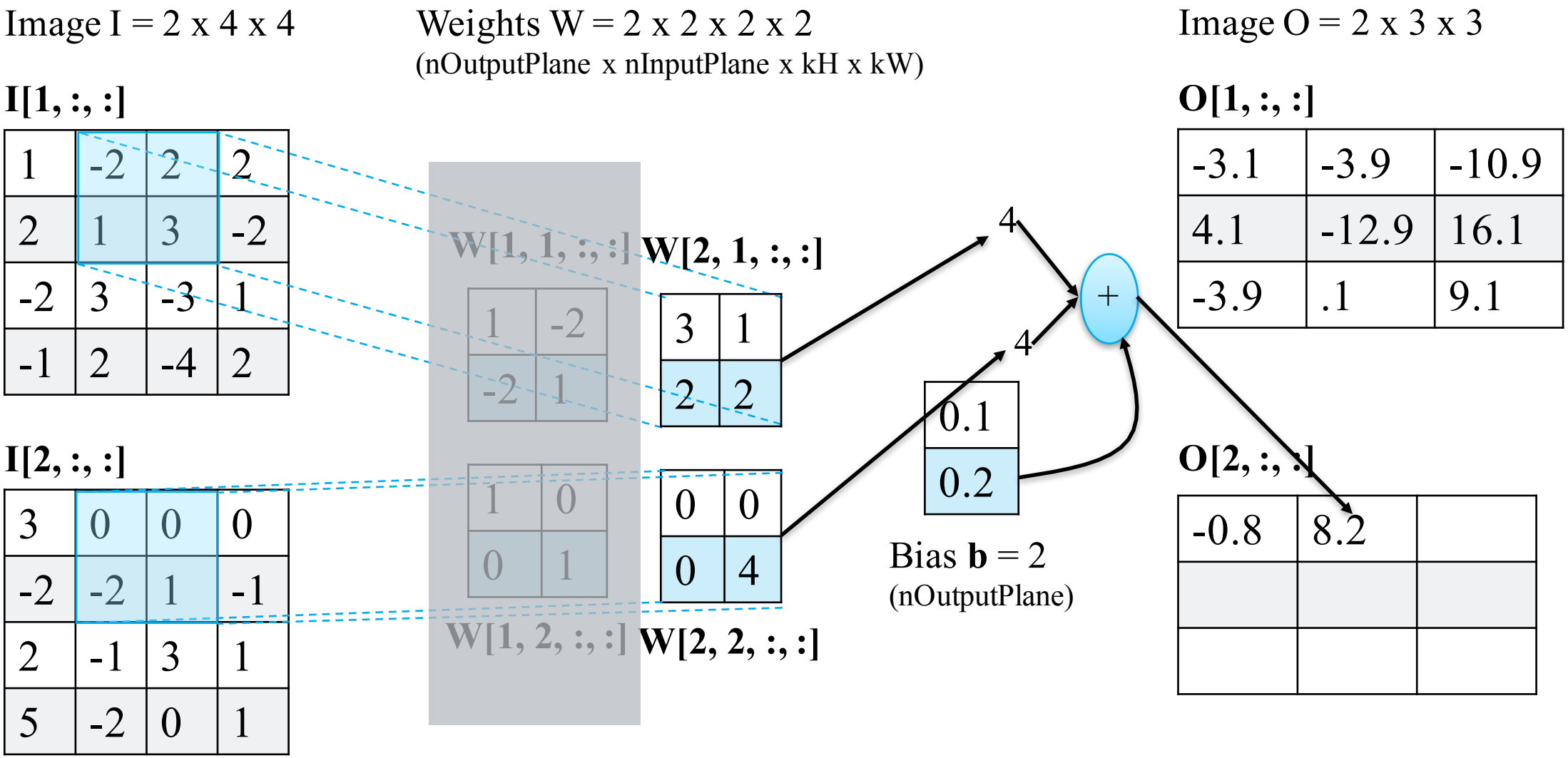
O[1, :, :]

-3.1	-3.9	-10.9
4.1	-12.9	16.1
-3.9	.1	9.1

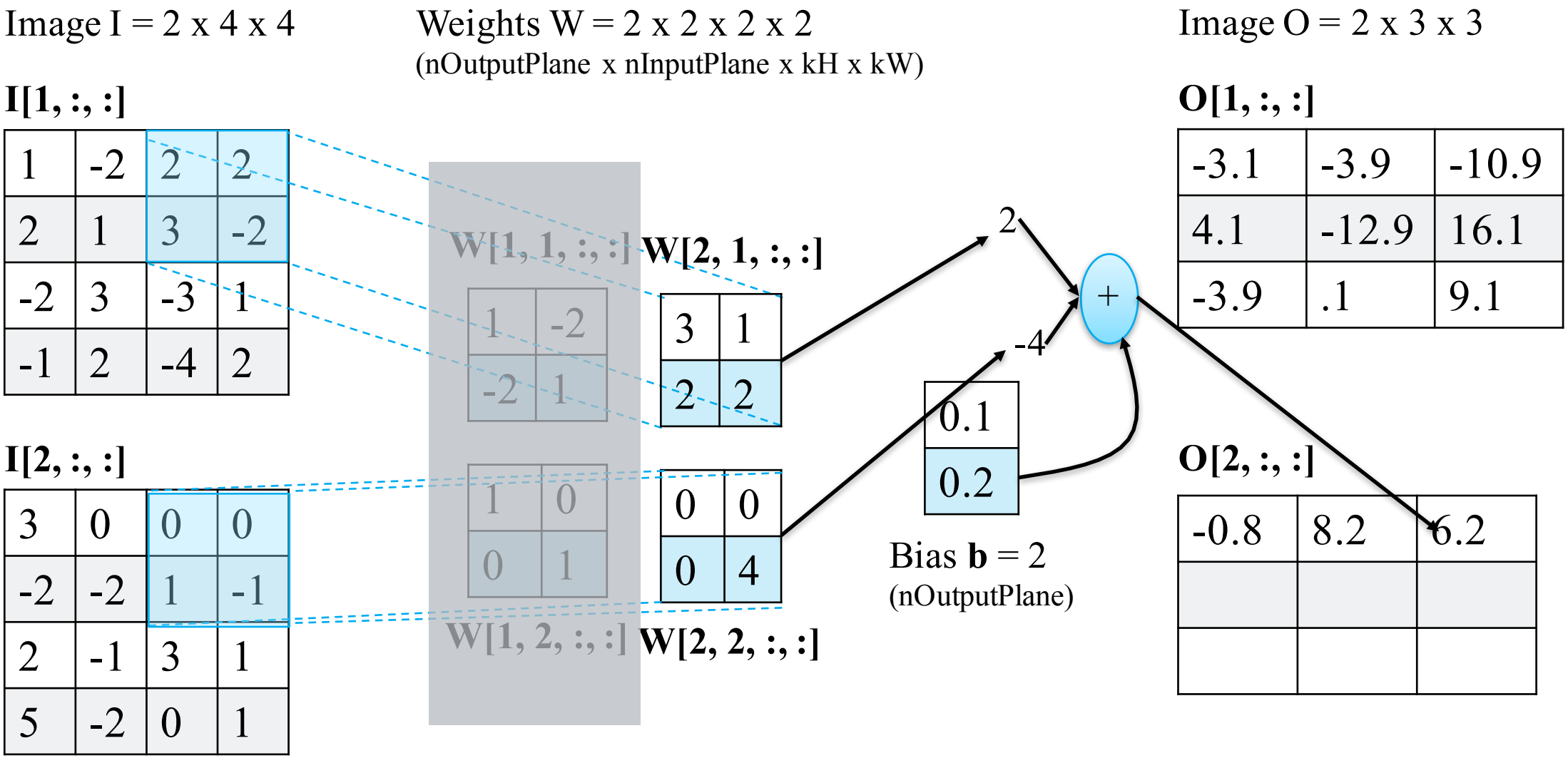
O[2, :, :]

-0.8		

Building Blocks - Convolution



Building Blocks - Convolution



Building Blocks - Convolution

Image I = 2 x 4 x 4

I[1, :, :]

1	-2	2	2
2	1	3	-2
-2	3	-3	1
-1	2	-4	2

I[2, :, :]

3	0	0	0
-2	-2	1	-1
2	-1	3	1
5	-2	0	1

Weights W = 2 x 2 x 2 x 2
(nOutputPlane x nInputPlane x kH x kW)

W[1, 1, :, :]

1	-2
-2	1

W[1, 2, :, :]

1	0
0	1

W[2, 1, :, :]

3	1
2	2

W[2, 2, :, :]

0	0
0	4

9

-4

Bias **b** = 2
(nOutputPlane)

0.1
0.2

+

Image O = 2 x 3 x 3

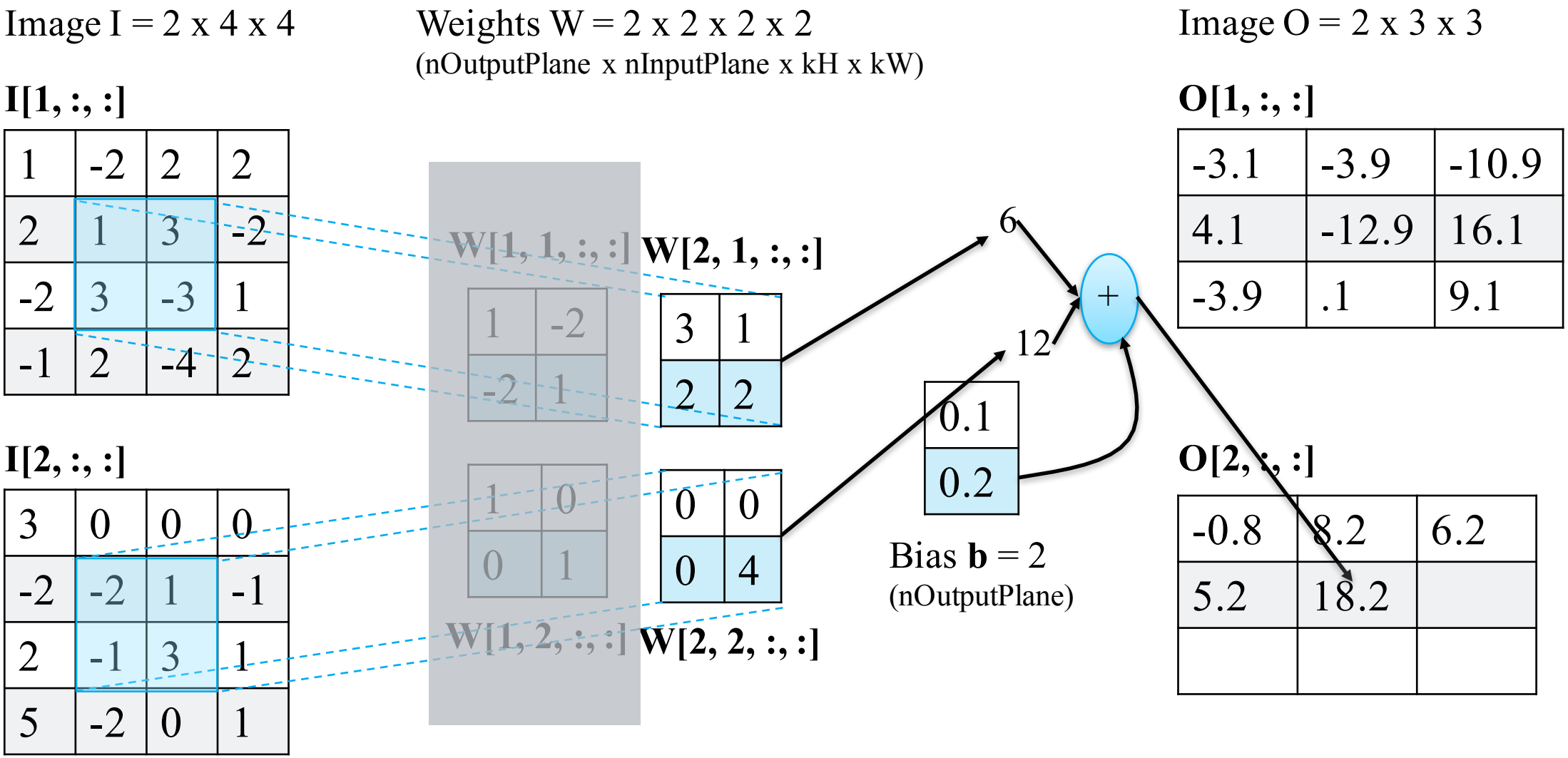
O[1, :, :]

-3.1	-3.9	-10.9
4.1	-12.9	16.1
-3.9	.1	9.1

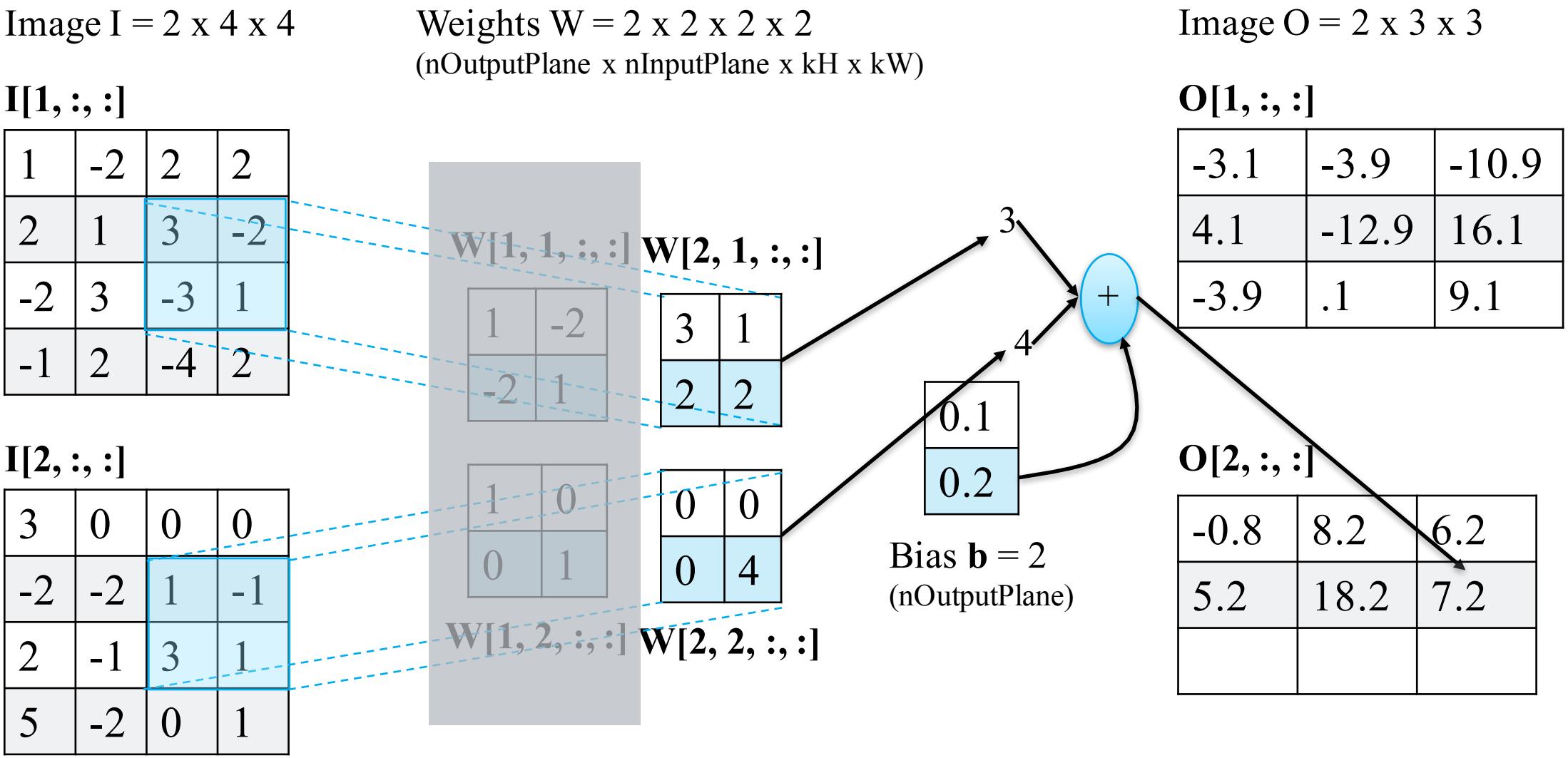
O[2, :, :]

-0.8	8.2	6.2
5.2		

Building Blocks - Convolution



Building Blocks - Convolution



Building Blocks - Convolution

Image I = 2 x 4 x 4

I[1, :, :]

1	-2	2	2
2	1	3	-2
-2	3	-3	1
-1	2	-4	2

I[2, :, :]

3	0	0	0
-2	-2	1	-1
2	-1	3	1
5	-2	0	1

Weights W = 2 x 2 x 2 x 2
(nOutputPlane x nInputPlane x kH x kW)

W[1, 1, :, :]

1	-2
-2	1

W[2, 1, :, :]

3	1
2	2

W[1, 2, :, :]

1	0
0	1

W[2, 2, :, :]

0	0
0	4

-1

-8

+

0.1
0.2

Bias **b** = 2
(nOutputPlane)

Image O = 2 x 3 x 3

O[1, :, :]

-3.1	-3.9	-10.9
4.1	-12.9	16.1
-3.9	.1	9.1

O[2, :, :]

-0.8	8.2	6.2
5.2	18.2	7.2
-8.8		

Building Blocks - Convolution

Image I = 2 x 4 x 4

I[1, :, :]

1	-2	2	2
2	1	3	-2
-2	3	-3	1
-1	2	-4	2

I[2, :, :]

3	0	0	0
-2	-2	1	-1
2	-1	3	1
5	-2	0	1

Weights W = 2 x 2 x 2 x 2
(nOutputPlane x nInputPlane x kH x kW)

W[1, 1, :, :]

1	-2
-2	1

W[2, 1, :, :]

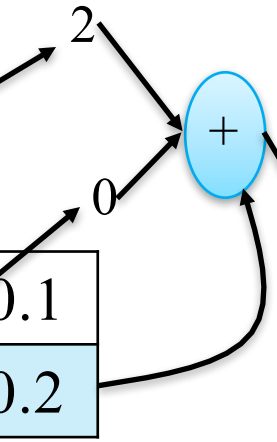
3	1
2	2

W[1, 2, :, :]

1	0
0	1

W[2, 2, :, :]

0	0
0	4



Bias **b** = 2
(nOutputPlane)

Image O = 2 x 3 x 3

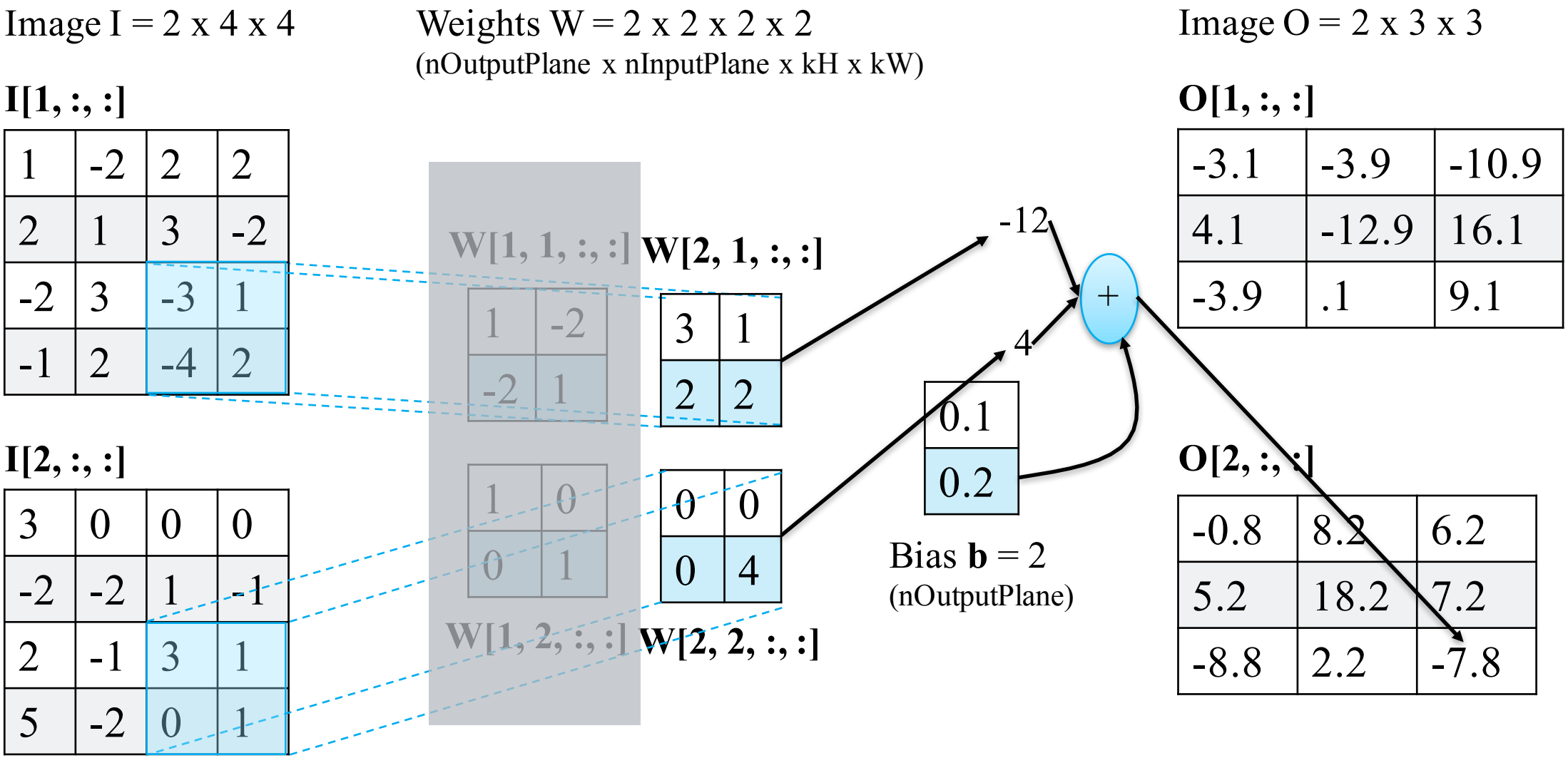
O[1, :, :]

-3.1	-3.9	-10.9
4.1	-12.9	16.1
-3.9	.1	9.1

O[2, :, :]

-0.8	8.2	6.2
5.2	18.2	7.2
-8.8	2.2	

Building Blocks - Convolution



Building Blocks – Convolution – in Torch7

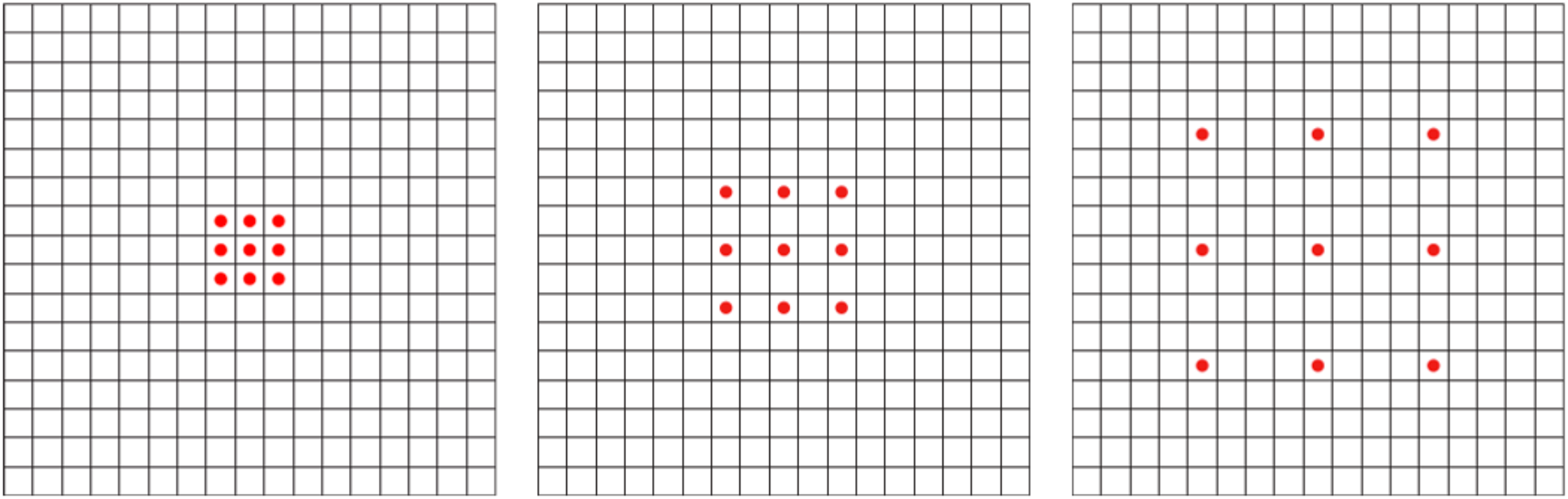
```
> I = torch.DoubleTensor({{{1,-2,2,2},{2,1,3,-2},{-2,3,-3,1},{-1,2,-4,2}}, {{3,0,0,0},{-2,-2,1,-1},{2,-1,3,1},{5,-2,0,1}}})
> conv = nn.SpatialConvolutionMM(2,2,2,2)
> conv.bias = torch.DoubleTensor({0.1, 0.2})
> conv.weight = torch.DoubleTensor({{1,-2,-2,1,1,0,0,1},{3,1,2,2,0,0,0,4}})
> O = conv:forward(I)
> =I
(1,.,.) =
  1 -2  2  2
  2  1  3 -2
 -2  3 -3  1
 -1  2 -4  2

(2,.,.) =
  3  0  0  0
 -2 -2  1 -1
  2 -1  3  1
  5 -2  0  1
[torch.DoubleTensor of size 2x4x4]

> =O
(1,.,.) =
   3.1000  -3.9000 -10.9000
   4.1000 -12.9000  16.1000
  -3.9000   0.1000   9.1000

(2,.,.) =
  -0.8000   8.2000   6.2000
   5.2000  18.2000   7.2000
  -8.8000   2.2000  -7.8000
[torch.DoubleTensor of size 2x3x3]
```

Building Blocks – Convolution – in Torch7

```
4  function SpatialConvolutionMM:__init(nInputPlane, nOutputPlane, kW, kH, dW, dH, padW, padH)
5     parent.__init(self)
6
7     dW = dW or 1
8     dH = dH or 1
9
10    self.nInputPlane = nInputPlane
11    self.nOutputPlane = nOutputPlane
12    self.kW = kW
13    self.kH = kH
14
15    self.dW = dW
16    self.dH = dH
17    self.padW = padW or 0
18    self.padH = padH or self.padW
19
20    self.weight = torch.Tensor(nOutputPlane, nInputPlane*kH*kW)
21    self.bias = torch.Tensor(nOutputPlane)
22    self.gradWeight = torch.Tensor(nOutputPlane, nInputPlane*kH*kW)
23    self.gradBias = torch.Tensor(nOutputPlane)
24
25    self:reset()
26 end
```

https://github.com/torch/nn/blob/master/SpatialConvolutionMM.lua

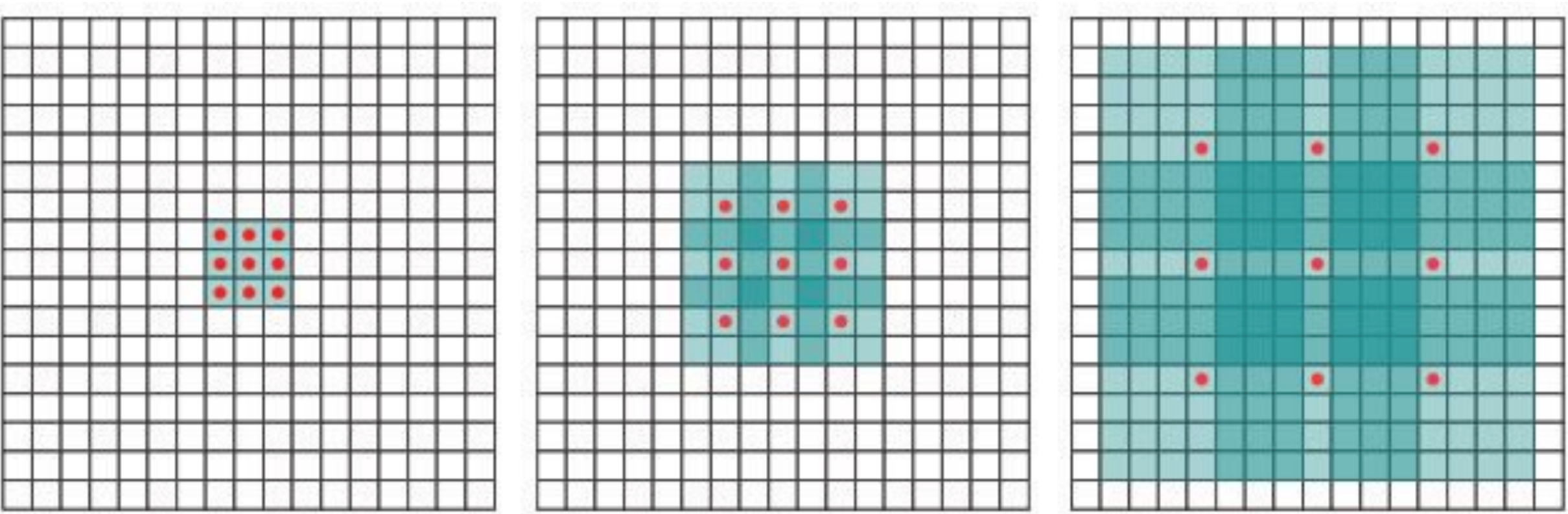
Aside: Dilated Convolution



(a) (b) (c)

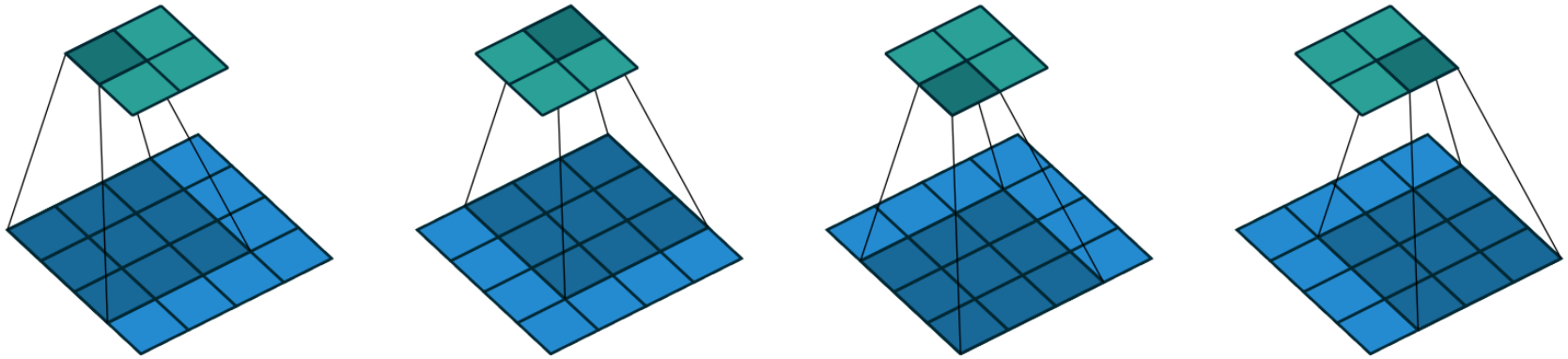
Aside: Dilated Convolution

nn.SpatialDilatedConvolution



Multi-Scale Context Aggregation by Dilated Convolutions
Fisher Yu, Vladlen Koltun

Building Blocks – Convolution



(No padding, unit strides) Convolving a 3 × 3 kernel over a 4 × 4 input using unit strides (i.e., i = 4, k = 3, s = 1 and p = 0).

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix} \begin{bmatrix} I_0 \\ I_1 \\ \vdots \\ I_{15} \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{I}} = W^T \cdot LO = Deconvolution$$

Building Blocks – Convolution

(Normal) Convolution Forward & Backward Pass

Layer with kernel size 3×3 and stride of $2, 2$

```
conv = nn.SpatialConvolutionMM(1,1,3,3,2,2)
conv.weight:uniform(0,2)
conv.bias:fill(0)
print(conv.weight)
```

```
 1.0160  0.4442  0.8234  1.7105  1.9389  1.8970  0.4721  1.4563  0.4023
[torch.DoubleTensor of size 1x9]
```

Input

Image of channel 1 and size 5×5

```
imgC = torch.Tensor(1,1,5,5)
imgC:uniform(0,5)
```

```
r=conv:forward(imgC)
print(r)
```

```
(1,1,.,.) =
  28.0467  21.1382
  23.1202  30.5335
[torch.DoubleTensor of size 1x1x2x2]
```

```
conv:backward(imgC,r)
```

```
(1,1,.,.) =
  28.4956  12.4596  44.5710   9.3905  17.4058
  47.9750  54.3784  89.3627  40.9838  40.0994
  36.7311  51.1151  71.3240  44.3476  33.6471
  39.5481  44.8267  96.0881  59.1999  57.9224
  10.9150  33.6697  23.7172  44.4655  12.2851
[torch.DoubleTensor of size 1x1x5x5]
```

Building Blocks – Convolution

(Normal) Convolution Forward & Backward Pass

Layer with kernel size 3×3 and stride of $2, 2$

```
conv = nn.SpatialConvolutionMM(1,1,3,3,2,2)
conv.weight:uniform(0,2)
conv.bias:fill(0)
print(conv.weight)
```

```
 1.0160   0.4442   0.8234   1.7105   1.9389   1.8970   0.4721   1.456
[torch.DoubleTensor of size 1x9]
```

Transpose Convolution Forward Pass

Layer with kernel size 3×3 and stride of $2, 2$

```
trancon = nn.SpatialFullConvolution(1, 1, 3, 3, 2, 2)
trancon.bias:fill(0)
```

```
trancon.weight:copy(conv.weight)
trancon.weight:reshape(1,1,3,3)
print(trancon.weight)
```

```
(1,1,.,.) =
  1.0160  0.4442  0.8234
  1.7105  1.9389  1.8970
  0.4721  1.4563  0.4023
[torch.DoubleTensor of size 1x1x3x3]
```

Input

Image of channel 1 and size 2×2

```
print(r)
```

```
(1,1,.,.) =
  28.0467  21.1382
  23.1202  30.5335
[torch.DoubleTensor of size 1x1x2x2]
```

Forward Pass - Note this is exactly same as the backward pass of (normal) convolution!

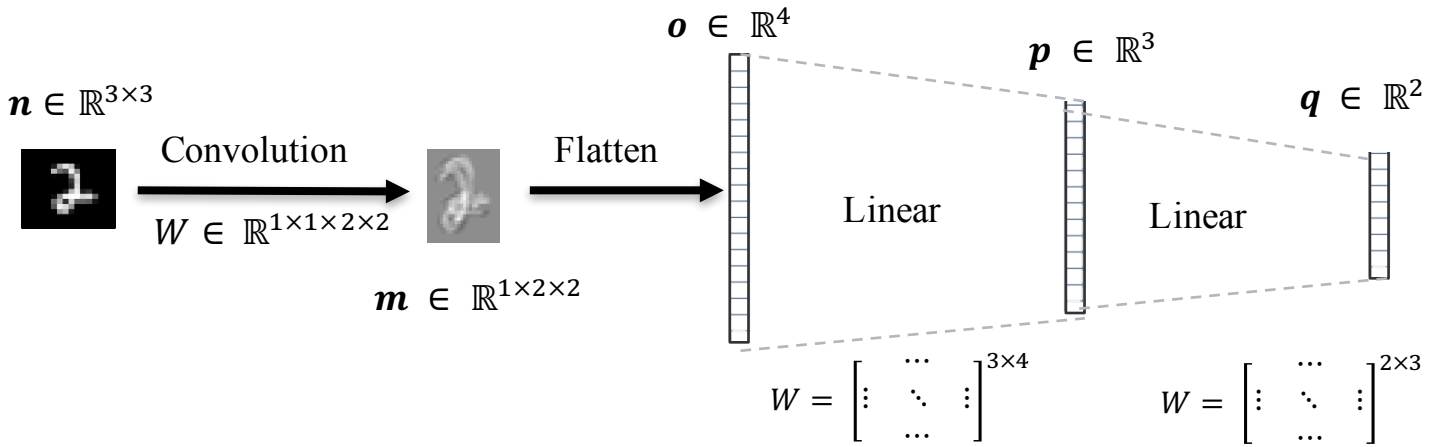
```
trancon:forward(r)
```

```
(1,1,.,.) =
  28.4956  12.4596  44.5710   9.3905  17.4058
  47.9750  54.3784  89.3627  40.9838  40.0994
  36.7311  51.1151  71.3240  44.3476  33.6471
  39.5481  44.8267  96.0881  59.1999  57.9224
  10.9150  33.6697  23.7172  44.4655  12.2851
[torch.DoubleTensor of size 1x1x5x5]
```

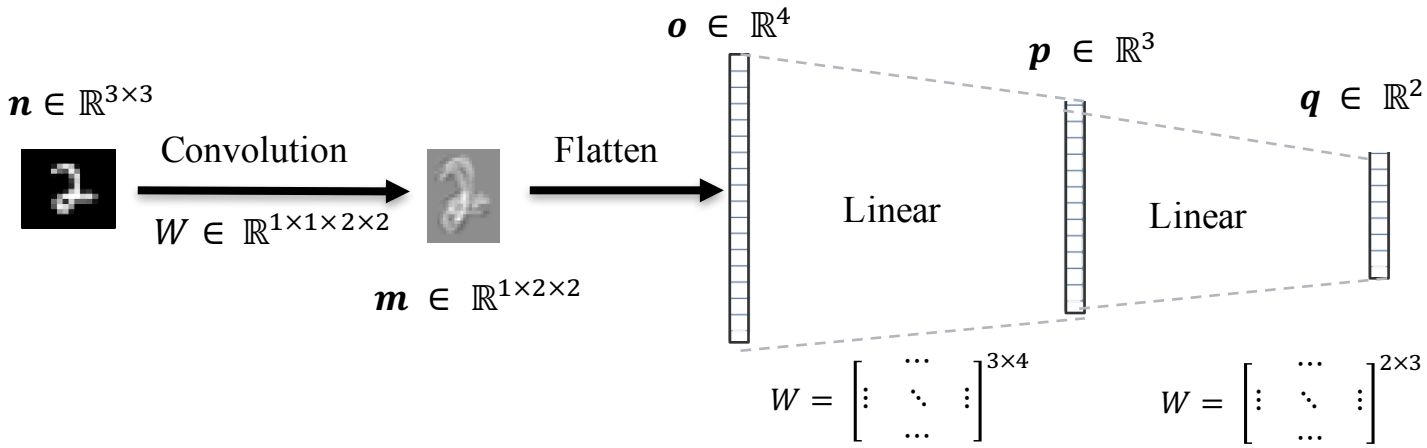
```
conv:backward(imgC,r)
```

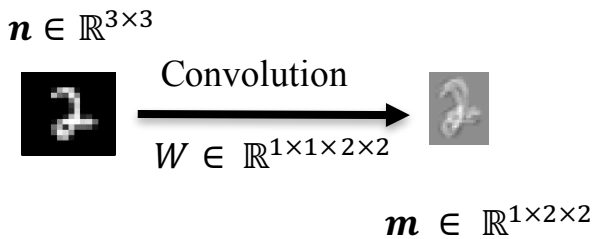
```
(1,1,.,.) =
  28.4956  12.4596  44.5710   9.3905  17.4058
  47.9750  54.3784  89.3627  40.9838  40.0994
  36.7311  51.1151  71.3240  44.3476  33.6471
  39.5481  44.8267  96.0881  59.1999  57.9224
  10.9150  33.6697  23.7172  44.4655  12.2851
[torch.DoubleTensor of size 1x1x5x5]
```

Everything is a Convolution!

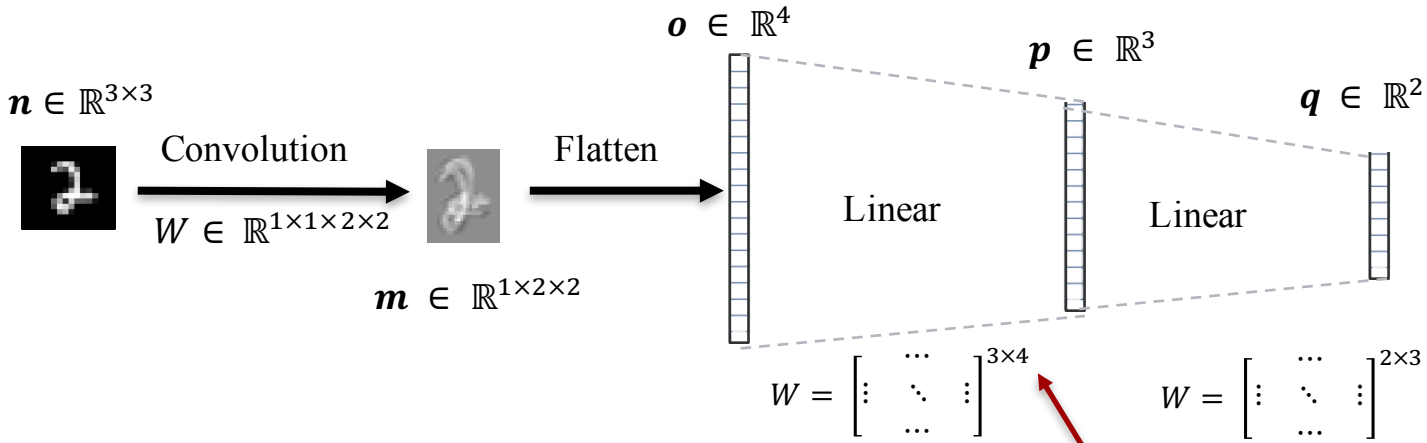


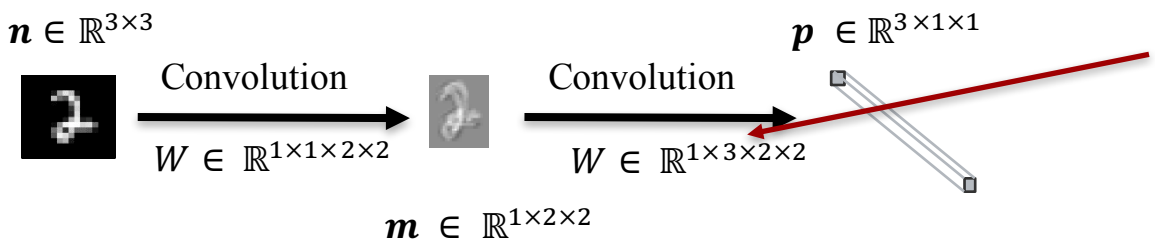
Everything is a Convolution!





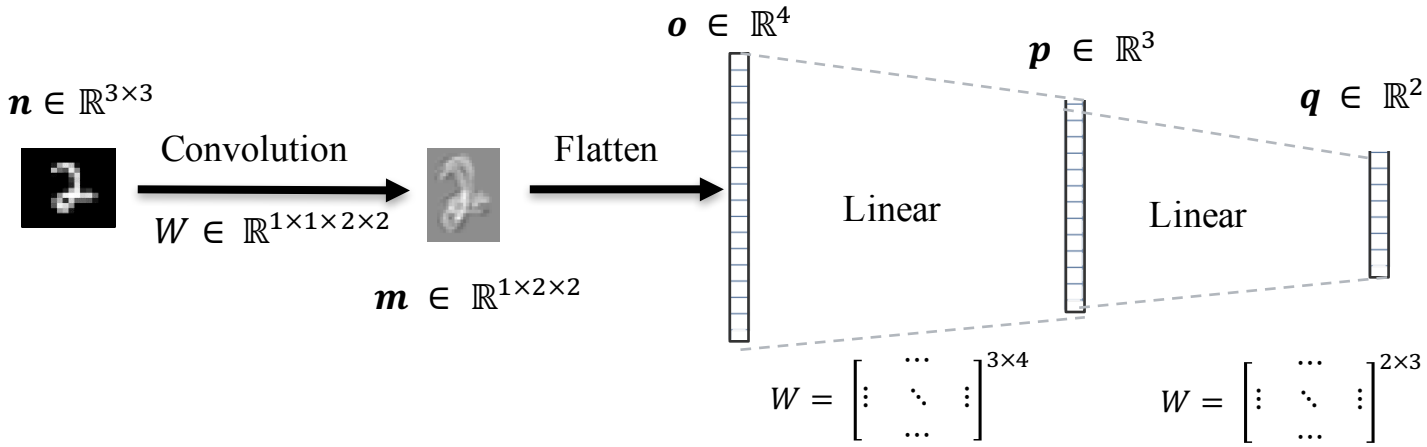
Everything is a Convolution!

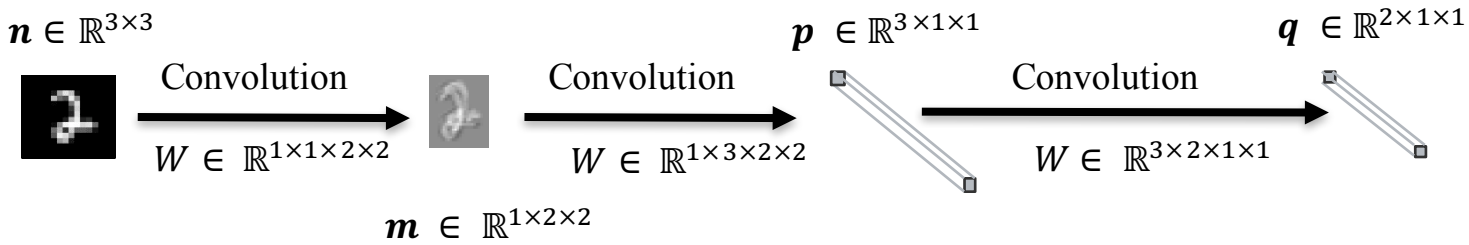




Each output channel (W[:, I, :, :]) corresponds to a row I of the matrix

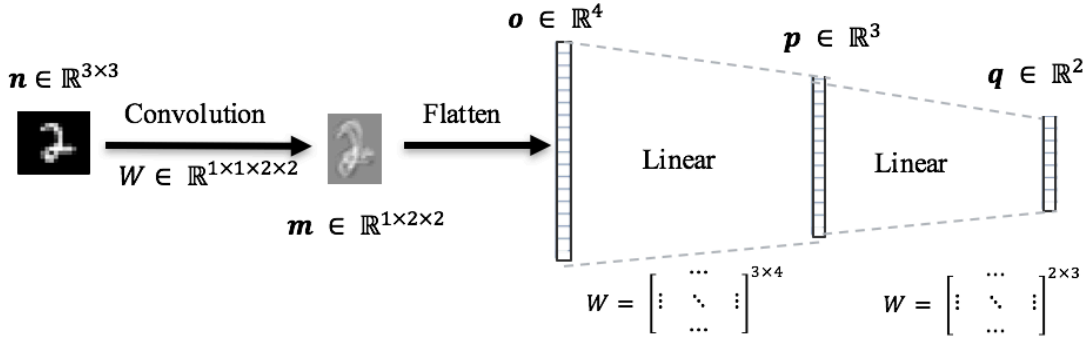
Everything is a Convolution!

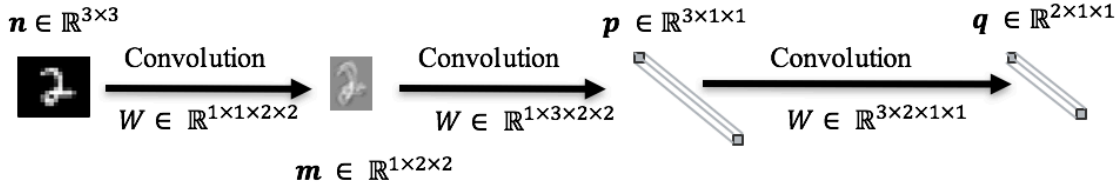




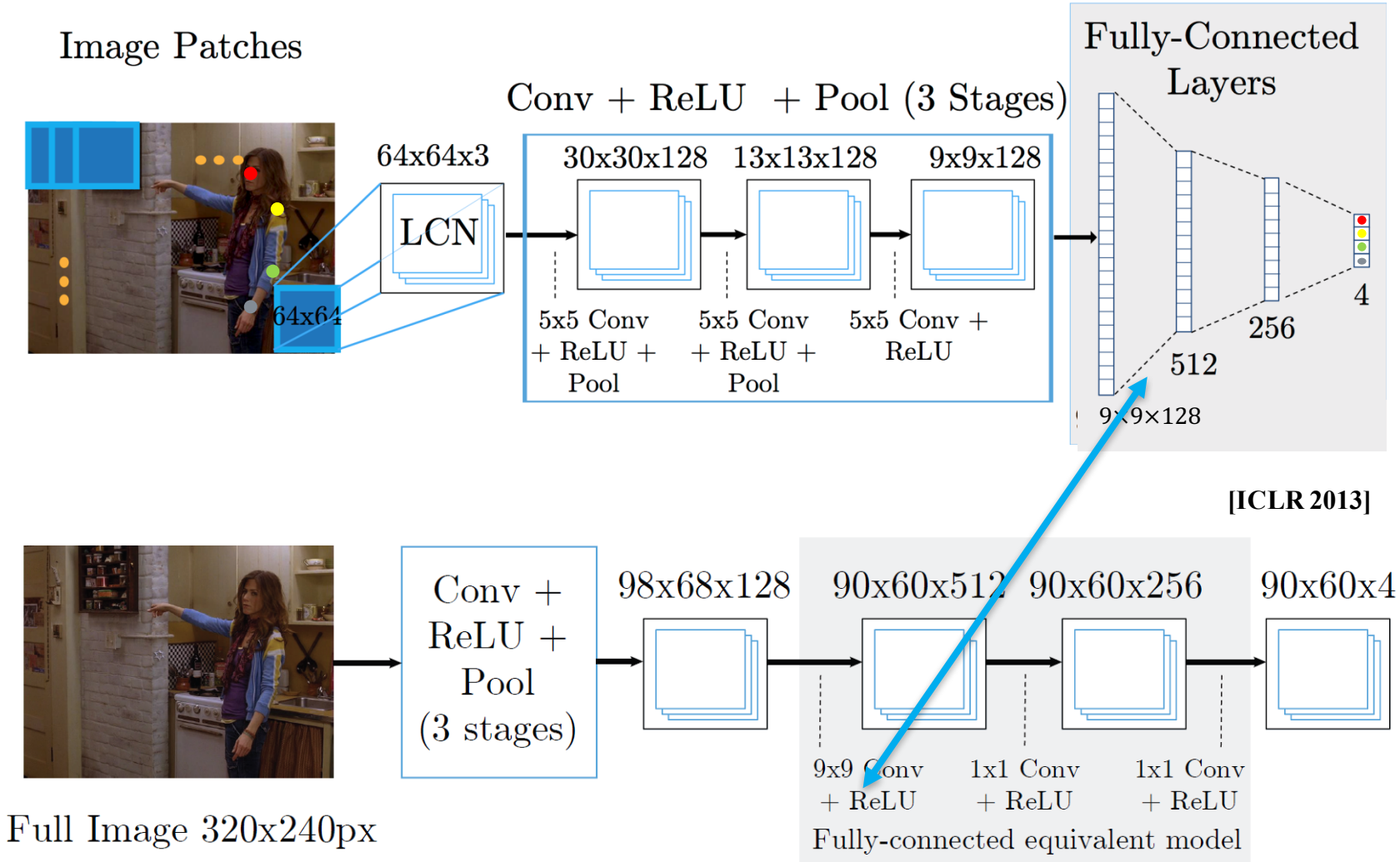
TL;DR: Converting Linear to Conv:

- Do not *flatten*, instead:
- Do a convolution with height and width of the kernel as the size of the features just before the flatten
- Number of input channels as the number of input channels in the features just before the flatten
- Number of output channels as the number of neurons after the linear layer!



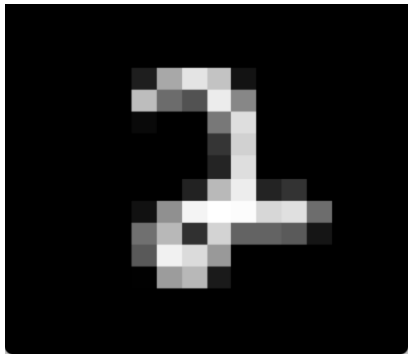


Sliding-window as Convolution

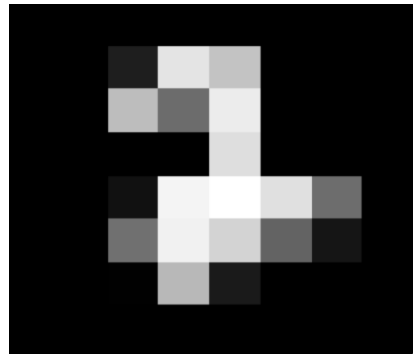


Building Blocks:
Max Pooling

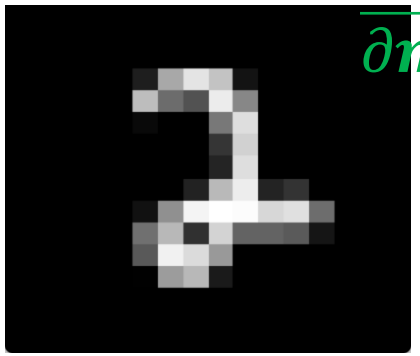
Building Blocks – Pooling (Max Pooling)



$\boldsymbol{n} \in \mathbb{R}^{4\times4}$ → Max Pooling → $\boldsymbol{m} \in \mathbb{R}^{2\times2}$



Building Blocks – Pooling (Max Pooling) – in Torch7

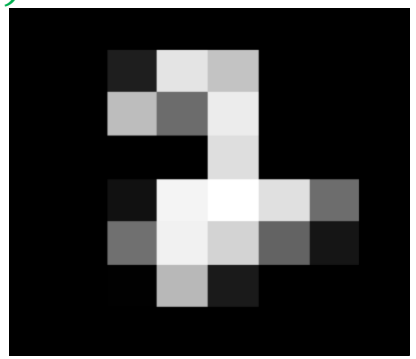


$\frac{\partial L}{\partial m} \cdot \frac{\partial m}{\partial n} \in \mathbb{R}^{1\times(4\times4)}$

$\frac{\partial m}{\partial n} \in \mathbb{R}^{(2\times2)\times(4\times4)}$

$\frac{\partial L}{\partial m} \in \mathbb{R}^{1\times(2\times2)}$

$n \in \mathbb{R}^{4\times4}$ → Max Pooling → $m \in \mathbb{R}^{2\times2}$



```
> n = torch.rand(1,4,4)
> pool = nn.SpatialMaxPooling(2, 2)
> m = pool:forward(n)
> =n
(1,.,.) =
  0.2692  0.4190  0.2095  0.9163
  0.2778  0.9199  0.5555  0.1638
  0.6936  0.2328  0.0553  0.1798
  0.3611  0.3225  0.9032  0.5106
[torch.DoubleTensor of size 1x4x4]

> =m
(1,.,.) =
  0.9199  0.9163
  0.6936  0.9032
[torch.DoubleTensor of size 1x2x2]
```

```
> nextgrad = torch.ones(1,2,2)
> pool:backward(n, nextgrad)
> =pool.gradInput
(1,.,.) =
  0  0  0  1
  0  1  0  0
  1  0  0  0
  0  0  1  0
[torch.DoubleTensor of size 1x4x4]
```

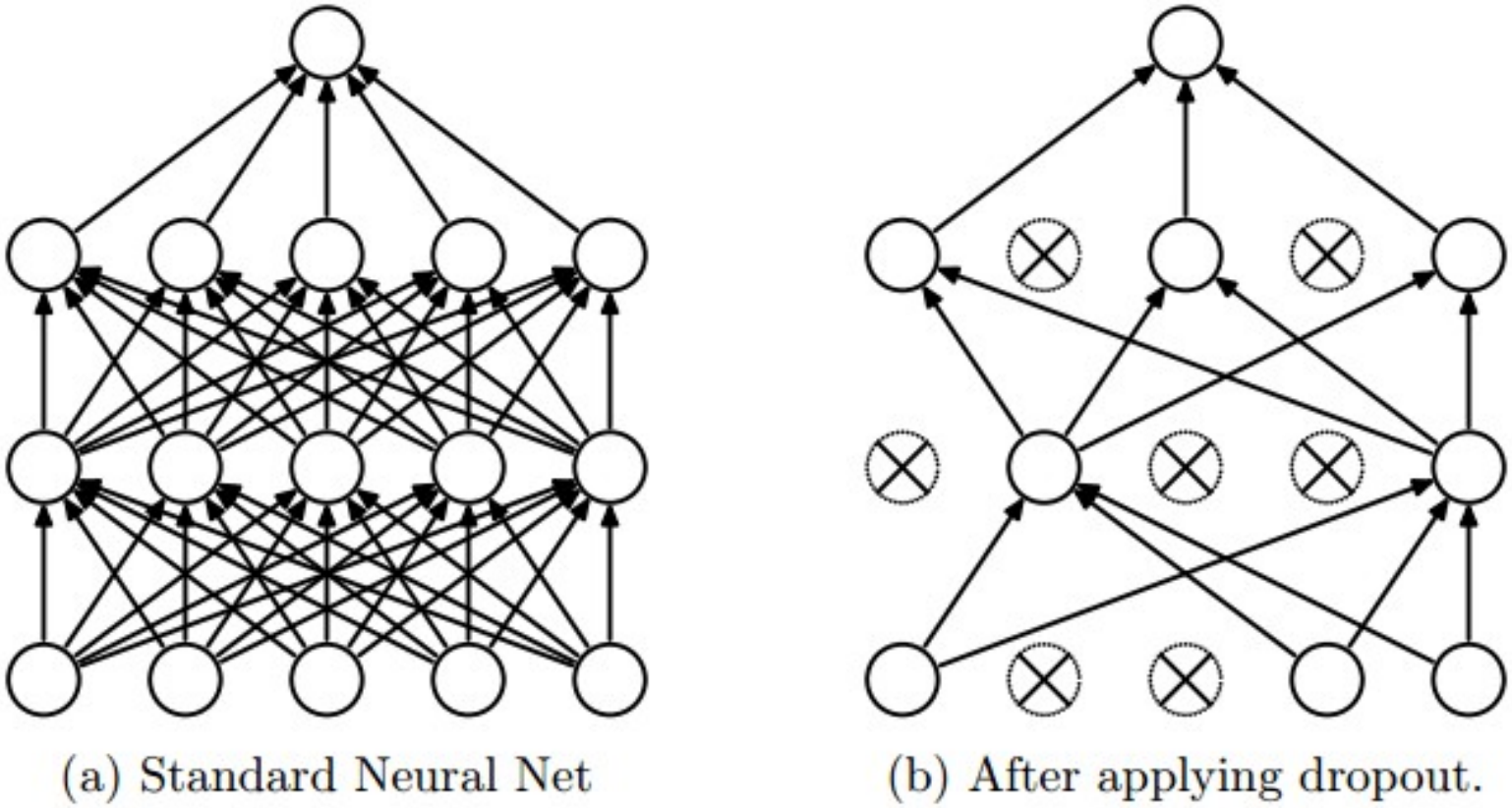
Other Pooling Layers

- Average Pooling
- No Pooling? **Striving for Simplicity: The All Convolutional Net**
  Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller

Building Blocks:
Dropout

Regularization: **Dropout**

"During training, randomly set some neurons to zero in the forward pass"



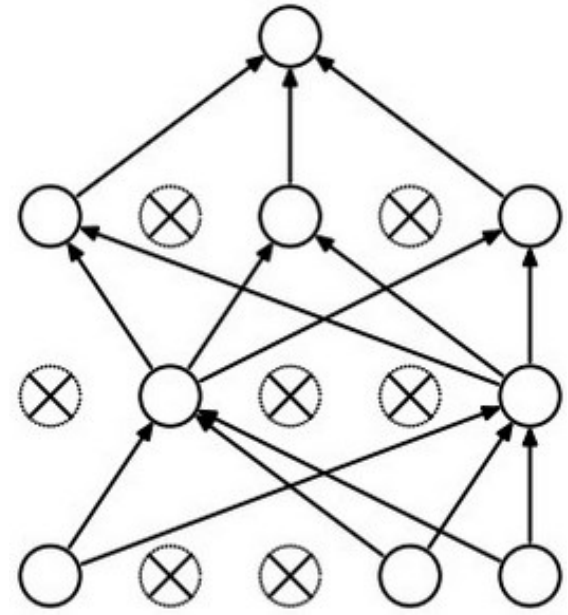
(a) Standard Neural Net

(b) After applying dropout.

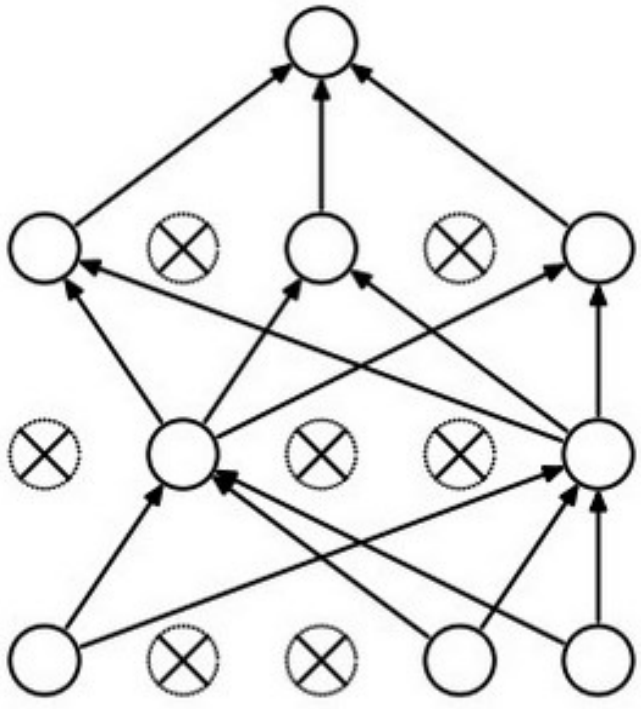
[Srivastava et al., 2014]

```
require 'nn';
p = 0.5
x = torch.rand(5)
L1 = nn.Linear(5, 5)
L2 = nn.Linear(5, 5)
L3 = nn.Linear(5, 5)
L4 = nn.Linear(5, 1)


H1 = L1:forward(x)
U1 = torch.rand(H1:size(1)):gt(p):double()
H1 = H1:cmul(U1)

H2 = L2:forward(H1)
U2 = torch.rand(H2:size(1)):gt(p):double()
H2 = H2:cmul(U2)

H3 = L3:forward(H2)
U3 = torch.rand(H3:size(1)):gt(p):double()
H2 = H3:cmul(U3)

out = L4:forward(H3)
```

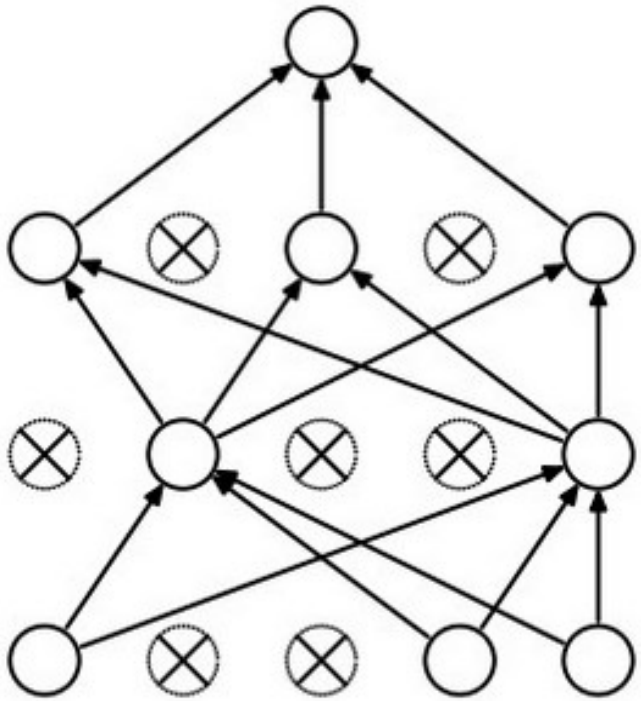
Example forward pass with a 3-layer network using dropout



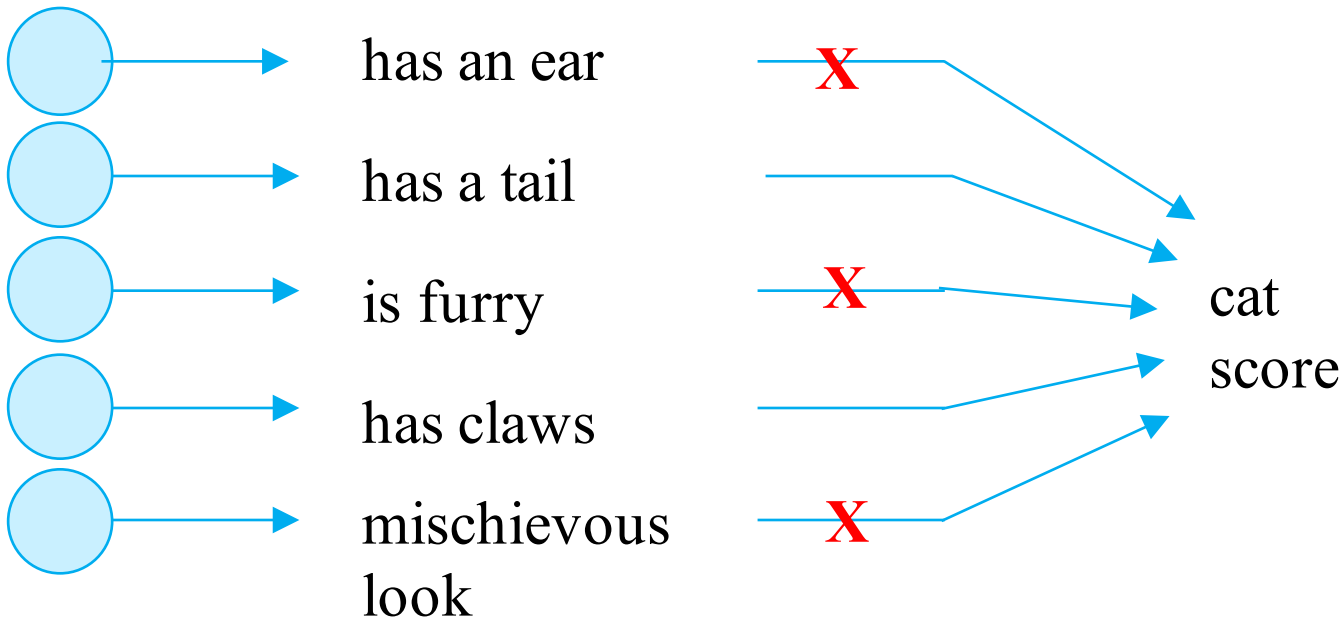
Waaaait a second…
How could this possibly be a good idea?



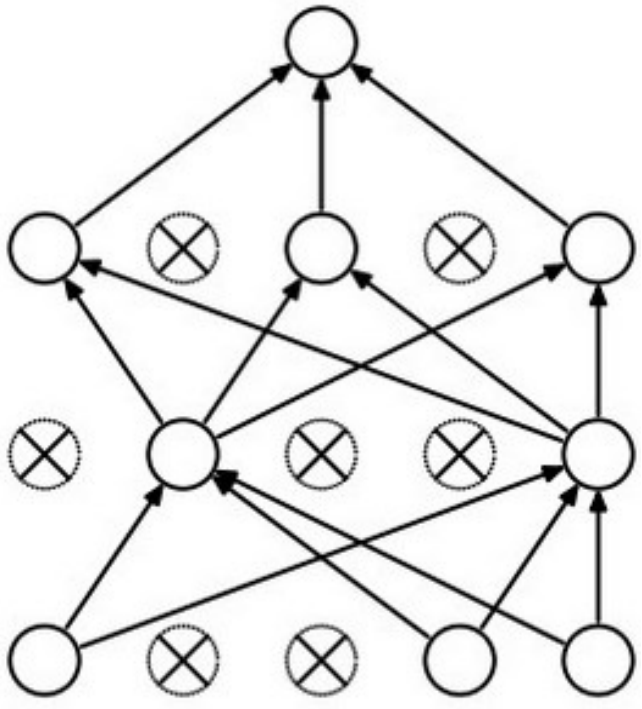
Waaaait a second…
How could this possibly be a good idea?



Forces the network to have a redundant representation.



Waaaait a second…
How could this possibly be a good idea?



Another interpretation:

Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model, gets trained on only ~one batch.

At test time….

At test time all neurons are always ON

We must scale the activations so that for each neuron:

output at test time = expected output at training time

Dropout Summary

In [21]:
```
require 'nn';
p = 0.5
x = torch.rand(5)
L1 = nn.Linear(5, 5)
L2 = nn.Linear(5, 5)
L3 = nn.Linear(5, 5)
L4 = nn.Linear(5, 1)

function forward_train(x)
    H1 = L1:forward(x)
    U1 = torch.rand(H1:size(1)):gt(p):double()
    H1 = H1:cmul(U1)

    H2 = L2:forward(H1)
    U2 = torch.rand(H2:size(1)):gt(p):double()
    H2 = H2:cmul(U2)

    H3 = L3:forward(H2)
    U3 = torch.rand(H3:size(1)):gt(p):double()
    H2 = H3:cmul(U3)

    out = L4:forward(H3)
    return out
end

function forward_test(x)
    H1 = L1:forward(x) * p
    H2 = L2:forward(H1) * p
    H3 = L3:forward(H2) * p
    out = L4:forward(H3)
    return out
end
```

Drop in forward pass

Compensate at test time

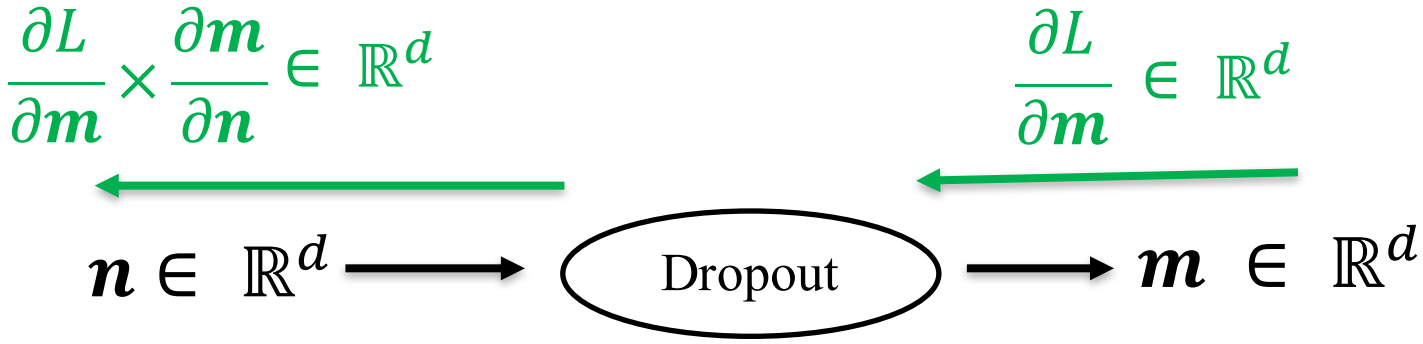
More common: "Inverted dropout"

In [ ]:

```
function forward_train(x)
    H1 = L1:forward(x)
    U1 = torch.rand(H1:size(1)):gt(p):double()
    H1 = H1:cmul(U1) / p

    H2 = L2:forward(H1)
    U2 = torch.rand(H2:size(1)):gt(p):double()
    H2 = H2:cmul(U2) / p

    H3 = L3:forward(H2)
    U3 = torch.rand(H3:size(1)):gt(p):double()
    H2 = H3:cmul(U3) / p

    out = L4:forward(H3)
    return out
end

function forward_test(x)
    H1 = L1:forward(x)
    H2 = L2:forward(H1)
    H3 = L3:forward(H2)
    out = L4:forward(H3)
    return out
end
```

test time is unchanged!
(as in Torch7)

Building Blocks – Dropout - Regularization




```
> n = torch.rand(5)
> drop = nn.Dropout(.5)
> drop.train = true
> =n
 0.6549
 0.2767
 0.2258
 0.5753
 0.6580
[torch.DoubleTensor of size 5]

> =drop:forward(n)
 1.3097
 0.5534
 0.4515
 1.1505
 1.3160
[torch.DoubleTensor of size 5]
```

```
> return drop:forward(n)
 1.3097
 0.5534
 0.4515
 0.0000
 0.0000
[torch.DoubleTensor of size 5]

> return drop:forward(n)
 0.0000
 0.0000
 0.4515
 0.0000
 0.0000
[torch.DoubleTensor of size 5]
```

```
> drop:backward(n, torch.ones(5))
> =drop.gradInput
 0
 0
 2
 0
 0
[torch.DoubleTensor of size 5]
```

```
> drop.train = false
> m = drop:forward(n)
> =m
 0.6549
 0.2767
 0.2258
 0.5753
 0.6580
[torch.DoubleTensor of size 5]
```

Thank you!