Convolutional Neural Networks

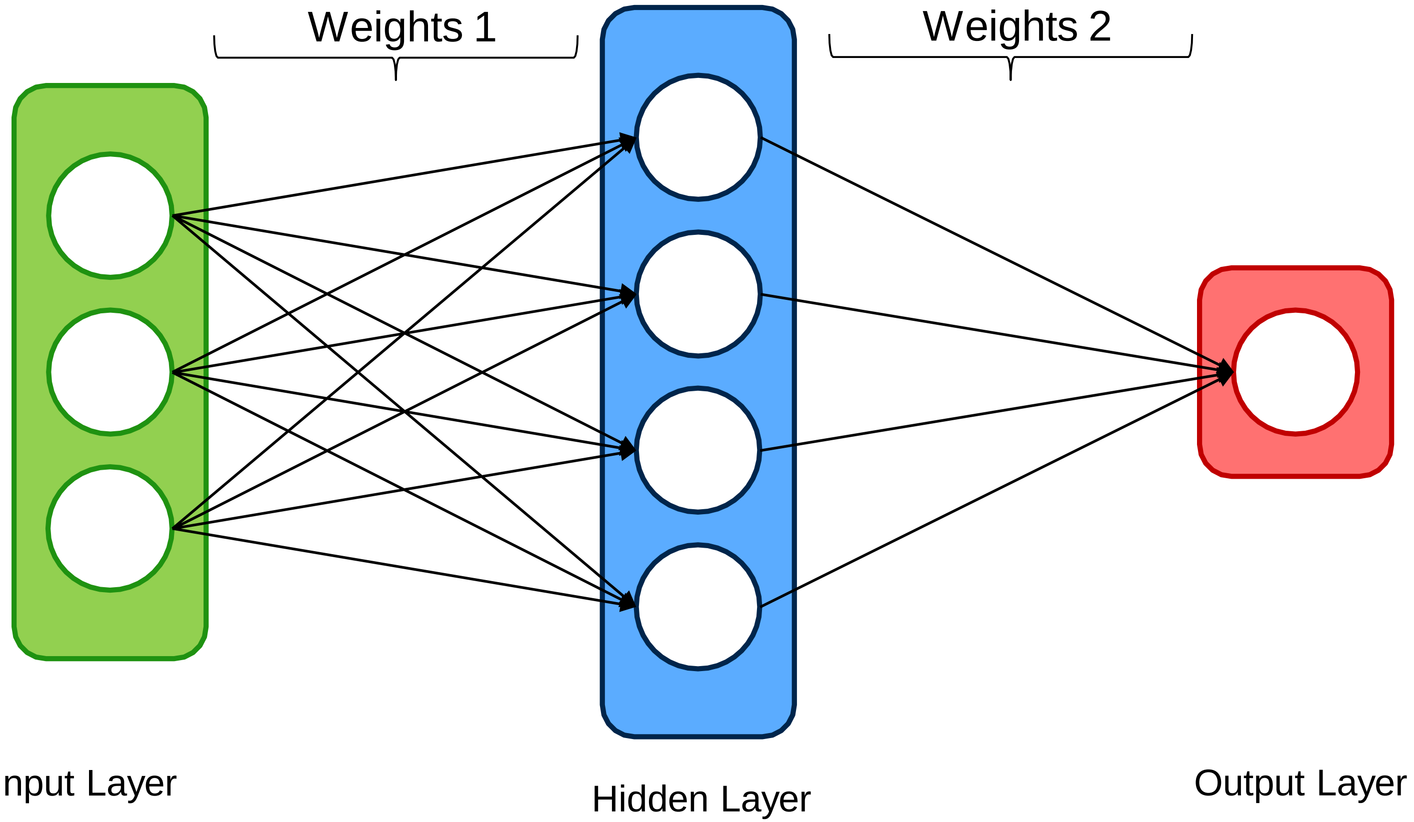
Introduction to Deep learning

Agenda

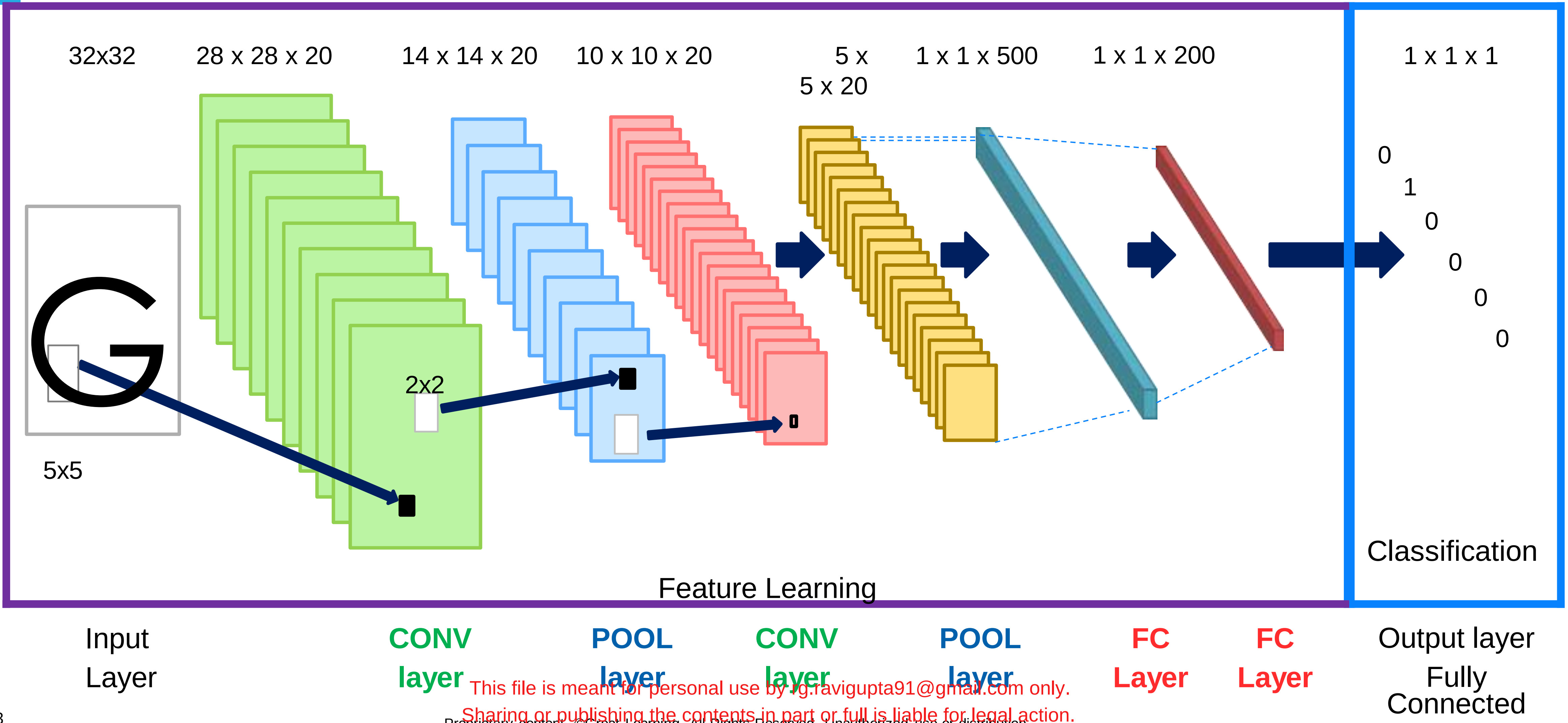
1. Architecture of CNN
2. Convolution and Filter - Hands on
3. Feature Map
4. Max-pool layers
5. Other pooling types
6. Sequential model compilation - Hands on
7. Cass study ; Image classification using CNN - Hands On

Convolution Network vs. Plain Neural Network

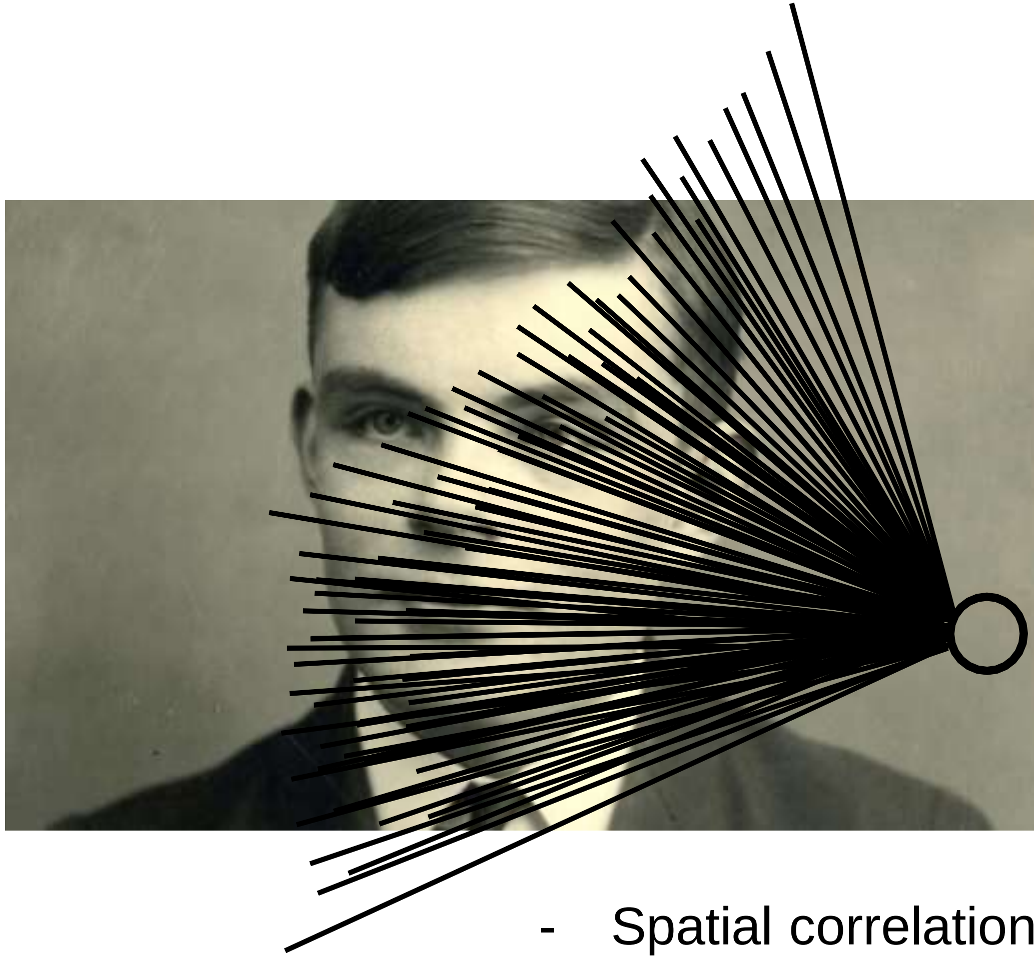
2 layered Plain Neural Network



Convolution Network vs. Plain Neural Network



Convolution

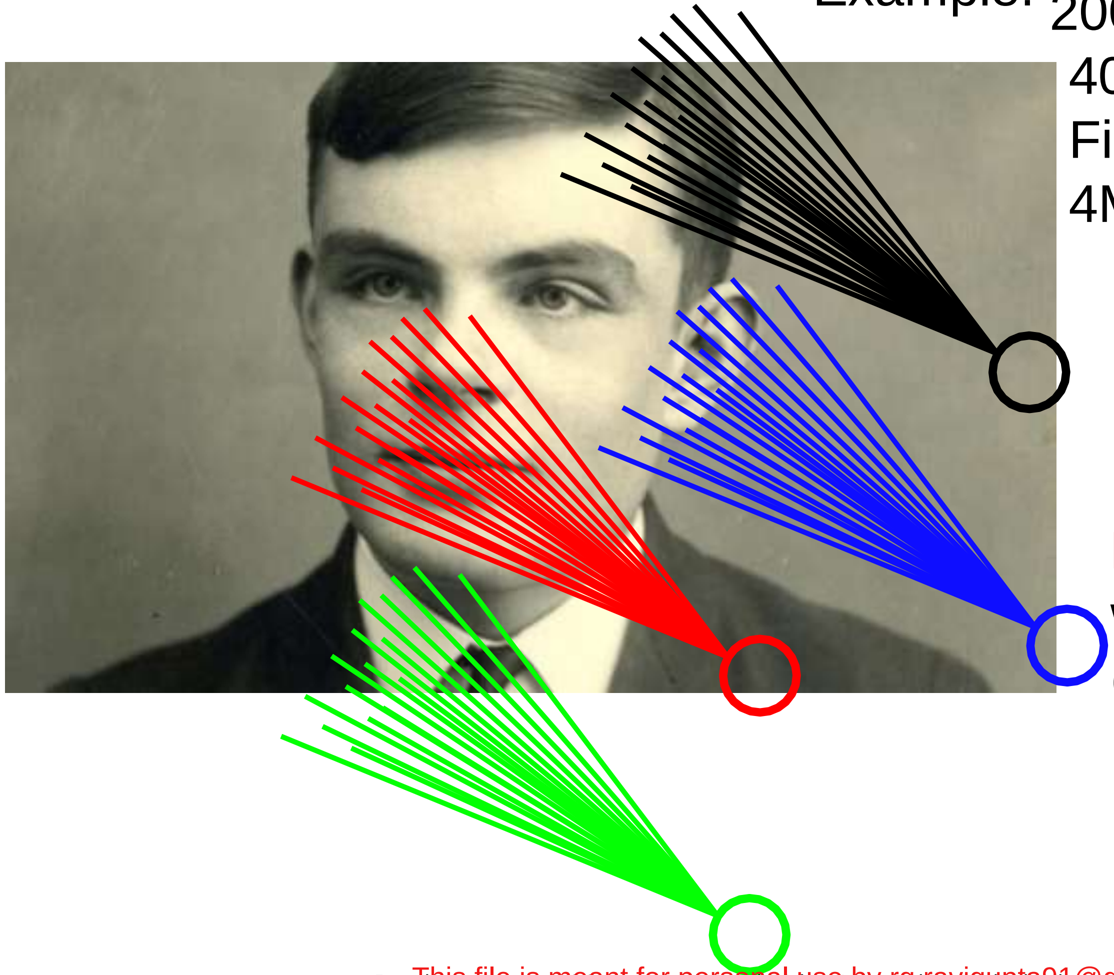


Example: 200x200 image
40K hidden units

→ **~2B parameters!!!**

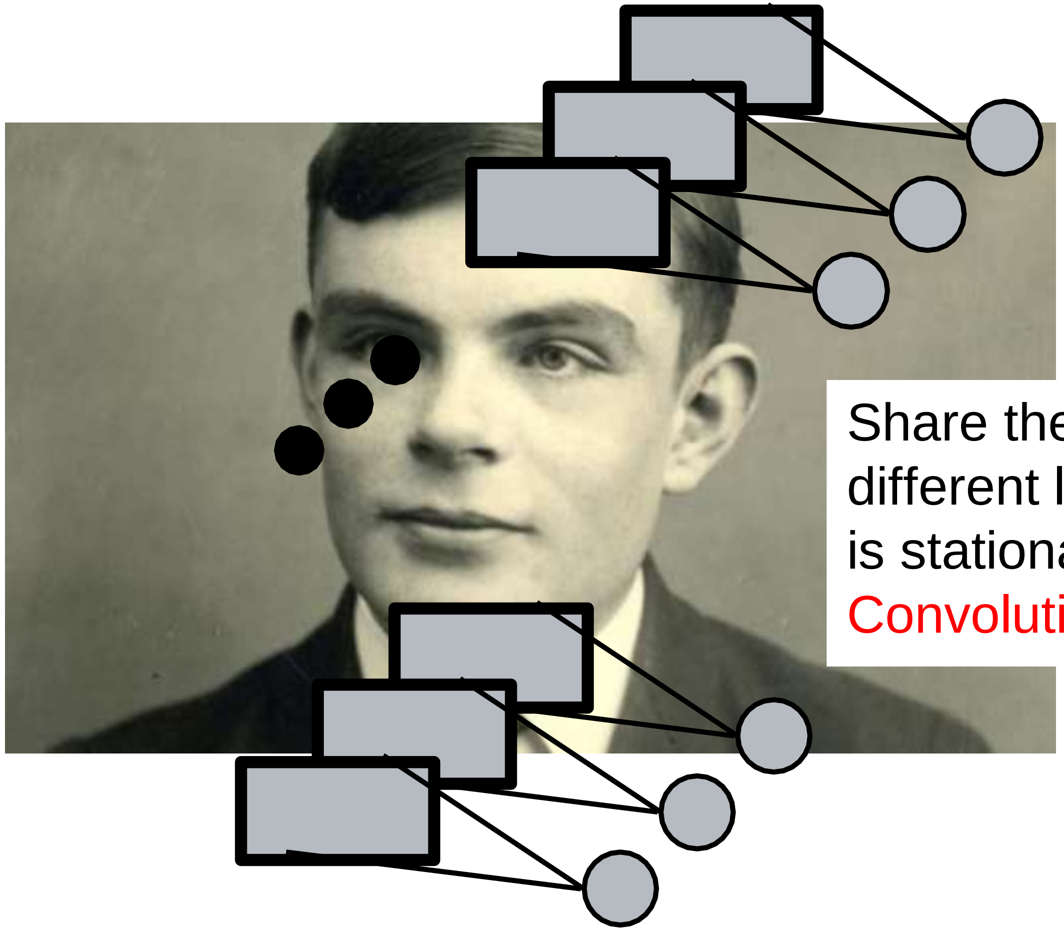
- Spatial correlation is local
- Waster of resources + we do not have enough training samples anyway

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters



Note: This parametrization is good when input image is registered (e.g. face recognition)

Transitional Invariance



Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

Convolution (Discrete 1D)

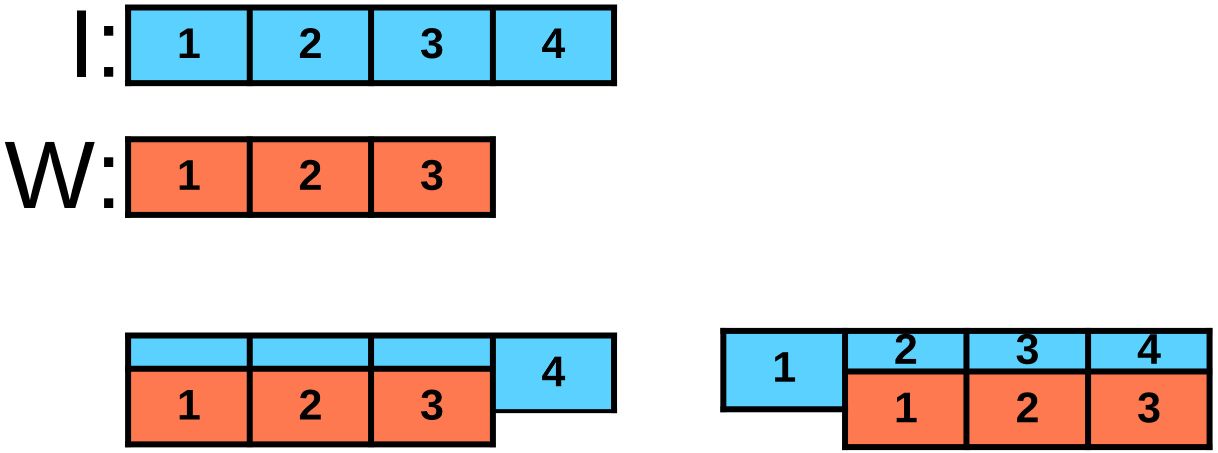
I:

1	2	3	4

W:

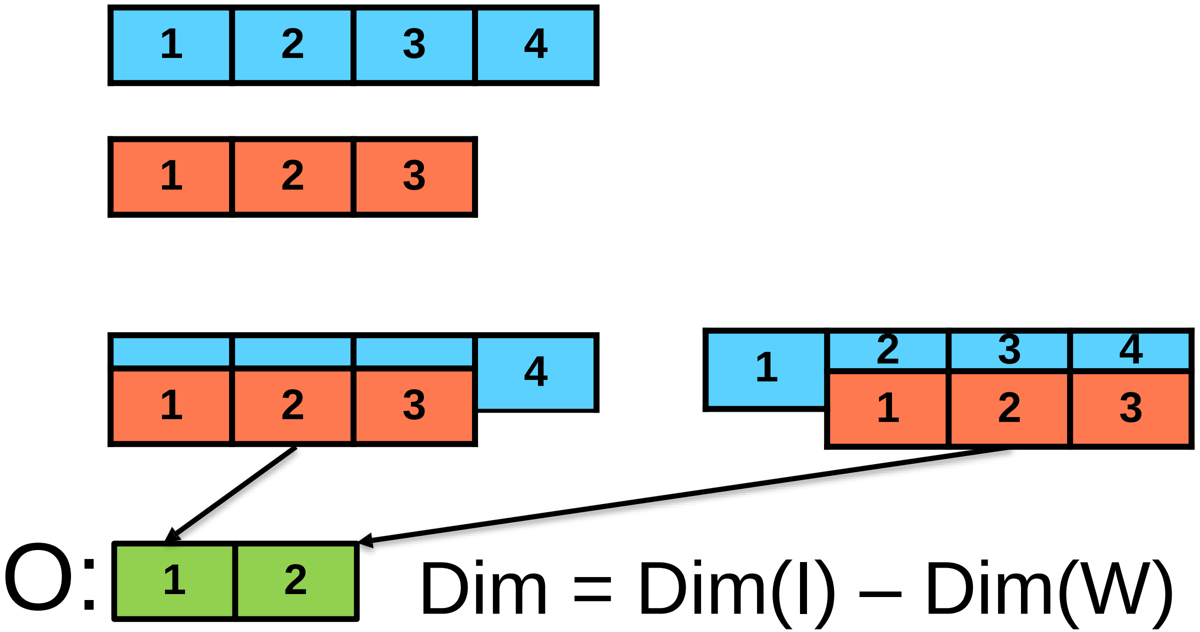
1	2	3

Convolution (Discrete 1D)



Slide

Convolution (Discrete 1D)



Slide

W:

Dim = Dim(I) – Dim(W) + 1

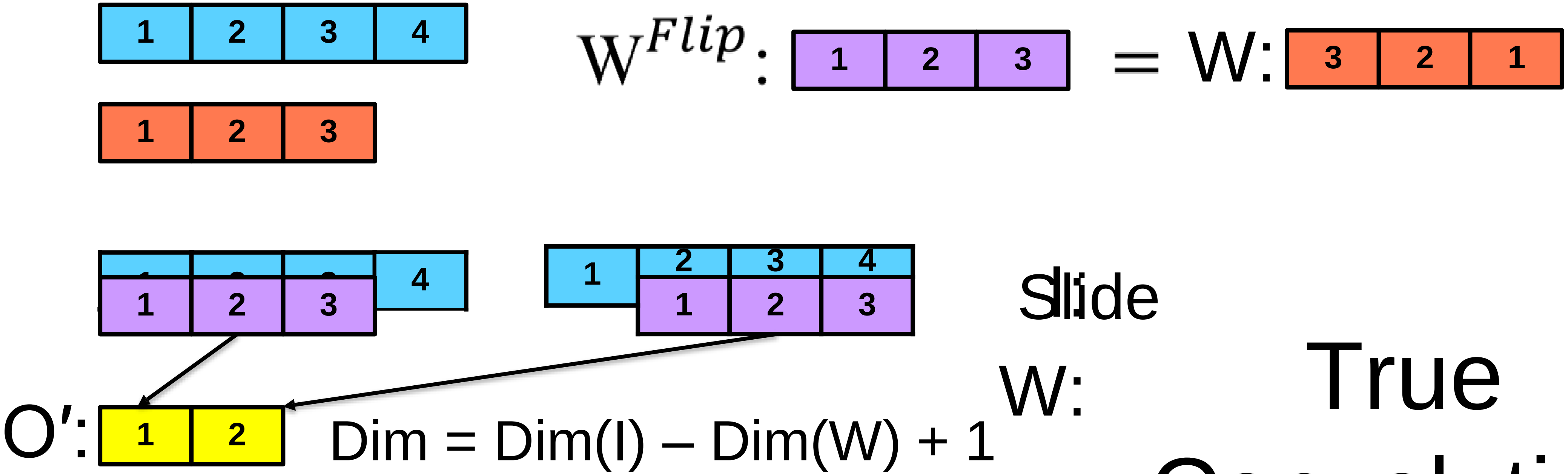
$$O_1 = I_1 W_1 + I_2 W_2 + I_3 W_3$$

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

Correlation

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

Convolution (Discrete 1D)



Slide W:

True Convolution

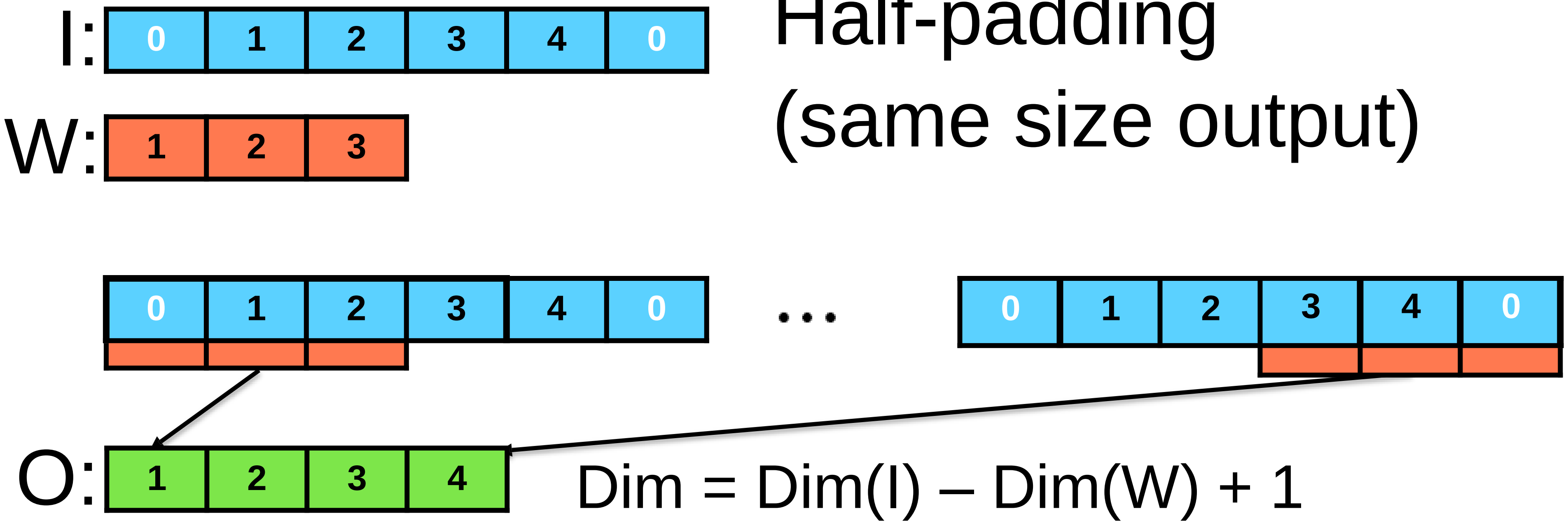
$$O'_1 = I_1 W_1^{Flip} + I_2 W_2^{Flip} + I_3 W_3^{Flip}$$

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

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

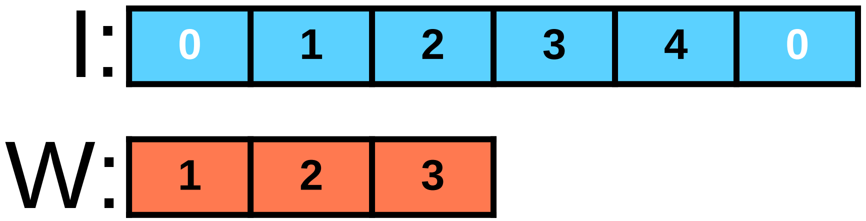
Convolution (Discrete 1D)

Half-padding
(same size output)

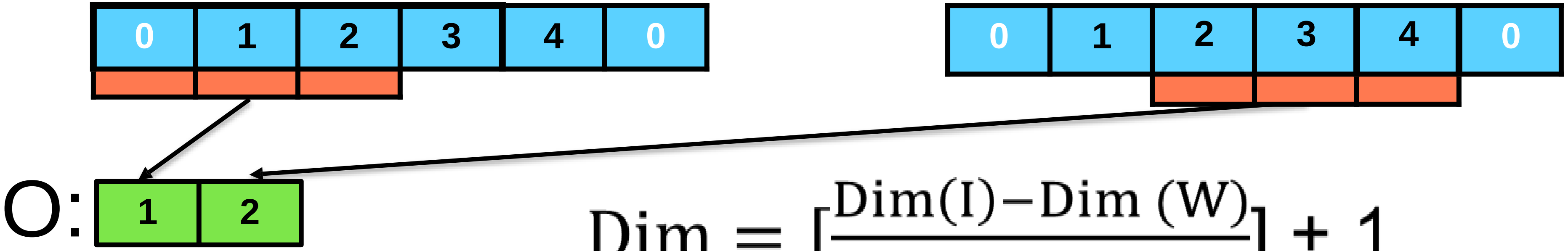


Dim = Dim(I) – Dim(W) + 1

Convolution (Discrete 1D)



Half-padding
(Stride = 2)



$$\text{Dim} = \left[\frac{\text{Dim(I)} - \text{Dim (W)}}{\text{Stride=2}}\right] + 1$$

Spatial Dimension - 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

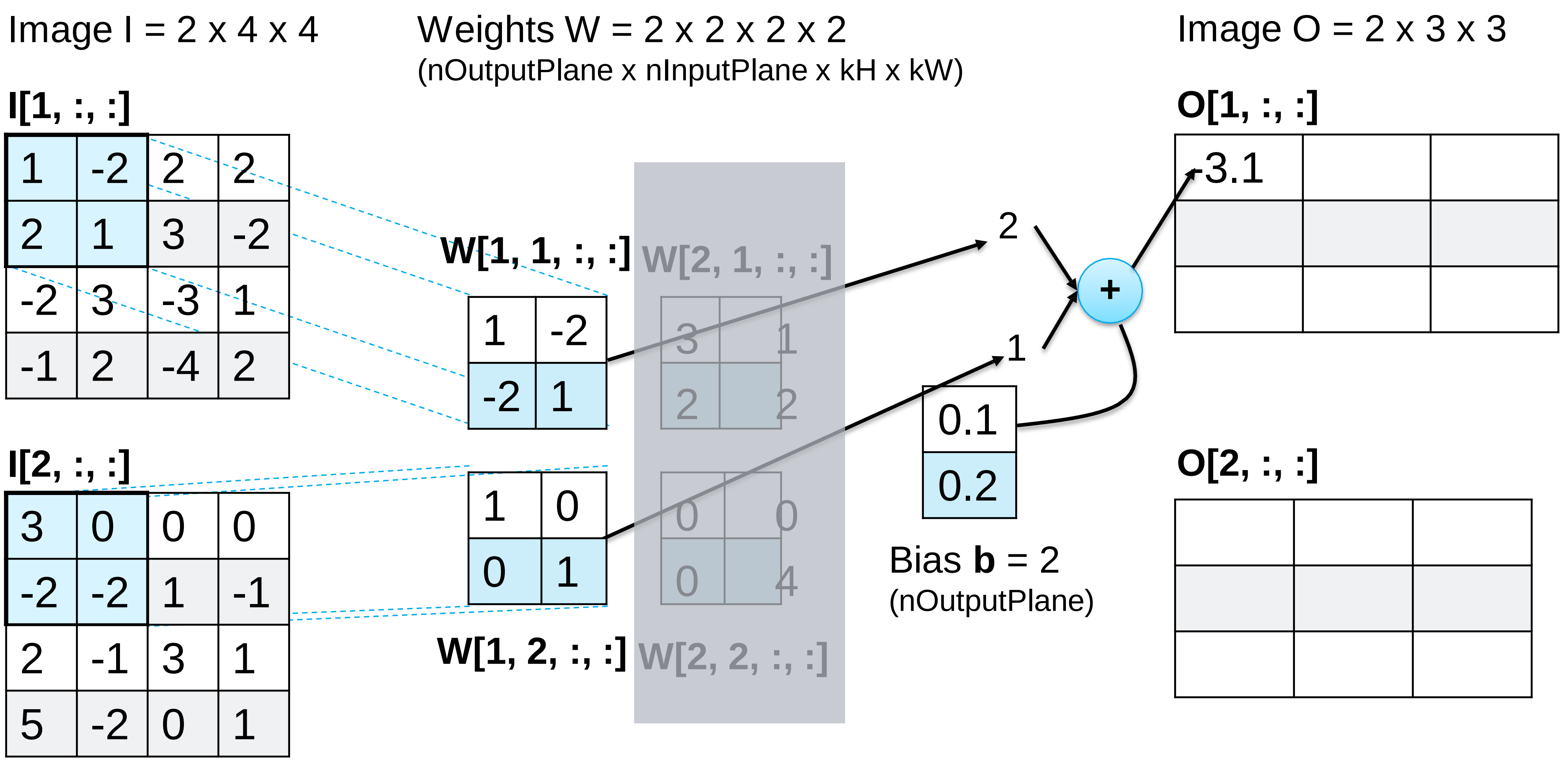
Bias **b** = 2
(nOutputPlane)

Image O = 2 x 3 x 3

O[1, :, :]

O[2, :, :]

Spatial Dimension - Convolution

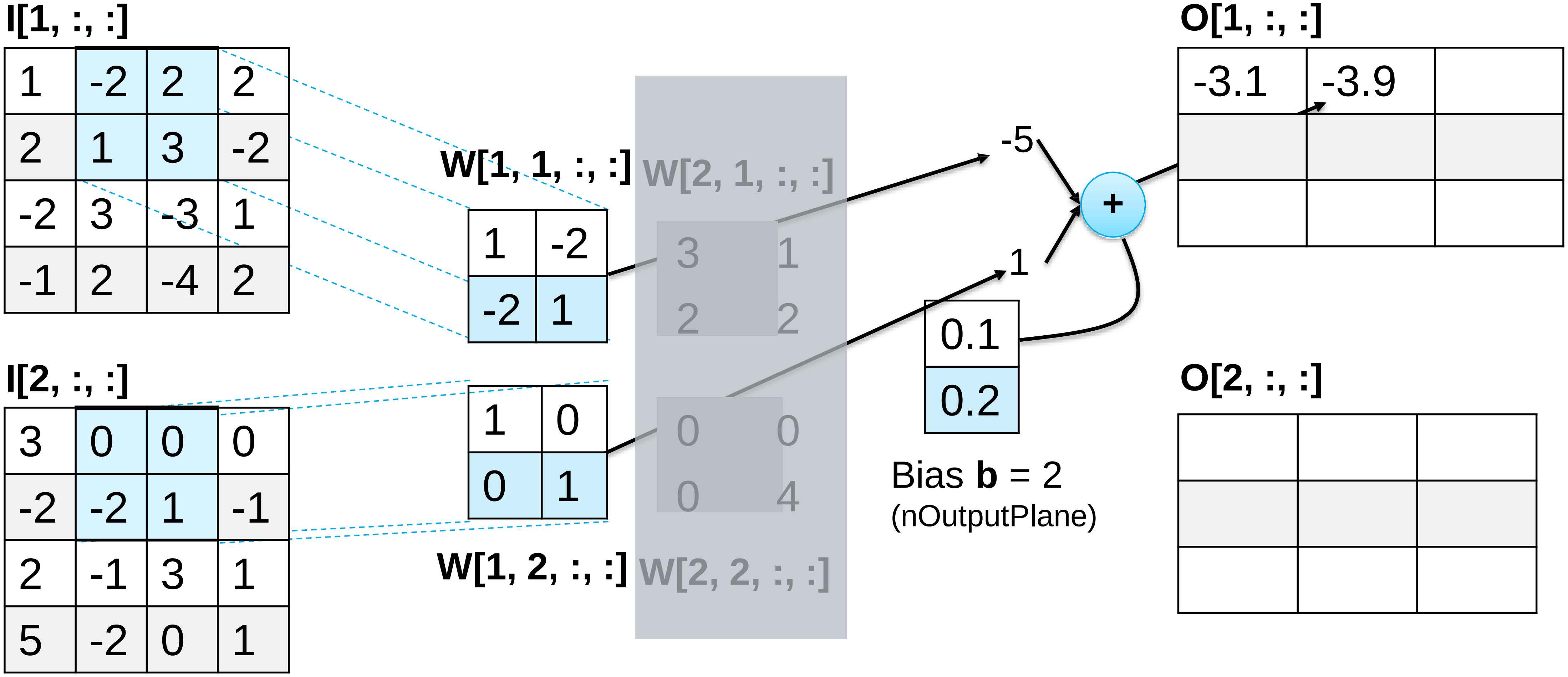


Spatial Dimension - Convolution

Image I = 2 x 4 x 4

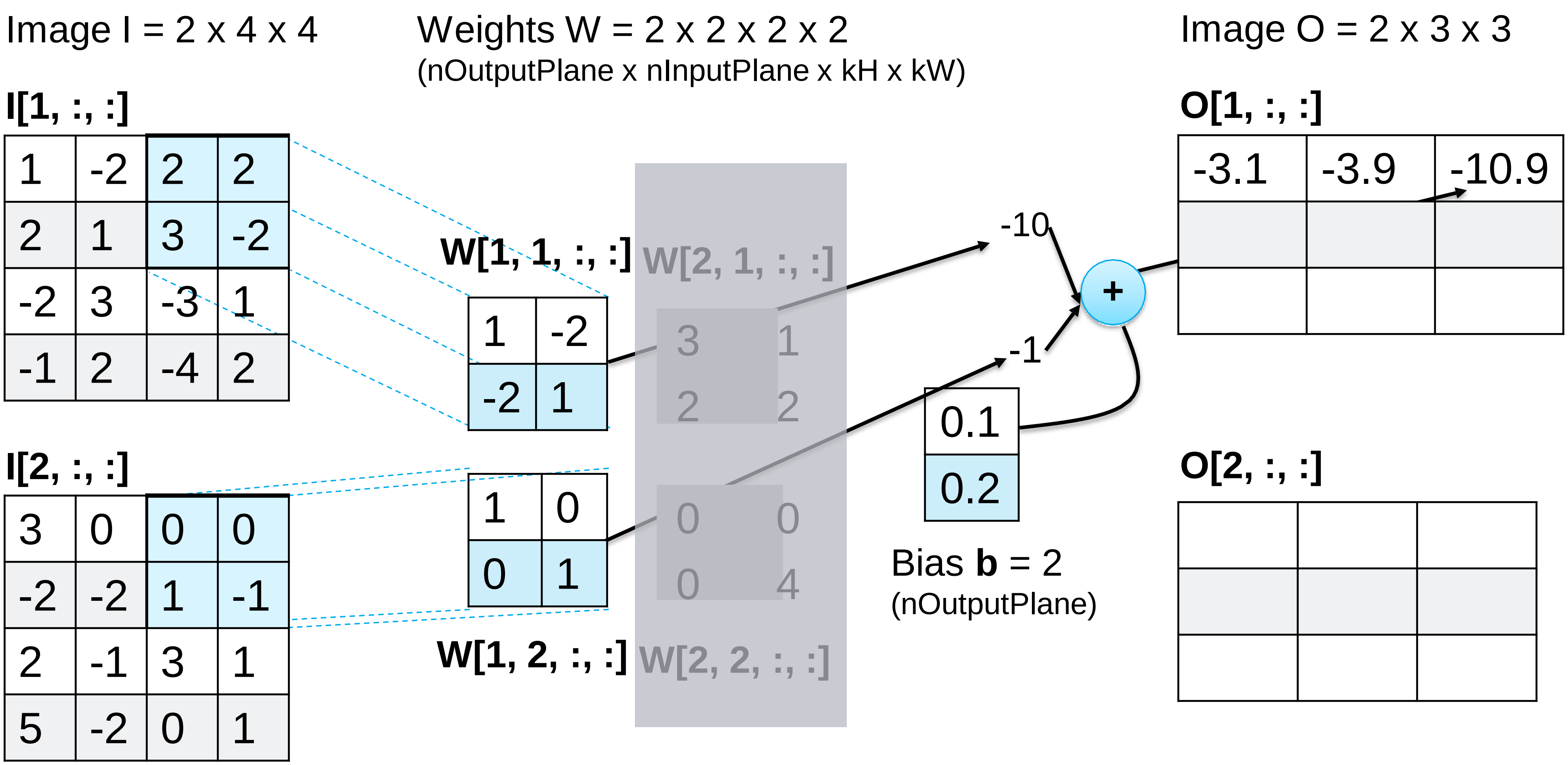
Weights W = 2 x 2 x 2 x 2
(nOutputPlane x nInputPlane x kH x kW)

Image O = 2 x 3 x 3



Bias **b** = 2
(nOutputPlane)

Spatial Dimension - Convolution



Spatial Dimension - 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

-13

1

+

0.1
0.2

Bias **b** = 2
(nOutputPlane)

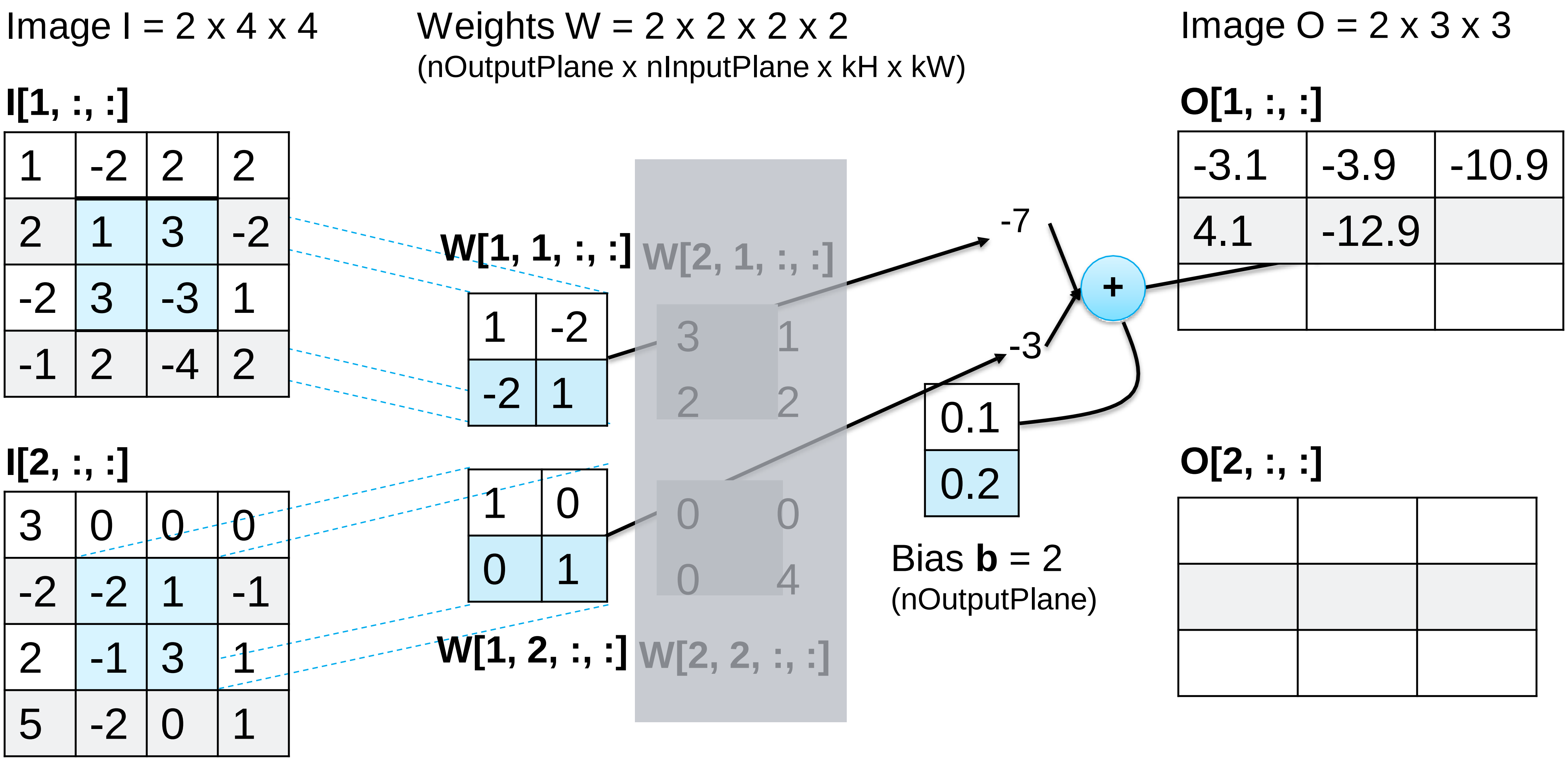
Image O = 2 x 3 x 3

O[1, :, :]

-3.1	-3.9	-10.9
4.1		

O[2, :, :]

Spatial Dimension - Convolution



Spatial Dimension - 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

14

2

+

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

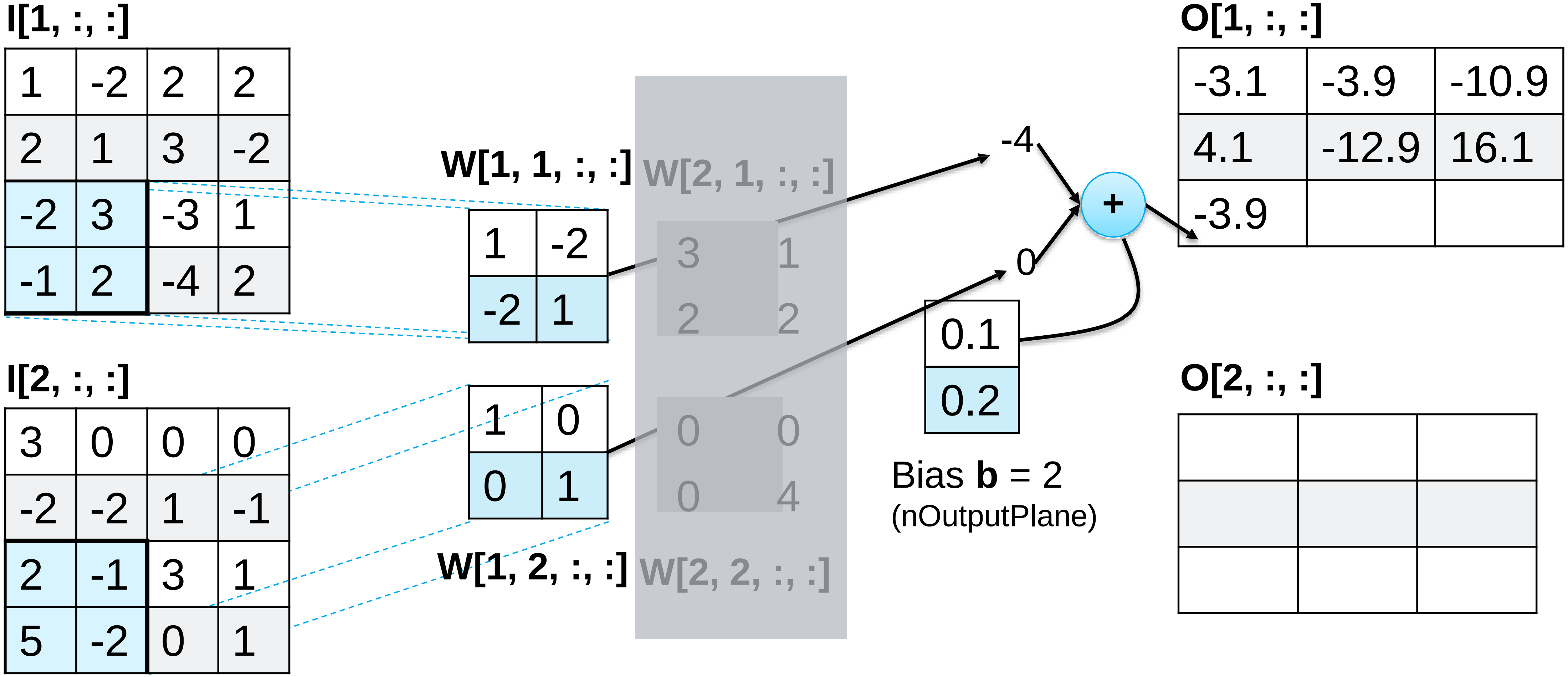
O[2, :, :]

Spatial Dimension - Convolution

Image I = 2 x 4 x 4

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

Image O = 2 x 3 x 3



Spatial Dimension - 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

1

-1

+

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	

O[2, :, :]

Spatial Dimension - 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

5

4

+

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, :, :]

Spatial Dimension - Convolution

greatlearning
Learning for Life

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

+

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		

Spatial Dimension - 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

4

4

+

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	

Spatial Dimension - 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

2

-4

+

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

Spatial Dimension - Convolution

greatlearning
Learning for Life

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

+

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		

Spatial Dimension - 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)

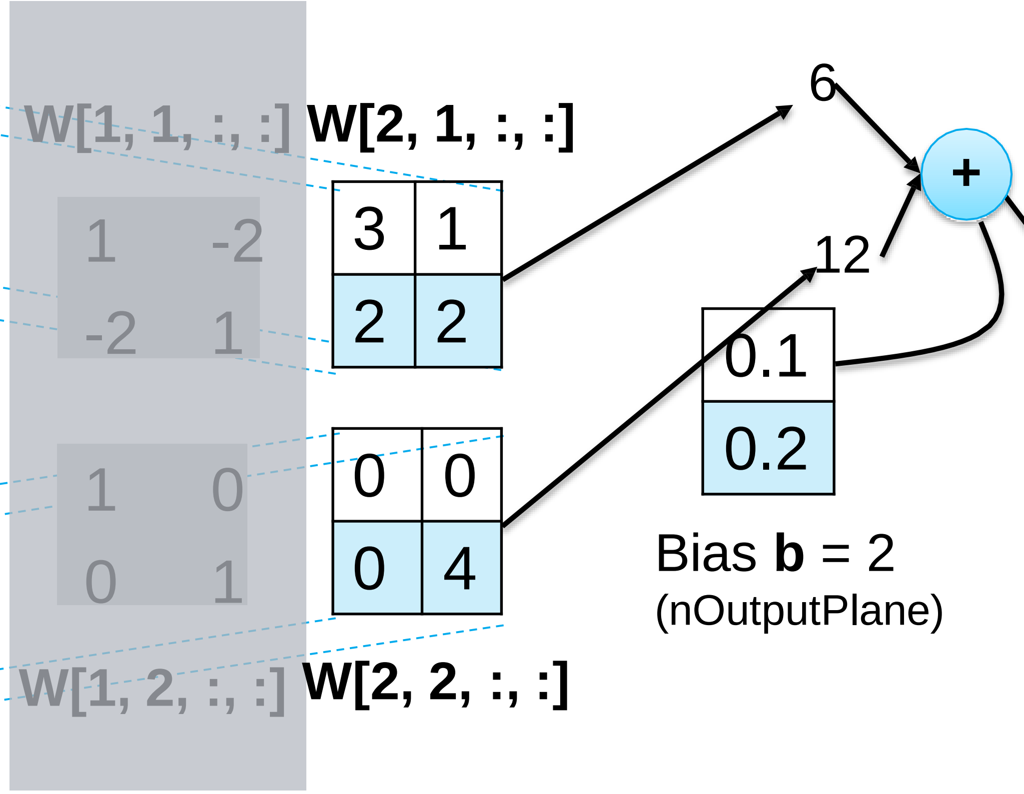


Image O = 2 x 3 x 3

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	

Spatial Dimension - Convolution

Image I = 2 x 4 x 4

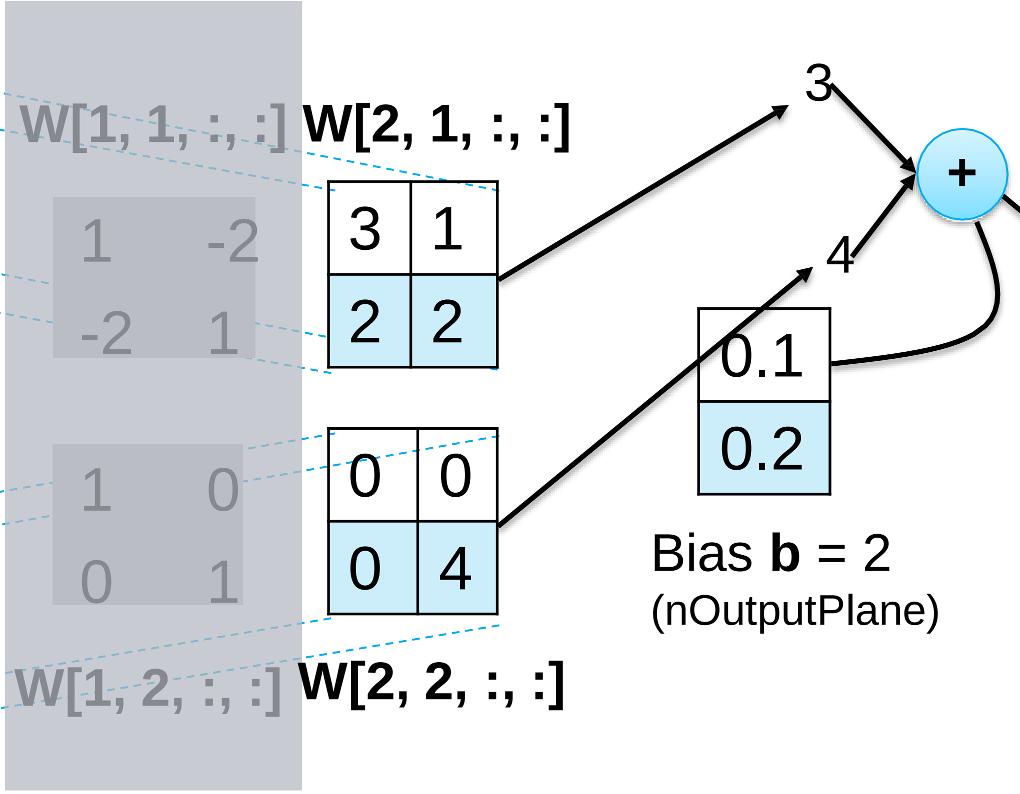
I[1, :, :]

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)



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

Spatial Dimension - 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		

Spatial Dimension - 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

2

0

+

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	2.2	

Spatial Dimension - 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

-12

4

+

0.1
0.2

Bias **b** = 2
(nOutputPlane)

Image O = 2 x 3 x 3

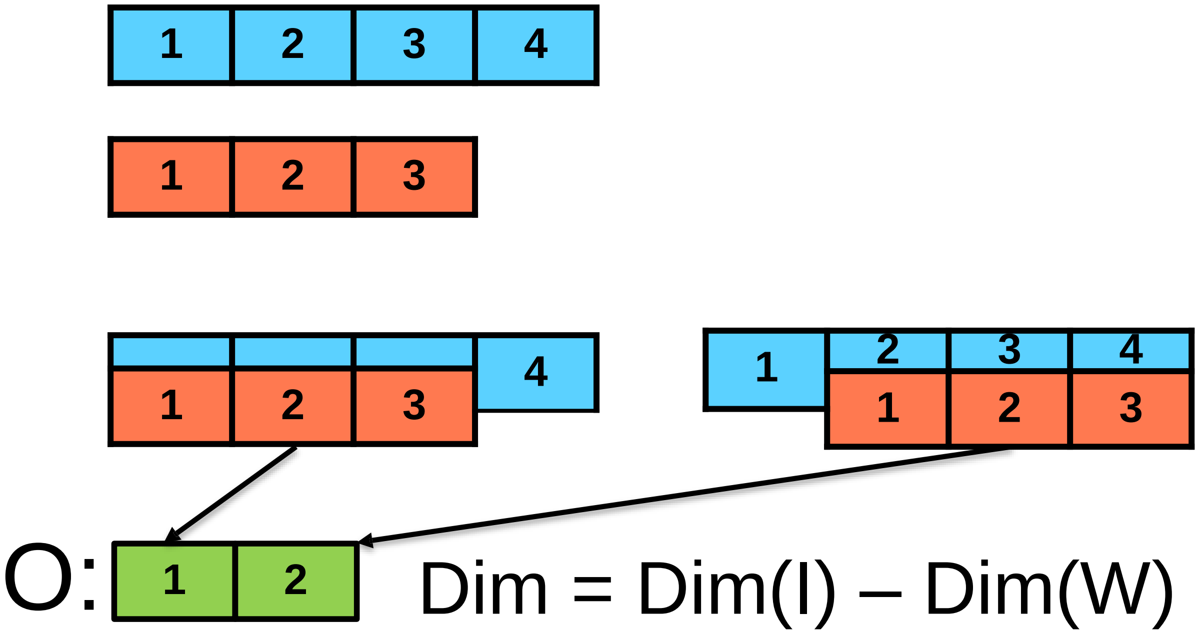
O[1, :, :]

-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	2.2	-7.8

Convolution - Backward



Slide

W:

Dim = Dim(I) – Dim(W) + 1

Correlation

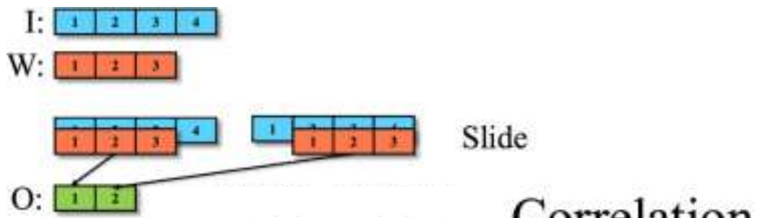
$$O_1 = I_1 W_1 + I_2 W_2 + I_3 W_3$$

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

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

Convolution - Backward

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



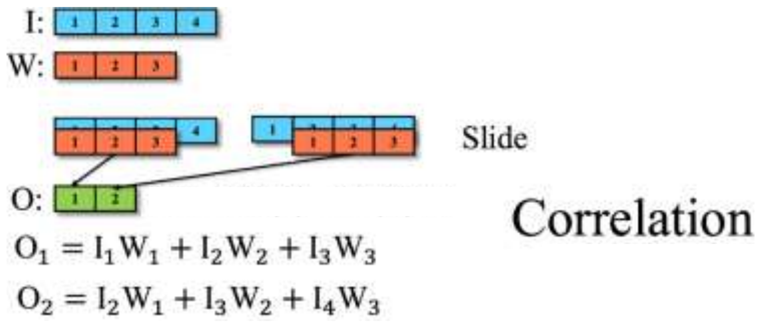
$$O_1 = I_1W_1 + I_2W_2 + I_3W_3$$

$$O_2 = I_2W_1 + I_3W_2 + I_4W_3$$

Convolution - Backward

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

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

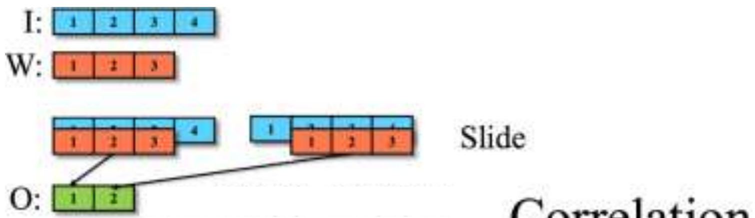


Convolution - Backward

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

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

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



$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

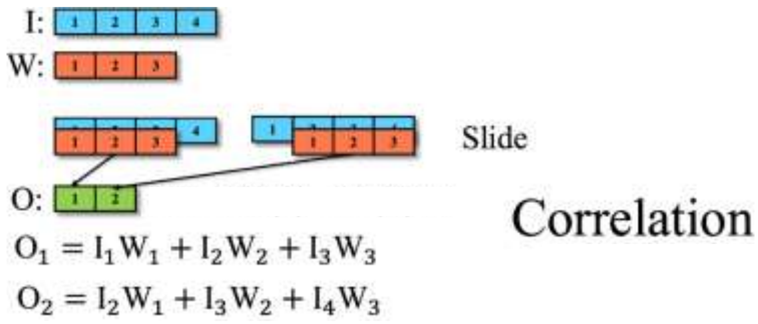
Convolution - Backward

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

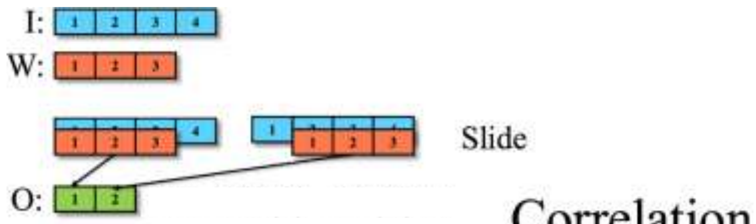
$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_3 & I_4 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{O}} = \begin{bmatrix} \partial LO_1 & \partial LO_2 \end{bmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} \partial LO_1 \times I_1 + \partial LO_2 \times I_2 & \partial LO_1 \times I_2 + \partial LO_2 \times I_3 & \partial LO_1 \times I_3 + \partial LO_2 \times I_4 \end{bmatrix}$$



Convolution - Backward



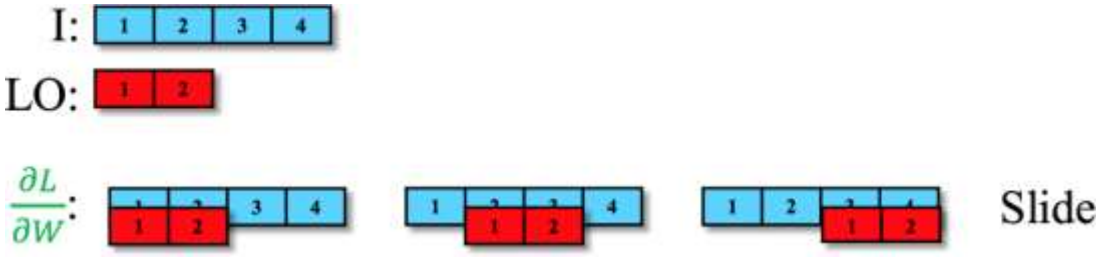
$$O_1 = I_1W_1 + I_2W_2 + I_3W_3$$
$$O_2 = I_2W_1 + I_3W_2 + I_4W_3$$

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

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

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

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



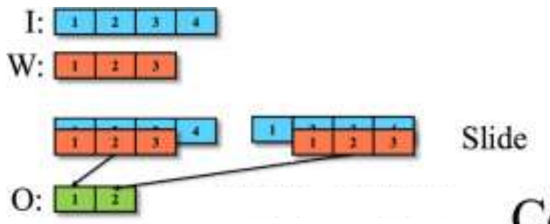
Convolution - Backward

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

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

$$\frac{\partial L}{\partial \boldsymbol{O}} = \begin{bmatrix} \partial LO_1 & \partial LO_2 \end{bmatrix}$$

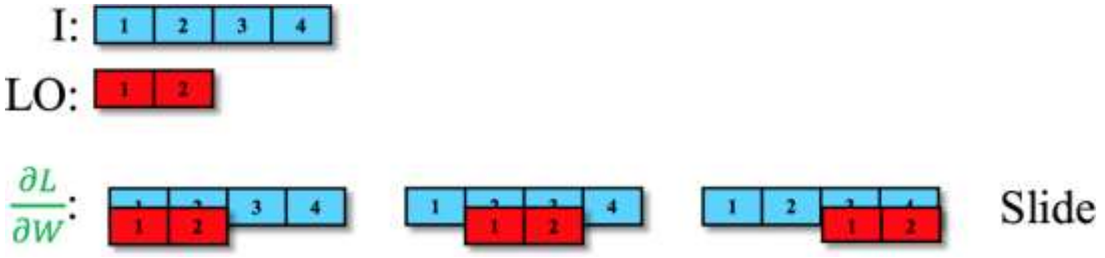
$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{W}} = \begin{bmatrix} \partial LO_1 \times I_1 + \partial LO_2 \times I_2 & \partial LO_1 \times I_2 + \partial LO_2 \times I_3 & \partial LO_1 \times I_3 + \partial LO_2 \times I_4 \end{bmatrix}$$



$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$

Correlation



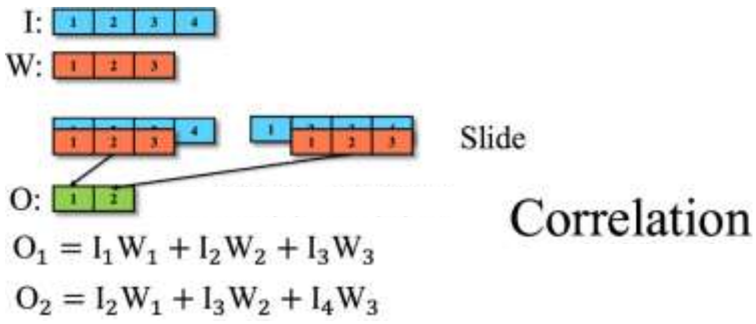
$$\frac{\partial L}{\partial \boldsymbol{W}} = \text{Correlation(I, LO)}$$

Convolution - Backward

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

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

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



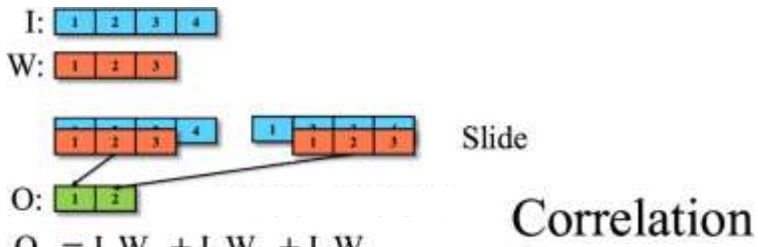
Convolution - Backward

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

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

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

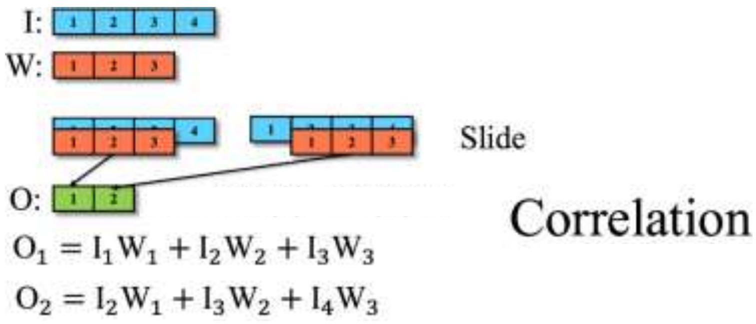
$$\frac{\partial L}{\partial \boldsymbol{I}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = [\partial LO_1 \times W_1 \quad \partial LO_1 \times W_2 + \partial LO_2 \times W_1 \quad \partial LO_1 \times W_3 + \partial LO_2 \times W_2 \quad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



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

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

Convolution - Backward

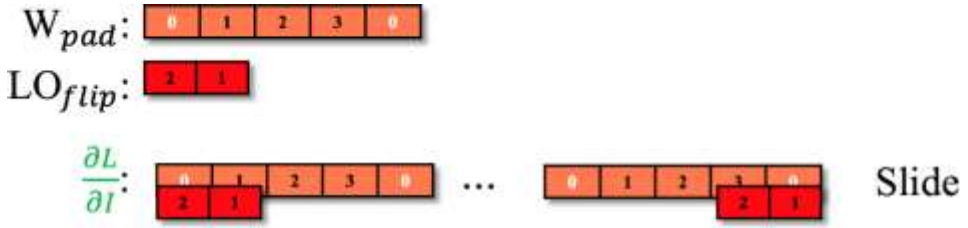


$$\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \begin{bmatrix} W_1 & W_2 & W_3 & 0 \\ 0 & W_1 & W_2 & W_3 \end{bmatrix}$$

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

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

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



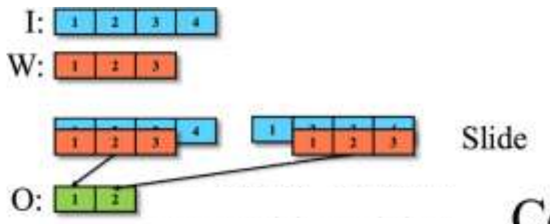
Convolution - Backward

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

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

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

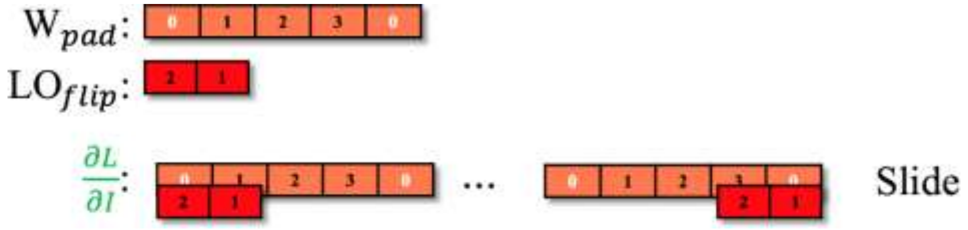
$$\frac{\partial L}{\partial \boldsymbol{I}} = \frac{\partial L}{\partial \boldsymbol{O}} \times \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = [\partial LO_1 \times W_1 \quad \partial LO_1 \times W_2 + \partial LO_2 \times W_1 \quad \partial LO_1 \times W_3 + \partial LO_2 \times W_2 \quad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



Correlation

$O_1 = I_1W_1 + I_2W_2 + I_3W_3$

$O_2 = I_2W_1 + I_3W_2 + I_4W_3$



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

Convolution - Forward

```python
n = tf.random_normal((1,4,1))
print(n)
```

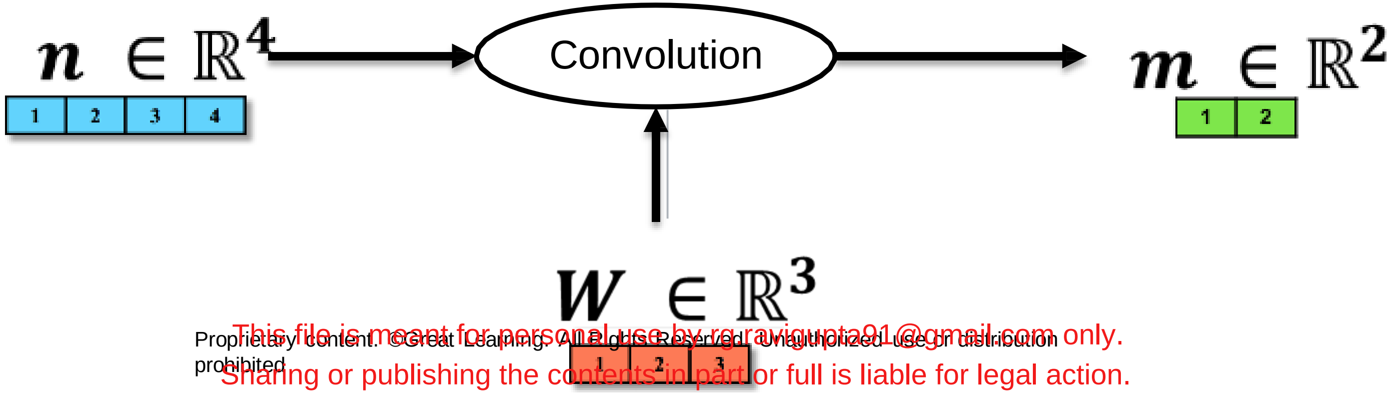
```
tf.Tensor(
[[[ 0.20350695]
  [-0.31481498]
  [-0.85656077]
  [ 0.5981919 ]]], shape=(1, 4, 1), dtype=float32)
```

```python
nextgrad = tf.random_normal((1,2,1))
print(nextgrad)
```

```
tf.Tensor(
[[[-0.47567356]
  [-2.3547919 ]]], shape=(1, 2, 1), dtype=float32)
```

```python
conv1 = tf.layers.Conv1D(1,3,1,'VALID',activation=None)
conv1.bias_initializer = tf.constant_initializer(0)
with tf.GradientTape(persistent=True) as t:
  t.watch(n)
  m = conv1(n)
  # A trick to make the gradOutput = nextgrad
  m1 = tf.multiply(nextgrad,m)
print(m)
```

```
tf.Tensor(
[[[-0.7662684]
  [ 0.2175143]]], shape=(1, 2, 1), dtype=float32)
```



Convolution - Backward

```
#This is equivalent to conv:backward(n,nextgrad)
gradWeight = t.gradient(m1,conv1.weights[0])
print(gradWeight)
```

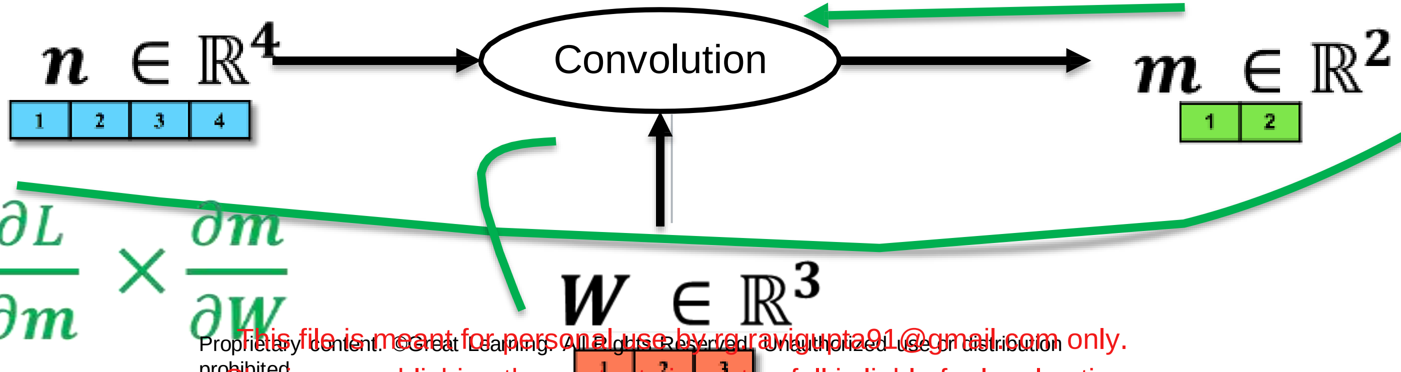
```
tf.Tensor(
[[[ 0.6445209]]

 [[ 2.1667714]]

 [[-1.0011742]]], shape=(3, 1, 1), dtype=float32)
```

```
conv1_back = tf.layers.Conv1D(1,2,1,'VALID',activation=None)
```

```
conv1_back.bias_initializer = tf.constant_initializer(0)
conv1_back.kernel_initializer = tf.constant_initializer(nextgrad.numpy())
gradWeight = conv1_back(n)
print(gradWeight)
```

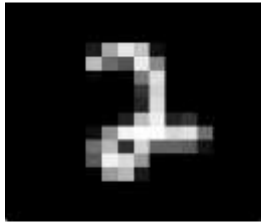
```
tf.Tensor(
[[[ 0.6445209]
  [ 2.1667714]
  [-1.0011742]]], shape=(1, 3, 1), dtype=float32)
```

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



$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial m} \times \frac{\partial m}{\partial W}$$

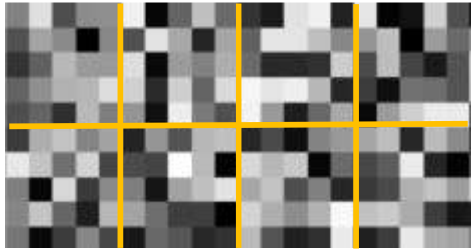
Convolution



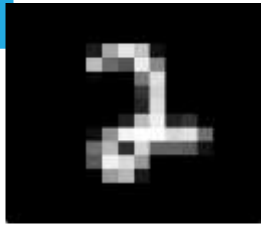
$\boldsymbol{n} \in \mathbb{R}^{16\times16}$ → Convolution → $\boldsymbol{m} \in \mathbb{R}^{8\times12\times12}$

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

$=$



Convolution



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

W

```
n = tf.random_normal((1,16,16,1))
print(n.shape)
```

(1, 16, 16, 1)

```
conv2 = tf.layers.Conv2D(8,[5,5],[1,1],
        "valid",activation=None)
```

```
m = conv2(n)
print(m.shape)
```

(1, 12, 12, 8)

```
print(conv2.weights[0].shape)
```

(5, 5, 1, 8)

```
print(conv2.weights[1].shape)
```

(8,)

Convolution

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

$\boldsymbol{n} \in \mathbb{R}^{2\times4\times4}$ → Convolution → $\boldsymbol{m} \in \mathbb{R}^{2\times3\times3}$

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

$W \in \mathbb{R}^{2\times2\times2\times2}$

```python
I = tf.constant([[[[1,-2],[2,1],[-2,3],[-1,2]],[[2,2],[3,-2],[-3,1],[-4,2]],[[3,0],[-2,-2],[2,-1],[5,-2]],[[0,0],[1,-1],
                 [3,1],[0,1]]]], dtype=tf.float32)
```

```python
print(I)
```

```python
conv = tf.layers.Conv2D(2,[2,2],[1,1],"valid",activation=None)
```

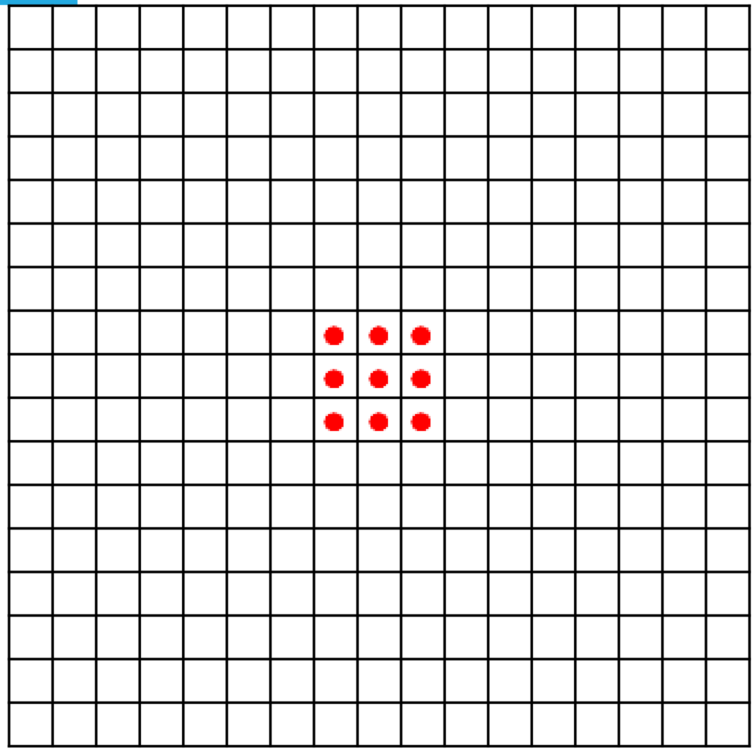
```python
conv.bias_initializer = tf.constant_initializer([0.1,0.2])
conv.kernel_initializer = tf.constant_initializer([[[[ 1, -2],[-2,  1]],[[1,  0],[0,  1]]],[[[3, 1],
                                   [2,  2]],[[0,  0],[ 0, 4]]]], dtype=tf.float32)
O = conv(I)
print(I.numpy()[0,:,:,:])
# conv.kernel_initializer = tf.constant_initializer()
```

```
[[[ 1. -2.]
  [ 2.  1.]
  [-2.  3.]
  [-1.  2.]]

 [[ 2.  2.]
  [ 3. -2.]
  [-3.  1.]
  [-4.  2.]]

 [[ 3.  0.]
  [-2. -2.]
  [ 2. -1.]
  [ 5. -2.]]
```

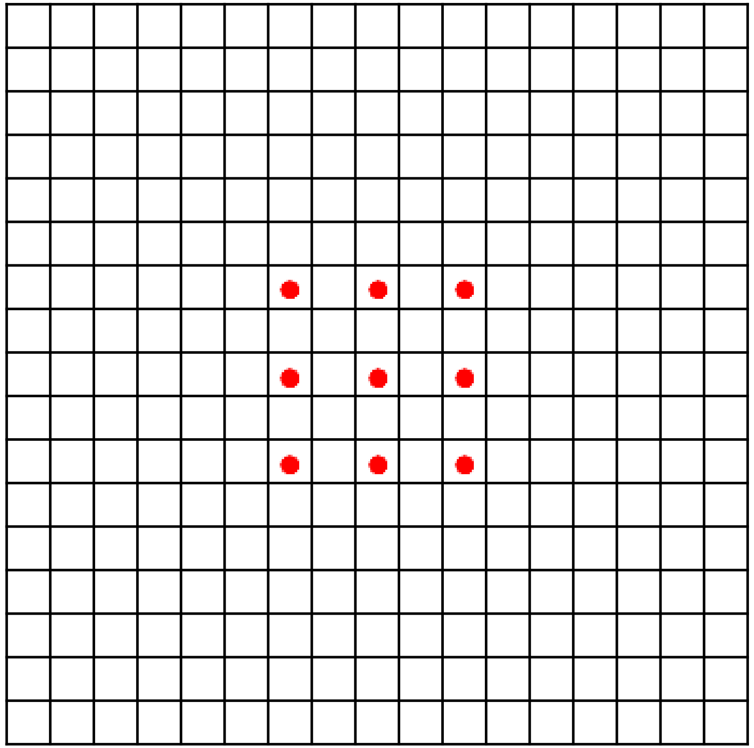
```python
print(O.numpy()[0,:,:,:])
```

```
[[[ 17.1  -4.8]
  [  3.1   3.2]
  [-15.9  16.2]]

 [[ 10.1  -8.8]
  [ -5.9 -16.8]
  [ -4.9   1.2]]

 [[  1.1 -11.8]
  [  5.1   4.2]
  [ 20.1   2.2]]]
```

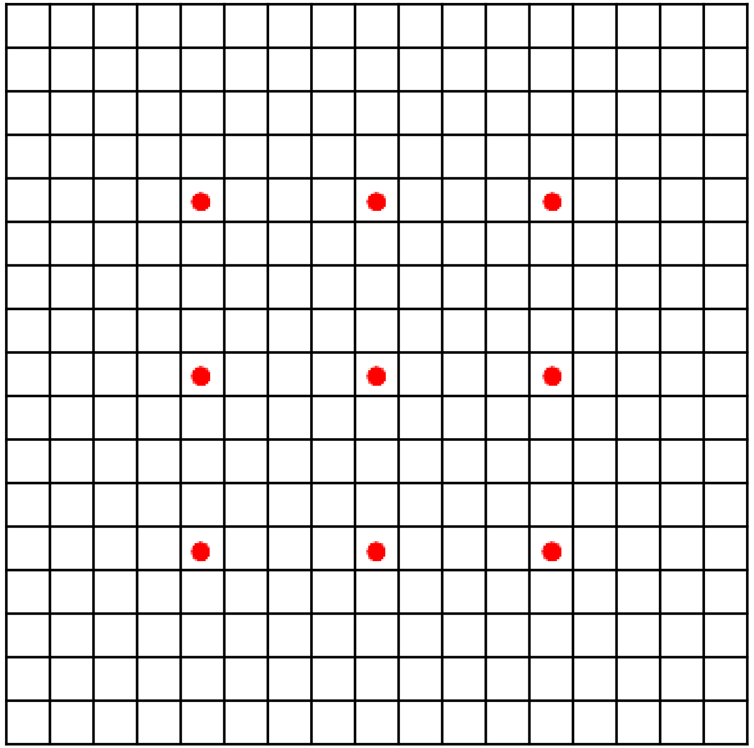
Aside: Dilated Convolution



(a)



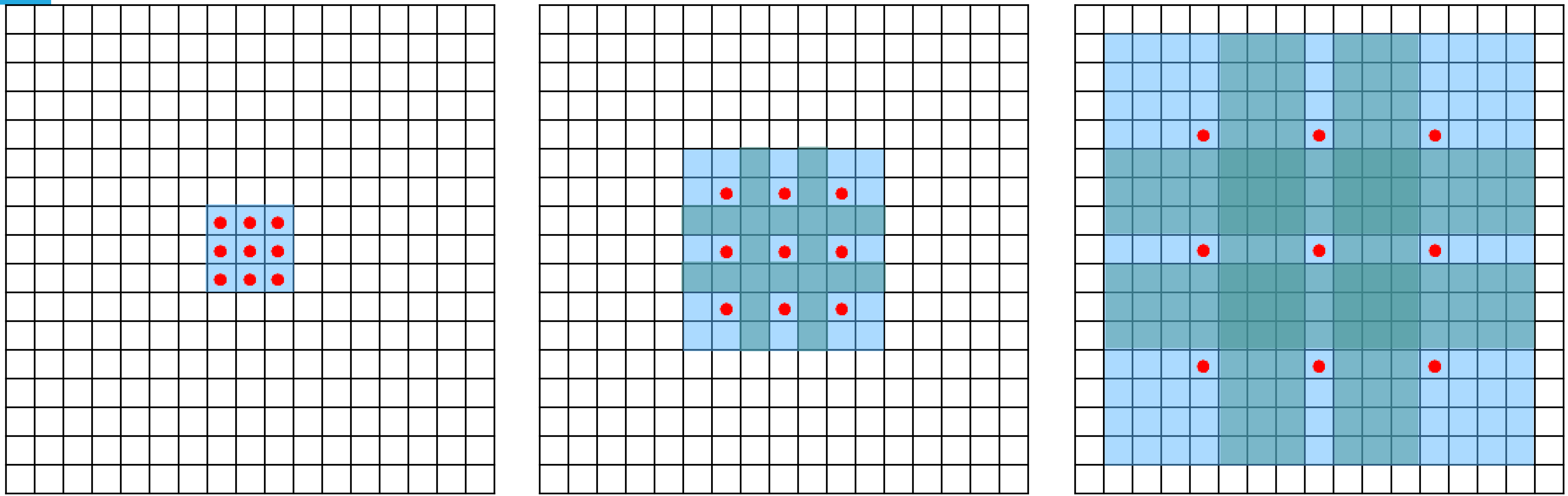
(b)



(c)

Aside: Dilated Convolution

nn.SpatialDilatedConvolution



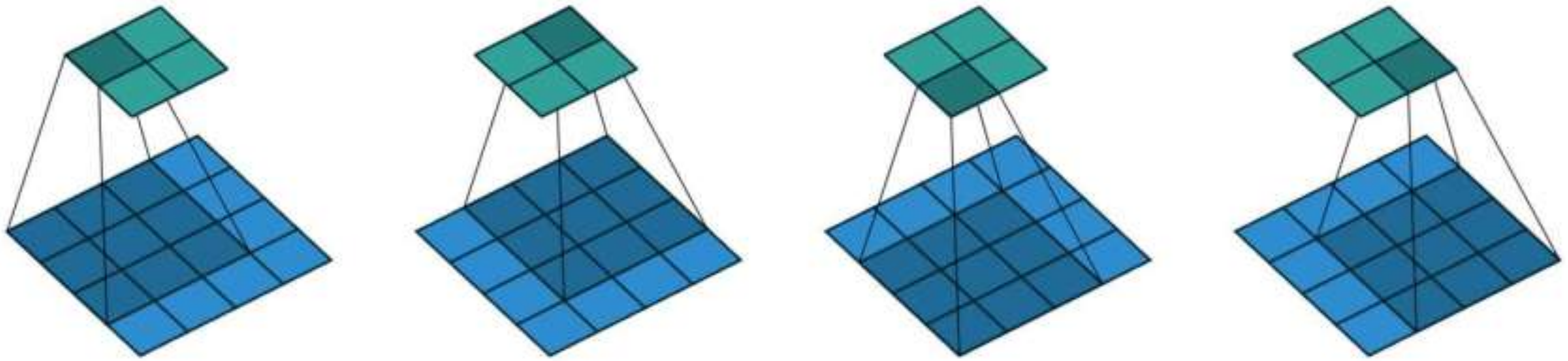
(a)

(b)

(c)

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

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).

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

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

Convolution

```
conv = tf.layers.Conv2D(1,[3,3],[2,2],"valid",activation=None)
w = tf.random_uniform((3,3),0,2)
conv.kernel_initializer = tf.constant_initializer(w.numpy())
conv.bias_initializer = tf.constant_initializer(0)
print(w)
```

```
tf.Tensor(
[[0.13429713 1.9649408  0.74939513]
 [0.6982393  1.2397809  1.4247327 ]
 [1.5534406  0.1190145  1.3256309 ]], shape=(3, 3), dtype=float32)
```

Normal Convolution Forward pass

```
imgC = tf.random_uniform((1,5,5,1),0,5)
with tf.GradientTape(persistent=True) as t:
  t.watch(imgC)
  r = conv(imgC)
  # Trick to give r as gradOutput
  r_1 = tf.multiply(0.5*r,r)
print(r)
```

```
tf.Tensor(
[[[[25.535583]
   [28.583742]]

  [[23.470192]
   [20.622599]]]], shape=(1, 2, 2, 1), dtype=float32)
```

Normal Convolution Backward pass

```
# This is equivalent to conv:backward(imgC,r)
grad_input = t.gradient(r_1,imgC)
print(grad_input)
```

```
tf.Tensor(
[[[[ 3.4293556]
   [50.17591  ]
   [22.974957 ]
   [56.16536  ]
   [21.420517 ]]

  [[42.819992 ]
   [49.156643 ]
   [98.61191  ]
   [43.924065 ]
   [53.345966 ]]

  [[17.829948 ]
   [31.65853  ]
   [56.339676 ]
   [35.437576 ]
   [40.724194 ]]

  [[16.387812 ]
   [29.097897 ]
   [47.83826  ]
   [25.567505 ]
   [29.381691 ]]

  [[36.45955  ]
   [ 2.7932932]
   [63.148792 ]
   [ 2.4543884]
   [27.337954 ]]]], shape=(1, 5, 5, 1), dtype=float32)
```

Convolution

```
trancon = tf.layers.Conv2DTranspose(1,[3,3],[2,2],"valid",activation=None)
trancon.bias_initializer = tf.constant_initializer(0)
trancon.kernel_initializer = tf.constant_initializer(w.numpy())

imgC_1 = trancon(r)
print(imgC_1)
```

```
tf.Tensor(
[[[[ 3.4293556]    [[42.819992 ]
   [50.17591  ]     [49.156643 ]
   [22.974957 ]     [98.61191  ]
   [56.16536  ]     [43.924065 ]
   [21.420517 ]]    [53.345966 ]]

  [[17.829948 ]    [[16.387812 ]
   [31.65853  ]     [29.097897 ]
   [56.339676 ]     [47.83826  ]
   [35.437576 ]     [25.567505 ]
   [40.724194 ]]    [29.381691 ]]

                   [[36.45955  ]
                    [ 2.7932932]
                    [63.148792 ]
                    [ 2.4543884]
                    [27.337954 ]]]],

shape=(1, 5, 5, 1), dtype=float32)
```

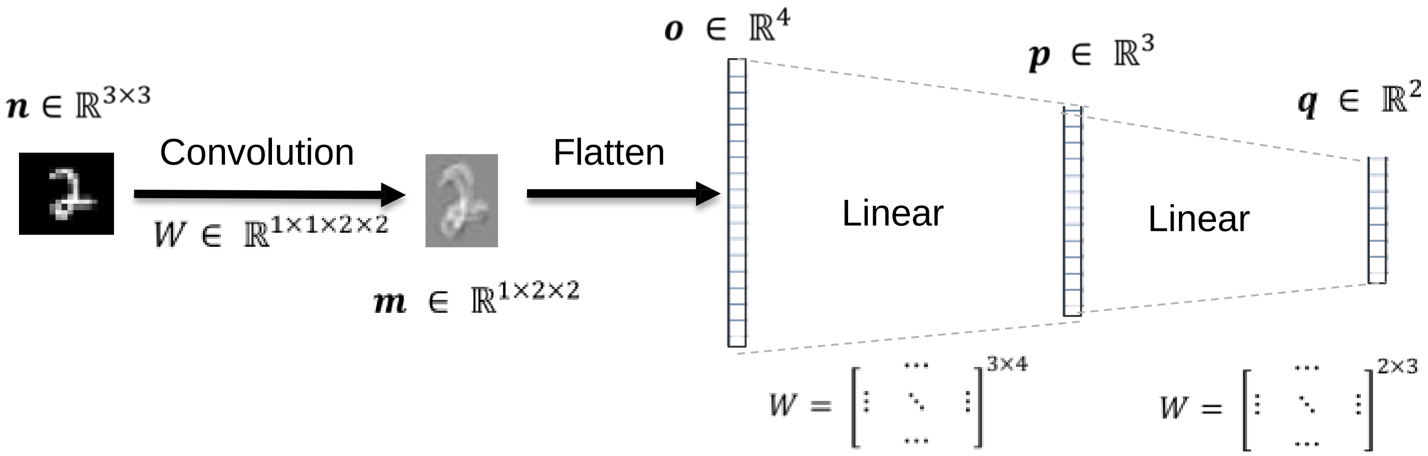
Forward pass is same as backward pass of normal convolution

Normal Convolution Backward pass

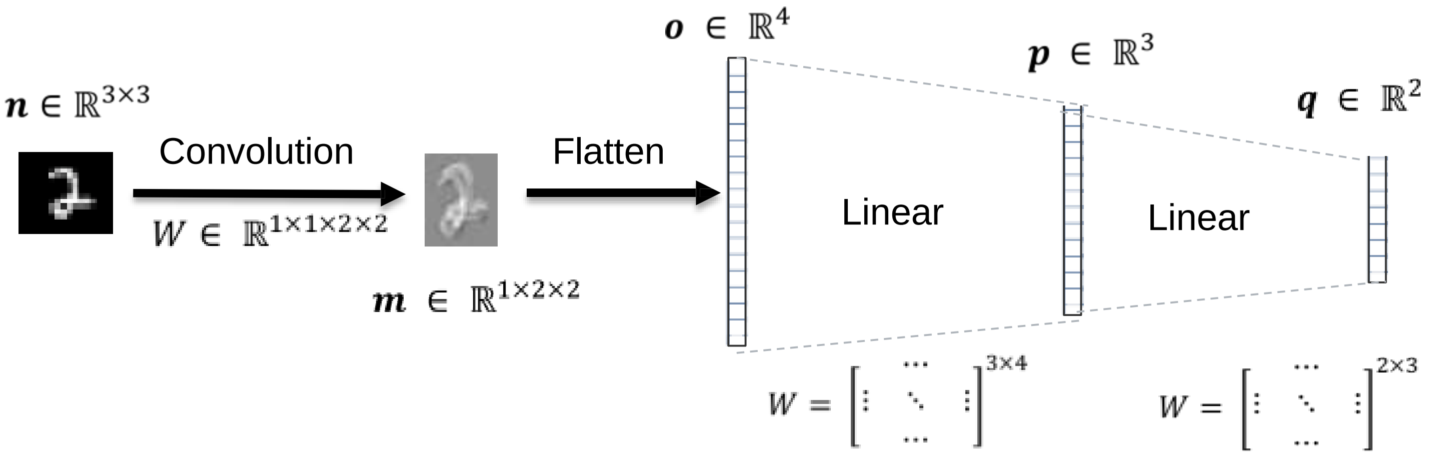
```
# This is equivalent to conv:backward(imgC,r)
grad_input = t.gradient(r_1,imgC)
print(grad_input)
```

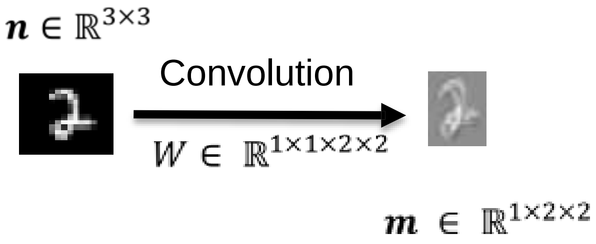
```
tf.Tensor(
[[[[ 3.4293556]    [[42.819992 ]
   [50.17591  ]     [49.156643 ]
   [22.974957 ]     [98.61191  ]
   [56.16536  ]     [43.924065 ]
   [21.420517 ]]    [53.345966 ]]

  [[17.829948 ]    [[16.387812 ]
   [31.65853  ]     [29.097897 ]
   [56.339676 ]     [47.83826  ]
   [35.437576 ]     [25.567505 ]
   [40.724194 ]]    [29.381691 ]]

                   [[36.45955  ]
                    [ 2.7932932]
                    [63.148792 ]
                    [ 2.4543884]
                    [27.337954 ]]]],

shape=(1, 5, 5, 1), dtype=float32)
```

Everything is a Convolution!

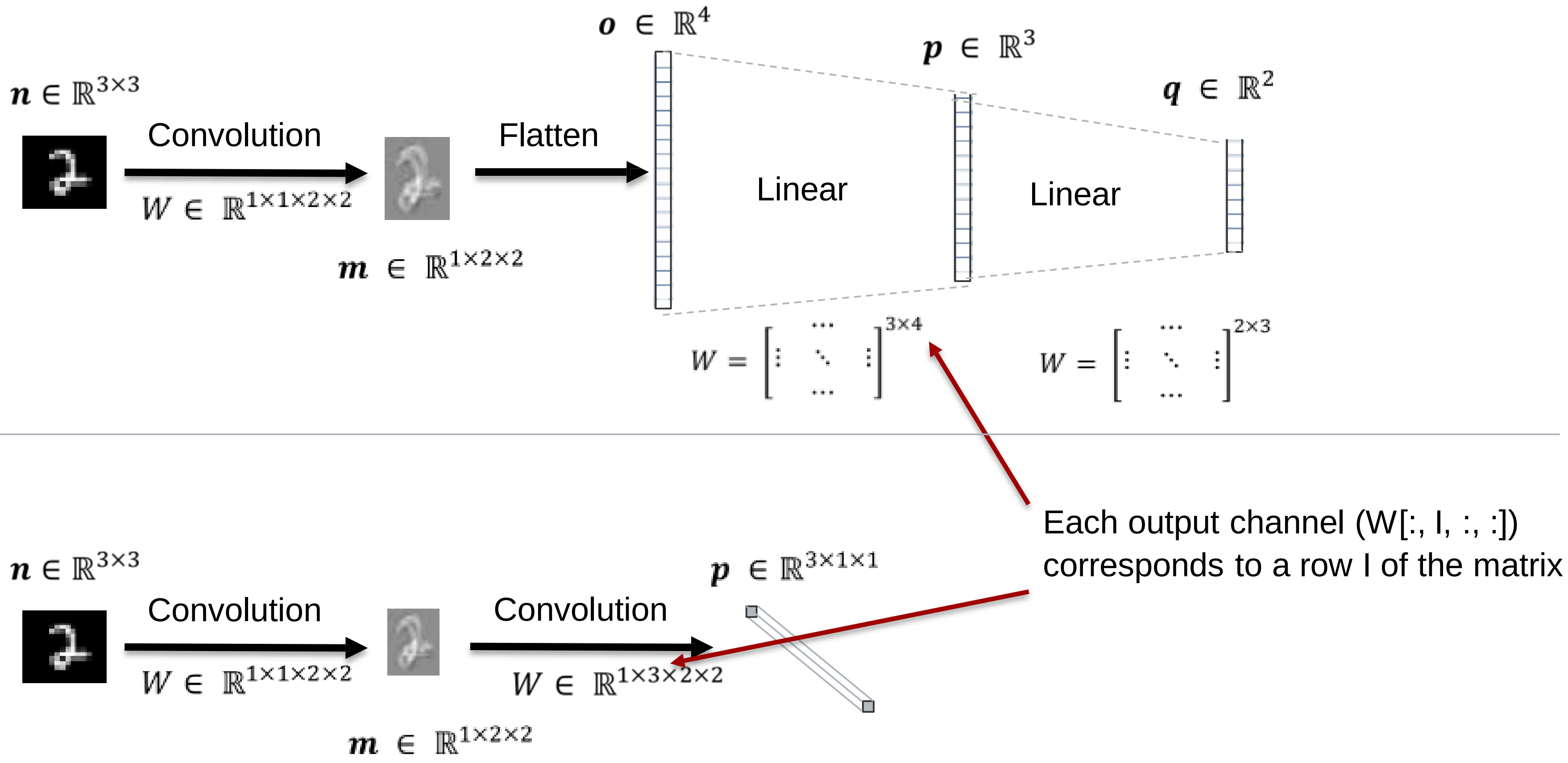


Everything is a Convolution!

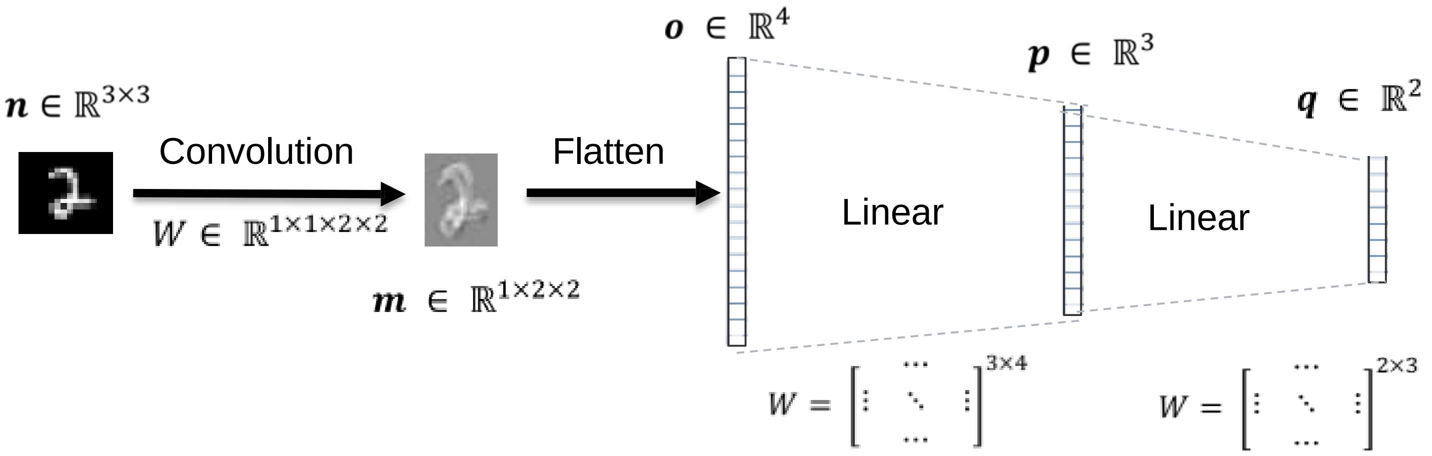


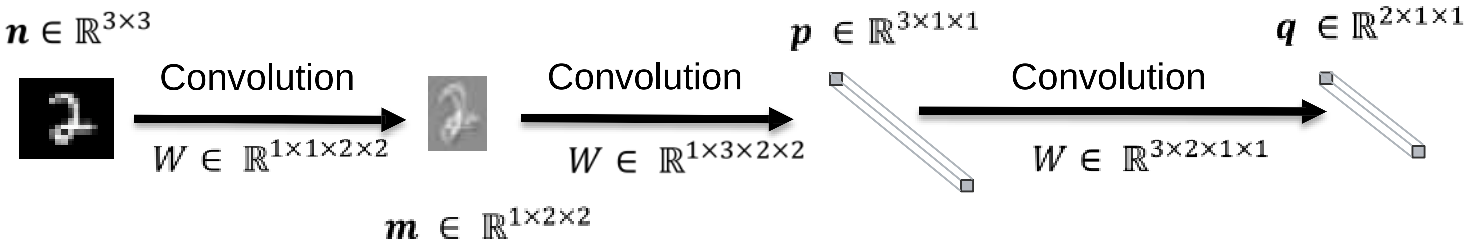


Everything is a Convolution!



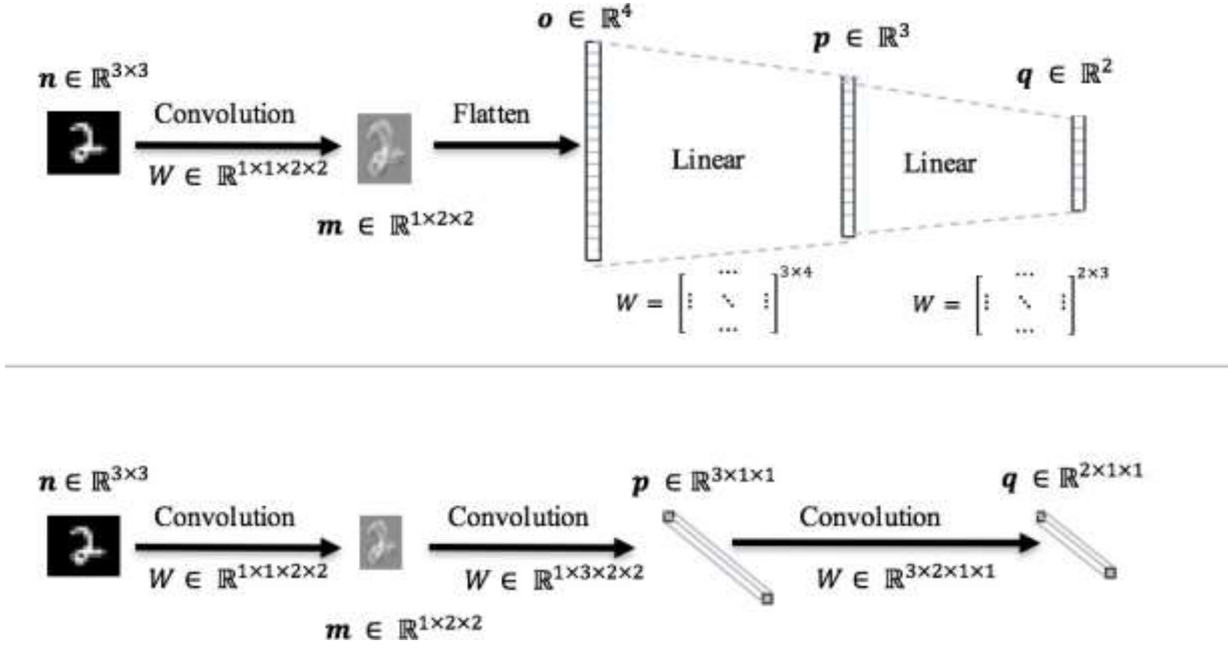
Everything is a Convolution!





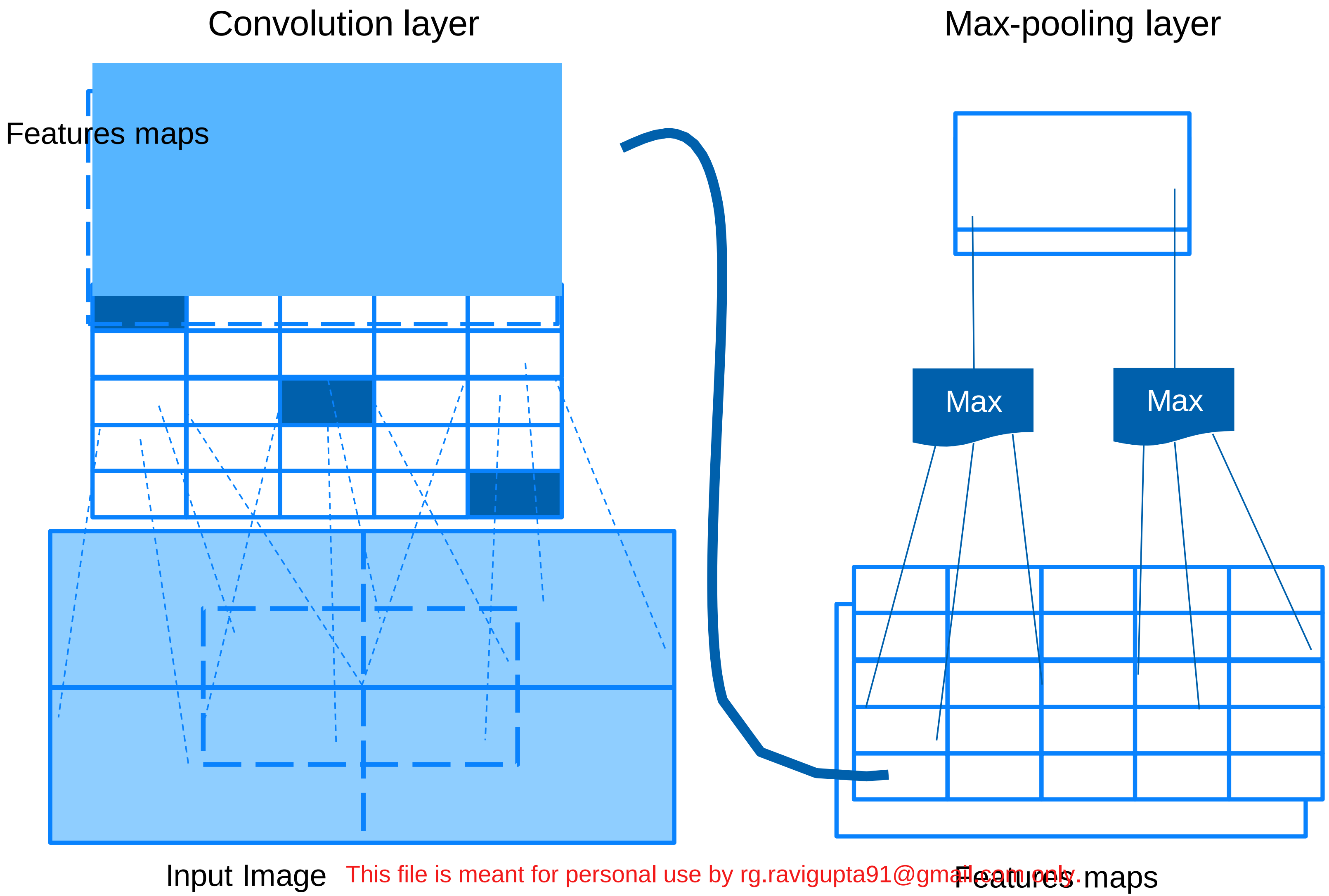
TL;DR: Converting Linear to Convolution

- 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!



Max Pooling

Pooling layer



Pooling in action

- Pooling doesn't change the depth. It only affects the length and the width of the input.
- It introduces no parameters.

Max Pooling

38	20	77	33
36	49	12	48
26	65	100	40
90	31	43	18

2x2
Pool size

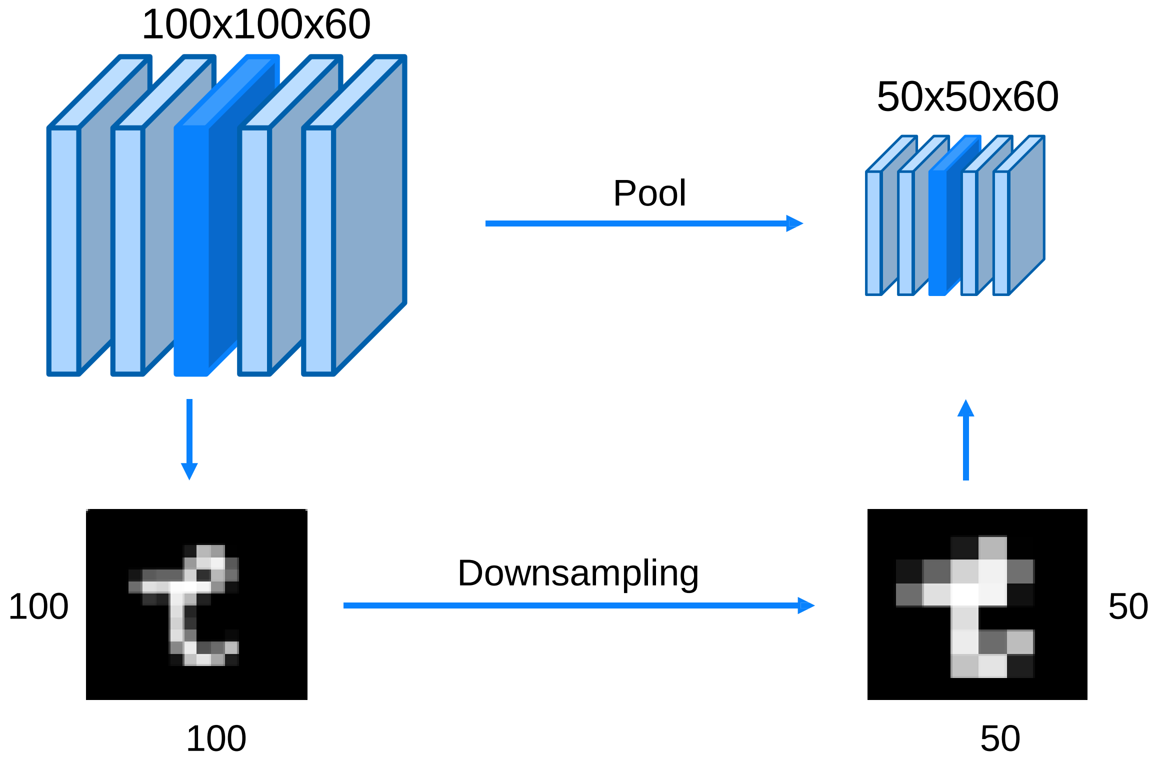
49	77
90	100

Average Pooling

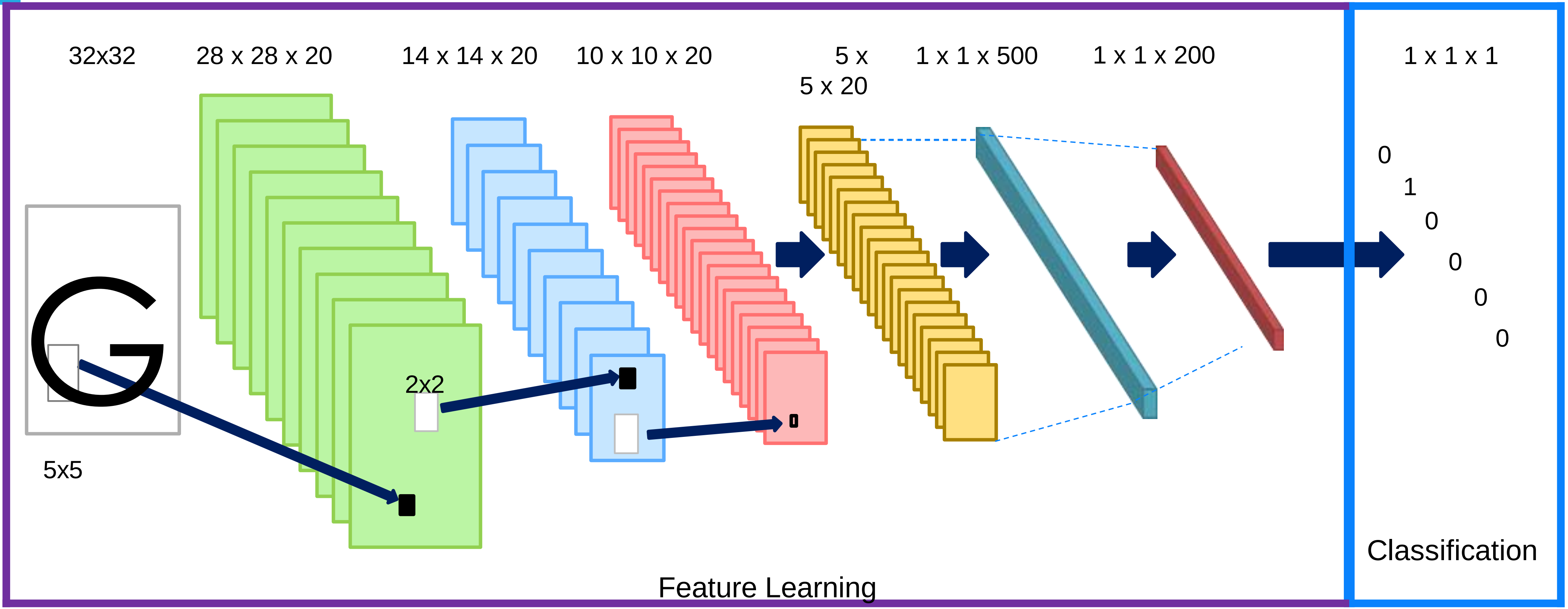
38	20	77	33
36	49	12	48
26	65	100	40
90	31	43	18

2x2
Pool size

36	43
53	50

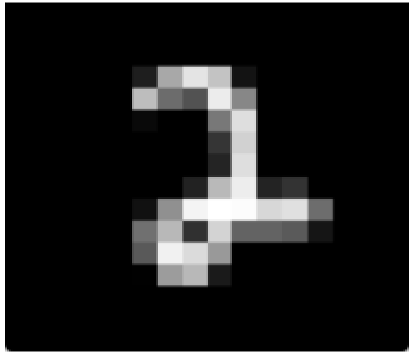


Convolution Network vs. Plain Neural Network

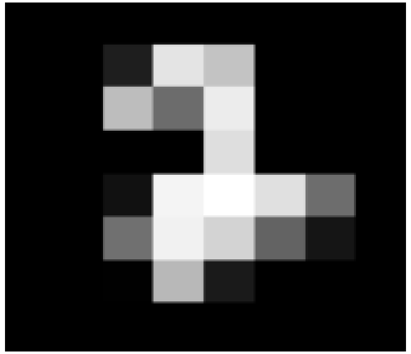


Input Image	**CONV Layer**	**POOL Layer**	**CONV Layer**	**POOL Layer**	**FC**	**FC**	Output layer
1 Map	**Filters = 20**	**Kernel 2x2**	**Filters = 20**	**Kernel 2x2**	**Layer**	**Layer**	Fully
Neurons 1024	**Kernel 5x5**	**Max Pool**	**Kernel 5x5**	**Max Pool**			Connected

Pooling (Max Pooling)

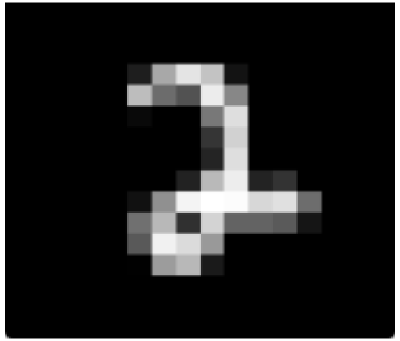


$n \in \mathbb{R}^{4\times4} \longrightarrow$ Max Pooling $\longrightarrow m \in \mathbb{R}^{2\times2}$

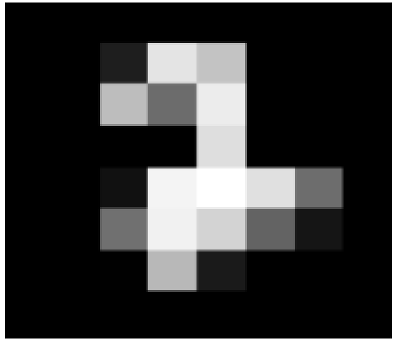


Pooling (Max Pooling) in Tensorflow

$$\frac{\partial L}{\partial m} \cdot \frac{\partial m}{\partial n} \in \mathbb{R}^{1\times(4\times4)} \qquad \frac{\partial m}{\partial n} \in \mathbb{R}^{(2\times2)\times(4\times4)} \qquad \frac{\partial L}{\partial m} \in \mathbb{R}^{1\times(2\times2)}$$



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



```
n = tf.random_normal((1,4,4,1))
pool = tf.layers.MaxPooling2D([2,2],[2,2],"valid")
nextgrad = tf.ones((1,2,2,1))
with tf.GradientTape(persistent=True) as t:
  t.watch(n)
  m = pool(n)
  m1 = tf.multiply(nextgrad,m)
print(n)
```

```
tf.Tensor(
[[[[ 0.58009243]        [[ 0.53753567]
   [ 1.1895669 ]         [ 0.10521288]
   [ 2.6777048 ]         [-0.43827865]
   [ 0.15597591]]        [-0.7727399 ]]

  [[-0.94100916]        [[-0.42974523]
   [-0.7437242 ]         [-1.669584  ]
   [-1.0578039 ]         [-0.92768306]
   [-0.30651447]]        [-0.19861951]]]],

  shape=(1, 4, 4, 1), dtype=float32)
```

```
print(m)
```

```
tf.Tensor(
[[[[ 1.1895669 ]
   [ 2.6777048 ]]

  [[ 0.53753567]
   [-0.19861951]]]], shape=(1, 2, 2, 1), dtype=float32)
```

```
gradInput = t.gradient(m1,n)
print(gradInput)
```

```
gradInput = t.gradient(m1,n)
print(gradInput)
```

```
tf.Tensor(
[[[[0.]        [[1.]
   [1.]         [0.]
   [1.]         [0.]
   [0.]]        [0.]]

  [[0.]        [[0.]
   [0.]         [0.]
   [0.]         [0.]
   [0.]]        [1.]]]],

shape=(1, 4, 4, 1), dtype=float32)
```

Calculating output dimensions

Input: $(n\ x\ n\ x\ n_c)$

Filter: $(f\ x\ f\ x\ n_c)$

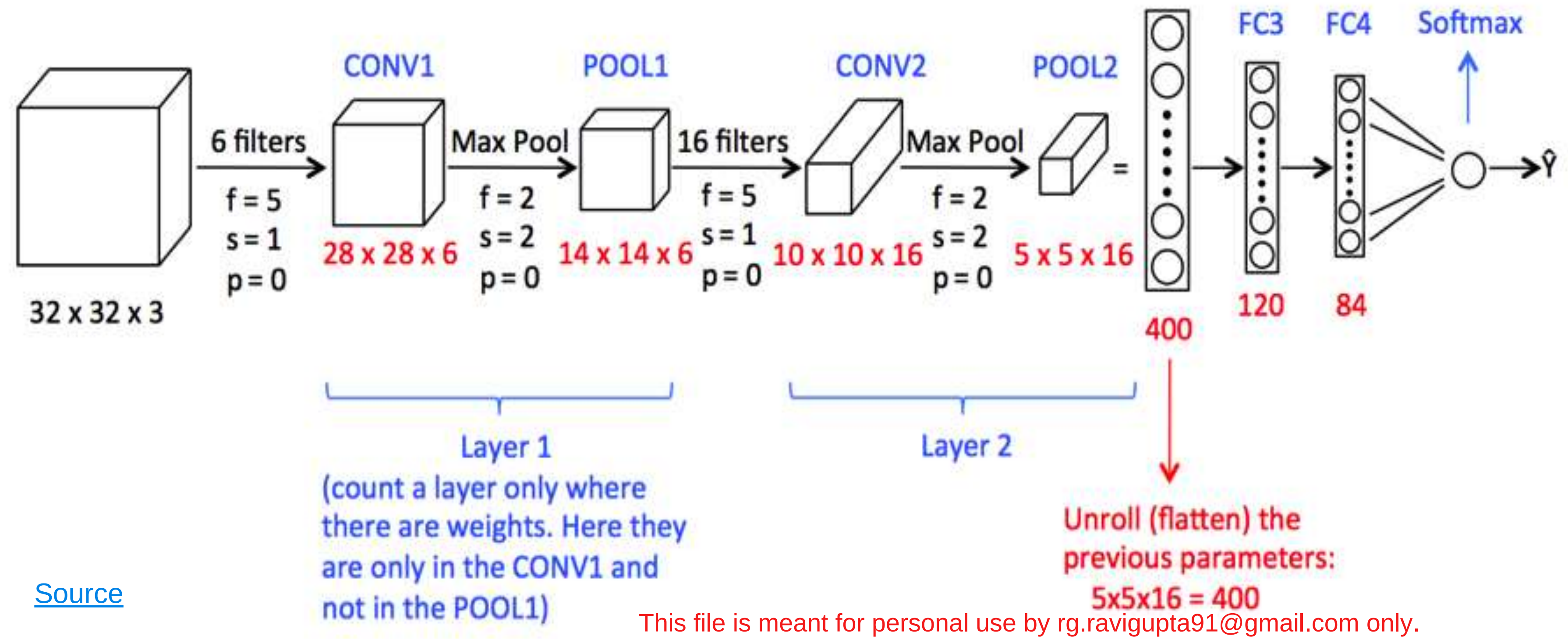
Output:

$$\left(\left[\frac{n+2p-f}{s}+1\right] x \left[\frac{n+2p-f}{s}+1\right] x\ n_c'\right)$$

f = size of filter

s = stride

p = padding



[Source]

Other Pooling layers

- No Pooling?

Striving for Simplicity: The All Convolutional Net
Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller

Thank you!