

# Deep Learning for Vision & Language

Computer Vision: CNNs and Architectures



RICE UNIVERSITY

# Color Images

- Can be viewed as tensors (3-dimensional arrays)



T =

0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

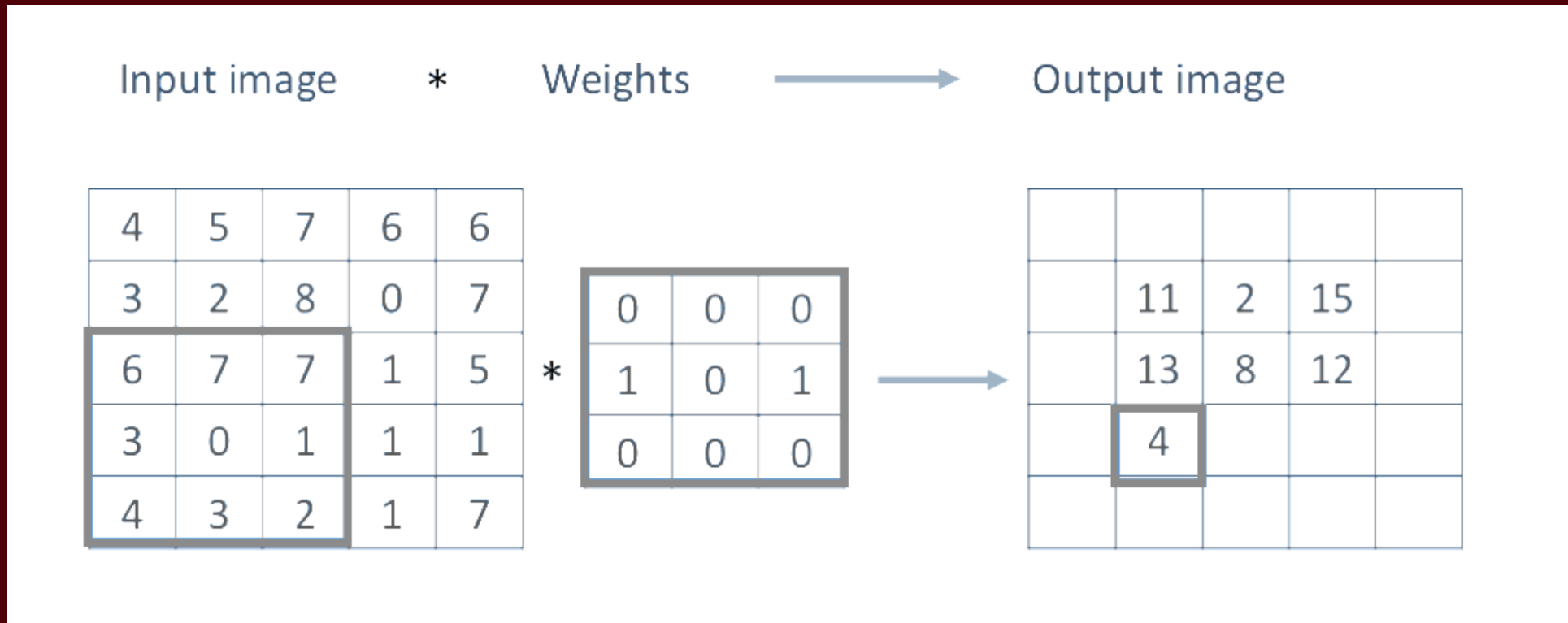
$\text{sizeof}(T) = 3 \times \text{height} \times \text{width}$

Channels are usually RGB: Red, Green, and Blue

Other color spaces: HSV, HSL, LUV, XYZ, Lab, CMYK, etc

# Most important operation for Computer Vision (\*)

- The Convolution Operation

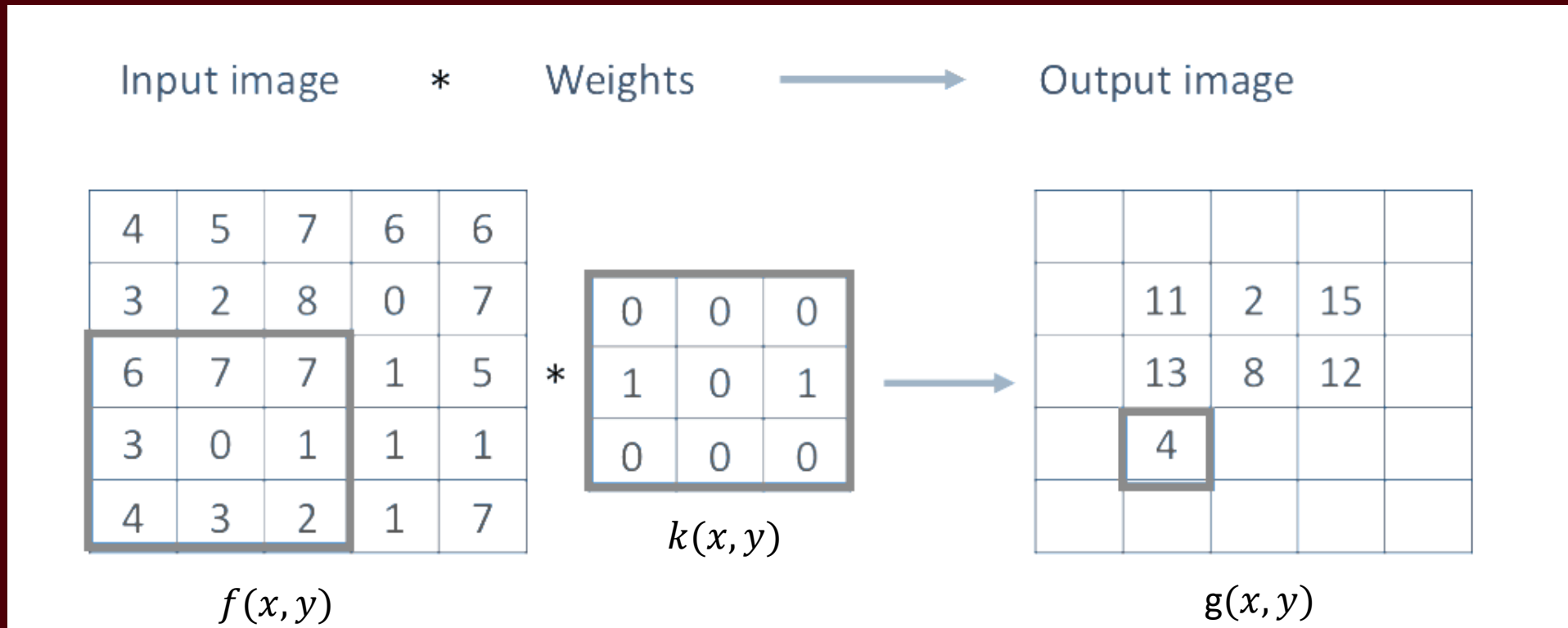


Convolutional filter  
Convolutional kernel  
Filter  
Kernel

(\*) Maybe

# Most important operation for Computer Vision (\*)

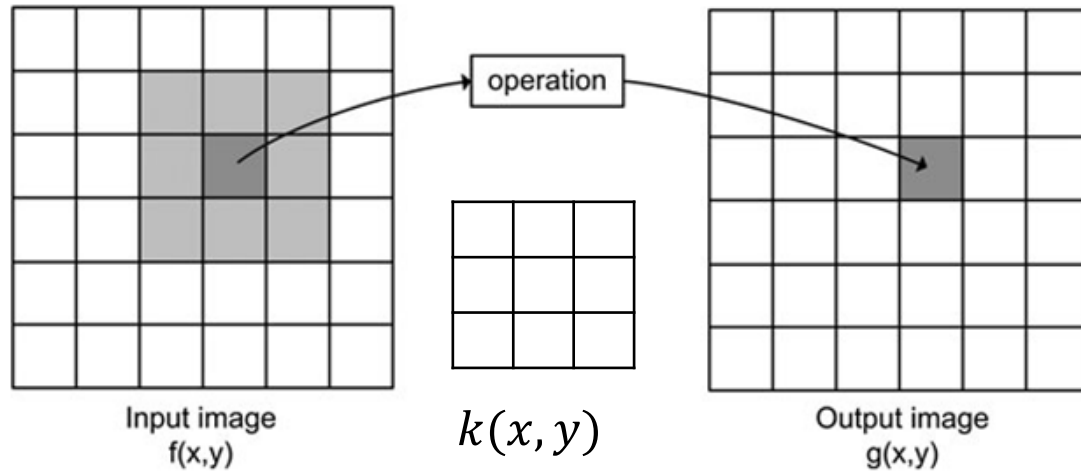
- The Convolution Operation



$$g(x, y) = \sum_v \sum_u k(u, v) f(x - u, y - v)$$

(\*) Maybe

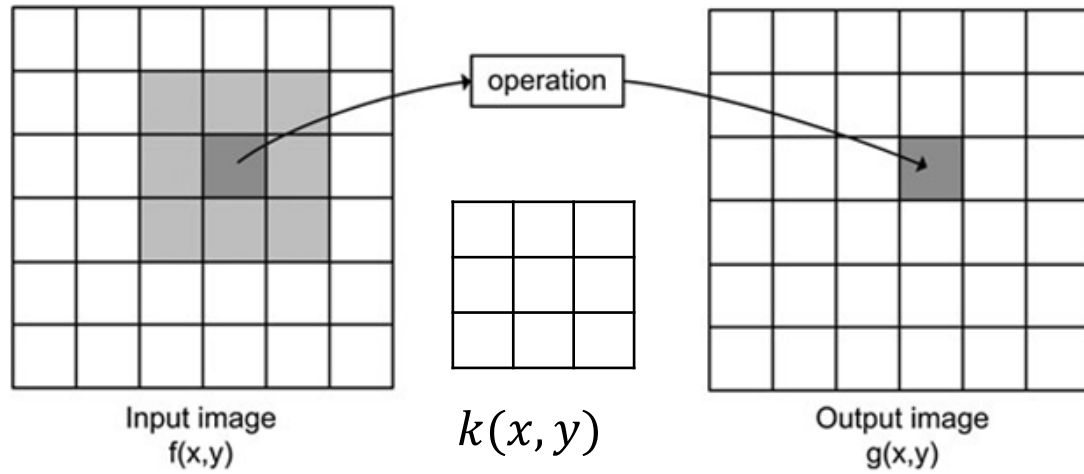
# Image filtering: Convolution operator e.g. mean filter



$$k(x,y) =$$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

# Image filtering: Convolution operator e.g. mean filter



$$k(x, y) =$$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

# Example: box filter

$g[\cdot, \cdot]$

	1	1	1
1	1	1	1
9	1	1	1

# The 2D Convolutional Layer in a Neural Network

Input image

\*

Weights



Output image

4	5	7	6	6
3	2	8	0	7
6	7	7	1	5
3	0	1	1	1
4	3	2	1	7

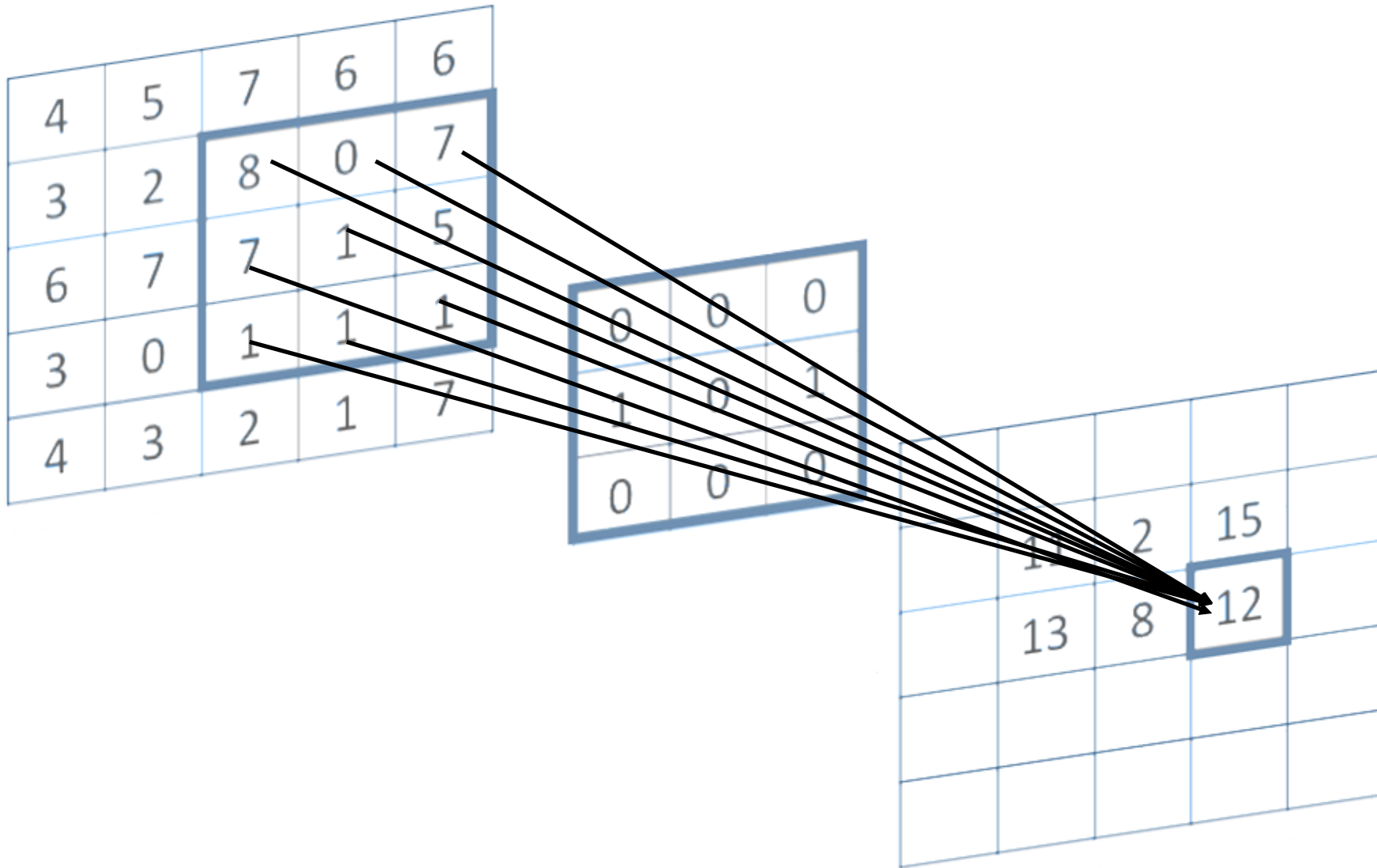
\*

0	0	0
1	0	1
0	0	0



	11	2	15	
	13	8	12	

# The 2D Convolutional Layer in a Neural Network



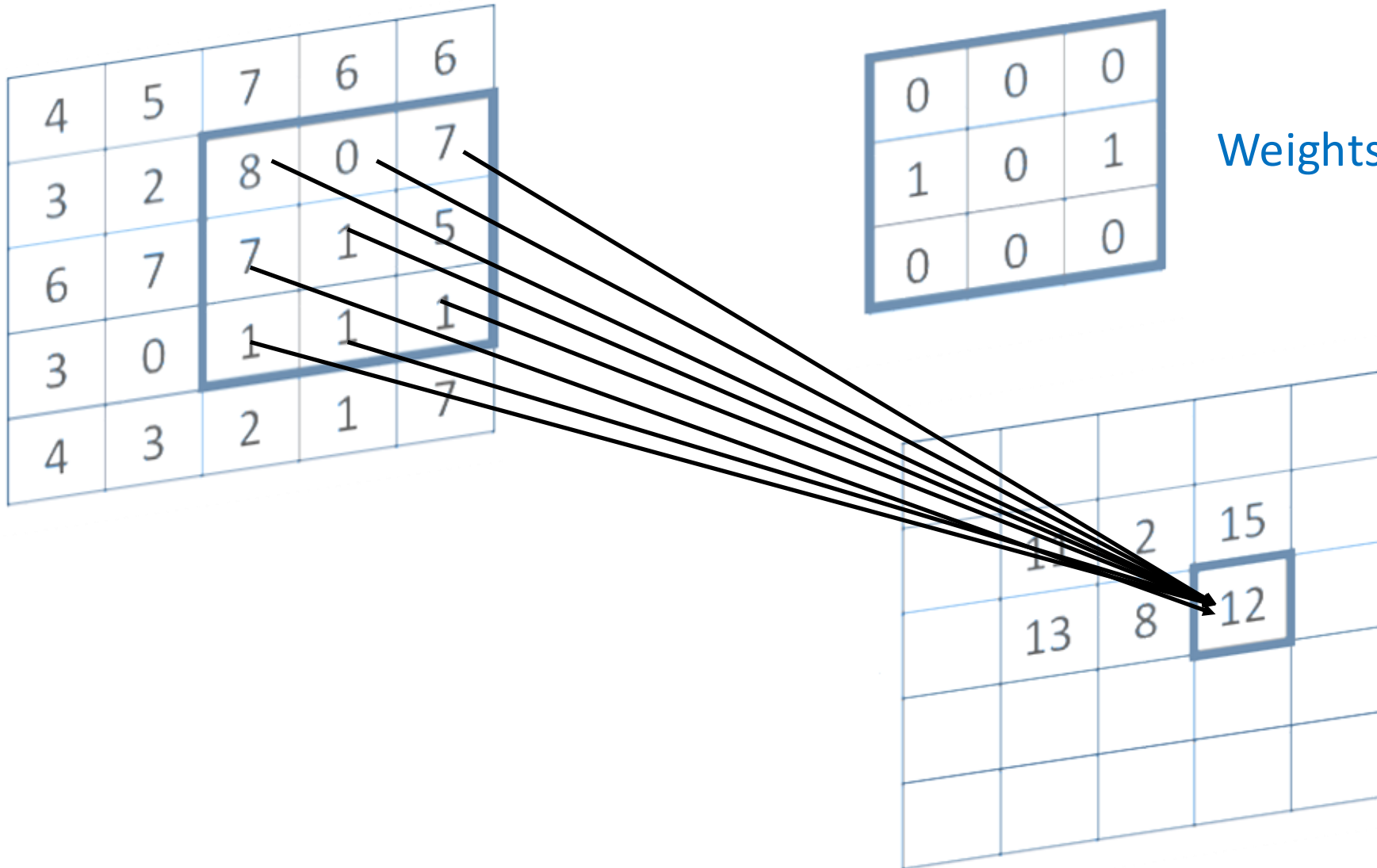
# The 2D Convolutional Layer in a Neural Network

4	5	7	6	6
3	2	8	0	7
6	7	7	1	5
3	0	1	1	1
4	3	2	1	7

0	0	0
1	0	1
0	0	0

Weights

	1	2	15	
	13	8	12	



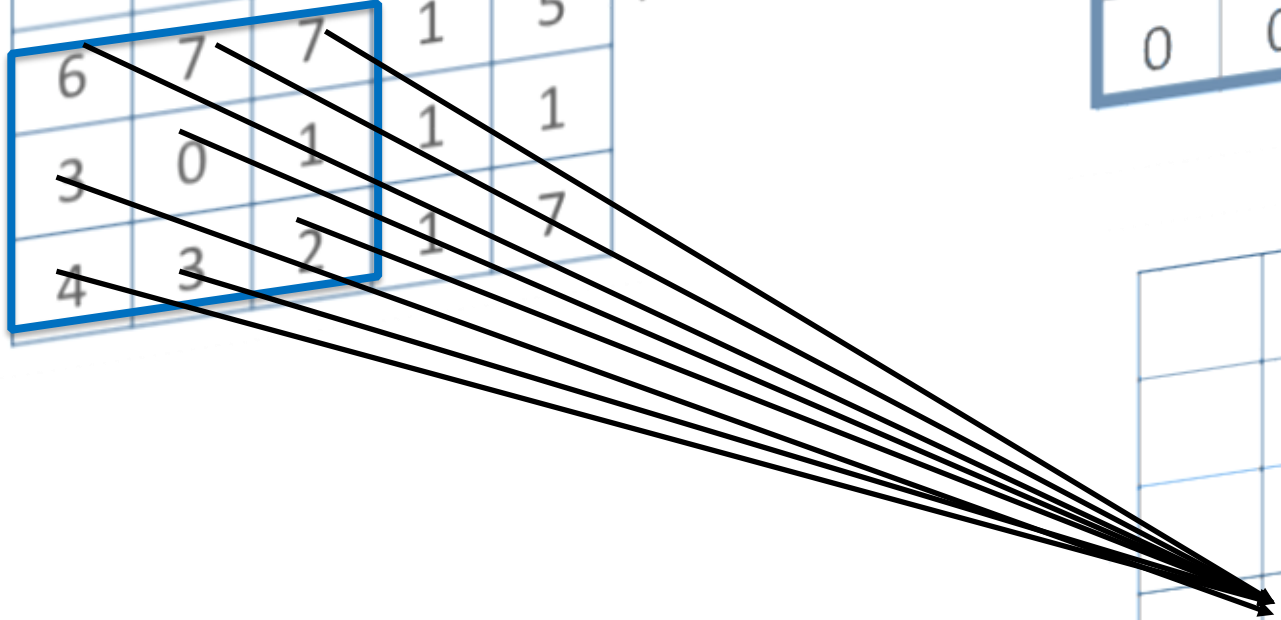
# The 2D Convolutional Layer in a Neural Network

4	5	7	6	6
3	2	8	0	7
6	7	7	1	5
3	0	1	1	1
4	3	2	1	7

0	0	0
1	0	1
0	0	0

Weights

	11	2	15	
	13	8	12	
	4			



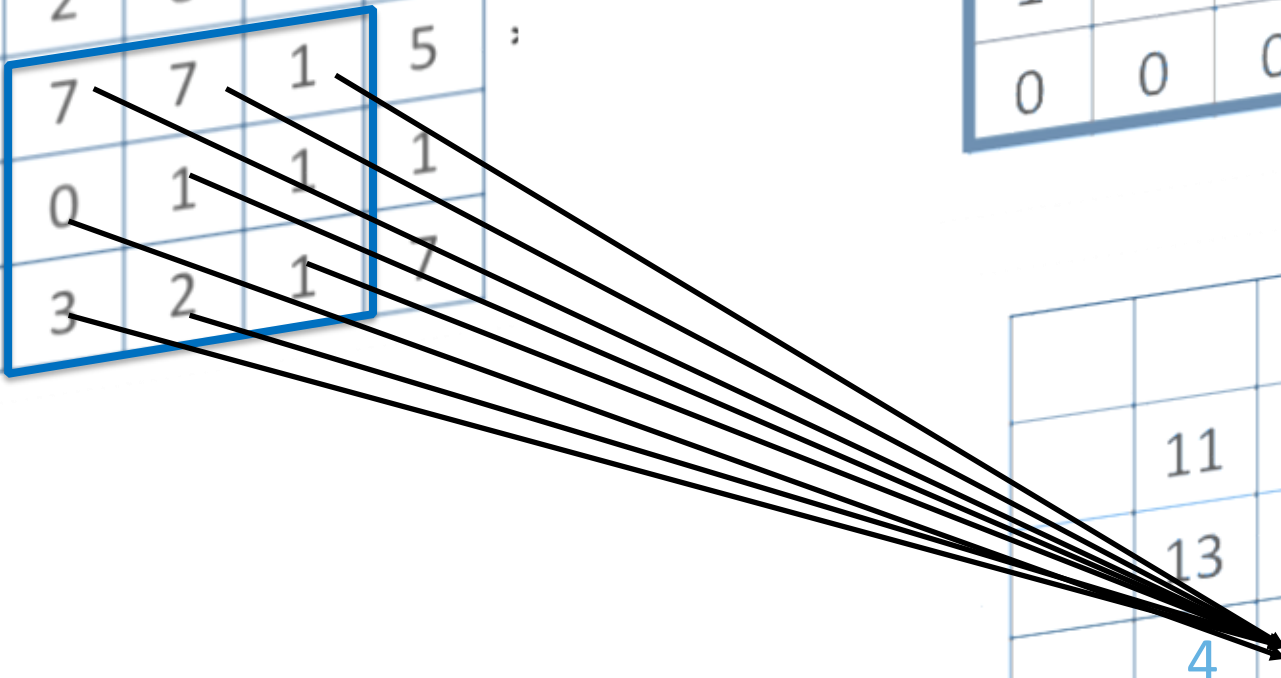
# The 2D Convolutional Layer in a Neural Network

4	5	7	6	6
3	2	8	0	7
6	7	7	1	5
3	0	1	1	1
4	3	2	1	7

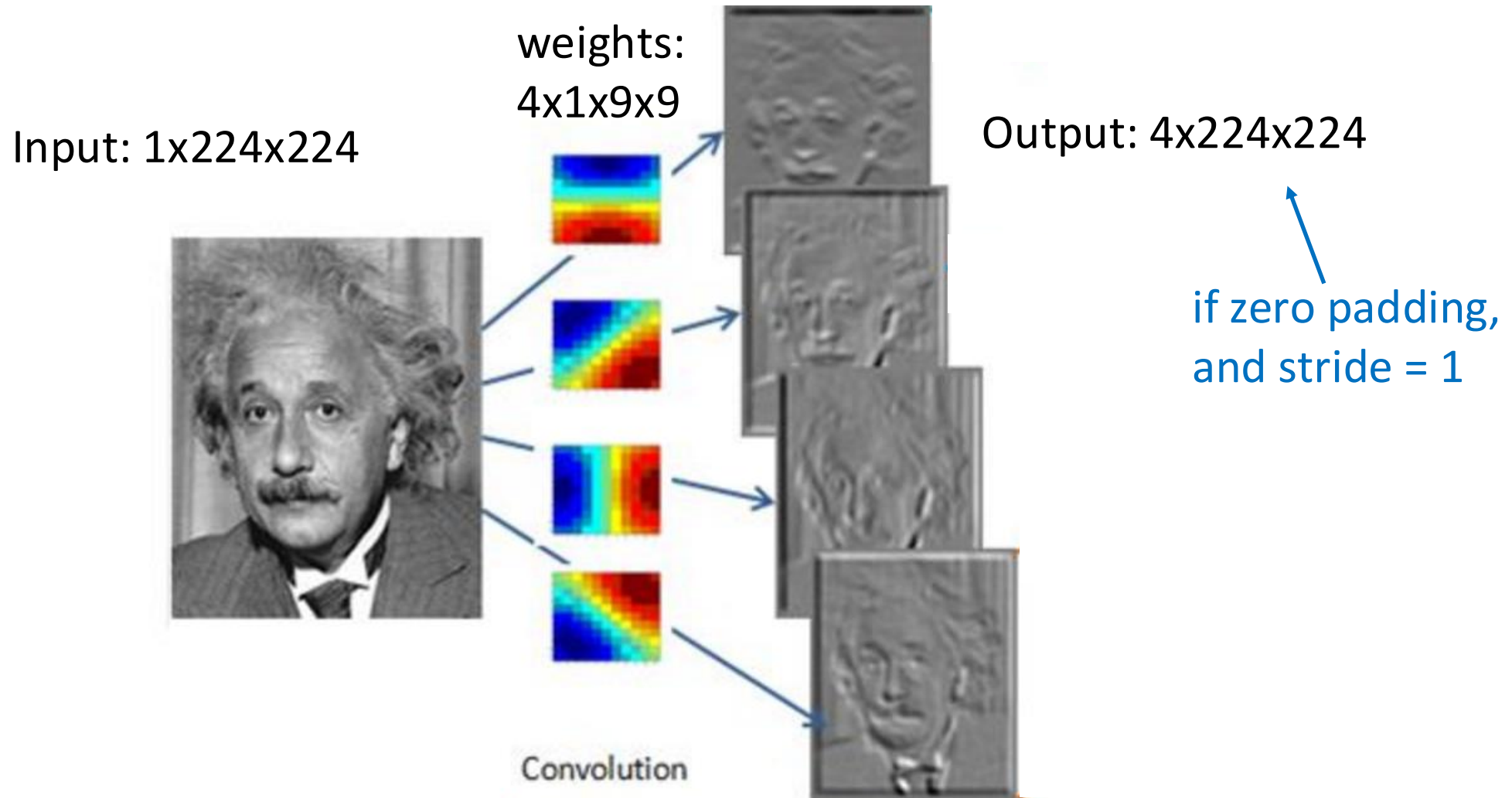
0	0	0
1	0	1
0	0	0

Weights

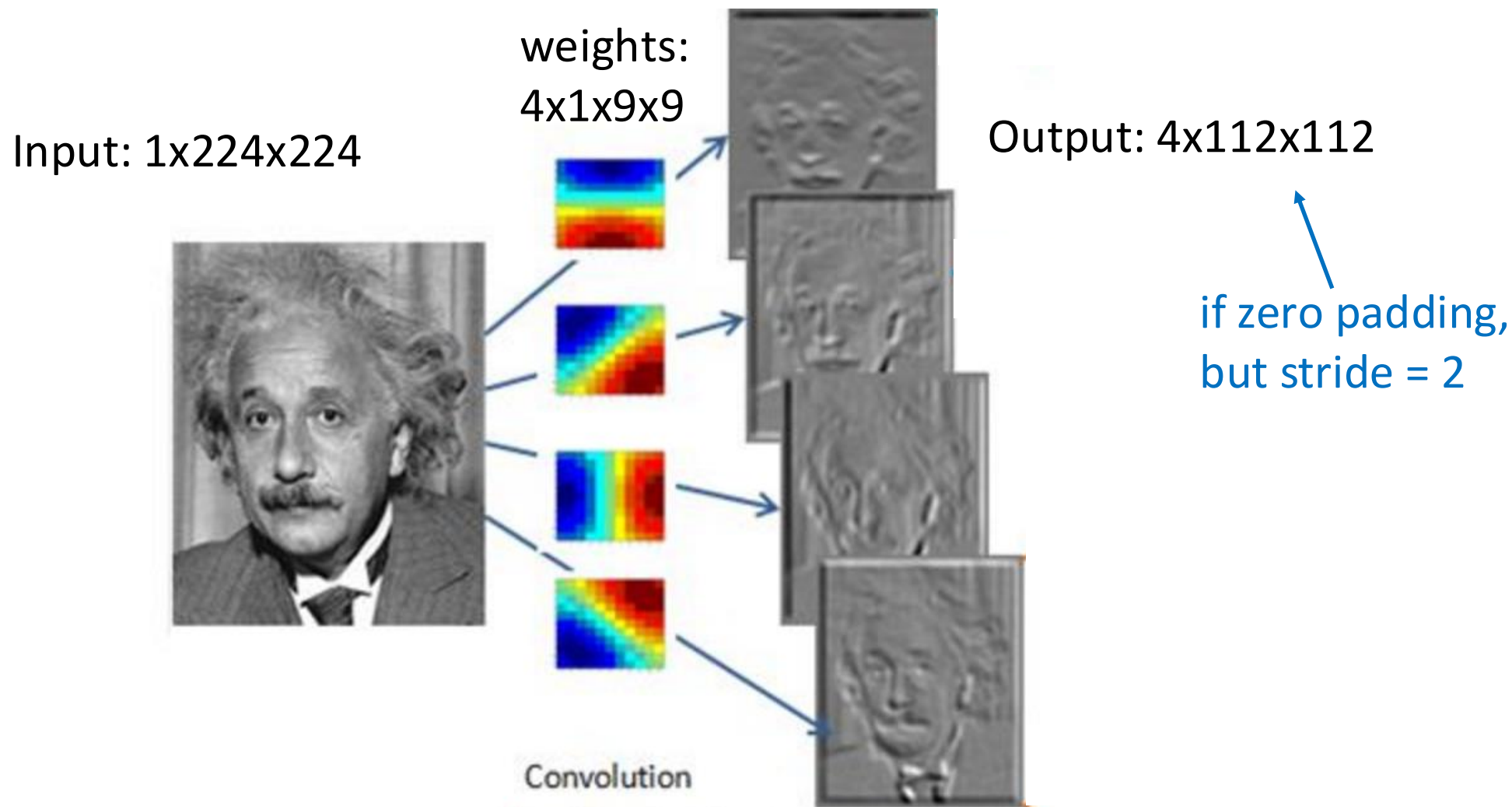
	11	2	15	
	13	8	12	
	4	1		



# Convolutional Layer (with 4 filters)

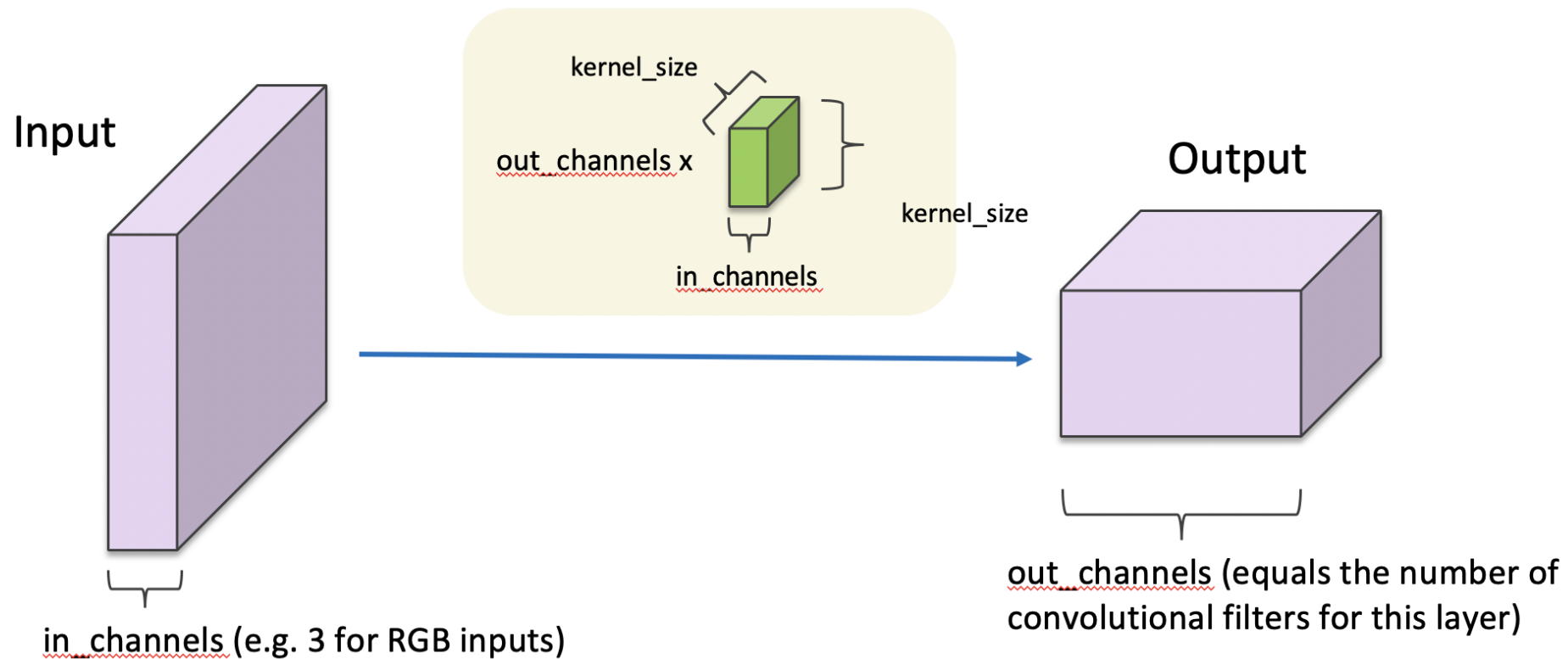


# Convolutional Layer (with 4 filters)

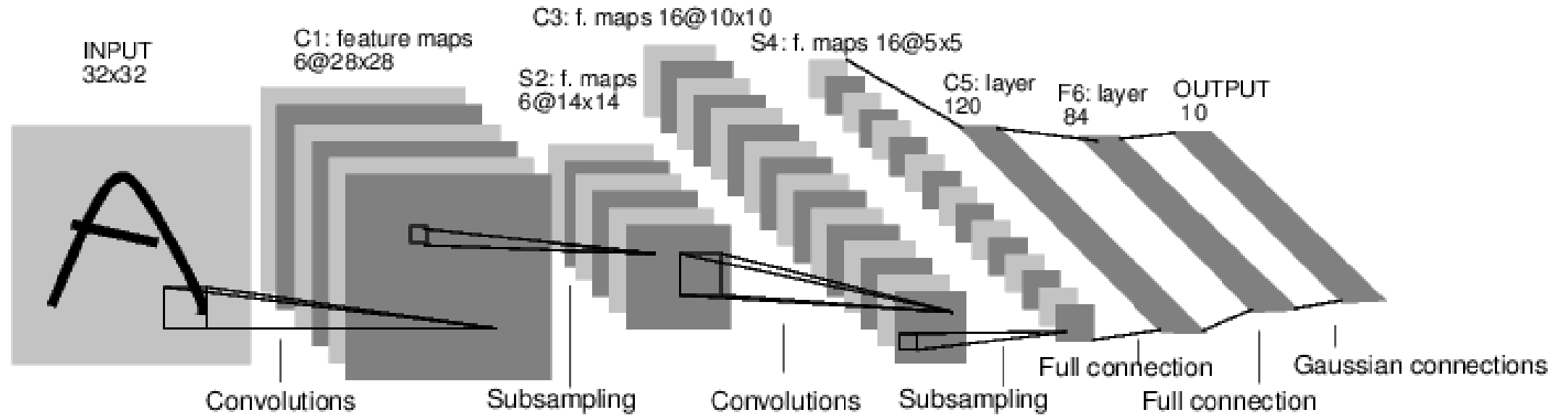


# Convolutional Layer in pytorch

```
class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,  
groups=1, bias=True) \[source\]
```



# Convolutional Network: LeNet



Yann LeCun

TITLE

[Gradient-based learning applied to document recognition](#)

Y LeCun, L Bottou, Y Bengio, P Haffner  
Proceedings of the IEEE 86 (11), 2278-2324

CITED BY

83330

YEAR

2002

# LeNet in Pytorch

```
# LeNet is French for The Network, and is taken from Yann Lecun's 98 paper  
# on digit classification http://yann.lecun.com/exdb/lenet/  
# This was also a network with just two convolutional layers.  
class LeNet(nn.Module):  
    def __init__(self):  
        super(LeNet, self).__init__()  
        # Convolutional layers.  
        self.conv1 = nn.Conv2d(3, 6, 5)  
        self.conv2 = nn.Conv2d(6, 16, 5)  
  
        # Linear layers.  
        self.fc1 = nn.Linear(16*5*5, 120)  
        self.fc2 = nn.Linear(120, 84)  
        self.fc3 = nn.Linear(84, 10)  
  
    def forward(self, x):  
        out = F.relu(self.conv1(x))  
        out = F.max_pool2d(out, 2)  
        out = F.relu(self.conv2(out))  
        out = F.max_pool2d(out, 2)  
  
        # This flattens the output of the previous layer into a vector.  
        out = out.view(out.size(0), -1)  
        out = F.relu(self.fc1(out))  
        out = F.relu(self.fc2(out))  
        out = self.fc3(out)  
        return out
```

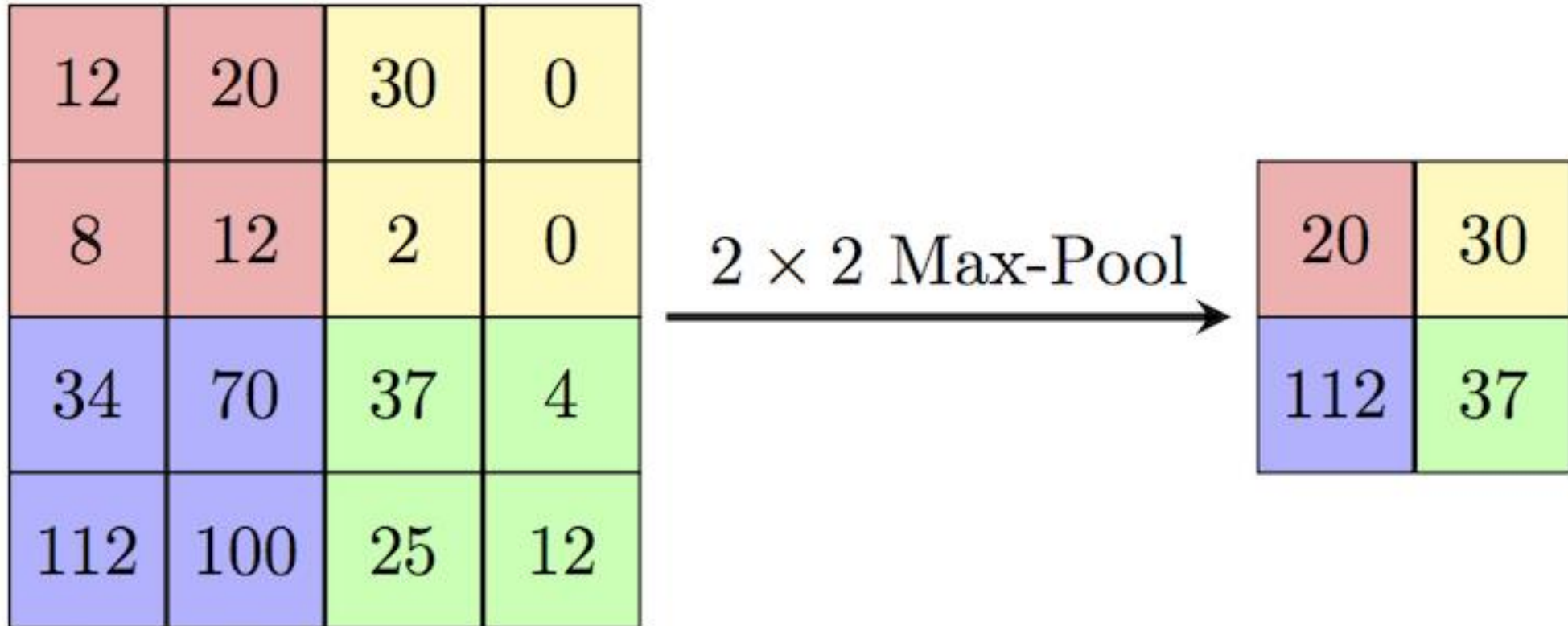
# SpatialMaxPooling Layer

4	5	7	6	6
3	2	8	0	7
6	7	7	1	5
3	0	1	1	1
4	3	2	1	7

take the max in this neighborhood

	1	8	8	
	8	8	8	

# SpatialMaxPooling Layer



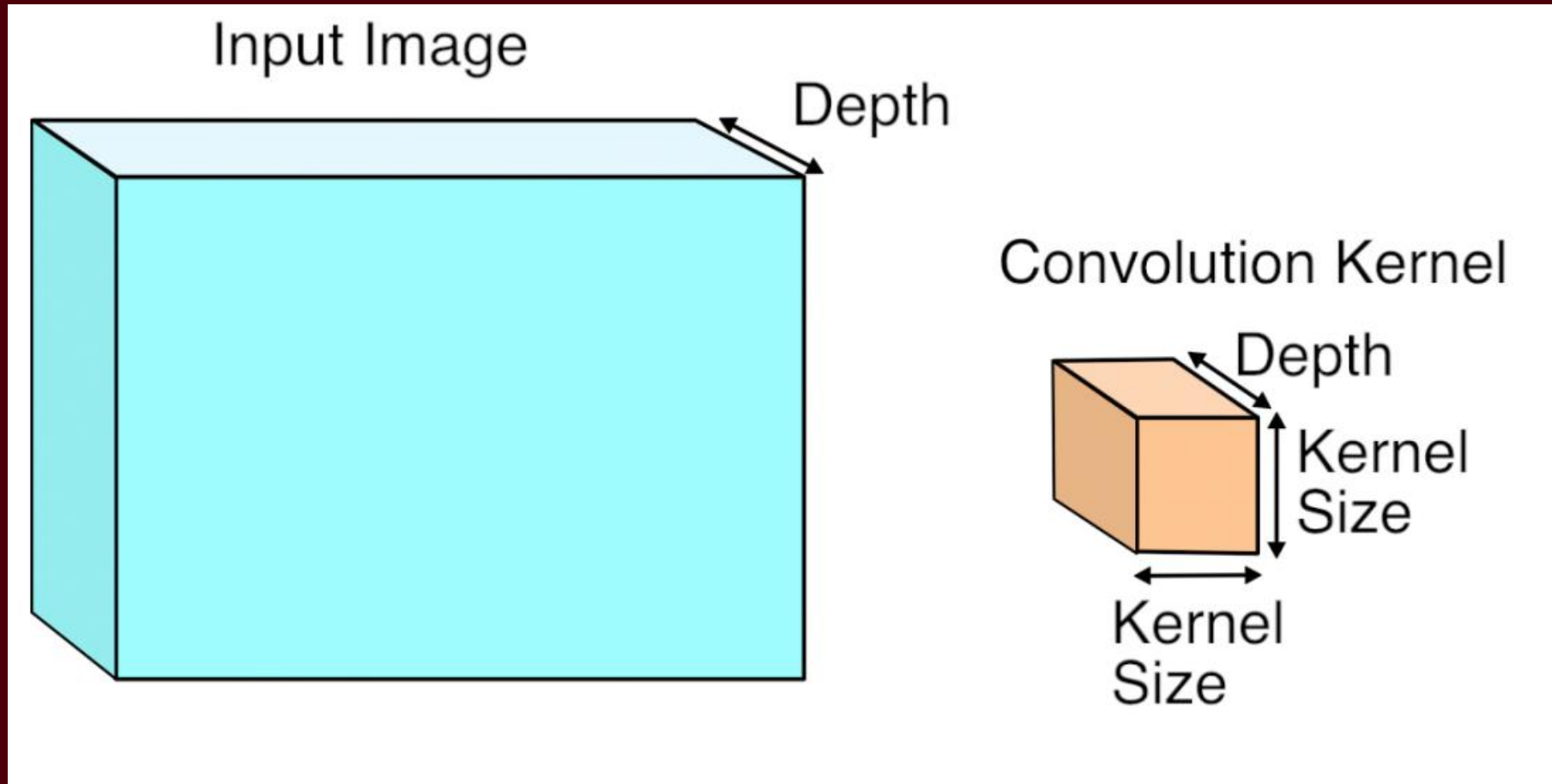
# LeNet Summary

- 2 Convolutional Layers + 3 Linear Layers
- + Non-linear functions: ReLUs or Sigmoids  
+ Max-pooling operations

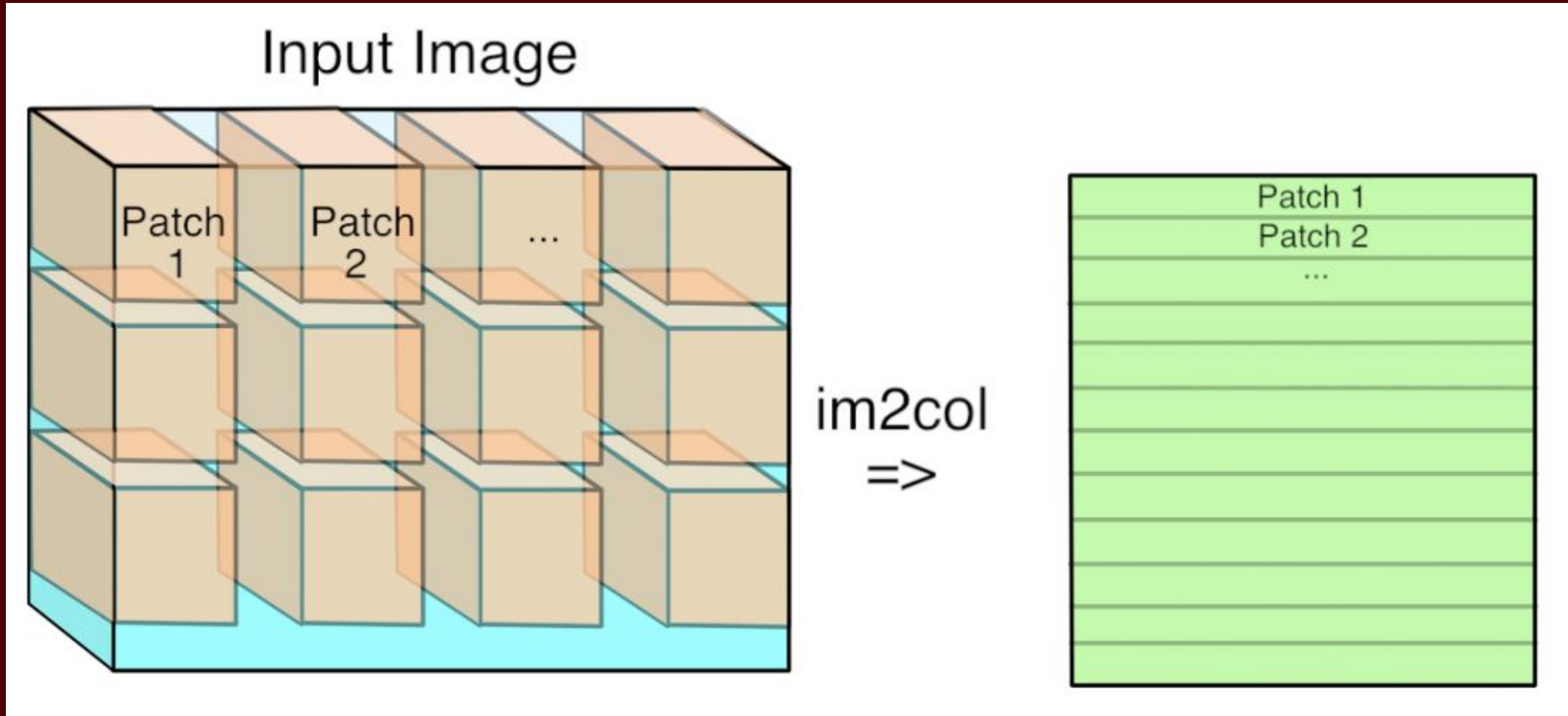
# New Architectures Proposed

- Alexnet (Krizhevsky et al NIPS 2012) [Required Reading]
- VGG (Simonyan and Zisserman 2014)
- GoogLeNet (Szegedy et al CVPR 2015)
- ResNet (He et al CVPR 2016)
- DenseNet (Huang et al CVPR 2017)

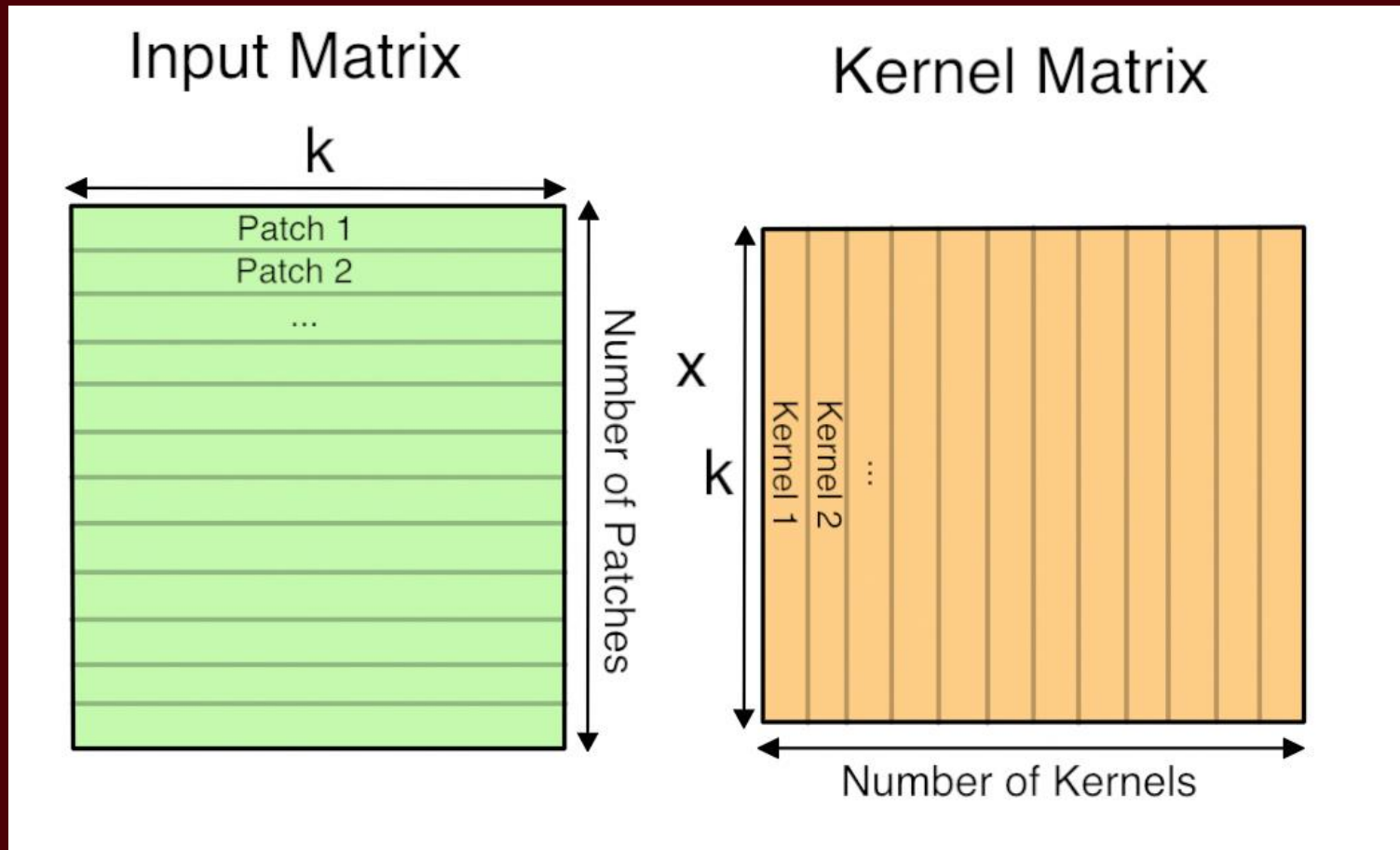
# Convolutional Layers as Matrix Multiplication



# Convolutional Layers as Matrix Multiplication



# Convolutional Layers as Matrix Multiplication



Pros?  
Cons?

# CNN Computations are Computationally Expensive

- However highly parallelizable
- GPU Computing is used in practice
- CPU Computing in fact is prohibitive for training these models

# Questions?