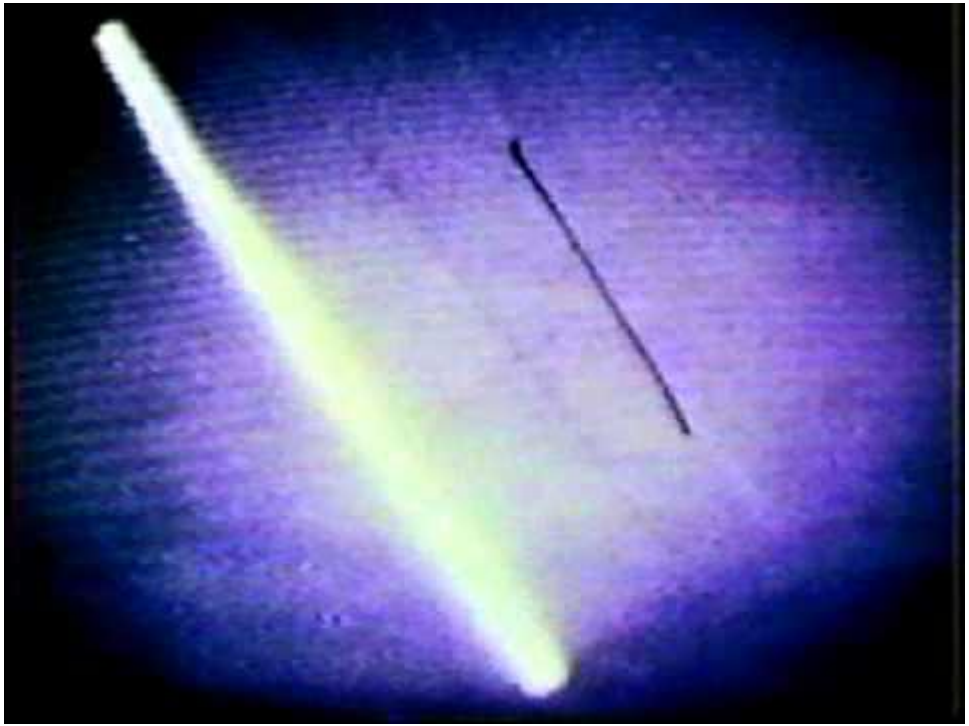

MONASH University

# Convolutional Neural Networks

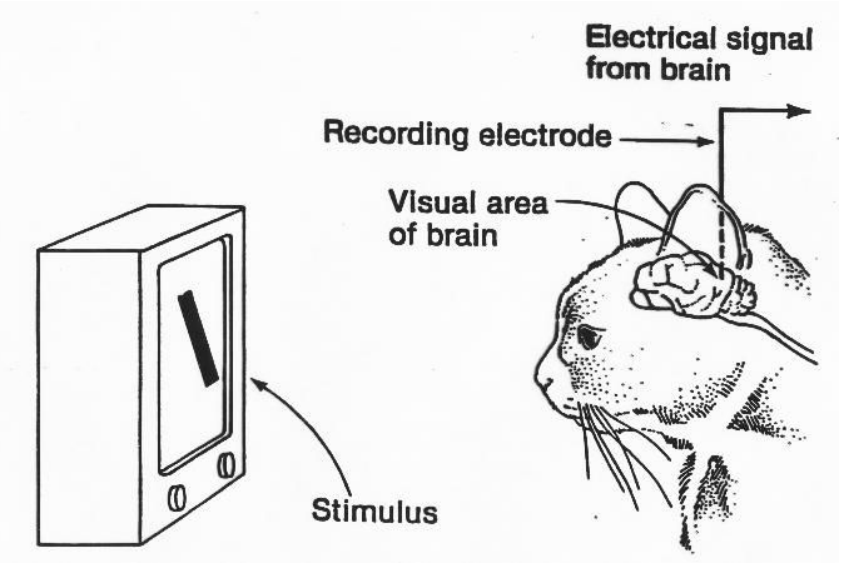
Mehrtash Harandi

# History



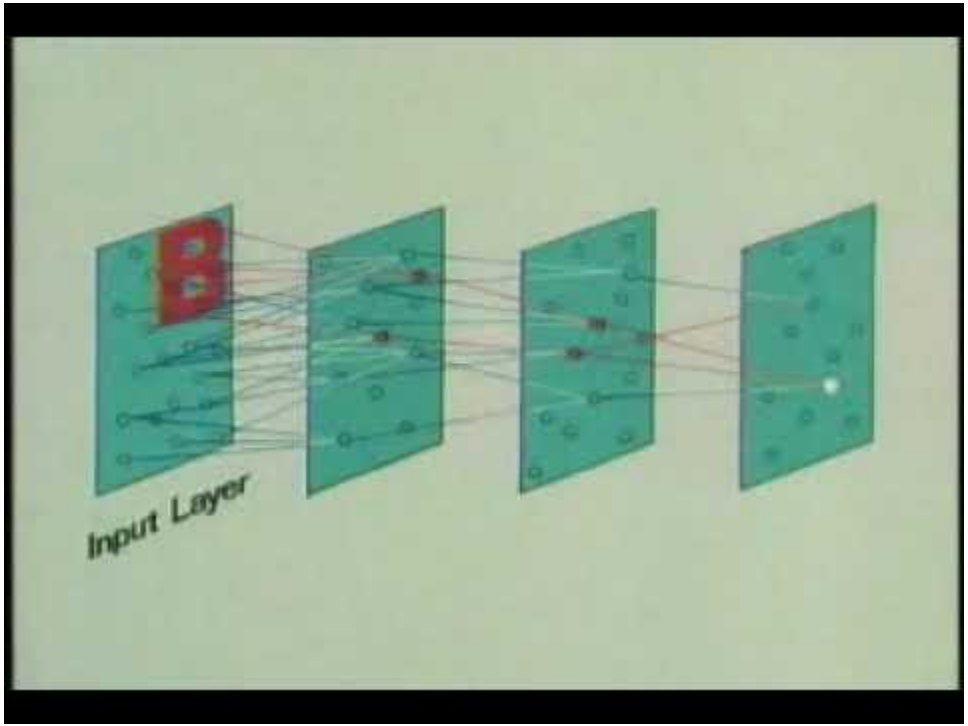
# History

Hubel and Wiesel (1962)



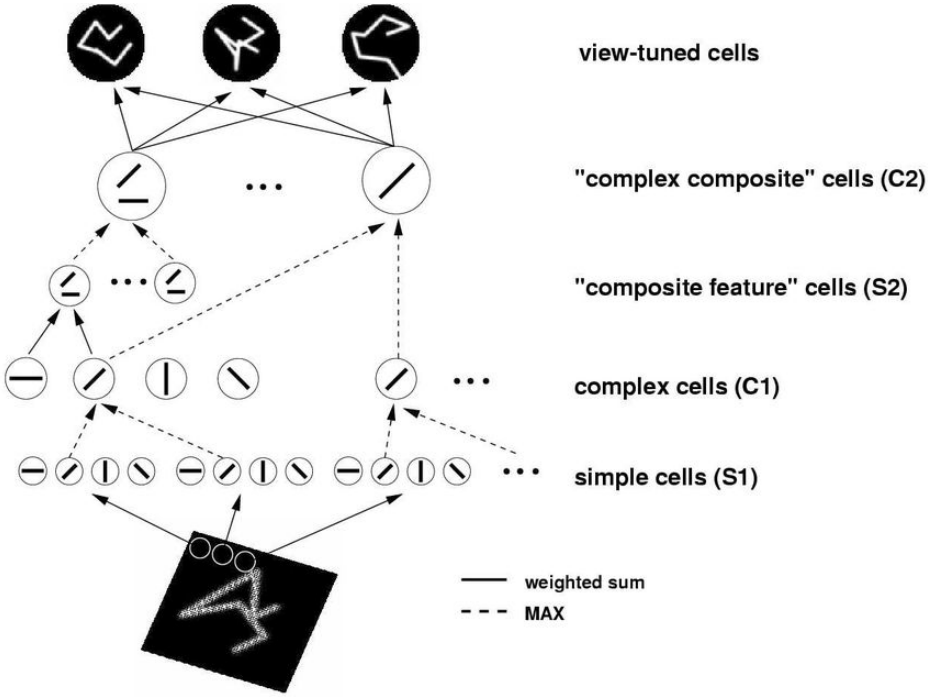
# Neocognitron

Fukushima (1979)



# HMax model

T. Poggio

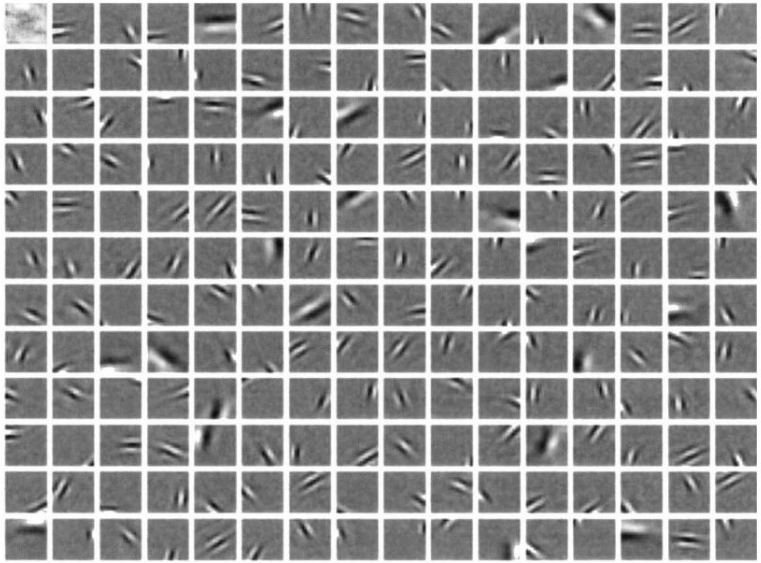


# History

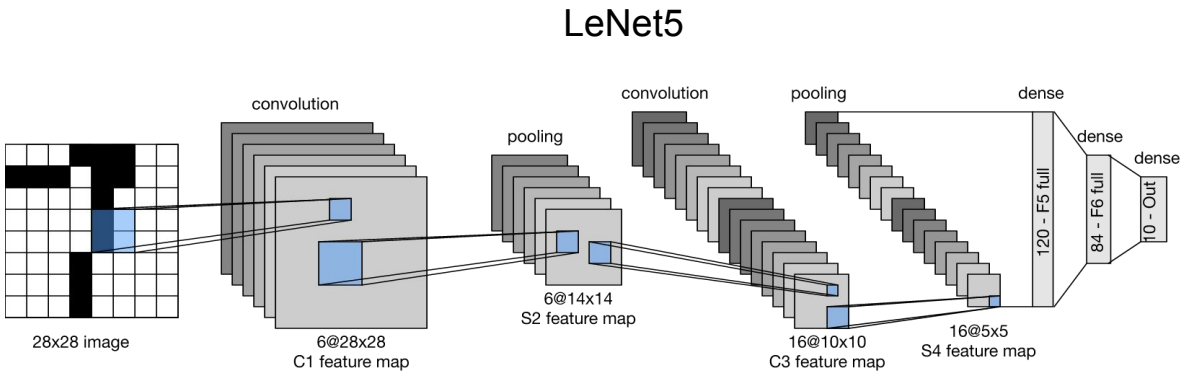
Sparsity (1996)

B. Olshausen

D. Field



# History




Yann Lecun

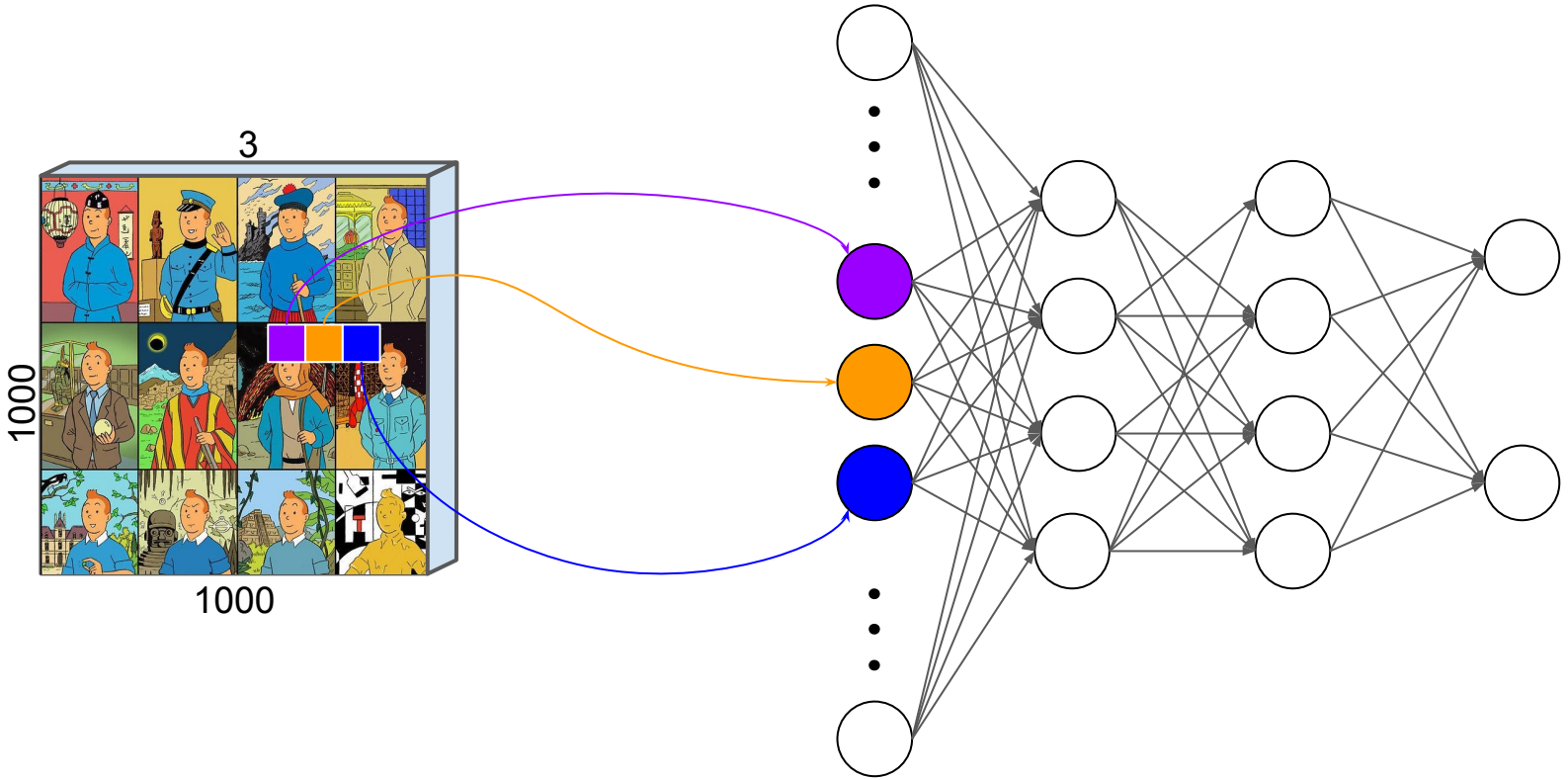
- developed at AT&T Bell Labs in 90s
- the purpose was to recognize handwritten digits in images
- the first compelling evidence that it was possible to train CNNs by backpropagation

# Why convolutions?

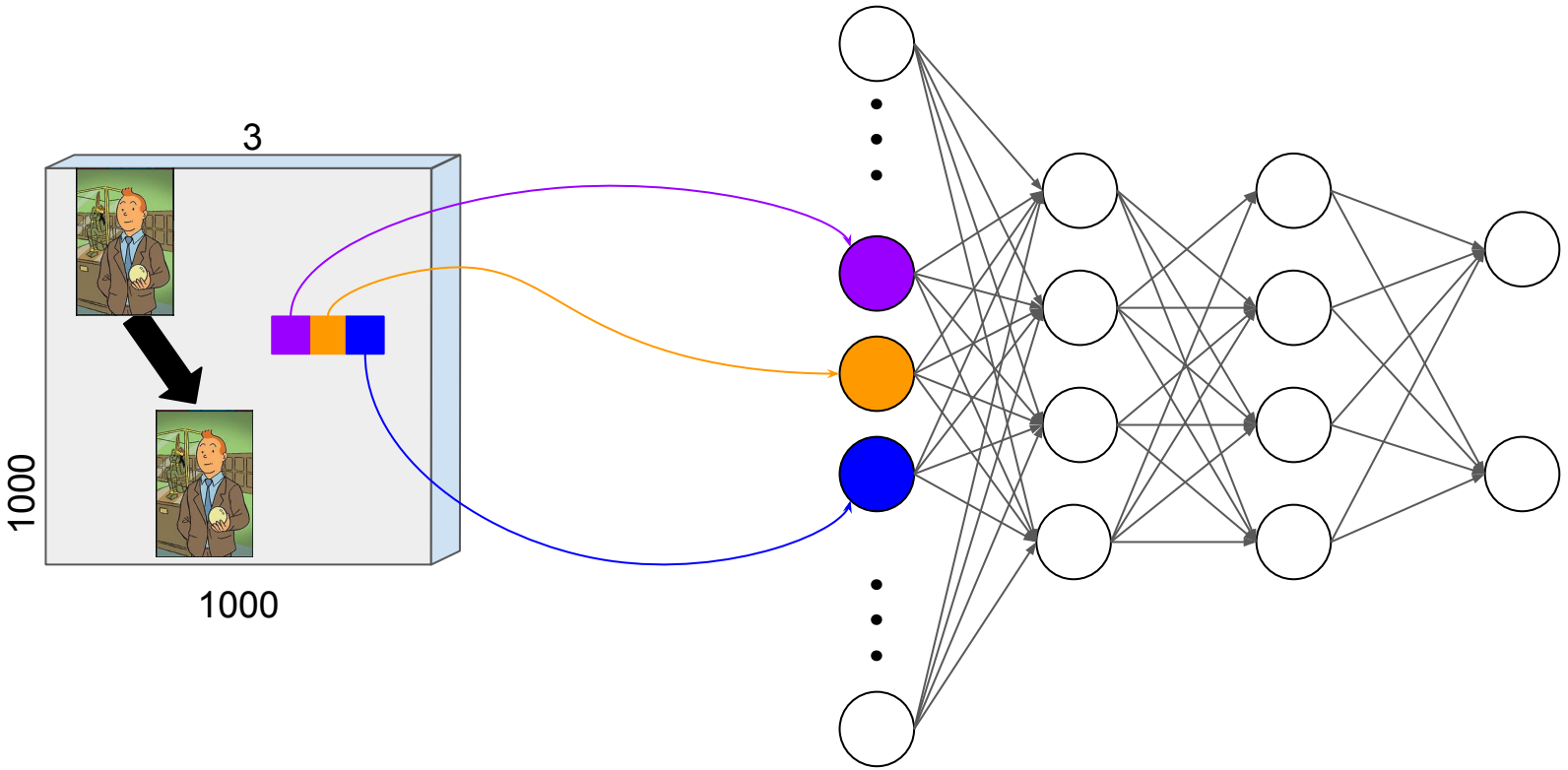
# Why convolutions?

Spatial information in images is essential

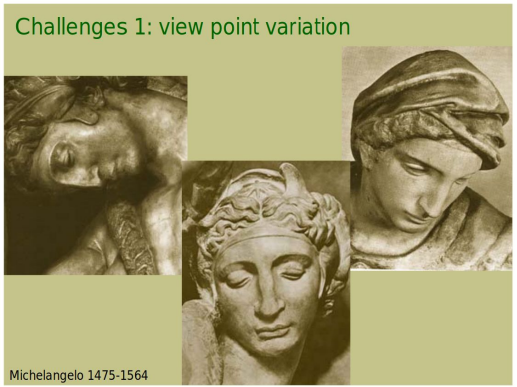
# Why convolutions?



# Why convolutions?

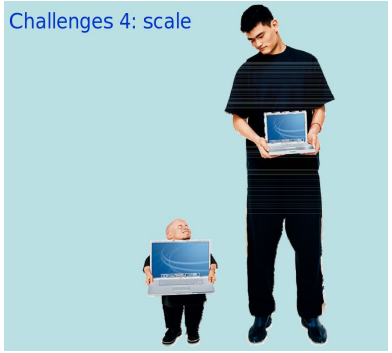


# Why convolutions?







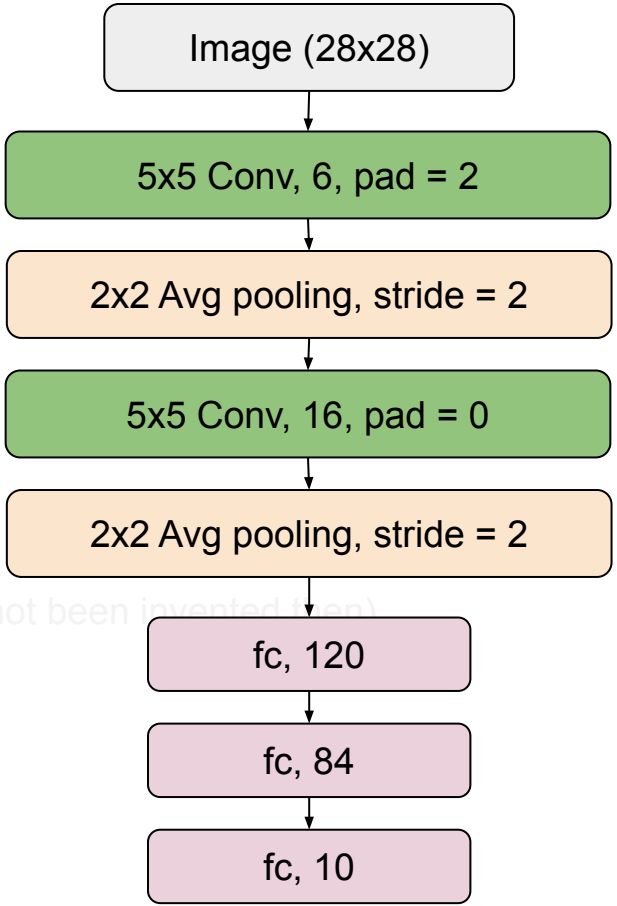




Slide credit:
Fei Fei Li

# LeNet - structure

- LeNet has two major parts
  - convolutional layers ?
  - fully-connected layers ✔
- Average pooling layers
  - max-pooling is more common these days
  - size of pooling was 2×2 with stride of 2
- convolutional layers use 5×5 kernels
  - Activation function used was the sigmoid (again ReLU not been invented then)

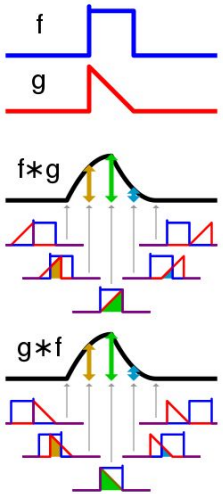


# Convolutions

# Convolutions

**Definition.** The convolution of two signals $f$ and $g$, denoted by $*$, is the integral of the product of the two signals after one being flipped and shifted

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau$$



# Convolutions

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t - \tau) d\tau$$

Discrete

$$(f * g)[t] = \sum_{-\infty}^{+\infty} f[\tau] g[t - \tau]$$

$$(f * g)[t] = (g * f)[t]$$

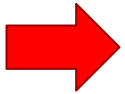
$$g[\tau] = 0; -k > \tau > k,$$

$$(f * g)[t] = \sum_{\tau=-k}^{+k} f[t - \tau] g[\tau]$$

# Convolutions

A related concept to convolution

$$(f * g)[t] = \sum_{\tau=-k}^{+k} f[t-\tau]g[\tau]$$



$$(f \star g)[t] = \sum_{\tau=-k}^{+k} f[t+\tau]g[\tau]$$

Extension to 2D

$$(f \star g)[r,c] = \sum_{\tau_x=-k_x}^{+k_x} \sum_{\tau_y=-k_y}^{+k_y} f[r+\tau_x, c+\tau_y]g[\tau_x, \tau_y]$$

# Convolutions

## Example

Input

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

| | |
|---|---|
| 0 | 1 |
| 2 | 3 |

Kernel

# Convolutions

## Example

Input

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

| | |
|---|---|
| 0 | 1 |
| 2 | 3 |

Kernel

?

# Convolutions

## Example

Input

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

| | |
|---|---|
| 0 | 1 |
| 2 | 3 |

Kernel

19

$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$

# Convolutions

## Example

Input

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

| | |
|---|---|
| 0 | 1 |
| 2 | 3 |

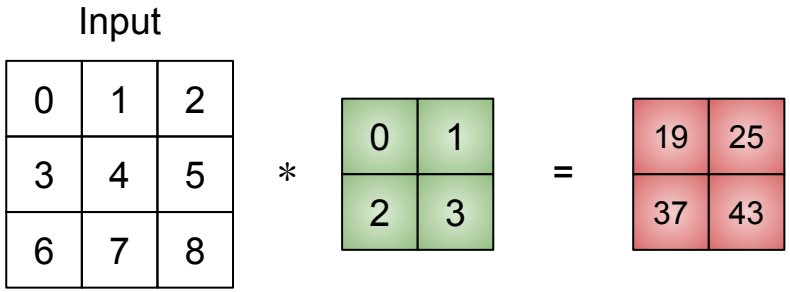
Kernel

| | |
|---|---|
| 19 | 25 |

$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25$

# Convolutions

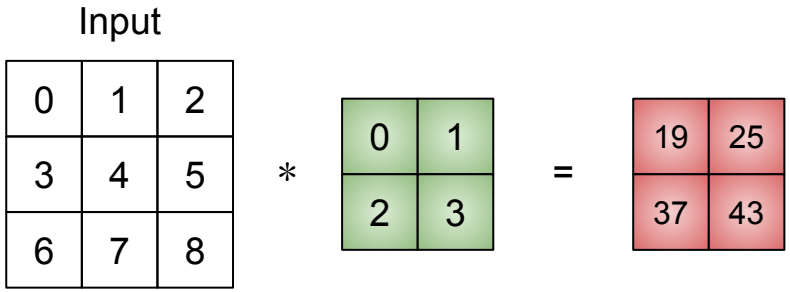
Example




What is the output size for an input of size H×W and a kernel of size k×k ?

# Convolutions

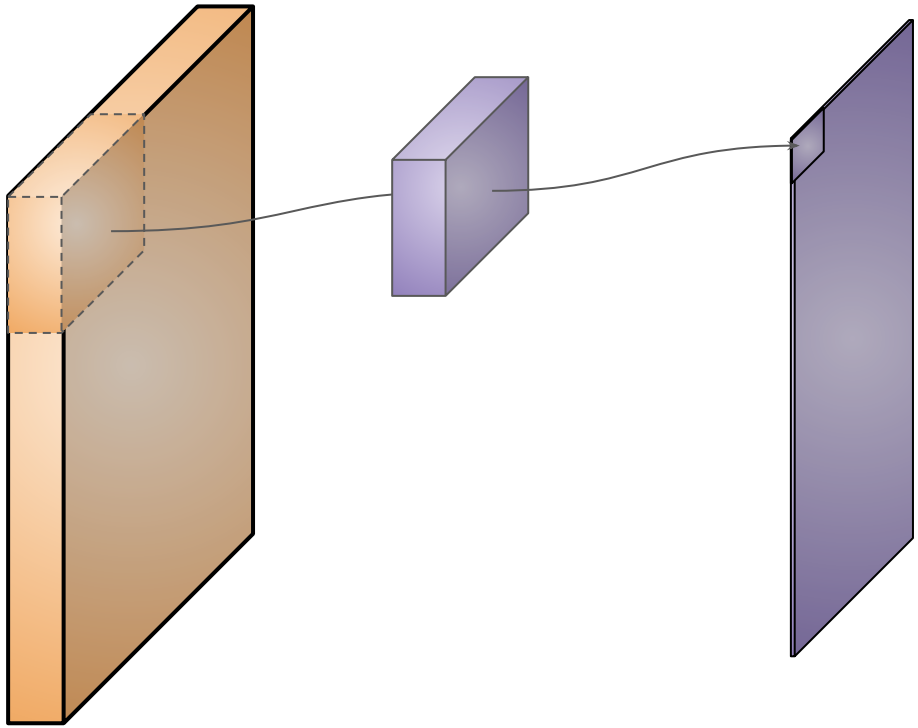
Example



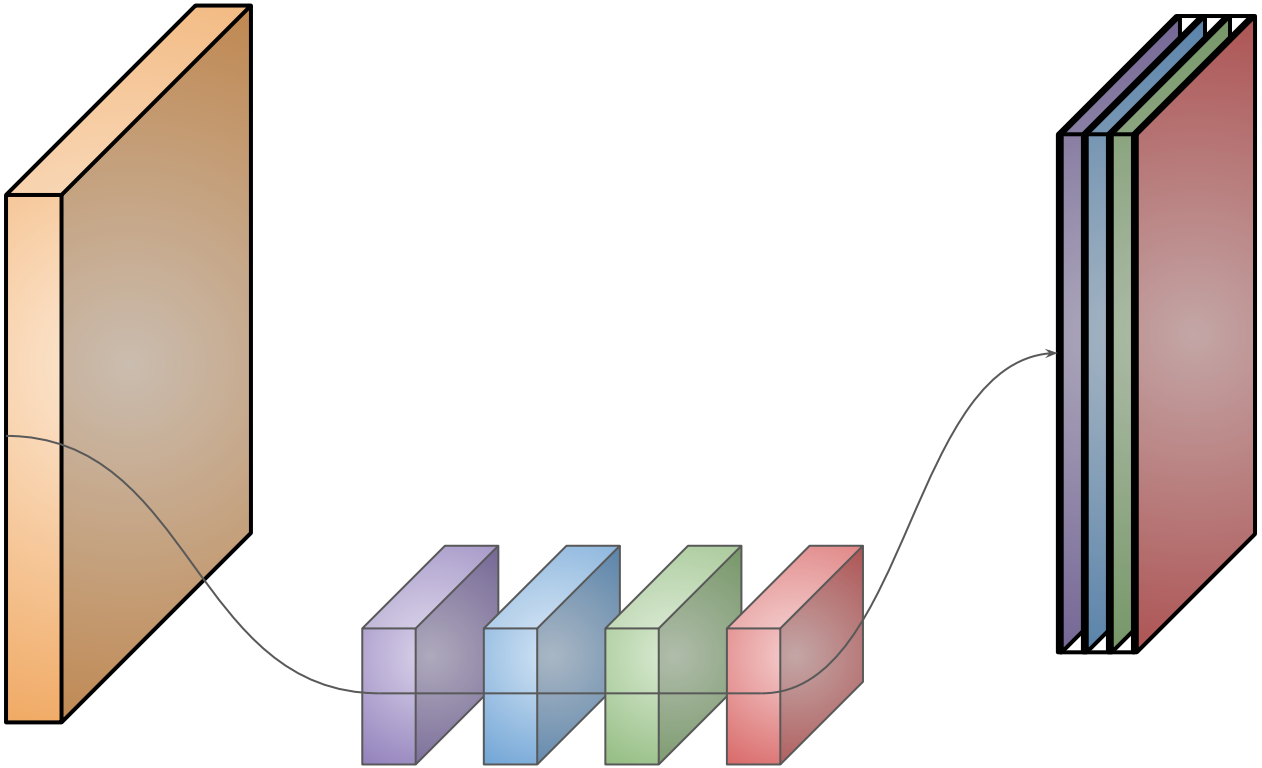

What is the output size for an input of size H×W and a kernel of size k×k ?

$$(H - k + 1)(W - k + 1)$$

# Convolution

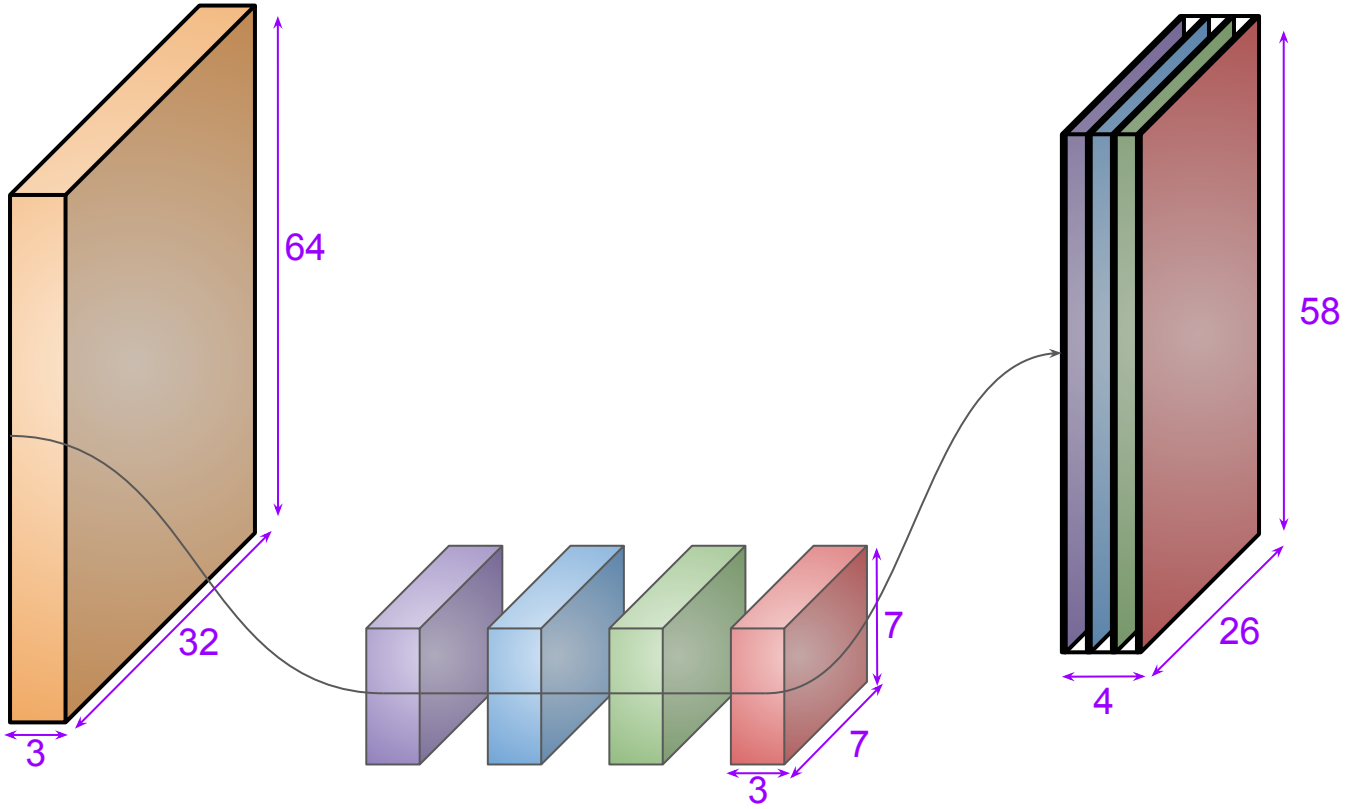


# Convolution



# Convolution

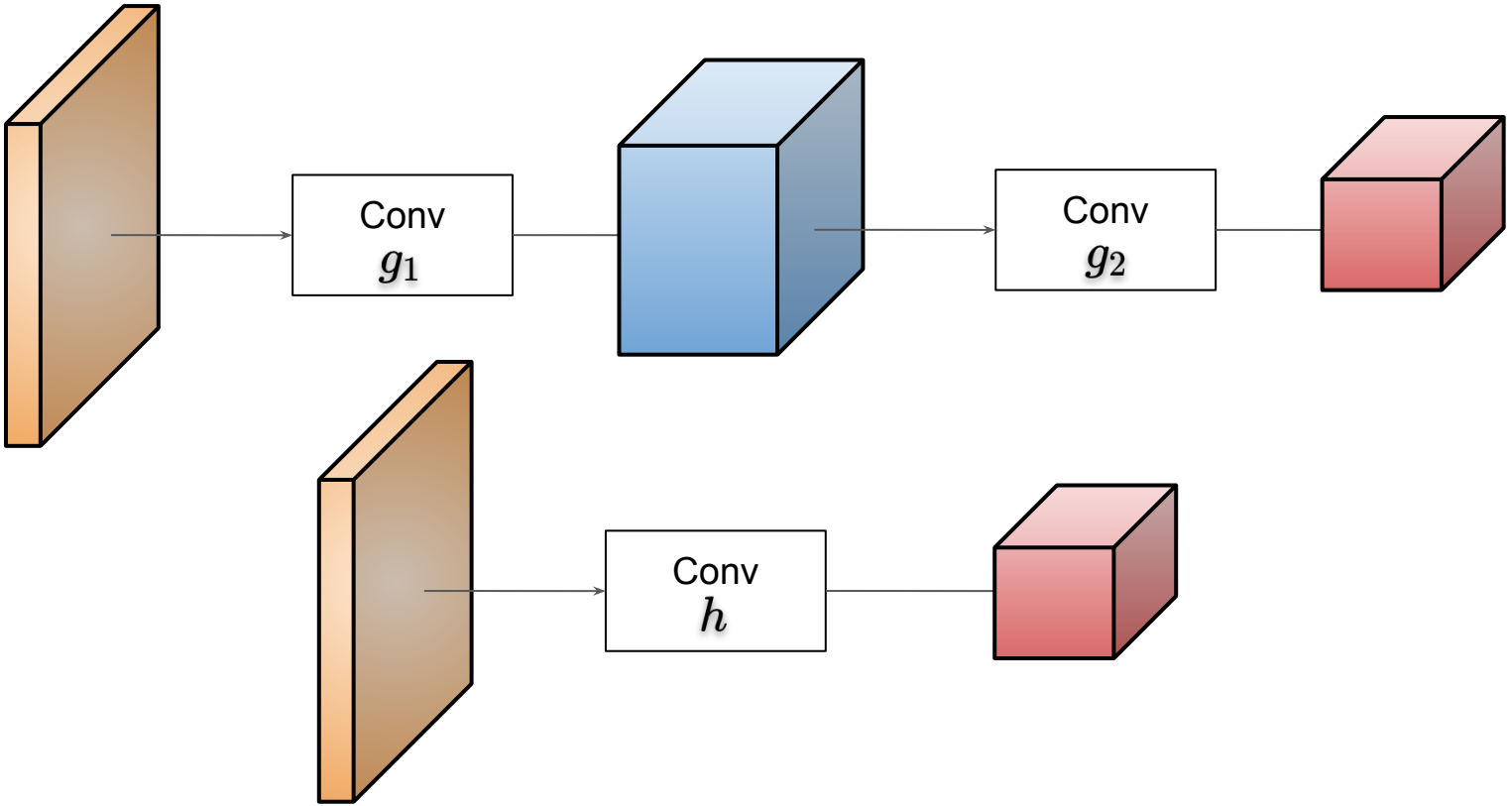
Example.



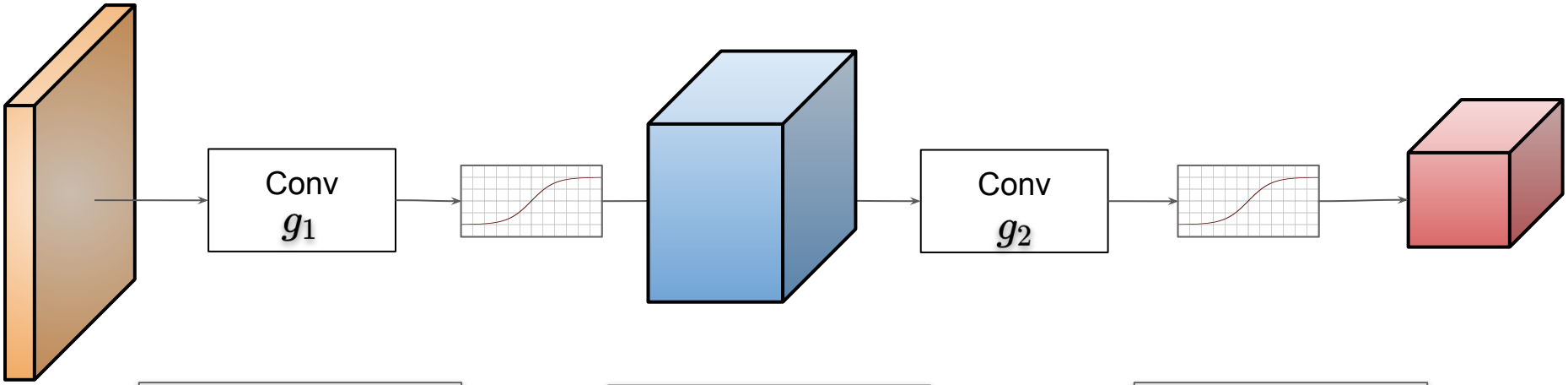
# Stacking Convolutions

Recall

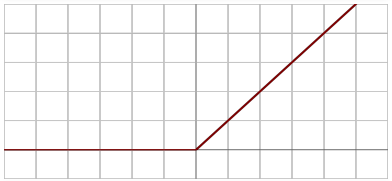
$$f * g_1 * g_2 = f * h; \quad h = g_1 * g_2$$



# Stacking Convolutions

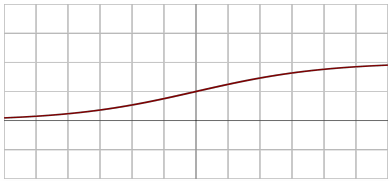



**ReLU**



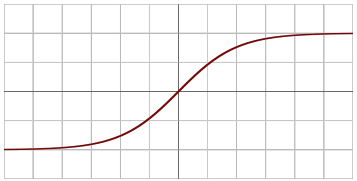
$f(x) = \max(0, x)$

**Sigmoid**



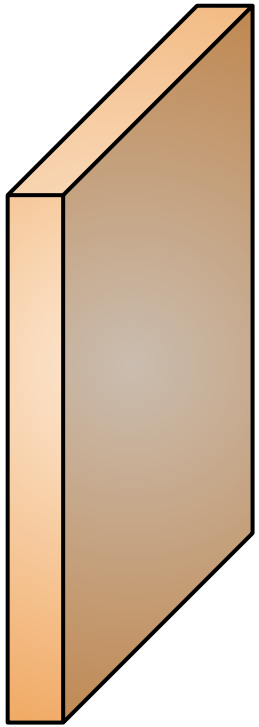
$f(x) = \frac{1}{1+\exp(-x)}$

**Tanh**



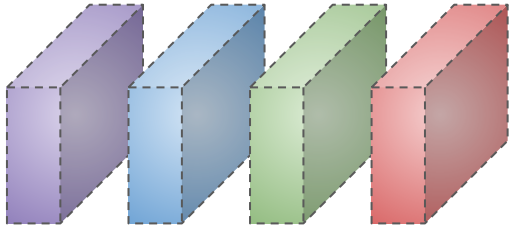
$f(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$

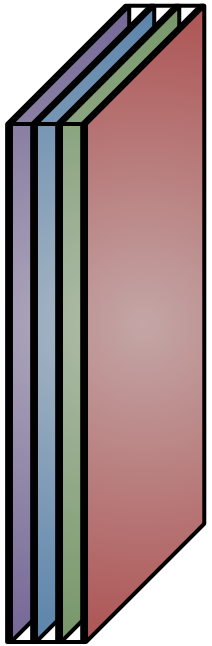
# Convolution



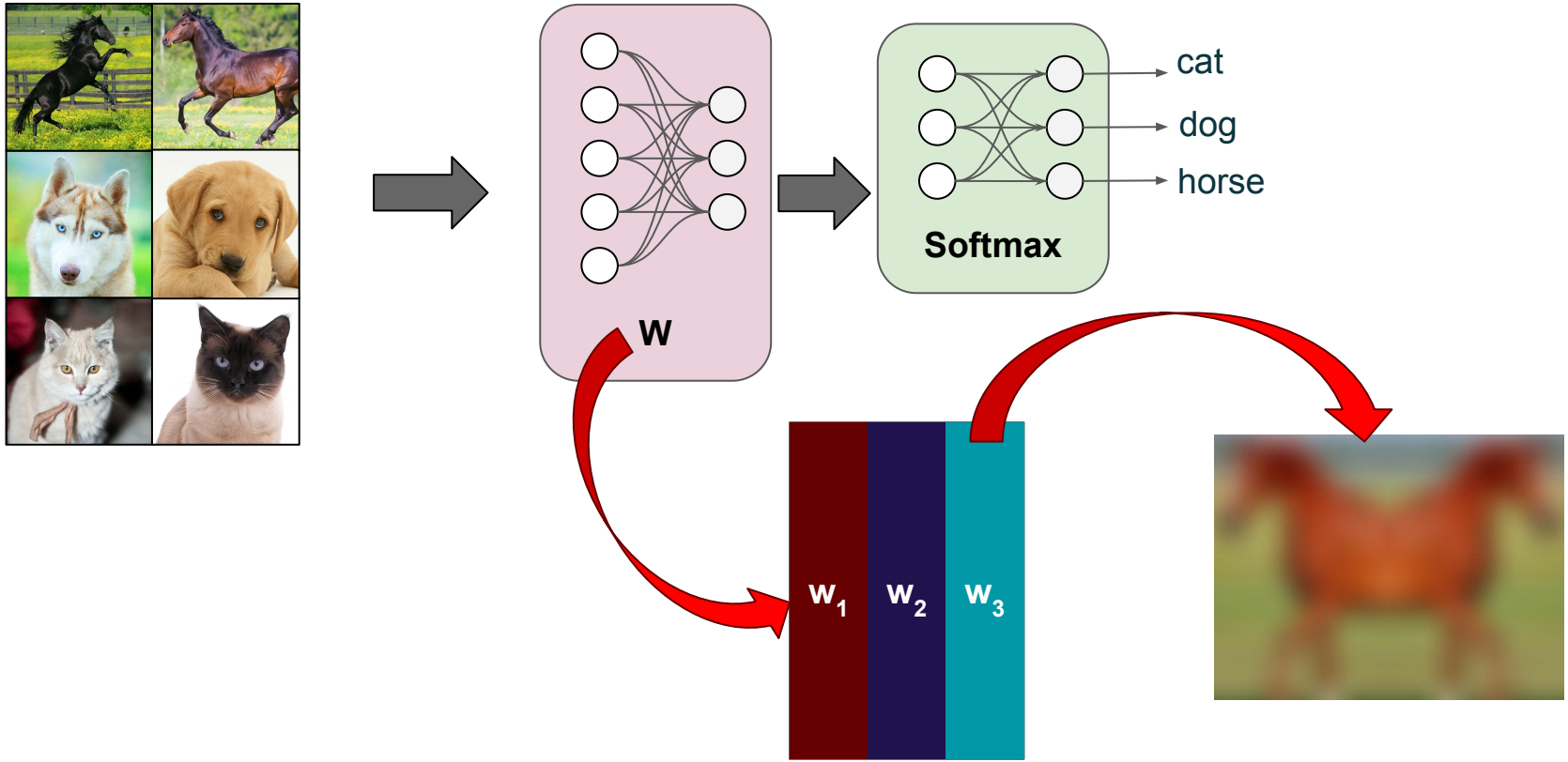
## CONV2D

CLASS `torch.nn.Conv2d(`*`in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros'`*`)` [SOURCE]

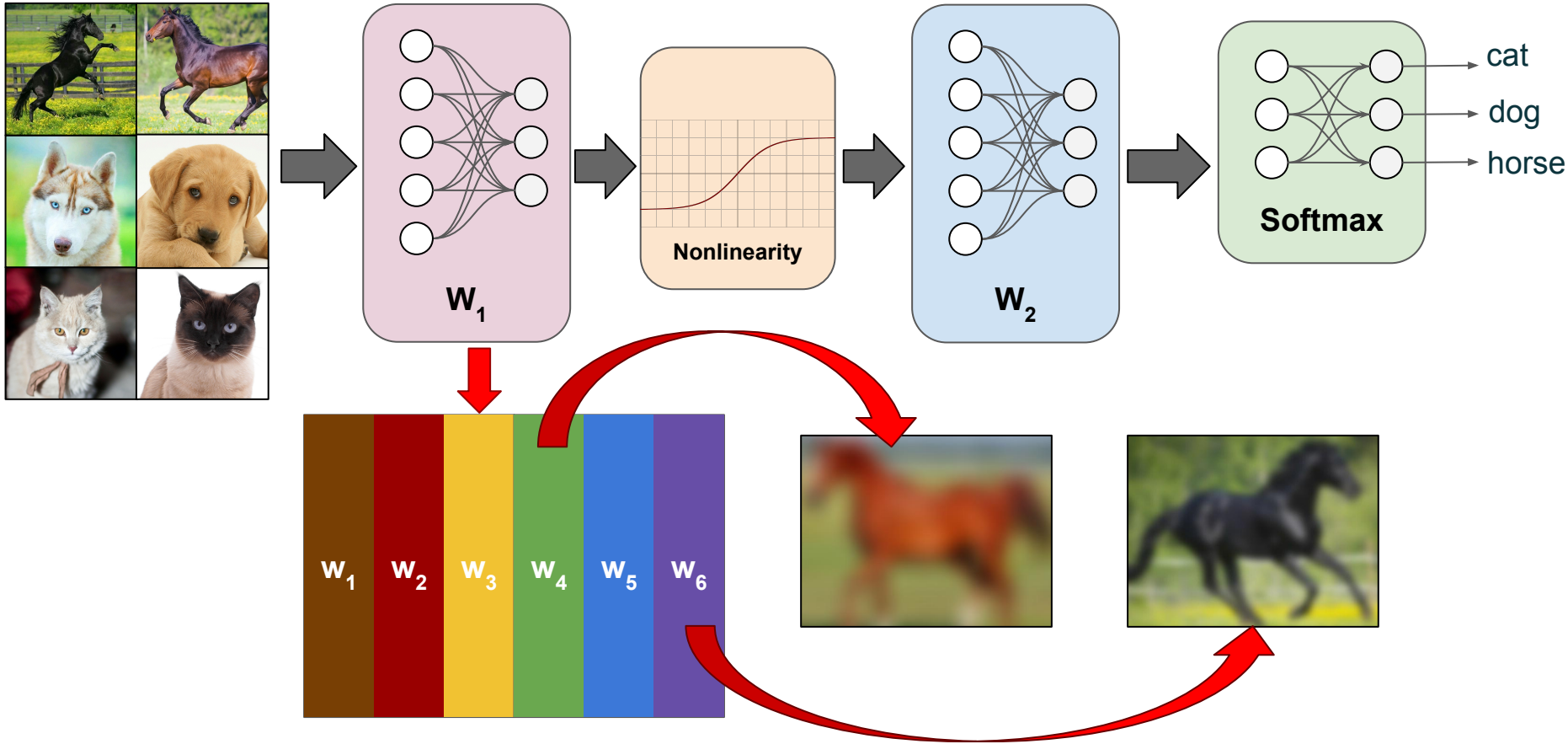




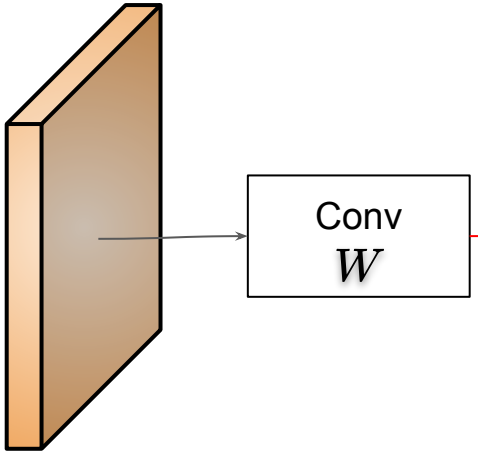
# Anatomy of an MLP

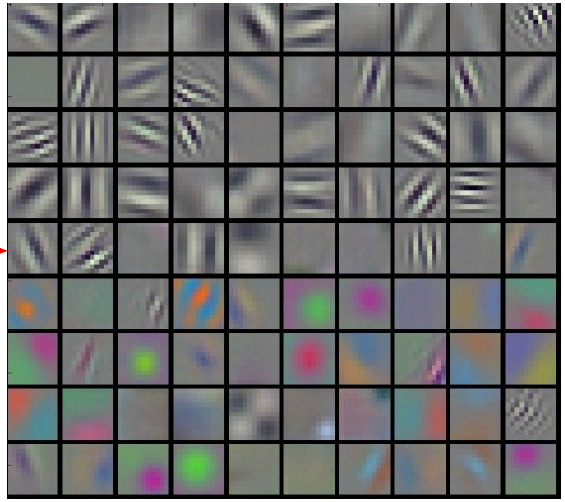


# Anatomy of an MLP



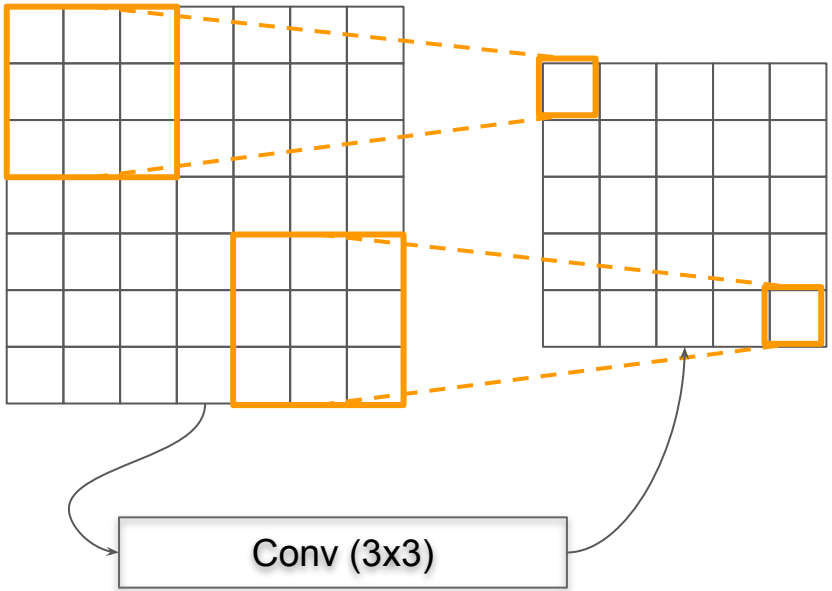
# Convolutional filters



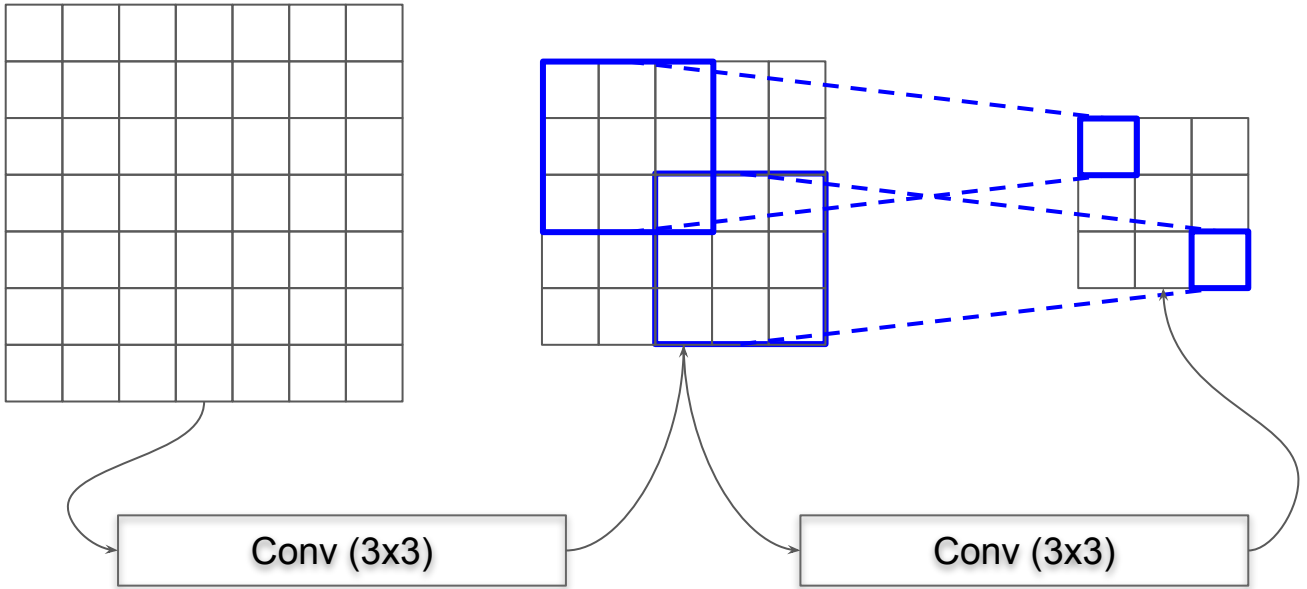



AlexNet: filters learned in the first convolutional layer, each 3x11x11

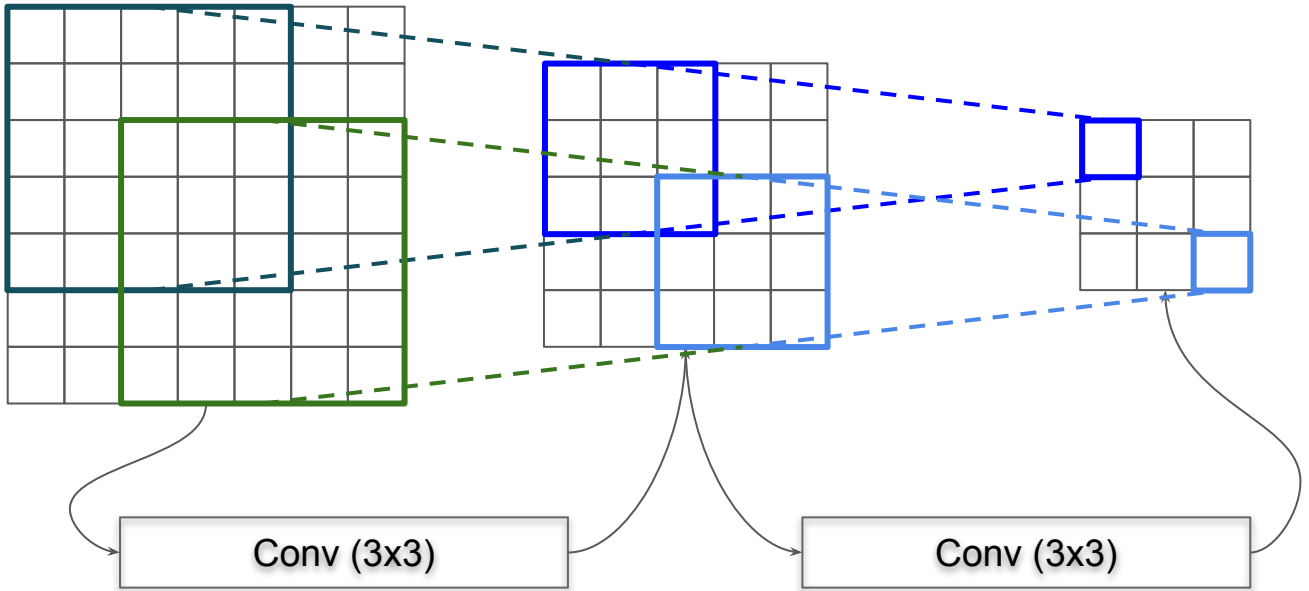
# Receptive Field



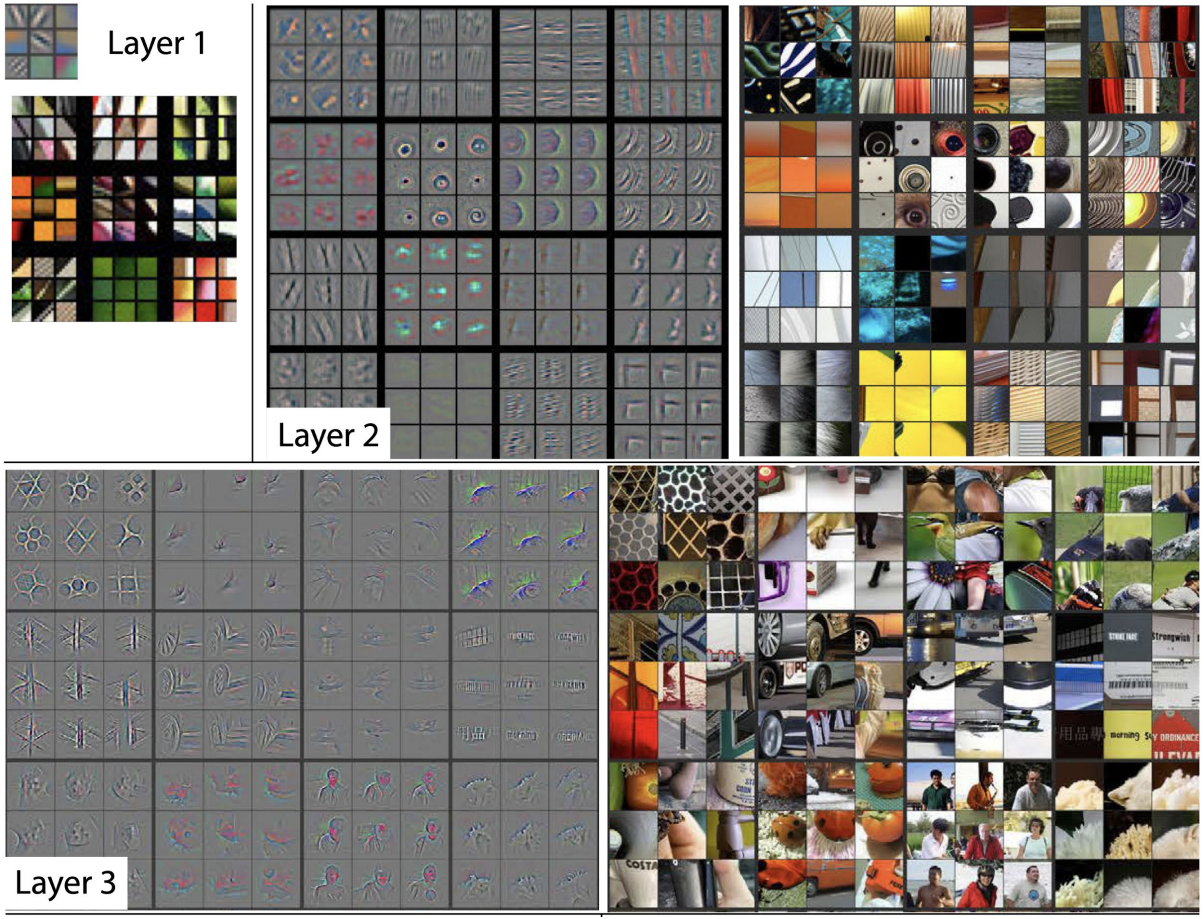
# Receptive Field



# Receptive Field

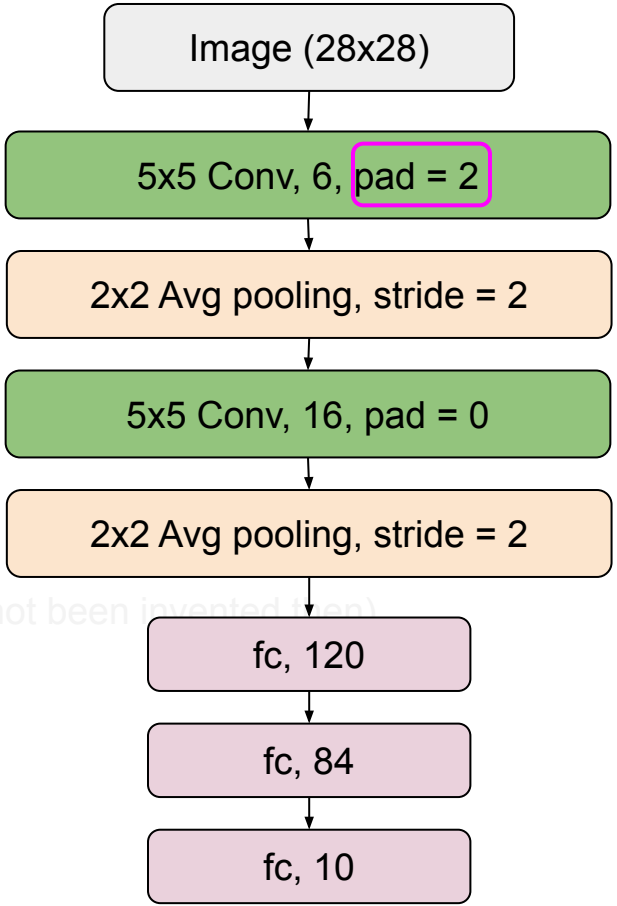


# Receptive Fields



# LeNet - structure

- LeNet has two major parts
  - convolutional layers ✔
  - fully-connected layers ✔
- Average pooling layers
  - max-pooling is more common these days
  - size of pooling was 2×2 with stride of 2
- convolutional layers use 5×5 kernels
  - Activation function used was the sigmoid (again ReLU not been invented then)



# Padding

# Padding

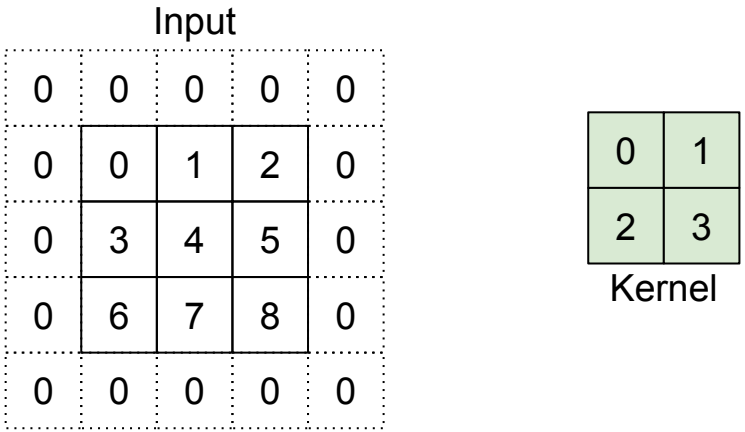
Input

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

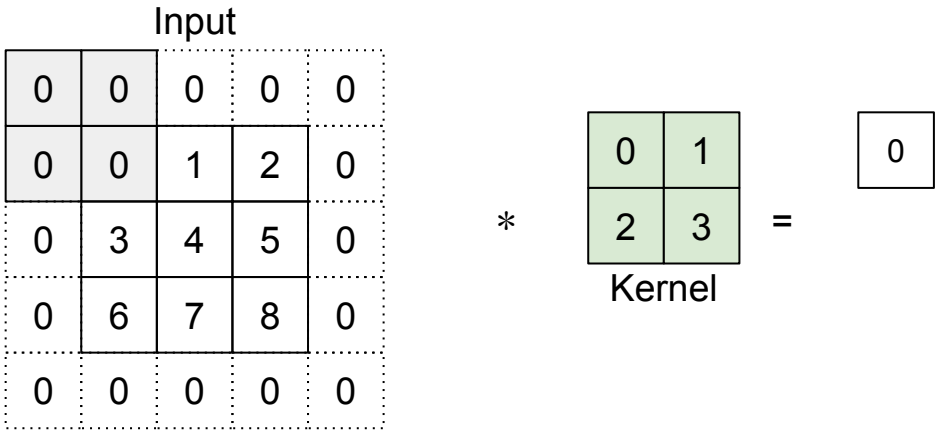
| | |
|---|---|
| 0 | 1 |
| 2 | 3 |

Kernel

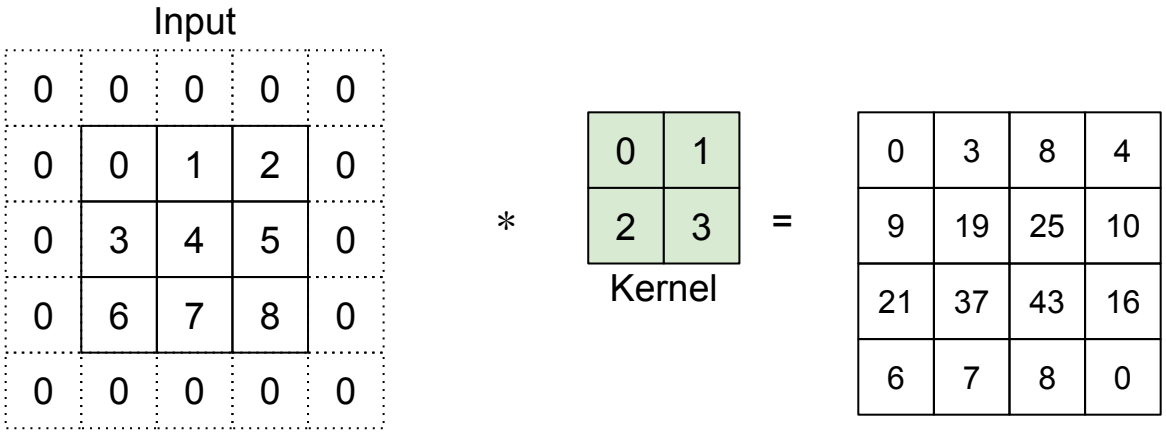
# Padding



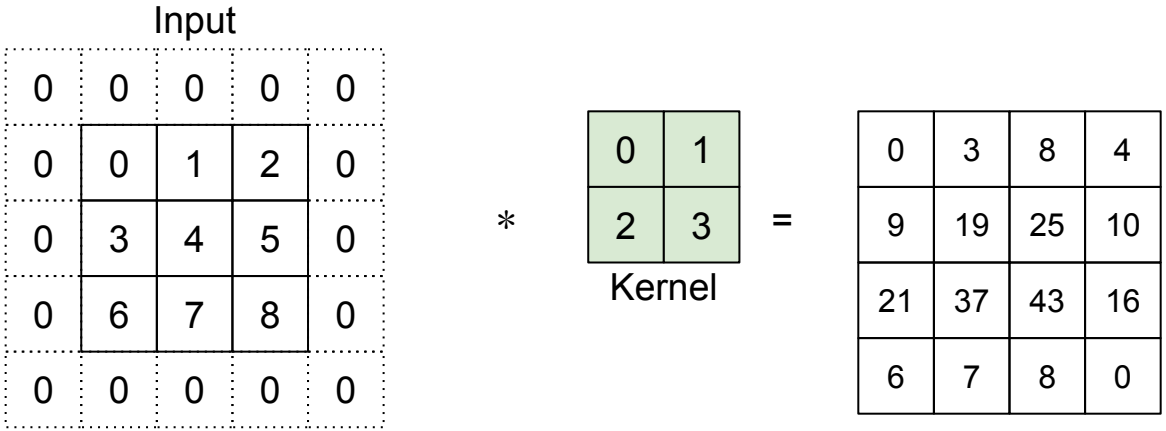
# Padding



# Padding


Input
0 0 0 0 0
0 0 1 2 0
0 3 4 5 0
0 6 7 8 0
0 0 0 0 0
*
0 1
2 3
Kernel
=
0 3 8 4
9 19 25 10
21 37 43 16
6 7 8 0

# Padding




Output size with padding:

$$(H - k + 1 + p_h)(W - k + 1 + p_w)$$

# Padding

Input

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

*

| | |
|---|---|
| 0 | 1 |
| 2 | 3 |

Kernel

=

| | | | |
|---|---|---|---|
| 0 | 3 | 8 | 4 |
| 9 | 19 | 25 | 10 |
| 21 | 37 | 43 | 16 |
| 6 | 7 | 8 | 0 |

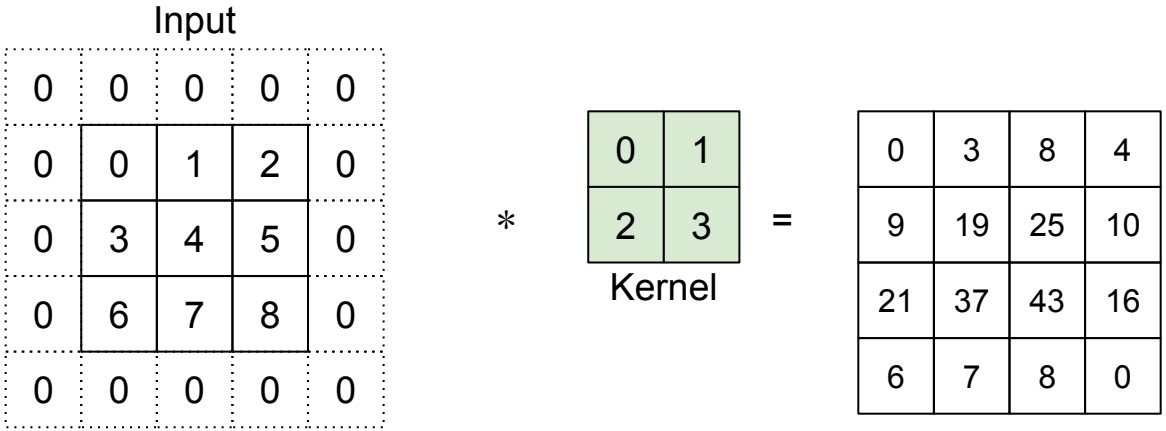
Output size with padding:

$$(H - k + 1 + p_h)(W - k + 1 + p_w)$$

total number of rows added (roughly half on top and half on bottom)

total number of columns added (roughly half on left and half on right)

# Padding




Output size with padding: $(H - k + 1 + p_h)(W - k + 1 + p_w)$

If $p_w = p_h = k - 1$, the size of input and output would be the same

# Padding

Assuming that $k$ is odd and $p_w = p_h = k - 1$, we will pad

- $p_h/2$ rows on top and bottom
- $p_w/2$ columns to left and right

If k is even, one possibility is to pad

- $\lceil p_h/2 \rceil$ rows on the top of the input and $\lfloor p_h/2 \rfloor$ rows on the bottom
- $\lceil p_w/2 \rceil$ columns to left of the input and $\lfloor p_w/2 \rfloor$ columns to right

To make things simpler, when we say we pad by $p$, we mean we will add $p$ rows/columns of zeros to the left, right, top and bottom of the input

# Padding

## CONV2D

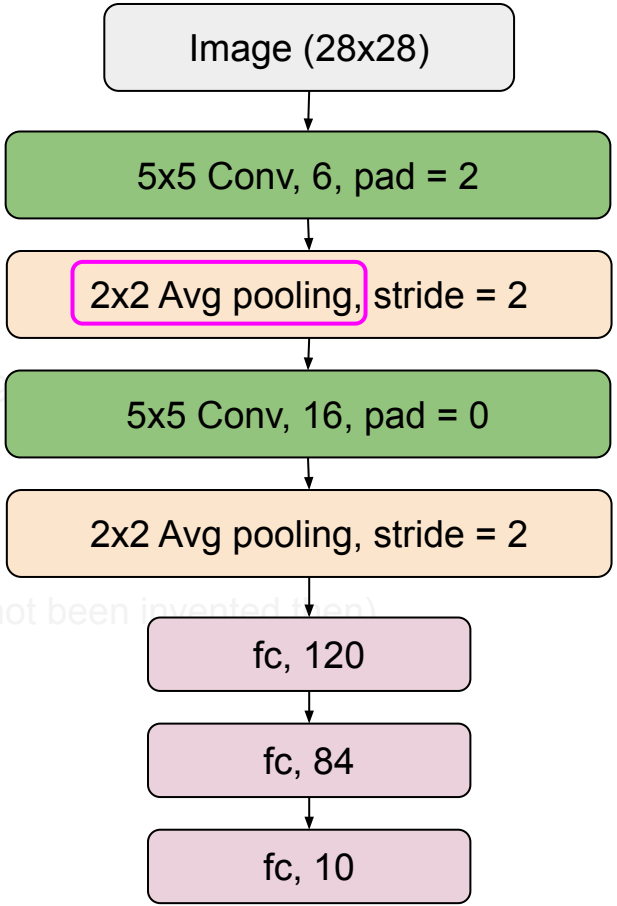
CLASS `torch.nn.Conv2d(in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros')` [SOURCE]

padding = 1 → Pad 1 column from left, 1 column from right, 1 row from top and 1 row from bottom

padding = (2,1) → Pad 2 columns from left, 2 columns from right, 1 row from top and 1 row from bottom

# LeNet - structure

- LeNet has two major parts
  - convolutional layers ✔
  - fully-connected layers ✔
- Average pooling layers ?
  - max-pooling works better, but not been invented in the 9
  - size of pooling was 2×2 with stride of 2
- convolutional layers use 5×5 kernels
  - Activation function used was the sigmoid (again ReLU not been invented then)

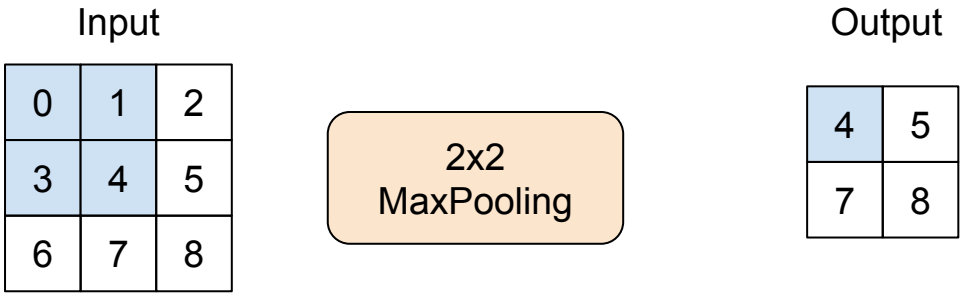


# Pooling

# Pooling

Convolution is sensitive to position

Max Pooling

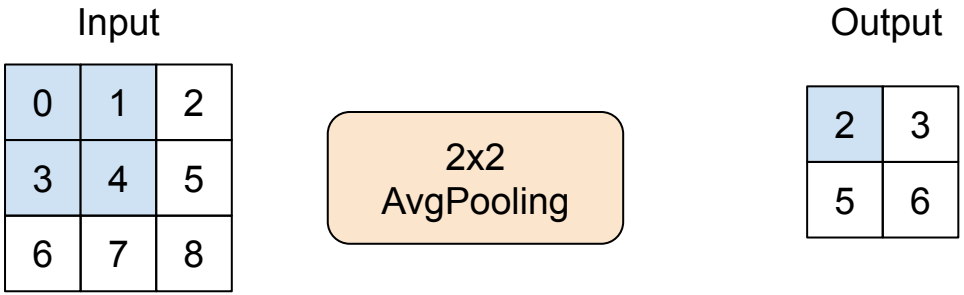



$$\max(0, 1, 3, 4) = 4$$

# Pooling

Convolution is sensitive to position

Avg Pooling




$$\text{mean}(0, 1, 3, 4) = 2$$

# Padding

## MAXPOOL2D

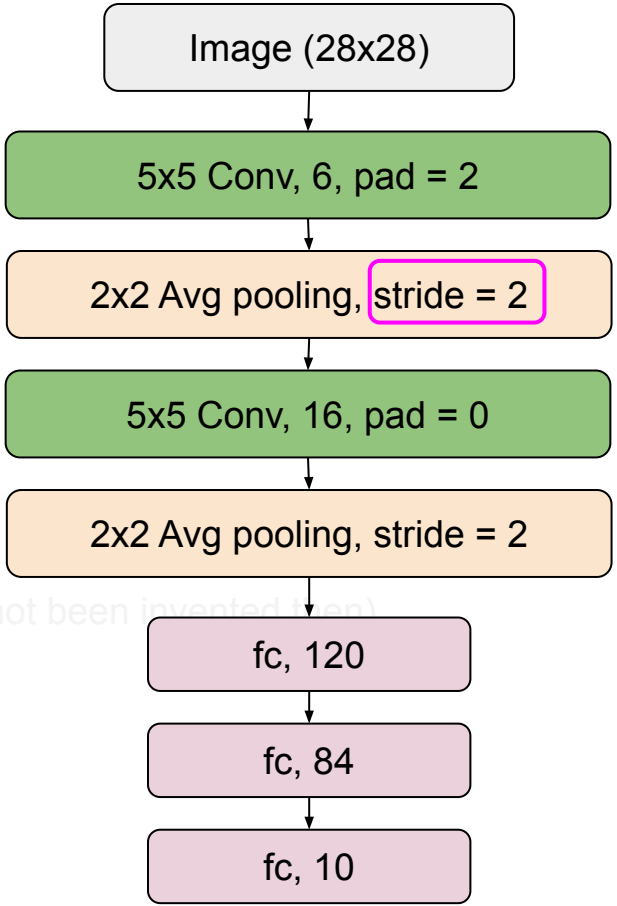
CLASS `torch.nn.MaxPool2d(kernel_size: Union[T, Tuple[T, ...]], stride: Optional[Union[T, Tuple[T, ...]]] = None, padding: Union[T, Tuple[T, ...]] = 0, dilation: Union[T, Tuple[T, ...]] = 1, return_indices: bool = False, ceil_mode: bool = False)` [SOURCE]

## AVGPOOL2D

CLASS `torch.nn.AvgPool2d(kernel_size: Union[T, Tuple[T, T]], stride: Optional[Union[T, Tuple[T, T]]] = None, padding: Union[T, Tuple[T, T]] = 0, ceil_mode: bool = False, count_include_pad: bool = True, divisor_override: bool = None)` [SOURCE]

# LeNet - structure

- LeNet has two major parts
  - convolutional layers ✔
  - fully-connected layers ✔
- Average pooling layers
  - max-pooling is more common these days
  - size of pooling was 2×2 with stride of 2
- convolutional layers use 5×5 kernels
  - Activation function used was the sigmoid (again ReLU not been invented then)



# Strides

# Strides

- Padding reduces dimensionality linearly with #layers
- Given an input of size 224 x 224, with a 5 x 5 kernel, after 40 layers, the feature maps are of size 64 x 64
- Stride is the #rows and #columns we slide the convolutional filter

# Strides

Horizontal stride = 2
Vertical stride = 3

Input

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

# Strides

Horizontal stride = 2
Vertical stride = 3

Input

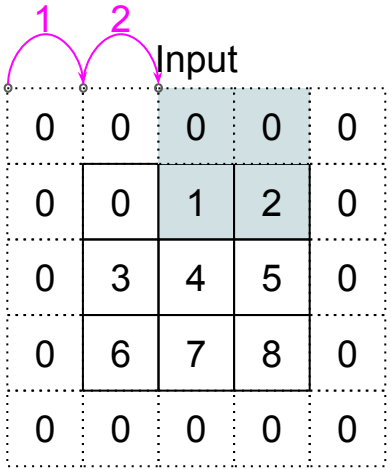
| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 3 | 4 | 5 | 0 |
| 0 | 6 | 7 | 8 | 0 |
| 0 | 0 | 0 | 0 | 0 |

# Strides

Horizontal stride = 2
Vertical stride = 3

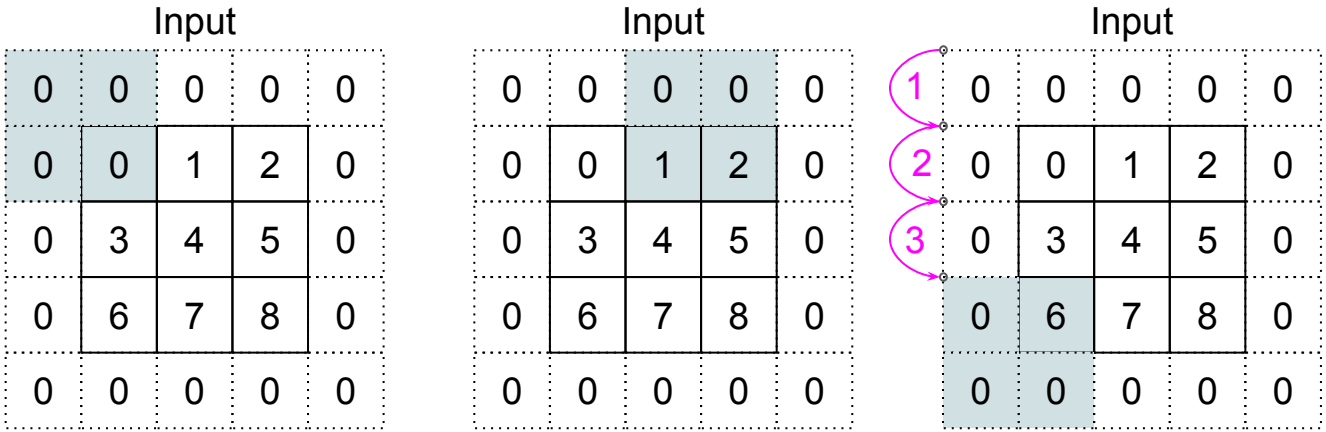
Input

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



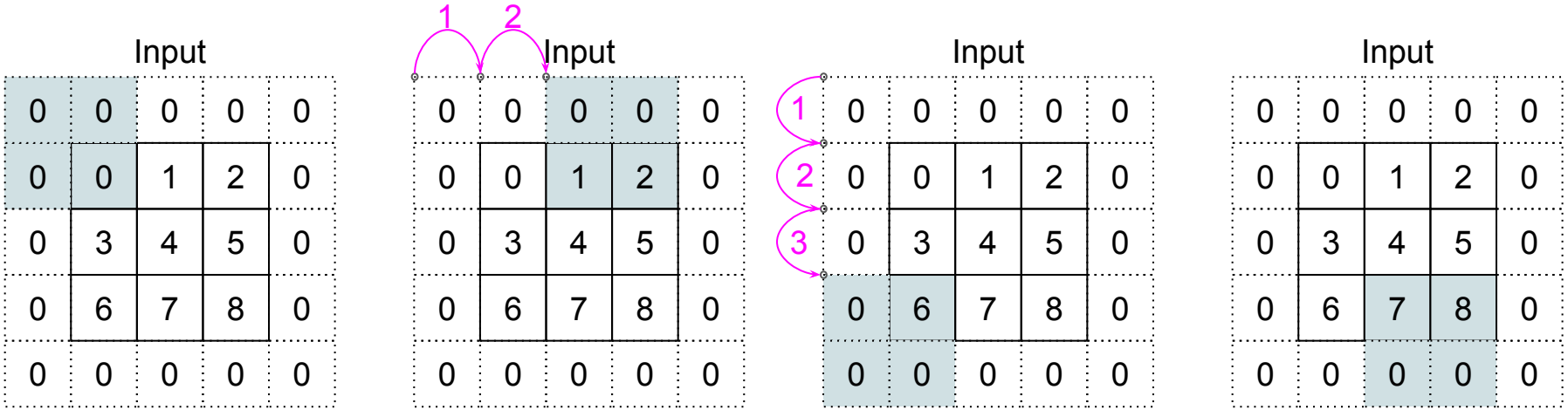
# Strides

Horizontal stride = 2
Vertical stride = 3



# Strides

Horizontal stride = 2
Vertical stride = 3



# Strides

## CONV2D

CLASS torch.nn.Conv2d(*in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros'*) [SOURCE]

$$W_o = \left\lfloor \frac{W_i - k_w + 2p_w}{s_w} + 1 \right\rfloor \qquad H_o = \left\lfloor \frac{H_i - k_h + 2p_h}{s_h} + 1 \right\rfloor$$

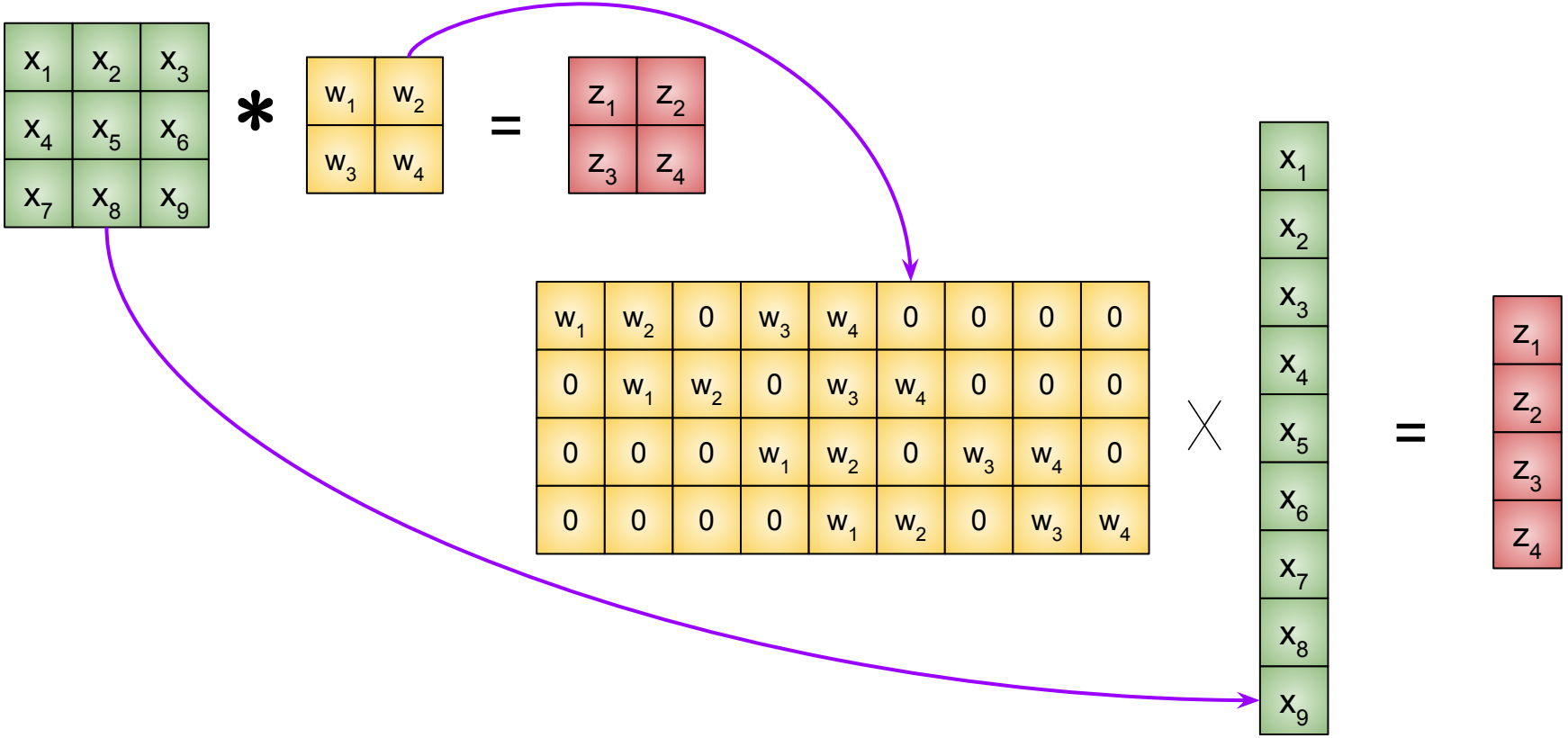
stride = 2

stride = (2,3)

Stride is 2 along the columns and 3 along the rows

# Unrolling the convolutions

# Unrolling


x1 x2 x3
x4 x5 x6
x7 x8 x9
*
w1 w2
w3 w4
=
z1 z2
z3 z4
w1 w2 0 w3 w4 0 0 0 0
0 w1 w2 0 w3 w4 0 0 0
0 0 0 w1 w2 0 w3 w4 0
0 0 0 0 w1 w2 0 w3 w4
x1 x2 x3 x4 x5 x6 x7 x8 x9
=
z1 z2 z3 z4

# Summary