



deeplearning.ai

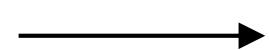
# Convolutional Neural Networks

---

## Computer vision

# Computer Vision Problems

Image Classification



Cat? (0/1)

Neural Style Transfer



Object detection



Andrew Ng

# Deep Learning on large images



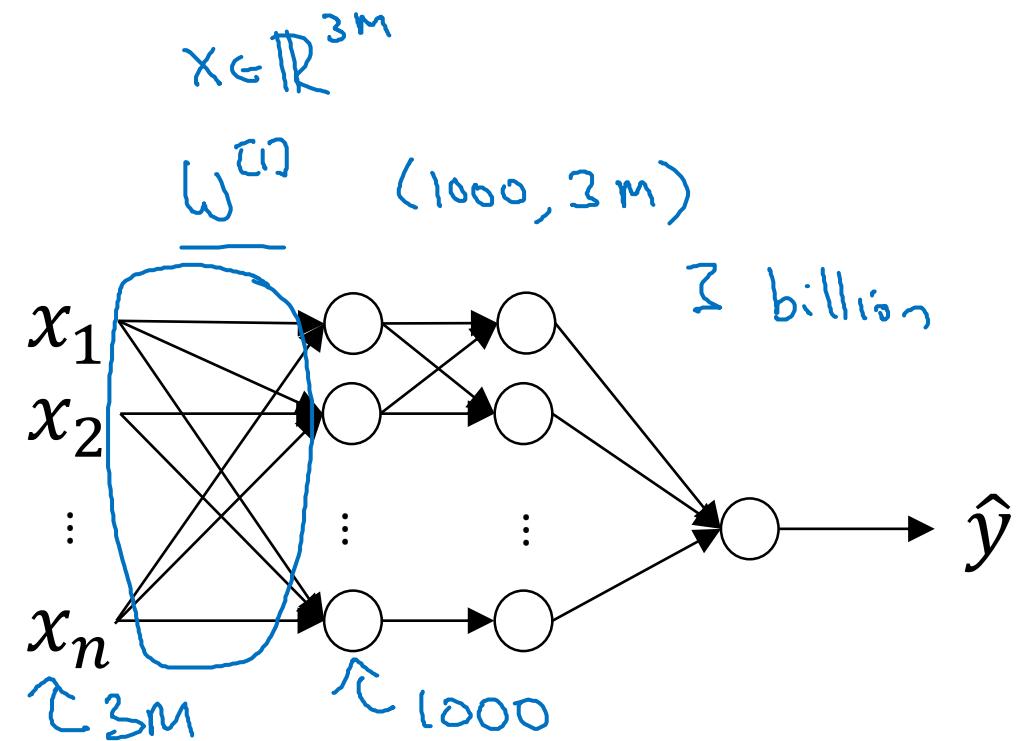
$64 \times 64 \times 3$

→ Cat? (0/1)

12288



$1000 \times 1000 \times 3$   
= 3 million





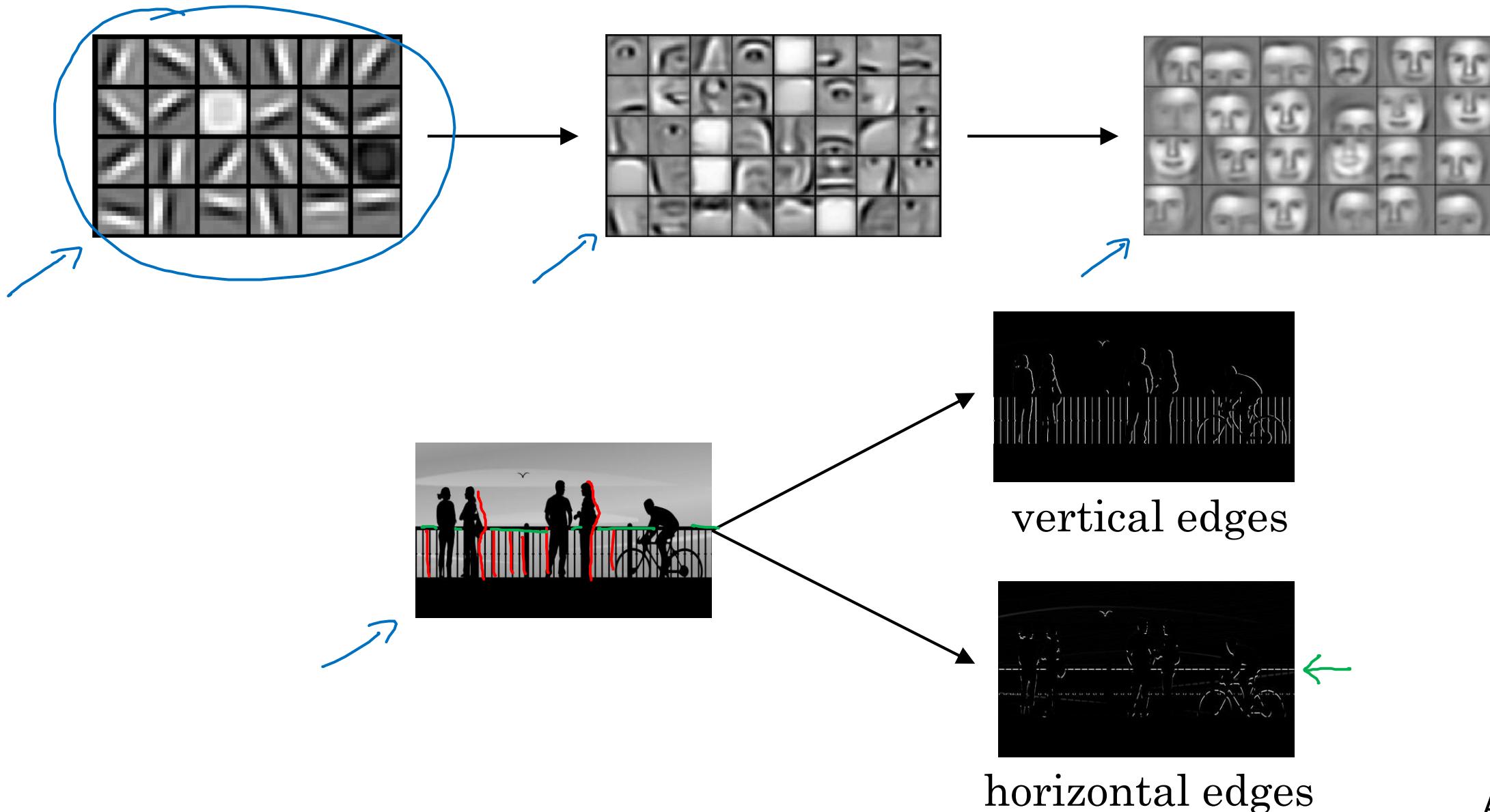
deeplearning.ai

# Convolutional Neural Networks

---

Edge detection  
example

# Computer Vision Problem

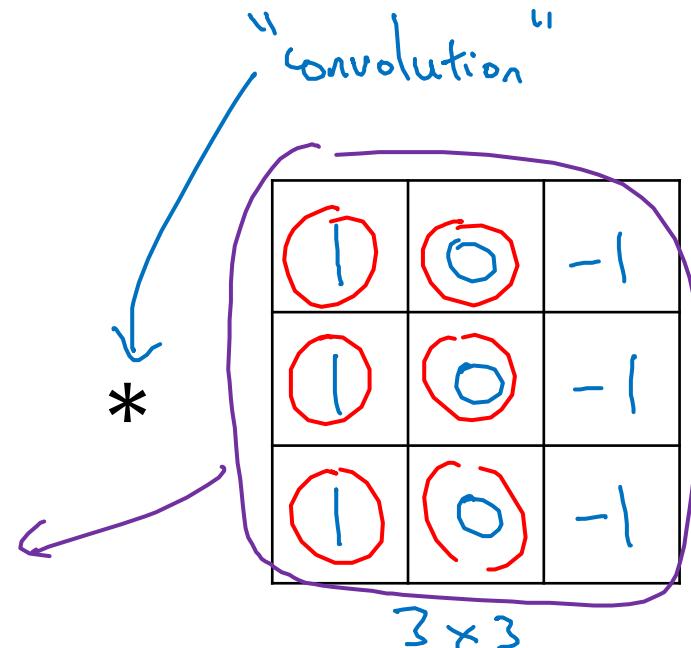


# Vertical edge detection

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times 1 + 8 \times -1 + 2 \times -1 = -5$$

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

$6 \times 6$



=

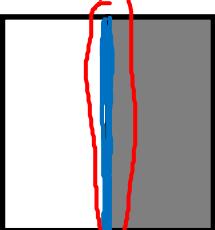
-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

$4 \times 4$

# Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$6 \times 6$



$\uparrow \uparrow \uparrow$

\*

1	0	-1
1	0	-1
1	0	-1

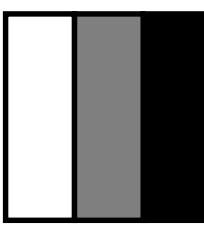
$3 \times 3$

=

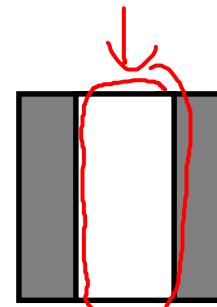
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

$\uparrow 4 \times 4$

\*



$\uparrow \uparrow \uparrow$



Andrew Ng



deeplearning.ai

# Convolutional Neural Networks

---

More edge  
detection

# Vertical edge detection examples

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0



0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10



\*

1	0	-1
1	0	-1
1	0	-1



=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



\*

1	0	-1
1	0	-1
1	0	-1

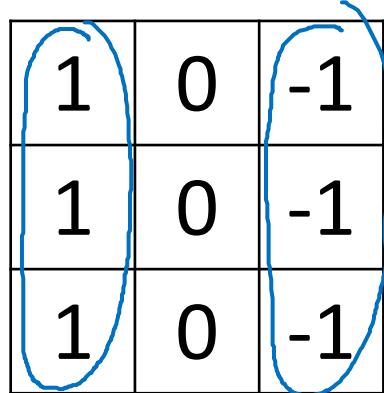


=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0



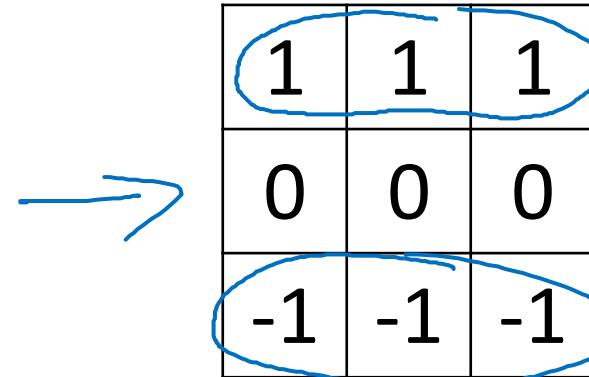
# Vertical and Horizontal Edge Detection



A 3x3 matrix with values 1, 0, -1 in each row. The first and third columns are circled in blue, indicating vertical edges.

1	0	-1
1	0	-1
1	0	-1

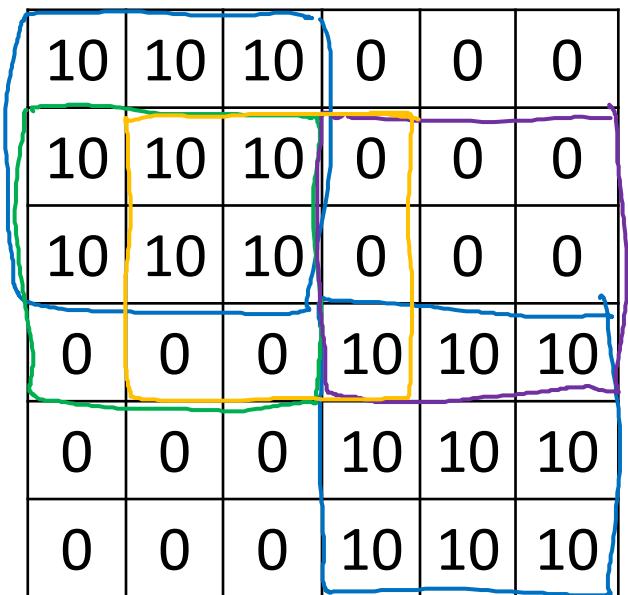
Vertical



A 3x3 matrix with values 1, 1, 1 in the first row and -1, -1, -1 in the third row. These rows are circled in blue, indicating horizontal edges.

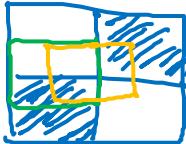
1	1	1
0	0	0
-1	-1	-1

Horizontal

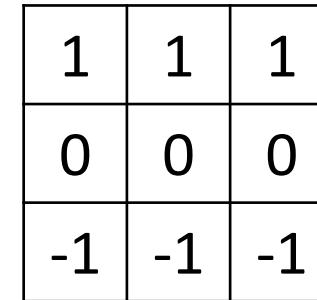


A 6x6 matrix with values 10, 0, and 10 in the first three rows. The last three rows are zero. A 3x3 kernel is applied to the first three rows. The result is shown in the bottom row, where the first three elements are 30, 10, and -10, and the last three are -30. The input matrix is labeled 6x6.

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10



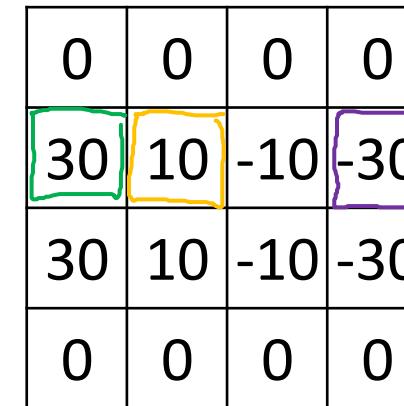
\*



A 3x3 matrix with values 1, 1, 1 in the first row and -1, -1, -1 in the third row. The middle value is 10. Below it is a small diagram showing a 3x3 grid with a central square containing 10 and other squares containing hatching patterns.

1	1	1
0	0	0
-1	-1	-1

=



A 4x4 matrix with values 0, 0, 0, 0 in the first row and 30, 10, -10, -30 in the second row. The input matrix is labeled 6x6 and the kernel is labeled 3x3.

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

# Learning to detect edges

1	0	-1
1	0	-1
1	0	-1

→

1	0	-1
2	0	-2
1	0	-1

3	0	-3
10	0	-10
3	0	-3

↑

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

Sobel filter

convolution

×

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

$3 \times 3$

=

$45^\circ$   
 $70^\circ$   
 $73^\circ$

↑




deeplearning.ai

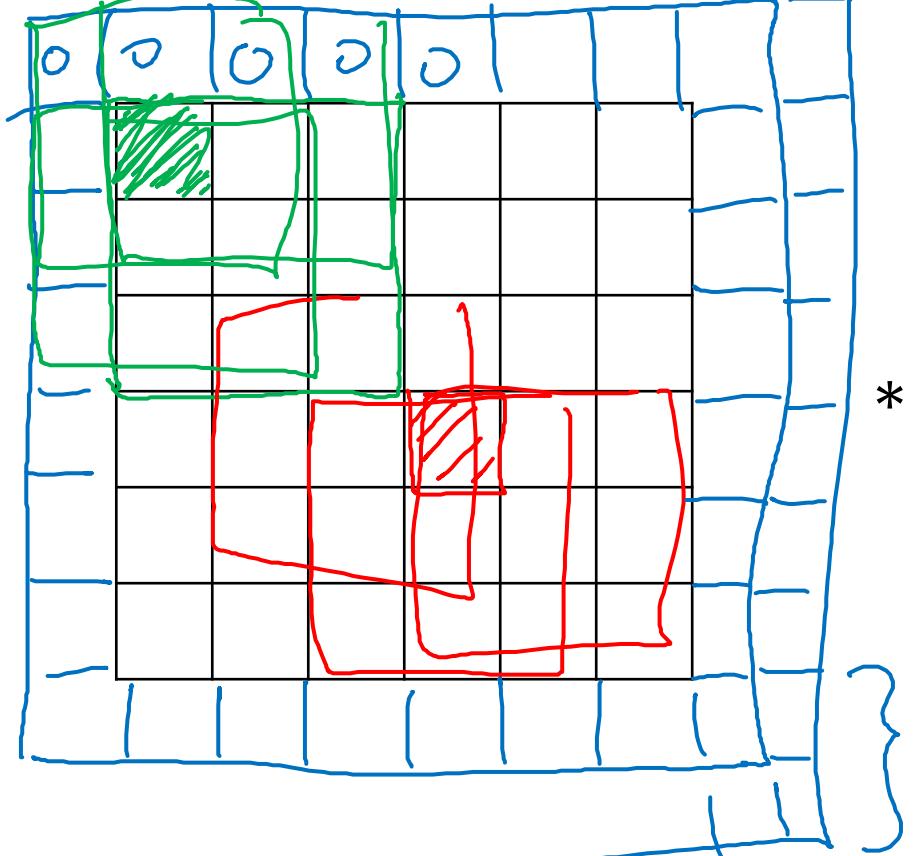
# Convolutional Neural Networks

---

## Padding

# Padding

- shrinky output
- throw away info from edge

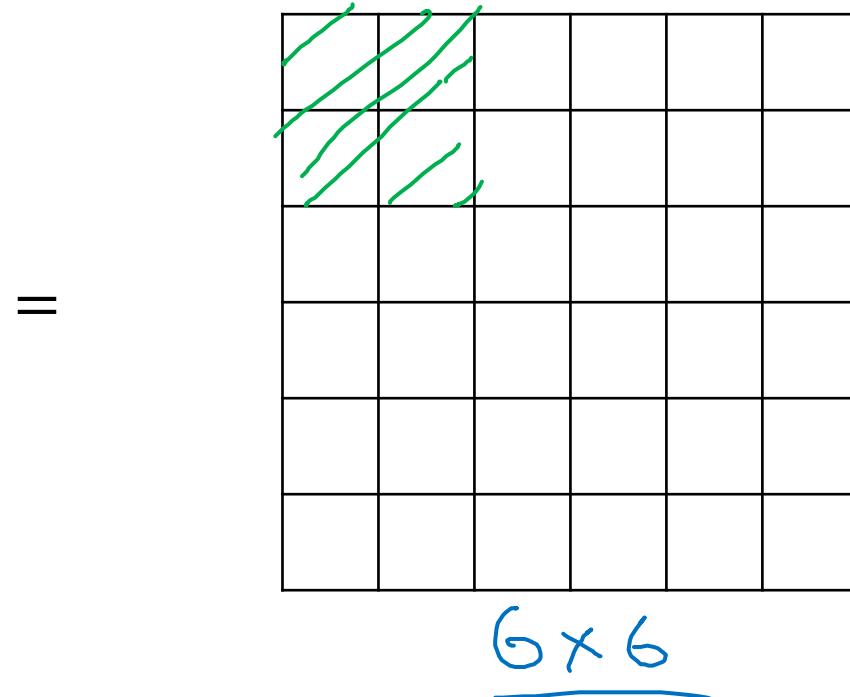


$$* \quad \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

$3 \times 3$   
 $f \times f$

$$n-f+1 \times n-f+1$$
$$6-3+1=4$$

$$P = \text{padding} = 1$$



$$\frac{n+2p-f+1 \times n+2p-f+1}{n \times n} = 6 \times 6$$

# Valid and Same convolutions

→ n → padding

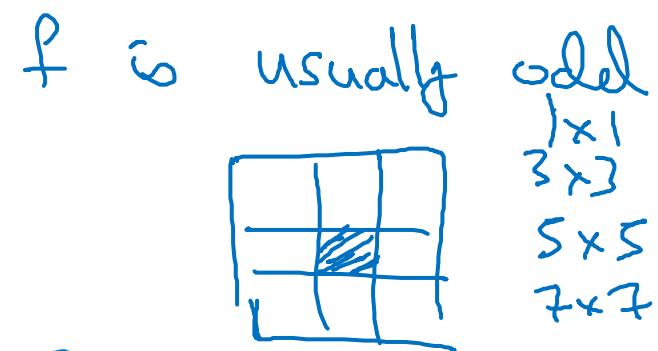
“Valid”:  $n \times n \quad * \quad f \times f \quad \rightarrow \frac{n-f+1}{f} \times \frac{n-f+1}{f}$

$6 \times 6 \quad * \quad 3 \times 3 \quad \rightarrow \quad 4 \times 4$

“Same”: Pad so that output size is the same as the input size.

$$n + 2p - f + 1 = n \Rightarrow p = \frac{f-1}{2}$$

$3 \times 3 \quad p = \frac{3-1}{2} = 1 \quad | \quad \begin{matrix} S \times S \\ f = 3 \end{matrix} \quad p=2$





deeplearning.ai

# Convolutional Neural Networks

---

## Strided convolutions

# Strided convolution

A 7x7 input matrix with values ranging from -1 to 9. A 3x3 kernel is applied to the top-left 3x3 submatrix. Blue arrows indicate the receptive field of the output unit at position (1,1).

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

7x7

$n \times n$  \*  $f \times f$   
 padding p      stride s  
 $s=2$

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline 3 & 4 & 4 \\ \hline 1 & 0 & 2 \\ \hline -1 & 0 & 3 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 91 & 100 & 83 \\ \hline 69 & 91 & 127 \\ \hline 44 & 72 & 74 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 3 \times 3 \\ \hline \lfloor z \rfloor = \text{floor}(z) \\ \hline \end{array}
 \end{array}$$

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

$$\frac{7+0-3}{2} + 1 = \frac{4}{2} + 1 = 3$$

# Summary of convolutions

$n \times n$  image       $f \times f$  filter

padding  $p$       stride  $s$

Output size:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \quad \times \quad \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$


# Technical note on cross-correlation vs. convolution

Convolution in math textbook:

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

$$\begin{matrix} & * & \begin{matrix} 3 & 4 & 5 \\ 1 & 0 & 2 \\ -1 & 9 & 7 \end{matrix} \\ \begin{matrix} 7 & 2 & 5 \\ 9 & 0 & 4 \\ -1 & 1 & 3 \end{matrix} & \end{matrix}$$

$$= \begin{matrix} & \begin{matrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \end{matrix}$$

$$(A * B) * C = A * (B * C)$$



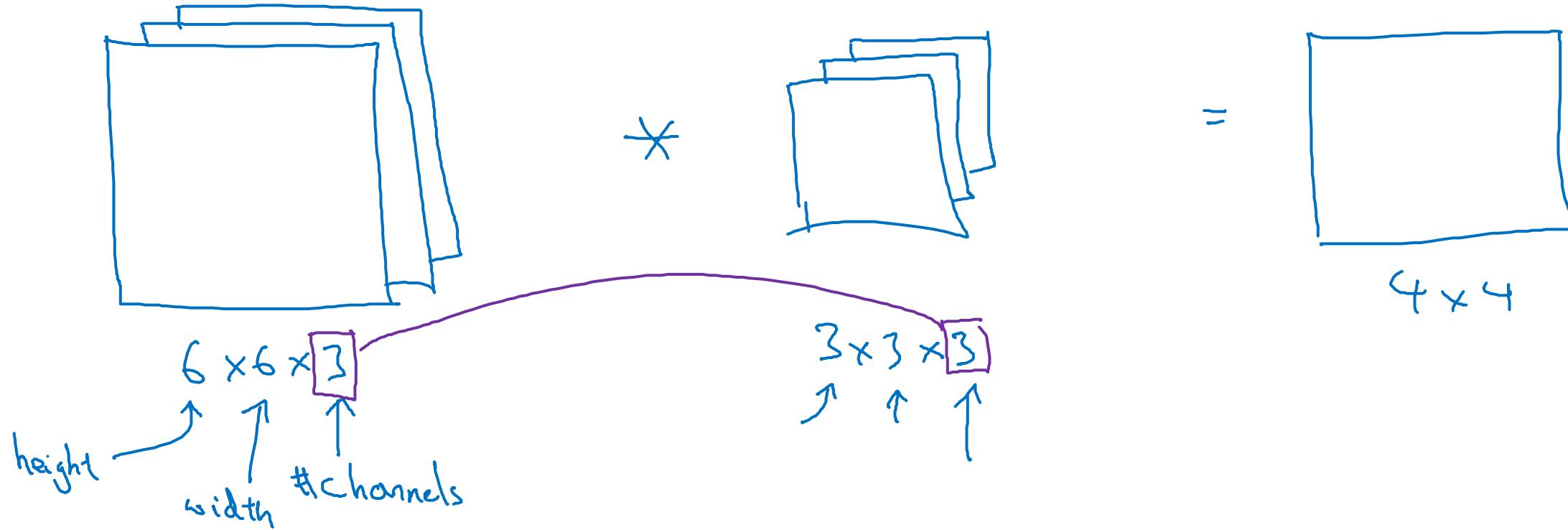
deeplearning.ai

# Convolutional Neural Networks

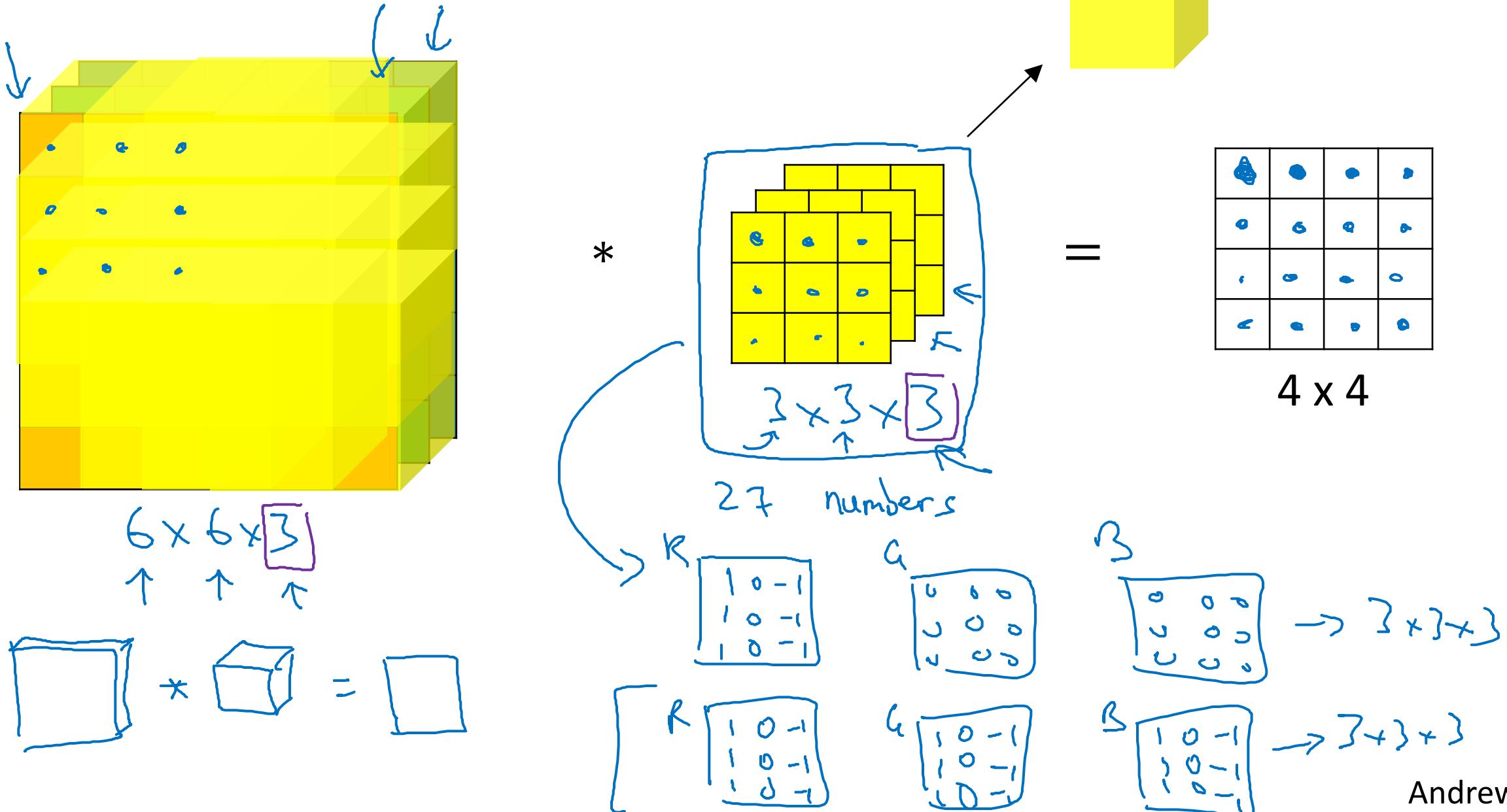
---

## Convolutions over volumes

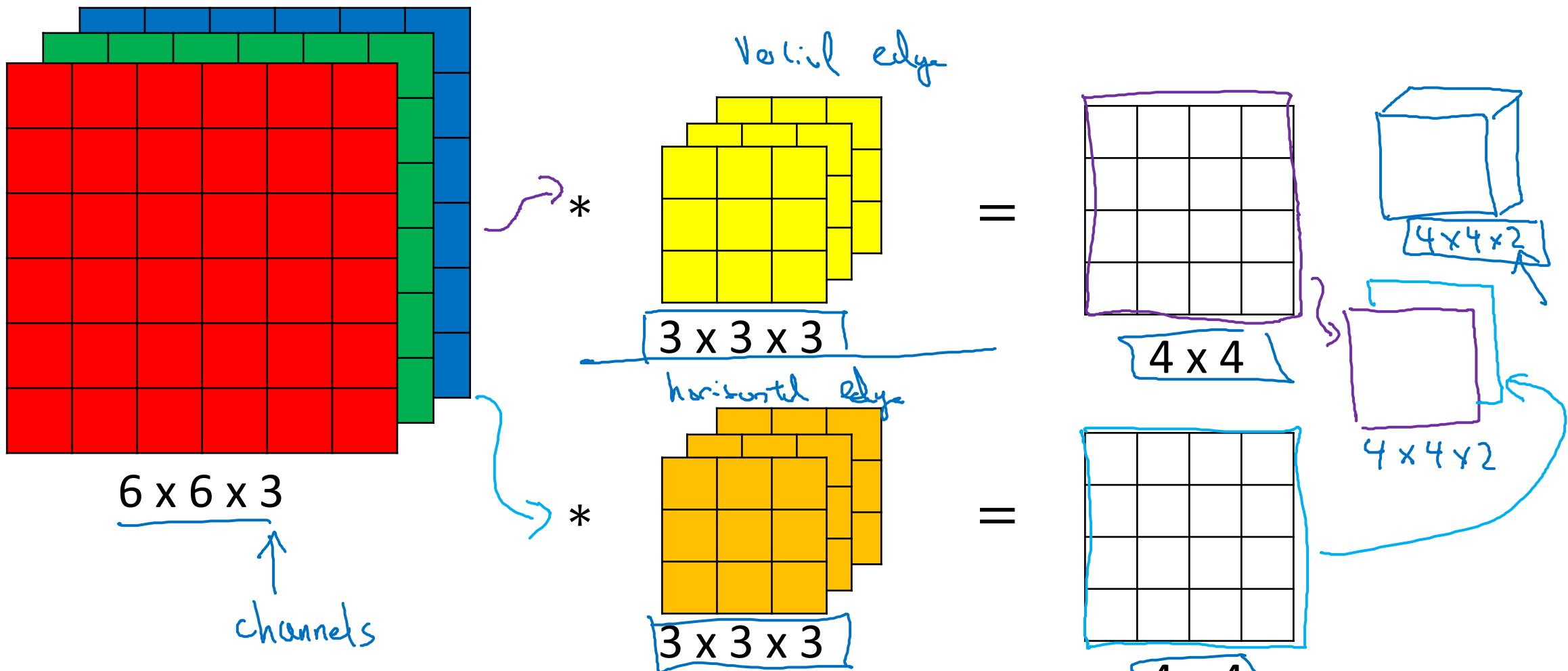
# Convolutions on RGB images



# Convolutions on RGB image



# Multiple filters



Summary:  $n \times n \times n_c$   $\times f \times f \times n_c$   $\rightarrow \frac{n-f+1}{4} \times \frac{n-f+1}{4} \times \frac{n_c}{2} \# \text{filters}$

$6 \times 6 \times 3$   $3 \times 3 \times 3$



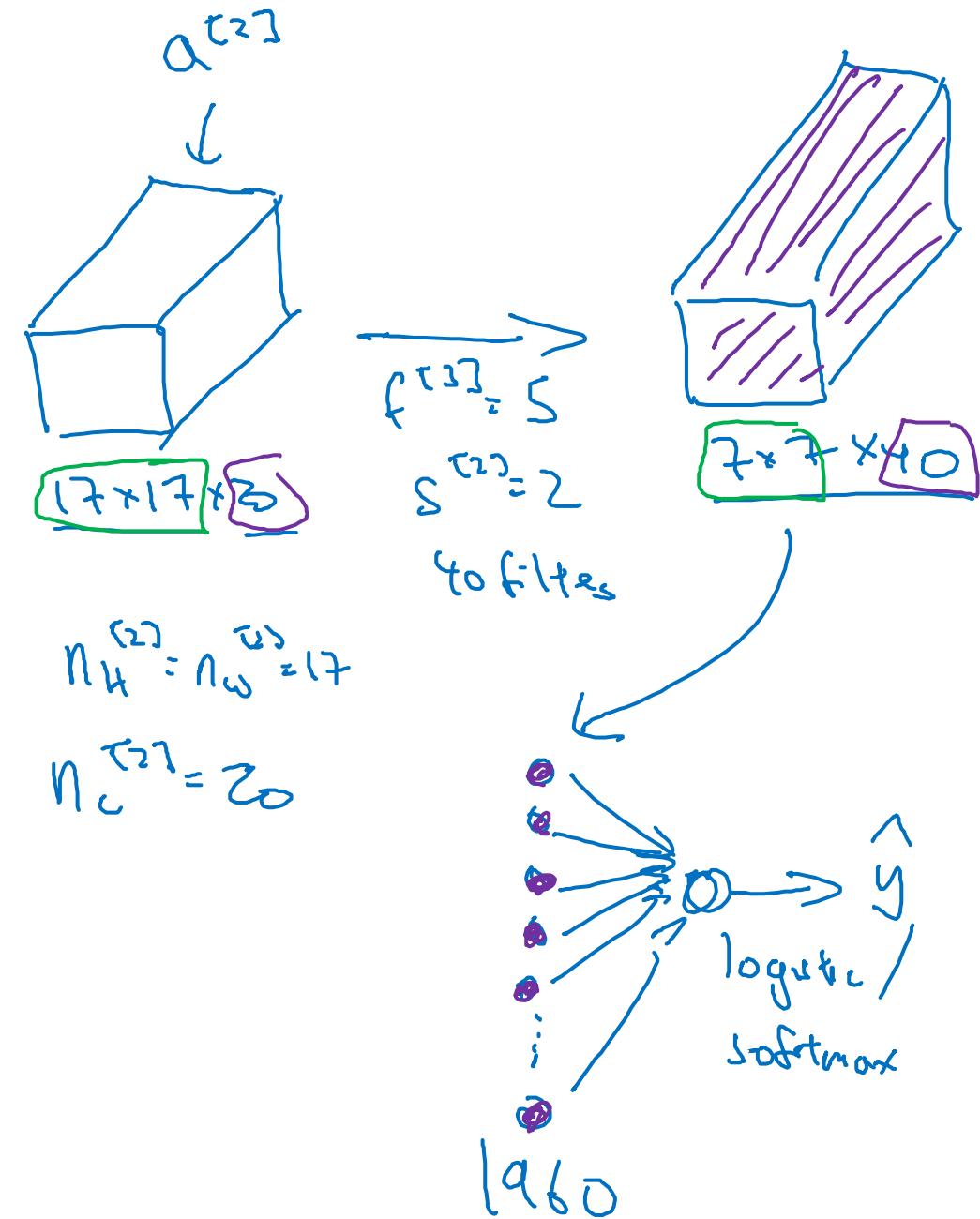
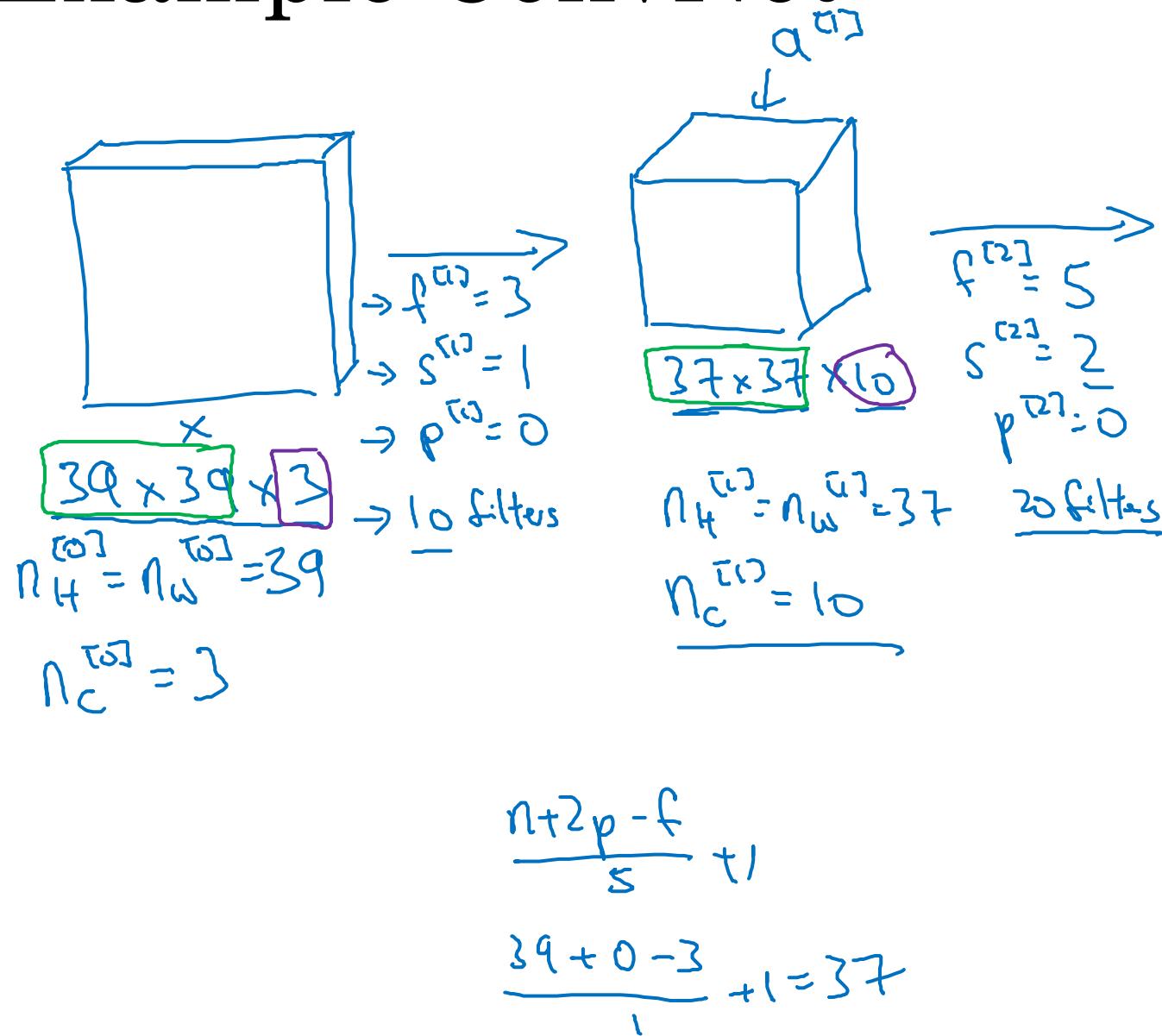
deeplearning.ai

# Convolutional Neural Networks

---

## A simple convolution network example

# Example ConvNet



# Types of layer in a convolutional network:

- Convolution (conv) ←
- Pooling (pool) ←
- Fully connected (Fc) ←



deeplearning.ai

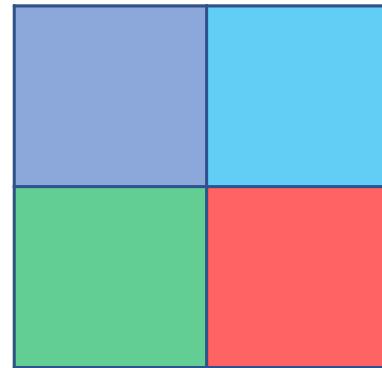
# Convolutional Neural Networks

---

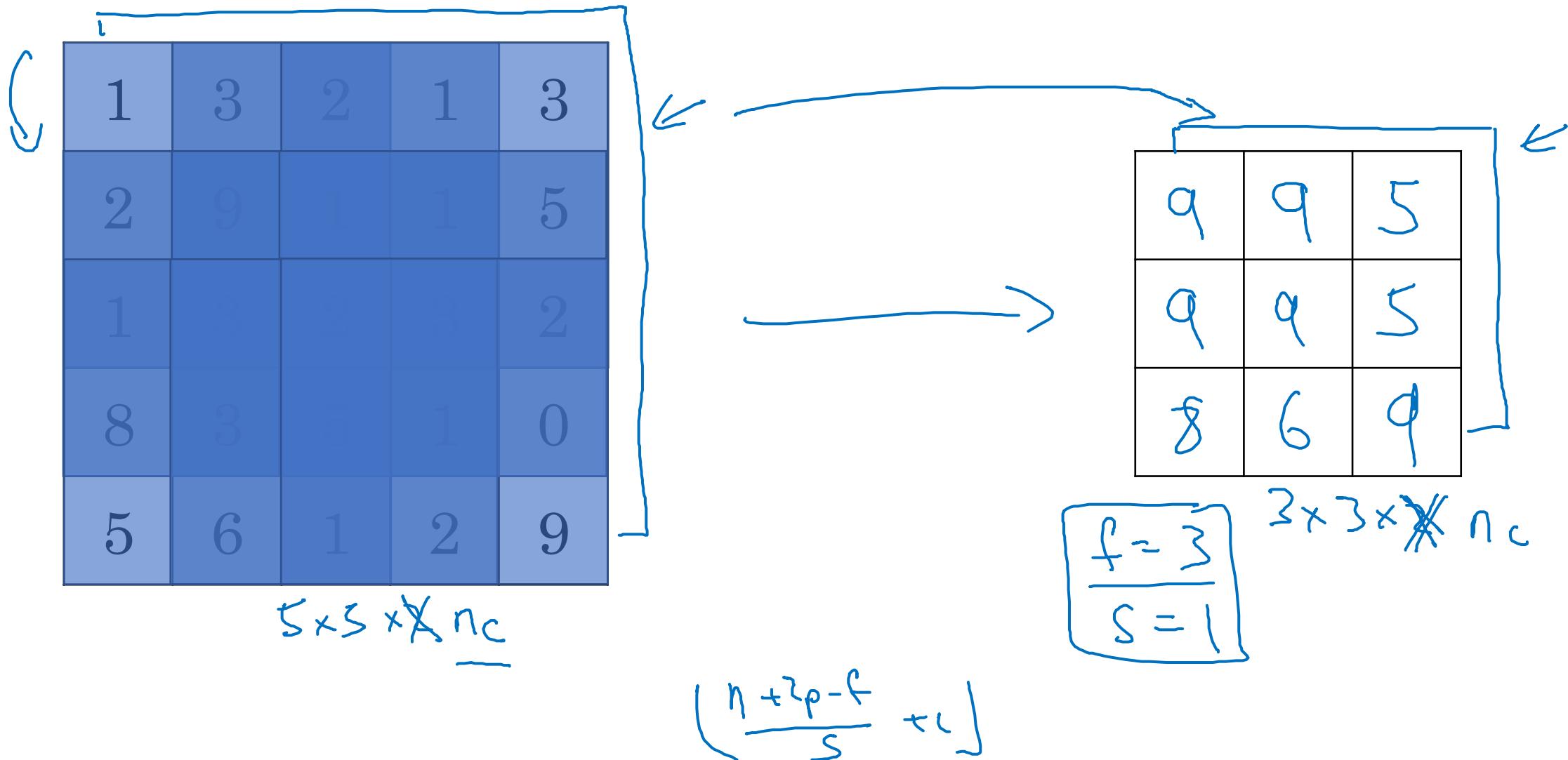
## Pooling layers

# Pooling layer: Max pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2



# Pooling layer: Max pooling



# Pooling layer: Average pooling

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2



3.75	1.25
4	2

$$f=2$$

$$s=2$$

$$\underbrace{7 \times 7 \times 1000}_{\rightarrow} \rightarrow 1 \times 1 \times 1000$$

# Summary of pooling

Hyperparameters:

$f$  : filter size

$$f=2, s=2$$

$s$  : stride

$$f=3, s=2$$

Max or average pooling

$\rightarrow p$ : padding.

No parameters to learn!

$$n_H \times n_w \times n_c$$



$$\left\lfloor \frac{n_H-f+1}{s} \right\rfloor \times \left\lfloor \frac{n_w-f}{s} + 1 \right\rfloor$$

$$\times n_c$$



deeplearning.ai

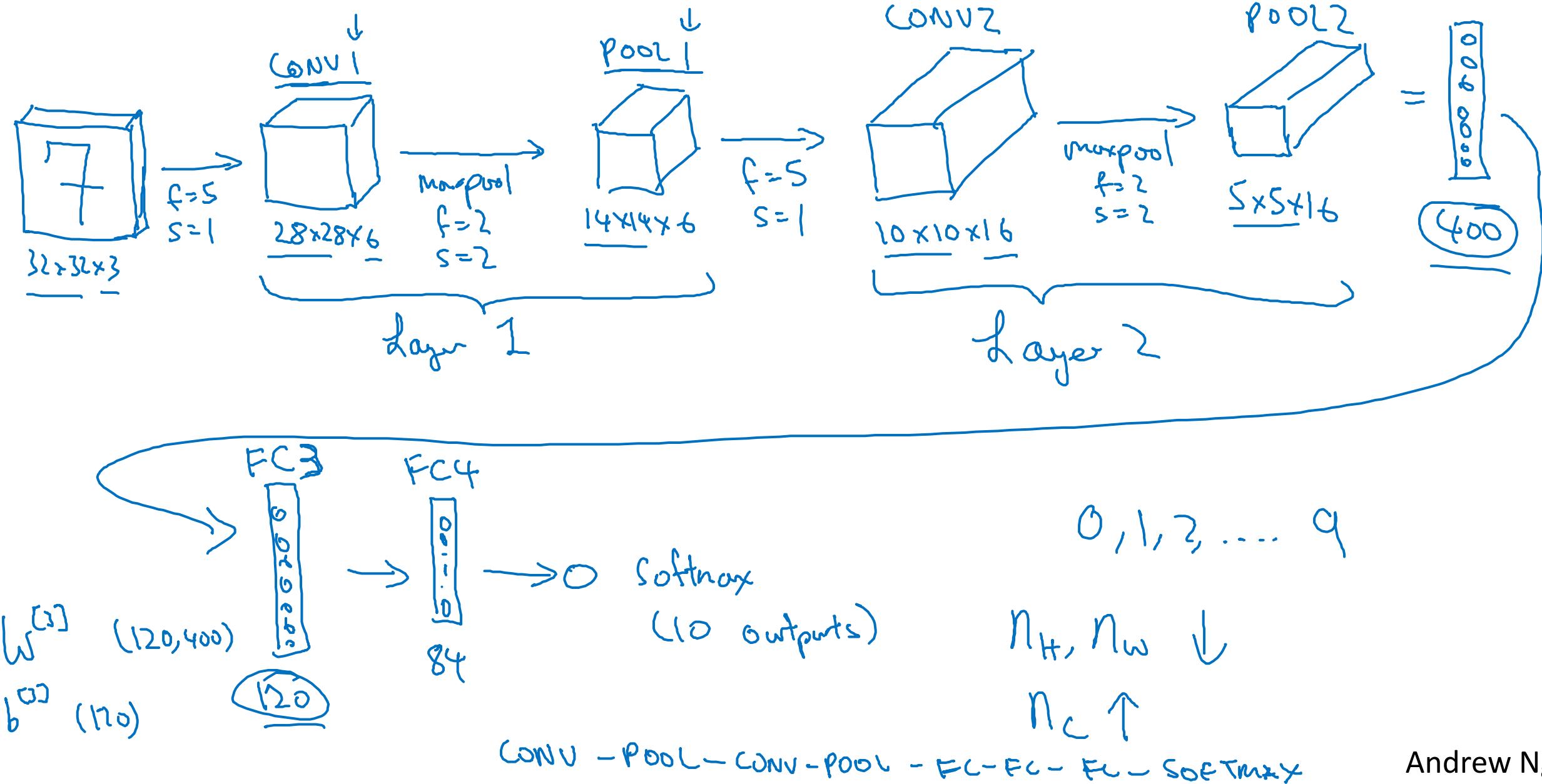
# Convolutional Neural Networks

---

## Convolutional neural network example

# Neural network example

(LeNet-5)



# Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	— 3,072 $a^{[32]}$	0
		—	←
		—	←
		—	←
		—	←
		—	←
		—	—
		—	—
		—	—
		—	—



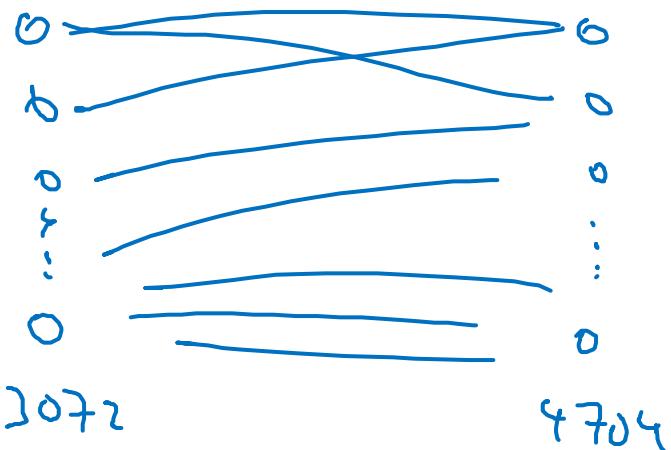
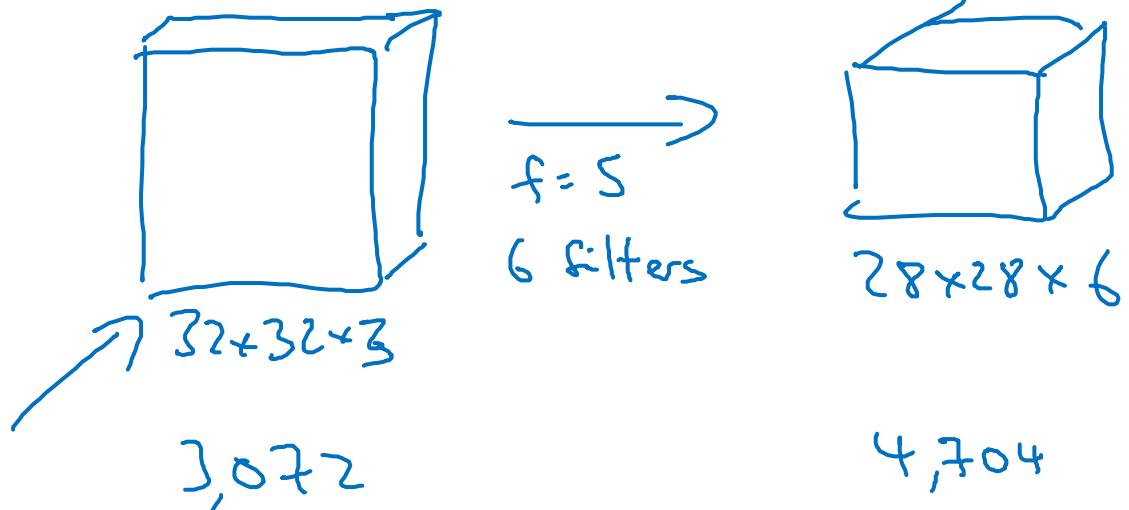
deeplearning.ai

# Convolutional Neural Networks

---

## Why convolutions?

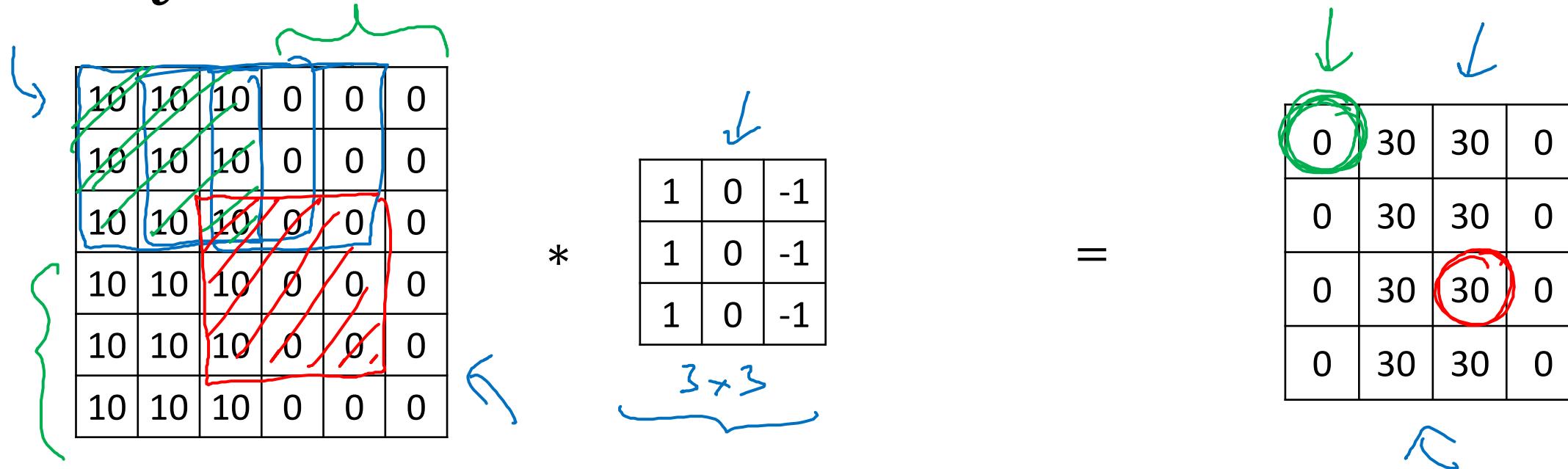
# Why convolutions



$$\begin{aligned} 5 \times 5 & - 25 \\ 26 & \\ 6 \times 26 & = 156 \text{ Parameters} \end{aligned}$$

$$3,072 \times 4,704 \approx 14M$$

# Why convolutions

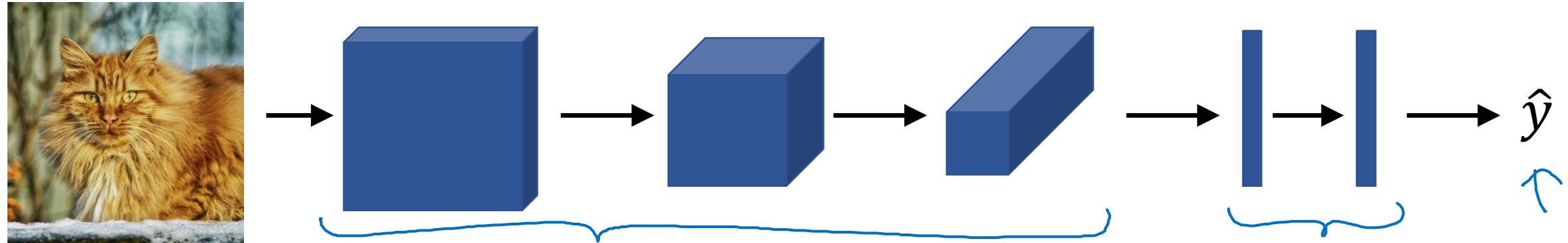


**Parameter sharing:** A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.

# Putting it together

Training set  $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$ .



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Use gradient descent to optimize parameters to reduce  $J$