deeplearning.ai

# Convolutional Neural Networks

## Computer vision

# Computer Vision Problems

## Image Classification

64x64

→ Cat? (0/1)

## Object detection

## Neural Style Transfer

# Deep Learning on large images

64x64 × 3

Cat? (0/1)

12288

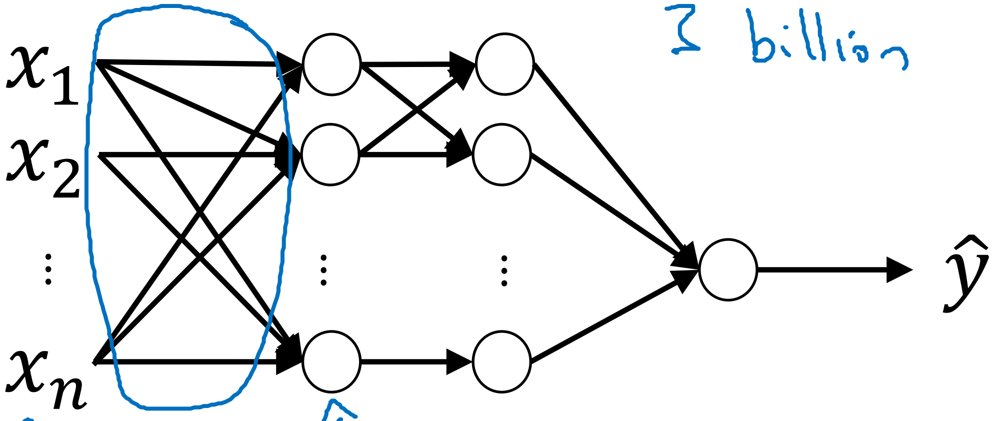
1000 × 1000 × 3

= 3 million

$x \in \mathbb{R}^{3M}$

$W^{[1]}$ (1000, 3M)

3 billion




3M

1000

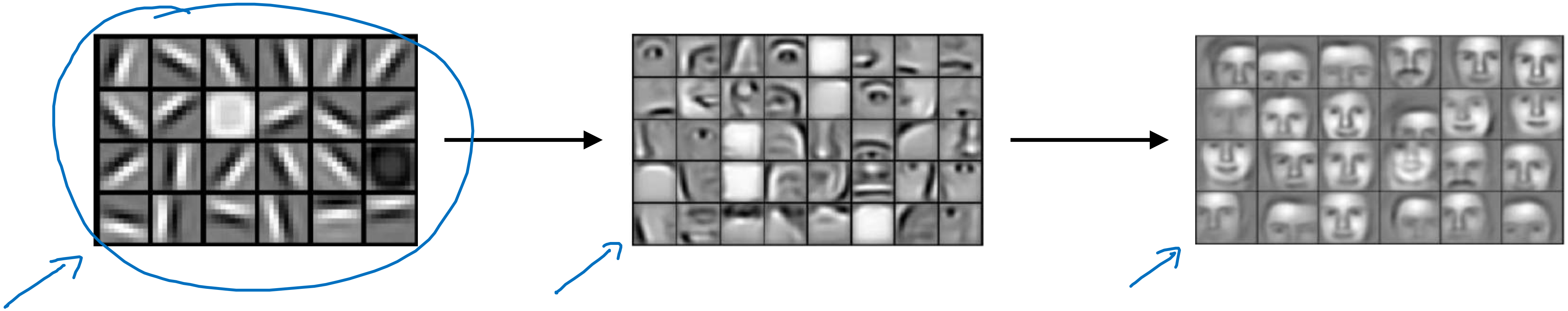
Andrew Ng

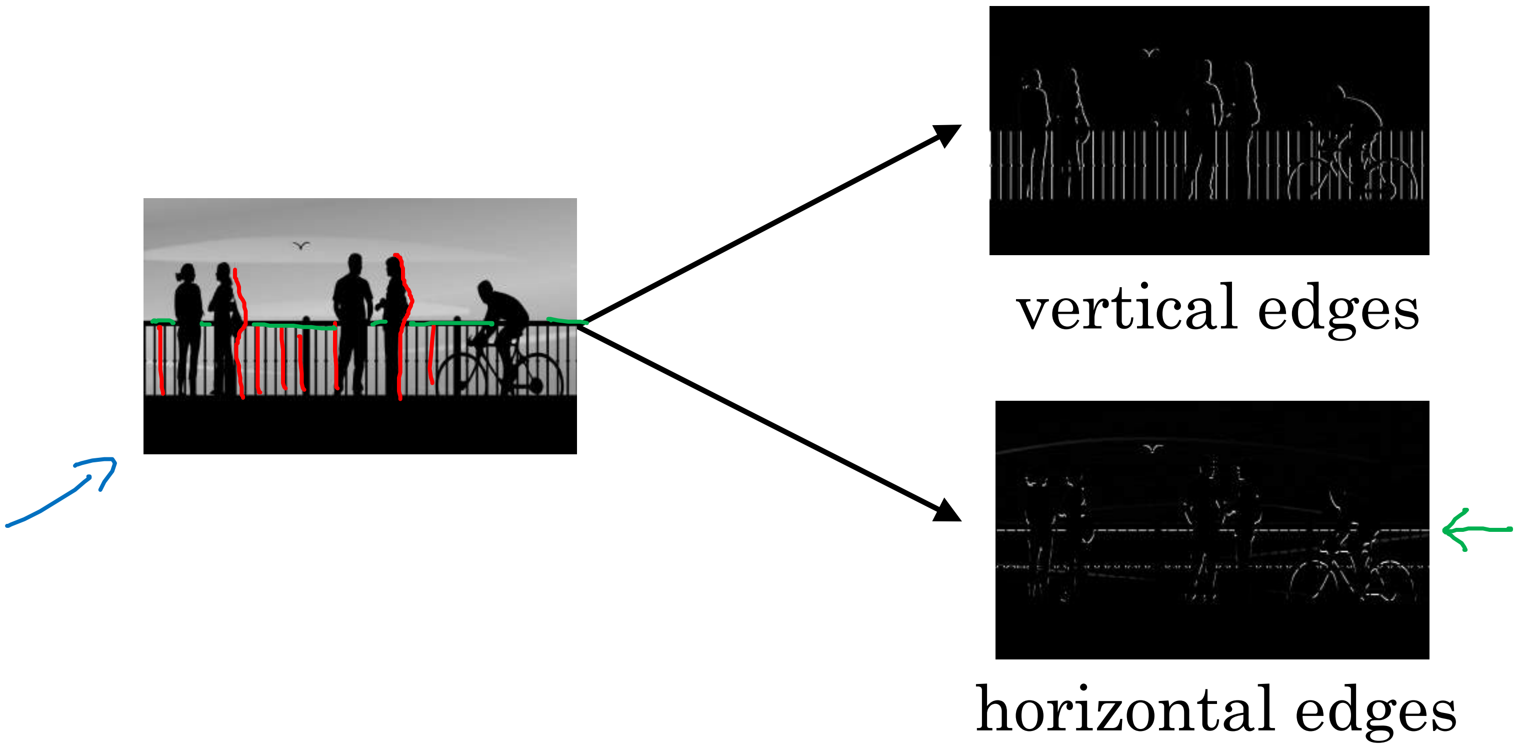
deeplearning.ai

# Convolutional Neural Networks

## Edge detection example

# Computer Vision Problem






Andrew Ng

# Vertical edge detection

$$3\times1 + 1\times1 + 2\times1 + 0\times0 + 5\times0 + 7\times0 + 1\times-1 + 8\times-1 + 2\times-1 = -5$$

| | | | | | |
|---|---|---|---|---|---|
| 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 |

6×6

"convolution"

$*$

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

3×3

→ filter

kernel

=

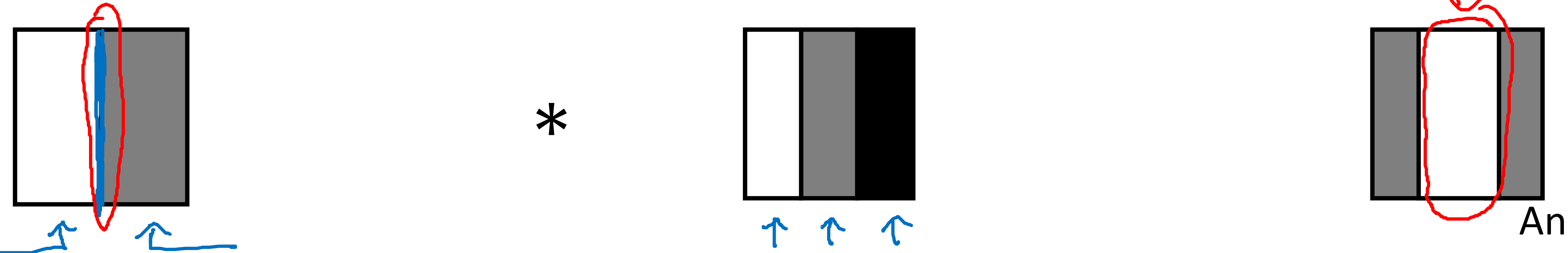
| | | | |
|---|---|---|---|
| -5 | -4 | 0 | 8 |
| -10 | -2 | 2 | 3 |
| 0 | -2 | -4 | -7 |
| -3 | -2 | -3 | -16 |

4×4

Andrew Ng

# Vertical edge detection

$$\underbrace{\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix}}_{6 \times 6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{3 \times 3} = \underbrace{\begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}}_{4 \times 4}$$




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 |

*

| | | |
|---|---|---|
| 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 |

| | | | | | |
|---|---|---|---|---|---|
| 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 |

Andrew Ng

# Vertical and Horizontal Edge Detection

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

Vertical

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| -1 | -1 | -1 |

Horizontal

| | | | | | |
|---|---|---|---|---|---|
| 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 |

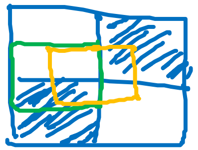
6x6

$*$

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| -1 | -1 | -1 |

$=$

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



Andrew Ng

# Learning to detect edges

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

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

Sobel filter

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

Scharr filter

| | | | | | |
|---|---|---|---|---|---|
| 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 |

convolution

$*$

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

$3 \times 3$

$=$

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

45°
70°
73°

Andrew Ng

# Convolutional Neural Networks

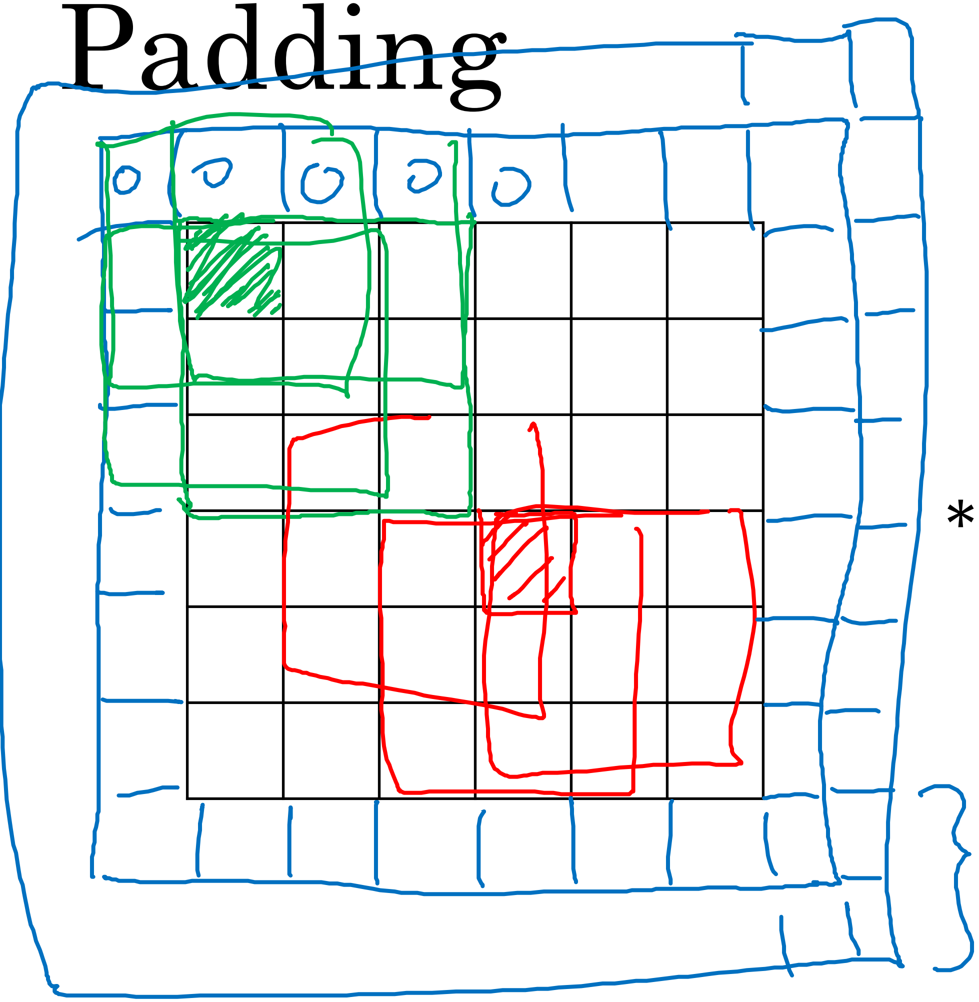
## Padding

deeplearning.ai

# Padding

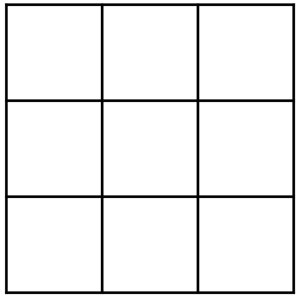
- shrinky output
- throw away info from edge

Padding prior to convolution helps:
- avoiding shrinking activations
- exploiting more the info in the edges of the image (because if we pad, the edge pixels will be used in more many convolution operations).




p=2

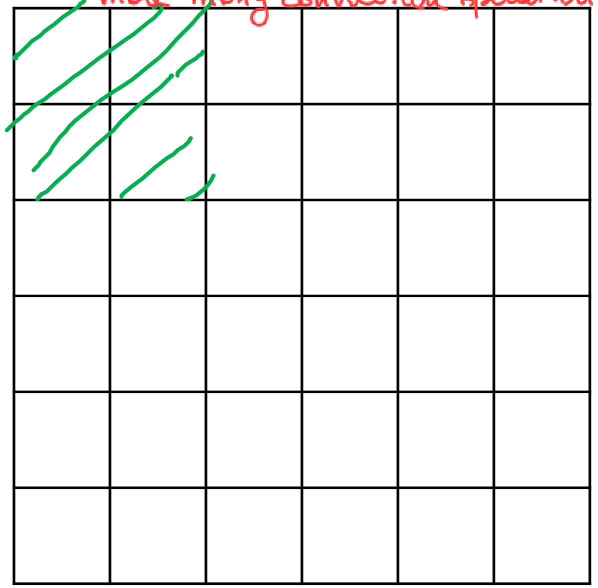
$*$



3x3

f x f

$=$



6x6

~~4x4~~

6x6 → 8x8

n x n

$n-f+1 \times n-f+1$

$6-3+1=4$

p = padding = 1

$n+2p-f+1 \times n+2p-f+1$

$6+2-3+1 \times \_\_ = 6\times6$

Andrew Ng

# Valid and Same convolutions

→ no padding

"Valid": $n \times n \ * \ f \times f \rightarrow n-f+1 \times n-f+1$

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

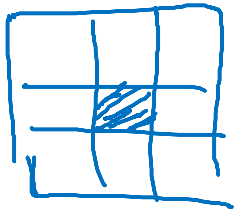
"Same": Pad so that output size is the same as the input size.

$n+2p-f+1 \times n+2p-f+1$

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

$3 \times 3 \quad p = \frac{3-1}{2} = 1$ | $5 \times 5$, $f=5$ $\quad p=2$

f is usually odd



$1 \times 1$
$3 \times 3$
$5 \times 5$
$7 \times 7$

Andrew Ng

# Convolutional Neural Networks

deeplearning.ai

## Strided convolutions

# Strided convolution

- to reduce activation size
- to reduce computational load

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

7×7

*

| | | |
|---|---|---|
| 3 | 4 | 4 |
| 1 | 0 | 2 |
| -1 | 0 | 3 |

3×3

=

| | | |
|---|---|---|
| 91 | 100 | 83 |
| 69 | 91 | 127 |
| 44 | 72 | 74 |

3×3

stride = 2

$\lfloor z \rfloor = \text{floor}(z)$

$n \times n \quad * \quad f \times f$

padding $p$ stride $s$

$s = 2$

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

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

Andrew Ng

# Summary of convolutions

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

padding $p$ $\qquad$ stride $s$

Output size:

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

Andrew Ng

# Technical note on cross-correlation vs. convolution

## Convolution in math textbook:

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

*

| | | |
|---|---|---|
| 3 | 4 | 5 |
| 1 | 0 | 2 |
| -1 | 9 | 7 |

=

| | | |
|---|---|---|
| 7 | 2 | 5 |
| 9 | 0 | 4 |
| -1 | 1 | 3 |

- In mathematics, the convolution operation requires preliminarily flipping the filter. In ML this step is omitted and what is called "convolution" is actually a "cross-correlation" between 2 matrices.

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

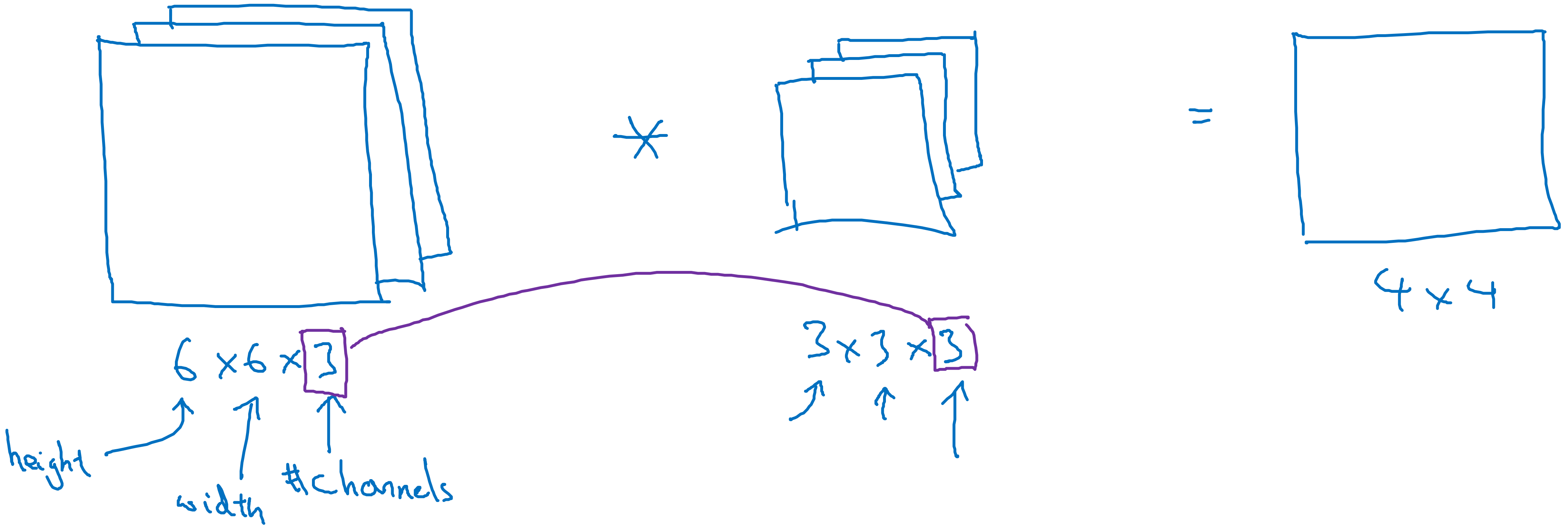
the filter's mirroring, which we don't use in ML is used to have this property, useful in signal processing but not necessarily in ML.

deeplearning.ai

# Convolutional Neural Networks

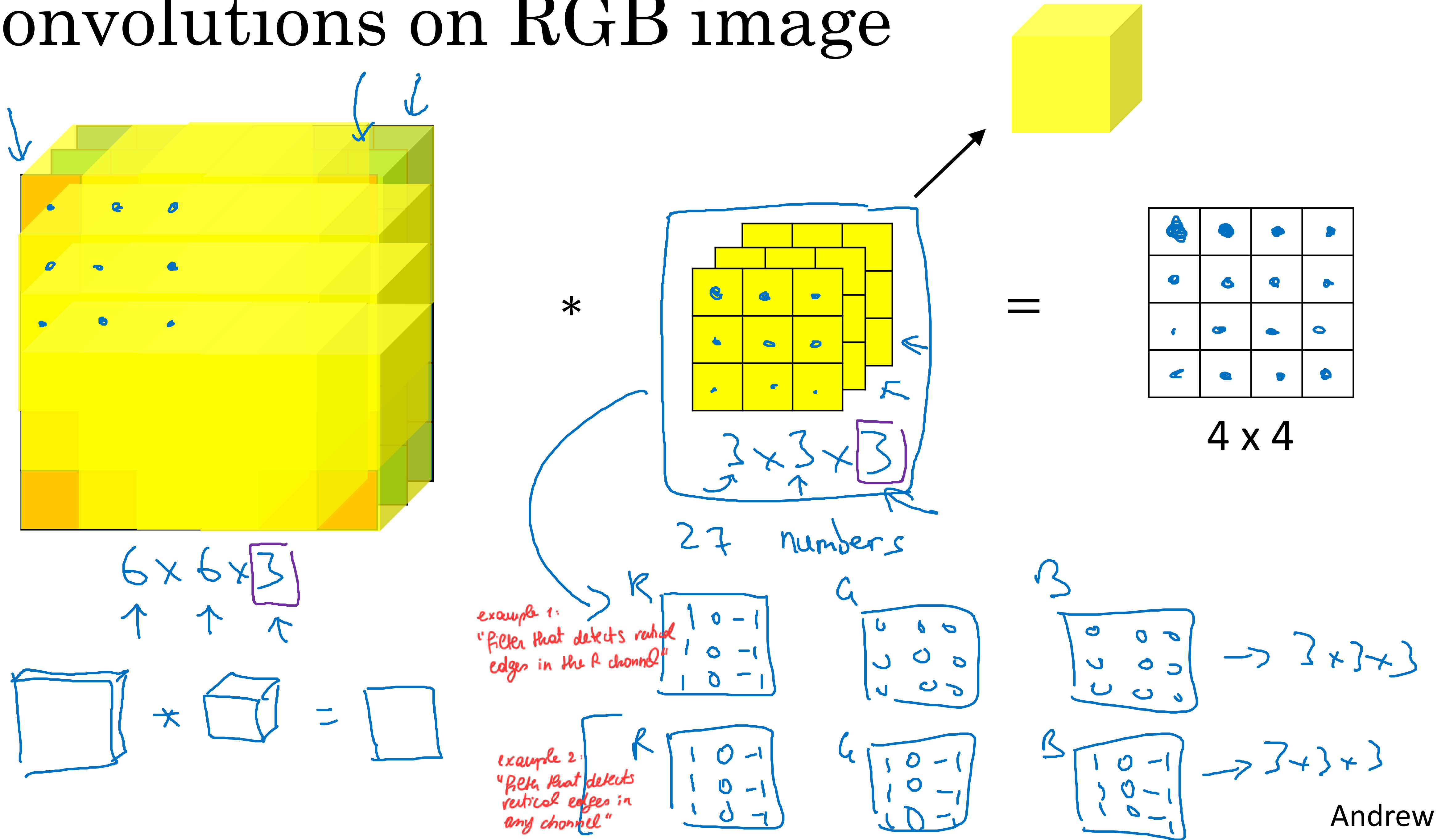
## Convolutions over volumes

# Convolutions on RGB images



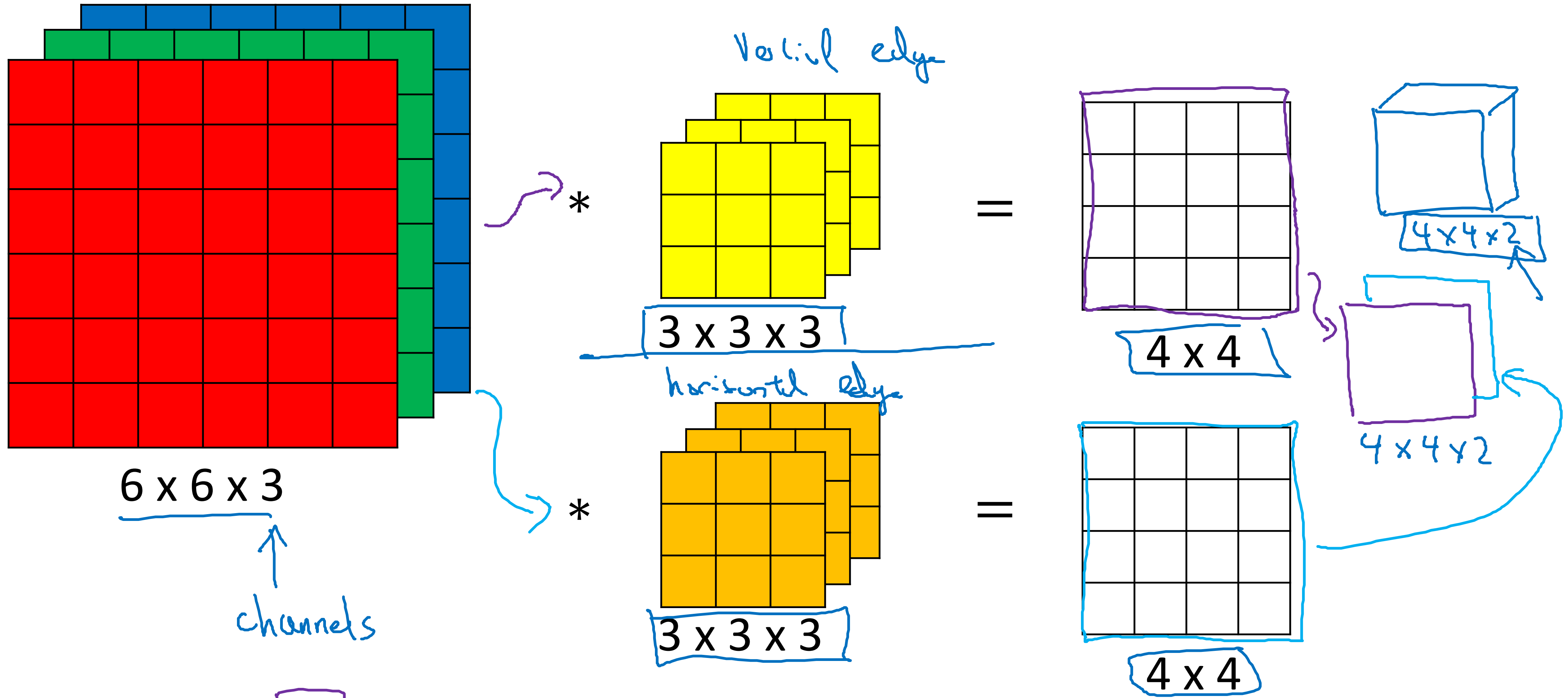

Andrew Ng

# Convolutions on RGB image




Andrew Ng

# Multiple filters




Summary: $n \times n \times \boxed{n_c} \ * \ f \times f \times \boxed{n_c} \longrightarrow \underline{n-f+1} \times \underline{n-f+1} \times n_c'$

$6 \times 6 \times 3 \qquad 3 \times 3 \times 3 \qquad 4 \times 4 \times 2$ ← # filters

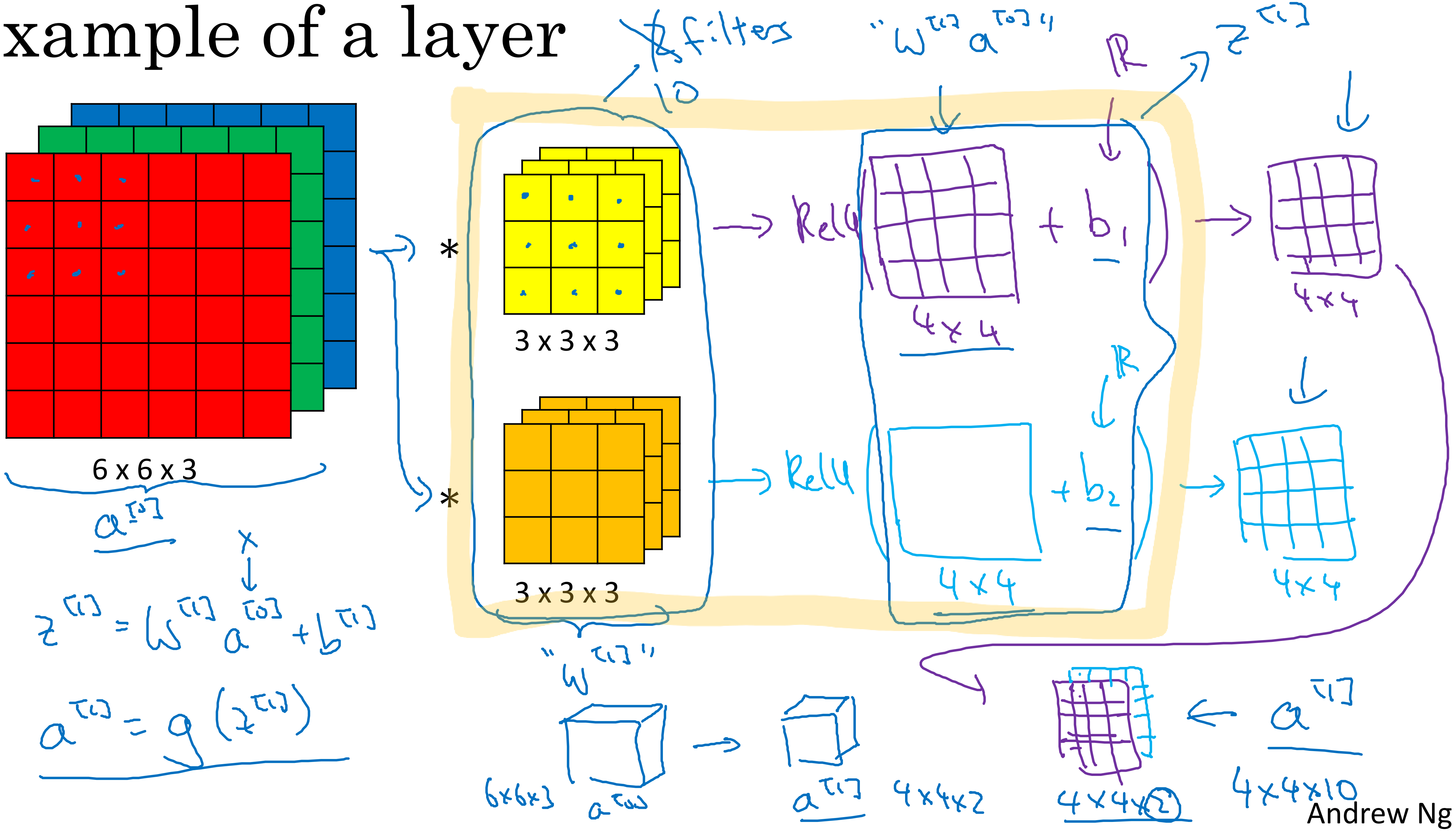
Andrew Ng

deeplearning.ai

Convolutional Neural Networks

# One layer of a convolutional network

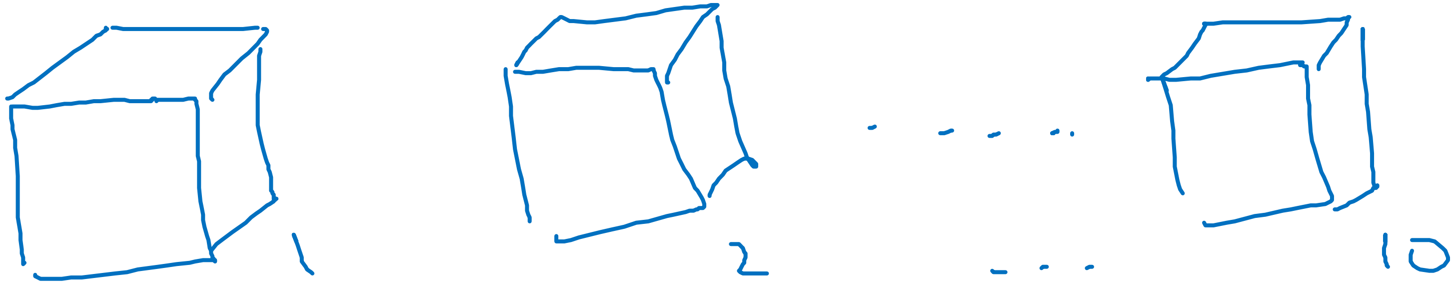
# Example of a layer




Andrew Ng

# Number of parameters in one layer

If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?




3x3x3

27 parameters.

+bias

→ 28 paramters.

280 parameters.

Andrew Ng

# Summary of notation

## If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$\boxed{n_c^{[l]}}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times \boxed{n_c^{[l-1]}}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $n_c^{[l]}$ — $(1,1,1,n_c^{[l]})$ ← #filters in layer l.

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times \boxed{n_c^{[l-1]}}$ ←

Output: $n_H^{[l]} \times n_W^{[l]} \times \boxed{n_c^{[l]}}$ ←

$$n_{\cancel{H}W}^{[l]} = \left\lfloor \frac{n_{\cancel{H}W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

#examples

$A^{[l]} \rightarrow \underline{m \times \underbrace{n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}}}$

$\underline{n_c} \times n_H \times n_W$

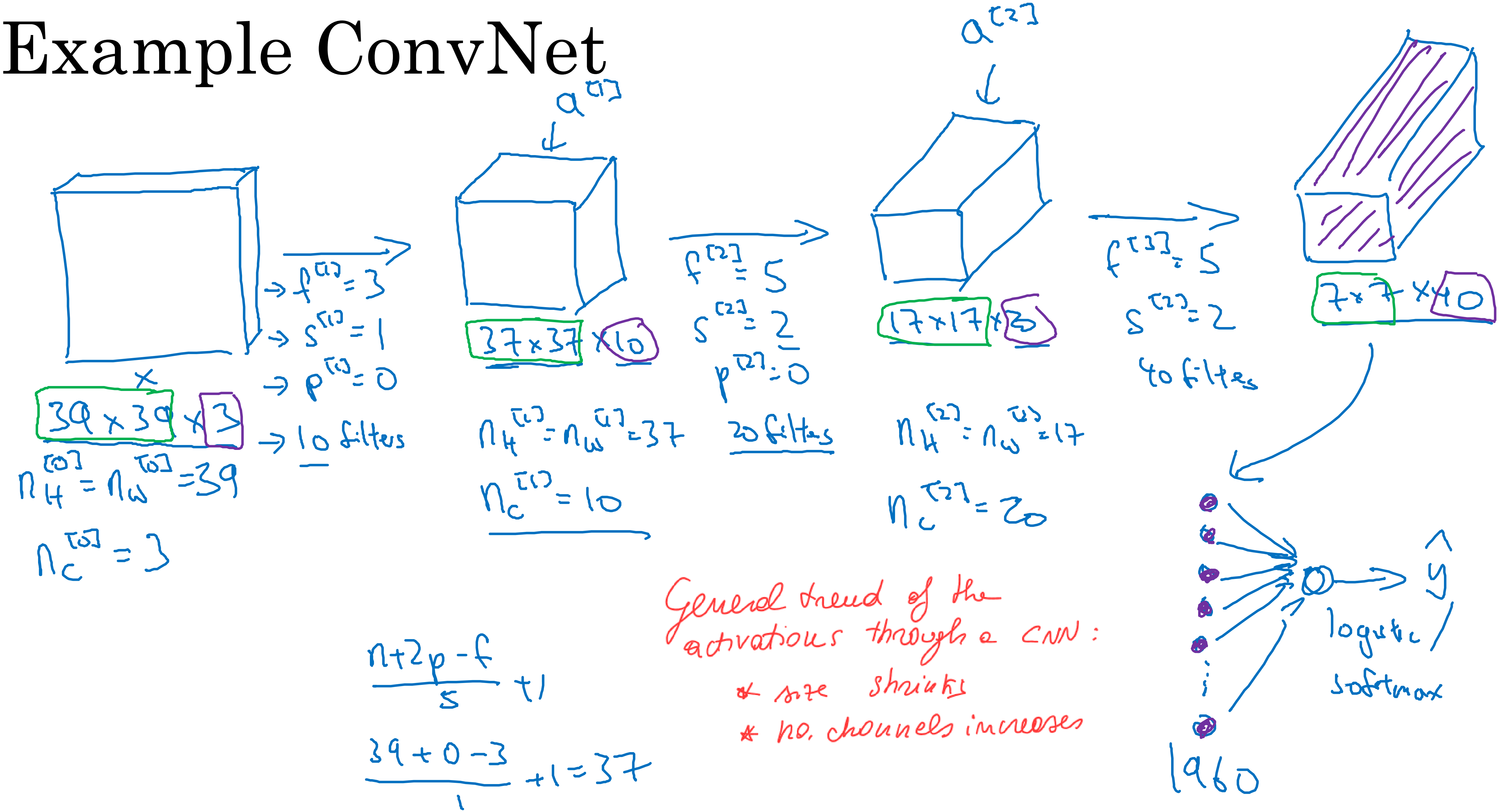
Andrew Ng

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

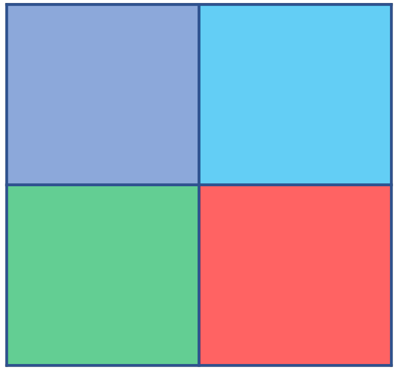
## Pooling layers

# Pooling layer: Max pooling

Shrinks activation maps by only retaining important info.

INTUITION: the activation of a filter indicates how much the detected feature appears in the input image.

| | | | |
|---|---|---|---|
| 1 | 3 | 2 | 1 |
| 2 | 9 | 1 | 1 |
| 1 | 3 | 2 | 3 |
| 5 | 6 | 1 | 2 |



By maxpooling the activation we discard some information while retaining <u>where</u> in the image the feature is more prominent.

- performed on each channel independently
- no parameters to learn!

# Pooling layer: Max pooling

| | | | | |
|---|---|---|---|---|
| 1 | 3 | 2 | 1 | 3 |
| 2 | 9 | 1 | 1 | 5 |
| 1 | 3 | 2 | 3 | 2 |
| 8 | 3 | 5 | 1 | 0 |
| 5 | 6 | 1 | 2 | 9 |

$5 \times 5 \times \cancel{1}\, n_c$

| | | |
|---|---|---|
| 9 | 9 | 5 |
| 9 | 9 | 5 |
| 8 | 6 | 9 |

$3 \times 3 \times \cancel{1}\, n_c$

$$\boxed{\begin{matrix} f = 3 \\ s = 1 \end{matrix}}$$

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

# Pooling layer: Average pooling

| | | | |
|---|---|---|---|
| 1 | 3 | 2 | 1 |
| 2 | 9 | 1 | 1 |
| 1 | 4 | 2 | 3 |
| 5 | 6 | 1 | 2 |

$\longrightarrow$

| | |
|---|---|
| 3.75 | 1.25 |
| 4 | 2 |

$f = 2$

$s = 2$

$\underline{7 \times 7} \times 1000 \longrightarrow 1 \times 1 \times 1000$

# Summary of pooling

Hyperparameters:

f : filter size

$f=2, s=2$

$f=3, s=2$

s : stride

Max or average pooling

→ ~~p: padding.~~

No parameters to learn!

$n_H \times n_W \times n_C$

$\downarrow$

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

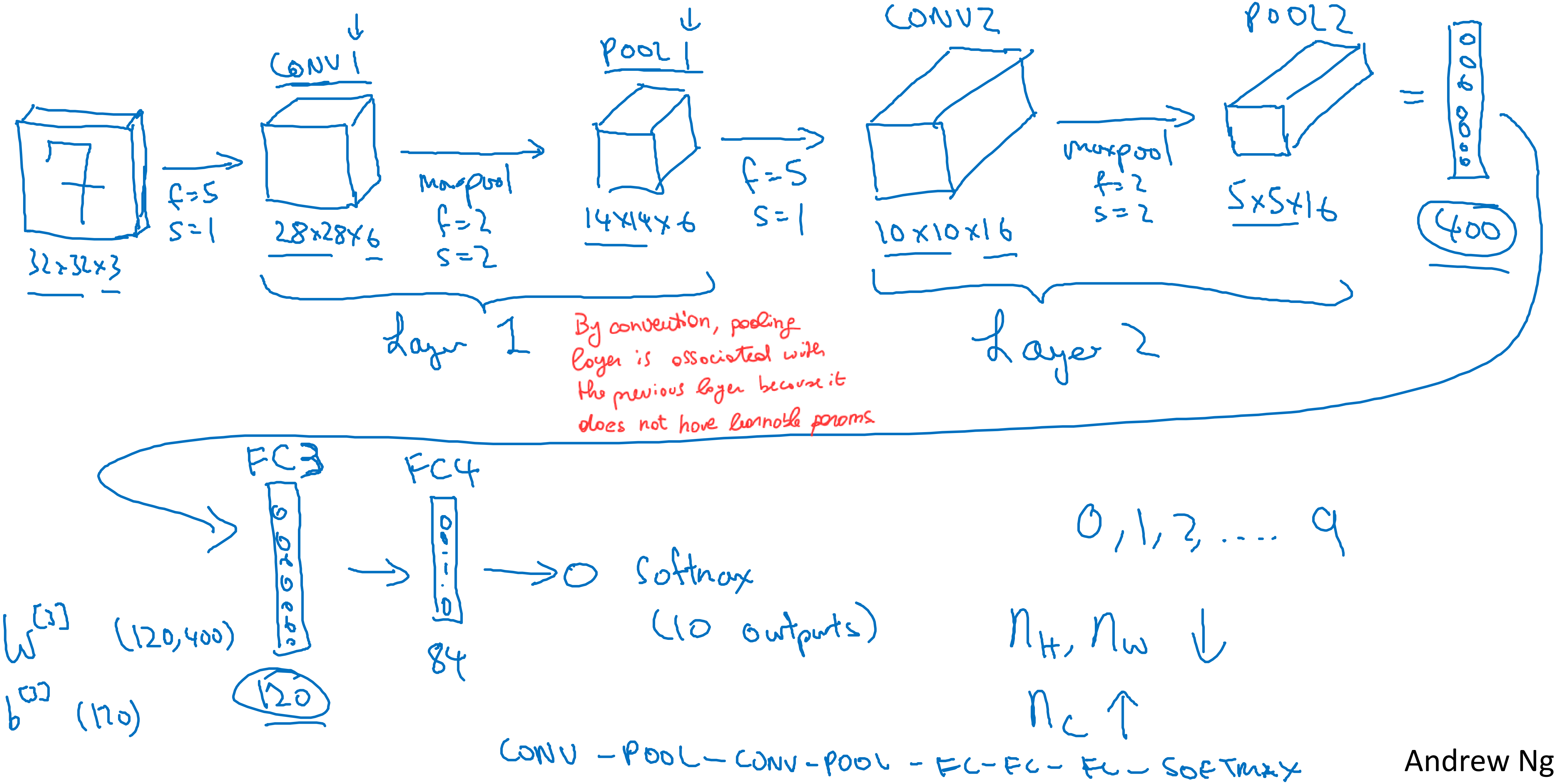
$\times n_C$

deeplearning.ai

# Convolutional Neural Networks

## Convolutional neural network example

# Neural network example (LeNet-5)




Andrew Ng

# Neural network example

| | Activation shape | Activation Size | # parameters |
|---|---|---|---|
| Input: | (32,32,3) | 3,072 $a^{[0]}$ | 0 |
| CONV1 (f=5, s=1) | 8 filters of size 5x5x3 (28,28,8) | 6,272 | 608 |
| POOL1 | (14,14,8) | 1,568 | 0 |
| CONV2 (f=5, s=1) | (10,10,16) | 1,600 | 3216 |
| POOL2 | (5,5,16) | 400 | 0 |
| FC3 | (120,1) | 120 | 48120 |
| FC4 | (84,1) | 84 | 10164 |
| Softmax | (10,1) | 10 | 850 |

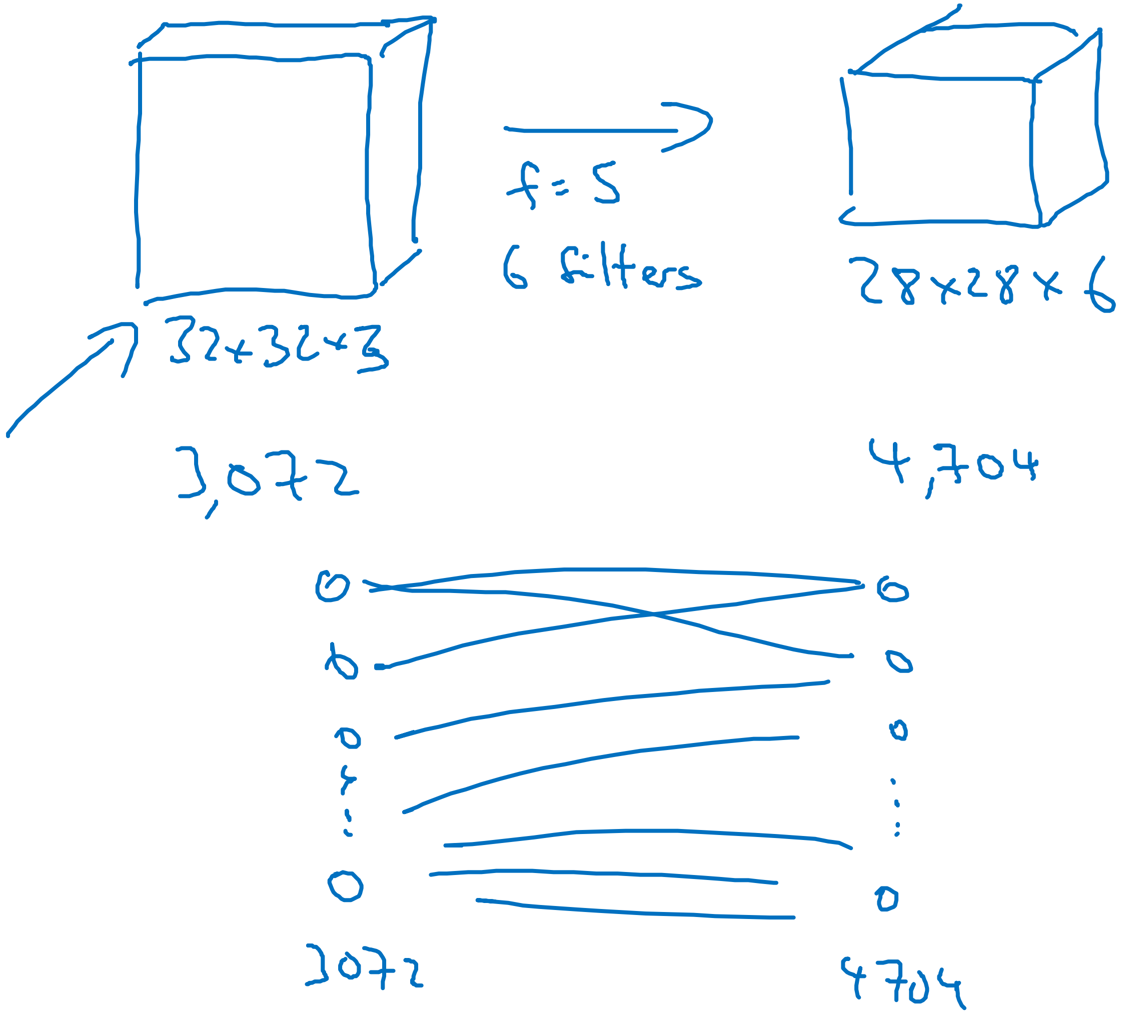
Andrew Ng

deeplearning.ai

# Convolutional Neural Networks

## Why convolutions?

# Why convolutions




$5 \times 5 - 25$

$26$

$6 \times 26 = 156$ parameters

$3{,}072 \times 4{,}704 \approx \underline{14M}$

# Why convolutions

translation ~~invariance~~ invariance

$$
\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{3\times 3} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}
$$

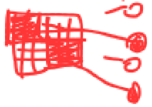
**Parameter sharing:** A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image. (As opposed to a FC layer connected to an image, which would probably relearn the same feature multiple times in different units).

↳ => also favors translation invariance. & reduces overfitting.

SPATIAL LOCALITY

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

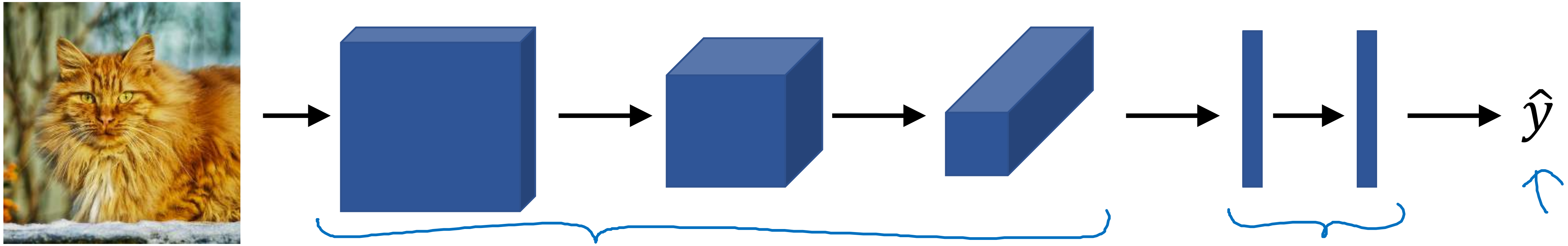
FC



# Putting it together

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

$\boxed{w, b}$




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

Use gradient descent to optimize parameters to reduce $J$

Andrew Ng