



deeplearning.ai

Convolutional Neural Networks

Computer vision

Computer Vision Problems

Image Classification



64x64

→ Cat? (0/1)

Object detection

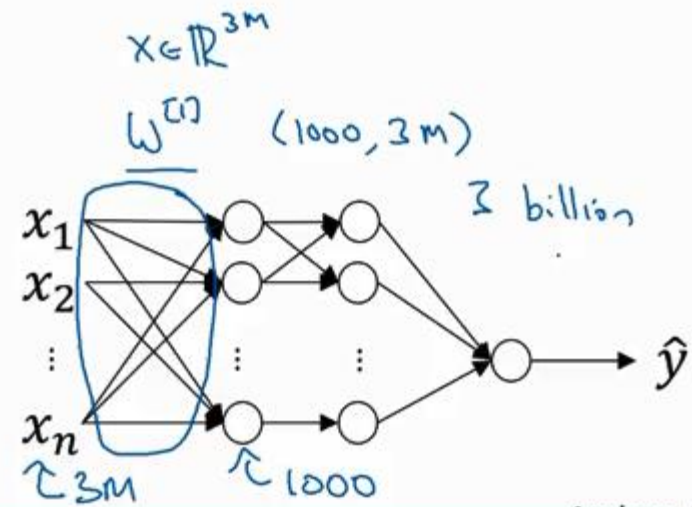
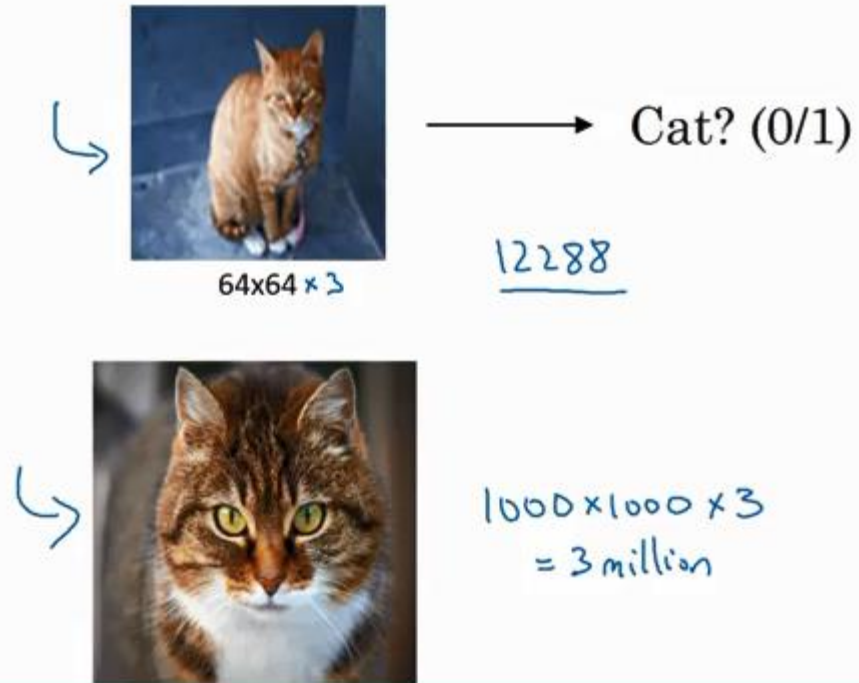


Neural Style Transfer ↙



Andrew Ng

Deep Learning on large images



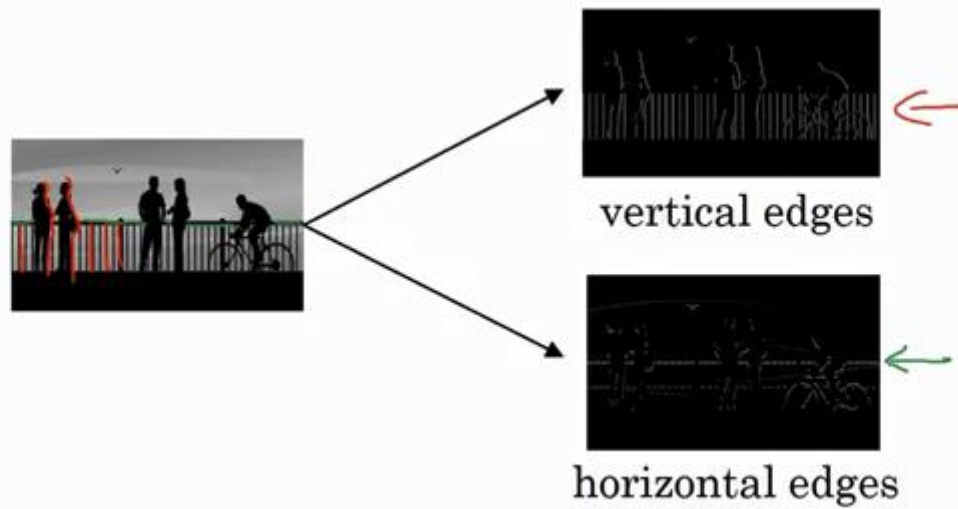
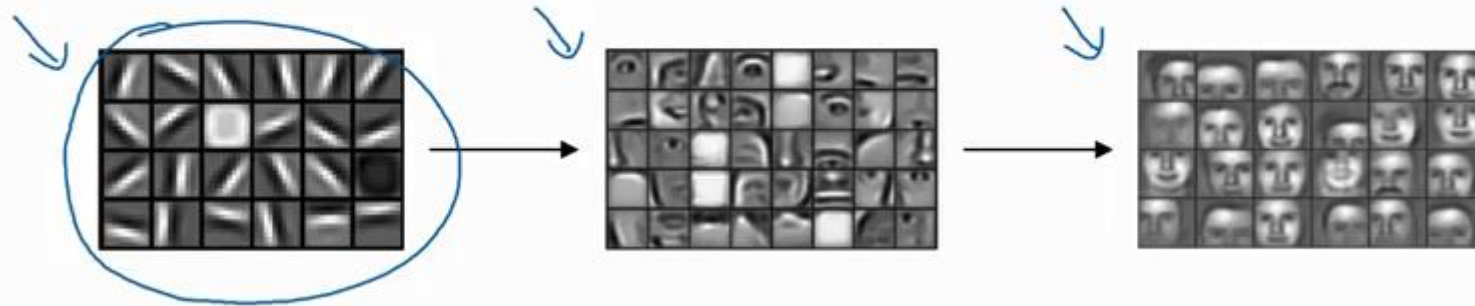


deeplearning.ai

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	<u>0</u>	<u>1</u>	<u>2</u>	<u>7</u>	<u>4</u>
1	<u>5</u>	<u>8</u>	<u>9</u>	<u>3</u>	<u>1</u>
2	<u>7</u>	<u>2</u>	<u>5</u>	<u>1</u>	<u>3</u>
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6

"convolution"
*

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

3x3
filter

=

<u>-5</u>	<u>-4</u>	0	<u>8</u>
<u>-10</u>	-2	2	3
0	-2	-4	-7
-3	-2	-3	<u>-16</u>

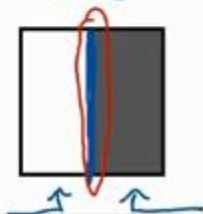
4x4

python: conv-forward
tensorflow: tf.nn.conv2d
keras: Conv2D

Vertical edge detection

$$\begin{array}{|c|c|c|c|c|c|}
 \hline
 10 & 10 & 10 & 0 & 0 & 0 \\
 \hline
 10 & 10 & 10 & 0 & 0 & 0 \\
 \hline
 10 & 10 & 10 & 0 & 0 & 0 \\
 \hline
 10 & 10 & 10 & 0 & 0 & 0 \\
 \hline
 10 & 10 & 10 & 0 & 0 & 0 \\
 \hline
 10 & 10 & 10 & 0 & 0 & 0 \\
 \hline
 \end{array}$$

6x6



*

$$\begin{array}{|c|c|c|}
 \hline
 1 & 0 & -1 \\
 \hline
 1 & 0 & -1 \\
 \hline
 1 & 0 & -1 \\
 \hline
 \end{array}$$

3x3

*



=

$$\begin{array}{|c|c|c|c|}
 \hline
 0 & 30 & 30 & 0 \\
 \hline
 0 & 30 & 30 & 0 \\
 \hline
 0 & 30 & 30 & 0 \\
 \hline
 0 & 30 & 30 & 0 \\
 \hline
 \end{array}$$

4x4



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

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

Learning to detect edges

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



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

→

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

Sobel filter



convolution
*

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

3x3

=
45°
70°
73°

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

Scharr filter



Andrew Ng



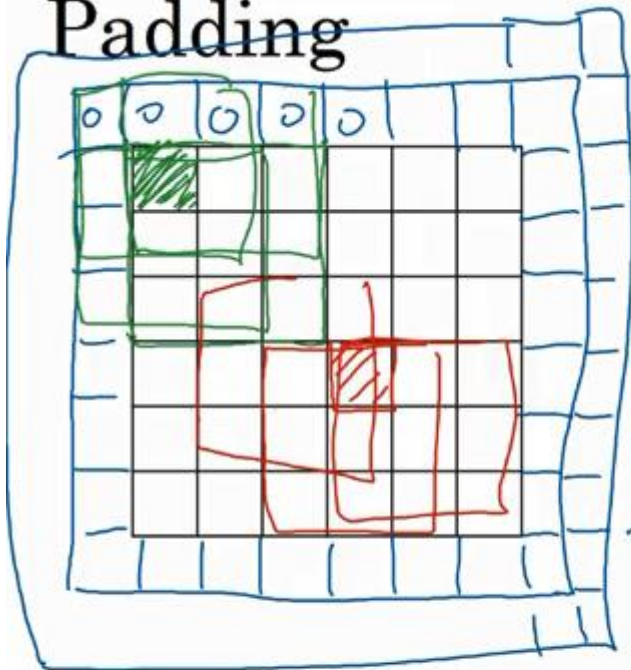
deeplearning.ai

Convolutional Neural Networks

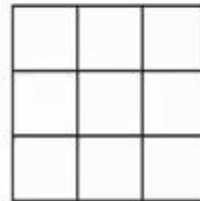
Padding

Padding

- shrinky output
- throw away info from edge

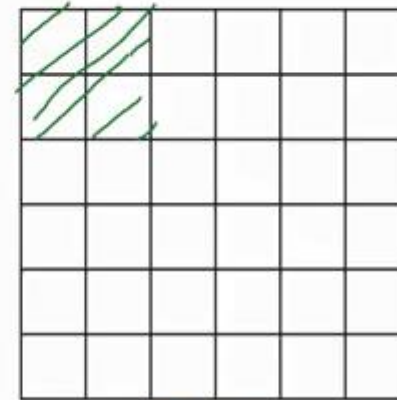


*



3x3
f x f

=



6x6

6x6 → 8x8
n x n

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

$p = \text{padding} = \underline{1}$

$n + 2p - f + 1 \times n + 2p - f + 1$
 $6 + 2 - 3 + 1 \times \underline{\quad} = 6 \times 6$

Andrew Ng

Valid and Same convolutions

→ no padding

“Valid”: $n \times n \quad * \quad f \times f \quad \rightarrow \quad \underline{n - f + 1} \times n - f + 1$
 $6 \times 6 \quad * \quad 3 \times 3 \quad \rightarrow \quad 4 \times 4$

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

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

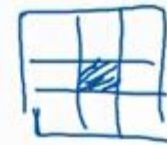
$$\cancel{n + 2p - f + 1} = \cancel{n} \Rightarrow \boxed{p = \frac{f - 1}{2}}$$

$$3 \times 3 \quad p = \frac{3 - 1}{2} = 1$$

$$5 \times 5 \quad f = 5$$

$$p = 2$$

f is usually odd



1x1
3x3
5x5
7x7

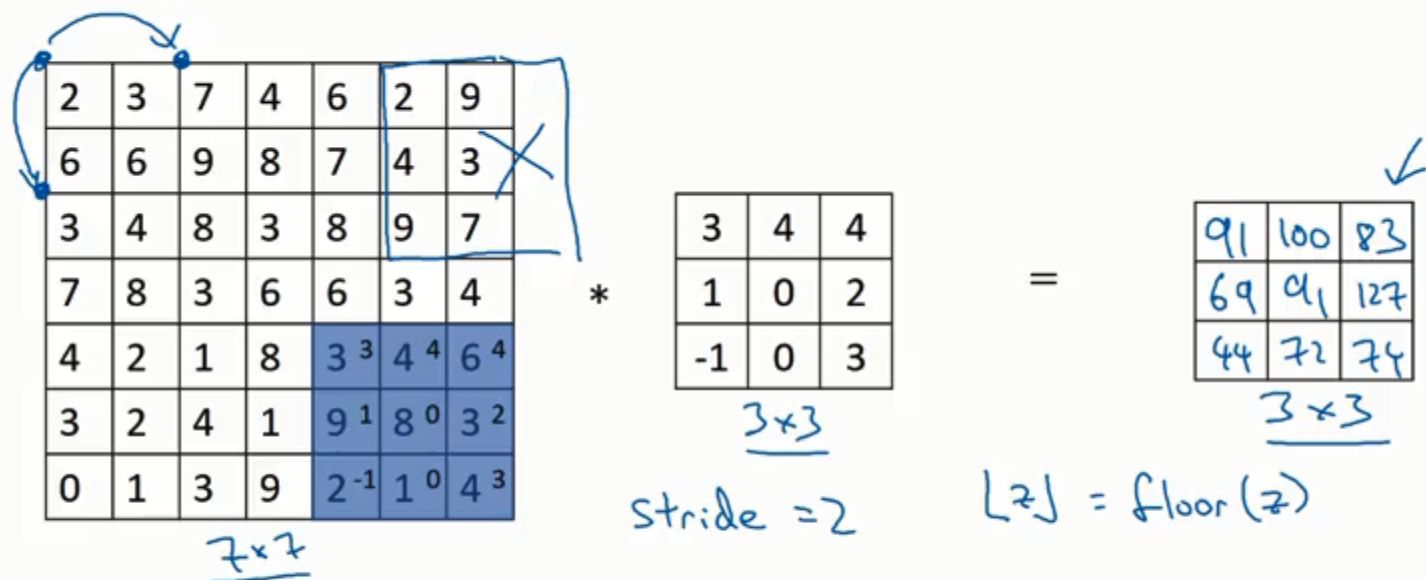


deeplearning.ai

Convolutional Neural Networks

Strided convolutions

Strided convolution



$$\begin{array}{l}
 n \times n \quad * \quad f \times f \\
 \text{padding } p \quad \text{stride } s \\
 s = 2
 \end{array}$$

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

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

Summary of convolutions

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

padding p stride s

Output size:

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

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

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

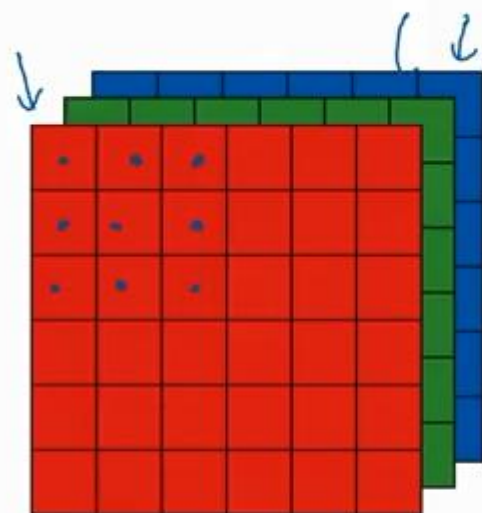


deeplearning.ai

Convolutional Neural Networks

Convolutions over volumes

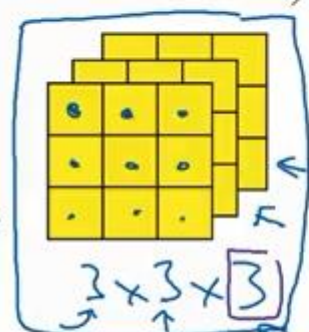
Convolutions on RGB image



$6 \times 6 \times 3$
 $\uparrow \quad \uparrow \quad \uparrow$

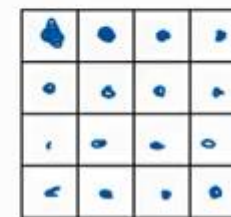


*



$3 \times 3 \times 3$
 27 numbers

=



4×4



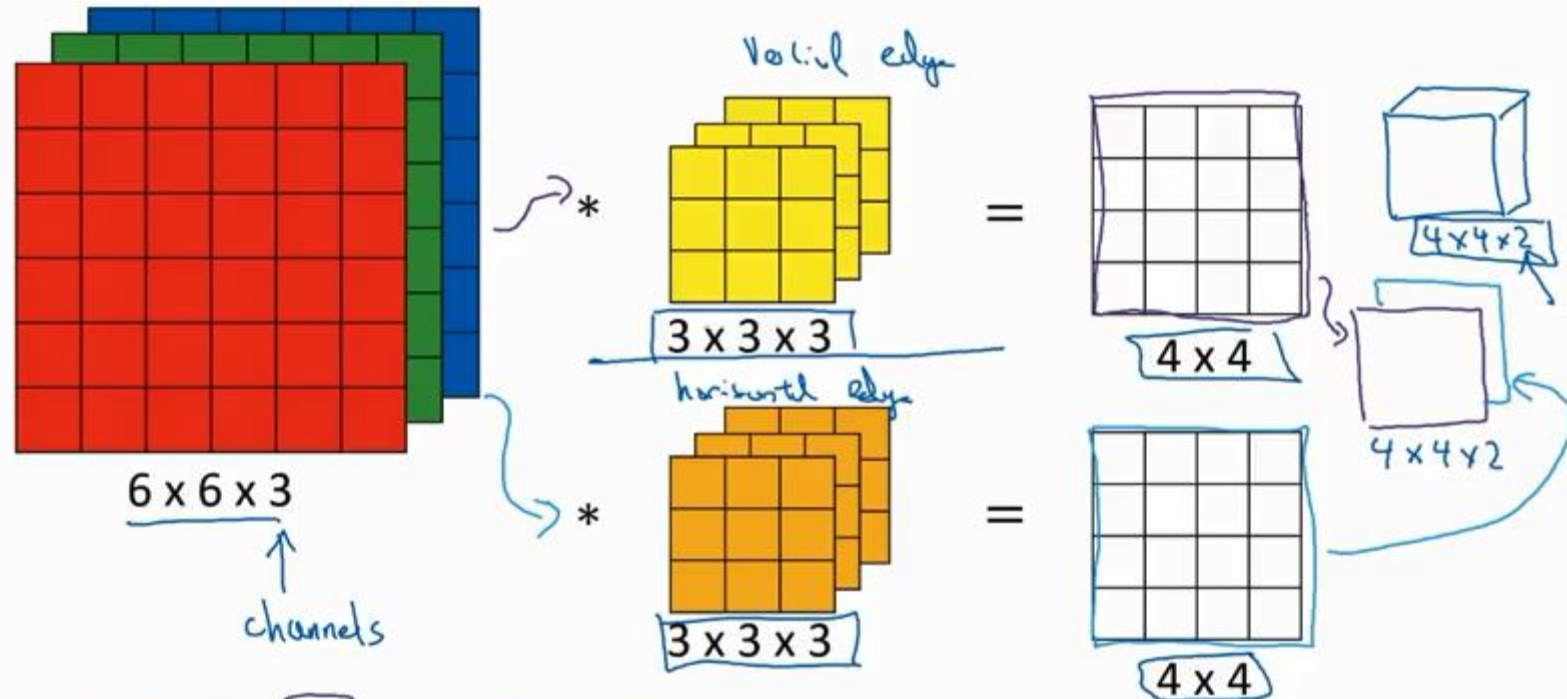
$\rightarrow 3 \times 3 \times 3$



$\rightarrow 3 \times 3 \times 3$

Andrew Ng

Multiple filters



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

$6 \times 6 \times 3$ $3 \times 3 \times 3$ $4 \times 4 \times 2$ \uparrow #filters

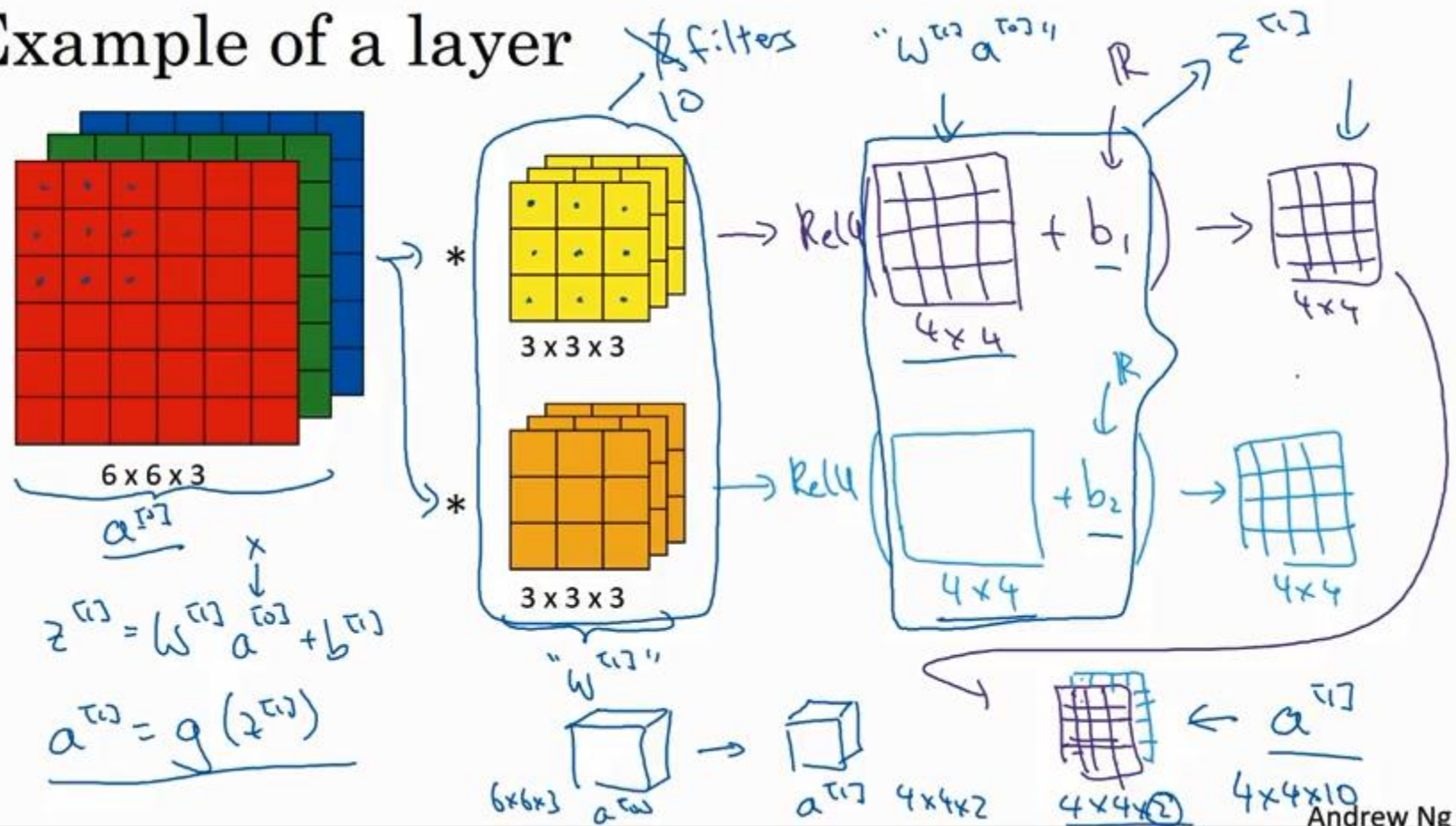


deeplearning.ai

Convolutional Neural Networks

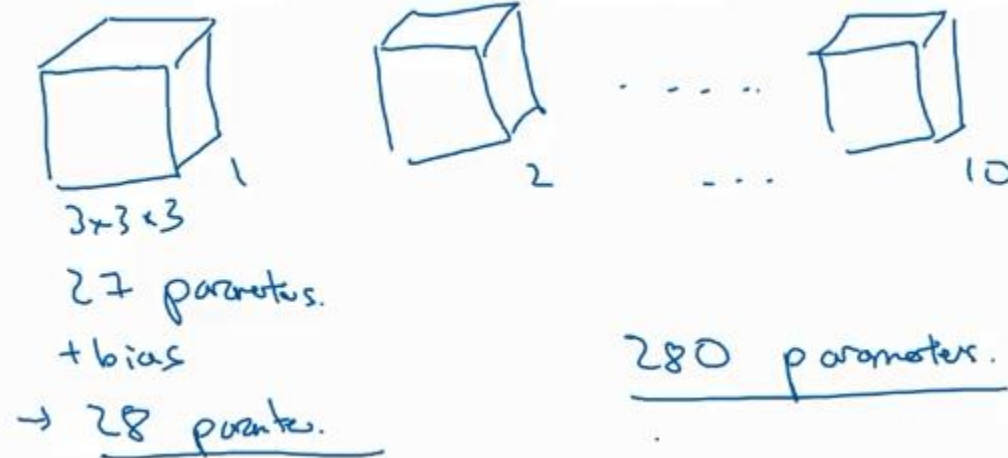
One layer of a
convolutional
network

Example of a layer



Number of parameters in one layer

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



Summary of notation

If layer l is a convolution layer:

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

$p^{[l]}$ = padding

$s^{[l]}$ = stride

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

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

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: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$
Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

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

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

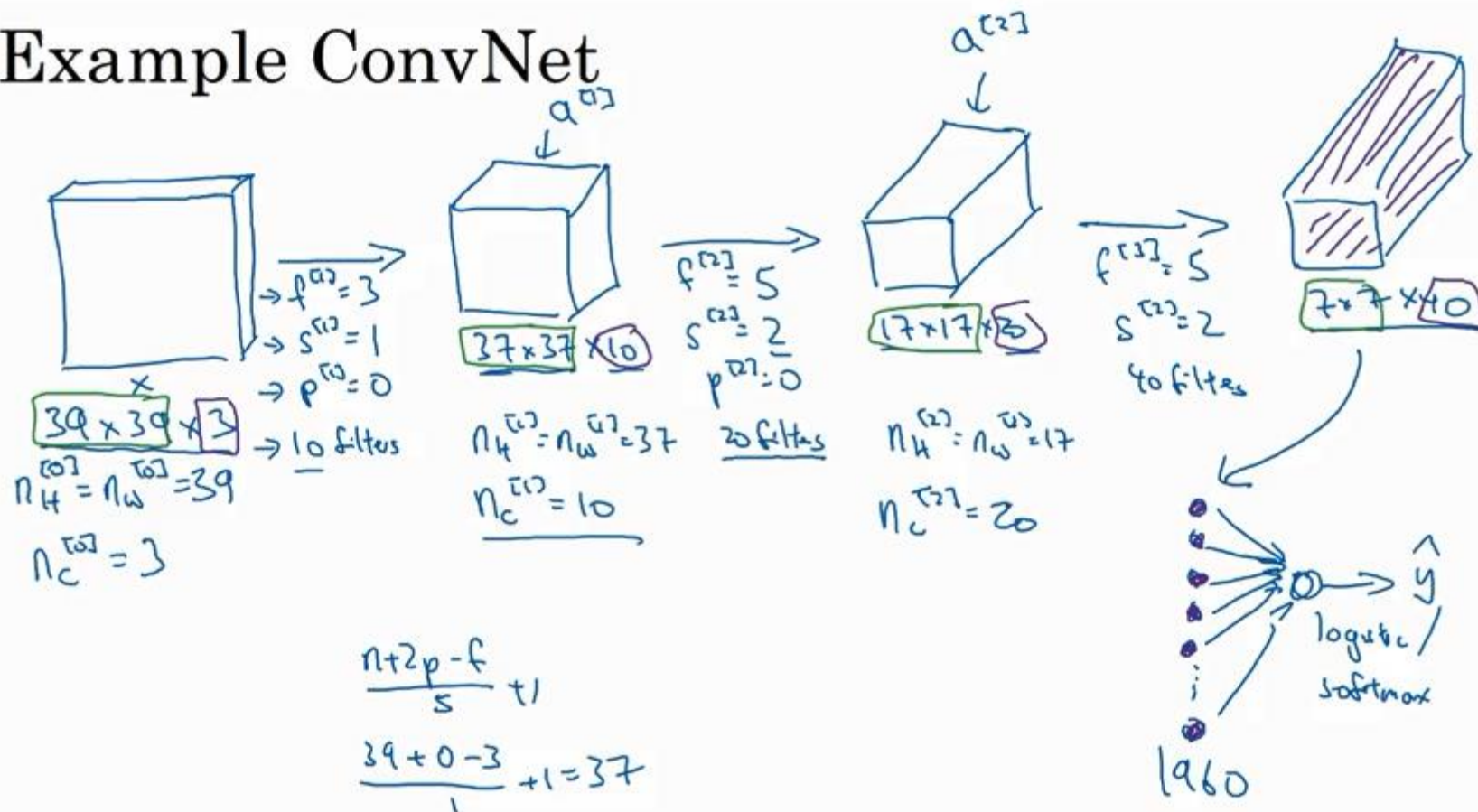


deeplearning.ai


Convolutional Neural Networks

A simple convolution network example

Example ConvNet



Types of layer in a convolutional network:

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

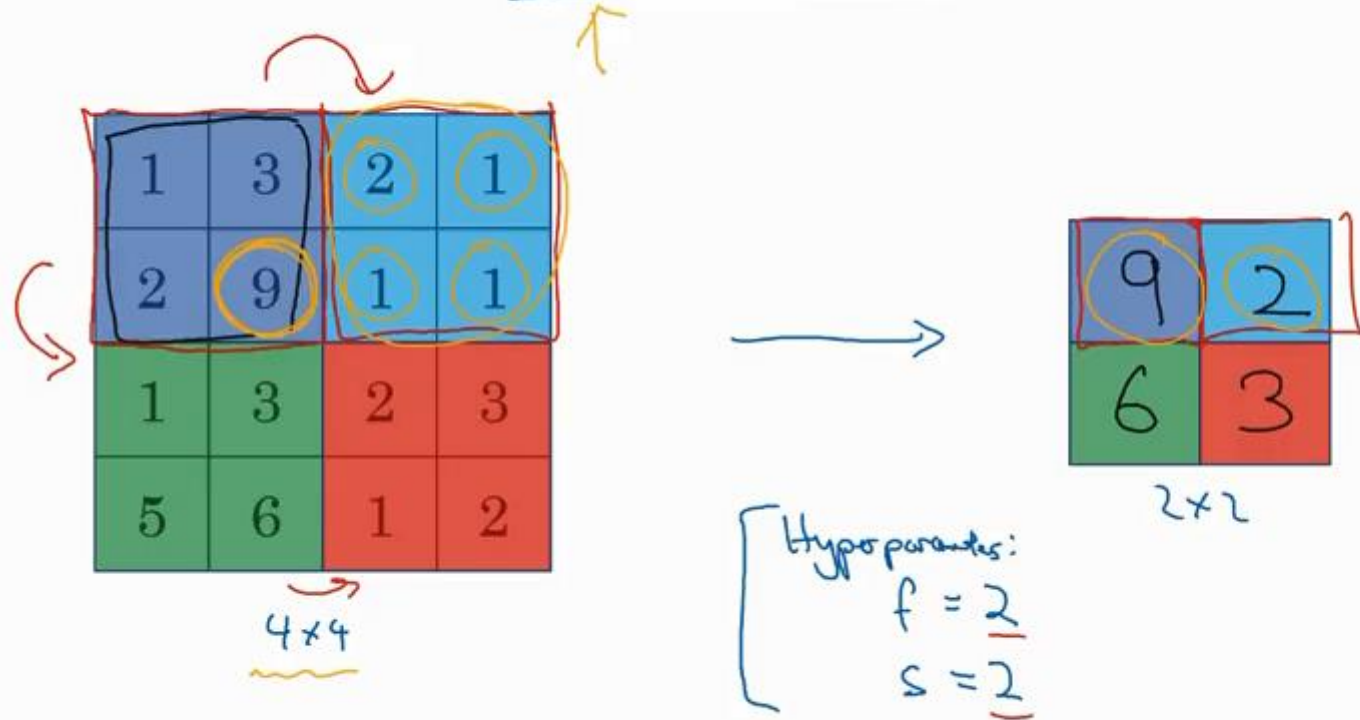


deeplearning.ai

Convolutional Neural Networks

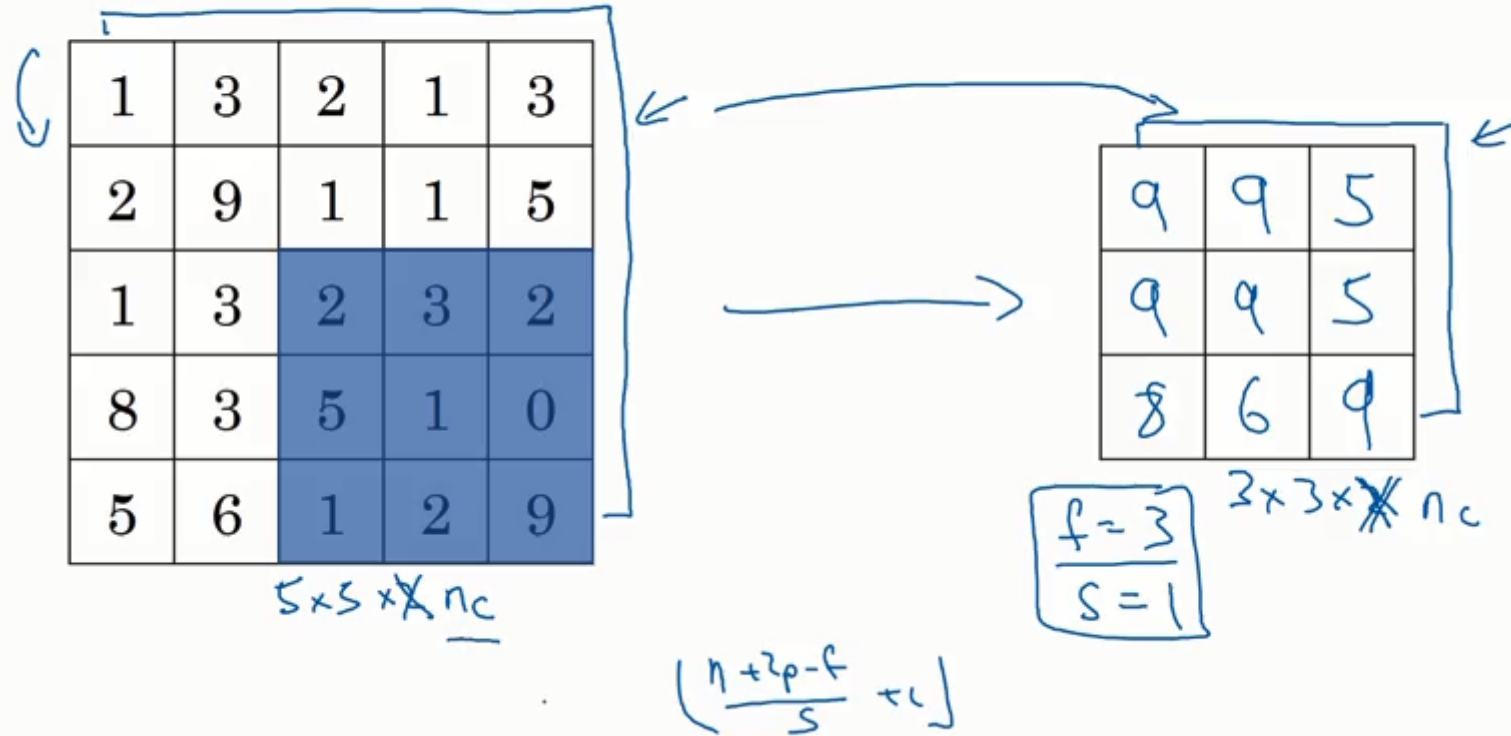
Pooling layers

Pooling layer: Max pooling



Hypoparameters:
 $f = \underline{2}$
 $s = \underline{2}$
No parameters!

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

$$f=2$$

$$s=2$$

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

Summary of pooling

Hyperparameters:

f : filter size
s : stride

f=2, s=2
f=3, s=2

Max or average pooling

~~⇒ p: padding.~~

No parameters to learn!

$$\begin{array}{c} n_H \times n_W \times \underline{n_C} \\ \downarrow \\ \left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \\ \times \underline{n_C} \end{array}$$

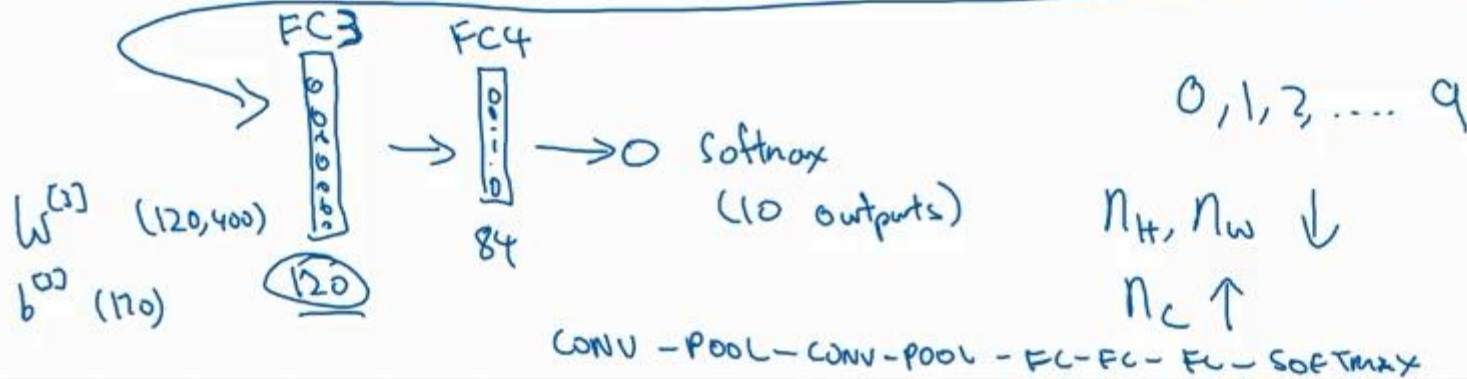
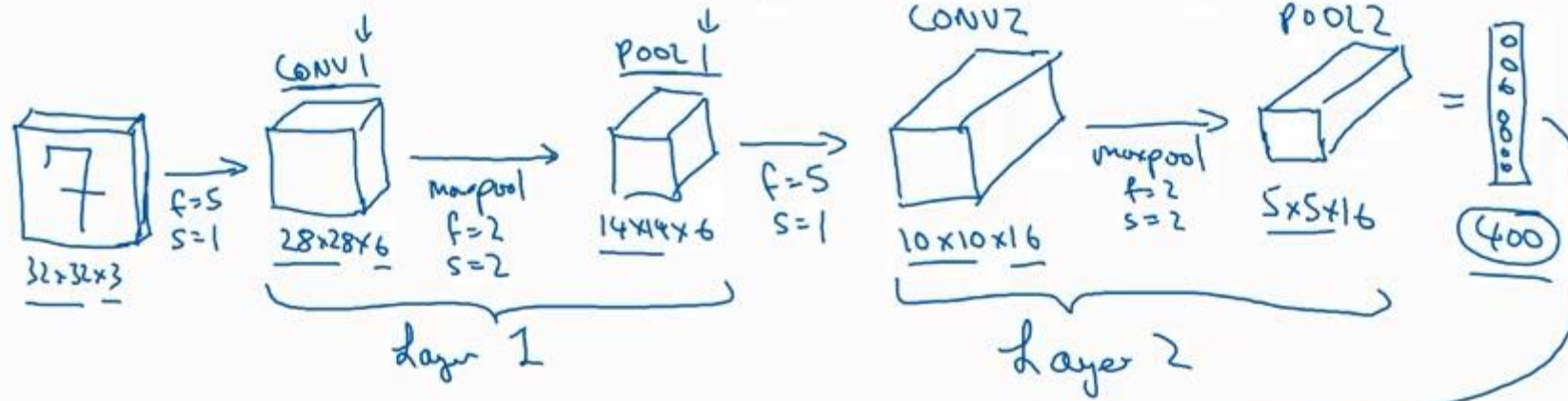


deeplearning.ai

Convolutional Neural Networks

Convolutional neural network example

Neural network example (LeNet-5)



CONV - POOL - CONV - POOL - FC - FC - FC - SOFTMAX

Andrew Ng

Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	— 3,072 $a^{[1]}$	0
CONV1 (f=5, s=1)	(28,28,8)	<u>6,272</u>	208 ←
POOL1	(14,14,8)	<u>1,568</u>	0 ←
CONV2 (f=5, s=1)	(10,10,16)	<u>1,600</u>	416 ←
POOL2	(5,5,16)	<u>400</u>	0 ←
FC3	(120,1)	<u>120</u>	48,001 }
FC4	(84,1)	<u>84</u>	10,081 }
Softmax	(10,1)	<u>10</u>	841

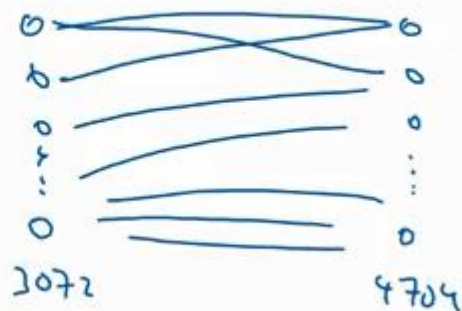
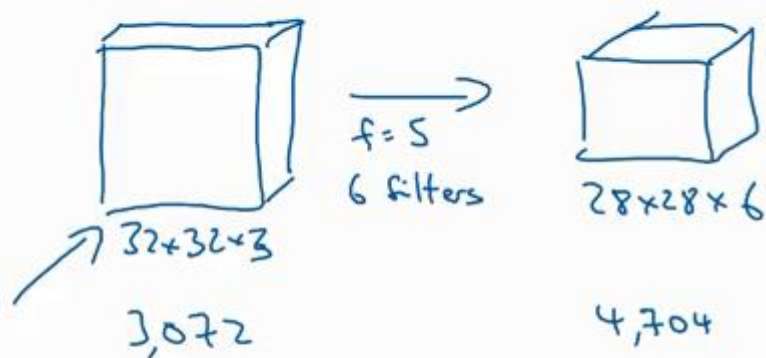


deeplearning.ai

Convolutional Neural Networks

Why convolutions?

Why convolutions



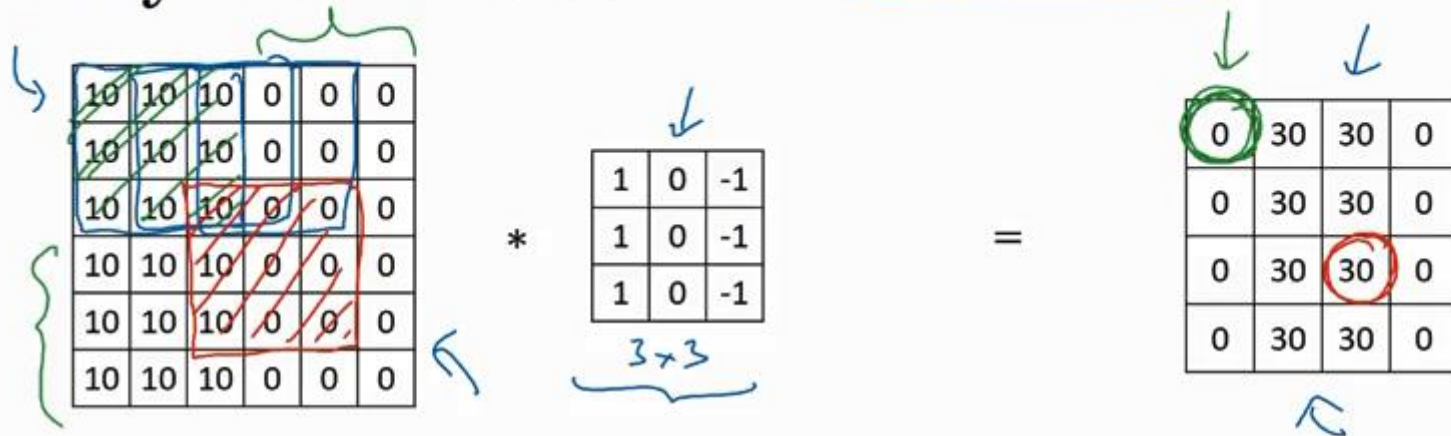
$$5 \times 5 = 25$$

$$26$$

$$6 \times 26 = 156 \text{ parameters}$$

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

Why convolutions

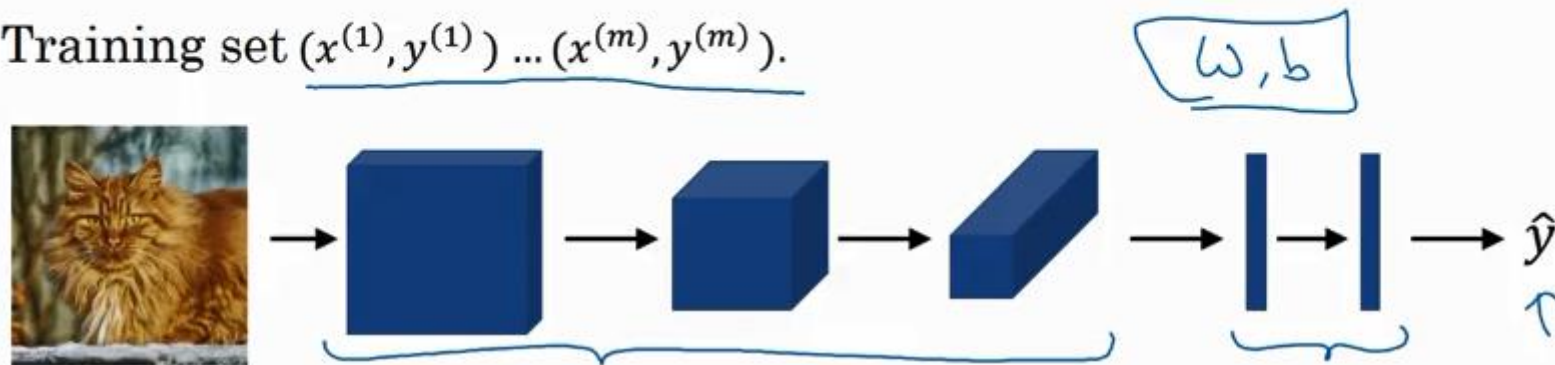


Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **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