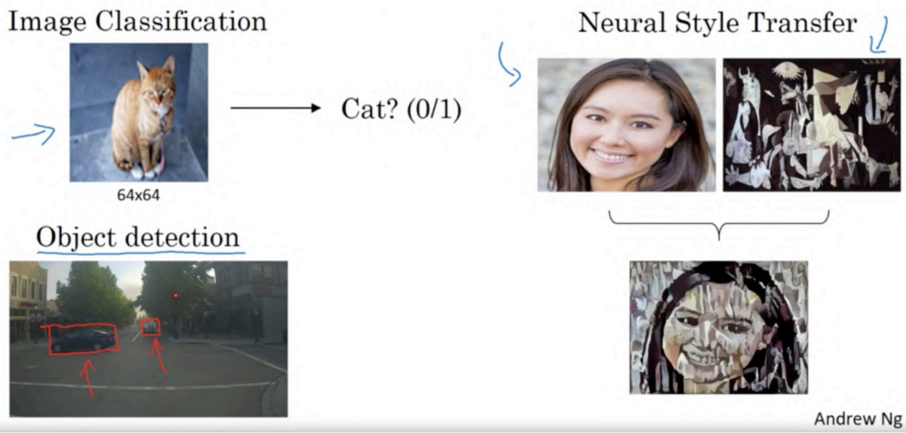
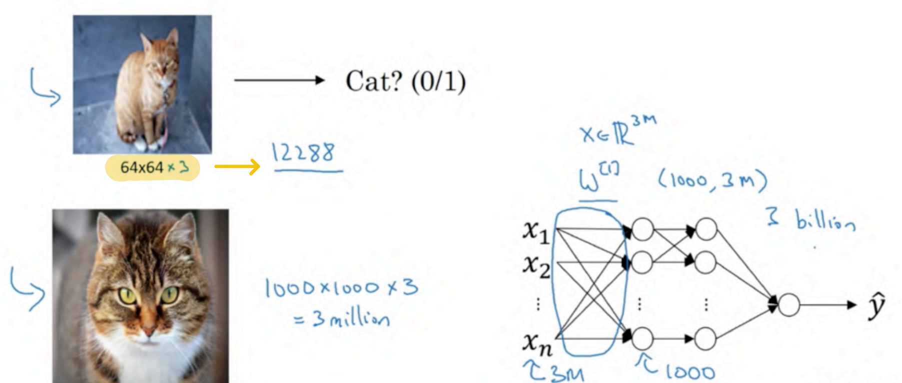
Computer Vision Problems



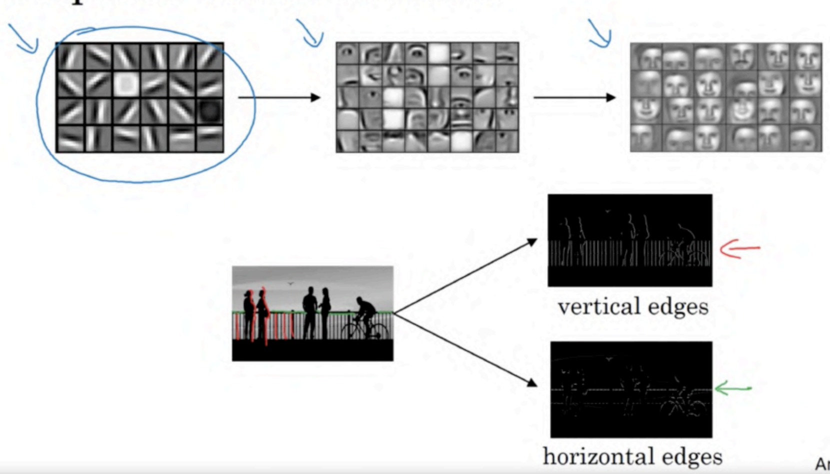
Andrew Ng

Deep Learning on large images



Andrew Ng

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

6x6

"convolution"

$*$

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

3x3 filter "Kernel"

=

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

4x4

python: conv_forward
tensorflow: tf.nn.conv2d
keras: Conv2D

Andrew Ng

Vertical edge detection

edge edge

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

6x6

$*$

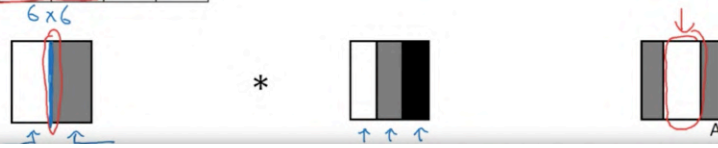
1	0	-1
1	0	-1
1	0	-1

3x3

=

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

4x4



Andrew Ng

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

$*$

Light .... Dark

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

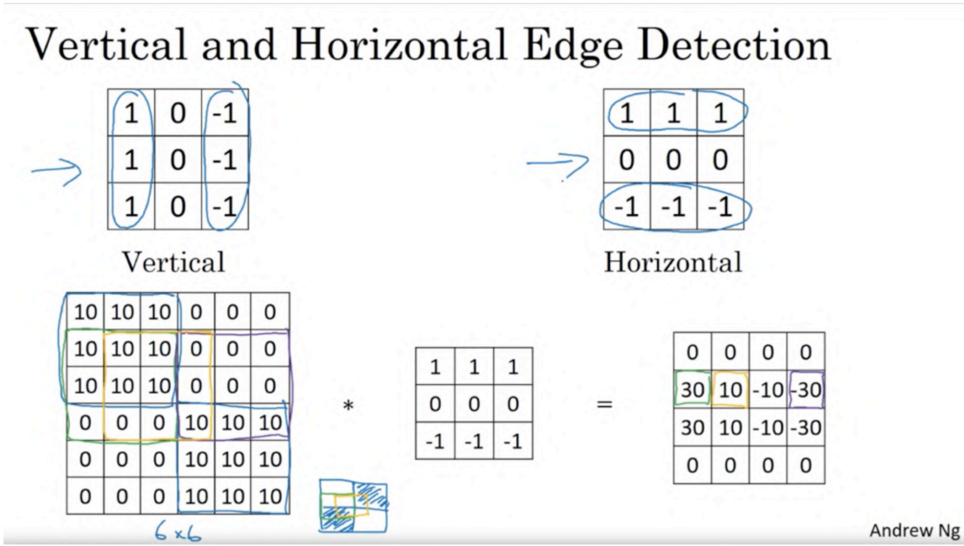
$*$

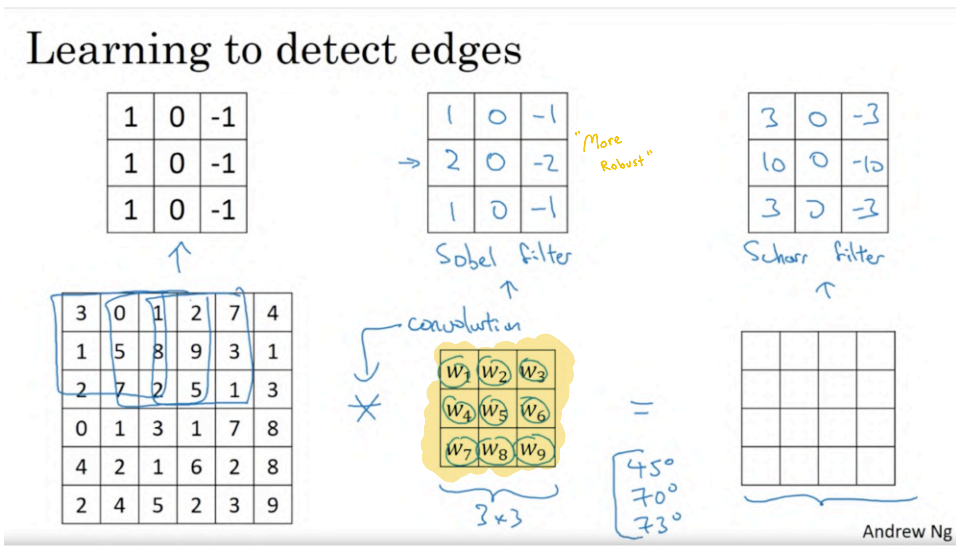
1	0	-1
1	0	-1
1	0	-1

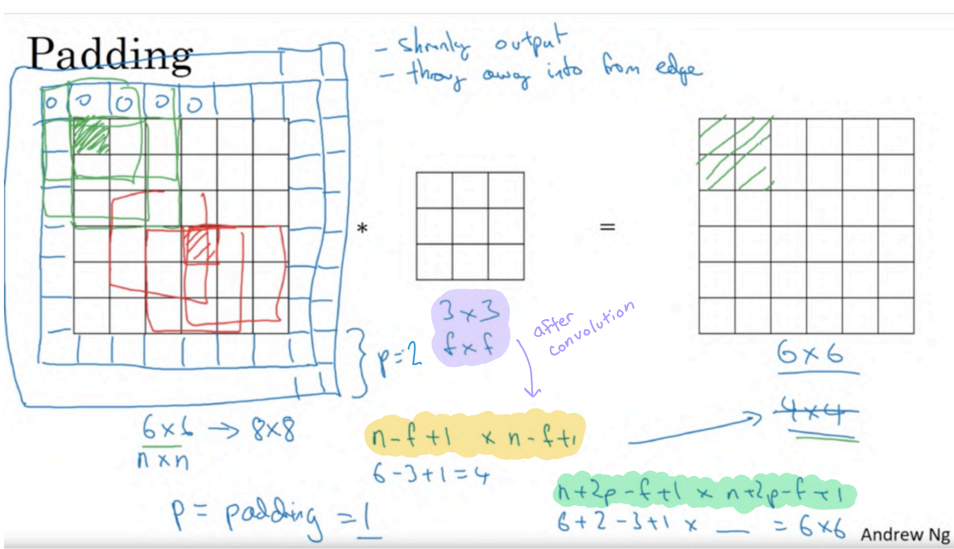
=

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

Andrew Ng







The two downsides to this; one is that, if every time you apply a convolutional operator, your image shrinks, so you come from six by six down to four by four then, you can only do this a few times before your image starts getting really small, maybe it shrinks down to one by one or something, so maybe, you don't want your image to shrink every time you detect edges or to set other features on it, so that's one downside, and the second downside is that, if you look the pixel at the corner or the edge, this little pixel is touched as used only in one of the outputs, because this touches that three by three region. Whereas, if you take a pixel in the middle, say this pixel, then there are a lot of three by three regions that overlap that pixel and so, is as if pixels on the corners or on the edges are use much less in the output. So you're throwing away a lot of the information near the edge of the image.

In order to fix both of these problems, what you can do is the full apply of convolutional operation. You can pad the image.

Valid and Same convolutions

no padding

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

$6 \times 6 \;*\; 3 \times 3 \longrightarrow 4 \times 4$

Two reasons for that; one is that if f was even, then you need some asymmetric padding. So only if f is odd that this type of same convolution gives a natural padding region, had the same dimension all around rather than pad more on the left and pad less on the right, or something that asymmetric. And then second, when you have an odd dimension filter, such as three by three or five by five, then it has a central position and sometimes in computer vision its nice to have a distinguisher, it's nice to have a pixel, you can call the central pixel so you can talk about the position of the filter.

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

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

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

$3\times3 \quad p = \frac{3-1}{2} = 1 \;\Big|\; 5\times5 \;(f=5) \quad p=2$

odd!

~~f is usually odd~~

1×1 ✓
3×3 ✓
5×5 ✓
7×7 ✓

Andrew Ng

Strided convolution

1 stride of 2

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 $\quad$ 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

Technical note on cross-correlation vs. convolution

Convolution in math textbook:

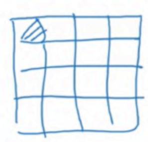
2^{7}	3^{2}	7^{5}	4	6	2
6^{9}	6^{0}	9^{4}	8	7	4
3^{-1}	4^{1}	8^{3}	3	8	9
7	8	3	6	6	3
4	2	1	8	3	4
3	2	4	1	9	8

*

3	4	5
1	0	2
-1	9	7

7	9	-1
2	0	1
5	4	3

=



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

Andrew Ng

At 4:44 when Andrew was explaining the technical note on cross-correlation vs convolution, the flipping of the filter was incorrect.

Originally, it was:

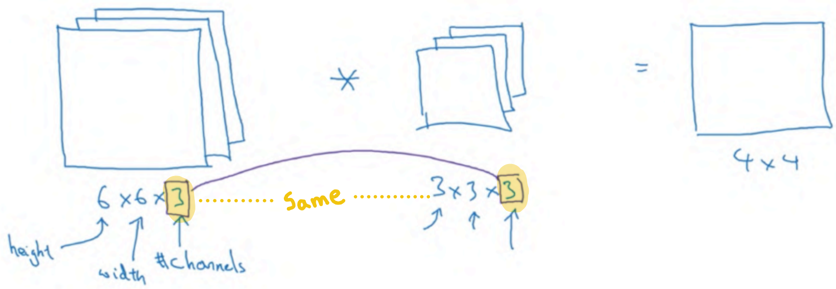
3	4	5
1	0	2
-1	9	7

The correct filter after flipping vertically and horizontally would be:

7	9	-1
2	0	1
5	4	3

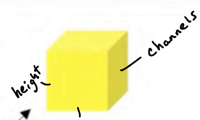
What Andrew did was more a flip over the diagonal.

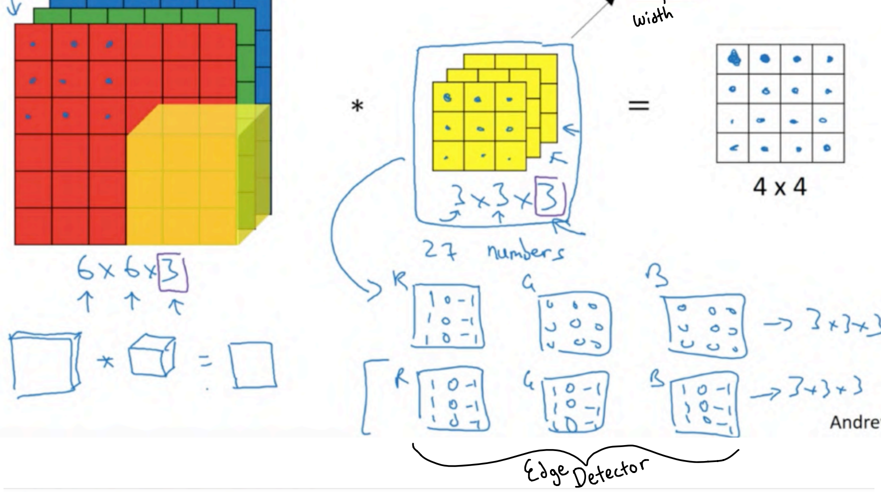
Convolutions on RGB images



Andrew Ng

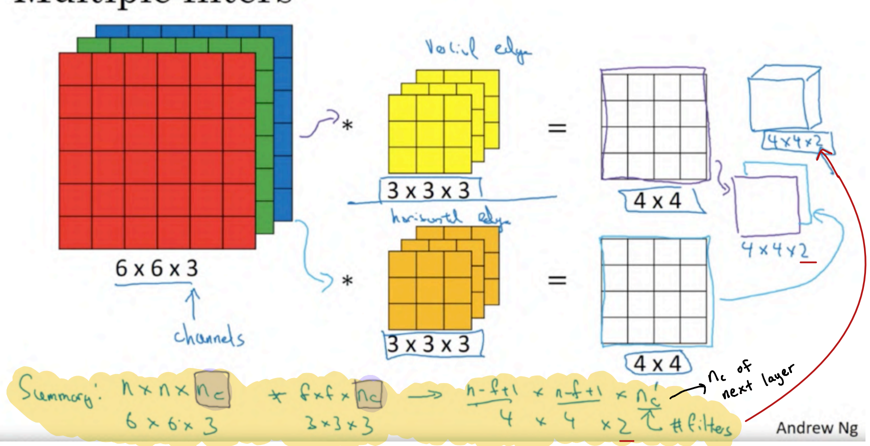
Convolutions on RGB image





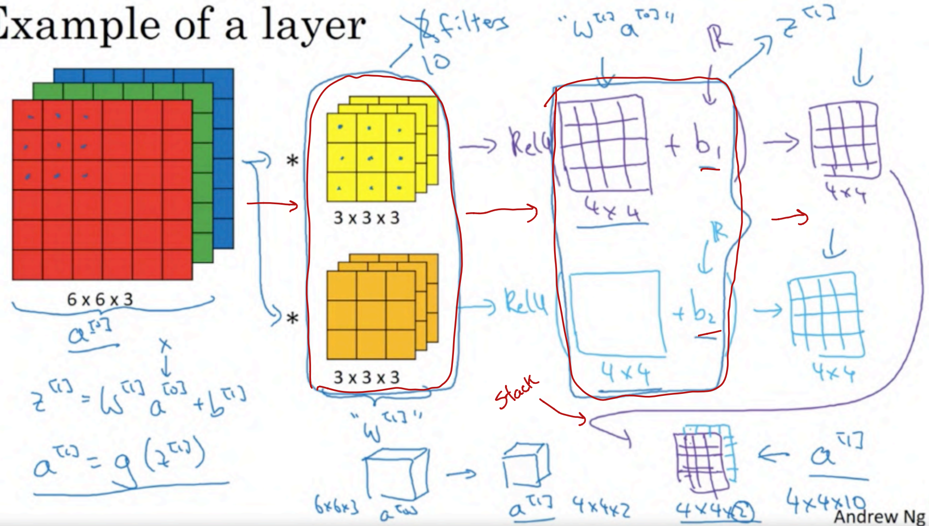
Andrew Ng

Multiple filters



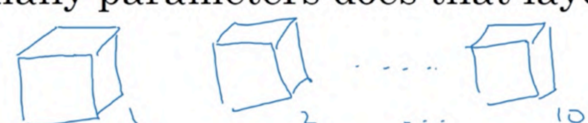
Andrew Ng

Example of a layer



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 params. } 1 filter

x 10 = 280 parameters.

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: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$
Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ ← #filters in layer l
bias: $n_c^{[l]}$ – $(1, 1, 1, n_c^{[l]})$

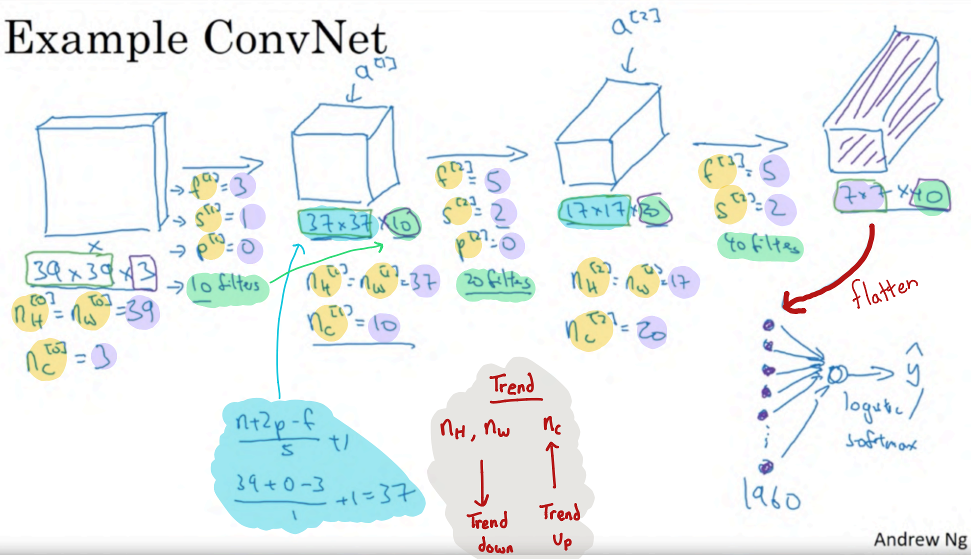
Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$
Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

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

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

$n_c \times n_H \times n_W$

Example ConvNet



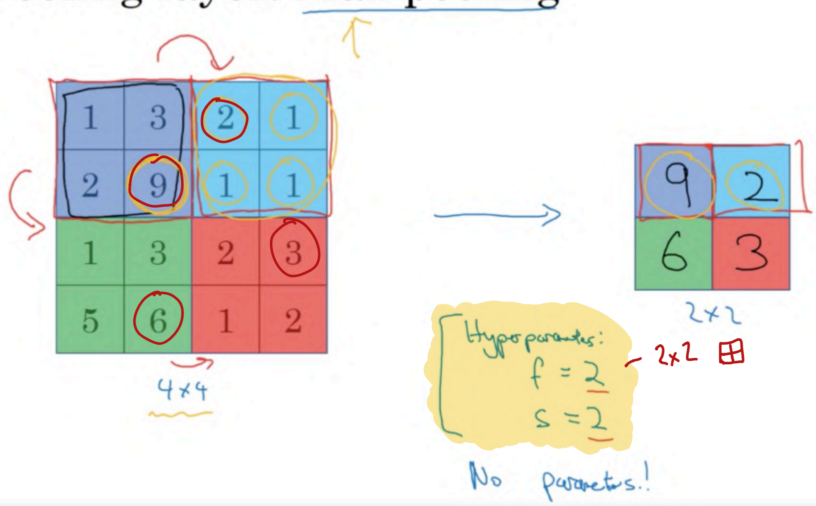
Andrew Ng

Types of layer in a convolutional network:

- Convolution (CONV)
- Pooling (POOL)
- Fully connected (FC)

Andrew Ng

Pooling layer: Max pooling



So here's the intuition behind what max pooling is doing. If you think of this four by four region as some set of features, the activations in some layer of the neural network, then a large number, it means that it's maybe detected a particular feature. So, the upper left-hand quadrant has this particular feature. It maybe a vertical edge or maybe a eye or whisker if you trying to detect a cat. Clearly, that feature exists in the upper left-hand quadrant. Whereas this feature, maybe it isn't cat eye detector. Whereas this feature, it doesn't really exist in the upper right-hand quadrant. So what the max operation does is a lots of features detected anywhere, and one of these quadrants , it then remains preserved in the output of max pooling. So, what the max operates to does is really to say, if these features detected anywhere in this filter, then keep a high number. But if this feature is not detected, so maybe this feature doesn't exist in the upper right-hand quadrant. Then the max of all those numbers is still itself quite small. So maybe that's the intuition behind max pooling.

Andrew Ng

Pooling layer: Max pooling

for 3D

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

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

→

If 3D

9	9	5
9	9	5
8	6	9

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

$\boxed{f=3,\ s=1}$

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

Andrew Ng

Pooling layer: Average pooling

Not used often

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

→

3.75	1.25
4	2

$f=2$
$s=2$

$7 \times 7 \times 1000 \rightarrow 1 \times 1 \times 1000$

You might use average pooling to collapse your representation from say, 7 by 7 by 1,000. An average over all the [inaudible] , you get 1 by 1 by 1,000.

Andrew Ng

Summary of pooling

Hyperparameters:

- f : filter size
- s : stride
- Max or average pooling

$f=2,\ s=2$
$f=3,\ s=2$

~~p: padding~~ rarely used; usually $p=0$

No parameters to learn!

$$n_H \times n_W \times n_c$$

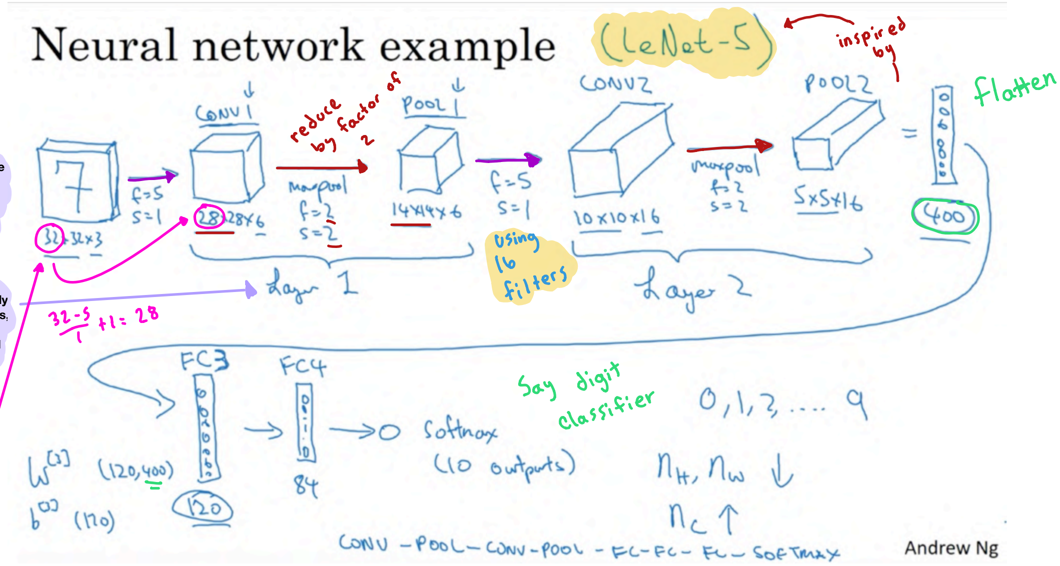
$$\downarrow$$

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

Andrew Ng

When people report the number of layers in a neural network usually people just record the number of layers that have weight, that have parameters. And because the pooling layer has no weights, has no parameters, only a few hyper parameters, I'm going to use a convention that Conv 1 and Pool 1 shared together

Since usually $p=0$, then $\frac{n+2p-f}{s}+1 = \frac{n-f}{s}+1$



Neural network example

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

Andrew Ng

Neural network example

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

Andrew Ng

Here are the 5 typos:

1. 208 should be (5*5*3 + 1) * 8 = 608

2. 416 should be (5*5*8 + 1) * 16 = 3216

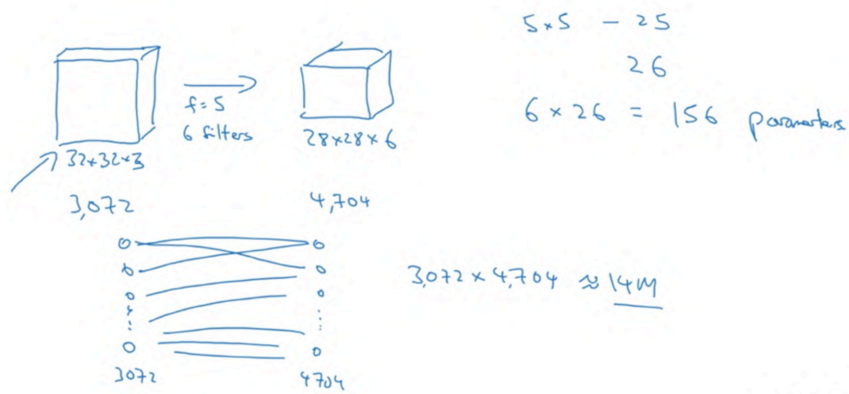
3. In the FC3, 48001 should be 400*120 + 120 = 48120, since the bias should have 120 parameters, not 1

4. Similarly, in the FC4, 10081 should be 120*84 + 84 (not 1) = 10164

(Here, the bias is for the fully connected layer. In fully connected layers, there will be one bias for each neuron, so the bias become In FC3 there were 120 neurons so 120 biases.)

5. Finally, in the softmax, 841 should be 84*10 + 10 = 850

Why convolutions



Andrew Ng

Why convolutions

translation invariance

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

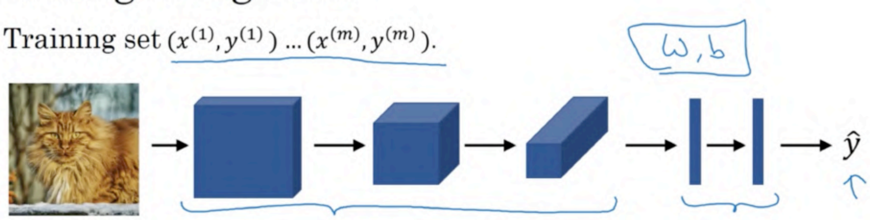
Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

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

Andrew Ng

Putting it together

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



$(5*5*3+1)*6 = 456$

This is based on the equation:

$$(f^{[l]} \times f^{[l]} \times n_c^{[l-1]} + 1) \times n_c^{[l]}.$$

$f^{[l]}$ is the filter height (and width).

$n_c^{[l-1]}$ is the number of channels in the previous layer.

$n_c^{[l]}$ is the number of channels in the current layer.

The "1" is the bias term.

(It was $(5*5+1)*6=156$ in the video.)

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