

Week 1 - Convolutional Neural Networks

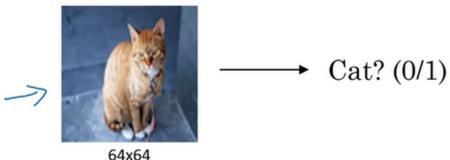
Wednesday, September 9, 2020 11:44 AM

① Computer Vision

→ computer vision problem

- Image classification
- Object detection
- Neural style transfer

Image Classification



Neural Style Transfer



Object detection



Andrew Ng

→ Regular neural networks become too big for images which are high dimensional data

→ Hence we need CNNs

② Edge detection example

→ we try to detect horizontal & vertical edges separately

→ we perform a convolution operation to do this

→ vertical edge detection

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

6x6

Input / Image

"convolution"
(not multiplication)

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

3x3
filter

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

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

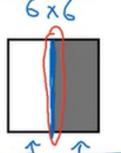
4x4

↑
Output

Andrew Ng

Vertical edge detection

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

③ More edge detection

→ Differentiating dark-to-light and light-to-dark edges

- The values are negative in light to dark edges

- We can use the absolute value if we do not care about light or dark

Vertical edge detection examples

$$\begin{array}{ccccccccc}
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0
 \end{array} * \begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array} = \boxed{\begin{array}{ccccccccc}
 0 & 30 & 30 & 0 & 0 & 0 & 0 \\
 0 & 30 & 30 & 0 & 0 & 0 & 0 \\
 0 & 30 & 30 & 0 & 0 & 0 & 0 \\
 0 & 30 & 30 & 0 & 0 & 0 & 0
 \end{array}}$$

$$\begin{array}{ccccccccc}
 0 & 0 & 0 & 10 & 10 & 10 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0
 \end{array} * \begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array} = \boxed{\begin{array}{ccccccccc}
 0 & -30 & -30 & 0 & 0 & 0 & 0 \\
 0 & -30 & -30 & 0 & 0 & 0 & 0 \\
 0 & -30 & -30 & 0 & 0 & 0 & 0 \\
 0 & -30 & -30 & 0 & 0 & 0 & 0
 \end{array}}$$

Andrew Ng

→ Vertical and Horizontal Edge detection

Vertical and Horizontal Edge Detection

$$\begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array}$$

Vertical

$$\begin{array}{ccc}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 -1 & -1 & -1
 \end{array}$$

Horizontal

$$\begin{array}{ccccccccc}
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 0
 \end{array}$$

$$* \begin{array}{ccc}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 -1 & -1 & -1
 \end{array} = \begin{array}{ccccccccc}
 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 30 & 10 & -10 & -30 & 0 & 0 & 0 \\
 30 & 10 & -10 & -30 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0
 \end{array}$$

→ works well on actual images

not well defined due to small images

Andrew Ng

→ Learning to detect edges

only for vertical

$$\begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array}$$

$$\begin{array}{ccc}
 1 & 0 & -1 \\
 2 & 0 & -2 \\
 1 & 0 & -1
 \end{array}$$

sobel filter

$$\begin{array}{ccc}
 3 & 0 & -3 \\
 10 & 0 & -10 \\
 3 & 0 & -3
 \end{array}$$

Scharr filter

- We can make our filters as learnable parameters to make better filters and detect edges that are not horizontal or vertical too

$$\begin{array}{ccc}
 w_1 & w_2 & w_3 \\
 w_4 & w_5 & w_6 \\
 w_7 & w_8 & w_9
 \end{array}$$

④ Padding

→ When we use filters,

- $n \times n \rightarrow \text{image}$
 $f \times f \rightarrow \text{filter}$

the output is $(n-f+1) \times (n-f+1)$

This results in shrinking output

- The edges don't get weightage as much as center

→ We use padding to avoid both these problems

- padding is generally done with zeros

- $p \rightarrow$ amount of padding (pixels)

$(n+p) \times (n+p) \rightarrow \text{image}$

$(n+2p-f+1) \times (n+2p-f+1) \rightarrow \text{output}$

→ Valid and same convolutions

"Valid": no padding

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

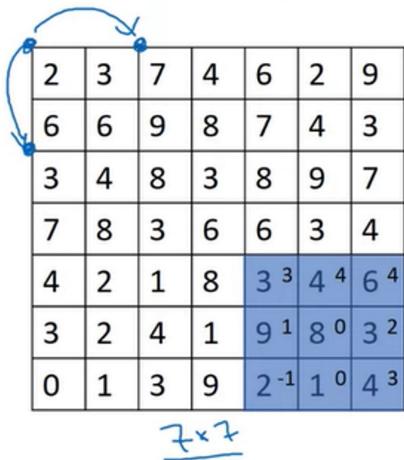
$$n+2p-f+1 = n$$

$$\Rightarrow 2p = f - 1$$

$$\Rightarrow p = \frac{f-1}{2} \quad (f \text{ is almost always odd})$$

⑤ Strided Convolutions

Strided convolution



$$\begin{matrix} * & \begin{matrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{matrix} & = & \begin{matrix} 91 & 100 & 83 \\ 69 & 91 & 127 \\ 44 & 72 & 74 \end{matrix} \\ & \underline{3 \times 3} & & \underline{3 \times 3} \\ & \text{Stride} = 2 & & \end{matrix}$$

$\rightarrow S \rightarrow \text{stride}$

$$\therefore \text{Output dimension} = \left\lceil \frac{n+2p-f}{s} \right\rceil + 1 \quad (\lfloor x \rfloor = \text{floor}(x))$$

We do not do computation if the filter goes outside the image and padding

\rightarrow In math, the filter is mirrored when applying convolutions.

The operation we perform is generally called cross-correlation.

But in DL, it is referred to as just convolution.

Technical note on cross-correlation vs. convolution

Convolution in math textbook:

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

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

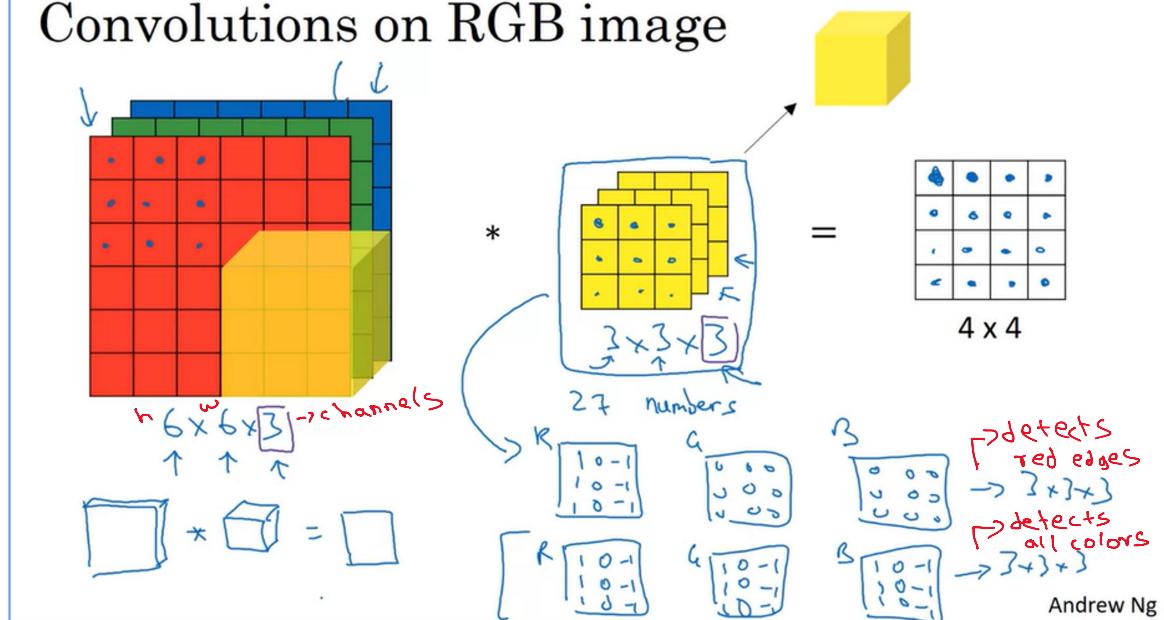
DL doesn't care about associativity so cross correlation is fine

$(A * B) * C = A * (B * C)$
used in Signal Processing

Andrew Ng

⑥ Convolutions over Volume

Convolutions on RGB image

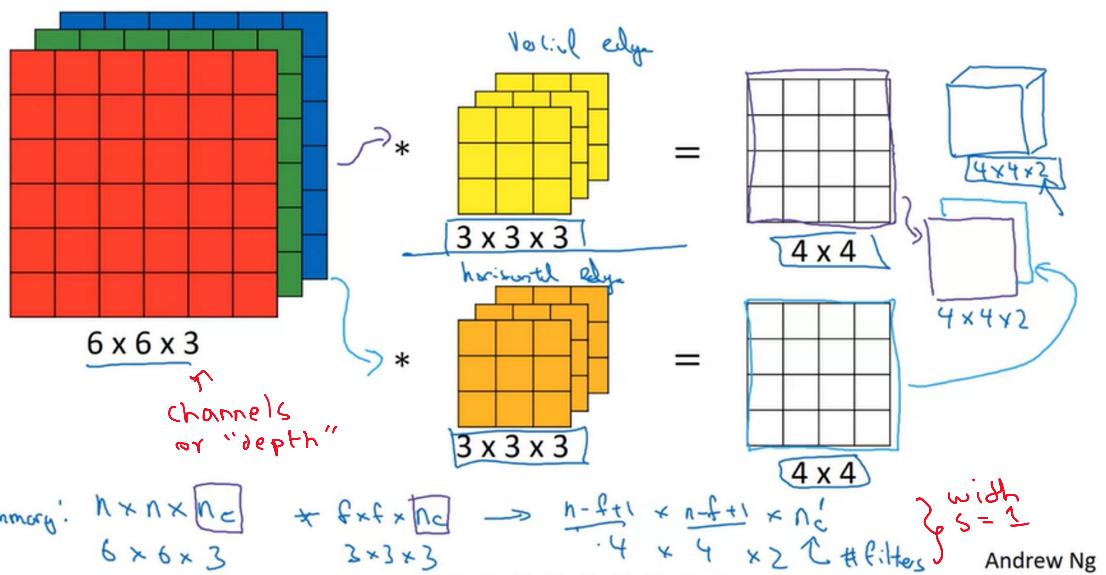


→ The number of channels must be same for image and filter

→ Multiple filters

- Multiple filters are used to detect horizontal and vertical edges.
- They are applied separately.

Multiple filters



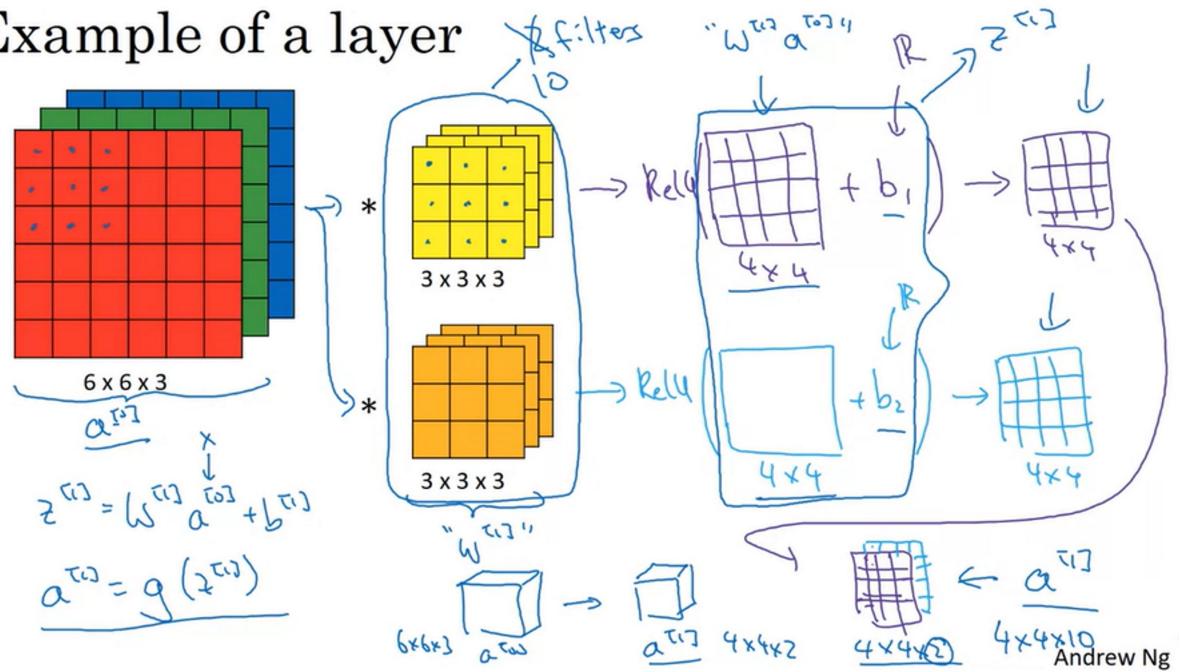
⑦ One layer of a Convolutional Network

- In a Convolutional Network, the image is the ~~input~~, the filters are ~~weights~~.
- We add a constant b to the output of the filter before applying the activation function (e.g. ReLU).
- ...

OUTPUT OF THE FILTER BEFORE APPLYING THE ACTIVATION FUNCTION (e.g. RELU)

→ we combine the outputs of all filters to give $a^{(l)}$

Example of a layer



Andrew Ng

→ Notation

- $f^{(l)}$ = Filter size in layer l
- $p^{(l)}$ = padding
- $s^{(l)}$ = stride
- $n_c^{(l)}$ = number of filters
- Each Filter is:
 $f^{(l)} \times f^{(l)} \times n_c^{(l)}$
- Activations
 $a^{(l)} \rightarrow n_H \times n_W \times n_C^{(l)}$
- $A \rightarrow m \times n_H \times n_W \times n_C^{(l)}$
↳ mini-batches
- Weights: $f^{(l)} \times f^{(l)} \times n_c^{(l-1)} \times n_c^{(l)}$
no. of filters in layer $l-1$
- bias: $n_c^{(l)} - (1, 1, 1, n_c^{(l)})$

$$\text{Input: } n_H^{(l-1)} \times n_W^{(l-1)} \times n_C^{(l-1)}$$

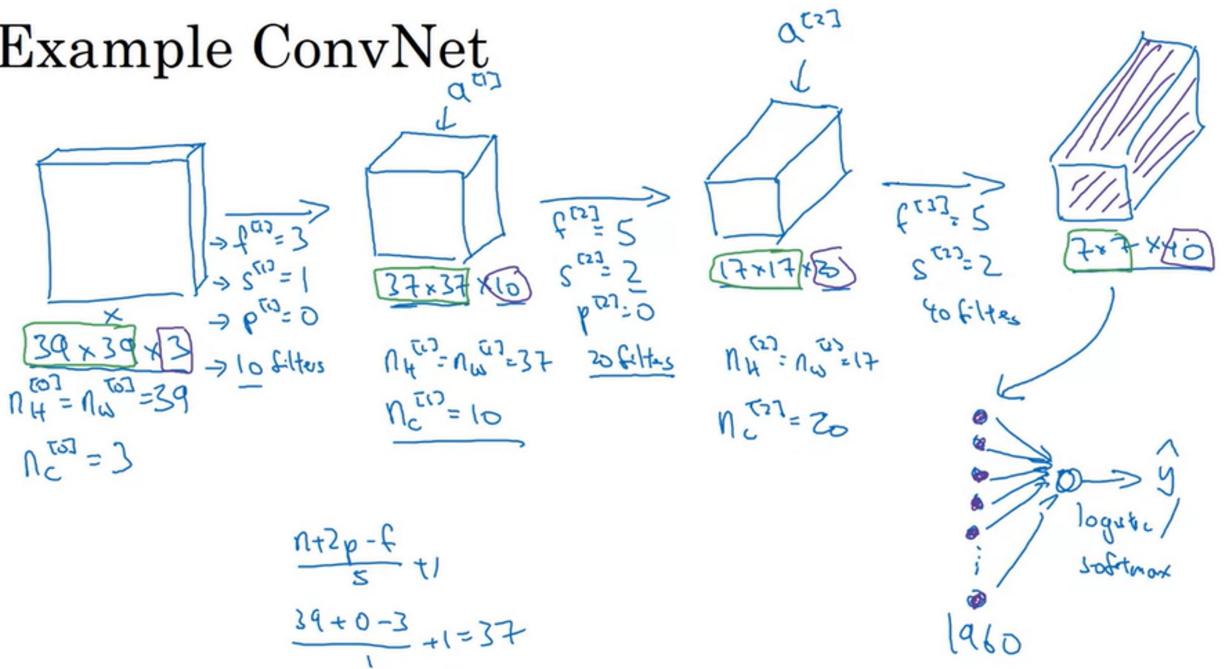
$$\text{Output: } n_H^{(l)} \times n_W^{(l)} \times n_C^{(l)}$$

$$n_H^{(l)} = \left\lfloor \frac{n_H^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1 \right\rfloor$$

$$n_W^{(l)} = \left\lfloor \frac{n_W^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1 \right\rfloor$$

⑧ A simple Convolutional Network example

Example ConvNet



Andrew Ng

→ As we go deeper, whereas the number of channels increases, image size reduces

- Types of layer in a convolutional Network
 - Convolution (CONV)
 - Pooling (POOL)
 - Fully connected (FC)

⑨ Pooling Layers

→ Types of pooling

- Max Pooling (Mostly used)
- Average Pooling

→ Pooling chooses the maximum value in the input (within every filter size) and outputs it

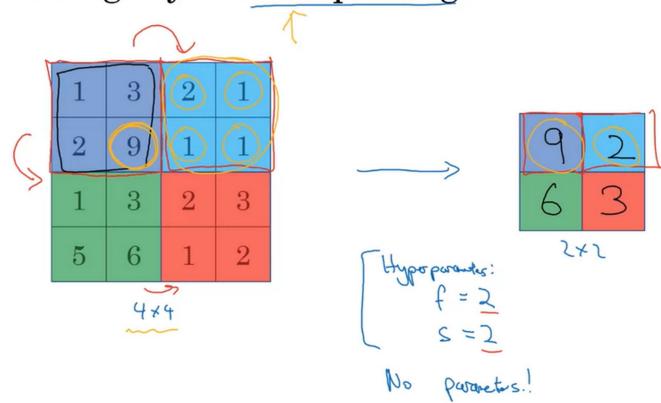
→ It has no parameters but only hyperparameters

f : filter size ($2, 3, \dots$)

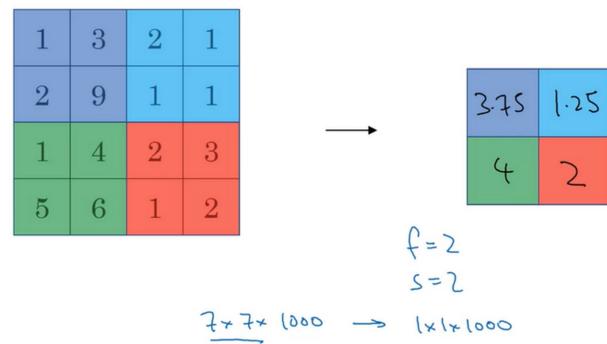
s : stride ($2, \dots$)

very rarely padding is applied

Pooling layer: Max pooling

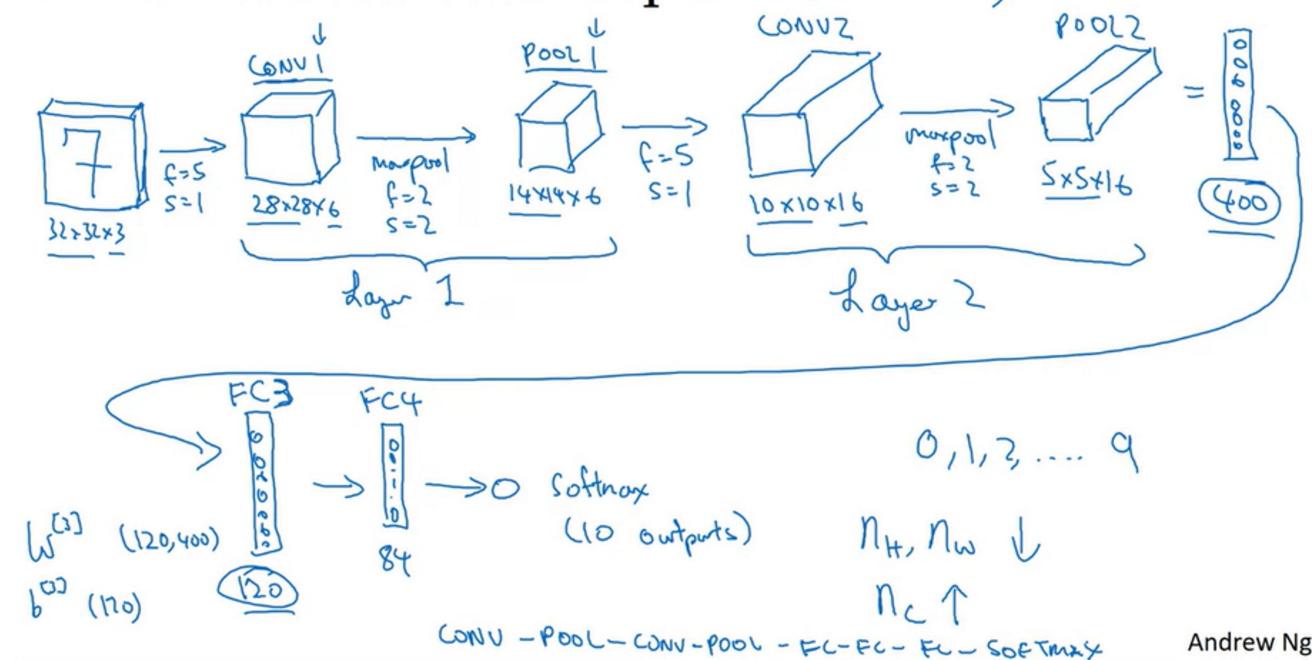


Pooling layer: Average pooling



⑩ CNN Example

Neural network example



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)	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

II 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
- Reduces no. of parameters greatly

→ Sparsity of connections

- In each layer, each output value depends only on a small number of inputs
- reduces overfitting

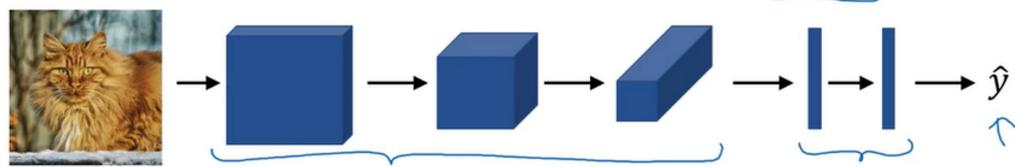
→ Translation invariance

- changing a few features would still result in a good detection

Putting it together

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

$\underline{\omega, b}$



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

Use gradient descent to optimize parameters to reduce J