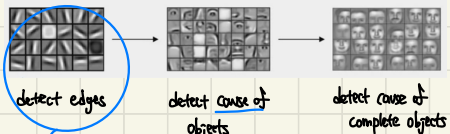
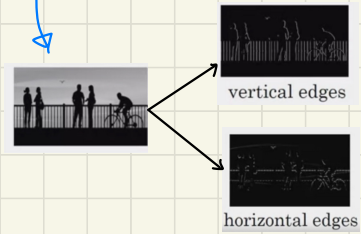
<Course 4> - Convolutional Neural Network

Edge Detection Example

- Computer Vision Problem



underlying factors that enable the identification & recognition of objects



- Horizontal Edge Detection

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

*

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

bright

horizontal filter

dark

=

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

- Vertical Edge Detection

$10\times1+10\times1+10\times0+10\times0+10\times0+10\times(-1)+10\times(-1)+10\times(-1)=0$

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	0	0	0

6x6

Convolution (합성곱)

*

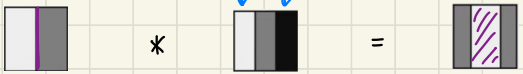
1	0	-1
1	0	-1
1	0	-1

3X3 filter (kernel)

=

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

4x4



if pixels↑ gonna be more accurate

- Vertical filters (make it horizontal by tilting 90°)

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

learn these filters & use these 9 #s as parameters ★

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

& tilting 45° 70° : apply degrees

Padding

- downside of convolutional operator

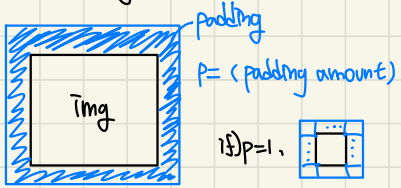
① apply convolutional op → img shrinks (shrinking output)

$(n \times n) * (f \times f) = \{(n-f+1) \times (n-f+1)\}$

② corner pixels apply f only once → throwing away corner info.

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

- Solution : Padding



p= (padding amount)

If p=1,

∴ output pixel : $(n+2p-f+1) \times (n+2p-f+1)$

- Valid & Same convolution

· "Valid", no padding

$(n \times n) * (f \times f) = \{(n-f+1) \times (n-f+1)\}$

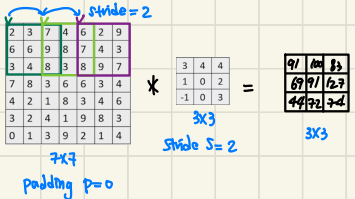
· "Same" ; pad so that output size is the same as input size

$(n+2p-f+1) \times (n+2p-f+1) \rightarrow n+2p-f+1 = n$ → is usually odd

$\therefore p = \frac{f-1}{2}$

1x1
3x3
5x5
7x7
⋮

Strided convolutions



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

padding p　　stride s　　round down $\lfloor z \rfloor = \text{floor}(z)$

- Cross-correlation vs convolution

For common convolution, we should flip f horizontally & vertically

But in ML, we don't.

→ It's called "convolution" in ML convention,
but its more similar to "cross-correlation"!

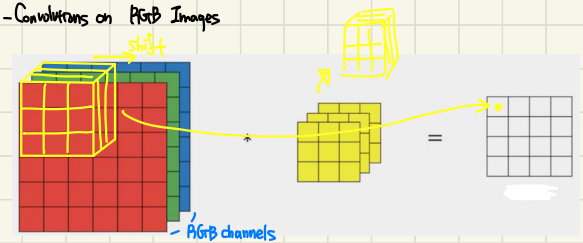
*Summary

$(n \times n \times n_c) \quad * \quad (f \times f \times n_c) \quad * \quad \rightarrow \quad \{(n-f+1) \times (n-f+1) \times n'_c\}$

↳ # channel (same)　　# filters

Convolutions over volumes

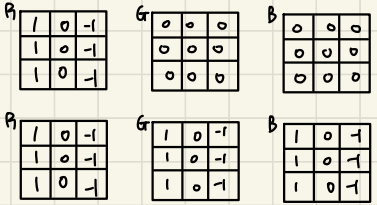
- Convolutions on RGB Images



6×6×3　　3×3×3　　4×4

height　width　# channel (depth of 3D volume)

filters



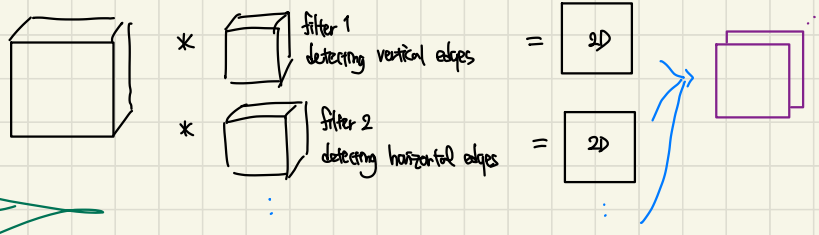
→ 3×3×3 filter detect vertical edges in Red channel

→ 3×3×3 filter detect vertical edges in any color

: tons of variation

- Multiple filters

applying multiple filters simultaneously

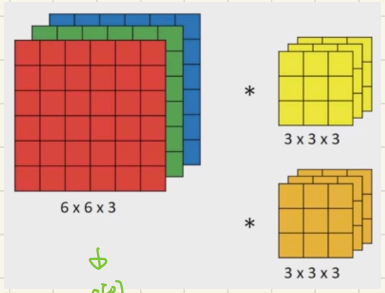


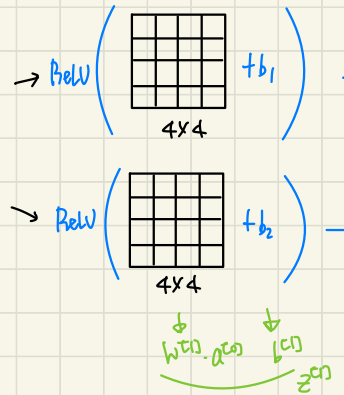
One layer of convolutional network

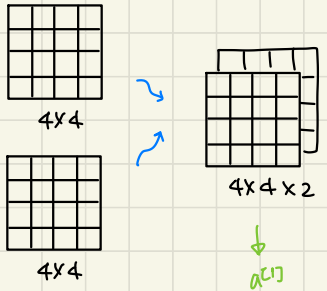
- Example of a layer

① apply bias ($b \in \mathbb{R}$) w/ Python broadcasting. apply bias to all params

② apply non-linearity ex) ReLU







transform 6x6x3 → 4x4x2

⇒ one layer of Convolutional neural network.

$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g(z^{[1]})$

② 3x3x3 matrix of 10 filters in one layer → how many parameters?

3x3x3 → 27 ws

1 bias (+)

28 parameters per filter

→ 10 filter x 28 params each = 280 params in total

⊕ Even though input size ↑, # param is not affected!

- Summary of Notation

if layer l is a convolutional layer:

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

$p^{[l]}$: padding

$s^{[l]}$: stride

$n_c^{[l]}$: # filter

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

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

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

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

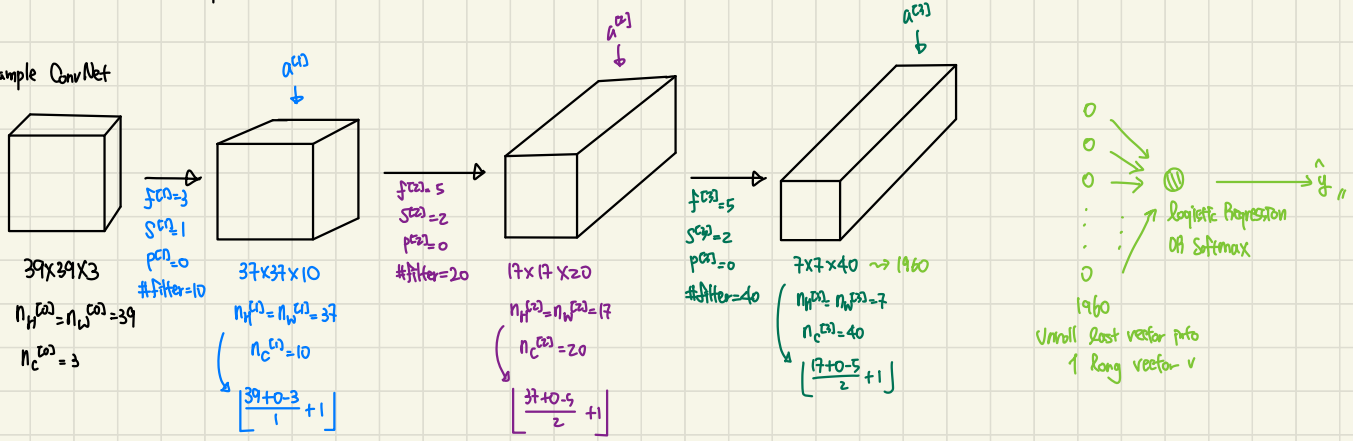
activation: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ → $A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ (m examples)

weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ ($f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$: each filter; $n_c^{[l]}$: # filter)

bias: $n_c^{[l]}$

Simple Convolutional Network Example

- Example ConvNet



- Types of layer in a convolutional network

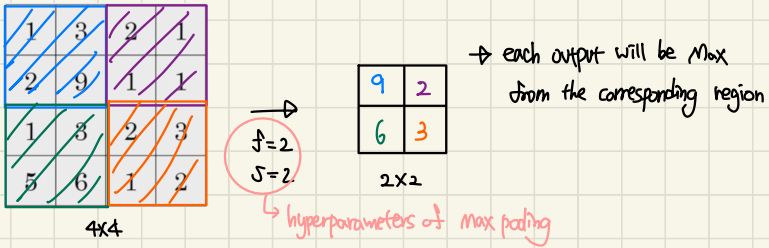
① Convolution (Conv)

② Pooling (POOL)

③ Fully Connected (FC)

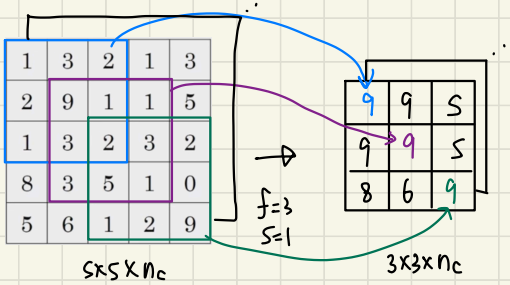
Pooling Layer

- Pooling Layer: Max pooling



→ each output will be Max from the corresponding region

↳ hyperparameters of max pooling



- Summary of pooling

Hyperparams:

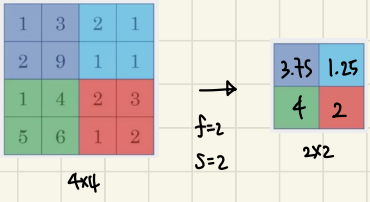
f: filter size

S: stride

usually no padding (p=0) ⇝ $n_H \times n_W \times n_c$ → $\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times n_c$ (input / output)

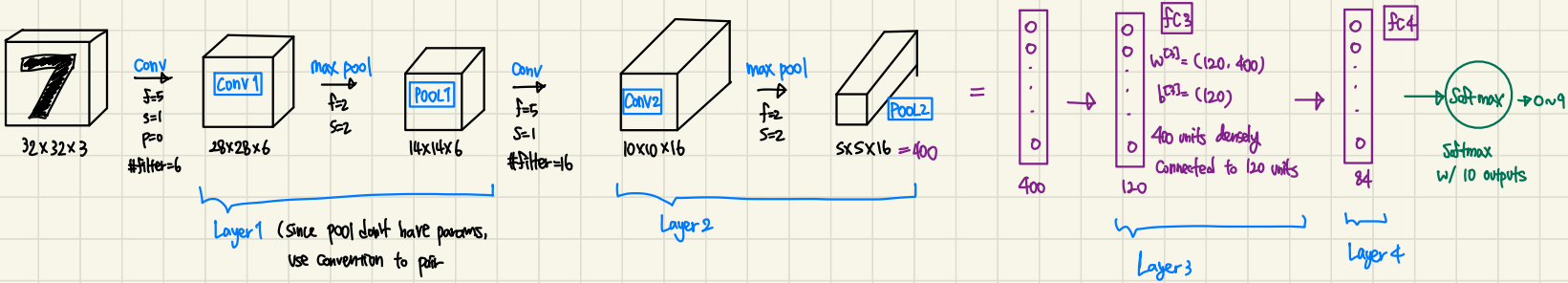
No parameters to learn!

- Pooling Layer: average pooling → not used much,,



CNN example

-Neural Network Example : handwritten digit recognition



	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5, s=1)	(28,28,6)	4,704	456
POOL1	(14,14,6)	1,176	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	2,416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,120
FC4	(84,1)	84	10,164
Softmax	(10,1)	10	850

① Max pooling don't have params

② Conv layer tend to have few params

③ lots of params for fc

④ activation size gradually ↓

⑤ n_H & n_W gradually ↑
n_C gradually ↓

⑥ Common pattern;
Conv → POOL → Conv → POOL → FC → FC → Softmax

*fc

In fully connected layer, each neuron is connected to every neuron in the prev layer

It takes high level features extracted by conv & pool, combines them to predict the final output.

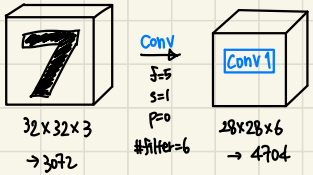
→ feature combination
→ classification

Why Convolutions?

① parameter sharing

② sparsity of connection

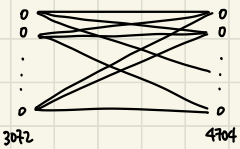
– Why convolutional over fully connected layers?



↳ Convolutional

each filter: $5 \times 5 \times 3 + 1 = f^{[l]} \times f^{[l]} \times n_c^{[l-1]} + 1$

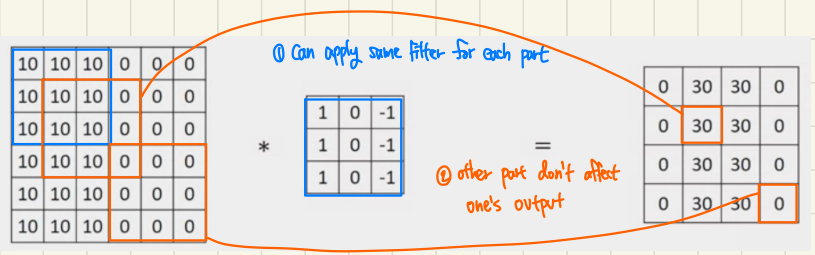
6 filters → $(5 \times 5 \times 3 + 1) \times 6$ ($n_c^{[l]}$) = 456



↳ fully connected network

$3072 \times 4704 \approx 13M$

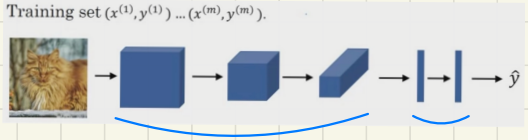
lots of params to train..



① 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.

② Sparsity of Connection: In each layer, each output value depends only on a small number of inputs.

– Putting it tgt



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

→ use gradient descent to optimize params to reduce J.