Course 4: CNN

WEEK 1

Computer Vision Problems:

- → Image classification
- → Object detection
- → Neural style transfer

One of the problem is that the inputs will be a lot. If we have a 1000 x 1000 px image then it'll be 1000 x 1000 x 3 (RGB) input values = 3M

So ~~then~~ if the 1st hidden layer has 1000 hidden values, then $w^{[1]}$ will be [1000, 3M] which is 3 billion values!

CNN: Edge detection example,

We will use convolution to get the vertical edge of an image.

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 x 6 image

convolution ↓ *

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

3 x 3 filter/kernel

=

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

4 x 4 output image

For -5,

3^{1}	0^{0}	1^{-1}
1^{1}	5^{0}	8^{-1}
2^{1}	7^{0}	2^{-1}

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

For -4,

0^{1}	1^{0}	2^{-1}
5^{1}	8^{0}	9^{-1}
7^{1}	2^{0}	5^{-1}

$$0\times1 + 5\times1 + 7\times1 + 1\times0 + 8\times0 + 2\times0 + 2\times-1 + 9\times-1 + 5\times-1 = -4$$

For -10,

1^{1}	5^{0}	8^{-1}
2^{1}	7^{0}	2^{-1}
3^{1}	1^{0}	3^{-1}

$$1\times1 + 1\times2 + 0\times1 + 0\times5 + 7\times0 + 1\times0 + 8\times-1 + 2\times-1 + 3\times-1 = -10$$

This is how its done.

Below is another example where you can see vertical edge detection

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
vertical

$=$

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

4x4

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

6x6

$*$

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

3x3
Vertical

$=$

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

4x4

Similarly

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

is horizontal filter

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 30 & 10 & -10 & -30 \\ 30 & 10 & -10 & -30 \\ 10 & 0 & 0 & 0 \end{bmatrix}$$

∴ Different filters help us to find horizontal and vertical edges

There are other filters,

1	0	-1
2	0	2
1	0	-1

Sobel filter

3	0	3
10	0	-10
3	0	-3

Schorr filter

We can even learn it as parameters using back propogation ....

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

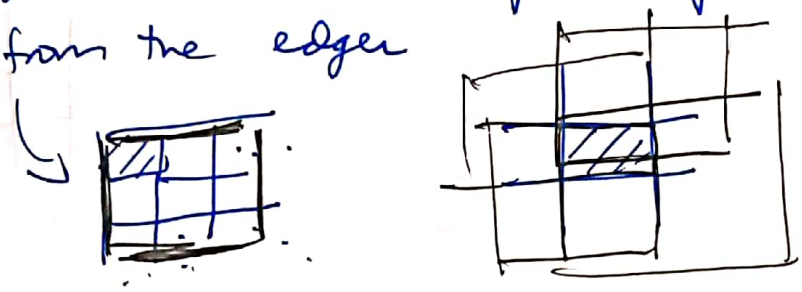
And it'll learn the best edge filter for that image data

Padding

There are ~~pro~~ 2 problems with the previous method:

1) The output shrinks from 6x6 to 4x4

2) We are throwing away information from the edges



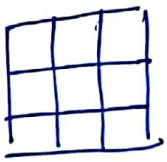
If its at edge we only consider it in 1 box, but if its at centre, we consider it in more boxes

The solution is padding (padding 0s on all sides so it becomes 8x8)

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

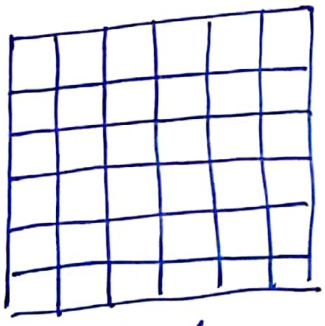
8x8

$*$



3x3

$=$



6x6

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

input size, filter size

Here padding, $p = 1$

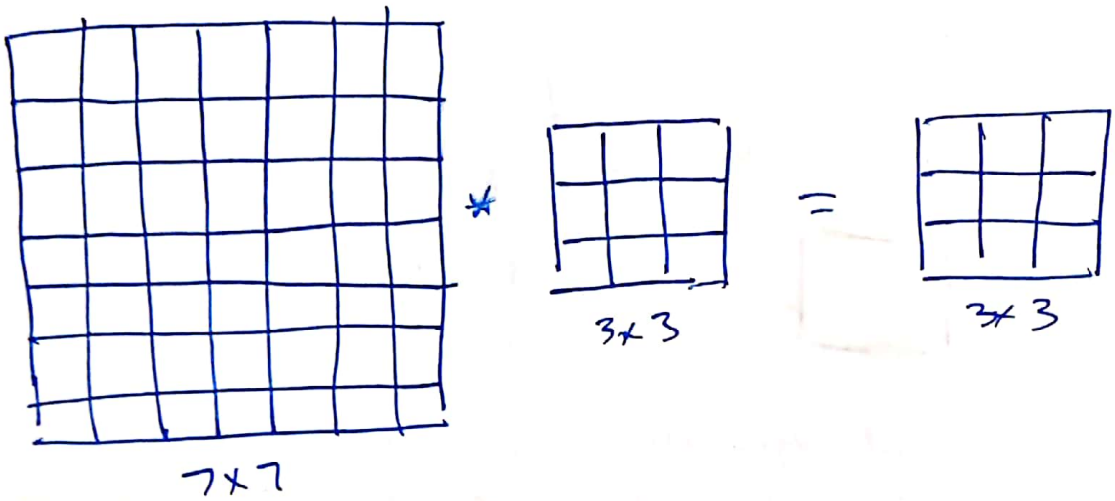
How to choose p:

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

- Valid convolution - no padding
- Same convolution - $p = \frac{f-1}{2}$ where f is filter size (usually f is odd).

Strided Convolutions

Instead of ~~shift~~ shifting (striding) the square by 1, you can select the stride s.



Summary:

$n \times n$ image, $f \times f$ filter, padding p, stride s

output size:

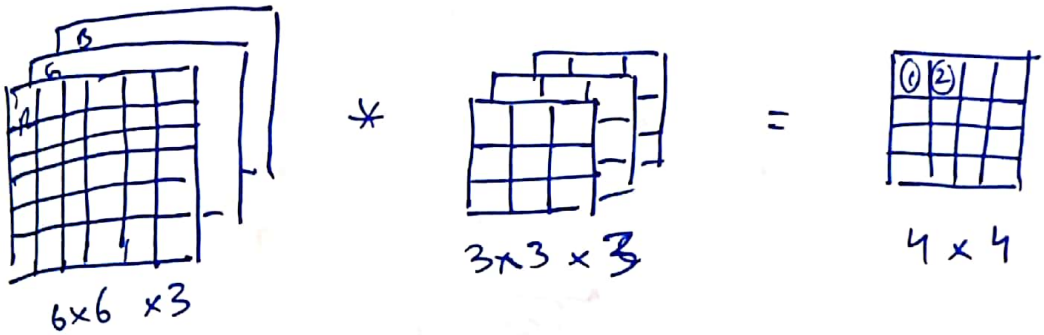
$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

- floor value

In actual convolution, we flip the filter before multiplying, so we are actually doing cross-convolution (no flipping) but we just call it convolution.

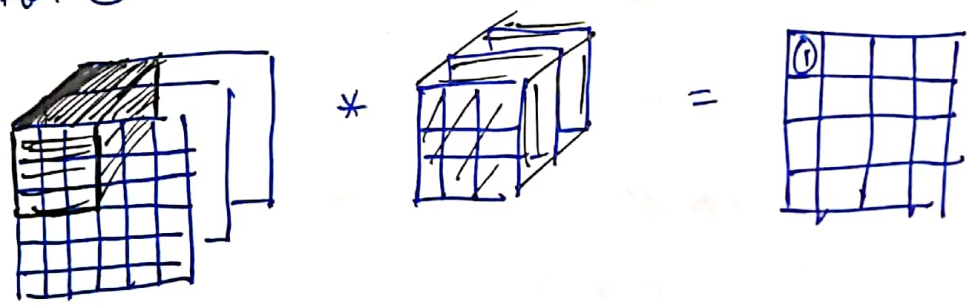
Convolutions over Volume

Convolutions over RGB image

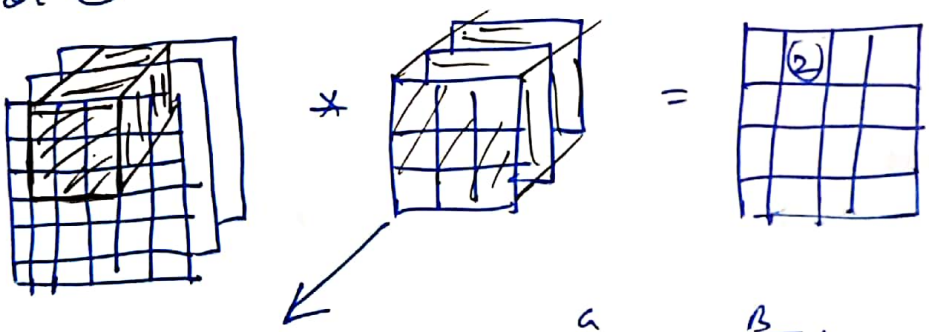


How this works is

For ①



For ②



If you set R

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

G

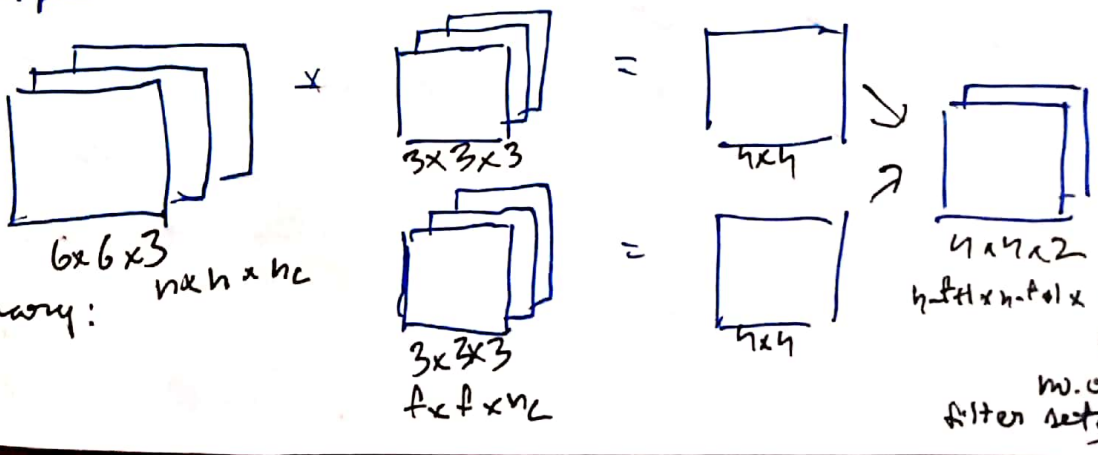
0	0	0
0	0	0
0	0	0

B

0	0	0
0	0	0
0	0	0

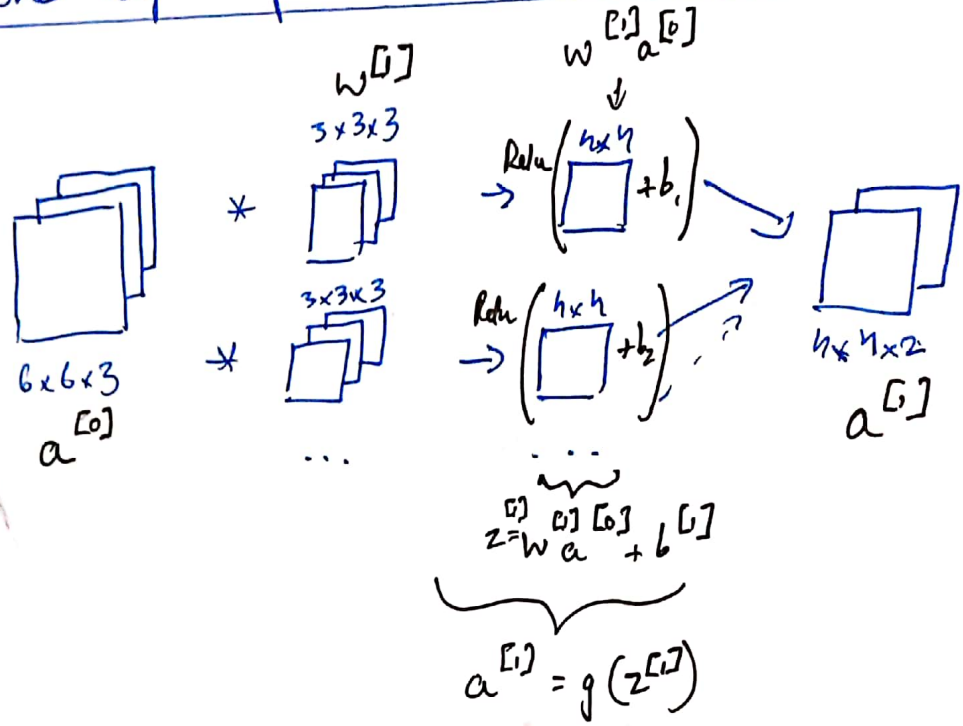
then it'll only check edges in red channel

Multiple filters



Summary:

One layer of a Convolutional Network



Number of parameters in one layer:

The no. of parameters remain the same for a NN. For example, for 10 filters that are 3x3x3,

1 filter = 3x3x3 = 27

+ 1 bias = 27+1 = 28

10 filters = 28x10 = 280 parameters

Notation summary:

If layer l is a convolution layer:

$f^{[l]}$ = filter size, $p^{[l]}$ = padding, $s^{[l]}$ = stride

$n_c^{[l]}$ = no. of filters, input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

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

where

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

floor value

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

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

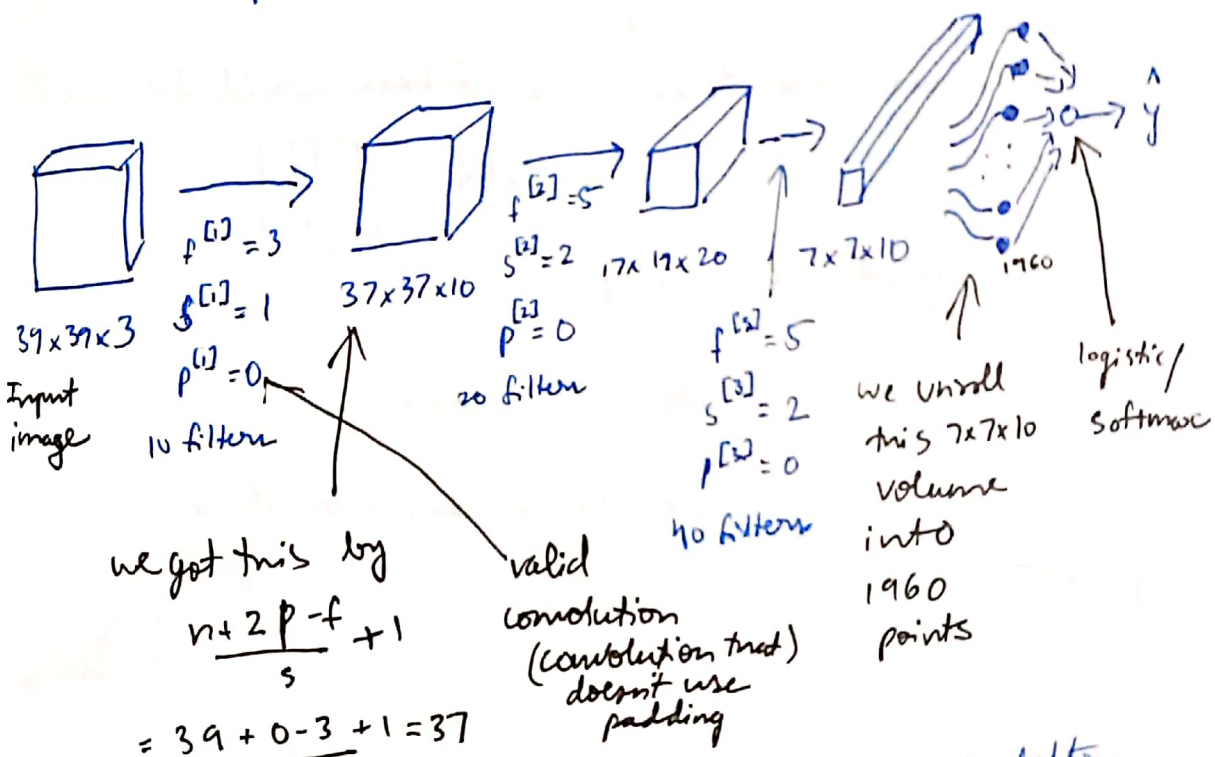
$$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$$

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

↑ #filters in layer l

bias : $n_c^{[l]}$ but in programming we use $(1, 1, 1, n_c^{[l]})$

Example of a convolutional network



→ Notice how the height & width decreases and filter size increases, this is a usual trend

Types of layer in a convolutional network:

→ Convolution (CONV) - we have discussed this

→ Pooling (POOL)

→ Fully Connected (FC)

} we will discuss this now

Pooling

It is a fixed function layer. No parameters are learnt here.

→ Max pooling - pick the largest no. in the window

Here largest is 9

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

→ $f = 2$, $s = 2$

9	2
6	3

Since we use this, the filter will be 2x2 and we will move like 

∴ output is 4x4

Another example (for all channels):

we do the same for other layers

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

$5 \times 5 \times n_c$

↑ no. of channels

9 is max

→ $f = 3$, $s = 1$

9	9	5
9	9	5
8	6	9

$3 \times 3 \times n_c$

There is also average pooling - where the average is taken instead of max. But this isn't used usually.

Hyperparameters for pooling:

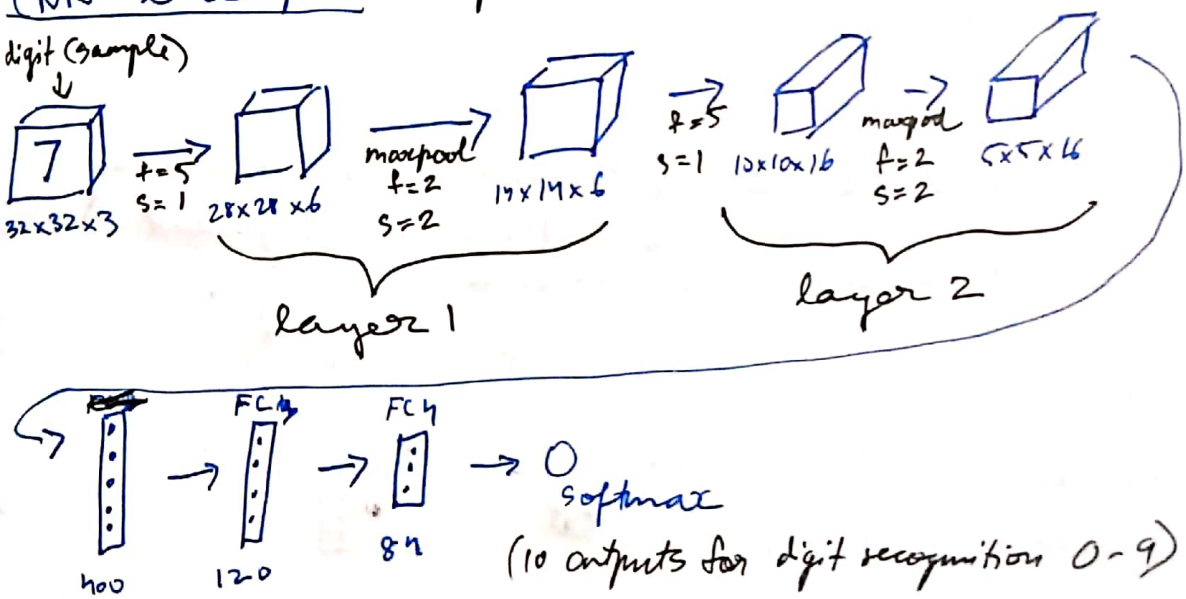
→ f : filter size

→ s : stride

→ Max or average pooling?

$$n_H \times n_W \times n_C \rightarrow \left[\frac{n_H - f}{s} + 1\right] \times \left[\frac{n_W - f}{s} + 1\right] \times n_C$$

CNN example - digit recognition



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

Layer 1 Layer 2

Table of above layer,

	Activation shape	Activation size	# parameters
Input:	(32, 32, 3)	3072	0
CONV1 (f=5, s=1)	(28, 28, 8)	6272	608 $(5*5*3+1)*8$
POOL1	(14, 14, 8)	1568	0
CONV2 (f=5, s=1)	(10, 10, 16)	1600	3216 $(5*5*8+1)*16$
POOL2	(5, 5, 16)	400	0
FC3	(120, 1)	120	48120 $400*120+120$ bias
FC4	(84, 1)	84	10164 $120*84+84$
Softmax	(10, 1)	10	850 $84*10+10$

Why convolution?

- 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
  - Therefore less parameters are required (If we used a normal NN, we may need millions of parameters)

- Sparsity of connections: In each layer, each output value depends only on a small no. of inputs 
  - allows translation invariance - if we shift the position of a cat in an image, it'll still recognise the feature