C4W1

Computer vision:-

if we have a image of 1000X1000 px. & want to do some deep learning operation on them. Then our input dimension will be 1000X1000X3 = 3M. Let's assume we have 1000 hidden unit in a single layer then our parameter size will be 3MX1000 = 3Billion which is a lot

Vertical image detection:-

Lets we have 6X6 gray scale 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

6X6

∗ convolution

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

Filter/Kernel (3X3)

=

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

4X4.

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

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

$1\times1 + 2\times1 + 0\times1 + 0\times5 + 0\times7 + 0\times1 + 8(-1) + 2(-1) + 3(-1) = -10$

⋮

How it detects edges:-

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

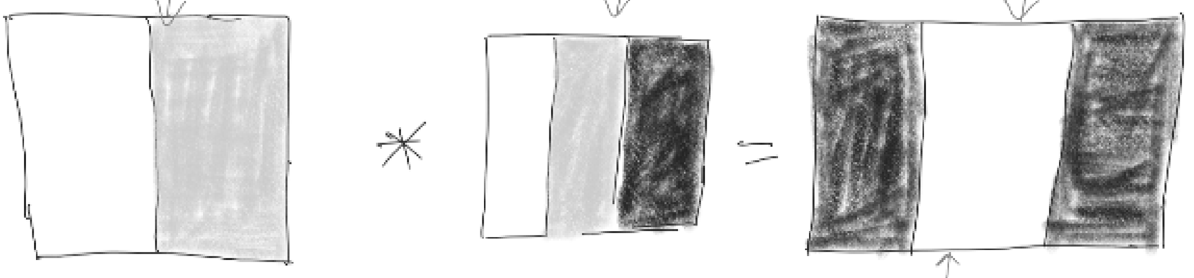
$*$

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

(4, 4).



there is a vertical line in the image

The values in the 4,4 matrix are positive that means we get a edge for light to dark color. But if get negetive value there then it means we get edges for dark to light color. if we don't care about the color we can just take the absolute value.

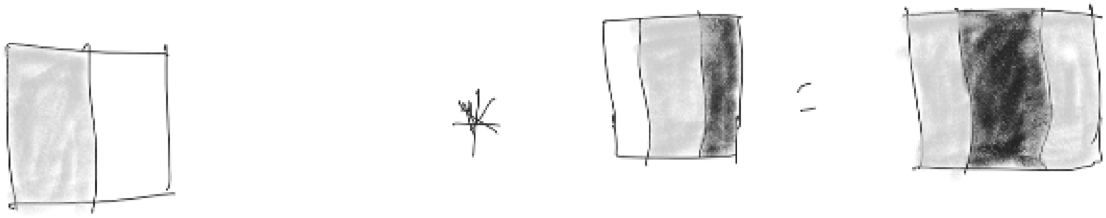
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$*$

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



Vertical & Horizontal edge detectors:-

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

Vertical edge detection filter.

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

Horizontal edge detection filter.

Other filters:

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

sobel filter
put more weight in the middle
makes it more robust.

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

scharr filter.

⇒ the better option is make the filter value as a learnable parameter through backpropagation:-

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

→ the NN will learn these automatically based on the image shape. it may also learn edge at any decline rate(45°, 73°, --)

Padding:-

If we have a matrix $n \times n$ & use filter $f \times f$ then we after applying convolution we will get a new matrix of $n-f+1$ by $n-f+1$

Downside:-

→ Everytime we apply filter our image shrink so the image get very small.

→ The corner pixel appears only once in the after filter images so we will loose a lot of information.

To fix both of the problem we can pad the image using the one extra pixel around the border. so if we have 6×6 image we will have 7×7 pixel image now. We usually pad by 0 & P is the padding amount. so the new filtered image becomes $n+2P-f+1$ by $n+2P-f+1$.

Valid convolution and same convolution:-

valid (no padding) → $n \times n * f \times f \rightarrow n-f+1 \times n-f+1$

Same :- Pad so that the output size is same as input size. for example:-

$6 \times 6 * 3 \times 3 \rightarrow 4$ we loose 2 pixel. pad so that our output is also 6.

$6 \times 6 \xrightarrow{\text{padding of 1 pixel}} 7 \times 7 * 3 \times 3 \xrightarrow{n+2P-f+1} 6 \times 6.$

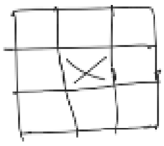
so to keep the same size we need to use the padding of:-

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

$$\Rightarrow 2P = f-1$$
$$\Rightarrow P = \frac{f-1}{2}$$

By convention of computer vision f is always odd. the reason is

- if f is even then we need assymetric padding $(P = \frac{f-1}{2})$
- in odd dimension filter it has a central position.



Strided convolution:-

Let stride size $= S$ so during applying fiter insted of sliding 1 px we will stride s pixel.

← we won't do the computation if we cannot fit the filter.

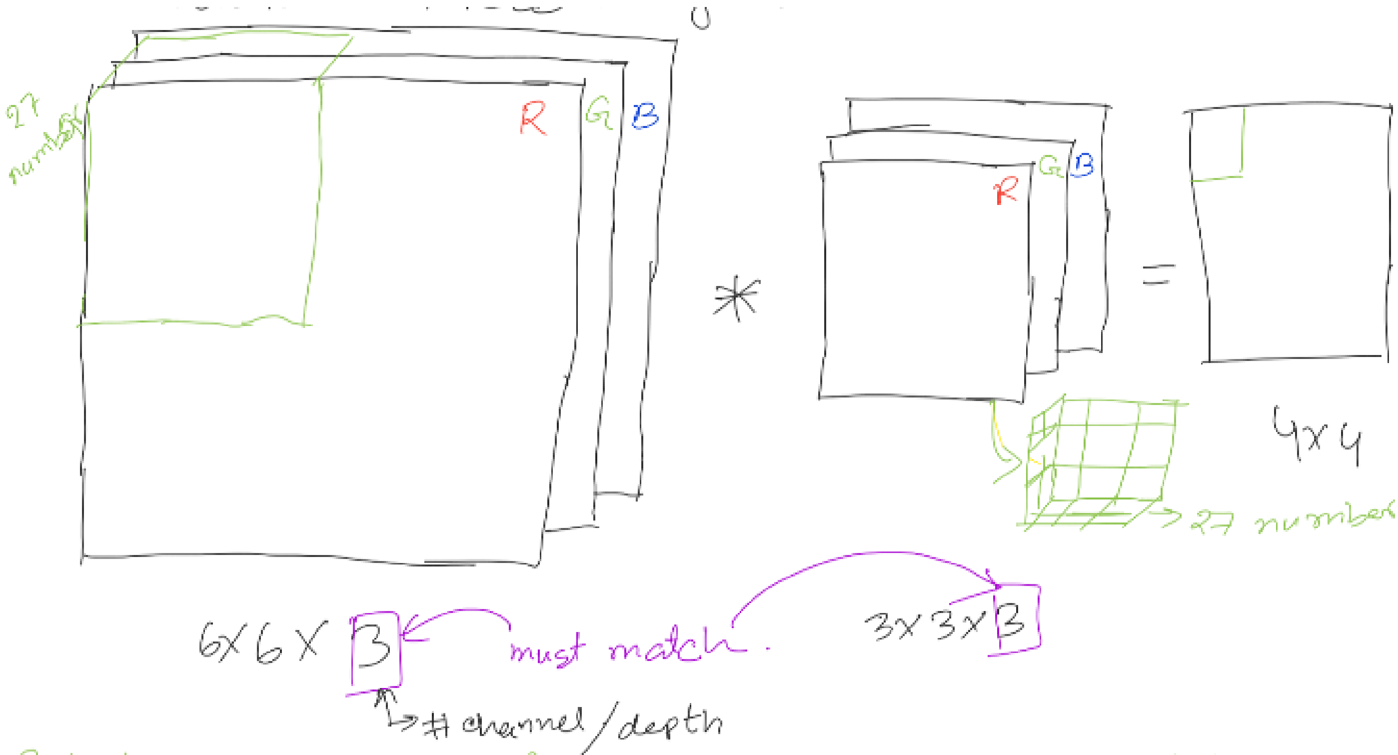
$$\begin{bmatrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ 4 & 2 & 1 & 8 & 3 & 4 & 6 \\ 3 & 2 & 4 & 1 & 9 & 8 & 3 \\ 0 & 1 & 3 & 9 & 2 & 1 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 91 & 100 & 83 \\ 69 & 91 & 127 \\ 44 & 72 & 74 \end{bmatrix}$$

new matrix size after applying stride size s will be :=

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

Convolution over volume:-

Convolution on RGB images:-



Put the 3D filter into the 3D image & multiply & add those ~~together~~. like before

if we only want to detect red edges then we will use this filter.

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

if we don't care about the color of the edge then we will use.

R:-

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

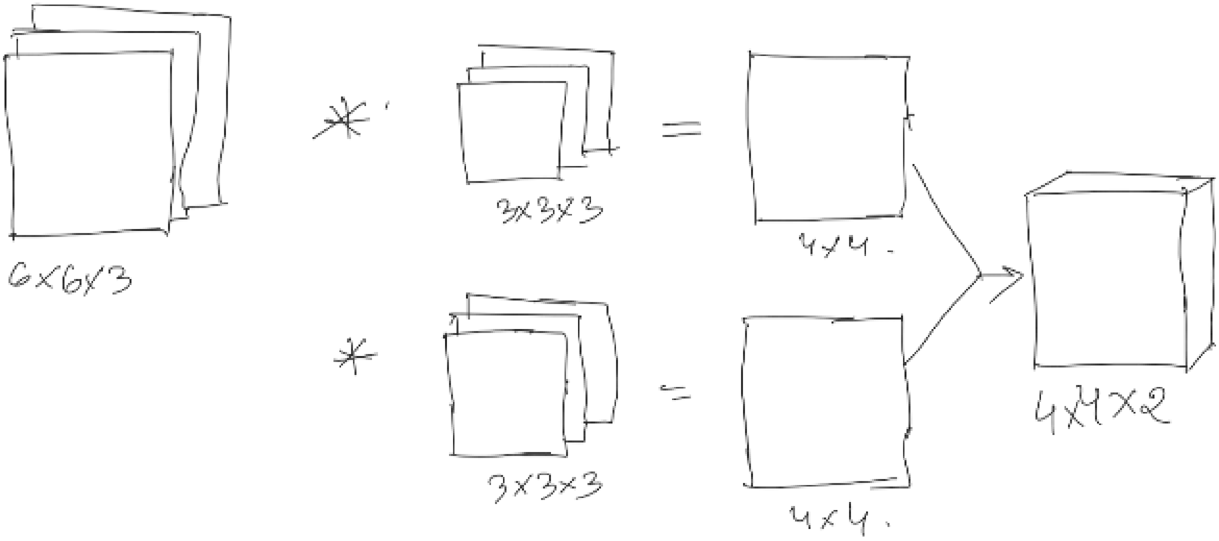
G:-

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

B:-

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

if we want to use multiple filter at the same time like want to detect both horizontal & vertical edges then we will stack the two filter result.

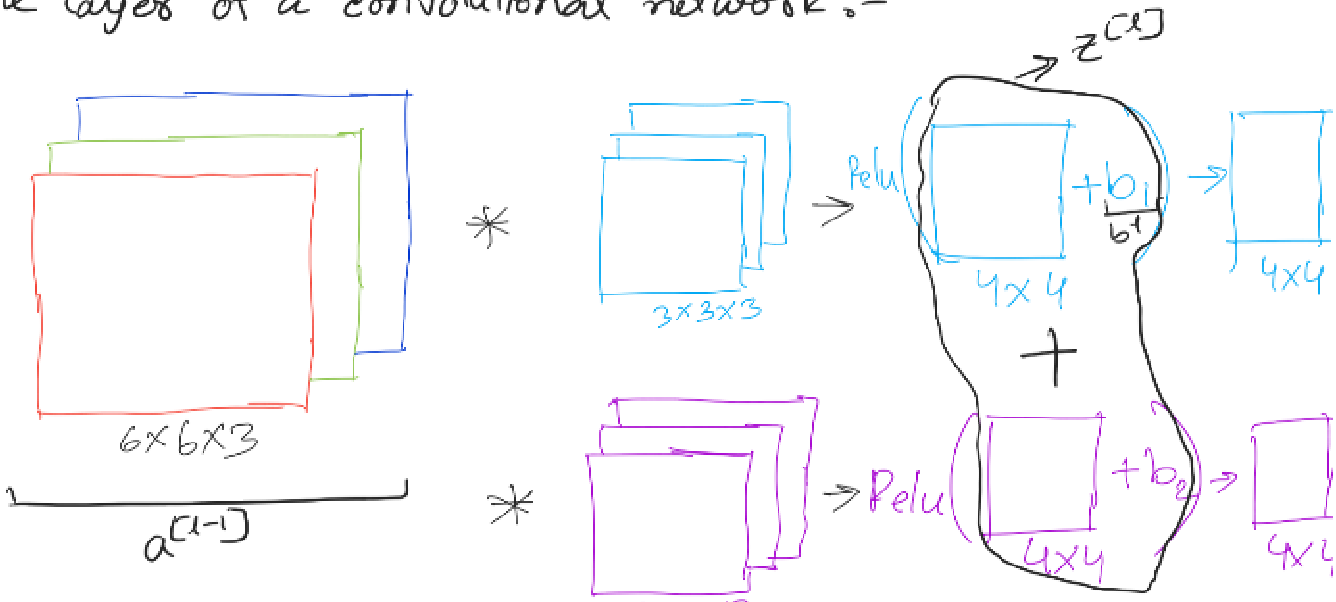


shape:-

number of filter

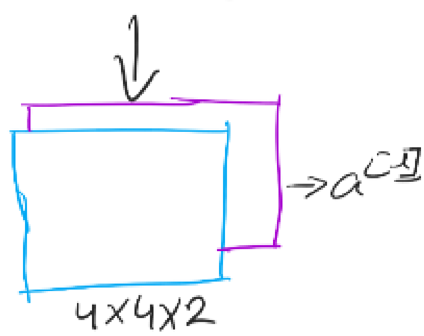
$$n \times n \times n_c \ast f \times f \times n_c \longrightarrow \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times n_c'$$

One layer of a convolutional network:-



$$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$$

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



Notation :-

$f^{[l]} \rightarrow$ filter size of layer l.

$p^{[l]} \rightarrow$ padding " " "

$s^{[l]} \rightarrow$ stride " " "

$n_c^{[l]} \rightarrow$ # filter used in layer l.

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

activation : $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ $\rightarrow$ for single training example.

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

weights :- $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times \boxed{n_c^{[l]}}$ $\rightarrow$ # filters in layer l.

Bias :- $n_c^{[l]}$

Input :-

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

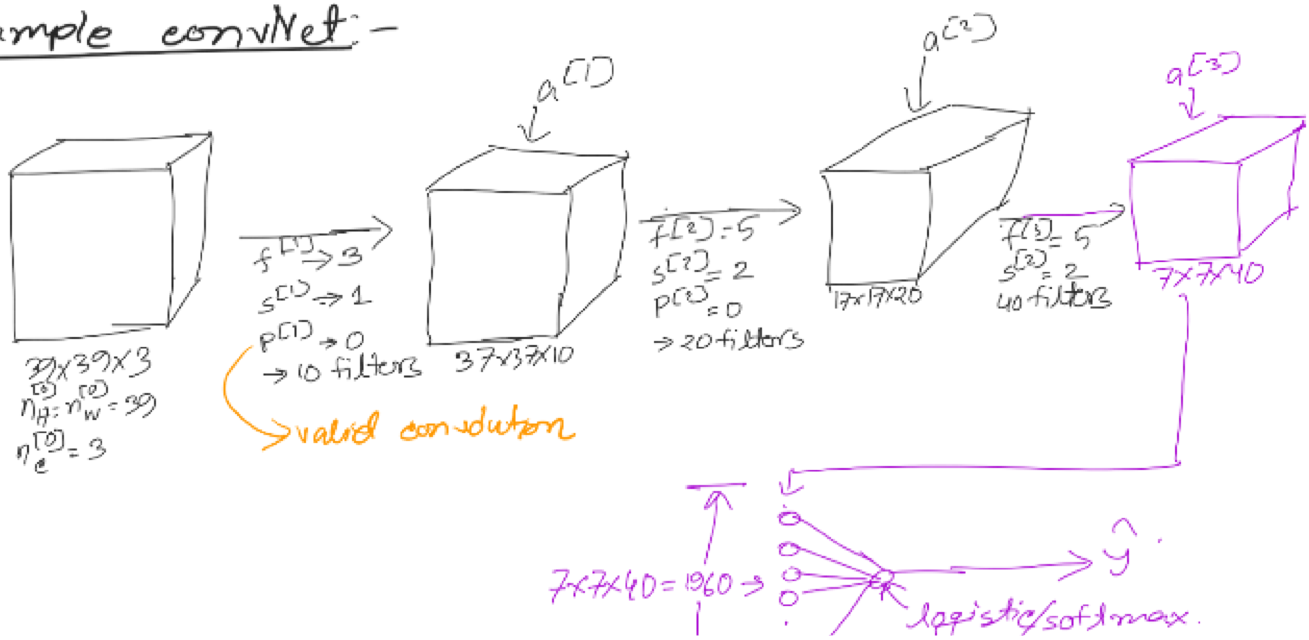
Output :-

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

where,

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

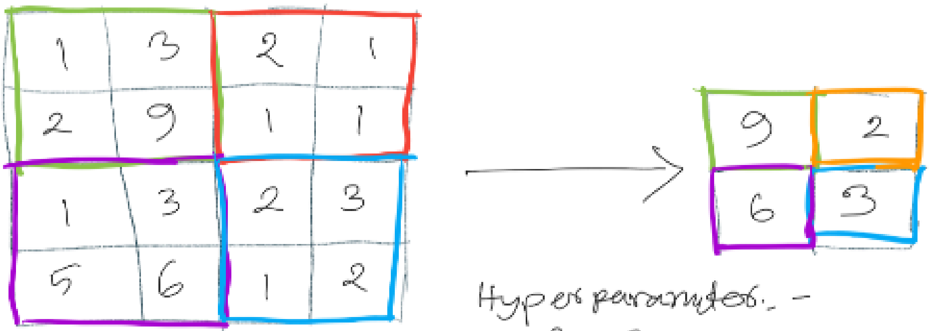
Example convNet :-



Types of layers in convolutional network:-

→ Convolution (conv)
→ Pooling (Pool)
→ Fully connected (FC)

<u>Pooling layer:-</u>



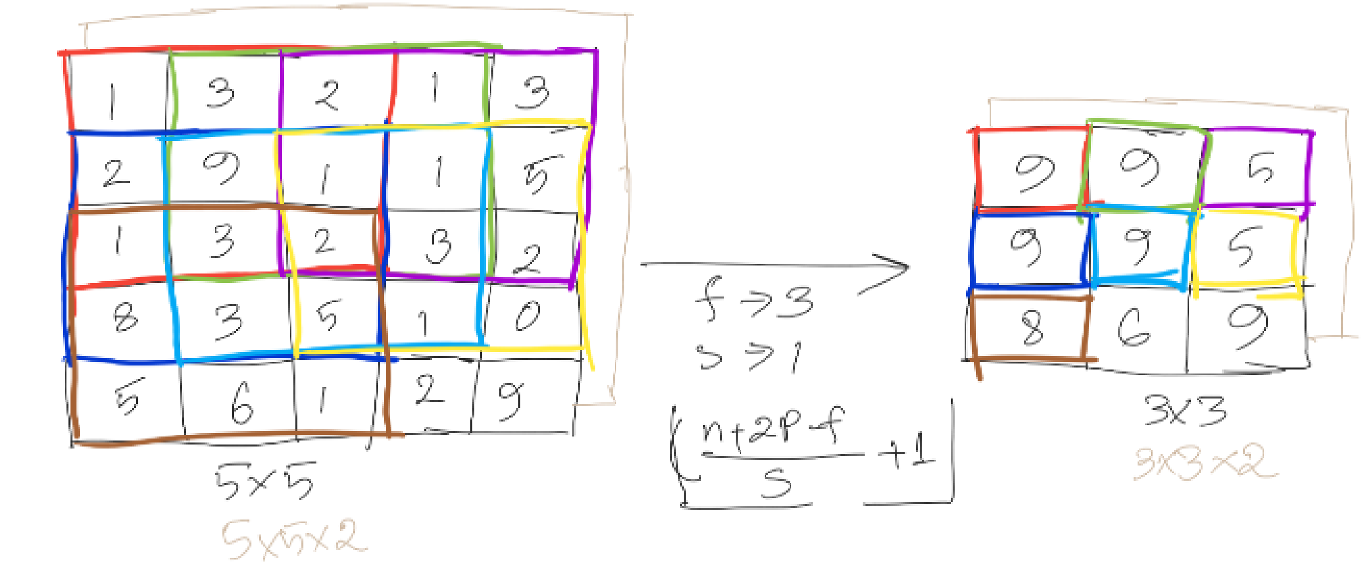
Hyperparameter:-
$f = 2$
$s = 2$

- There is nothing to learn for gradient descent in pooling layer.

Intuition:-

A feature might present in any one of the four section. So we are taking maximum of each section that represent how likely the feature is present in each section. However, there is no prove that this is the main reason of maxpooling working principle. It just work better in practice and no body knows why (for sure).

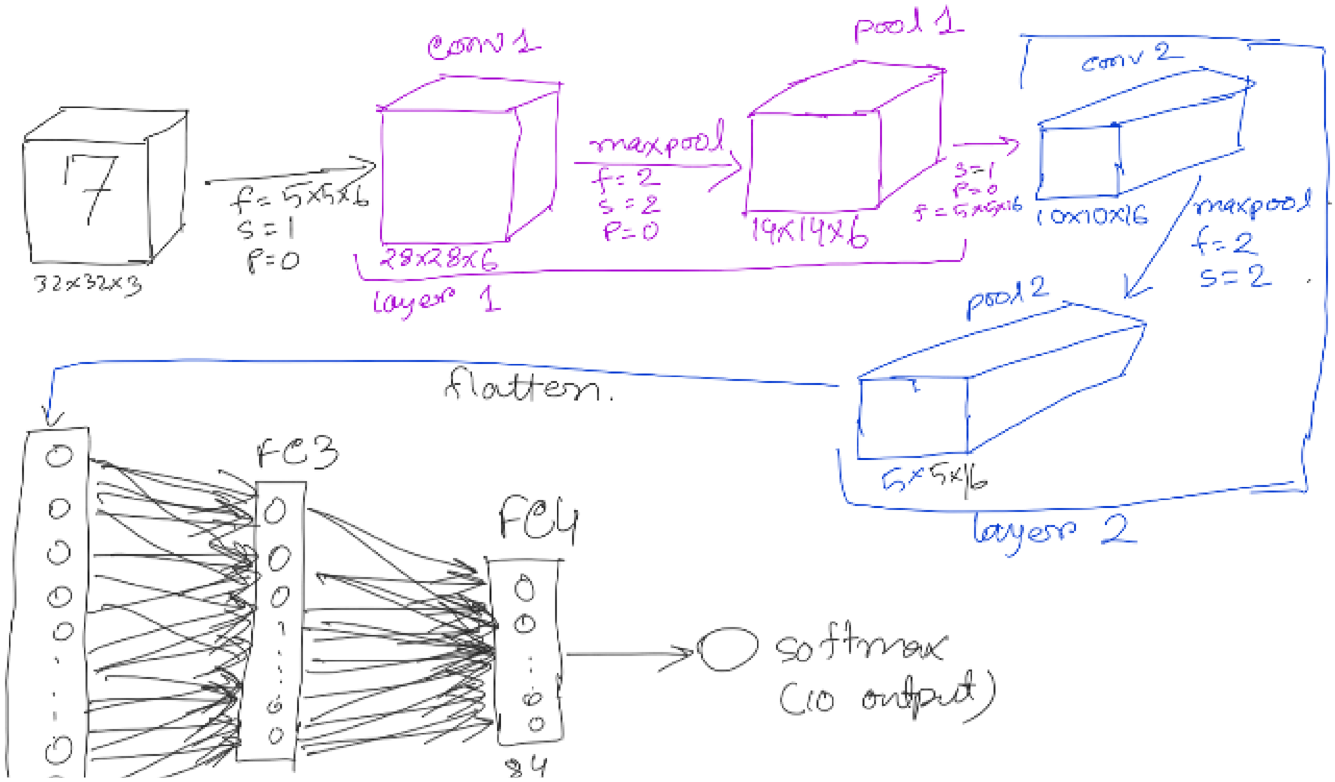
Another example:-



max pooling is done independently in each of the channel.

Average pooling:- Instead of taking max we will take average. Maxpooling is used much more than average pooling.

Convolutional neural network example:- (LeNet-5).



input
(400,)

120 hidden unit
$w^{[3]}$ (120,400)
$b^{[3]}$ (120,)

hidden unit

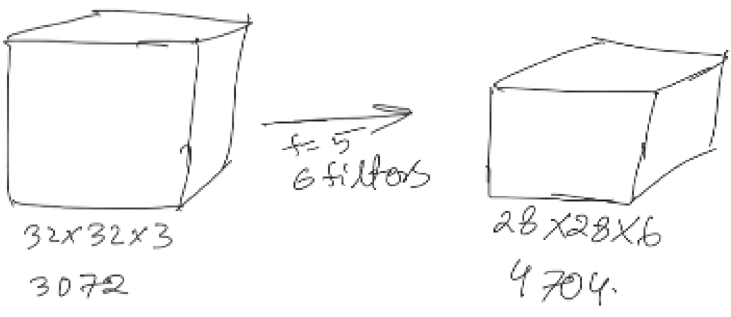
conv – pool – conv – pool – FC – FC – FC – softmax.

Parameter analysis.

	Activation shape	Activation size	# parameters
Input:-	32,32,3	3,072	0
conv1 (f=5, s=1)	28,28,8	6,272	((5×5×3)+1)×8= 608
POOL 1	14, 14, 8	1,568	0
CONV2 (f=5, s=1)	10, 10, 16	1,600	((5×5×8)+1)×16= 3216
POOL 2	5,5,16	400	0
FC3	(120,1)	120	400×120+120= 48120
FC4	(84,1)	84	120×84+84= 10164
softmax	(10,1)	10	84×10+10= 850

Activation size should gradually decrease if it decreases drastically the we won't get the performance.

why convolution?.



if we don't use CNN then need $3072 \times 4704 = 14M$ parameter
" " " " " " $(5 \times 5 \times 3 + 1) \times 6 = 456$ parameter

The reason that CNN needs less parameters is **parameter sharing**:- A feature detectors (i.e. vertical edge detector) that's useful in one part of the image is probably useful another part of the image (there might be multiple vertical edge in a single image).

Sparsity of connection:- In each layer each output value depends only on a small number of input.