Convolution Neural Network

Page No.
Date

CNN

- Image Recognition
- Object Detection & Localization
- Semantic segmentation
- Medical Image Analysis

Appln in Computer Vision & Image Analysis

for CNN we applied Image as Input, different types of Image Gray Scale, RGB.

Input Image → 2D metrix of pixels having perticular Numerical value (8 bit → 0 to 255) ($n_x * n_y$ → Input size)

$[n_x * n_y * 3]$ – Input size (3 → channels)

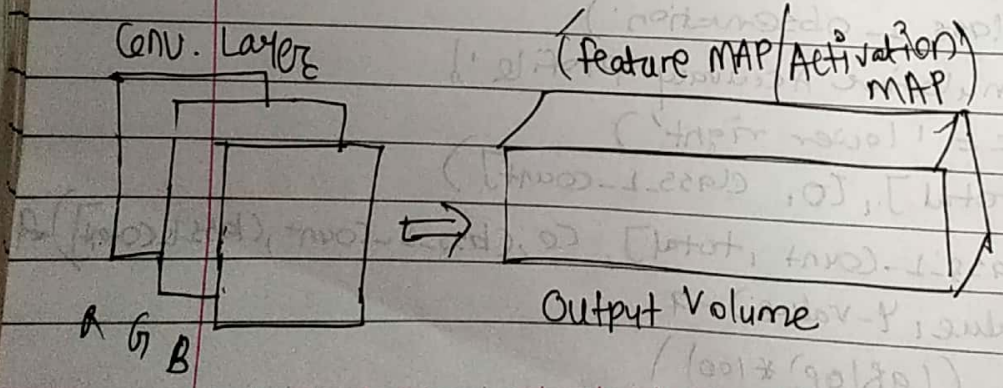
CNN layers & operation

Q. what CNN consist of ?

Like ANN, CNN contains stack of sequence layers followed by output layers - classifier layer

Each layer perforems a Convolution operation or a pooling operation

The conv. layer & pooling layers are alternatively stack leading to a series of fully connected layers followed by a output layer.



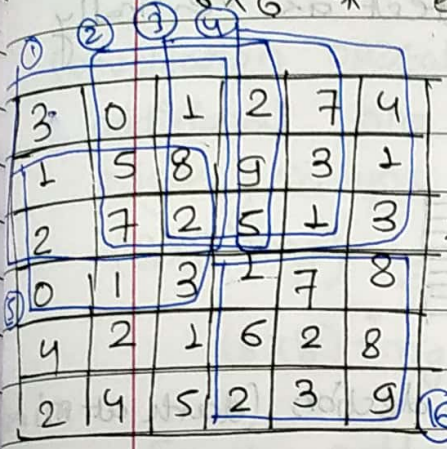
I/p have stack of feature vector

I/P I/p don't contain more than 3 channels then also

Andrew Ng.

Vertical Edge detection :

Ilp 6×6 * filter 3×3 ⇒ output 4×4



convolution

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

=

olp 4×4

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

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

$= 3+0-1+1+0-8+2+0-2 = -5$ ... at (1,1) in olp.

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

$= 0+0-2+5+0-9+7+0-5 = -4$ .... at (1,2) in olp

$(1\times1)+(2\times0)+(7\times-1)+(8\times1)+(9\times0)+(3\times-1)+(2\times1)+(5\times0)+(1\times-1)$

$= 1+0-7+8+0-3+2+0-1 = 0$ .... at (1,3) in olp

similarly do all of this shift column by column IF column end then shift 1 row down side, do same process for this also.

olp gives 4×4 Matrix

for Implementing CNN Convolution layer

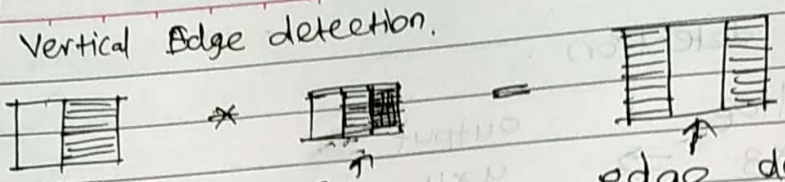
In python : conv_forward

tensorflow : tf.nn.conv2d

keras : conv2D

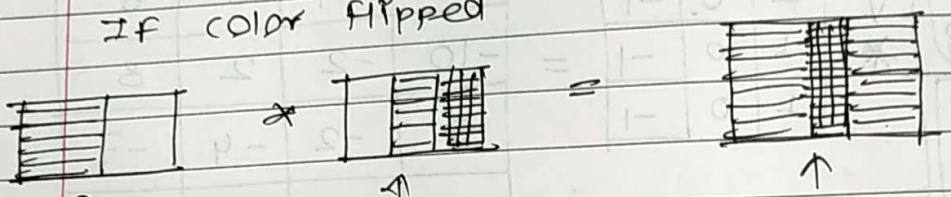
>1 - white 0 → darker 1 - ~~lighter~~ lighter <0 — more dark (black)

Vertical Edge detection.



conversion betn lighter to darker filter edge detect as vertically

If color flipped



dark to light filter O/p to edge dection (dark at middle)

So different filters allows you to find out vertical as well as horizental Edges

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

← sobel filter

Advantage
- more robust

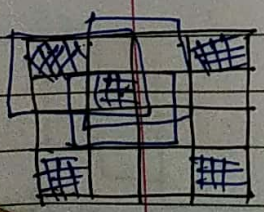
3	0	-3
10	0	-10
3	0	-3

← ~~Shaw~~ schurr filter

vertical & Horizental edge dection

Padding

Input diamention (nXn) Img * Filter diamention (FXF) = O/p Img diamention (n-F+1)x(n-f+1)



your corner pixel Images get evaluated once means it gives much less o/p & middle pixels ~~give~~ evaluated much more time so to evaluate this problems

you have to know both the shrinking outputs & Throwing away information from edges (corners) of Img

To fix both of these problems you can before Appling convolution operation you can padd the Image, with additional one border of one pixel around the edges, so you can do that then (6x6 Img) → Padded to (8x8 Img) & In convolution

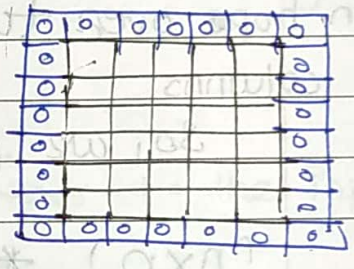
(8x8) Img * (3x3) filter ⇒ get (6x6) Img o/p

So you padded 0's

∴ 'p' is padding amount

ie p = 1 in this case

you padd extra 1 pixel around Img



In o/p become → $(n+2p-f+1) \times (n+2p-f+1)$

Ex - $(6+2(1)-3+1) \times (6+2(1)-3+1)$

$= (6+2-2) \times (6+2-2)$

$= (6) \times (6)$ Imp o/p

Valid & same Convolutions —

Valid → No padding

$(n \times n)$ Img $*$ $(f \times f)$ filter → $(n-f+1) \times (n-f+1)$ o/p

same → Pad so that output size is the same as the Input size then,

$(n+2p-f+1) * (n+2p-f+1) \Rightarrow n \times n$ then,

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

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

Ex - $f = 3$

$$p = \frac{3-1}{2} = \frac{2}{2} = \boxed{1}$$

↑ padd around Image

Page No.
Date

usually, F is always odd.

same padding gives you a natural padding as same image at (I/P = output).

Strided Convolutions

It is another basic building block of Convolutions as used in CNN.

In this when we slide the window of convolution, Instead of sliding one by one slide 2 steps so, we can hope over two steps like so on for rows as well as columns

So, we get (3×3) less diamention o/p Img.

$$(n \times n) * (f \times f) \Rightarrow \left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right)$$

Padding p Steide s

Ex – p=0, s=2

$$(7\times7) * (3\times3) \Rightarrow \left[\left(\frac{7+2(0)-3}{2}+1\right)\right] \times \left[\left(\frac{7+2(0)-3}{2}\right)+1\right]$$

Take floor

$$= (2+1) \times (2\times1)$$

$$= (3) \times (3)$$

Technical Notes on Cross-Correlation Vs Convolution

- Most Convolution Techniciens before applying convolution flip the Mateix of Filter first then apply convolution Operation.

flip

3	4	5
1	0	2
-1	9	7

flip

⇒

7	2	5
9	0	4
-1	1	3

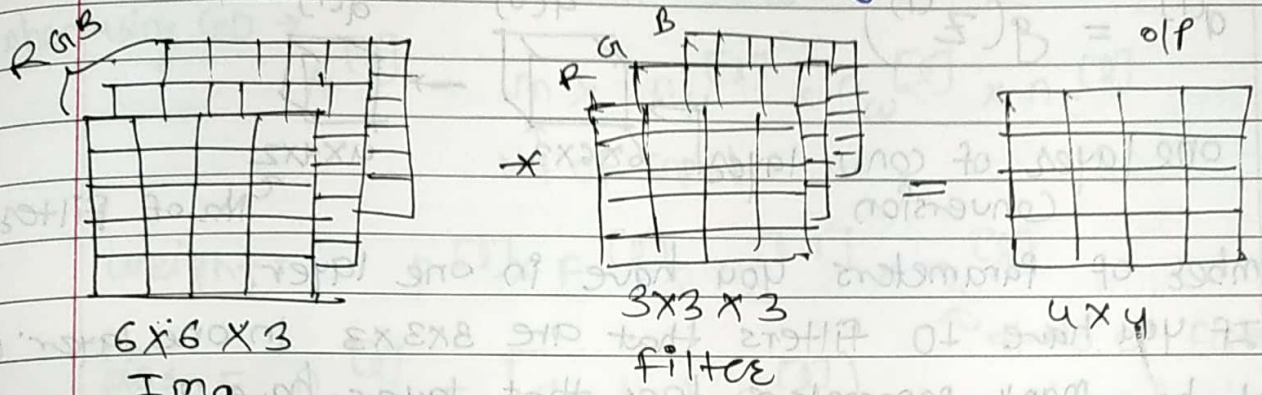
Cross-Correlation

But to simplify code In CNN we ommit this operation. & done convolution (should not effect anything)

Date

Convolutions over Volume

Convolution on RGB Image -



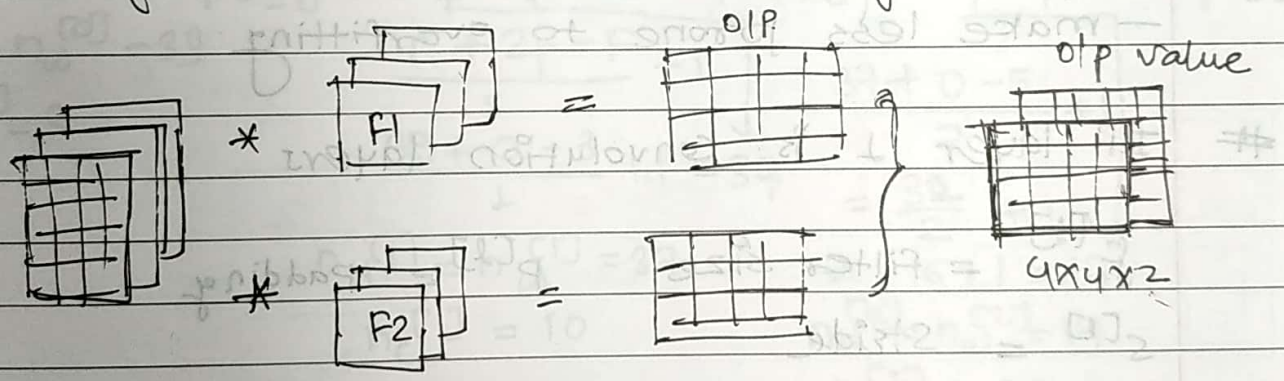
for 1st slide Sum all 27 No. = get single cell value in o/p Img.

Same for all cells.

Multiple filters -

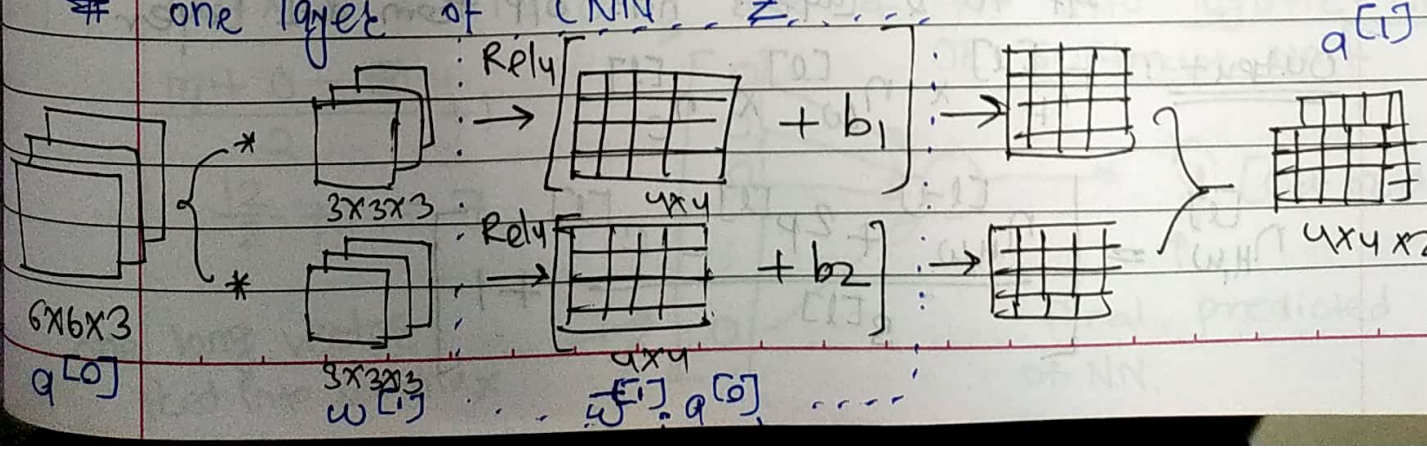
Apply more than one ~~inputs~~ filters to the inputs.

we get that much of 4x4 o/p Img.



$n \times n \times n_c \;*\; f \times f \times n_c \rightarrow (n-f+1) \times (n-f+1) \times n_c$

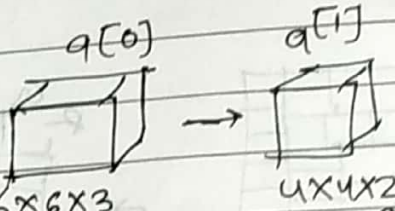
one layer of CNN



. . . w[1] . a[0] . . . .

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

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



one layer of conv. layer ↑ conversion

Number of Parameters you have in one layer.

Q. If you have 10 filters that are 3x3x3 in one layer of NN, how many parameters does that layer have?

→ filter (3x3x3) = 27 + bias = 28 x 10 filters = 280 Parameters

— No matter how big your i/p Img is → o/p parameter is 280 only

~~—~~ This ~~detects~~ features. of Ip img

property of CNN

— make less prone to overfitting

If layer l is convolution layers

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

$s^{[l]}$ = stride

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

H - Height W - width C - Channels

I/p to l layers has o/p from l-1 layer

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

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

$n_c^{[l]}$ – No. of filters Filter → $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

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

when using GD →

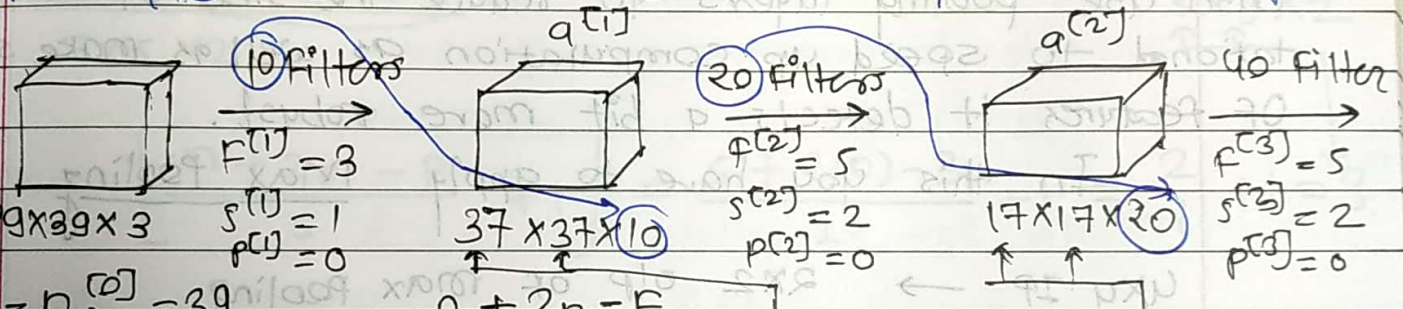
$$A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$$

(m ↑ No. of Input)

weights = $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ ('l' ↑ # No. of filters in layer)

bias = $n_c^{[l]} - [1, 1, 1, n_c^{[l]}]$

simple Convolution Network Example. [cat or dog]



$n_H^{[0]} = n_W^{[0]} = 39$

$n_c^{[0]} = 3$

$\frac{n + 2p - f}{s} + 1$

$\frac{39 + 0 - 3}{1} + 1 = 37$

$n_H^{[1]} = n_W^{[1]} = 37$

$n_c^{[1]} = 10$

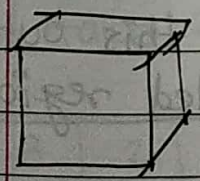
$\frac{37 + 0 - 5}{2} + 1$

$= \frac{32}{2} + 1$

$= 16 + 1 = 17$

$n_H^{[2]} = n_W^{[2]} = 17$

$n_c^{[2]} = 20$

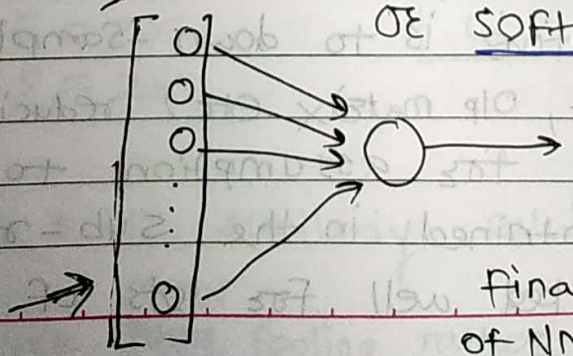


$7 \times 7 \times 40 = 1960$

$\frac{17 + 0 - 5}{2} + 1$

$\frac{12}{2} + 1 = 7$

Now take this volume (1960) size & fatten it (unroll it) into & fit into logistic unit or softmax unit



one long vector feed into softmax

final predicted o/p of NN.

as you go deeper in NN

Img (39x39) → (37x37) → (17x17) → (7x7)

& No of channels increasing as from

3 → 10 → 20 → 40

Types of layers in Convolutional NN –

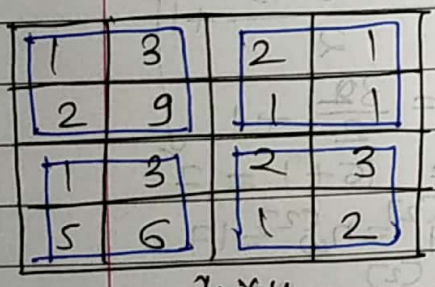
- Convolution Layer
- Pooling layer
- Fully connected layer

Pooling Layer – Max Pooling

other than Convolutional layers CNN often also use pooling layers to reduce the size of representational to speed up computation as well as make some of features it detects a bit more robust.

In this you have to apply Max Pooling

4x4 IP → 2x2 o/p of max Pooling

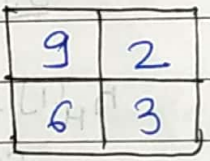


4x4

filter size = 2

regions

S = 2



2x2

take i/p break into diff. regions each of this output will be just max from correspondingly shaded region.

maxpooling hyperparameters

f = 2 , s = 2

Its objective is to down-sample an input representation (Img, hidden-layer, o/p matrix etc) reducing its dimensionality & allowing for assumptions to be made about features contained in the sub-regions.

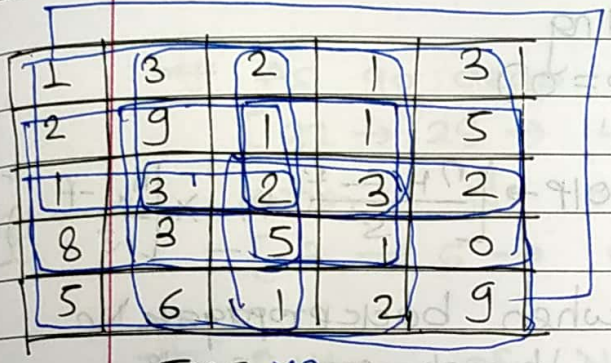
- It worked well for lots of example.

Page No.
Date

property of max pooling –

- It has no parameters to learn right there's actually nothing for gradient descent to learn. once you fix 'f' & 's' then GD desn't change anything

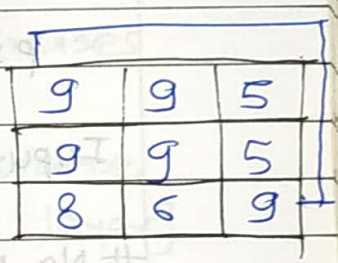
Ex-



5x5 x2

F = 3

S = 1



3x3 x2

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor = \frac{5+2(0)-3}{1} + 1 = \frac{2}{1} + 1 = 3$$

∴ Max Pooling Computation done Independtly on each Channels.

Pooling layer : Avorage Pooling

Instead of taking max value within each filter you take the average

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

Hyperparameters

F = 2

S = 2

3.75	1.25
4	2

$\frac{1+3+2+9}{4} = \frac{15}{4} = 3.75$ similarly calculate for all

Now a days we use ~~Max~~ Avg Pooling rather than ~~Avg~~ Max Pooling

Except, sometimes very deep in NN you might use Avg. pooling

Max pooling use much more in NN than Avg pooling

Hyperparameters of Pooling

f: filter size s: stride

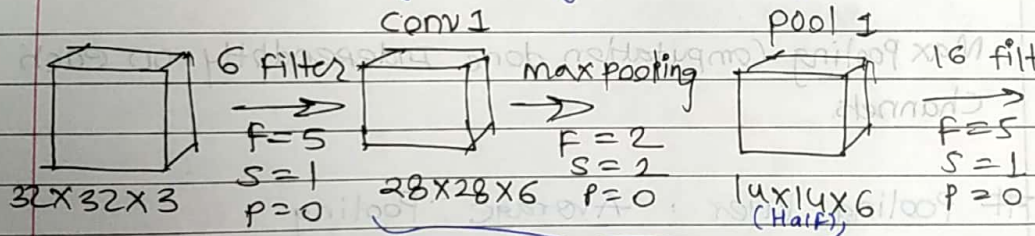
Max or Avg pooling

p: padding (mostly p=0)

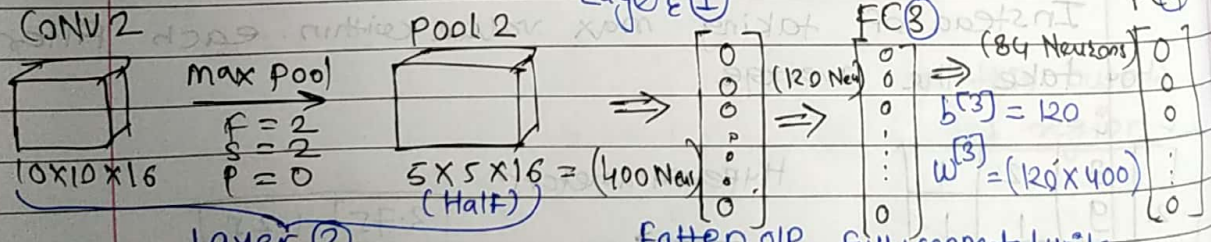
Input – $n_H \times n_w \times n_c$ → o/p → $\left[\frac{n_H - f}{s} + 1\right] \times \left[\frac{n_w - f}{s} + 1\right] \times n_c$

No parameters to learn → when backpropagate No parameters to backprop. adapt through max pooling i.e. Nothing to learn → fix parameters

NN. Ex- Recognize digit (0 to 9)



Layer 1



Layer 2

flatten o/p fully connected units

Generally the layer which has o/p as weight & parameters then calculate (said) layer but Pooling don't has any parameters & weight so (conv1 + pooling) → layer 1 called.

Layers = layers that have weights.

Fully connecte
Fully connected
each of the
Neurons just like
that you saw in
you have weight
also have

As go
[32 →
where
[3 →

Pattern
[conv
to see in

NN

Page No.
Date

Fully connected layer

Fully connected :-

each of the 400 Neurons connected to each of 120 Neurons just like the single Neural Network layer that you saw in courses 1 & 2 standard NN, where you have weight matrix.

also have Bias parameter.

As go deep your $n_H \times n_W$ decreases

[32 → 28 → 14 → 10 → 5]

where No. of channels will increase

[3 → 6 → 16 → Fully connected layer]

Pattern of NN (common pattern)

[conv + pool + conv + pool ... + FC + FC + .. + softmax]

to see in NN

NN Ex-

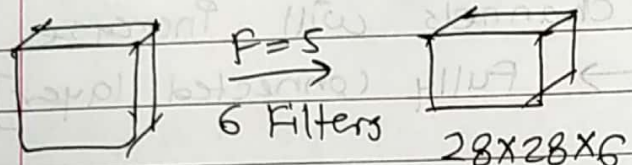
	Input	Activation shape	Activation size	Parameters
	→	(32,32,3)	3,072	0
	conv 1 (f=5, s=1)	(28,28,8)	6,272	208
finally recognize o/p [0 to 9]	pool 1	(14,14,8)	1,568	0
→ ŷ (10 o/ps)	conv 2 (f=5, s=1)	(10,10,16)	1,600	416
Softmax	pool 2	(5,5,16)	400	0
	FC3	(120,1)	120	48,001
	FC4	(84,1)	84	10,081
	Softmax	(10,1)	10	841

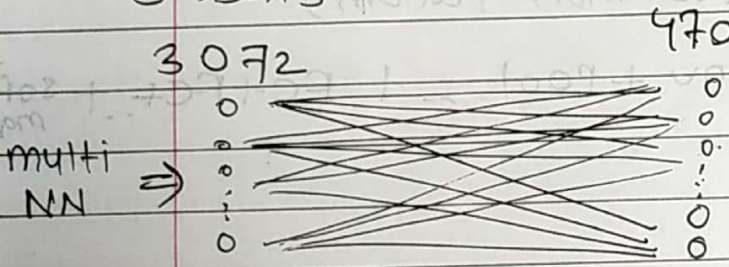
Activation size decrease gradually as you go in deeper → finally softmax o/p

Why Convolutions are so useful?

Advantages of CNN -

① parameters sharing
② Sparsity of connections





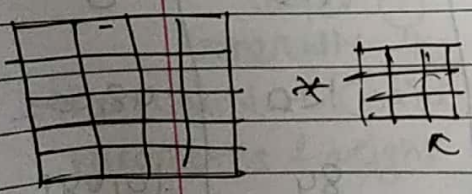
W = 3072 X 4704
W ≅ 14M

~~Not~~ so that is lots of parameters to train

Conv ⇒ (5x5) Filters = 25 + 1 bias = 26
6 X 26 = 156 parameters is small

Reason for conv NN have small parameters

① parameter sharing
- motivated by observation that a feature detector such as vertical edge that's useful in one part of the image is probably useful in another part of image



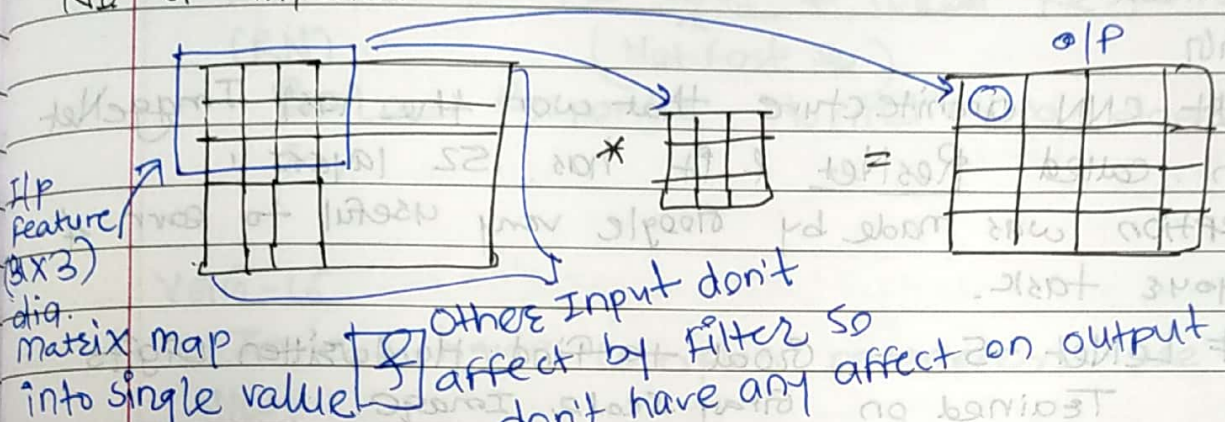
Same filter (parameters) are used for whole Image for feature extraction. There is probability that the upper left corner has same filters (parameters) used & same as lower bottom also, that's why having Less parameters

In each layer, each output value depends only on a small Number of inputs which makes it translation Invariance

Page No.
Date

② Sparsity connection:

In each layer, each olp value depends only on a small No of Inputs.



that why called sparse of connections. (loosely connected). so through this two mechanizam a CNN has a few parameters which allows it to be trained to smaller training sets & less prone to be overfitting.

After putting together.

fully connected NN at the end has weight & bias also, & works same as other NN.

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

Use gradient descent to optimize parameters to reduce J (cost function)