Convolutional Neural Networks

Computer Vision Application

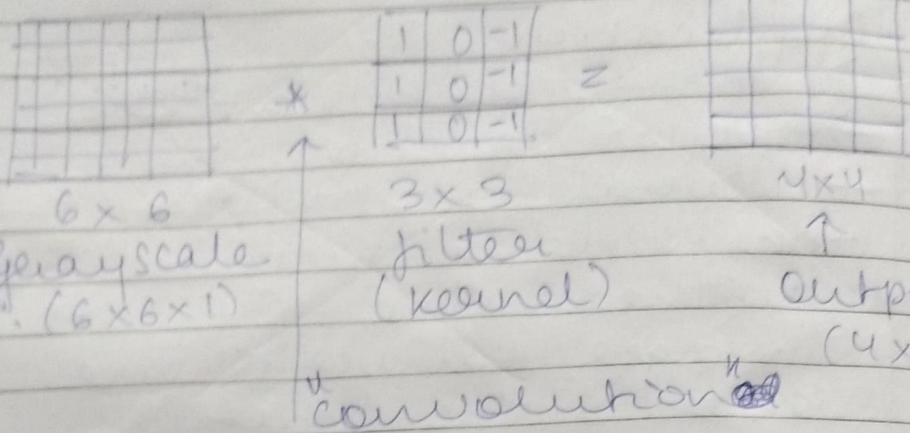
→ Image classification
→ Object detection
→ Neural style transfer → re-painting a picture in a different style. Inputs 2 images converts one to style of the other

→ One of the challenges of CV problems is that the inputs can get really big.
Eg for a 1000 × 1000 image input will have 1000 × 1000 × 3 input features x as its in RGB. If 2nd layer has 1000 units then W1 will be (1000, 3 million) dimensional matrix ∴ W1 will have 3 billion parameters
→ its difficult to prevent overfitting also there are computational & memory requirements
→ Therefore we use convolutional networks

Edge detection

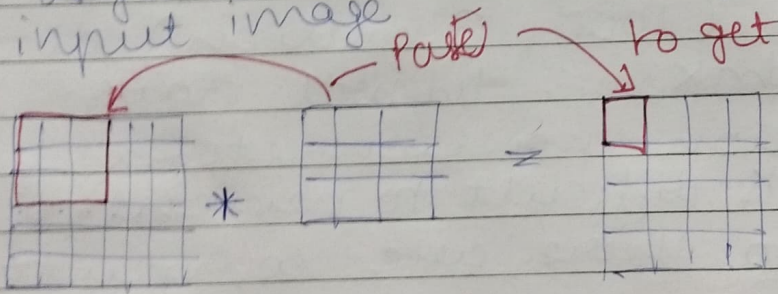
→ Convolution operation is a fundamental building block of convolutional neural networks

How edge detection works



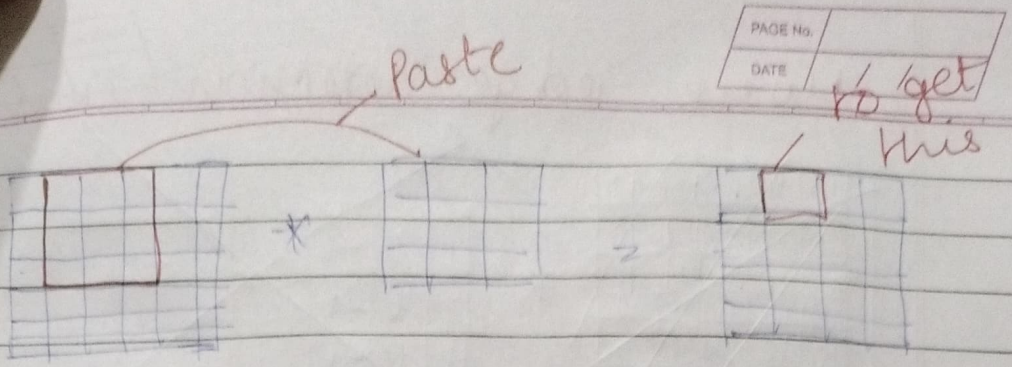
→ In math * is convolution
→ Since in python * is multiplic it is overloaded

→ To calculate upper left of 4×4 what we do is take the 3×3 filter and paste it on top of originale the 3×3 region of orginal input image



Take element wise product when we paste the filter
→ Then add up the sum of all elements

→ Move filter by 1 column to get 2nd pixel and follow same steps



→ and so on to fill the 4×4 matrix

→ This is our edge detector

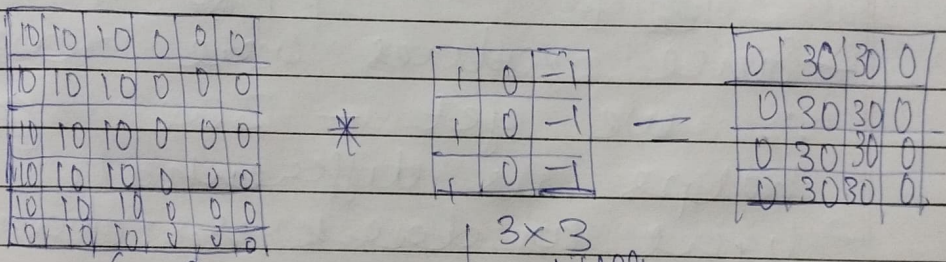
→ To denote the convolution various languages have various functions

Python : Conv-forward

tensorflow : tf.nn.conv2d

Keras : Conv2D

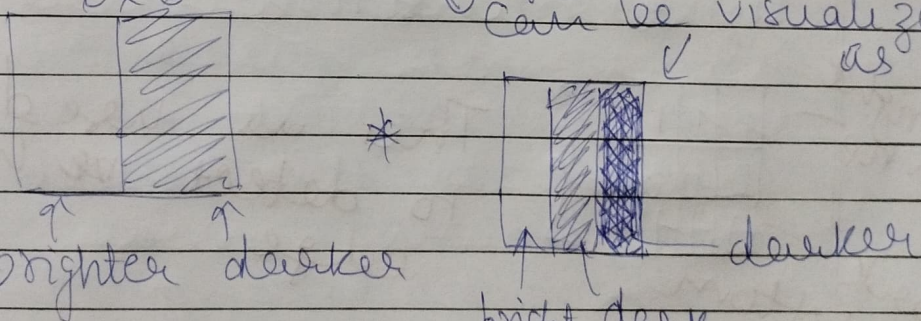
Vertical edge detection



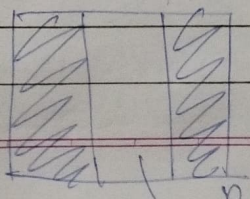
6×6

3×3 filter

can be visualized as



If product is plotted (4×4) matrix:



Brighter region in middle corresponds to having detected an edge down the middle

Bright center

6×6 image

Eg 2)

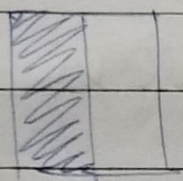
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

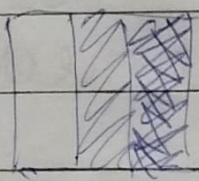
*

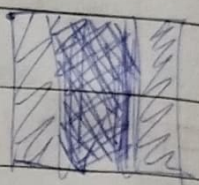
1	0	-1
1	0	-1
1	0	-1

=

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







- Since values are reversed
∴ we get − 30 this helps us differentiate btw light to dark & dark to light transition

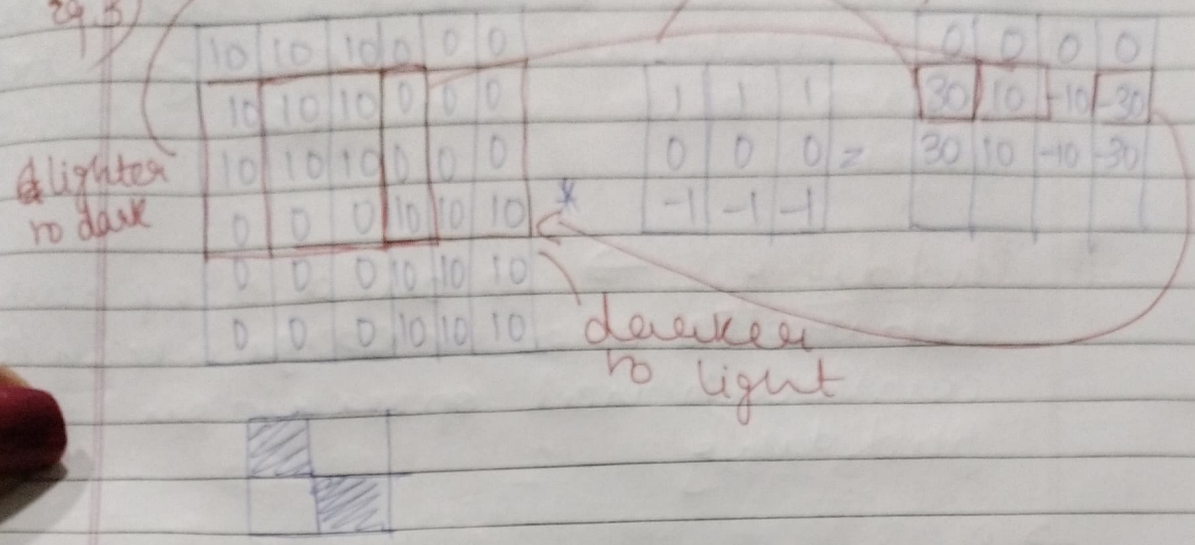
bright on top

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

dark on bottom

This is used to detect the horizontal edge

Eg 3)



→ 3×3 filter is one possible choice

→ | 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

Puts a little more weight to central pixel ∴ more robust

Sobel filter

→ | 3 | 0 | -3 |
| 10 | 0 | -10 |
| 3 | 0 | -3 |

Scharr filter

→ To detect vertical edges in complicated image you don't need to handpick nos for the filter we can treat them as parameters and learn using backpropagation to get a good edge detector

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

→ This can be used not only for vertical & horizontal edges but also for individual edges

Padding → To build deep neural networks one modification to convolutional operation is padding

→ If I convolve an n×n image by f×f filter we get the o/p image by
$(n-f+1) \times (n-f+1)$

→ We can only do convolutions a few times as image shrinks slightly every time and we dont want image to shrink every time we detect edges on it

→ Second downside is that pixels at corners or edges are used much less in output ∴ lot of info near the edge of image is thrown away

This is solved by padding image by adding a border of 1 pixel around the image.

∴ 6×6 becomes 8×8
Where p = padding = 1
→ Padding is done usually with 0's conventionally

→ New o/p image size:
$(n+2p-f+1) \times (n+2p-f+1)$

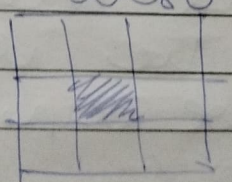
Valid & same convolutions

→ These are 2 common choices in terms of how much to pad

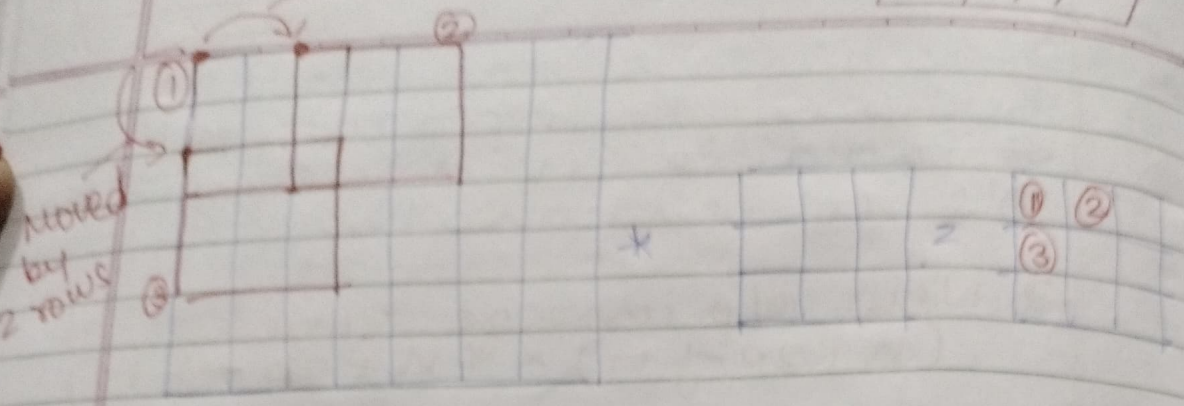
no padding
"valid" : $n \times n * f \times f \rightarrow (n-f+1) \times (n-f+1)$

→ "same" : Pad so o/p size is same as i/p size

filters are usually of odd order $f \times f$ (f is odd (helps give asymmetric pattern & without asymmetric padding) also has a central position



Strided convolutions



If stride is 2 instead of moving by 1 column or row we move by 2

order of o/p matrix :-

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

If this is not a whole no we round off to nearest integer

(z) → floor(z) — means taking z & rounding to nearest integer

→ It means that if the filter does not lie entirely in the image we skip that computation

→ In math to before moving the filter and doing all the steps there is one step to be done before

→ Flip the filter on horizontal as well as vertical axis

Eg

3	4	5
1	0	2
-1	9	7

→

7	9	-1
2	0	1
5	4	3

→ If we skip the above operation it is technically cross-correlation instead of convolution

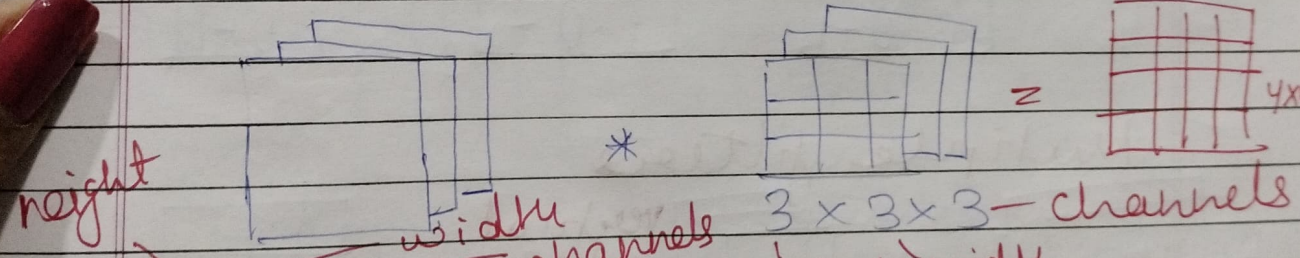
→ This operation allows associativity

$$(A * B) * C = A * (B * C)$$

→ This is not needed for convolutional neural networks so we skip it

Convolutions over volume

Convolutions on RGB images



→ An RGB image is a stack of 3 6×6 images so the filter is a stack of 3 3×3 matrices

→ Channels of image & filter must be equal

→ We can consider the 3×3×3 filter as a 3D cube
→ The 3×3×3 filter has 27 parameters
→ We multiply each layer of filter with corresponding R/G/B channel then add all 27 multiplications (9 for each channel)
→ Each filter layer can have diff parameters

Eg

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

→ 3×3×3

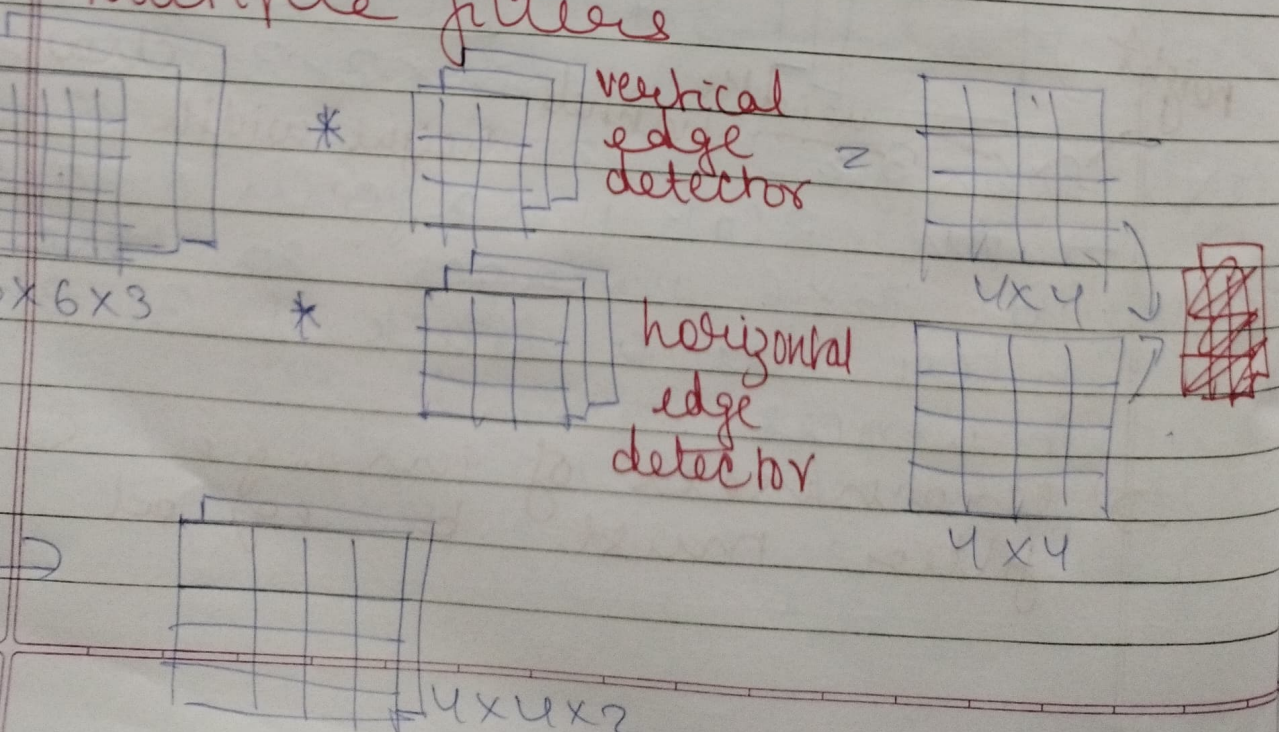
1	0	-1
1	0	-1
1	0	-1

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

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

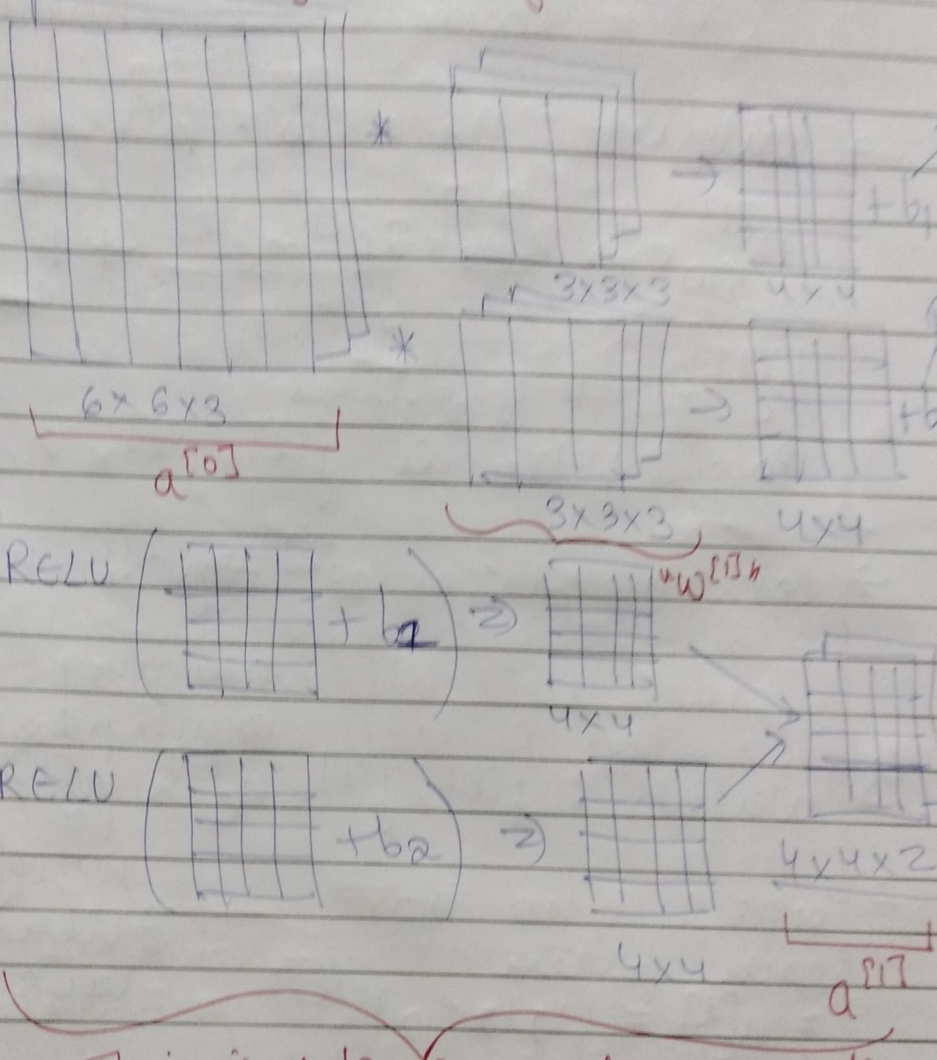
→ 3×3

Multiple filters



→ This is used to find horizontal and vertical edge simultaneously

→ Example of a layer



This is 1 layer of convolutional Network

→ The parameters depend on size of filter & no of filters ∴ no matter how big an image the no of parameters are unaffected. This prevents overfitting

Summary of notation

→ If l is a convolution layer,
$f^{[l]}$ = filter size
$p^{[l]}$ = padding
$s^{[l]}$ = stride

→ Input = $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$
(n_H = height, n_W = width)

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

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

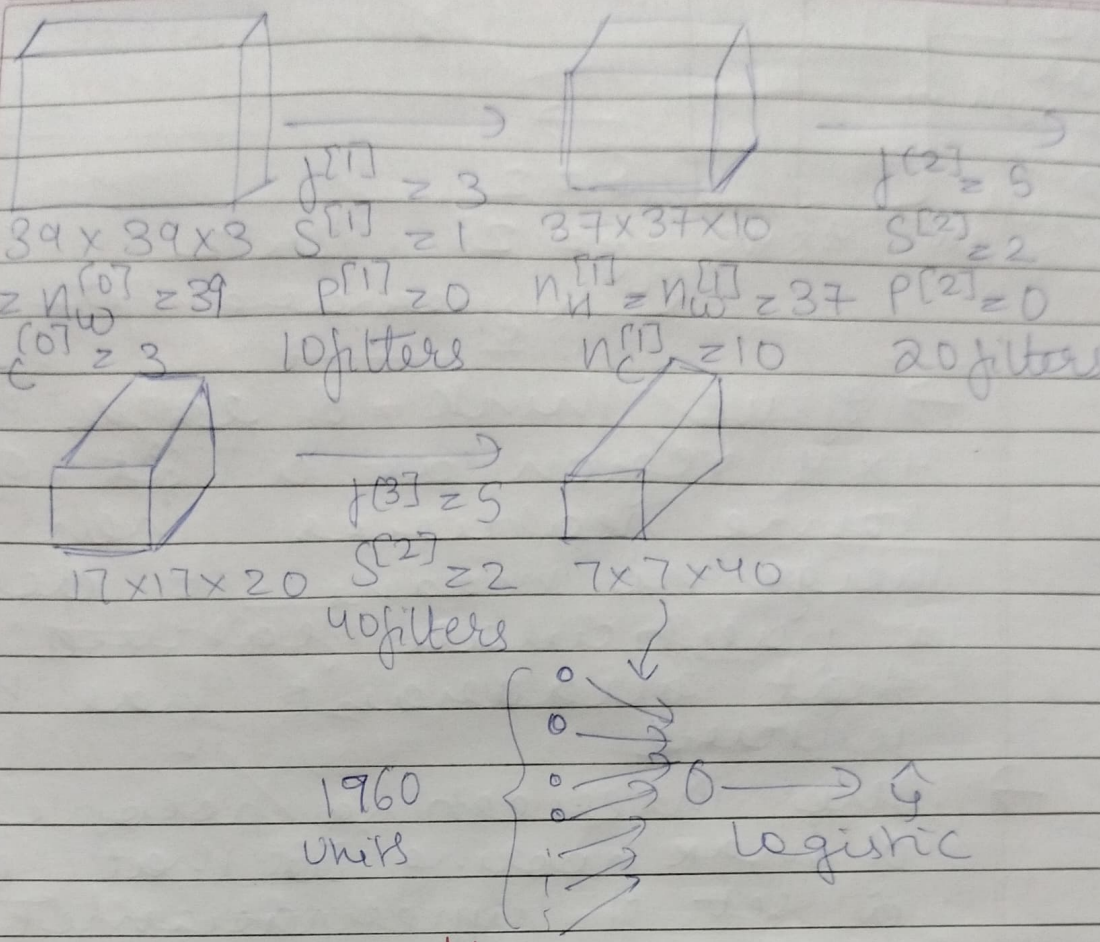
$n_C^{[l]}$ = no of filters
Each filter is $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$

→ Activations : $a^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
(m = training examples)

→ Weights : $f^{[l]} \times f^{[l]} \times n_C^{[l-1]} \times n_C^{[l]}$
($n_C^{[l]}$ = no of filters in layer l)

→ bias : 1 no for each filter
$= n_C^{[l]} = (1, 1, 1, n_C^{[l]})$

Simple convolutional Network

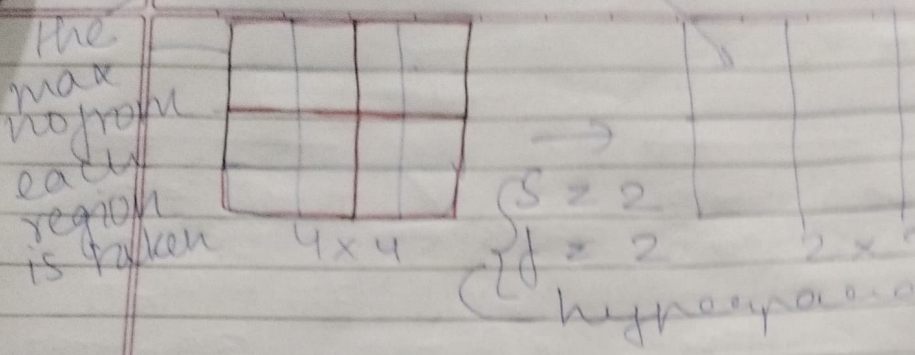


→ We usually see that as no of channels ↑ the height & width ↓

☆ Types of layer in Convolutional network
→ Convolution (CONV)
→ Pooling (POOL)
→ Fully Connected (FC)

Pooling Layer:

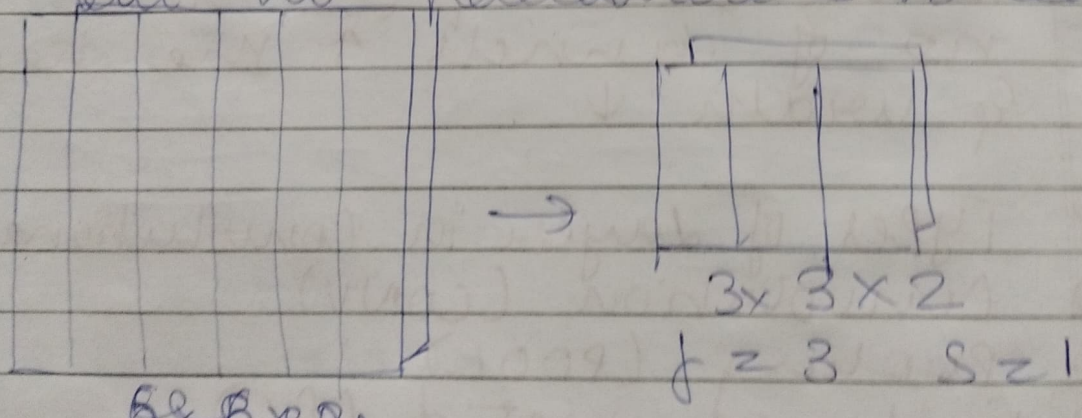
Max Pooling



→ If we think of each the 4×4 region as a set of some features then a large no means it detected a particular feature so that quadrant has that feature

→ One interesting property of max pooling is that it has a set of hyperparameters but no parameters to learn

Eg



→ Max pooling is done independently on each channel

→ It is not used very often

Average pooling - v take avg instead of max

→ Sometimes in deep network we use avg pooling to collapse representation from say 7×7×1000 to 1×1×1000

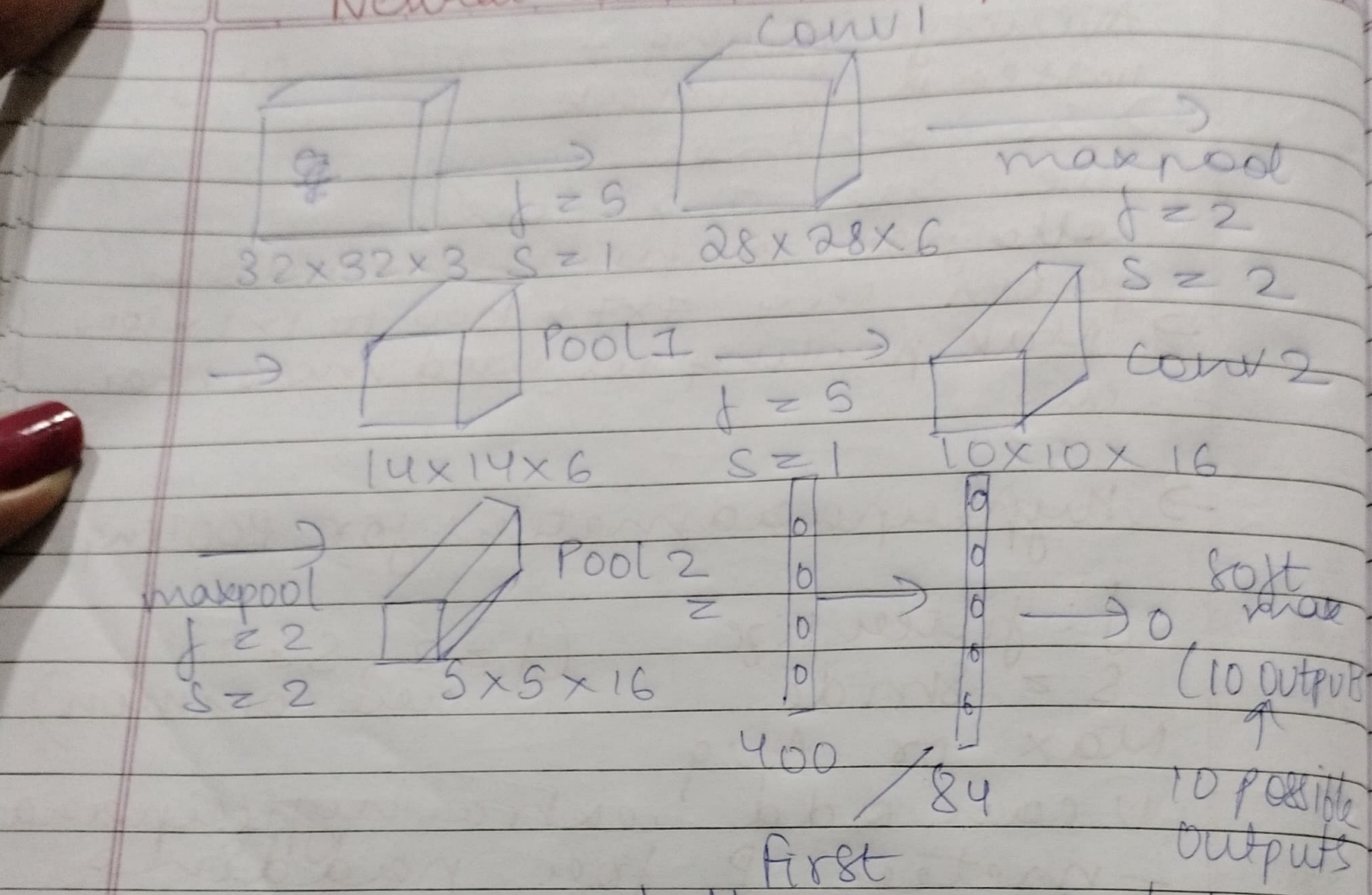
→ Max pooling is used more than avg

→ Hyperparameters for Pooling:

f: filter size
S = Stride (f=2, S=2 are common)
Max or Avg
v can add extra hyperparameter P for padding although not used in pooling mostly.

$n_H \times n_w \times n_c \longrightarrow \left[\frac{n_H - f + 1}{S}\right] \times \left[\frac{n_w - f + 1}{S}\right] \times n_c$

Neural Network Example



Conv1 to Pool1 is Layer 1
Conv2 to Pool2 is Layer 2

Conv → Pool → Conv → Pool → FC → FC → FC → Softmax

⇒ Max pooling layers don't have any parameters

⇒ Size (ie no of activations in a layer) reduces as network progresses

PAGE No.
DATE / /

Why convolutions

→ No of parameters to train is less as they only depend on filter size (and biases) and no of filters

→ Reasons for less Parameters

1) Parameter sharing — A feature detector (such as vertical edge detector) that's useful in one part is also useful in another part of the image (eg the same filter is pasted on different sections of the image)

→ Sparsity of connections — In each layer each O/P value depends only on small no of inputs (value of pixel in O/P layer only depends on those of I/P layer used to derive it)