Convolutional Neural Networks

-Intro.
-Filters : Edge detection ex.
-Intuition

Divyanshu Vyas

THE CONVOLUTION (*) (Example) Operation : Edge Detection Example

Early layers → Detect edges

⇓

Later layers (Deeper) → Detect parts of objects

⇓

Still Deeper layers → Detect complex objects.

How to identify edges →

Suppose we have a grayscale image → (x, y, 1)

[RGB : (x, y, 3)]

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

6x6x1

grayscale image

*

Convolution operator

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

3x3

Filter OR kernel

=

A_1	B_1	C_1	D_1
A_2	B_2	C_2	D_2
A_3	B_3	C_3	D_3
A_4	B_4	C_4	D_4

4x4

Final (F)

(termbyterm)

$$A_1 = \sum \begin{bmatrix} 3 & 0 & 1 \\ 1 & 5 & 8 \\ 2 & 7 & 2 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \longrightarrow \begin{pmatrix} 3 \\ +1 \\ +2 \end{pmatrix} + \begin{pmatrix} 0 \\ +0 \\ +0 \end{pmatrix} + \begin{pmatrix} -1 \\ -8 \\ -2 \end{pmatrix}$$

$$\downarrow$$

$$\boxed{-5}$$

∴ $\boxed{A_1 = -5}$

$B_1 = 12 + 0 - 16 = \boxed{-4}$ → → →

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

A 6x6 matrix convolved with a 3x3 matrix gives you a 4x4 matrix.

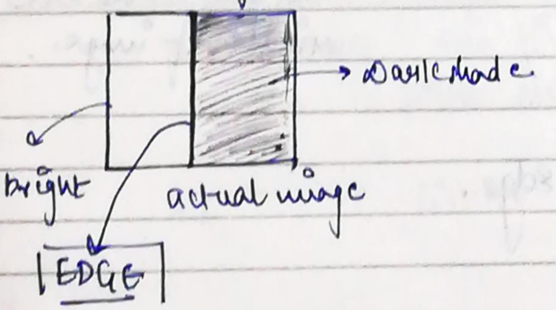
→ Images & Filters are just matrices of various dimensions.

Original image (6x6) $\circledast$ Filter (3x3) = Convolved image (4x4)

ways to implement convolution → ① python : conv-forward
② Tensorflow : tf.nn.conv2d
③ Keras : Conv2D

→ The above example was of a vertical edge detection in a grayscale image.

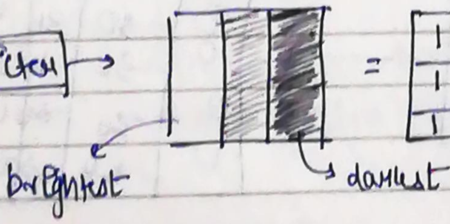
Lets clarify it by following example :→



grayscale →

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

Filter →



=

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

∴

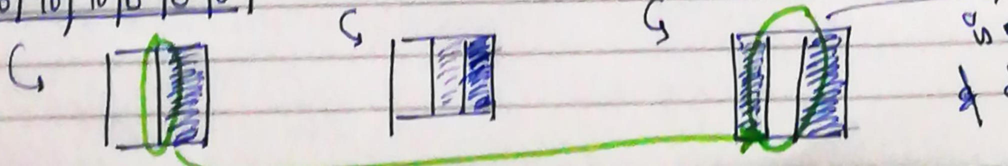
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$\circledast$

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



The edge is magnified & detected.

Vertical Edge Intuition → A ~~regi~~ matrix (3x3 etc.) having white pixels on the left & Dark pixels on the right.

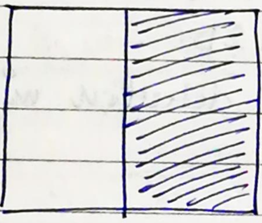
ex:-

10		0
10		0
10		0

→ Vertical Edge.

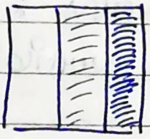
→ Convolution operation helps in edge detection

MORE ON EDGE DETECTION → +ve Edges | -ve Edges etc...



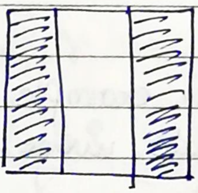
Light to Dark Edge Image

*



Vertical Edge Detection Filter.

=



Vertical Edge detected in the middle of image.

Similarly → Case 2 → "Dark to light" Edge.

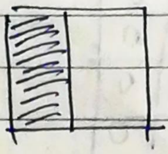
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

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

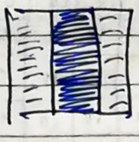
→

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



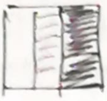
*

→



FILTERS

→ Vertical Edge Filter $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ →



→ Horizontal Edge Filter $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ →

OTHER filters

① SOBEL Filter → $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ → more weight to central row
→ claimed to be more robust.

② SCHARR filter $\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$ Note: Turn any filter by 90° it goes from V ⇔ H filter.

WHERE deep learning pops in → rather than explicitly providing or selecting this filter, we can rather let each filter element be a parameter (w_{ij})

Filter = $\begin{bmatrix} W_1 & W_2 & W_3 \\ W_4 & W_5 & W_6 \\ W_7 & W_8 & W_9 \end{bmatrix}$

And just like a NN learns weights & biases, let it learn these filter parameters by Deep learning.

This eliminates explicit handcoding of filters.

– Convolution operation still remains though.