Module 30 : Convolutional Neural Networks.

fully Connected Neural Network : Every neuron in previous layer is connected to every neuron on the layer ahead.

↓

interested in neural networks due to the universal approximation theorem, which states the strength of FNNs, as fn. approximators. These can be toned & optimized using backpropogation.

↓ (many parameters)

DNNs are prone to overfitting (high variance models), and gradients can vanish due to long chains. To overcome these issues, some tricks and methods were understood that included weight initialization, diff. optimisation methods, regularization, deff. activation fns.

*) More complex DNNs having fewer parameters (avoiding overfitting)

→ The convolution operation : An example scenario : A plane takes off from Chennai, and we have a measuring instrument, which measures its distance at regular intervals. To account for sensor noise, we take the weighted avg. of all positions of the plane to get a more

reliable reading.

$$s_t = \sum_{a=0}^{\infty} x_{t-a}\, w_{-a} = (x * w)_t$$

Now, rather than using all previous values to compute the reading, we use the previous 10 readings to get the current estimate, since the contribution of terms after that is comparable to 0, and doesn't have much affect on the result. ∴

w	0.01	0.01	0.02	0.02	0.04	0.4	0.5

x	1	1.1	1.2	1.4	1.7	1.8	1.9	2.1	2.2	2.4	2.5	2.7

convolution operation deals performing elementwise operation using a mask and an underlying structure Mostly in neural networks, that would be doing an elementwise multiplication and then summation.

The example above is convolution for 1 Dimensional data.

↓

Boils down to taking a given i/p and re-estimating it as a weighted avg of all the i/ps around it, i.e. certain values/weights are assigned to the neighbours of a pixel and the convolution is used to re-estimate its

value Mathematically :

$$S(i,j) = (I * K)_{i,j} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a,b}$$

$S(i,j)$: estimating a new value for the $(i,j)^{th}$ pixel.

I : input

K : Kernel

→ weighted sum of the neighbourhood around it.

input :

a	b	c	d
e	f	g	h
i	j	k	l

Kernel :

w	x
y	z

output :

$aw + bx + ey + fz$	$bw + cx + fy + gz$	$cw + dx + gy + hz$
$ew + fx + iy + jz$	$fw + gx + jy + kz$	$gw + hx + ky + lz$

*) Here we are considering only the neighbours ahead, we should also consider the previous pixels as well (previous rows and previous columns), i.e.

$$S_{i,j} = (I * K)_{i,j}$$

$$= \sum_{a=\lfloor -m/2 \rfloor}^{\lfloor m/2 \rfloor} \sum_{b=\lfloor -n/2 \rfloor}^{\lfloor n/2 \rfloor} I_{i-a, j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

Here the origin has been modified. Generally odd dimensional kernels are used.

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \rightarrow$$

averaging / smoothing filter. Blurs an image. demphasizes the value at the origin pixel.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \rightarrow$$

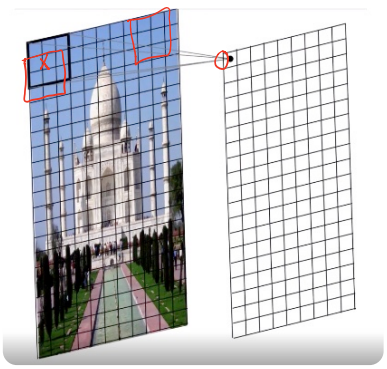
emphasizes the value at the origin by multiplying by 5 & subtracting neighbourhood information. Used for sharpening an image

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \rightarrow$$

used to find points of boundary i.e. there is a difference in the value at the origin & the neighbourhood.

↓

hence detects the edges in an image

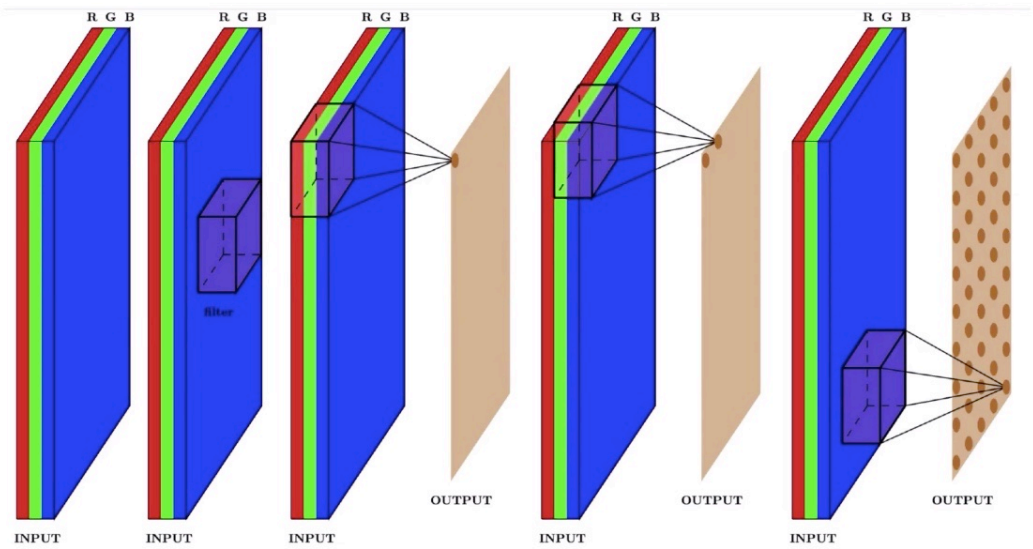


*) The kernel is first slid horizontally and then vertically

*) 3D input : a color image is a 3D input due to the presence of multiple channels, for example RGB, which are superimposed over each other .i.e. every pixel has 3 values associated to it, which can be considered the depth.

↓

convolution in 3D : using a 3D kernel, i.e. the convolution is applied across the different channels for a pixel as well. i.e. weighted average of its 3D neighbourhood. Here since the depth of the 3D kernel is always equal to that of the i/p, hence the movement of the kernel is only required in two directions and only one channel is output at all times for a given image.



Another way of looking at this is as a 2D operation

using 3D image and kernel. since we are sliding only vertically and horizontally

Now, we apply multiple filters across the image to get multiple outputs, and now these outputs can be combined into one single volume. The depth of this volume will be equal to the no. of filters applied on the image.

*) some terminology related to convolution operation

input width (W_I), input height (H_I) and input depth (D_I)

output width (W_o), output height (H_o), and output depth (D_o)

Spatial extent of a filter (F) → all kernels are assumed to be symmetric, hence this represents the size in all req. dimensions.

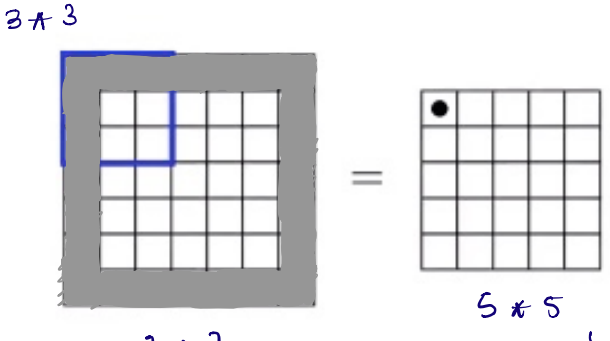
↓

(depth of the i/p = depth of the kernel)

No. of filters (K)

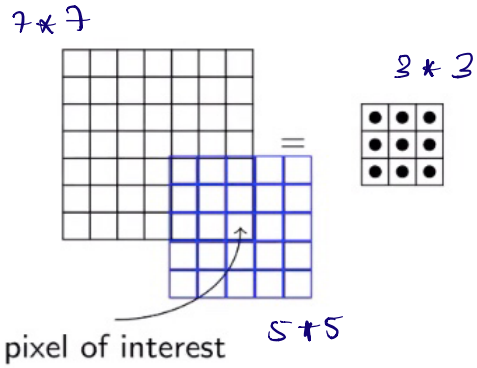
Padding (P) and Stride (S)

*) computing W_o, H_o, D_o.



↗ clear that the o/p is smaller than the i/p since the boundary pixels are not taken into the convolution operation.

↳ This is the result of applying 3*3 filter



→ The o/p gets smaller, since the kernel got bigger and the invalid area increases.

relationship W_o and W_I $\Rightarrow$ $W_o = W_I - \frac{(f-1)}{2} * 2$

↓ accounting for both sides

$= W_I - (f-1)$

$= W_I - f + 1$

similarly, $H_o = H_I - f + 1$

*) If we want the o/p to be of the same size as the i/p.

↓

we add an auxilliary boundary to the image with 0 values. Now instead of the earlier values, these 0.

↓

This process is called padding. Adding 1 row below & above, and 1 column on the left and right refers to a padding of 1.

↓

$\text{padding} = \left(\frac{f-1}{2}\right)$ → To get an o/p of the same size as the i/p.

*) The o/p width $= W_I - F + 2P + 1$

↓

accounting for the addition on both sides.

*) Stride : defines the interval at which the filter is applied

The updated formula for W_o, H_o

$$W_o = \frac{W_I - F + 2P}{S} + 1, \quad H_o = \frac{H_I - F + 2P}{S} + 1$$

Stride is used to avoid blowing up of the o/p.