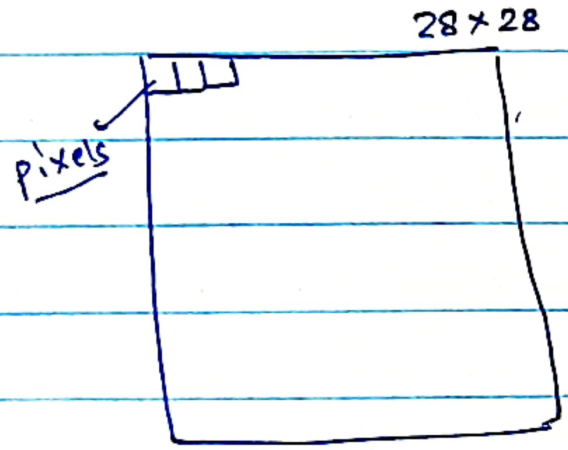
* Images - Pixels.

Why CNN?

* For a simple neural network we have to train a ~~lot~~ lot of parameters. Number of parameters are quite high.
  ↓
  Its very hard to train any network and hence we need to reduce the number of parameters.

Image



→ When flattening happens
↓
Converts this image into single 1D but Image was a 2D information. If we pass the 2D information in 1D.
↙
ANN doesn't learn those features.

A similar issue happens in NLP. for eg "I am going to market"
↙
we need to capture the content and not simply convert it to numbers.

① Patterns | Edges : when we zoom the images and are able to see some pattern and edges.

② Objects : details in an image, person, etc

③ Parts of objects : eyes nose ears of the person

④ Image : complete information that we process

⑤ Scene : multiple images → video] scene

Edges → Pattern → Parts of object → Object → Image → Scene

0	0	5	0	0
0	5	18	32	0
0	18	100	64	0
0	100	32	4	0
0	1	2	0	0

If I want to capture this information, i.e spatial information we cannot capture this if we are performing flattening operation.

Image processing by brain

$V_1, V_2, V_3, V_{3A}, V_4, V_5$] → these layers are a network

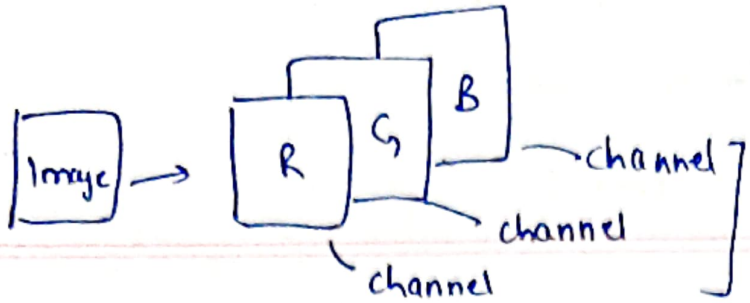
we see edges - pattern - part of objects → objects - image

Colors / ① RGB

② CMYK → This colour space is good for reading
└ Newspaper Printing

good for human reading

α - Transparency

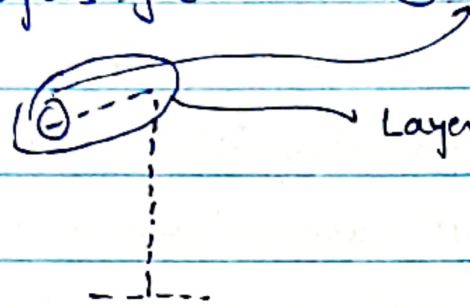


↓

Container or bucket which keep feature captured from different channels.

CNN
- features 95%
- Colours 5%

Early layers of CNN — ① feature Extraction - Similar features.



Kernels

① 3×3 : Most optimized Kernel till now

② 1×1 : point wise convolution - not used for feature extraction but rather for dimensionality reduction

③ 5×5

④ 7×7

⑤ 11×11

as we can see we are just writing odd numbers - why?

There is a symmetry issue. 8.

↳ This is a experimental idea

$7\times7 \xrightarrow{(3\times3)} 5\times5 \xrightarrow{(3\times3)} 3\times3 \xrightarrow{(3\times3)} 1\times1$

More your Kernel looks at the pixels, more its going to extract information from the image.

Most optimised filter : 3×3 - b/c it has least number of parameters
↳ NVIDEA has optimised the GPU to respond to 3×3 very fast

Kernel may not be a square matrix but also something like 3×1 or 1×3 matrix but experimentally this has been proved.

Always define the number of filters to the tune of 2^n. Similar thing goes for images 512×512, etc.

1) Number of filters = number of feature maps.

Parameter sharing :- same weights can be shared for two similar objects. for eg. parameters will be same for both the eyes.

Basic Components of CNN

1. Filters/ Kernels/ Feature Extractors.
2. Channels.
3. Feature Maps
4. Stride
5. Padding

Basic components of Convolution operations

6. Receptive field
7. Output Dimension Size
8. Max Pooling
9. Network design
10. Example of MNIST

Receptive field

Lets there be an Image (I) which is 7×7

$I_{7\times7} \xrightarrow{K_1(3\times3)} I_{5\times5} \xrightarrow{K_2(3\times3)} I_{3\times3}$

filters are odd to maintain symmetry around origin

1. local receptive fields → 7×7 K_1
2. global receptive fields → 5×5 K_2
   → K_2 5×7 + 7×7

In local receptive field Kernel can only see the preceeding image but not beyond that. Global receptive field could see beyond the preceeding image. Eg. K_2 in a local receptive field can only see 5×5 image but K_2 in global receptive field can see both 5×5 and 7×7 image

Designing a Network

$I_{size} \longrightarrow O_{size}$: $200 \times 200 \xrightarrow{K(3\times3)} (1\times1)$

close to 100 convolution layers.

There will be huge number of parameters higher training time and cumbersome

My entire image has been seen by the network

1. Try not to use many layers. Not more than 15
2. Generally layers b/w 5 - 15
3. Rare cases 15 - 30

but there is no fixed rule if image has hiper resolution then one needs more layers.

You can increase the size of kernel to reduce the image size quickly and have less layers.

Max Pooling Layer : It reduces the size quickly and reduce the number of layers. (3)

By default it reduce the image by 2

eg 50 x 50 —MaxPooling→ 25 x 25

Convention :- Do not use two maxpooling layer simultaneously so, to reduce the dimension of image quickly than Max Pooling is our best friend. Filters / Mask are used to save location of the highest feature.

Input Image ⟶ Output Image

① No stride No padding

② Only padding

③ Only stride

$$n_{out\,dim} = (n_{in} - f + 1) \quad \text{— (1)}$$

$n_{out\,dim}$ ↓ Output size

n_{in} ↳ Input size

f ↳ filter size

In deep learning we mostly use zero padding and tend to avoid the reflective padding ↳ discontinuity of Information

Two types of Padding → Valid — Normal, no padding (Image Reduced)

→ Same - Input = Output, No Redn.

Only padding, no stride or stride = 1

$$n_{in} - f + 2p + 1 = n_{out}$$

(4)

Case3: ~~only stride~~ padding + stride → easy computation
↳ jis skip pixels.

$$n_{out} = \frac{n_{in} - f + 2p}{s} + 1$$

Hyperparameter:

- Kernel size - 3×3, 5×5, 7×7
- Stride - 2
- Padding - Same, valid
- Dense layer / Number of Neurons → 64, 128
- Activation function - ReLU
- Loss function → Categorical
- Optimiser → Adam. In CV (SGD)
- Image Resolution
- Number of Convolution Layers.
  ↳ Image Reduction
- Epochs.