CNN

1. Image
2. Filter | Kernel
3. Convolution + Relu
4. Pooling
5. Flatten (2D - 1D)
6. FC

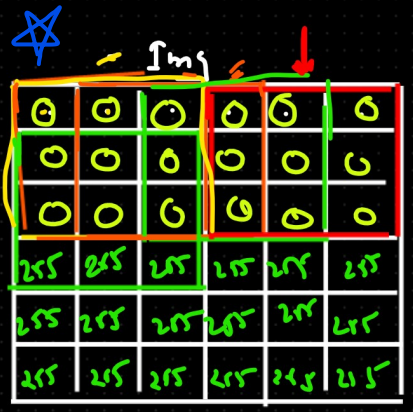
Different Stages/Operations of CNN:
- Image = Collection of pixels
- Filter/Kernel
- Convolution(Matrix multiplication b/w image and filter) + Relu(Why we add Relu will be discussed)
- Pooling
- Flatten(2D to 1D)
- Fully connected Neural Network

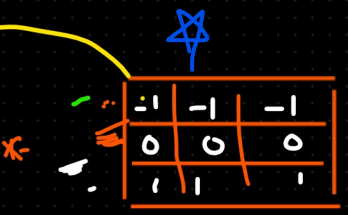
Convolution means feature(pixels/kernels) extraction. For Ex: Given the image of person, we attempt to extract the features or pixels representing the hair followed by, classifying the type of the hair (Spikey, straight, curly, normal etc)

MATHEMATICAL IMPLEMENTATION OF CNN ON GRAY IMAGE(Black=0 and White=255)

256 = [0-255]

Image → Gray Scale, Color Img (3 - Channel) ⇒ R, G, B

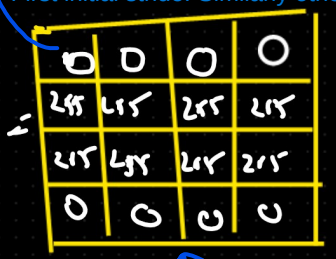




Filter | Feature | Kernel

(0×-1) + 0×(-1) + 0×(-1) + 0×0 + 0*0+0*0+0*0+1*0+1*0+1*0 = 0

First initial stride. Similarly others



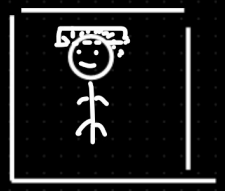
Here Feature Map is created by moving Stride of 1 along horizontal and vertical traversal of the image

Feature map

- Multiplication of image and filter is normal multiplication of corresponding pixels and filters and not matrix multiplication.
- Filter is having some window size(3X3 here). This window traverses through the whole image and perform multiplication in the respective window. During traversal filter performs jump in order to capture other pixel in next iteration which is known as STRIDE.
- Filters are square matrix(we may experiment with non square matrix) with preferably odd size?
Like: 1X1, 3X3, 5X5, 7X7 etc
Based on how much information or pixels we want to collect in each stride we are deciding the size of the filter

Convolution => (1) extract the feature from img.

We can do as many convolution(image and filter multiplication) as we want since this is hyperparameter

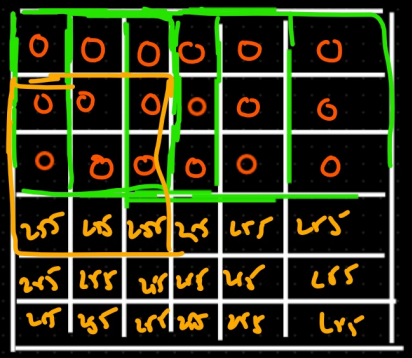


Feature extraction

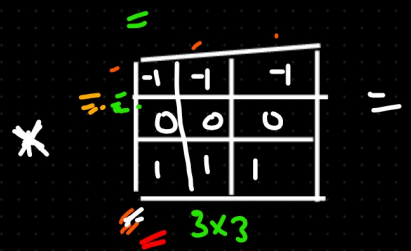
This is the hard coded filter found by Scientists that can perform the Horizontal Edge Detection

- Odd.
  - Alexnet
  - VGG16, 19
  - Resnet
  - Inception

So, Yann Lenn in his LENET architecture has created these filters randomly(not hard coded) and then applied back propagation to find the optimal filter weights in order to perform detection of the respective objective during training. This means that in LENET FILTERS become the learnable parameter. This why this become famous and treated as first official CNN architecture. Whereas, CNN architectures before LENET like Neocognitron uses hard coded filters and there was no back propagation to embed the self learning of these filters.



6x6



3x3

horizontal edge Detection

feature map

4x4

+ 0x1

LENET

Kernel | filter => Dimension 1x1, 2x2, 3x3, 4x4, 5x5, 6x6, 7x7

Initially => (1980-1990)

Sensor => Image

Hardcoded

3x3

machine

algorithm

Feature

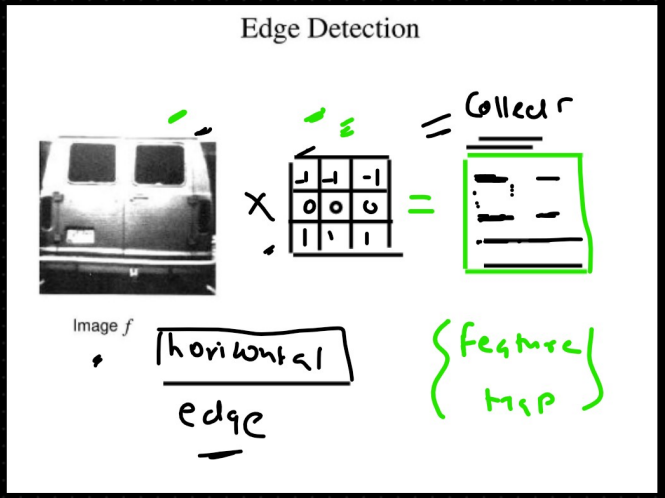
3x3

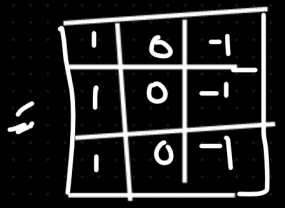
5x5

7x7

9x9

11x11



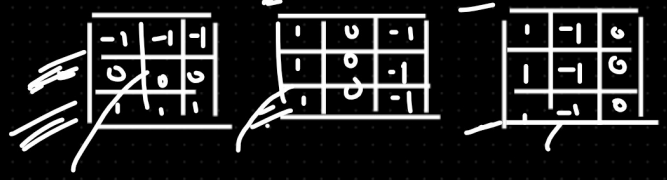


Hard Coded Filter for Vertical Edge Detection already discovered by scientists through extensive research

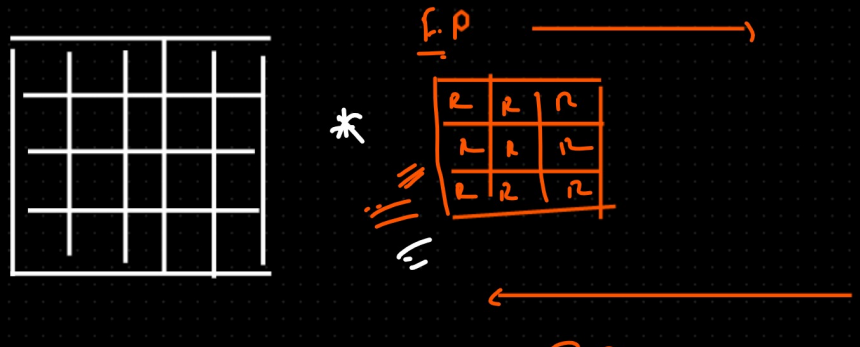
= Vertical edge detection

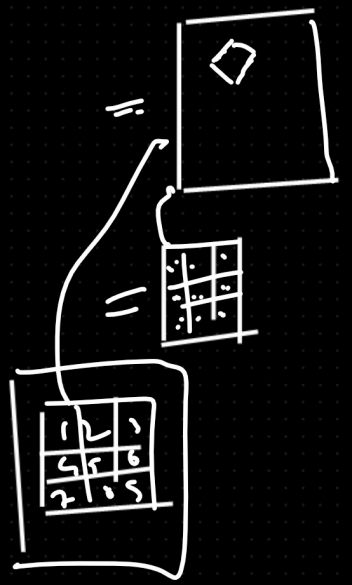
Viola-Jones Object Detection Framework is used for determining the hard coded filters for object detection

Filter ⇒ For extracting the feature from the image



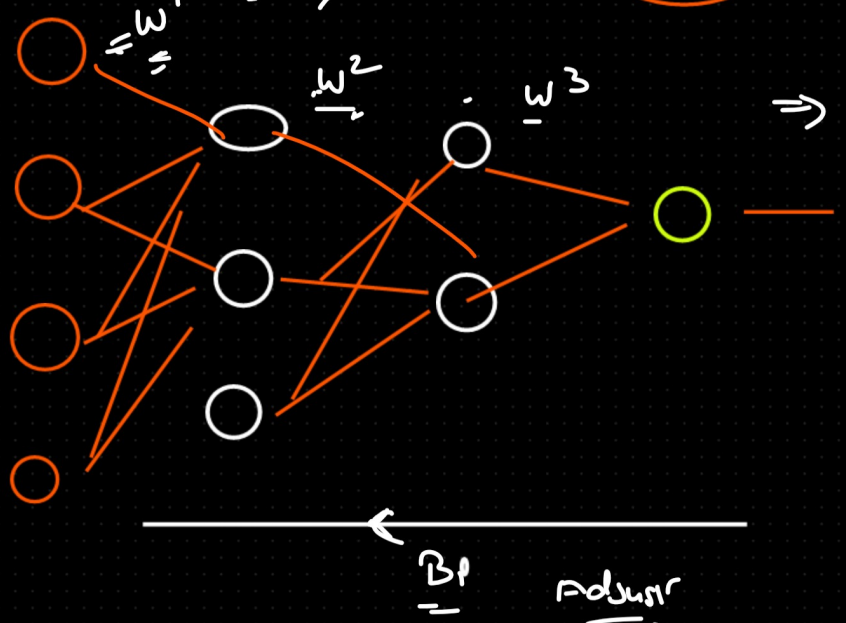
CNN = LE-NET ⇒ first Architecture in CNN





Learning

1990 :: Le-Net → CNN ⇒ Learnable



⇒ Random value

NN ⇒ it is learning by itself

it is learning based on data

BP:

$\frac{\partial L}{\partial w}$

General formula for determining the shape of convoluted image = (n-m+1)X(n-m+1) where n and m represents the dimension of Image and Filters respectively. Please note that here we have taken the consideration that all 3(image, filter and convoluted image) is in square nature (rows=columns)

Image Size		Filter Size		Convoluted Image/Feature Map Size
6×6	*	3×3	=	4×4
8×8	*	3×3	=	6×6
28×28	*	3×3	=	26×26

$n \times n * m \times m \Rightarrow (n-m+1) \times (n-m+1)$

$6 \times 6 * 3 \times 3 = (6-3+1) \times (6-3+1)$

$= (3+1) \times (3+1)$

$= 4 \times 4$

$$img\_size - filter + 1$$

$64 \times 64 * 5 \times 5 = 60 \times 60$

$64 \times 64 = 60 \times 60$

$(64 - 5 + 1)$

$(59 + 1) = 60$

Padding

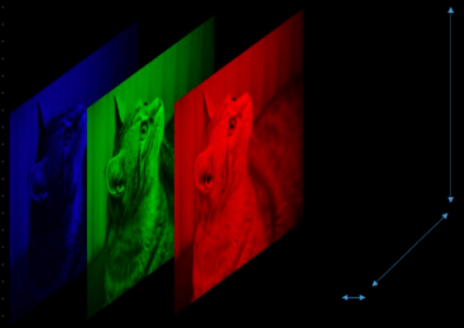
Stride x Padding

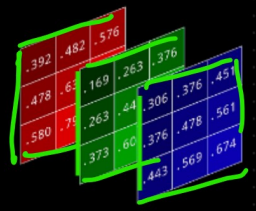
Every feature

low level feature

MATHEMATICAL IMPLEMENTATION OF CNN ON COLORED IMAGE(3 Channels RGB):

Please note that for Gray scale image we will have image, filter and convoluted image in 2 dimensions. Whereas, for Colored Image we will be having Image and Filter in 3 dimensions and convoluted image in 2 dimensions





Color → [R, G, B]

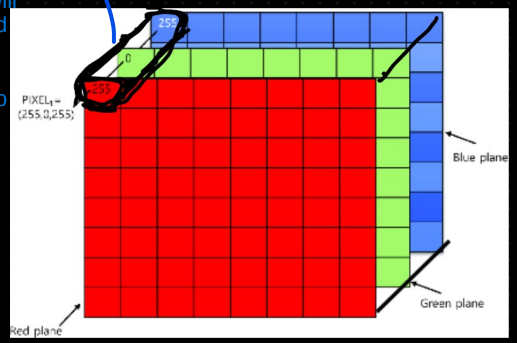
In convoluted image if we are getting pixel value more than 255 (image and filter multiplication) then round it to 255 since we have pixel value up to maximum 255 (Range is [0.255]. Same happens in the lower bound as well)

Filter size is also the hyperparameter

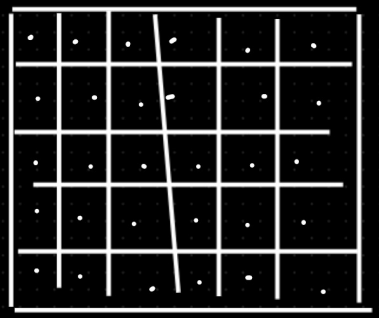
In convolution we are loosing the data. So in case we do not want to loose the original size of the data then we can use PADDING

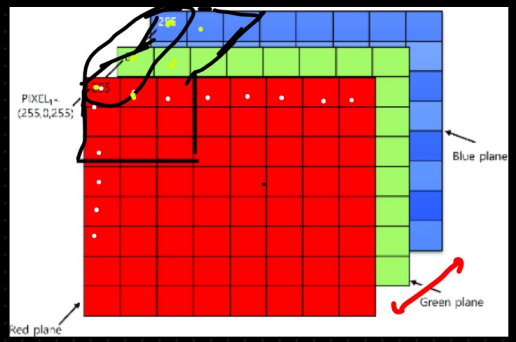
Color (R, G, B)

Third dimension (depth RGB) will always be fixed in each stride that's why we are having Two dimensions in the convoluted image for colored image



Gray Scale image





$(8 \times 8 \times 3)$

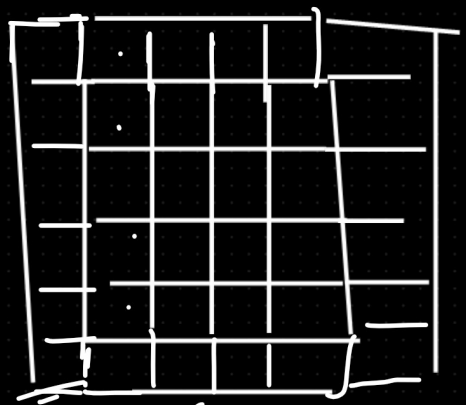
Color

$*$

$(3 \times 3 \times 3)$

$(n - m + 1)$

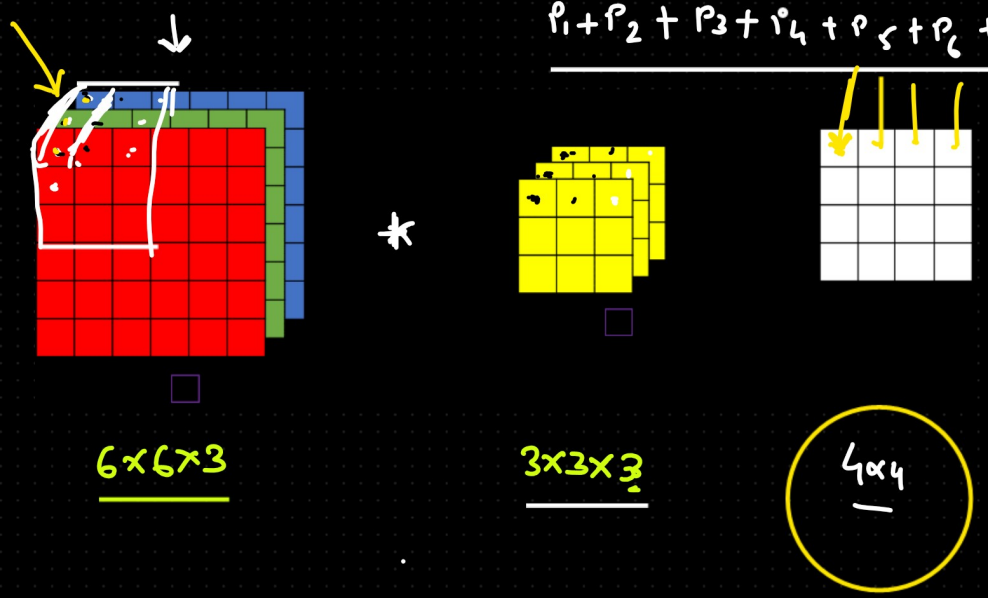
$=$



6×6

F. M

Single Channel



Here also for each stride multiply each corresponding image pixel with corresponding filter weight to get respective pixel for convoluted image

Practical

We can apply as much as filter as we want(where each feature is performing different detection or extraction) on top of image as number of filter applied is a hyperparameter and at the end we are basically stacking up all the convoluted images formed with respective filters which is called the Feature Map.

image x filter

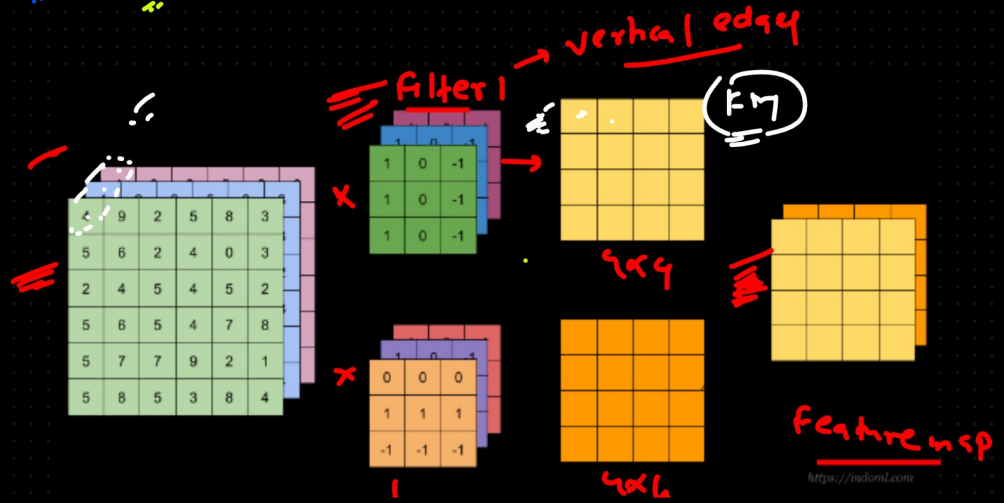


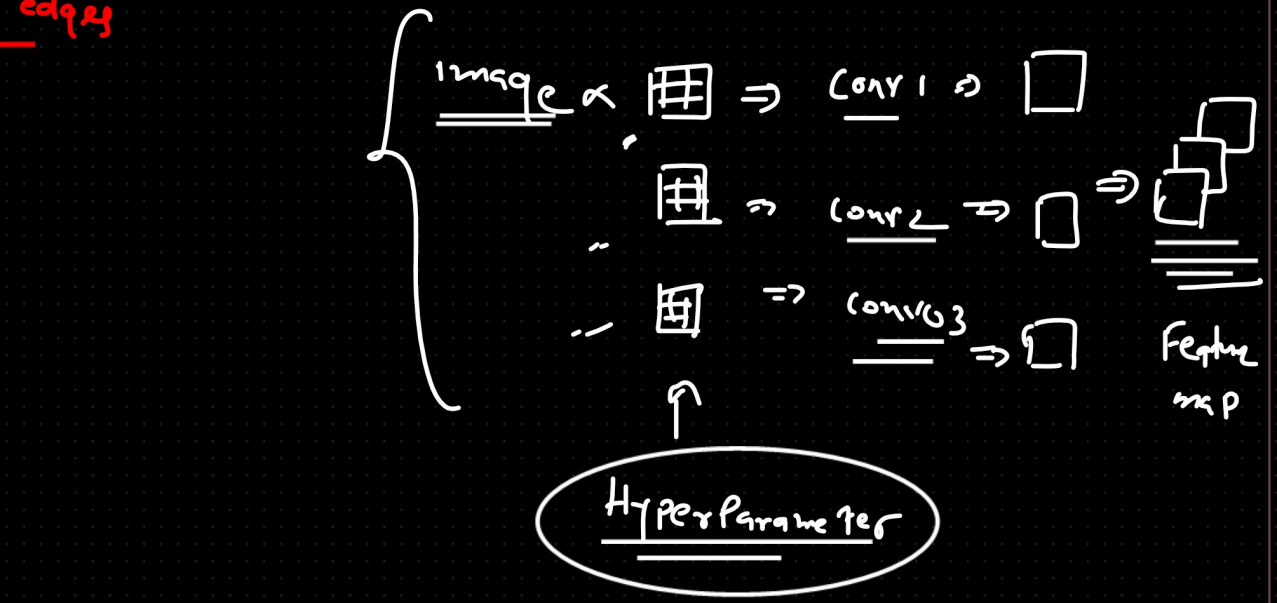
Image x filters

↓

Extract information from img

Image ⇒ side edge
horizontal edge
circle

↘ filter 2 Horizontal edges



Padding | Stride | Pooling

Convert a original image of 5X5 image into 7X7.
If we fill the extended pixels using Padding with 0 then it is called Zero padding
Motive if to prevent the data loss by retaining the original image size in convoluted image formed by multiplying respective filter with the padded image

Padding

image x filter ⇒ Convoimg

6x6 3x3 ⇒ 4x4

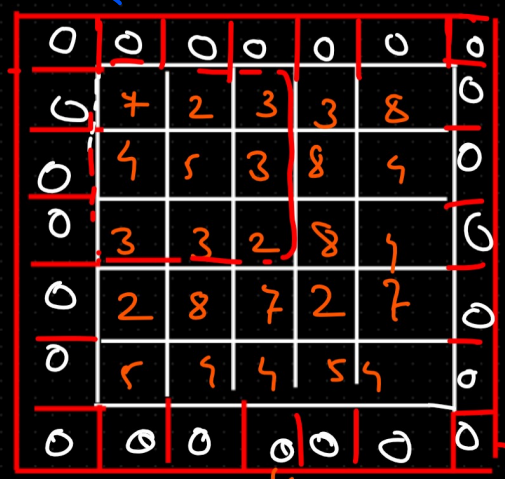
n-p+1 ⇒ 6-3+1

3+1 = 4

Here we have done zero padding on 5X5 image ad converted it into 7X7 image

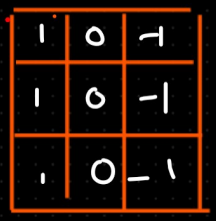
↑

Padding

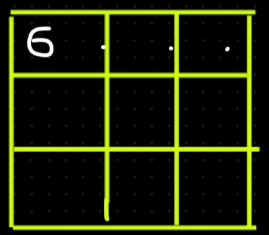


7X7

*



=



3X3

5X5 ?

7x1 + 4x1 + 3x1 + 2x0 + 5x0 +
3x0 + 3x-1 + 3x-1 + 2x-1 = 6

Convoluted image will be 3X3 if filter multiplied on image without padding but if multiplied with padded image then will get convoluted image of shape 5X5 which is similar to the original shape of the image.

$n \times n$

$= 5 \times 5$

$f \times f$

3×3

$= n-f+1$

$5-3+1 = 3 \times 3$

$5 \times 5 \Rightarrow 5 \times 5$?

$n-f+1 = 5$

$n-f+1$

$7-3+1$

$4+1 = 5$

$n = 5-1+f$

$= 5-1+3$

$n = 7$

$n = 7 \Rightarrow 5$

Generalizing this formula further using padding

$n-f+1 =$

$5-3+1 = 3 \Rightarrow 3 \times 3$

5×5

$(n+2p-f+1)$

$P = 0$

$n+2 \times 0-f+1$

$(n-f+1)$

$P = 1$

$n+2 \times 1-f+1$

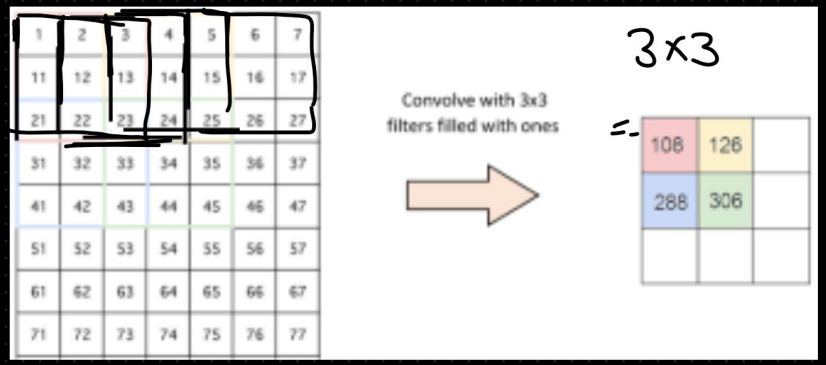
$5+2(1)-3+1$

$= 5+2-3+1 = 5$

Here P represents the number of pixels padded on each corner of the original image. If P=0 means no padding hence dimension of convoluted image will be n-f+1. Whereas, if P=1 means we will pad the original image with 1 pixel on all 4 corners. In such a case dimension of convoluted image will be given as n+2P-f+1

Jump from one pixel to another pixel

Stride ⇒ Jump is called Stride



1 Jump ⇒ Stride

While we are iterating over the img

Stride = 1

- Stride =2 means we are making jump of 2 pixel when making the each next jump
- If we increase the stride then loss of information from original image also increases. So, below are the cases when someone should consider to increase the stride:
  i) When High level feature
  ii) When computation cost is getting high

$n-f+1 \Rightarrow \left(\frac{n-f+1}{s}\right)$ $s=1$ $\Rightarrow \left(\frac{n-f+1}{1}\right)$ $n-f+1$

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

6x6 3x3 $\Rightarrow \left(\frac{6-3+1}{1}\right) \Rightarrow 6-3+1 = 4$

4x4

Same as above this is the generalized formula for the dimension convoluted image including both Stride and Padding

Ques: When to use padding?
Ans: When in convoluted image data loss as compared to the dimensions of the original image is very huge

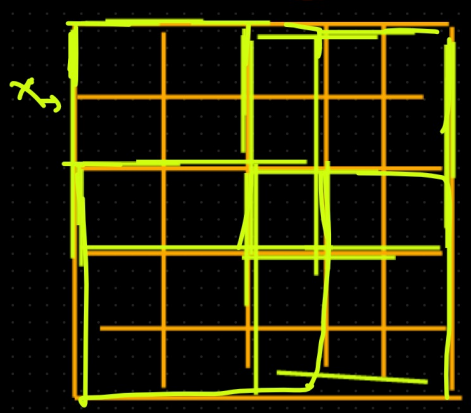
$6 \times 6 \quad 3 \times 3 \Rightarrow \frac{6 + 2p - 3}{2} + 1$

$\{p = 0, s = 2\} \Rightarrow \frac{6 + (-3)}{2} + 1 \Rightarrow \frac{6-3}{2} + 1 = \frac{3}{2} + 1$

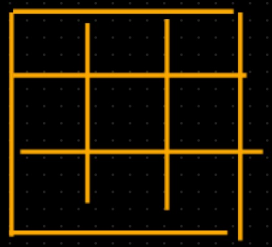
$= 1.5 + 1 = 2.5 \rightarrow 2 \times 2$

$6 \times 6 \rightarrow 2 \times 2$

$\Rightarrow s = 2$



$*$



$\Rightarrow 2 \times 2$

$5 \times 5 \quad 3 \times 3 \Rightarrow$

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

1. High level feature

2. Computation

64 x 64 x 3

↓

low level

$$\left[\frac{n+2p-f}{s}+1\right] \Rightarrow \text{Resultant image}$$

n = Size of image

p = Padding

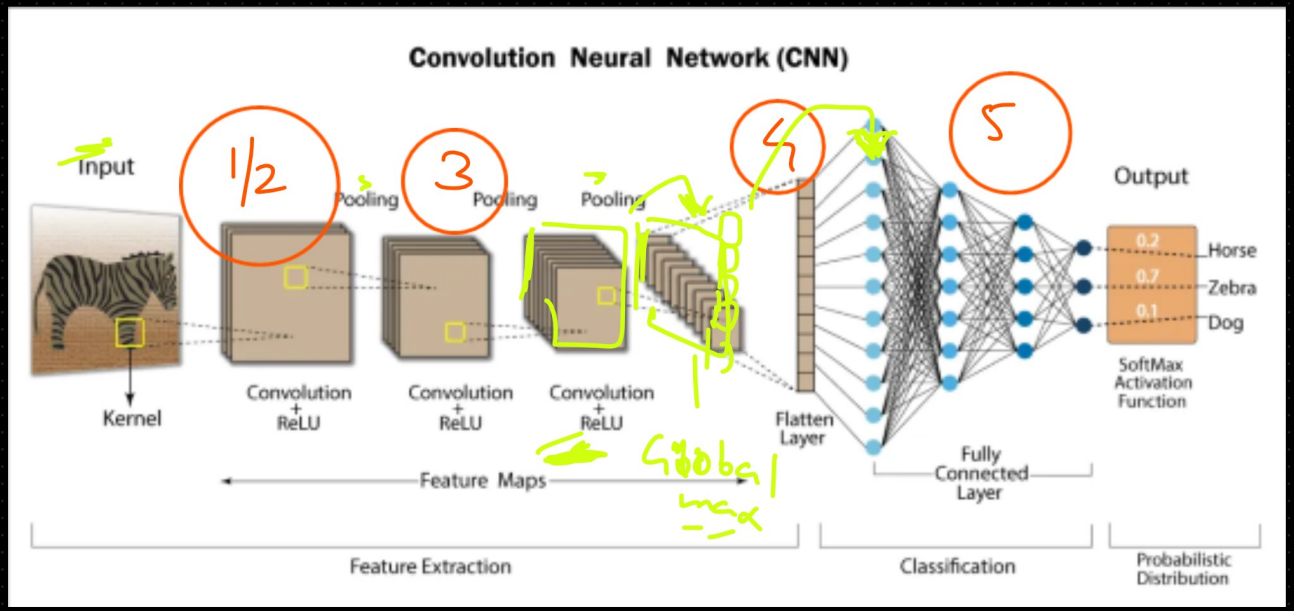
f = filter size

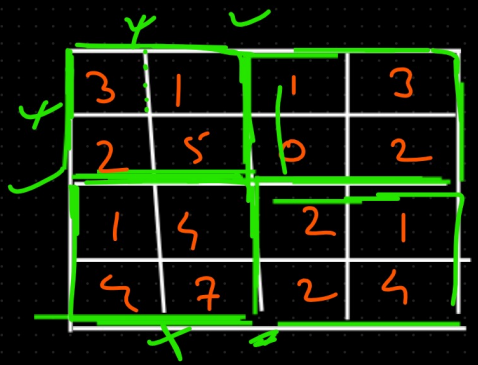
s = stride value

s = 2, 3, 4, ...

Ques: In CNN why we need Relu?
Ans: To remove the -ve value or in other words to capture the non-linear relationship as well.

1. Convolution
2. Relu (to remove -ve value)
3. Pooling
4. Flatten
5. FC





Feature map

Convoluted image

⇒ Pooling (i want to Pool the imp feature)

1. max Pooling
2. min Pooling
3. averge Pooling
4. Global Pooling

Size = 2x2
Stride = 2
type = max

For performing pooling also we take pooling window and stride

image * filter

5	3
7	4

2 * 2

specific feature

intensity of image

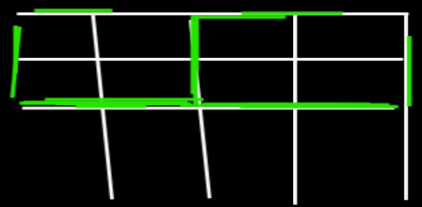
This is the Feature Extracted for the above Feature map using Max Pooling where in each widow of respective pool size we are picking up the one having larger value. Similarly, in the case of Min. Pooling we will pick up the smallest value in each window whereas, in the case of Average Pooling we will simply take the average of all the pixels in each window

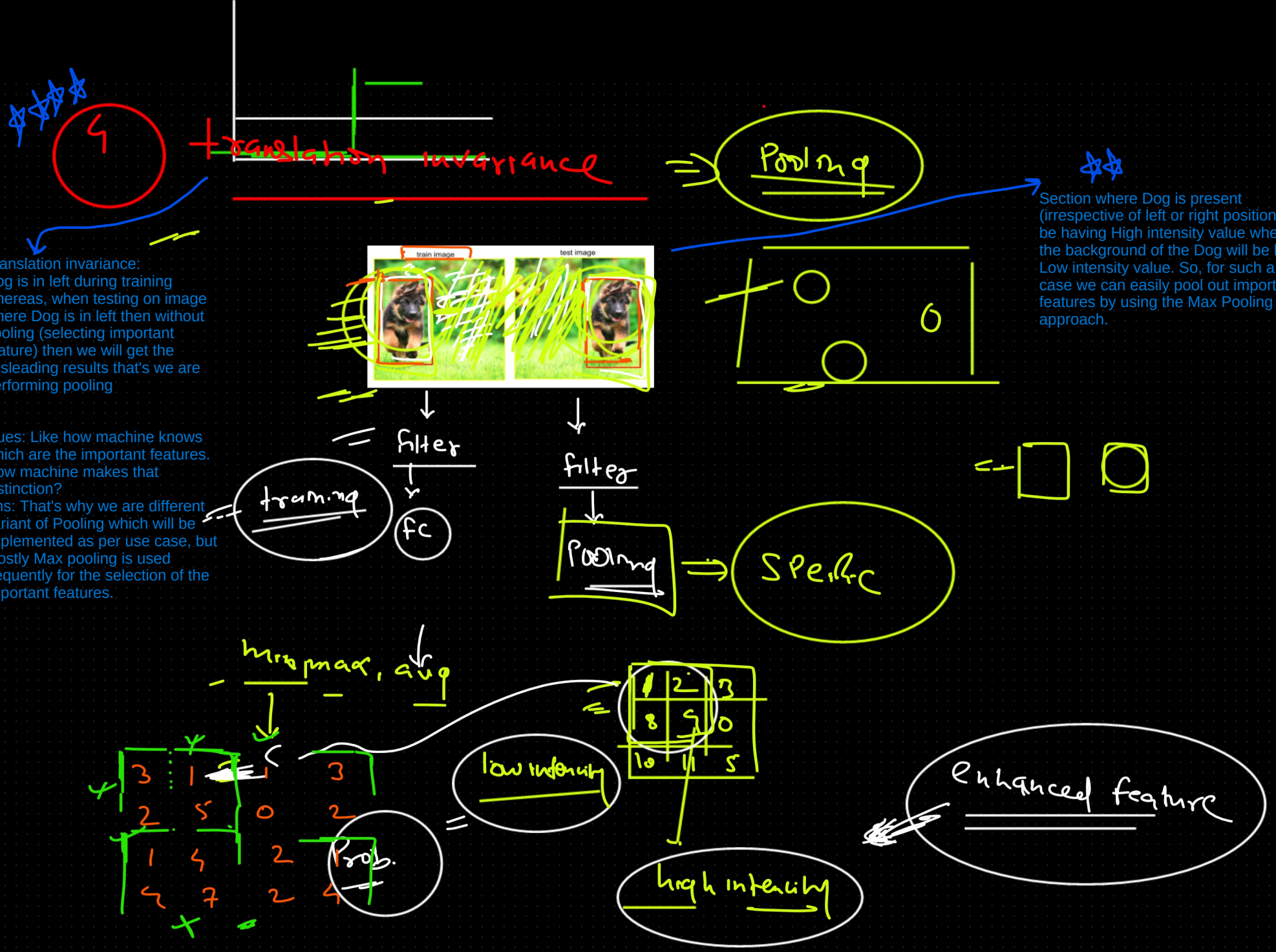
huge

$228 \times 228 \times 3 \; * \; \frac{3 \times 3 \times 3}{(100)} = 226 \times 226 \times 100 \Rightarrow (2 \times 2) \rightarrow 2 \Rightarrow 113 \times 113 \times 100$

Below are the reasons behind performing Pooling:

1. reduce the size
2. computational cost
3. focus on the specific feature

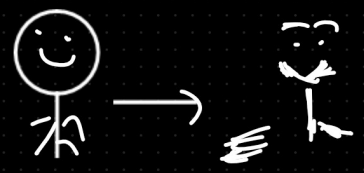


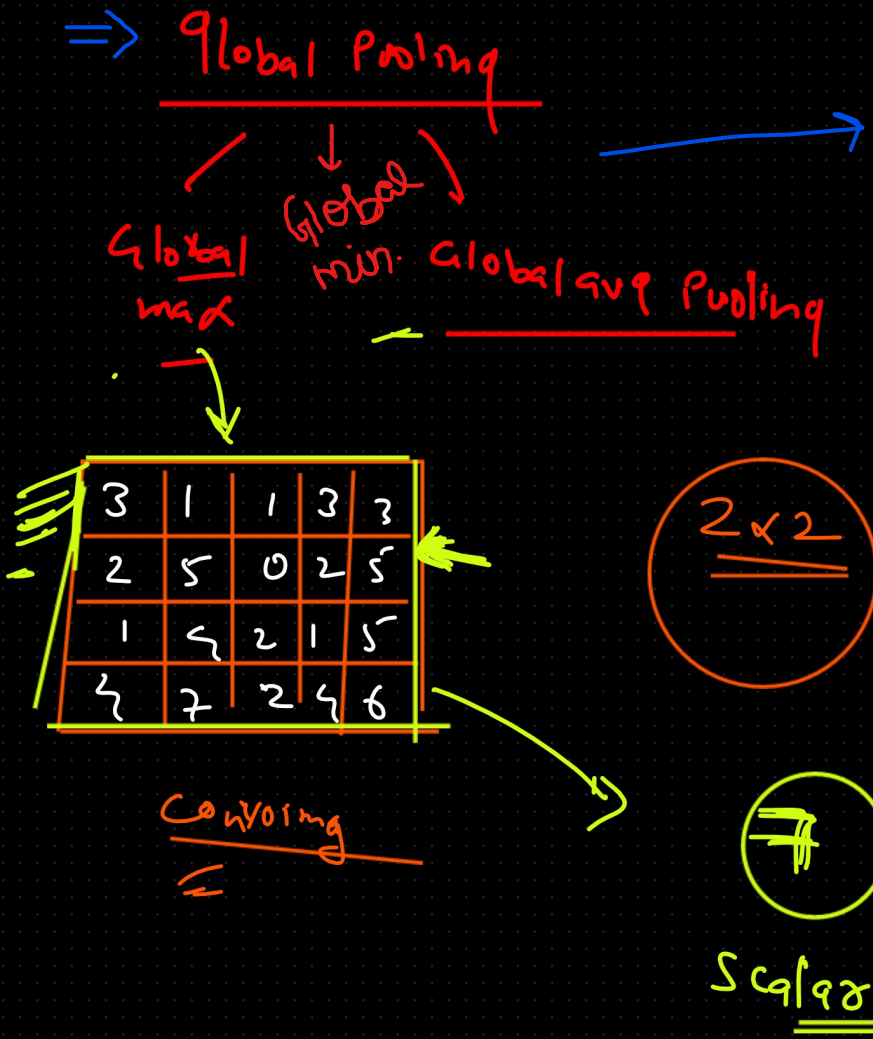


Translation invariance:
Dog is in left during training whereas, when testing on image where Dog is in left then without pooling (selecting important feature) then we will get the misleading results that's we are performing pooling

Ques: Like how machine knows which are the important features. How machine makes that distinction?
Ans: That's why we are different variant of Pooling which will be implemented as per use case, but mostly Max pooling is used frequently for the selection of the important features.

Section where Dog is present (irrespective of left or right position), will be having High intensity value whereas, the background of the Dog will be having Low intensity value. So, for such a use case we can easily pool out important features by using the Max Pooling approach.

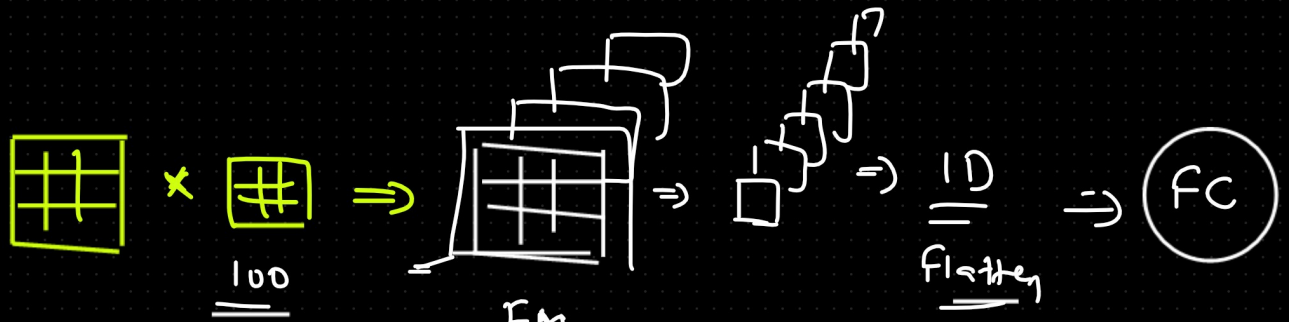




Global max we are converting the whole into single scalar which is maximum of the entire feature map

Whereas, in Global avg we are taking the average of entire feature map and form a single scalar

Similarly, Global min .. so on



Global max pooling

low intentence

↓

bmm