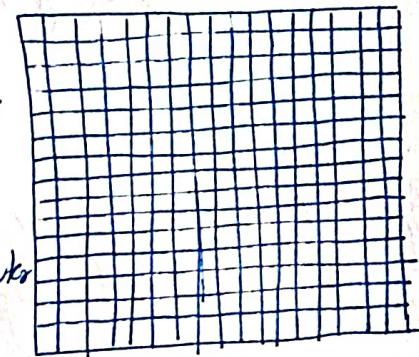


- 1) What are Images & pixels
- 2) Inspiration of Convolution neural networks
- 3) How Edge detection works
- 4) What is convolution
- 5) padding, strides, pooling
- 6) conv. of RGB (color) Images
- 7) How to build CNN Network for Image classification?

1) Images & pixels

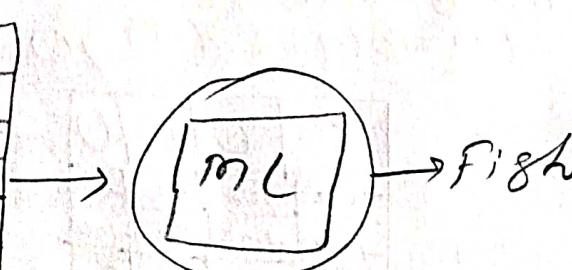
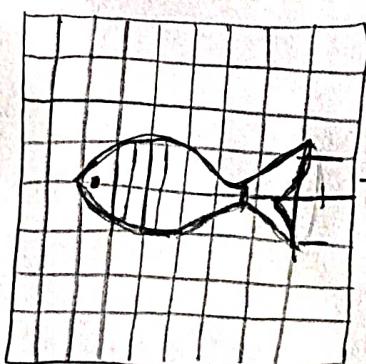
In computer everything is numbers
 Computer can't understand image as it is, ~~we~~ we have to convert this image into numbers for computer to understand. Here pixels come into this picture to help for converting picture to numbers.



Size of Image.

Pixels - Here, representation of image, a smallest unit of picture is called pixels. Here, every box is a pixel. For computer every box will have a value that is called a pixel value. This value determine color of a particular block.

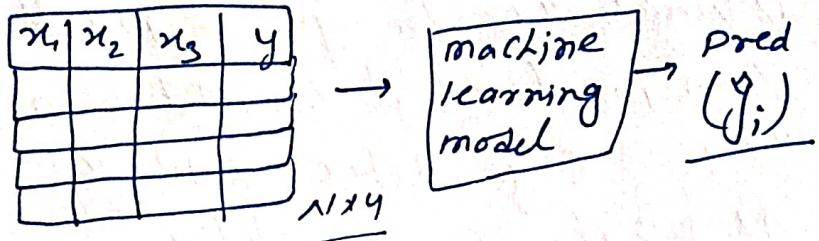
If a pixel no's are less then there is a chances of multiple color will come into the one box/pixel. So if we increase the no. of pixels that will represent one color in one unit.



CNN

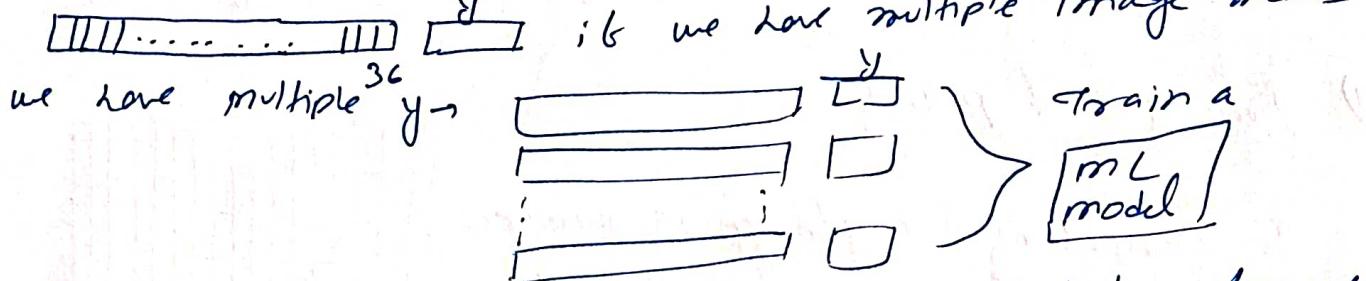
Traditional ML :-

$$D(x_i, y_i) \quad | \quad x_i \in R^3 \\ y_i \in (0, 1)$$



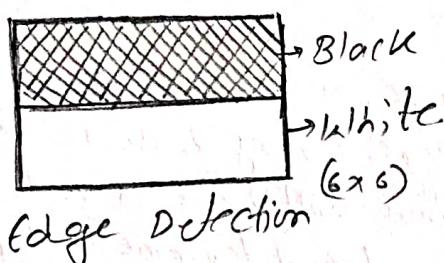
What if my current input will image not a tabular data
one approach could be :-

1) Convert (6×6) image into 36 length vector. bcoz $6 \times 6 = 36$



will it work? Technically it will work but performance you will get that is very poor.

for improving the performance CNN comes to picture.



0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

6×6

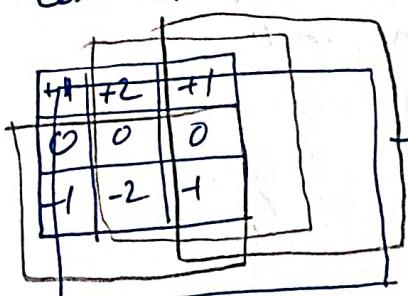
In computer pixel values varies b/w 0 to 255. this is a standard value. Black \rightarrow 0, White \rightarrow 255

convolution standard matrix \rightarrow

+1	+2	+1
0	0	0
-1	-2	-1

3×3

; sign of convolution ($*$)



$$0 \times (+1) + 0 \times (+2) + 0 \times (+1) + (0 \times 0) + 0 \times (0) + 0 \times (0) + 0 \times (-1) + (0 \times -2) + 0 \times (-1) \Rightarrow 0$$

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

0	0	0	0
-1020	1020	1020	-1020
1020	-1020	-1020	-1020
0	0	0	0

we have range only 0 to 255
but g am getting -1020 that will
be a issue. so scientist remove
minimum value with 0 and max
value with 255

$$\text{Here, min} \rightarrow -1020 \rightarrow 0 \\ \text{max} \rightarrow 0 \rightarrow 255$$

255	255	255	255
0	0	0	0
0	0	0	0
255	255	255	255

4x4



Sobel horizontal Kernel \rightarrow
$$\begin{array}{|c|c|c|} \hline +1 & +2 & +1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$
 \rightarrow Detect horizontal Edges

Vertical Kernel \rightarrow
$$\begin{array}{|c|c|c|} \hline +1 & 0 & -1 \\ \hline +2 & 0 & -2 \\ \hline +1 & 0 & -1 \\ \hline \end{array}$$
 For detecting vertical edges.

is there any other Kernel which can detect slant lines
depth information, a lot of other stuff \rightarrow yes; we have lot of
Kernels to understand the image. There are diff-diff Kernel with diff-diff
shapes. we are using (3×3) only b'coz of industry standards.

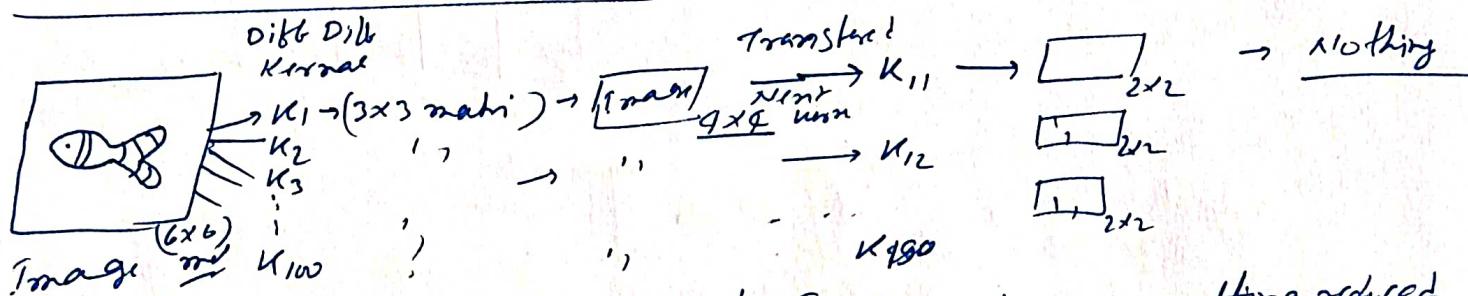
\rightarrow By using a operation called convolution by using a
concept called kernel, we will be able to detect edges,
But edges are not only thing needed to understand an image. we need to understand the color. we need to
understand the depth, we need to understand lot of aspect of a given image.

take one example

Visual cortex layer in human

v_1 ← let's assume we have an image and our brain see
 v_2 this image using layer v_1 first this will detect edge and
 v_3 whatever output it will detect. it will pass to the layer
 v_4 v_2 and it will detect depth (may be) next this will pass
 \vdots to the v_3 that will detect color. etc at the end passed
 thru multiple layer it can able to understand image

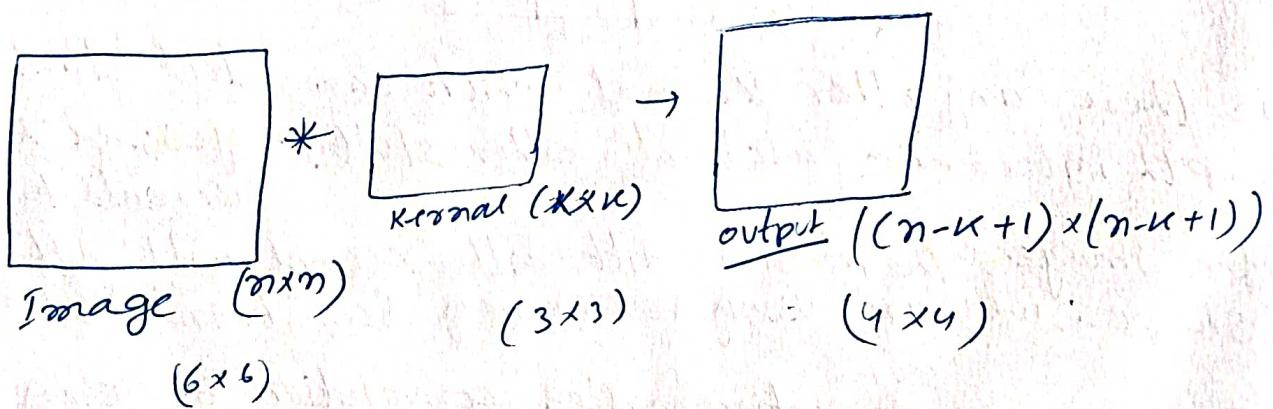
let's take one more example

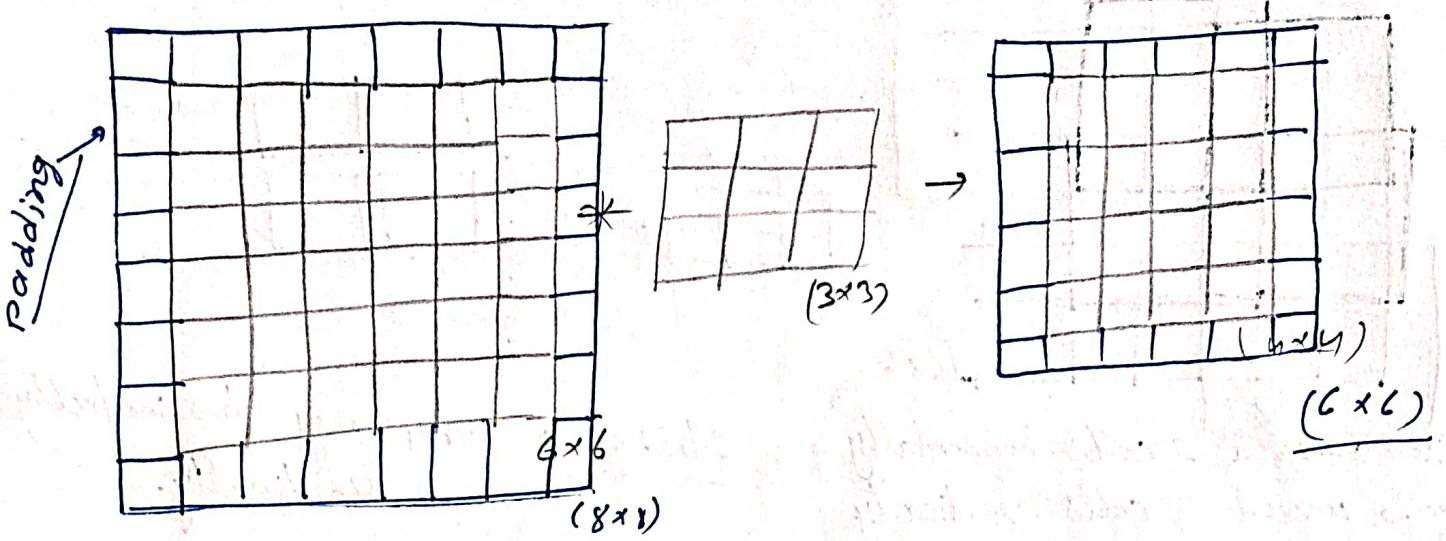


Q How many no. of kernel
Don't know, this is a
hyperparameter tell me it.

Timage shape is getting reduced
every time after using kernel.
so at this point, we don't have any
info. so here new concepts come
padding to resolve the issue.

So, given a image we pass it to the first layer of convolution
the output pass next layer of convolution, and this will learn some-
thing new, and we will pass it sequentially but we noticed a
problem that the image will get disappeared b'coz of reducing
in the shapes





The prob. is what will be the value in the edges in input b'coz we created edges artificially.

if i fill the zero's or one's into the extra edge there is the chances of may be inside value is diff. so it we create diff edges that could be wrong. so ~~we~~ one more approach could be fill the nearby value which will not differentiate much in that image. so we are just expanding the image and not creating any artificial edges.

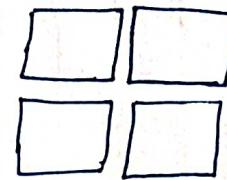
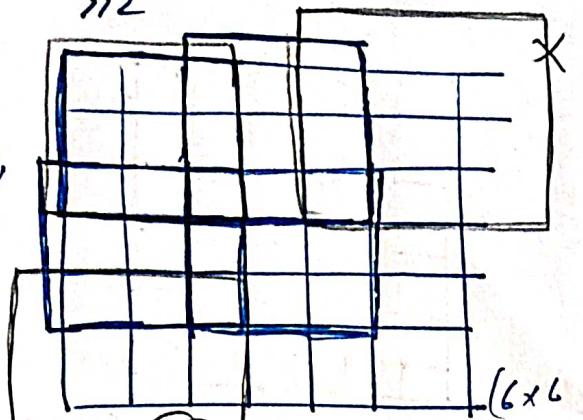
→ I have a 6×6 image, 3×3 Kernel, and I am getting image of 4×4 . But my requirement is getting a image 6×6 , so I don't want to reduce the input size, so if I don't want to reduce my image input size, and I can't change my Kernel b'coz it comes from research, so what I can do, expand the input size of image. and every given pixel value, we can either fill the nearby values or we can just fill zeroes, based on requirement now my output will be the same as input image, that concept is called padding.

Padding $\rightarrow (1, 1)$ means 1 Row up & 1 Row down and 1 Row left and 1 column right.

$$\begin{array}{l}
 P = 1 \\
 K = 3 \\
 n = 6
 \end{array}
 \quad
 \begin{array}{c}
 \text{Input: } 6 \times 6 \\
 \text{Kernel: } 3 \times 3
 \end{array}
 \rightarrow
 \begin{array}{c}
 \text{Output: } (n-K+2P+1) \times (n-K+2P+1) \\
 (6-3+2(1)+1) \times (6-3+2(1)+1) \rightarrow (6, 6)
 \end{array}
 \quad (5)$$

Strides :-

→ 2



→ 2 → moved 2 cells horizontally
↓ → moved 2 cells vertically.

strides → jumping horizontally and vertically.

→ Can't jump b'coz we don't have data flow.
so we can't do convolution. if we are jumping horizontally 2 blocks & vertically 2 blocks we get an image (2x2)

Here,

$$n = 6, s = 2, K = 3 \text{, formula: } \left(\frac{n-K}{s} + 1 \right) \times \left(\frac{n-K}{s} + 1 \right)$$

Pooling :-

O/P Shape

n = 6 (i) Only Kernel $\rightarrow (n-K+1) (n-K+1)$

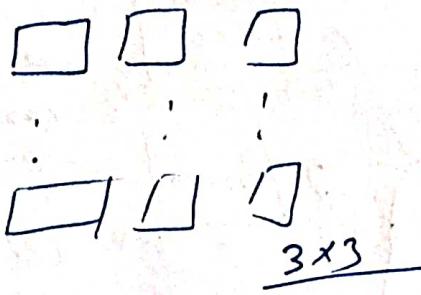
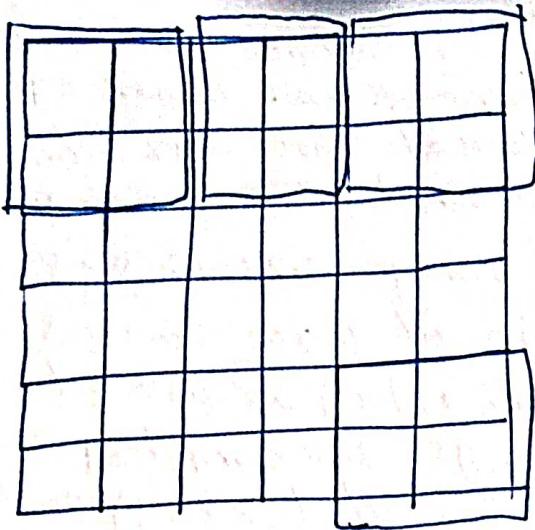
K = 3 (ii) padding + Kernel $\rightarrow (n-K+2P+1) (n-K+2P+1)$

S = (2,2) P = (1,1) (iii) Strides + Kernel $\rightarrow \left(\frac{n-K}{S} + 1 \right) \left(\frac{n-K}{S} + 1 \right)$

(iv) padding + strides
Kernel $\rightarrow \left(\frac{n-K+2P}{S} + 1 \right) \left(\frac{n-K+2P}{S} + 1 \right)$

Pooling :-

Let's we have an image, and one pooling matrix with a shape of 2×2 , it could be $3 \times 3, 4 \times 4$, etc. In the pooling matrix we don't have any value, it's just an empty matrix. we will add pooling matrix over the top of image pixels. all pixels will have some value, so then we will take max value from the pooling matrix shape. after that shift pooling matrix to next and take max value again. This will repeat to all pixels.



while we are doing pooling we are doing half of the image. if we are using 2×2 pooling.
why we are doing this?

→ aisa to sakte hai ki uss jagah ke sabhi cell me same color ho jo ki same value ko represent kar raha ho, to hamne eski jarurat nahi hai jo ki same information ko multiple time carry kare, to hum aisa karte hai ki uss pooling matrix ke shape ne size me jaake max^m value ko rakh lete hain. esse kya hogा ki hum log encapsulate kar raho hain max^m value/info ~~info~~ information from the particular matrix.

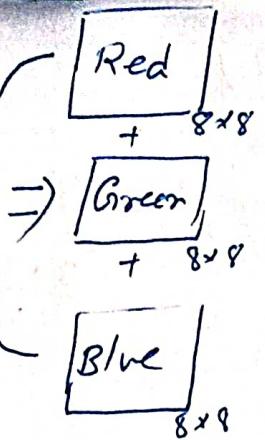
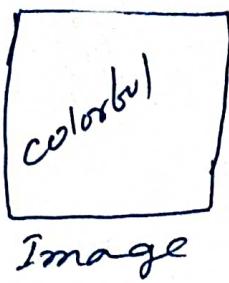
why only take max^m value? → B'coz humlog max^m information ko retain karne chahte hain from the given image.

instead of fetching max^m value we can fetch the min^m and avg as well, based upon situation,

max pooling

Avg pooling

Agar hamare paas 500×500 ka image hain aur hum 3×3 ka convolution matrix use karte hain to hamne bahut saare calculation karne parenge. Jisne time bhi bahut lagega. to esiliye hm pooling ka use karte hain, kyuki agar multiple box me same information hai to sath me se ek ko lekar aage badhte hain (esse calculation bhi jaldi hoga, time bhi kam lagega) aur training time bhi reduce ho jayega.) eske baad hamne 250×250 ka matrix mila ab eske upar 3×3 ka Kernel use karenge. ↗



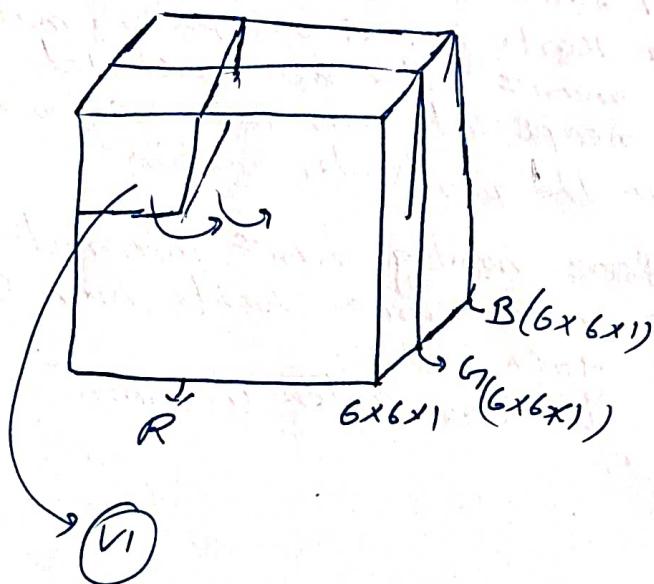
In computer science we have a colorful image and human eye can only look at three colors, Red, Green & Blue. So what we are doing.

Let's our image size is 8×8

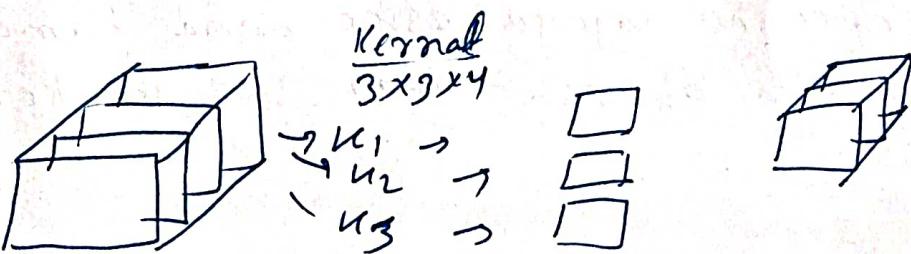
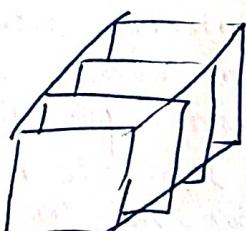
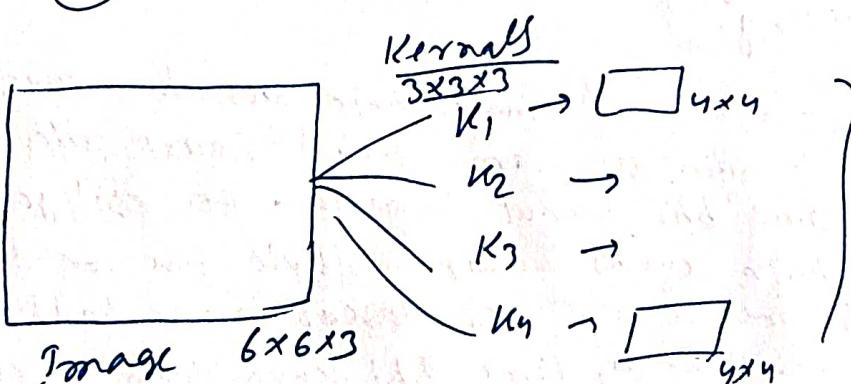
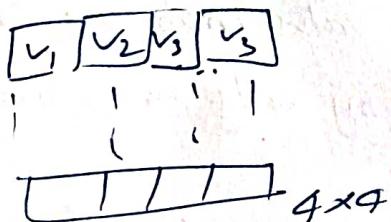
If we mix all three images (Red + Green + Blue) we will get colorful image.

In computer colorful image is basically $(8 \times 8 \times 3)$. So this

3 is called channels. We have one channel for (Red, Green, Blue). If you are reading an image in Python your image dimension will be $(n \times n \times 3)$ each of the channel of same input shape.



\rightarrow Layer 1 + Layer 2 + Layer 3
value = v_1



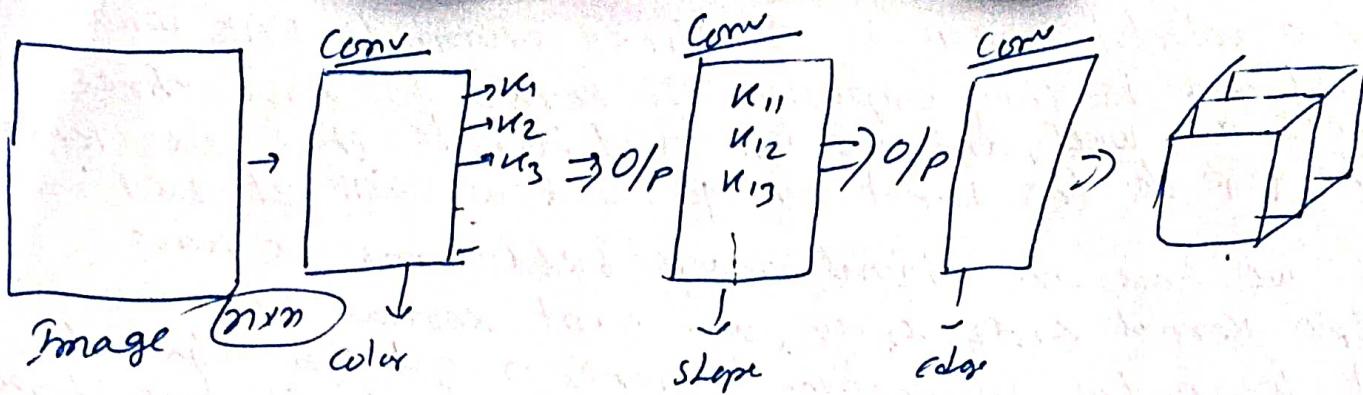
Because a colorful image is of three channels RGB what scientist did is they expanded the kernel to have three channels as well, and we will just iterate this 3×3 kernel on the top of the input image, and we will get 4×4 image.

Let's we have an original image $6 \times 6 \times 3$ and we have multiple kernel K_1, K_2, K_3, K_4 , we want kernel b'coz we want learn the information. so here, we get an output of 4×4 after doing kernel and that kernel will have shape $3 \times 3 \times 3$ and our output will 4×4 , now we will add all the images side by side so we have now $4 \times 4 \times 4 \rightarrow 110$ of kernels.

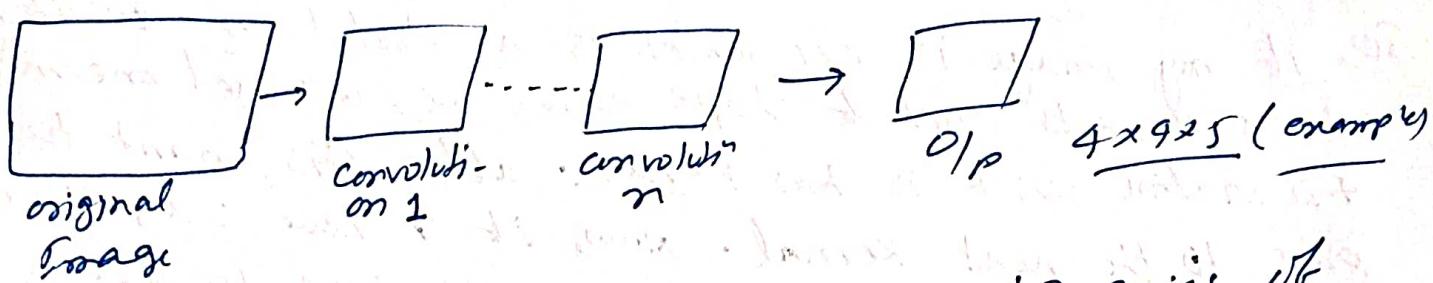
So, if my image is 6×6 which is a colorful image if i do 4 kernels each of shape $3 \times 3 \times 3$ we, will get one more 4-D matrix which is has $4 \times 4 \times 4$. after this, want to pass to the next kernel. now, if i have 3×3 kernel b'coz i have 4-channels here i would have to add a kernel $3 \times 3 \times 4$ then only will be able to do convolution. Let's assume kernel K_1, K_2, K_3 each one have $3 \times 3 \times 4$ all give one output. Now these three images we will add side by side here it will become $n \times n \times 3 \rightarrow 110$ of kernels. This is how convolution works if you are having a RGB image.

Why do we need all of this? how we are going to detect?

Let's we have an image ($n \times n$) we pass it to a convolution layer. convolution layer is basically kernels, bunch of kernels (K_1, K_2, K_3, \dots) it could be anything. whatever the output we get we learned something here, and this output we will pass into the next convolution layer. we have some another kernels so there are diff-kernels. and this kernel shape depend on the output of previous output. on this output we will perform previous operation. at last we will get bunch of images their shape might be (n, n, n, K) . so after doing this convolution we are having some output as an image.



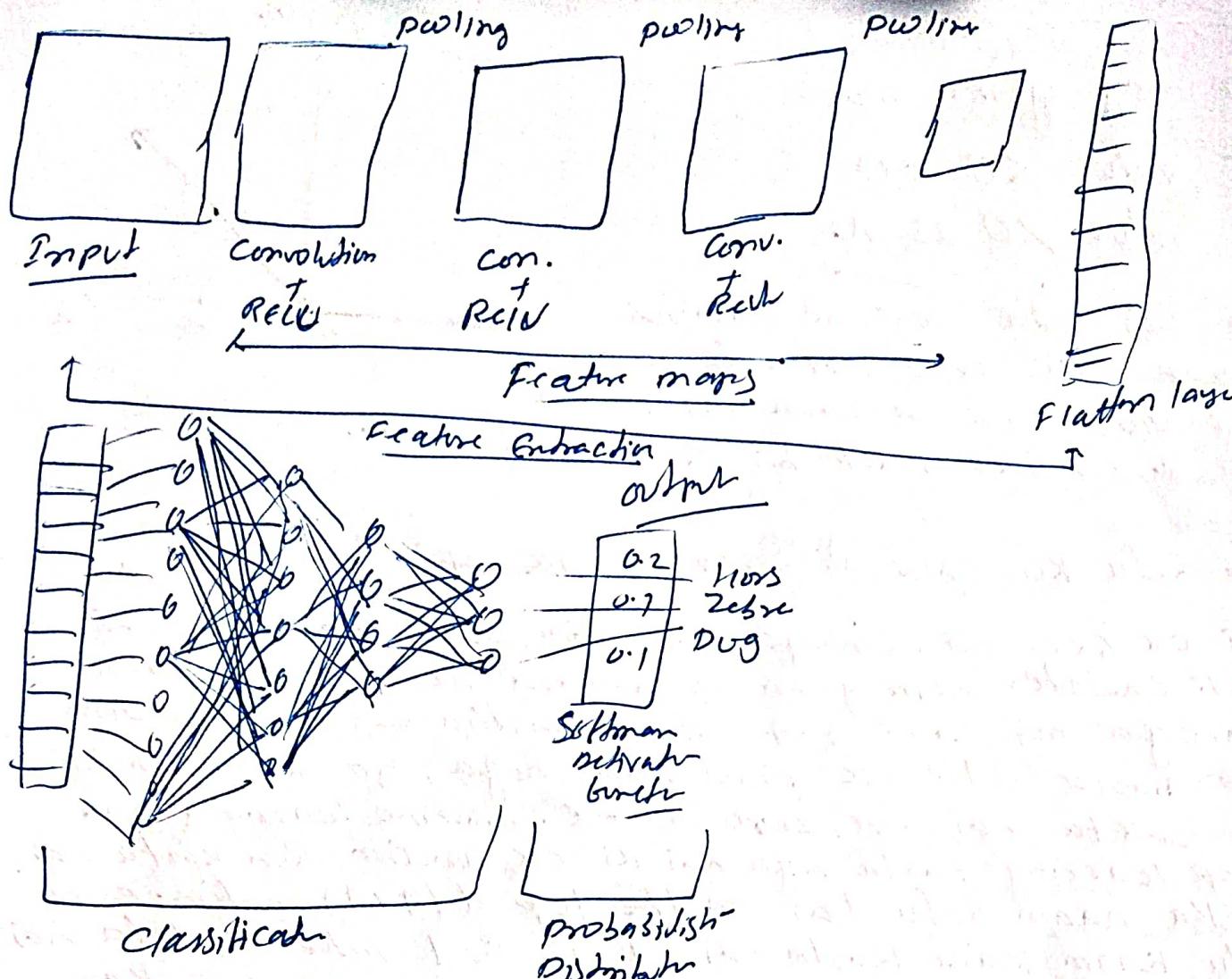
Now why do we need this, what we are going to do with this output that is where now we will apply ml concepts



So I have an original image I pass to series of convolutions and each of the convolutions do padding, strides pooling etc. but at the end we will get this output.

Now this output is having $4 \times 9 \times 5$ shape. Total no. of elements is $4 \times 9 \times 5 = 180 \rightarrow$ so if it stretched this in to a 180 vector $\boxed{1111} \dots \boxed{111}$ Now this

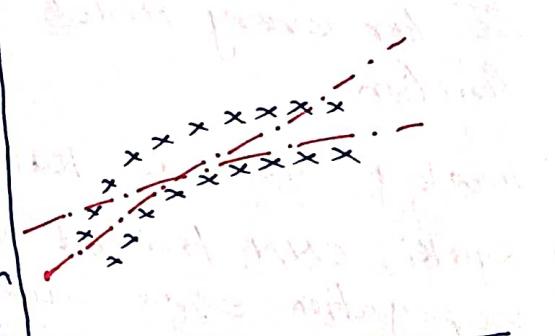
I will pass it to machine learning or multi-layer perceptrons to train my model. So this is basically capturing the all information and able to convert in to vector. This is an image embedding. Embedding is basically we are encapsulating the information from my original image by passing through a series of convolution, able to get to this vector. So this vector is having all of the information that is being learned, so this is a actual representation of my original image. Now I will pass this vector to a ML model. I can predict so compare to the first one in which we have directly image in to vector by not doing any convolution, their accuracy and after doing convolutions we do the modelling on the top of this we get an accuracy of 13% better than previous one. and then they moved to DL/MLP that is how you train a ML model



Activation function

Here, g cannot classify both the class using straight line we need a curve so, here, Activation function comes in to the picture.

So the main importance of activation function is to introduce Non-linearity becoz in real life most of the data that you see is of Non-linear in nature.



there are some famous activation function are

- ① Sigmoid
- ② ReLU
- ③ tanh
- ④ Leaky ReLU

Relu-

for any given value

if value $< 0 = 0$

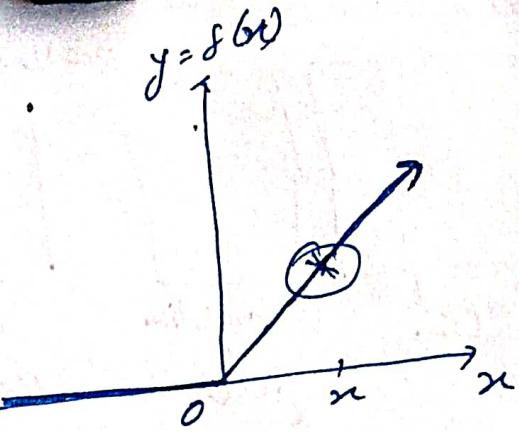
& value > 0 as x

Agora koi value zero se kam hai

to aap usko zero kar dete ho aur

agar value zero se bara hoi to aap

usko x consider karte ho according to Relu.

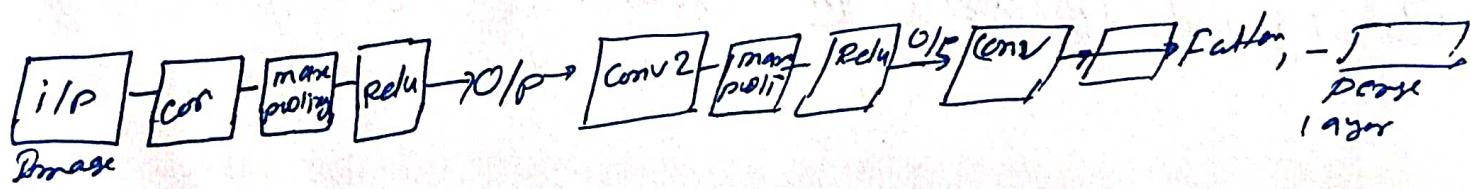


Ham Relu ko CNN ~~fi. abt~~ use kar ~~krte~~ krte.

Esi ek 6×6 abi image leta hain, eske upar 3×3 abi conv. chalate hain yaha se hamara model kuchh learn karega aur hamne 4×4 abi ek matrix dega jis matrix abt har ek value ek pixel value hoga, jo ki kuchh bhi ho sakti hoi, -ve, zero \oplus +ve. machine learning and deep learning karta kya hai ki ek function Run karta hai jiska naam Relu hai; on the top of (4×4) matrix. esme Relu kaam kaise karta hai ki ye first value ko leta hai agar eska value zero se kam hoi, to esko zero kar dega. phir next value par chala jata hoi agar waha par value -ve hoga to usko zero kar dega aur agar wala par value 5 hoi to usko 5 hi chhor dega. So for every pixel value we will apply the Relu function.

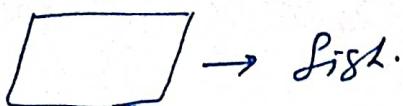
Q. hum log ye sab kar kyu raho hain?

→ Kyuki, CNN trained kaise hota hoi, through forward propagation eske baad loss calculate karte hain, aur eske baad loss ko minimise karte hain using backward propagation. esi process me hum-log bahut saare differentiation karte hain, to hamare bahut saare mathematical complexity kam ho jata hoi, aur process bhi fast ho jata hoi



ext

I have an image, I have to classify this as a fish

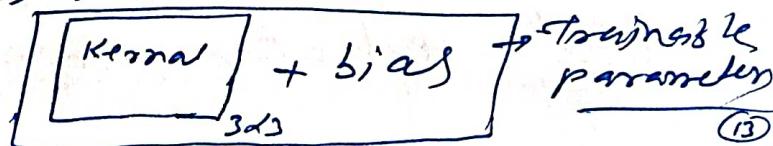


→ Fish.

I have my input image, and I pass pass it to only one convolution layer which is 3×3 with only one kernel and I get output, after getting output I will expand it to flatten, and this I will train a ml model and I am getting a prediction \hat{y}_i . So if this kernel learn something then this flatten vector should be able to differentiate what is a fish. 'coz this output at the end encapsulates the information that is present in the given input image so, let's say in the initial stages we don't know what is the value inside of the kernel, so what you did is you randomly give values $(0, 1, 2, 3, 4, 5, 6, 7, 8)_{3 \times 3}$ and you apply convolution on the top of the ip image and at the end you got the kernel, and at the time of predicting that image as a cat. So what happen, your kernel values are not learning that it is a fish, we are having error. we, will say to model that you made mistake, so what model will do, it will change the kernel values. so in the next iteration, with the same input image you are changing the kernel value after that at the time of classifying the image you classify as a fish. that means this kernel is working for us. on this input image to classify a given image. and how it is learning it by passing this image multiple time to this kernel and calculating this output and output is not what we expect, then it will go back and update this values

Bias in CNN: → In CNN we have kernel so for each kernel, I will add one variable called bias so this could be any value. and

this bias will shift the values of the kernel properly based upon the input values, so whenever you pass a kernel by default it is having bias variable as well. so all of them are trainable parameters.



`OS.listdir('path of dir')` → cat folder ka path denge to saare
cat ka image ka list load kar lega.
dog

for print 9 image randomly

`random.sample(OS.listdir('path of folder'), 9)`

→ result is not an image that is only file name.

bt g want to read this as an image then g have to
get the absolute path

what is absolute path → /content/dogs-vs-cats/train/cats/cat-0.jpg
This is the absolute path of certain
image.

we need to join

/content/dogs-vs-cats/train/cats
+

random image of dogs/cats that we have

/cat-0.jpg

so that is where we need to run

`image_path = OS.path.join(directory_path, image_file)`

Now g will do the

`Image.open(image_path)`

`OS.Walk (directory path)` → g's a looping function which will
go through all the values of all
the files inside of your given directory. ~~so inside that~~

what `OS.walk` will do is → it will go through each and
every directory and it will list down all the content that
is available in this directory.

ImageDataGenerator → load images in batches, batch could
be any no.

`validation_split = 0.2` → means train → 80%.

Validation → 20%.

flow_from_directory

directory → from which directory g want to get the data
target size → while getting the data convert each image
into the shape $(256, 256)$ → so whatever the size
of the image that have it will convert
 $(256, 256)$

color_mode = 'rgb'

class_mode → 'categorical', 'binary' → if you have binary classification
use binary else categorical &
multiclass classification

shuffle = True → while loading the data we will shuffle the
image.

batch_size = 32, 100, 200, etc

if our training accuracy is more compare to validation
accuracy → overfitting.

if training accuracy is less & validation accuracy is
also less → underfitting.

Model checkpoint:→ Epoch 1 validation loss improved
and model got saved this is
called model checkpoint.

verbose = 1 → whenever it is saving everytime
it is saving it will print, i.e.

verbose = 2 → For every two time it will print, but
it will save in the background. it will
not clutter your output

What are non-trainable parameters:→

Let's say we have max-pooling layer there we have nothing
actually, we don't train anything. we just decreasing the
value, these are called non-trainable parameters

Data Augmentation:

Data Augmentation is a technique to increase the no. of images by creating modified version of existing data through methods such as rotation, translation, scaling or flipping with the aim of improving model generalisation & performance.

`restore_best_weights = True`

↪ jab bhi ham no. of epoch chalate hain. tas ham har ek epoch par model ko save bhi karte jaate hain, wala par hamne accuracy aur validation loss bhi milta hai. jab hamara validation loss me jyada ka improvement hona band ho jata hai. tas hum pickle waale epoch me jaate hain jada par hamara validation loss aur accuracy best point par tha. aur hum usko save kar lete hain.

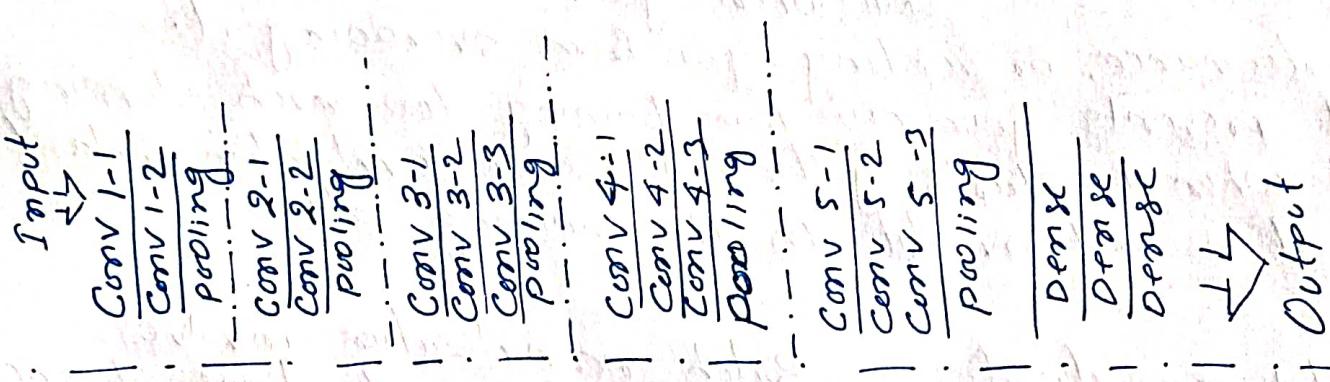
Transfer learning

Transfer learning in CNNs refers to the practice of taking a pre-trained model (Convolution Neural Network) which has been trained on a larger dataset for a specific task like image classification, and then adapting it to a different task with limited data by fine-tuning its parameters.

e.g.: you can take a pre-trained model (CNN) trained on ImageNet image classification and fine-tune it on a smaller dataset for a specific classification task like identifying different species of flowers. This approach saves time and computational resources while often achieving good performance especially when the new task is related to the original task.

VGG16 Architectures.

VGG16 is object detection and classification algorithm which is able to classify 1000 images with 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.



16 trainable layers

Conv Layers - capture spatial info

FC Layer - Classification Task.

Convolution layers use 3x3 filter with stride 1 and always used the same padding. Maxpool layer of 2x2 filter of stride 2.

Conv-1 layer has 64 number of filters.

Conv-2 has 128 filters.

Conv-3 has 256 filters.

Conv-4 has 512 filters.

Three Fully Connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one of each class). The final layer is the softmax layer.

Model - checkpoints → you are taking snapshots of your CNN's progress during training, it ensures you don't lose your best performing model in case something goes wrong during training, so you can always go back to your best check points.

Early stopping: Early stopping is like having a coach who tells you to stop practicing a skill when you've reached your peak performance, preventing you from overdoing it. It prevents your CNN from training too much, stopping it when it starts getting worse on a validation dataset. So it doesn't overfit to the training data.

Batch normalization: Batch normalization involves normalizing the activations of each layer in a neural network by adjusting and scaling them to have a mean of zero and a standard deviation of one, based on the mini-batch of data during training.

- (i) Stability: It helps stabilize the learning process by reducing the chances of activations becoming too large or too small, which can slow down or destabilize training.
- (ii) Faster training: By normalizing activations, it allows the neural network to converge faster during training, reducing the no. of epochs needed to reach a certain level of performance.
- (iii) Regularization: Batch normalization acts as a form of regularization, reducing the need for other regularization techniques like dropout which can sometimes slow down training.