Dt:- 10/05/22
10:00pm

Convolutional Neural Network

CNN = image classification

Q:- How Computer read image

Ans :- Pixels

* As pixel increases, picture quality increases.

EX: TV

- 4K → 4000Pixels
- HD → 720 Pixels
- Full HD → 1080 Pixels
- 2K → 2000Pixels

* Maximum intensity of pixel = 255 pixels

* Minimum intensity of Pixel = 0 pixels

EX: Marks
Max - 100
min - 0.

* images
  - Black & white image [2D] array
  - color image [3D] array

Black & white [2x2 px]

Pixel 1	Pixel 2
Pixel 3	Pixel 4

"2D" array →

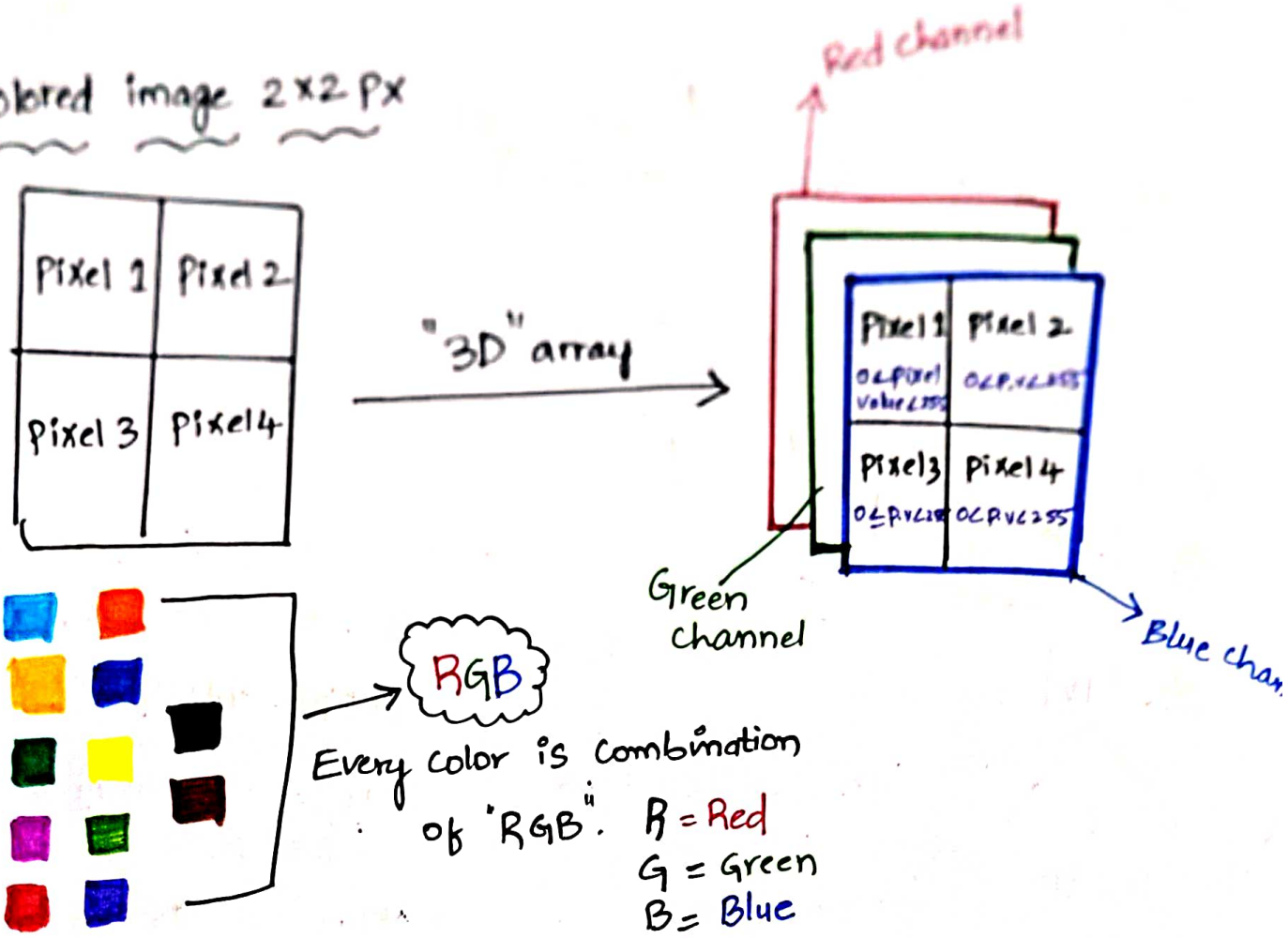
Pixel 1	Pixel 2
0 < pixel value ≤ 255	0 < p.v < 255
Pixel 3	Pixel 4
0 < p.x < 255	0 < p.x < 255

255 → B
0 → white.

→ color intensity = 255

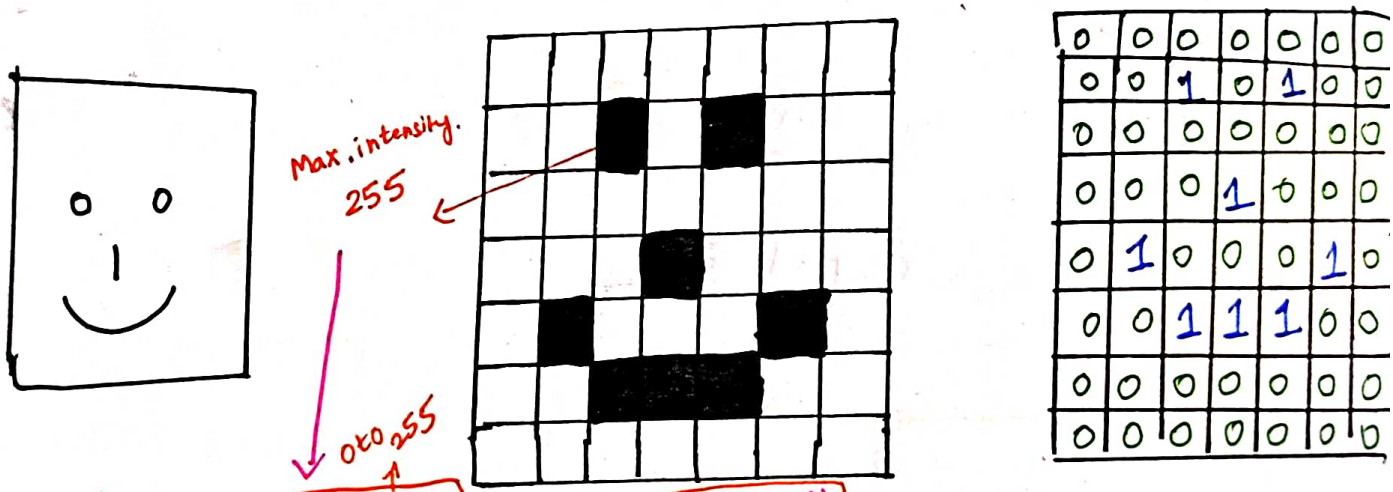
→ color intensity = 140

* Colored image 2×2 Px



Every color is combination of "RGB". R = Red
G = Green
B = Blue

EX:-

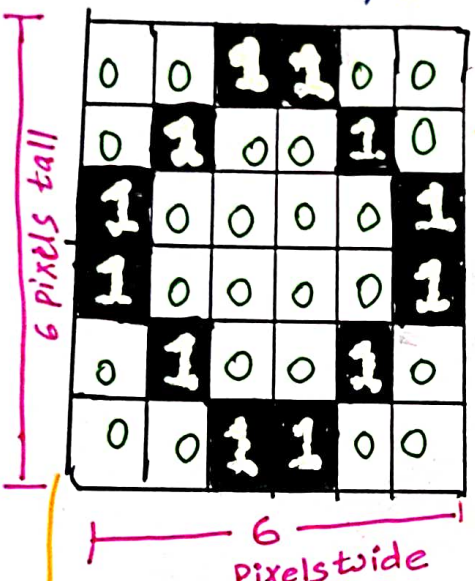


0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

*** Converting the values of pixels between "zero" to "one" is known as "Min max scaling"

Q. Why not fully Connected networks ?

why neural network is not Considered for Image classification.

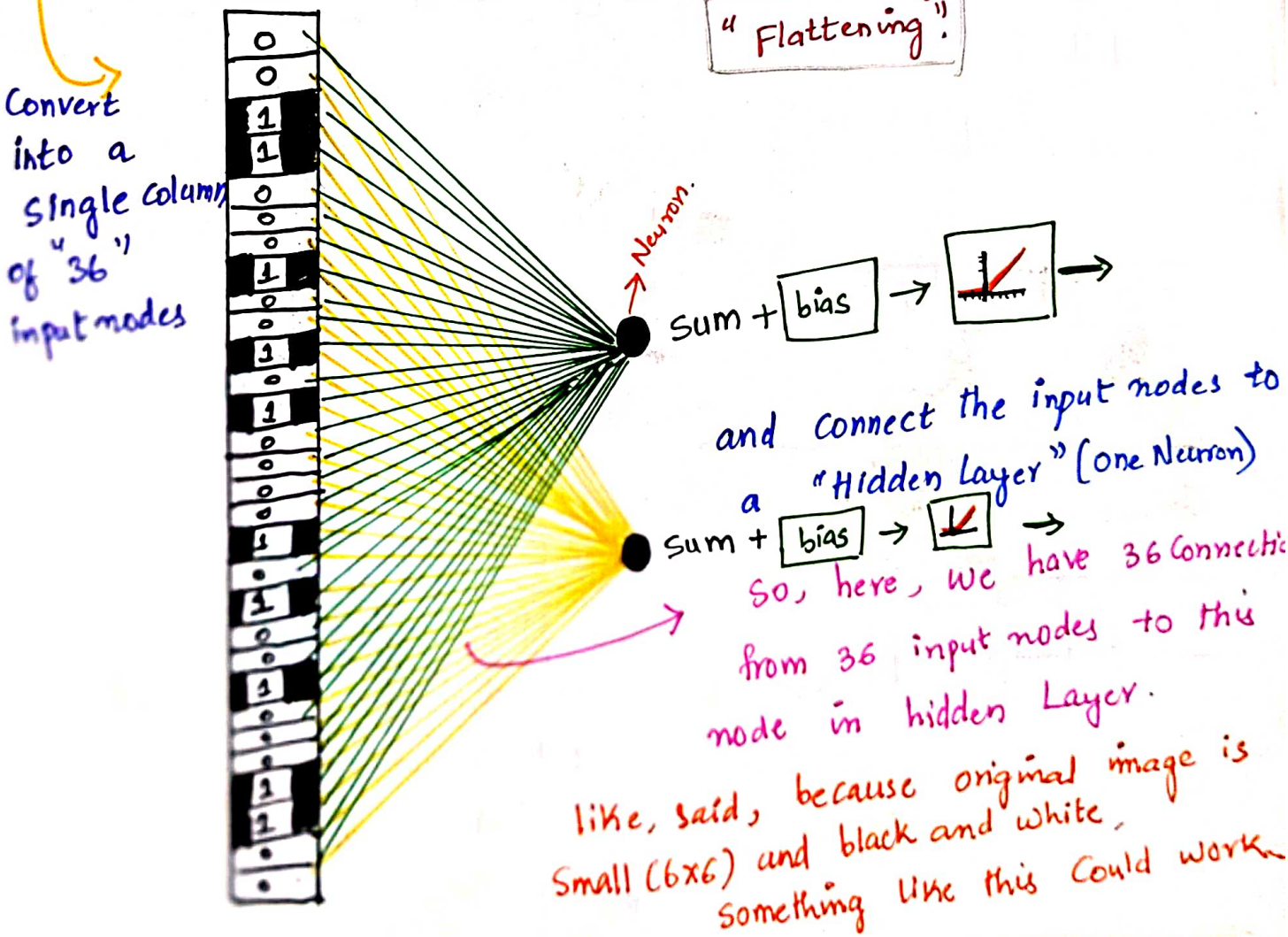


We will start with the image of letter "0".

6/6 pixels

its possible to make a normal, Every day Neural Network that can correctly classify it.

we convert 2Dimensional (or) 3 dimensional array to one dimensional array is called as "Flattening".



But, if we had Larger image,
Like 100 pixels by 100 pixels, which is still Pretty small Compared to real world pictures...

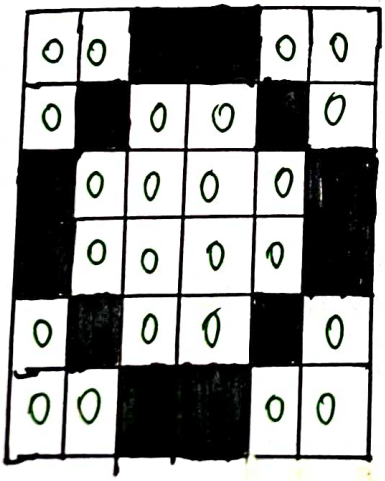
Then, we would end up with having to Estimate 10,000 weights per node in the hidden Layer!!!

So, this method doesn't scale very well.

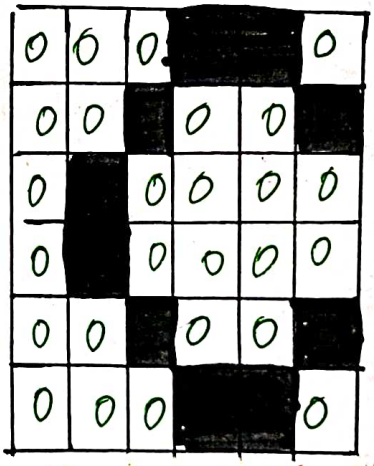
1 neuron = 10,000 weights
if more neurons, more weights.

* Another problem is that, its not clear that "Neural network" will still perform well if the image is shifted by one pixel.

■ = 1



Shifted to the right 1 pixel



Than, it is not clear that the "neural networks" will still recognise this letter "O" correctly if each pixel is shifted to the right by one.

Ex:-

a a a a

Everything "a" is same for us. but in "Neural network"

a a

These both images are "different

* Classification of Large and Complicated images is usually done using "Convolutional Neural network".

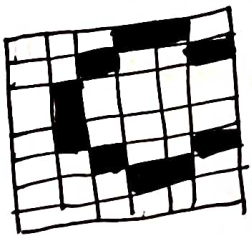
Why Convolutional Neural Networks ?

* Convolutional neural networks

1. Reduce the number of input nodes.
2. tolerate small shifts in where the pixels are in the image.

Ex:-

← Same as This ←

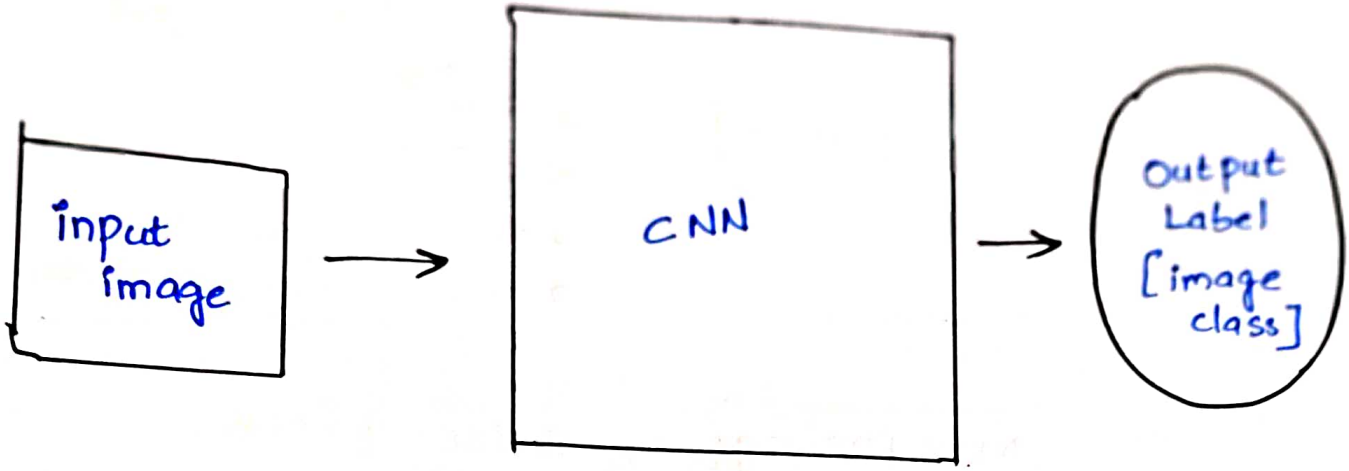


What is Convolutional Neural network ?

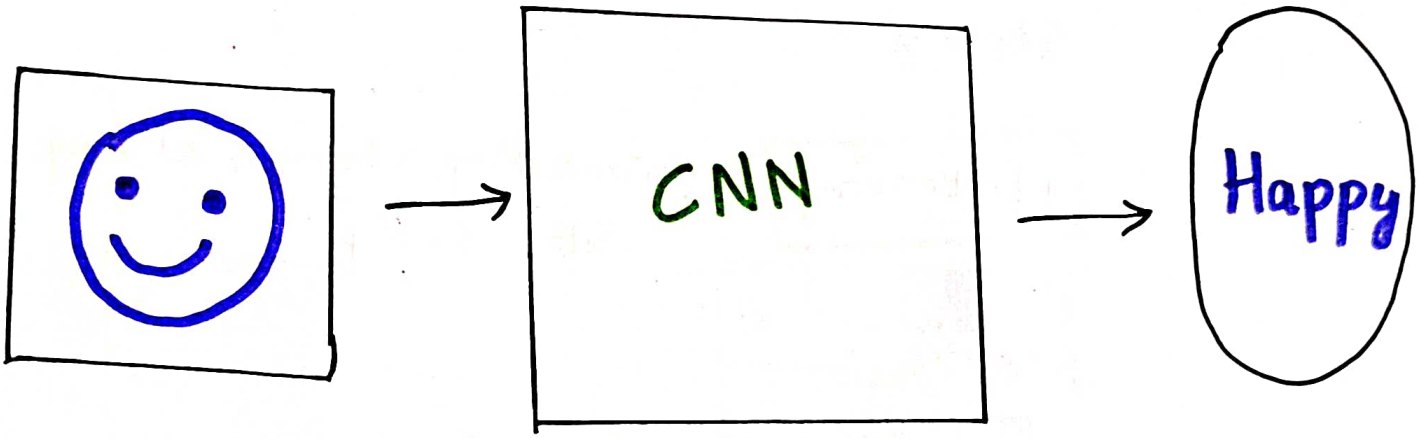
CNN (or) Convolutional Neural network is a type of feed forward Artificial neural network in which the Connectivity pattern between its neurons is inspired by the organization of animal visual cortex.

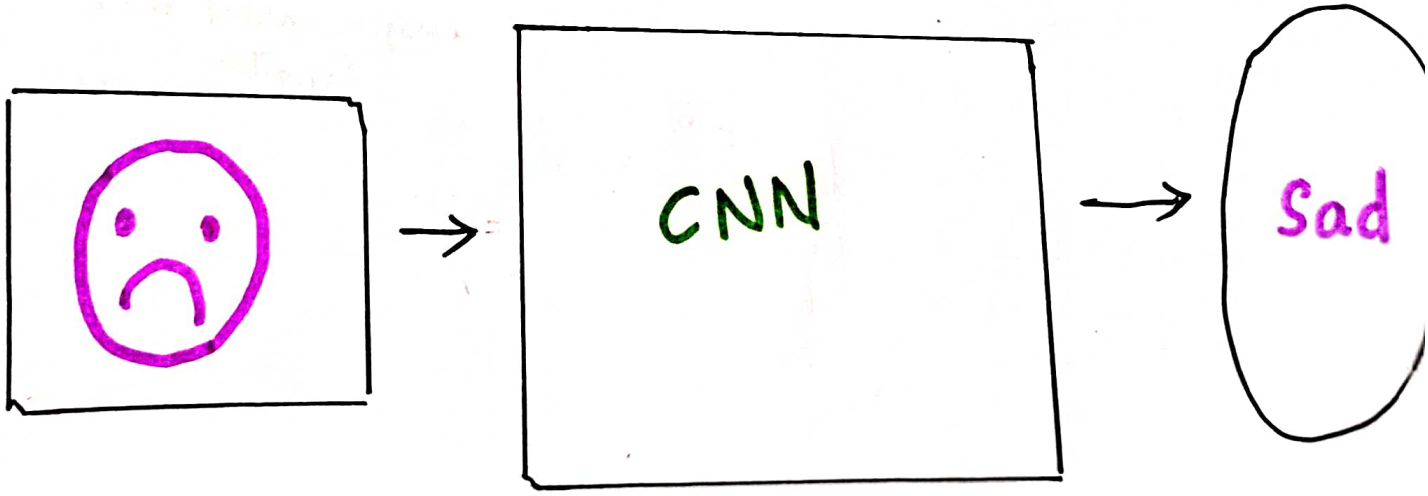
* Visual cortex has small regions of cells that are Sensitive to specific regions of visual field.
* Some individual neuronal cells in the brain responds (or) fires only in the presence of edges of a certain Orientation.

Ex:- Some neurons fires when exposed to vertical edges and Some when horizontal (or) diagonal edges.



Example :-





STeps

STep 1 :

Convolution —

② step(a): Convolution operati
④ step(b): ReLu Layer
③ step(c): padding
① step(d): Min max scaling.

Step 2 :

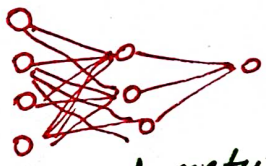
Max Pooling → resize of image.

Step 3 :

Flattening — Converting 2d array & 3d array into 1d array

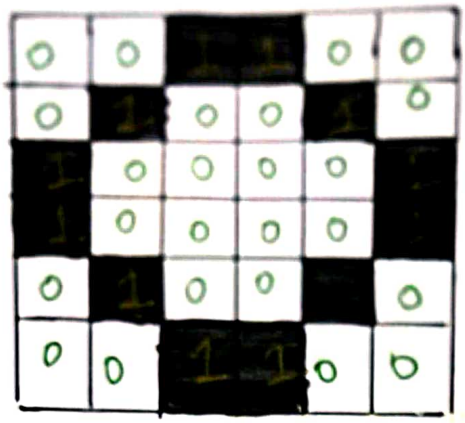
Step 4 :

Full Connections —

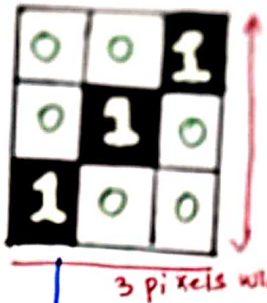


Neural network.
* weights updated back propogation.

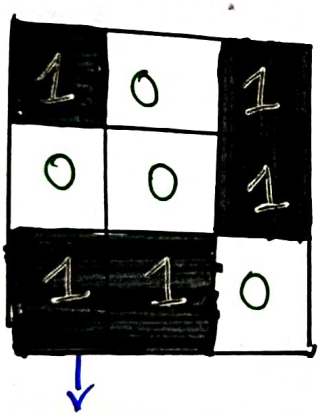
input image



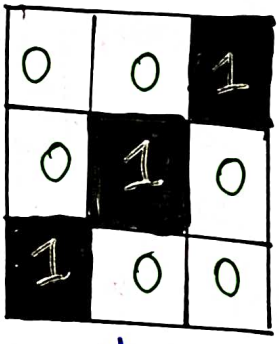
Filter



The first thing a "convolution neural net
does apply a "filter" to a input image



In other, words, before training a "CNN", we start with a random Pixel values

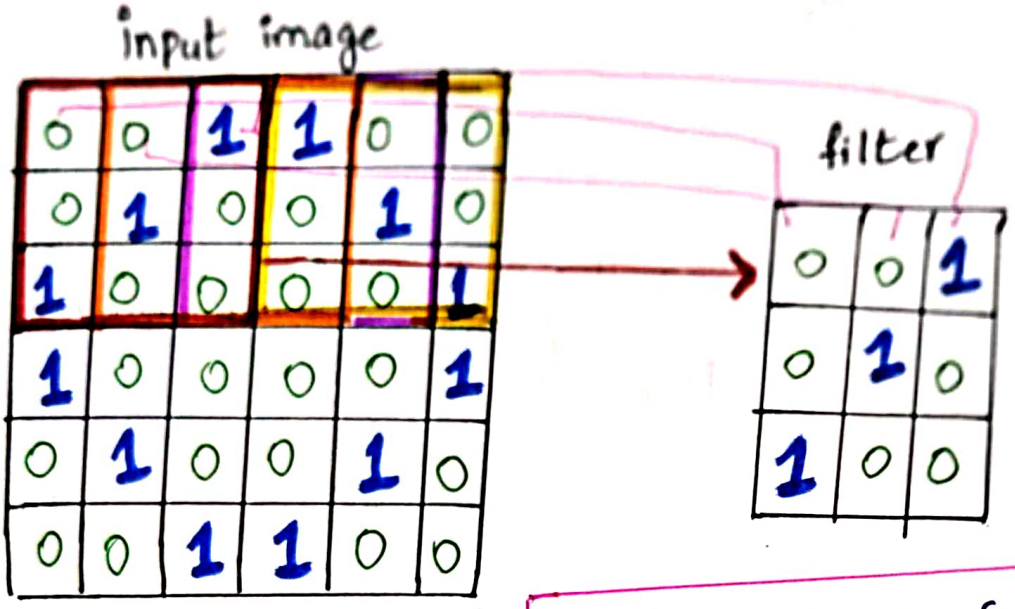


after, training to with "Back propagation" ,
end up with somet
more useful.

What is Dot Product ?

Element wise multiplication, is known as Dot Product.

↳ position wise

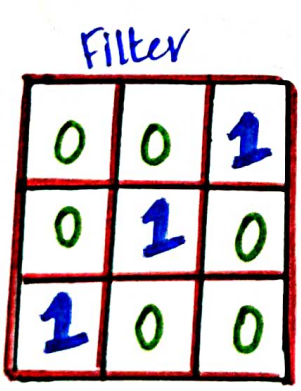


$$= \begin{aligned} &(0\times0)+(0\times0)+(1\times1) \\ &+(0\times0)+(1\times1)+(0\times0) \\ &+(1\times1)+(0\times0)+(0\times0) \end{aligned}$$

Dot Product ↓ Sum of product

$= 3$

By computing "Dot product" between the input and "Filter", we can say Filter is convolved with the input, and that what gives "Convolutional Neural networks". this name



$= 3$

Now, we add Bias term to the output of Filter

+-2

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	1

"Feature Map"

Input image : 2

0	1	1
1	0	0
0	0	0

Filter

0	0	1
0	1	0
1	0	0

→ +-2 ↓ bias

Filter

1	1	0
0	0	1
0	0	0

$(0\times0)+(1\times0)+(1\times1)$
$+(1\times0)+(0\times1)+(0\times0)$
$+(0\times1)+(0\times0)+(0\times0)$

$= 1+(-2)$

$= -1$

Filter

1	0	0
0	1	0
0	0	1

$= (1\times0)+(1\times0)+(0\times1)$
$+(0\times0)+(0\times1)+(1\times0)$
$+(0\times1)+(0\times0)+(0\times0)$

$= 0+(-2)$

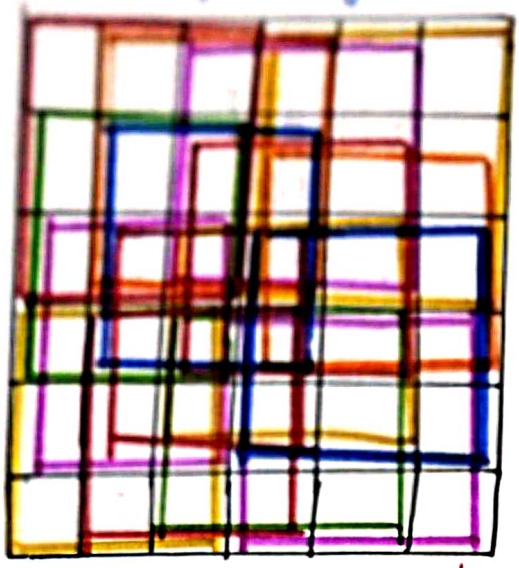
$= -2$

$= (0\times1)+(0\times0)+(1\times0)$
$+(0\times0)+(1\times1)+(0\times0)$
$+(0\times1)+(0\times0)+(1\times0)$

$= 1+(-2)$

$= -1$

Input image.

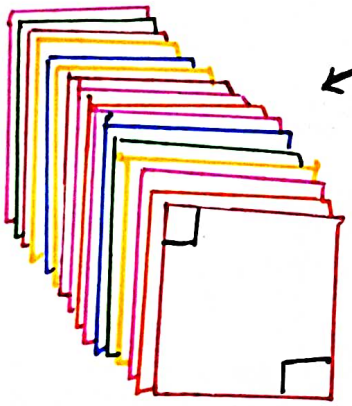


Filter

0	0	1
0	1	0
1	0	0

↓

from these, we obtain "Feature Map".



"Convolutional Layer"

We create many Feature maps to convolution Layer.

converting all values of pixels

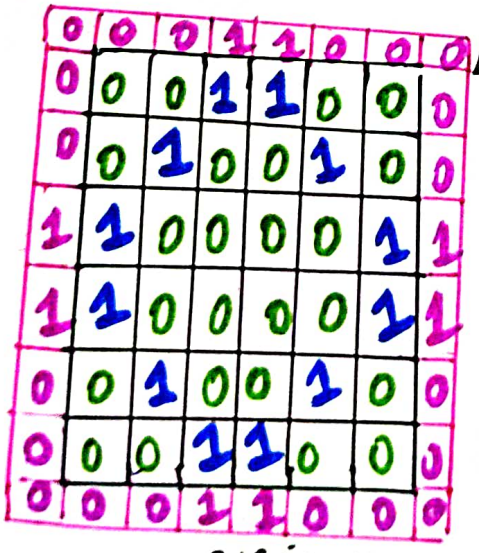
0	0	255
0	255	0
255	0	0

→

0	0	1
0	1	0
1	0	0

min. max scaling

Padding



8×8 image.

concept of adding layer top of Image is called "padding"

we are protecting image by adding another layer on top.

$n - f + 1$

$= 8 - 3 + 1 = 6$

different paddings

1. Zero paddings
2. nearest Value

Step : B

ReLu Layer.

↓

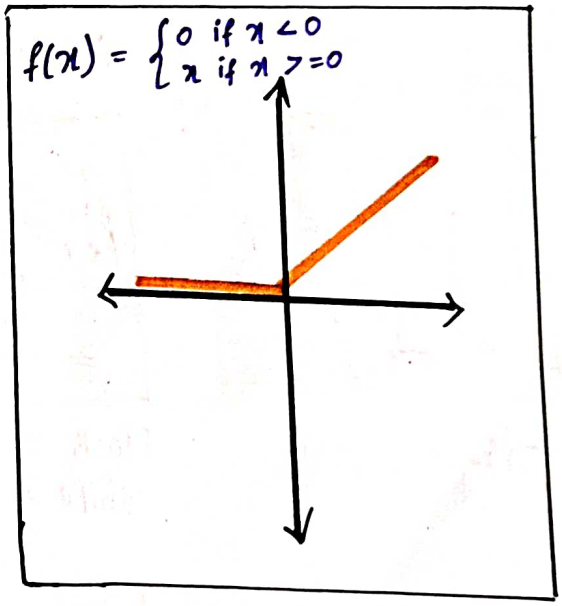
Rectified Linear unit.

" Where ever the negative Value, Every negative Value is converted = "0".

" what ever the positive value, Every positive value should be considered Same (or) directly.

Ex :-

x	$f(x)=x$	$F(x)$
-3	f(-3)	0
-5	f(-5)	0
3	f(3)	3
5	f(5)	5



Rectified Linear unit (Relu)

transform function only activates a node if the input is a certain quantity, while the input is below zero, The o is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent Variable.

* Combination of filter + Relu

Feature Map

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-2	1

Relu

Feature Map, Post ReLu

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

Now typically, we run The "Feature Map" Through "Relu activation function"

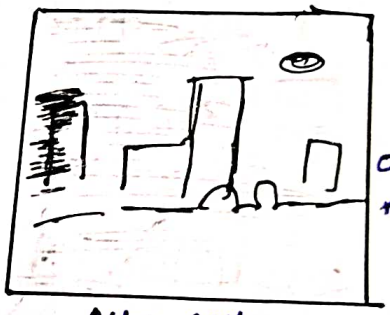
and That means that all of the negative values are set to "0" and positive values will be Same as before

Ex:-

Original image

Applying Filter ↓ Edge (Feature Map)

Black = negative
White = positive value



Only non-negative values.

After apply Relu
Edges will clearly visible

↓

Edge detection.
Filter is applyed. in the picture.

↓

post Relu

Step-2

Max pooling

orginal image.

tilted

Shrinkage

Pooling Layer

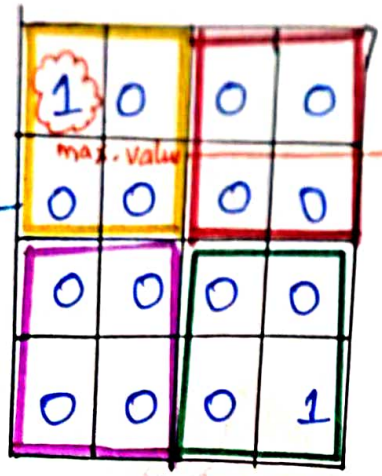
In this Layer, we shrink the image stack into Smaller size.

Steps :

1. pick a window size (usually 2 or) 3)
2. pick a stride (usually 2)
3. Walk your window across your filtered images
4. From Each window, take the minimum Value.

Feature Map, Post Relu.

* Max, pool size (2,2)



Max. pooling

↳ Extracting

Maximum value

Exatracting the Maximum value from "post ReLu" is called as "Max pooling".

We, apply another filter to New "Feature Map".

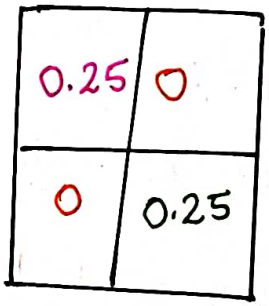
Min. pooling.

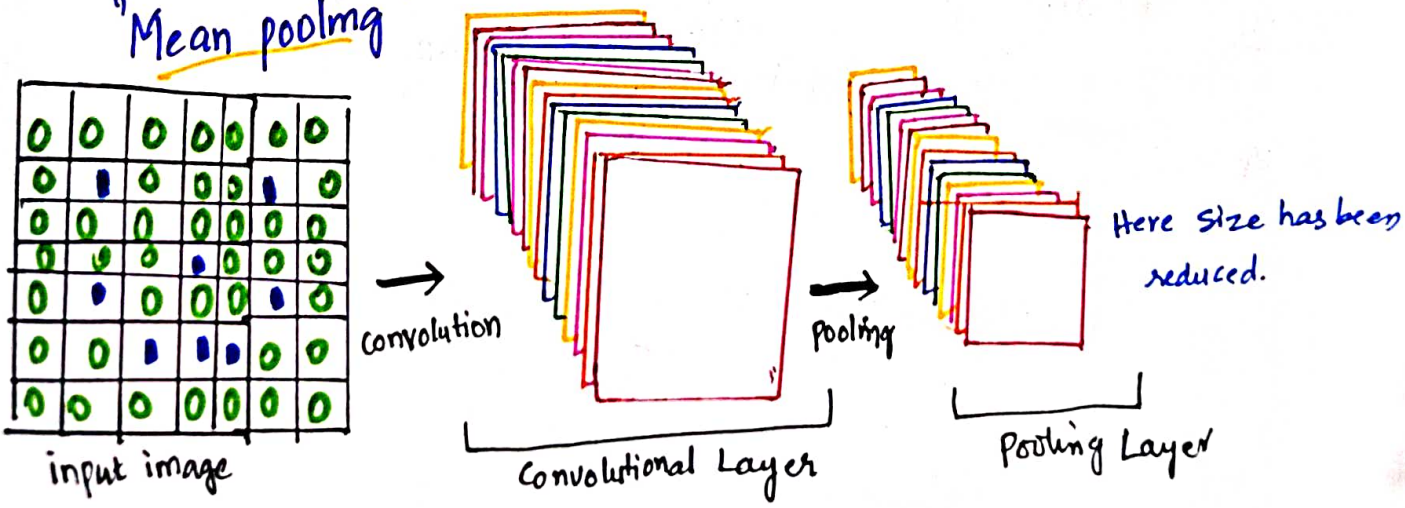
↑ Extracting Minimum value from post relu

Mean pooling.

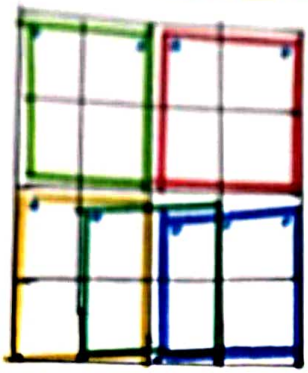
↓ We calculate the average value for each region and that would be "called Average (or) "Mean pooling"

Mean Pooled





Stride



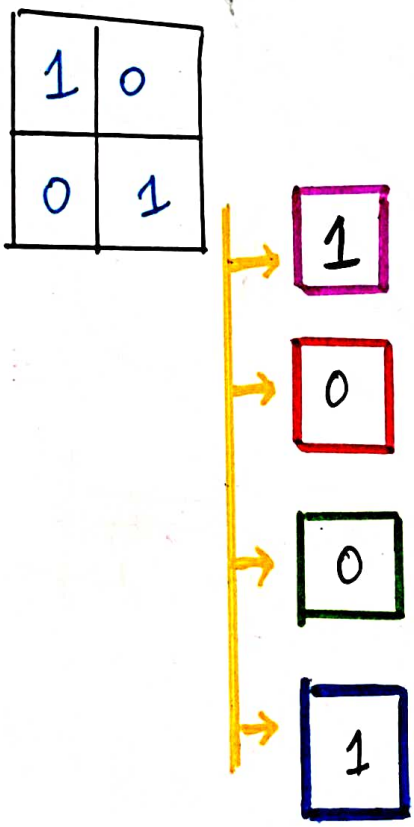
⇒ stride = 2

⇒ stride = 1

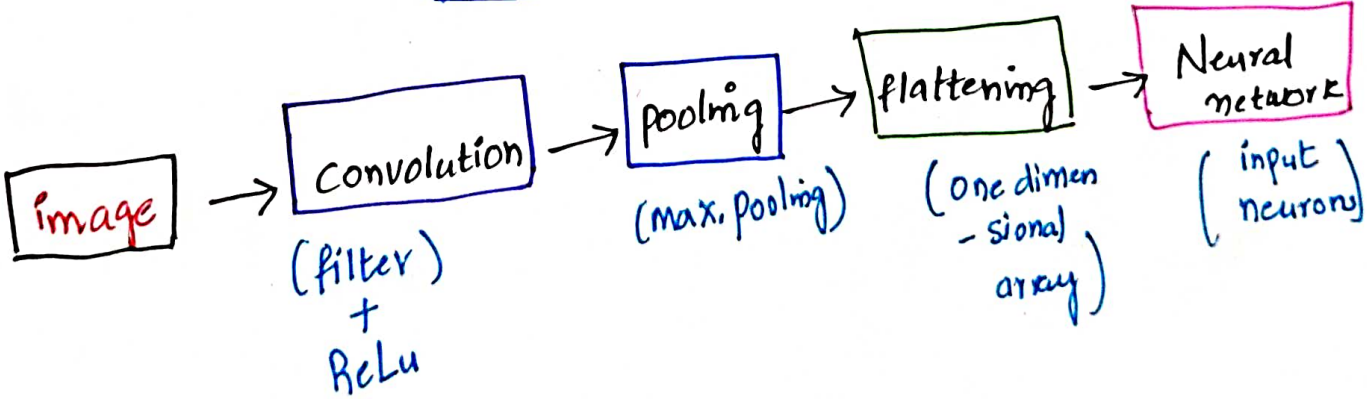
STep-3

Flattening

After Max.pooling, we apply Flattening

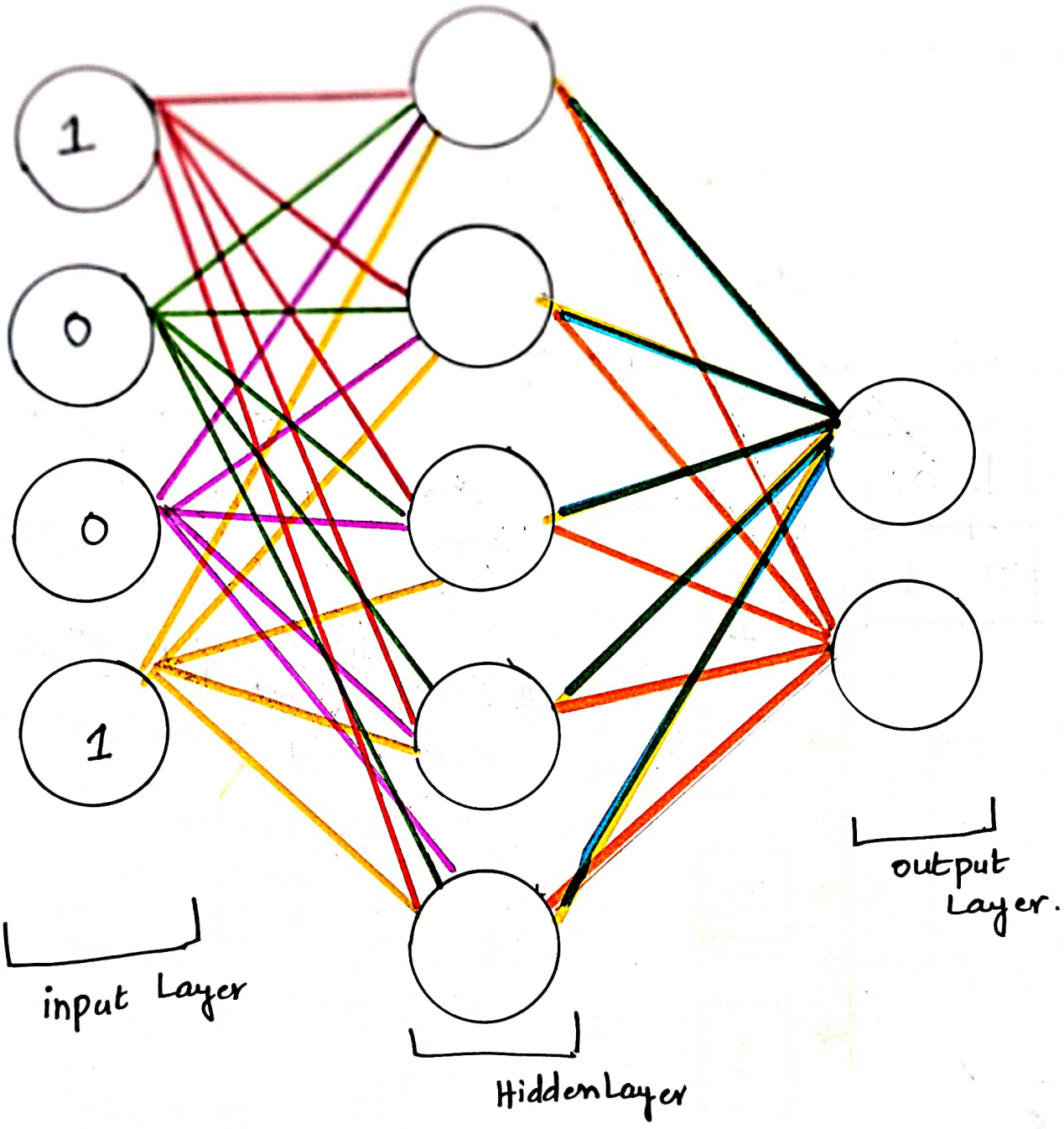


Lastly, Let's plug the input Nodes into a normal, Everyday "Neural network"



Step 1

Full connections



Code

```
import tensorflow as tf
import keras
```

Part 1 Data Preprocessing

```
from keras.preprocessing.image import ImageDataGenerator
```

↓ Select the data and Generate.

Processing the training set

whatever the image taken, Divided 1/255

```
#train_datagen = ImageDataGenerator(rescale = 1/255,
                                    shear_range = 0.2, zoom_range = 0.2
```

shear_range → If some images tilted

zoom_range → shrink pics.

```
# training_set = train_datagen.flow_from_directory
```

flow_from_directory → one after one, from given directory

```
location ← ("dataset/training_set",
            target_size = (64,64),
            batch_size = 32,
            class_mode = "binary")
```

target_size → Every image 64/64 consider.

batch_size → Each batch = 32 images.

class_mode → cat (or) dog.

Out: Found 8048 images belonging to 2 classes.

- Dogs : 4000
- cats : 4043

Ex:- change directory.

```
import os
os.getcwd()
```

Out: "C:\\Users\\user"

change directory

#Os.chdir("D:\\datascience\\srk\\9. Deeplearning\\CNN")

present directory

pwd

Out. "D:\\datascience\\srk\\9. Deep learning\\CNN")

#training_set.class_indicies

Out: {"cats":0, "dogs":1}

Preprocessing Test set

test_datagen = ImageDataGenerator(rescale = 1/255)

test_set = test_datagen.flow_from_directory

("dataset/test_set",

target_size = (64,64), — Fixed

batch_size = 32,

class_mode = "binary").

Out: Found 2000 images belonging to 2 classes.

Part-2 Building the CNN

```
# Initialising the CNN
from keras.models import Sequential
# classifier = Sequential()
```
(Sequential — capital)

Step-1 Convolution

```
from keras.layers import Conv2D
# classifier.add(Conv2D(filters = 32, kernel_size = 3,
                        activation = "relu", input_shape = [64, 64, 3]))
```
(Conv2D — capital; filters = 32 → 32 Feature maps / images; kernel_size = 3 → Filter size 3/3; input_shape → Fixed; [64, 64, 3] → Every images of 64, 64, 3 → dimension color image.)

Step-2 pooling

```
from keras.layers import MaxPooling2D.
# classifier.add(Maxpooling2D(pool_size = 2, strides = 2))
```

Adding a Second Convolutional Layer.

```
# classifier.add(Conv2D(filters = 32, kernel_size = 3,
                        activon = "relu"))
# classifier.add(Maxpooling2D(pool_size = 2, Strides
```

Step : 3 - Flattening.

```
from keras.layer import Flatten.

# classifier.add(Flatten())
```

Step 4 : Full Connection.

```
#hidden Layer
from keras.Layers import Dense
#classifier.add(Dense(units = 128, activation = "relu"))
```

↳ connections.

Step5 : Output Layer.

```
# classifier.add(Dense(units = 1, activation = "sigmoid"))
```

Part-3 Training the CNN

Compiling the CNN

```
# classifier.compile(optimizer = "adam", loss = "binary-
                     crossentropy", metrics = ["accuracy"])
```

Training The CNN

8048 / 32 = 252 images
↓ Batch size
↳ Every time, it continue with in Epoches 1-25

On the training set and Evaluating it on test set

```
# classifier.fit(x = training_set, validation_data =
                 test_set, epochs = 25)
```

Out
Epoch 1/25
Epoch 2/25 ----------) ETA: 5:54, loss: 0.8589, Acc = 0.507

Part-4 Making a Single prediction.

```
# import numpy as np
  From keras.preprocessing import image

# test_image = image.load_img("dataset/single_predict
                              /cat_or_dog_2.jpg", target_size
                              =(64,64))
# test_image = image.img_to_array(test_image)
# test_image = np.expand_dims(test_image, axis =
```

Predict

```
# result = classifier.predict(test_image)
# training_set.class_indices → classify how many types of images
```

Out: {"cat":0, "dogs":1}

```
if model_pred[0][0] > 0.5:
    prediction = "dog"
else:
    prediction = "cat"
```

```
# print (prediction)
```

out cat

Ex :-

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

6X6

10/05/22
6:30 Am

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

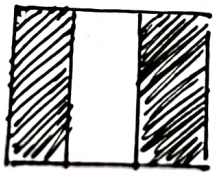
3X3
(Vertical Edge)
filter

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

4X4

ex)
n = 6
filter (f) = 3

6X6 = 3X3 = 4X4



n - f + 1

(dp)

6 - 3 + 1 = 4