UNIT-3

Convolution Neural Network (CNN)

Also called ConvNet

→ CNNs are widely used in images recognition, image classifications, objects detections, recognition faces etc.

→ CNN image classification takes an input image, process it & classify it under certain categories.

→ CNN is another type of neural network that can be used to enable machines to visualize things & perform tasks such as image classification, image recognition, object detection etc.

→ Image classification is the task of taking an input image & outputting a class (a cat, dog etc) or a probablity of classes that best describes the image.

→ CNN is a specialized type of neural network model designed for working with image data.

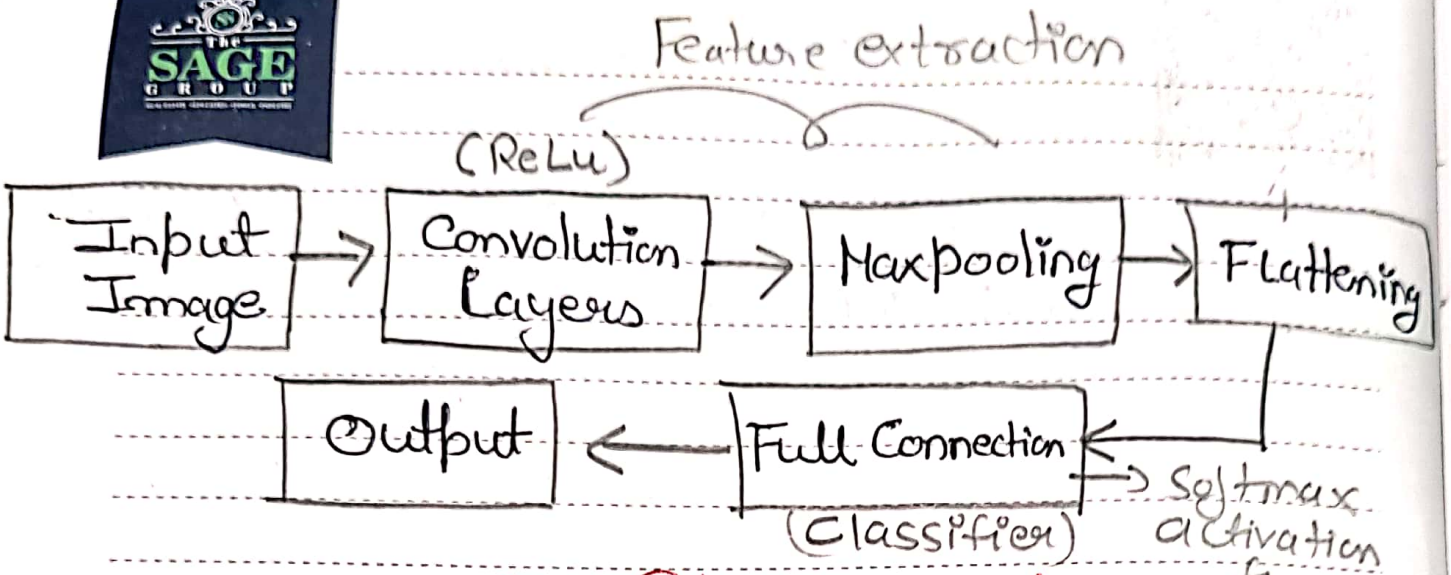


Fig → Structure of CNN

Input Image → Convolution Layer (ReLu or Leaky) → Pooling Layer (Maxpooling) → Fully Connected Layer (Flattening)

Flattening → convert 2D into 1D

Softmax activation func ← O/P (Multiclass)

Input Image → CNN → output Label (Image Class)

Cat Image size → 480 × 480 × 3 ← RGB values

Array of Pixel values

Output numbers that describe the probability of the image being a certain class

[0.90, 0.1, 0.9]

Cat Dog

Array of nos

→ In a 3D image every pixel will 3 values separate channels for red, green, & blue color values.

→ @ 0-255 value describes the pixel intensity at that point.

→ We want the computer to do is able to differentiate between all the images it's given.

→ For that computer perform image classification by looking for low-level features such as edges & curves & then building up to more abstract concept through a series of convolutional layers.

→ In CNN, the input image pass through a series of convolutional layers, pooling (downsampling) layers, & full connected layers & finally produce the output which can be a simple class or a probability of class that best describes the image.

P.T.O.

Why CNN

→ Feed forward neural networks can learn a single features representations of the image but in case of complex images, Neural Network will fail to give better predictions, this is because it cannot learn pixel dependencies present in the images.

CNN can learn multiple layers of features representations of an image by applying different filters / transformations.

→ In CNN, number of parameters for the network to learn is significantly lower than Multi-layer network

STEP-1 CONVOLUTION LAYER

→ This layer performs an operation called a convolution, hence network is called CNN

→ It extract features from the input image Convolution is a linear operation that involves the multiplication of a set of weights with the input

→ The technique was designed for 2D input ( )

→ The multiplication is performed between an array of input data & a 2D array of weights called a filter / kernel

Convolutional operation

It is a linear application of a smaller filter to a larger input that results in an output feature map.

→ Size of filter is smaller than the input data

→ Multiplication performed between a filter size patch of the input data & the filter is a dot product, which is then summed & generate a single value.

→ The filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom.

→ If the filters is designed to detect a specific type of features in the input (image) then

It discuss that features are there in the image.

→ The output from ~~on~~ multiplying the filter with the input array, one time is a single ~~array~~ value, as the filter is applied multiple times to the input array, it produce a 2D array of output values called ~~failure~~ map features

n=7

7×7 input image

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

Stride=1

(*)

0	0	1
1	0	0
0	1	1

3×3 f=3

filter/kernel (feature Detector)

0 → white

1 - black

=

0	2			

5×5

Feature Map

Input image → array of pixel values
filter → set of weights
feature map → filter is systematically applied to the input data to create a feature map.

→ Multiple small filters can be designed to detect multiple features in the image

→ Values of the filters are weight to be learned during the training of the network. So that most useful features are extracted from the input image to classify it

→ CNN learn multiple factors in parallel it learn from 32 to 512 filters in parallel for a given input

→ Convolutional layers not only applied to input data (raw pixel values) but they can also be applied to the output of other layer

→ Stacking of convolutional layers allows a hierarchical decomposition of the input

Raw pixel (input image) → filter → Extract lines → filter → extract multiple lines to express shapes → filter → Extracting faces, animals, houses

→ If 3 filters have been used then feature map have depth of 3

→ The more numbers of filters the more accurate result

★ Stride:-

It is the number of pixels by which we slide our filters matrix over the input Matrix

Stride = 1 → move the filters one pixel at a time

Stride = 2 → Move the filters two pixel at a time

→ Layer stride will produce smaller feature maps

0 → White
1 → black

Let Stride = 1

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

⊗

0	0	1
1	0	0
0	1	1

L to R

f = 3×3 filter/kernel (feature Detector)

n = 7

7×7 input image

=

0	2	-	-	-
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-

5×5

Feature Map

0+0+0+0+0+0+0+0+0+0 = 0

0+0+1+1+0+0+0+0+0 = 2

Stride = 1

= n-f+1 ⇒ 7-3+1 = 5

→ Size of filter is smaller than input data.

→ Multiplication performed between a

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

T to B

filter size patch of the input data & the filter is a dot product, which is then summed & generate a single value.

→ The filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right top to bottom.

→ If the filter is designed to detect a specific type of feature in the input (image) then it discover that feature anywhere in the image

→ The output from multiplying the filter with the input array one time is a single value, as the filter is applied multiple times to the input array,

Weight Sharing:-

In CNN same weights are reuse to compute the outputs

In RNN, each neuron present in the hidden layer were have separate weights for itself.

Problem with Convolution:-

1) Shrinks output
2) Throwing away a lot of information that are in the edges

Solution

Pad the input image before convolution by adding some rows & column to it

1) Padding:-

→ In Convolution operation reduces the size of image. (Spatial dimensions decrease)

→ Losing some information:-

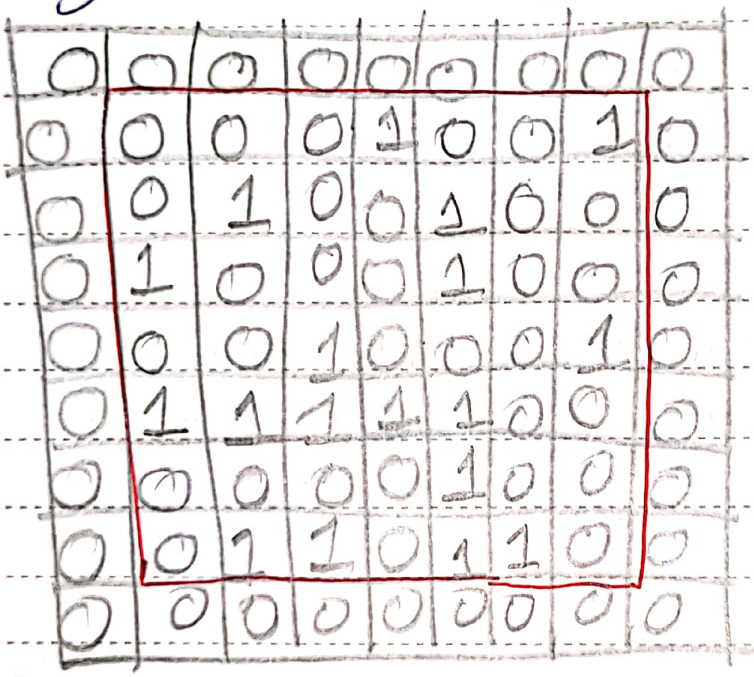
As we keep applying Convolution layers, the size of the volume (feature map) will decrease faster.

→ Zero padding allows us to control the

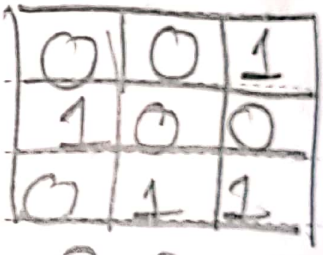
size of the feature maps.

→ Padding is used to make output size is same as the input size

→ P:= Padding amount = nos. of rows/columns that we will insert in top, bottom, left & right of the image



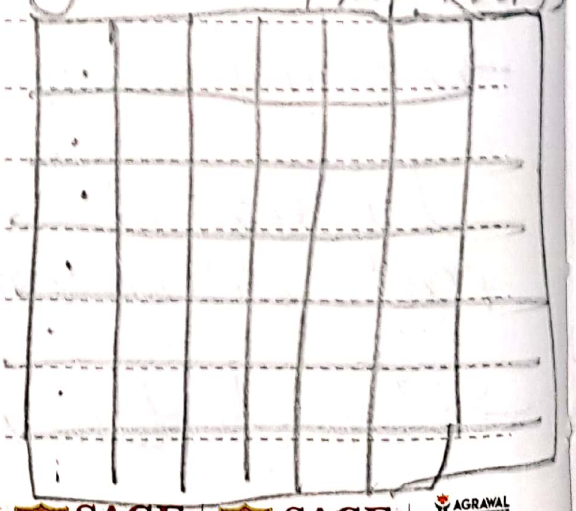
9×9



3×3 filter

f=3

9×9 Input with zero padding



7×7 (feature map)

Input = 7×7 (P=1)

n=7

n*n = Size of Input Image

f*f = filter

formula

$= n + 2p - f + 1$

$= 7 \times 2 \times 1 - 3 + 1$

$= 7$

Stride(S) :- It tells us the number of pixels we will jump when we are convolving filter.

$[n * n] * [f * f]$

input, filter

p → Padding

S → Stride

$$\left[\frac{n+2p-f}{S}+1\right] \times \left[\frac{n+2p-f}{S}+1\right]$$

feature Map

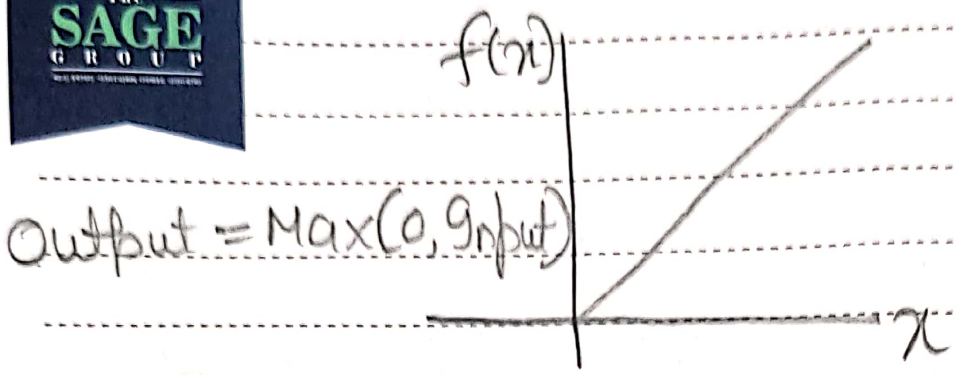
floor Value

→ The convolution operation shrinks the matrix if $f > 1$

2) ReLU layer

After every convolution operation ReLU operation has been used

ReLU → Rectified Linear unit is a non-linear activation function



$$f(x) = \text{Max}(0, x)$$

→ ReLU is an element-wise operation that means if applied per pixel & replace all negative pixel values in the feature map by zero

→ Convolution is a linear operation, non-linearity is introduced by using a non-linear activation function like ReLU

1	-2	8
-3	-1	1
1	1	-5

Feature Map

ReLU →

1	0	8
0	0	1
1	1	0

CNN uses Pooling:

Layers to reduce the size of inputs speedup computation & to make some of the features it detects more relevant

Step-2 Pooling (Spatial Pooling) Layer

Also called as subsampling or down sampling

→ It is applied after convolution & ReLU operation

→ It reduces the dimensionality of each feature map but retains the most important information

→ Since the number of hidden layers required to learn the complex relations present in the image would be large

→ We apply pooling operation to reduce the input feature representation thereby reducing the computational power required for the networks

→ Spatial pooling can be of different types

i) Max pooling ii) Average pooling
iii) Sum pooling

1) Max pooling / Sub Sampling

Once we obtain the feature map of the input, we will apply a filter of determined shape across the feature map to get the maximum value from that portion of the feature map.

→ It is also known as Sub sampling because from the entire portion of the feature map covered by filter/kernel we are sampling one single maximum value.

1	4	7	5
0	2	3	2
8	2	3	5
7	1	2	2

4×4 feature map

2×2 filter, stride = 2, P = 0

4	7
8	5

2×2

2) Average Pooling

It computes the average value of the feature value of the feature map covered by kernel/filter.

3) Sum pooling
It computes the sum of all elements in that window

Step-3 Flattening:-
After a series of convolution & pooling operation on the feature representation of the image. We will flatten the output of the final pooling layer into a 2D single long continuous linear array/vector.

2	1	1
0	3	4
5	2	7

Pooled feature map

Flattening →

2
1
1
0
3
4
5
2
7

The process of converting all the resultant 2D ~~ana~~ arrays into a vector is called flattening

Step-4: Fully Connected Layer (dense layer)

→ Flatten output is feed as input to the fully connected ANN.

→ ANN have varying number of hidden layers to learn the non-linear complexities present with the feature representation.

→ The aim of FC layer is to use high level features of input image produced by convolutional & pooling layer for classifying the input image into various classes based on the training dataset.

→ Fully connected means every neuron in the previous layer is connected to every neuron in the next layer.

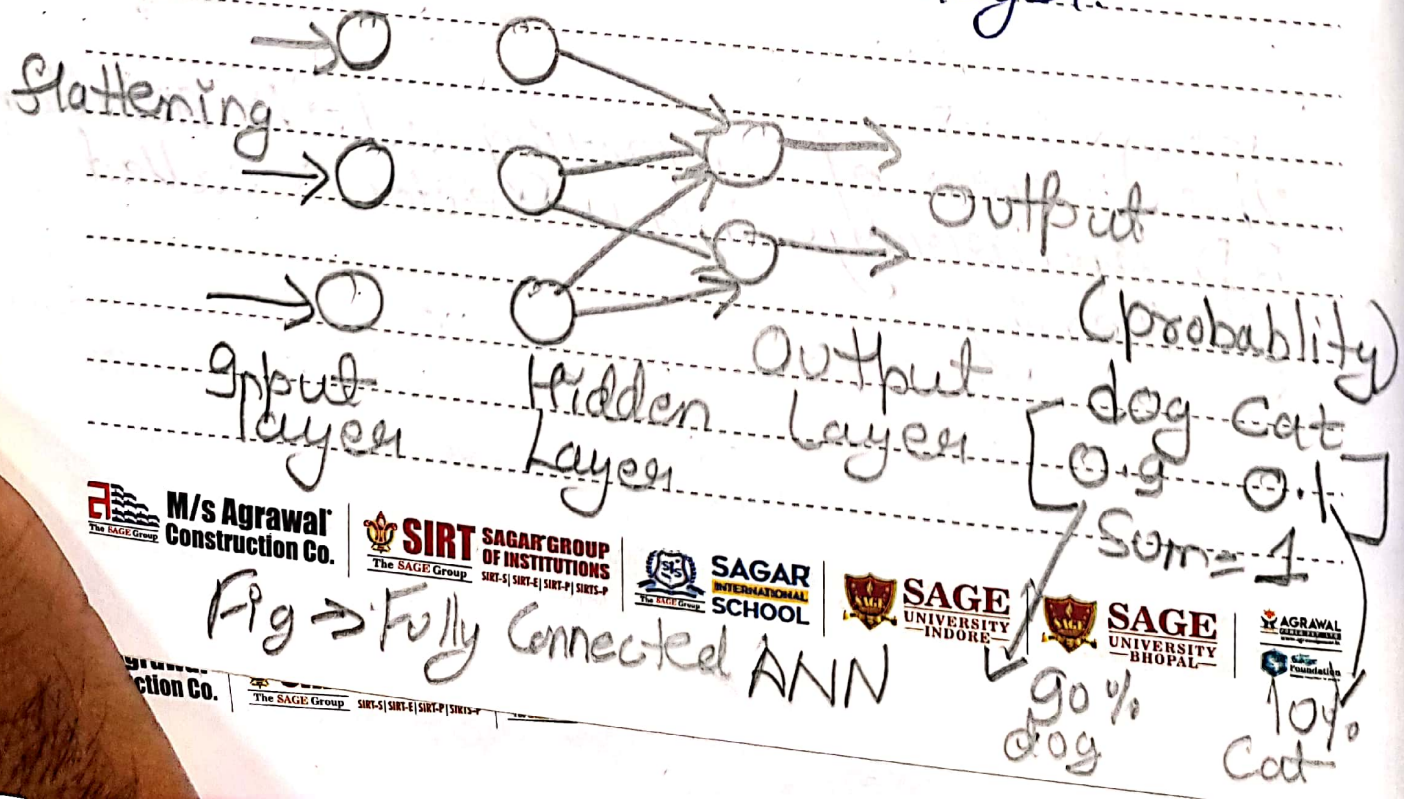


Fig → Fully Connected ANN

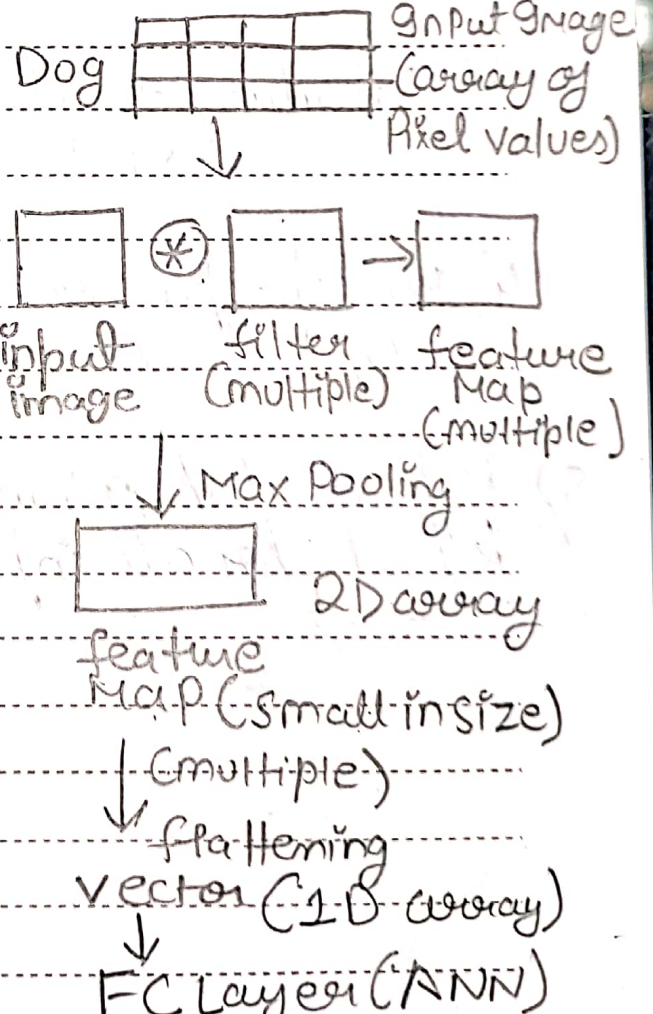
→ Sum of output probabilities from the FC layer is one.

→ Fully connected uses a softmax activation functions in the output layer

→ The softmax function takes a vector of arbitrary real-valued scores & transform it to a vector of values between zero & one that sums to one.

Working of Convolutional Neural Network.

Input Image
↓
Convolution (multiple filters) + ReLU
↓
feature Map
↓
Map Pooling (reduce the size of feature map)
↓
flattening
↓
Dense Layer / Fully Connected Layers
↓
Output Vector



Predicted [0.9 0.1]
Actual [1 0]

⇒ one filter creates one corresponding feature map.

Training of CNN

1) Initialize all filter weights with random values

2) Forward propagation (Input → Convolution → ReLU → Pooling → FC → o/p probability for each class)
(Input Image)

3) Calculate total error $\left\{\frac{1}{2}\sum(\text{target}_p - \text{output}_p)^2\right\}$ (SSE)

4) Back propagation (using gradient descent)
Update weights & parameters

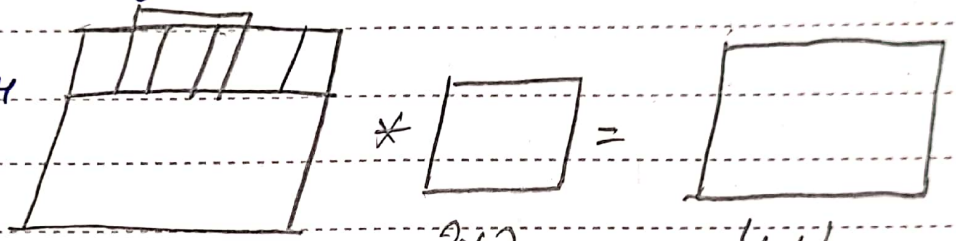
5) Repeat till minimum loss/error
Repeat for all input images (training set)

Advantages of Convolutional Neural Network

1) Parameter Sharing

A feature detector that is useful in one part of the image may also be useful on another part of the same image

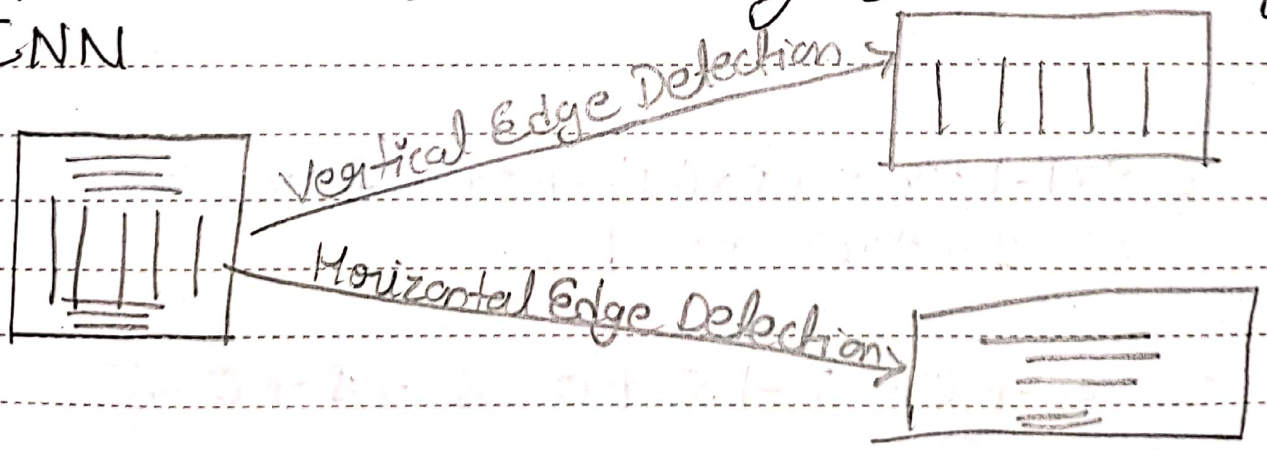
Less parameter Compared to FFNN (feed forward Neural Network



2) Sparsecity of Connections

In each layer, output depends only on small number of input.

Vertical & Horizontal Edge Detection using CNN



1) Vertical Edge Detection

Ex-1

5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0

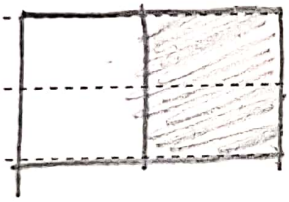
Input Image

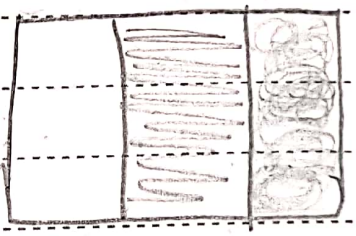
6×6

*

1	0	-1
1	0	-1
1	0	-1

3×3 filter





=

0	15	15	0
0	15	15	0
0	15	15	0
0	15	15	0

4×4 feature Map

$= 5\times1+5\times0+5\times(-1)+5\times1+5\times0+5\times-1+5\times1+5\times0+5\times-1$

$= 5+0-5+5+0-5+5+0-5$

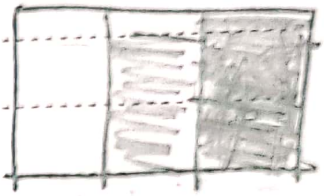
$= 0$

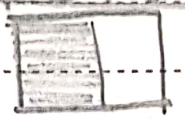
Ex-2

0	0	0	5	5	5
0	0	0	5	5	5
0	0	0	5	5	5
0	0	0	5	5	5
0	0	0	5	5	5
0	0	0	5	5	5

*

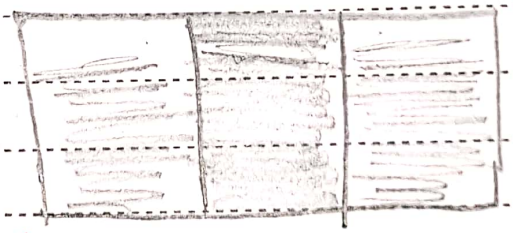
1	0	-1
1	0	-1
1	0	-1





=

0	-15	-15	0
0	-15	-15	0
0	-15	-15	0
0	-15	-15	0



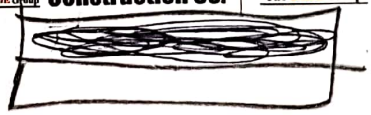
2) Horizontal Edge Detection

5	5	5	5	5	5
5	5	5	5	5	5
5	5	5	5	5	5
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

*

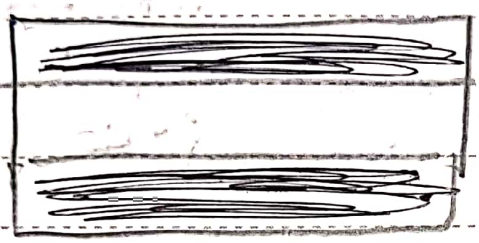
1	1	1
0	0	0
-1	-1	-1

=



0	0	0	0
15	15	15	15
15	15	15	15
0	0	0	0

=

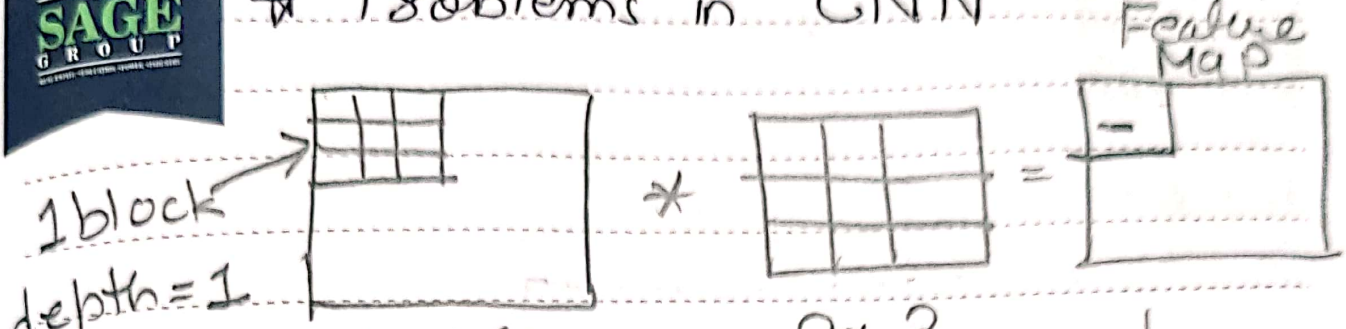


$= 5\times1+5\times1+5\times1+5\times0+5\times0+5\times0+5\times-1+5\times-1+5\times-1$

$= 5+5+5+0+0+0-5-5-5$

$= 0$

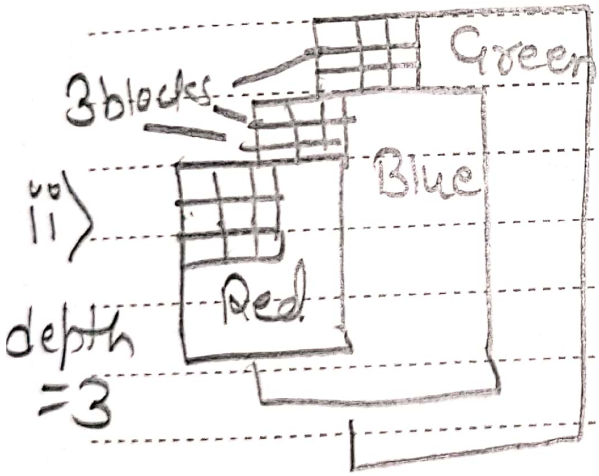
Problems in CNN



i) Input image with one Channel (gray scale image)

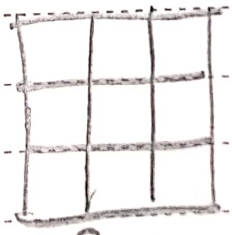
3*3 filter

3x3 filter applied in 3x3x1 blocks (9) to create a single value



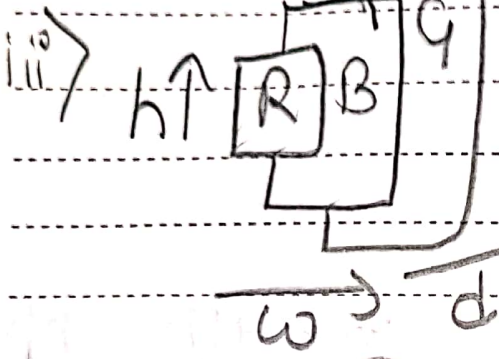
ii) Color image

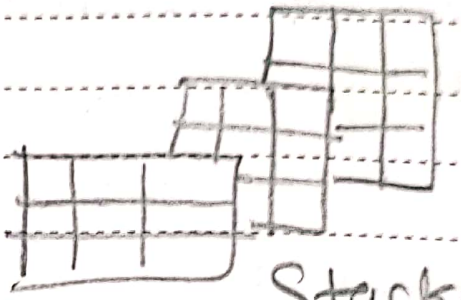
[3 channels for red, blue, green]



= 3x3 filter will be applied in 3x3x3 blocks (27) to create a single value

3 Matrix

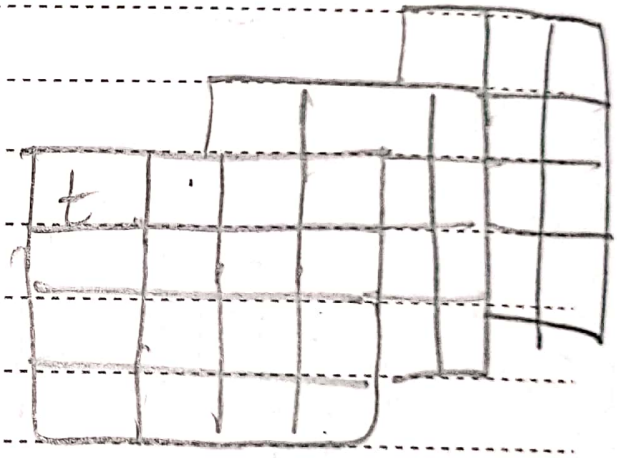




Stacked filters for each column

6 × 6 × (3) → channel

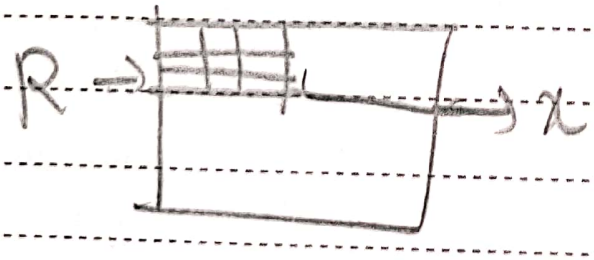
image

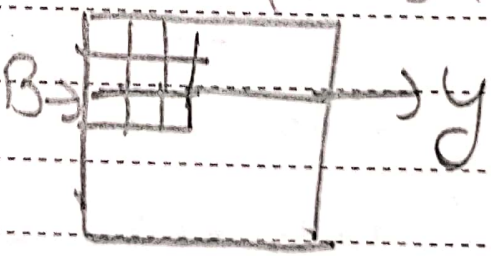


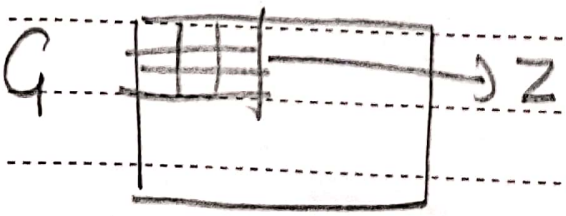
4 × 4 × # filters

3 × 3 – R
3 × 3 – G
3 × 3 – B
27 parameters

$\begin{cases} n-f+1 \\ 6-3+1=4 \end{cases}$







$t = x + y + z$

Convolutions over 3D image

1x1 Convolution (Network in Network)

Problems:

Suppose depth = 64, then 3x3 filter will be applied in 3x3x64 blocks to create a single value to make up the single output feature map.

→ When the depth of the network increases the depth of input or number of filters used in convolutional layer often increases resulting ~~to~~ an increase in the number of resulting feature map.

→ A large number of feature maps in a CNN can cause a problem as a convolutional operation must be performed down through the depth of the input.

→ If the size of filter is relatively large like 5x5 or 7x7 pixels, it can result in considerably more parameters (weights) which require large computation to perform the convolutional operations.

→ Convolution & pooling layers reduce the height & width of feature map.

leaving the depth, number of channels, or number of filters, (Nos. of feature Maps) unchanged.

→ So deep CNNs require a layer that can down sample or reduce the depth or number of feature maps

Solution

→ Use a 1×1 filter to down sample the depth or number of feature maps

→ 1×1 convolution can be used to reduce only depth as it slices through the input volume with just 1×1 unit

→ It saves on computation task

→ It adds non-linearity

→ It allows to learn more complex functions

1	2	3
2	1	1
3	2	1

3×3×1

3

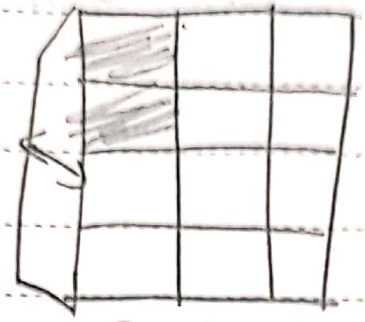
1×1

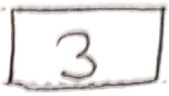
Feature Map

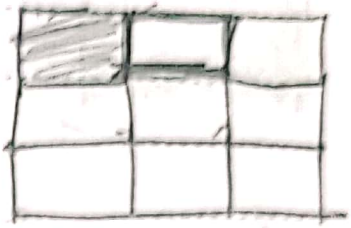
3	6	9
6	3	3
9	6	3

3×3×#

#Hers







ReLU

→ It generates an element wise ~~product~~ product across the depth, slices through the entire volume

→ Useful when layer filter sizes are used, such as 5×5, 7×7

→ It is used for dimensionality reduction

→ It is so useful in many CNN models

6×6×32 (32 ↓ channels) * 1×1×32 (#filters=5) → 6×6×5 (5 ↓ Channels)

Input — output

→ It has been used in a lot of modern CNN implementations like R.

→ It is useful when we want to shrink the number of channels called as feature

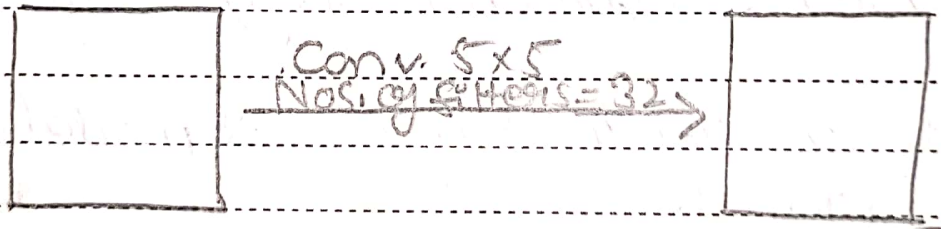
transformation. This save a lot of computations.

Inception Network

→ During design of a CNN, you have to decide all the layers yourself.

Choices —
- 3×3 Conv.
- 5×5 Conv.
- Max pooling layer

5×5 Convolution

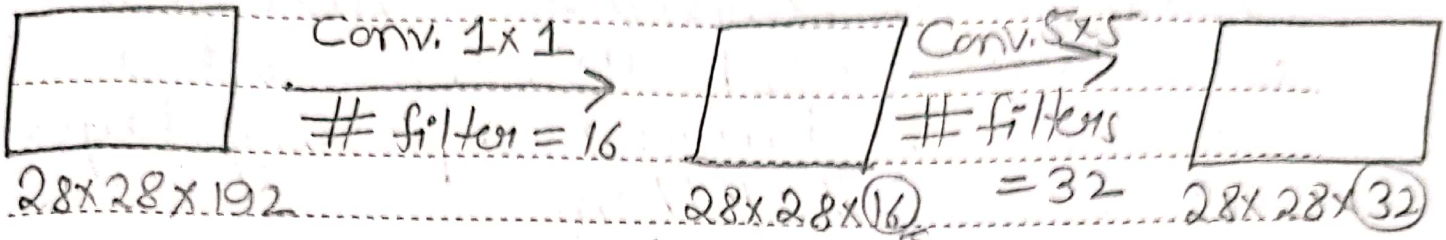


28×28×192
Input Image

28×28×32

Computation Cost = 28×28×5×5×192×32
(Nos. of multiplication needed) = 120M (High computation Cost)

* Using a 1x1 convolution with 5x5 Convolution → to reduce computational cost



Computation Cost = (28x28x16) x (1x1x192) + (28x28x32) x (5x5x16)

= 12.4M (approx)

Hence, 12.4M is immensely smaller than 120M
It means, 1x1 Convolution can help to reduce model size which in then also help to reduce overfitting problem

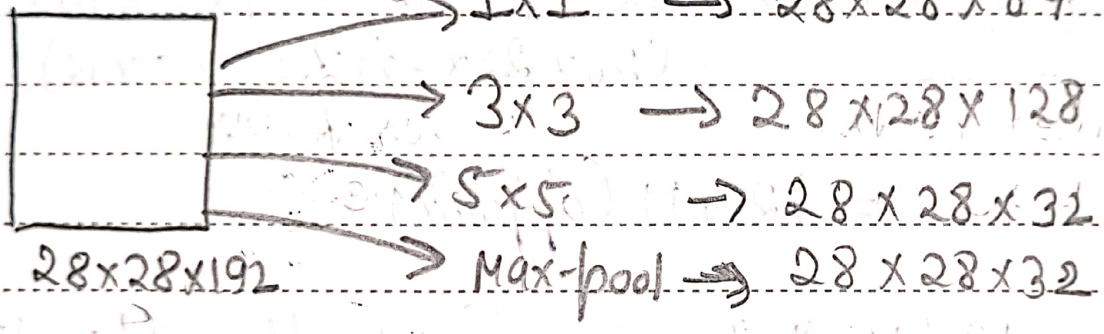
Inception Network

→ Instead of choosing a 1x1, 3x3 or 5x5 filter or a pooling layer we apply all together

→ This makes the network architecture more complicated but remarkably improves performance as well.

→ The Inception network or an inception layer says instead of choosing the desired filter size in a Conv. layer or whether we want a Convolutional layer or a pooling layer let's apply all of them.

* Inception Module, Navie Version:-



$= 64 + 128 + 32 + 32$

$= 256$

$28 \times 28 \times 192 \longrightarrow 28 \times 28 \times 256$

(Input) Channel (Output)

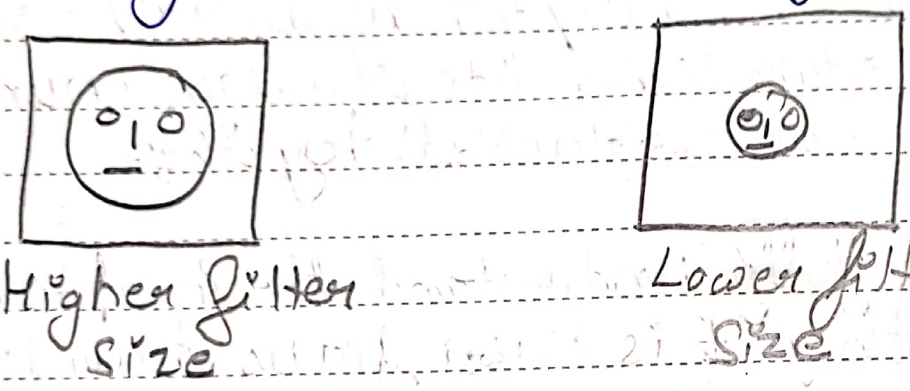
High Computational Cost

→ Here we used all the Convs & pools we might want & will let the neural network learn & decide which it want to use most

→ Inception layer allows the internal layers

to pick & choose which filter size will be relevant to learn the required information.

→ So even if the size of the face is in the image is different, the layer works according to recognize the face.



* Inception Module, dimensions reduction Version:-

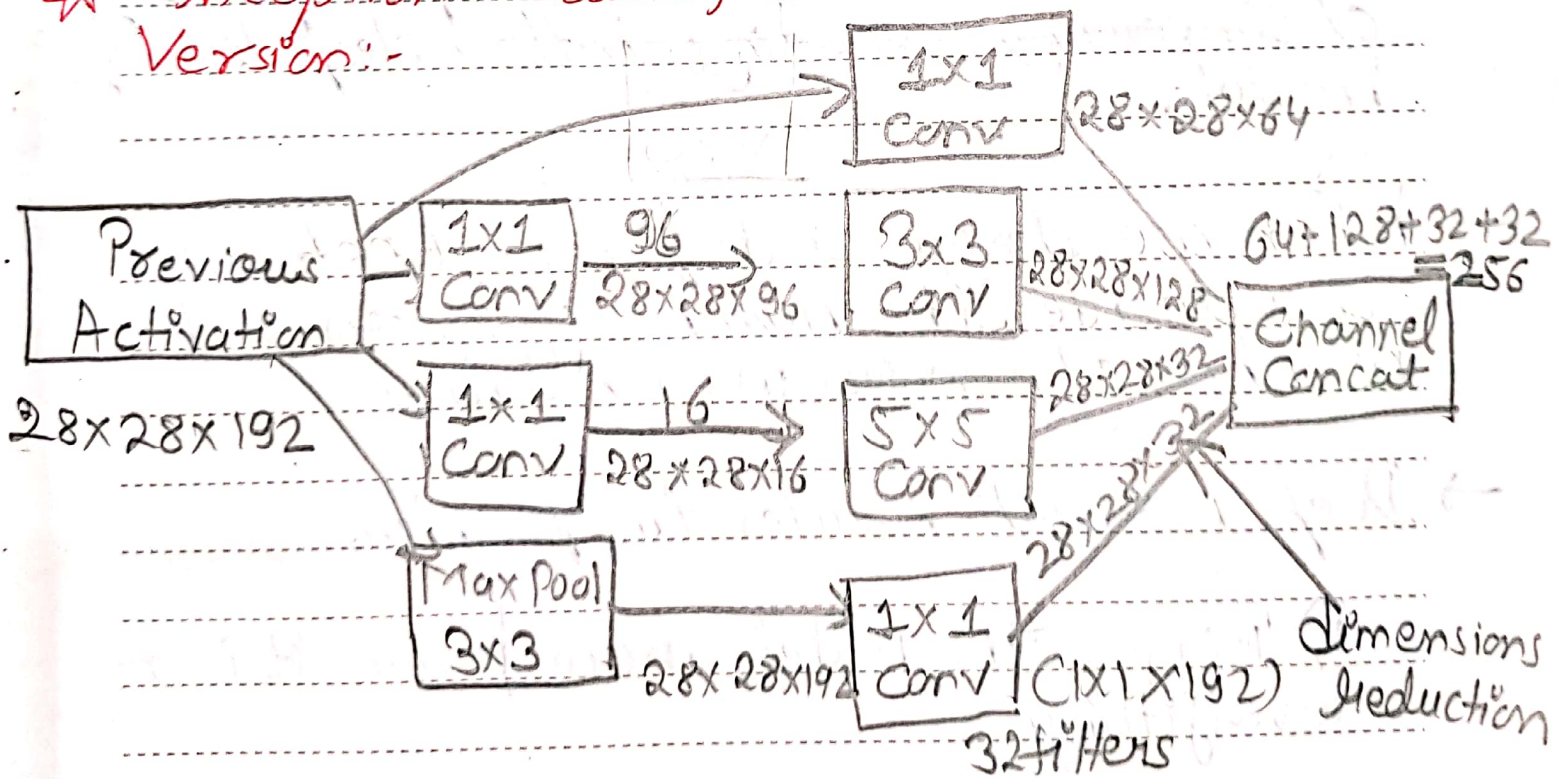


Fig → Inception module example

→ The inception module takes ~~the~~ an input the activation or the output from some previous layers.

→ Let 28 x 28 x 192 is the volume of previous activation

→ This was one inception module. The overall inception network consists of a layer number of such modules stacked together.

→ When we fully understand the inception module, then it is easy for us to understand the whole inception networks.

→ Inception network provides a powerful way of ~~converting~~ creating better deep learning models.

→ When we design a deep neural network, we decide the number of layers or the number of nervous in each layer

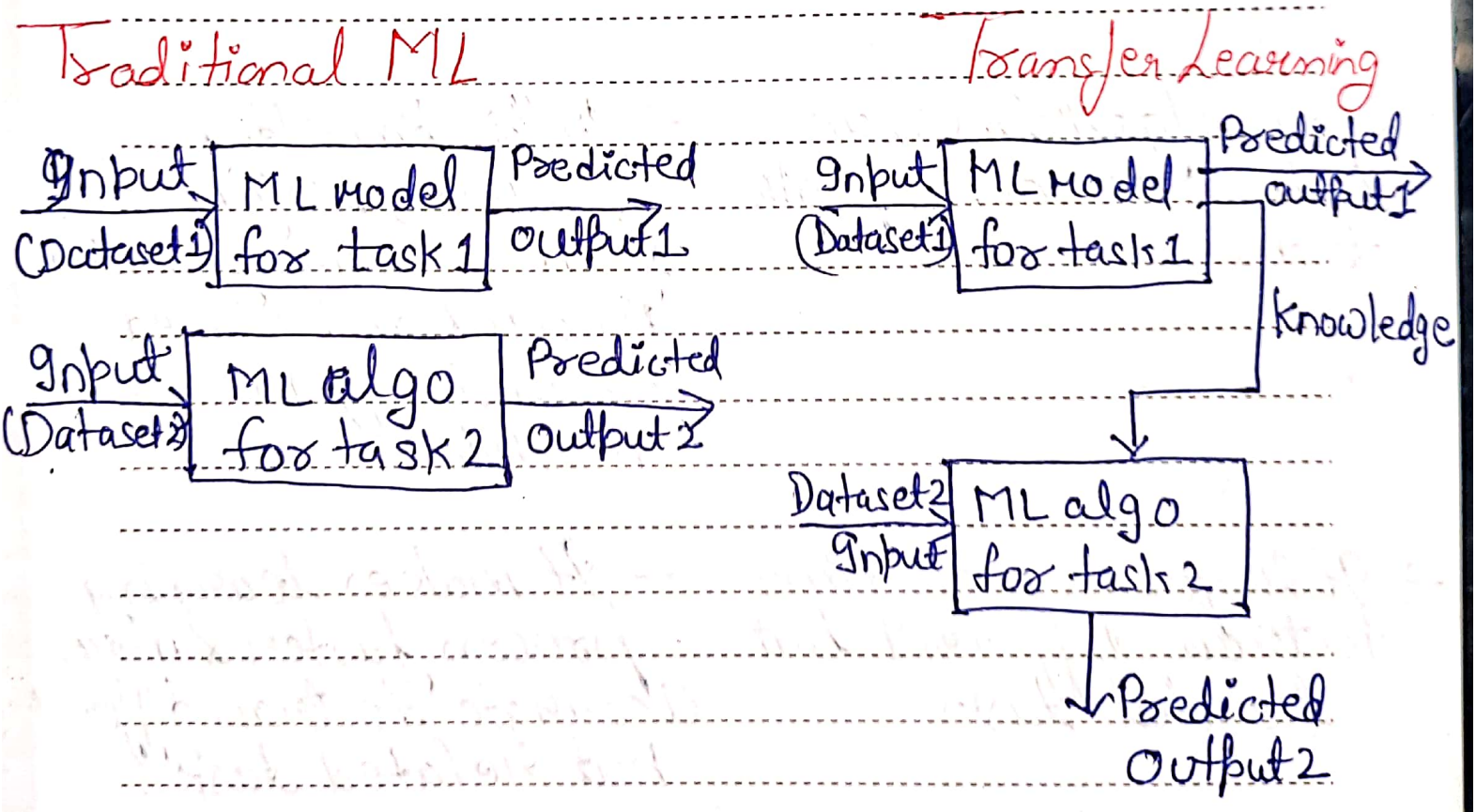
→ Deep network creates two problem

i) Bigger the model, more prone it is to overfitting.

ii) Increasing the number of parameters Means, we need to increase existing computational resources.

→ Solution is to use inception network.
→ This maintain the computational budget while increasing the depth and width of the network

Transfer Learning & Learning from other knowledge ?



→ It is a single task learning for every different related task, the models have to be rebuilt from scratch

→ Learning of new tasks relies on the previous tasks.

→ Conventional ML & deep learning algorithm have been traditionally designed to work in isolation

→ In this, algorithms not work in isolation

→ These algorithms are trained to solve specific tasks

→ In this, knowledge (features, weights etc). from previously trained model can be used for training newer models

→ It requires more time to train different but related tasks.

→ It makes learning process faster & more accurate to train different but related tasks.

→ It requires more training data

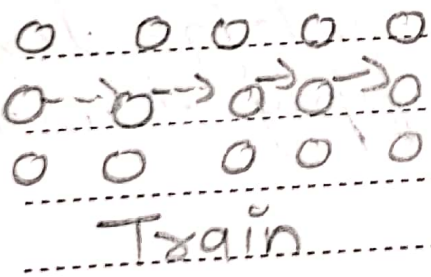
→ It requires less training data. It uses already existing labelled data of some related task or domain

⇒ In traditional learning, is isolated & purely based on specific tasks, dataset & training separate isolated models on them. No knowledge is retained which can be transferred from one model to another

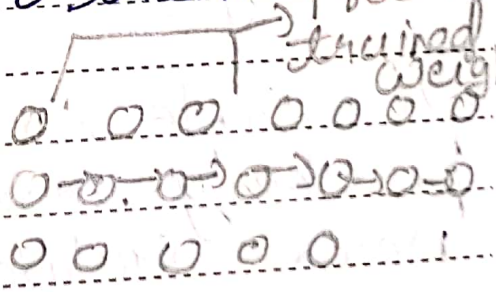
→ In transfer learning, knowledge is retained from previously trained model & use for training ~~models~~ newly models. It also solve the problem of having less data for the newer task.

→ It enable us to utilize knowledge from previously learned task & apply them to newer related ones.

Training from Scratch	Transfer learning
→ Build CNN from scratch	→ only last few layers need to be trained
→ Need to tune large number of hyper parameter	→ Only a few parameters need to be tuned
→ Large computation ~~process~~ power is required	→ less computation power needed
→ Huge dataset needed to avoid overfitting	→ A small dataset is enough
→ ~~May~~ May take weeks or months.	→ May take hours to train



Train



* Transfer learning

→ It is one of the most important design technique of deep learning that work with images

or object detection.

→ During the process of transfer learning we need to decide

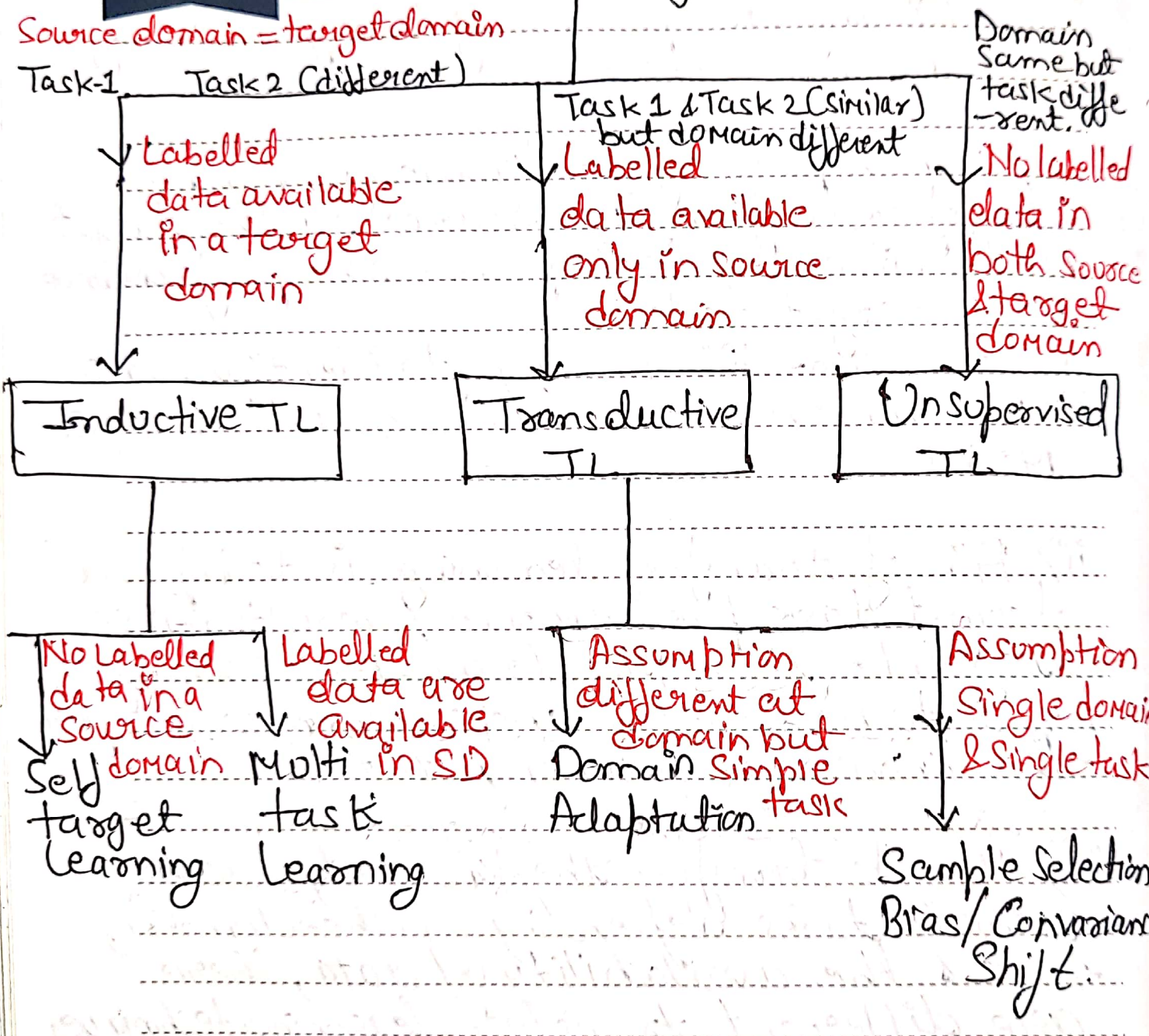
i) Which part of the knowledge can be transferred from the source to the target model in order to improve the performance of the target task.
For this we try to identify what is common between the source & target model.

ii) Aim of transfer learning is to improve target task performance / results & not degrade them. To achieve this we need to be careful about when to transfer the knowledge & when not to.

iii) How to transfer the knowledge across domain / tasks depending on the domain, task & the availability of data, there are different transfer learning techniques are used

P.T.O.

Transfer Learning Strategies



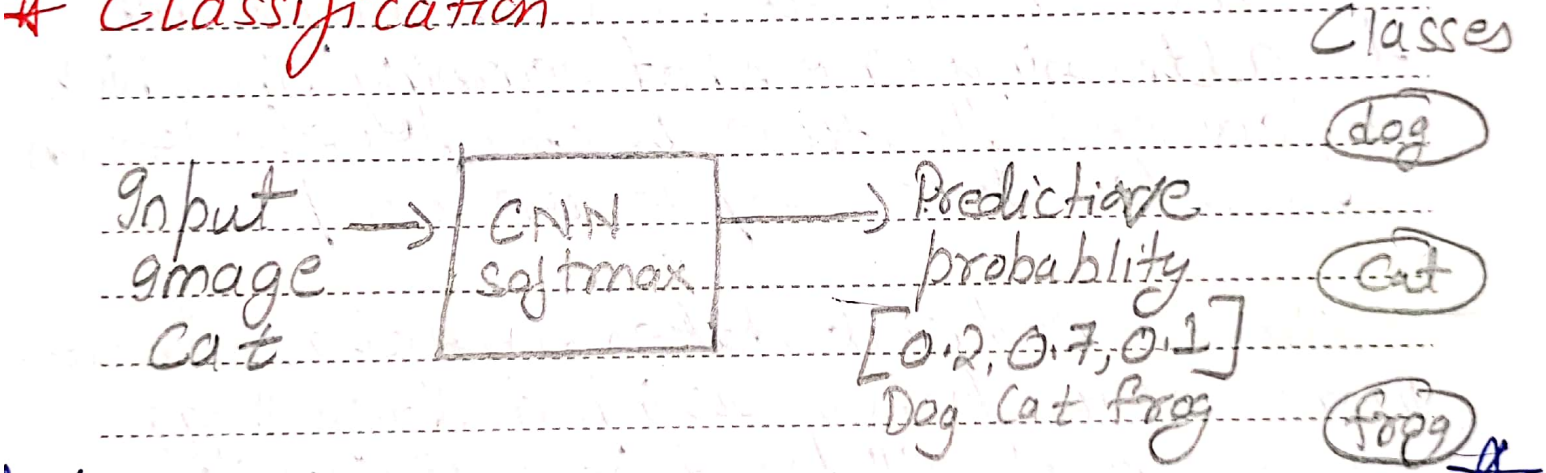
One-Shot learning

→ Deep CNNs are used for image classification tasks, but they require a lot of labelled data for training.

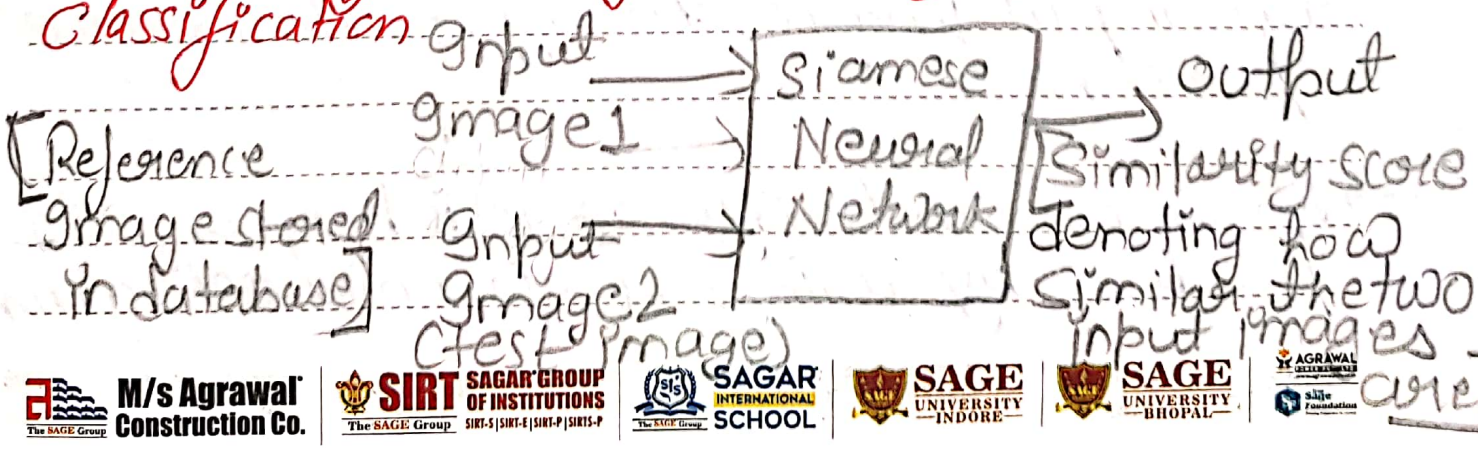
→ In many applications, collecting large amount of labelled data is sometimes not possible.

→ To solve this problem, one shot learning is used.

Classification



Face recognition system using one shot classification



→ One shot learning is a classificati. task where one or a few examples are used to classify many new examples in the future

→ It is used in face recognition such as face identification & face verification, when people must be classified correctly with different facial expressions lighting conditions, accessories & hairstyles given one or a few template photos.

→ Siamese network are an approach to addressing one shot learning in which a learned feature vector for the known & candidate example are compared.

→ Siamese network takes an extra reference image of the person as input & test image of person & will produce a similarity score denoting the chances that the two input images belong to the source person

→ Range of similarity score is between 0 & 1 produce by using a sigmoid activation

function

0 → No similarity

1 → full similarity

b/w 0 & 1 → Some Similarity

→ For training of this network only few training examples are enough to build a good model

→ For face recognition of newly formed employee only single image of his face will be stored in the database

Using this, as reference image, the network will calculate the similarity for any new image presented to it for same person.

→ That means the network predicts the similarity in one-shot.

---

* Classification in 1-shot (continue)

→ In standard classification, the input image is fed to the CNN & it produce output in probablity distribution over all the classes using a softmax activation function (above example, Nos of classes = 3)

→ For ex, if we are trying to classify an image ~~or~~ cat or dog or frog then for every input image, CNN generate 3 probablities, indicating the probability of the image belonging to each of the 3 classes

Hence, i) During the training process, we require a large nos. of images for each of the class. (dog, cats, frog).

ii) If the network is trained only on above 3 classes, then we cannot expect to test it on any other classes (ex → Mouse).

If we want our model to classify the image of mouse, also then we need to first get a lot of mouse images & then we must re ~~train~~ train the model again

→ But there are applications wherein

i) No enough data is available for each class

ii) Total nos. of classes are huge. &

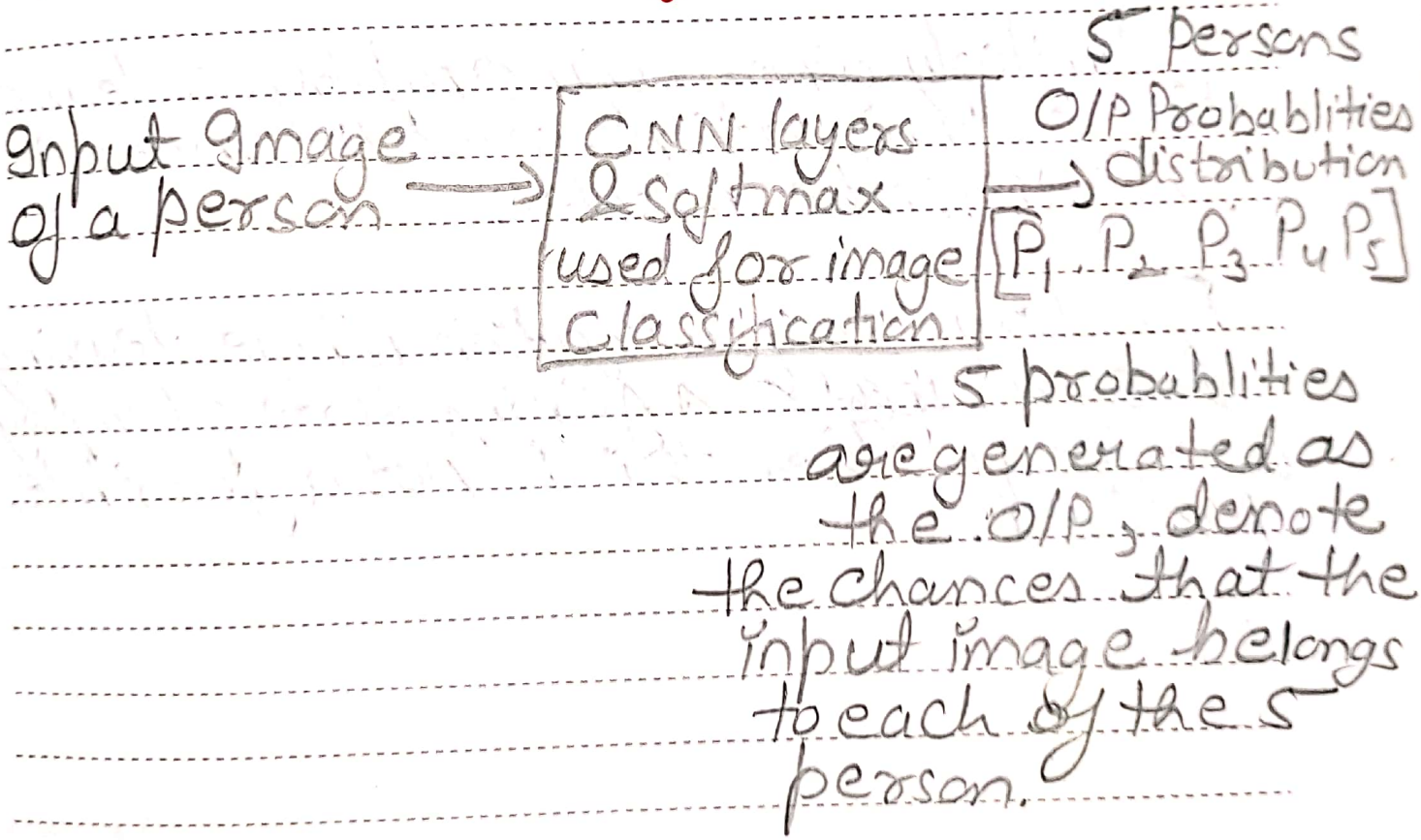
dynamically changing

Therefore, the cost of data collection & periodical retraining is too high.

One shot classification

→ Here we require only one training example for each class. Hence the name one shot

* Face recognition system using a traditional classification approach



☆ Problems in this approach are as follows:-

1) For training, we require lots of different images of each person. If there are thousands of employees in large organization then it might not be feasible.

2) If a new person joins the organization, we need to collect data again & re-train the model again.

Similarly when any employee leave the organization, we also need to retrain the model.

For large organization where leaving & joining rate of employee are large then this is not a feasible Method.