Digital Sreeni Channel

U-Net

⇒ studied about segmentation problem using deep learning.

⇒ image segmentation using an architecture called U-Net.

architecture ⇓ arrangement of deep learning tools conv., max polling, take these tools and arrange in such a way that the result would be image segmentation.

UNet architecture is designed for Semantic segmentation.

Object detection:- bounding box create for each of its individual object.

~~ima~~ Semantic Segmentation..
paints the pixels corresponding to human in this image is semantic segmentation
(Same color of ~~each~~ same individual)

Instance Segmentation:-
Seperate each individual (extention of semantic segmentation)

Padding = same :- The output is same size. $p = \frac{k-1}{2}$

"valid" :- The filter window stays inside the image.

"full padding"

$p = k-1$ (kernel size)

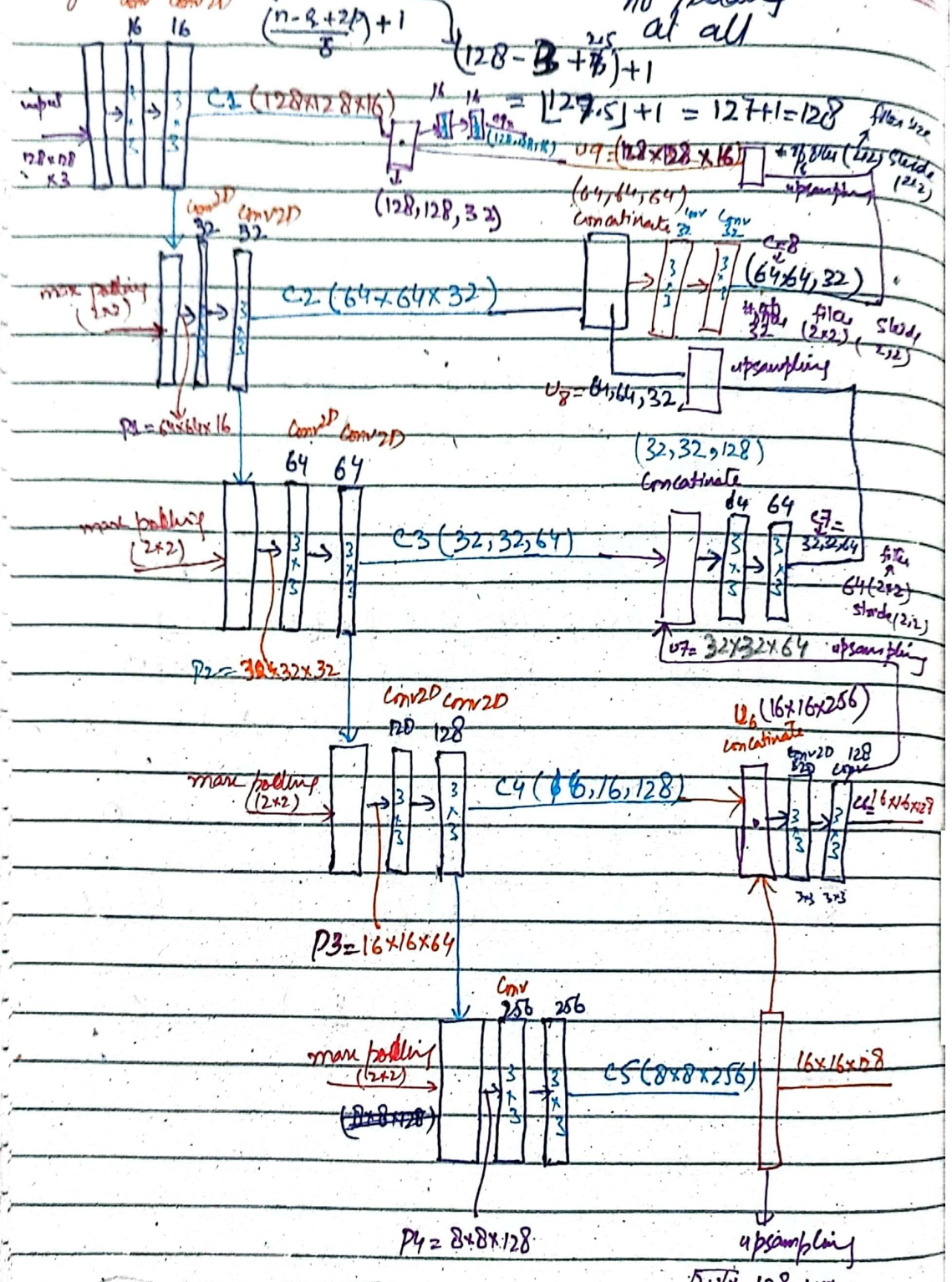
$(128 - B + 2p) + 1$

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

$(128 - B + 2p) + 1$

no padding at all

$= \lfloor 127.5 \rfloor + 1 = 127 + 1 = 128$



Padding = same

input dimension × stride

8×2

$= 16$

$(I - 1) \times s_h + k_h - 2p$

$(8 - 1) \times 2 + 2 - 2(0)$

$7 \times 2 + 2$

(16)

$= I/2$ (output dim)

kernel size

# of dimension $= \left\lfloor \frac{i - k + 2p}{s} \right\rfloor + 1$

padding:
$P(\text{same}) = k/2$
$P(\text{valid}) = 0$
~~$P(\text{full padding}) = k-1$~~

input	output dimension	parameters
input		
128×128×3	128×128×3	0
Conv 2D (3×3)×16 padding = same ~~dropout (0.1)~~	$\left\lfloor \frac{128 - 3 + 2(\frac{3}{2})}{1} \right\rfloor + 1$ $\lfloor 127.5 \rfloor + 1 = (128, 128, 16)$	$3 \times 3 \times 3 \times 16 + 16 = 448$
Conv 2D (3×3)×16 padding = same	$c_1 = (128, 128, 16)$	448
dropout (0.1)	"	0
max pooling (2×2)	$\left(\frac{128 - 2 + 2(0)}{2}\right) + 1 = (64, 64, 16)$	0
Conv 2D (3×3), 32 ~~dropout (0.2)~~	$\left\lfloor \frac{64 - 3 + 2(\frac{3}{2})}{1} \right\rfloor + 1$ $\lfloor 63.5 \rfloor + 1 = (64, 64, 32)$	$3 \times 3 \times 16 \times 32 + 32$ 4640
Dropout (0.2)	(64, 64, 32)	0
Conv 2D (3,3), 32	(64, 64, 32)	4640
Max pooling (2×2)	$c_2 = (32, 32, 32)$	0

Keras

Compile method

configure the model for training

model.compile
(

optimizer: 'rmsprop', 'adagrad', 'Adadelta', 'adam', 'Adamax', 'Nadam'

loss: loss function

loss_weight: (optional) or dictionary (list)
→ weight the loss contribution of different model output
→ loss value that will be minimized by the model will then be weighted sum of all individual losses.

metrics:- list of metrics to be evaluated by the model during training and testing

metrics (list): ['accuracy']

metrics (dict): { 'a': 'accuracy',
b: ['accuracy', 'mse'] }

metrics (list of metrics): [['accuracy'], ['accuracy', 'mse']]

weighted_metrics: list of metrics to be evaluated during training and testing

run_eagerly: bool
True ⇒ model forward pass will never be compiled

C5 = (8, 8, 256)

Transpose (2 0)
(128) (2, 2)
Slide (2, 2)

$2 \times 2 \times 256 \times 128 + 128$
$= 131200$

Callbacks API

is an object that perform actions at various stages of training (at the start or end of an epoch, before or after a single batch etc)

* ModelCheckPoint

Keras.callbacks.ModelCheckPoint(

→ filepath :
→ monitor : metric name to monitor
* loss
* val_loss
→ verbose
→ save_best_only : bool
→ mode : ["min", "max", "auto"]
→ save_weights_only : True
→ Save_freq : "epoch"
save for each epoch
→ initial_value_threshold :
best value of the metric to be monitored.

* TensorBoard

Enable visualization for tensorboard.

* Matrics summary plot
* training graph visualization
* weight histogram
* Sample profiling

recommend
~~False~~ False during learning
True " debugging

fit method

train the model for fixed number of epochs.

x : learning data
y : " " target
batch-size :
epochs :
verbose : show
auto, 0, 1, 2 → one line per epoch
auto → 1
0 → silent
1 → progress bar
call-back :
validation-split :
validation-data :
shuffle

Image Classification:-

→ classify an image to a specific class.
→ The whole image represents one class.
→ we don't need to know exactly where are the object.

Classification with Localization:-

⇒ Given an image we need to learn
↳ class of the image.
↳ Where are the class location in the image.
→ usually only one object is represented

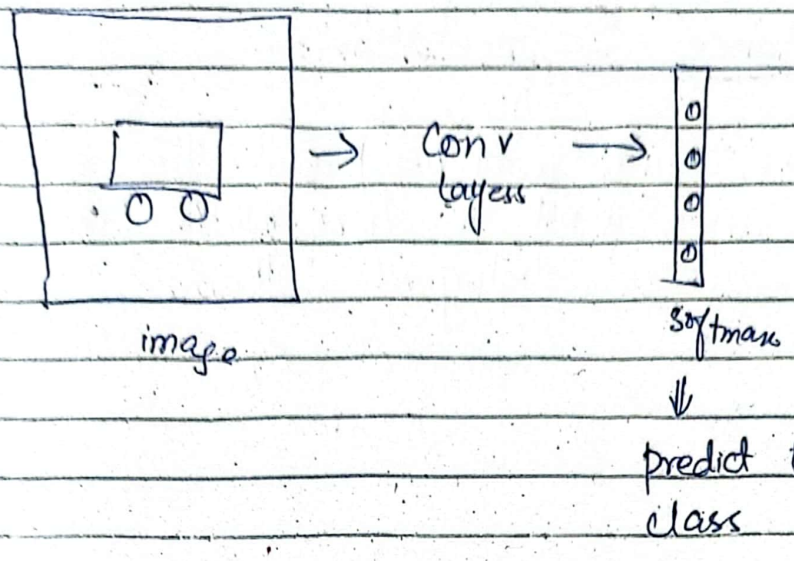
Object Detection:-

⇒ Given an image we want to detect
↳ all the object in the image that belong to a specific class
↳ and give their location.
⇒ An image can contain more than one object with different classes.

Semantic Segmentation:-

→ label each pixel in the image with a category level.
⇒ Don't differentiate instances, only care about pixels.
⇒ it detect no object just pixels.

Image classification :-

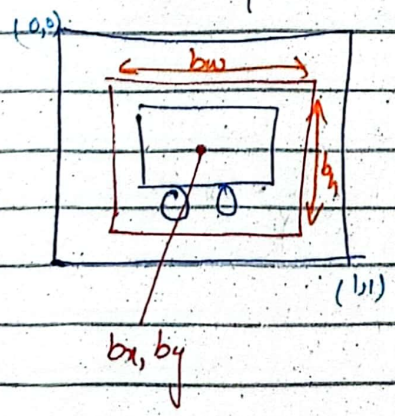


you are building self driving car may be your object categories are the following

1 - Pedestrian
2 - Car
3 - Motorcycle
4 - Background (none of above)

} 4 possible output

How to localize the car in the image as well
⇒ you can change your neural n/w to have a few more output unit. that output a bounding box



so now if your training set contains not just object class labels but also contain four addition number where is the bounding box, the object is detected.

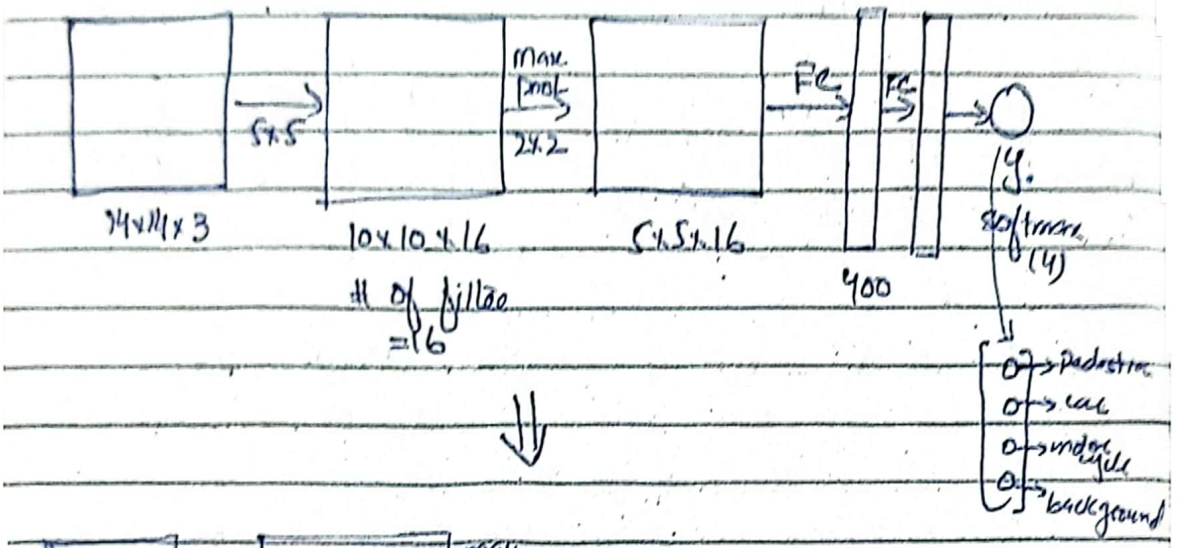
Instance Segmentation :-

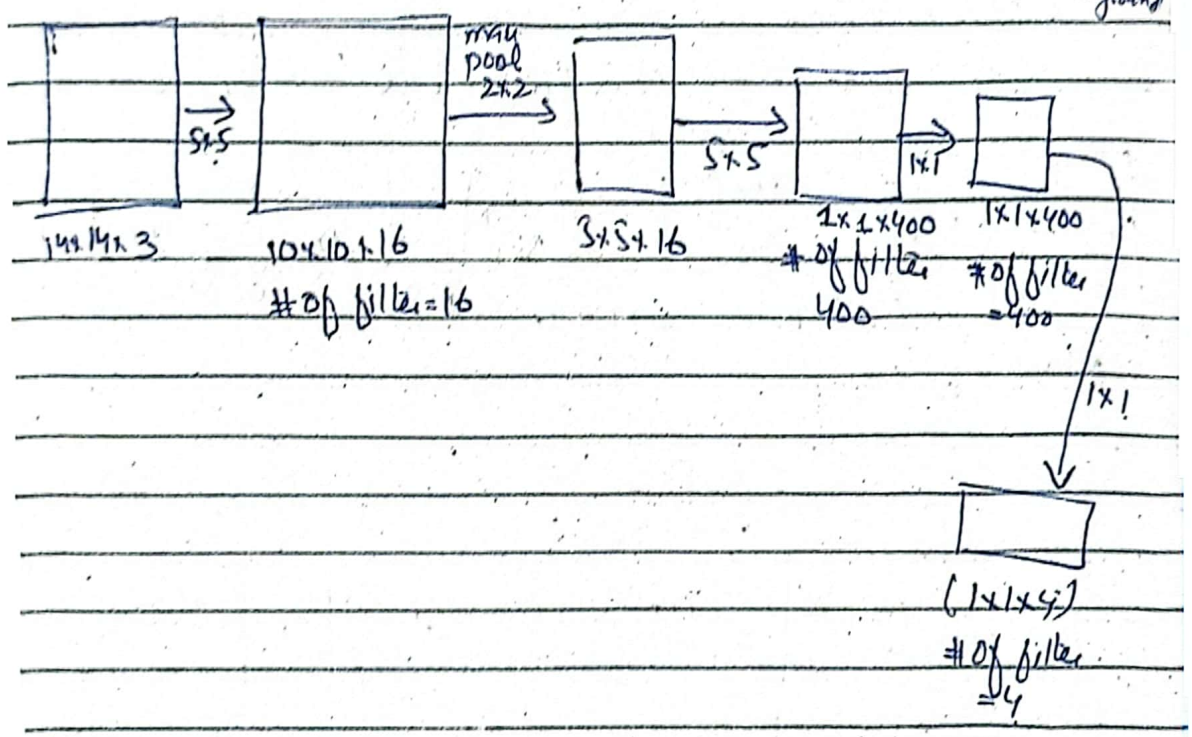
Rather than we want to predict the bounding box, we want to know which pixel label but also distinguish them..

CamScanner

Convolution implementation of sliding windows

Turning Fully Connected into Convolutional layers





In practice

Logistic regression loss ⇒ p_c
Log likely hood ⇒ classes (softmax)
squared error ⇒ bounding box

Object Detection

using ConvNet
⇓
Sliding window
Detection Algorithm

choose a sliding window with some stride and feed to an image
⇒ each sliding window detect is there is an object or not.
⇒ store the sliding window that contain object.

⇒ if two or more sliding window intersects, choose the rectangle with the best accuracy.

⇒ Pick larger/smaller sliding window and repeat the process 2 to 3.

↳ used linear regression for classification then used sliding window technique
↳ linear regression is cheap b/w in deep learning it is expensive due to complexity of deep learning model.
↳ to solve this problem, implement sliding window with a Convolutional approach.

$b_x = 0.5$
$b_y = 0.7$
$b_h = 0.3$
$b_w = 0.4$

Defining the target label "y"

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

is there an objects "1" or "0"
↓ object ↓ not object
Probality of object detected

if object is ~~detected~~ in image

$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

if object is not ~~detected~~ in image

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

→ don't care

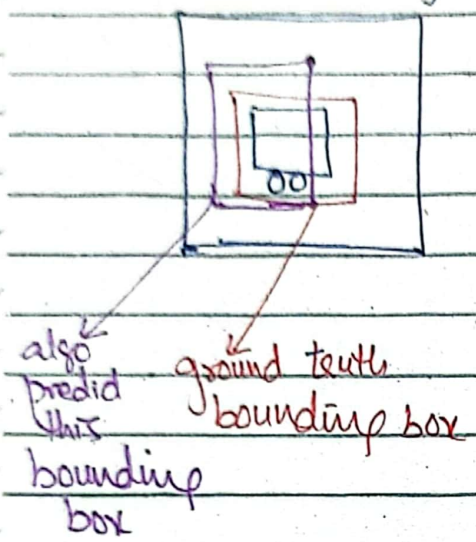
Loss function :-

$$\mathcal{L}(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + (\hat{y}_8 - y_8)^2 & \Longrightarrow \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \Longrightarrow \text{if } y_1 = 0 \end{cases}$$

↳ because all of rest are don't care.

Intersection over union (IOU)

In object detection task, you expected to localize the object as well.



IOU

it computes the intersection over union of these two bounding boxes.

$$= \frac{A \cap B}{A \cup B}$$

if ground truth and the predicted are overlap perfectly then IOU is 1.

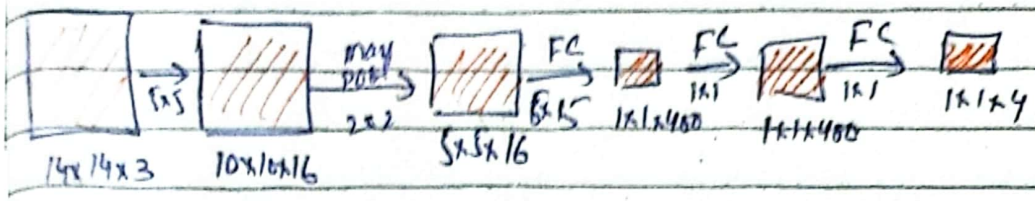
If IOU ≥ 0.5 pretty decent. $\Rightarrow$ correct

More generally, IOU is a measure of the overlap b/w two bounding boxes.

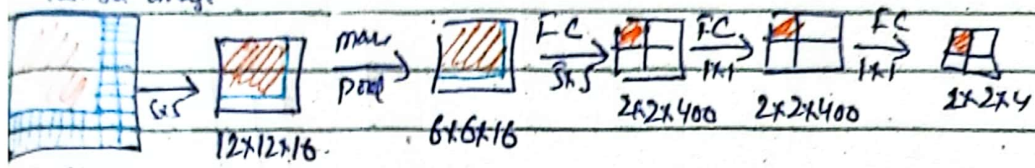
The higher the IOU, the better the accuracy.

Sliding window is (14x14)

Training

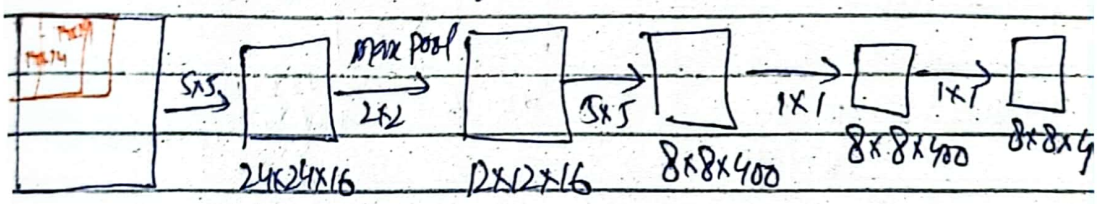


Test set image



So the original sliding window algorithm, you might (14x14x3) then slide 2 and so on you run 4 times ~~to get~~

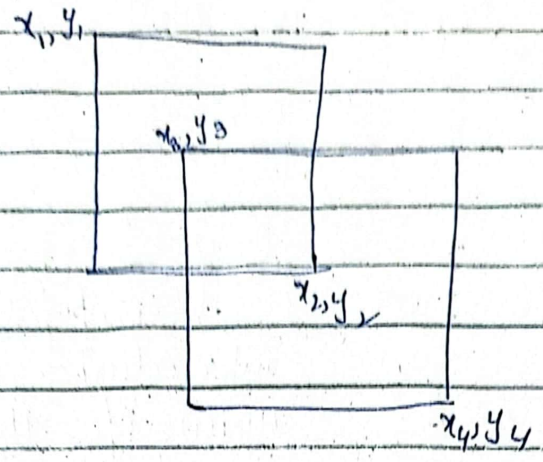
⇒ It is more efficient because it now shares the computation of the four times needed



Problem :-

Position of bounding box is not too accurate.

Consider the following boxes
(IOU) Intersection over union



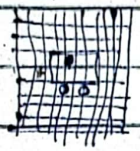
$x_1, y_1 = 100, 150$
$x_2, y_2 = 250, 300$
$x_3, y_3 = 130, 170$
$x_4, y_4 = 270, 360$

$$IOU = \frac{A \cap B}{A \cup B}$$

Non-Max Suppression :-

One of the problem in the object detection is
your algo may find multiple detection of the same object.

Non max suppression ⇒ is to make sure your algo predict the object only once.



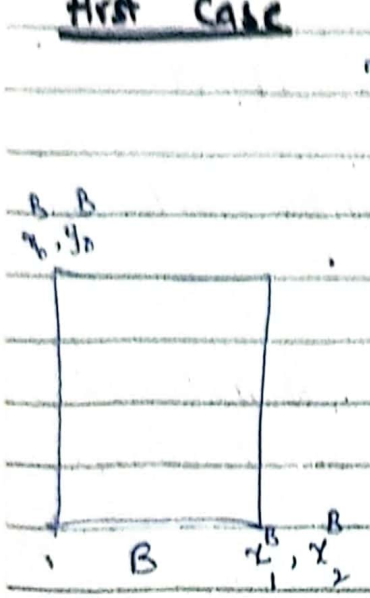
Suppose
Each car has two or more detection with different probalities. This come from some of the grids that thinks that this is the center point of the object.

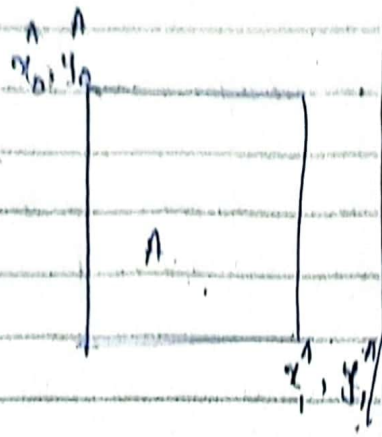
Non Max suppression Algorithm:-

↳ let's assume that we are targeting one class as an output class.
↳ y should be $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$. P_c is the probality if that object occur.

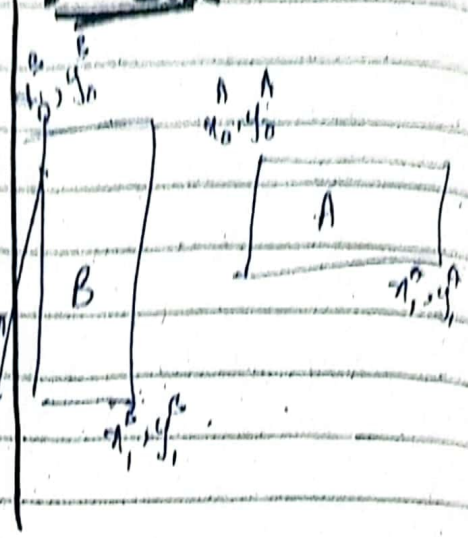
↳ Discard all boxes with $P_c < 0.6$
↳ while there are any remaining boxes.
  ↳ pick the box with the largest P_c output that as a prediction
  ↳ Discard any remaining box with IOU > 0.5 with that box output

first case





2nd case



$$x_0^I = \max(x_0^A, x_0^B)$$
$$y_0^I = \max(y_0^A, y_0^B)$$
$$x_1^I = \min(x_1^A, x_1^B)$$
$$y_1^I = \min(y_1^A, y_1^B)$$

width − I $= x_1^I - x_0^I$

height − I $= y_1^I - y_0^I$

if any negative value of height and width then there is no intersection

both negative area. not intersection

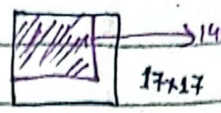
$$A \cap B = (x_1^I - x_0^I) * (y_1^I - y_0^I)$$

$$A \cup B = \text{area } A + \text{area } B - A \cap B$$

$$\text{area } A = (x_1^A - x_0^A) * (y_1^A - y_0^A)$$
$$\text{area } B = (x_1^B - x_0^B) * (y_1^B - y_1^B)$$

$$IOU = \frac{A \cap B}{A \cup B}$$

Consider an image 17x17, we need to map it to a space of 7x7



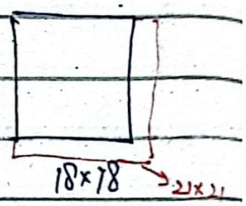
the required stride is $\frac{17}{7} = \lfloor 2.42 \rfloor = 2$

Since stride 2.42 is meaningless. ROI pooling will quantize this value by rounding it down to. however in doing so, it only consider the top (14x14) in (17x17) region.

The remaining points are lost, not only is there a loss of data but can also lead to misalignment.

image 18x18 ⇒ map 7x7

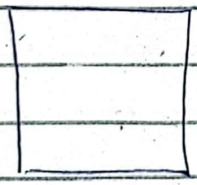
$\frac{18}{17} = \lceil 2.57 \rceil = \lceil 3 \rceil$



this is mis-allignment when we perform pooling here.

Now to address this problem **ROI align** is used, no quantization take place.

ROI allign



$\frac{17}{7} = 2.42$

we conside window 2.42 as it is, however this is meaningless each cell is divided into 2x2 bin.

RoI Polling :-

RoI polling is Quantized stride
Hence, Data loss.

usually applying max polling on a RoI.

1	3	2	9
7	4	1	5
8	5	2	3
4	2	1	4

7	9
8	4

stride = 2 is the number of cells by which we move our sliding window during pooling or during convolution

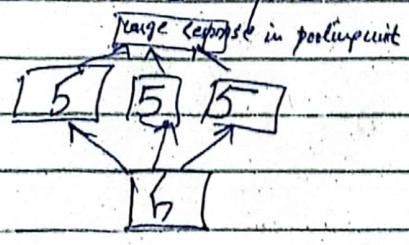
The bounding box compute during object detection here the name RoI pool.

⇓

stride is Quantized

pooling is used for downsampling the features and is used to introduce invariance of minor distortions in input.
This minor distortion may be some minor rotation in an image.

Consider this image as a "5" but even it is rotated, our model predict as a "5"



in the previous step.

- if there are multiple classes/object types (c) you want to detect, you must run non max suppression c times, once for every output class.

Convolutional operator

edges → primitive features
→ slightly complex
→ full face
→ classify

Basic of image

Gray Scale image :- (2D)

1 channel

0 – 255 ↓ normalize

0 – 1

↓ black ↓ white

color image ⇒ 3 channel

Red Green Blue

⇒ Edges are basically changes in intensity.
⇒ filters / kernals

Horizonal edge detection

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

CamScanner

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

image

$\circledast$

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

filter/kernel

$\downarrow =$

feature map

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

$\Downarrow$

Place filter in image each pixel multiply and then add

vertical edge detection

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

slight edges

/ \

In deep learning, you can't take tention what values of filter, you first initialize random, then it back propagate and automatically decide what value.

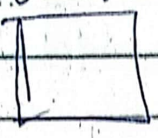
filter value behave like ANN weights
We know that filter value also
in training time.

image	filter		feature map
6x6	3x3	⇒	$[i-k]+1$
i j	k		$[6-3]+1$
			$3+1$
			(4×4)

Color image

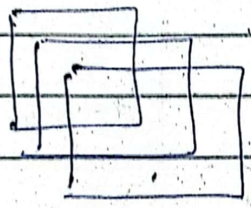
$\underbrace{6\times6}\times3$ → channel

filter



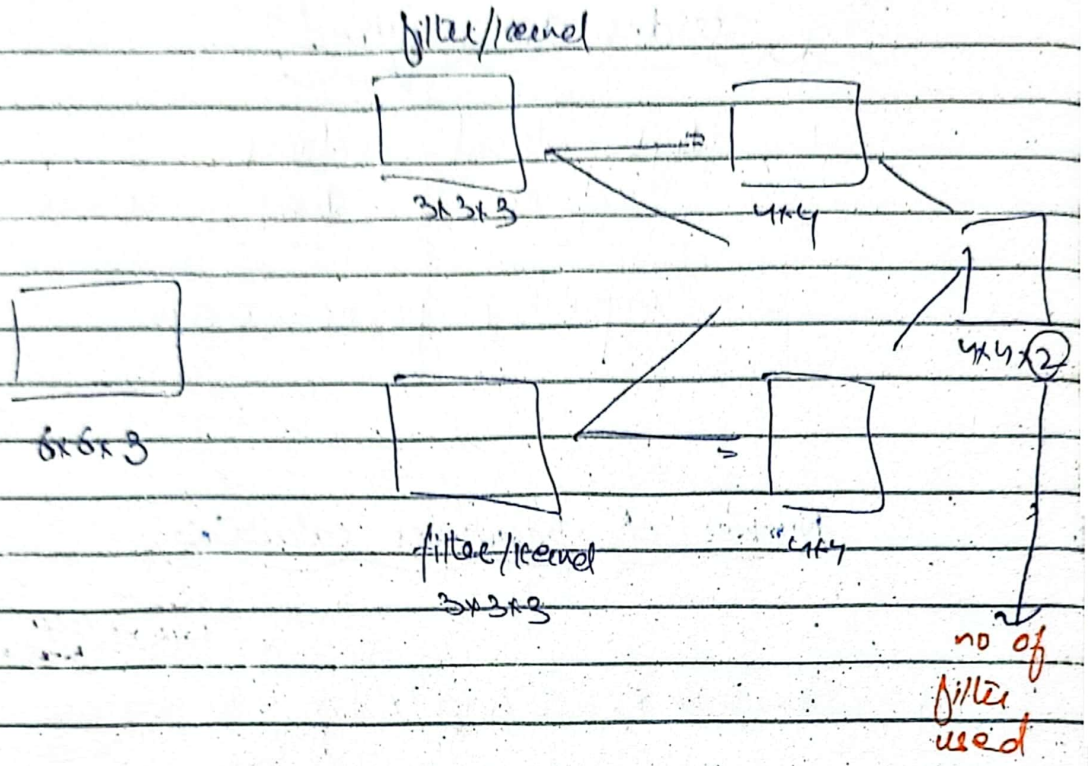
$\underbrace{3\times3}\times3$ → channel

⇒ 4×4 → single channel



$m\times m\times c$ $\quad$ $n\times n\times c$

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



Padding

Problem

⇒ convolution layer move
↳ size reduce after each convolution

⇒ border convolution less

that problem solve padding

$[i - k] + 1$ | $\left[\frac{i - k + 2p}{s}\right] + 1$

both convolution and pooling layer.

image 6x6
filter 3x3

$= \left[\frac{6-3}{2}\right] + 1 = \frac{3}{2} + 1$
$= 2$

Why strides are required?

1- High level features capture, if stride increase

2- Computing power reduce

Number of parameter calculation

of channel
↳ if no filter in prev layer

(filter width × filter height × # of filter in prev layer ~~× # of filter in this layer~~

+1) × # of filter in this layer

Transpose Convolution

Input

3	3	2	1
0	0	1	3
3	1	2	2
2	0	0	2

Kernel

0	1	2
2	2	0
0	1	2

Stride = 1

$$\Downarrow$$

$$\begin{bmatrix} 0 & 1 & 2 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

3	3	2	1
0	0	1	3
3	1	2	2
2	0	0	2

$$\Downarrow$$

3	3	2	1
0	0	1	3
3	1	2	2
2	0	0	2

$$\begin{bmatrix} 0 & 0 & 1 & 2 \\ 0 & 2 & 2 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Downarrow$$

3	3	2	1
0	0	1	3
3	1	2	2
2	0	0	2

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

$$\Downarrow$$

3	3	2	1
0	0	1	3
3	1	2	2
2	0	0	2

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 2 & 2 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix}$$

$$\Downarrow$$

Different types of Convolution layers:

- → Simple Convolution
- → 1x1 Convolution (pointwise convolution)
- → Flattened Convolution
- → Spatial and cross channel convolution
- → Depthwise Seperable Convolution
- → Grouped Convolution
- → Shuffle Grouped Convolution
- → Transpose Convolution
- → Dialated Convolution

⇓

Dialated convolution is a convolution applied to input with defined gaps.

Dialated Rate $k=1$ ⇒ normal convolution

$k=2$ skipping one pixel per input

$k=4$ skipping 3 pixel

Motivation behind Dialated Convolution

- → detection of fine details by processing input into higher resolution.
- → to capture more contextual information.
- → Faster run time with fewer parameters.

input image height stride kernel padding

$$O_h = (I_h - 1) \times S_h + K_h - 2p$$

$$O_w = (I_w - 1) \times S_w + K_w - 2p$$

Transpose convolution is similar to deconvolution layer.

input = (4x4)
Stride = 1
kernel = 3x3
padding = 0

Deconvolution

reverse the layer of standard convolution

if the output of the standard convolution layer is deconvolved with the deconvolution layer. then the output will be same.

$= (4 - 1) \times 1 + 3 - 2(0)$

$= 3 + 3 = 6$

Transpose convolution value will not be same, it can reverse to the same dimension.

(6x6) => every input pick and multiply by whole kernel.

0	3+3	6+1+0	6+2+0	4+1	4
6+0	6+6+0+0	0+6+4+0	3+4+2+0	0+2	0
0+0	3+0+0+0	6+3+0+0	6+2+0+0	4+1	2
0	0+0	0+	0		

used for image generation,
image superresolution
image segmentation

upsampling the input data.
converting low resolution image into high resolution.

$$C = \begin{bmatrix} 0 & 1 & 2 & 0 & 2 & 2 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 2 & 2 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 2 & 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 2 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

4×16

input

$$\begin{bmatrix} 3 & 3 & 2 & 1 \\ 0 & 0 & 1 & 3 \\ 3 & 1 & 2 & 2 \\ 2 & 0 & 0 & 0 \end{bmatrix} \Rightarrow I = \begin{bmatrix} 3 \\ 3 \\ 2 \\ 1 \\ 0 \\ 0 \\ 1 \\ 3 \\ 3 \\ 1 \\ 2 \\ 2 \\ 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$R_{conv} = \underset{4 \times 16}{C} \cdot \underset{16 \times 1}{I} \Rightarrow (4 \times 1) \Rightarrow (2 \times 2)$$

$$\begin{bmatrix} 12 \\ 12 \\ 10 \\ 17 \end{bmatrix} = \begin{bmatrix} 12 & 12 \\ 10 & 17 \end{bmatrix}$$

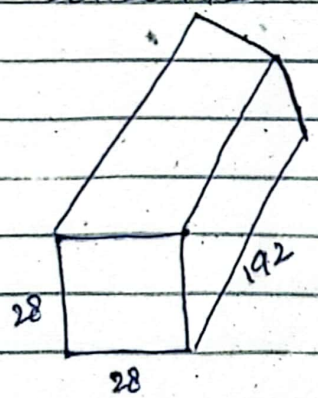
Pointwise Convolution (1x1 Convolution)

⇒ that used 1x1 ~~convolution~~ kernel
↳ Kernel that iterate through every single point
↳ The kernel has a depth ~~of~~ however many channels the input image has.

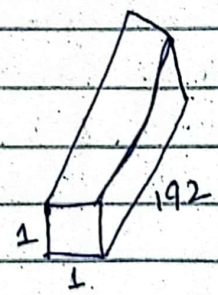
⇒ it is used in conjunction with depthwise convolutions to produce an efficient class of convolution known as Depthwise - Seperable Convolution.

Q. input image has M channel
Image height ← 28 x 28 x M → image width, → channel
then in pointwise its depth is
1x1xM ⇓ channel

image
28x28x192



pointwise-conv
1x1x192



# of filter
= 16

output

⇒ $\frac{n-f+2p+1}{s}$

$\frac{28-1+0+1}{1}$

$27+1$

$=28$

output image = (28x28x16)
↓ I.h ↓ I.w ↓ # of filter ⇓ depth

Grouped Convolution

- input channel is divided into several groups.
- Each group of channels is convolved independently using a seperate set of filters.
- The results from all groups are concatenated along the channel axis to produce the final output.

# of operations:-

output image dim × filter image size

$$\left(H \times W \times \frac{C_{out}}{G} \times F_H \times F_W \times \frac{C_{in}}{G}\right) \times G$$

Advantage of Grouped Convolution:-

1- Reduced computation Cost.

⇒ dividing channel into groups decrease the # of parameters and reduce the floating point operation ~~(FLOPs)~~ (FLOPs)

Cross channel Convolution.

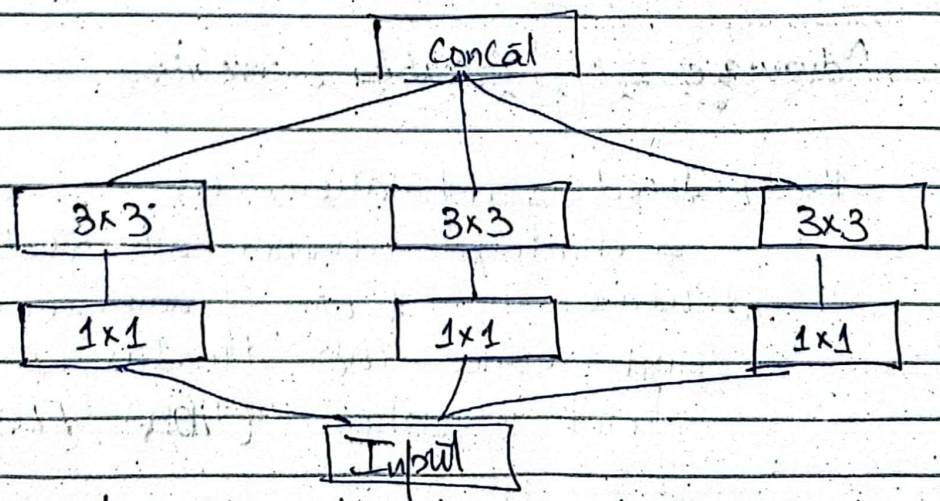
(Point wise Convolution)

Refer to the operation that applies filters across the channel dimension (depth) of the input feature map without affecting the spatial dimension.

⇒ combine features across channel, often used for dimensionality reduction or expansion.

⇒ often used depth wise Seperable Convolution and bottleneck layer.

Spatial and cross channel convolution was widely used in Inception Network.



⇒ used for dimention reduction.

Flatten Convolution

Process of converting the multidimentional output of a convolution layer into a 1D vector.

= height x weight x channel

Spatial and cross channel Convolution

Spatial Convolution:-

refer to standard Convolution operation that applies filter (kernel) over the spatial dimension of the input feature map.

Height, width

3x3 conv
5x5 conv

⇒ capturing pattern like edges, textures and structure by looking at neighbouring pixels.

moving the kernel every part of image $5 \times 5 \times 1 \times 1$

2- Pointwise Convolution

image	conv	output image
$8 \times 8 \times 3$	$1 \times 1 \times 3$	$8 \times 8 \times 256$
	# of filter $\times 256$	

moving the kernel every part of image $1 \times 1 \times 3 \times 256$

input-image → Depthwise Conv. → Pointwise Conv.

$12 \times 12 \times 3 \rightarrow 5 \times 5 \times 1 \times 1 \rightarrow 1 \times 1 \times 3 \times 256$

output-img $8 \times 8 \times 256$

$8 \times 8 \times 3 \times 5 \times 5 \times 1 = \cancel{4800}\ 4400$

$8 \times 8 \times 256 \times 1 \times 1 \times 3 = 49152$ +

# of multiplication = ~~53552~~ 53952

# of multiplication

= output image dim × Convolution

= 8×8×256 × 5×5×3

= 13,4,572,800 12,28,800

= 310 Million 1.2 Million

Depthwise Seperable Convolution

Seperate normal convolution into two parts.

1- Depthwise Convolution
2- Point wise Convolution

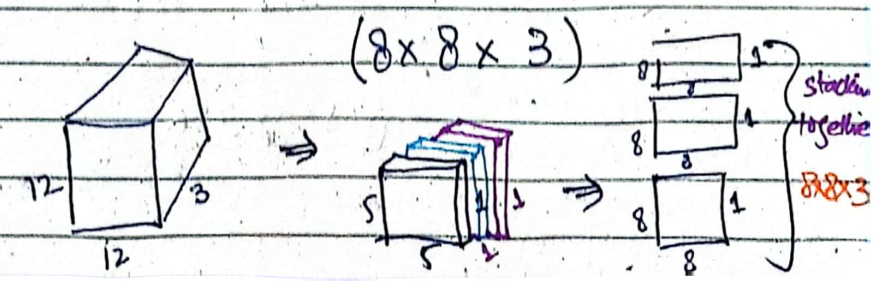
1- Depthwise Convolution

⇒ input image

⇒ a convolution layer without changing its depth.

image	Conv
12×12×3	5×5×1 } using 3 kernel of 5×5×1
	5×5×1
	5×5×1

output



Depthwise Seperable Convolution:-

Normal Convolution:- image 12x12x3
conv 5x5

input image	convolution	output image
12x12x3	5x5x3 → depth change	$\frac{12-5+2(0)}{1}+1$
	# of filter 1	$=7+1=8$
		(8x8x1)

input image has 3 channel, our convolution need to be have 3 channel as well.

# of multiplication :-

= output-image-dim × Convolution layer dim
= 8x8x1 × 5x5x3
= 4800

one times kernel move, total number of multiplication is

filter-height × filter width × channel/depth
5 × 5 × 3
= 75 multiplication.

---

input image	convolution	output
12x12x3	5x5x3	(8x8x256)
	# of filter = 256	

moving the kernel from every part of image (one pass)...

5x5x3x256
height, width, input channel, # of output channel in kernel

# of multiplications

= output image size × filter image size

because output image each pixel came from multiplication of filter size image.

$= 28 \times 28 \times 16 \times 1 \times 1 \times 192$

$= 24,08,448 \approx 2.4\ M$

24 lakhs

1 million = 10 lakh

Simple Convolution :-

C1

[illegible]	0.9219	0.7588	0.3811	0.0262	0.3594	0.7933	0.7811	0.4643	0.6329

C2

[illegible]	0.2302	0.8003	0.7353	0.7477	0.5885	0.6226	0.8429	0.6105	0.1248

two input channel, every input channel will have corresponding kernel.

So two input channel implies you have two kernel.

however, one kernel per channel is not really sufficient.

This is a way, we make used of filter bank.

By default, we have one filter bank in convolution layers.

Filter Bank 1	Filter Bank 2
0.2666 -0.2354 0.2918	-0.4055 0.3677 -0.2291
-0.0586 -0.0846 -0.2907	-0.1721 0.1457 -0.0002

F1

[illegible]	0.1724	0.1202	0.2003	0.1539	0.137	0.1637	0.2376

F2

[illegible]	-0.2949	-0.3265	-0.2687	-0.207	-0.3304	-0.2853

-0.1372

output:-

[illegible]	-0.1166	-0.2063	-0.0624	-0.1181	0.1934	-0.1222

0.1463

F1

0.0047	-0.1822	-0.1736	-0.2273	-0.0603	-0.033

-0.1409 -0.291

F2

-0.0817	0.0768	-0.0307	-0.0177	-0.0474	-0.0056

0.0155 -0.0567

output:-

-0.0771	-0.1054	-0.2098	-0.245	-0.2077

-0.0385 -0.1254 -0.3471

2- Efficient learning :-

allow the n/w to learn different type of features in parallel.
⇒ each group learn seperate feature then combined.

3- Scalability
increase the number of filters or groups without an excessive increase in computation cost.

used in

→ ResNext
used for improve accuracy
⇒ exploring a new dimension through grouped connections, cardinality (set the size of transformers)
⇓
we can increase accuracy by increasing it.

→ AlexNet
⇓
distribute the model over multiple GPUs as an engineering compromise.

→ MobileNet
used depthwise seperatable convolution
↳ Depth wise
one group per input channel.

example
⇓
(chained convolution)

Shuffle Grouped Convolutions

Shuffle Net proposed how to eliminate main side effect of the grouped convolutions that

"output from a certain channel are only derived from a small fraction of input channel".

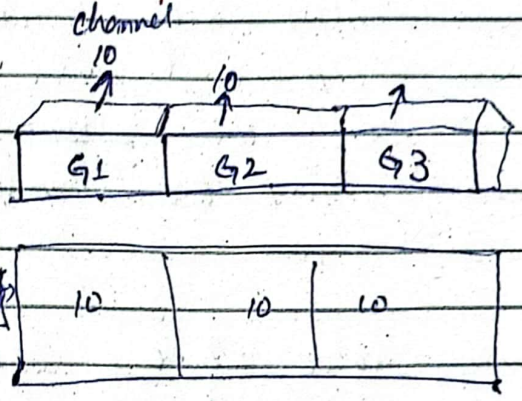
The proposed shuffle channels in such way (layer with g groups whose output has g×n channels)

- reshape the output channel dimension into (g,n)
- transpose output
- flatten output.

channel filtering is also differentiable

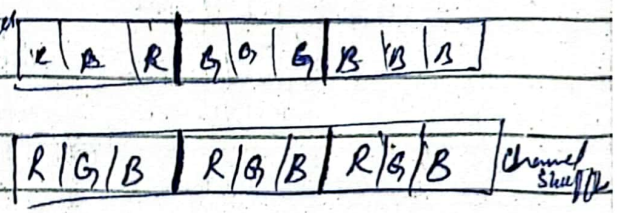
Application

drones
robots
smartphone

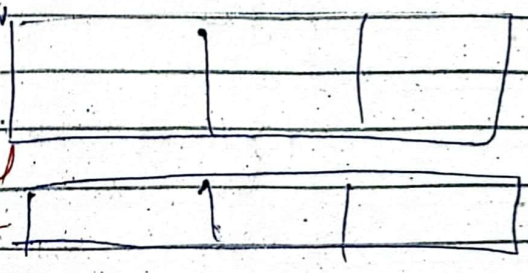


channel shuffle

g → no of groups
n → number of channel per group
gn → total number of channel



reshape to (g,n)
transpose ⇒ channel shuffle
flatten



4 input channel
2 Groups
2 output channel

```
X = torch.rand(
        size = (1, 4, 10)
        dtype = torch.float32
)
```

input channel

```
Conv_layer = nn.Conv1d(
        in_channels = 4,
        out_channels = 2,
        kernel_size = 3,
        padding = "valid",
        groups = 2
        bias = False,
        dtype = torch.float32
)
```

out_channel
⇓
control the number of filter bank.
here we have 2 outchannel, so we will have two filter bank.

Now, is an optimization to reduce the number of channel/kernel.

<u>Rule</u> :- Kernel in filter bank 1 will only convolve some input channel while filter in filter bank 2 will convolve remaining input channel.

Subset of input channel will be visible to each filter bank.

we achieve this by grouping the input channel.

group 1 { C1, C2 } ⇓ filter bank 1

{ C3, C4 } ⇓ filter bank 2

Filter bank 1			Filter bank 2		
0.0137	0.3338	0.3102	0.1088	-0.0131	0.2269
0.1591	-0.0319	-0.3051	0.3413	-0.3456	0.3831

⇓ output (1x8)

⇓ output (1x8)

for example, if you have 64 input channel, so will have 64 kernel per channel filter bank.

it should be clear, more computation will be required as well as more memory will be required.

Grouped Convolution is the technique to ~~used~~ reduce the number of kernels and ~~by~~ thereby reducing the computation requirements.

input

C1

0.3958	0.9219	0.7588	0.3811	0.0262	0.3594	0.7933	0.7811	0.4643	0.6329

C2

0.6689	0.2302	0.2003	0.7353	0.7477	0.5585	0.6226	0.8429	0.6105	0.1248

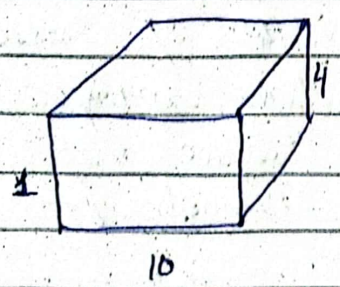
C3

0.8265	0.2117	0.8574	0.4282	0.3964	0.144	0.0034	0.9504	0.2194	0.2893

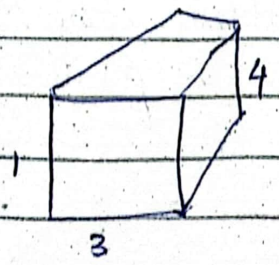
C4

0.6784	0.4997	0.9144	0.2833	0.5739	0.2444	0.2476	0.121	0.6869	0.6617

input

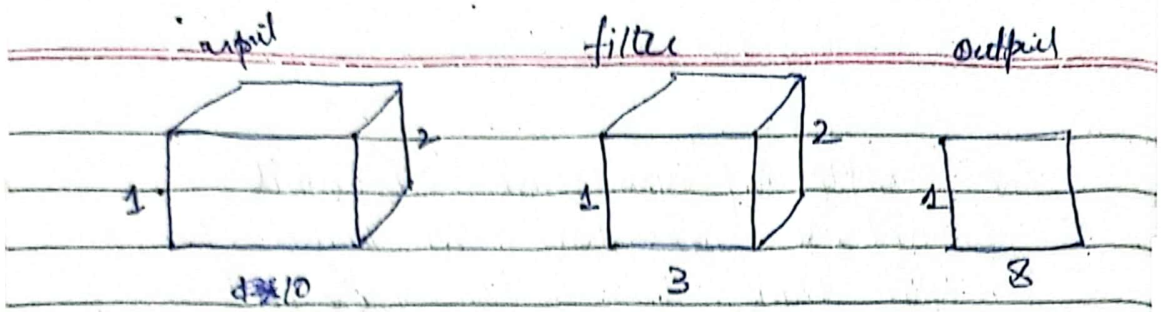


in simple convolution it filter will also have 4 channel



1 x 10 x 4

(1 x 3 x 4)



$1 \times 10 \times 2$ $\quad$ $1 \times 3 \times 2$ $\quad$ 1×8

channel same

# of filter = 1

$\frac{i - f + 2p + 1}{s}$, $\frac{i - f + 2p + 1}{s}$

$\frac{1 - 1 + 0 + 1}{1}$, $\frac{10 - 3 + 0 + 1}{1}$ $= 8$

1 , 8

Consider a situation, when you have very large number of input channel.

{ output of one layer is the input of another neural n/w }

start with 3 channel, and ~~you~~ every quickly in the middle of neural n/w you will start to have many feature map and these feature map will become the input channel of next subsequent layer.

Dilated Convolution:-

increase rate

is type of convolution that "inflate" the kernel by inserting holes b/w the kernel elements.

An additional parameter l (dialation rate) indicate how much the kernel is widened.

They are usually $l-1$ spaces inserted b/w the kernel elements

Dialated Convolution is always known as atrous convolution, is type of convolution neural n/w (CNNs) that enables the n/w to have a larger receptive field without increasing the number of parameters.

$l = 1$

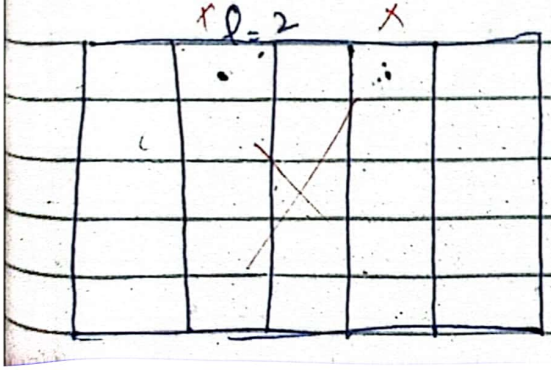
Kernel 3x3

1	6	3
2	5	4
4	7	5

spaces $= l - 1 = 1 - 1 = 0$

1	6	3
2	5	4
4	7	5

$l = 2$



$l = 2$

spaces $= l - 1 = 2 - 1 = 1$

Ok

1		6		3
2		5		4
4		7		5

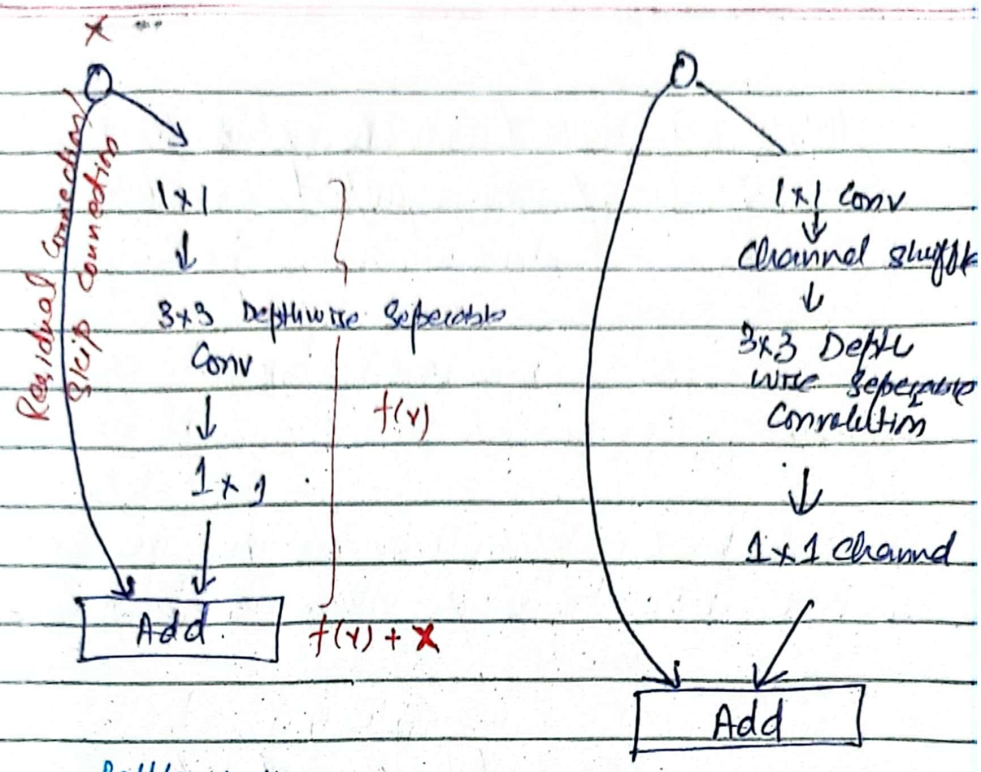
Bottle neck layer reduce the
→ number of parameters
→ matrix multiplication

Input → 1 × 1 conv → 3 × 3
~~filter~~ 256 channel
64 no of filter
↓
output ← 1×1 conv
# of filter = 256
⇓
restore vertical ~~filter~~ depth channel.

Bottle neck also include residual connection (skip connection) that adds the original input to the output of the block.

This help with training in deep n/w by allowing gradients to flow more easily through the n/w.

Bottleneck layer introduced in ResNet architecture, and also used in deeper
ResNet – 50
ResNet – 101



Bottle Neck Layer

Bottleneck layer reduce the number of channel (depth) of feature map.
→ help to lower the computational cost and the number of parameters while preserving/keeping the important features.
reduce memory overhead.

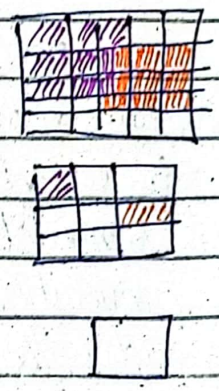
involve three convolutions.
↳ input 256 channel
1x1 → this reduce the number of channels of the input feature map effectively compressing the data.
# of filter = 64
64 channel computation
↳ 3x3 → feature map with spatial filtering.
1x1 → restore the depth back to the original or desired number of channels
# of filter: 256

What is receptive field?

The receptive field is the portion of the input needed to produce the feature at this layer.

To find receptive field of a feature in the 2nd convolution, we can work backwards until we get the input

Input
↓
Conv
↓
Conv



→ This layer receptive field size is "5".

$$r_{l+1} = s_i \times r_i (k_i - s_i)$$

This formula is different for other ~~kind~~ of architecture choices in your convolutional neural n/ws like pooling, stride and dialated convolution.

Advantage:-

- increased receptive field without increasing parameters
- can capture feature at multiple scales.
- reduced spatial resolution loss compared to regular convolutions with larger filters.

Disadvantage:-

- reduced spatial resolution in the output feature map compared to the input feature map.
- increased computation cost compared to regular convolutions with the same filter size and stride.

CamScanner

Data Augmentation

is derived from original data with some minor changes.

→ we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of training set.

flipping (horizontal or vertically)	Exposure (light)
Rotation (90°, 180°)	Contrast (darkness)
Blur	Grayscale (inverse or desaturate)
Noise addition	Scaling
Zooming	Translation

↳ Shift the image horizontally or vertically by a certain distance

When should you used Data Augmentation?

→ to prevent model from overfitting
→ the initial training set is too small.
→ to improve model accuracy
→ reduce the operational cost of labeling and cleaning the raw data

receptive field is in calculating the number of layers necessary to all of your input.

image
1000 x 1000

filter
3x3

how many layers do you need so that last layer is looking at all of the image.

3x3
5x5
7x7
⋮

2* no of layer + 1

2* no of layer+1 ≥ 1000
2 * no of layer ≥ 999
no of layers ≥ 499.5

500 layers!

receptive field
9

layer 4

7

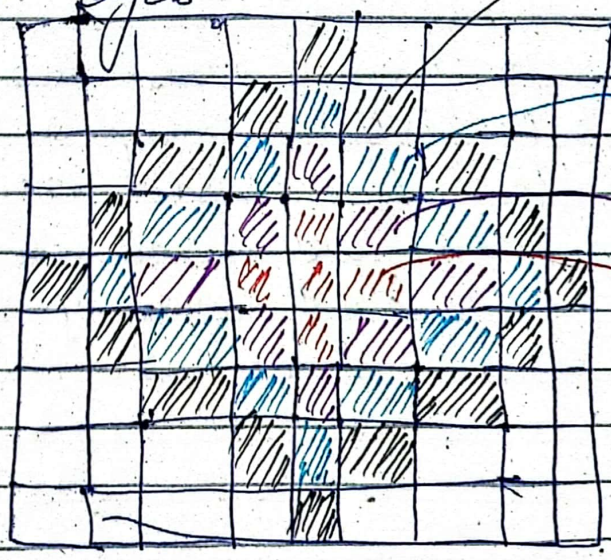
layer 3

5

layer 2

layer 1
3

0	1	0
1	1	1
0	1	0



Pretrained Model :-

such CNN architecture / neural n/w architecture which build someone else ~~and~~ in other dataset.
That model is best and you used in own model.

Why use pretrained Model?

↳ Deep learning is Data hungry

e.g classification problem
↳ image data → label data

you need lot of images ~~but~~ you download in google 10,000

but manually label is difficult
↳ financially impossible

↳ Time
model training takes larger time.

ImageNet :-

visual database of images.

Why ?
In 2006 Fei Fei Li thought deep learning is hype. In future data is needed to train deep learning model.

Limitation of Data Augmentation:-

→ The biases in the original dataset persist (continue) in the Augmented data

→ Quality Assurance for data Augmentation is expensive.

Synthetic Data:-

generated artificially without using the original dataset.

→ It often uses DNNs (Deep Neural N/ws) and GAN (Generative Adversarial N/W) to generate Synthetic data.

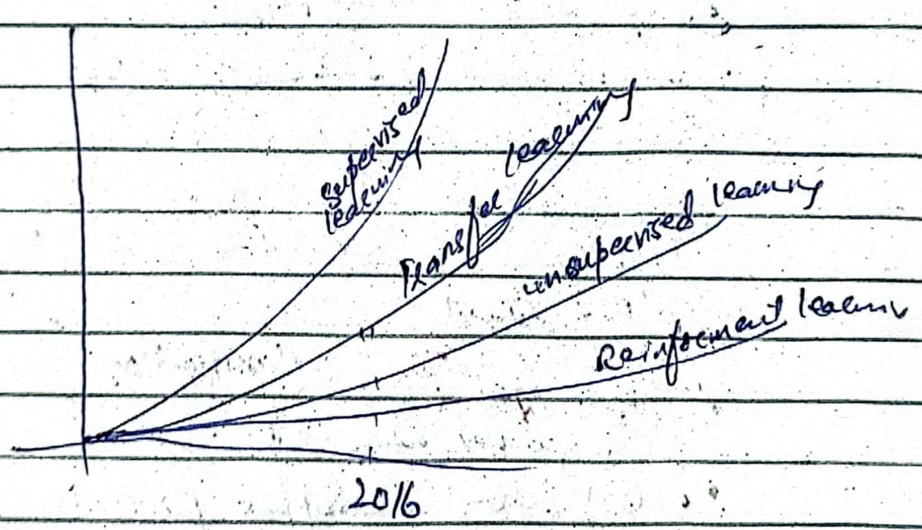
CamScanner

Transfer learning:-

Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

⇒ another data trained CNN model
tryout in own dataset.

2015 to onward big deal



Deep learning

2012 AlexNet → CNN

↳ GPU used

RELU activation layer used

16% error rate

2013	ZFNet	11.7%	
2014	VGG	7.3%	⇒ famous
2015	GoogleNet	6.7%	
2016	ResNet	3.5%	⇒ human stage cross

↳ human error rate is 5%

Xception

MobileNet

so he makes Databases
1.4 Billion (1.4 crore)
↳ 20,000 categories
{cat, dog, Women} ↳ daily bases.

↳ labeled all images
↳ 1 million images ⇒ bounding box make
↳ 10 lakh ⇓
Object localization.

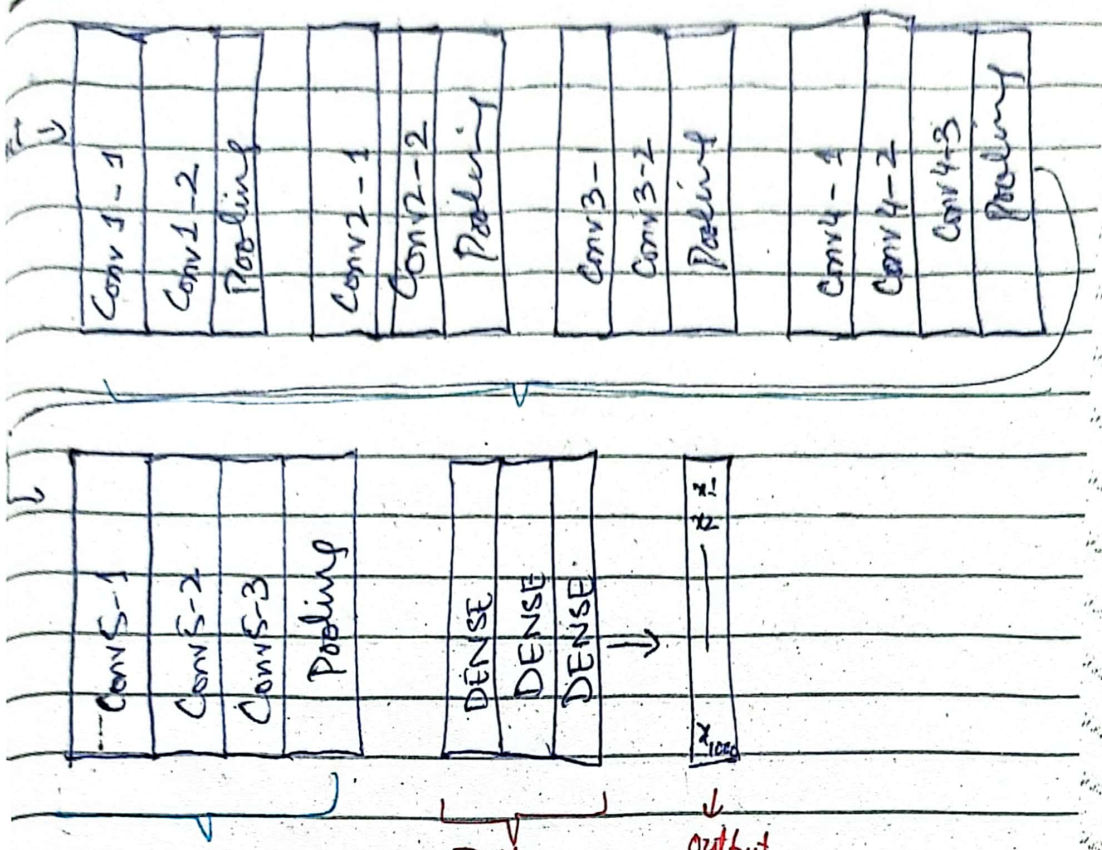
How? crowd Sourcing (helping taking from people).
Amazon Mechanical Turc.

ILSVRC build a compitition
Image net large scale visual Recognization Challenge.
start in 2010
↳ Goal
↳ Best image classification model find
⇒ In this compition, only subset of dataset used
1 Million ⇒ 10 lakh
1000 classes

Machine learning model
→ Manually feature → SHIFT feature extract, HOG feature extract → model

2010 → 28 % error rate [100 images see, 28 false predict]
2011 → 25 % " "

VGG-16



→ feature extract

→ classification

early layer primitive feature extract (edges) → complex feature → shape

- cat / dog classifier problem.

⇒ used VGG-16 that already trained on imageNet dataset.

⇒ VGG-16 Dense layer part replaced with own Dense layer and output layer.

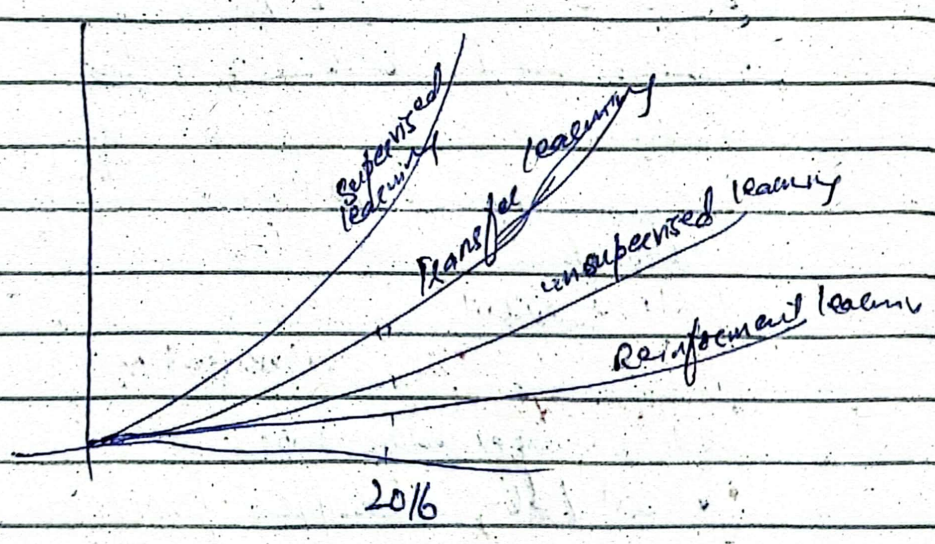
⇒ Convolution layer Freeze.

Transfer learning:-

Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

⇒ another data trained CNN model
tryout in own dataset.

2015 to onward big deal



Deep learning

2012 AlexNet → CNN
↳ GPU used
RELU activation layer used

16% error rate

2013 ZFNet 11.7%.
2014 VGG 7.3%. ⇒ famous
2015 GoogleNet 6.7%.
2016 ResNet 3.5% ⇒ human stage cross
↳ human error rate is 5%.

Xception
MobileNet

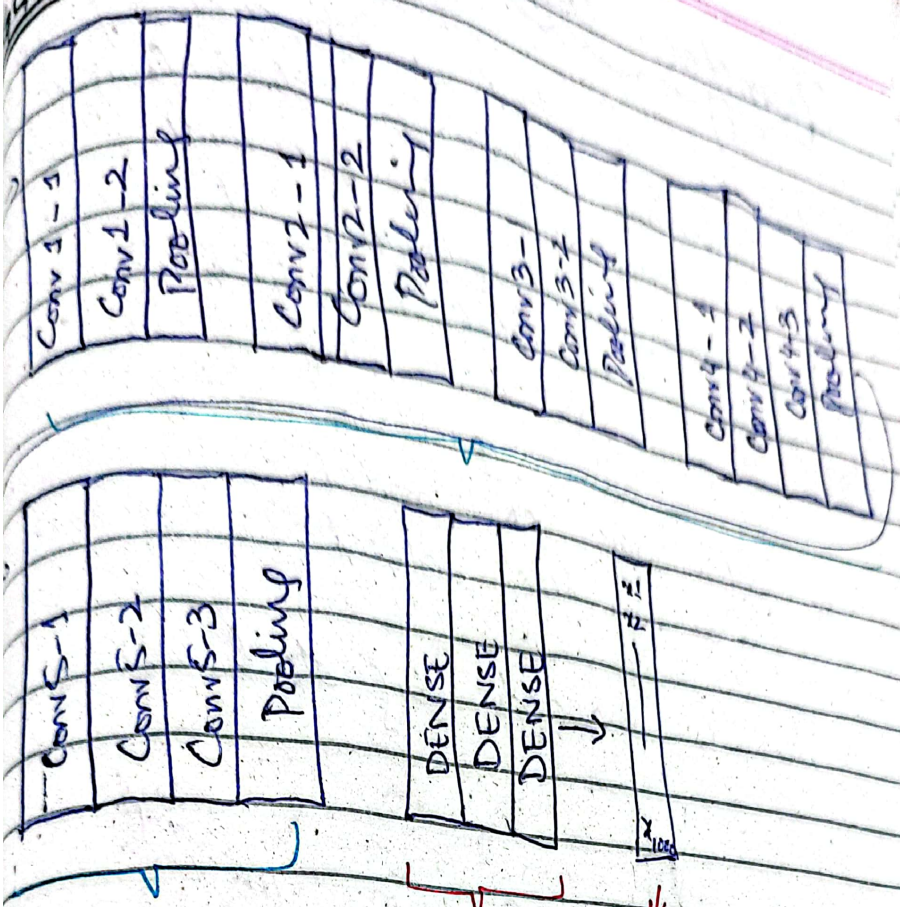
only fully connected layer trained

starting layers imageNet data has around 1000 classes data. Any thing that exist in real world have same primitive features. (edges, shapes)
↳ starting thing is similar in all basic in image.

Ways of doing transfer learning

- Feature extraction
  - Dense layer replaced with Conv layer.
  - Conv layer Freeze
    - ↳ when labels is similar to imageNet dataset
- fine tuning
  - ⇒ some last conv layer and Dense layer trained
    - ↳ & such problem you work that requirement is different from imageNet.
    - ⇒ more time

VGG-16



→ feature extract → classification

early layer primitive feature extract (edges) → complex feature → shape

- cat / dog classifier problem.

⇒ used VGG-16 that already trained on imageNet dataset.

⇒ VGG-16 Dense layer part replaced with own Dense layer and output layer.

⇒ Convolution layer Freeze.

Keras

Feature extraction

```
from keras.applications.vgg16 import VGG16
conv_base = VGG16(
        weights = 'imagenet'
        include_top = False,
        input_shape = (150,150,3)
    )
conv_base.summary()
```

```
model = Sequential()

model.add(conv_base)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
    )
model.summary()
```

//Dense layer.

Fine Tuning

```
conv_base = VGG16(
        weights = 'imagenet',
        include_top = False,
        input_shape = (150,150,3)
    )
```

```
conv_base.trainable = True

set_trainable = False

for layer in conv_base.layers:
    if layer.name = 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else
        layer.trainable = False

for layer in conv_base.layers:
    print (layer.name, layer.trainable)

model.sequential()

model.add(conv_base)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

NATIONAL UNIVERSITY
of Computer & Emerging Sciences

Course

Serial No 21703

Roll No

Section

Date

Main Answer Book Number

Input

3	3	2	1
0	0	1	3
3	1	2	2
2	0	0	2

Kernal

0	1	2
2	2	0
0	1	2

$(I_h - 1) \times S + K_h - 2p$

$(4 - 1) \times 1 + 3 - 2(0)$

$3 + 3$

(6×6)

1

0	3	6			
0	0	0			
0	1	4			

2

0	2	2			
0	2	0			
0	2	4			

3

		0	1		
		2	6		
		0	2		

Q / Part No.

Rough Work

image

3	3	2	1
0	0	1	3
3	1	2	2
2	0	0	2

Conv

0	1	2
2	2	0
0	1	2

First row

0	3+0	6+3+0	6+2+0	4+1	2
6	6+6	0+6+4	0+4+2	0+2	0
0	3+0	6+3+0	6+2+0	4+1	2

2nd row

0	1+0	0+0+0	0+1+0	2+3	6
0	0+0	0+0+2	0+2+6	0+6	0
0	0+0	0+0+0	0+1+0	2+3	6

3rd

0	3+0	6+1+0	2+2+0	4+2	4
6	6+2	0+2+4	0+4+4	0+4	0
0	3+0	6+1+0	2+2+0	4+2	4

0	2+0	4+0+0	0+0+0	0+2	4
4	4+0	0+0+0	0+0+4	0+4	0
0	2+0	4+0+0	0+0+0	0+2	4

Q / Part No.

Rough Work

$(2-1)\times 1+3$

1	0	3	9	8	5	2
2	6	13	10	7	7	6
3	0	6	16	20	17	6
4	6	8	6	9	9	6
5	4	7	7	8	10	4
6	0	2	4	0	2	4