Computer Vision problems

Image classification : cat ? 0/1

object detection :- Identify cars in a given image

Neural Style transfer :- Some face image + Some Style (Scenary painting image

face embedded in Scenary

Convolution operation

In a given image - we would like to find edges (horizontal as well as vertical edges)

Eg

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Convolution operator

*

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

3X3

filter (or) kernal

=

-5	-4	0	
-10	-2	2	
0	-2	-4	
-3	-2	-3	

4X4

$(3\times1) + 0(0) + (1\times-1) + (1\times1) + (5\times0) + (8\times-1) + (2\times1) + (7\times0) + (2\times-1) =$

$3 + 0 - 1 + 1 + 0 - 8 + 2 + 0 - 2 = -5$

Summary

$$6\times6 \text{ image} \quad * \quad 3\times3 \text{ kernel or filter} \quad = \quad 4\times4 \text{ image}$$

Filter

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

→ detects vertical edges

Example

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6×6

*

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

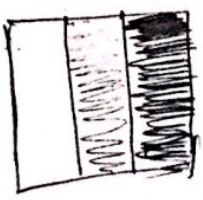
3×3

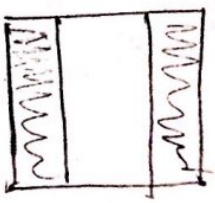
=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

image from

*





More on Vertical edge detection Exam

Eg $\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$

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

detect Vertical edge

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

detect horizontal edge

Eg $\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix}$ 6×6 $* \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 30 & 10 & -10 & -30 \\ 30 & 10 & -10 & -30 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

Note:- for large images (other than 6×6) → These filters will find horizontal edges more accurately

learning to detect edges

① $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ → called Sobel filter

② $\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$ → schars filter

to detect vertical edge in literature

Can we learn a new filter?

Ans **Yes**

NN can learn $\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$, this kind of filter by using backpropagation

* which may find not only horizontal, & vertical also can find tilted edges.

Padding ??

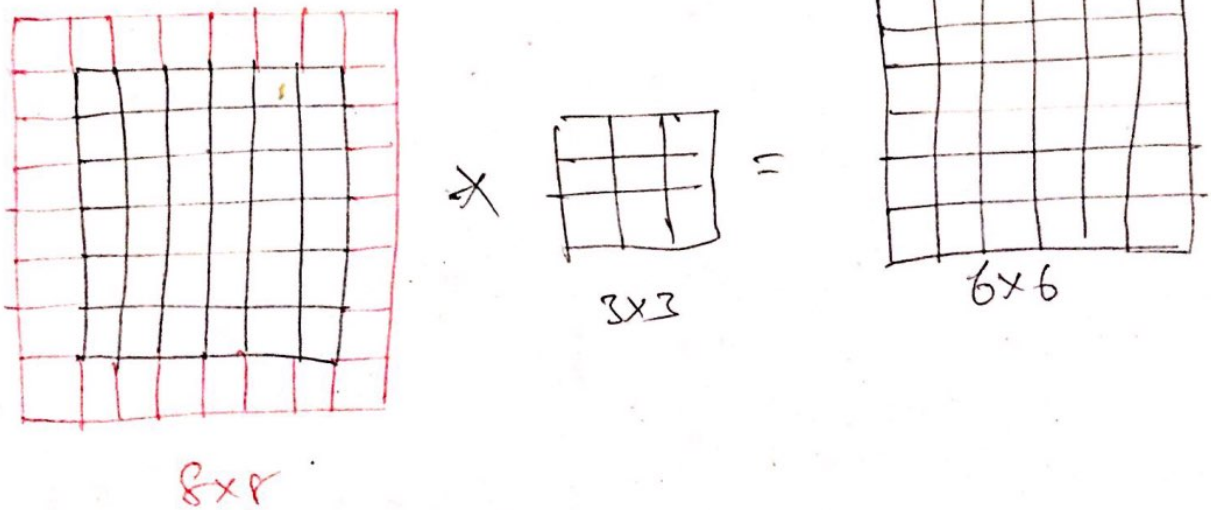
$(6\times6) * (3\times3) = (4\times4)$

in general

$(n\times n) * (f\times f) = (n-f+1 \times n-f+1)$

Note: In the prev filter moving over image (of size say 6x6) → the corner pixels are less contributing (loosing lot of info at corners) → To address (each pixel to be equally considered in convolution)

Image is also shrinking after convolution



Here p = padding = 1

After Conv. image size will be

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

$6+2-3+1 \rightarrow (6\times6)$

→ Is it any padding $p=1$ always?

NO

we can have padding $p=2$, or any integer

↓

But, on one condition

pad so that output size is the same as the input size.

Eg: (5×5) image $*$ $(3,3)$ filter $=$ (3×3) image after convolution

Image shrinking

how much padding we can apply?

$n=5 \quad f=3 \quad p=x$

$n+2p-f+1 = n$

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

$2x = 3-1$

$= 2$

$x = 1$

If $n=5$, $f=5$

Then $p = \frac{f-1}{2} = 2$

NOTE :

Always f should be odd (Usual convention)

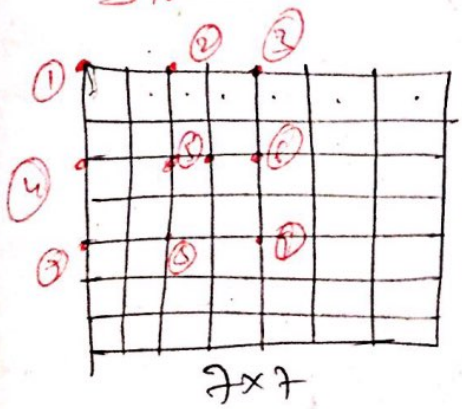
Remark

If $p=0$ (no padding)

is usually called "Valid"

"Same" : "pad so that output size is the same as the input size.

Strided Convolution

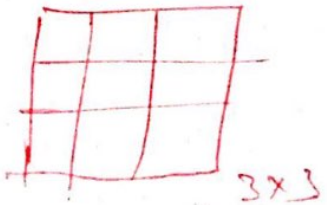


$*$

3	4	4
1	0	2
-1	0	3

3×3

$=$



Stride = 2 (means (jump stride) the filter by two pixels)

formula $(n \times n) * (f \times f)$

padding : p

stride : s

result image will be

$$\text{floor}\left(\left\lfloor \frac{n+2p-f}{s} \right\rfloor + 1, \left\lfloor \frac{n+2p-f}{s} \right\rfloor + 1\right)$$

here $n=7$ $f=3$, $s=2$

$$\frac{7+0-3}{2}+1 = 3$$

∴ The resultant image will be (3×3)

$\lfloor 2.4 \rfloor = 2$ (floor operation)

Summary of Convolution

$(n \times n) \quad * \quad (f \times f) = \left(\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \right)$

padding p

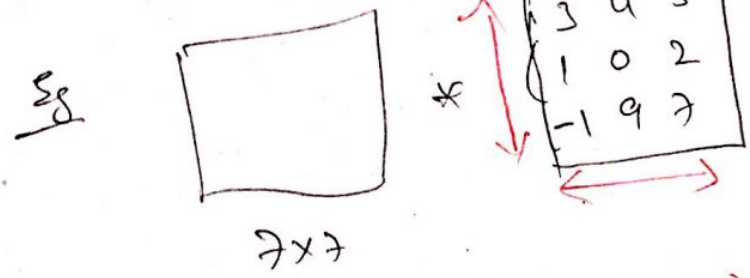
Stride s

→ output Image size

Cross-Correlation Vs. Convolution

In classical literature we will mirroring operation → what is this?

Eg



before doing Convolution we rotate and then apply Convolution "Mirroring operation"

$$\begin{bmatrix} 7 & 9 & -1 \\ 2 & 0 & 1 \\ 5 & 4 & 3 \end{bmatrix}$$

Note: ① we use the terminology only Convolution

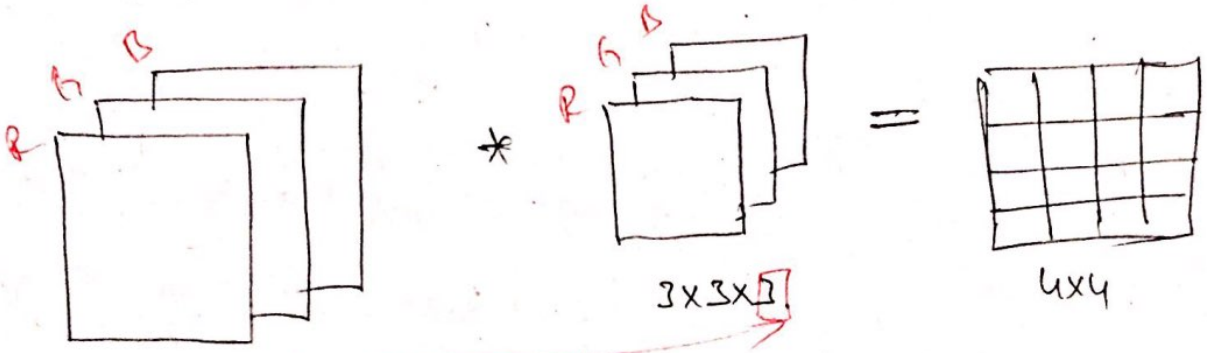
② In some texts Conv. & Cross-Correlation Synonymously but we use "Convolution"

③ Associative property $A * (B * C) = (A * B) * C$

Convolution over Volumes

We see not on grey scale images but on Color images

Eg



Note: **# channels = # f**

Eg If our filter is

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}_R \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_G \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_B$$

⇓

It will find vertical edge in Red channel

If our filter channels are

$$\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}_R \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}_G \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}_B \Rightarrow$$

detects vertical edges in all 3 channels

Can we apply multiple filter

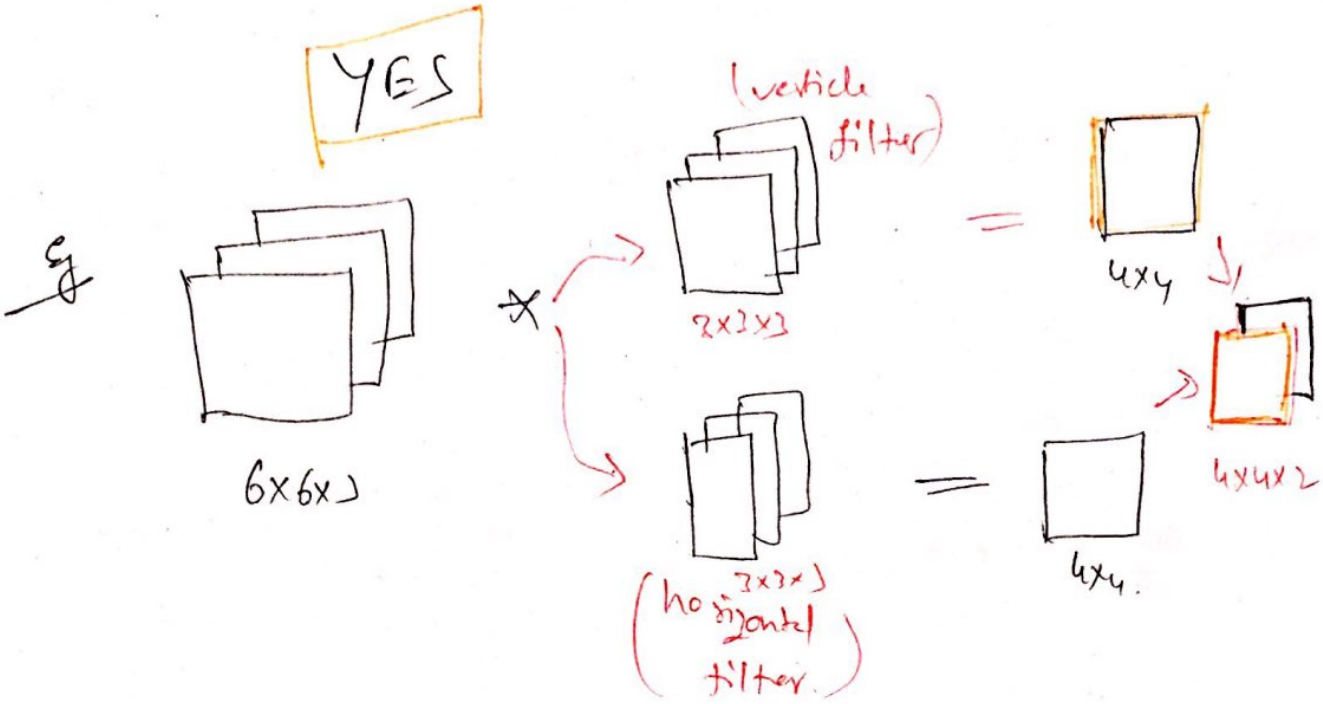
filter 1 → to find horizontal edges

filte 2 → to find vertical filter

⋮ 45° edges (60° edges etc)

⋮ 75° edges

YES



Summary $n \times n \times n_c$ input image (n_c = # Channels)

$f \times f \times n_c$ filter

$(n \times n \times n_c) * (f \times f \times n_c) \rightarrow [(n-f+1) \times (n-f+1) \times n_c']$

n_c' = # filters

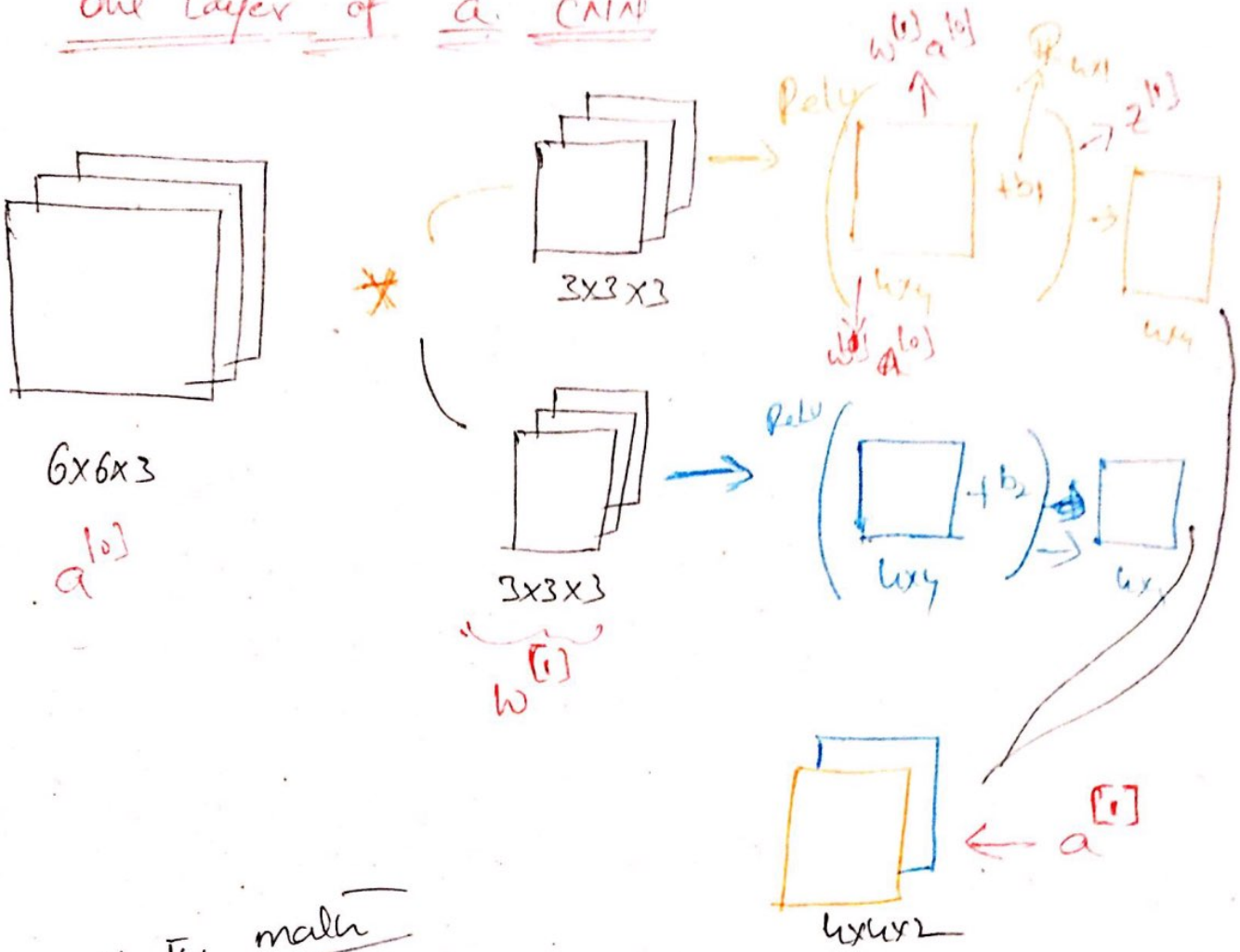
Note that n_c, n_c' are different

n_c → # Channels

n_c' → # filter

Some time n_c is also called as DEPTH

One Layer of a CNN

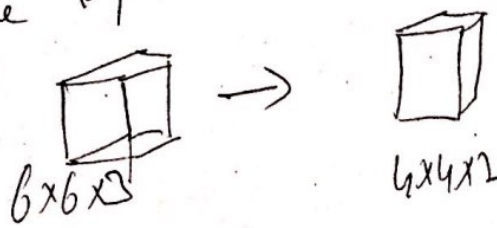


Recall the math

$$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

∴ In one layer of a CNN

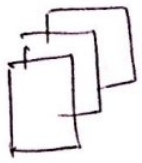


If there are 10 ~~layers~~ filters (each filter 3x3x3)

Then 6x6x3 → 4x4x10 ✓

Number of parameters in one layer?

If 10 filters each 3x3x3

for each filter  3x3x3 = 27

+ bias

---

28 parameters

∴ for 10 filters 28x10 = 280 parameters

Summary Notation

l : layer number

$f^{[l]}$ = filter size in layer l

$p^{[l]}$ = padding

$s^{[l]}$ = stride

Input $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

output : $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ → $n_c^{[l]}$ = # filters

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

④ each filter size

$$f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$$

⑤ activations: $a^{[l]} \Rightarrow n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

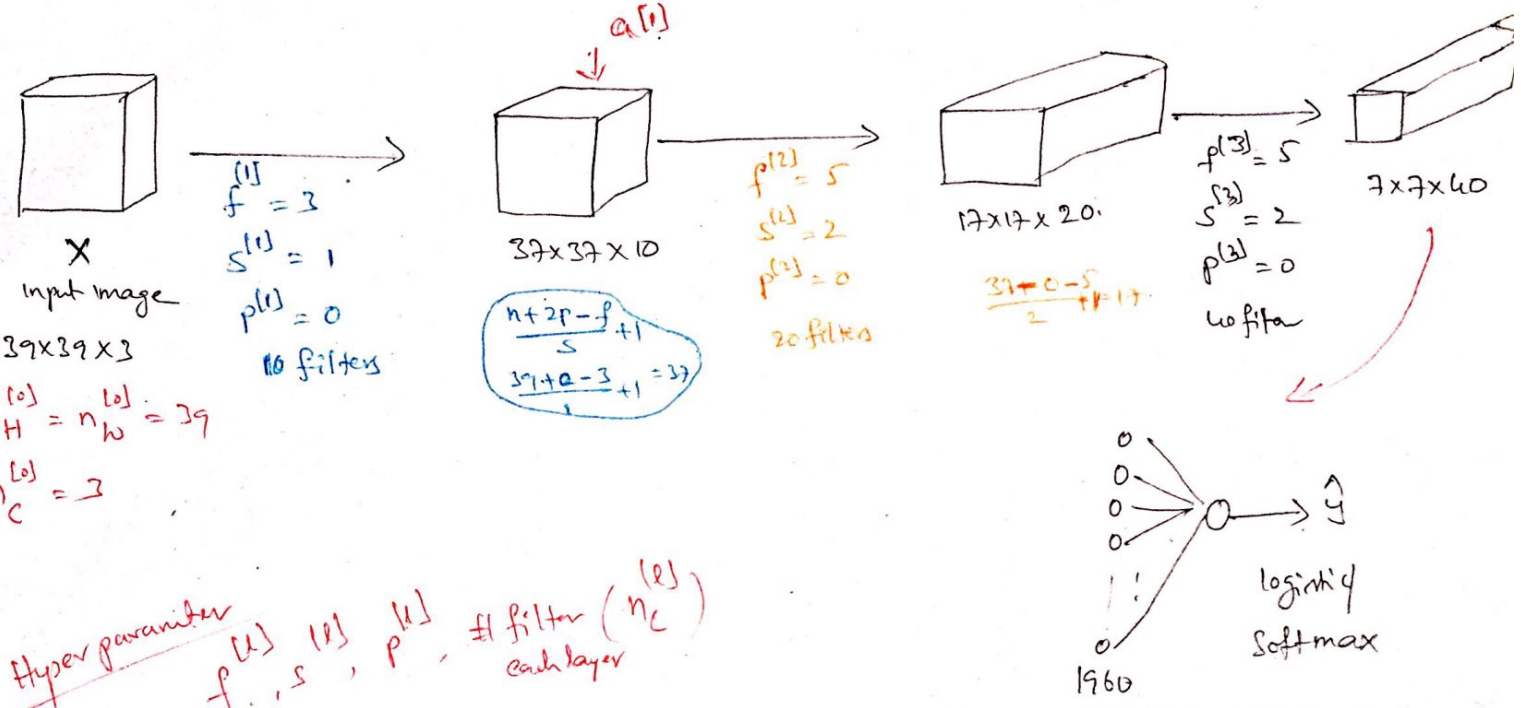
vectorized implement- $A^{[l]} = m \times n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

⑥ weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ → # filters in layer l.

⑦ bias: $n_c^{[l]} \rightarrow (1, 1, 1, n_c^{[l]})$

A Simple Convolution NN

ConvNet

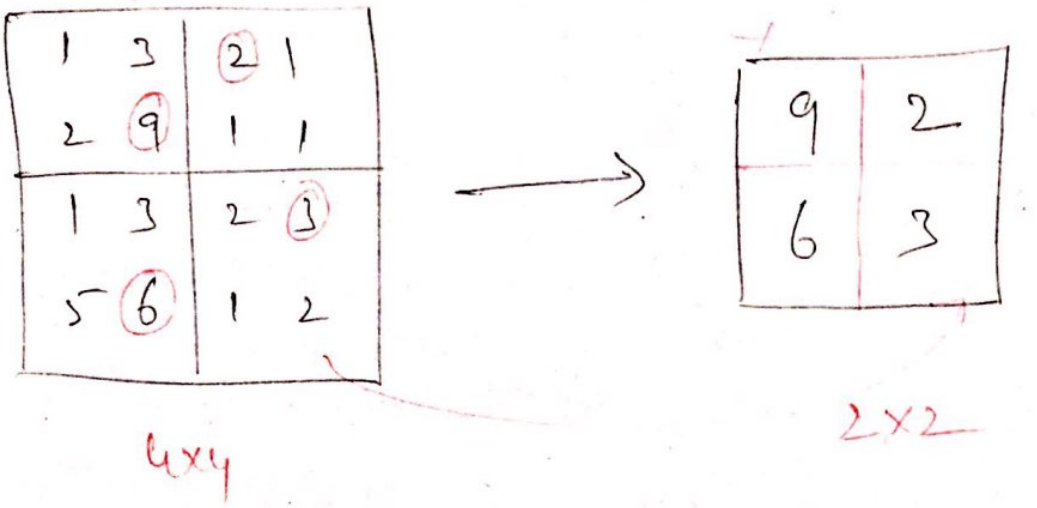


Hyper parameter $f^{[l]}, s^{[l]}, p^{[l]}$, # filter ($n_C^{[l]}$) each layer

types of layer in a ConvNet

1. Convolution layers (CONV)
2. Pooling layers (POOL)
3. Fully Connected layer (FC)

MAX POOLING



4x4

2x2

Ex: 1

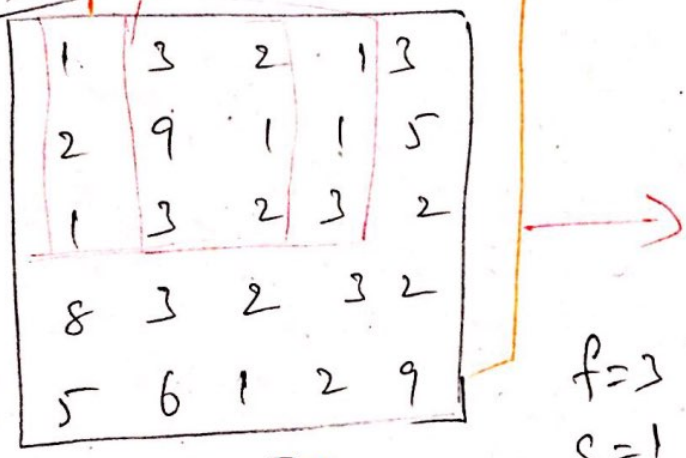
Hyperparameters

$f = 2$

$s = 2$

No parameters to learn

Ex: 2



5x5x2

$n_H = n_W = 5$

$f = 3$

$s = 1$

$p = 0$

3x3x2

$$\frac{n + 2p - f}{s} + 1 = \frac{5 + 0 - 3}{1} + 1 = 3$$

Input image size

$n_H \times n_W \times n_C$

output image size

$$\left(\frac{n_H + 2p - f}{s} + 1 \times \frac{n_W + 2p - f}{s} + 1 \times n_C\right)$$

Hyperparameters

f

s

: Average pooling

filter 2×2

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2

4×4.

Input

$f=2$

$S=2$

$P=0$

3.75	1.25
4	2

2×2

output

$\frac{4+0-2}{2}+1=2$

SUMMARY

Hyper paramter : output image size

f : filter size $\left(\left\lfloor \frac{n_H+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{n_W+2p-f}{s}+1 \right\rfloor \times n_C\right)$

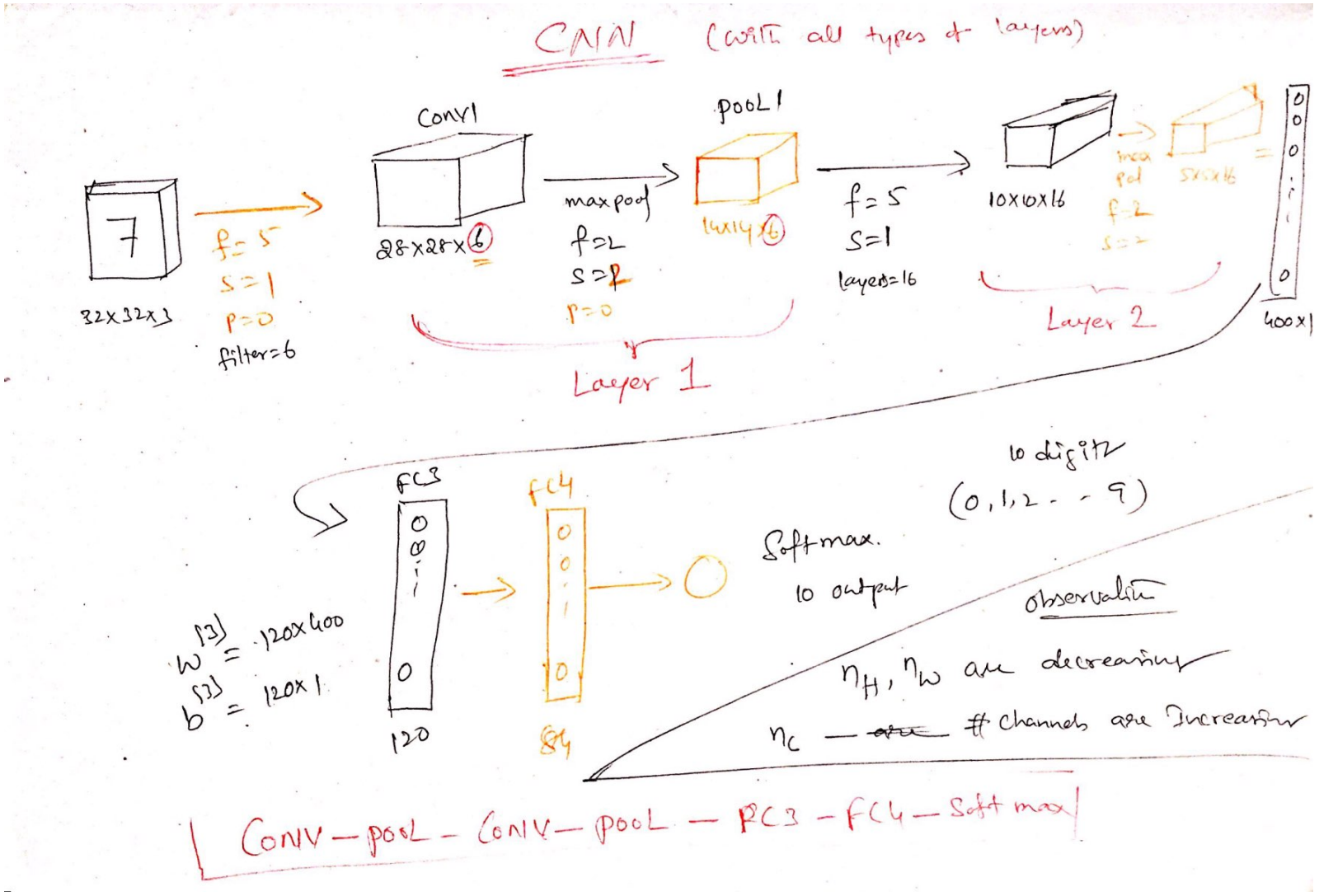
S : Stride

max or Avg pooling

(Usually padding doesn't make sense)

Note

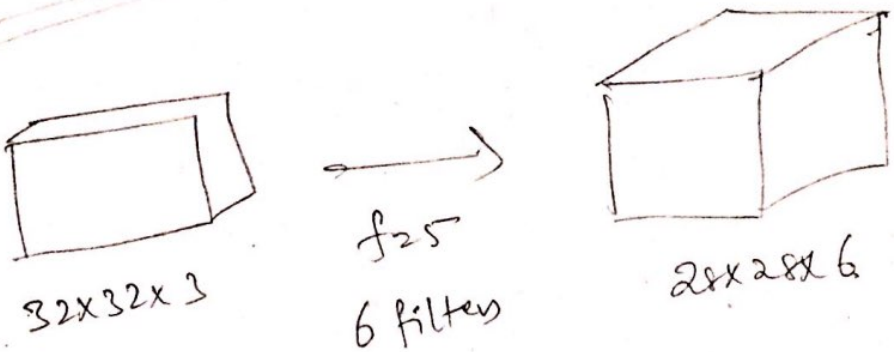
NO PARAMETERS TO LEARN



Summary

	Activation shape	activation size	# parameters
Input	(32×32×3)	3072 $a^{[0]}$	0
Conv1 (f=5, s=1)	(28, 28, 8)	6272	8×(5×5×3+1) = 608
Pool1	(14, 14, 8)	1568	0
Conv2 (f=5, s=1)	(10, 10, 16)	1600	(5×5×8+1)×16 = 3216
Pool2	(5, 5, 16)	400	0
FC3	(120, 1)	120	400×120+120 = 48,120
FC4	(84, 1)	84	120×84+84 = 10,164
Softmax	(10, 1)	10	84×10+10 = 850

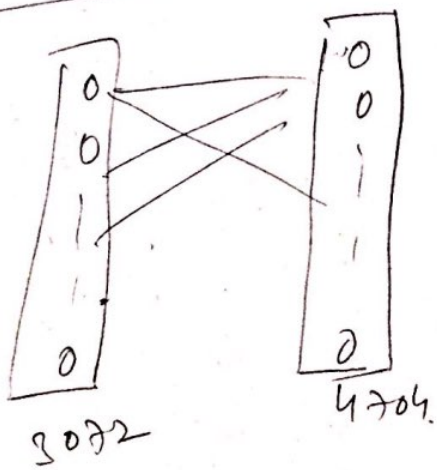
Why Convolutions?



parameters in each filter = $5 \times 5 = 25 + 1$ (bais) $= 26$

$6 \times 26 = 156$ parameters

In general fully connected DNN



weight matrix

3072×4704

$\approx$ 14 Million

Summary No parameters in Conv layers are small when compare to the fully connected layer in NN

Parameter Sharing:- A feature detector (such as vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

Sparsity of Connections

In each layer, each output value depends only on a small number of inputs. (Recall filters for edge detection)

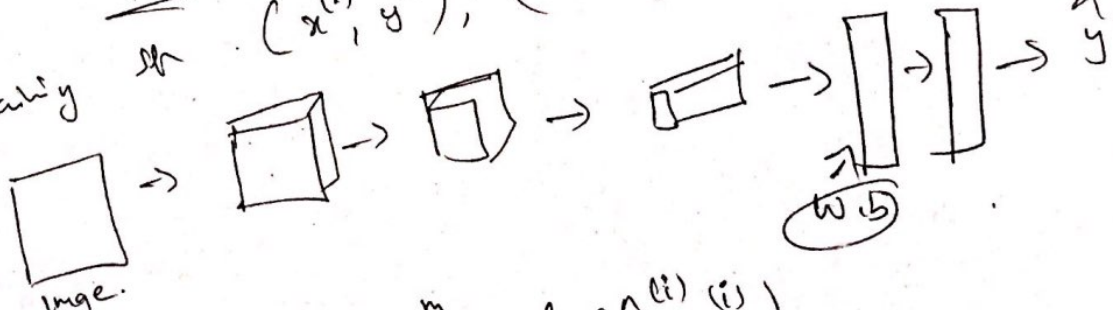
Translation Invariance

Why Convolution

1. parameter sharing
2. sparsity of connections
3. Translation invariance

PUT TOGETHER

Training set $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})$



Cost $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Use Grad. Descent to optimize parameters to reduce J

Case Studies- CNNs

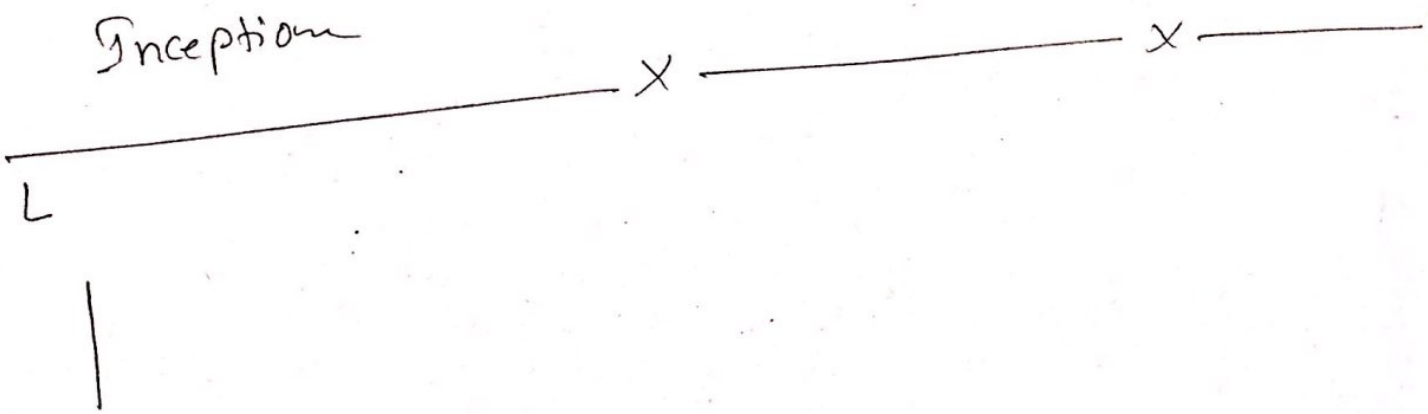
Deep Convolutional Models

Classic Networks:

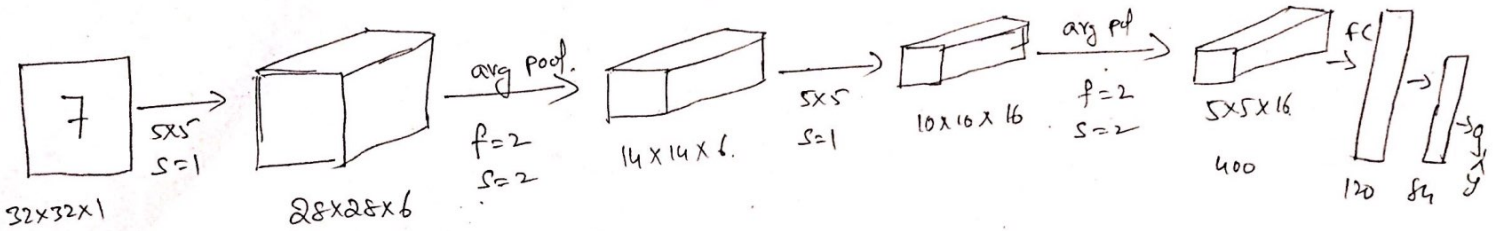
* LeNet-5
* AlexNet ←
* VGG

ResNet (Residual Network)

Inception

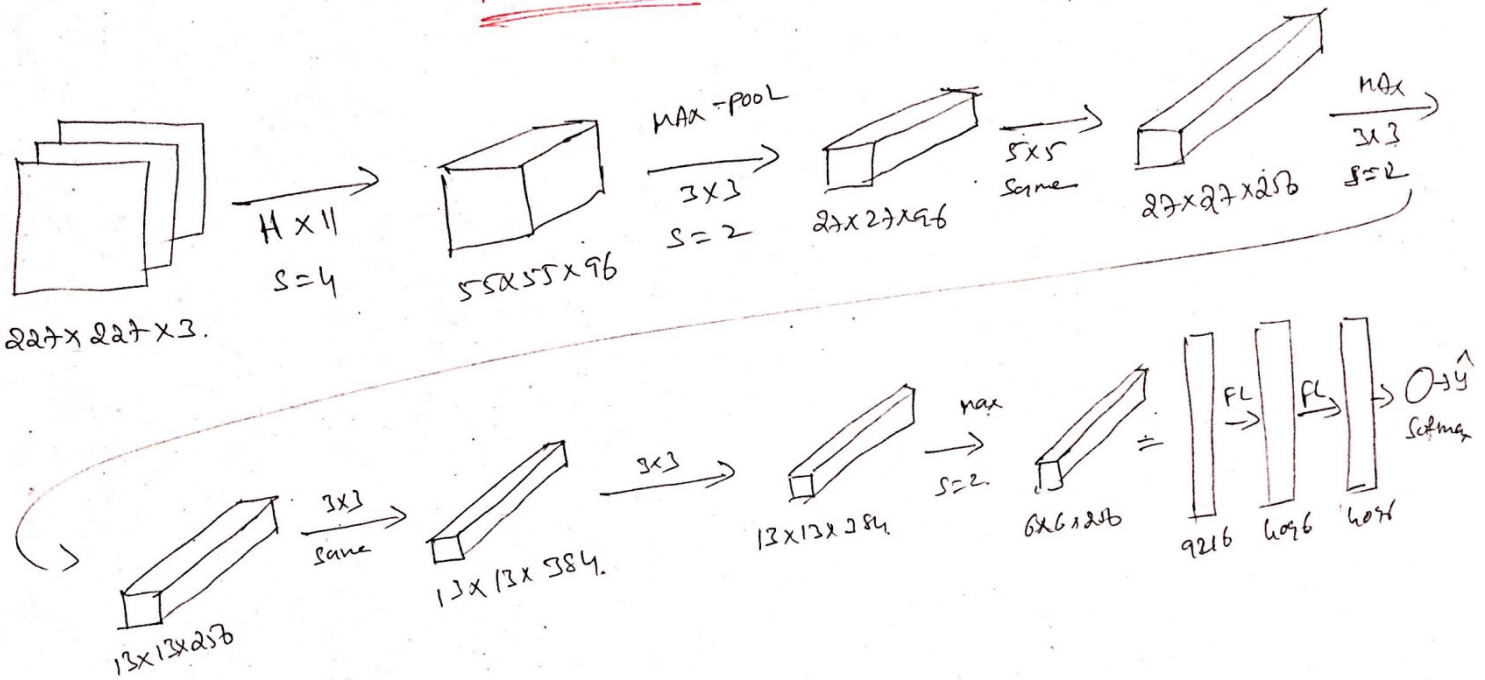


LeNet-5



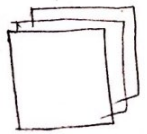
here $n_H, n_W \downarrow$ and $n_C \uparrow$.

Alex Net



VGG-16

Conv = 3x3 filter, s=1, same MAX-pool = 2x2, s=2

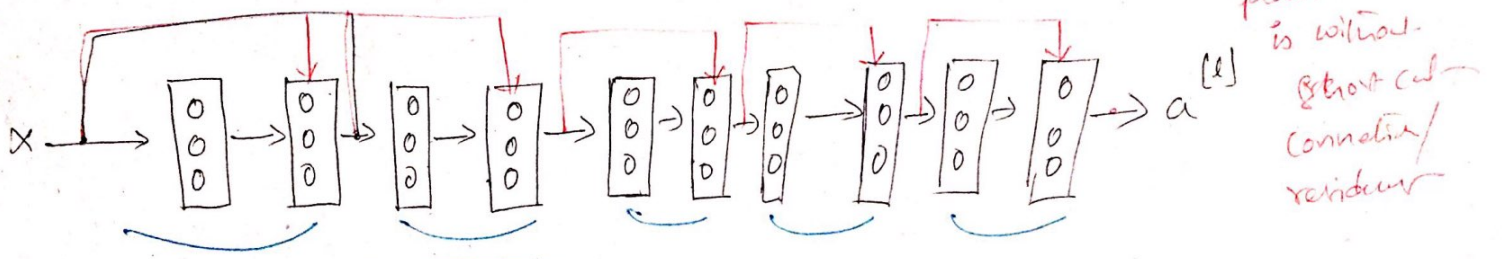


224x224x3 → CONV 64 x2 (2 layers) → 224x224x64 → POOL → 112x112x64 → Conv 128 x2 (2 layers) → 112x112x128 → Pool → 56x56x128 → Conv 256 x3 (3 layers) → 56x56x256 → pool → 28x28x256 → Conv512 x3 → 28x28x512 → pool → 14x14x512 → Conv 512 x3 → 14x14x512 → pool → 7x7x512 → FC 4096 → fc 4096 → Softmax 1000.

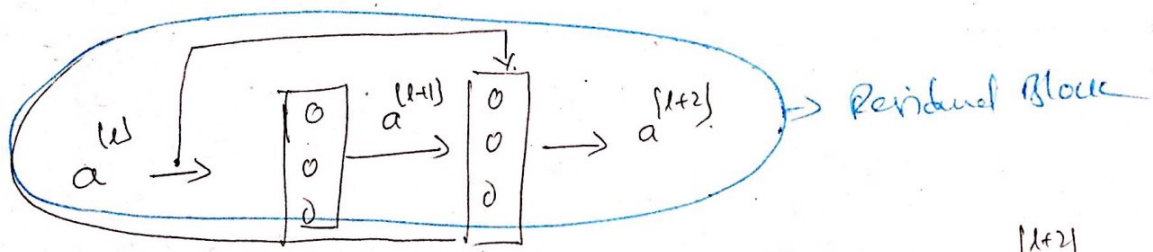
≃ 138 million parameters

Another improvement VGG-19

Residual Network : (ResNets)



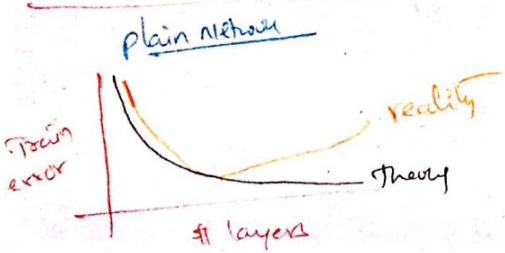
plain network is without Skip cut / connection / residuum

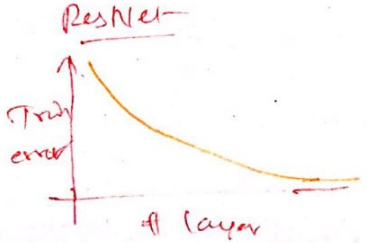


→ Residual Block

$a^{[l]}$ → linar → Relu. → $a^{[l+1]}$ → linear → (+) → Relu → $a^{[l+2]}$

Short cut / Skip Connection

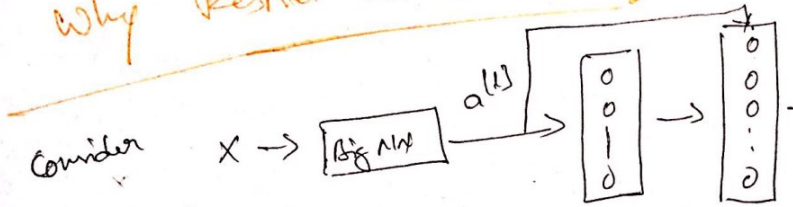




$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

Why ResNet works well ?

Consider $X \rightarrow$ [Big NN] $\rightarrow a^{[l]} \rightarrow \rightarrow a^{[l+2]}$



- as Relu results values ≥ 0. $\therefore a \geq 0$.

$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right) = g\left(\underbrace{w^{[l+2]}\, a^{[l+1]} + b^{[l+2]}}_{} + a^{[l]}\right)$$

because of L2 regularization if $w^{[l+2]} = 0$, $b^{[l+2]} = 0$.

$$a^{[l+2]} = g(a^{[l]}) = a^{[l]}$$

Summary

- Identity function is easy for residual block to learn.
- ~~easy~~ if $z^{[l+2]}$ (or $a^{[l+2]}$) and $a^{[l]}$ are of same dimension
- If not we multiply $a^{[l]}$ by another weight matrix W

  eg $a^{[l+2]} = 256 \times 1$ $\left[a^{[l]}_{128\times 1}, W_{256\times 128}\right]$ → new weight matrix