CNNs

→ majorly used in computer vision problems
- → image classification
- → object detection
- → Neural style transfer

A big challenge in CV is **inputs can get really big**.

for an image of 1000×1000 pixel image. (1MB)

the dimension of input features [x] becomes 1000×1000×3 = 3 million !!

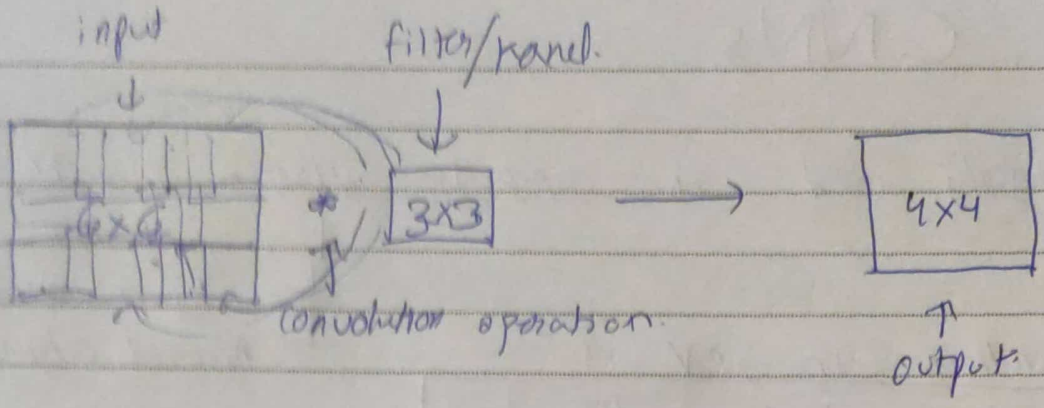
eg:-
for a fully connected network
- for 3M input features, even if your 1st layer contains just 1000 ~~not~~ neurons, the size of weight matrix is 3M×1000 = 3GB ← enormous.

→ with that much size of data, chances of overfitting increase v much, + computer, memory requirement increase.

→ that doesn't mean we can't use big images with NN
- → CNNs allow us to do so

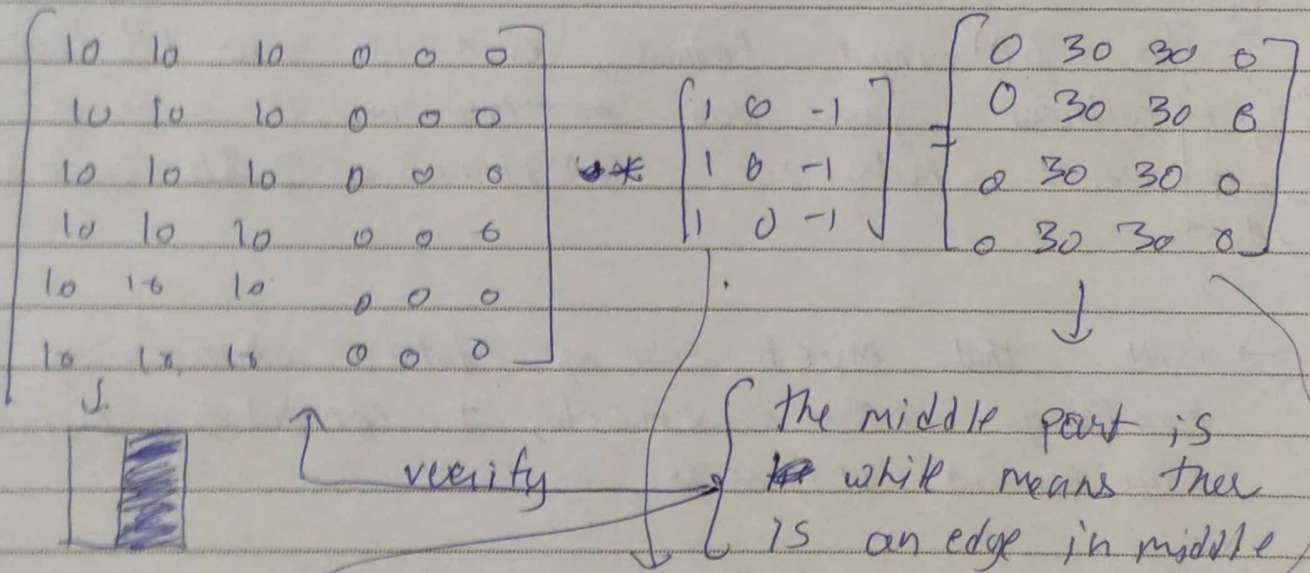
CNNs example :- Edge Detection

Convolution operation → { Big matrix (image.) x small matrix (generally) }
- ↓ gives the output image.
- ↓ elt wise product over the entire matrix.



* Different types of filters/kernels detect different features

eg:- $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ → detects vertical edge.



the detected edge looks thick bcz 6x6 is a v small image for larger size images its perf

→ what if input was $\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix}$ → output becoms $\begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$

* so the 1 on left and -1 on right signifies it only detects edges such that left is bright and right is dark.

if left of image is bright then $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ is the right kernel to use.

* similarly $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ detects horizontal edges such that ~~light~~ image is light on top and dark at bottom

* To detect edges $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ is a common filter

Sobel filter

puts more weight to middle part

→ diff types of filters give us diff properties

---

* → how long will you keep handpicking the right kernels?

→ you're not even sure what are "the right features" to detect to get the desired output

So you stop handpicking them and let the machine learn it.

* In CNNs the kernels are your weights

the machine "learns the right kernels" to apply which will give the required output

Input image * $\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$ = $\begin{bmatrix} \text{required} \\ \text{output} \end{bmatrix}$

↖ weights of your model

DOMS

PADDING

$6 \times 6 \quad * \quad 3 \times 3 \longrightarrow 4 \times 4$

$n \times n \quad * \quad f \times f \longrightarrow (n-f+1) \times (n-f+1)$

solution

→ so your image shrinks after every convolution operation

→ edge pixels dont contribute ~~much since they are~~ → you're throwing away relevant info from edges

→ you increase the size of input image

if you add p layers on top ~~and bottom~~ left, right

$n \times n$ padding → $(n+2p) * (n+2p) \quad * \quad f \times f \longrightarrow (n+2p-f+1) * (n+2p-f+1)$

what do you add in these p layers?

→ 0

Valid Convolution → no padding

Same Convolution → input size = output size

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

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

// f is usually odd.

↳ // even f doesnt give same convolutions

↳ // even f doesnt have a center

in CV its good to have a center to your filter

Strided Convolutions.

convolution operation means taking a kernel and sliding it over the input image.
at every step you take elt-wise product.

→ but how much do you "slide" your kernel by → Stride.
normally it's 1
but it can be whatever you want

// stride is same in both horizontal and vertical direction.

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

padding p
stride s

if not integer, take floor.
means, you never take your filter out of your image.

// in some math textbooks the convolution operation means rotating/flipping the kernel horizontally then vertically than taking the elt wise product over the entire image, but in dl it's just the last part //

also $(A * B) * C = A * (B * C)$ ✓ associativity

Convolutions Over Volume.

what if you've an RGB image instead of grayscale

$6 \times 6 \times 3$ ← height, width, no. of channels

6×6

3 must equal 3

→ now your filter changes from 3×3 to 3×3×3 !!
must with no. of channels

no. of channels in image must equal no. of channels in kernel

6×6×3 * 3×3×3 → 4×4
not 4×4×3

like you had 9 elts in an elt wise product
→ you summed up those 9 elt to get output elt in output

Now you'll have 27 elts in each elt wise product
you sum up those 27 elts to get corresponding elt for output.

// multiply the 1st layer of image with 1st layer of 1 channel input
2nd layer with 2nd layer of input
gives 9 elt gives 9 elt 3rd with 3rd gives 9 elt.

#Multiple filters

ofn you want to detect multiple features,
so you convolve the input with multiple filters

how do you get the output?

6×6×3 * 3×3×3 → 4×4 (to detect V edges or anything else)
* 3×3×3 → 4×4 (to detect horizontal edges or anything else)

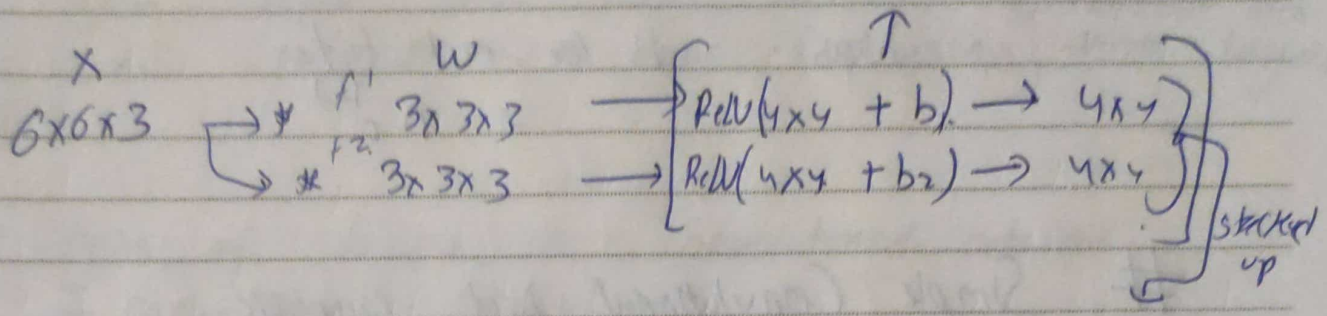
stacked up → 4×4×2 → output

can be generalised to n

just like a normal NN !!

$Relu(Wx + b)$

One Layer of CNN



4x4x2

output.

NOTATIONS:

If layer l is a convolution layer.

for layer l:

$f^{[l]}$ = 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 of prv layer

Output : $n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

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

same to $n_w^{[l]}$

$n_c^{[l]}$ = no. of filters you're using

Each filter is : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$ → must match no. of channels in input.

Activations : $a^{[l]} \rightarrow n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$A^{[l]}$ for m examples.

DOMS

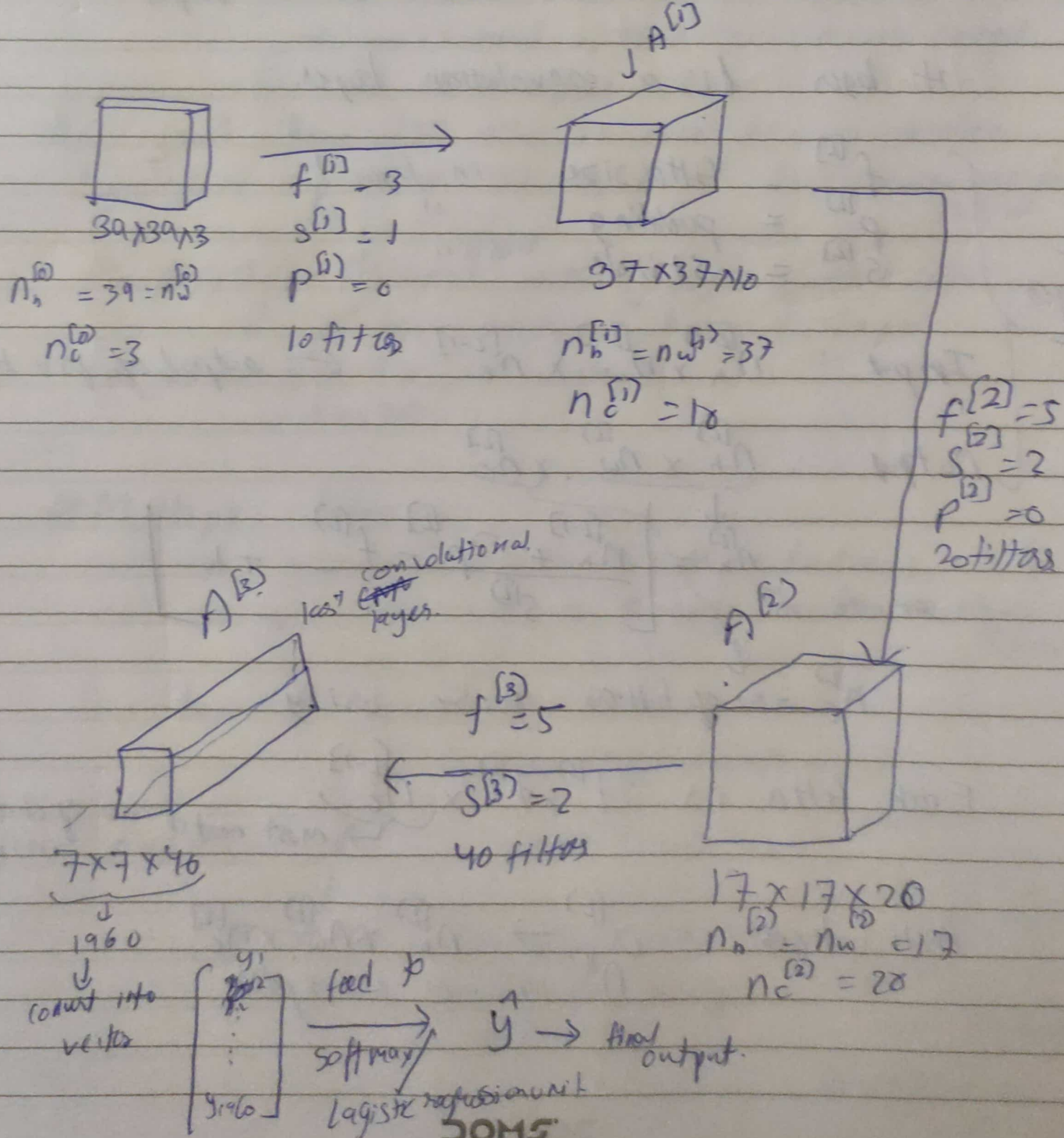
Weights : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

(size of each filter; no. of filters)

bias : 1 for each filter
$= n_c^{[l]}$

Simple Convolutional Net example.

~~dfdsfsi~~ cat classification

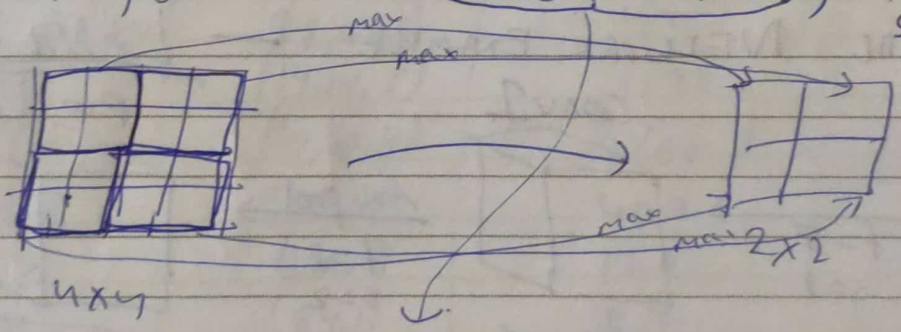


Usually in convolutional networks
your input size keeps decreasing } through each layer
and channel size keeps increasing }

types of layers in convolutional network.
- convolution (conv)
- pooling (pool)
- Fully connected (FC)

POOLING LAYERS.

Max pooling → you break your input into certain sections, then take each section's max



certain sections = filter size

for 4x4 → 2x2

hyperparameters { filter size = 2
stride = 2

nothing to learn.
// there are no parameters in Max pooling layers //

But what is max pooling doing ?

→ if a feature is detected in any filter
keep it preserved
→ also it "just works well"

Average pooling → you take avg, instead of max
↳ maybe not used much
used to collapse your network & in very deep networks

$n_h \times n_w \times n_c \xrightarrow{\text{pooling}} \left\lfloor \frac{n_h - f + 1}{s} \right\rfloor \times \left\lfloor \frac{n_w - f + 1}{s} \right\rfloor \times n_c$

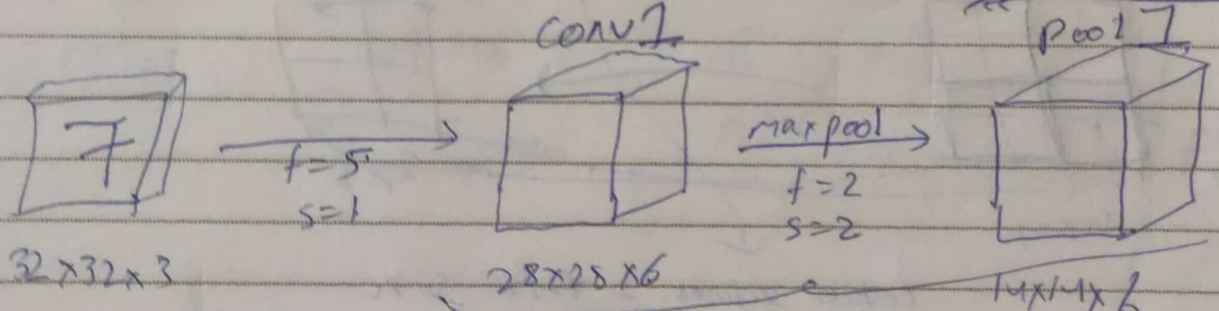
↓

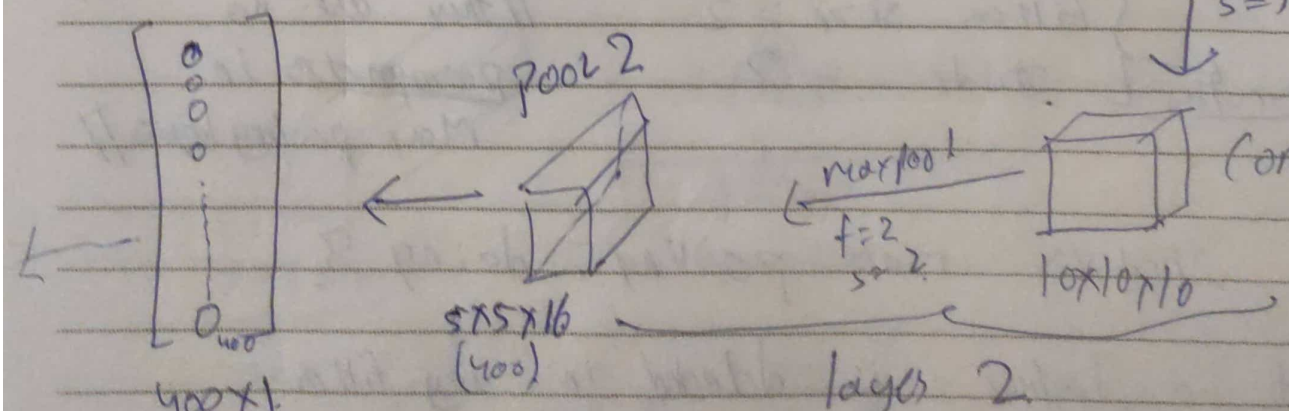
pooling is applied to each channel individually

// padding is very rarely used in pooling

usually pooling layers are not considered layers bcz they dont have any parameters

NEURAL NETWORK EXAMPLE — LeNet-5



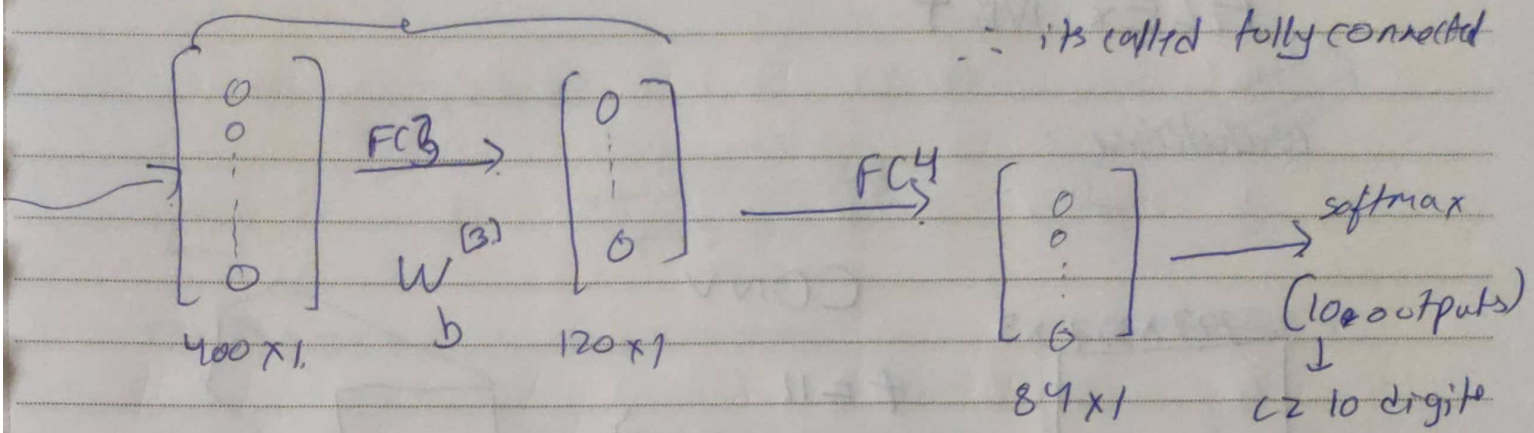


normal layers like we saw in NNs

$W^{[3]} = 120 \times 400$

bcz each unit is connected

∴ it's called fully connected



// alot of hyperparas!!

✱ Analysis:

[A.]

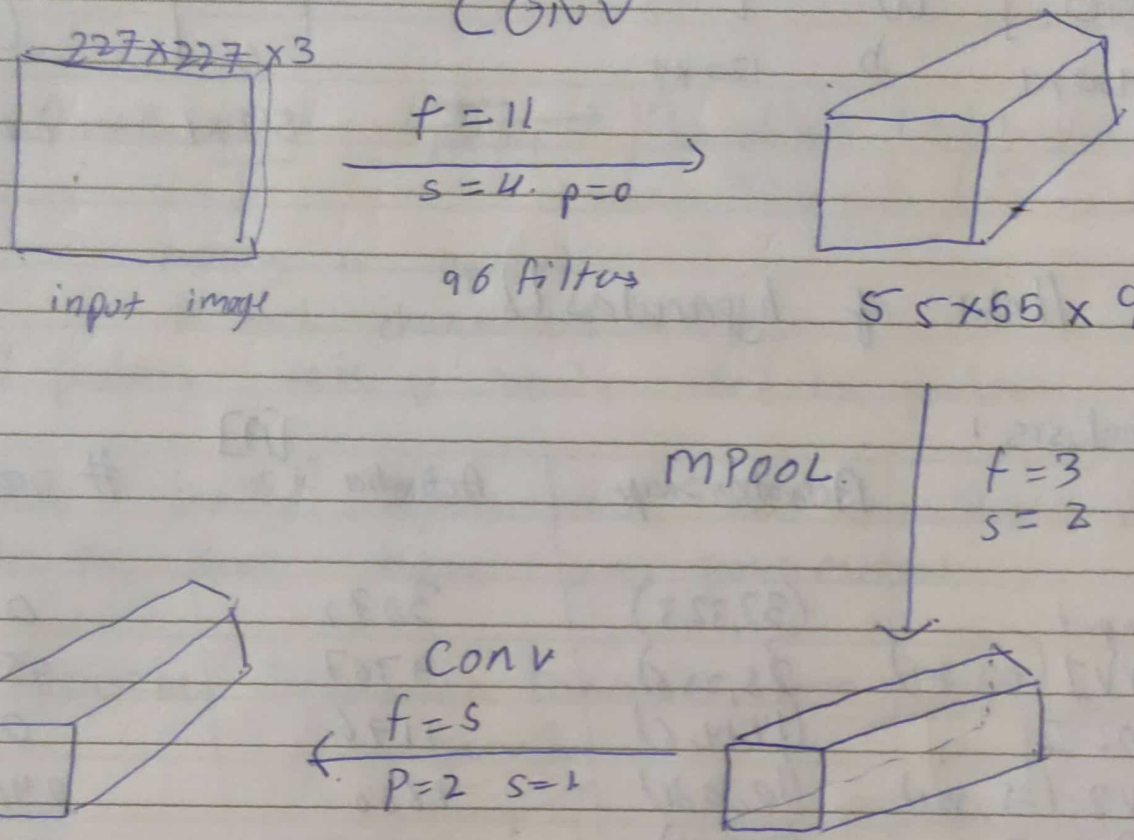
	Activation shape	Activation Size	# parameters
Input	(32,32,3)	3072	0
CONV1 (f=5 s=1)	(28,28,6)	4704	456
POOL 1	(14,14,6)	1176	0
CONV2 (f=5, s=1)	(10,10,16)	1600	2416
POOL 2	(5,5,16)	400	0
FC3	(120,1)	120	48210
FC4	(84,1)	84	10164
Softmax	(10,1)	10	850

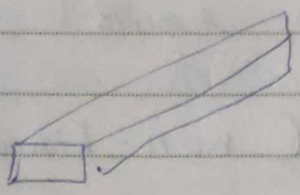
Advanced things:- → Sigmoid/Tanh was used as activations.

→ had non-linear activation after Pooling!!

ALEX NET

architecture:

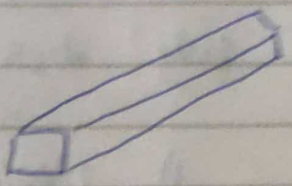




13 x 13 x 384.

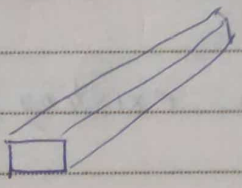
CONV
f=3 p=1
s=1

384 filters



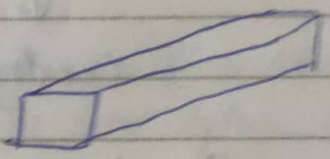
13 x 13 x 384

256 filters f=3 p=1 s=1 CONV



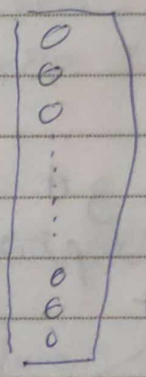
6 x 6 x 256 = 9216

MPOOL
f=3
s=2 p=0



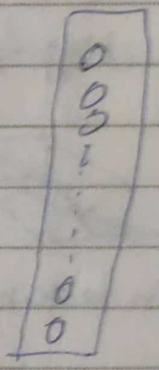
13 x 13 x 256

FC



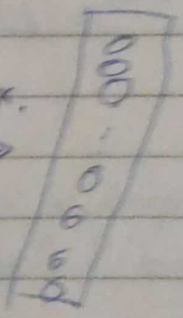
4096

FC



4096

softmax.
FC



1000 (output)

CONV → MPOOL → CONV → MPOOL → CONV → CONV → MPOOL ← CONV ← CONV

FC ← FC ← MPOOL

FC

output

5 conv + 3 FC + 3 MPool

have parameters

Why does Alexnet achieve better results?

1) Applied ReLU activation (non-linearity)

2) Standardization (local response normalization)

3) Dropout regularization

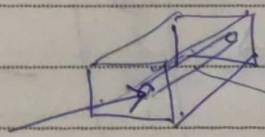
some advanced CNN stuff ←4) Data Augmentation (Enhanced Data)

↓

Creating more data from existing data

5) Local Respons Normalization.

LRN → suppose you've a 13x13x256 volume



→ to get a better representation of that pt

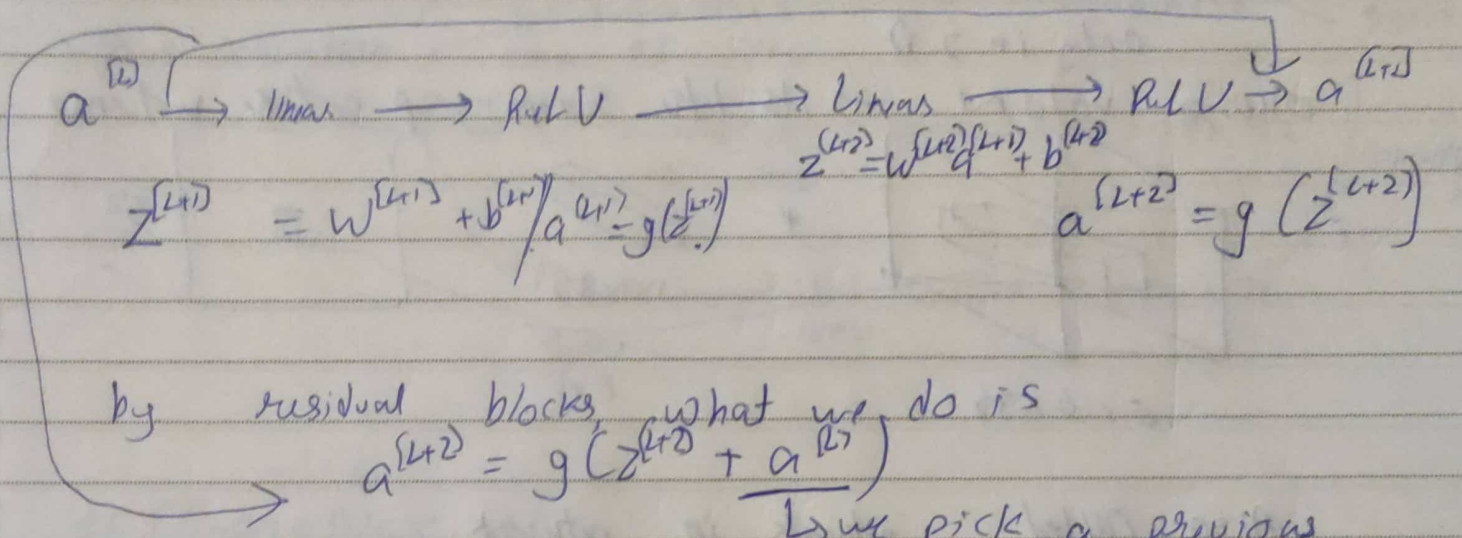
suppose you're looking at this point

We normalize that points value over this entire volume

Alex has a lots of hyperparameters

RESNETS

↳ build out of residual blocks



by residual blocks, what we do is

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

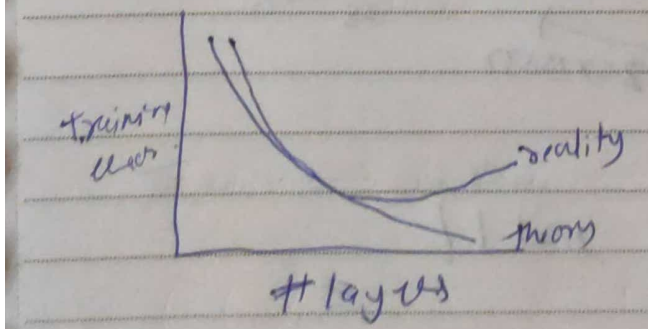
↳ we pick a previous layer and add it to some future layer

This is also called "shortcut/skip connection"

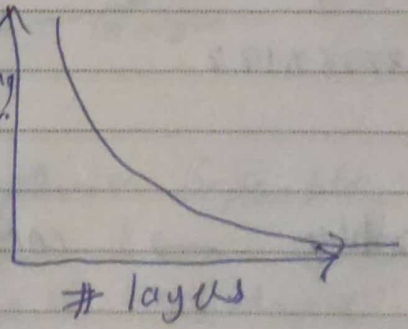
$a^{[l]}$ // skips a few middle layers and goes to $a^{[l+2]}$

→ Observation → Residual Network allow to train v deep models (100+)

Plain network



RESNETS



✱ Why do RESNETS work?

if using L2 regularizat^n → W, b will decrease ≈ 0 for some layers so res blocks act like identity f^n keeping $a^{[l+2]} \approx a^{[l]}$

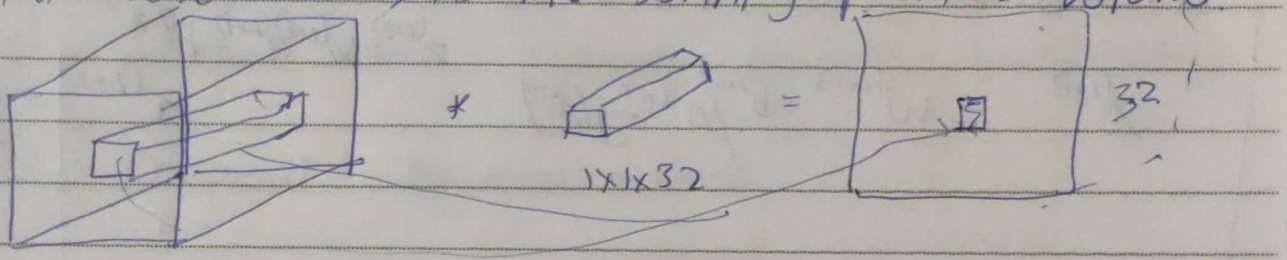
1×1 convolutions

→ looks like just multiplying by a number

↓

only in 2D

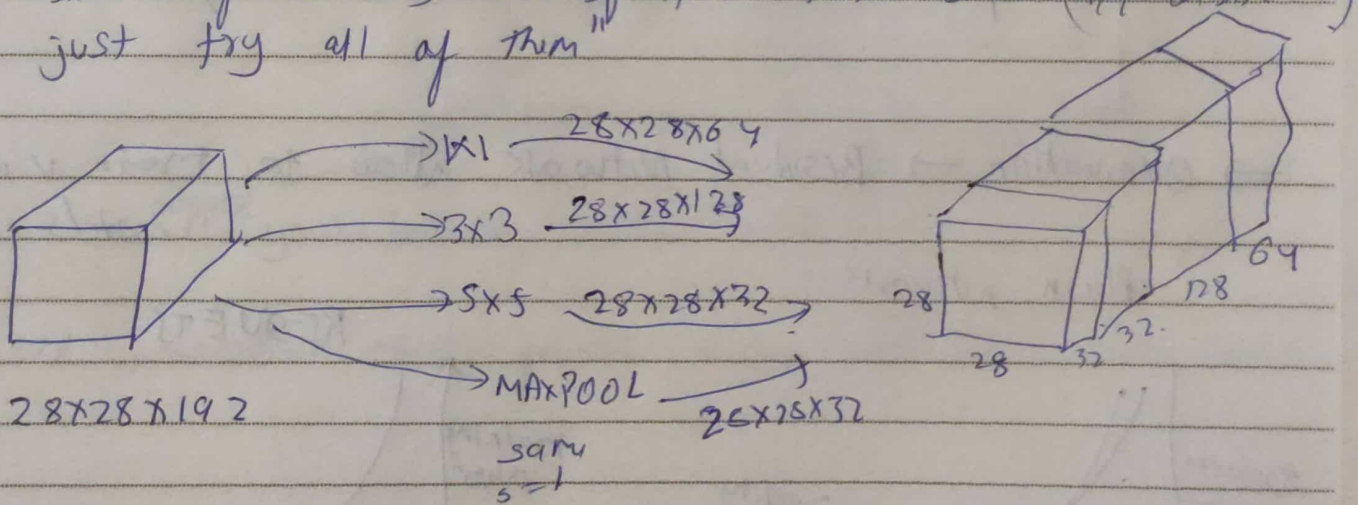
over a volume. → its like summing up entire volume.



Also called network in network.

Inception Network

"instead of chosing a specific filter size (1×1 or 3×3...) or which layer to apply just try all of them"

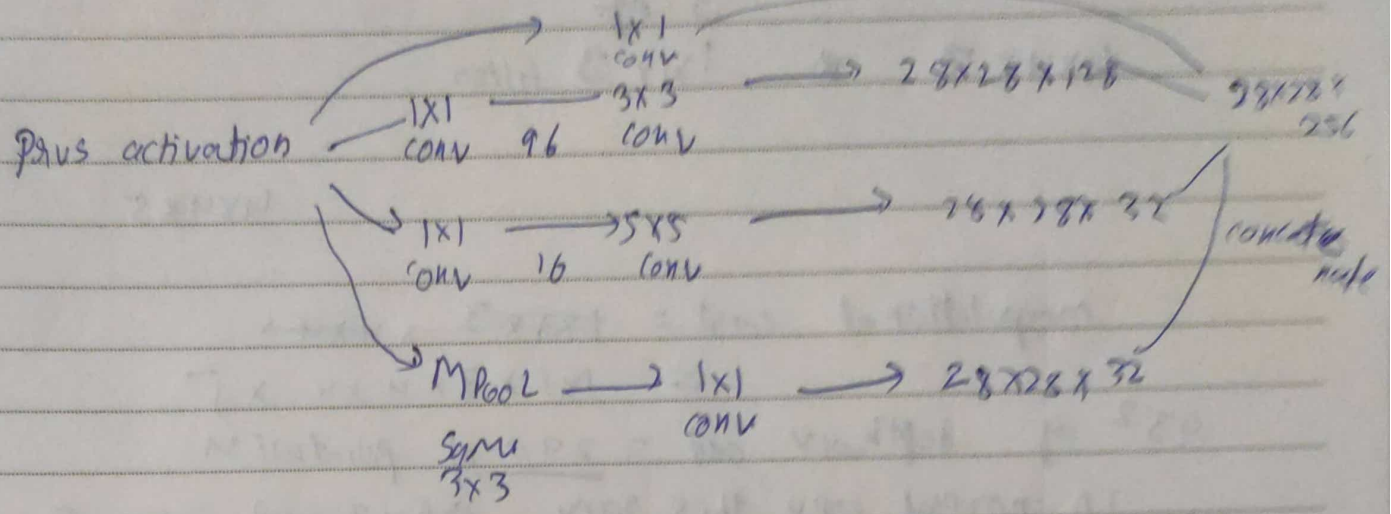


Problem → computational cost!!

1×1 convolution can be used to reduce volume. "Bottleneck layers"

28×28×192 —(1×1 conv, 16)→ 28×28×16

So for inception network



MOBILENET

↳ run NN architectures on less powerful computational devices.

In a normal convolution

computational cost = # filter params × # filter positions × # of filters

$$= f \times f \times n_c \times n_h \times n_w \times n_c'$$

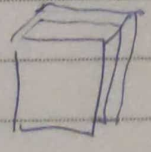
(n_c: in output; $n_h \times n_w$: in output; n_c': in output)

There is something called [Depthwise Separable Convolut°]

↓

depthwise + pointwise

depth wise.



$6 \times 6 \times 3$ * $3 \times 3 \times 3$ = $4 \times 4 \times 3$

are all same!!

this is the intuition convolut°

Pointwise.

5 of

4x4x3 * 1x1x3 filters

4x4x5

computational cost = ~~4x4x3 x 4x4x5~~

= 1x1x3x4x4x5

$4 3^2$ of depthwise and = 240 of pointwise

in normal conv this same 6x6x3 → 4x4x5

2160 computations were done

672 vs 2160

The ratio of depthwise to normal is

$$\frac{1}{n_c} + \frac{1}{f^2}$$