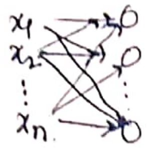
Convolutional Neural Networks :

* Computer vision :
  - inputs can be really big.



Ex: $1000 \times 1000 \times 3$ (RGB)

* Edge detection example :

~~earlier edges ex.~~

image → vertical edges
image → Horizontal edges

Ex: 6x6 grayscale img

elementwise mult & add

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}$$

→ -16

6x6

"convolution"

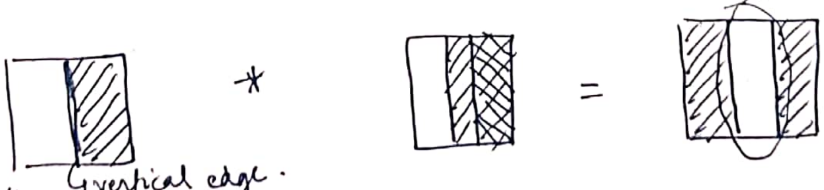
* (overloaded notation)

3x3 filter. → to detect vertical img (also called kernel)

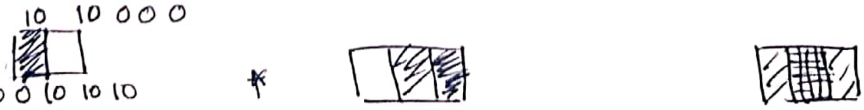
4x4.

→ vertical edge detector :
python : conv-forward.
tensorflow : tf.nn.con2d

Ex:



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



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

$$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$ → for horizontal edge.

filter

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

→ shows horizontal edges

$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} \qquad \begin{matrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{matrix}$$

sobel filter. scharr filter.

flip by 90° to get horizontal detection.

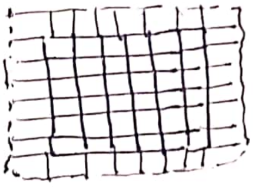
– we can also learn these parameters.

* Padding :

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

pixels on the corner are used less as compared to pixels in mid.

soln: "pad" the img



$6\times6 \rightarrow 8\times8 \; * \; 3\times3 \rightarrow 6\times6$

∴ image conserved.

p → padding amount.

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

Valid & same convolutions

"valid" (no padding) : $n\times n * f\times f \rightarrow n-f+1 \times n-f+1$

"same" : size of i/p = size of o/p.

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

$\boxed{p = \frac{f-1}{2}}$ → for same size o/p.

f is almost always odd.

* Strided convolutions :

instead of stepping the filter by 1 step, we step 2 steps.

$$\begin{bmatrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ \vdots \\ 0 & 1 & 3 & 9 & 2 & 1 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 91 & 100 & 83 \\ 69 & 91 & 127 \\ 44 & 72 & 74 \end{bmatrix}$$

$n \times n * f \times f$ padding p
stride 's'.

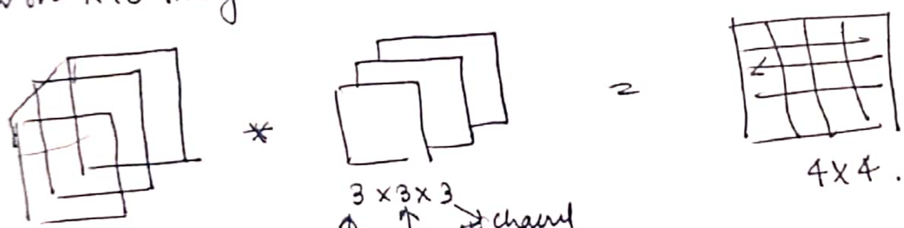
o/p. $\boxed{\frac{n+2p-f}{s} + 1 \times \frac{n+2p-f}{s} + 1}$

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

if blue box doesn't fit, then don't proceed.

here, we've been using cross corelation (most literature call this convolution)

* convolutions over volume :
conv on RGB images :



4x4.

6x6x3
height width #channel depth

3x3x3
height width #channel depth.

three 3x3x3 filter taken and placed in first slot, 27 nos multiplied and added, ...

R
1 0 -1
1 0 -1
1 0 -1

G
0 0 0
0 0 0
0 0 0

B
0 0 0
0 0 0
0 0 0

→ red edge detector.

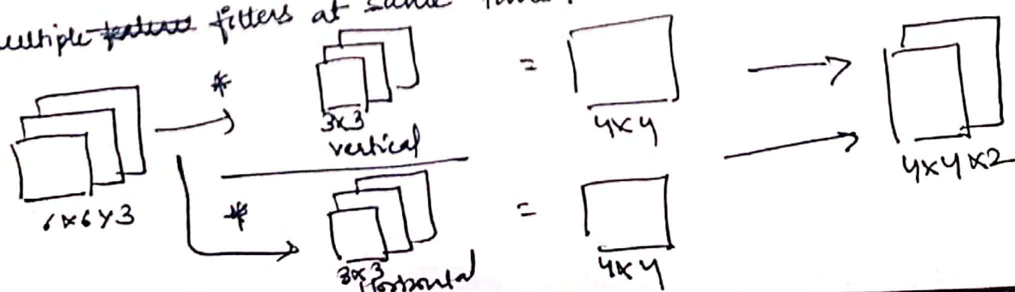
R
1 0 -1
1 0 -1
1 0 -1

G
1 0 -1
1 0 -1
1 0 -1

B
1 0 -1
1 0 -1
1 0 -1

→ any color edge detected.

• using multiple ~~features~~ filters at same time.

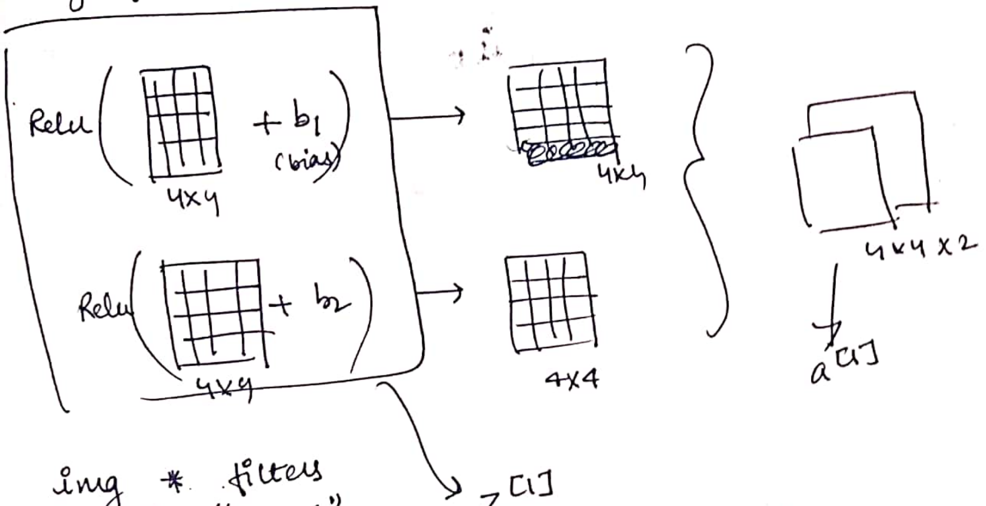


summary :

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

$\Downarrow$ # filters.

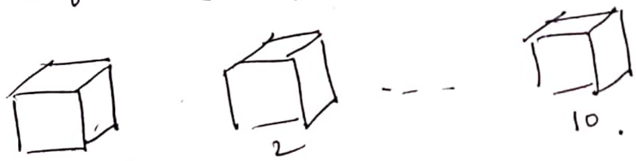
* one layer of a CNN .



If 10 filters, then $a^{[1]}$ had 4x4x10

img * filters $\rightarrow z^{[1]}$
"$a^{[0]}$" "$w^{[1]}$"

Ex : if 10 filters (3x3x3)



27 params
+ bias
= 28 params

$\therefore$ 280 parameters .

If layer l is a convolution layer:

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

$p^{[l]}$ = padding

$s^{[l]}$ = stride

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

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

H → height
W → width .

Output : $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

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

activations $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

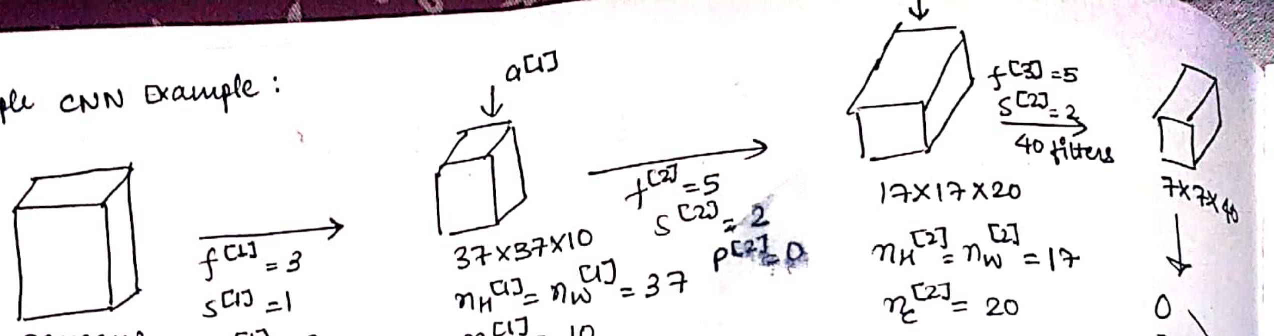
$A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

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

↳ # filters in layer 'l'.

bias : $n_c^{[l]}$ – $(1, 1, 1, n_c^{[l]})$

* Simple CNN Example :



$39 \times 39 \times 3$ $\xrightarrow{}$ $f^{[1]} = 3$, $s^{[1]} = 1$, $p^{[1]} = 0$, 10 filters

$n_H^{[0]} = n_W^{[0]} = 39$

$n_c^{[0]} = 3$

$a^{[1]}$: $37 \times 37 \times 10$

$n_H^{[1]} = n_W^{[1]} = 37$

$n_c^{[1]} = 10$

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

$$\frac{39+0-3}{1}+1 = 37.$$

$\xrightarrow{}$ $f^{[2]} = 5$, $s^{[2]} = 2$, $p^{[2]} = 0$

$17 \times 17 \times 20$

$n_H^{[2]} = n_W^{[2]} = 17$

$n_c^{[2]} = 20$

$$\frac{37+0-5}{2} = 17$$

$f^{[3]} = 5$, $s^{[2]} = 2$, 40 filters

$7 \times 7 \times 40$

0
0
0
0
⋮
0

$\therefore$ only 1 long vector fed into softmax.

- Types of layers in CNN
  - Convolution (CONV)
  - Pooling (POOL)
  - Fully connected (FC)

1.* Pooling layers :

- Max pooling

$$\begin{bmatrix} 1 & 3 & 2 & 1 \\ 2 & 9 & 1 & 1 \\ 1 & 3 & 2 & 3 \\ 5 & 6 & 1 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 9 & 2 \\ 6 & 3 \end{bmatrix}$$

4×4 → 2×2

max of 1 region

Hyperparams
- $f = 2$ (like a filter size = 2)
- $S = 2$ (stride)

No parameters to learn.

max pool : if feature detected, keep the high number.

max value for each patch of feature map.

$$\begin{bmatrix} 1 & 3 & 2 & 1 & 3 \\ 2 & 9 & 1 & 1 & 5 \\ 1 & 3 & 2 & 3 & 2 \\ 8 & 3 & 5 & 1 & 0 \\ 5 & 6 & 1 & 2 & 9 \end{bmatrix} \longrightarrow \begin{bmatrix} 9 & 9 & 5 \\ 9 & 9 & 5 \\ 8 & 6 & 9 \end{bmatrix}$$

$f = 3$

$S = 1$.

if $5 \times 5 \times 2$ o/p is $3 \times 3 \times 2$.

do same computation on previous slice.

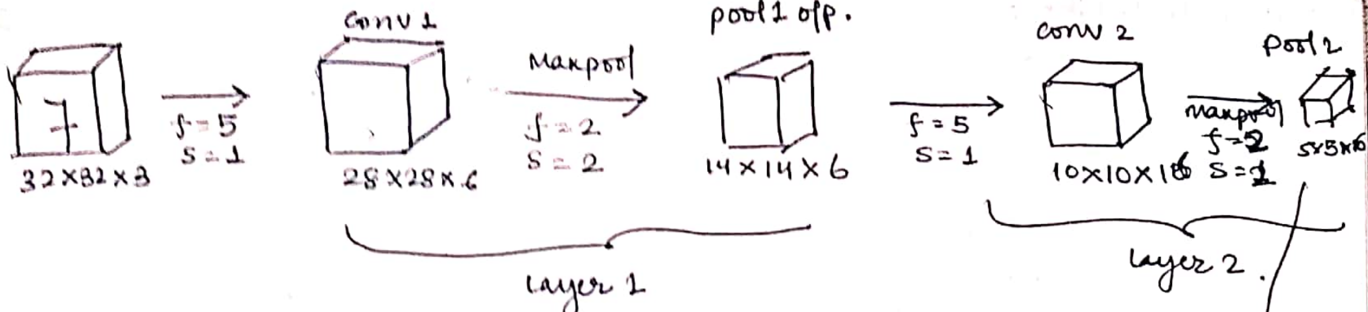
- Average pooling

average of ~~pooling~~ values in the patch of feature size.

Hyperparams
- f: filter size
- s: stride
- Max or average pooling.
- p: padding.

no of i/p channels = no of o/p channels

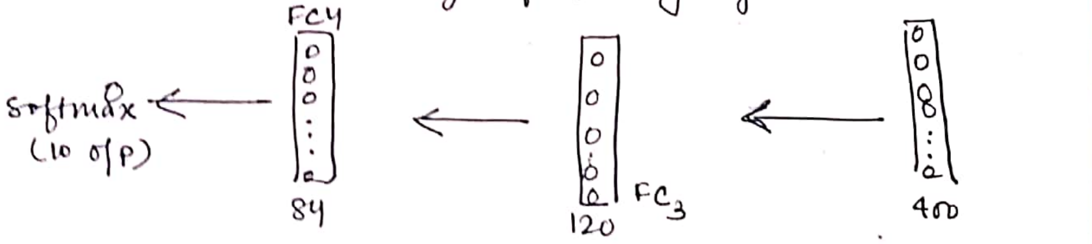
* example of complex CNN (LeNet-5)



as no of layers have weights & params.

weight for pooling layer = 0

5×5×16 = 400



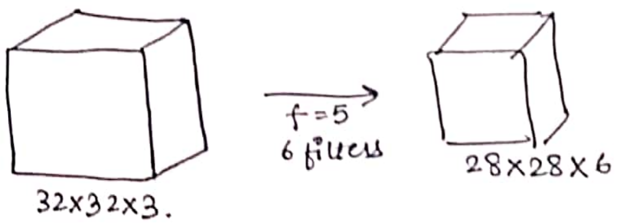
	Activ^n shape	Size	# params
input:	(32,32,3)	3072	0
CONV1 (f=5, s=1)	(28,28,8)	6272	208
			⋮

$w^{[3]} = (120 \times 400)$

$b^{[3]} = (120 \times 1)$

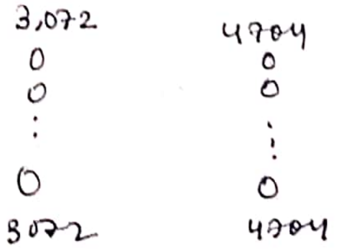
⋮

* why convolutions?



filter
5x5 = 25 + bias
= 26 params
6 x 26 = 156 params
↓
filters.

weight matrix.

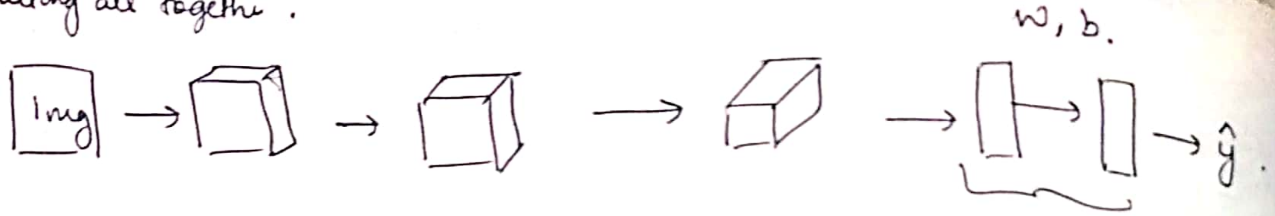


3072x4704 ≈ 14M

Parameters sharing: A feature detector (ex vertical edge detector) that's useful in 1 part of img is prolly useful in other part.

Sparsity of connection: in each layer o/p depends only on ~~small~~ a small no of i/p.

Putting all together.



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

use gradient descent to optimise params to reduce J.

Deep convolutional models: case study.

* why look at case studies :-

Classic network

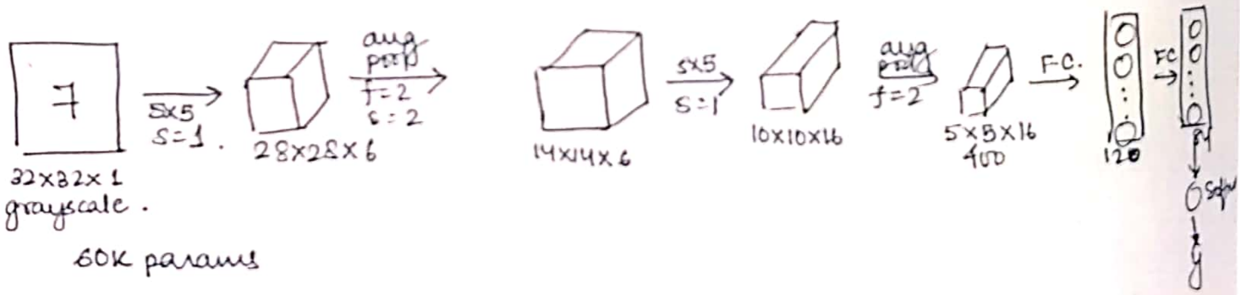
- LeNet-5
- AlexNet
- VGG

ResNet (Residual Network)

- 152 layers deep.

* Classic Networks :

- LeNet-5



60K params

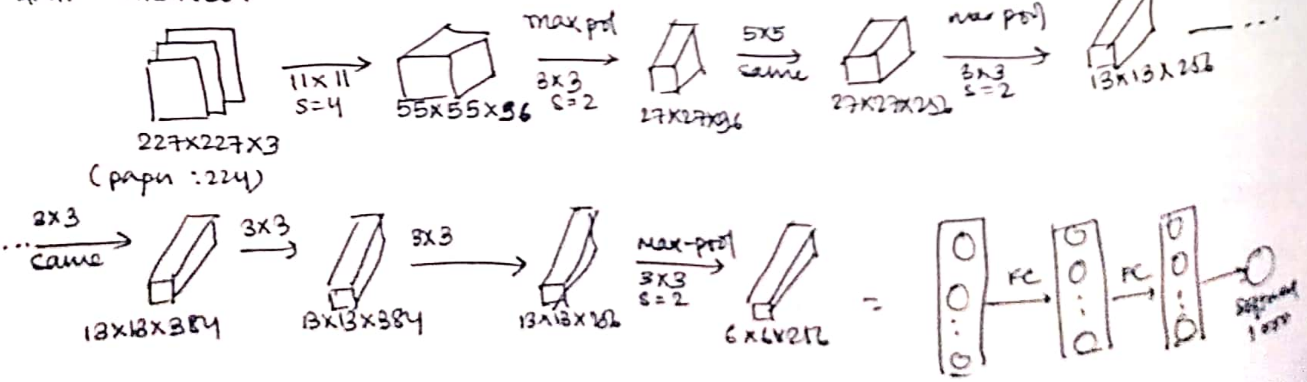
$n_H, n_W \downarrow \quad n_C \uparrow$

conv → pool → conv → pool → fc → fc → output.

Sigmoid/tanh on the research paper.

Le-Net5 had non linearity after pooling. (sections II & III)

- AlexNet.



- much bigger
- 60M Parameters
- Relu $activ^n$ function • when this was written, GPUs were slower.
- Has a local Response normalisation.

• VGG-16 :

conv = 3x3 filter s=1, same   Maxpool = 2x2, s=2

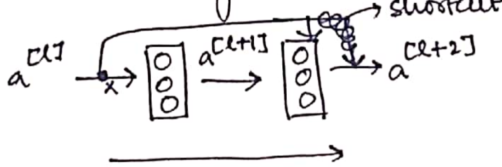
224x224x3 → [conv64 x2] → 224x224x64 → MPool s=2 → 112x112x64 → [conv128 x2] → 112x112x128 → pool → 56x56x128 → conv256 x3 → 56x56x256 → pool → 28x28x256 → conv512 x3 → 28x28x512 → pool → 14x14x512 → conv512 x3 → 14x14x512 → pool → 7x7x512 → FC 4096 → FC 4096 → softmax 1000

138 M parameters . very large network.

notes

*ResNets (Residual N/ws)

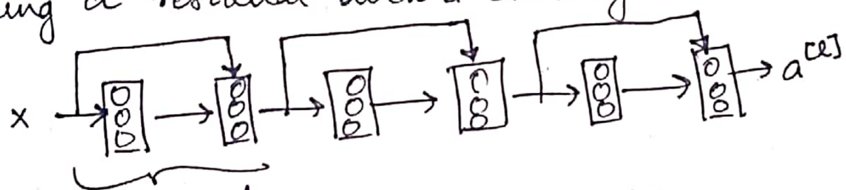
skip connections : taking a layer & connecting it to a much deeper layer in the NN.

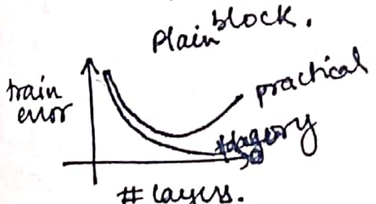


$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

main path.

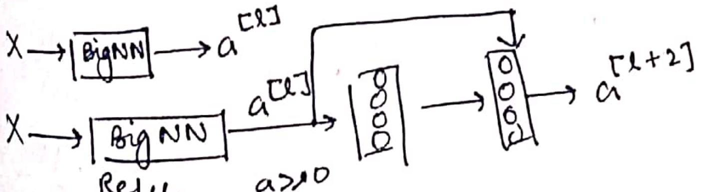
Taking a residual block & stacking them.





Helps with vanishing & exploding gradient problem.

Why ResNets work?



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

$= g(w^{[l+2]} a^{[l+2]} + b^{[l+2]} + a^{[l]}) = g(a^{[l]}) = a^{[l]}$ (if $w^{[l+2]} = 0$, $b^{[l+2]} = 0$)

identity $func^n$ is easy for residual block to learn.

$z^{[l+2]}, a^{[l]}$ have same dimensions ∴ same convolutions, so that dimensions are preserved.

o/p is 256 & i/p is 128 dim;
- add W_s (to normalise dimension)

* Networks in Networks & 1x1 convolutions:

6x6 * $\boxed{2}$ = 6x6. (each no doubled)

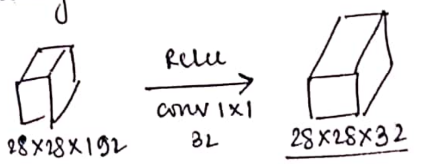
6x6x32 *  1x1x32 = ReLU 6x6x#filters.

32 → #filters $n_c^{[l+1]}$

Network in network.

1x1 conv filter can be used to change dimensionality in filter space.

* • using 1x1 convolutions.

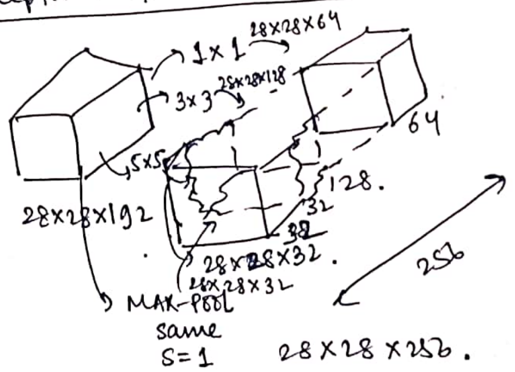


to ↓se n_c $\quad$ n_H, n_W, n_c.

1x1x192
#
each filter.

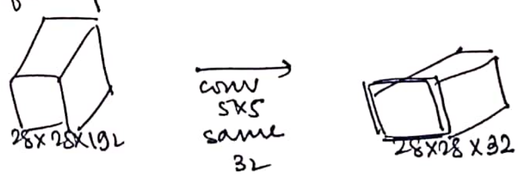
∴ no of channels can be ↓sed by 1x1 conv. ∴ reducing computation.

* Inception N/W motivation:



28x28x256.

Problem of comp. cost:

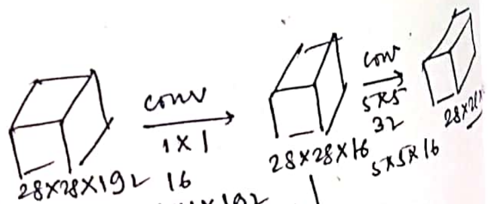


32 filters $\quad$ filters are 5x5x192

$(28\times28\times32)\times5\times5\times192 = \underline{\underline{120M}}$.

costly.

alternative architecture (saves computation)



smaller intermediate val (bottleneck layer)

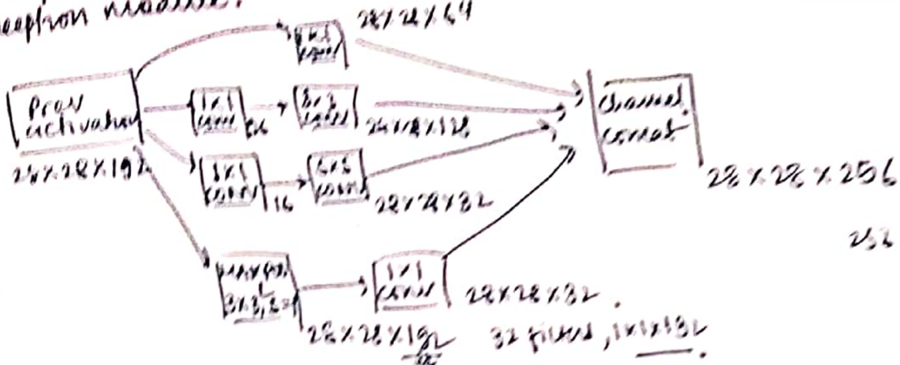
1st $(28\times28\times16)(1\times1\times192) = 2.4M$

2nd $(28\times28\times32)\times(5\times5\times16)$ filters $= 10.0M$

∴ $\underline{\underline{12.4M}}$.

∴ 12.4M << 120M

* Inception module:



256 = 64 + 128 + 32 + 32

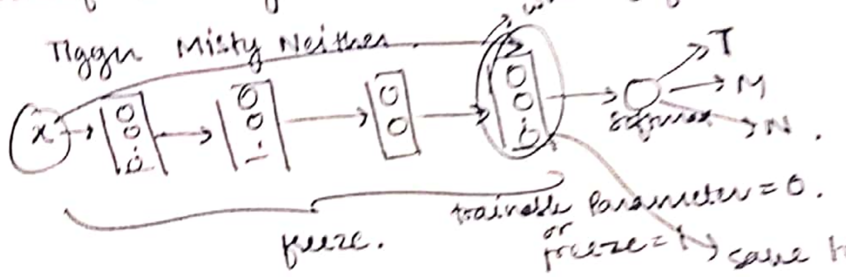
28×28×192 → 1×1 → # channels

32 filters, 1×1×192.

GoogLeNet

* Using open source implementation:
- github.

* Transfer learning:-



when initial layers are frozen, directly transfer

→ if smaller data set, freeze more layers.

if larger data set, freeze lesser layers, train later layers.

if more larger data set, train whole NN, as we take all weights to initialize rather than random initialization.

* Data Augmentation:

method:

mostly used. { - mirroring
- random cropping (isn't perfect, but works well)
- rotation
- shearing
- local warping

- color shifting: add different distortions to RGB channels.

Ex: R G B
↓ ↓ ↓
+20 −20 +20

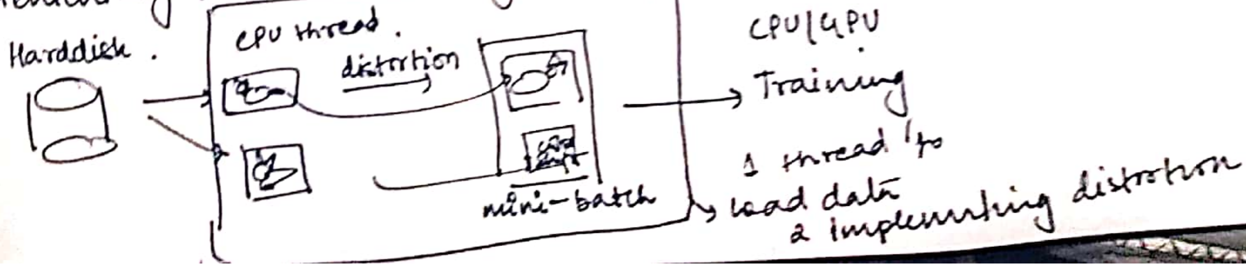
- PCA (Principle component Analysis) color Augmentations.

⇓ AlexNet paper

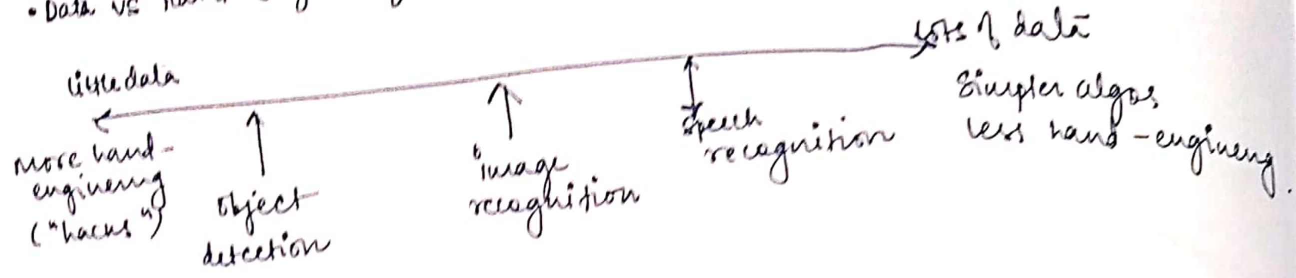
RB → subtracts lot to RB

G → subtracts less for G.

- Implementing distortions during training.



* State of computer Vision:
• Data vs Hand engineering.



↑ Transfer learning:

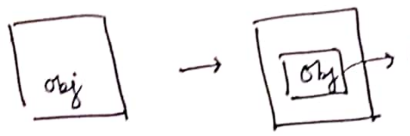
• Tips for doing well on benchmarks / winning competitions
— Ensembling
→ Train several N/w & average their o/p.

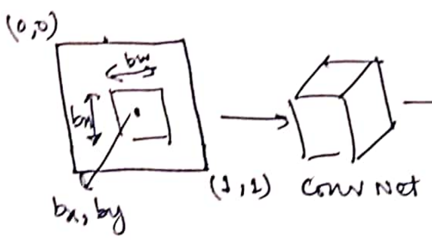
— Multicrop at test time
— Run classifier on multiple version of test img & average result.
Ex: 10-crop.

— use open source code.
• use architecture of N/w published in literature
• use pretrained models & fine-tune on your dataset.

⊕ Object Detection:
* Object localization:

Classification prob : 1 ~~prob~~.obj
Detection prob : multiple obj.





→ O softmax (4)
1 – ped
2 – car
3 – cycle
4 – background.

→ b_x, b_y, b_H, b_w "bounding box" parameters.

• Defining Tagt label 'y'.
1 – ped
2 – car
3 – motorcycle
4 – background.

Need to output b_x, b_y, b_H, b_w, class label (1-4).

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_H \\ b_w \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

is there an obj? (0 for background only)

if no obj, $P_c = 0$
rest are don't care

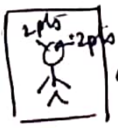
Loss: $L(\hat{y}, y)$

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

if $P_c = 1$

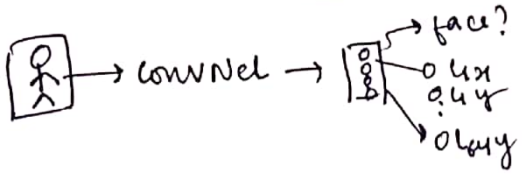
\# landmark detection

$l_{1x}, l_{1y}, l_{2x}, l_{2y}$



maybe 64 landmarks on face.

l_{1x}, l_{1y}
l_{2x}, l_{2y}
⋮
l_{64x}, l_{64y}



129 units.
~~params~~
1 for face or not
128 landmarks (64x, 64y)

Ex: snapchat filters.

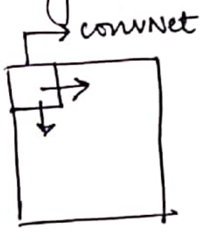
Ex: car detection.

x	y
car	1
car2	1
No car	0
⋮	⋮

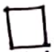
car → convNet → y

Data set has cropped car images.

• sliding windos detection.



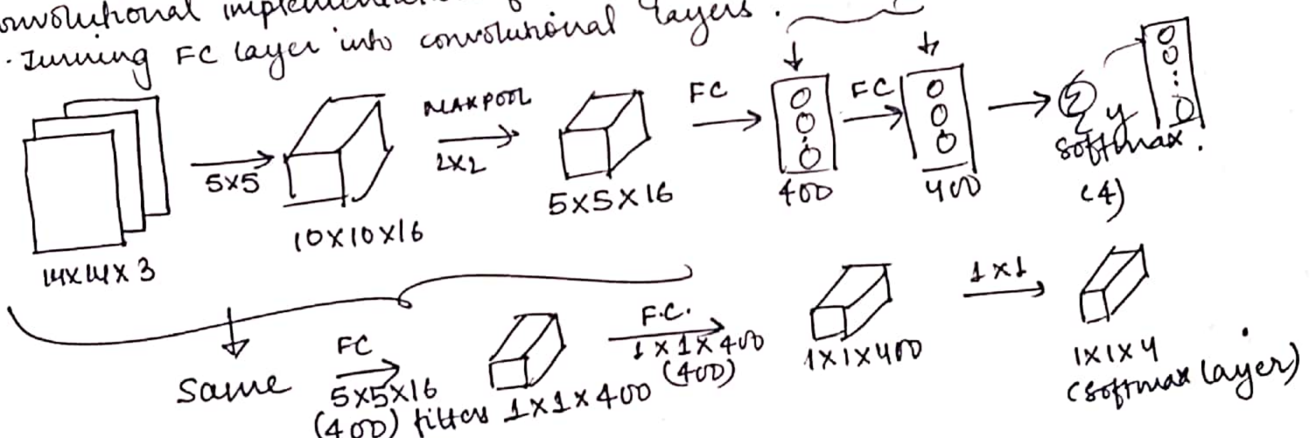
repeat using a larger window, may do 3rd time, even larger window. There will be eventually a window which contains a car.



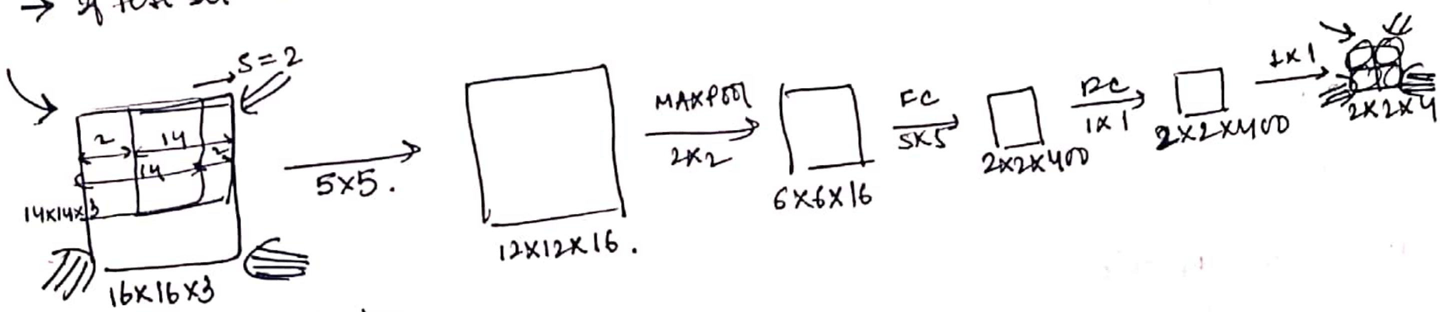
sliding window. (faster if large slide)

\# convolutional implementation of sliding window :-

· Turning FC layer into convolutional layers. → To convNet.



→ If test set has 16x16x3 img.



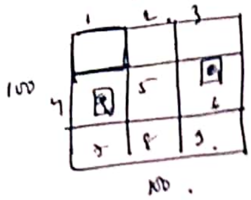
∴ share computation.

28x28x3 $\xrightarrow{5\times5}$ 24x24x16 $\xrightarrow[2\times2]{\text{MAXPOOL}}$ 12x12x16 $\xrightarrow{5\times5}$ 8x8x400 $\xrightarrow{1\times1}$ 8x8x400 $\xrightarrow{1\times1}$ 8x8x4

*Bounding box predictions:
– As ConvNet sliding window pred. doesn't predict bounding boxes accuracy.

• YOLO algorithm (You Only look Once)

3x3 grid.



labels for traing

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_H \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Yolo research papers.

for 1, 2, 3, 5, 7, 8, 9.

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

– midpt of object determines grid cell.

$$4: \; y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_H \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad 6: \; y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_H \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Trgt o/p: $\underbrace{3 \times 3}_{\text{grids}} \times \underbrace{8}_{\text{8 units in o/p.}}$

3x3x8.

∴  $\xrightarrow{\text{conv}}$ conv layer → MAXPOOL - - - - →  (y).

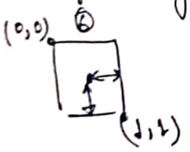
100x100x3

3x3x8

↑se 3x3 so that (grid cell has) obj.

• Specify bounding boxes:

Ex: for 6

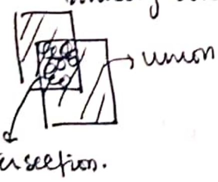


$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_H \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

b_x – 0.4 } b/w 0 & 1
b_y – 0.3 }
b_H – 90% blue line ∴ 0.9 }
b_w – 0.5 (50% of total grid) } could be > 1.

* Intersection Over Union:

(IoU)



$$\boxed{\text{IoU} = \frac{\text{Size of intersection}}{\text{Size of union}}}$$

"Correct" if IoU ≥ 0.5 (threshold)

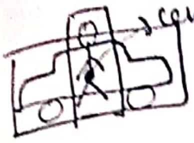
accuracy ∝ IoU.

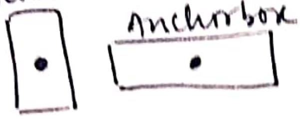
*Non-max suppression.

– makes sure that an object is detection only once.
– cleans up multiple detections
– rectangles with highest IoU part are highlighted, rest suppressed.
– Discard all boxes with $p_c \leq 0.6$
  – while any remaining boxes
    pick the one with highest p_c, discard the rest.

*Anchor Boxes:

• overlapping objects: Anchor box 1 Anchor box 2





$$y = \left.\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ \vdots \\ c_3 \end{bmatrix}\right. \begin{matrix} \text{anchor box 1} \\ \text{anchor box 2} \end{matrix}$$

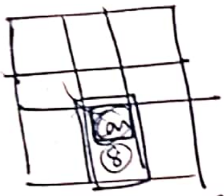
Now, o/p 'y' becomes 3×3×16

With 2 anchor boxes:
- Each obj in Tr. image is assigned to grid cell that contains object's midpt & anchor box for grid cell with highest IoU.

If only car:
- for anchor box 2: specify parameters
  for anchor box 1: $p_c = 0$, rest are don't care.

*YOLO Algorithm:

Training 1 - Ped
2 - car
3 - cycle.



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

for '8'. $\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ \vdots \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$ That's why 1 is empty, as 1 is bigger

y is 3×3×2×8
↑ #anchors
5 + #classes

• Outputting non max suppression o/p:
  • For each grid, get 2 predicted bounding boxes
  • get rid of low probability predictions
  • For each class, (ped, car, motorcycle) use non max suppression to generate final predictions.

* Region proposals: R-CNN.
  - pick just a few regions with objects.
  - runs a segmentation algo.
  - Output label + bounding box.

Fast R-CNN
- use convolution implementation of sliding windows to classify all proposed regions.

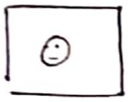
⊕ Facial Recognition :-

* Face recog:
* What is face recognition :
  - Face verification vs Face recognition

    Verification
    - I/p name / ID
    - Output whether i/p img is that person or not.

    Recognition
    - Database of k persons
    - get an i/p img
    - Output ID if img is of any of the k persons (or "not recognised")

* One Shot learning :
  - learn from one example to recognise the person again.



→ CNN → O softmax (5) : Not very good approach.

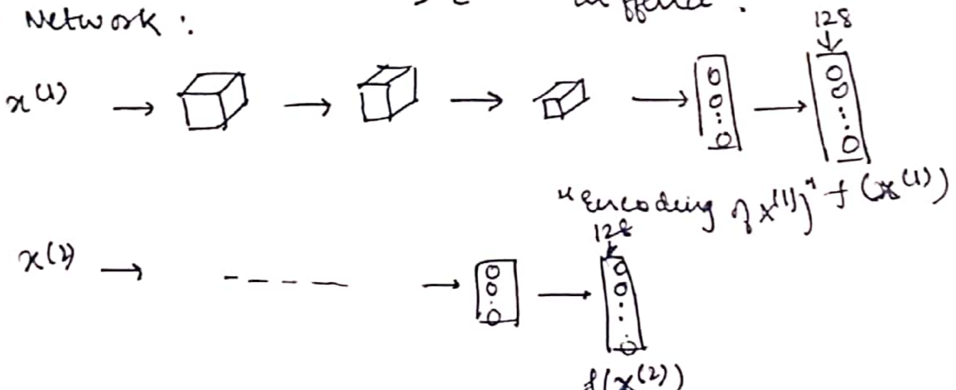
  - good approach : learn similarity f^n

    → $d(img1, img2)$ = degree of difference b/w images

    if $d(img1, img2) \le \tau$ same

    $> \tau$ difference.

* Siamese Network :



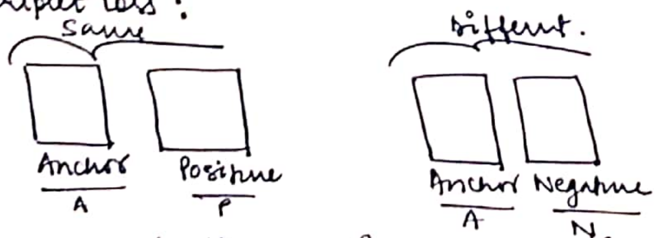
$$d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_2^2$$

- Deepface closing the gap to human level performance

if $x^{(i)}$ & $x^{(j)}$ are same person : $\| f(x^{(i)}) - f(x^{(j)}) \|^2$ small

else large.

* Triplet loss :



want : $\underbrace{\| f(A) - f(P) \|^2}_{d(A,P)} \le \underbrace{\| f(A) - f(N) \|^2}_{d(A,N)}$

$$\| f(A) - f(P) \|^2 - \| f(A) - f(N) \|^2 \le 0 - \alpha$$

α = hyperparam that prevents NN from outputting trivial sol^n.

or

~~$d(A) = d(P)$~~ '

Given 3 images $\overrightarrow{A, P, N}$

$$L(A,P,N) = \max\left(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0\right)$$

$$J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$$

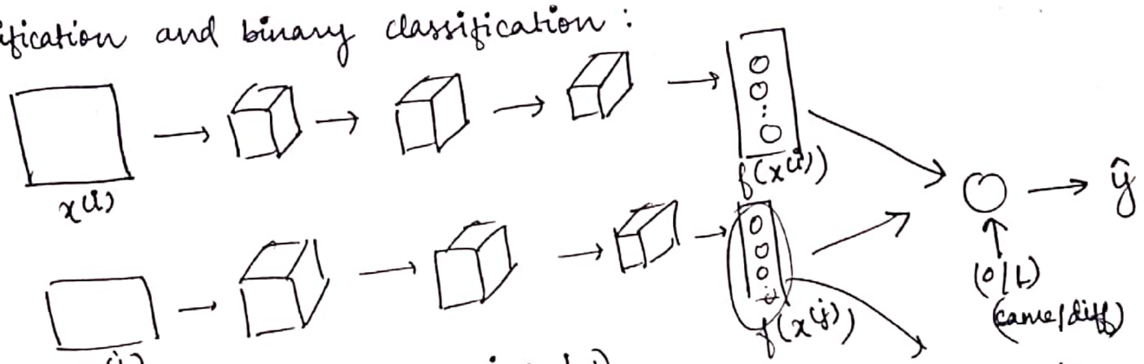
Paper: FaceNet.

Training set: 10k pics of 1k persons

- If A, P, N is chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is easily satisfied.
- chose triples that are "hard" to train on

$$d(A,P) + \alpha \leq d(A,N)$$
$$d(A,P) \approx d(A,N)$$

* Face verification and binary classification:



$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

k^th encoding

$f(x^{(i)})$ & $f(x^{(j)})$ vector

or

$$\frac{\left(f(x^{(i)})_k - f(x^{(j)})_k\right)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$$

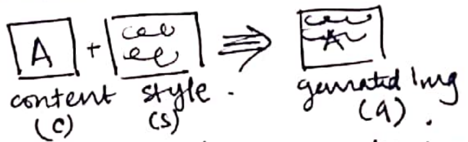
χ^2 (chi square formula)

precompute.

upper convnet computes encoding & precomputed is compared with.

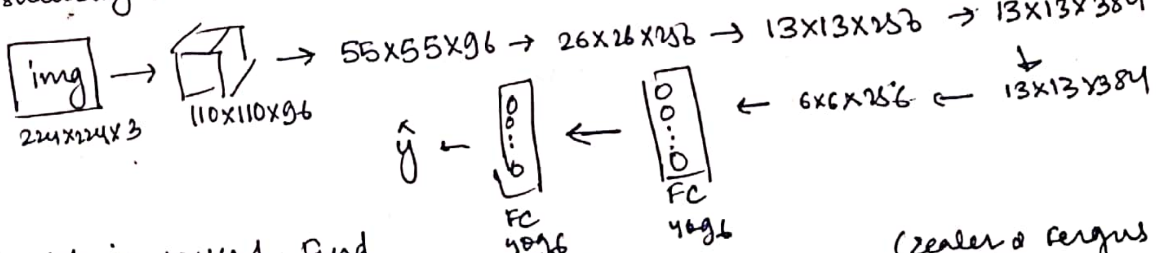
Neural Style Transfer.

* What is Neural Style Transfer?



* What are deep convnets learning?

- visualizing:



- Pick a unit in layer 1. Find the nine img patches that maximise unit's activation

- Repeat.

(Zeiler & Fergus Visualizing & understanding convolutional networks.

* Cost function :

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

How similar is content — How similar is cost

(A neural algo of artistic style — Gatys et al.

α, β – Hyperparams .

• Finding generated image 'G'.

1. Initialise G randomly
$G : 100 \times 100 \times 3$

2. use G.D. to minimise $J(G)$.
$$G := G - \frac{\partial}{\partial G} J(G).$$

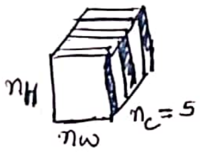
* Content cost function : $(J(C, G))$

• say you use 'l' to compute content cost.
• use pre-trained ConvNet (Eg: VGG N/w)
• let $a^{[l](C)}$ and $a^{[l](G)}$ be activation of layer l on imgs.
• if $a^{[l](C)}$ & $a^{[l](G)}$ are similar, both have similar content.

$$\therefore J_{content}(C, G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

* Style cost func^n : $(J(S, G))$

• 'l' to measure "style".



How correlated are activations across chanels ?

let channel 1 corresponds to sth, chanel 2 corresponds to sth, then check how much they are correlated.

• style matrix

let $a^{[l]}_{ijk}$ = activation at (i, j, k). $G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$ (H, W, C)

$G^{[l]}_{kk'}$, $k = 1 \ldots n_c^{[l]}$ (no of channels)

$$G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{ijk} \, a^{[l](S)}_{ijk'} .$$

do this for every val of k' & k to complete style matrix

$$G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{ijk} \, a^{[l](G)}_{ijk'}$$

"gram matrix"

$$J^{[l]}_{style}(S, G) = \| G^{[l](S)} - G^{[l](G)} \|^2_F$$

$$= \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} (G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'})^2$$

$$\boxed{J^{[l]}_{style}(S, G) = \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_c^{[l]})^2} \sum_k \sum_{k'} (G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'})}$$

$$J_{style}(S, G) = \sum_l \lambda^{[l]} J^{[l]}_{style}(S, G)$$

↑ hyperparameter

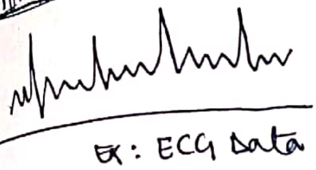
1D and 3D generalisations:

• conv in 2D & 1D.

1	20	15	3	18	2	4

$*$

= 10 dimensional



Ex: ECG Data

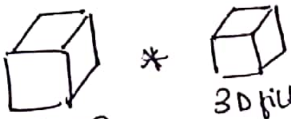
14

1	3	10	3	1

5

3D data:

Ex: CT scan.



3D vol $*$ 3D filter

$\underset{H}{14} \times \underset{W}{14} \times \underset{D}{14} \times 1 * 5 \times 5 \times 5 \times 1 \xrightarrow{16 \text{ filter}} 10 \times 10 \times 10$ volume $\rightarrow 10 \times 10 \times 10 \times 16$

1 → #chanels

1 → #chanls.