Course 4 (CNN)

5/4/18

Deep learning computer vision is helpful in self driving cars, better face recognition.

Applications: Image classifica^n, object detec^n, Neural style transfer
(content image + style image)

Problem $\bar{c}$ CNN:

If i/p image is (1000×1000) resolu^n image, then i/p features = 1000×1000×3 = 3 m i/p features. If hidden layer with 1000 units is used;

$w^{[1]}$ = (1000, 3 million) dimensional matrix i.e 3 billion parameters

So, many parameters lead to overfitting in absence of sufficient amount of data. Also, memory and computational requirements of 3 billion parameters is quite infeasible. We use convolu^n computation to solve this problem.

→ Edge detecⁿ example to understand convoluⁿ operation:

Note A gray scale image (matrix) is just 2-D bcoz no RGB codes. Every pixel has either a value 1 or 0 i.e black or white. [or may be some other nos. that are shades of black and white]

The convoluⁿ operaⁿ detects vertical & horizontal lines in the i/p image.

e.g.

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

(6x6) i/p image

(a)

* "convoluⁿ"

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

3x3 filter/kernel

(b)

=

-5	-4	0	8
10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4x4 vertical edge output.

(c)

C_{11} is computed by doing element wise multiplicaⁿ of (a) and (b) and adding all the 9 terms.
(1st window)

To get C_{12}, shift window 1 step to the right and repeat above procedure. [i.e. $(3\times1)+(1\times1)+(2\times1)+(0\times0)+(5\times0)+(7\times0)+(1\times-1)+(8\times-1)+(2\times-1)] = -4$

Hence, a 6x6 matrix convolve of 3x3 matrix gives 4x4 matrix.

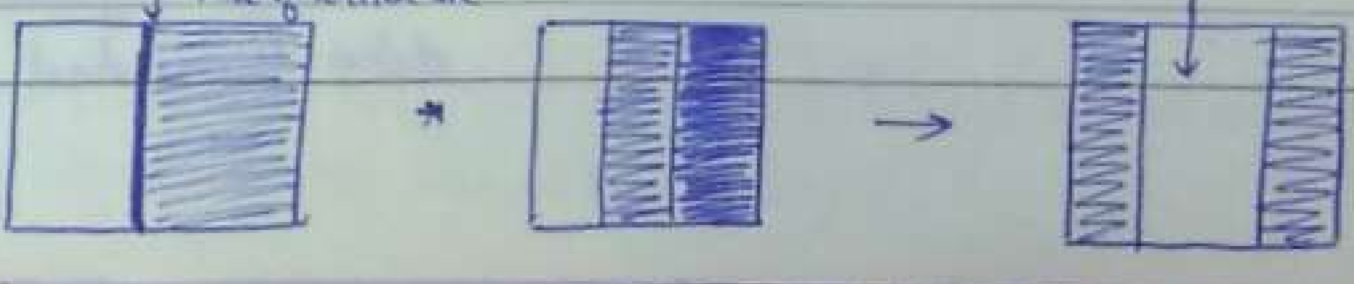
To implement the above convoluⁿ fn:

in python we have: conv-forward

in tensorflow " " : tf.nn.conv2d

in Keras " " : Conv2D

Note Above works as a vertical edge detector. This can be shown through the example below.



i.e.

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

*

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

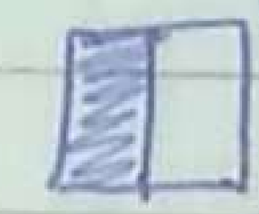
(edge detec^n filter)

→

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

(vertical edge detector)

This indicates the p̊nce of a strong vertical line in middle of the image. The dimensions seem to be a little bit wrong bcoz the i/p image is just 6x6. If it was of higher dimension, better results would be obtained.

Note Here, +30 shows that there is a light to dark transition.
If  was the i/p image, we would have got -30 showing transi^n from dark to light.
We can take absolute of 30 if we don't care which of the two cases it is.

Also

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

Horizontal edge detector

Note

1	0	-1
2	0	-2
1	0	-1

(Sobel filter)
(another vertical edge detector giving more weightage to the central row making it more robust)

3	0	-3
10	0	-10
3	0	-3

(Schars filter)

Note for very big images, these 9 values of the filter are learnt through back propagation and treated as parameters. They can not only be used to detect vertical & horizontal edges but edges at some angles as well like 45° or 73° or so on. This learning happens through data provided as i/p.

→ Padding: modificaⁿ to convolⁿ operaⁿ.

$(n\times n) \underset{\downarrow}{*} (f\times f) \rightarrow ((n-f+1)\times(n-f+1))$

convolve

- One drawback in what we were doing is that the pixels at the corners and the edges are considered only once as only 1 filter covers it while the pixels at the center are covered by many $(f\times f)$ filters which make us to ignore a lot of info near the edge or corner.
- Other drawback is that the output obtained is a shrinked image, and so if on every hidden layer it will shrink this way, we will get an extremely compressed image as the final output.

To fix these problems, we can pad the image with an addition of border of 'p' pixel all around the image.

In case of 6x6, $p=1$ as $(n-f+1) = 6$ i.e same sized image as output
$(8-3+1)$ {$n = 6+2$, bcoz of padding}

Convolution < Valid convoluⁿ (no padding)
Same convoluⁿ (padding such that o/p size = i/p size)

i.e $n+2p-f+1 = n \rightarrow \boxed{p = \frac{f-1}{2}}$

{f = filter size}.
f is almost always odd in computer vision conventⁿ.
∴ So, that p is not fractional and we have a central pixel to refer.

→ Strided convolution:

Stride = 2 means the f×f filter that is used is hopped by 2 steps instead of 1 in the n×n input image.

$$(n\times n) \underset{\text{convolve}}{*} (f\times f) \xrightarrow[\text{Stride 's'}]{\text{Padding 'p'}} \left(\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]\right)$$

↳ floor is taken if it is not an integer.

Note the $f\times f$ filter must lie completely inside the image while hopping over it to do the computations.

Note Cross correlation v/s convolution

What we have doing till now was cross correl^n. Convol^n in actual involves one more step i.e flipping of the filter matrix.

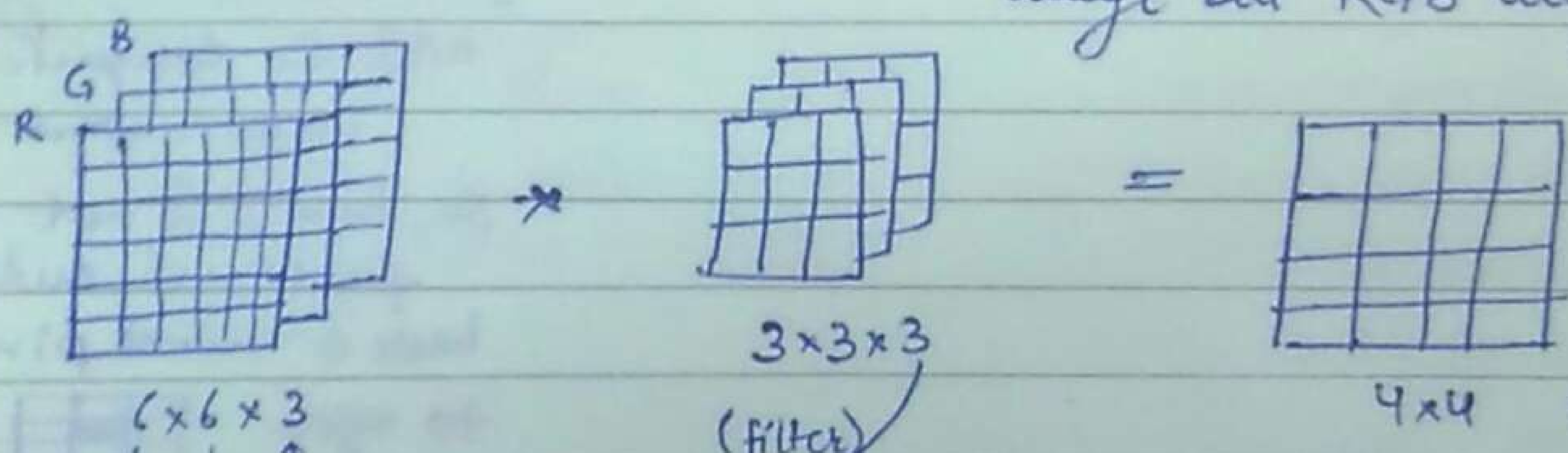
$$\begin{bmatrix} 3 & 4 & 5 \\ 1 & 0 & 2 \\ -1 & 9 & 7 \end{bmatrix} \longrightarrow \begin{bmatrix} 7 & 2 & 5 \\ 9 & 0 & 4 \\ -1 & 1 & 3 \end{bmatrix}$$

(Narrowing of filter both on vertical and horizontal axis)

The element wise product and summing is actually done after this step but generally we skip this step and call that also as convol^n instead of cross correl^n.

$\Rightarrow (A*B)*C = A*(B*C)$ Associativity of convol^n.

→ Convolutions over volumes: (to detect feature not in a grey scale image but RGB image)



$6\times 6\times 3$ (height, width, #channels or depth) (i/p image)

$3\times 3\times 3$ (filter)

these 2 nos. need to be same.

4×4 (no 3 in the end) (2D output)

This $3\times3\times3$ cube is swiped over the i/p image and this time 27 nos. are correspondingly multiplied and added to give a number of the 4×4 o/p matrix. Similar to the previous grey scale image convolu^n. Only difference is that it is in 3D.

To detect vertical edges only in the red channel, $3\times3\times3$ filter is like:

$$\underset{R}{\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix}} \quad \underset{G}{\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}} \quad \underset{B}{\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}}$$

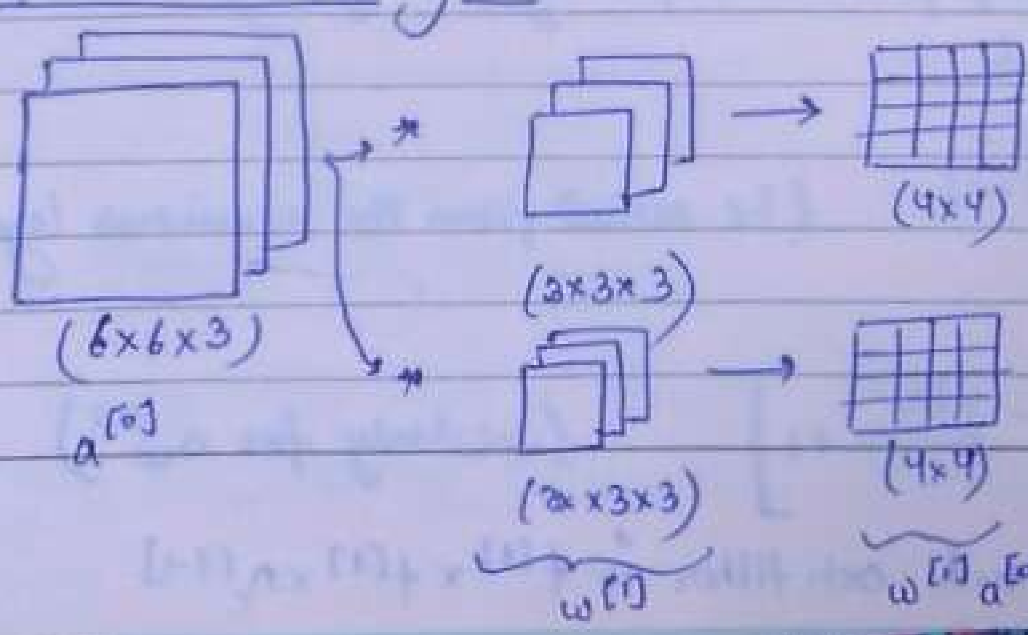
$$(n\times n\times n_c) * (f\times f\times n_c) \longrightarrow (n-f+1)\times(n-f+1)\times n_c'$$

→ no. of filters used.

(e.g. horizontal filter + vertical filter, then $n_c = 2$) 1st channel giving horizontal edges and 2nd channel giving vertical edges detected in the i/p image.

<u>Note</u> Adv. of convolution on volume is that now we can use multiple filters thus detecting multiple features in the input image. No. of filters used = no. of channels in o/p image.

→ <u>Convolution NN layer</u>:



$\text{Relu}(w^{[1]} a^{[0]} + b_1^{\rightarrow \in \mathbb{R}}) \longrightarrow$ 4×4

$\text{Relu}(w^{[1]} a^{[0]} + b_2) \longrightarrow$ 4×4

$\underbrace{\qquad\qquad\qquad}$

$\text{Relu}(z^{[1]}) = a^{[1]}$

$>$ 4×4×(2) $\longrightarrow$ depends on no. of filters taken

(corresponding to 2 filters used)

Hence, activaⁿ for next layer obtained.

Convoluⁿ step was basically the linear operaⁿ step i.e $W^{[1]}X$, then bias was added giving $z^{[1]}$ and then $\sigma(z')$ gave $a^{[1]}$.

Q. calculate no. of parameters if we have 10 filters that are 3×3×3 in 1 layer of a NN.

Ans $((3\times3\times3) + \underset{\text{bias}}{\underset{\downarrow}{1}}) \times 10 = 280$ parameters.

Note No matter how big the i/p image is, no. of parameters = 280 is fixed for 10 3×3×3 filters. This property saves CNN from overfitting.
↳ to extract different features from images.

Notations:

$f^{[l]}$ = filter size (i.e $f\times f$ filter used for layer l of NN)

$p^{[l]}$ = padding in layer l

$s^{[l]}$ = stride in layer l.

Input : $n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$ {i.e activaⁿ from the previous layer}

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

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

$n_c^{[l]}$ = no. of filters . and each filter is $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

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

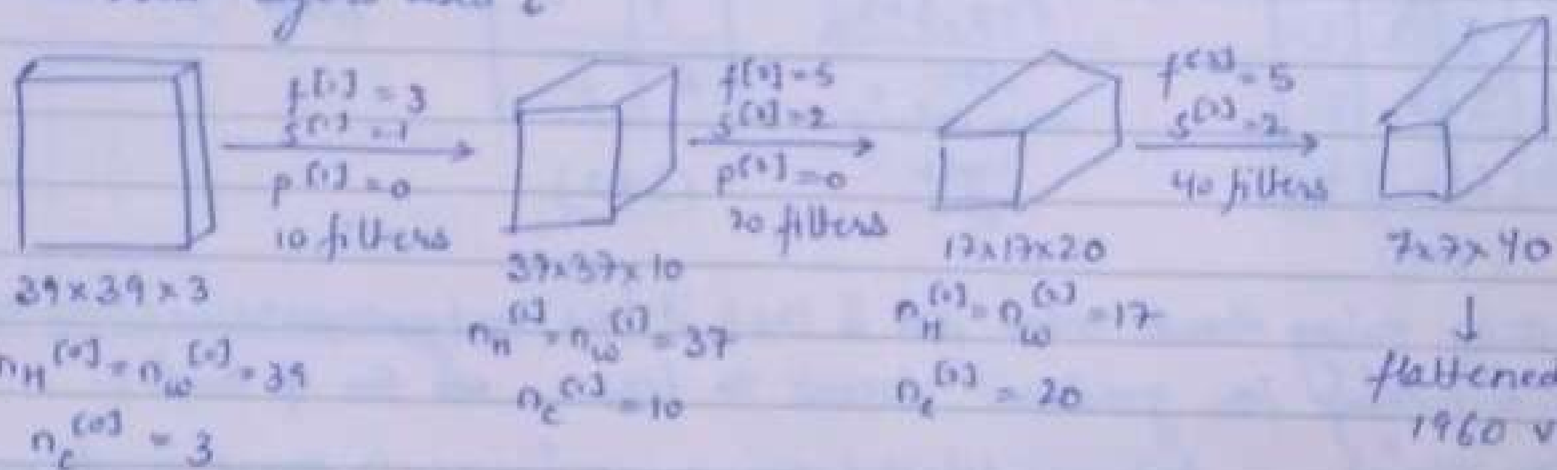
$A^{[l]} = m \times a^{[l]}$ {for m inputs}.

weights : $\underbrace{f^{[l]} \times f^{[l]} \times n_c^{[l-1]}}_{\text{dimension of filter}} \times \underbrace{n_c^{[l]}}_{\text{\# filter}}$

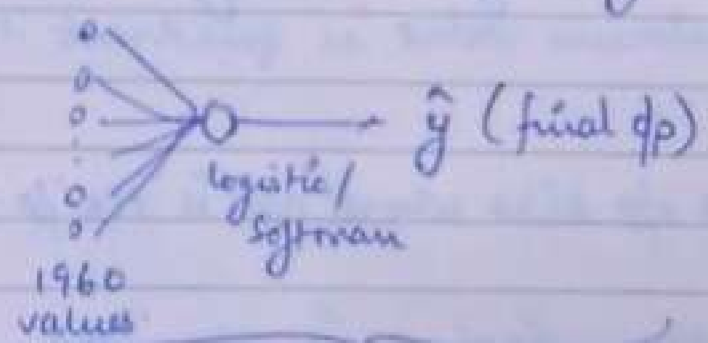
bias : $n_c^{[l]} \to (1,1,1,n_c^{[l]})$

→ Simple CNN example : (ConvNet)

3 convolution layers used :



Hence, (39×39×3) i/p image converted to (7×7×40) features for this image.



Simple regression applied to 1960 values giving features of the i/p image.

Note: As we go deeper in CNN, n_H and n_W generally ↓ while n_c ↑es. Main work in CNN is to decide the hyperparameters i.e $f^{[l]}$, $s^{[l]}$, $p^{[l]}$, no. of filters.

Note: Although its possible to design a good CNN using just convolution layers but most NN architectures also have a few pooling layers and a few fully connected layers.

So, types of layers in CNN — Convolution, Pooling, Fully connected.

① Pooling layer: to reduce the size of the representation, to speed up computation

Pooling layer: Max Pooling

region 1 → region 2

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

(4×4)

$\xrightarrow{f=2,\ s=2}$ for max pooling layer

max value from region 1

9	2
6	3

→ max value from region 2

(2×2)

Interesting thing about max pool is that it has no hyperparameters to learn i.e nothing for gradient decent to learn. we fix f and s.

Intuition behind using maxpool: features detected anywhere in one of these quadrants remain preserved in the o/p of max poling. If feature is not detected, the maximum value in quadrant itself is small.

for max pooling also, $n' = \frac{s+n+2p-f}{s}$ is the o/p size where $n \times n$ is the i/p size.

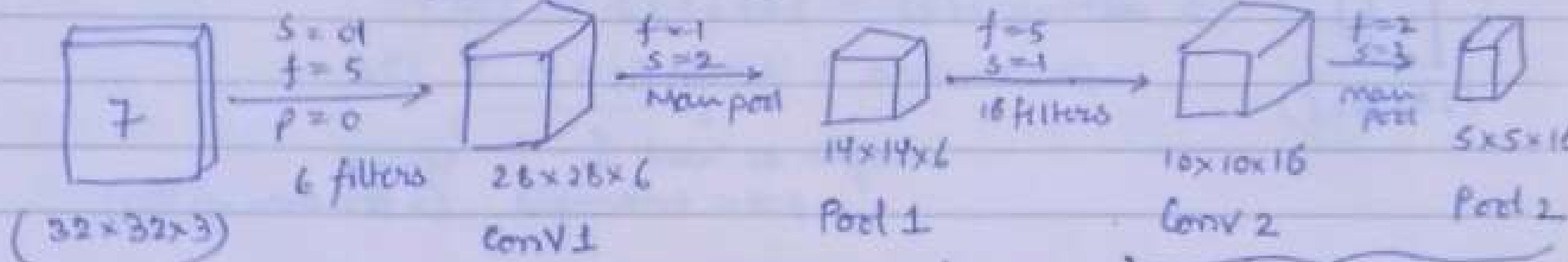
Max pooling in 3D: If $n \times n \times n_c$ is i/p then $n' \times n' \times n_c$ is o/p. The filter is applied independently to all the channels, unlike in case of 3D convolu' where o/p was 2D even for 3D output and 3D filter. Here, in pooling we have 3D output for 3D input & 2D filter applied to each channel independently.

Note Avg. pooling is also used but not much where avg. of quadrant is taken instead of max value.

f=2, s=2 are generally used to shrink the height & width to approximately $\frac{2}{3}$ in the o/p. p=0 generally in max pooling.
i.e (of half size)

Hyperparameters: f, s, max/avg pooling
(binary 0 or 1)

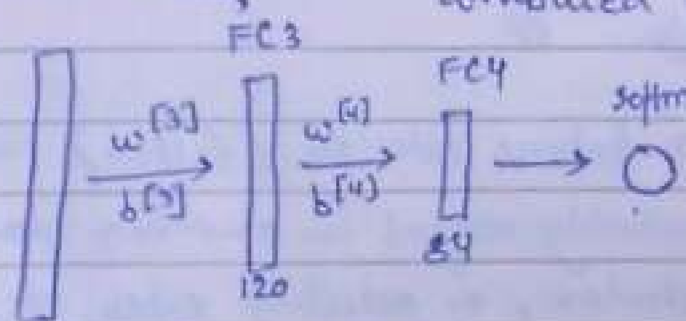
→ Complex CNN example: (inspired by LeNet-5)
we need to identify which no. from 0 to 9 is +nt in the image.



called 1 layer of the NN, though parameters hyper are taken twice, but weights & biases are considered only once during the convolu. operation hence it is combined called 1 layer.

Layer 2 of the NN

fully connected (layer 3)



softmax (ε 10 outputs in this case i.e 0 to 9)

400 i.e (5×5×16 flattened as 400 pixels)

$w^{[3]}$ = (120, 400) dimensional
$b^{[3]}$ = (120) " "
$w^{[4]}$ = (84, 120)
$b^{[4]}$ = (84)

{ just like the original NN that we saw before CNN }

Fully connected as every i/p neuron is connected to every o/p neuron.

Common CNN pattern. (no. of layers may vary).

Conv layers → Pooling layer → Conv layers → Pooling layer → Fully connected layer → Fully connected layer → Softmax.

Note: Activa^n size drops as we go deep in NN. No. of parameters is 0 for Pooling layer, too many for FC layer and relatively small for CNN as we discussed earlier. The drop in activa^n size should not be too quick.

(28x28x6)

⇒ activa^n size = 28x28x6 = 4704

parameters = $\underbrace{(5 \times 5 \times 6)}_{\text{weights as used in prev example}} + \underbrace{6}_{\text{biases (1 for each filter)}} = 156$

→ Why convolu^n over fully connected?

i) Less no. of parameters needed, hence less chance of overfitting.

In above e.g. conv needed 156 parameters. But if FC was used instead of conv here then $\underbrace{4704}_{o/p} \times \underbrace{(32 \times 32 \times 3)}_{i/p}$ parameters i.e 14 million parameters would be needed.

Conv uses parameter sharing bcoz. a feature detector (filter) that's useful in one part of the image is probably useful in another part of the image, like the vertical edge detector, or detecting eyes, etc as well. Also, in conv, in each layer each o/p value depends only on small no. of i/p i.e sparse connections unlike in FC, i.e.

6x6 * 3x3 = 4x4

This o/p depends only on the shaded region of the i/p. No other pixels affect this o/p.

ii) It captures the desirable property of translation invariance. i.e an image of a cat slightly shifted is labelled same in o/p as the image $\bar{c}$ cat in middle.

$$\text{Cost } J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) \quad \{m = \text{no. of training examples}\}.$$

Now, use gradient decent to optimize parameters to reduce J.

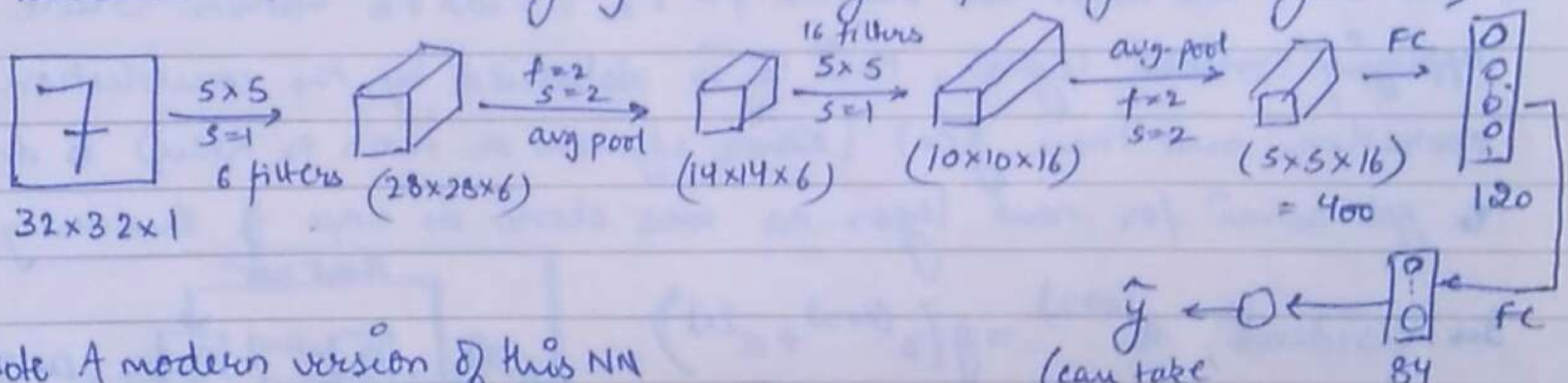
⇒ Some classic NN:

- LeNet-5
- AlexNet
- VGG Net

} → lay the founda° for modern computer vision.

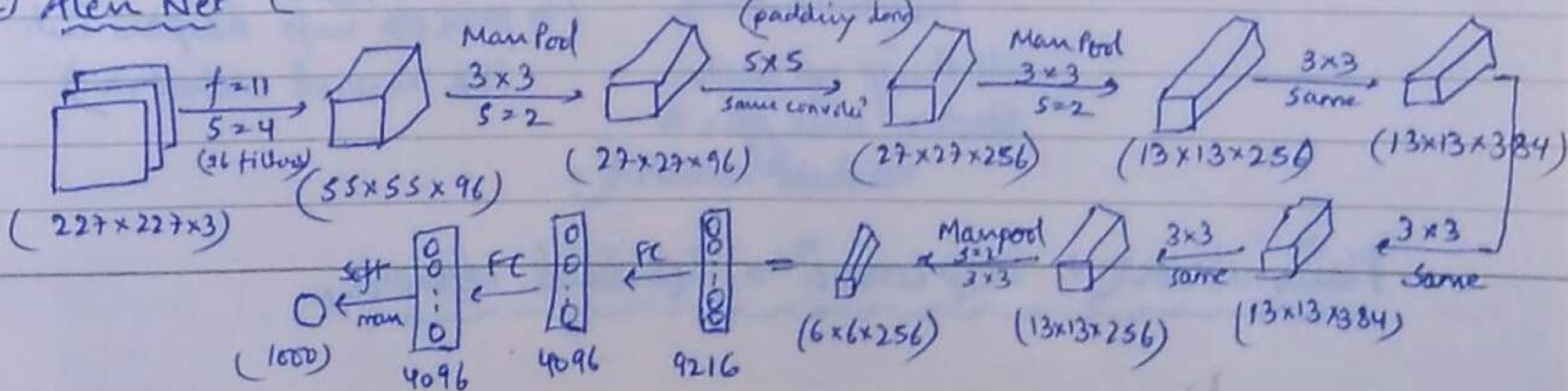
- Resnet (152 layers)
- Inception

① LeNet-5 (trained on grey scale images for digit recognition).



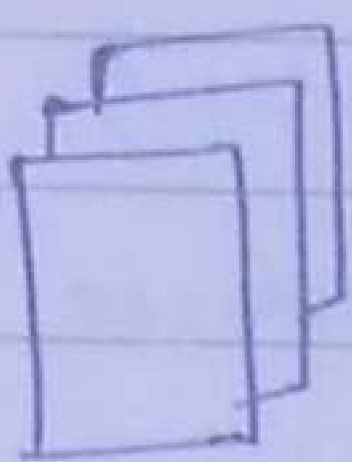
Note A modern version of this NN will use a softmax layer $\bar{c}$ a 10 way classifica° output.

② AlexNet (much more parameters than LeNet-5)



VGG-16 net (simplified architecture) has around 138 million parameters and thus a pretty large network

(16 layers having weights) { CONV = 3×3 filter, s=1, same convolution (i.e padding done) } Always in this VGG-16.
{ maxpool = 2×2, s=2 }



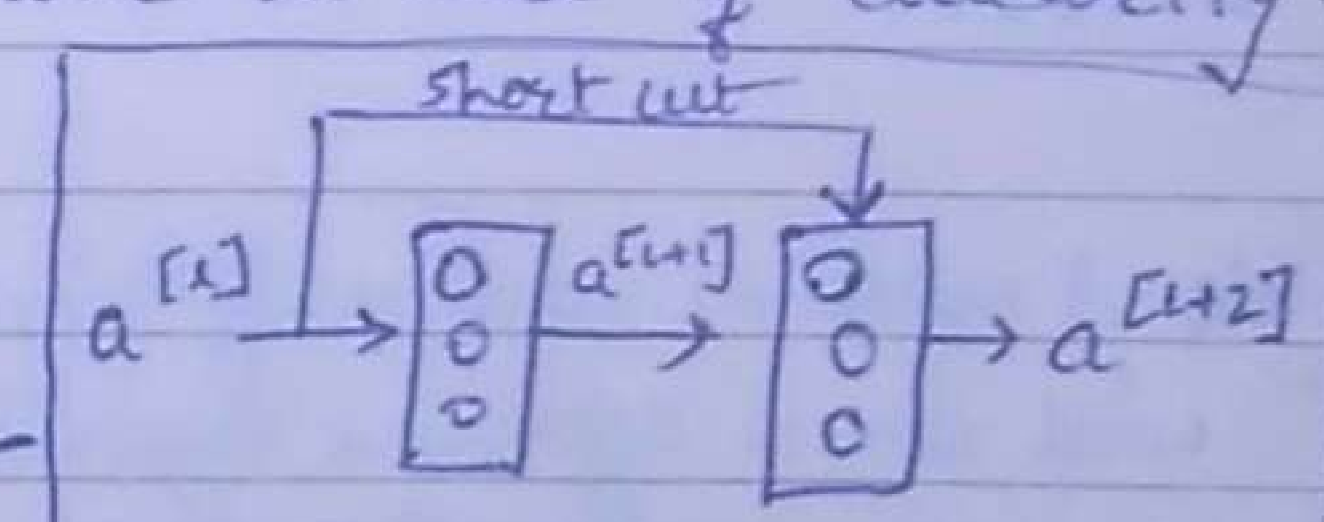
(224×224×3) $\xrightarrow[(\times 2)]{\text{CONV 64 filters}}$ (224×224×64) $\xrightarrow[\text{POOL}]{}$ (112×112×64) $\xrightarrow[(\times 2)]{\text{CONV 128 filters}}$ (112×112×128) $\xrightarrow{\text{POOL}}$ (56×56×128) ---- too many CONV and POOL ---- $\xrightarrow{\text{POOL}}$ (7×7×512) $\xrightarrow{\text{FC}}$ 4096 $\xrightarrow{\text{FC}}$ 4096 $\xrightarrow[\text{max}]{\text{Soft}}$ 1000

ii) ResNet (enables us to train very deep NN, sometimes upto 100 layers)

Note ResNets are built out of Residual blocks.

Note In CNN, we don't use linear fN i.e $z = wx + b$ rather while applying convolution layer, this z is obtained by the convolution operation and then $g(z)$ {either sigmoid or tanh or ReLU} is done to get activation for next layer as was done in case of linearity.

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



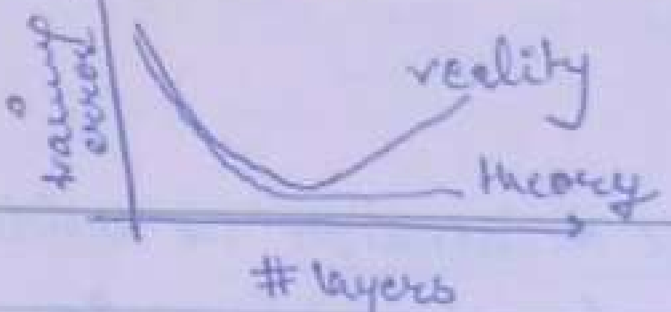
short cut ≡ skip connection (to process info deeper into the Neural Network)

Residual block

stacking up such blocks will give a residual network.

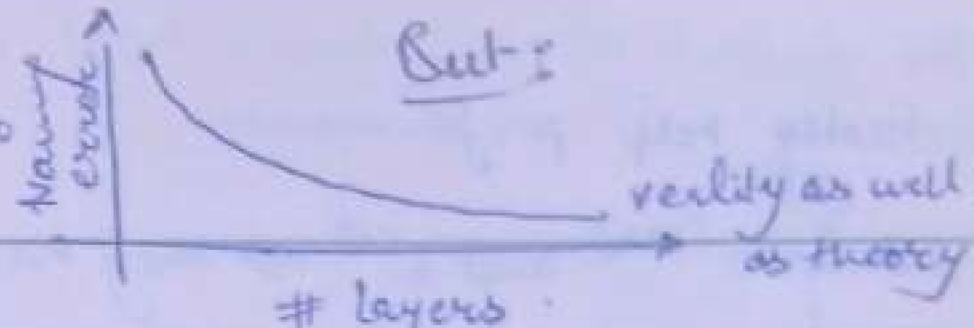
Plain network + skip connection = Residual network

Note



(for Plain Network)

But :



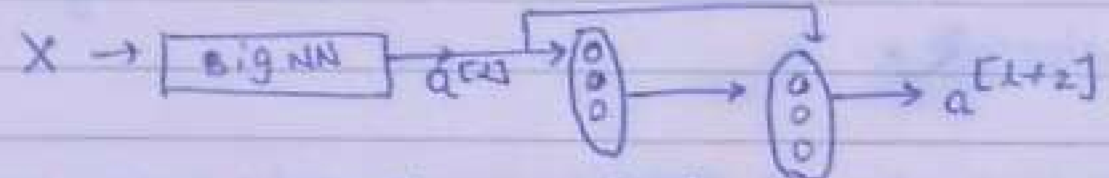
(Residual Network)

* {helps with the vanishing and exploding gradient problems}.

→ Why resnets do well?

Going much deeper in the NN might hurt the network's ability to train the network to do well on the training set but this is much less true while training a ResNet.

$X \longrightarrow \boxed{\text{Big NN}} \longrightarrow a^{[l]}$



{supposing we use ReLU so, all activation will be ≥ 0}

$$\text{so, } a^{[l+2]} = g(z^{[l+2]} + a^{[l]}) \quad \text{— (1)}$$

$$= g(\underbrace{w^{[l+2]} a^{[l+1]} + b^{[l+2]}} + a^{[l]})$$

L_2 regularization tends to shrink w and b with tiny layers and so at some point they might become 0 getting :

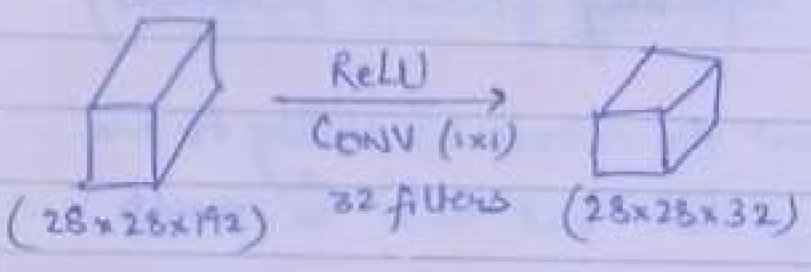
$$a^{[l+2]} = g(a^{[l]}) = a^{[l]} \quad \{\text{as ReLU of +ve is no. itself}\}.$$

Note Hence, it is easy for residual block to learn identity f^N, giving $a^{[l+2]} = a^{[l]}$. Thus adding the residual block in middle or in the end doesn't hurt the NN performance. These skip connection bcoz of being not in plain network, the result get worse with deepening of layer

The residual network doesn't hurt performance, and can sometimes actually help performance.

Adding $z^{[l+2]}$ and $a^{[l]}$ ⇒ they have <u>same dimension</u> i.e Residual network uses <u>same convolution</u> resulting in same dimension matrices.

⇒ <u>1×1 Convolutions</u> : (also called Network in Network)

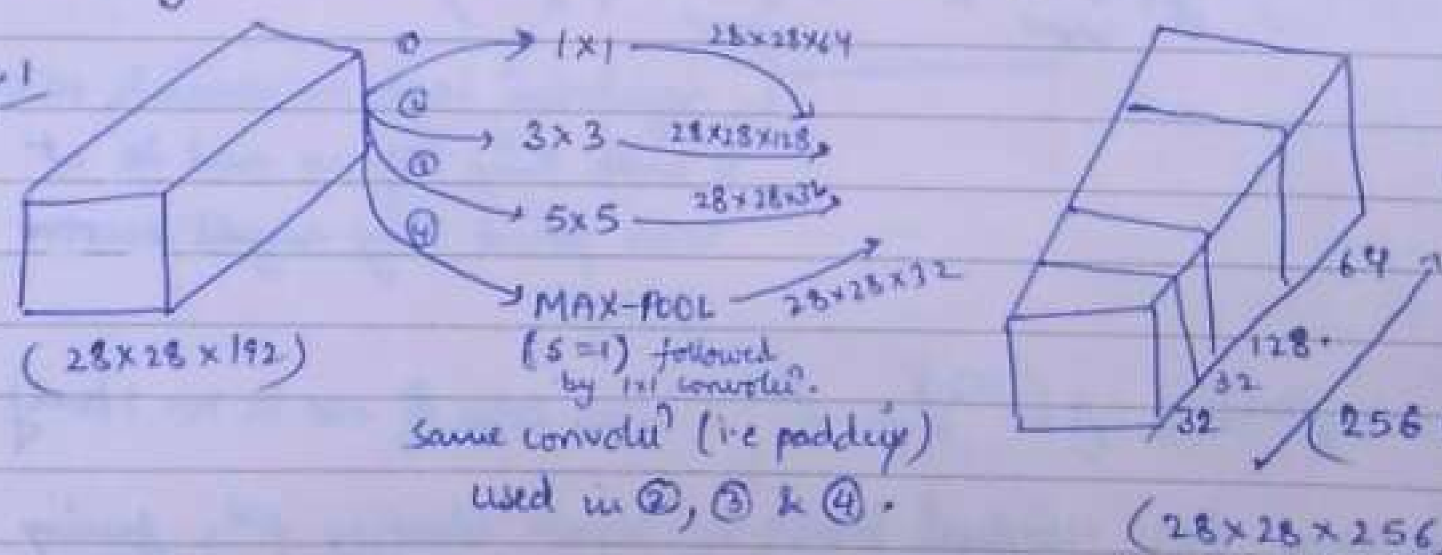


{helps reduce no. of channels, from 192 to 32 like in this case unlike maxpool which helps reduce only n_H and n_W.}

1×1 convolution helps by adding non-linearity i.e taking ReLU of the convoluted result, it helps ↑, ↓ and even keep same the n_C. This is useful in building the inception network.

⑤ <u>Inception network</u> : It says why to chose if we want to use a conv layer or a pool layer and with what parameters. It says lets try them all and let the network decide.

e.g. 1



[all results concatenated]

Same convolution (i.e padding) used in ②, ③ & ④.

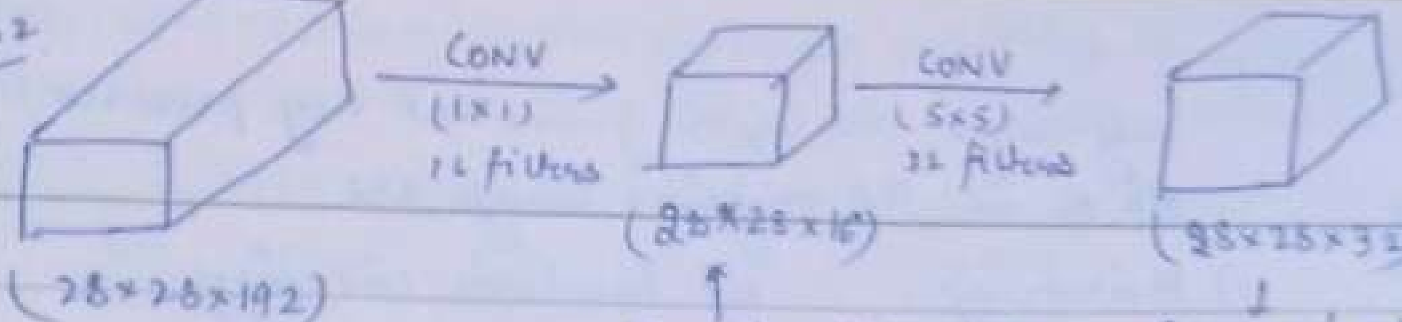
(256 channels)

(28×28×256)

Inception network problem is computation cost.

Note <u>This computation cost can be reduced by using 1×1 convolution layer.</u>

e.g. 2



Shrunken intermediate form with less no. using 1×1 convolution.

Same o/p obtained as in previous case but with less no. of computation

Computation cost for ③ in e.g. 1 = $(5\times5\times192)\times(28\times28\times32)$ = 120 million.

" " for e.g. 2 = $[(1\times1\times192)\times(28\times28\times16)] + [(5\times5\times16)\times(28\times28\times32)]$

= 12.4 million = $\frac{1}{10}$th of above cost.

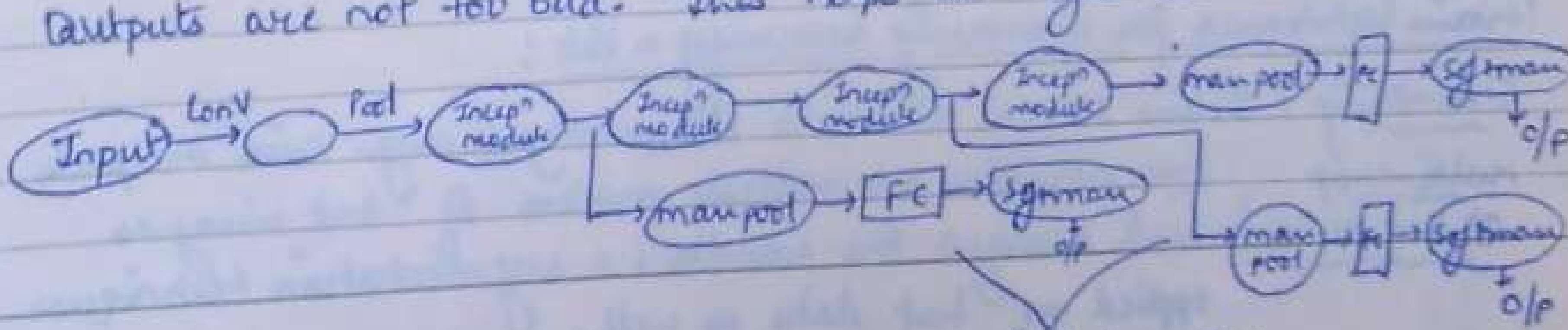
↳ 'bottleneck layer' (intermediate layer obtained after 1×1 convolution)

↓

This shrinking down doesn't hurt the performance much but reduces computation cost significantly.

e.g. 1 is an <u>inception module</u>. Too many inception modules used together in a network give rise to <u>inception network</u>.

Note Side branches are added in between the inception modules to show and confirm that if output is calculated from the intermediate modules using hidden layers +nt upto it only, outputs are not too bad. This helps in regularization as well.



Side branches with o/p similar to the final o/p

⇒ Data Augmentation: to ↑ the training dataset present ē us. Various augmenta^n techniques given that they preserve the o/p as in the original image are:

make the learning algo more robust to changes in the original images.

i) Mirroring
ii) Random cropping
iii) Rota^n
iv) Shearing
v) Local warping
vi) Color shifting

* Loading data and getting trained can happen simultaneously.

little data → more hand engineering required ("hacks")
Object detec^n
image recogni^n
Speech recogni^n
lots of data → simpler algorithms & less hand engineering

Two sources of knowledge or factors affecting model's performance:
- labelled data
- hand engineering (required more when less data is available.)

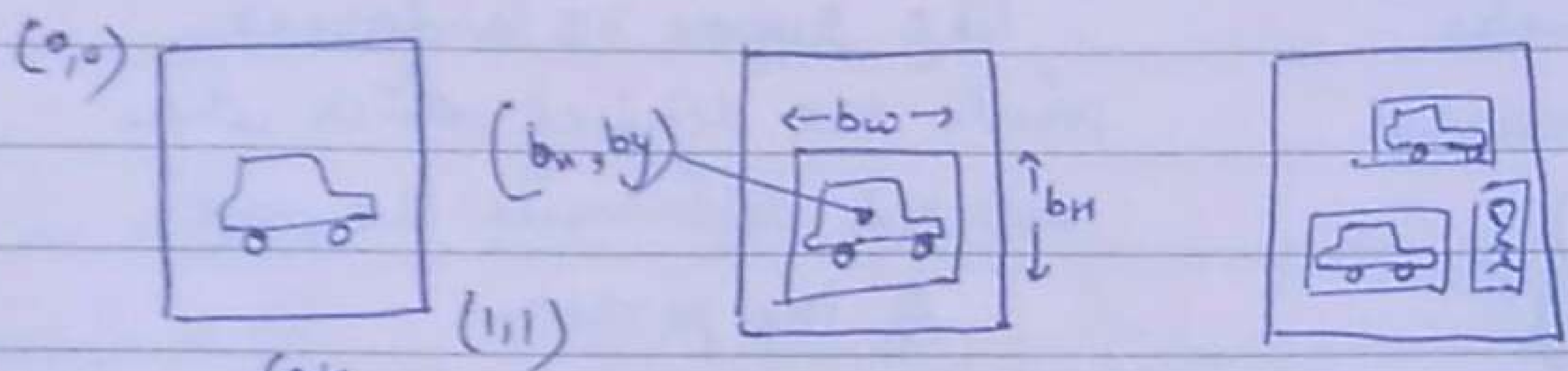
Note: Transfer learning helps in case of less availability of data.
↳ using other people's trained models and training just the last few layers of their network with our data along with changing the o/p of the softmax layer.

Certain techniques for improving accuracy a bit:

① Ensembling: Train several networks independly & avg. their outputs.
② Multi-crop at test time: Run classifier on multiple versions of test images and average the results i.e augmentation techniques applied on test data as well.

these techniques improve accuracy a little bit but use a lot of memory especially ensembling as it uses several networks. Hence, these techniques are used for winning competitions but not for product° (deployment) purposes.

⇒ Localization and Detection:



Car

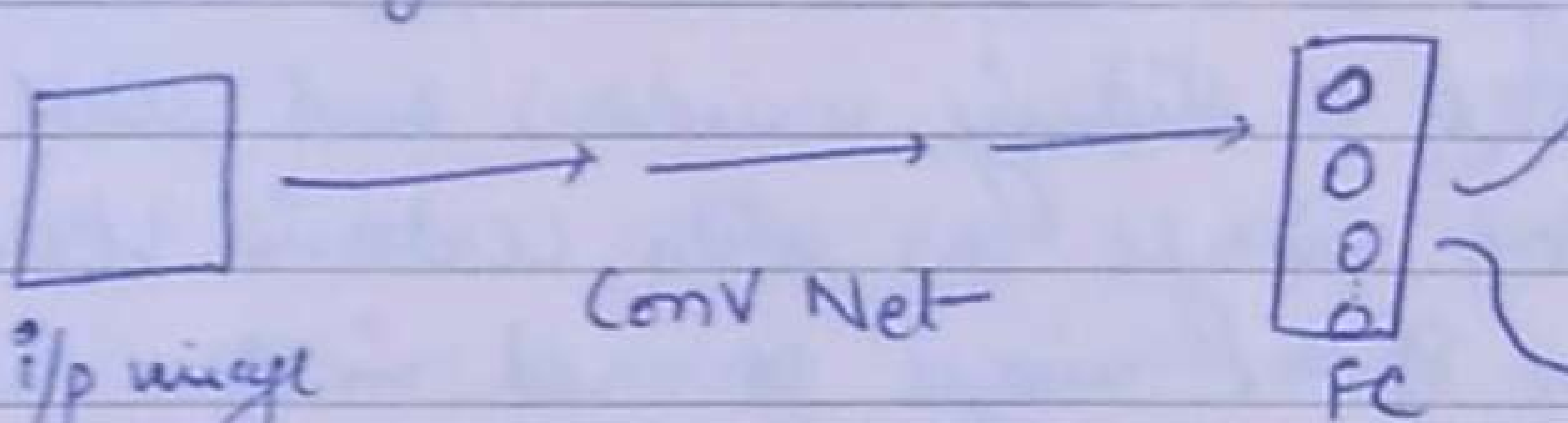
Image classifica°

Classifica° c̄ localiza°.

Detection

usually have 1 big object in the middle that needs to be recognized & localized.

there can be multiple objects



softmax (for classifying the object detected)

→ b_x, b_y, b_H, b_w (parameters for bounding box of the detected object)

e.g. for an image c̄ one object to be detected:

$$y \text{ or output label} = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_w \\ b_H \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

P_c → Probability that there is an object. (0 or 1)

b_x, b_y, b_w, b_H } bounding box parameters if $P_c = 1$

c_1, c_2, c_3 } probability of belonging to a class of object if $P_c = 1$, provided object can be either a car, bike or cycle.

if $P_c = 0$, then rest i.e $b_x, b_y, b_w, b_H, c_1, c_2, c_3$ are dont cares (?).

Landmark detecⁿ {helpful in detecting face emotions}.

(i.e detecⁿ of important points in an image, now the o/p layer will have the coordinates of these important landmarks in the image as well) along ē the o/p unit telling if its a face or not.

Or in snapchat, etc, or in pose detection as well.

↓

like suppose 32 landmark points are defined which when detected determine the pose of the person.

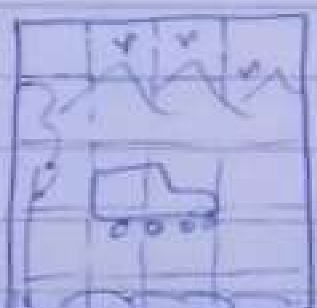


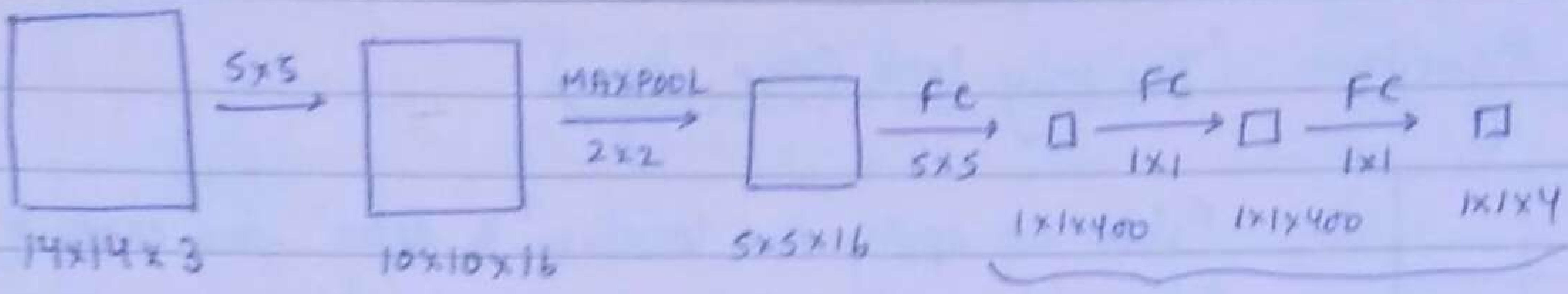
Image in which object is to be detected. (a)

Object detecⁿ using sliding window detection.

A conv net is trained suppose for detecting if an i/p image is a car or not. So, what we do is we take a sliding window and slide it all across the image (a) with certain stride. The portion of the image covered in the window is passed into the conv net that was trained to detect a car, and an o/p is obtained if that portion of the image contained a car or not. This process is repeated with ↑ing window size in a hope that there will be some window that will bound the car tnt in (a) and output 1 i.e car tnt will be obtained in the conv net on passing that window as i/p to it.

Drawback of sliding window detecⁿ : Computation cost bcoz if fine strides are not used, we end up not being able to localize the objects that accurately within the image.

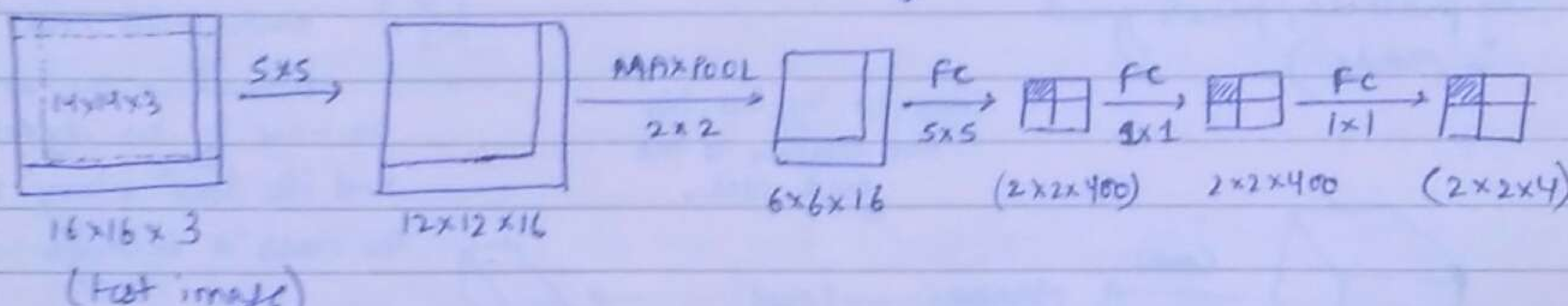
→ To reduce the computaⁿ cost of sliding window detecⁿ, <u>convolutional implementation</u> of <u>sliding windows</u> is done.



↳ convolutional implement of fully connected layer

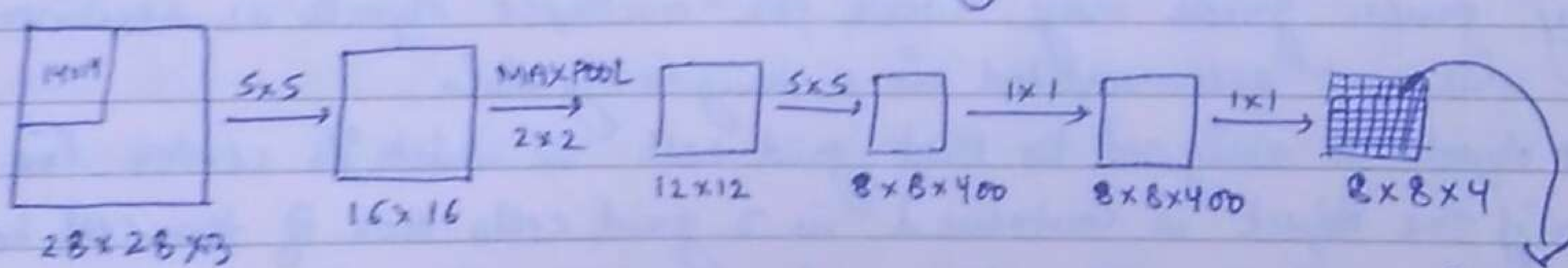
In sliding window detecⁿ, lot of computaⁿ is repetitive, so convolutional implementaⁿ allows to share the common computaⁿ thus reducing the computaⁿ cost. e.g.

e.g.



So, in the above process, the last layer gives the classificaⁿ result for 4 <u>filters</u> (windows) simultaneously which have a stride of 2. This save 4 times computaⁿ cost as the common computaⁿ of the 4 windows is done simultaneously.

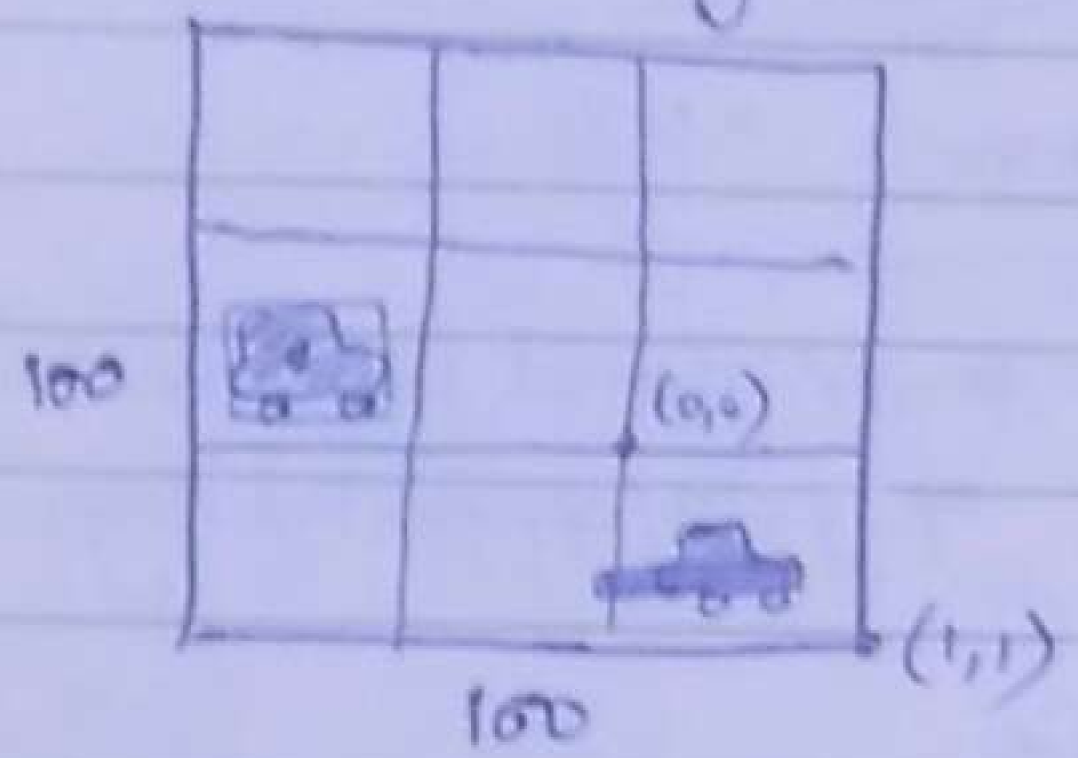
e.g.



computaⁿ for (s = 2), 64 windows is done simultaneously unlike before convoluⁿ implementaⁿ where 64 times sequentially the computaⁿ was done.

Drawback of this implementation is that we don't get exact bounding boxes, approx. boundaries are obtained bcoz of using a fixed stride in the windows.

Soln: YOLO Algorithm (You only look Once Algo)



(i/p image)

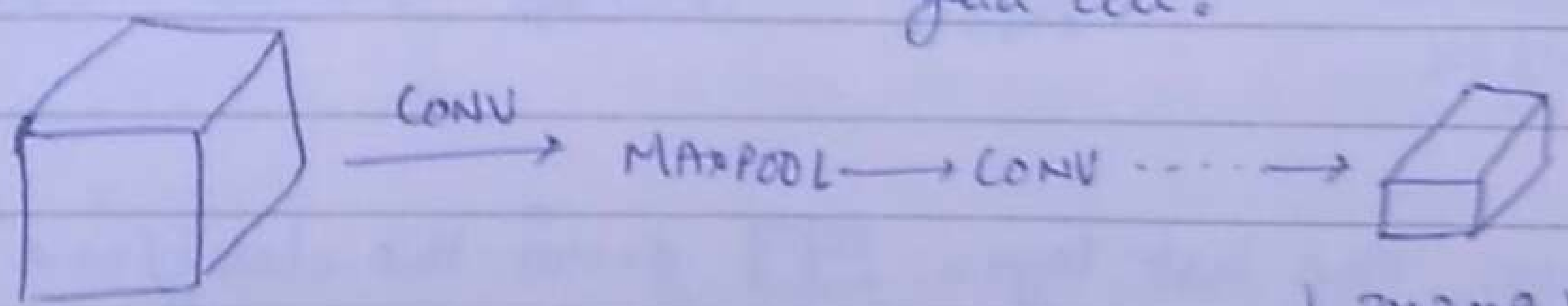
(In practice, finer grid is made.)

First we apply the localizn & detecn for each of the grid cell giving an output y i.e $\underbrace{3\times3}_{\text{total grid cells}}\times 8$.

(y)

(as discussed earlier) $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ → dimension for each grid cell indicating the +ve or -ve of an object and its locn along ē the class it belongs to.

Note b_x, b_y, b_w, b_h are relative to the grid cell.



100 x 100 x 3

(i/p image)

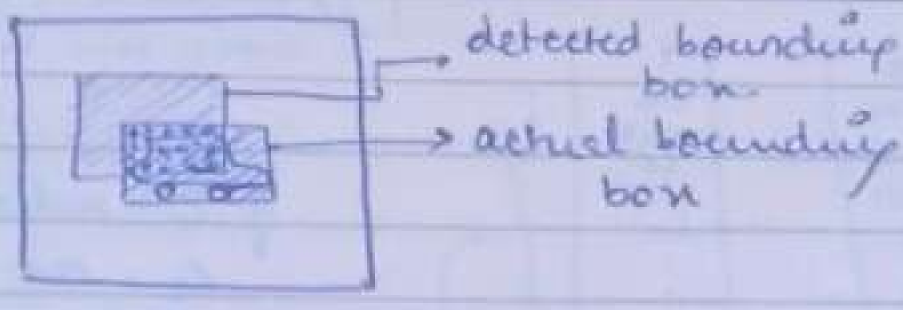
(3x3x8)

output (y)

This algo works fine as long as each grid has just one object in it. Making finer grids may work for multiple objects as now no 2 objects will get accomodated in a single grid.

Note The object is assigned to that grid cell in which its center lies. So, even if the object is contained in 2 grid cells bcoz of the cell being small, it would be assigned only to 1 grid cell that contains its center. Hence, YOLO helps in getting more accurate bounding boxes as now we do not get the grid cell boxes as boundaries but boundaries formed inside these grid cells. It runs very fast bcoz of conv implementation.

→ <u>Intersecⁿ over union</u> {to evaluate the object detecⁿ algorithm}
(IoU function)



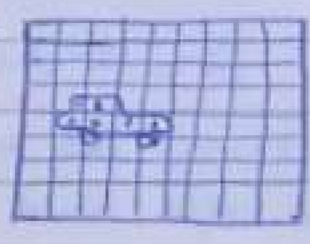
I/p image.

$$IoU = \frac{\text{Size of } \boxed{} \text{ (intersec area)}}{\text{Size of } \boxed{} \text{ (union area)}}$$

If IoU ≥ 0.5 (o/p is correct)

More the IoU, better is the bounding box predicⁿ, i.e the object is correctly localized if IoU = 1 and approx. correct if IoU ≥ 0.5.

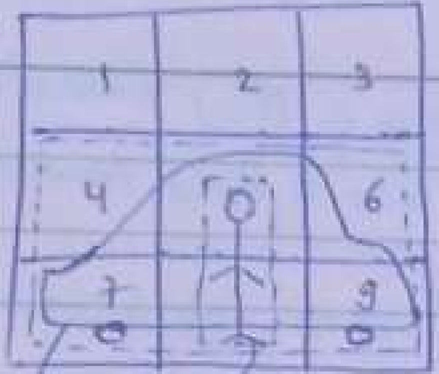
→ <u>Non-max suppression</u> {to make sure that our algo detects each object only once}



Though the car has just 1 midpoint, but we run the algo for each grid cell and bcoz every cell will consider its object centre as the centre of the object, car will be detected multiple times.

Non max suppression removes the extra detections of an object using the probability pc. The output ŷ with highest pc i.e the max probability that it contains an object is kept and the rest of the boxes with low pc or higher IoU with the box having higher pc are removed. Hence the name, i.e suppressing the ones with non-max pc.
To apply non-max suppression on an image with multiple class objects, the non-max suppression is run as many times as the no. of classes, i.e individually for each class of object.

→ <u>Ancher boxes</u> {to enable a grid cell to detect multiple objects}



car pedestrian

(mid point of both of them lie in the same grid cell)

lets define 2 ancher boxes $\boxed{1}$, $\boxed{2}$. even more can be defined

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_H \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_H \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_H \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_H \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

(for grid cells)

corresponds to ancher box 1.

corresponds to anchor box 2

if c_1 = pedestrian
c_2 = car

we see that bounding box of pedestrian resembles ancher box 1 so we assign the pedestrian, $\boxed{1}$ and the bounding box of car is similarly assigned $\boxed{2}$.

So, the output comes out as (3x3x16) for ancher box algorithm, as we have 3x3 grid cells and 2 ancher boxes and 8 outputs corresponding to each ancher box where 8 = 5+(# classes)

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

if the grid cell has only a car and no pedestrian and the car has bounding box similar to ancher 2.

Drawbacks: i) three objects in a grid cell but only 2 ancher boxes available.
ii) both objects resemble the same ancher box.

The above 2 condi^n can't be handled by the above method of ancher boxes.

Note Anchor box shapes are defined in such a way that a particular object if present in any shape can be detected by the anchor box. This helps algorithm specialize in classifying different object on basis of their shapes and sizes.

RCNN: Regions c̄ CNN that is the regions that make sense to run CNN on. Only those regions in an image which have some interesting object to be detected are selected to run CNN on or window on.

Segmentaⁿ algorithm helps finding such regions (blobs) and run CNN on just those blobs, thus reducing the computaⁿ cost significantly.

⟹ Face Recognition:

Face verificaⁿ: Input image, name/ID
Output whether the i/p image is that of the claimed person.

Face recognition: Input image and identify the name of the person if it exists in the database.

One shot learning: Learning from just one example to recognize the person again, this learning is generally required in face recogniⁿ systems where we have just 1 picture of our employee available to train the network. This is in contrast c̄ our DNN which we have studied so far and which needs lot of training data to learn the features.

So, for one shot learning, we use a "similarity function". Neural network learns the fⁿ I

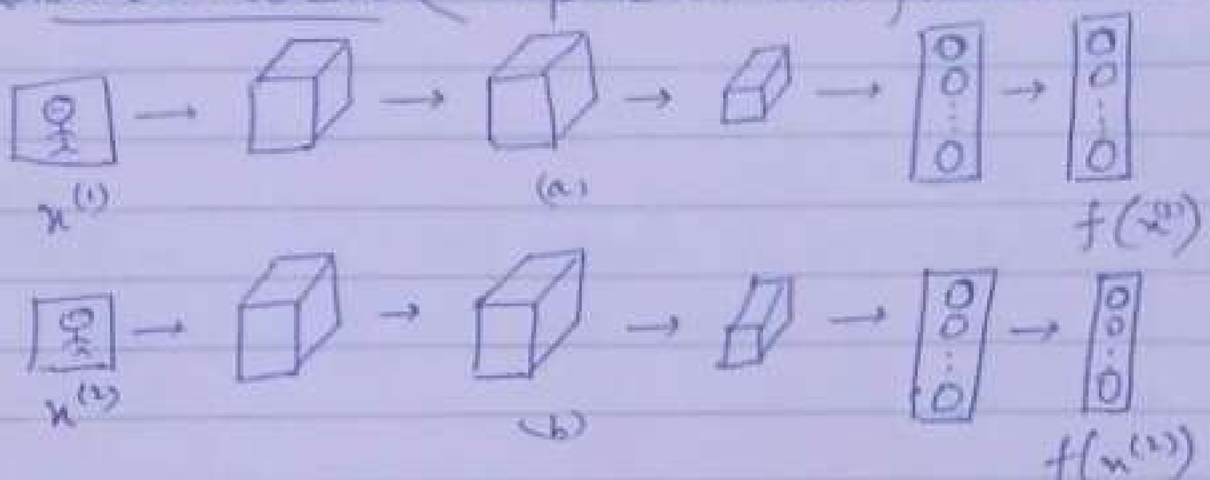
$$\boxed{d(\text{img }1, \text{img }2) = \text{degree of difference b/w images}}$$

If $d(img1, img2) \le \tau$ "same person"
$> \tau$ "different person" } Image verification.

τ is a hyperparameter.

for face recogniⁿ the same formula is applied with every image present in the database. If the i/p image matches some image in the database $d(img1, img2)$ for it is quite low.
So, if any new person joins the team, simply a new image can be added to the database and 'd' remains the same.

<u>Siamese Network</u> help learn this function 'd'.



{vectored representaⁿ of the i/p image} say of 128 dimension in this case.

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

* Network (a) and (b) are the same with same parameters learnt such that if $x^{(i)}$ and $x^{(j)}$ are the same person, then $d(x^{(i)}, x^{(j)})$ is small, and " " " " different " " " " " large.

→ <u>Triplet loss function</u>: This aims at tuning the network parameters such that encodings of the images of same person is close enough and encoding of images of different persons is farther enough. As, in this we deal with 3 images, it is called so.

$\boxed{A}$ $\boxed{P}$ $\boxed{A}$ $\boxed{N}$

Image of person A — Another image of person A — Image of person A — Another image but not of person A.

Now, triple fn wants : $d(A,P) \leq d(A,N)$

$$\|f(A)-f(P)\|^2 \leq \|f(A)-f(N)\|^2$$

(Modifica^n) $\Rightarrow \|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha \leq 0$

↳ margin to avoid NN from giving trivial sol^n i.e encoding all images as zero vector or as same vectors.

Note this α is a hyperparameter which actually pushes AP pair and AN pair further away from each other.

$\therefore$ $L(A,P,N) = \max\left(d(A,P) - d(A,N) + \alpha, 0\right)$
(loss on a single triplet)

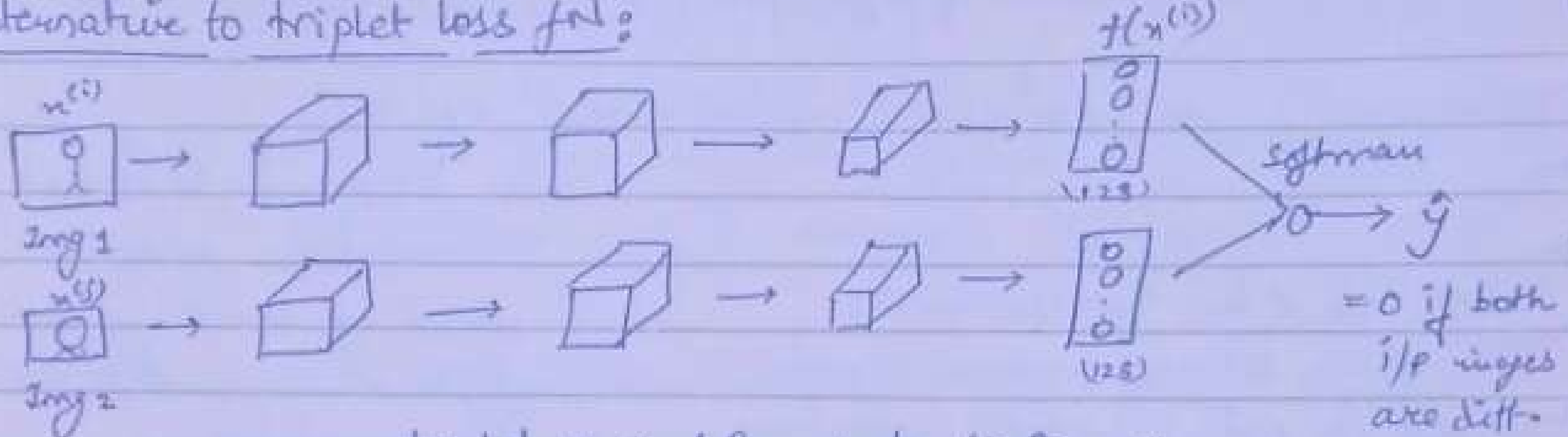
$J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$ {where m is the no. of triplets for training purpose.}
overall cost fn for NN

If we have 10 k images of 1k people, then triplets APN need to be formed for train dataset.

Note↓
Hence, a good train data is needed to train the NN for one shot learning. Once the hyperparameters are properly tuned to encode an image, one shot learning is possible.

Triplets that are 'hard' to train on are chosen. Bcoz if triplets are chosen randomly, there is higher probability of $L(A,P,N)$ to be satisfied. Such triplets are chosen where $d(A,P) \approx d(A,N)$ so that the network tries hard to learn the hyperparameters so as to meet the constraints i.e $d(A,P) + \alpha \le d(A,N)$. In case of hard triplets only the gradient descent would learn something bcoz otherwise in case of random triplets, the network would get it right everytime and hence not learn anything.

<u>Alternative to triplet loss fn:</u>



<u>treated as a binary classificⁿ problem</u>

The 128 features are fed into the logistic regression unit i.e softmax layer

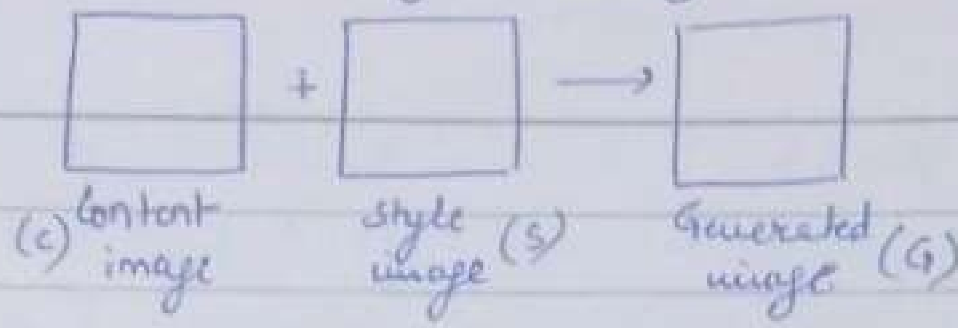
and $\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \underbrace{\left| f(x^{(i)})_k - f(x^{(j)})_k \right|}_{\text{there can be other variations to this to compute the difference b/w the two f's.}} + b\right)$

parameters as used in any logistic regression.

Note ↓

So, in this approach the training set is not a triple but a pair of images where the target label is 1 if the pair has images of same person and 0 if the pair has images of diff person.

⇒ Neural Style Transfer:



Going deeper in NN, more and more complex features are detected by the hidden units. Starting with lines and edges, going to detecting shapes and so on.

- for Generating a Neural style transfer image, a cost f^n is defined:

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

$J(G)$ ↓ cost f^n in generating G.

$J_{content}(C,G)$ ↓ measures how similar is the content of generated image to the content of the content image C.

$J_{style}(S,G)$ ↓ measures how similar is the style of the image G to the style of the image S.

steps involved:

i) Initialize G randomly.

ii) Use Gradient decent to minimize $J(G)$.

$$G = G - \frac{\partial}{\partial G} J(G)$$

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

$a^{[l](C)}$ ↓ activa^n of l^{th} hidden layer for image C.

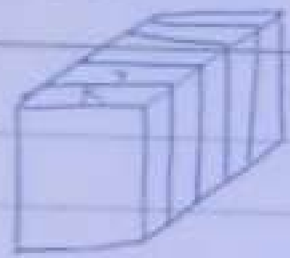
$a^{[l](G)}$ ↓ activa^n of l^{th} hidden layer for image G.

{ l is a middle layer not too shallow, not too deep in the ConvNet. (which is pretrained to generate features of the I/p image) }

middle layer is taken so that neither too simple nor too complex features are extracted from the image.

If layer l's activa^n is used to measure "style", then style is defined as correlation between activations across channels.

e.g.



activa^n block of layer l.

If correla^n b/w R and Y channel is calculated, it means calculating the probability that whatever part of the image has features detected by Y also has features which are detected by channel R. So, it is basically finding out what features tend to occur together and what features are rarely seen together in different parts of an image.

So, let $a^{[l]}_{i,j,k}$ = activation at $(\overset{H}{i},\overset{W}{j},\overset{C}{k})$. $\underset{\text{style matrix of hidden layer } l \text{ having } n_c \text{ channels.}}{G^{[l]}}$ is $\underbrace{n_c^{[l]} \times n_c^{[l]}}_{\text{to measure the correla}^n \text{ b/w each pair of channel.}}$

$$\boxed{G^{[l]}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l]}_{ijk}\, a^{[l]}_{ijk'}}$$ where k and k' ∈ {1, 2, ... n_c}

calculate for every k and k' to find $G^{[l]}$ or the style matrix.

Note If the two channels k and k' are correlated $G_{kk'}$ will be large
otherwise if " " " " " uncorrelated " " " small.

$$\boxed{J^{[l]}_{style}(S,G) = \| \underset{\text{style matrix of image S.}}{G^{[l](S)}} - \underset{\text{style matrix of generated image G.}}{G^{[l](G)}} \|^2}$$

$$J^{[l]}_{style}(S,G) = \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'}\right)^2$$

$$J_{style}(S,G) = \sum_l \underset{\text{(hyperparameter)}}{\lambda^{[l]}} J_{style}^{[l]}(S,G)$$ {taking into account both low level & high level feature while calculating correlations}

Note: Convolun operan can be performed on 1D, 2D as well as 3D inputs.

e.g. of 1D.

7 * 3 = 5 × 16 (using stride of 1)

(16 filters used)

e.g. of 2D.

(10×10) * (5×10) = (6×16)

↓ 16 filters used

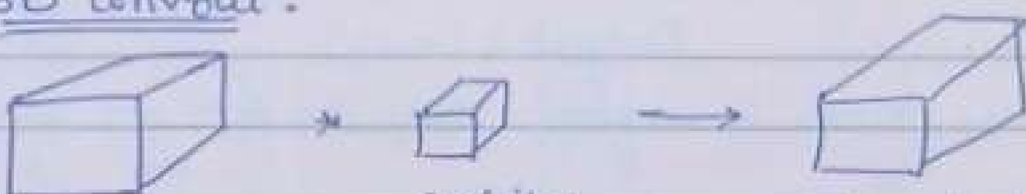
e.g. of 3D

(10×10×16) * (5×5×16) = (6×32)

↓ 32 such filters used.

For 1D data, recurrent NN are more used, though CNN can also be applied as seen above.

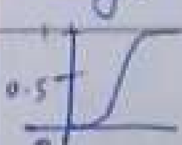
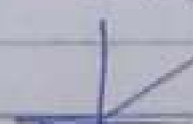
→ 3D Convolun:



3D volume (e.g. CT scan) say (14×14×14) (e.g. movie data) * 3D filter (5×5×5) (16 filters) → 3D output (10×10×10)×16

{Data can also have different no. of channels as we saw in case of 2D. In this e.g. $n_c=1$}

Note: Activation functions:

Sigmoid	tanh	Relu
$\frac{1}{1+e^{-x}}$	$2\,\text{sigmoid}(2x)-1$	$\max(0, x)$
range: (-1, 1)	range: (-1, 1)	range: (0, ∞)
		
Disadv: vanishing gradient decent	less vanishing GD	non-linear fN, hence no problem like linear fN.

- No vanishing gradient problem
- leaky Relu over dying Relu
- Relu is sparse i.e not all activan are processed to describe the o/p of network, hence reducing cost, as 50% network yields 0 bcoz of property of Relu.