

Convolutional Neural Networks

open your eyes hehehehe ~~~~!!!

Computer vision

→ CV problems

- Image classification
- object detection
- Neural Style Transfer

↳ Take an image ask the Computer to make that image into the style of some other given image.

→ The input in computer vision become really big.

ex→

1000 x 1000 x 3 pixel image

then, the input feature vector would be 3 million,

so imagine if you have a neural net & its first layer has 1000 units then the first layer itself will have 3 billion parameters (3 million x 1000)

To fix this you need to implement convolution operation

Edge detection "an example for convolution operation"

So

an image has got edges for example lets say you take a photo of a dog who is standing on a pole in front of the sea you will have edges, well there will be pixels that contains dogs image & sea image & the dramatic shift from dog pixel's to sea pixels will correspond to an image.

Cx→ Vertical edge detection "convolution operation symbol"

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

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

*

= []_{4x4}

3x3

"filter"

or
"kernel"

→ Now multiply the matrix in red with filter element wise & add all the element & that element will become the first element (a_{11}) of our _{4x4} matrix resultant 4x4 matrix.

$$\begin{aligned} \text{for example } a_{11} &= 3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 5 + 5 \times 0 \\ &\quad + 7 \times 0 + -1 \times 1 + -1 \times 8 + -1 \times 2 \\ &\Rightarrow -5 \end{aligned}$$

So now what about the next element in the resultant matrix. (a_{12}), that just shift the filter one element to the right, it will look like

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

Now let's say you have found all the element of the first row $[a_{11} \dots a_{14}]$, how are you supposed to get the elements of second row a_{21} here's how you will place the filter,

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

So after doing all the operations we get the resultant matrix :-

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

In python: we use "conv-forward" function to perform convolution operation

ex> for vertical edge detection

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

6x6

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

in an image it will look like



after applying the convolution operation you can see that it detects the edges nicely.

More Edge detection-:

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

 $\times$

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

 $=$

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

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

 $\times$

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

 $=$

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

Vertical & horizontal edge detection

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

Vertical
"kernel"
or
"filter"

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

Horizontal
"kernel"
or
"filter"

The procedure for horizontal edge detection are same as vertical, you just have to change the filter.

Some other filters are -:

1) Sobel filter

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

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

Sobel filter
these are vertical filters, just rotate them 90° to right you will get horizontal filters

Scharr filter

treating filters values as learnable parameters.

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

$$\begin{matrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{matrix} = []_{4 \times 4}$$

→ to get a new set of filter values you can just treat them as learnable parameters & perform supervised learning to get appropriate values of filters.

Padding

So when you perform convolution operation, the size of the matrix decreases, & hence to preserve the decrease original size of the matrix we apply padding,

0	0	0	0	0	0	0	0
0	///						0
0							0
0			///				0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

6 x 6

$h \times n$

$$* \begin{bmatrix} \end{bmatrix}_{3 \times 3} = \begin{bmatrix} \end{bmatrix}_{6 \times 6}$$

You layer the matrix just like above & the layers have a value of 0, all of them

Notice that the image pixel shaded in red at the corner only gets to participate in the convolution process one single time & the middle pixel shaded in red gets to participate multiple times, so the red corner pixel is throwing away a lot of the information,

So we also do padding to ~~preserve~~ preserve some information too

The formula by which the size decreases is $\Rightarrow$

$$\Rightarrow n - f + 1$$

$n \rightarrow$ dimension of the image

$f \rightarrow$ dimension of the filter

ex $\Rightarrow$ let's take
6x6 matrix &
3x3 filter
so,

$$n - f + 1 \Rightarrow 6 - 3 + 1 \Rightarrow 4$$

so the resultant image will be of 4x4

Types of convolutions

1) valid $\Rightarrow$ in this operation we don't do padding just saw image in the convolution process.

2) Same $\rightarrow$ Pad so that output size is the same as the input size.
formula will be

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

$f \rightarrow$ dimension of the filter.

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

ex $\Rightarrow$ let's say your filter is of 5x5

then, $P = \frac{5-1}{2} \Rightarrow 2$

// & the f is usually an odd number dimension.

$$P = 2$$

Strided convolution :-

In strided convolution instead of doing it the usual way, ~~insted~~ instead of shifting the convolution one step to the left we shift it two steps or more than two steps every time, & same goes when you are changing # going from one row to another. Stride is just the number by which you are ~~chr~~ shifting your convolution.

$$\begin{bmatrix} \quad \end{bmatrix}_{7 \times 7} * \begin{bmatrix} \quad \end{bmatrix}_{3 \times 3} = \begin{bmatrix} \quad \end{bmatrix}_{3 \times 3}$$

$$\begin{array}{l} n \times n * f \times f \\ \text{padding } P \quad \text{stride } S \end{array}$$

Now, if you have a matrix with dim $n \times n$ & a filter with dimension $f \times f$, padding = P , Stride = S ,

then the dimension of the resultant matrix will be,

$$\left[\frac{n + 2P - f}{S} + 1 \right] \times \left[\frac{n + 2P - f}{S} + 1 \right]$$

dimensional

Cross-correlation vs Convolution

- in true mathematical convolution, the filter is flipped horizontally & vertically before being doing element-wise multiplication & summation.
- In deep learning, we usually skip this flipping step.
- Because of that, what Neural Net actually computes is technically cross-correlation, not strict convolution.
- But by convention, the deep learning community still calls it convolution.

Convolutions over volume

let's say you have an image of depth 3 meaning it has (R, G, B) color's, so

it will be

1000 x 1000 x 3 dimensions

Now,

to convolve this image you will need a filter that also has depth 3 for example,

& the output of the convolution will be a 2d matrix $50 \times 50 \times 3$

ex $\rightarrow$

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}_{6 \times 6 \times 3} * \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}_{3 \times 3 \times 3} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}_{4 \times 4}$$

the depth can be also said stated as channels like saying 3 channels will be equal to saying depth is 3.

One layer of a Convolutional Network

⇒ A convolutional neural network has filters & there can be "n" number of filters. Each filter has different elements.

$$\begin{aligned} \left[\begin{array}{c} \text{ } \\ \text{ } \\ \text{ } \end{array} \right]_{6 \times 6 \times 3} &\rightarrow * \left[\begin{array}{c} \text{ } \\ \text{ } \\ \text{ } \end{array} \right]_{3 \times 3 \times 3} \rightarrow \text{Relu} \left[\left[\begin{array}{c} \text{ } \\ \text{ } \\ \text{ } \end{array} \right]_{4 \times 4} + b_1 \right] \rightarrow \left[\begin{array}{c} \text{ } \\ \text{ } \\ \text{ } \end{array} \right]_{4 \times 4} \\ &\quad \vdots \\ &\rightarrow * \left[\begin{array}{c} \text{ } \\ \text{ } \\ \text{ } \end{array} \right]_{3 \times 3 \times 3} \rightarrow \text{Relu} \left[\left[\begin{array}{c} \text{ } \\ \text{ } \\ \text{ } \end{array} \right]_{4 \times 4} + b_2 \right] \rightarrow \left[\begin{array}{c} \text{ } \\ \text{ } \\ \text{ } \end{array} \right]_{4 \times 4} \end{aligned}$$

The output after each getting convolved, we stack it & that becomes the input for the other layer.

⇒ No. of parameters in one layer

↳ if you have 10 filters that are $3 \times 3 \times 3$ in one layer of a NN, how many parameters does that layer have

$$\Rightarrow 10 \times 3 \times 3 \times 3 \Rightarrow 270$$

a single filter will have 27 params
10 of them will have 270 params

Notations :-

if layer 1 is a convolution layer :-

$f^{[1]}$ = filter size

$p^{[1]}$ = Padding

$s^{[1]}$ = stride

$n_c^{[1]}$ = number of filters

Input: $n_H^{[1]} \times n_W^{[1]} \times n_C^{[1]}$

Output: $n_H^{[1]} \times n_W^{[1]} \times n_C^{[1]}$

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

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

Each filter is:- $f^{[1]} \times f^{[1]} \times n_C^{[1]}$

Activations :- $a^{[1]} \rightarrow n_H^{[1]} \times n_W^{[1]} \times n_C^{[1]}$

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

bias :- $n_C^{[1]}$, $(1, 1, 1, n_C^{[1]})$

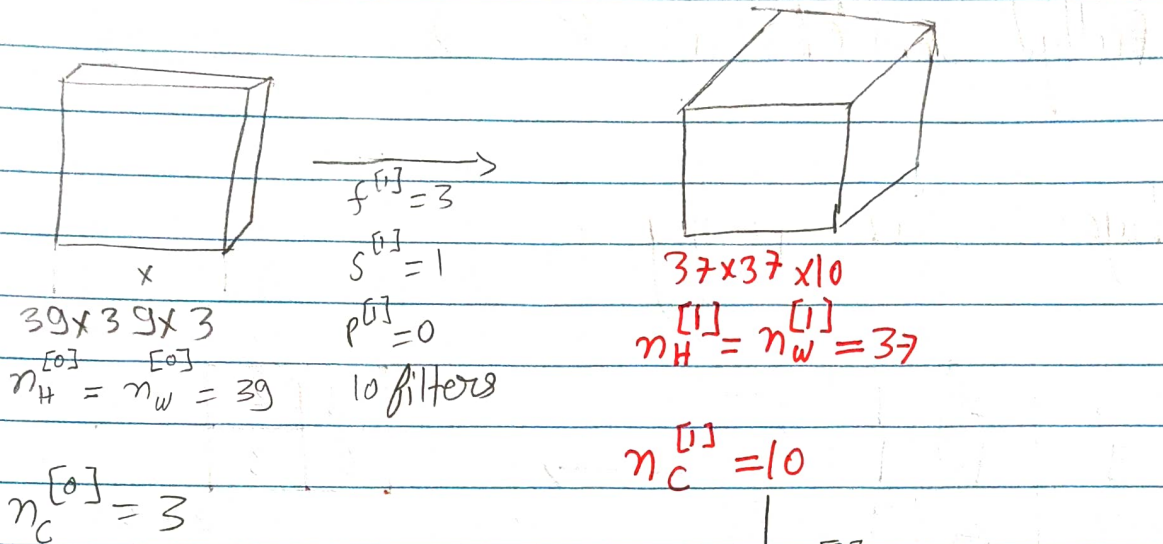
Activation when you do multiple forward prop :-

$A^{[1]} \rightarrow m \times n_H^{[1]} \times n_W^{[1]} \times n_C^{[1]}$

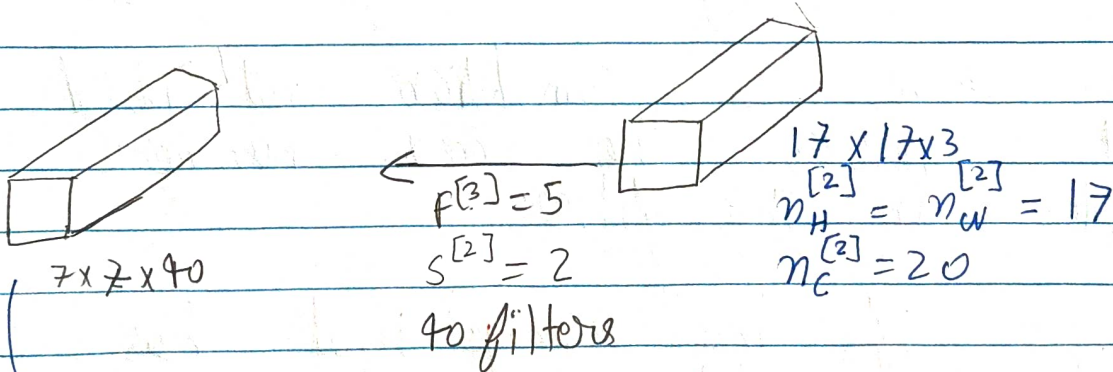
$m \rightarrow$ no. of training examples

A simple convolution network example:

Example of a ConvNet



$f^{[2]} = 5$
 $s^{[2]} = 2$
 $p^{[2]} = 0$
20 filters



Flatten this out, it will be an array of 1960 input feature vectors. Throw them into a neural net & make that nn give prediction.

Types of layers in Convolutional Neural Network:

- Convolution (conv) ←
- Pooling (pool) ←
- Fully connected (FC) ←

Pooling layer: Max pooling

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

4x4



9	2
6	3

2x2

Hyperparameters

Filter size = $f = 2$

Stride = $s = 2$

it's same as convolution but instead it just chooses the max number that resides in the filter.

Suppose if the image input has 3 channels then what?

The output image will also have ~~on~~ ~~to~~ 3 channels, just do the pooling on each channel separately & stack them, that's it. The output dimension would still be given by,

$$\left[\frac{n + 2p - f}{s} + 1 \right]$$

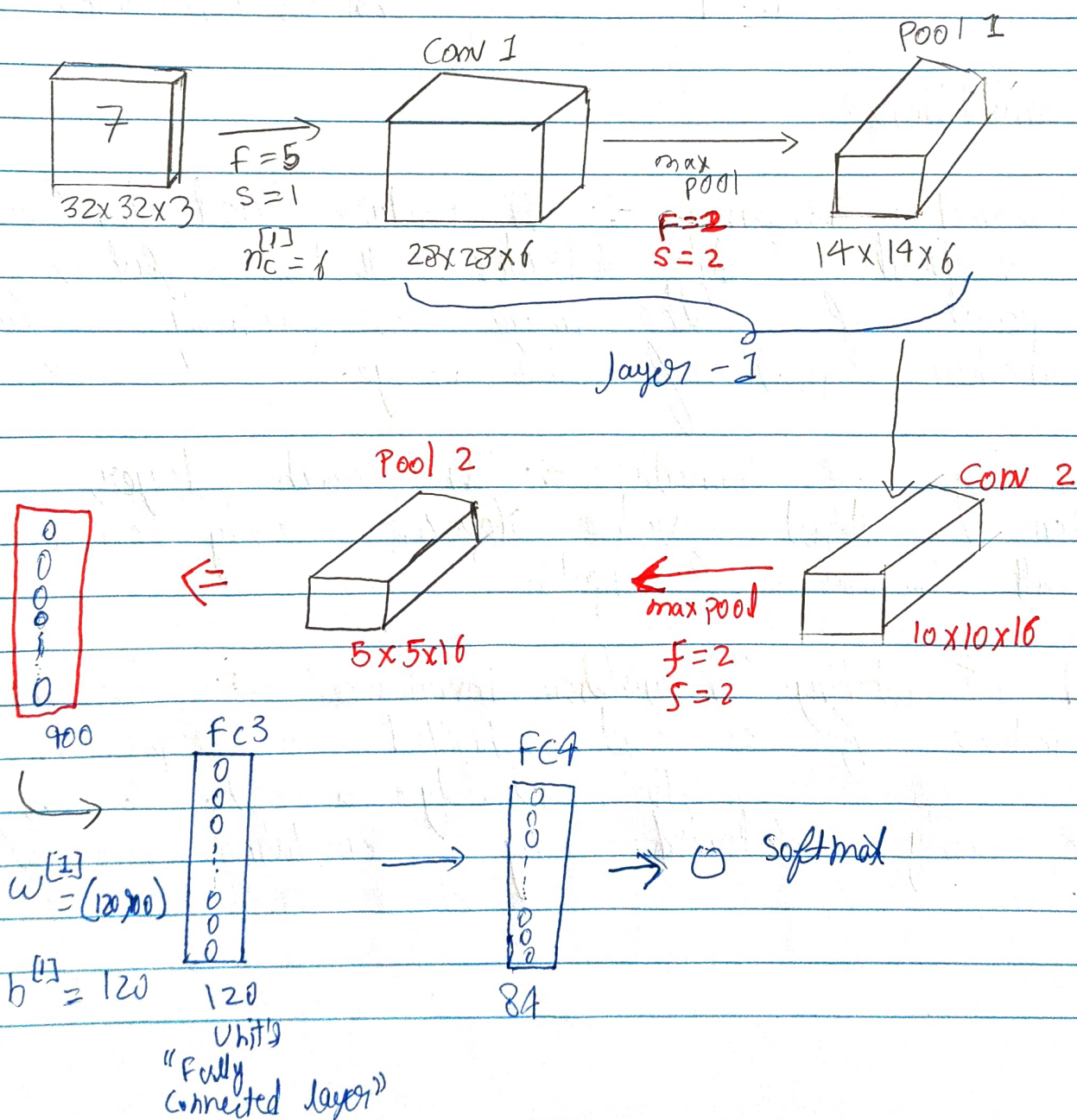
Pooling layer : Average pooling

→ it's the same as max pooling but here,

instead of taking the max filter no. from the no. numbers inside the filter we take the average of all the numbers

→ The pooling layers aren't trainable meaning no gradient descent optimization

CNN example



in the CNN example given behind this page we pair conv layers to pooling layers. & then we throw in some fully connected layers then we have a Softmax or any other activation unit at the end to get us output

as you go deeper in Neural Net,

$n_H, n_W \downarrow$ (Image height & width)
&
 $n_C \uparrow$ (no of channels increase)

Why Convolutions

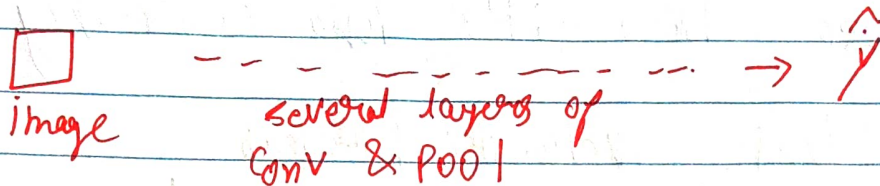
→ **Parameter sharing** $\because$ A feature detector (such as a vertical (edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections** $\because$ In each layer, each output value depends only on a small number of inputs.

→ **Good at catching "translation invariance"** $\because$ meaning a picture of a cat shifted a couple of pixels is still clearly a cat. & this is easily detected by convolution neural net.

Putting it together

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



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

Use gradient descent to optimize parameters to reduce J .

Week-2

Why look at case studies

↳ To build intuition

↳ to see how it's done correctly

→ Outline

↳

Classic Networks

→ LeNet-5

→ AlexNet

→ VGG

Modern

→ Resnet

→ Inception

Classic Networks

LeNet-5

had $\rightarrow$ 60k parameters

had sigmoid / ~~etc~~ tanh activation.

it has the same working as we have discussed earlier.

Alexnet

→ Similar to LeNet, but much bigger

→ Relu activations were introduced.

VGG-16

~~this was a very simple NN architecture.~~
~~if the size of the filter was 3×3~~
~~so, after convolution there~~
it has ,

Conv = 3×3 filter, | Max-Pool = 2×2 , $S=2$
 $S = \text{Stride} = 1$
Same padding

So in this architecture, we use the same padding hence to decrease the size of the n_H & n_W ~~& increase the number~~ ~~of~~, we use pooling layers.

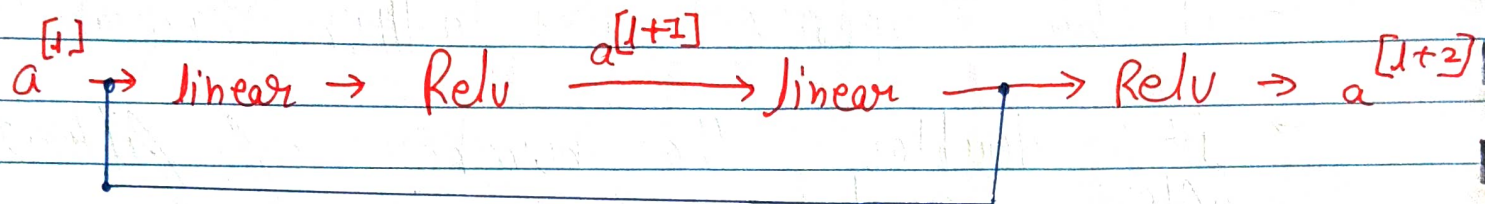
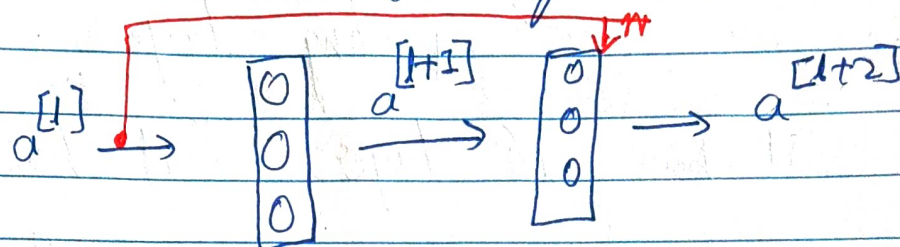
→ It has 16 layers that has weights

~ it has approx 138 million parameters

→ it doubles the number of filters after each pass.

Residual Networks (Res Nets)

→ resnet's are build out of residual blocks



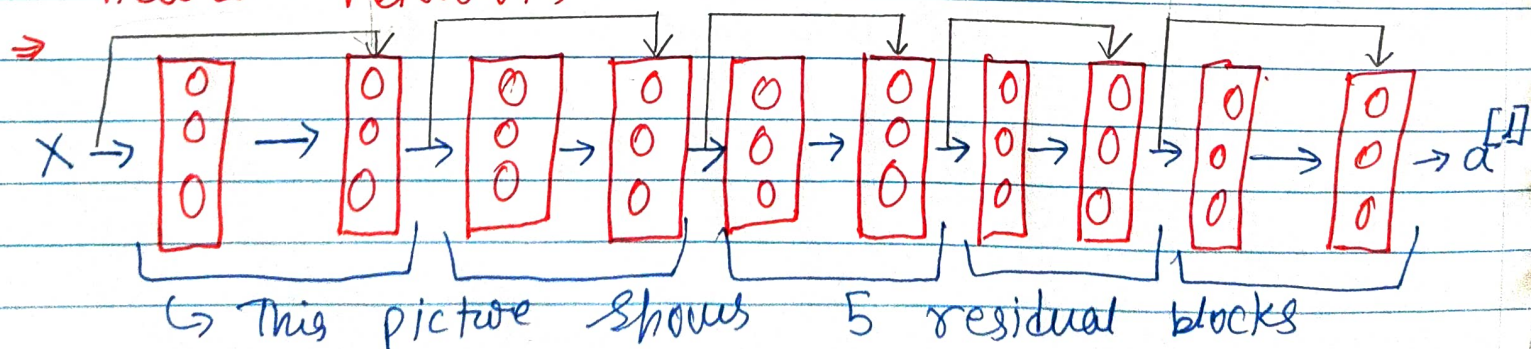
$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]} \rightarrow a^{[l+1]} = g(z^{[l+1]})$$

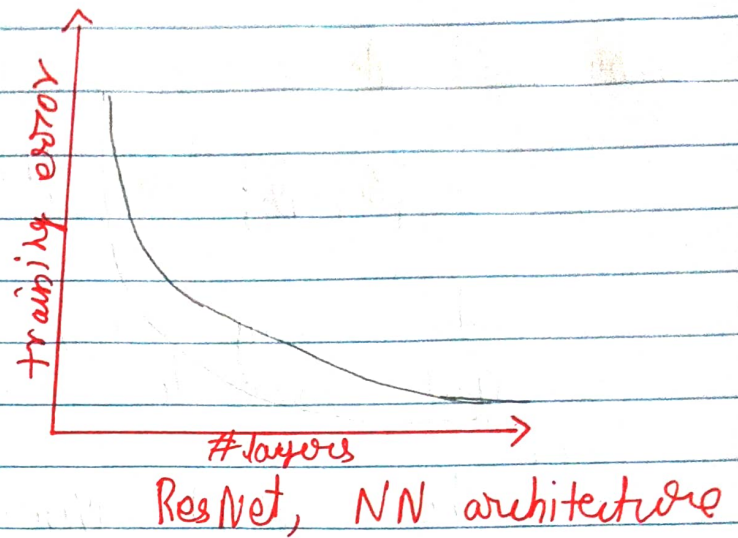
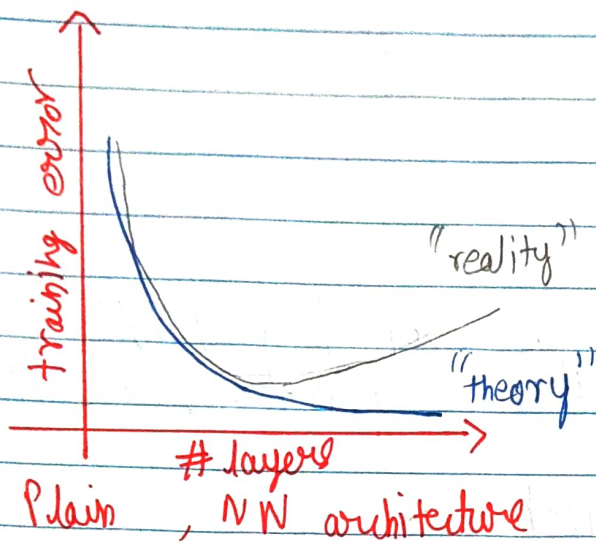
$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

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

→ in this an activation skips 2 linear layers & passes the input features, to the second activation.

it allows you to build much deeper neural networks





⇒ in theory the error should go down as you increase the number of layers in the neural network but in practical, as you increase the no of layers first the error will go down but after that the error will start to go up. & in that

case ~~the~~ ResNet's actually help because as you increase the number of layers the error most certainly goes down.

Why ResNets work?

$$x \rightarrow \boxed{\text{Big NN}} \rightarrow a^{[l]}$$

So let's say you have a very big NN & you have to make them learn something. Now,

the current Neural network has some problems learning identity function meaning give the same output as input, let's

say if the other layers in the Neural Net has got nothing to learn then they will make the weights & bias of that layers to go zero & the output will be same,

$$Cx \Rightarrow$$

$$\begin{aligned} a^{[l+2]} &= g(z^{l+2} + a^{[l]}) \\ &= g(\cancel{w^{[l+2]}} \cancel{a^{[l+1]}} + \cancel{b^{[l+2]}} + w_s a^{[l]}) \\ &\Rightarrow g(w_s a^{[l]}) \\ &\Rightarrow w_s a^{[l]} \end{aligned}$$

Here we ~~add~~ multiply ' w_s ' because let's say $(z)^{[l+2]}$ is $[256 \times 256]$ & $a^{[l]}$ is $[128 \times 128]$ then to make $a^{[l]}$ the same size as $z^{[l+2]}$,

$$w_s \rightarrow \mathbb{R}^{256 \times 128}$$

} a matrix of real numbers
rows $\rightarrow 256$
columns $\rightarrow 128$

Network in Network & 1x1 Convolutions

$$\begin{bmatrix} \end{bmatrix}_{6 \times 6} * \begin{bmatrix} 2 \end{bmatrix}_{1 \times 1} = \begin{bmatrix} \end{bmatrix}_{6 \times 6}$$

} you just end
up taking the
image &
multiplying it
by 2

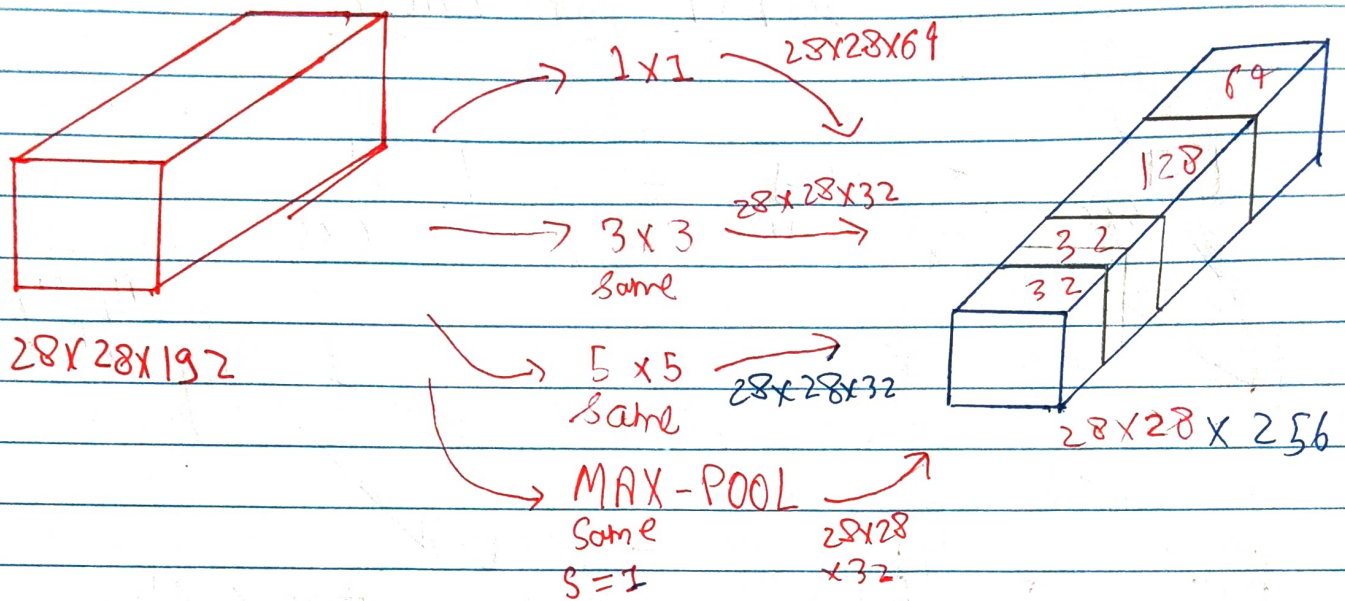
$$\begin{bmatrix} \end{bmatrix}_{6 \times 6 \times 32} * \begin{bmatrix} \end{bmatrix}_{1 \times 1 \times 32} = \begin{bmatrix} \end{bmatrix}_{6 \times 6 \times \text{\#No of filters}}$$

So, to shrink the height & width we have pooling layers but to shrink the channels or depth we can do that by ~~by~~ multiplying performing convolution using 1x1 filters

$$\begin{bmatrix} \end{bmatrix}_{28 \times 28 \times 192} * \begin{bmatrix} \end{bmatrix}_{1 \times 1} \longrightarrow \begin{bmatrix} \end{bmatrix}_{28 \times 28 \times 32}$$

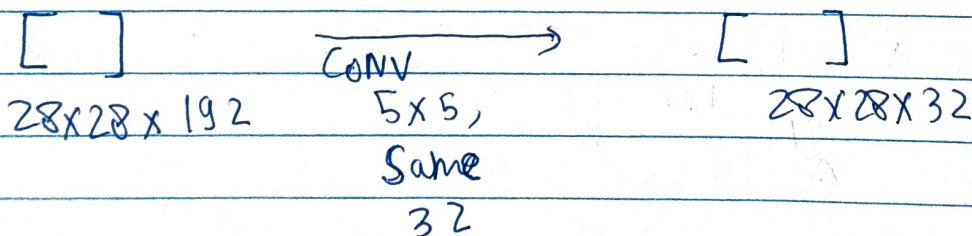
$\begin{bmatrix} 1 \end{bmatrix}_{n_c} = 32$

Inception Network Motivation



In the Inception Network, we implement multiple filters & pooling in a single iteration & stack up the output of all the filters & pooling to get a final output.

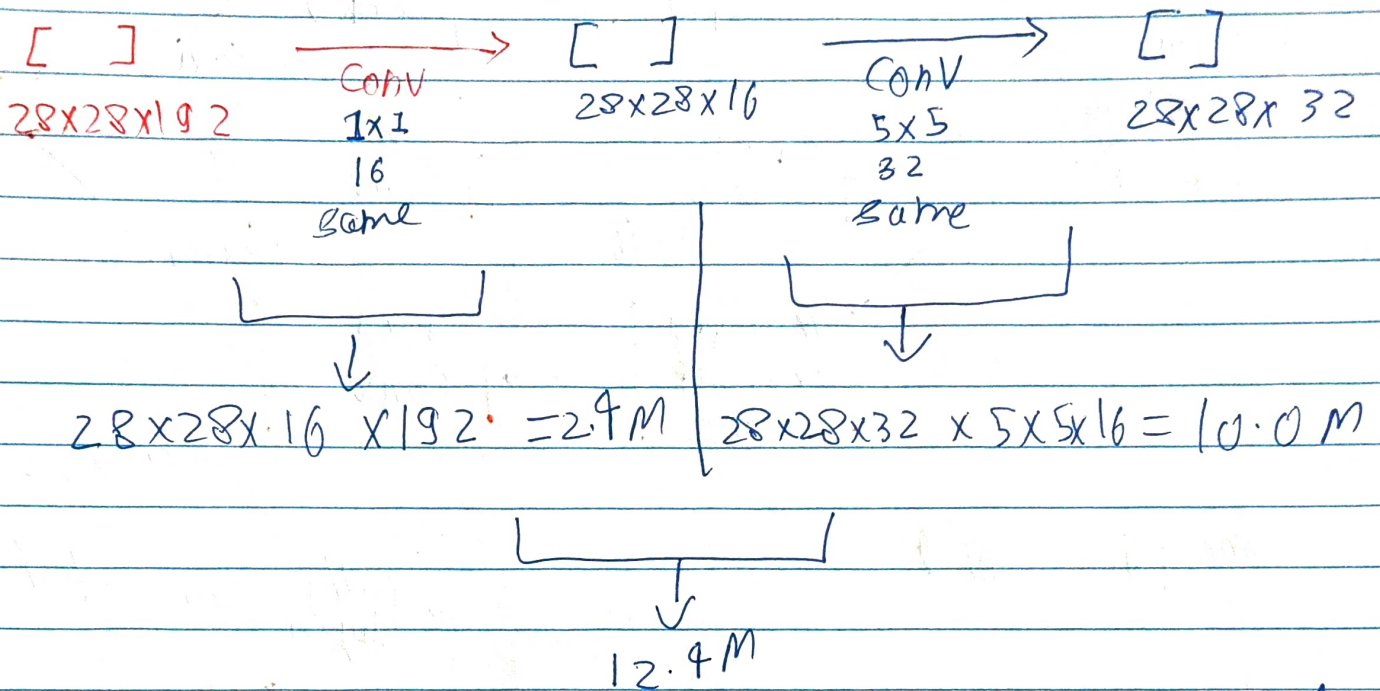
→ The problem of Computational Cost,



Here there are 32 ($5 \times 5 \times 192$) filters

$$28 \times 28 \times 32 \times 5 \times 5 \times 192 = 120 \text{ Million Operations}$$

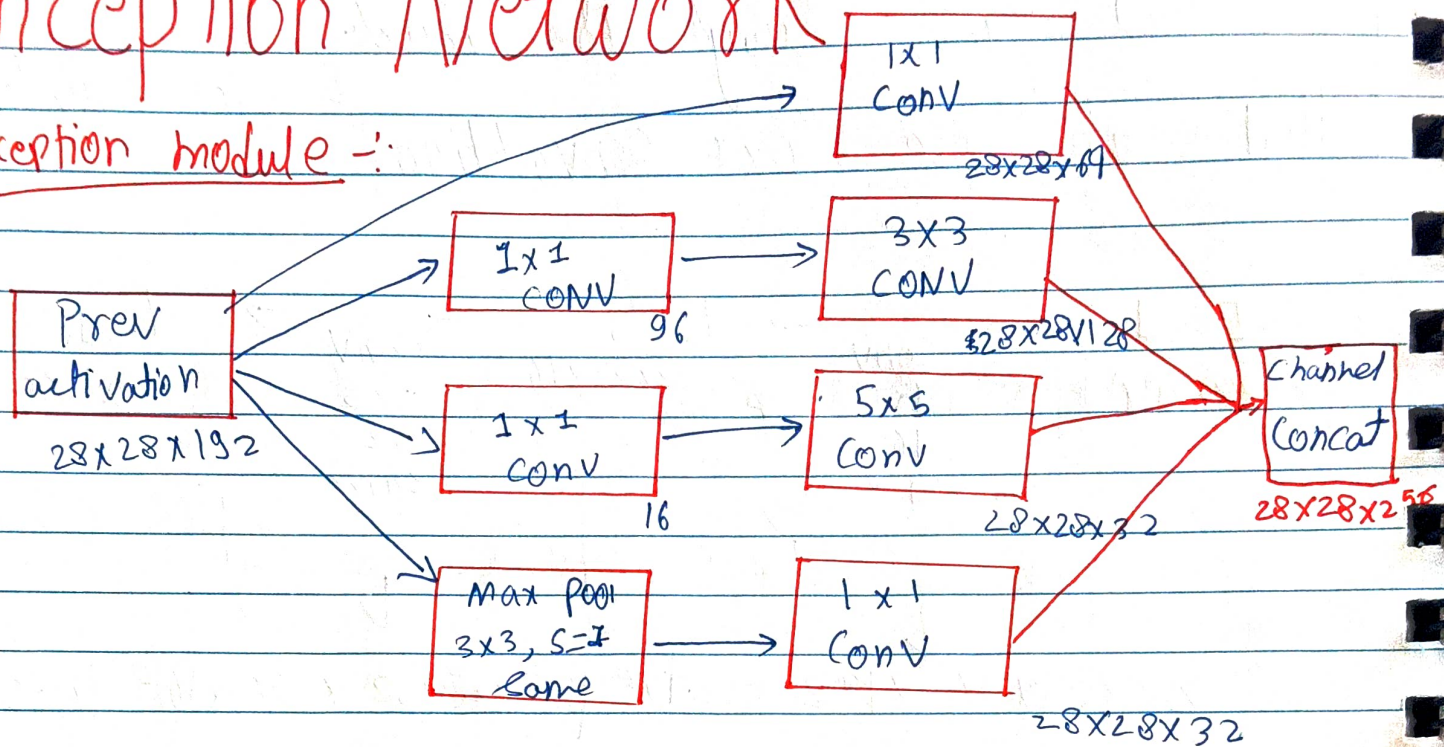
Here the number of operations are 120M
 & that is way too much. & to counter
 that we use "1x1 convolution"



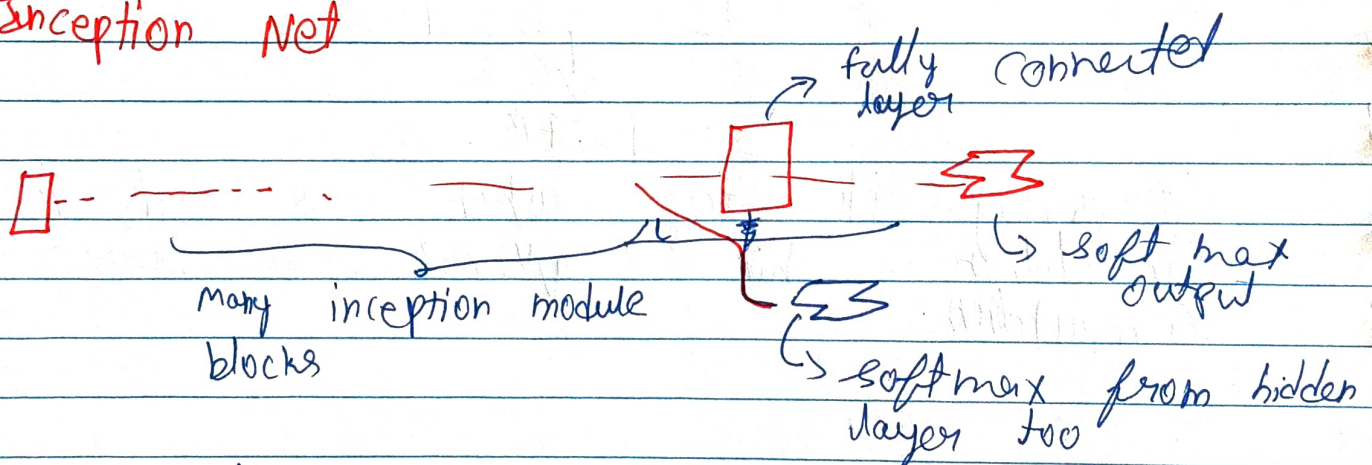
Comparing ~~it~~ to when we did direct
 Convolution this requires 10 times less
 computation.

Inception Network

Inception module :-



Inception Net



Inception Net is many inception modules layers & it also has softmax from hidden layers too it does sort off regularization in the NN architecture, & that's it

also The name is derived from the movie Inception. "We Need To Go Deeper"

MobileNet

- Low computational cost at deployment
- Here instead of doing normal convolution we do depthwise convolution
- In contrast with normal convolution you are going to have 2 separate operations in the depthwise convolution

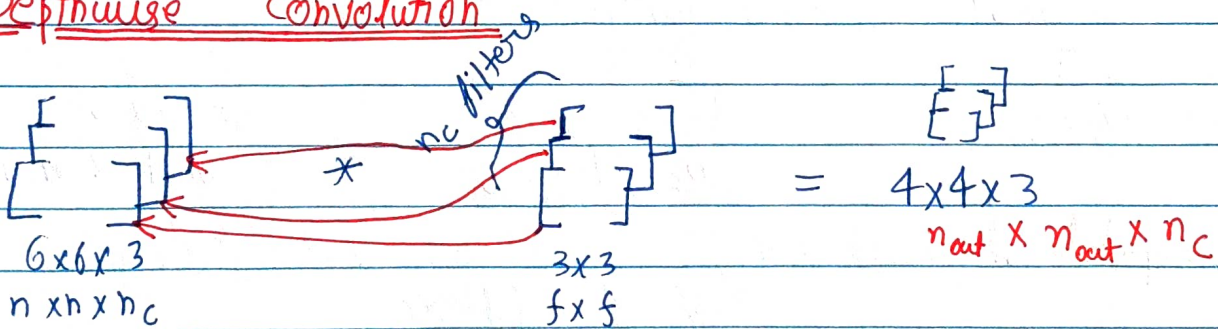
ex →

Depthwise Separable Convolution

Image 2

$$\begin{array}{c}
 \boxed{} \times \boxed{} \times \boxed{} = \boxed{} \\
 \text{nxnxnc} \quad \quad \quad \text{3x3xnc} \quad \quad \quad \text{nxnxnc} \quad \quad \quad \text{some dimension} \\
 \uparrow \quad \quad \quad \uparrow \\
 \text{Depthwise} \quad \text{Pointwise} \\
 \text{Convolution} \quad \text{conv}
 \end{array}$$

Depthwise Convolution



You apply one of each of the filters to one of each of these input channels

Computational cost = #filter params $\times$ #filter positions $\times$ #of filters.

$\Rightarrow 3 \times 3 \quad \times \quad 4 \times 4 \quad \times \quad 3$

$\Rightarrow 432$

Pointwise Convolution

$$\begin{array}{ccccc}
 \begin{array}{c} \boxed{\begin{array}{cc} \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \end{array}} \\ 4 \times 4 \times 3 \\ n_{out} \times n_{out} \times n_c \end{array} & * & \begin{array}{c} \boxed{\begin{array}{c} \boxed{} \\ \boxed{} \end{array}} \\ 1 \times 1 \times 3 \\ 1 \times 1 \times n_c \end{array} & = & \begin{array}{c} \boxed{\begin{array}{ccc} \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \end{array}} \\ 4 \times 4 \times 5 \\ n_{out} \times n_{out} \times n_c \end{array}
 \end{array}$$

$n_c \text{ filters} = 5$

Computational cost = $\frac{\# \text{filter params} \times \# \text{filter positions}}{\# \text{of filters}}$

$$\Rightarrow 1 \times 1 \times 3 \times 4 \times 4 \times 5$$

$$\Rightarrow \boxed{240} \text{ multiplications}$$

Total cost of depth wise separable convolution is $\therefore 240 + 432$

$$\Rightarrow \boxed{672}$$

for normal convolution

$$\begin{array}{ccccc}
 \boxed{} & * & \boxed{} & = & \boxed{} \\
 6 \times 6 \times 3 & & 3 \times 3 \times 3 & & 4 \times 4 \times 5 \\
 & & n_c \text{ filters} = 5 & & n_{out} \times n_{out} \times n_c
 \end{array}$$

Computational cost = $\frac{\# \text{filter params} \times \# \text{filter positions}}{\# \text{of filters}}$

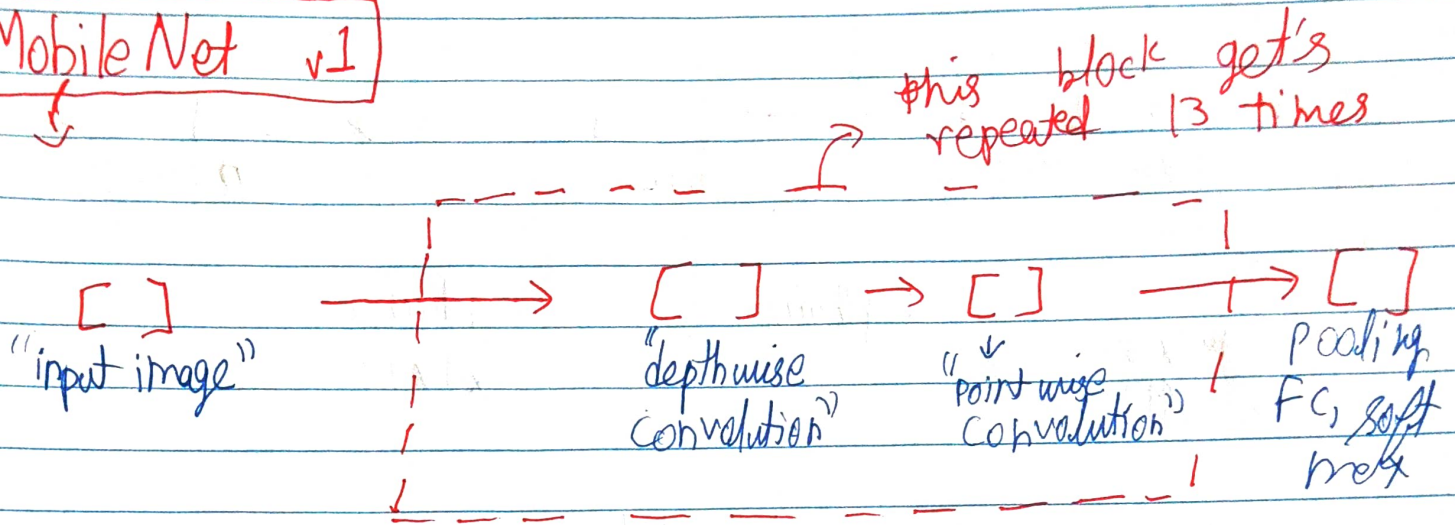
$$\Rightarrow 3 \times 3 \times 3 \times 4 \times 4 \times 5$$

$$\Rightarrow \boxed{2160}$$

$\Rightarrow$ To get the same ^{dimension} output, Mobile-net used 672 multiplication & Normal CNN used 2160 multiplications

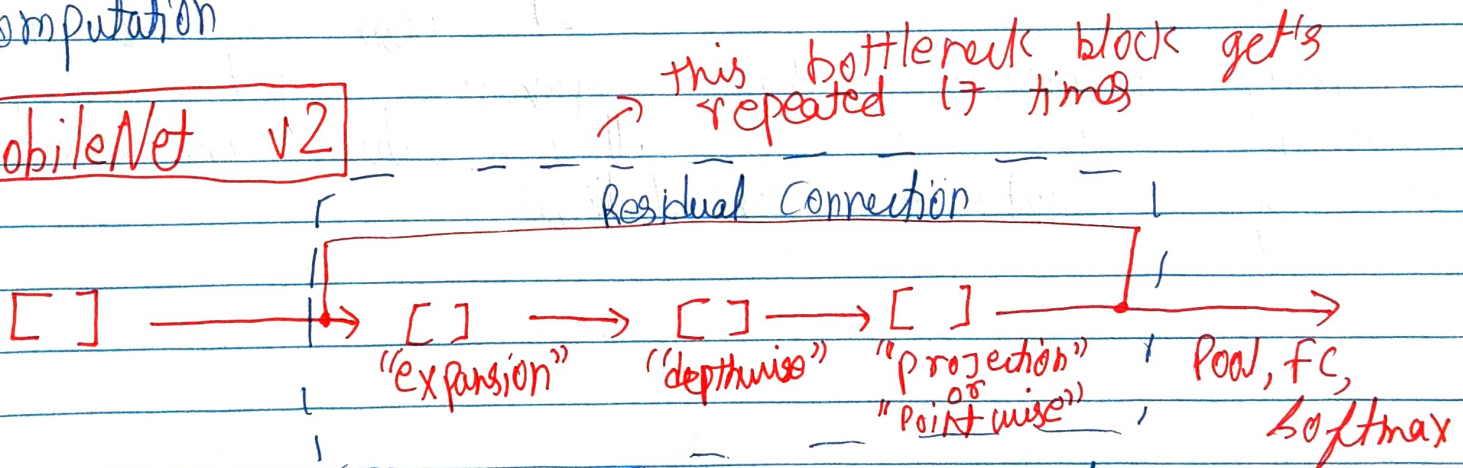
MobileNet Architecture

MobileNet v1



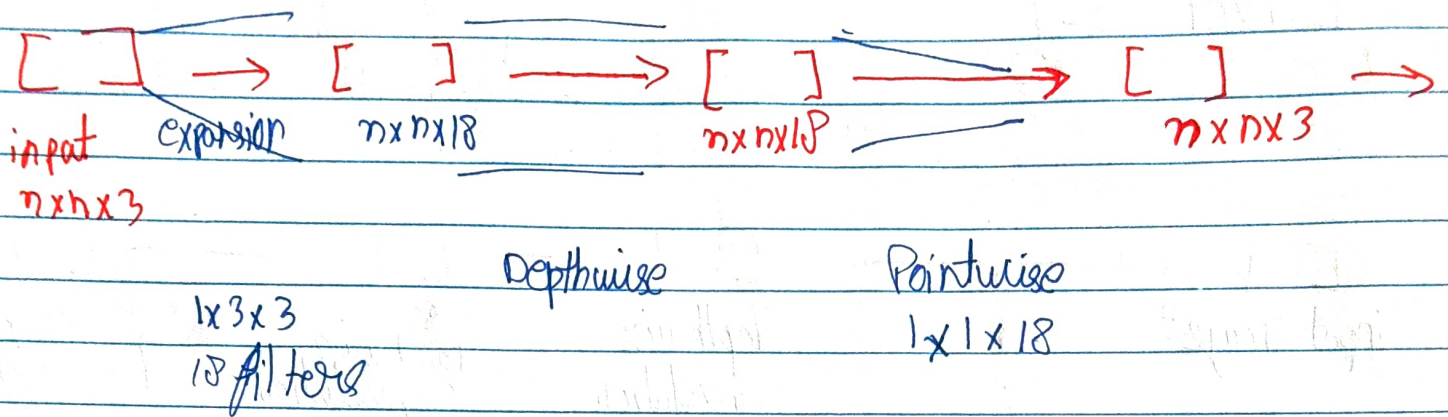
→ in mobile net architecture we replace the convolution operation filter by depth wise separable convolution, & it reduces the computation

MobileNet v2



here in the MobileNet v2 architecture we introduce an expansion layer too

MobileNet v2 Bottleneck



→ In Bottleneck block by expansion we increase the size of representation within the bottleneck block, & it allows NN to learn a richer function with just increase in a little bit computation.

It also saves more memory because the expansion gets reduced to a smaller value in the end using pointwise convolution.

Efficient Net

→ Based on the hardware requirements we need a Neural Net that works efficiently

→ We need to modify the NN to make sure that it works on that specific hardware

→ So how do we modify model size intelligently for that specific hardware.

→ **So how do we scale the model?**
→ there are 3 different knobs that you can tweak

1. **image resolution (r)**

→ increase resolution → more detail → heavy computation

→ decrease resolution → less detail → low computation.

⇒ But decreasing resolution will make you loose on data.

2. **Increase depth (d)** {Not ~~no~~ in channel's of image but the no of layers in Architecture}

→ Adding more layers result in more complex pattern recognition, but expensive in computation.

Using open-source implementation

3.7 Increase width (w) $\therefore$

or increase the width of the filter
& more stronger representation, it also increases

Computation cost

$\rightarrow$ you can scale all these 3 knobs
to get a desired architecture

Using open-source implementation.

- Neural Nets are difficult & finicky to replicate so doing it from scratch & finding good hyperparams, as a waste of time & tough ^, instead you can get the open-source implementation & build on top of it.
- "you should try to do it from scratch, it an opinion from learning perspective"

Transfer learning

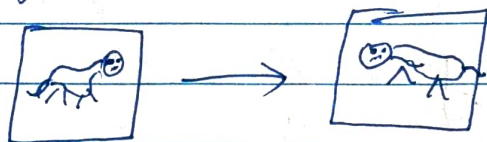
- When you have a small dataset
- You can just turn off all the layers of NN except the softmax one & train the layer.
- Pre activation caching - ∴ So you can just calculate the pre activations & save them & at the time of training you can use those pre activation insted of computing the pre activation everytime.
- If you have a larger dataset then you can just train some layers of the neural net
- if you have pretty huge dataset then you can just train every layer of the Neural Net

Data Augmentation

→ So in ^{training} CNN's we don't have large enough data on the internet so we do data augmentation to get or make more data to train the model

Types :-

(i) Mirroring :-



Just mirror the images

(ii) Random cropping :-

You can crop an image randomly to generate new data. Notice that cat's face should be in the image otherwise it's a random image.

(iii) Rotation :- rotate the image to some degree

(iv) Shearing :-



(v) Local warping

(iv) Color shifting - $\therefore$ change the RGB Channels of the image to get that same image in different color's

Data vs hand-engineering

$\rightarrow$ Since we don't have enough image ^{labeled} data on the internet we do ~~data~~ hand-engineering features, network architecture & other components to get the best out of our network.

Tip for doing well on benchmarks / winning Competitions

$\rightarrow$ Ensembling

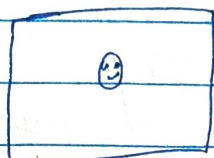
- Train several networks independently & average their outputs not for "deployment"

Multi-Crop at test time

$\rightarrow$ Run classifier on multiple versions of test images & average the result's
It will be good for the test time to get higher accuracy.

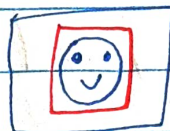
Week - 3
Obj detection

Object Localization



"smile"

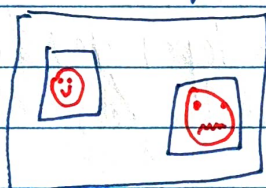
Image classification



"smile"

Classification with localization

Localization refers to figuring out where in the image the object is & drawing a boundary around it



Detection

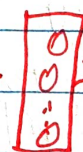
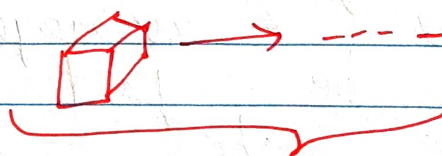
} various objects to detect in a single image.

Classification with localization

→

$\vec{x}$

image



~~set~~ ~~set~~ max

bounding box

b_x, b_y, b_h, b_w

Defining the target label y .

1. Pedestrian
2. Car
3. Motorcycle
4. background

} Need to output b_x, b_y, b_h, b_w , Class label (1-4)

→ Probability of an object being present in the image

$y =$

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

here
 b_x & b_y are coordinate of the centre point of the rectangle
 & b_h & b_w are the height & width of the ~~from~~ the point

c_1, c_2, c_3

are categories

then
$$L(\hat{y}, y) = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2 \text{ if}$$

$y_1 = 1$

$(\hat{y}_1 - y_1)^2 \text{ if } y_1 = 0$

Landmark detection

So, just like in object localization, in landmark detection, when

let's take example of face & let's say you want to find the landmark or coordinates of the ^{both} eyes, you can

do that by labeling the image & other parts of the face, & then you can get $y = \begin{bmatrix} \vdots \end{bmatrix}$ & then you can

form get loss & ~~create~~ perform gradient descent & optimize it

its application will be , Snap chat filters
where let's say a filter that
put's crown on someone's head,

So
what it will do is ~~take~~ take your
face image find the landmark points
& paste the crown on the
desired landmark which is your
forehead.

Object detection

Car detection example :-

→ So in the car detection example
you can have a # closely cropped
training set of cars. & the images with
cars are labeled 1.

Training set :



y
1



0



0

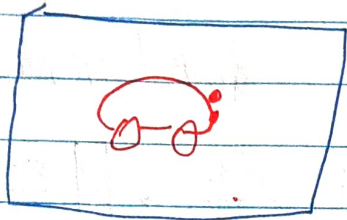


1

} A convnet will
take these closely
cropped images &
predict the output

" The sliding window approach :- in this
approach you start by picking a window
size & then you will input a

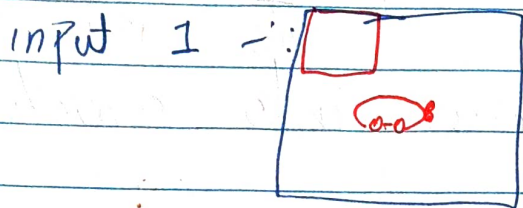
small rectangular region of the image
as same size of the window.



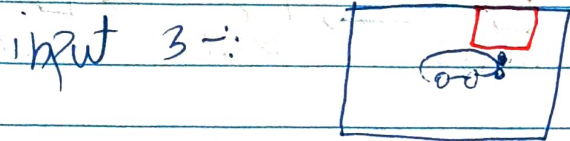
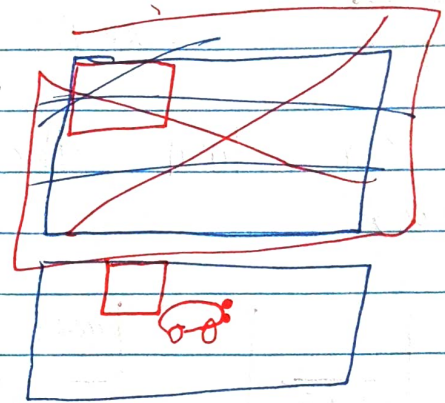
Image



↓
window



input 2 -:



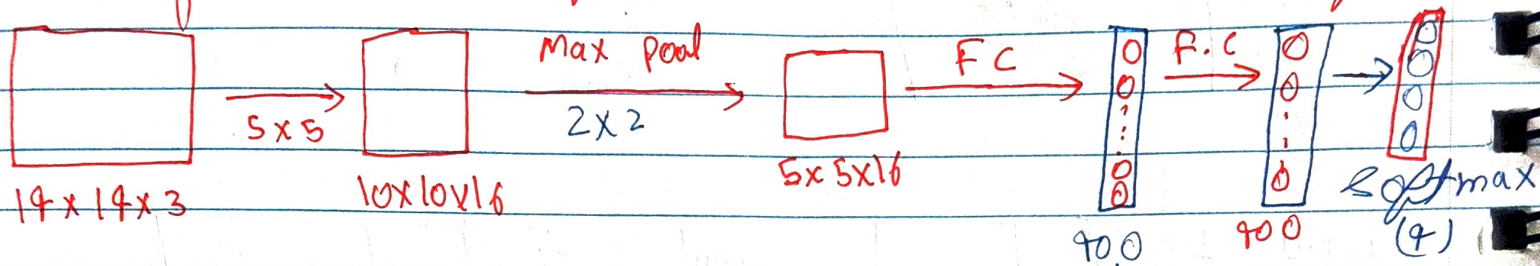
& so on

So we will basically pass these pieces
of window of the image to the conv net
to predict if there is a car or not
& if there is, the model will say '1', &
after you are done with a specific part of
the window then you can make the
window size bigger & try again.

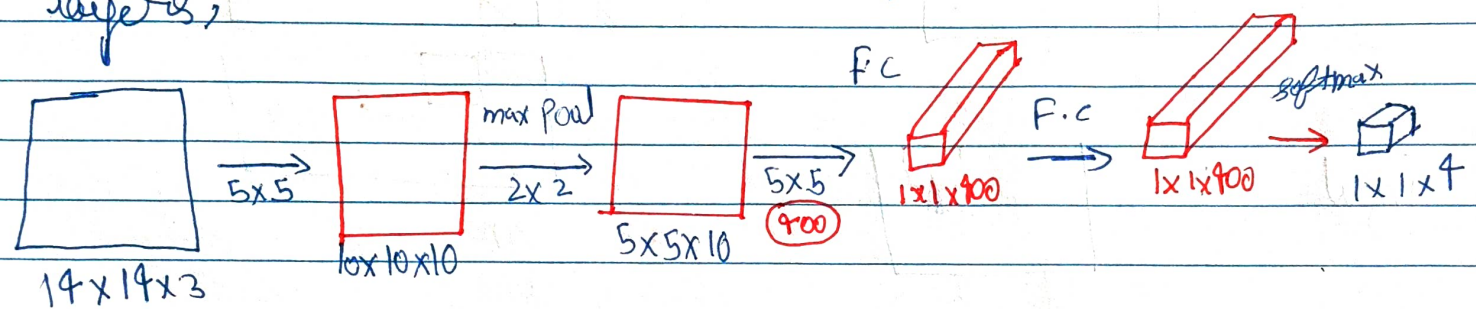
The disadvantage of sliding windows detection is
the computational cost, & unless you use a
fine very fine granularity or a very small
stride you end up not able to localize the object
that accurately within the image as well

Convolutional Implementation of the sliding windows

→ Turning FC layers into Convolutional layers

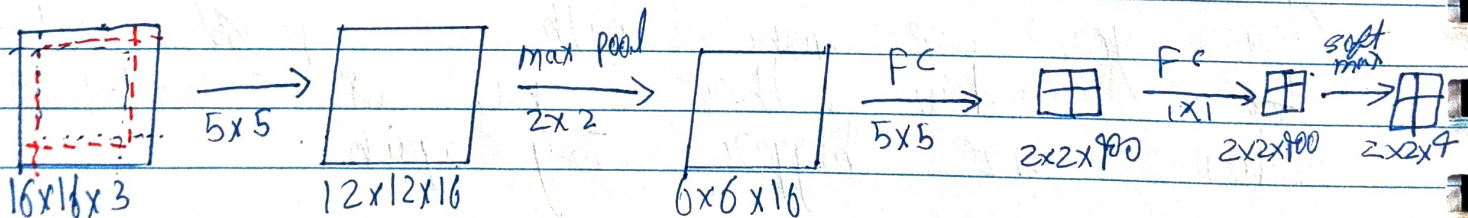


We can turn the FC layers into convolutional layers,



So if you try to enter an image with size greater than 14x14x3 then what will happen?

So let's say you enter a 16x16x3 image then



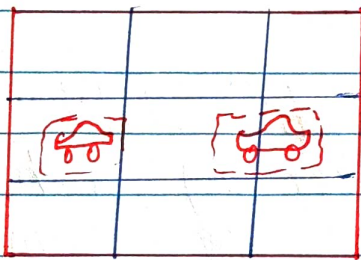
So, to completely check the 16x16x3 image we will require 4 ~~convolution~~ ^{forward pass} operations but, implementing the above method you can do it in one single pass, the output you get is 2x2x4

meaning the output of all four images
 $2 \times 2 \Rightarrow 4$, & has depth 4, meaning
 4 windows ~~images~~ output's & depth is the four classes
 for each image.

Bounding box Predictions

Yolo Algorithm $\therefore$

you get Yolo "You only look once" in this
 also you draw grid on the image



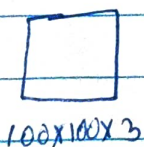
labels for each grid

$\rightarrow$

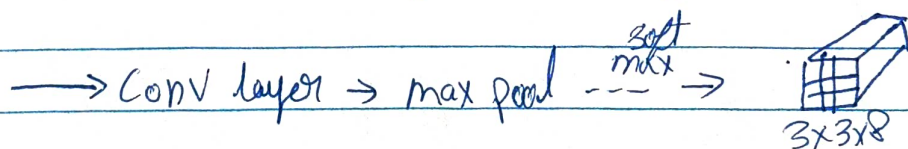
P_c
b_x
b_y
b_h
b_w
C_1
C_2
C_3

$P_c \rightarrow$ object present or not.
 $b_x, b_y \rightarrow$ x & y coordinate of the image boundary
 $b_h, b_w \rightarrow$ height & width of image boundary
 $C_1, C_2, C_3 \rightarrow$ classes of the object.

Now,



100x100x3



3x3x8

$\rightarrow$ Each grid in the output will have ~~8~~ ^{depth} & this time, since the algorithm has the context of all the grids it can output much better bounding box of the image even if the car isn't fully inside a grid.

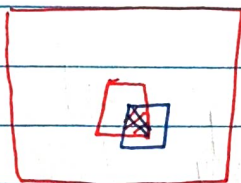
Intersection over union

→ Evaluating object localization

→ how can you tell if an algorithm is good at evaluating object localization.

→ one way to do it is Intersection over union,

let's say you have an image



& here the red is the predicted boundary & blue is actual boundary.

So the intersection is the ~~shaded~~ shaded area & the area surrounded taken by both boxes is the area of union so union over intersection would be,

$$\Rightarrow \frac{\text{size of shaded area}}{\text{Area covered by both boxes}}$$

Correct if $\text{IoU} \geq 0.5$, or whatever value you decide.

NON - Max Suppression

→ Your algorithm may find multiple detection of the same object, It makes sure that your algo detect's the object only one, Each grid cell predict's

- Prob an obj exist's (P_c)
- bounding box coordinates.

so multiple grid boxes detect the prob of the car

→ Core idea is to keep the best bounding box & remove the duplicate overlapping ones.

Step-1 → Remove weak predictions.
→ discard boxes where
 $P_c < \text{threshold}$.
ex $\Rightarrow P_c < 0.6$

Step-2 → Pick the Strongest box
→ choose the box with highest probability,

Step-3 → Remove overlapping boxes.
→ if another boxes overlaps a bit with the chosen one
ex $\text{IoU} > \text{threshold}$ (ex: 0.5)
remove it

Step-4 - Repeat

- from remaining boxes
- Pick the next highest
- Check overlap again
- Remove overlapping boxes

Step-5 - Continue until no boxes remain
final prediction.

Car #1.

Car #2.

→ NMS is applied per object class.

Ex → class : ① cars ② Pedestrian ③ Motorcycle

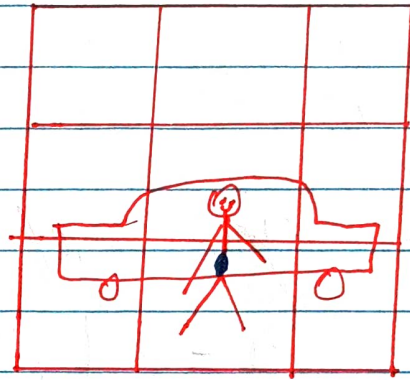
You run NMS for 3 times

→ because overlapping boxes of different object's shouldn't be removed.

Anchor boxes

→ each of the grid cells can only detect one object,
what if a grid cell wants to detect multiple objects?,
we use the idea of anchor boxes,

Ex →



let's say the mid point of both of the object is in the same ^{block of} grid.

So, if a grid cell can only output

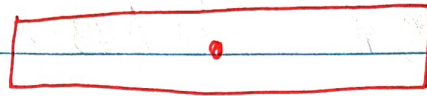
$$\hat{y} = \begin{bmatrix} p_c \\ b_n \\ b_y \\ b_n \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

, then it can't give two outputs because the no. of coordinate it can generate is 1,

now, with anchor boxes what we are going to do is predefine shapes called anchor boxes, & now you associate 2 predictions with 2 anchor boxes



Anchor box: 1



Anchor box: 2

So you define class label as

$y =$

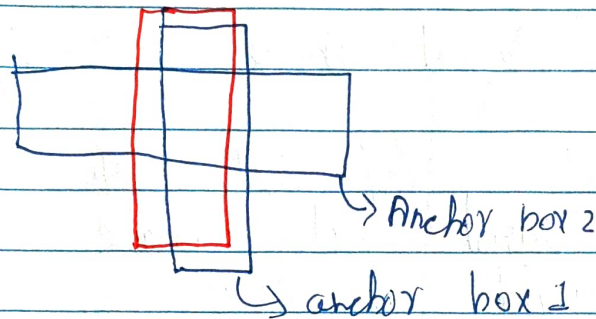
P_c
b_x
b_y
b_h
b_w
C_1
C_2
C_3
P_c
b_x
b_y
b_h
b_w
C_1
C_2
C_3

8 output's associated with anchor box 1
"use this for pedestrian"

8 output's associated with anchor box 2
"use this for car"

Previously :- Each object in training image is assigned to grid cell that contains that object's midpoint

with two anchor boxes :- Each object in training image is assigned to grid cell that contains object's midpoint & anchor box for the grid cell with highest IOU.



→ in the above diagram the ground truth label is in red, & the anchor box 1 is overlapping ~~more~~ with ground truth ^{more} than anchor box 2

So, anchor box 1 will be assigned
So the highest IOU boxes will be assigned

So, now the output y will be "3x3x16"

YOLO Algorithm

"You only look once" in this algo you look at the image once & get the output of object's boundaries & their predictions.

- images are split into grids
- each grid is assigned a finite amount of anchor boxes.
- for each object there is only one anchor box.
- the image gets fed into the Neural Net & then the neural net make predictions on all the grid at once.
- during prediction the algo outputs lot's of boxes
- ~~to~~ the boxes with low probability are & dropped.
- then we use Non-Max Suppression to get the final prediction.

Region Proposals or R-CNN

Now, to make the yolo algorithm more fast, what we can do is, only check the regions that might have object's init, there are some regions which have no objects, the

R-CNN only performs the check on the regions that might have object & it does it with the help of segmentation algorithm

R-CNN: Proposes regions classify proposed regions one at a time. Output label + bounding box.

Fast R-CNN: Proposes regions use convolution implementation of sliding windows to classify all the proposed regions.
the fast R-CNN is still quite slow because it takes time to propose regions.

Faster R-CNN: Use Convolutional network to propose regions.

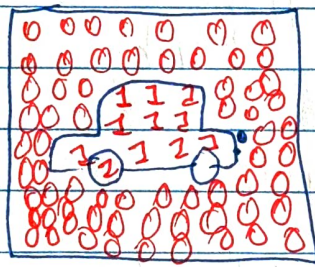
Semantic Segmentation with U-Net

→ Here the goal is to draw an outline around the object detected, so that you know which pixel belong to the object & which don't

for example :-

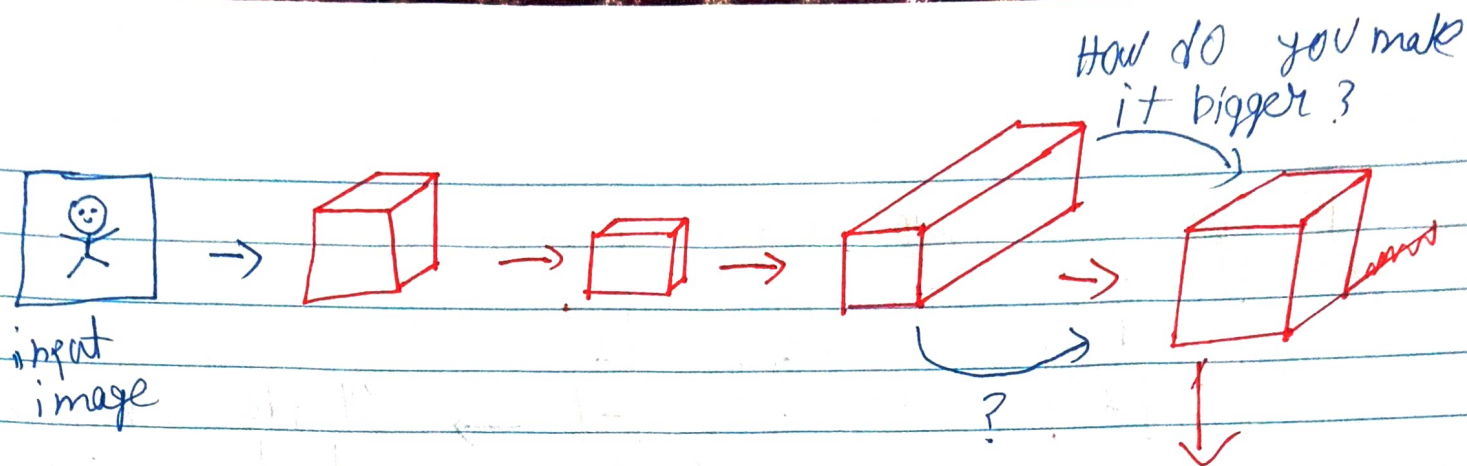
A road lane, if the algorithm rather than detecting over & placing a box over the road, the algorithm will try to map every single pixel as drivable road or not

→ There are per pixel class labels.



→ In this case if you want to segment out car from the background then every pixel is labeled "0" except car which is labeled as "1".

deep learning for semantic segmentation
so it's the same as the object detection you just get rid of the last few F.C layers & know that you gradually make the output bigger & the channels smaller



→ The input is the human image the output is human segmented from the background



→ But how do you get from a small height & width to the greater height & width, It is done by implementing

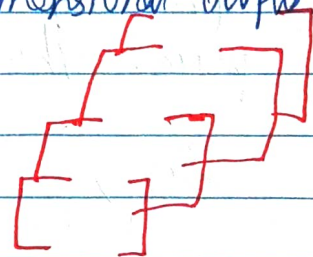
Transpose Convolution

→ How do you take a 2×2 input & blow it up to get 4×4 dimensional output

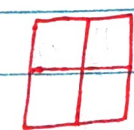
Normal convolution

$$\begin{bmatrix} \square & \square \\ \square & \square \end{bmatrix} * \begin{bmatrix} \square & \square \\ \square & \square \end{bmatrix} =$$

input image filter

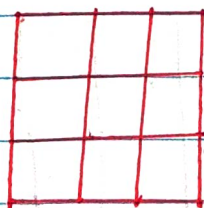


Transpose Convolution



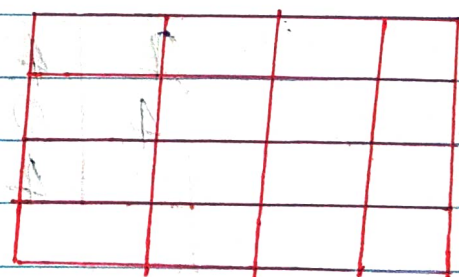
2×2

*



3×3

=



4×4

Transpose Convolution example

2	1
3	2

2x2

input image

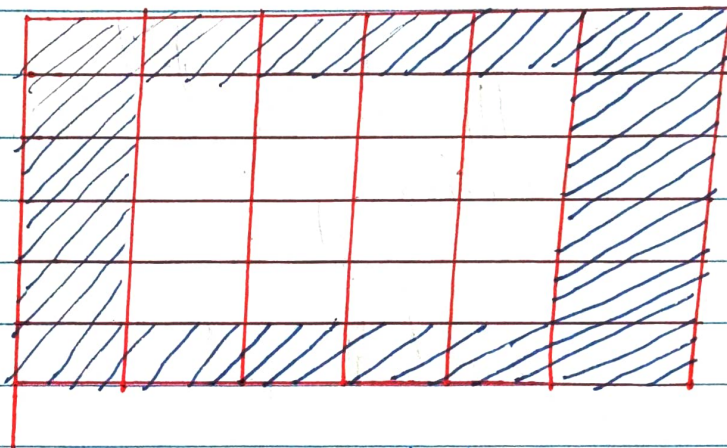
*

1	2	1
2	0	1
0	2	1

filter

$f \times f = 3 \times 3$

→ stride on the output, $s = 2$
padding = 1



So step 1, multiply the first element with all the elements in the filter & then place the output result on the output

So, result $\Rightarrow$

2	4	2
4	0	2
0	4	2

So on the output it will look like.

2	4	2		
4	0	2		
0	4	2		

ignore the results in the shaded area

Now again do this for all elements. ~~into~~
 & if the ^v output ~~result's~~ overlap in the output simply
 add them,

output →

2	4	2	2	1
4	0	2+2	0	1
0	4	2+0	2	1

result →

1	2	1
2	0	1
0	2	1

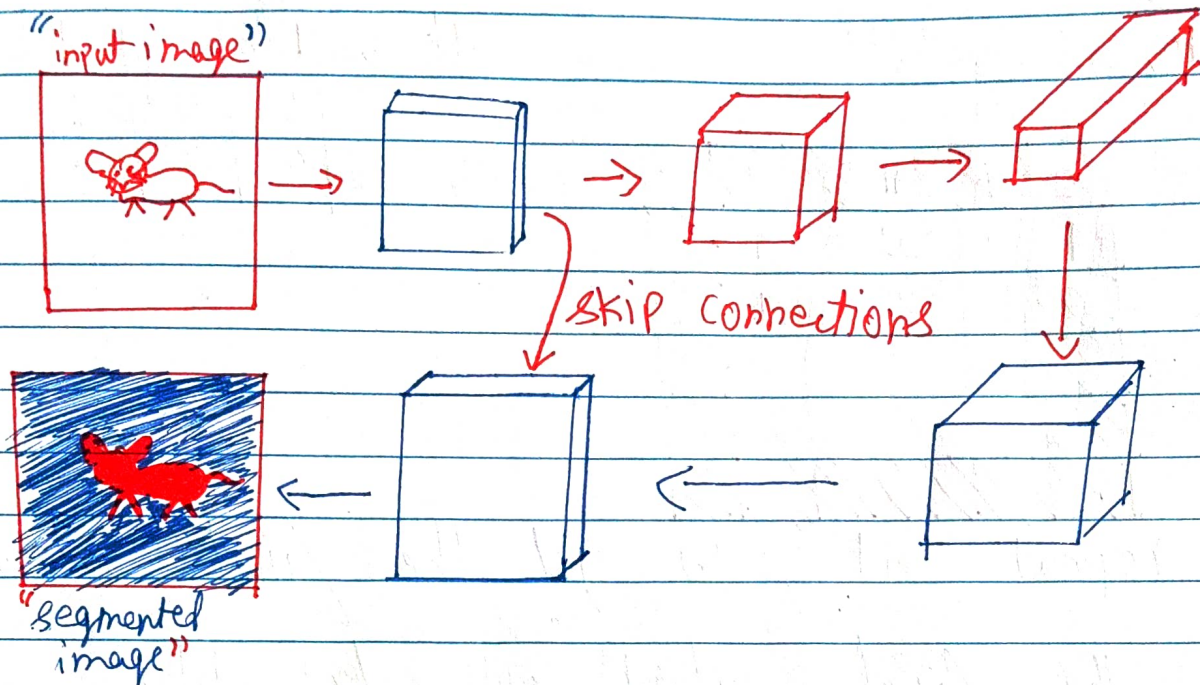
now repeat it to get the final output

So, the final output will look like

0	4	0	1
10	7	6	3
0	7	0	2
6	3	4	2

after the operation the 2x2 dimension
 input image became 4x4 dimension.

U-NET Architecture



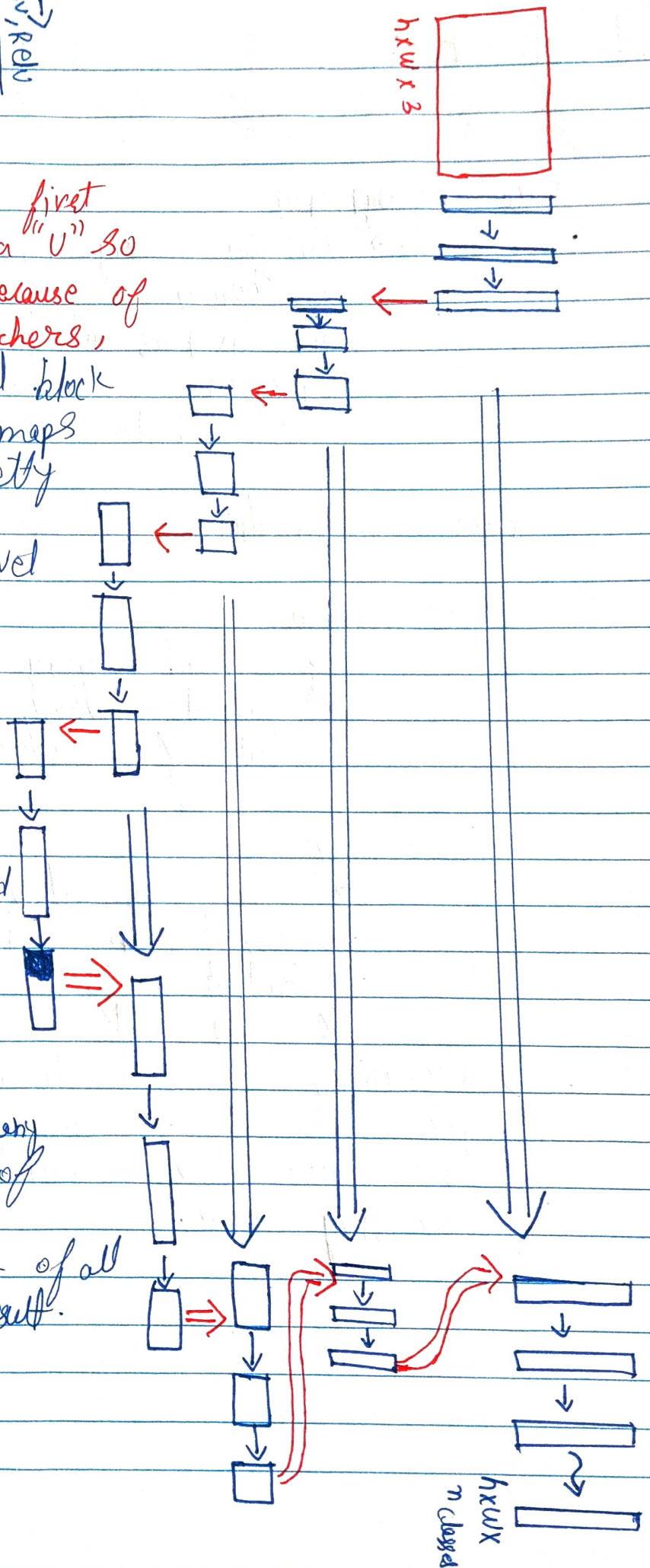
→ The U-Net Architecture introduces skip connections that makes the prediction better

$\Rightarrow$ skip
 $\Rightarrow$ conv
 $\Rightarrow$ conv 1×1
 $\Rightarrow$ max pool
 $\Rightarrow$ Trans conv
 $\Rightarrow$ conv, ReLU

So, in the U-Net, well first
 of all it look's like a "U" so
 it's called a U-Net because of
 that goofy ahh ~~for~~ researchers,
 & Yeah it has residual block
 like connections & it maps
 out the segmentation pretty
 accurately,
 so that's U-Net
 for U, HaHaHa lol,

we just shrink the size
 & then make it bigger the
 fkin figure is self
 explanatory matter of
 fact if you can't understand
 it by the diagram, my
 g you might be
 dumb.

the output will have as many
 channels as the number of
 classes,
 of them you take argmax of all
 to get the result.



Week - 4 :: Special Applications: Face recognition & neural style transfer

Face recognition

- ↳ (i) face recognition :: detecting the face of a human
- (ii) liveness detection :: detecting whether the person showed is alive or just a photo

Face verification vs face recognition.

verification

- input image, name/ID
- output whether the input image is that of claimed person.

recognition

- Has a database of K persons
 - Get an input image
 - output ID if the image is any of the K persons or "not recognised"
- recognition is a tough task because if you have n amount of employees then you have n number of chances of messing up.

One shot learning

→ It means learning from one example,

so if you were to ^{train} give a neural net the images of employee & take a softmax output it might not work,

because you don't have a lot of training examples & what if you have a new employee, so that is not doable,

Instead you need a Neural Net to learn a "similarity" function
so let's say the fn is -:

$d(\text{img 1}, \text{img 2}) =$ degree of difference between images.

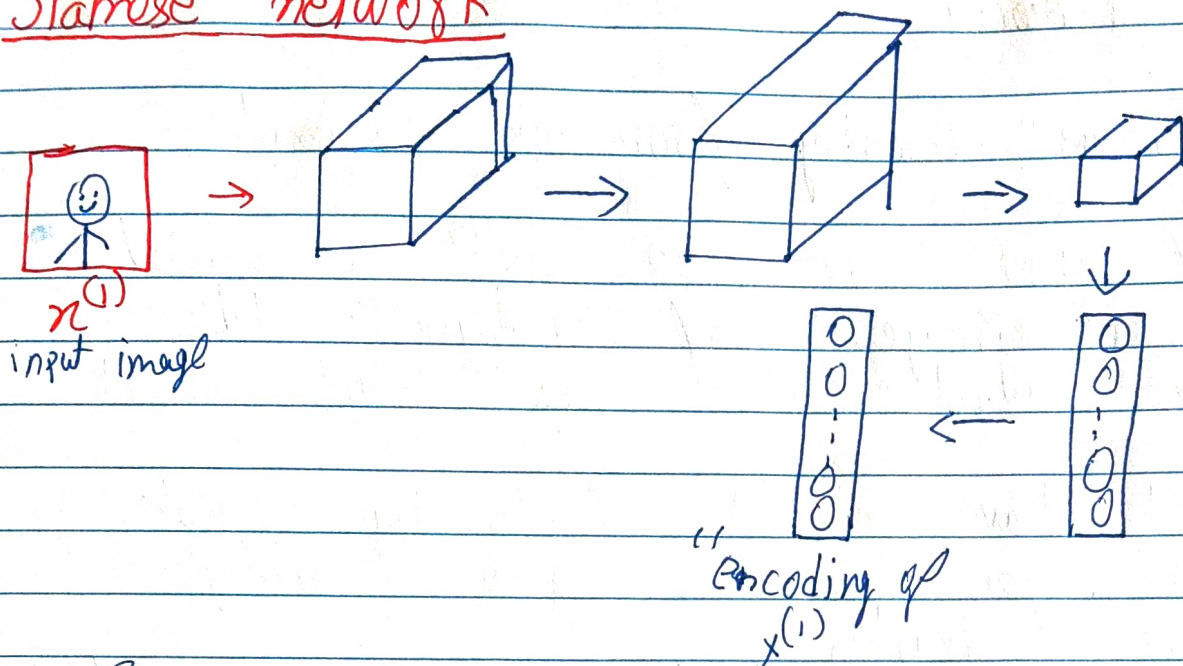
if $d(\text{img 1}, \text{img 2}) \leq \tau$ "same"

if $d(\text{img 1}, \text{img 2}) > \tau$ "different"

} verification

$\tau \rightarrow$ it's just a constant that we use for the threshold

Siamose network



So, remove the softmax or any other last layer from the neural net & treat the output like an encoding of the image

Similarly $x^{(2)} = \dots$

So, for the fn $d(x^{(1)}, x^{(2)})$ we will take the L2-norm,

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

→ Now, use back propagation to learn the parameters so that,

if $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is small

if $x^{(i)}, x^{(j)}$ are different persons, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is large

Triplet loss :-

Learning objective

You will have 3 images (i) Anchor image
(ii) Positive image (iii) Negative image.

You will always be looking at 3 images at a time.

You want :-

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

Now,

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0 \quad - (i)$$

→ we will slightly change the eqn (i) because one trivial way to satisfy is the functions start outputting 0, so $0 \leq 0$, hence we need to introduce a new hyperparameter called margin " α "

So we let's say the difference btw $d(A,P)$ & $d(A,N)$ be smaller than zero,

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

So

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

for example

$$d(A, P) = 0.5 \text{ \& } d(A, N) = 0.51$$

even though $0.51 > 0.5$, that's not good enough
we want much bigger gap,
gap to be at least 0.7 or higher,

then you want the
The margin
parameter pushes anchor +ve pairs & anchor -ve
pairs away from each other.

Loss function will look like
given 3 images, A, P, N :

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

So,

as long as long as,

$d(A, P) - d(A, N) + \alpha \leq 0$, the loss
is zero but if not then that
number will be the loss
so, cost J will become,

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

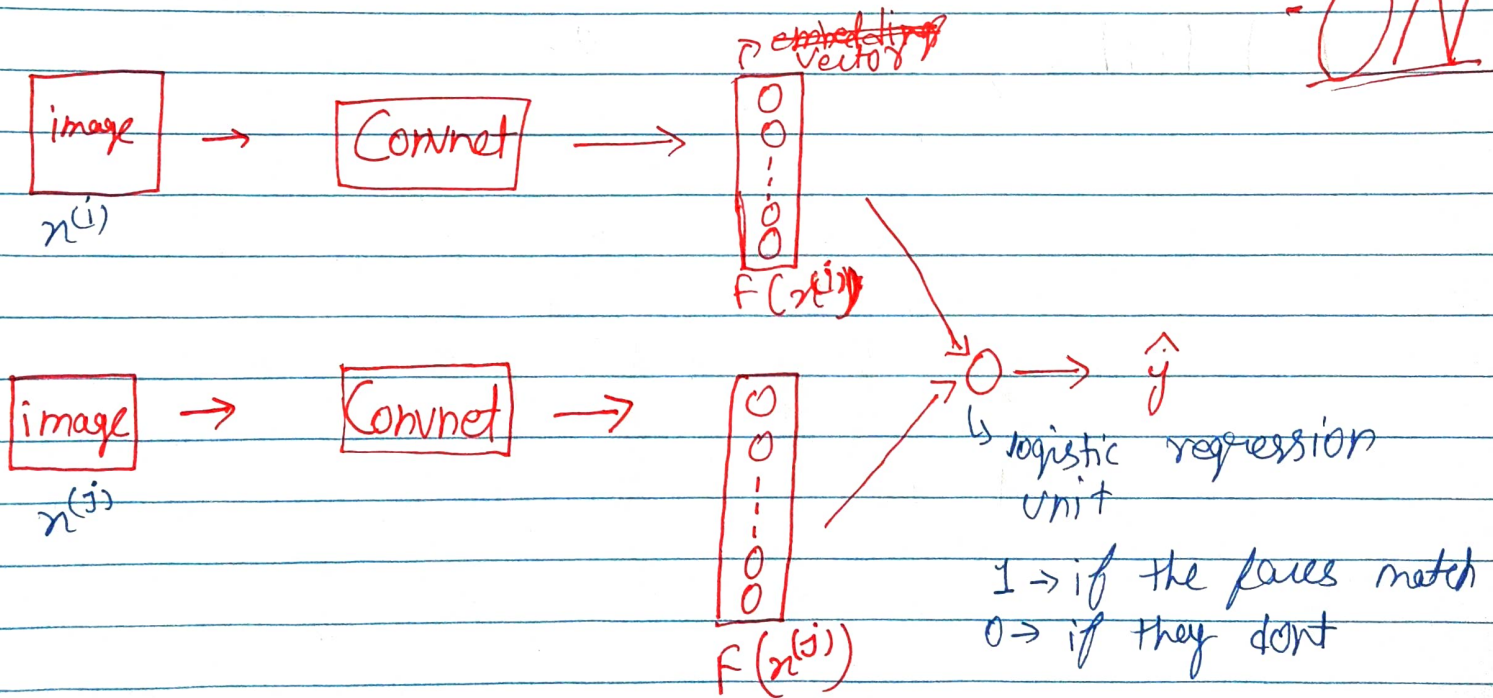
Training set: 10 k pictures of 2 k persons,
so, you should have at least 10 images of
a person otherwise the algo will not work.

So how do you choose the triplet's (A, P, N) Anchor, positive, Negative?

Solⁿ $\Rightarrow$

During training, if A, P, N are chosen randomly, $d(A, P) + \alpha \leq d(A, N)$ is easily satisfied

Face Verification & BINARY CLASSIFICATION



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

$\rightarrow$ We take the absolute values of element wise differences between the two vectors as an input.

$F \rightarrow$ The subscript k means to select out the k^{th} component of the vector F

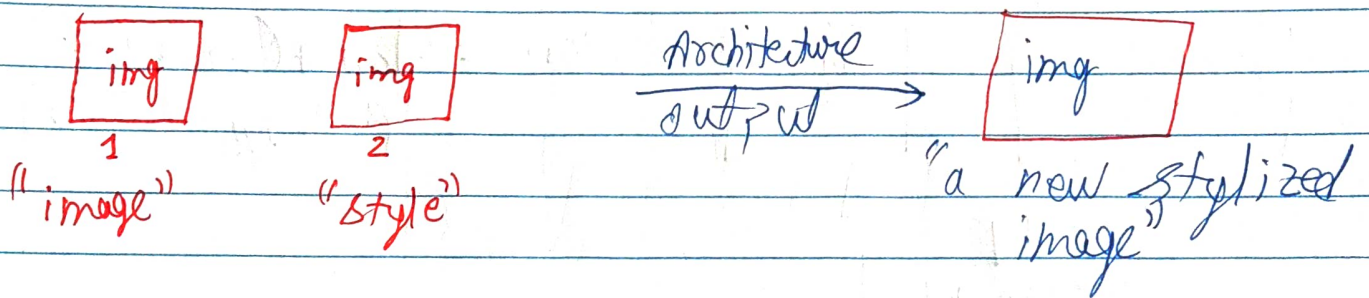
→ one way to speed up the verification process at the deployment time is to already compute the feature vector in your database & that can be used instantly for logistic regression

So, in supervised learning approach you take pairs of images not triplets & label them 1 if same person & 0 if not

x		y
		1
		0

Neural Style transfer

→ It is a process of transferring an art style from one image to another, for example,

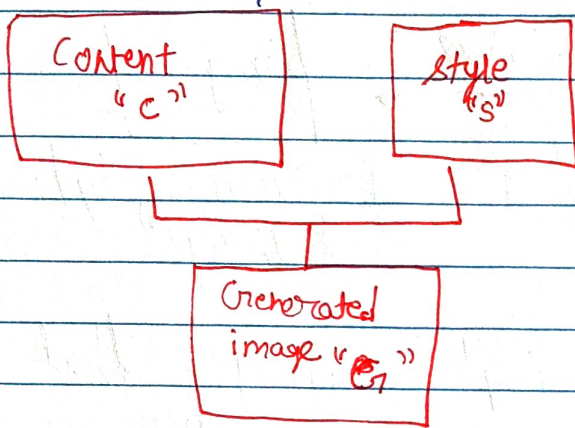


ex → you have a simple image that you captured & then you have a style image, for ex → a style can be like a painting or any other image which has a specific style or something that makes it stand out. Now you can just transfer that style to the image captured or just any other image.

What are deep ConvNet's learning?

→ So the first few layers detect's edges & textures but the deeper layers detect the complex objects.

Cost function :-



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

the fn $J_{\text{content}}(C, G)$ measures how similar content & Generated image are,

$J_{\text{style}}(S, G)$ measures how similar is the style of image S to G, & we

weight them with two hyper parameters α & β to specify the relative weighting between the content costs & the style cost.

Now this is what you do to generate the image

1> Initiate G randomly

$G: 100 \times 100 \times 3$

2> Use Gradient descent to minimize $J(G)$

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

that we are updating

here " G " is the pixel values of this image

Content Cost Function

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

↳ Content cost function

→ Say you use hidden layer " l " to compute content cost.

→ Use Pre-trained ConvNet. (E.g., VGG Net)

⇒ "So in the hidden layer " l " that you choose if you choose the first few layers the Neural Net will try to match the exact edges of the images meaning it will be a lot similar if too deep then it might try to match specific object which is not good. So choose the layer which is not too deep or too shallow.

→ & now let $a^{[l]}(c)$ & $a^{[l]}(G)$ be the activation of layer l on the images.

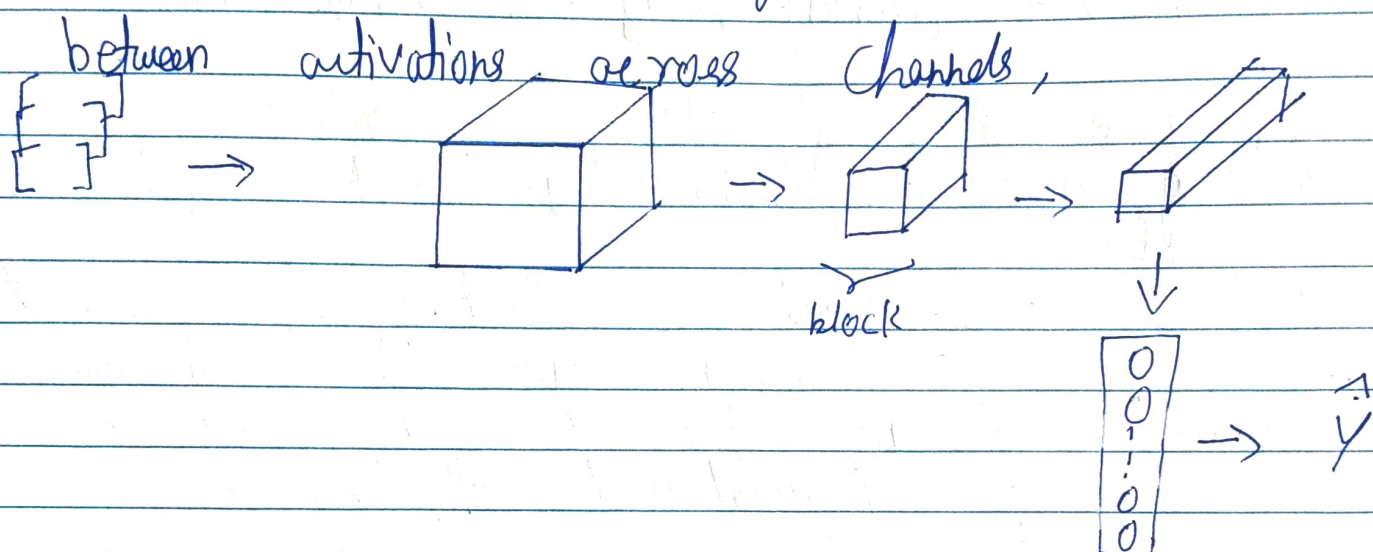
→ If $a^{[l]}(c)$ and $a^{[l]}(G)$ are similar, both images have similar content the cost fn is defined by

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

Style Cost Function

Say you are using Layer 1's activation to measure "style".

We define style as correlation



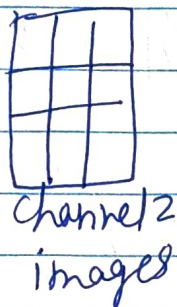
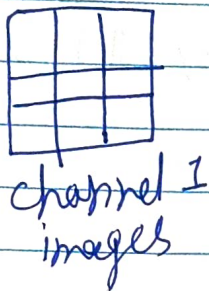
take a block, { we take a few chosen blocks but for example let's take one block here }

so, how correlated are these activations across different channels

so, let's find out how correlated are the first 2 channels.

Intuition about style of an image

So you have let's say 2 channels



So, let's say channel 1 has specific texture,
& channel 2 is orange color,

These images are highly correlated then, whenever you see the color orange there will be that specific texture or that specific texture will have color orange, ~~but if~~
If these images are highly uncorrelated then that specific texture will never have orange color.

So, given an image we will compute style matrix
Let $a_{ijk}^{[l]}$ = activation at (i, j, k)

$$G^{[l]} \text{ is } n_c^{[l]} \times n_c^{[l]}$$

So, $G_{kk'}^{[l]}$ will measure how correlated are the activations in channel k compared to channel k' ,
row = k , column = k'

So,

$$G_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l]} * a_{ijk'}^{[l]}$$

This matrix is called style matrix or Gram matrix

& you do this for both images, S & G

So, finally style cost function

$$J_{style}^{[l]}(S, G) = \frac{1}{F} \| G^{[l]}(S) - G^{[l]}(G) \|^2_F$$

here, $F \rightarrow$ means forbinus norm,

So, we just take the forbinus norm of both the style matrix.

So, the style cost function will look like

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

→ here we are dividing by this particular normalization constant, it won't matter much because it's gonna get multiplied by " β " in the total cost function

→ you will get more visually pleasing ^{results} if you use style cost fn from multiple different layers, not all but some layers

So,

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

Here we also introduce λ parameter hyperparameter just to make it weighted

So, the total cost fn will be,

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

1D & 3D GENERALIZATION

So, you have learned all of that stuff but for only 2d images, but it also works on 1D & 3D data as well

→ In a 2d image you use 2d filters to detect spatial patterns.

→ & In a 1D Time series you use 1D filter to detect pattern that occurs in time or time pattern

ex → EKG machine that records electrical activity of your heart.

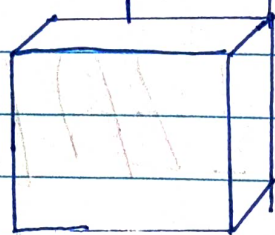
$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 20 & 15 & 3 & 18 & 12 & 4 & 17 & 32 & 9 & 8 & 6 & 5 & 7 \\ \hline \end{array} \quad \begin{array}{c} 14 \times 1 \end{array} \quad * \quad \begin{array}{|c|c|c|c|c|} \hline 1 & 3 & 15 & 5 & 2 \\ \hline \end{array} \quad \begin{array}{c} 5 \times 1 \end{array} = \begin{array}{|c|} \hline \\ \hline \end{array} \quad \begin{array}{c} 10 \times 1 \end{array}$$

& if you have 16 of those 5x1 filters then output will be 10x16.

In 3D Convolution -:

ex → there you can take the example of CT scan, that has height width & depth too,

A 3D Convolution looks for spatial & depth patterns.



3D volume
 $14 \times 14 \times 14 \times 1$



3D
filter.

$5 \times 5 \times 5 \times 1$

"16 filters"



[]

$10 \times 10 \times 10 \times 16$

→ The other example you can treat as a 3D data is a movie or video where each slices of the movie represents time. You can use 3D convolution to detect motion, Action & all stuff, Imao