SOC

CNN

Computer Vision — examples: Image Classification, Object detection, neural Style Transfer

Why CNN? — for large data ... 3 ... computer vision

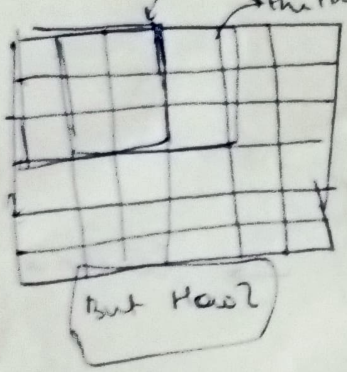
Edge detection

(6×6 ...) * [3×3 ...] convolution = (4×4) ...

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$ ← Filter/Kernel

python - conv-forward
tensorflow - tf.nn.conv2d
keras - conv2D

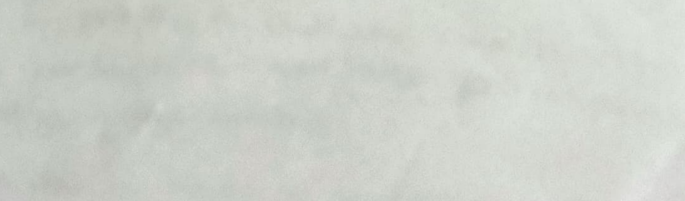
first ... this blue
→ then this blue



Python ⇒ conv-forward
Tensorflow = tf-nn-conv2d
Keras = 2D

e.g. (6×6)

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

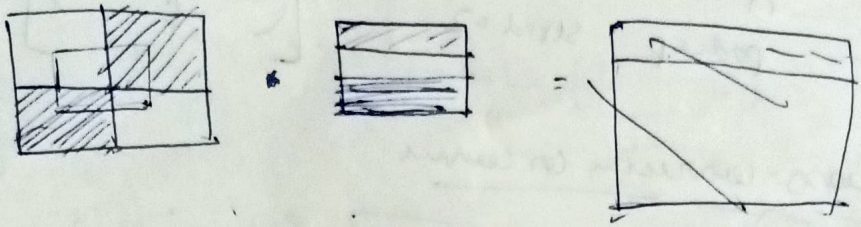


Vertical

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

Horizontal

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



0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Sobel filter ⇒

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

Scharr filter

3	0	3
-10	0	-10
3	0	3

6×6 →

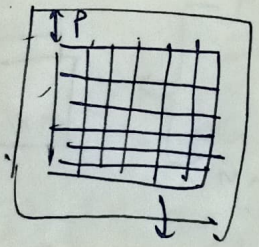
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Padding

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

edges are detected less

$n \times n$ Padding → $(n+2p) \times (n+2p)$



f → usually odd

$n+2p-f+1$

Valid Conv ⇒ no padding $p=0$ i.e. valid

Same Conv ⇒ output size is same as input size

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

Strided Convolution

instead of one step block jump 'n' stride step

$n \times n * f \times f$

padding = p stride = s

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

Cross-correlation (or Convolution)

Conv →

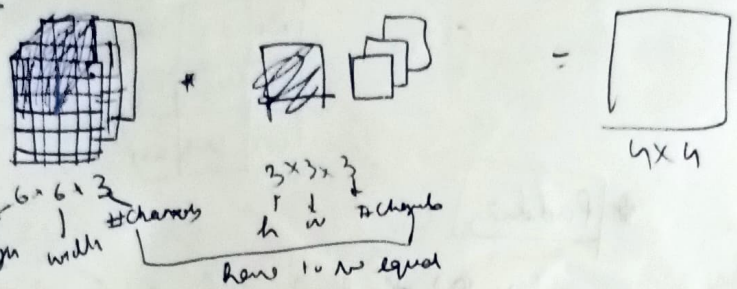
$6 \times 6 *$ [flipped 3×3] ↓?

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

What we call Conv. is a cross-correlation

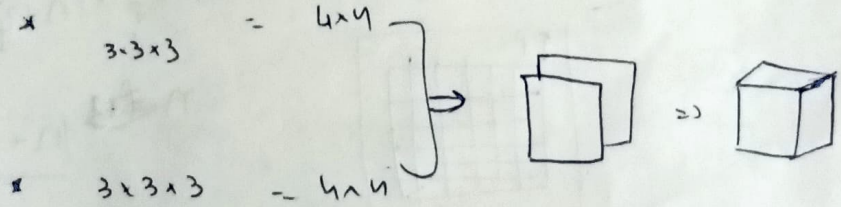
Convolution Over Volume

RGB image



benefit → to filter out only R or B or G like R $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ G $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ B $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

multiple fun



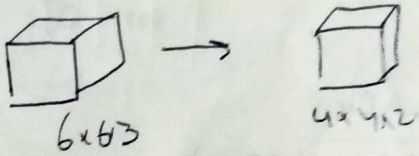
... of Deep

6×6×3 * 3×3×1 → ReLU(4×4 + b1) → 4×4

* 3×3×3 → ReLU(4×4 + b2) → 4×4

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

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



No of parameters in one layer.

10 → 3×3×3 filter in each

no of para = (27+1) × 10

= 280

layer l

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

$p^{[l]}$ = padding

$s^{[l]}$ = stride

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

$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ output : $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

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

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

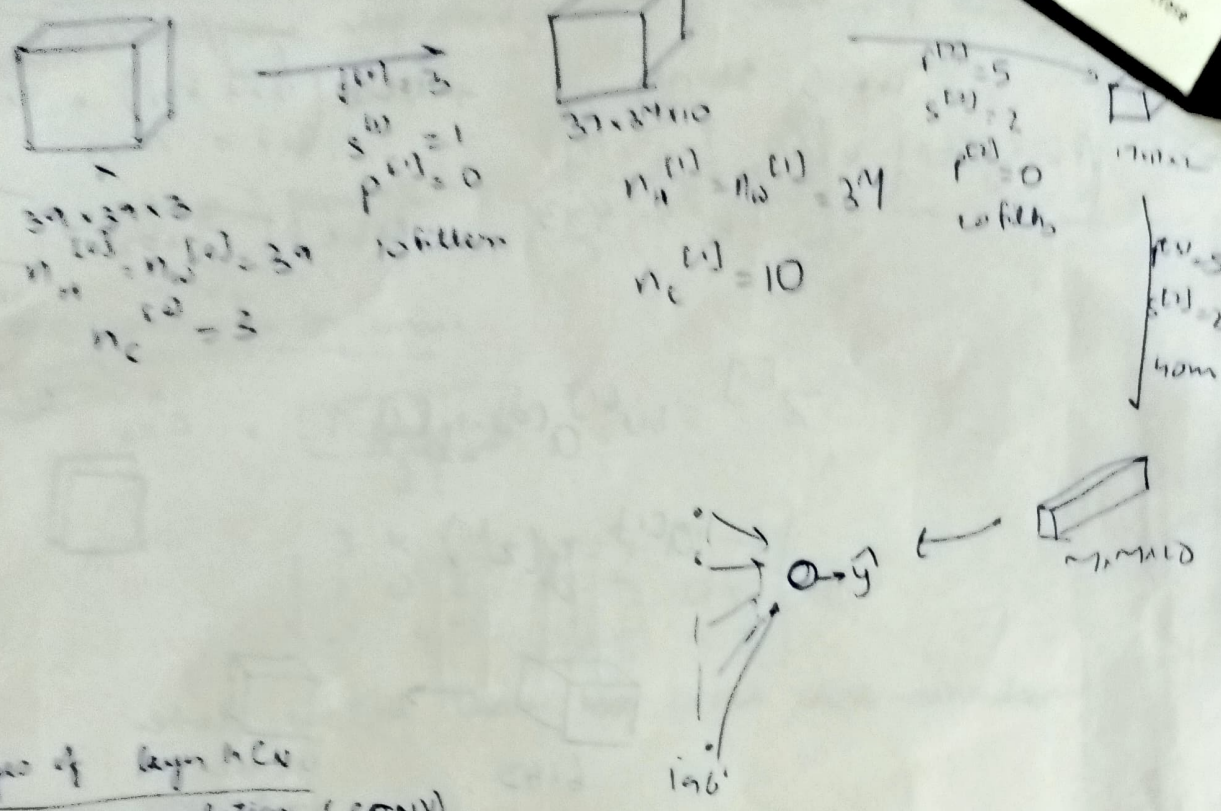
$$n_{H/W}^{[l]} = \left\lfloor \frac{n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

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

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

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

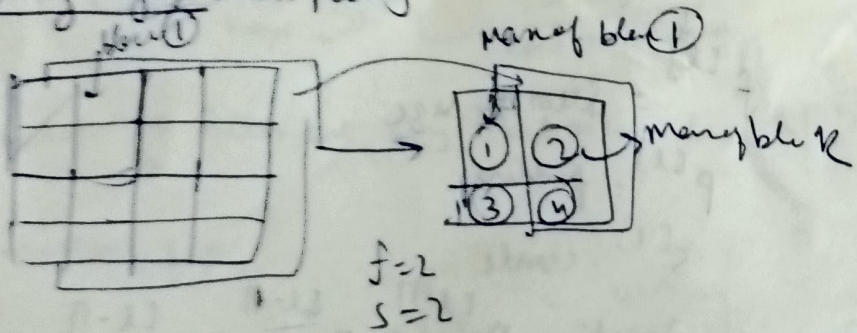
Deep CNN



Types of layer in CN

- → convolution (CONV)
- → Pooling (POOL)
- → Fully connn (FC)

Pooling layer: Max pooling

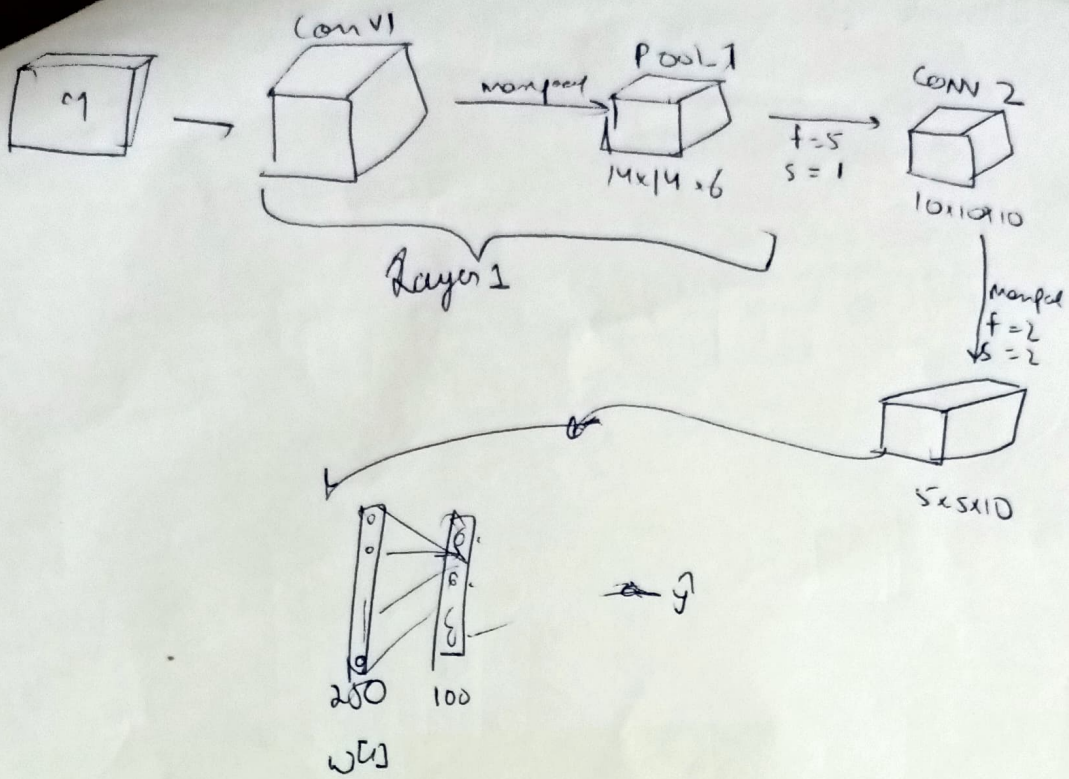


Max pooling done on seperate on each every channel

Average pooling : avg instead of max

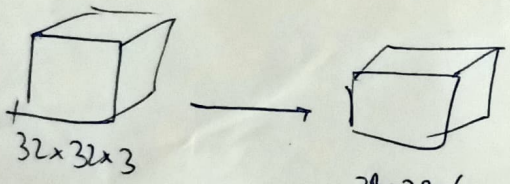
Max pooling is more useful

⇒ No params to learn in pooling.



CONV - POOL - CONV - POOL - FC - FC ---- Softmax

Why Convolution

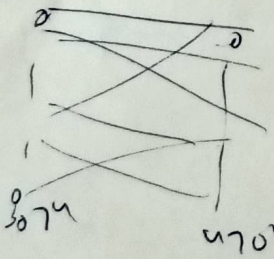


params

$(5*5+1)*6$

$= 156$ params

param sharing



Sparsity of connections

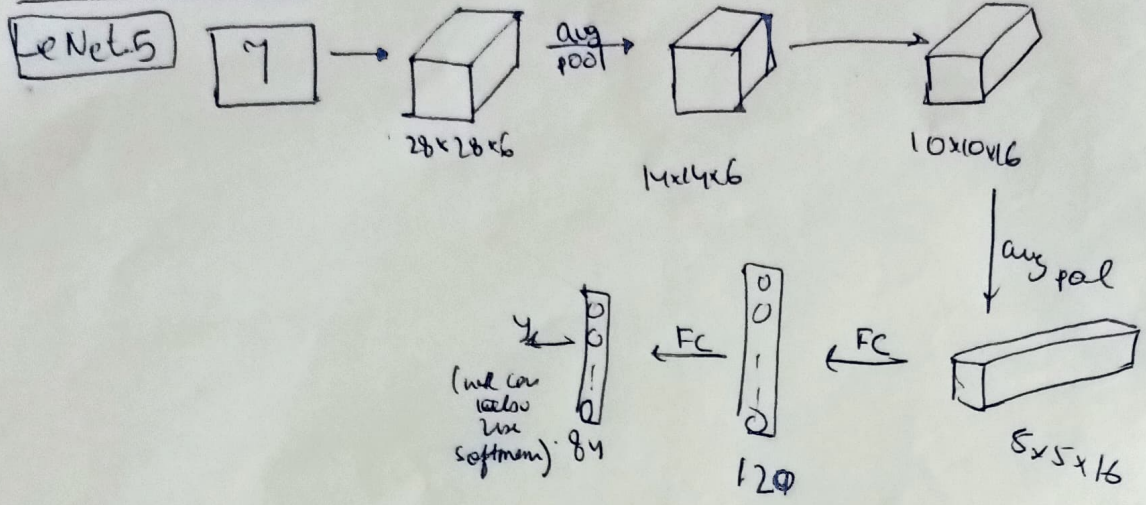
each output depends on a small # of inputs

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

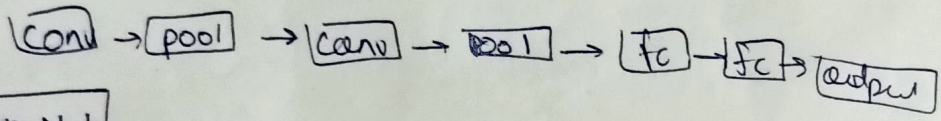
Classical networks → LeNet-5
→ AlexNet
↘ VGG

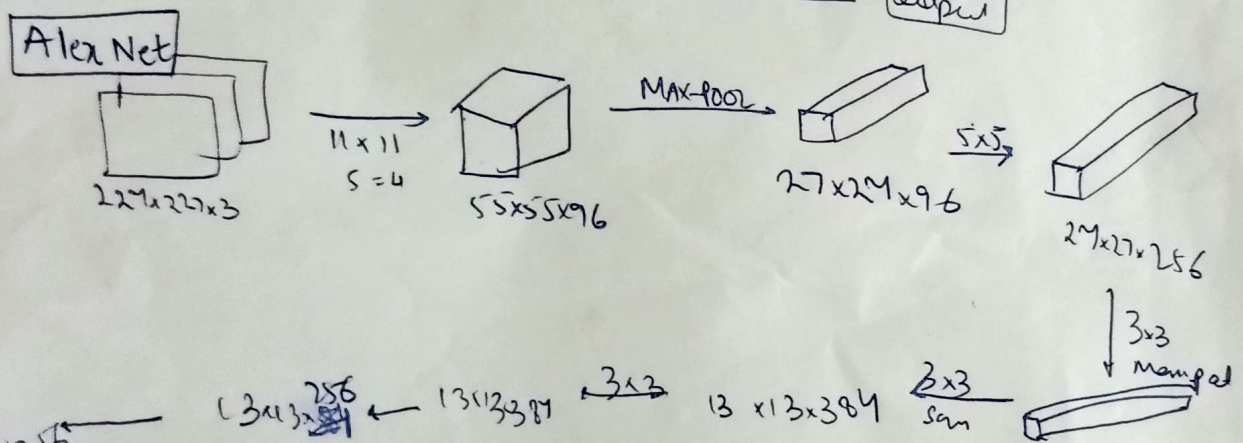
ResNet
Inception

Classical Network



or



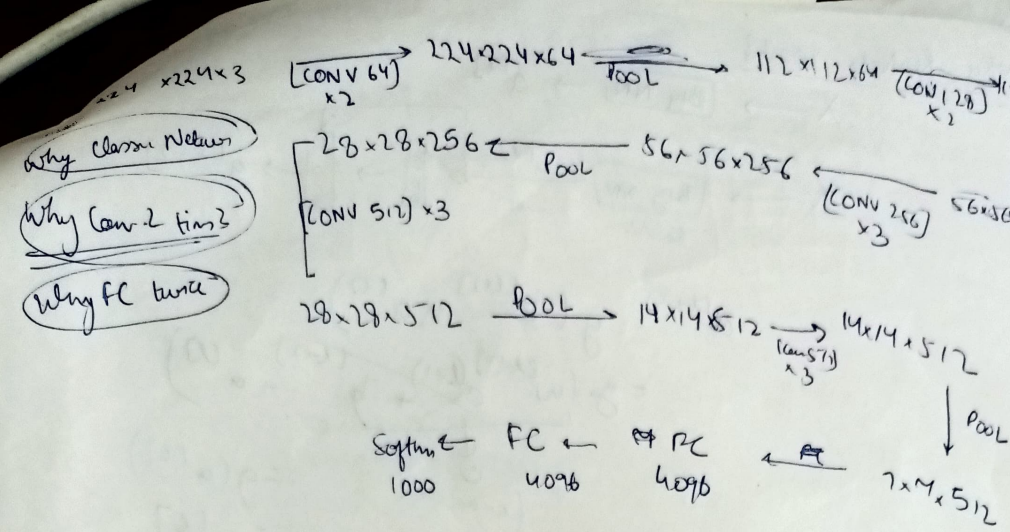


→ Similar to LeNet, but much big

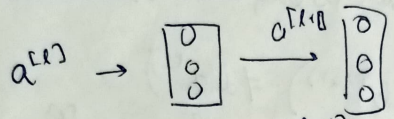
→ ReLU

Why? ... this → Multiple GPU

→ Local Response Normalization (LRN)



ResNet (Residual Network)



$a^{[l]} \rightarrow$ Linear $\rightarrow$ ReLU $\xrightarrow{a^{[l+1]}}$ Linear $\rightarrow$ ReLU

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

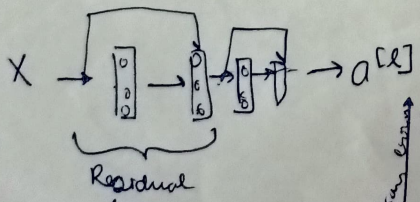
$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$, $a^{[l+2]} = g(z^{[l+2]})$

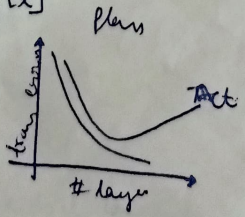
$a^{[l]} \rightarrow$ ReLU $\rightarrow a^{[l+2]}$ (~~direct~~ shortcut / skip connection)

Here

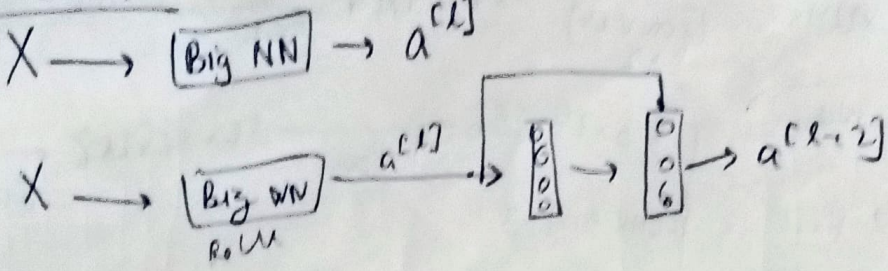
$a^{[l]} \xrightarrow{\text{shortcut}} \text{ReLU} \rightarrow a^{[l+2]}$

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





Why ResNets work?



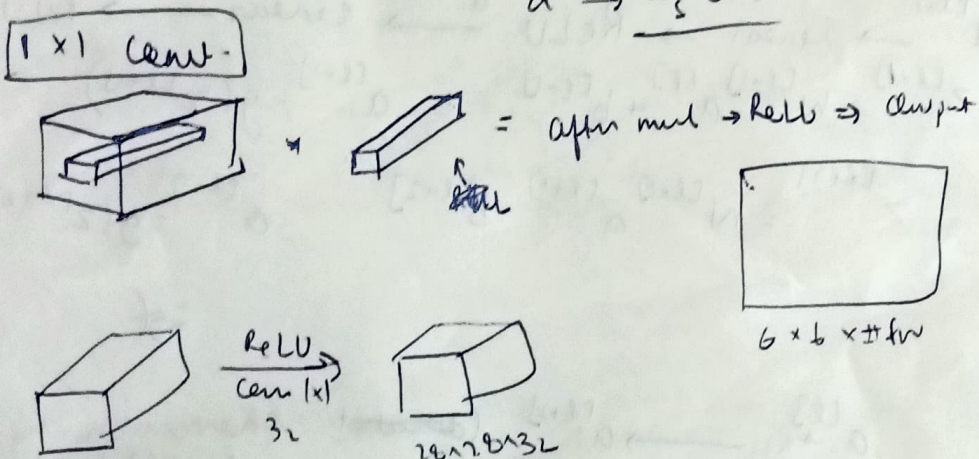
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$
$$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

(w ≈ 0, b = 0)

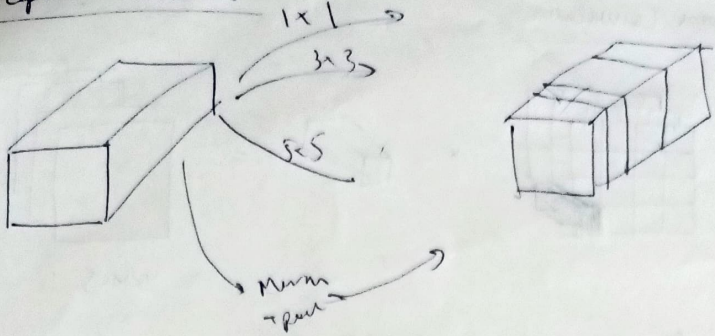
$$= g(a^{[l]}) = a^{[l]}$$

⇒ so identity is easy to learn.

$$a^{[l+2]} \rightarrow w_s a^{[l]}$$



Inception Network

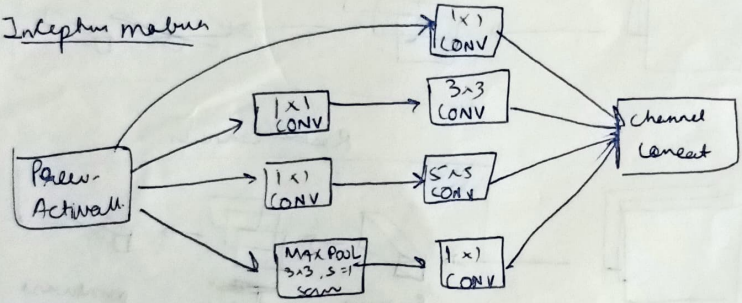


Concatenation of
→ several filters

Using 1×1 conv

$28 \times 28 \times 192$ → CONV 1×1, 16, $1 \times 1 \times 192$ → $28 \times 28 \times 16$ → CONV 5×5, 32, $5 \times 5 \times 16$ → $28 \times 28 \times 32$

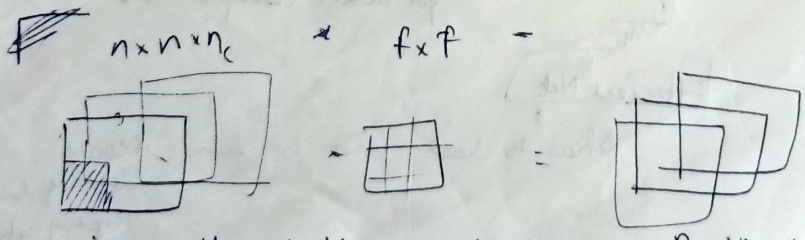
Inception module



MobileNet

→ low compute cost at deployment

→ Depthwise separable convolution



instead of block by block do it layer by layer

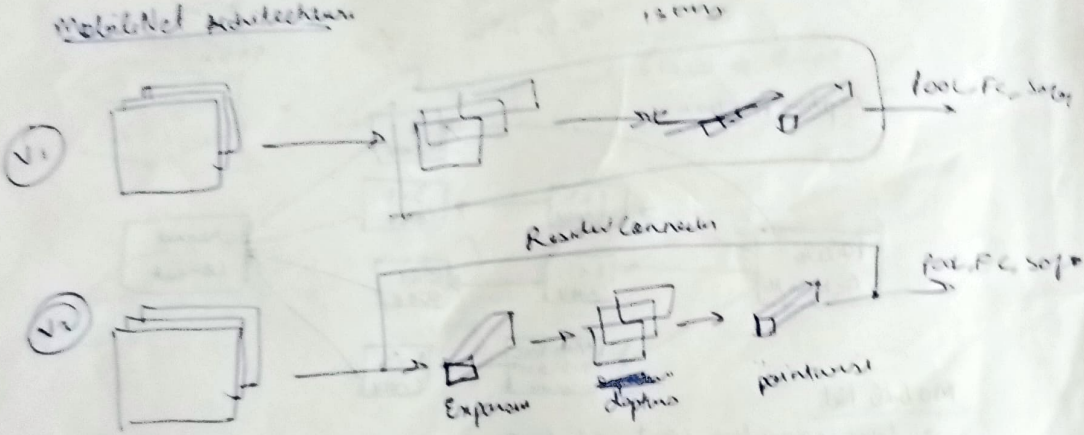
$n_{out} \times n_{out} \times n_c$

Depthwise Convolution

Cost of depthwise separable conv

$$\frac{\text{Cost}_{DSC}}{\text{Cost}_{Norm}} = \frac{1}{N_c'^2} + \frac{1}{f^2}$$

MobileNet Architecture



why → increasing the size of bottleneck blocks for better computation.

Efficient Net

How to choose what to scale research intern of by dept of nature

YOLO algorithm

so we say obj. is 1 if $p_c = 1$, if $p_c = 0$ then other values are just ignored values.

→ For each grid cell, get 2 predicted bounding boxes.

→ get rid of low probability predictions

→ For each class, use non-max to generate final predictions.

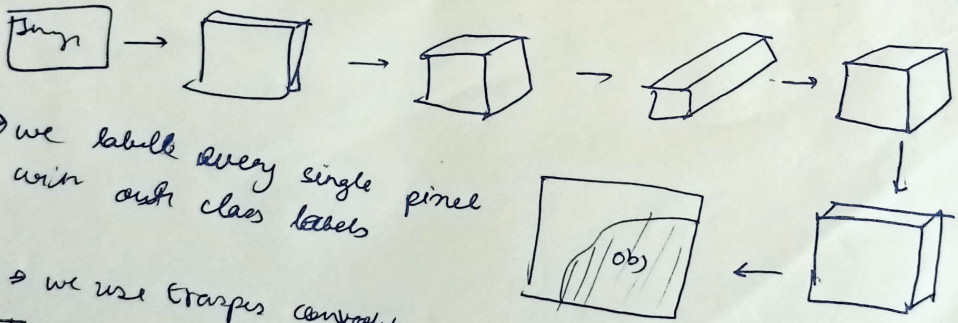
R-CNN

→ We run CNN on selected rectangles using segmentation algorithm. (find useful boxes & implement on every)

Find blobs using segmentation algorithm & the crossfire.

Fast R-CNN ⇒ convolution implementation of sliding windows

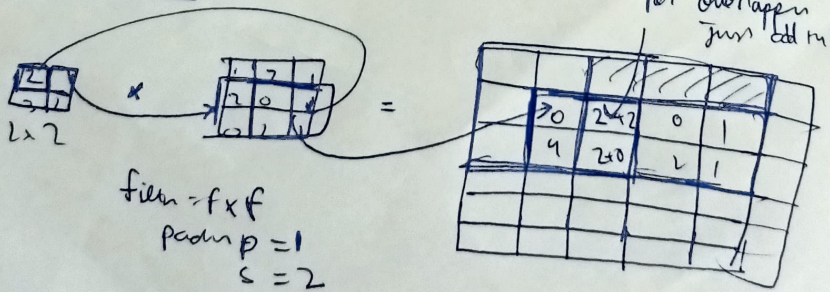
Semantic Segmentation ⇒ figures out what's there in every single pixel.



⇒ we labelled every single pixel with each class labels

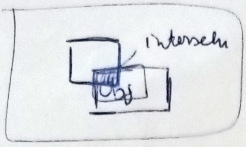
⇒ we use transpose convolution.

Transpose Convolution



Intersection Vs union

e.g.



⇒ $\frac{\text{size of intersection}}{\text{size of union}}$

"correct" if $IoU \geq 0.5$ ← measure of overlap b/w two bounding boxes.

Non-max suppression

↓

one detection per object, boxes / obj.

Each output pred : $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

discard all boxes with $p_c \leq 0.6$

⇒ pick out the box with largest p_c for pred

→ discard any remaining box with $IoU \geq 0.5$

Anchor boxes

⇒ If there are more than one object in a grid

we use anchor boxes

we change y to → $y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ \vdots \\ c_3 \end{bmatrix}$ anchor box 1, anchor box 2

Each object in training is assigned grid cell that contains object's midpoint & anchor box for grid cell with highest IoU with object's shape.

If the shape of anchor box is similar to the shape of object we give it $p_c = 1$