Outline

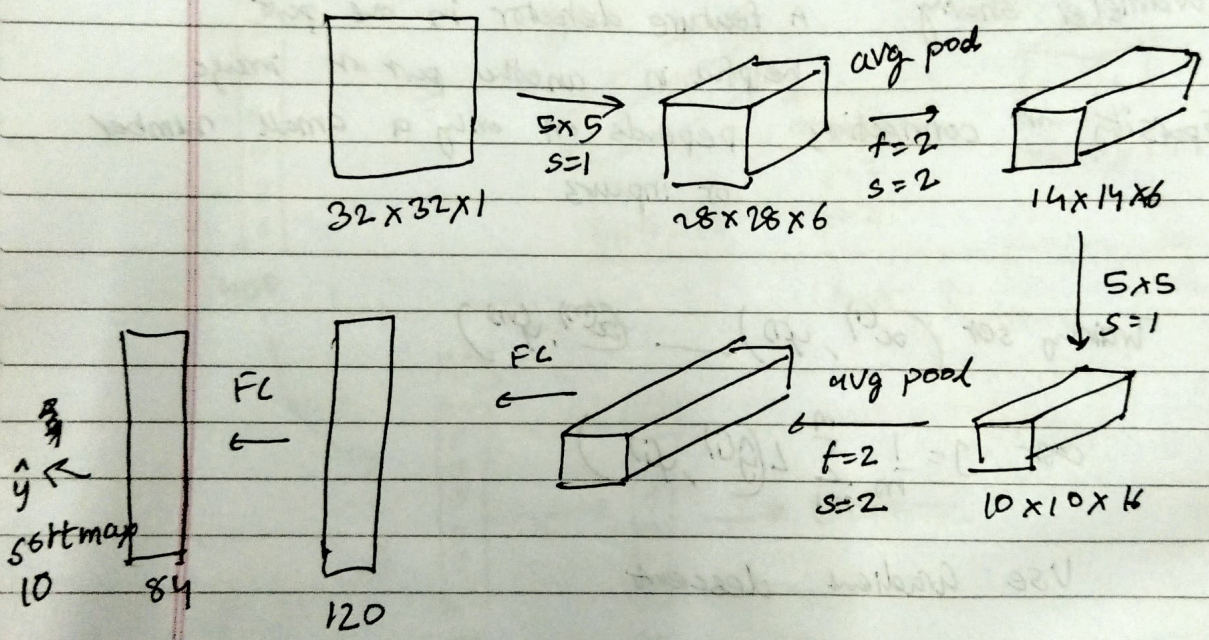
Classic networks
- LeNet-5
- AlexNet
- VGG

ResNet (152)

Inception

CLASSIC NETWORKS

LeNet-5

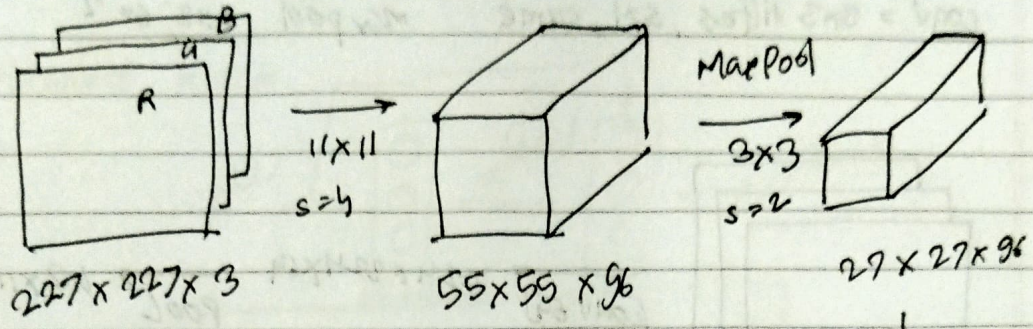


$n_H, n_W \downarrow \quad n_C \uparrow$

conv pool conv pool fc fc output

$n_H \times n_W \times n_C \quad f \times f \times n_C$

☆ AlexNet



60 M parameters

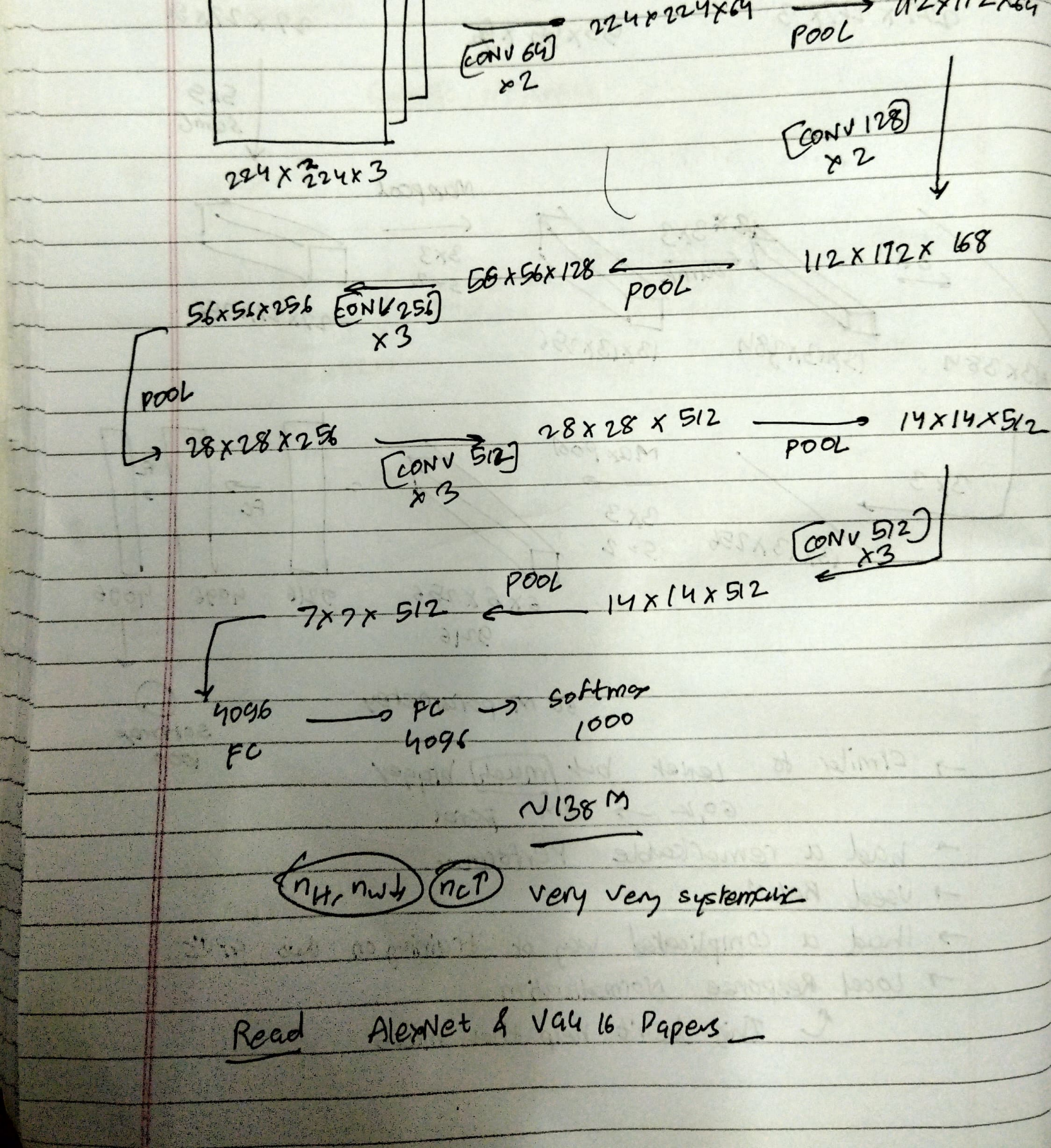
→ Similar to LeNet but much bigger
60,K → 60M para

→ had a remarkable Performance

→ used ReLU

→ had a complicated way of training on two GPU's

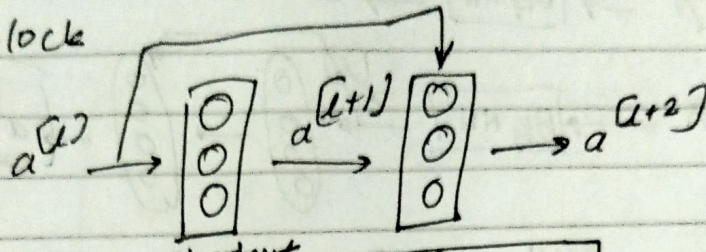
→ Local Response Normalization
↳ This doesn't help that much


224×224×3
CONV 64 ×2
224×224×64
POOL
CONV 128 ×2
112×112×64
112×112×128
POOL
56×56×128
CONV 256 ×3
56×56×256
POOL
28×28×256
CONV 512 ×3
28×28×512
POOL
14×14×512
CONV 512 ×3
14×14×512
POOL
7×7×512
4096 FC
FC 4096
Softmax 1000
~138 M
$n_H, n_W \downarrow$
$n_C \uparrow$
very very systematic
Read AlexNet & VGG 16 Papers

Residual Networks

skip connection

Residual block



$a^{[l]}$ → linear → ReLU → $a^{[l+1]}$ → linear → (+) → ReLU → $a^{[l+2]}$

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

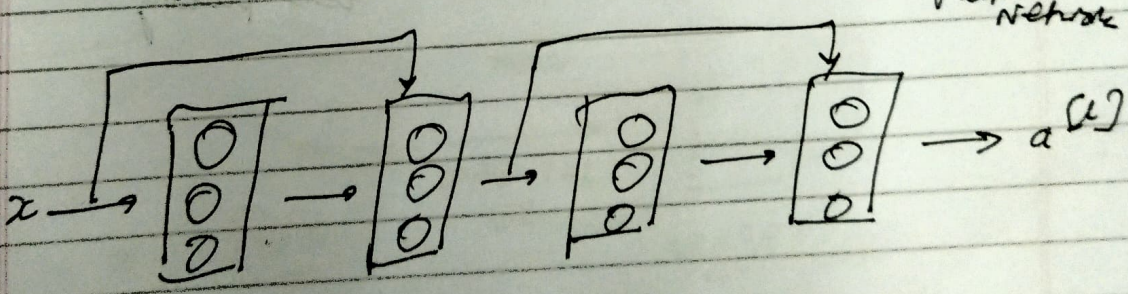
$$a^{l+1} = g(z^{[l+1]})$$

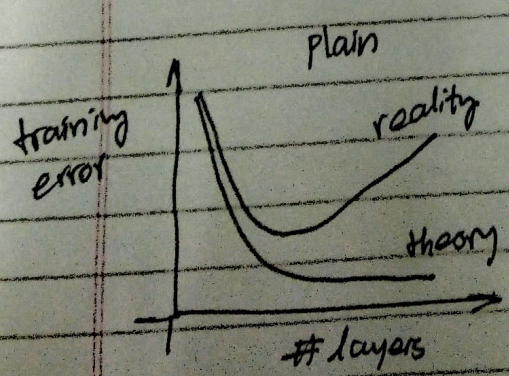
$$a^{[l+2]} = g(z^{[l+2]})$$

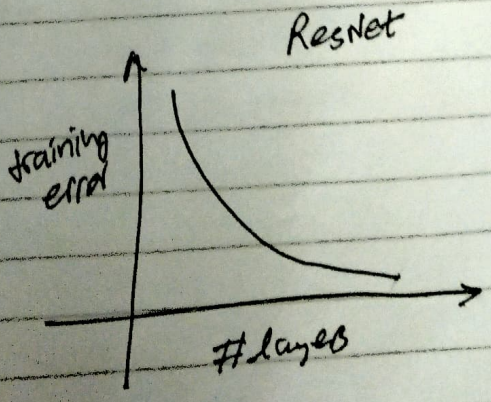
← sim for $z^{[l+2]}$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$ ← directly adding before using the ReLU non-linearity

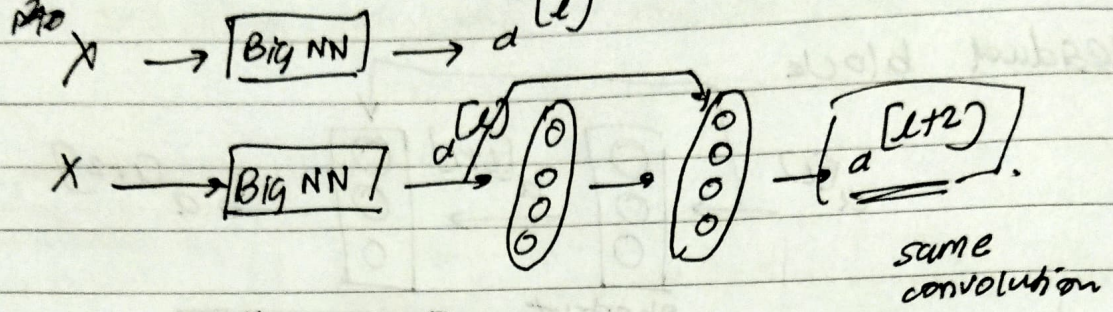
This shortcut → skip connection







Why ResNets work so well?



ReLU $a \geq 0$

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

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

If $w^{[l+2]} = 0$, $b = 0$

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

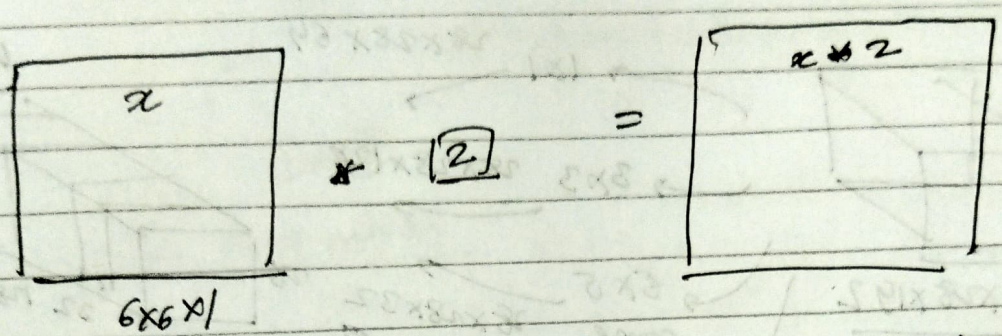
256 → 128

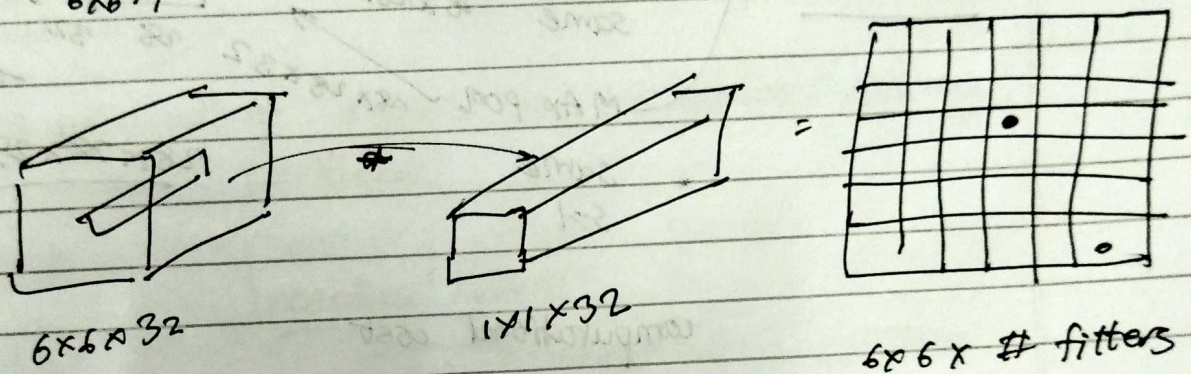
W_s → matrix or parameters

or → that just be a matrix with zero padding

1×1 convolutions in NN

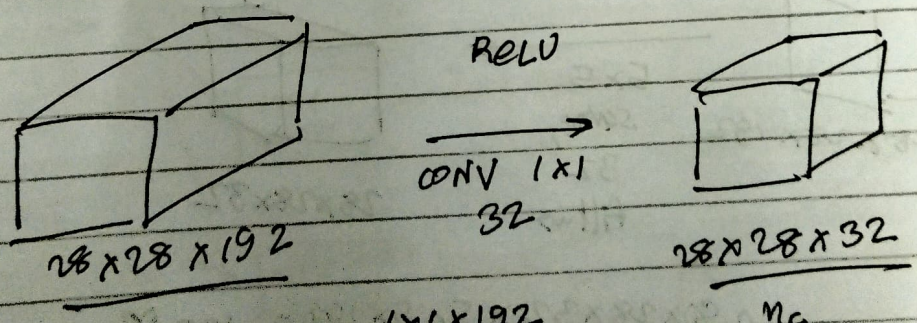
What does 1×N convolutions do?





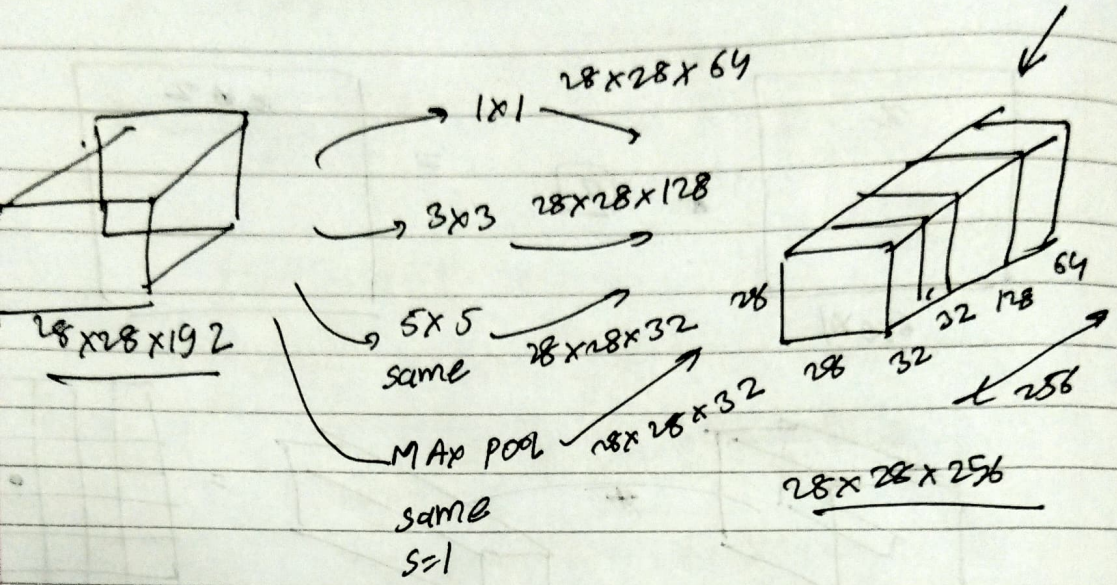
32 → # filters

$n_c^{[l+1]}$



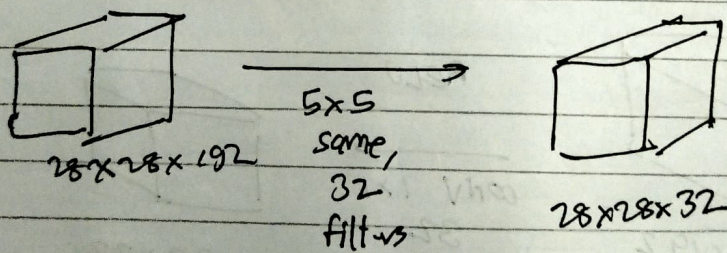
INCEPTION NETWORK

Motivation for inception network



computational cost

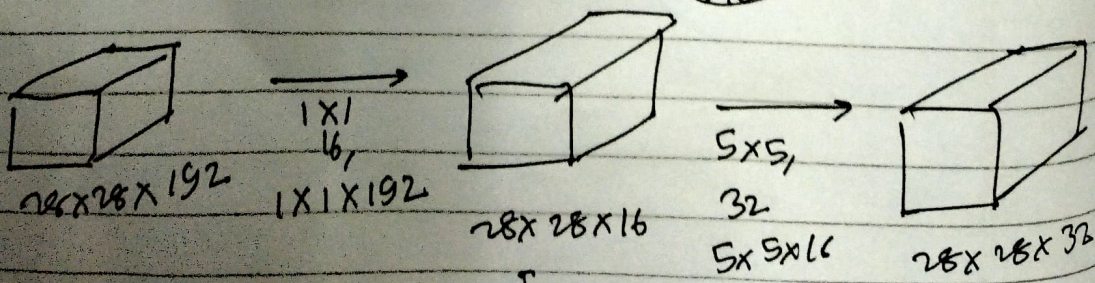
5×5 part



$28 \times 28 \times 32 \times 5 \times 5 \times 192 = 120\,M$

Using 1×1 conv

$\downarrow$ $\frac{1}{10}^{th}$



$28 \times 28 \times 16 \times 192 = 2.4 \text{ million}$ — ①

2nd

$28 \times 28 \times 32 \times 55 \times 5 \times 16 = 10.0 \text{ million}$ — ②

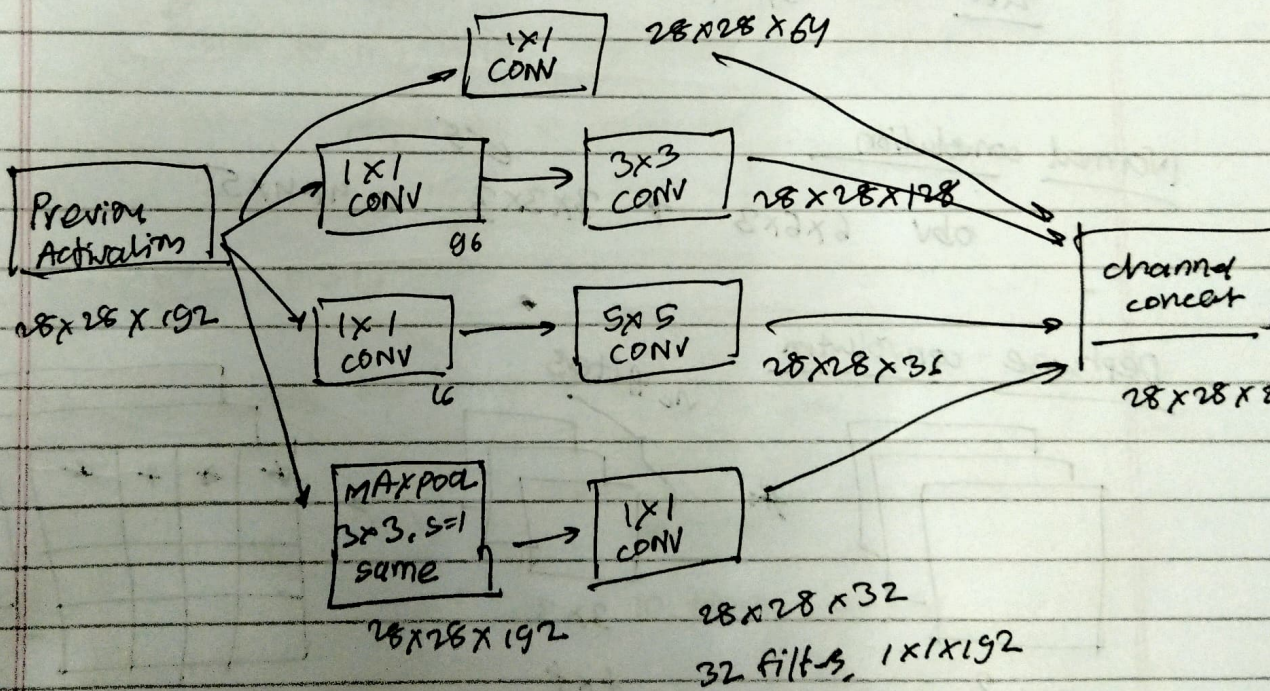
① + ②

12.4 million

120 million

does not affect much the performance / accuracy (Bottleneck)

Inception Network



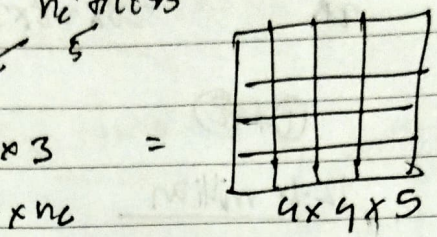
Puts a lots of these module together

Read Inception Paper

MobileNet

for computing in low computing cost

→ useful for mobile & embedded system application

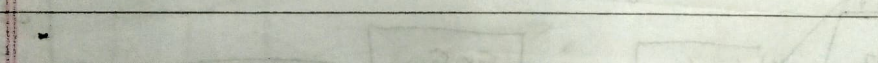


~~# filter params~~ ~~# of filters~~

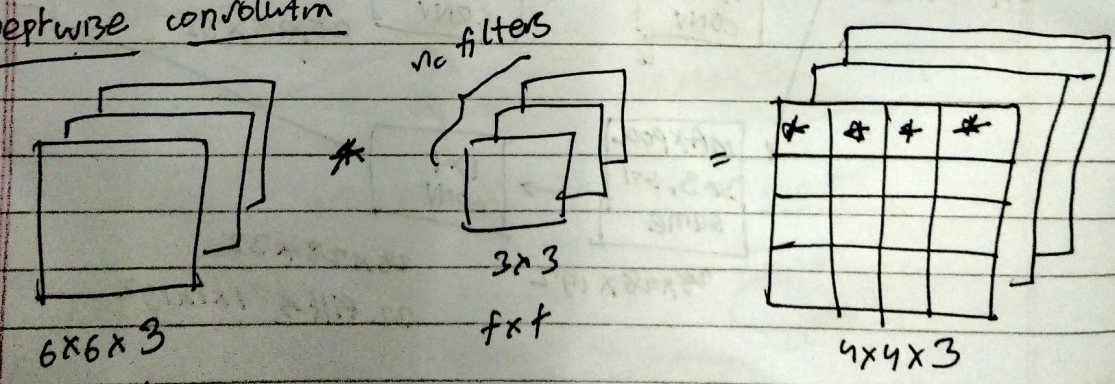
computational cost = # filter params × # filter positions × # of filters

2160 = 3×3×3 × 4×4 × 5

normal convolution



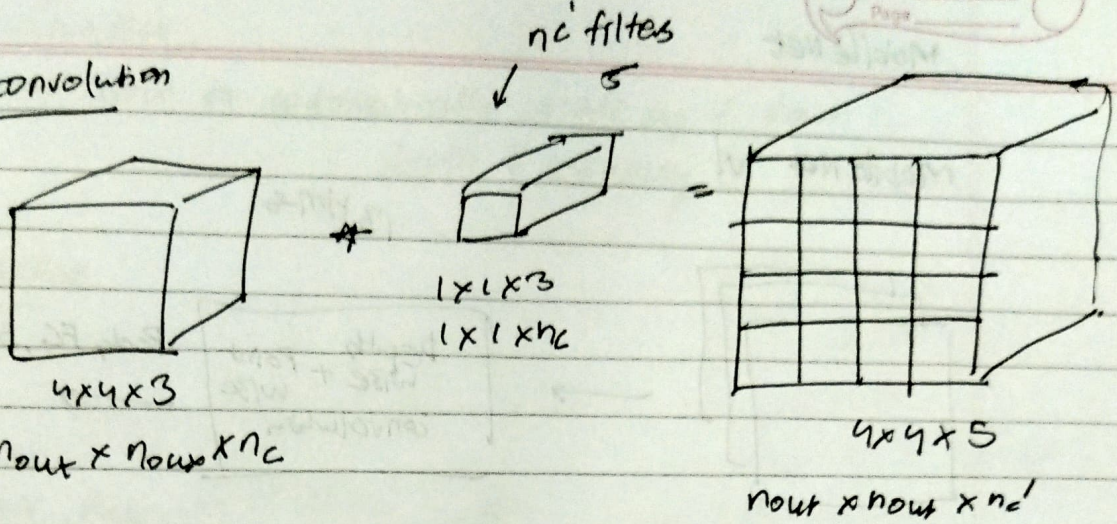
depthwise convolution



computational cost = # filte params × # filter positions × # filtes

432 = 3×3 × ~~3×3~~ 4×4 × 3

Pointwise convolution



cc = # filterparams x # filter pos

240 = 1x1x3 x 4x4 x 5

cost of Normal convolution (2160)

cost of depthwise separable convolution

depthwise + pointwise

432 + 240 = (672)

$\frac{672}{2160} = 0.31$

↳ this comes out to be = $\frac{1}{nc'} + \frac{1}{f^2}$

$= \frac{1}{5} + \frac{1}{9} = 0.31$

$\frac{1}{512} + \frac{1}{3^2} \sim$ 10 times cheaper

⇒ Depthwise (dimensions)

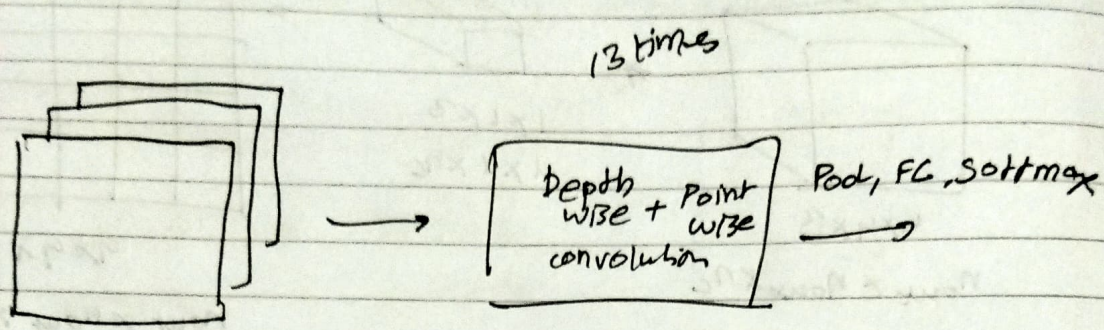
↳ 6x6xnc * 3x3xnc = 4x4xnc

⇒ Pointwise

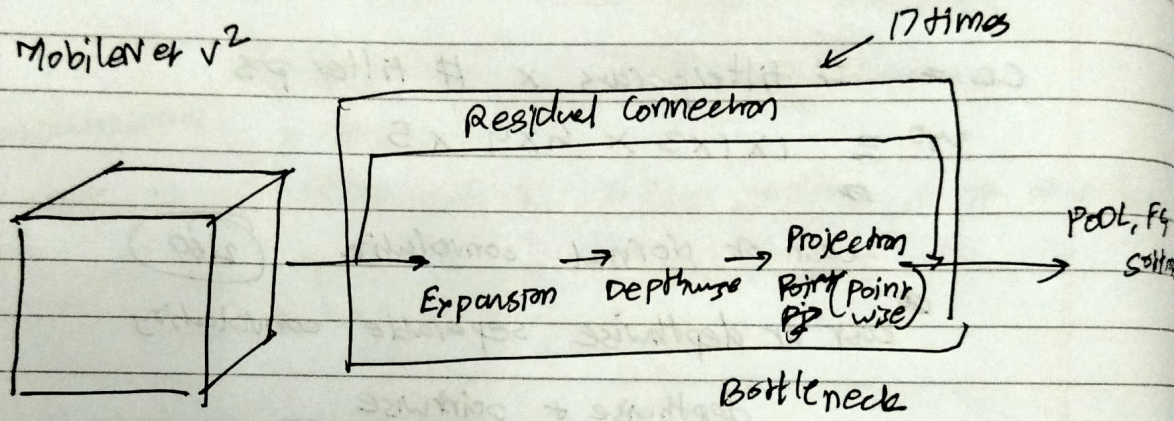
↳ 4x4xnc * 1x1xnc = 4x4xnc'

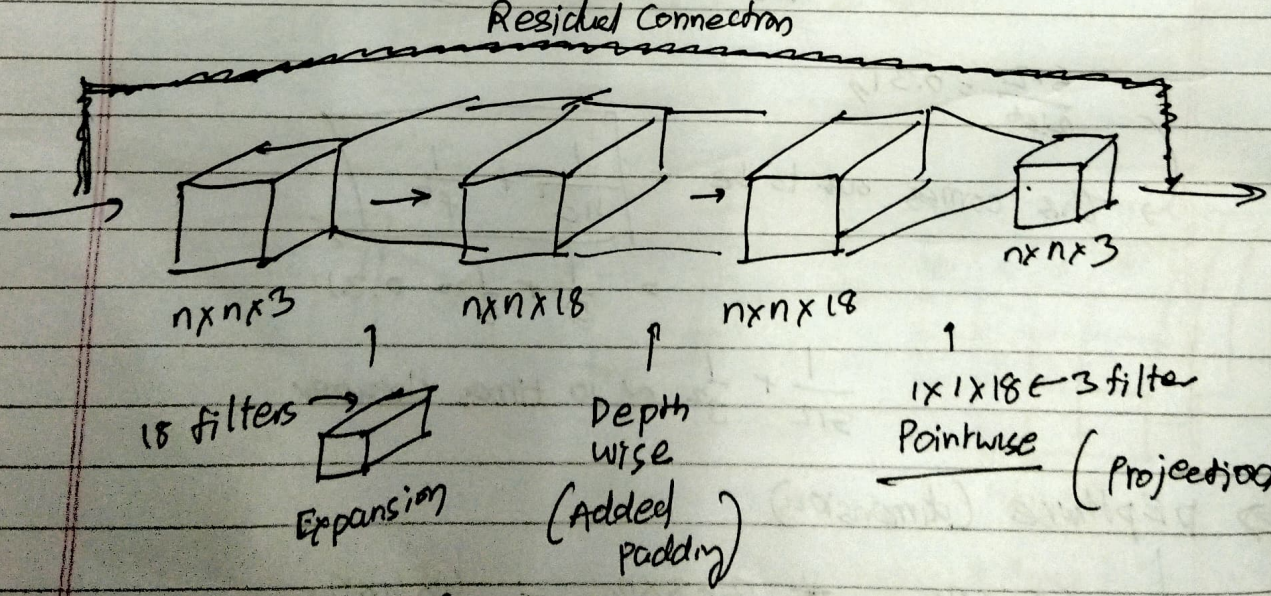
MobileNet

MobileNet v1



MobileNet v2





Why Bottleneck?

→ Between two uses heavy computing
2 to 3 2 to 3

→ Keeping the amount of memory and computing as small as possible

EfficientNet

→ How to automatically scale up & down depth & accuracy?

Baseline

→ r resolution
→ d depth
→ w wider

Higher Resolution / Deeper / wider

whats good choice according to the resources we have?

what to increase & how much?

2) How to adapt for a particular device?
look at one of the opensource implementation of efficientNet

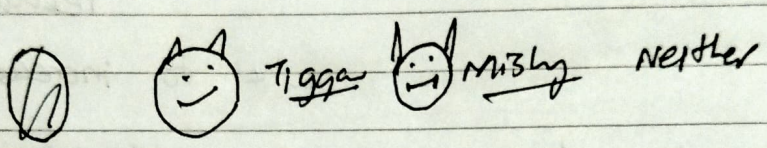
Practical Advices

- Using open-soure implementations

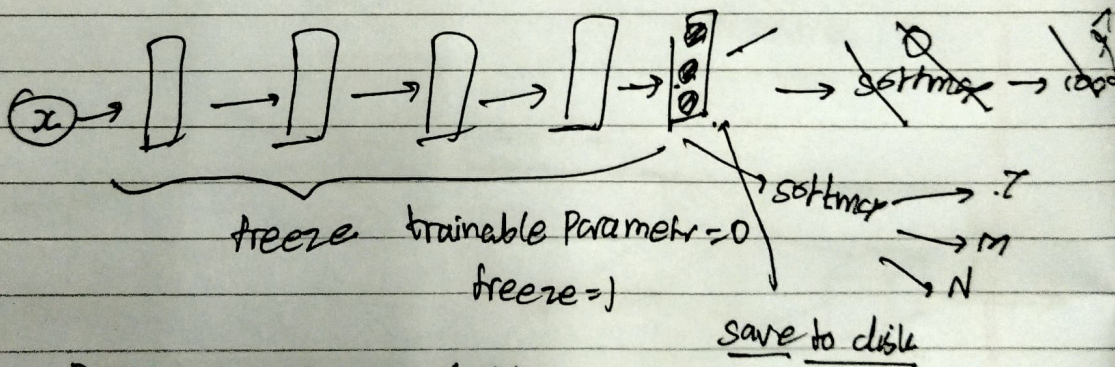
Github

git clone url
to download on local harddisk

Tranfer learning



Download open souree implementation
↳ code + weights



Precompute till last layer
maps its to last layer using your training set

If you have larger dataset than
freeze only some layers (first few)
and use last few layers to train

for lots of data
you can train the whole unit data
so just use weights as initialisation

Data Augumentation

Some common techniques

- Mirror
- cropping (Beaware)
- Roration
- Shearing
- local warping

} more complicated ps don't suggest

Color shifting

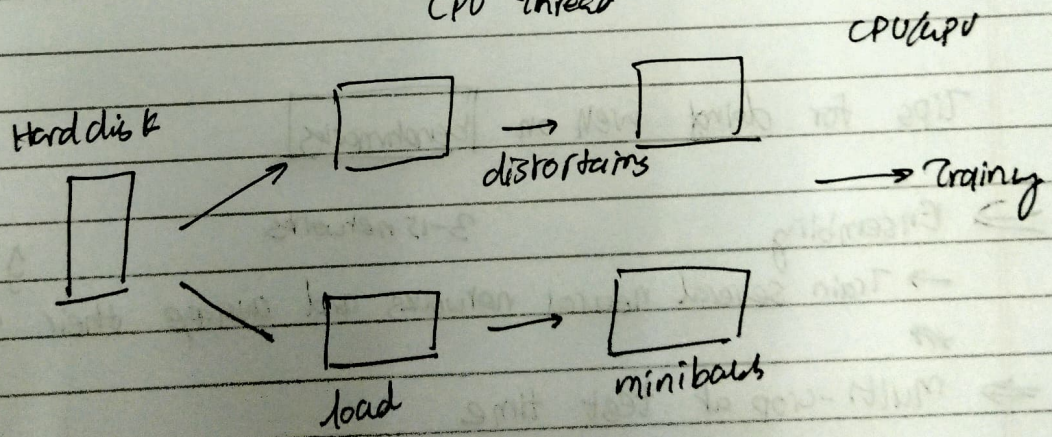
→ make training data robust

→ By making slight changes in the colors of images

R G B

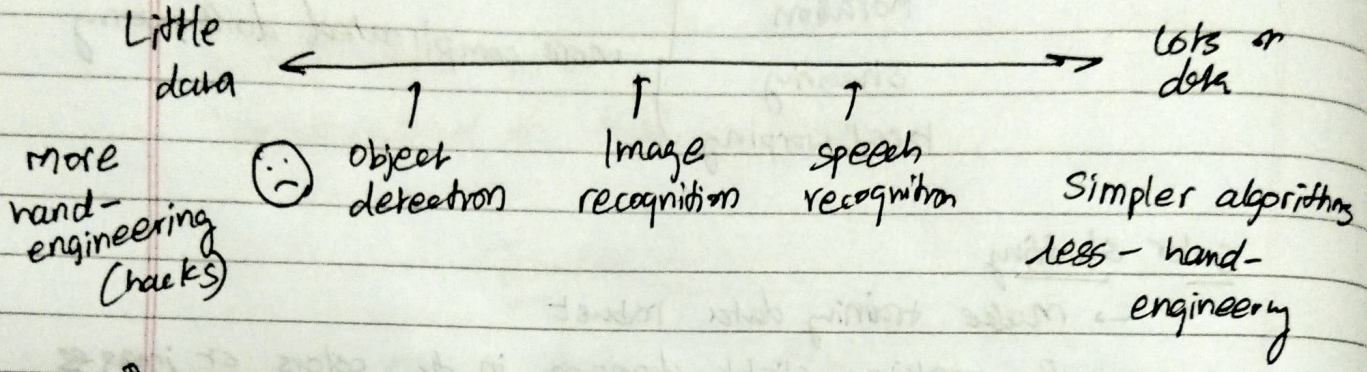
Distort this channel a little bit and model should work good even then

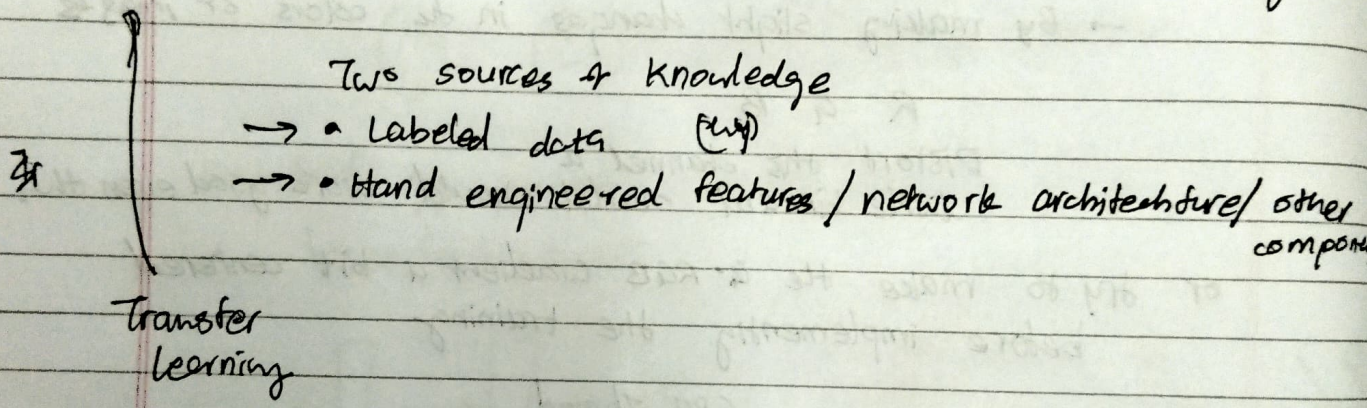
or try to make the R-G-B gradient a bit centered before implementing the training



State of computer vision

Data vs hand engineering





Tips for doing well on benchmarks

- Ensembling — 3-15 networks
  - Train several neural networks and average their outputs
- Multi-crop at test time
  - Run classifier on multiple versions of test images and average outputs