Outline:

Classic Networks:

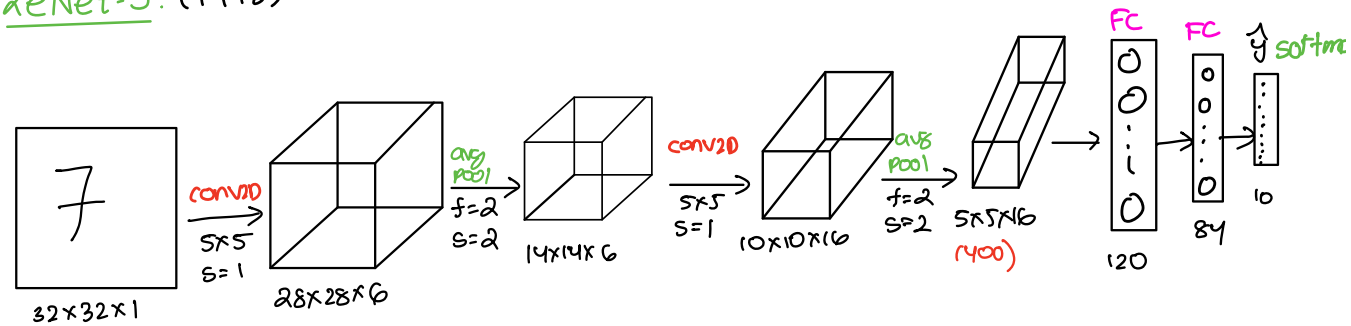
- LeNet-5
- AlexNet
- VGG

ResNet (152 layers)

Inception.

Dec2: Classical Networks:

LeNet-5: (1998)



- parameters ≈ 60k parameters
- $n_H, n_W \downarrow$, $n_C \uparrow$
- conv, pool, conv, pool fc fc output.

Advanced: used sigmoid / tanh function → use ReLU

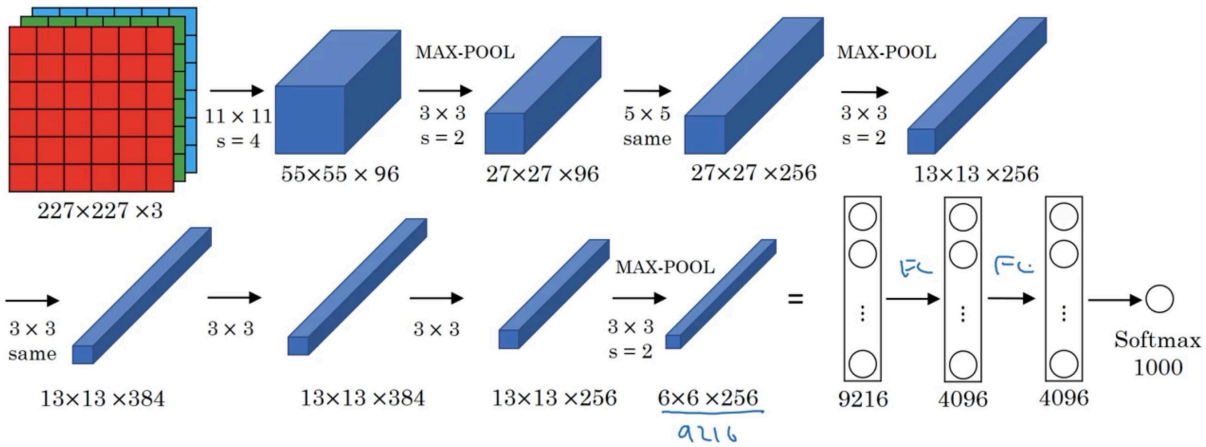
→ filter channel size is different than input channel size.

(n_H, n_W, n_C) (f, f, n_C)

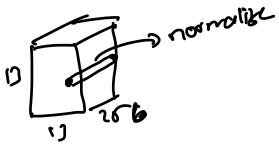
different (due to computation saving)

- use non linearity after pooling (sigmoid).

AlexNet: (2012)



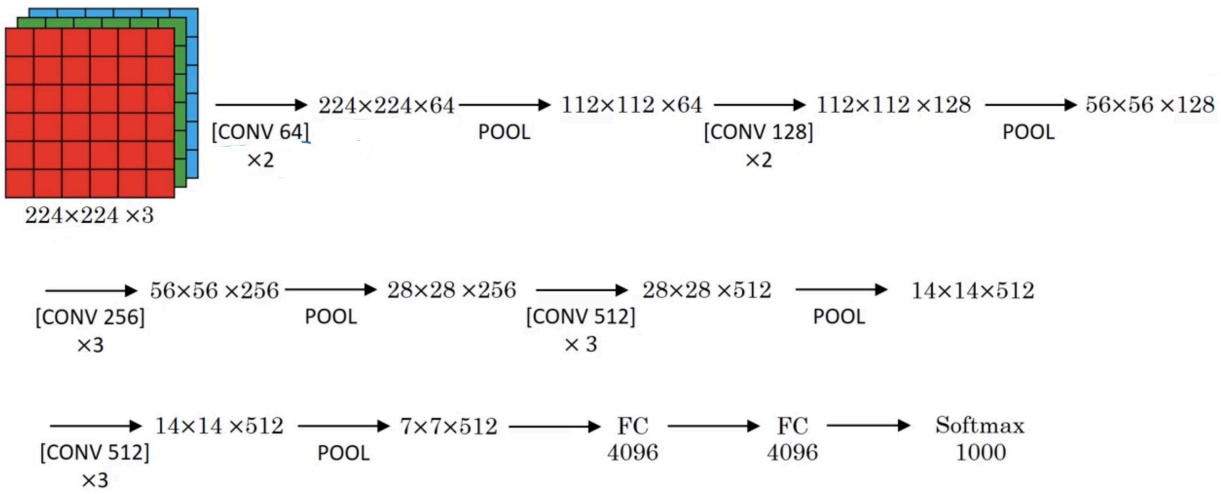
- Much bigger than LeNet, has 60M parameters.
- use ReLU
- Multiple GPU's training
- (less important) • local response Normalization



VGG-16 (2015) (VGG-19)

CONV = 3x3 filter, s=1, same

MAX-POOL = 2x2, s=2



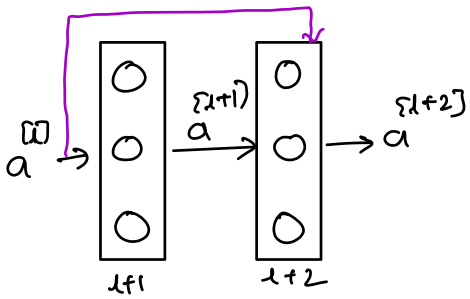
→ 16 layers which have weights.

→ simple hparameters to tune

→ filters (64, 128, 256, 512, 512)

→ large network to train. (138 M parameters)

dec3. Residual Block:



residual is added before activation

'shortcut / skip connection'

$a^{[l]} \rightarrow$ linear $\rightarrow z^{[l+1]} \rightarrow$ ReLU $\rightarrow a^{[l+1]} \rightarrow$ linear $\rightarrow z^{[l+2]} \rightarrow$ ReLU $\rightarrow a^{[l+2]}$

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

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

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

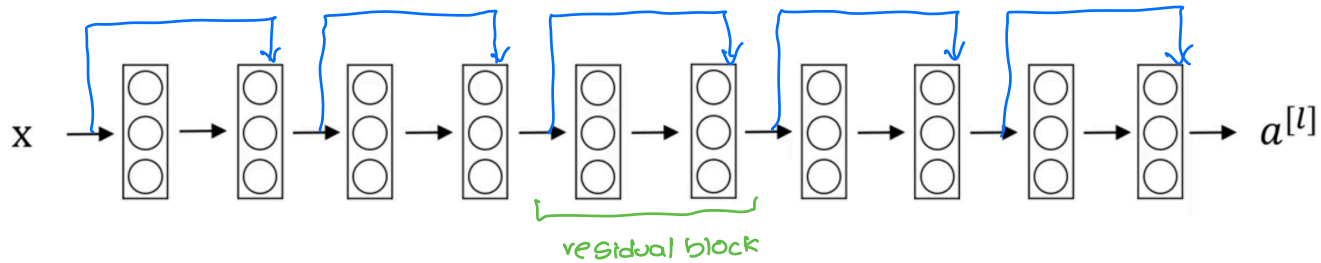
$$a^{[l+2]} = g(z^{[l+2]}) \quad \times$$

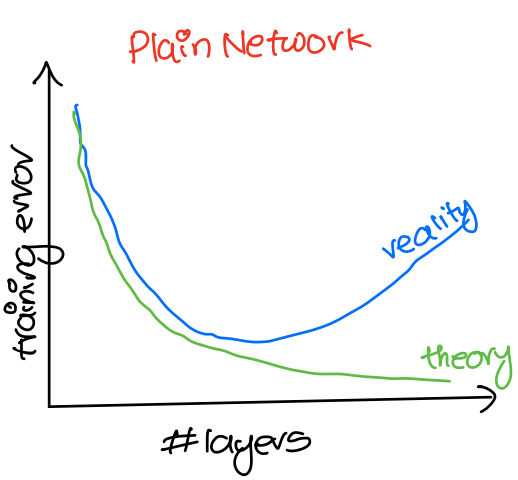
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

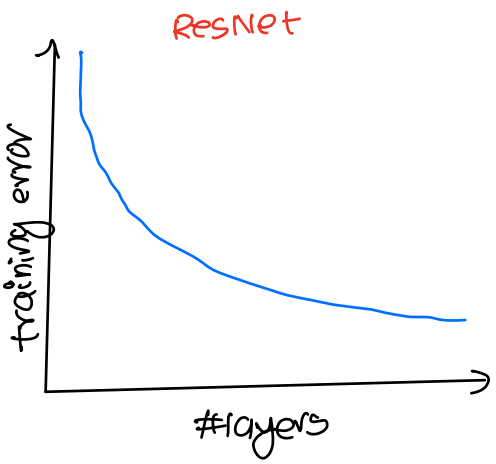
Advantages:

- using residual blocks allows you train much bigger deep learning networks.

Residual Network:

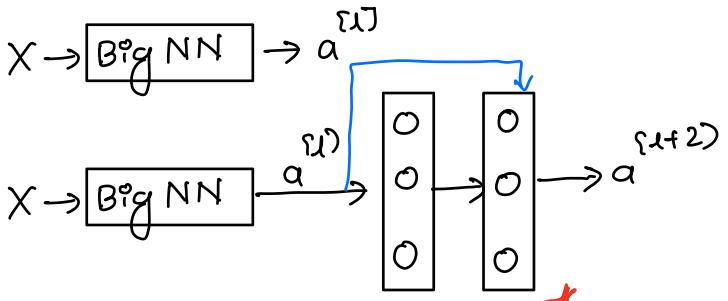






→ helps in vanishing/exploding gradient problem and train much deeper networks.

Lec 4: Why ResNets Work:



assume ReLU activation for all layers.
$\therefore a \geq 0$

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

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

assume if $W^{[l+2]} = 0$ $b^{[l+2]} = 0$

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

$$\therefore a^{[l+2]} = a^{[l]} \quad [\because a \geq 0]$$

$[\dim z^{[l+2]} = a^{[l]}]$

if $\dim z^{[l+2]} \neq a^{[l]}$

$$a^{[l+2]}_{256} = g(z^{[l+2]} + a^{[l]}_{128})$$

add W_s (256, 128)

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

W_s can be learned

W_s can be fixed, and zero pad $a^{[l]}$

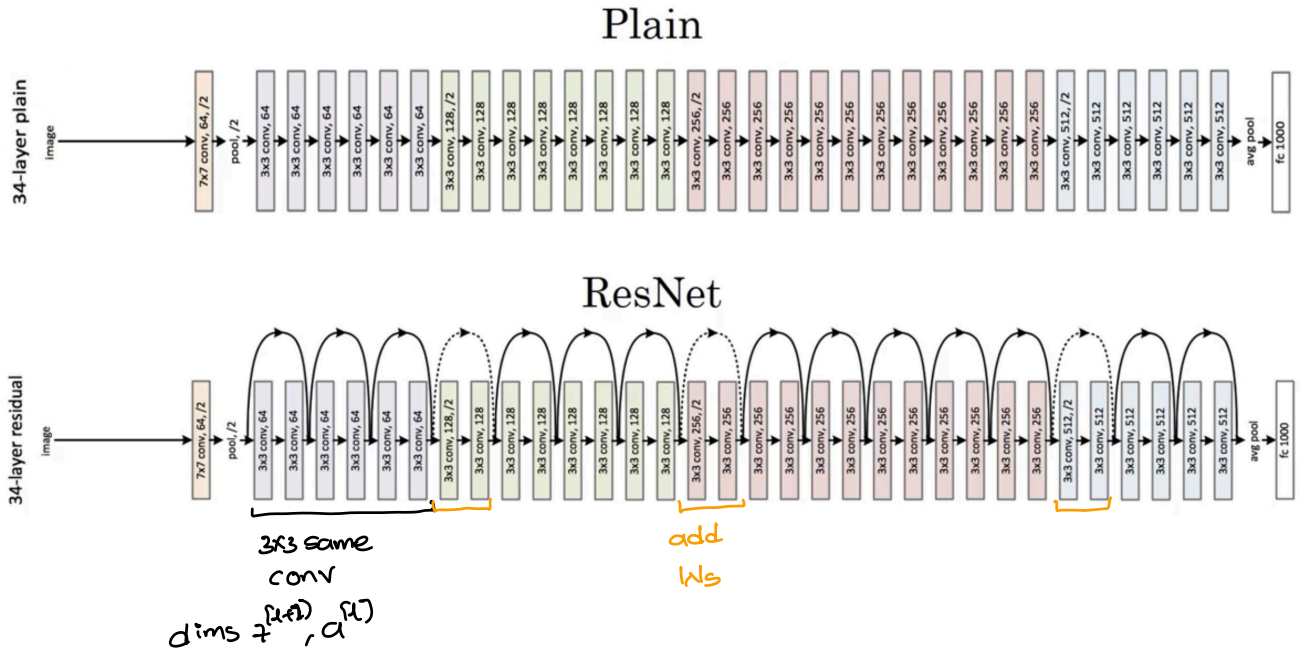
Shows:

- Identity function is easy to learn for residual block, because of skip connection.

- adding extra two layers with skip connection, doesn't hurt your NN ability to do as well as simpler NN.

- Hence we can train bigger Networks, we can learn better than identity function.

ResNet: (2015)



Lec 5: 1x1 convolutions:

1	2	3	6	5	8
3	5	5	1	3	4
2	1	3	4	9	3
4	7	8	5	7	9
1	5	3	7	4	8
5	4	9	8	3	5

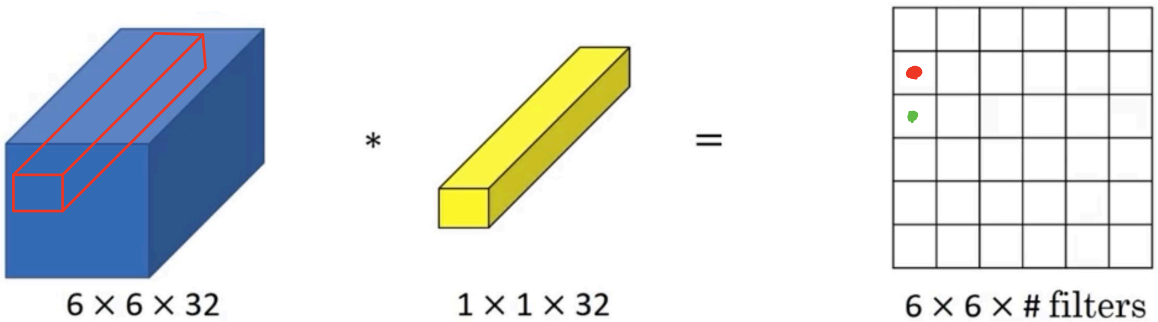
6 × 6 × 1

* 2 =

2	4	6	12	10	16
6					
4					
8					
2					
10					

multiply each element by 2.

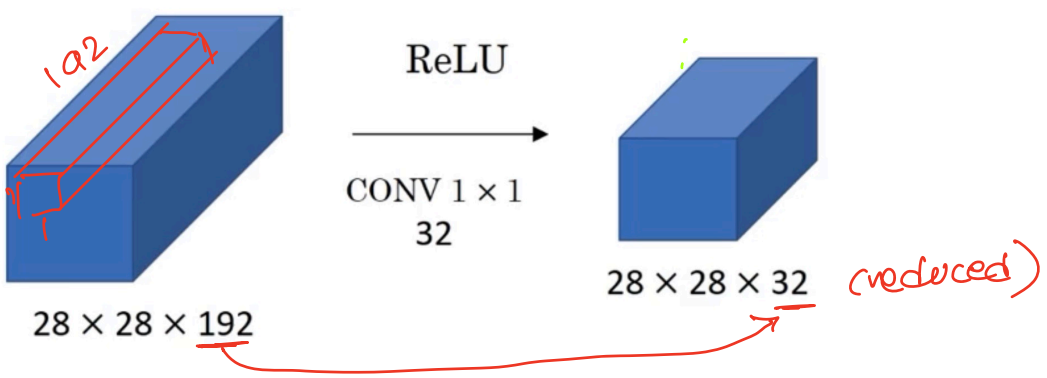
Multiple channel: (Network in Network)



$n_c^{[l+1]}$

→ element wise product between 32 numbers and apply ReLU, gives single scalar.

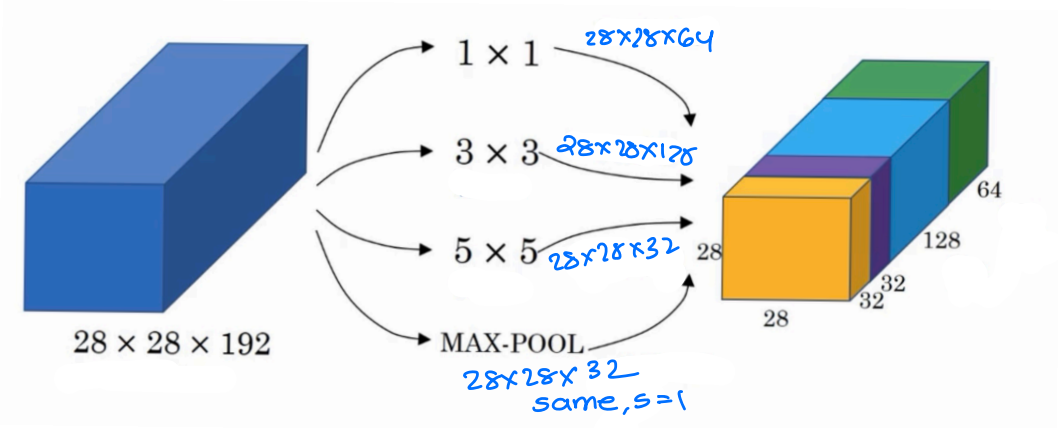
Using 1×1 convolutions:



Filter dim: 1×1×192 (reduces the no. of channels)

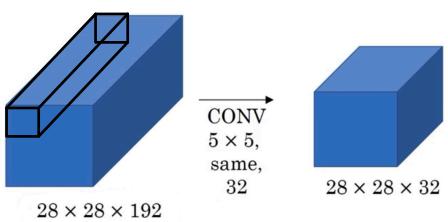
no. of filters: 32

Lec6: Inception Network:



- Instead of choosing what filter size you want, conv, or pooling use them all.

Problem: computational cost:



cost:

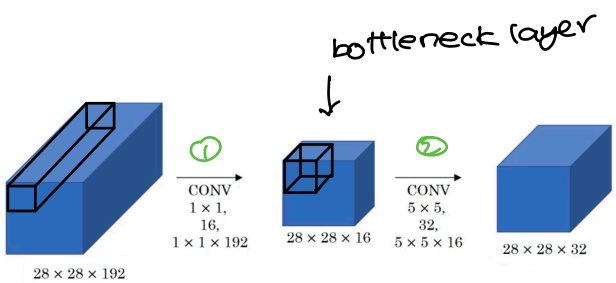
32 filters

filter size = 5×5×192

outputsize = 28×28×32

no.of multi = 28×28×32×5×5×192

= 120M



cost:

① 16 filters

filter size = 1×1×192

outsize = 28×28×16

no.of multi = 28×28×16×1×1×192

= 2.4M

② 32 filters

filter size = 5×5×16

outsize = 28×28×32

no.of multi = 28×28×32×5×5×16

= 10.0M

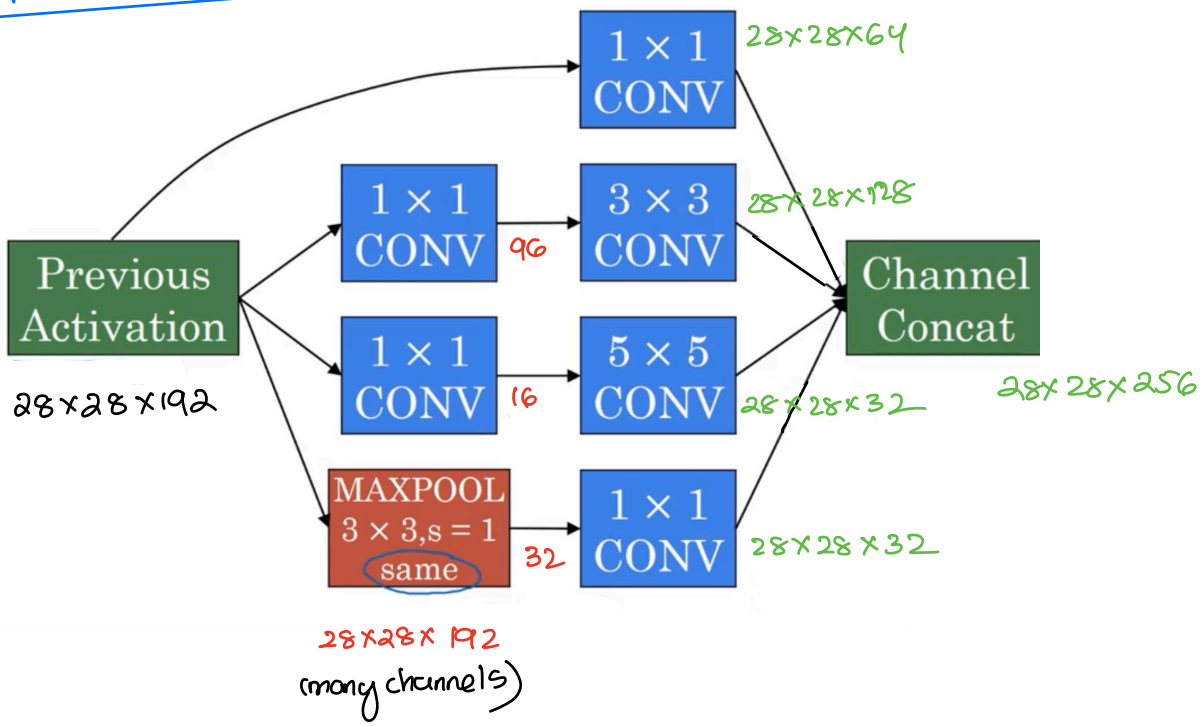
① = 120M

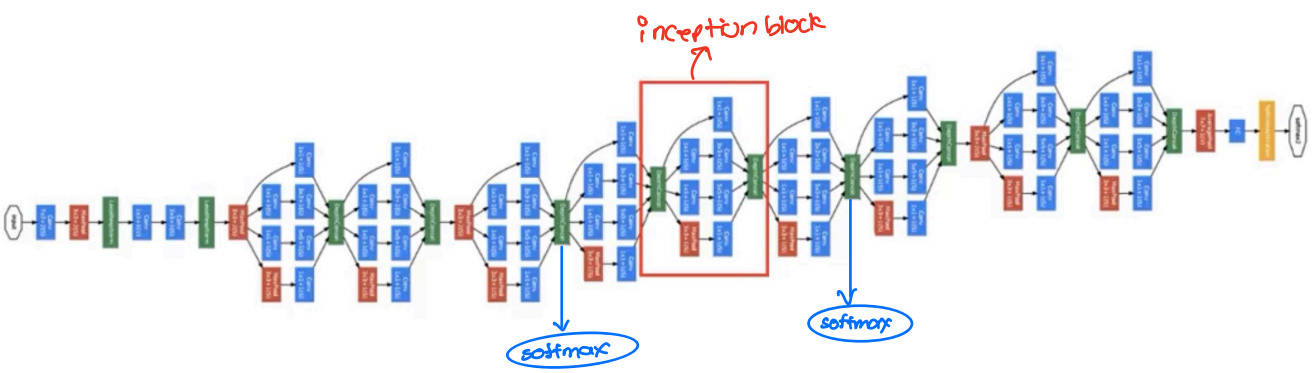
Reduce the cost

① + ② = 12.4M

Lec7: Inception Network: (GoogleNet)

Inception module:



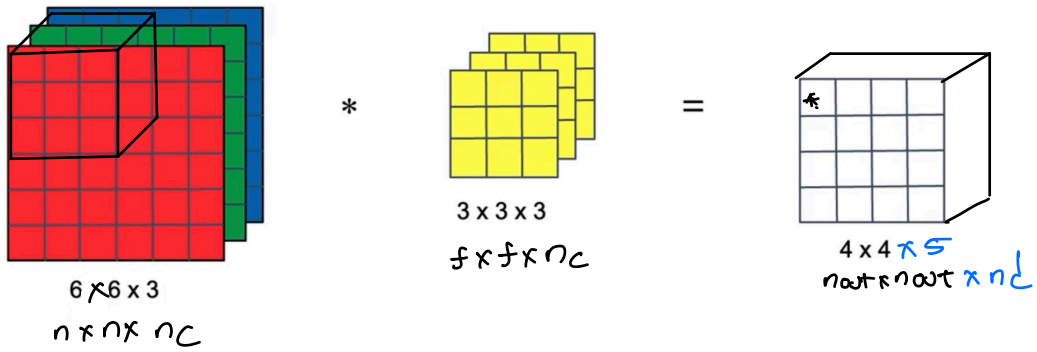


- multiple inception blocks
- side branches, for better regularization effect.

Inception Network → Inception Movie.

Lec8: MobileNet:

key idea: Normal vs depthwise separable convolutions

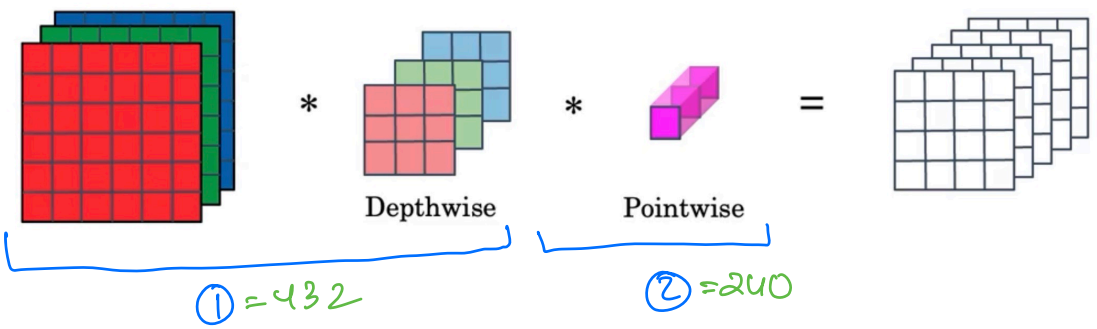


computation cost:

= filter params x filter positions x no. of filters

= 3 x 3 x 3 x 4 x 4 x 5 = 2160

Depthwise Separable Convolution



compute cost: 672 multiplications

$\frac{672}{2160} = 0.31$ save 3x compute cost

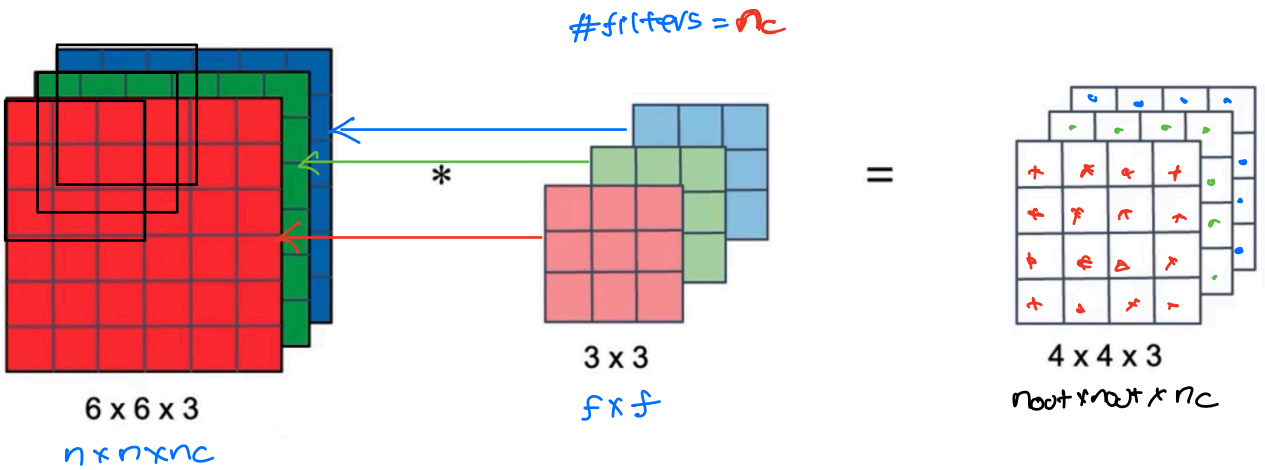
General, cost ratio = $\frac{\text{depth}}{\text{normal}}$

$= \frac{1}{n_c'} + \frac{1}{f^2} = \frac{1}{5} + \frac{1}{3^2} = 0.31$

If $n_c' = 512$, $f = 3$

cost ratio = 0.11 ⇒ 10x time cheaper

Depthwise Convolution:



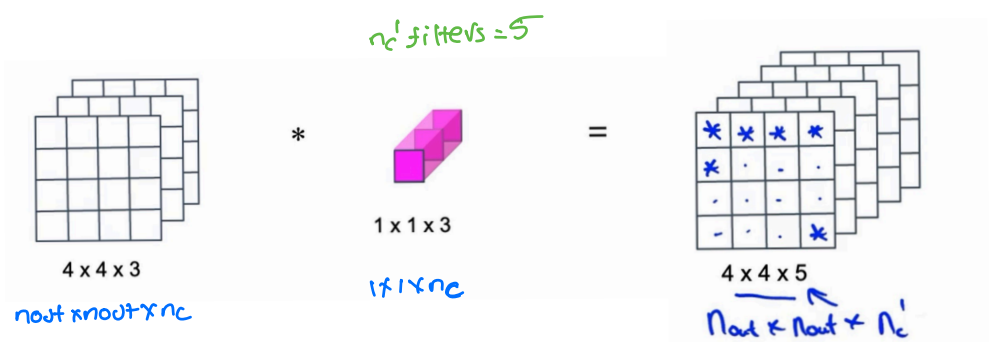
compute cost:

= filter params x filter position x no. of filters

= 3x3 x 4x4 x 3

= 432

Pointwise Convolution:



compute cost:

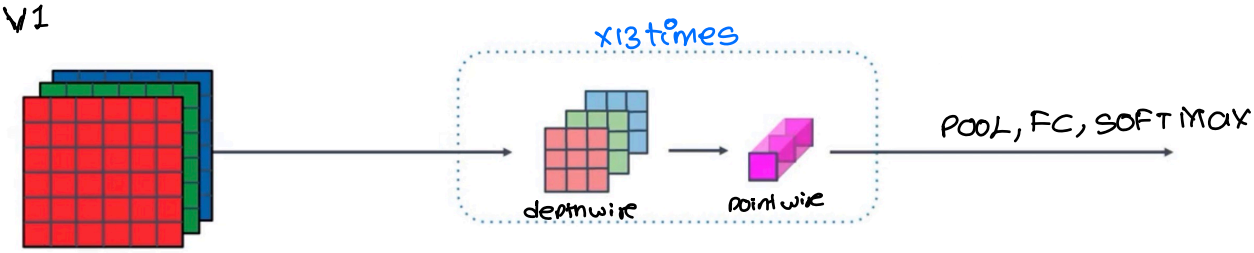
= filter params x filter positions x no. of filters

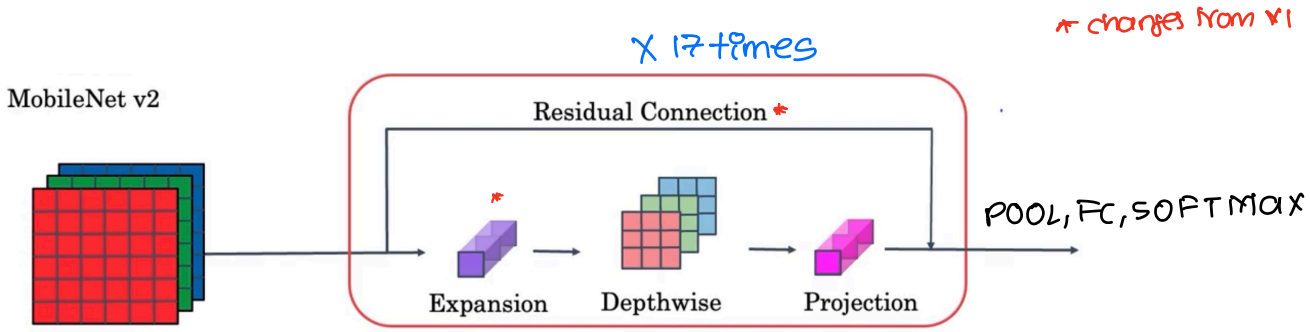
= 1x1x3x4x4x5

= 240

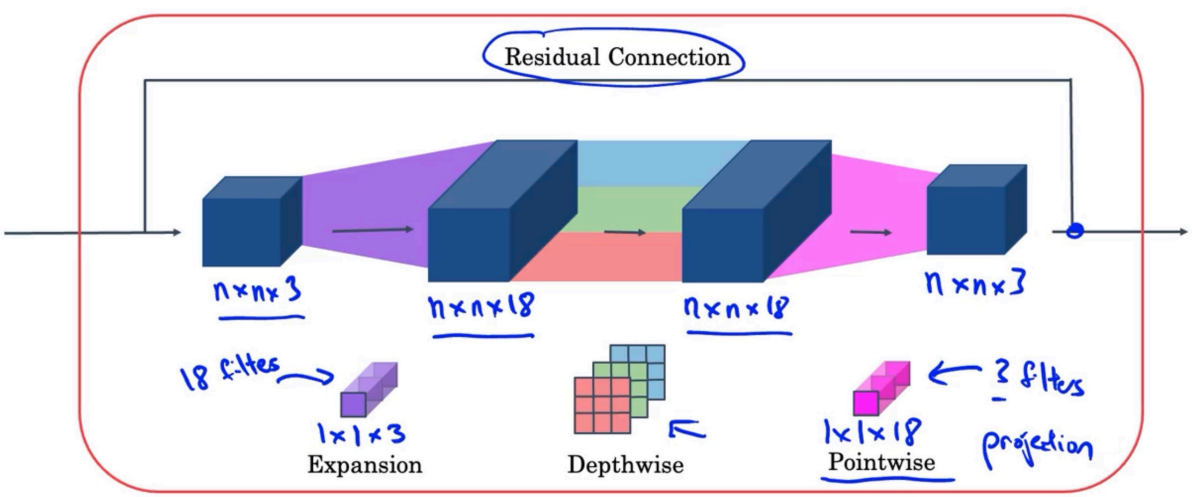
Recap: MobileNet: (2019)

V1





V2: Inverted residuals and linear bottlenecks.



projection.

expansion factor: 6

3 → 18

1×1×5430

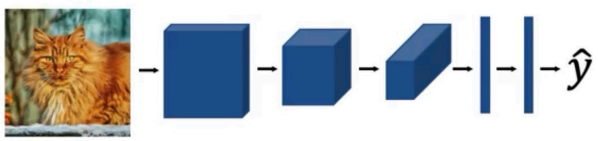
3×3×190

1×1×7040

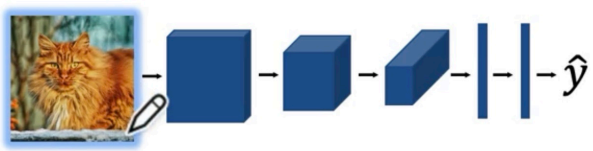
dec10: EfficientNet (2019)

- Model scaling for CNN.

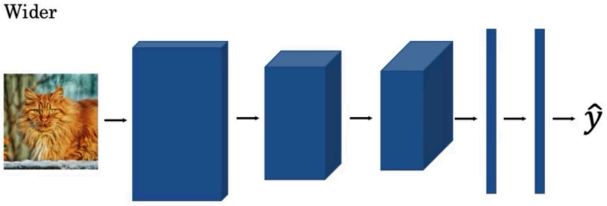
Baseline



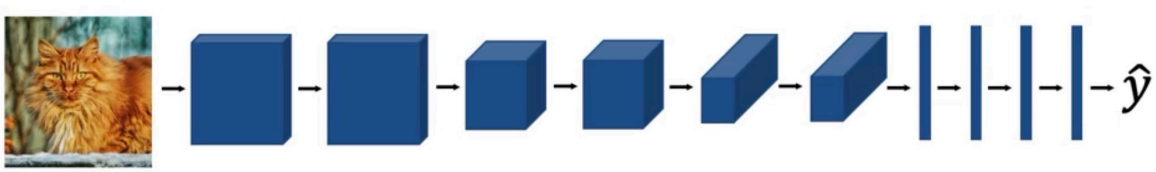
Higher Resolution



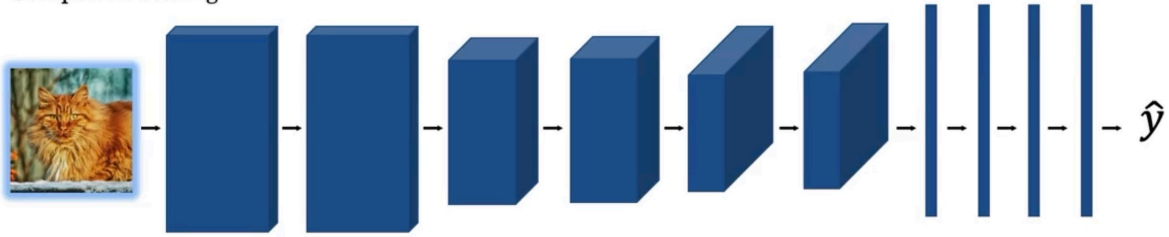
Wider



Deeper



Compound scaling



r - resolution

d - depth of NN

w - width of layers

Question?

Given a computational budget, what is good choice of (r, d, w) ?

$\text{compound} = \alpha_1 r + \alpha_2 d + \alpha_3 w$

- look at efficientNet code.

Advice:

- use open source code from github.
- use pretrained weights, do transfer learning. (from ImageNet)

→ freeze = 1

→ trainable parameter = 0

→ train softmax layer weights.

→ precompute activation for all training examples (before softmax).

→ if m >> large, then train bit more layers.

if you have alot of data

- train the whole network with your training data.

Data Augmentation:

- More data, better NN training for CV model
- Mirroring vertical
- random cropping (resonable large subset).
- color shifting (+20, −20, +20), (−20, +20, +20) - - - -

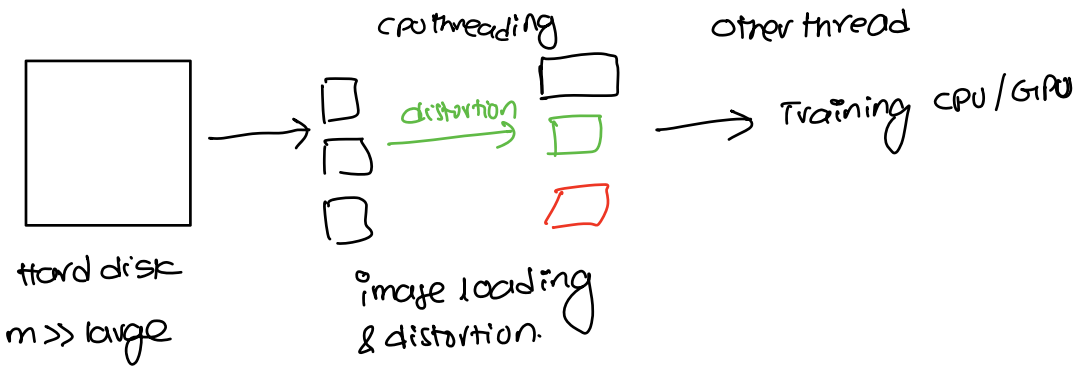
sampling R,G,B

- PCA color agumentation (AlexNet paper)

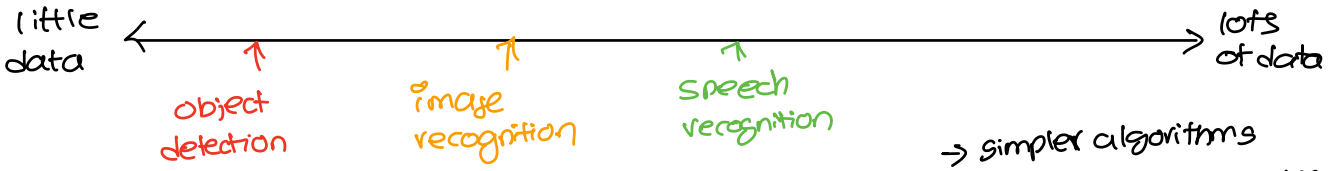
if image has more R, B and less G

then PCA add, sub alot in R, B, and less G

⇒ overall color remains same.


CPU threading
other thread
distortion
Training CPU / GPU
Hard disk
m >> large
image loading
& distortion.

State of Computer Vision :



→ more hacks
→ hand engineering
→ transfer learning

→ simpler algorithms
→ less hand engineering

Source of knowledge :

① labeled data (x, y)

② hand engineered features / NN architecture / other components . .

Doing well on benchmarks / winning competitions:

① ensembling (2 to 15 NN)

- train several NN independently and average their output. ($\hat{y}$)

② Multi crop at test time

- run classifier on multiple version of test image and average results. (10 crop)

③ use open source code.