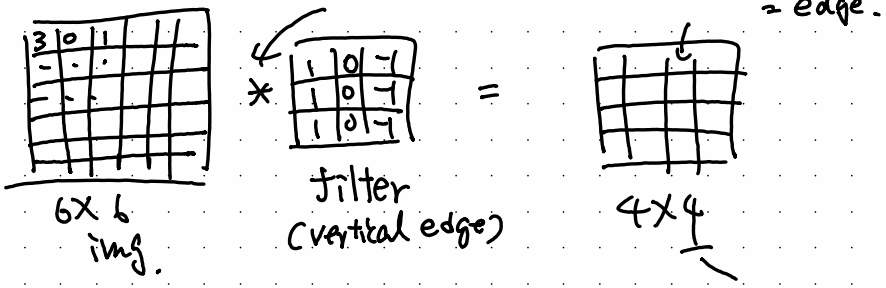
Course 4

Convolutional Neural Networks

CovNet

Computer Vision

Edge Detection



python: conv-forward

tensor: tf.nn.conv2d

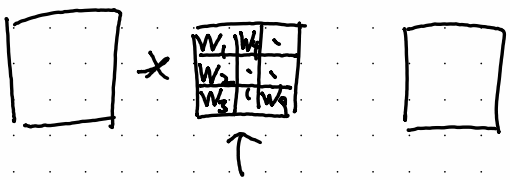
⋮

sobel filter

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

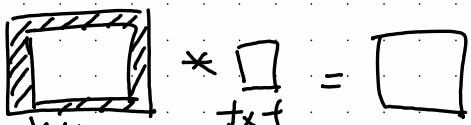
Schors filter.

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



use NN to learn filter

Padding



valid (no padding) — $n \times n$ * $f \times f$ = $n-f+1 \times n-f+1$

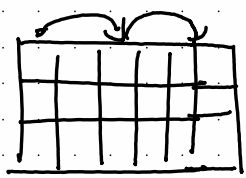
same (padding) — $(n+p) \times (n+p)$ * $f \times f$ = $n+2p-f+1 \times n+2p-f+1$

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

e.g. $f=3 \rightarrow p=1$, $f=5 \rightarrow p=2$

f is almost always odd — 3×3, 5×5, ⋮

Strided convolution



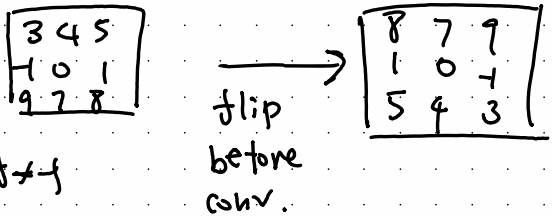
$n \times n$ * $f \times f$

pad: p, stride: s $\Rightarrow \left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{n+2p+f}{s}+1 \right\rfloor$

Convolution in math

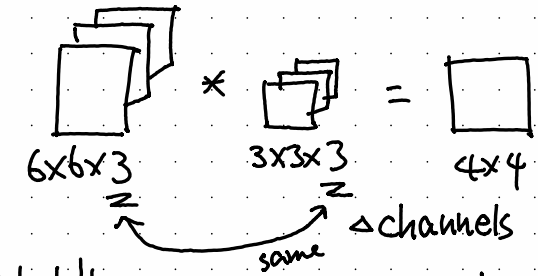
cross-corr : no flip filter

conv : flip filter

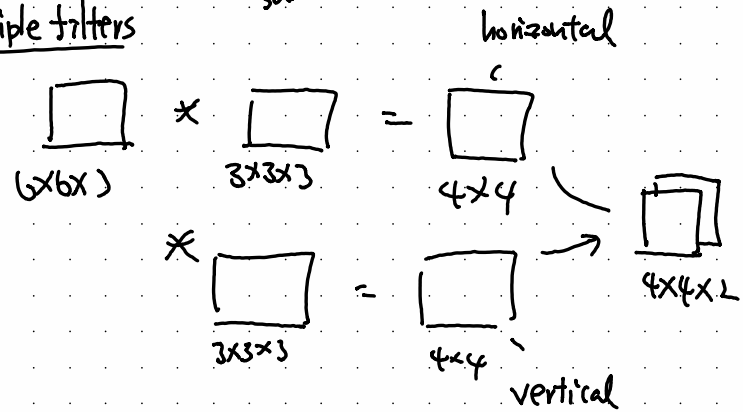


ML community just call cross-corr as convolution

Conv in volume

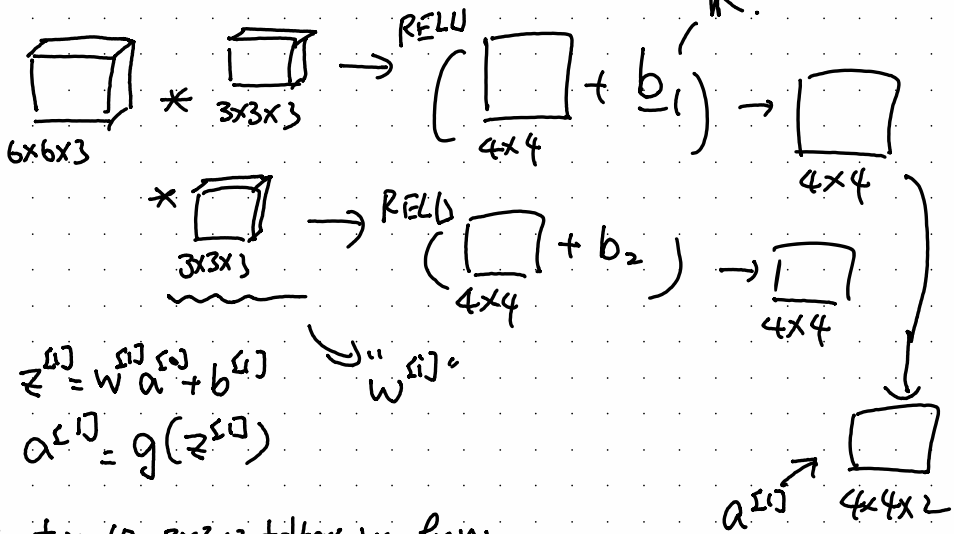


Multiple filters



$n \times n \times n_c \quad * \quad f \times f \times n_c \quad = \quad n-f+1 \times n-f+1 \times n_c$

one-layer CNN



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

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

e.g. for 10 3x3x3 filters, we have

$(3 \cdot 3 \cdot 3 + 1) \times 10 = 280$ parameters

* parameters are independent from image resolution

Notation

$f^{[l]}$ - filter

$p^{[l]}$ - padding

$s^{[l]}$ - stride

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

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

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

$$n_w \left\lfloor \frac{h_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

each filter:

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

Activation

$a^{[l]} \rightarrow n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$ $\quad A^{[l]} \rightarrow m \times n_H^{[l]} \times h_w^{[l]} \times n_c^{[l]}$

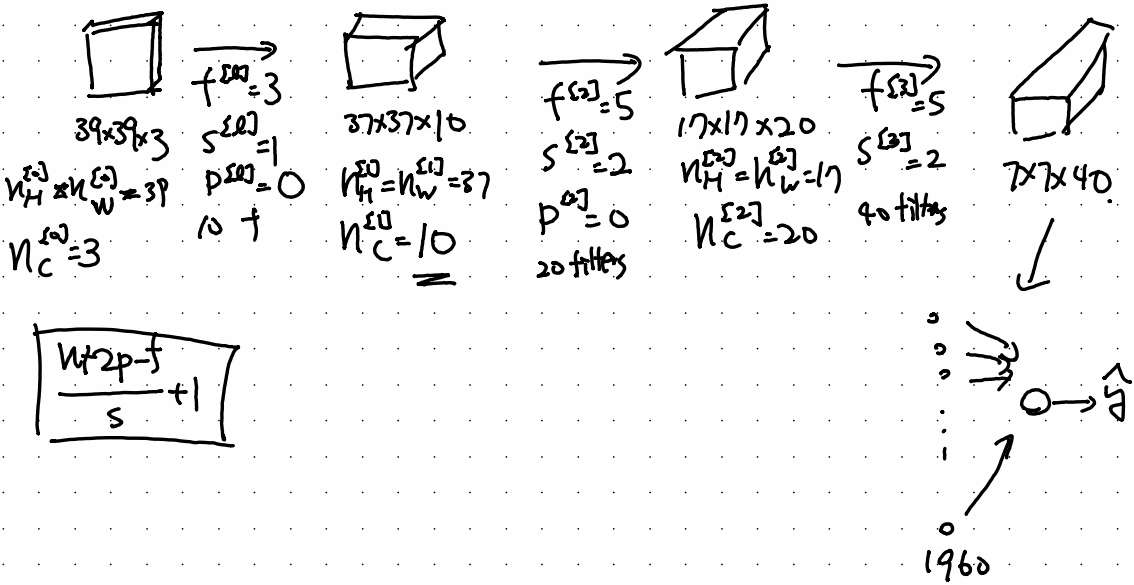
weights

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

bias:

$h_c^{[l]} - (1,1,1,h_c^{[l]})$

CNN



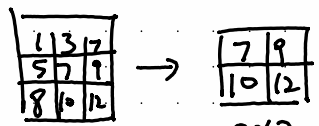
$$\frac{n+2p-f}{s}+1$$

TYPE of LAYERS in CNN

- Convolution (Conv)
- Pooling (Pool)
- fully connected (FC)

POOLING LAYER

Max pooling - preserve feature?



4×4 → 2×2

f-2
s-2

△ no parameters to learn △

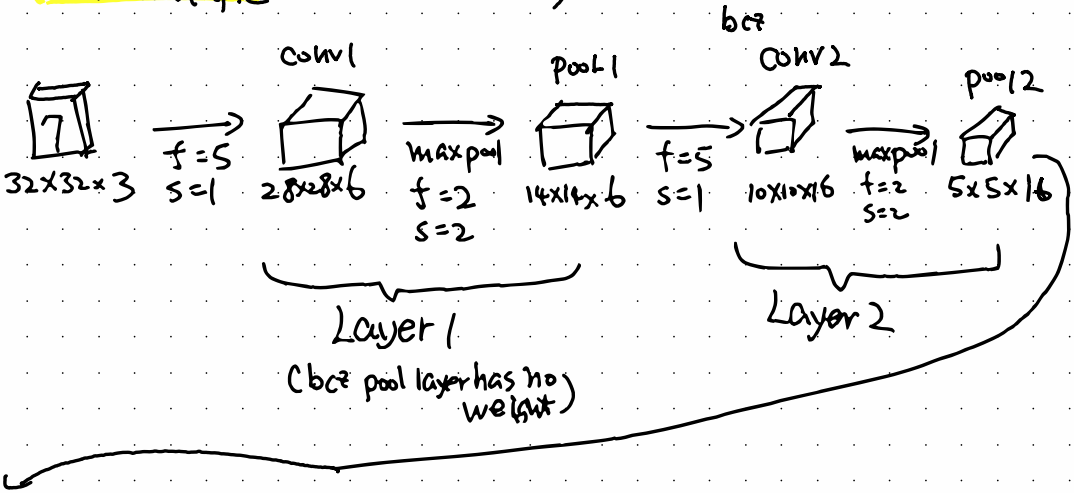
Average pooling - not often used

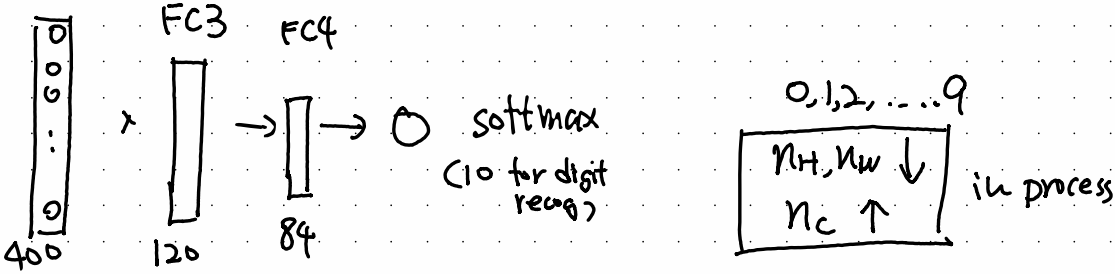
hyperparameters
- f - filter size — f=2, s=2; f=3, s=2
- s - stride
- type of pooling layer

$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor$

* no parameter to learn

NN example (LeNet-5)





$W^{[3]}$ (120, 400)

$b^{[3]}$ (120)

△ FC layers have lots parameters!

✡ Why conv? ✡

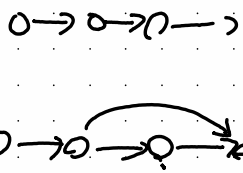
reduce # parameters compared

- parameter sharing - feature detector useful for all parts of image
- sparsity of connections - each output value depends on small part of image

△ translation invariant.
(pooling layer)

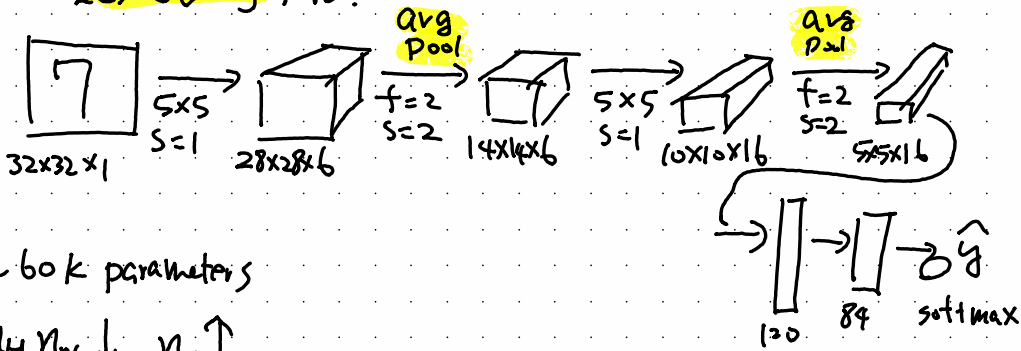
Keras.

- sequential API
- functional API



CASE STUDY

LeNet-5, 98.



~60k parameters

$n_H\ n_w \downarrow \quad n_c \uparrow$

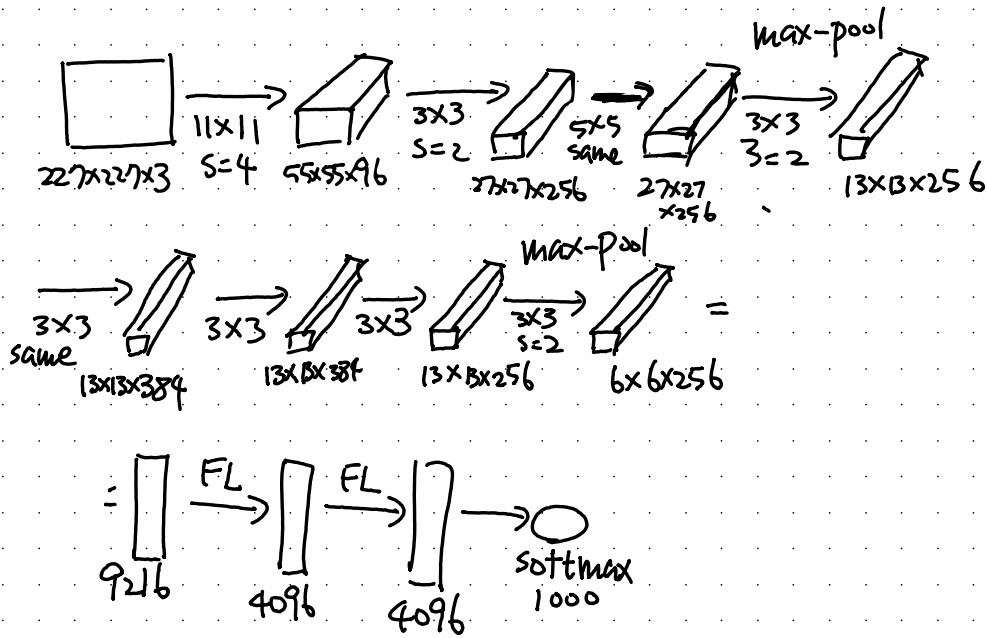
conv pool conv pool fc fc pool

original

sigmoid/tanh
different filter
nonlinearity after pooling

section II, III

AlexNet. 2012



~ 60M parameters

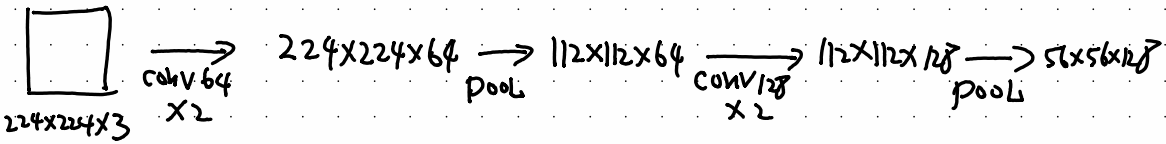
- ReLU

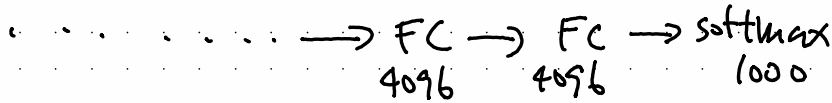
Orig

- multiple GPU
- local response normalization (LRN)
  - normalize along axis
  - less often used.

VGG-16, 2015

CONV = 3×3, S=1, same, MAX-POOL = 2×2, S=2





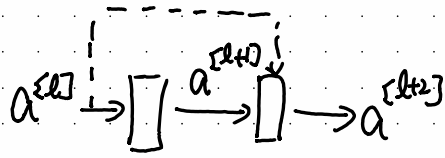
~138M parameters LARGE!

~uniform

△ nH nW ↓ nC ↑

Residual Network (ResNets) 2015

Residual Block.

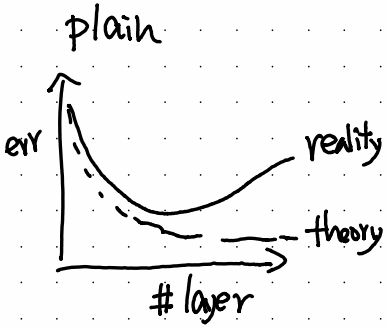


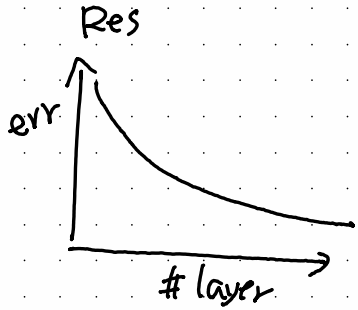
SHORT CUT / SKIP CONNECTION

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

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

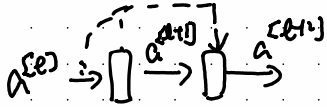
ResNet $\Rightarrow a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$





△ help training deeper network.

Why ResNet Work?



$W_s a^{[l]}$ if dimension changes.

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

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

IF $w^{[l+2]} \to 0$, $b^{[l+2]} \to 0$

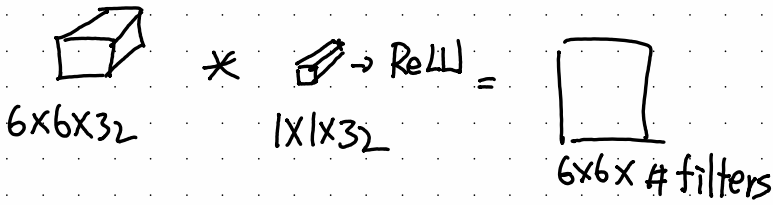
$\Rightarrow a^{[l+2]} = g(a^{[l]})$

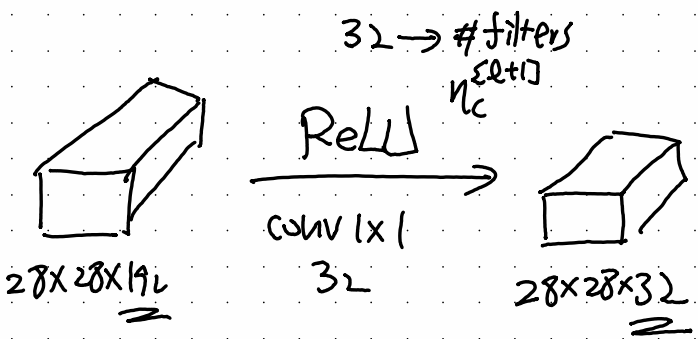
→ identity function is easy to learn

→ ① res block doesn't hurt.

→ ② prevent deeper layers making result worse.

1×1 convolution / network in network

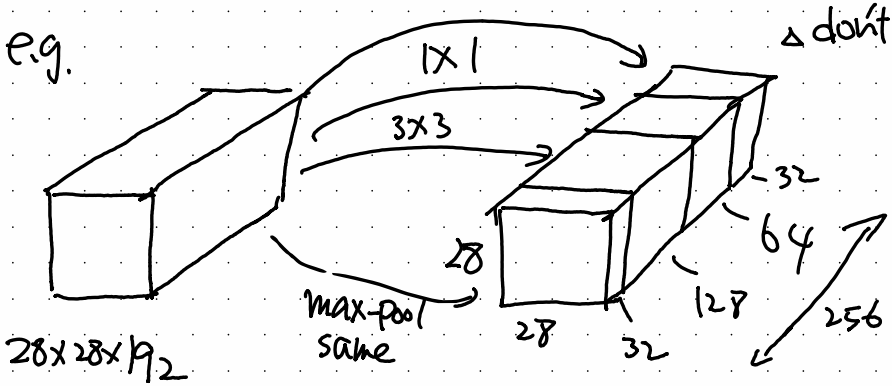




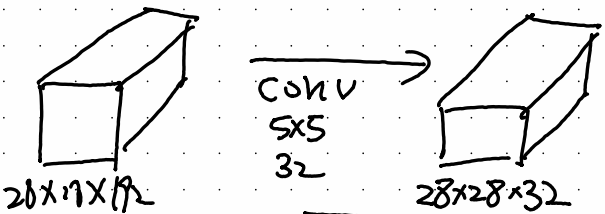
1×1 → shrink n_c
filter → shrink w, h.

shrink n_c, if we like.

Inception Network, 2014

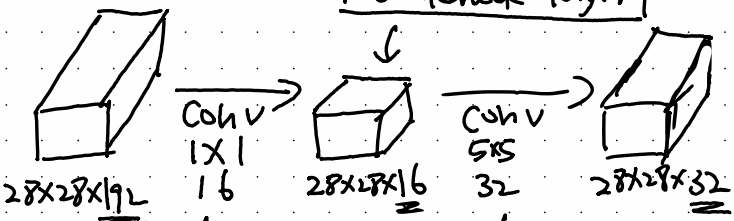


△ try all different size at once!



#op
28x28x192 x 29x28x32 ≅ 120M!

bottleneck layer



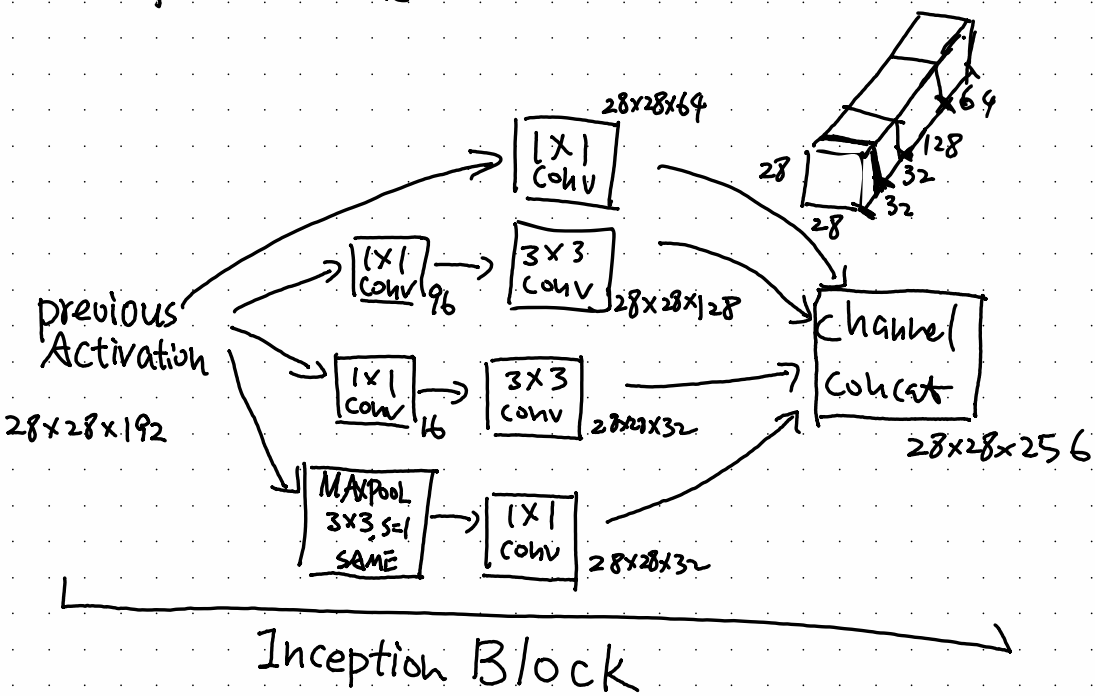
← Inception

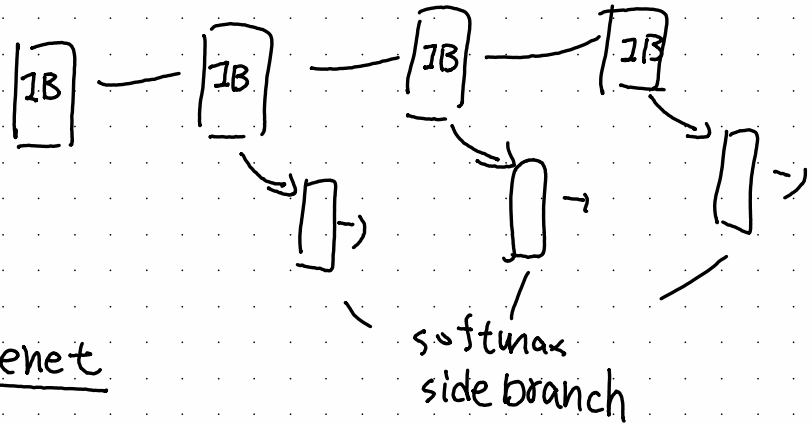
28x28x192x16
=2.4M

28x28x16x5x5x16 ≅ 10M

⇒ ~12.4M

Inception Module





goolenet

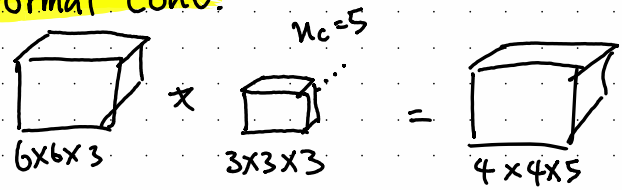
"we need to go deeper"

Inception

MobileNet. 2017

- low computational cost.
- Normal v.s depthwise-separable convolution
  ⇒ depth-wise + point-wise.

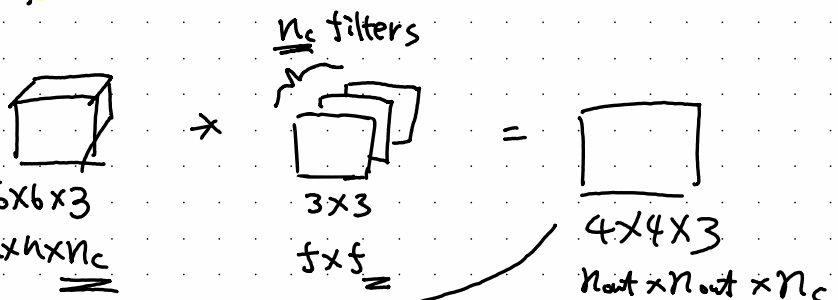
Normal Conv.



cost = # filter para x # filter pos x # filter

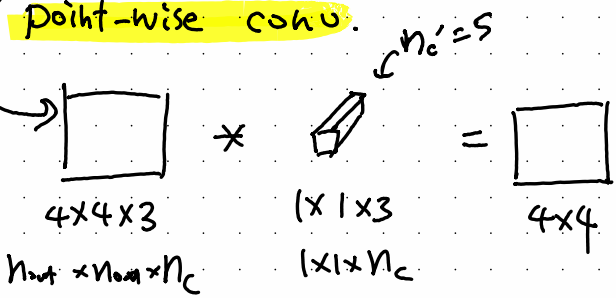
= 2160 op.

Depth-wise Conv



→ 432 op

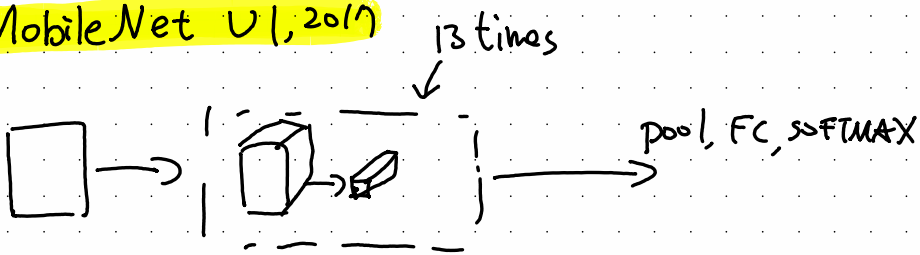
Point-wise conv.



→ 240 op

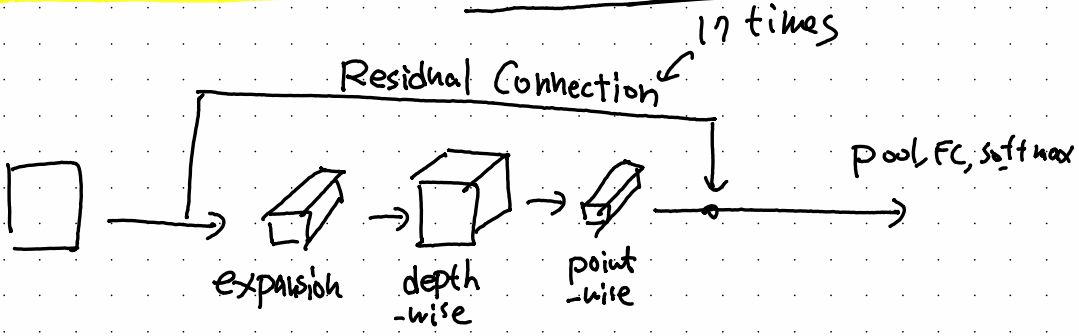
~10 times cheaper

MobileNet V1, 2017

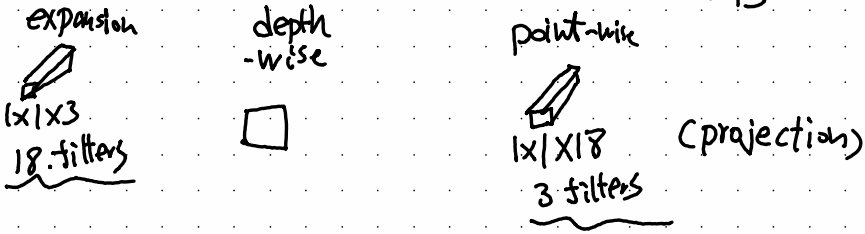


MobileNet V2, 2019

bottleneck blocks



eg. $n \times n \times 3 \rightarrow n \times n \times 18 \rightarrow n \times n \times 18 \rightarrow n \times n \times 3$



- expansio → increase representation in bottleneck block
- projection → reduce bandwidth between layers

EfficientNet 2019

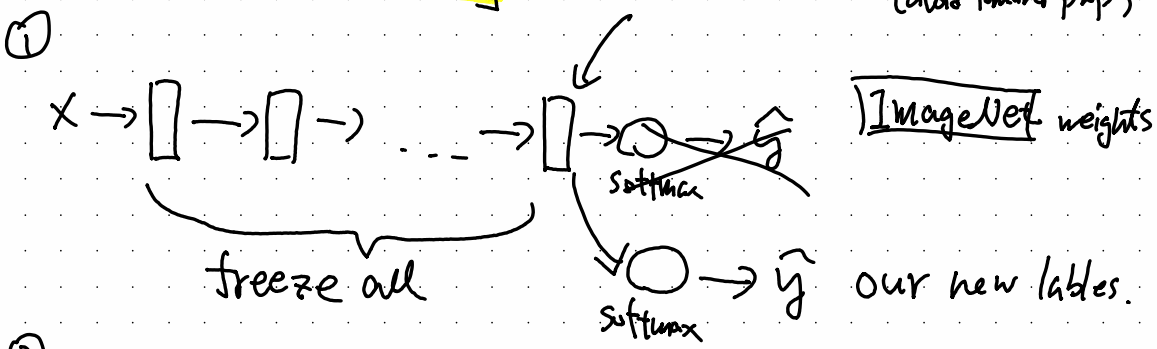
- automatic scale up/down model

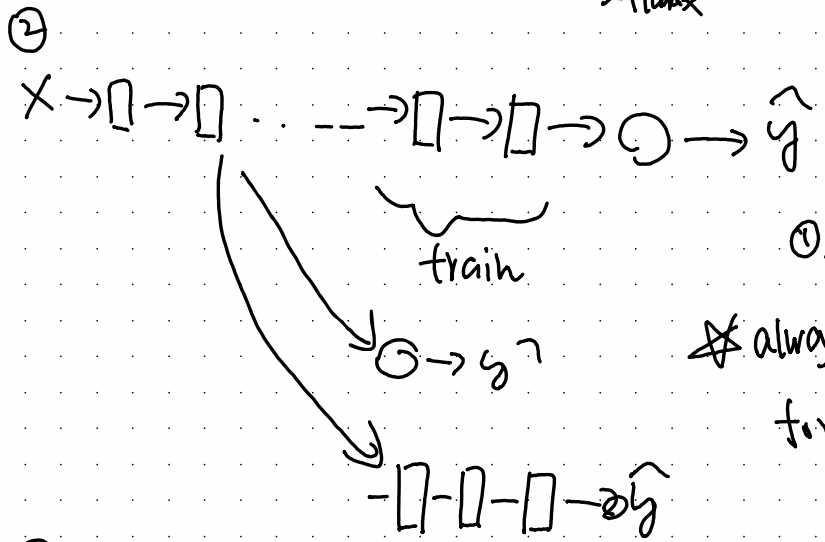
scale
- r resolution
- d depth
- w width

the model selects r,d,w that suits device.

Transfer Learning

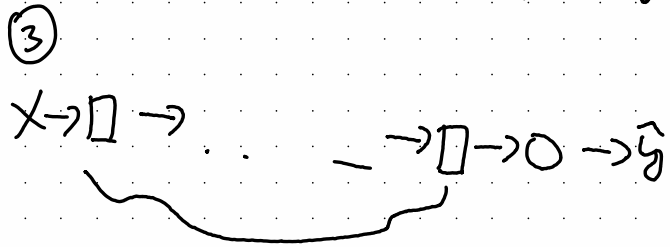
precompute this, to accelerate (avoid forward pass)





①, ② if small dataset.

✡ always start from existing weights.
for CV problems

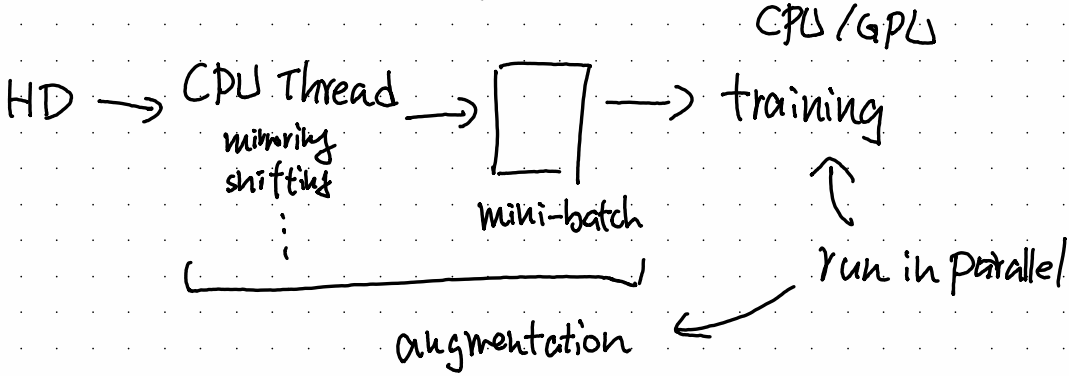


use previous weights
as initialization
(if we have a large data set)

Data Augmentation

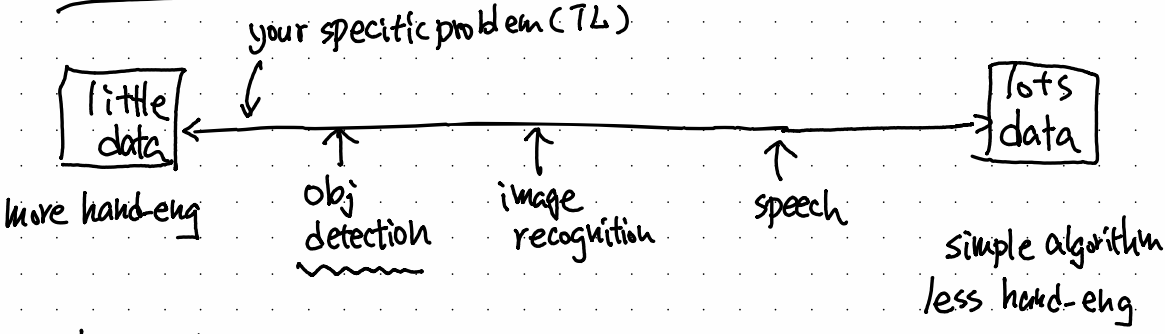
- mirroring
- random cropping
  - might crop some less informative area
- color shifting
  - PCA color augmentation
  - simple RGB shifting, e.g different sunlight.

Distortion during training



State for CV.

data vs hand-engineering (hack)



knowledge
- label
- hand engineered features / network architecture / ...

Tips for doing well on benchmarks

- Ensembling (3~15 networks)
  - train several networks and average outputs ($\hat{c}_j$)
    (less useful in practical)
- Multi-crop at test time
  - run classifier on multiple vers of test images and avg results
    e.g. (10-crops)

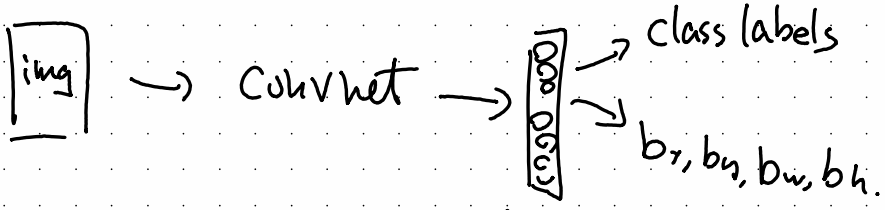
Use open source code

- use network arch.
- use implementation
- use pre-trained model

object detection

- classification - car, cat etc
- localization - bounding box
- detection - multiple objs of different categories.

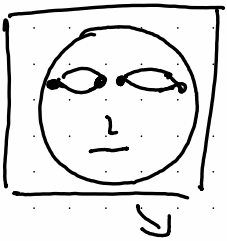
classification with localization



$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ \vdots \end{bmatrix} \begin{matrix} \leftarrow \text{prob of objects exist} \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{bb} \\ \left.\begin{matrix} \\ \\ \end{matrix}\right\} \text{category of the obj} \end{matrix}$$

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + \dots (\hat{y}_j - y_j)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

Landmark detection



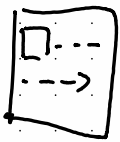
eyes: l_{1x}, l_{1y} l_{2x}, l_{2y} ⋮ l_{4x}, l_{4y}

additional landmarks e.g. mouse. nose ⋮

- same for pose detection
  skeleton landmarks in a photo.

obj detection

sliding window detection

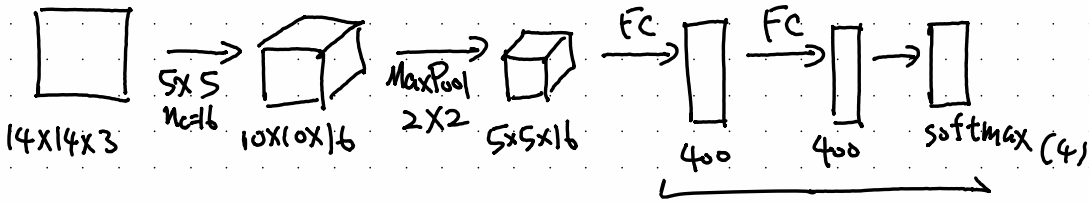


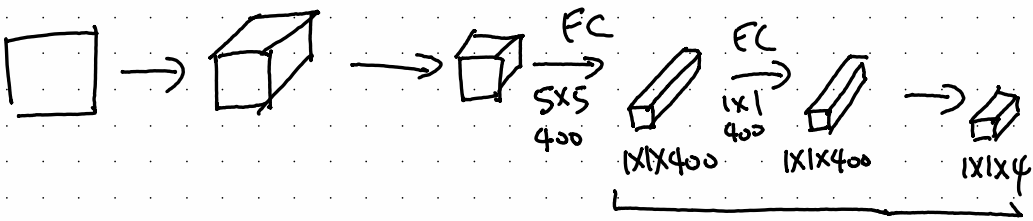
send each window through ConvNet.
from small to large window size

- high computation cost.

Convolutional Implementation

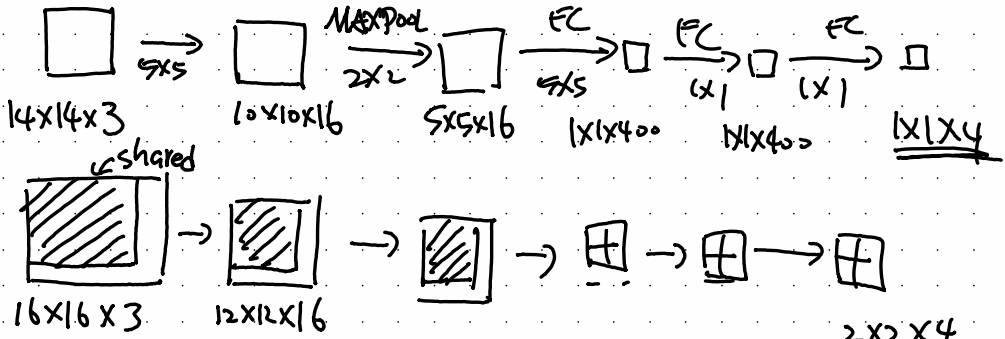
turn FC to convolutional layer





OverFeat. 2014.

Conv implementation



→ reuse overlapped computation result.

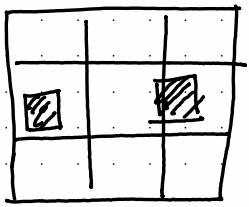
✡ 2x2x4 corresponding to 4 sliding window position

→ instead of sequentially send each sliding window through ConvNet, simultaneously do prediction for all window position

→ the window position is not precise though.

Output accurate bounding boxes

YOLO- You only look once, 2015 (a difficult paper to read)



100

100

target output

3x3x8

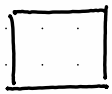
grid

P, b, c

for each grid cell

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

- assign obj to grid based on mid-point
- relative coord to the grid cell
- b_x, b_y - between 0~1
  b_w, b_h - could be > 1



100x100x3

→ conv → max pool →

3x3x8

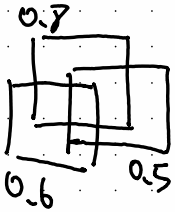
⇒ increase grid resolution if need higher precision.

~ still one convnet implementation

Intersection over Union (IoU)

correct if IoU > 0.5

Non-max suppression



find max → supress others with high IoU

each grid cell $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

discard all boxes with $P_c \leq 0.6$

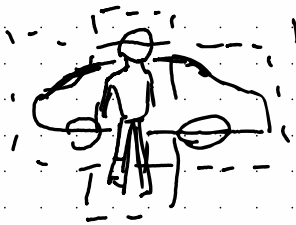
for remaining boxes
- pick largest P_c

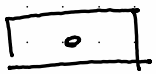
discard remaining with IoU ≥ 0.5 with other boxes.

Anchor boxes

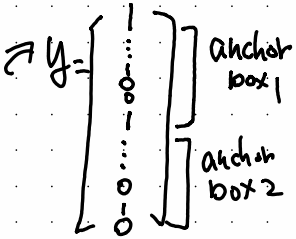
predefined shapes

box1 box2



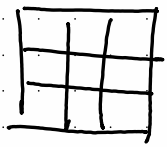


$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ c_1 \\ c_2 \\ c_3 \\ P_c \\ b_x \\ b_y \\ i \end{bmatrix}$ anchor box1, anchor box2



each obj, assigned to cell
- Contain mid-point
- anchor box for the cell with high IoU

Yolo algorithm



c
1 - pedestrian
2 - car
3. - motocycle

y $3 \times 3 \times 2 \times 8$

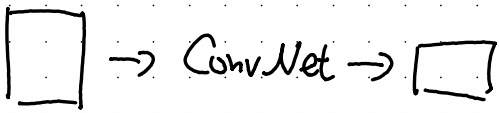
2: anchor boxes; 8: 5 + #classes

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \\ P_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

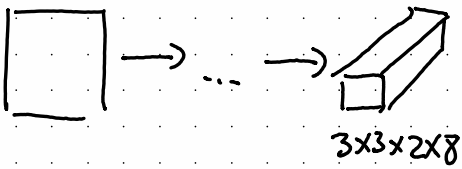
anchor box 1

anchor box 2

Training



Prediction



$$y = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ 0.2 \\ 0.3 \\ 0.2 \\ 0.1 \\ 1 \\ 0 \\ 0 \\ \vdots \\ ? \end{bmatrix}$$

non-max supression

for each cell,

- get 2 predicted bb (for each anchor boxes)
- get rid of low P_c prediction
- for each class, run non-max supression

Region Proposals. R-CNN, 2013

- pick interesting region to run CNN

segmentation → run CNN on interesting regions

R-CNN: propose region, CNN, output

Fast R-CNN: propose region, Convolution implementation, output (2015)

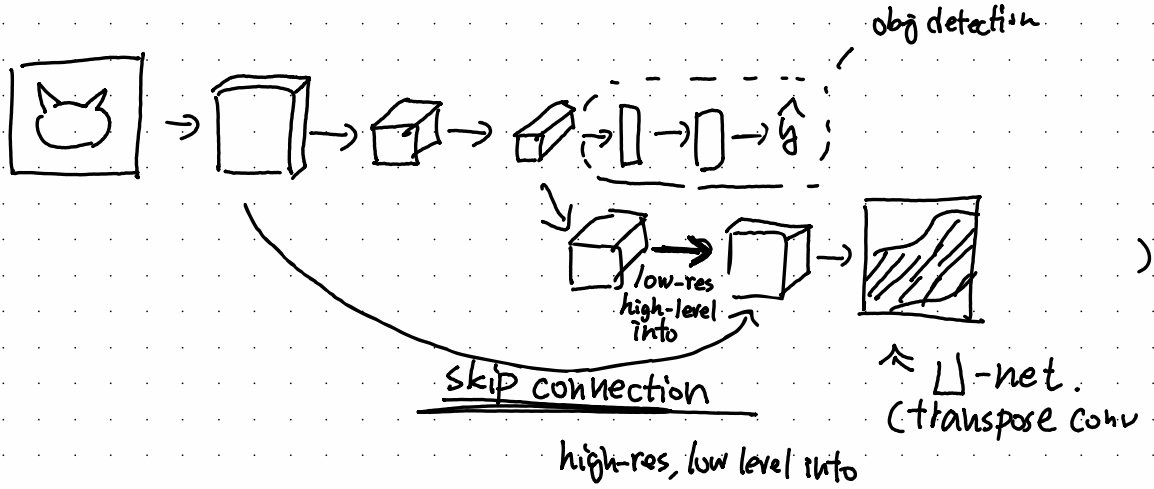
Faster R-CNN: convnet to propose regions (2016)

still slower than YOLO...

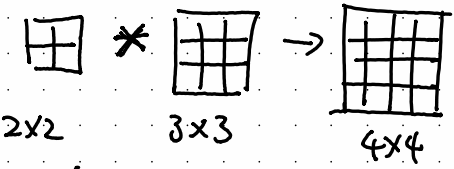
Semantic Segmentation U-net, 2017

obj detection - bb

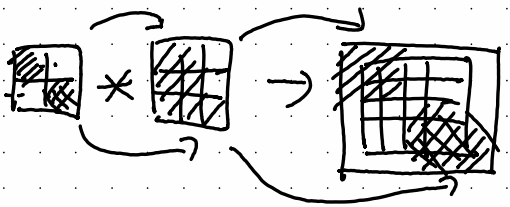
semantic seg - per-pixel labeling



Transpose Conv, deconvolution, upscale conv

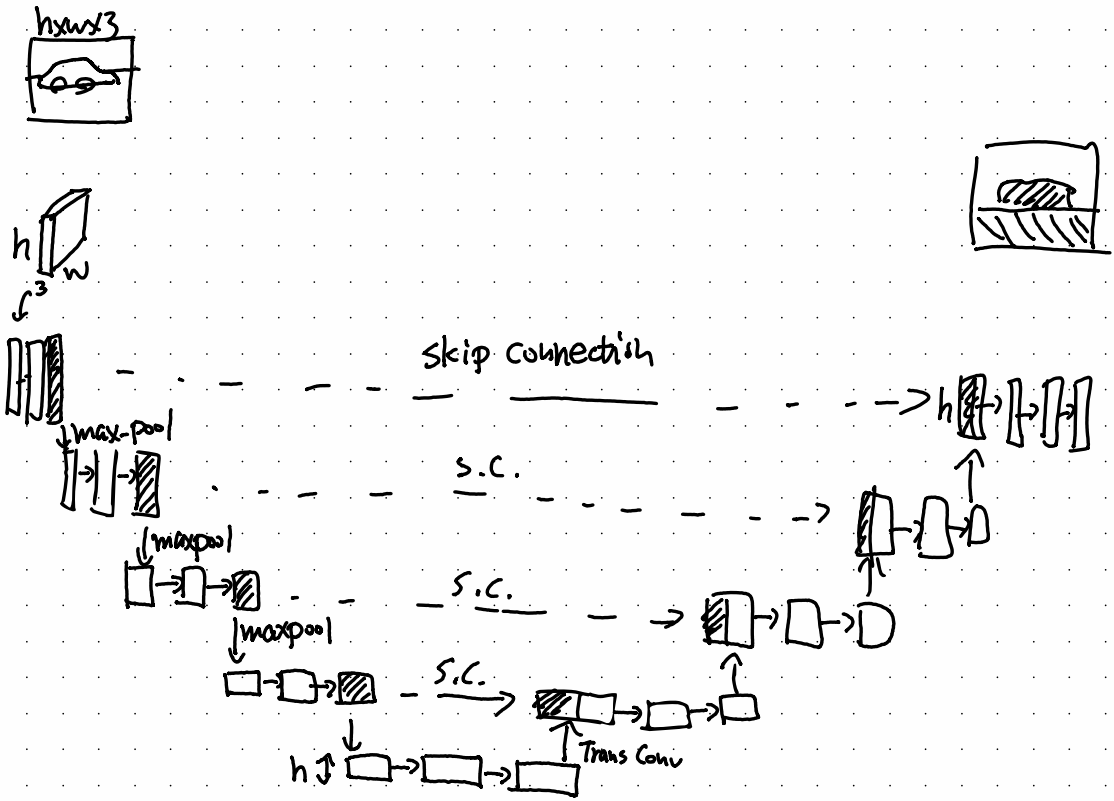


f=3, p=1, s=2



if overlap, add values together.

U-Net, 2015 (on biomedical image seg)



Face Recognition

- verification
  - input image, name
  - output whether image/name match (1 to 1)
- Recognition
  - database of k people
  - input image
  - output ID of the image among k. (1 to k)

one-shot learning

learn from one example to recognise the person again

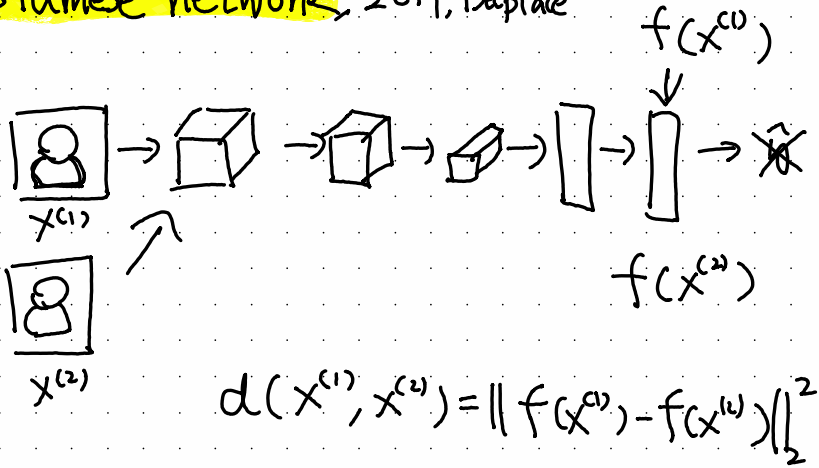
learn a "similarity" function

d(img1, img2) = degree of difference between imgs.

if $d(img1, img2) > \tau$, same person

$< \tau$, no match

Siamese network, 2014, DeepFace



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

parameters of NN define $f(x^{(i)})$

learn parameters that

if $x^{(i)}, x^{(j)}$ are same person $d(x^{(i)}, x^{(j)})$ small

different large

Triplet Loss FaceNet, 2015


Anchor
A


Positive
P


Anchor
A


Negative
N

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

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

α: margin
to prevent trivial solution
e.g. $f(img) = 0$
or $f(img^{(1)}) = f(img^{(2)})$,

Loss function

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

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

training set: 10k picture of 1k persons

Choosing A, P, N

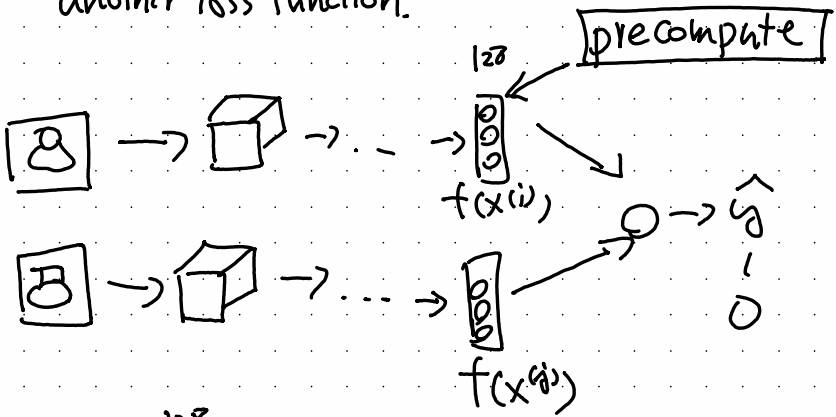
if chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is too easy.

→ choose A, P, N that are "hard" (NN learns little)
(check paper)

current K = 100m images
commercial

DeepFace. 2014

another loss function.



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

- kth feature

or χ^2

Neural Style Transfer, 2015

Content (C) + Style (S) → generated (G)

Visualize ConvNet 2013

hidden unit covers larger patch →

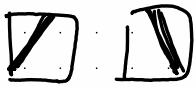
[8] → conv → conv → ..... → FC → FC → $\hat{y}$

pick a unit in layer l, find image patches that maximize unit activation

repeat for all units

(patch size ↑, for deeper layer.)

lower lv –



higher level

Neural Style Transfer. 2015

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

1. init G randomly (like noise)
2. gradient descent to minimize $J(G)$

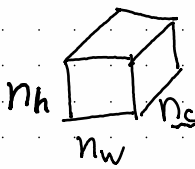
$J_{content}(C,G)$

- use hidden layer l to compute content cost (usually middle of network)
- use pre-trained ConvNet.
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be activation of layer l

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

$J_{style}(S,G)$

at layer l, define "style" as correlation between activation across channels



$a^{[l]}_{i,j,k}$ = activation at (i,j,k). $G^{[l]}: n_c^{[l]} \times n_c^{[l]}$

intuition: each channel correspond to hidden units that capture different levels of features

$G^{[l](S)}_{kk'} = \sum_i \sum_j a^{[l]}_{ijk} a^{[l]}_{ijk'}$ ← style

"Gram Matrix"

$G^{[l](G)}_{kk'} = \ldots$ ← generated

$$J_{style}(S,G) = \| G^{[l](S)} - G^{[l](G)} \|_F^2 = \frac{1}{2 n_H^{[l]} n_w^{[l]} n_c^{[l]}} \sum_k \sum_{k'} (G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'})$$