Cousera Deep learning course 4 Note

CNN

Week 1

Vertical edge detection

$$\begin{bmatrix} 3 & 0 & 1 & & & \\ 1 & 5 & 8 & & & \\ 2 & 7 & 2 & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix} \underset{\text{"convolution"}}{*} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ & & & \\ & & & \end{bmatrix}$$

6 x 6 * 3 x 3 filter = 4 x 4

$3\times1 + 1\times1 + 2\times1 + 0\times0$

$+5\times0 + 7\times0 + 1\times-1 + 8\times-1 + 2\times-1 = -5$

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

6x6 3x3 4x4

vertical edges

$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} \qquad \begin{matrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{matrix}$$

sobel filter   Schar filter

$$\left.\begin{matrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{matrix}\right\} \text{can be trained}$$

Padding

3×3

6×6 $\quad f \times f$

$n \times n$ $\quad n-f+1 \times n-f+1 \longrightarrow 4 \times 4$

$6-3+1=4$

Padding

3×3

$6 \times 6 \rightarrow 8 \times 8$ $\quad f \times f$

$n \times n$ $\quad n+2p \times n+2p$ $\quad n+2p-f+1 \times n+2p-f+1 \longrightarrow 6 \times 6$

$8-3+1=6$

"Valid" convolution $\quad n \times n \ast f \times f \longrightarrow n-f+1 \times n-f+1$

usually odd

"Same" convolution $(n+2p-f+1) \times (n+2p-f+1)$

Strided convolution

$7 \times 7 \ast 3 \times 3 = 3 \times 3$

floor

stride = 2

$n \times n \ast f \times f$ $\quad = \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

$p=0 \quad s=2$ $\quad 3 \times 3$

Convolution on RGB images

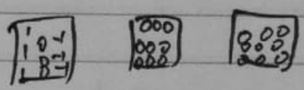
$6 \times 6 \times 3$ (height width channels) $*$ $3 \times 3 \times 3$ (height width channels) $= 4 \times 4$ (height width)

R.G.B

27 numbers $*$ 27 numbers $=$ 1 number

$(9+9+9)$

R G B



multiple filter

$$6 \times 6 \times 3 \begin{cases} * \ 3\times3\times3 = 4\times4 & \text{vertical edge} \\ * \ 3\times3\times3 = 4\times4 & \text{horizontal edge} \end{cases} \rightarrow 4\times4\times2$$

$$n \times n \times n_c \ * \ f \times f \times n_c \rightarrow n-f+1 \times n-f+1 \times n_c'$$

n_c ↑ channel (or depth)

n_c' filters

n_c' ↑ # n_c' filters

One layer of convolution network

"$W^{[1]}a^{[0]}$"　bias　$z^{[1]}$

g　$\mathbb{R}$

$$6\times6\times3 \longrightarrow * \; 3\times3\times3 \longrightarrow \text{Relu}(4\times4 + b_1) \rightarrow 4\times4$$

"$a^{[0]}$"

$$\rightarrow * \; 3\times3\times3 \longrightarrow \text{Relu}(4\times4 + b_1) \rightarrow 4\times4$$

"$W^{[1]}$"

$$4\times4\times2$$

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

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

e.g.: 10 filters that are $3\times3\times3$ in one layer
How many parameters does that layer have?

$3\times3\times3$

27 parameters

+ bias

= 28 parameters $\times 10$ = 280 parameters

Summary of notation

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

l = layer number

$s^{[l]}$ = stride

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

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

Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

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

Each of filters is:

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

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ # of training examples

$$A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ ← # filters in layer l

bias: $n_c^{[l]}$ — $(1, 1, 1, n_c^{[l]})$

Simple Convolutional network example

Example ~~Sov~~ ConvNet

$39 \times 39 \times 3$ $\xrightarrow[\substack{s^{[1]}=1 \\ p^{[1]}=0 \\ n_c^{[1]}=10}]{f^{[1]}=3}$ $37 \times 37 \times 10$ $\xrightarrow[\substack{s^{[2]}=2 \\ p^{[2]}=0 \\ n_c^{[2]}=20}]{f^{[2]}=5}$ $17 \times 17 \times 20$

$n_H^{[0]} = n_W^{[0]} = 39$, $n_c^{[0]} = 3$

$n_H^{[1]} = n_W^{[1]} = 37$, $n_c^{[1]} = 10$

$n_H^{[2]} = n_W^{[2]} = 17$, $n_c^{[2]} = 20$

$\xrightarrow[\substack{s^{[2]}=2 \\ n_c^{[3]}=40}]{f^{[3]}=5}$ $7 \times 7 \times 40$ $\longrightarrow$ softmax $\longrightarrow \hat{y}$

$n_H^{[3]} = n_W^{[3]} = 7$, $n_c^{[3]} = 40$

$1960 \rightarrow 1$

$n_H, n_W \downarrow$

$n_c \uparrow$

Types of layer

Convolution (CONV)

Pooling (POOL)

Fully Connected (FC)

Pooling layers

Max pooling: take max of numbers in a region

$4\times4 \xrightarrow[f=2,\ s=2]{} 2\times2 \rightarrow$ parameters do not change in gradient decent

$5\times5\times2 \xrightarrow[f=3,\ s=1]{} 3\times3\times2$

Average pooling: take average of numbers in a region

note: max pooling is used more often than averge pooling

Hyperparameters

f, s, max or average pooling

No parameters to learn

$n_H \times n_W \times n_C$

$\downarrow$

$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \times n_C$

Neural network example (LeNet-5)

$$\underbrace{32\times32\times3 \xrightarrow[\substack{f=5\\ s=1\\ n_C=6}]{} \overset{\text{Conv 1}}{28\times28\times6} \xrightarrow[\substack{f=2\\ s=2}]{\text{maxpool}} \overset{\text{Pool 1}}{14\times14\times6}}_{\text{layer 1}} \underbrace{\xrightarrow[\substack{f=5\\ s=1\\ n_C=16}]{} \overset{\text{Conv 2}}{10\times10\times16} \xrightarrow[\substack{f=2\\ s=2}]{\text{maxpool}} \overset{\text{Pool 2}}{5\times5\times16}}_{\text{layer 2}} = 400 \rightarrow \overset{\text{FC 3}}{120} \rightarrow \overset{\text{FC 4}}{84} \rightarrow 1$$

FC3: $W^{[3]}$ (120, 400), $b^{[3]}$ (120)

FC4: $W^{[4]}$ (84, 120), $b^{[4]}$ (84)

softmax (10 outputs)

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

Shape

	Activation shape	Activation Size	# parameters
Input	(32, 32, 3)	32×32×3 → 3072 $a^{[0]}$	0
Conv1 (f=5, s=1)	$\frac{32+0-5}{1}+1$ → (28, 28, 8)	28×28×8 → 6272	5×5×8 + 8 → 208
POOL1	28/2 → (14, 14, 8)	14×14×8 → 1568	no parameters to learn → 0
Conv2 (f=5, s=1)	$\frac{14+0-5}{1}+1$ → (10, 10, 16)	10×10×16 → 1600	5×5×10 + 16 → 416
POOL2	10/2 → 5×5×16	5×5×16 → 400	no parameters to learn → 0
FC3	(120, 1)	120	400×120+1 → 48001
FC4	(84, 1)	84	120×84+1 → 10081
Softmax	(10, 1)	10	84×10+1 → 841

Why convolutions

* Small ~~set~~ num of parameters
  - parameter sharing
  - sparsity of connections
* translation invariance

Put it together

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

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

optimize parameters to reduce J

Classic networks:

- LeNet-5
- AlexNet
- VGG

ResNet (152)

Inception

Case studies
Classic networks

LeNet-5

$$\underset{32\times32\times1}{\boxed{7}} \xrightarrow[s=1]{5\times5} 28\times28\times6 \xrightarrow[f=2,\ s=2]{\text{avg pool}} 14\times14\times6 \xrightarrow[s=1]{5\times5} 10\times10\times16$$

$$\xrightarrow[f=2,\ s=2]{\text{avg pool}} \underbrace{5\times5\times16}_{400} \xrightarrow{FC} 120 \xrightarrow{FC} 84 \rightarrow \underset{\text{softmax (10)}}{\hat{y}}$$

60k parameters

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

conv pool conv pool fc fc output

AlexNet

$$227 \times 227 \times 3 \xrightarrow[\substack{11\times 11 \\ s=4}]{} 55 \times 55 \times 96 \xrightarrow[\substack{3\times 3 \\ s=2}]{\text{max-pool}} 27 \times 27 \times 96 \xrightarrow[\text{same}]{5\times 5} 27 \times 27 \times 256$$

$$\xrightarrow[\substack{3\times 3 \\ s=2}]{\text{max-pool}} 13 \times 13 \times 256 \xrightarrow[\text{same}]{3\times 3} 13 \times 13 \times 384 \xrightarrow[\text{same}]{3\times 3} 13 \times 13 \times 384 \xrightarrow[\text{same}]{3\times 3} 13 \times 13 \times 256$$

$$\xrightarrow[\substack{3\times 3 \\ s=2}]{\text{max-pool}} \underbrace{6 \times 6 \times 256}_{9216} = 9216 \xrightarrow{FC} 4096 \xrightarrow{FC} 4096 \xrightarrow{\text{softmax}} 1000$$

- Similar to leNet, but much bigger: 60 m parameters

VGG-16

CONV = 3x3 filters, s=1, same   MAX-POOL = 2x2, s=2

$$224 \times 224 \times 3 \xrightarrow[\times 2]{\text{CONV 64}} 224 \times 224 \times 64 \xrightarrow[\text{POOL}]{} 112 \times 112 \times 64 \xrightarrow[\times 2]{\text{CONV 128}} 112 \times 112 \times 128$$

$$\xrightarrow[\text{POOL}]{} 56 \times 56 \times 128 \xrightarrow[\times 3]{\text{CONV 256}} 56 \times 56 \times 256 \xrightarrow[\text{POOL}]{} 28 \times 28 \times 256 \xrightarrow[\times 3]{\text{CONV 512}} 28 \times 28 \times 512$$

$$\xrightarrow[\text{POOL}]{} 14 \times 14 \times 512 \xrightarrow[\times 3]{\text{CONV 512}} 14 \times 14 \times 512 \xrightarrow[\text{POOL}]{} 7 \times 7 \times 512 \rightarrow \underset{4096}{FC} \rightarrow \underset{4096}{FC} \rightarrow \underset{1000}{\text{softmax}}$$

138 m parameters, 16 ~~weightet~~ weighted layers

Residual block (help train very deep network)

"short cut"/skip connection

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

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

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

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

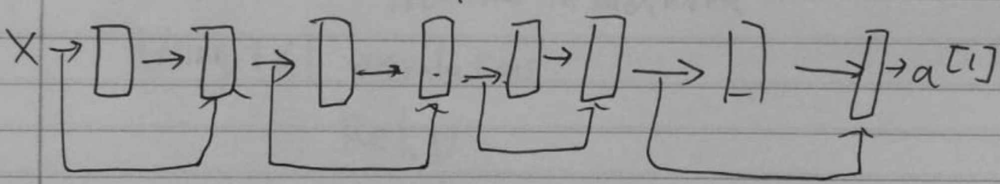
$a^{[l+2]} = g(z^{[l+2]})$

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

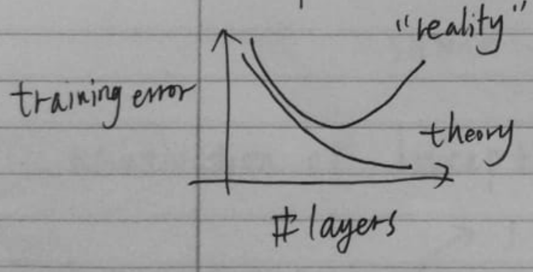
main

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

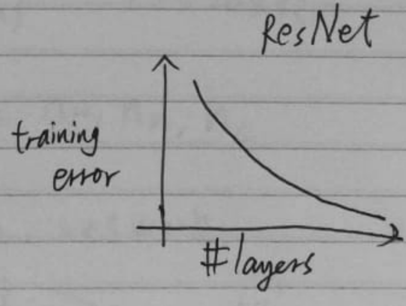
"plain network"



Plain



ResNet



Why Resnet works?

can be also $g(z^{[l+2]} + b^{[l+2]} + W_s a^{[l]})$

help adjust dimension

$X \rightarrow$ Big NN $\rightarrow a^{[l]}$

$X \rightarrow$ Big NN $\xrightarrow{a^{[l]}}$ (8 8 8) $\rightarrow$ (o o 8) $\rightarrow a^{[l+2]}$

ReLU $a \geq 0$

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

same dimension

(can be regularized)

make sure doesn't hurt performance

1x1 convolution

$6 \times 6 * \boxed{2}\ 1 \times 1 = 6 \times 6$

$6 \times 6 \times 32 * 1 \times 1 \times 32 = 6 \times 6 \times \#\text{filters}$

ReLU

$32 \longrightarrow \#\text{filters}$

$n_c^{[l+1]}$

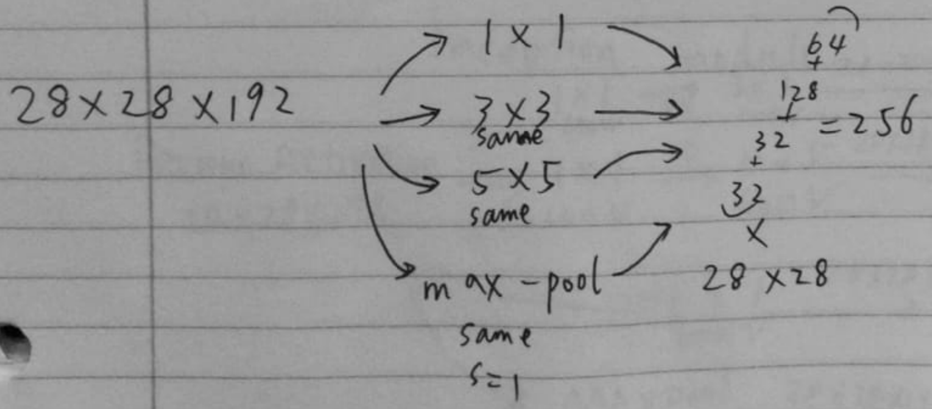
Network in Network

Using 1x1 convolutions

$28 \times 28 \times 192 \xrightarrow[\text{conv } 1\times1,\ 32]{\text{ReLU}} 28 \times 28 \times 32$

shrink to n_H, n_W, n_C

Motivation of Inception network



The problem of computational cost

$28\times28\times192 \xrightarrow[\substack{5\times5,\\ \text{same}\\ 32}]{\text{conv}} 28\times28\times32$

32 filter

Computation: $\underline{28\times28\times32} \times \underline{5\times5\times192} = 120M$

Using 1×1 convolution

"bottleneck layer"

$28\times28\times192 \xrightarrow[\substack{1\times1,\\ 16,\\ 1\times1\times192}]{\text{conv}} 28\times28\times16 \xrightarrow[\substack{5\times5,\\ 32,\\ 5\times5\times16}]{\text{conv}} 28\times28\times32$

Computation: $28\times28\times16\times192 = 2.4m$

$+ \quad 28\times28\times32\times5\times5\times16 = 10.0m$ $\quad = 12.4m$

Inception module

Previous Activation $28\times28\times192$

$\to$ 1×1 conv $\xrightarrow{96}$ 3×3 conv $\xrightarrow{28\times28\times128}$

$\to$ 1×1 conv $\xrightarrow{16}$ 5×5 conv $\xrightarrow{28\times28\times32}$

$\to$ 1×1 conv $\xrightarrow{28\times28\times64}$

$\to$ MAXpool 3×3, s=1, same $\xrightarrow{28\times28\times192}$ 1×1 conv $\xrightarrow{28\times28\times32}$

$\to$ channel Concat $28\times28\times256$

Googlenet

advice for using ConvNets

1. Using open-source implementation

2. transfer learning

① (If I have small dataset)

$X \rightarrow \boxed{NN} \rightarrow$ ~~softmax $\rightarrow \hat{y}$~~

freeze

trainable Parameters=0

freeze=1

$\rightarrow$ softmax $\rightarrow$ A, B, C

$\downarrow$

(my problem)

② (If I have many pictures)

$X \rightarrow \boxed{NN} \rightarrow$ ~~softmax $\rightarrow \hat{y}$~~

freeze some layers

train some layers (or creat my own hidden layers)

$\rightarrow$ softmax $\rightarrow$ A, B, C

$\downarrow$

(my problem)

③ (If I have very large dataset)

$X \rightarrow \boxed{NN} \rightarrow$ ~~softmax $\rightarrow \hat{y}$~~

train all layers

$\rightarrow$ softmax $\rightarrow$ A, B, C

$\downarrow$

(my problem)

3. Data augmentation

① mirroring

② Random Cropping

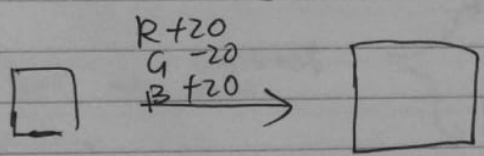
③ Rotation

④ shearing

⑤ Local warping

⋮

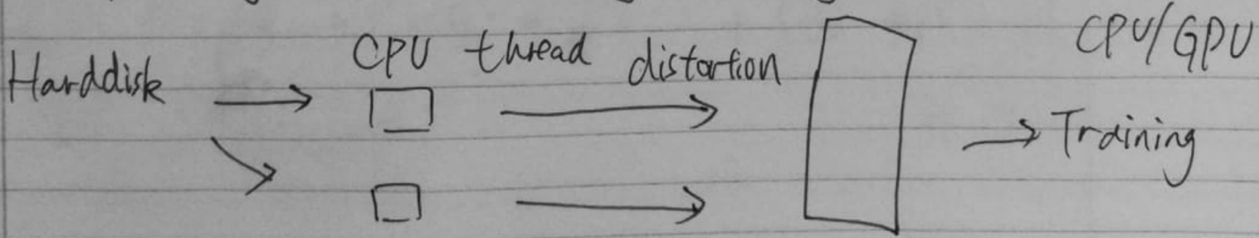
Color shifting



Advanced:

PCA color ~~auf~~ augmentation (from AlexNet Paper)

Implementing distortions during training



Tips for doing well on benchmarks

Ensembling (not used in production)
(Average different models' output)

~~Multip~~

Multi-crop at test time

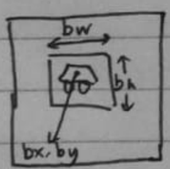
Classification with localization

1. pedestrian
2. car
3. motorcycle
4. background

picture → convnet ~~tova~~ → softmax classlabel (1-4)

→ b_x, b_y, b_h, b_w

bounding box



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$ ← is there any object

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \cdots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

landmark detection

face picture → convnet → features → face?
l_{x1}, l_{y1}
l_{x2}, l_{y2}
⋮
$l_{x64} l_{y64}$ } landmarks

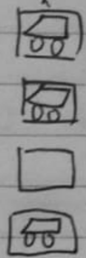
e.g

• black points are landmarks

Object detection

Car detection example

Training set
x y



1
1
0
1

→ convnet

Sliding windows detection

Disadvantage: computational cost, slow

Turning FC layer into convolutional layers

$14 \times 14 \times 3 \xrightarrow[5\times5]{} 10 \times 10 \times 16 \xrightarrow[2\times2]{\text{MAXpool}} 5 \times 5 \times 16 \xrightarrow{\text{FC}} 400 \xrightarrow{\text{FC}} 400 \rightarrow y$ softmax(4)

Change FC to conv layer:

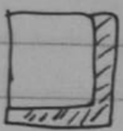
same as FC

$14 \times 14 \times 3 \xrightarrow[5\times5]{} 10 \times 10 \times 16 \xrightarrow[2\times2]{\text{MAX POOL}} 5 \times 5 \times 16 \xrightarrow[5\times5, \boxed{400}]{\text{FC}} 1 \times 1 \times 400 \xrightarrow[1\times1, 400]{\text{FC}} 1 \times 1 \times 400 \xrightarrow[1\times1, 4]{} 1 \times 1 \times 4$

Convolutional implementation of sliding windows

$14\times14\times3 \xrightarrow[5\times5]{} 10\times10\times16 \xrightarrow[2\times2]{\text{MAX POOL}} 5\times5\times16 \xrightarrow[FC]{5\times5} 1\times1\times400 \xrightarrow[1\times1]{FC} 1\times1\times400 \xrightarrow[1\times1]{FC} 1\times1\times4$

add stripe as follows

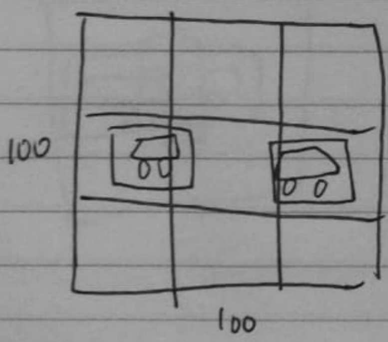


$16\times16\times3 \xrightarrow[5\times5]{} 12\times12\times16 \xrightarrow[2\times2]{\text{MAX POOL}} 6\times6\times16 \xrightarrow[5\times5]{FC} 2\times2\times400 \xrightarrow[1\times1]{FC} 2\times2\times400 \xrightarrow[1\times1]{FC} 2\times2\times4$

disadvantage: bounding box not too accurate

Output accurate bounding box

YOLO algorithm



Labels for traing

For each grid cell

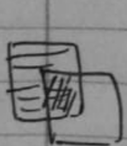
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} \vdots \end{bmatrix} \quad \begin{bmatrix} \vdots \end{bmatrix} \quad \begin{bmatrix} \vdots \end{bmatrix}$$

Target output $3\times3\times8$

Specify the bounding boxes

Evaluating object localization

Intersection over Union (IOU)



$$= \frac{\text{size of intersection}}{\text{size of}}$$

"correct" if $IoU \geq 0.5$

Non-max suppression example

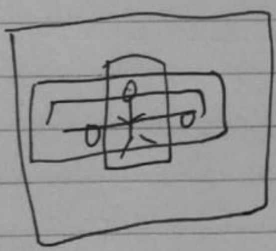
look at the probability of detections, and output max one, suppress lower probability ones

Discard all boxes with $p_c \le 0.6$

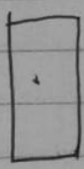
While there are many remaining boxes

- Pick the box with largest p_c
  Output that as a prediction
- Discard any remaining box with $IOU \geq 0.5$ with the box output in the previous step

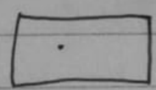
Anchor ~~ch~~ boxes



Anchor box1



Anchor box 2



define $y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_H \\ b_W \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ \vdots \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ $\begin{matrix} \left.\begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix}\right\} \text{anchor box 1} \\ \left.\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right\} \text{anchor box 2} \end{matrix}$

object is assigned to grid cell that contains object's mid point and anchor box which has highest IOU

Region proposal

R-CNN Propose regions

Fast R-CNN Use convolution implementation of sliding window to ~~prope~~ propose regions

Fastter R-CNN Use convolution network to propose regions

Face verification vs. face recognition

Verification

- Input image, name/ID
- Output whether the input image is that of the 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 recognized")

One-shot learning

Learn from one example to recognize the person again

$d(\text{img1}, \text{img2})$ = degree of difference between images

$$\text{if } \left\{ \begin{array}{ll} d \le \tau & \text{"same"} \\ > \tau & \text{"different"} \end{array} \right\} \text{verification}$$

Siamese network

$x^{(1)} \rightarrow$ CNN $\rightarrow$ ~~softmax~~

$f(x^{(1)})$ "encoding of $x^{(1)}$"

$x^{(2)} \rightarrow$ CNN $\rightarrow f(x^{(2)})$ "encoding of $x^{(2)}$"

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

Learning objective

(A) Anchor   (P) Positive   (A) Anchor   (N) negative

d to be ~~large~~ small   d to be large

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

$$\| f(A) - f(P) \|^2 - \| f(A) - f(N) \|^2 + \underset{\uparrow}{\alpha} \le 0$$

margin → make sure $f(img) = \vec{0}$ condition is covered

Loss function

Given 3 images A, P, N

$$\mathcal{L}(A, P, N) = \max(\underbrace{\| f(A) - f(P) \|^2 - \| f(A) - f(N) \|^2 + \alpha}_{>0}, 0)$$

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

Training set: 10k pictures of 1k persons
(Note: there may be more than 1 pics for one person)

Choosing the triplets ~~of~~ A, P, N
if A, P, N are chosen randomly,
$d(A, P) + \alpha \le d(A, N)$ is easily satisfied

Choose triplets that're "hard" to train on
$d(A, P) + \alpha \le d(A, N)$

$$\underset{\downarrow}{d(A, P)} \approx \underset{\uparrow}{d(A, N)}$$

Verification (as binary classification)

Learning the similarity function

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

or

$$\left(\frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}\right)$$

$\uparrow \chi^2$

new $\rightarrow$ CNN $\rightarrow f(x^{(i)})$

database $\rightarrow$ CNN $\rightarrow f(x^{(j)})$

$\rightarrow \sigma \rightarrow \hat{y}$

can be precompute

Neural style transfer

Content (c) Style (s) $\rightarrow$ Generated image (G)

Content (c) style (S) $\rightarrow$ Generated image (G)

Cost function

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

Find the generated image G

1. Initiate G randomly
   G: 100 x 100 x 3
2. Use gradient descent to minimize J(G)

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

Content cost function

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

~~chose~~ choose: layer l: $a^{[l](C)}$, $a^{[l](G)}$ are the activations

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

style: correlation between activations across channels

style matrix

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

$$\begin{array}{ccc} H & W & C \\ \downarrow & \downarrow & \downarrow \\ i, & j, & k \end{array}$$

$G^{[l]}_{kk'}$ ← compare correlations between k and k'

← style image

$$G^{[l](S)}_{kk'} = \sum_i^{n_H^{[l]}} \sum_j^{n_W^{[l]}} a_{ijk}^{[l](S)} a_{ijk'}^{[l](S)}$$

also called "gram matrix"

$$G^{[l](G)}_{kk'} = \sum_i^{n_H^{[l]}} \sum_j^{n_W^{[l]}} a_{ijk}^{[l](G)} a_{ijk'}^{[l](G)}$$

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

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

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

Convolutions in 2D and 1D

2D

$14 \times 14 \times 3 * 5 \times 5 \times 3 \rightarrow 10 \times 10 \times 16$

1D

$14 \times 1 \quad * \quad 5 \times 1 \rightarrow 10 \times 16$

3D

$\overset{H}{14} \times \overset{W}{14} \times \overset{D}{14} \times \overset{n_c}{1} * 5 \times 5 \times 5 \times 1 \rightarrow 10 \times 10 \times 10 \times \overset{n_{c+1}}{16}$

16 filter

n_{c+1}