py ⇒ * (element wise multiplication)

Page No.
Date

CONVOLUTIONAL NEURAL NETWORKS

• computer vision

→ Image classification (0/1)
→ object detection (Identify objects & draw boxes around them)
→ Neural style transfer (Required image = content image + style image)

RGB images (3 channels ⇒ 64×64×3)
Grayscale images (1 channel ⇒ 64×64×1)

• Edge detection

CONVOLUTION

6×6 grayscale image (*) 3×3 filter or kernel ⟶ 4×4 matrix (image)

vertical edge detection:

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

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

Light → Dark

*

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

3 white, 6 grey, 3 black

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Dark-Light-Dark

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	10	10	10

* 1 0 -1 / 1 0 -1 / 1 0 -1 (White, Grey, Black) = 0 -30 -30 0 / 0 -30 -30 0 / 0 -30 -30 0 / 0 -30 -30 0

Dark → Light

Grey - Black = -Grey

Horizontal edge detection =

1	1	1	W
0	0	0	G
-1	-1	-1	B

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

* 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

Alternatives:

- vertical edge

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

⇒ sobel : More weight given to filer central row

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

⇒ scharr filter

* without hand-coding the NN can learn & weights are assigned to detect any angle rotation.

- Padding:

input image (n×n) * filter/kernel (f×f) = output image (n-f+1)×(n-f+1)

Disadvantages: (w/o padding)

→ shrinked output

→ Throw away info from edge

padding input image preserves the original size after convolution

6×6 → ↓padding

8×8 * 3×3 = o/p image 6×6

padding (p=1) ; n=6 ; f=3

∴ o/p = (n+2p-f+1) × (n+2p-f+1)

= (6+2-3+1) × (6+2-3+1)

o/p = 6×6

⊛ valid & same convolutions

(a) valid → No padding (p=0)

(m×n) * (f×f) = (n-f+1) × (n-f+1)

(b) same → pad so that output size is same as input size

padding = p

∴ output = (n+2p-f+1) × (n+2p-f+1)

n+2p-f+1 = n ⇒ $p = \frac{f-1}{2}$

NOTE f is usually **ODD**
→ has a central pixel

- **Strided convolution**

STRIDE = 2

$7 \times 7 * 3 \times 3 = 3 \times 3$

input image | kernel | output image

$n \times n$ = input image
p = padding
s = stride
$f \times f$ = kernel

$\frac{7+0-3}{2}+1 = 3$

IMP ∴ output image $= \left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor$

$\lfloor z \rfloor = floor(z)$ ≈ Rounding z down to nearest integer

- Associativity ⇒ $A*(B*C) = (A*B)*C$

- **convolutions over volumes**

eg: convolutions over RGB images

ht ↓ width (filter)

$6 \times 6 \times 3 * 3 \times 3 \times 3 = 4 \times 4$

(image) no. of channels (equal)

3D 3D 2D

→ Filter to detect vertical edges in red channel:

R

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

G

0	0	0
0	0	0
0	0	0

B

0	0	0
0	0	0
0	0	0

→ Detect edges in any colour:

filter =

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

NOTE $(n \times n \times n_c) * (f \times f \times n_c) = (n-f+1 \times n-f+1 \times n_c')$

$6 \times 6 \times 3$ (ch.) $3 \times 3 \times 3$ $4 \times 4 \times 2$

features ← (no. of filters)

∴ case for padding = 0 & stride = 1

- **Build one layer of CNN**

$6 \times 6 \times 3$ → $* \; 3 \times 3 \times 3$ → ReLU$(4 \times 4 + b_1)$ → 4×4
input ($a^{[0]}$ or X) → $* \; 3 \times 3 \times 3$ → ReLU$(4 \times 4 + b_2)$ → 4×4

$W^{[1]}$ $\quad g(z^{[1]})$ $\quad (4 \times 4 \times 2)$ $a^{[1]}$

REVIEW

forward propagation:

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

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

Q. 10 ~~layers~~ filters that are 3 x 3 x 3 in one layer of a neural network.

∴ no. of parameters = 10 x 28 = 280

(10 = filters, 28 = parameters)

3 x 3 x 3 = 27

↓ +1 (bias)

28

* layer l is convolutional layer:

$f^{[l]}$ = filter size -- f x f in layer l

$p^{[l]}$ = padding ; $n_c^{[l]}$ = no. of filters

$s^{[l]}$ = stride

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ ← from prev. layer

Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ ... $\left\lfloor \frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

Filter: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
(weights)

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

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

Bias: $n_c^{[l]}$ -- $(1, 1, 1, n_c^{[l]})$

• Example of ConvNet

39 x 39 x 3 ⟶ 37 x 37 x 10

$n_H^{[0]} = n_W^{[0]} = 39$

$n_c^{[0]} = 3$

$f^{[1]} = 3$

$s^{[1]} = 1$

$p^{[1]} = 0$

filters = 10

$n_H^{[1]} = n_W^{[1]} = 37$

$n_c^{[1]} = 10$

$f^{[2]} = 5$

$s^{[2]} = 2$

$p^{[2]} = 0$

20 filters

⟶ 17 x 17 x 20

$n_H^{[2]} = n_W^{[2]} = 17$

$n_c^{[2]} = 20$

$f^{[3]} = 5$

$s^{[3]} = 2$

40 filters

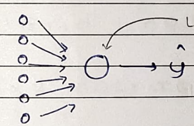
⟶ 7 x 7 x 40

$n_H^{[3]} = 7$

$n_W^{[3]} = 7$

$n_c^{[3]} = 40$

↓ 1960 (flatten / Roll out)



Layers: 1) convolution

2) Pooling

3) Fully connected

Page No.
Date

pooling layer (down sampling) [FIXED FUNCTION]

1) Reduce size of convolution
2) Speed up computation
3) Make system robust

* MAX POOLING → usually doesn't use padding

1	3	2	1
2	9	1	1
1	3	9	3
5	6	1	2

I/p.

→ max pool (2 × 2)

9	2
6	9

Hyperparameters: $f = 2$ ← filter size
$s = 2$ ← stride

⇒ feature detected in any quadrant
→ high number
not detected
→ low number

* Has hyperparameters but no parameters to learn (fixed computation, gradient descent doesn't change anything)
(NO parameters that backprop. will adapt through max pooling)

$$\text{output size} = \frac{n + 2p - f + 1}{s}$$

3-D: $5 \times 5 \times n_c \xrightarrow{\text{m.p.}} 3 \times 3 \times n_c$
max pooling done independently on each of the n_c channels

* AVERAGE POOLING

I/p → average pooling →

3.75	1.25
4	2

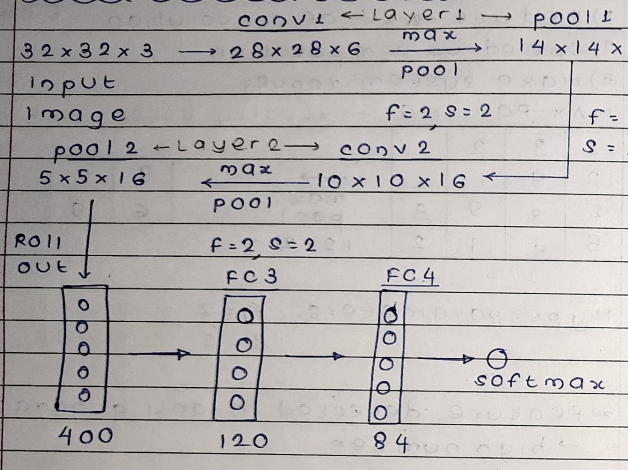
parameters = (9+1) × 5 = 50

one filter (3×3) = 9 no of filters = 5

Bias = 1 per filter

Page No.
Date

Neural network example



AS layers progress: $n_H, n_W \downarrow$ & $n_C \uparrow$

⇒ conv - pool - conv - pool - FC - FC - FC - softmax

	Activation shape	Activation size	# parameters
Input	(32, 32, 3)	3072	0
conv1 (f=5, s=1)	(28, 28, 6)	4704	
pool1	(14, 14, 6)	1176	0
conv2 (f=5, s=1)	(10, 10, 16)	1600	
pool2	(5, 5, 16)	400	0
FC3	(120, 1)	120	
FC4	(84, 1)	84	
softmax	(10, 1)	10	

Page No.
Date

Why convolutions?

Advantages of using convolutions over just fully connected layers:

1) Parameter sharing (feature detector used)
2) Sparsity of connections (each layer → each output depends only on a small number of inputs)

* Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})$

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

→ use gradient descent to optimize parameters to reduce J.