7-12-2021

week#15 lecture#20 Tuesday

Input → Conv2D() → relu → Conv2D() → relu →

flatten → fc → relu → fc

overall scores

To reduce height/width of image.

10x10x3

either use dilation or size to reduce size of image

but Problem in stride is that it doesn't cover whole image

so use stride = 1

Padding → $\frac{f-1}{2}$

input size = output size

Pooling.

To reduce feature Map size

eg

0	5	2	3
0	1	1	9
5	1	6	2
0	2	1	3

feature

Concept of Max Pooling.

e.g 2x2 Pooling

Take max value from | 0 | 5 | / | 0 | 1 | = 5

Take max value from | 2 | 3 | / | 1 | 9 | = 9

Take " " " | 5 | 1 | / | 0 | 2 | = 5

" " " | 6 | 2 | / | 1 | 3 | = 6

5	9
5	6

Almost like down sampling the image

Translation in variance

Size reduced $\frac{3}{4^{th}}$ Pooke size $= \frac{1}{4^{th}}$

Pooling different P instead of ~~taking~~ max we take L2 or L1.

It tells us which pixels contribute most.

Usually after 2 Conv2D() layers we use Pooling.

Stride for Pooling is 2 in this case. ∴ even number input ~~should~~ image size should be ensured

Pattern

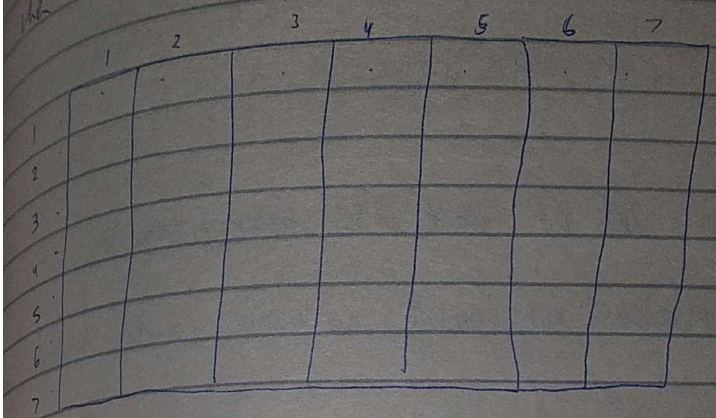
Input → [[Conv2D() → relu] * M → Pooling] * N → [fc → relu] * K → fc → output layer

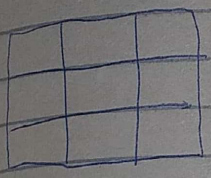
More filters Smaller filter 3x3 x3 = 27 each filter in separate layers.
Less filters, bigger filter. 7x7 = 49 could miss smaller details.

Which is more preferable

What smaller filter does is scan over image again & again but in different layers.

which is equivalent to that one big filter does when it scans the image once.





3x3 filter

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p	q	r	s	t
u	v	w	x	y

Another disadvantage of using big filter is that we lose information of border values. As we downsample image. & we are using multiple Activation function so non-linearity is increased & more complex function can be learned in this way.

No. of weights are also reduced.

e.g. $10 \times 10 \times C \rightarrow 7 \times 7 \times C \times C$

Input — conv layer — $49C^2$

$3 \times 3 \times C \times C = 9C^2 \times 3 = 27C^2$

Smaller filter lets us don't lose border information

as time increases, non-linearity increases, complex function capability increases.

Implementation:

a	b	c
d	e	f
g	h	i
j	k	l

image 4×3

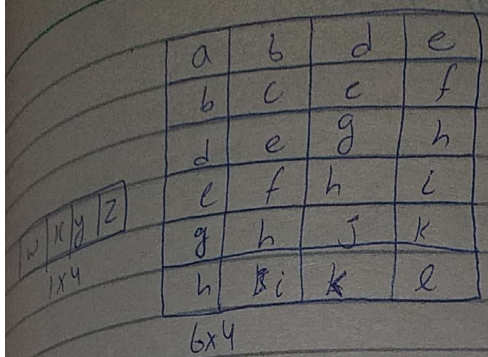
w	x
y	z

filter 2×2

Can we directly multiply these 2 with Stride 1?

No.

what if we do



$f \cdot I_{:c}^{T}$ = feature map^{T}

1x6

if we have 128x4 4x6

128x6 = feature Map

Im2Col function to convert image to columns.

reshaping is done as

image = 4x3

filter = 2x2

So, = 4 - 2 + 1 = 3

= 3 - 2 + 1 = 2

feature map size = 3x2

So 128x6 ⇒ 128x3x2 reshaped

↑ no. of filters feature Map

Memory Requirements of NN:

if input image size

224 x 224 x 3

filter = 3 x 3 x 3 x 64
(No. of filters)

3 x 3 x 3 x 64 = 1728 x 8 = 13824 = 13.824 KB

224 x 224 x 64 = 3211264 x 8 = 2569012 = 25.69 MB
(After Padding)

(Conv) 1 layer Output.

1 image

for batch = 1 layer output = 32 images x 25 MB

= 822 MB

for 1 layer

Transposed Convolution / Deconvolution / Inverse convolution:

from feature Map to Image

we up sample feature map

$O_r = \frac{(I_r - f_r + 2P)}{S} + 1$

on rearranging, we get

$I_r = (O_r - 1) * S - 2P + f_r$

if feature map is

a	b
c	d

2x2

& stride = 1

$P_0 = 0$

$I_r = O_r - 1 + f_r$

if filter is

1	2
3	4

2x2

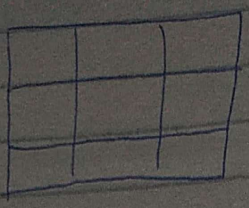
$I_r = 2 - 1 + 2$

$I_r = 3$

similarly for I_c, we get.

$I_c = 3$

Image size 3x3



em/fy image.

Now, for each activation value in feature Mat & multiply with filter & save in Image

a	2a	
3a	4a	

then b

a	2a+b	2b
3a	4a+3b	4b

then c

a	2a+b	2b
3a+c	4a+3b+2c	4b
3c	4c	

then d

a	2a+b	2b
3a+c	4a+3b+2c	4b+2d
3c	4c+3d	4d

224x224 -> 112x112x3 -> 56x56 -> 112x112 -> 224x224