DL spec // course IV // week 1

Convolutional Neural Networks

Computer Vision Example

good quality image 1000×1000×3 = 3 million

$n_h = 1000$

$x_1, x_2, \dots, x_n$ → $\hat{y}$

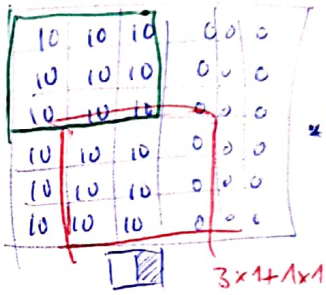
$x \in \mathbb{R}^{3M}$

W (1000, 3 million) = 3 billion

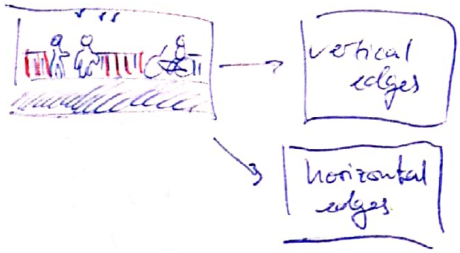
⇒ need convolution to have efficient algorithm

Edge Detection Example

edges → eyes, ears, mouth → faces



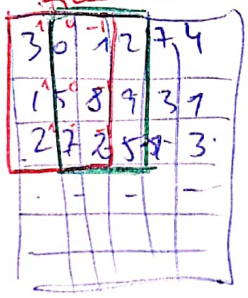
$3\times1+1\times1+2\times1+0\times0+5\times0+7\times0$
$-1\times1-8\times1-2\times1=-5$



vertical edges

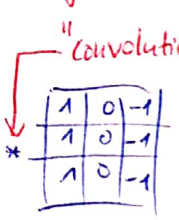
horizontal edges

a vertical edge is a 3×3 region with bright pixels on left and dark pixels on the right

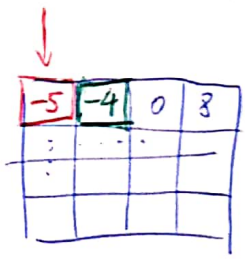


6×6 (grey-scale)

"convolution"



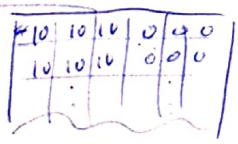
3×3 filter (kernel)



4×4

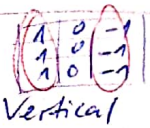
More edge detection

example with



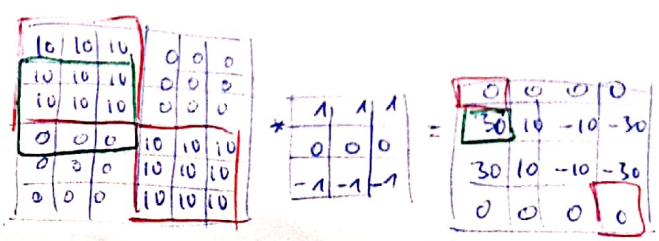
if 10s are negative in input the result is too

Vertical and horizontal edge detection



Vertical

Horizontal



Learning to detect edges

$$\begin{vmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{vmatrix} \rightarrow \begin{vmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{vmatrix}$$

Sobel filter
→ focus on center row
→ more robust

$$\begin{vmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{vmatrix}$$

Scharr filter

just for vertical edges!

filter numbers don't need to be handpicked, can also be learned! using back-prop.

$$* \begin{vmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{vmatrix}$$

↳ can also learn different edges than only vertical and horizontal like 45°, 70°, 73°...
→ Neural Networks learning low-level features like edges...

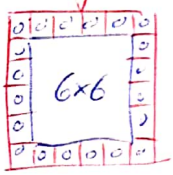
Padding

$6\times6 * 3\times3 = 4\times4$ → because there are only 4×4 possible positions to fit the filter in the input matrix

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

2 downsides:
→ each time with convolution the image shrinks
→ pixel at edge is only processed once while a pixel in the center of input image is "used" more often

} to prevent, you can pad the image



add a boarder of 1 pixel around the edges with 0-value
6×6 → 8×8

now $8\times8 * 3\times3 = 6\times6$
→ managed to preserve initial input size

p = padding = 1

Output: $n+2p-f+1 \times n+2p-f+1$

p=2 would be another boarder around input image

Valid and Same convolutions

"Valid": no padding
$n \times n$ image $* f \times f$ filter $= n-f+1 \times n-f+1$ output

"Same": Pad so that output size is the same as the input size

$n+2p-f+1 \times n+2p-f+1$

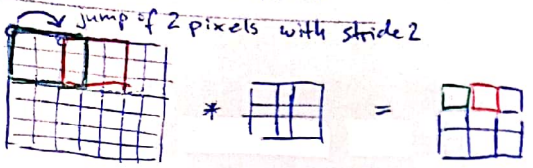
$n+2p-f+1 \overset{!}{=} n \Rightarrow p = \frac{f-1}{2}$

so for 3×3 filter $p = \frac{3-1}{2} = 1$

or for 5×5 filter $p = \frac{5-1}{2} = 2$

↳ f is usually odd
↳ if f was even we'd have unsymmetric padding
↳ odd dimension filter has a central position → benefit for position of filter

Strided Convolutions



$7\times7 \ * \ 3\times3 \rightarrow 3\times3$

stride 2 → take 2 steps

Summary of convolutions

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

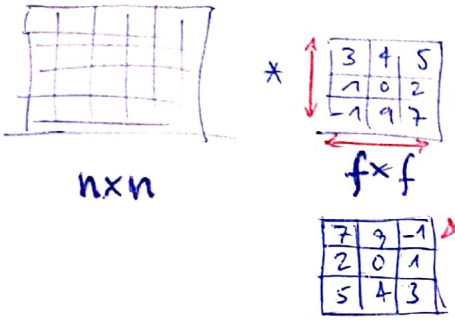
padding p stride s $s=2$

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

$\lfloor \ \rfloor$ is symbol for "floor", so rounding down to next integer

Technical note on cross-correlation vs. convolution

Convolution in math-textbooks:



$= \dots$

actually more interesting for signal processing

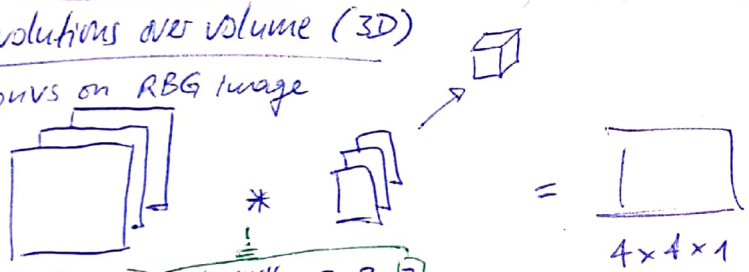
so that "associativity" is valid

$(A * B) * C = A * (B * C)$

↳ doesn't matter for DL

Convolutions over volume (3D)

convs on RBG image



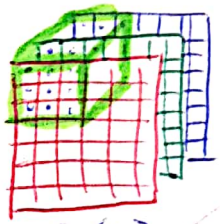
$6\times6\times3 \ * \ 3\times3\times3 = 4\times4\times1$

channels have to match!

RGB

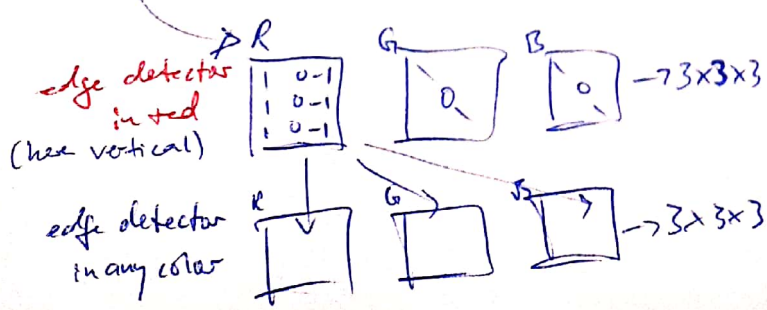
height width #channels

filter height width #channels

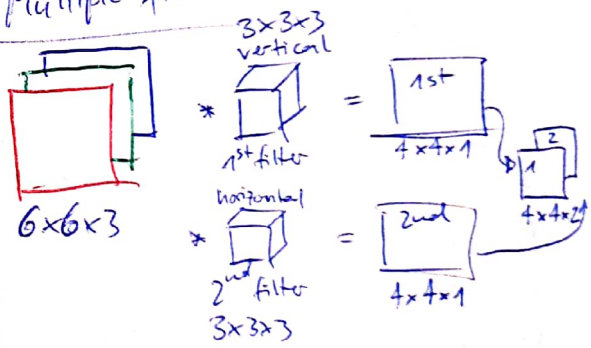


$6\times6\times3 \ * \ 3\times3\times3 = 4\times4\times1$

↳ same shifting of 3D filter as in 2D images...



Multiple filters



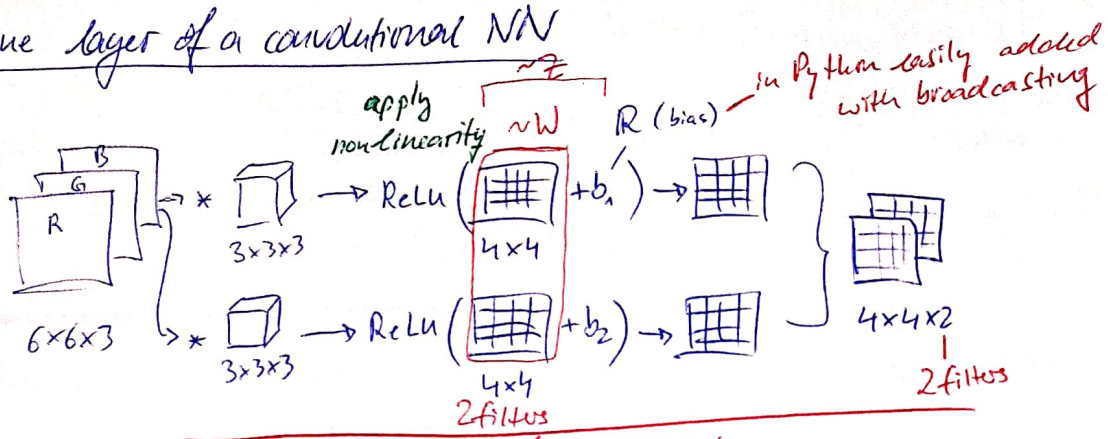
Summary

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

$6\times6\times3 \quad 3\times3\times3 \quad 4\times4\times2$

n_c' = # filters

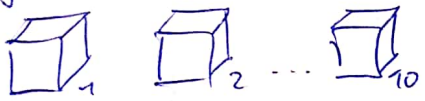
One layer of a convolutional NN



↳ One layer of a CNN

Number of parameters in one layer

If you have 10 filters with 3×3×3 ... how many params in one layer?



3×3×3 . . . .

= 27 + bias

→ 28 params → 280 parameters

(independent of input image size)

Summary of notation

If layer l is a convolution layer:

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

$p^{[l]}$ = padding

$s^{[l]}$ = stride

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

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ (height, width of previous layer)

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

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

Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

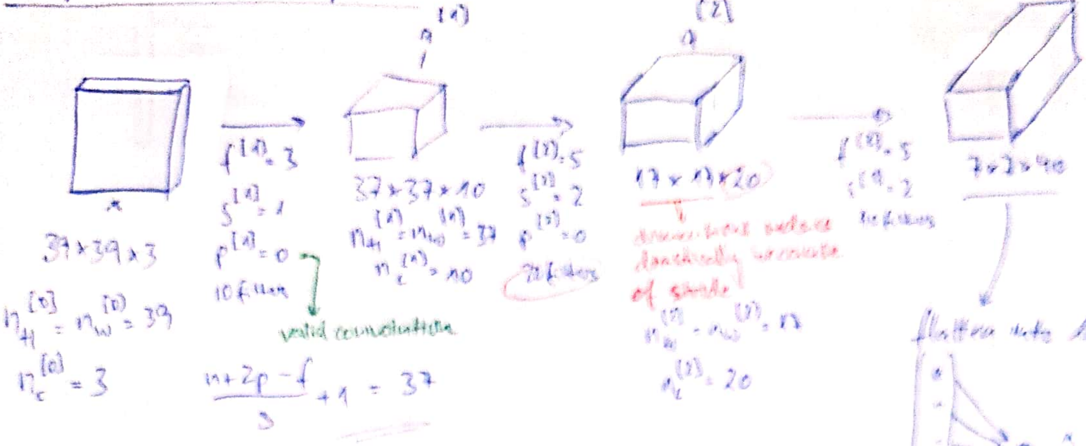
Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

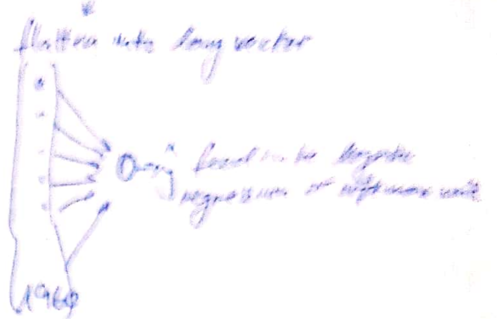
vectorized: $A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

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

bias: $n_c^{[l]} - (1,1,1,n_c^{[l]})$ ← # filters in layer l

Simple CNN example





Hyperparameters: filtersize, stride, padding, # filters

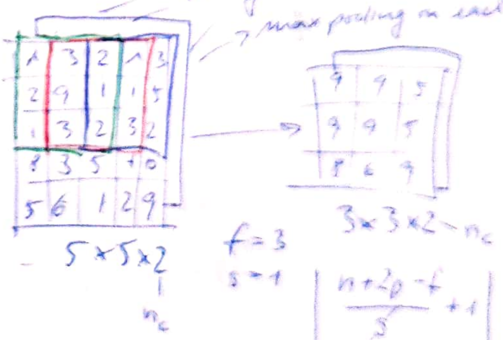
Types of layer in a CNN

- Convolution (CONV)
- Pooling (POOL)
- Fully connected (FC)

Pooling layers

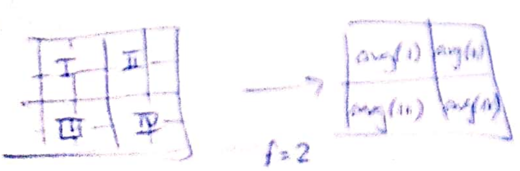
Max Pooling

Hyperparams
filtersize = 2
stride = 2
↳ no params to learn! Fix computation! (with GD)



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

Average Pooling (can be used deeper in a CNN)



f = 2
s = 2

Summary of pooling

Hyperparameters (no parameters to learn)

f filtersize   f=2, s=2
s stride   f=3, s=2

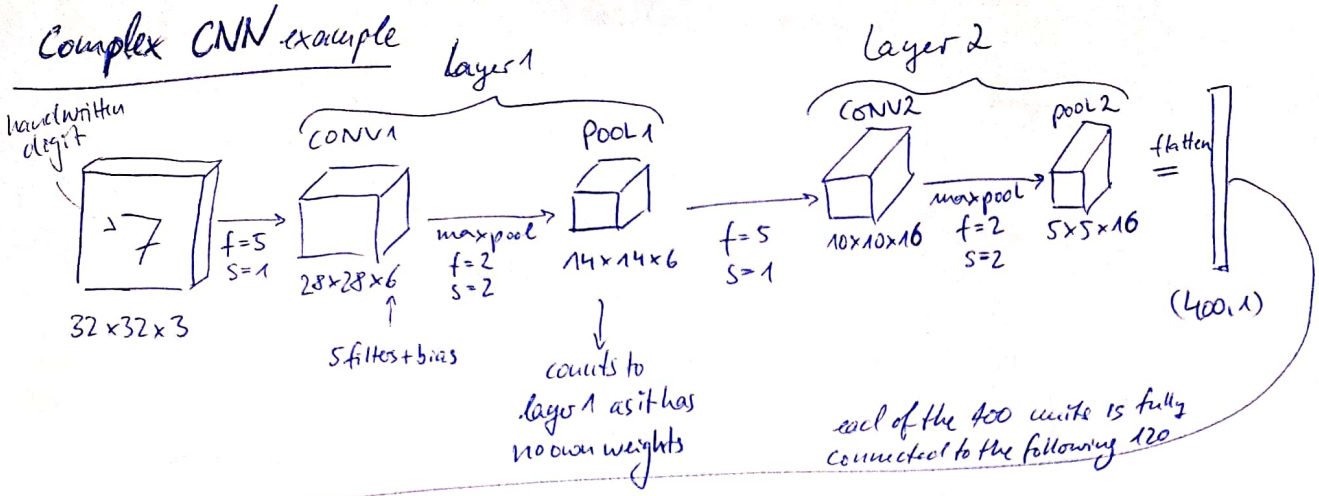
Max or average pooling
(usually no padding)

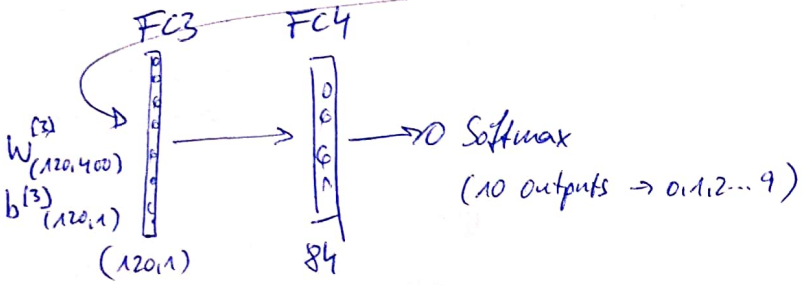
because pooling applied to each channel independently → # of input channels is equal to output channels

$n_H \times n_W \times n_c$

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

Complex CNN example





look for hyperparameters in literature

$n_H, n_W \downarrow$

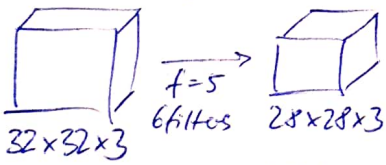
$n_c \uparrow$

usually CONV-POOL-CONV-POOL-FC-FC-FC-SOFTMAX

→ POOL layers have no parameters
→ CONV layers' params increase with depth
→ FC have huge # params!
→ Activation size will decrease with depth but should not drop quickly → performance issues

Why Convolutions?

2 main advantages over fully connected layers
→ parameter sharing
→ sparsity of connections



=3072 =4704

as NN the connections would lead to a weight matrix of 14 million parameters!
→ can easily become too large

filter 5×5 → 25+bias = 26 params

$(5 \times 5 \times 3 + 1) = 456$

Parameter Sharing: A feature detector (vertical edges) that is useful in one part of the image is probably useful in another part of the image
→ apply same filter to different regions

Sparsity of connections: In each layer, each output value depends only on a small number of inputs

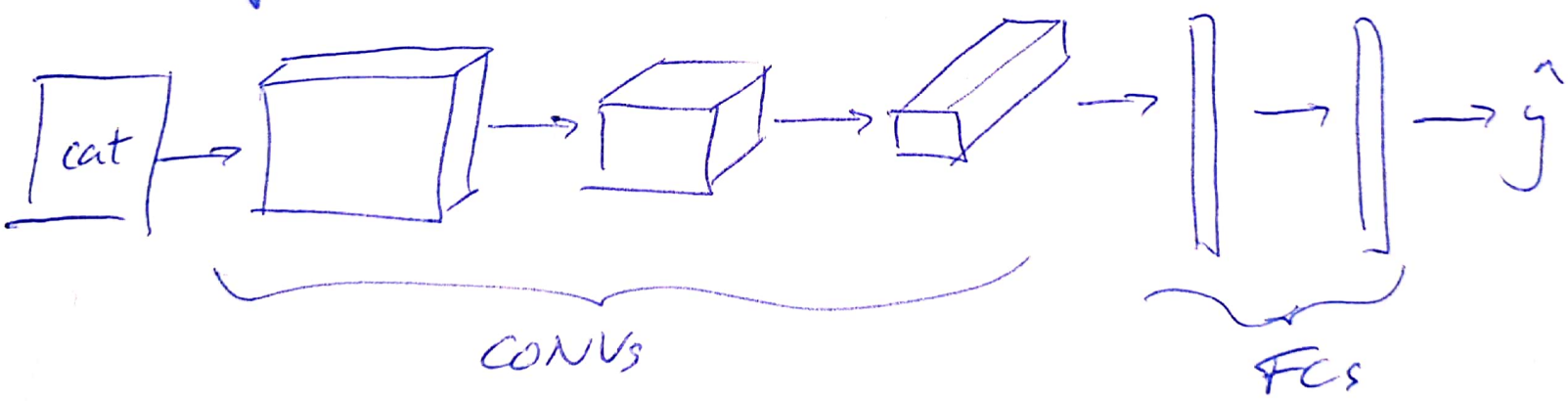
CNN → translation invariance
↳ a picture of a cat shifted a few pixels to the right still looks like a cat
↳ the fact that we apply the same filter adds to robustness

DL spec // courte IV // week 1

Putting it together

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

W, b — randomly initialize



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

Use gradient descent (or RMS, adam) to optimize parameters to reduce J