CONV NETS - SETTING THE BASE

Edge Detection, Convolution

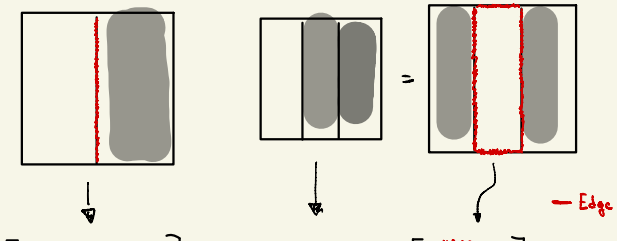
convolution (tf. nn. conv2d)

$$\begin{bmatrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

6x6 (grey scale img) * 3x3 (filter) = 4x4

→ Each element is the result of the element wise product between the img and the filter shifted across the matrix

Vertical Edge Detector



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

6 x 6 [n x n] * 3 x 3 [f x f] = 4 x 4 [n-f+1 x n-f+1]

- Light -> Dark, Dark -> Light edges
- Vertical vs Horizontal edge
- Different types of filter (ie Sobel filter)
- We can treat the 9 numbers in the filter as parameters and learn the values for those. This allows to learn different types of edges
- Valid vs Same convolution
- Strided convolution: skipping the filter by skipping steps with a stride of ie: 2

→ Padding p, Stride s=2: $\left[\frac{n+2p-f}{s}+1 \times \frac{n+2p-f}{s}+1\right]$

- RGB images: (2D)

img 6x6x3 * filter 3x3x3 = img 4x4

channels ⟷ must be the same

- We can even apply multiple filters to a 3D img:

$n \times n \times n_c * n \times n \times n_c =$

$= n-f+1 \times n-f+1 \times n'_c$ → number of filters

Padding

- Every time you apply convolution, your img shrinks
- Edge pixels are used much less than other pixels

} Cons of convolution

- if padding with 1 pixel: $p=1 \rightarrow [n+2p-f+1 \times n+2p-f+1]$

→ We can pad the img before applying convolution: adding a border to the img of pixels with 0

Example of layer

(4x4 -> n-f+1 -> 6-3+1 -> 4x4)

6x6x3

* 3x3x3 (w[1]) -> Relu(4x4 + b1) (w[1] a[0]) -> 4x4 ($z^{[1]}$)

* 3x3x3 -> Relu(4x4 + b2) -> 4x4

→ 4x4x2 ($a^{[1]}$) (# of filters applied)

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

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

- 10 filters 3x3x3 in 1 layer, # of params:

3x3x3 + bias = 28 params | 1 filter ×10 → 280 params

Notation →

Notation

Convolution layer l:

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

$p^{[l]}$ = padding size

$s^{[l]}$ = stride size

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

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

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

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

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

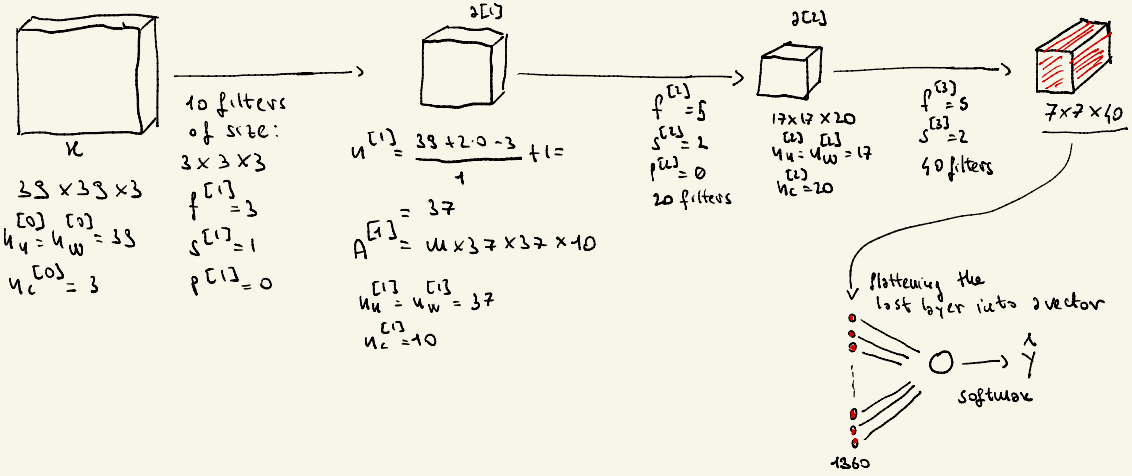
→ Activations: $A^{[l]} \to m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

→ Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ (number of filters in layer l)

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

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

m = # of examples



[Type of layers in a conv net]

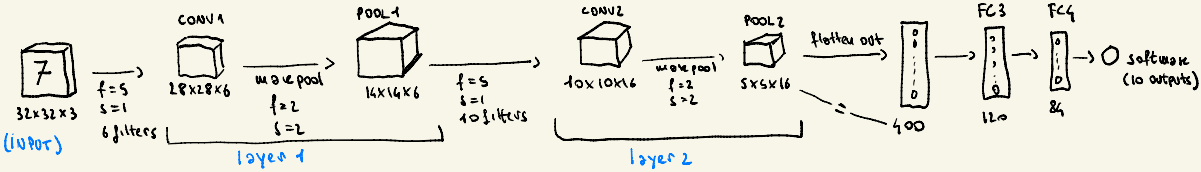
1) Convolution (CONV) 2) Pooling (POOL) 3) Fully Connected (FC)

2) Pooling

- Max pooling: $\begin{bmatrix} 4\times4 \end{bmatrix} \xrightarrow{\text{Report only the max value}} \begin{bmatrix} 2\times2 \end{bmatrix}$
  - it helps detecting a feature in the different quadrants where it's applied
  - it has no parameters to learn, just hyperparameters (f = size, s = stride)
- Average pooling: Report only the avg value. It's less used

[CNN Example]

→ Recognition of hand digits



Network Size

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{[0]}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

[Why Convolutions]

⇒ Less parameters to learn if compared to FC layers

A) Parameter sharing: a feature detector that's useful in one part of the img is probably useful in another part of the img

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

A + B → Training with smaller training sets
→ Less prone to overfitting
→ Good at capture translation in variance