ML & Opt Lecture 16

Convolutional Neural Networks (CNNs)

- convolutional filters
- CNN architecture
- LeNet 5

Why CNNs

- appropriate for computer vision tasks:
  - bounding boxes for objects
  - background subtraction
  - image classification

  (also used for problems from other domains)

- key to good performance: choice of feature map

  $\phi : \mathbb{R}^{k \times k} \to \mathbb{R}^D$ (implemented as a NN)

  these feature maps can be learned on one dataset and used on others

- so far in this class, to do image classification:
  - ϕ from flattening image into a vector
  - ϕ from autoencoder (use encoder as feature map)
  - ϕ from MLP

then use logistic regression

CNNs: another, biologically inspired, approach to learning good feature maps for vision problems

use convolutional filters

1	2	1	0	0
2	1	1	2	0
0	0	1	2	1
2	1	1	1	0
0	1	2	0	2

I

*

-1	0	-1
0	4	0
-1	0	-1

K

=

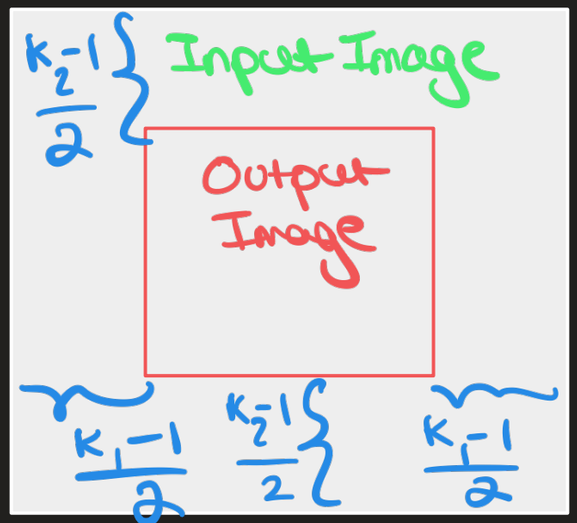
1	0	5
-6	-1	6
1	1	-2

I * K

- range may be different
- I * K is smaller than I

our basic CNN layer takes as input an image I, convolves it with filter K, and returns $\sigma(I * K + b)$

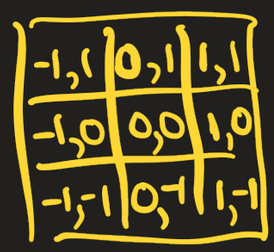
Note that the filters always have size $K_1 \times K_2$ where K_1 & K_2 are odd



(d_1-1, d_2-1)

If the input image is size $d_1 \times d_2$ > the output inge is size

$$d_1-(K_1-1) \times d_2-(K_2-1)$$



indices into K

(0,0)

Formula for Convolutions

$$(I * K)_{i,j} = \sum_{c=0}^{K_1-1} \sum_{r=0}^{K_2-1} K_{cr} I_{i+c,j+r}$$

Idea of CNNs: use convolutional filters to build our nonlinear feature map

Q: Why? A: it has very desirable <u>inductive biases</u>

— we expect that vision is translational invariant

— intuition that image features should be local

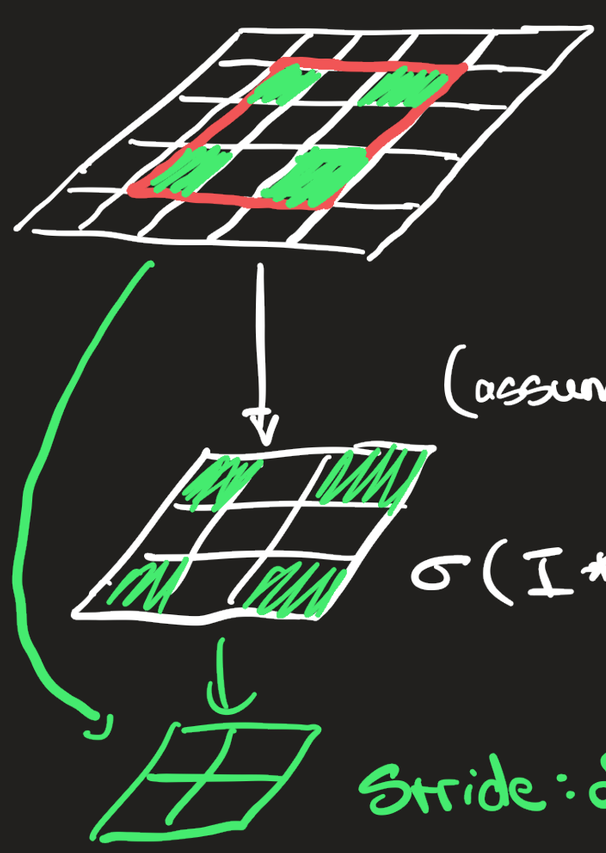
low-level!

- cheap compared to MLP layer

  ($k_1 \cdot k_2$ parameters vs $(d_1 - k_1 + 1)(d_2 - k_2 + 1) \times d_1 d_2$)

- easier to optimize b/c parameter sharing

Convolutional layers



Input: $d_1 \times d_2$

Filter: $k_1 \times k_2$

Output: $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1)$

(Stride: 1)

(assuming $k_1 = k_2 = 3$)

$$\frac{d_1 - k_1}{s} + 1 \times \frac{d_1 - k_2}{s} + 1$$

$\sigma(I * K)$ learn K with backprop

Stride: 2 in each direction.

In general, with a stride of $s_1 \times s_2$ we evaluate the filter at x-intervals of s_1 and y-intervals of s_2

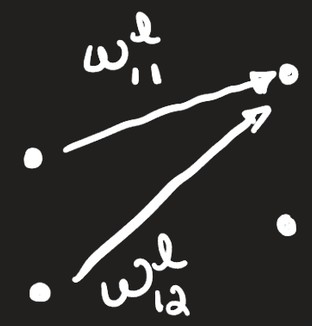
- Equivalent to computing $I * K$ and then throwing away the unused outputs

Multichannel Images

- arise as RGB images or if we use multiple convolutional filters in the previous layer

Given an n channel image $\{I_1, \cdots, I_m\}$, one convolutional layer takes the form $\sigma(I_1 * K_1 + I_2 * K_2 + \cdots + I_m * K_m + b)$

MLP



$n_{l-1} = 2$

$n_l = 3$

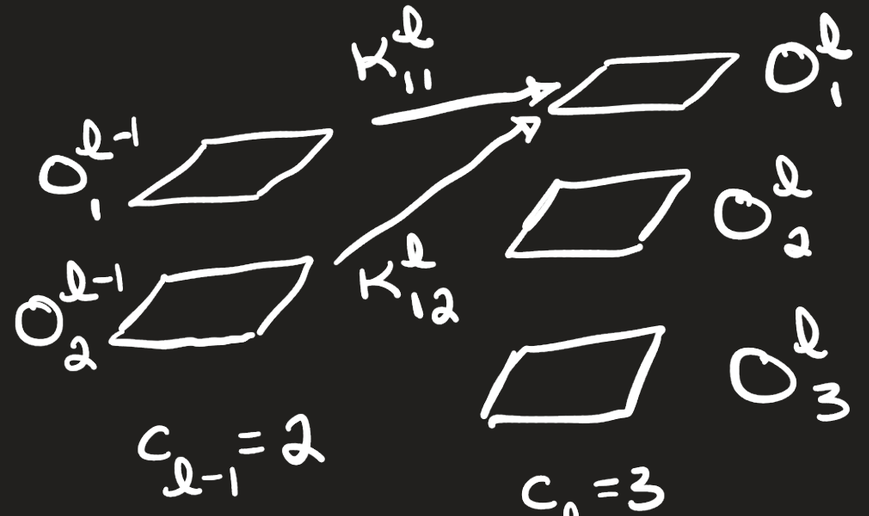
$$o^l = \sigma(W^l o^{l-1} + b^l)$$

There are

$$n_{l-1} n_l + n_l$$

parameters to learn

CNN (convolutional layers)



where each K^l_{ij} is $K_1 \times K_2$

The neurons in the MLP are replaced by channels.
The ith output channel on layer l is computed as

$$O^l_i = \sigma\left(\sum_{j=1}^{c_{l-1}} O^{l-1}_j * K^l_{ij} + b^l_i\right)$$

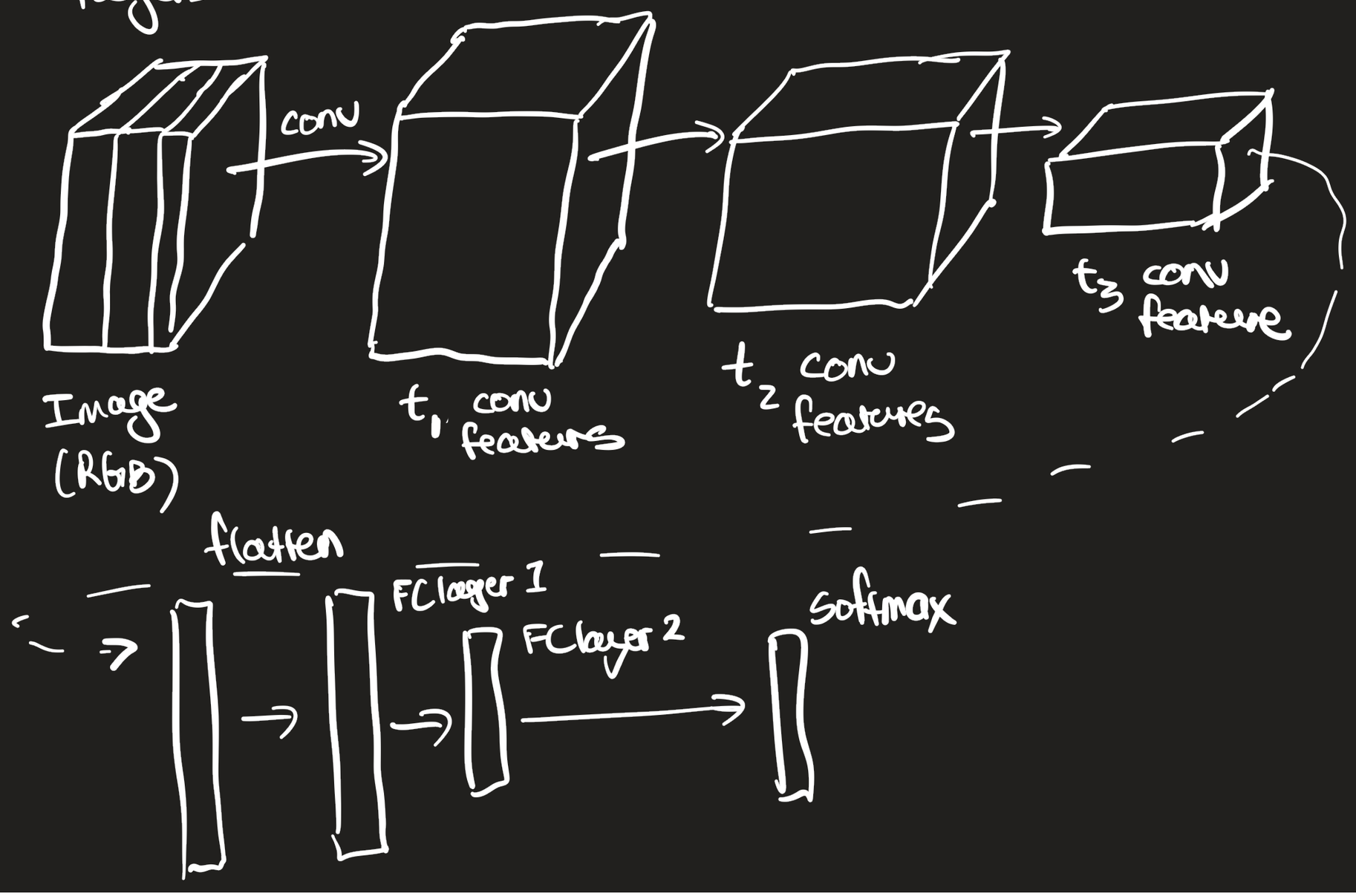
where all the K^l_{ij} are $K_1 \times K_2$ filters and b^l_i are scalars.

There are a total of $c_{l-1} c_l K_1 K_2 + c_l$

learnable parameters for layer l

CNNs:

- combine convolutional layers with fully-connected layers

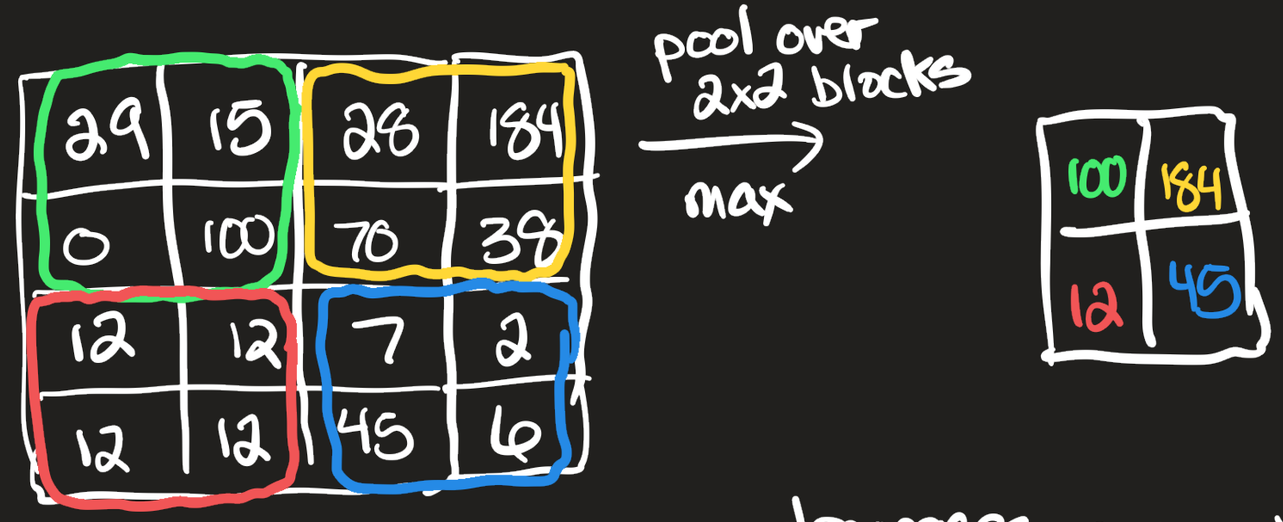


Pooling

Pooling reduces the dimensionality of a convolutional layer by aggregating locally

Typical pooling types:

- max pooling
- average pooling



Pooling's advantages:

- decreases computation downstream
- makes model more robust to shifts in location of features

Example of a CNN LeNet-5 (1997)

5-layer CNN for 10 class classification

Layer	Features	Filter Size	Stride	Activation
Input	1	—	—	—
Conv2D	6	5×5	1	tanh
Avg Pooling	6	2×2	2	—
Conv2D	16	5×5	1	tanh
Avg Pooling	16	2×2	2	—
Conv 2D	120	5×5	1	tanh
FC	84	—	—	tanh
FC	10	—	—	softmax