Convolutional Neural Networks

- appropriate for computer vision tasks
  (where they were originally introduced, but they are useful in other domains as well)

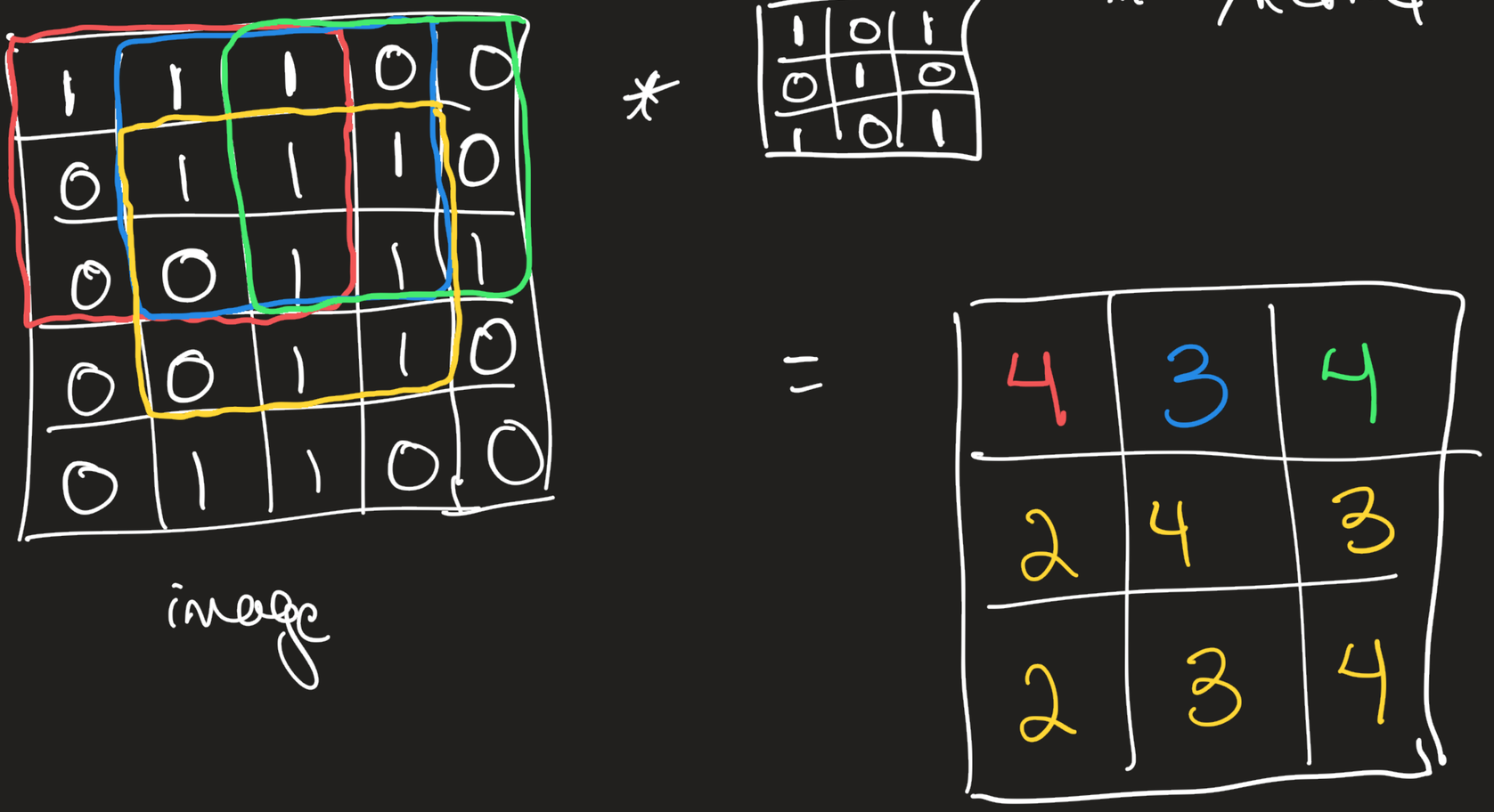
- computer vision tasks:
  - classification
  - segmentation
  - bounding box determination

- key to good performance: choice of our feature map
  $\phi : \mathbb{R}^{d_1 \times d_2} \to \mathbb{R}^D$

- so far in this class:
  - flattened images into vectors
  - kernel approach (implicitly specify ϕ)
  - autoencoders to get ϕ

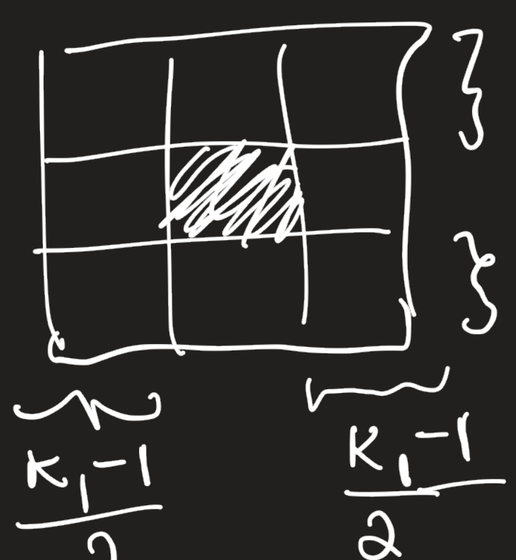
new approach: convolutional filters

treat our image directly as a two-dimensional table of numbers and compute local features:



image

Note that the filters has size $k_1 \times k_2$ and k_1 & k_2 must be odd



I - image K - kernel

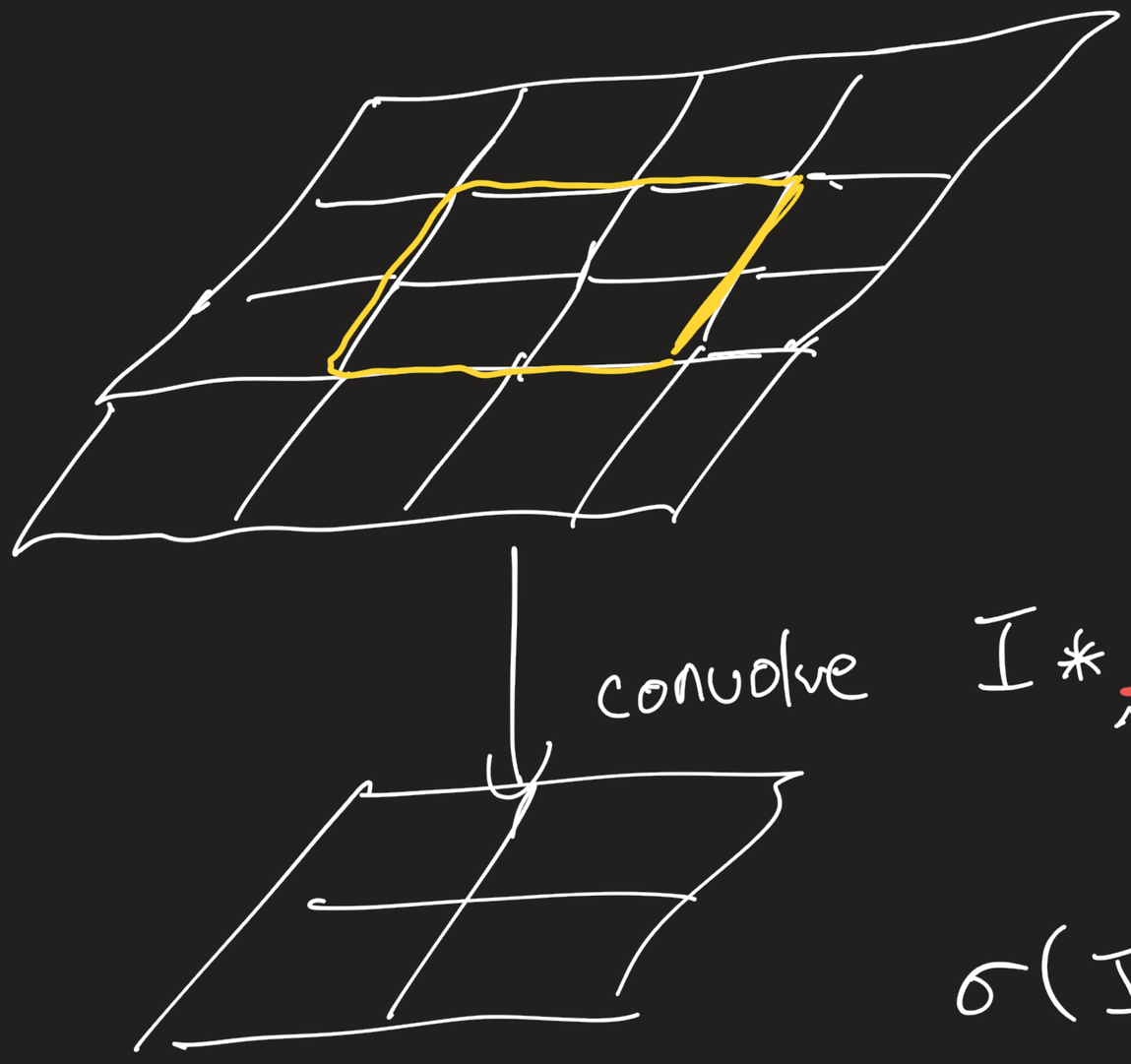
$$(I * K)_{i,j} = \sum_{m=-\frac{k_1}{2}}^{k_1/2} \sum_{n=-\frac{k_2}{2}}^{k_2/2} I_{i-m,\,j-n} \cdot K_{mn}$$

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

Why?

- intuition that image features should be local
- turns out to be cheap compared to a fully connected layers
- parameter sharing which makes learning better behaved

Convolutional Layer:



Input image

K - 3×3 filer

convolve $I * K$

the preactivation

$\sigma(I * K)$ output from the convolutional layer

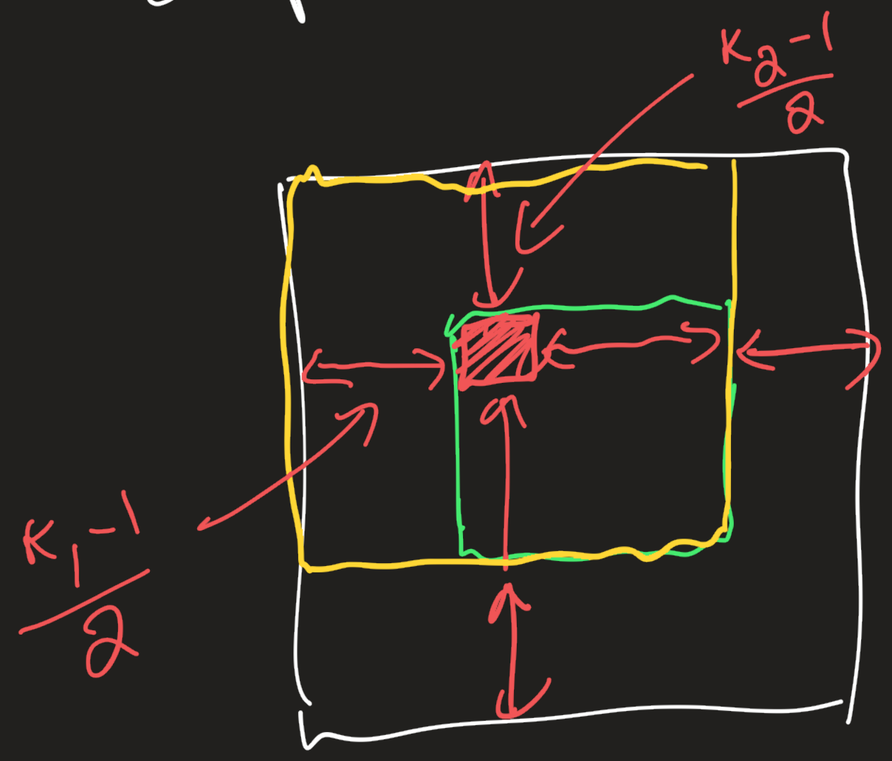
K is the parameter for this convolutional layer

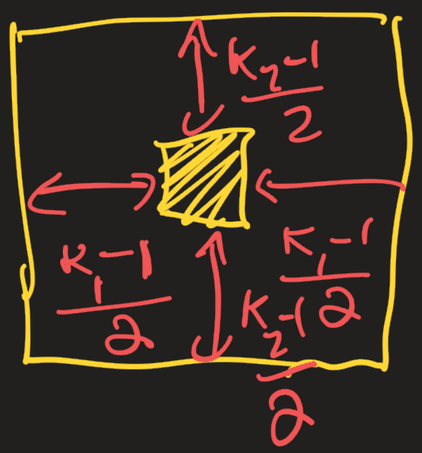
Convolutional Layer

Input: $d_1 \times d_2$

Filter: $k_1 \times k_2$

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





$k_1 \times k_2$ filter

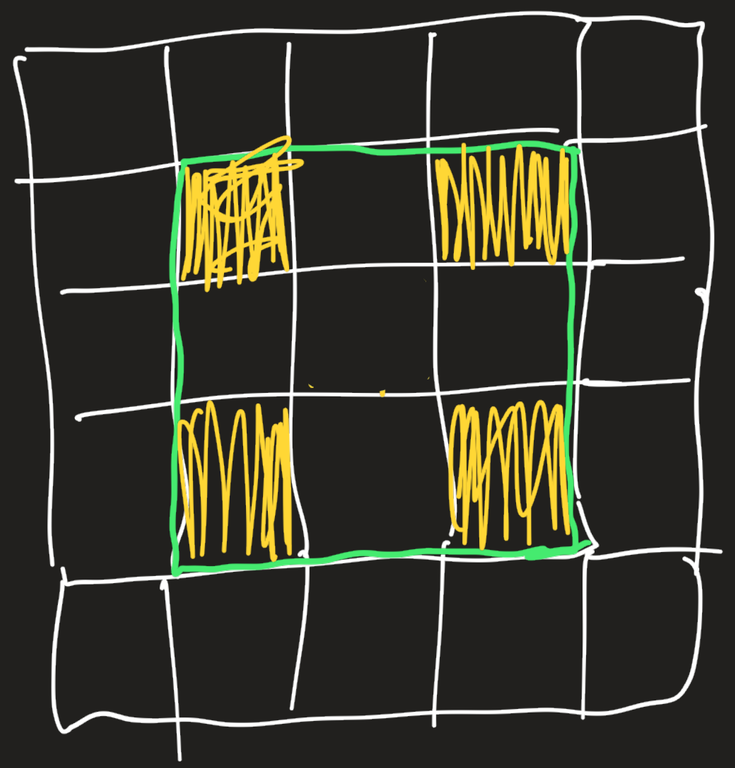
Convolutional Layers w/ Stride

Input: $d_1 \times d_2$

Kernel: $k_1 \times k_2$

Stride: $s_1 \times s_2$

- evaluate the filter at x-intervals of s_1 and y intervals of s_2
- equivalent to taking $I * K$ and sampling in the x dimension each s_1 pixels and in the y dimension each s_2 pixels



K - 3×3

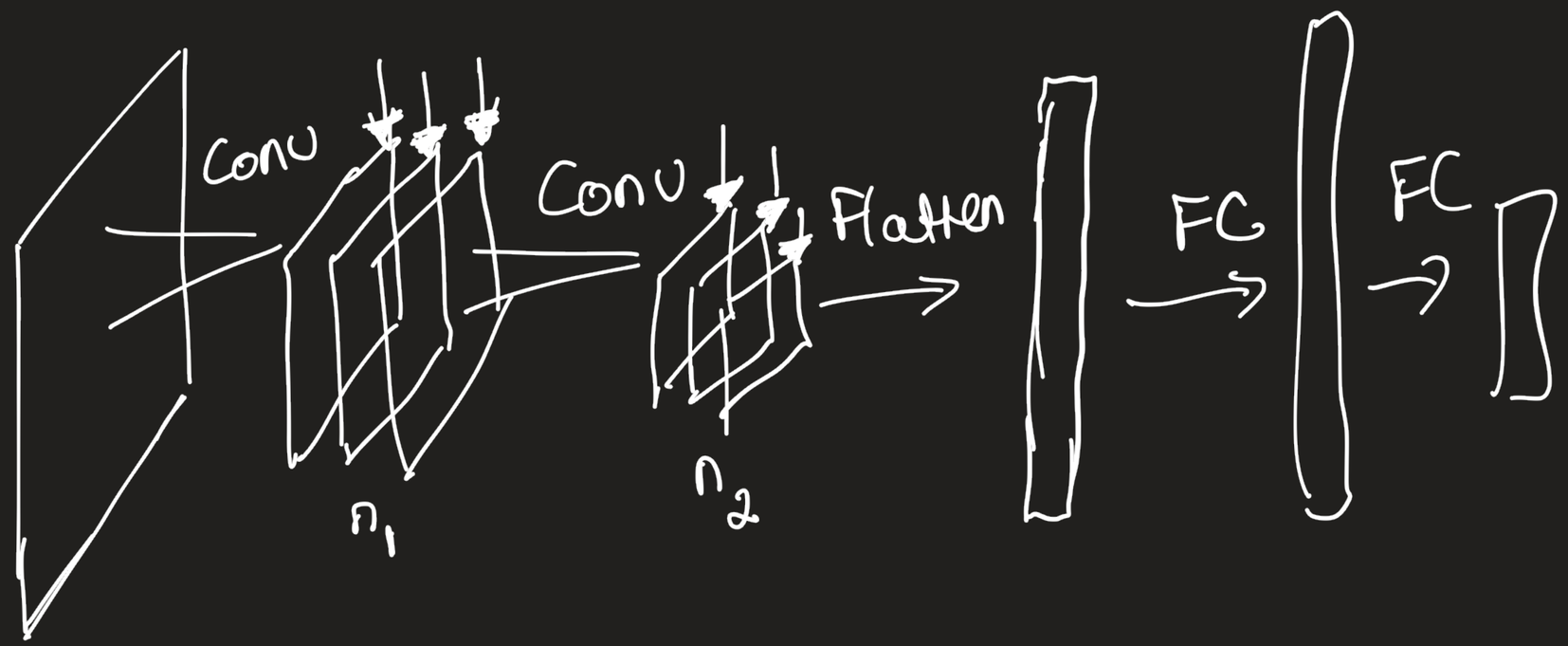
I * K — 3×3

I * K stride of (2,2)

— 2×2

I - 5×5

CNN diagram



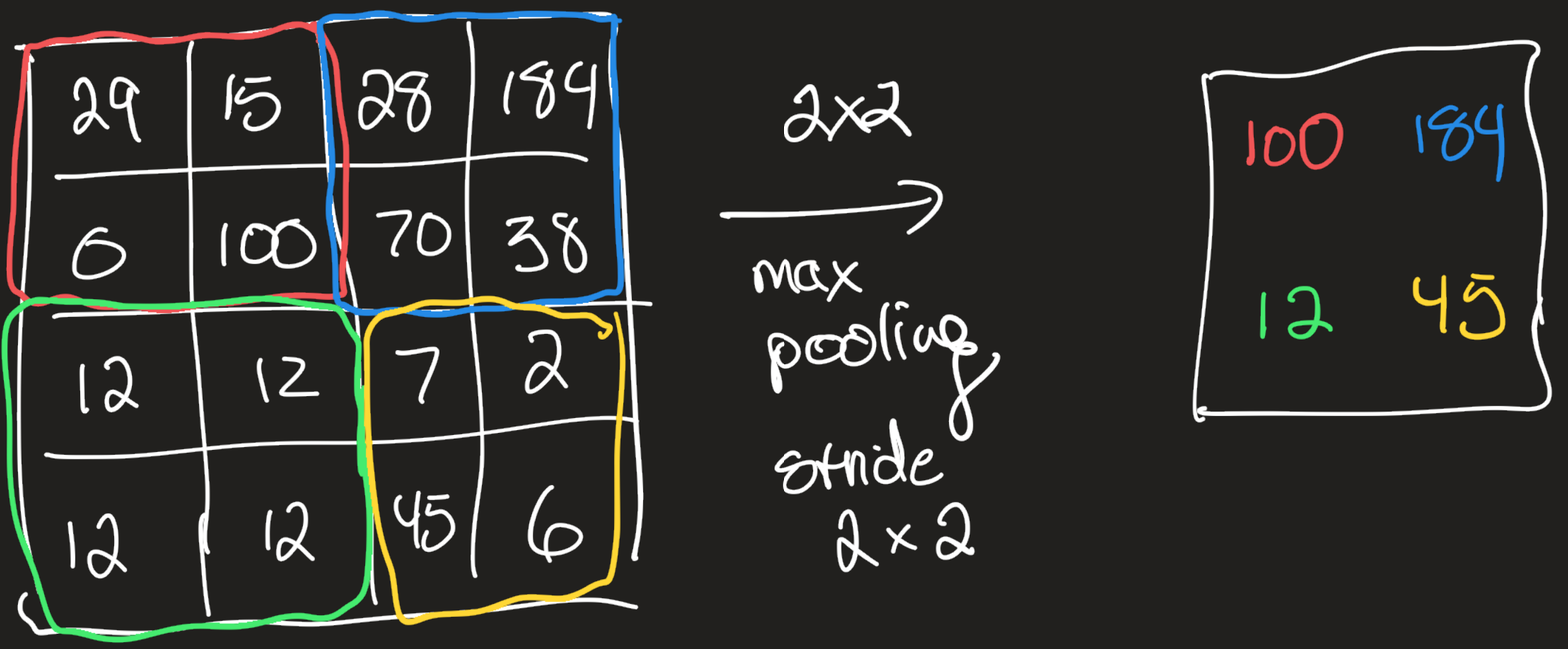
Image

Pooling

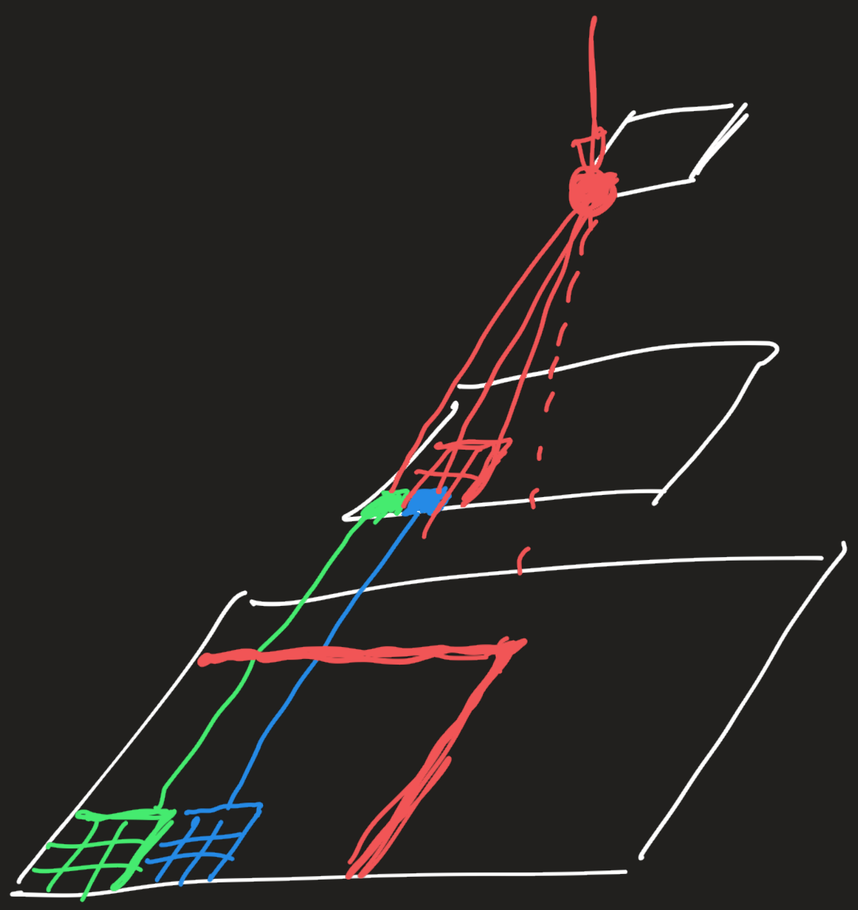
Pooling reduces the dimensionality of a convolutional layer by aggregating locally

Typical examples:

- max pooling
- average pooling



Effective Receptive Fields



- Effective Receptive Field for a feature is the set of pixels on which it depends, in the original image

Ex convolutional NN: LeNet (1997)

5-layer CNN

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