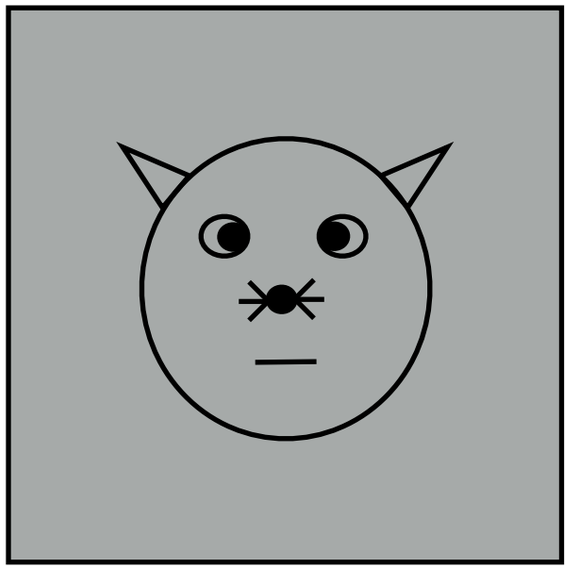
Deep Learning

Holger Schmidt

- **Motivation**
  - Require the input–output function to be insensitive to irrelevant variations of the input, while being very sensitive to particular minute variations.
  - In images, local combinations of edges form motifs, motifs assemble into parts, and parts form objects.

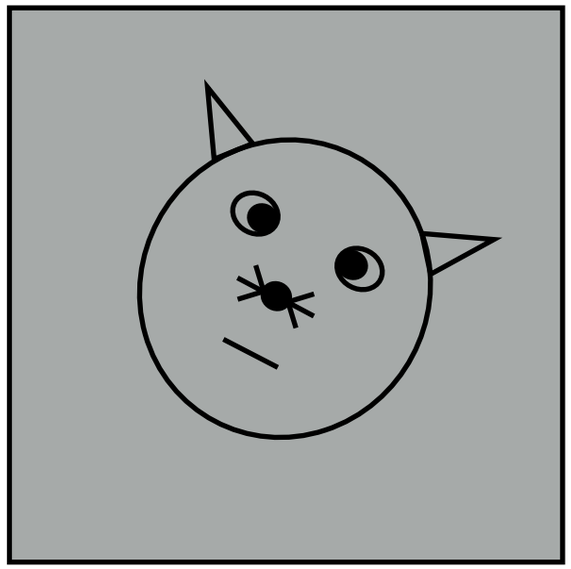
- **Two-fold reason for Convolution/Pooling:**
  - First, in array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected.
  - Second, the local statistics of images and other signals are invariant to location. In other words, if a motif can appear in one part of the image, it would appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array.

Want Input-Output-Function to be **insensitive** to



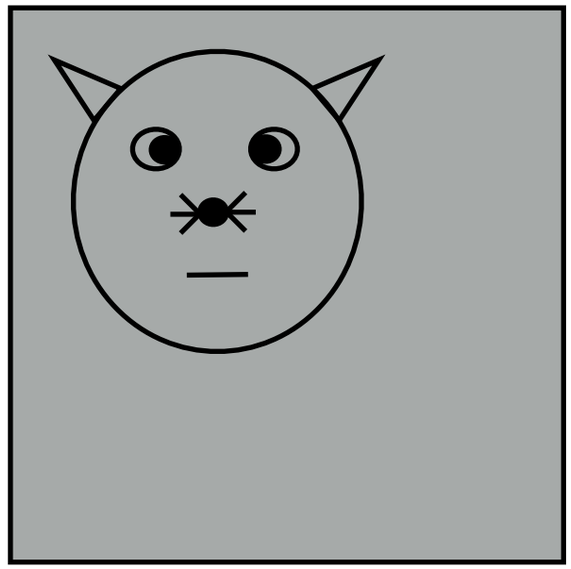
Target: Cat

Orientation



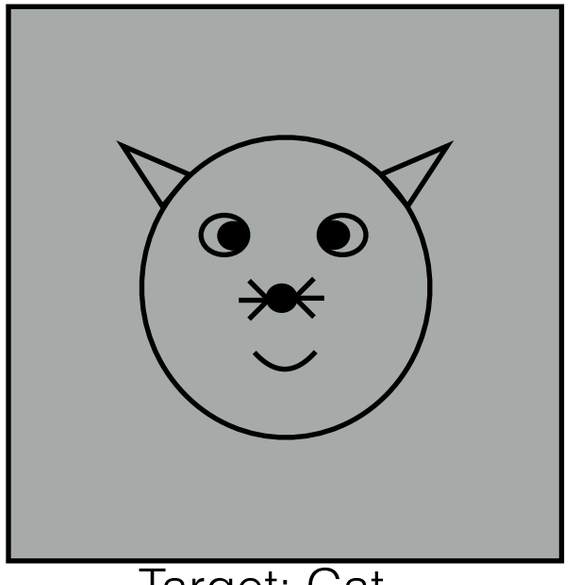
Target: Cat

Position



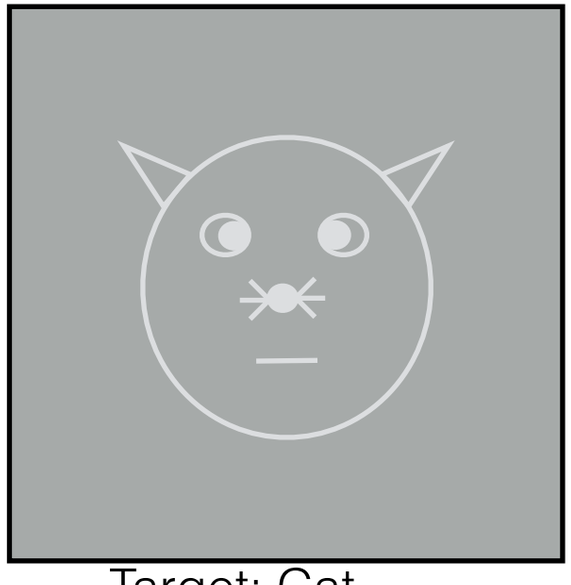
Target: Cat

Pose



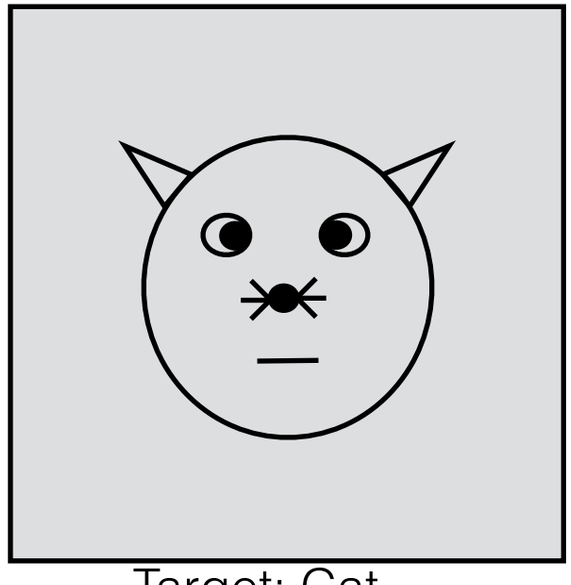
Target: Cat

Illumination



Target: Cat

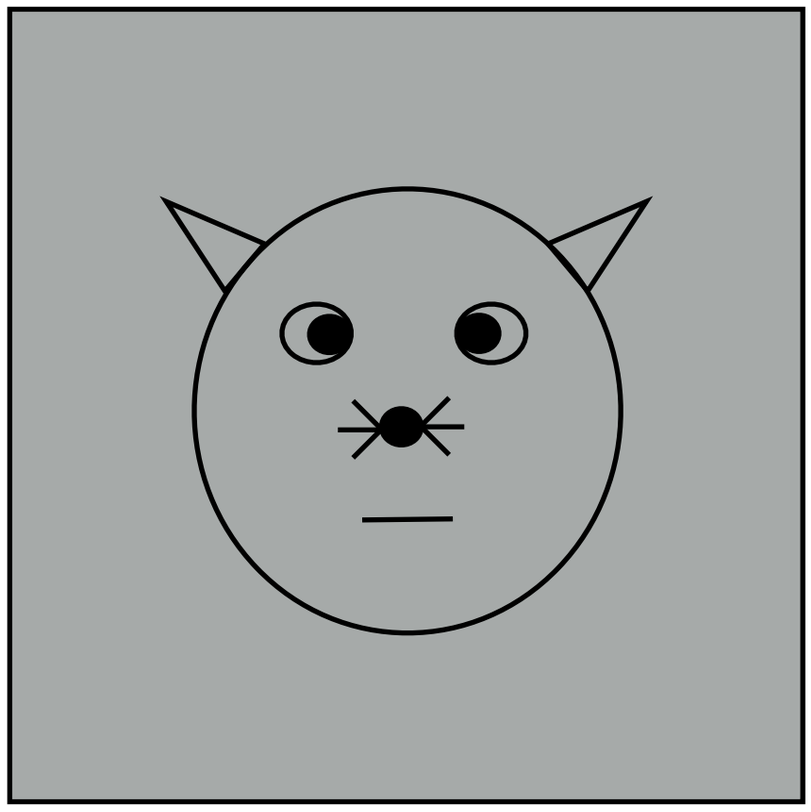
Background



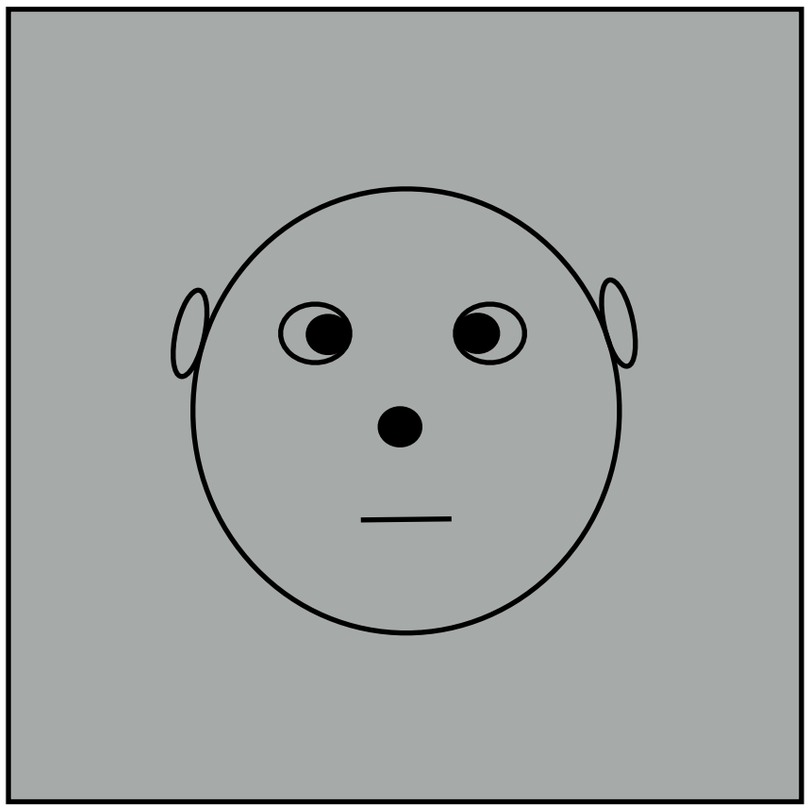
Target: Cat

Want Input-Output-Function to be **very sensitive** to

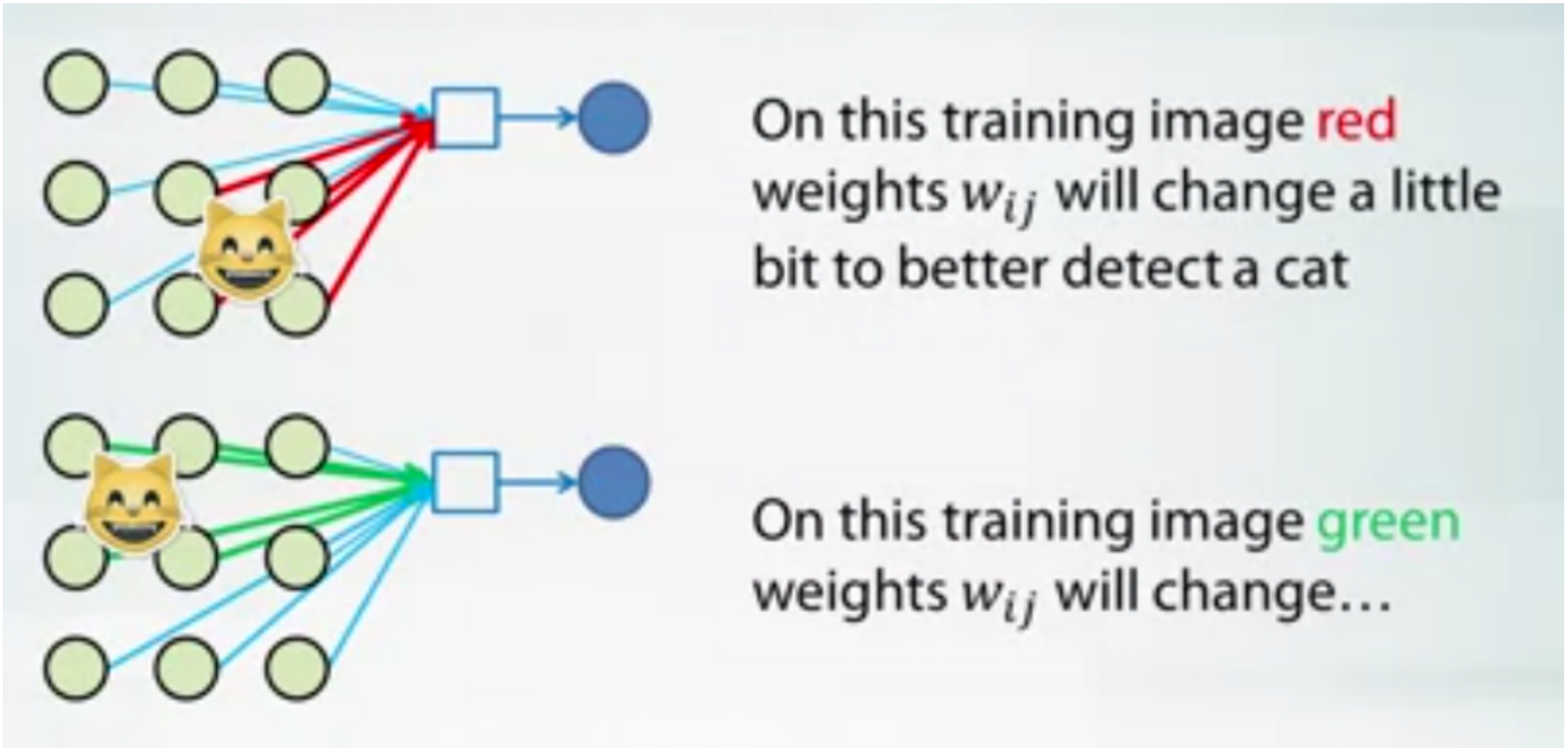
particular minute variations



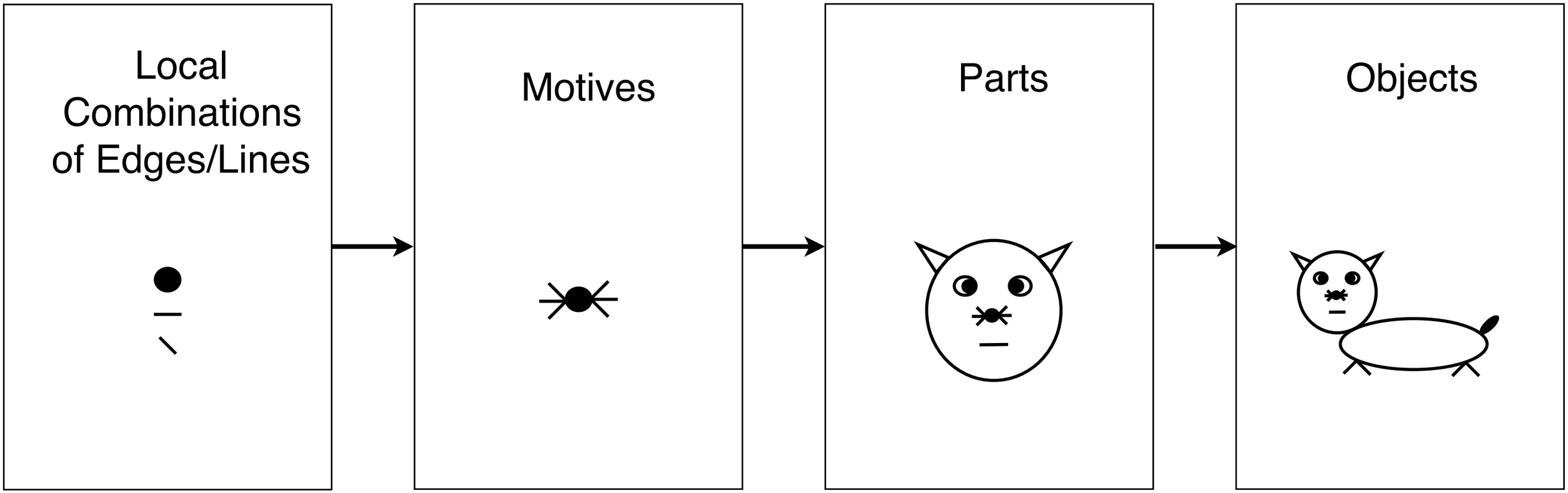
Target: Cat



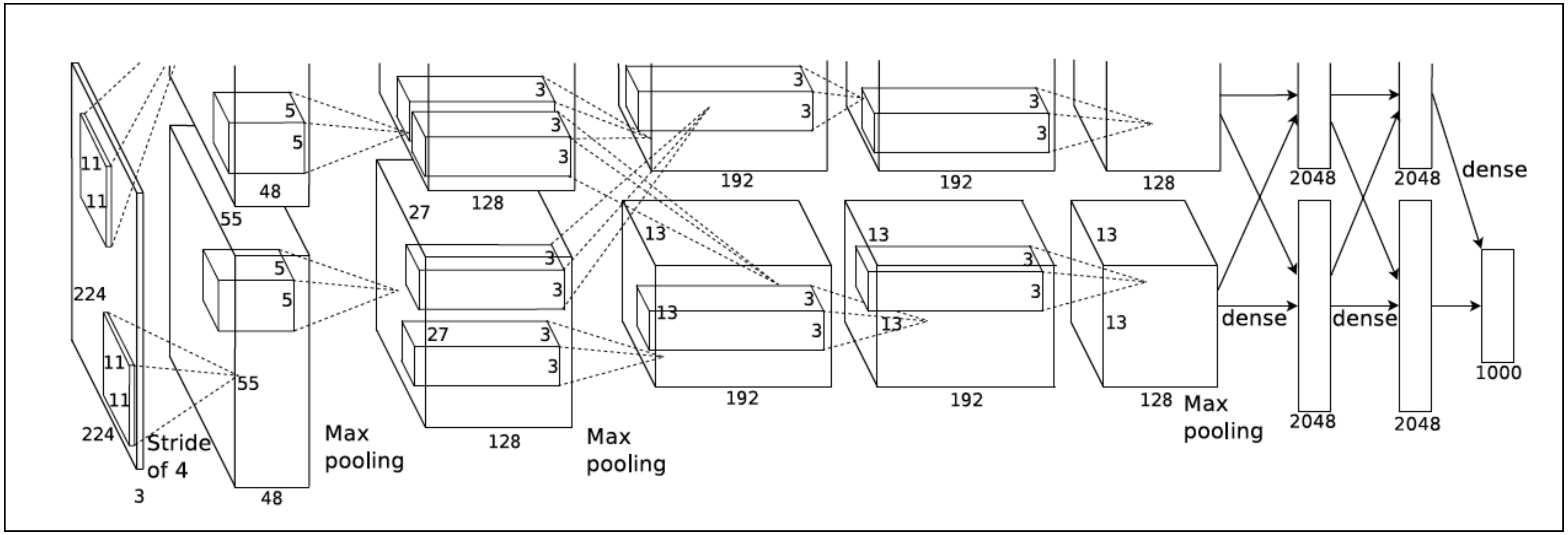
Target: Human


On this training image red weights w_{ij} will change a little bit to better detect a cat
On this training image green weights w_{ij} will change…

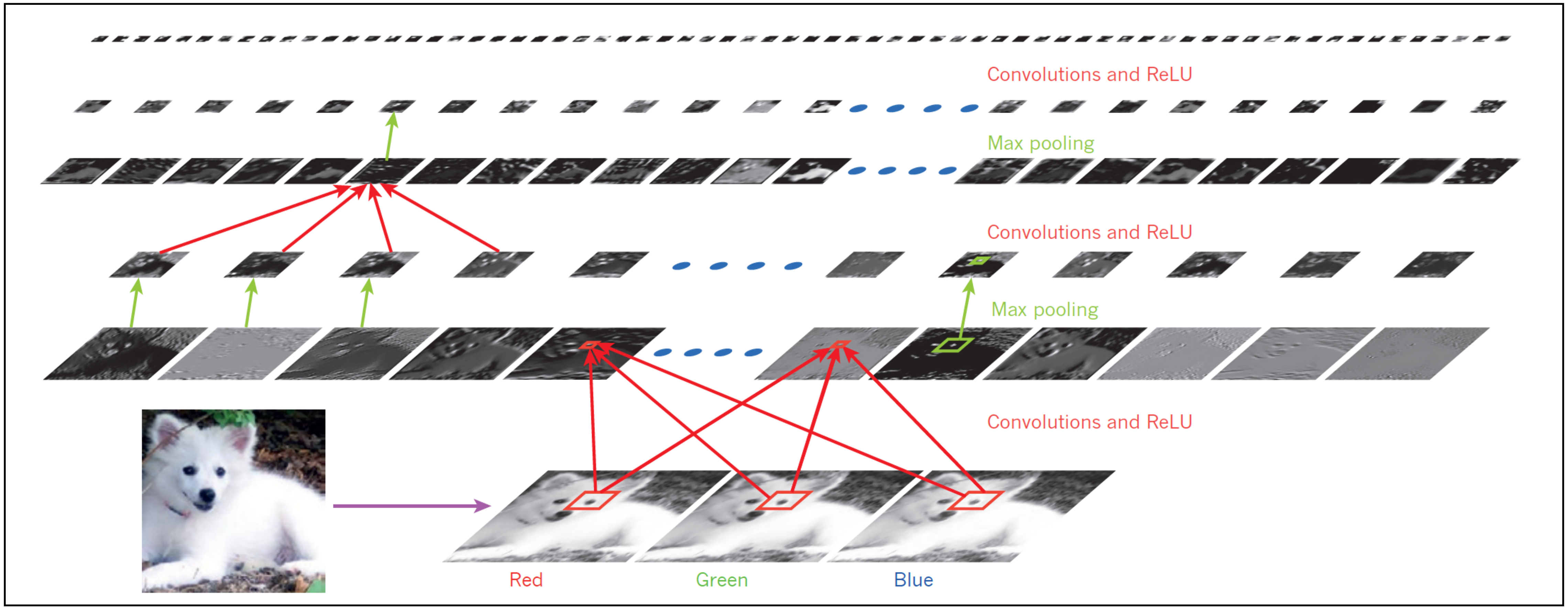
Convolutional Neural Networks



AlexNet



224
224
3
11
11
Stride
of 4
55
55
48
5
5
48
Max
pooling
27
27
128
128
3
3
Max
pooling
13
13
192
192
128
Max
pooling
dense
dense
dense
2048
2048
1000

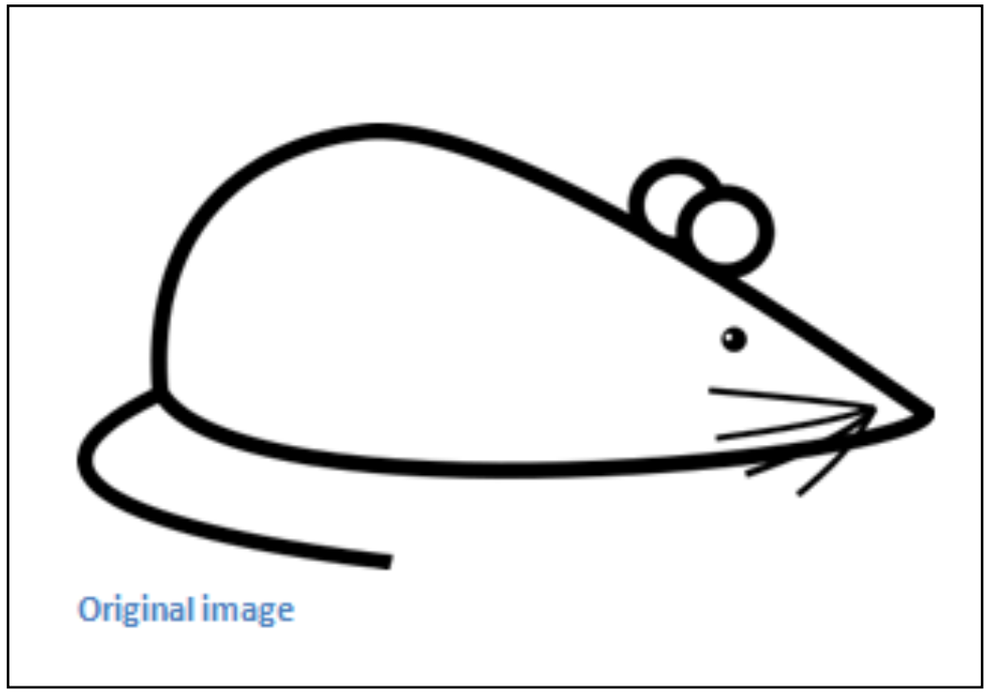


Convolutions and ReLU
Max pooling
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

Convolutional Neural Networks solve all this issues by using

- Convolution
- Pooling
- Fully Connected Layer

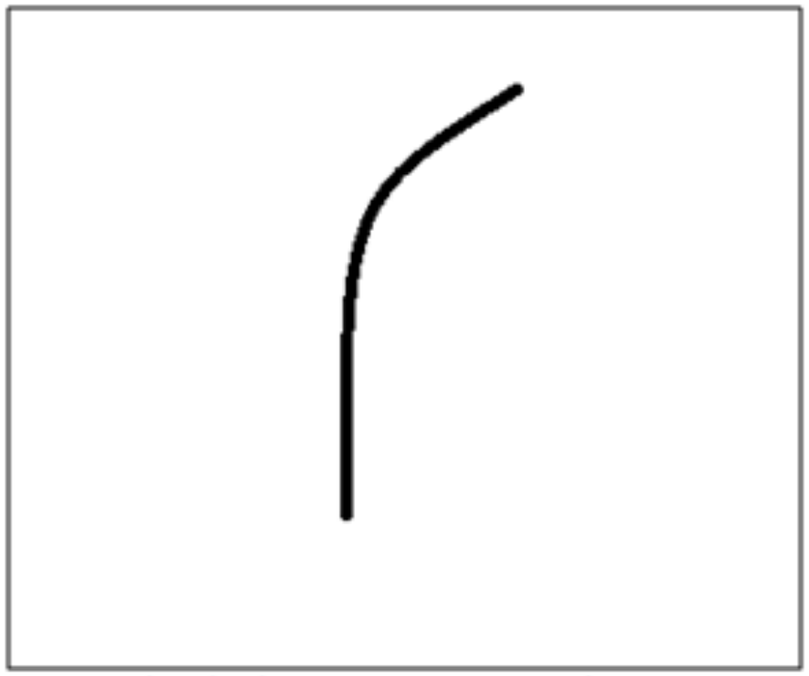
Convolutional Neural Networks



Originalbild

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

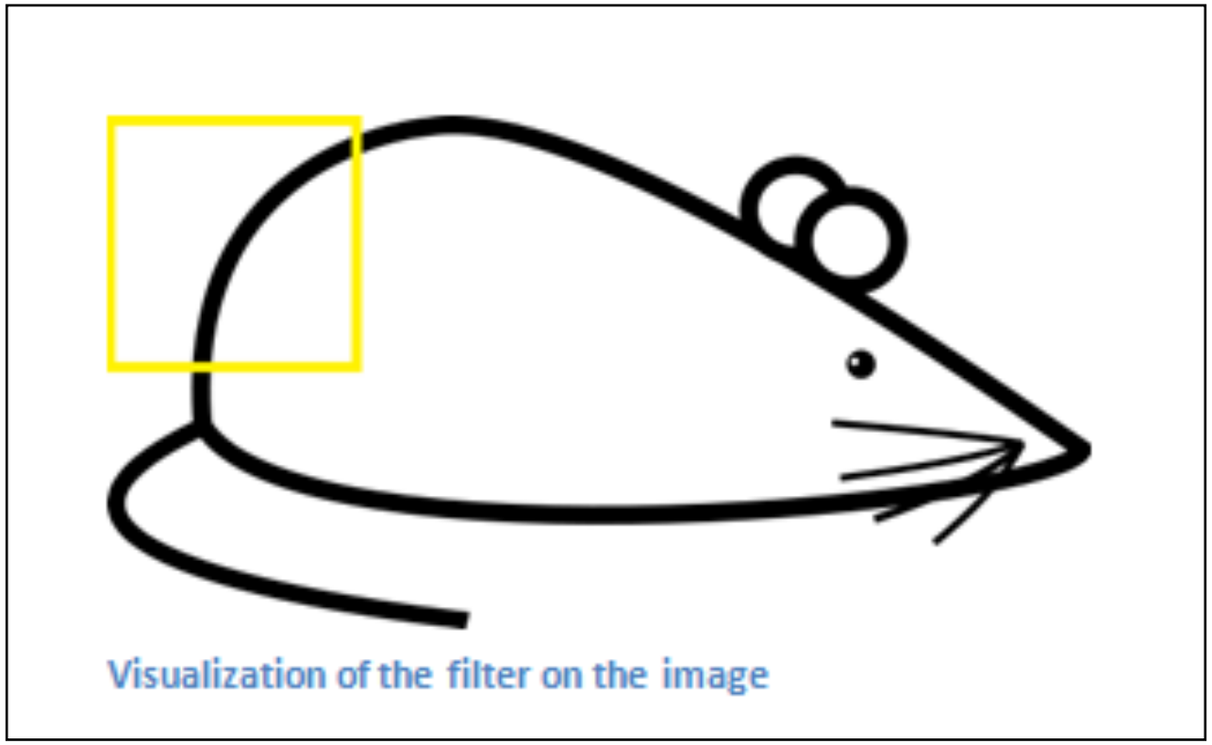
Pixel representation of filter



Visualization of a curve detector filter

7 x 7 Filter-Kernel

Convolutional Neural Networks



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

$= 6000$

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

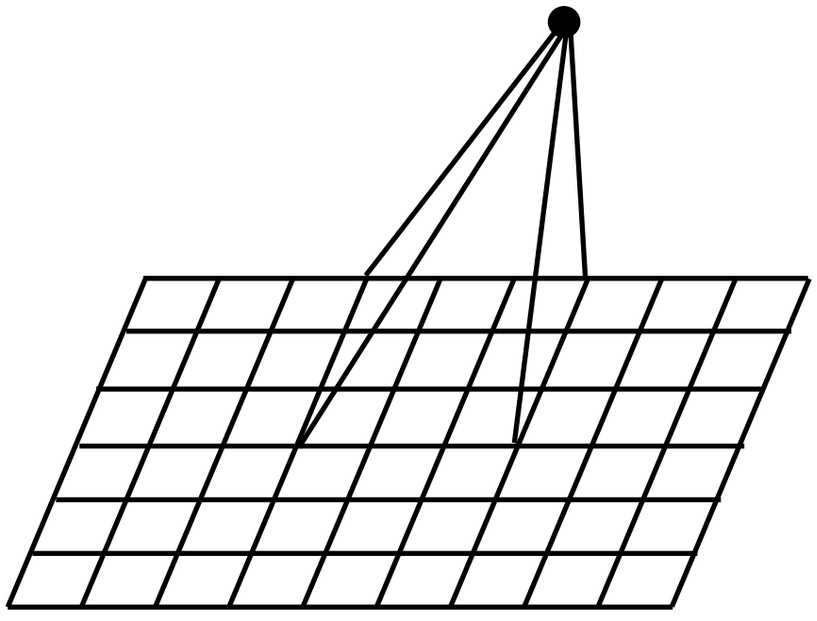
$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

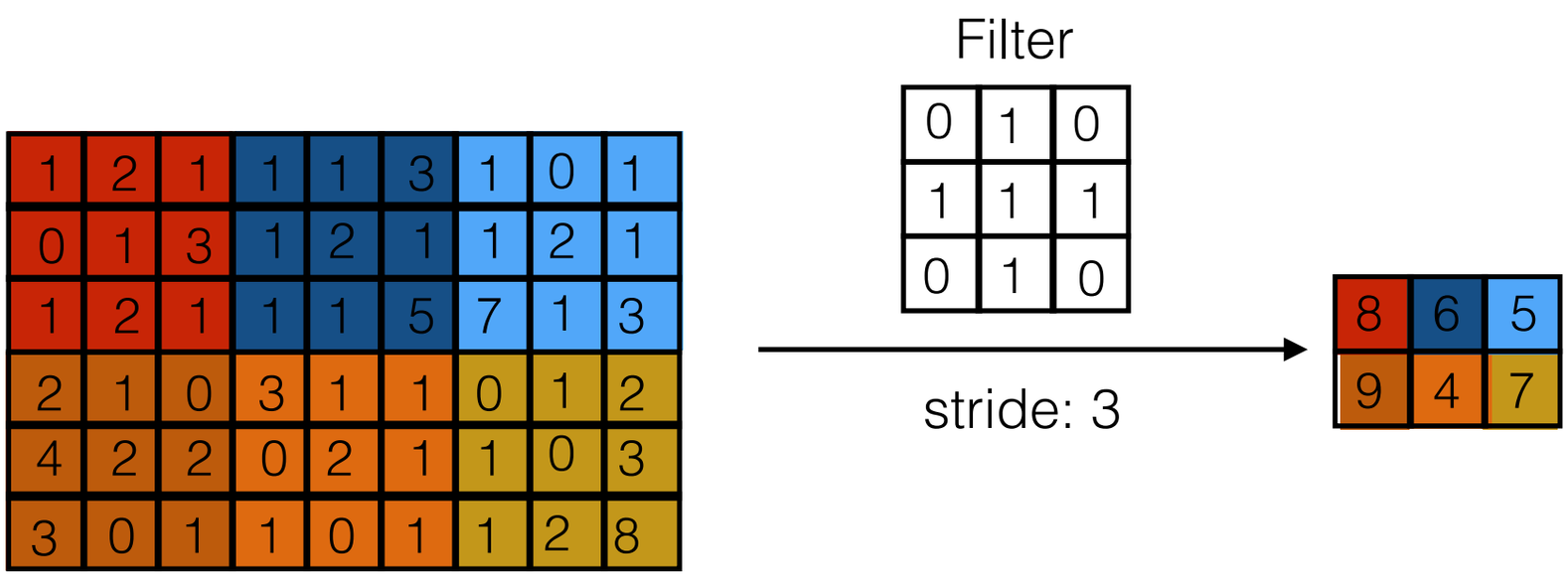
Pixel representation of filter

$= 0$

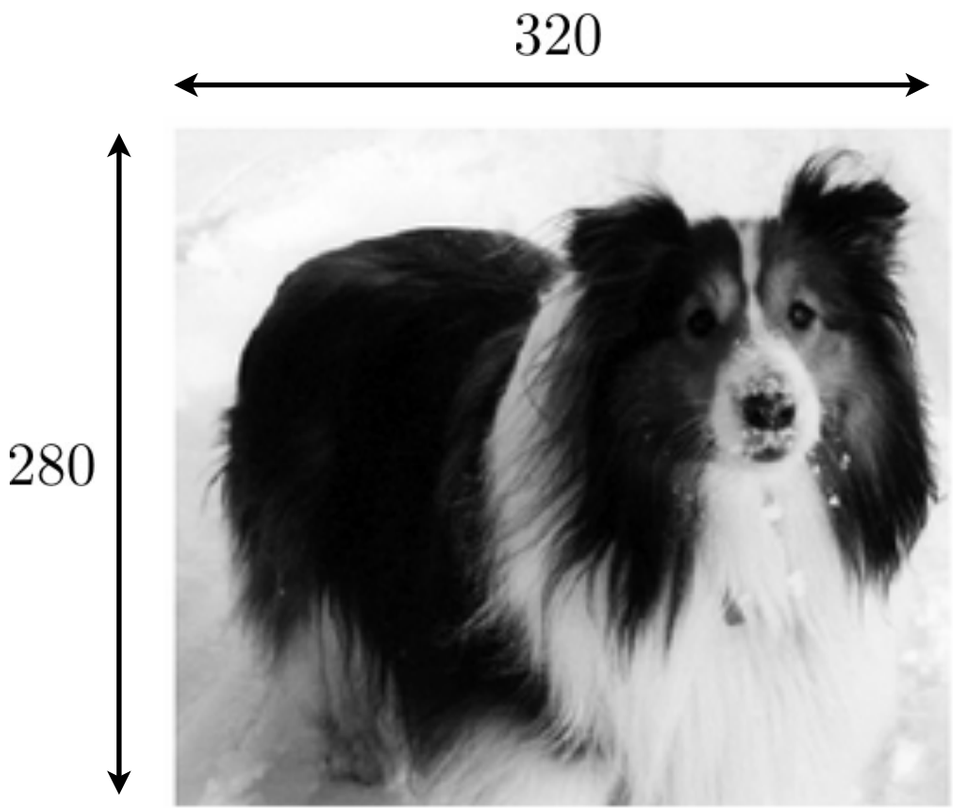
Convolutional Neural Networks



$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n)$$

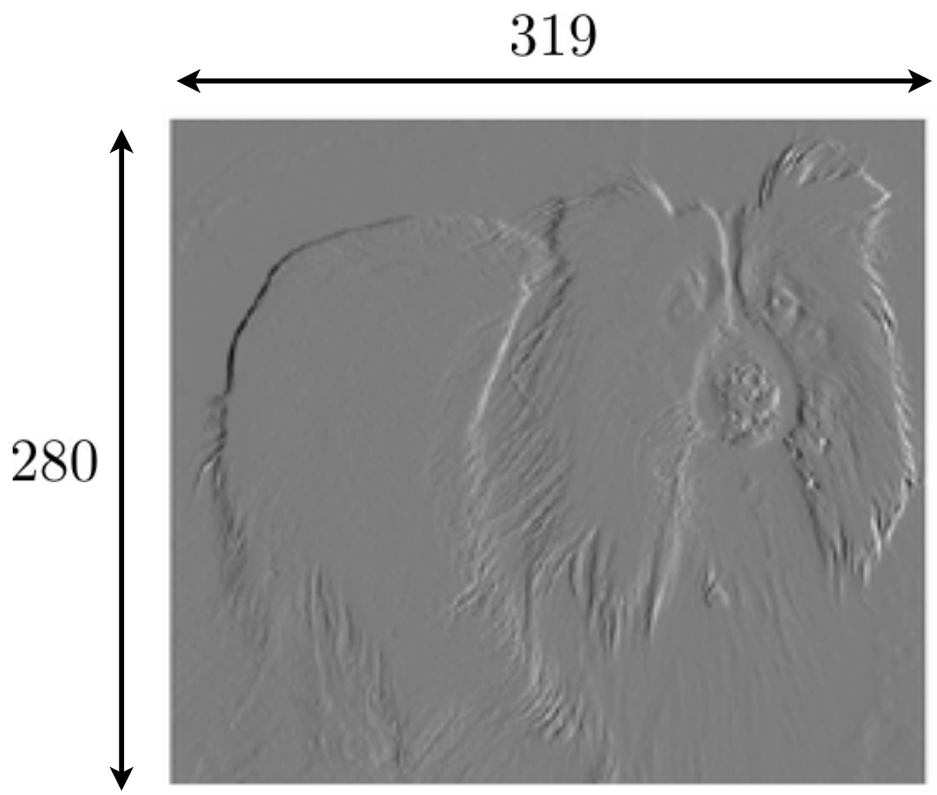


Convolutional Neural Networks



Filter

+1	−1



Convolution: 319 × 280 × 3 = 267.960 FLOPs

Matrix Multiplication: 319 × 280 × 320 x 280 x 2 = 8 Milliarden FLOPs

Convolution is **60.000 faster** than Matrix-Multiplikation

Convolutional Neural Networks