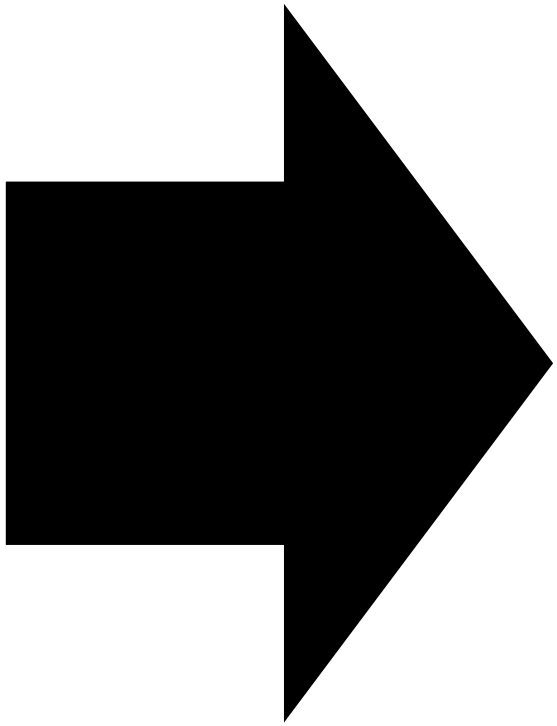
Neuroengineering (I)
4. Deep Learning with ANN

- **Scuola di Ingegneria Industriale e dell'Informazione** – Politecnico di Milano
- Prof. Pietro Cerveri

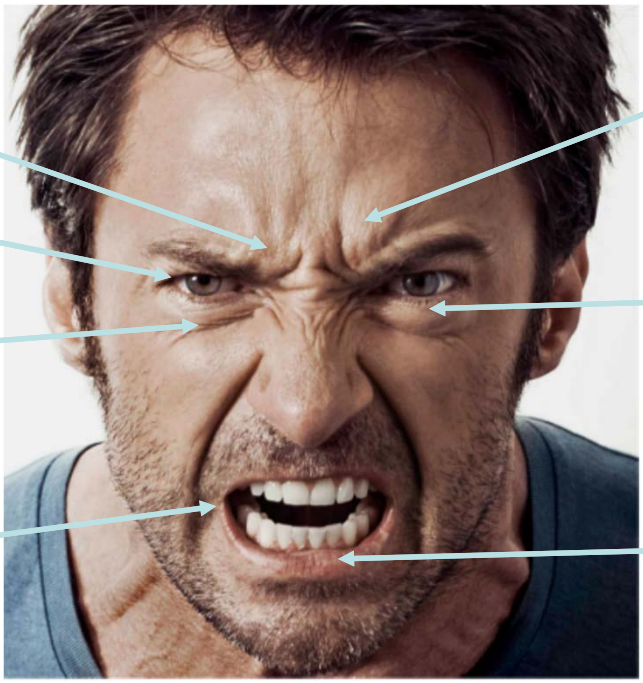
Cognition is based on abstract models



Visual models

From sensation to semantics
Synthesizing features

Visual features of anger



Dense scalar signal → Deconstruction → Features

Deconstruction – Learning in depth

Dense scalar signal

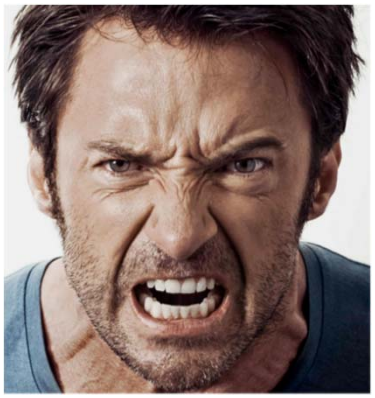
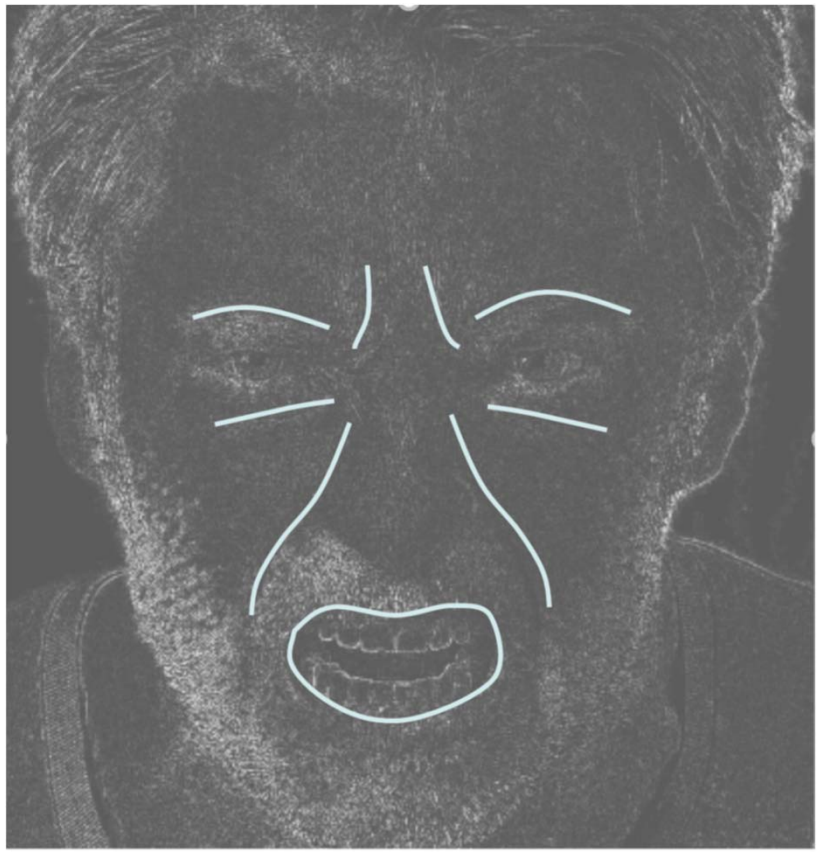


Image processing



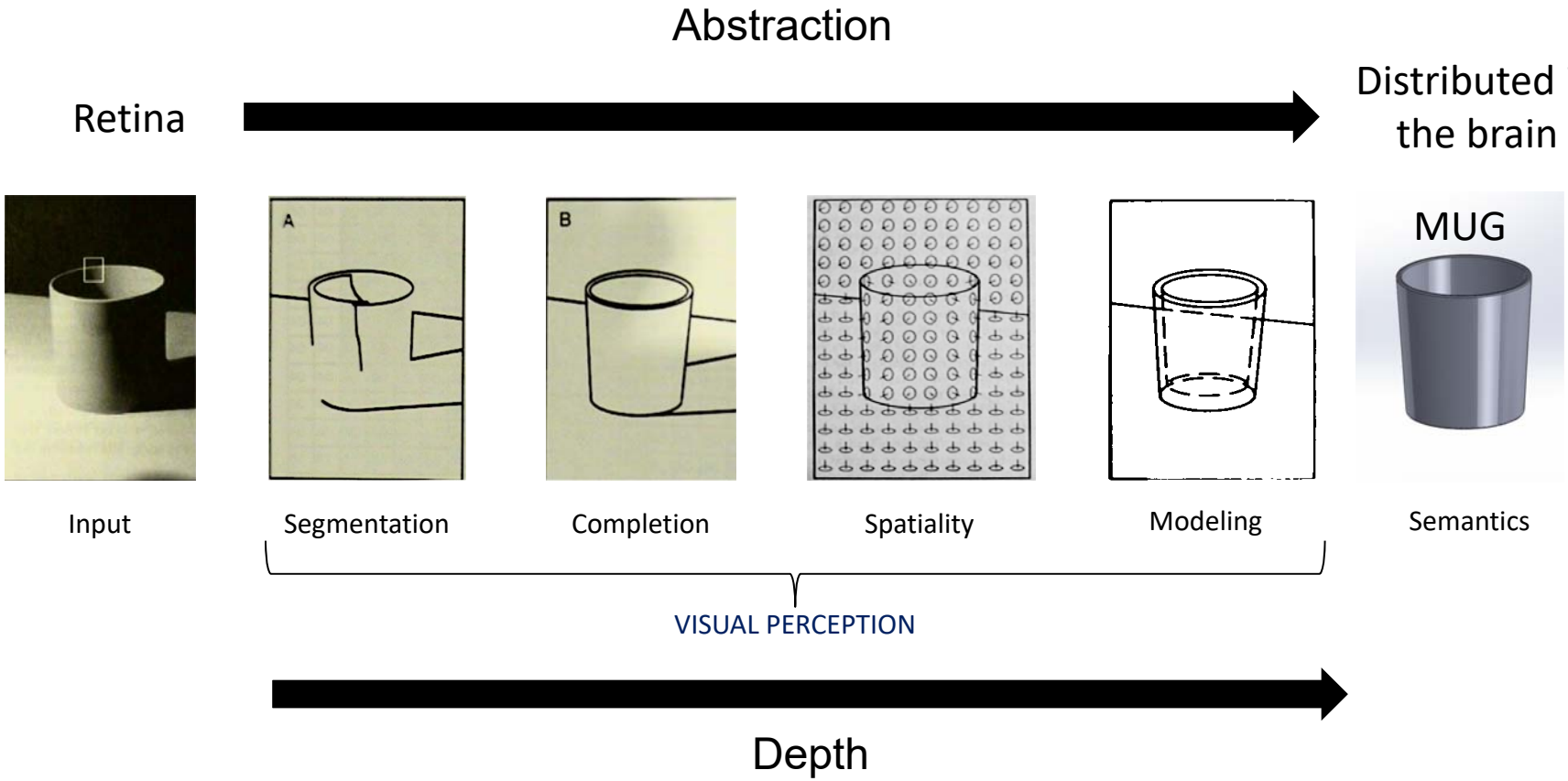
Features

Deep learning

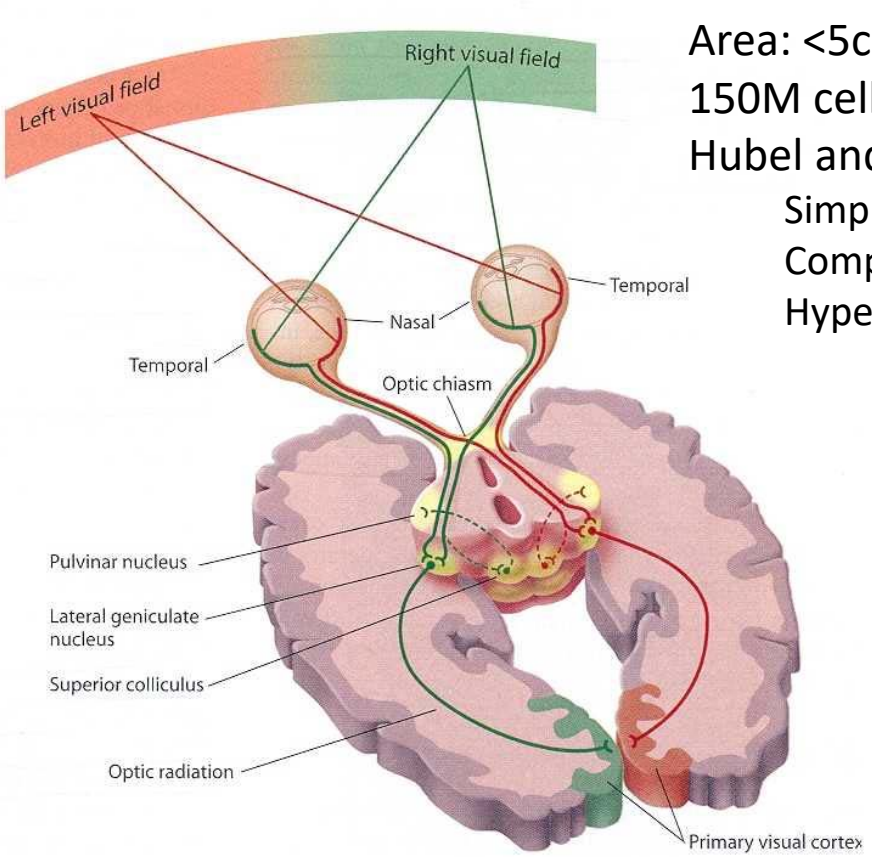
- From sensation to semantics
- Increasing abstraction
- Heterogeneous components (feature maps)
  - Physical (pixel intensity, signal frequencies, …)
  - Scalar maps (lines, contours, curves, colors, ..)
  - Vector descriptors (geometries)
  - Perceptual (completion, depth, motion,..)
  - Emotional (aesthetics, affection, personal experiences,..)
  - Semantic (language/concepts)

1. Sensorial experience is deconstructed into the brain
2. Representation made of a cascade of layers
3. Mechanisms of knowledge reconstruction

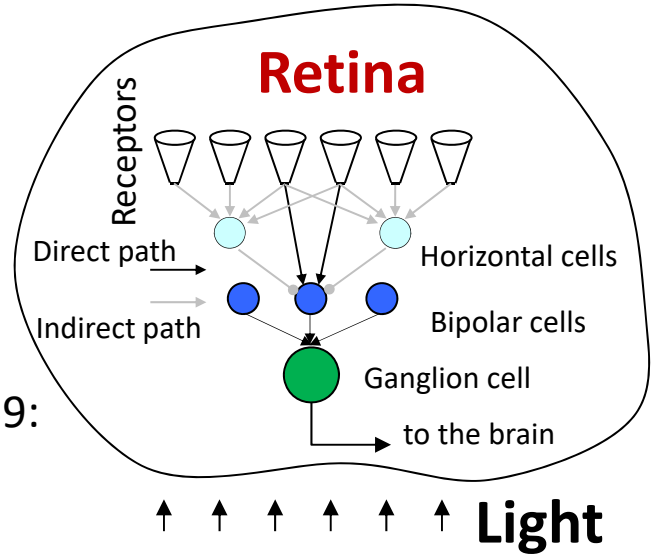
Deconstruction (progressive encoding)

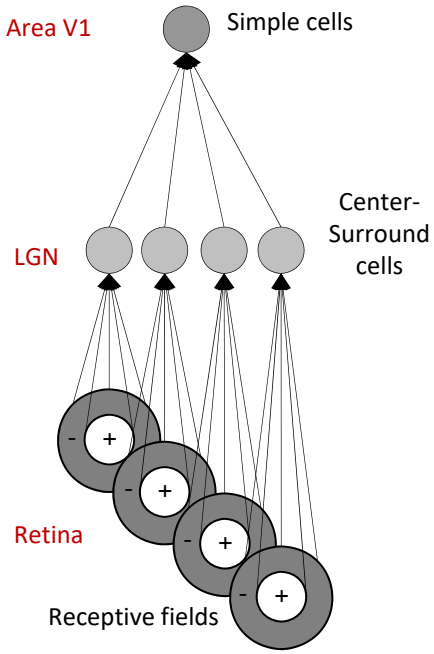


Primary visual cortex



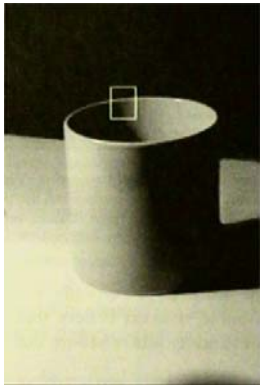
Thickness: 2mm
Area: <5cm^2
150M cells
Hubel and Wiesel, 1959:
- Simple cells
- Complex cells
- Hyper-complex cells



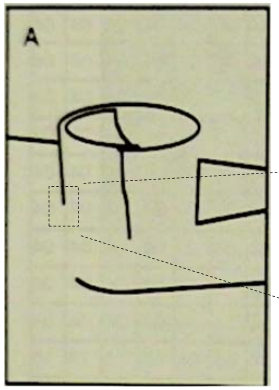


Visual encoding

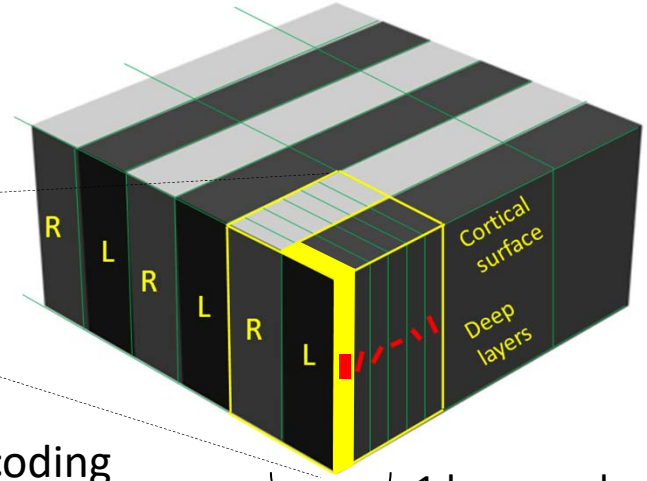
Retina



Edge detector

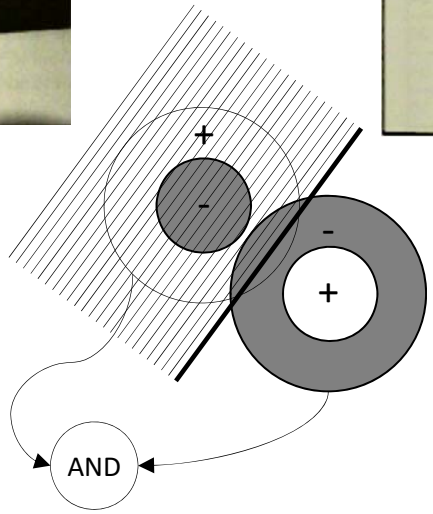


Retinotopic map in primary visual cortex (V1)



Edge encoding

1 hyper-column for orientation

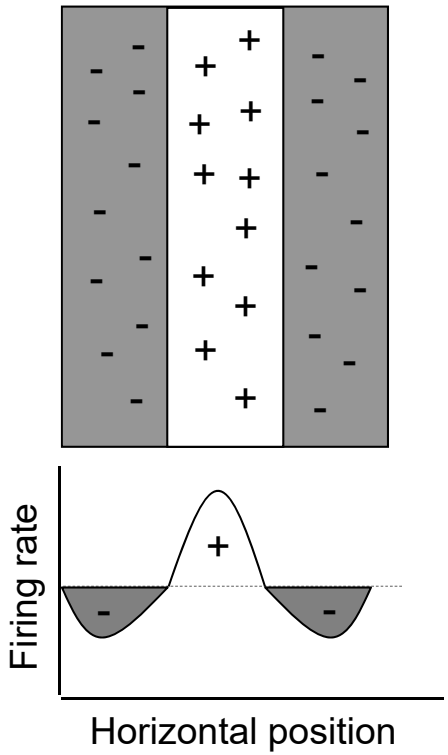


Neural implementation in LGN (signals from ganglion cells)

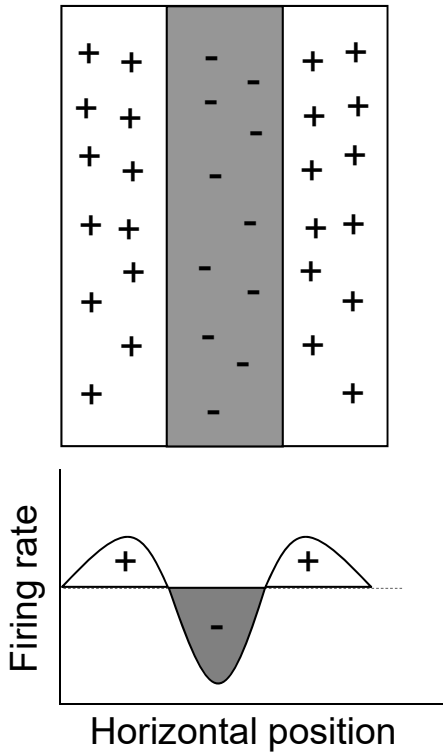
- On-center ∩ Off-center

Receptive field of simple cells

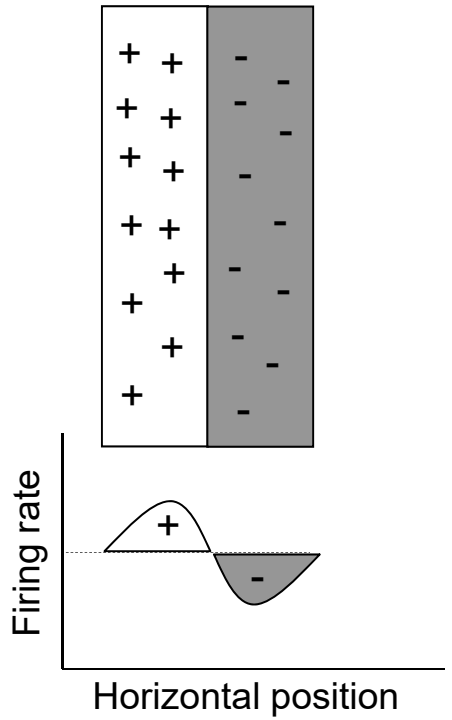
Detector for bright vertical line



Detector for dark vertical line



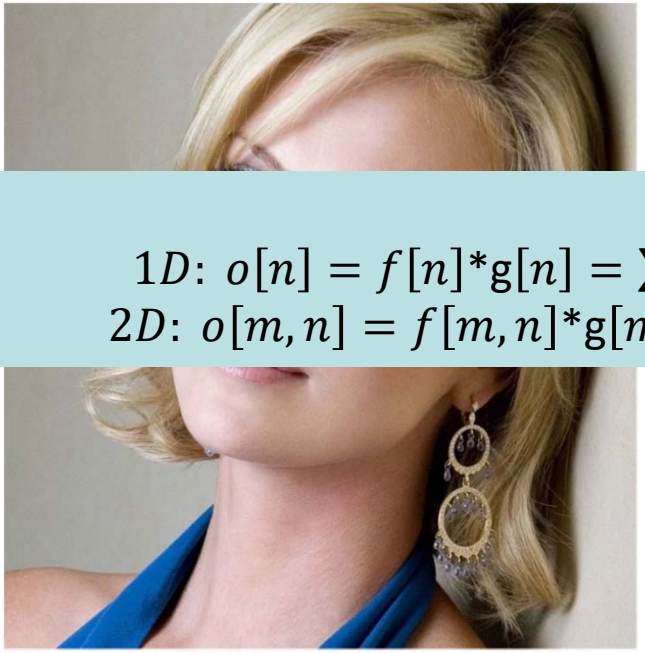
Detector for dark-bright border



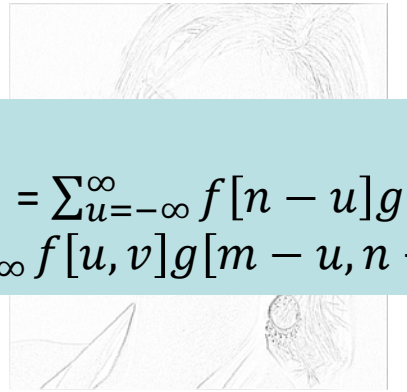
Convolutional neural networks (CNN)

- Artificial implementation of physiological deep learning
- Based on mathematical convolution
- CNN use three basic ideas:
  - *local receptive fields*
  - *shared weights*
  - *rectified linear units* and *pooling*

Input image



Output of the convolution operation



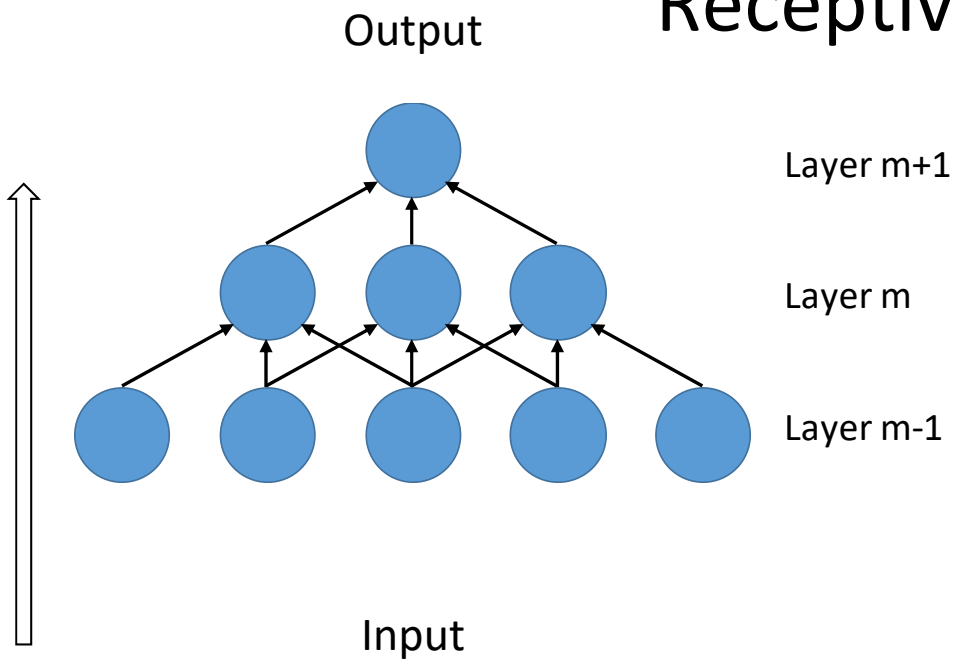
Filtering and down-sampling

Recall that

$$1D:\ o[n] = f[n]*g[n] = \sum_{u=-\infty}^{\infty} f[u]g[n-u] = \sum_{u=-\infty}^{\infty} f[n-u]g[u]$$

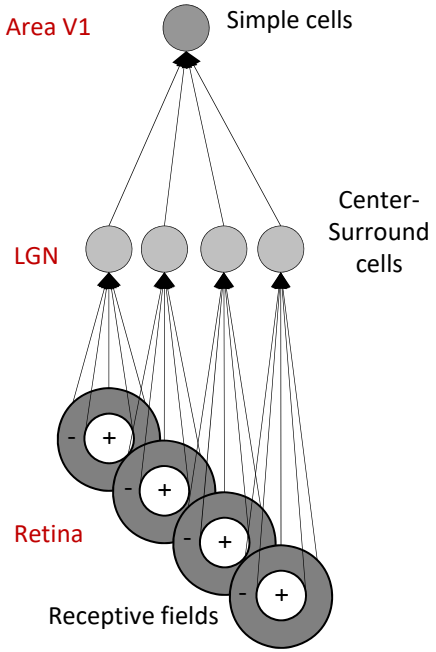
$$2D:\ o[m,n] = f[m,n]*g[m,n] = \sum_{u=-\infty}^{\infty}\sum_{v=-\infty}^{\infty} f[u,v]g[m-u,n-v]$$

Receptive field

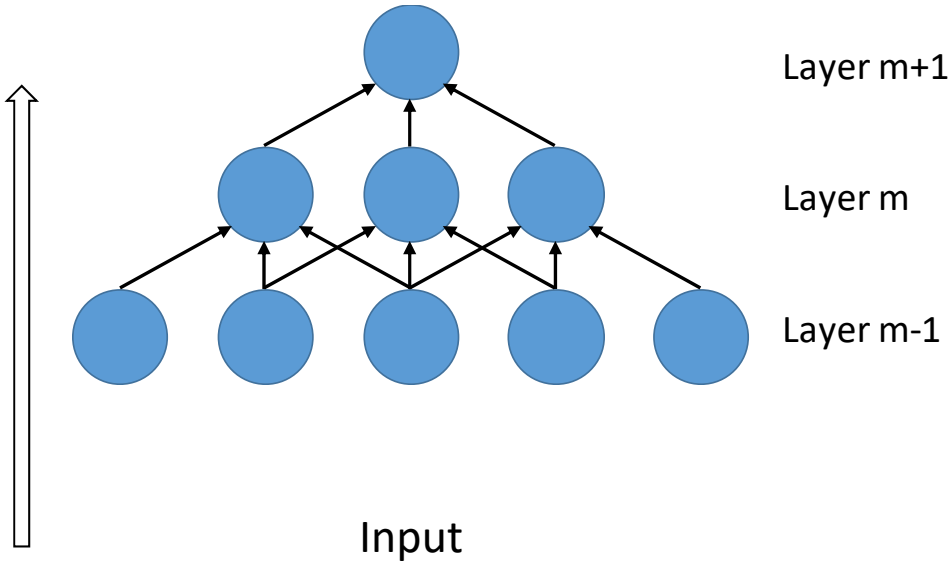


n-unit local receptive field

Each unit is unresponsive to variations outside of its receptive field



Receptive field



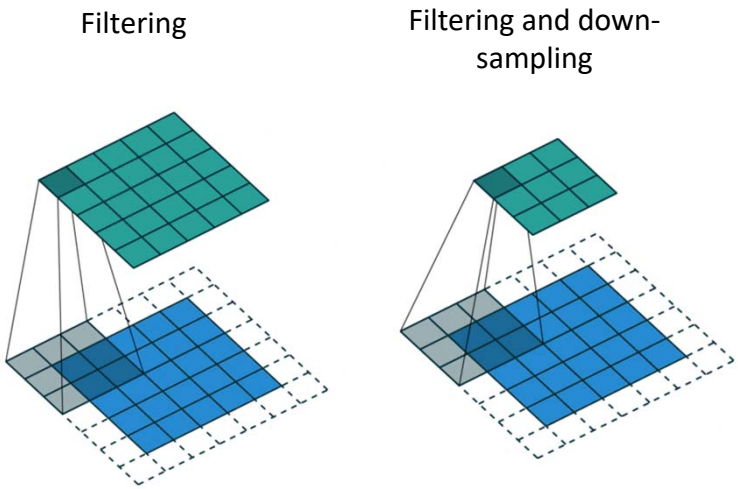
n-unit local receptive field

Each unit is unresponsive to variations outside of its receptive field

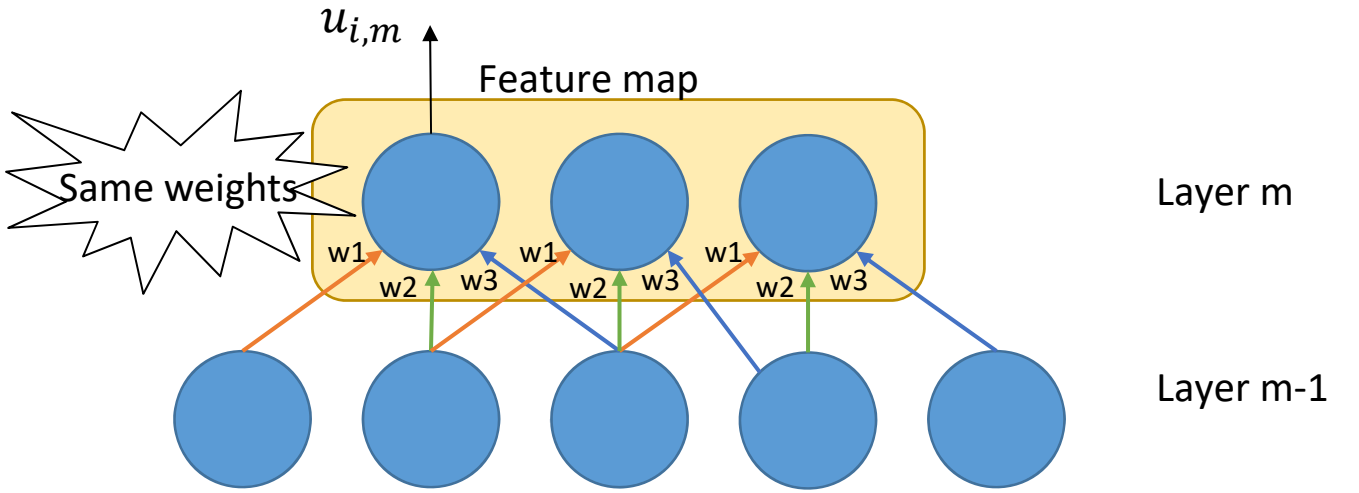
A convolution operation forms a many-to-one relationship

Implemented by means of a spatial filter

Size and stride



Shared weights and feature map

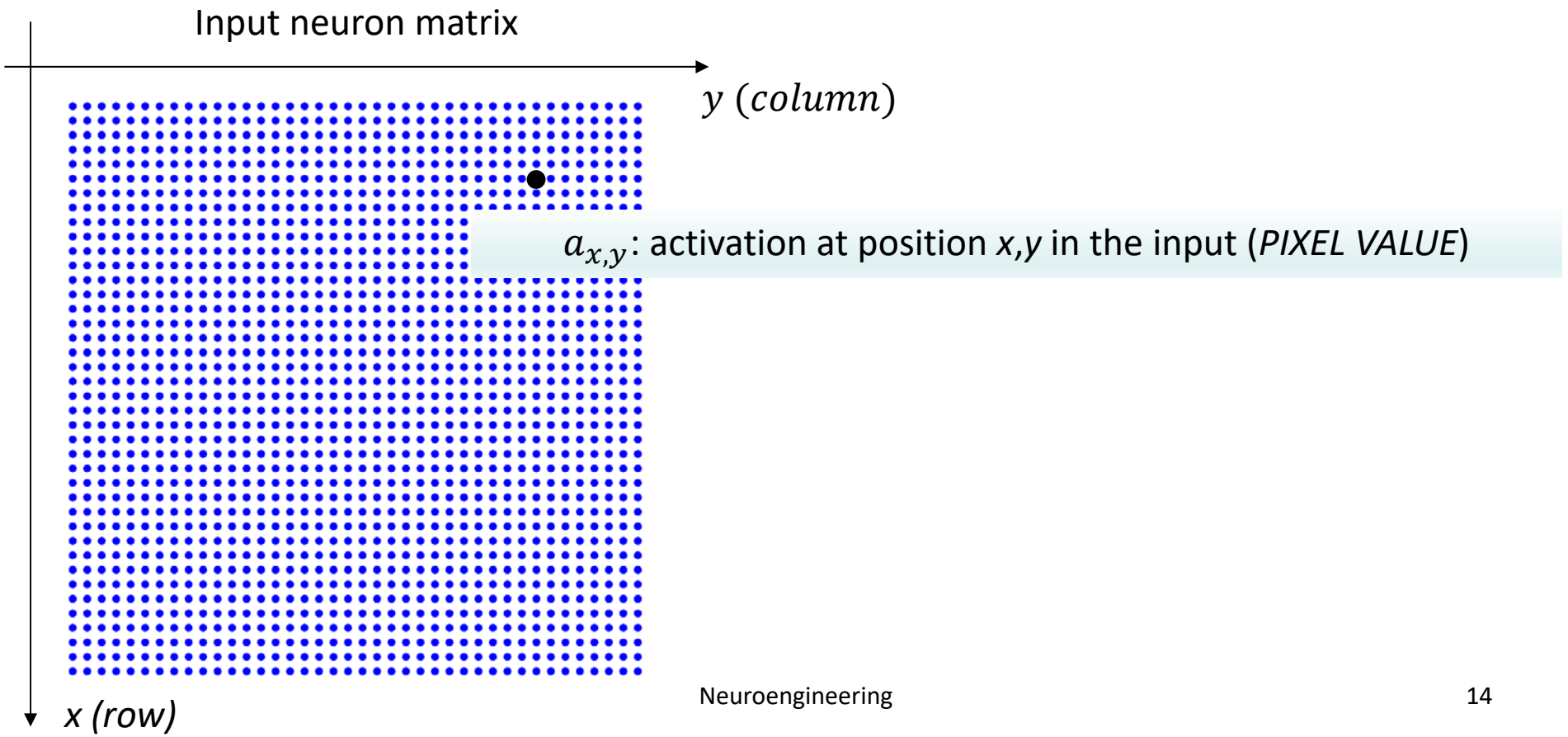


Each neuron shares the same weights in the layer (feature map)

$$u_{i,m} = tanh\left(b + \sum_{n=1}^{N} w_n a_{n,m-1}\right)$$

Convolutional layer: image processing

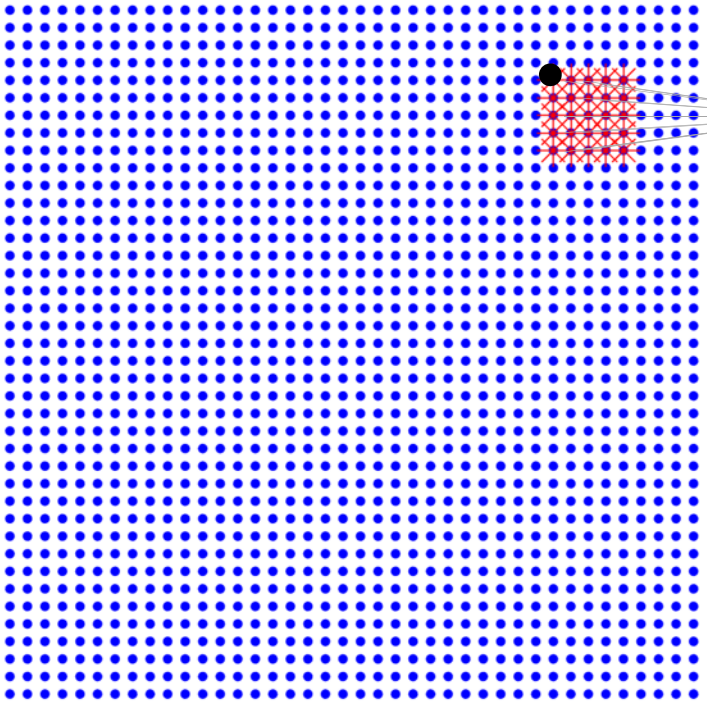
- $K \times K$ pixel input image



Convolutional layer: image processing

- *K*×*K* pixel input image and *n*×*n* LCF (local receptive field)

Input neuron matrix

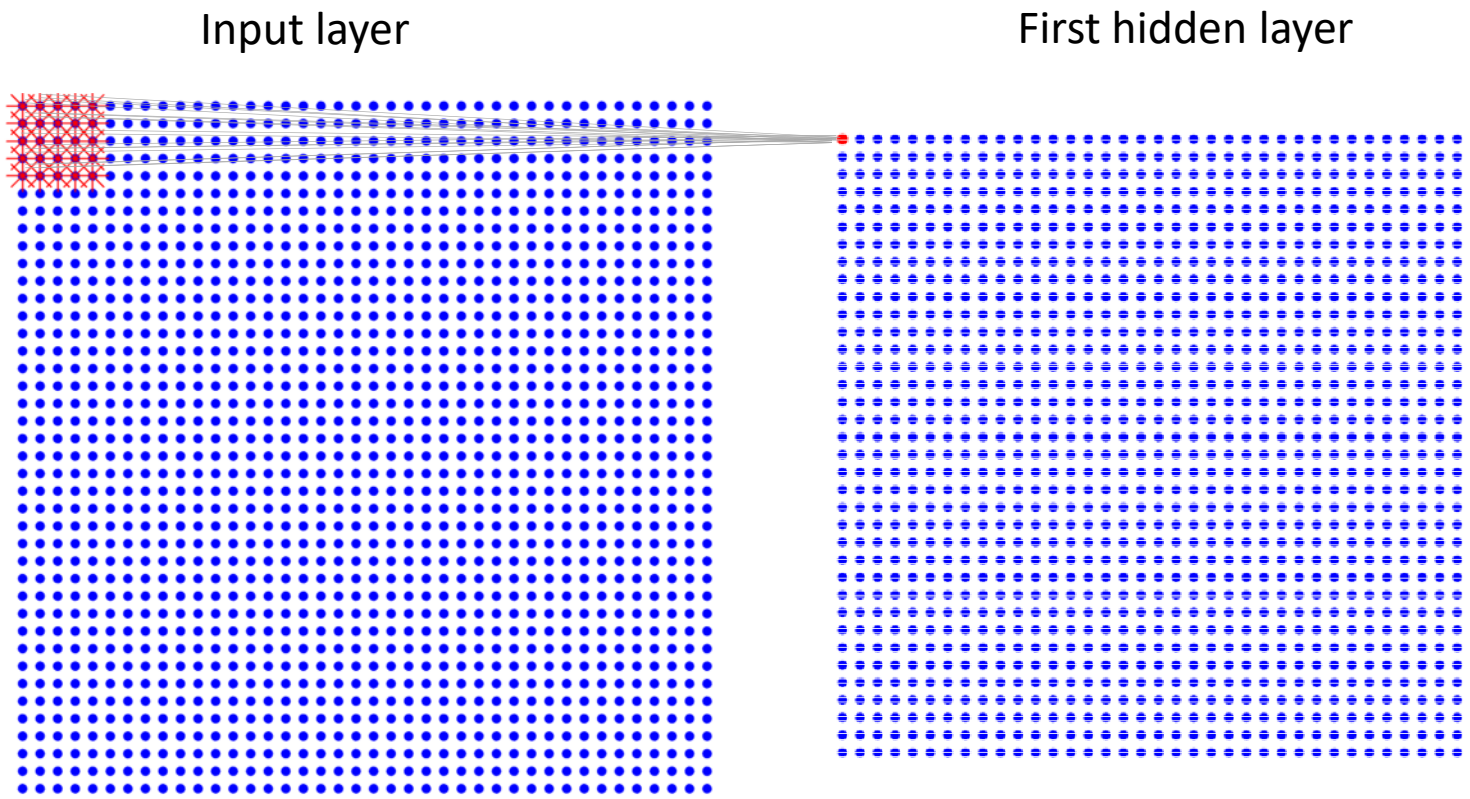


$$u_{j,k} = tanh\left(b + \sum_{l=1}^{L}\sum_{m=1}^{M} w_{l,m} a_{j-1+l,k-1+m}\right)$$

Neuron *j*,*k* in the first hidden layer (matrix representation)

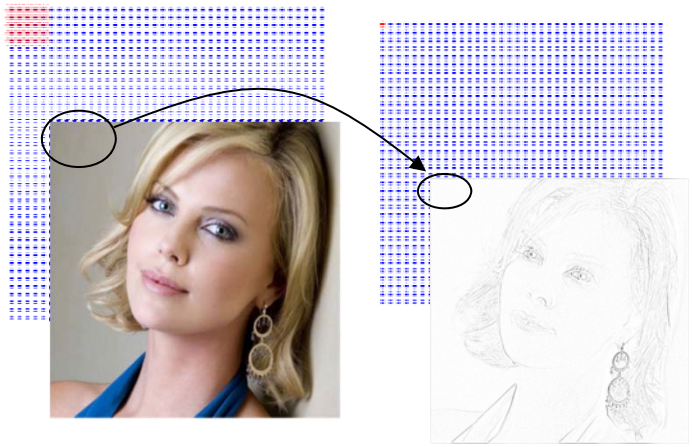
Scanning the image using the LCF

- $K \times K$ pixel input image and $n \times n$ LCF



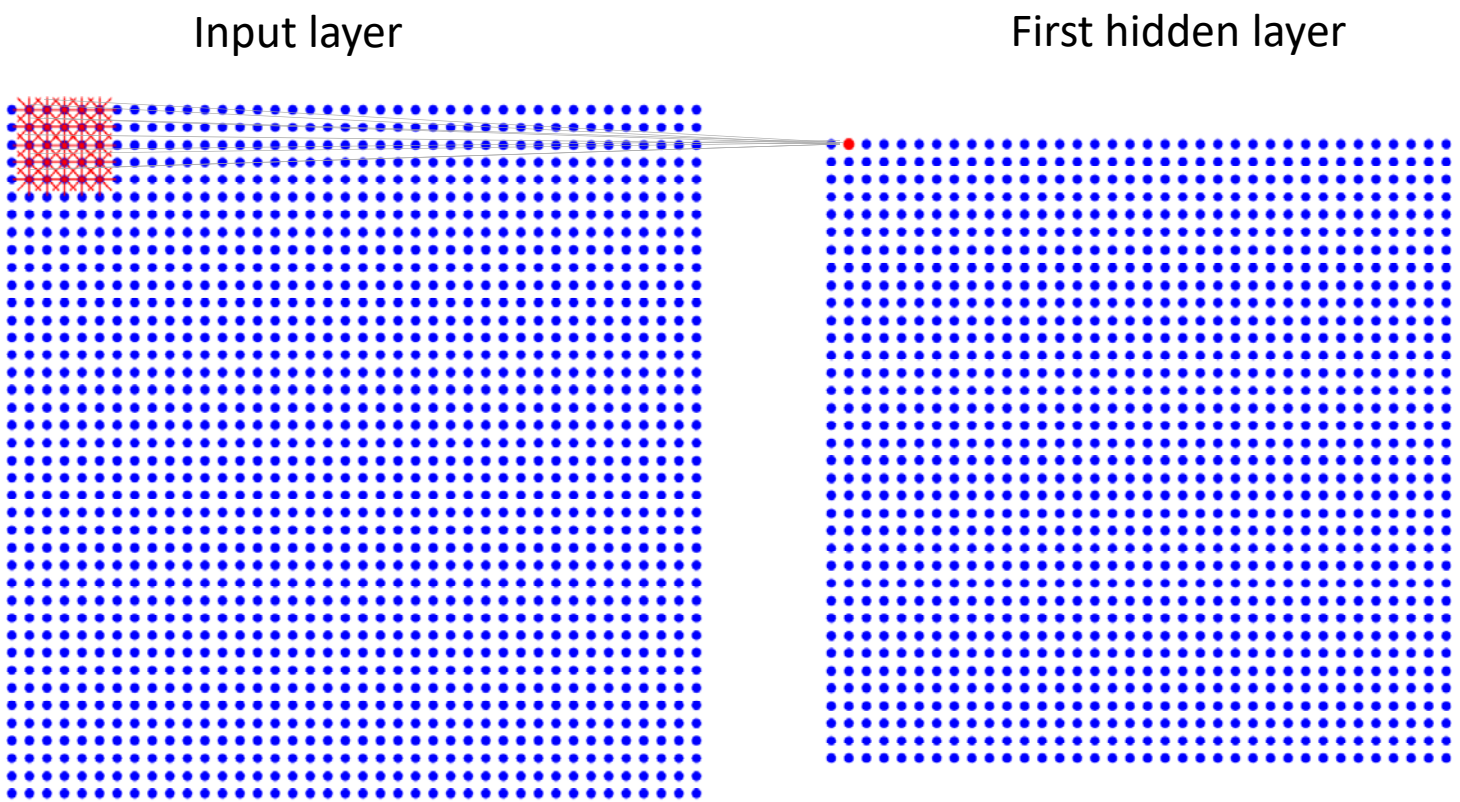
Scanning the image using the LCF

- One important point of such convolution operation is that the positional connectivity exists between the input values and the output values
- For example, the top left values in the input matrix affect the top left values of the output matrix



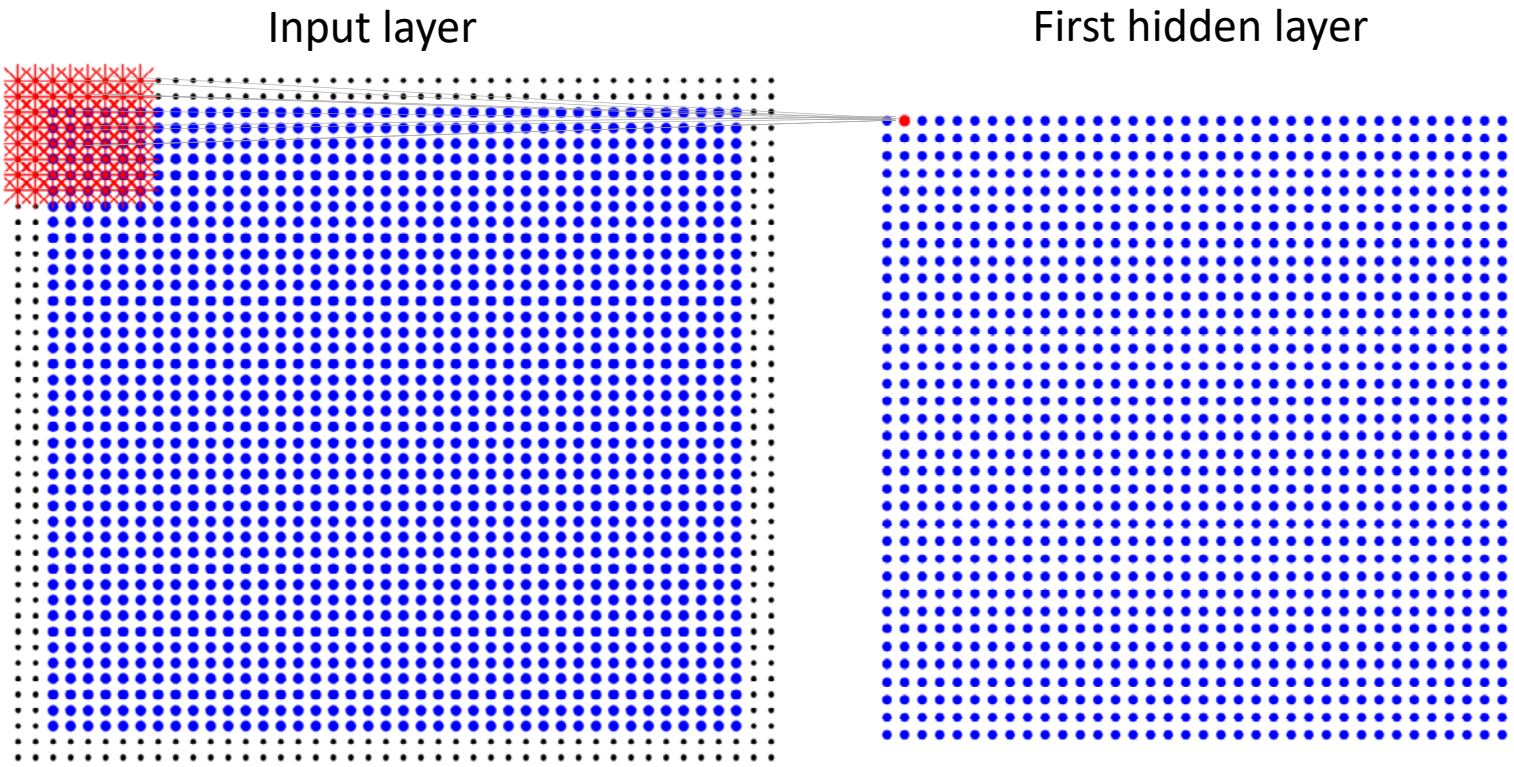
Scanning the image using the LCF

- $K \times K$ pixel input image with $n \times n$ LCF
- Stride length: 1 (pixel by pixel)



Scanning the image using the LCF

- $K \times K$ pixel input image with $n \times n$ LCF
- Zero-padding



The output size (s_x, s_y) of each convolutional layer can be computed as:

$$s_x = \frac{(I_x + 2z_x - f_x)}{L_x} + 1$$
$$s_y = \frac{(I_y + 2z_y - f_y)}{L_y} + 1$$

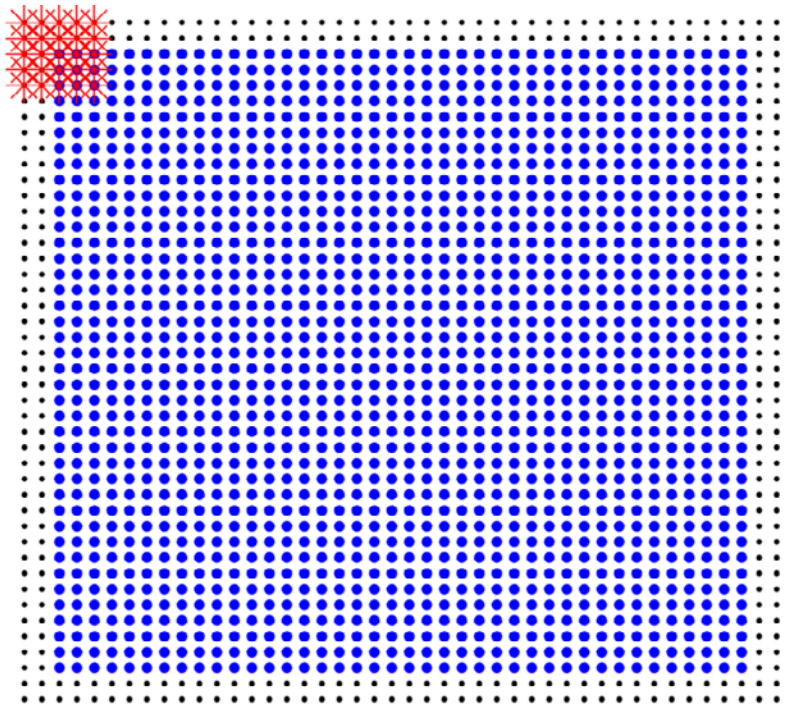
I_x and I_y : input image sizes in x and y directions
z_x and z_y: zero-padding sizes
f_x and f_y : filtering sizes
L_x and L_y: stride lengths.

Let us assume a 240×240 pixel images, a convolutional filter of 7×7 pixels with a stride length of 3 and a 2-pixel zero padding both in x and y directions.
The output will be then 80×80 pixels.

Scanning the image using the LCF

- 40×40 pixel original image with 5×5 LCF
- 2 pixel zero-padding
- Stride length: 1 (pixel by pixel)

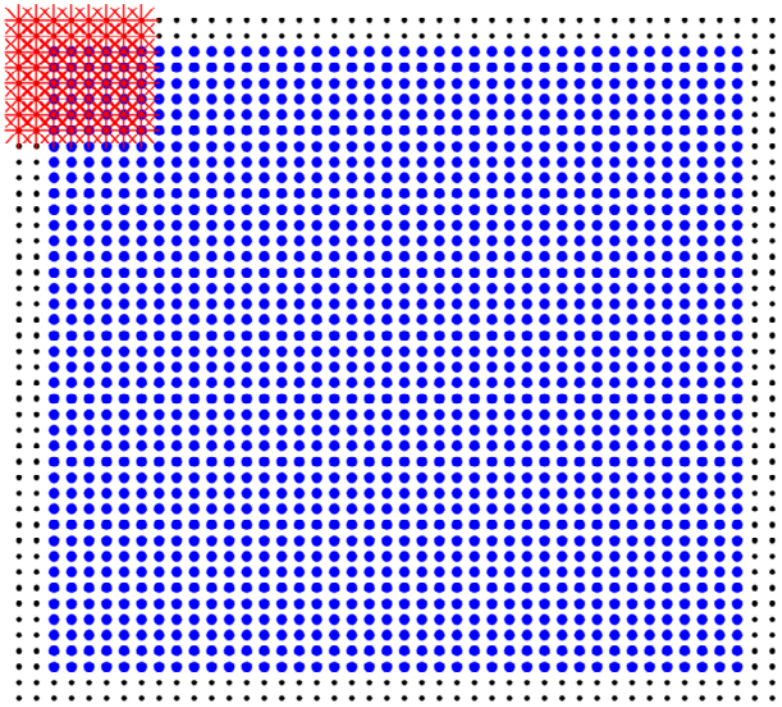
$$S_x = \frac{(I_x + 2z_x - f_x)}{L_x} + 1 = \frac{(40+4-5)}{1} + 1 = 40$$



First hidden layer (40×40 neurons)
NO size compression

Scanning the image using the LCF

- 40×40 pixel original image with 8×8 LCF
- 2 pixel zero-padding
- Stride length: 2



$$s_x = \frac{(I_x + 2z_x - f_x)}{L_x} + 1 = \frac{(40+4-8)}{2} + 1 = 19$$

First hidden layer (19×19 neurons)

To understand and remember: each neuron in the convolutional layer will be forced to have the same weights and the same threshold

Shared weights and biases

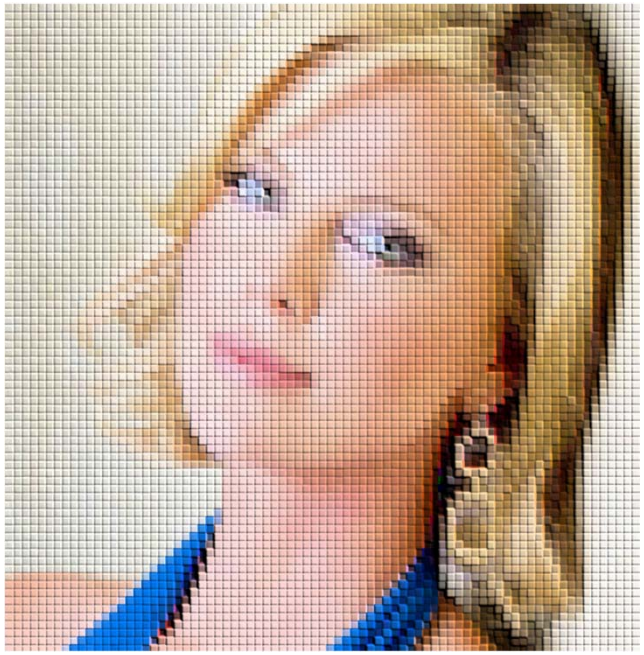
- 40×40 pixel image and 5×5 LCF
- **Feature map**: same weights and biases for all (36×36) the hidden neurons
- For neuron (*j*, *k*)

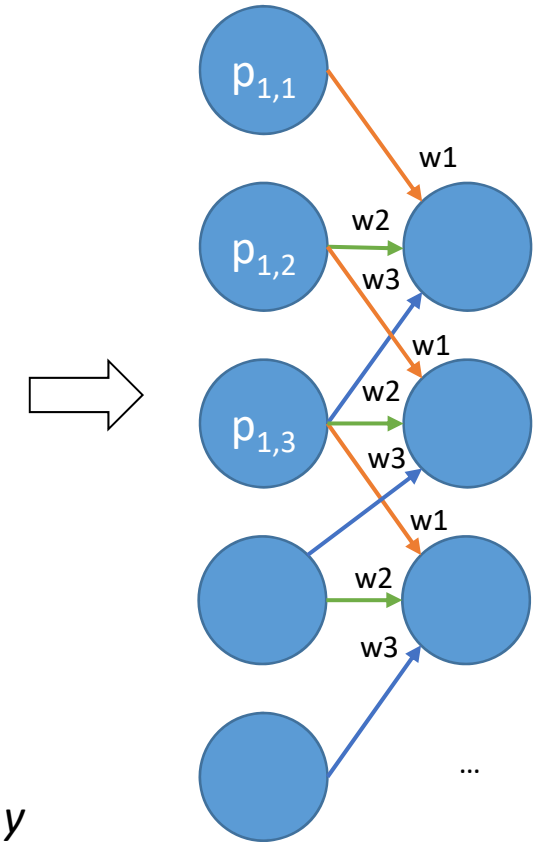
$$u_{j,k} = tanh\left(b + \sum_{l=1}^{5}\sum_{m=1}^{5} w_{l,m} a_{j-1+l,k-1+m}\right)$$

$a_{x,y}$ Input activation at position *x*,*y*

- 5×5 LCF requires only 26 parameters (25 weights + 1 bias/threshold)

Feeding an image into a network

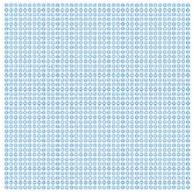




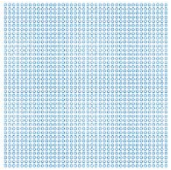
$p_{x,y}$= pixel value at row x and column y

Image flattening (transform into a vector) with pixel connectivity constraint

40×40=1600 inputs



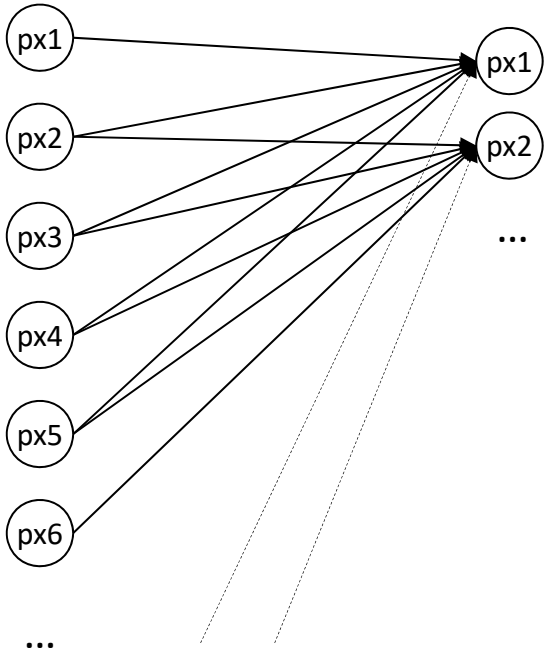
36x36=1296 units



40×40 pixel input image with 5×5 LCF
stride 1 pixel and no padding

Input layer (row order)

Hidden layer



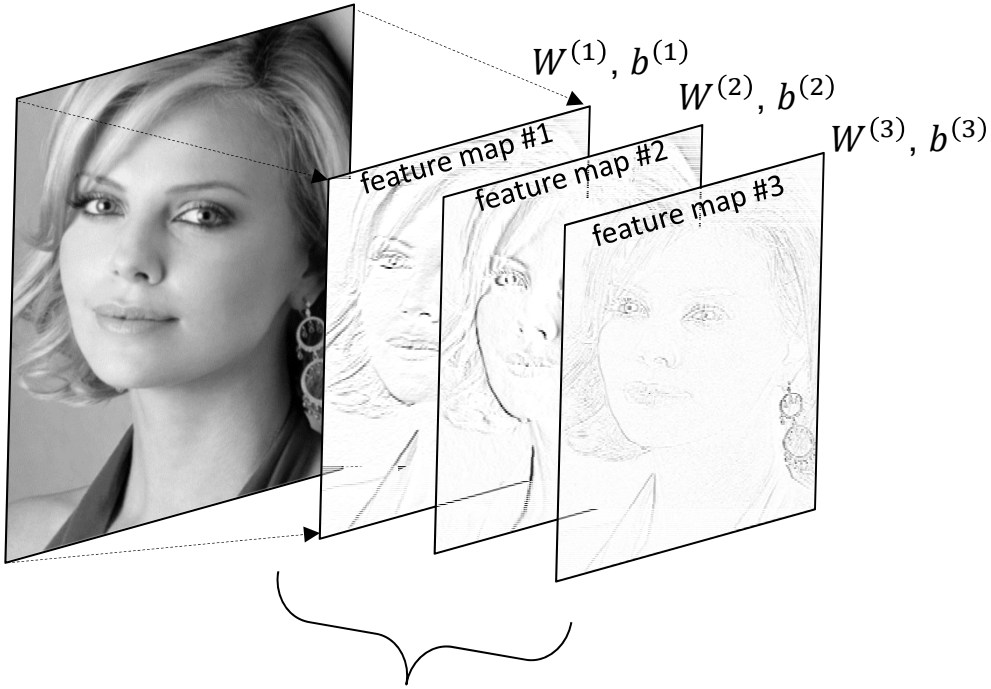
From neuron in the next rows

Wiring the net

Not fully connected

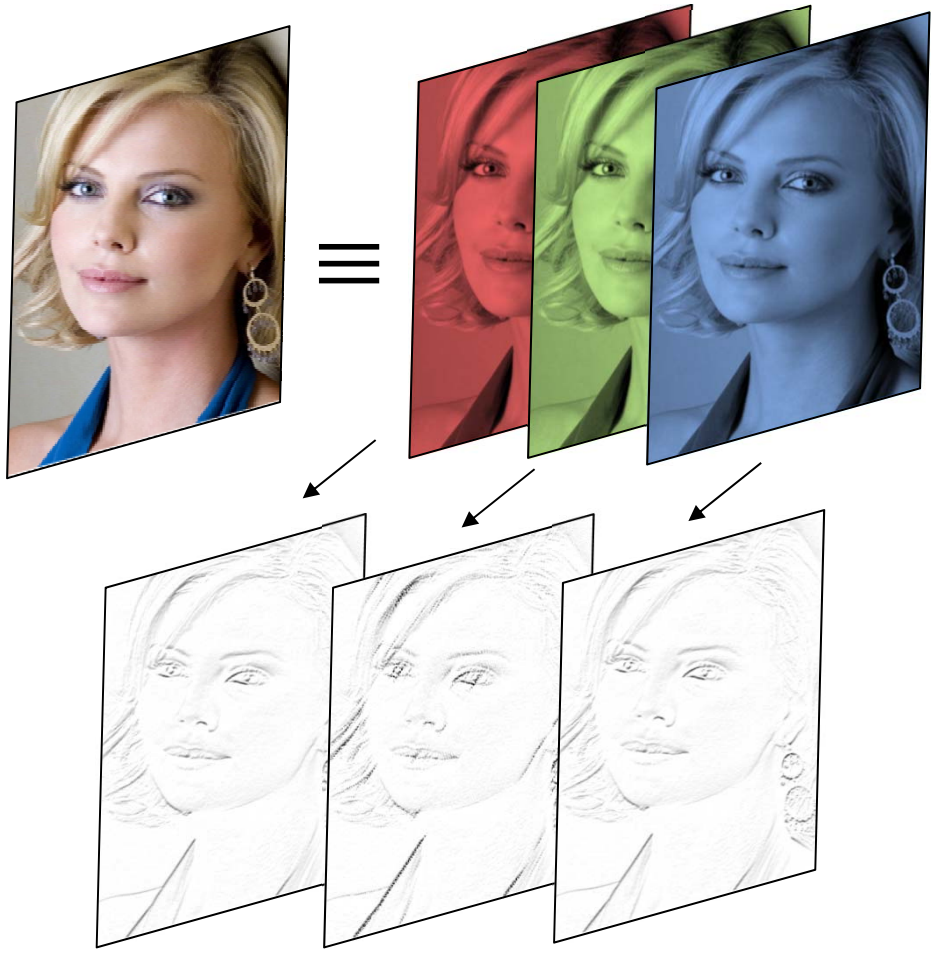
Many feature maps

- A *convolutional layer can include* several different feature maps, each corresponding to a filter
- E.g. $n \times n$ weights and a single shared bias identify one feature map
- For instance: oriented line detectors but they can be used to learn more complex patterns



convolutional layer (e.g. horizontal lines, vertical lines, oriented lines, ..)

RGB images

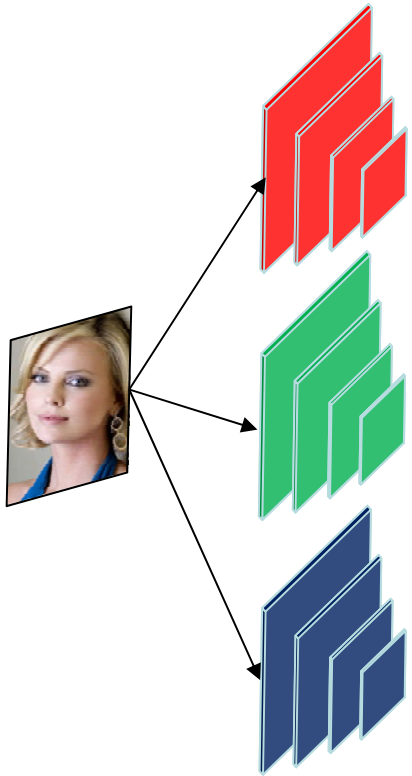


Feature map for red channel

Feature map for green channel

Feature map for blue channel

Three parallel lines of processing

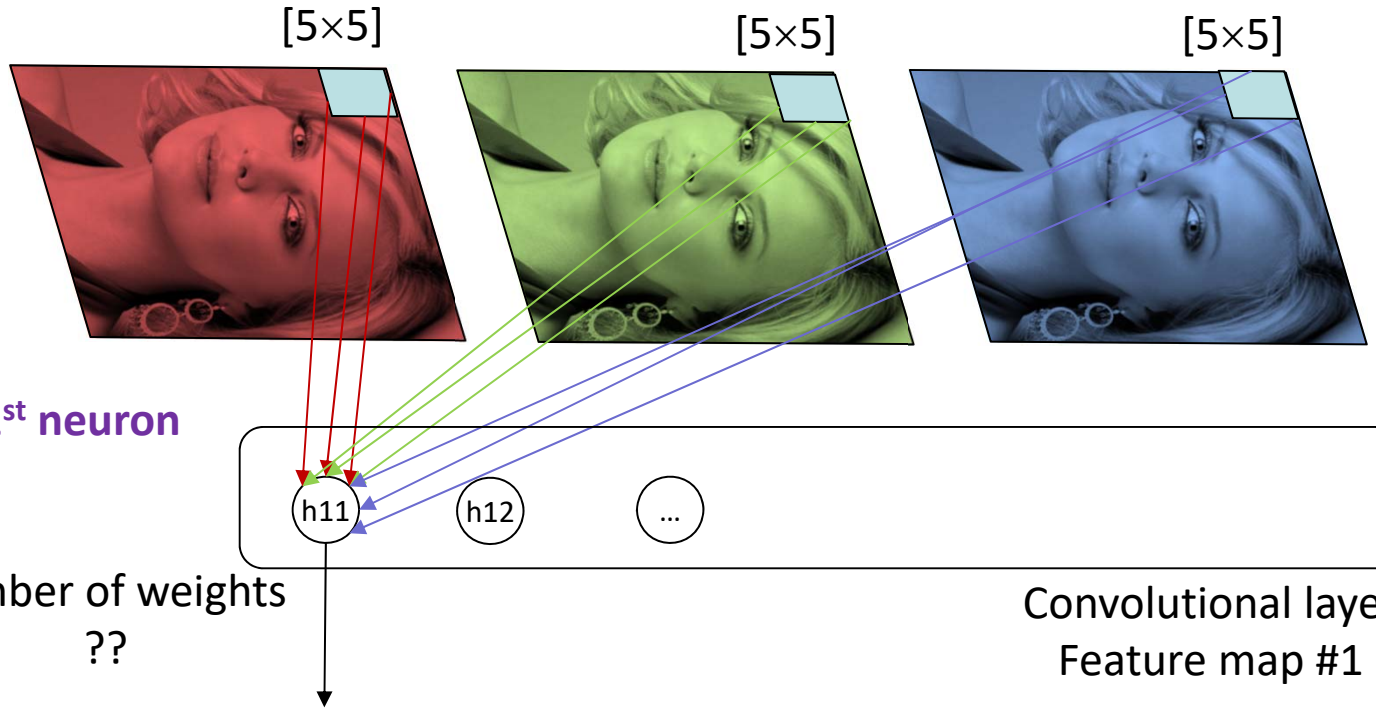


DepthwiseConv2D

RGB images

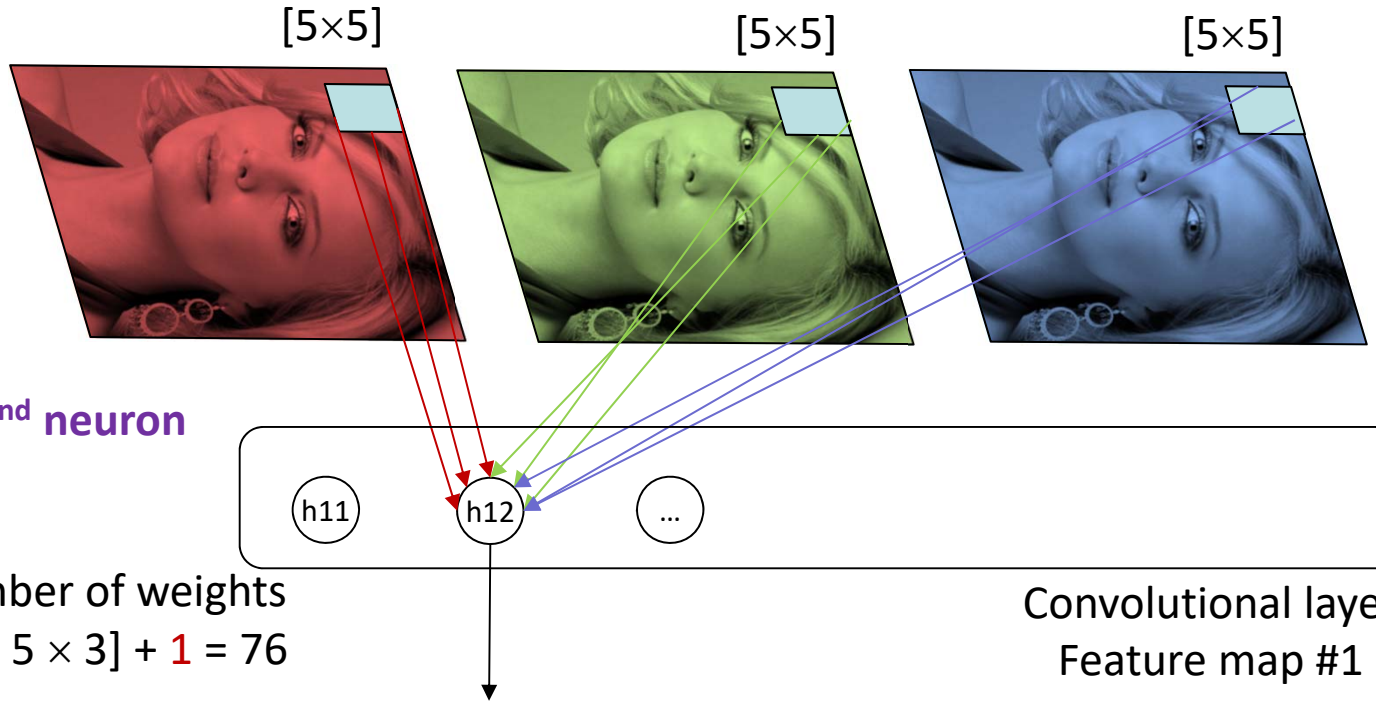
40×40 pixel × 3 channels.

??



Output of the 1st neuron in the 1st feature map in the 1st convolutional layer

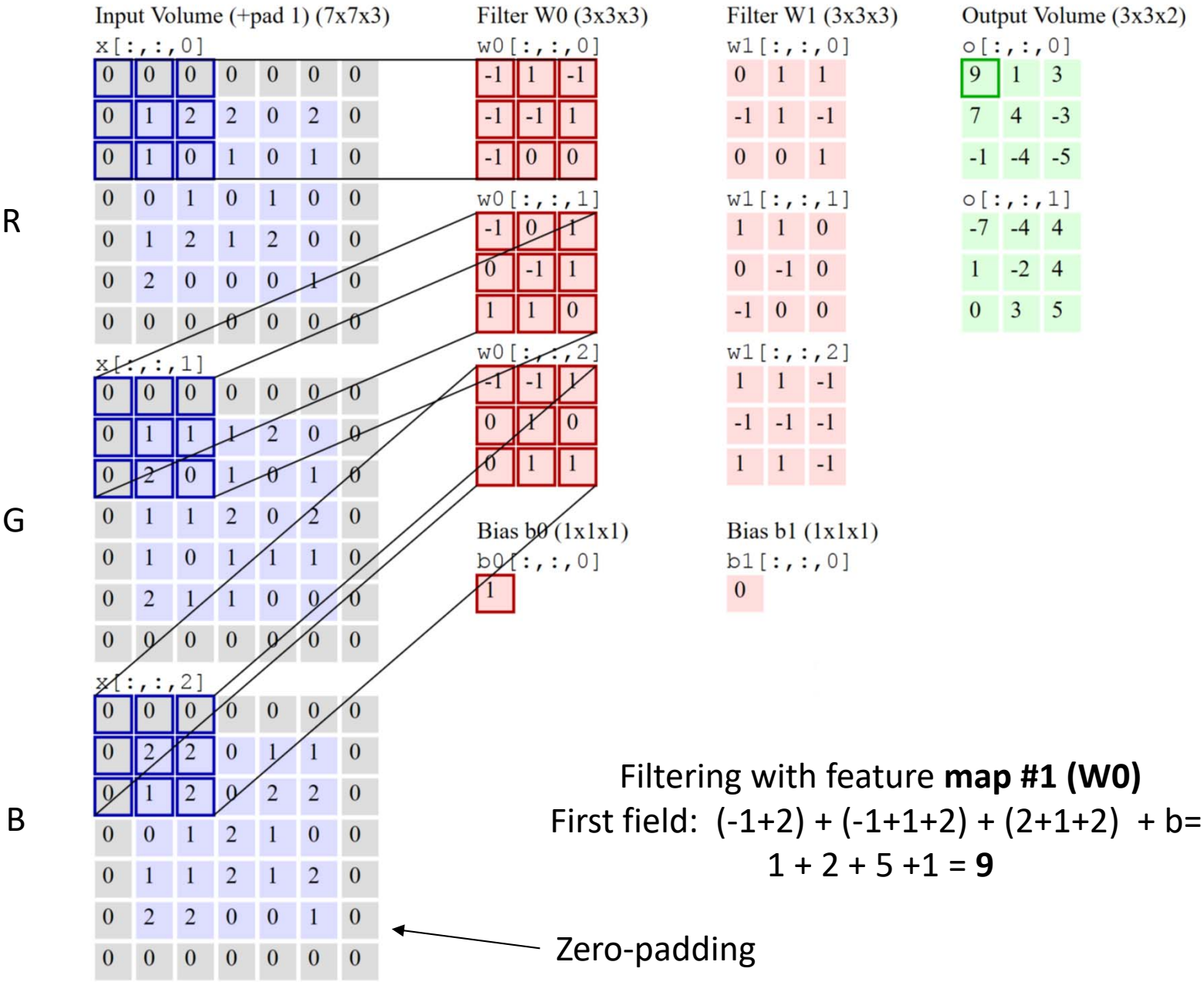
RGB images



Output of the 2nd neuron in the 1st feature map in the 1st convolutional layer

Shared weights

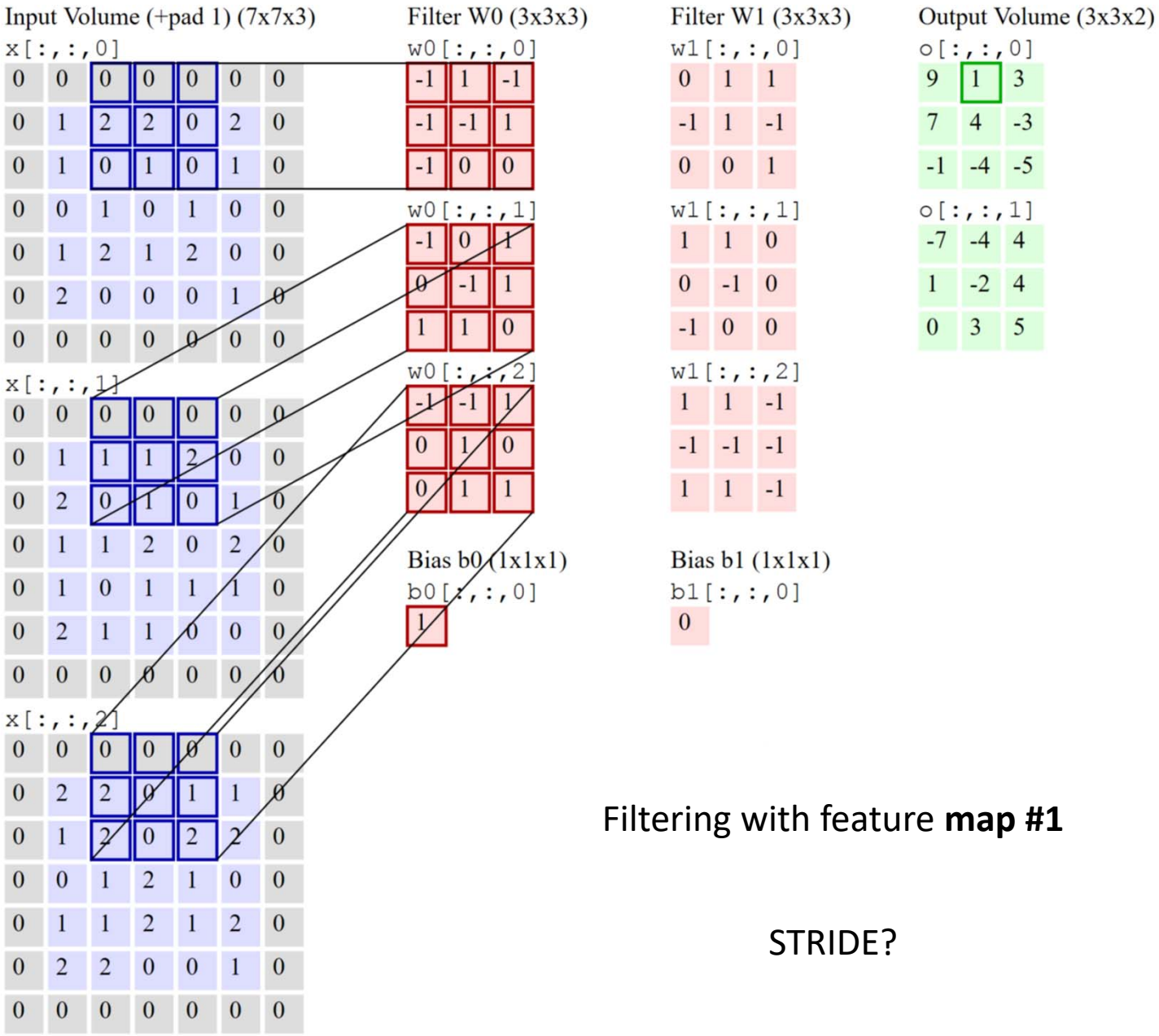
75 weights + 1 bias parameter

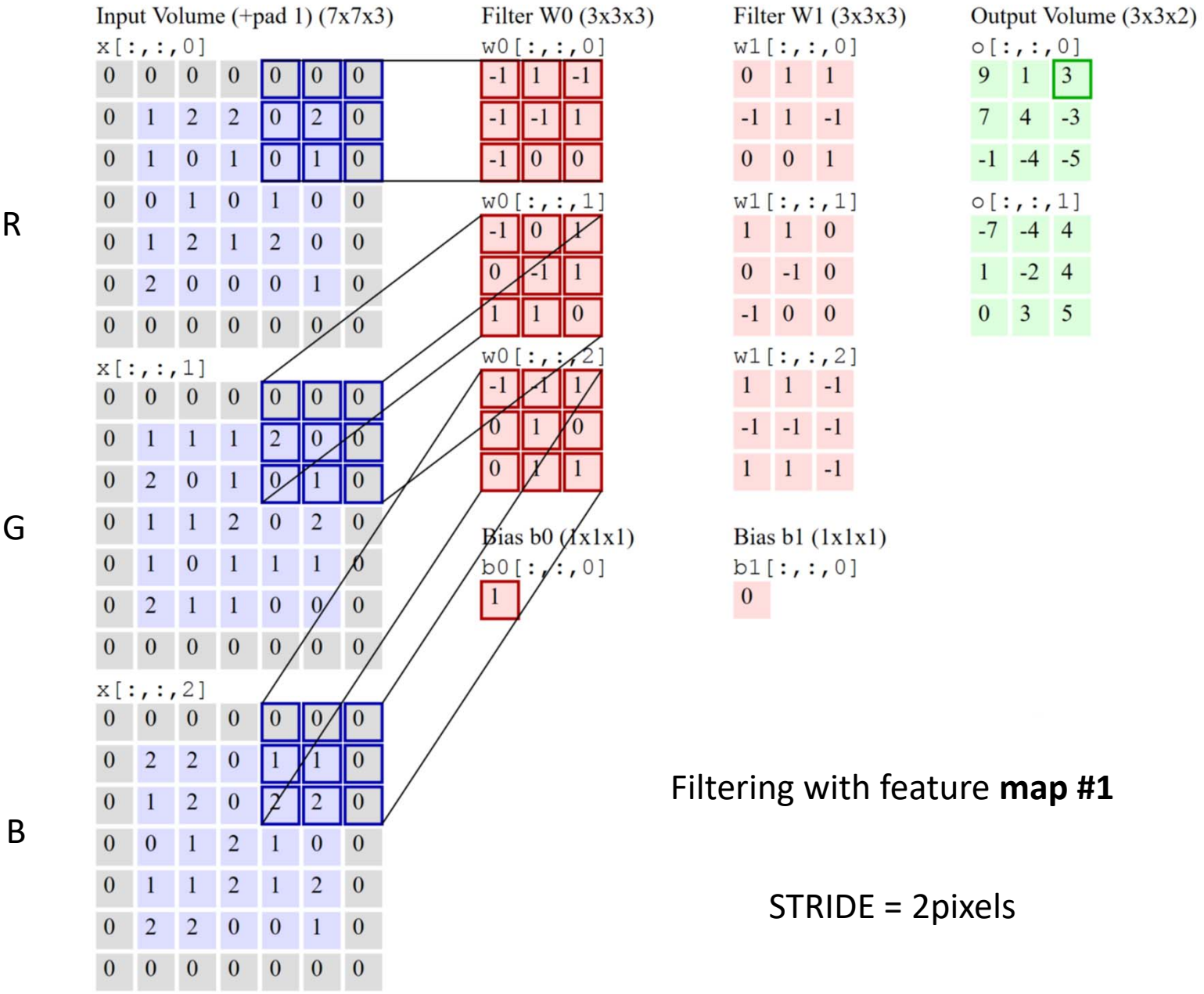


Filtering with feature **map #1 (W0)**

First field: (-1+2) + (-1+1+2) + (2+1+2) + b=

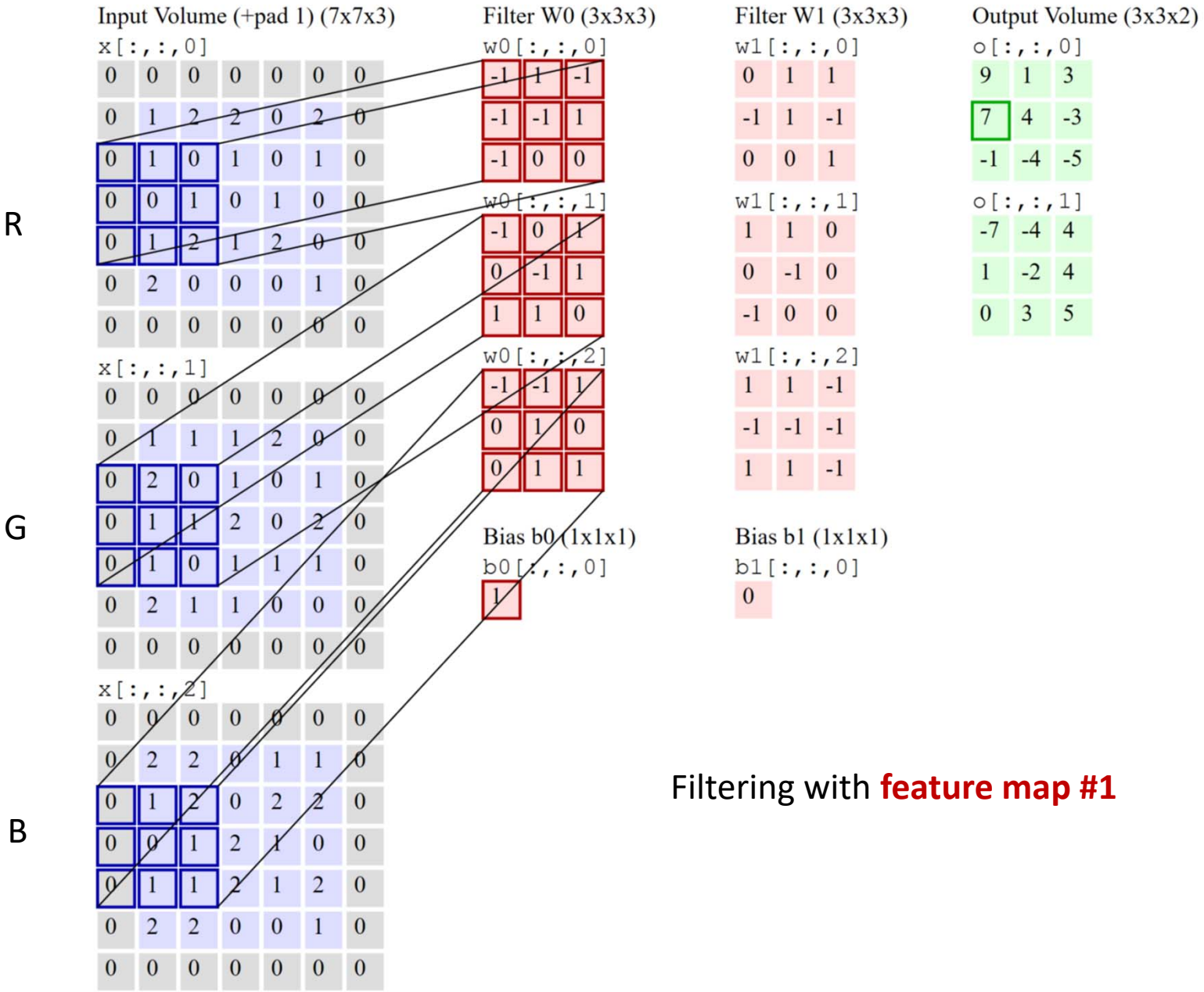
1 + 2 + 5 +1 = **9**


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 0 2 0
0 1 0 1 0 1 0
0 0 1 0 1 0 0
0 1 2 1 2 0 0
0 2 0 0 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 0 0
0 2 0 1 0 1 0
0 1 1 2 0 2 0
0 1 0 1 1 1 0
0 2 1 1 0 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 2 0 1 1 0
0 1 2 0 2 2 0
0 0 1 2 1 0 0
0 1 1 2 1 2 0
0 2 2 0 0 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 1 -1
-1 -1 1
-1 0 0
w0[:,:,1]
-1 0 1
0 -1 1
1 1 0
w0[:,:,2]
-1 -1 1
0 1 0
0 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
0 1 1
-1 1 -1
0 0 1
w1[:,:,1]
1 1 0
0 -1 0
-1 0 0
w1[:,:,2]
1 1 -1
-1 -1 -1
1 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
9 1 3
7 4 -3
-1 -4 -5
o[:,:,1]
-7 -4 4
1 -2 4
0 3 5
R
G
B
Filtering with feature **map #1**
STRIDE?

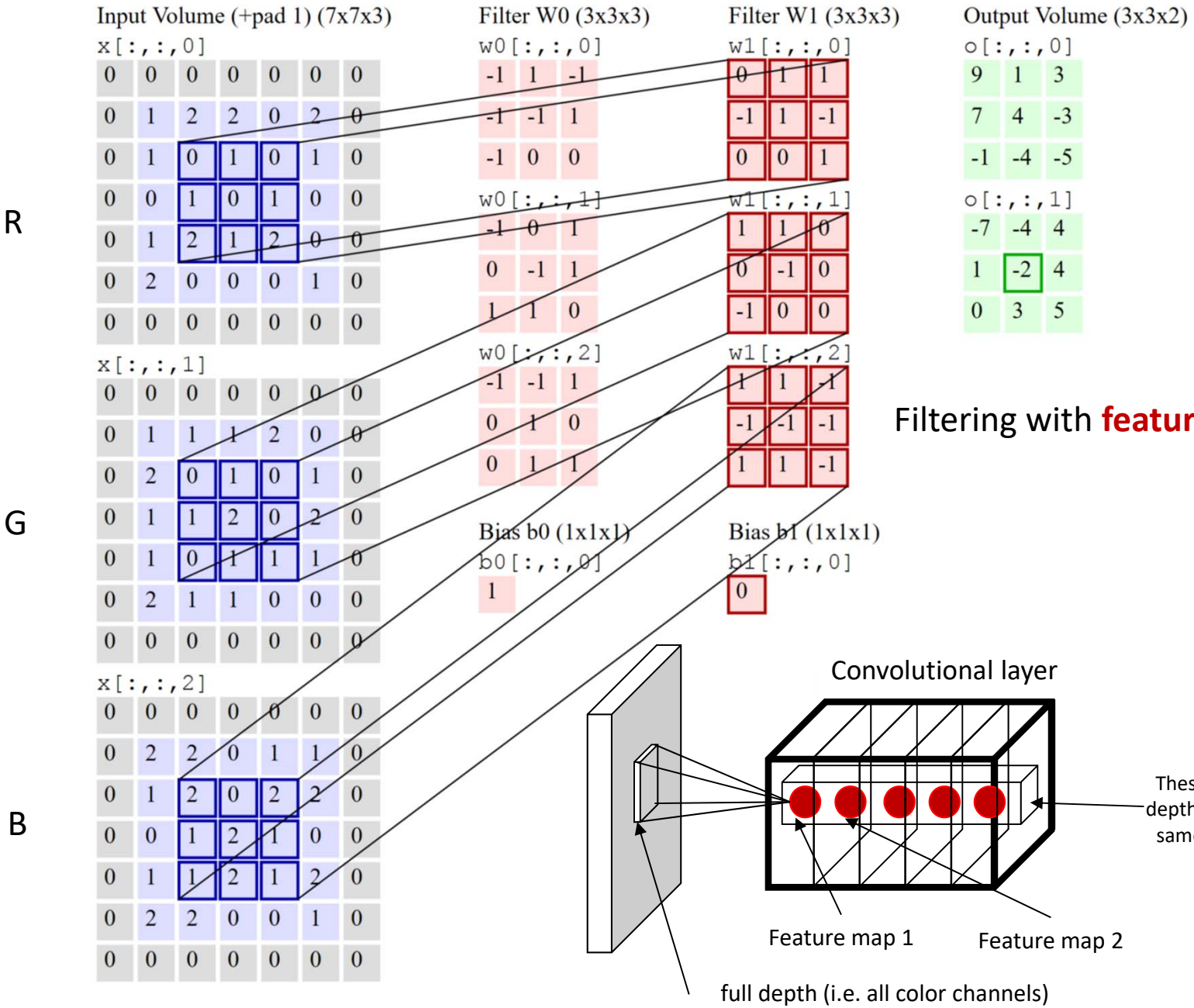


Filtering with feature **map #1**

STRIDE = 2pixels



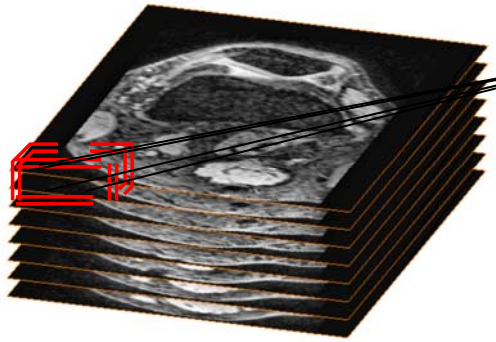
Filtering with **feature map #1**


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 0 2 0
0 1 0 1 0 1 0
0 0 1 0 1 0 0
0 1 2 1 2 0 0
0 2 0 0 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 0 0
0 2 0 1 0 1 0
0 1 1 2 0 2 0
0 1 0 1 1 1 0
0 2 1 1 0 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 2 0 1 1 0
0 1 2 0 2 2 0
0 0 1 2 1 0 0
0 1 1 2 1 2 0
0 2 2 0 0 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 1 -1
-1 -1 1
-1 0 0
w0[:,:,1]
-1 0 1
0 -1 1
1 1 0
w0[:,:,2]
-1 -1 1
0 1 0
0 1 1
Filter W1 (3x3x3)
w1[:,:,0]
0 1 1
-1 1 -1
0 0 1
w1[:,:,1]
1 1 0
0 -1 0
-1 0 0
w1[:,:,2]
1 1 -1
-1 -1 -1
1 1 -1
Output Volume (3x3x2)
o[:,:,0]
9 1 3
7 4 -3
-1 -4 -5
o[:,:,1]
-7 -4 4
1 -2 4
0 3 5
Bias b0 (1x1x1)
b0[:,:,0]
1
Bias b1 (1x1x1)
b1[:,:,0]
0
R
G
B
Filtering with feature map #2
Convolutional layer
These neurons along the depth are all looking at the same region in the input
Feature map 1
Feature map 2
full depth (i.e. all color channels)

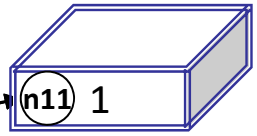
Volumetric biomedical images

3D scan: 33800 voxels

Size of the input: 65×65×8



Neuron #1 of the feature map #1



50 weights +
1 threshold

How many neurons in the map?

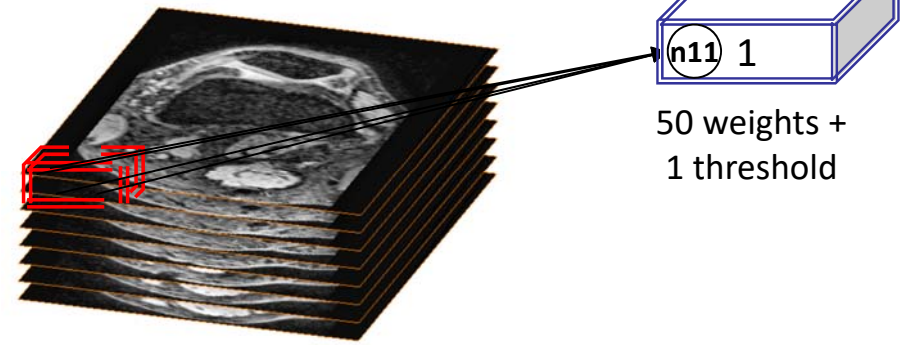
Filter size: 5×5×2

- 5 pixel in x direction
- 5 pixel in y direction
- 2 slices

Volumetric biomedical images

3D scan: 33800 voxels

Size of the input: 65×65×8



Neuron #1 of the feature map #1

50 weights +
1 threshold

How many neurons in the map?

$$s_x = \frac{(I_x + 2z_x - f_x)}{L_x} + 1 = 31$$

$$s_y = \frac{(I_y + 2z_y - f_y)}{L_y} + 1 = 31$$

$$s_z = \frac{(n_s + 2z_z - f_z)}{L_z} + 1 = 4$$

3844

Filter size: 5×5×2
- 5 pixel in x direction
- 5 pixel in y direction
- 2 slices

Padding: none

Stride: 2 voxel

I_x and I_y : input image sizes in x and y directions

z_x, z_y and z_z : zero-padding sizes

f_x, f_y and f_z: filtering sizes

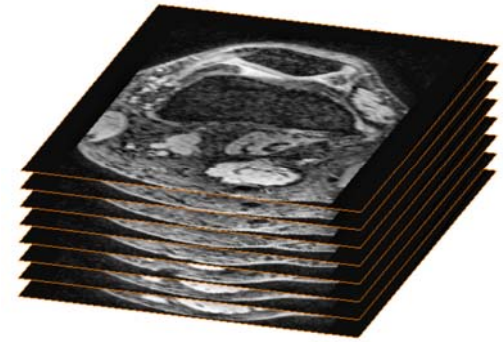
L_x, L_y, and L_z : stride lengths

n_s : number of slices

Volumetric biomedical images

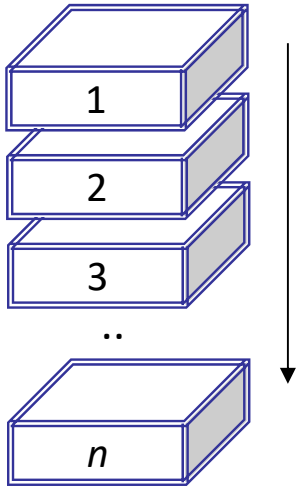
3D scan: 33800 voxels

Size of the input: 65×65×8



Filter size: 5×5×2
- 5 pixel in x direction
- 5 pixel in y direction
- 2 slices

First convolutional layer



n feature maps (*n* different filters)
Each one endowed with 3844 neurons

How many free parameter in the layer?

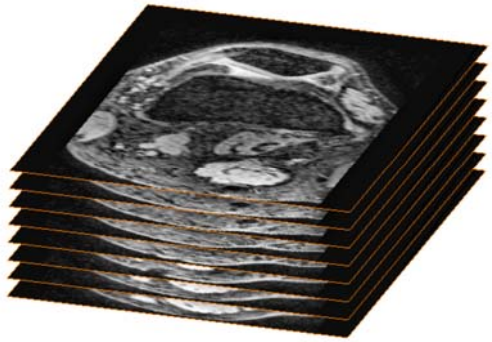
51**n*

and the next layer?

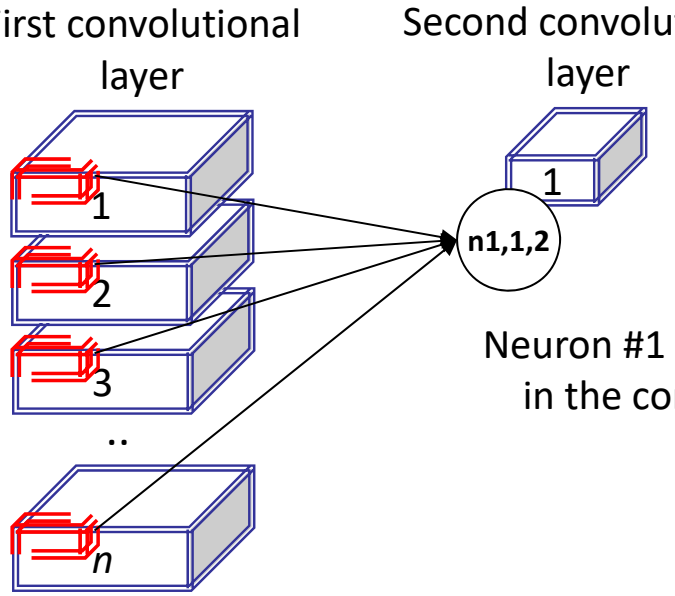
Volumetric biomedical images

3D scan: 33800 voxels

Size of the input: 65×65×8



Filter size: 3×3×2
- 3 pixel in x direction
- 3 pixel in y direction
- 2 slices



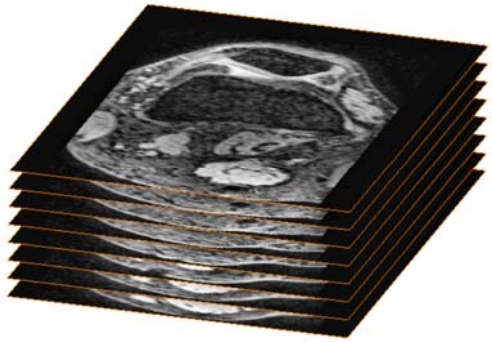
Neuron #1 of the feature map #1 in the convolutional layer #2

n feature maps (*n* different filters)
Each one endowed with 3844 neurons

Volumetric biomedical images

3D scan: 33800 voxels

Size of the input: 65×65×8



Filter size: 3×3×2

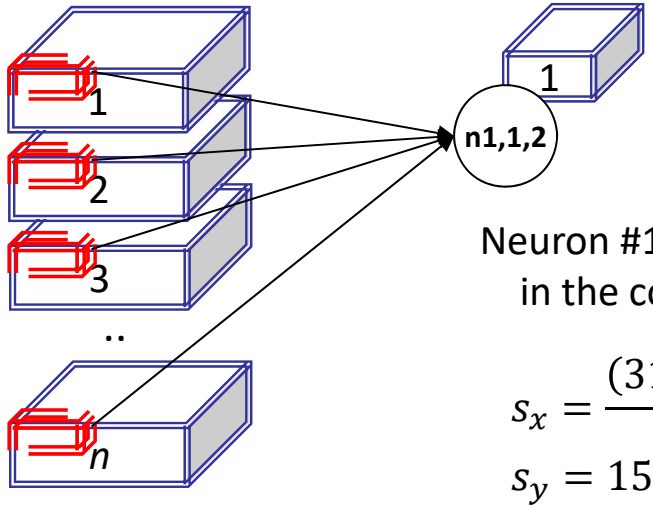
3 pixel in x direction

3 pixel in y direction

2 slices

First convolutional layer

Second convolutional layer



n feature maps (*n* different filters)

Each one endowed with 3844 neurons

Neuron #1 of the feature map #1 in the convolutional layer #2

$$s_x = \frac{(31 + 0 - 3)}{2} + 1 = 15$$

$$s_y = 15$$

$$s_z = 2$$

450 neurons

Number of parameters for :

3*3*2*n +1

Conv+ReLU+Pooling

INPUT
[40×40×3]

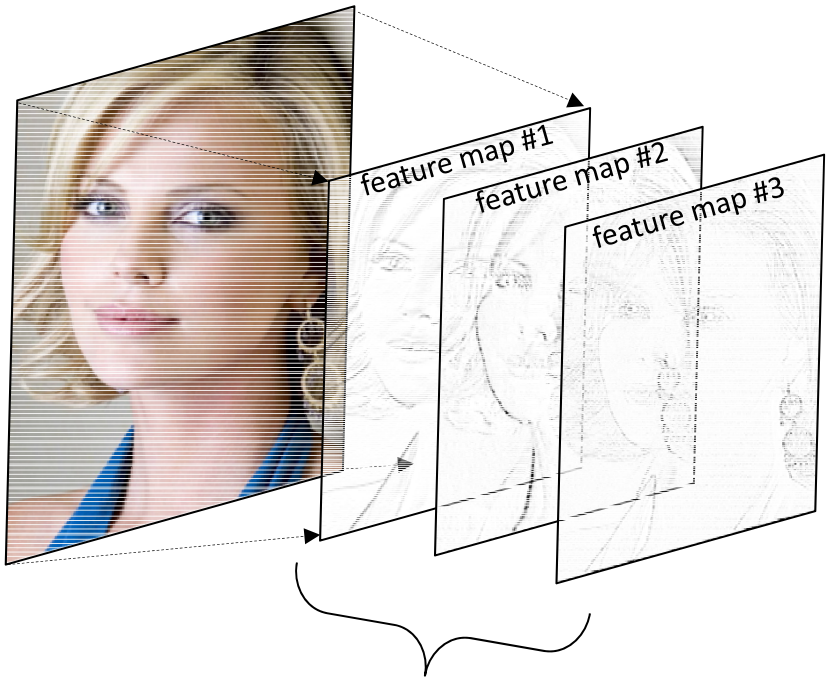
CONV layer
[36×36×12]

e.g. we decided to use 12 filters

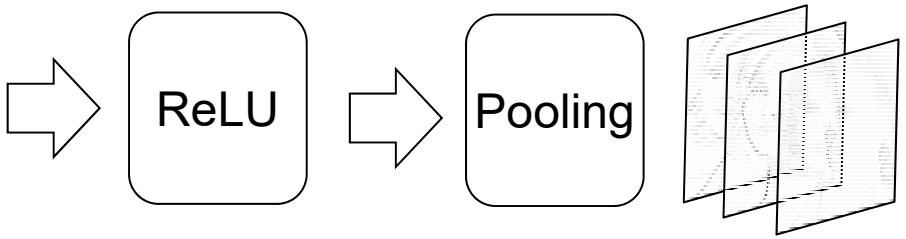
ReLU layer
[36×36×12]

POOL layer
[18×18×12]

size 2x2 applied with a stride of 2
downsamples

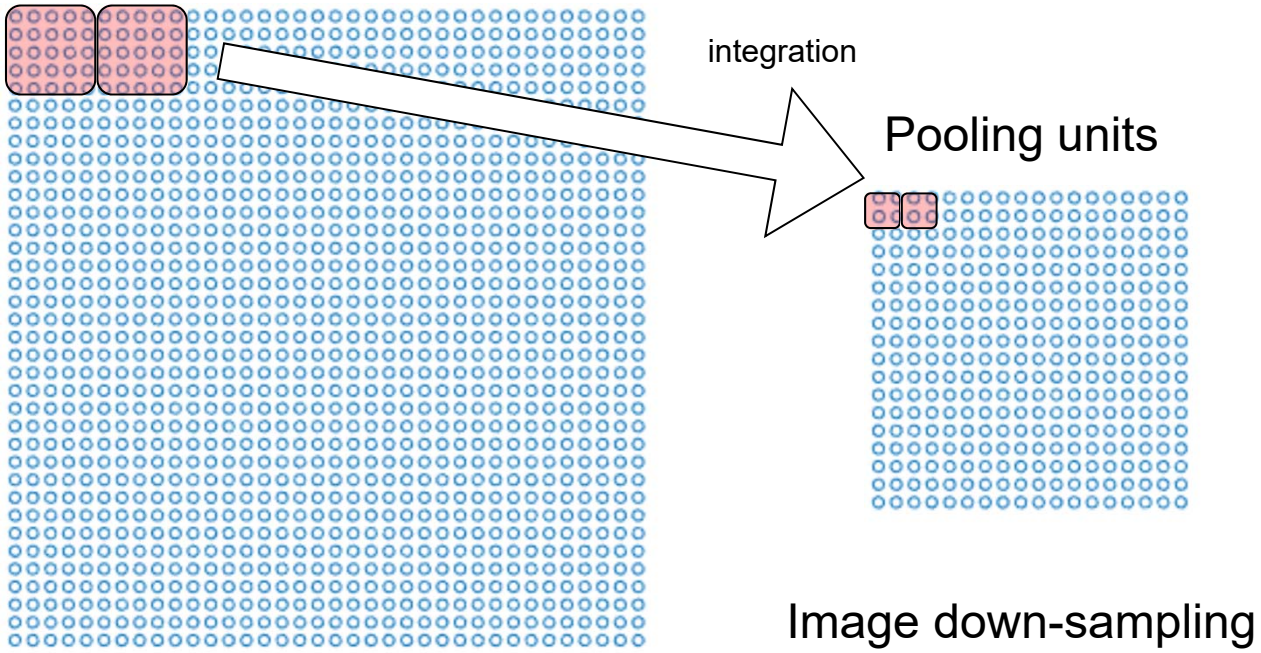


convolutional layer

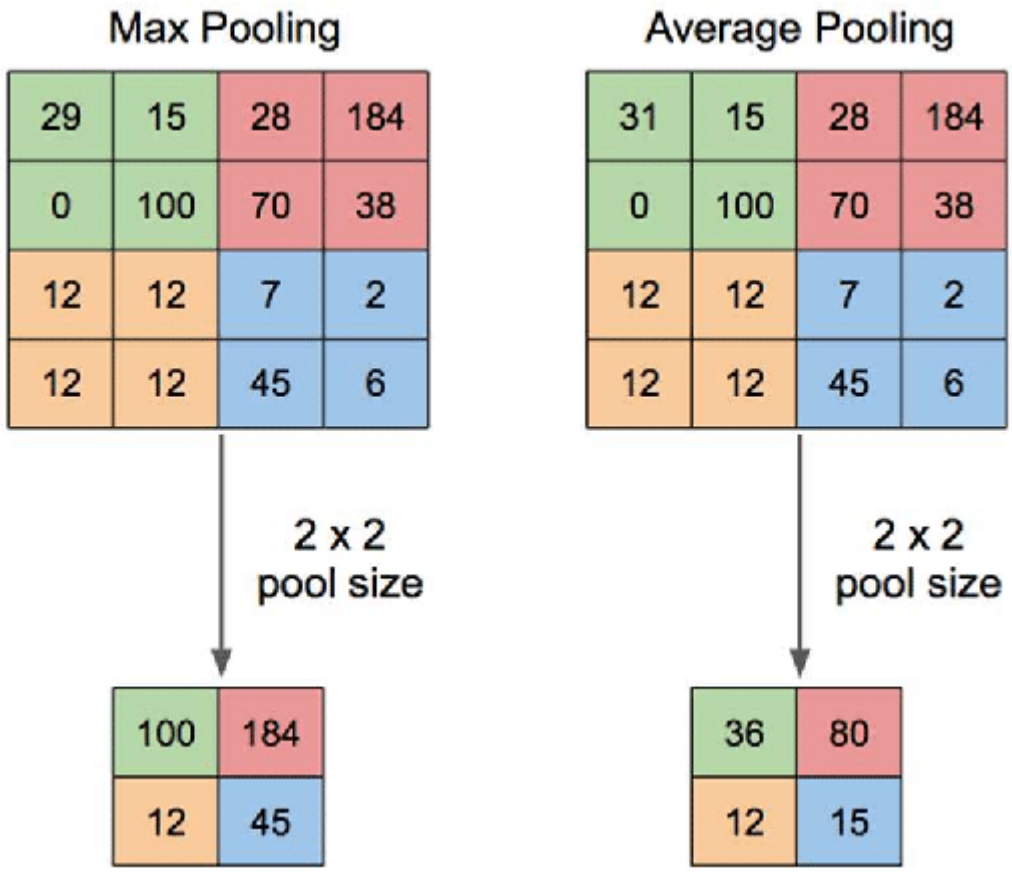


Pooling layer (integration/smoothing)

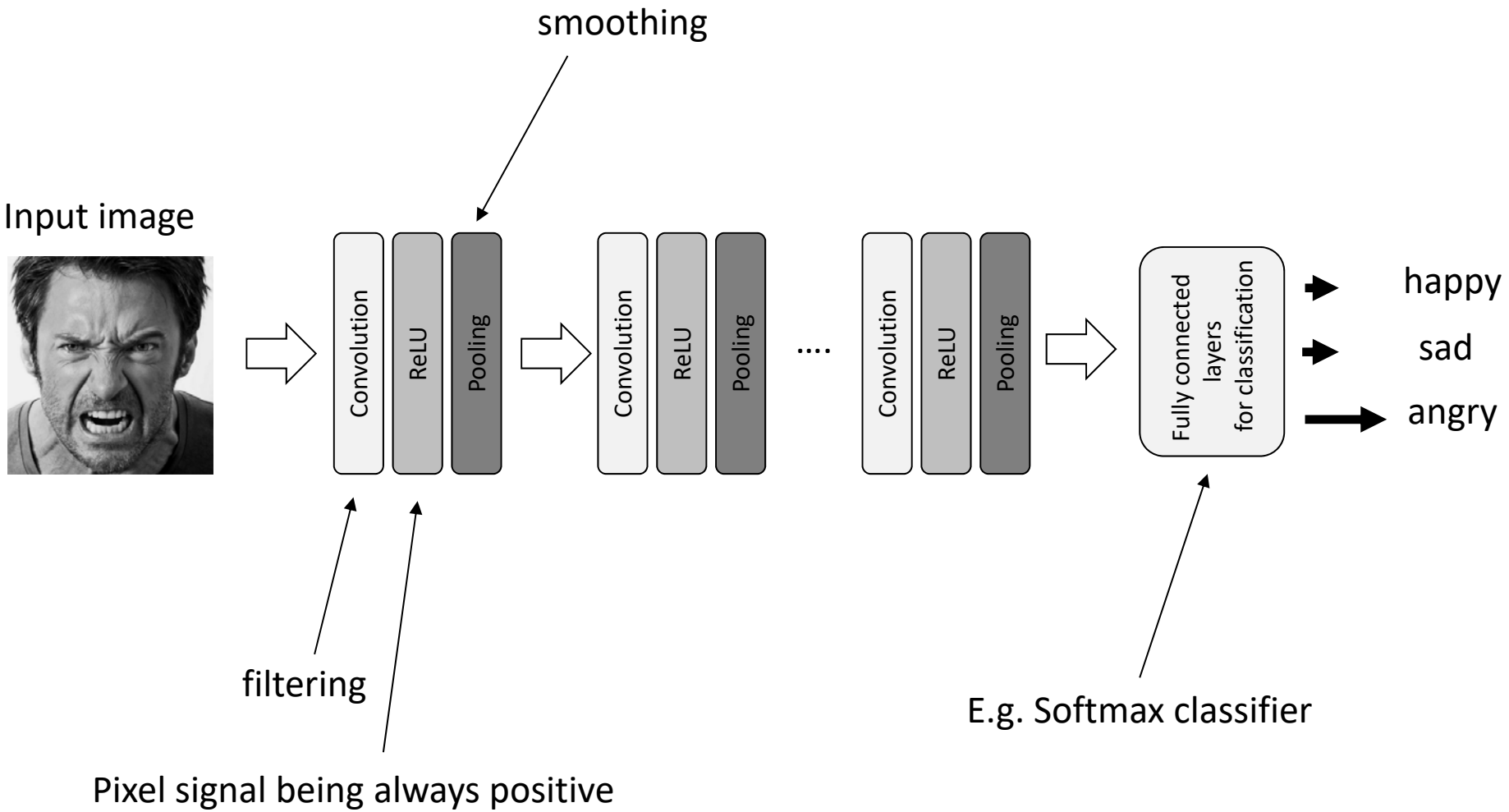
Hidden layer (output after feature map and ReLU)



Pooling layer (integration/smoothing)



ANN able to recognize emotions



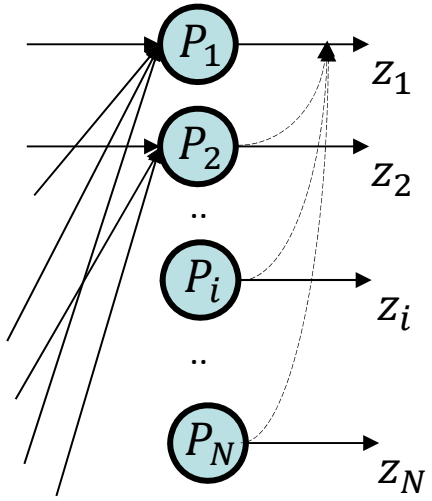
Softmax classifier

Only two classes: traditional logsig classifer

- One output neuron with sigmoidal activation
- Training: Delta/Backprop rule applied to error function

More than two classes: Softmax classifier

- Normalized exponential activation
- Number of output neurons as much as the number of classes
- Training: cost function is the negative log probability of the correct label and cross-entropy regime (very similar to KL divergence)
- Expected output signals:
  1 0 0 0 .. 0: class #1
  0 1 0 0 .. 0: class #2
  0 0 1 0 .. 0: class #3
  ..
  ..
  0 0 0 0.. 1: class #N



$$z_i = \frac{e^{P_i}}{\sum_j^N e^{P_j}}$$

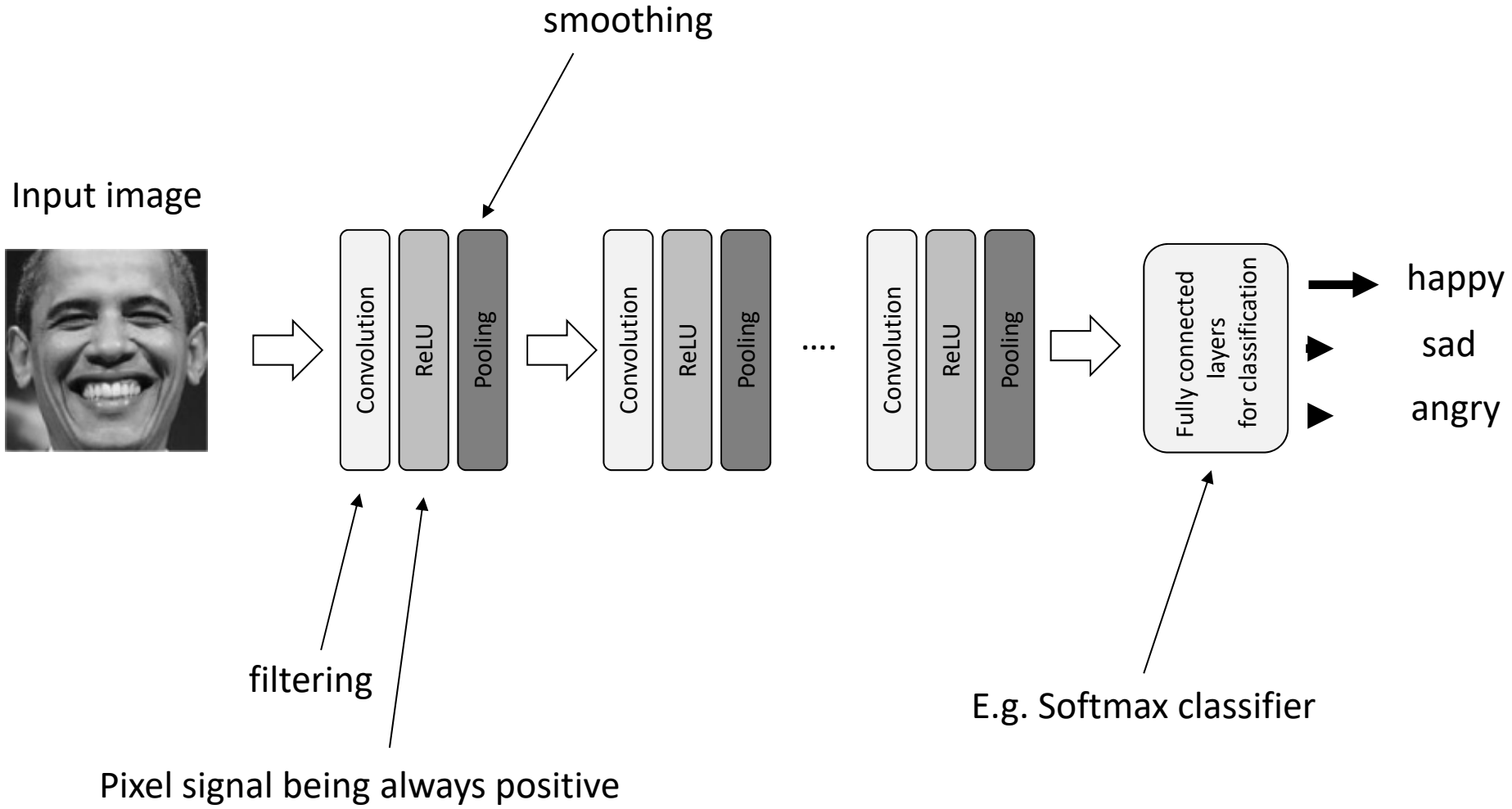
$$\frac{\partial z_i}{\partial P_i} = z_i\,(1 - z_i)$$

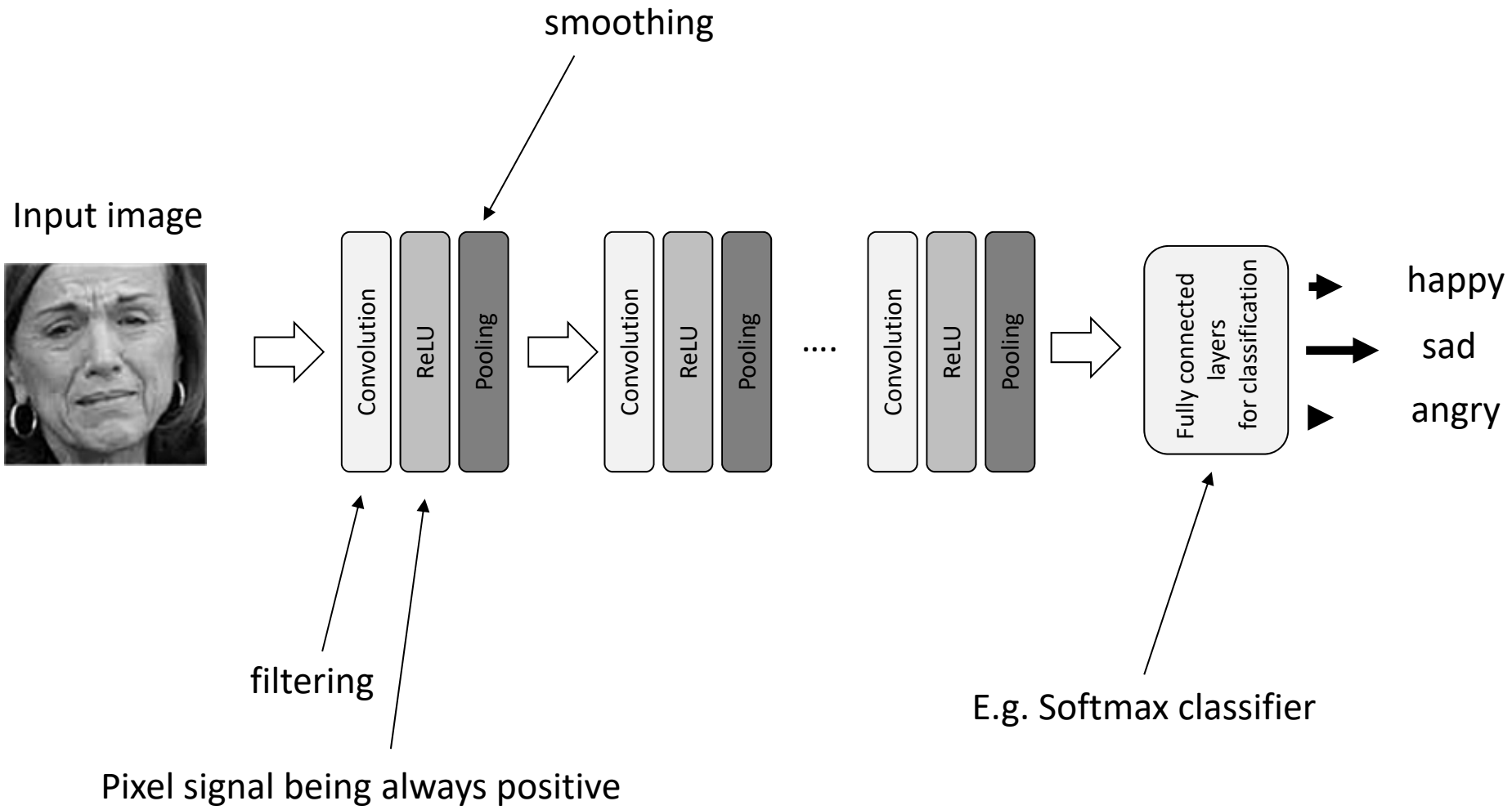
$$E = -\sum_j^N t_j \ln z_j$$

$$\frac{\partial E}{\partial P_i} = \sum_j^N \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial P_i} = z_i - t_i$$

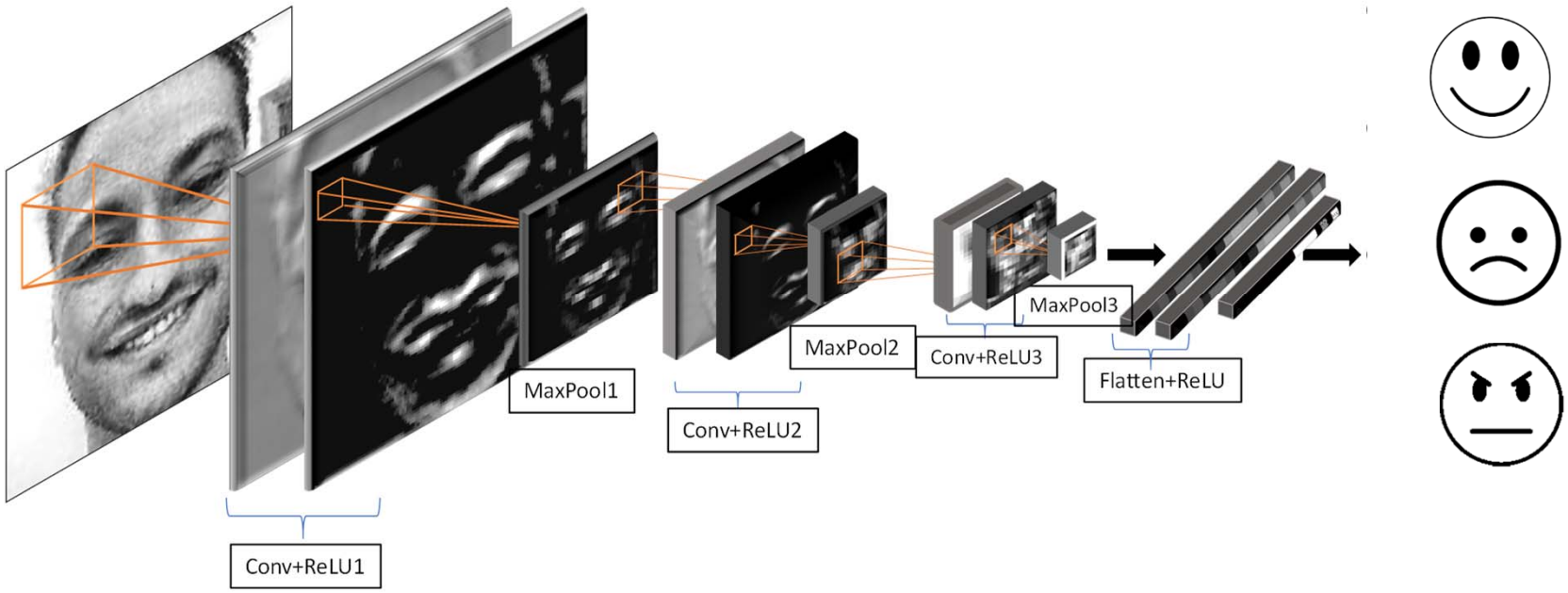
"maps" a *N*-dimensional vector of arbitrary real values to a *N*-dimensional vector of real values in the range (0, 1) that add up to 1.

Apply deep analysis to classification problem



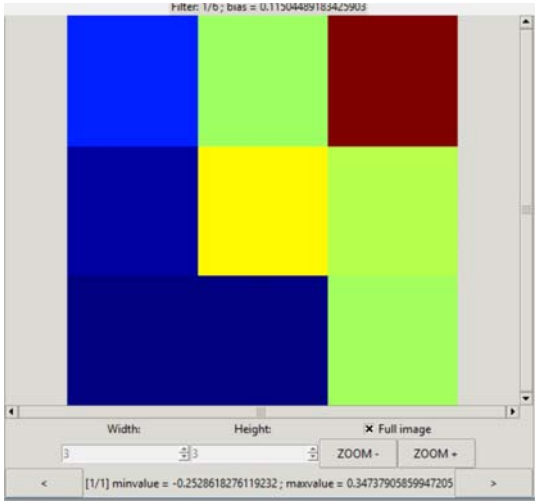

smoothing
Input image
Convolution
ReLU
Pooling
Convolution
ReLU
Pooling
....
Convolution
ReLU
Pooling
Fully connected layers for classification
happy
sad
angry
filtering
E.g. Softmax classifier
Pixel signal being always positive

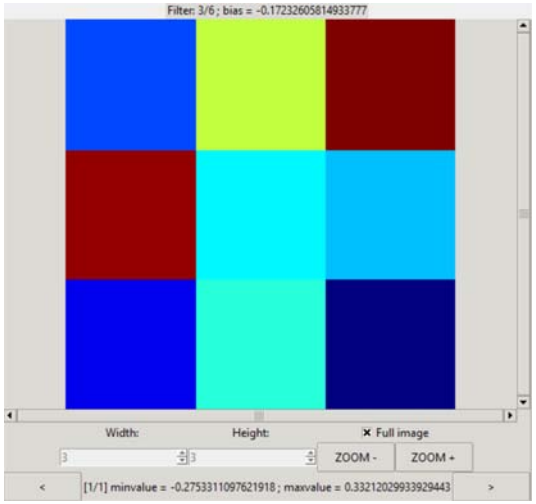
ANN able to recognize emotions

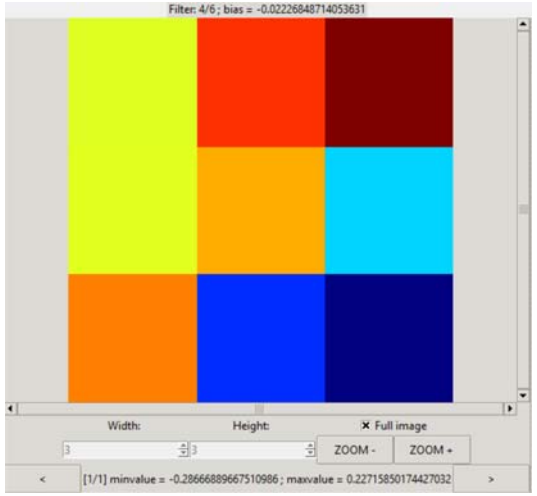


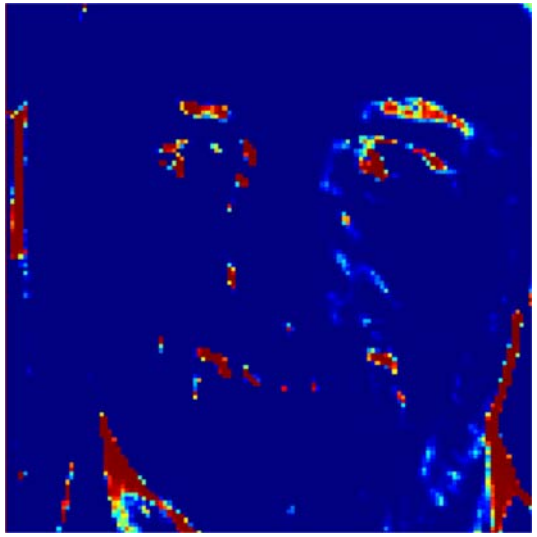
Training data: image with the corresponding label
Loss function for multi-class classification: Categorical Cross-Entropy

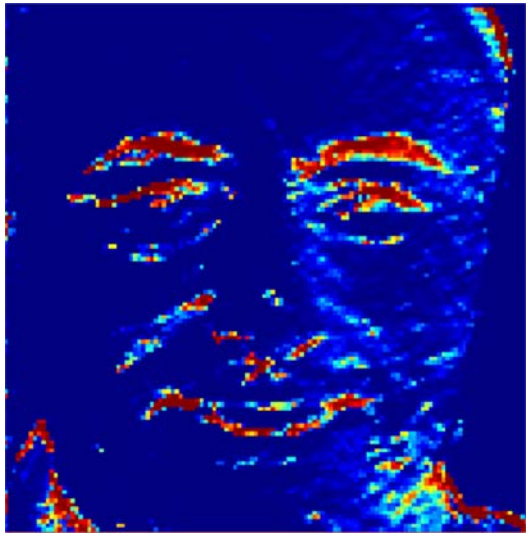
Three feature maps of the 1st layer

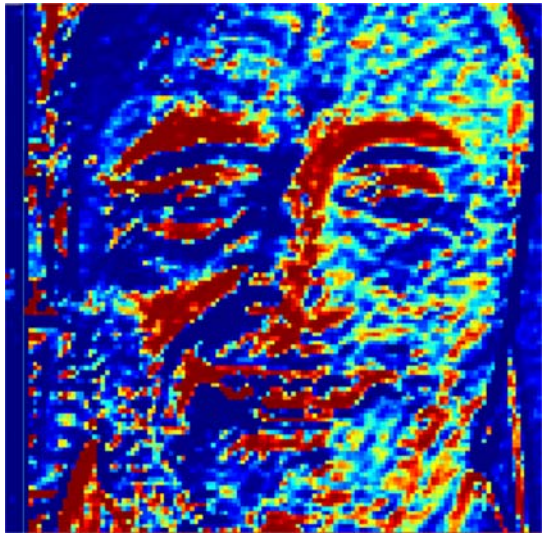












Three feature maps of the 2nd layer