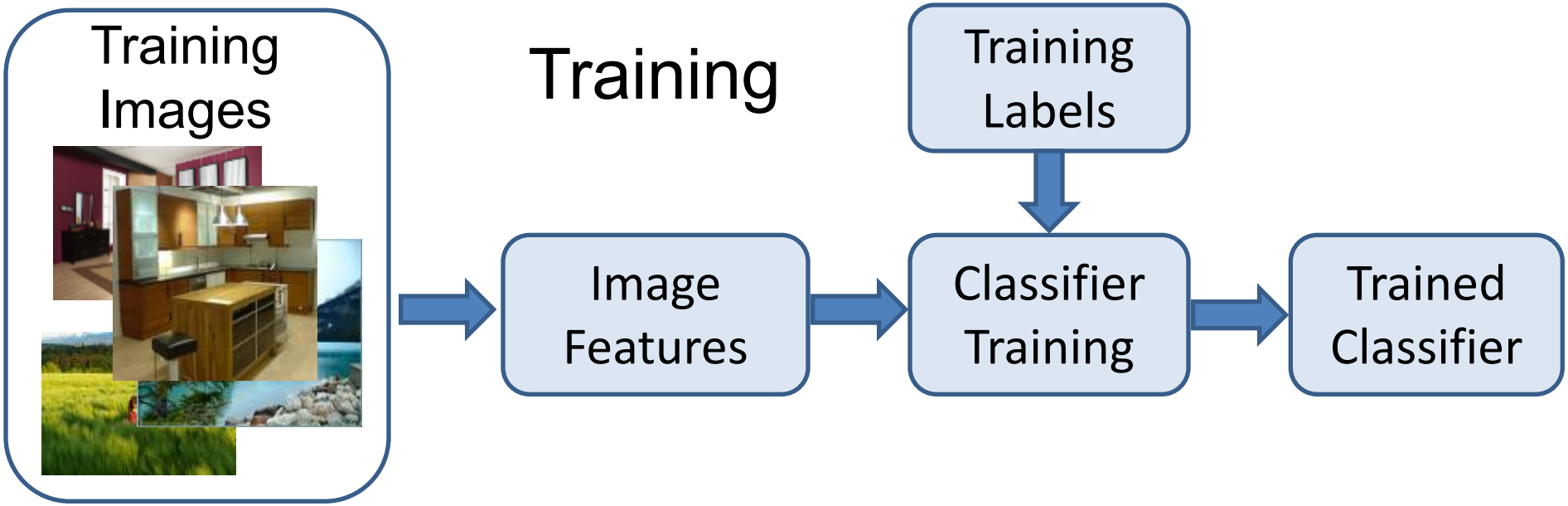
CSE 152: Computer Vision

Hao Su

Lecture 7: Convolutional Neural Networks

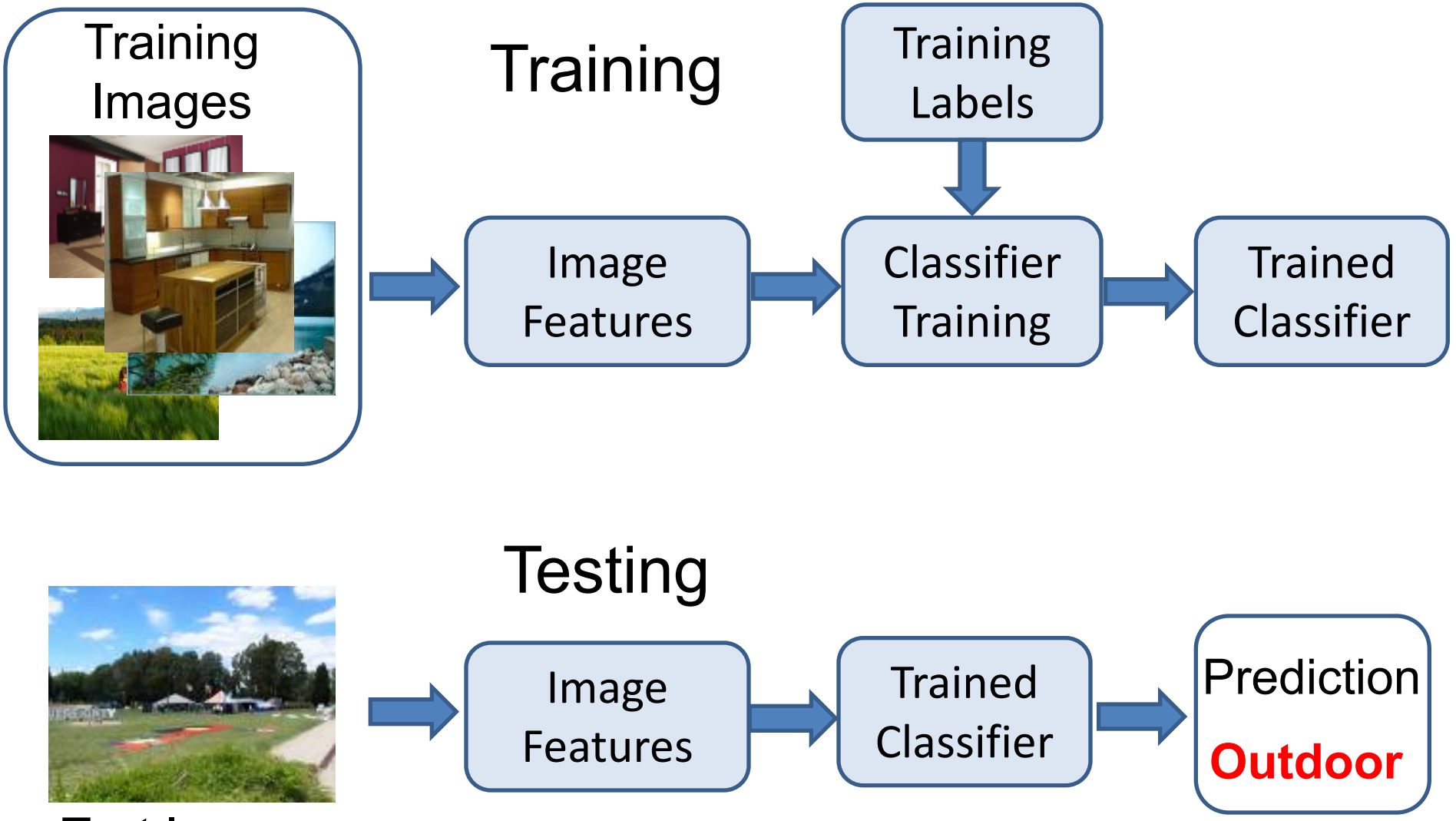
Credit: Manmohan Chandraker

Traditional Image Categorization: Training phase



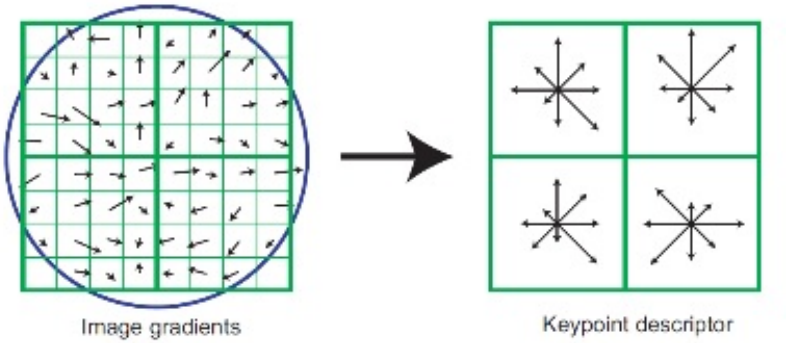
Slide credit: Jia-Bin Huang

Traditional Image Categorization: Testing phase

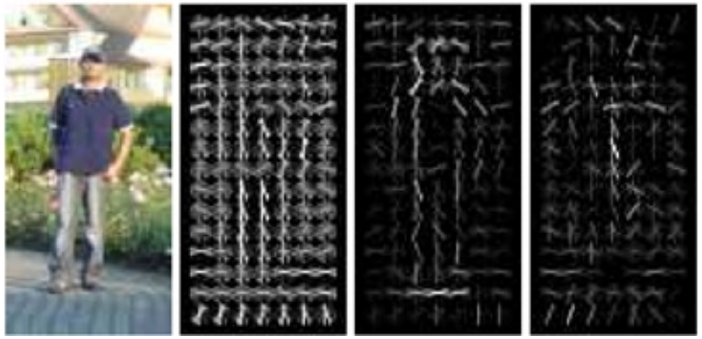


Slide credit: Jia-Bin Huang

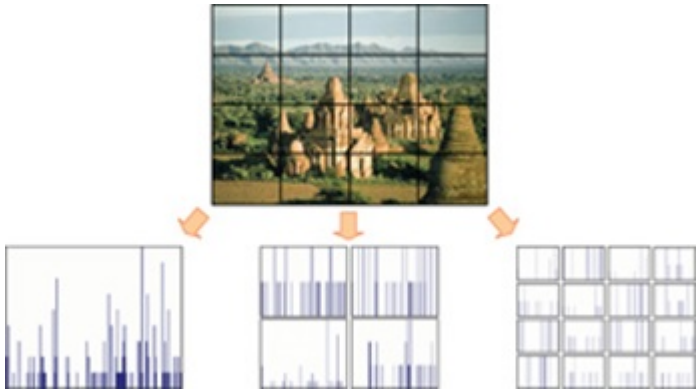
Features have been key



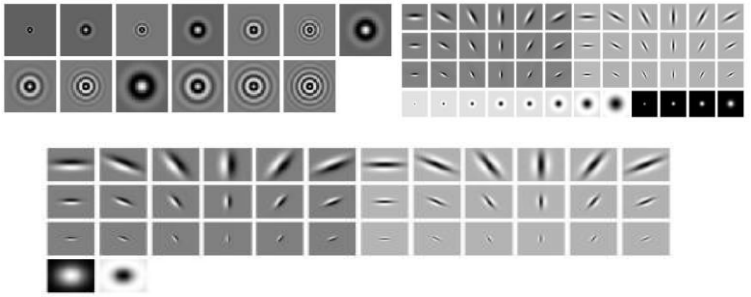
SIFT [Lowe IJCV 04]



HOG [Dalal and Triggs CVPR 05]



SPM [Lazebnik et al. CVPR 06]

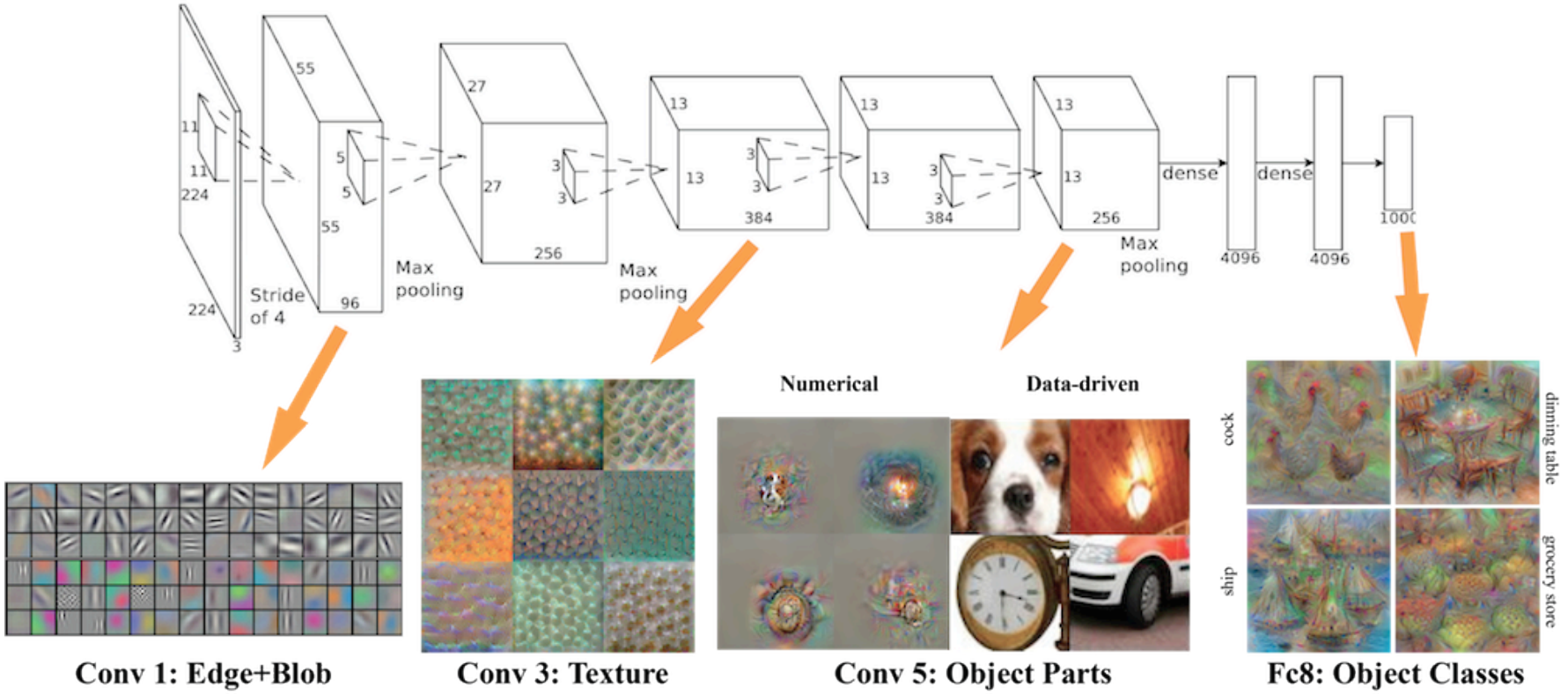


Textons

and many others:

SURF, MSER, LBP, GLOH, …..

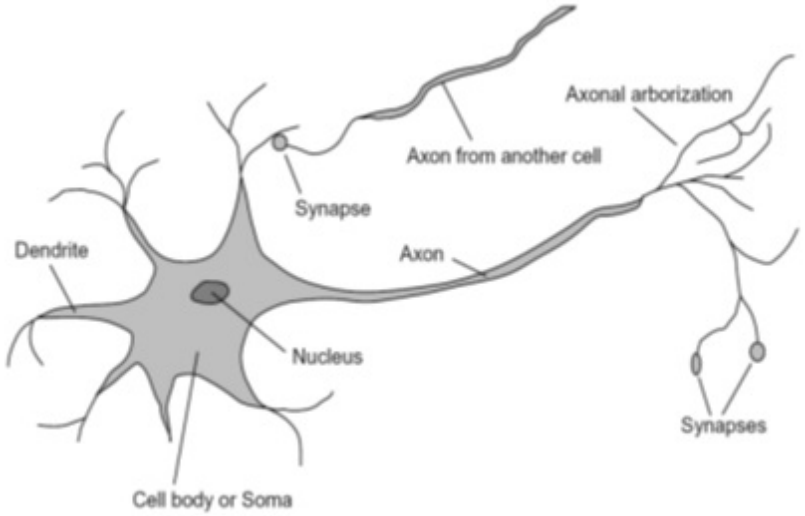
Learning a Hierarchy of Feature Extractors



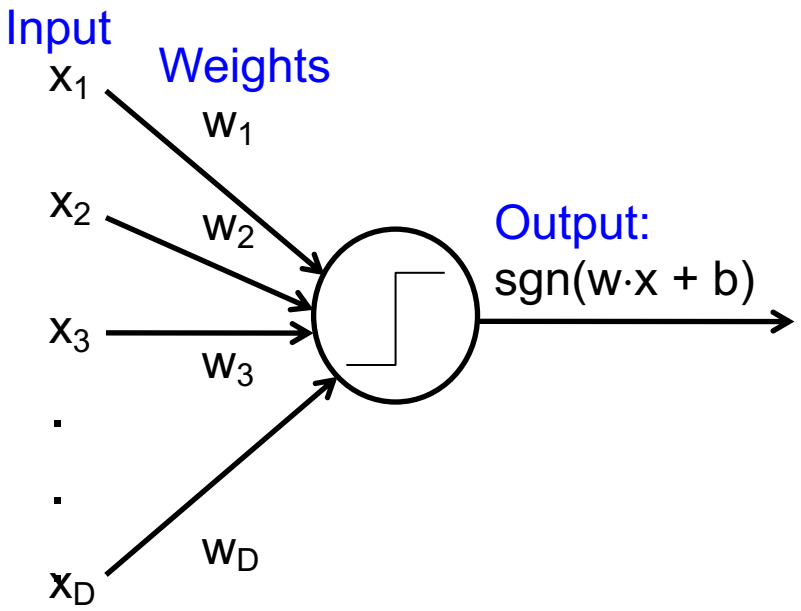
- Hierarchical and expressive feature representations
- Trained end-to-end, rather than hand-crafted for each task
- Remarkable in transferring knowledge across tasks

Neural Networks

Biological neuron and Perceptrons



A biological neuron

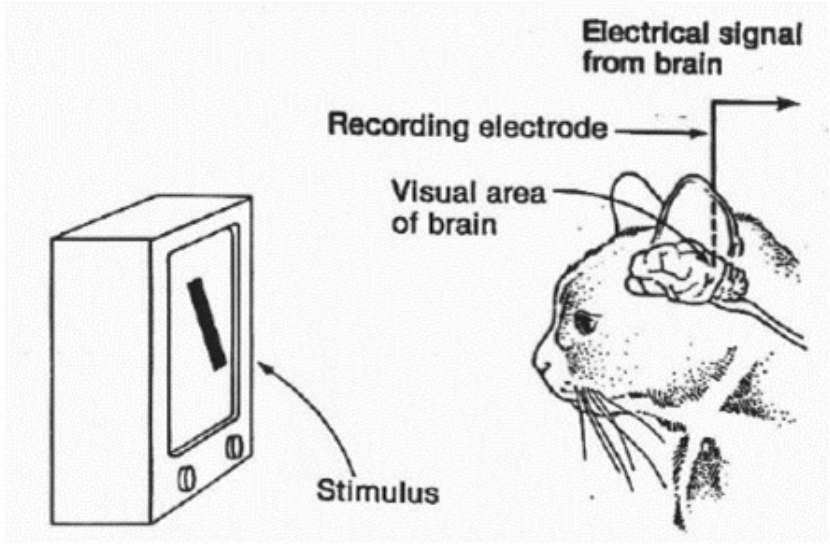


An artificial neuron (Perceptron)

Slide credit: Jia-Bin Huang

Simple, Complex and Hypercomplex cells

David H. Hubel and Torsten Wiesel

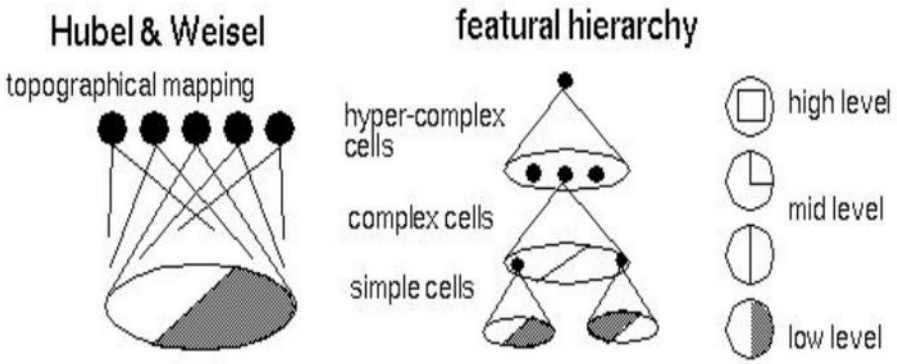


Suggested a **hierarchy** of **feature detectors** in the visual cortex, with higher level features responding to patterns of activation in lower level cells, and propagating activation upwards to still higher level cells.

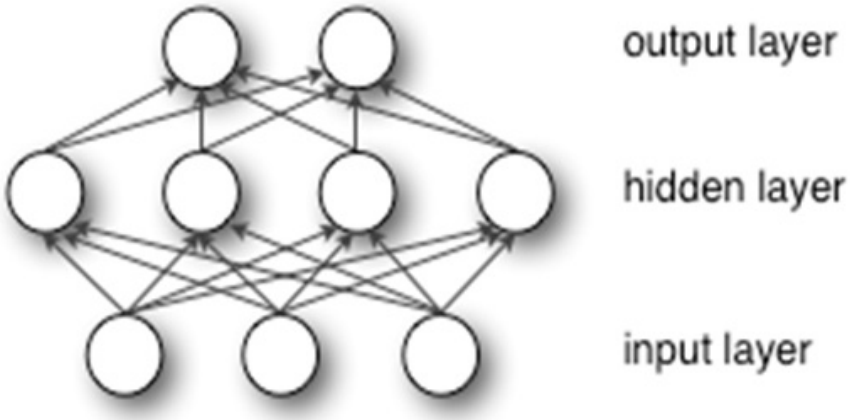
David Hubel's Eye, Brain, and Vision

Slide credit: Jia-Bin Huang

Hubel-Wiesel Architecture and Multi-layer Neural Network



Hubel and Weisel's architecture

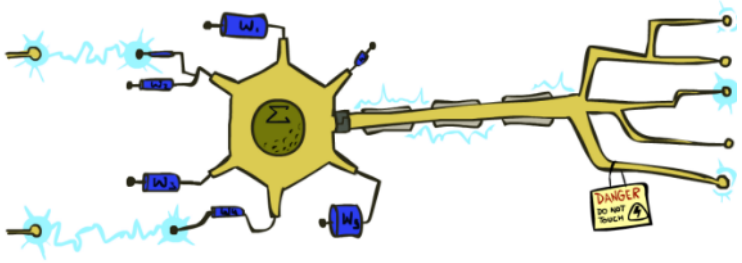


Multi-layer Neural Network

Slide credit: Jia-Bin Huang

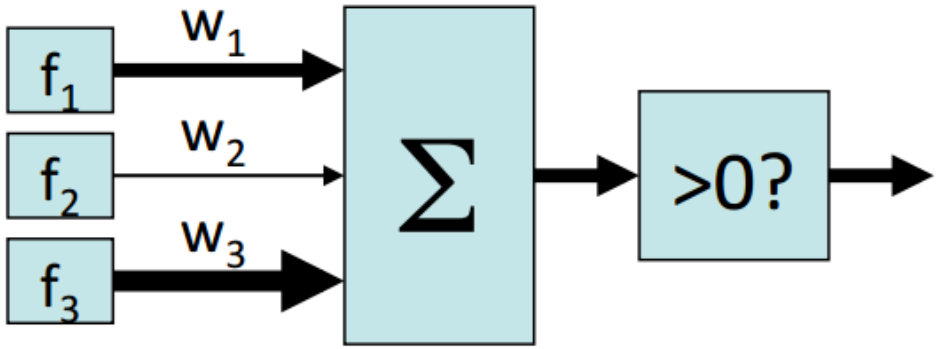
Neuron: Linear Perceptron

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



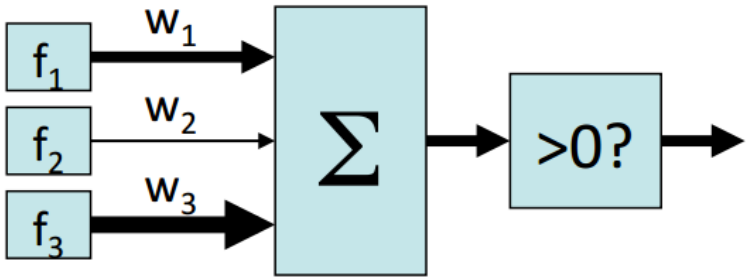
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1



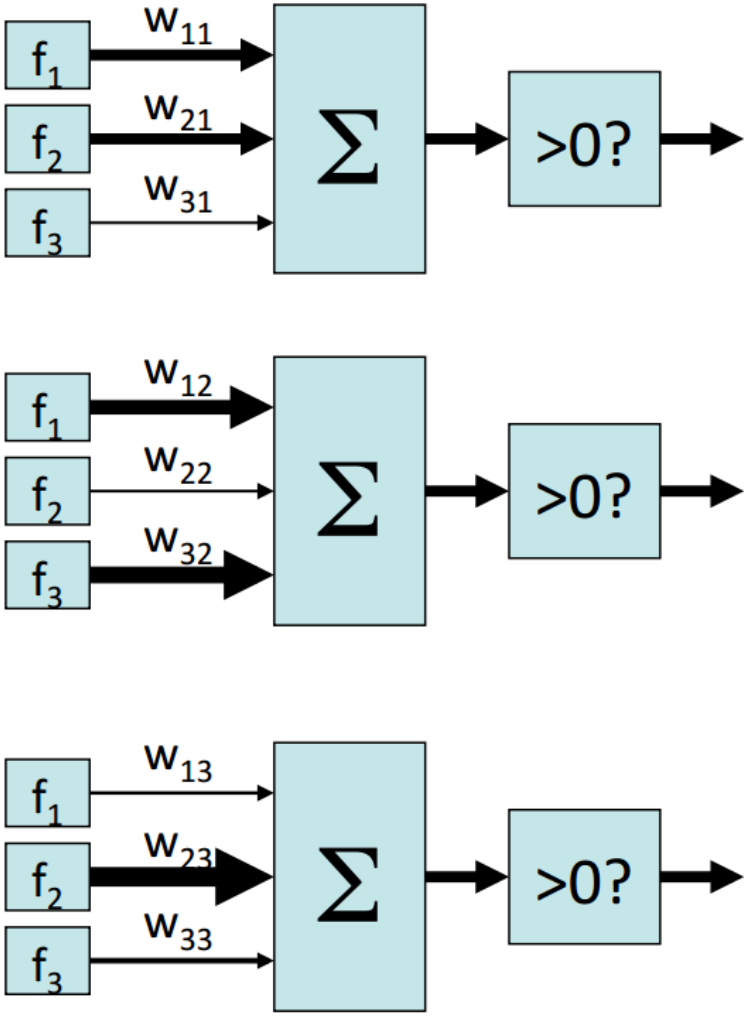
Slide credit: Pieter Abeel and Dan Klein

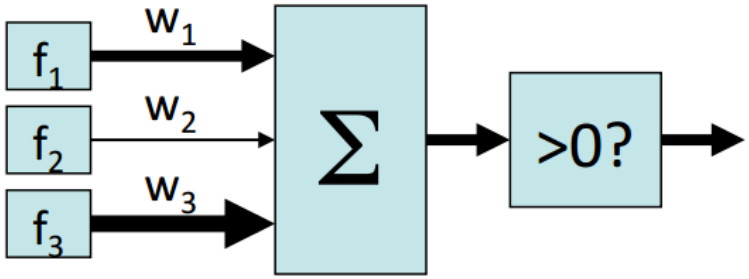
Two-layer perceptron network



Slide credit: Pieter Abeel and Dan Klein

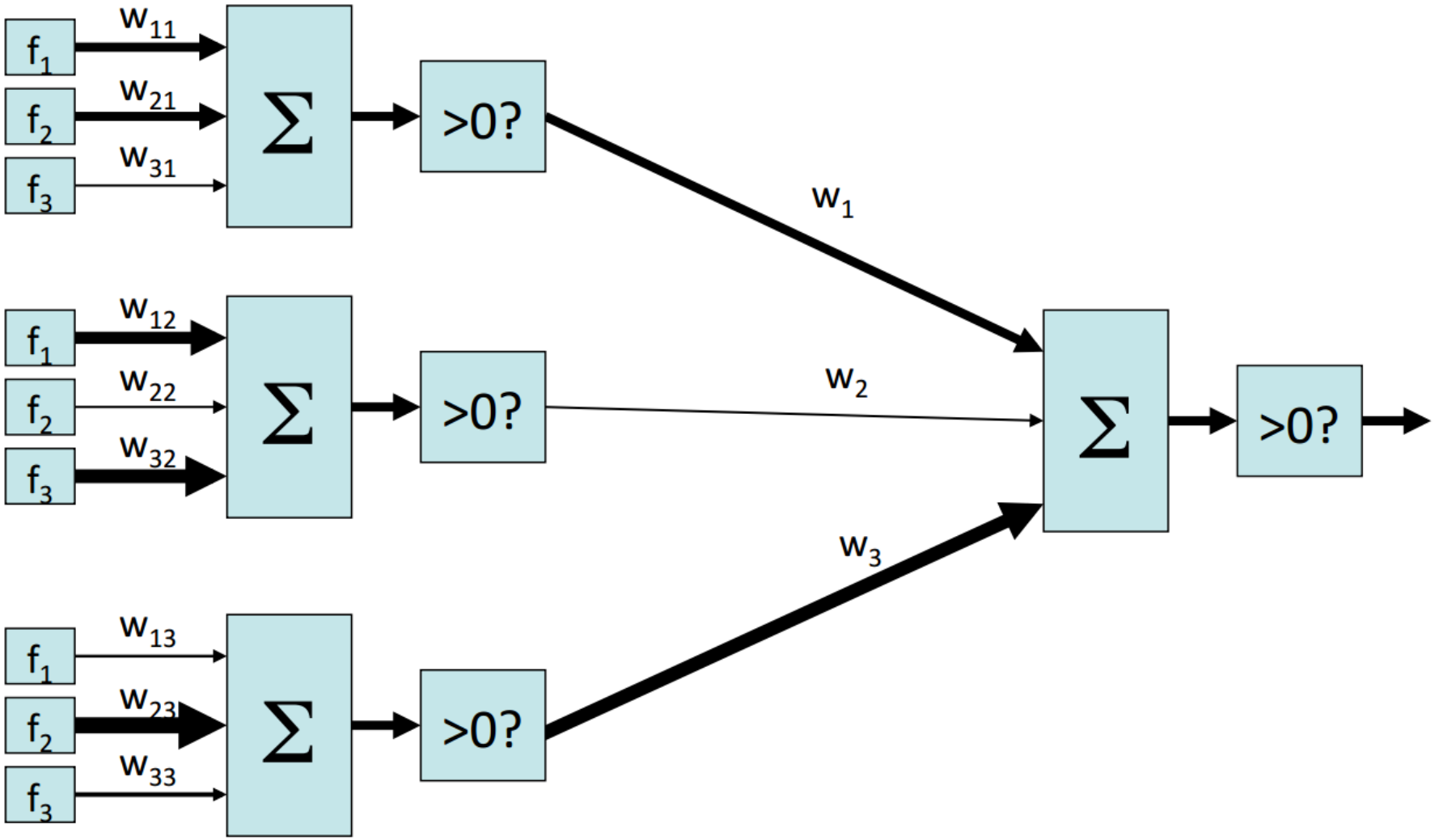
Two-layer perceptron network





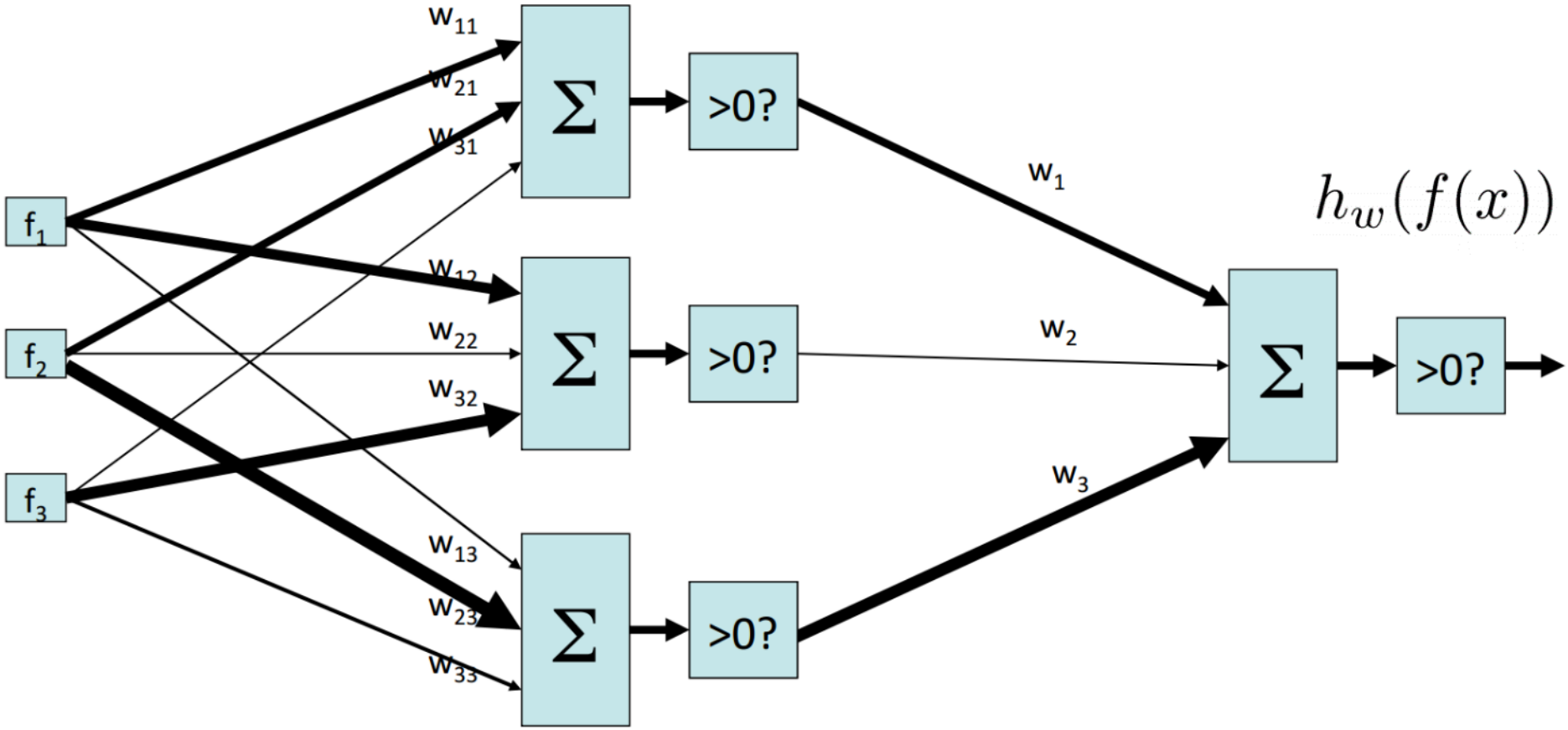
Slide credit: Pieter Abeel and Dan Klein

Two-layer perceptron network



Slide credit: Pieter Abeel and Dan Klein

Two-layer perceptron network

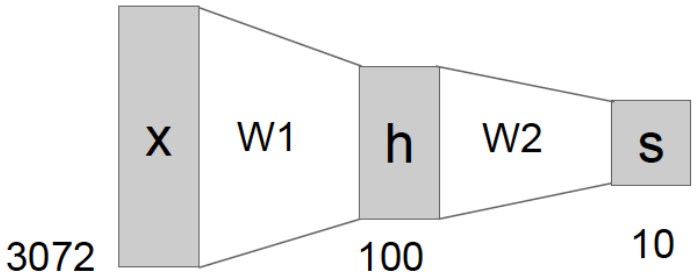


Slide credit: Pieter Abeel and Dan Klein

Neural networks

Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$

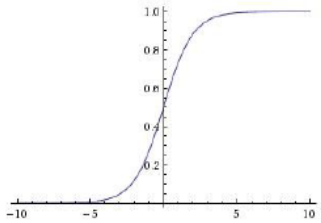


Non-linearity

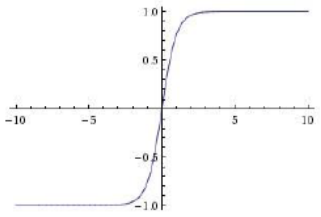
Activation functions

Sigmoid

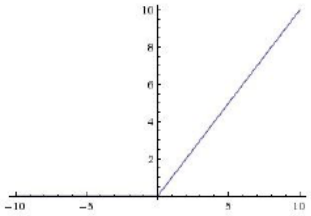
$$\sigma(x) = 1/(1+e^{-x})$$



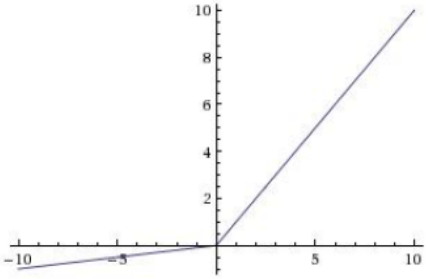
tanh tanh(x)



ReLU max(0,x)

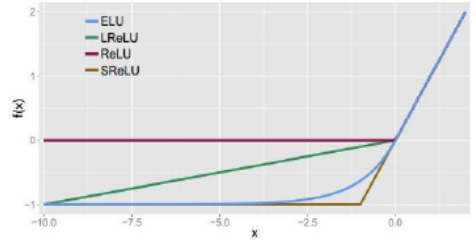


Leaky ReLU
max(0.1x, x)



ELU

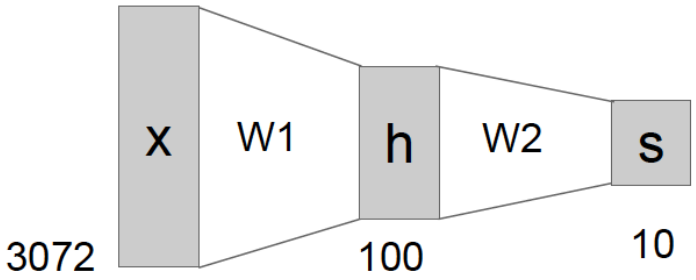
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



Neural networks

Linear score function: $f = Wx$

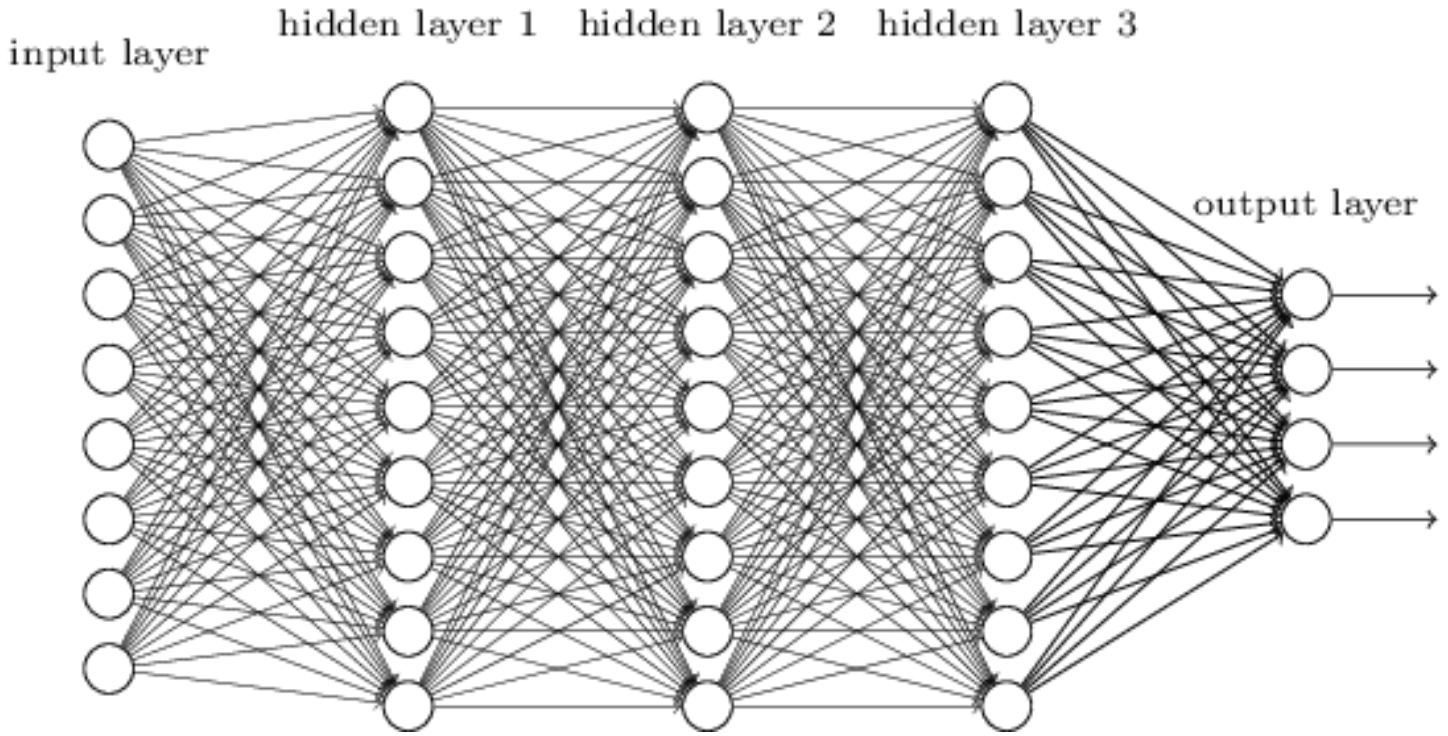
2-layer Neural Network $f = W_2 \max(0, W_1 x)$



3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Multi-layer neural network



Learning w

- Training examples

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$$

- Objective: a misclassification loss

$$\min_{w} \sum_{i=1}^{m} \left(y^{(i)} - h_w(f(x^{(i)})) \right)^2$$

- Procedure:
  - Gradient descent or hill climbing

Slide credit: Pieter Abeel and Dan Klein

Neural network properties

- Theorem (Universal function approximators): A two-layer network with a sufficient number of neurons can approximate any continuous function to any desired accuracy

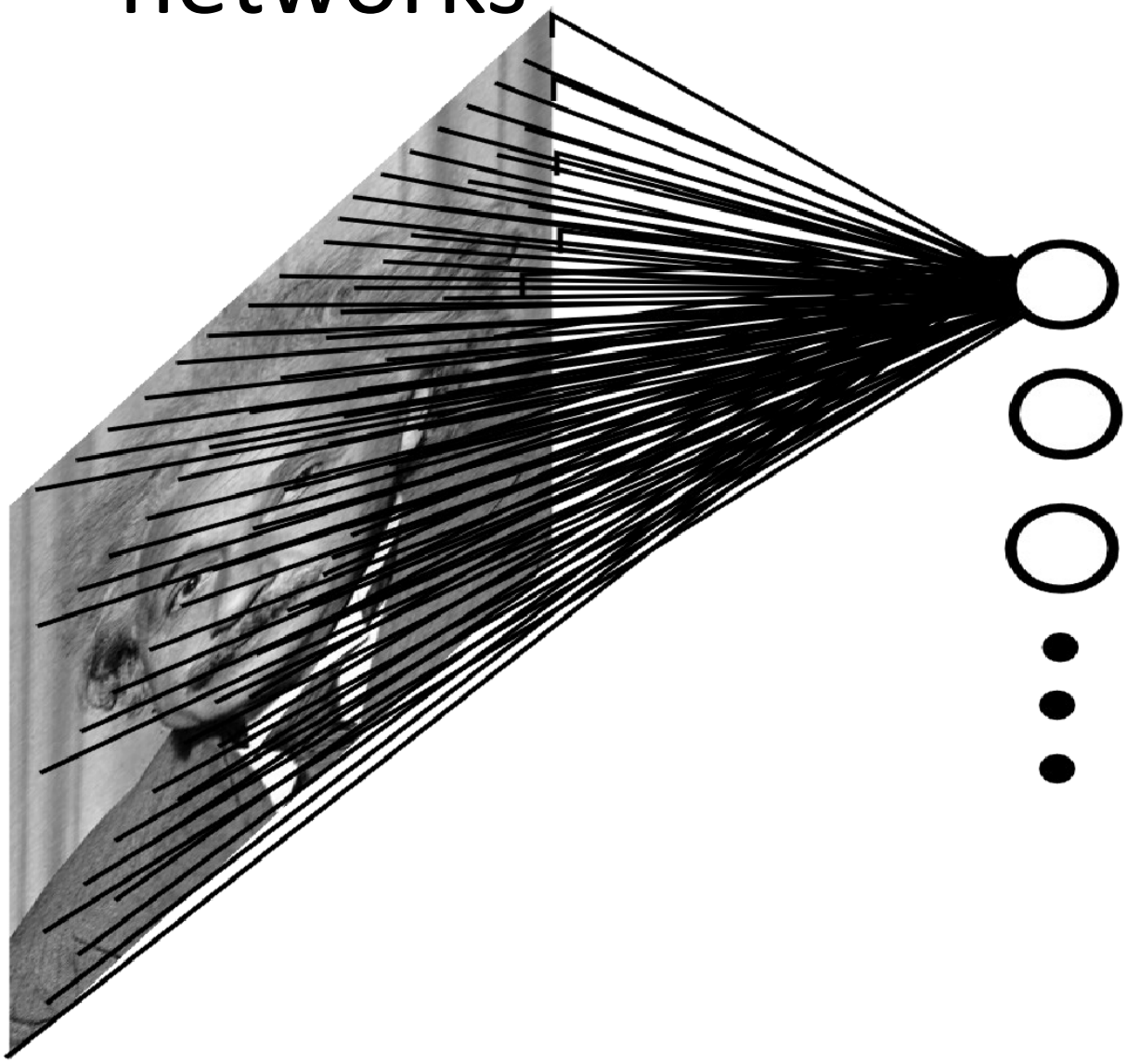
- Practical considerations:
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
  - Hill-climbing can get stuck in local optima

Approximation by Superpositions of Sigmoidal Function,1989

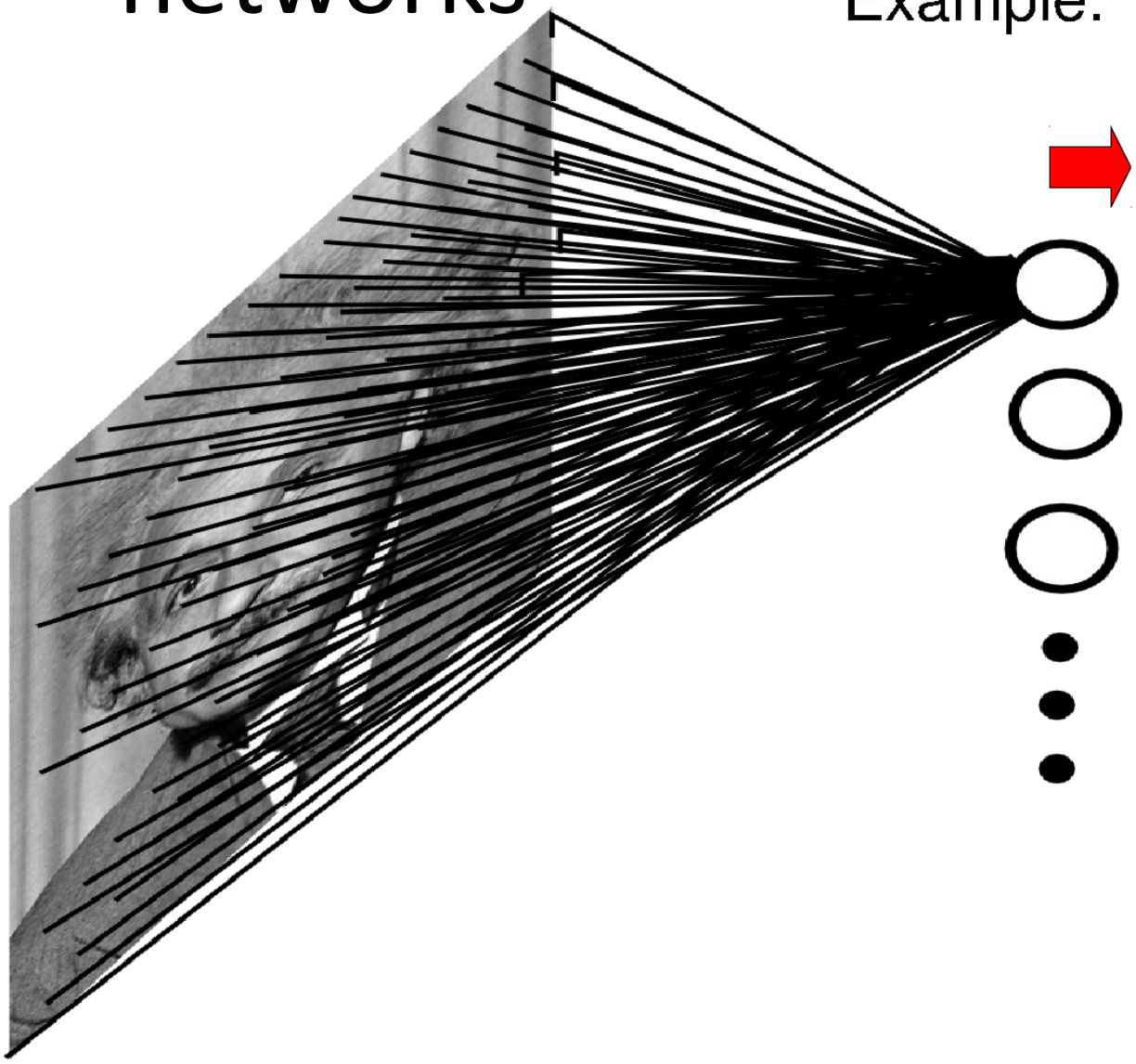
Slide credit: Pieter Abeel and Dan Klein

Convolutional Neural Networks

Images as input to neural networks



Images as input to neural networks

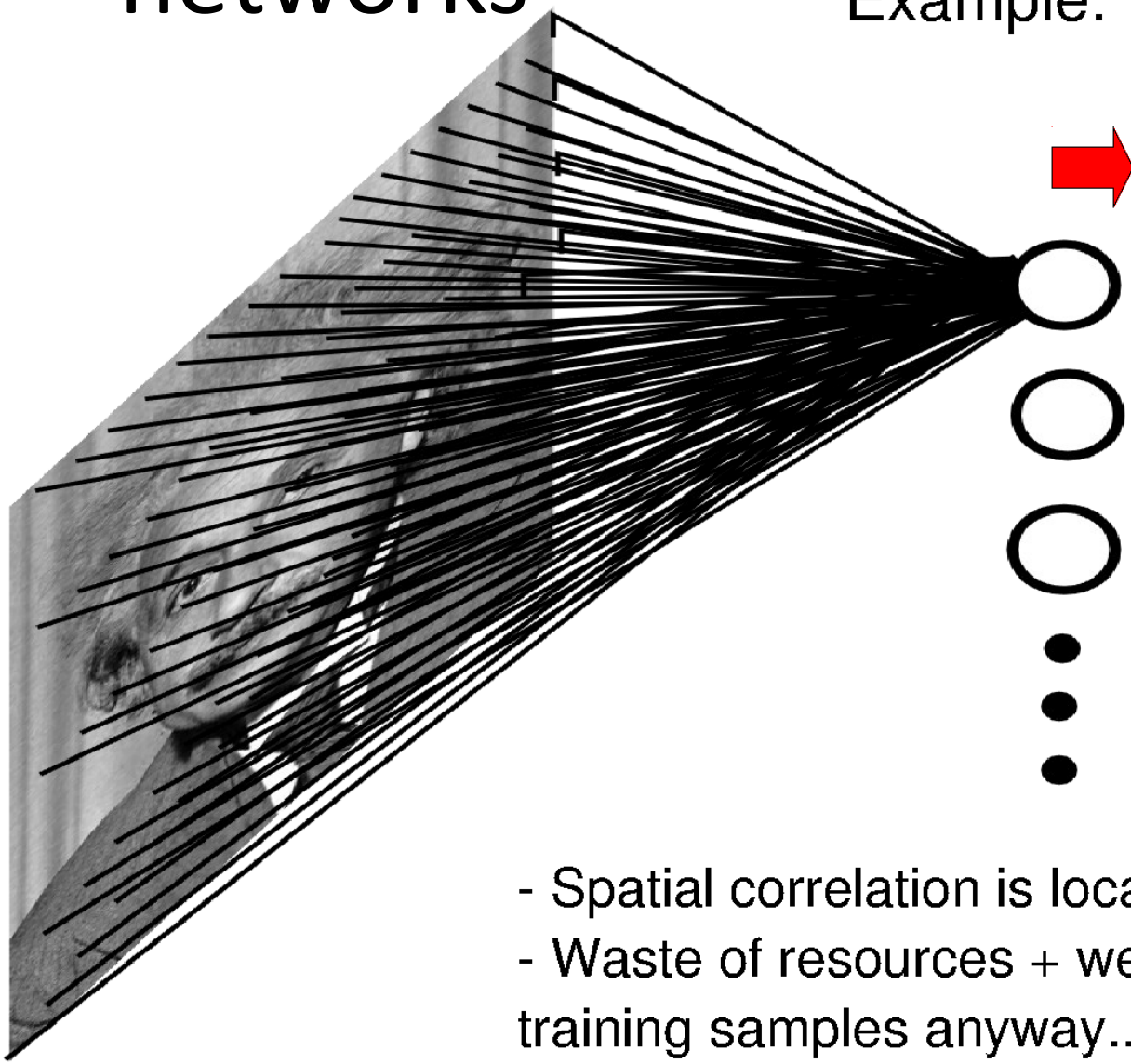


Example: 200x200 image

40K hidden units

~2B parameters!!!

Images as input to neural networks

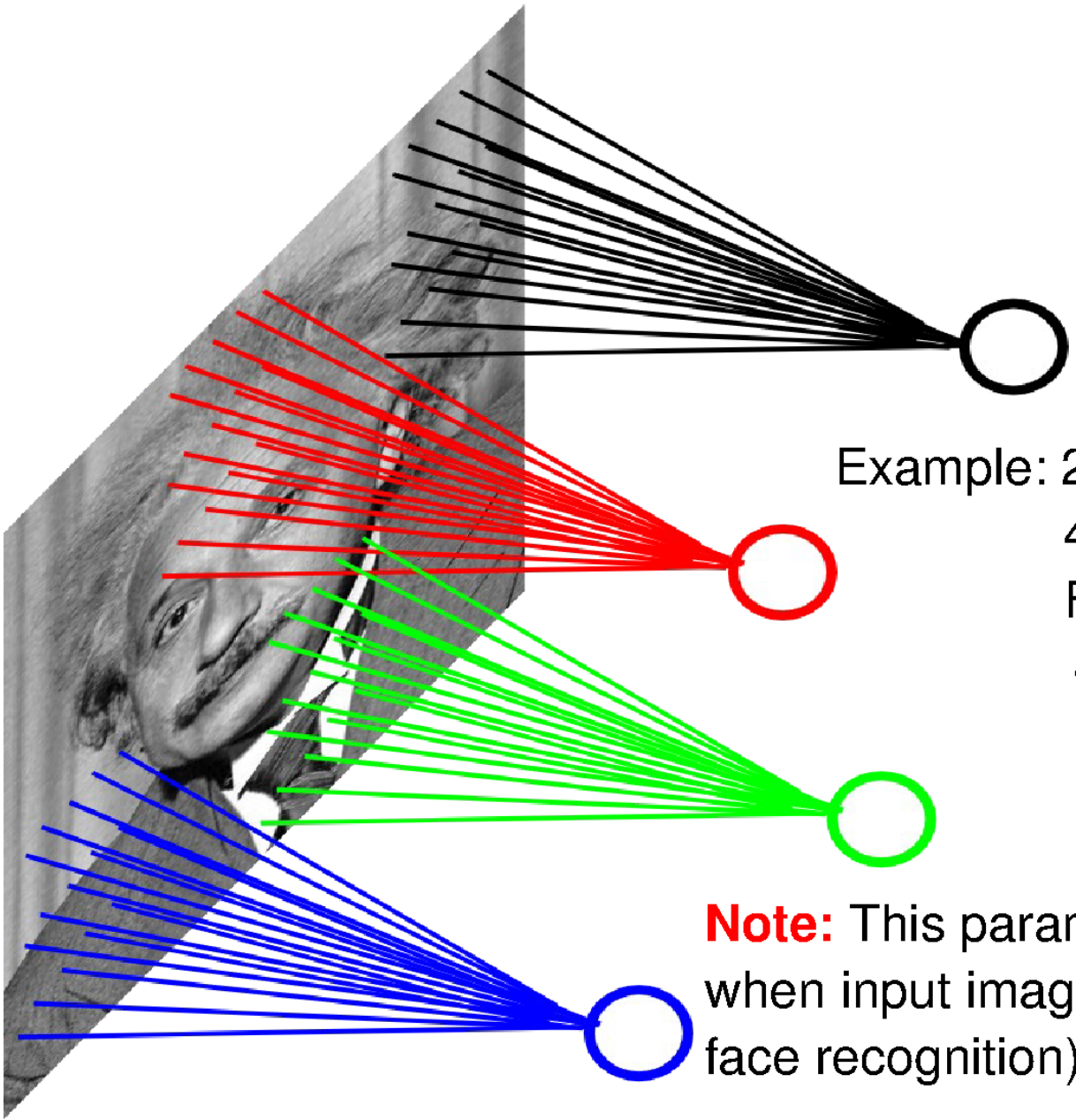


Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

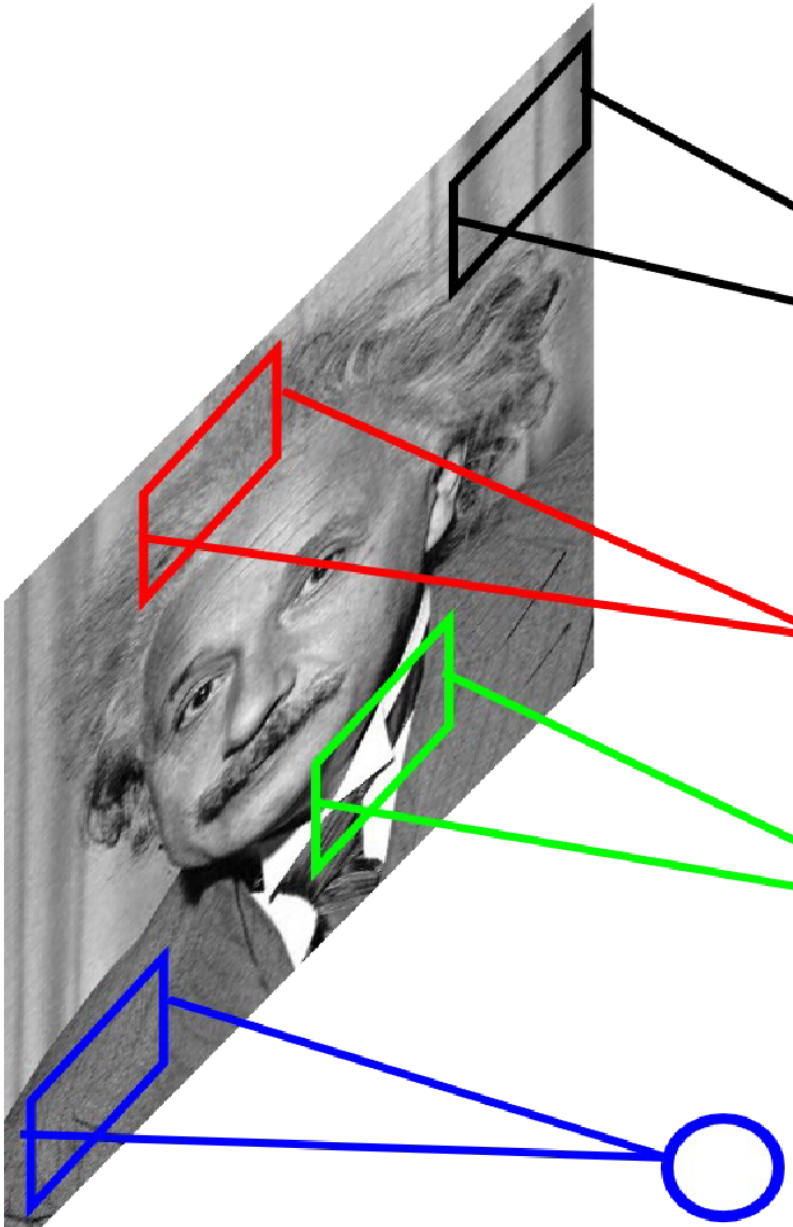
Motivation

- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about ‘local neighborhoods’ only for a given neural network layer.
  - Composition of layers will expand local -> global.



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).



STATIONARITY? Statistics is similar at different locations

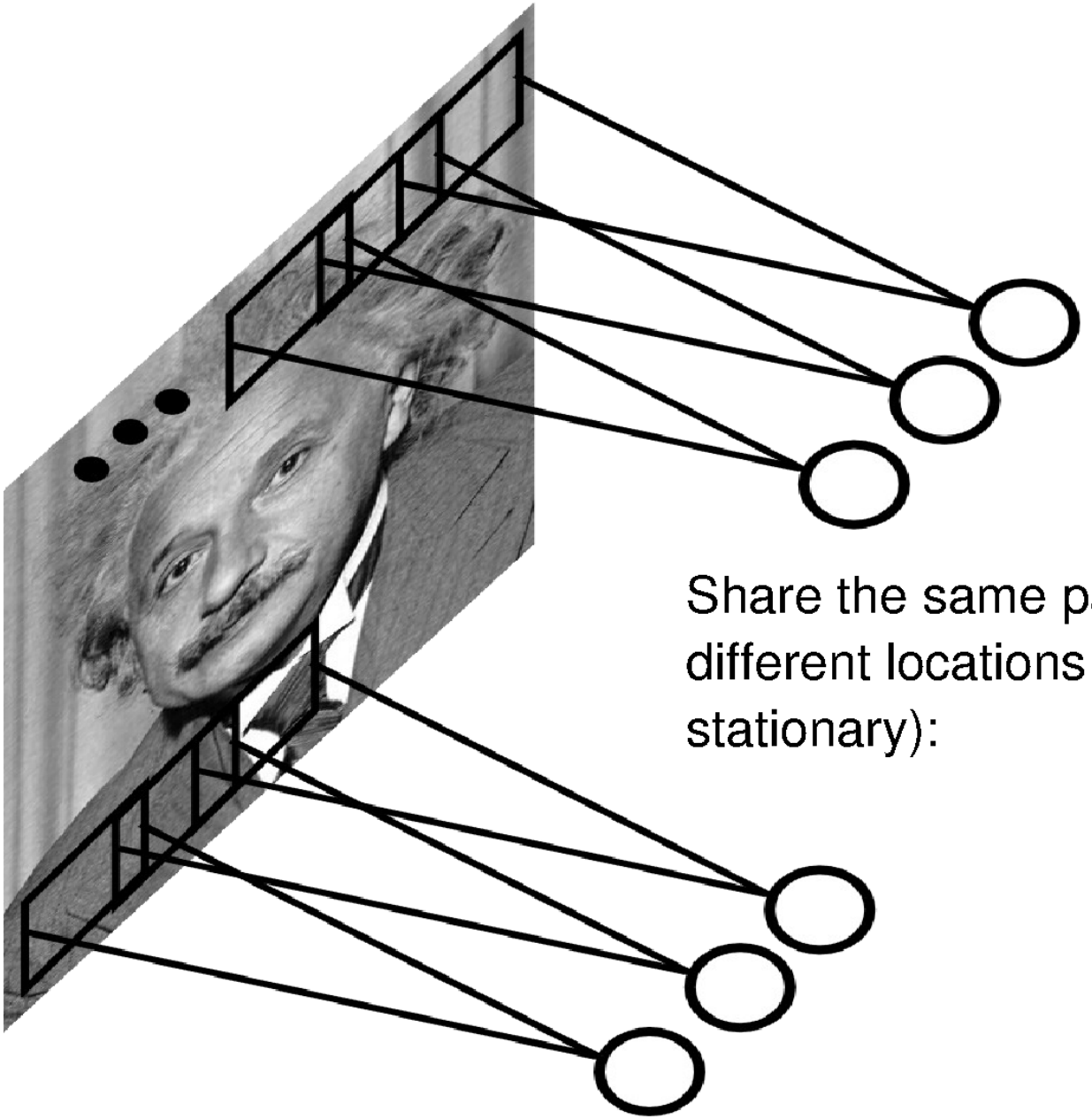
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Motivation

- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about 'local neighborhoods' only for a given neural network layer.
  - Composition of layers will expand local -> global.

- Parameter sharing
  - 'Tied weights' – use same weights for more than one perceptron in the neural network.
  - Leads to *equivariant representation*
    - If input changes (e.g., translates), then output changes similarly



Share the same parameters across different locations (assuming input is stationary):

Filtering reminder:
Correlation (rotated convolution)

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

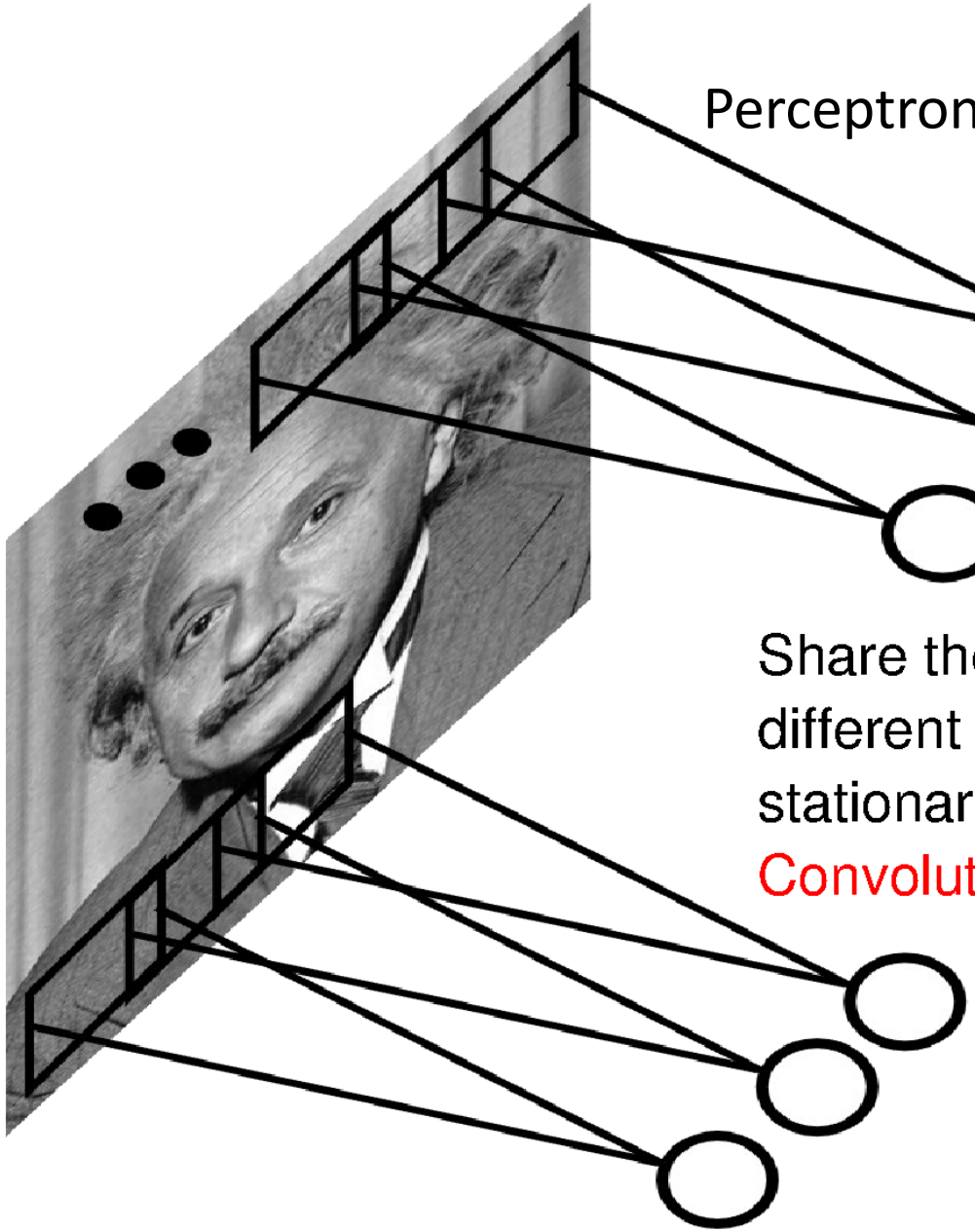
$h[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

Credit: S. Seitz

Convolutional Layer



Perceptron: $\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$

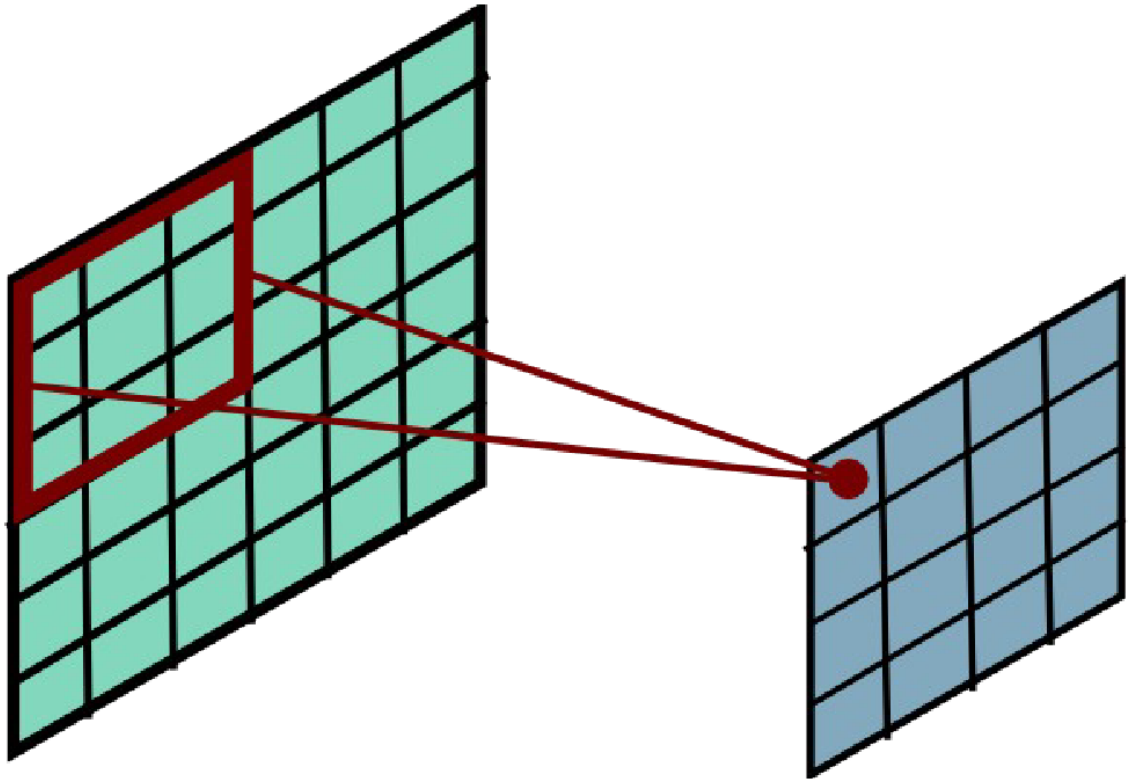
$w \cdot x \equiv \sum_j w_j x_j$

This is convolution!

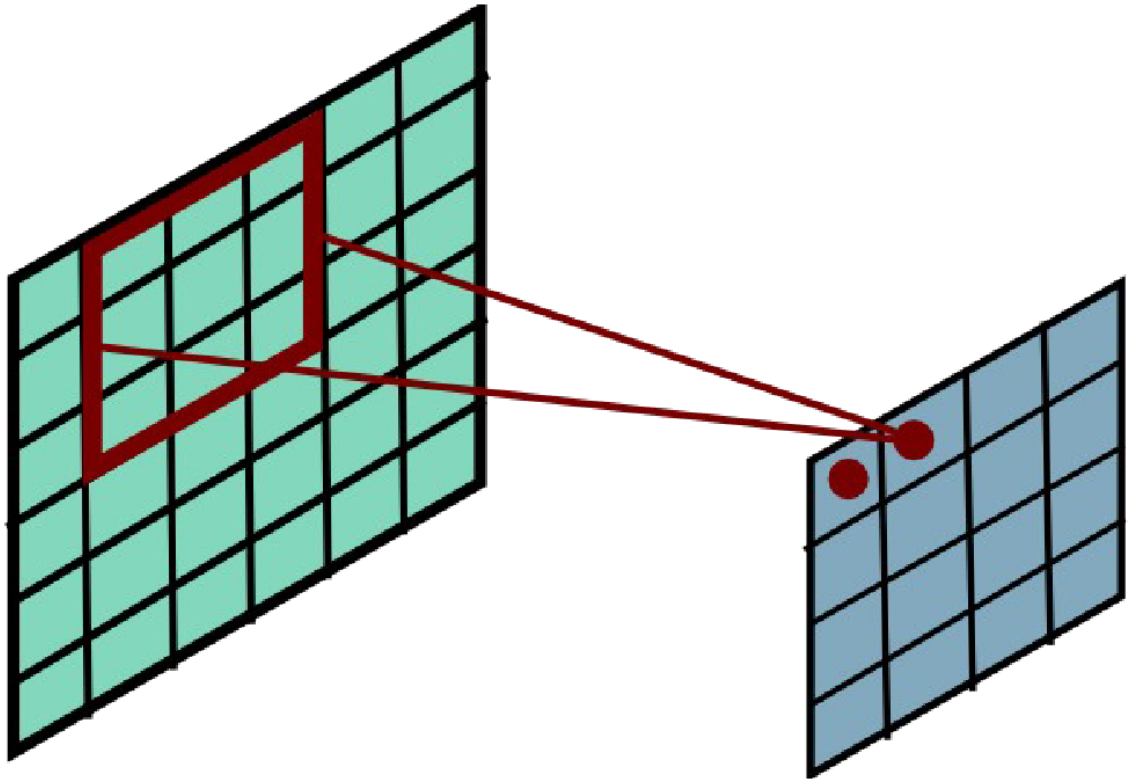
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

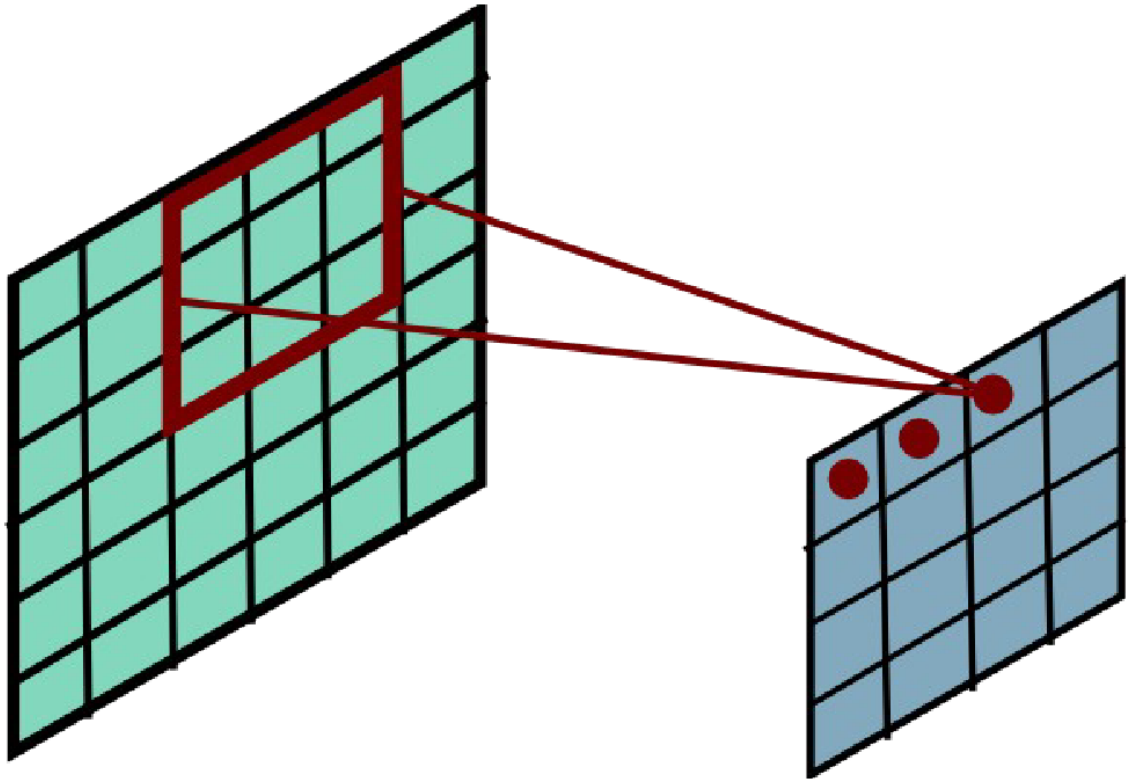
Convolutional Layer



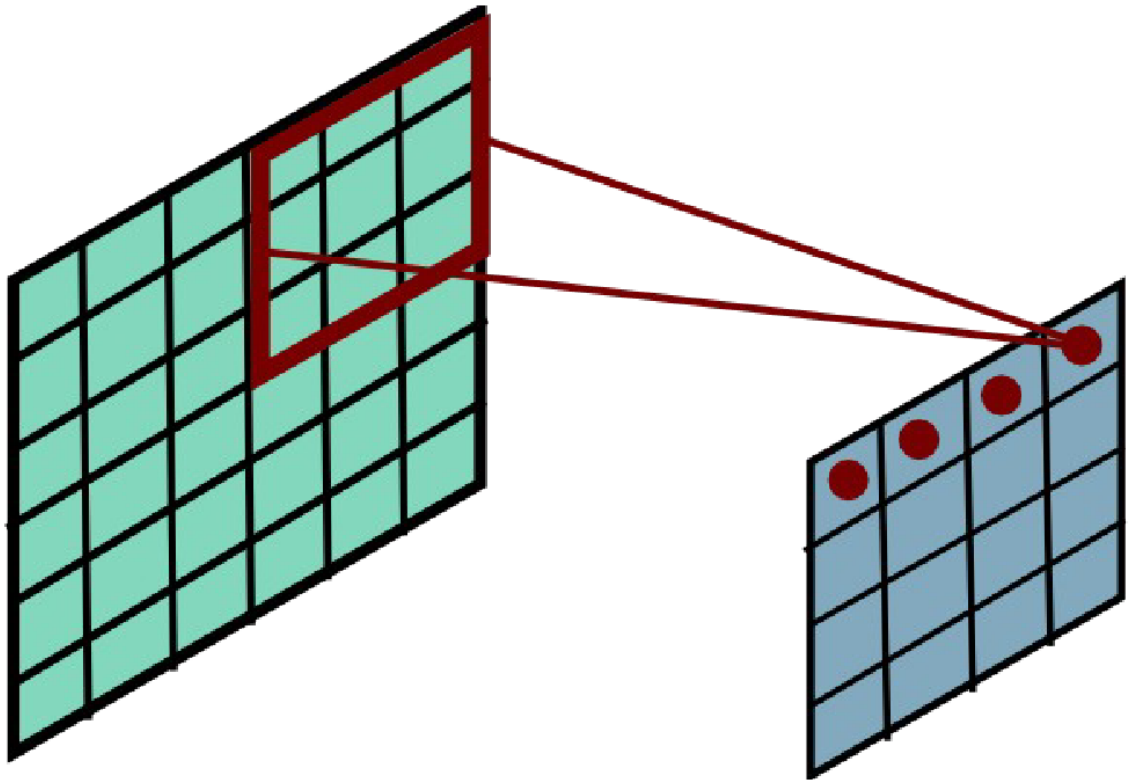
Convolutional Layer



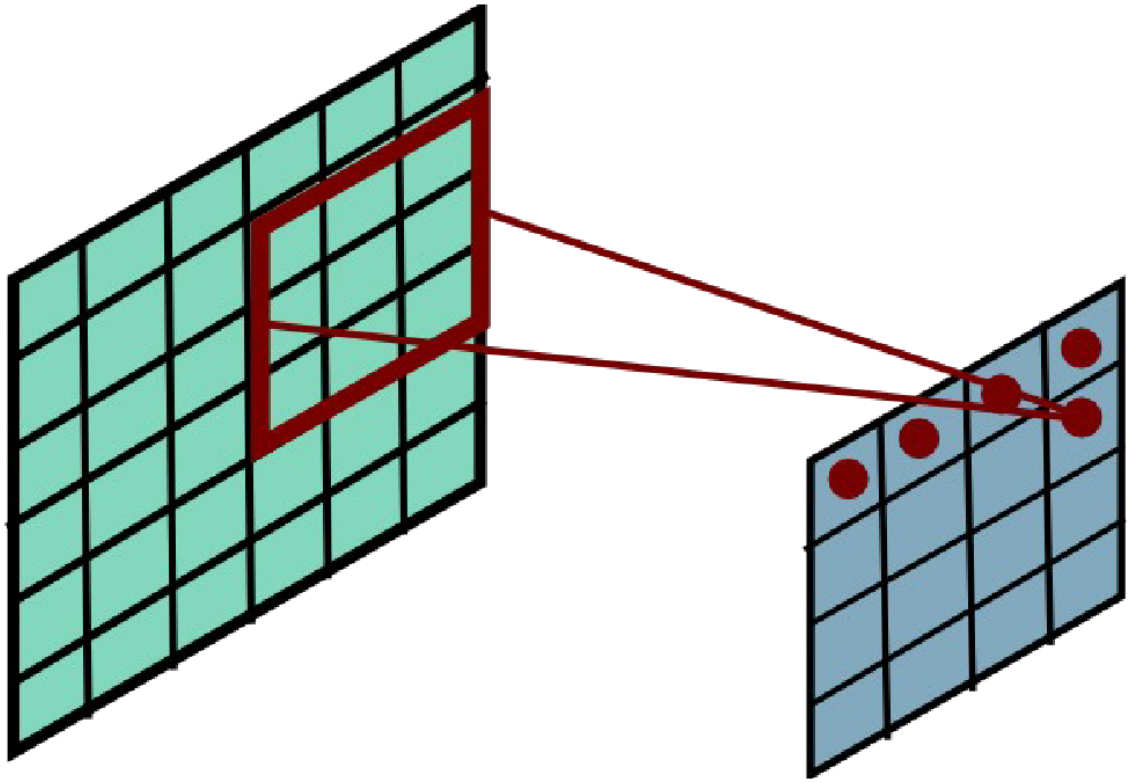
Convolutional Layer



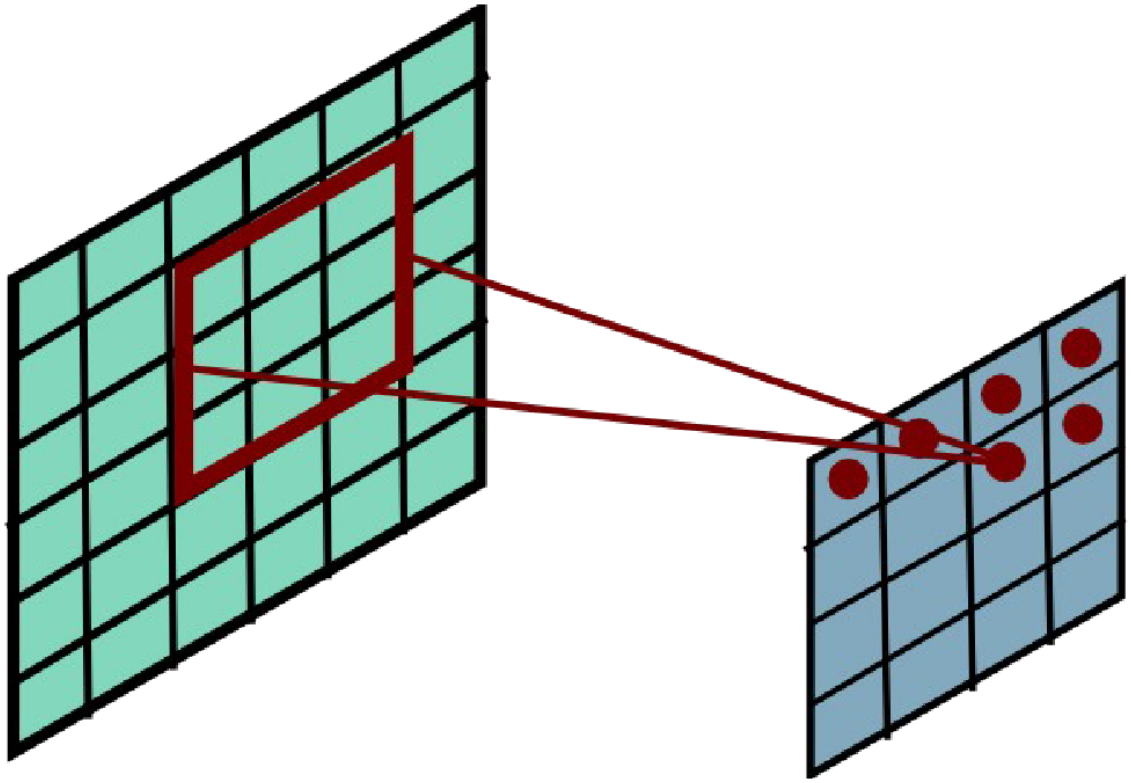
Convolutional Layer



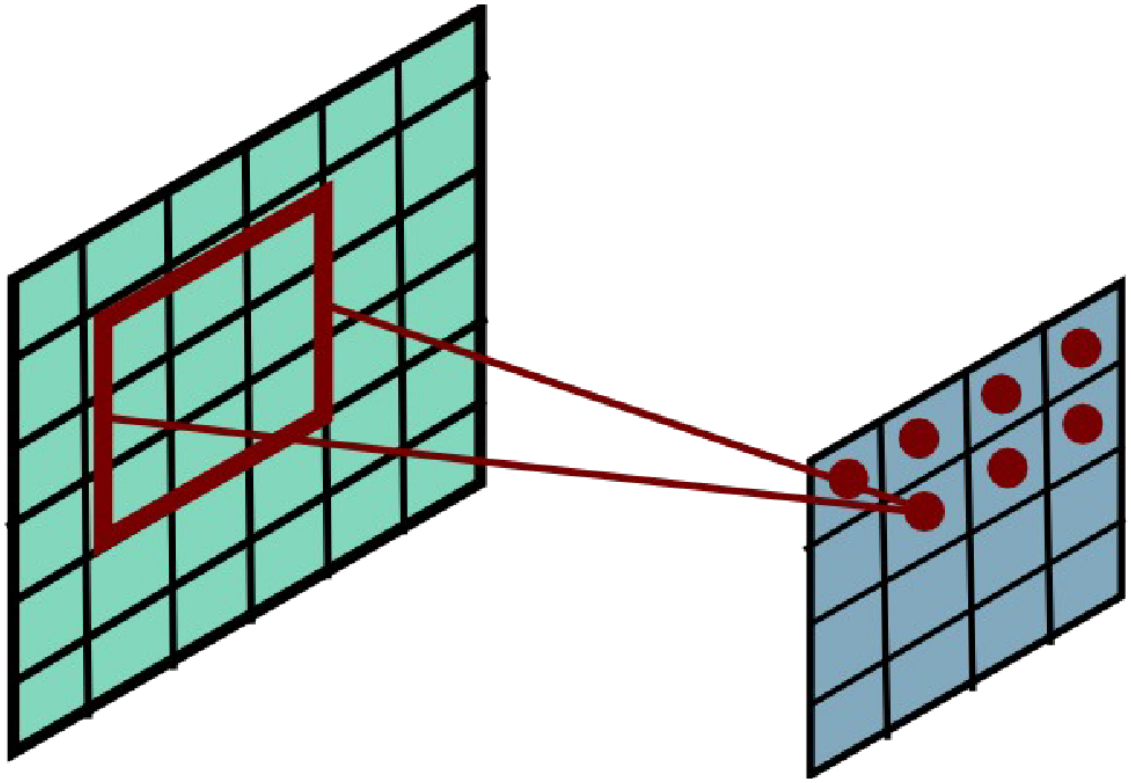
Convolutional Layer



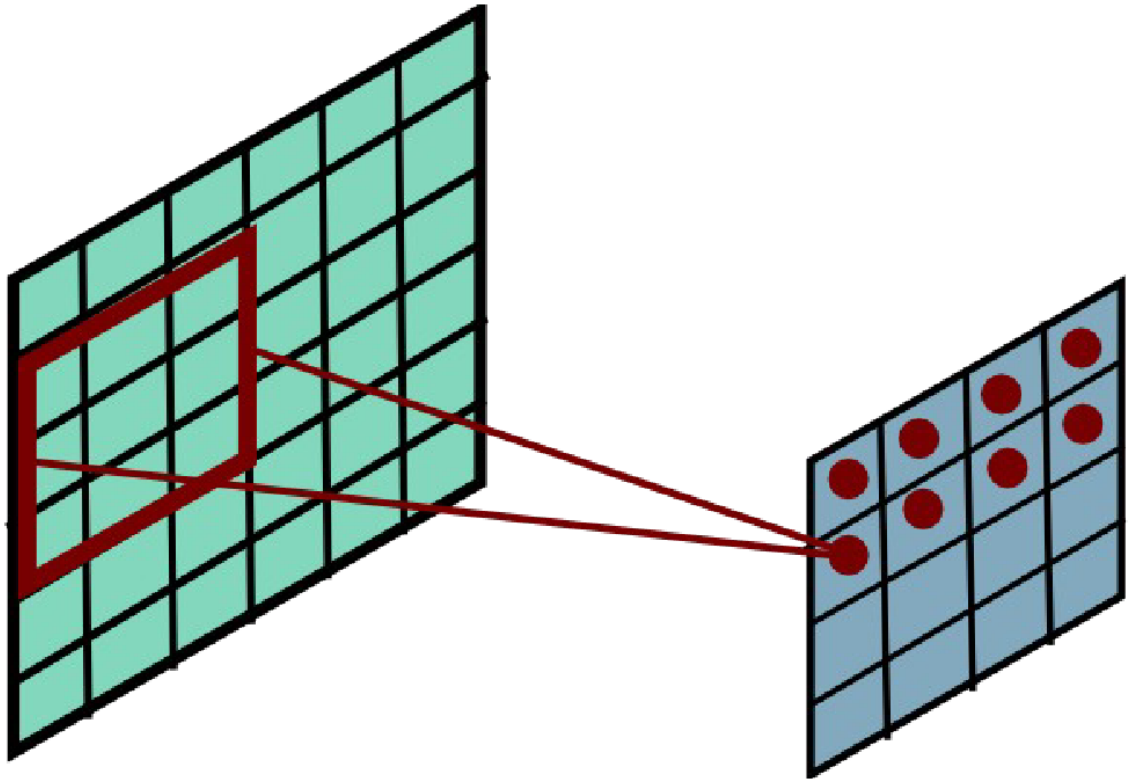
Convolutional Layer



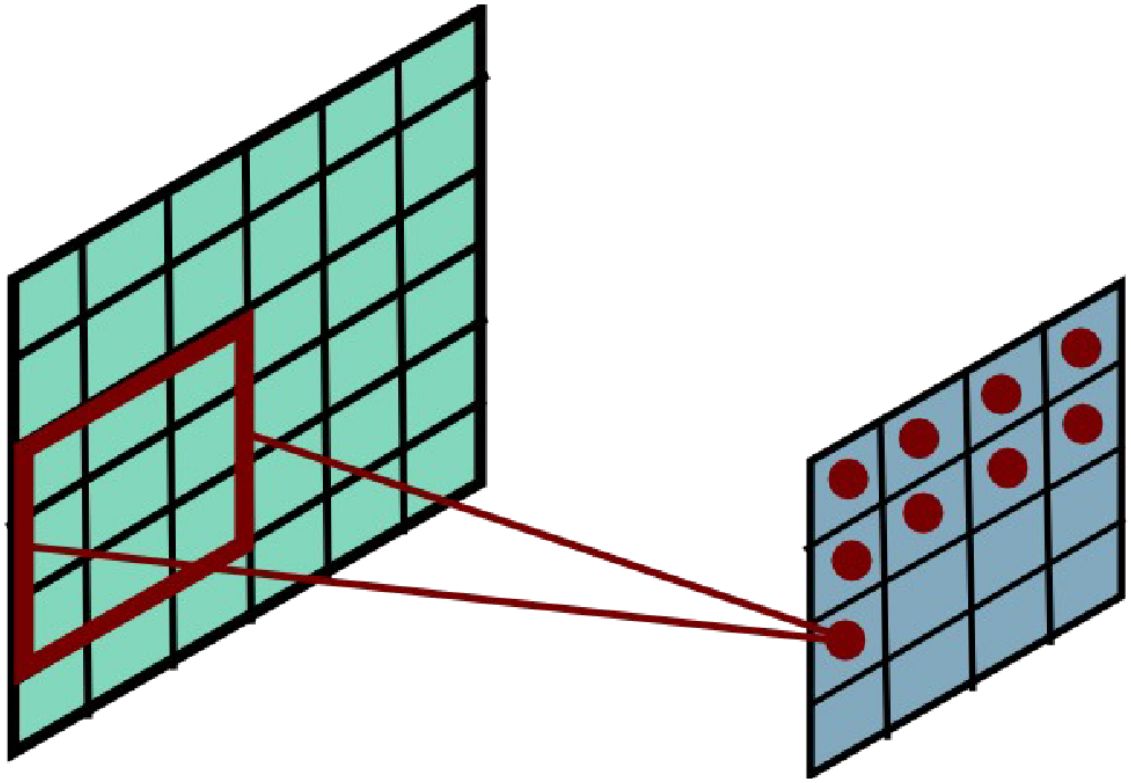
Convolutional Layer



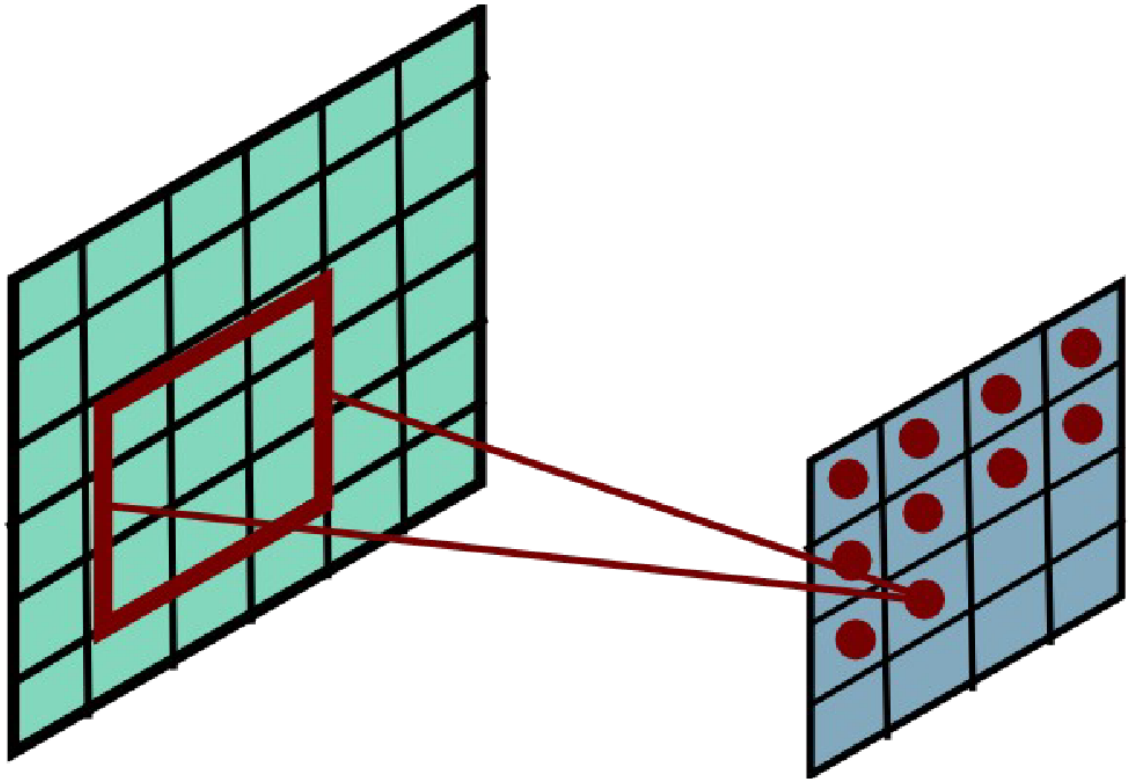
Convolutional Layer



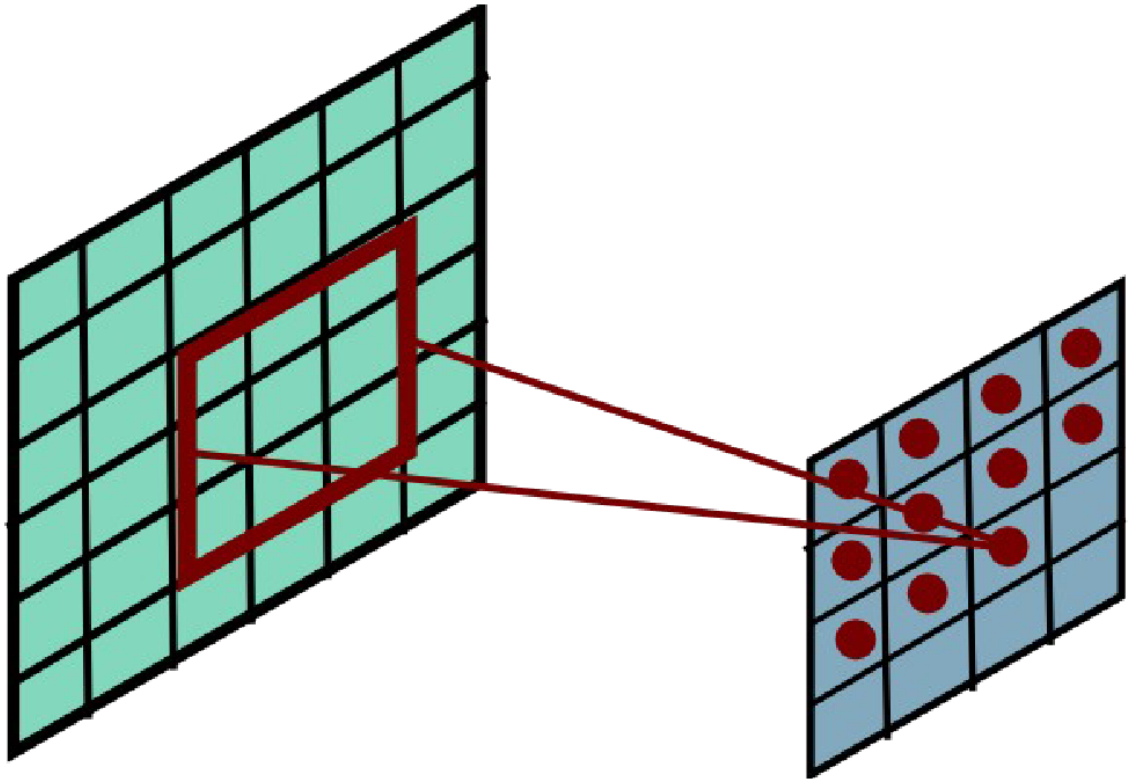
Convolutional Layer



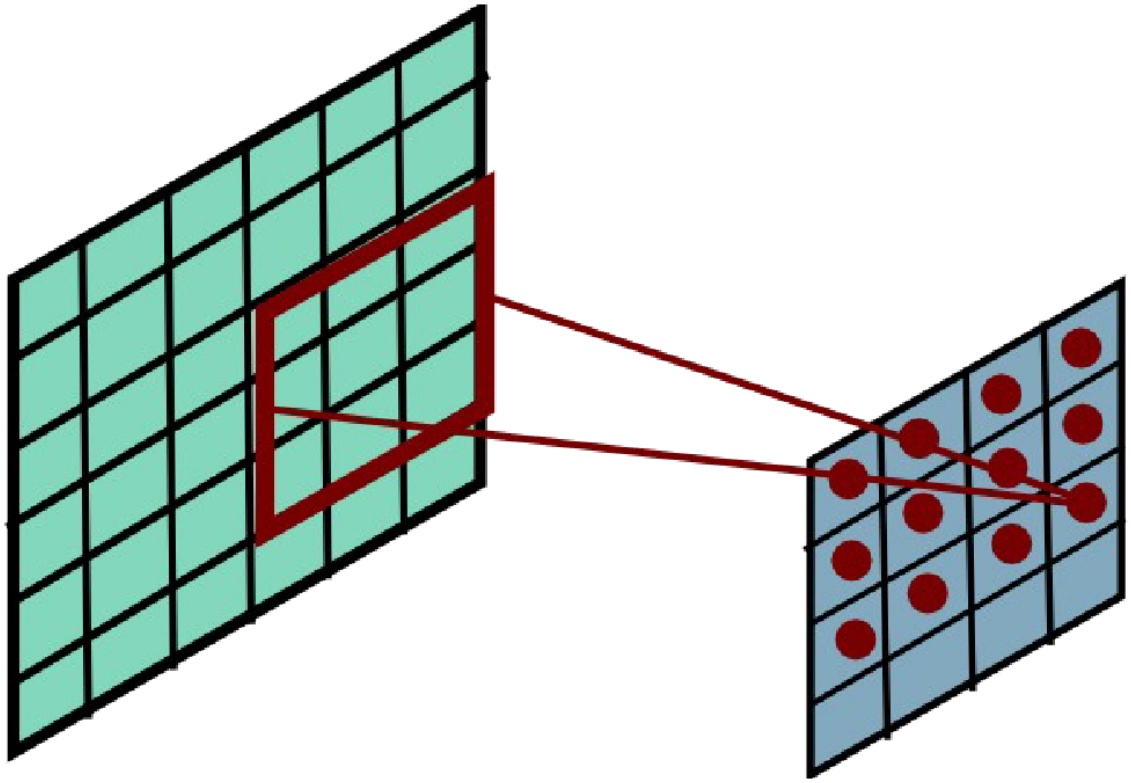
Convolutional Layer



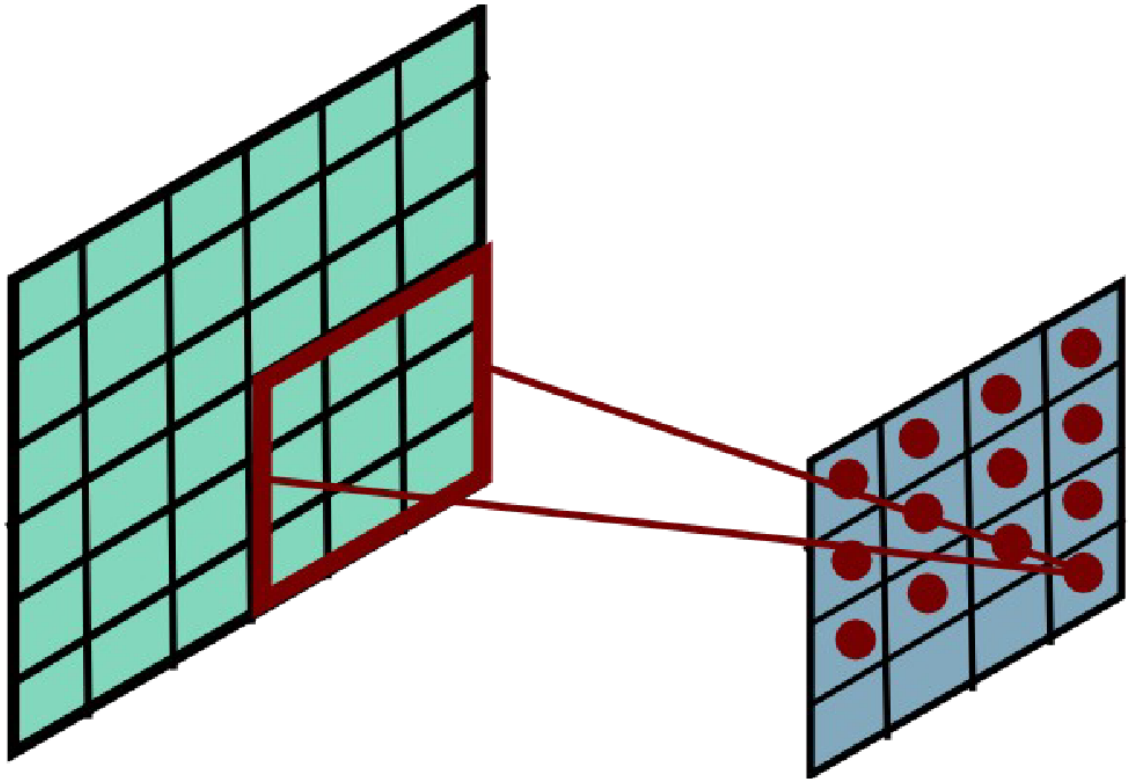
Convolutional Layer



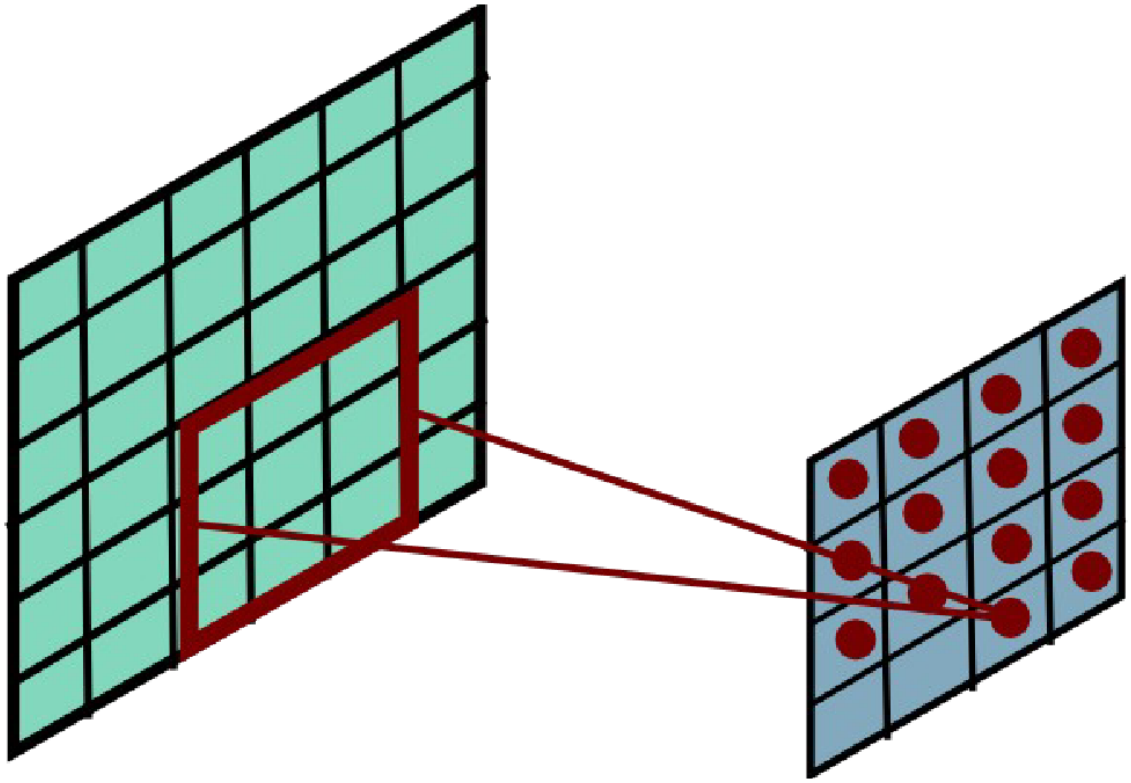
Convolutional Layer



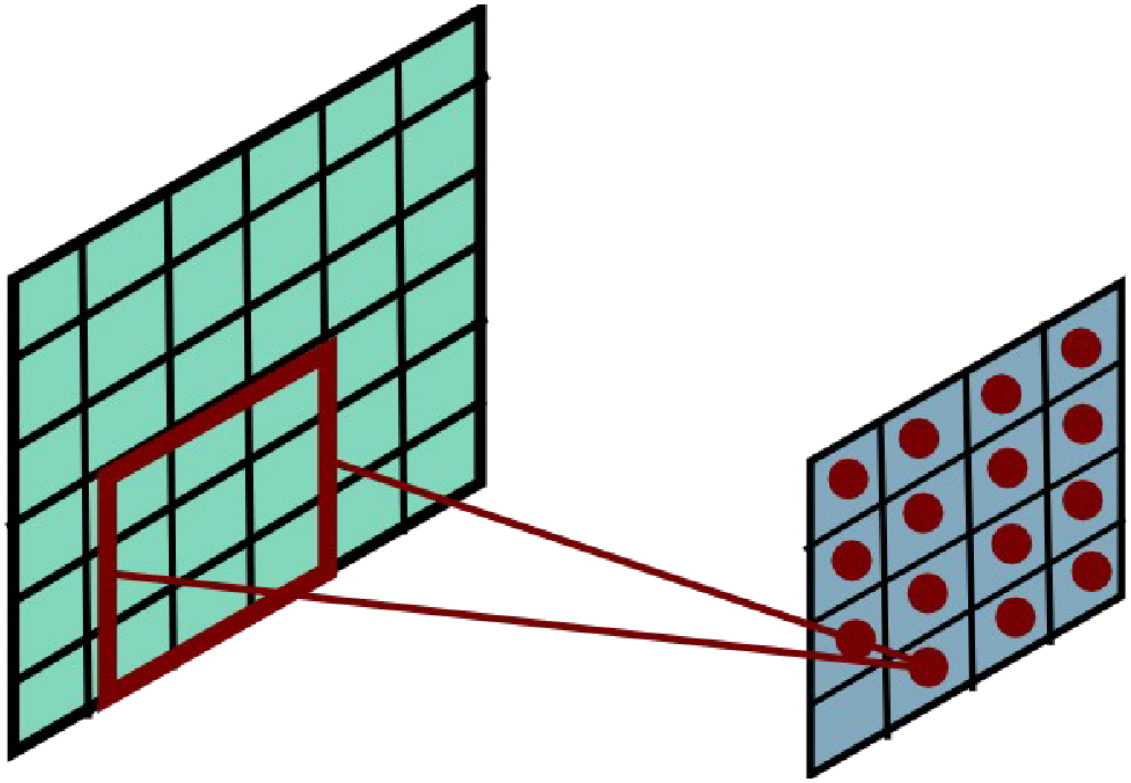
Convolutional Layer



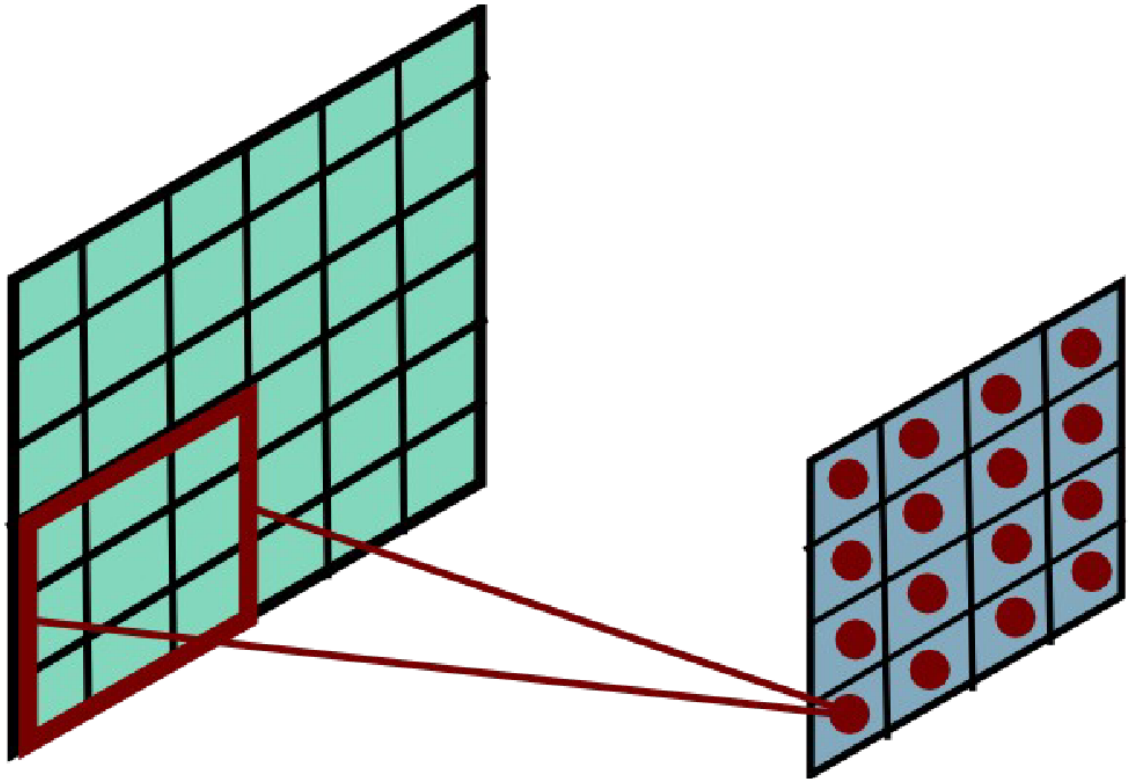
Convolutional Layer



Convolutional Layer



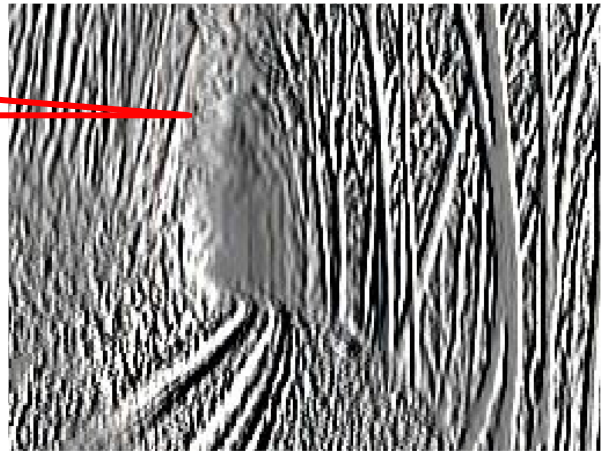
Convolutional Layer



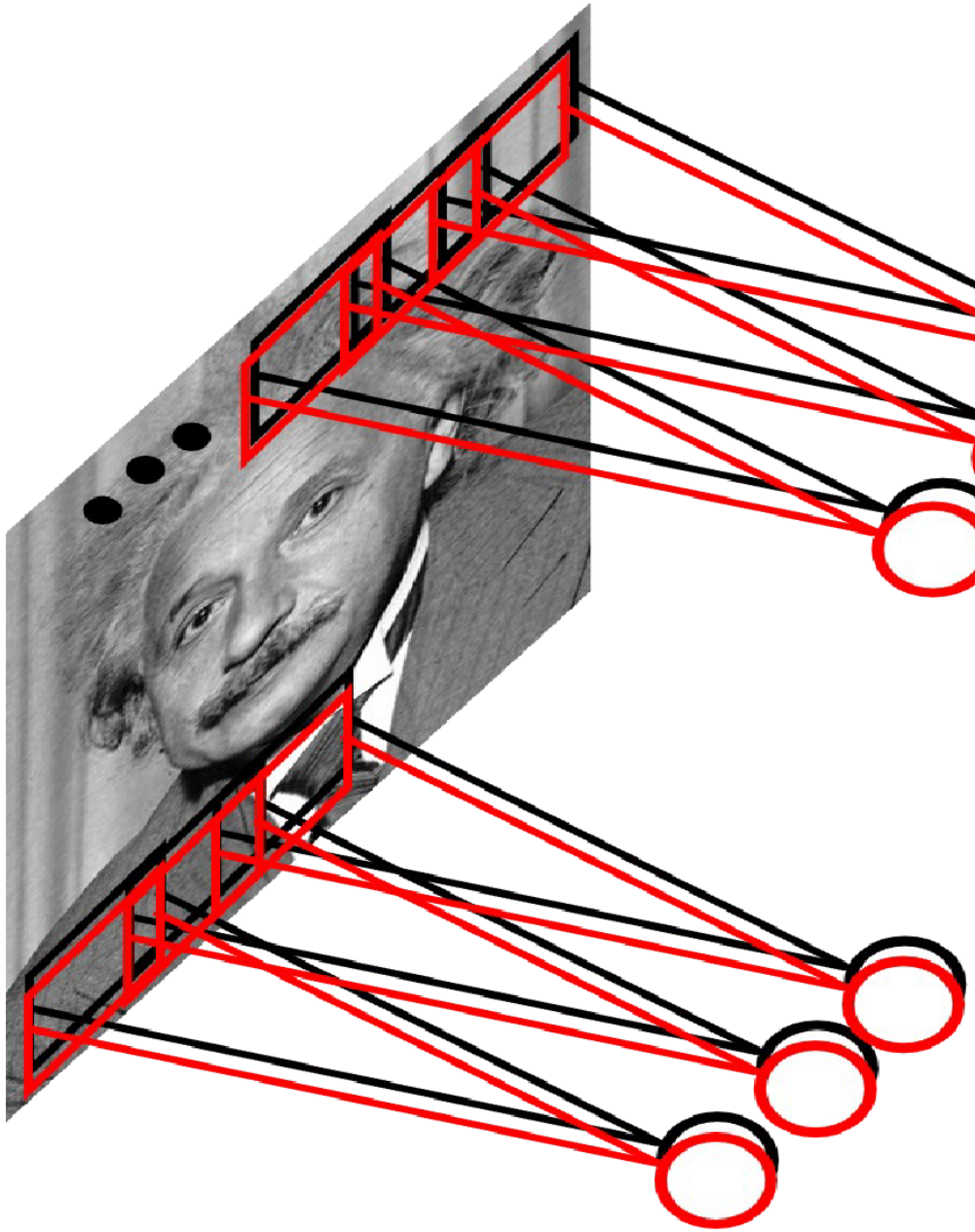
Convolutional Layer

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

Shared weights



Convolutional Layer

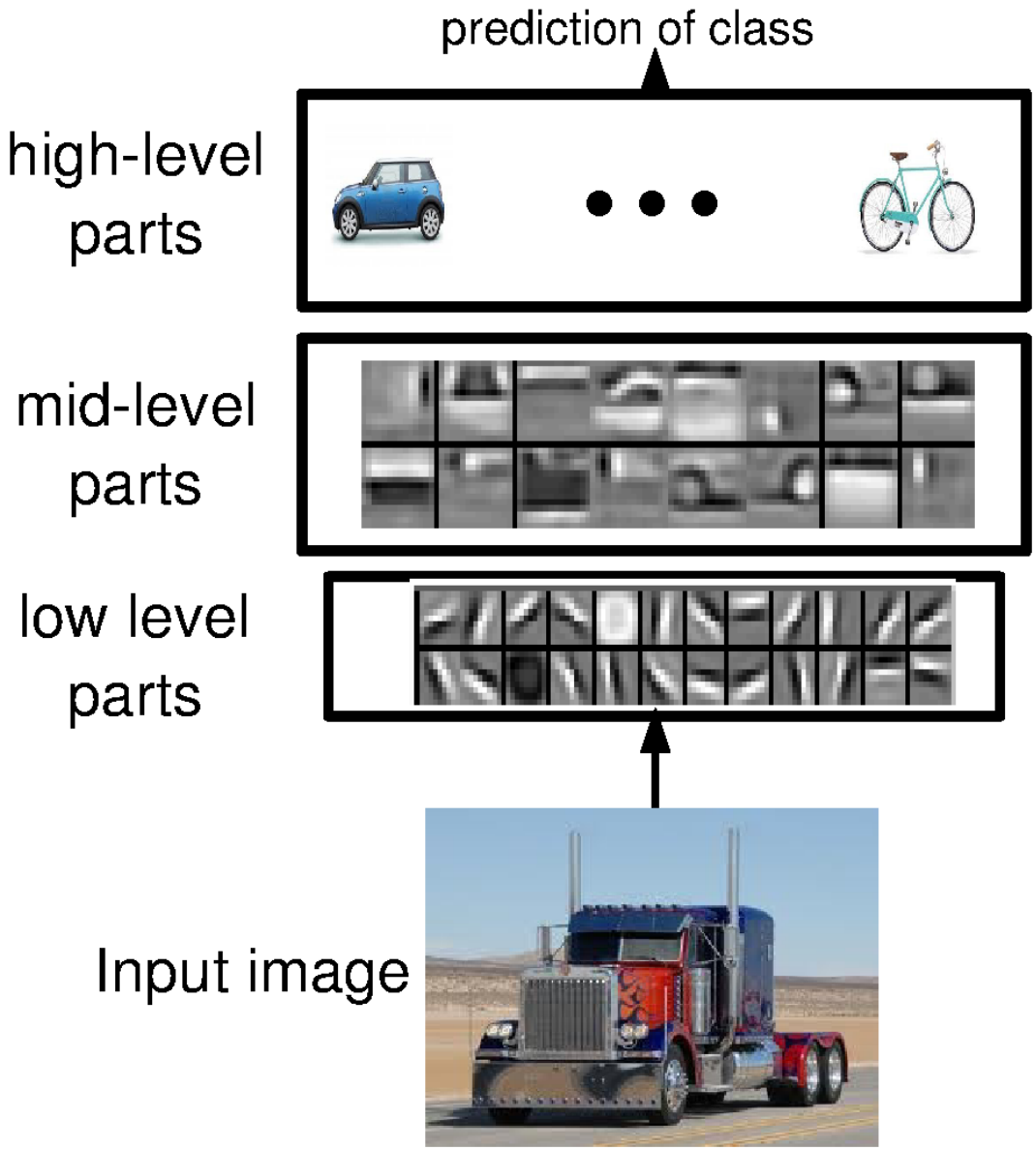


Learn multiple filters.

Filter = 'local' perceptron.
Also called *kernel.*

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

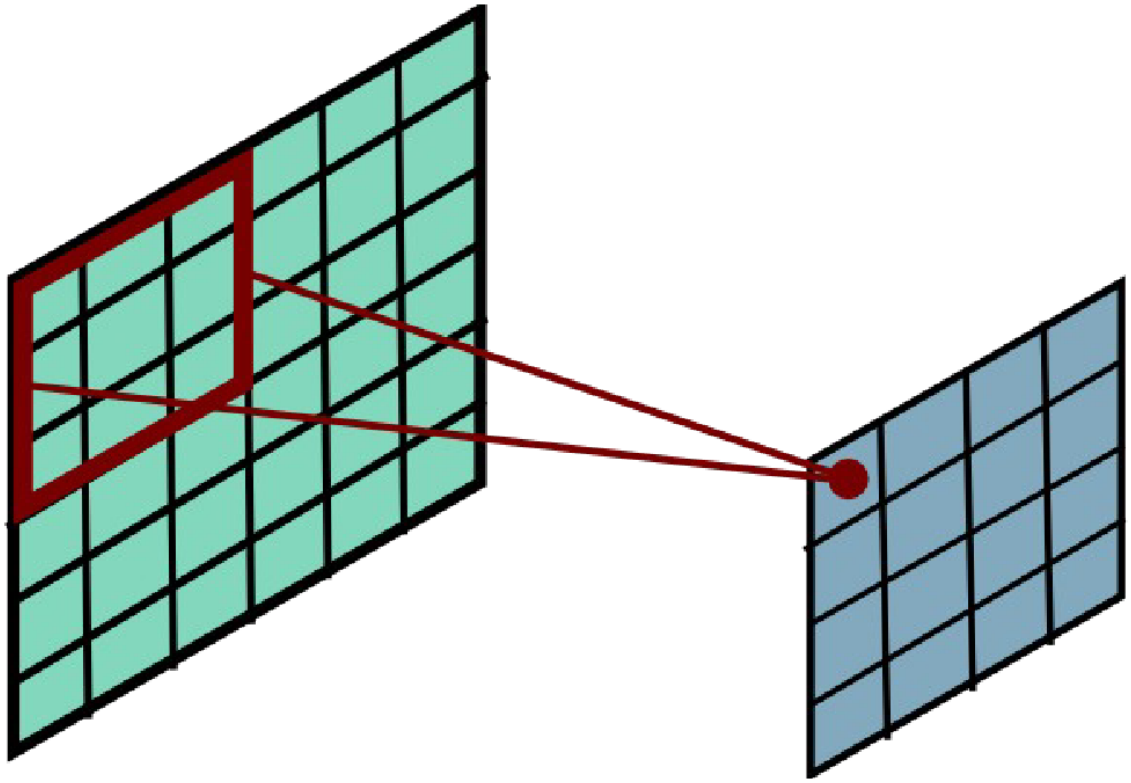
Interpretation



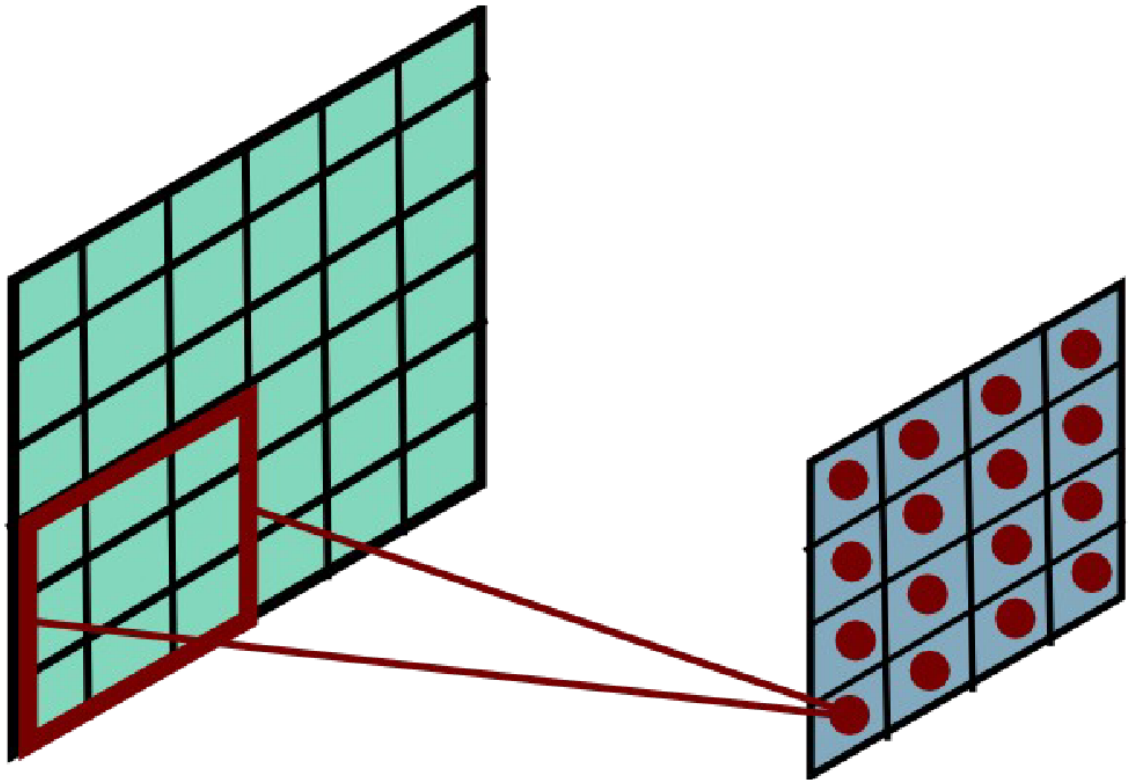
- distributed representations
- feature sharing
- compositionality

Lee et al. “Convolutional DBN's ...” ICML 2009

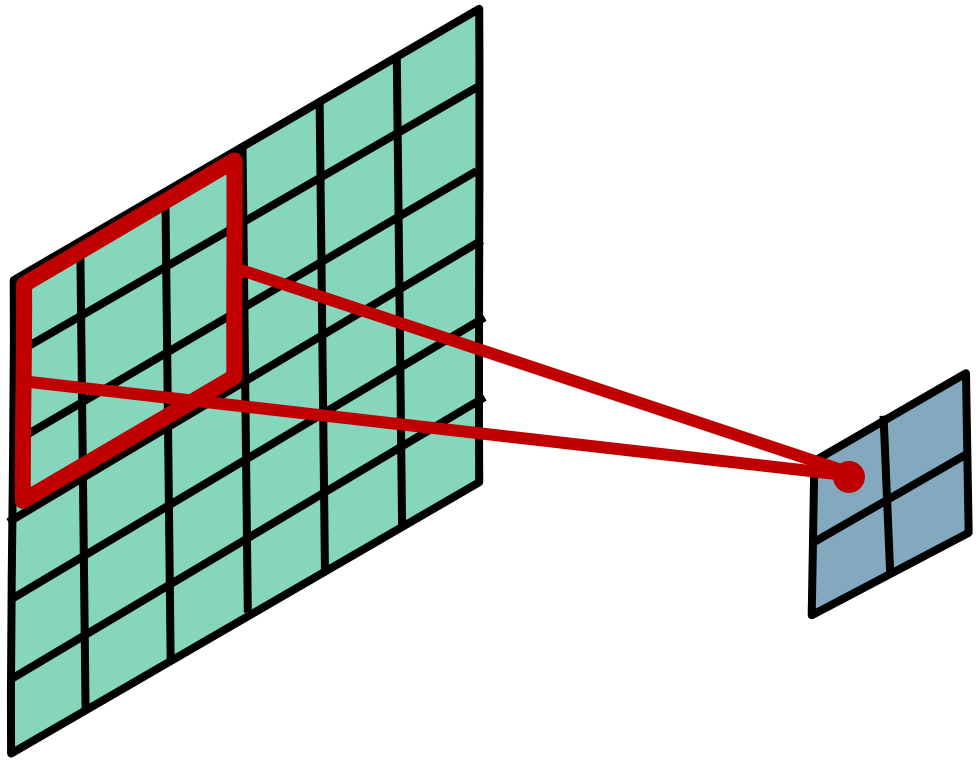
Stride = 1



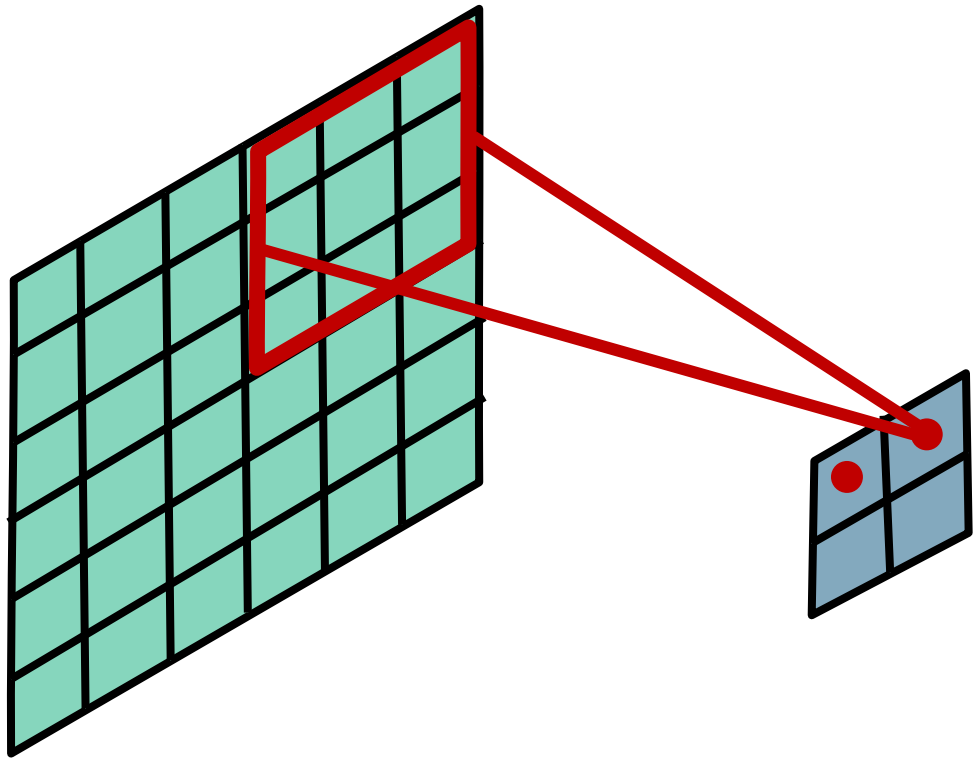
Stride = 1



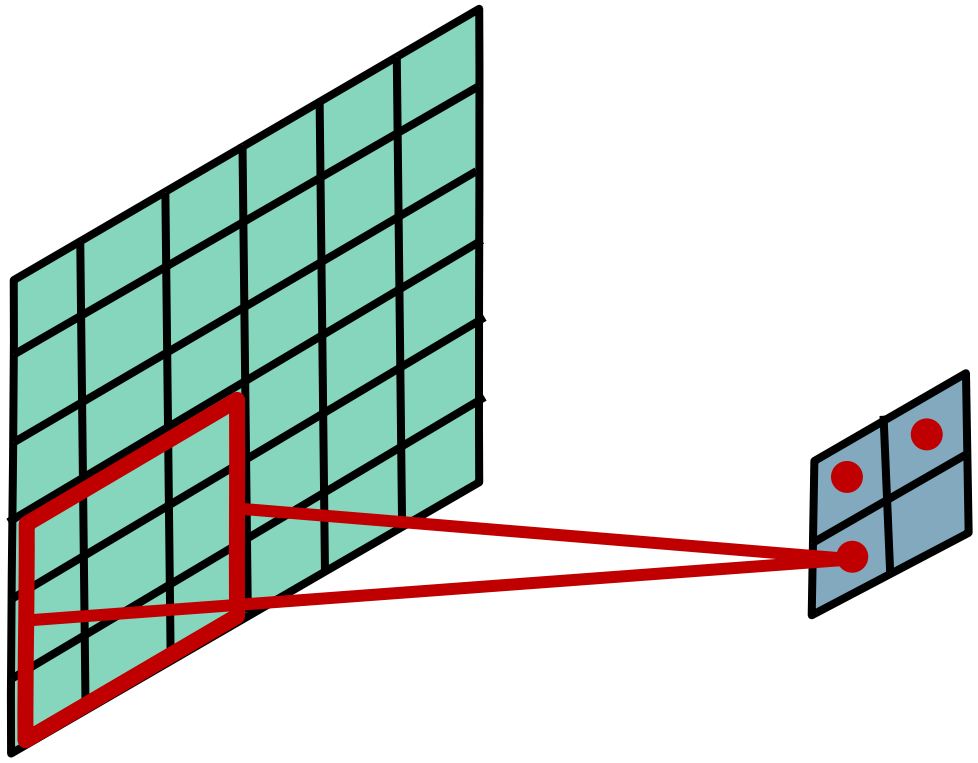
Stride = 3



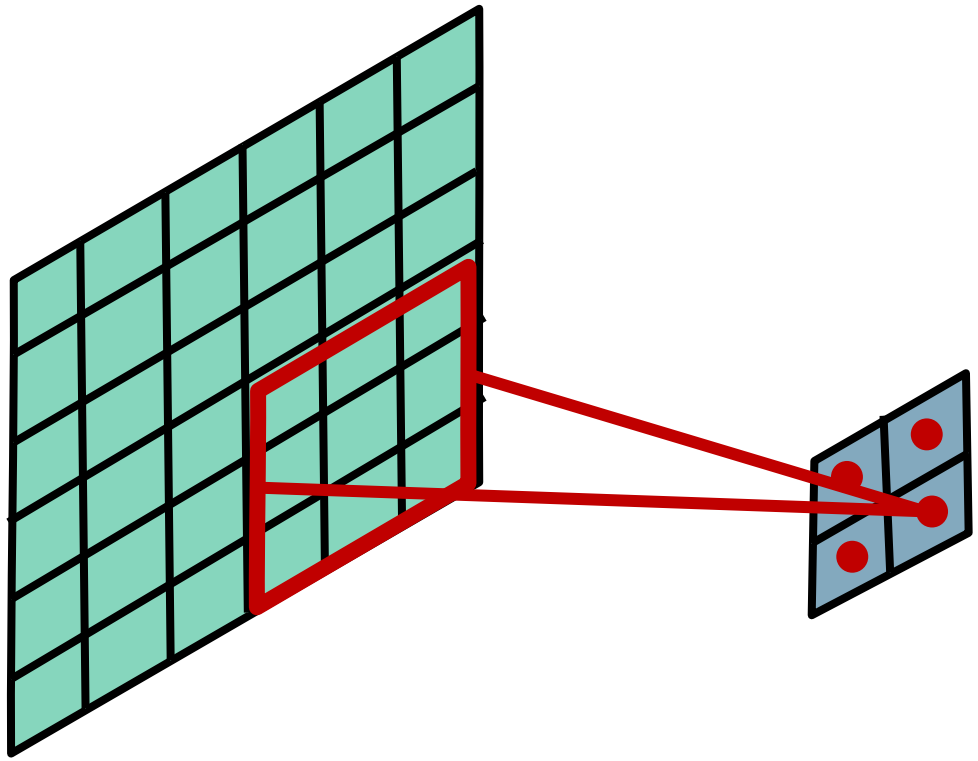
Stride = 3



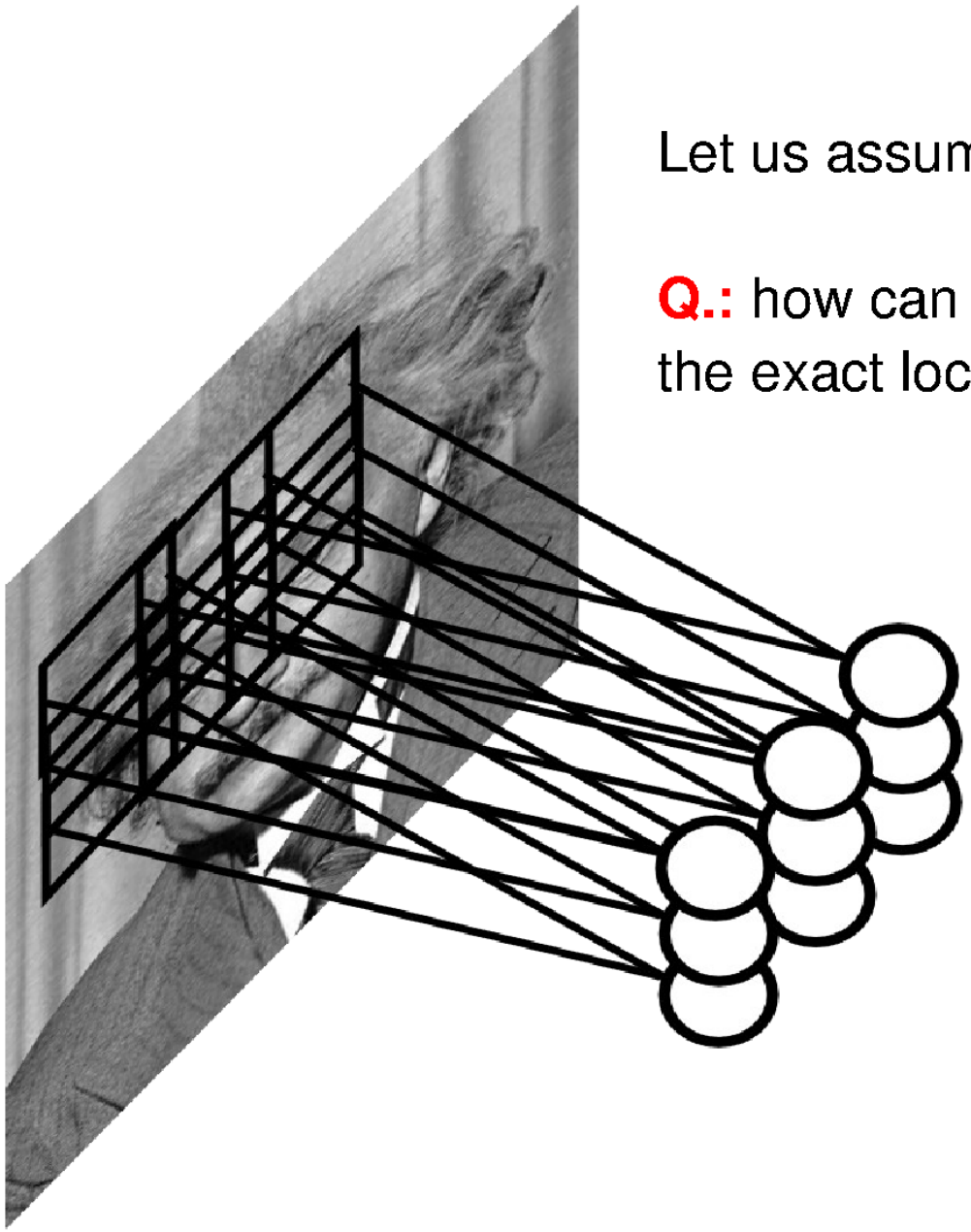
Stride = 3



Stride = 3



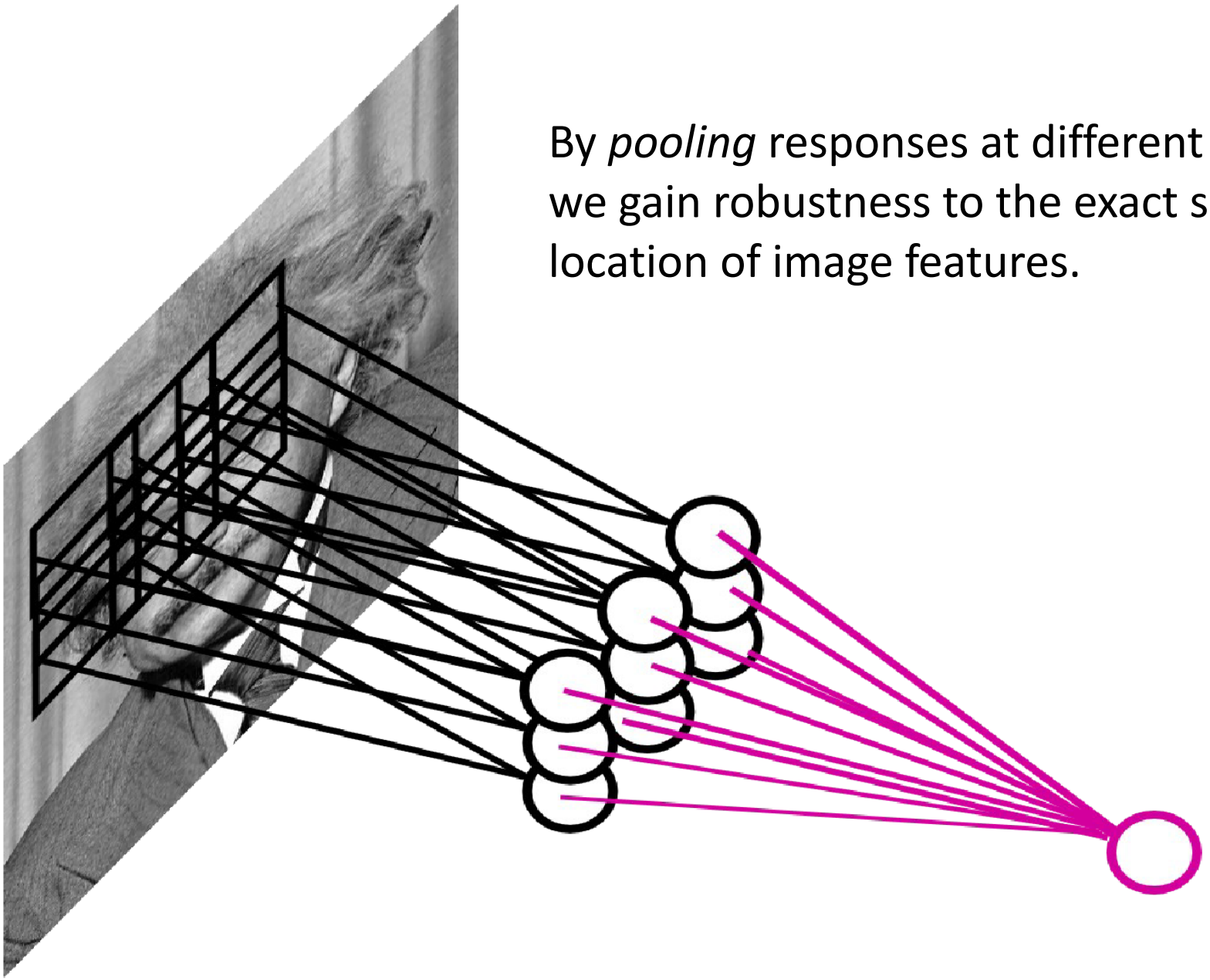
Pooling Layer



Let us assume filter is an “eye” detector.

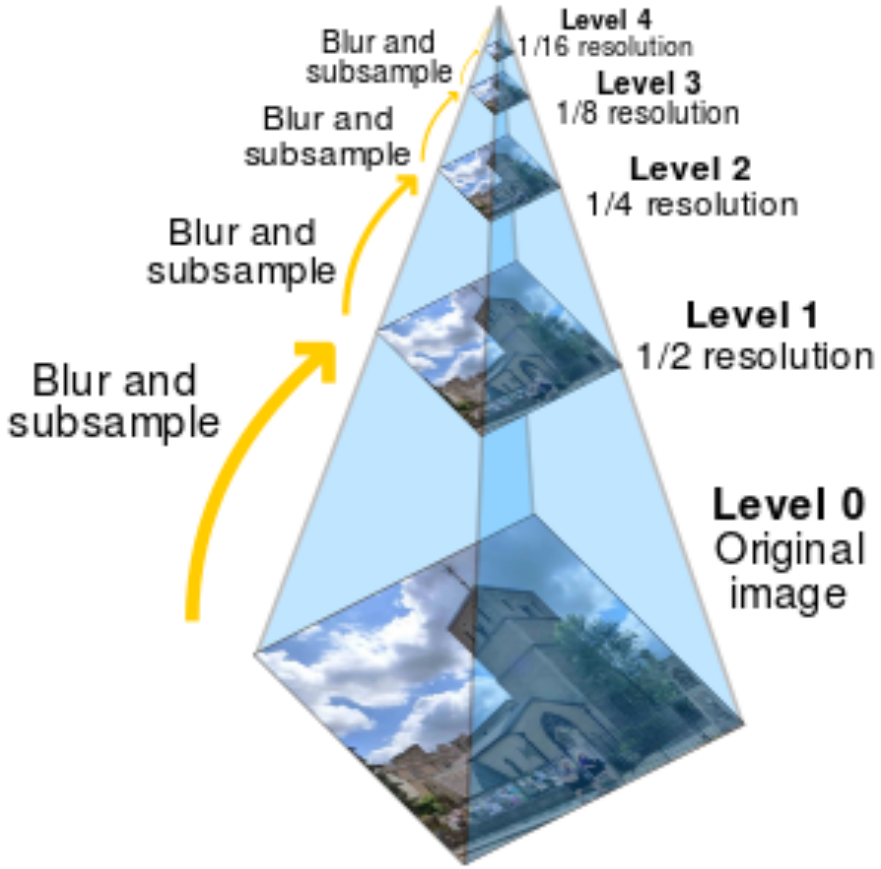
Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer



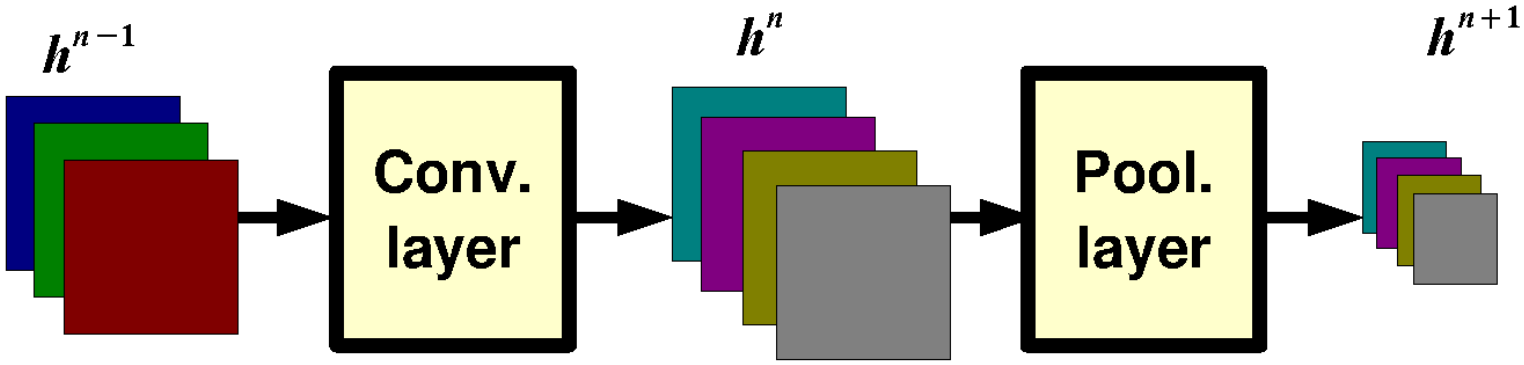
By *pooling* responses at different locations, we gain robustness to the exact spatial location of image features.

Pooling is similar to downsampling



…except sometimes we don't want to blur,
as other functions might be better for classification.

Pooling Layer: Receptive Field Size



Pooling Layer: Examples

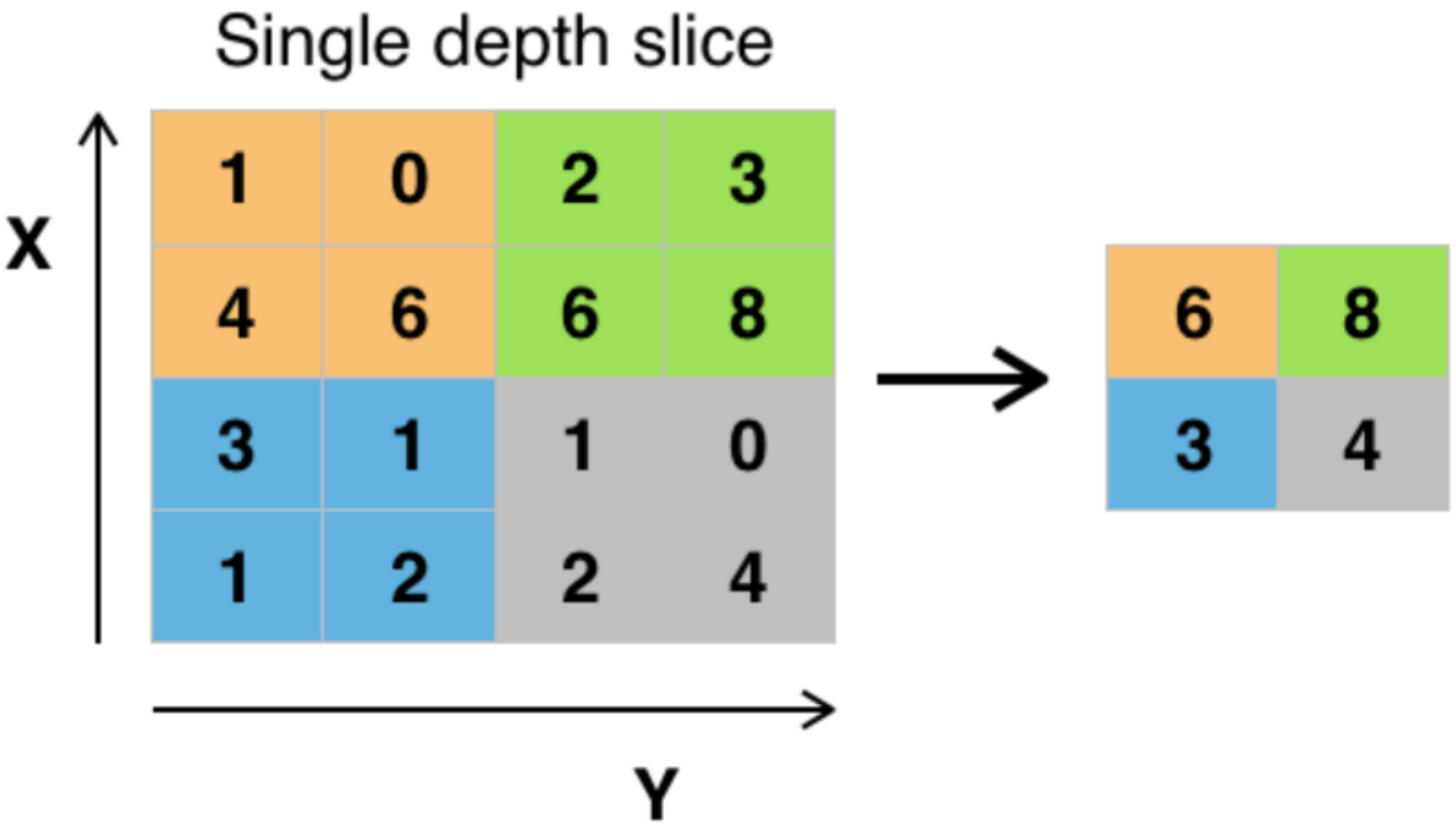
Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

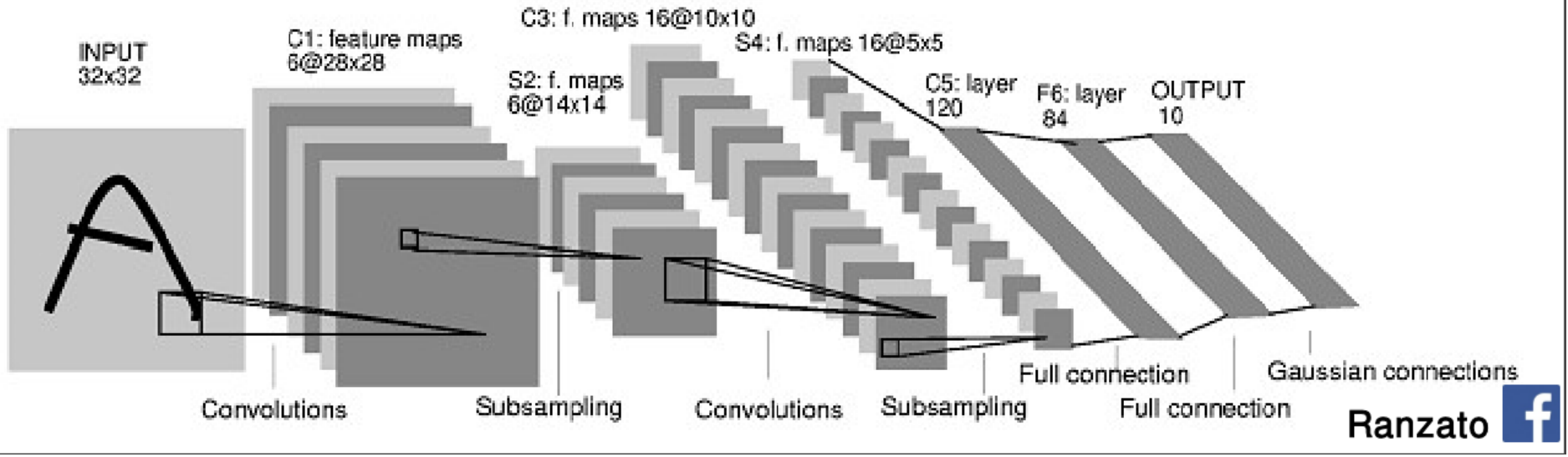
$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Max pooling

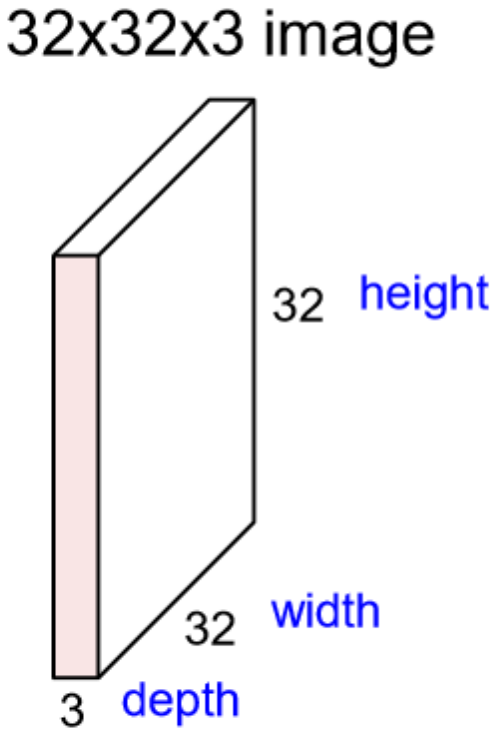


Wikipedia

Yann LeCun's MNIST CNN architecture

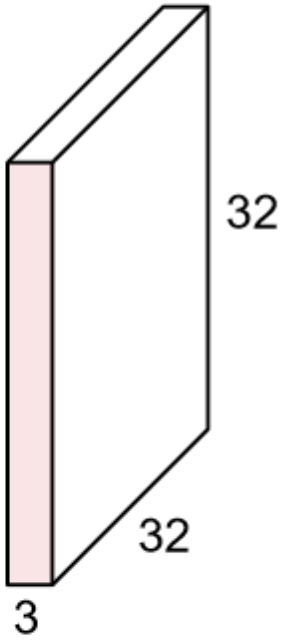


Convolutions: More detail



Convolutions: More detail

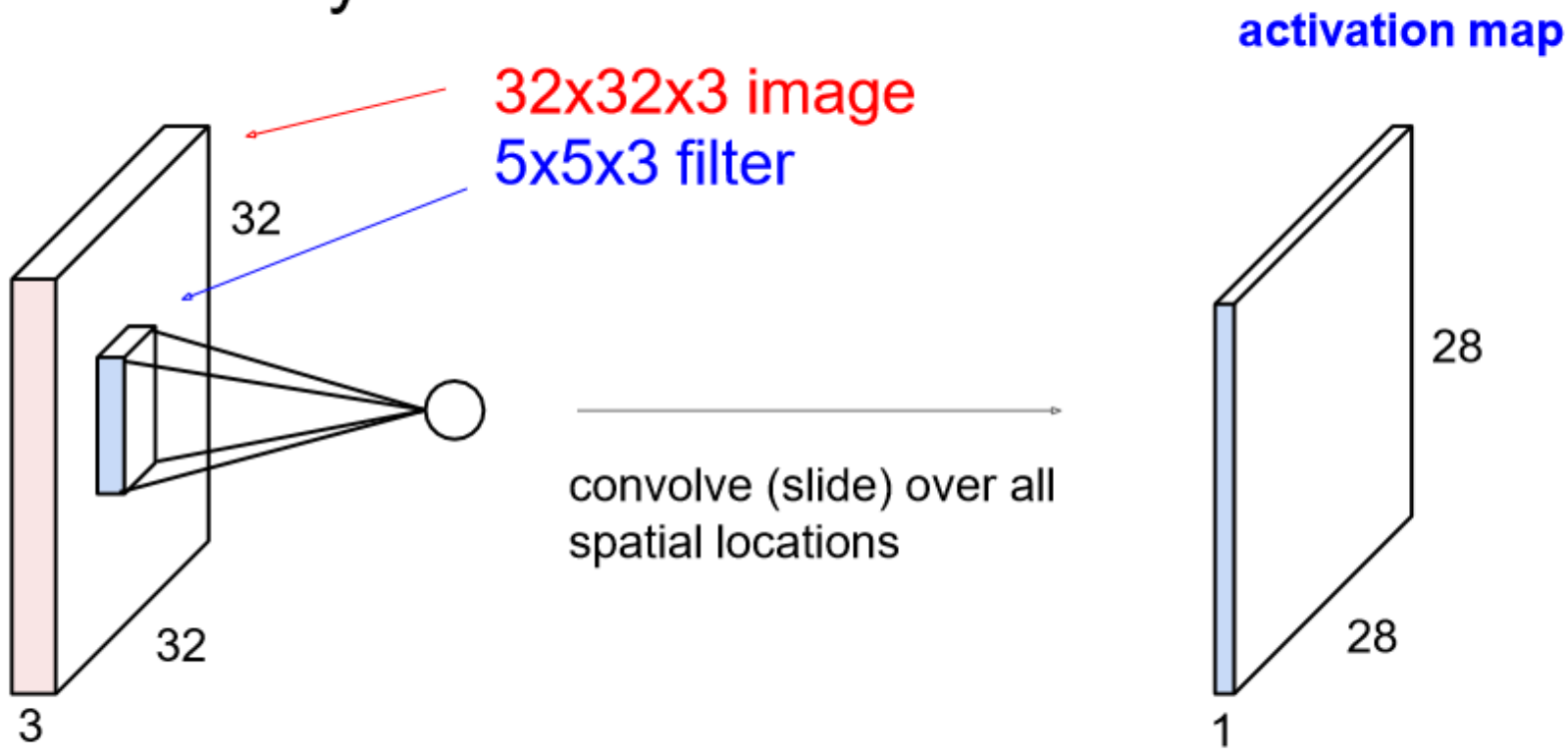
32x32x3 image



5x5x3 filter

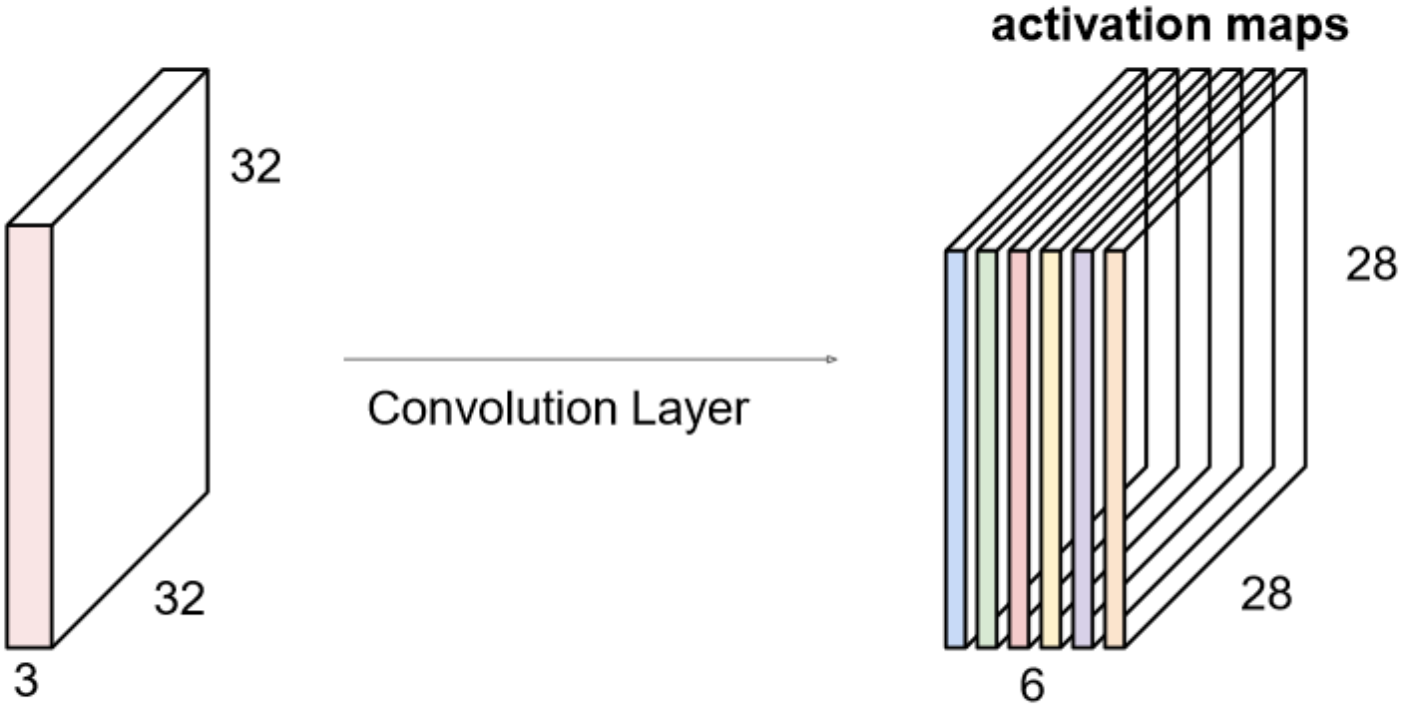
Convolutions: More detail

Convolution Layer



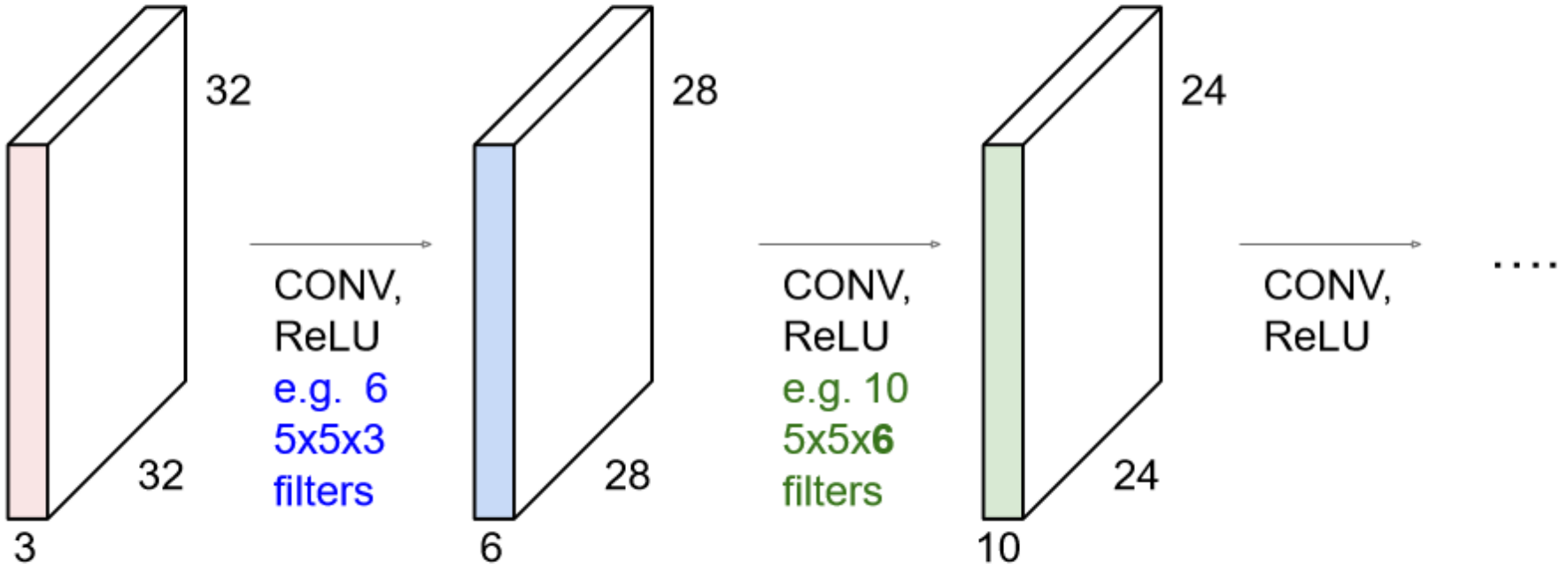
Convolutions: More detail

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

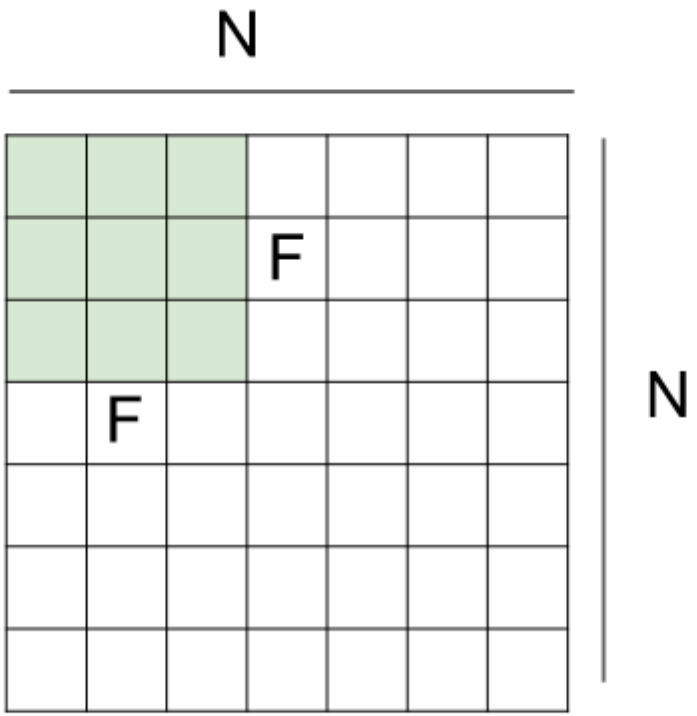


We stack these up to get a "new image" of size 28x28x6!

Convolutions: More detail

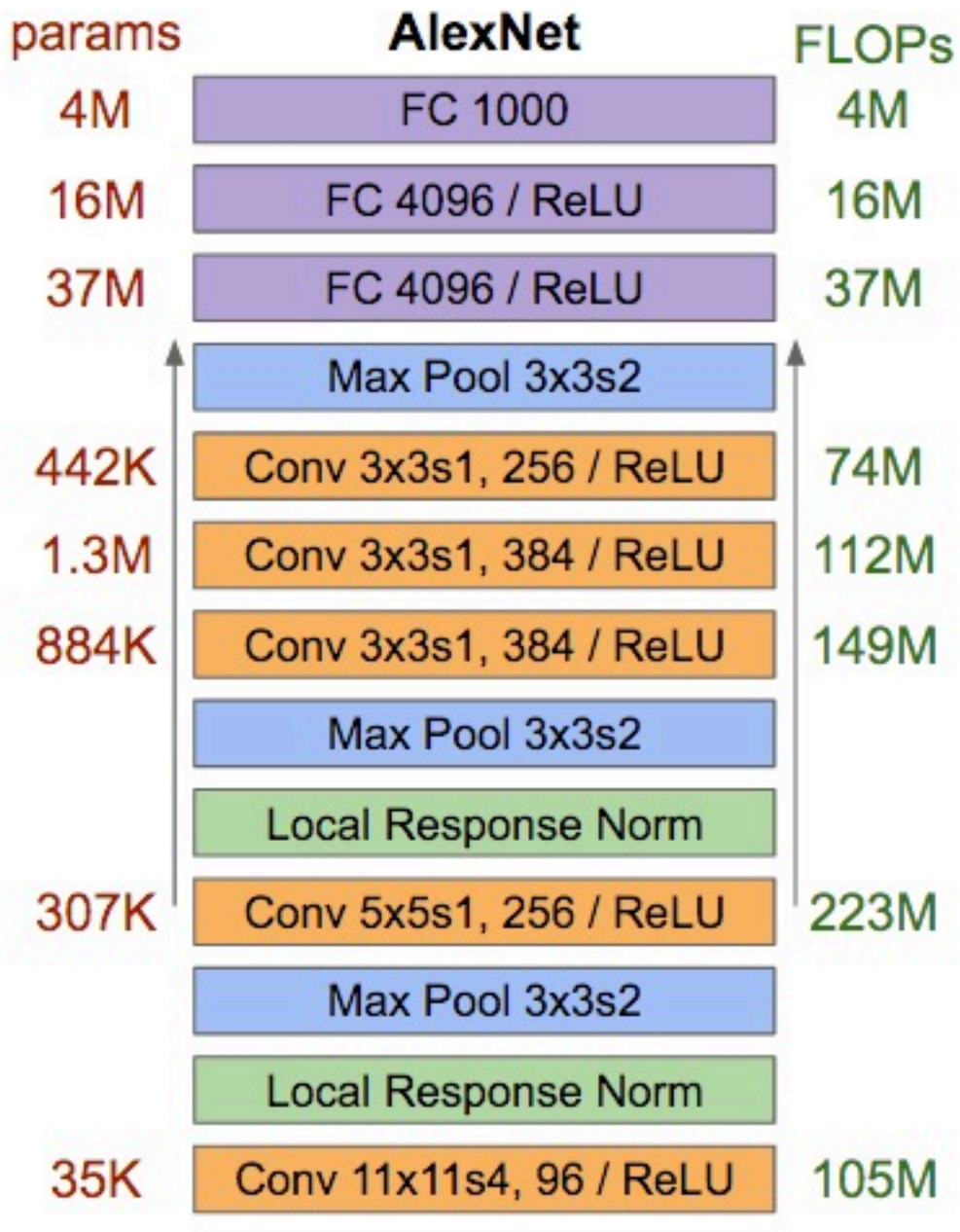


Convolutions: More detail



Output size:
(N - F) / stride + 1

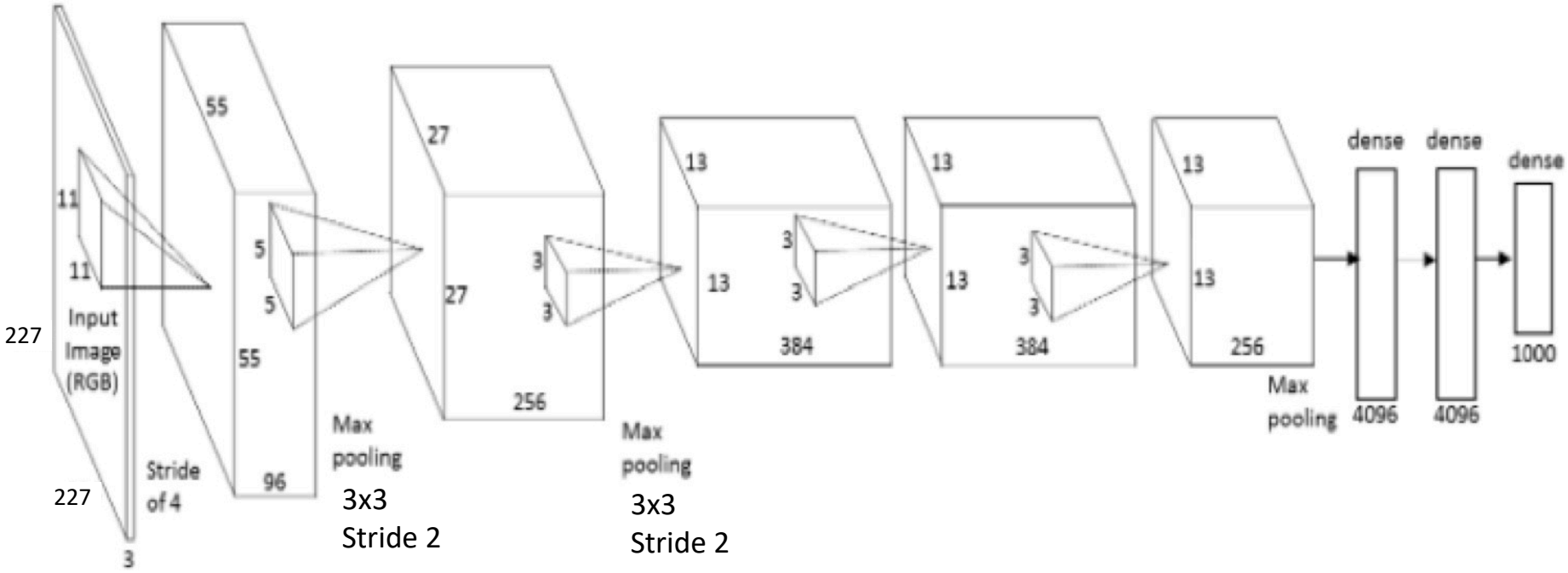
Layers
- Kernel sizes
- Strides
- # channels
- # kernels
- Max pooling



[Krizhevsky et al. 2012]

AlexNet diagram (simplified)

Input size
227 x 227 x 3



Conv 1
11 x 11 x 3
Stride 4
96 filters

Conv 2
5 x 5 x 48
Stride 1
256 filters

Conv 3
3 x 3 x 256
Stride 1
384 filters

Conv 4
3 x 3 x 192
Stride 1
384 filters

Conv 4
3 x 3 x 192
Stride 1
256 filters