

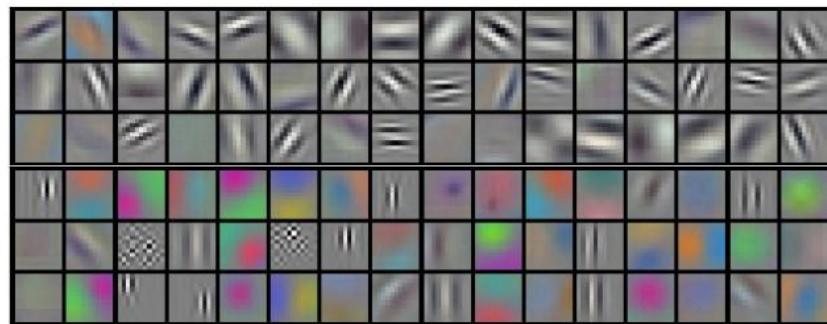
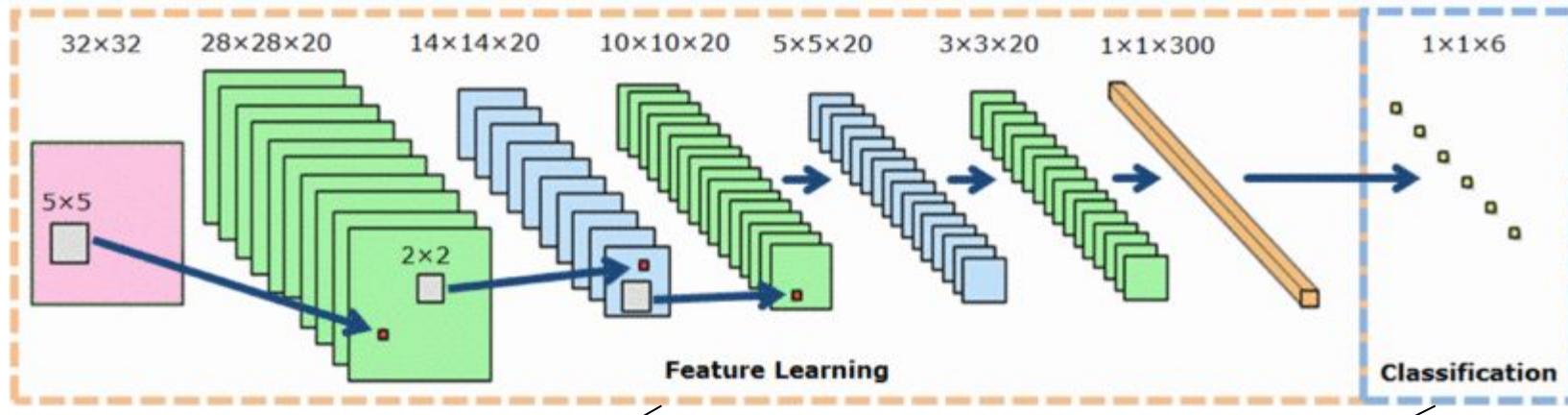
# Visual Learning

Prof. Barbara Caputo  
Dip. Ingegneria Informatica, Automatica e Gestionale, Roma



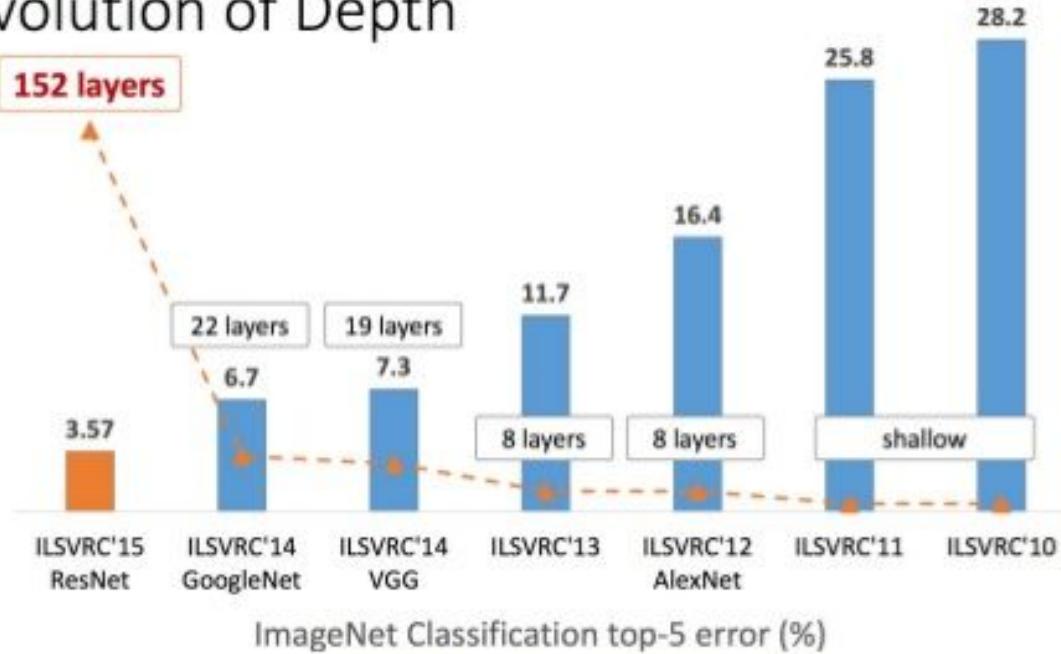
SAPIENZA  
UNIVERSITÀ DI ROMA

# Deep Convolutional Neural Networks for Image Classification



# Deep Convolutional Neural Networks for Image Classification

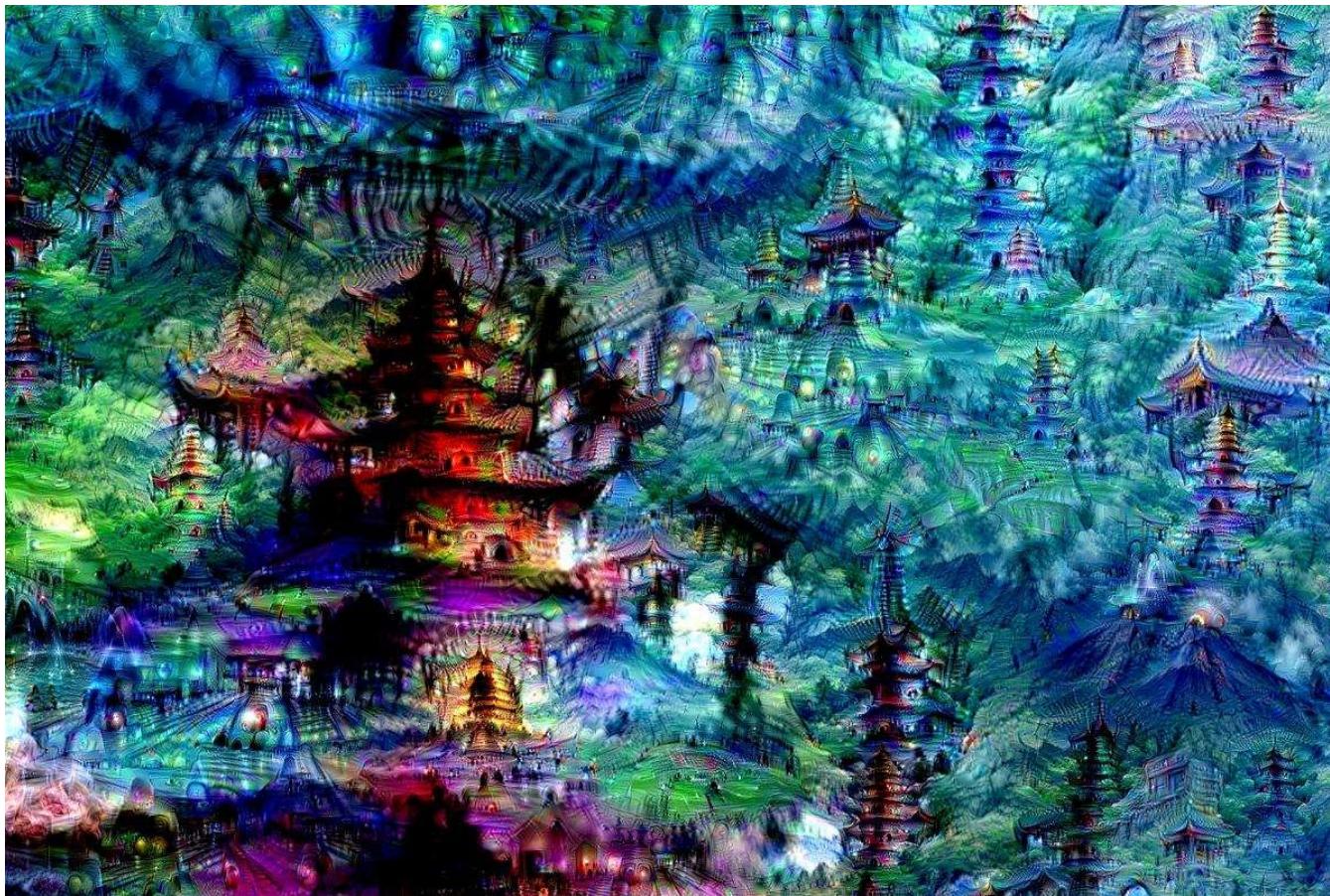
## Revolution of Depth



He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. ["Deep Residual Learning for Image Recognition."](#) arXiv preprint arXiv:1512.03385 (2015). [\[slides\]](#)

80

# Google AI Dream



# Google AI Dream



# Neural Art



# Neural Art

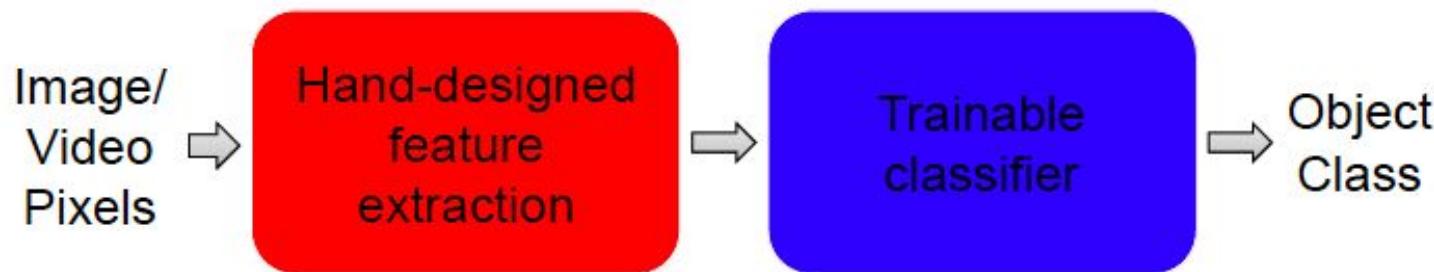


# Overview

---

- Shallow vs. deep architectures
- Background
  - Traditional neural networks
  - Inspiration from neuroscience
- Stages of CNN architecture

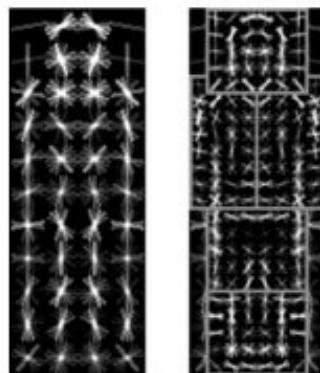
# Traditional Recognition Approach



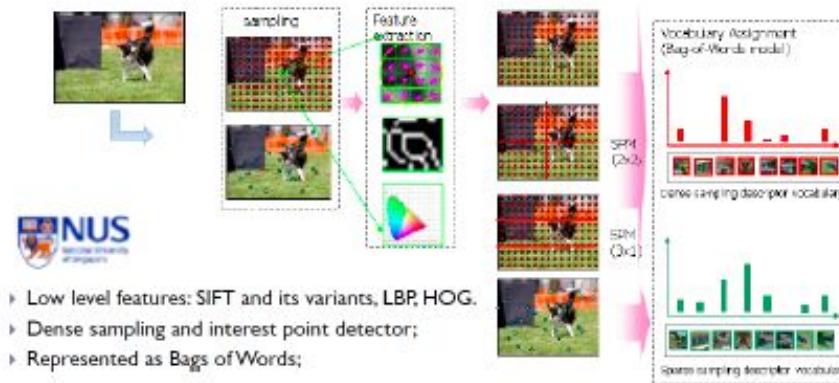
- Features are not learned
- Trainable classifier is often generic (e.g. SVM)

# Traditional Recognition Approach

- Features are key to recent progress in recognition
- Multitude of hand-designed features currently in use
  - SIFT, HOG, .....
- Where next? Better classifiers? Or keep building more features?



Felzenszwalb, Girshick,  
McAllester and Ramanan, PAMI 2007



Yan & Huang  
(Winner of PASCAL 2010 classification competition)

# What about learning the features?

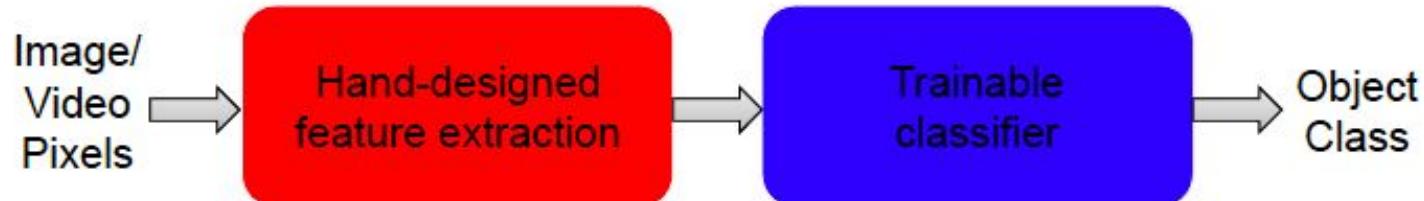
- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly



# “Shallow” vs. “deep” architectures

---

## Traditional recognition: “Shallow” architecture



## Deep learning: “Deep” architecture



# Background: Perceptrons

---

Input

Weights

$$x_1 \quad w_1$$

$$x_2 \quad w_2$$

$$x_3 \quad w_3$$

.

.

$$x_d \quad w_d$$

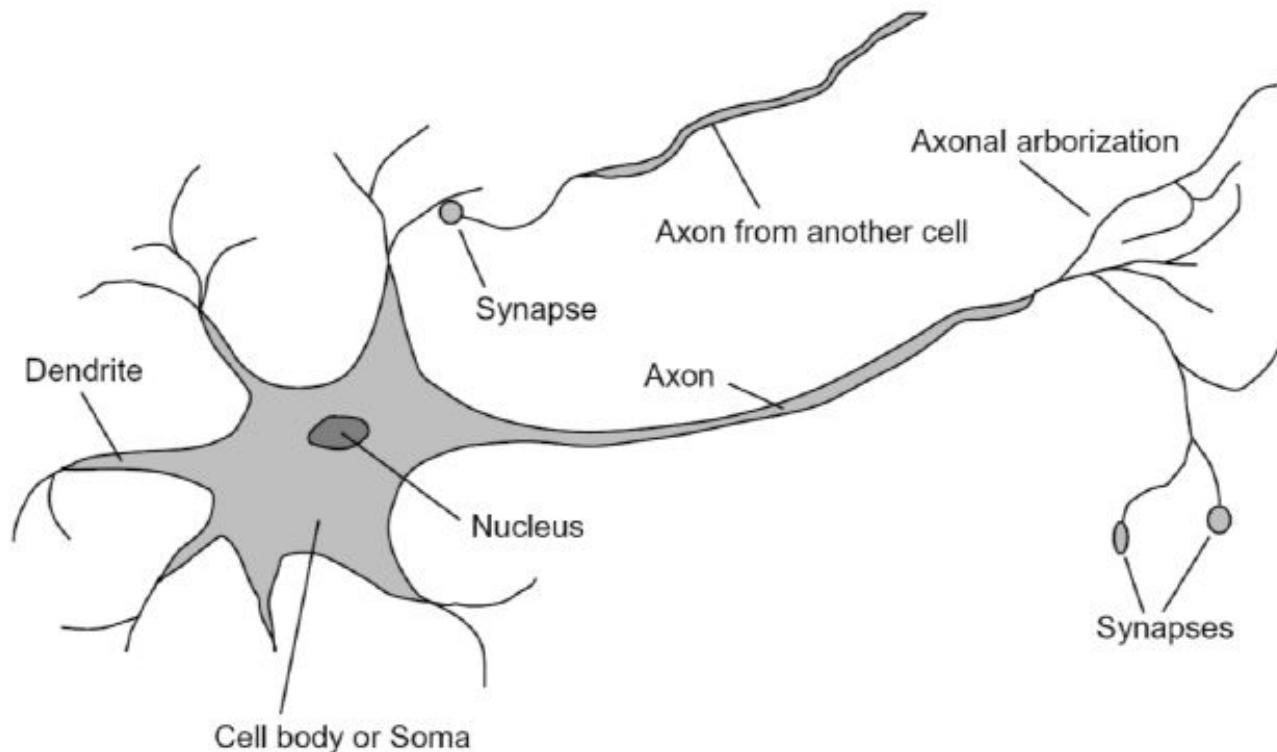
Output:  $\sigma(w \cdot x + b)$

Sigmoid function:

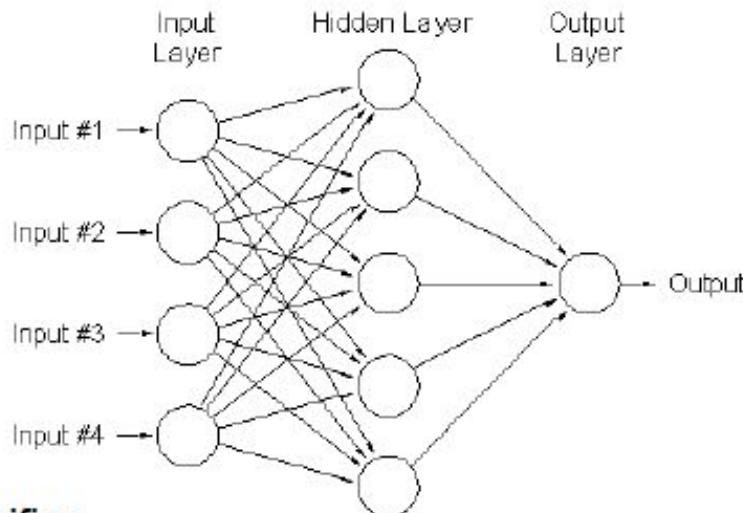
$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

# Inspiration: Neuron cells

---



# Background: Multi-Layer Neural Networks



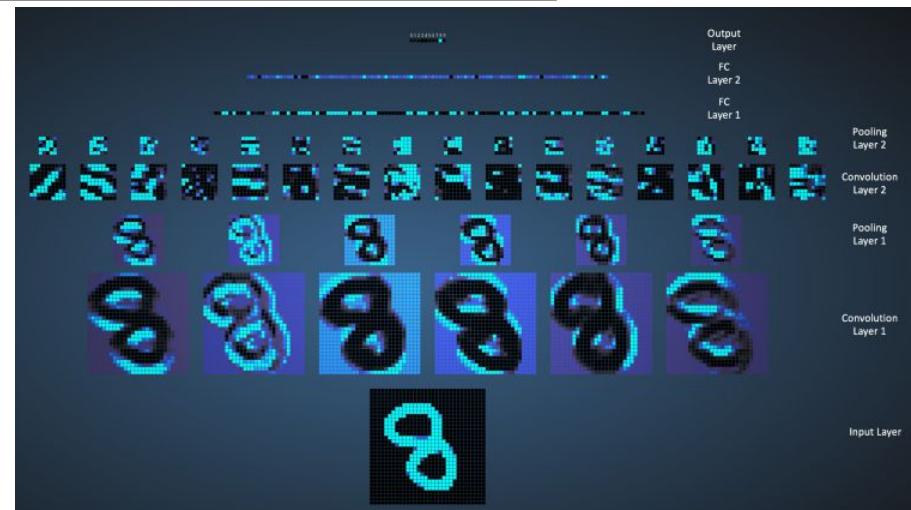
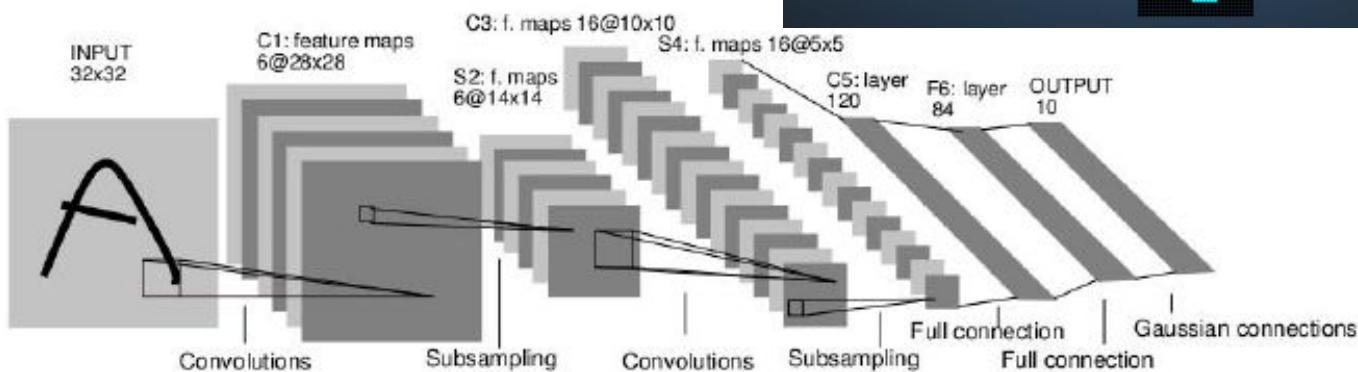
- Nonlinear classifier
- **Training:** find network weights  $w$  to minimize the error between true training labels  $y_i$  and estimated labels  $f_w(\mathbf{x}_i)$ :

$$E(\mathbf{w}) = \sum_{i=1}^N (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2$$

- Minimization can be done by gradient descent provided  $f$  is differentiable
  - This training method is called **back-propagation**

# Convolutional Neural Networks (CNN, Convnet)

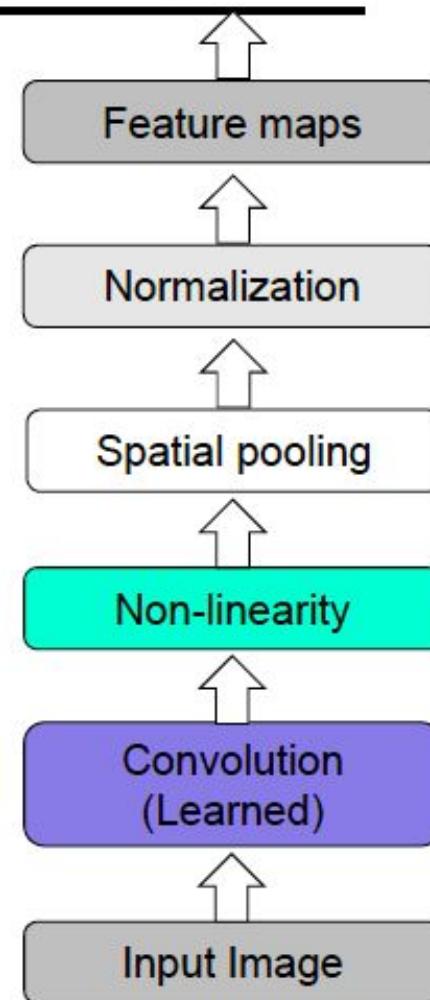
- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, [Gradient-based learning applied to document recognition](#), Proceedings of the IEEE 86(11): 2278–2324, 1998.

# Convolutional Neural Networks (CNN, Convnet)

- Feed-forward feature extraction:
  1. Convolve input with learned filters
  2. Non-linearity
  3. Spatial pooling
  4. Normalization
- Supervised training of convolutional filters by back-propagating classification error



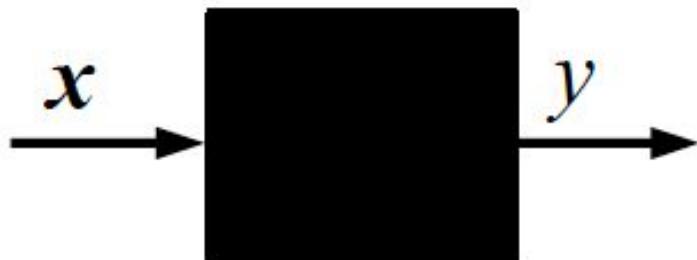
# Supervised Learning

$\{(\mathbf{x}^i, y^i), i=1 \dots P\}$  training dataset

$\mathbf{x}^i$  i-th input training example

$y^i$  i-th target label

$P$  number of training examples



Goal: predict the target label of unseen inputs.

# Neural Networks

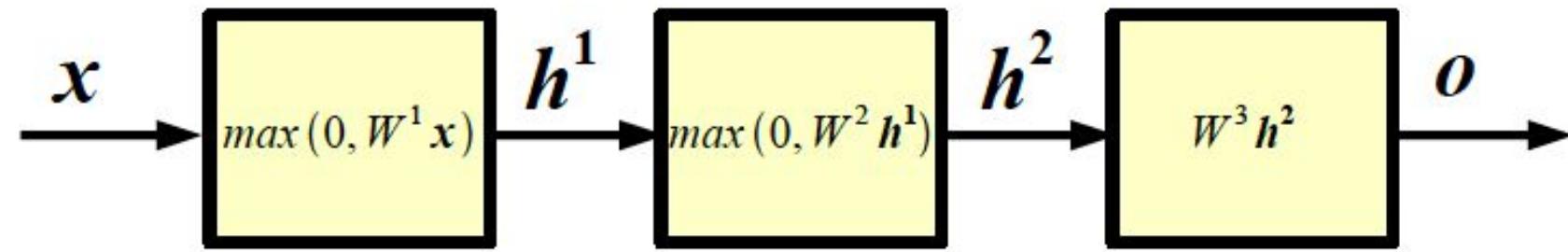
Assumptions (for the next few slides):

- The input image is vectorized (disregard the spatial layout of pixels)
- The target label is discrete (classification)

**Question:** what class of functions shall we consider to map the input into the output?

**Answer:** composition of simpler functions.

# Neural Networks: example



$x$  input

$h^1$  1-st layer hidden units

$h^2$  2-nd layer hidden units

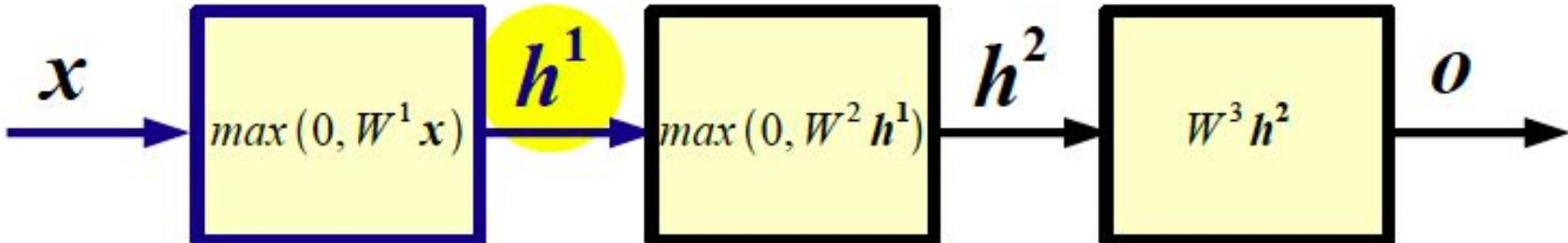
$o$  output

Example of a 2 hidden layer neural network (or 4 layer network,  
counting also input and output).

# Forward Propagation

**Def.:** Forward propagation is the process of computing the output of the network given its input.

# Forward Propagation



$$x \in R^D \quad W^1 \in R^{N_1 \times D} \quad b^1 \in R^{N_1} \quad h^1 \in R^{N_1}$$

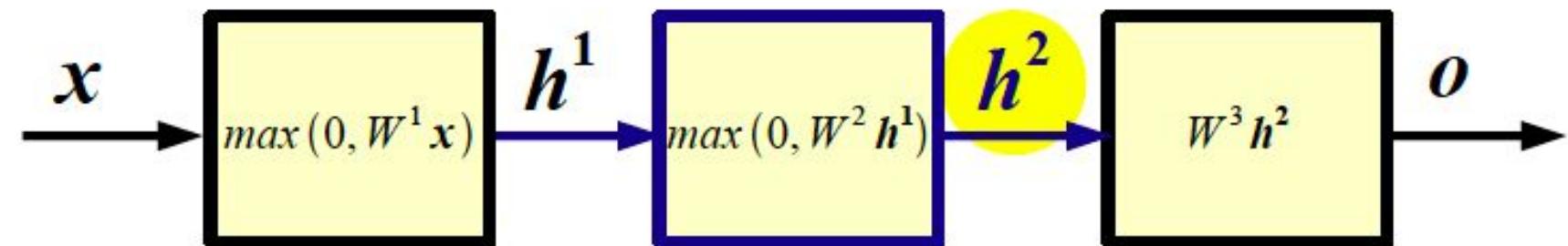
$$h^1 = \max(0, W^1 x + b^1)$$

$W^1$  1-st layer weight matrix or weights

$b^1$  1-st layer biases

The non-linearity  $u = \max(0, v)$  is called **ReLU** in the DL literature. Each output hidden unit takes as input all the units at the previous layer: each such layer is called “**fully connected**”.

# Forward Propagation



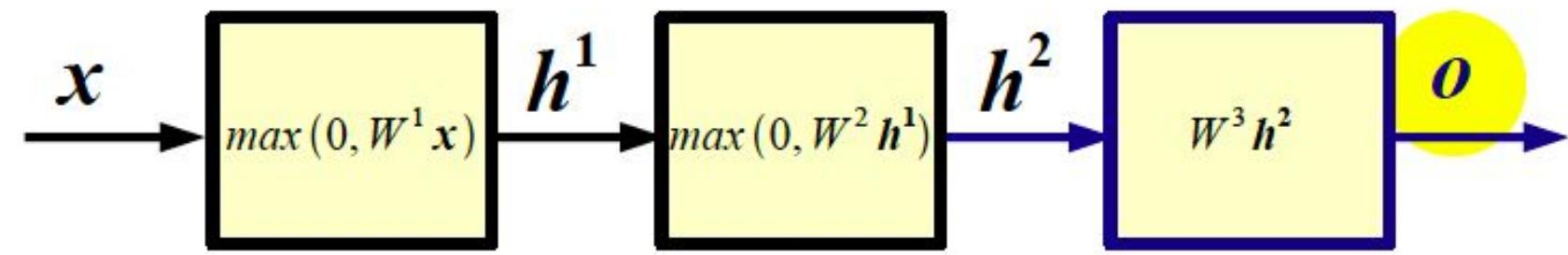
$$h^1 \in R^{N_1} \quad W^2 \in R^{N_2 \times N_1} \quad b^2 \in R^{N_2} \quad h^2 \in R^{N_2}$$

$$h^2 = \max(0, W^2 h^1 + b^2)$$

$W^2$  2-nd layer weight matrix or weights

$b^2$  2-nd layer biases

# Forward Propagation



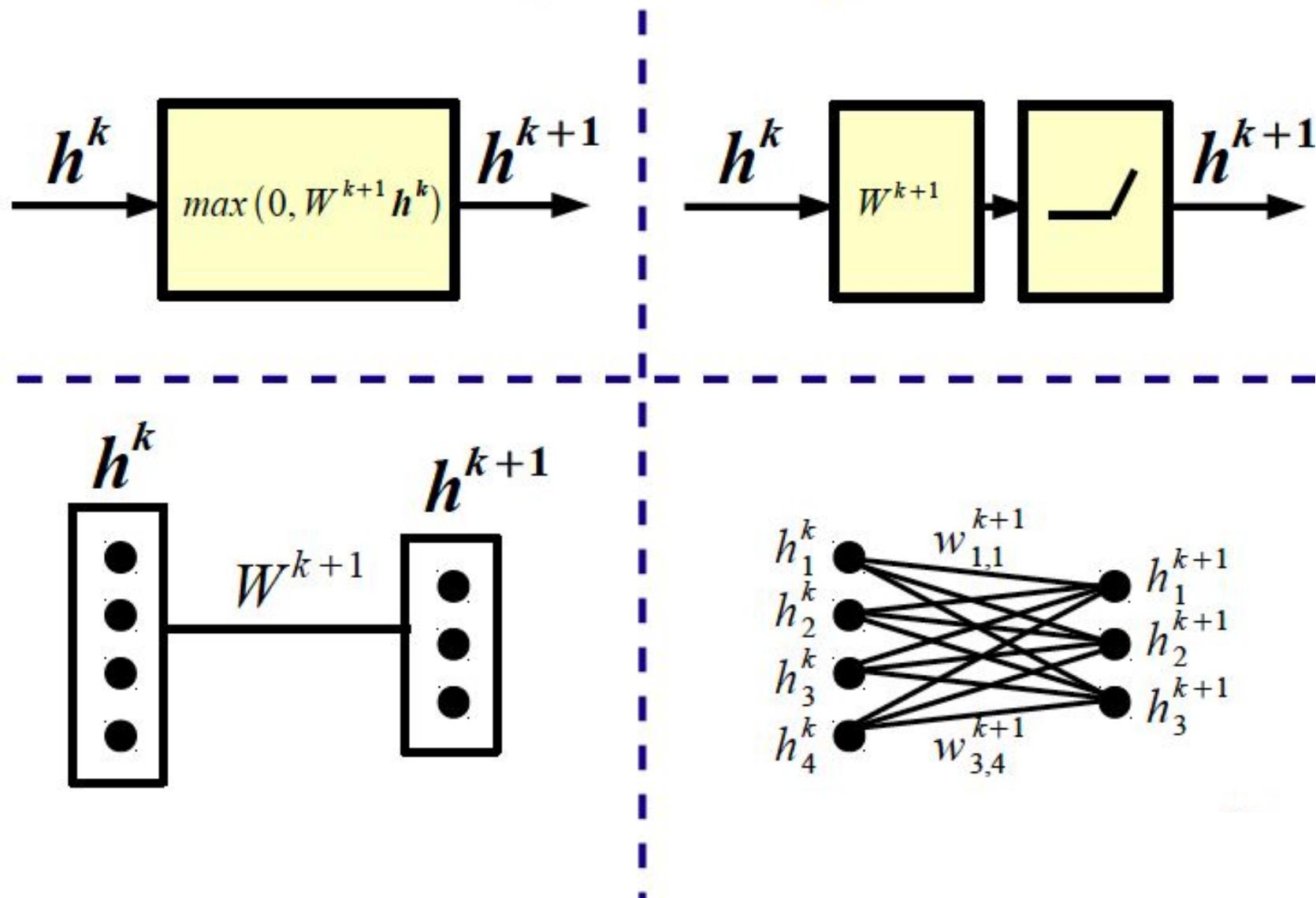
$$h^2 \in R^{N_2} \quad W^3 \in R^{N_3 \times N_2} \quad b^3 \in R^{N_3} \quad o \in R^{N_3}$$

$$o = \max(0, W^3 h^2 + b^3)$$

$W^3$  3-rd layer weight matrix or weights

$b^3$  3-rd layer biases

# Alternative Graphical Representation



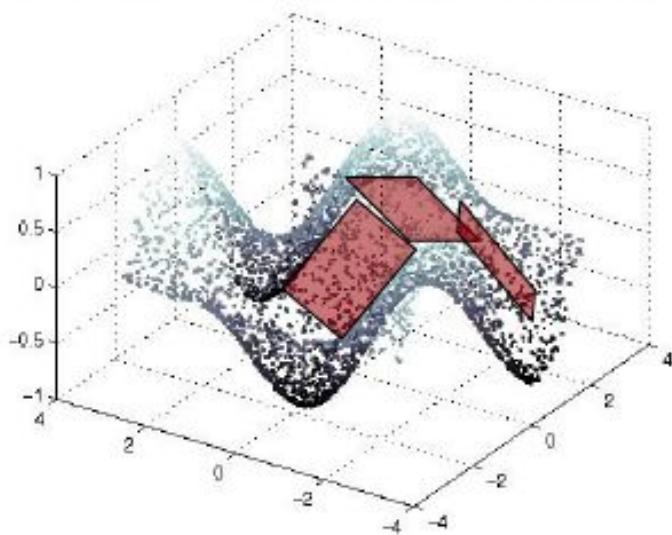
# Interpretation

**Question:** Why can't the mapping between layers be linear?

**Answer:** Because composition of linear functions is a linear function.  
Neural network would reduce to (1 layer) logistic regression.

**Question:** What do ReLU layers accomplish?

**Answer:** Piece-wise linear tiling: mapping is locally linear.

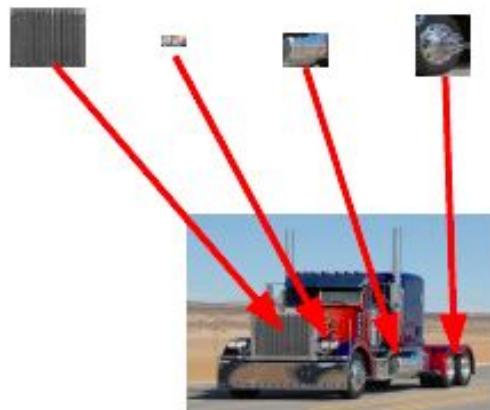


# Interpretation

**Question:** Why do we need many layers?

**Answer:** When input has hierarchical structure, the use of a hierarchical architecture is potentially more efficient because intermediate computations can be re-used. DL architectures are efficient also because they use **distributed representations** which are shared across classes.

[0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 ... ] truck feature

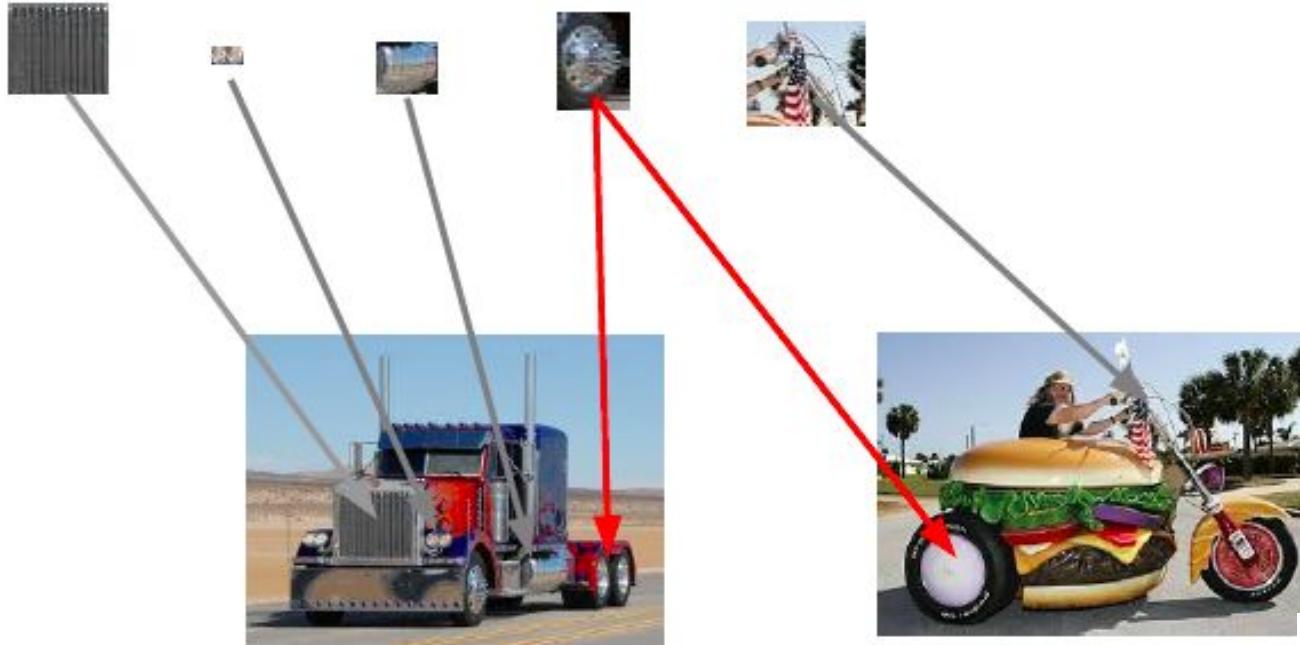


Exponentially more efficient than a  
1-of-N representation (a la k-means)

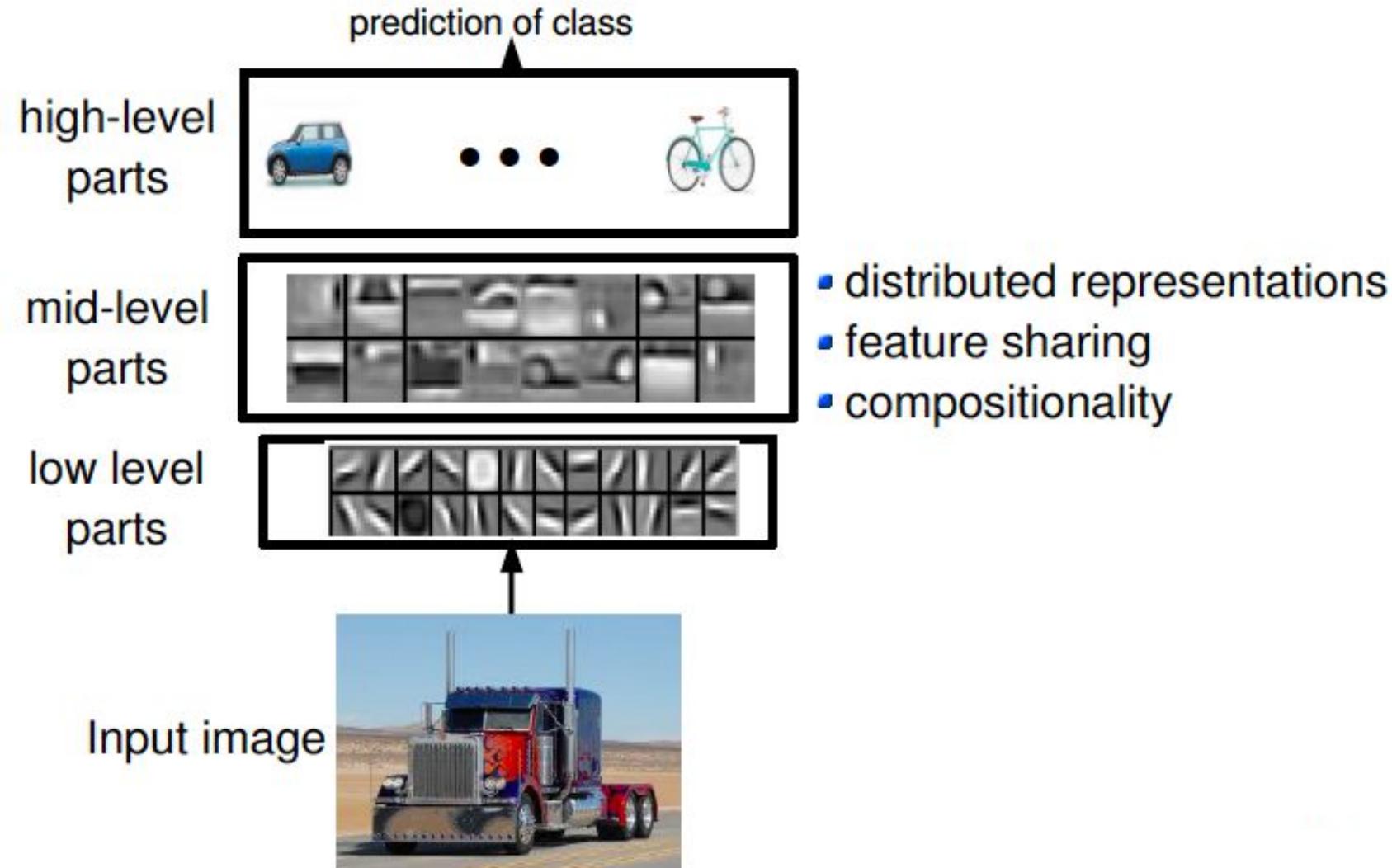
# Interpretation

[1 1 0 0 0 1 0 **1** 0 0 0 0 1 1 0 1... ]    motorbike

[0 0 1 0 0 0 0 **1** 0 0 1 1 0 0 1 0 ... ]    truck



# Interpretation



Lee et al. "Convolutional DBN's ..." ICML 2009

# Interpretation

**Question:** What does a hidden unit do?

**Answer:** It can be thought of as a classifier or feature detector.

**Question:** How many layers? How many hidden units?

**Answer:** Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

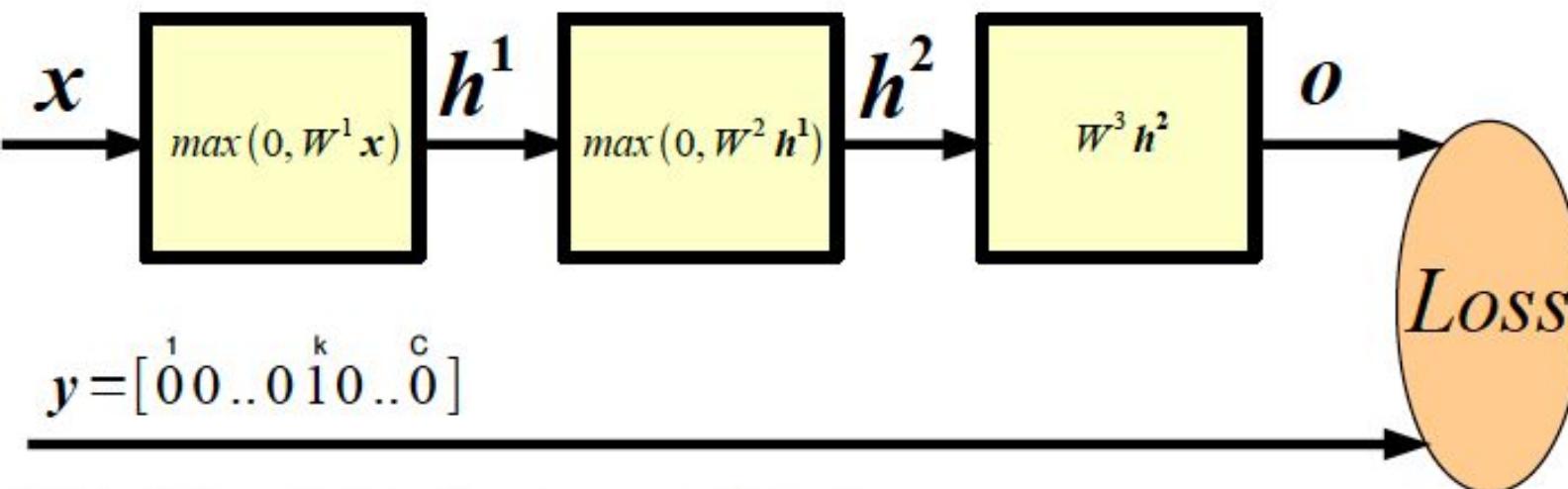
**Question:** How do I set the weight matrices?

**Answer:** Weight matrices and biases are learned.

First, we need to define a measure of quality of the current mapping.  
Then, we need to define a procedure to adjust the parameters.

17

# How Good is a Network?



Probability of class  $k$  given input (softmax):

$$p(c_k=1|x) = \frac{e^{o_k}}{\sum_{j=1}^c e^{o_j}}$$

(Per-sample) **Loss**; e.g., negative log-likelihood (good for classification of small number of classes):

$$L(x, y; \theta) = -\sum_j y_j \log p(c_j|x)$$

# Training

**Learning** consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{n=1}^P L(\mathbf{x}^n, y^n; \boldsymbol{\theta})$$

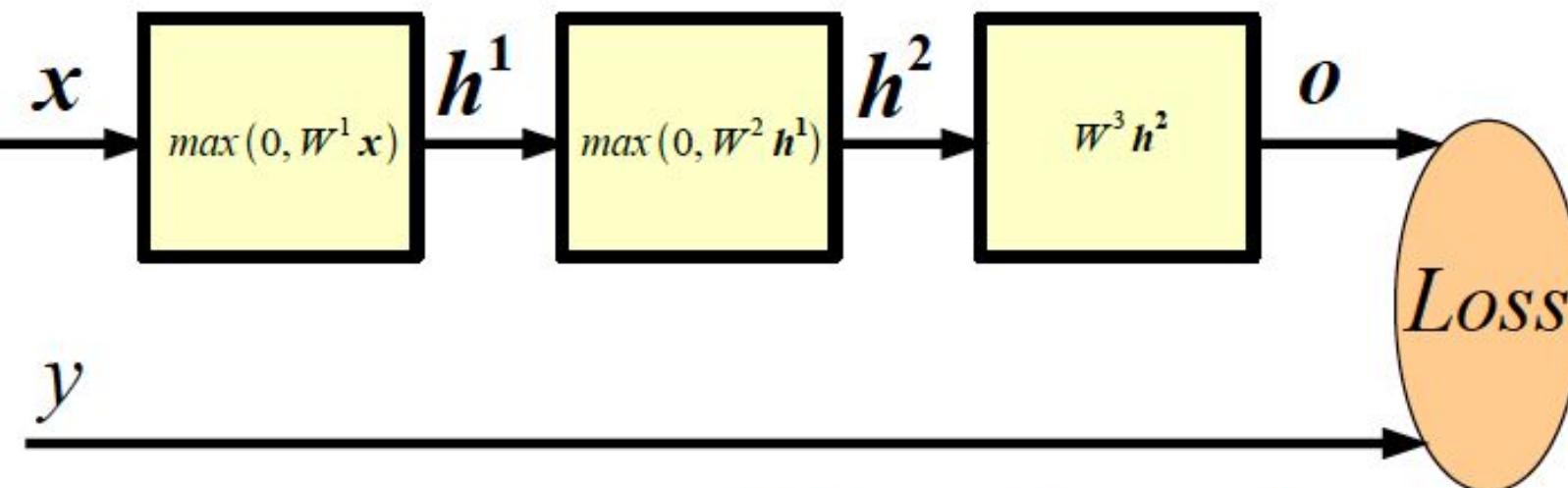
**Question:** How to minimize a complicated function of the parameters?

**Answer:** Chain rule, a.k.a. **Backpropagation**! That is the procedure to compute gradients of the loss w.r.t. parameters in a multi-layer neural network.

19

Rumelhart et al. "Learning internal representations by back-propagating.." Nature 1986

# Key Idea: Wiggle To Decrease Loss



Let's say we want to decrease the loss by adjusting  $W_{i,j}^1$ .

We could consider a very small  $\epsilon = 1e-6$  and compute:

$$L(\mathbf{x}, y; \boldsymbol{\theta})$$

$$L(\mathbf{x}, y; \boldsymbol{\theta} \setminus W_{i,j}^1, W_{i,j}^1 + \epsilon)$$

Then, update:

$$W_{i,j}^1 \leftarrow W_{i,j}^1 + \epsilon \operatorname{sgn}(L(\mathbf{x}, y; \boldsymbol{\theta}) - L(\mathbf{x}, y; \boldsymbol{\theta} \setminus W_{i,j}^1, W_{i,j}^1 + \epsilon))$$

# Derivative w.r.t. Input of Softmax

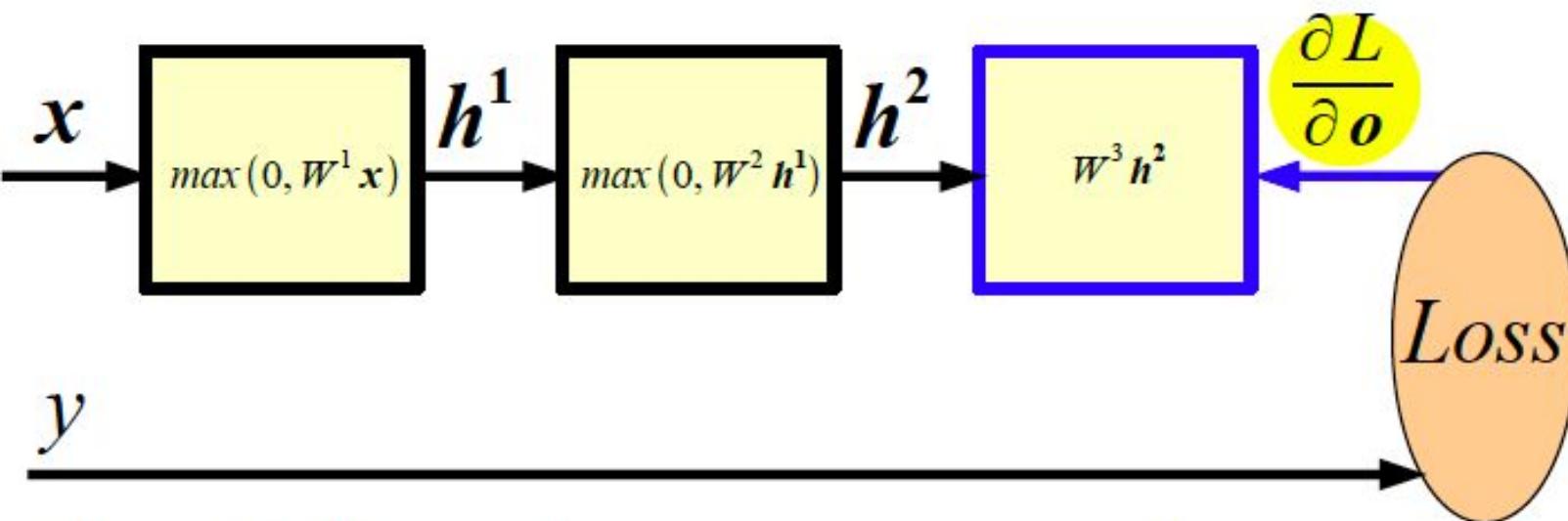
$$p(c_k=1|\mathbf{x}) = \frac{e^{o_k}}{\sum_j e^{o_j}}$$

$$L(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = -\sum_j y_j \log p(c_j|\mathbf{x}) \quad \mathbf{y} = [0^1 0^0 \dots 0^k 0^1 \dots 0^c]$$

By substituting the first formula in the second, and taking the derivative w.r.t.  $\mathbf{o}$  we get:

$$\frac{\partial L}{\partial o} = p(c|\mathbf{x}) - \mathbf{y}$$

# Backward Propagation

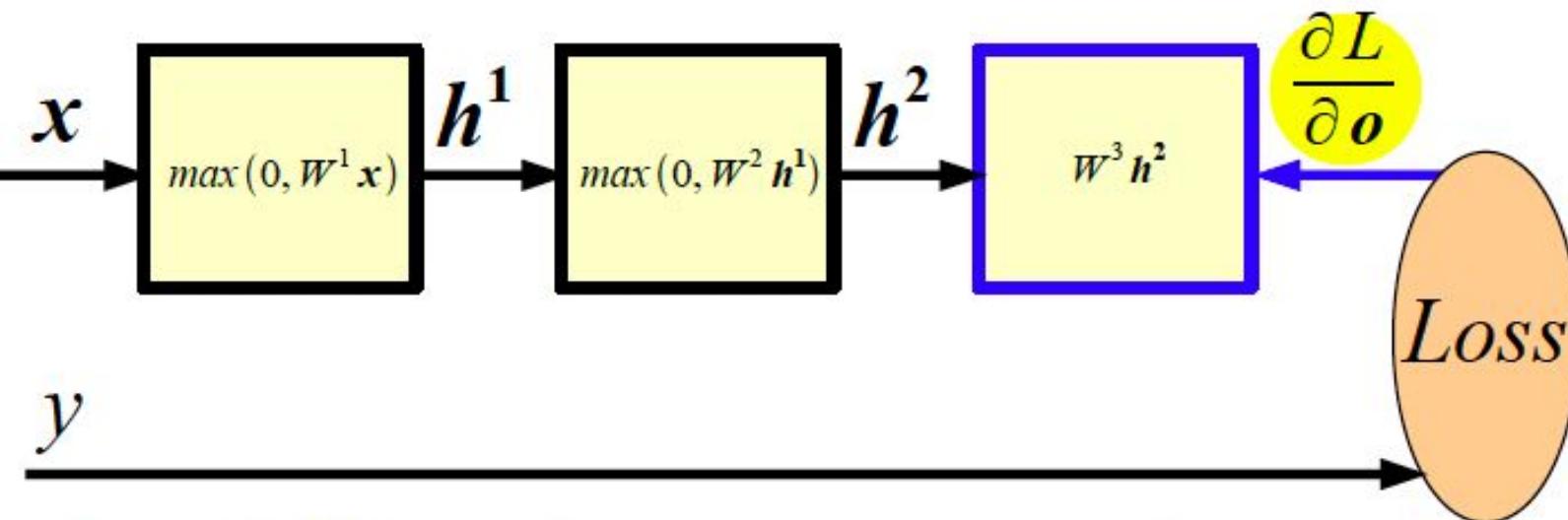


Given  $\frac{\partial L}{\partial \mathbf{o}}$  and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \mathbf{o}} \frac{\partial \mathbf{o}}{\partial W^3}$$

$$\frac{\partial L}{\partial \mathbf{h}^2} = \frac{\partial L}{\partial \mathbf{o}} \frac{\partial \mathbf{o}}{\partial \mathbf{h}^2}$$

# Backward Propagation



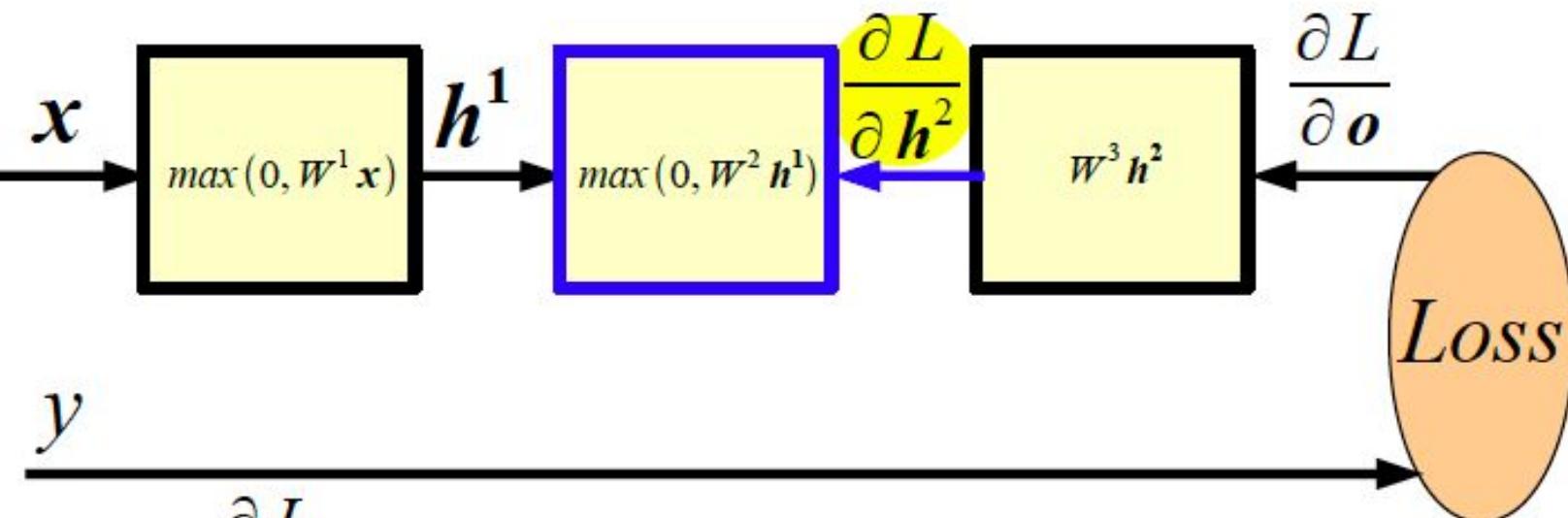
Given  $\frac{\partial L}{\partial \mathbf{o}}$  and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \mathbf{o}} \frac{\partial \mathbf{o}}{\partial W^3}$$

$$\frac{\partial L}{\partial \mathbf{h}^2} = \frac{\partial L}{\partial \mathbf{o}} \frac{\partial \mathbf{o}}{\partial \mathbf{h}^2}$$

$$\frac{\partial L}{\partial W^3} = (p(c|\mathbf{x}) - y) \mathbf{h}^{2T} \quad \frac{\partial L}{\partial \mathbf{h}^2} = W^{3T} (p(c|\mathbf{x}) - y)_{23}$$

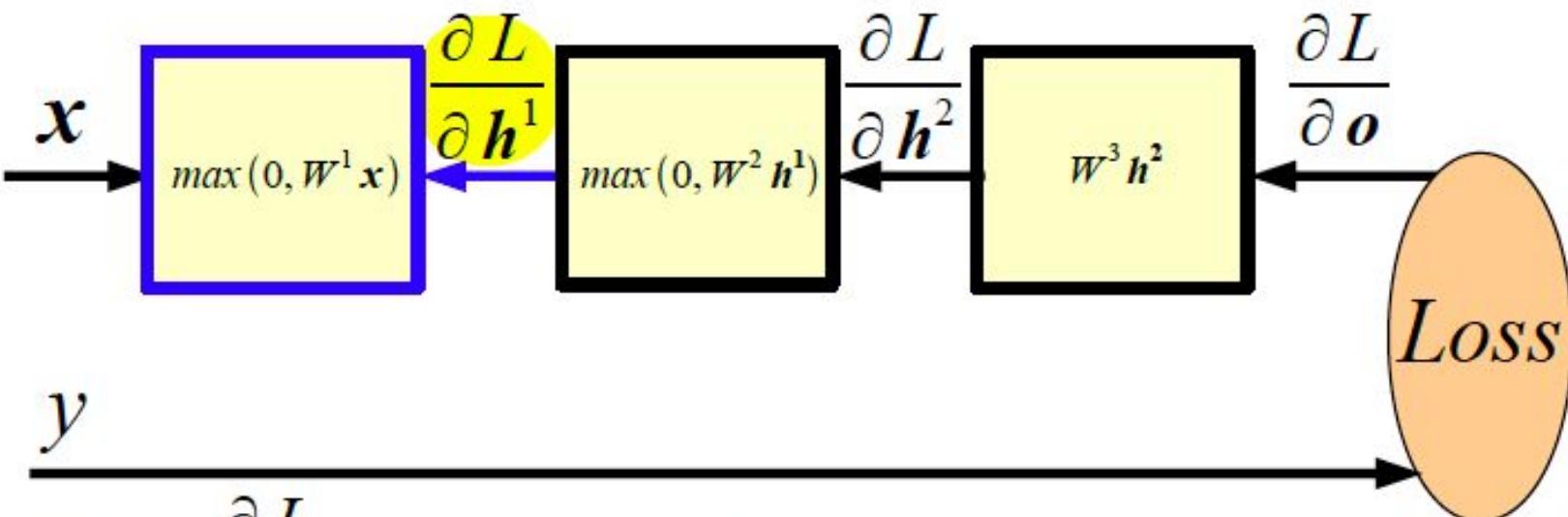
# Backward Propagation



Given  $\frac{\partial L}{\partial h^2}$  we can compute now:

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial W^2} \quad \frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial h^1}$$

# Backward Propagation



Given  $\frac{\partial L}{\partial h^1}$  we can compute now:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial h^1} \frac{\partial h^1}{\partial W^1}$$

# Backward Propagation

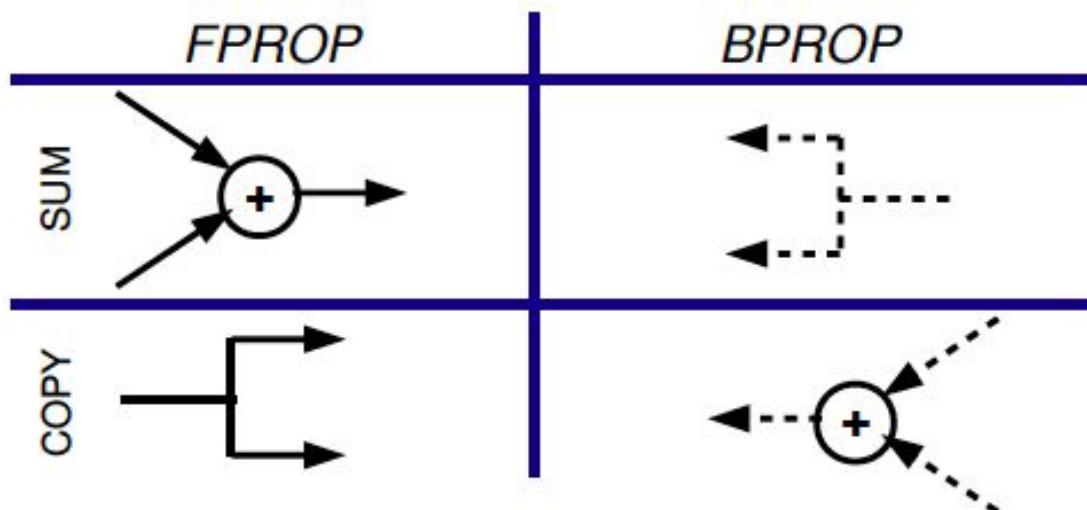
**Question:** Does BPROP work with ReLU layers only?

**Answer:** Nope, any a.e. differentiable transformation works.

**Question:** What's the computational cost of BPROP?

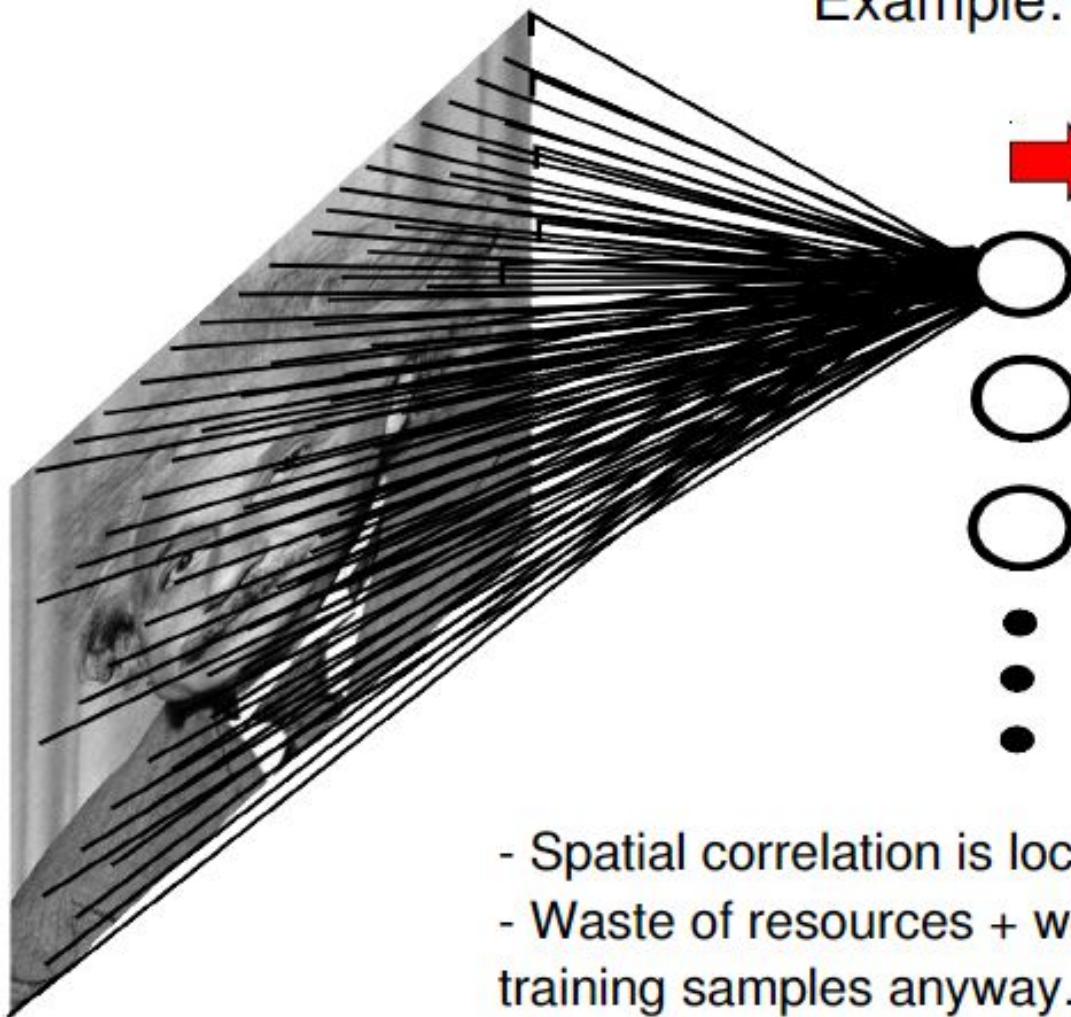
**Answer:** About twice FPROP (need to compute gradients w.r.t. input and parameters at every layer).

**Note:** FPROP and BPROP are dual of each other. E.g.,:

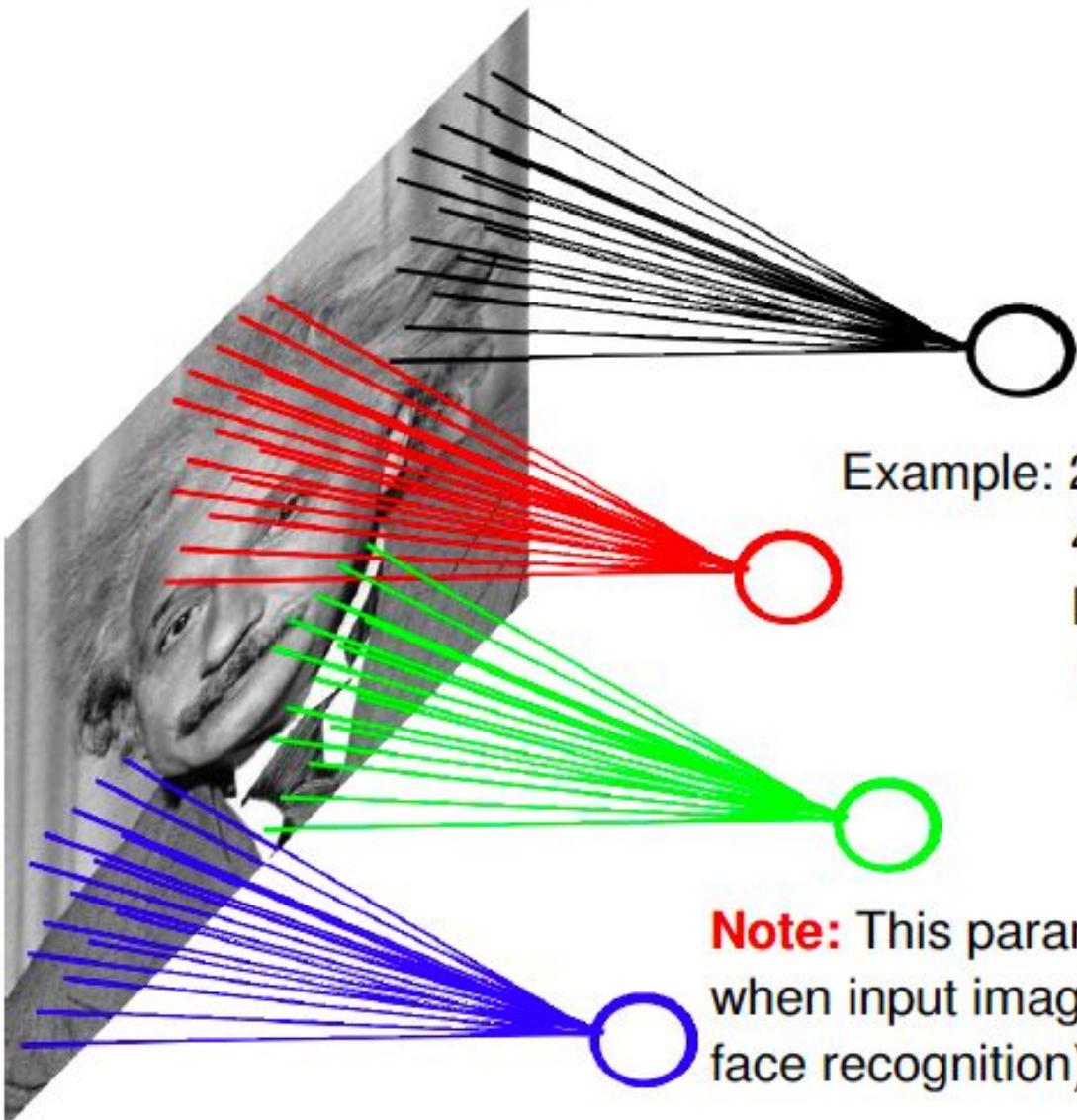


**5 minutes break**

# Fully Connected Layer



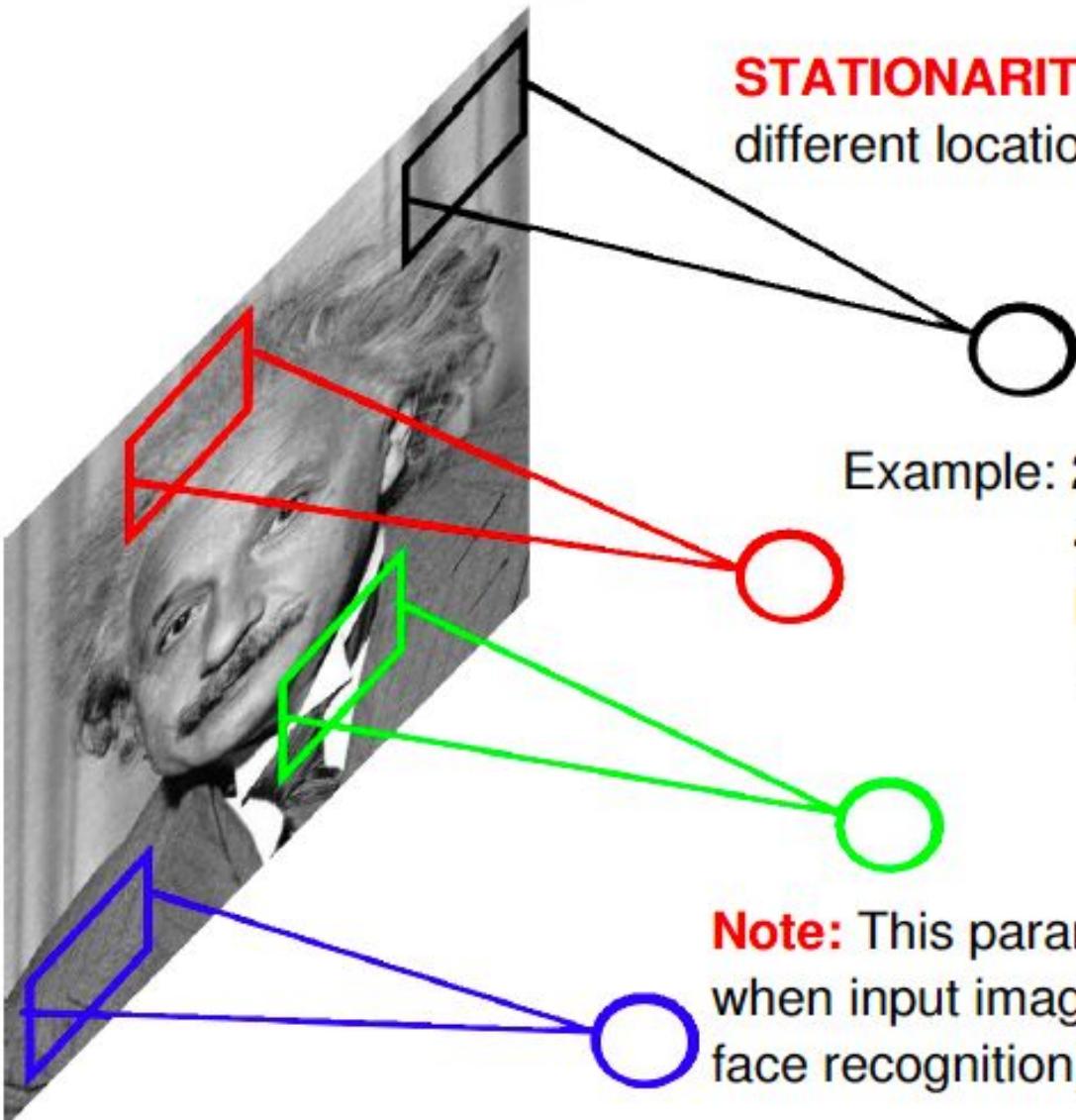
# Locally Connected Layer



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).

# Locally Connected Layer



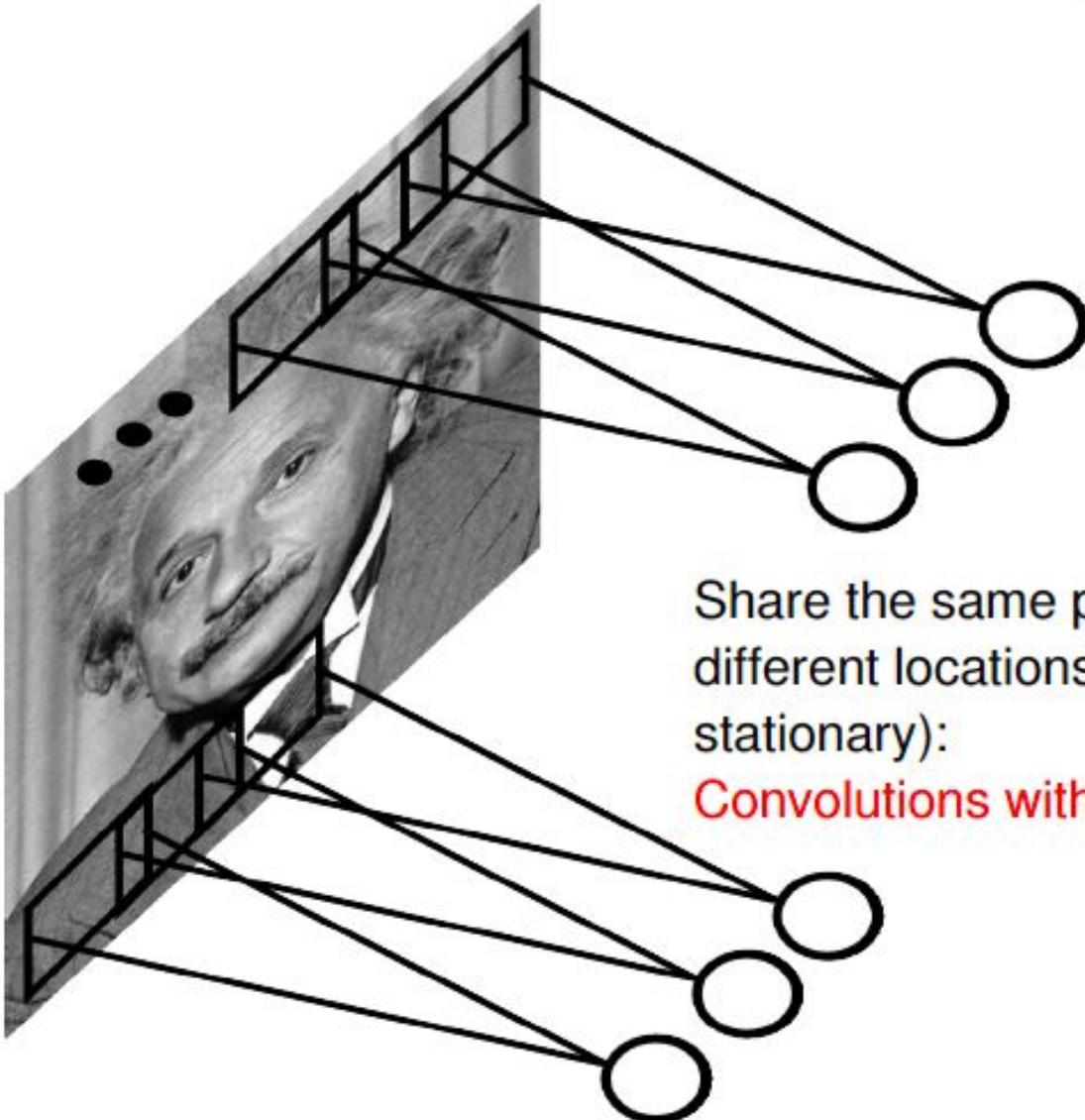
**STATIONARITY?** Statistics is similar at different locations

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).

35

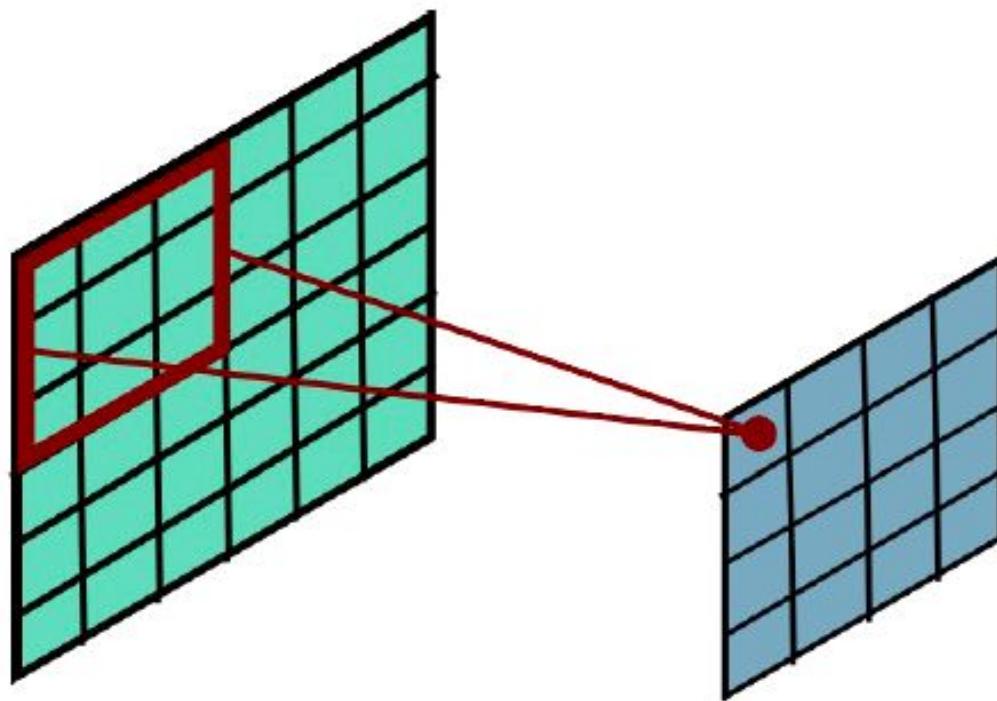
# Convolutional Layer



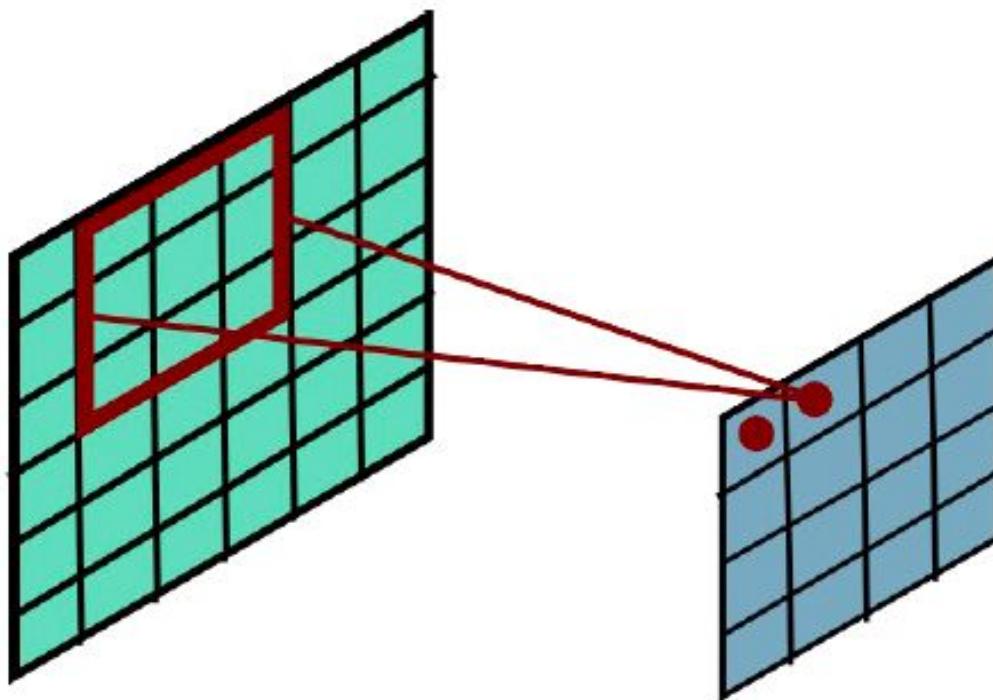
Share the same parameters across  
different locations (assuming input is  
stationary):

**Convolutions with learned kernels**

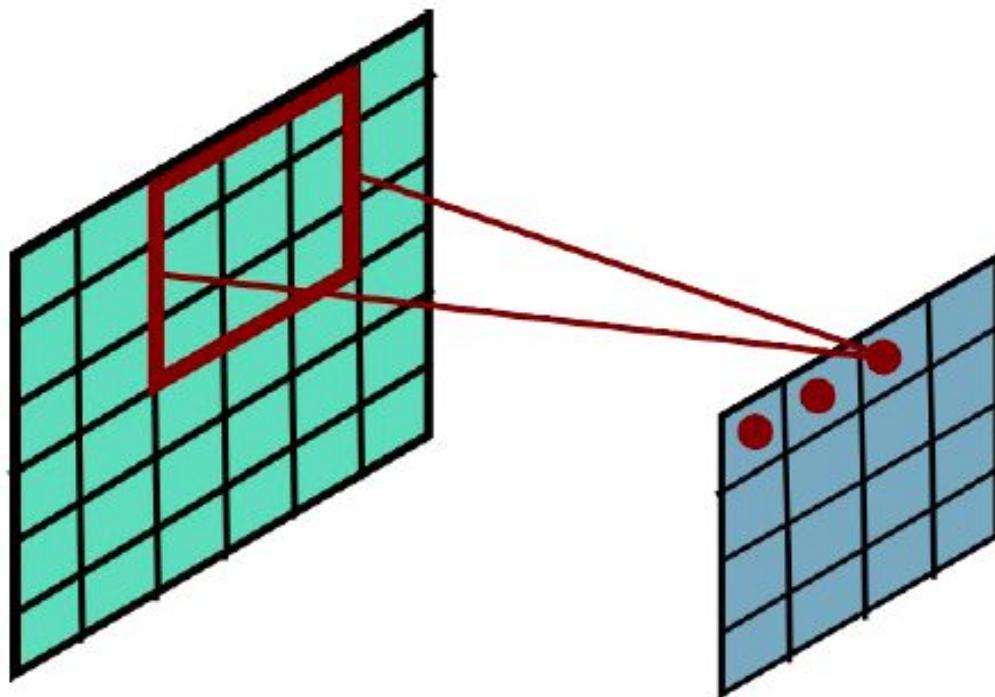
# Convolutional Layer



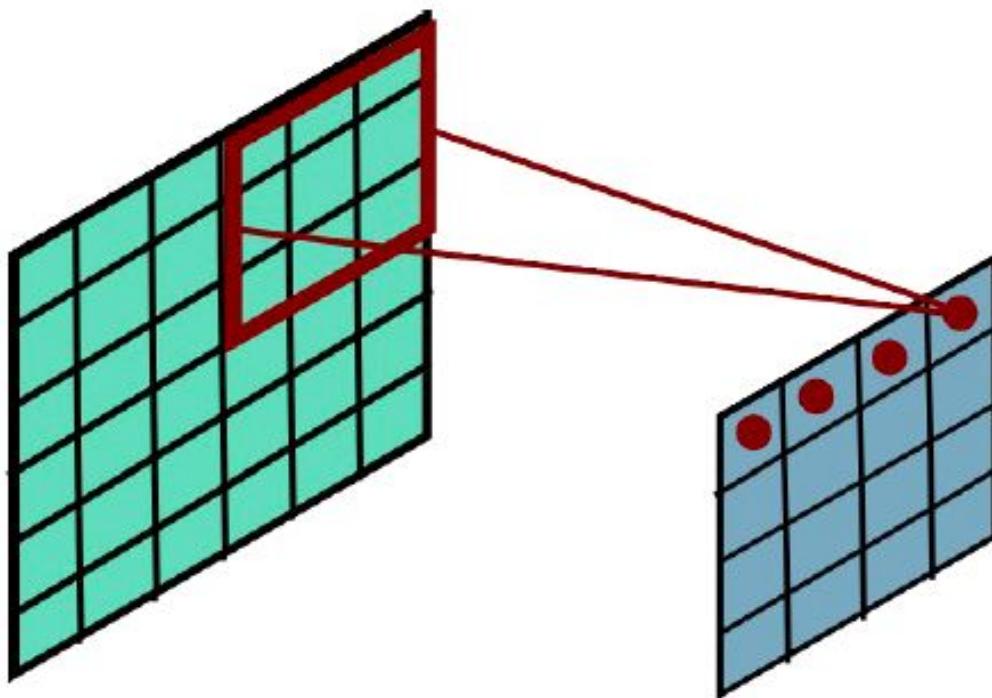
# Convolutional Layer



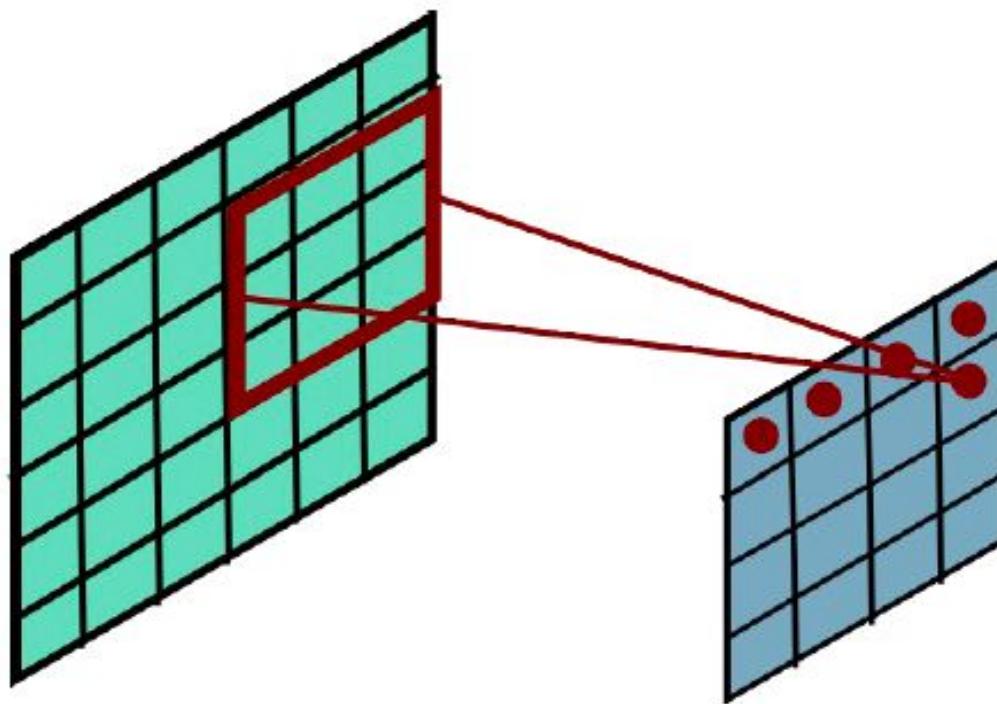
# Convolutional Layer



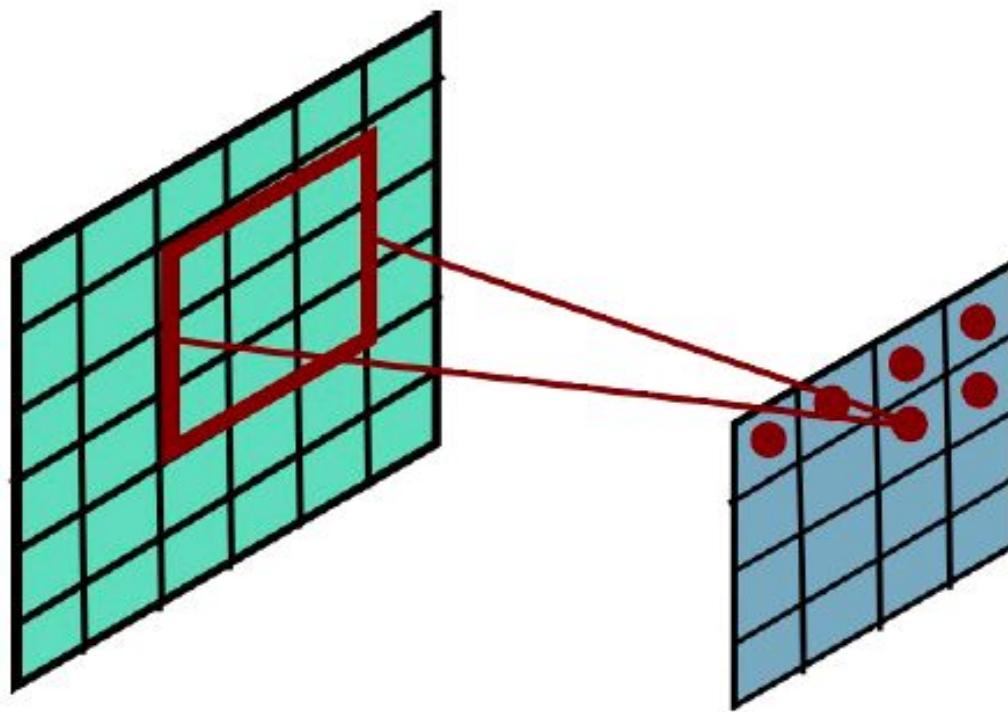
# Convolutional Layer



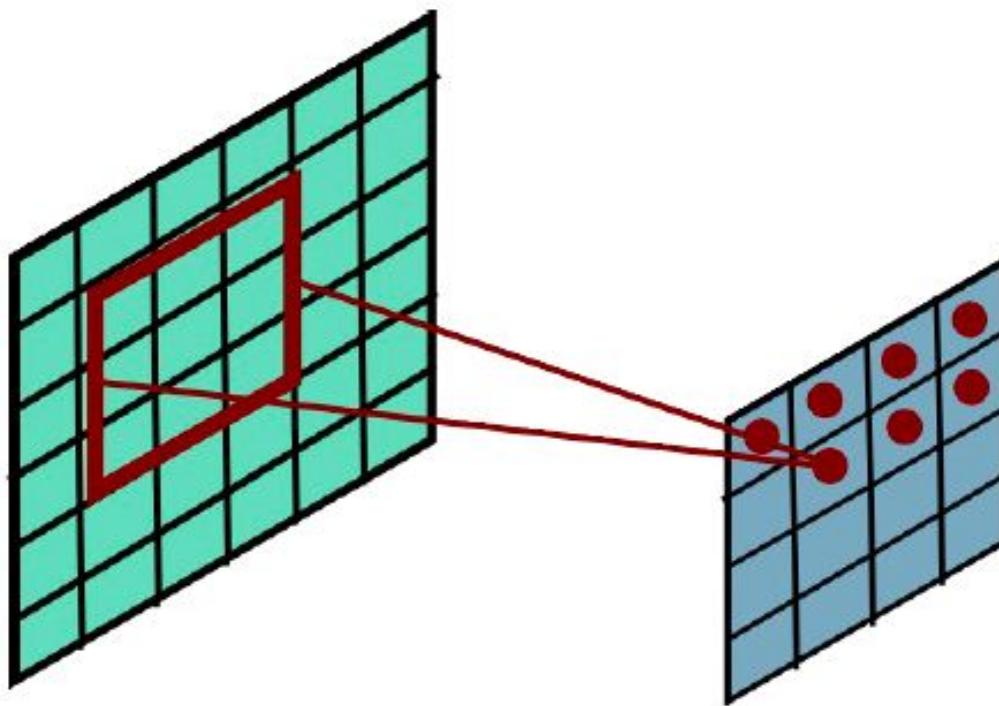
# Convolutional Layer



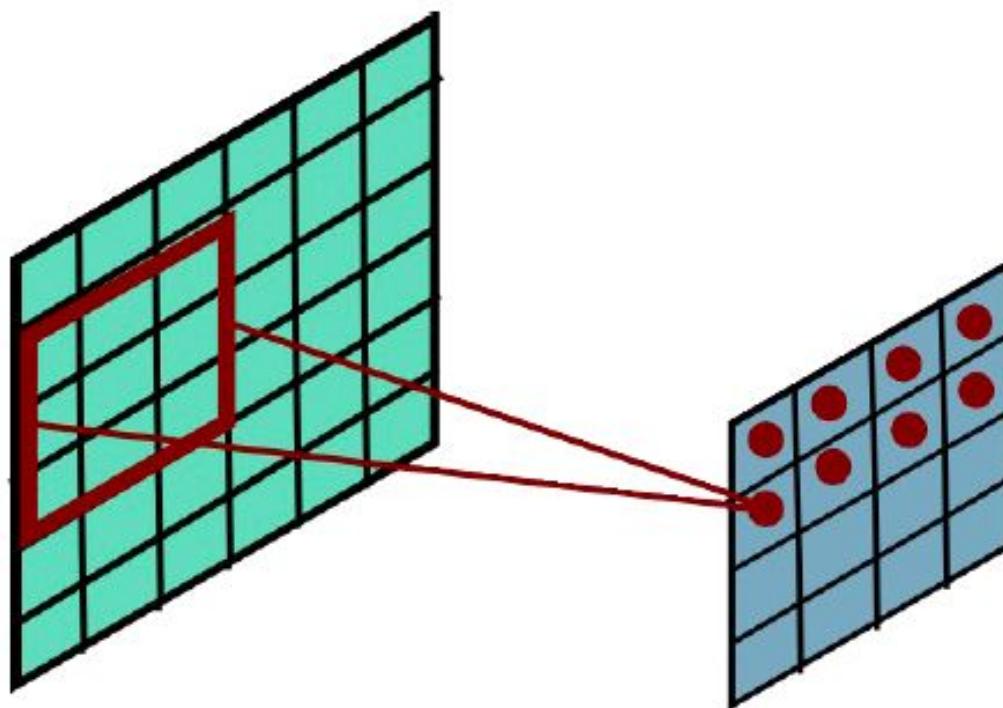
# Convolutional Layer



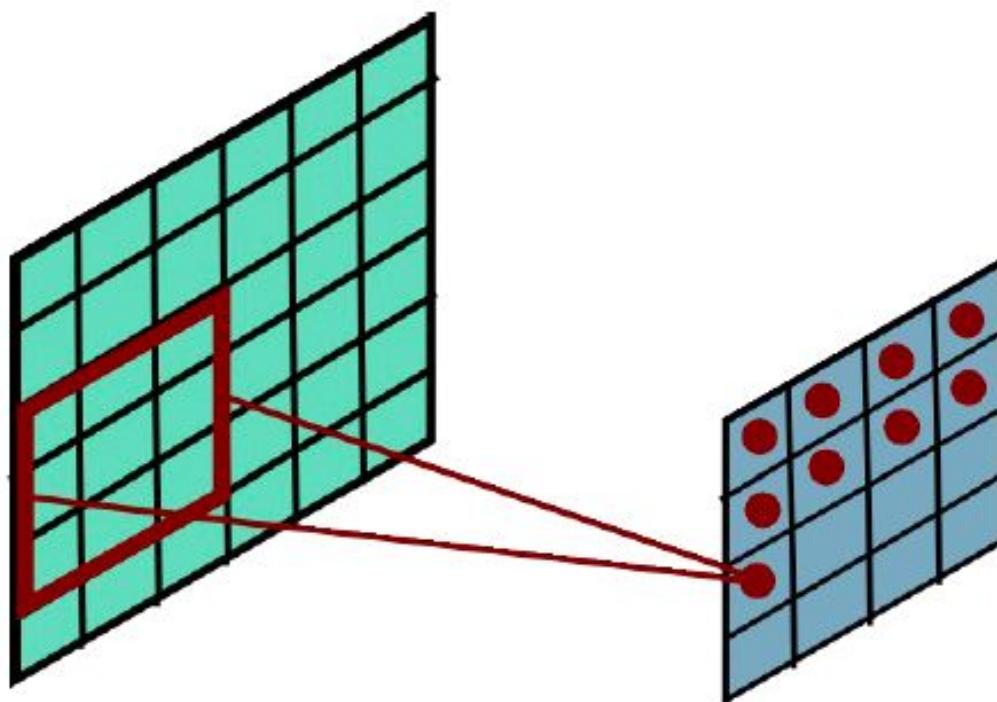
# Convolutional Layer



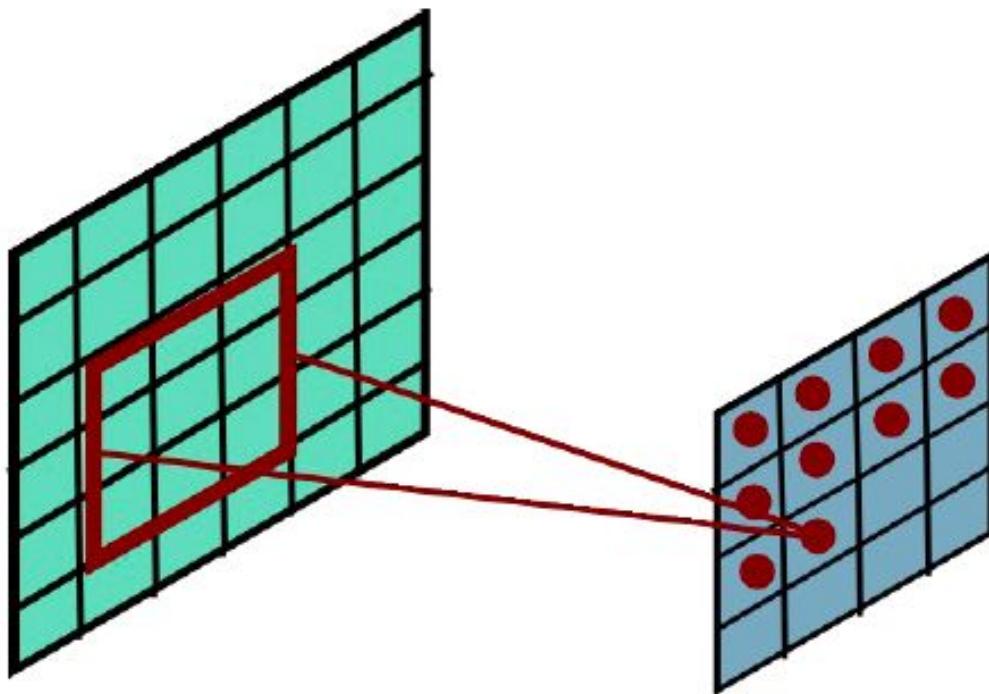
# Convolutional Layer



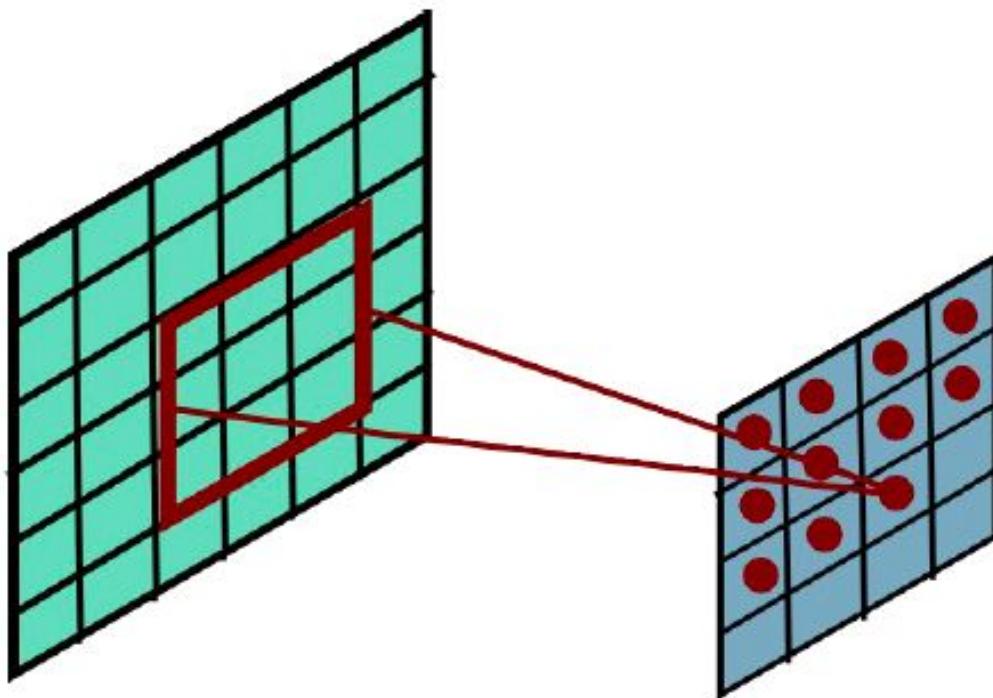
# Convolutional Layer



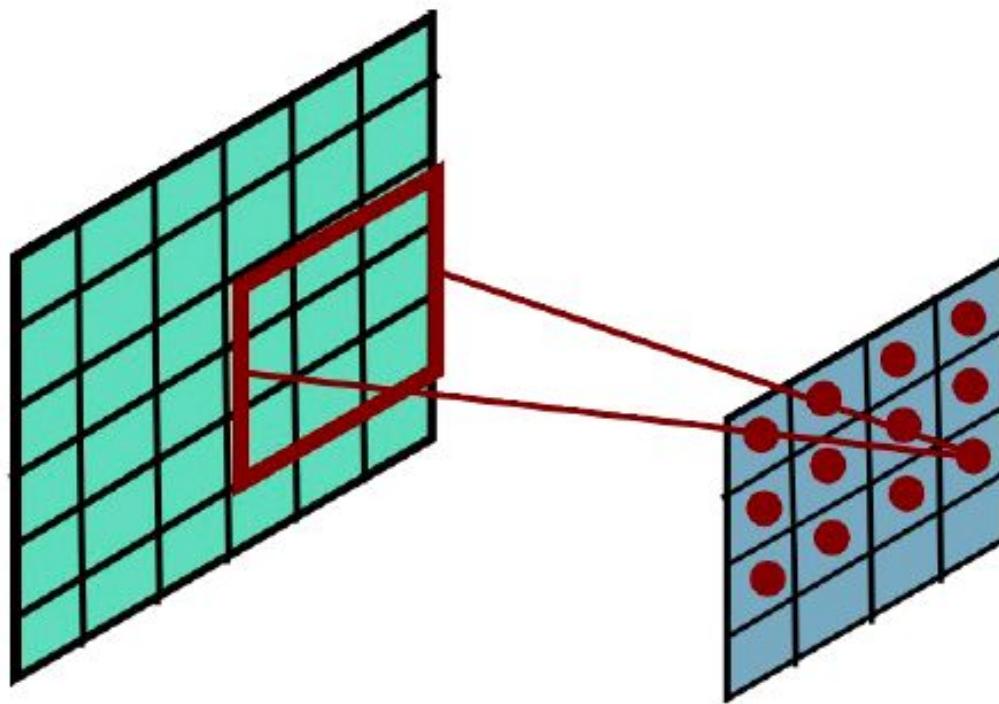
# Convolutional Layer



# Convolutional Layer



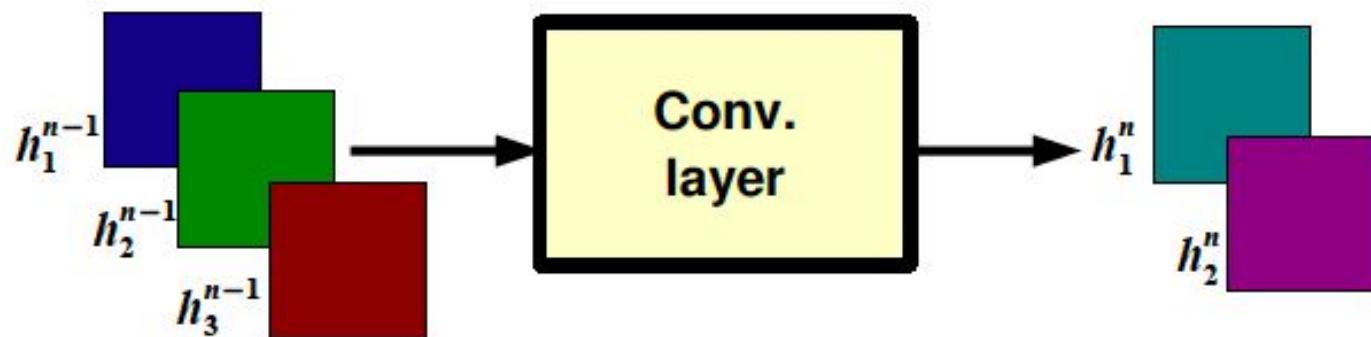
# Convolutional Layer



# Convolutional Layer

$$h_j^n = \max \left( 0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n \right)$$

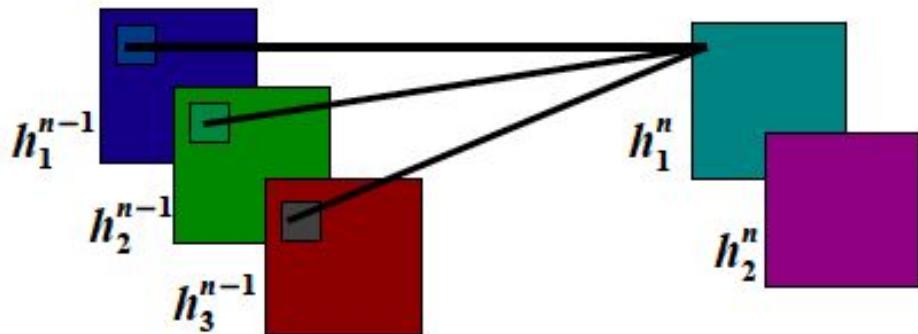
output feature map      input feature map      kernel



# Convolutional Layer

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n)$$

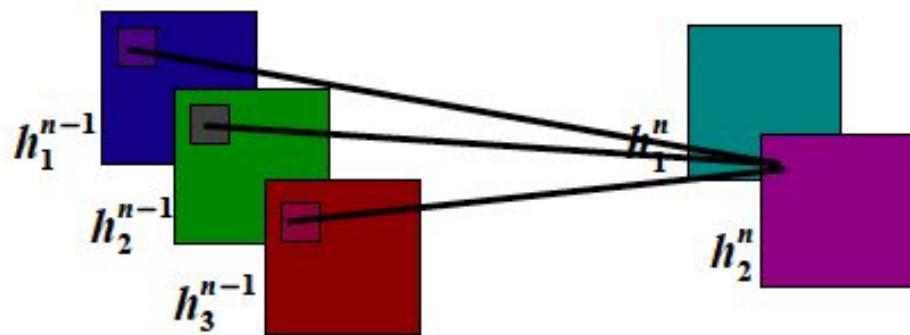
output feature map      input feature map      kernel



# Convolutional Layer

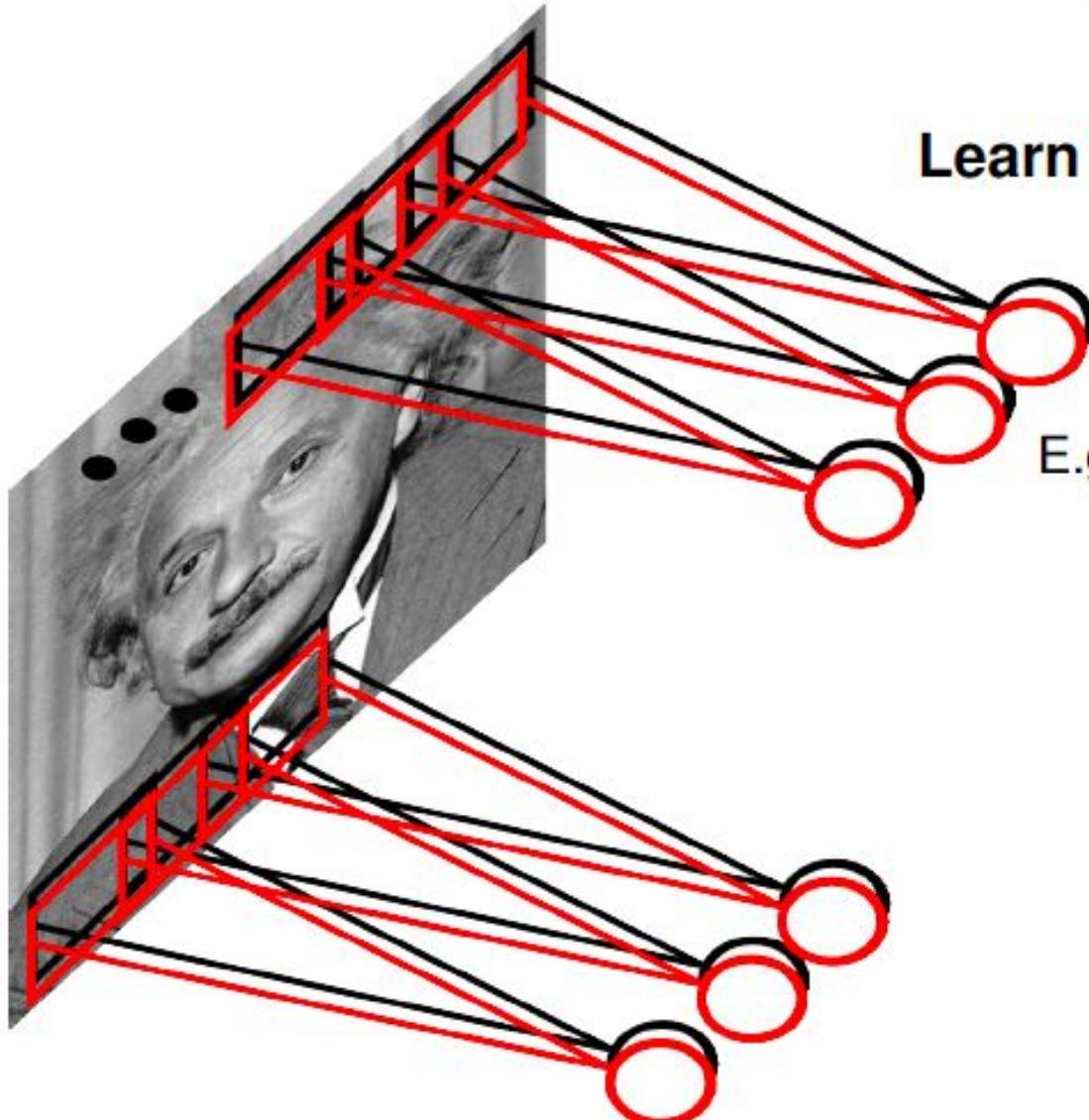
$$h_j^n = \max \left( 0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n \right)$$

output feature map      input feature map      kernel



57

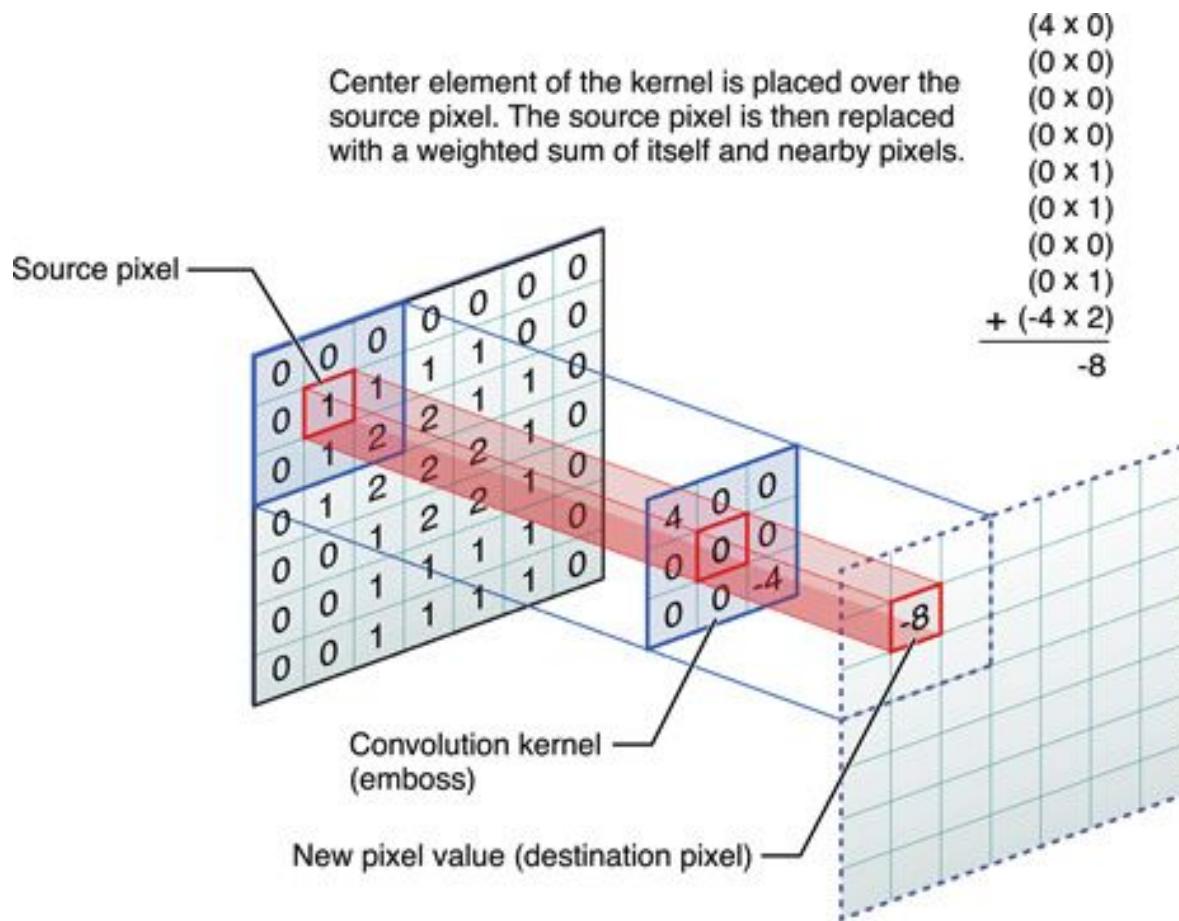
# Convolutional Layer



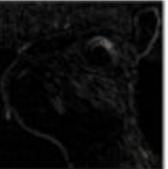
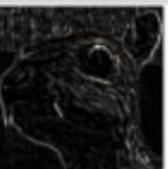
**Learn multiple filters.**

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

# Convolutional Layer



# Convolutional Layer

Operation	Filter	Convolved Image		
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ 
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$		Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ 
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$		Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ 
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$			

# Convolutional Layer

**Question:** What is the size of the output? What's the computational cost?

**Answer:** It is proportional to the number of filters and depends on the stride. If kernels have size  $K \times K$ , input has size  $D \times D$ , stride is 1, and there are  $M$  input feature maps and  $N$  output feature maps then:

- the input has size  $M @ D \times D$
- the output has size  $N @ (D - K + 1) \times (D - K + 1)$
- the kernels have  $M \times N \times K \times K$  coefficients (which have to be learned)
- cost:  $M \cdot K^2 \cdot N \cdot (D - K + 1)^2$

**Question:** How many feature maps? What's the size of the filters?

**Answer:** Usually, there are more output feature maps than input feature maps. Convolutional layers can increase the number of hidden units by big factors (and are expensive to compute).  
The size of the filters has to match the size/scale of the patterns we want to detect (task dependent).

# Key Ideas

A standard neural net applied to images:

- scales quadratically with the size of the input
- does not leverage stationarity

Solution:

- connect each hidden unit to a small patch of the input
- share the weight across space

This is called: **convolutional layer**.

A network with convolutional layers is called **convolutional network**.

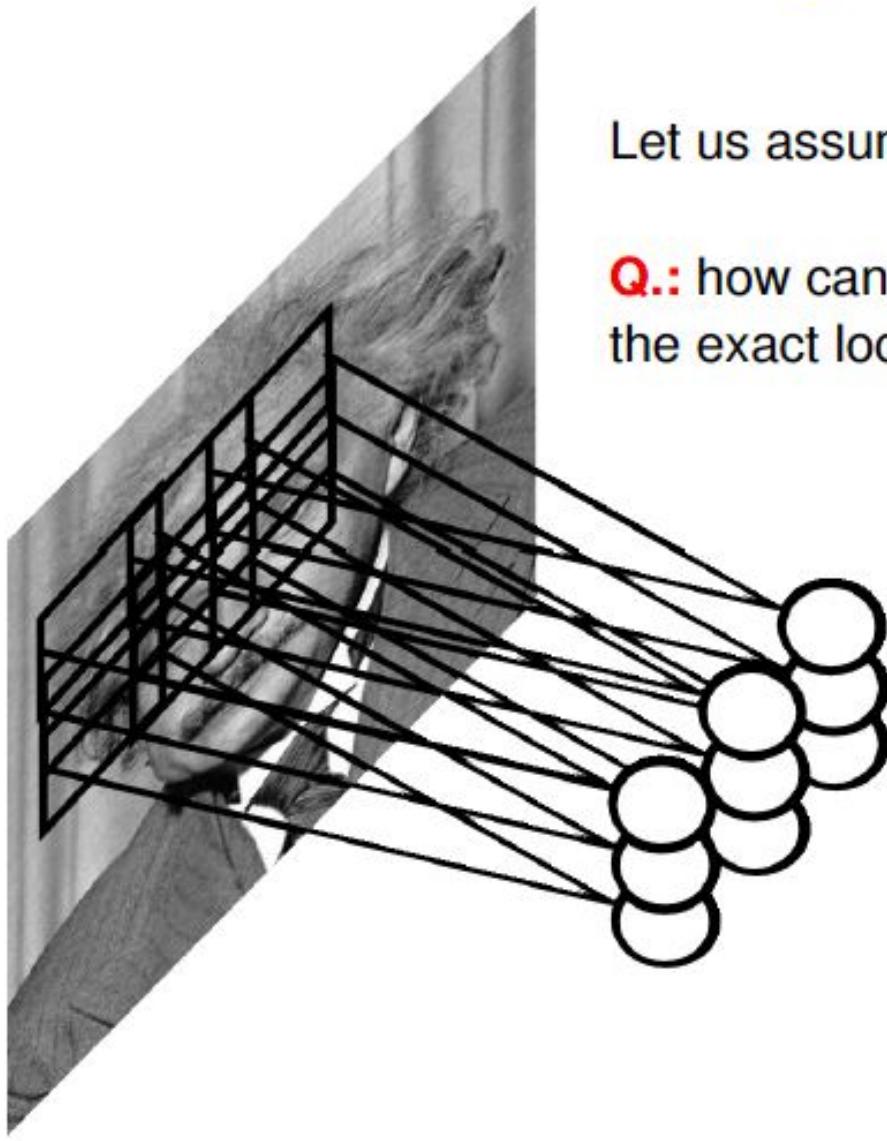
59

LeCun et al. "Gradient-based learning applied to document recognition" IEEE 1998

# Pooling Layer

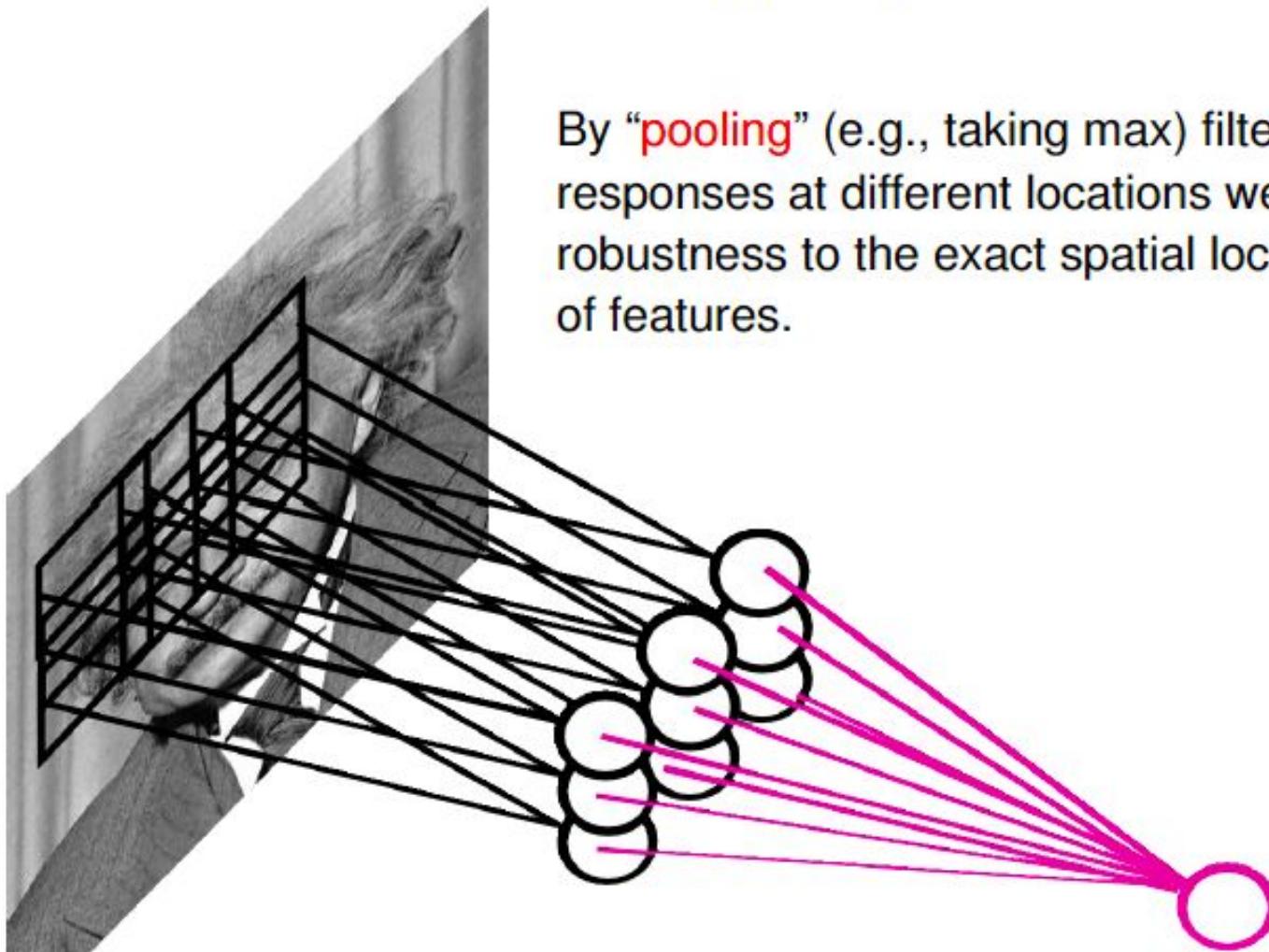
Let us assume filter is an “eye” detector.

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



# Pooling Layer

By “**pooling**” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.



61

F

# 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:

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

L2-pooling:

$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

L2-pooling over features:

$$h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

# Pooling Layer

**Question:** What is the size of the output? What's the computational cost?

**Answer:** The size of the output depends on the stride between the pools. For instance, if pools do not overlap and have size  $K \times K$ , and the input has size  $D \times D$  with  $M$  input feature maps, then:

- output is  $M @ (D/K) \times (D/K)$
- the computational cost is proportional to the size of the input (negligible compared to a convolutional layer)

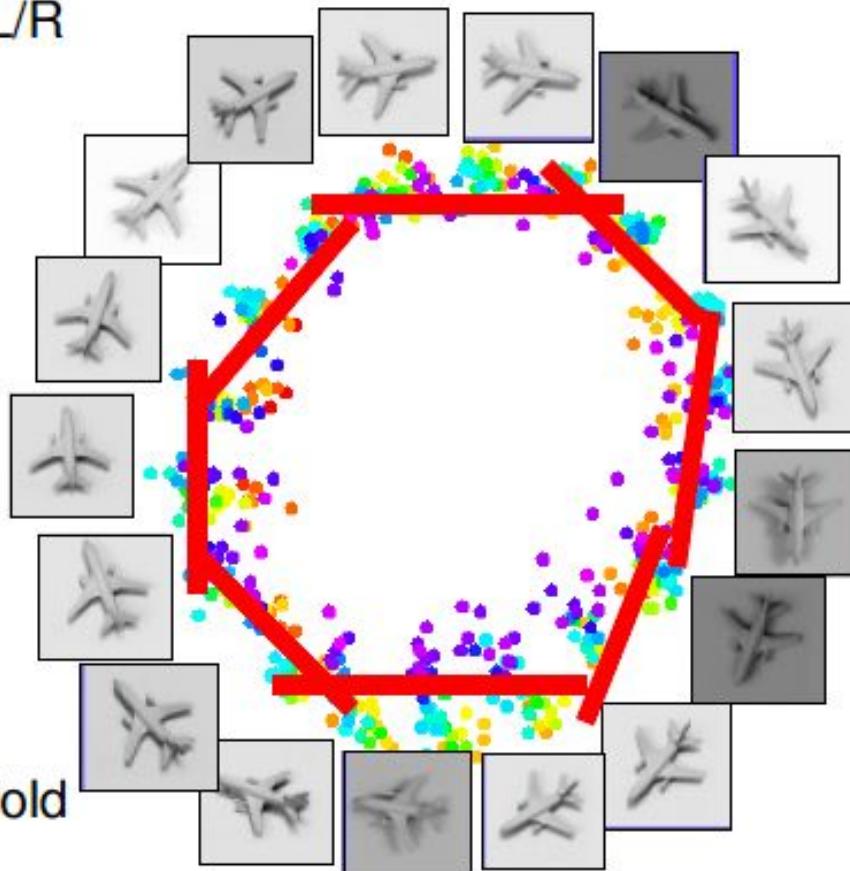
**Question:** How should I set the size of the pools?

**Answer:** It depends on how much “invariant” or robust to distortions we want the representation to be. It is best to pool slowly (via a few stacks of conv-pooling layers).

# Pooling Layer: Interpretation

Task: detect orientation L/R

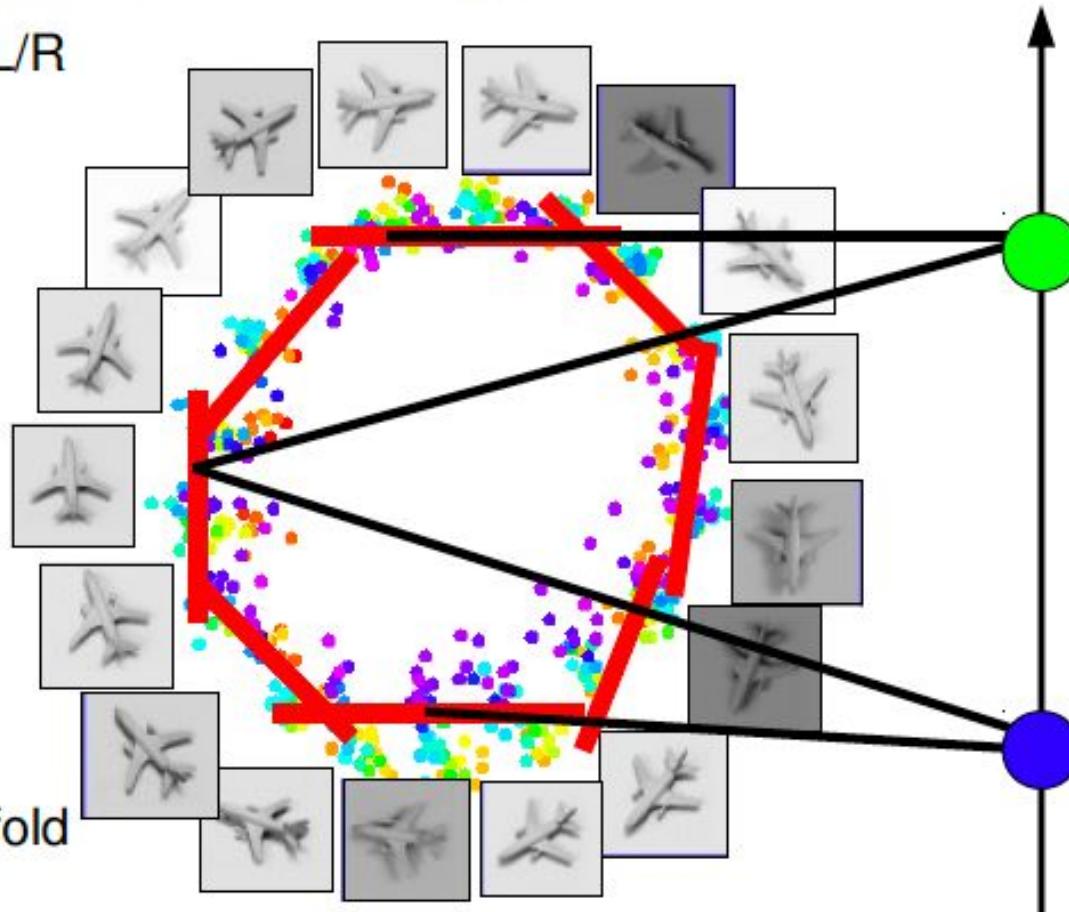
Conv layer:  
linearizes manifold



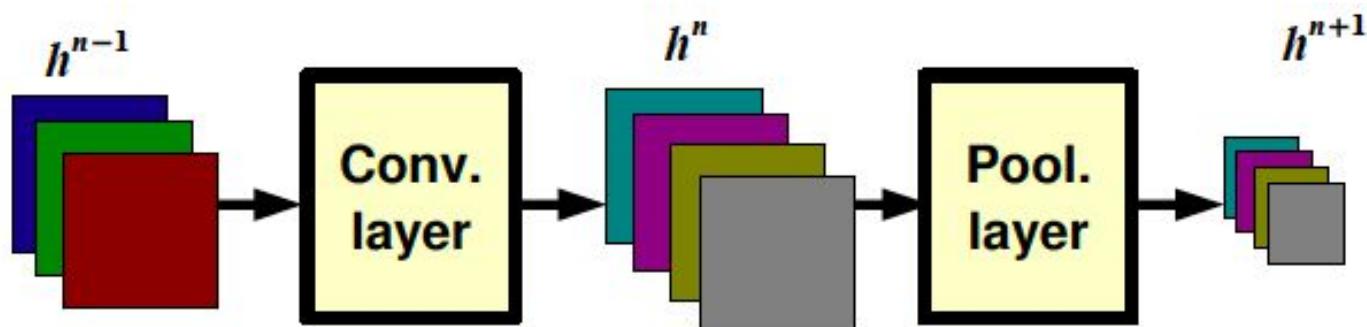
# Pooling Layer: Interpretation

Task: detect orientation L/R

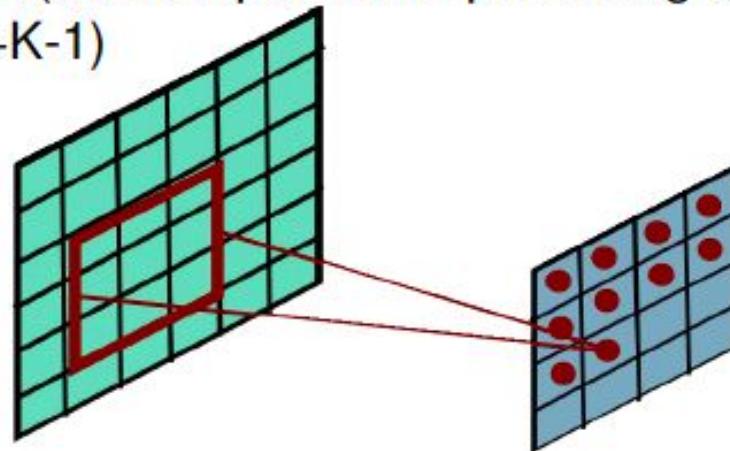
- Conv layer:  
linearizes manifold
- Pooling layer:  
collapses manifold



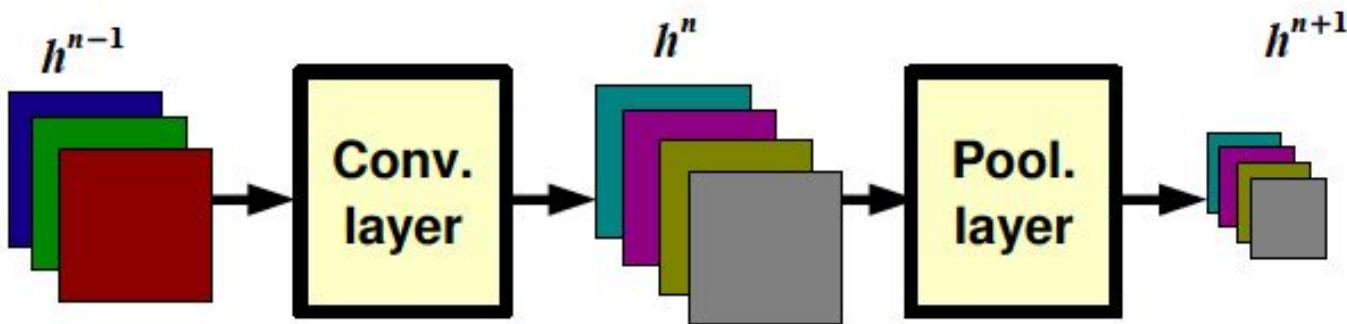
# Pooling Layer: Receptive Field Size



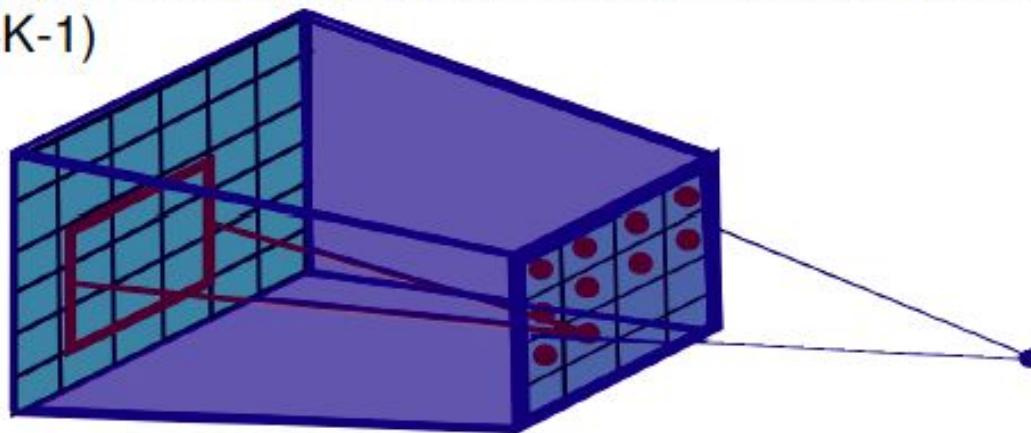
If convolutional filters have size  $K \times K$  and stride 1, and pooling layer has pools of size  $P \times P$ , then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size:  
 $(P+K-1) \times (P+K-1)$



# Pooling Layer: Receptive Field Size



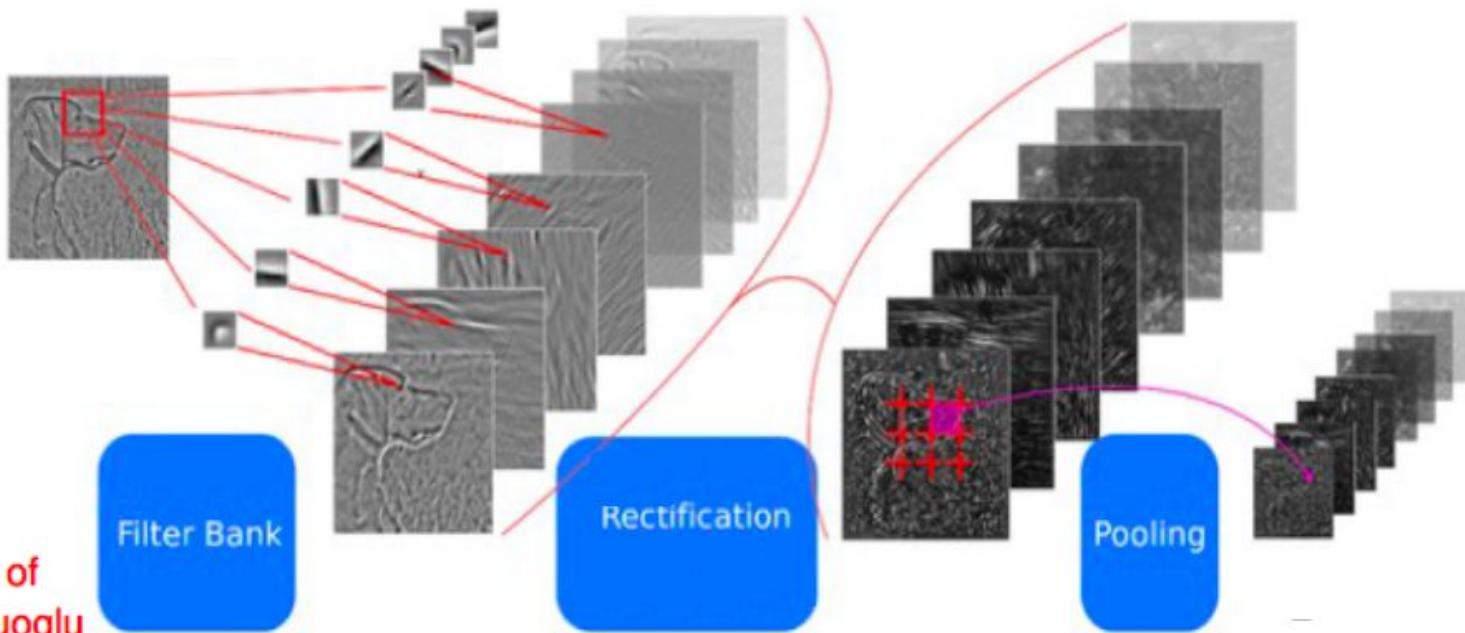
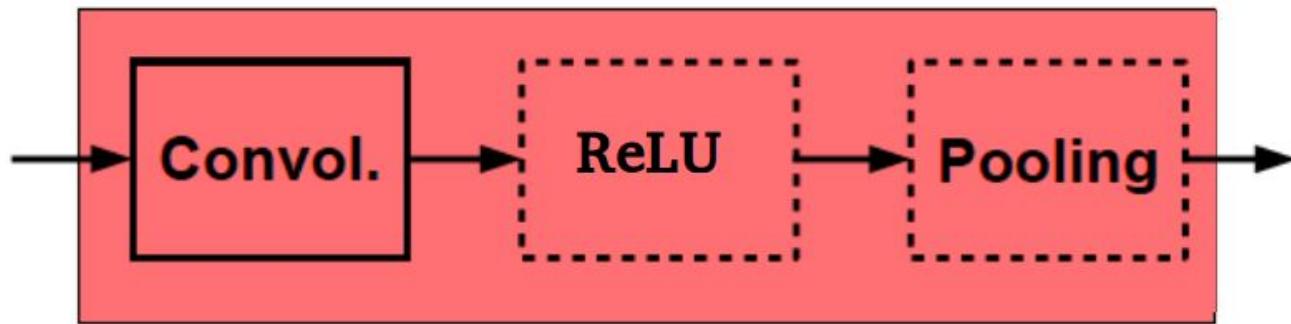
If convolutional filters have size  $K \times K$  and stride 1, and pooling layer has pools of size  $P \times P$ , then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size:  $(P+K-1) \times (P+K-1)$



67

# ConvNets: Typical Stage

One stage (zoom)



courtesy of  
K. Kavukcuoglu

# Convnet Successes

---

- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese [Ciresan et al. 2012]
- Simpler recognition benchmarks
  - CIFAR-10 (9.3% error [Wan et al. 2013])
  - Traffic sign recognition
    - 0.56% error vs 1.16% for humans [Ciresan et al. 2011]
- But until recently, less good at more complex datasets
  - Caltech-101/256 (few training examples)



# ImageNet Challenge 2012



[Deng et al. CVPR 2009]

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- Challenge: 1.2 million training images, 1000 classes

A. Krizhevsky, I. Sutskever, and G. Hinton, [ImageNet Classification with Deep Convolutional Neural Networks](#), NIPS 2012

**That's all!**