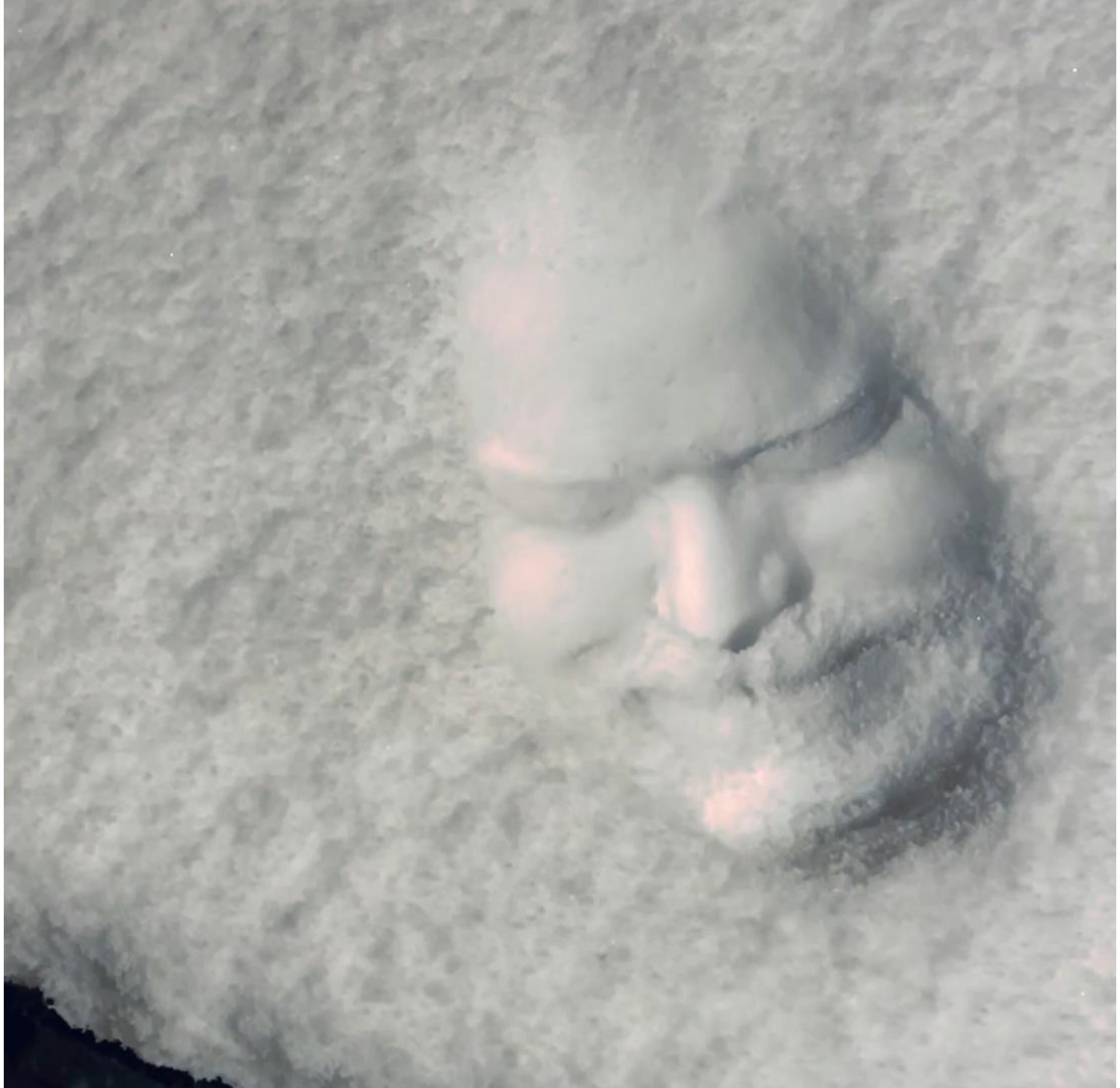
www.grand-illusions.com



source

Deep Learning
Neural Net Basics

Computer Vision

James Hays

Many slides by Marc'Aurelio Ranzato

Outline

- Neural Networks
- *Convolutional* Neural Networks
- Variants
  - Detection
  - Segmentation
  - Siamese Networks
- Visualization of Deep Networks

Supervised Learning

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

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

y^i i-th target label

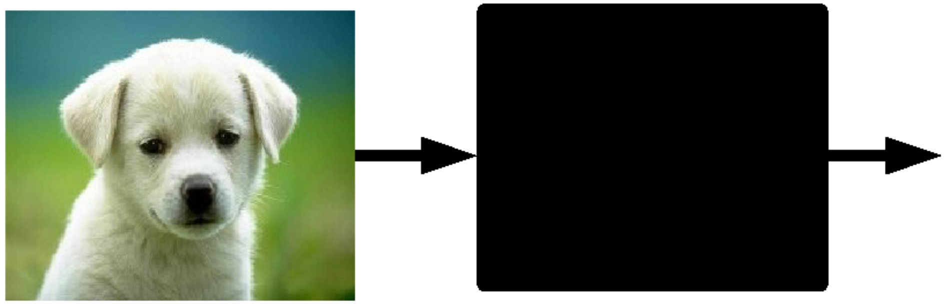
P number of training examples

$\boldsymbol{x}$
y

Goal: predict the target label of unseen inputs.

Supervised Learning: Examples

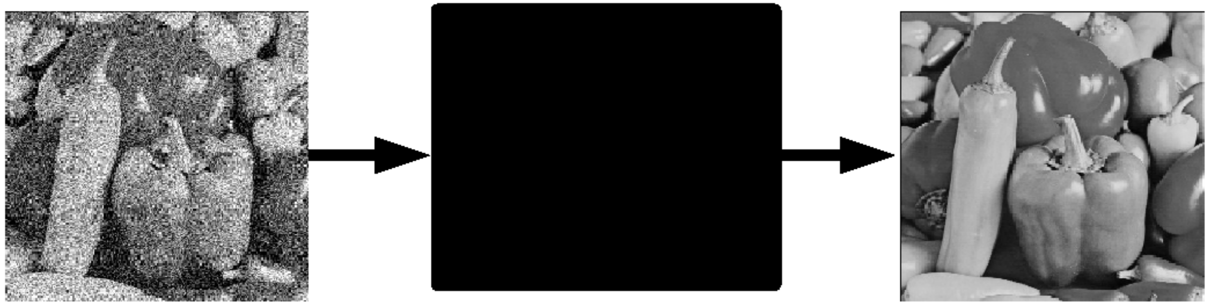
Classification



“dog”

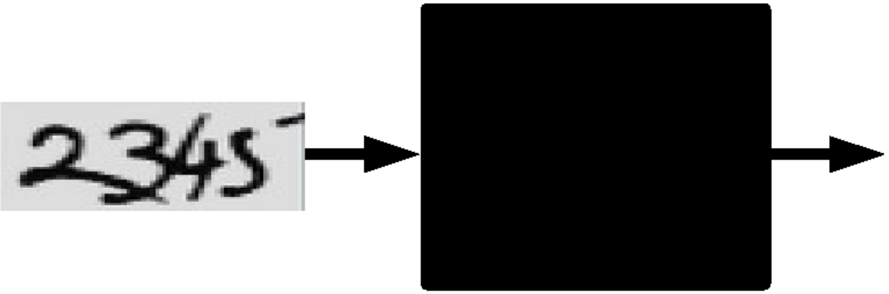
classification

Denoising



regression

OCR

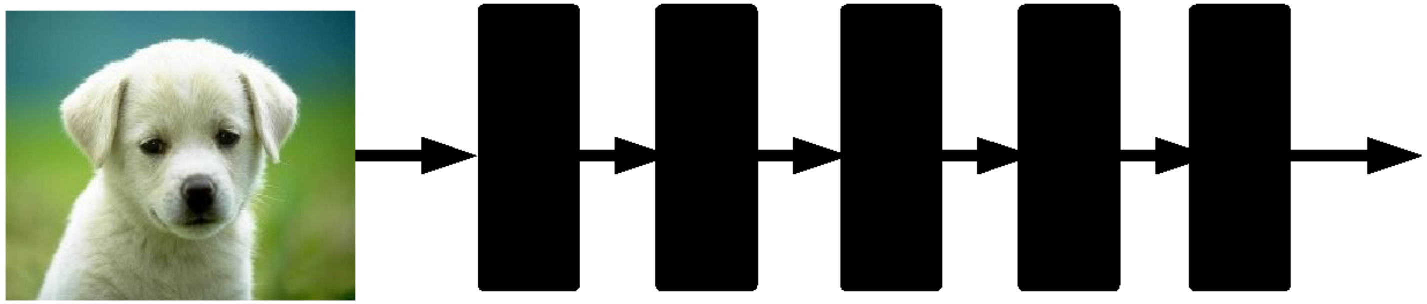


“2 3 4 5”

structured prediction

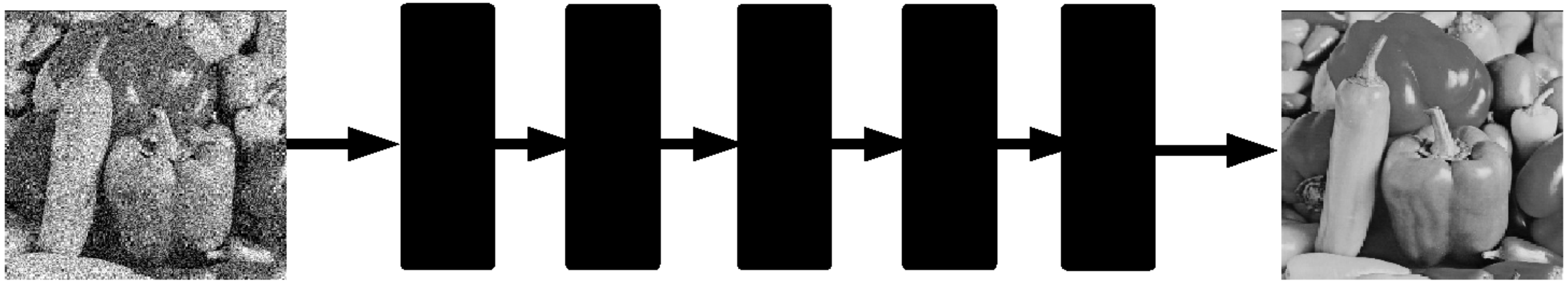
Supervised Deep Learning

Classification

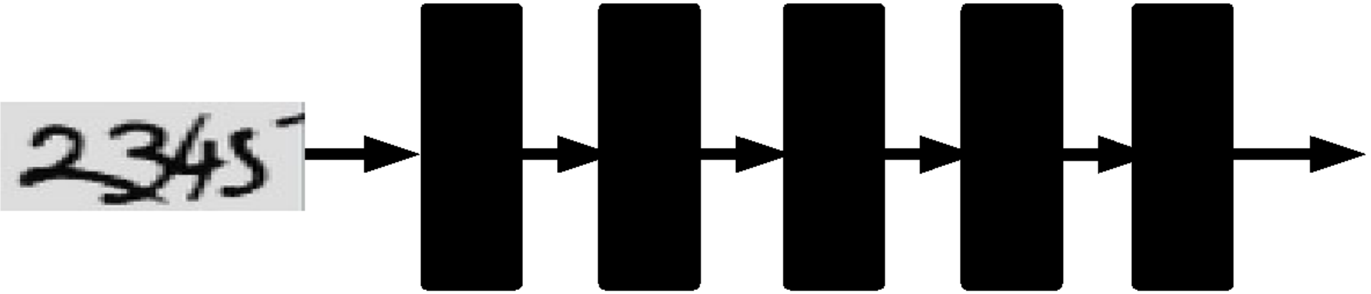


“dog”

Denoising

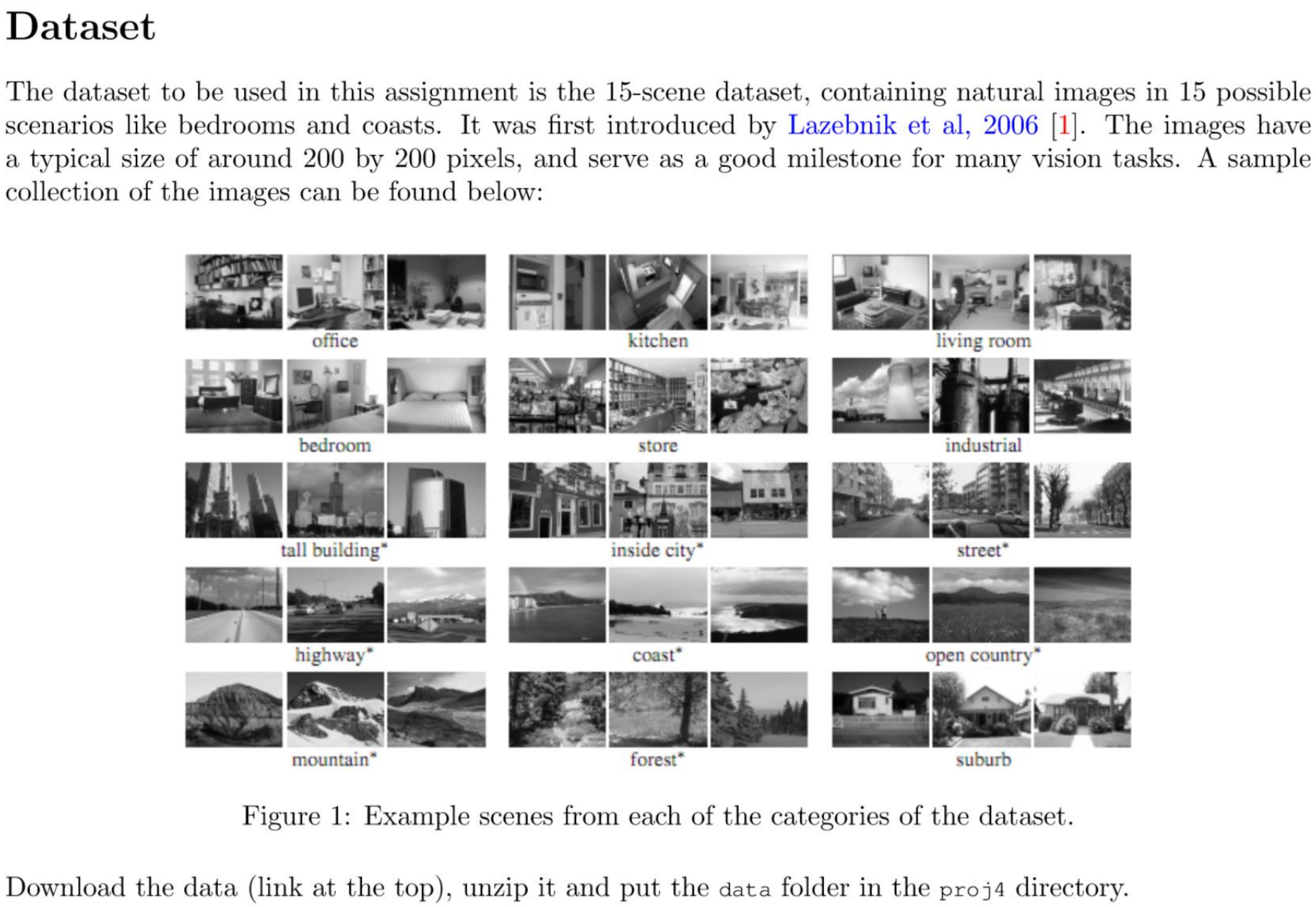


OCR



“2 3 4 5”

Project 4: Scene Classification with Deep Nets



Dataset

The dataset to be used in this assignment is the 15-scene dataset, containing natural images in 15 possible scenarios like bedrooms and coasts. It was first introduced by Lazebnik et al, 2006 [1]. The images have a typical size of around 200 by 200 pixels, and serve as a good milestone for many vision tasks. A sample collection of the images can be found below:

Figure 1: Example scenes from each of the categories of the dataset.

Download the data (link at the top), unzip it and put the `data` folder in the `proj4` directory.

1 Part 1: SimpleNet

Introduction

In this project, scene recognition with deep learning, we are going to train a simple convolutional neural net from scratch. We'll be starting with some modification to the dataloader used in this project to include a few extra pre-processing steps. Subsequently, you will define your own model and optimization function. A trainer class will be provided to you, and you will be able to test out the performance of your model with this complete pipeline of classification problem.

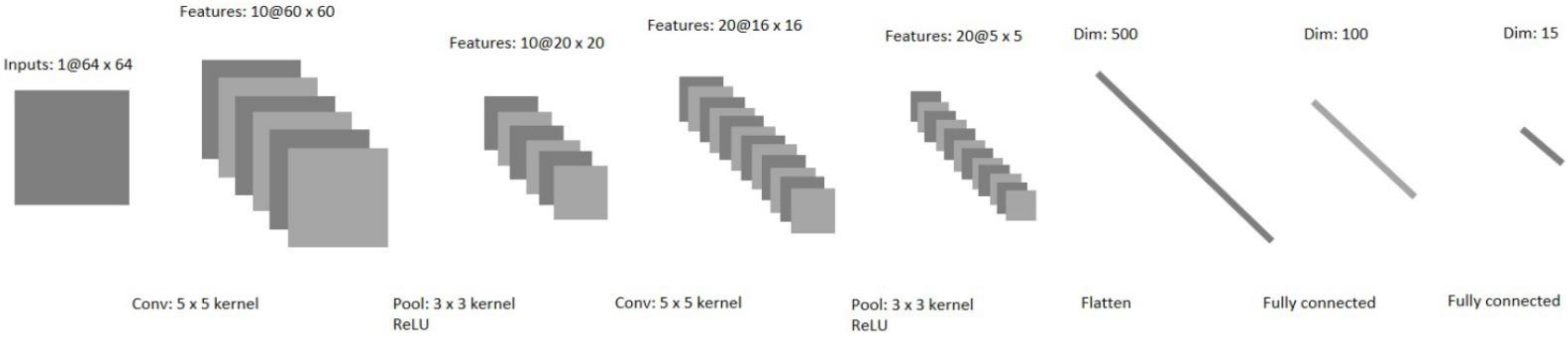


Figure 2: The base SimpleNet architecture for Part 1.

Outline

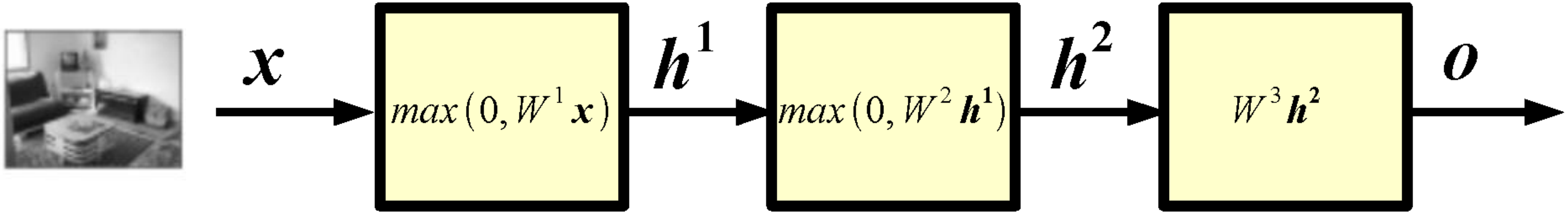
- **Neural Networks**
- *Convolutional* Neural Networks
- Variants
  - Detection
  - Segmentation
  - Siamese Networks
- Visualization of Deep Networks

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)

Neural Networks: example



x input

h^1 1-st layer hidden units

h^2 2-nd layer hidden units

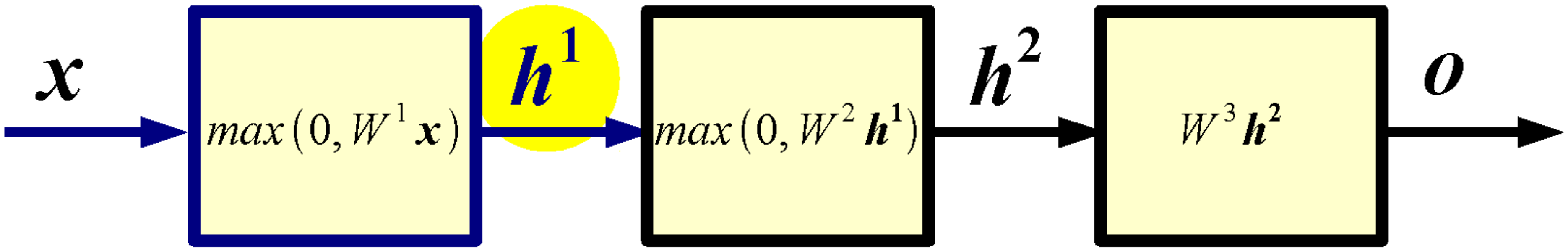
o output

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



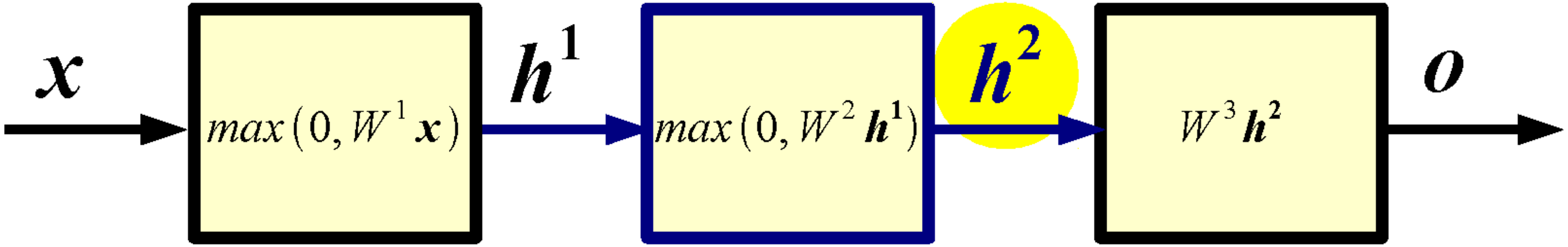
$\boldsymbol{x} \in R^D \quad W^1 \in R^{N_1 \times D} \quad \boldsymbol{b}^1 \in R^{N_1} \quad \boldsymbol{h}^1 \in R^{N_1}$

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

W^1 1-st layer weight matrix or weights
$\boldsymbol{b}^1$ 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



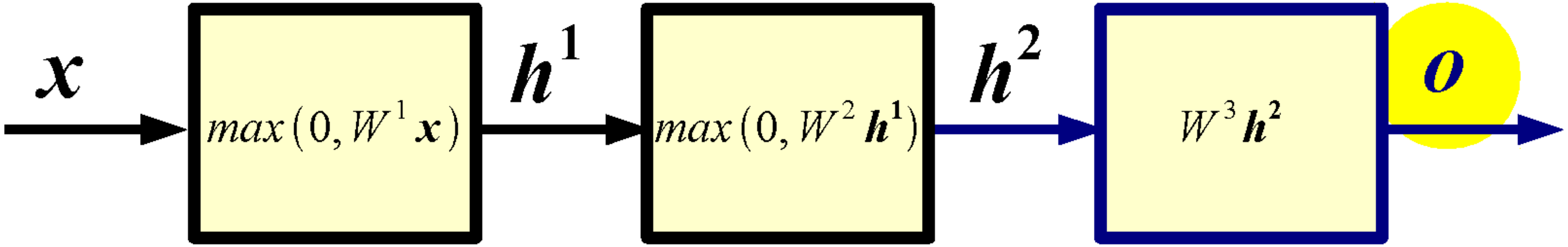
$\boldsymbol{h}^1 \in R^{N_1} \quad W^2 \in R^{N_2 \times N_1} \quad \boldsymbol{b}^2 \in R^{N_2} \qquad \boldsymbol{h}^2 \in R^{N_2}$

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

W^2 2-nd layer weight matrix or weights

$\boldsymbol{b}^2$ 2-nd layer biases

Forward Propagation



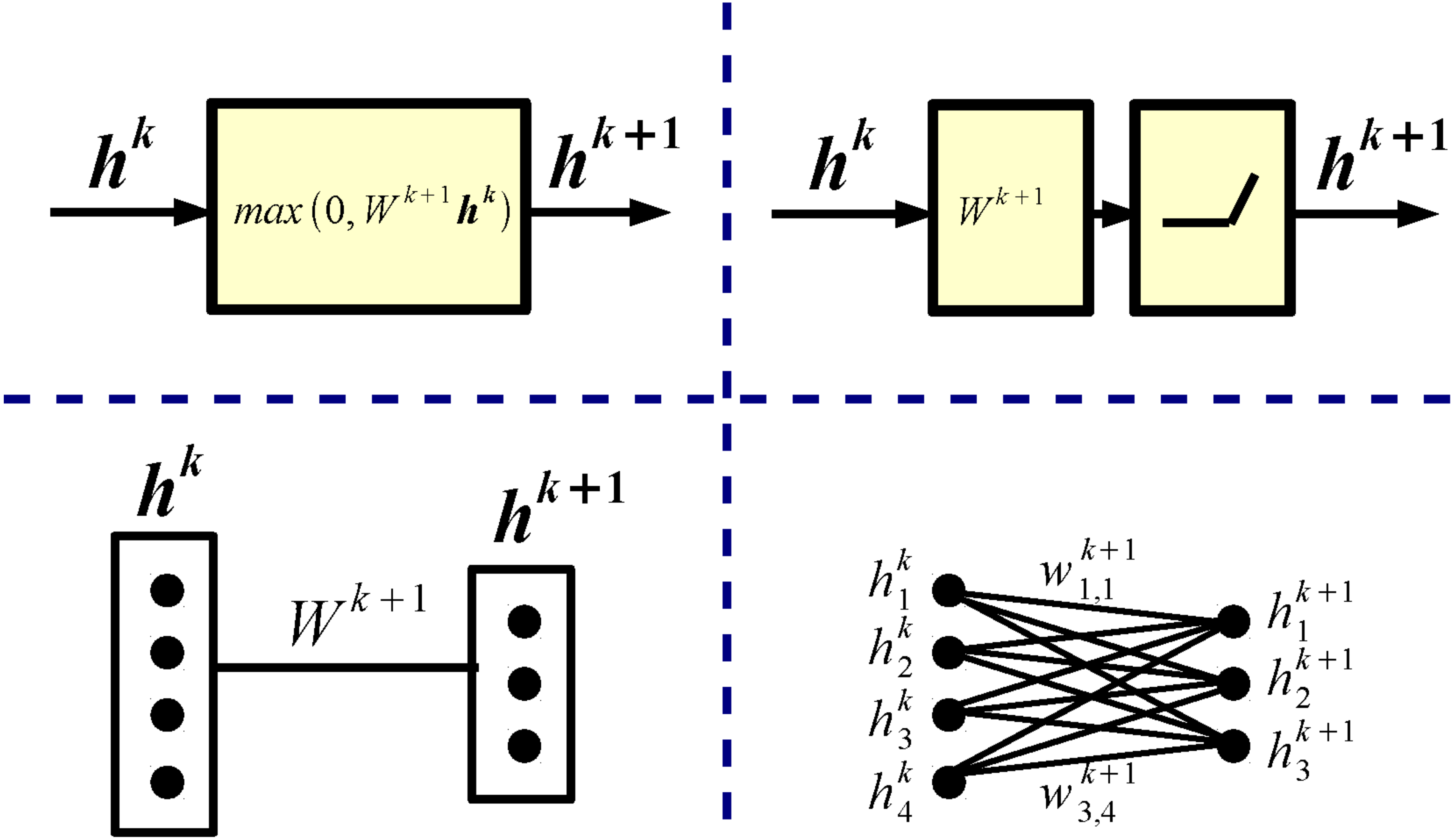
$\boldsymbol{h}^2 \in R^{N_2} \quad W^3 \in R^{N_3 \times N_2} \quad \boldsymbol{b}^3 \in R^{N_3} \qquad \boldsymbol{o} \in R^{N_3}$

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

W^3 3-rd layer weight matrix or weights

$\boldsymbol{b}^3$ 3-rd layer biases

Alternative Graphical Representation

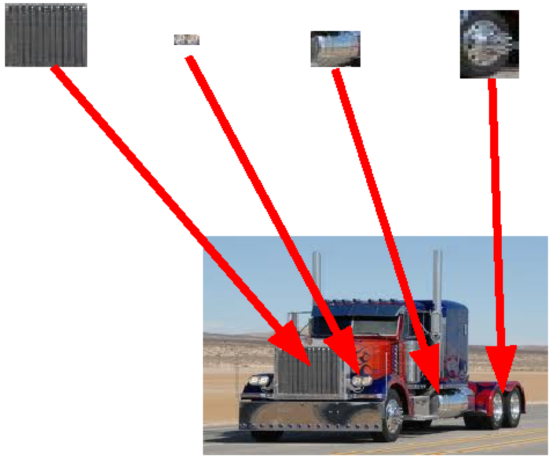


Interpretation

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

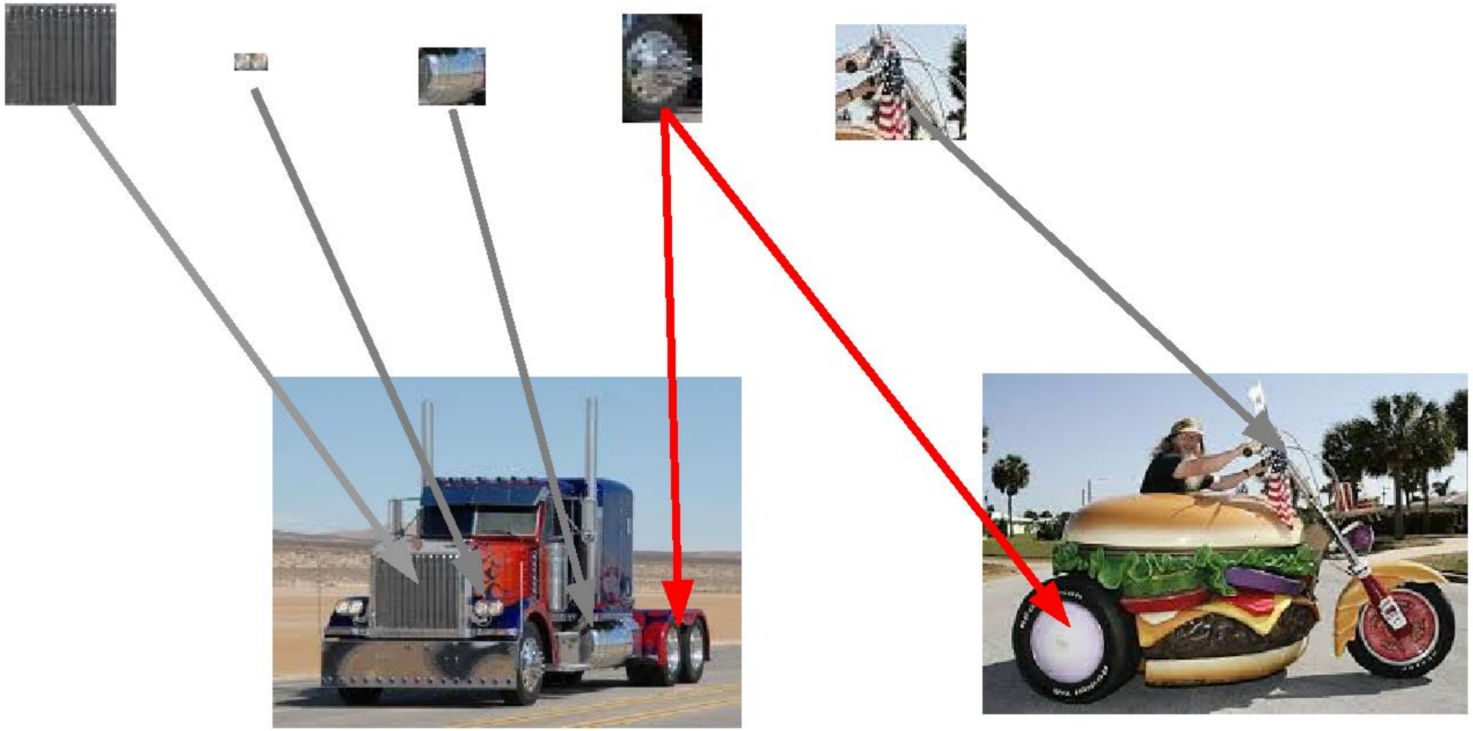


Exponentially more efficient than a 1-of-N representation (a la k-means)

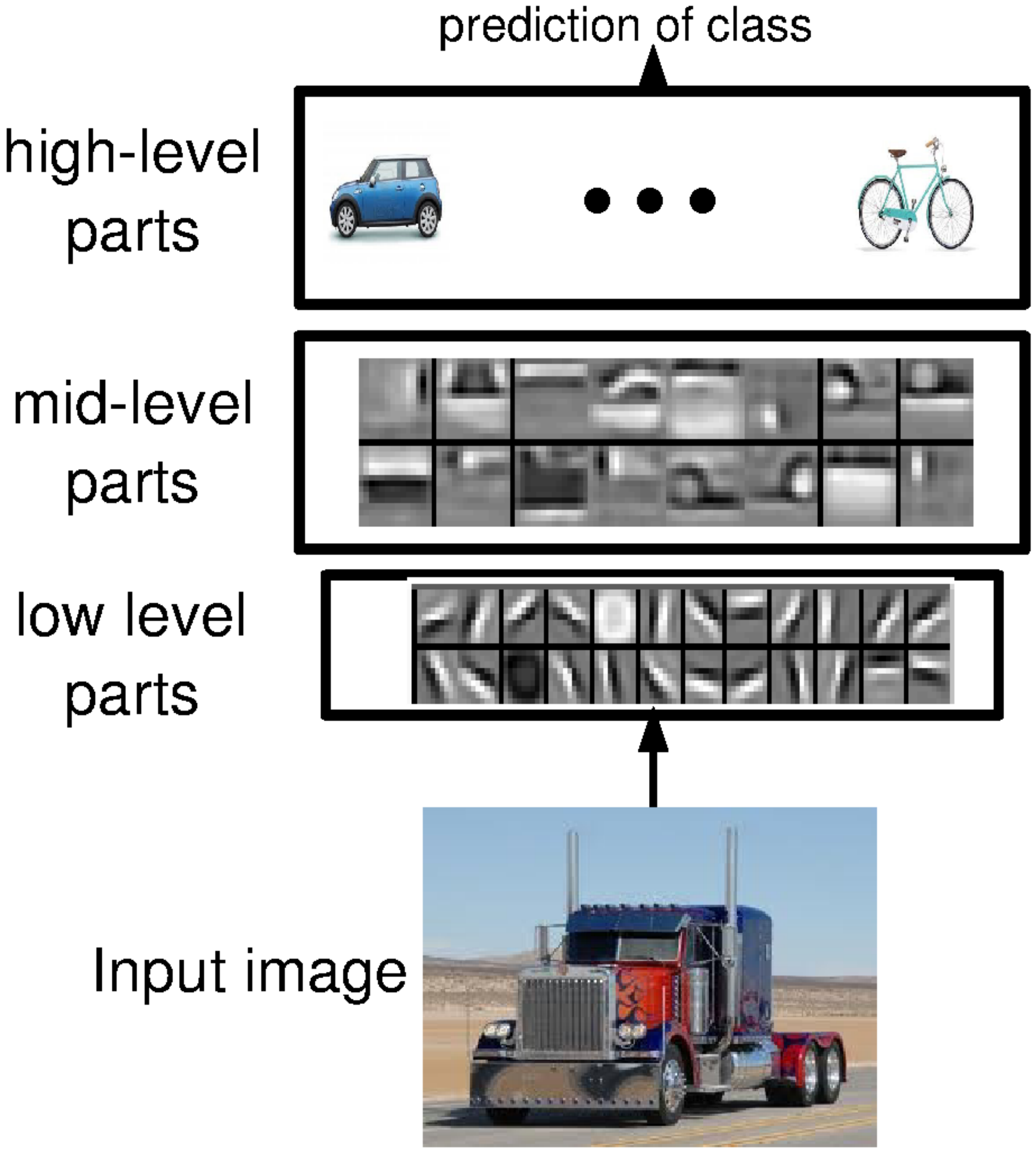
Interpretation

[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



prediction of class

high-level parts

mid-level parts

low level parts

Input image

- distributed representations
- feature sharing
- compositionality

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.

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.

Interpretation

Question: What does a hidden unit do?

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

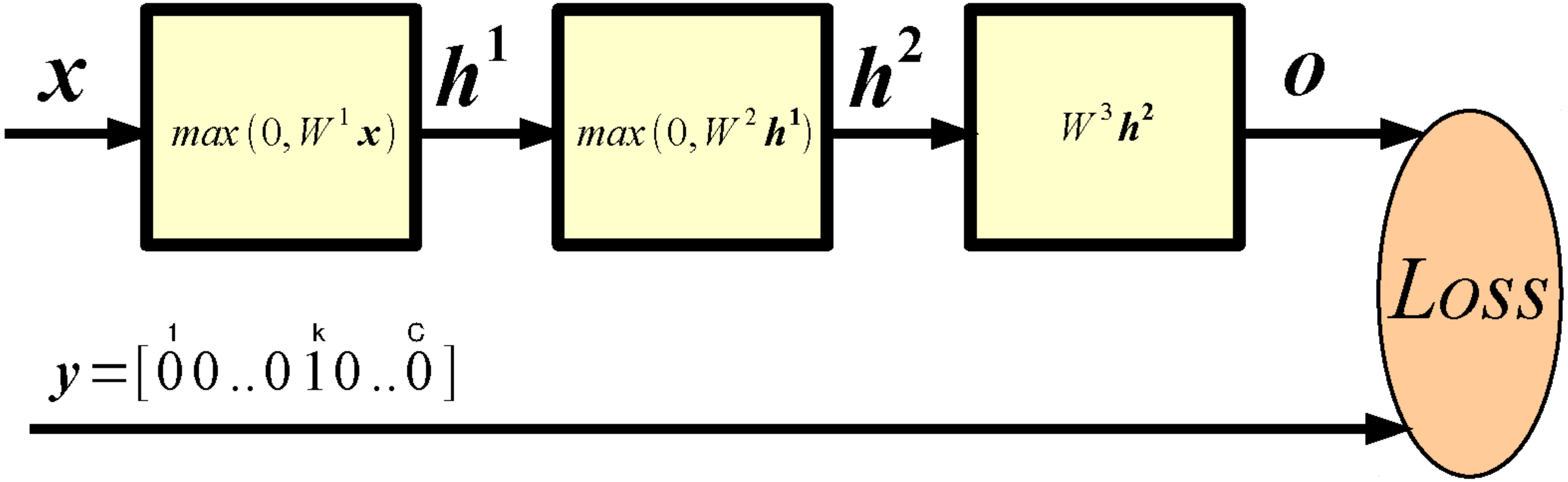
Question: How many layers? How many hidden units?

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.

How Good is a Network?


x
$\max(0, W^1 x)$
h^1
$\max(0, W^2 h^1)$
h^2
$W^3 h^2$
o
$Loss$
$y = [\overset{1}{0}\, 0 \,..\, 0\, \overset{k}{1}\, 0 \,..\, \overset{C}{0}]$

Probability of class k given input (softmax):

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

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

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

Training

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

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

Training

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

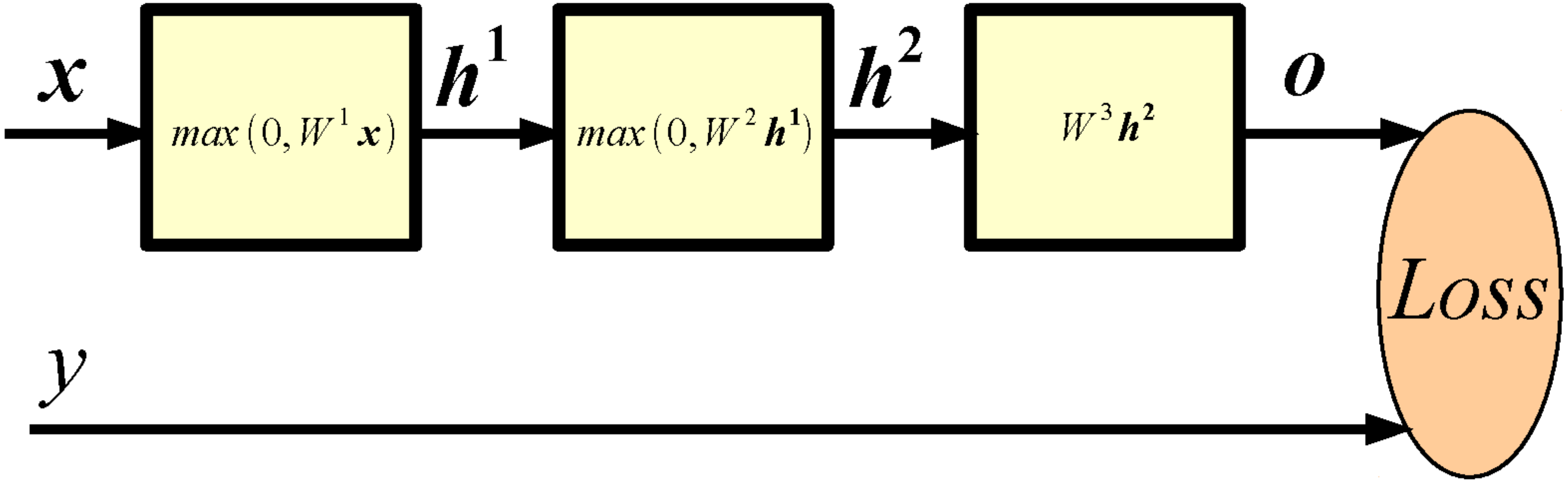
$$\boldsymbol{\theta}^* = arg\, min_{\boldsymbol{\theta}} \sum_{n=1}^{P} L(\boldsymbol{x}^n, y^n; \boldsymbol{\theta})$$

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.

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 $W^1_{i,j}$.
We could consider a very small $\epsilon = 1\text{e-}6$ and compute:

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

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

Then, update:

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

Derivative w.r.t. Input of Softmax

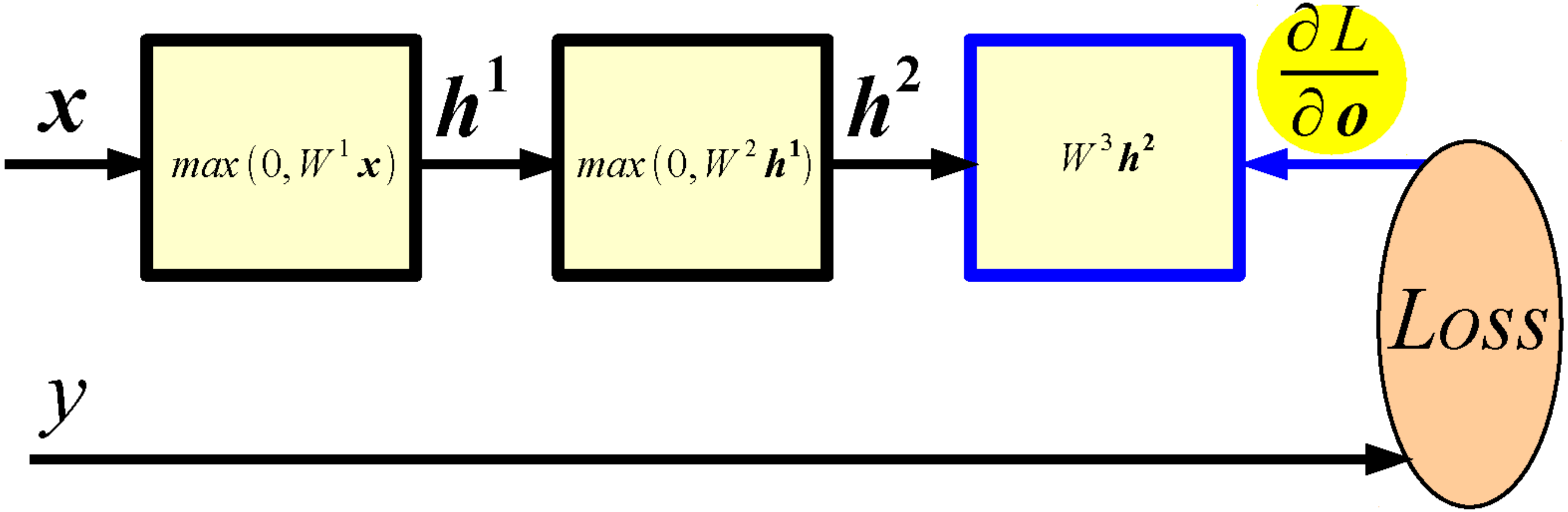
$$p(c_k = 1 | \boldsymbol{x}) = \frac{e^{o_k}}{\sum_j e^{o_j}}$$

$$L(\boldsymbol{x}, y; \boldsymbol{\theta}) = -\sum_j y_j \log p(c_j | \boldsymbol{x}) \qquad \boldsymbol{y} = [\overset{1}{0}\, 0 \,..\, 0\, \overset{k}{1}\, 0 \,..\, \overset{C}{0}]$$

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

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

Backward Propagation

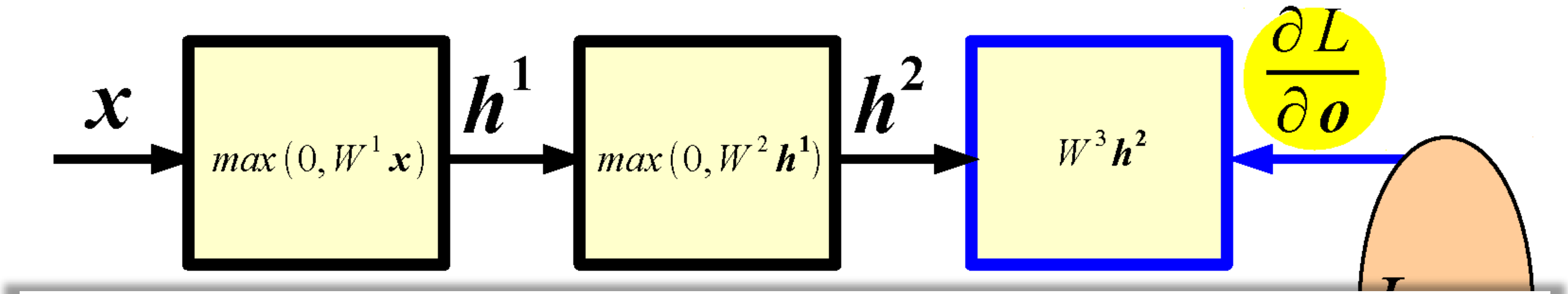


x
$\max(0, W^1 \boldsymbol{x})$
$\boldsymbol{h}^1$
$\max(0, W^2 \boldsymbol{h}^1)$
$\boldsymbol{h}^2$
$W^3 \boldsymbol{h}^2$
$\frac{\partial L}{\partial \boldsymbol{o}}$
$Loss$
y

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

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^2}$$

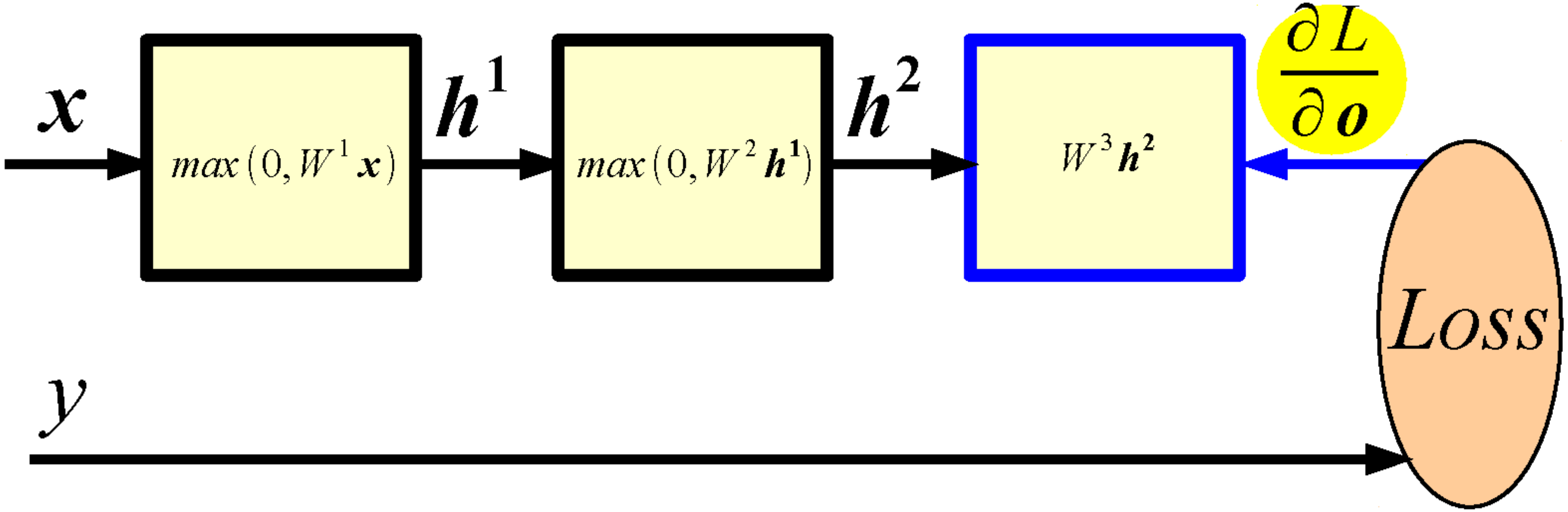
Backward Propagation



Suppose $\mathbf{f} : \mathbf{R}^n \to \mathbf{R}^m$ is a function such that each of its first-order partial derivatives exist on $\mathbf{R}^n$. This function takes a point $\mathbf{x} \in \mathbf{R}^n$ as input and produces the vector $\mathbf{f}(\mathbf{x}) \in \mathbf{R}^m$ as output. Then the Jacobian matrix of $\mathbf{f}$ is defined to be an $m \times n$ matrix, denoted by $\mathbf{J}$, whose (i,j)th entry is $\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$, or explicitly

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

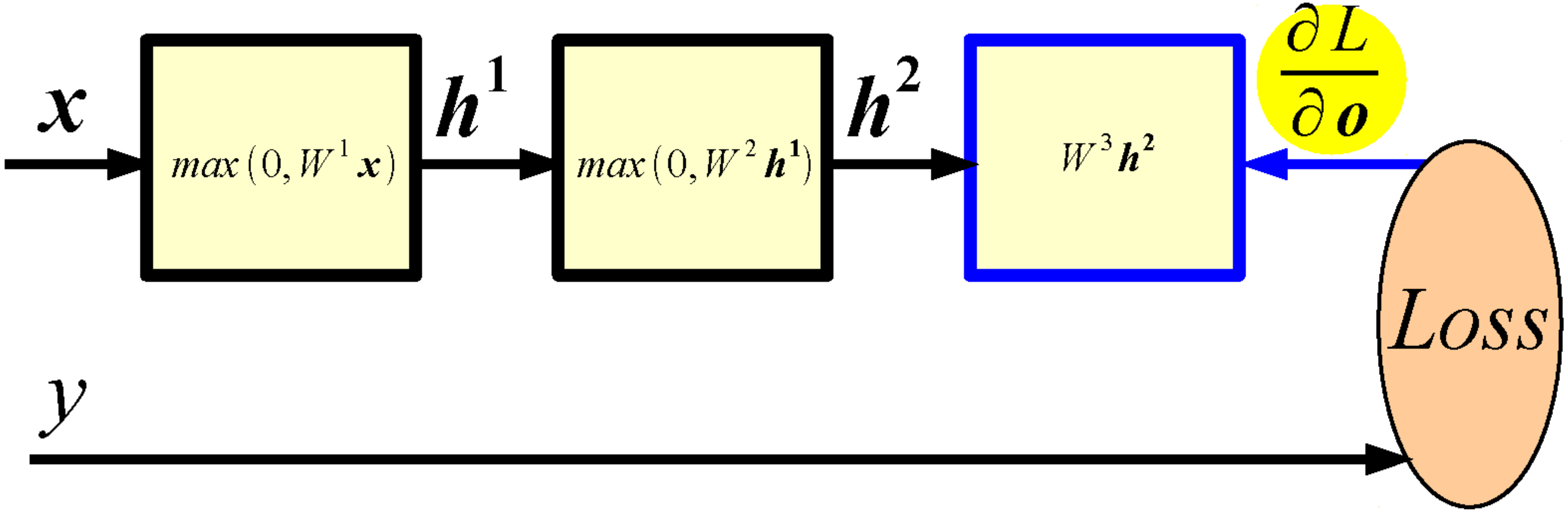
Backward Propagation



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

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial W^3} \qquad \frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2}$$

Backward Propagation



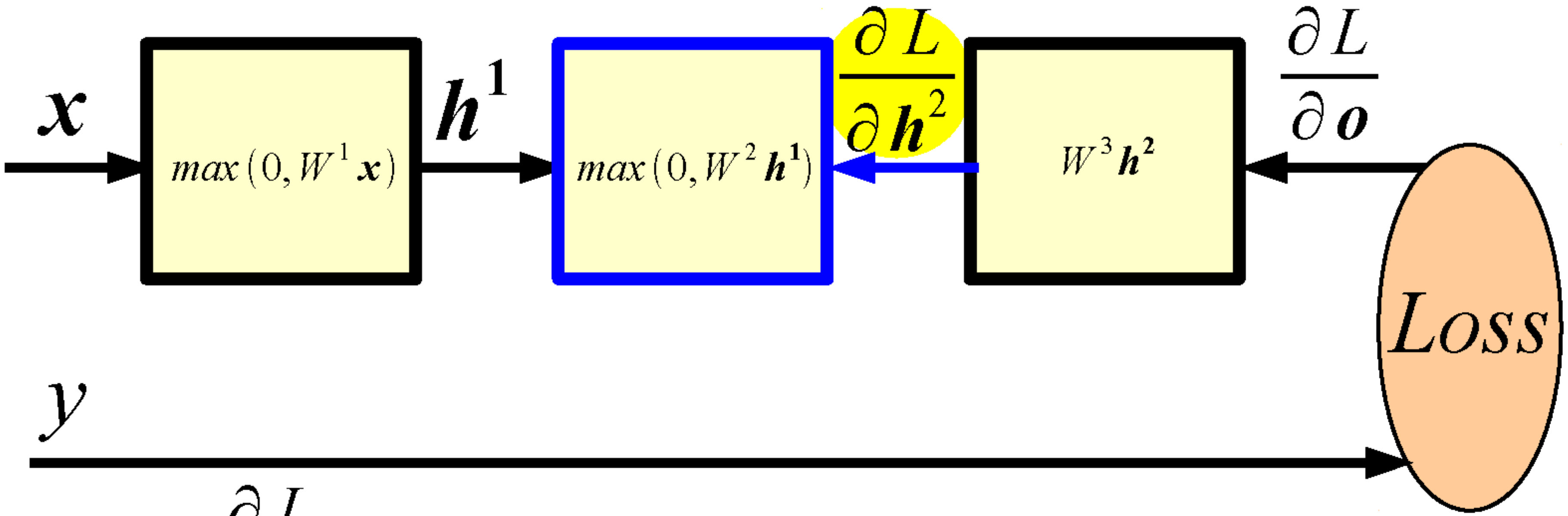
x
$max(0, W^1 \boldsymbol{x})$
$\boldsymbol{h}^1$
$max(0, W^2 \boldsymbol{h}^1)$
$\boldsymbol{h}^2$
$W^3 \boldsymbol{h}^2$
$\frac{\partial L}{\partial \boldsymbol{o}}$
$Loss$
y

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

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^2}$$

$$\frac{\partial L}{\partial W^3} = (p(c|\boldsymbol{x}) - \boldsymbol{y})\ \boldsymbol{h}^{2\,T} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = W^{3\,T}(p(c|\boldsymbol{x}) - \boldsymbol{y})$$

Backward Propagation

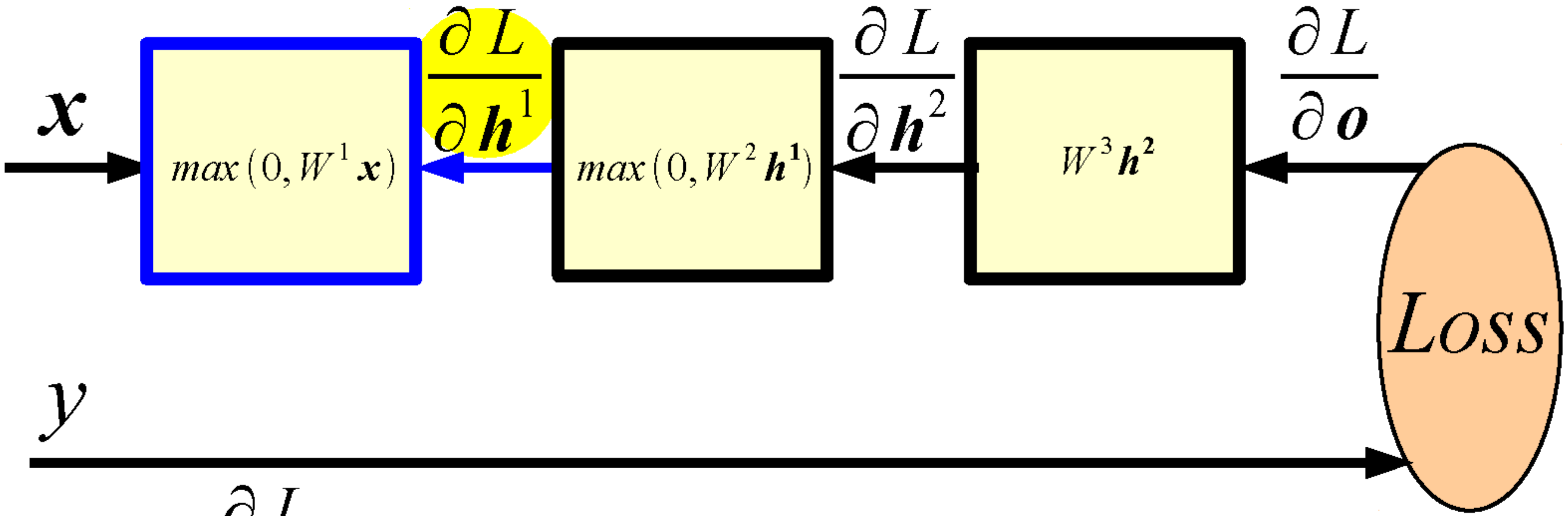


$\boldsymbol{x}$
$max(0, W^1 \boldsymbol{x})$
$\boldsymbol{h}^1$
$max(0, W^2 \boldsymbol{h}^1)$
$\frac{\partial L}{\partial \boldsymbol{h}^2}$
$W^3 \boldsymbol{h}^2$
$\frac{\partial L}{\partial \boldsymbol{o}}$
$Loss$
y

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

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

Backward Propagation


x
$max(0, W^1 x)$
$\frac{\partial L}{\partial h^1}$
$max(0, W^2 h^1)$
$\frac{\partial L}{\partial h^2}$
$W^3 h^2$
$\frac{\partial L}{\partial o}$
Loss
y

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

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

Backward Propagation

Question: Does BPROP work with ReLU layers only?

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

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).

Optimization

Stochastic Gradient Descent (on mini-batches):

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial L}{\partial \boldsymbol{\theta}}, \eta \in (0, 1)$$

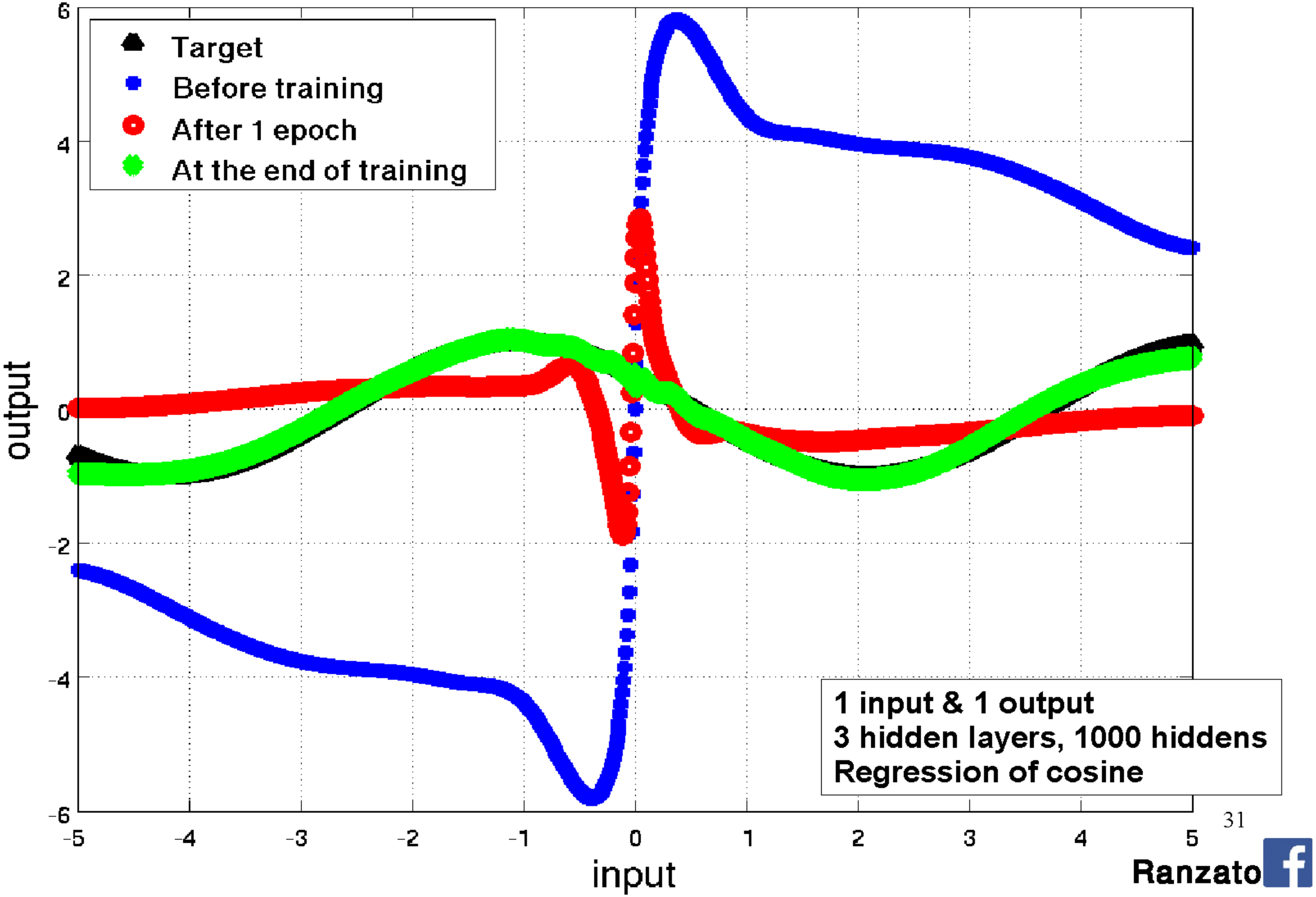
Stochastic Gradient Descent with Momentum:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{\Delta}$$

$$\boldsymbol{\Delta} \leftarrow 0.9 \boldsymbol{\Delta} + \frac{\partial L}{\partial \boldsymbol{\theta}}$$

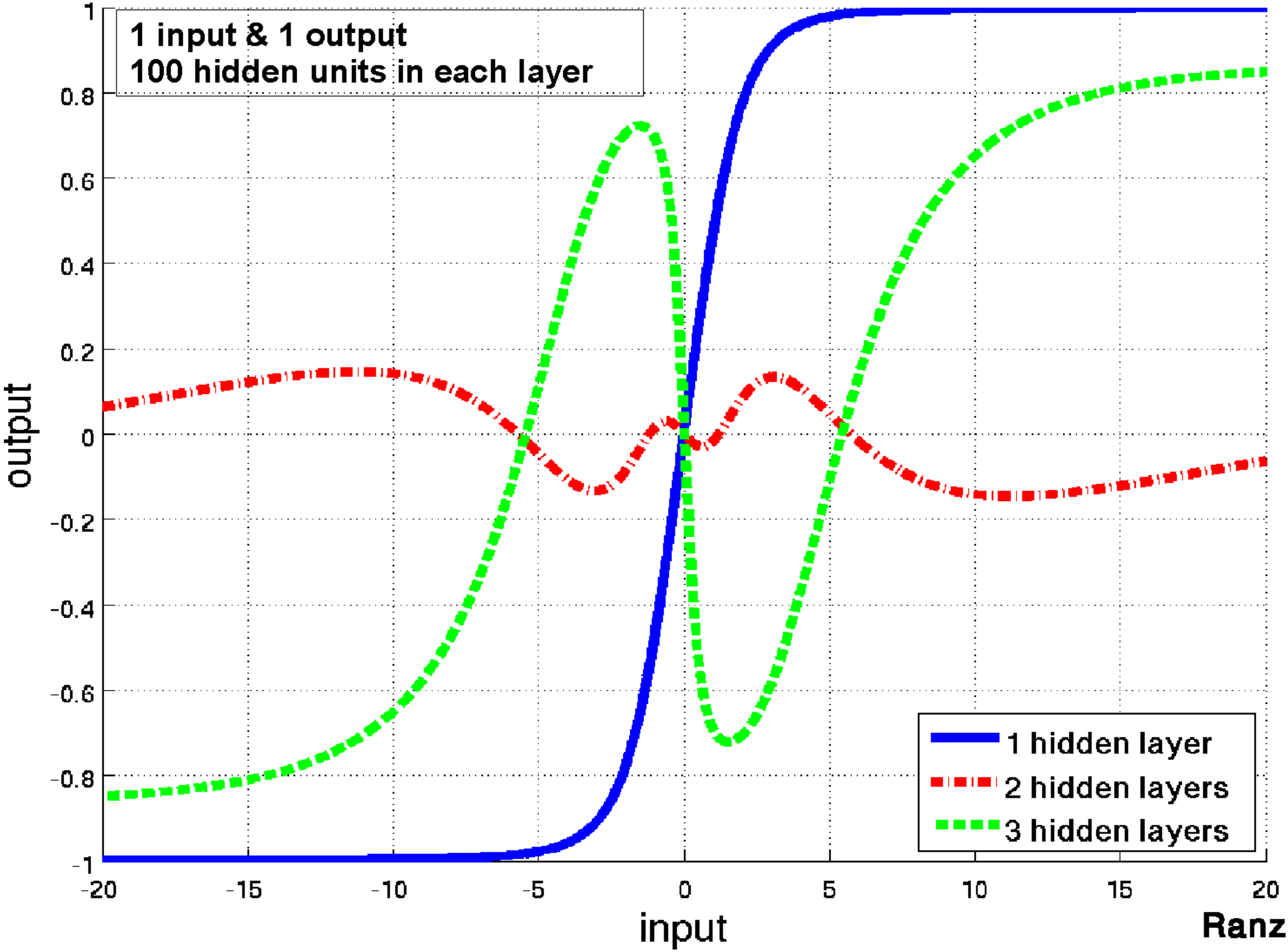
Note: there are many other variants...

Toy Example: Synthetic Data



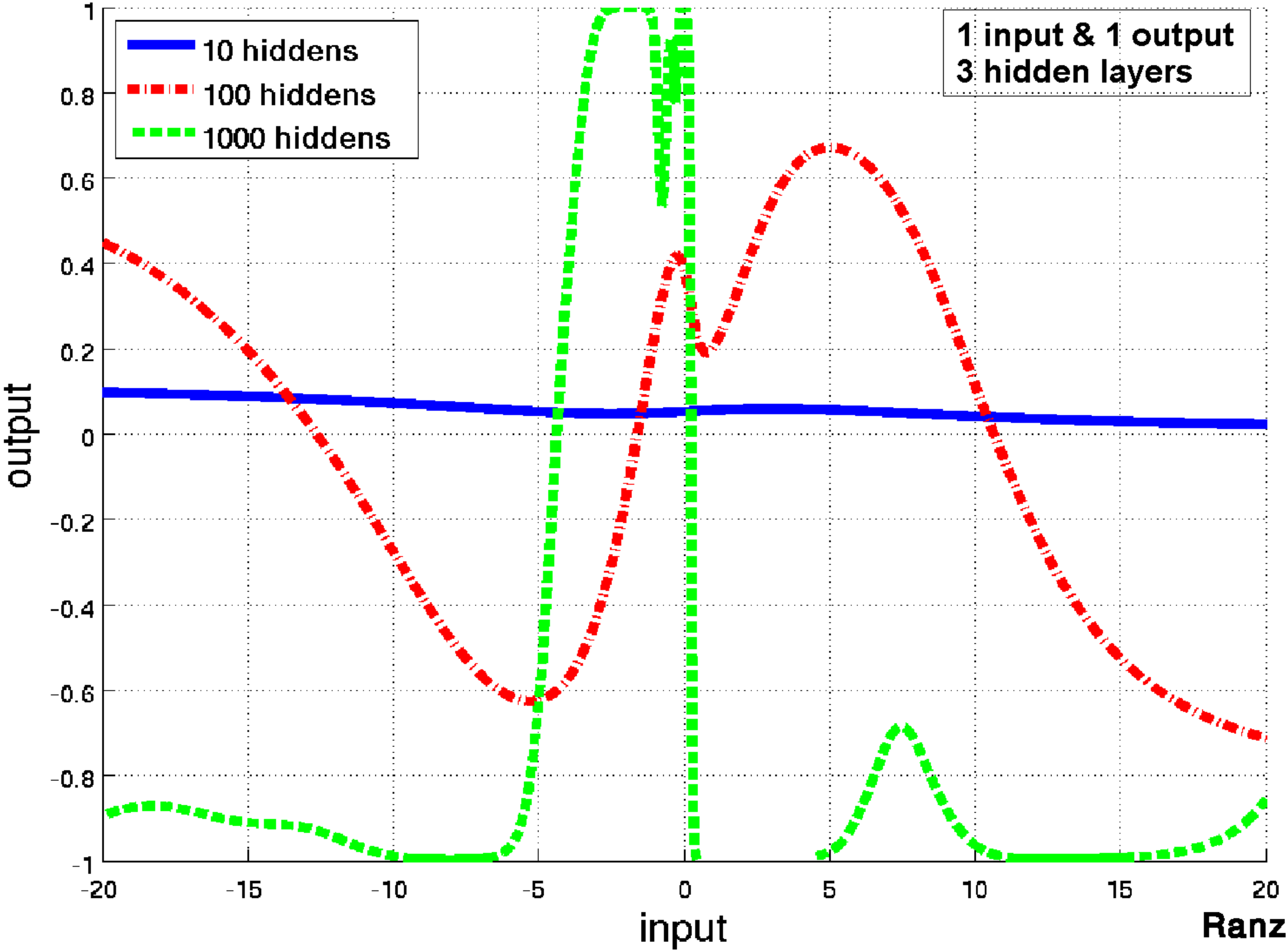
1 input & 1 output
3 hidden layers, 1000 hiddens
Regression of cosine

Toy Example: Synthetic Data



Ranzato

Toy Example: Synthetic Data



Ranzato

Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips

This all seems pretty complicated. Why are we using Neural Networks? James's rough assessment:

Learning method	Ease of configuration
Neural Network	1
Nearest Neighbor	10
Linear SVM	10
Non-linear SVM	5
Decision Tree or Random Forest	4

This all seems pretty complicated. Why are we using Neural Networks? James's rough assessment:

Learning method	Ease of configuration	Ease of interpretation
Neural Network	1	1
Nearest Neighbor	10	10
Linear SVM	10	9
Non-linear SVM	5	4
Decision Tree or Random Forest	4	4

This all seems pretty complicated. Why are we using Neural Networks? James's rough assessment:

Learning method	Ease of configuration	Ease of interpretation	Speed / memory when training
Neural Network	1	1	1
Nearest Neighbor	10	10	8
Linear SVM	10	9	10
Non-linear SVM	5	4	2
Decision Tree or Random Forest	4	4	4

This all seems pretty complicated. Why are we using Neural Networks? James’s rough assessment:

Learning method	Ease of configuration	Ease of interpretation	Speed / memory when training	Speed / memory at test time
Neural Network	1	1	1	6
Nearest Neighbor	10	10	8	4
Linear SVM	10	9	10	10
Non-linear SVM	5	4	2	2
Decision Tree or Random Forest	4	4	4	8

This all seems pretty complicated. Why are we using Neural Networks? James's rough assessment:

Learning method	Ease of configuration	Ease of interpretation	Speed / memory when training	Speed / memory at test time	Accuracy w/ lots of data
Neural Network	1	1	1	6	10
Nearest Neighbor	10	10	8	4	7
Linear SVM	10	9	10	10	5
Non-linear SVM	5	4	2	2	8
Decision Tree or Random Forest	4	4	4	8	7

This all seems pretty complicated. Why are we using Neural Networks? James's rough assessment:

Learning method	Ease of configuration	Ease of interpretation	Speed / memory when training	Speed / memory at test time	Accuracy w/ lots of data
Neural Network	1	1	1	6	10
Nearest Neighbor	10	10	8	4	7
Linear SVM	10				
Non-linear SVM	5				
Decision Tree or Random Forest	4				

Representation design matters more for all of these

Outline

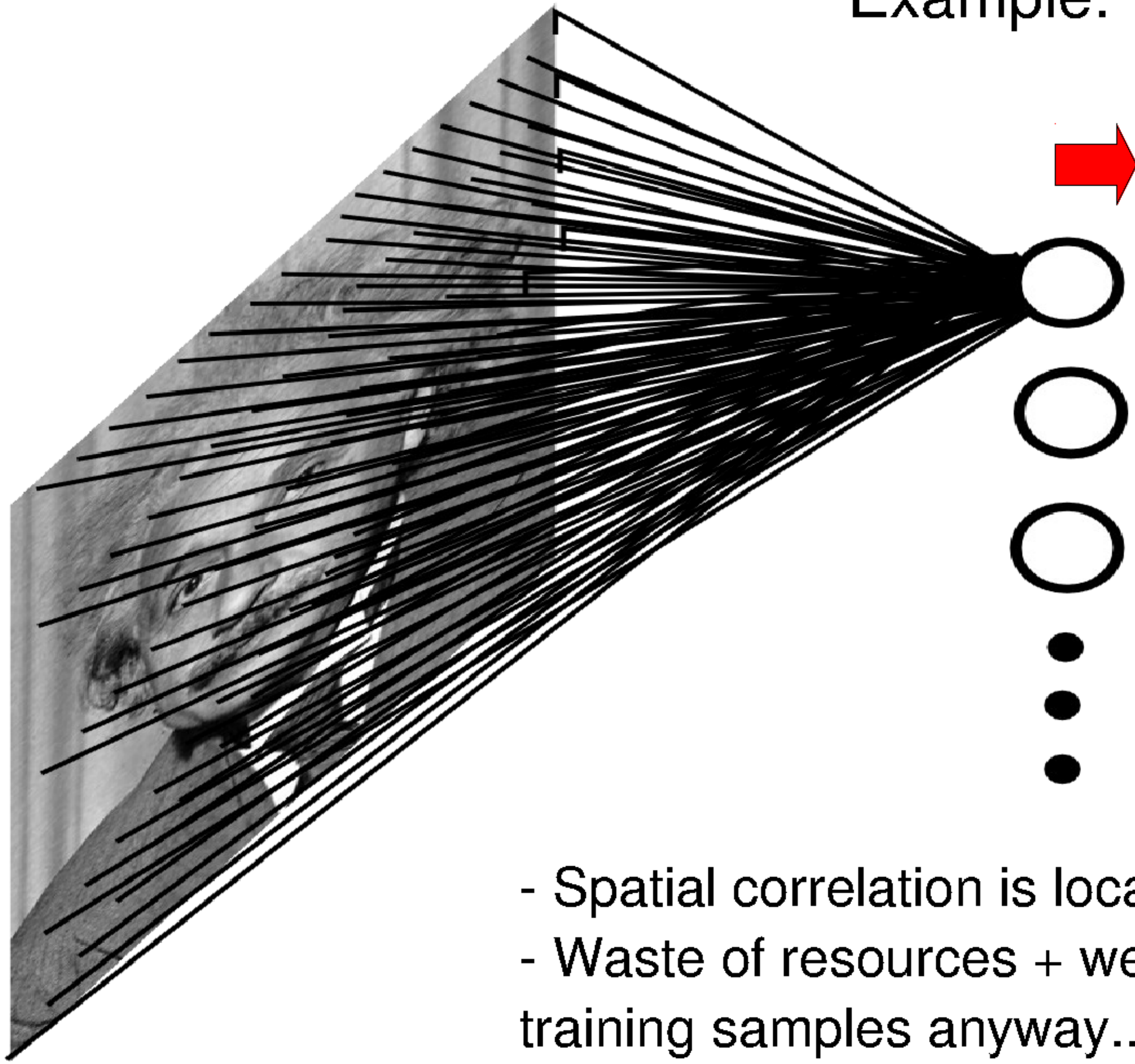
- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips

Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips

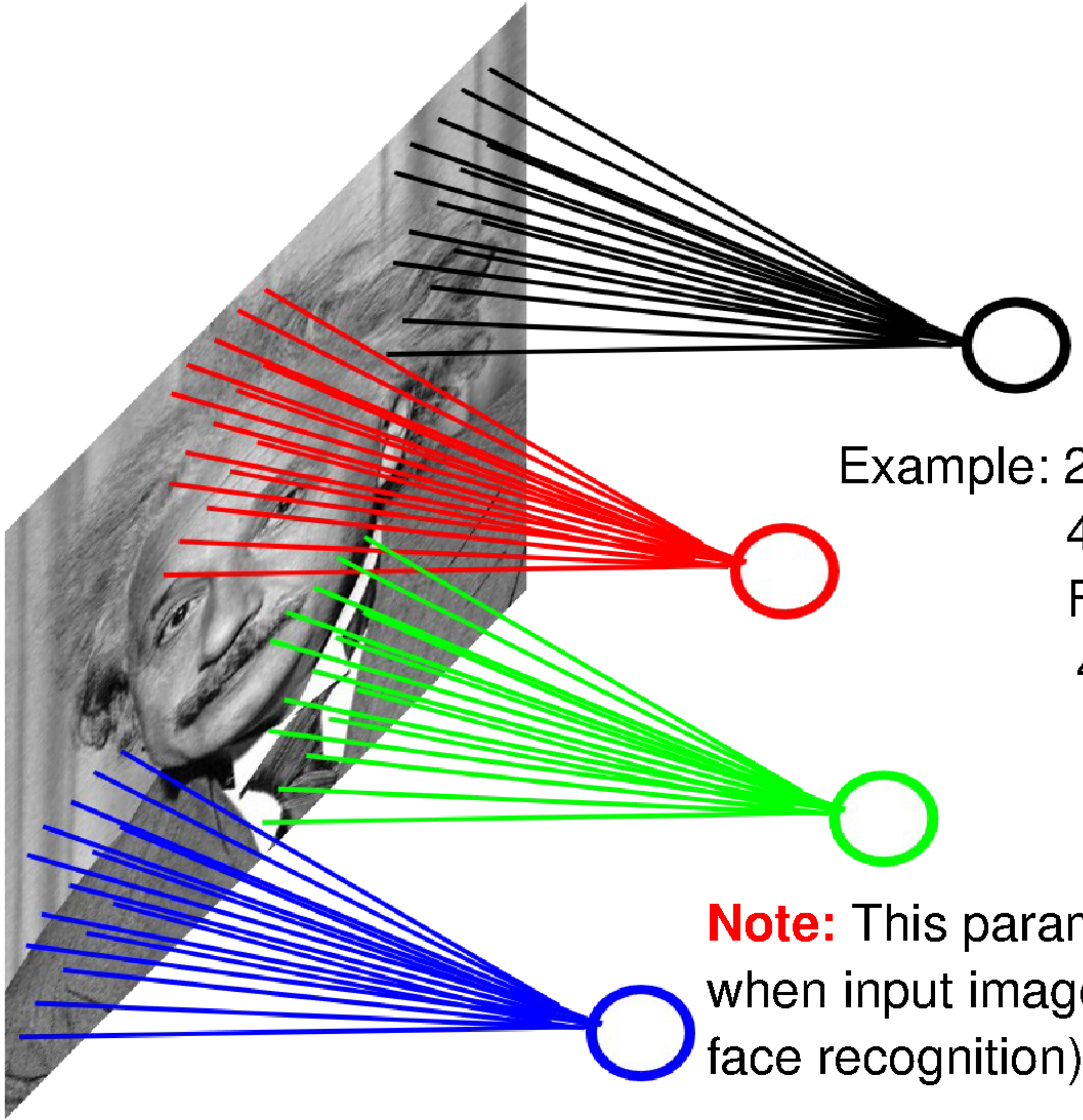
Fully Connected Layer

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



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

Locally Connected Layer

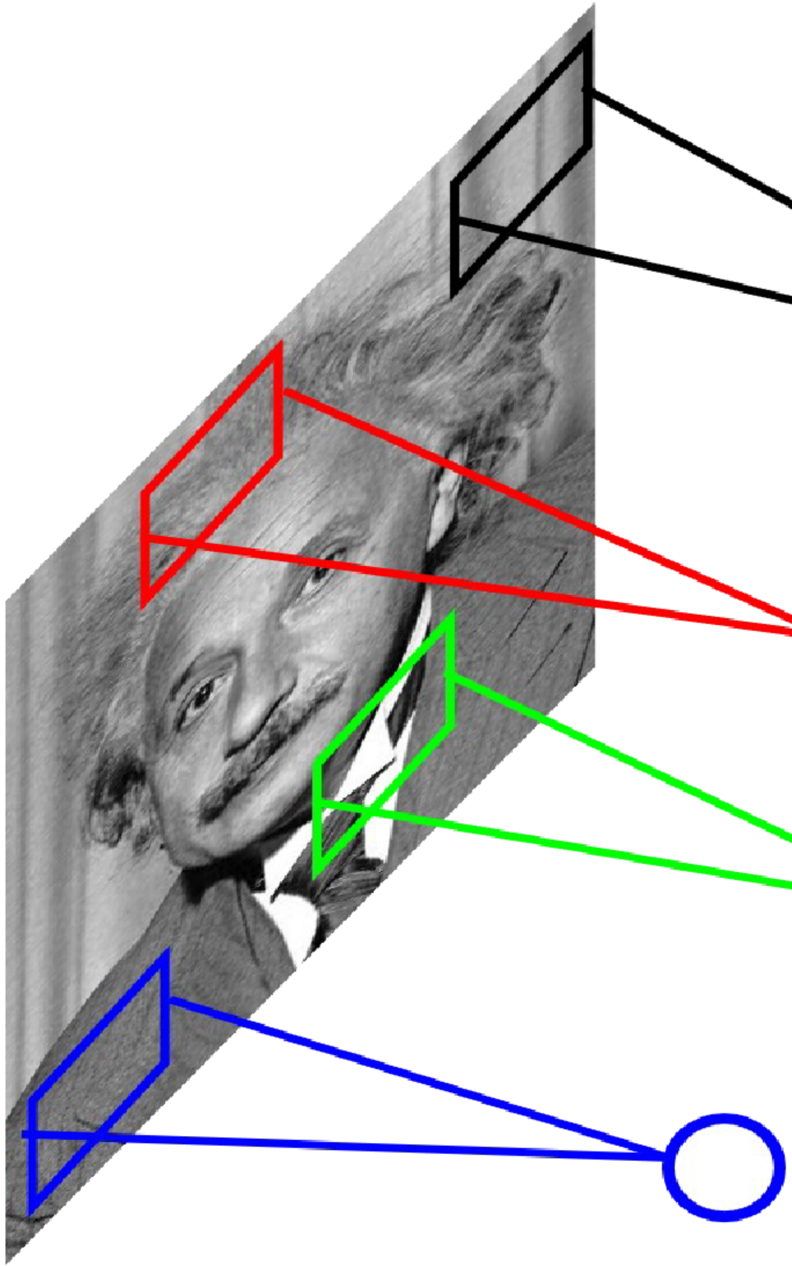


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

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

Ranzato

Locally Connected Layer

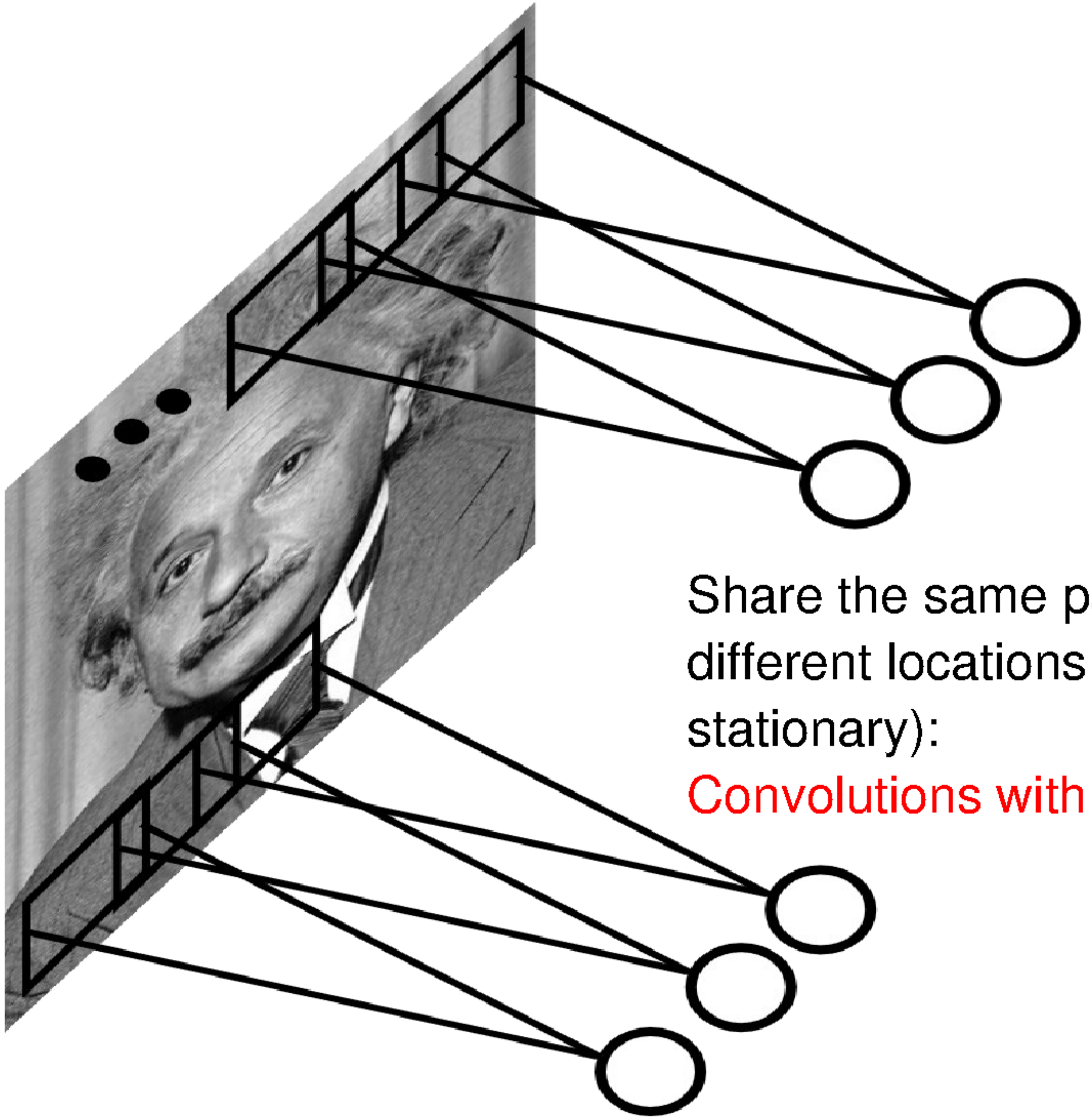


STATIONARITY? Statistics is similar at different locations

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).

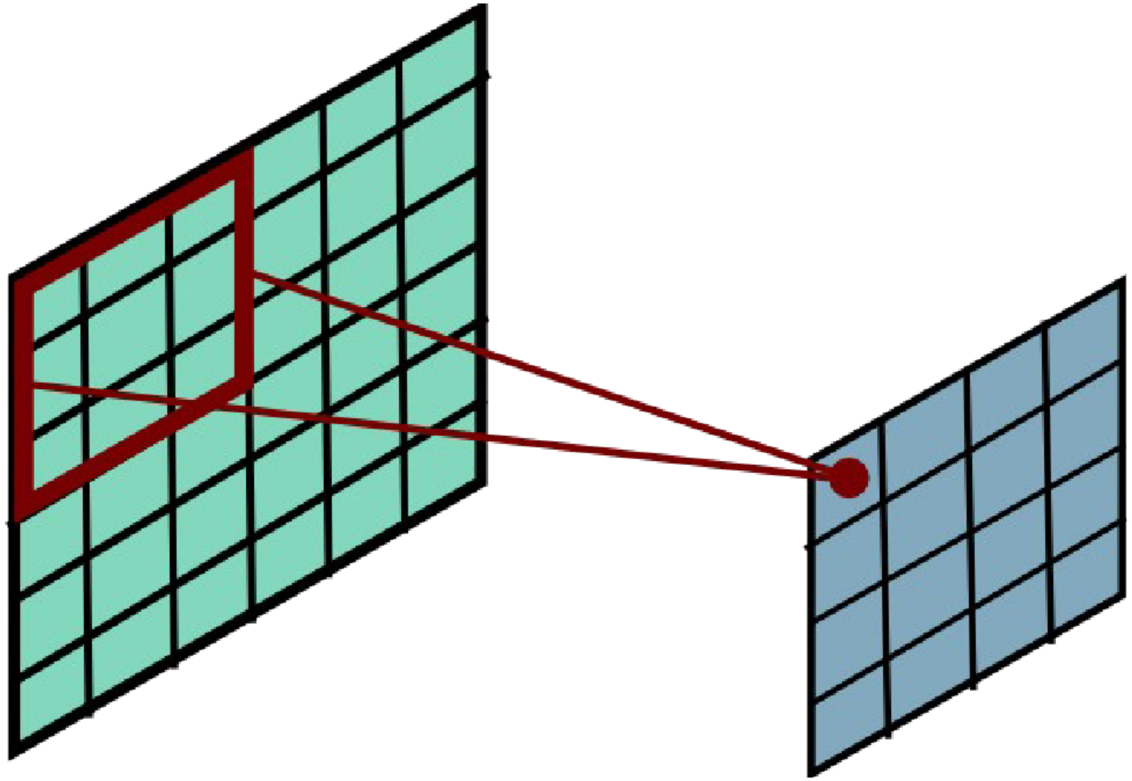
Convolutional Layer



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

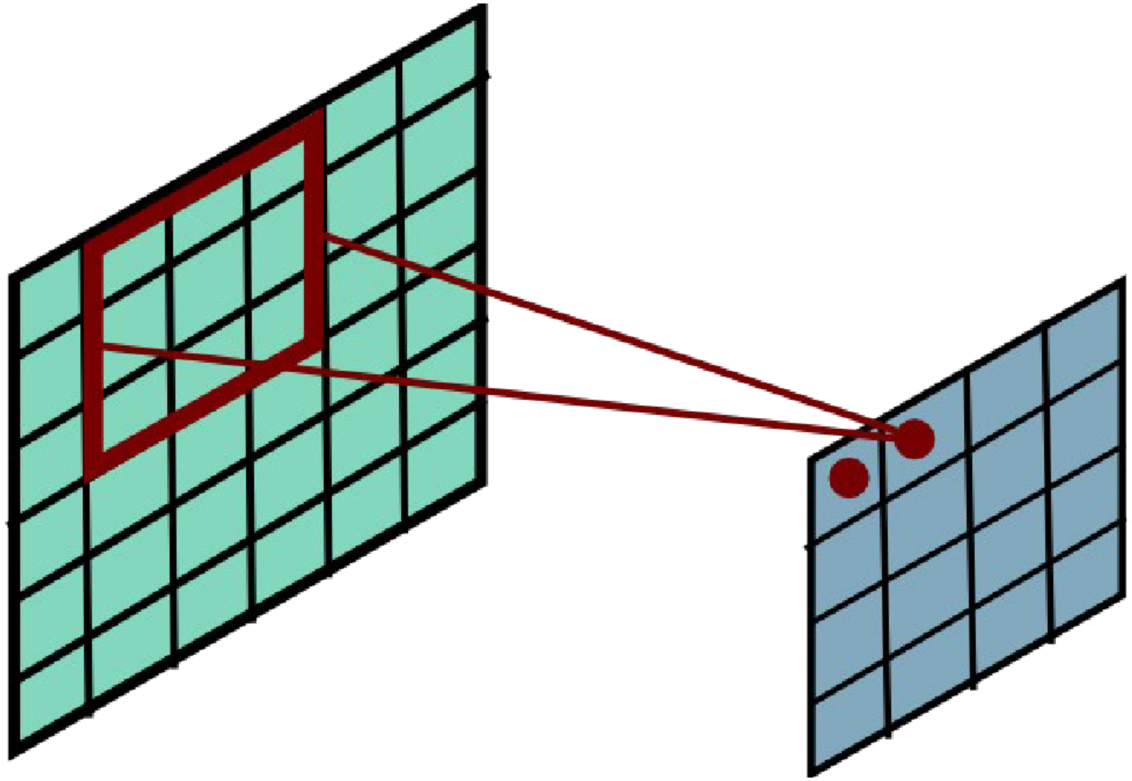
Convolutions with learned kernels

Convolutional Layer



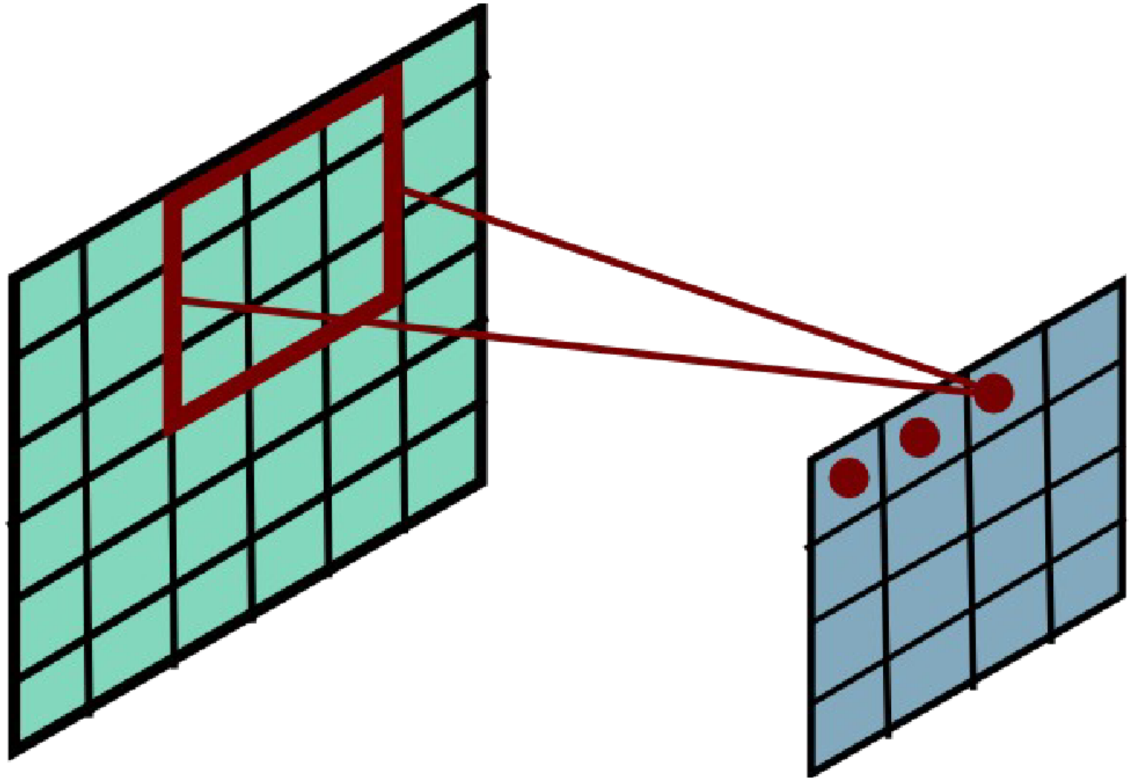
Ranzato

Convolutional Layer



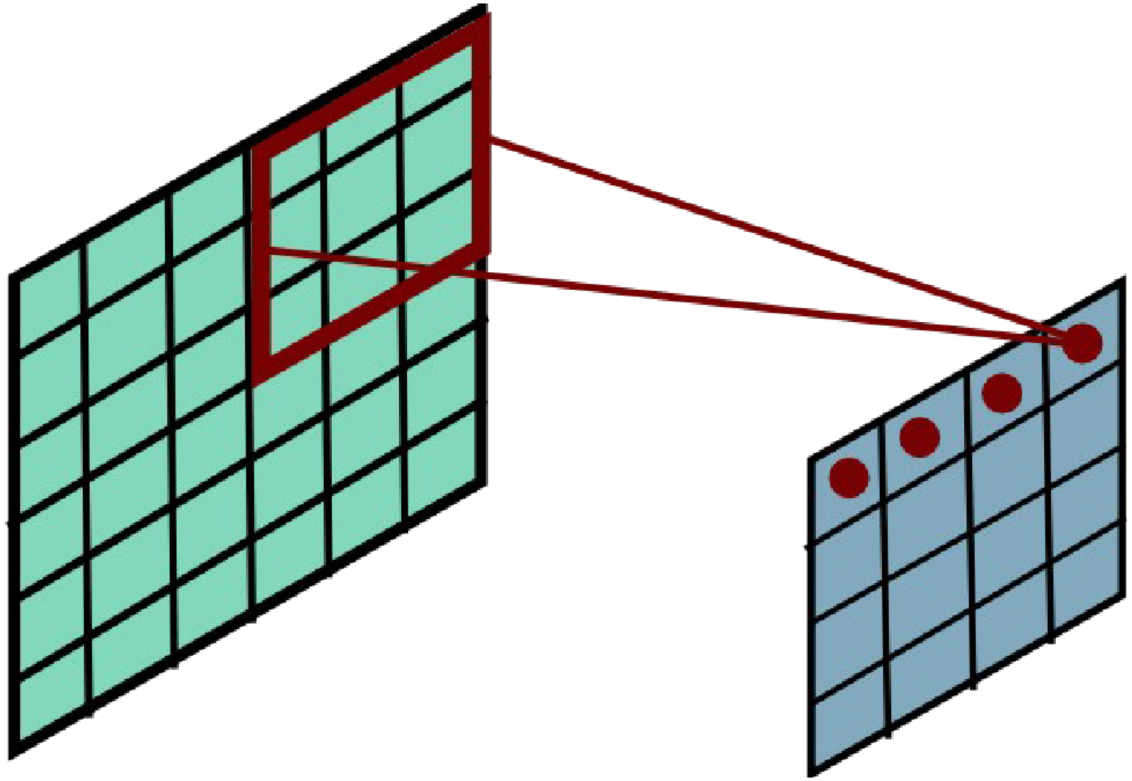
Ranzato

Convolutional Layer



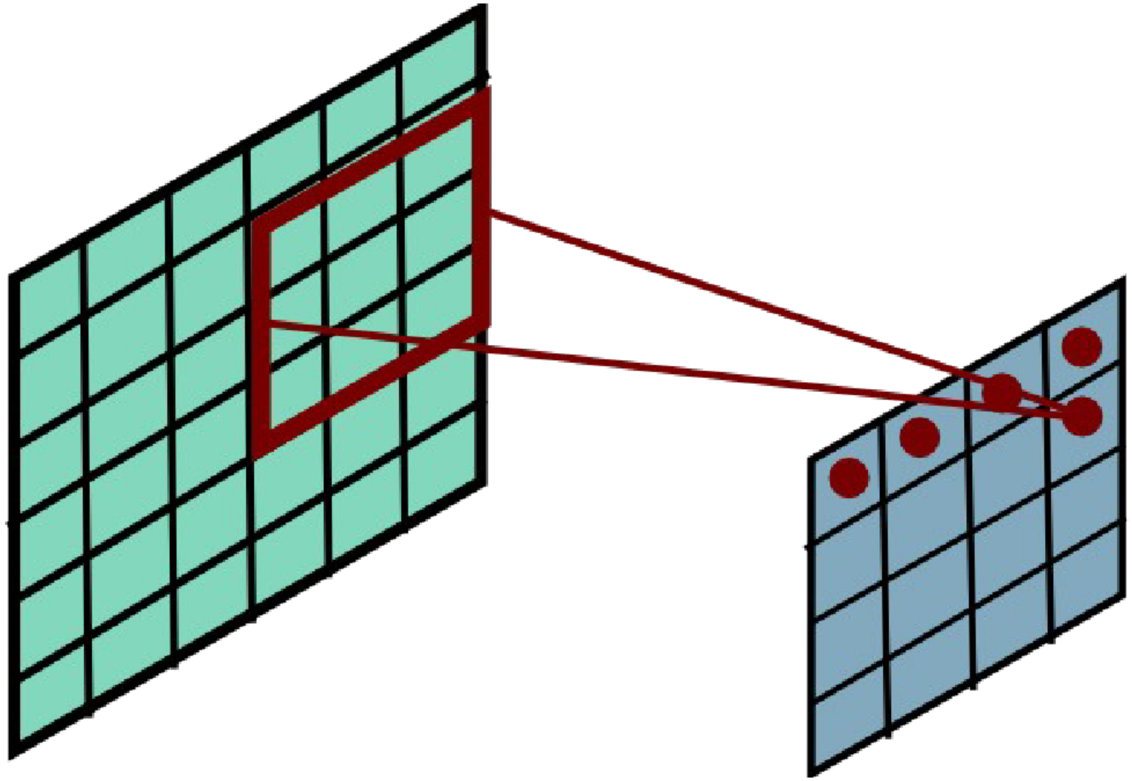
Ranzato

Convolutional Layer



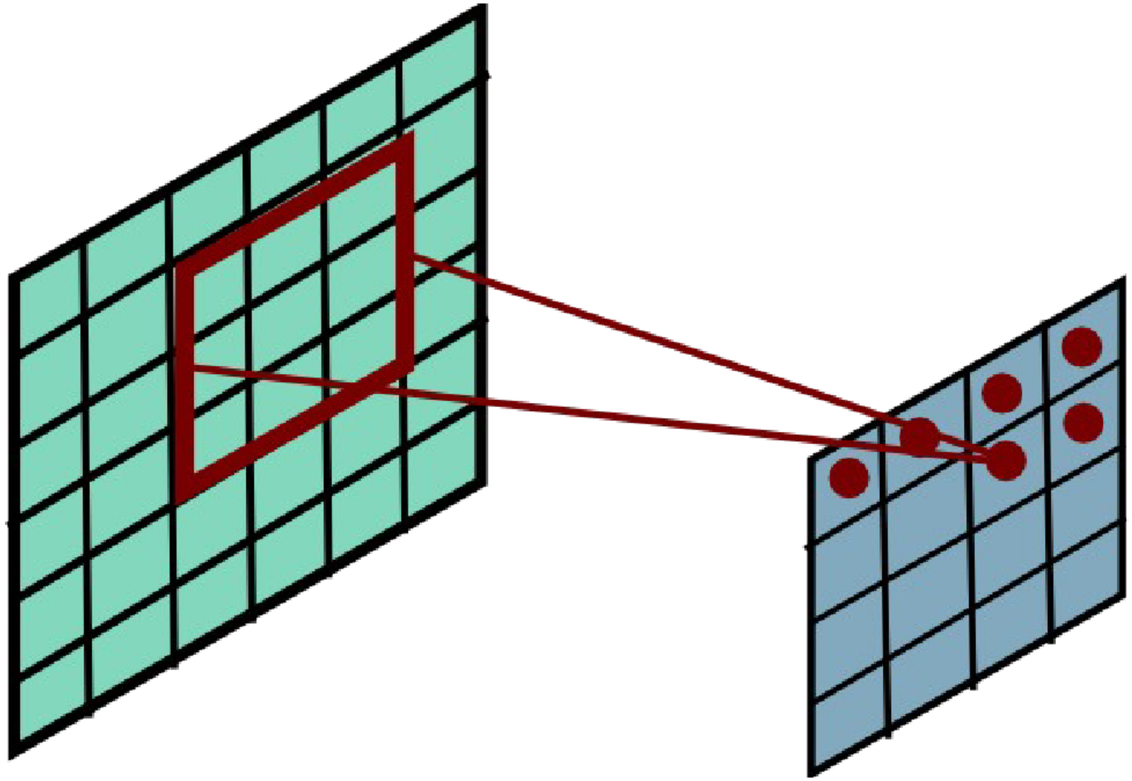
Ranzato

Convolutional Layer



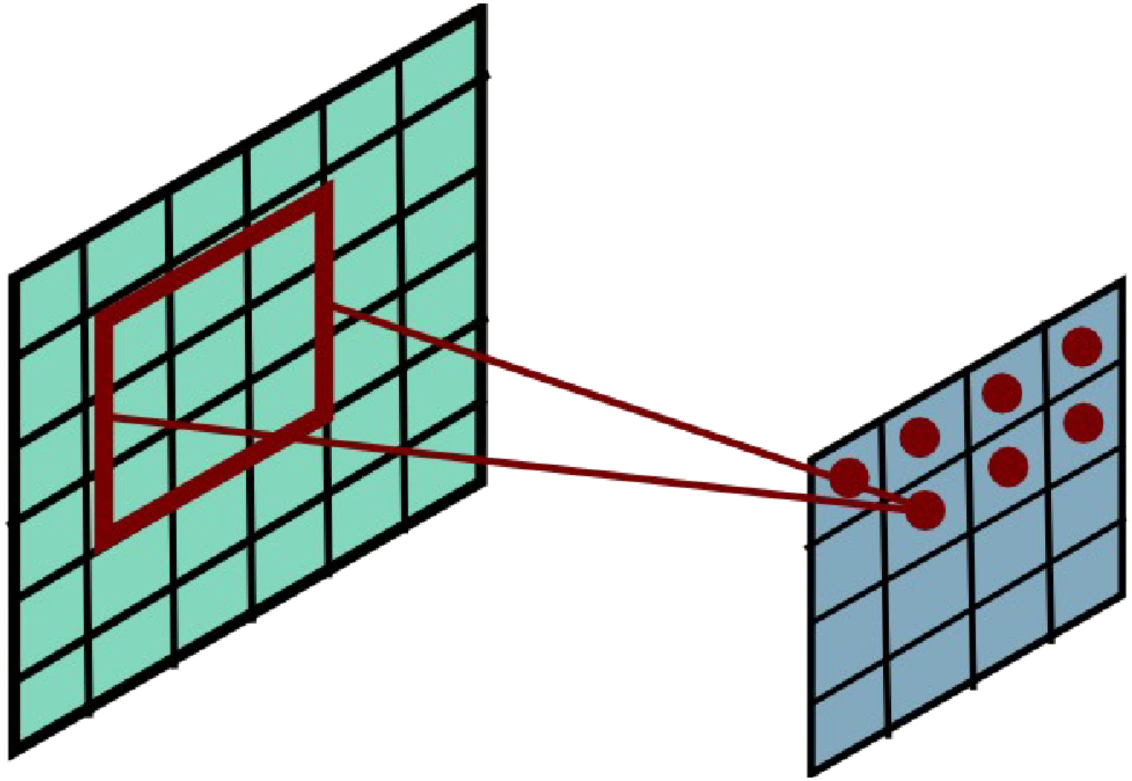
Ranzato

Convolutional Layer



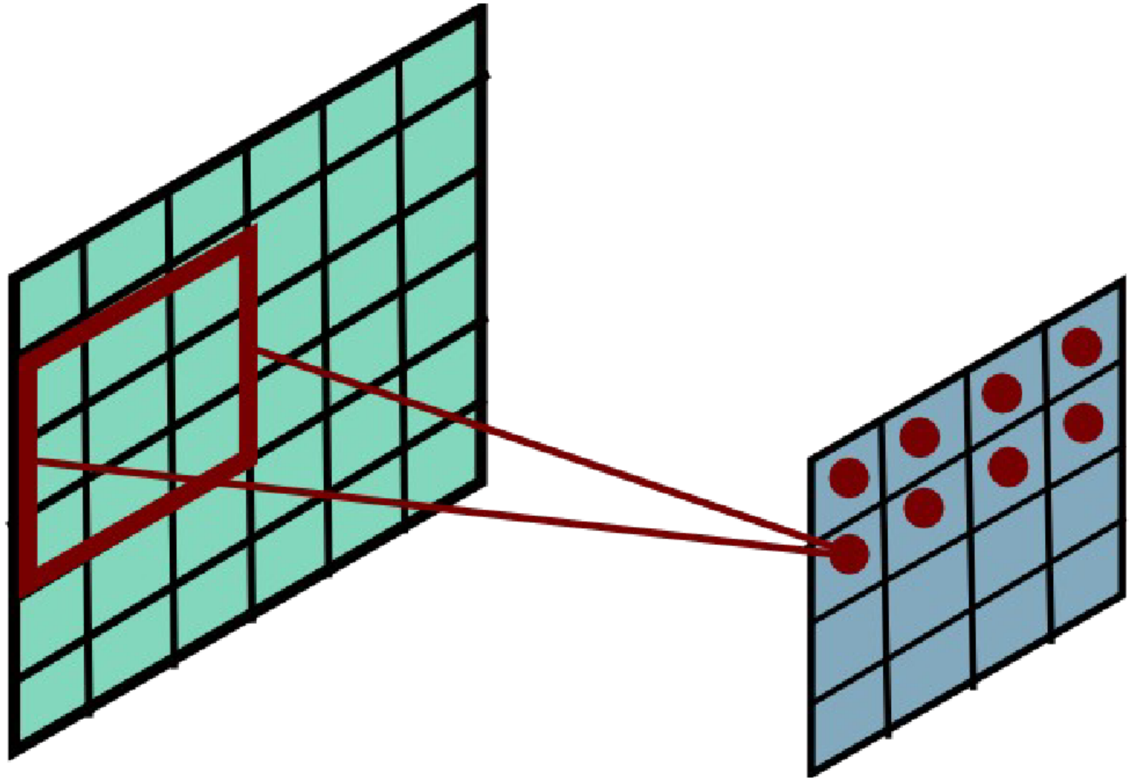
Ranzato

Convolutional Layer



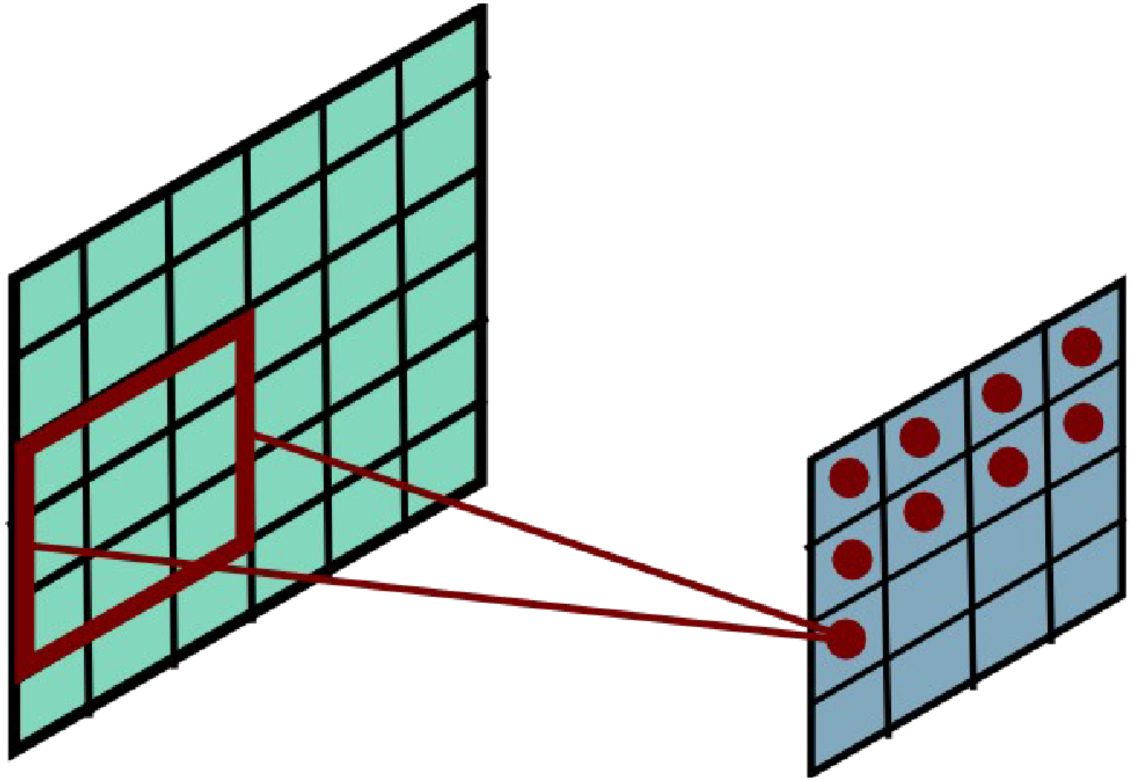
Ranzato

Convolutional Layer



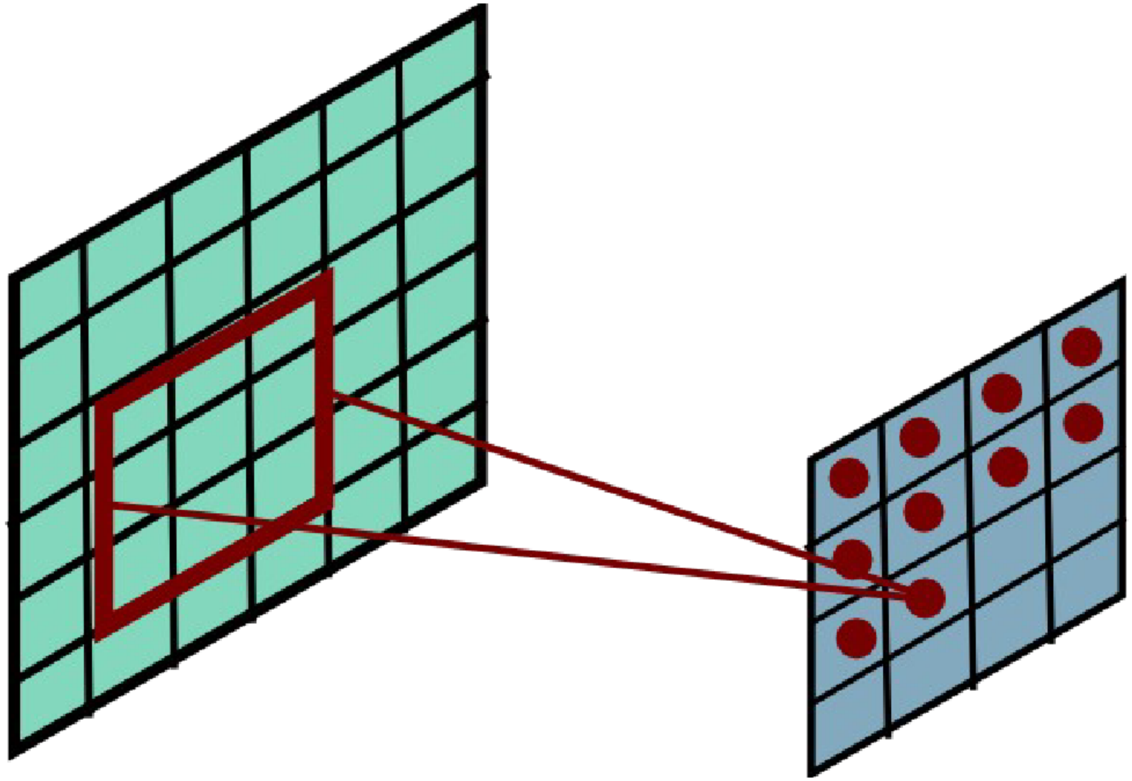
Ranzato

Convolutional Layer



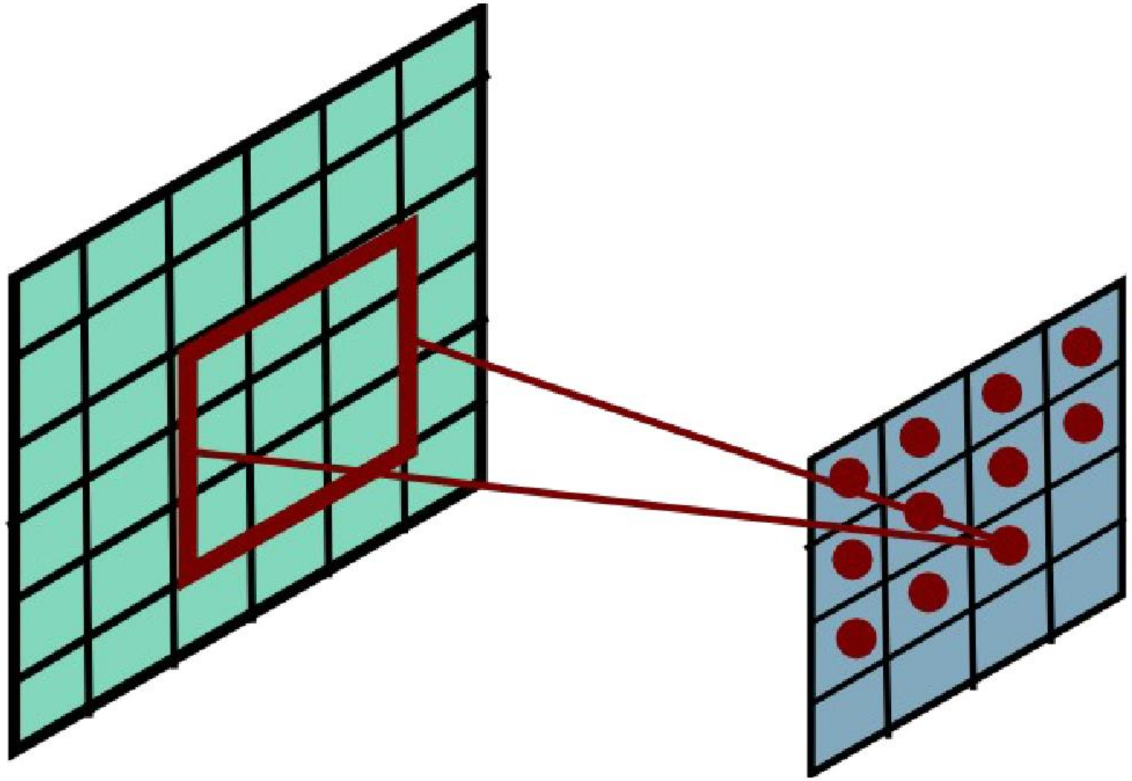
Ranzato

Convolutional Layer



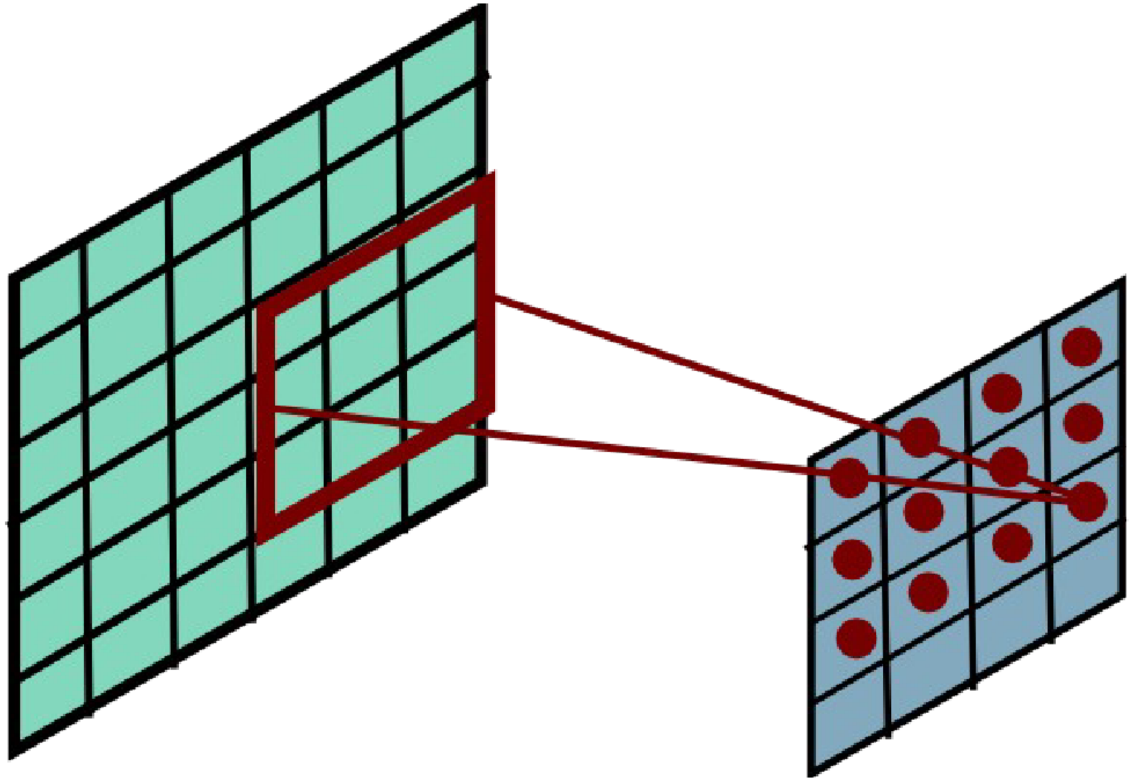
Ranzato

Convolutional Layer



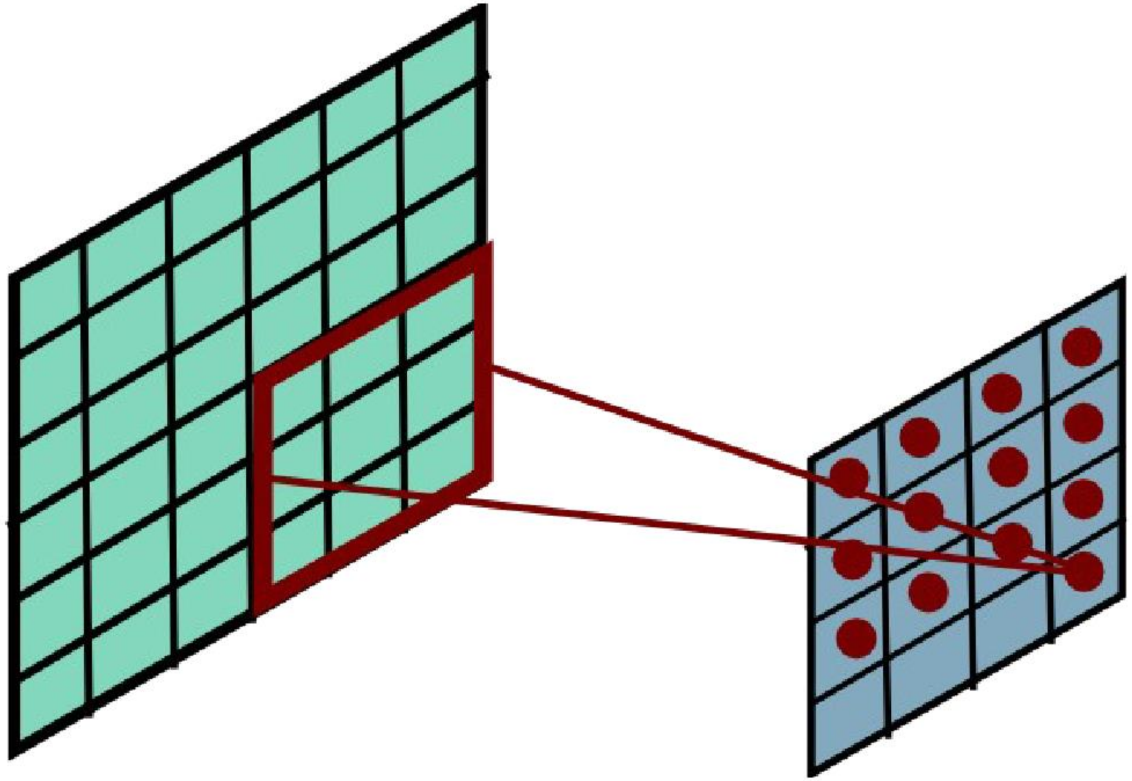
Ranzato

Convolutional Layer



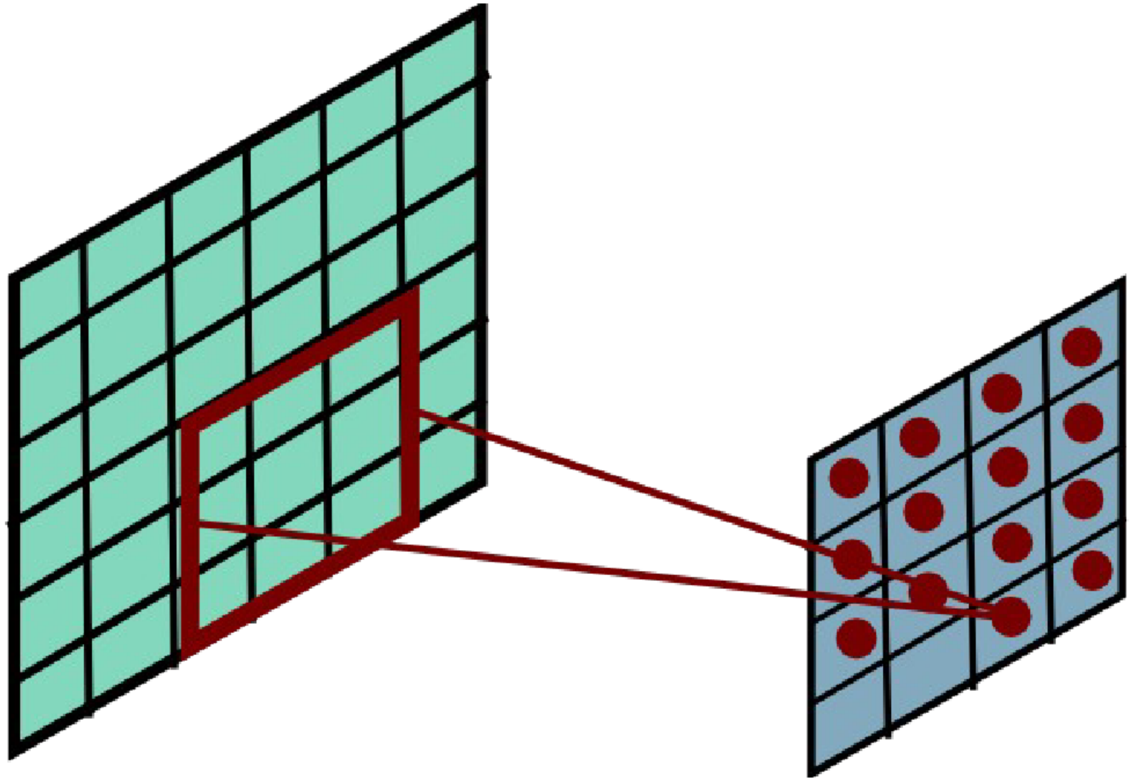
Ranzato

Convolutional Layer



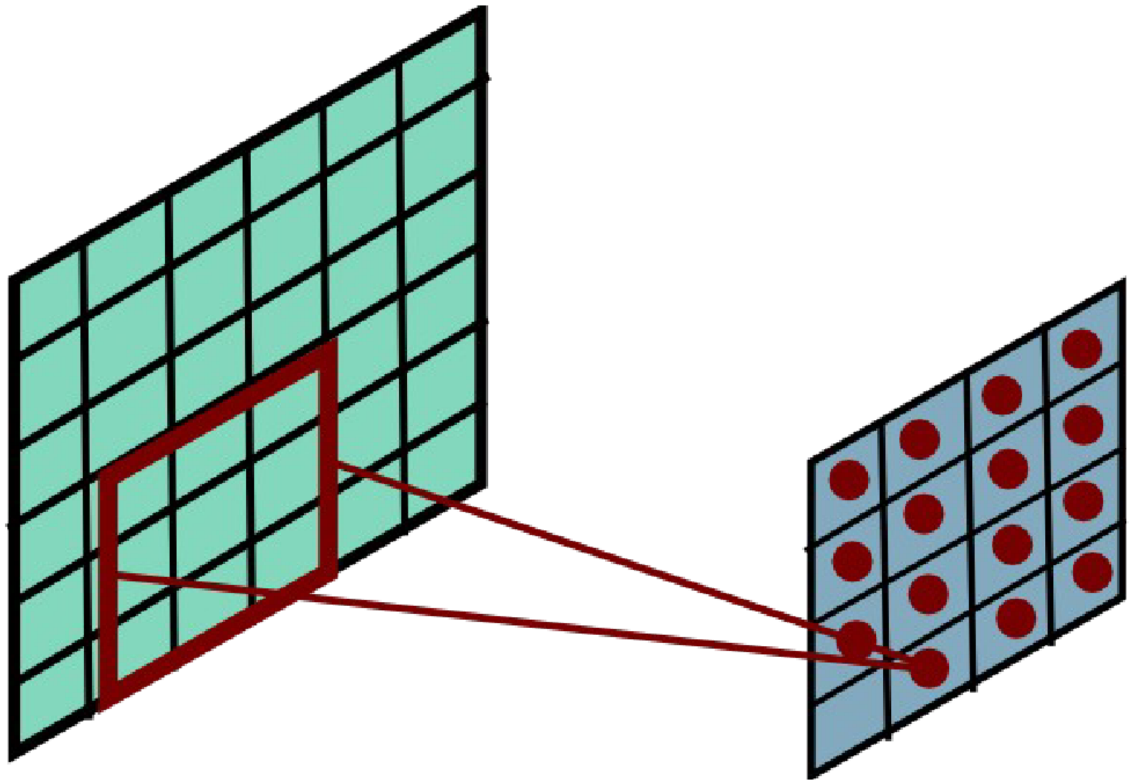
Ranzato

Convolutional Layer



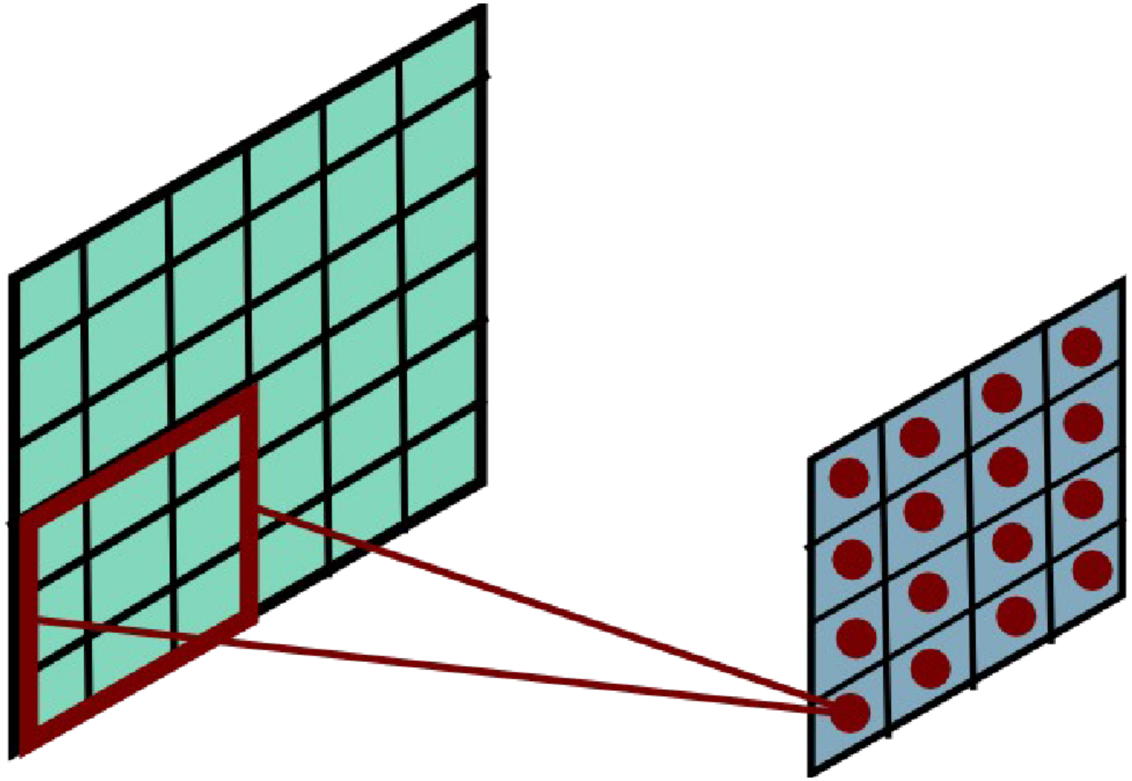
Ranzato

Convolutional Layer



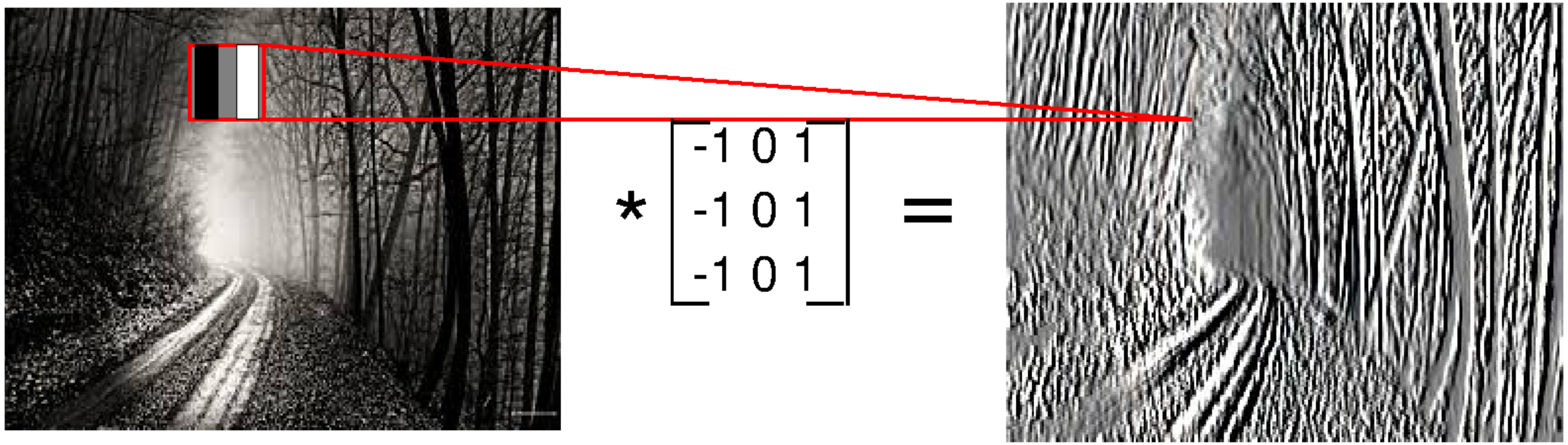
Ranzato

Convolutional Layer



Ranzato

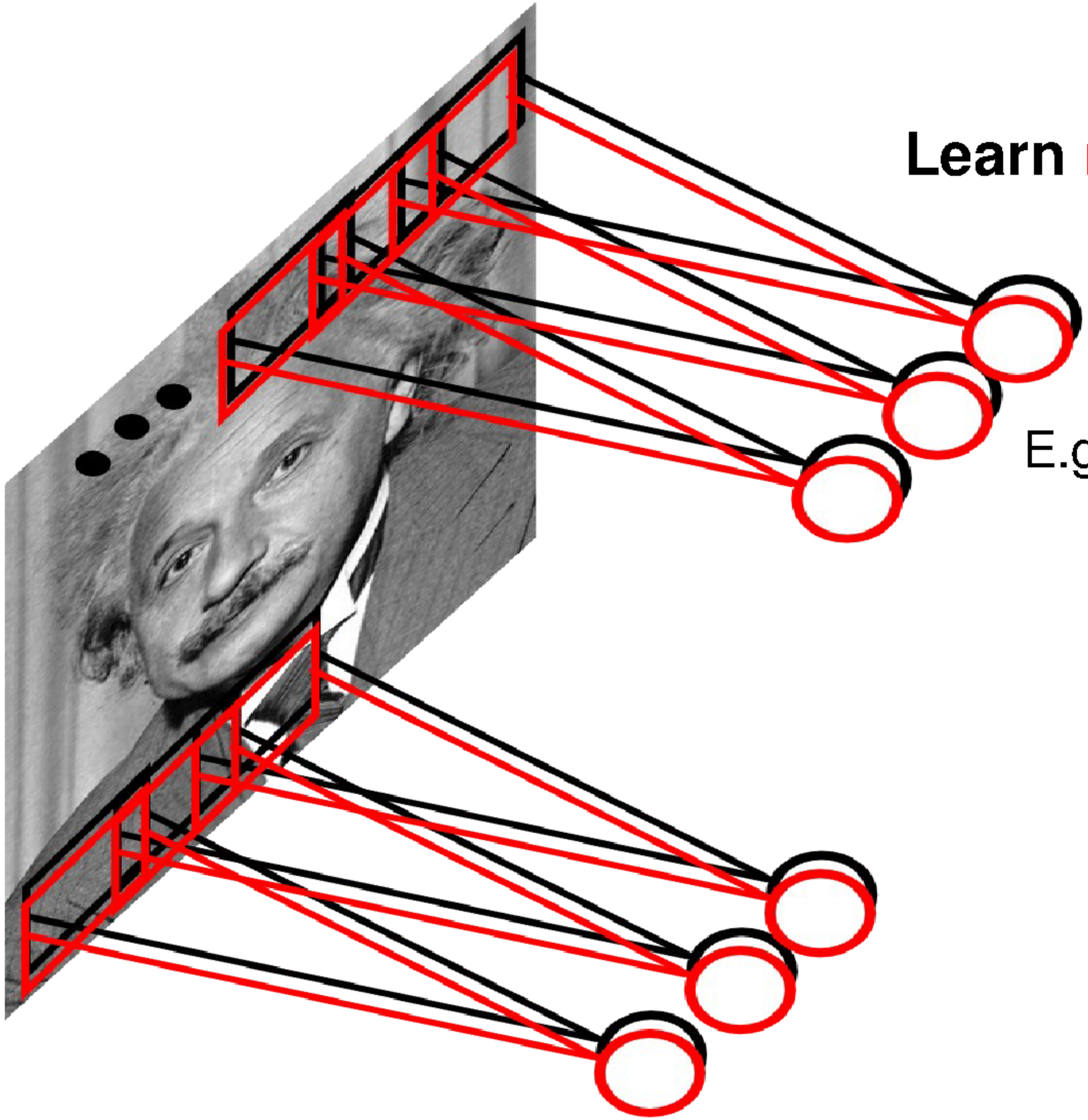
Convolutional Layer



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

Ranzato

Convolutional Layer



Learn multiple filters.

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

Ranzato

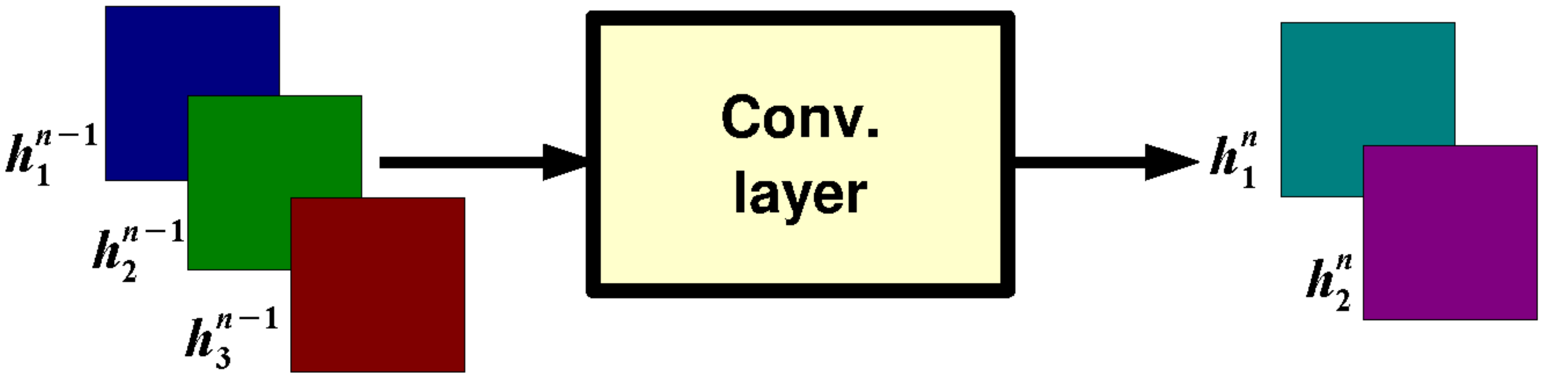
Convolutional Layer

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

output feature map

input feature map

kernel



Ranzato

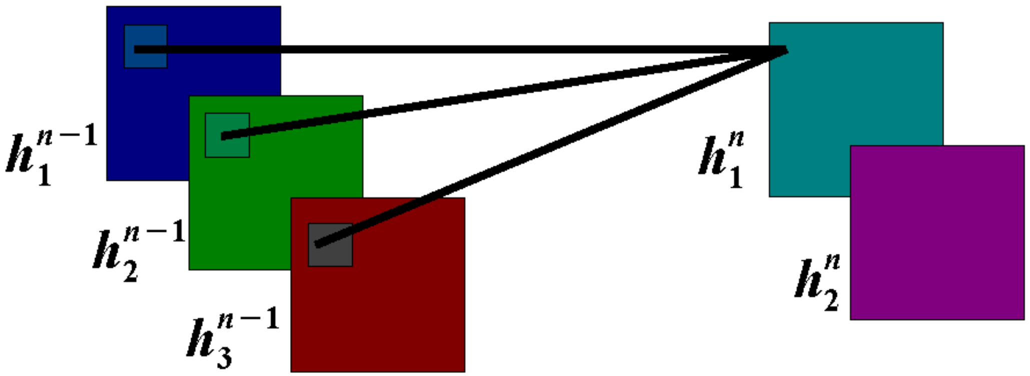
Convolutional Layer

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

output feature map

input feature map

kernel



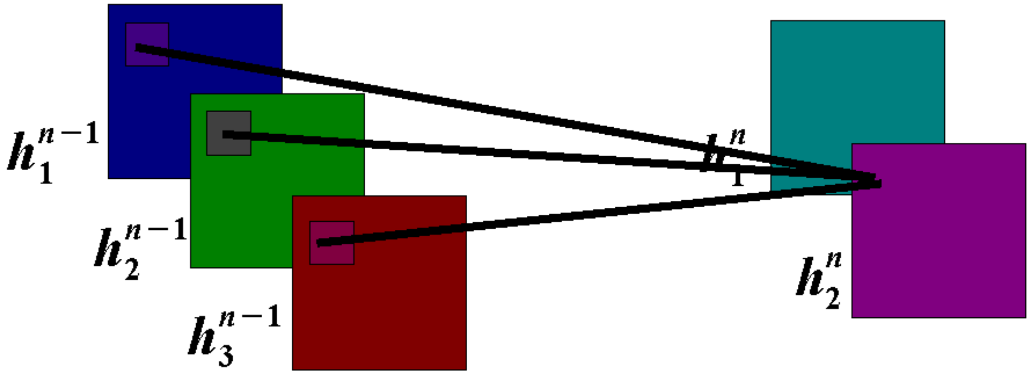
Convolutional Layer

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

output feature map

input feature map

kernel



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 KxK, input has size DxD, stride is 1, and there are M input feature maps and N output feature maps then:
- the input has size M@DxD
- the output has size N@(D-K+1)x(D-K+1)
- the kernels have MxNxKxK coefficients (which have to be learned)
- cost: M*K*K*N*(D-K+1)*(D-K+1)

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

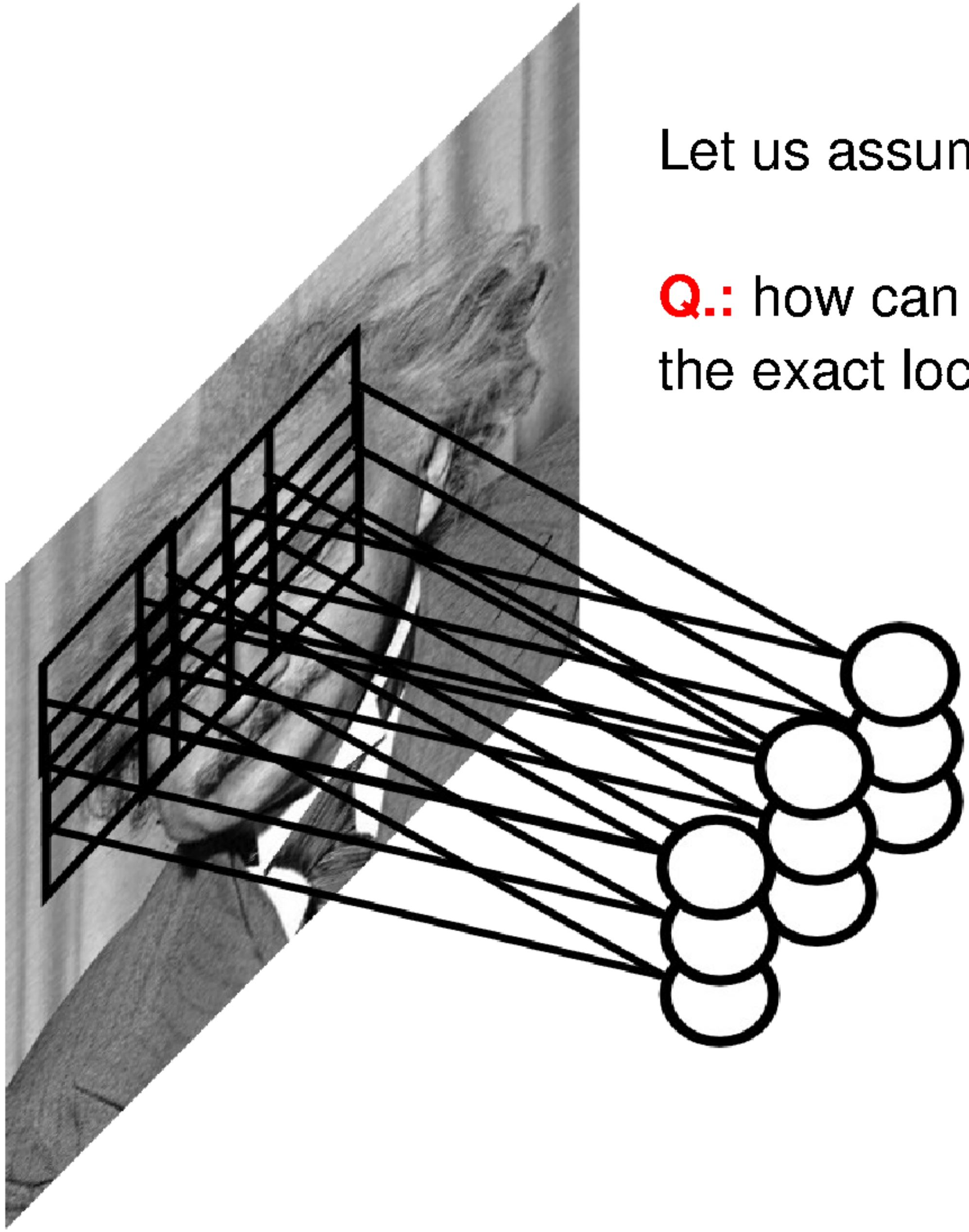
Solution:

- 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.**

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

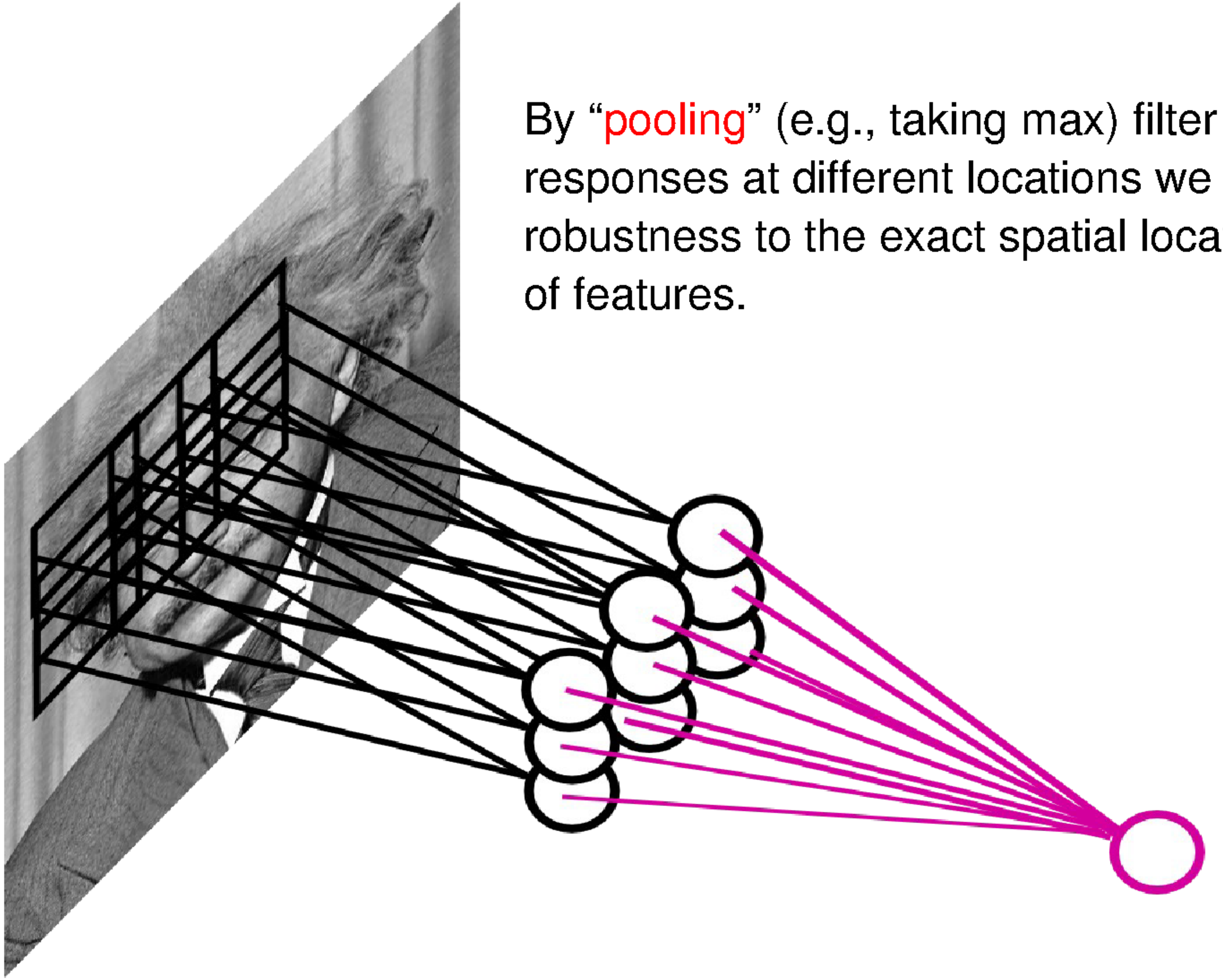
Pooling Layer



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.

Ranzato

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

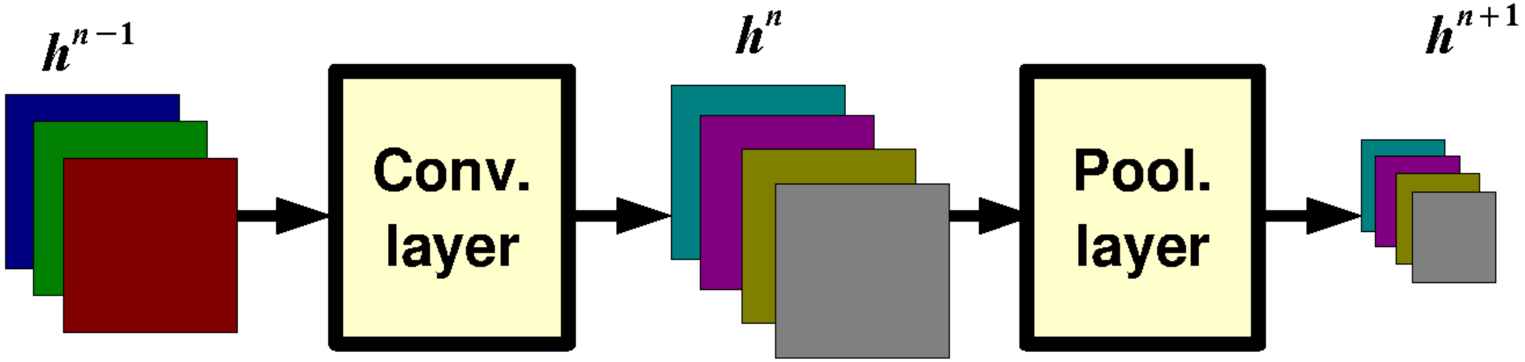
Question: What is the size of the output? What's the computational cost?

Answer: The size of the output depends on the stride between the pools. For instance, if pools do not overlap and have size KxK, and the input has size DxD with M input feature maps, then:
- output is M@(D/K)x(D/K)
- the computational cost is proportional to the size of the input (negligible compared to a convolutional layer)

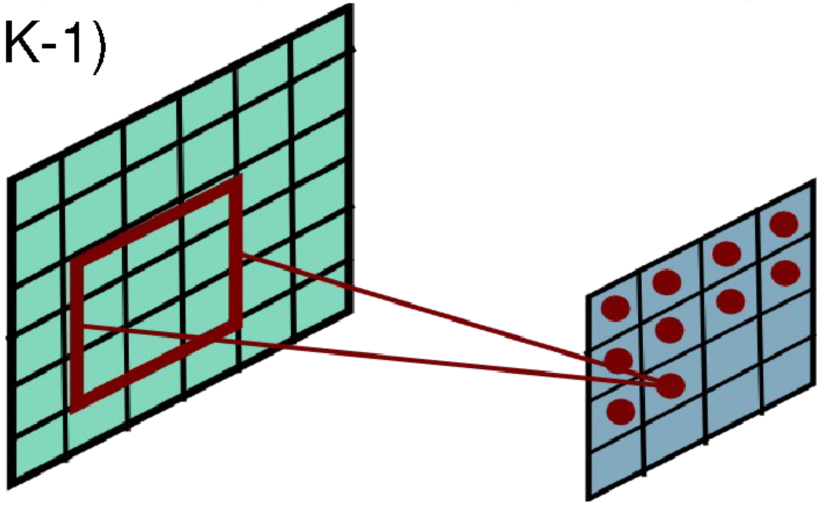
Question: How should I set the size of the pools?

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: Receptive Field Size

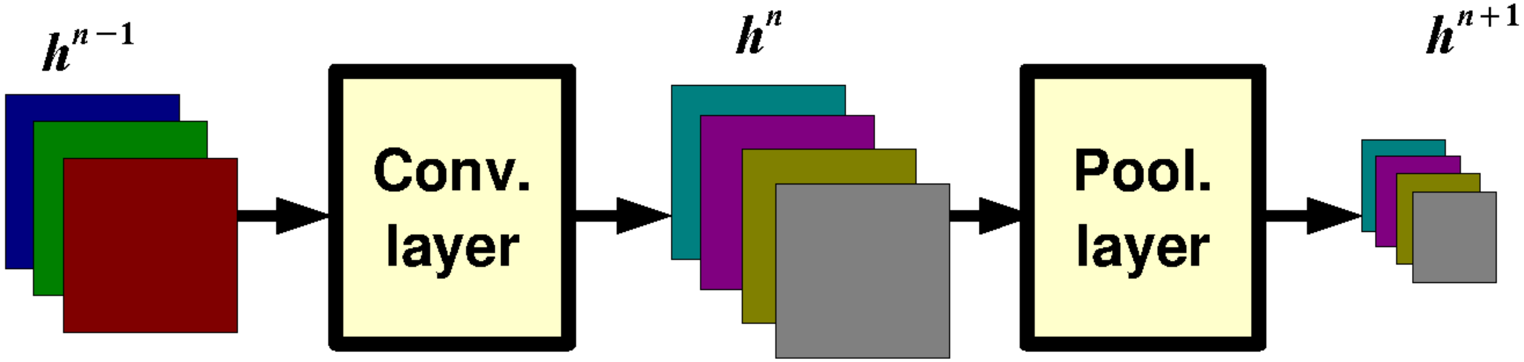


If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)

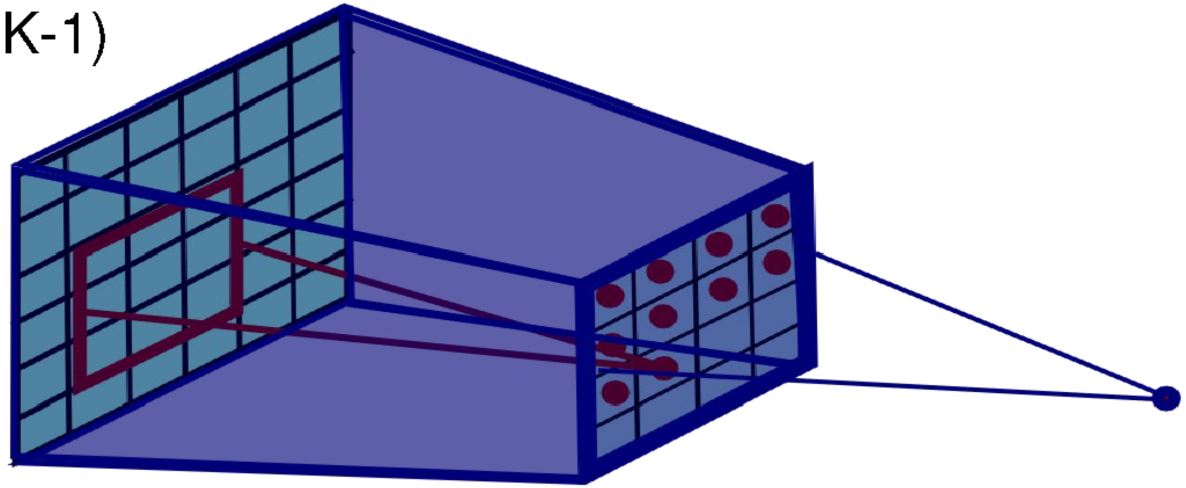


Ranzato

Pooling Layer: Receptive Field Size



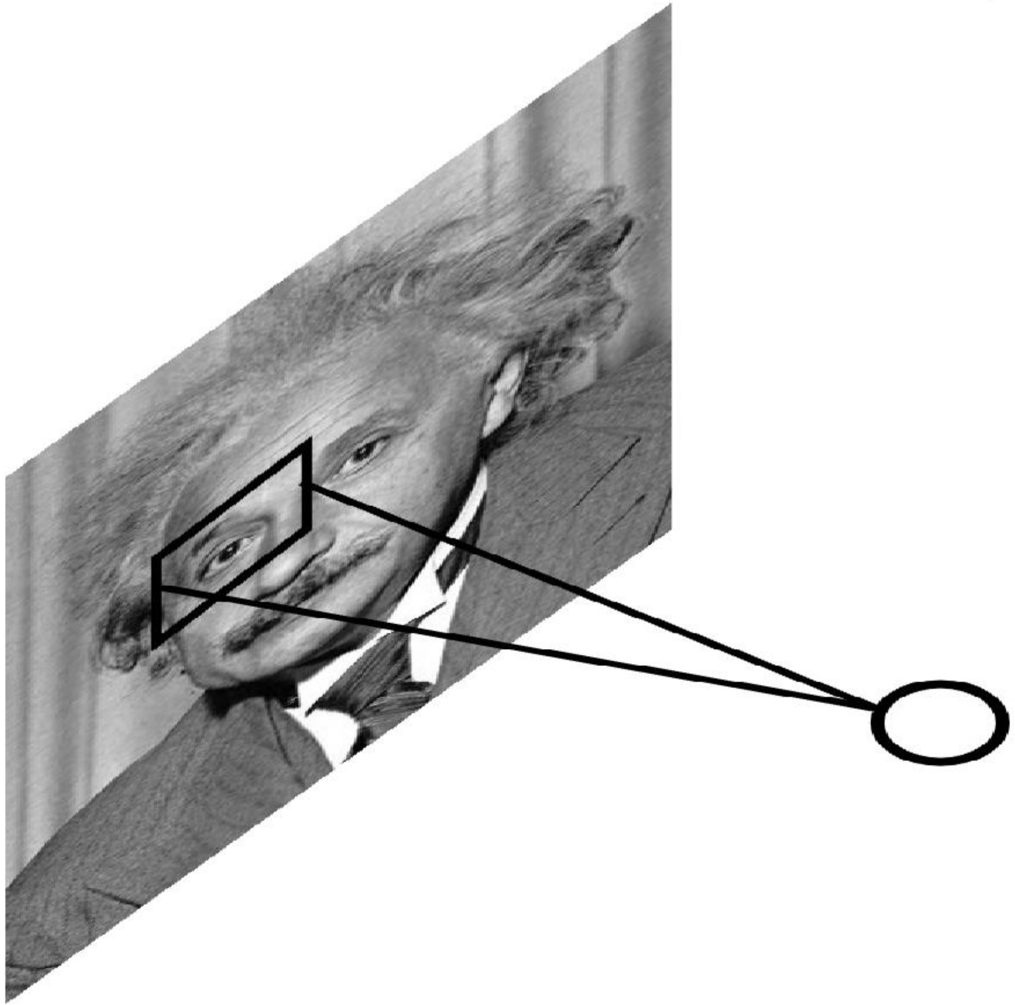
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Ranzato

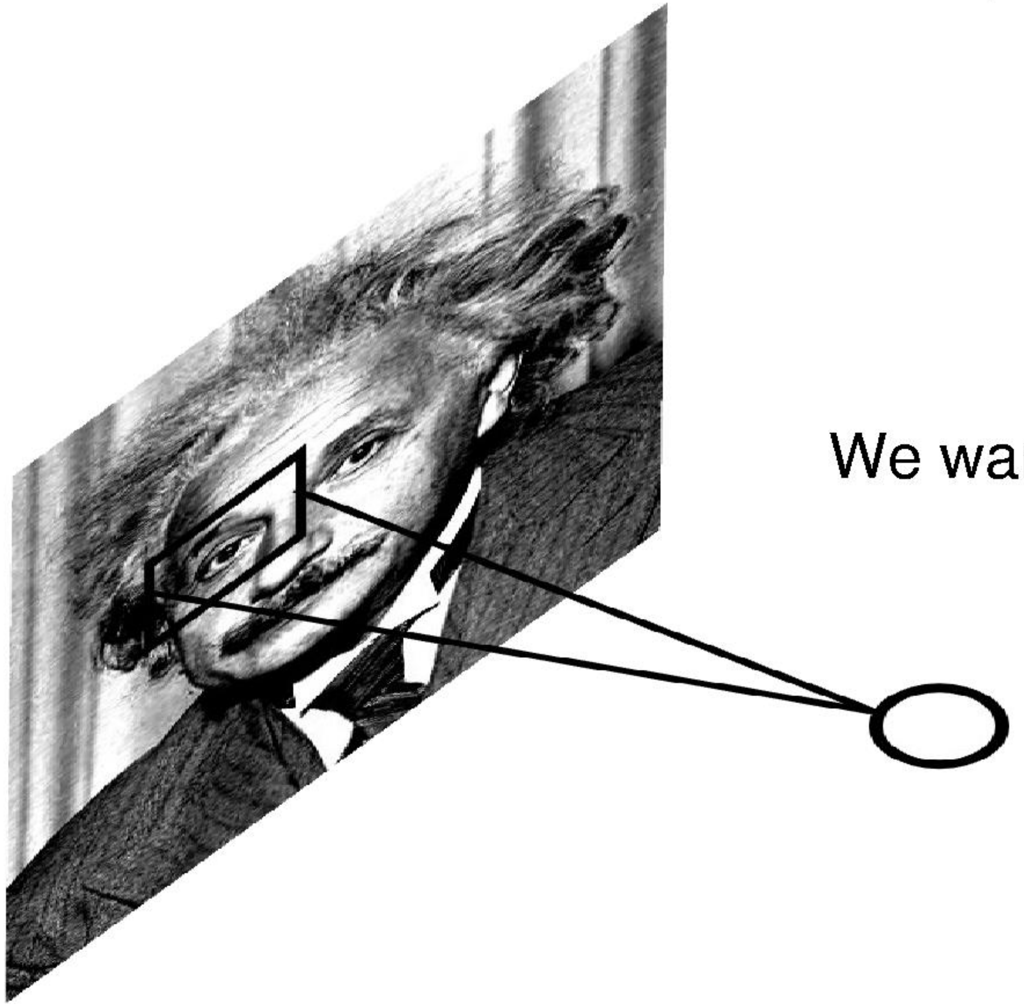
Local Contrast Normalization

$$h^{i+1}(x,y) = \frac{h^{i}(x,y) - m^{i}(N(x,y))}{\sigma^{i}(N(x,y))}$$



Local Contrast Normalization

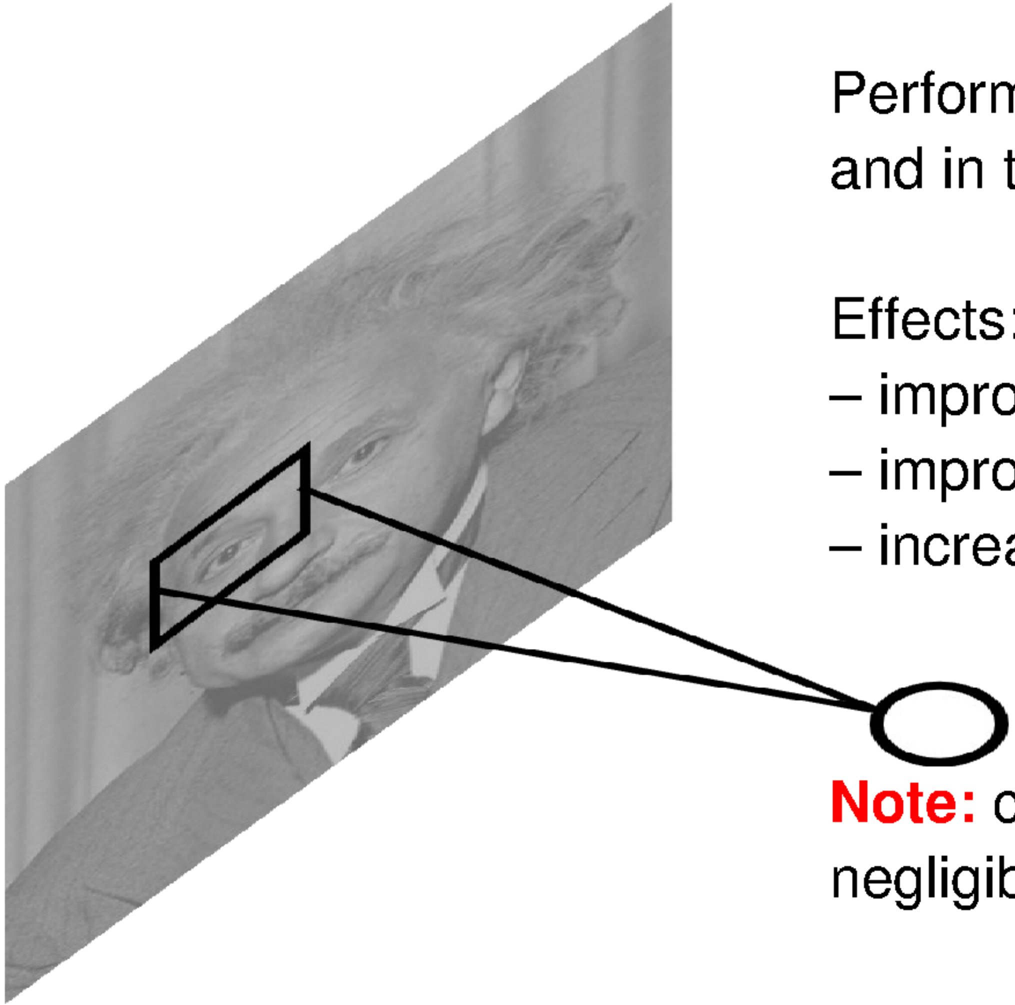
$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{\sigma^i(N(x,y))}$$



We want the same response.

Local Contrast Normalization

$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{\sigma^i(N(x,y))}$$



Performed also across features and in the higher layers..

Effects:

– improves invariance

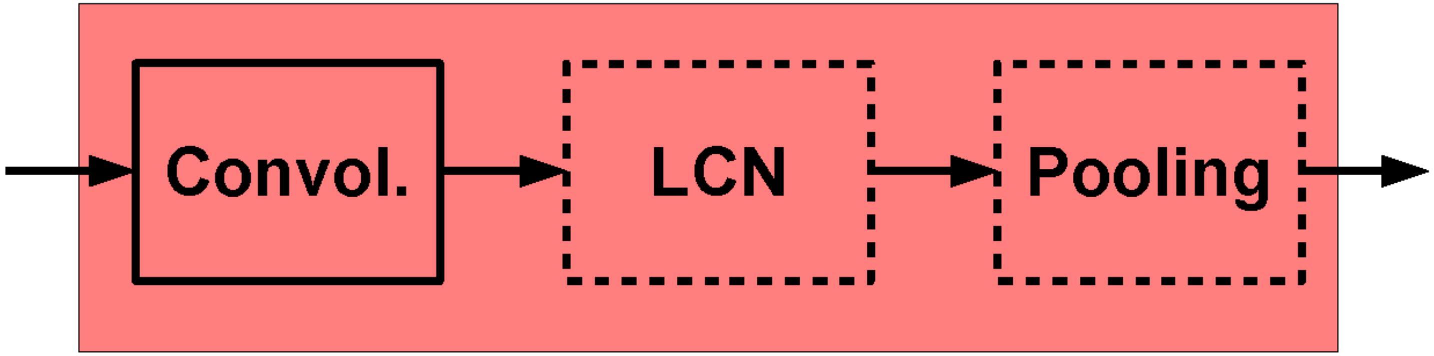
– improves optimization

– increases sparsity

Note: computational cost is negligible w.r.t. conv. layer.

ConvNets: Typical Stage

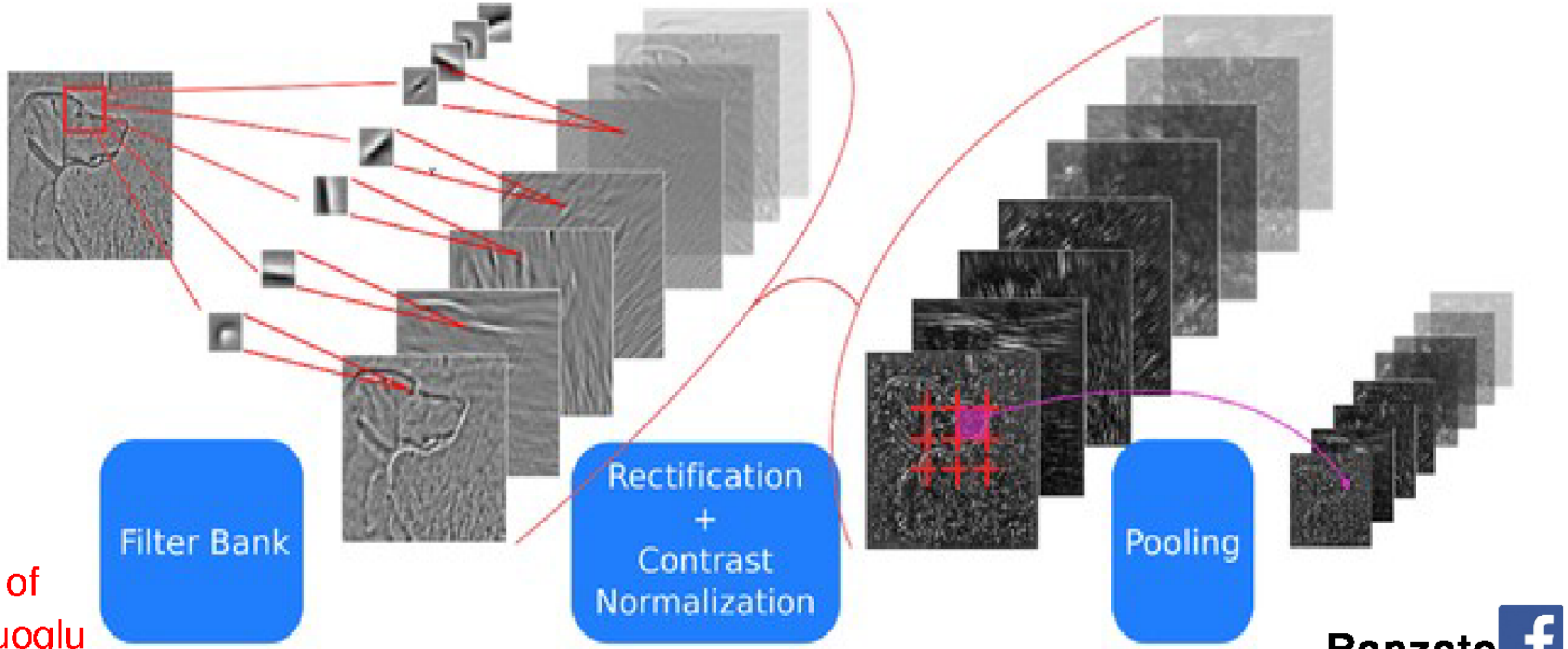
One stage (zoom)



Convol.

LCN

Pooling

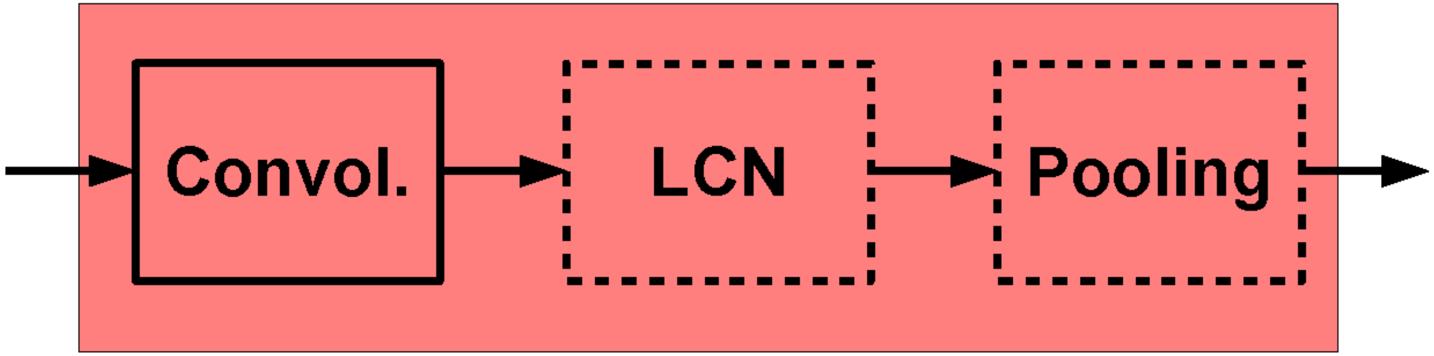


courtesy of
K. Kavukcuoglu

Ranzato

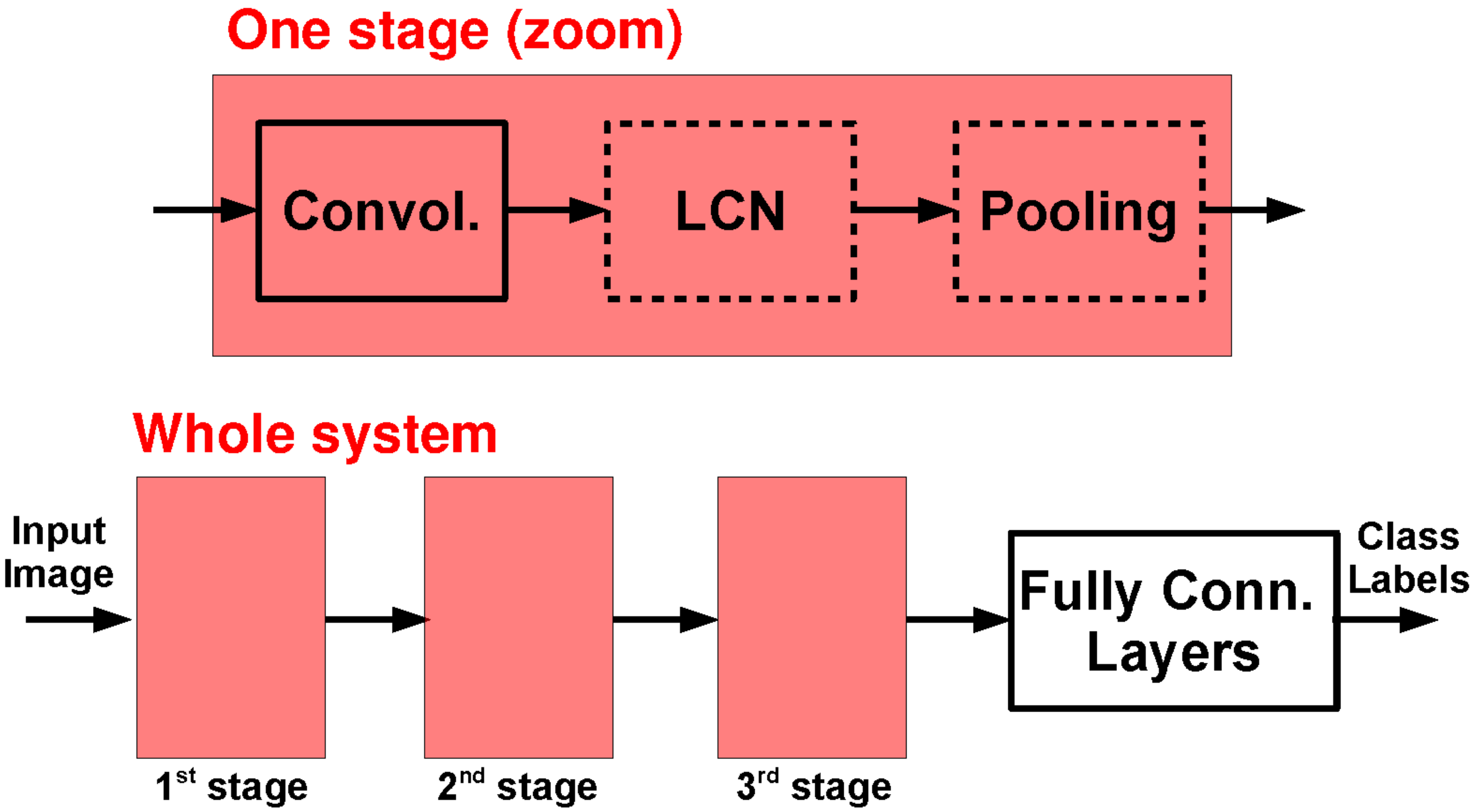
ConvNets: Typical Stage

One stage (zoom)

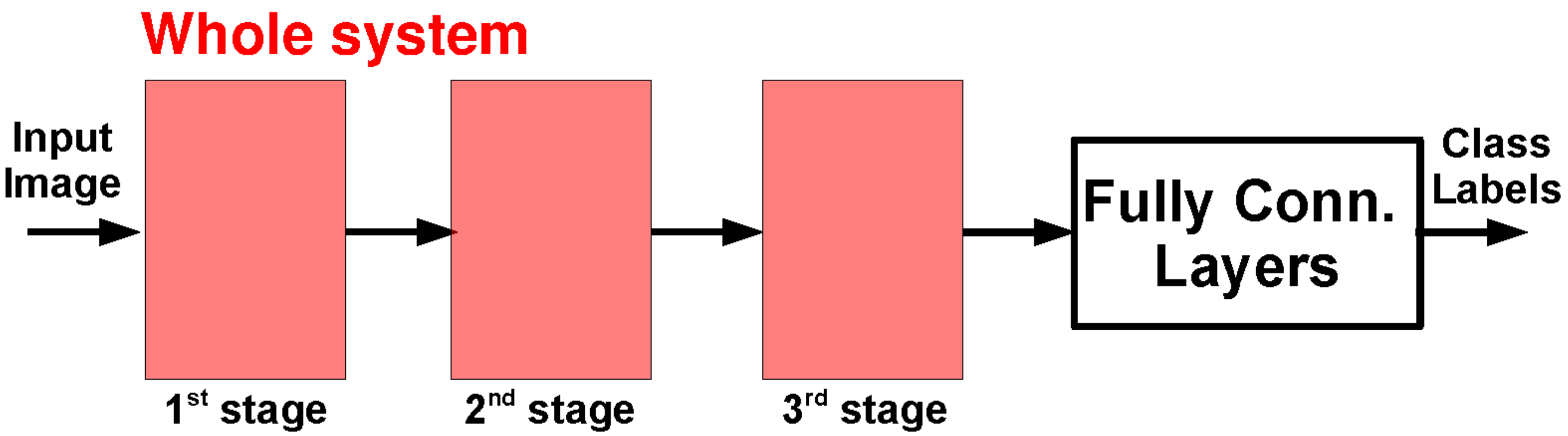


Conceptually similar to: SIFT, HoG, etc.

ConvNets: Typical Architecture



ConvNets: Typical Architecture



Conceptually similar to:

SIFT → K-Means → Pyramid Pooling → SVM

Lazebnik et al. "...Spatial Pyramid Matching..." CVPR 2006

SIFT → Fisher Vect. → Pooling → SVM

Sanchez et al. "Image classifcation with F.V.: Theory and practice" IJCV 2012