Large-Scale Visual Recognition With Deep Learning

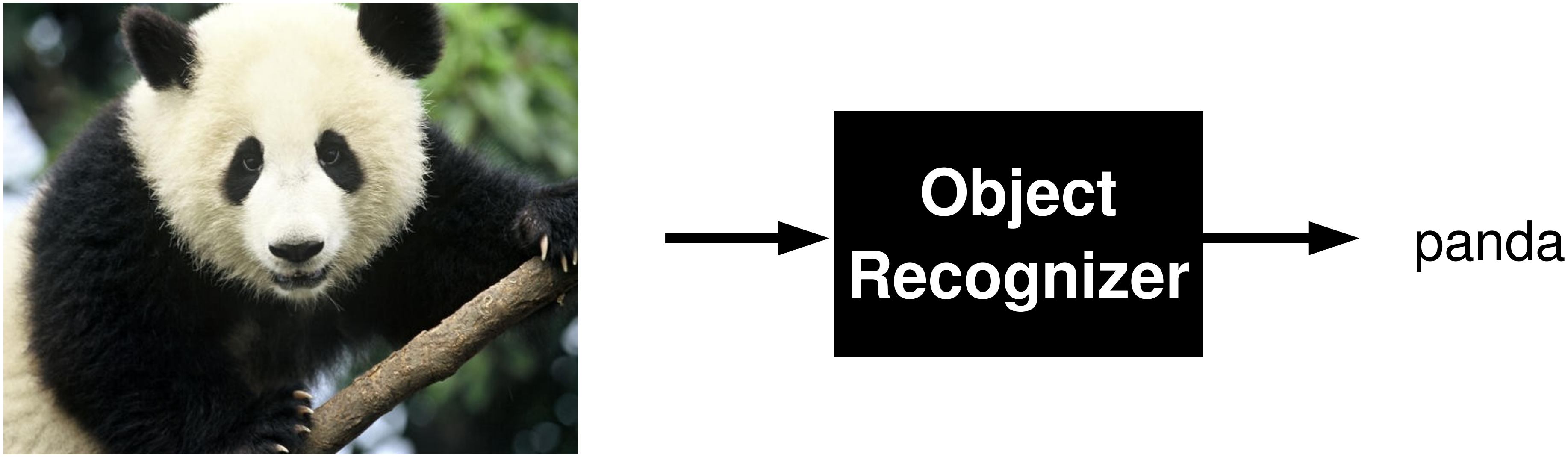
Marc'Aurelio Ranzato

Google

ranzato@google.com
www.cs.toronto.edu/~ranzato

Sunday 23 June 2013

Why Is Recognition Hard?

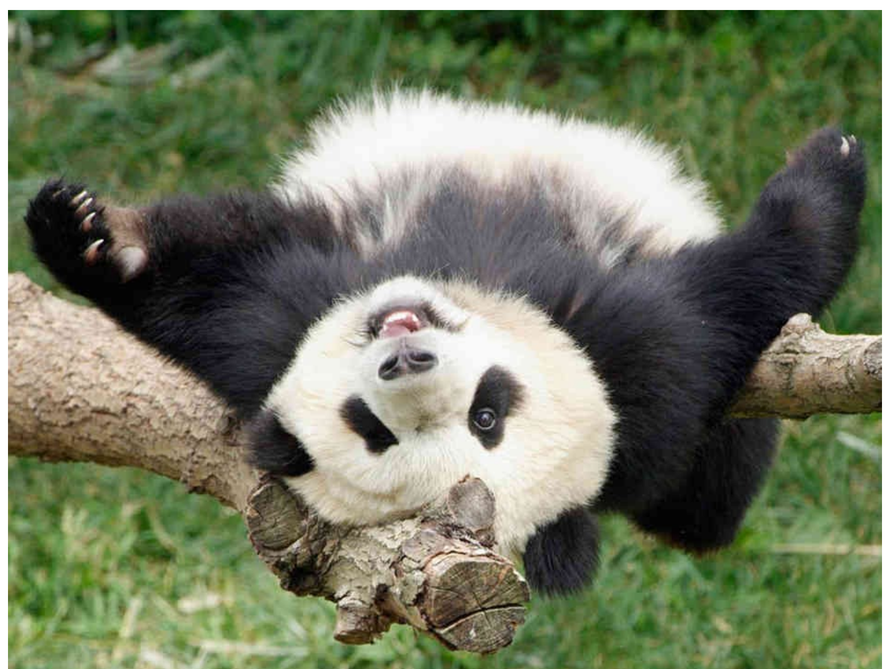


Why Is Recognition Hard?

Object Recognizer

panda

Pose



Why Is Recognition Hard?

Object Recognizer

panda

Occlusion

Why Is Recognition Hard?

→ **Object Recognizer** → panda

Multiple objects

Why Is Recognition Hard?

→ **Object Recognizer** → panda

Inter-class similarity

Ideal Features

Ideal Feature Extractor

- window, top-left
- clock, top-middle
- shelf, left
- drawing,middle
- statue, bottom left
- …
- hat, bottom right

Q.: What objects are in the image? Where is the clock?
What is on the top of the table? ...

Ideal Features Are Non-Linear

I_1

→ **Ideal Feature Extractor** →

- club, **angle = 90**
- man, frontal pose

...

? → **Ideal Feature Extractor** →

- club, **angle = 270**
- man, frontal pose

...

I_2

→ **Ideal Feature Extractor** →

- club, **angle = 360**
- man, side pose

...

Ideal Features Are Non-Linear

I_1

→ **Ideal Feature Extractor** →

- club, **angle = 90**
- man, frontal pose

...

INPUT IS NOT THE AVERAGE!

→ **Ideal Feature Extractor** →

- club, **angle = 270**
- man, frontal pose

...

I_2

→ **Ideal Feature Extractor** →

- club, **angle = 360**
- man, side pose

...

Ideal Features Are Non-Linear

I_1

Ideal Feature Extractor

- club, **angle = 90**
- man, frontal pose

...

Ideal Feature Extractor

- club, **angle = 270**
- man, frontal pose

...

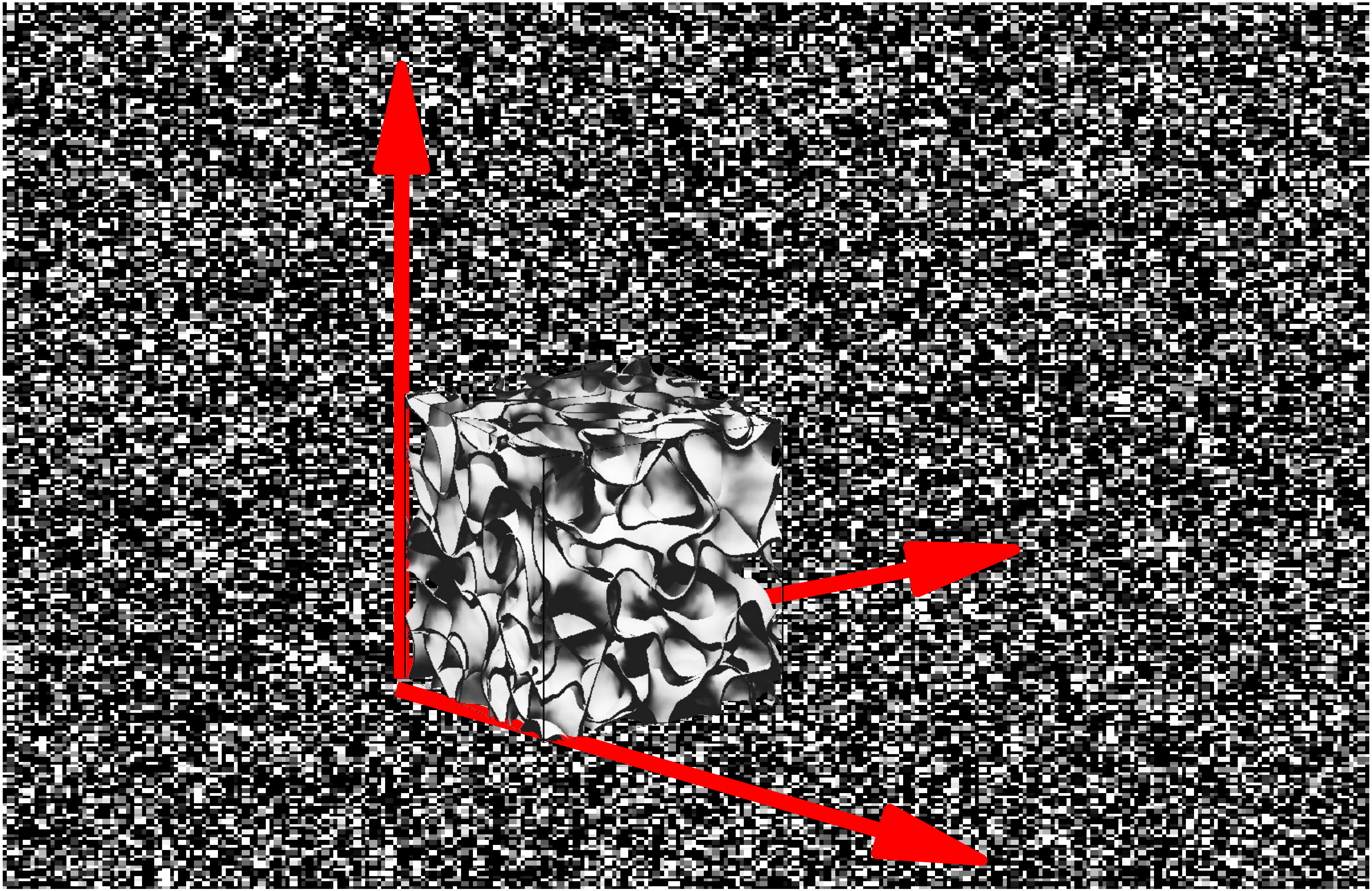
I_2

Ideal Feature Extractor

- club, **angle = 360**
- man, side pose

...

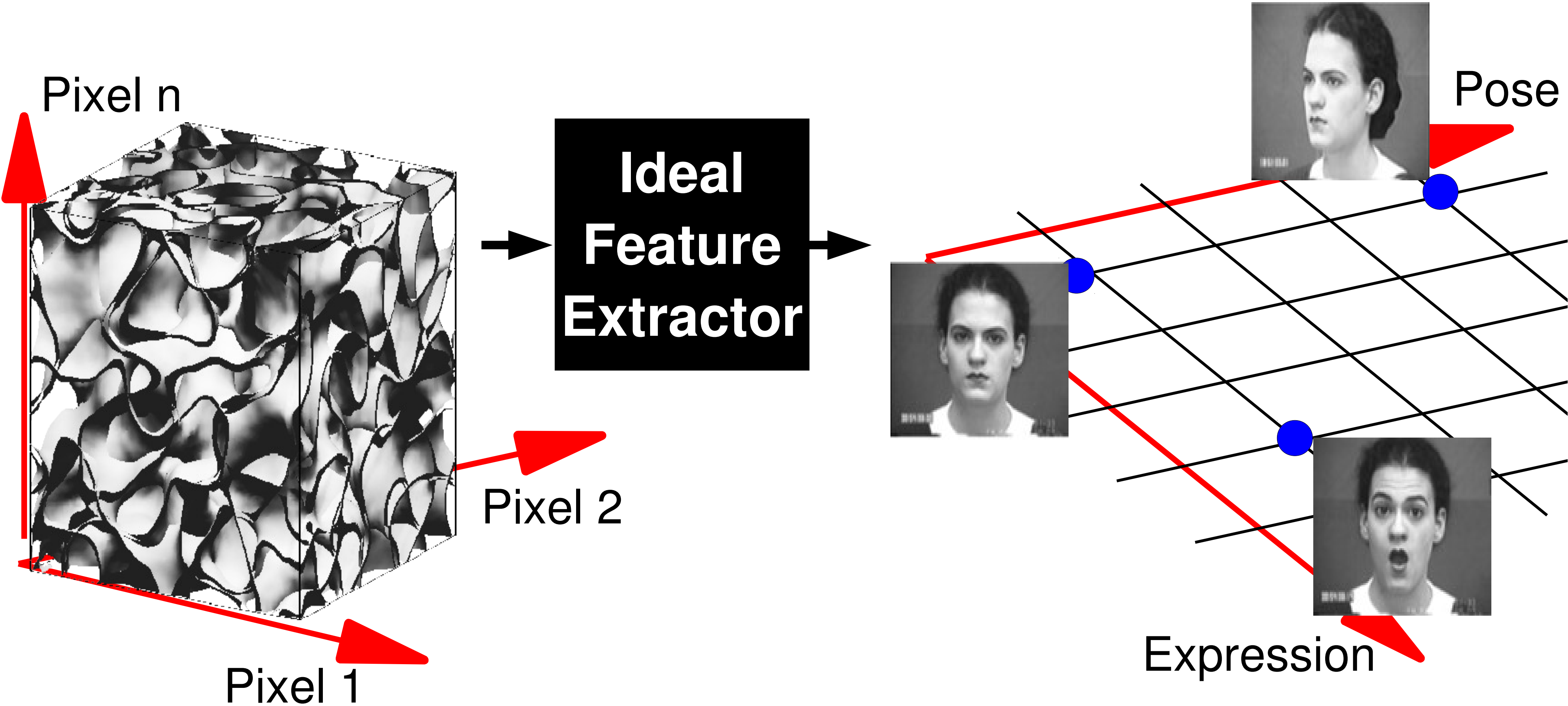
The Manifold of Natural Images



The Manifold of Natural Images

We need to linearize the manifold: learn non-linear features!

Ideal Feature Extraction



Learning Non-Linear Features

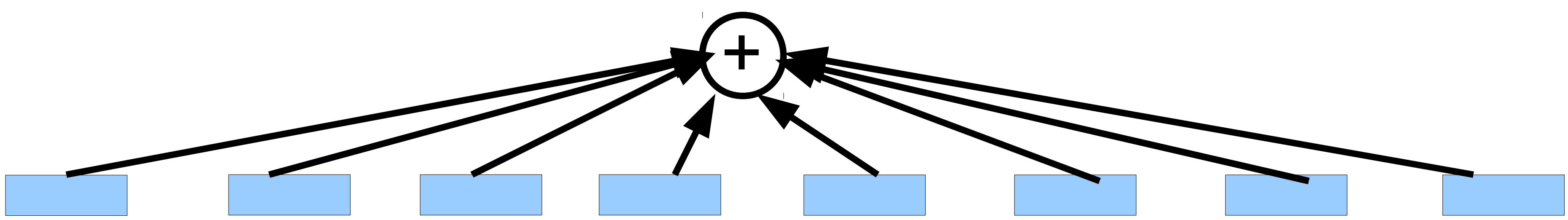
$$\longrightarrow \boxed{f(x;\theta)} \longrightarrow \text{features}$$

Q.: which class of non-linear functions shall we consider?

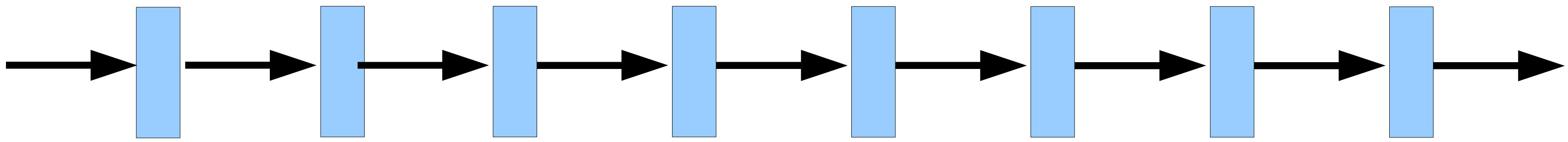
Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

Proposal #1: linear combination $f(x) \approx \sum_j g_j$



Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

Proposal #1: linear combination $f(x) \approx \sum_j g_j$

- Kernel learning
- Boosting
- ...

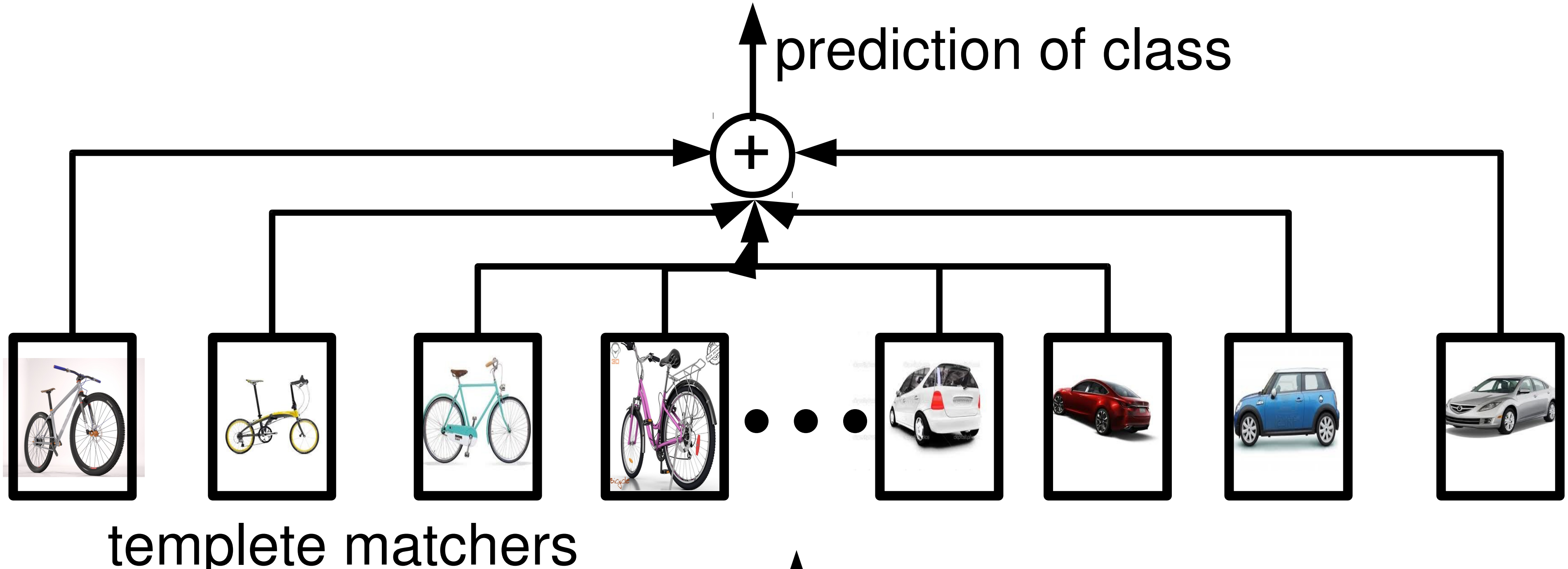
Shallow

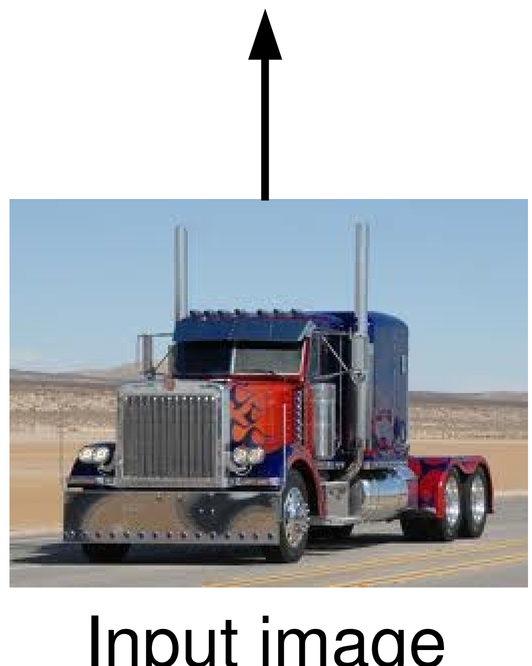
Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x) \ldots))$

- Deep learning
- Scattering networks (wavelet cascade)
- S.C. Zhou & D. Mumford "grammar"

Deep

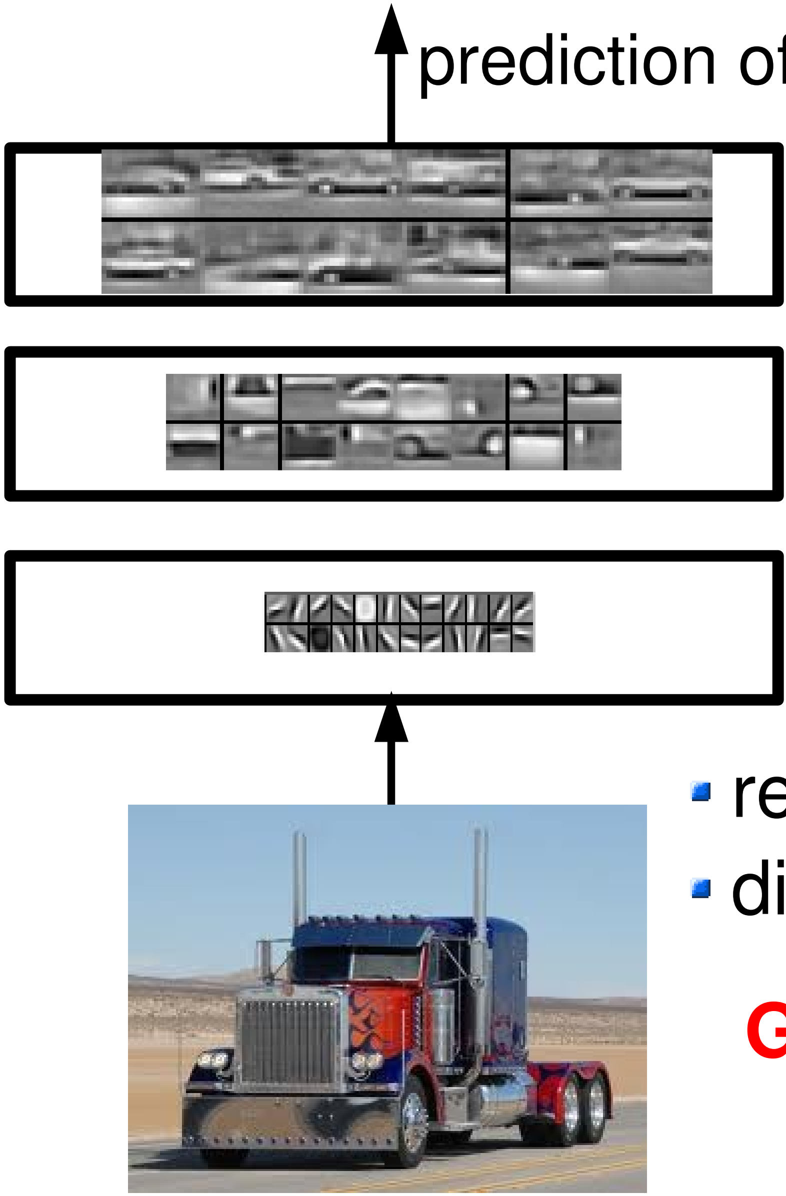
Linear Combination





BAD: it may require an exponential nr. of templates!!!

Composition



Input image

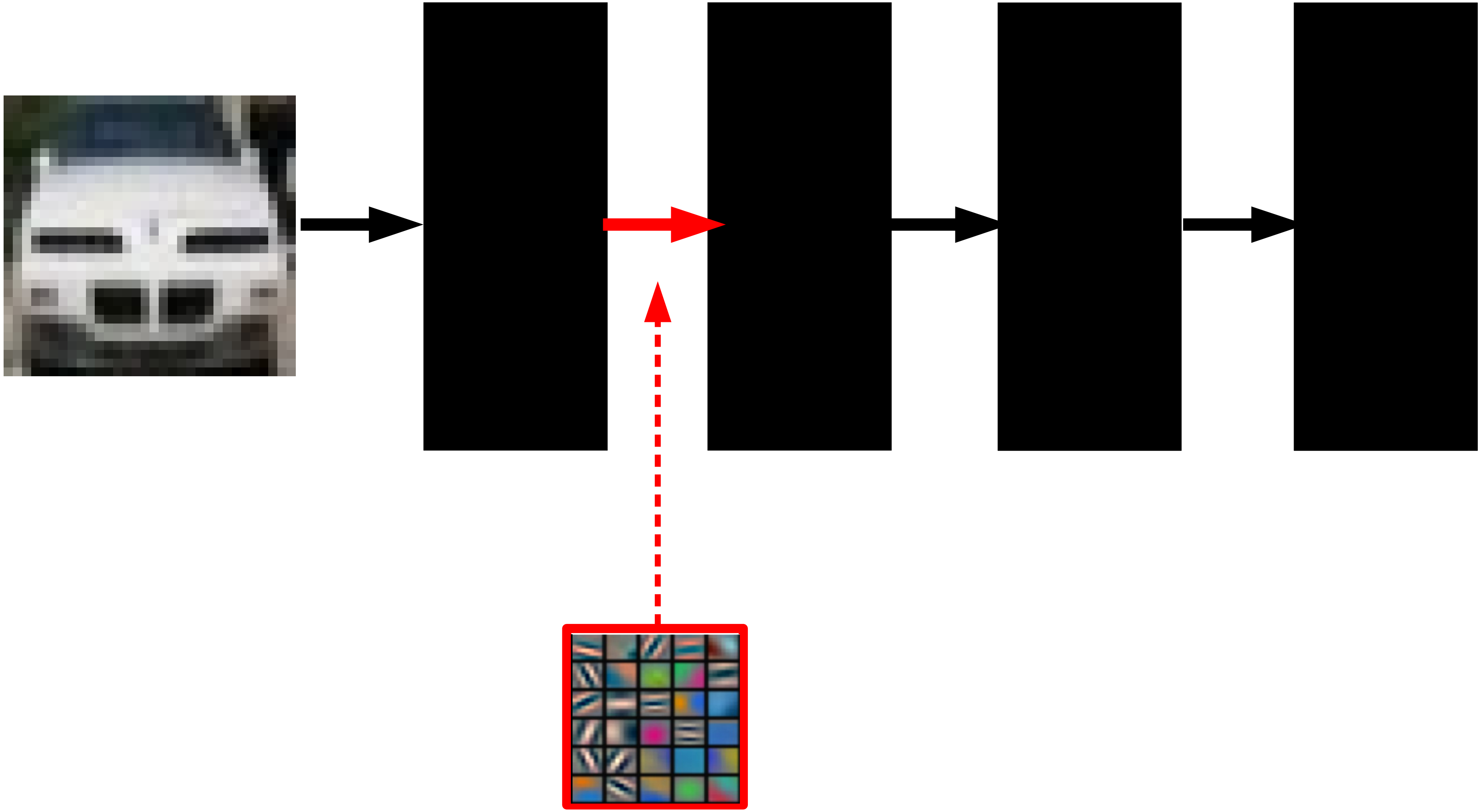
- reuse of intermediate parts
- distributed representations

GOOD: (exponentially) more efficient

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

The Big Advantage of Deep Learning

Efficiency: intermediate concepts can be re-used

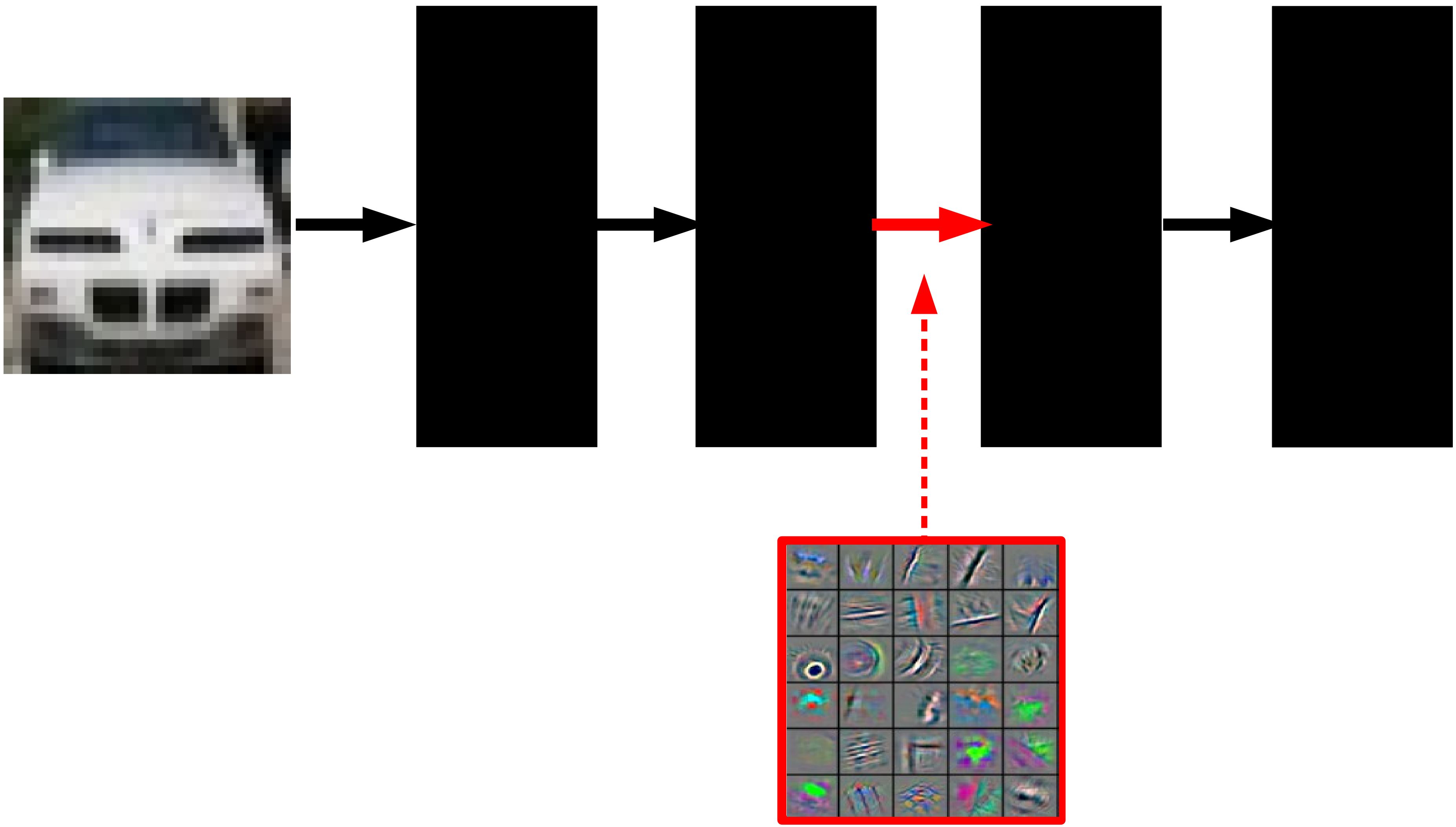


Zeiler, Fergus 2013

Ranzato

The Big Advantage of Deep Learning

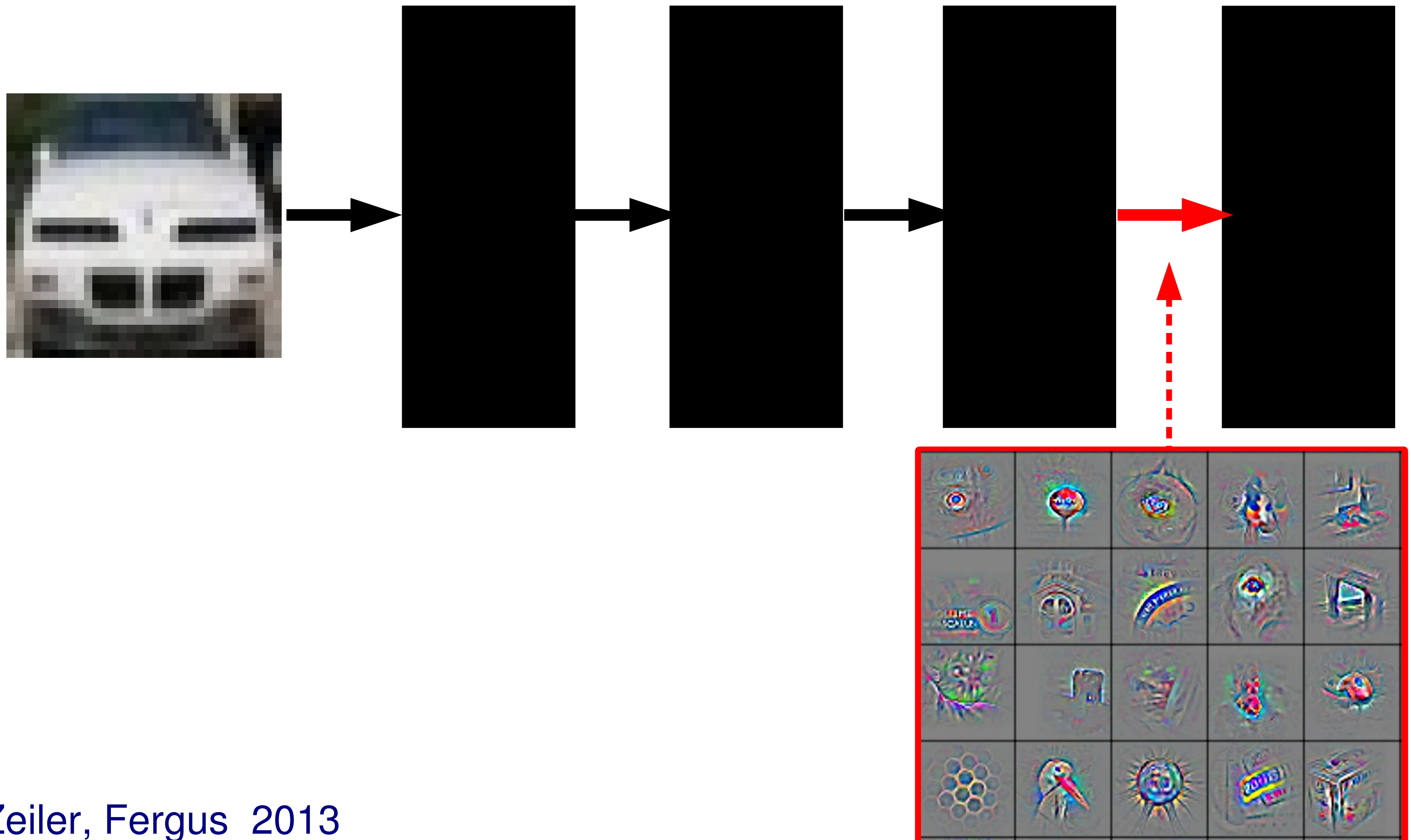
Efficiency: intermediate concepts can be re-used



Zeiler, Fergus 2013

The Big Advantage of Deep Learning

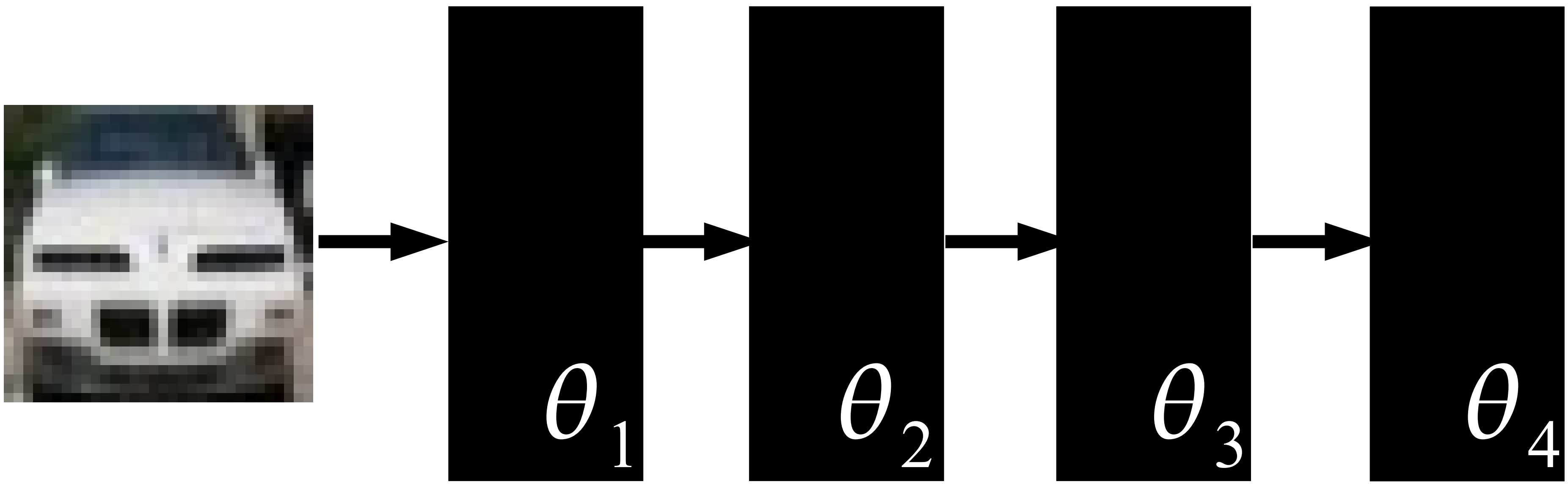
Efficiency: intermediate concepts can be re-used



Zeiler, Fergus 2013

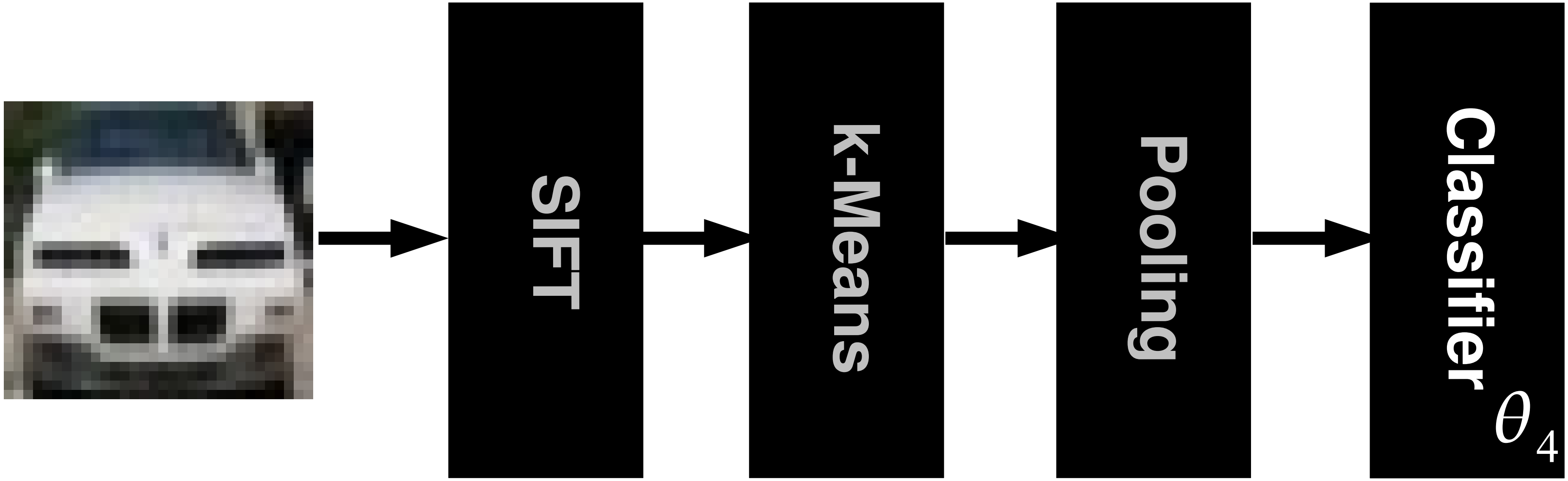
A Potential Problem with Deep Learning

Optimization is difficult: non-convex, non-linear system



A Potential Problem with Deep Learning

Optimization is difficult: non-convex, non-linear system

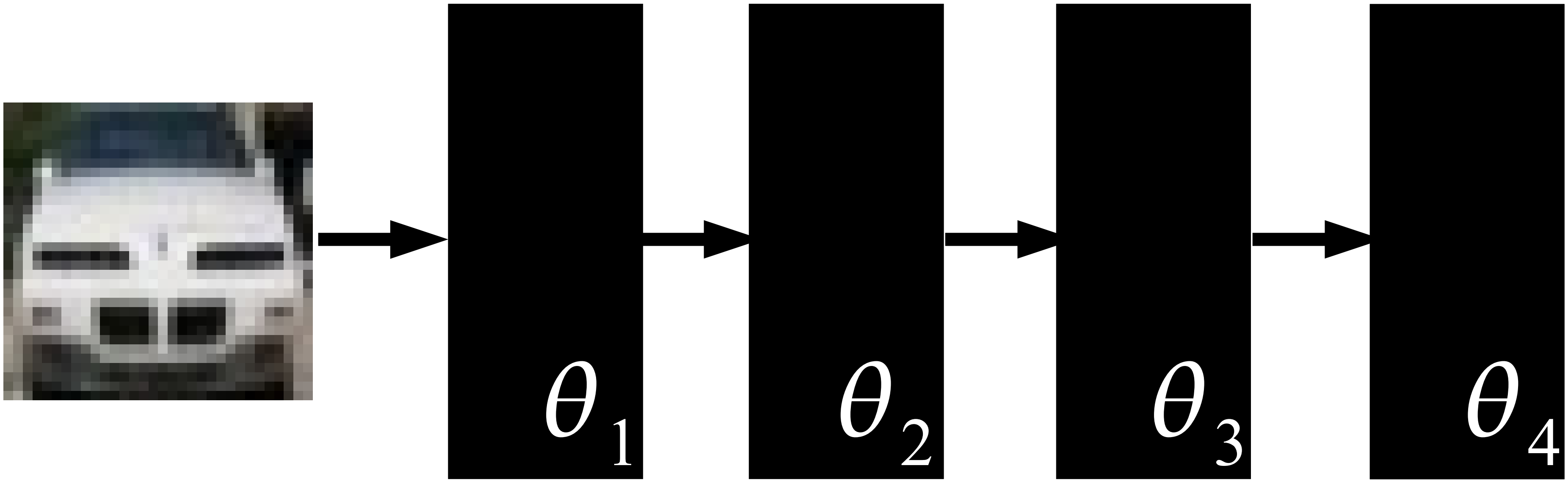


Solution #1: freeze first N-1 layer (engineer the features)

It makes it shallow!

A Potential Problem with Deep Learning

Optimization is difficult: non-convex, non-linear system



Solution #2: live with it!

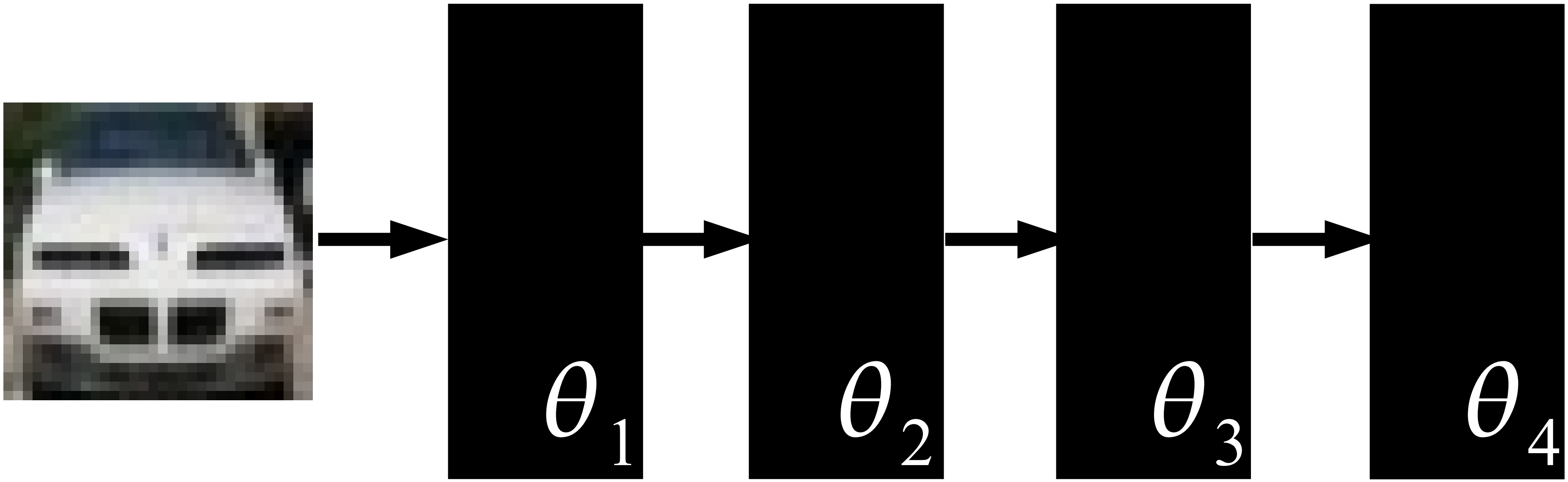
It will converge to a local minimum.

It is much more powerful!!

Given lots of data, engineer less and learn more!!

Deep Learning in Practice

Optimization is easy, need to know a few tricks of the trade.



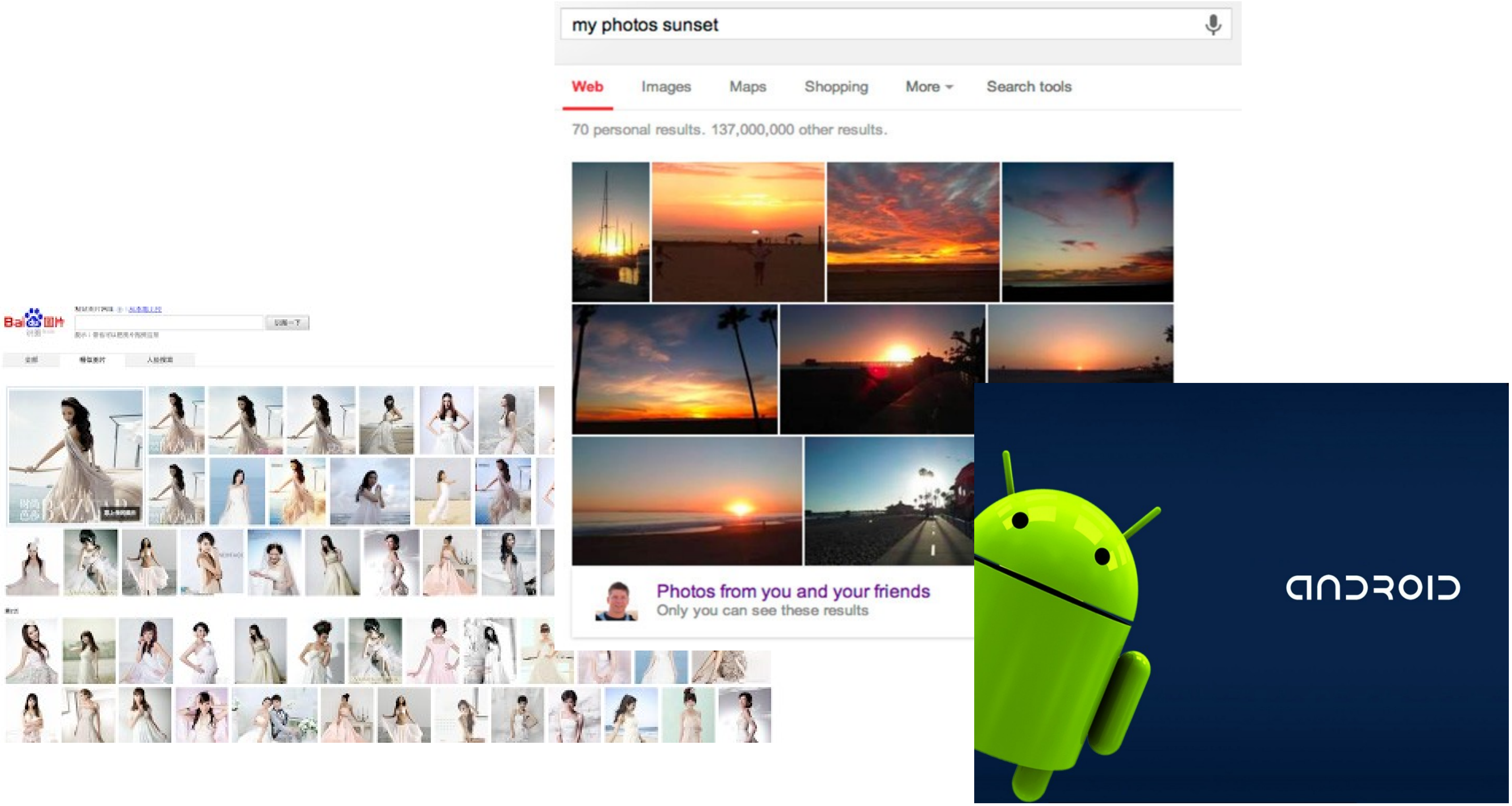
Q: What's the feature extractor? And what's the classifier?

A: No distinction, end-to-end learning!

Ranzato

Deep Learning in Practice

It works very well in practice:



KEY IDEAS: WHY DEEP LEARNING

- We need non-linear system
- We need to learn it from data
- Build feature hierarchies (function composition)
- End-to-end learning

Outline

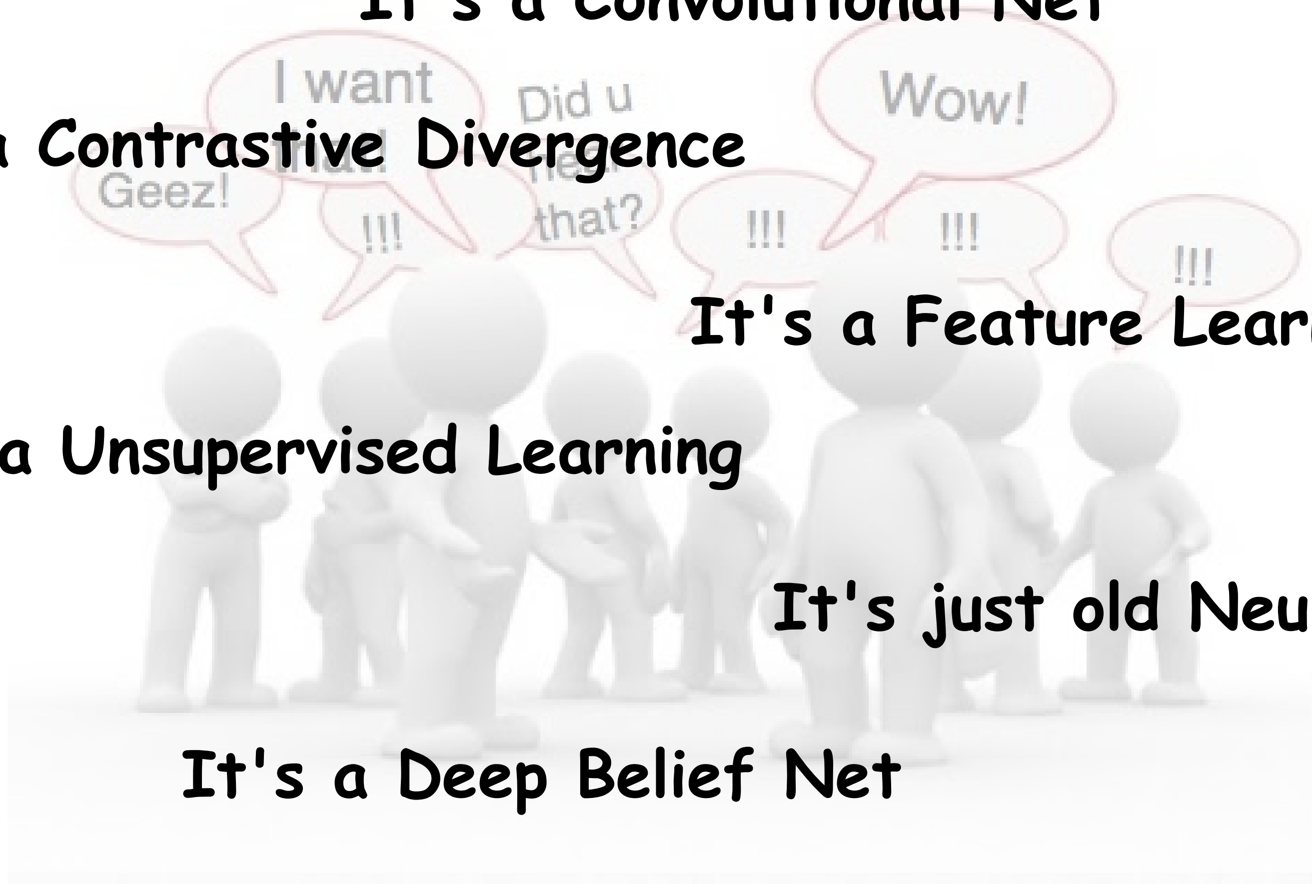
- Motivation
- Deep Learning: The Big Picture
- From neural nets to convolutional nets
- Applications
- A practical guide

Outline

- Motivation
- Deep Learning: The Big Picture
- From neural nets to convolutional nets
- Applications
- A practical guide

What Is Deep Learning?

Buzz Words



It's a Convolutional Net

It's a Contrastive Divergence

It's a Feature Learning

It's a Unsupervised Learning

It's just old Neural Nets

It's a Deep Belief Net

(My) Definition

A Deep Learning method is: a method which makes predictions by using a sequence of non-linear processing stages. The resulting intermediate representations can be interpreted as feature hierarchies and the whole system is jointly learned from data.

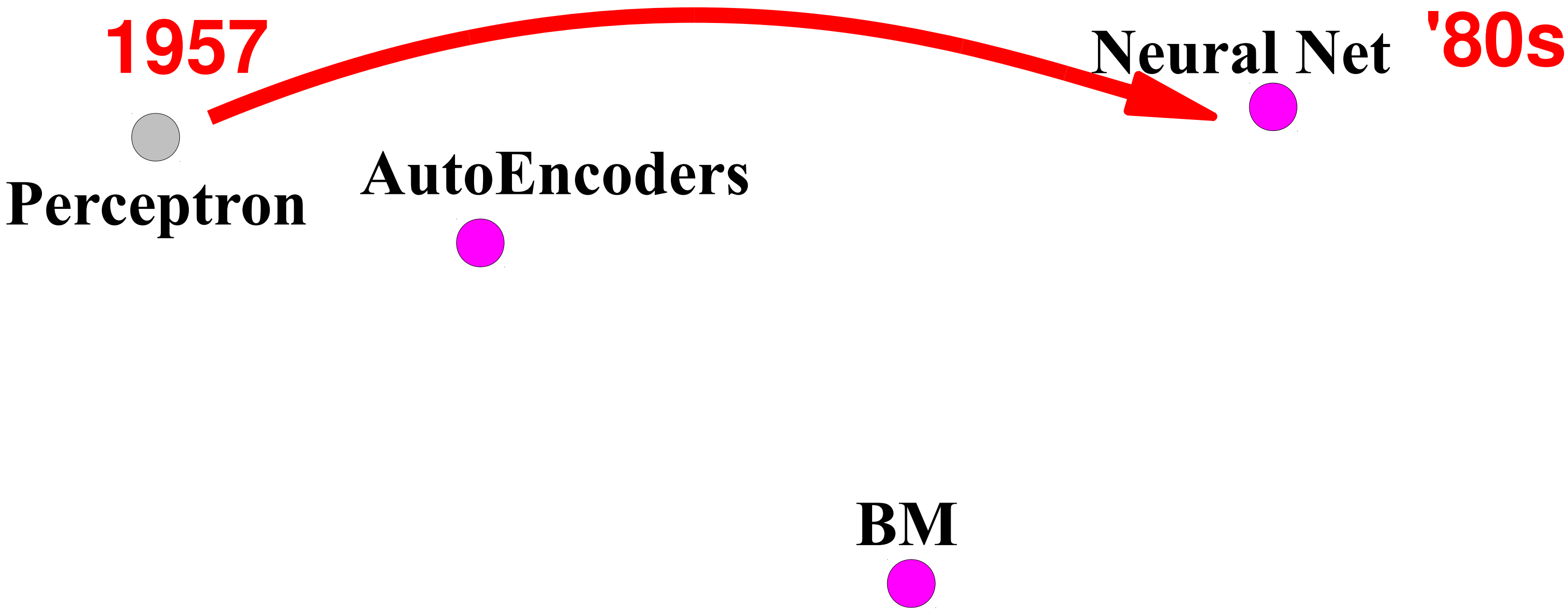
Some deep learning methods are probabilistic, others are loss-based, some are supervised, other unsupervised...

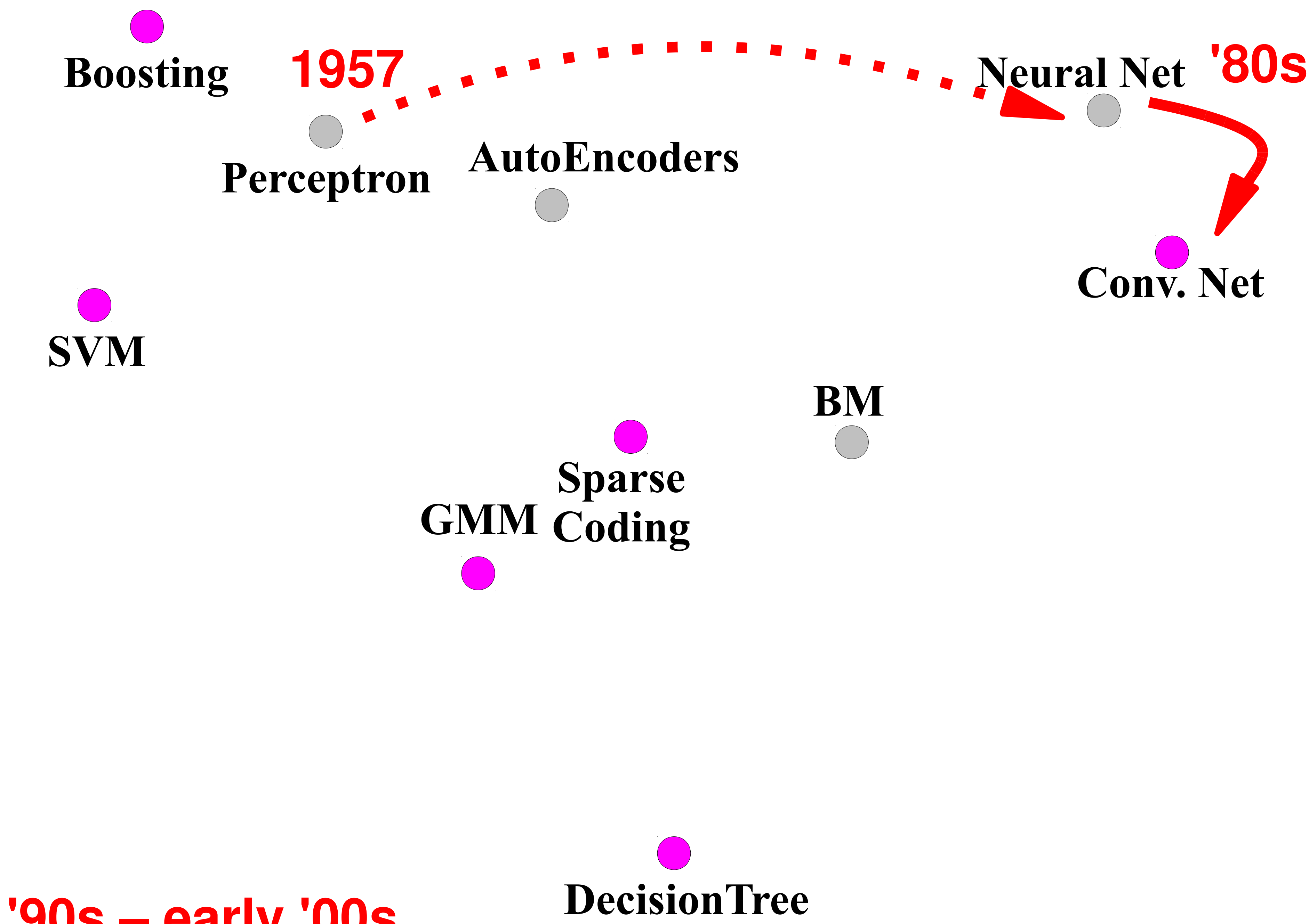
It's a large family!

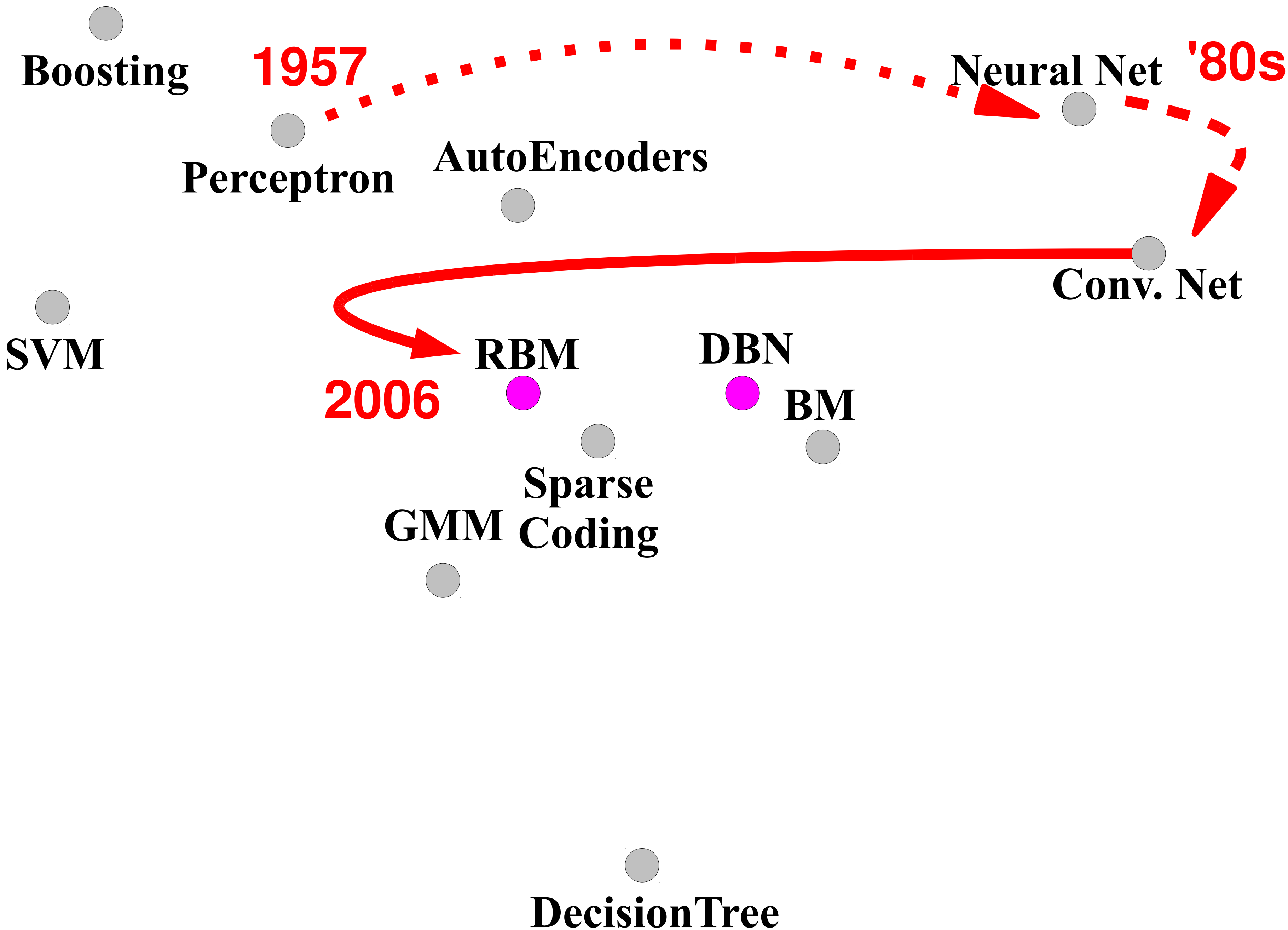
1957

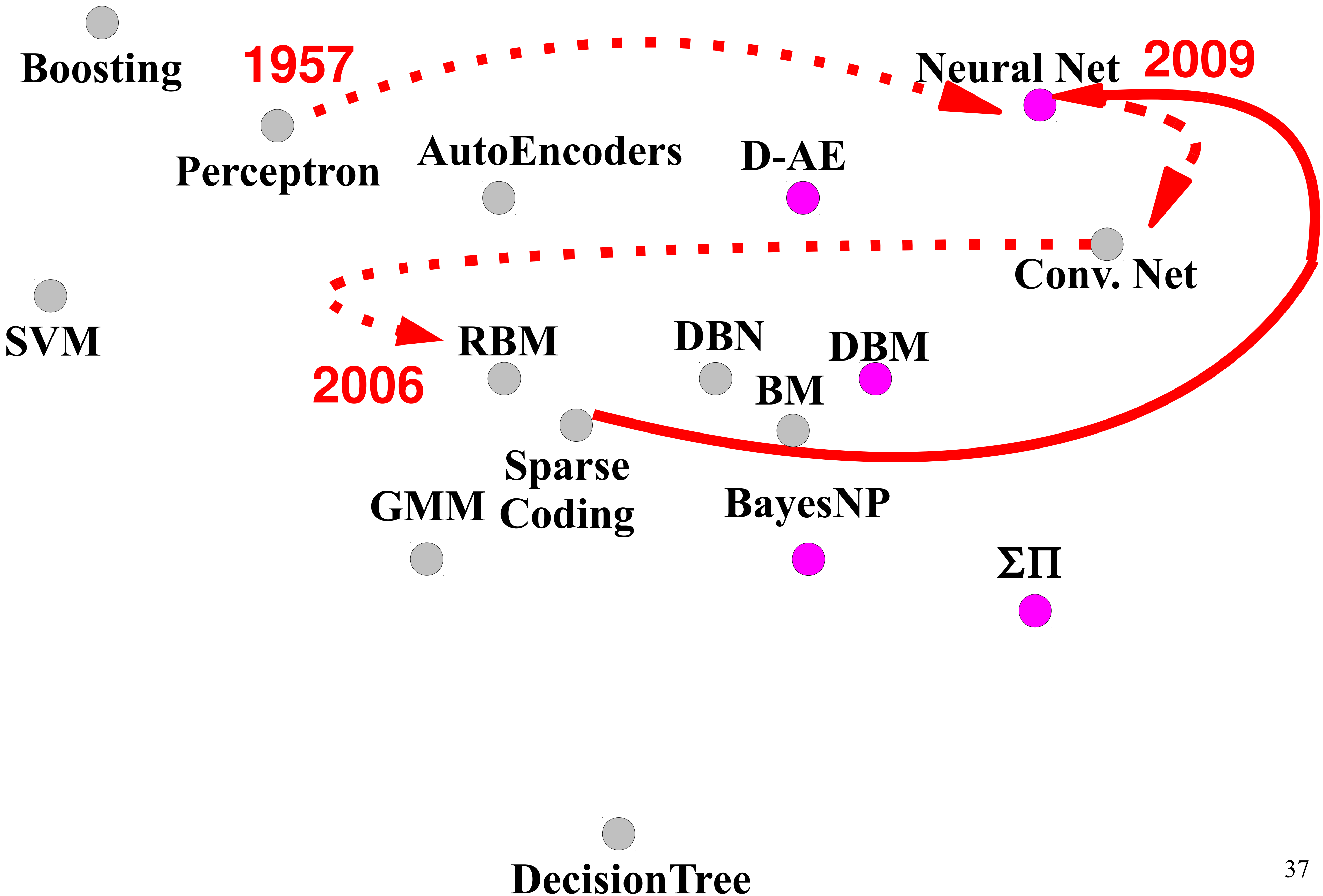
Perceptron

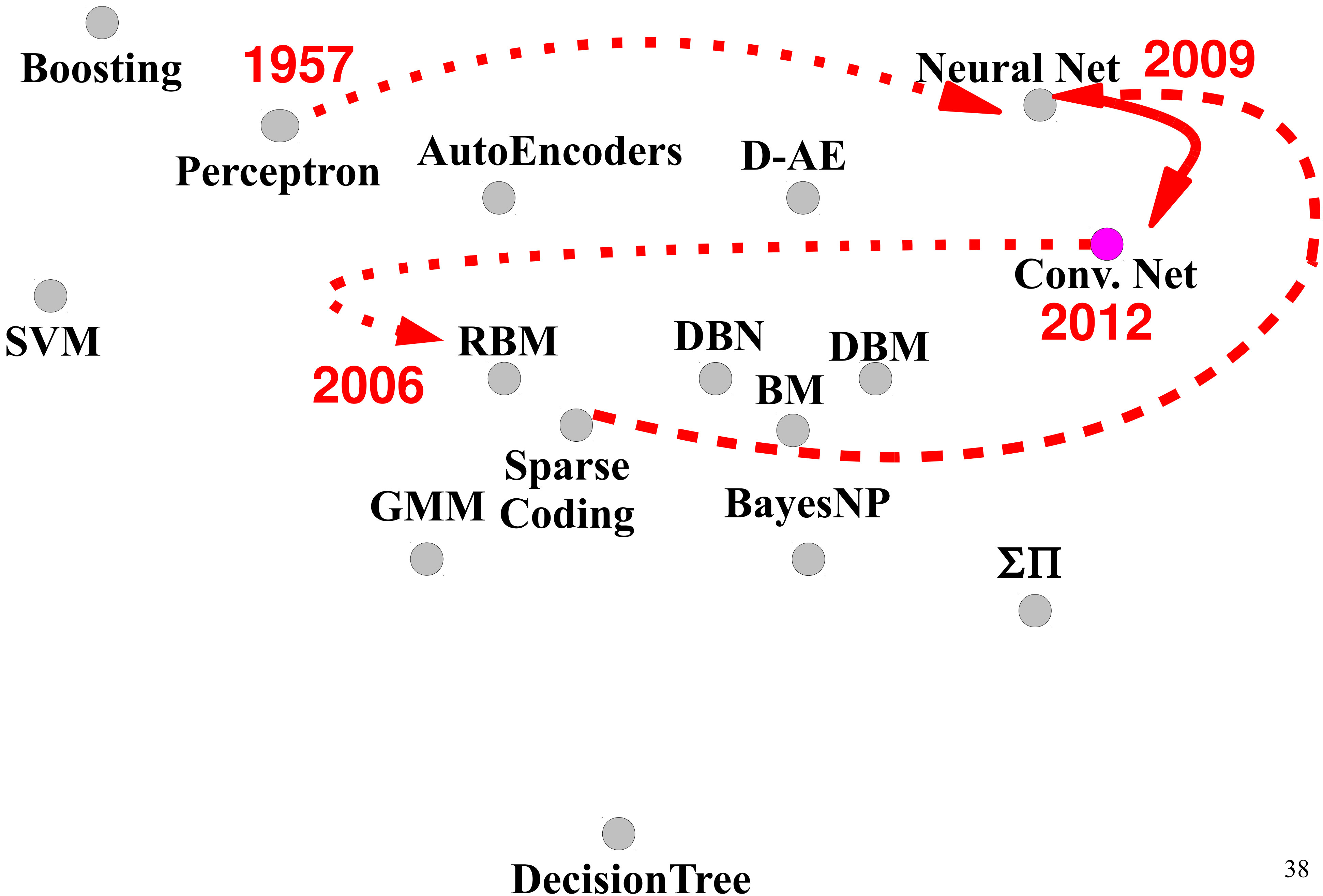
THE SPACE OF MACHINE LEARNING METHODS

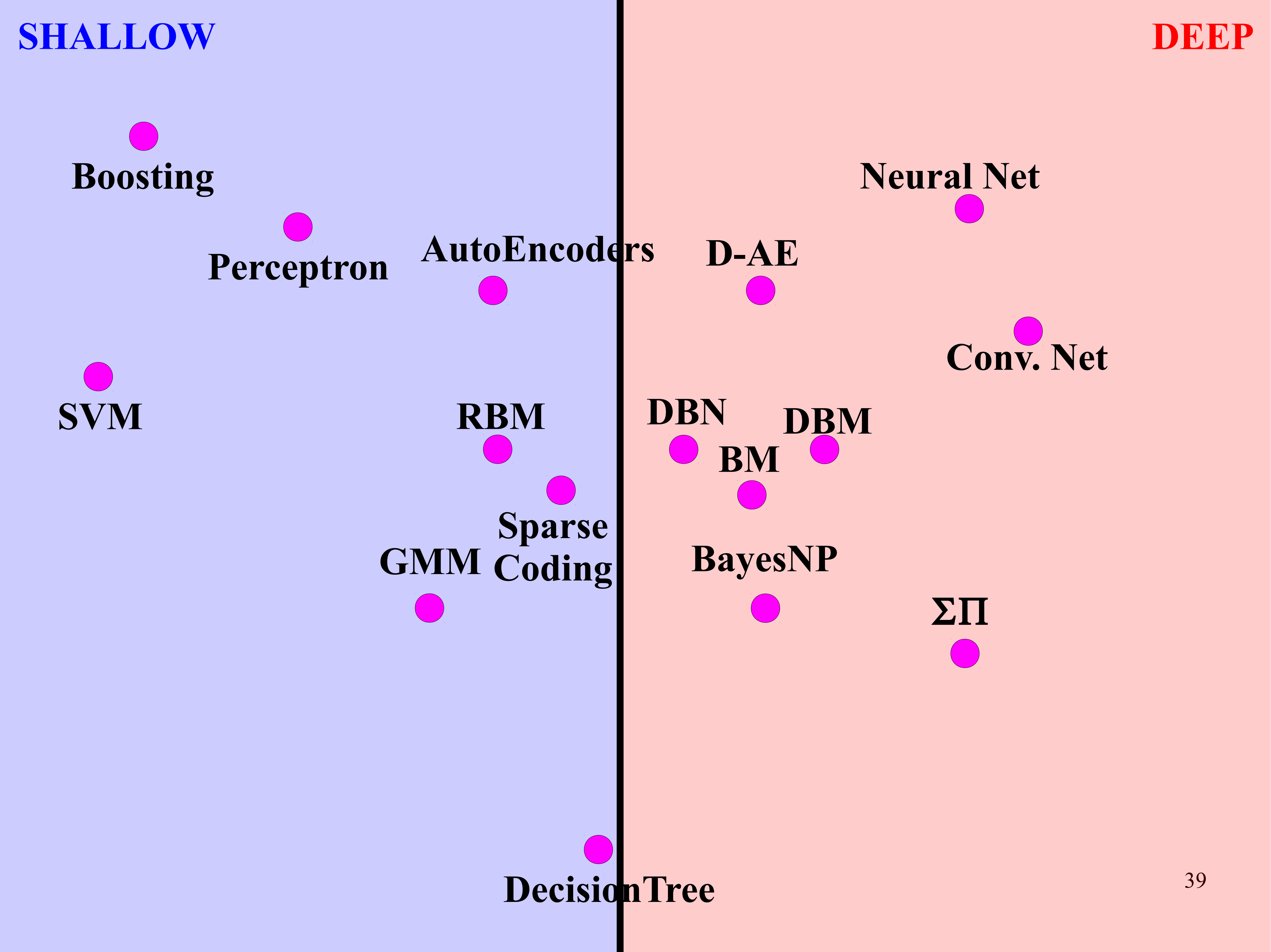

1957
Perceptron
AutoEncoders
Neural Net
'80s
BM

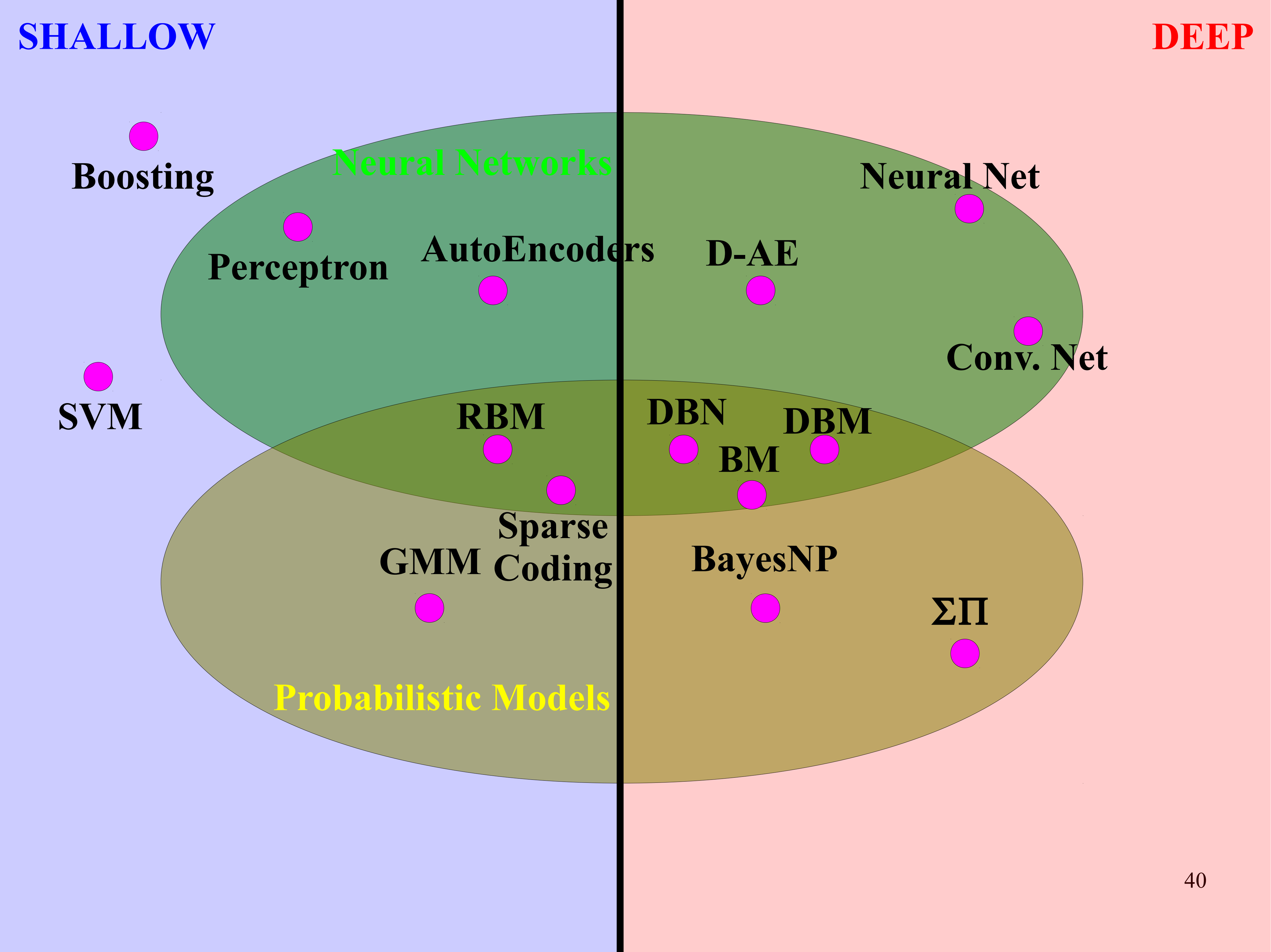

Boosting
1957
Neural Net
'80s
Perceptron
AutoEncoders
Conv. Net
SVM
BM
Sparse Coding
GMM
DecisionTree
'90s – early '00s

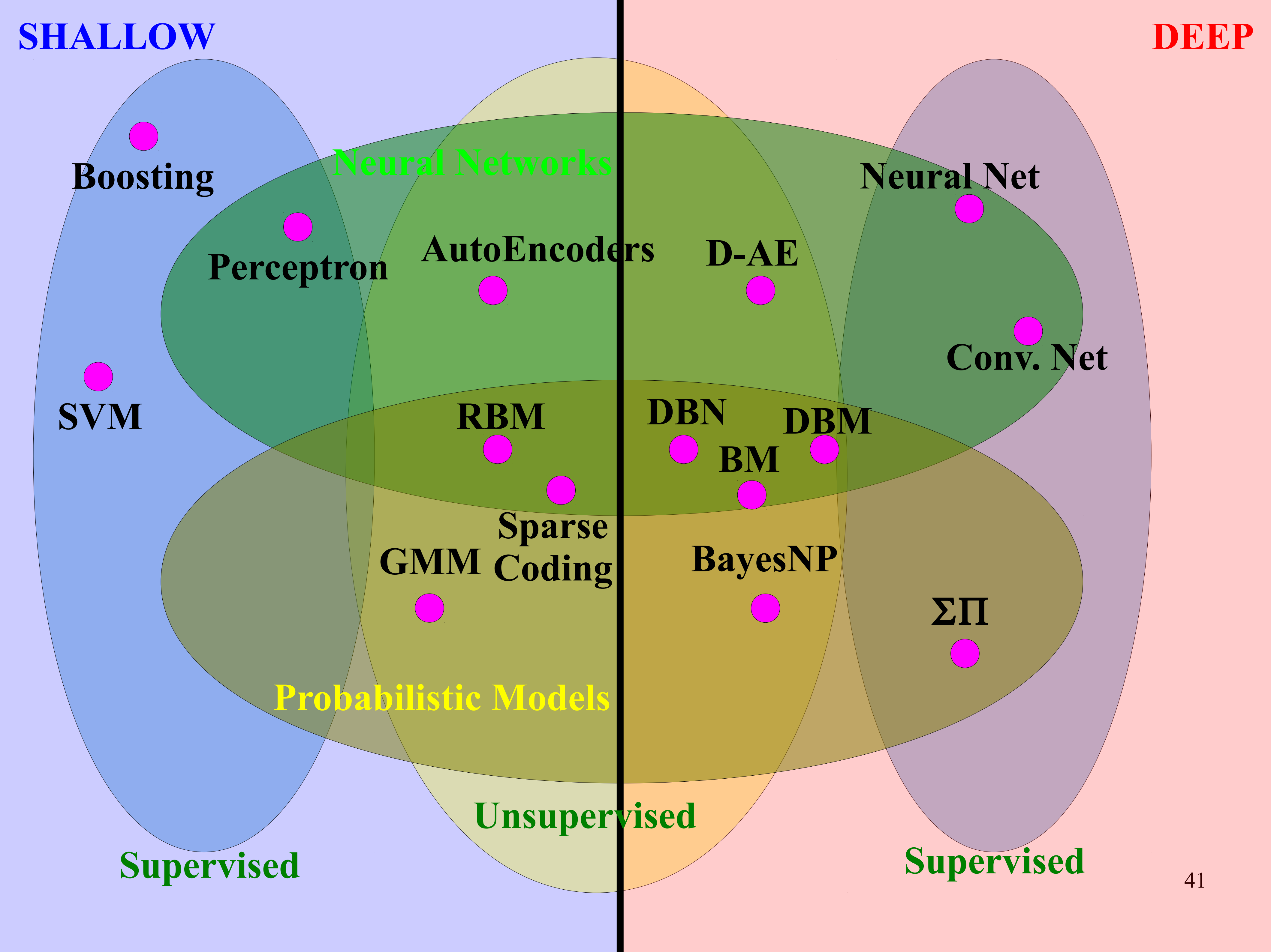

Boosting
1957
Neural Net
'80s
Perceptron
AutoEncoders
Conv. Net
SVM
RBM
DBN
2006
BM
Sparse
Coding
GMM
DecisionTree


Boosting
1957
Neural Net
2009
Perceptron
AutoEncoders
D-AE
Conv. Net
SVM
RBM
DBN
DBM
2006
BM
Sparse
Coding
GMM
BayesNP
ΣΠ
DecisionTree


Boosting
1957
Neural Net
2009
Perceptron
AutoEncoders
D-AE
Conv. Net
2012
SVM
RBM
DBN
DBM
2006
BM
Sparse
Coding
GMM
BayesNP
ΣΠ
DecisionTree


SHALLOW
DEEP
Boosting
Neural Net
Perceptron
AutoEncoders
D-AE
Conv. Net
SVM
RBM
DBN
DBM
BM
Sparse
Coding
GMM
BayesNP
ΣΠ
DecisionTree


SHALLOW
DEEP
Boosting
Neural Networks
Neural Net
Perceptron
AutoEncoders
D-AE
Conv. Net
SVM
RBM
DBN
DBM
BM
Sparse Coding
GMM
BayesNP
ΣΠ
Probabilistic Models


SHALLOW
DEEP
Boosting
Neural Networks
Neural Net
Perceptron
AutoEncoders
D-AE
Conv. Net
SVM
RBM
DBN
DBM
BM
Sparse Coding
GMM
BayesNP
ΣΠ
Probabilistic Models
Unsupervised
Supervised
Supervised

In this talk, we'll focus on **convolutional networks**.

Outline

- Motivation
- Deep Learning: The Big Picture
- From neural nets to convolutional nets
- Applications
- A practical guide

Linear Classifier: SVM

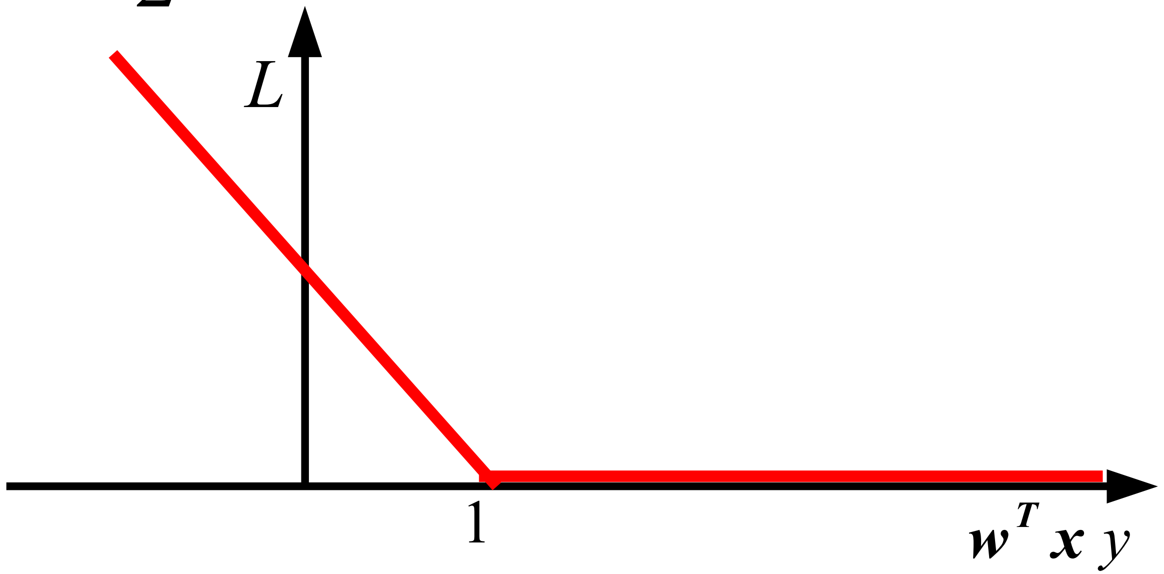
Input: $\boldsymbol{x} \in R^D$

Binary label: $y \in \{-1, +1\}$

Parameters: $\boldsymbol{w} \in R^D$

Output prediction: $\boldsymbol{w}^T \boldsymbol{x}$

Loss: $L = \frac{1}{2} ||\boldsymbol{w}||^2 + \lambda\, max[0, 1 - \boldsymbol{w}^T \boldsymbol{x}\, y]$



Hinge Loss

Ranzato

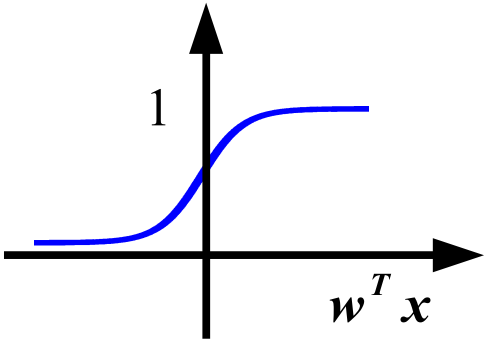
Linear Classifier: Logistic Regression

Input: $\boldsymbol{x} \in R^D$

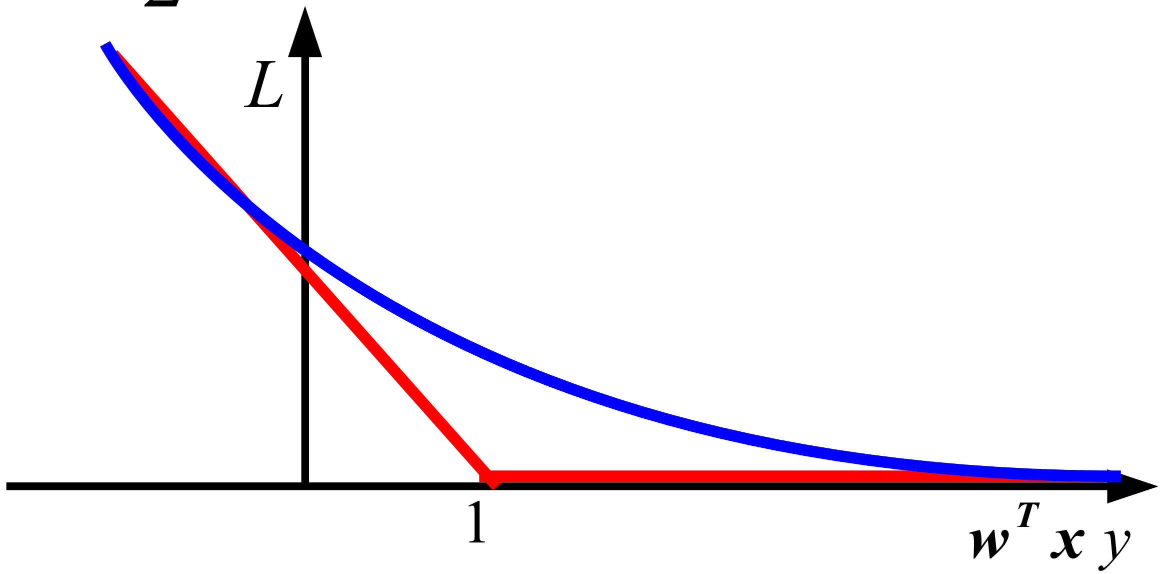
Binary label: $y \in \{-1, +1\}$

Parameters: $\boldsymbol{w} \in R^D$

Output prediction: $p(y=1|\boldsymbol{x}) = \dfrac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$



Loss: $L = \dfrac{1}{2}\|\boldsymbol{w}\|^2 + \lambda \log(1+\exp(-\boldsymbol{w}^T\boldsymbol{x}\, y))$



Log Loss

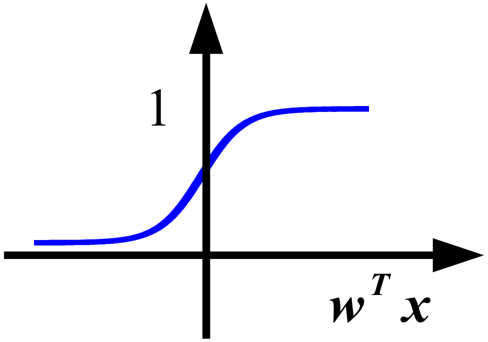
Linear Classifier: Logistic Regression

Input: $\boldsymbol{x} \in R^D$

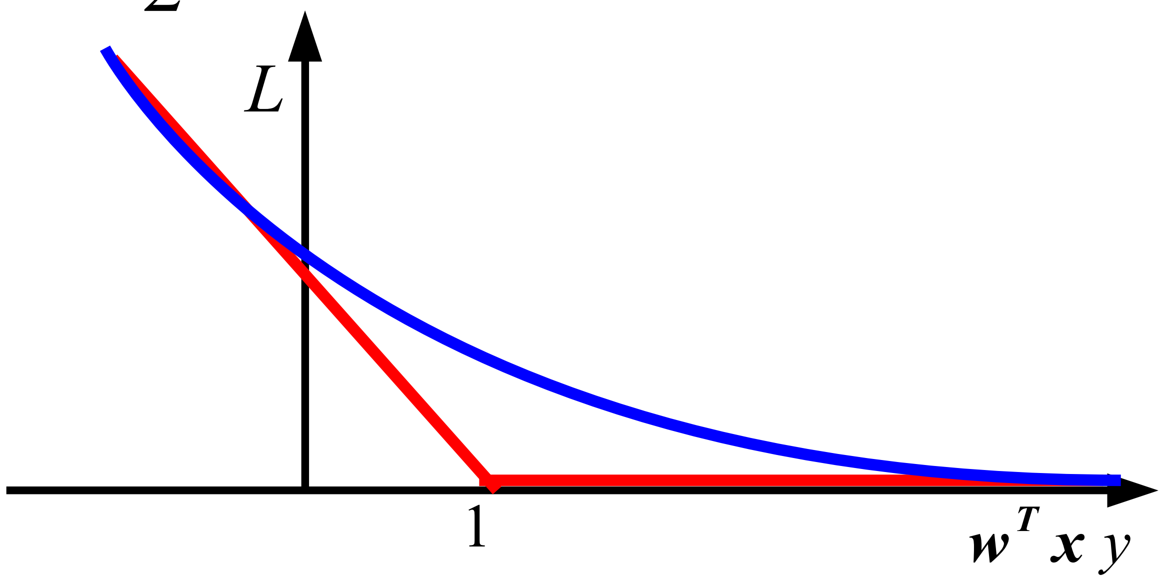
Binary label: $y \in \{-1, +1\}$

Parameters: $\boldsymbol{w} \in R^D$

Output prediction: $p(y=1|\boldsymbol{x}) = \dfrac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}}}$



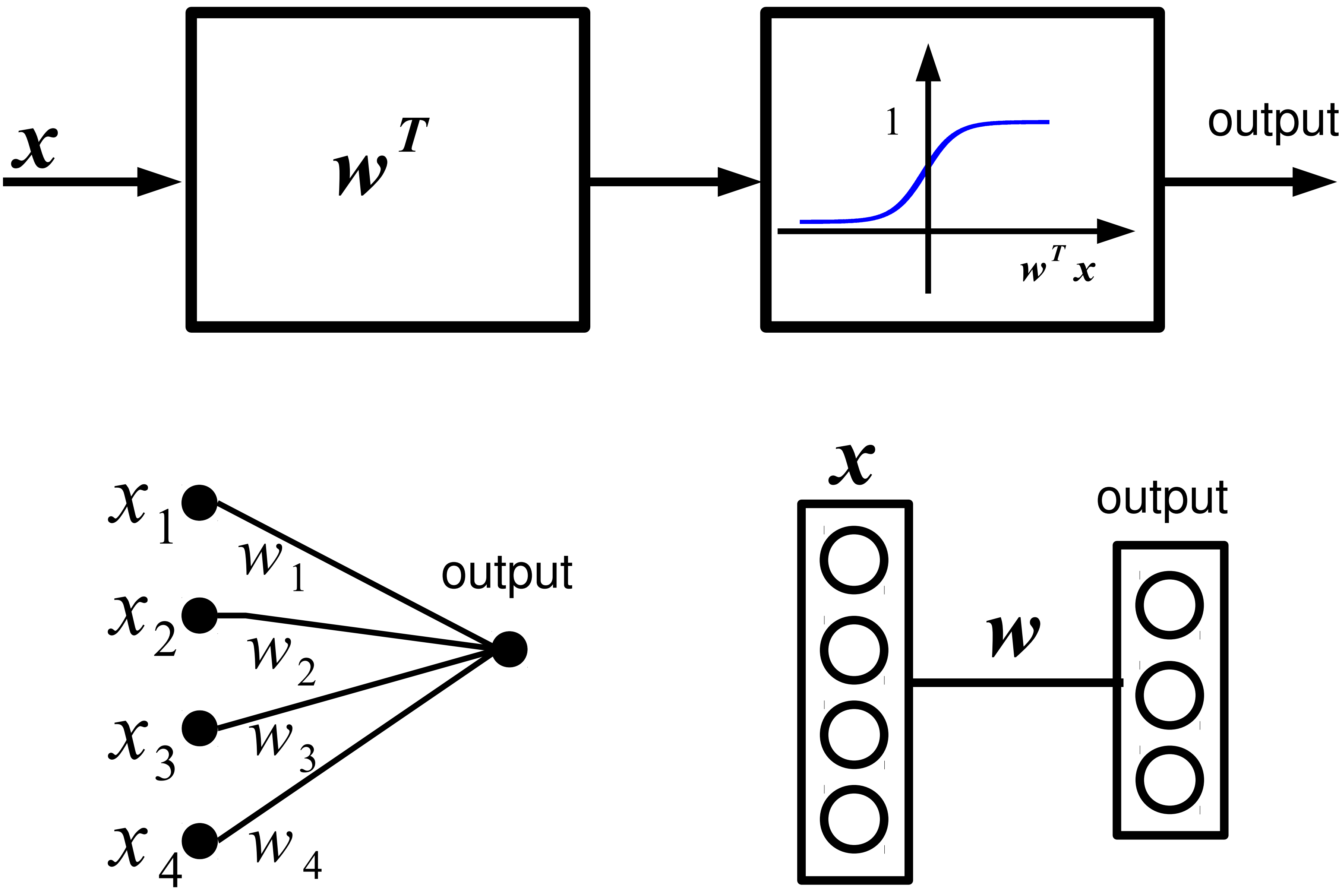
Loss: $L = \dfrac{1}{2}\|\boldsymbol{w}\|^2 - \lambda \log(p(y|\boldsymbol{x}))$



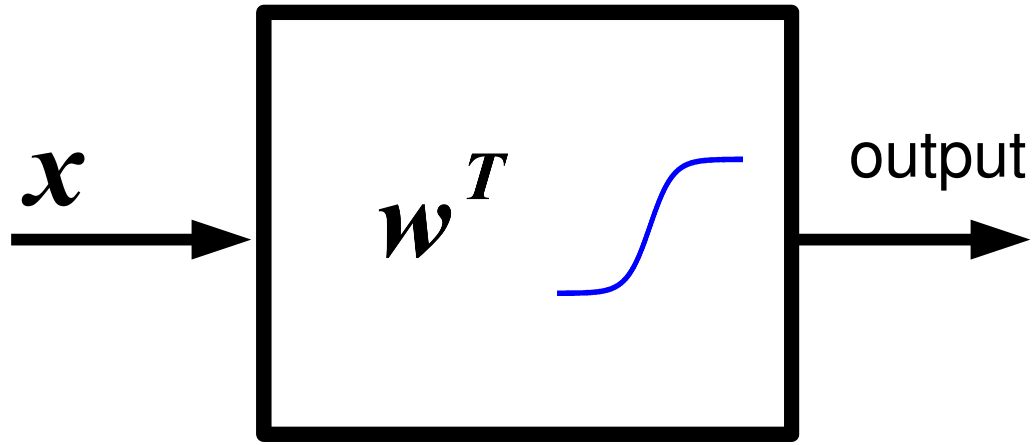
Log Loss

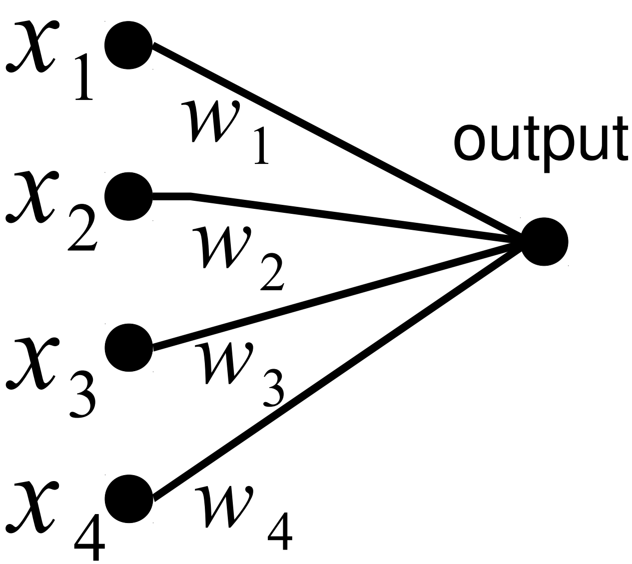
Ranzato

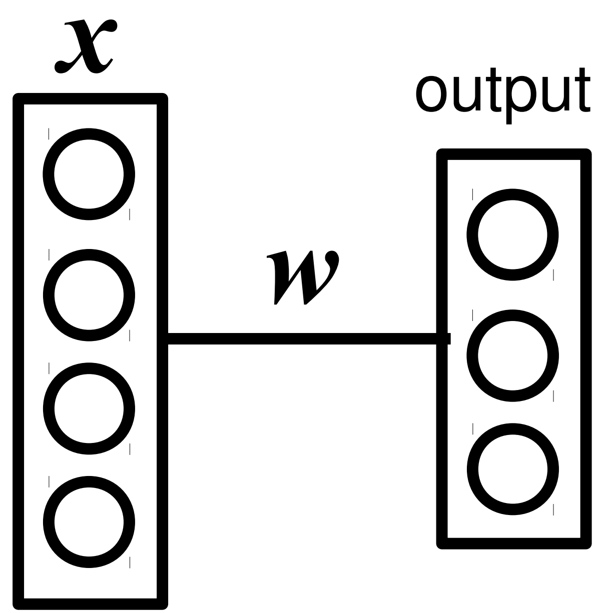
Graphical Representation



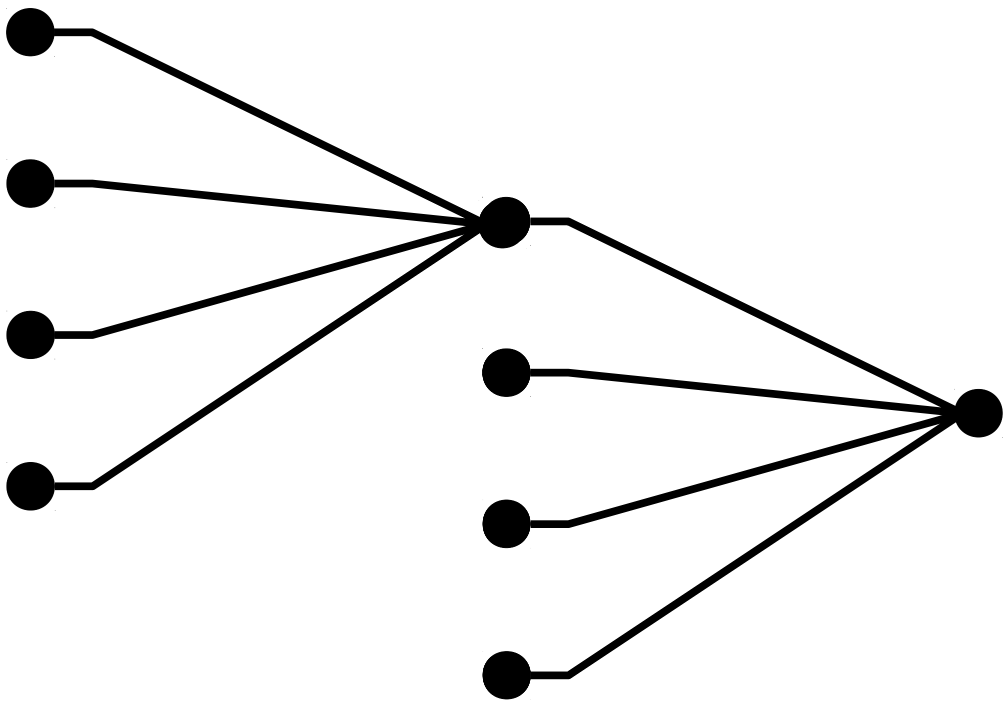
Graphical Representation



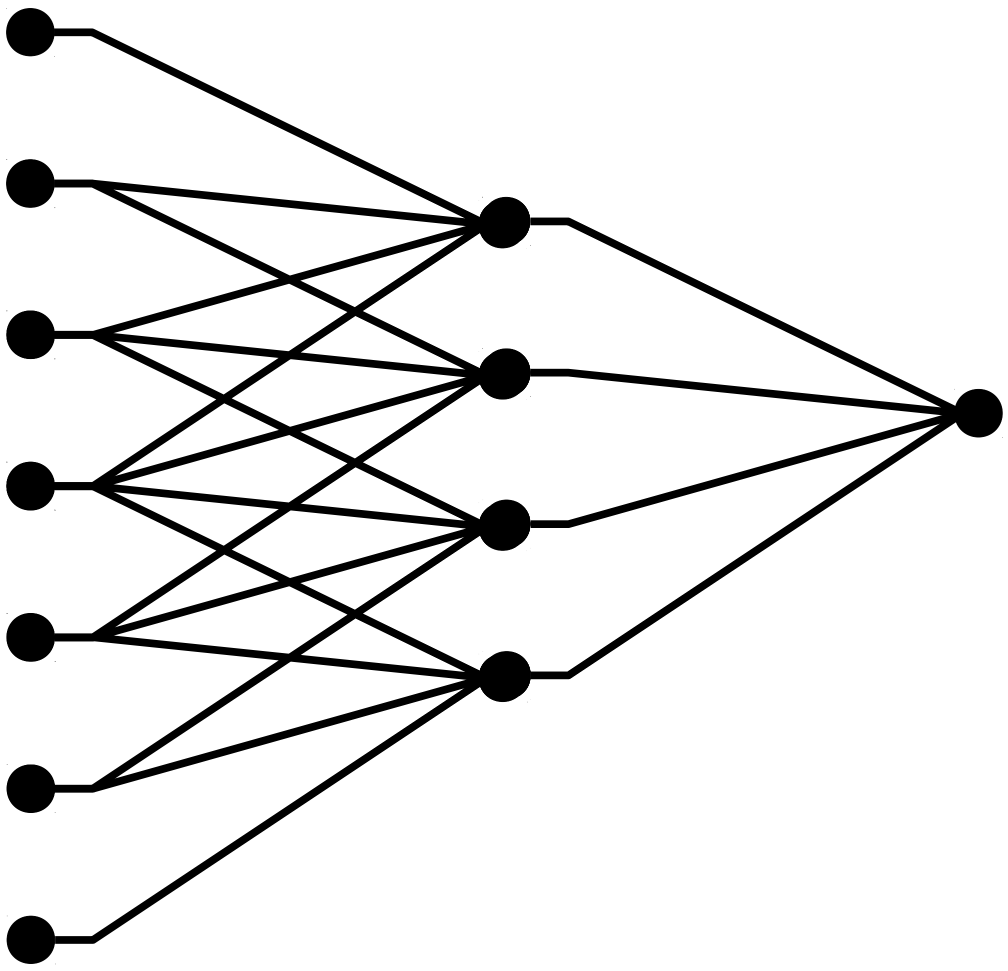




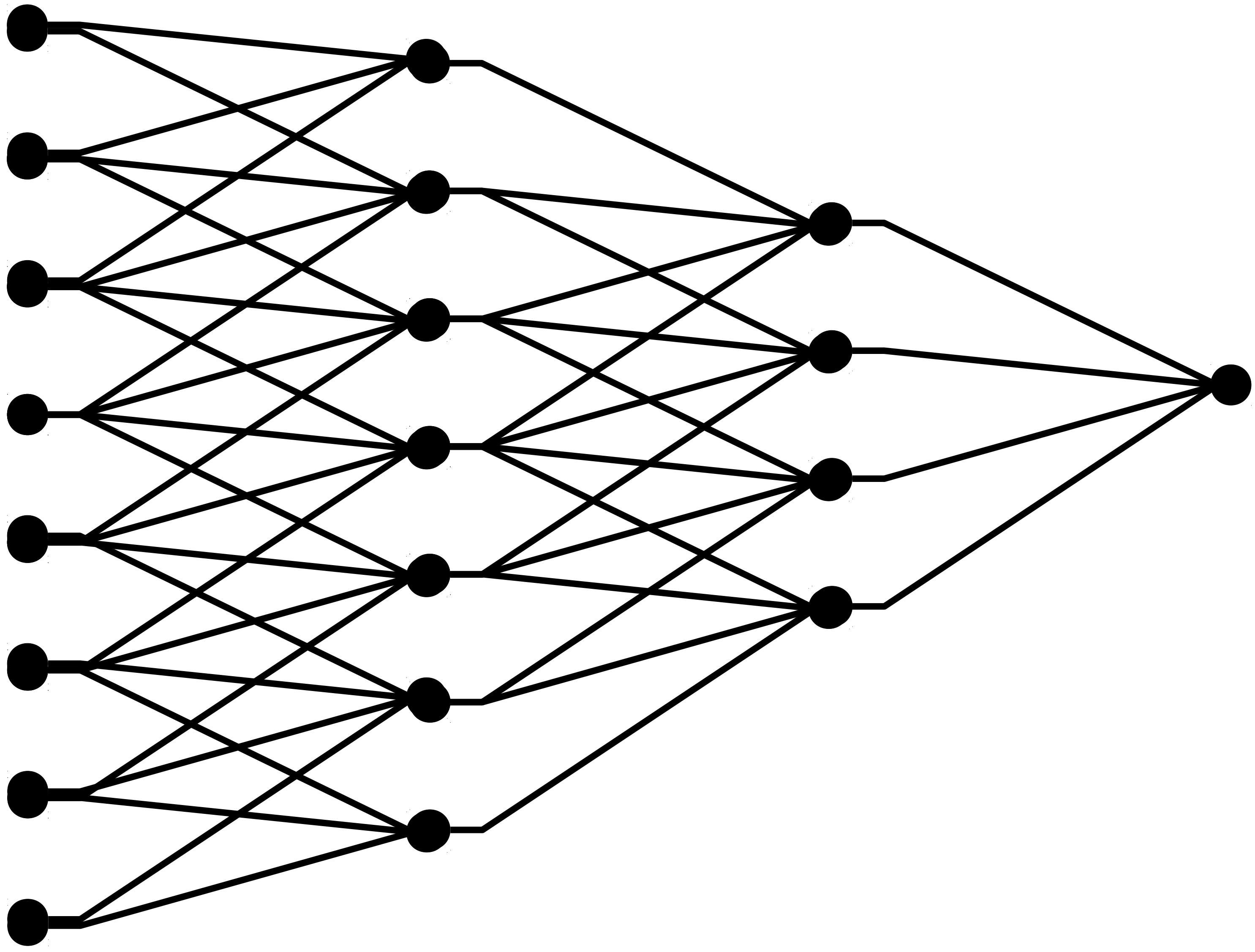
From Logistic Regression To Neural Nets



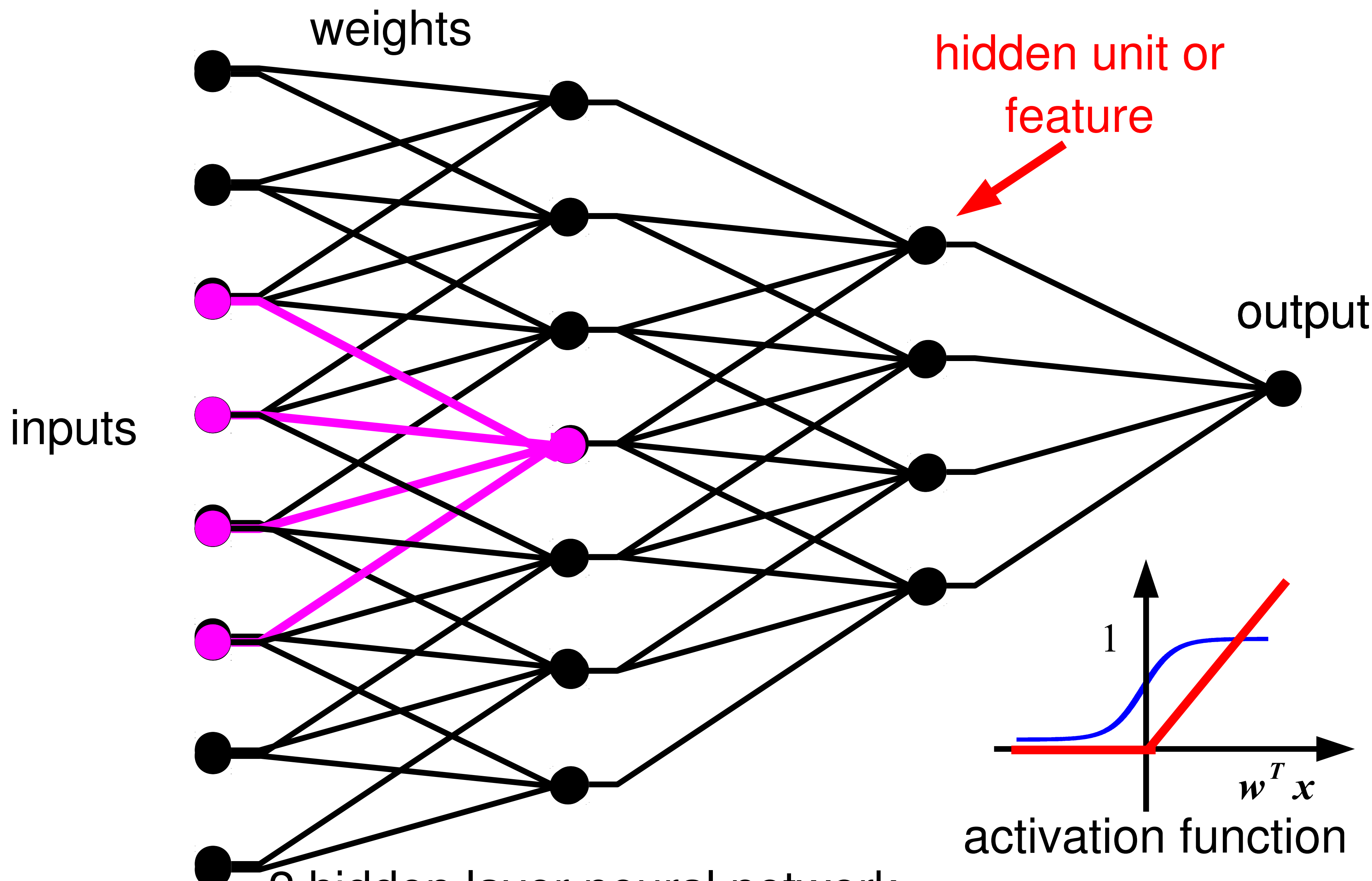
From Logistic Regression To Neural Nets



From Logistic Regression To Neural Nets



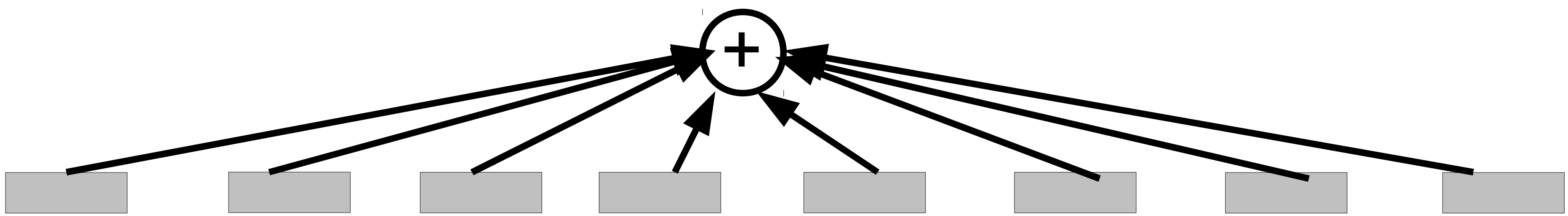
Neural Network



2 hidden layer neural network
(4 layer neural network)

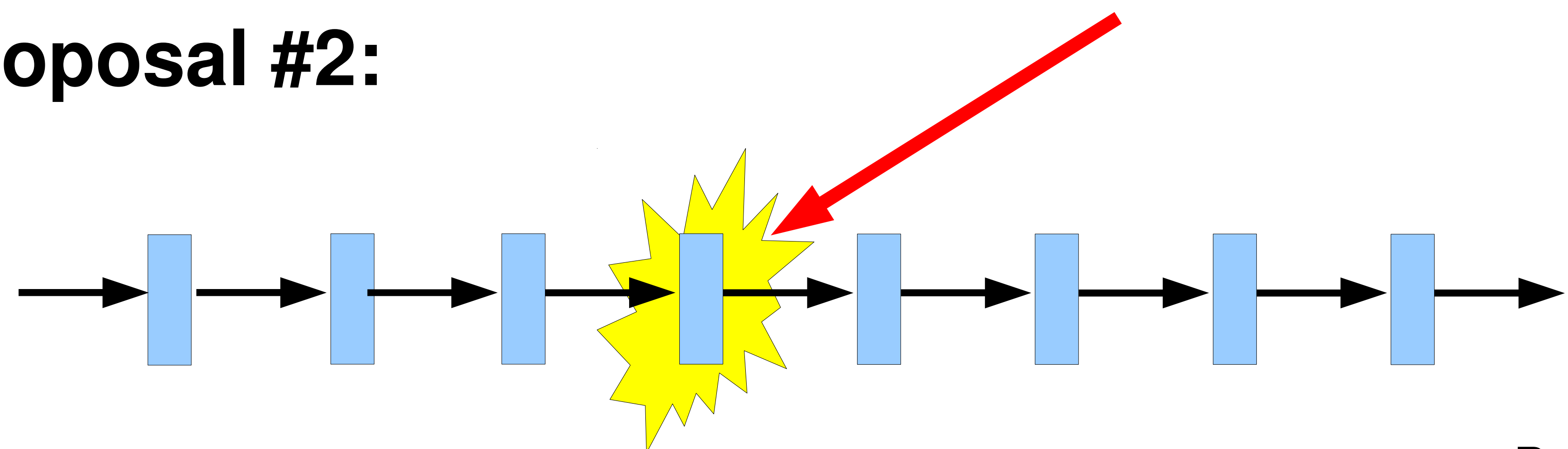
Learning Non-Linear Features

Proposal #1:

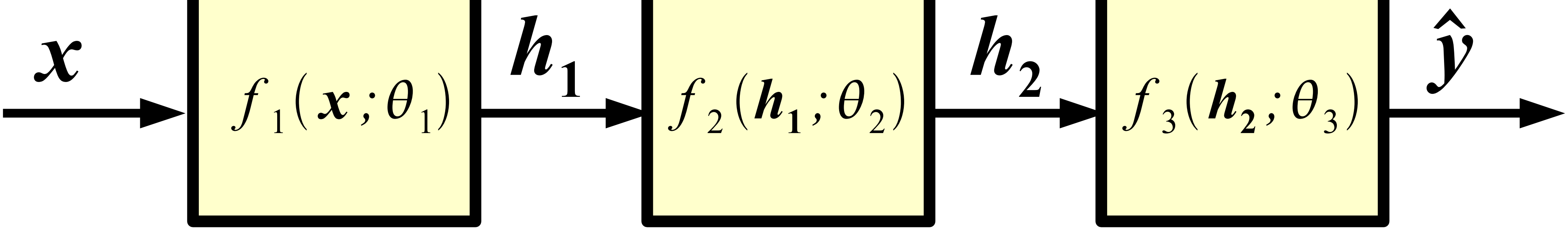


Each of box is a feature detector

Proposal #2:



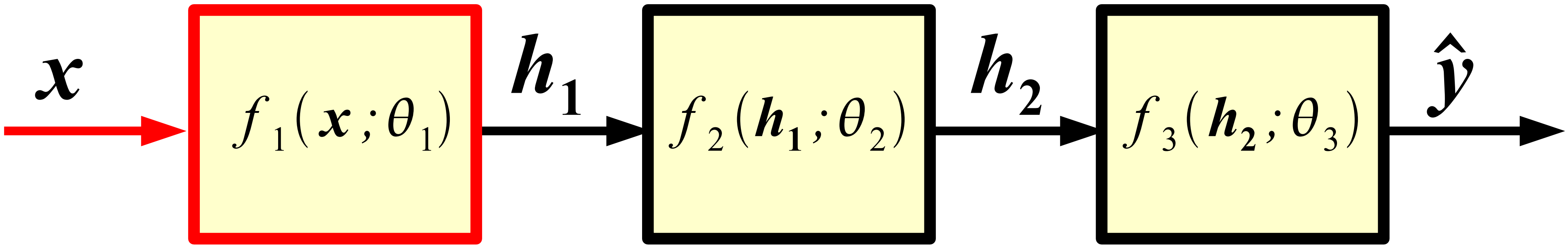
Neural Nets



NOTE: In practice, each module does NOT need to be a logistic regression classifier.

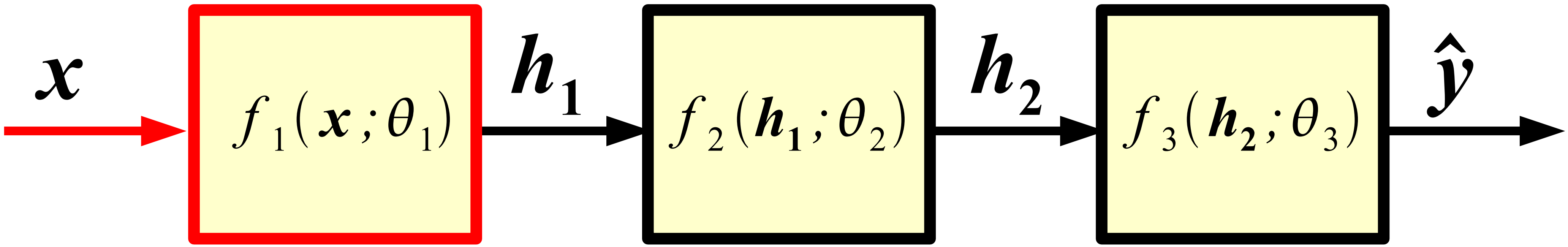
Any (a.e. differentiable) non-linear transformation is potentially good.

Forward Propagation (FPROP)



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

Forward Propagation (FPROP)

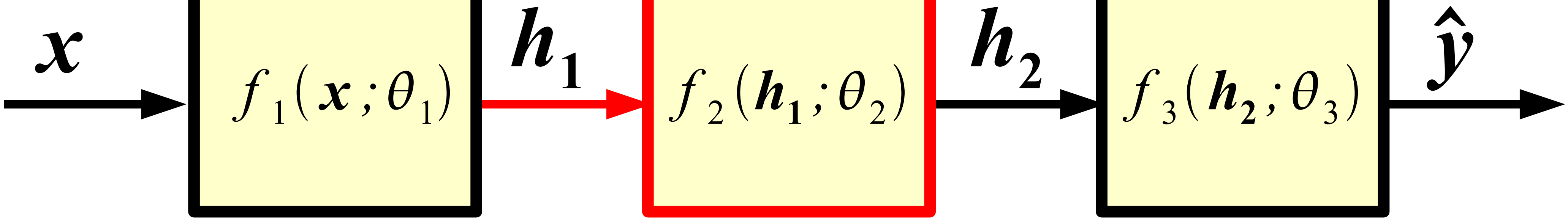


1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

For instance,

$$\boldsymbol{h_1} = max(0, W_1 \boldsymbol{x} + \boldsymbol{b}_1)$$

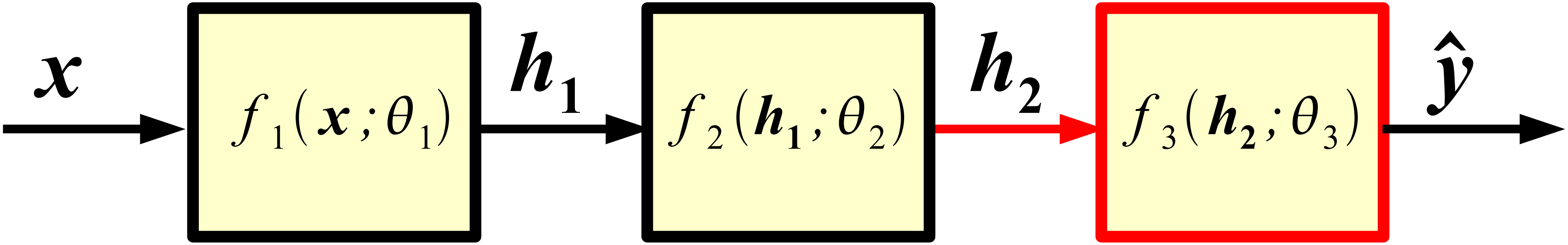
Forward Propagation (FPROP)



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

2) Given $\boldsymbol{h_1}$ compute: $\boldsymbol{h_2} = f_2(\boldsymbol{h}_1;\theta_2)$

Forward Propagation (FPROP)

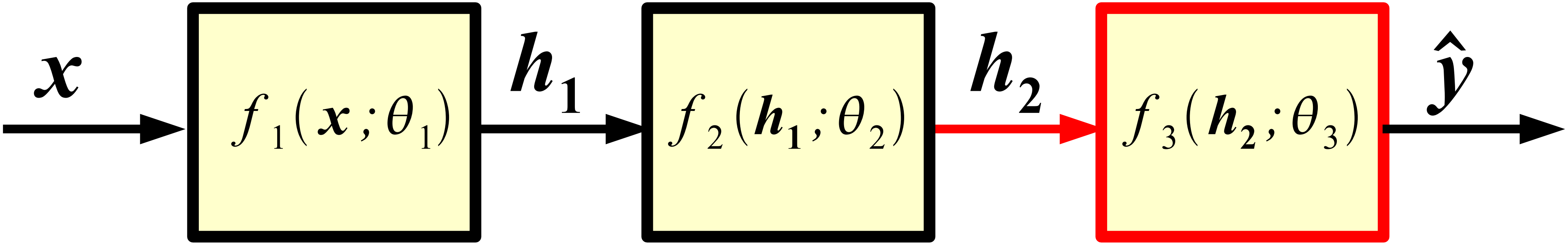


1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

2) Given $\boldsymbol{h_1}$ compute: $\boldsymbol{h_2} = f_2(\boldsymbol{h}_1;\theta_2)$

3) Given $\boldsymbol{h_2}$ compute: $\hat{\boldsymbol{y}} = f_3(\boldsymbol{h}_2;\theta_3)$

Forward Propagation (FPROP)



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

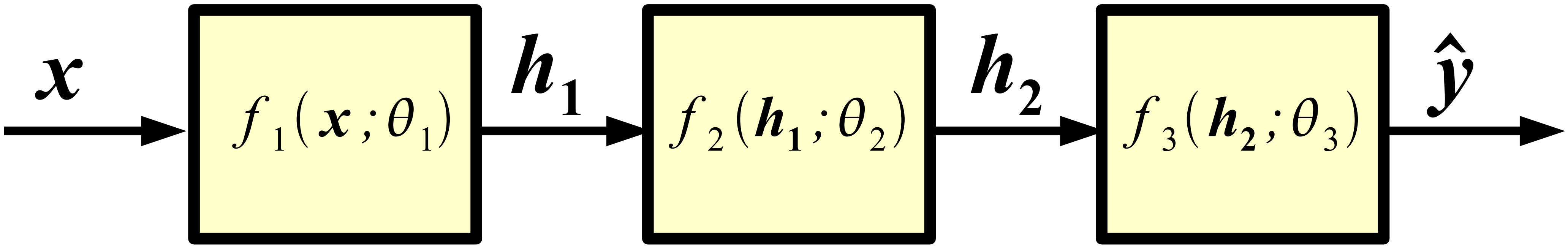
2) Given $\boldsymbol{h_1}$ compute: $\boldsymbol{h_2} = f_2(\boldsymbol{h}_1;\theta_2)$

3) Given $\boldsymbol{h_2}$ compute: $\hat{\boldsymbol{y}} = f_3(\boldsymbol{h}_2;\theta_3)$

For instance,

$$\hat{y}_i = p(class = i | \boldsymbol{x}) = \frac{e^{W_{3i}\boldsymbol{h}_2 + b_{3i}}}{\sum_k e^{W_{3k}\boldsymbol{h}_2 + b_{3k}}}$$

Forward Propagation (FPROP)



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

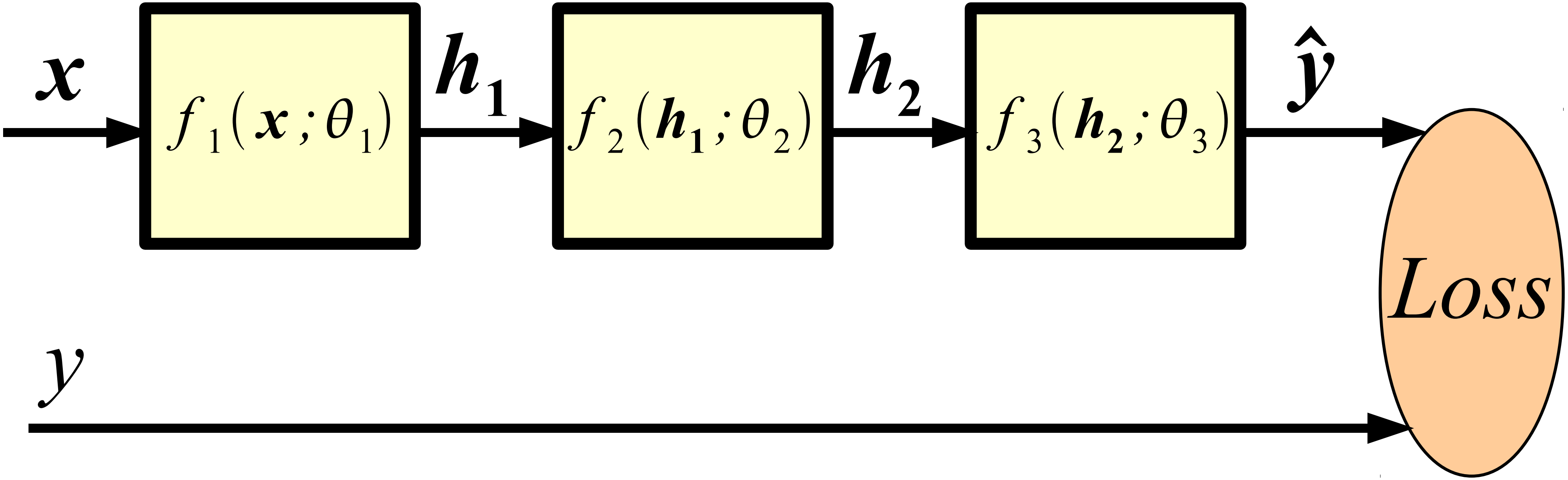
2) Given $\boldsymbol{h_1}$ compute: $\boldsymbol{h_2} = f_2(\boldsymbol{h}_1;\theta_2)$

3) Given $\boldsymbol{h_2}$ compute: $\hat{\boldsymbol{y}} = f_3(\boldsymbol{h}_2;\theta_3)$

This is the typical processing at test time.

At training time, we need to compute an error measure and tune the parameters to decrease the error.

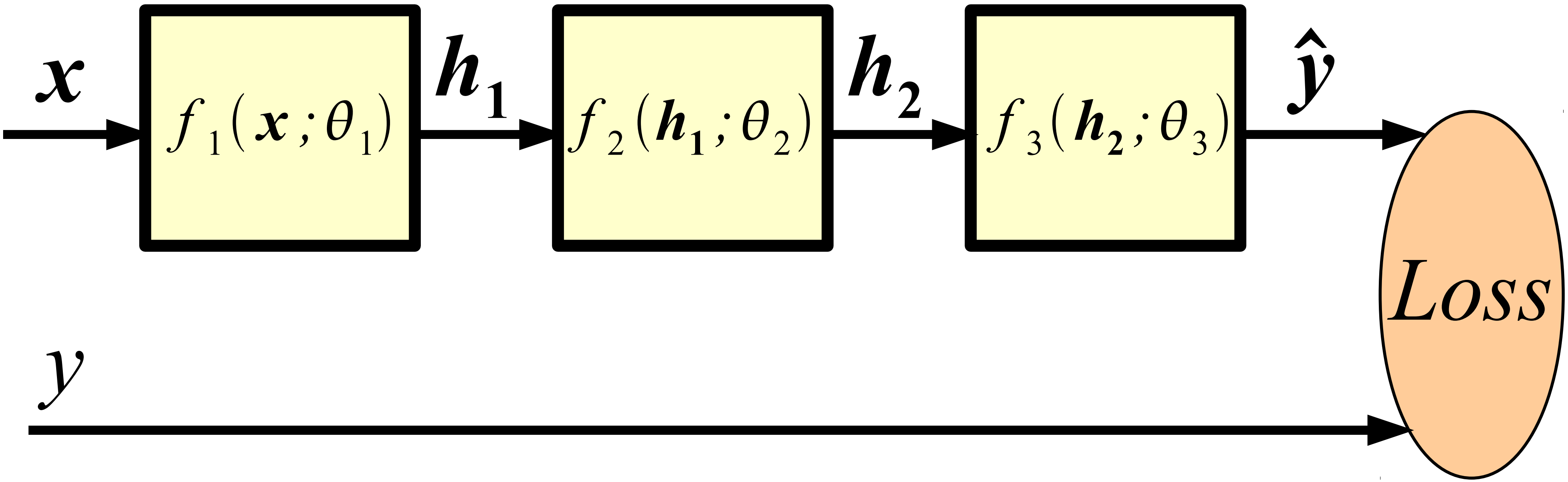
Loss



The measure of how well the model fits the training set is given by a suitable loss function: $L(\boldsymbol{x}, y; \boldsymbol{\theta})$

The loss depends on the input $\boldsymbol{x}$, the target label y, and the parameters $\boldsymbol{\theta}$.

Loss

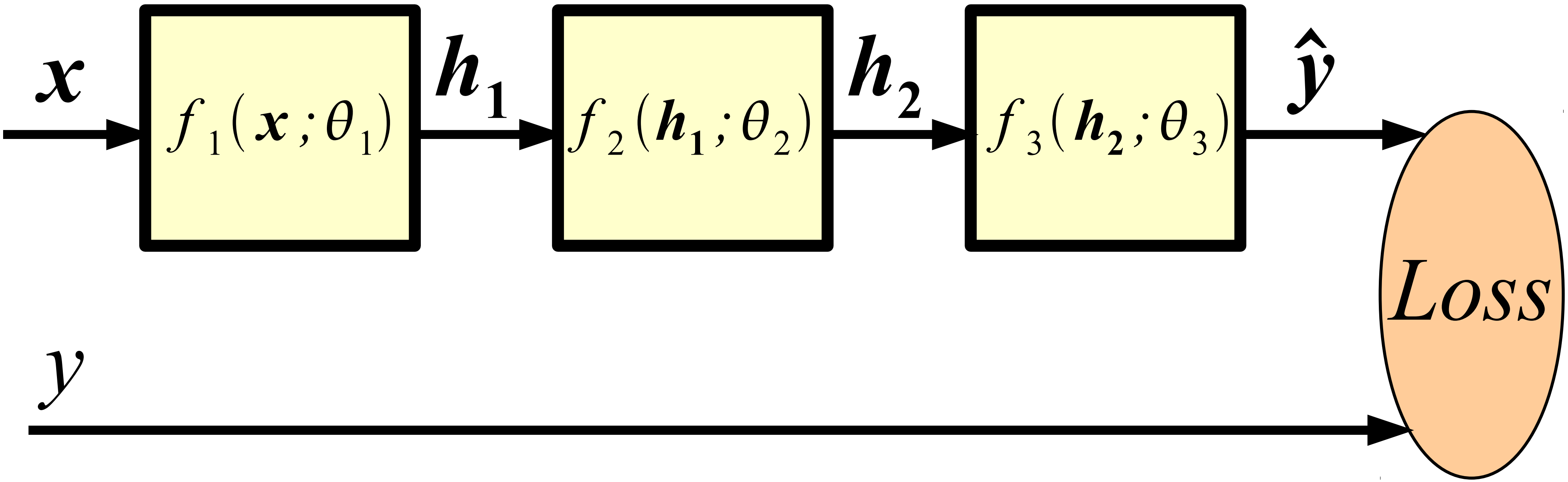


The measure of how well the model fits the training set is given by a suitable loss function: $L(\boldsymbol{x}, y; \boldsymbol{\theta})$

For instance,

$$L(\boldsymbol{x}, y=k; \boldsymbol{\theta}) = -\log(p(class=k|\boldsymbol{x}))$$

Loss



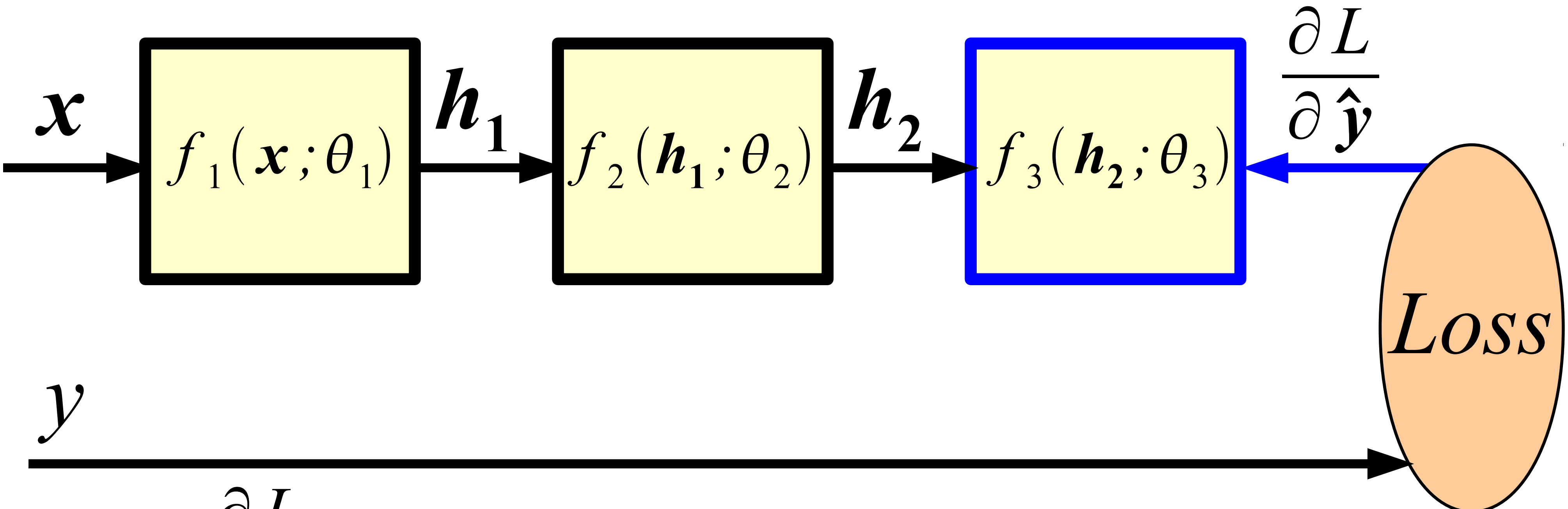
Q.: how to tune the parameters to decrease the loss?

If loss is (a.e.) differentiable we can compute gradients.

We can use chain-rule, a.k.a. **back-propagation**, to compute the gradients w.r.t. parameters at the lower layers.

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

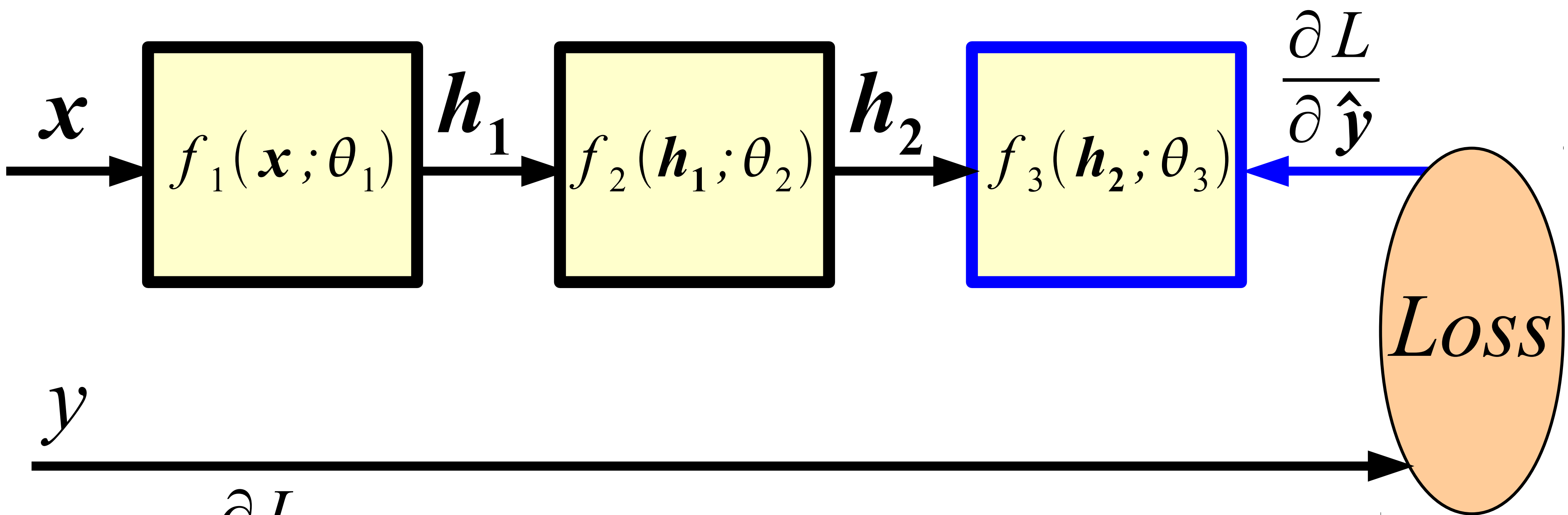
Backward Propagation (BPROP)



Given $\frac{\partial L}{\partial \hat{\boldsymbol{y}}}$ and assumiing the Jacobian of each module is easy to compute, then we have:

$$\frac{\partial L}{\partial \theta_3} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \theta_3} \qquad \frac{\partial L}{\partial \boldsymbol{h}_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{h}_2}$$

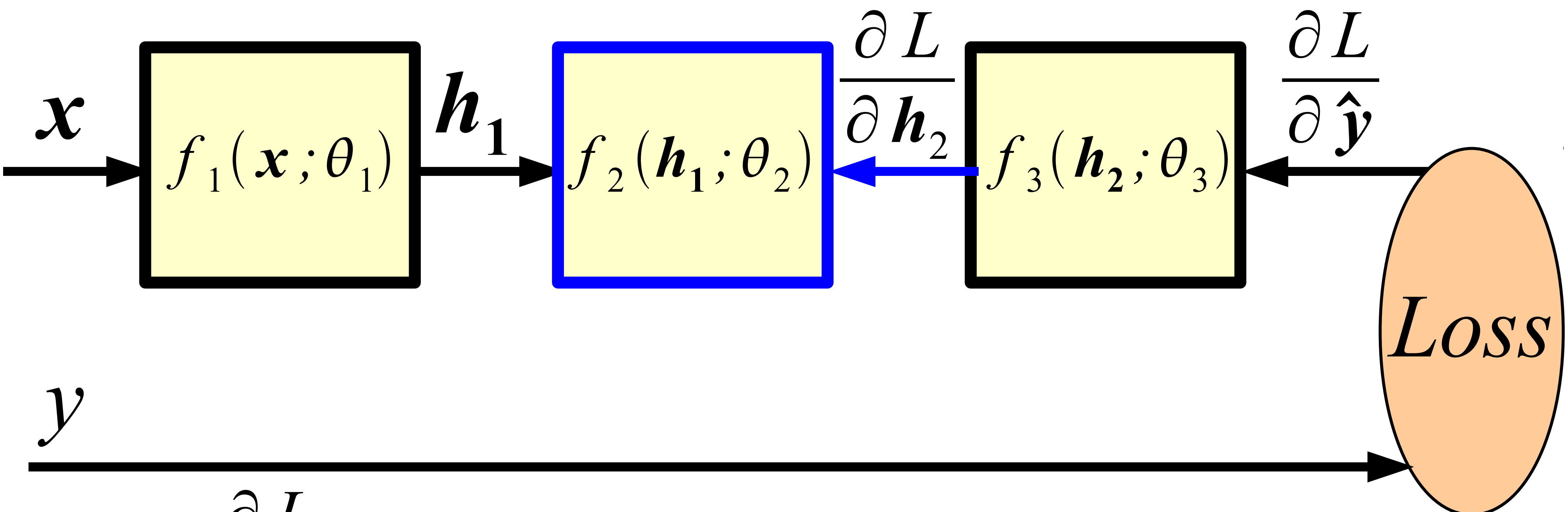
Backward Propagation (BPROP)



Given $\frac{\partial L}{\partial \hat{\boldsymbol{y}}}$ and assumiing the Jacobian of each module is easy to compute, then we have:

$$\frac{\partial L}{\partial \theta_3} = (\hat{\boldsymbol{y}} - \boldsymbol{y})\ \boldsymbol{h}_2{}' \qquad \frac{\partial L}{\partial \boldsymbol{h}_2} = (\hat{\boldsymbol{y}} - \boldsymbol{y})\ \theta_3{}'$$

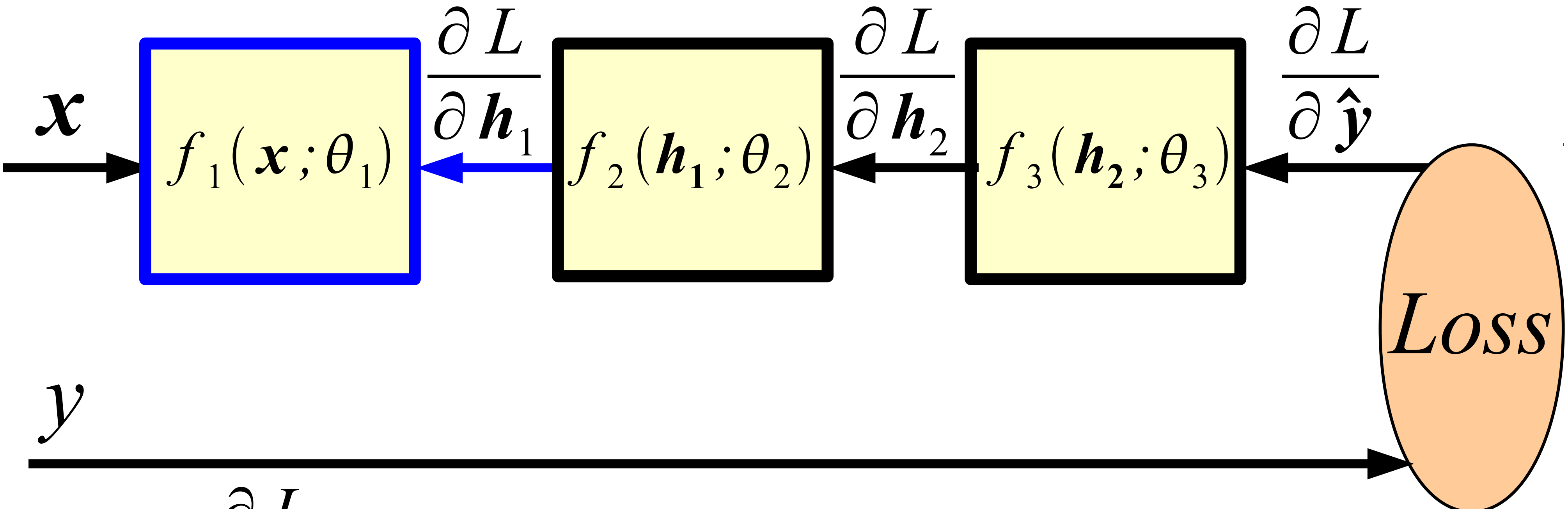
Backward Propagation (BPROP)



Given $\frac{\partial L}{\partial \boldsymbol{h}_2}$ we can compute now:

$$\frac{\partial L}{\partial \theta_2} = \frac{\partial L}{\partial \boldsymbol{h}_2} \frac{\partial \boldsymbol{h}_2}{\partial \theta_2} \qquad \frac{\partial L}{\partial \boldsymbol{h}_1} = \frac{\partial L}{\partial \boldsymbol{h}_2} \frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{h}_1}$$

Backward Propagation (BPROP)



Given $\frac{\partial L}{\partial \boldsymbol{h}_1}$ we can compute now:

$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial \boldsymbol{h}_1} \frac{\partial \boldsymbol{h}_1}{\partial \theta_1}$$

Optimization

Stochastic Gradient Descent (on mini-batches):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}, \eta \in R$$

Stochastic Gradient Descent with Momentum:

$$\theta \leftarrow \theta - \eta \Delta$$

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

LeCun et al. “Efficient BackProp” Neural Networks: Tricks of the trade 1998

Schaul et al. “No more pesky learning rates” ICML 2013

Sutskever et al. “On the importance of initialization and momentum...” ICML 2013

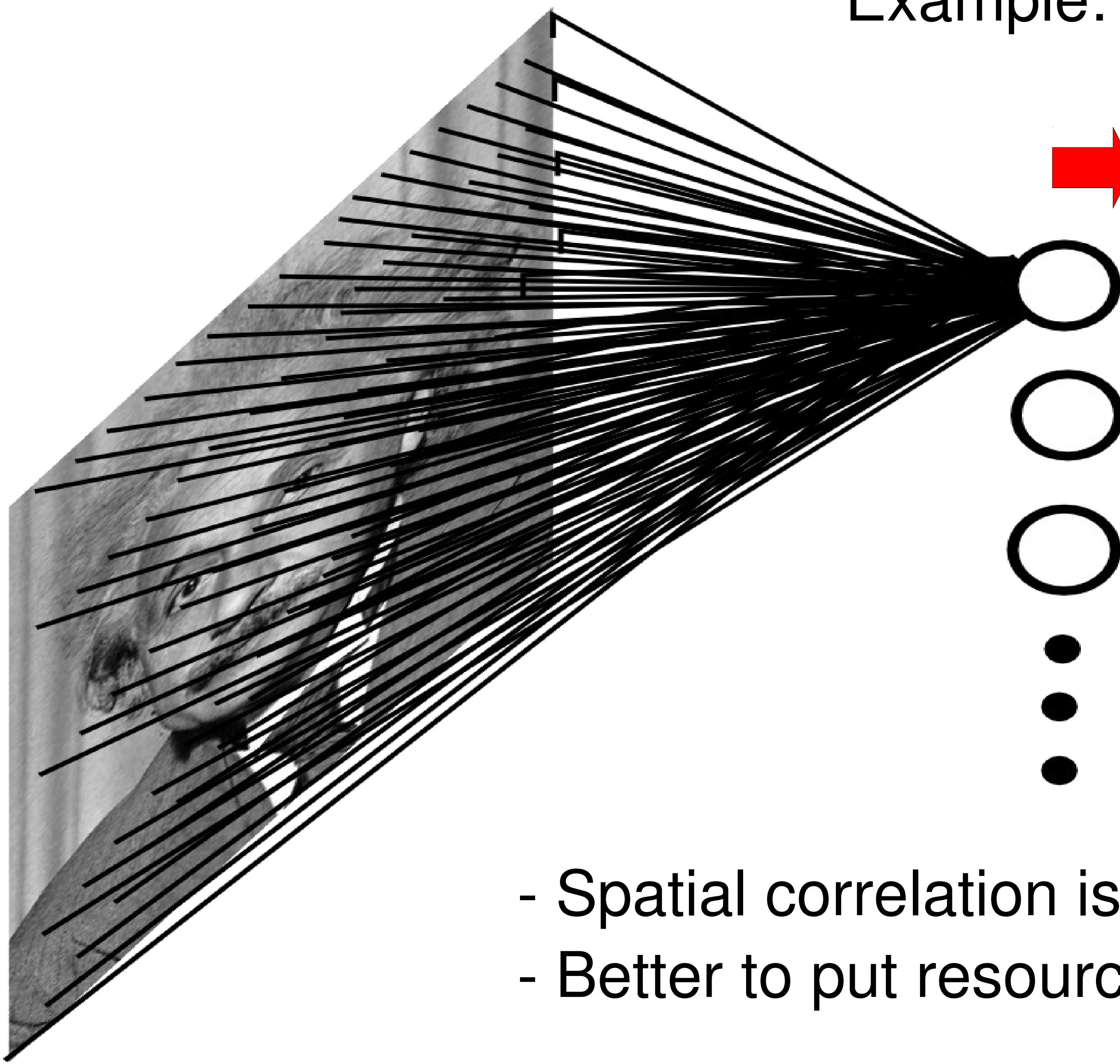
Toy Code: Neural Net Trainer

```
% F-PROP
for i = 1 : nr_layers - 1
  [h{i}  jac{i}]  =  nonlinearity(W{i} * h{i-1} +  b{i});
end
h{nr_layers-1}  =  W{nr_layers-1} * h{nr_layers-2}  +   b{nr_layers-1};
prediction  =  softmax(h{l-1});


% CROSS ENTROPY LOSS
loss  =  -  sum(sum(log(prediction)  .*  target)) / batch_size;


% B-PROP
dh{l-1}  =  prediction  -  target;
for i = nr_layers - 1 : -1 : 1
  Wgrad{i}  =  dh{i} * h{i-1}';
  bgrad{i}  =  sum(dh{i}, 2);
  dh{i-1}  =  (W{i}' * dh{i})  .*  jac{i-1};
end


% UPDATE
for i = 1 : nr_layers - 1
  W{i}  =  W{i}  -  (lr / batch_size)  *  Wgrad{i};
  b{i}  =  b{i}  -  (lr / batch_size)  *  bgrad{i};
end
```

KEY IDEAS: Training NNets

- Neural Net = stack of feature detectors
- F-Prop / B-Prop
- Learning by SGD

FULLY CONNECTED NEURAL NET



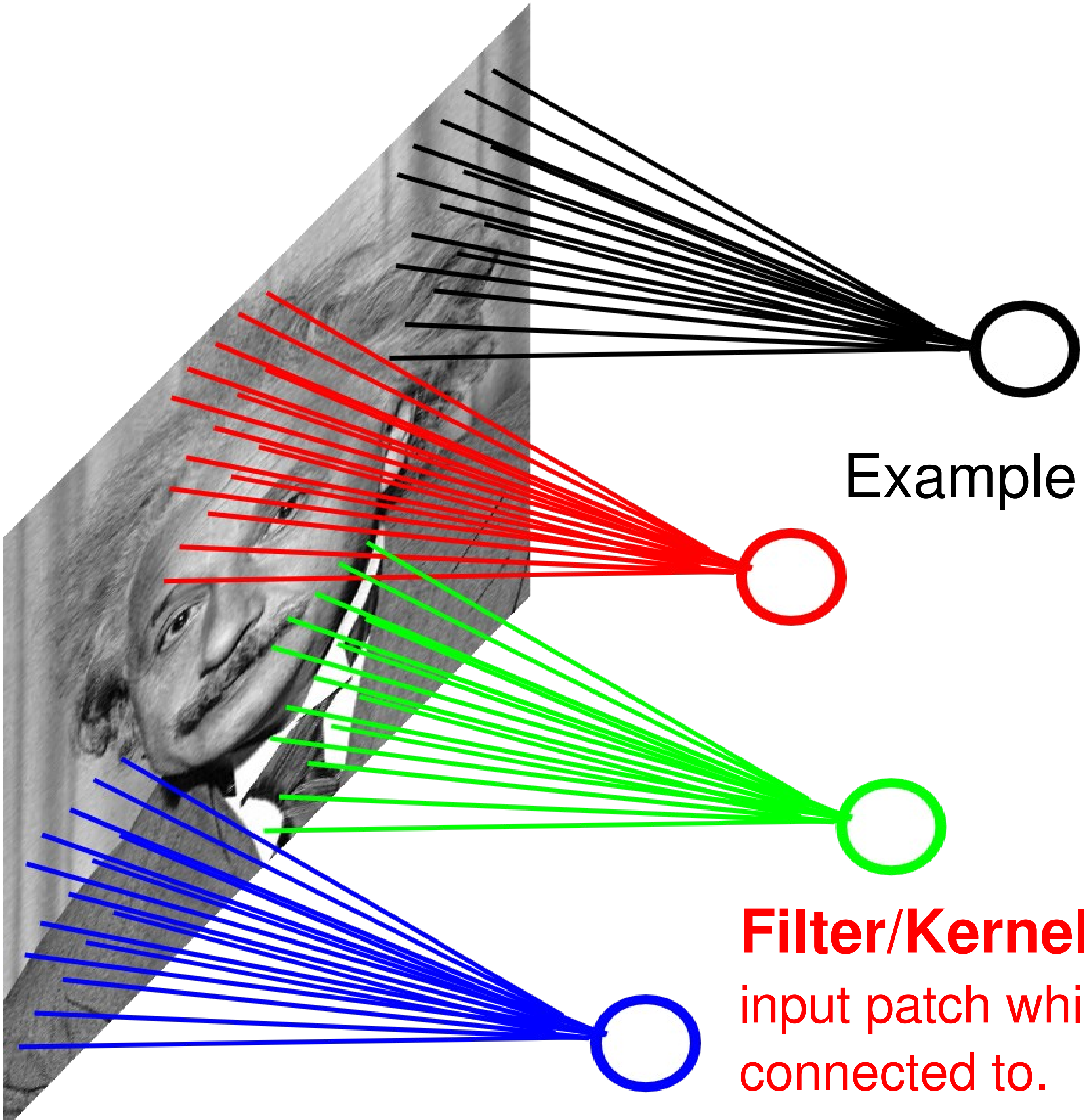
Example: 1000x1000 image

1M hidden units

→ **10^12 parameters**!!!

- Spatial correlation is local
- Better to put resources elsewhere!

LOCALLY CONNECTED NEURAL NET

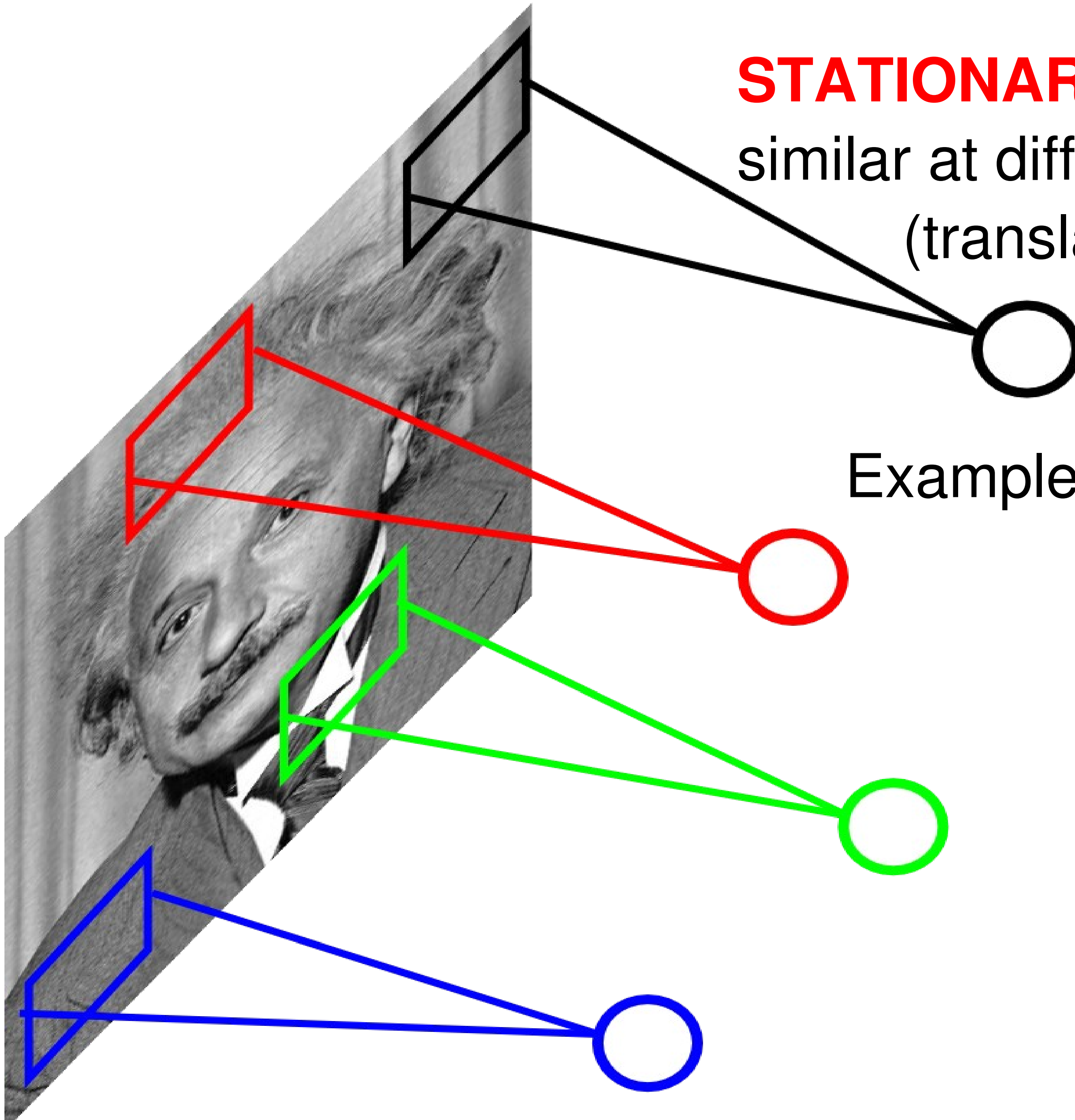


Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Filter/Kernel/Receptive field:
input patch which the hidden unit is connected to.

Ranzato

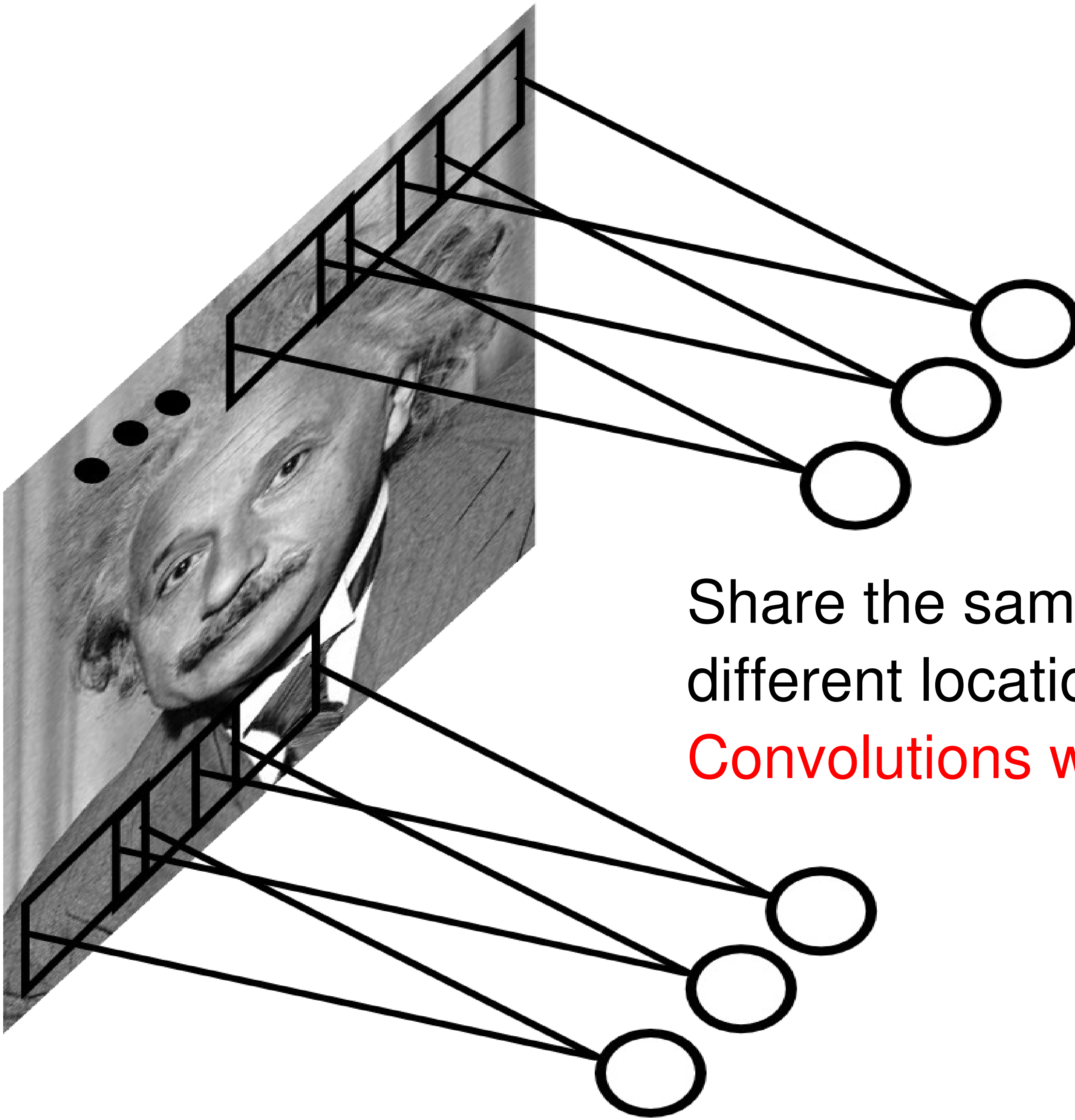
LOCALLY CONNECTED NEURAL NET



STATIONARITY? Statistics are similar at different locations (translation invariance)

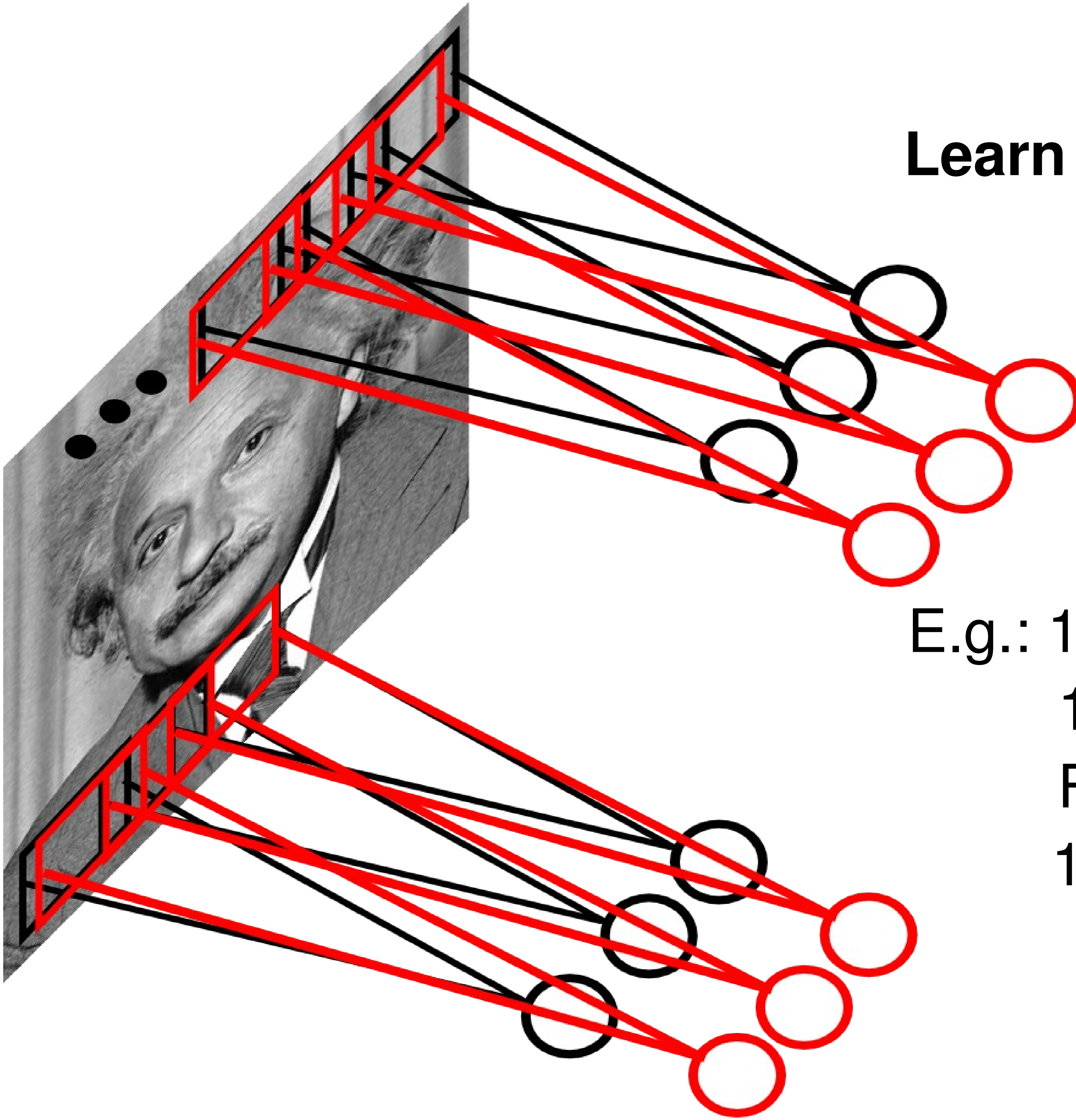
Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

CONVOLUTIONAL NET



Share the same parameters across different locations:
Convolutions with learned kernels

CONVOLUTIONAL NET



Learn multiple filters.

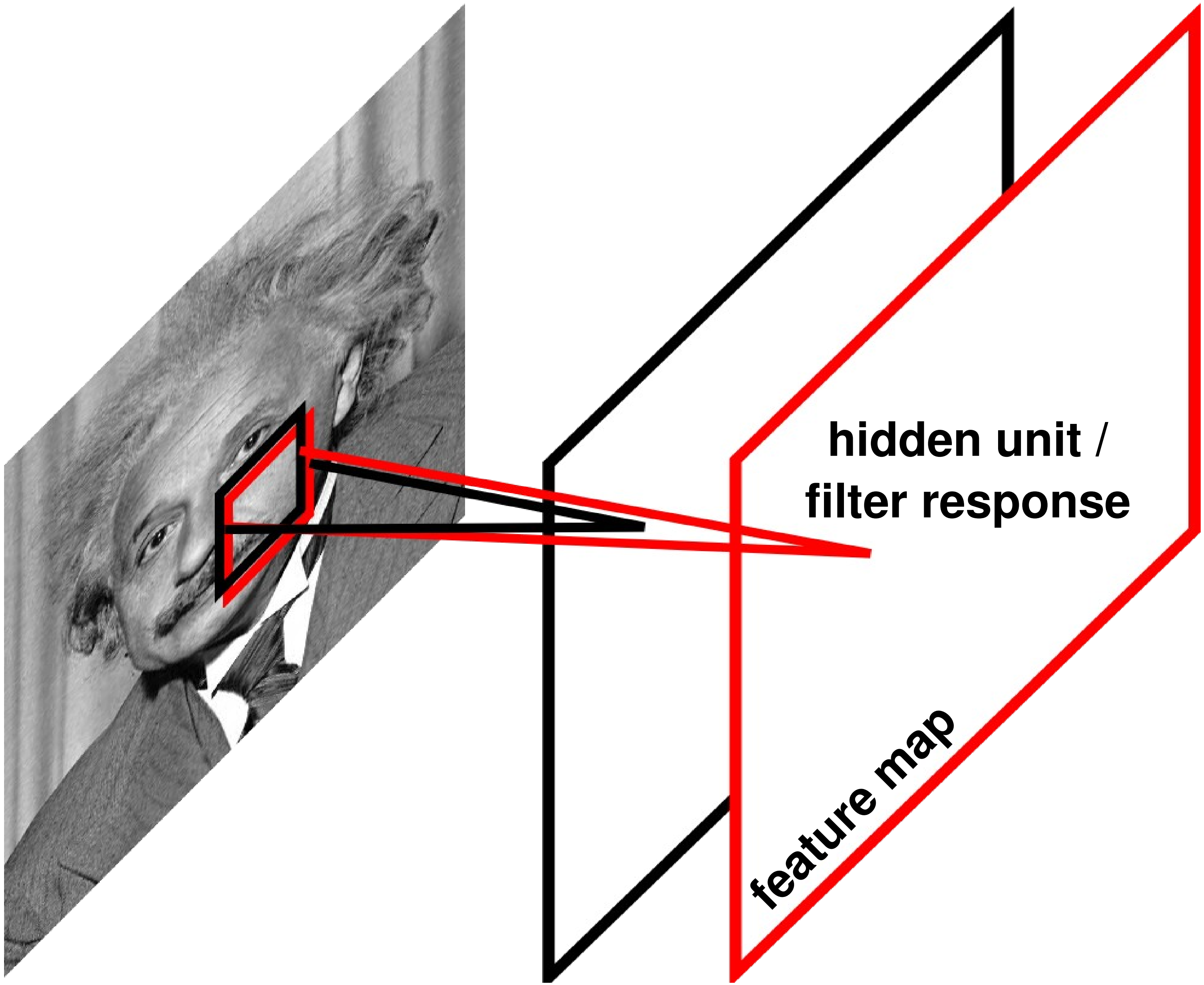
E.g.: 1000x1000 image

100 Filters

Filter size: 10x10

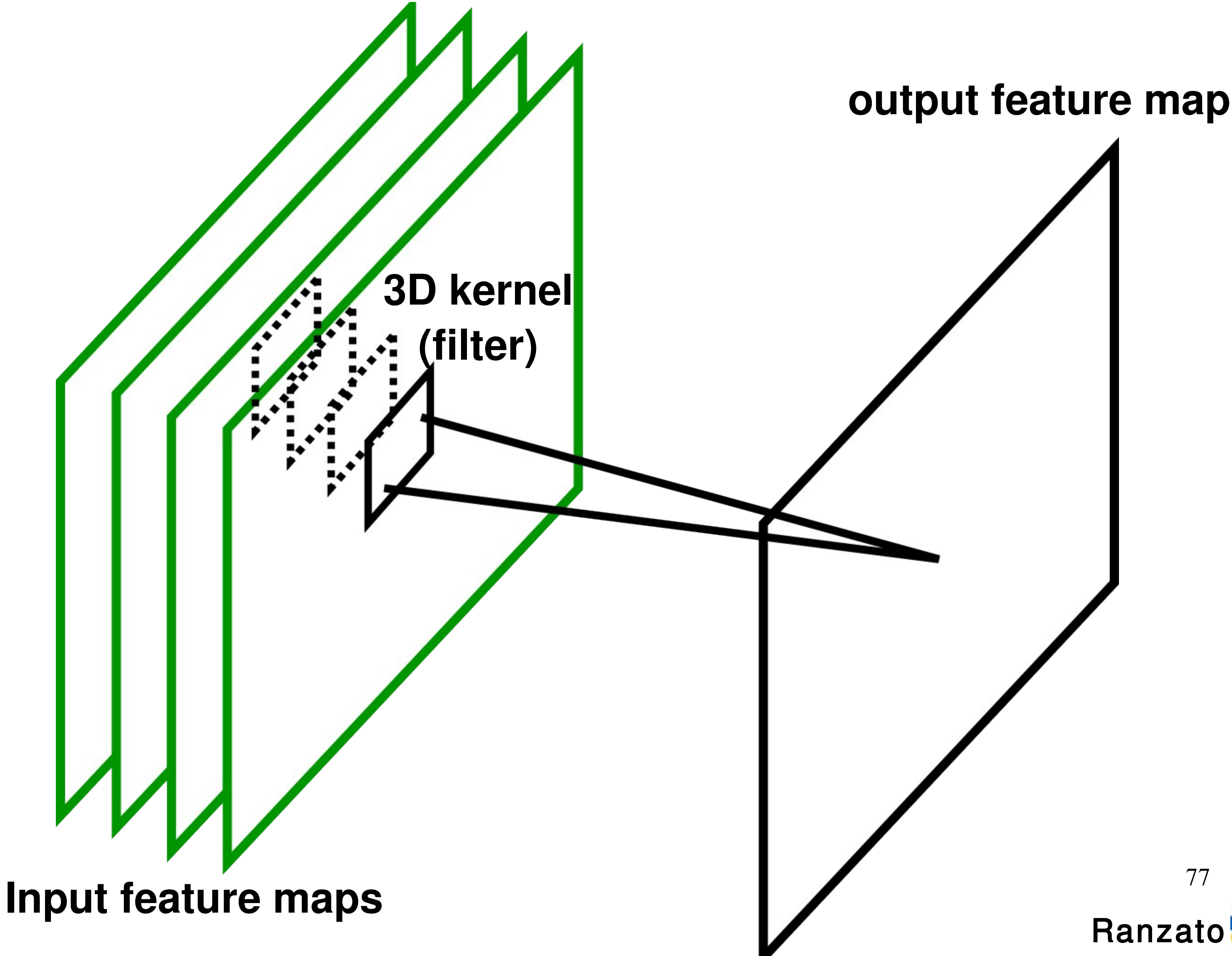
10K parameters

CONVOLUTIONAL NET

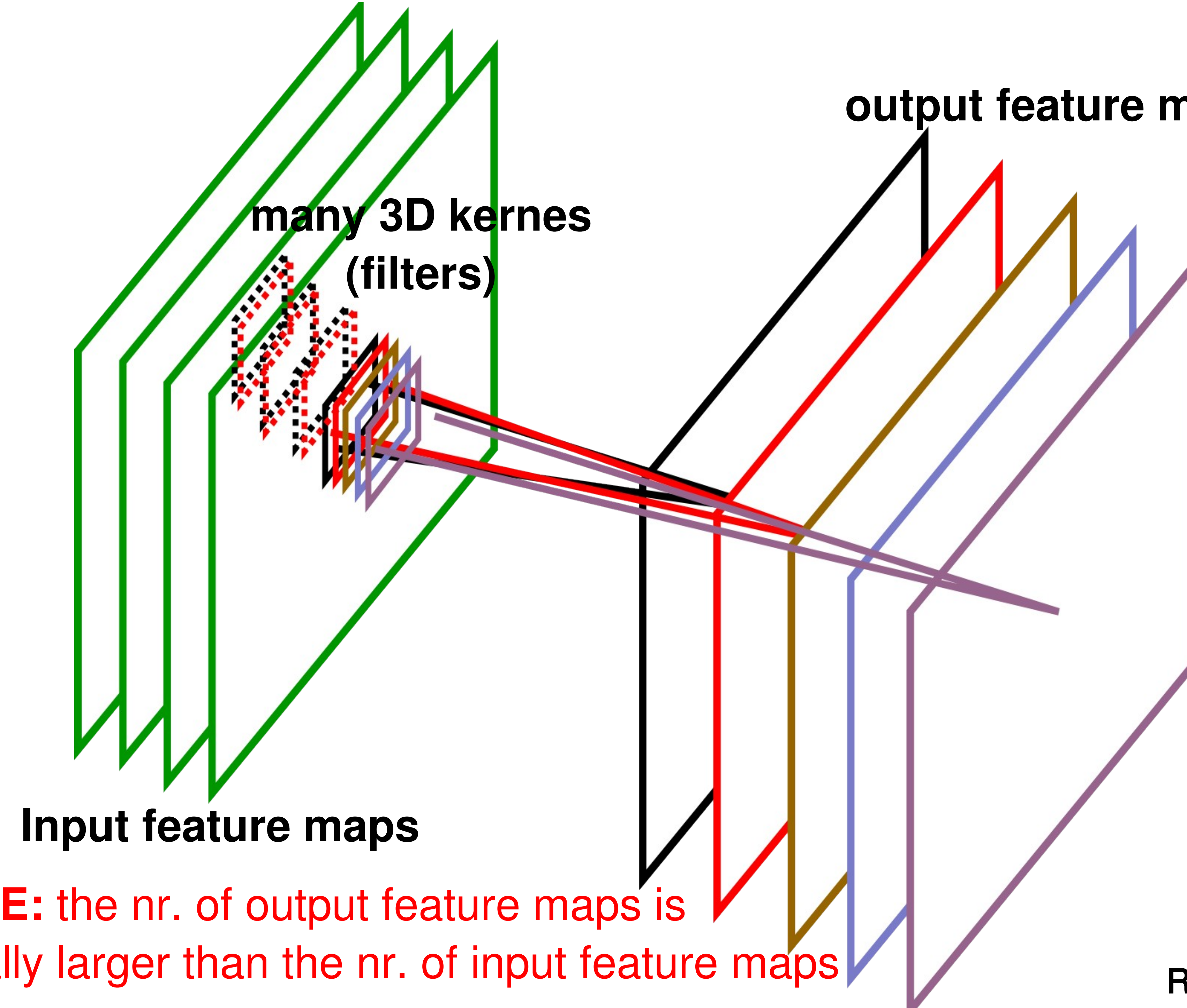


Ranzato

CONVOLUTIONAL LAYER



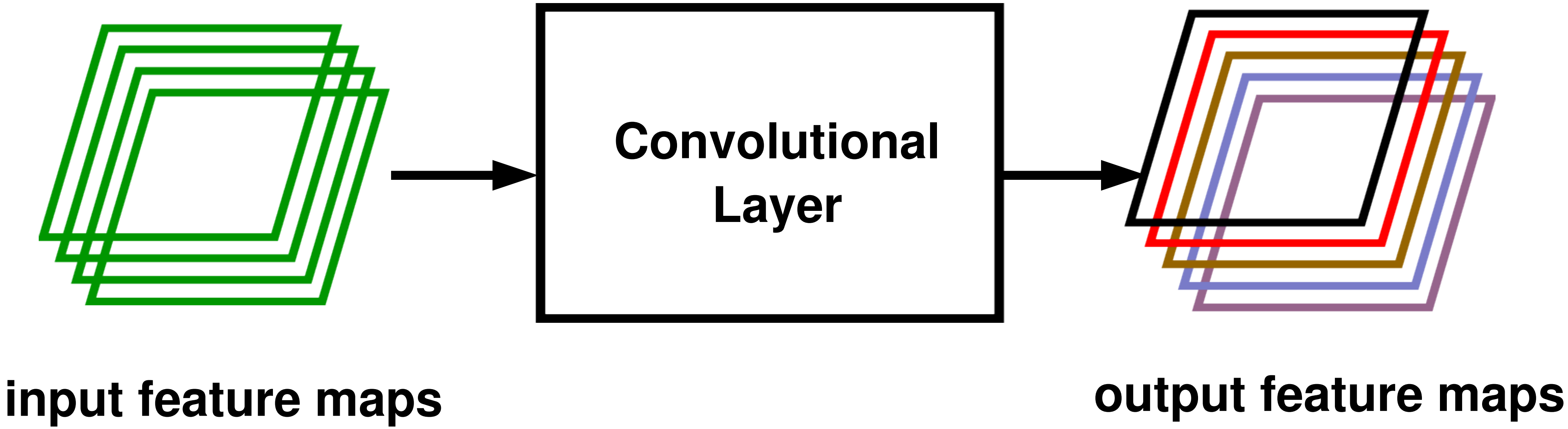
CONVOLUTIONAL LAYER



NOTE: the nr. of output feature maps is usually larger than the nr. of input feature maps

Ranzato

CONVOLUTIONAL LAYER



NOTE: the nr. of output feature maps is usually larger than the nr. of input feature maps

KEY IDEAS: CONV. NETS

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 hidden units

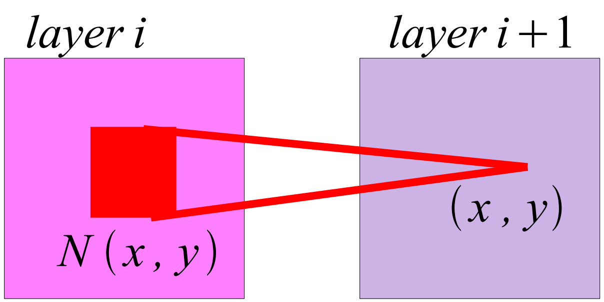
This is called: **convolutional network.**

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

SPECIAL LAYERS

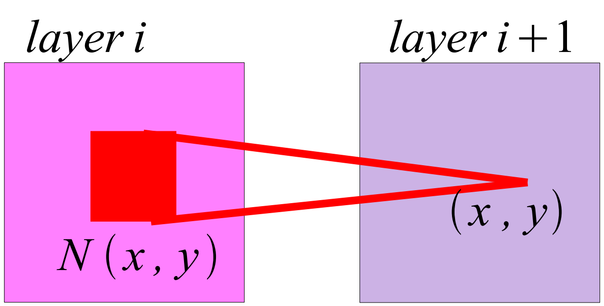
Over the years, some new modules have proven to be very effective when plugged into conv-nets:

- **Pooling (**average, L2, max**)**



$$\boldsymbol{h}_{i+1,x,y} = max_{(j,k)\in N(x,y)} \boldsymbol{h}_{i,j,k}$$

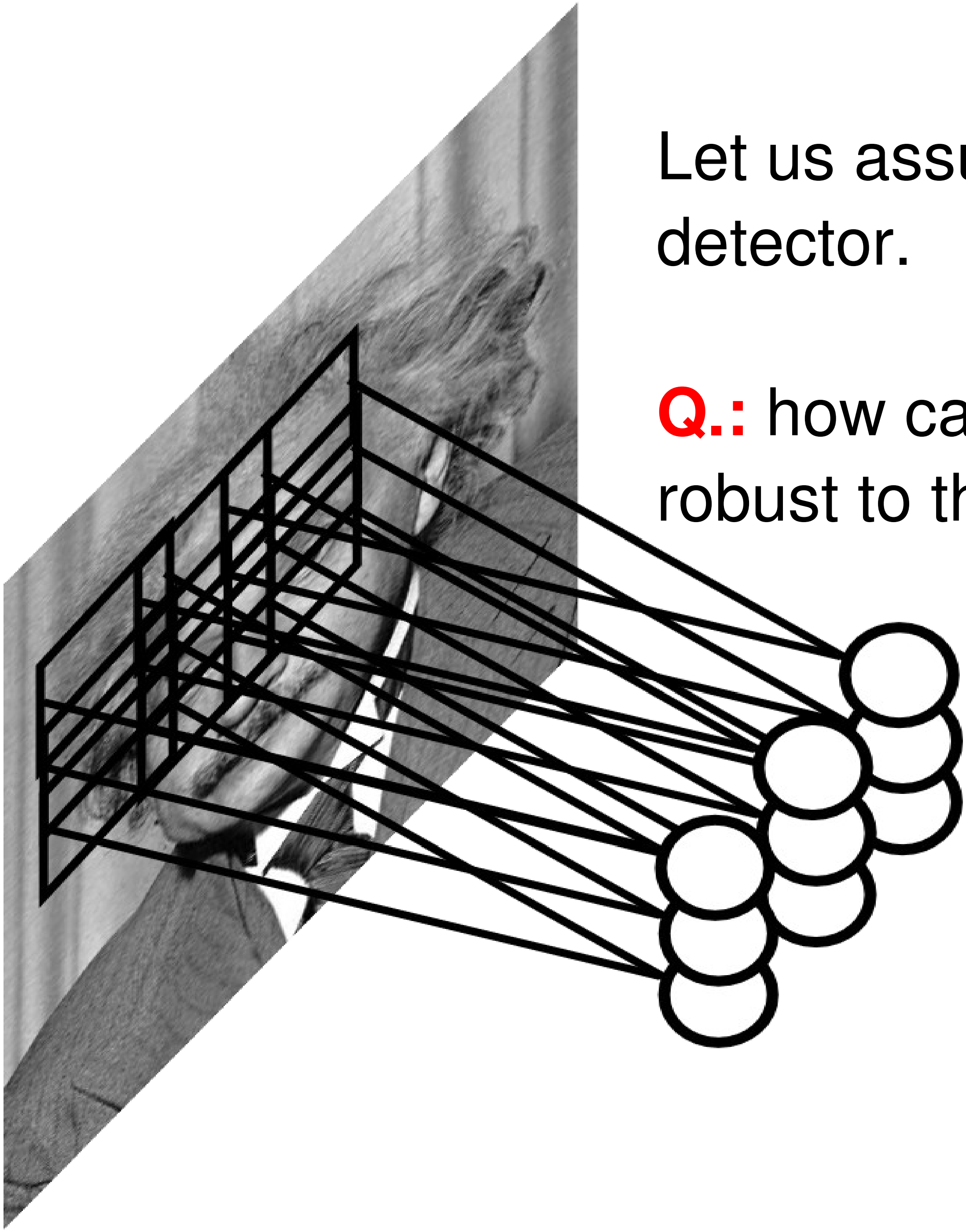
- **Local Contrast Normalization (**over space / features**)**



$$\boldsymbol{h}_{i+1,x,y} = \frac{\boldsymbol{h}_{i,x,y} - m_{i,x,y}}{\sigma_{i,x,y}}$$

Jarrett et al. “What is the best multi-stage architecture...?” ICCV 2009

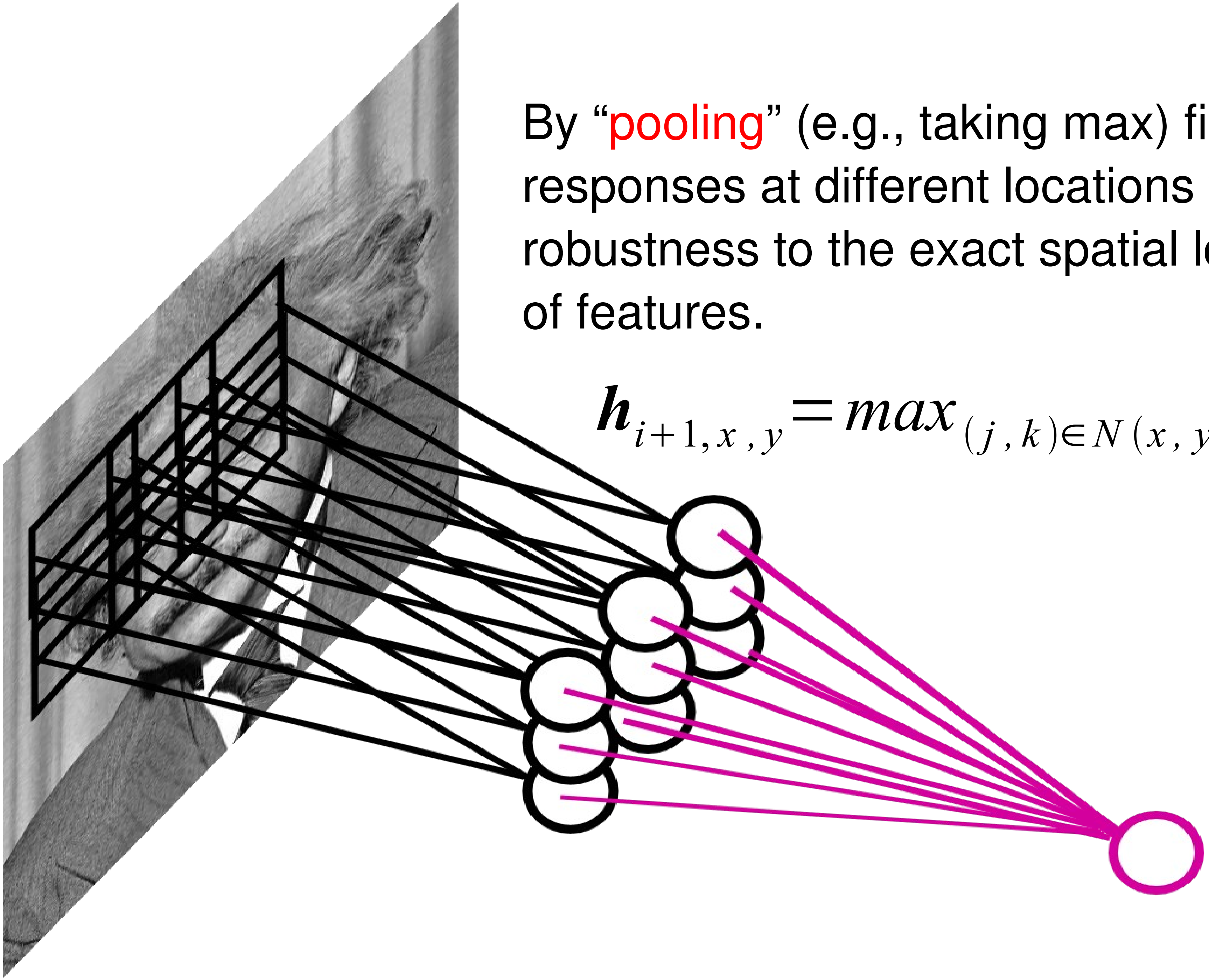
POOLING



Let us assume filter is an “eye” detector.

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

POOLING



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

$$h_{i+1,x,y} = max_{(j,k) \in N(x,y)} h_{i,j,k}$$

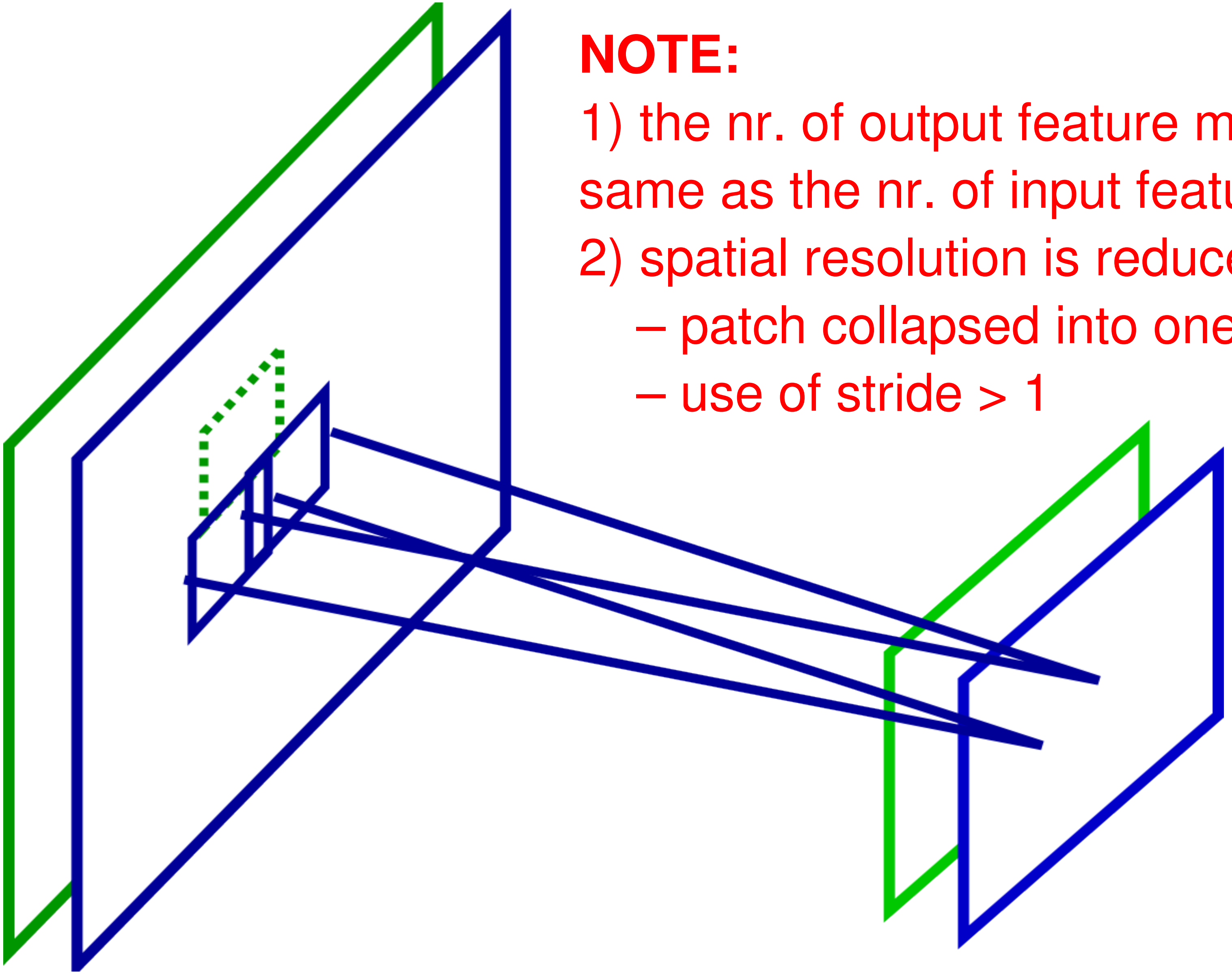
Ranzato

POOLING LAYER

NOTE:

1) the nr. of output feature maps is the same as the nr. of input feature maps

2) spatial resolution is reduced
   - patch collapsed into one value
   - use of stride > 1



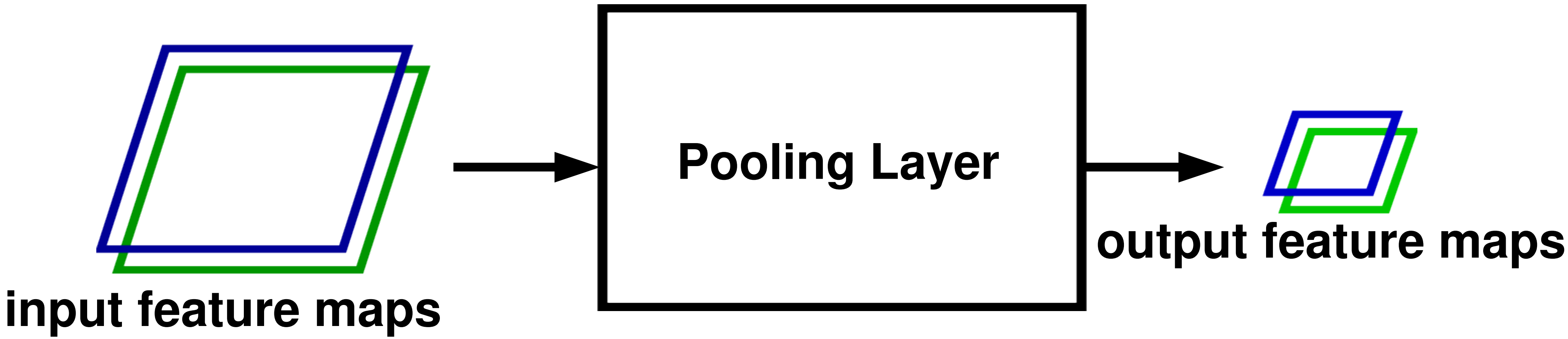
Input feature maps **output feature maps**

POOLING LAYER

NOTE:

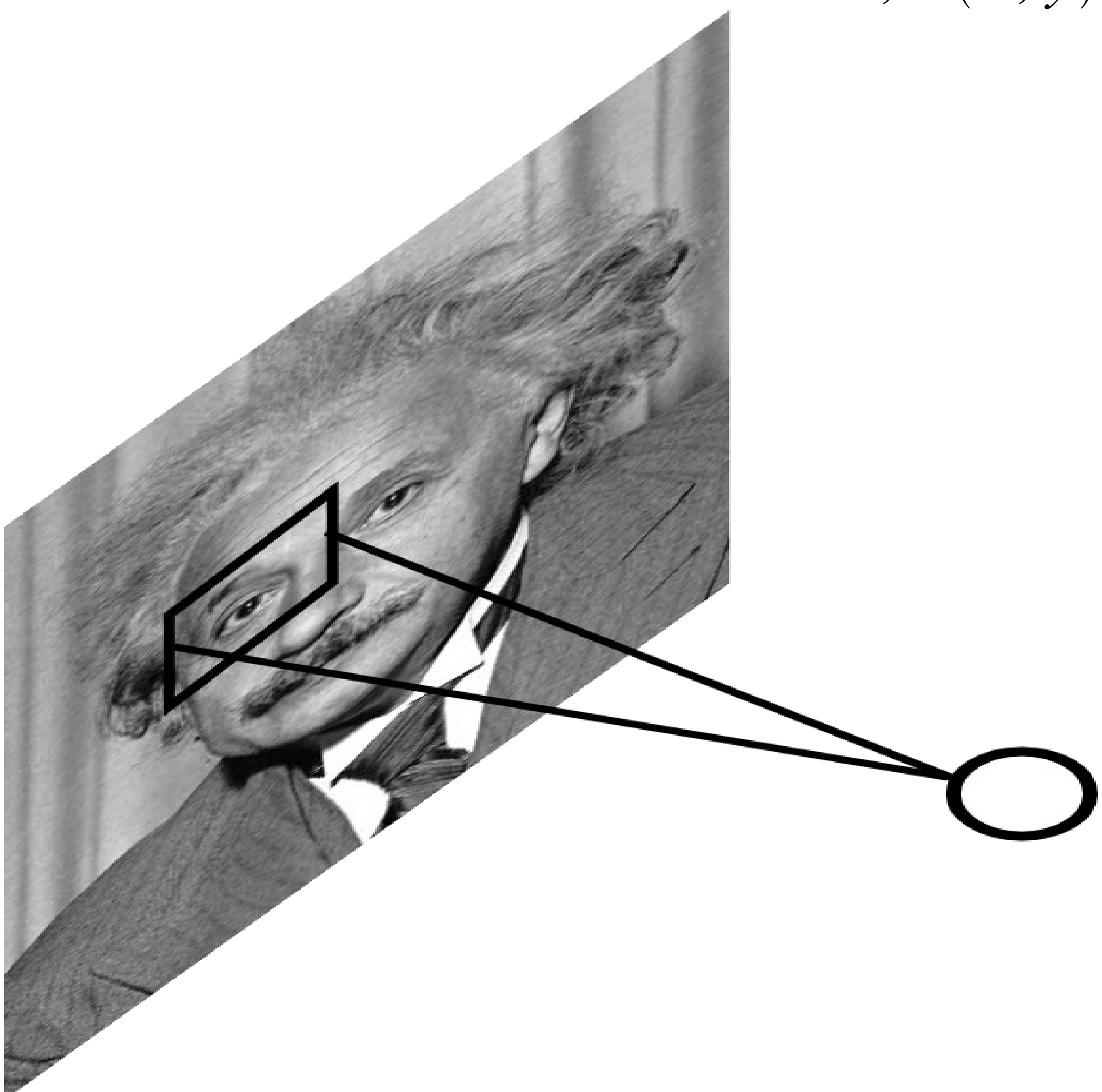
1) the nr. of output feature maps is the same as the nr. of input feature maps

2) spatial resolution is reduced
   - patch collapsed into one value
   - use of stride > 1



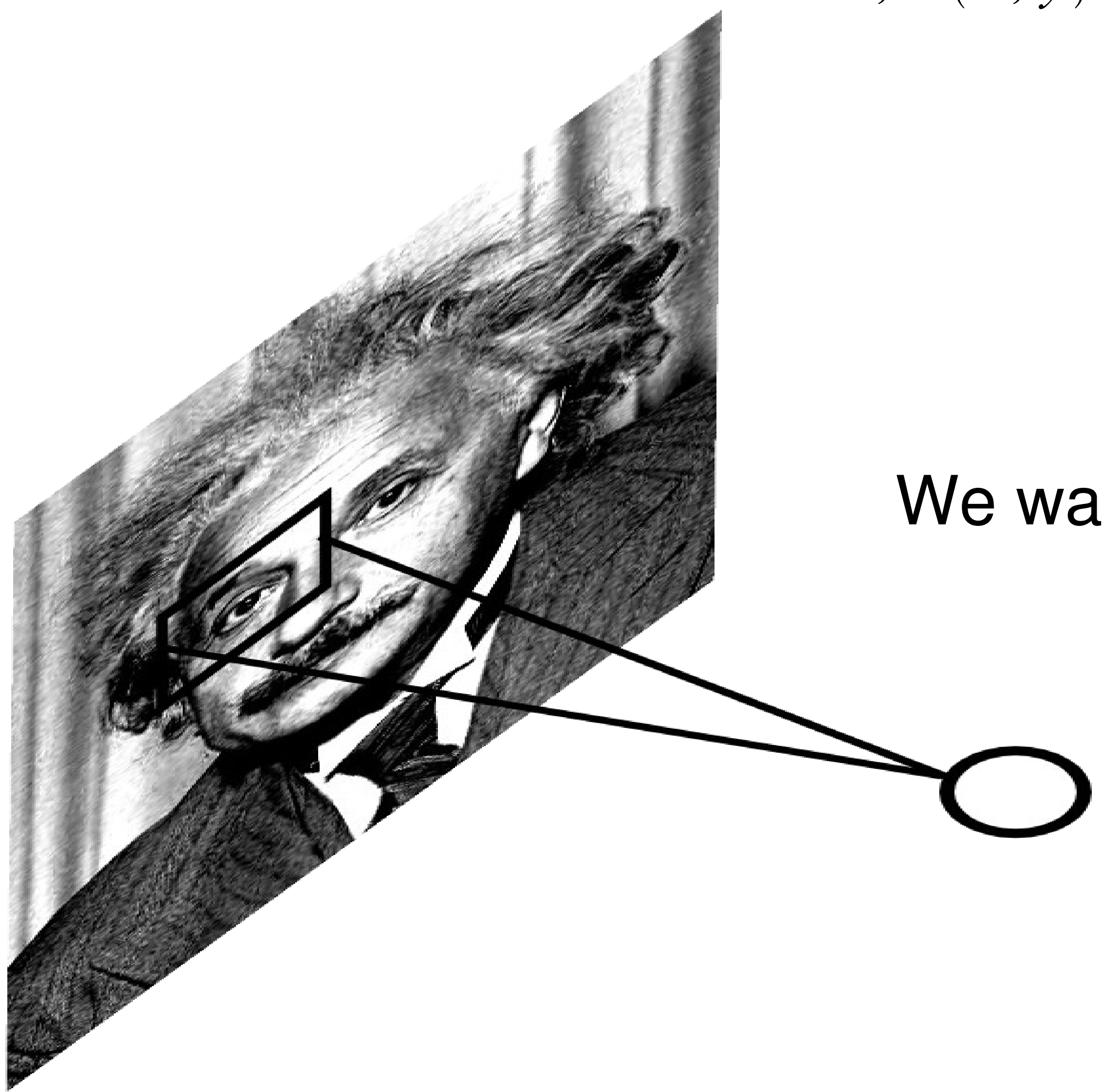
LOCAL CONTRAST NORMALIZATION

$$h_{i+1,x,y} = \frac{h_{i,x,y} - m_{i,N(x,y)}}{\sigma_{i,N(x,y)}}$$



LOCAL CONTRAST NORMALIZATION

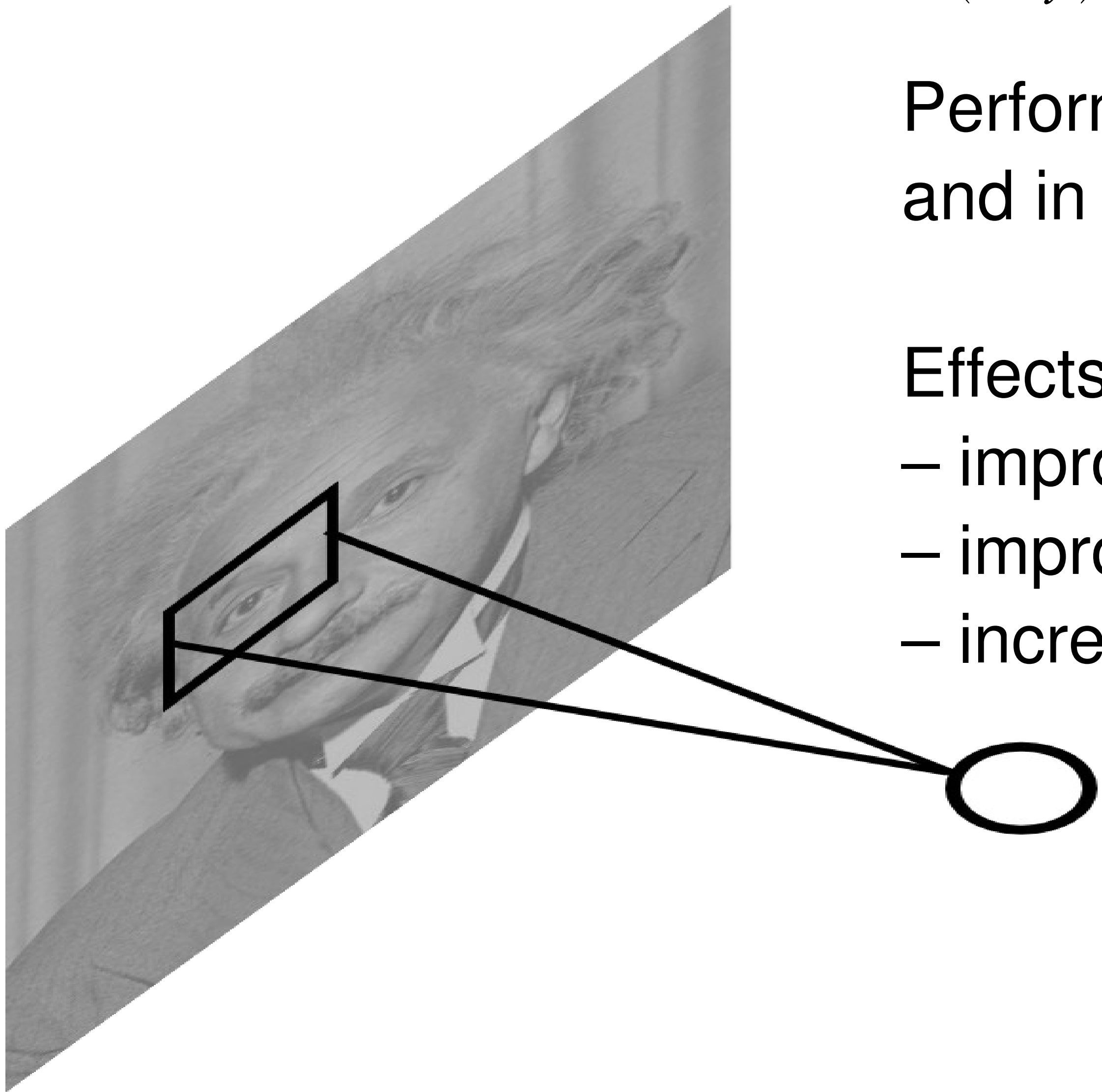
$$h_{i+1,x,y} = \frac{h_{i,x,y} - m_{i,N(x,y)}}{\sigma_{i,N(x,y)}}$$



We want the same response.

LOCAL CONTRAST NORMALIZATION

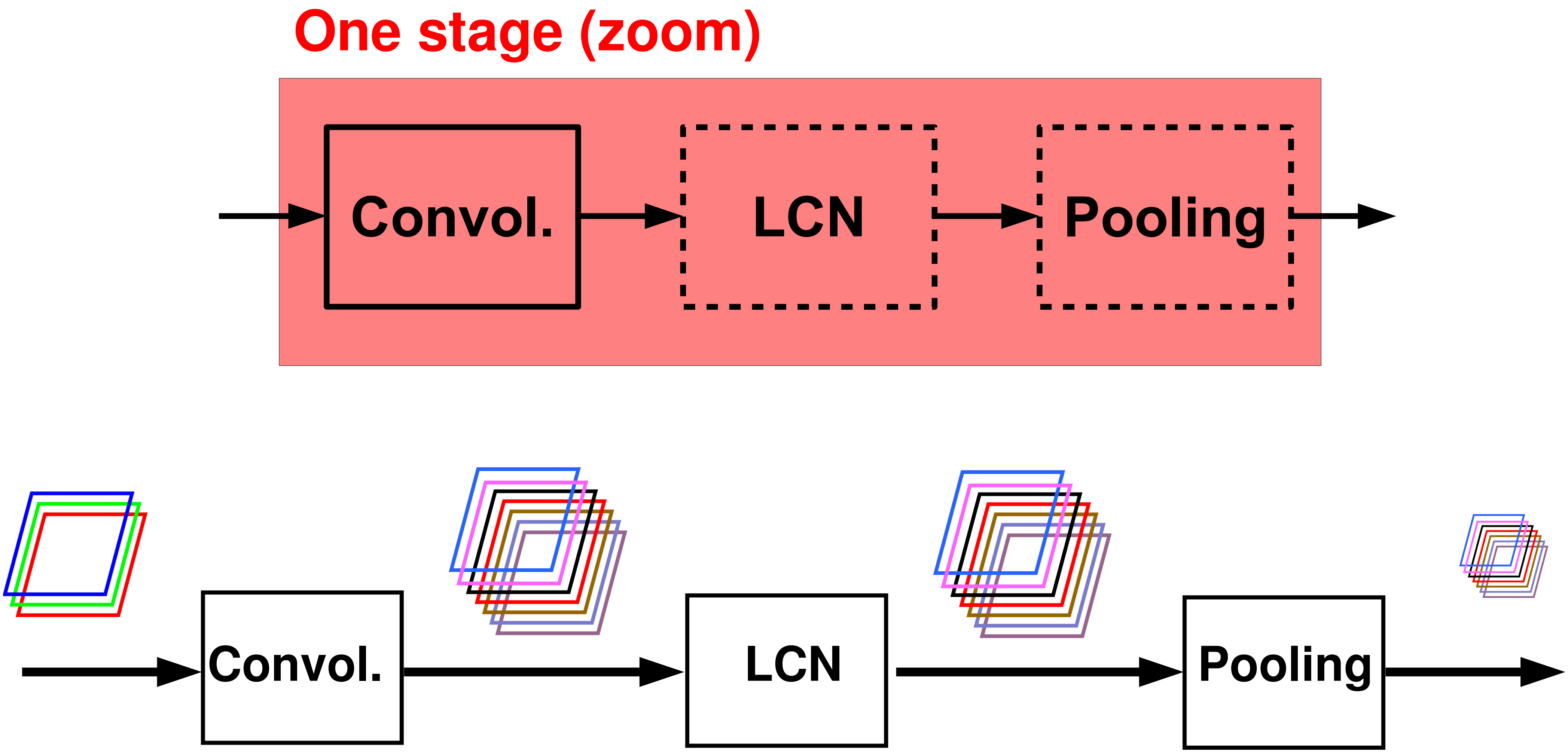
$$h_{i+1,x,y} = \frac{h_{i,x,y} - m_{i,N(x,y)}}{\sigma_{i,N(x,y)}}$$



Performed also across features and in the higher layers.

Effects:
– improves invariance
– improves optimization
– increases sparsity

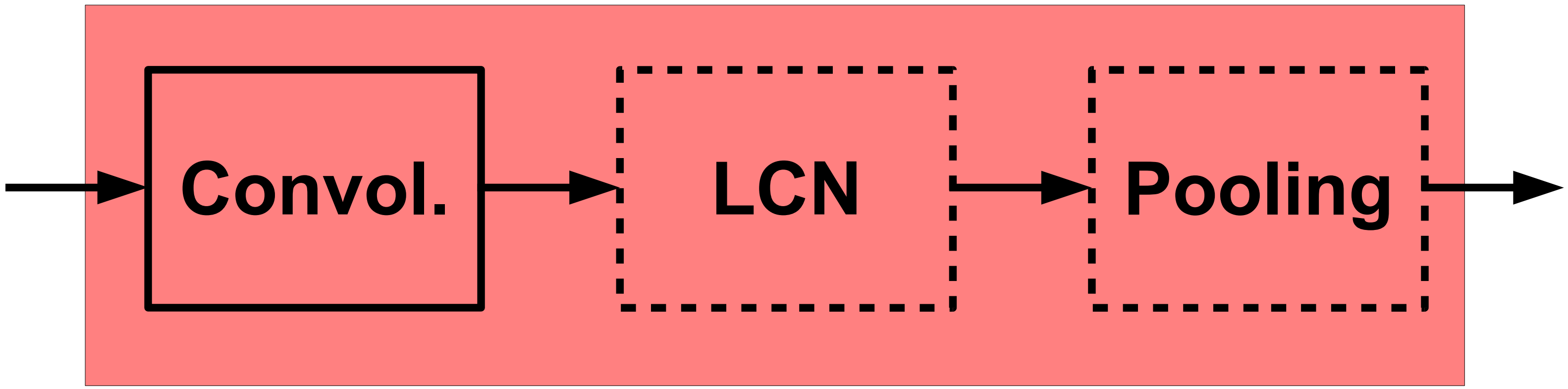
CONV NETS: TYPICAL ARCHITECTURE

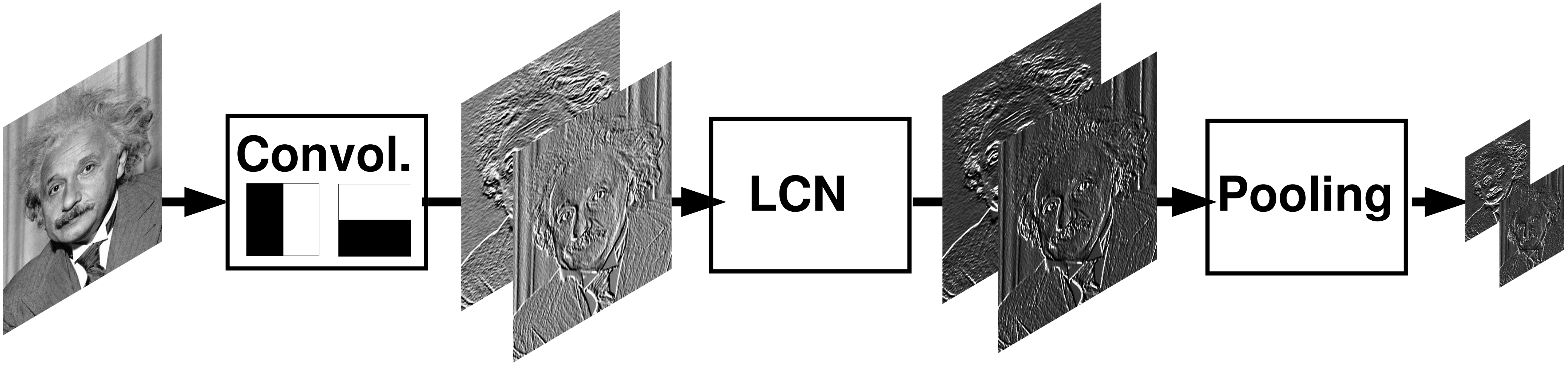


Convolutional layer increases nr. feature maps.
Pooling layer decreases spatial resolution.

CONV NETS: TYPICAL ARCHITECTURE

One stage (zoom)

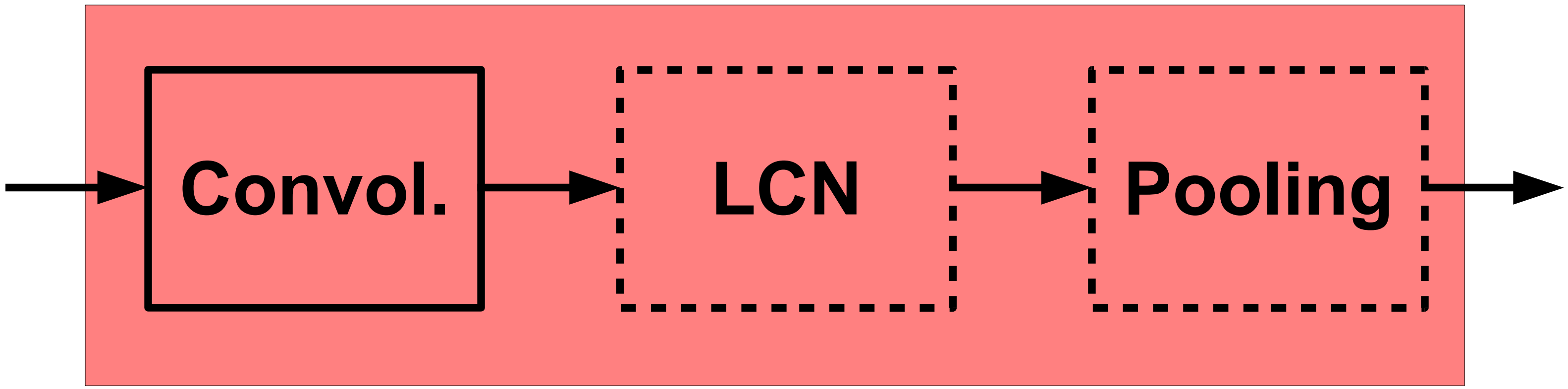


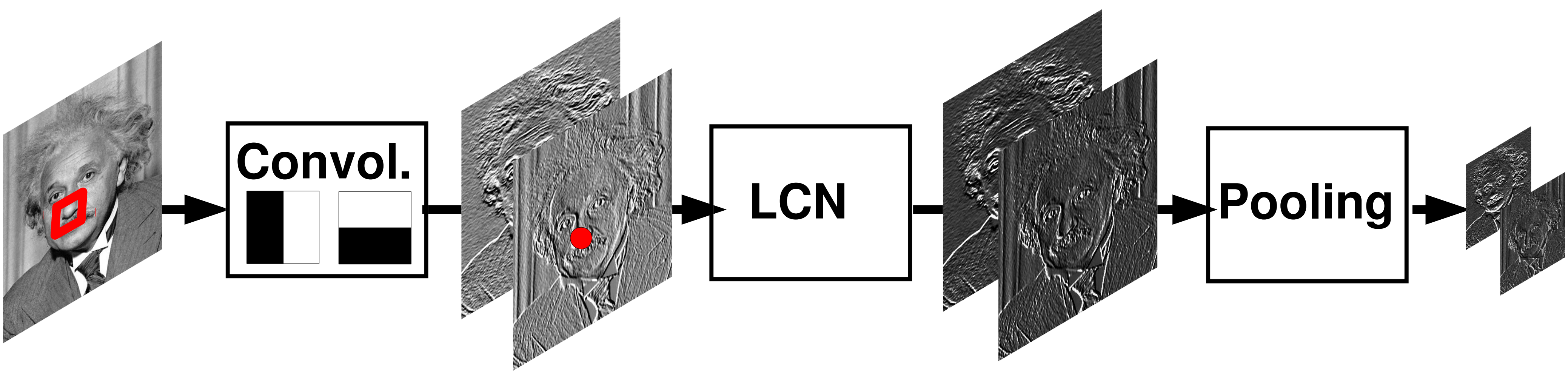


Example with only two filters.

CONV NETS: TYPICAL ARCHITECTURE

One stage (zoom)

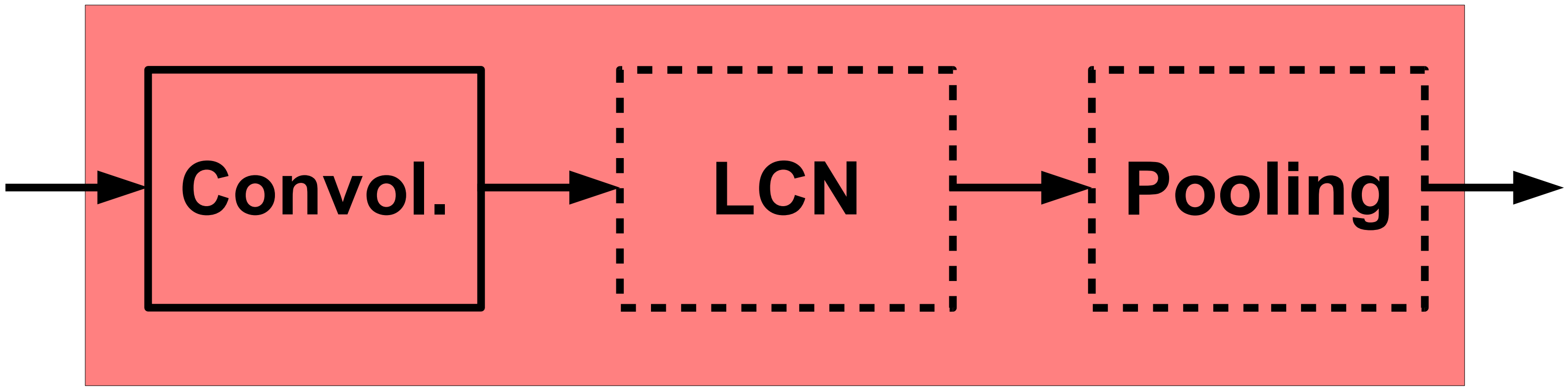


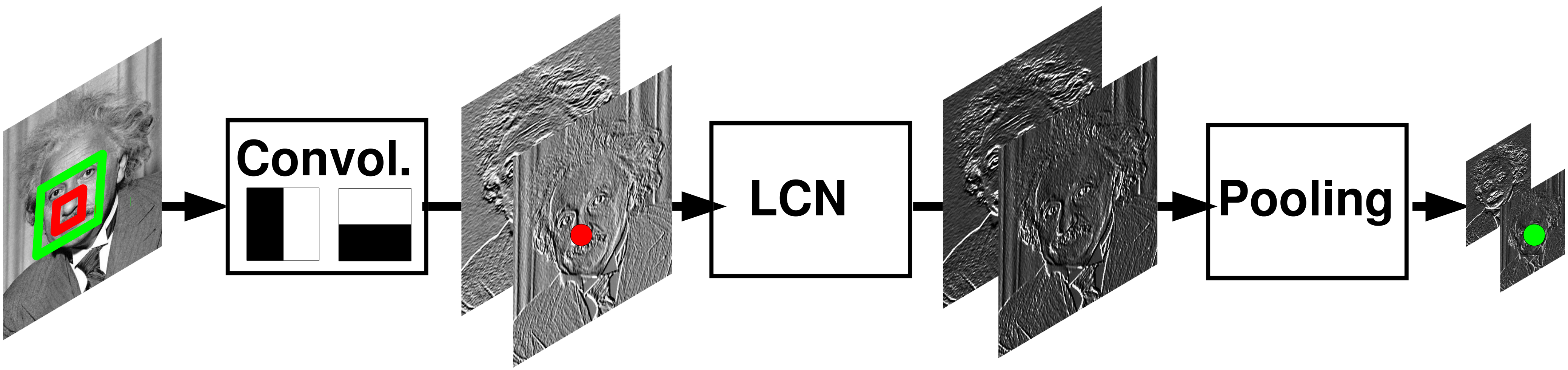


A hidden unit in the first hidden layer is influenced by a small neighborhood (equal to size of filter).

CONV NETS: TYPICAL ARCHITECTURE

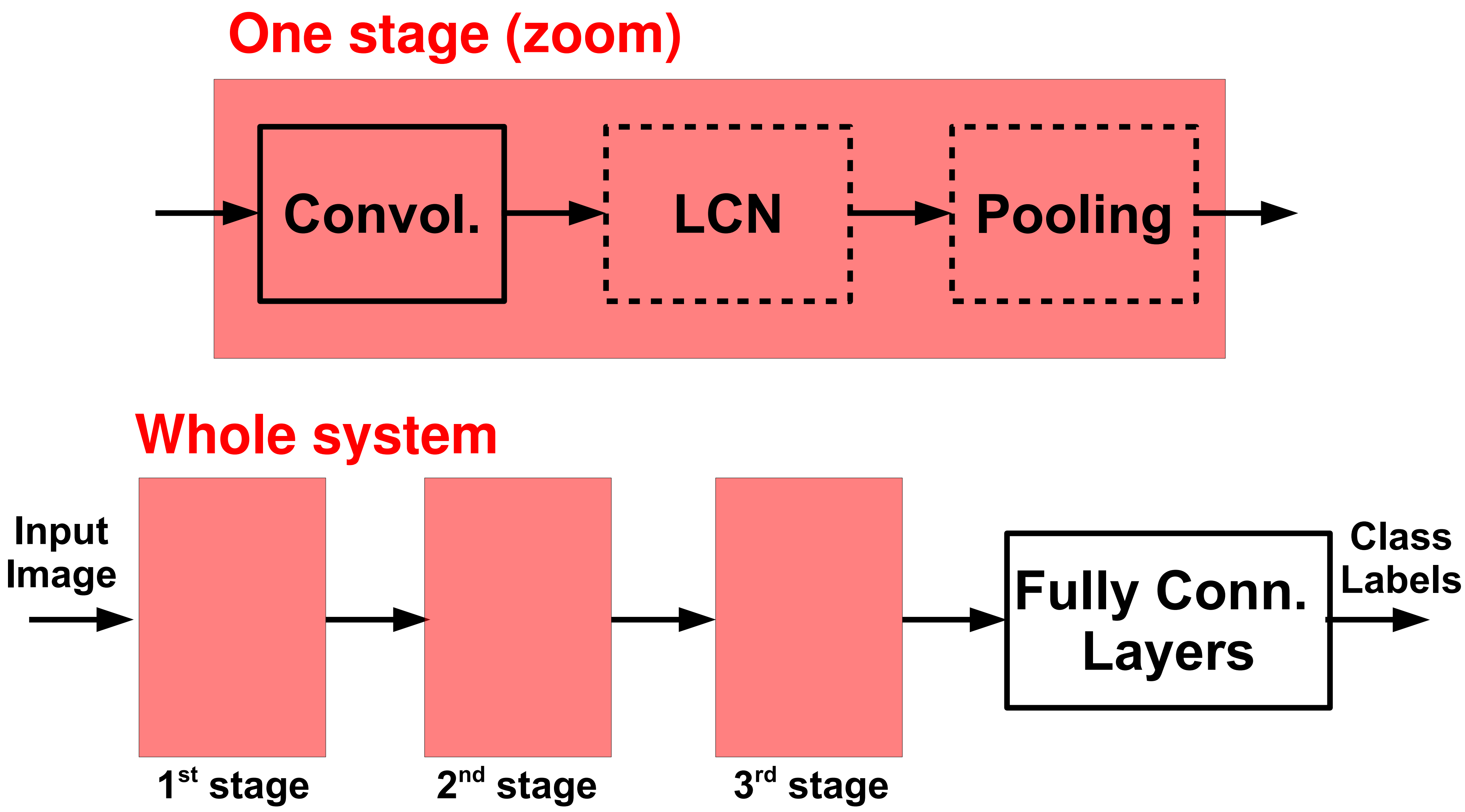
One stage (zoom)





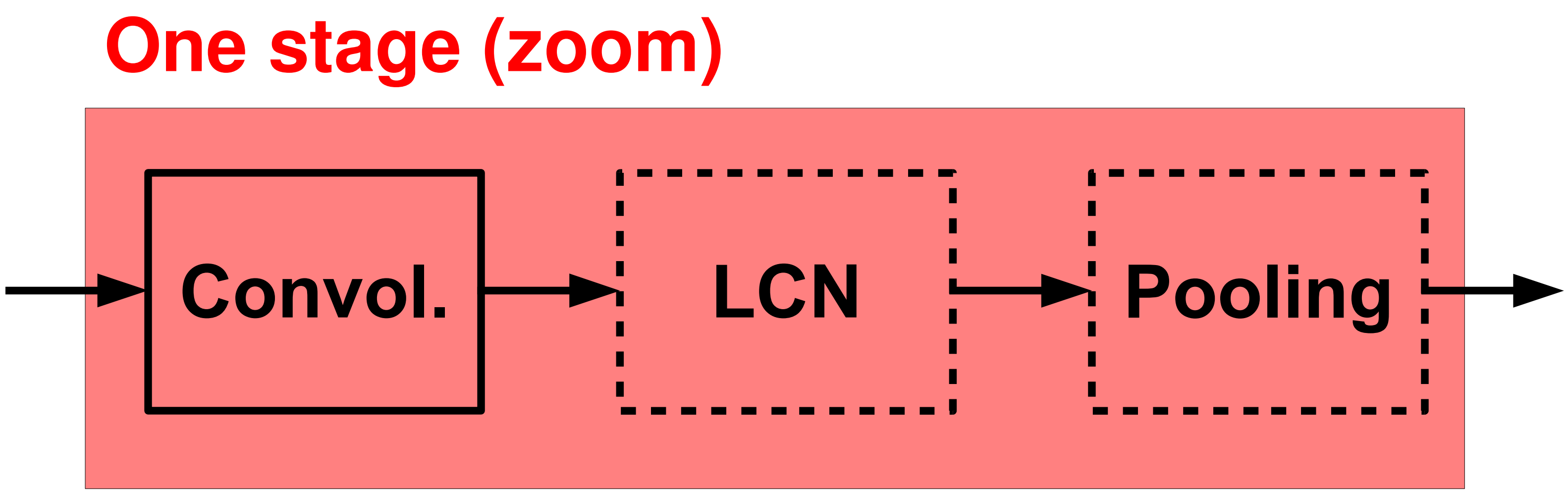
A hidden unit after the pooling layer is influenced by a larger neighborhood (it depends on filter sizes and strides).

CONV NETS: TYPICAL ARCHITECTURE



After a few stages, residual spatial resolution is very small.
We have learned a descriptor for the whole image.

CONV NETS: 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

CONV NETS: TRAINING

All layers are differentiable (a.e.).
We can use standard back-propagation.

Algorithm:

Given a small mini-batch
- F-PROP
- B-PROP
- PARAMETER UPDATE

KEY IDEAS: CONV. NETS

Conv. Nets have special layers like:
– pooling, and
– local contrast normalization
Back-propagation can still be applied.

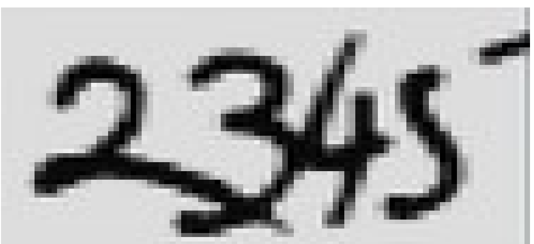
These layers are useful to:
– reduce computational burden
– increase invariance
– ease the optimization

Outline

- Motivation
- Deep Learning: The Big Picture
- From neural nets to convolutional nets
- Applications
- A practical guide

CONV NETS: EXAMPLES

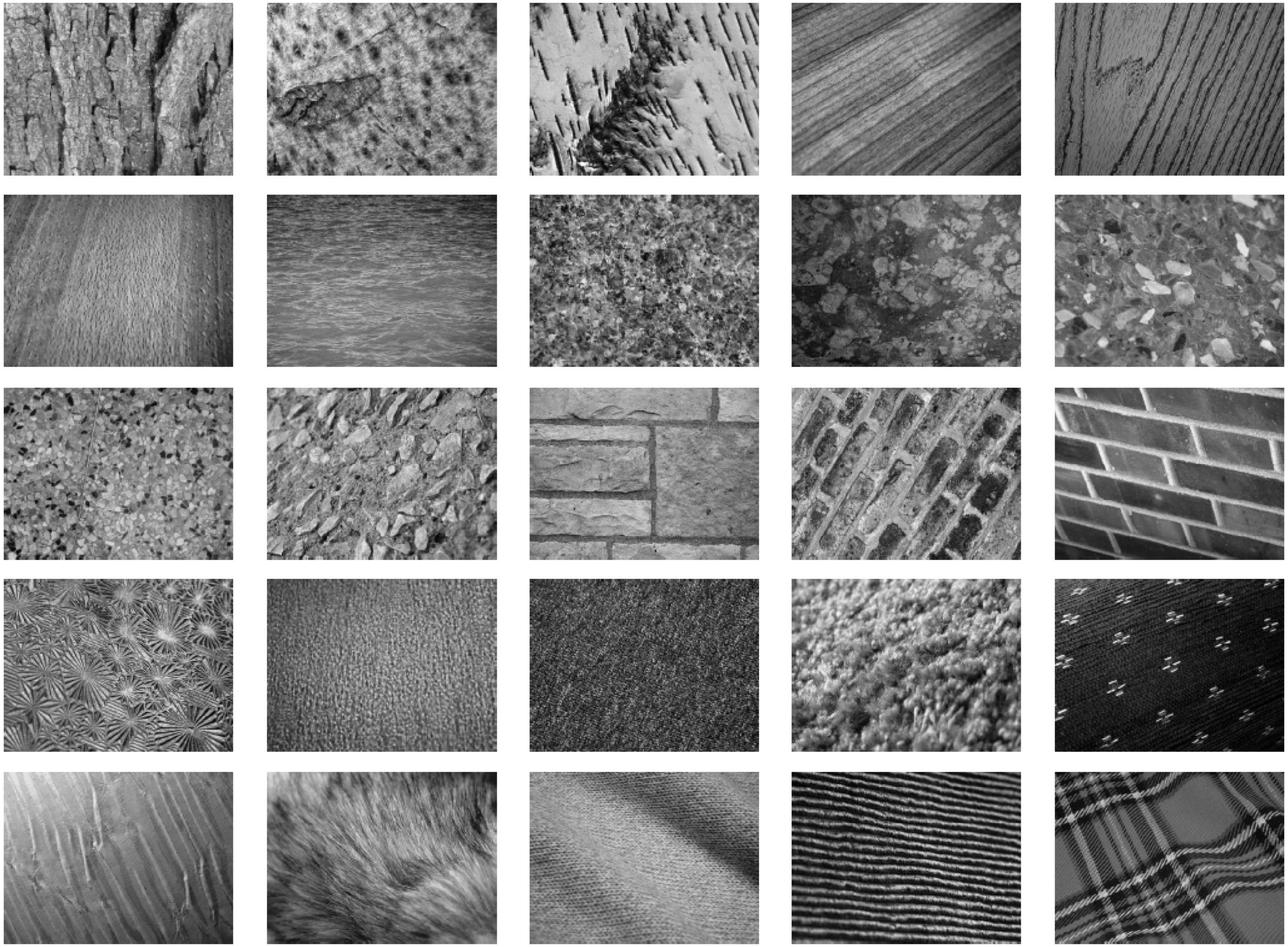
- **OCR / House number & Traffic sign classification**



Ciresan et al. “MCDNN for image classification” CVPR 2012
Wan et al. “Regularization of neural networks using dropconnect” ICML 2013

CONV NETS: EXAMPLES

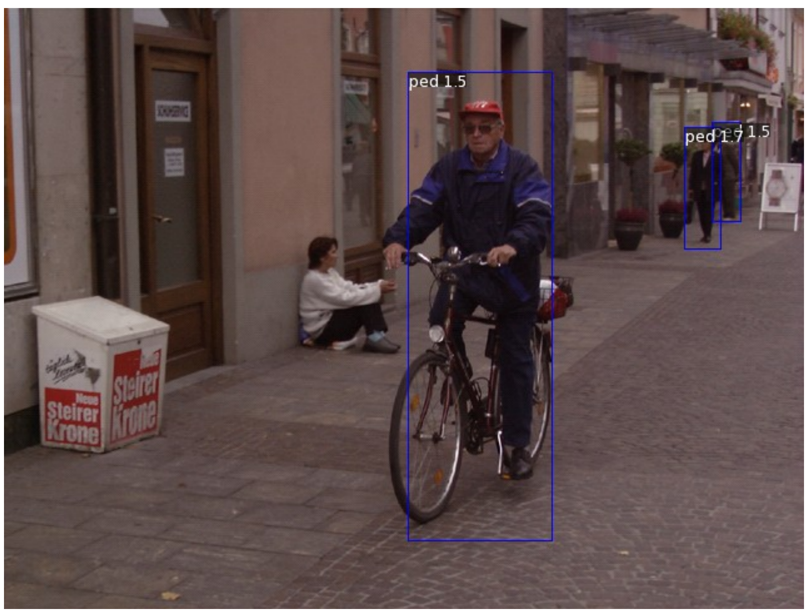
- **Texture classification**

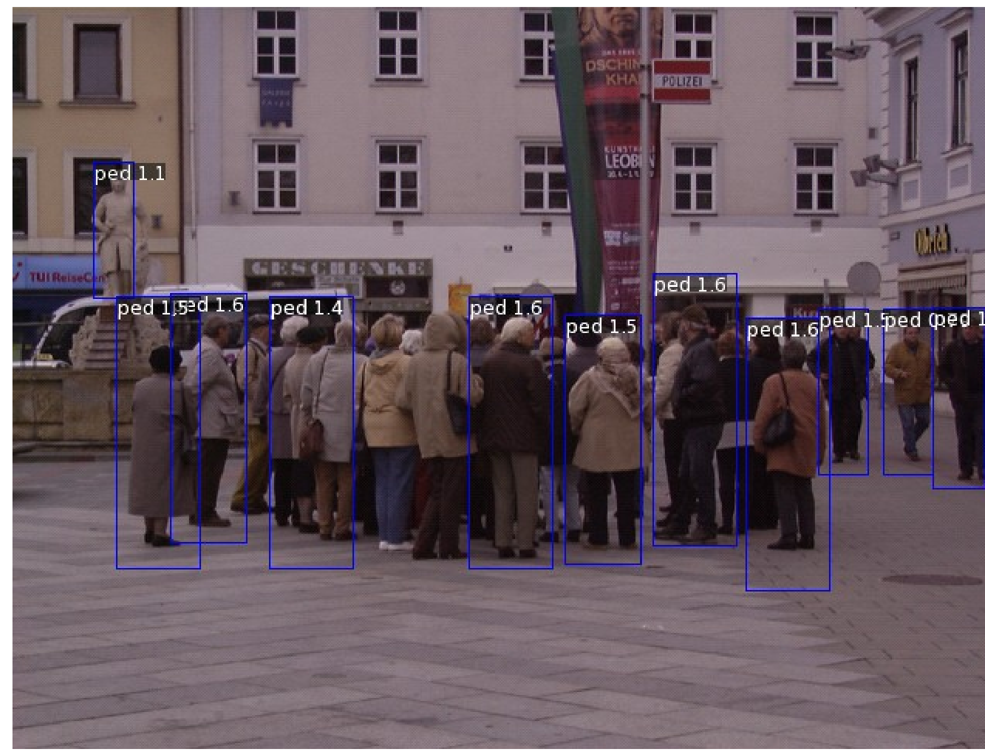


Sifre et al. “Rotation, scaling and deformation invariant scattering...” CVPR 2013

CONV NETS: EXAMPLES

- **Pedestrian detection**

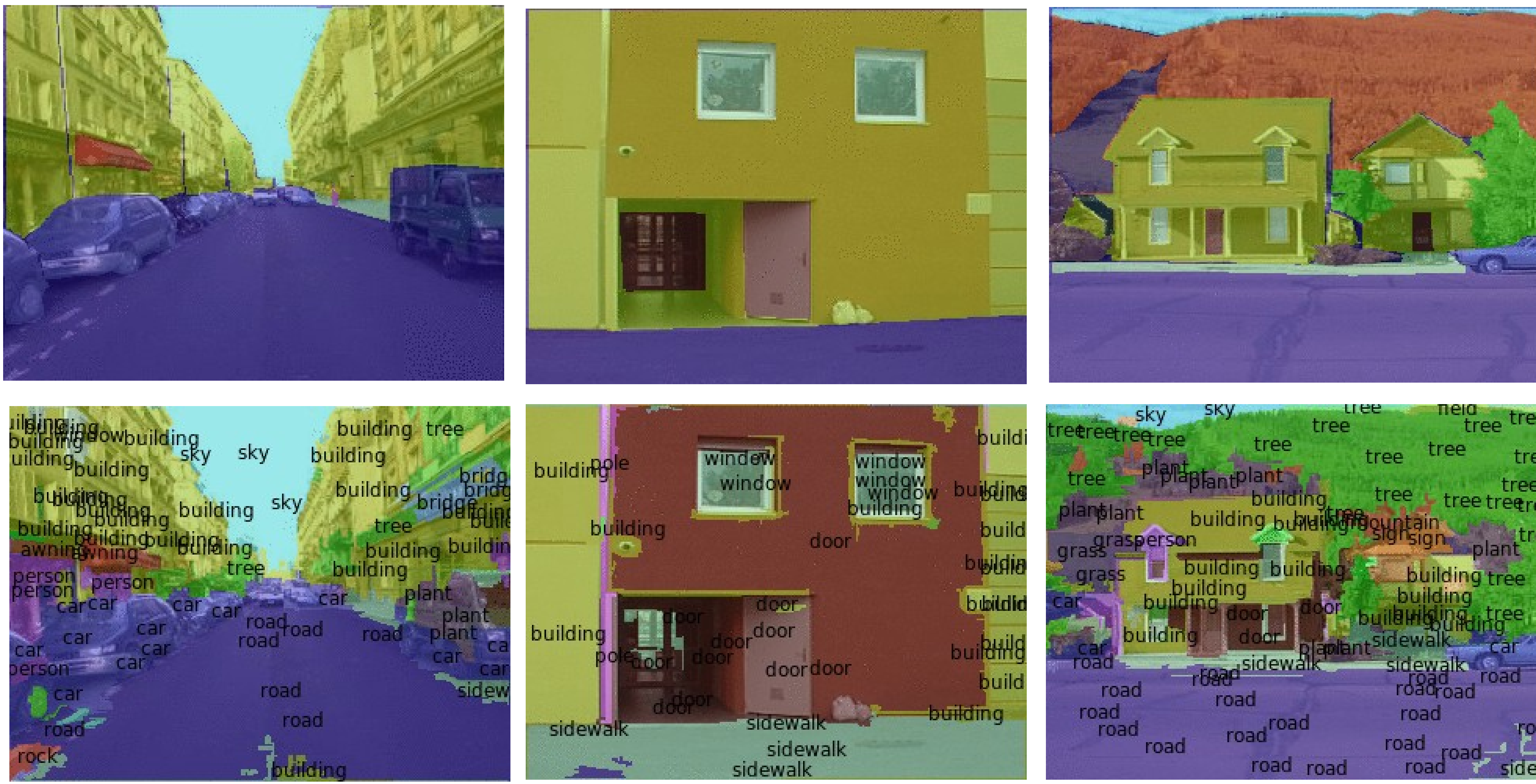




Sermanet et al. “Pedestrian detection with unsupervised multi-stage..” CVPR 2013

CONV NETS: EXAMPLES

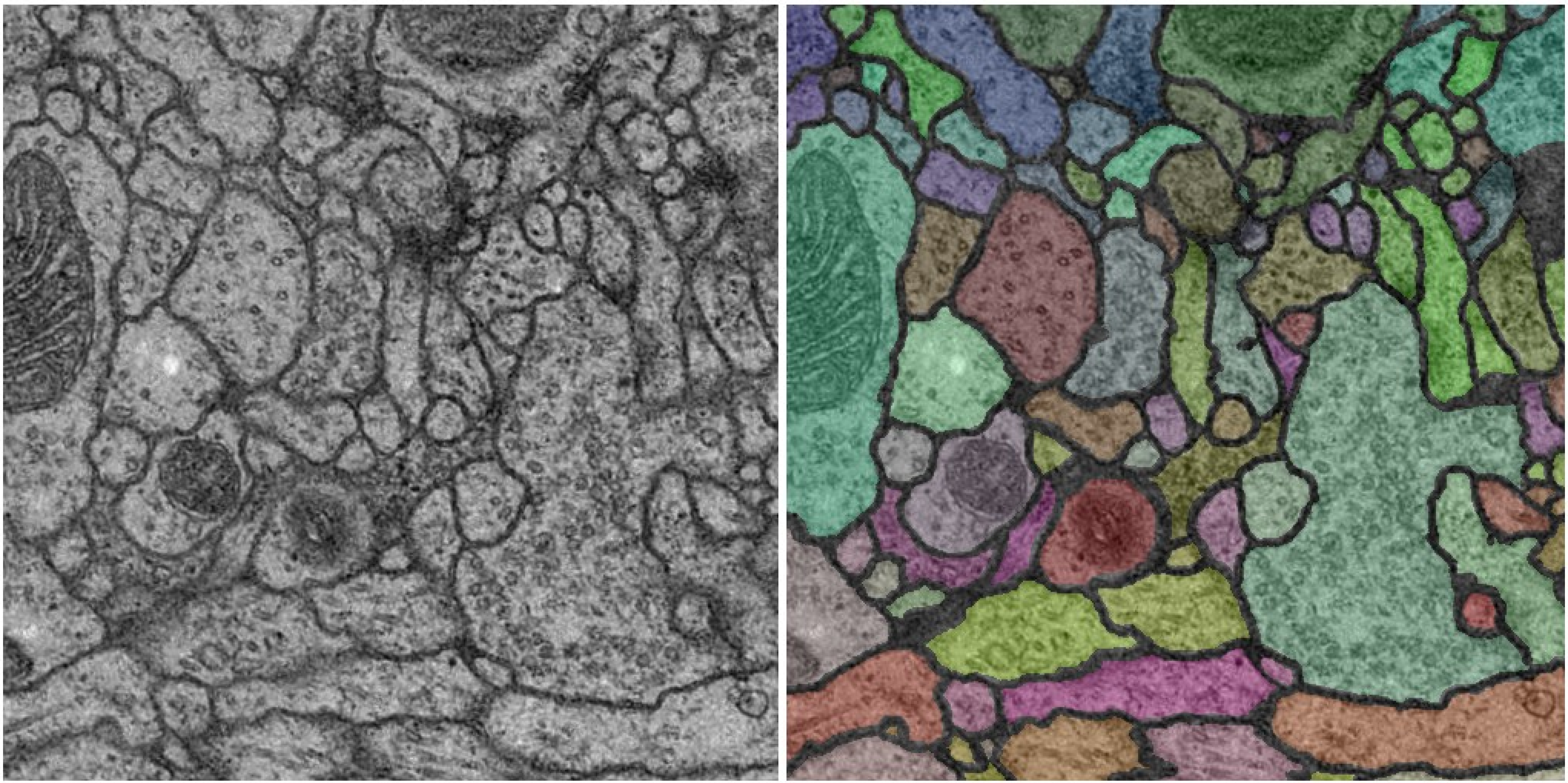
- **Scene Parsing**



Farabet et al. “Learning hierarchical features for scene labeling” PAMI 2013

CONV NETS: EXAMPLES

- **Segmentation 3D volumetric images**



Ciresan et al. “DNN segment neuronal membranes...” NIPS 2012
Turaga et al. “Maximin learning of image segmentation” NIPS 2009

CONV NETS: EXAMPLES

- **Action recognition from videos**

Taylor et al. “Convolutional learning of spatio-temporal features” ECCV 2010

CONV NETS: EXAMPLES

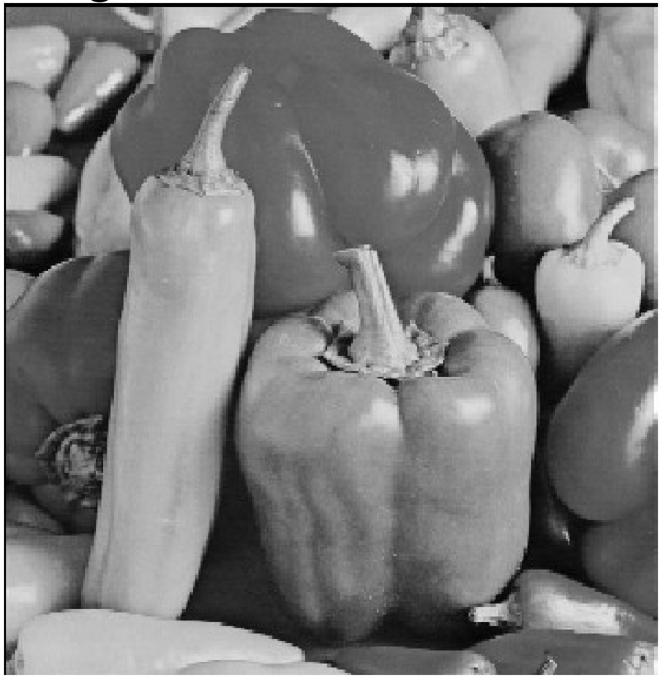
- **Robotics**

Sermanet et al. “Mapping and planning ...with long range perception” IROS 2008

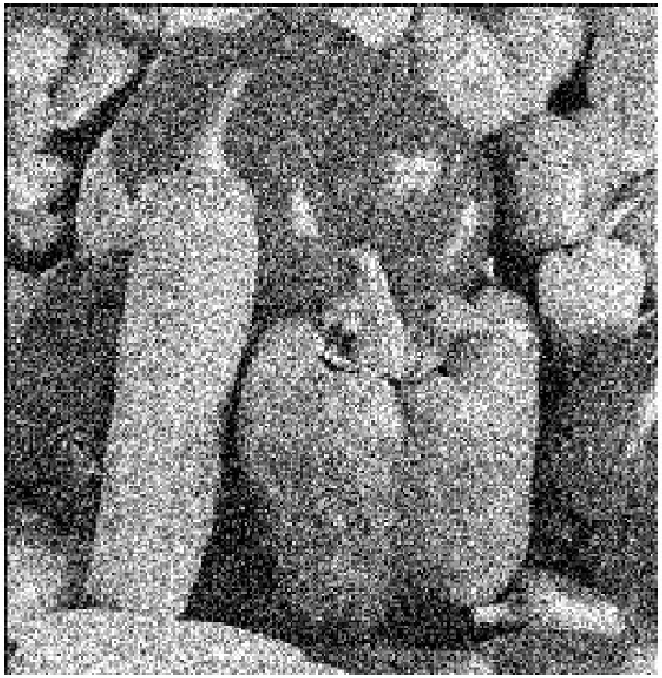
CONV NETS: EXAMPLES

- **Denoising**

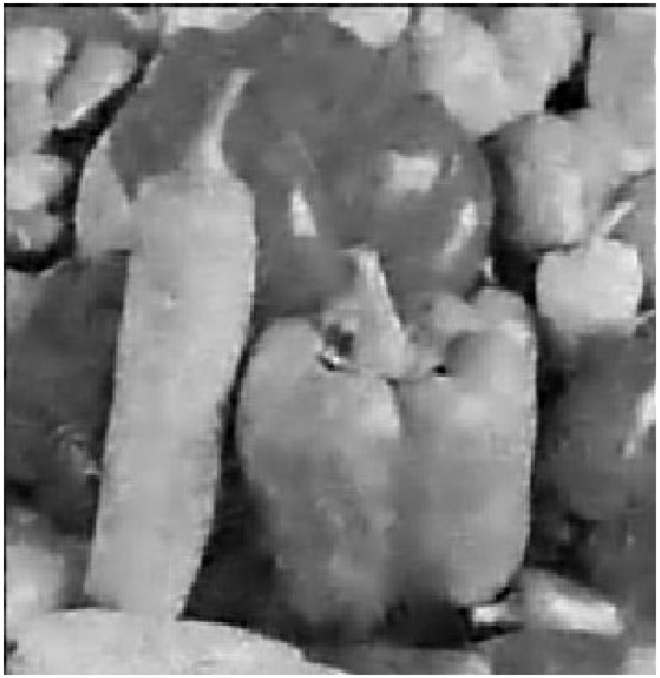
original



noised



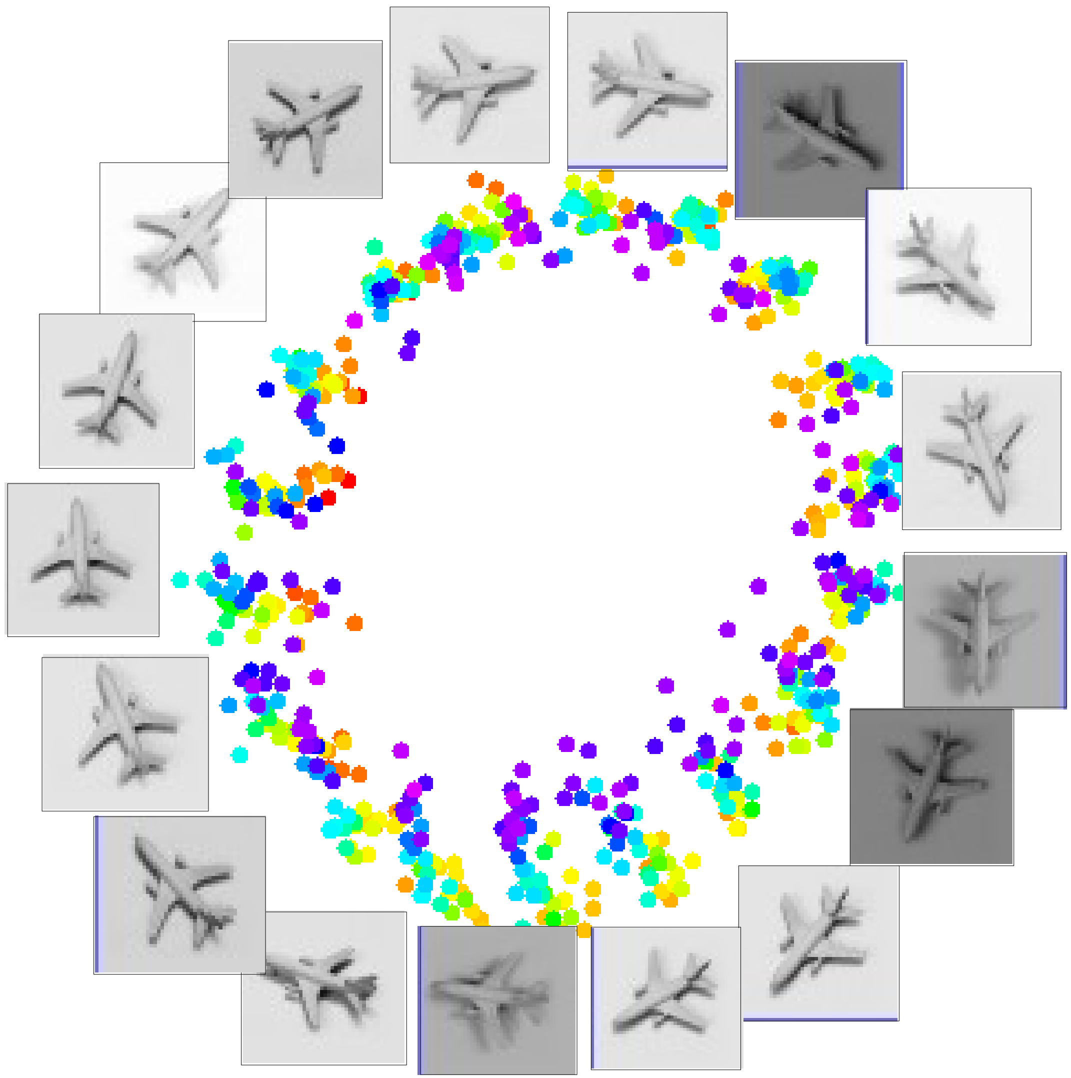
denoised



Burger et al. “Can plain NNs compete with BM3D?” CVPR 2012

CONV NETS: EXAMPLES

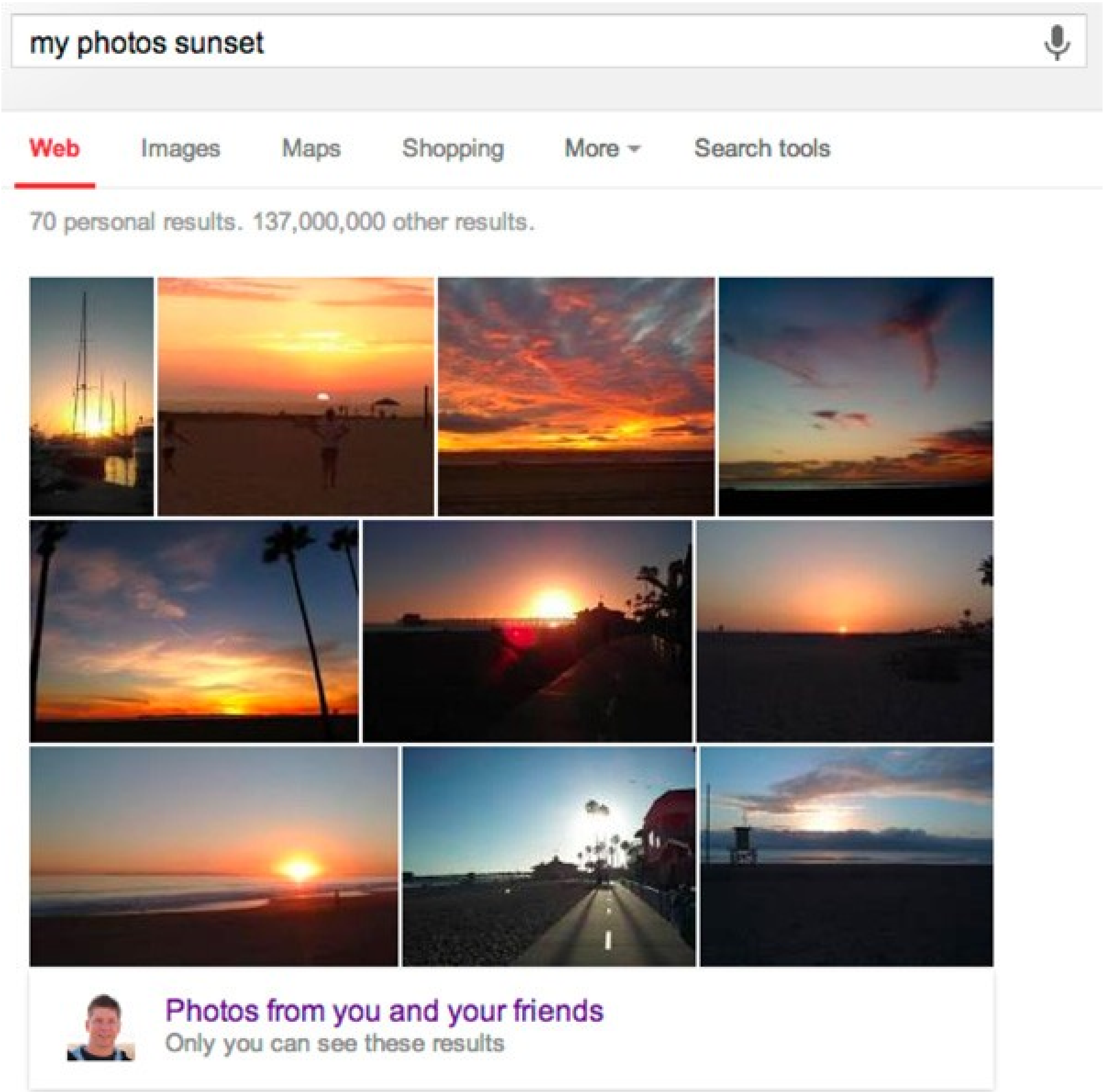
- **Dimensionality reduction / learning embeddings**



Hadsell et al. “Dimensionality reduction by learning an invariant mapping” CVPR 2006

CONV NETS: EXAMPLES

- **Deployed in commercial systems (Google & Baidu, spring 2013)**



CONV NETS: EXAMPLES

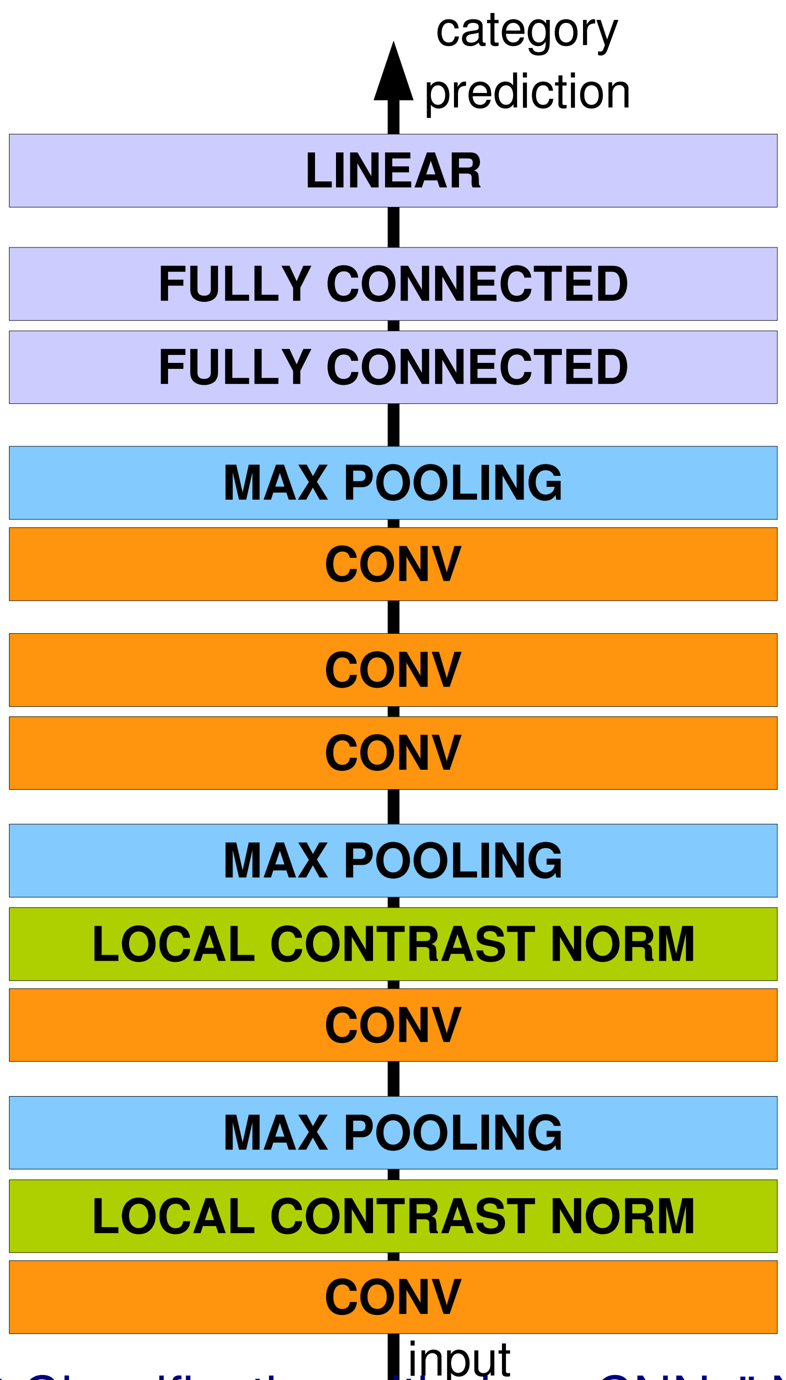
- **Image classification**

IMAGENET

→ **Object Recognizer** → railcar

Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012

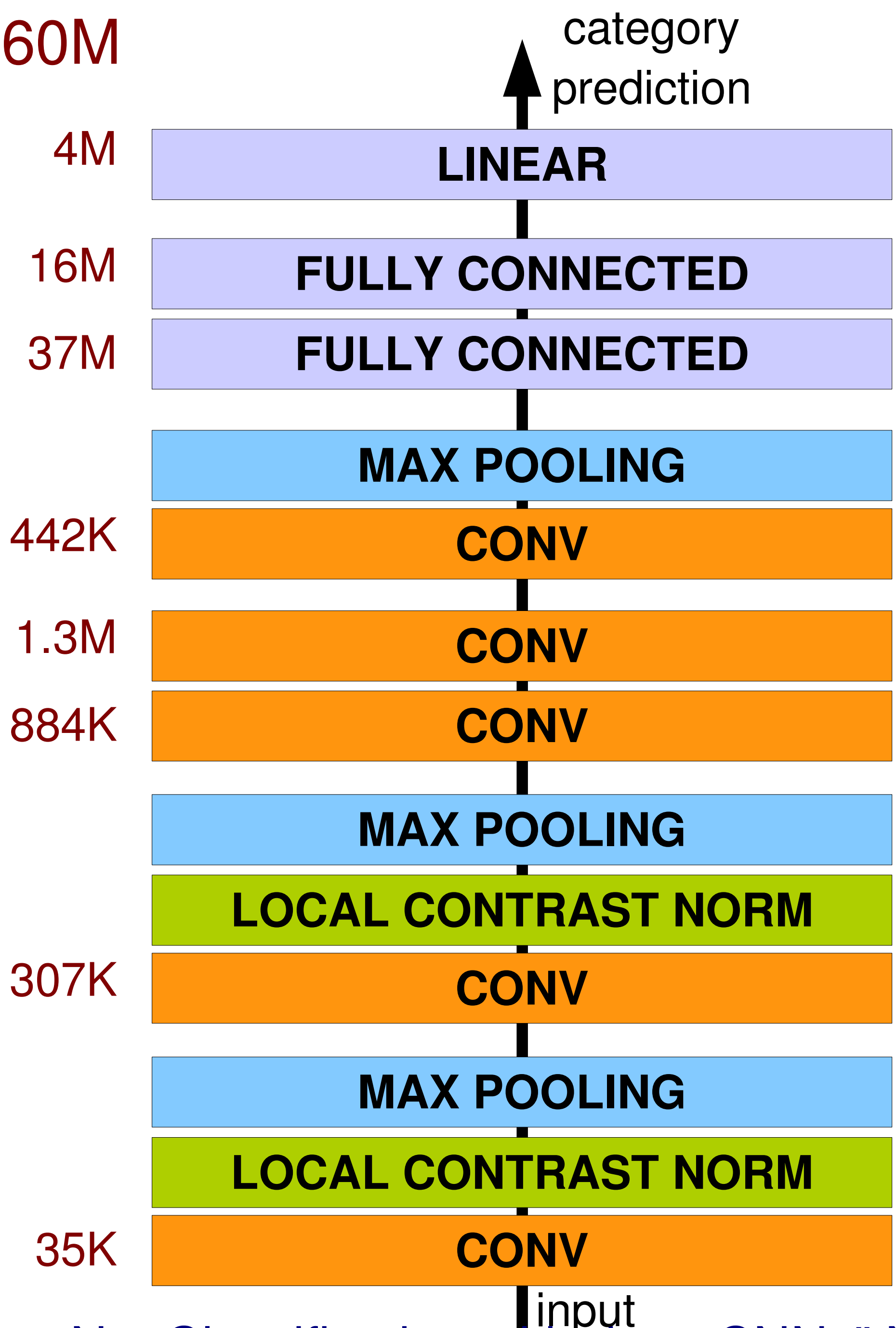
Architecture



Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

Architecture

Total nr. params: 60M

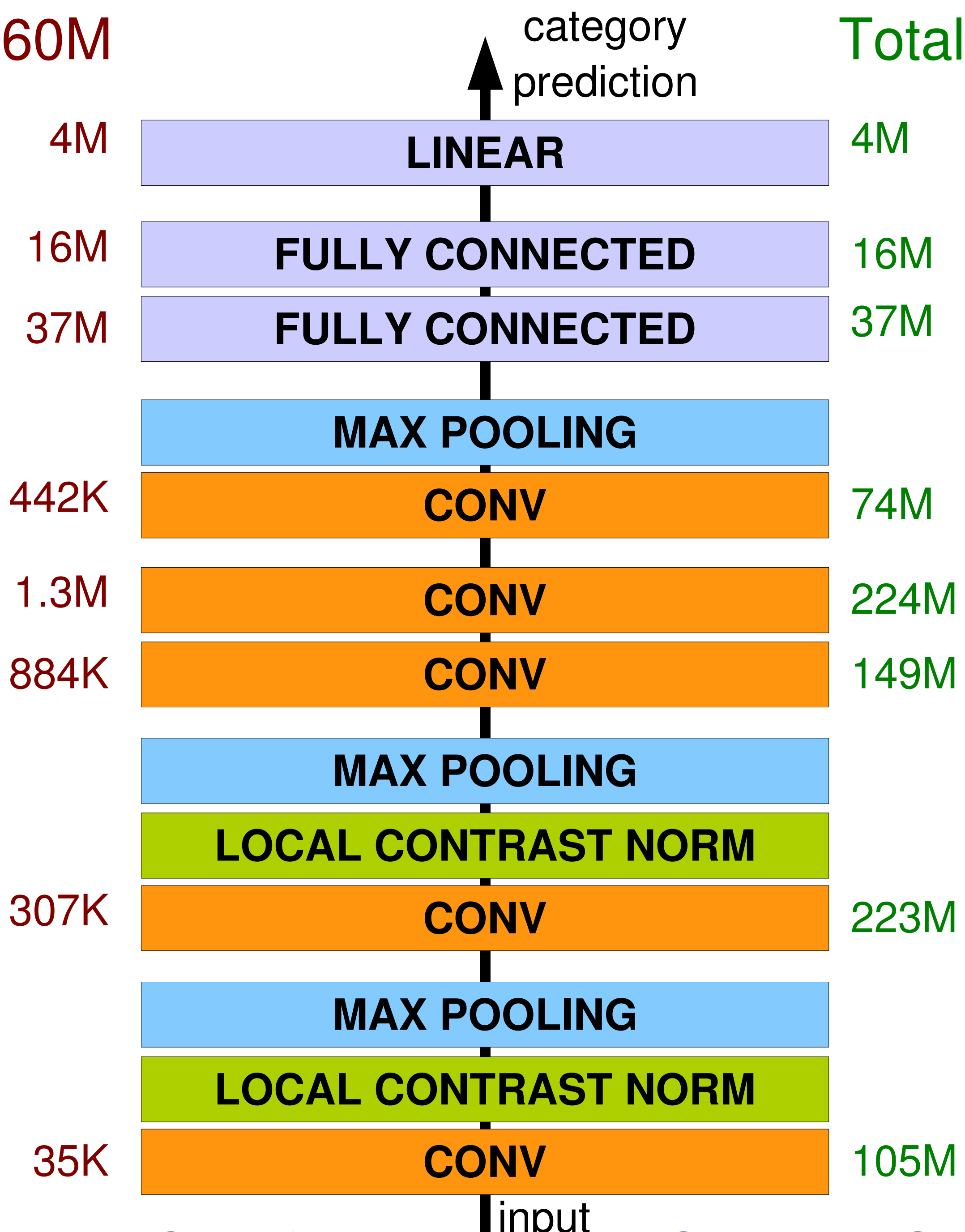


Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012

Architecture

Total nr. params: 60M

Total nr. flops: 832M



Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012

Optimization

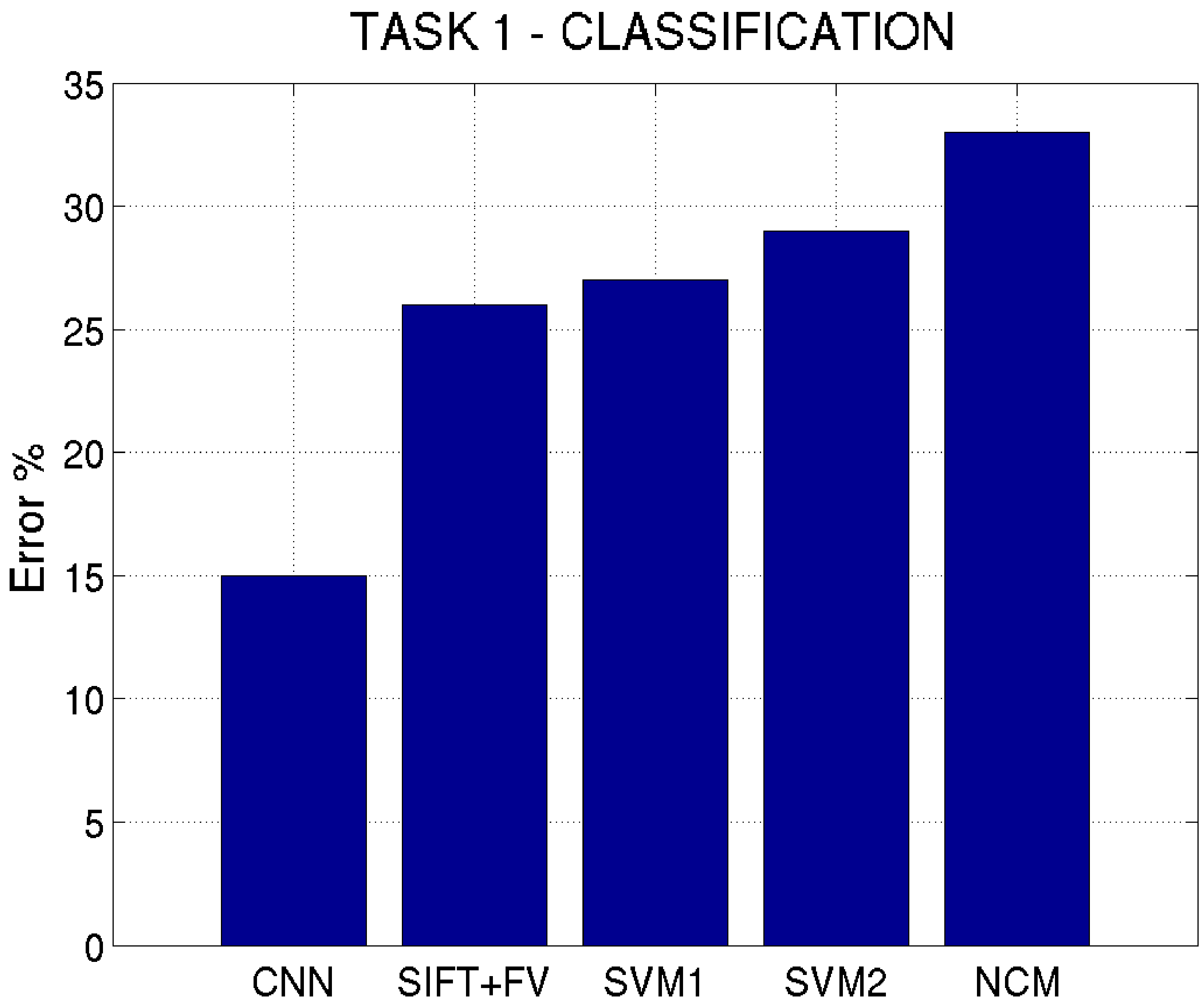
SGD with momentum:

- Learning rate = 0.01
- Momentum = 0.9

Improving generalization by:

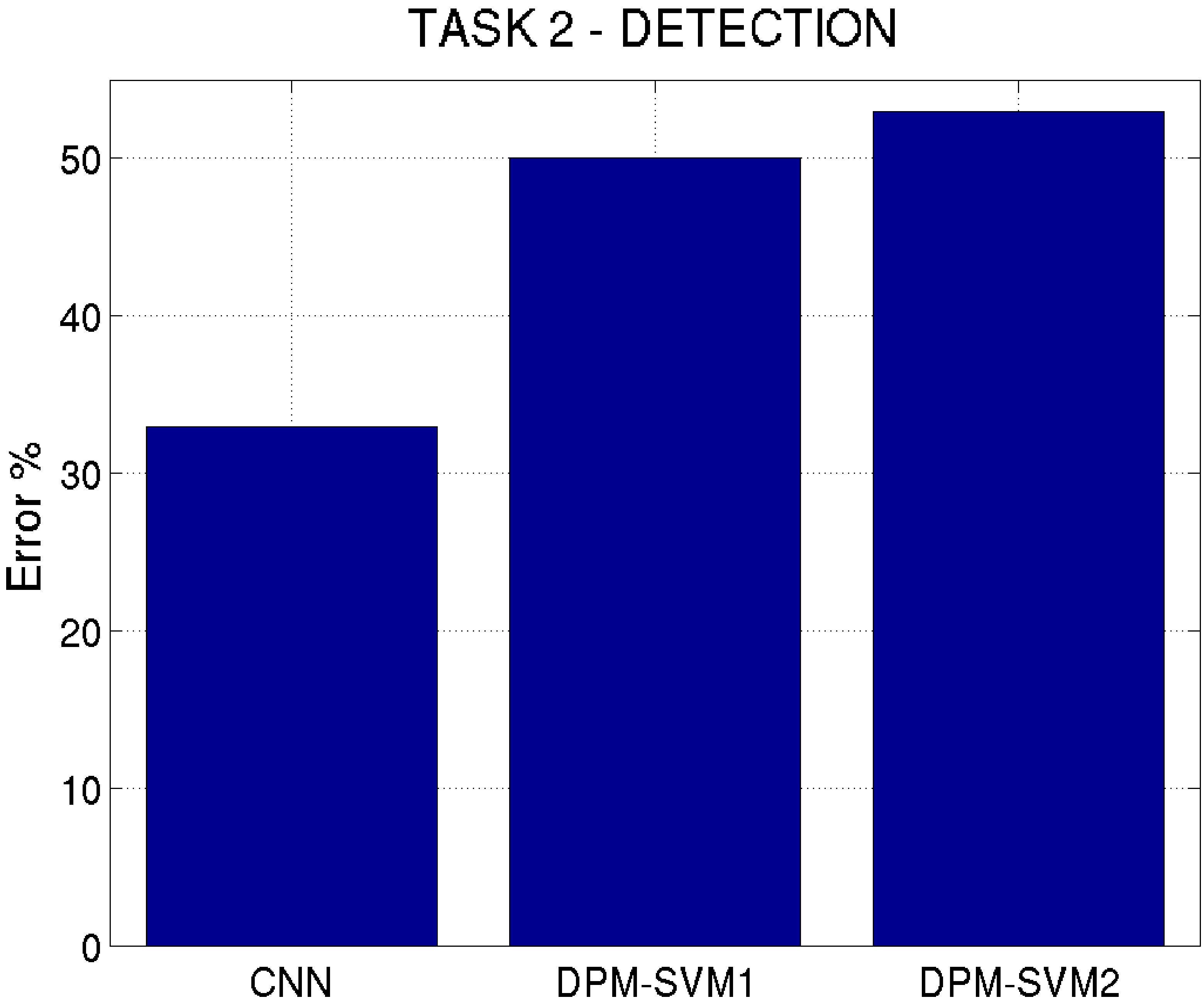
- Weight sharing (convolution)
- Input distortions
- Dropout = 0.5
- Weight decay = 0.0005

Results: ILSVRC 2012



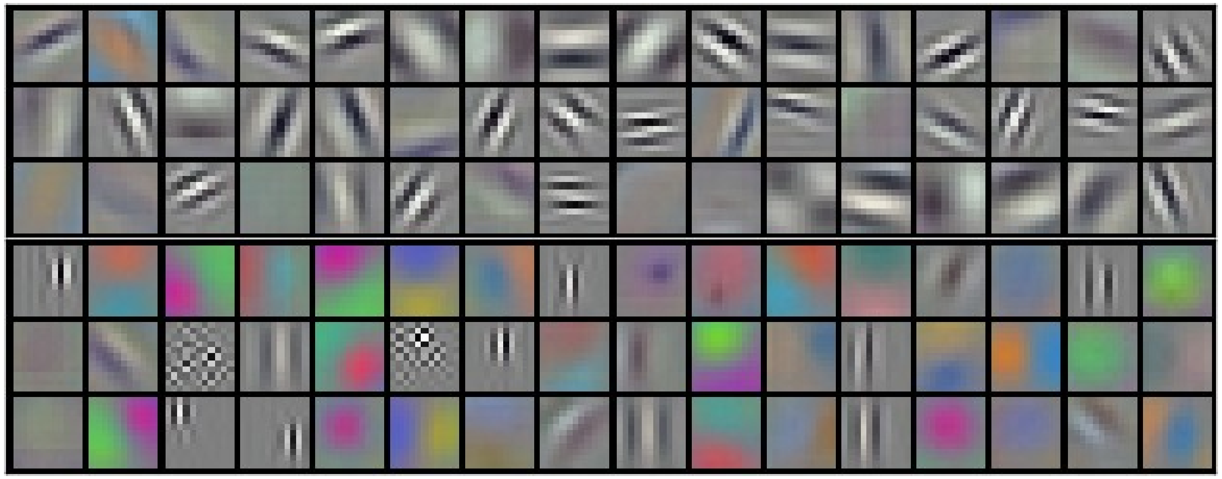
Ranzato

Results: ILSVRC 2012



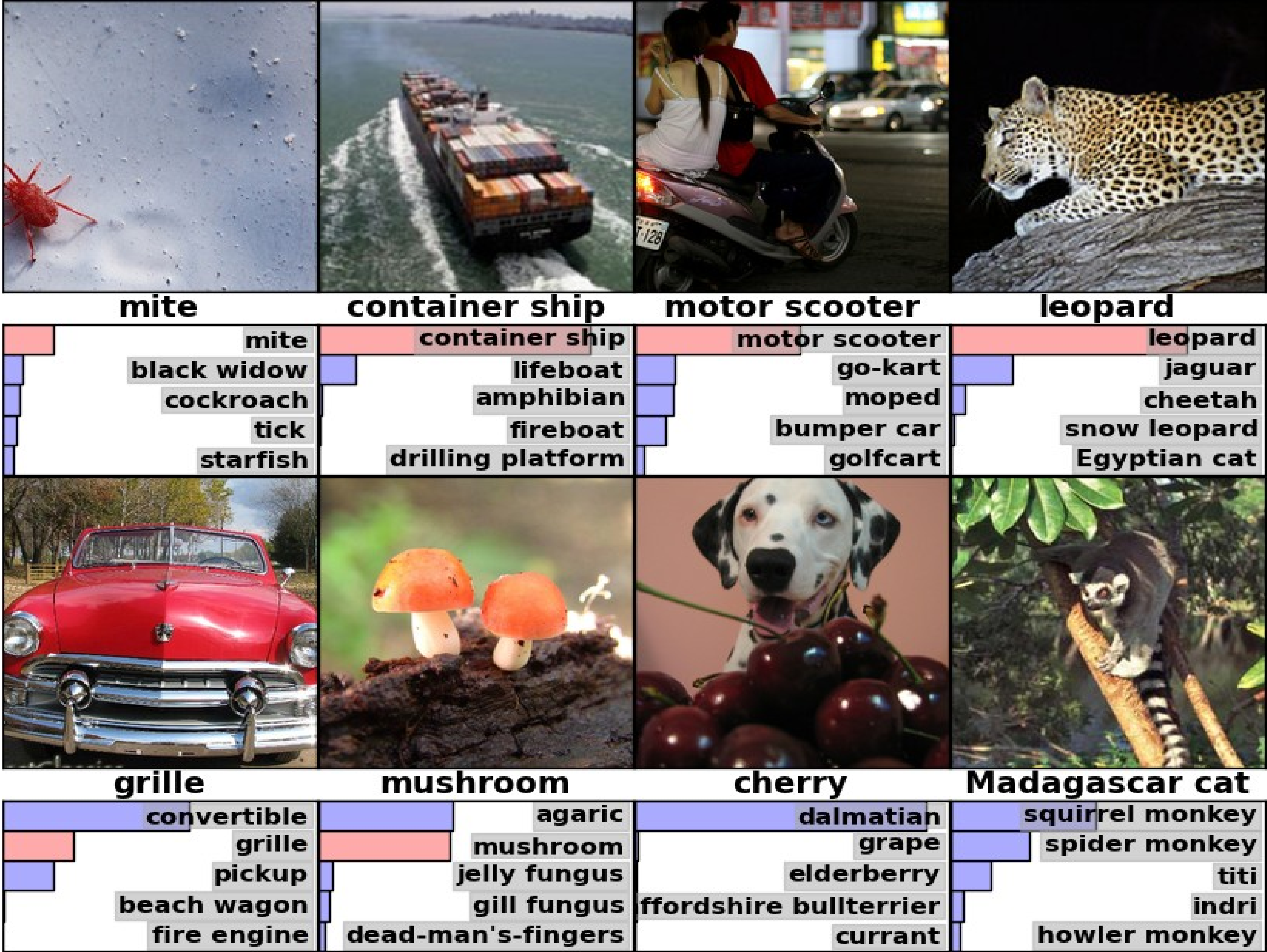
Ranzato

Results

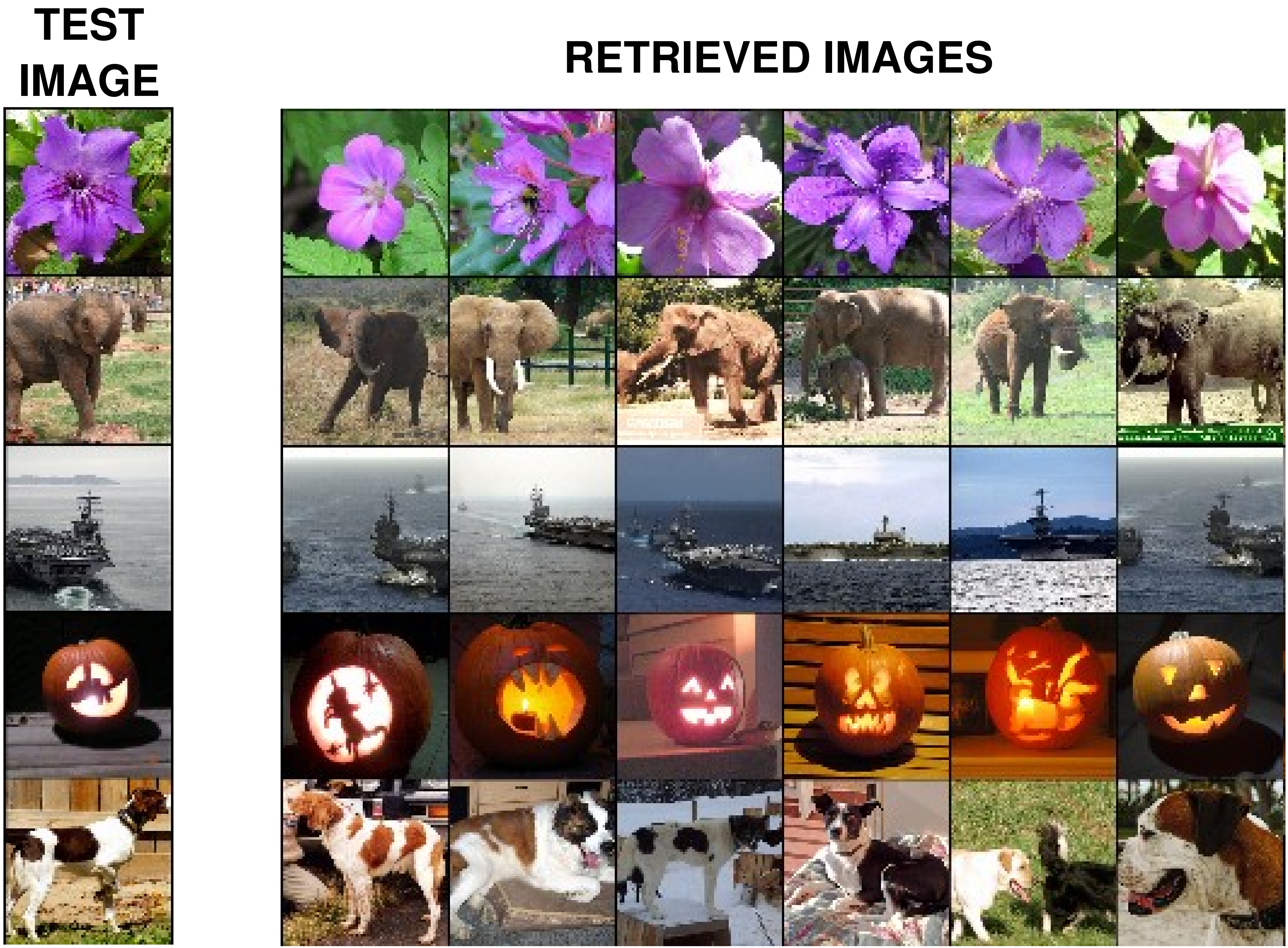


First layer learned filters (processing raw pixel values).

Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012



mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey


TEST IMAGE
RETRIEVED IMAGES

Outline

- Motivation
- Deep Learning: The Big Picture
- From neural nets to convolutional nets
- Applications
- A practical guide

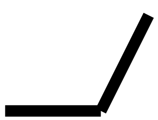
CHOOSING THE ARCHITECTURE

- [Convolution $\rightarrow$ LCN $\rightarrow$ pooling]* + fully connected layer
- Cross-validation
- Task dependent
- The more data: the more layers and the more kernels
  - Look at the number of parameters at each layer
  - Look at the number of flops at each layer
- Computational cost
- Be creative :)

HOW TO OPTIMIZE

- SGD (with momentum) usually works very well

- Pick learning rate by running on a subset of the data
  Bottou "Stochastic Gradient Tricks" Neural Networks 2012
  - Start with large learning rate and divide by 2 until loss does not diverge
  - Decay learning rate by a factor of ~100 or more by the end of training

- Use  non-linearity

- Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.

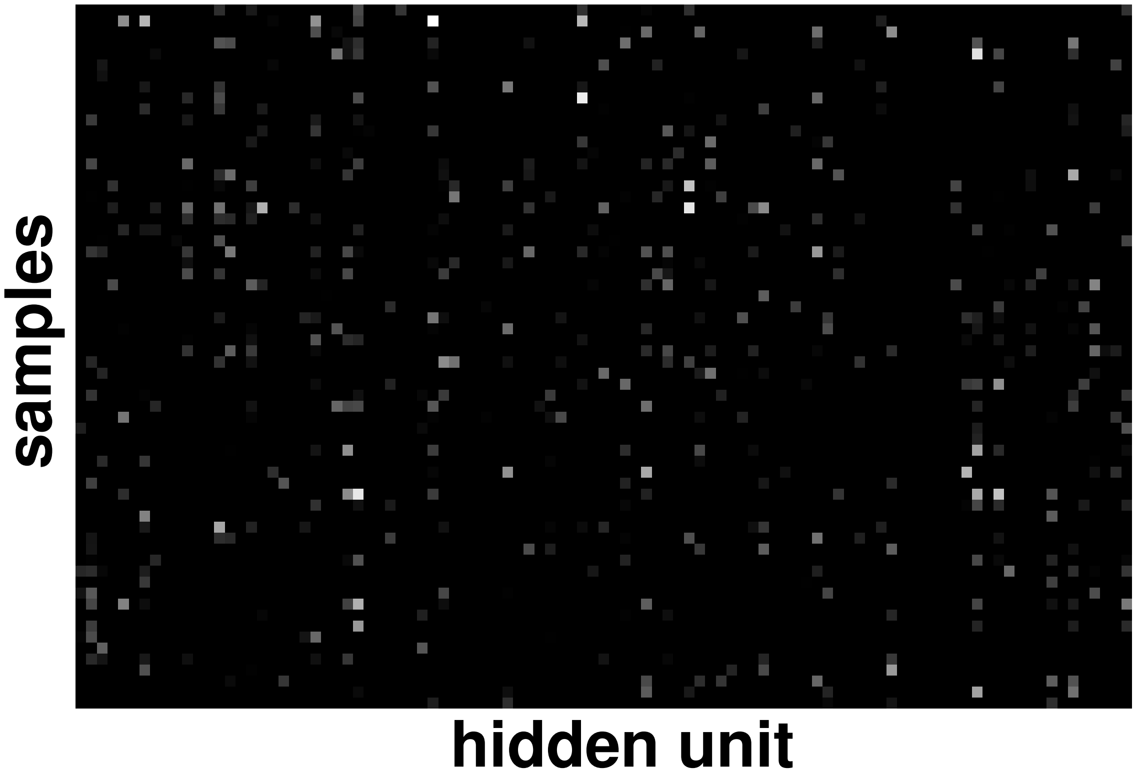
HOW TO IMPROVE GENERALIZATION

- Weight sharing (greatly reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, etc.)
- Dropout

  Hinton et al. "Improving Nns by preventing co-adaptation of feature detectors" arxiv 2012
- Weight decay (L2, L1)
- Sparsity in the hidden units
- Multi-task (unsupervised learning)

OTHER THINGS GOOD TO KNOW

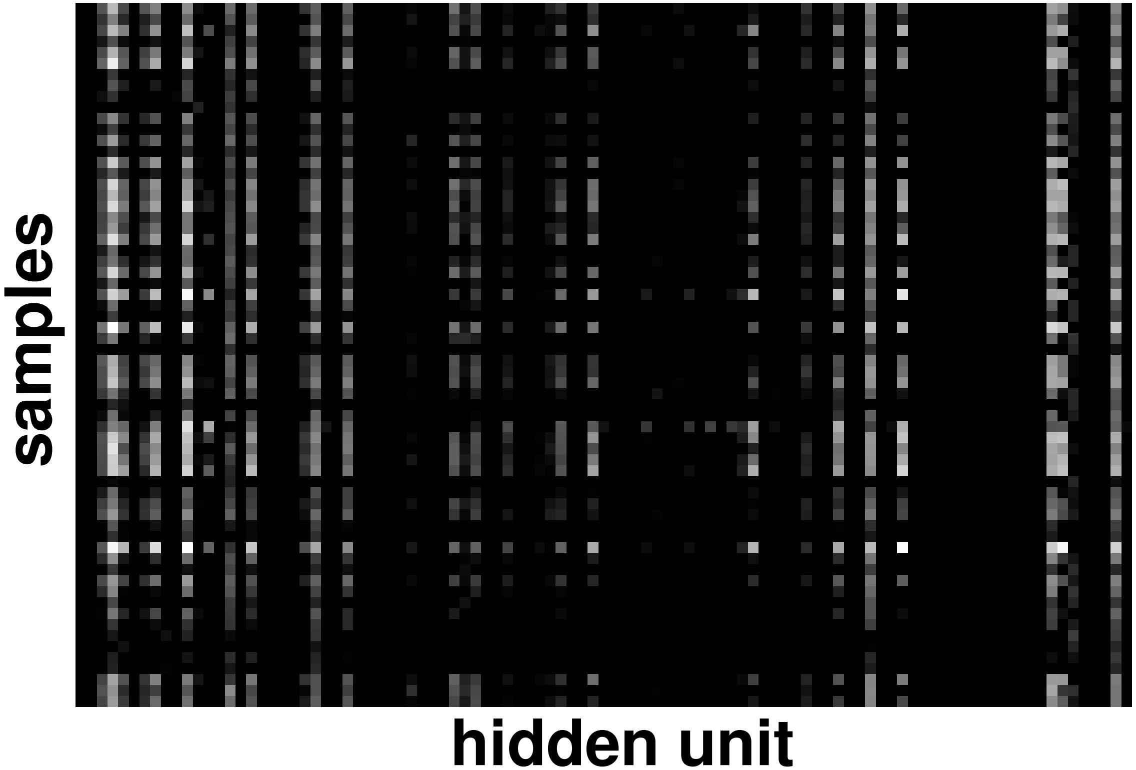
- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.



Good training: hidden units are sparse across samples and across features.

OTHER THINGS GOOD TO KNOW

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.

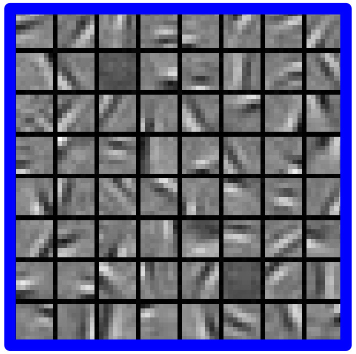


Bad training: many hidden units ignore the input and/or exhibit strong correlations.

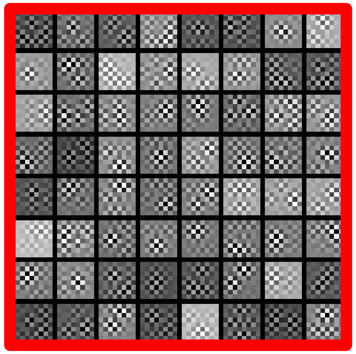
OTHER THINGS GOOD TO KNOW

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.
- Visualize parameters

GOOD

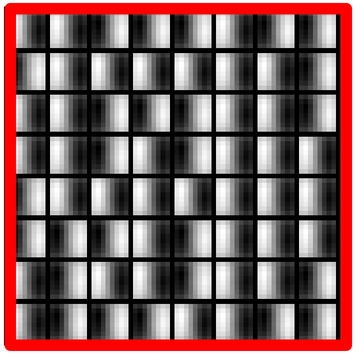


BAD



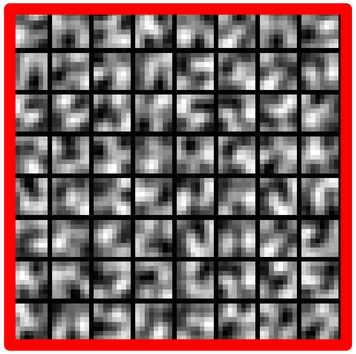
too noisy

BAD



too correlated

BAD



lack structure

Good training: learned filters exhibit structure and are uncorrelated.

OTHER THINGS GOOD TO KNOW

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.
- Visualize parameters
- Measure error on both training and validation set.
- Test on a small subset of the data and check the error $\rightarrow 0$.

WHAT IF IT DOES NOT WORK?

- Training diverges:
  - Learning rate may be too large $\rightarrow$ decrease learning rate
  - BPROP is buggy $\rightarrow$ numerical gradient checking
- Parameters collapse / loss is minimized but accuracy is low
  - Check loss function:
    - Is it appropriate for the task you want to solve?
    - Does it have degenerate solutions?
- Network is underperforming
  - Compute flops and nr. params. $\rightarrow$ if too small, make net larger
  - Visualize hidden units/params $\rightarrow$ fix optmization
- Network is too slow
  - Compute flops and nr. params. $\rightarrow$ GPU,distrib. framework, make net smaller

FUTURE CHALLENGES

- Scalability
  - Hardware
    - GPU / distributed frameworks
  - Algorithms
    - Better losses
    - Better optimizers
- Learning better representations
  - Video
  - Unsupervised learning
  - Multi-task learning
- Feedback at training and inference time
- Structure prediction
- Black-box tool (hyper-parameters optimization)

Snoek et al. “Practical Bayesian optimization of ML algorithms” NIPS 2012

SUMMARY

- Want to efficiently learn non-linear adaptive hierarchical systems
- End-to-end learning
- Gradient-based learning
- Adapting neural nets to vision:
  - Weight sharing
  - Pooling and Contrast Normalization
- Improving generalization on small datasets:
  - Weight decay, dropout, sparsity, multi-task
- Training a convnet means:
  - Design architecture
  - Design loss function
  - Optimization (SGD)
- Very successful (large-scale) applications

SOFTWARE

Torch7: learning library that supports neural net training

http://www.torch.ch

http://code.cogbits.com/wiki/doku.php (tutorial with demos by C. Farabet)

Python-based learning library (U. Montreal)

- http://deeplearning.net/software/theano/ (does automatic differentiation)

C++ code for ConvNets (Sermanet)

– http://eblearn.sourceforge.net/

Efficient CUDA kernels for ConvNets (Krizhevsky)

– code.google.com/p/cuda-convnet

REFERENCES

Convolutional Nets

– LeCun, Bottou, Bengio and Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998

- Krizhevsky, Sutskever, Hinton “ImageNet Classification with deep convolutional neural networks” NIPS 2012

– Jarrett, Kavukcuoglu, Ranzato, LeCun: What is the Best Multi-Stage Architecture for Object Recognition?, Proc. International Conference on Computer Vision (ICCV'09), IEEE, 2009

- Kavukcuoglu, Sermanet, Boureau, Gregor, Mathieu, LeCun: Learning Convolutional Feature Hierachies for Visual Recognition, Advances in Neural Information Processing Systems (NIPS 2010), 23, 2010

– see yann.lecun.com/exdb/publis for references on many different kinds of convnets.

– see http://www.cmap.polytechnique.fr/scattering/ for scattering networks (similar to convnets but with less learning and stronger mathematical foundations)

REFERENCES

Applications of Convolutional Nets

– Farabet, Couprie, Najman, LeCun. Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers", ICML 2012

– Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala and Yann LeCun: Pedestrian Detection with Unsupervised Multi-Stage Feature Learning, CVPR 2013

- D. Ciresan, A. Giusti, L. Gambardella, J. Schmidhuber. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. NIPS 2012

- Raia Hadsell, Pierre Sermanet, Marco Scoffier, Ayse Erkan, Koray Kavackuoglu, Urs Muller and Yann LeCun. Learning Long-Range Vision for Autonomous Off-Road Driving, Journal of Field Robotics, 26(2):120-144, 2009

– Burger, Schuler, Harmeling. Image Denoisng: Can Plain Neural Networks Compete with BM3D?, CVPR 2012

– Hadsell, Chopra, LeCun. Dimensionality reduction by learning an invariant mapping, CVPR 2006

– Bergstra et al. Making a science of model search: hyperparameter optimization in hundred of dimensions for vision architectures, ICML 2013

REFERENCES

Deep Learning in general

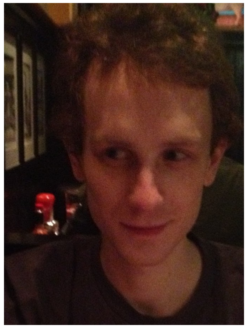
– deep learning tutorial slides at ICML 2013

– Yoshua Bengio, Learning Deep Architectures for AI, Foundations and Trends in Machine Learning, 2(1), pp.1-127, 2009.

– LeCun, Chopra, Hadsell, Ranzato, Huang: A Tutorial on Energy-Based Learning, in Bakir, G. and Hofman, T. and Schölkopf, B. and Smola, A. and Taskar, B. (Eds), Predicting Structured Data, MIT Press, 2006

ACKNOWLEDGEMENTS

Yann LeCun - NYU



Alex Krizhevsky - Google

Jeff Dean - Google

THANK YOU!