Deep Learning Tutorial

ICML, Atlanta, 2013-06-16

Yann LeCun

Center for Data Science & Courant Institute, NYU

yann@cs.nyu.edu

http://yann.lecun.com

Marc'Aurelio Ranzato

Google

ranzato@google.com

http://www.cs.toronto.edu/~ranzato

Deep Learning = Learning Representations/Features

- **The traditional model of pattern recognition (since the late 50's)**
  - Fixed/engineered features (or fixed kernel) + trainable classifier

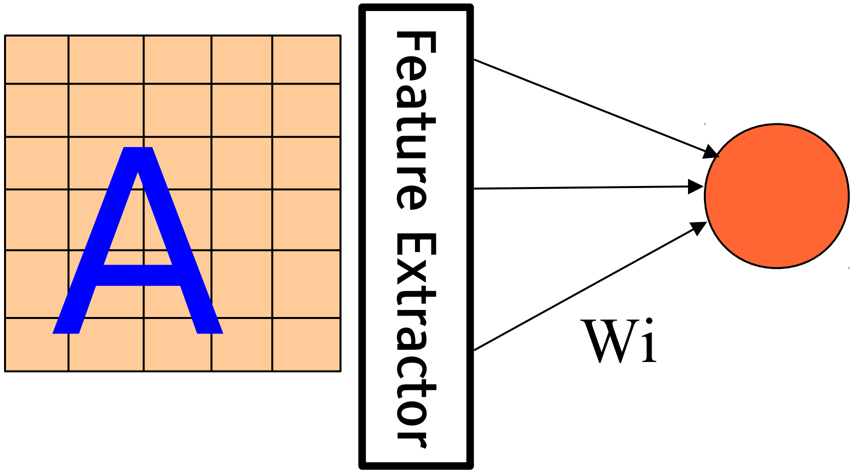
→ hand-crafted Feature Extractor → "Simple" Trainable Classifier →

- **End-to-end learning / Feature learning / Deep learning**
  - Trainable features (or kernel) + trainable classifier

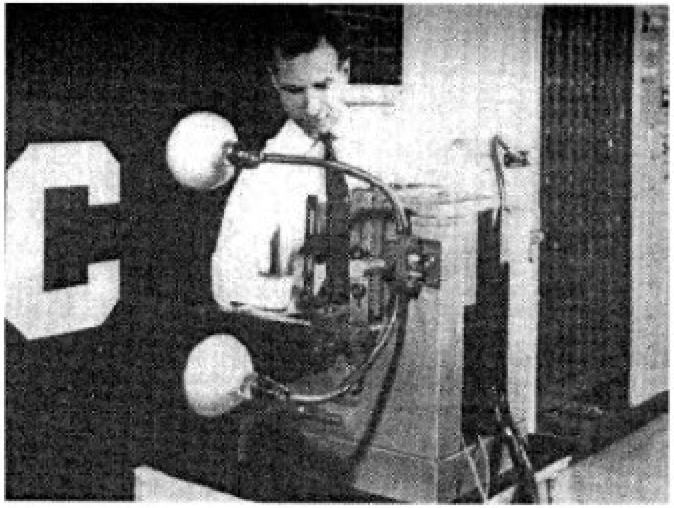
→ Trainable Feature Extractor → Trainable Classifier →

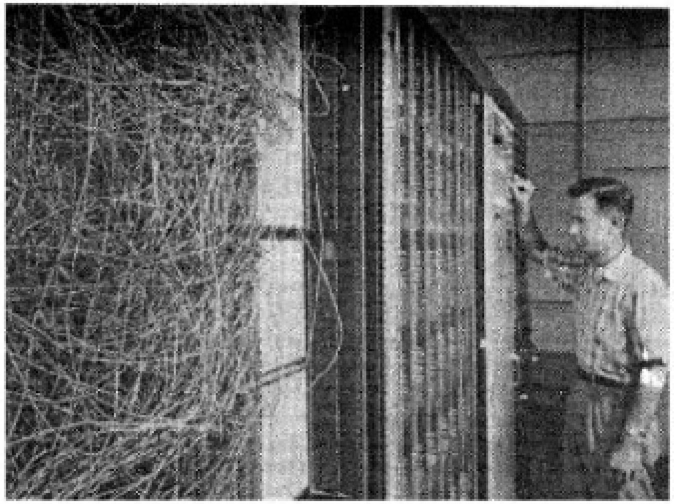
This Basic Model has not evolved much since the 50's

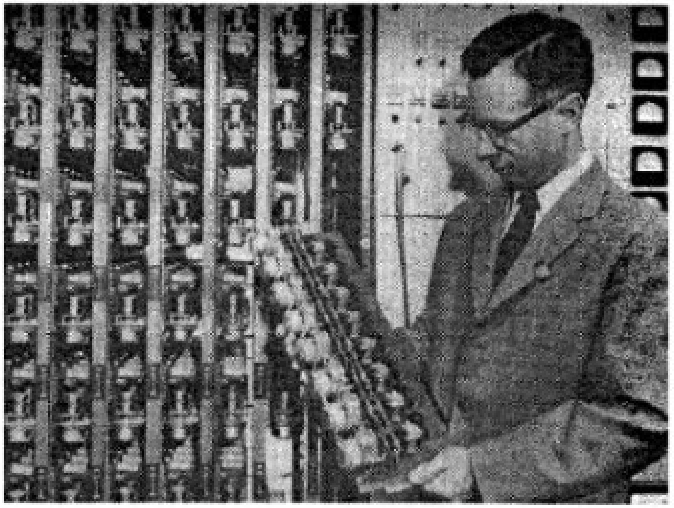
- The first learning machine: the Perceptron
  - Built at Cornell in 1960
- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor requires considerable efforts by experts.



$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$



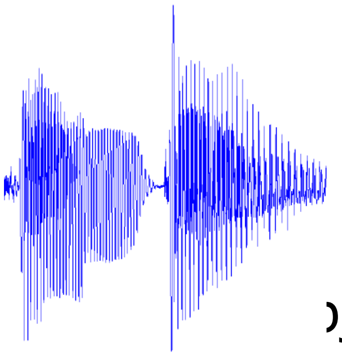




Architecture of "Mainstream" Pattern Recognition Systems

Y LeCun
MA Ranzato

- **Modern architecture for pattern recognition**
  - Speech recognition: early 90's – 2011



MFCC → Mix of Gaussians → Classifier →

fixed | unsupervised | supervised

bject Recognition: 2006 – 2012

SIFT HoG → K-means Sparse Coding → Pooling → Classifier →

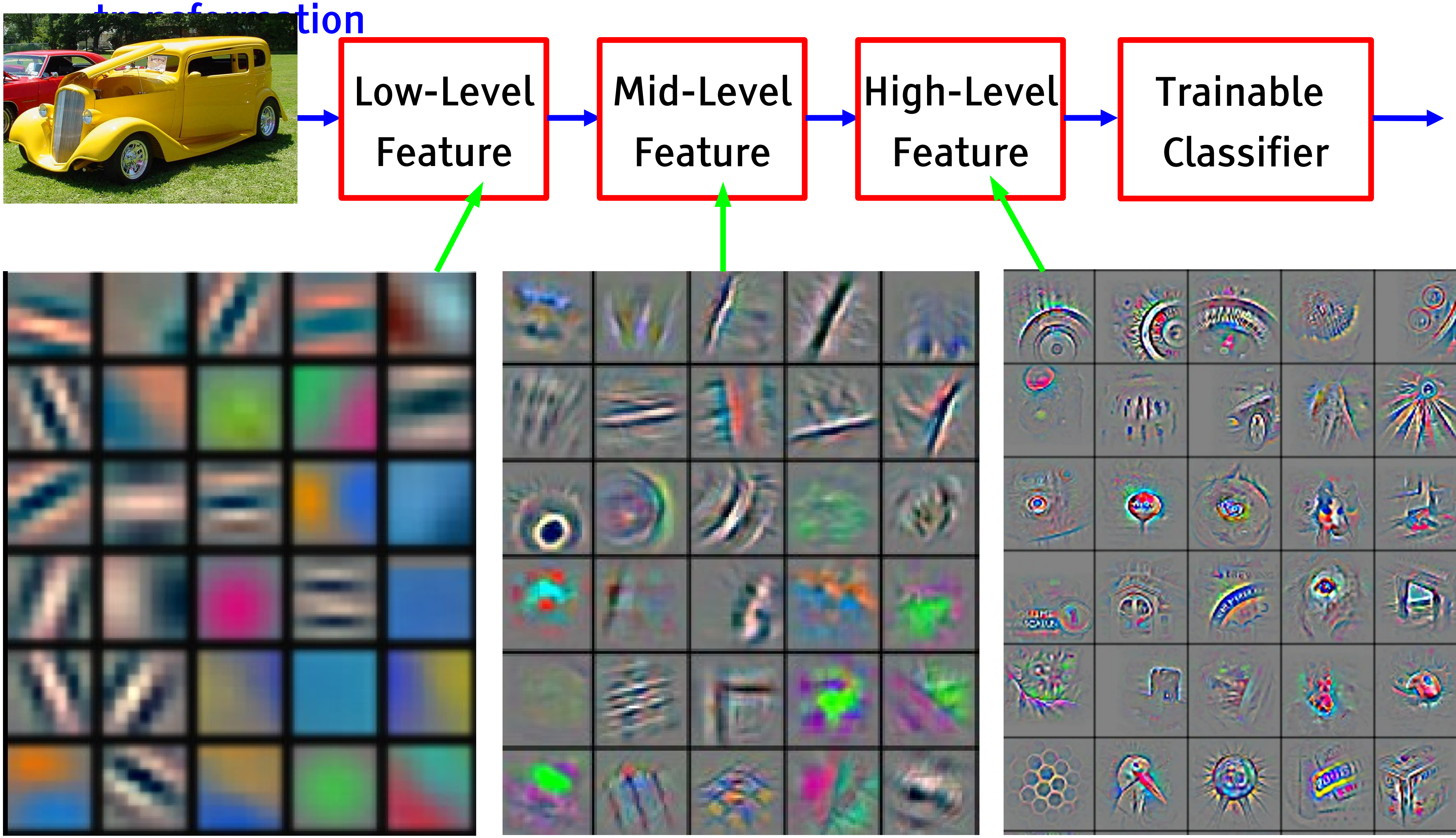
fixed | unsupervised | supervised

Low-level Features | Mid-level Features

Deep Learning = Learning Hierarchical Representations

Y LeCun
MA Ranzato

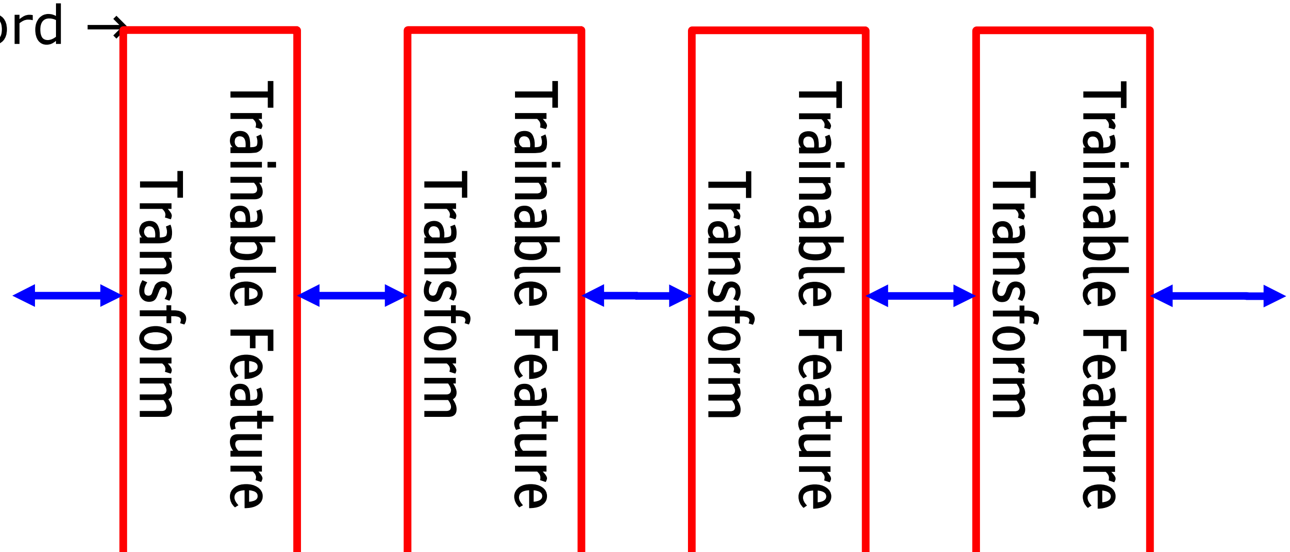
- It's deep if it has more than one stage of non-linear feature transformation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

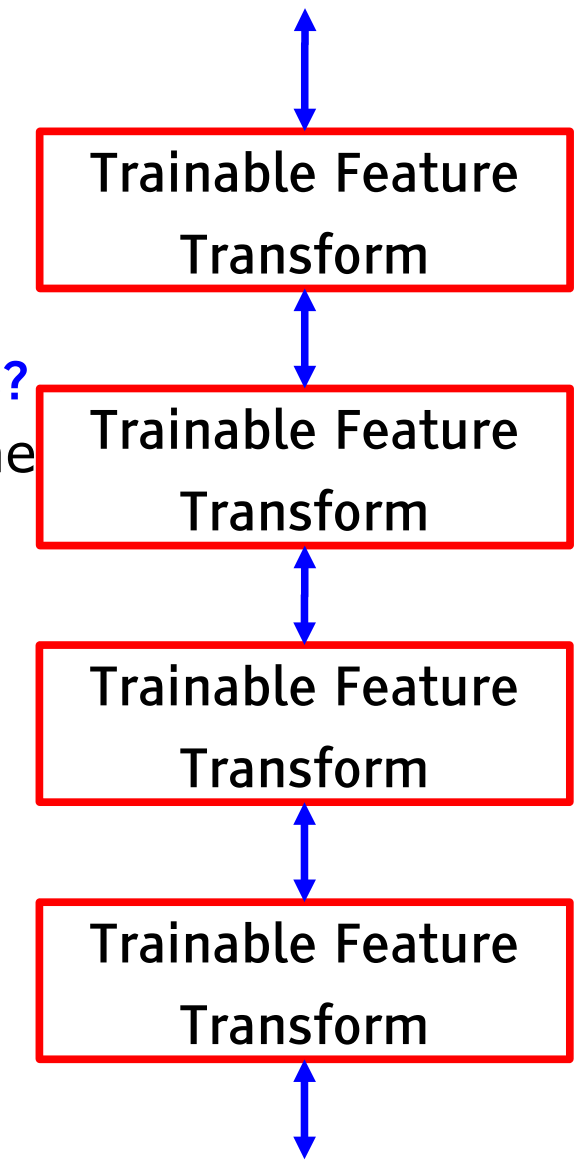
Trainable Feature Hierarchy

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - Sample → spectral band → sound → … → phone → phoneme → word →



Learning Representations: a challenge for ML, CV, AI, Neuroscience, Cognitive Science...

- How do we learn representations of the perceptual world?
  - How can a perceptual system build itself by looking at the world?
  - How much prior structure is necessary
- ML/AI: how do we learn features or feature hierarchies?
  - What is the fundamental principle? What is the learning algorithm? What is the architecture?
- Neuroscience: how does the cortex learn perception?
  - Does the cortex “run” a single, general learning algorithm? (or a small number of them)
- CogSci: how does the mind learn abstract concepts on top of less abstract ones?

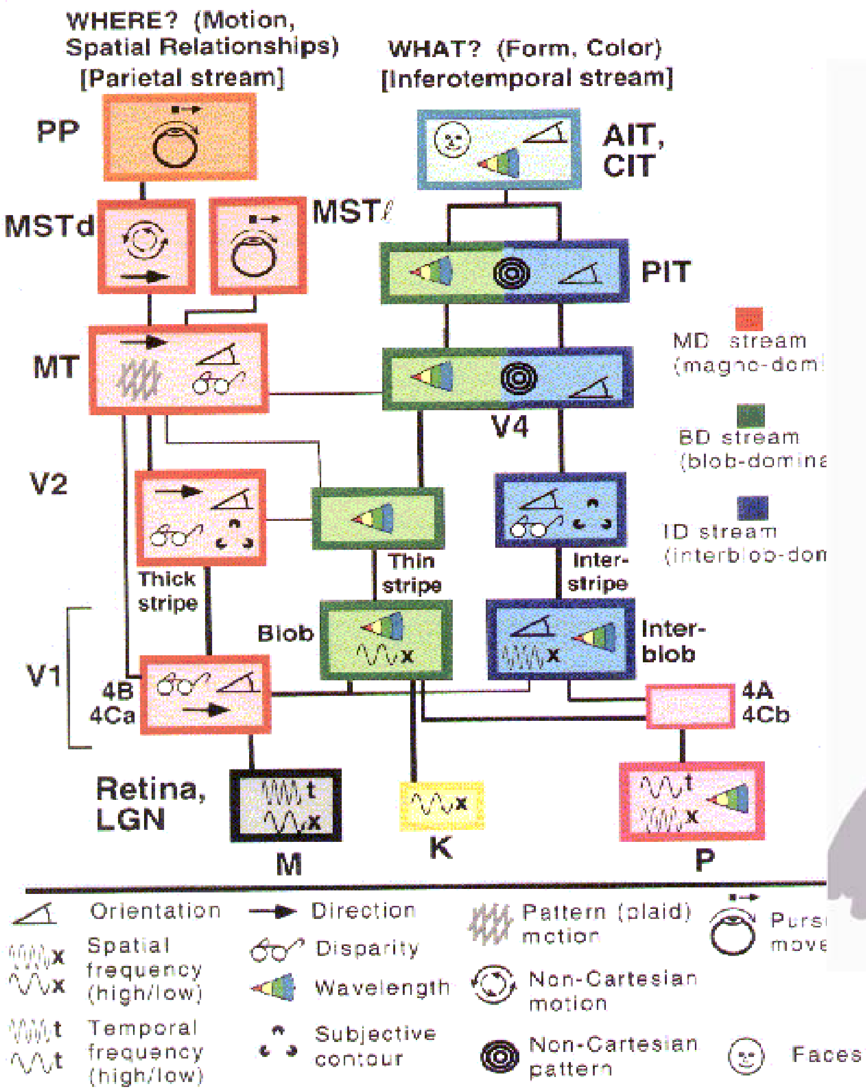


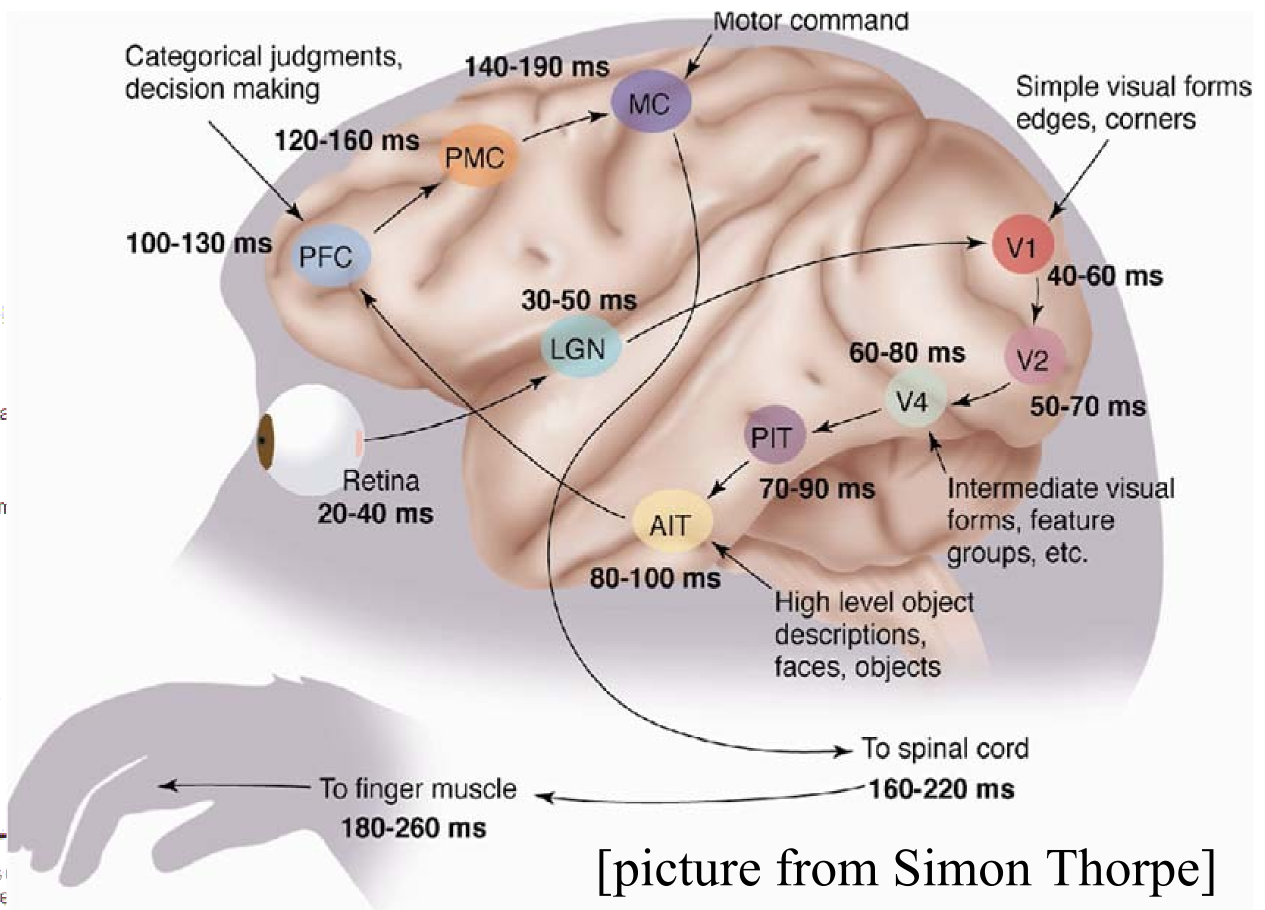
- Deep Learning addresses the problem of learning hierarchical representations with a single algorithm

The Mammalian Visual Cortex is Hierarchical

Y LeCun
MA Ranzato

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations





[picture from Simon Thorpe]

[Gallant & Van Essen]

Let's be inspired by nature, but not too much

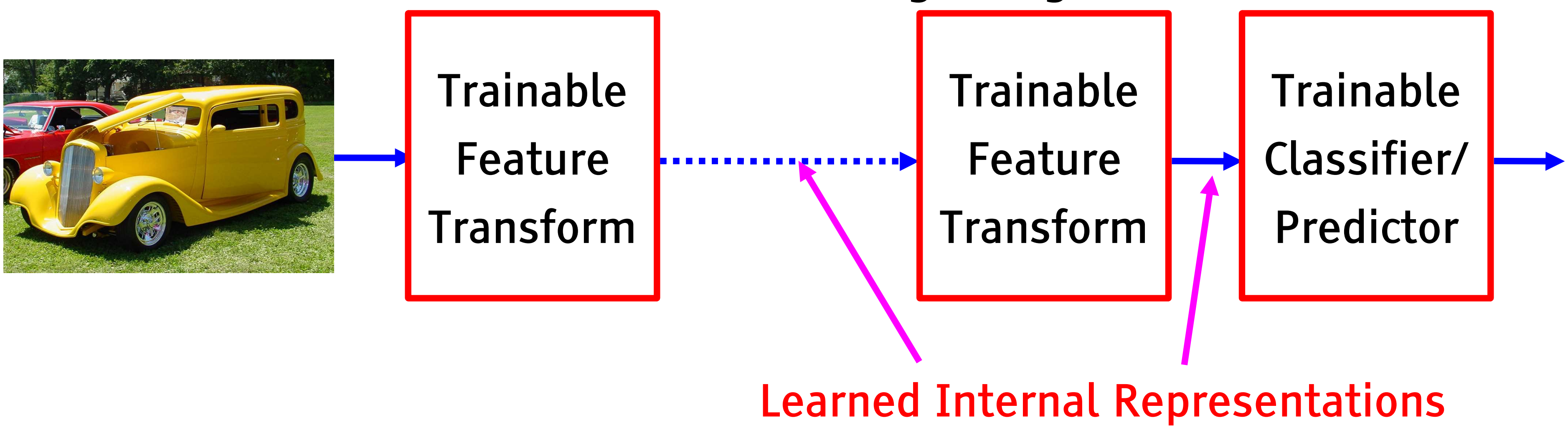
Y LeCun
MA Ranzato

- It's nice imitate Nature,
- But we also need to understand
  - How do we know which details are important?
  - Which details are merely the result of evolution, and the constraints of biochemistry?
- For airplanes, we developed aerodynamics and compressible fluid dynamics.
  - We figured that feathers and wing flapping weren't crucial
- QUESTION: What is the equivalent of aerodynamics for understanding intelligence?

L'Avion III de Clément Ader, 1897
(Musée du CNAM, Paris)
His Eole took off from the ground in 1890, 13 years before the Wright Brothers, but you probably never heard of it.

Trainable Feature Hierarchies: End-to-end learning

- **A hierarchy of trainable feature transforms**
  - Each module transforms its input representation into a higher-level one.
  - High-level features are more global and more invariant
  - Low-level features are shared among categories

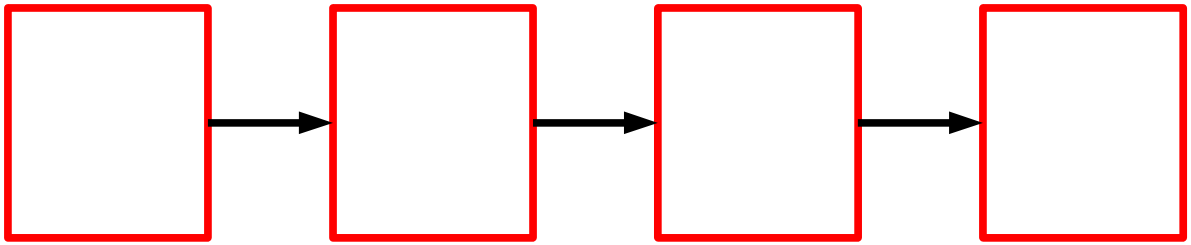


- **How can we make all the modules trainable and get them to learn appropriate representations?**

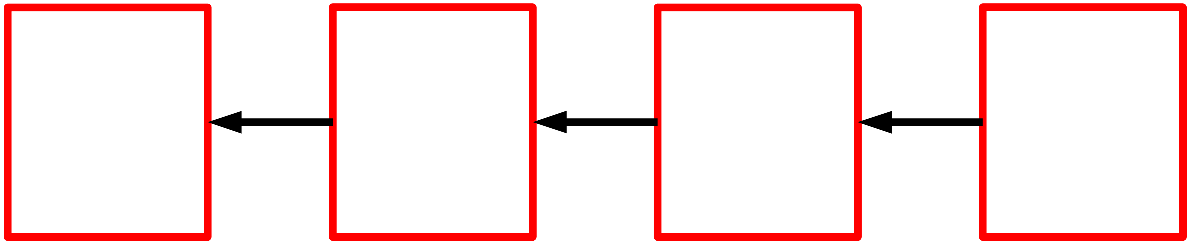
Three Types of Deep Architectures

Y LeCun
MA Ranzato

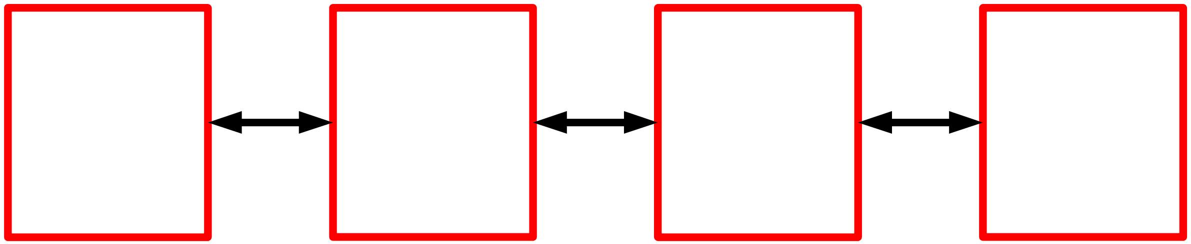
- Feed-Forward: multilayer neural nets, convolutional nets



- Feed-Back: Stacked Sparse Coding, Deconvolutional Nets



- Bi-Drectional: Deep Boltzmann Machines, Stacked Auto-Encoders



Three Types of Training Protocols

Y LeCun
MA Ranzato

- **Purely Supervised**
  - Initialize parameters randomly
  - Train in supervised mode
    - typically with SGD, using backprop to compute gradients
  - Used in most practical systems for speech and image recognition
- **Unsupervised, layerwise + supervised classifier on top**
  - Train each layer unsupervised, one after the other
  - Train a supervised classifier on top, keeping the other layers fixed
  - Good when very few labeled samples are available
- **Unsupervised, layerwise + global supervised fine-tuning**
  - Train each layer unsupervised, one after the other
  - Add a classifier layer, and retrain the whole thing supervised
  - Good when label set is poor (e.g. pedestrian detection)
- **Unsupervised pre-training often uses regularized auto-encoders**

Do we really need deep architectures?

Y LeCun
MA Ranzato

- Theoretician's dilemma: "We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?"

$$y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \qquad y = F(W^1.F(W^0.X))$$

  - kernel machines (and 2-layer neural nets) are "universal".

- Deep learning machines

$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

- Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition
  - they can represent more complex functions with less "hardware"
- We need an efficient parameterization of the class of functions that are useful for "AI" tasks (vision, audition, NLP...)

Why would deep architectures be more efficient?

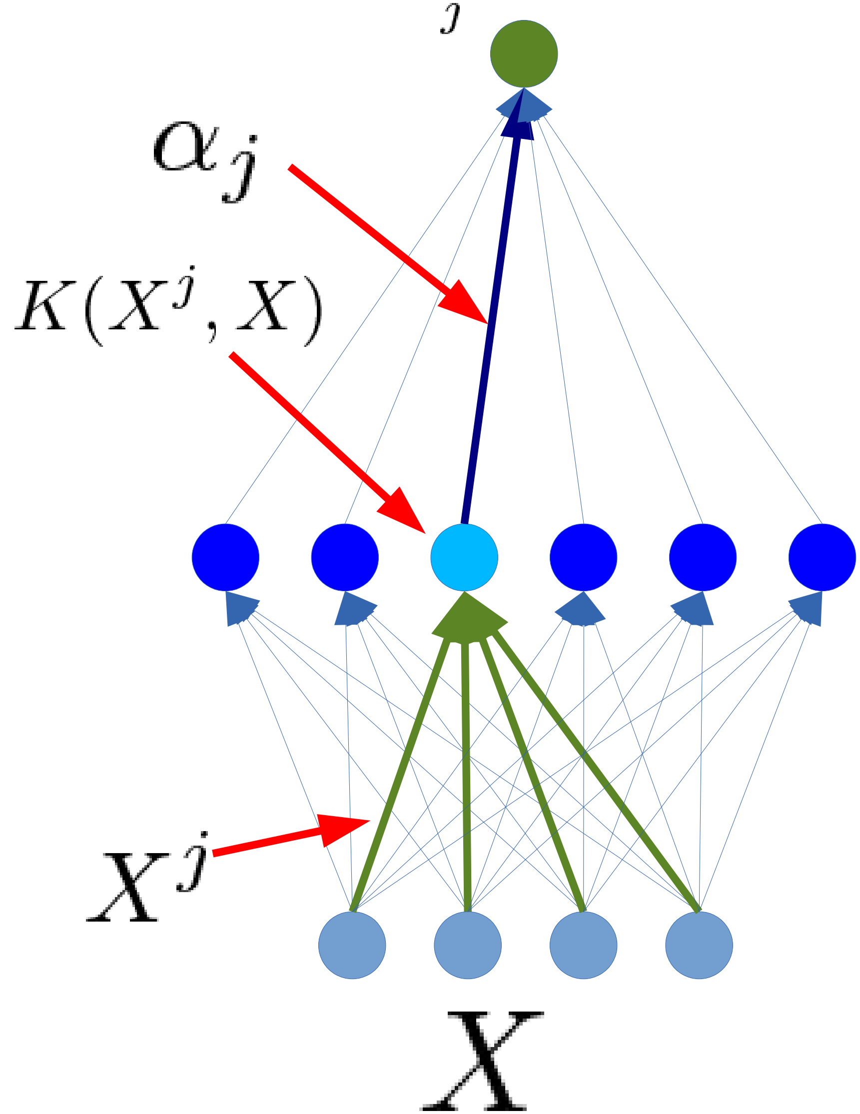
[Bengio & LeCun 2007 "Scaling Learning Algorithms Towards AI"]

- **A deep architecture trades space for time (or breadth for depth)**
  - more layers (more sequential computation),
  - but less hardware (less parallel computation).
- **Example1: N-bit parity**
  - requires N-1 XOR gates in a tree of depth log(N).
  - Even easier if we use threshold gates
  - requires an exponential number of gates of we restrict ourselves to 2 layers (DNF formula with exponential number of minterms).
- **Example2: circuit for addition of 2 N-bit binary numbers**
  - Requires O(N) gates, and O(N) layers using N one-bit adders with ripple carry propagation.
  - Requires lots of gates (some polynomial in N) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
  - Bad news: almost all boolean functions have a DNF formula with an exponential number of minterms O(2^N).....

Which Models are Deep?

- **2-layer models are not deep (even if you train the first layer)**
  - Because there is no feature hierarchy
- **Neural nets with 1 hidden layer are not deep**
- **SVMs and Kernel methods are not deep**
  - Layer1: kernels; layer2: linear
  - The first layer is "trained" in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
- **Classification trees are not deep**
  - No hierarchy of features. All decisions are made in the input space

$$G(X,\alpha) = \sum_j \alpha_j K(X^j, X)$$

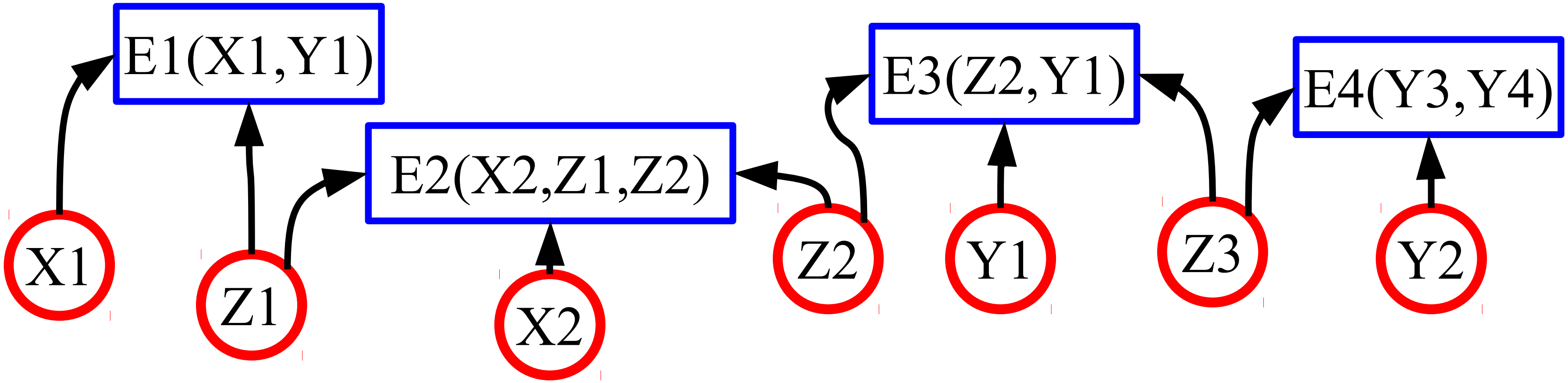


Are Graphical Models Deep?

Y LeCun
MA Ranzato

- **There is no opposition between graphical models and deep learning.**
  - Many deep learning models are formulated as factor graphs
  - Some graphical models use deep architectures inside their factors
- **Graphical models can be deep (but most are not).**
- **Factor Graph: sum of energy functions**
  - Over inputs X, outputs Y and latent variables Z. Trainable parameters: W

$$-\log P(X,Y,Z/W) \propto E(X,Y,Z,W) = \sum_i E_i(X,Y,Z,W_i)$$



- **Each energy function can contain a deep network**
- **The whole factor graph can be seen as a deep network**

Deep Learning: A Theoretician's Nightmare?

- **Deep Learning involves non-convex loss functions**
  - With non-convex losses, all bets are off
  - Then again, every speech recognition system ever deployed has used non-convex optimization (GMMs are non convex).

- **But to some of us all "interesting" learning is non convex**
  - Convex learning is invariant to the order in which sample are presented (only depends on asymptotic sample frequencies).
  - Human learning isn't like that: we learn simple concepts before complex ones. The order in which we learn things matter.

Deep Learning: A Theoretician's Nightmare?

- **No generalization bounds?**
  - Actually, the usual VC bounds apply: most deep learning systems have a finite VC dimension
  - We don't have tighter bounds than that.
  - But then again, how many bounds are tight enough to be useful for model selection?

- **It's hard to prove anything about deep learning systems**
  - Then again, if we only study models for which we can prove things, we wouldn't have speech, handwriting, and visual object recognition systems today.

Deep Learning: A Theoretician's Paradise?

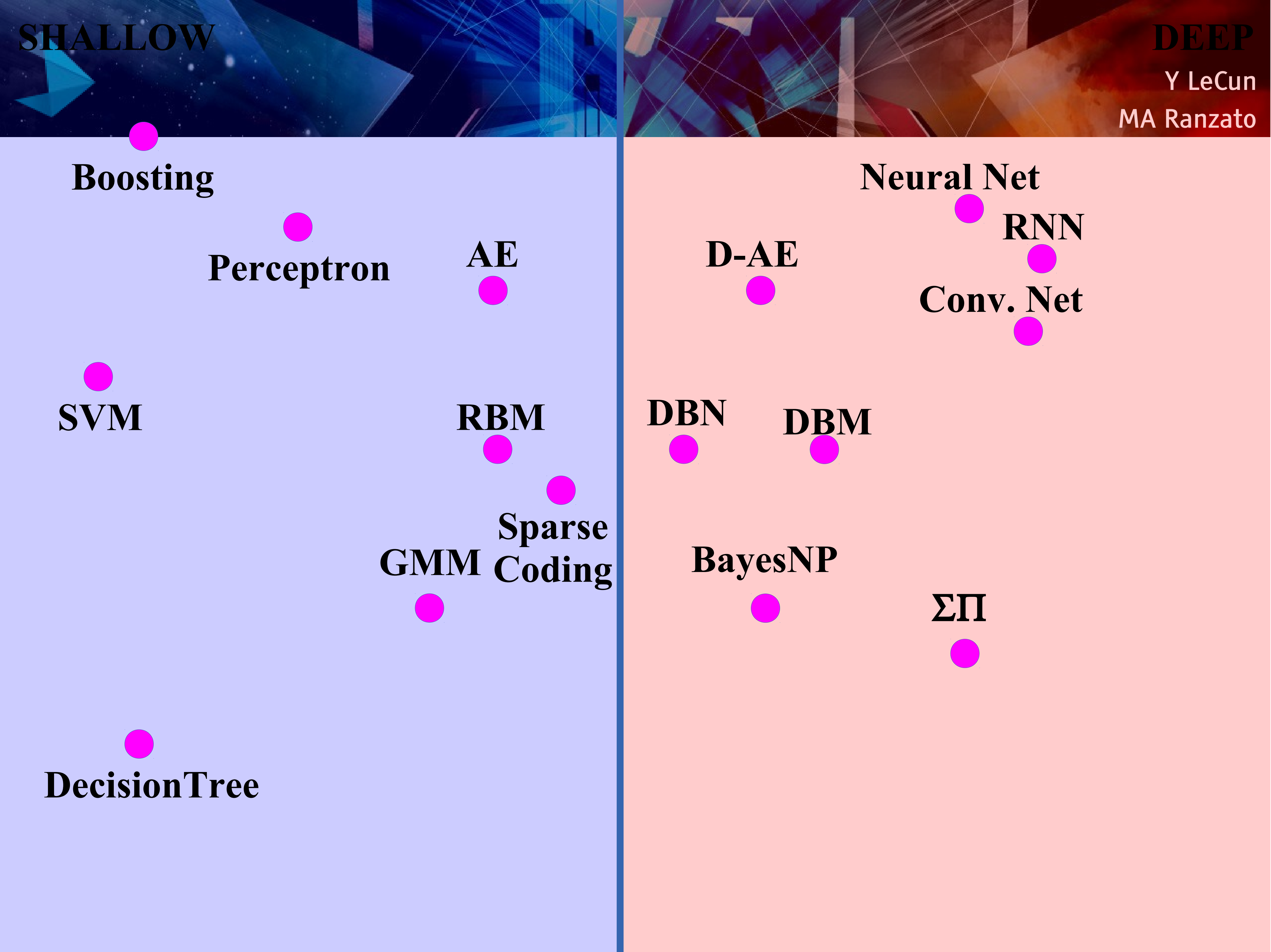
- **Deep Learning is about representing high-dimensional data**
  - There has to be interesting theoretical questions there
  - What is the geometry of natural signals?
  - Is there an equivalent of statistical learning theory for unsupervised learning?
  - What are good criteria on which to base unsupervised learning?
- **Deep Learning Systems are a form of latent variable factor graph**
  - Internal representations can be viewed as latent variables to be inferred, and deep belief networks are a particular type of latent variable models.
  - The most interesting deep belief nets have intractable loss functions: how do we get around that problem?
- **Lots of theory at the 2012 IPAM summer school on deep learning**
  - Wright's parallel SGD methods, Mallat's “scattering transform”, Osher's “split Bregman” methods for sparse modeling, Morton's “algebraic geometry of DBN”,....

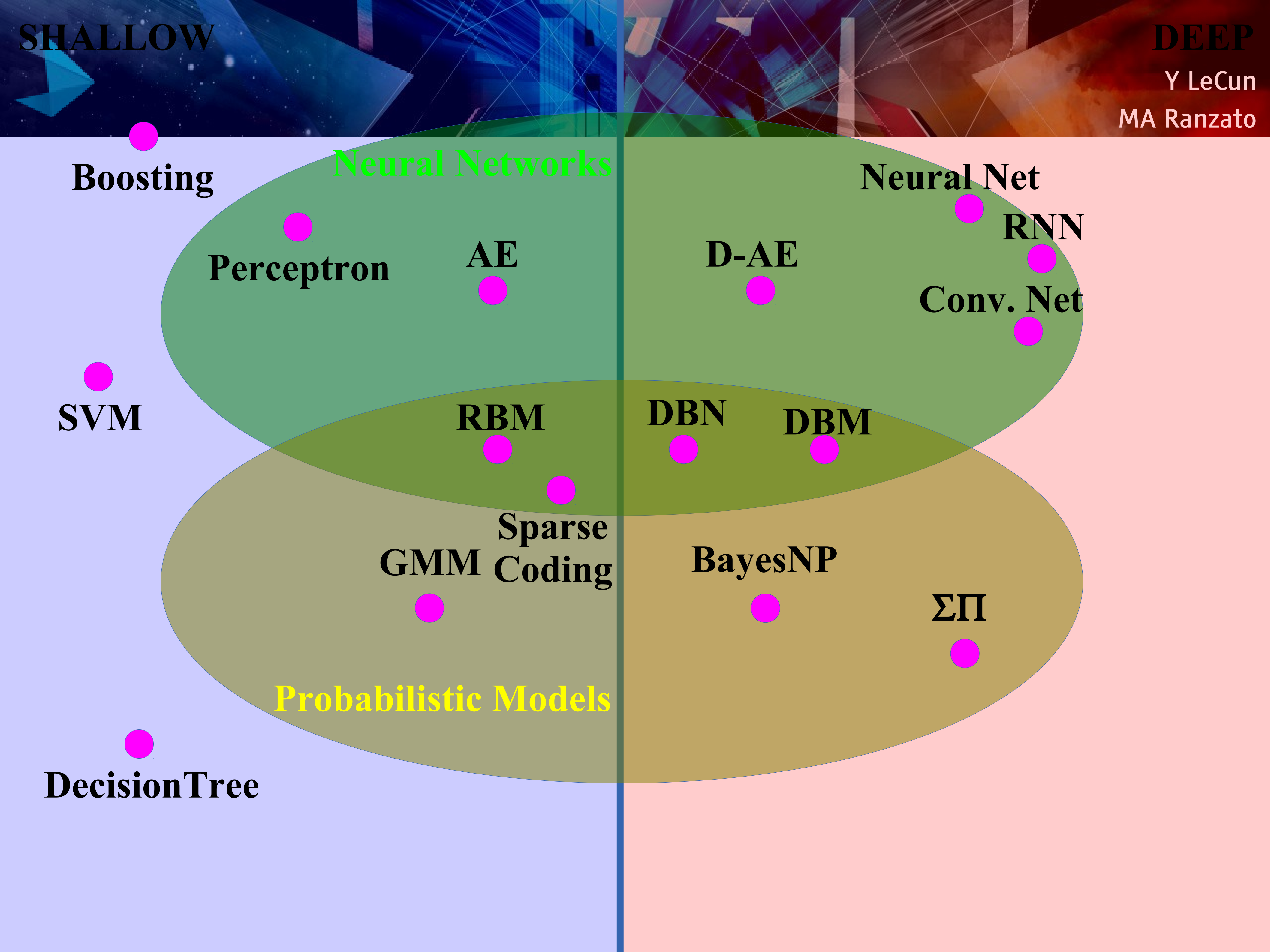
Deep Learning and Feature Learning Today

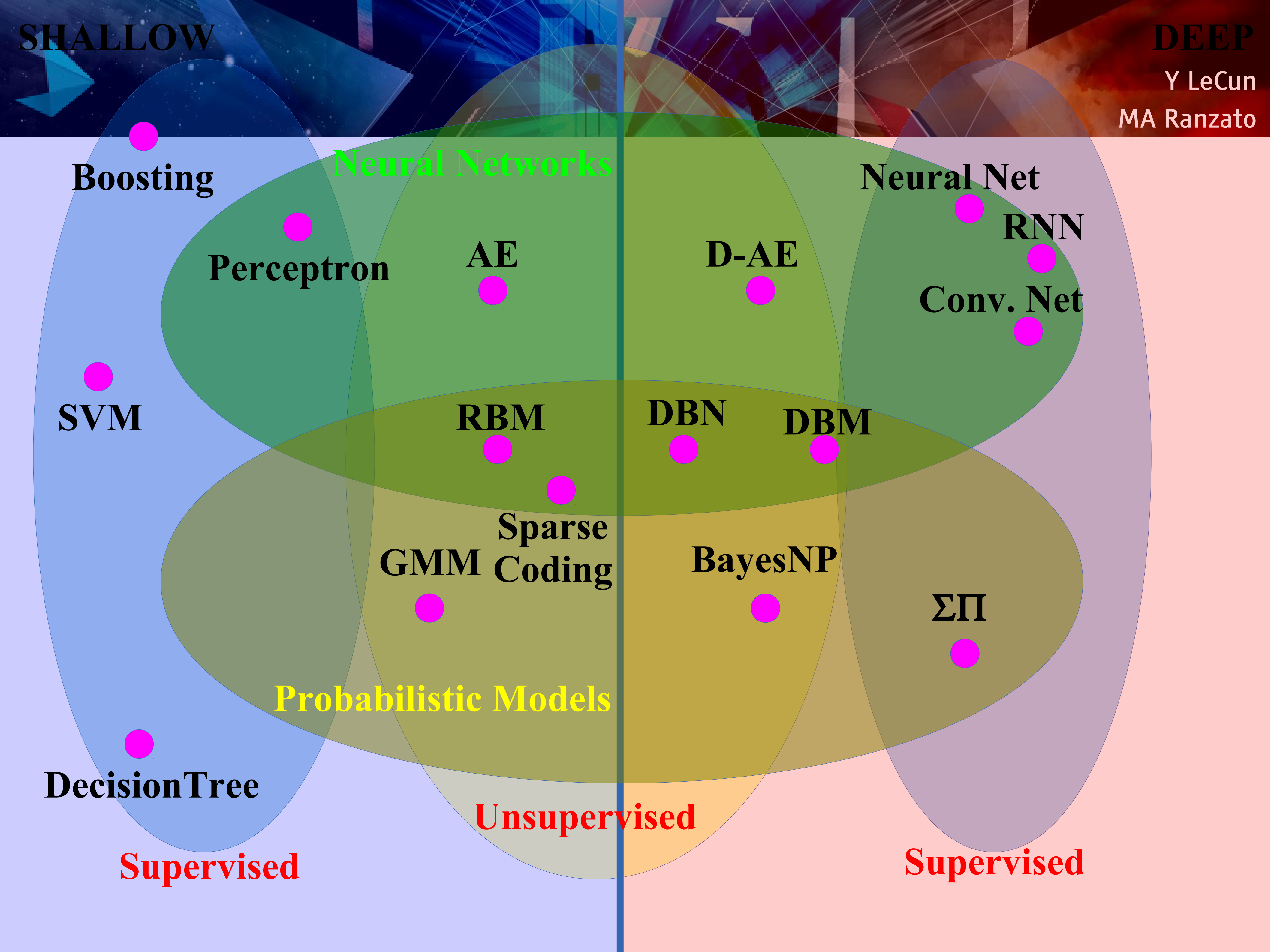
- **Deep Learning has been the hottest topic in speech recognition in the last 2 years**
  - A few long-standing performance records were broken with deep learning methods
  - Microsoft and Google have both deployed DL-based speech recognition system in their products
  - Microsoft, Google, IBM, Nuance, AT&T, and all the major academic and industrial players in speech recognition have projects on deep learning
- **Deep Learning is the hottest topic in Computer Vision**
  - Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
  - But the record holders on ImageNet and Semantic Segmentation are convolutional nets
- **Deep Learning is becoming hot in Natural Language Processing**
- **Deep Learning/Feature Learning in Applied Mathematics**
  - The connection with Applied Math is through sparse coding, non-convex optimization, stochastic gradient algorithms, etc...

In Many Fields, Feature Learning Has Caused a Revolution (methods used in commercially deployed systems)

- **Speech Recognition I (late 1980s)**
  - Trained mid-level features with Gaussian mixtures (2-layer classifier)
- **Handwriting Recognition and OCR (late 1980s to mid 1990s)**
  - Supervised convolutional nets operating on pixels
- **Face & People Detection (early 1990s to mid 2000s)**
  - Supervised convolutional nets operating on pixels (YLC 1994, 2004, Garcia 2004)
  - Haar features generation/selection (Viola-Jones 2001)
- **Object Recognition I (mid-to-late 2000s: Ponce, Schmid, Yu, YLC....)**
  - Trainable mid-level features (K-means or sparse coding)
- **Low-Res Object Recognition: road signs, house numbers (early 2010's)**
  - Supervised convolutional net operating on pixels
- **Speech Recognition II (circa 2011)**
  - Deep neural nets for acoustic modeling
- **Object Recognition III, Semantic Labeling (2012, Hinton, YLC,...)**
  - Supervised convolutional nets operating on pixels


SHALLOW
DEEP
Y LeCun
MA Ranzato
Boosting
Perceptron
AE
SVM
RBM
Sparse Coding
GMM
DecisionTree
Neural Net
RNN
D-AE
Conv. Net
DBN
DBM
BayesNP
ΣΠ


SHALLOW
DEEP
Y LeCun
MA Ranzato
Boosting
Neural Networks
Neural Net
RNN
Perceptron
AE
D-AE
Conv. Net
SVM
RBM
DBN
DBM
Sparse
Coding
GMM
BayesNP
ΣΠ
Probabilistic Models
DecisionTree



SHALLOW
DEEP
Y LeCun
MA Ranzato
Boosting
Neural Networks
Neural Net
RNN
Perceptron
AE
D-AE
Conv. Net
SVM
RBM
DBN
DBM
Sparse
Coding
GMM
BayesNP
ΣΠ
Probabilistic Models
DecisionTree
Unsupervised
Supervised
Supervised

SHALLOW

DEEP

Y LeCun
MA Ranzato

Boosting

Perceptron

AE

SVM

RBM

Sparse Coding

GMM

DecisionTree

Neural Net

RNN

D-AE

Conv. Net

DBN

DBM

BayesNP

ΣΠ

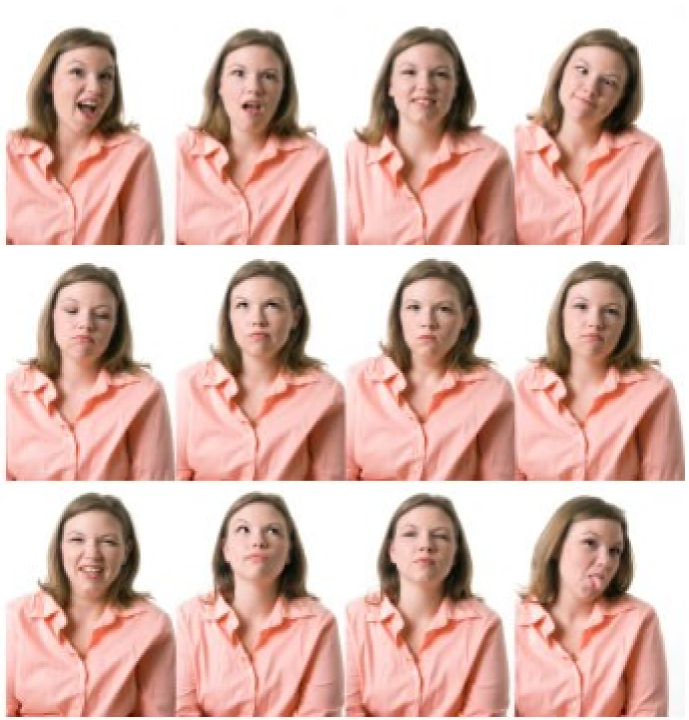
In this talk, we'll focus on **the simplest and typically most effective methods**.

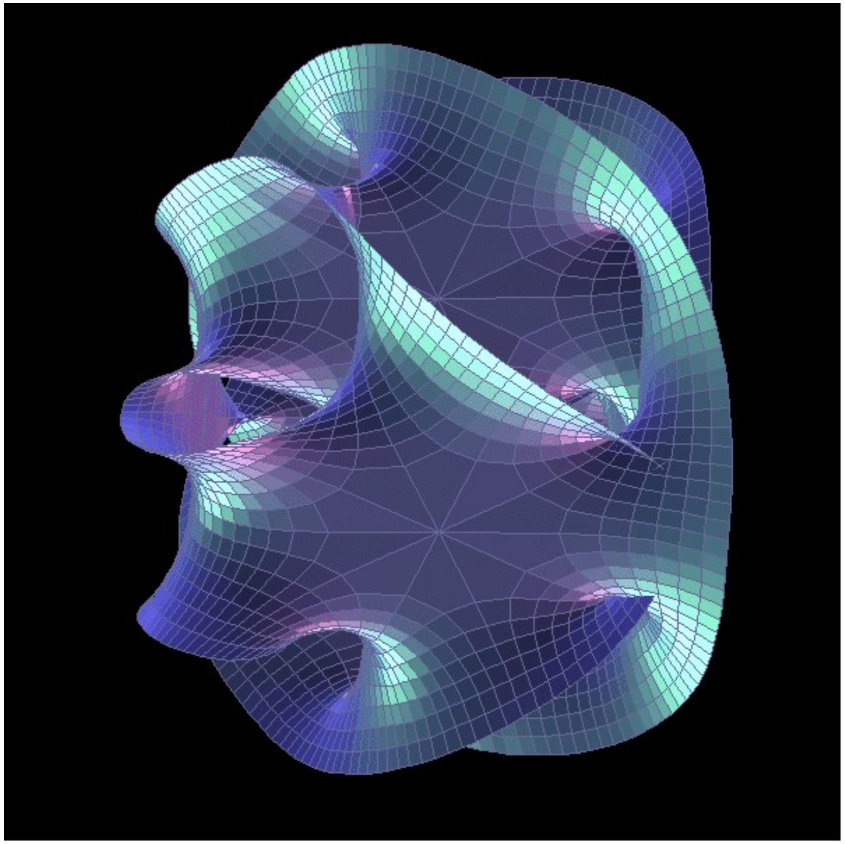
What Are
Good Feature?

Discovering the Hidden Structure in High-Dimensional Data
The manifold hypothesis

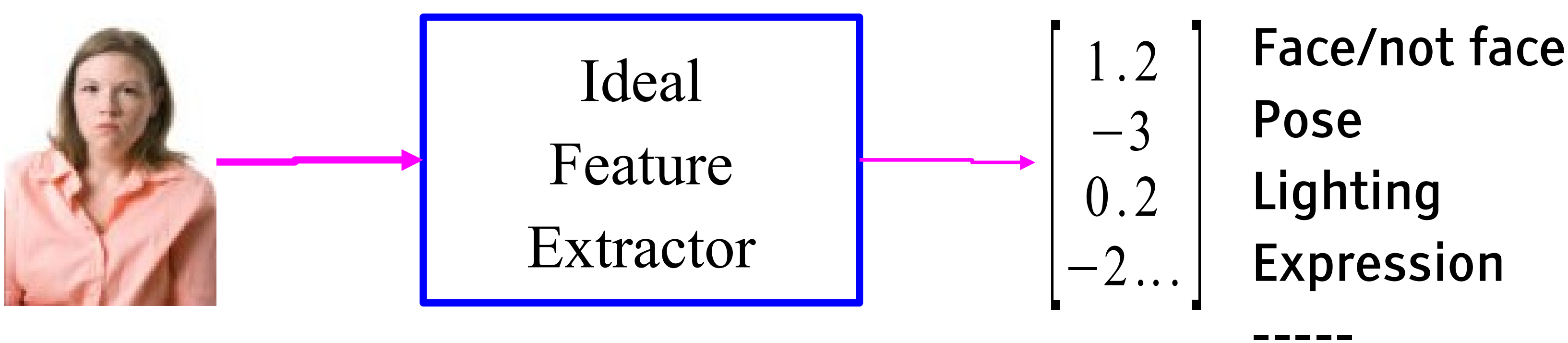
Y LeCun
MA Ranzato

- Learning Representations of Data:
  - **Discovering & disentangling the independent explanatory factors**
- The Manifold Hypothesis:
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data





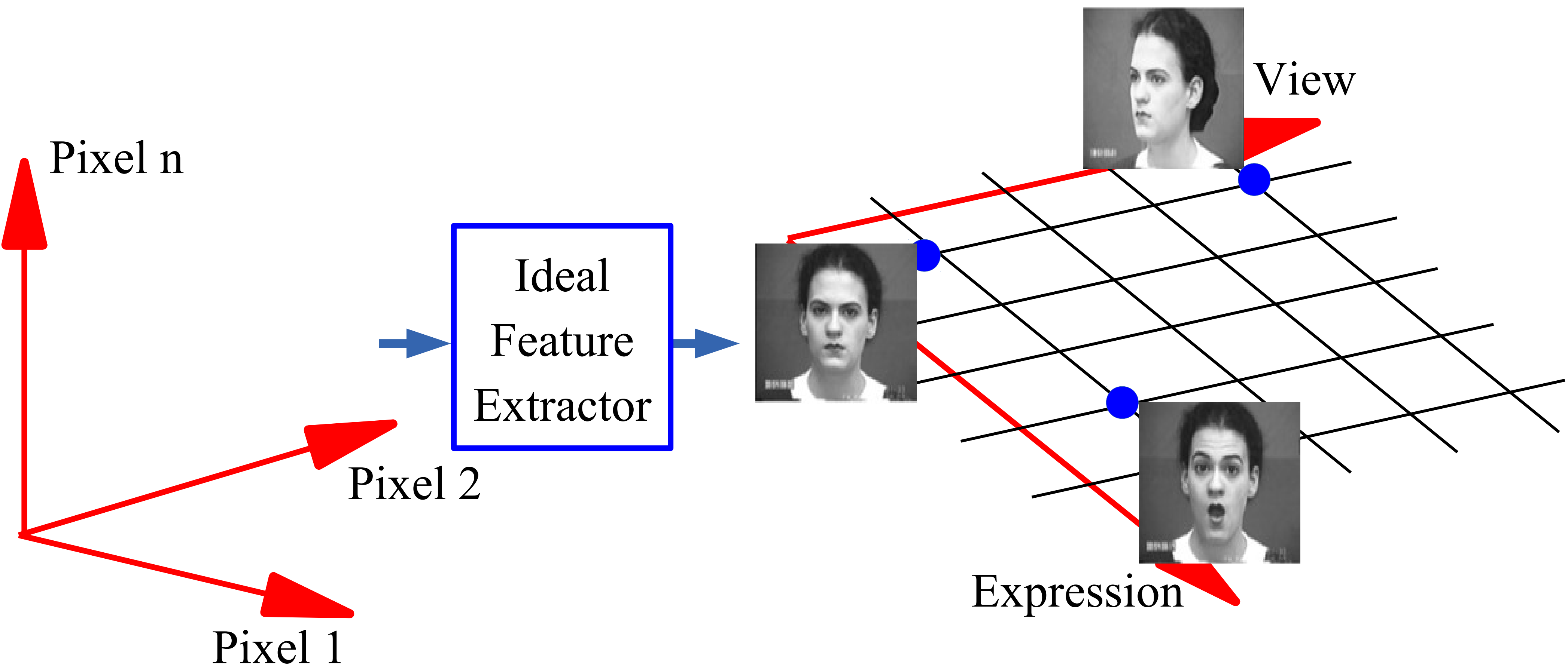
- **Example: all face images of a person**
  - 1000x1000 pixels = 1,000,000 dimensions
  - But the face has 3 cartesian coordinates and 3 Euler angles
  - And humans have less than about 50 muscles in the face
  - Hence the manifold of face images for a person has <56 dimensions
- **The perfect representations of a face image:**
  - Its coordinates on the face manifold
  - Its coordinates away from the manifold
- **We do not have good and general methods to learn functions that turns an image into this kind of representation**



Disentangling factors of variation

Y LeCun
MA Ranzato

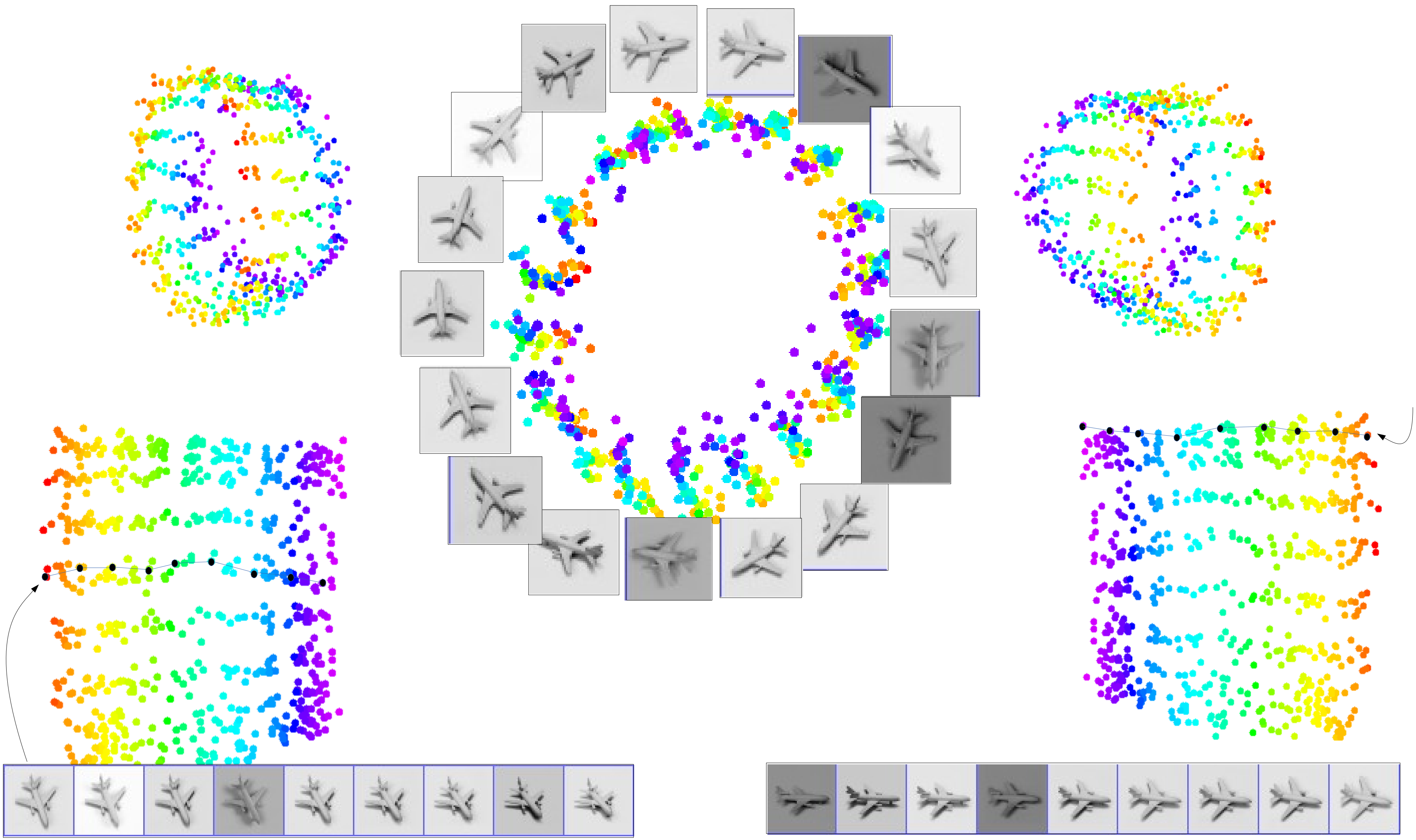
The Ideal Disentangling Feature Extractor



Data Manifold & Invariance: Some variations must be eliminated

Y LeCun
MA Ranzato

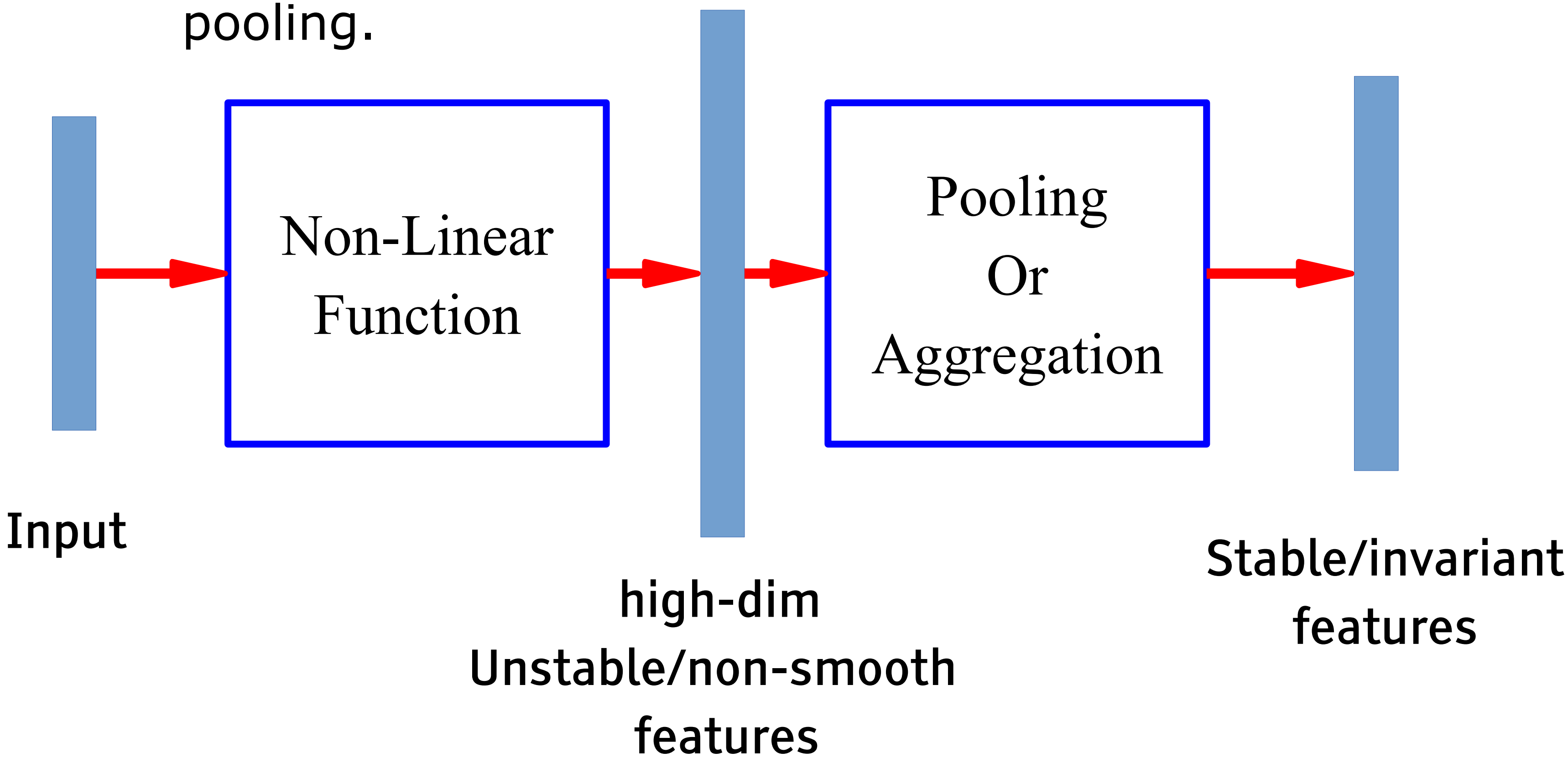
- Azimuth-Elevation manifold. Ignores lighting. [Hadsell et al. CVPR 2006]



Basic Idea fpr Invariant Feature Learning

Y LeCun
MA Ranzato

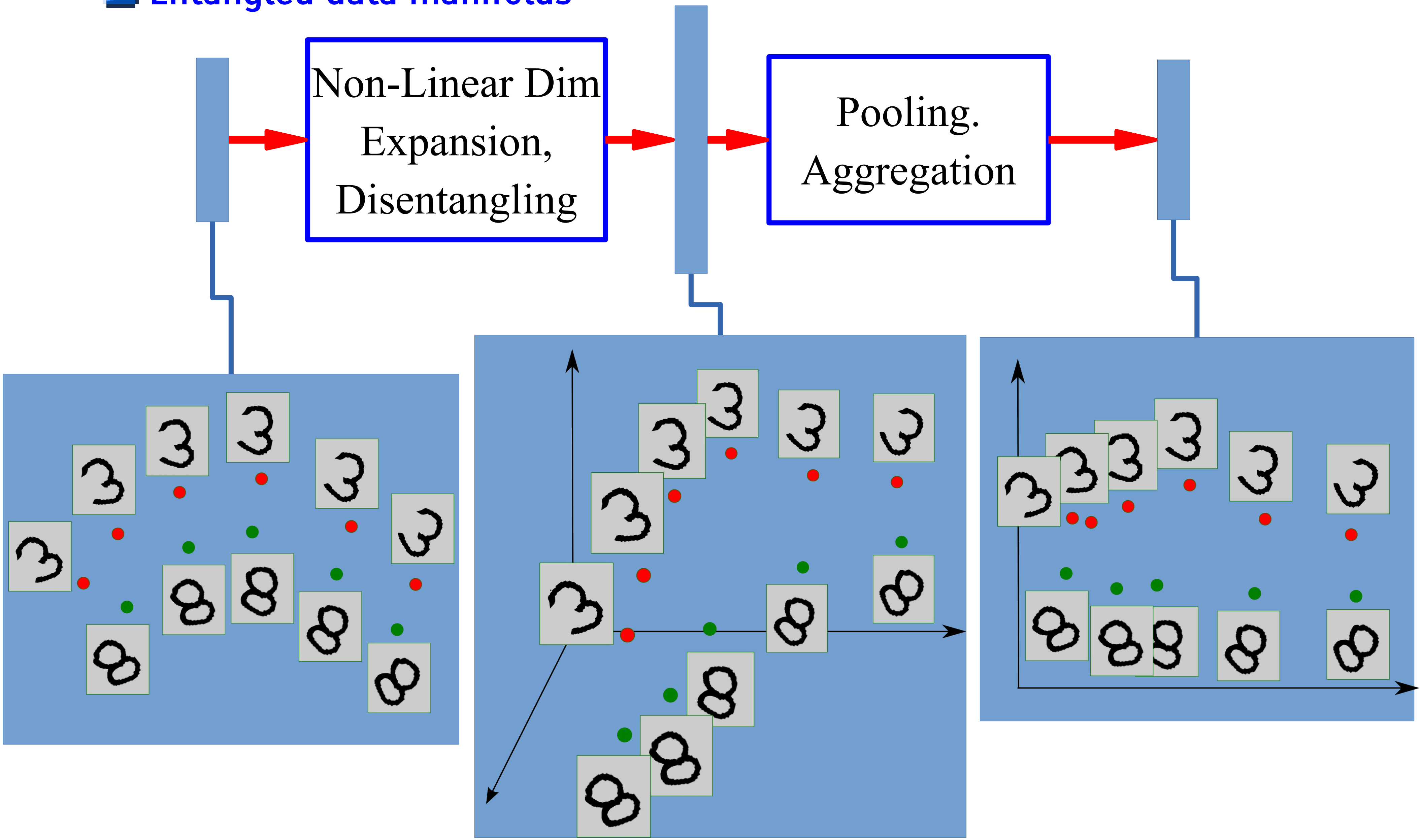
- Embed the input non-linearly into a high(er) dimensional space
  - In the new space, things that were non separable may become separable
- Pool regions of the new space together
  - Bringing together things that are semantically similar. Like pooling.



Non-Linear Expansion → Pooling

Y LeCun
MA Ranzato

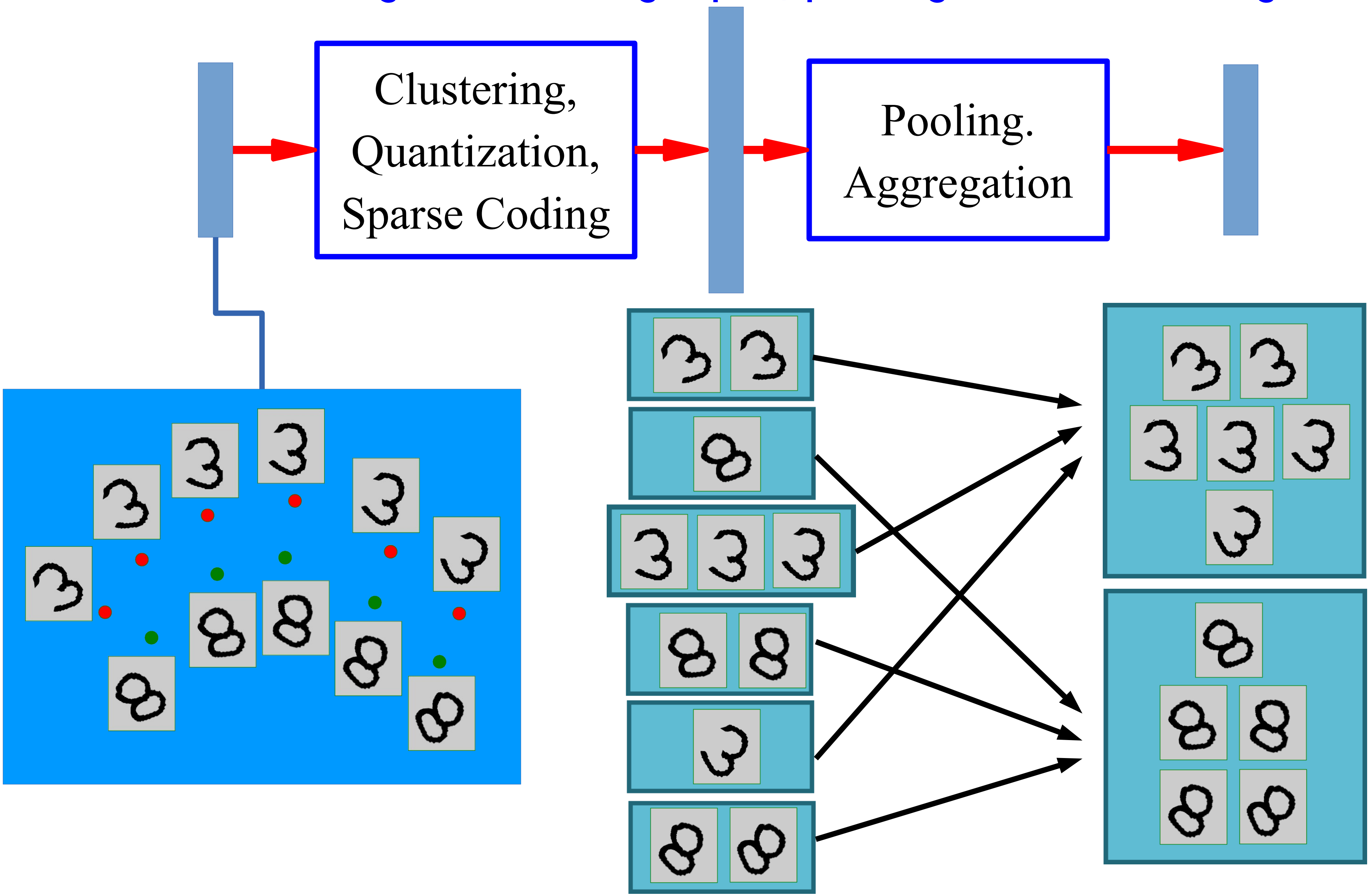
- **Entangled data manifolds**



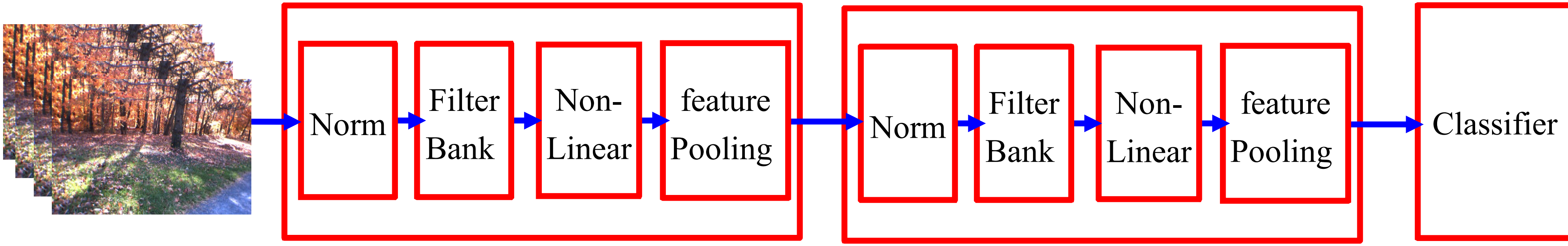
Sparse Non-Linear Expansion → Pooling

Y LeCun
MA Ranzato

- Use clustering to break things apart, pool together similar things



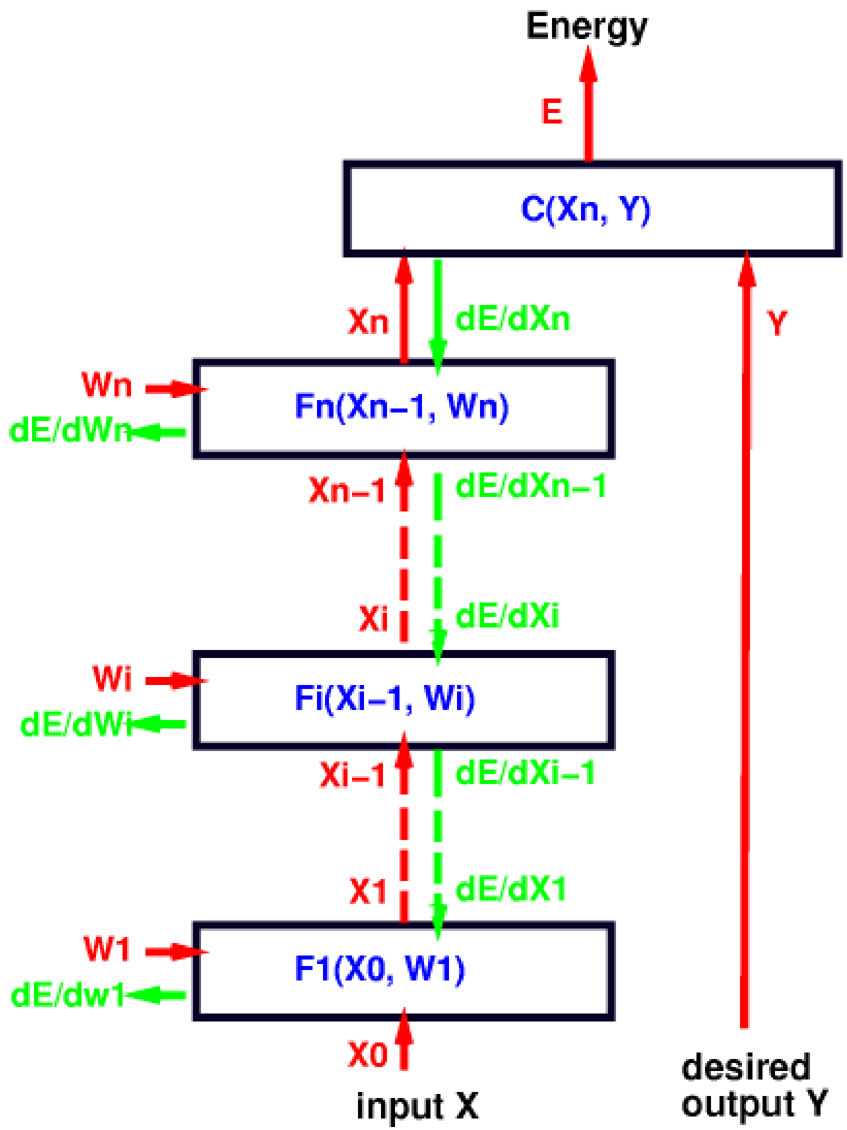
Overall Architecture: Normalization → Filter Bank → Non-Linearity → Pooling



- **Stacking multiple stages of**
  - [Normalization → Filter Bank → Non-Linearity → Pooling].
- **Normalization: variations on whitening**
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank: dimension expansion, projection on overcomplete basis**
- **Non-Linearity: sparsification, saturation, lateral inhibition....**
  - Rectification (ReLU), Component-wise shrinkage, tanh, winner-takes-all
- **Pooling: aggregation over space or feature type**
  - $X_i;\quad L_p: \sqrt[p]{X_i^p};\quad PROB: \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$

Deep Supervised Learning (modular approach)

Multimodule Systems: Cascade



- Complex learning machines can be built by assembling modules into networks
- Simple example: sequential/layered feed-forward architecture (cascade)
- Forward Propagation:

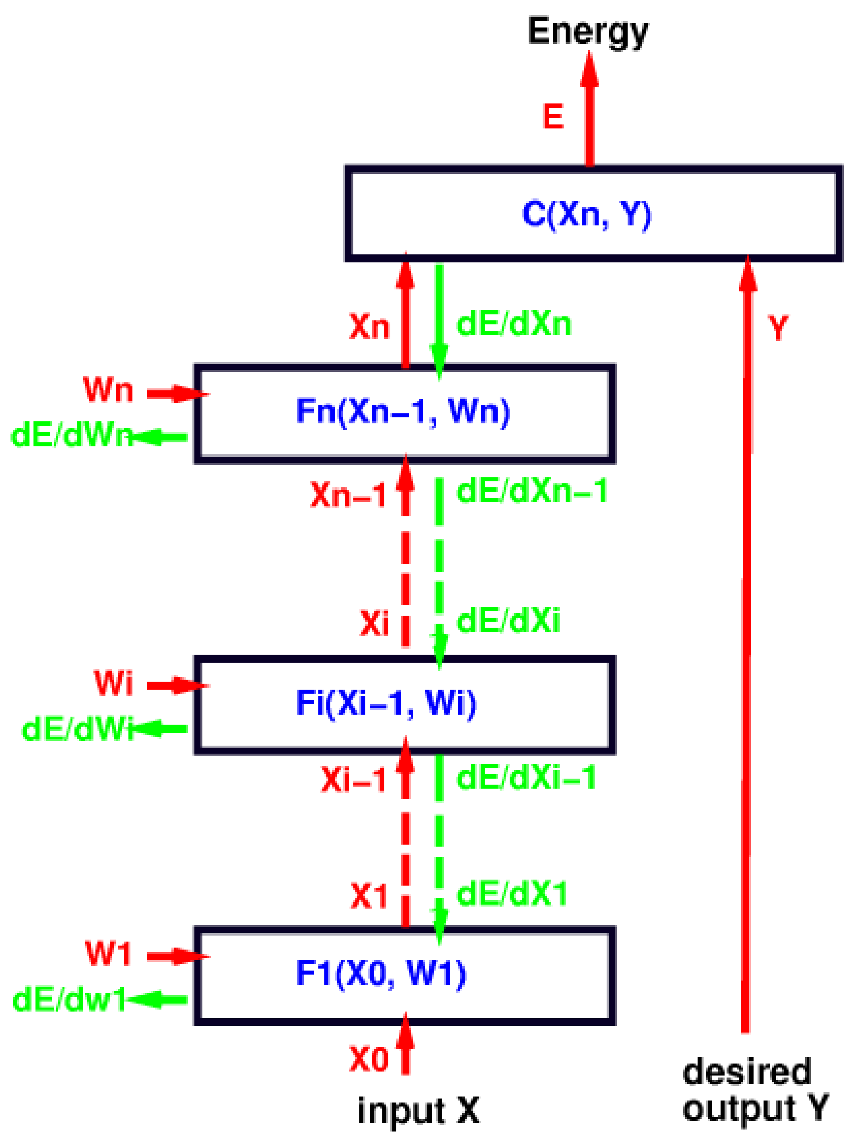
$$\text{let } X = X_0,$$

$$X_i = F_i(X_{i-1}, W_i) \quad \forall i \in [1, n]$$

$$E(Y, X, W) = C(X_n, Y)$$

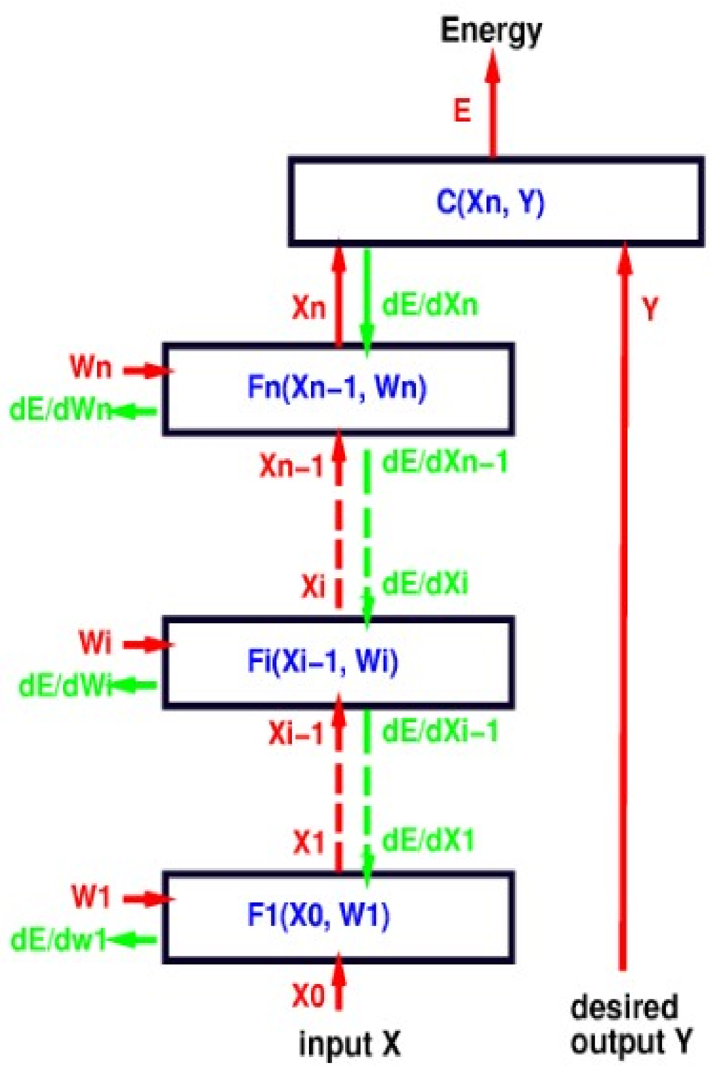
Multimodule Systems: Implementation

Y LeCun
MA Ranzato



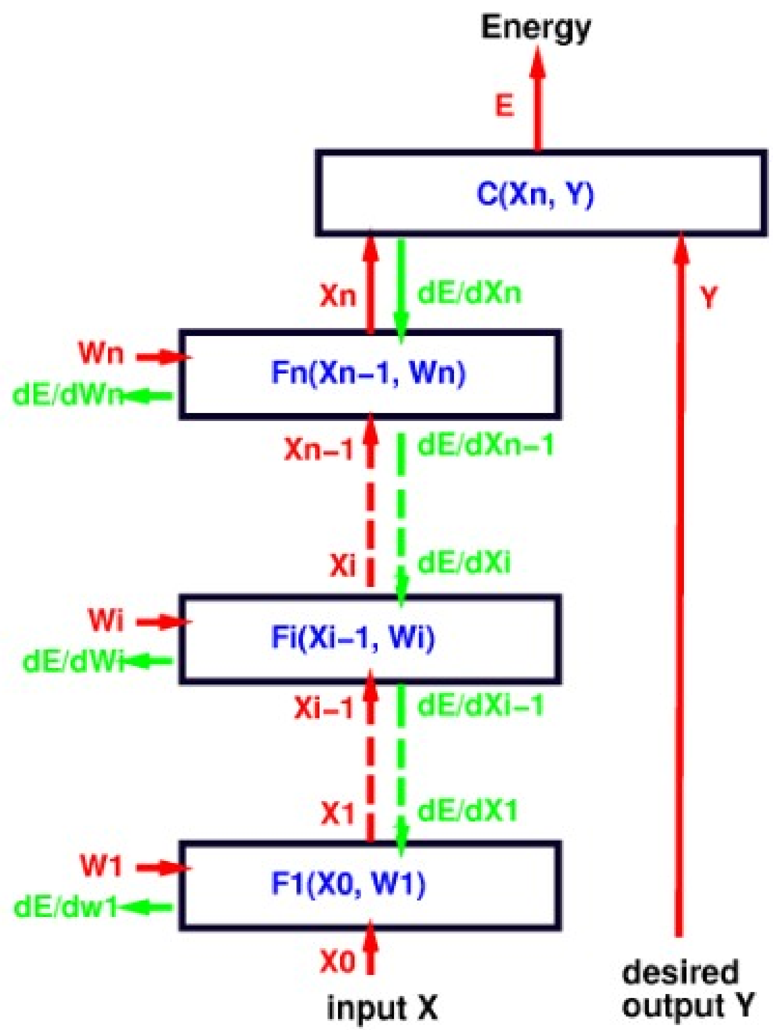
- **Each module is an object**
  - Contains trainable parameters
  - Inputs are arguments
  - Output is returned, but also stored internally
  - Example: 2 modules m1, m2
- **Torch7 (by hand)**
  - `hid = m1:forward(in)`
  - `out = m2:forward(hid)`
- **Torch7 (using the nn.Sequential class)**
  - `model = nn.Sequential()`
  - `model:add(m1)`
  - `model:add(m2)`
  - `out = model:forward(in)`

Computing the Gradient in Multi-Layer Systems



- To train a multi-module system, we must compute the gradient of E with respect to all the parameters in the system (all the W_i).
- Let's consider module i whose fprop method computes $X_i = F_i(X_{i-1}, W_i)$.
- Let's assume that we already know $\frac{\partial E}{\partial X_i}$, in other words, for each component of vector X_i we know how much E would wiggle if we wiggled that component of X_i.

Computing the Gradient in Multi-Layer Systems



- We can apply chain rule to compute $\frac{\partial E}{\partial W_i}$ (how much E would wiggle if we wiggled each component of W_i):

$$\frac{\partial E}{\partial W_i} = \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i}$$

$$[1 \times N_w] = [1 \times N_x].[N_x \times N_w]$$

- $\frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i}$ is the *Jacobian matrix* of F_i with respect to W_i.

$$\left[\frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i}\right]_{kl} = \frac{\partial \left[F_i(X_{i-1}, W_i)\right]_k}{\partial [W_i]_l}$$

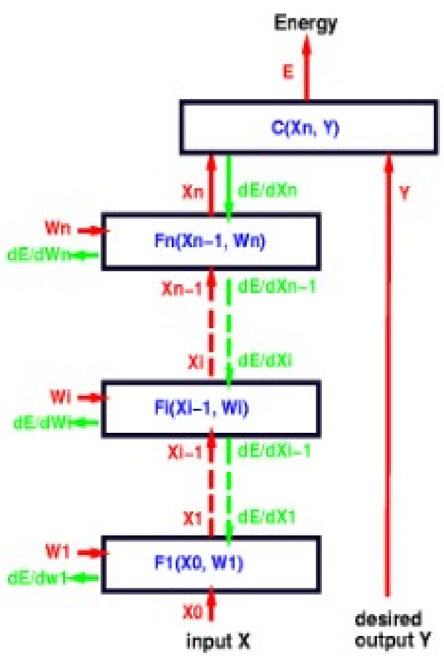
- Element (k, l) of the Jacobian indicates how much the k-th output wiggles when we wiggle the l-th weight.

Computing the Gradient in Multi-Layer Systems

Using the same trick, we can compute $\frac{\partial E}{\partial X_{i-1}}$. Let's assume again that we already know $\frac{\partial E}{\partial X_i}$, in other words, for each component of vector X_i we know how much E would wiggle if we wiggled that component of X_i.

- We can apply chain rule to compute $\frac{\partial E}{\partial X_{i-1}}$ (how much E would wiggle if we wiggled each component of X_{i-1}):

$$\frac{\partial E}{\partial X_{i-1}} = \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}}$$



- $\frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}}$ is the *Jacobian matrix* of F_i with respect to X_{i-1}.
- F_i has two Jacobian matrices, because it has to arguments.
- Element (k, l) of this Jacobian indicates how much the k-th output wiggles when we wiggle the l-th input.
- **The equation above is a recurrence equation!**

Jacobians and Dimensions

- derivatives with respect to a column vector are line vectors (dimensions: $[1 \times N_{i-1}] = [1 \times N_i] * [N_i \times N_{i-1}]$)

$$\frac{\partial E}{\partial X_{i-1}} = \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}}$$

- (dimensions: $[1 \times N_{wi}] = [1 \times N_i] * [N_i \times N_{wi}]$):

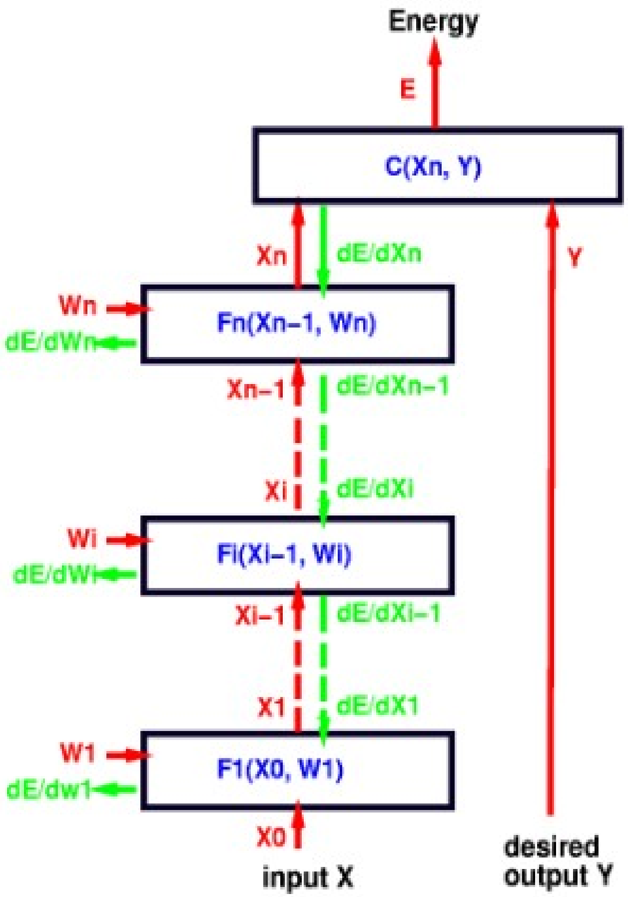
$$\frac{\partial E}{\partial W_i} = \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial W}$$

- we may prefer to write those equation with column vectors:

$$\frac{\partial E}{\partial X_{i-1}}' = \frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}}' \frac{\partial E}{\partial X_i}'$$

$$\frac{\partial E}{\partial W_i}' = \frac{\partial F_i(X_{i-1}, W_i)}{\partial W}' \frac{\partial E}{\partial X_i}'$$

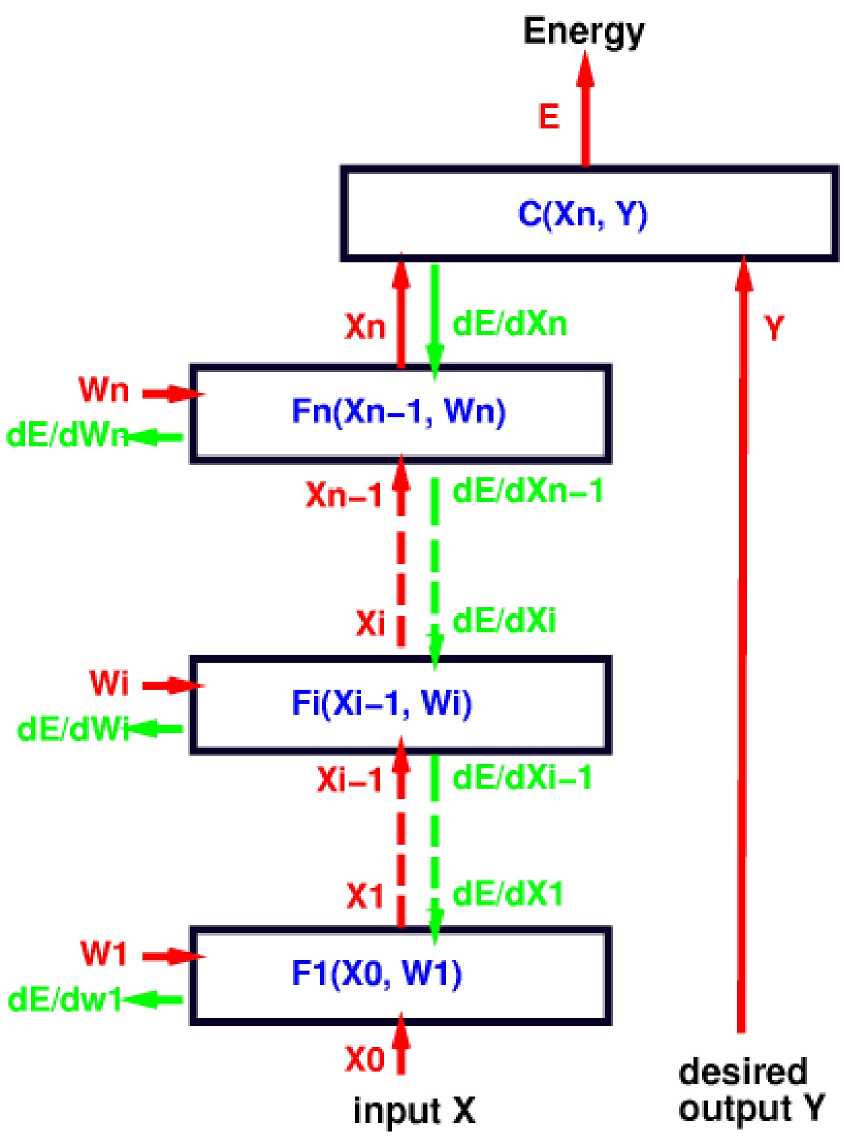
To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$



- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$.

Multimodule Systems: Implementation

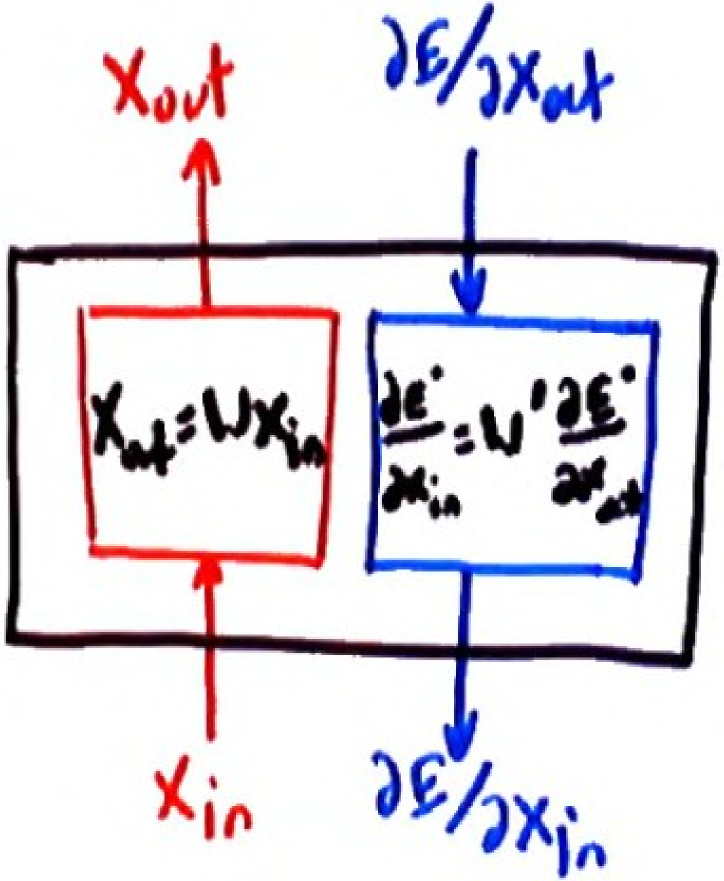
Y LeCun
MA Ranzato



- **Backpropagation through a module**
  - Contains trainable parameters
  - Inputs are arguments
  - Gradient with respect to input is returned.
  - Arguments are input and gradient with respect to output
- **Torch7 (by hand)**
  - `hidg = m2:backward(hid,outg)`
  - `ing = m1:backward(in,hidg)`
- **Torch7 (using the nn.Sequential class)**
  - `ing = model:backward(in,outg)`

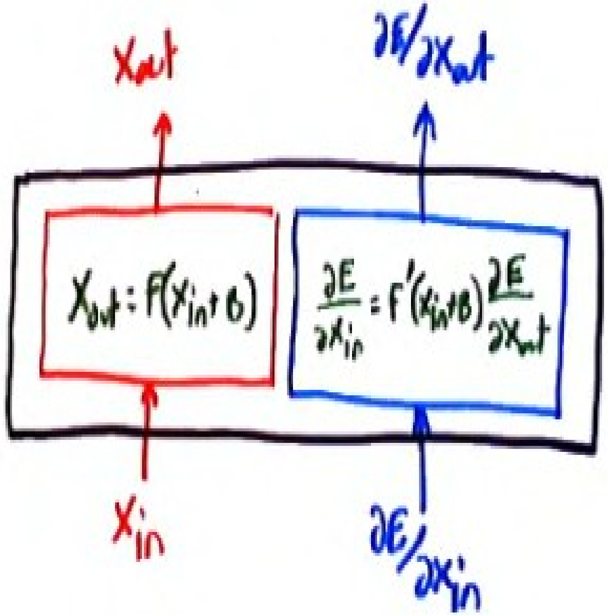
Linear Module

The input vector is multiplied by the weight matrix.



- fprop: $X_{\text{out}} = W X_{\text{in}}$
- bprop to input:
  $\frac{\partial E}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} \frac{\partial X_{\text{out}}}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} W$
- by transposing, we get column vectors:
  $\frac{\partial E}{\partial X_{\text{in}}}' = W' \frac{\partial E}{\partial X_{\text{out}}}'$
- bprop to weights:
  $\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial X_{\text{out}i}} \frac{\partial X_{\text{out}i}}{\partial W_{ij}} = X_{\text{in}j} \frac{\partial E}{\partial X_{\text{out}i}}$
- We can write this as an outer-product:
  $\frac{\partial E}{\partial W}' = \frac{\partial E}{\partial X_{\text{out}}}' X_{in}'$

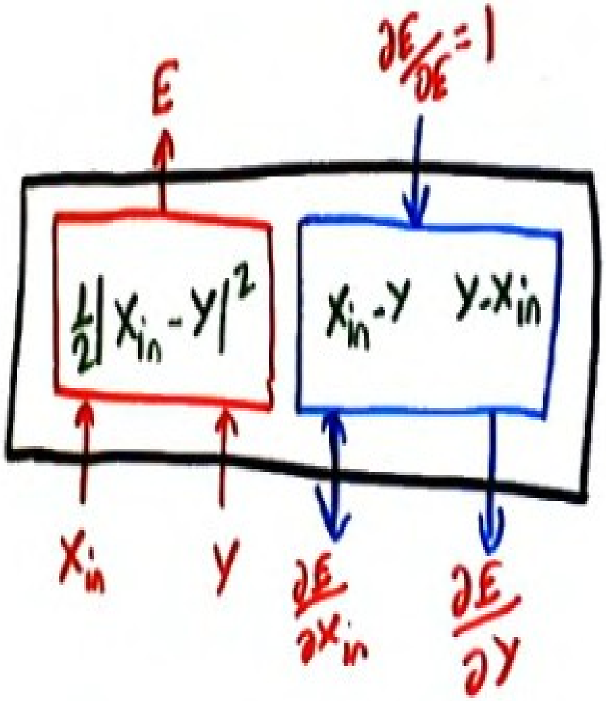
Tanh module (or any other pointwise function)



- fprop: $(X_{\text{out}})_i = \tanh((X_{\text{in}})_i + B_i)$
- bprop to input:
  $(\frac{\partial E}{\partial X_{\text{in}}})_i = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- bprop to bias:
  $\frac{\partial E}{\partial B_i} = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- $\tanh(x) = \frac{2}{1+\exp -x} - 1 = \frac{1-\exp(-x)}{1+\exp(-x)}$

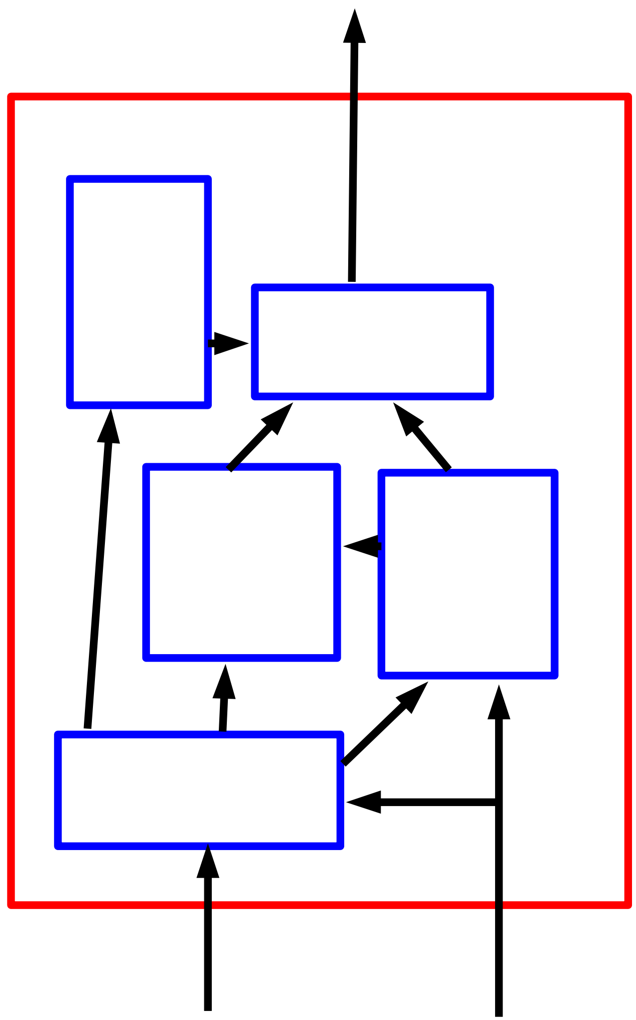
Euclidean Distance Module

Y LeCun
MA Ranzato



- fprop: $X_{\text{out}} = \frac{1}{2}||X_{\text{in}} - Y||^2$
- bprop to X input: $\frac{\partial E}{\partial X_{\text{in}}} = X_{\text{in}} - Y$
- bprop to Y input: $\frac{\partial E}{\partial Y} = Y - X_{\text{in}}$

Any Architecture works



- **Any connection is permissible**
  - Networks with loops must be "unfolded in time".
- **Any module is permissible**
  - As long as it is continuous and differentiable almost everywhere with respect to the parameters, and with respect to non-terminal inputs.

Module-Based Deep Learning with Torch7

Y LeCun
MA Ranzato

- **Torch7 is based on the Lua language**
  - Simple and lightweight scripting language, dominant in the game industry
  - Has a native just-in-time compiler (fast!)
  - Has a simple foreign function interface to call C/C++ functions from Lua
- **Torch7 is an extension of Lua with**
  - A multidimensional array engine with CUDA and OpenMP backends
  - A machine learning library that implements multilayer nets, convolutional nets, unsupervised pre-training, etc
  - Various libraries for data/image manipulation and computer vision
  - A quickly growing community of users
- **Single-line installation on Ubuntu and Mac OSX:**
  - curl -s https://raw.github.com/clementfarabet/torchinstall/master/install | bash
- **Torch7 Machine Learning Tutorial (neural net, convnet, sparse auto-encoder):**
  - http://code.cogbits.com/wiki/doku.php

Example: building a Neural Net in Torch7

- Net for SVHN digit recognition
- 10 categories
- Input is 32x32 RGB (3 channels)

```
Noutputs = 10;
nfeats = 3; Width = 32; height = 32
ninputs = nfeats*width*height
nhiddens = 1500
```

- 1500 hidden units

- Creating a 2-layer net
- Make a cascade module
- Reshape input to vector
- Add Linear module
- Add tanh module
- Add Linear Module
- Add log softmax layer

```
-- Simple 2-layer neural network
model = nn.Sequential()
model:add(nn.Reshape(ninputs))
model:add(nn.Linear(ninputs,nhiddens))
model:add(nn.Tanh())
model:add(nn.Linear(nhiddens,noutputs))
model:add(nn.LogSoftMax())
```

- Create loss function module

```
criterion = nn.ClassNLLCriterion()
```

See Torch7 example at http://bit.ly/16tyLAx

Example: Training a Neural Net in Torch7

Y LeCun
MA Ranzato

```
for t = 1,trainData:size(),batchSize do
  inputs,outputs = getNextBatch()
  local feval = function(x)
    parameters:copy(x)
    gradParameters:zero()
    local f = 0
    for i = 1,#inputs do
      local output = model:forward(inputs[i])
      local err = criterion:forward(output,targets[i])
      f = f + err
      local df_do = criterion:backward(output,targets[i])
      model:backward(inputs[i], df_do)
    end
    gradParameters:div(#inputs)
    f = f/#inputs
    return f,gradParameters
  end   – of feval
  optim.sgd(feval,parameters,optimState)
end
```

- one epoch over training set
- Get next batch of samples
- Create a “closure” feval(x) that takes the parameter vector as argument and returns the loss and its gradient on the batch.
- Run model on batch
- backprop
- Normalize by size of batch
- Return loss and gradient
- call the stochastic gradient optimizer

Toy Code (Matlab): Neural Net Trainer

Y LeCun
MA Ranzato

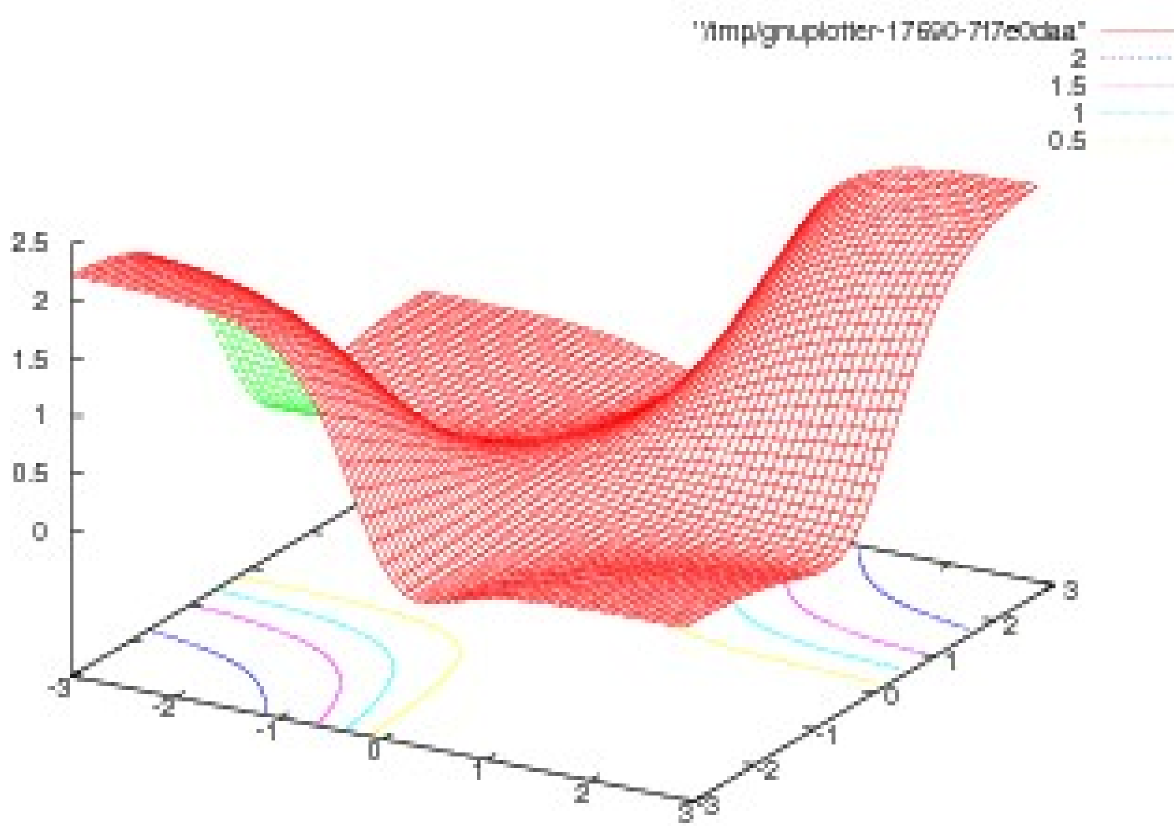
```
% F-PROP
for i = 1 : nr_layers - 1
  [h{i} jac{i}] = nonlinearity(W{i} * h{i-1} + b{i});
end
h{nr_layers-1} = W{nr_layers-1} * h{nr_layers-2} + b{nr_layers-1};
prediction = softmax(h{l-1});
% CROSS ENTROPY LOSS
loss = - sum(sum(log(prediction) .* target)) / batch_size;
% B-PROP
dh{l-1} = prediction - 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
```

Deep Supervised Learning is Non-Convex

Y LeCun
MA Ranzato

- **Example: what is the loss function for the simplest 2-layer neural net ever**
  - Function: 1-1-1 neural net. Map 0.5 to 0.5 and -0.5 to -0.5 (identity function) with quadratic cost:

$$y = \tanh(W_1 \tanh(W_0.x)) \qquad L = (0.5 - \tanh(W_1 \tanh(W_0 0.5)^2$$



Backprop in Practice

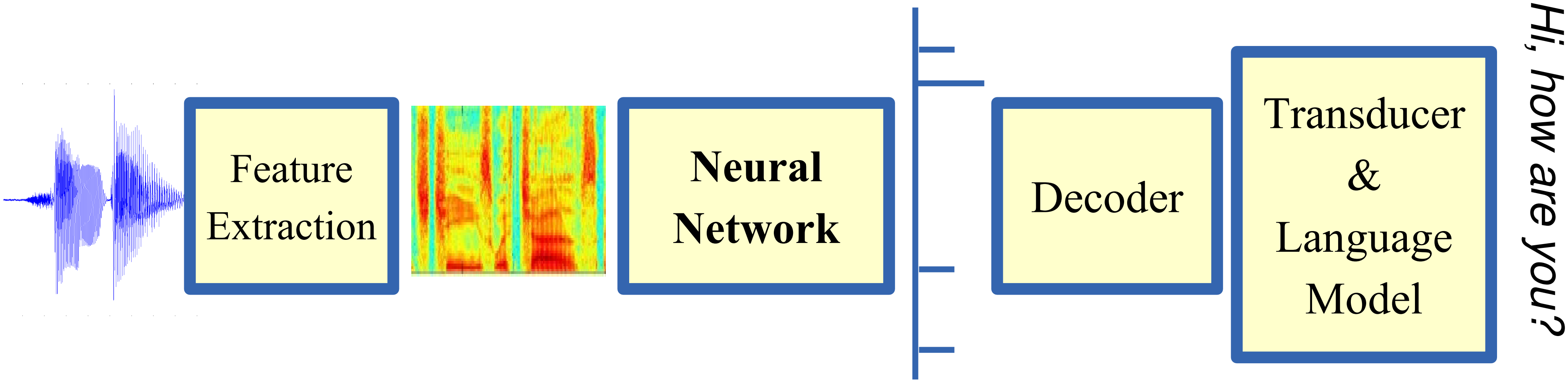
- Use ReLU non-linearities (tanh and logistic are falling out of favor)
- Use cross-entropy loss for classification
- Use Stochastic Gradient Descent on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 or L2 regularization on the weights (or a combination)
  - But it's best to turn it on after a couple of epochs
- Use "dropout" for regularization
  - Hinton et al 2012 http://arxiv.org/abs/1207.0580
- Lots more in [LeCun et al. "Efficient Backprop" 1998]
- Lots, lots more in "Neural Networks, Tricks of the Trade" (2012 edition) edited by G. Montavon, G. B. Orr, and K-R Müller (Springer)

Deep Learning
In Speech Recognition

Case study #1: Acoustic Modeling

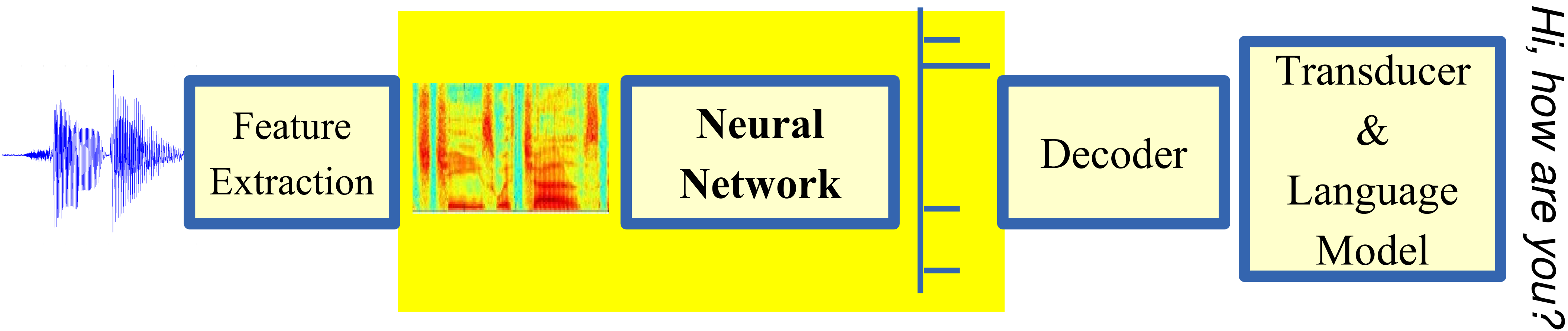
Y LeCun
MA Ranzato

A typical speech recognition system:



Case study #1: Acoustic Modeling

A typical speech recognition system:



- Here, we focus only on the prediction of phone states from short time-windows of spectrogram.
- For simplicity, we will use a fully connected neural network (in practice, a convolutional net does better).

Mohamed et al. "DBNs for phone recognition" NIPS Workshop 2009
Zeiler et al. "On rectified linear units for speech recognition" ICASSP 2013

Data

Y LeCun
MA Ranzato

- US English: Voice Search, Voice Typing, Read data
- Billions of training samples
- Input: log-energy filter bank outputs
  - 40 frequency bands
  - 26 input frames
- Output: 8000 phone states

Zeiler et al. “On rectified linear units for speech recognition” ICASSP 2013

Architecture

Y LeCun
MA Ranzato

- From 1 to 12 hidden layers
- For simplicity, the same number of hidden units at each layer:

$1040 \rightarrow 2560 \rightarrow 2560 \rightarrow \ldots \rightarrow 2560 \rightarrow 8000$

- Non-linearities: __/ output = max(0, input)

Zeiler et al. "On rectified linear units for speech recognition" ICASSP 2013

Energy & Loss

- Since it is a standard classification problem, the energy is:

$$E(\boldsymbol{x},\boldsymbol{y}) = -\boldsymbol{y}\, f(\boldsymbol{x}) \qquad \boldsymbol{y} \text{ 1-of-N vector}$$

- The loss is the negative log-likelihood:

$$L = E(\boldsymbol{x},\boldsymbol{y}) + \log\left(\sum_{\bar{\boldsymbol{y}}} \exp(-E(\boldsymbol{x},\bar{\boldsymbol{y}}))\right)$$

Zeiler et al. “On rectified linear units for speech recognition” ICASSP 2013

Optimization

Y LeCun
MA Ranzato

- SGD with schedule on learning rate

$$\theta_t \leftarrow \theta_{t-1} - \eta_t \frac{\partial L}{\partial \theta_{t-1}}$$

$$\eta_t = \frac{\eta}{max\left(1, \frac{t}{T}\right)}$$

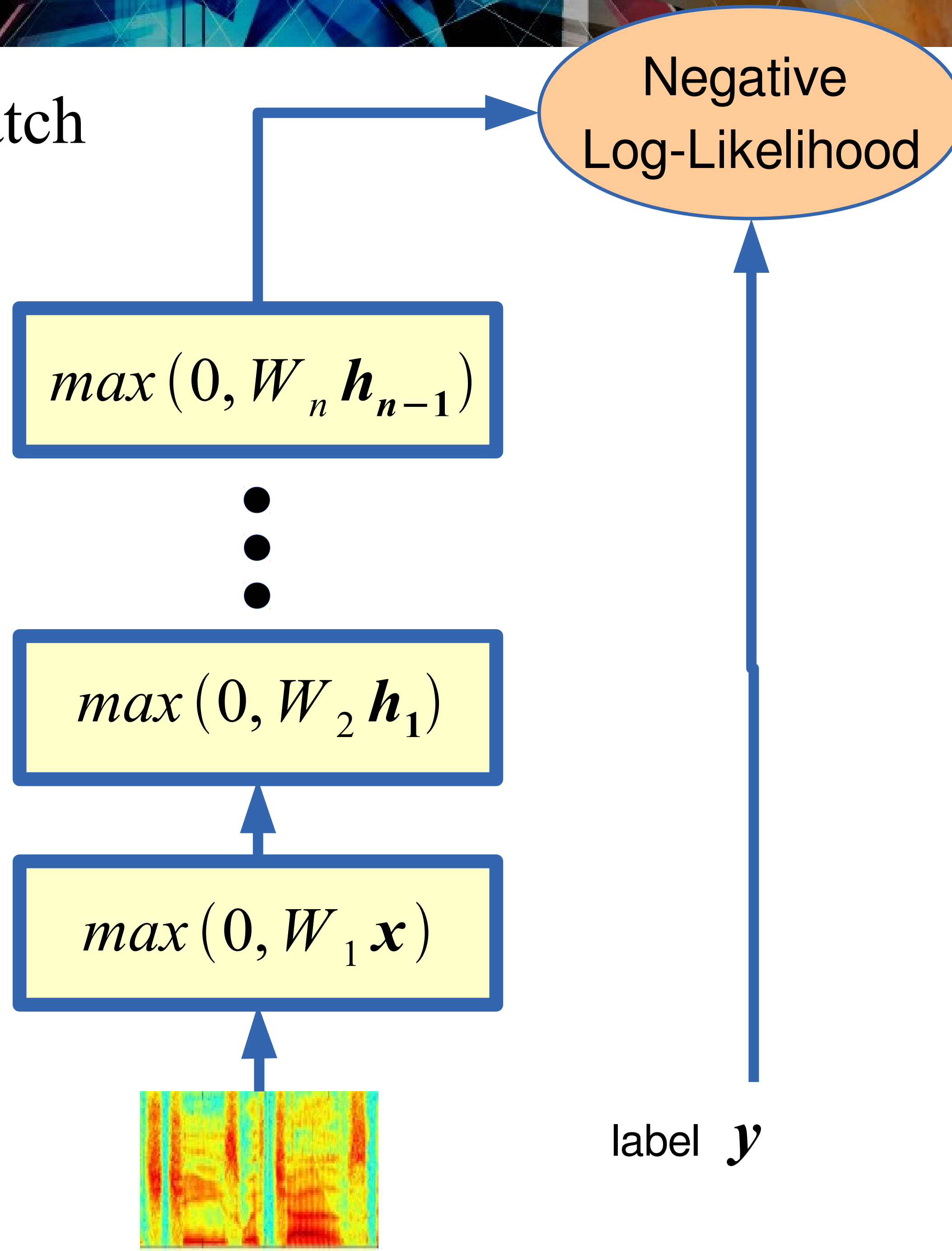
- Mini-batches of size 40
- Asynchronous SGD (using 100 copies of the network on a few hundred machines). This speeds up training at Google but it is not crucial.

Zeiler et al. “On rectified linear units for speech recognition” ICASSP 2013

Training

- Given an input mini-batch

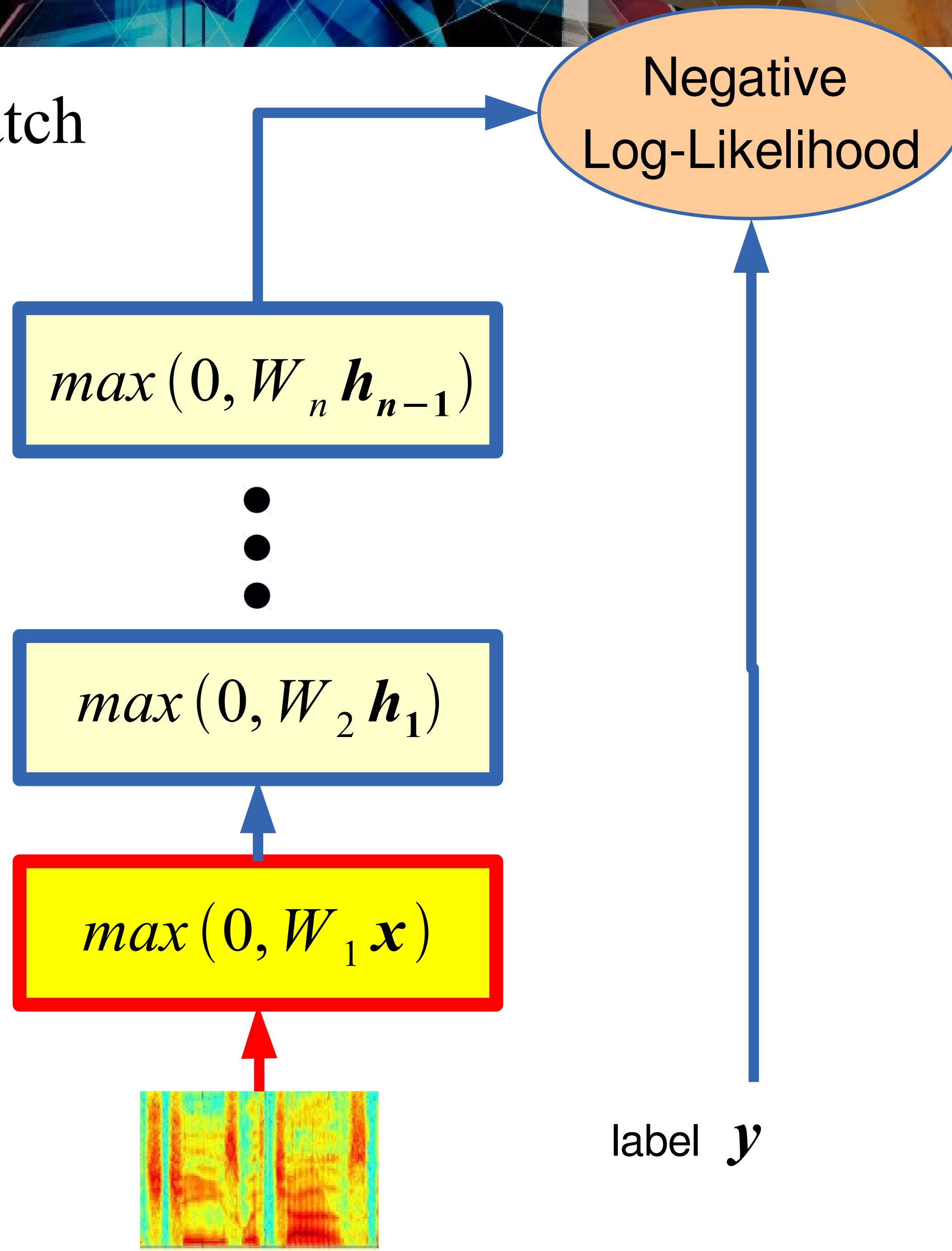
FPROP



Training

Y LeCun
MA Ranzato

- Given an input mini-batch



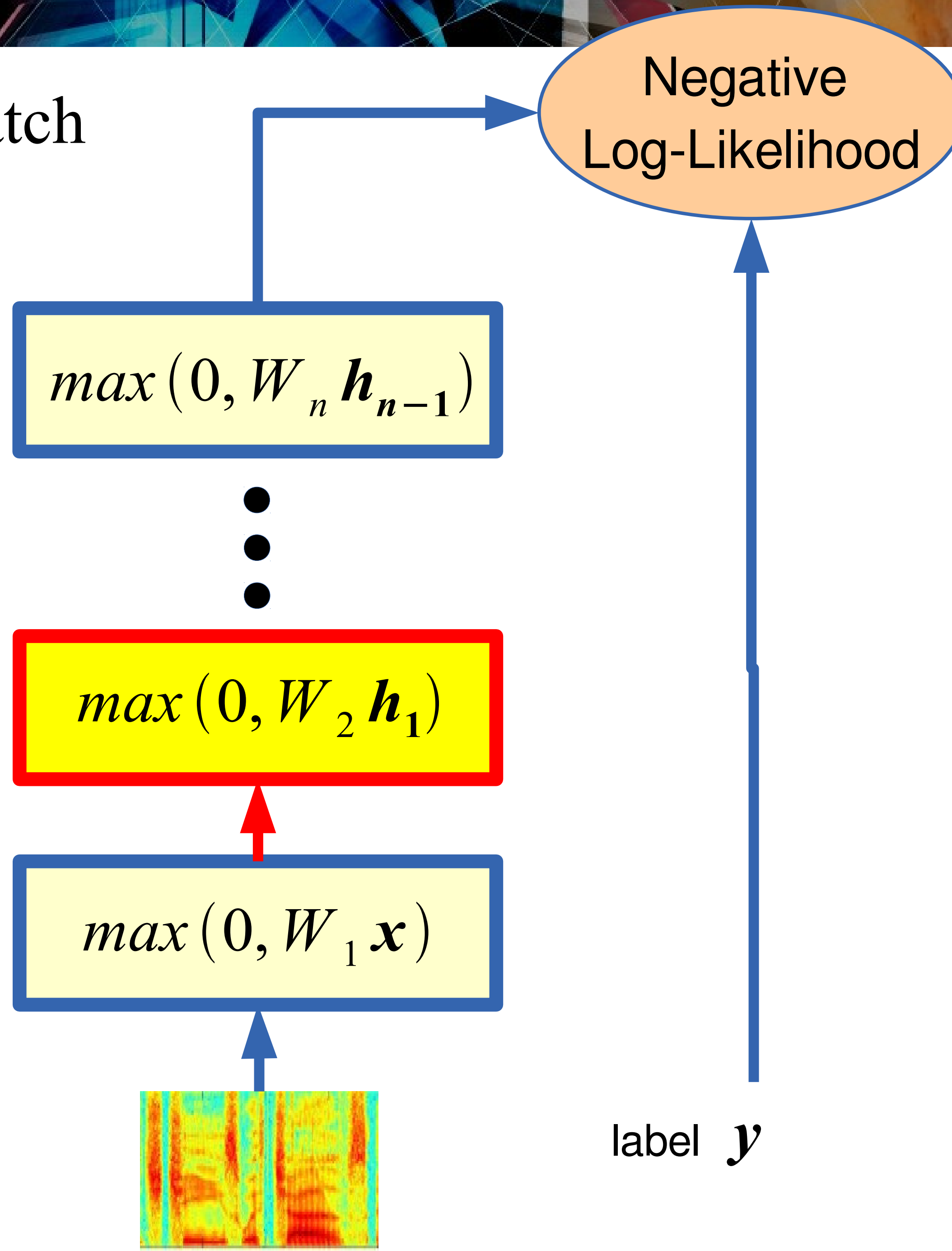
FPROP

$$h_2 = f(x; W_1)$$

Training

Y LeCun
MA Ranzato

- Given an input mini-batch



FPROP

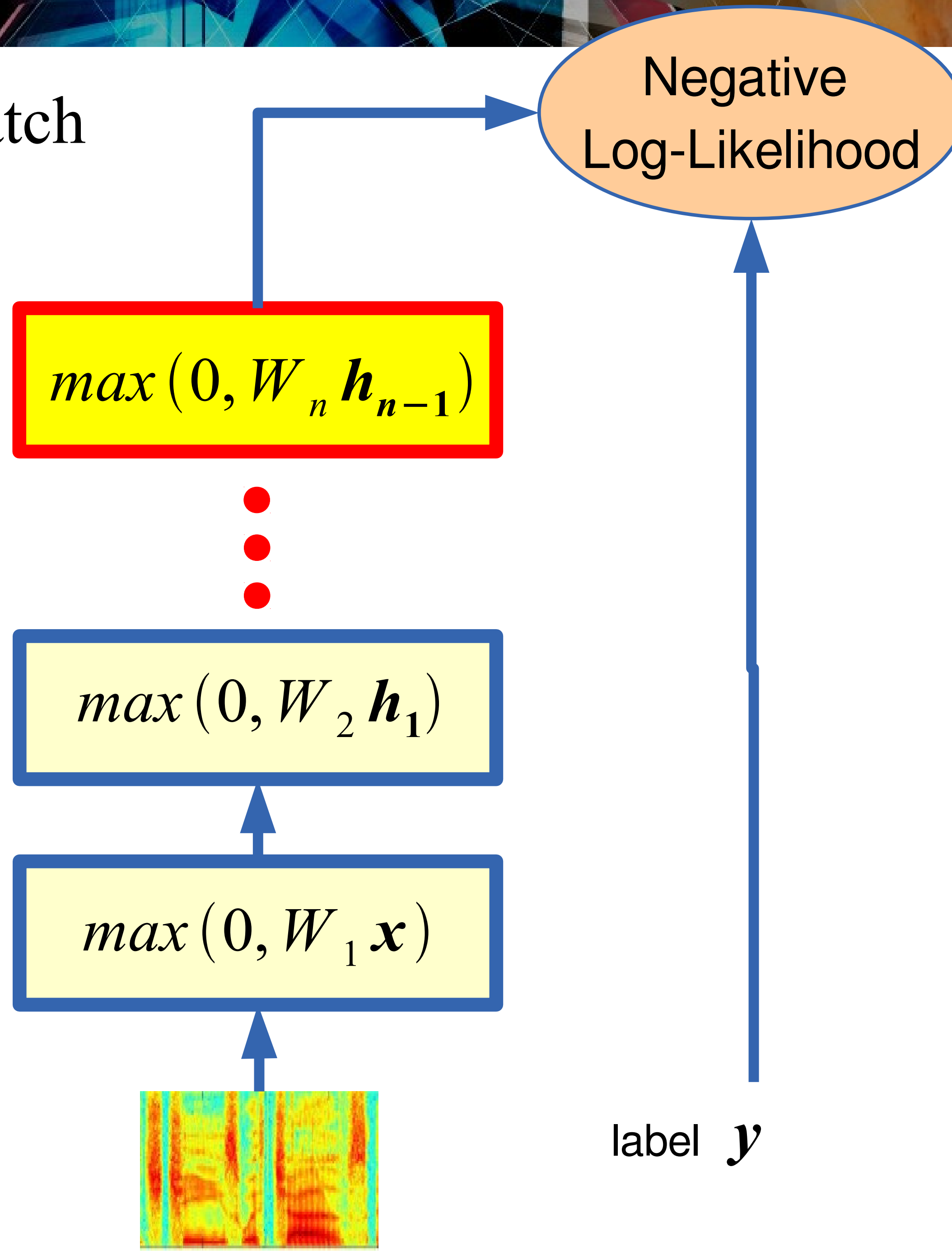
$$\boldsymbol{h}_2 = f(\boldsymbol{h}_1; W_2)$$

Training

- Given an input mini-batch

FPROP

$$\boldsymbol{h}_n = f(\boldsymbol{h}_{n-1})$$

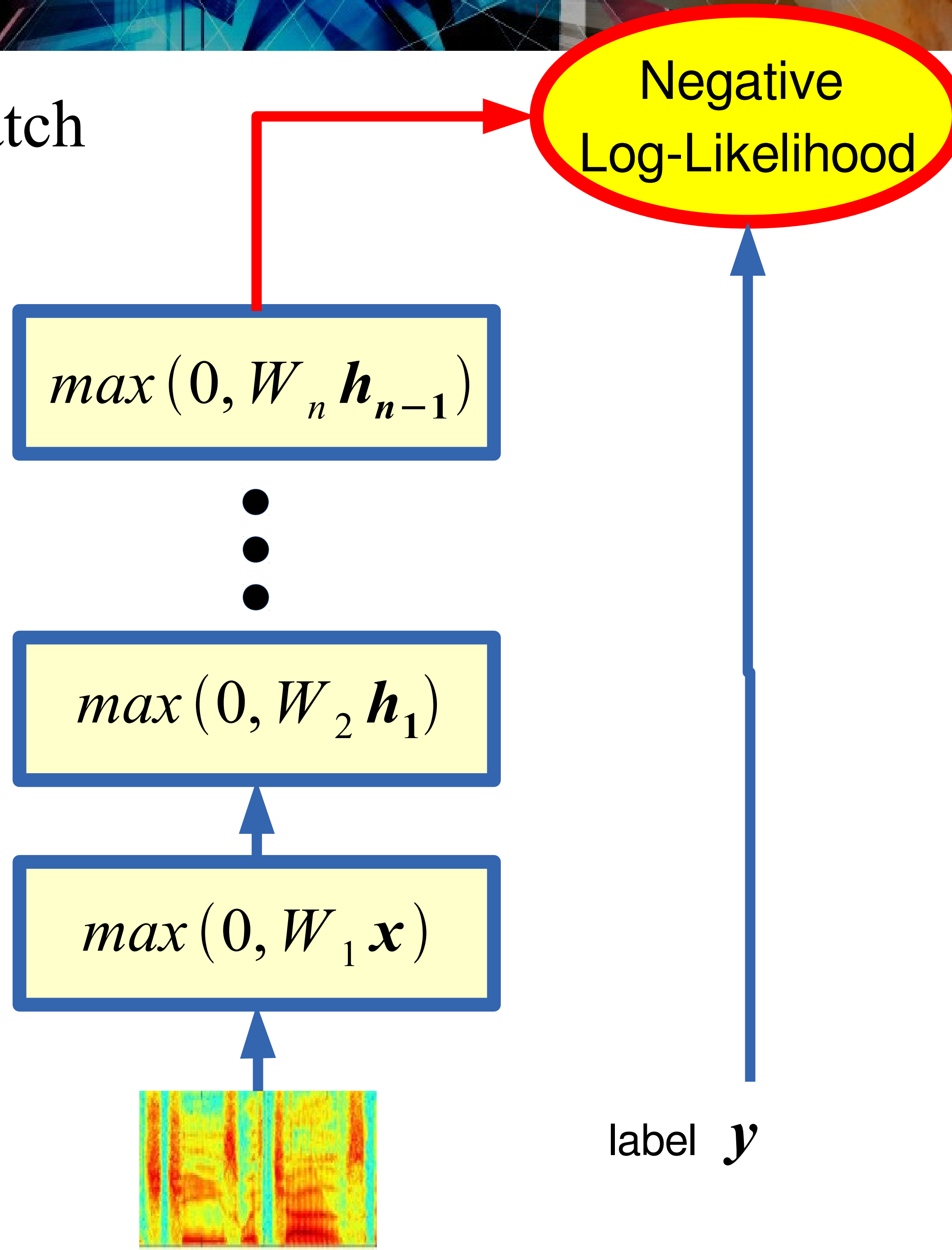


Training

Y LeCun
MA Ranzato

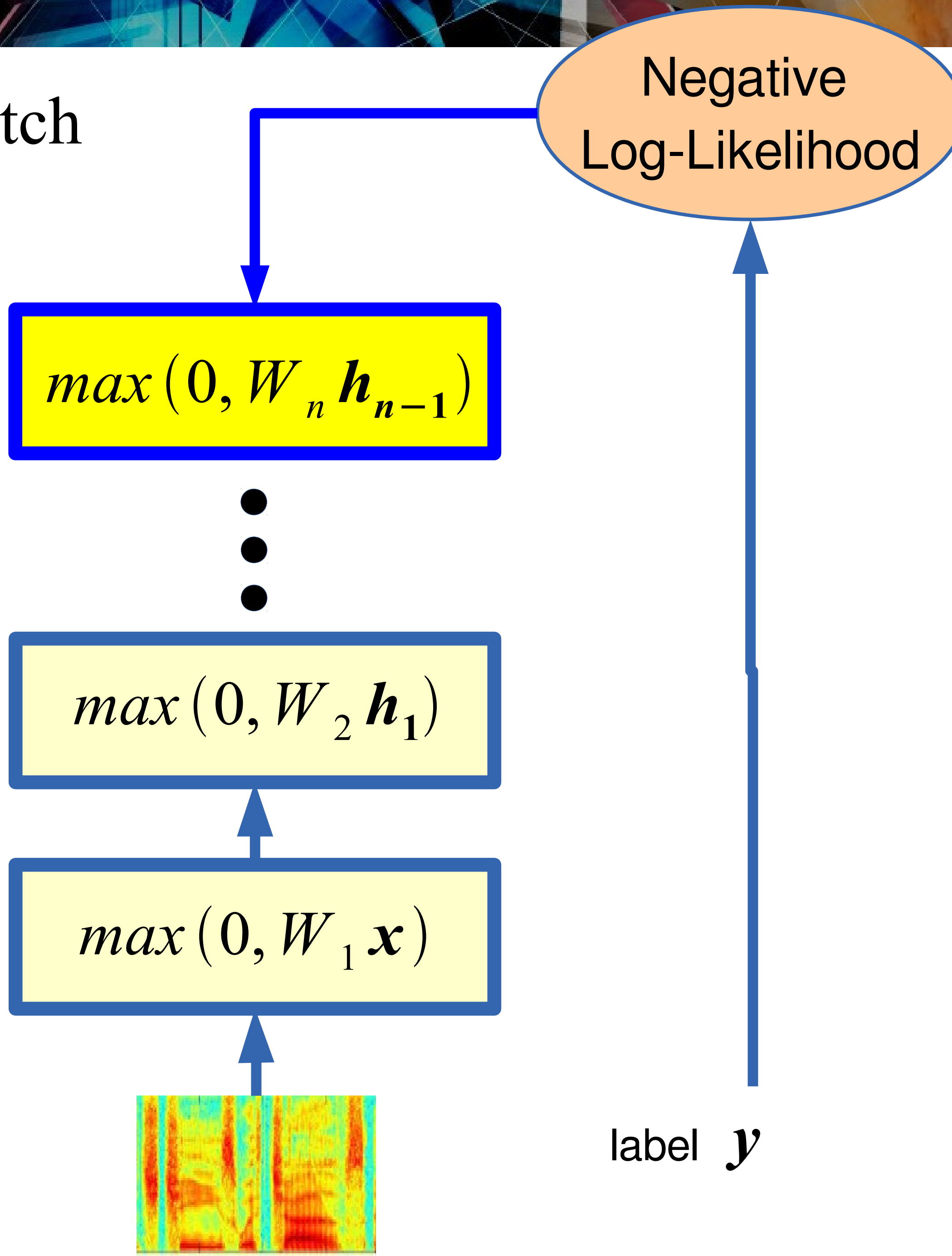
- Given an input mini-batch

FPROP



Training

- Given an input mini-batch



BPROP

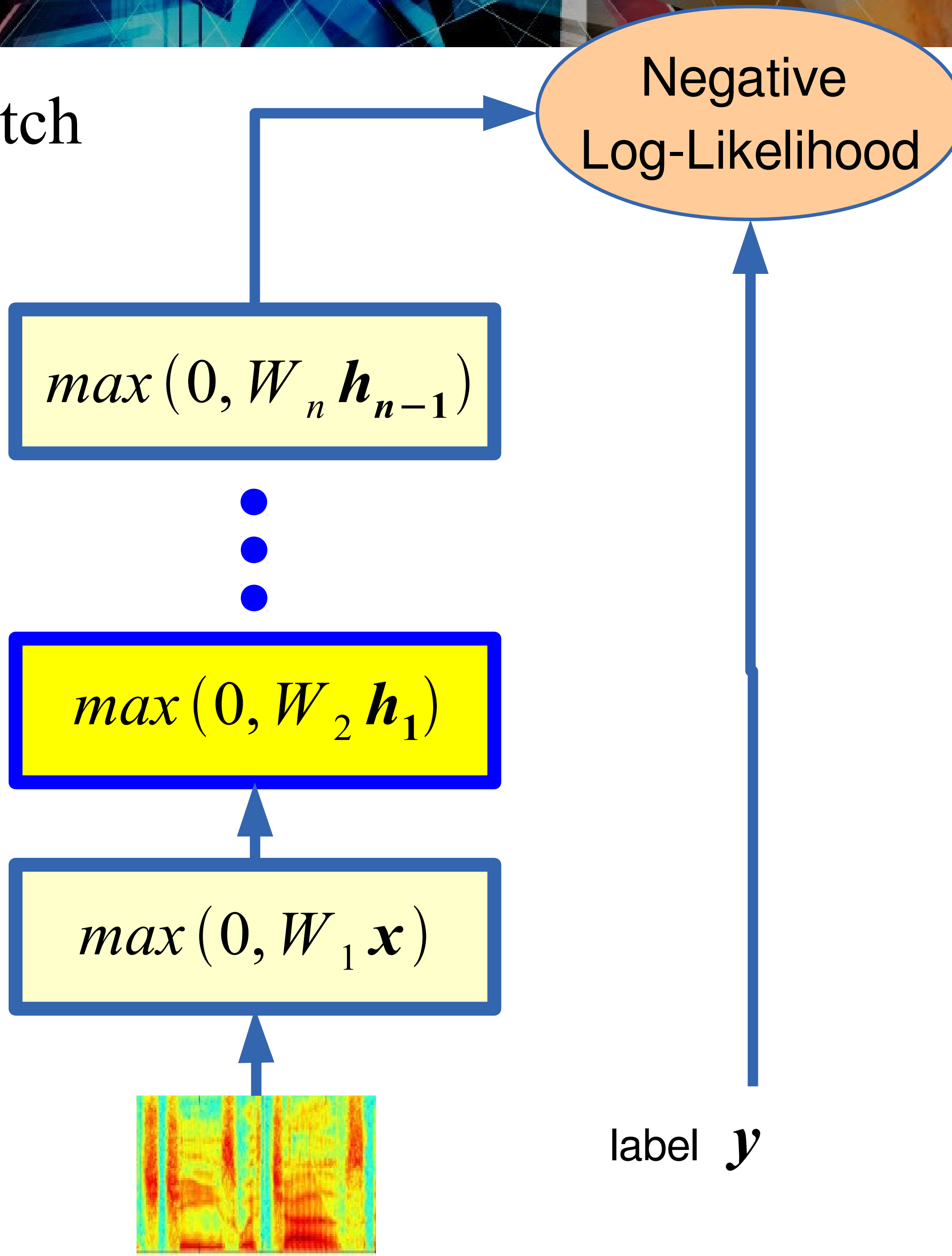
$$\frac{\partial L}{\partial \boldsymbol{h}_{n-1}} = \frac{\partial L}{\partial \boldsymbol{h}_n} \frac{\partial \boldsymbol{h}_n}{\partial \boldsymbol{h}_{n-1}}$$

$$\frac{\partial L}{\partial W_n} = \frac{\partial L}{\partial \boldsymbol{h}_n} \frac{\partial \boldsymbol{h}_n}{\partial W_n}$$

Training

Y LeCun
MA Ranzato

- Given an input mini-batch



BPROP

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

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \boldsymbol{h}_2} \frac{\partial \boldsymbol{h}_2}{\partial W_2}$$

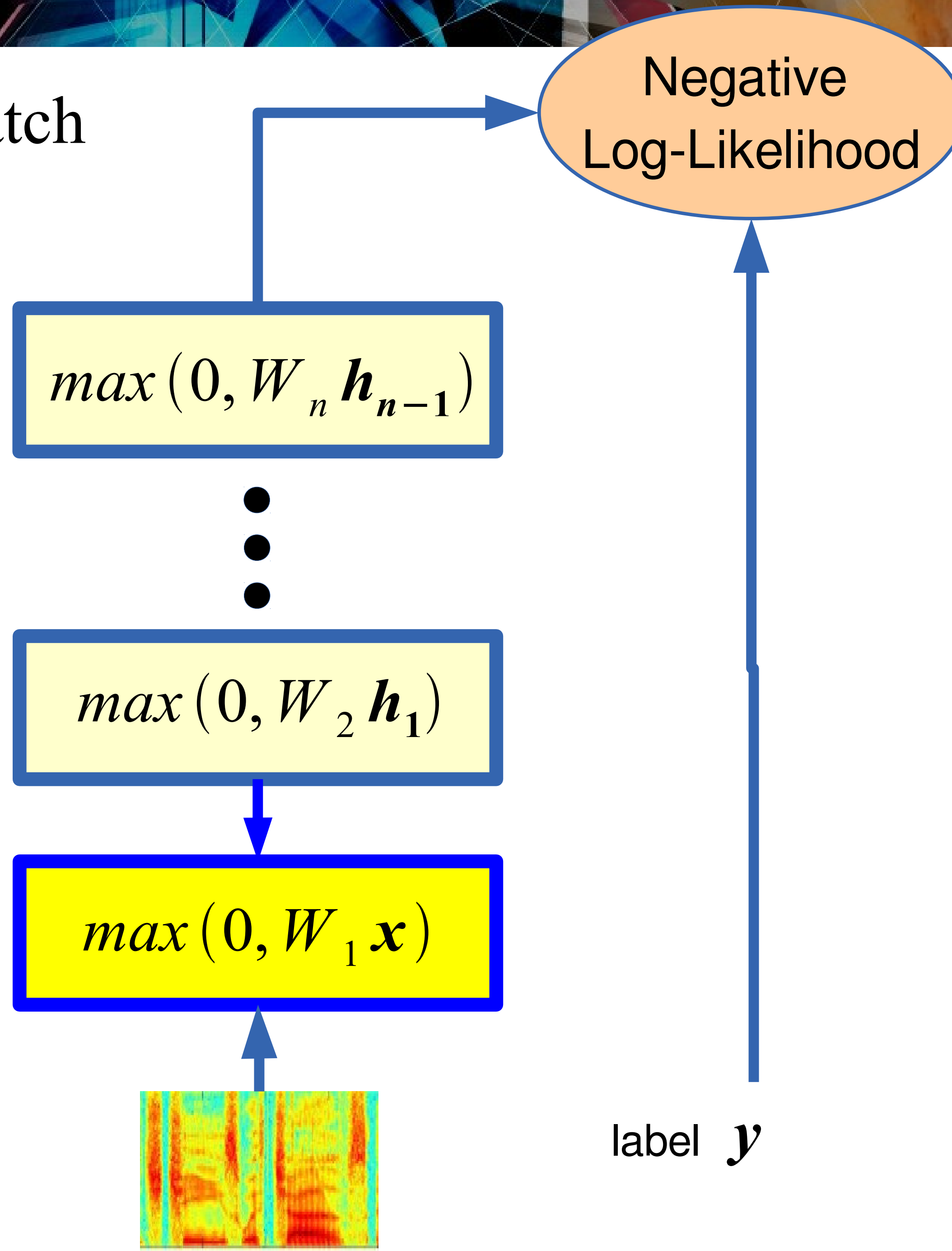
Training

Y LeCun
MA Ranzato

- Given an input mini-batch

BPROP

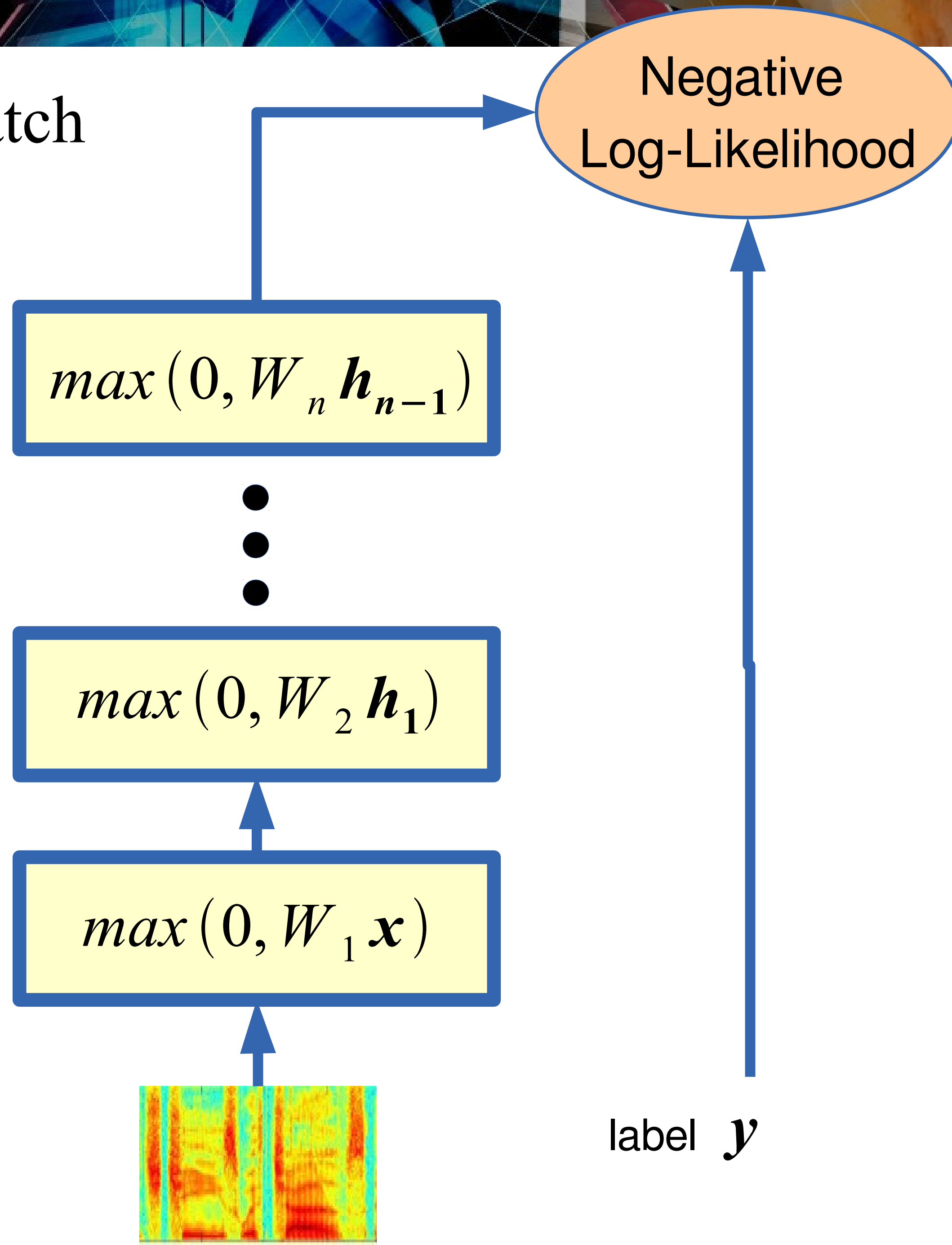
$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \boldsymbol{h}_1} \frac{\partial \boldsymbol{h}_1}{\partial W_1}$$



Training

Y LeCun
MA Ranzato

- Given an input mini-batch

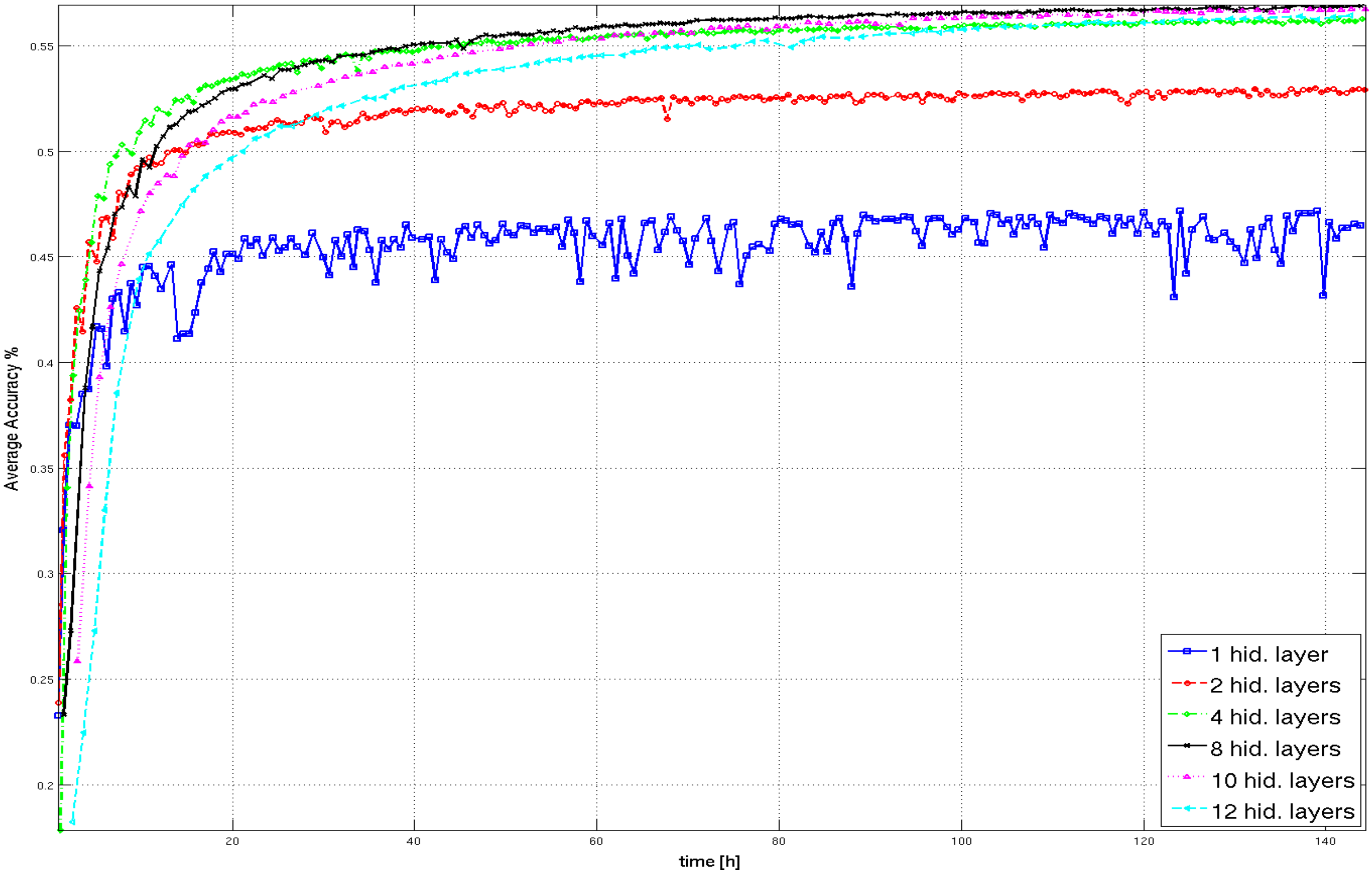


Parameter update

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

Training

Y LeCun
MA Ranzato



Word Error Rate

Y LeCun
MA Ranzato

Number of hidden layers	Word Error Rate %
1	16
2	12.8
4	11.4
8	10.9
12	11.1

GMM baseline: 15.4%

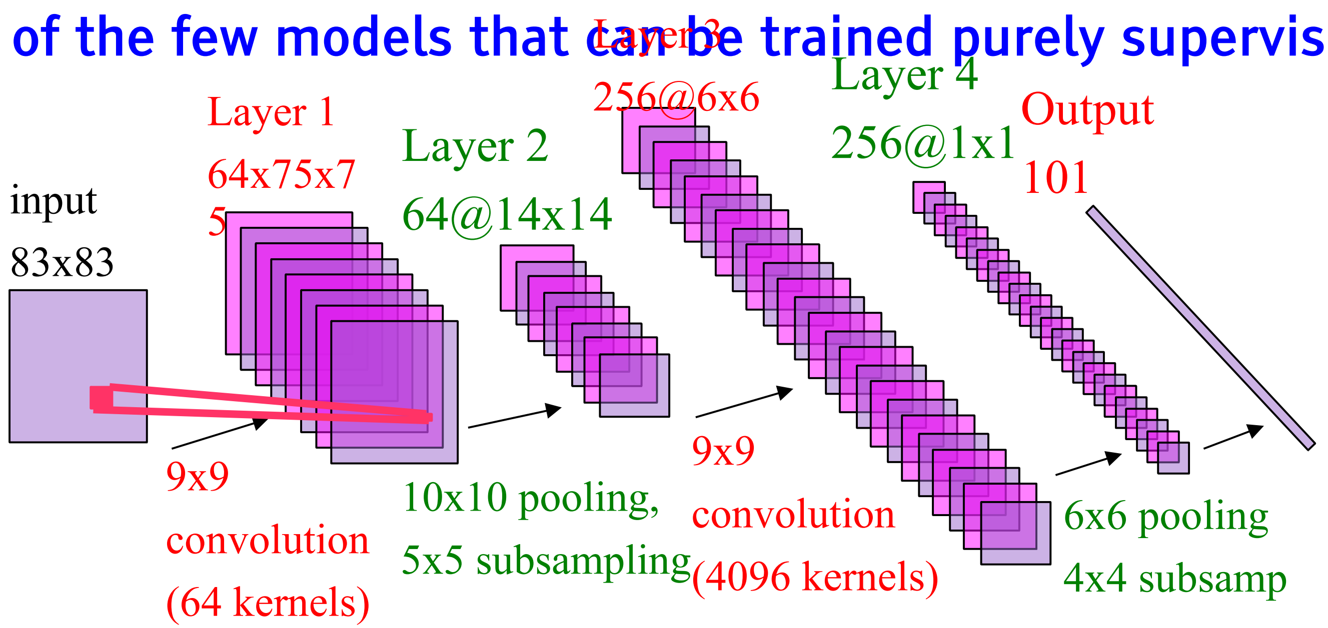
Zeiler et al. “On rectified linear units for speech recognition” ICASSP 2013

Convolutional Networks

Convolutional Nets

Y LeCun
MA Ranzato

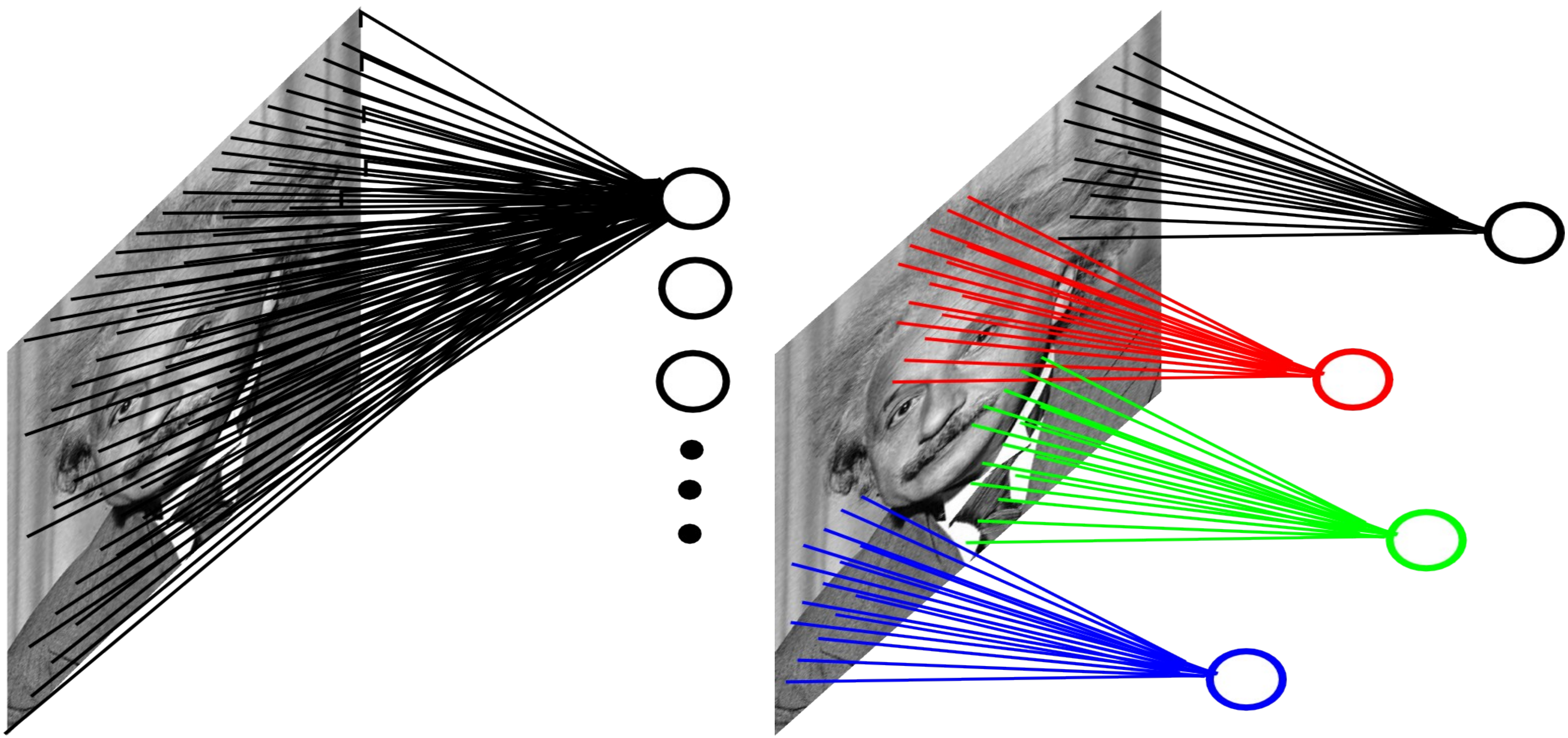
- **Are deployed in many practical applications**
  - Image recognition, speech recognition, Google's and Baidu's photo taggers
- **Have won several competitions**
  - ImageNet, Kaggle Facial Expression, Kaggle Multimodal Learning, German Traffic Signs, Connectomics, Handwriting....
- **Are applicable to array data where nearby values are correlated**
  - Images, sound, time-frequency representations, video, volumetric images, RGB-Depth images,.....
- **One of the few models that can be trained purely supervised**



Fully-connected neural net in high dimension

Y LeCun
MA Ranzato

- **Example: 200x200 image**
  - Fully-connected, 400,000 hidden units = 16 billion parameters
  - Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
  - Local connections capture local dependencies



Shared Weights & Convolutions: Exploiting Stationarity

Y LeCun
MA Ranzato

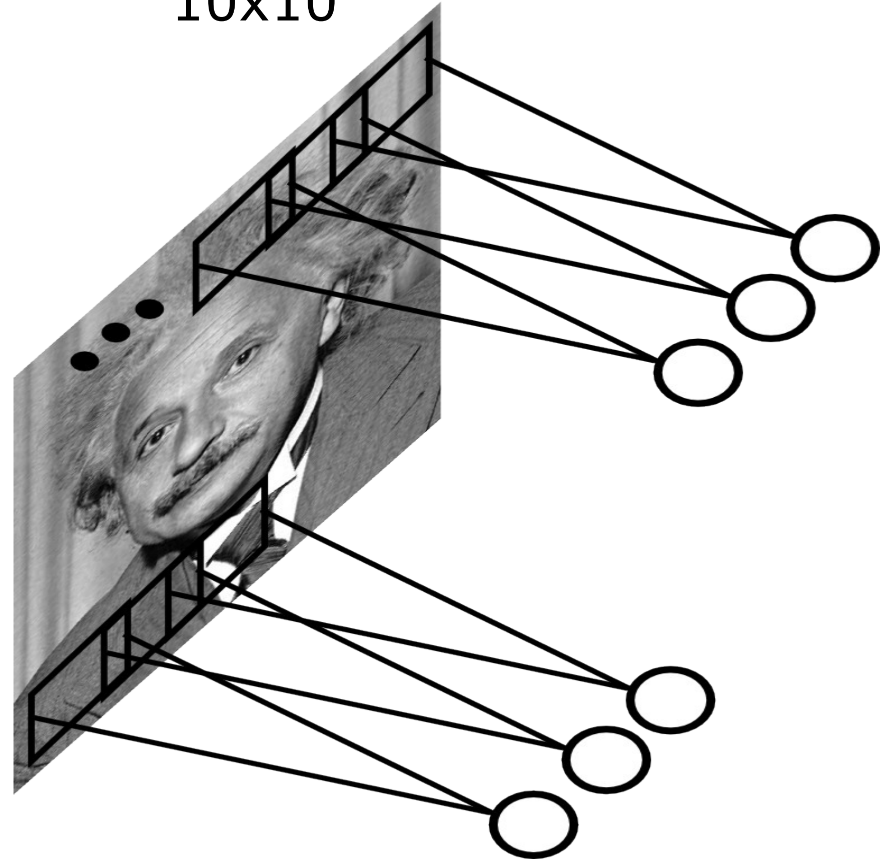
- Features that are useful on one part of the image and probably useful elsewhere.
- All units share the same set of weights
- Shift equivariant processing:
  - When the input shifts, the output also shifts but stays otherwise unchanged.
- Convolution
  - with a learned kernel (or filter)
  - Non-linearity: ReLU (rectified linear)

$$A_{ij} = \sum_{kl} W_{kl} X_{i+j.k+l}$$

- The filtered "image" Z is called a feature map

$$Z_{ij} = max(0, A_{ij})$$

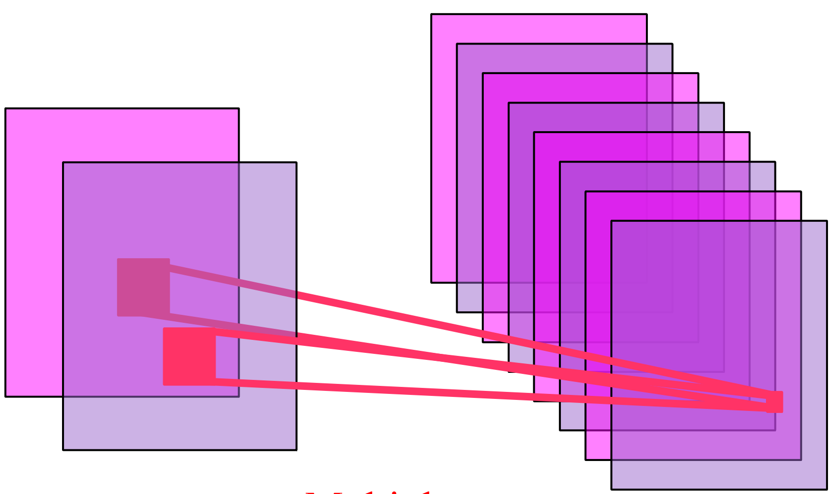
- Example: 200x200 image
  - 400,000 hidden units with 10x10 fields = 1000 params
  - 10 feature maps of size 200x200, 10 filters of size 10x10



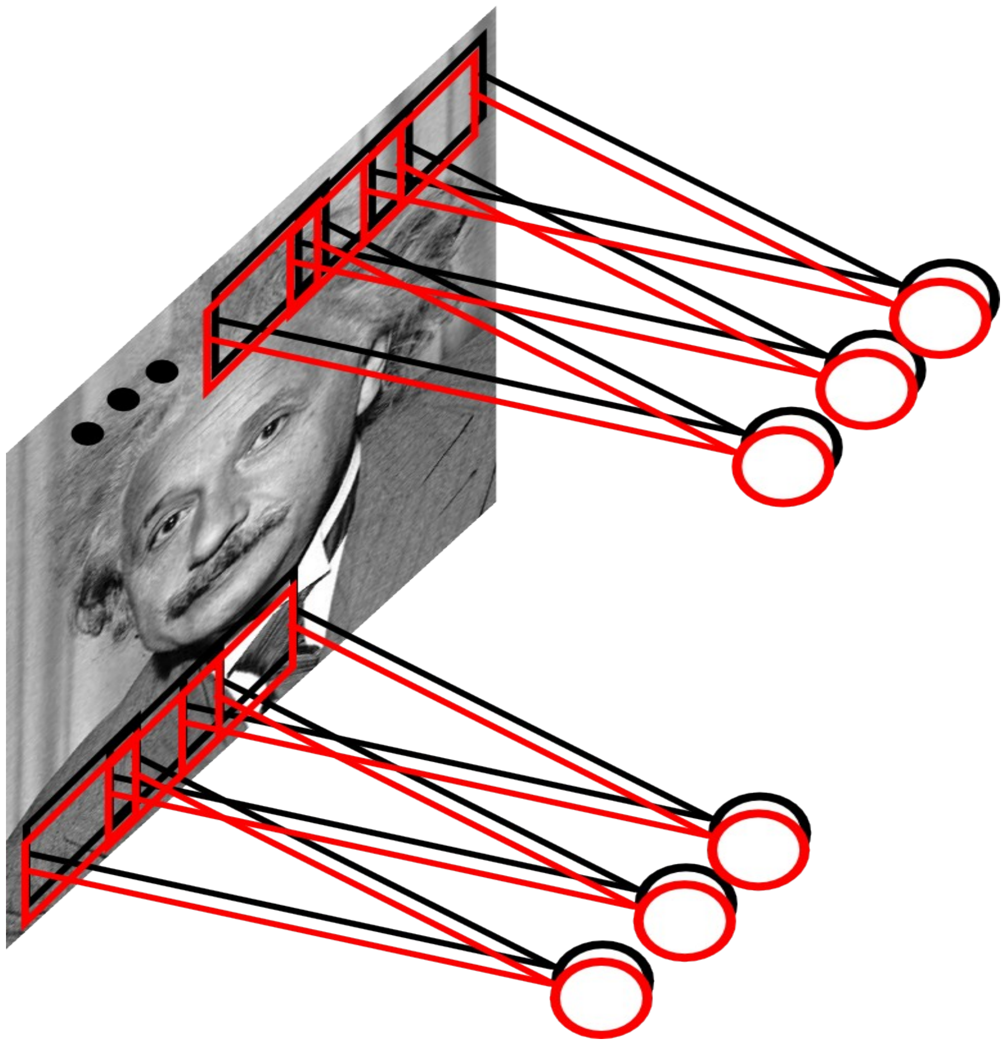
Multiple Convolutions with Different Kernels

Y LeCun
MA Ranzato

- Detects multiple motifs at each location
- The collection of units looking at the same patch is akin to a feature vector for that patch.
- The result is a 3D array, where each slice is a feature map.



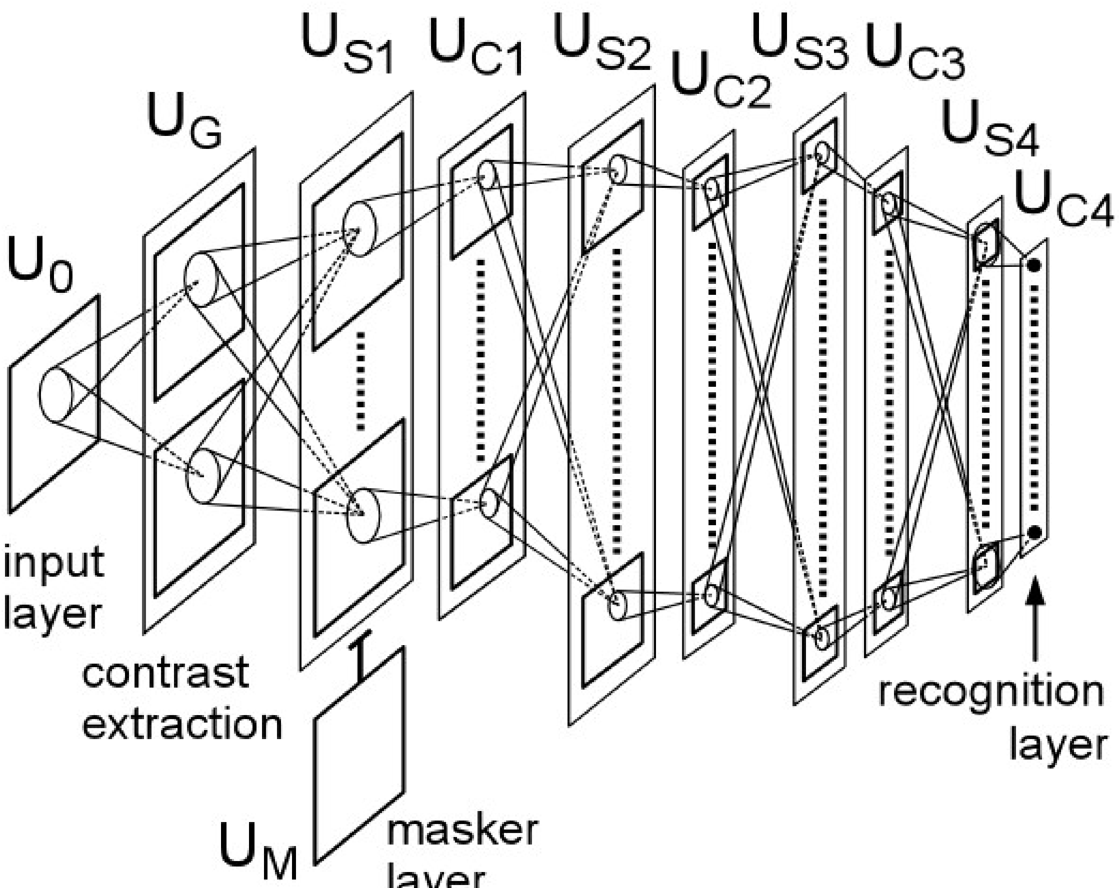
Multiple convolutions

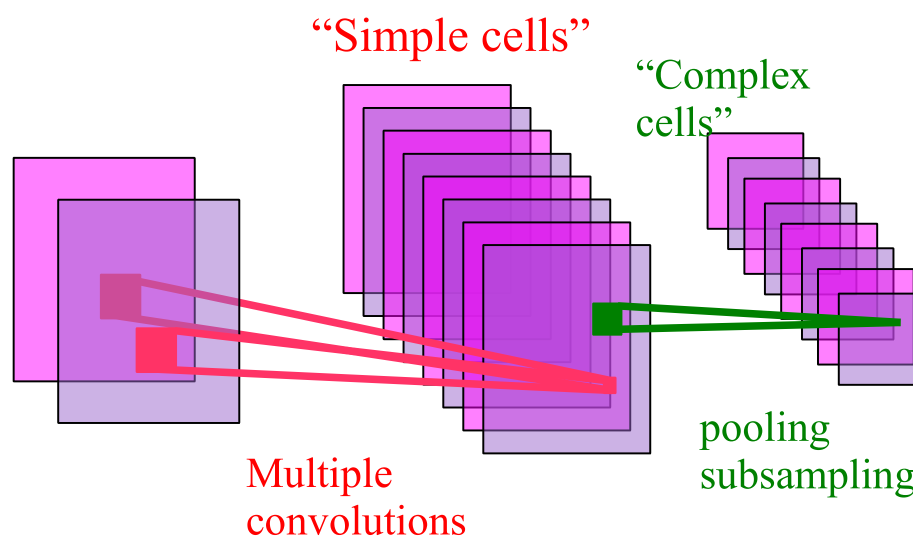


Early Hierarchical Feature Models for Vision

Y LeCun
MA Ranzato

- [Hubel & Wiesel 1962]:
  - simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.

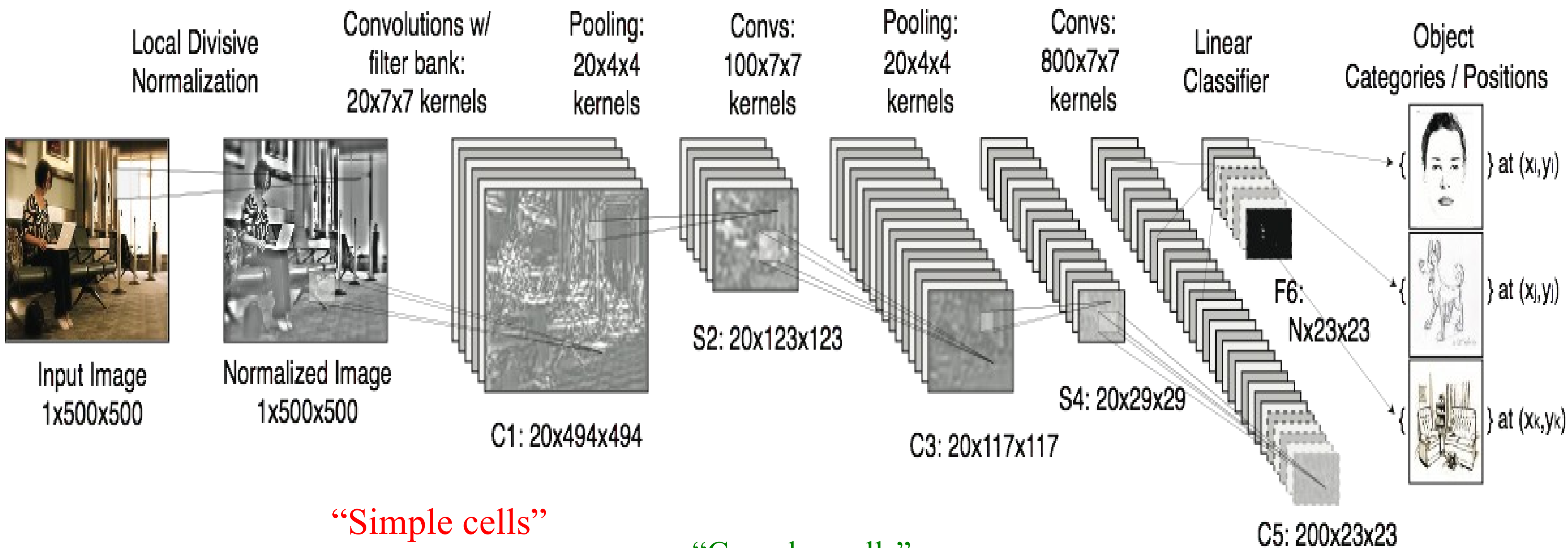


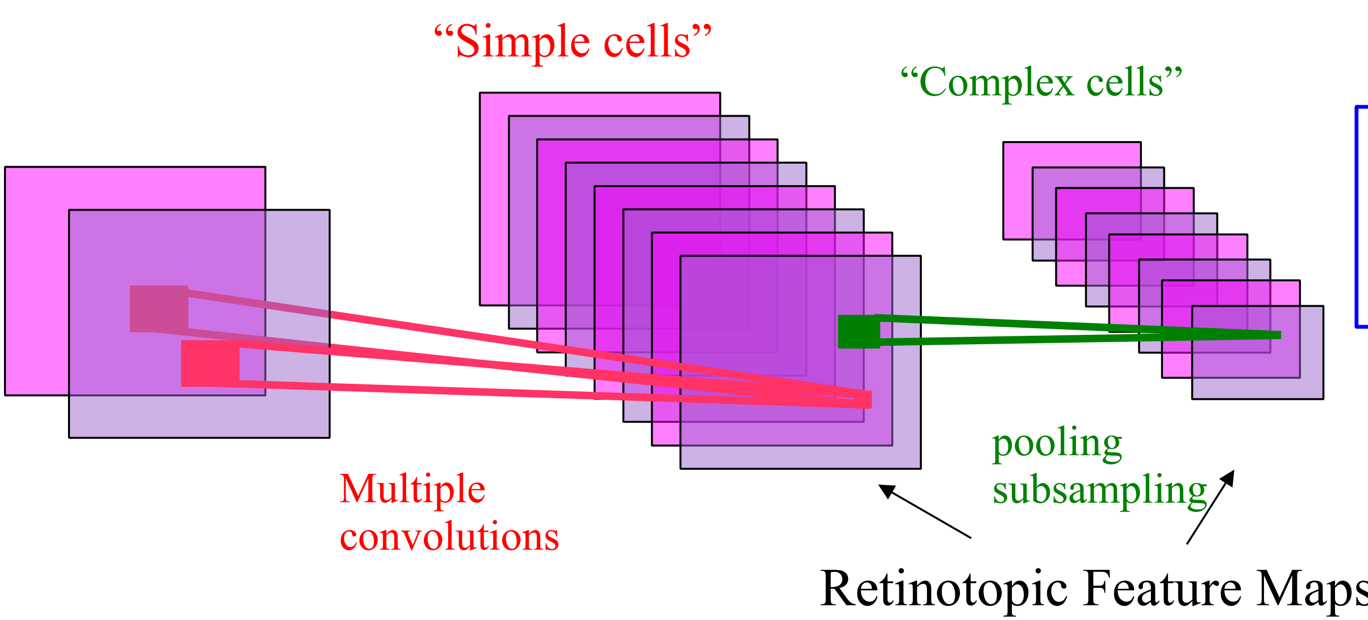


Cognitron & Neocognitron [Fukushima 1974-1982]

The Convolutional Net Model
(Multistage Hubel-Wiesel system)

Y LeCun
MA Ranzato



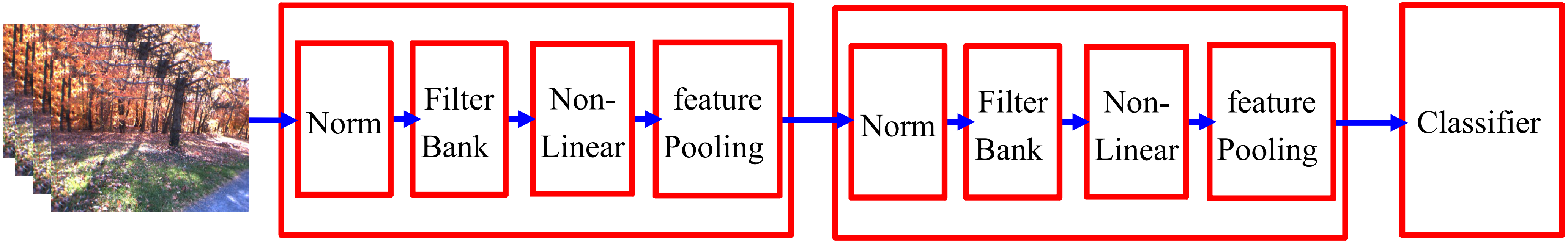


- **Training is supervised**
- **With stochastic gradient descent**

[LeCun et al. 89]
[LeCun et al. 98]

Feature Transform:
Normalization → Filter Bank → Non-Linearity → Pooling

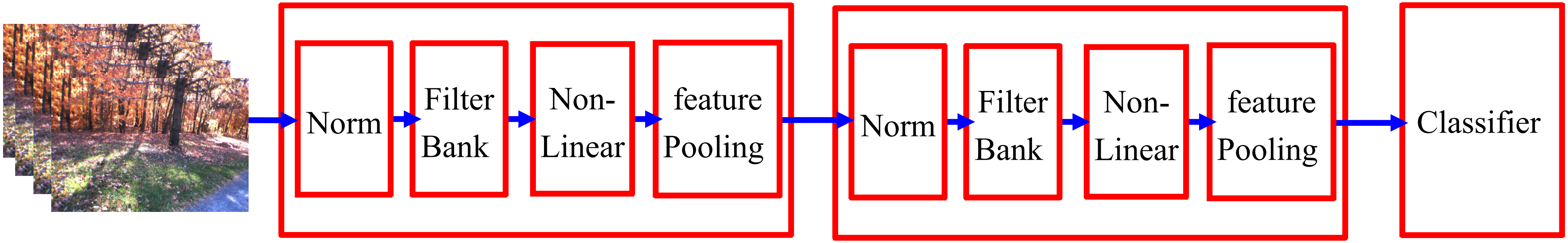
Y LeCun
MA Ranzato



- **Stacking multiple stages of**
  - [Normalization → Filter Bank → Non-Linearity → Pooling].
- **Normalization: variations on whitening**
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank: dimension expansion, projection on overcomplete basis**
- **Non-Linearity: sparsification, saturation, lateral inhibition....**
  - Rectification, Component-wise shrinkage, tanh, winner-takes-all
- **Pooling: aggregation over space or feature type, subsampling**
  - $X_i; \quad L_p: \sqrt[p]{X_i^p}; \quad PROB: \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$

Feature Transform:
Normalization → Filter Bank → Non-Linearity → Pooling

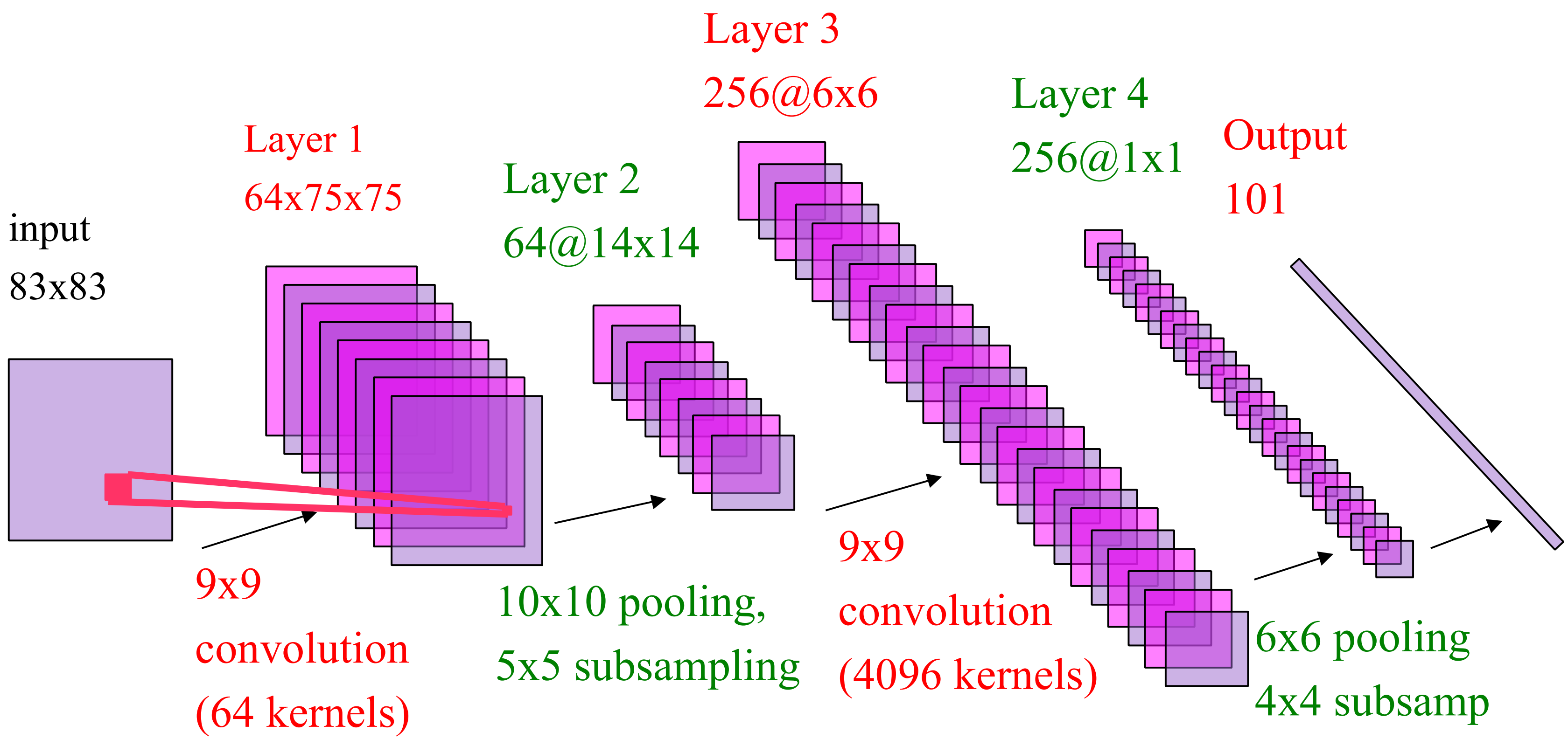
Y LeCun
MA Ranzato



- Filter Bank → Non-Linearity = Non-linear embedding in high dimension
- Feature Pooling = contraction, dimensionality reduction, smoothing
- Learning the filter banks at every stage
- Creating a hierarchy of features
- Basic elements are inspired by models of the visual (and auditory) cortex
  - Simple Cell + Complex Cell model of [Hubel and Wiesel 1962]
  - Many "traditional" feature extraction methods are based on this
  - SIFT, GIST, HoG, SURF...
- [Fukushima 1974-1982], [LeCun 1988-now],
  - since the mid 2000: Hinton, Seung, Poggio, Ng,....

Convolutional Network (ConvNet)

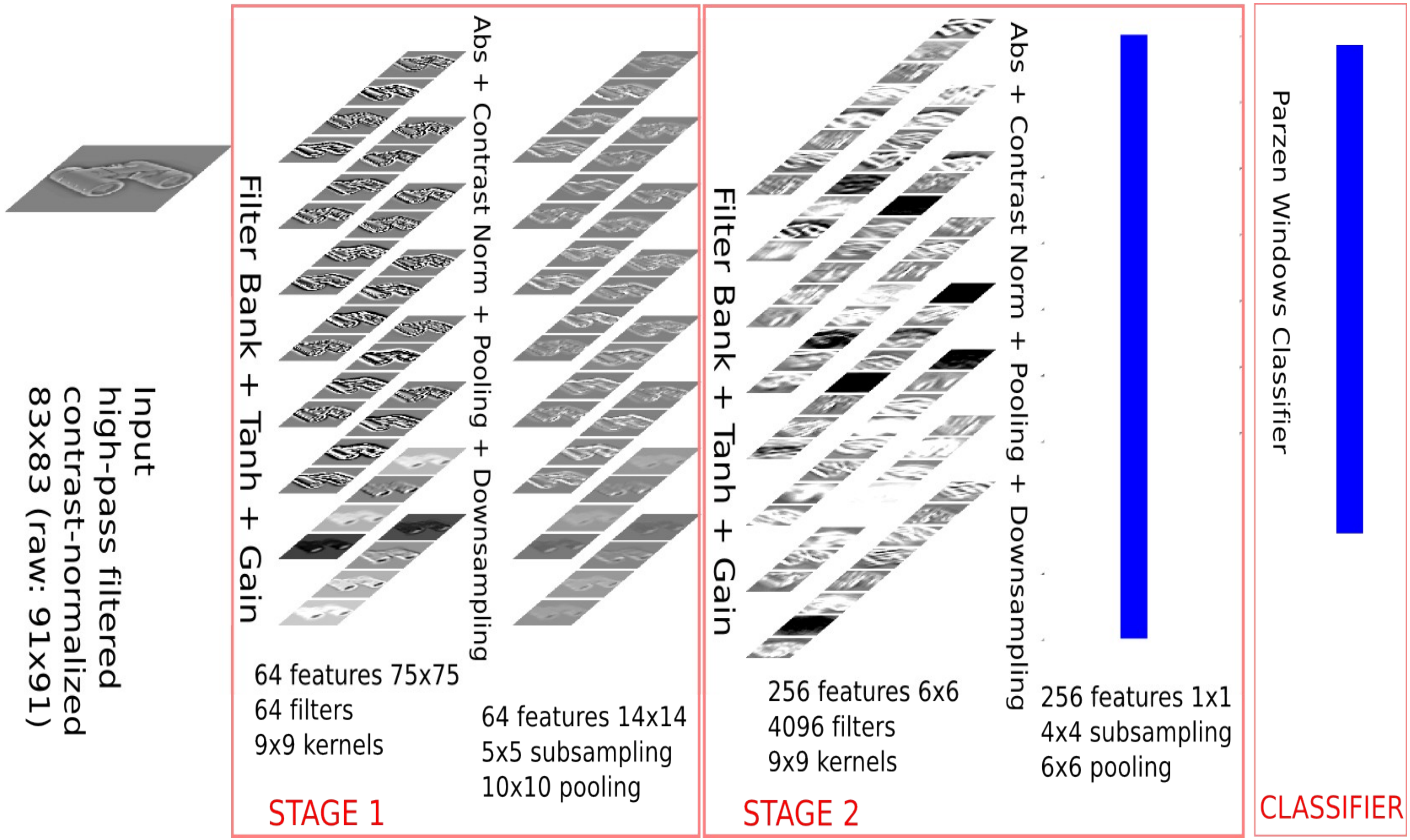
Y LeCun
MA Ranzato



- Non-Linearity: half-wave rectification, shrinkage function, sigmoid
- Pooling: average, L1, L2, max
- Training: Supervised (1988-2006), Unsupervised+Supervised (2006-now)

Convolutional Network Architecture

Y LeCun
MA Ranzato

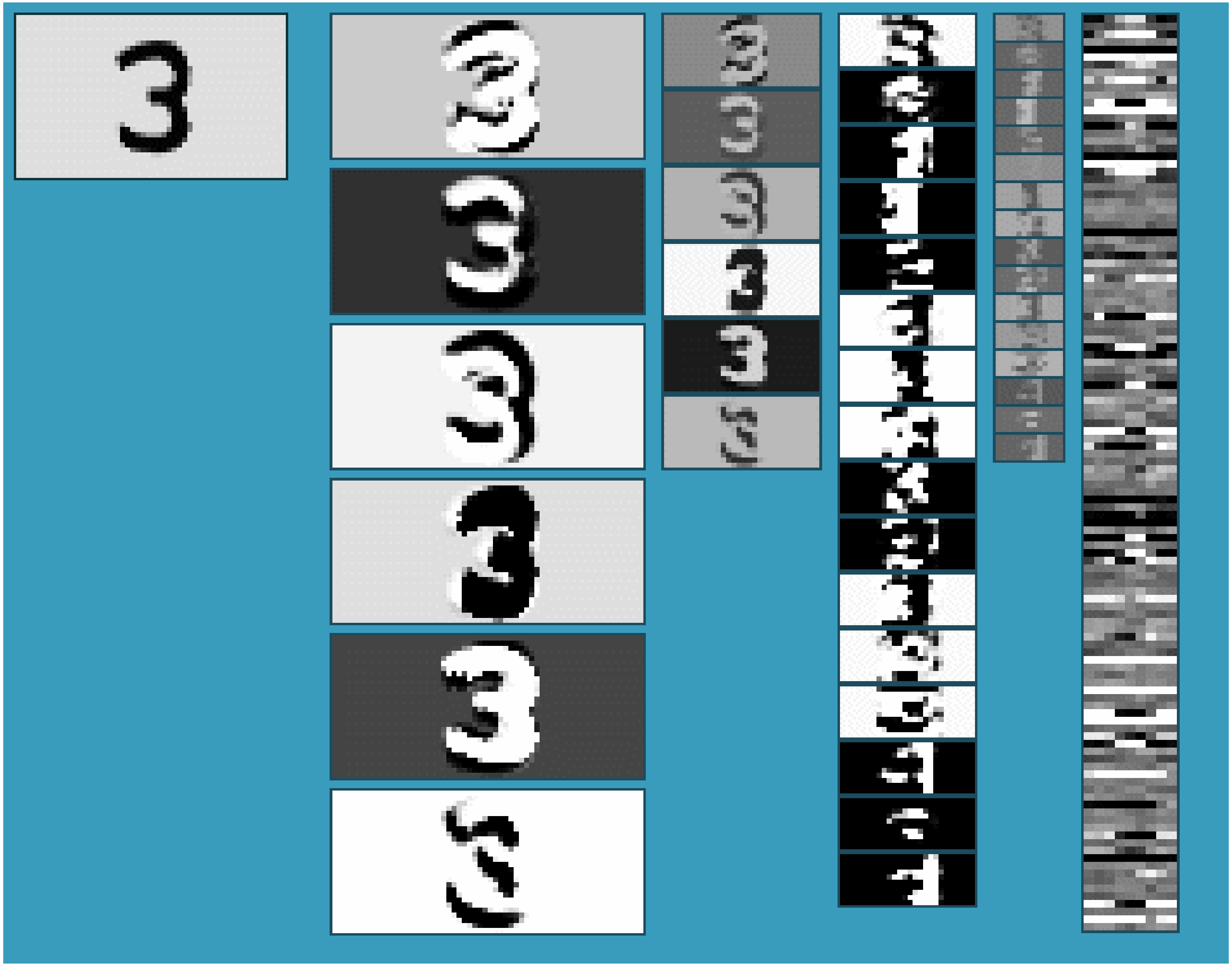


Convolutional Network (vintage 1990)

Y LeCun
MA Ranzato

- filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh

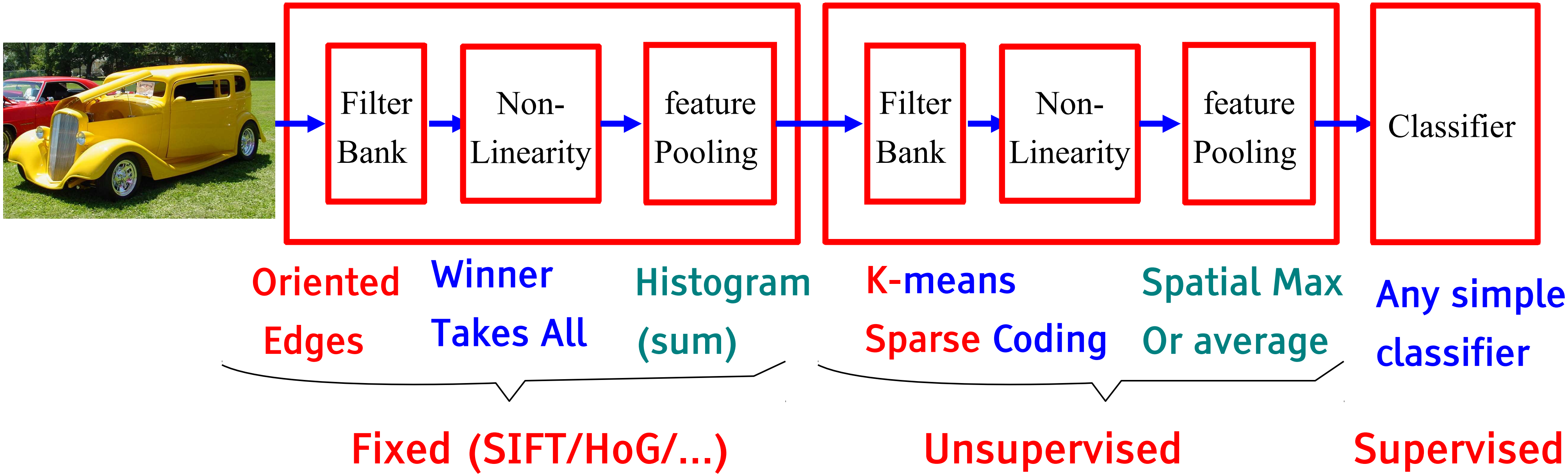
Curved manifold



Flatter manifold

“Mainstream” object recognition pipeline 2006-2012: somewhat similar to ConvNets

Y LeCun
MA Ranzato



- **Fixed Features + unsupervised mid-level features + simple classifier**
  - SIFT + Vector Quantization + Pyramid pooling + SVM
    - [Lazebnik et al. CVPR 2006]
  - SIFT + Local Sparse Coding Macrofeatures + Pyramid pooling + SVM
    - [Boureau et al. ICCV 2011]
  - SIFT + Fisher Vectors + Deformable Parts Pooling + SVM
    - [Perronin et al. 2012]

Tasks for Which Deep Convolutional Nets are the Best

Y LeCun
MA Ranzato

- Handwriting recognition MNIST (many), Arabic HWX (IDSIA)
- OCR in the Wild [2011]: StreetView House Numbers (NYU and others)
- Traffic sign recognition [2011] GTSRB competition (IDSIA, NYU)
- Pedestrian Detection [2013]: INRIA datasets and others (NYU)
- Volumetric brain image segmentation [2009] connectomics (IDSIA, MIT)
- Human Action Recognition [2011] Hollywood II dataset (Stanford)
- Object Recognition [2012] ImageNet competition
- Scene Parsing [2012] Stanford bgd, SiftFlow, Barcelona (NYU)
- Scene parsing from depth images [2013] NYU RGB-D dataset (NYU)
- Speech Recognition [2012] Acoustic modeling (IBM and Google)
- Breast cancer cell mitosis detection [2011] MITOS (IDSIA)

- The list of perceptual tasks for which ConvNets hold the record is growing.
- Most of these tasks (but not all) use purely supervised convnets.

Ideas from Neuroscience and Psychophysics

Y LeCun
MA Ranzato

- The whole architecture: simple cells and complex cells
- Local receptive fields
- Self-similar receptive fields over the visual field (convolutions)
- Pooling (complex cells)
- Non-Linearity: Rectified Linear Units (ReLU)
- LGN-like band-pass filtering and contrast normalization in the input
- Divisive contrast normalization (from Heeger, Simoncelli....)
  - Lateral inhibition
- Sparse/Overcomplete representations (Olshausen-Field....)
- Inference of sparse representations with lateral inhibition
- Sub-sampling ratios in the visual cortex
  - between 2 and 3 between V1-V2-V4
- Crowding and visual metamers give cues on the size of the pooling areas

Simple ConvNet Applications with State-of-the-Art Performance

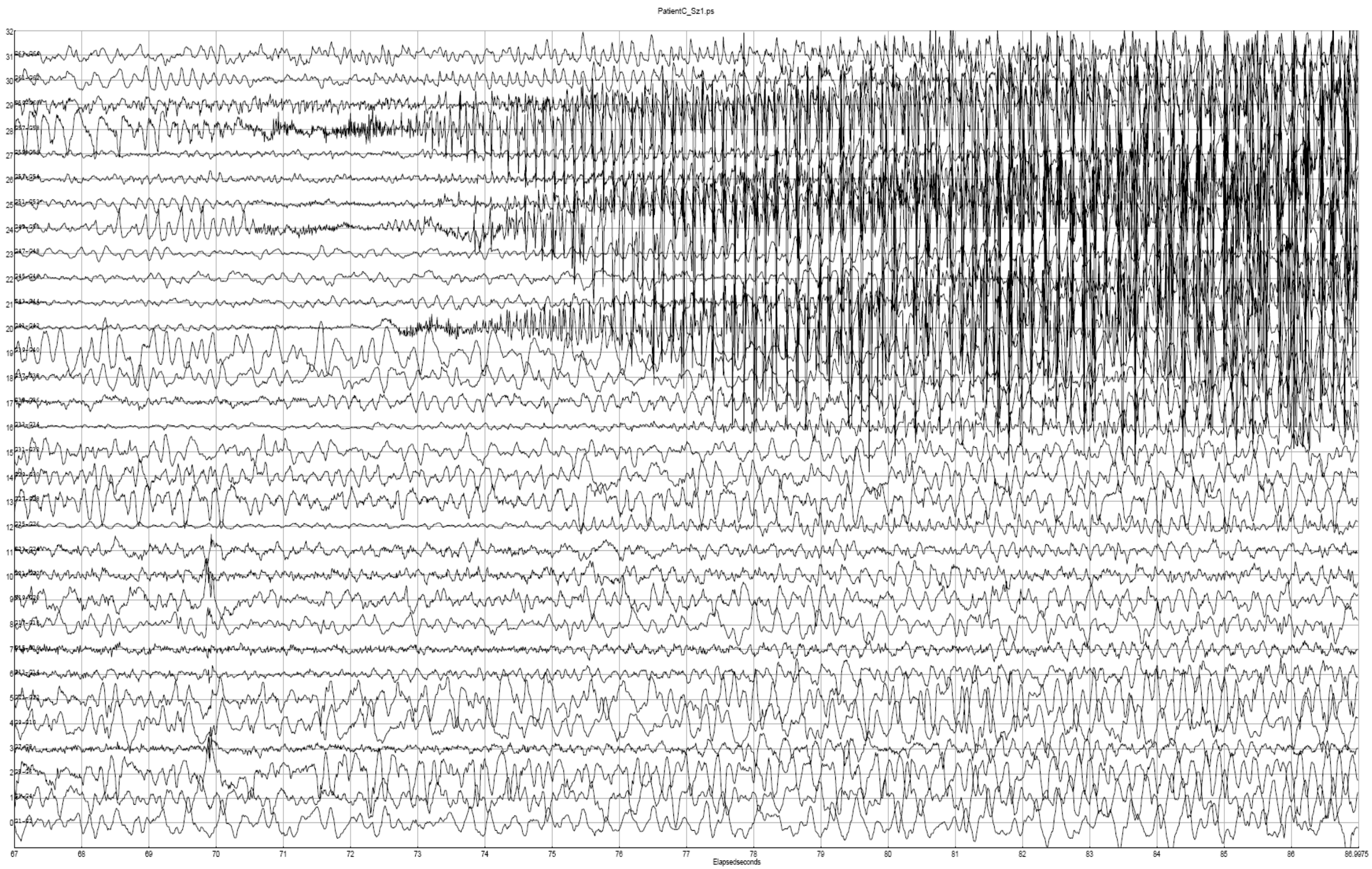
- **Traffic Sign Recognition (GTSRB)**
  - German Traffic Sign Reco Bench
  - 99.2% accuracy

- **House Number Recognition (Google)**
  - Street View House Numbers
  - 94.3 % accuracy

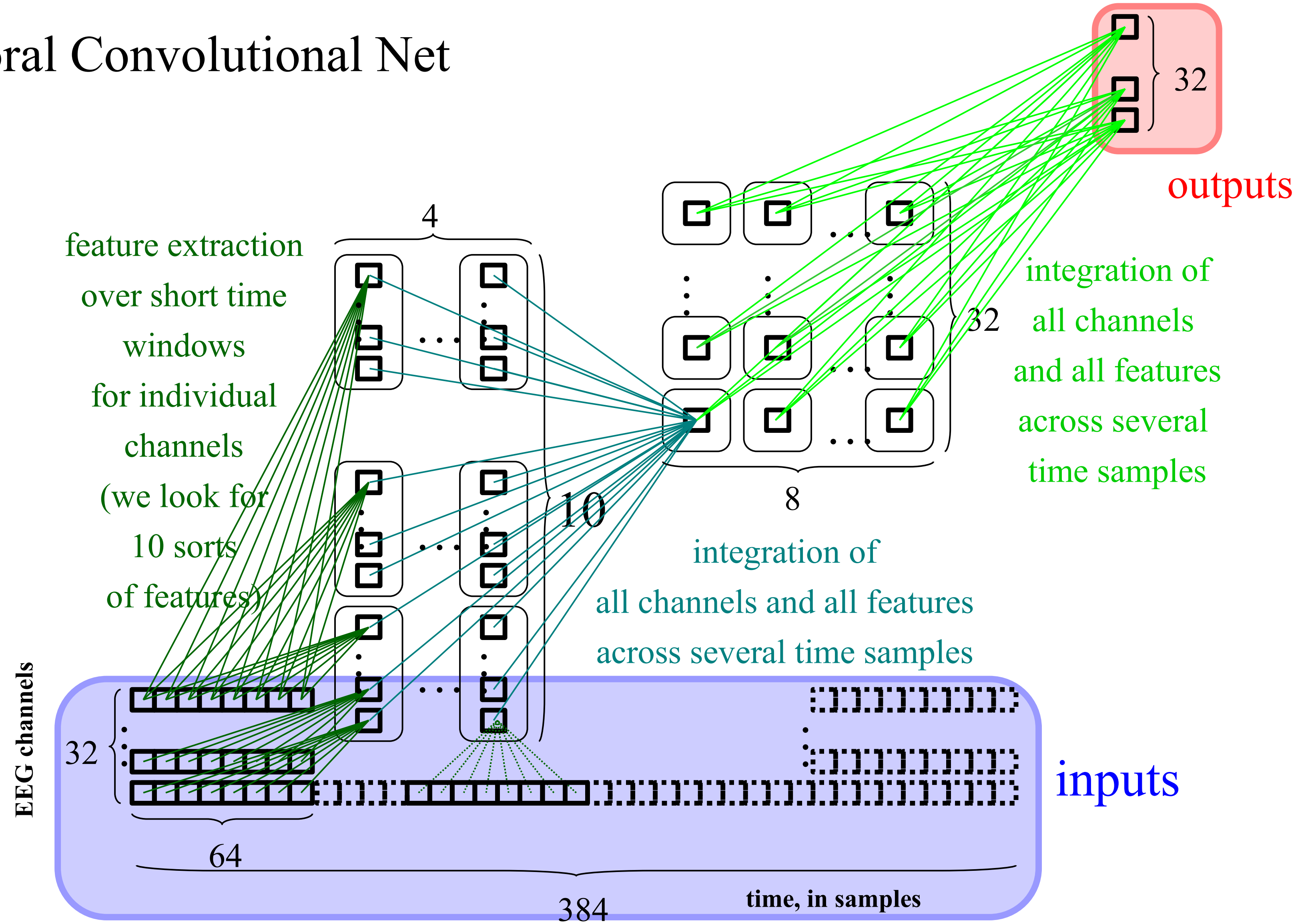
Prediction of Epilepsy Seizures from Intra-Cranial EEG

Y LeCun
MA Ranzato

Piotr Mirowski, Deepak Mahdevan (NYU Neurology), Yann LeCun



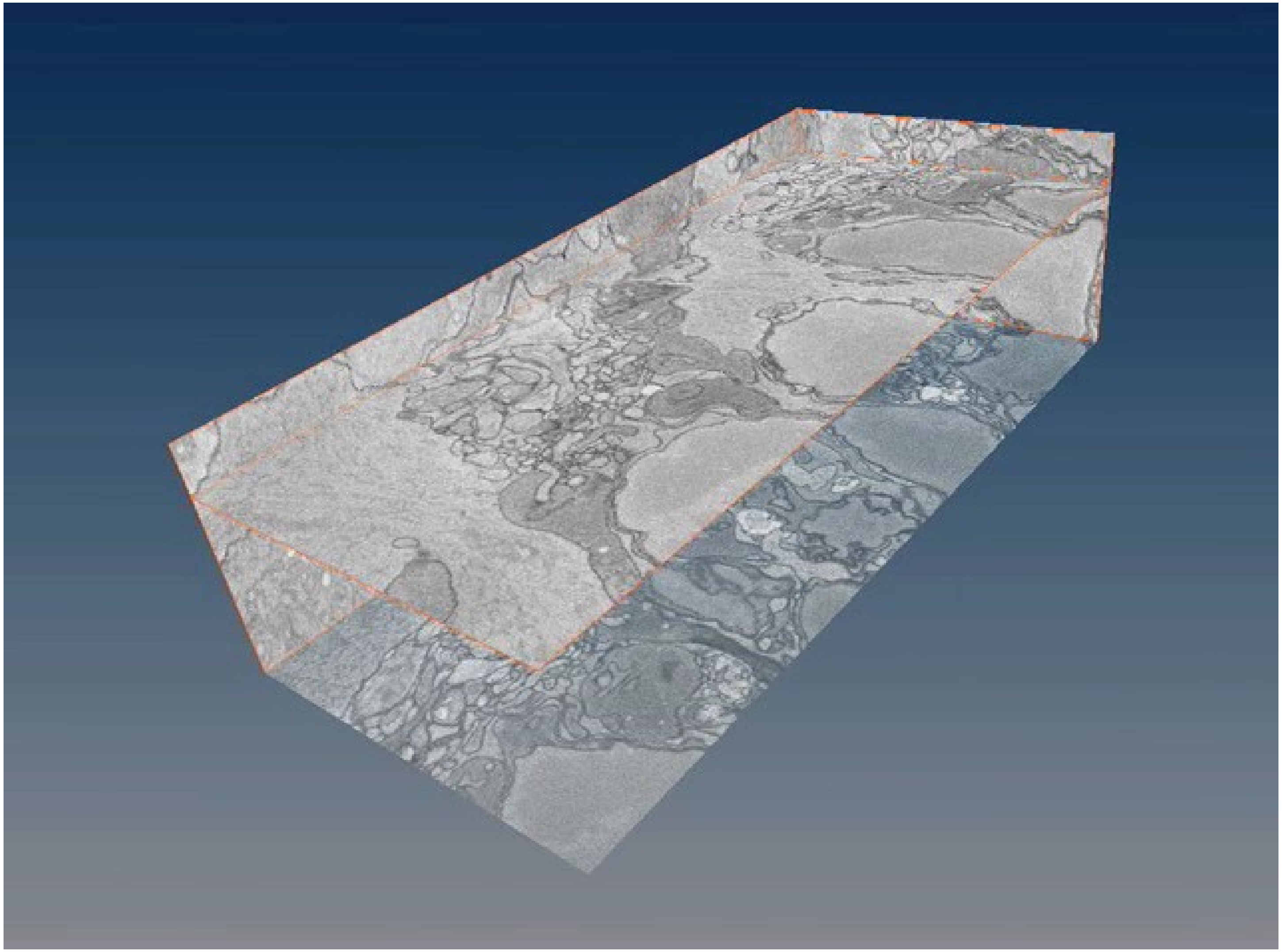
Temporal Convolutional Net



ConvNet in Connectomics [Jain, Turaga, Seung 2007-present]

Y LeCun
MA Ranzato

- 3D convnet to segment volumetric images

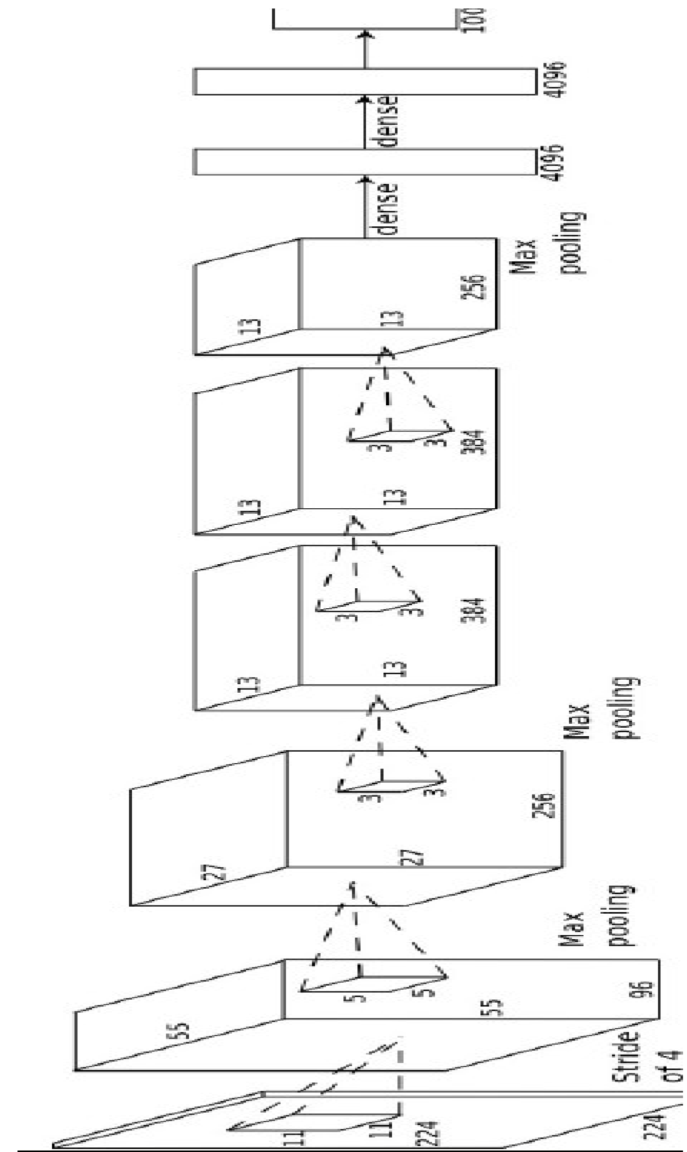


Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

Y LeCun
MA Ranzato

- Won the 2012 ImageNet LSVRC. 60 Million parameters, 832M MAC ops

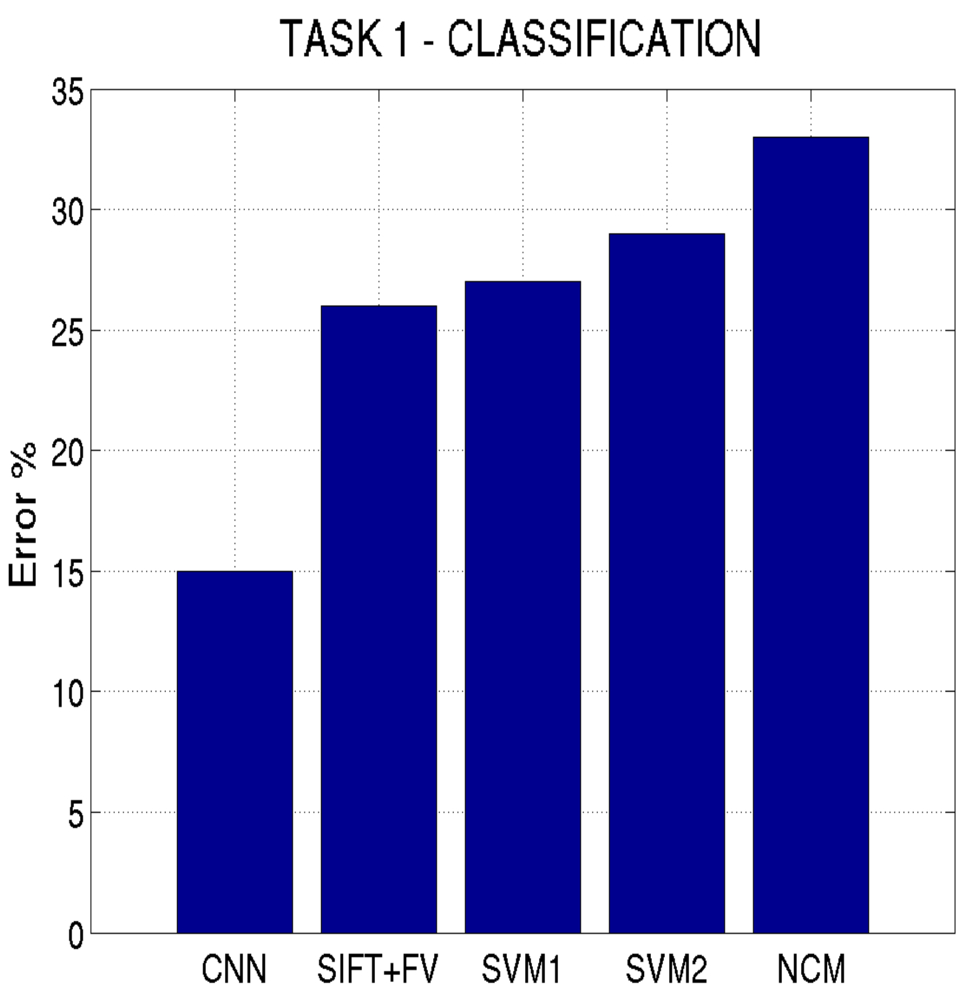
4M	FULL CONNECT	4Mflop
16M	FULL 4096/ReLU	16M
37M	FULL 4096/ReLU	37M
	MAX POOLING	
442K	CONV 3x3/ReLU 256fm	74M
1.3M	CONV 3x3ReLU 384fm	224M
884K	CONV 3x3/ReLU 384fm	149M
	MAX POOLING 2x2sub	
	LOCAL CONTRAST NORM	
307K	CONV 11x11/ReLU 256fm	223M
	MAX POOL 2x2sub	
	LOCAL CONTRAST NORM	
35K	CONV 11x11/ReLU 96fm	105M

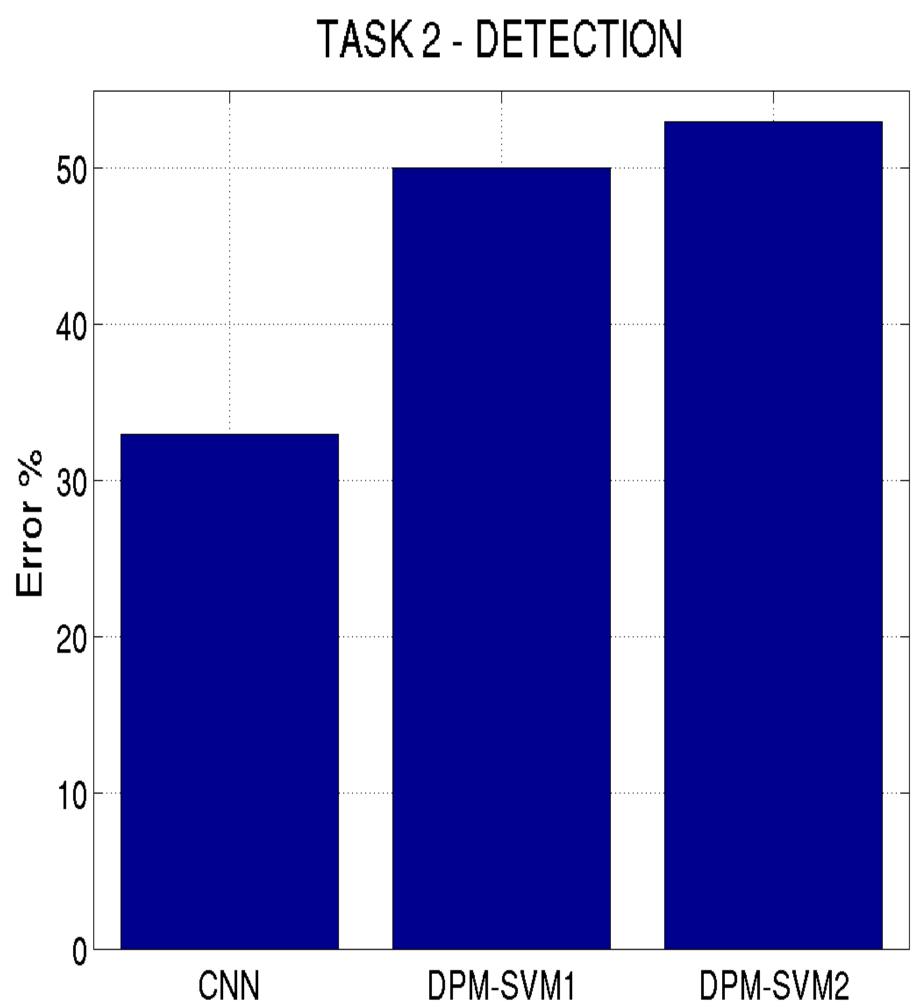


Object Recognition: ILSVRC 2012 results

Y LeCun
MA Ranzato

- ImageNet Large Scale Visual Recognition Challenge
- 1000 categories, 1.5 Million labeled training samples





Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

Y LeCun
MA Ranzato

- Method: large convolutional net
  - 650K neurons, 832M synapses, 60M parameters
  - Trained with backprop on GPU
  - Trained “with all the tricks Yann came up with in the last 20 years, plus dropout” (Hinton, NIPS 2012)
  - Rectification, contrast normalization,...
- Error rate: 15% (whenever correct class isn't in top 5)
- Previous state of the art: 25% error

- A REVOLUTION IN COMPUTER VISION

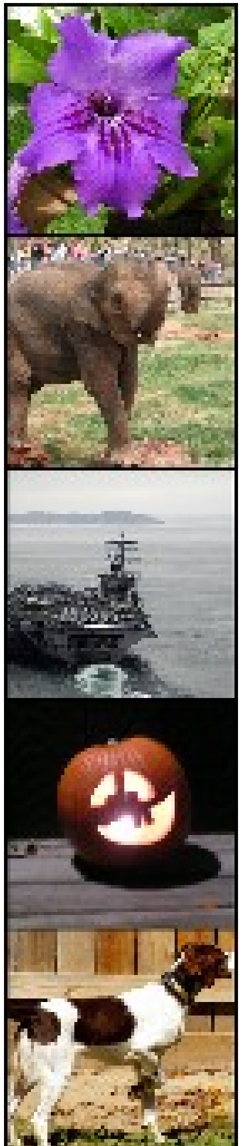
- Acquired by Google in Jan 2013
- Deployed in Google+ Photo Tagging in May 2013

Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

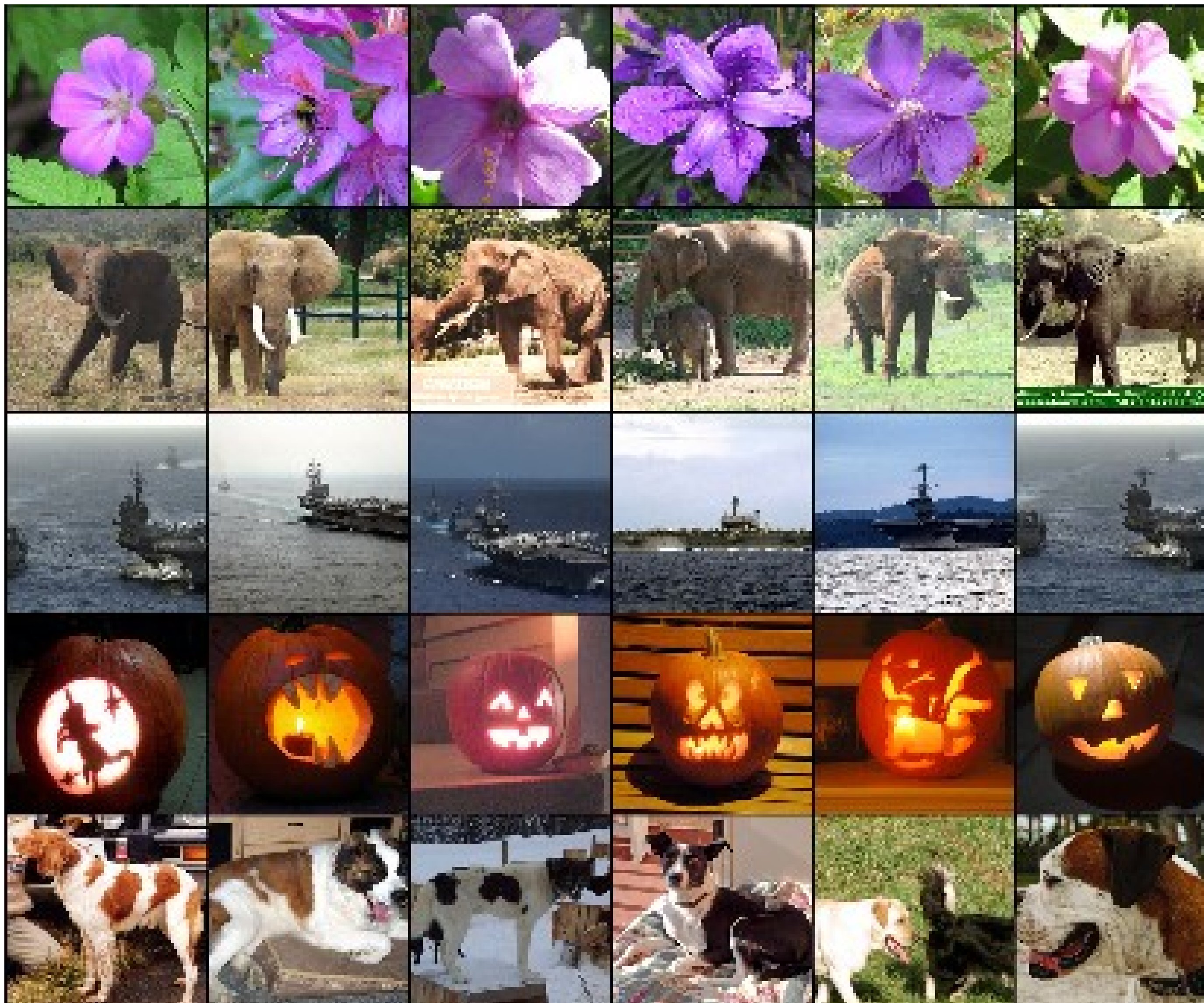
Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

Y LeCun
MA Ranzato

TEST IMAGE



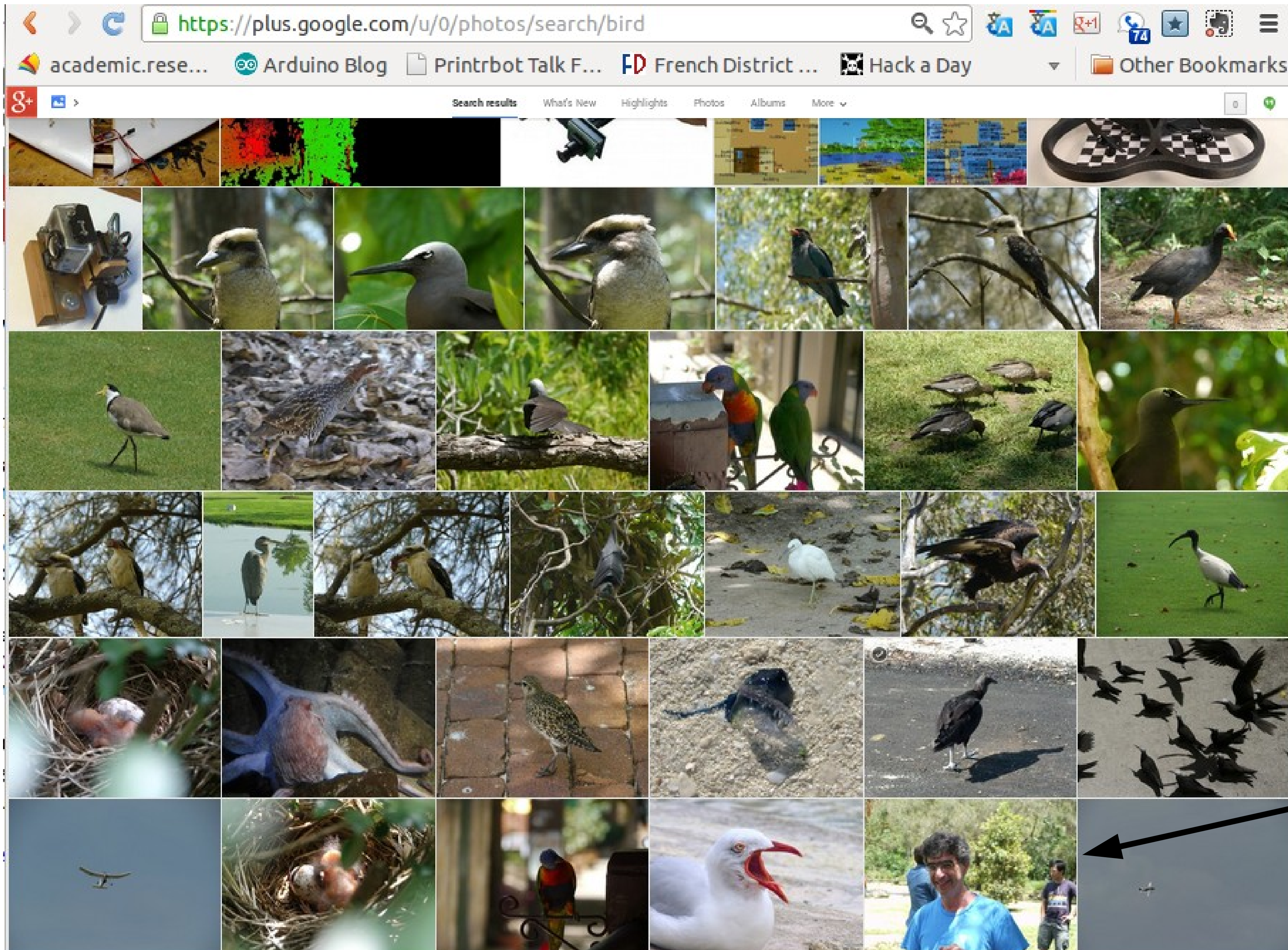
RETRIEVED IMAGES



ConvNet-Based Google+ Photo Tagger

Y LeCun
MA Ranzato

- Searched my personal collection for "bird"

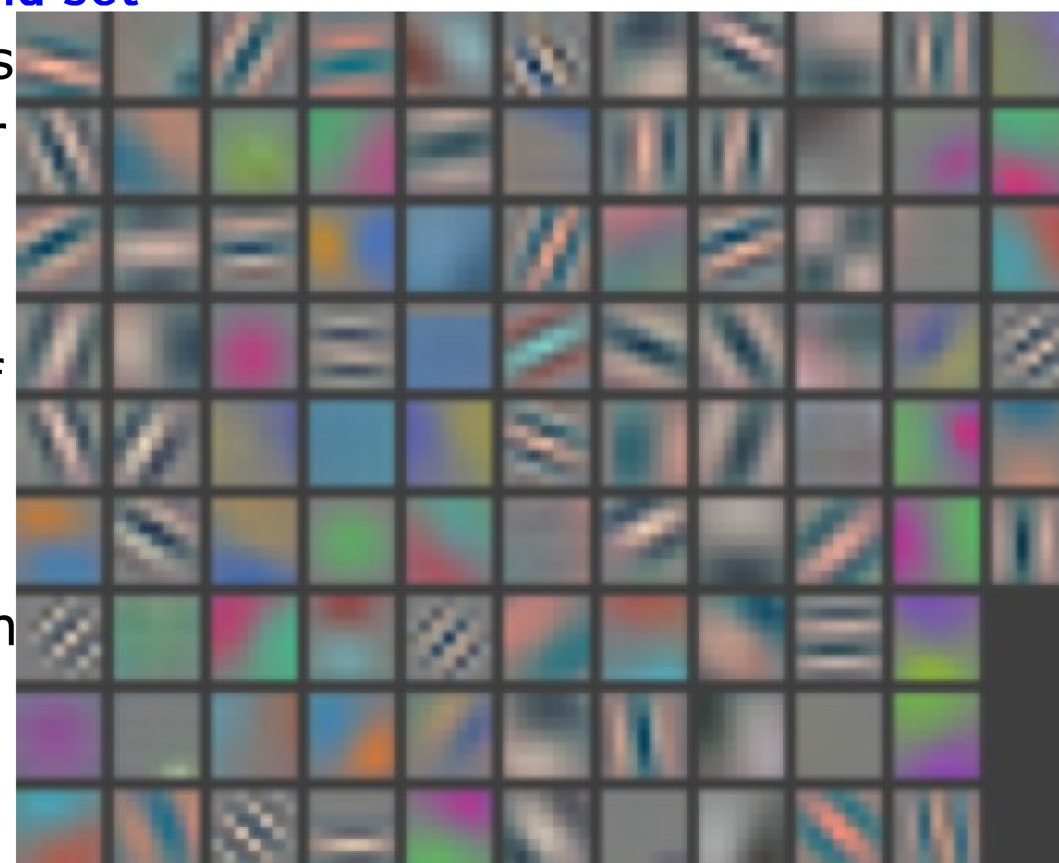


Samy
Bengio
???

Another ImageNet-trained ConvNet [Zeiler & Fergus 2013]

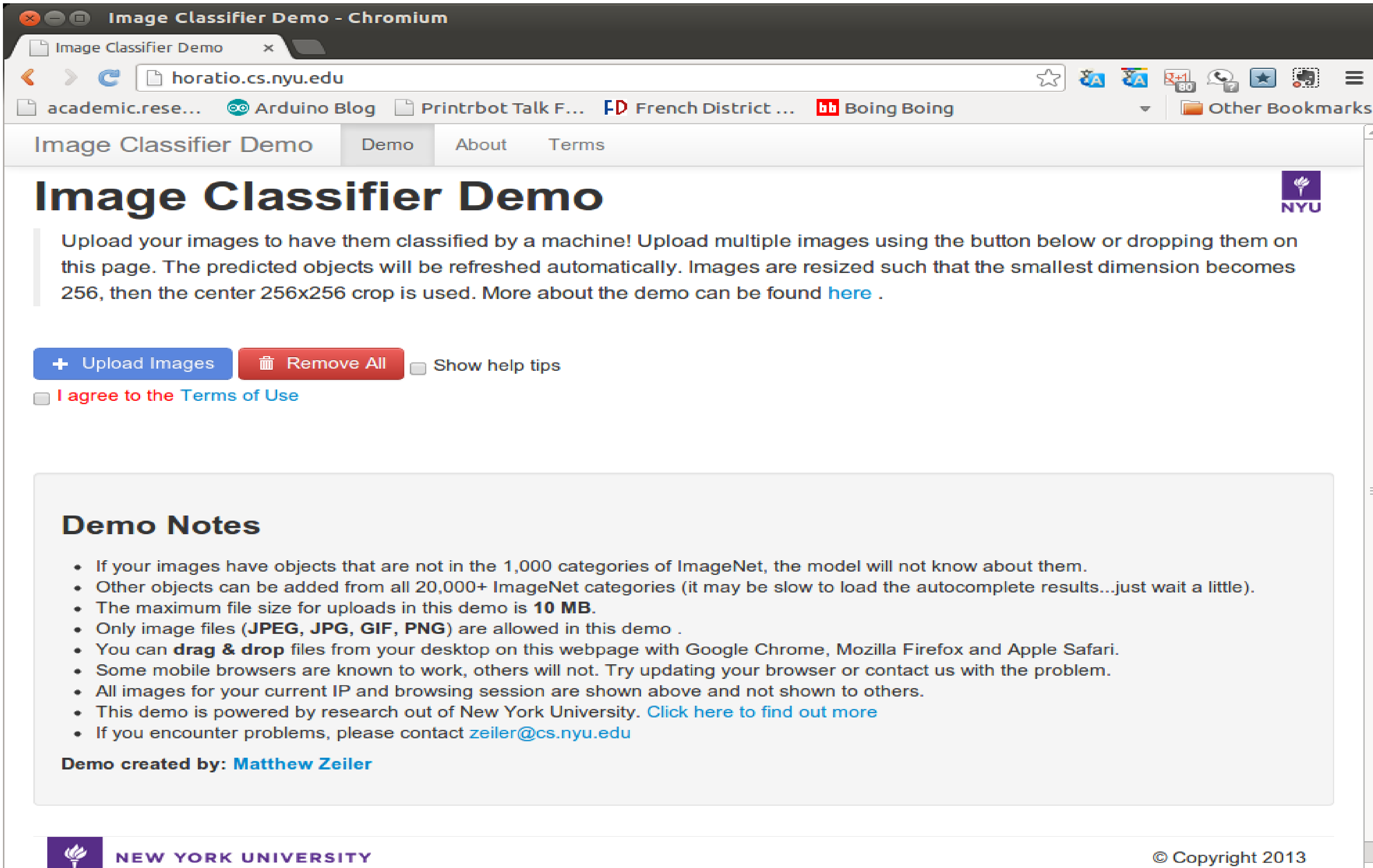
Y LeCun
MA Ranzato

- **Convolutional Net with 8 layers, input is 224x224 pixels**
  - conv-pool-conv-pool-conv-conv-conv-full-full-full
  - Rectified-Linear Units (ReLU): y = max(0,x)
  - Divisive contrast normalization across features [Jarrett et al. ICCV 2009]
- **Trained on ImageNet 2012 training set**
  - 1.3M images, 1000 classes
  - 10 different crops/flips per
- **Regularization: Dropout**
  - [Hinton 2012]
  - zeroing random subsets of
- **Stochastic gradient descent**
  - for 70 epochs (7-10 days)
  - With learning rate annealin



Object Recognition on-line demo [Zeiler & Fergus 2013]

http://horatio.cs.nyu.edu



ConvNet trained on ImageNet [Zeiler & Fergus 2013]

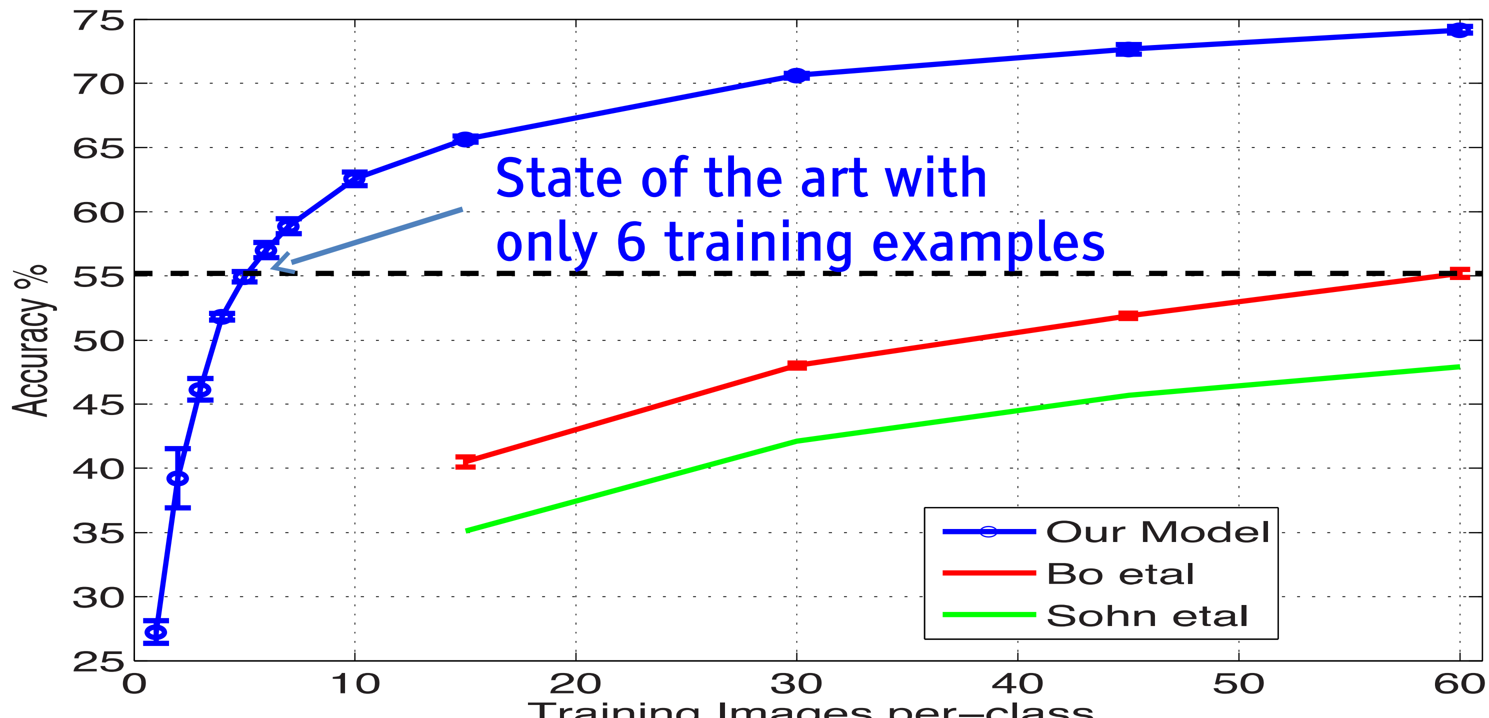
Y LeCun
MA Ranzato

Error %	Val Top-1	Val Top-5	Test Top-5
Deng *et al.* SIFT + FV [7]	−−	−−	26.2
Krizhevsky *et al.* [12], 1 convnet	40.7	18.2	−−
Krizhevsky *et al.* [12], 5 convnets	38.1	16.4	16.4
*Krizhevsky *et al.* [12], 1 convnets	39.0	16.6	−−
*Krizhevsky *et al.* [12], 7 convnets	36.7	15.4	15.3
Our replication of [12], 1 convnet	41.7	19.0	−−
1 convnet - our model	38.4 ± 0.05	16.5 ± 0.05	−−
5 convnets - our model (a)	36.7	15.3	15.3
1 convnet - tweaked model (b)	37.5	16.0	16.1
6 convnets, (a) & (b) combined	**36.0**	**14.7**	**14.8**

Features are generic: Caltech 256

Y LeCun
MA Ranzato

- Network first trained on ImageNet.
- Last layer chopped off
- Last layer trained on Caltech 256,
- first layers N-1 kept fixed.
- State of the art accuracy with only 6 training samples/class



# Train	Acc % 15/class	Acc % 30/class	Acc % 45/class	Acc % 60/class
Sohn *et al.* [16]	35.1	42.1	45.7	47.9
Bo *et al.* [3]	40.5 ± 0.4	48.0 ± 0.2	51.9 ± 0.2	55.2 ± 0.3
Non-pretr.	9.0 ± 1.4	22.5 ± 0.7	31.2 ± 0.5	38.8 ± 1.4
ImageNet-pretr.	**65.7 ± 0.2**	**70.6 ± 0.2**	**72.7 ± 0.4**	**74.2 ± 0.3**

3: [Bo, Ren, Fox. CVPR, 2013] 16: [Sohn, Jung, Lee, Hero ICCV 2011]

Features are generic: PASCAL VOC 2012

- Network first trained on ImageNet.
- Last layer trained on Pascal VOC, keeping N-1 first layers fixed.

Acc %	[15]	[19]	Ours	Acc %	[15]	[19]	Ours
Airplane	92.0	**97.3**	96.0	Dining table	63.2	**77.8**	67.7
Bicycle	74.2	**84.2**	77.1	Dog	68.9	83.0	**87.8**
Bird	73.0	80.8	**88.4**	Horse	78.2	**87.5**	86.0
Boat	77.5	85.3	**85.5**	Motorbike	81.0	**90.1**	85.1
Bottle	54.3	**60.8**	55.8	Person	91.6	**95.0**	90.9
Bus	85.2	**89.9**	85.8	Potted plant	55.9	**57.8**	52.2
Car	81.9	**86.8**	78.6	Sheep	69.4	79.2	**83.6**
Cat	76.4	89.3	**91.2**	Sofa	65.4	**73.4**	61.1
Chair	65.2	**75.4**	65.0	Train	86.7	**94.5**	91.8
Cow	63.2	**77.8**	74.4	Tv/monitor	77.4	**80.7**	76.1
Mean	74.3	**82.2**	79.0	# won	0	**15**	5

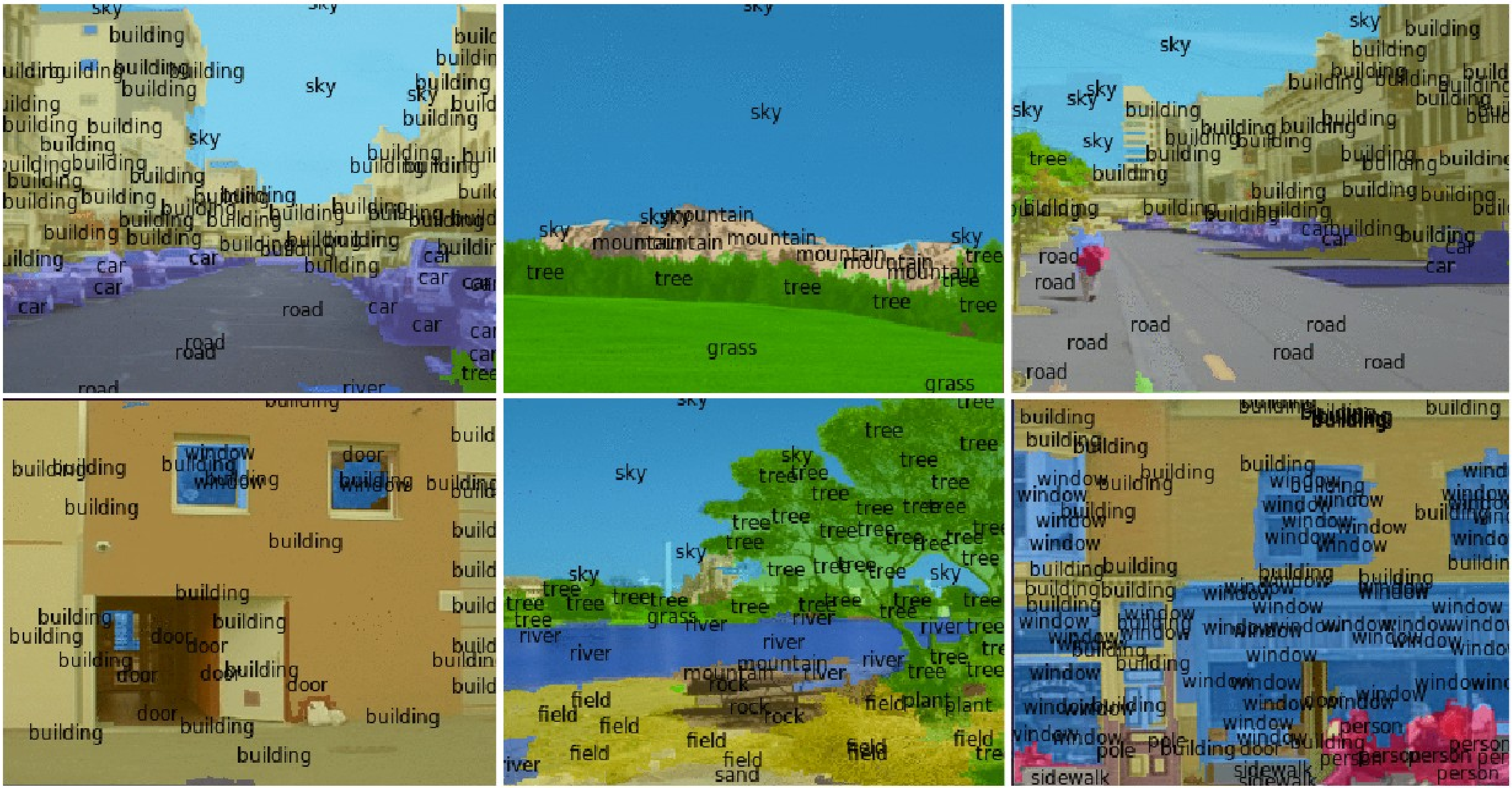
[15] K. Sande, J. Uijlings, C. Snoek, and A. Smeulders. Hybrid coding for selective search. In PASCAL VOC Classification Challenge 2012,

[19] S. Yan, J. Dong, Q. Chen, Z. Song, Y. Pan, W. Xia, Z. Huang, Y. Hua, and S. Shen. Generalized hierarchical matching for sub-category aware object classification. In PASCAL VOC Classification Challenge 2012

Semantic Labeling:
Labeling every pixel with the object it belongs to

Y LeCun
MA Ranzato

- Would help identify obstacles, targets, landing sites, dangerous areas
- Would help line up depth map with edge maps

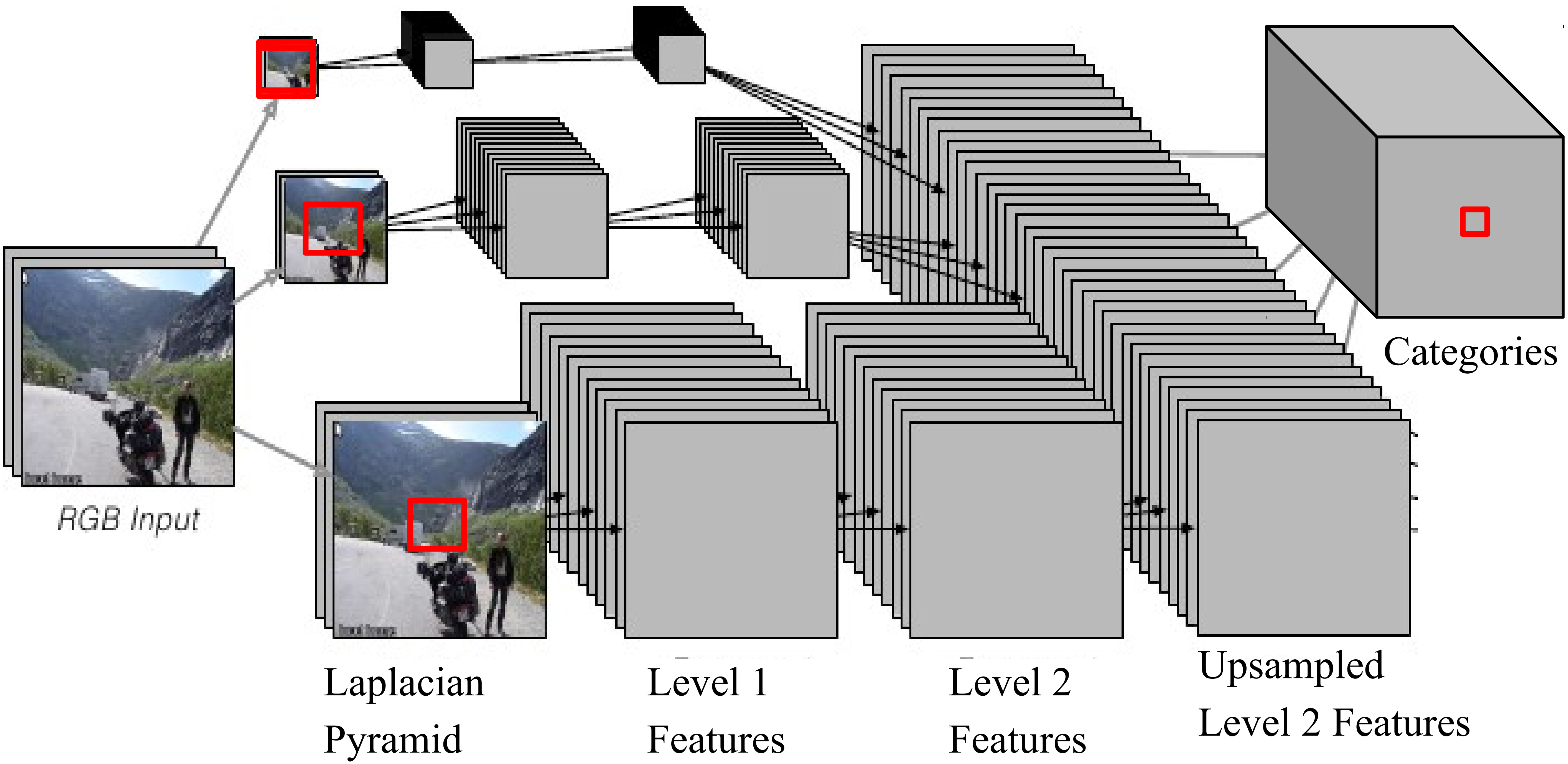


[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling: ConvNet Architecture

Y LeCun
MA Ranzato

- **Each output sees a large input context:**
  - **46x46** window at full rez; **92x92** at ½ rez; **184x184** at ¼ rez
  - [7x7conv]->[2x2pool]->[7x7conv]->[2x2pool]->[7x7conv]->
  - Trained supervised on fully-labeled images



Scene Parsing/Labeling: Performance

- Stanford Background Dataset [Gould 1009]: 8 categories

	Pixel Acc.	Class Acc.	CT (sec.)
Gould *et al.* 2009 [14]	76.4%	-	10 to 600s
Munoz *et al.* 2010 [32]	76.9%	66.2%	12s
Tighe *et al.* 2010 [46]	77.5%	-	10 to 300s
Socher *et al.* 2011 [45]	78.1%	-	?
Kumar *et al.* 2010 [22]	79.4%	-	$< 600s$
Lempitzky *et al.* 2011 [28]	**81.9%**	72.4%	$> 60s$
singlescale convnet	66.0 %	56.5 %	0.35s
multiscale convnet	78.8 %	72.4%	0.6s
multiscale net + superpixels	80.4%	74.56%	**0.7s**
multiscale net + gPb + cover	80.4%	75.24%	61s
multiscale net + CRF on gPb	81.4%	**76.0%**	60.5s

[Farabet et al. IEEE T. PAMI 2013]

Scene Parsing/Labeling: Performance

Y LeCun
MA Ranzato

	Pixel Acc.	Class Acc.
Liu *et al.* 2009 [31]	74.75%	-
Tighe *et al.* 2010 [44]	76.9%	29.4%
raw multiscale net[1]	67.9%	45.9%
multiscale net + superpixels[1]	71.9%	**50.8%**
multiscale net + cover[1]	72.3%	**50.8%**
multiscale net + cover[2]	**78.5%**	29.6%

- SIFT Flow Dataset
- [Liu 2009]:
- 33 categories

	Pixel Acc.	Class Acc.
Tighe *et al.* 2010 [44]	66.9%	7.6%
raw multiscale net[1]	37.8%	**12.1%**
multiscale net + superpixels[1]	44.1%	**12.4%**
multiscale net + cover[1]	46.4%	**12.5%**
multiscale net + cover[2]	**67.8%**	9.5%

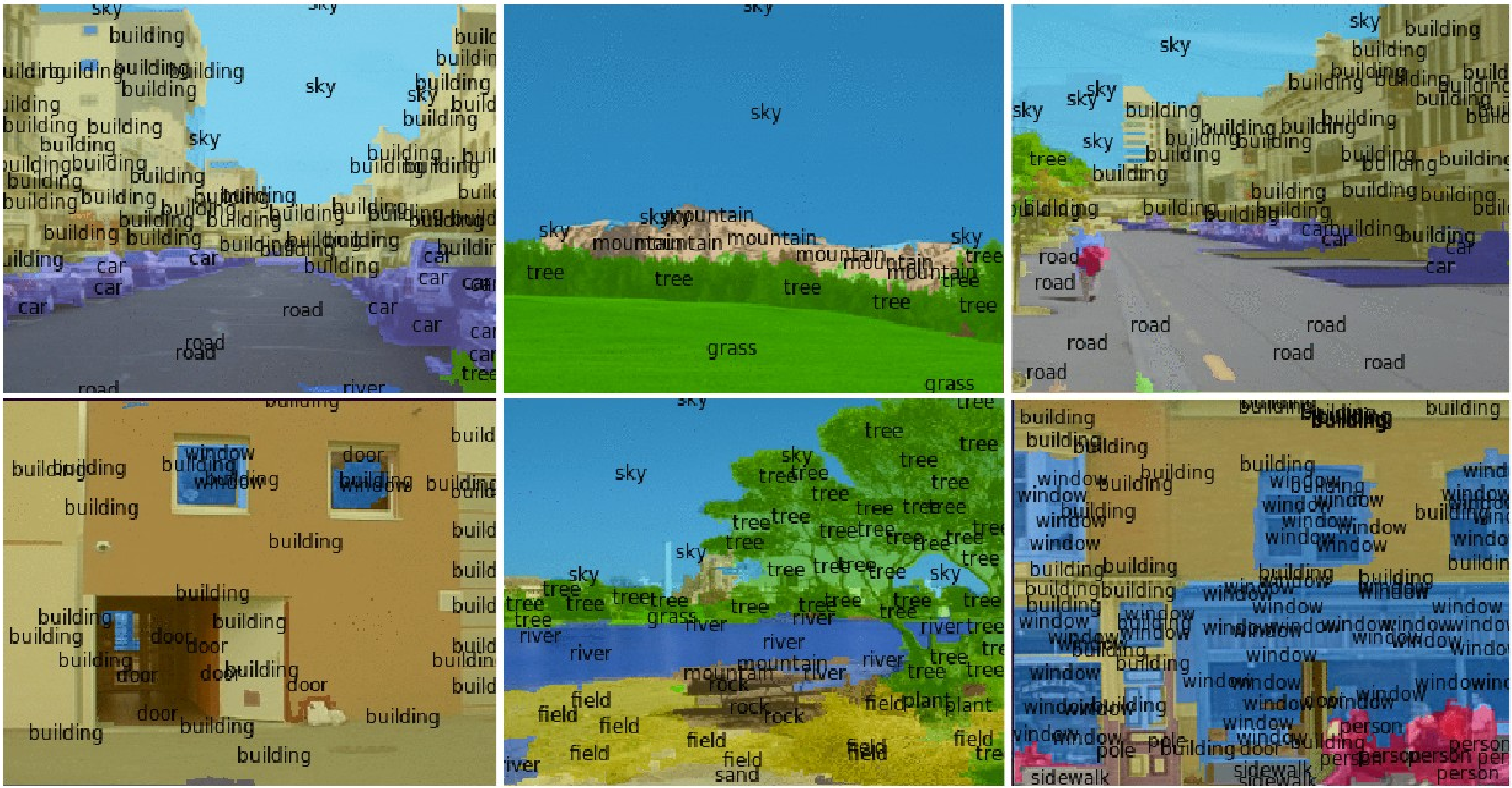
- Barcelona dataset
- [Tighe 2010]:
- 170 categories.

[Farabet et al. IEEE T. PAMI 2012]

Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

Y LeCun
MA Ranzato

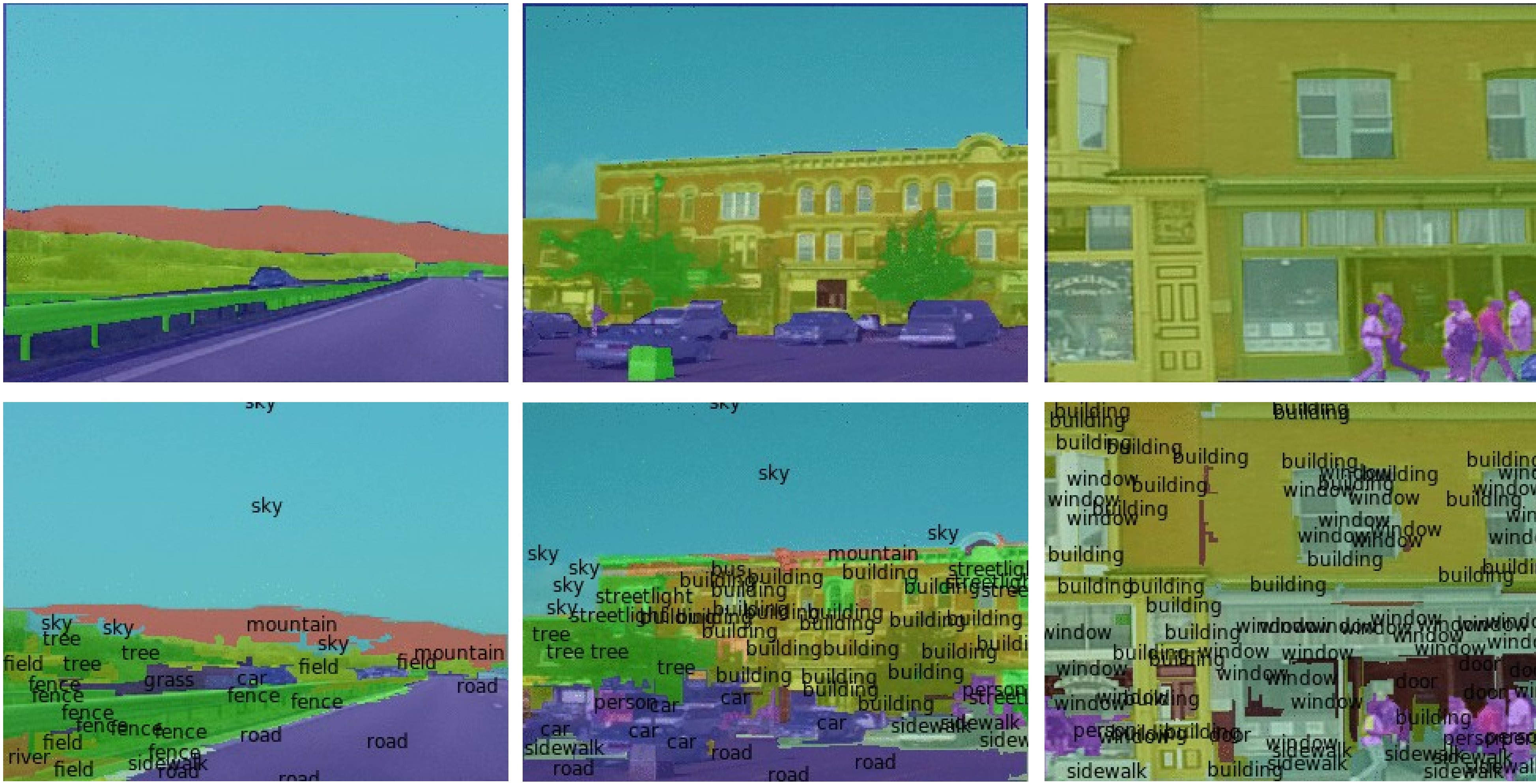
- Samples from the SIFT-Flow dataset (Liu)



[Farabet et al. ICML 2012, PAMI 2013]

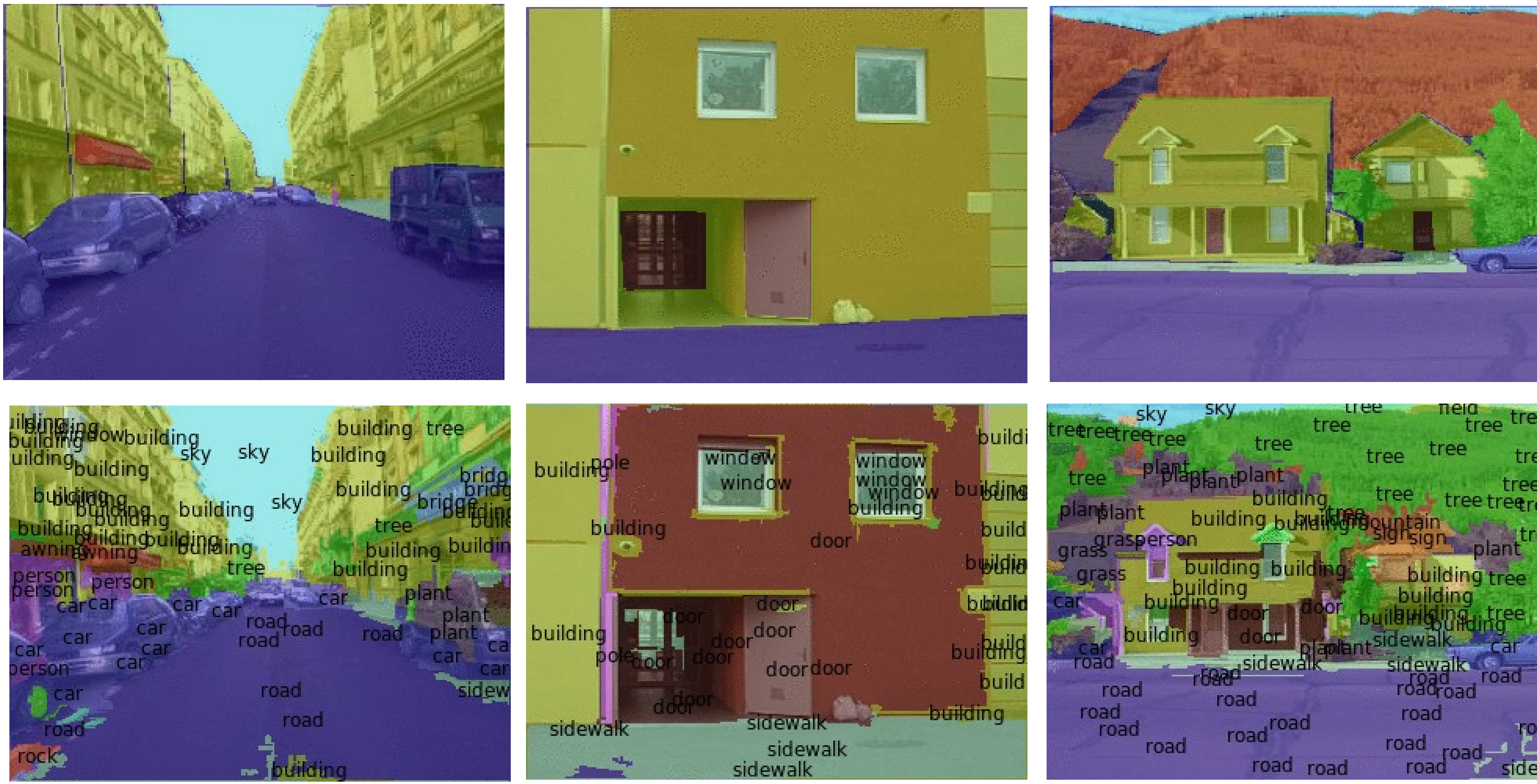
Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

Y LeCun
MA Ranzato



[Farabet et al. ICML 2012, PAMI 2013]

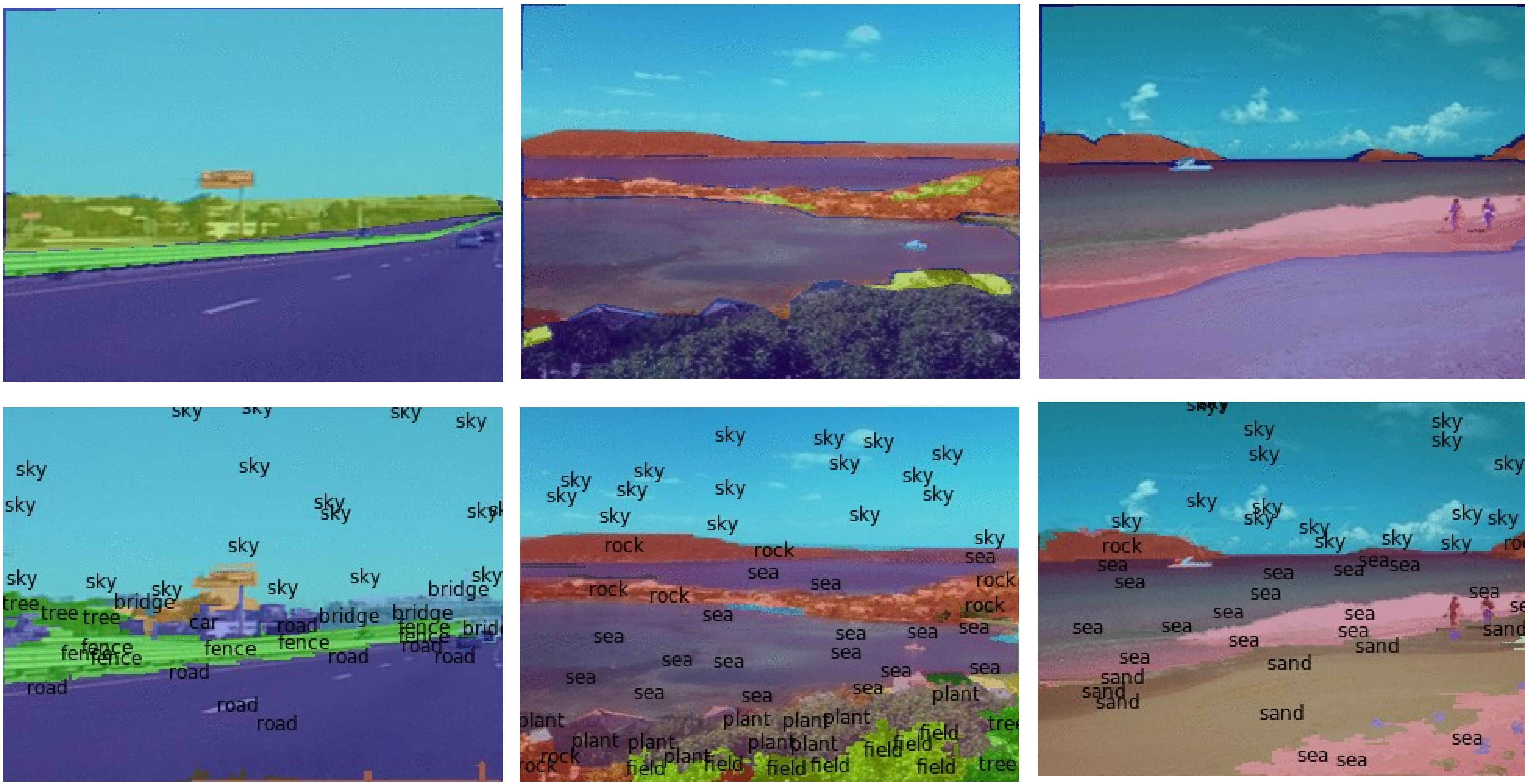
Scene Parsing/Labeling



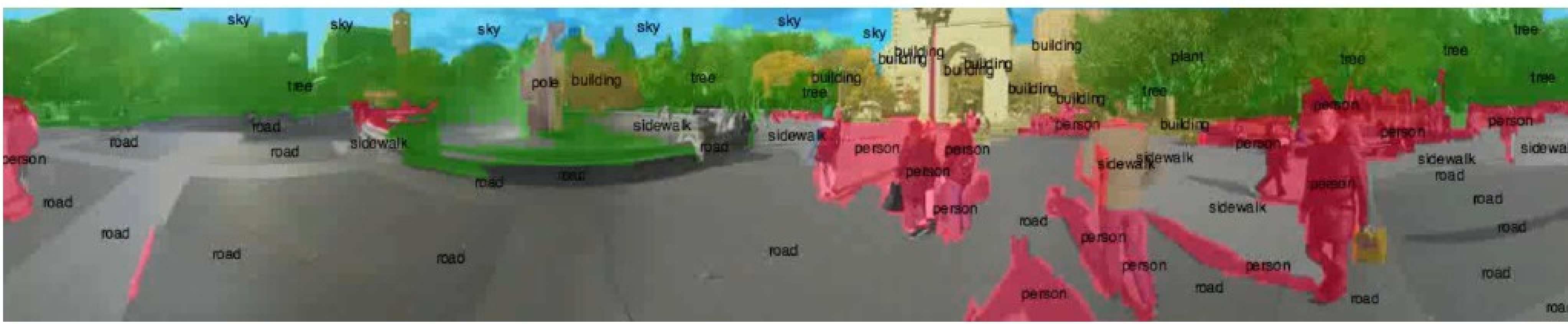
[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling

Y LeCun
MA Ranzato

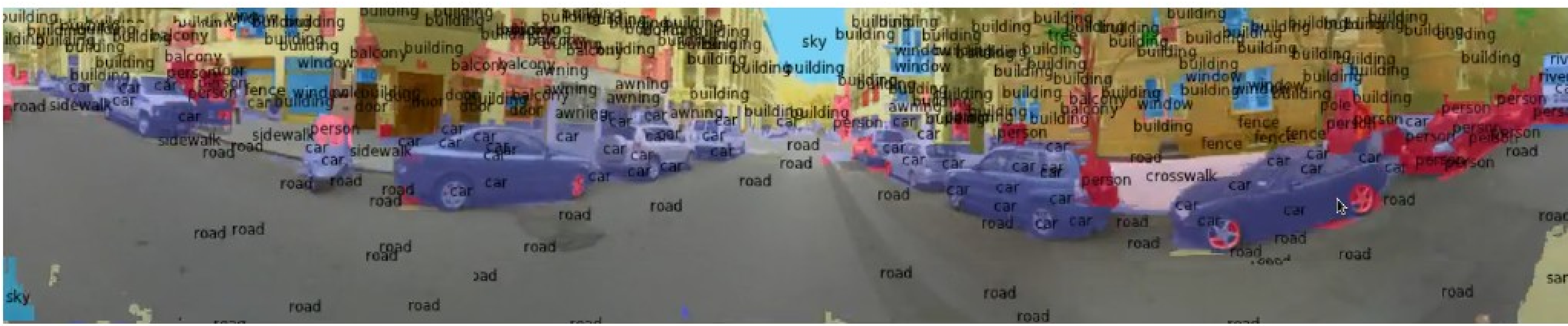


[Farabet et al. ICML 2012, PAMI 2013]



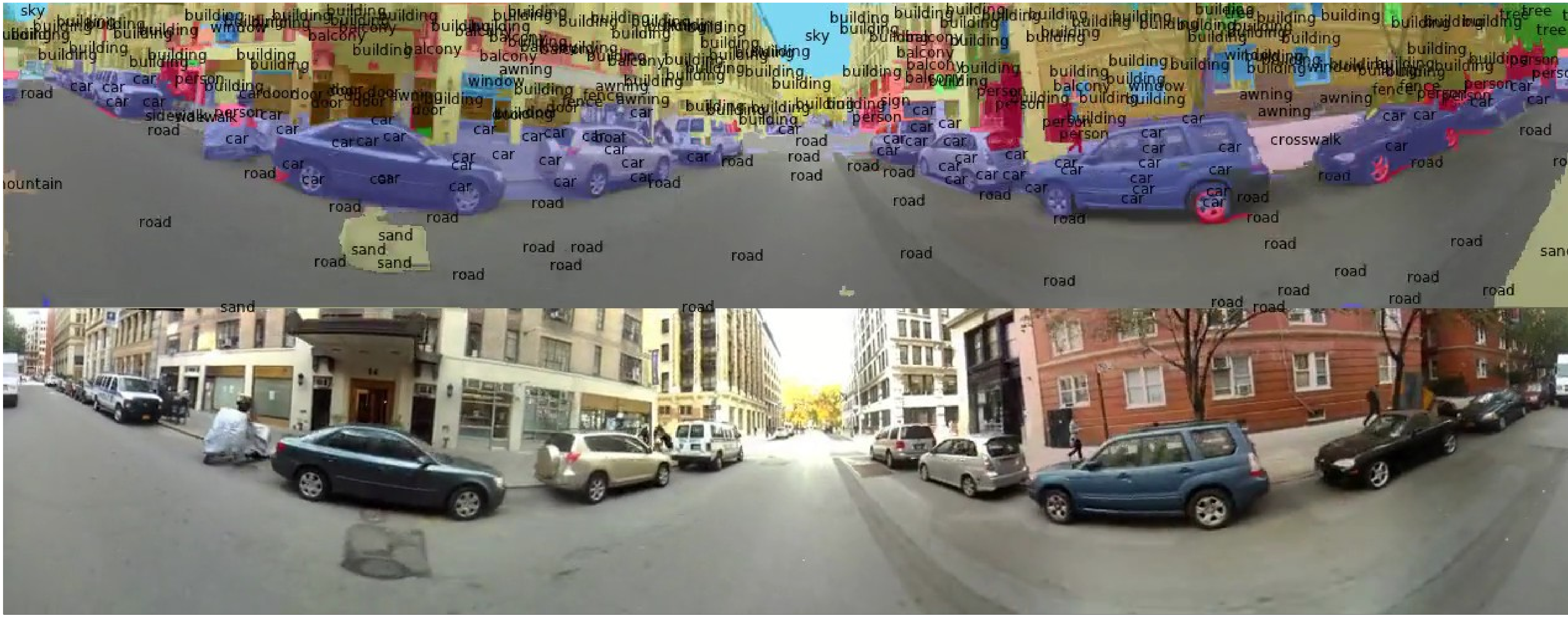
[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling



[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling



- No post-processing
- Frame-by-frame
- ConvNet runs at 50ms/frame on Virtex-6 FPGA hardware
  - But communicating the features over ethernet limits system performance

Scene Parsing/Labeling: Temporal Consistency

- Causal method for temporal consistency

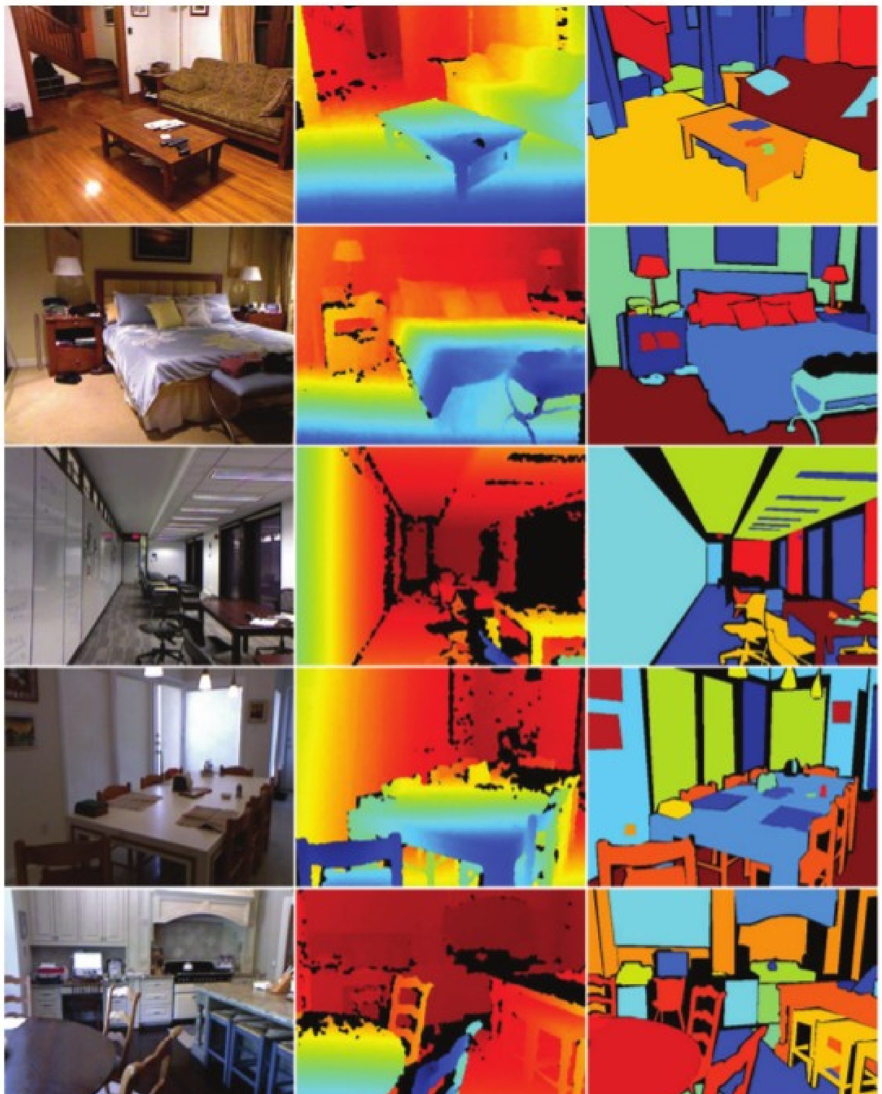
[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]

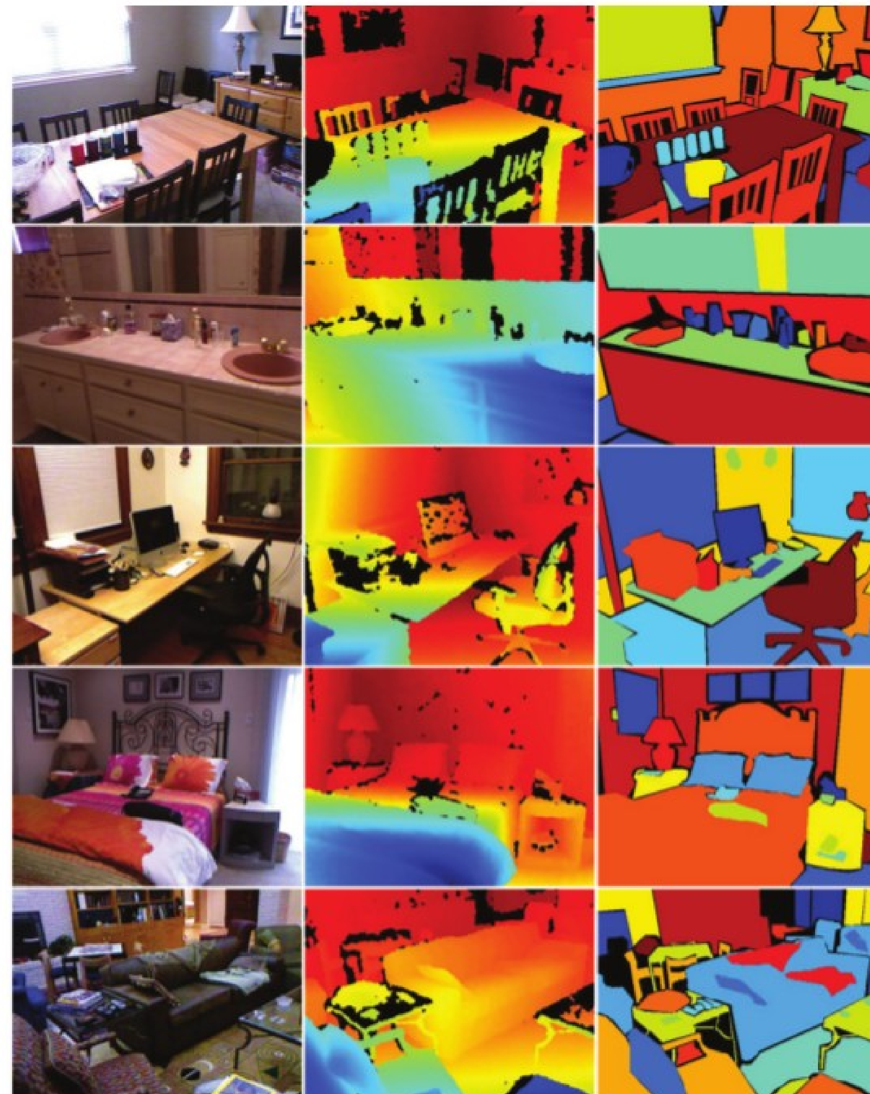
NYU RGB-Depth Indoor Scenes Dataset

Y LeCun
MA Ranzato

- 407024 RGB-D images of apartments
- 1449 labeled frames, 894 object categories

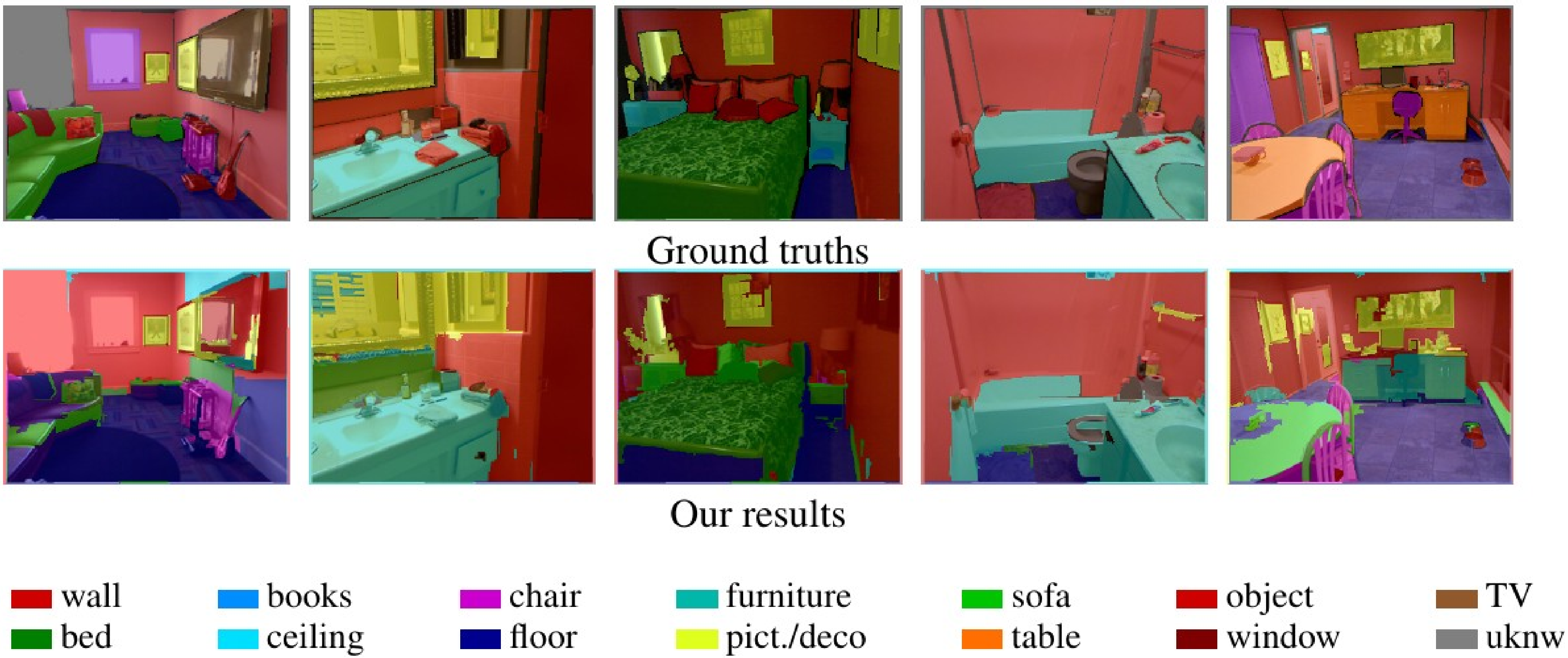
[Silberman et al. 2012]





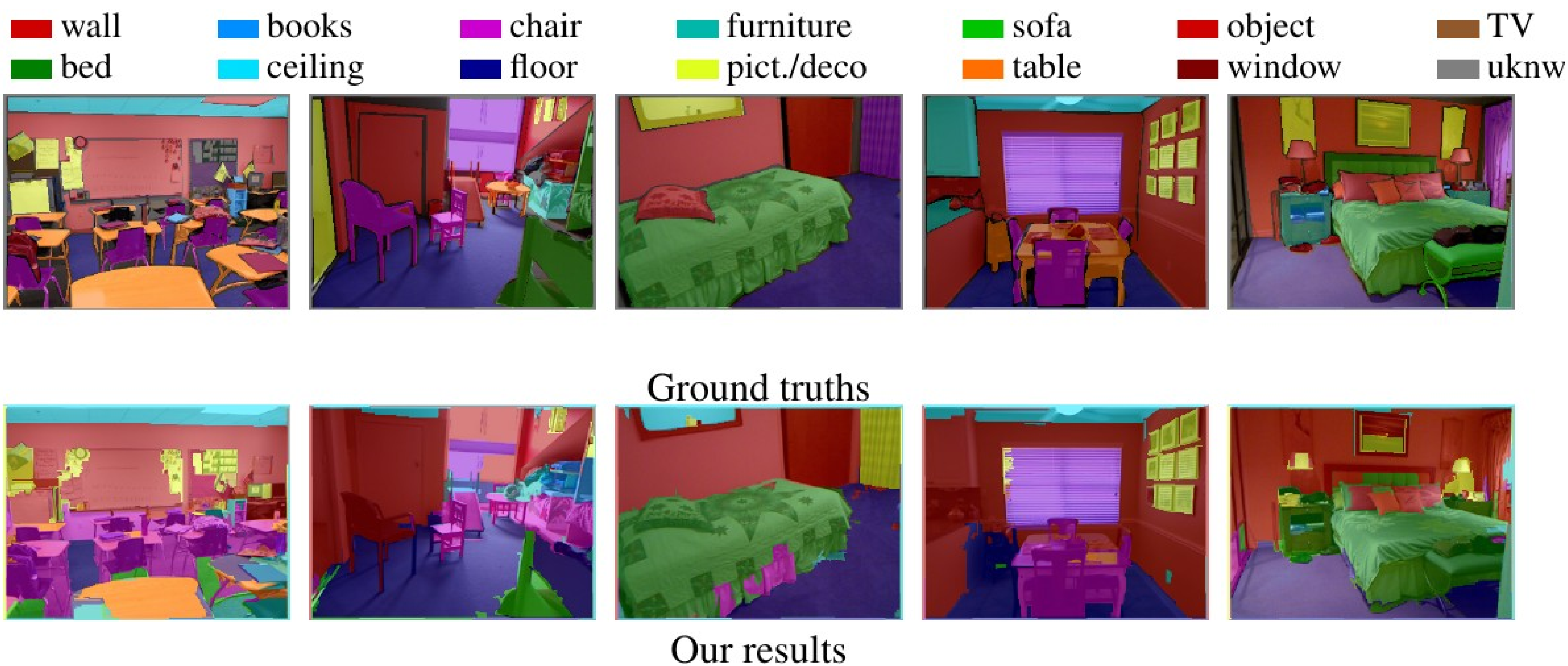
Scene Parsing/Labeling on RGB+Depth Images

Y LeCun
MA Ranzato



[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]

Scene Parsing/Labeling on RGB+Depth Images



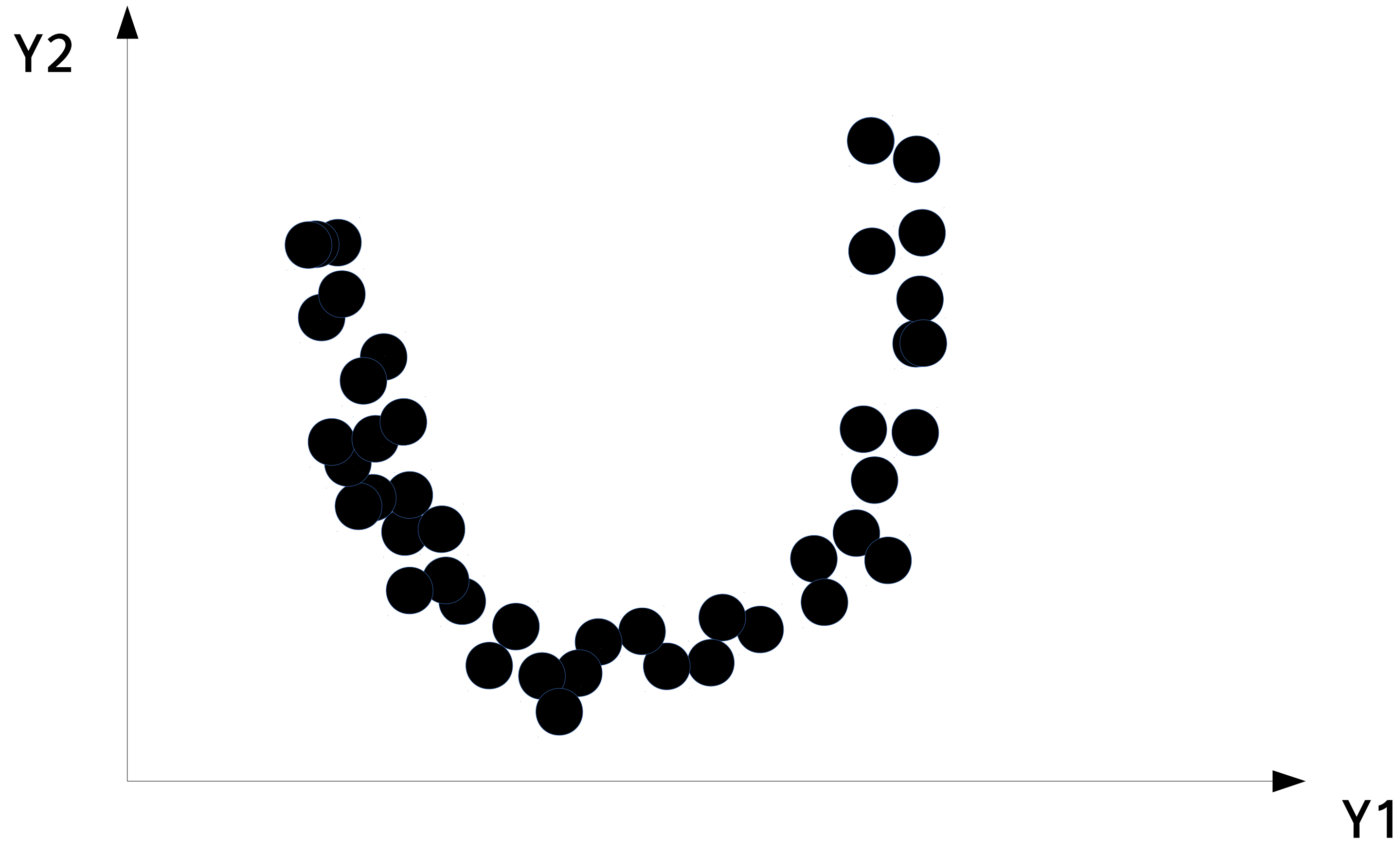
[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]

Semantic Segmentation on RGB+D Images and Videos

[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]

Energy-Based Unsupervised Learning

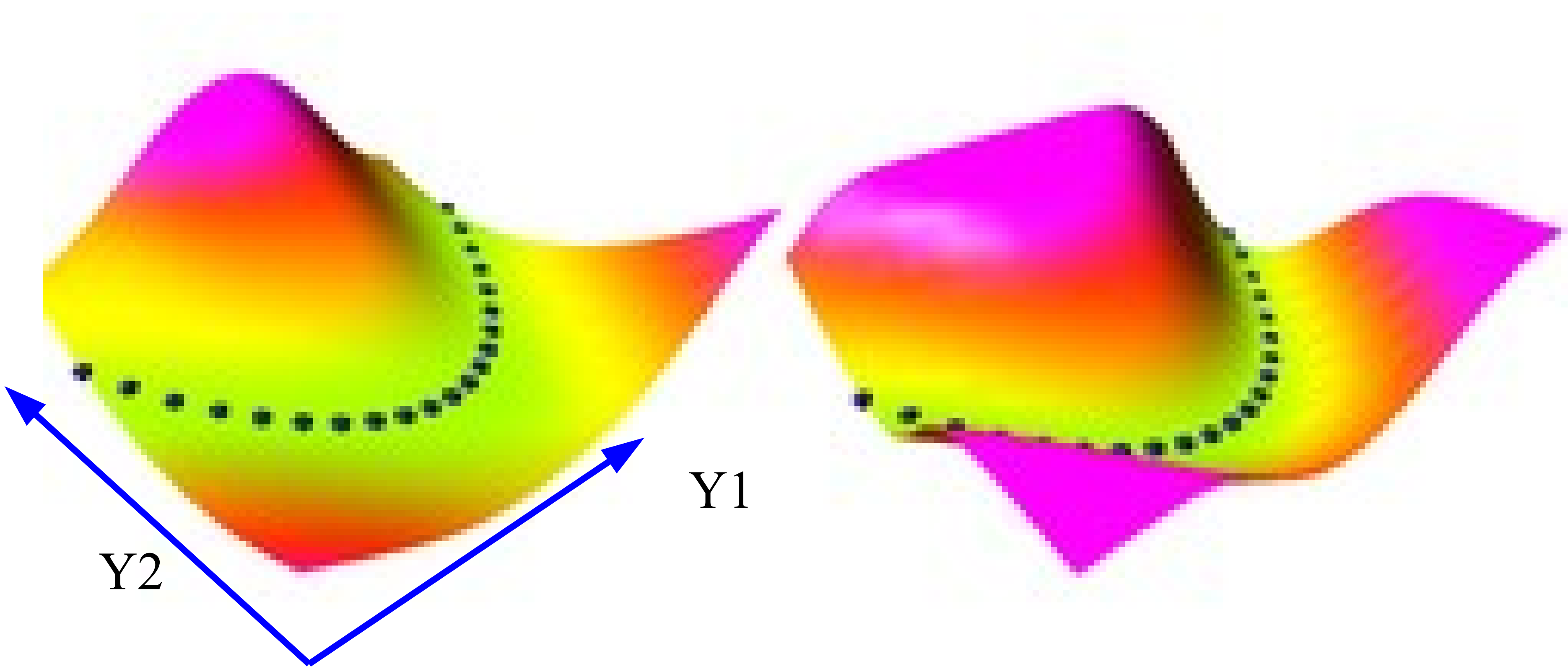
- Learning an energy function (or contrast function) that takes
  - Low values on the data manifold
  - Higher values everywhere else



Capturing Dependencies Between Variables with an Energy Function

Y LeCun
MA Ranzato

- **The energy surface is a "contrast function" that takes low values on the data manifold, and higher values everywhere else**
  - Special case: energy = negative log density
  - Example: the samples live in the manifold $Y_2 = (Y_1)^2$



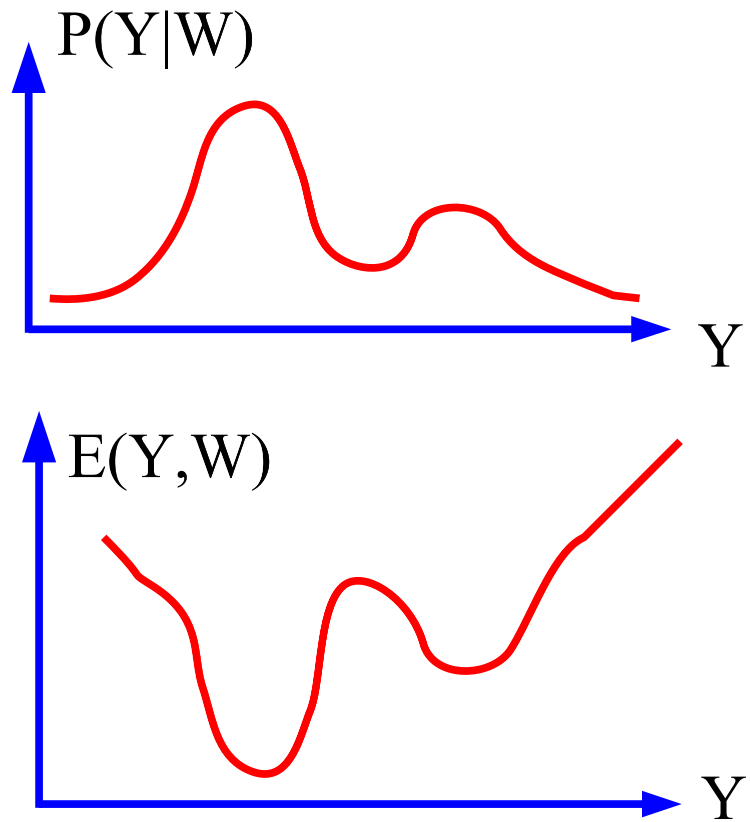
Transforming Energies into Probabilities (if necessary)

Y LeCun
MA Ranzato

- **The energy can be interpreted as an unnormalized negative log density**
- **Gibbs distribution: Probability proportional to exp(-energy)**
  - Beta parameter is akin to an inverse temperature
- **Don't compute probabilities unless you absolutely have to**
  - Because the denominator is often intractable

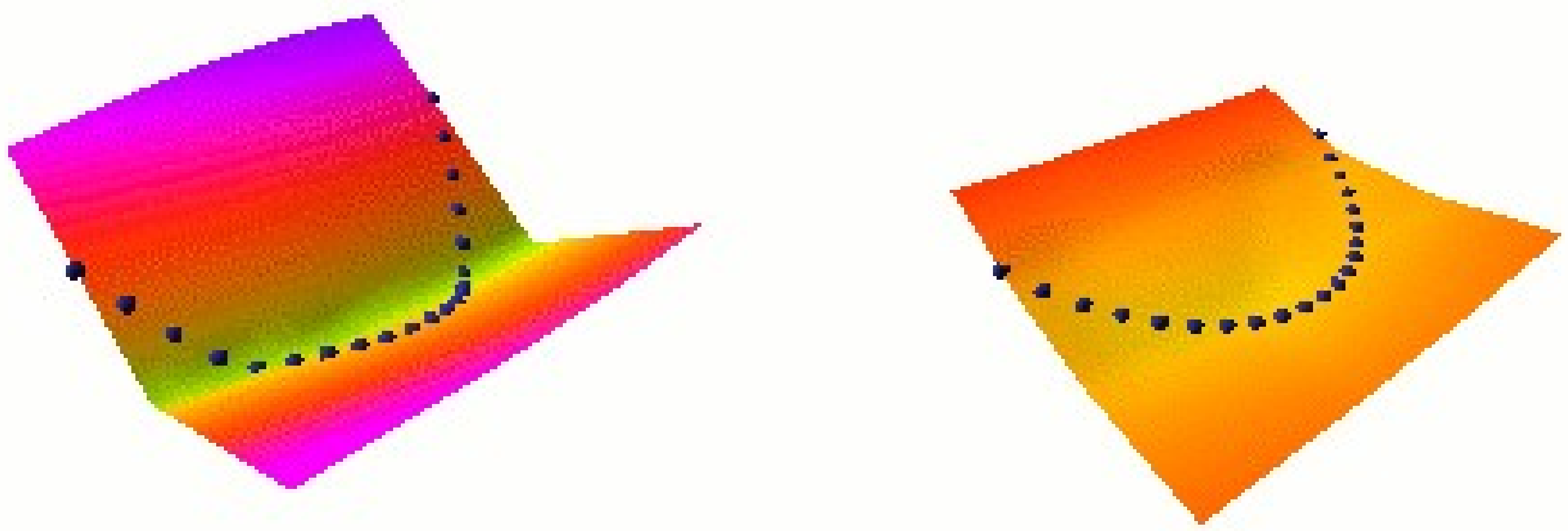
$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

$$E(Y,W) \propto -\log P(Y|W)$$



Learning the Energy Function

- **parameterized energy function E(Y,W)**
  - Make the energy low on the samples
  - Make the energy higher everywhere else
  - Making the energy low on the samples is easy
  - But how do we make it higher everywhere else?



Seven Strategies to Shape the Energy Function

- **1. build the machine so that the volume of low energy stuff is constant**
  - PCA, K-means, GMM, square ICA
- **2. push down of the energy of data points, push up everywhere else**
  - Max likelihood (needs tractable partition function)
- **3. push down of the energy of data points, push up on chosen locations**
  - contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow
- **4. minimize the gradient and maximize the curvature around data points**
  - score matching
- **5. train a dynamical system so that the dynamics goes to the manifold**
  - denoising auto-encoder
- **6. use a regularizer that limits the volume of space that has low energy**
  - Sparse coding, sparse auto-encoder, PSD
- **7. if E(Y) = ||Y - G(Y)||^2, make G(Y) as "constant" as possible.**
  - Contracting auto-encoder, saturating auto-encoder

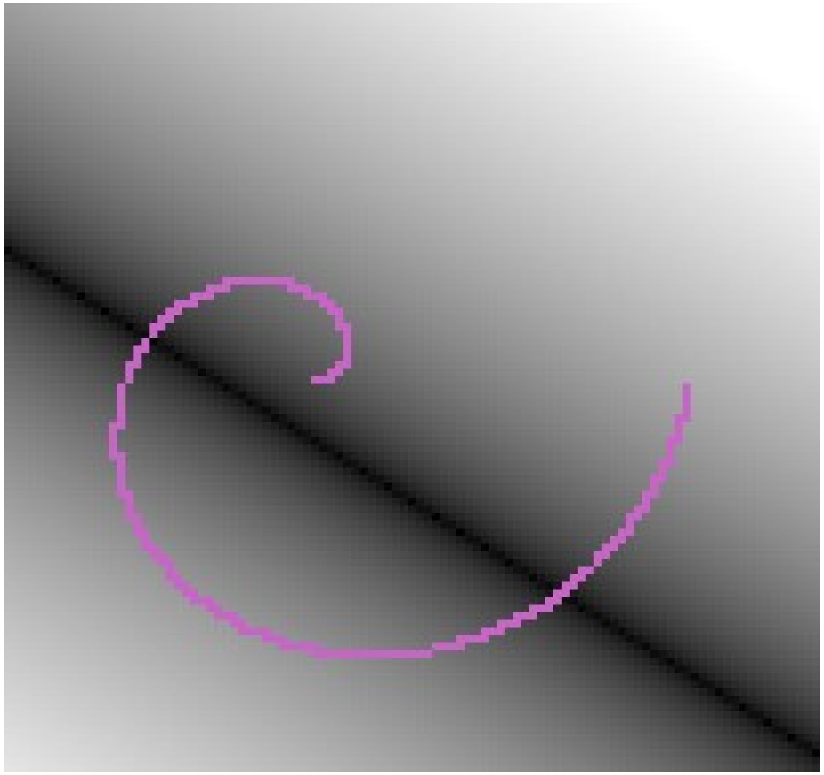
#1: constant volume of low energy

Y LeCun
MA Ranzato

- 1. build the machine so that the volume of low energy stuff is constant
  - PCA, K-means, GMM, square ICA...

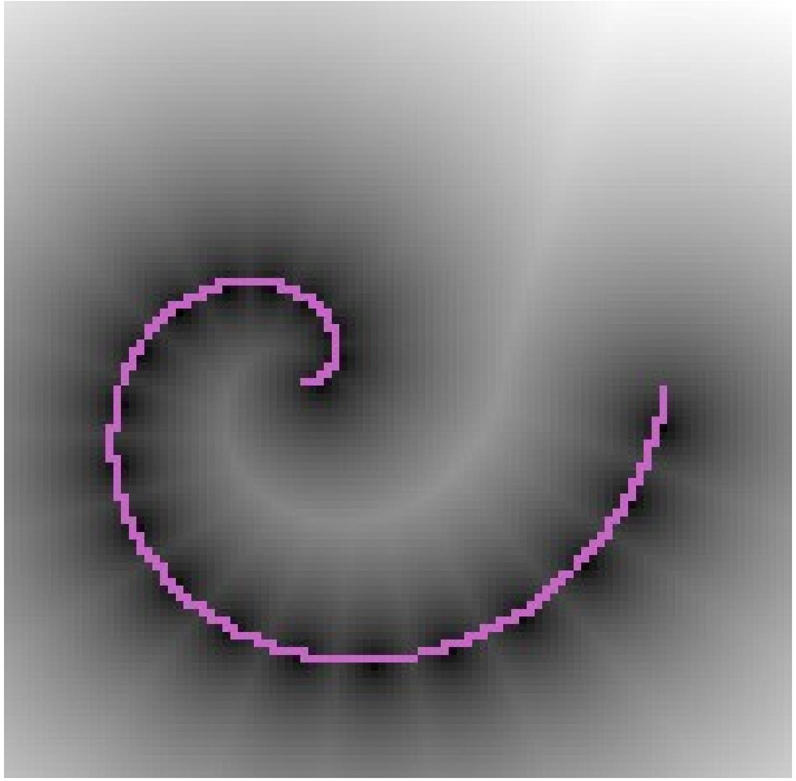
PCA

$$E(Y) = \|W^T W Y - Y\|^2$$



K-Means,
Z constrained to 1-of-K code

$$E(Y) = min_z \sum_i \|Y - W_i Z_i\|^2$$



#2: push down of the energy of data points, push up everywhere else

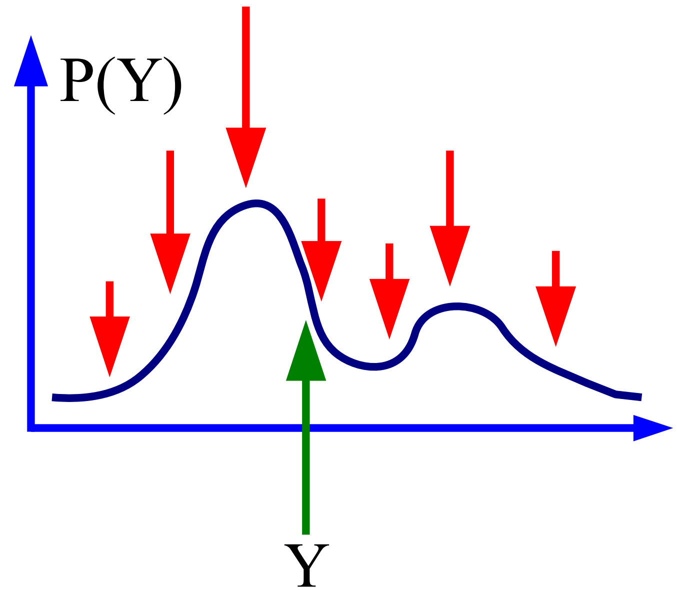
- **Max likelihood (requires a tractable partition function)**

Maximizing P(Y|W) on training samples

make this big

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

make this small

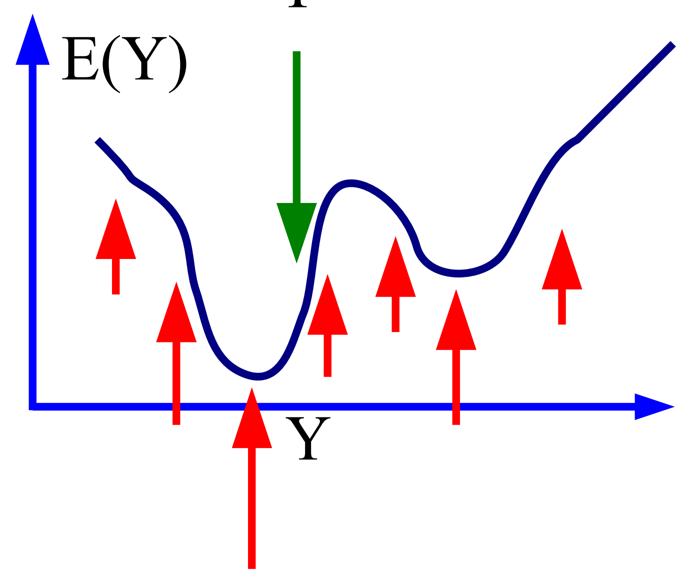


Minimizing -log P(Y,W) on training samples

$$L(Y,W) = E(Y,W) + \frac{1}{\beta} \log \int_y e^{-\beta E(y,W)}$$

make this small

make this big



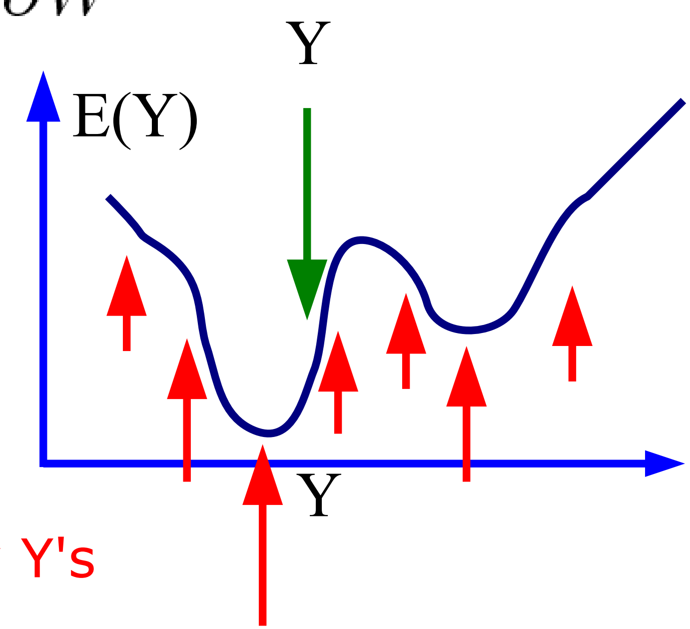
#2: push down of the energy of data points, push up everywhere else

Gradient of the negative log-likelihood loss for one sample Y:

$$\frac{\partial L(Y,W)}{\partial W} = \frac{\partial E(Y,W)}{\partial W} - \int_y P(y|W)\frac{\partial E(y,W)}{\partial W}$$

Gradient descent:

$$W \leftarrow W - \eta \frac{\partial L(Y,W)}{\partial W}$$



Pushes down on the energy of the samples

Pulls up on the energy of low-energy Y's

$$W \leftarrow W - \eta \frac{\partial E(Y,W)}{\partial W} + \eta \int_y P(y|W)\frac{\partial E(y,W)}{\partial W}$$

#3. push down of the energy of data points, push up on chosen locations

Y LeCun
MA Ranzato

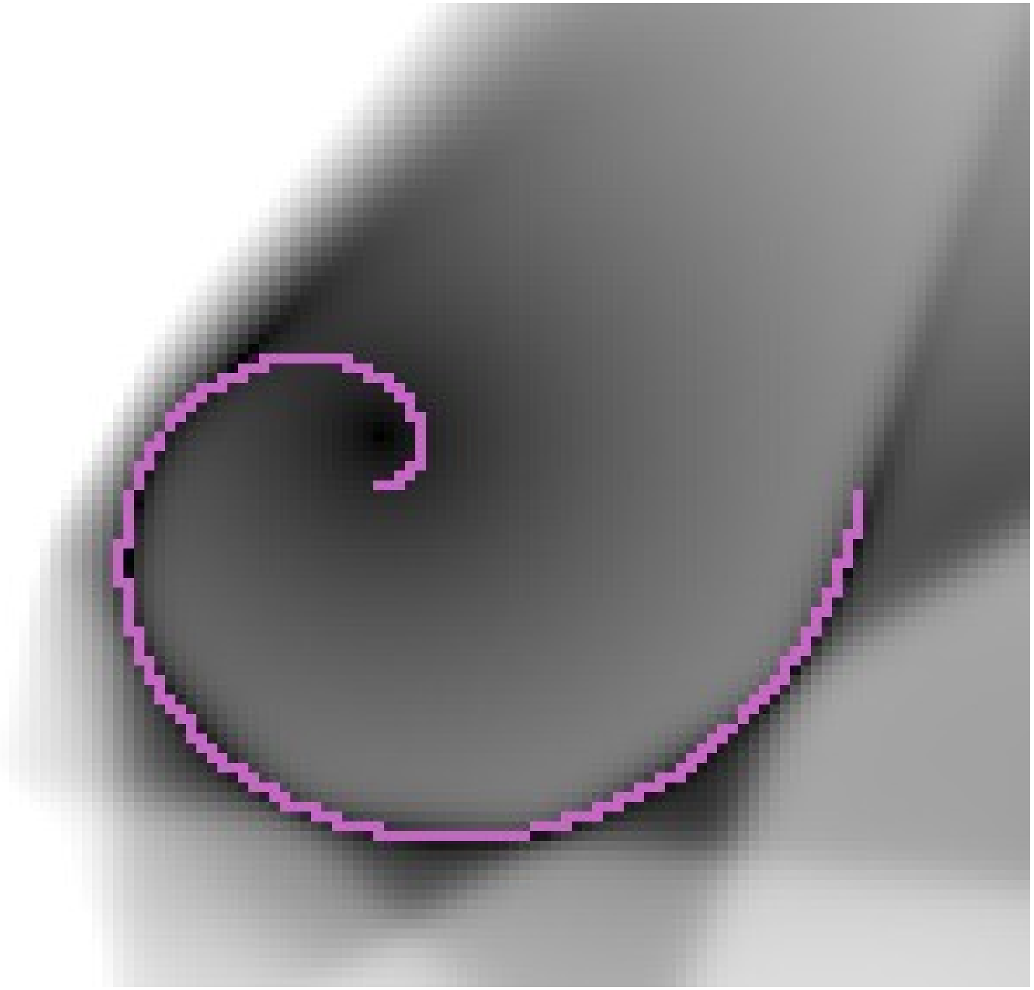
- contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow
- Contrastive divergence: basic idea
  - Pick a training sample, lower the energy at that point
  - From the sample, move down in the energy surface with noise
  - Stop after a while
  - Push up on the energy of the point where we stopped
  - This creates grooves in the energy surface around data manifolds
  - CD can be applied to any energy function (not just RBMs)
- Persistent CD: use a bunch of “particles” and remember their positions
  - Make them roll down the energy surface with noise
  - Push up on the energy wherever they are
  - Faster than CD
- RBM

$$E(Y,Z) = -Z^T W Y \qquad E(Y) = -\log \sum_z e^{Z^T W Y}$$

#6. use a regularizer that limits the volume of space that has low energy

Y LeCun
MA Ranzato

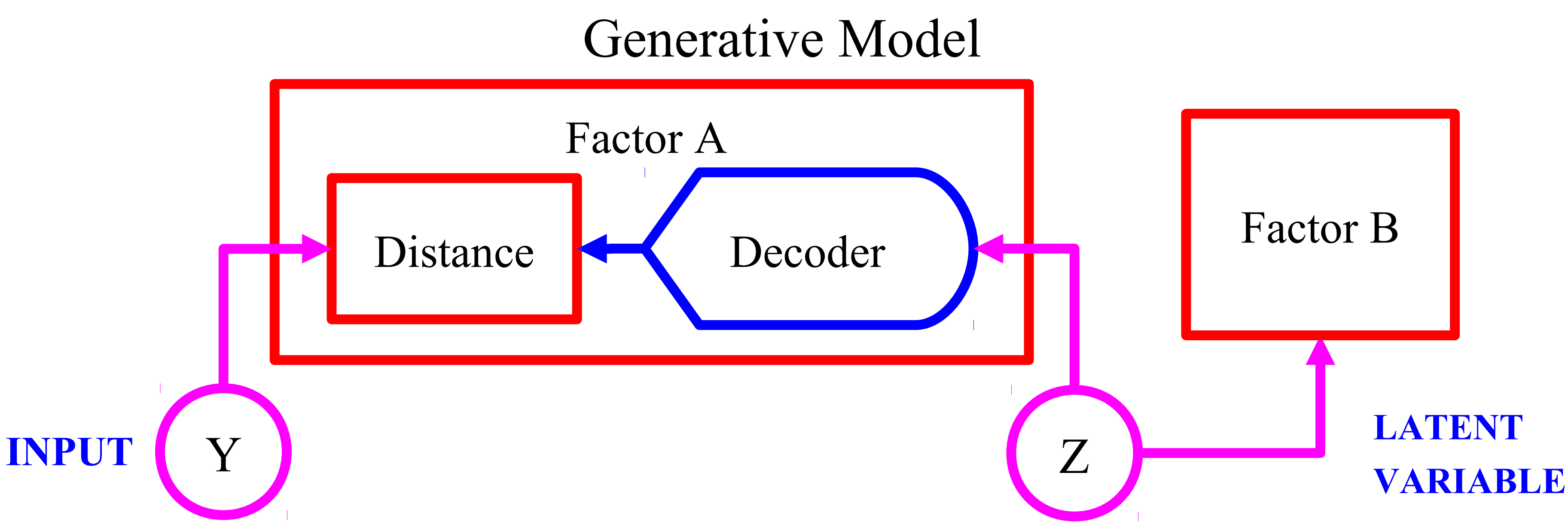
- Sparse coding, sparse auto-encoder, Predictive Saprse Decomposition



Sparse Modeling,
Sparse Auto-Encoders,
Predictive Sparse Decomposition
LISTA

How to Speed Up Inference in a Generative Model?

- Factor Graph with an asymmetric factor
- Inference Z → Y is easy
  - Run Z through deterministic decoder, and sample Y
- Inference Y → Z is hard, particularly if Decoder function is many-to-one
  - MAP: minimize sum of two factors with respect to Z
  - Z* = argmin_z Distance[Decoder(Z), Y] + FactorB(Z)
- Examples: K-Means (1of K), Sparse Coding (sparse), Factor Analysis

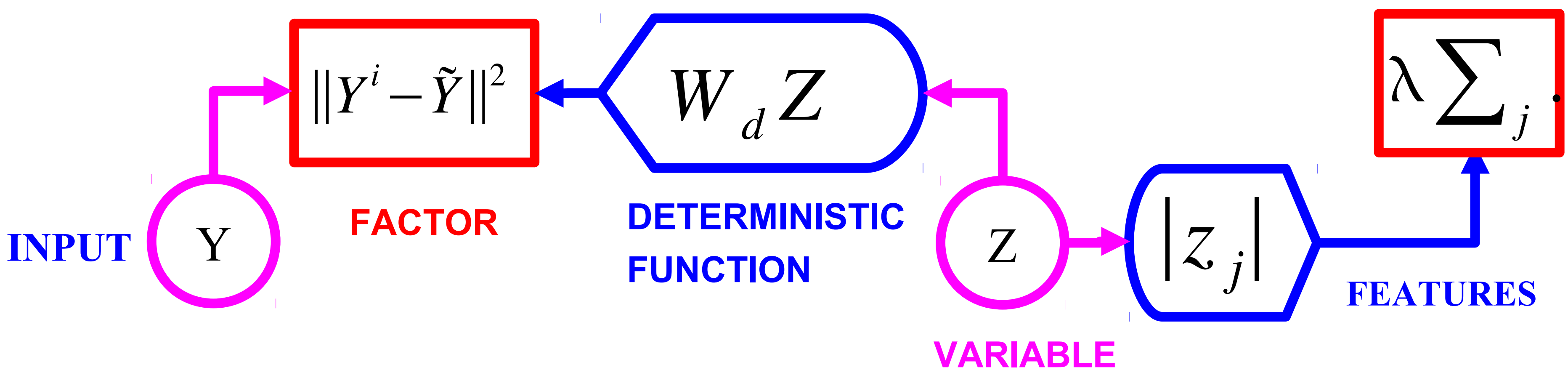


Y LeCun
MA Ranzato

[Olshausen & Field 1997]

- Sparse linear reconstruction
- Energy = reconstruction_error + code_prediction_error + code_sparsity

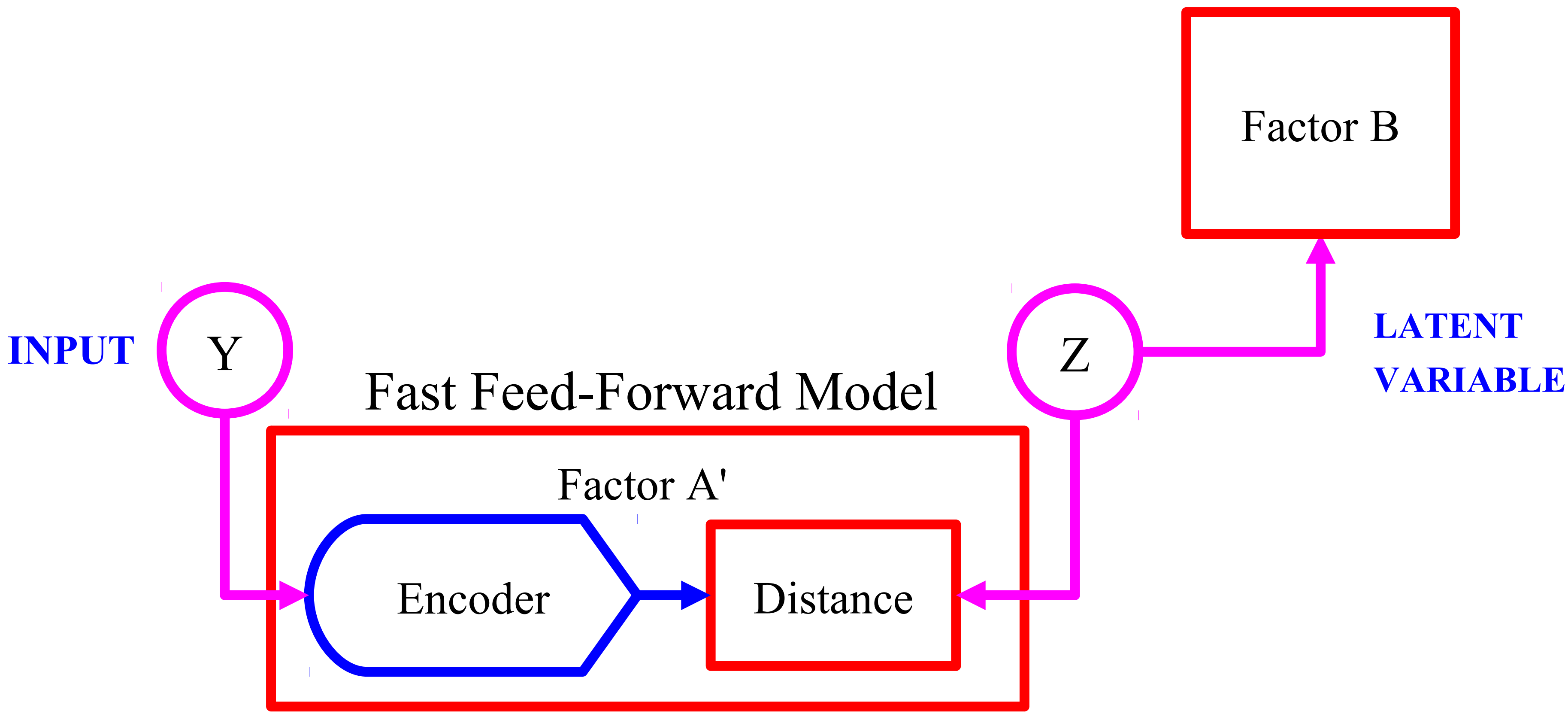
$$E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \lambda \sum_j |z_j|$$



- Inference is slow $Y \rightarrow \hat{Z} = argmin_Z E(Y, Z)$

Encoder Architecture

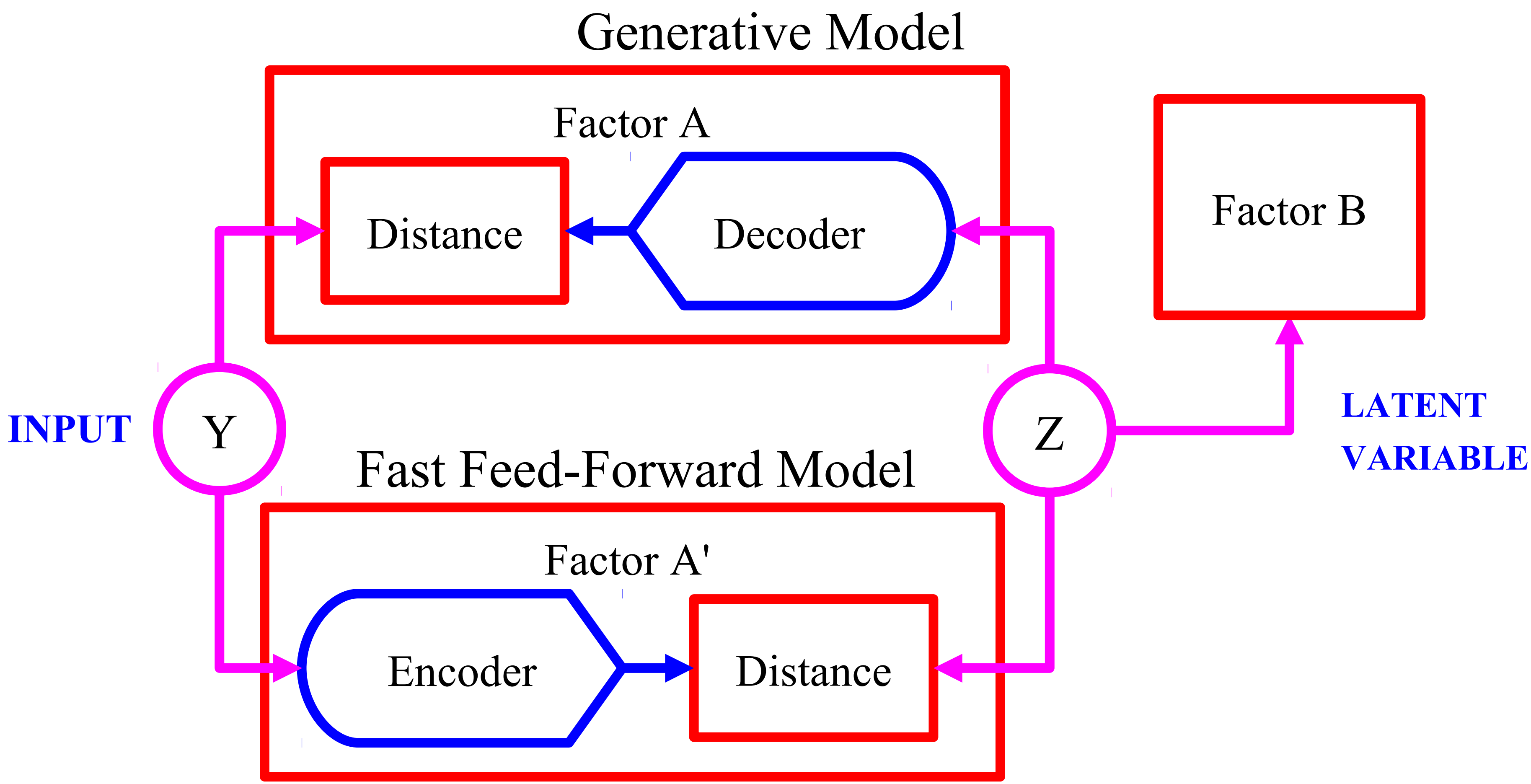
Examples: most ICA models, Product of Experts



Encoder-Decoder Architecture

[Kavukcuoglu, Ranzato, LeCun, rejected by every conference, 2008-2009]

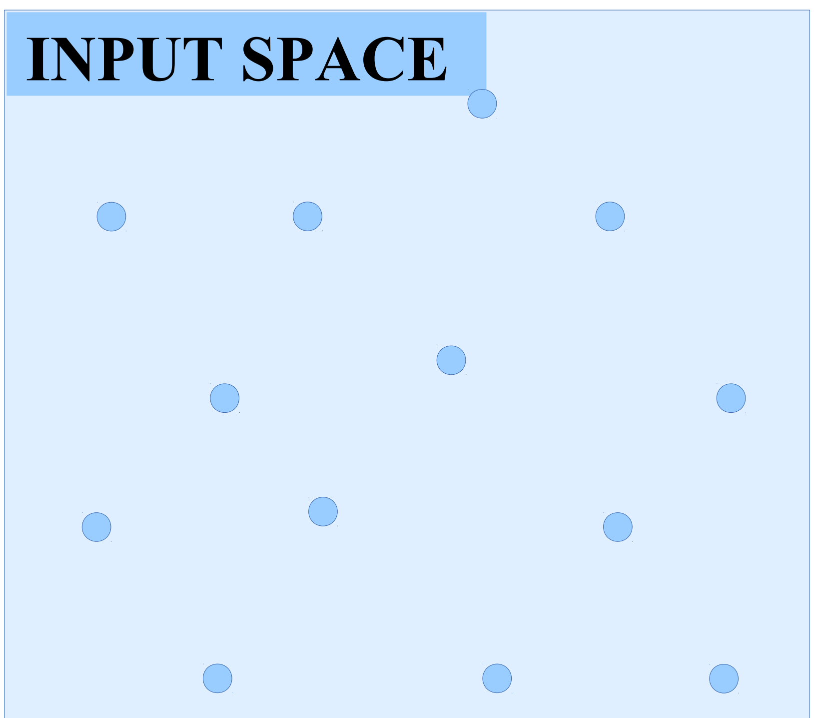
- Train a "simple" feed-forward function to predict the result of a complex optimization on the data points of interest

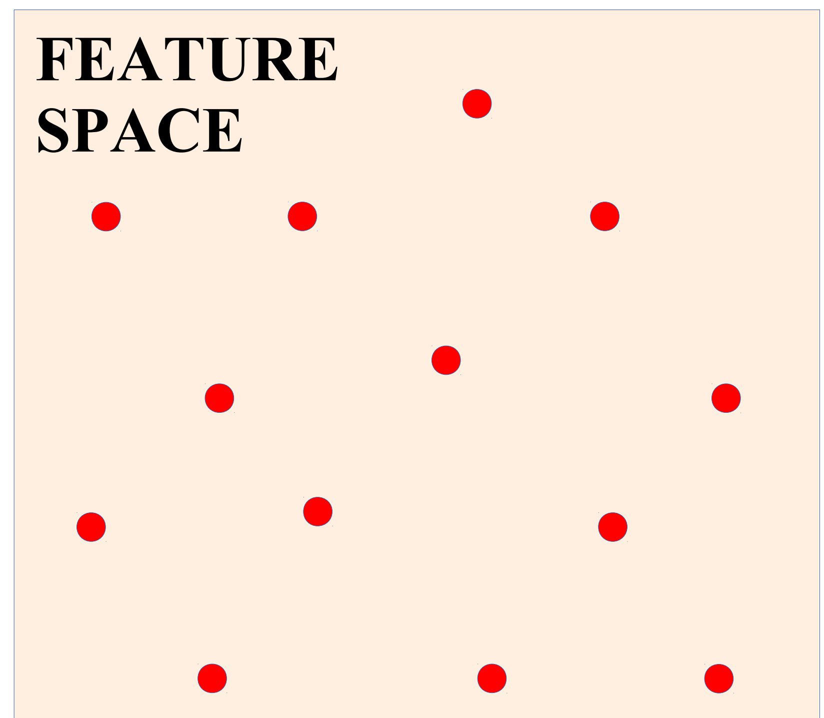


- 1. Find optimal Zi for all Yi; 2. Train Encoder to predict Zi from Yi

Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

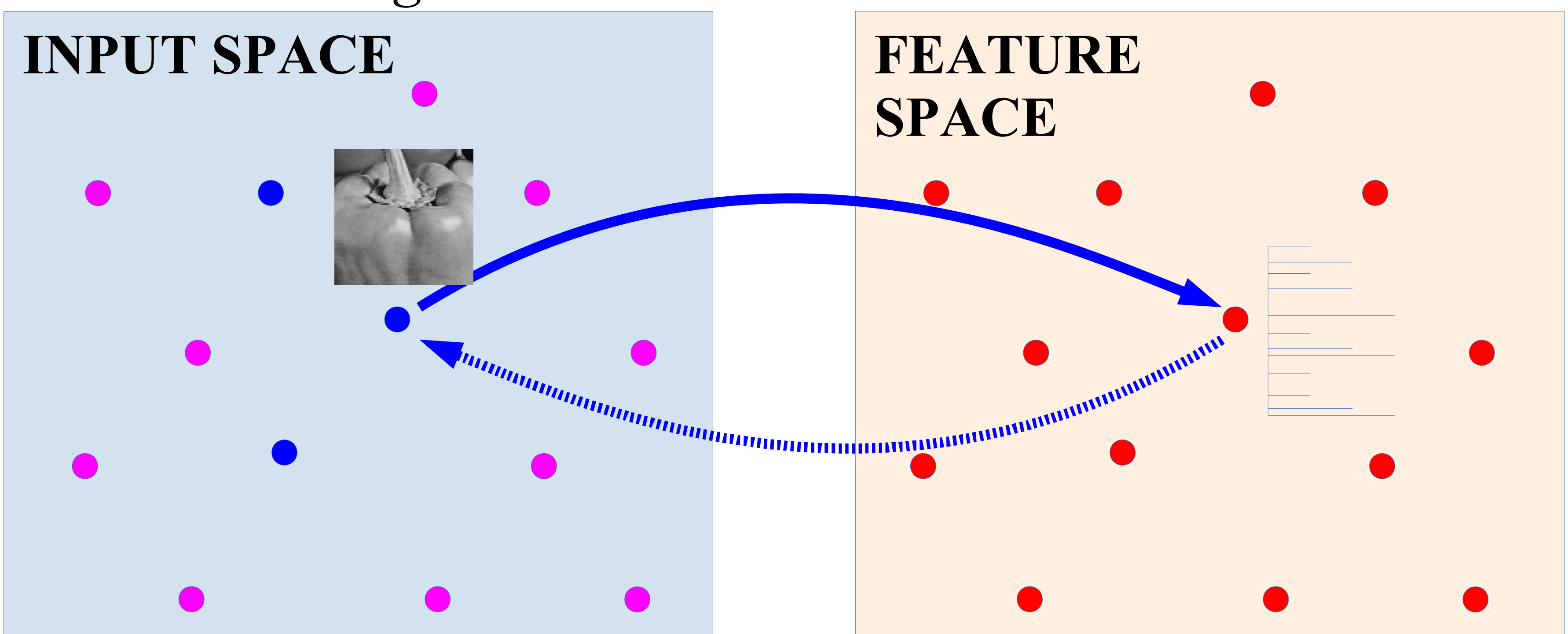




Why Limit the Information Content of the Code?

- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**

Training based on minimizing the reconstruction error over the training set

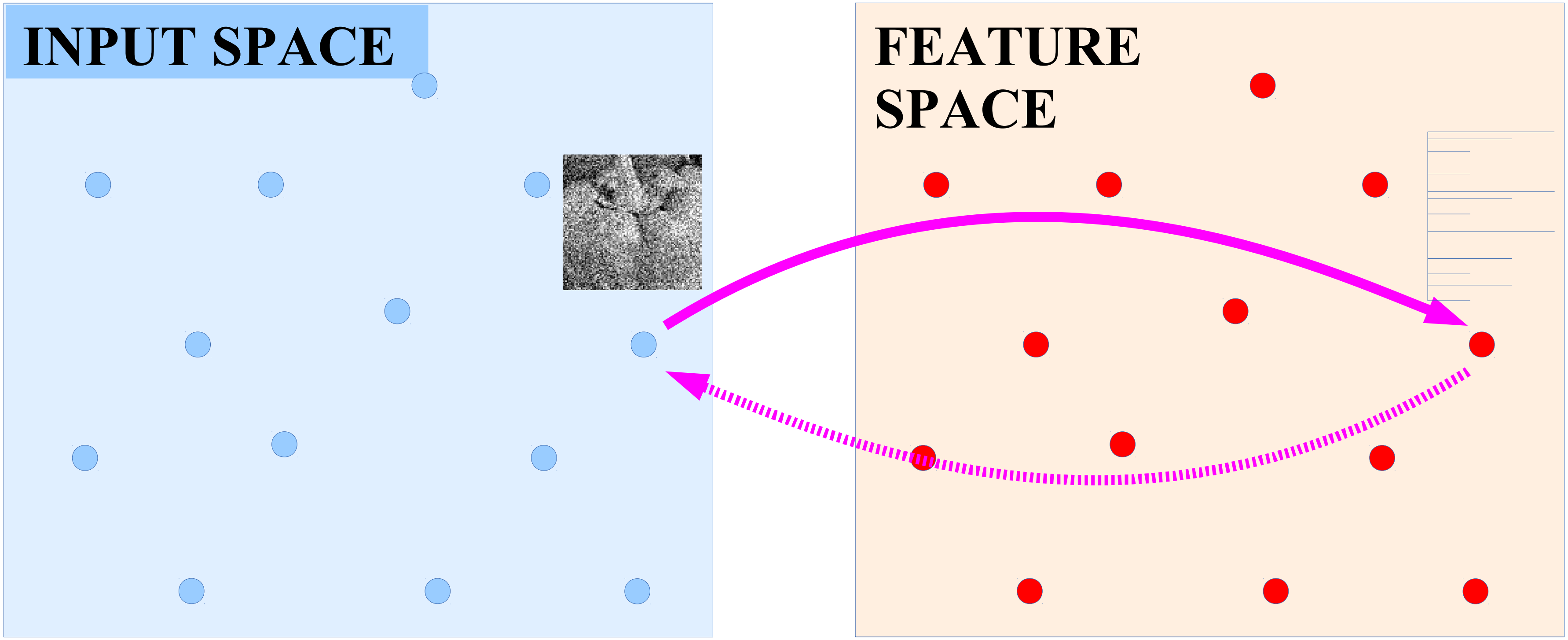


Why Limit the Information Content of the Code?

- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**

BAD: machine does not learn structure from training data!!

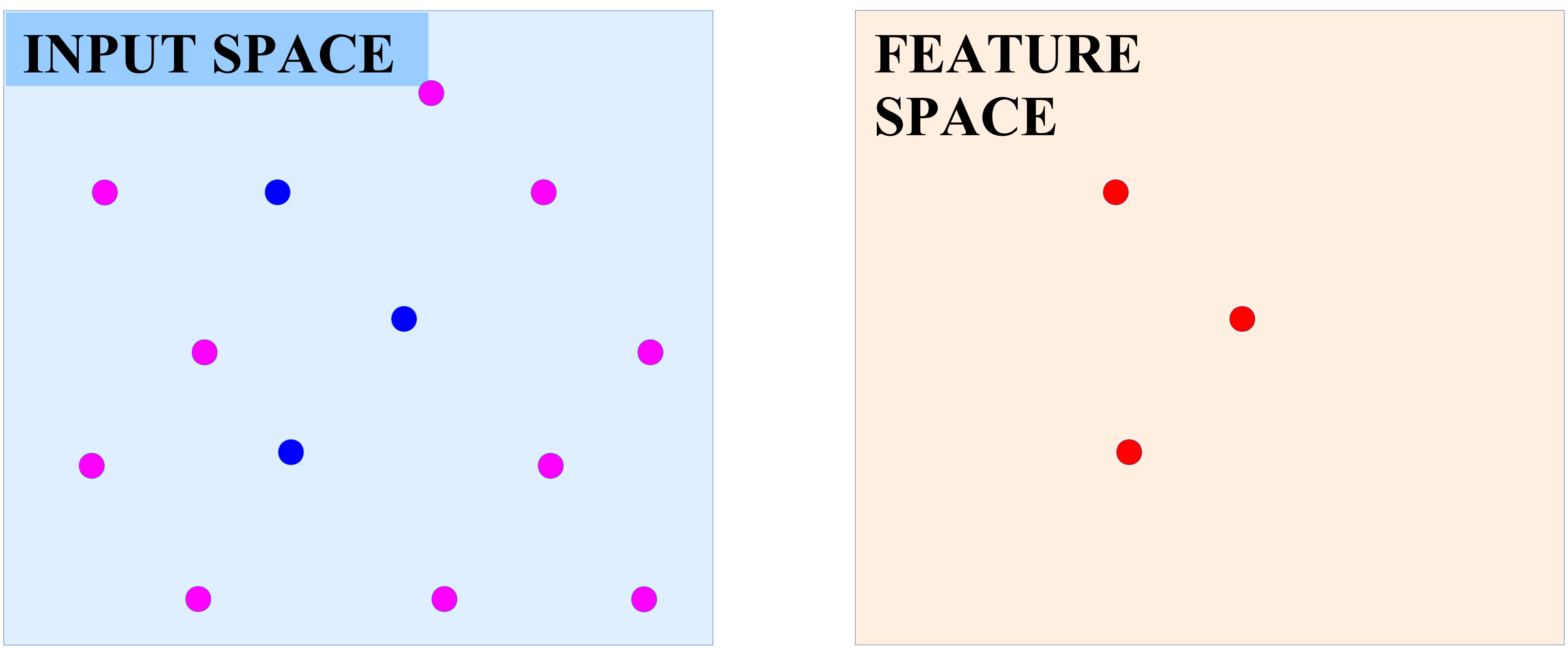
It just copies the data.



Why Limit the Information Content of the Code?

- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**

IDEA: reduce number of available codes.

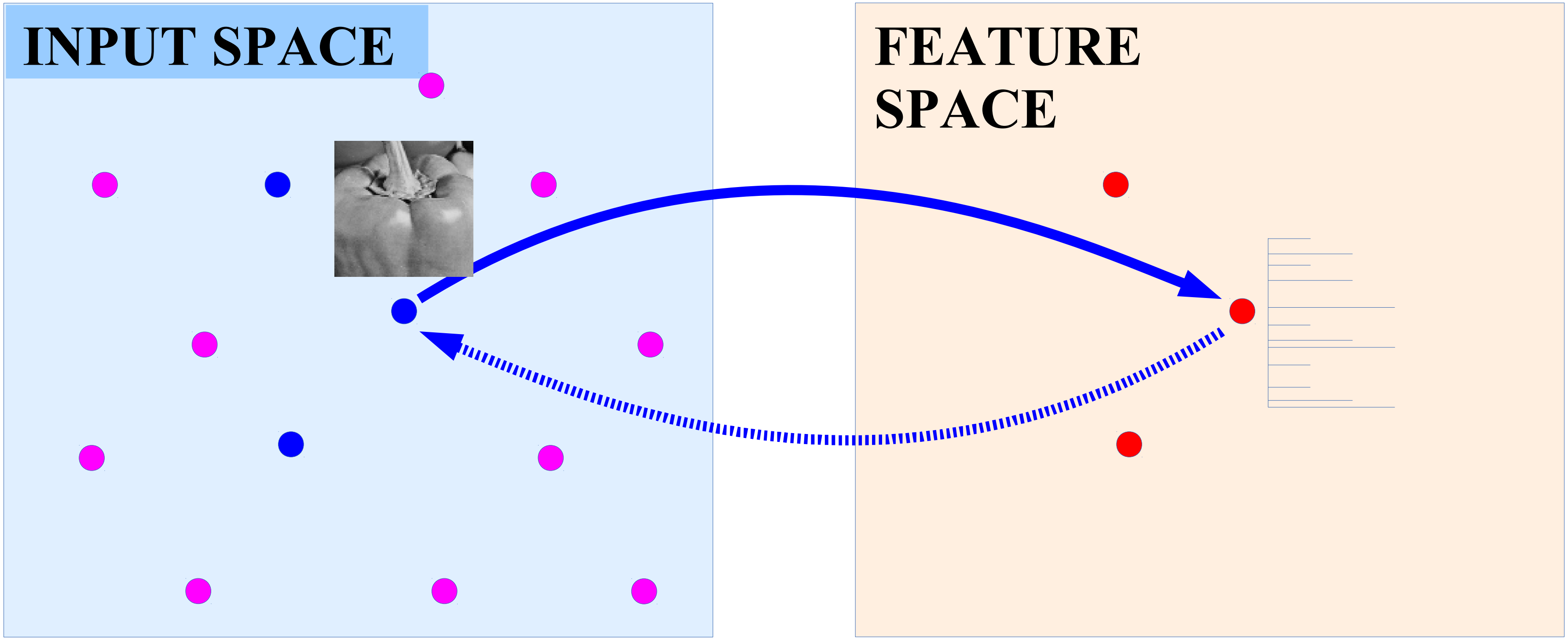


Why Limit the Information Content of the Code?

Y LeCun
MA Ranzato

- Training sample
- Input vector which is NOT a training sample
- Feature vector

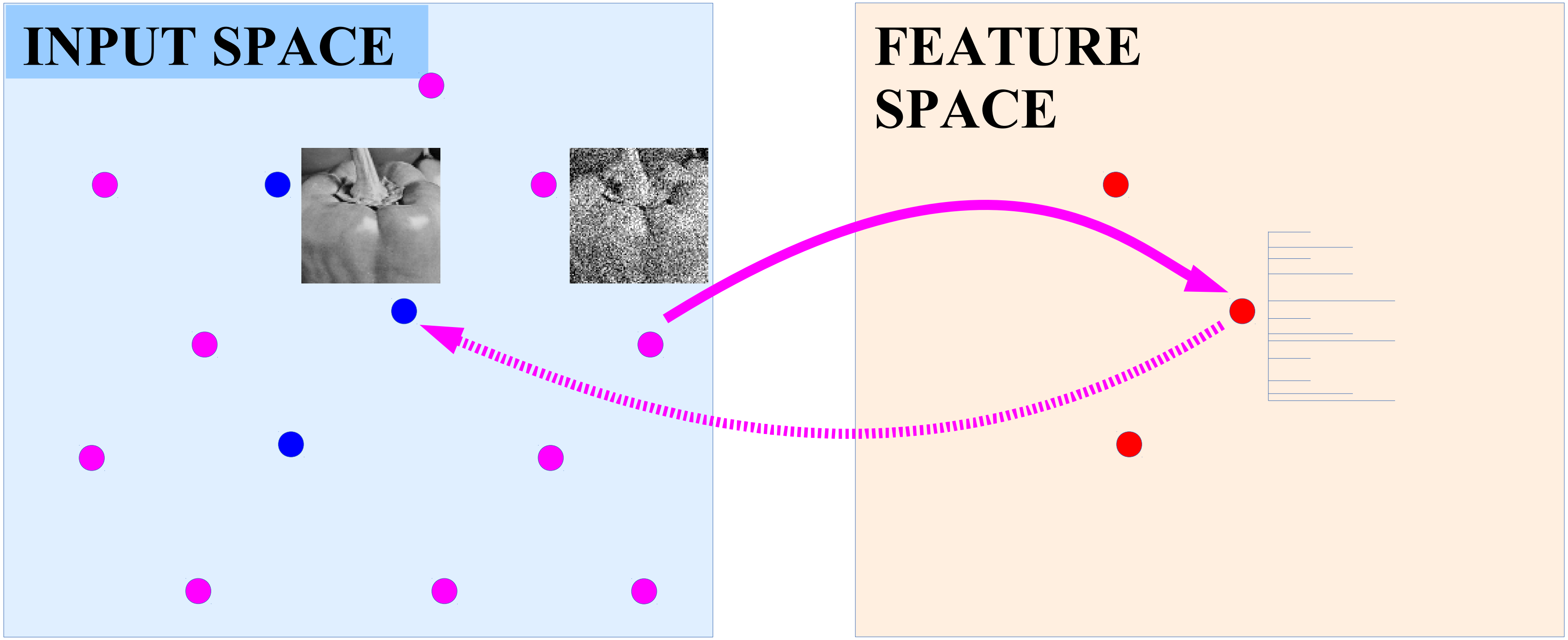
IDEA: reduce number of available codes.



Why Limit the Information Content of the Code?

- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**

IDEA: reduce number of available codes.



Predictive Sparse Decomposition (PSD): sparse auto-encoder

Y LeCun
MA Ranzato

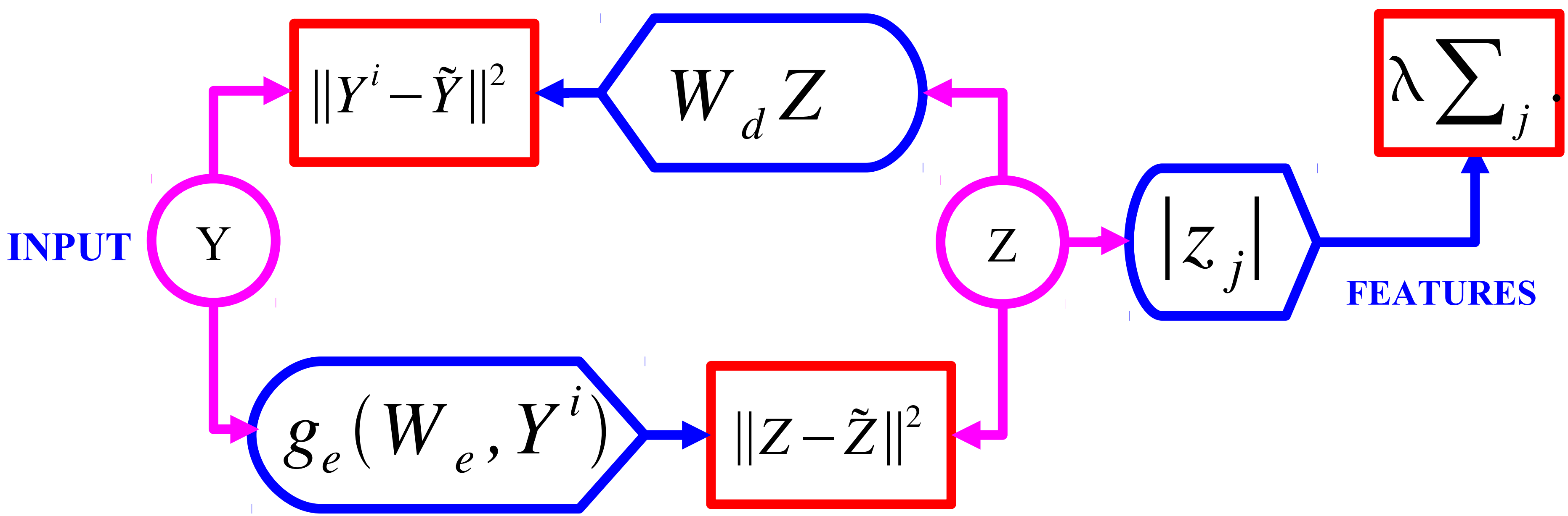
[Kavukcuoglu, Ranzato, LeCun, 2008 → arXiv:1010.3467],

Prediction the optimal code with a trained encoder

Energy = reconstruction_error + code_prediction_error + code_sparsity

$$E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \|Z - g_e(W_e, Y^i)\|^2 + \lambda \sum_j |z_j|$$

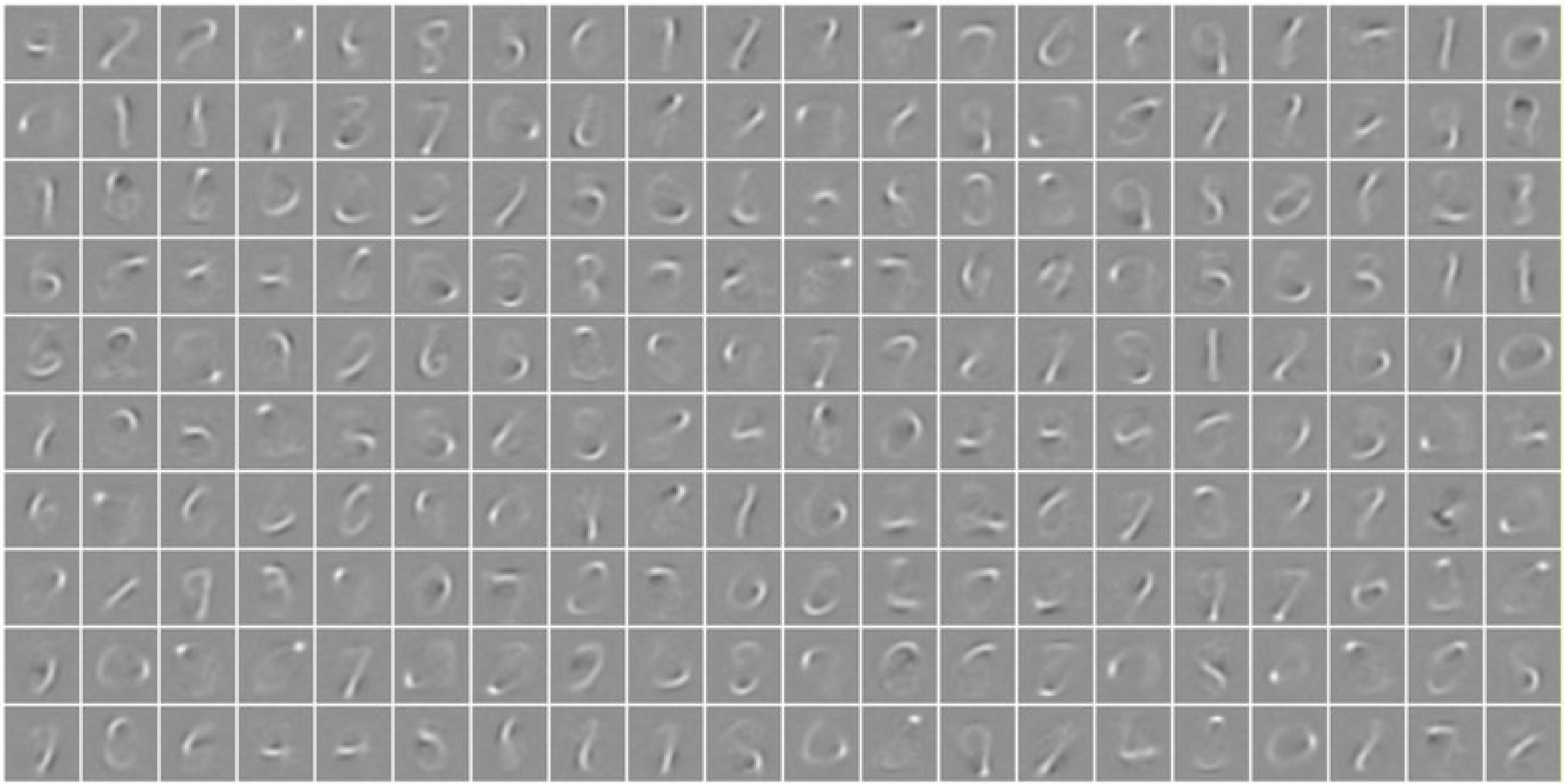
$$g_e(W_e, Y^i) = shrinkage(W_e Y^i)$$



PSD: Basis Functions on MNIST

Y LeCun
MA Ranzato

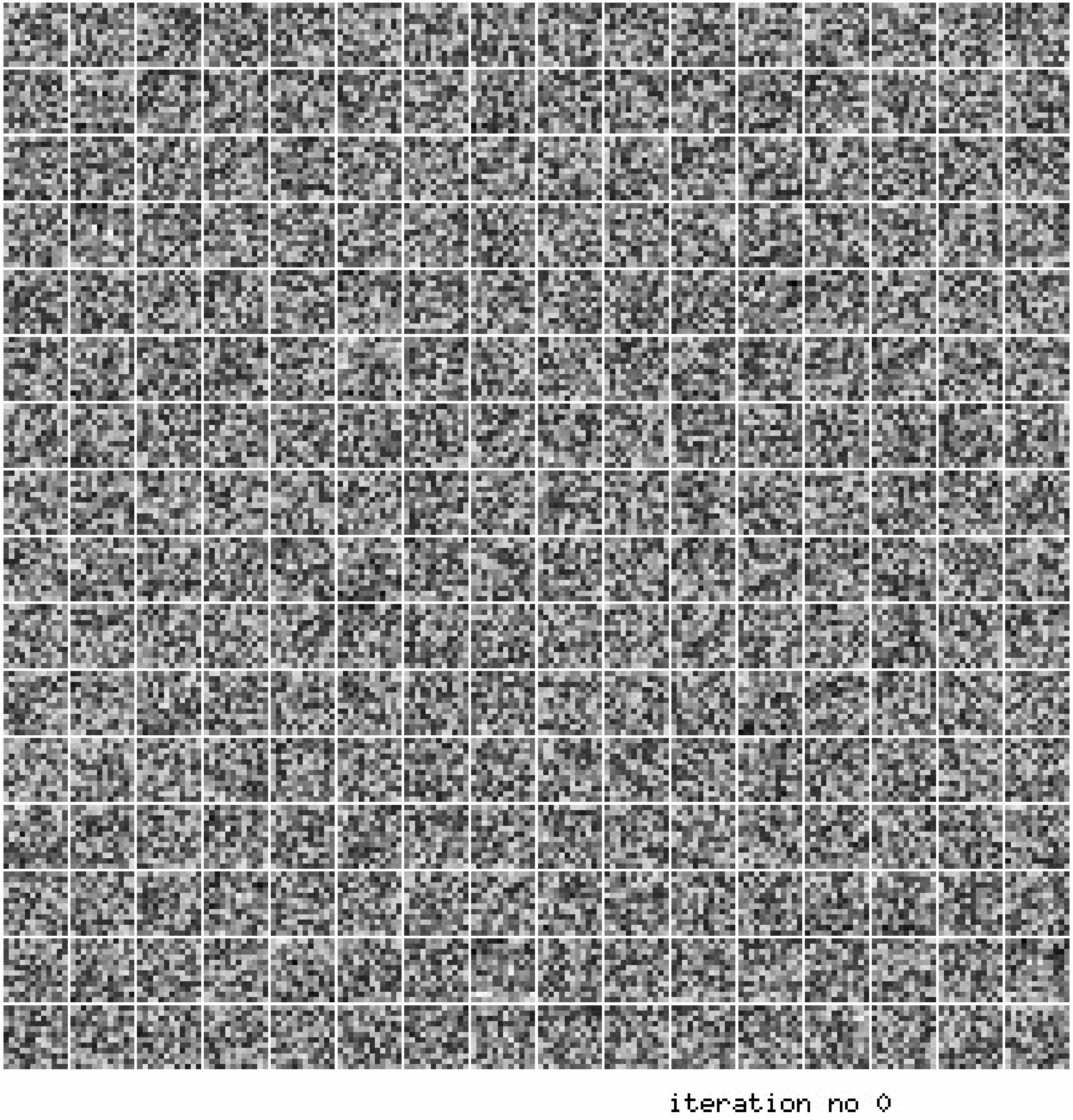
- Basis functions (and encoder matrix) are digit parts



Predictive Sparse Decomposition (PSD): Training

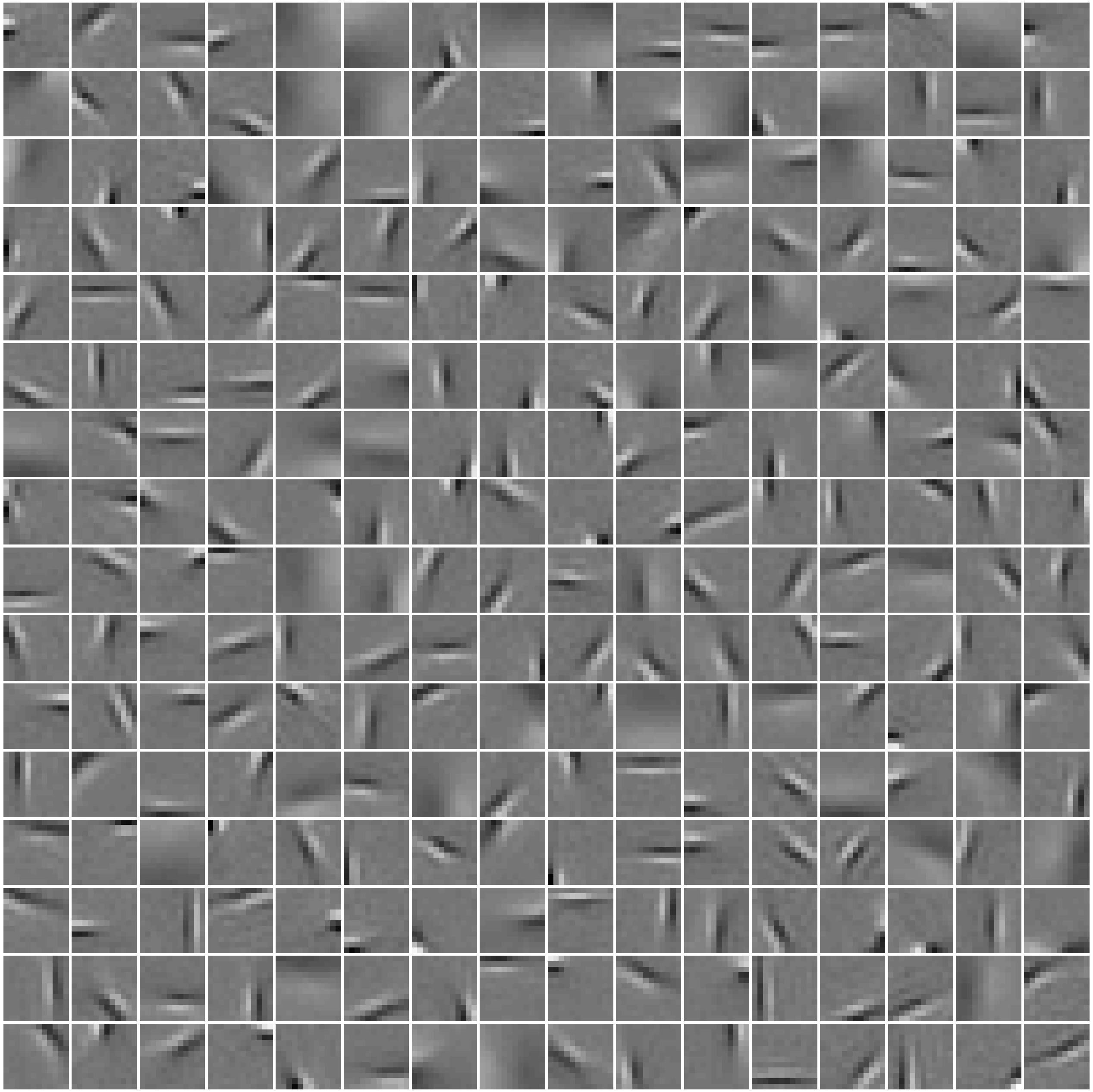
Y LeCun
MA Ranzato

- **Training on natural images patches.**
  - 12X12
  - 256 basis functions



Learned Features on natural patches: V1-like receptive fields

Y LeCun
MA Ranzato

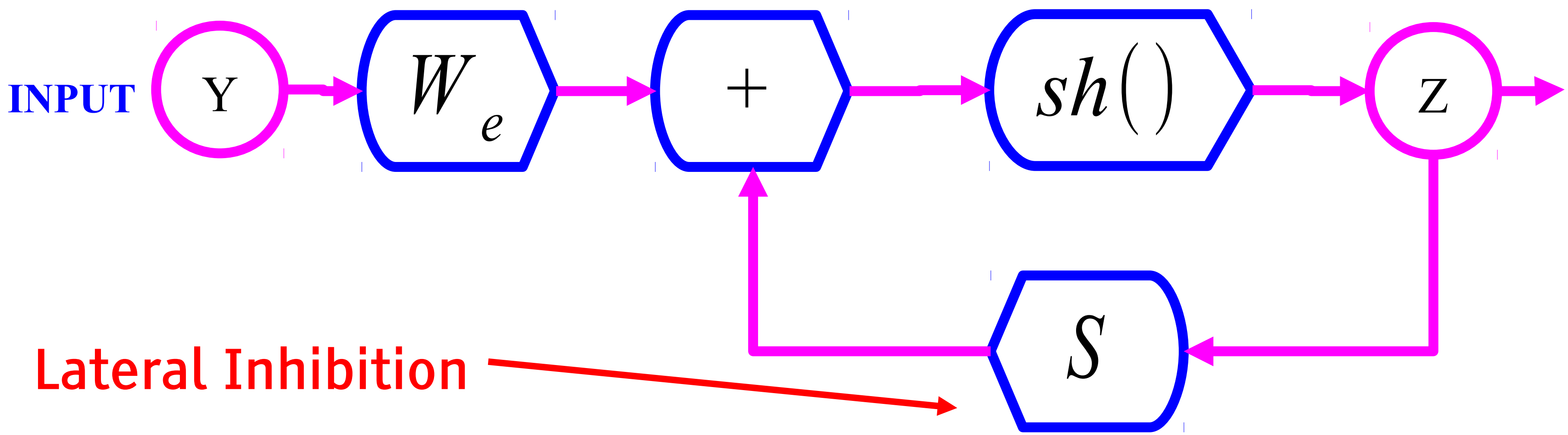


Better Idea: Give the "right" structure to the encoder

Y LeCun
MA Ranzato

- **ISTA/FISTA: iterative algorithm that converges to optimal sparse code**

[Gregor & LeCun, ICML 2010], [Bronstein et al. ICML 2012], [Rolfe & LeCun ICLR 2013]



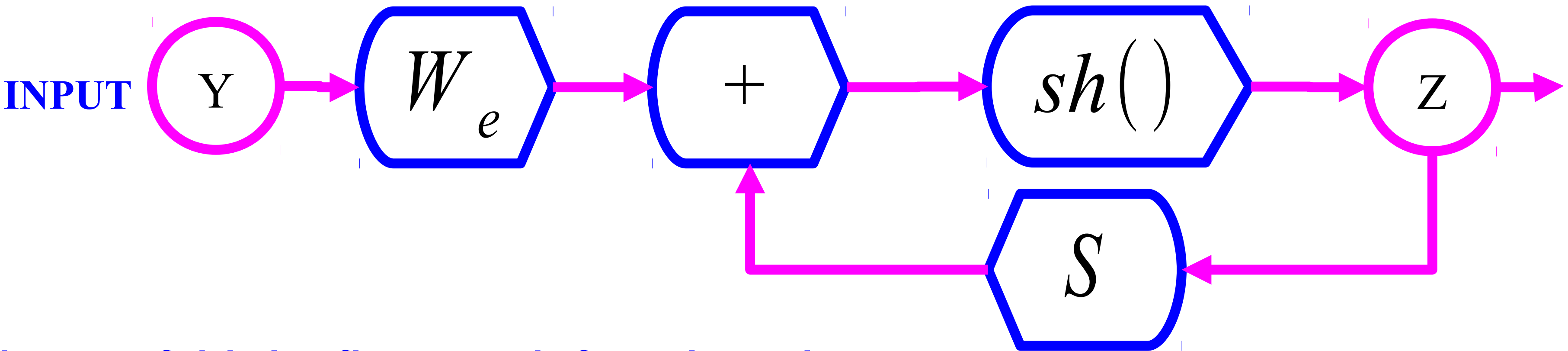
$$Z(t+1) = \text{Shrinkage}_{\lambda/L}\left[Z(t) - \frac{1}{L}W_d^T(W_d Z(t) - Y)\right]$$

$$Z(t+1) = \text{Shrinkage}_{\lambda/L}\left[W_e^T Y + SZ(t)\right]; \quad W_e = \frac{1}{L}W_d; \quad S = I - \frac{1}{L}W_d^T W_d$$

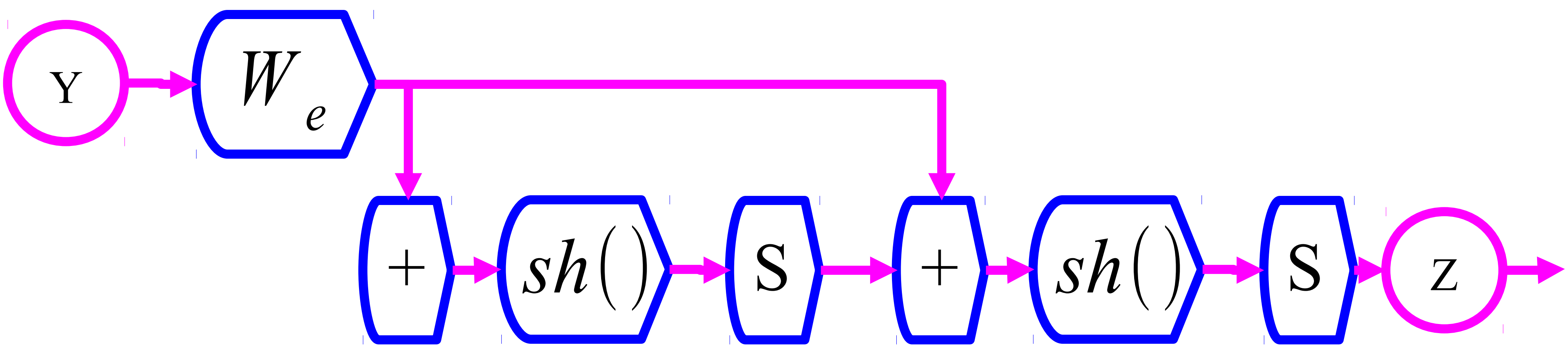
LISTA: Train We and S matrices to give a good approximation quickly

Y LeCun
MA Ranzato

- Think of the FISTA flow graph as a recurrent neural net where We and S are trainable parameters



- Time-Unfold the flow graph for K iterations
- Learn the We and S matrices with "backprop-through-time"
- Get the best approximate solution within K iterations



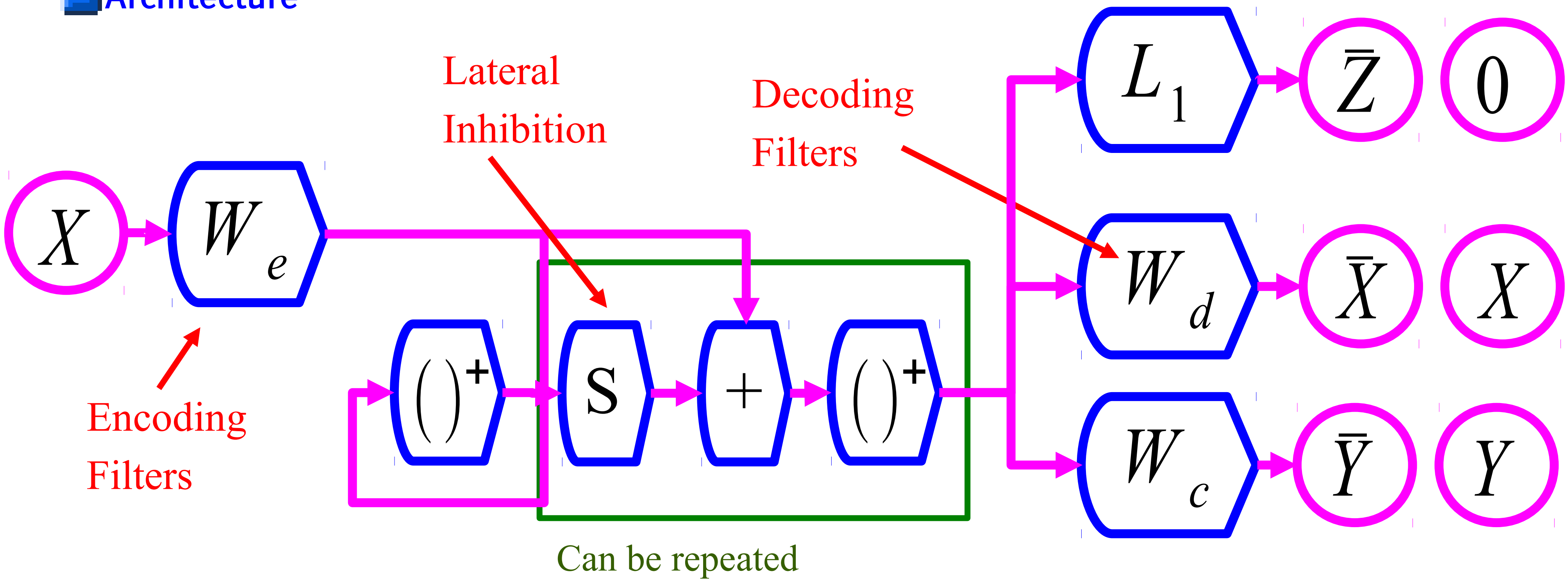
Learning ISTA (LISTA) vs ISTA/FISTA

LISTA with partial mutual inhibition matrix

Learning Coordinate Descent (LcoD): faster than LISTA

Discriminative Recurrent Sparse Auto-Encoder (DrSAE)

- Architecture



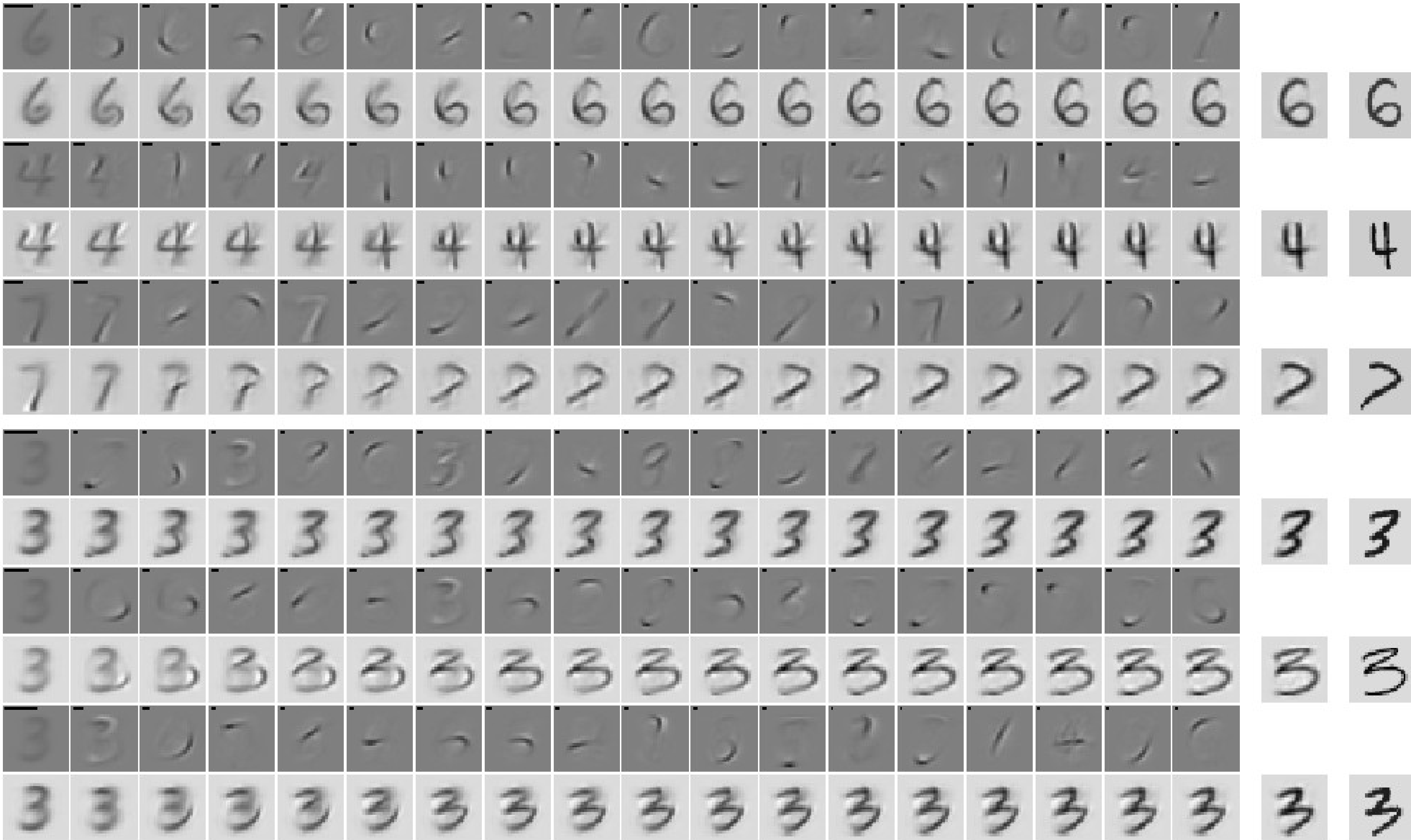
- Rectified linear units
- Classification loss: cross-entropy
- Reconstruction loss: squared error
- Sparsity penalty: L1 norm of last hidden layer
- Rows of Wd and columns of We constrained in unit sphere

[Rolfe & LeCun ICLR 2013]

DrSAE Discovers manifold structure of handwritten digits

Y LeCun
MA Ranzato

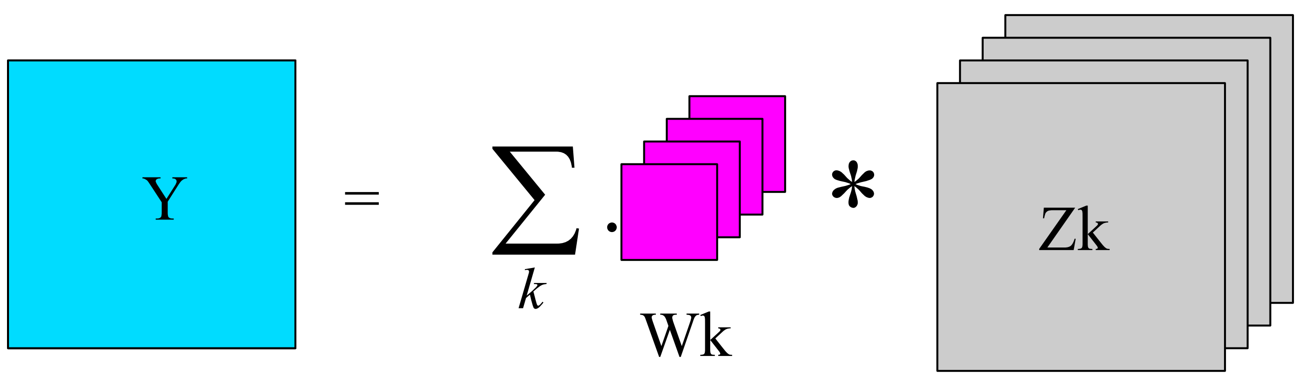
- Image = prototype + sparse sum of “parts” (to move around the manifold)



Convolutional Sparse Coding

Y LeCun
MA Ranzato

- **Replace the dot products with dictionary element by convolutions.**
  - Input Y is a full image
  - Each code component Zk is a feature map (an image)
  - Each dictionary element is a convolution kernel
- **Regular sparse coding** $$E(Y,Z) = ||Y - \sum_k W_k Z_k||^2 + \alpha \sum_k |Z_k|$$
- **Convolutional S.C.** $$E(Y,Z) = ||Y - \sum_k W_k * Z_k||^2 + \alpha \sum_k |Z_k|$$



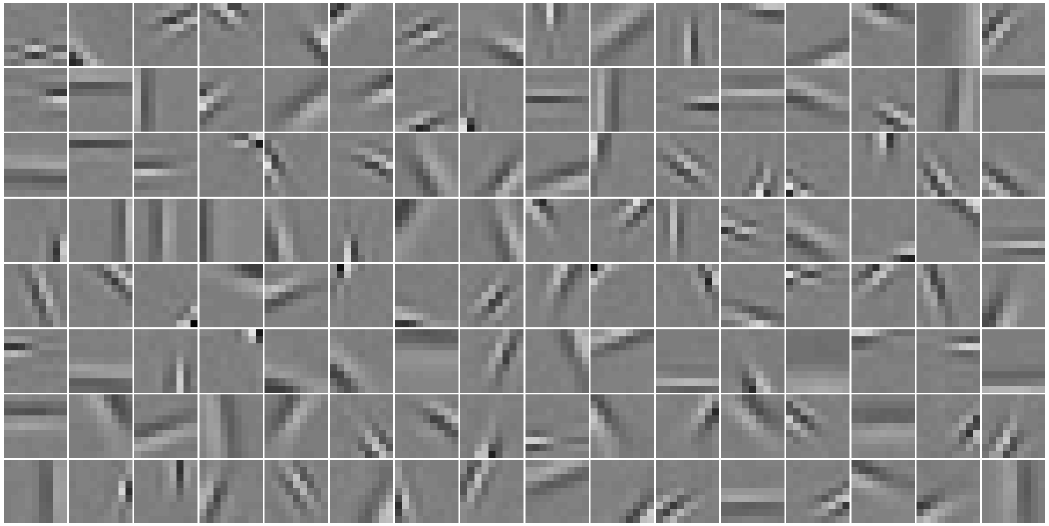
"deconvolutional networks" [Zeiler, Taylor, Fergus CVPR 2010]

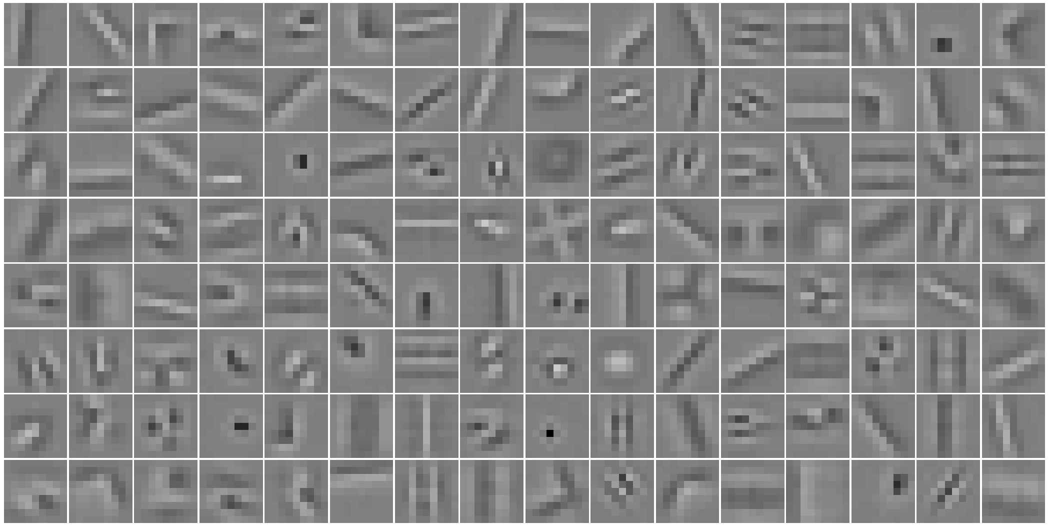
Convolutional PSD: Encoder with a soft sh() Function

Convolutional Formulation

- Extend sparse coding from ***PATCH*** to ***IMAGE***

$$\mathcal{L}(x,z,\mathcal{D}) = \frac{1}{2}||x - \sum_{k=1}^{K} \mathcal{D}_k * z_k||_2^2 + \sum_{k=1}^{K} ||z_k - f(W^k * x)||_2^2 + |z|_1$$



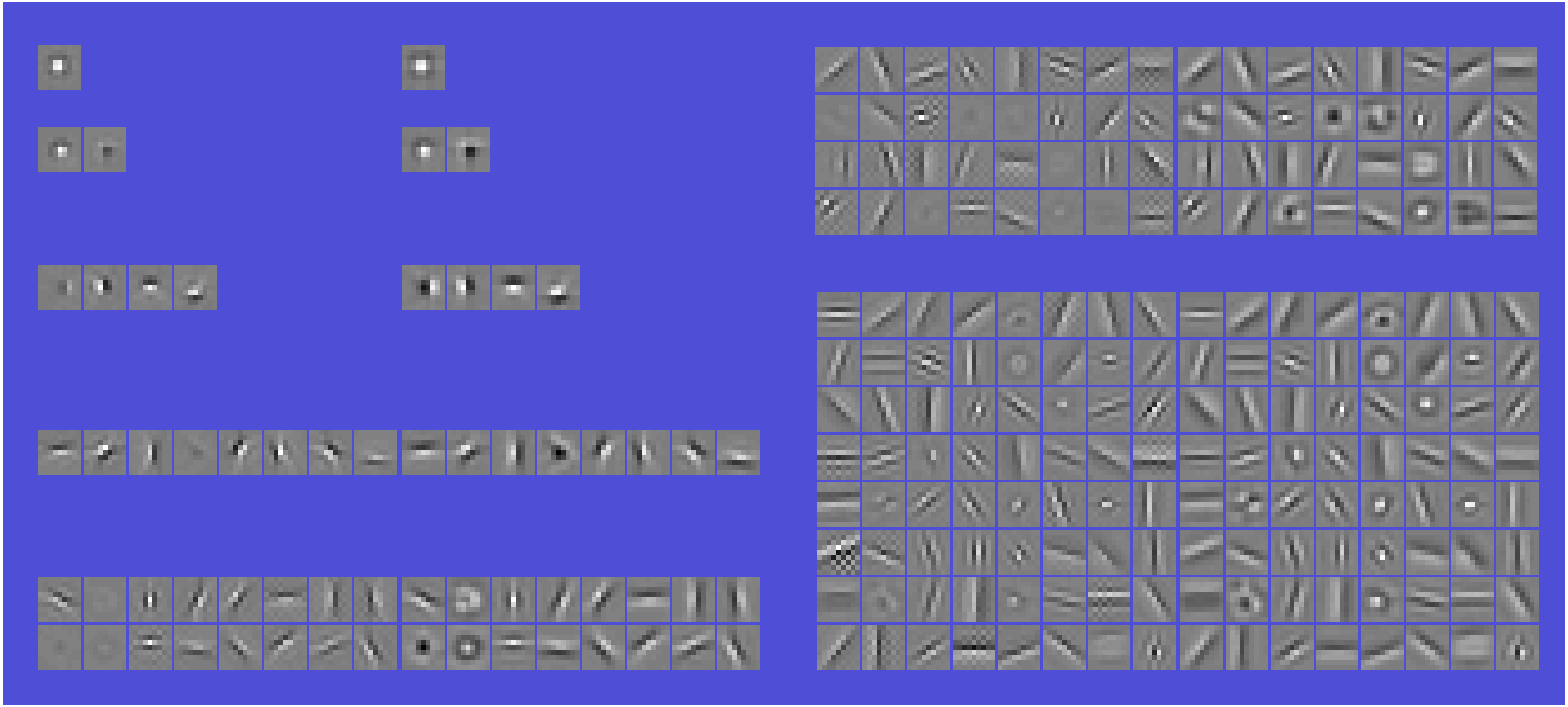


- ***PATCH*** based learning
- ***CONVOLUTIONAL*** learning

Convolutional Sparse Auto-Encoder on Natural Images

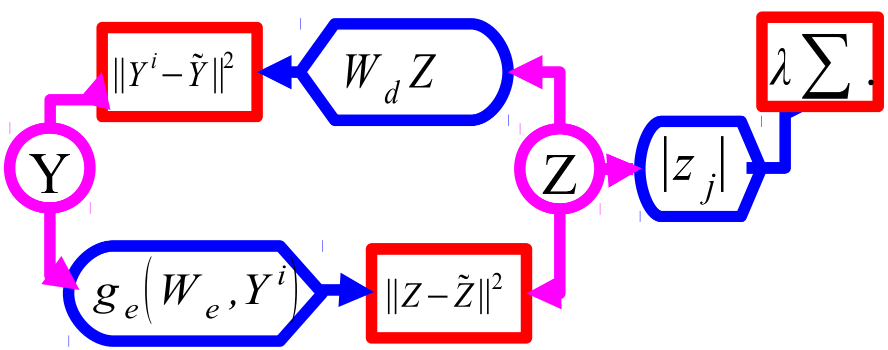
Y LeCun
MA Ranzato

- Filters and Basis Functions obtained with 1, 2, 4, 8, 16, 32, and 64 filters.



Using PSD to Train a Hierarchy of Features

- Phase 1: train first layer using PSD

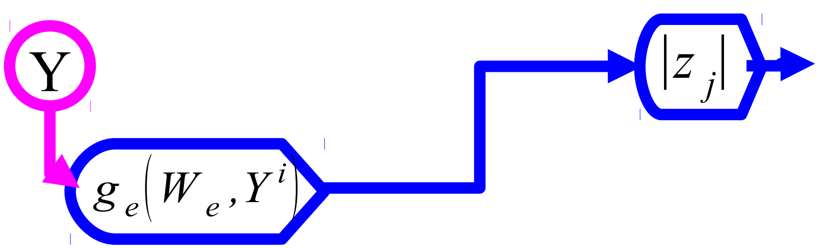


FEATURES

Using PSD to Train a Hierarchy of Features

Y LeCun
MA Ranzato

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor

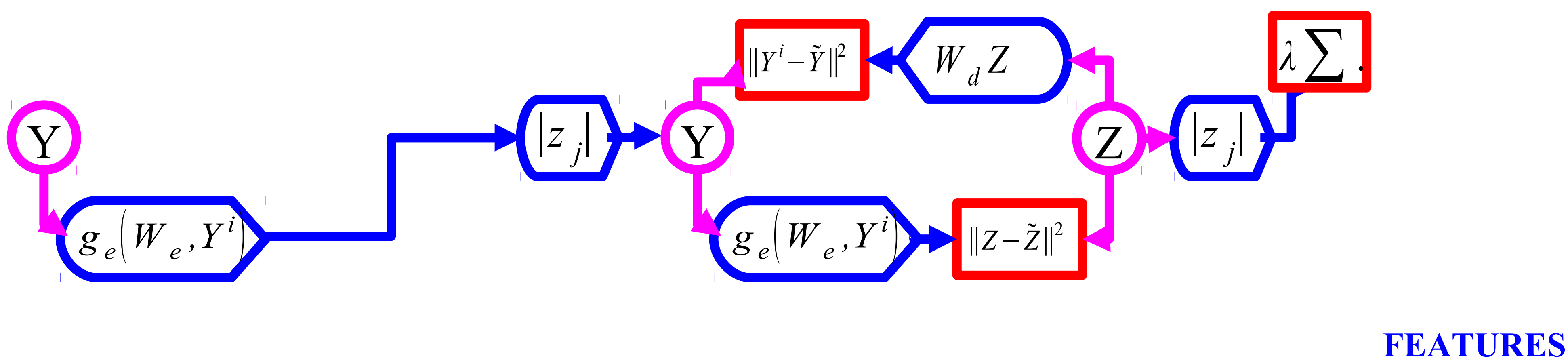


FEATURES

Using PSD to Train a Hierarchy of Features

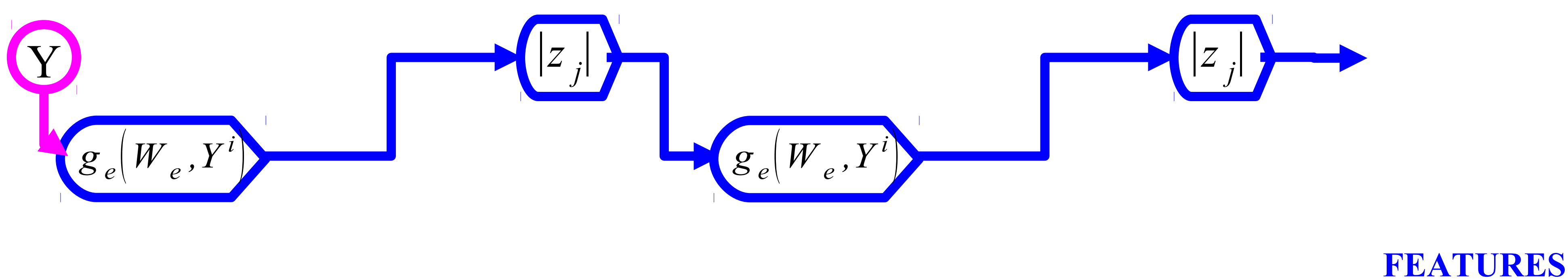
Y LeCun
MA Ranzato

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD



Using PSD to Train a Hierarchy of Features

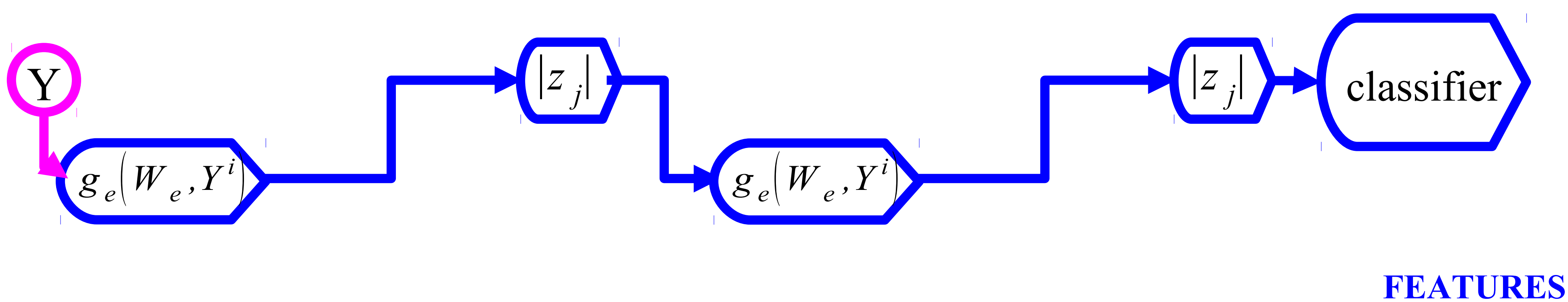
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor



Using PSD to Train a Hierarchy of Features

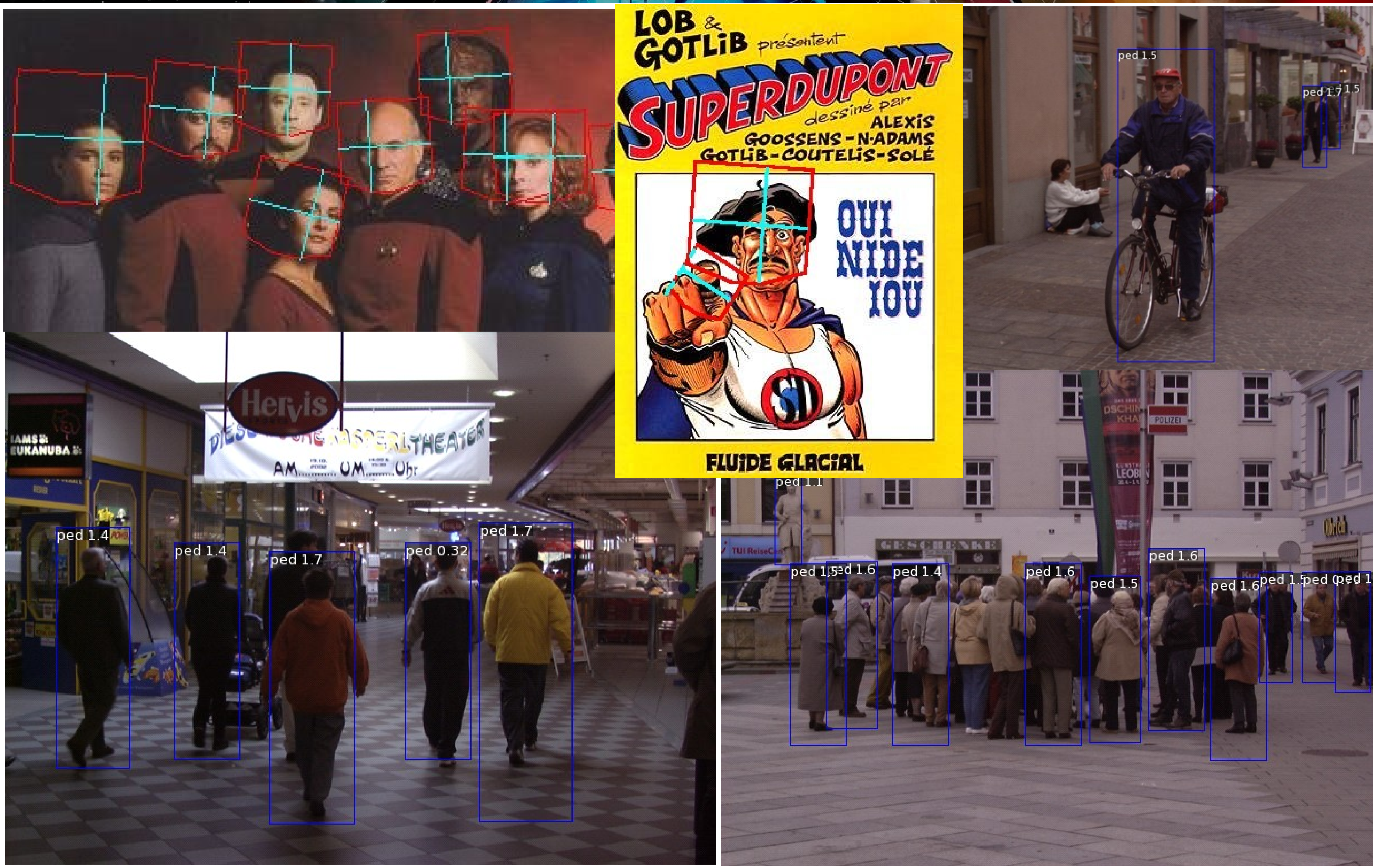
Y LeCun
MA Ranzato

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor
- Phase 5: train a supervised classifier on top
- Phase 6 (optional): train the entire system with supervised back-propagation



Pedestrian Detection, Face Detection

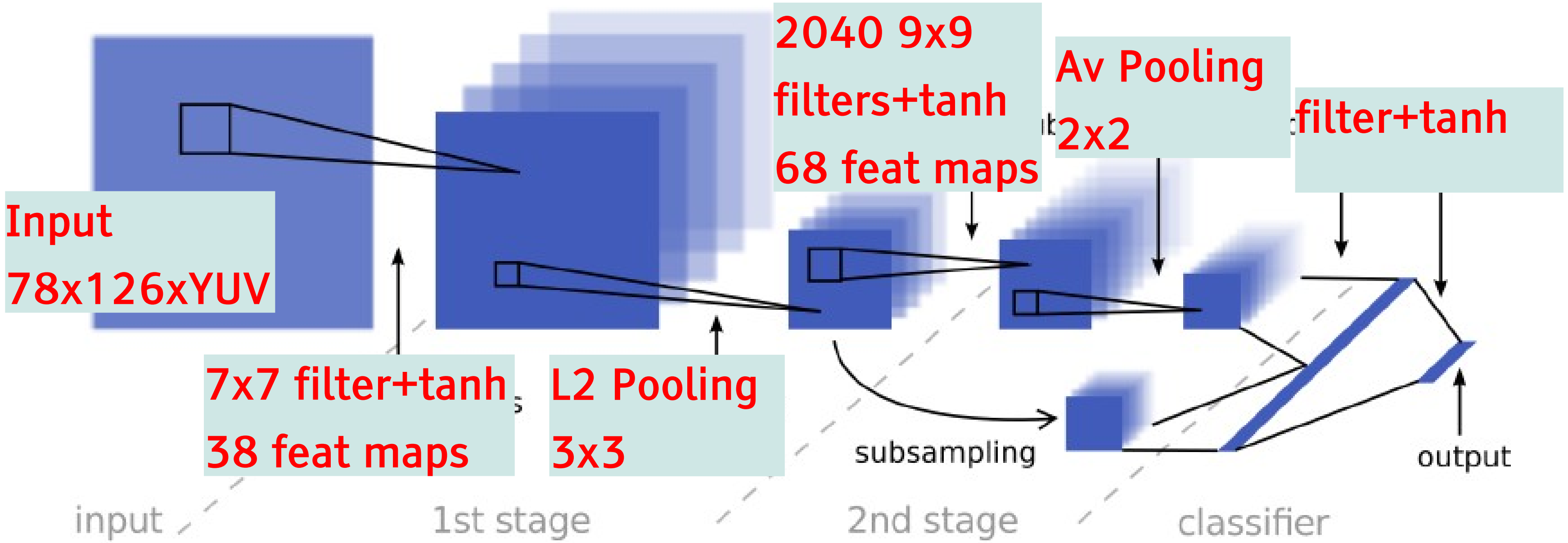
Y LeCun
MA Ranzato



[Osadchy,Miller LeCun JMLR 2007],[Kavukcuoglu et al. NIPS 2010] [Sermanet et al. CVPR 2013]

ConvNet Architecture with Multi-Stage Features

- **Feature maps from all stages are pooled/subsampled and sent to the final classification layers**
  - Pooled low-level features: good for textures and local motifs
  - High-level features: good for "gestalt" and global shape

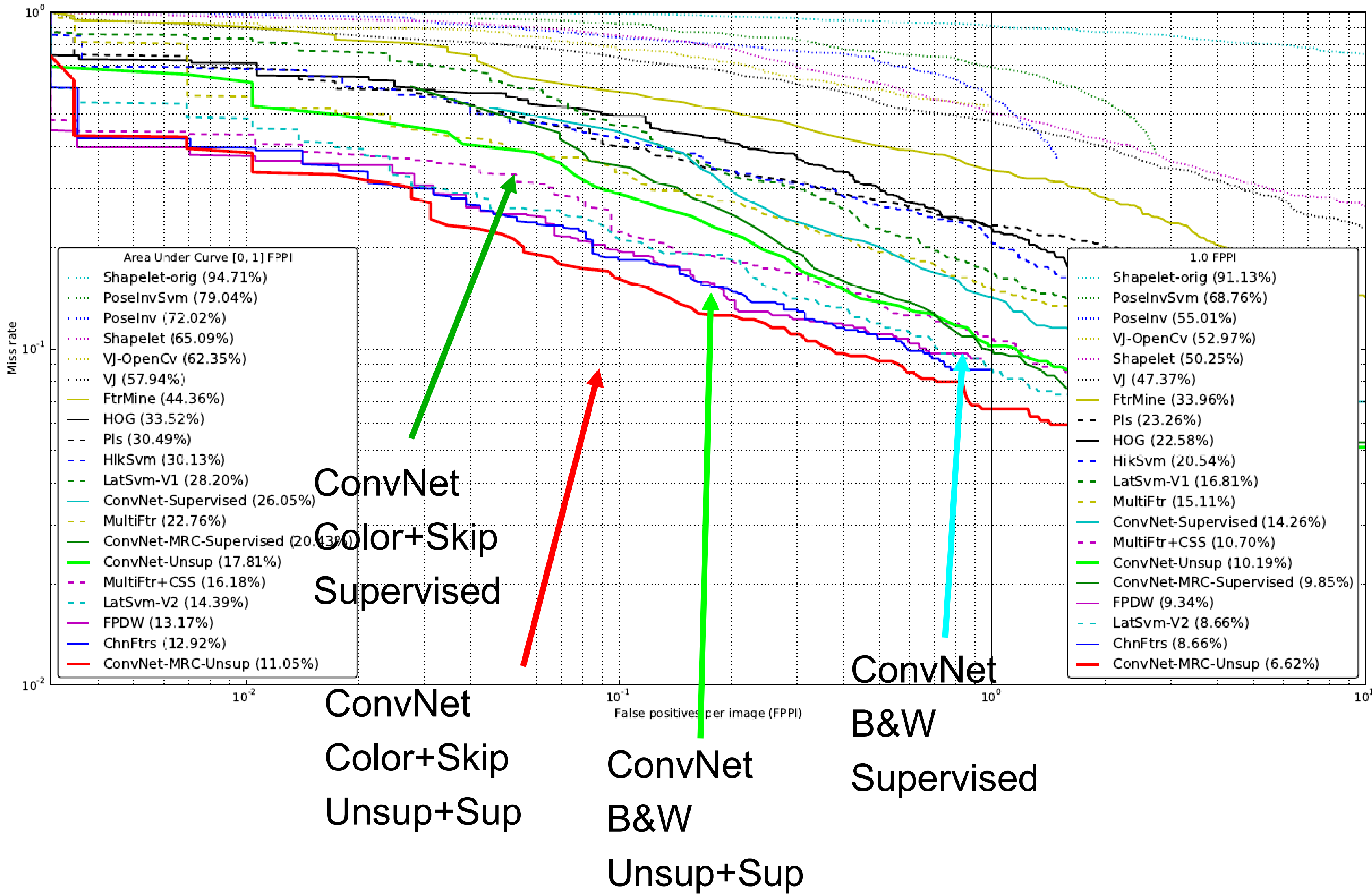


Task	Single-Stage features	Multi-Stage features	Improvement %
Pedestrians detection (INRIA)	14.26%	9.85%	31%
Traffic Signs classification (GTSRB) [33]	1.80%	0.83%	54%
House Numbers classification (SVHN) [32]	5.54%	5.36%	3.2%

[Sermanet, Chintala, LeCun CVPR 2013]

Pedestrian Detection: INRIA Dataset. Miss rate vs false positives

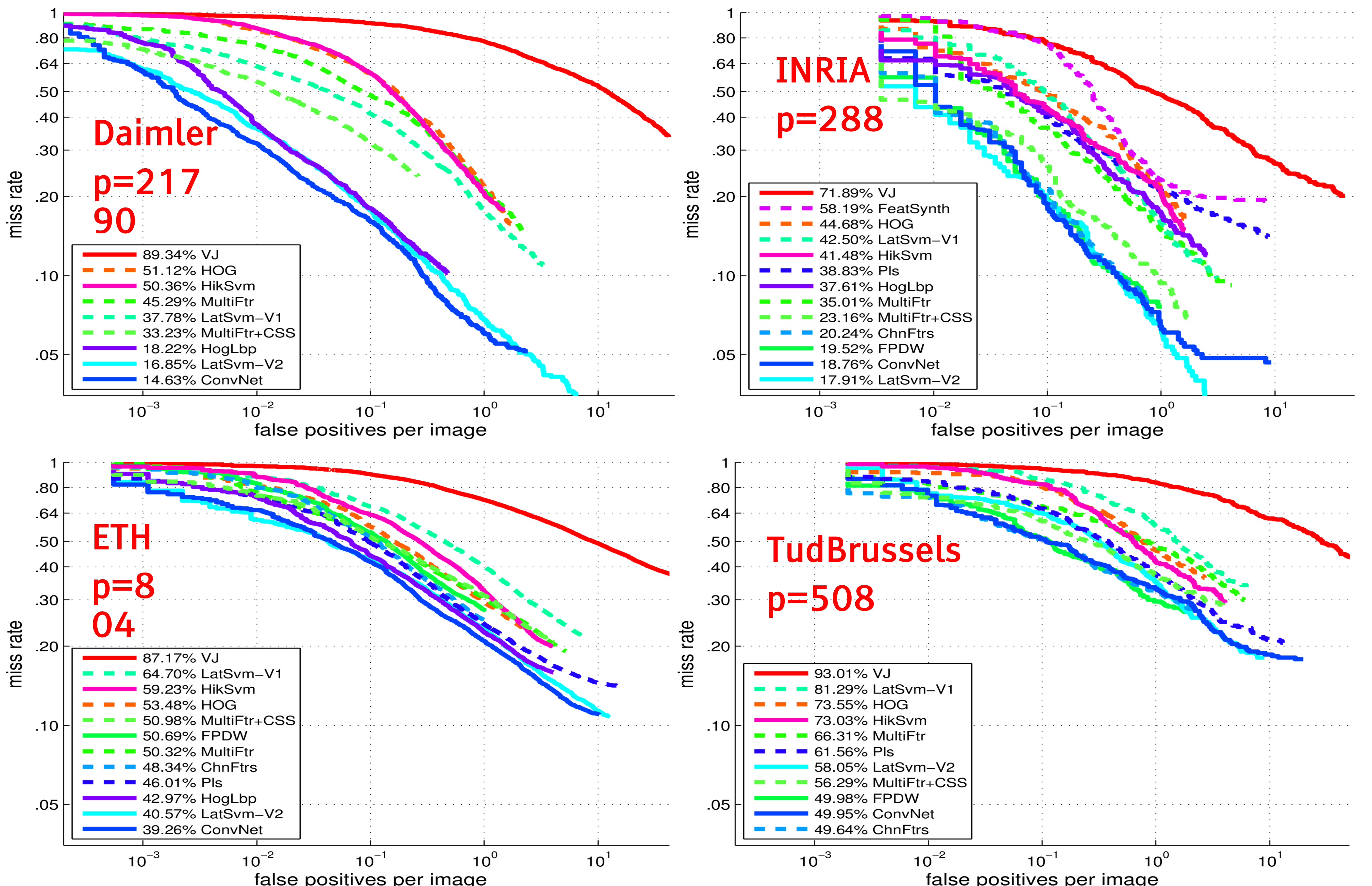
Y LeCun
MA Ranzato



[Kavukcuoglu et al. NIPS 2010] [Sermanet et al. ArXiv 2012]

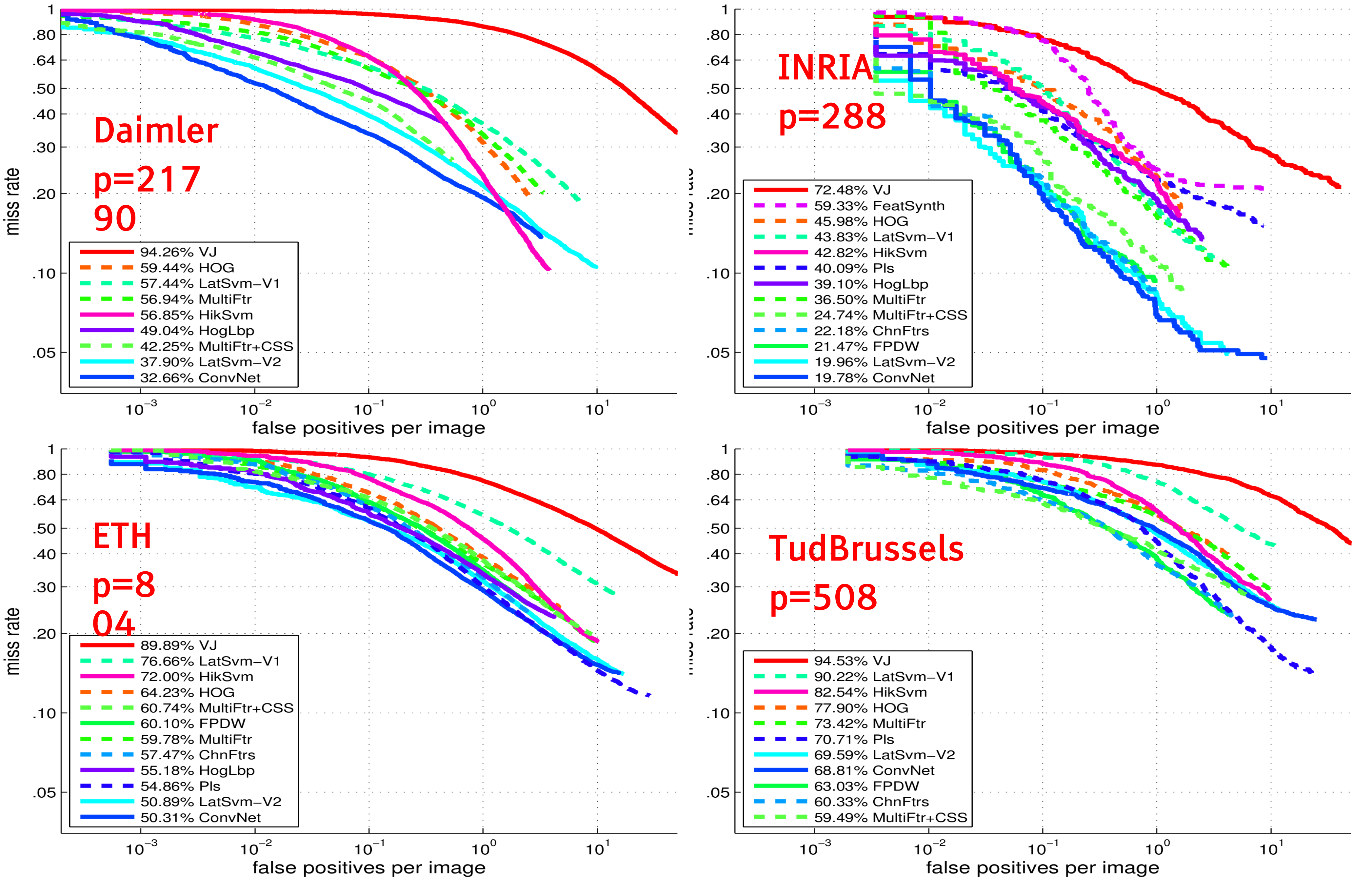
Results on "Near Scale" Images (>80 pixels tall, no occlusions)

Y LeCun
MA Ranzato



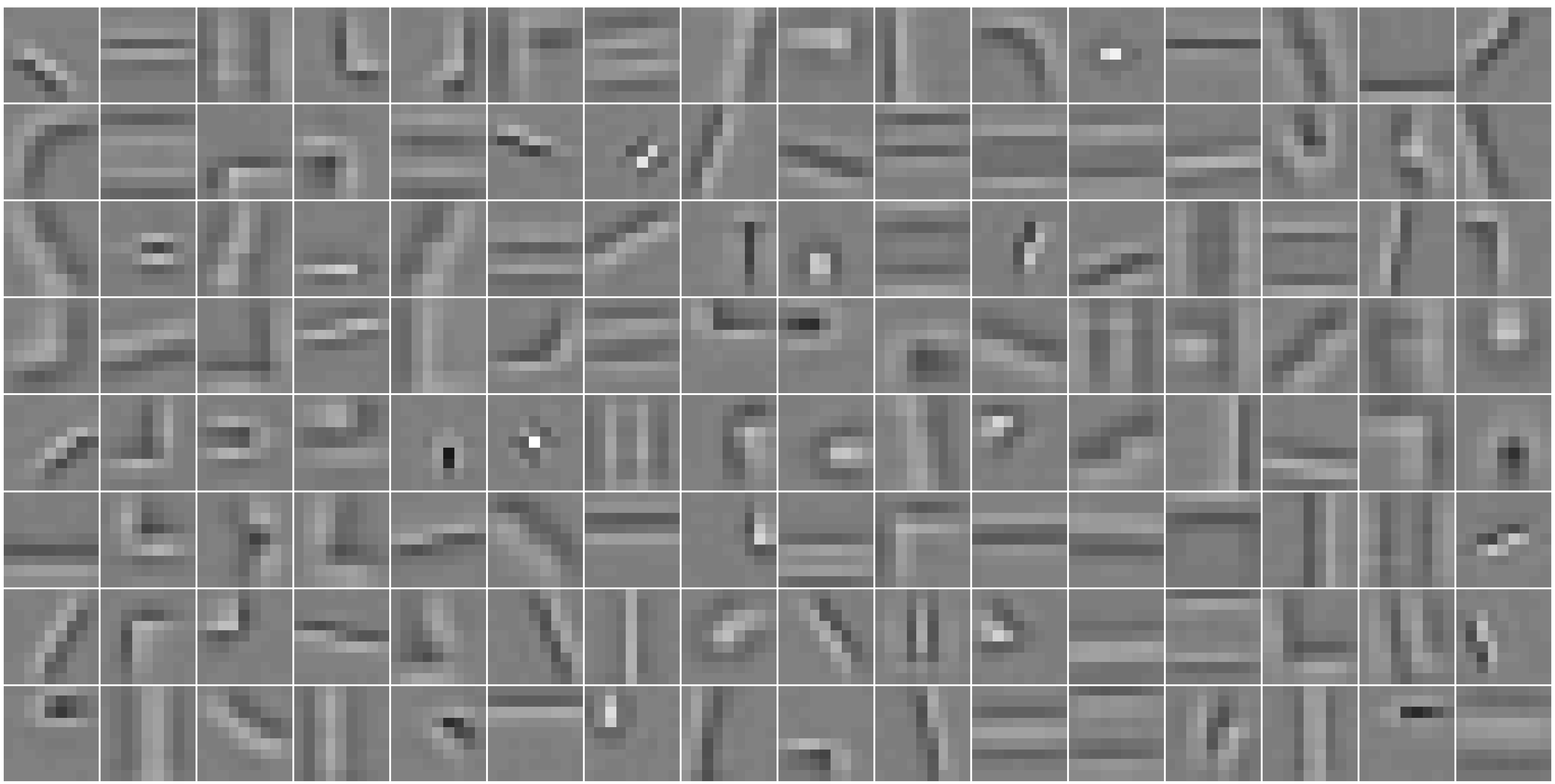
Results on “Reasonable” Images (>50 pixels tall, few occlusions)

Y LeCun
MA Ranzato



Unsupervised pre-training with convolutional PSD

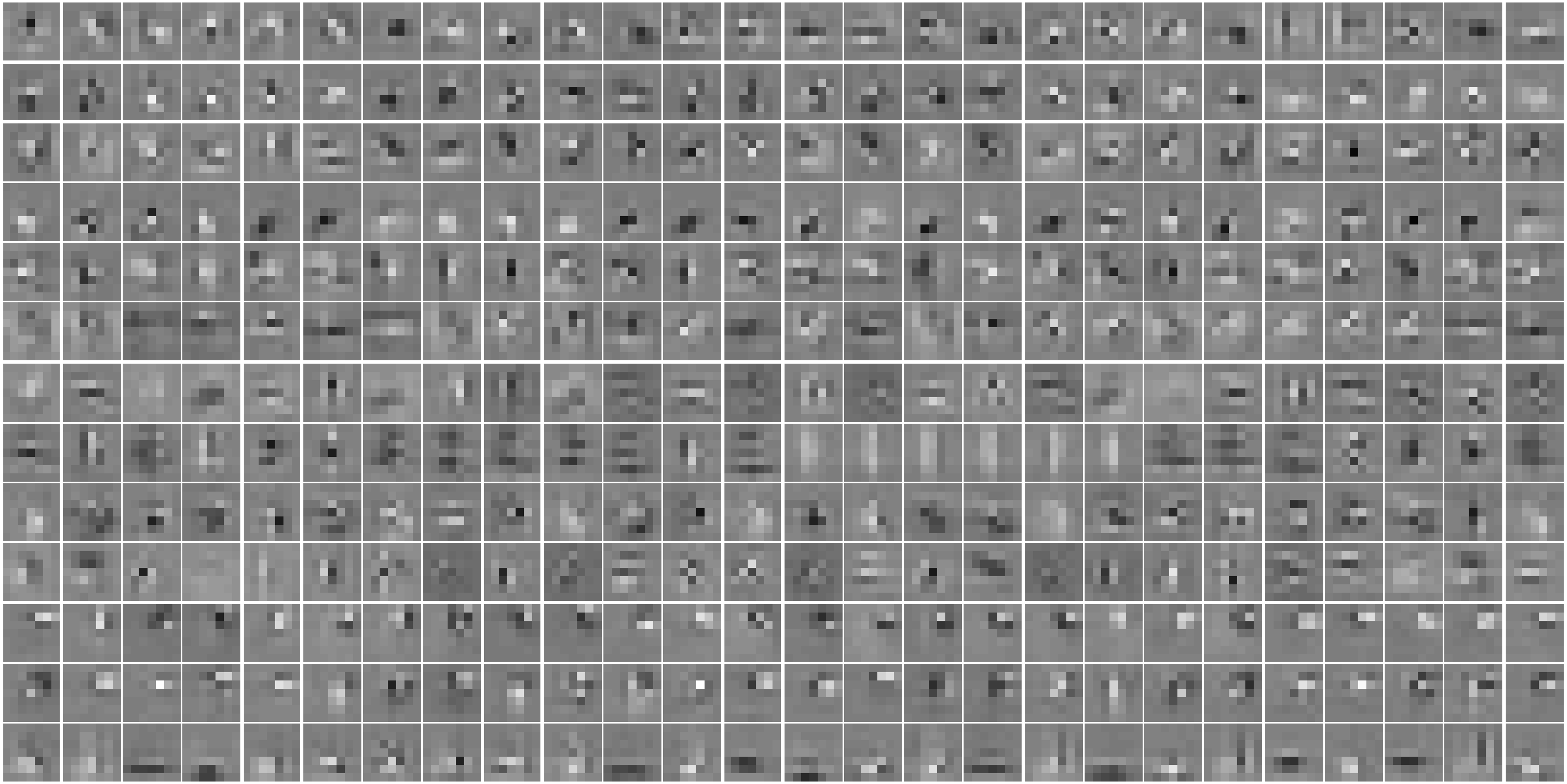
- 128 stage-1 filters on Y channel.
- Unsupervised training with convolutional predictive sparse decomposition



Unsupervised pre-training with convolutional PSD

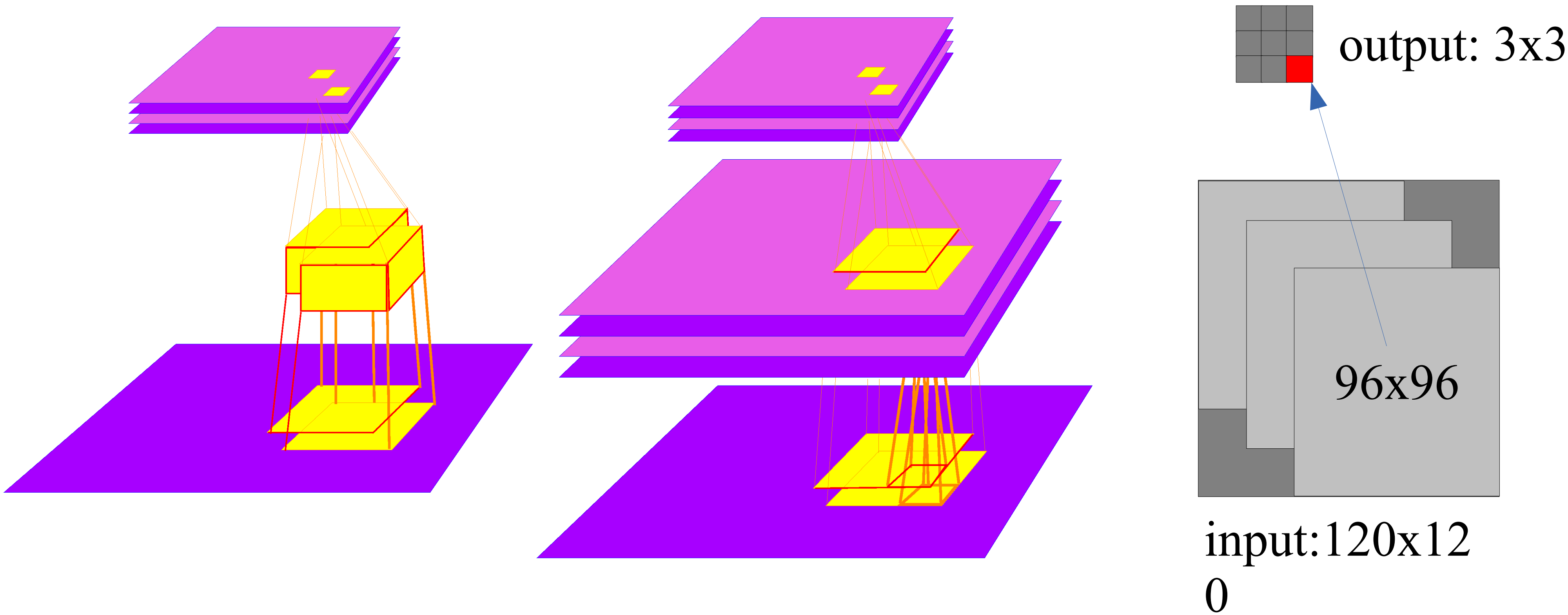
Y LeCun
MA Ranzato

- Stage 2 filters.
- Unsupervised training with convolutional predictive sparse decomposition



Applying a ConvNet on Sliding Windows is Very Cheap!

Y LeCun
MA Ranzato

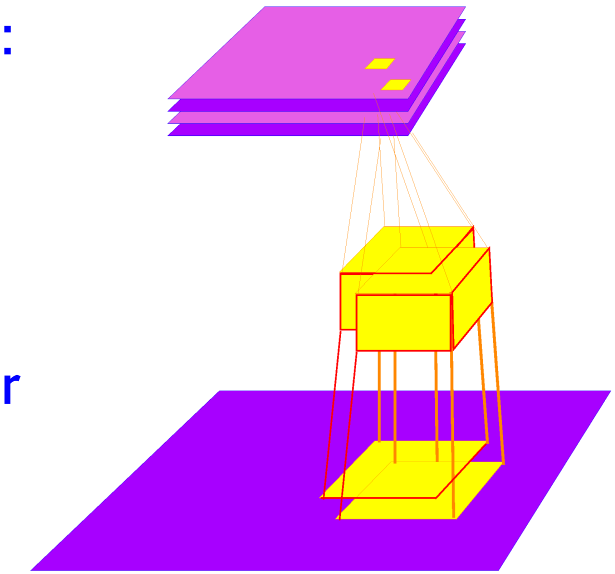


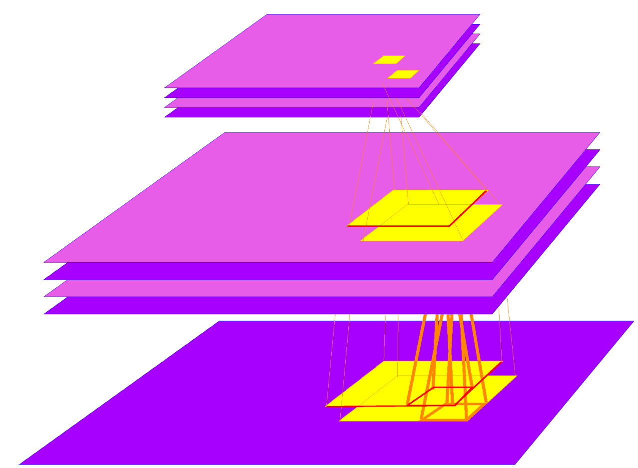
- Traditional Detectors/Classifiers must be applied to every location on a large input image, at multiple scales.
- Convolutional nets can replicated over large images very cheaply.
- The network is applied to multiple scales spaced by 1.5.

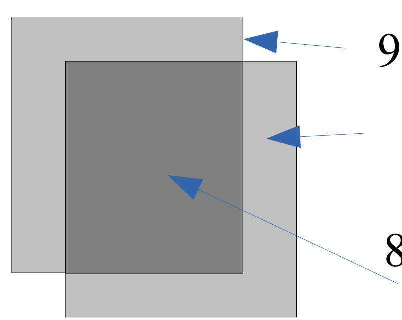
Building a Detector/Recognizer: Replicated Convolutional Nets

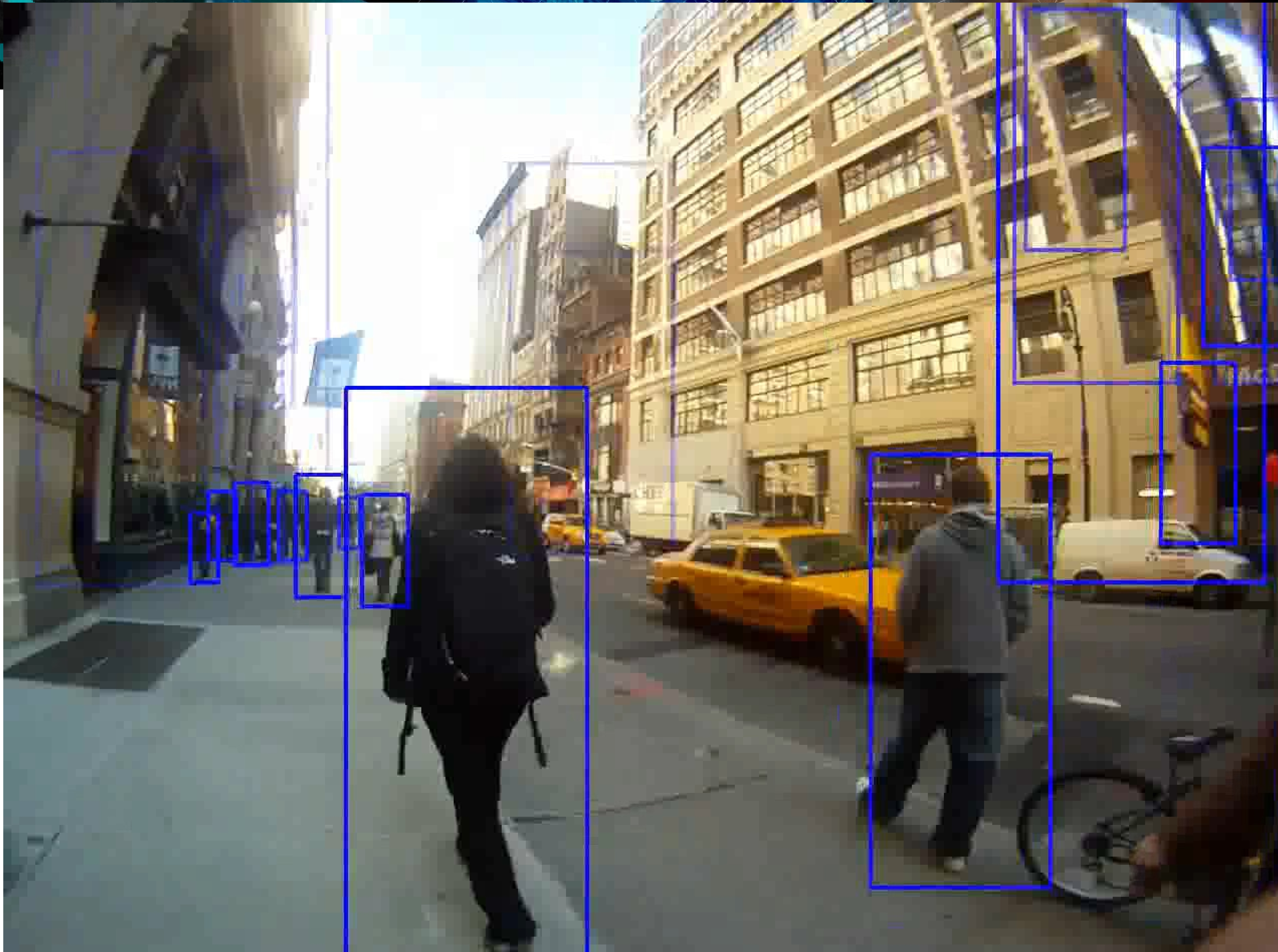
Y LeCun
MA Ranzato

- Computational cost for replicated convolutional net:
  - 96x96 -> 4.6 million multiply-accumulate operations
  - 120x120 -> 8.3 million multiply-accumulate ops
  - 240x240 -> 47.5 million multiply-accumulate ops
  - 480x480 -> 232 million multiply-accumulate ops
- Computational cost for a non-convolutional detector of the same size, applied every 12 pixels:
  - 96x96 -> 4.6 million multiply-accumulate operations
  - 120x120 -> 42.0 million multiply-accumulate operations
  - 240x240 -> 788.0 million multiply-accumulate ops
  - 480x480 -> 5,083 million multiply-accumulate ops









Cun
ato

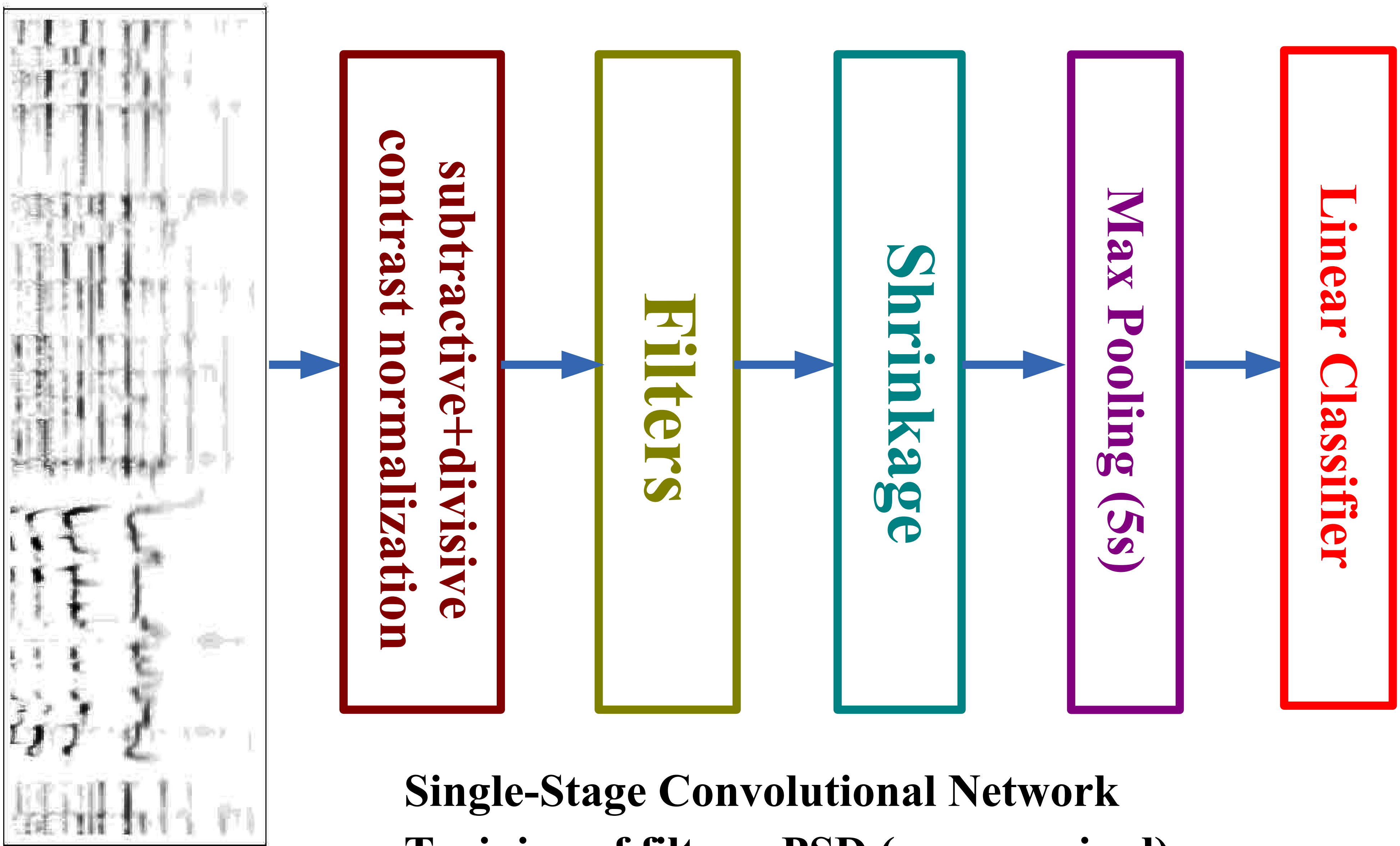
Musical Genre Recognition with PSD Feature

Y LeCun
MA Ranzato

- **Input: "Constant Q Transform" over 46.4ms windows (1024 samples)**
  - 96 filters, with frequencies spaced every quarter tone (4 octaves)
- **Architecture:**
  - Input: sequence of contrast-normalized CQT vectors
  - 1: PSD features, 512 trained filters; shrinkage function → rectification
  - 3: pooling over 5 seconds
  - 4: linear SVM classifier. Pooling of SVM categories over 30 seconds
- **GTZAN Dataset**
  - 1000 clips, 30 second each
  - 10 genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and rock.
- **Results**
  - 84% correct classification

Architecture: contrast norm → filters → shrink → max pooling

Y LeCun
MA Ranzato

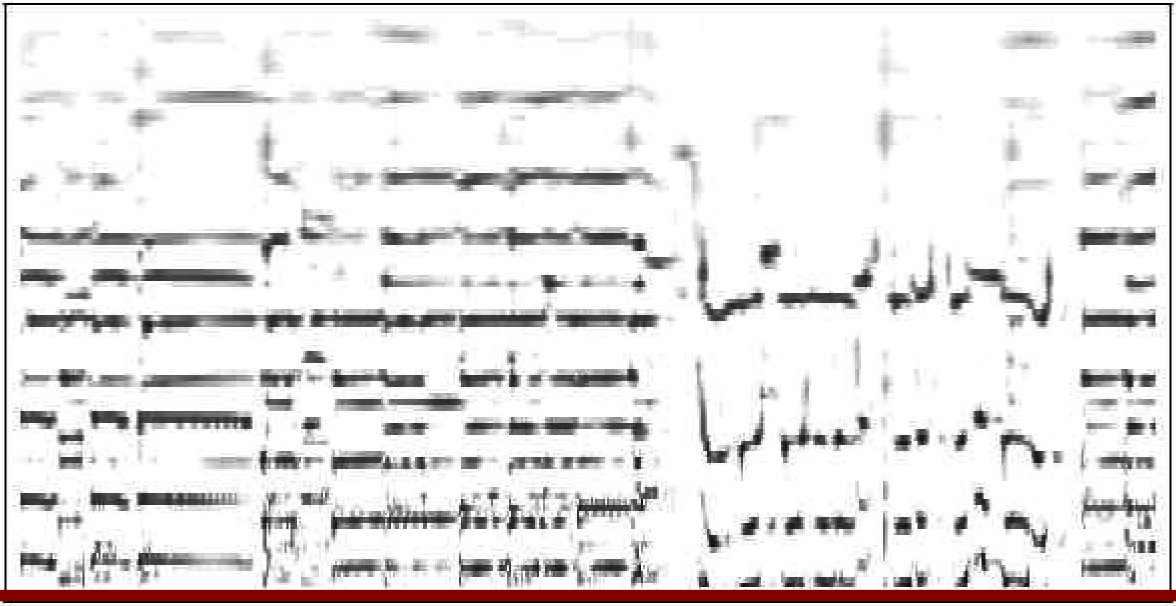


Single-Stage Convolutional Network

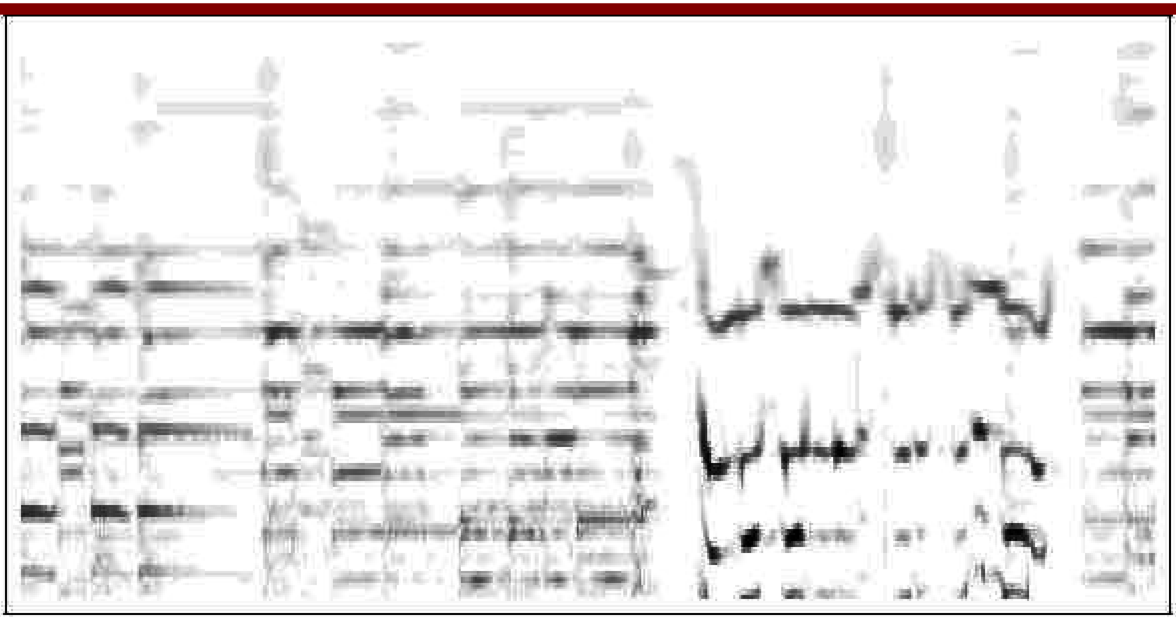
Training of filters: PSD (unsupervised)

Constant Q Transform over 46.4 ms → Contrast Normalization

Y LeCun
MA Ranzato



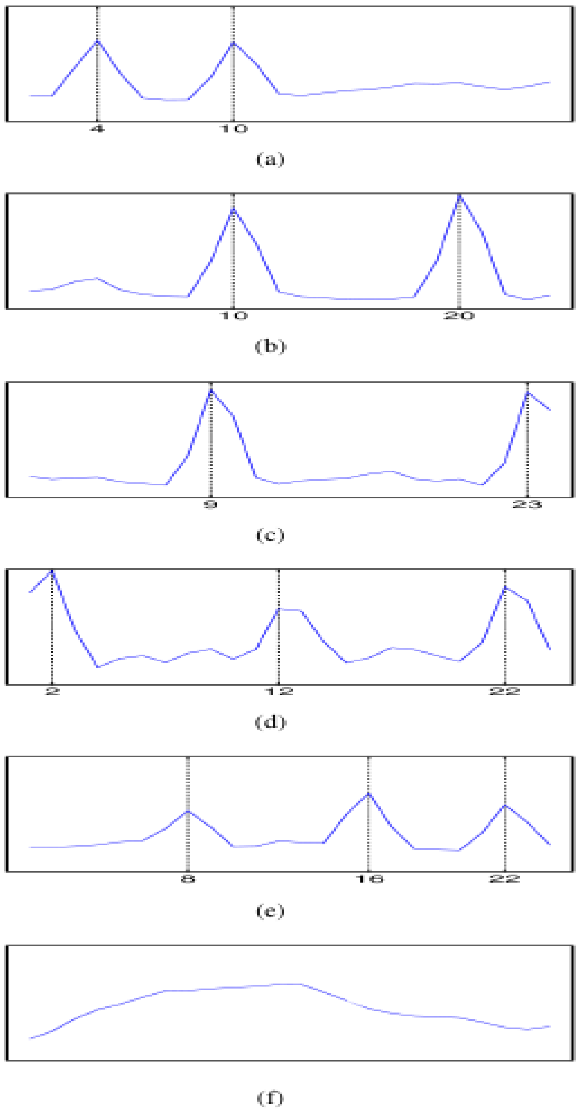
subtractive+divisive contrast normalization



Convolutional PSD Features on Time-Frequency Signals

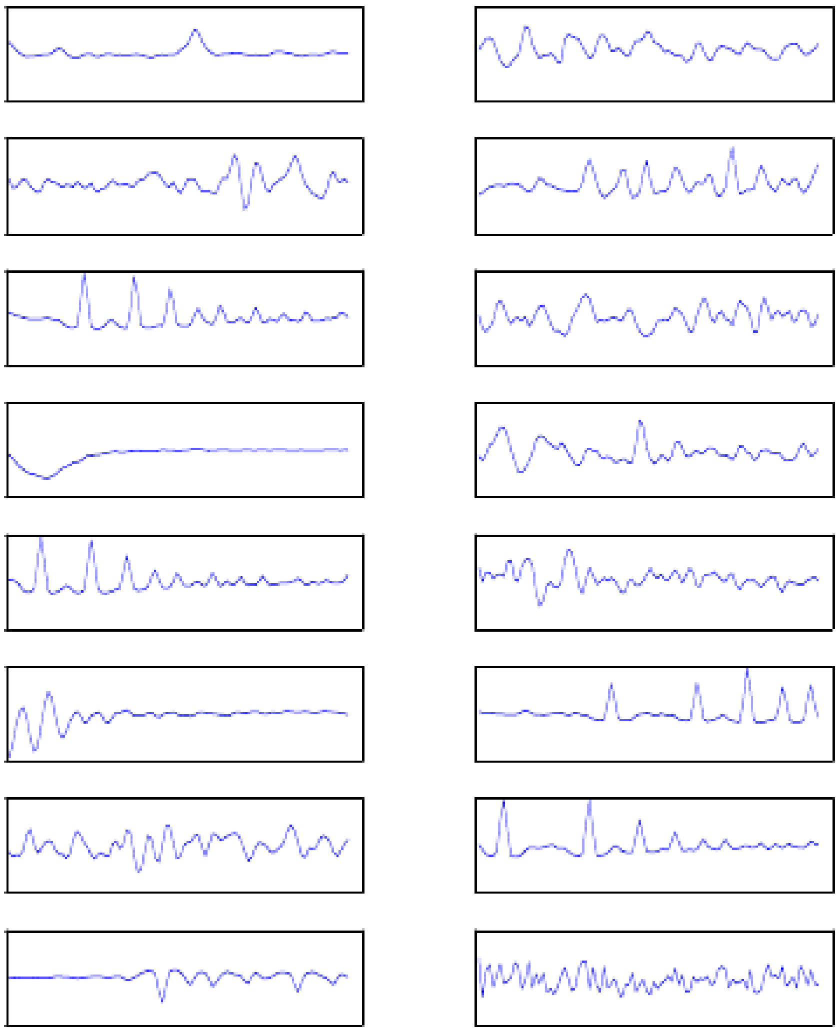
Y LeCun
MA Ranzato

- Octave-wide features



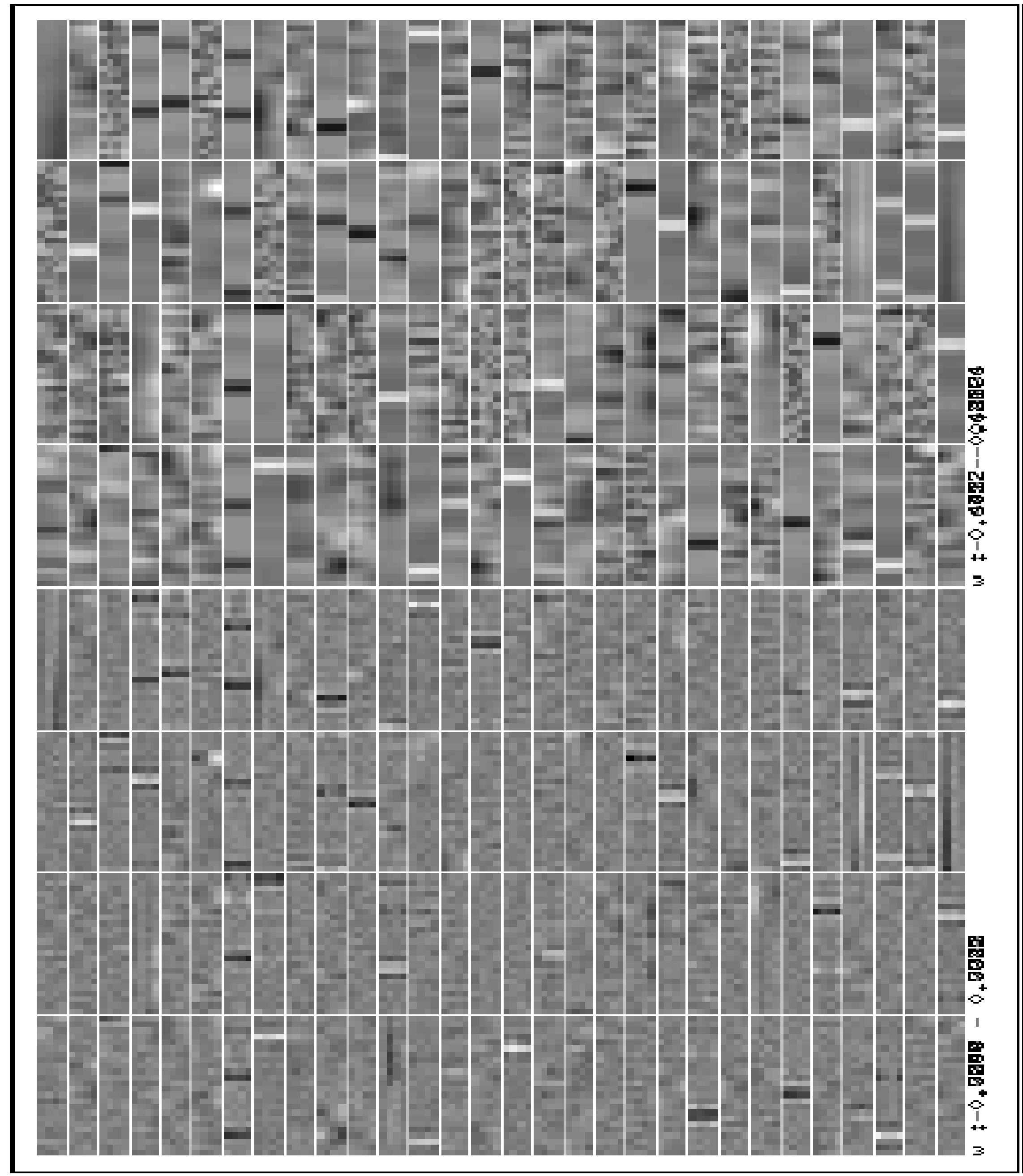
- Minor 3rd
- Perfect 4th
- Perfect 5th
- Quartal chord
- Major triad
- transient

full 4-octave features



PSD Features on Constant-Q Transform

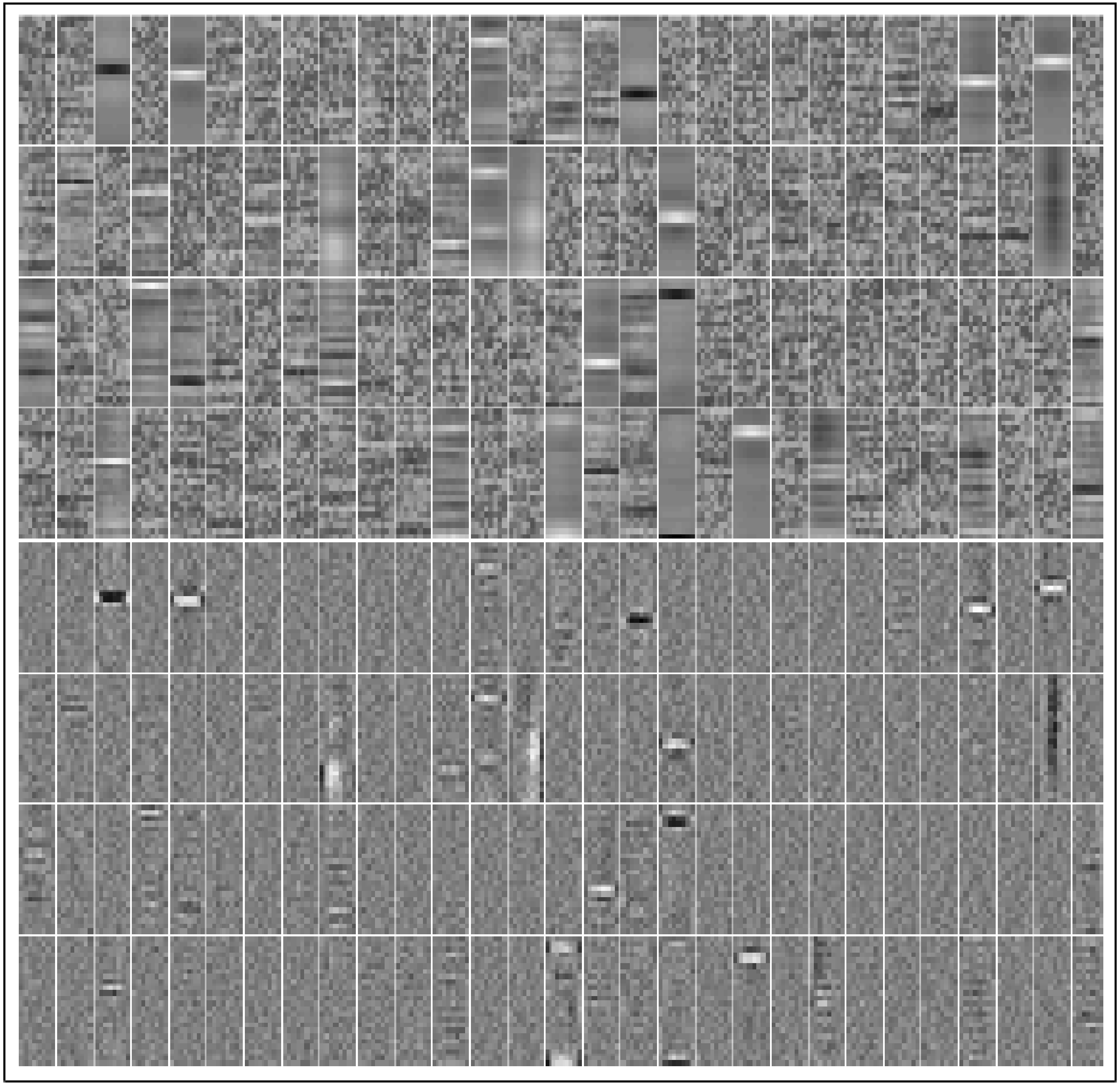
- **Octave-wide features**
  - Encoder basis functions
  - Decoder basis functions



Time-Frequency Features

Y LeCun
MA Ranzato

- **Octave-wide features on 8 successive acoustic vectors**
  - Almost no temporal structure in the filters!



Accuracy on GTZAN dataset (small, old, etc...)

- Accuracy: 83.4%. State of the Art: 84.3%
- Very fast

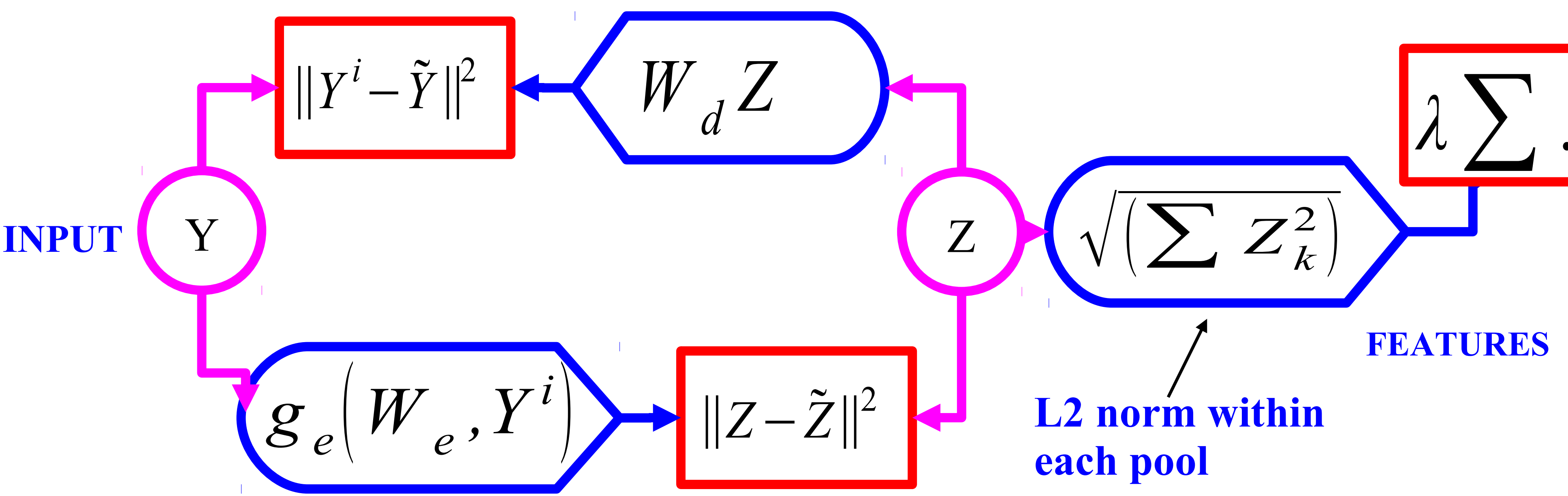
Classifier	Features	Acc. (%)
[redacted]		
RBF-SVM	Learned using DBN [12]	84.3
Linear SVM	Learned using PSD on octaves	**83.4 ± 3.1**
AdaBoost	Many features [2]	83
Linear SVM	Learned using PSD on frames	**79.4 ± 2.8**
SVM	Daubechies Wavelets [19]	78.5
Log. Reg.	Spectral Covariance [3]	77
LDA	MFCC + other [18]	71
Linear SVM	Auditory cortical feat. [25]	70
GMM	MFCC + other [29]	61

Unsupervised Learning: Invariant Features

Learning Invariant Features with L2 Group Sparsity

- Unsupervised PSD ignores the spatial pooling step.
- Could we devise a similar method that learns the pooling layer as well?
- Idea [Hyvarinen & Hoyer 2001]: group sparsity on pools of features
  - Minimum number of pools must be non-zero
  - Number of features that are on within a pool doesn't matter
  - Pools tend to regroup similar features

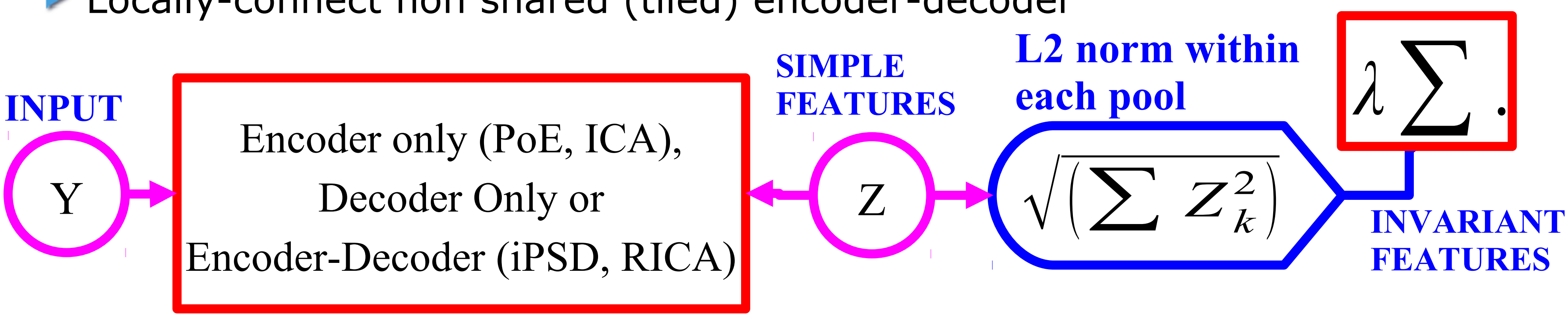
$$E(Y,Z) = \|Y - W_d Z\|^2 + \|Z - g_e(W_e, Y)\|^2 + \sum_j \sqrt{\sum_{k \in P_j} Z_k^2}$$



Learning Invariant Features with L2 Group Sparsity

Y LeCun
MA Ranzato

- **Idea: features are pooled in group.**
  - Sparsity: sum over groups of L2 norm of activity in group.
- **[Hyvärinen Hoyer 2001]: "subspace ICA"**
  - decoder only, square
- **[Welling, Hinton, Osindero NIPS 2002]: pooled product of experts**
  - encoder only, overcomplete, log student-T penalty on L2 pooling
- **[Kavukcuoglu, Ranzato, Fergus LeCun, CVPR 2010]: Invariant PSD**
  - encoder-decoder (like PSD), overcomplete, L2 pooling
- **[Le et al. NIPS 2011]: Reconstruction ICA**
  - Same as [Kavukcuoglu 2010] with linear encoder and tied decoder
- **[Gregor & LeCun arXiv:1006:0448, 2010] [Le et al. ICML 2012]**
  - Locally-connect non shared (tiled) encoder-decoder



Groups are local in a 2D Topographic Map

Y LeCun
MA Ranzato

- The filters arrange themselves spontaneously so that similar filters enter the same pool.
- The pooling units can be seen as complex cells
- Outputs of pooling units are invariant to local transformations of the input
  - For some it's translations, for others rotations, or other transformations.

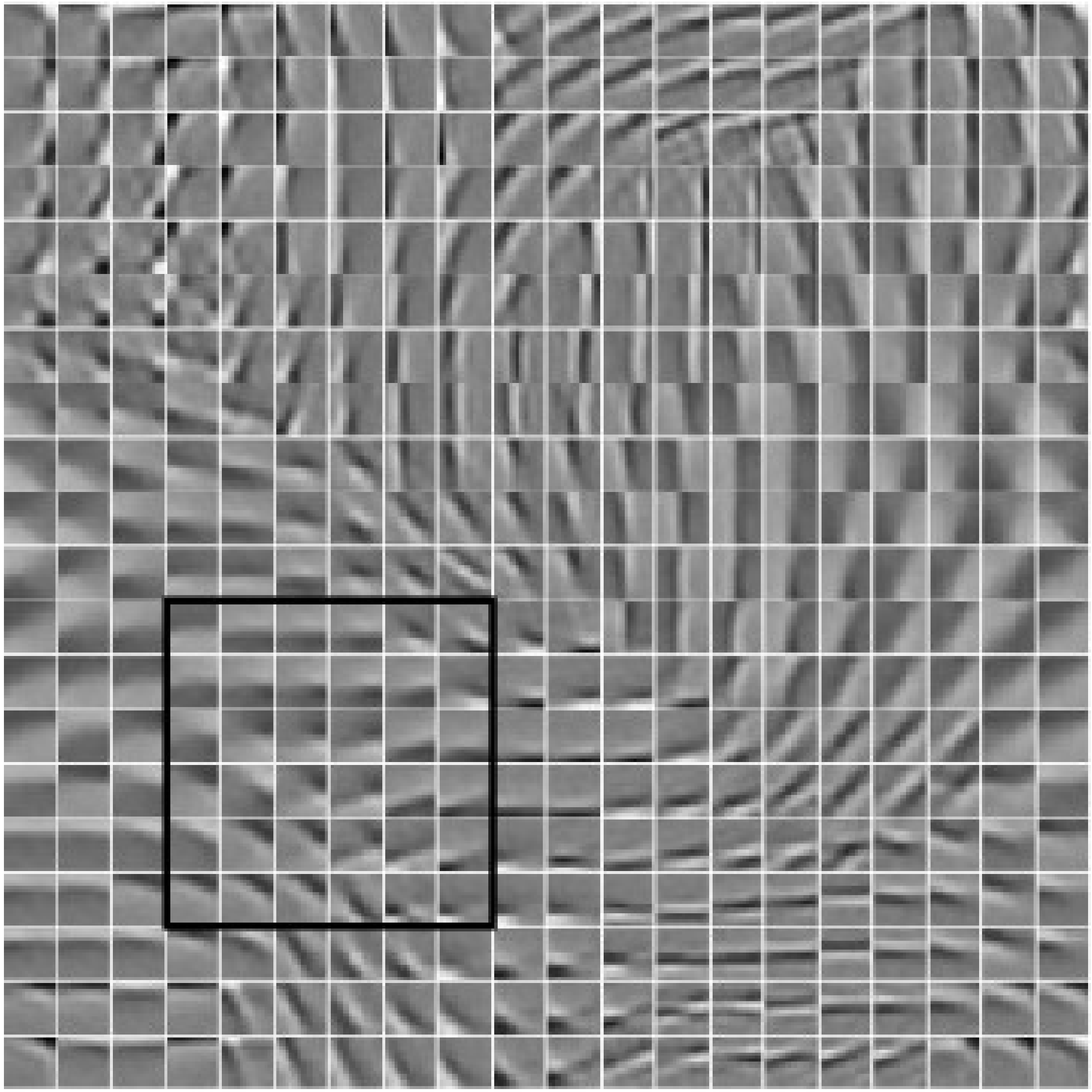


Image-level training, local filters but no weight sharing

Y LeCun
MA Ranzato

- Training on 115x115 images. Kernels are 15x15 (not shared across space!)
  - [Gregor & LeCun 2010]
  - Local receptive fields
  - No shared weights
  - 4x overcomplete
  - L2 pooling
  - Group sparsity over pools

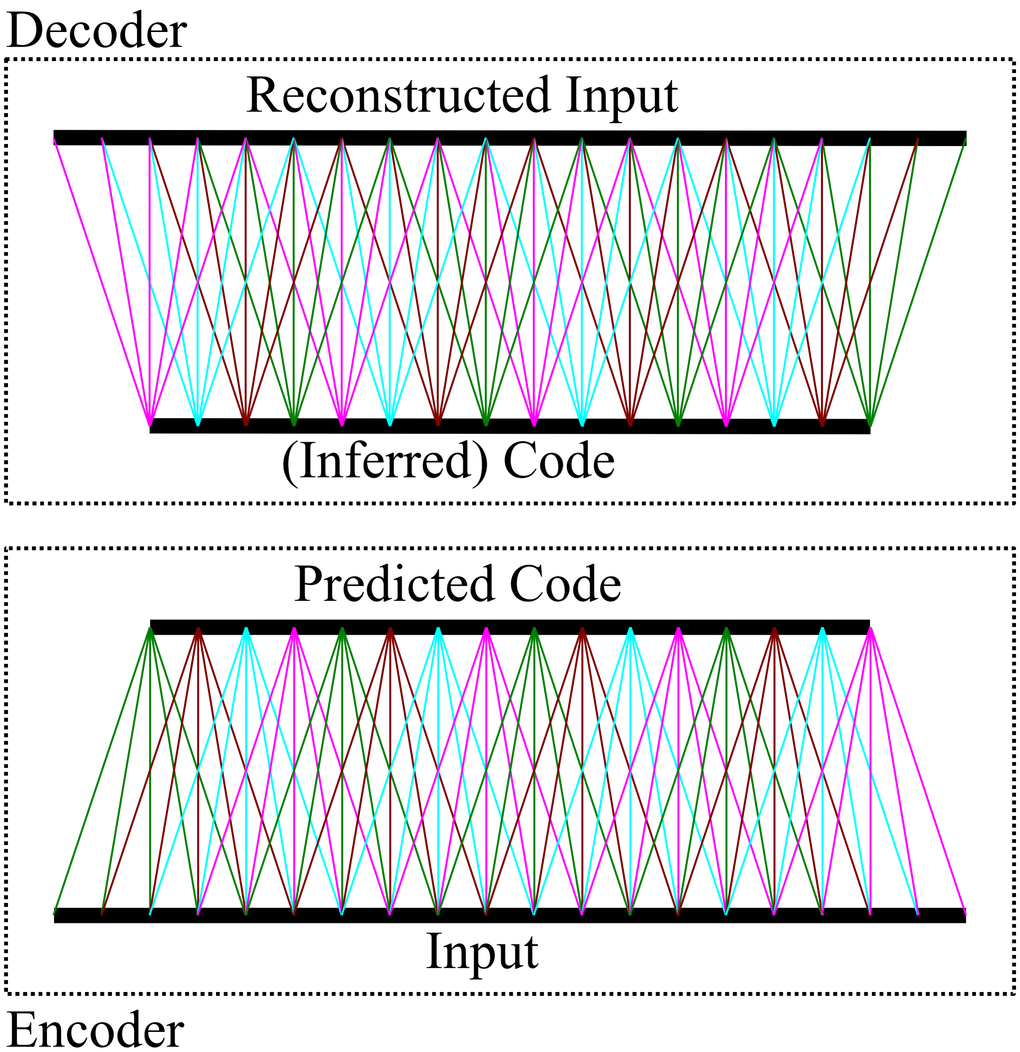
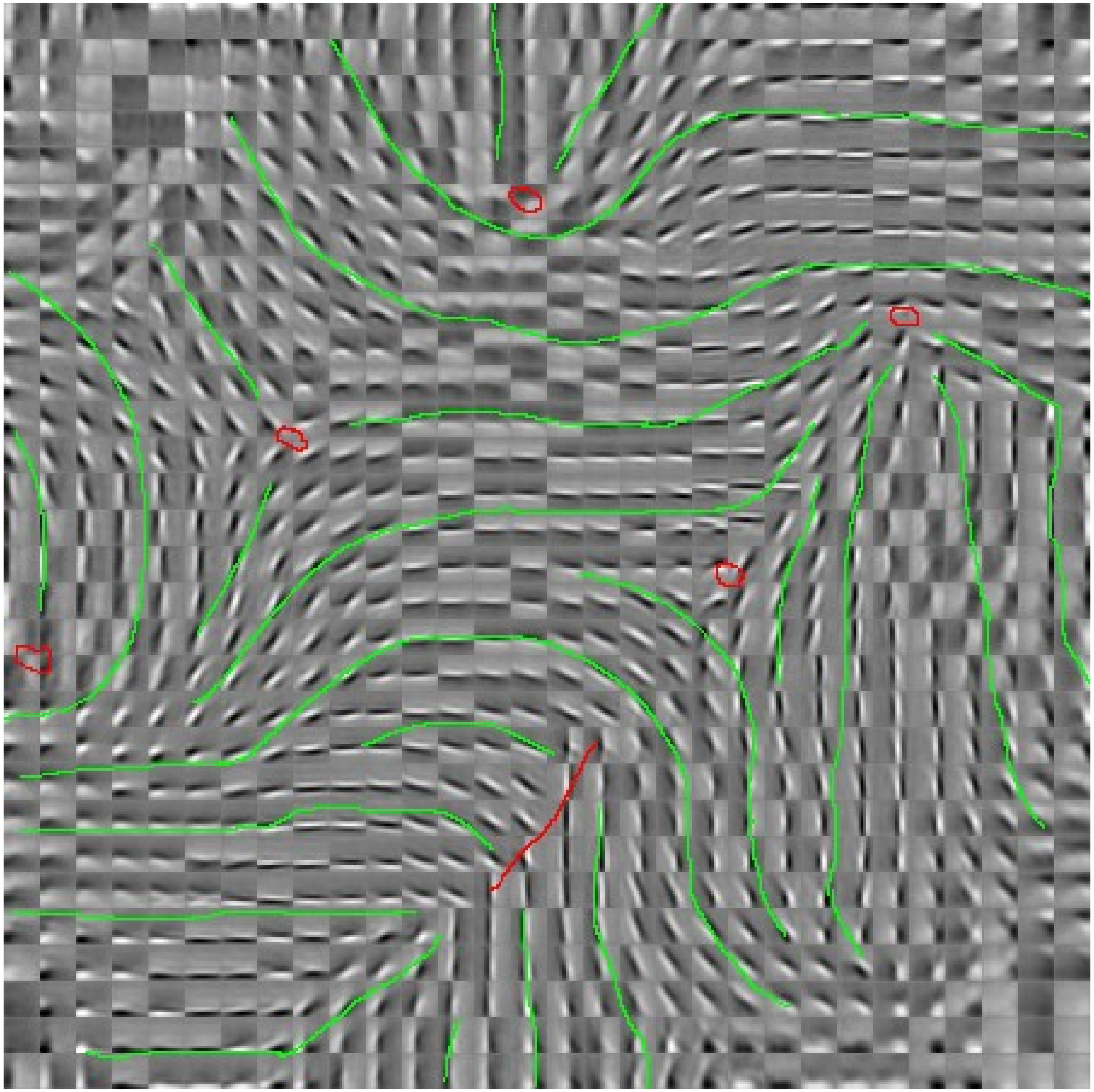


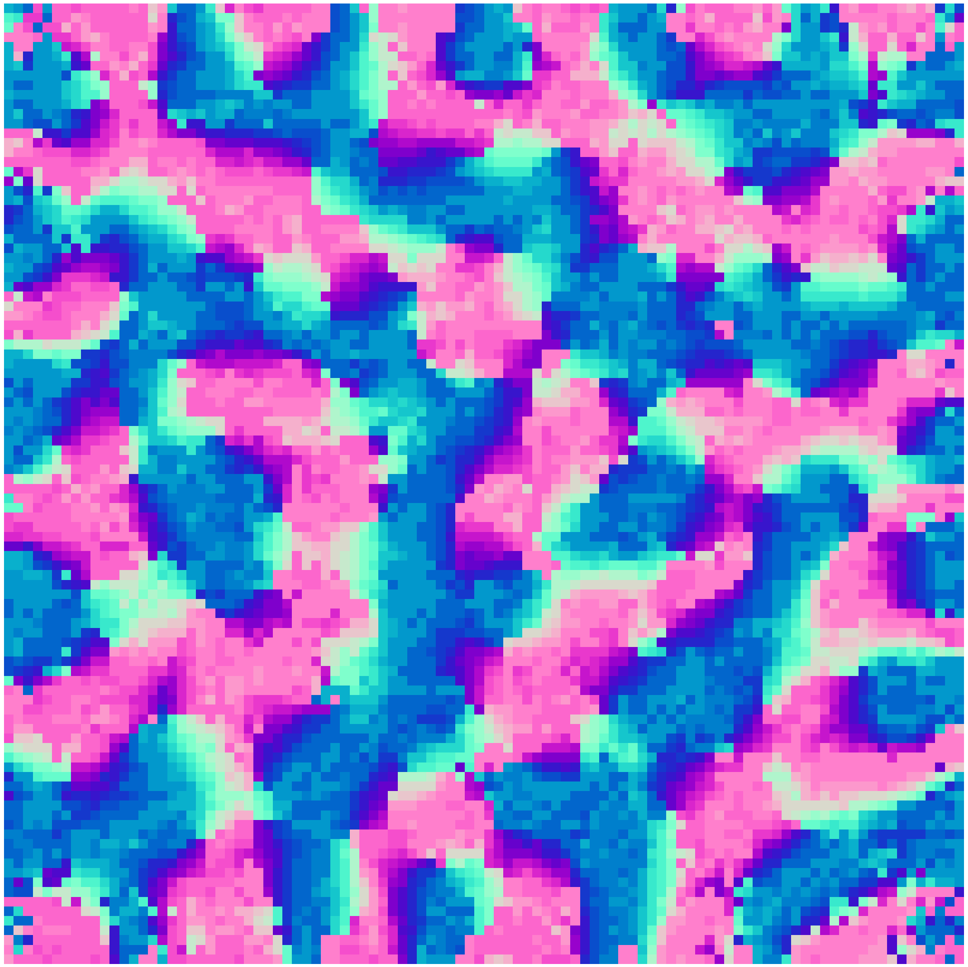
Image-level training, local filters but no weight sharing

- Training on 115x115 images. Kernels are 15x15 (not shared across space!)



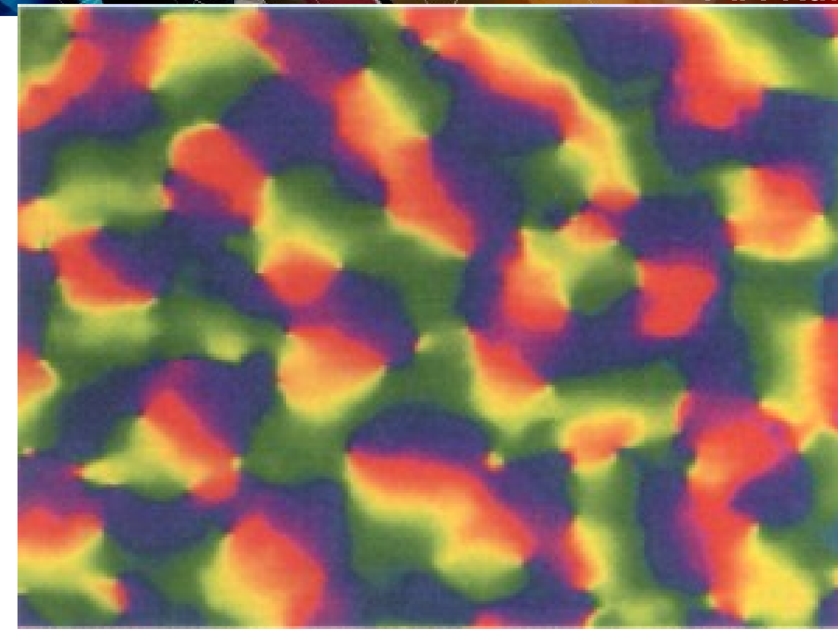
Topographic Maps

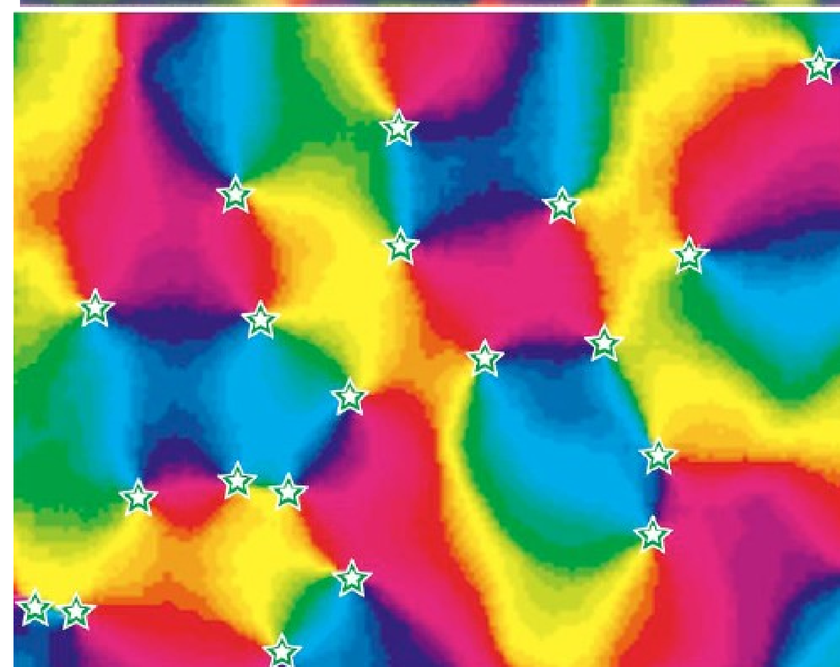
Y LeCun
MA Ranzato



119x119 Image Input
100x100 Code
20x20 Receptive field size
sigma=5

K Obermayer and GG Blasdel, Journal of Neuroscience, Vol 13, 4114-4129 (**Monkey**)



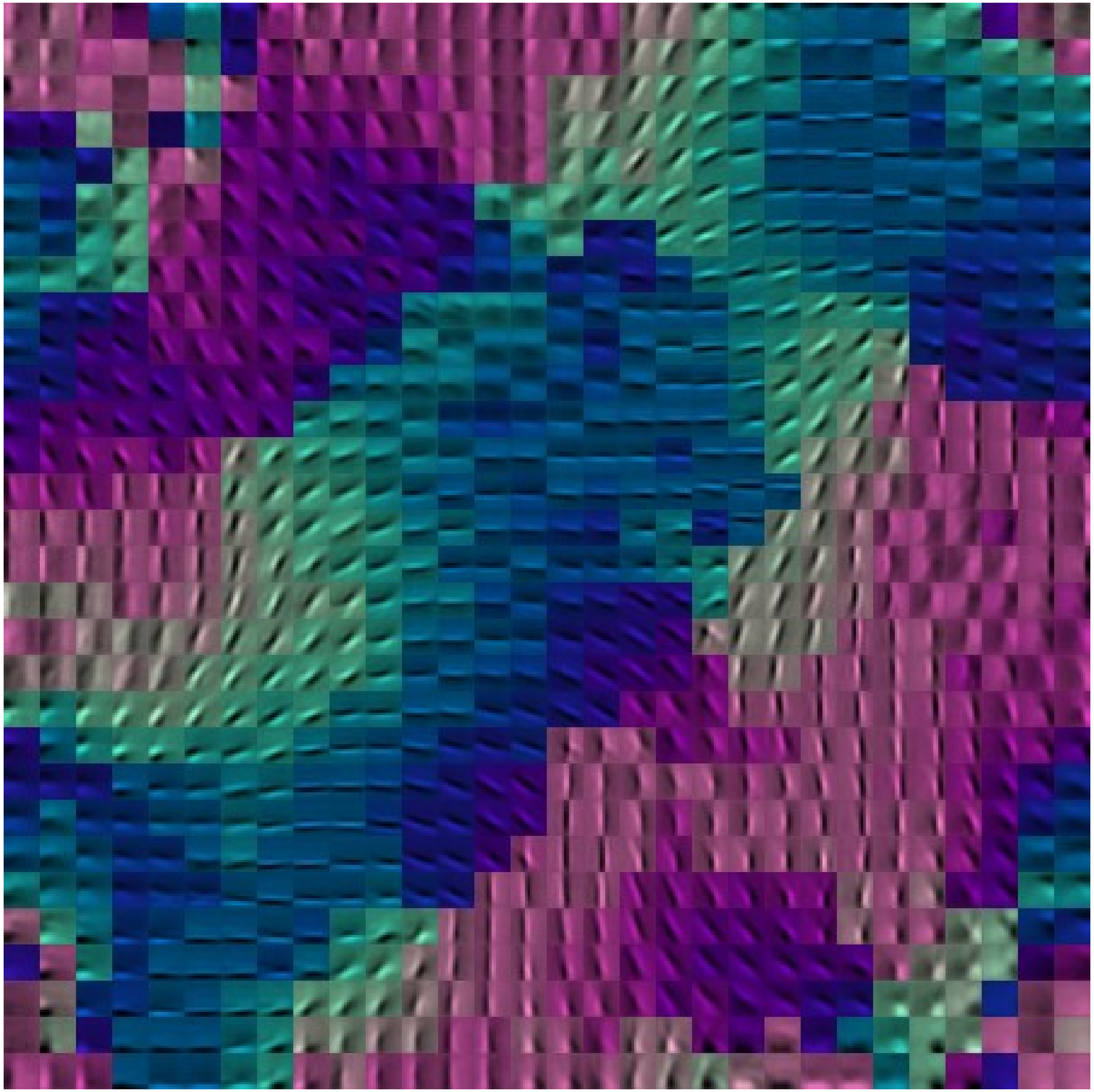


Michael C. Crair, et. al. The Journal of Neurophysiology Vol. 77 No. 6 June 1997, pp. 3381-3385 (**Cat**)

Image-level training, local filters but no weight sharing

Y LeCun
MA Ranzato

- Color indicates orientation (by fitting Gabors)

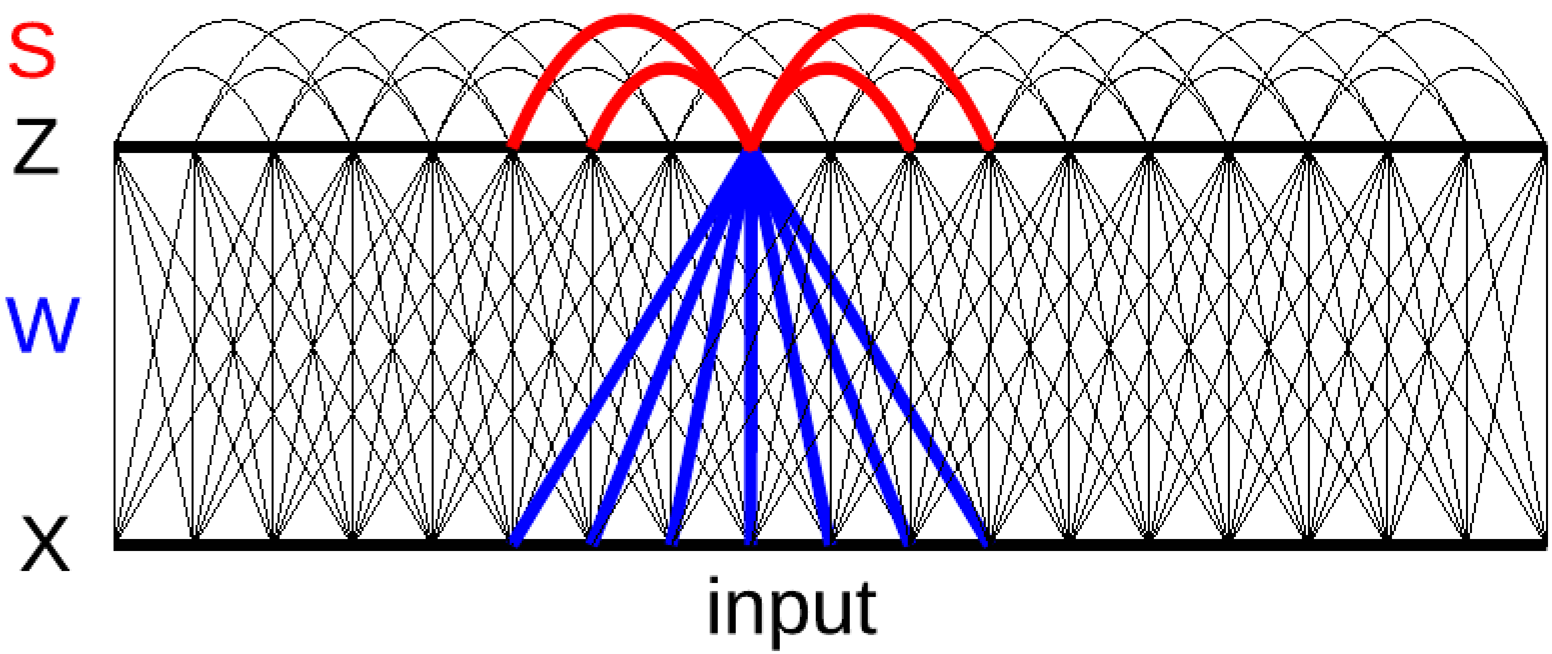


Invariant Features Lateral Inhibition

Y LeCun
MA Ranzato

- Replace the L1 sparsity term by a lateral inhibition matrix
- Easy way to impose some structure on the sparsity

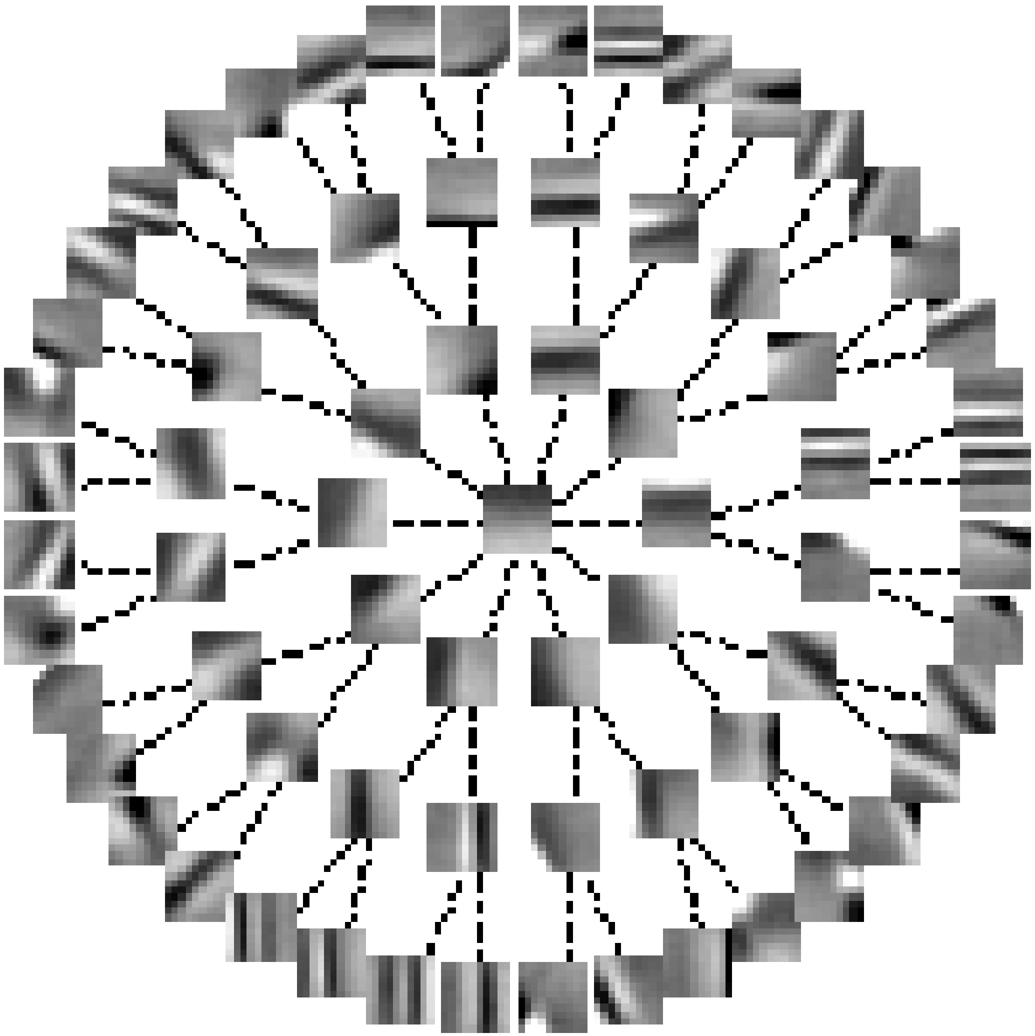
$$\min_{W,Z} \sum_{x \in X} ||Wz - x||^2 + |z|^T S|z|$$



[Gregor, Szlam, LeCun NIPS 2011]

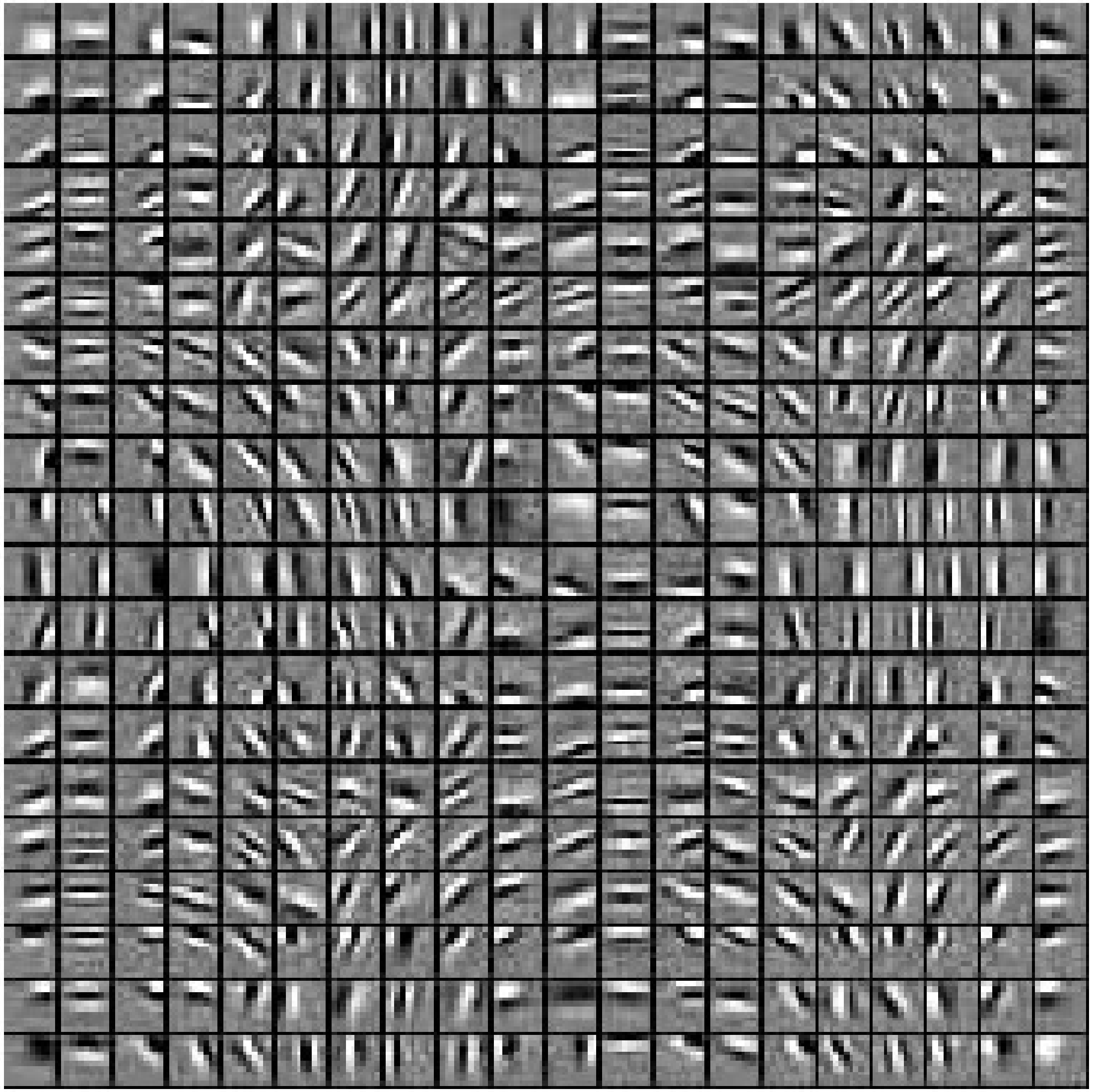
Invariant Features via Lateral Inhibition: Structured Sparsity

- Each edge in the tree indicates a zero in the S matrix (no mutual inhibition)
- Sij is larger if two neurons are far away in the tree



Invariant Features via Lateral Inhibition: Topographic Maps

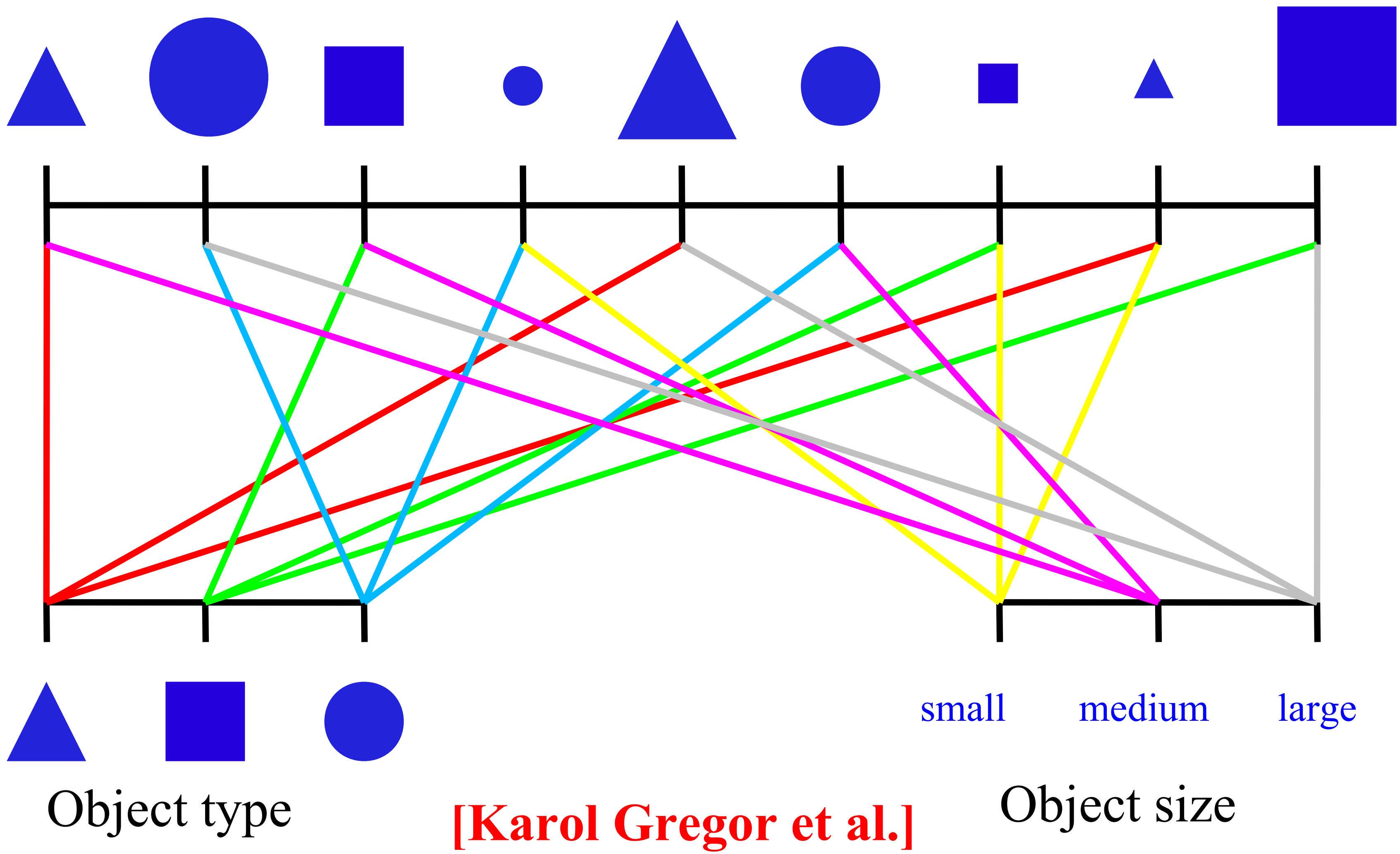
- **Non-zero values in S form a ring in a 2D topology**
  - Input patches are high-pass filtered



Invariant Features through Temporal Constancy

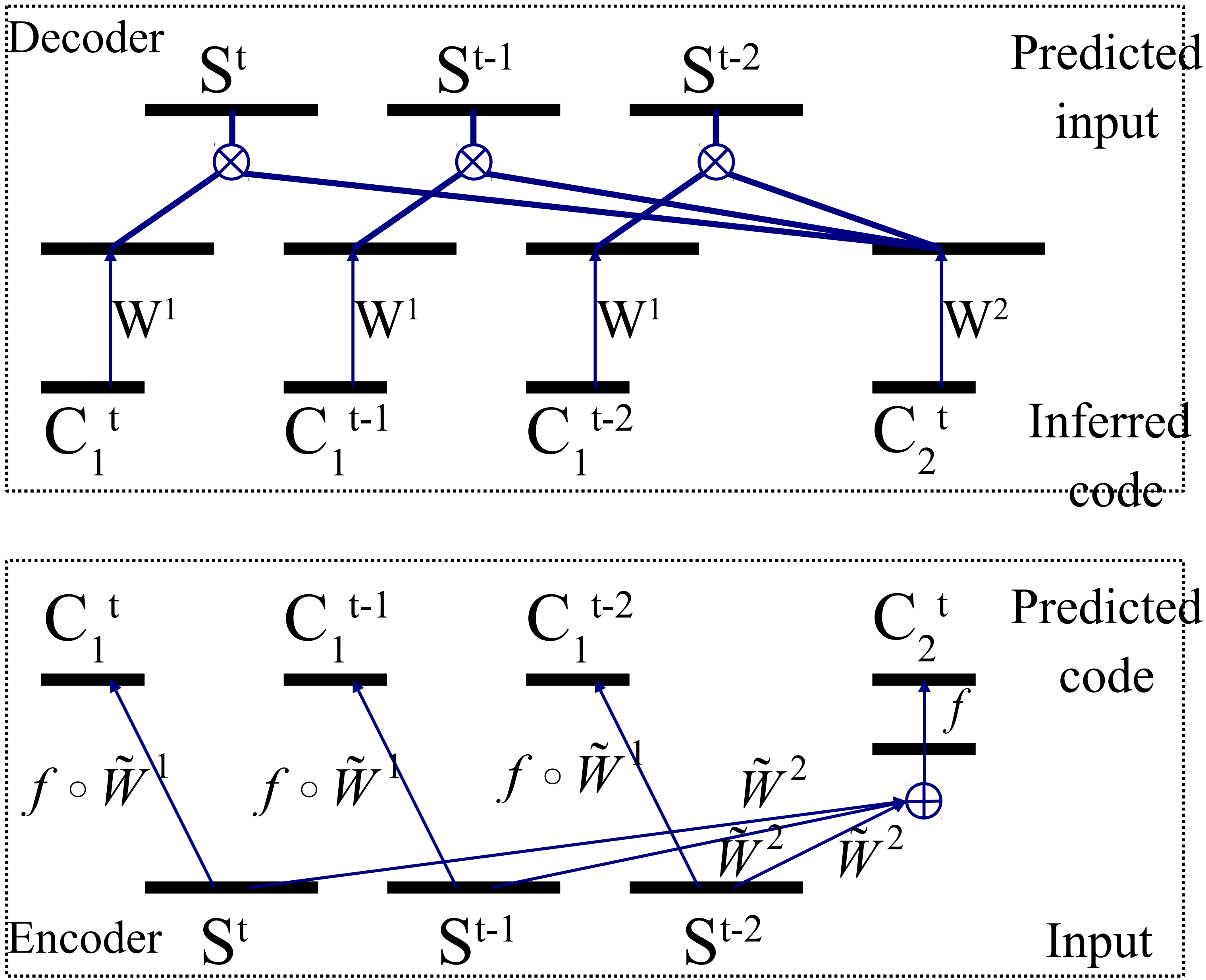
Y LeCun
MA Ranzato

- Object is cross-product of object type and instantiation parameters
  - Mapping units [Hinton 1981], capsules [Hinton 2011]



What-Where Auto-Encoder Architecture

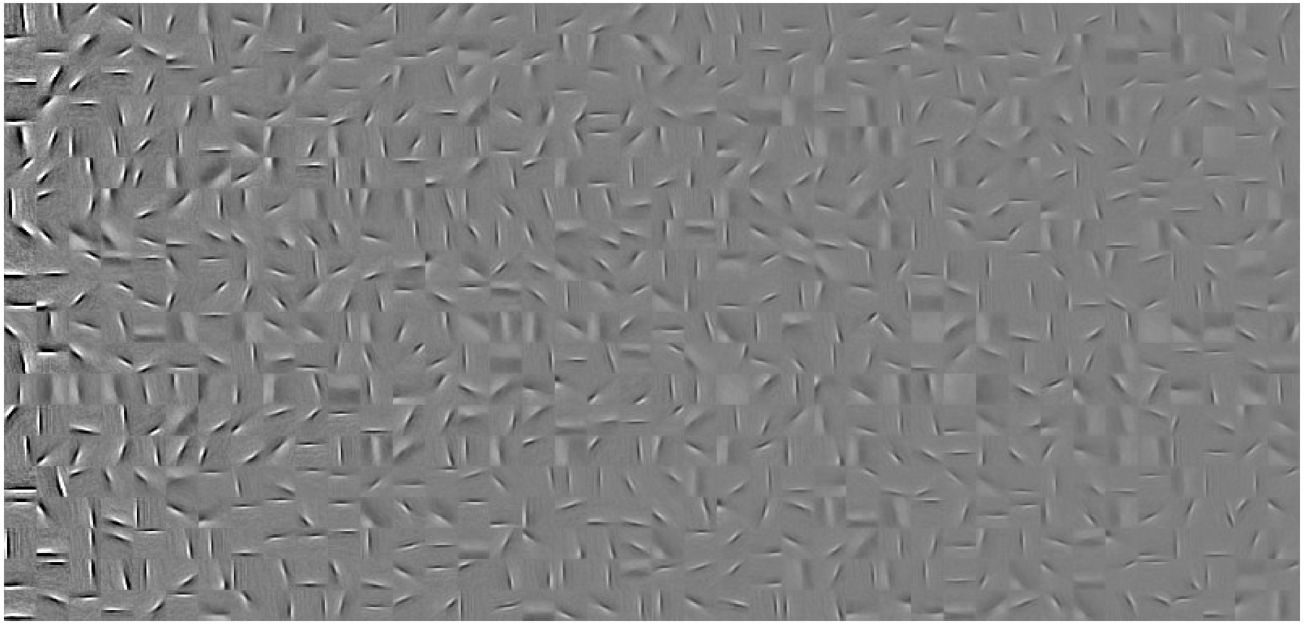
Y LeCun
MA Ranzato



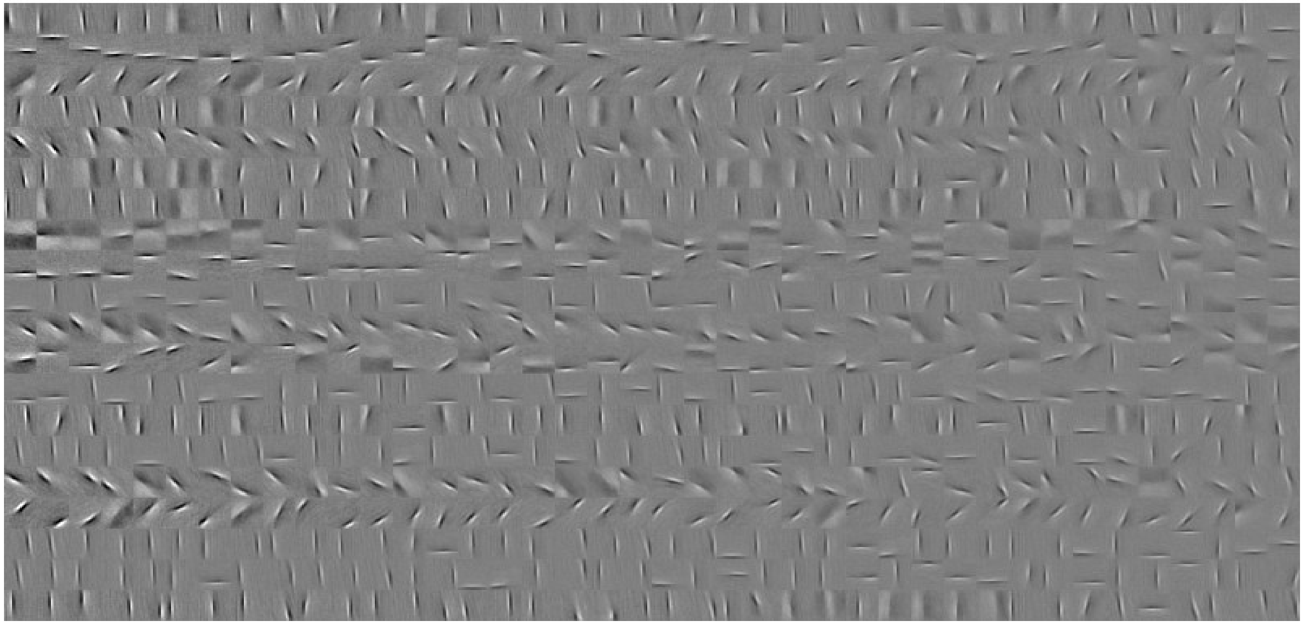
Low-Level Filters Connected to Each Complex Cell

Y LeCun
MA Ranzato

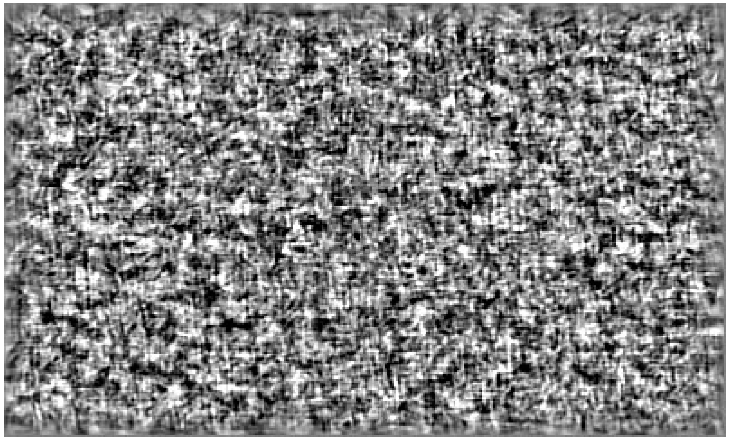
C1
(where)

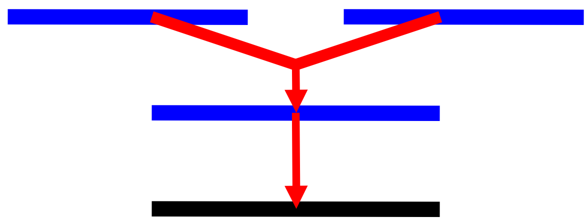


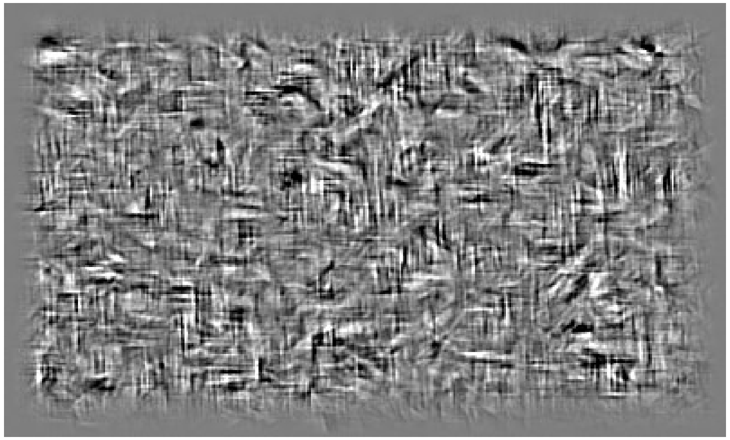
C2
(what)

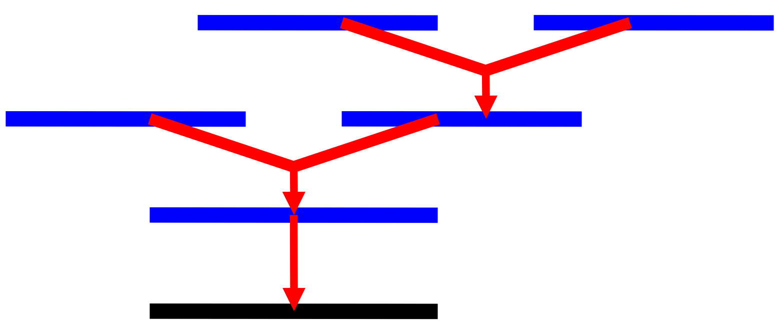


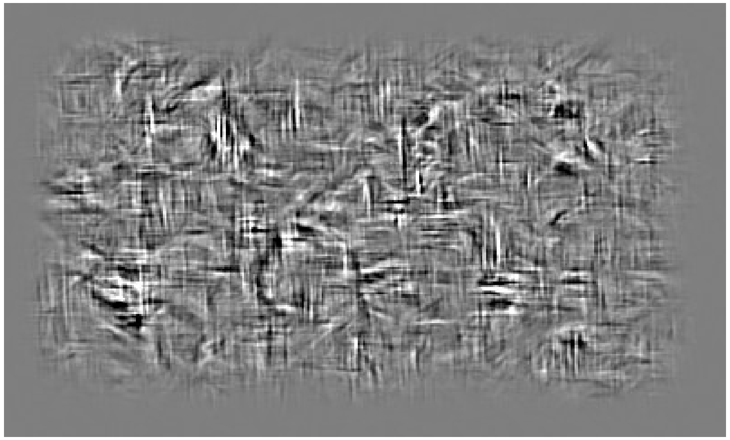
Generating images







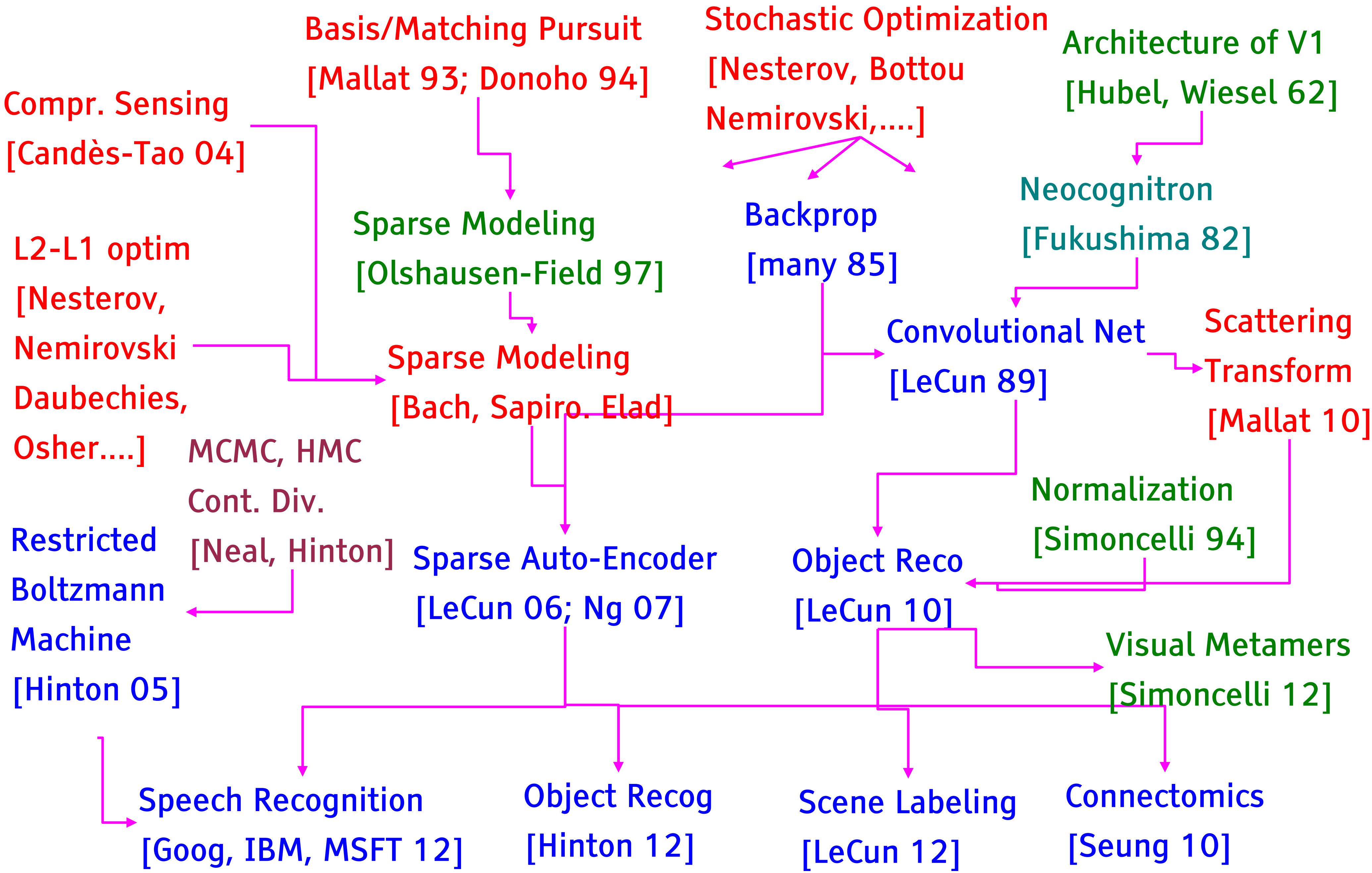




Future Challenges

The Graph of Deep Learning ↔ Sparse Modeling ↔ Neuroscience

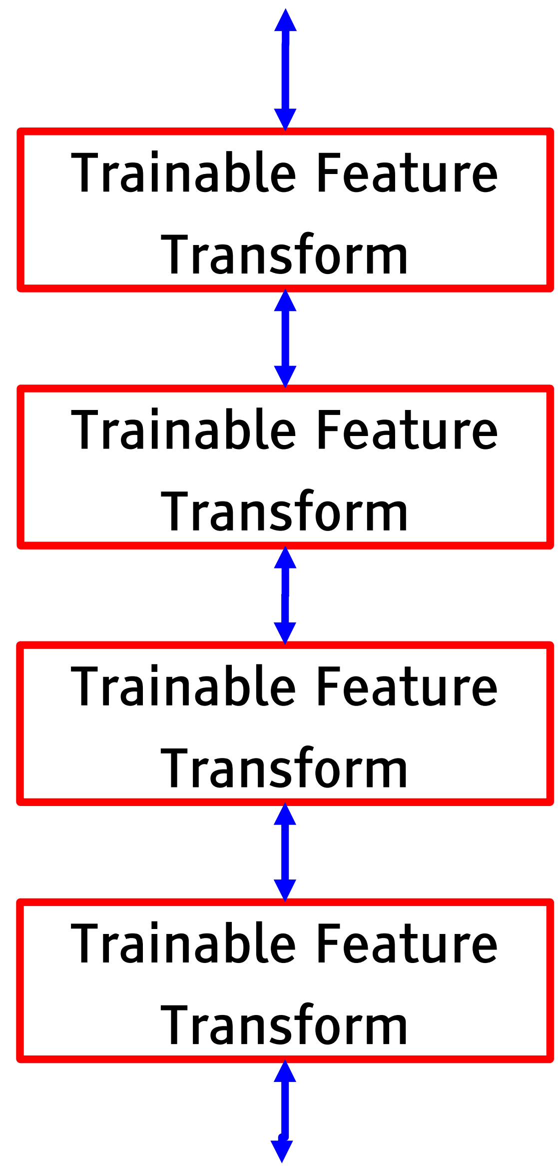
Y LeCun
MA Ranzato



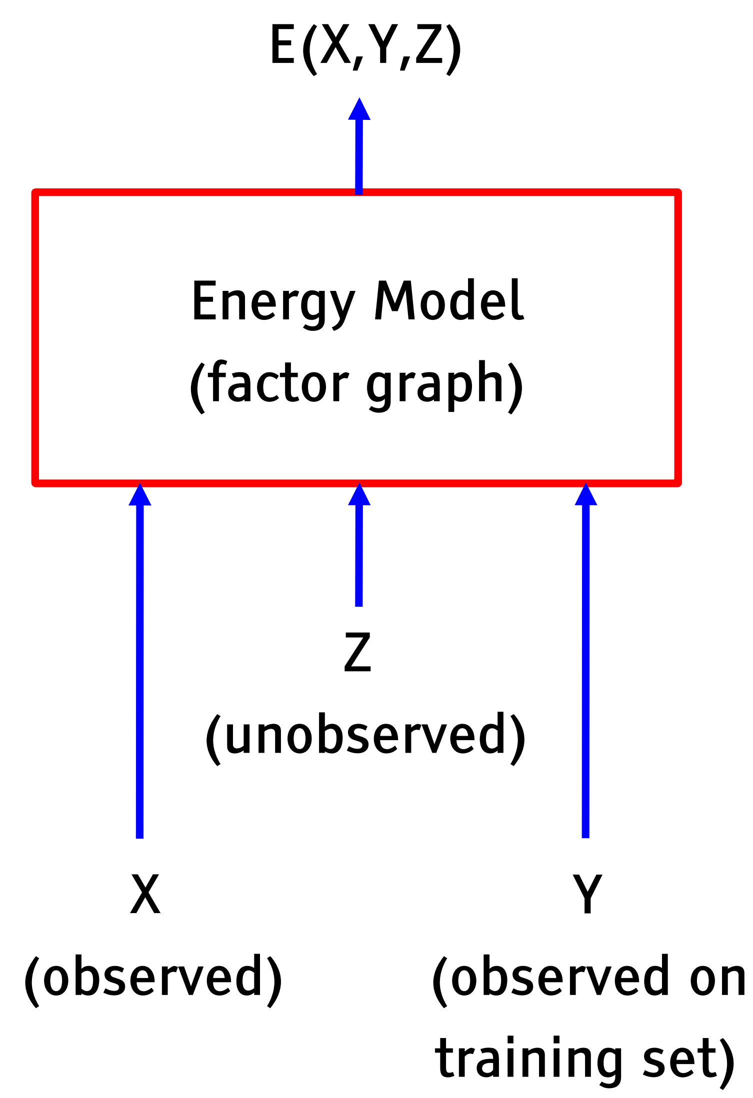
Integrating Feed-Forward and Feedback

- **Marrying feed-forward convolutional nets with generative "deconvolutional nets"**
  - Deconvolutional networks
    - [Zeiler-Graham-Fergus ICCV 2011]

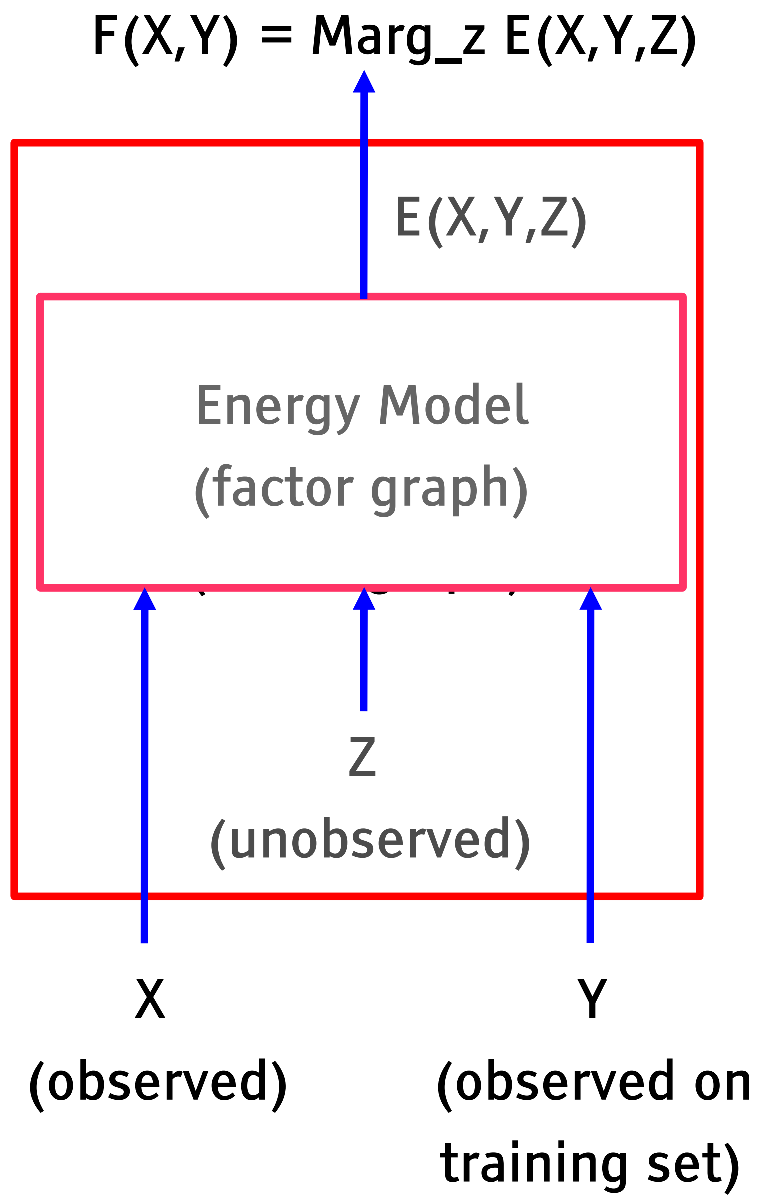
- **Feed-forward/Feedback networks allow reconstruction, multimodal prediction, restoration, etc...**
  - Deep Boltzmann machines can do this, but there are scalability issues with training



- **Deep Learning systems can be assembled into factor graphs**
  - Energy function is a sum of factors
  - Factors can embed whole deep learning systems
  - X: observed variables (inputs)
  - Z: never observed (latent variables)
  - Y: observed on training set (output variables)
- **Inference is energy minimization (MAP) or free energy minimization (marginalization) over Z and Y given an X**

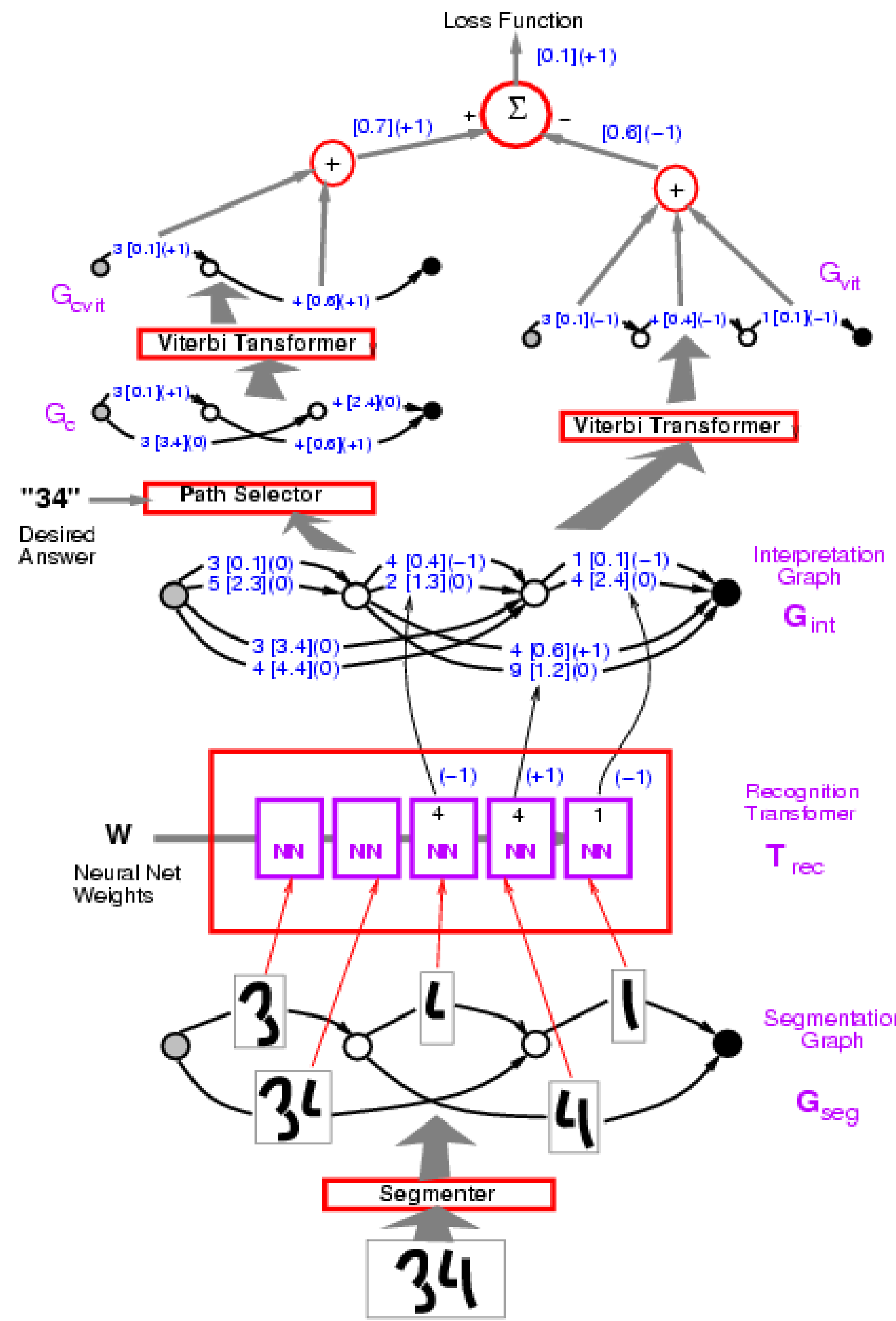


- **Deep Learning systems can be assembled into factor graphs**
  - Energy function is a sum of factors
  - Factors can embed whole deep learning systems
  - X: observed variables (inputs)
  - Z: never observed (latent variables)
  - Y: observed on training set (output variables)
- **Inference is energy minimization (MAP) or free energy minimization (marginalization) over Z and Y given an X**
  - F(X,Y) = MIN_z E(X,Y,Z)
  - F(X,Y) = -log SUM_z exp[-E(X,Y,Z)]



Integrating Deep Learning and Structured Prediction

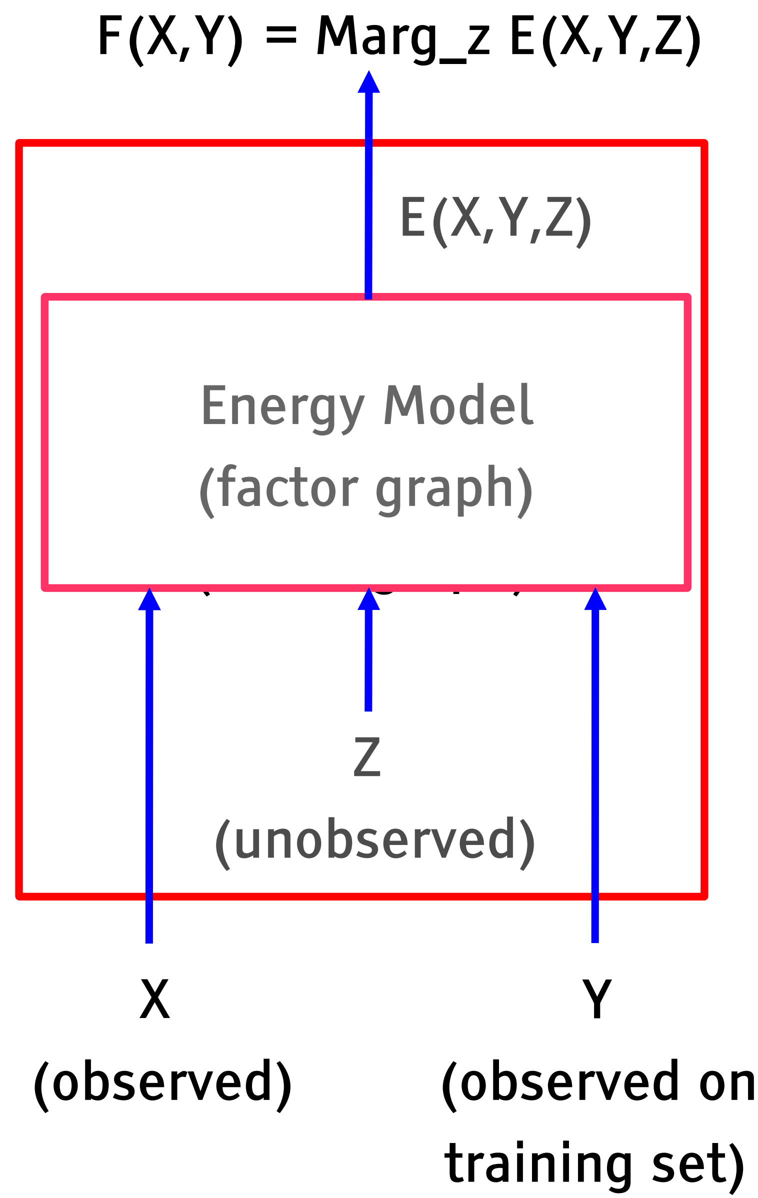
- **Integrting deep learning and structured prediction is a very old idea**
  - In fact, it predates structured prediction
- **Globally-trained convolutional-net + graphical models**
  - trained discriminatively at the word level
  - Loss identical to CRF and structured perceptron
  - Compositional movable parts model
- **A system like this was reading 10 to 20% of all the checks in the US around 1998**



Integrating Deep Learning and Structured Prediction

Y LeCun
MA Ranzato

- **Deep Learning systems can be assembled into factor graphs**
  - Energy function is a sum of factors
  - Factors can embed whole deep learning systems
  - X: observed variables (inputs)
  - Z: never observed (latent variables)
  - Y: observed on training set (output variables)
- **Inference is energy minimization (MAP) or free energy minimization (marginalization) over Z and Y given an X**
  - F(X,Y) = MIN_z E(X,Y,Z)
  - F(X,Y) = -log SUM_z exp[-E(X,Y,Z)]



Future Challenges

- **Integrated feed-forward and feedback**
  - Deep Boltzmann machine do this, but there are issues of scalability.
- **Integrating supervised and unsupervised learning in a single algorithm**
  - Again, deep Boltzmann machines do this, but....
- **Integrating deep learning and structured prediction (“reasoning”)**
  - This has been around since the 1990's but needs to be revived
- **Learning representations for complex reasoning**
  - “recursive” networks that operate on vector space representations of knowledge [Pollack 90's] [Bottou 2010] [Socher, Manning, Ng 2011]
- **Representation learning in natural language processing**
  - [Y. Bengio 01],[Collobert Weston 10], [Mnih Hinton 11] [Socher 12]
- **Better theoretical understanding of deep learning and convolutional nets**
  - e.g. Stephane Mallat's “scattering transform”, work on the sparse representations from the applied math community....

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)
- http://eblearn.sf.net (C++ Library with convnet support by P. Sermanet)

Python-based learning library (U. Montreal)

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

RNN

– www.fit.vutbr.cz/~imikolov/rnnlm (language modeling)
– http://sourceforge.net/apps/mediawiki/rnnl/index.php (LSTM)

CUDAMat & GNumpy

– code.google.com/p/cudamat
– www.cs.toronto.edu/~tijmen/gnumpy.html

Misc

– www.deeplearning.net//software_links

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, February 2009

– Burger, Schuler, Harmeling: Image Denoisng: Can Plain Neural Networks Compete with BM3D?, Computer Vision and Pattern Recognition, CVPR 2012,

REFERENCES

Applications of RNNs

– Mikolov “Statistical language models based on neural networks” PhD thesis 2012

– Boden “A guide to RNNs and backpropagation” Tech Report 2002

– Hochreiter, Schmidhuber “Long short term memory” Neural Computation 1997

– Graves “Offline arabic handwrting recognition with multidimensional neural networks” Springer 2012

– Graves “Speech recognition with deep recurrent neural networks” ICASSP 2013

REFERENCES

Deep Learning & Energy-Based Models

– Y. 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

– M. Ranzato Ph.D. Thesis “Unsupervised Learning of Feature Hierarchies” NYU 2009

Practical guide

– Y. LeCun et al. Efficient BackProp, Neural Networks: Tricks of the Trade, 1998

– L. Bottou, Stochastic gradient descent tricks, Neural Networks, Tricks of the Trade Reloaded, LNCS 2012.

– Y. Bengio, Practical recommendations for gradient-based training of deep architectures, ArXiv 2012