Deep Learning & Convolutional Networks In Vision

VRML, Paris 2013-07-23

Yann LeCun
Center for Data Science & Courant Institute, NYU
yann@cs.nyu.edu
http://yann.lecun.com

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 →

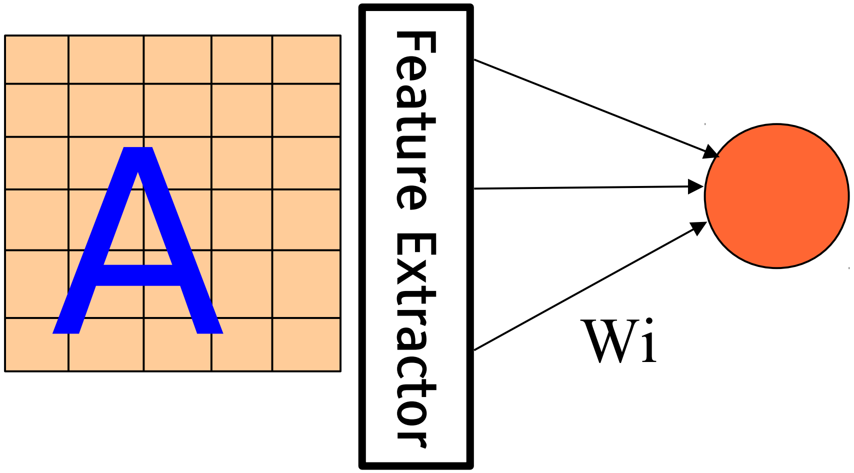
- **End-to-end learning / Feature learning / Deep learning**
  - Trainable features (or kernel) + trainable classifier

Trainable Feature Extractor → Trainable Classifier →

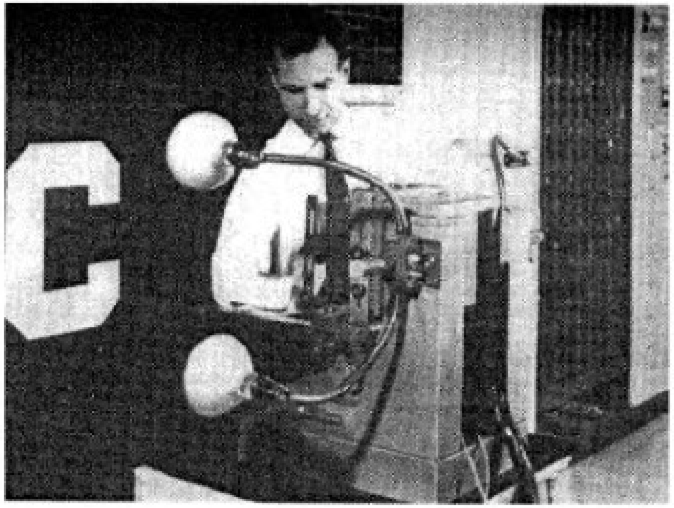
This Basic Model has not evolved much since the 50's

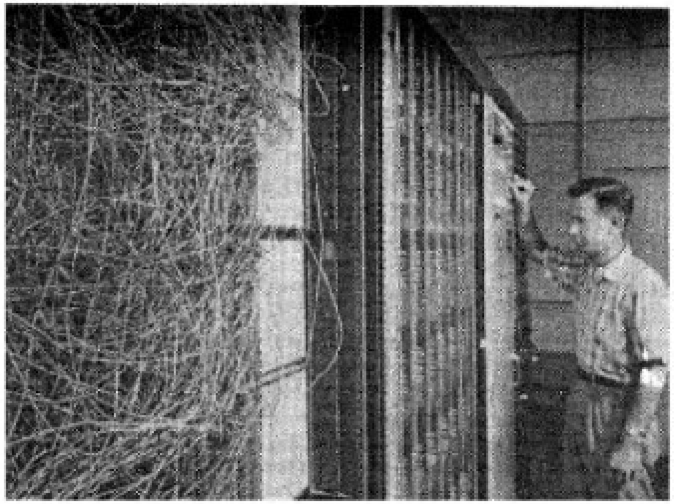
Y LeCun

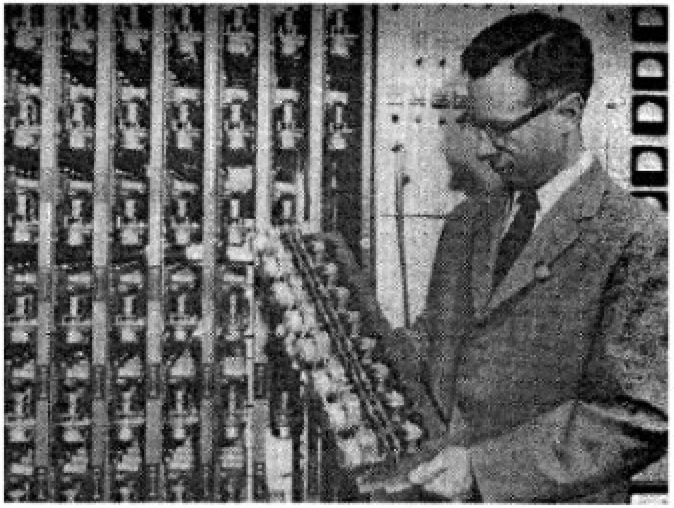
- The first learning machine: the Perceptron
  - Built at Cornell in 1960
- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor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



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





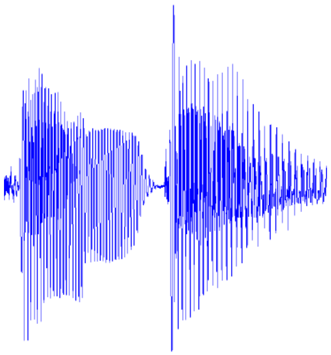


Architecture of “Mainstream” Pattern Recognition Systems

Y LeCun

Modern architecture for pattern recognition

- Speech recognition: early 90's – 2011



MFCC → Mix of Gaussians → Classifier →

fixed | unsupervised | supervised

- Object Recognition: 2006 - 2012

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

fixed | unsupervised | supervised

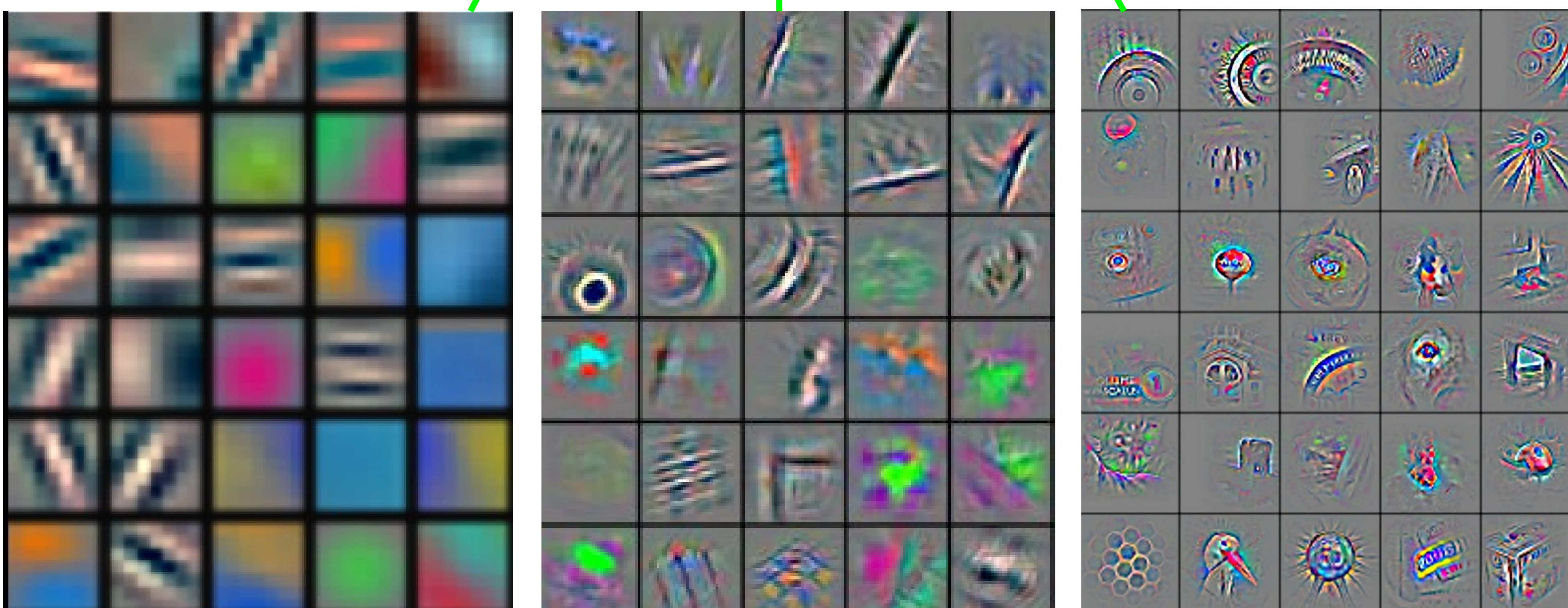
Low-level Features | Mid-level Features

Deep Learning = Learning Hierarchical Representations

Y LeCun

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

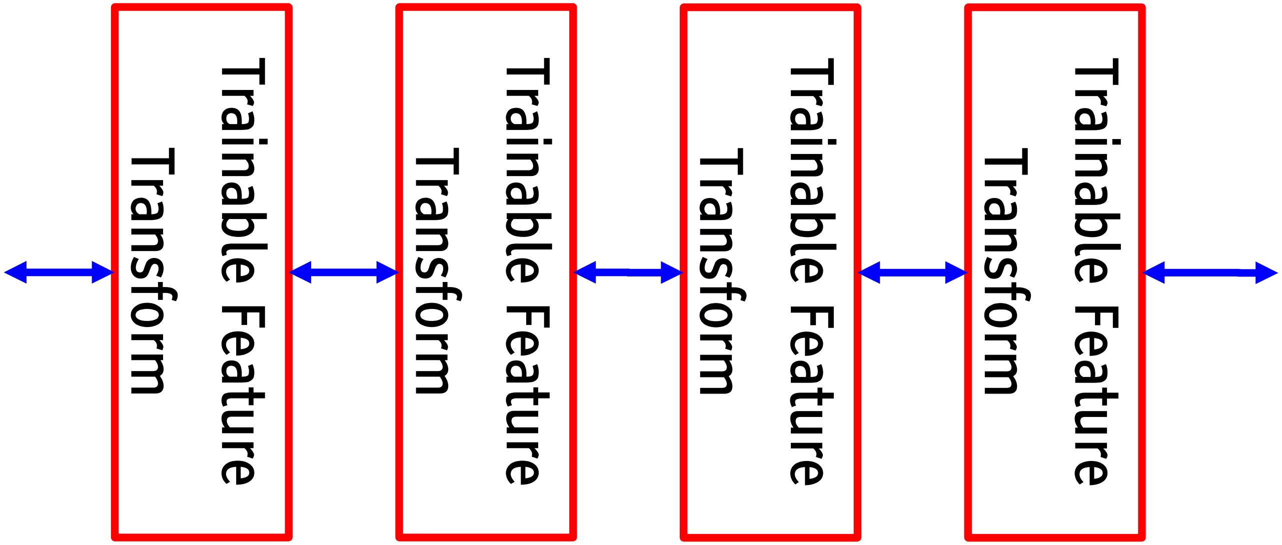
Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier →



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

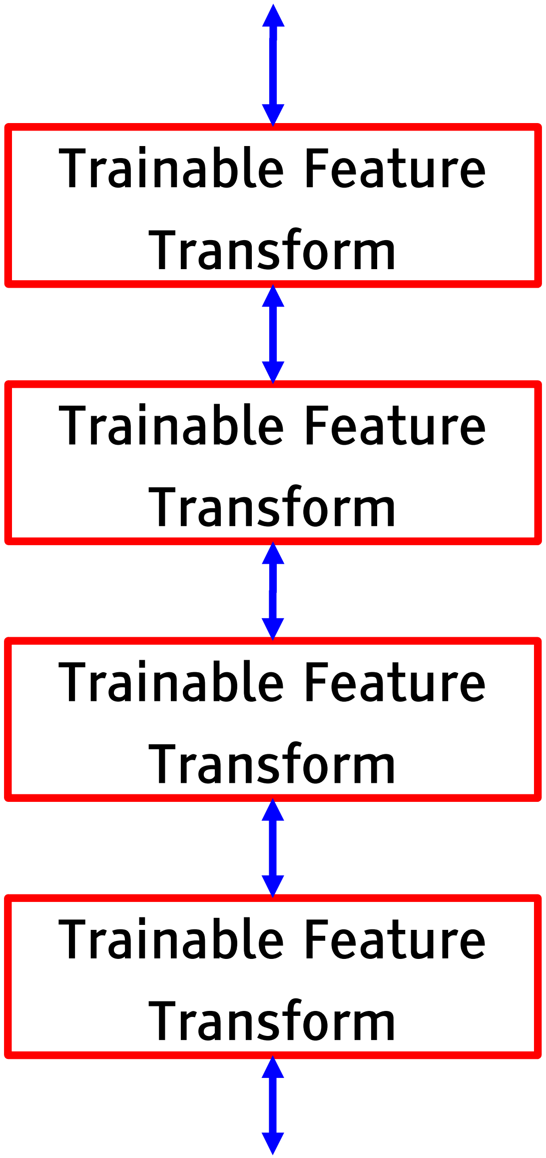
Trainable Feature Hierarchy

- **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



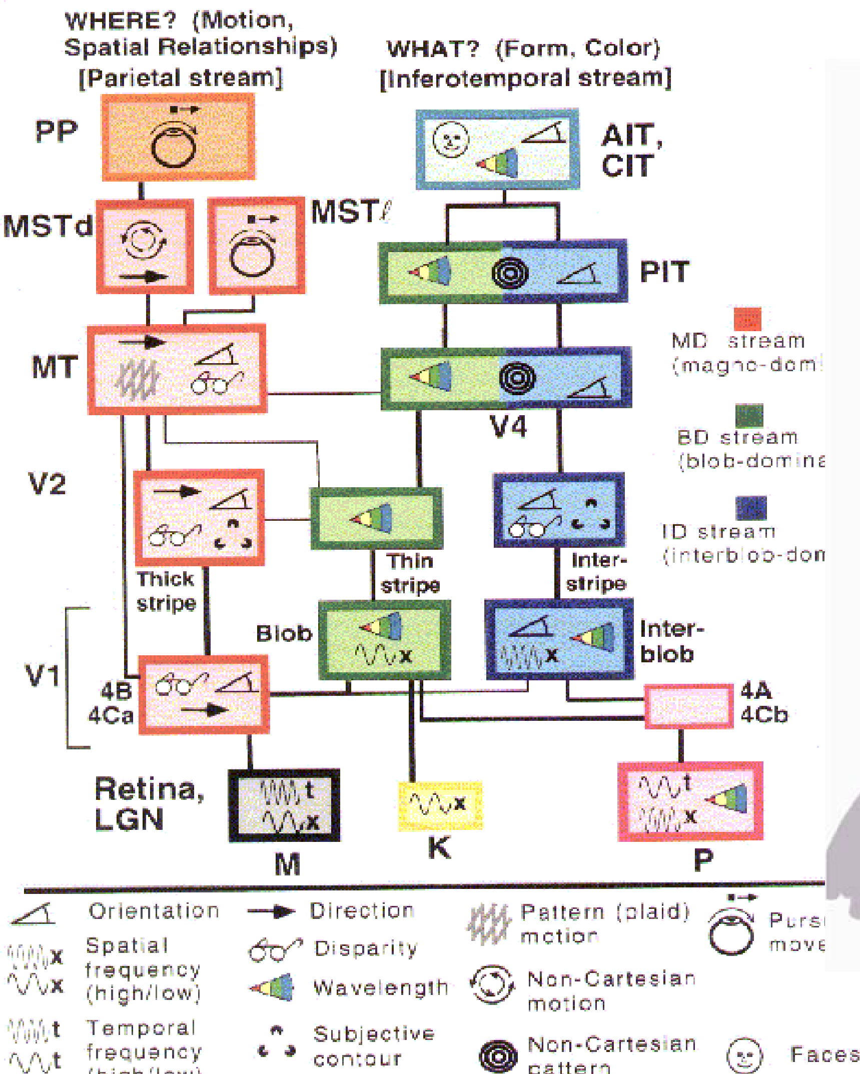
Learning Representations: a challenge for ML, CV, AI, Neuroscience, Cognitive Science...

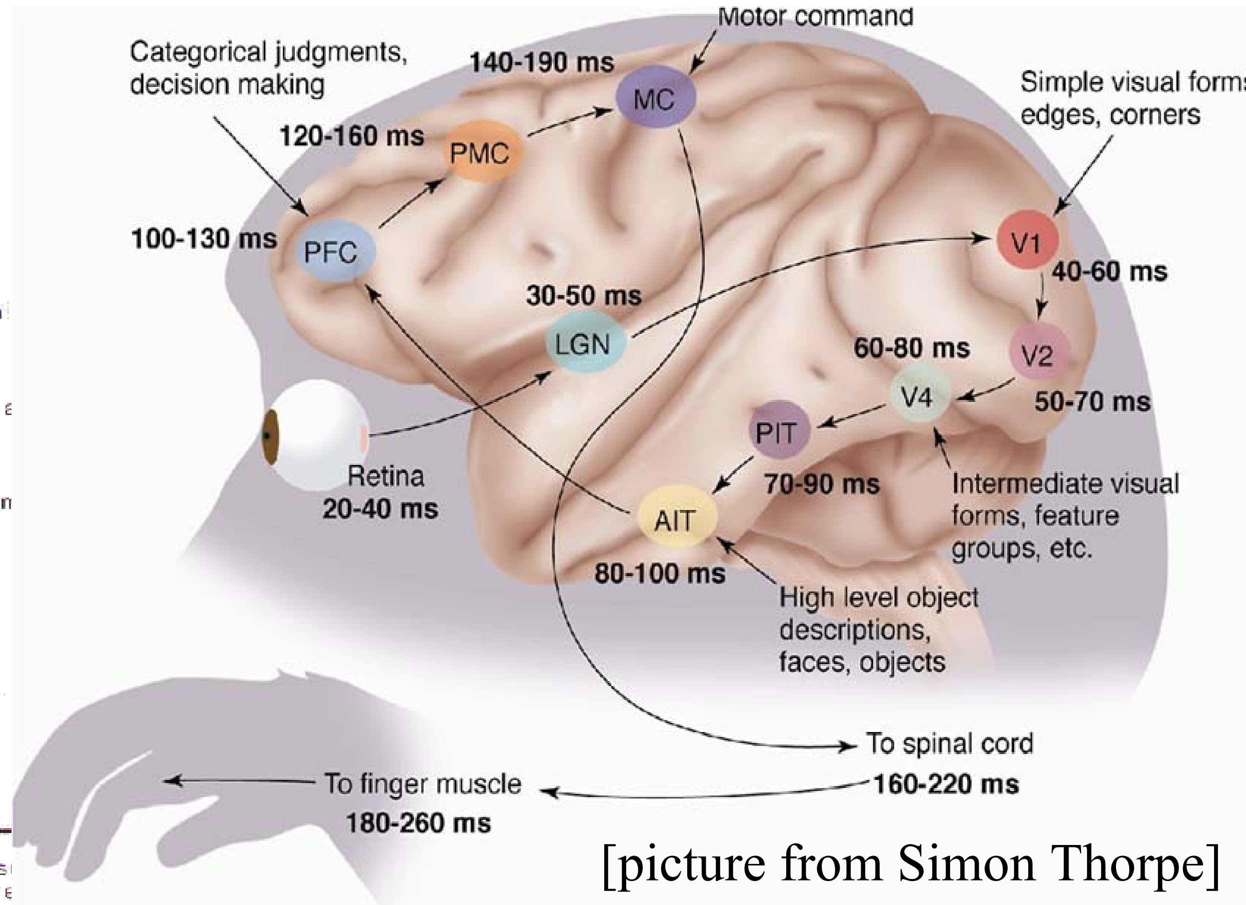
- **How do we learn representations of the perceptual world?**
  - How can a perceptual system build itself by looking at the world?
  - How much prior structure is necessary
- **ML/AI: how do we learn features or feature hierarchies?**
  - What is the fundamental principle? What is the learning algorithm? 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**
  - or perhaps with a few algorithms



The Mammalian Visual Cortex is Hierarchical

- 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

- 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 (unless you are french).

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

Trainable Feature Transform

Trainable Feature Transform

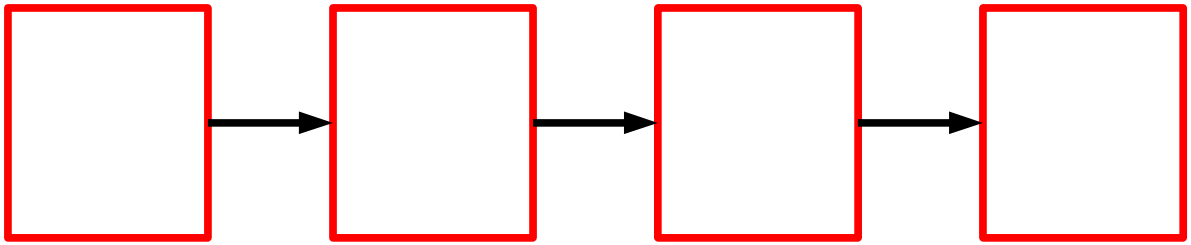
Trainable Classifier/ Predictor

Learned Internal Representations

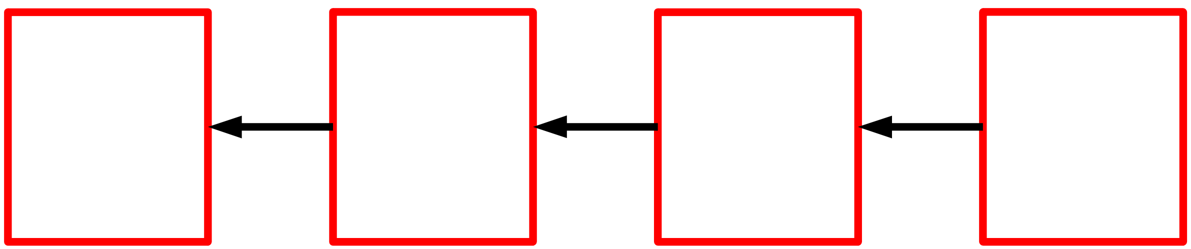
- **How can we make all the modules trainable and get them to learn appropriate representations?**

Three Types of Deep Architectures

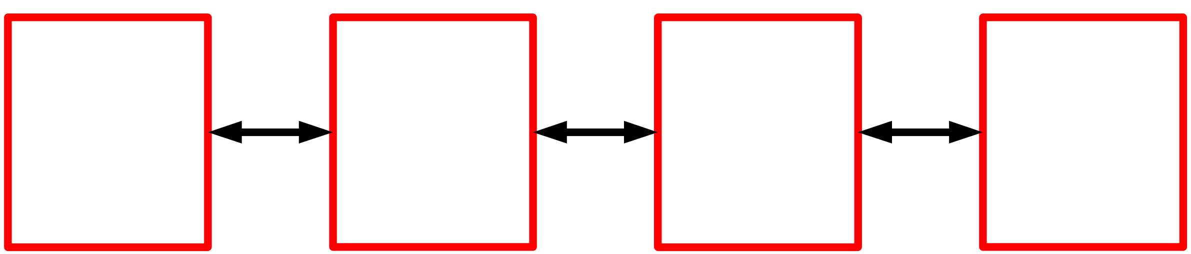
- Feed-Forward: multilayer neural nets, convolutional nets



- Feed-Back: Stacked Sparse Coding, Deconvolutional Nets [Zeiler et al.]



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



Three Types of Training Protocols

- **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?

- 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...)

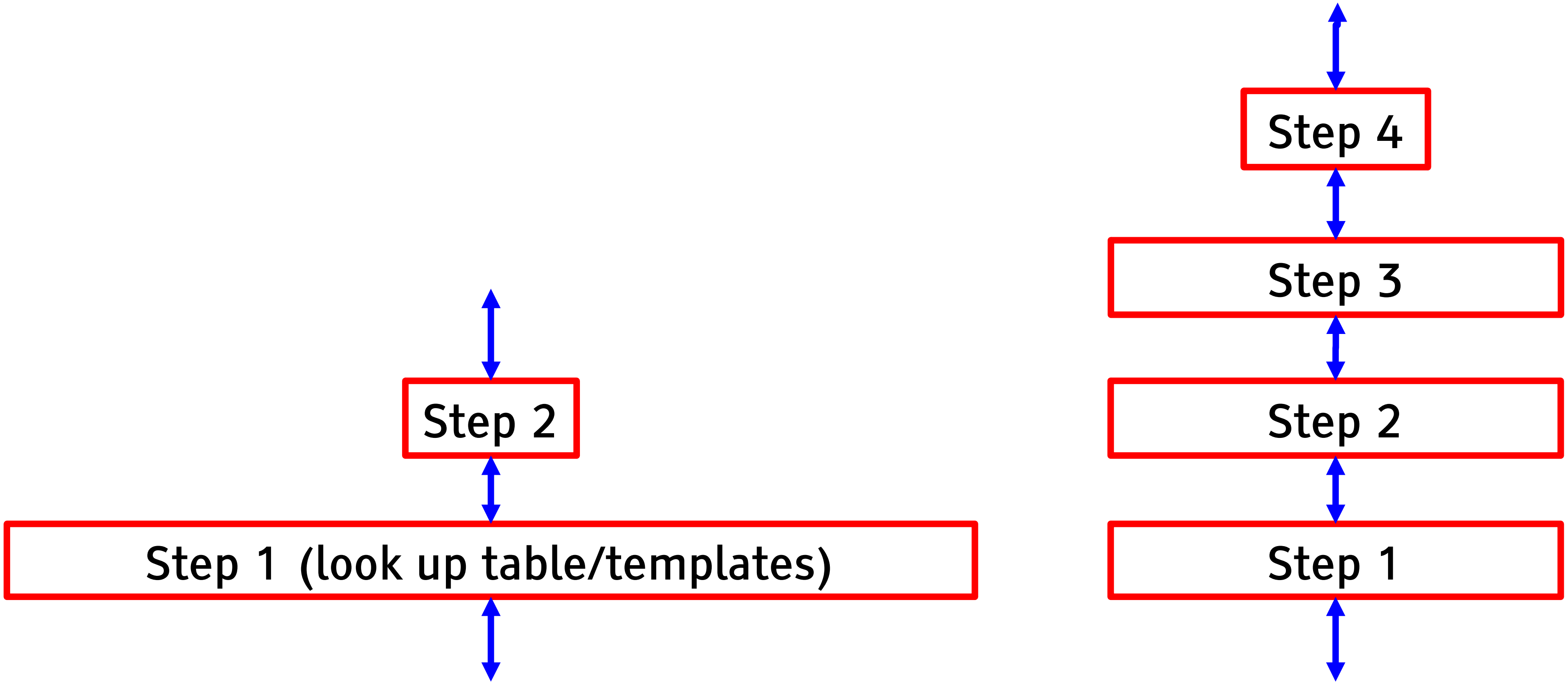
Why would deep architectures be more efficient?

[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).....

Shallow vs Deep == lookup table vs multi-step algorithm

- **"shallow & wide" vs "deep and narrow" == "more memory" vs "more time"**
  - Look-up table vs algorithm
  - Few functions can be computed in two steps without an exponentially large lookup table
  - Using more than 2 steps can reduce the "memory" by an exponential factor.

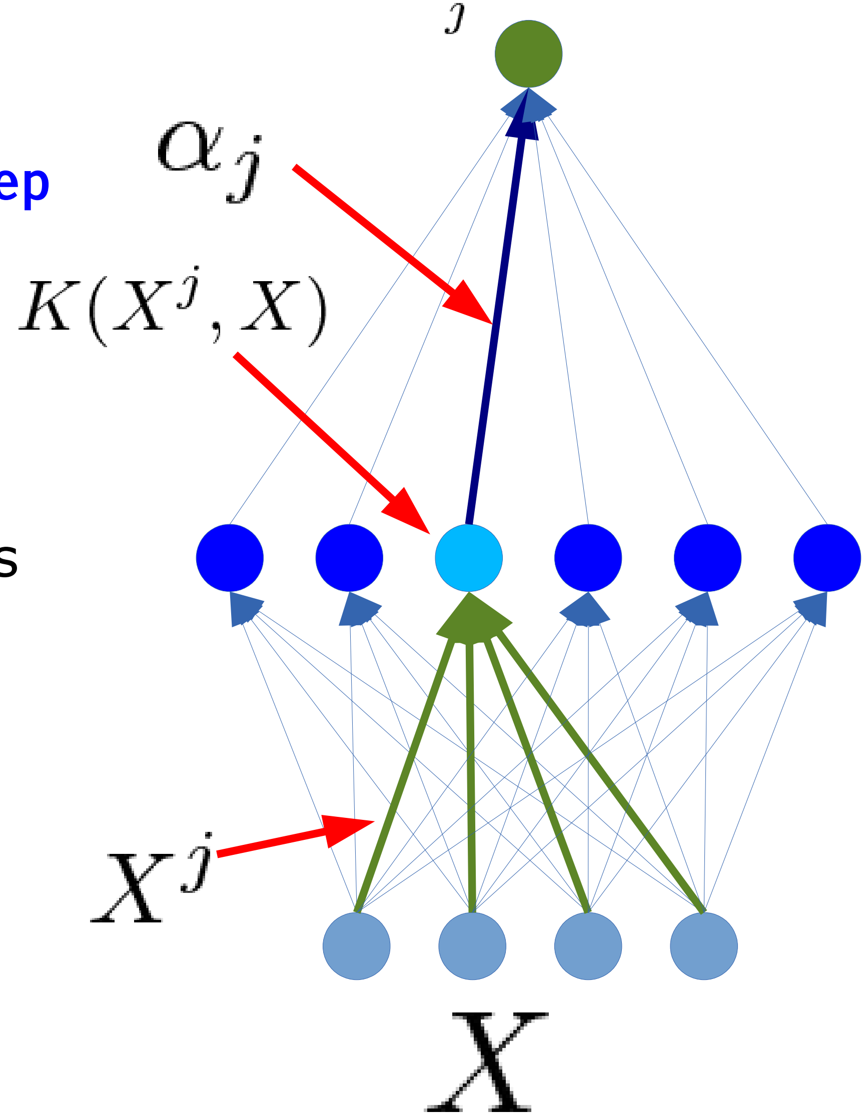


Which Models are Deep?

Y LeCun

- **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.
  - "glorified template matching"
- **Classification trees are not deep**
  - No hierarchy of features. All decisions are made in the input space

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

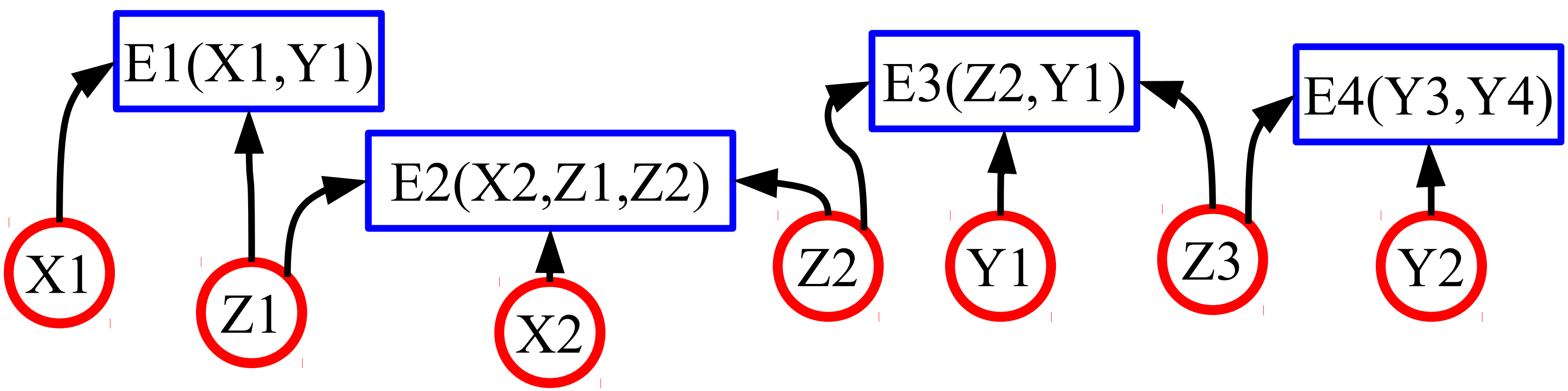


Are Graphical Models Deep?

Y LeCun

- 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...

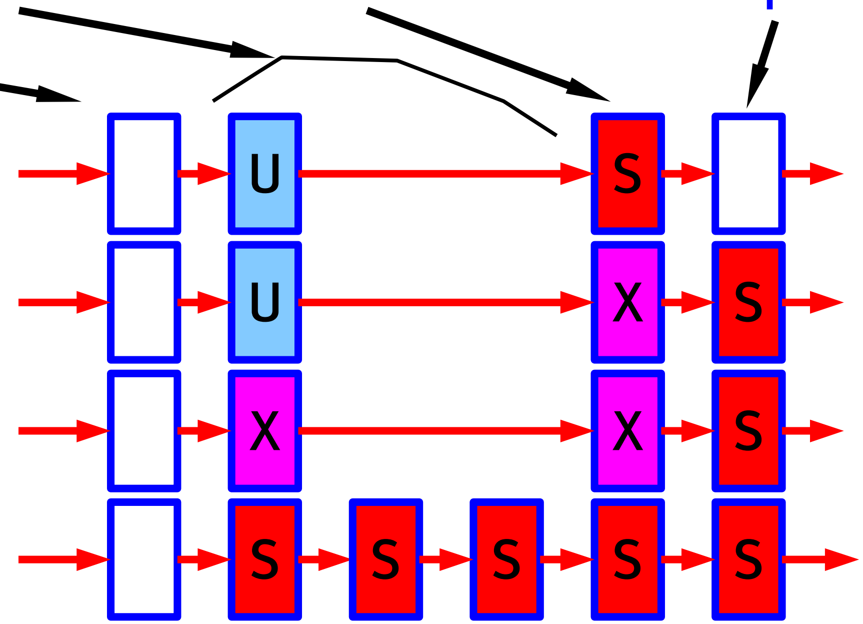
In Many Fields, Feature Learning Has Caused a Revolution (methods used in commercially deployed systems)

- **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

Y LeCun

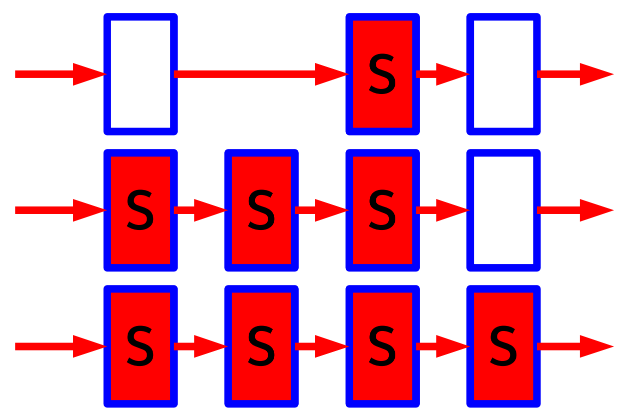
In Several Fields, Feature Learning Has Caused Revolutions: Speech Recognition, Handwriting Recogntiion

- U= unsupervised, S=supervised, X=unsupervised+supervised
- Low-level feat. → mid-level feat. → classifier → contextual post-proc



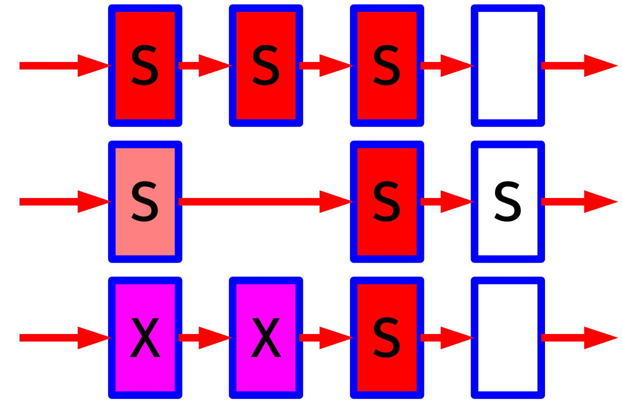
- Speech Recognition
  - Early 1980s: Dyn. time Warping
  - Late 1980s: Gaussian Mix. Model
  - 1990s: discriminative GMM
  - 2010: deep neural nets

- Handwriting Recognition and OCR
  - Early 80's: features+classifier
  - Late 80's: supervised convnet
  - Mid 90's: convnet+CRF

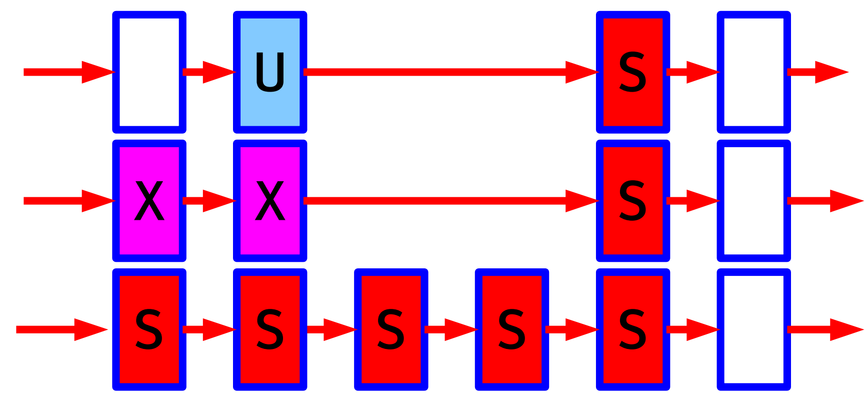


In Several Fields, Feature Learning Has Caused Revolutions: Object Detection, Object Recognition, Scene Labeling

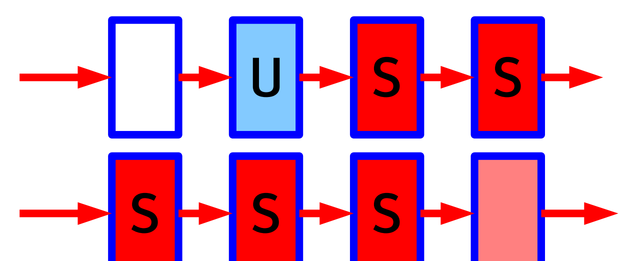
- Face & People Detection (1993-now)
  - Supervised ConvNet on pixels (93, 94, 05, 07)
  - Selected Haar features + Adaboost (2001)
  - Unsup+Sup ConvNet on raw pixels (2011)



- Object Recognition
  - SIFT/HoG+sparse code+pool+SVM (06)
  - unsup+sup convnet (07,10)
  - supervised convnet (2012)



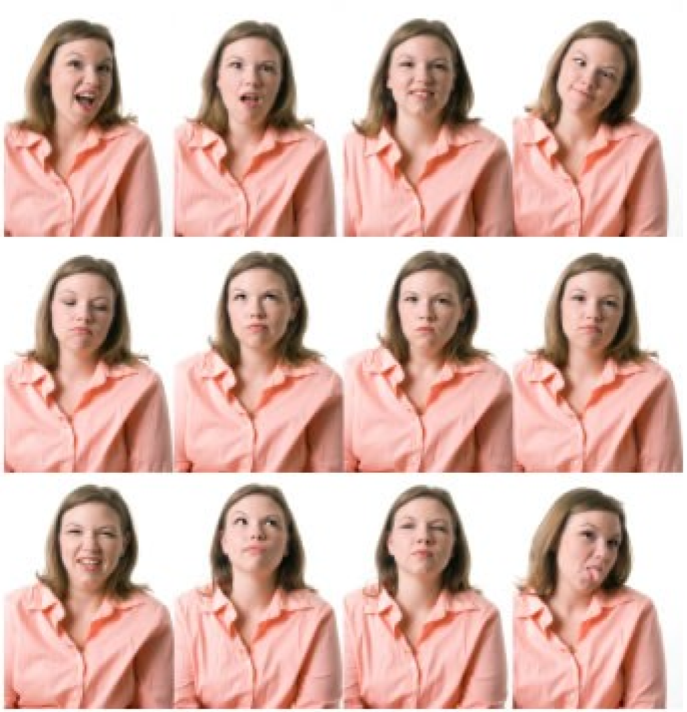
- Semantic Segmentation / scene labeling
  - unsup mid-lvl, CRF (2009, 10, 11, 12)
  - supervised convnet (2008, 12, 13)

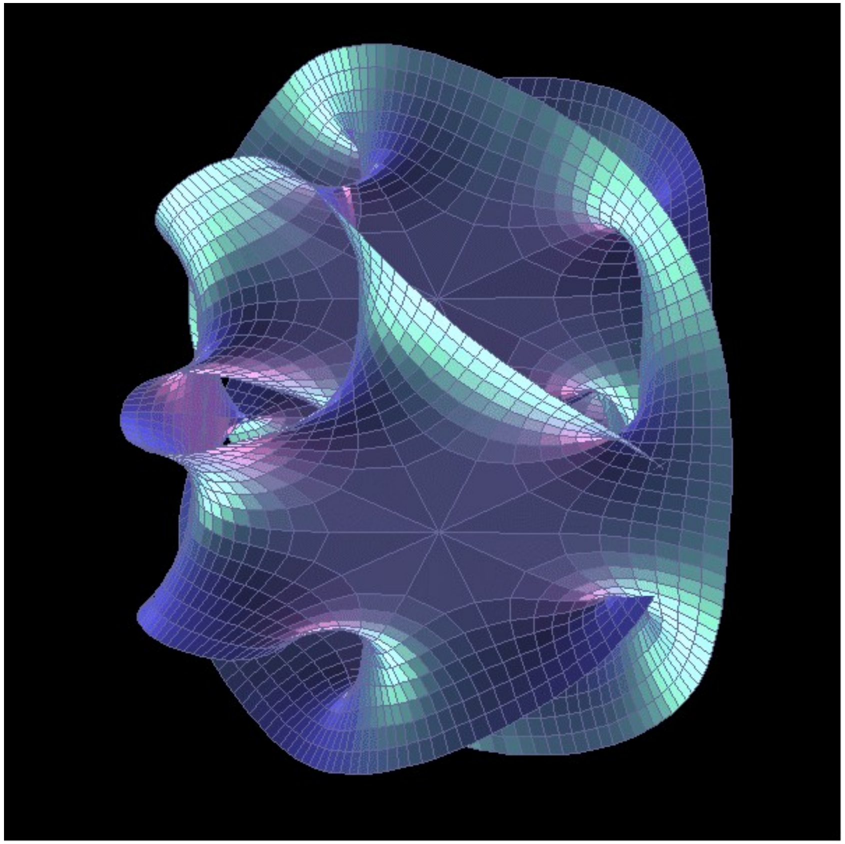


What Are
Good Feature?

The manifold hypothesis

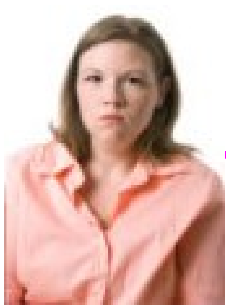
- Learning Representations of Data:
  - **Discovering & disentangling the independent explanatory factors**
- The Manifold Hypothesis:
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data





Discovering the Hidden Structure in High-Dimensional 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**



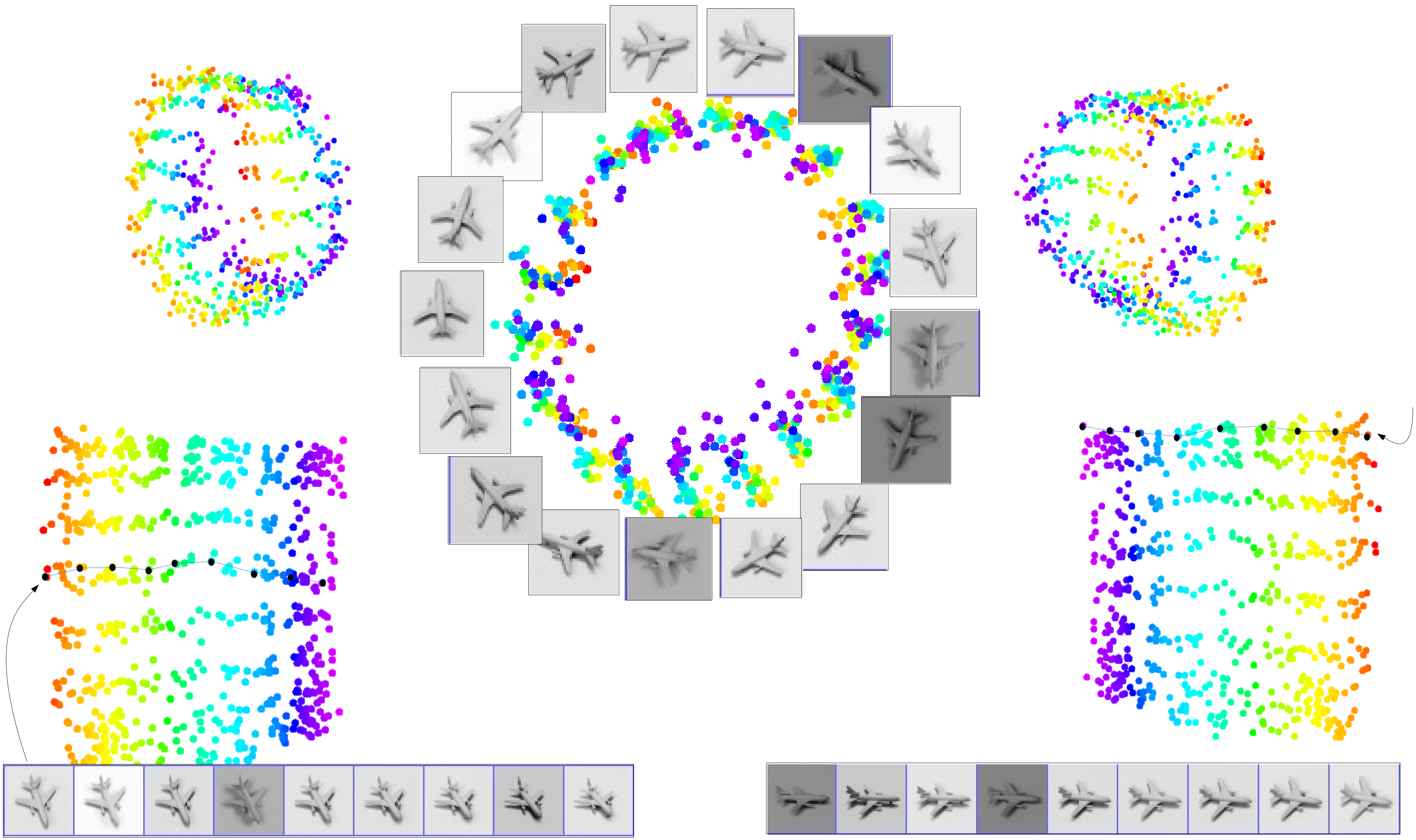
→ Ideal Feature Extractor →

$$\begin{bmatrix} 1.2 \\ -3 \\ 0.2 \\ -2\ldots \end{bmatrix}$$

Face/not face
Pose
Lighting
Expression

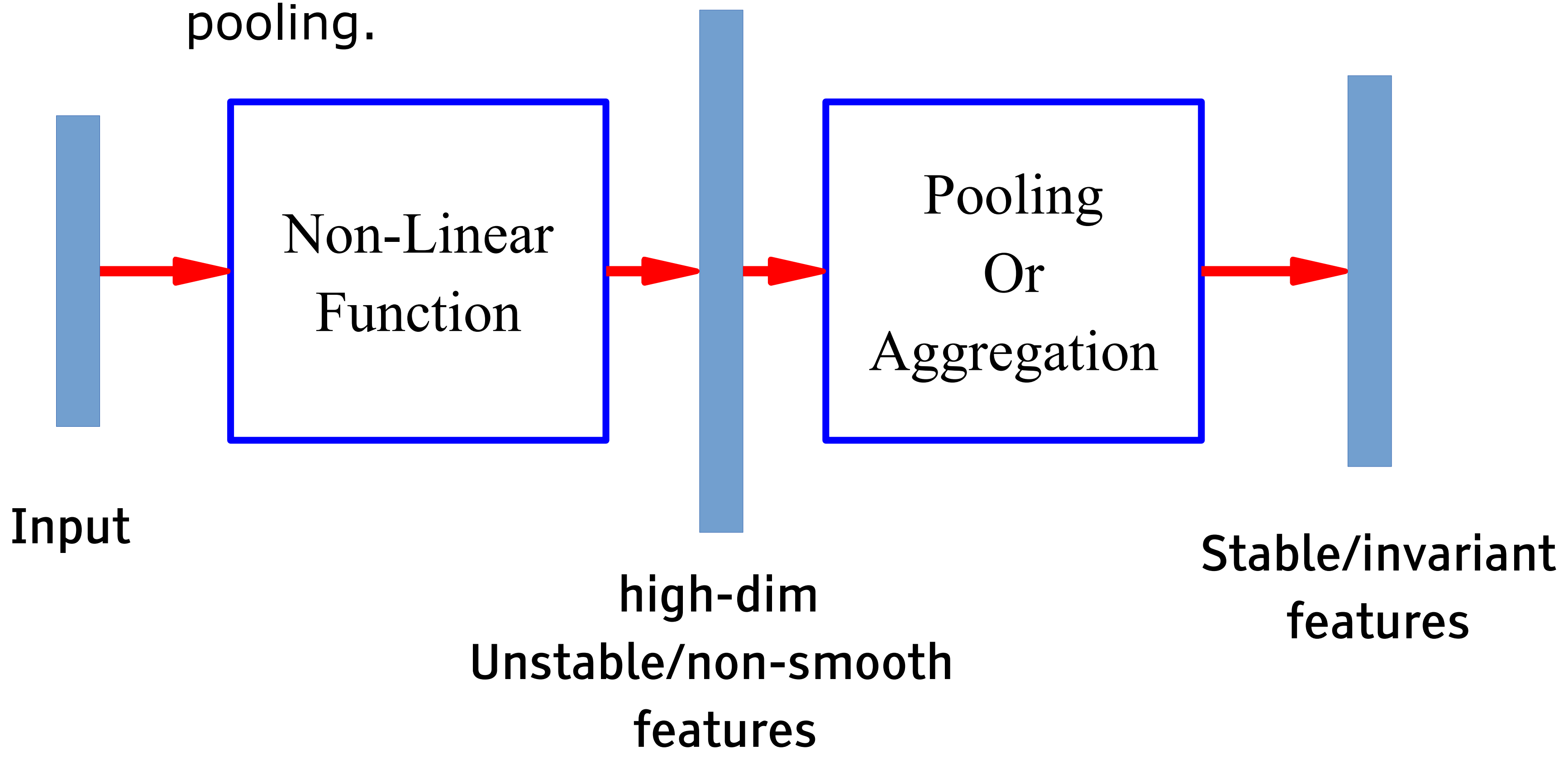
Data Manifold & Invariance: Some variations must be eliminated

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



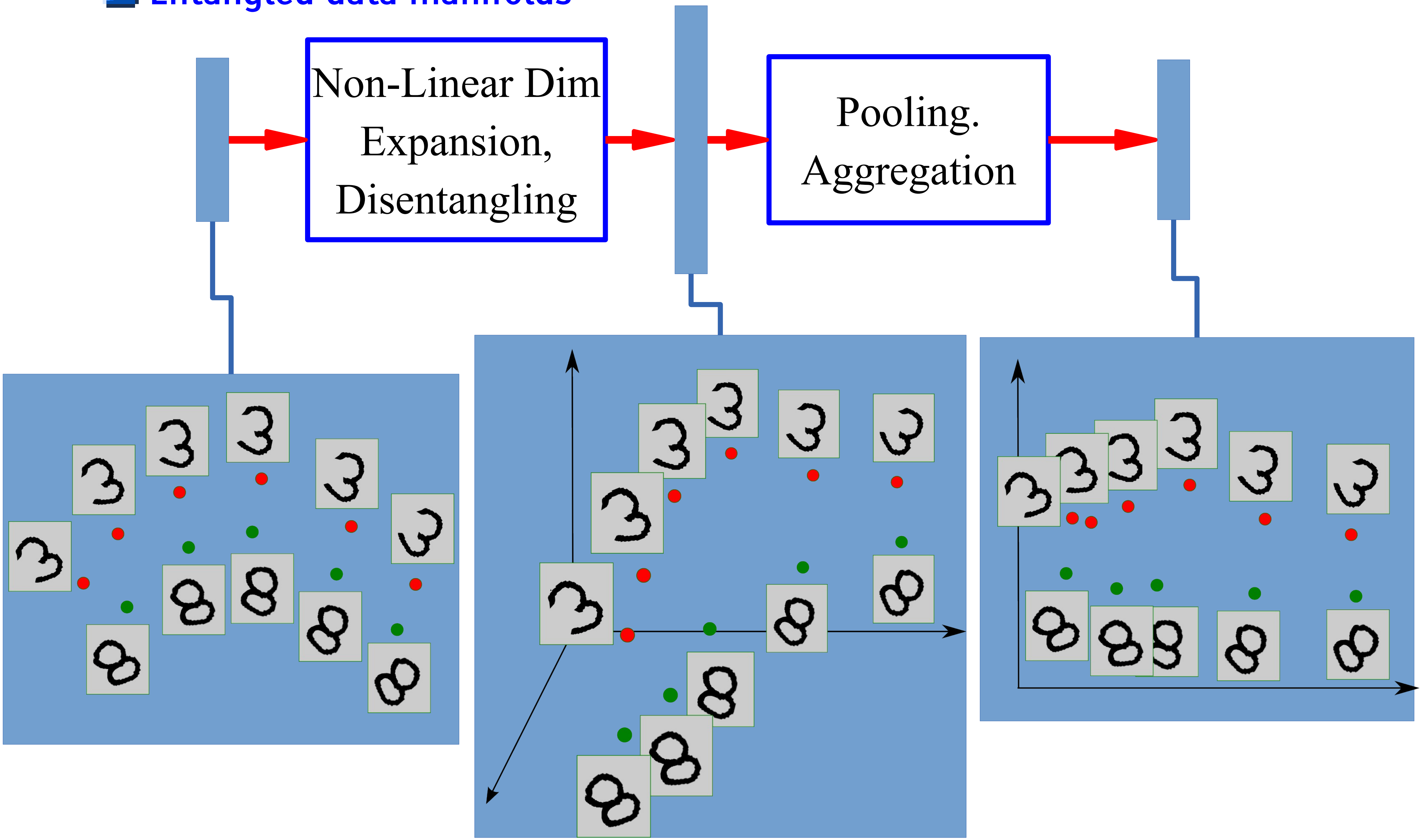
Basic Idea for Invariant Feature Learning

- 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

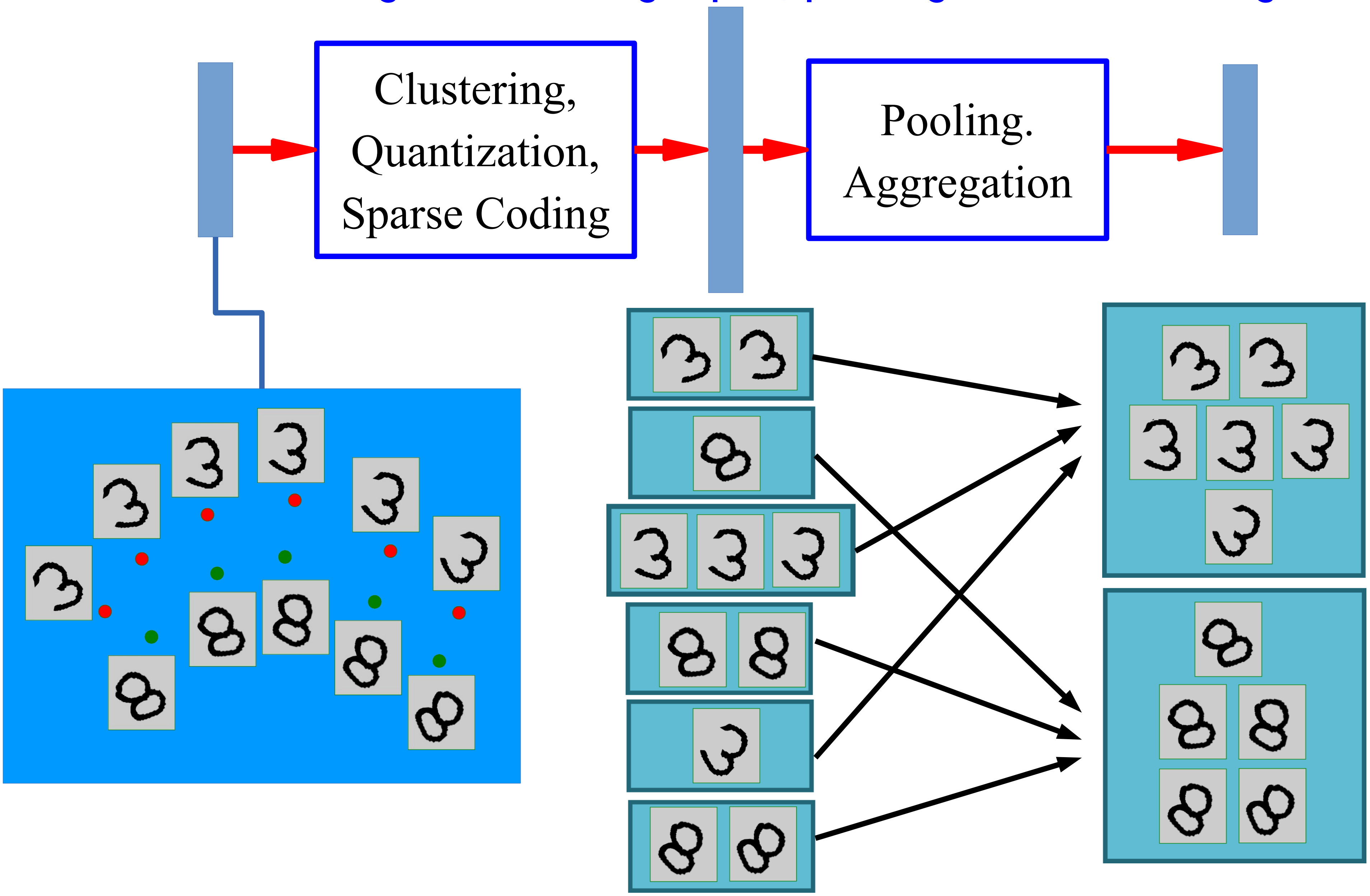
- Entangled data manifolds



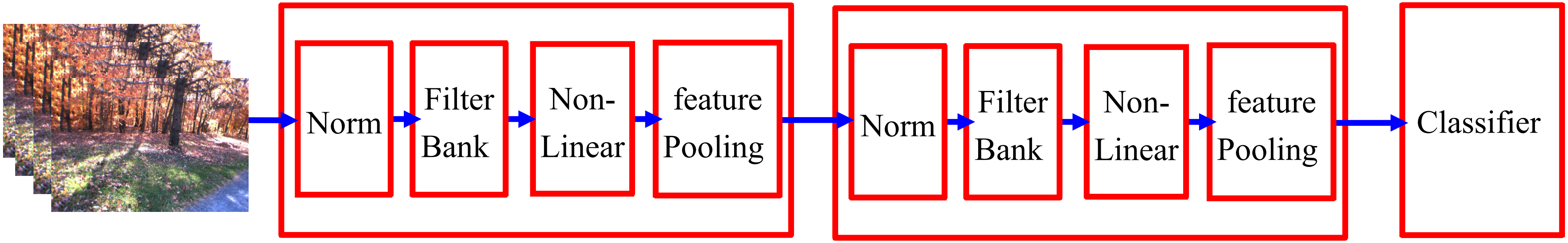
Sparse Non-Linear Expansion → Pooling

Y LeCun

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



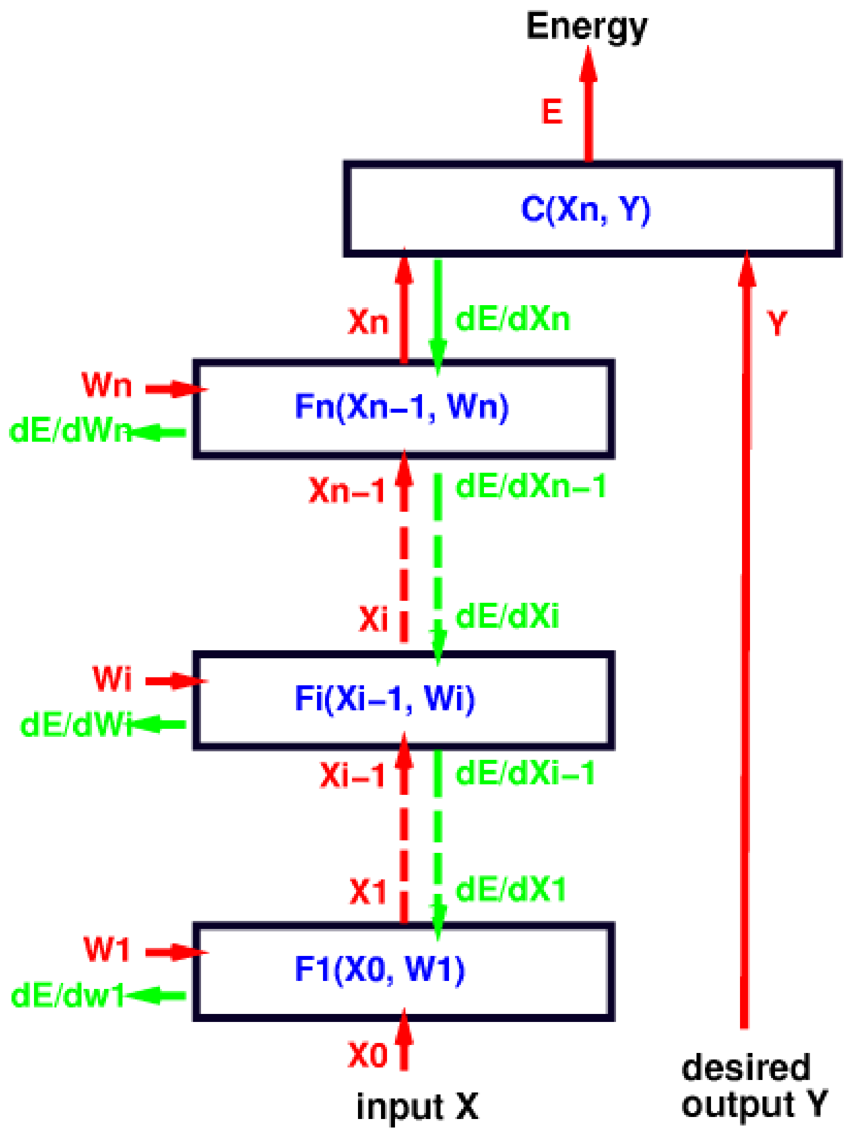
Overall Architecture:
Normalization → Filter Bank → Non-Linearity → Pooling



- **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



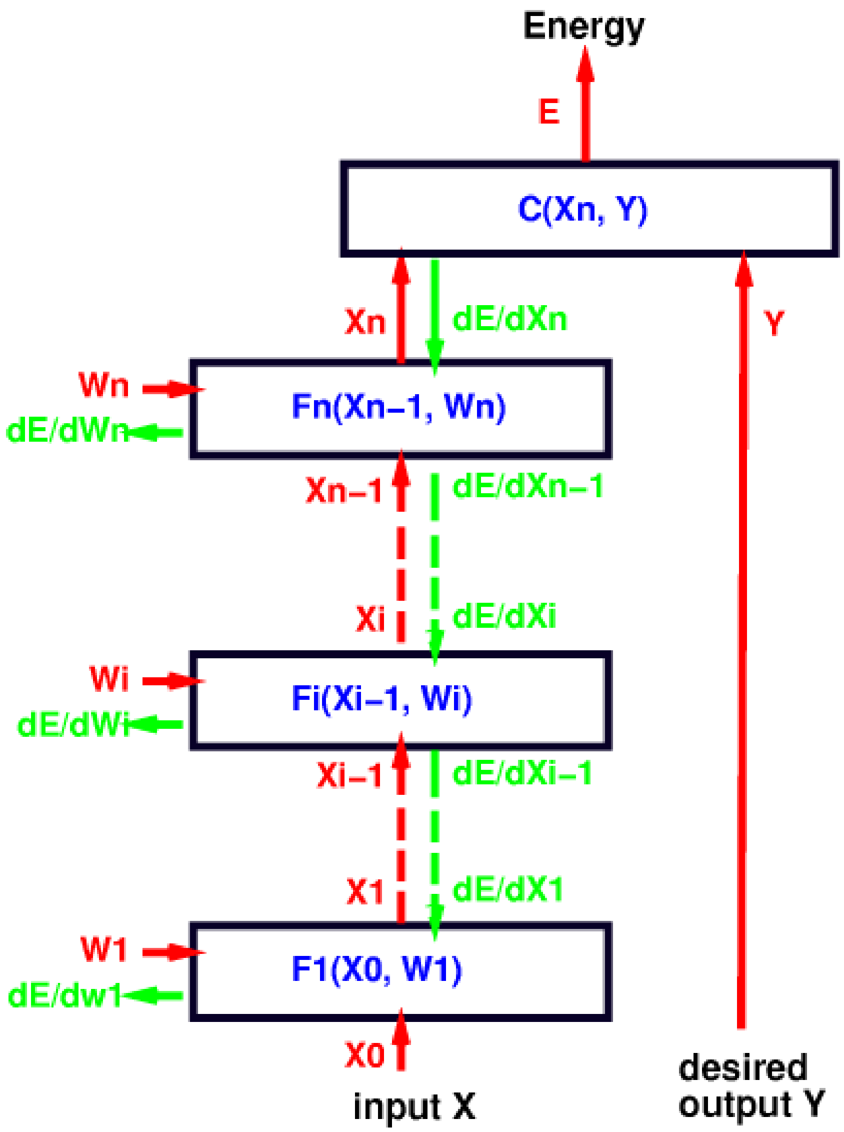
- Complex learning machines can be built by assembling modules into networks
- Simple example: sequential/layered feed-forward architecture (cascade)
- Forward Propagation:

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

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

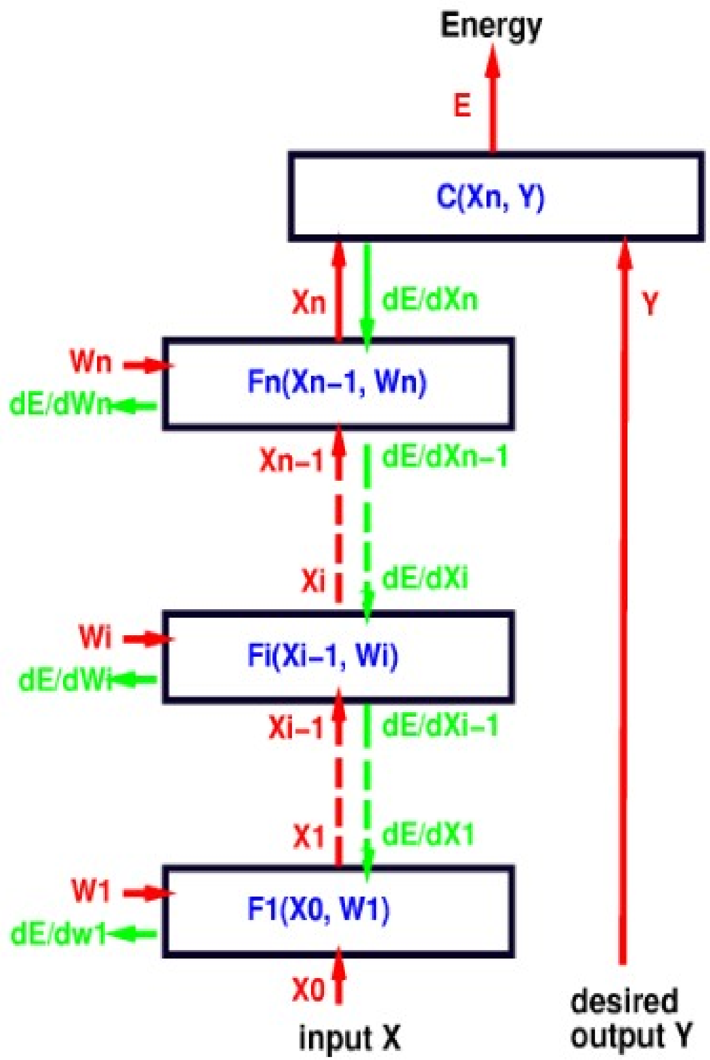
$$E(Y, X, W) = C(X_n, Y)$$

Multimodule Systems: Implementation



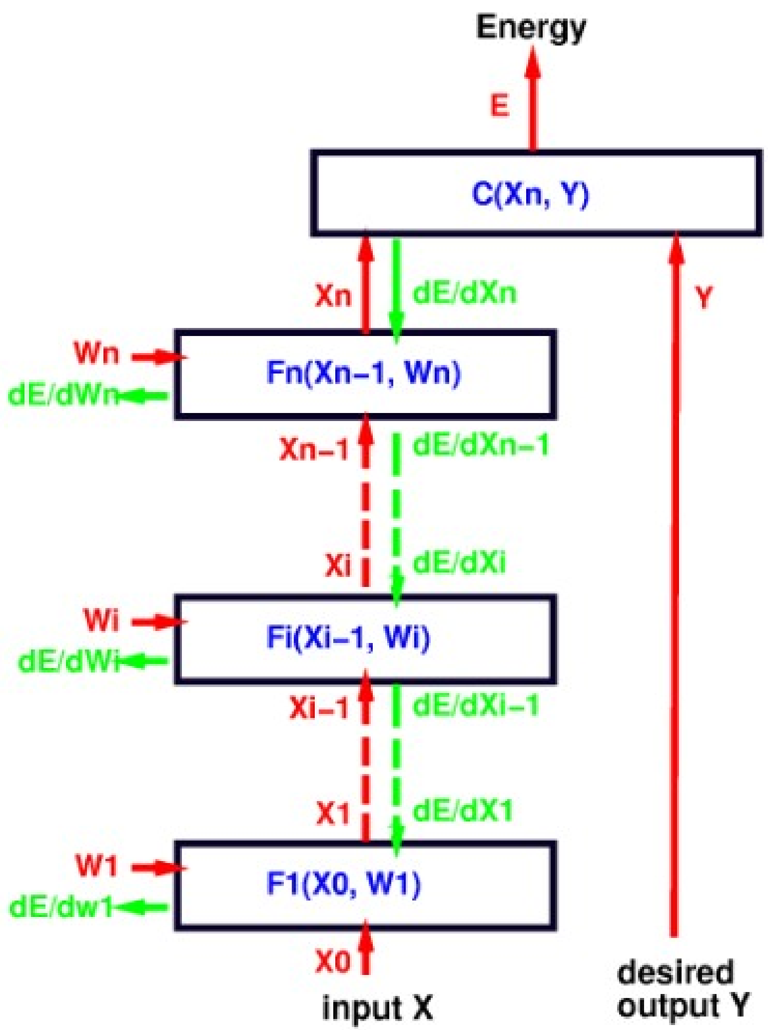
- **Each module is an object**
  - Contains trainable parameters
  - Inputs are arguments
  - Output is returned, but also stored internally
  - Example: 2 modules m1, m2
- **Torch7 (by hand)**
  - `hid = m1:forward(in)`
  - `out = m2:forward(hid)`
- **Torch7 (using the nn.Sequential class)**
  - `model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

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

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

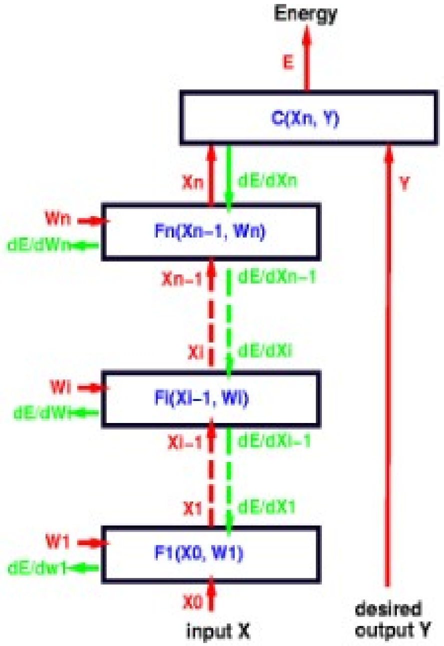
- $\frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i}$ is the *Jacobian matrix* of F_i with respect to W_i.

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

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

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

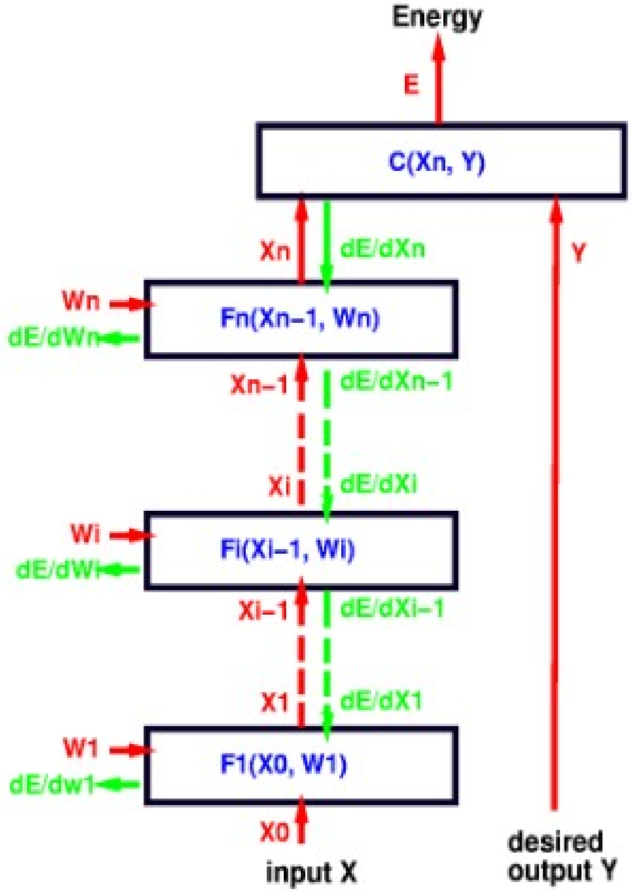
- we may prefer to write those equation with column vectors:

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

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

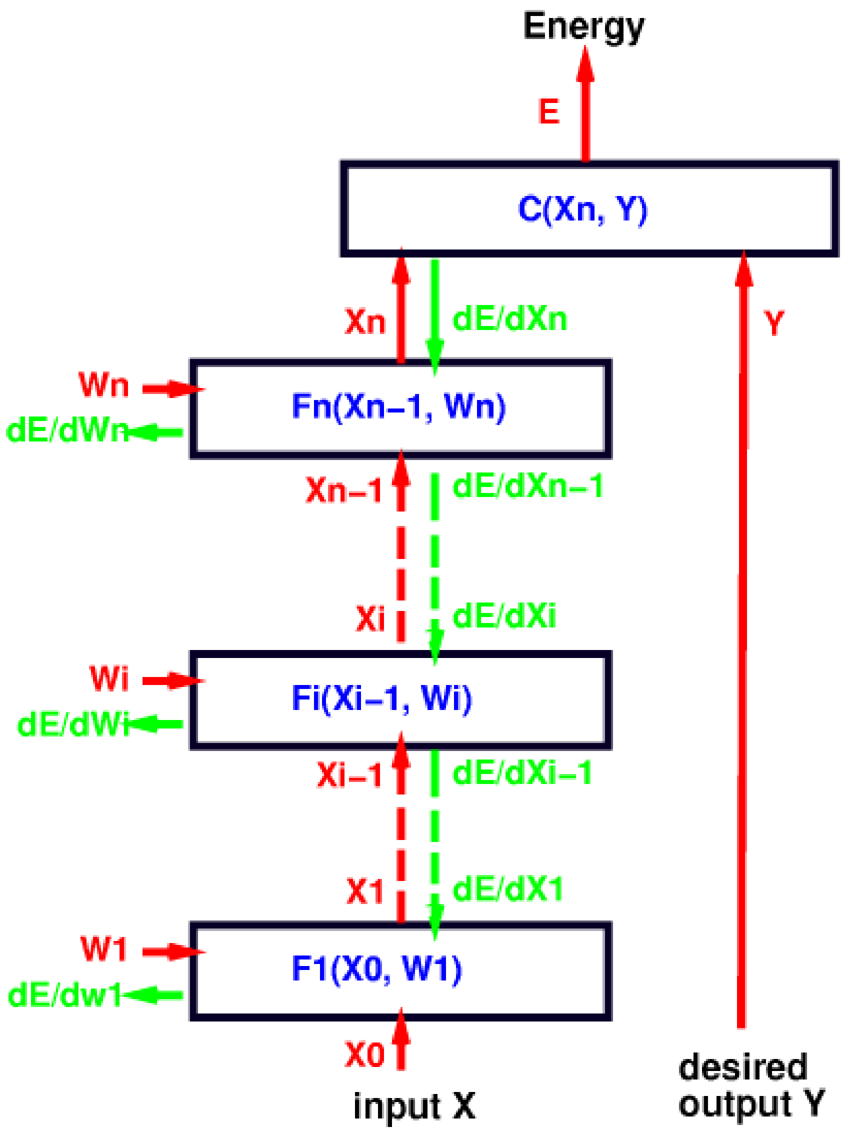
Back Propgation

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 $i \in [1, n]$.

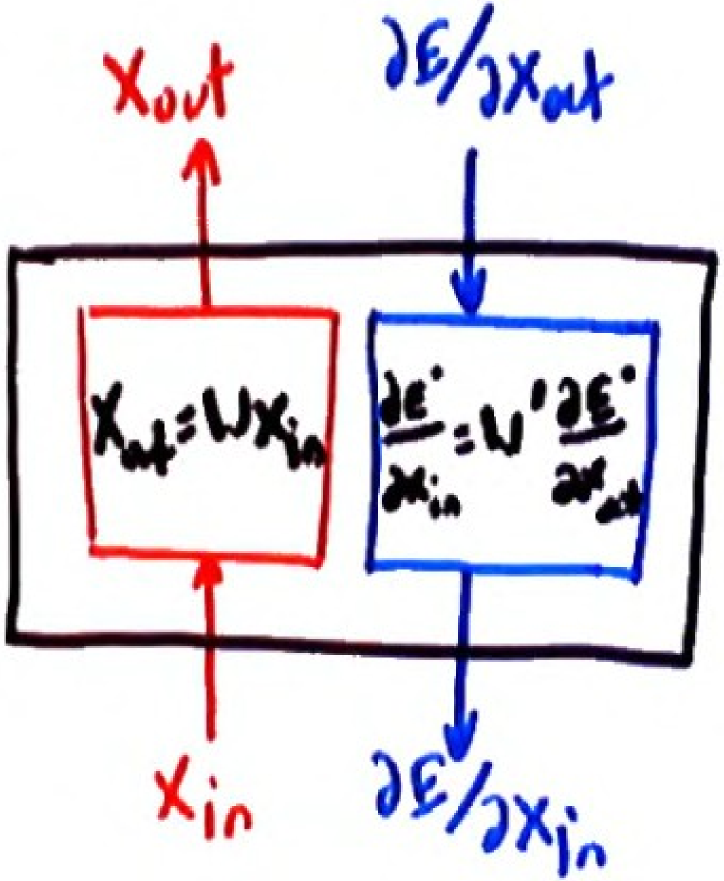
Multimodule Systems: Implementation



- **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

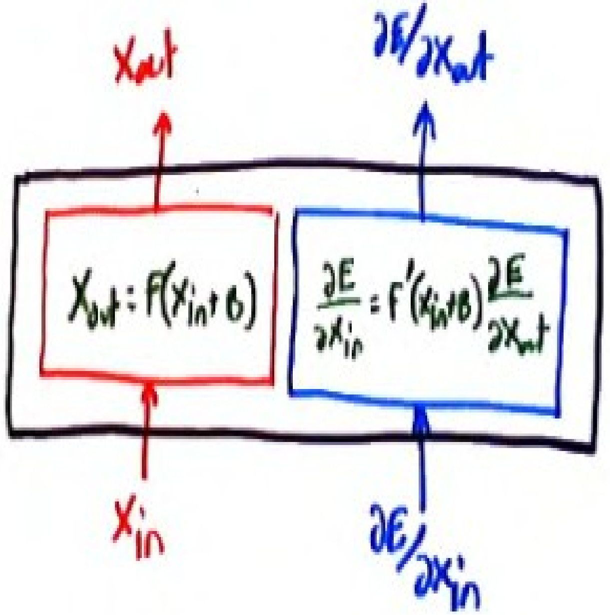
The input vector is multiplied by the weight matrix.



- 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:
  $$\frac{\partial E}{\partial W}' = \frac{\partial E}{\partial X_{\text{out}}}' X'_{in}$$

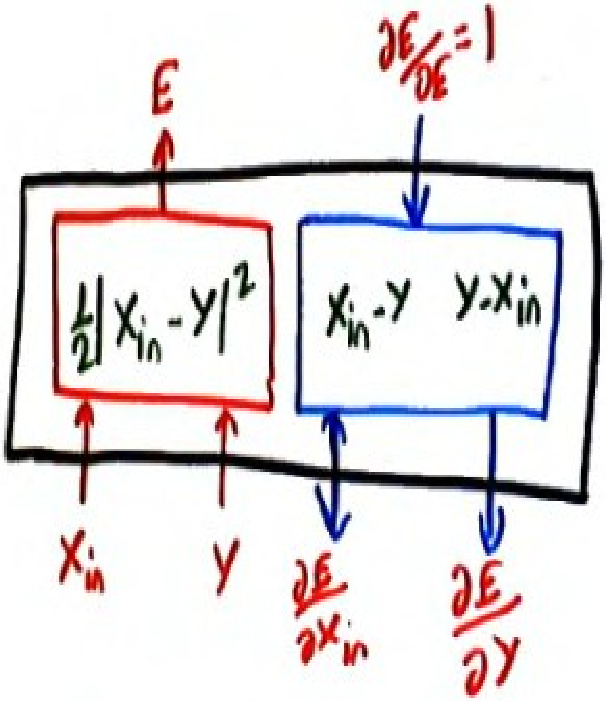
Tanh module (or any other pointwise function)

Y LeCun



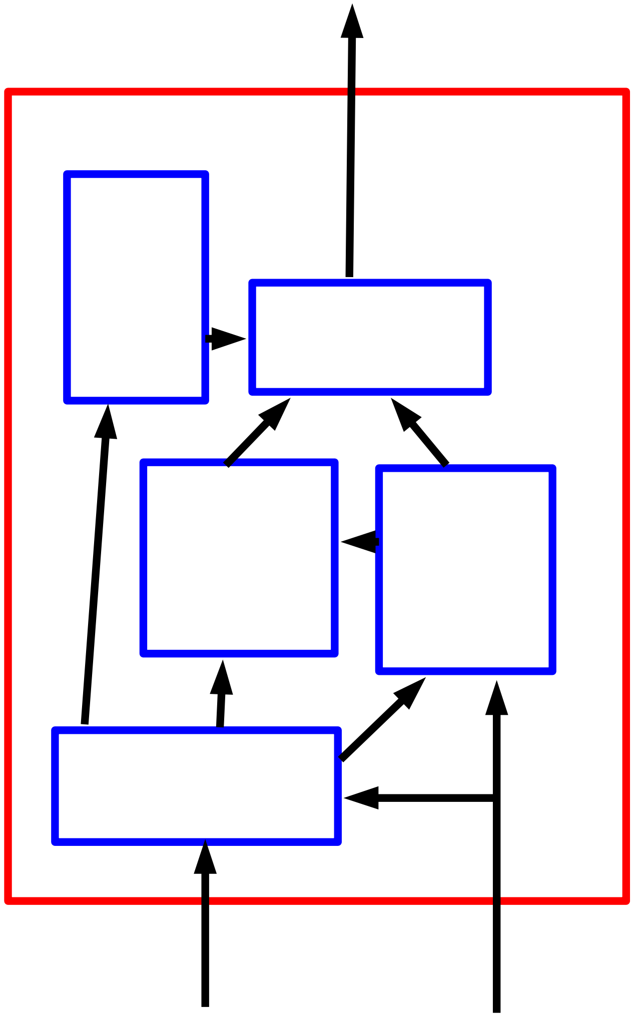
- fprop: $(X_{\text{out}})_i = \tanh((X_{\text{in}})_i + B_i)$
- bprop to input:
  $(\frac{\partial E}{\partial X_{\text{in}}})_i = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- bprop to bias:
  $\frac{\partial E}{\partial B_i} = (\frac{\partial E}{\partial X_{\text{out}}})_i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

Euclidean Distance Module



- 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

- **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-all | 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

Y LeCun

- Net for SVHN digit recognition
- 10 categories
- Input is 32x32 RGB (3 channels)
- 1500 hidden units

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

- 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

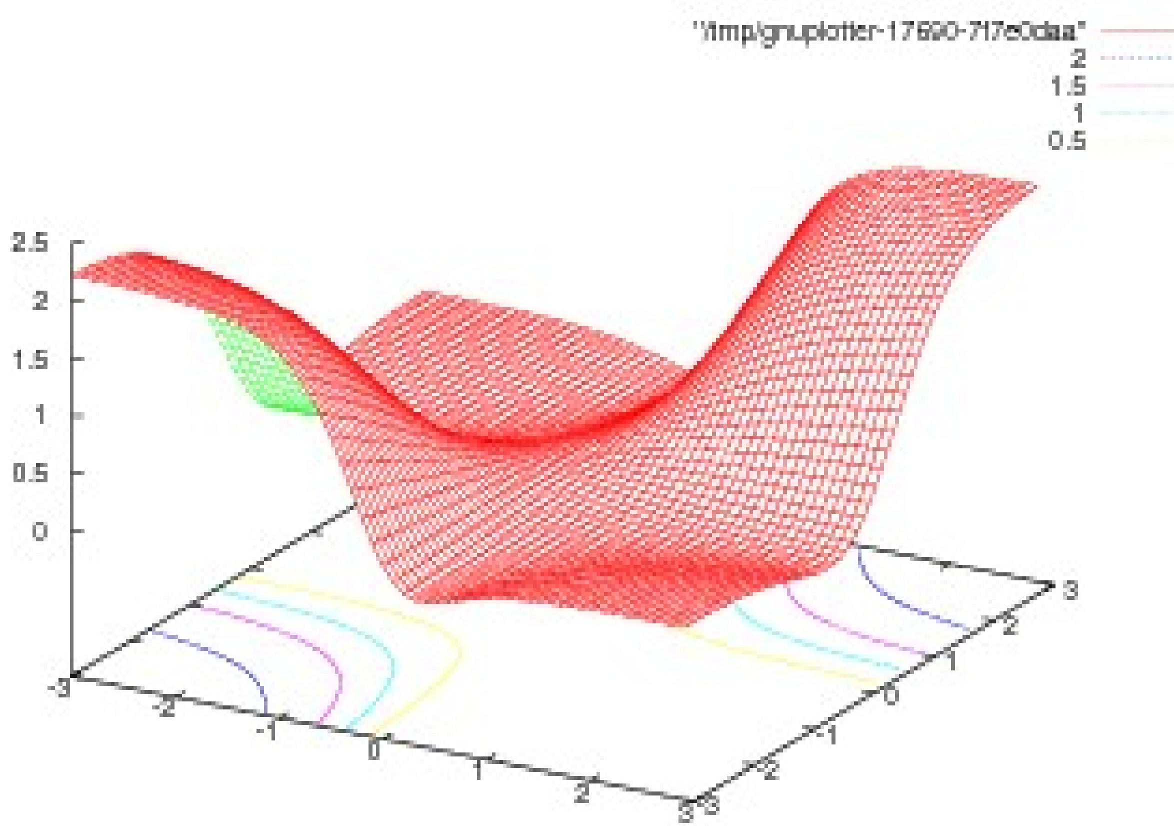

```
for t = 1,trainData:size(),batchSize do
  inputs,outputs = getNextBatch()
  local feval = function(x)
    parameters:copy(x)
    gradParameters:zero()
    local f = 0
    for i = 1,#inputs do
      local output = model:forward(inputs[i])
      local err = criterion:forward(output,targets[i])
      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

Deep Supervised Learning is Non-Convex

- **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:

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



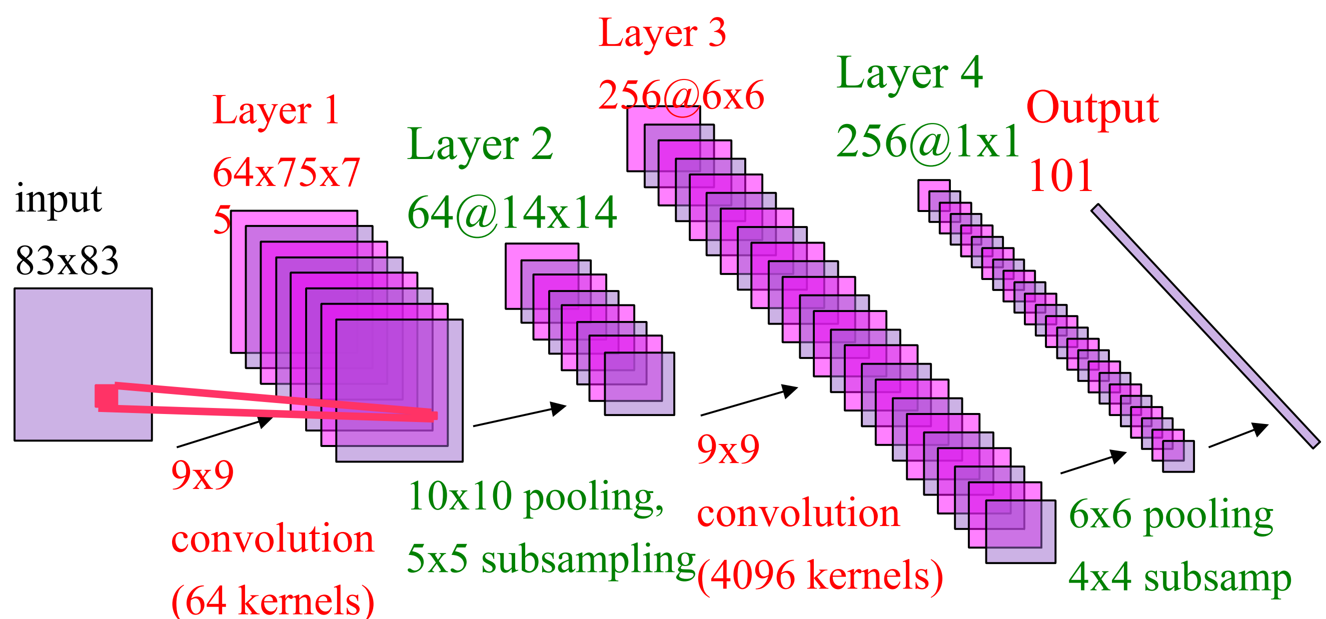
Backprop in Practice

- 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)

Convolutional Networks

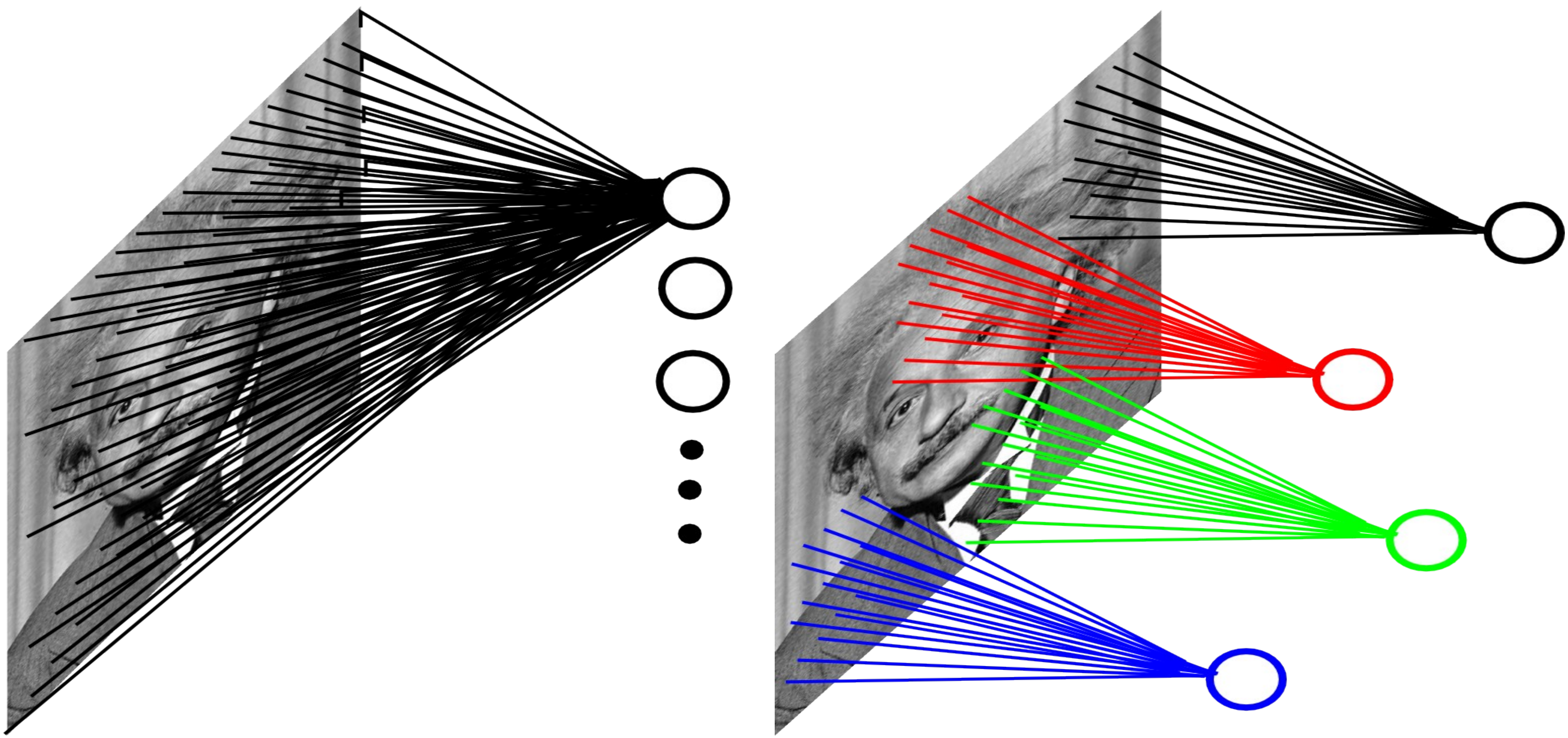
Convolutional Nets

- **Are deployed in many practical applications**
  - Image reco, speech reco, 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

- 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

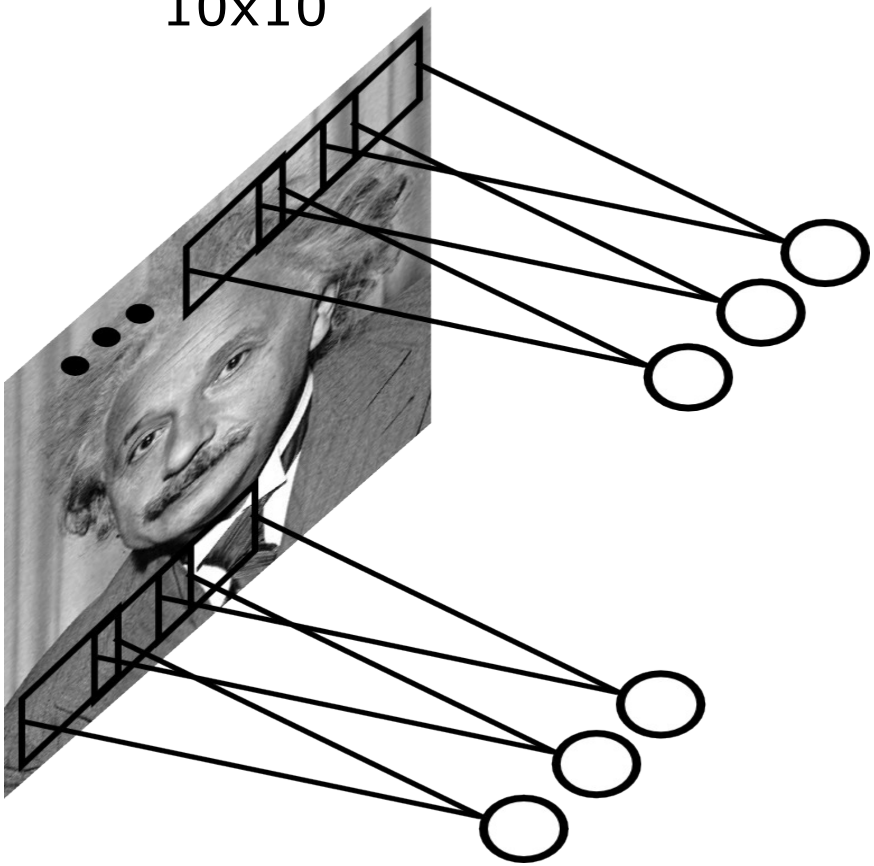
- Features that are useful on one part of the image and probably useful elsewhere.
- All units share the same set of weights
- Shift equivariant processing:
  - When the input shifts, the output also shifts but stays otherwise unchanged.
- Convolution
  - with a learned kernel (or filter)
  - Non-linearity: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

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

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

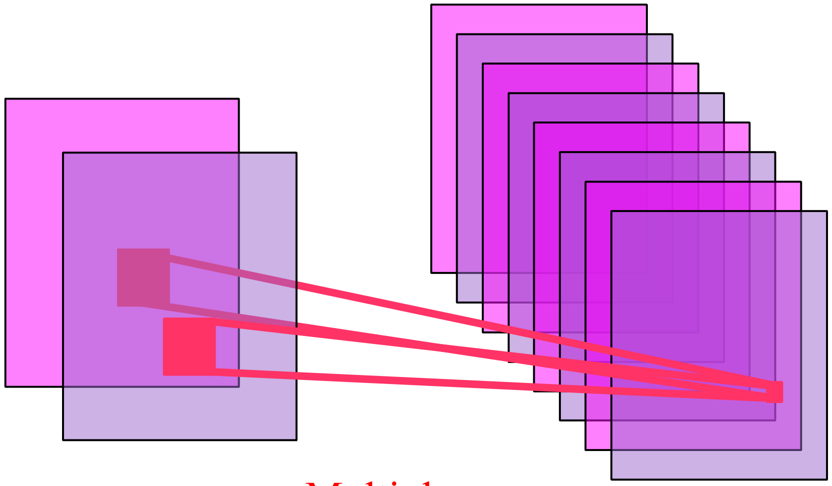
$$Z_{ij} = max\left(0, A_{ij}\right)$$

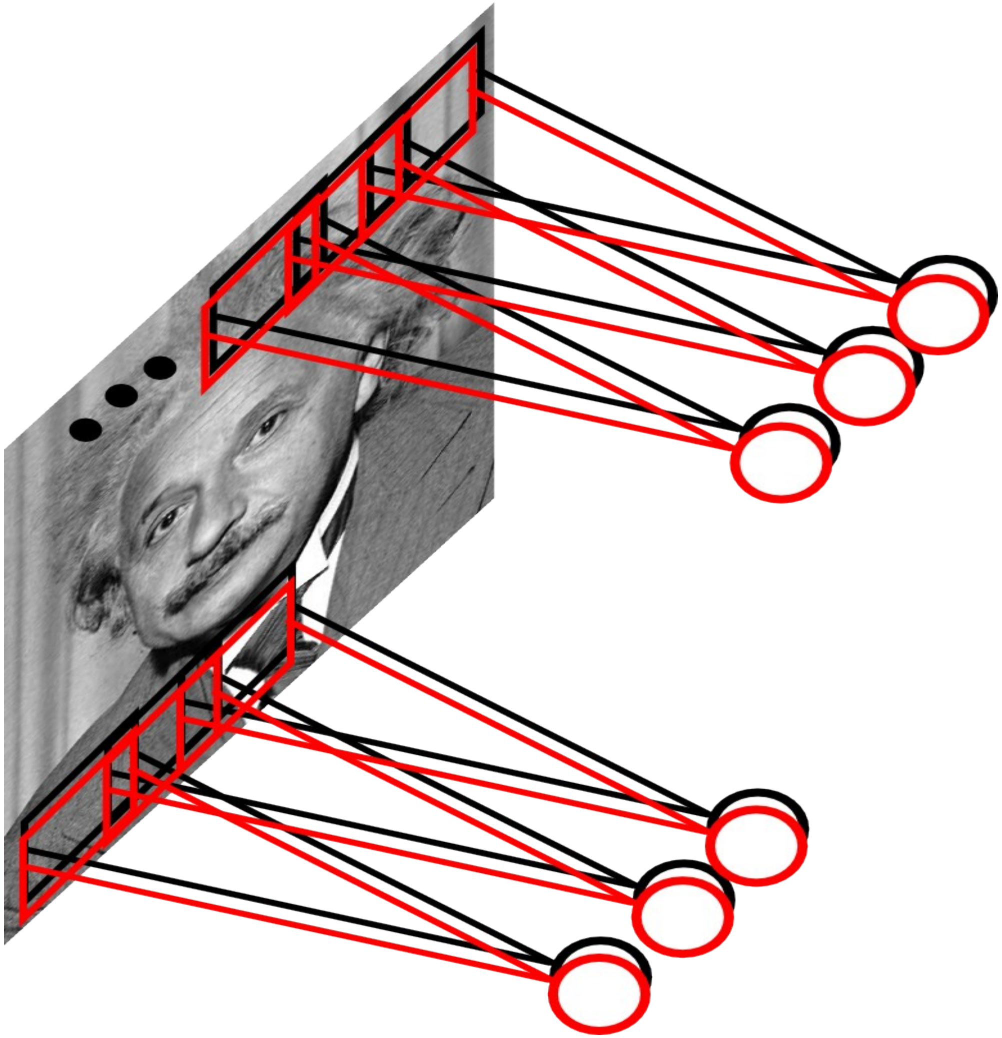
- Example: 200x200 image
  - 400,000 hidden units with 10x10 fields = 1000 params
  - 10 feature maps of size 200x200, 10 filters of size 10x10



Multiple Convolutions with Different Kernels

- 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.

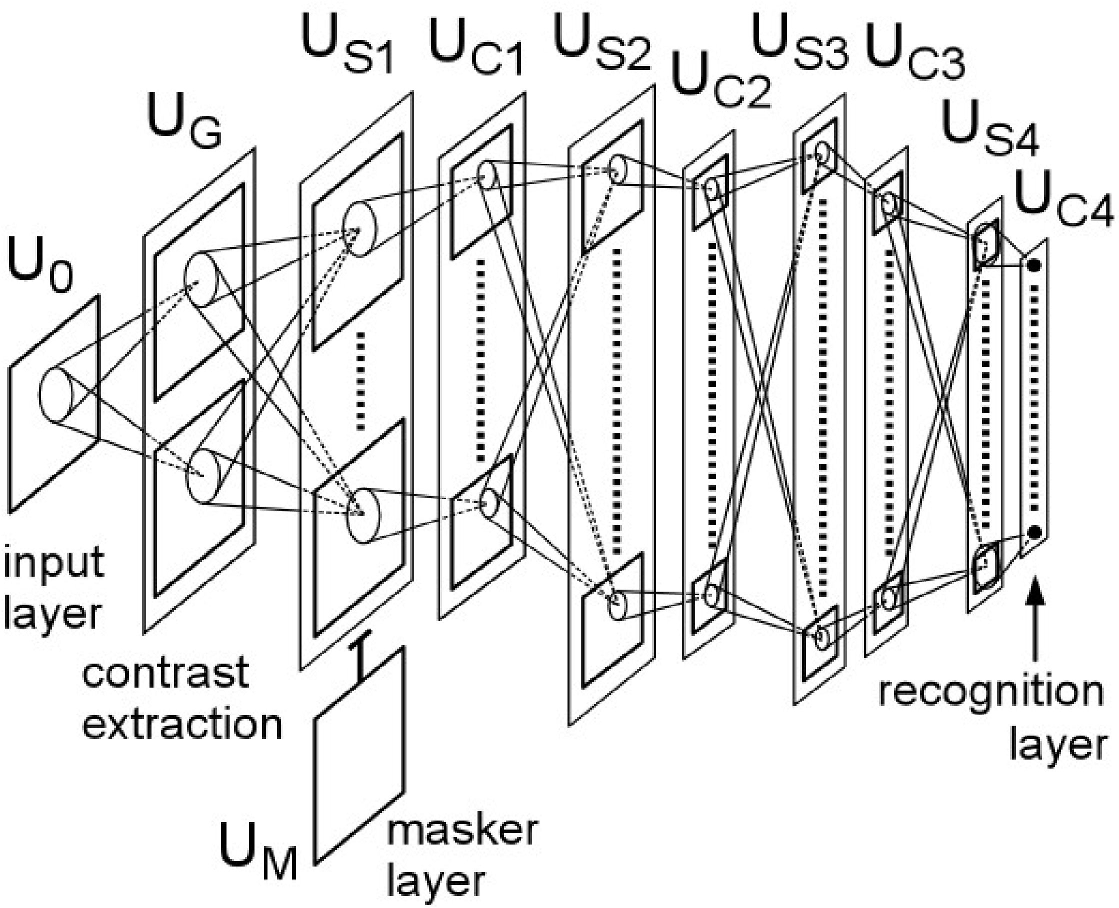


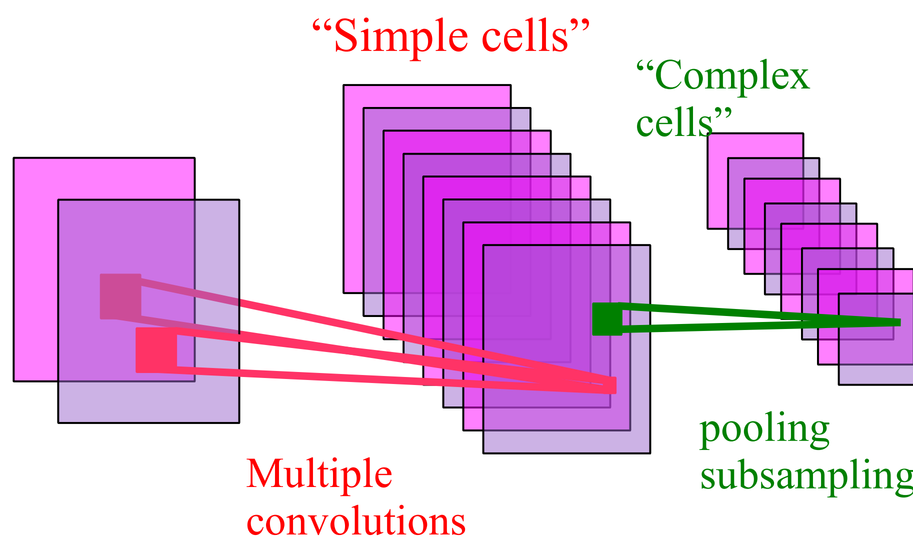


Early Hierarchical Feature Models for Vision

Y LeCun

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

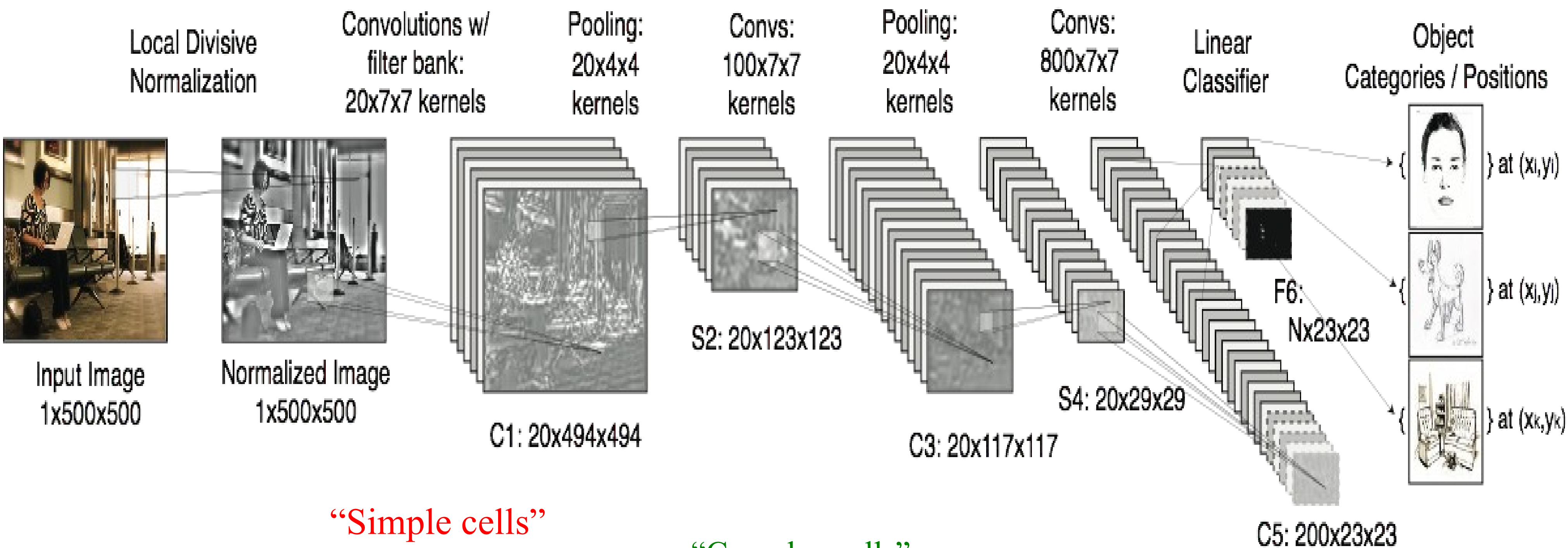


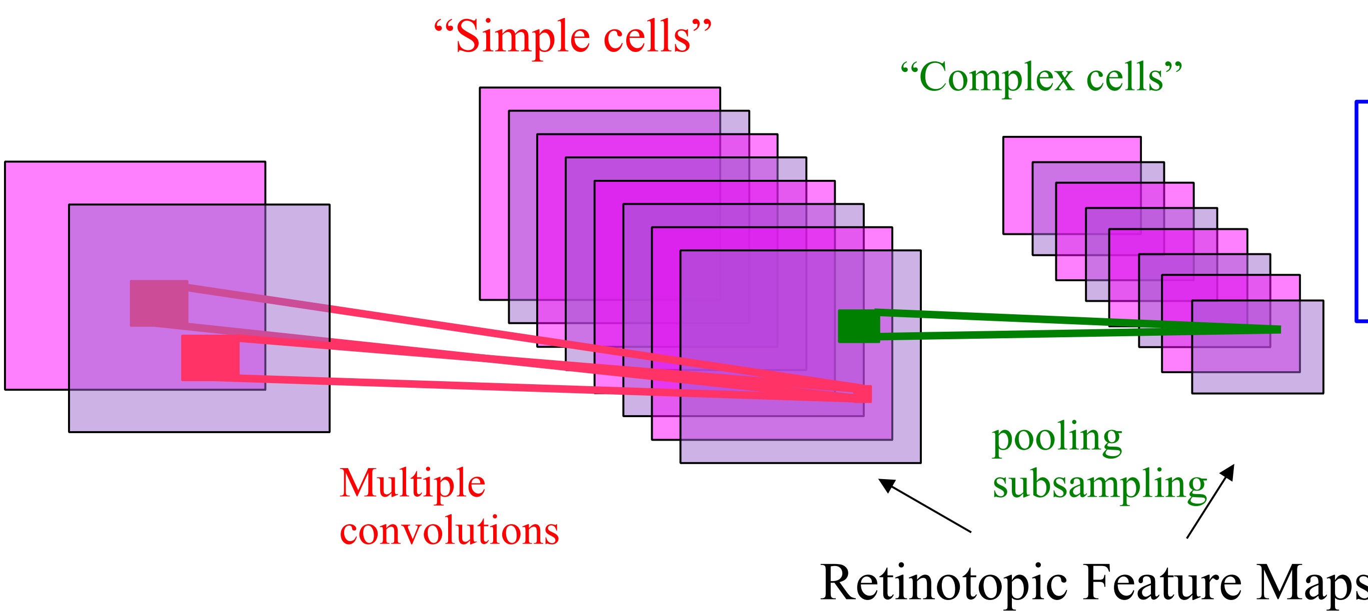


Cognitron & Neocognitron [Fukushima 1974-1982]

The Convolutional Net Model
(Multistage Hubel-Wiesel system)

Y LeCun



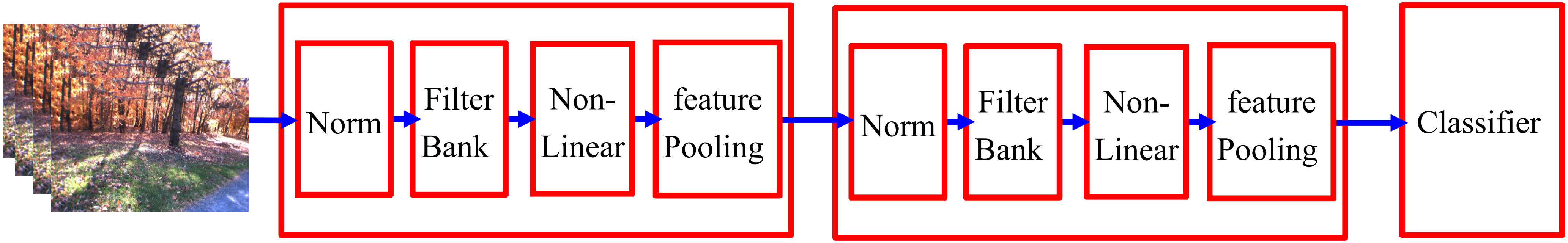


- Training is supervised
- With stochastic gradient descent

[LeCun et al. 89]

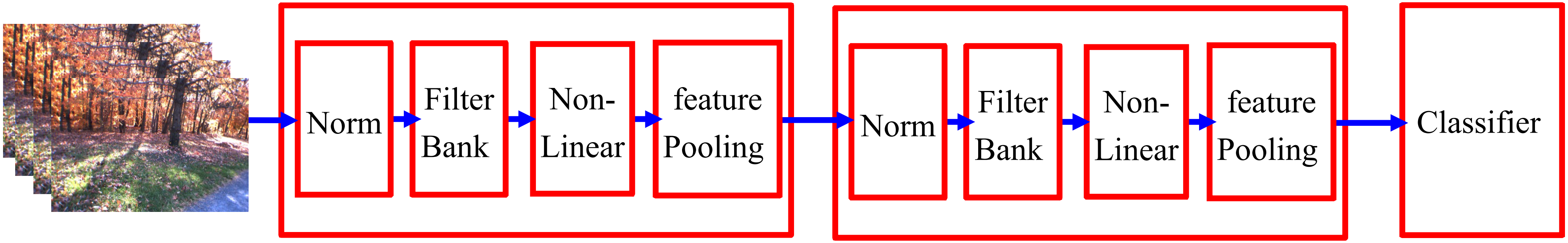
[LeCun et al. 98]

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



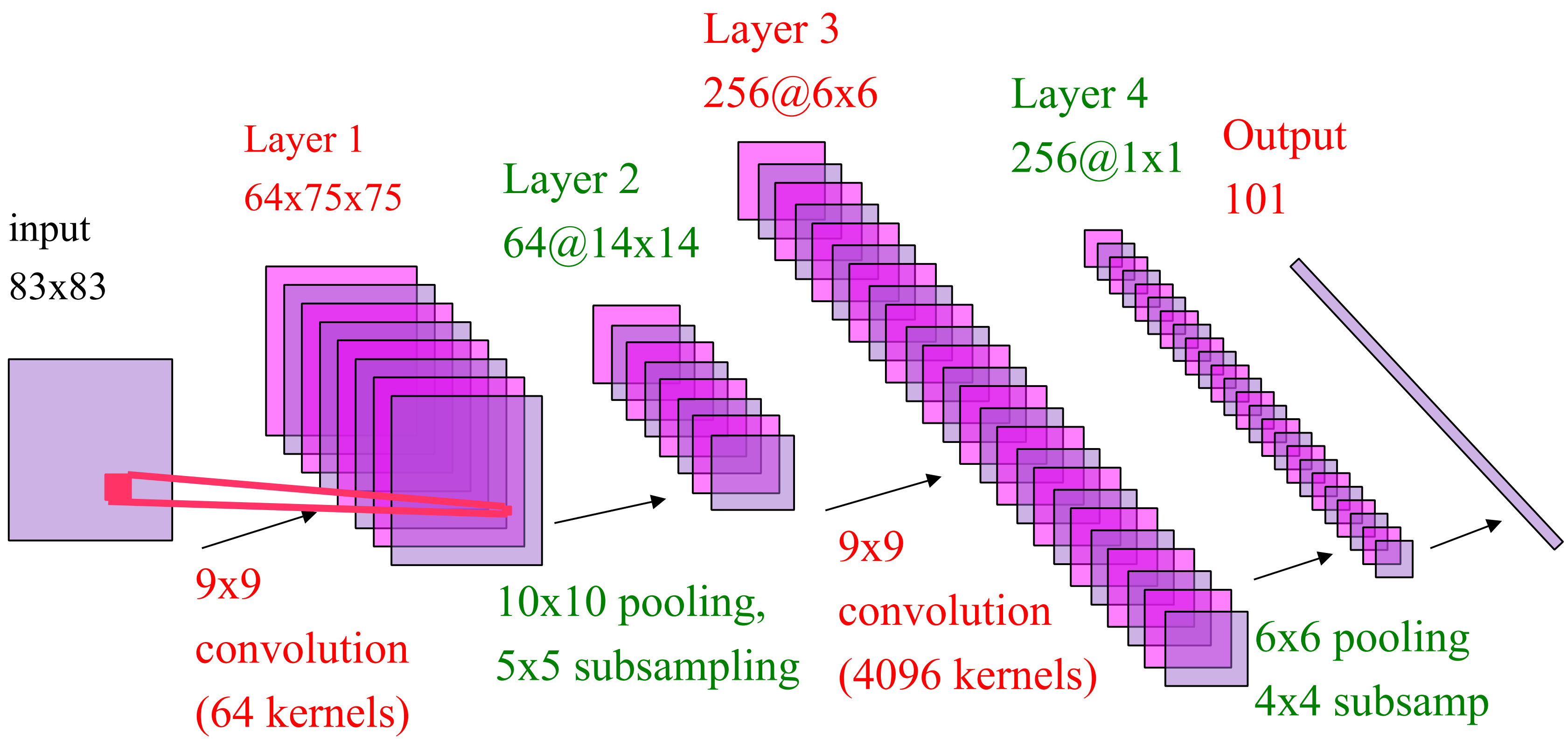
- **Stacking multiple stages of**
  - [Normalization → Filter Bank → Non-Linearity → Pooling].
- **Normalization: variations on whitening**
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank: dimension expansion, projection on overcomplete basis**
- **Non-Linearity: sparsification, saturation, lateral inhibition....**
  - Rectification, 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



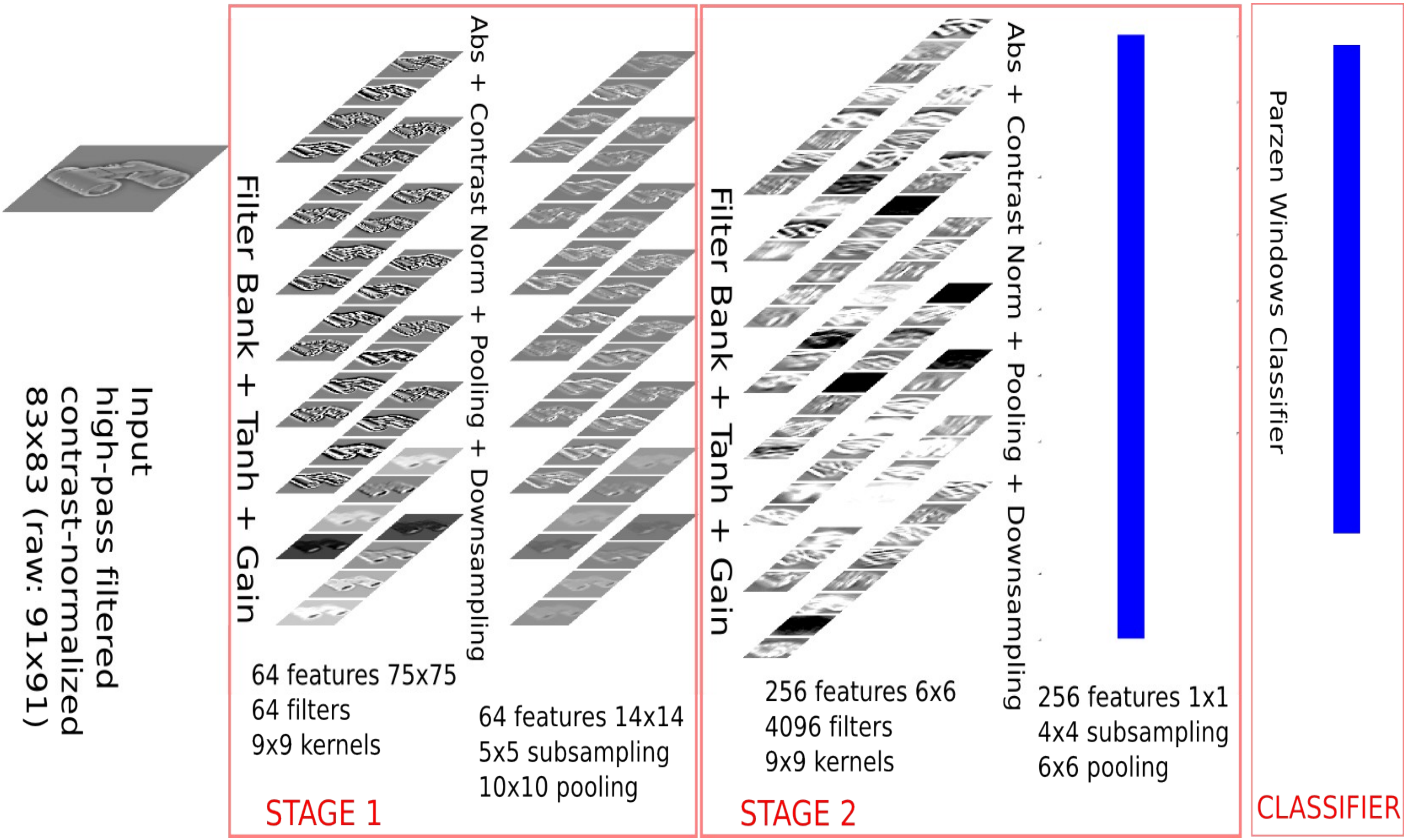
- Filter Bank → Non-Linearity = Non-linear embedding in high dimension
- Feature Pooling = contraction, dimensionality reduction, smoothing
- Learning the filter banks at every stage
- Creating a hierarchy of features
- Basic elements are inspired by models of the visual (and auditory) cortex
  - Simple 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)



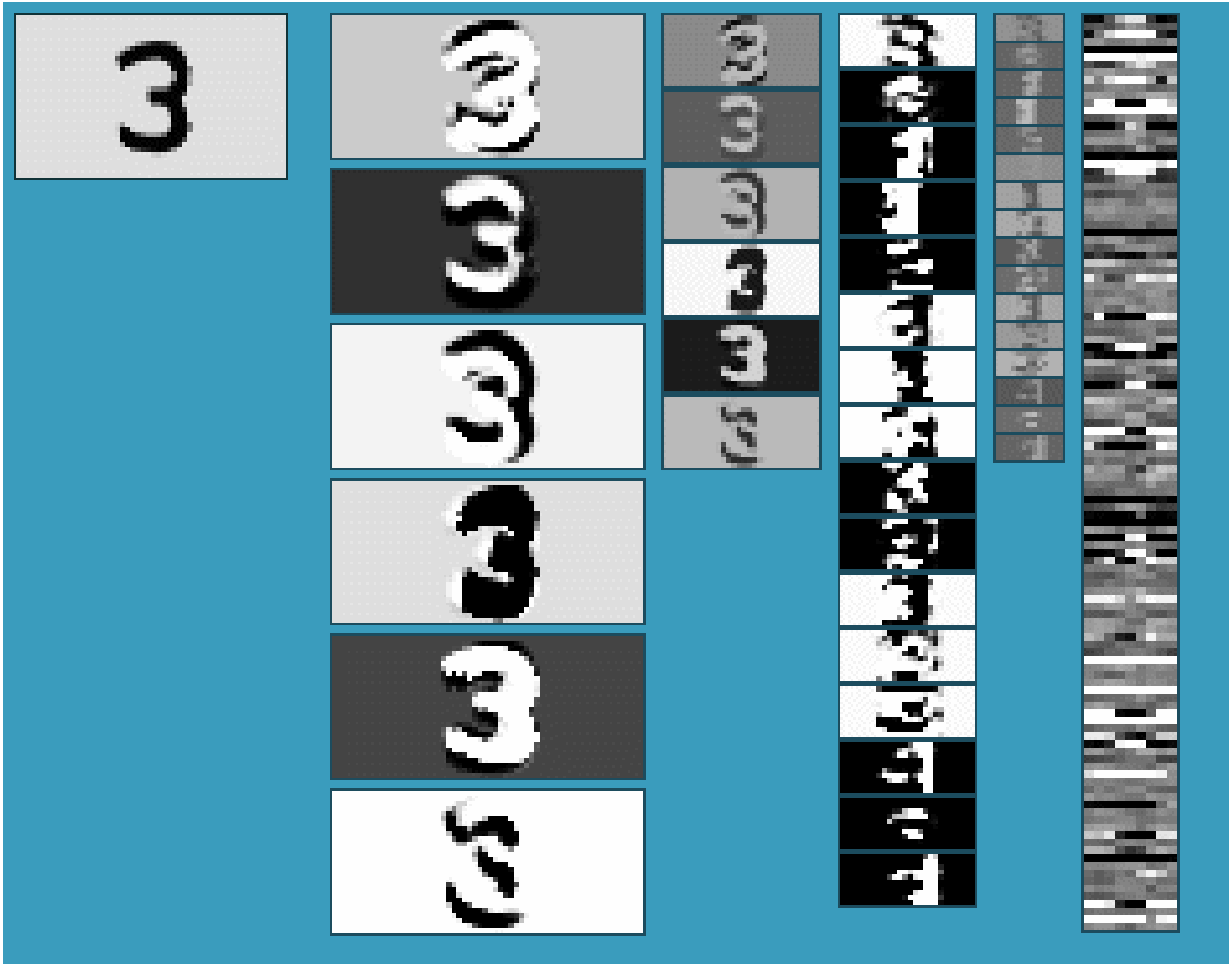
- Non-Linearity: half-wave rectification, shrinkage function, sigmoid
- Pooling: average, L1, L2, max
- Training: Supervised (1988-2006), Unsupervised+Supervised (2006-now)

Convolutional Network Architecture



Convolutional Network (vintage 1990)

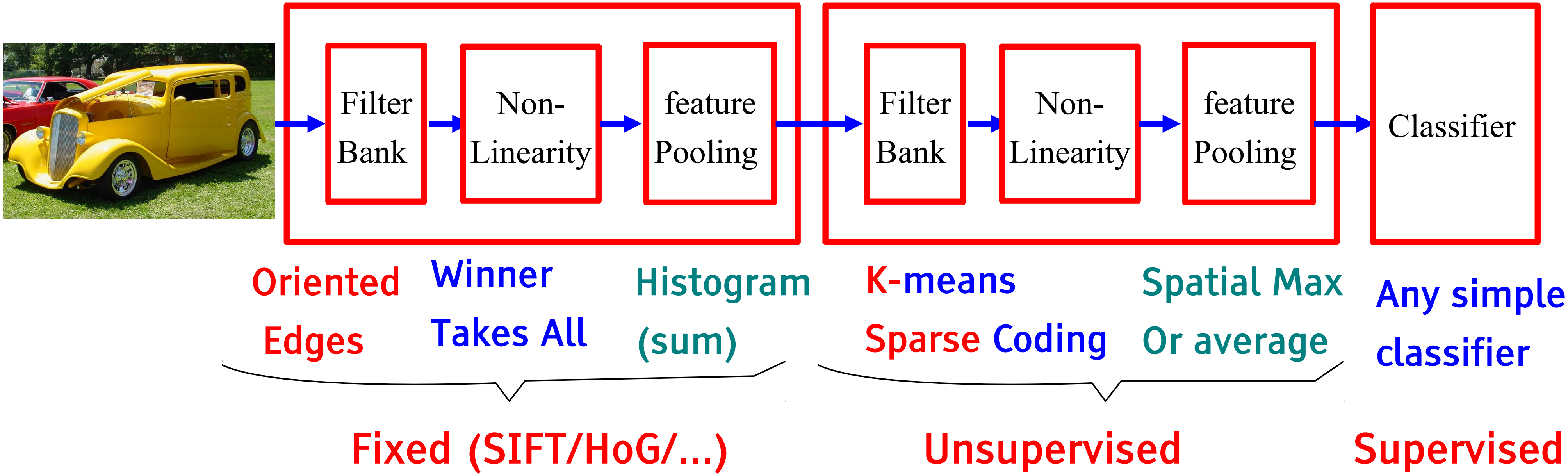
- filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh



Curved manifold

Flatter manifold

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



- 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

- 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

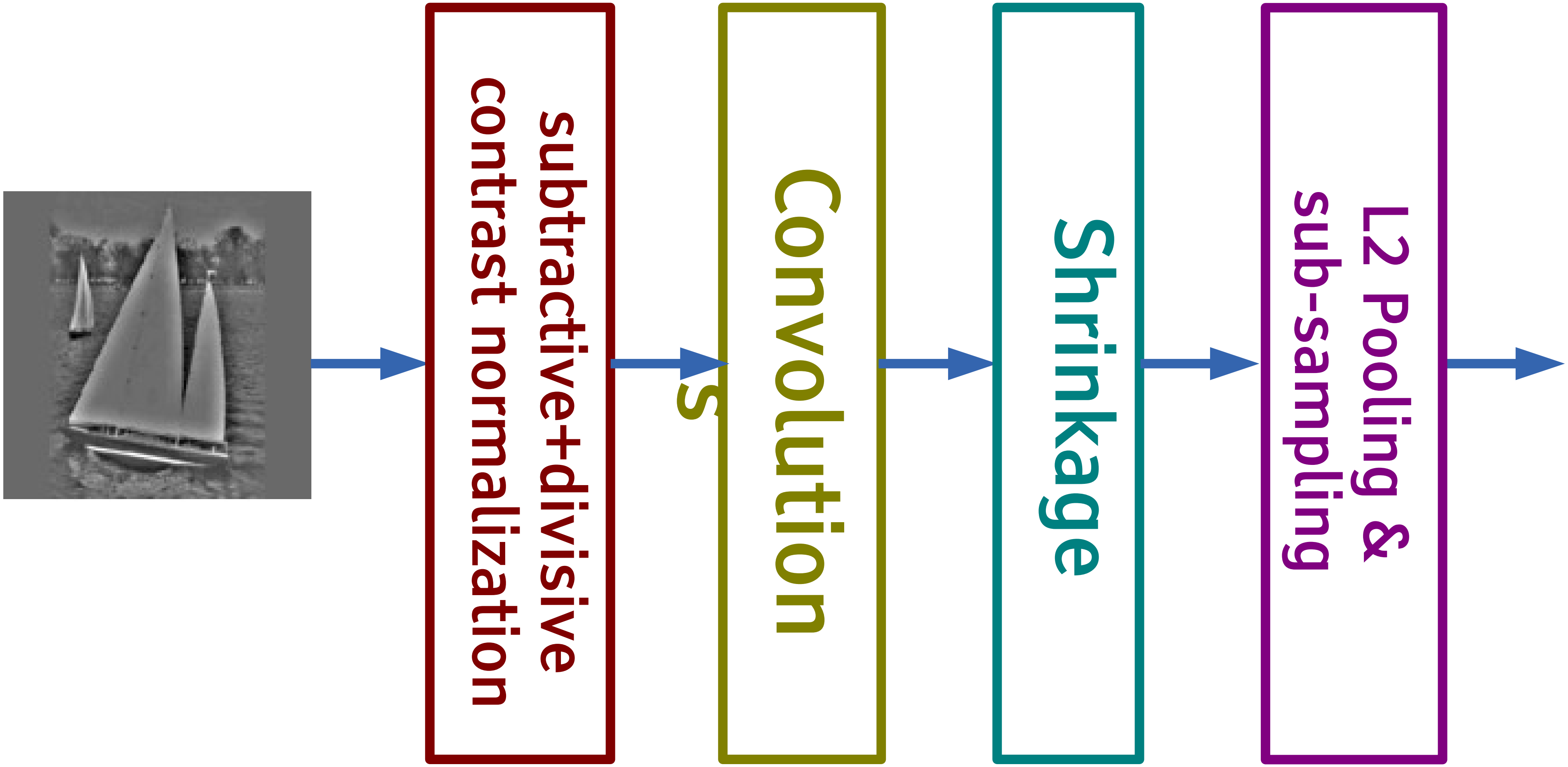
- The whole architecture: simple cells and complex cells
- Local receptive fields
- Self-similar receptive fields over the visual field (convolutions)
- Pooling (complex cells)
- Non-Linearity: Rectified Linear Units (ReLU)
- LGN-like band-pass filtering and contrast normalization in the input
- Divisive 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

One Stage: Contrast Norm → Filter Bank → Shrinkage → L2 Pooling



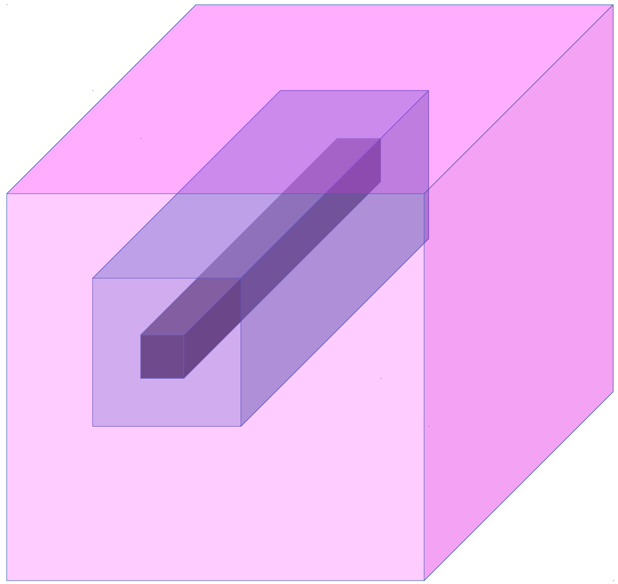
THIS IS ONE STAGE OF THE CONVNET

Results on Caltech101 with sigmoid non-linearity

Single Stage System: $[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ **- log_reg**					
R/N/P	$R_{abs} - N - P_A$	$R_{abs} - P_A$	$N - P_M$	$N - P_A$	P_A
U^+	54.2%	50.0%	44.3%	18.5%	14.5%
R^+	54.8%	47.0%	38.0%	16.3%	14.3%
U	52.2%	43.3%(±1.6)	44.0%	17.2%	13.4%
R	53.3%	31.7%	32.1%	15.3%	12.1%(±2.2)
G	52.3%				
Two Stage System: $[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}] - [256.F^{9\times9}_{CSG} - R/N/P^{4\times4}]$ **- log_reg**					
R/N/P	$R_{abs} - N - P_A$	$R_{abs} - P_A$	$N - P_M$	$N - P_A$	P_A
U^+U^+	65.5%	60.5%	61.0%	34.0%	32.0%
R^+R^+	64.7%	59.5%	60.0%	31.0%	29.7%
UU	63.7%	46.7%	56.0%	23.1%	9.1%
RR	62.9%	33.7%(±1.5)	37.6%(±1.9)	19.6%	8.8%
GT	55.8%	← like HMAX model			
Single Stage: $[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ **- PMK-SVM**					
U	64.0%				
Two Stages: $[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}] - [256.F^{9\times9}_{CSG} - R/N]$ **- PMK-SVM**					
UU	52.8%				

Local Contrast Normalization

- Performed on the state of every layer, including the input
- Subtractive Local Contrast Normalization
  - Subtracts from every value in a feature a Gaussian-weighted average of its neighbors (high-pass filter)
- Divisive Local Contrast Normalization
  - Divides every value in a layer by the standard deviation of its neighbors over space and over all feature maps
- Subtractive + Divisive LCN performs a kind of approximate whitening.



The Effect of Architectural Elements

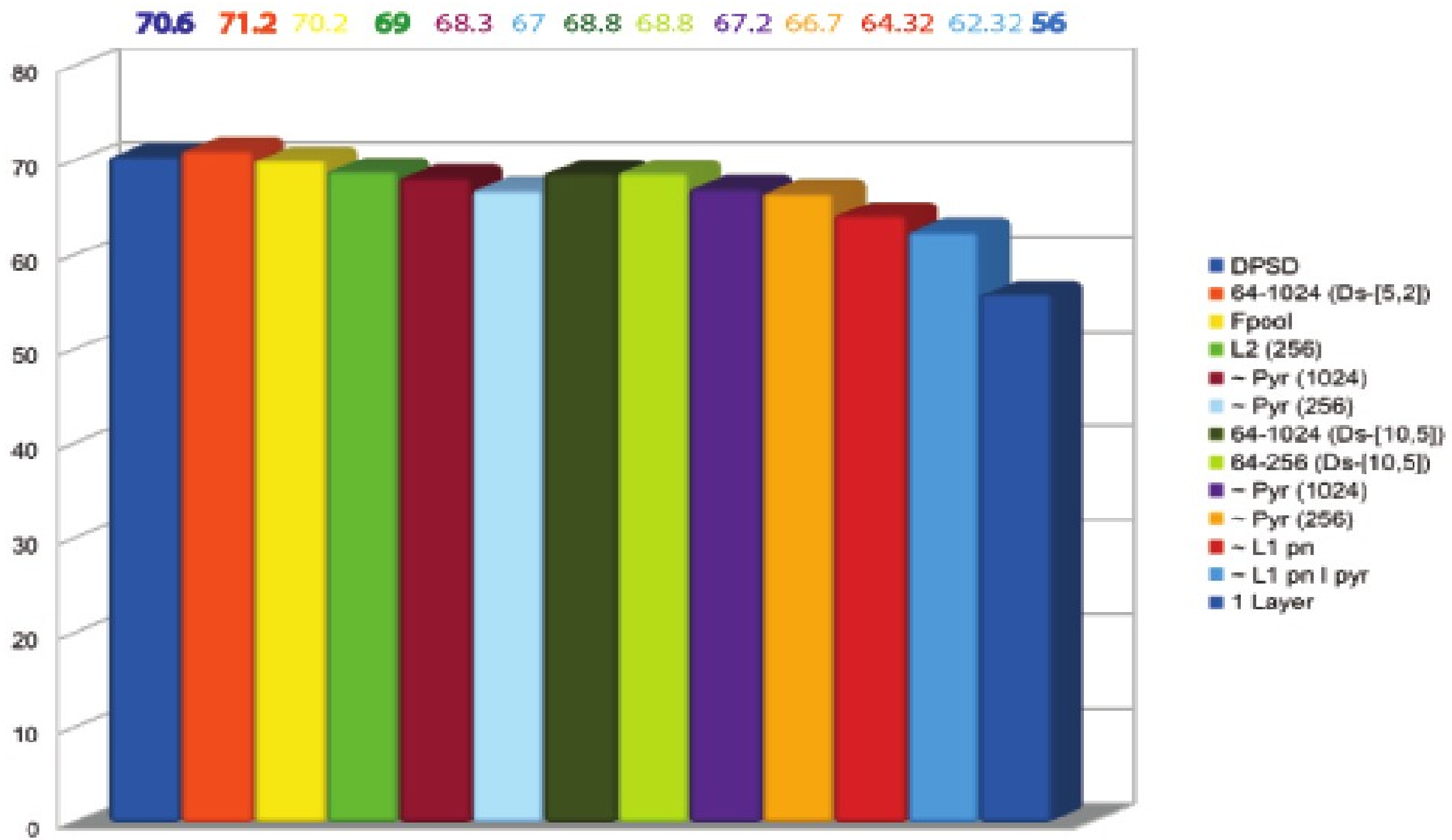
- Pyramid pooling on last layer: 1% improvement over regular pooling
- Shrinkage non-linearity + lateral inhibition: 1.6% improvement over tanh
- Discriminative term in sparse coding: 2.8% improvement

Architecture	Protocol	%
(1) $F_{\tanh} - R_{abs} - N - P_A^{pyr}$	$\mathbf{R^+R^+}$	65.4 ± 1.0
(2) $F_{\tanh} - R_{abs} - N - P_A^{pyr}$	$\mathbf{U^+U^+}$	66.2 ± 1.0
(3) $F_{si} - R_{abs} - N - P_A$	$\mathbf{R^+R^+}$	63.3 ± 1.0
(4) $F_{si} - R_{abs} - N - P_A$	$\mathbf{UU}$	60.4 ± 0.6
(5) $F_{si} - R_{abs} - N - P_A$	$\mathbf{U^+U^+}$	66.4 ± 0.5
(6) $F_{si} - R_{abs} - N - P_A^{pyr}$	$\mathbf{U^+U^+}$	67.8 ± 0.4
(7) $F_{si} - R_{abs} - N - P_A$	$\mathbf{DD}$	66.0 ± 0.3
(8) $F_{si} - R_{abs} - N - P_A$	$\mathbf{D^+D^+}$	68.7 ± 0.2
(9) $F_{si} - R_{abs} - N - P_A^{pyr}$	$\mathbf{D^+D^+}$	$\mathbf{70.6 \pm 0.3}$

Y LeCun

Results on Caltech101: purely supervised with soft-shrink, L2 pooling, contrast normalization

- Supervised learning with soft-shrinkage non-linearity, L2 complex cells, and sparsity penalty on the complex cell outputs: 71%
- Caltech101 is pathological, biased, too small, etc...

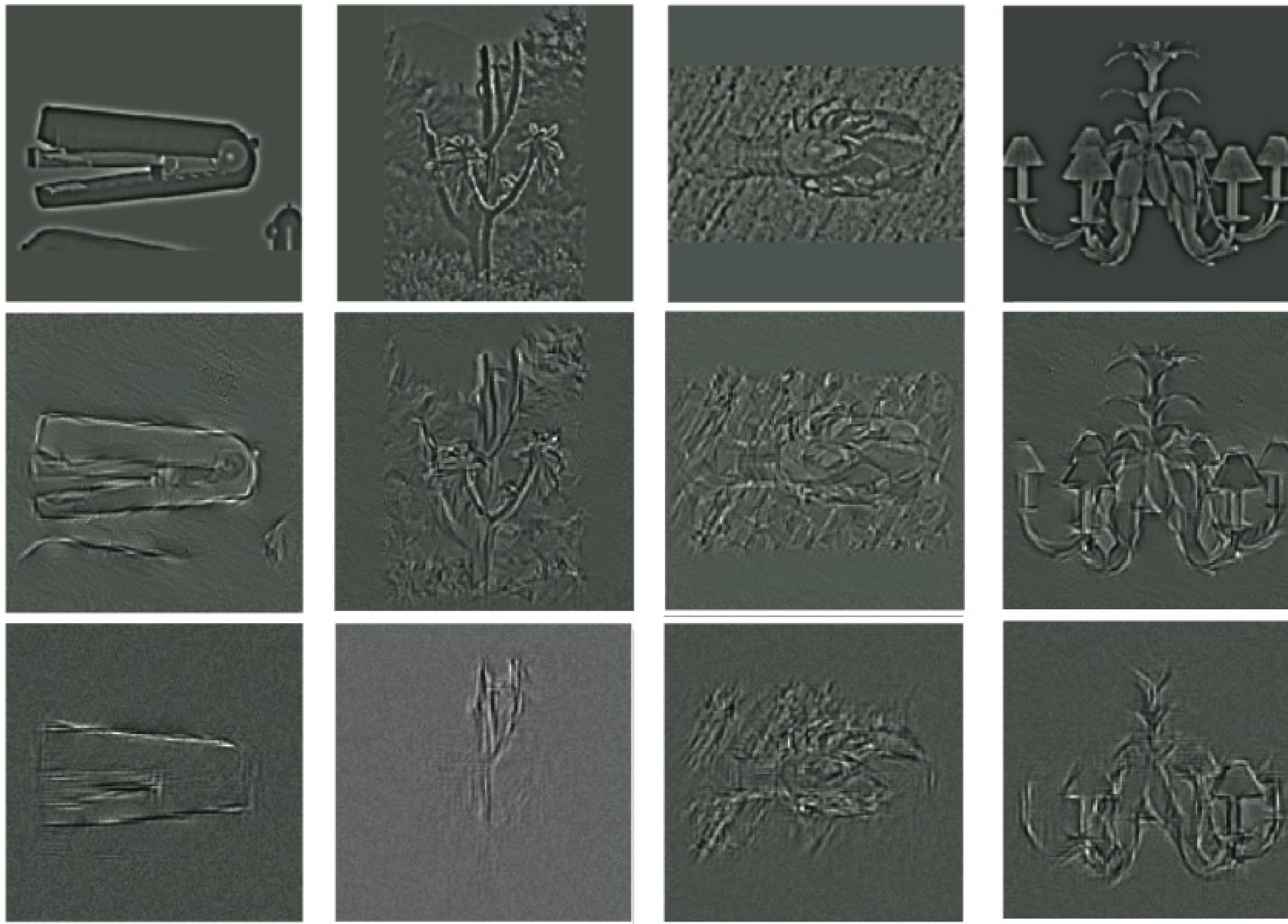


What does Local Contrast Normalization Do?

Original

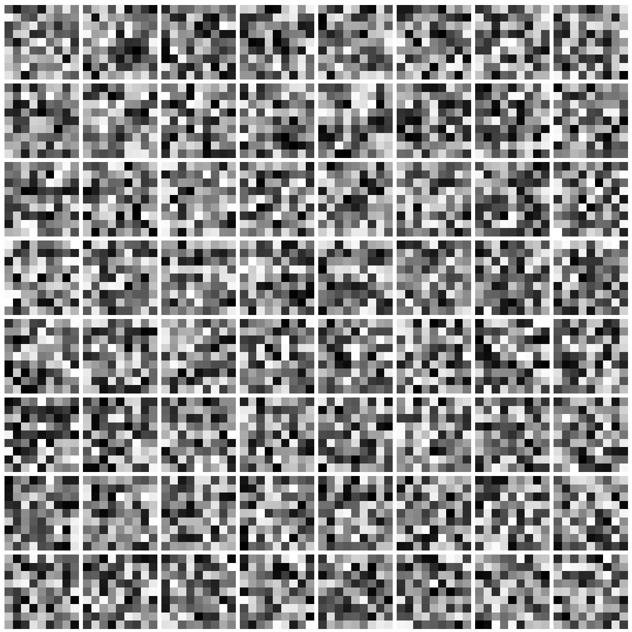
Reconstuction
With LCN

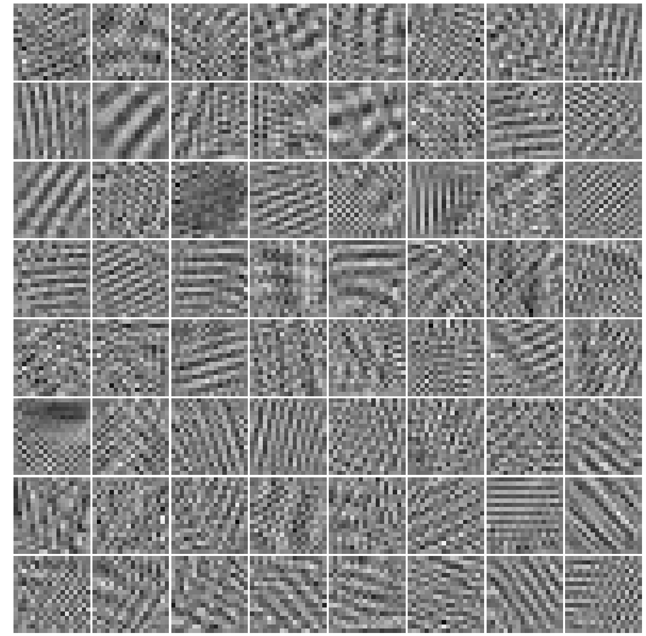
Reconstruction
Without LCN



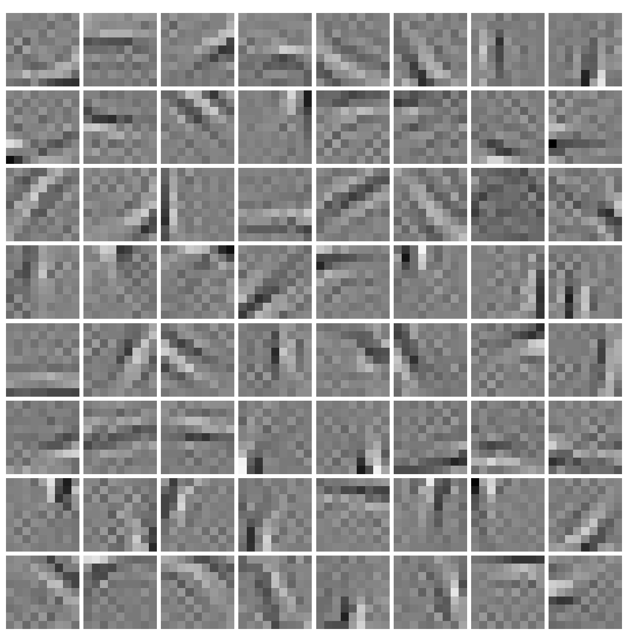
Why Do Random Filters Work?

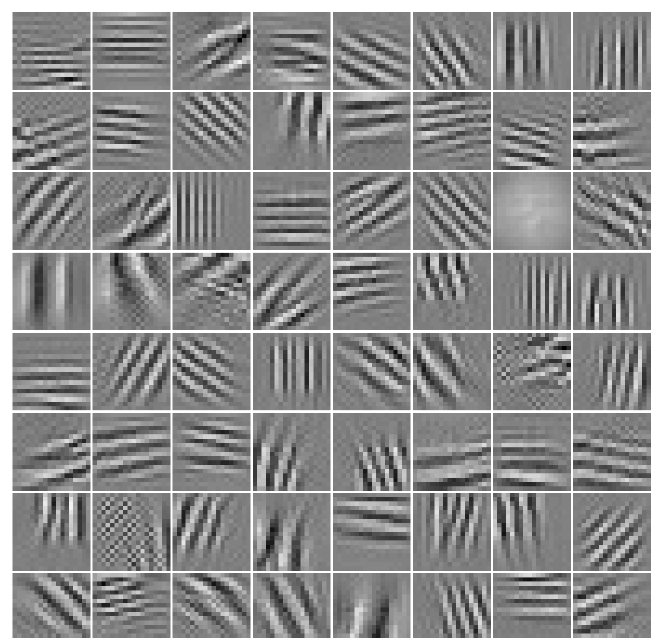
Random Filters For Simple Cells





Trained Filters For Simple Cells

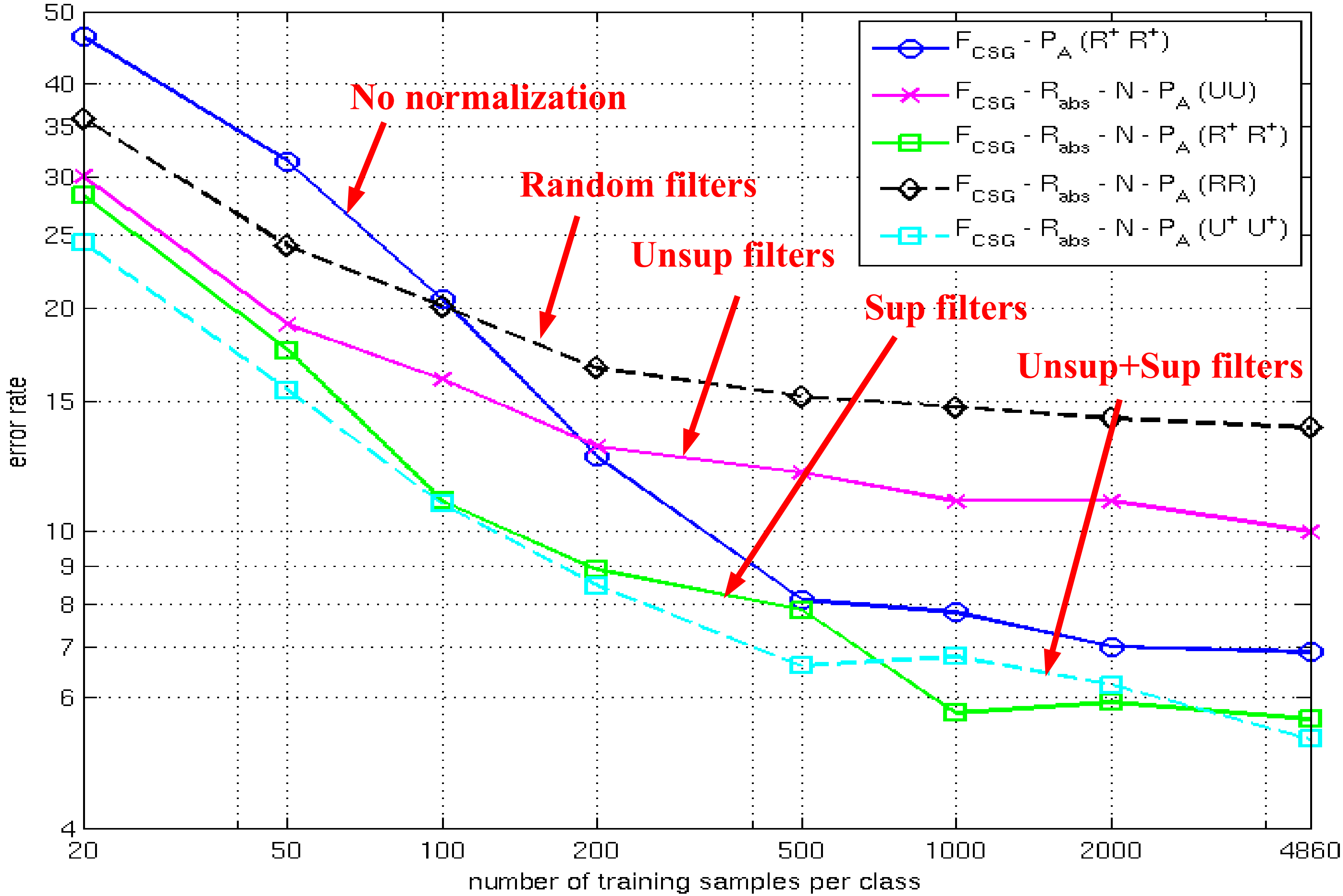




Optimal Stimuli for each Complex Cell

Small NORB dataset

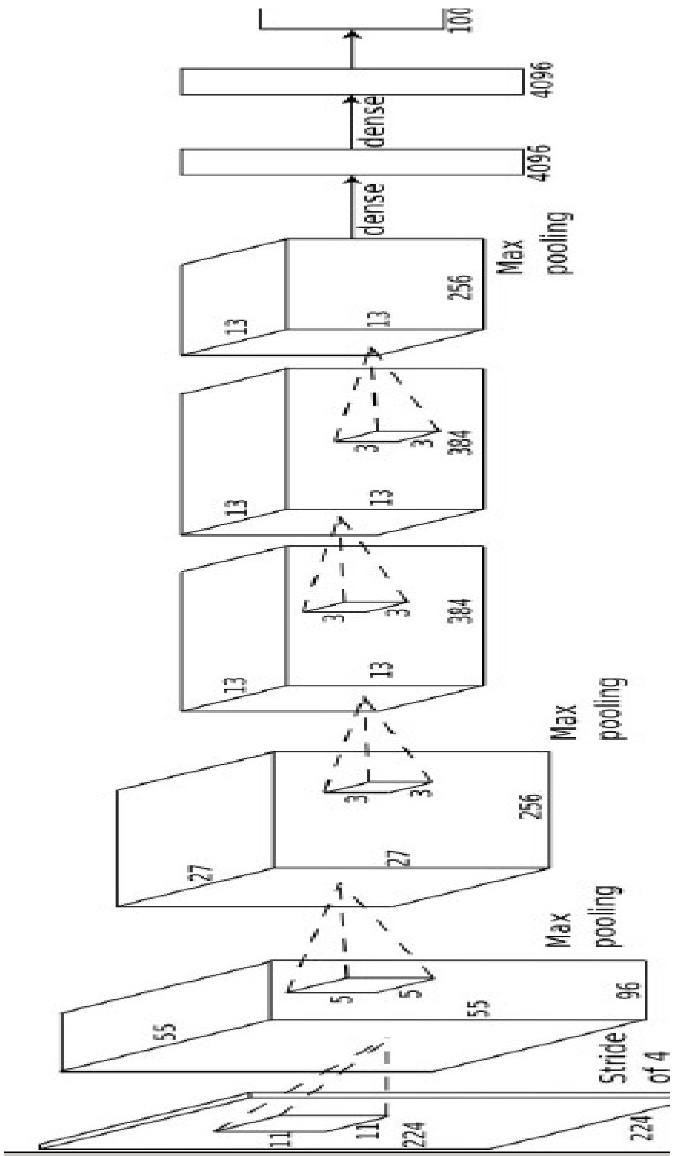
- Two-stage system: error rate versus number of labeled training samples



Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

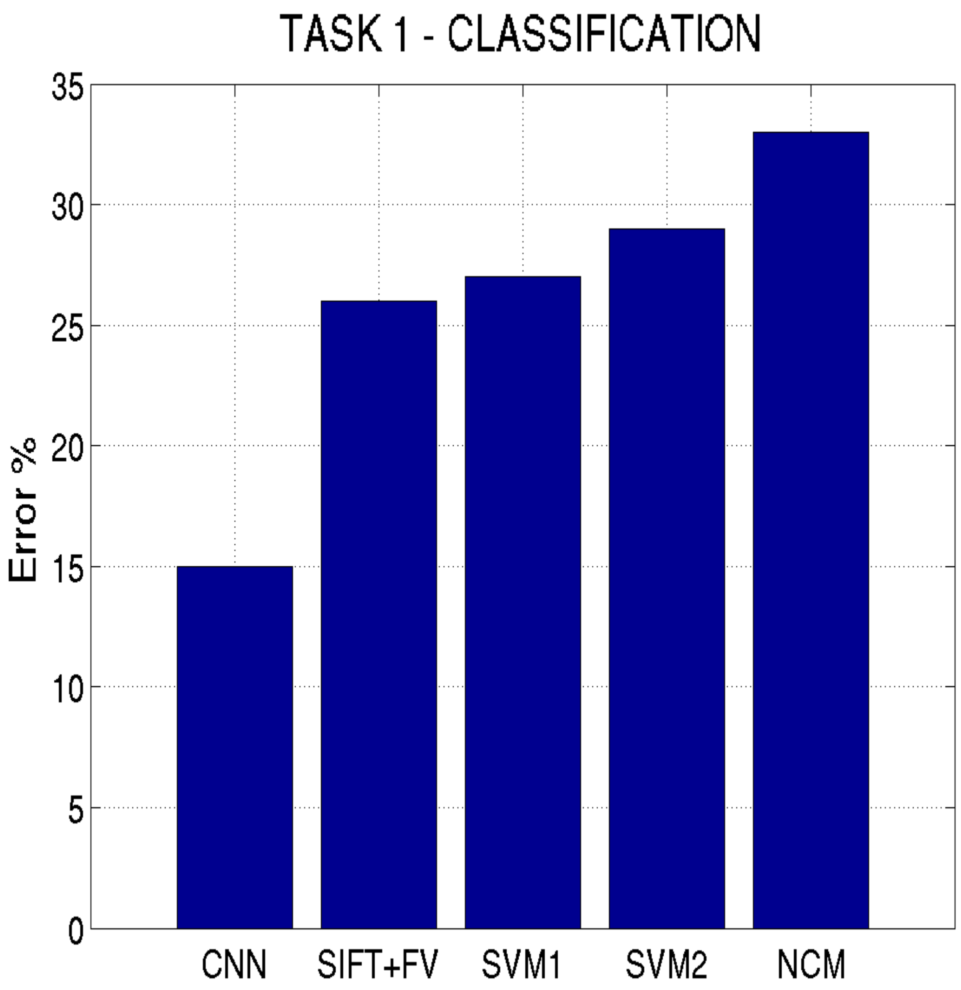
- Won the 2012 ImageNet LSVRC. 60 Million parameters, 832M MAC ops

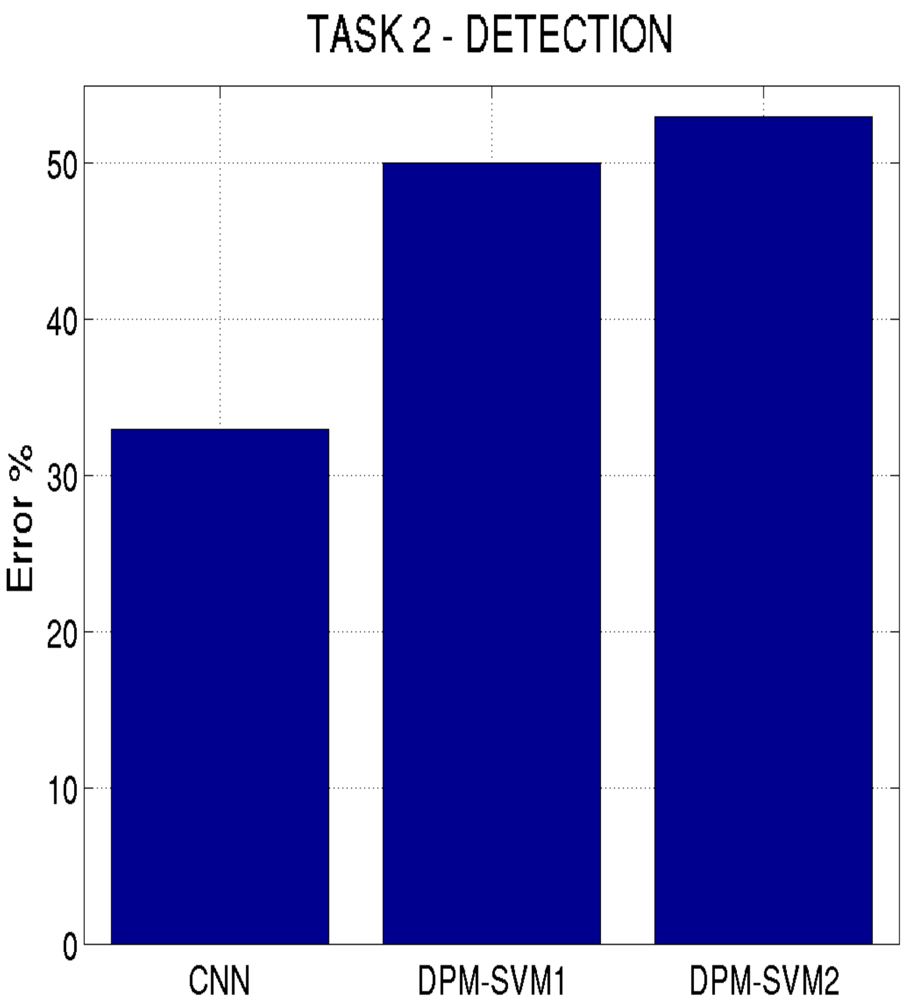
4M	FULL CONNECT	4Mflop
16M	FULL 4096/ReLU	16M
37M	FULL 4096/ReLU	37M
	MAX POOLING	
442K	CONV 3x3/ReLU 256fm	74M
1.3M	CONV 3x3ReLU 384fm	224M
884K	CONV 3x3/ReLU 384fm	149M
	MAX POOLING 2x2sub	
	LOCAL CONTRAST NORM	
307K	CONV 11x11/ReLU 256fm	223M
	MAX POOL 2x2sub	
	LOCAL CONTRAST NORM	
35K	CONV 11x11/ReLU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



Object Recognition: ILSVRC 2012 results

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





Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

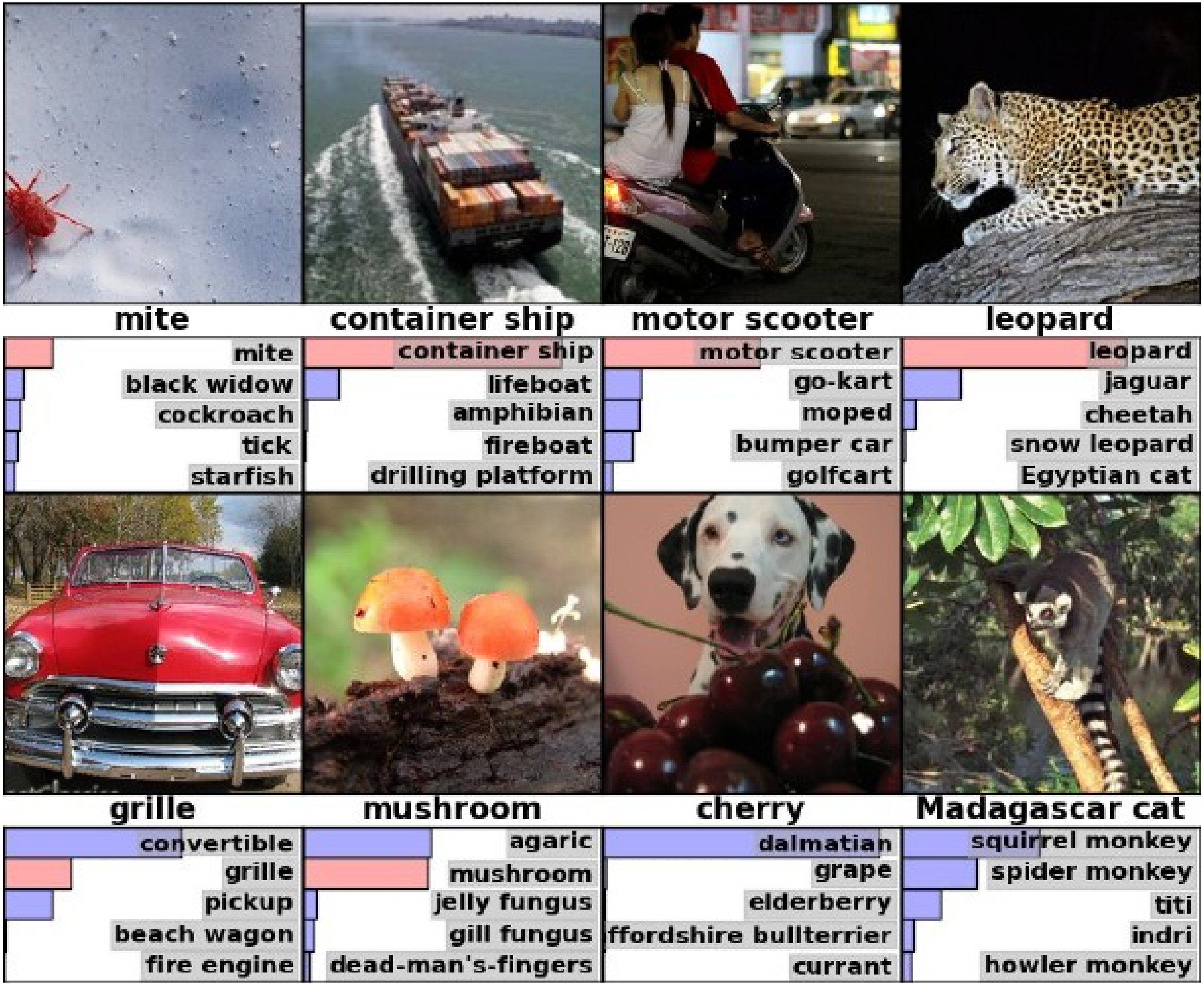
- Method: large convolutional net
  - 650K neurons, 832M synapses, 60M parameters
  - Trained with backprop on GPU
  - Trained "with all the tricks Yann came up with in the last 20 years, plus dropout" (Hinton, NIPS 2012)
  - Rectification, contrast normalization,...
- Error rate: 15% (whenever correct 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

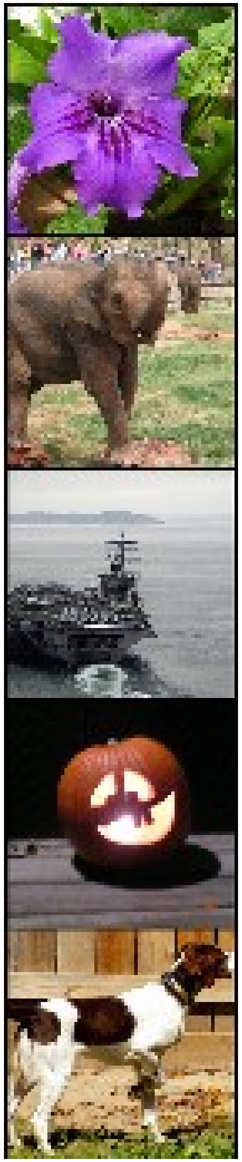
Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

Y LeCun

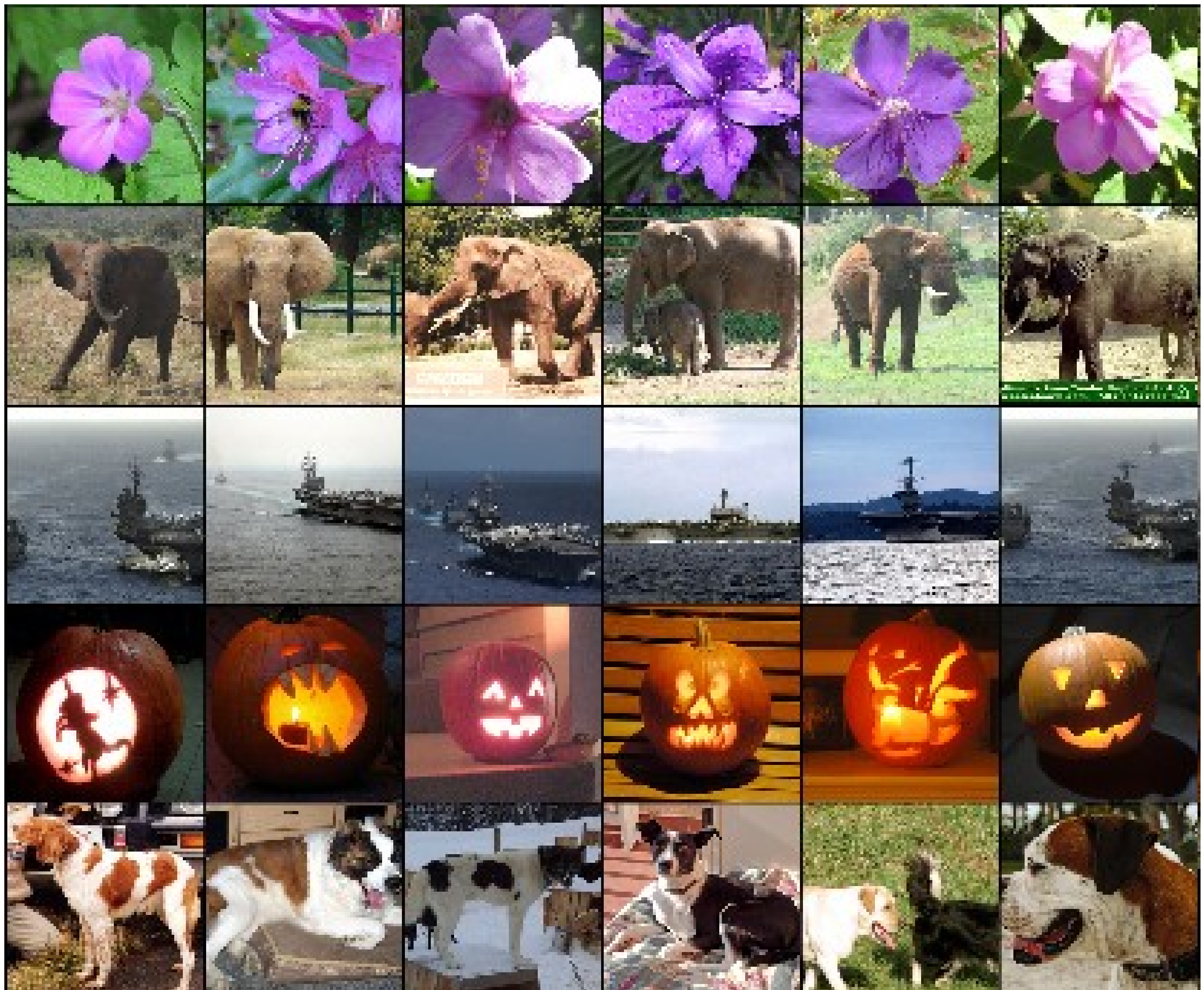


Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

TEST IMAGE

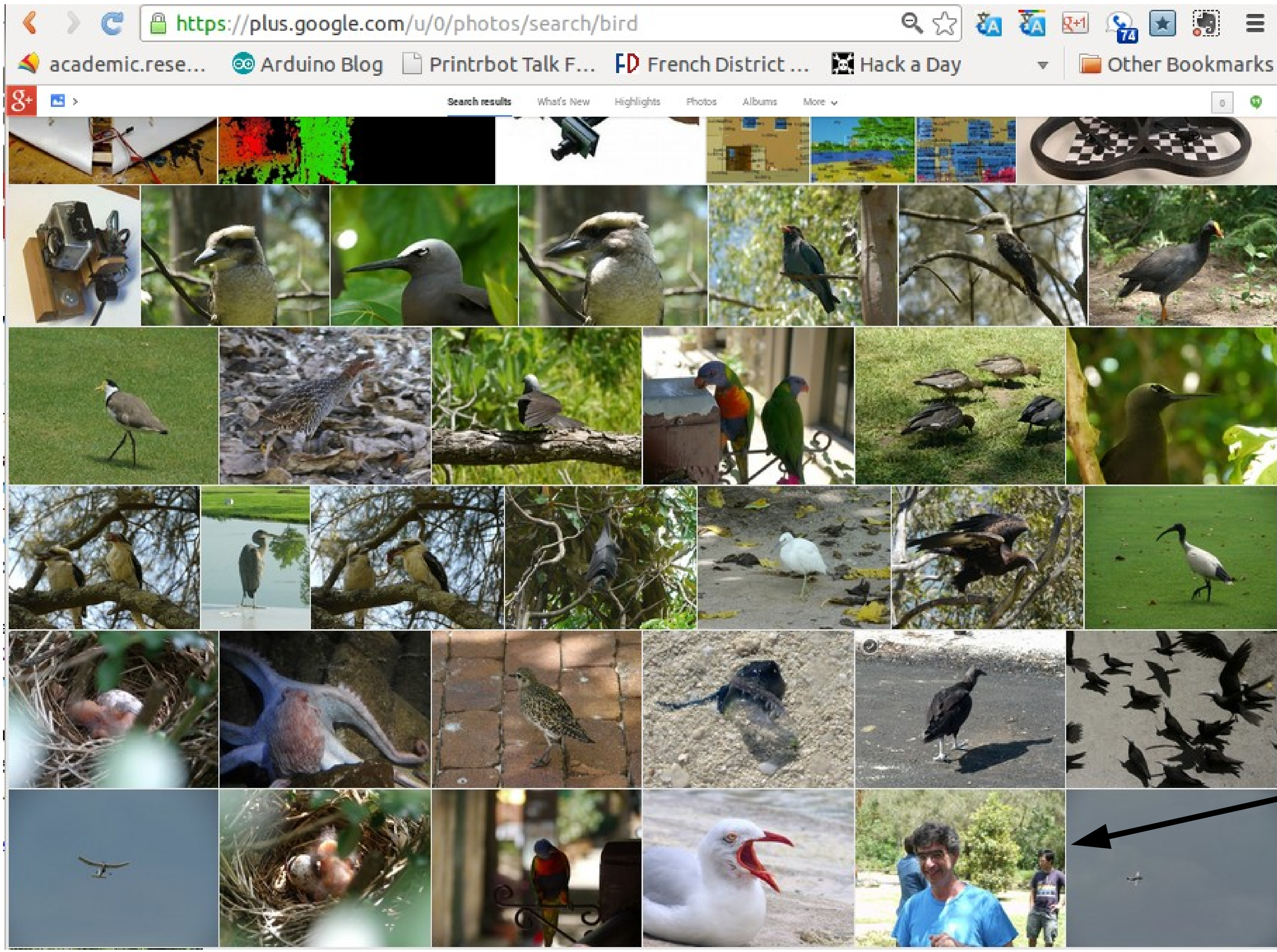


RETRIEVED IMAGES



ConvNet-Based Google+ Photo Tagger

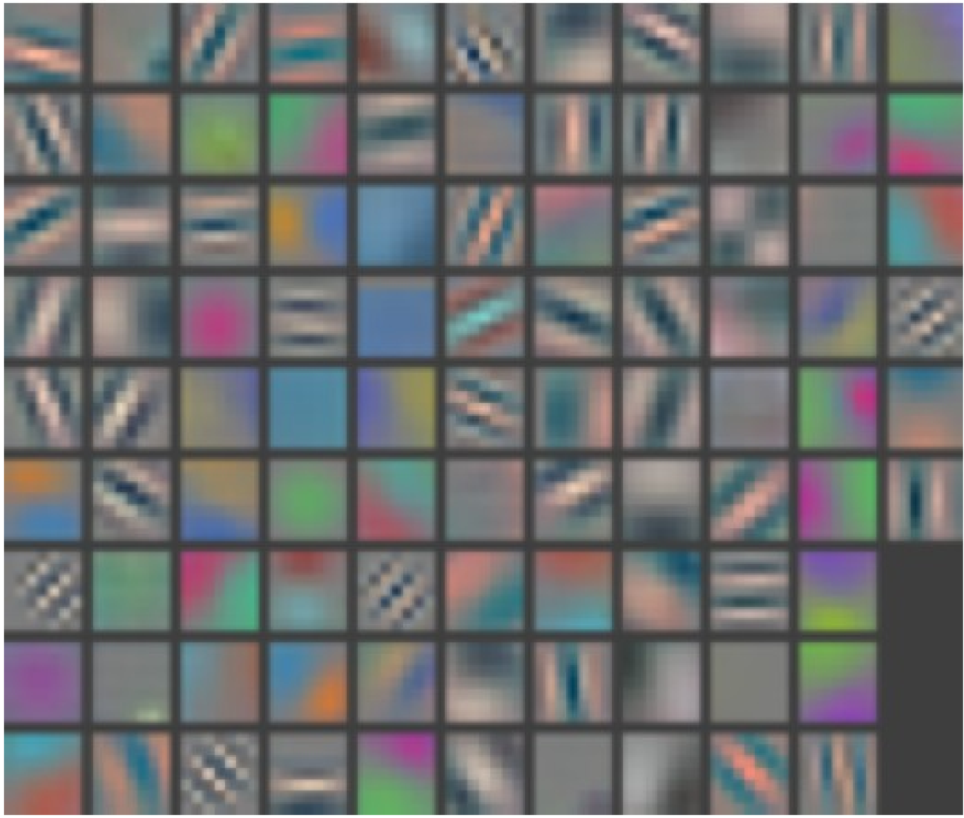
Searched my personal collection for "bird"



Samy
Bengio
???

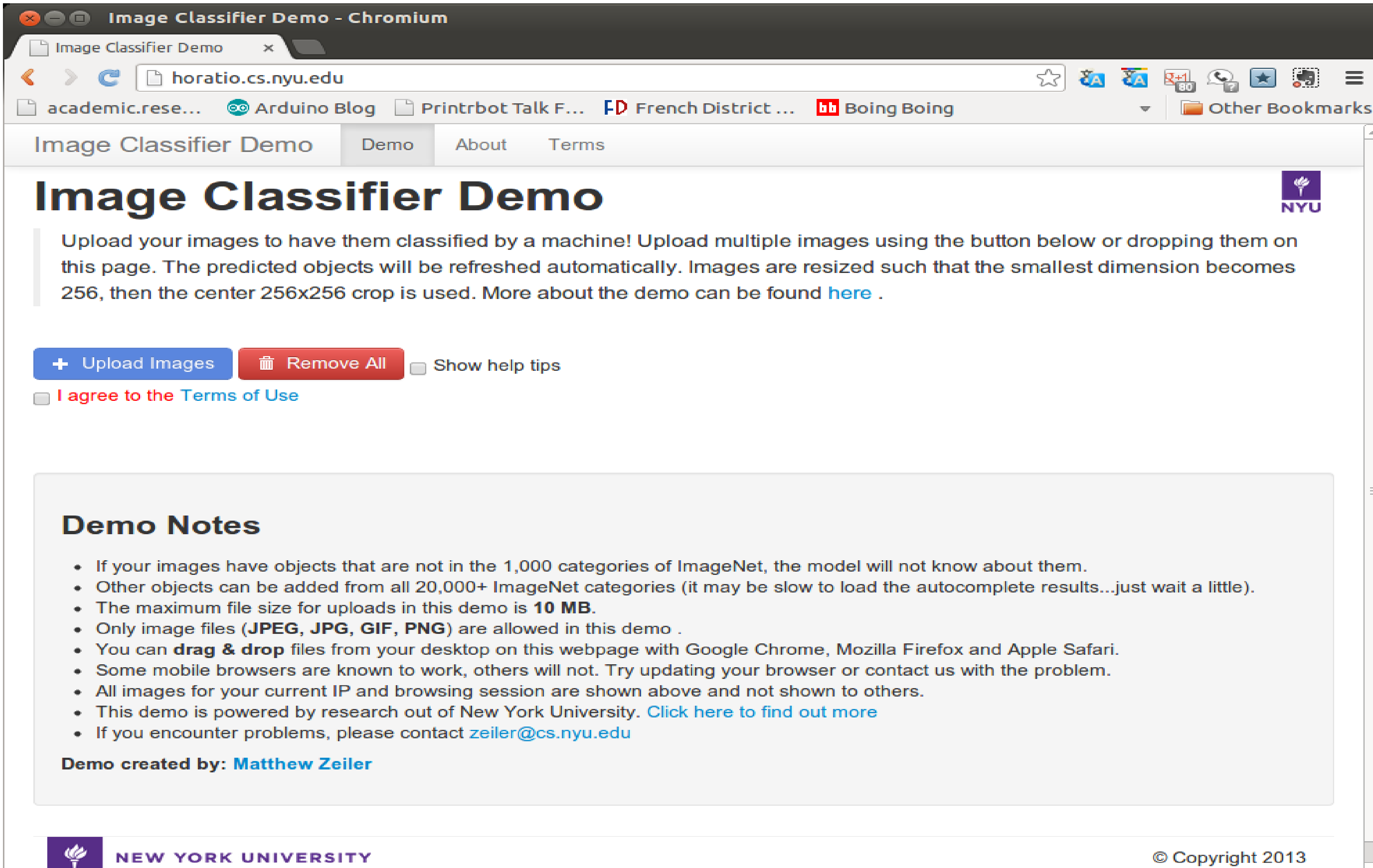
Another ImageNet-trained ConvNet [Zeiler & Fergus 2013]

- **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 image
- **Regularization: Dropout**
  - [Hinton 2012]
  - zeroing random subsets of units
- **Stochastic gradient descent**
  - for 70 epochs (7-10 days)
  - With learning rate annealing



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

http://horatio.cs.nyu.edu

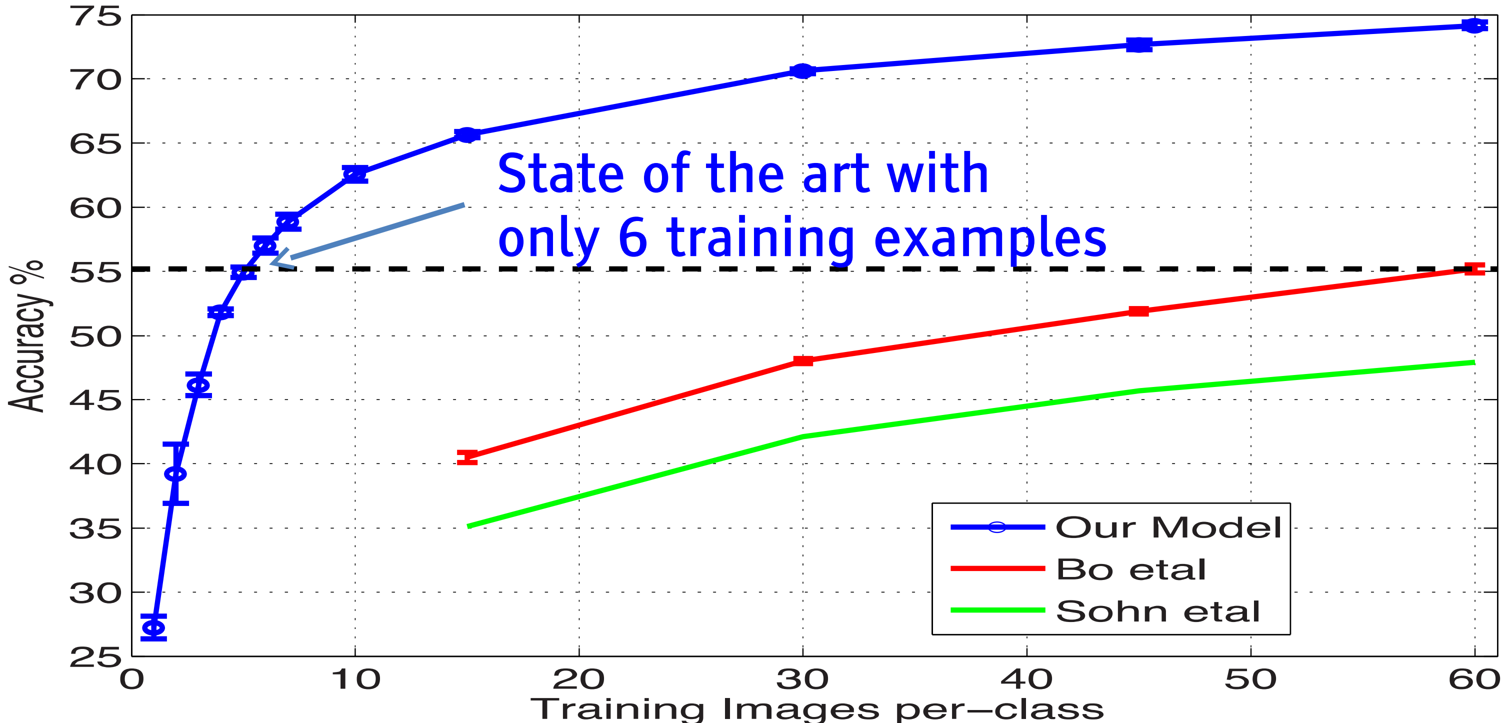


ConvNet trained on ImageNet [Zeiler & Fergus 2013]

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

- 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

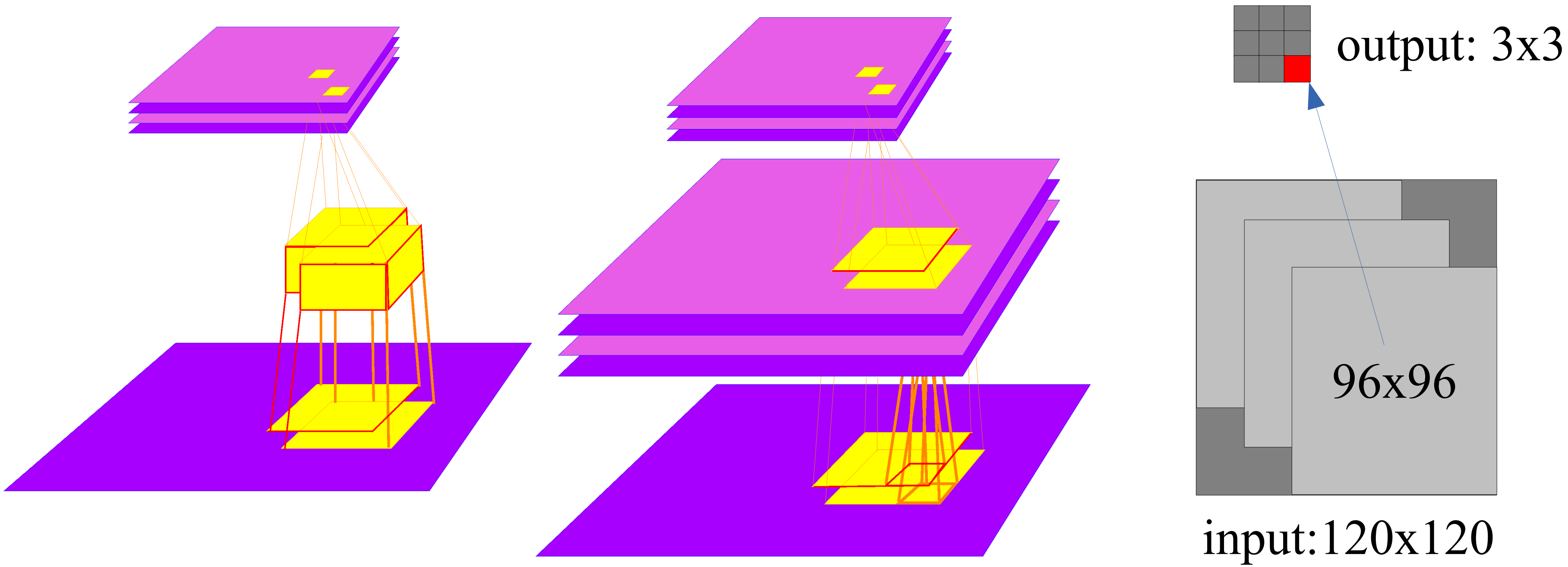
- Network first trained on ImageNet.
- Last layer trained on Pascal VOC, keeping N-1 first layers fixed.

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

Applying a ConvNet on Sliding Windows is Very Cheap!

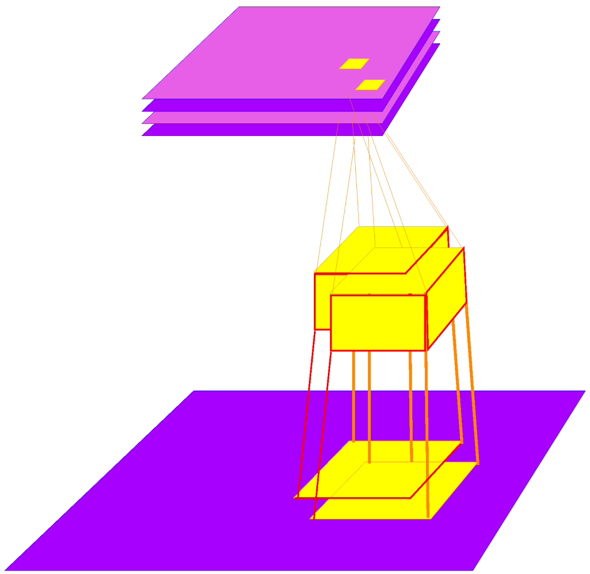


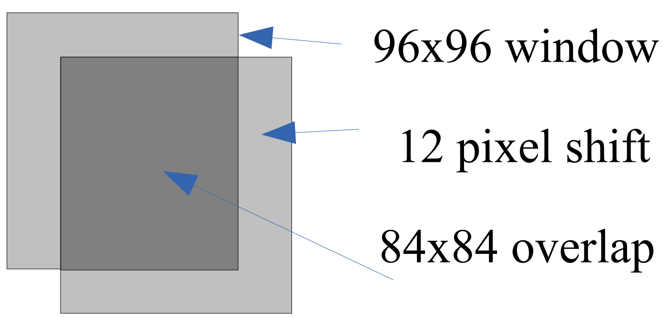
- Traditional Detectors/Classifiers must be applied to every location on a large input image, at multiple scales.
- Convolutional nets can replicated over large images very cheaply.
- The network is applied to multiple scales spaced by 1.5.

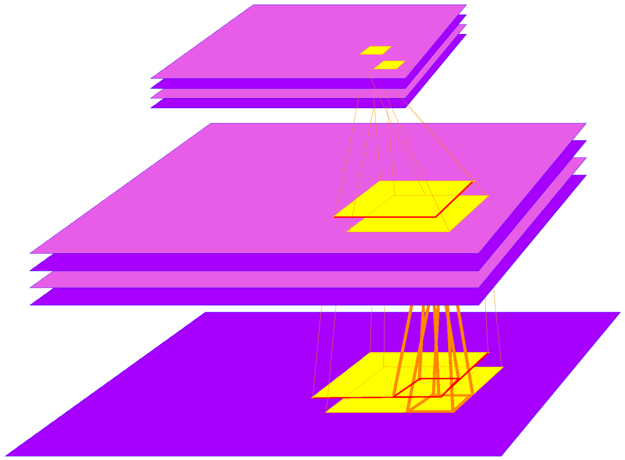
Building a Detector/Recognizer: Replicated Convolutional Nets

Y LeCun

- 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

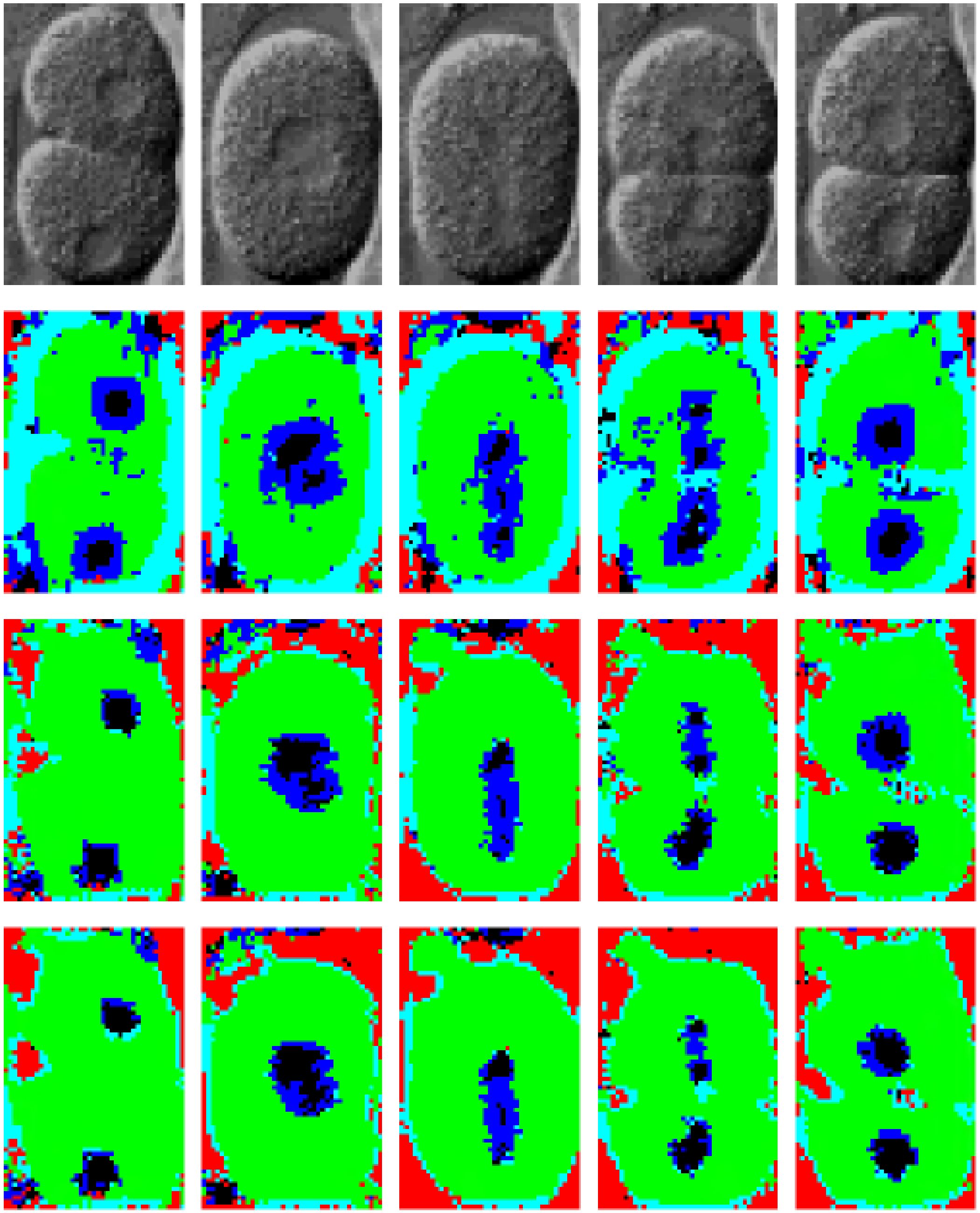






ConvNets for Image Segmentation

- Biological Image Segmentation
  - [Ning et al. IEEE-TIP 2005]
- Pixel labeling with large context using a convnet
- Cleanup using a CRF
  - Similar to a field of expert



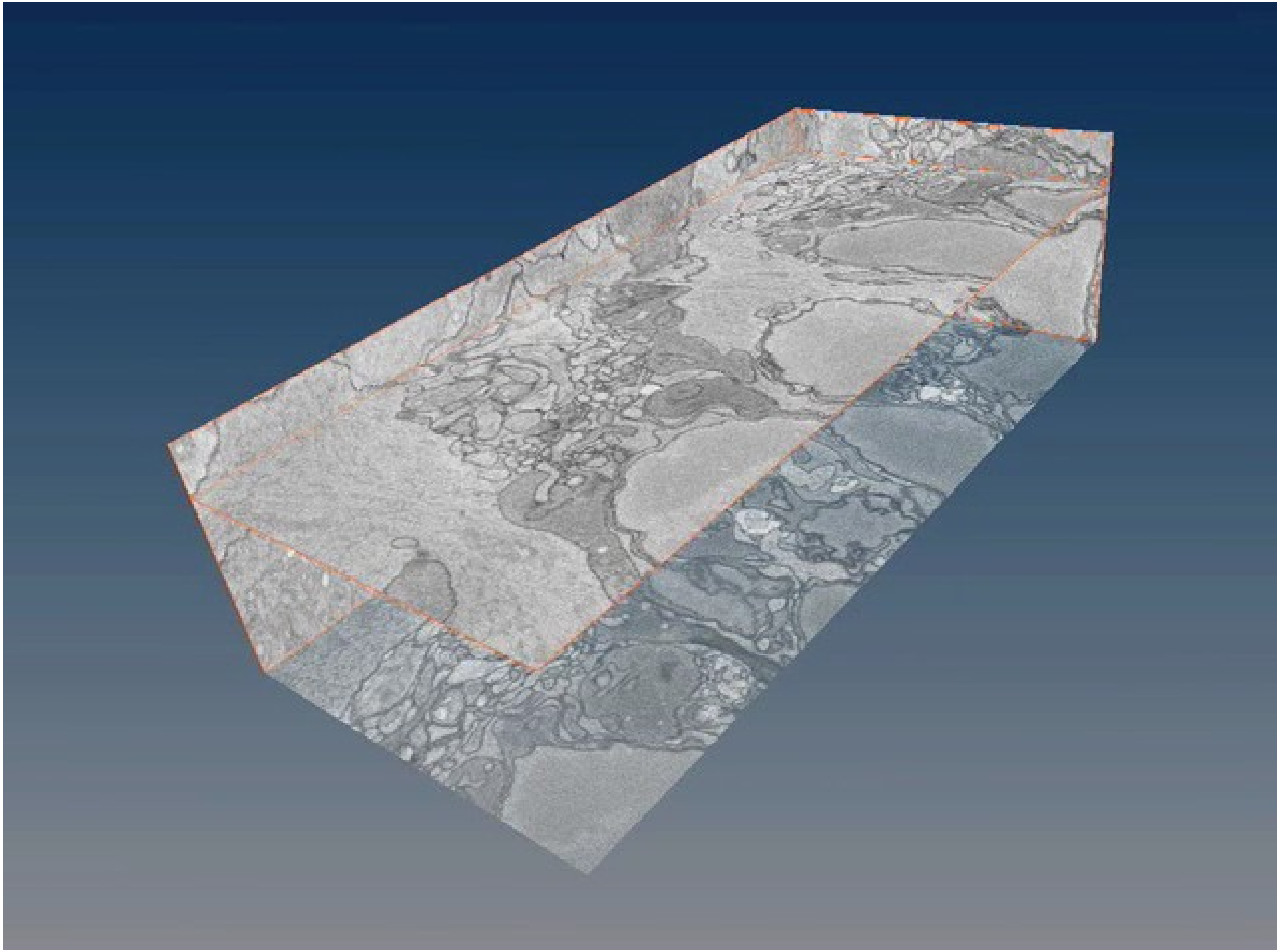
ConvNet in Connectomics [Jain, Turaga, Seung 2007-present]

3D ConvNet

Volumetric

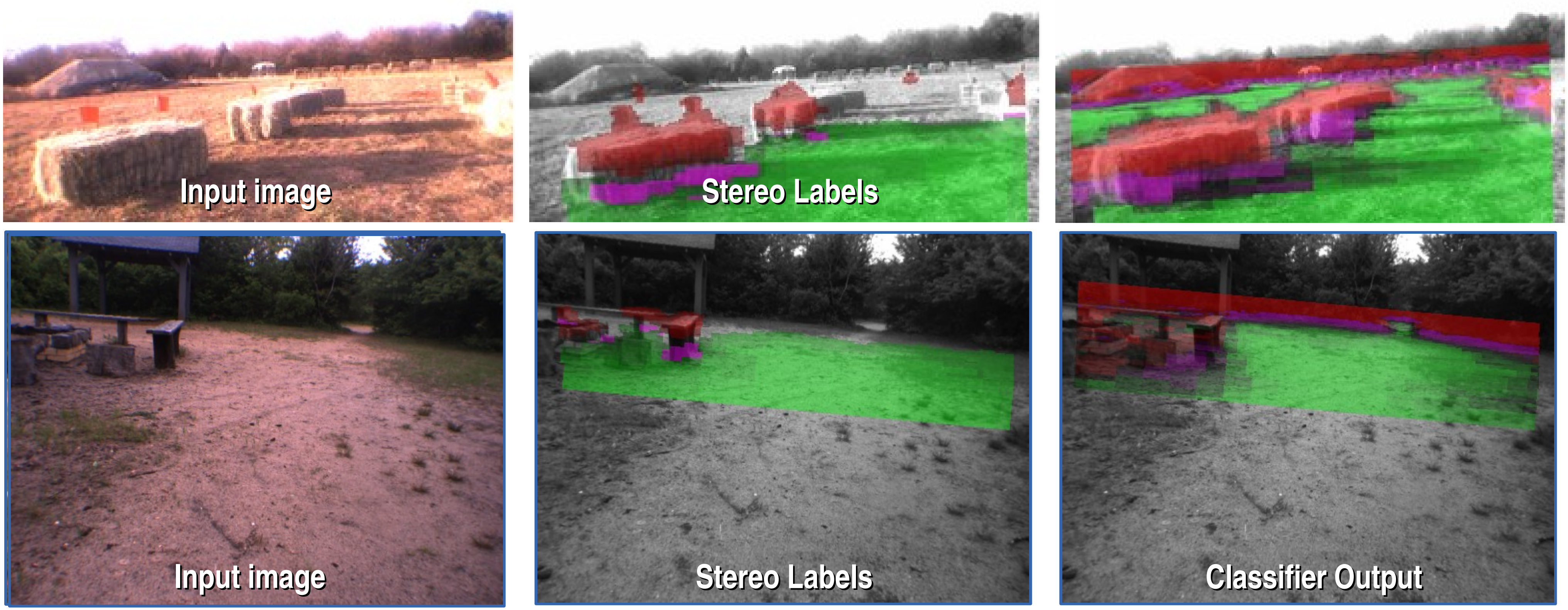
Images

Each voxel labeled as "membrane" or "non-membrane" using a 7x7x7 voxel neighborhood

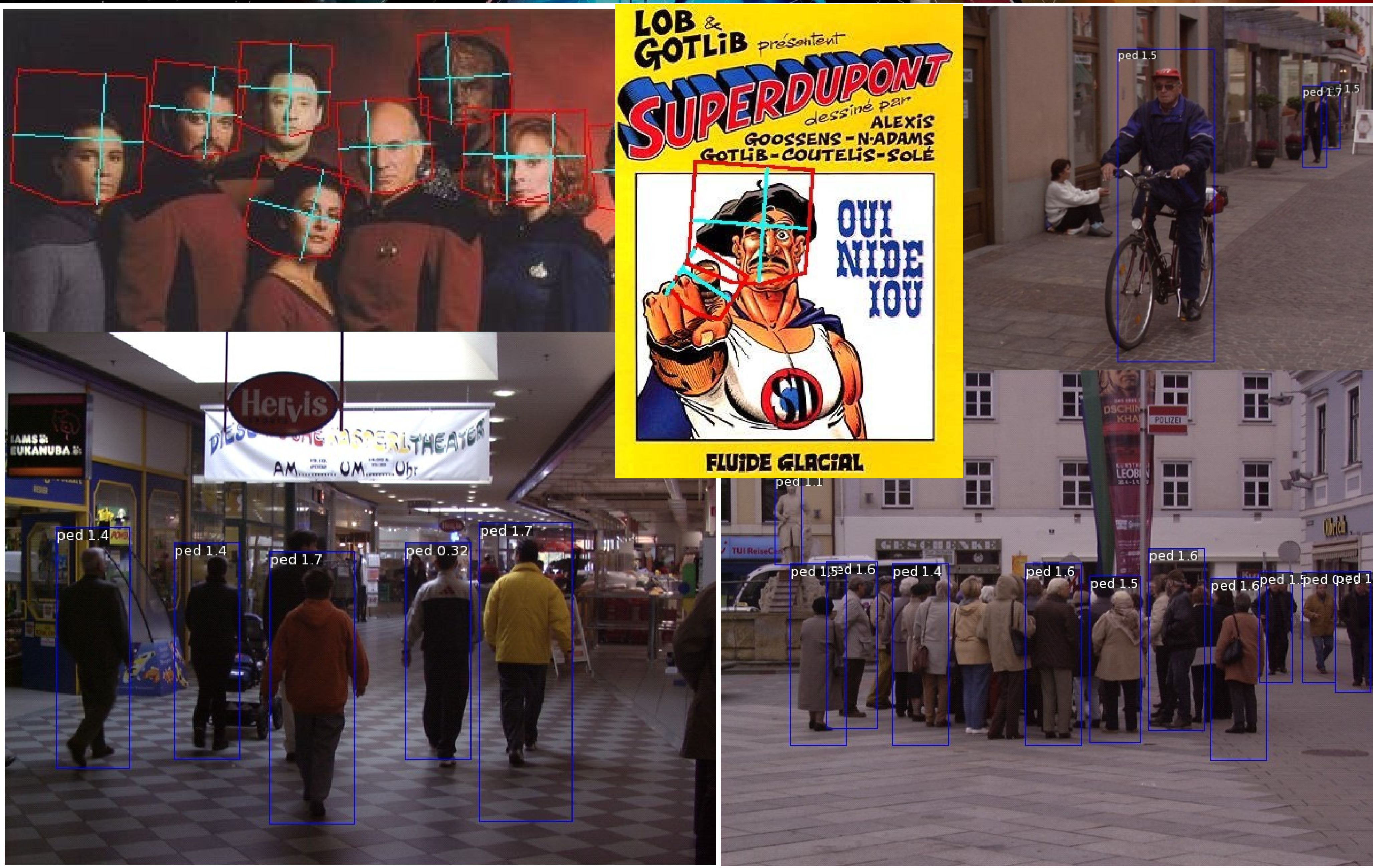


ConvNets for Image Segmentation

- **Image Labeling for Off-Road Robots [Hadsell JFR 2008]**
  - ConvNet labels pixels as one of 3 categories
  - Traversible/flat (green), non traversible (red), foot of obstacle (purple)
  - Labels obtained from stereo vision and SLAM



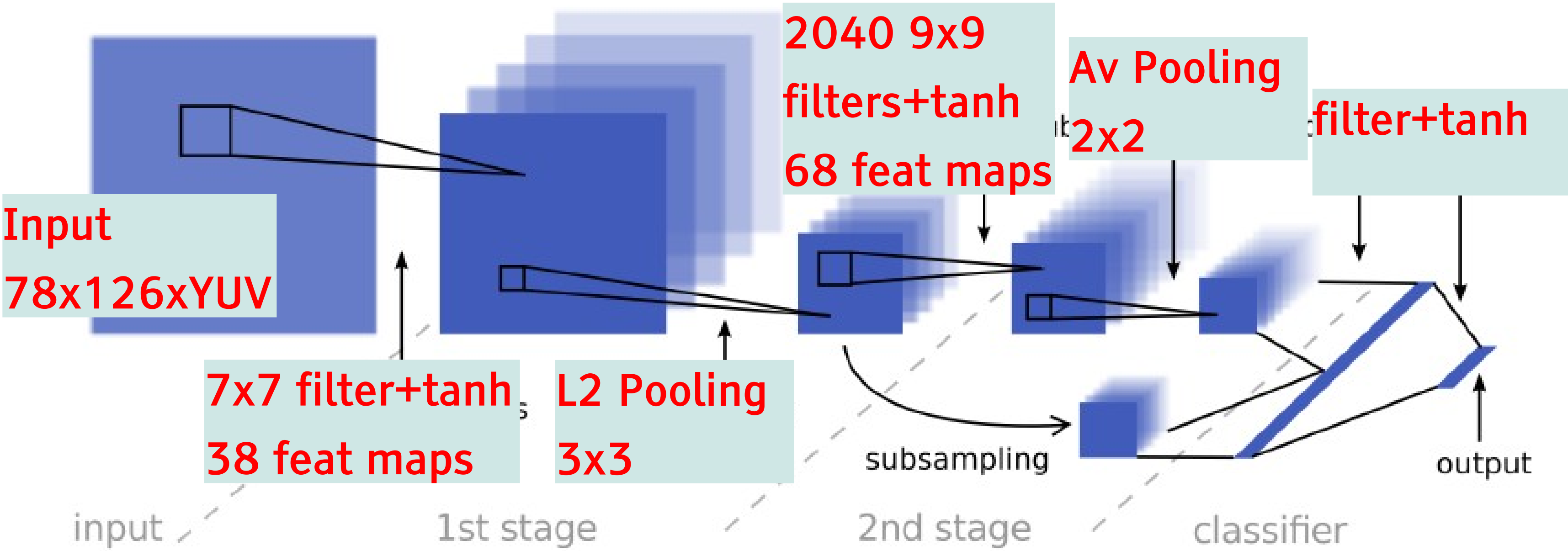
Pedestrian Detection, Face Detection



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

ConvNet Architecture with Multi-Stage Features

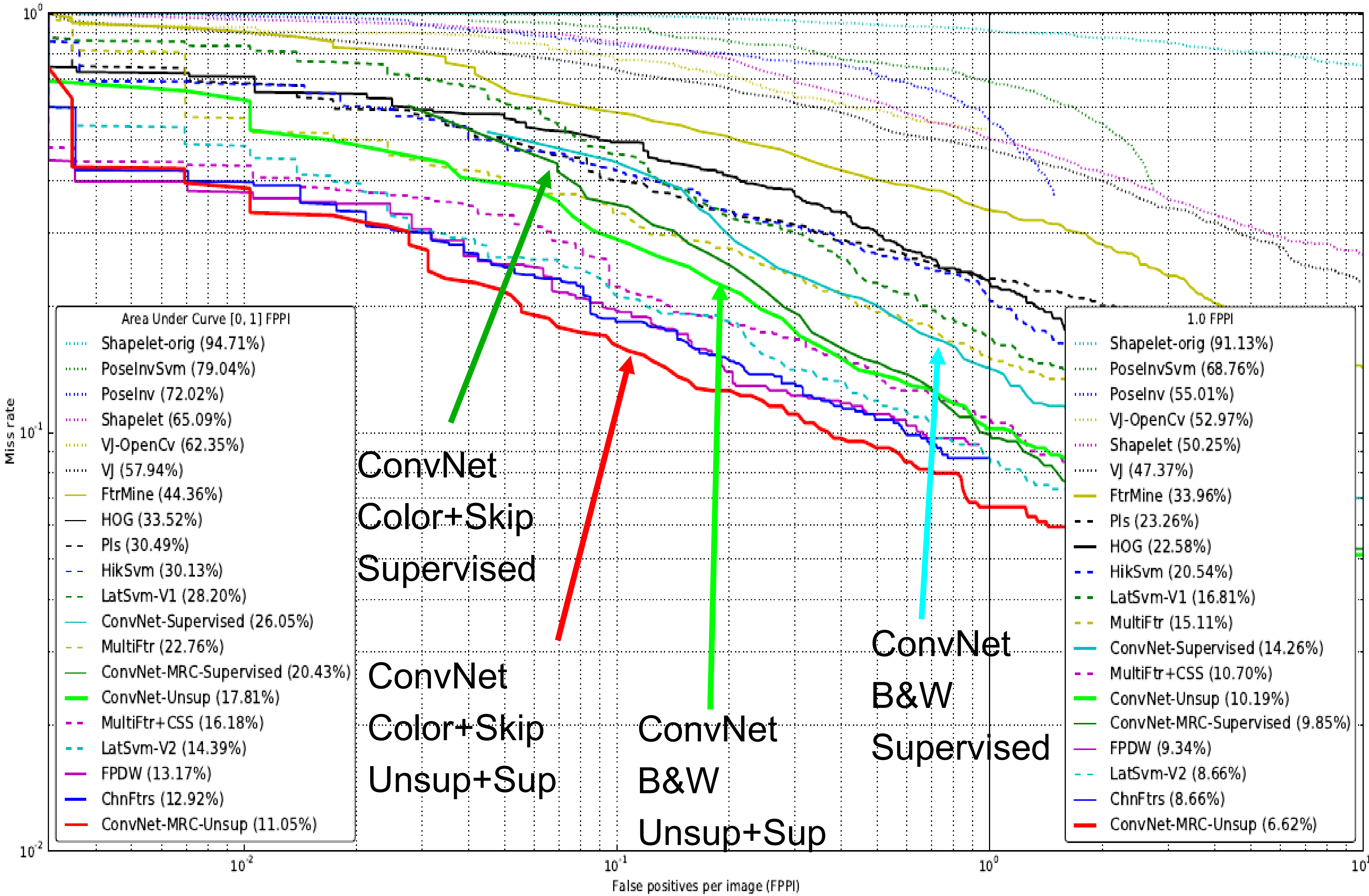
- **Feature maps from all stages are pooled/subsampled and sent to the final classification layers**
  - Pooled low-level features: good for textures and local motifs
  - High-level features: good for "gestalt" and global shape



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]

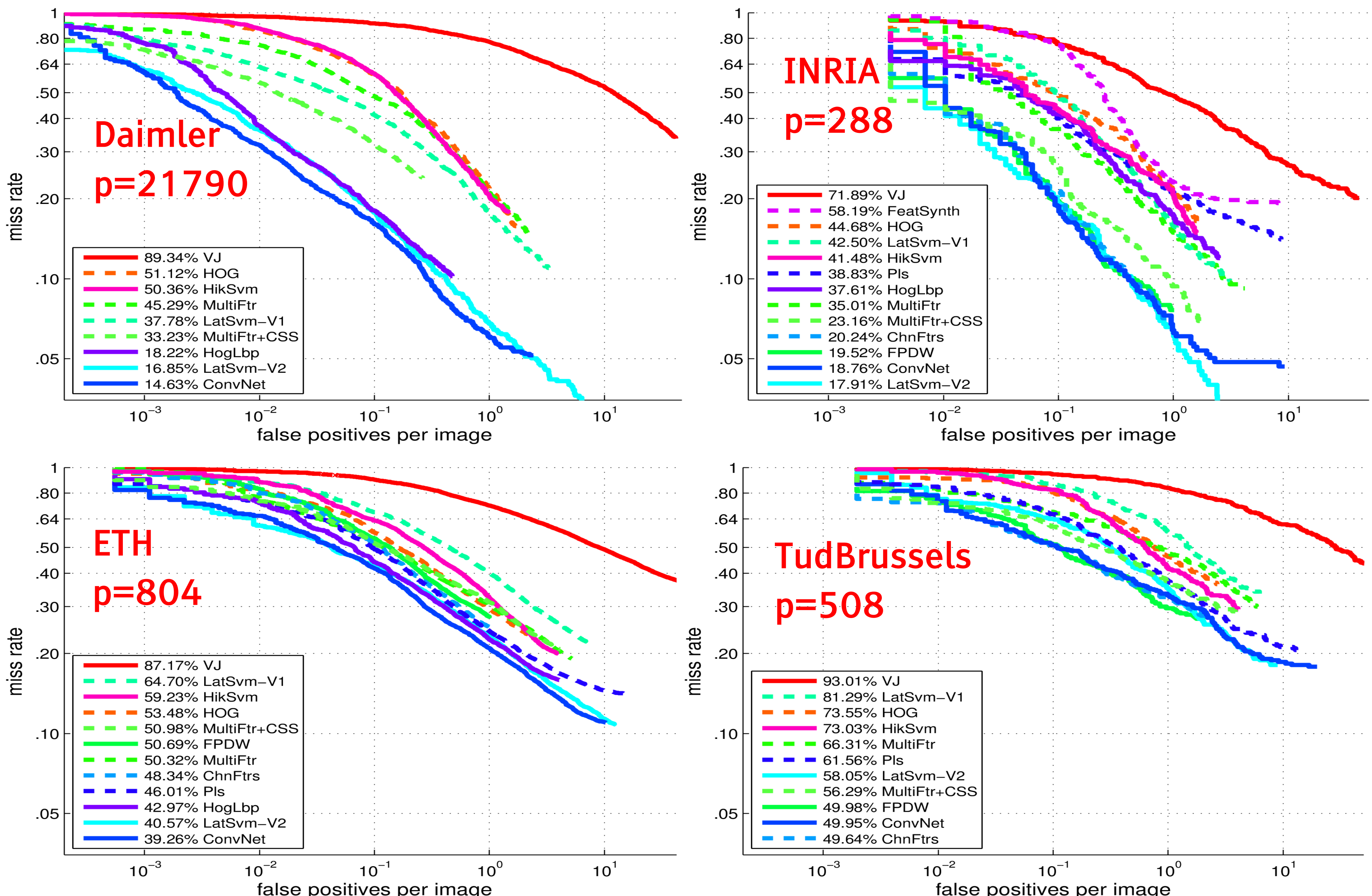
Pedestrian Detection: INRIA Dataset. Miss rate vs false positives



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

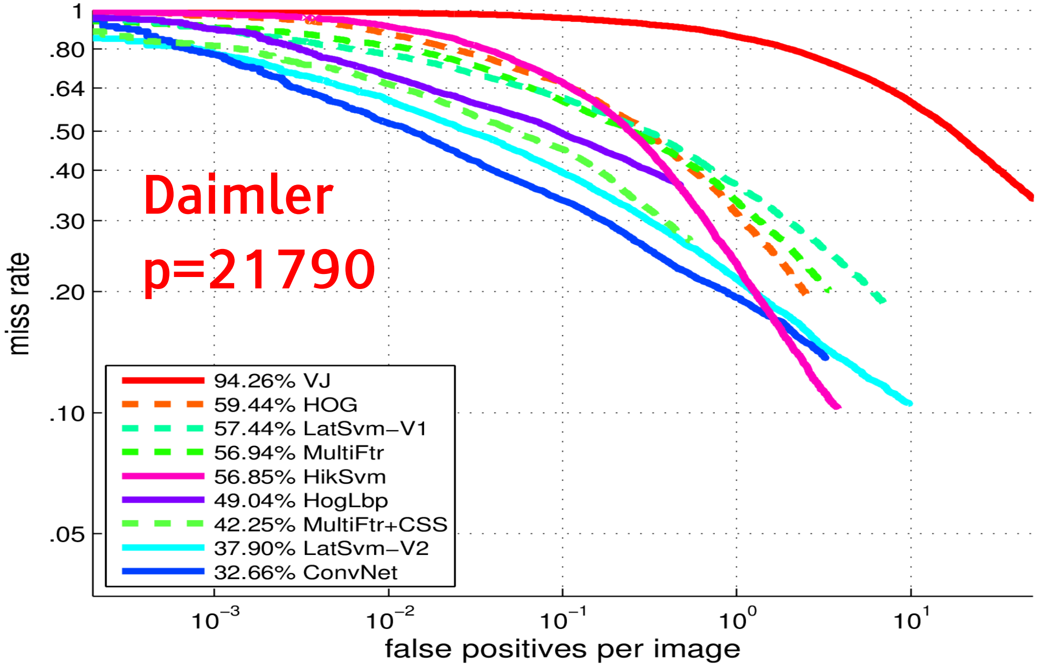
Results on “Near Scale” Images (>80 pixels tall, no occlusions)

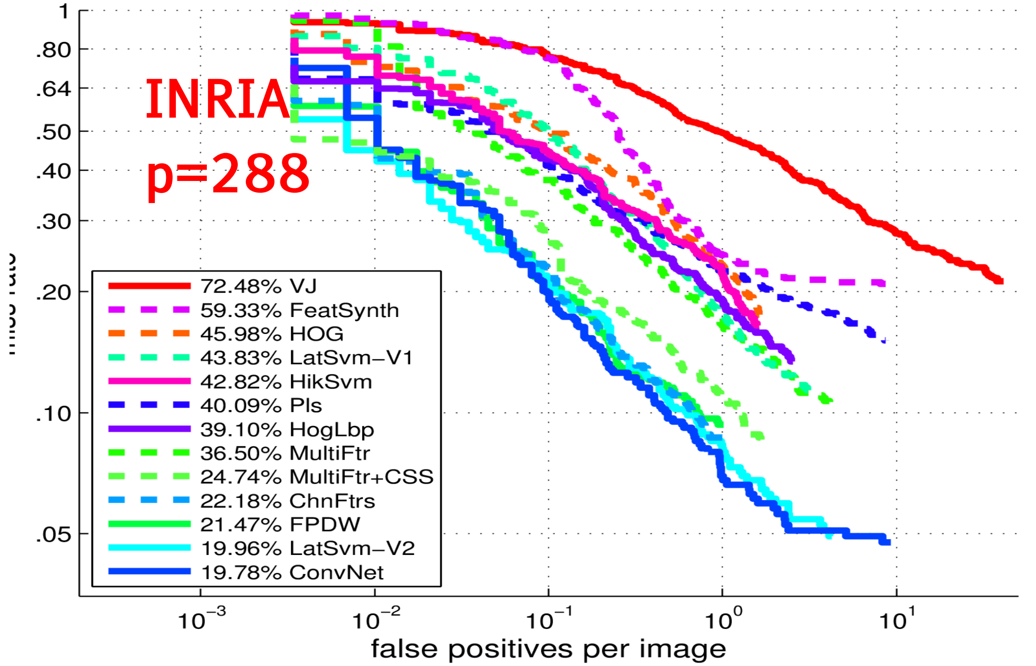
Y LeCun

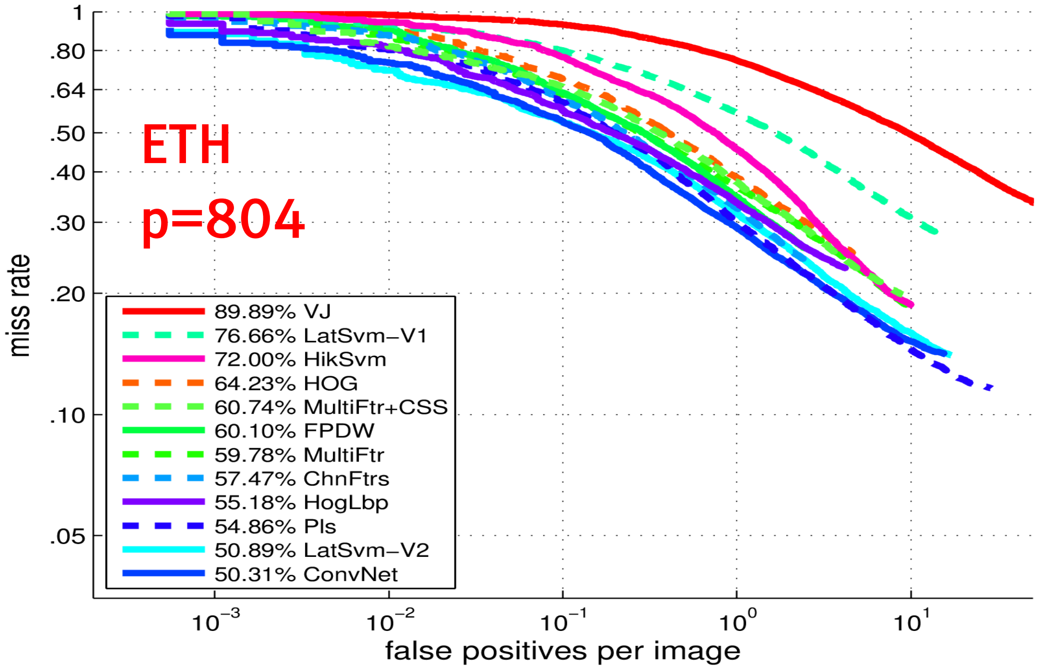


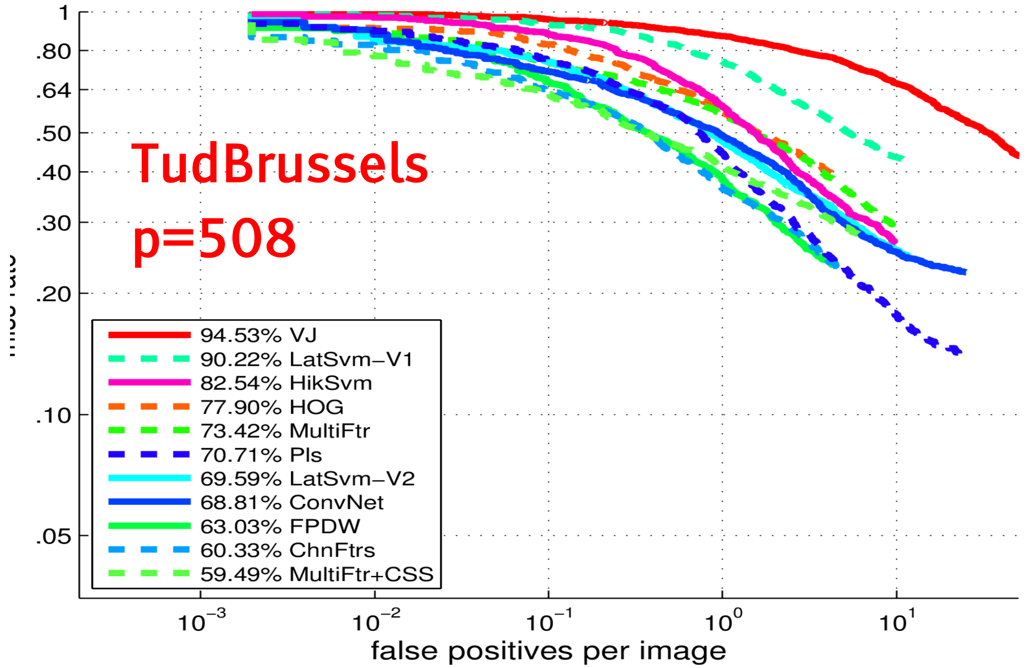
Results on "Reasonable" Images (>50 pixels tall, few occlusions)

Y LeCun



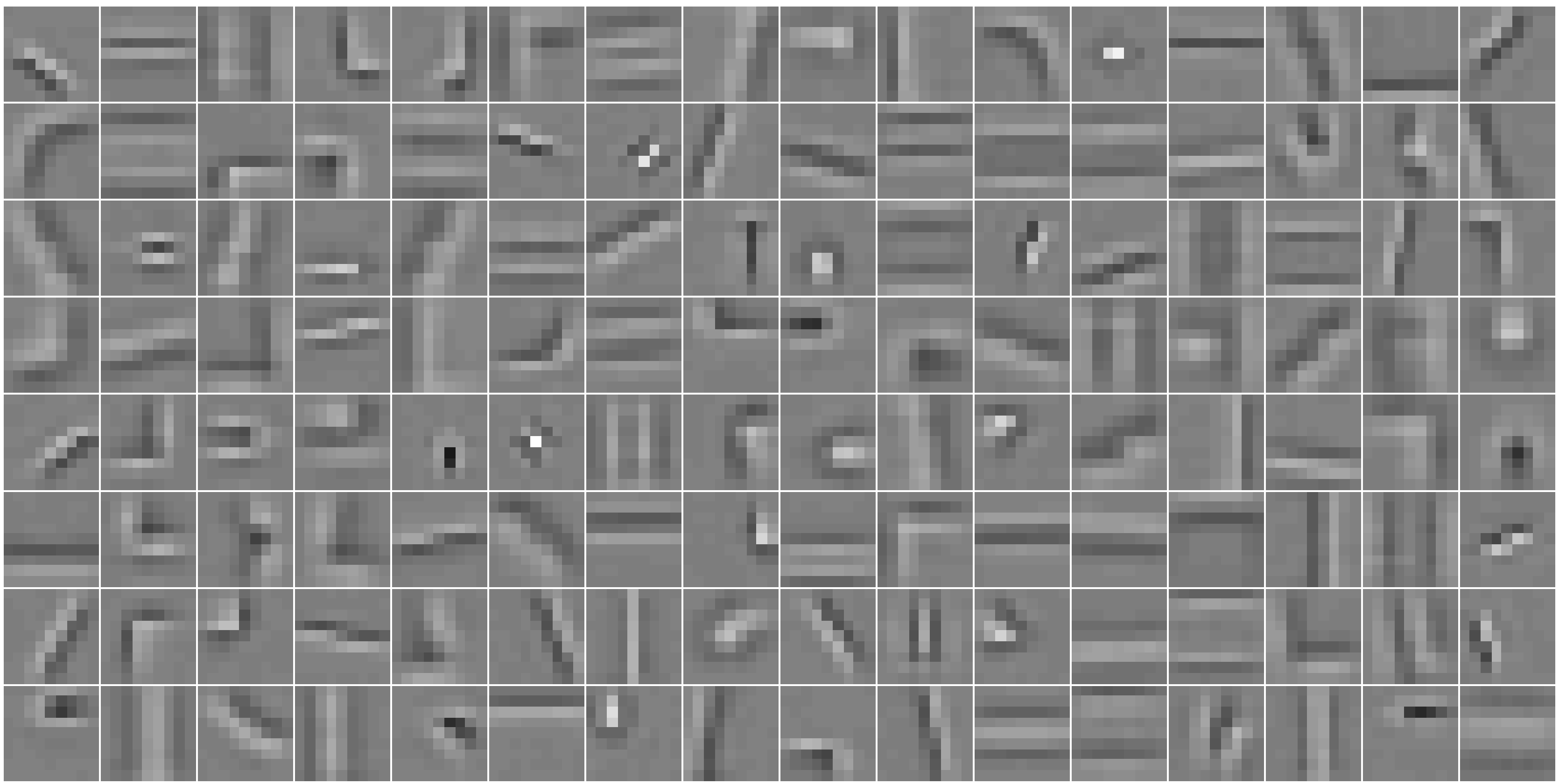






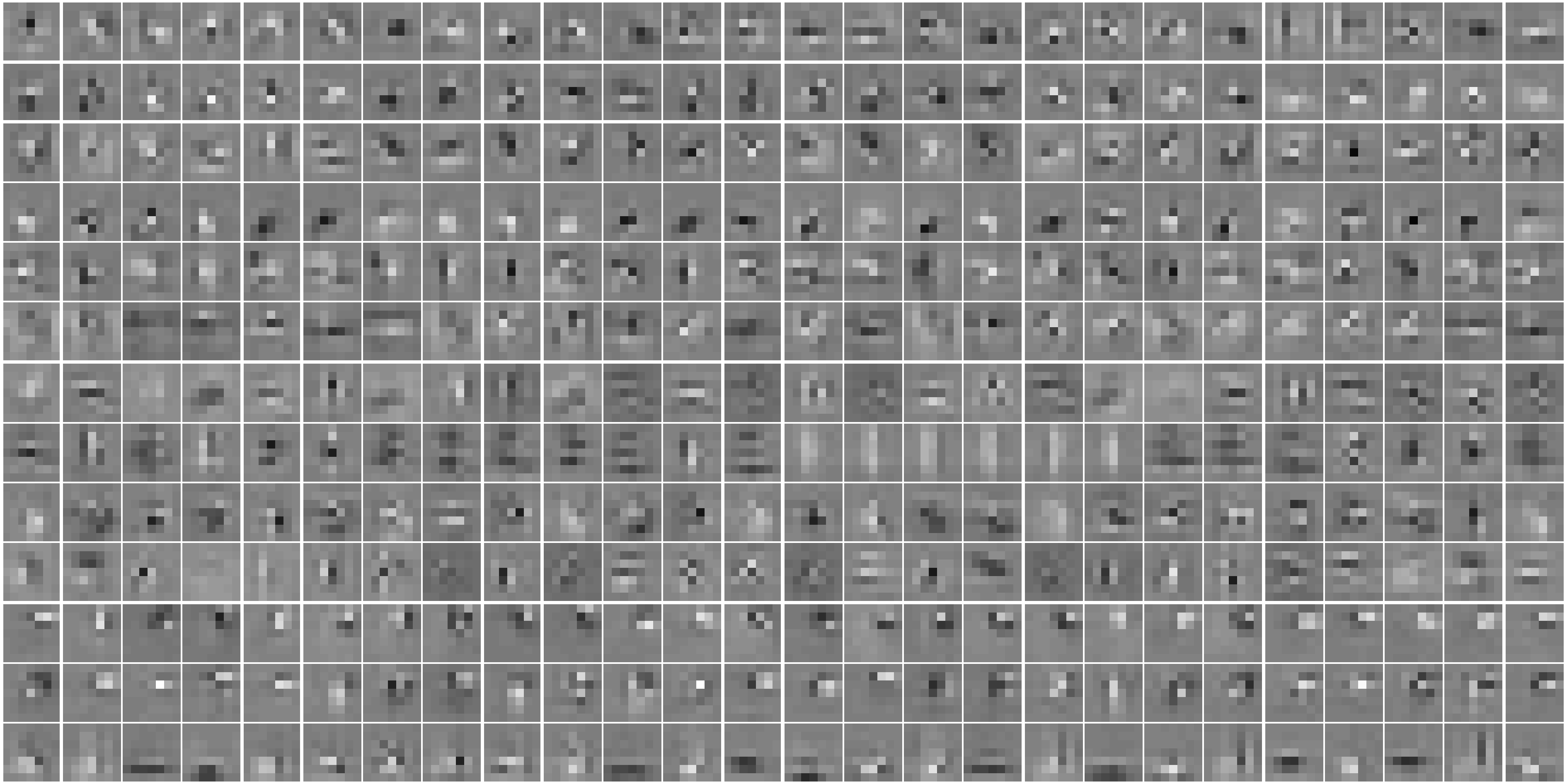
Unsupervised pre-training with convolutional PSD

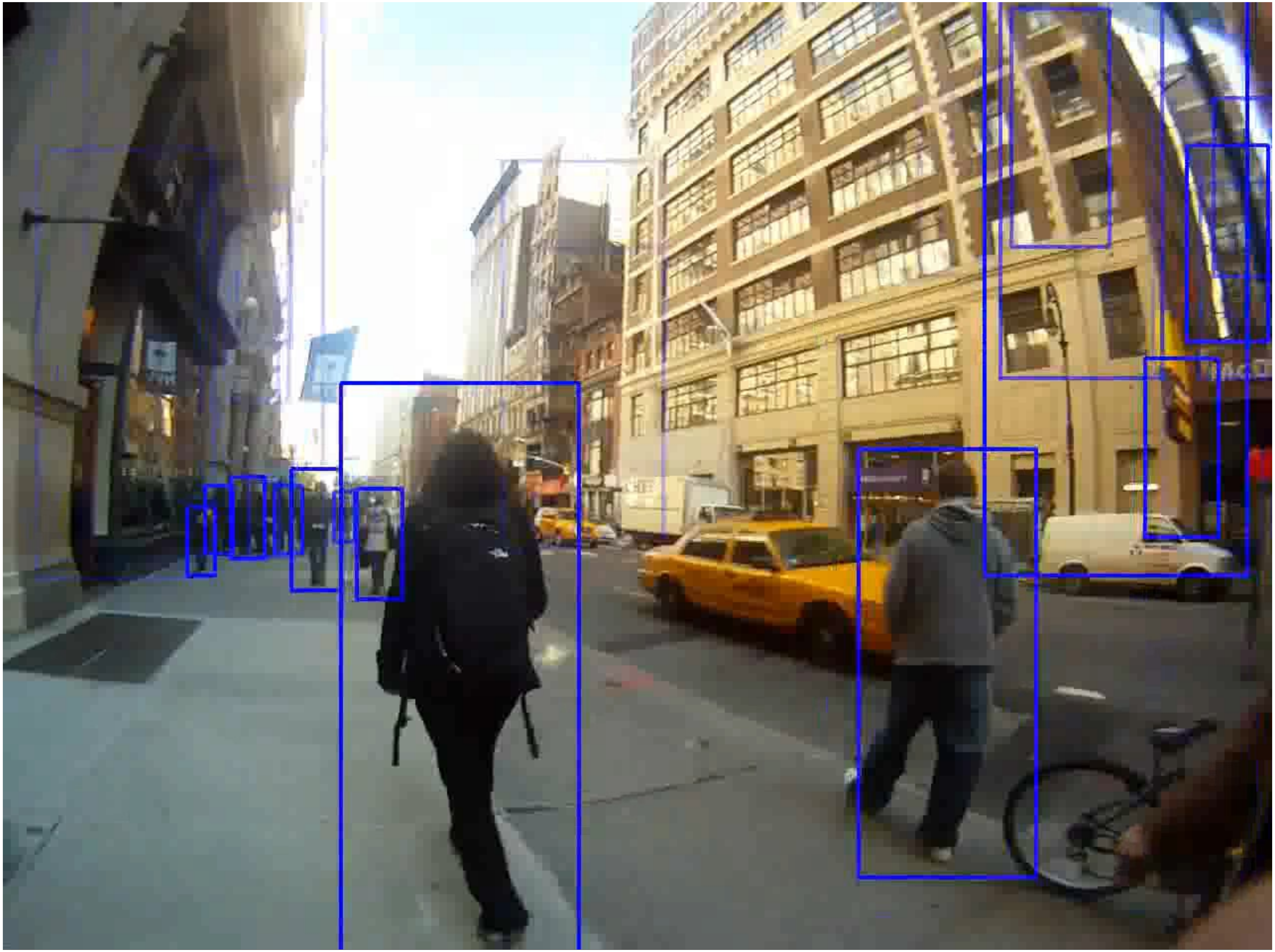
- 128 stage-1 filters on Y channel.
- Unsupervised training with convolutional predictive sparse decomposition



Unsupervised pre-training with convolutional PSD

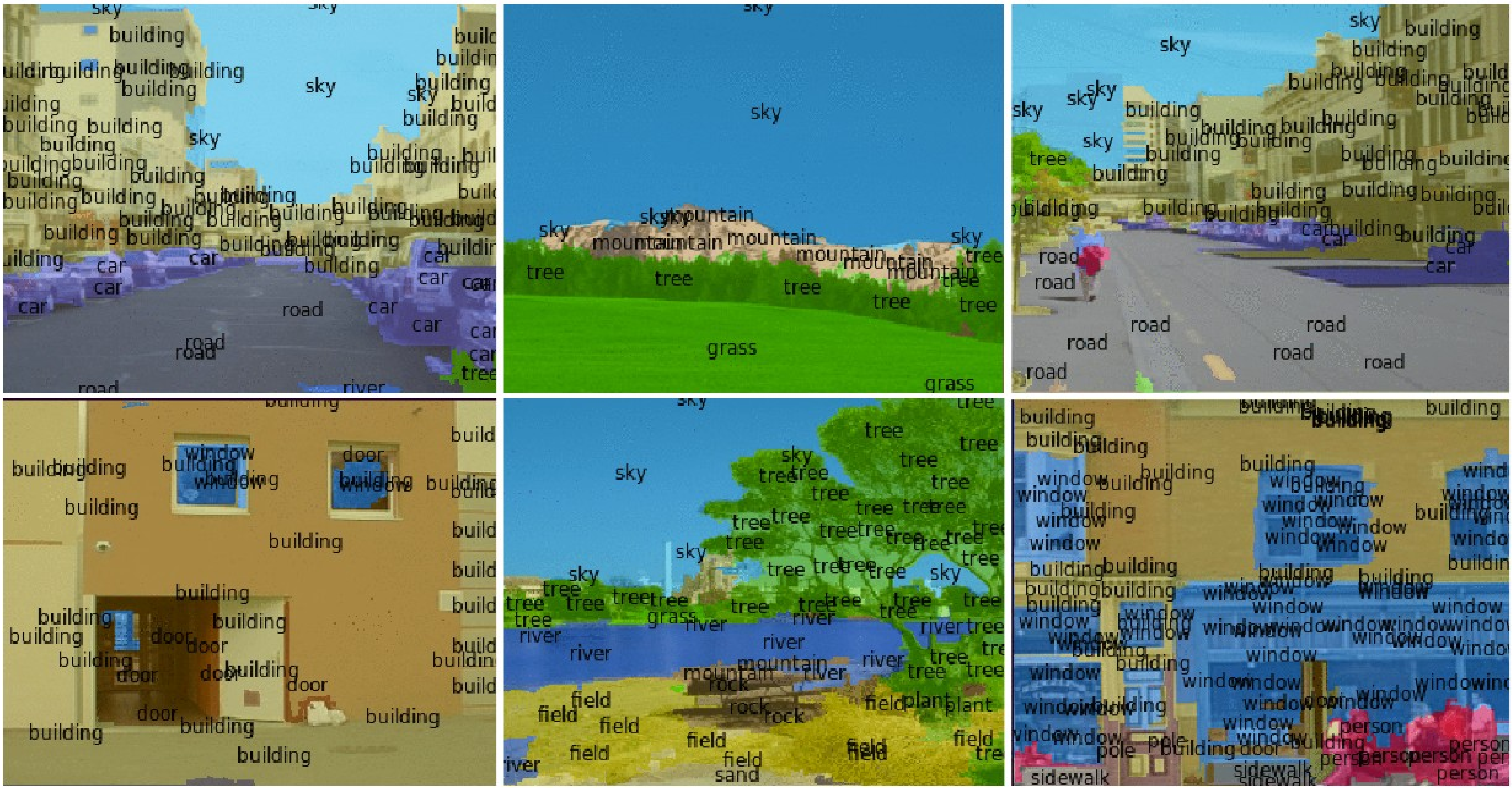
- Stage 2 filters.
- Unsupervised training with convolutional predictive sparse decomposition





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

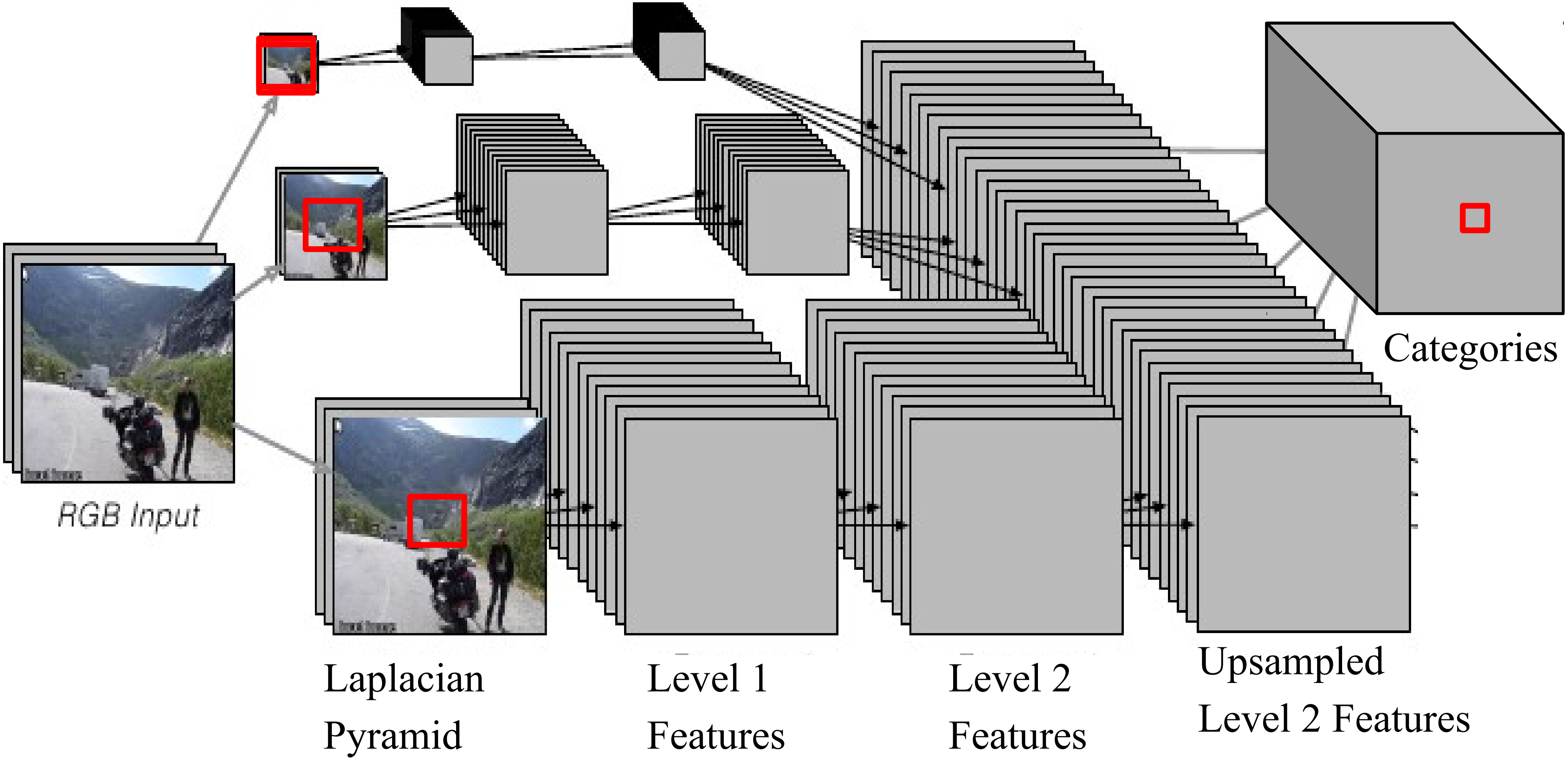
- Would help identify obstacles, targets, landing sites, dangerous areas
- Would help line up depth map with edge maps



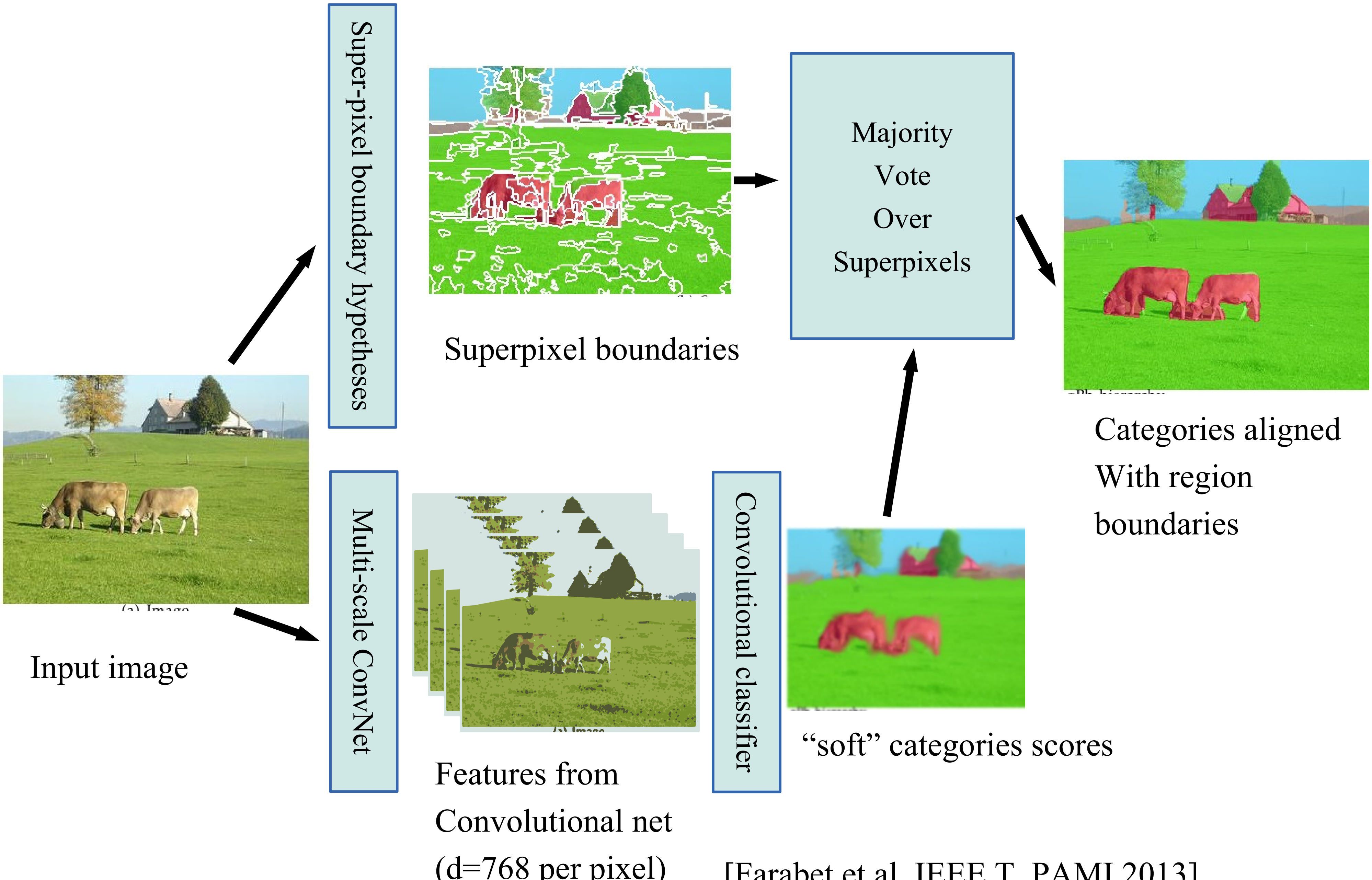
[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling: ConvNet Architecture

- 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

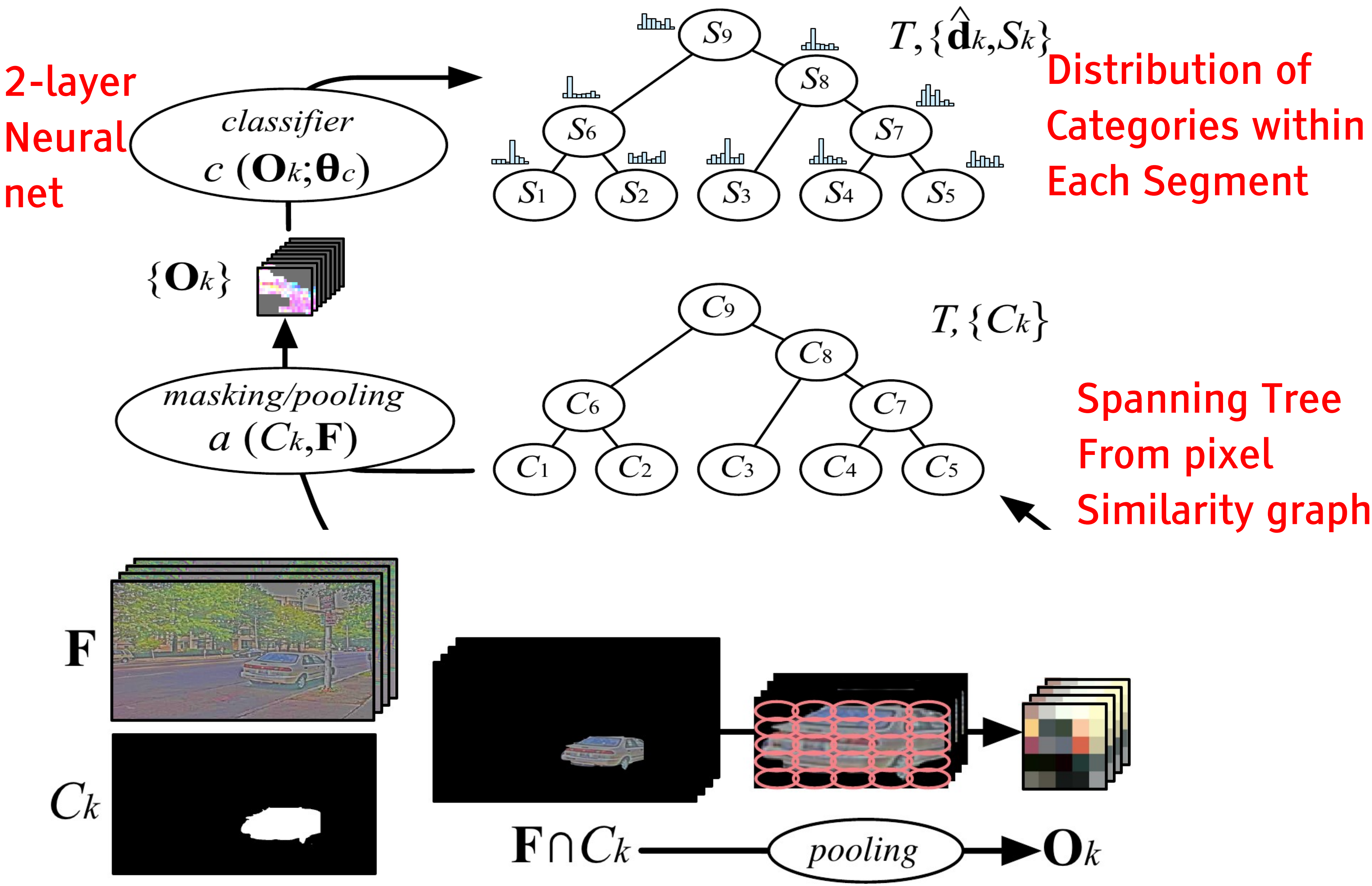


Method 1: majority over super-pixel regions



[Farabet et al. IEEE T. PAMI 2013]

Method 2: optimal cover of purity tree



[Farabet et al. ICML 2012]

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

	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

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

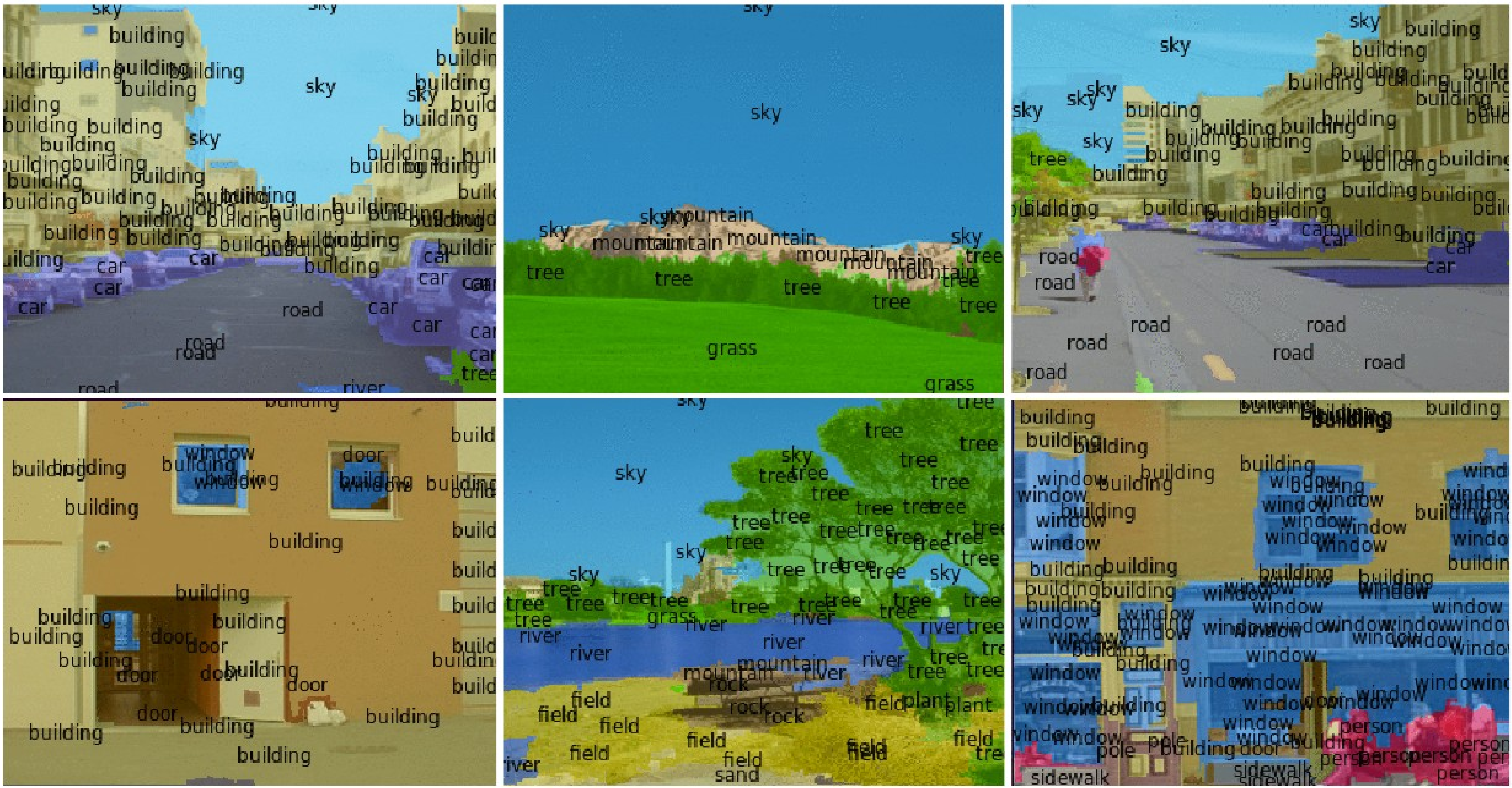
	Pixel Acc.	Class Acc.
Tighe *et al.* 2010 [44]	66.9%	7.6%
raw multiscale net[1]	37.8%	**12.1%**
multiscale net + superpixels[1]	44.1%	**12.4%**
multiscale net + cover[1]	46.4%	**12.5%**
multiscale net + cover[2]	**67.8%**	9.5%

[Farabet et al. IEEE T. PAMI 2012]

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

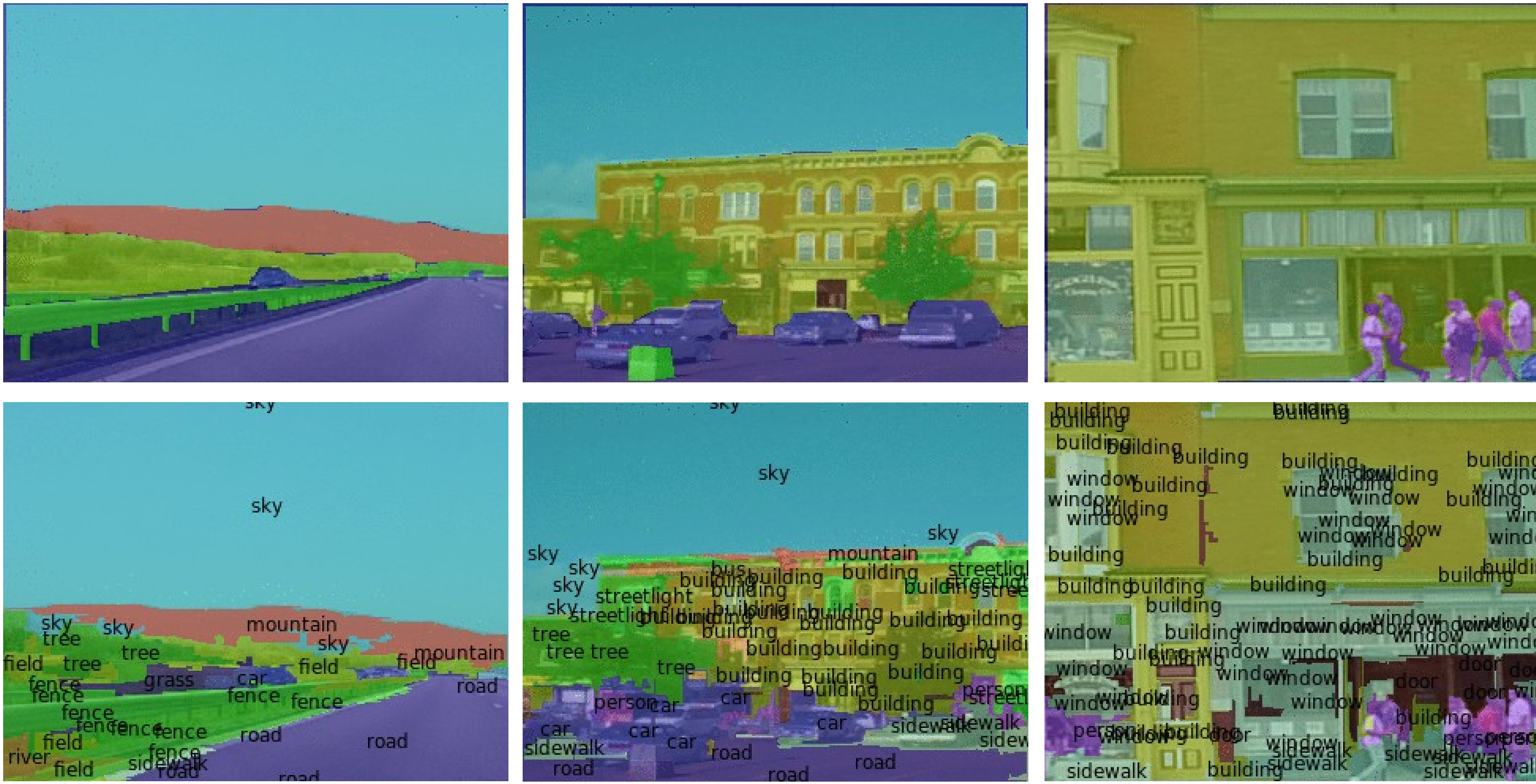
Y LeCun

- Samples from the SIFT-Flow dataset (Liu)



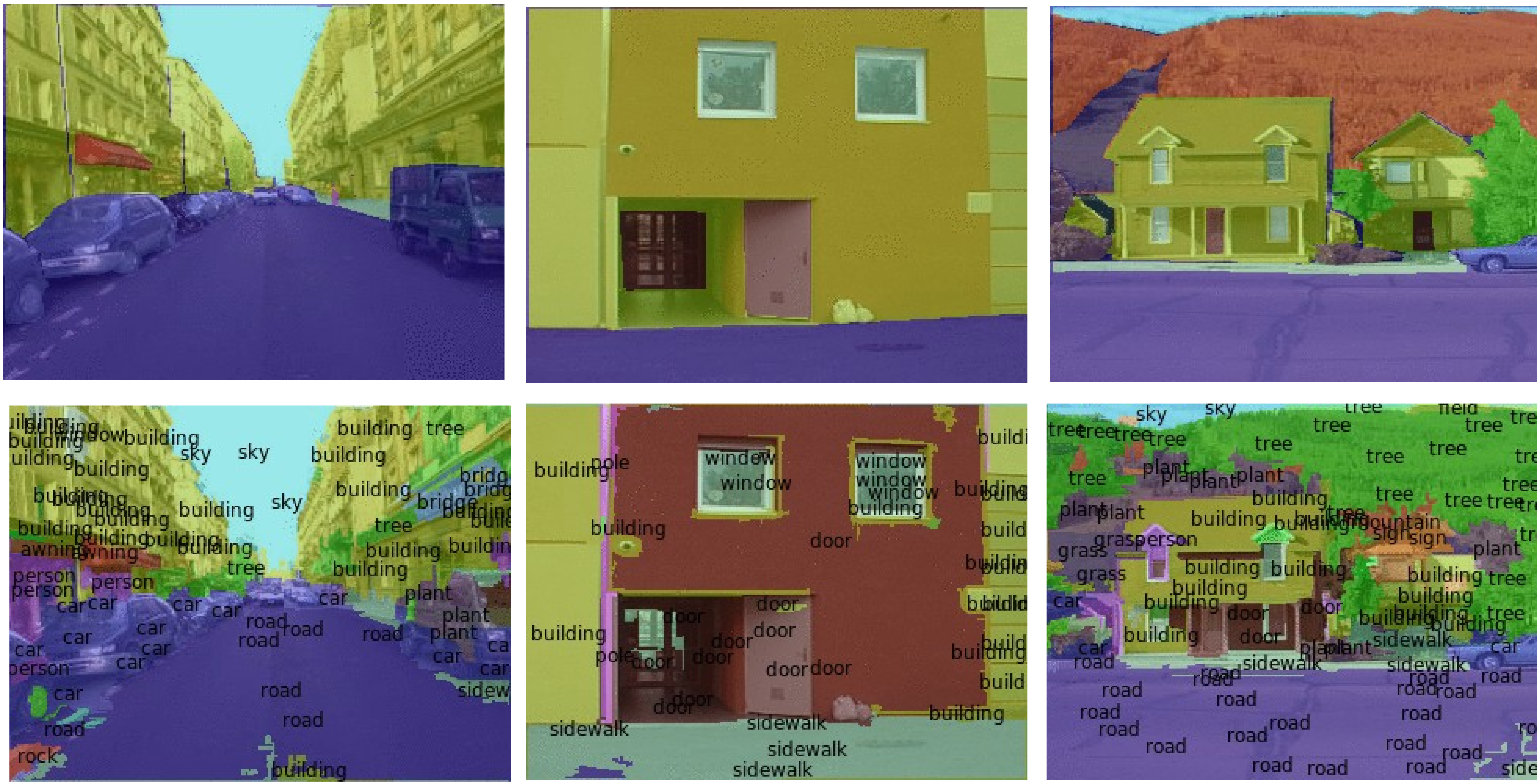
[Farabet et al. ICML 2012, PAMI 2013]

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



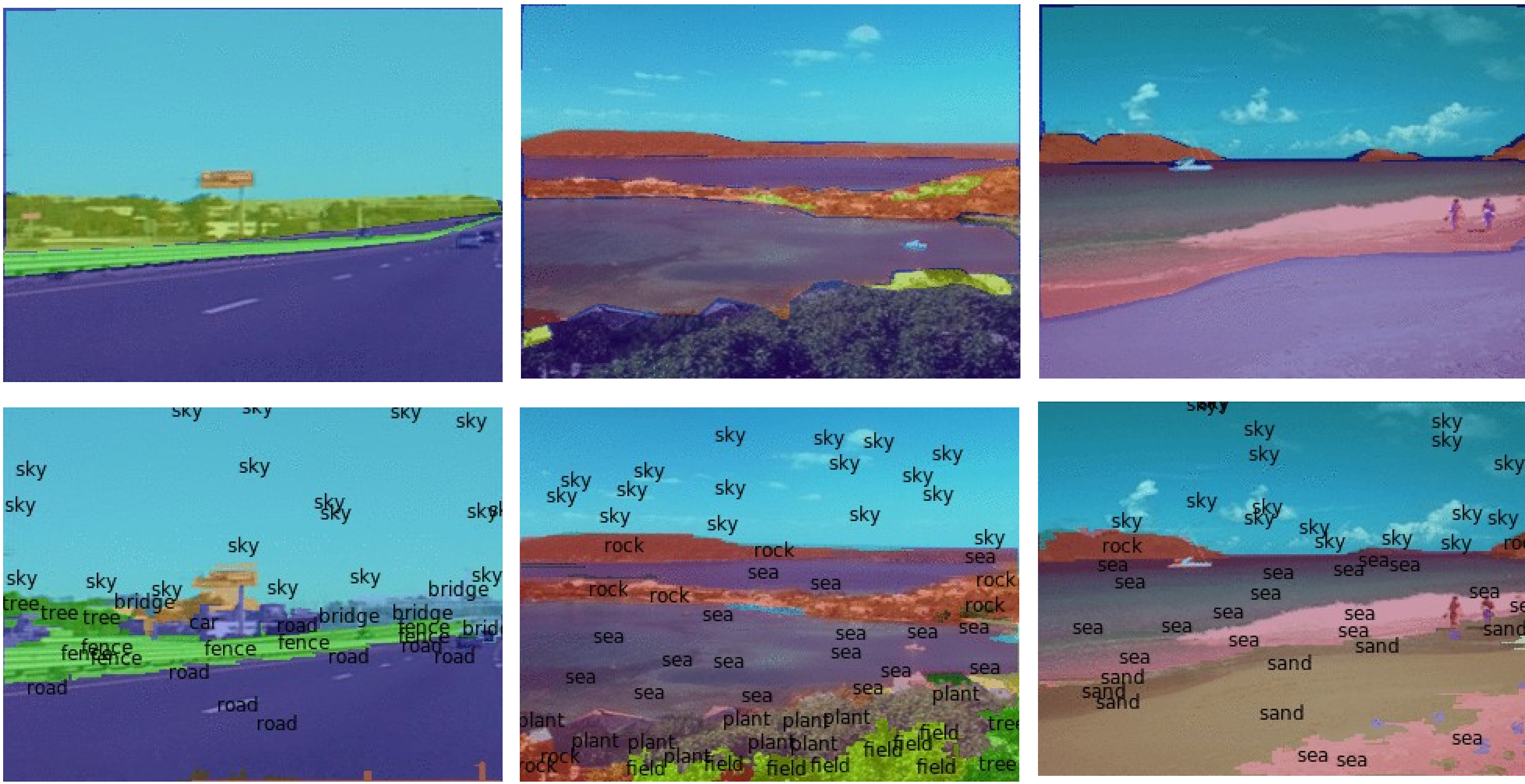
[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling



[Farabet et al. ICML 2012, PAMI 2013]

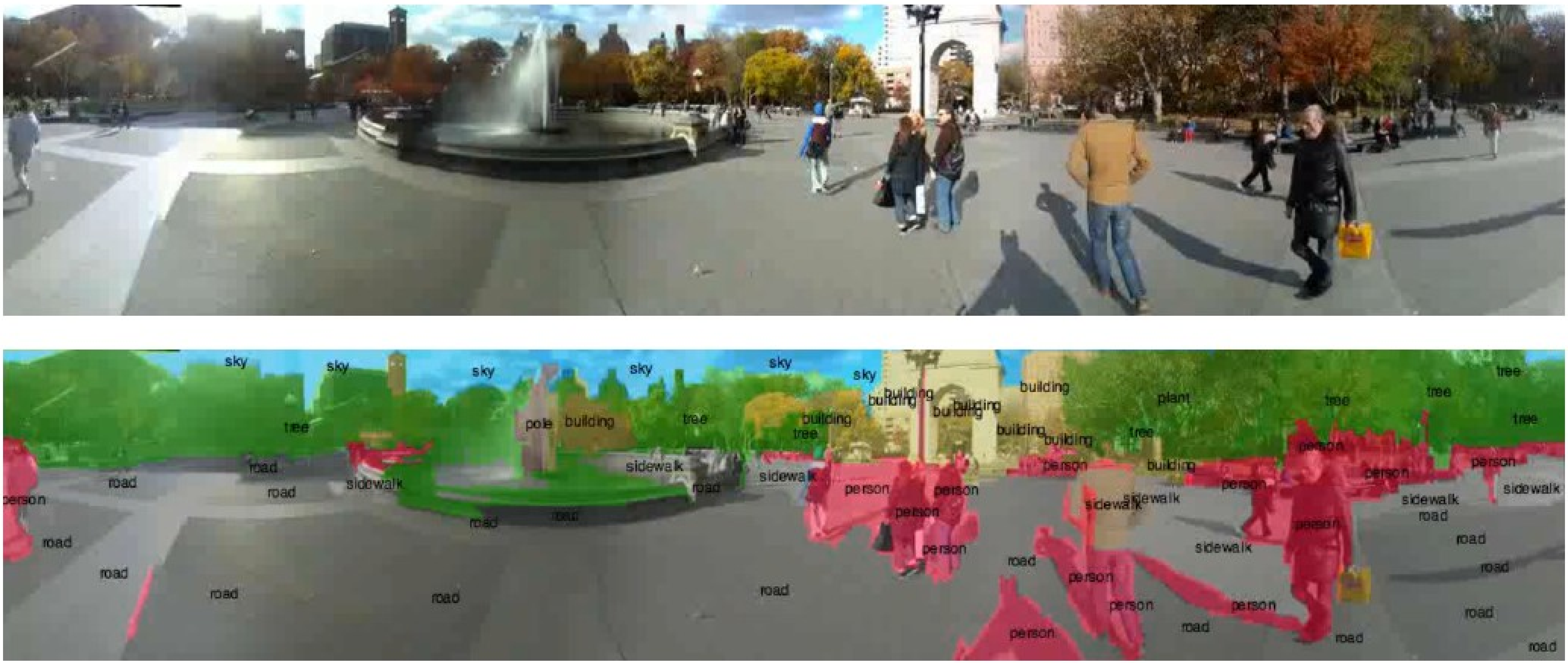
Scene Parsing/Labeling



[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling

Y LeCun

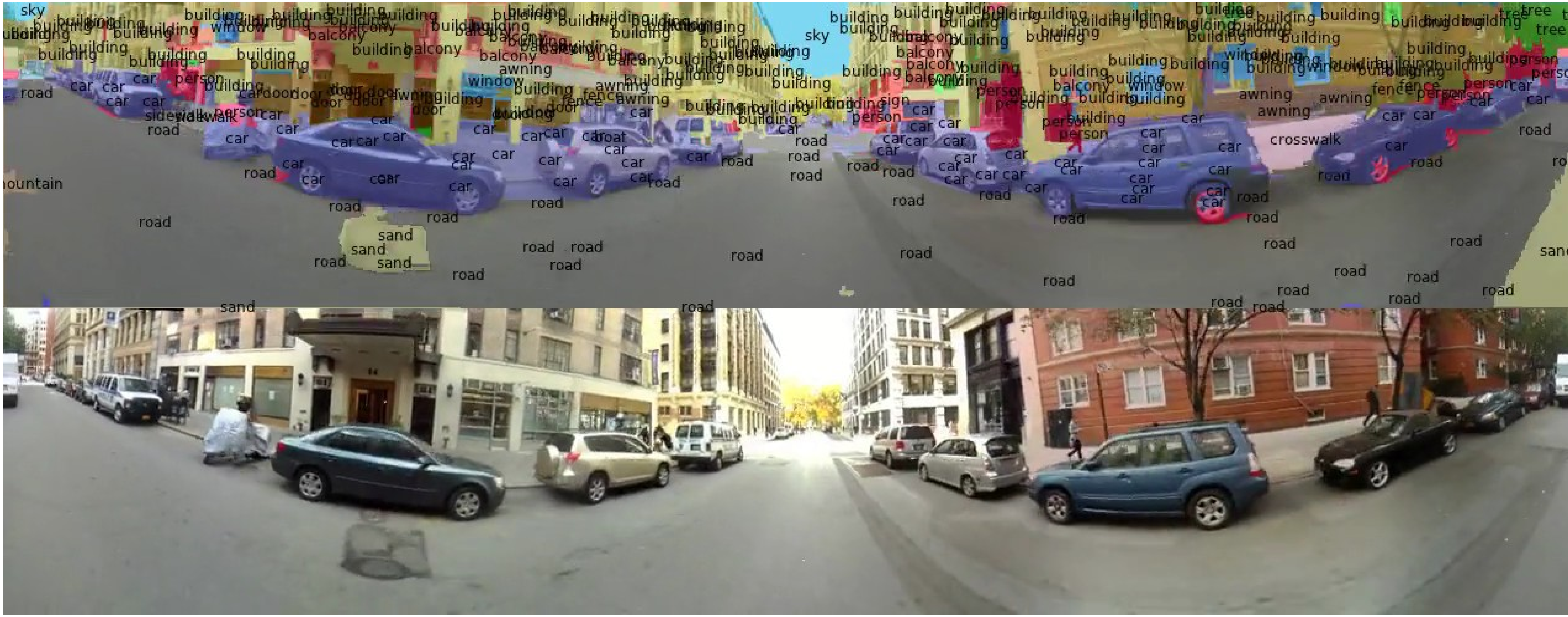


[Farabet et al. ICML 2012, PAMI 2013]

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

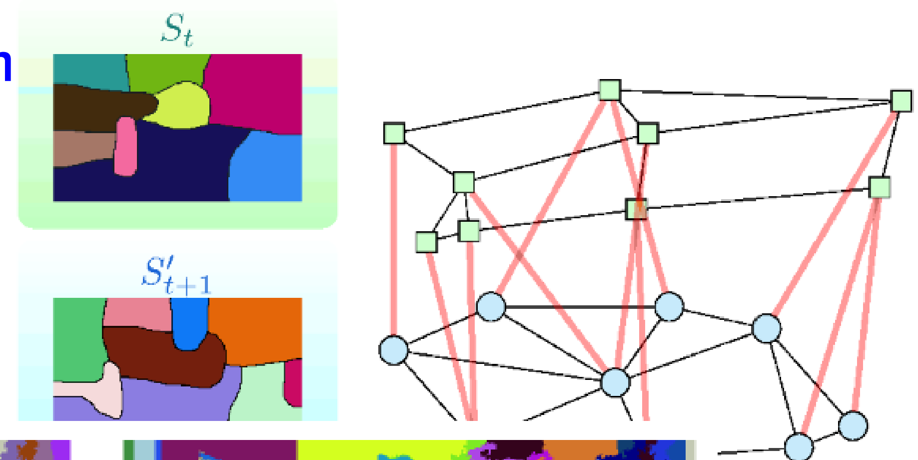
Scene Parsing/Labeling: Temporal Consistency

- Causal method for temporal consistency

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

Temporal Consistency

- Spatio-Temporal Super-Pixel segmentation
  - [Couprie et al ICIP 2013]
  - [Couprie et al JMLR under review]
  - Majority vote over super-pixels



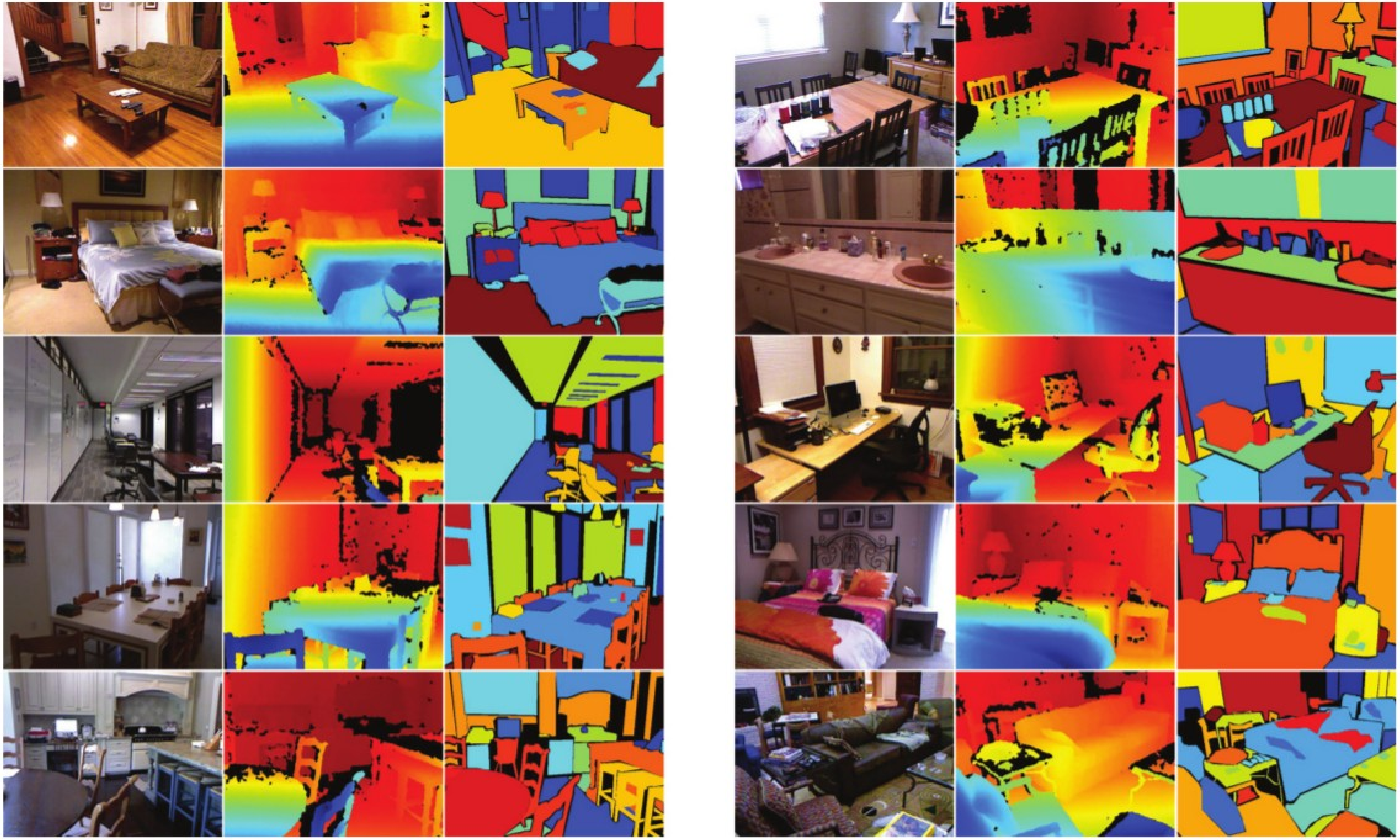
Independent segmentations S'_1, S'_2 and S'_3

Temporally consistent segmentations $S_1(=S'_1)$, S_2, and S_3

NYU RGB-Depth Indoor Scenes Dataset

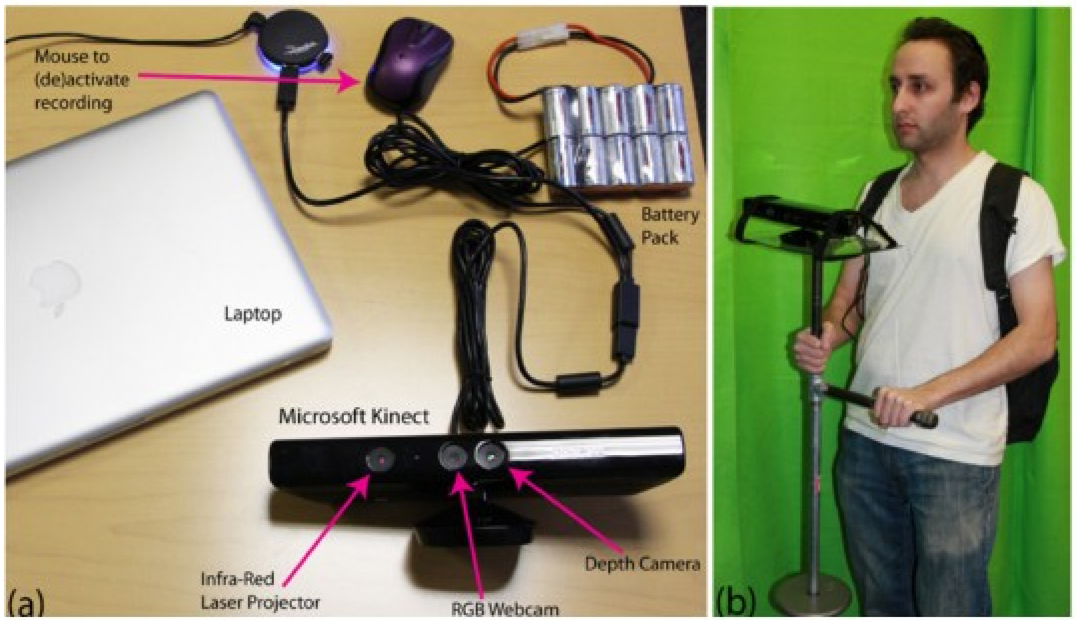
- 407024 RGB-D images of apartments
- 1449 labeled frames, 894 object categories

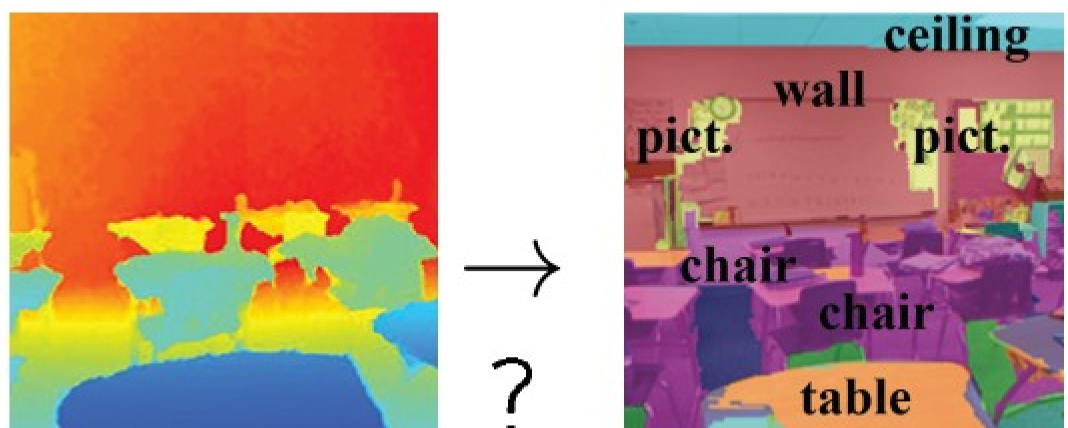
[Silberman et al. 2012]



NYU RGB-D Dataset

- Captured with a Kinect on a steadycam





Results

	Class Occurrences	Multiscale Convnet Acc. Farabet et al. (2013)	MultiScl. Cnet +depth Acc.
bed	4.4%	30.3	**38.1**
objects	7.1 %	**10.9**	8.7
chair	3.4%	**44.4**	34.1
furnit.	12.3%	28.5	**42.4**
ceiling	1.4%	33.2	**62.6**
floor	9.9%	68.0	**87.3**
deco.	3.4%	38.5	**40.4**
sofa	3.2%	**25.8**	24.6
table	3.7%	**18.0**	10.2
wall	24.5%	**89.4**	86.1
window	5.1%	**37.8**	15.9
books	2.9%	**31.7**	13.7
TV	1.0%	**18.8**	6.0
unkn.	17.8%	-	-
Avg. Class Acc.	-	35.8	**36.2**
Pixel Accuracy (mean)	-	51.0	**52.4**
Pixel Accuracy (median)	-	51.7	**52.9**
Pixel Accuracy (std. dev.)	-	15.2	15.2

Results

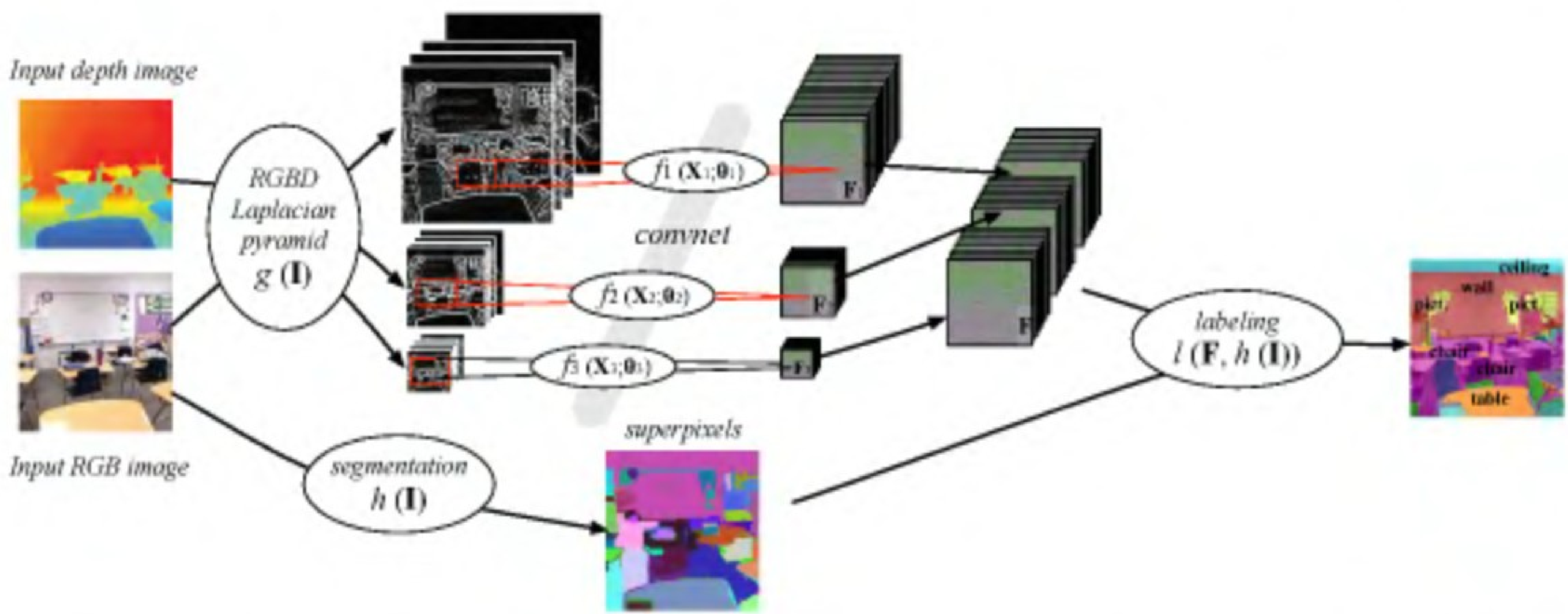
- **Depth helps a bit**
  - Helps a lot for floor and props
  - Helps surprisingly little for structures, and hurts for furniture

	Ground	Furniture	Props	Structure	Class Acc.	Pixel Acc.	Comput. time (s)
Silberman et al. (2012)	68	**70**	**42**	59	59.6	58.6	>3
Cadena and Kosecka (2013)	**87.9**	**64.1**	31.0	77.8	**65.2**	**66.9**	1.7
Multiscale convnet	68.1	51.1	29.9	**87.8**	59.2	63.0	**0.7**
Multiscale+depth convnet	**87.3**	45.3	**35.5**	**86.1**	**63.5**	**64.5**	**0.7**

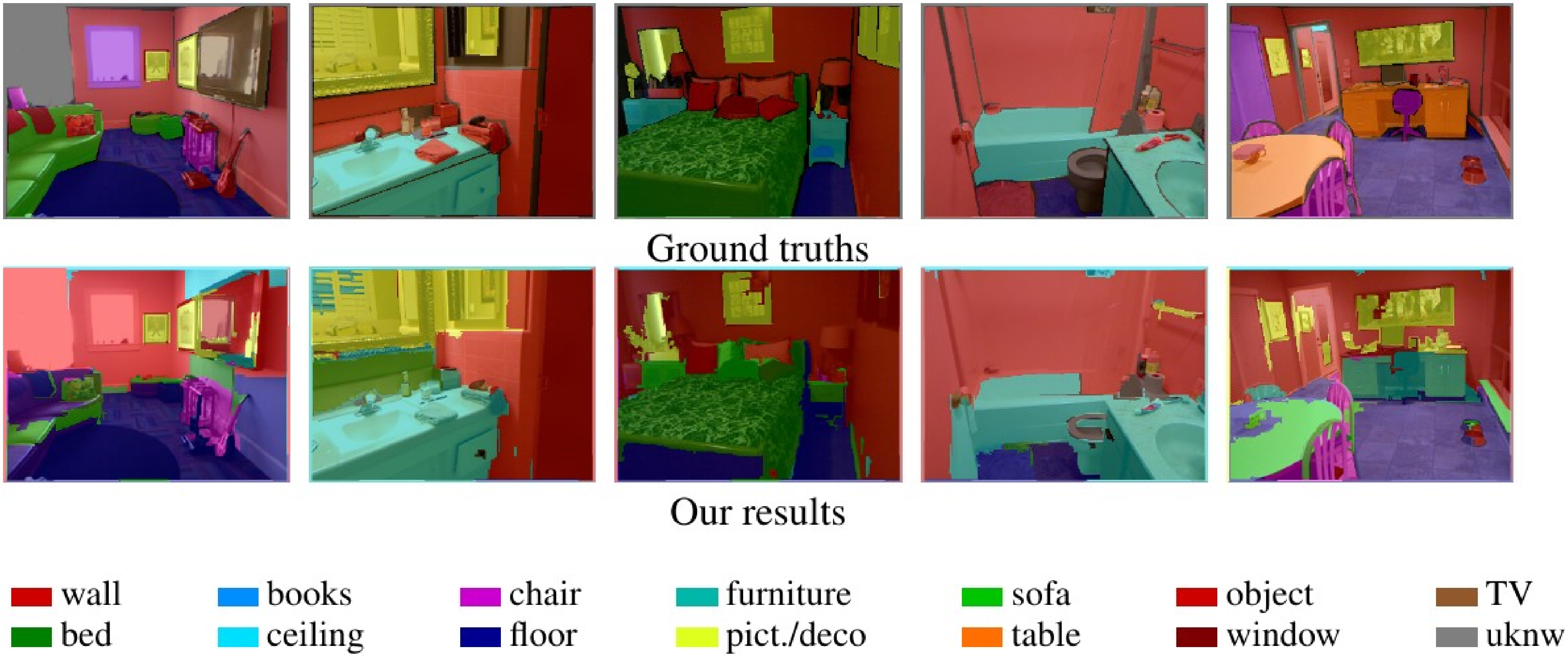
[C. Cadena, J. Kosecka "Semantic Parsing for Priming Object Detection in RGB-D Scenes" Semantic Perception Mapping and Exploration (SPME), Karlsruhe 2013]

Architecture for indoor RGB-D Semantic Segmentation

- Similar to outdoors semantic segmentation method
  - Convnet with 4 input channels
  - Vote over superpixels

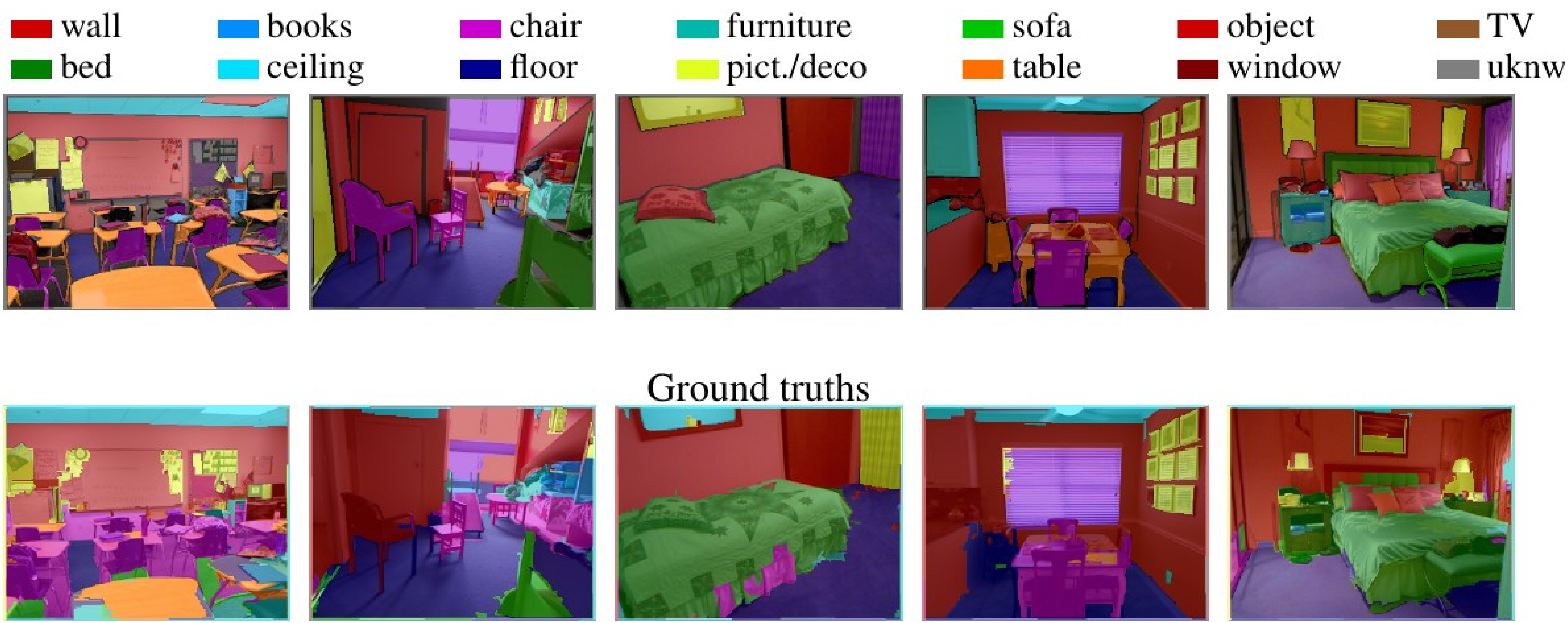


Scene Parsing/Labeling on RGB+Depth Images



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

Scene Parsing/Labeling on RGB+Depth Images



Ground truths

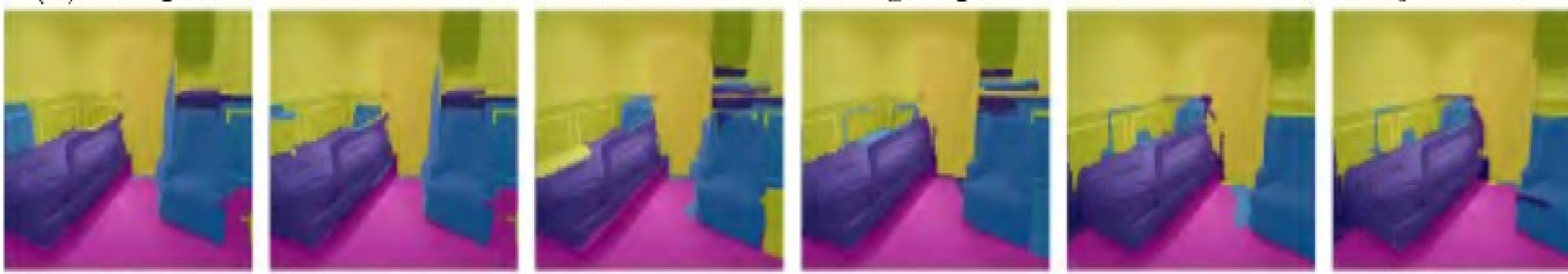
Our results

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

Labeling Videos

- Temporal consistency

(a) Output of the Multiscale convnet trained using depth information - frame by frame



(b) Results smoothed temporally using Couprie et al. (2013a)

[Couprie, Farabet, Najman, LeCun ICLR 2013]
[Couprie, Farabet, Najman, LeCun ICIP 2013]
[Couprie, Farabet, Najman, LeCun submitted to JMLR]

Semantic Segmentation on RGB+D Images and Videos

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

Building a ConvNet Model: Example in Torch7

```
model = nn.Sequential()
-- stage 1 : filter bank -> squashing -> L2 pooling -> normalization
model:add(nn.SpatialConvolutionMM(nfeats, nstates[1], filtsiz, filtsiz))
model:add(nn.Tanh())
model:add(nn.SpatialLPPooling(nstates[1],2,poolsiz,poolsiz,poolsiz,poolsiz))
model:add(nn.SpatialSubtractiveNormalization(nstates[1], normkernel))
-- stage 2 : filter bank -> squashing -> L2 pooling -> normalization
model:add(nn.SpatialConvolutionMM(nstates[1],nstates[2],filtsiz,filtsiz))
model:add(nn.Tanh())
model:add(nn.SpatialLPPooling(nstates[2],2,poolsiz,poolsiz,poolsiz,poolsiz))
model:add(nn.SpatialSubtractiveNormalization(nstates[2], normkernel))
-- stage 3 : 2 fully-connected layers
model:add(nn.Reshape(nstates[2]*filtsize*filtsize))
model:add(nn.Linear(nstates[2]*filtsize*filtsize, nstates[3]))
model:add(nn.Tanh())
model:add(nn.Linear(nstates[3], noutputs))
```

– http://www.torch.ch (Torch7: Lua-based dev environment for ML, CV....)

– http://code.cogbits.com/wiki/doku.php (Torch7 tutorials/demos by C. Farabet)

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

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)