Supervised and Unsupervised Methods for Learning Invariant Feature Hierarchies

Yann LeCun

The Courant Institute of Mathematical Sciences

New York University

Collaborators:

Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, Sumit Chopra

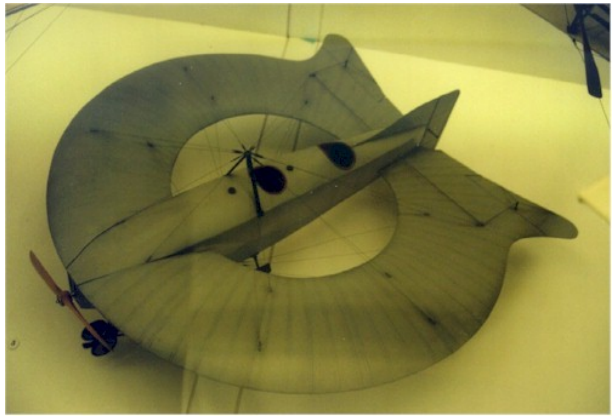
See: [LeCun et al. 2006]: "A Tutorial on Energy-Based Learning"

[Ranzato et al. AI-Stats 2007], [Ranzato et al. NIPS 2006]

http://yann.lecun.com/exdb/publis/

Challenges of Computer Vision (and Visual Neuroscience)

- **How do we learn "invariant representations"?**
  - From the image of an airplane, how do we extract a representation that is invariant to pose, illumination, background, clutter, object instance....
  - How can a human (or a machine) learn those representations by just looking at the world?
- **How can we learn visual categories from just a few examples?**
  - I don't need to see many airplanes before I can recognize every airplane (even really weird ones)


Challenges of Visual Neuroscience (and Computer Vision)

- **The recognition of everyday objects is a very fast process.**
  - Experiments by Simon Thorpe and others have shown that the recognition of common object is essentially "feed forward."
  - Not all of vision is feed forward (what would all those feed-back connection be there for?).
- **How much of the visual system is the result of learning?**
  - How much prior structure must be built into the visual system to enable it to learn to see?
  - Are V1/V2/V4 neurons learned or hard-wired?
- **If the visual system is learned, what is the learning algorithm?**
  - What learning algorithm can train neural network as "deep" as the visual system (10 layers?).
  - Much of the learning is unsupervised.
- **Can we train an artificial vision system from end to end?**

Questions?

- **Is there a magic bullet for visual learning?**
  - Is there a general principle, or should we just resort to clever engineering (or to a large collection of tricks)?
  - Is there a universal learning algorithm/architecture which, given a small amount of appropriate prior structure, can produce an intelligent vision system?
  - Or do we need to accumulate a large repertoire of "modules" to solve every specific problem an intelligent vision system must solve. How would we assemble those modules?
- **How far can we get by training a vision system end to end**
  - Let us train a complete vision system from raw pixels to object categories, or to robot actions.

Two Big Problems in Machine Learning and Computer Vision

- **1. The "Intractable Partition Function Problem"**
  - Give high probability (or low energy) to good answers
  - Give low probability (or high energy) to bad answers
  - There are too many bad answers!
  - The normalization constant of probabilistic models is a sum over too many terms.
- **2. The "Deep Learning Problem"**
  - Training "Deep Belief Networks" is a necessary step towards solving the invariance problem in vision (and perception in general).
  - How do we train deep architectures with lots of non-linear stages?
- **This talks has four parts:**
  - supervised methods for deep visual learning: convolutional nets
  - Energy-Based learning: circumventing the intractable partition function problem.
  - Unsupervised learning for energy-based models
  - "Deep belief networks": stacking unsupervised modules to learn feature hierarchies

Part 1: Deep Supervised Learning for Vision: The Convolutional Network Architecture

- **Convolutional Networks:**
  - [LeCun et al., Neural Computation, 1988]
  - [LeCun et al., Proc IEEE 1998]
- **Face Detection and pose estimation with convolutional networks:**
  - [Vaillant, Monrocq, LeCun, IEE Proc Vision, Image and Signal Processing, 1994]
  - [Osadchy, Miller, LeCun, JMLR vol 8, May 2007]
- **Category-level object recognition with invariance to pose and lighting**
  - [LeCun, Huang, Bottou, CVPR 2004]
  - [Huang, LeCun, CVPR 2005]
- **autonomous robot driving**
  - [LeCun et al. NIPS 2005]

The Traditional Architecture for Recognition

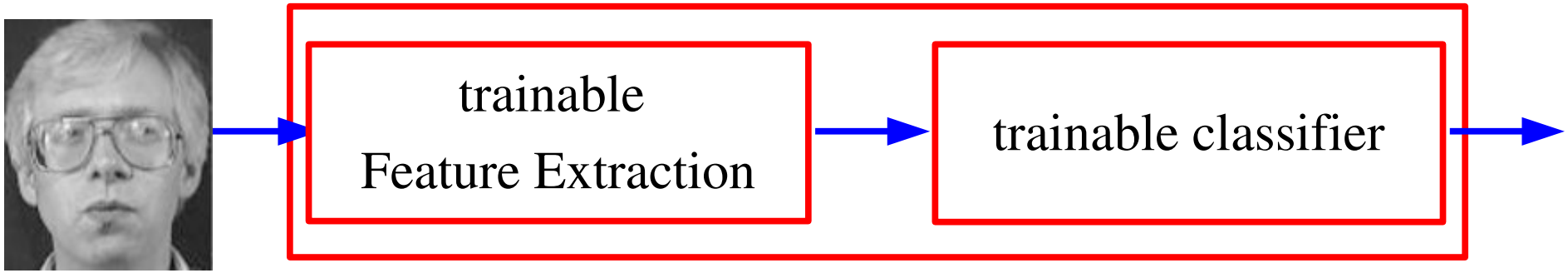
Pre-processing / Feature Extraction

Trainable Classifier

this part is mostly hand-crafted

- The raw input is pre-processed through a hand-crafted feature extractor
- The trainable classifier is often generic (task independent)

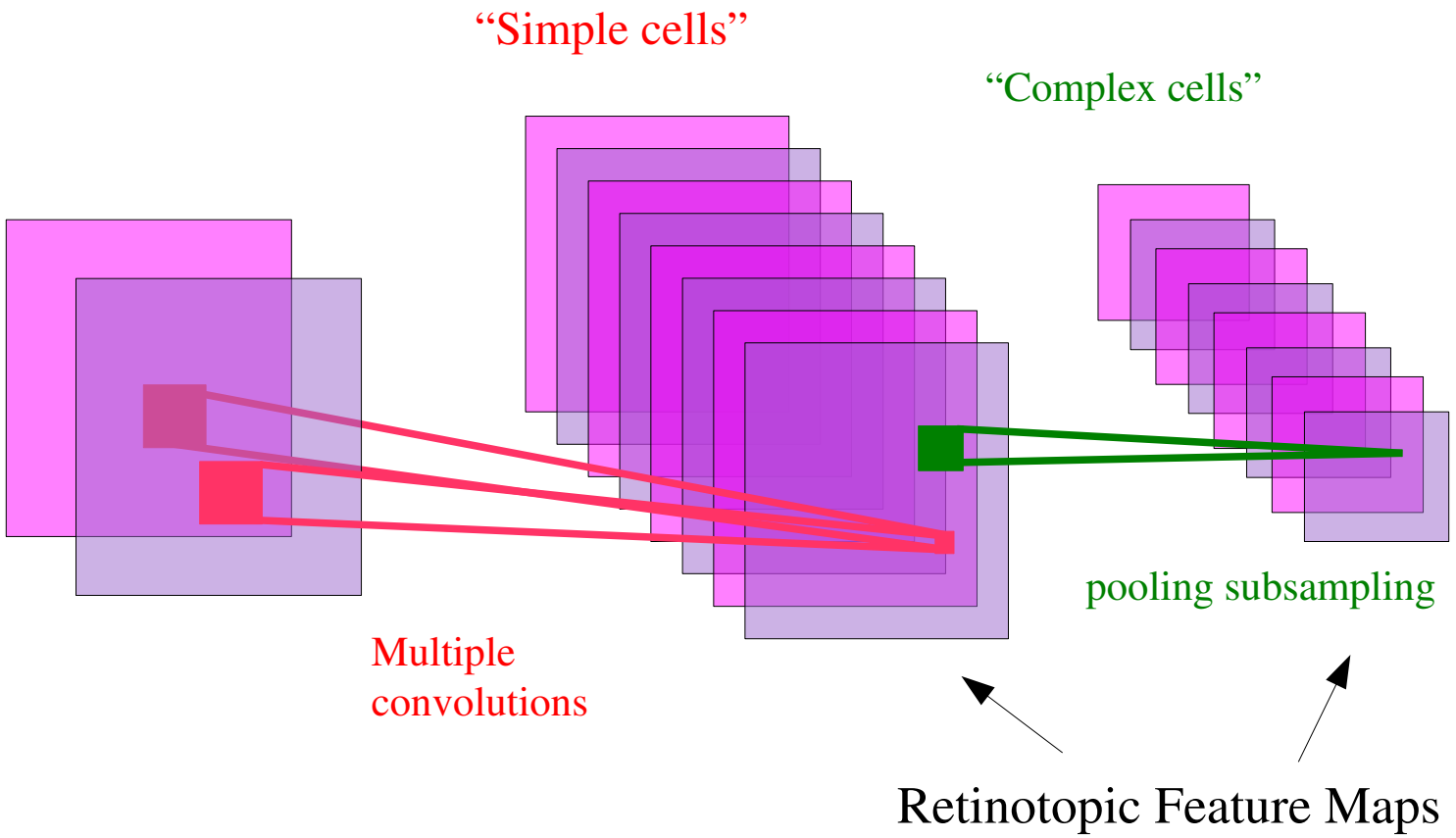
End-to-End Learning



- The entire system is integrated and trainable "end-to-end".
- In some of the models presented here, there will be no discernible difference between the feature extractor and the classifier.
- We can embed general prior knowledge about images into the architecture of the system.

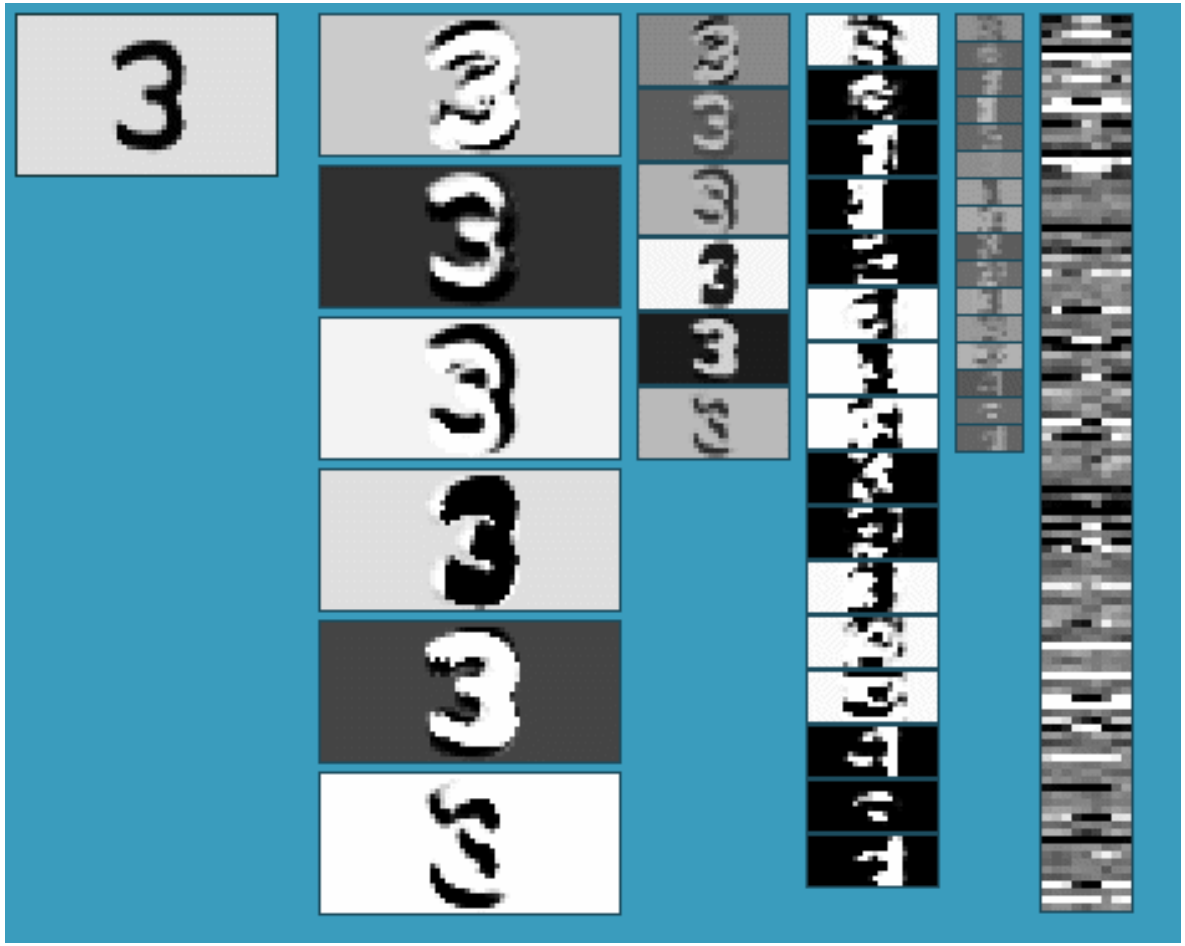
An Old Idea for Local Shift Invariance

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

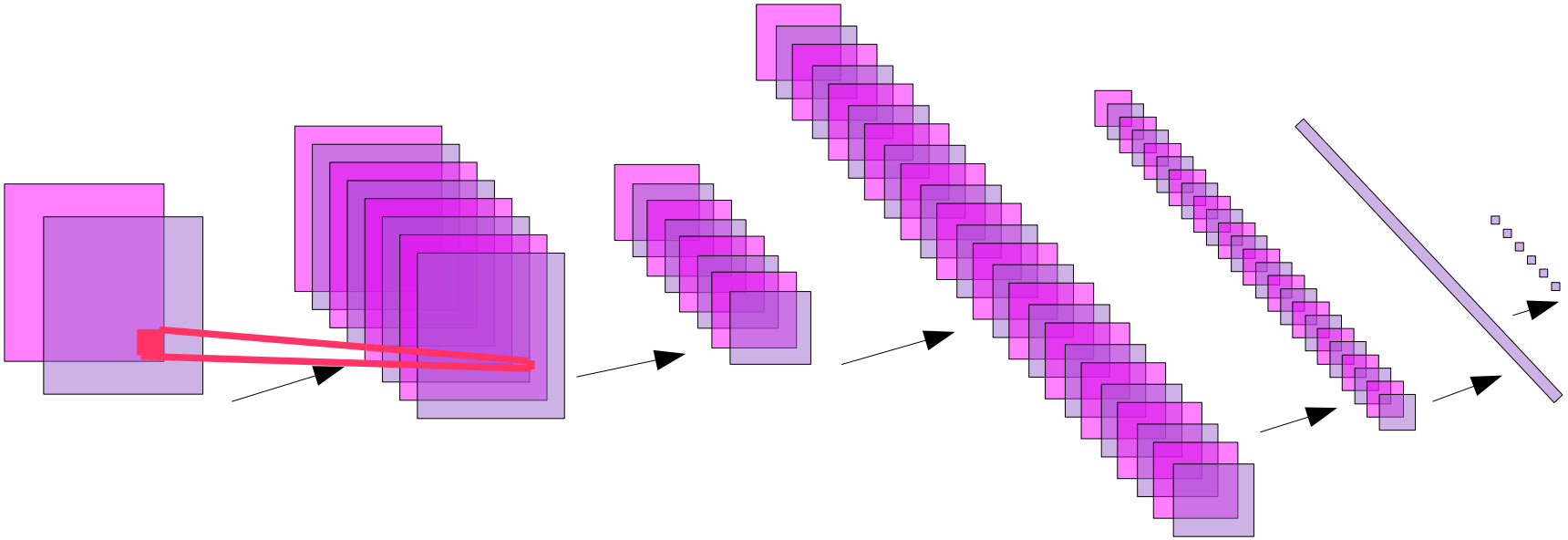


The Multistage Hubel-Wiesel Architecture

- **Building a complete artificial vision system:**
  - Stack multiple stages of simple cells / complex cells layers
  - Higher stages compute more global, more invariant features
  - Stick a classification layer on top
  - [Fukushima 1971-1982]
    - neocognitron
  - [LeCun 1988-2007]
    - convolutional net
  - [Poggio 2002-2006]
    - HMAX
  - [Ullman 2002-2006]
    - fragment hierarchy
  - [Lowe 2006]
    - HMAX
- **QUESTION: How do we find (or learn) the filters?**

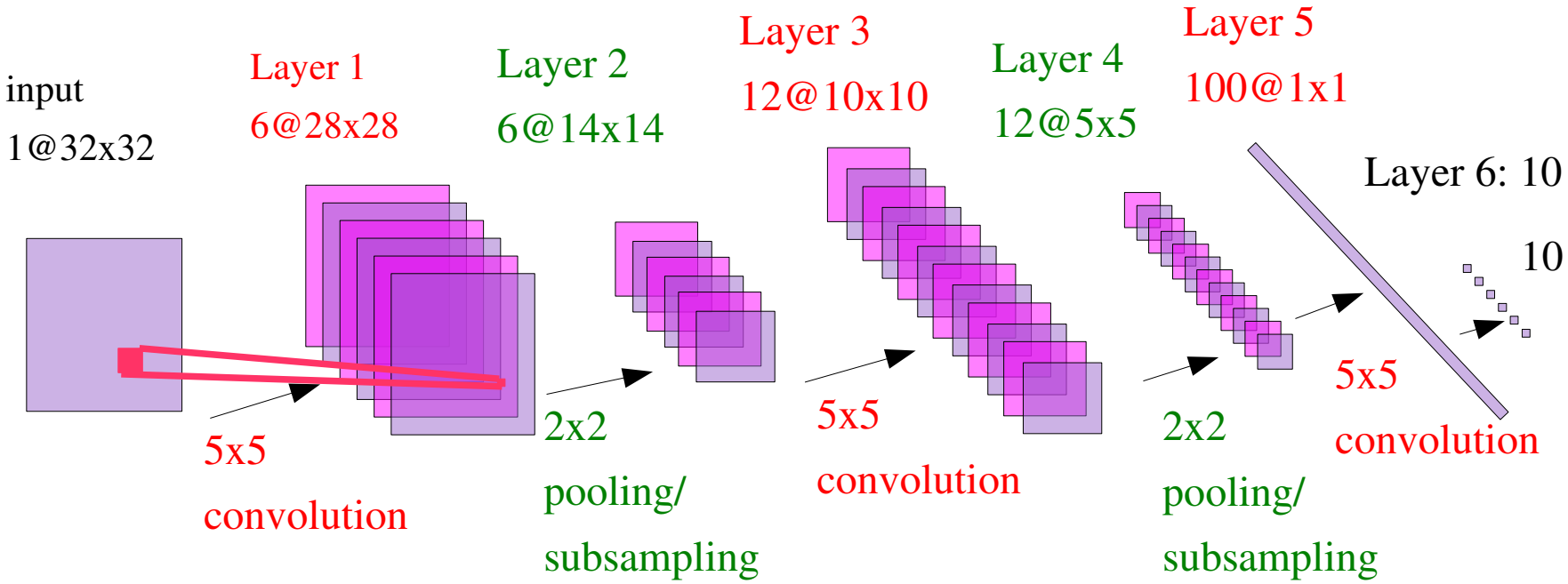


Getting Inspiration from Biology: Convolutional Network



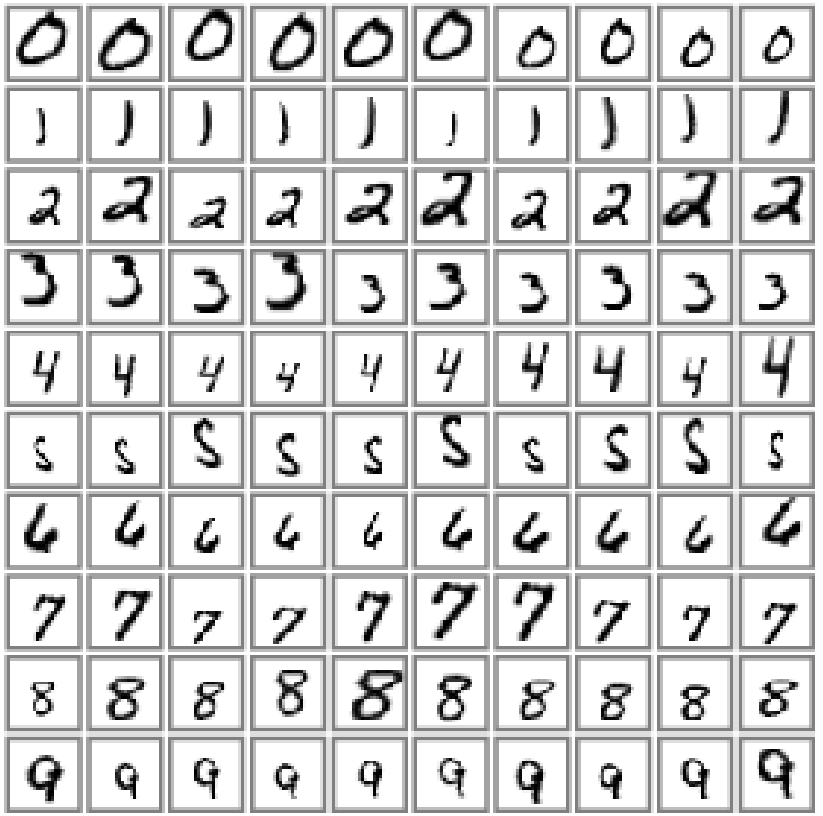
- **Hierarchical/multilayer:** features get progressively more global, invariant, and numerous
- **dense features:** features detectors applied everywhere (no interest point)
- **broadly tuned (possibly invariant) features:** sigmoid units are on half the time.
- **Global discriminative training:** The whole system is trained "end-to-end" with a gradient-based method to minimize a global loss function
- **Integrates segmentation, feature extraction, and invariant classification in one fell swoop.**

Convolutional Net Architecture



- **Convolutional net for handwriting recognition** (400,000 synapses)
- Convolutional layers (simple cells): all units in a feature plane share the same weights
- Pooling/subsampling layers (complex cells): for invariance to small distortions.
- **Supervised gradient-descent learning using back-propagation**
- **The entire network is trained end-to-end. All the layers are trained simultaneously**.

MNIST Handwritten Digit Dataset



- Handwritten Digit Dataset MNIST: 60,000 training samples, 10,000 test samples

Results on MNIST Handwritten Digits

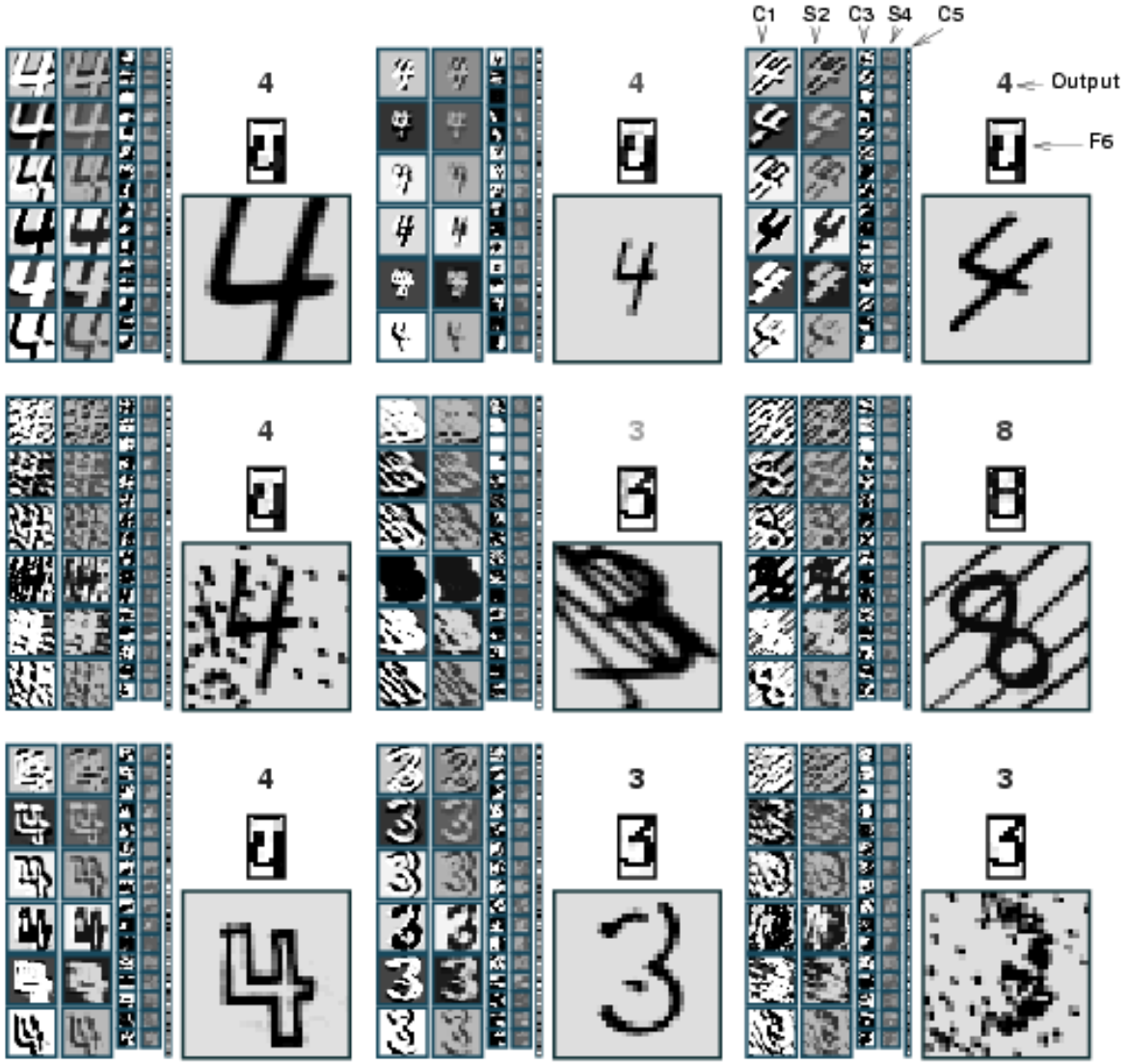
CLASSIFIER	DEFORMATION	PREPROCESSING	ERROR (%)	Reference
linear classifier (1-layer NN)		none	12.00	LeCun et al. 1998
linear classifier (1-layer NN)		deskewing	8.40	LeCun et al. 1998
pairwise linear classifier		deskewing	7.60	LeCun et al. 1998
K-nearest-neighbors, (L2)		none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, (L2)		deskewing	2.40	LeCun et al. 1998
K-nearest-neighbors, (L2)		deskew, clean, blur	1.80	Kenneth Wilder, U. Chicago
K-NN L3, 2 pixel jitter		deskew, clean, blur	1.22	Kenneth Wilder, U. Chicago
K-NN, shape context matching		shape context feature	0.63	Belongie et al. IEEE PAMI 2002
40 PCA + quadratic classifier		none	3.30	LeCun et al. 1998
1000 RBF + linear classifier		none	3.60	LeCun et al. 1998
K-NN, Tangent Distance		subsamp 16x16 pixels	1.10	LeCun et al. 1998
SVM, Gaussian Kernel		none	1.40	
SVM deg 4 polynomial		deskewing	1.10	LeCun et al. 1998
Reduced Set SVM deg 5 poly		deskewing	1.00	LeCun et al. 1998
Virtual SVM deg-9 poly	Affine	none	0.80	LeCun et al. 1998
V-SVM, 2-pixel jittered		none	0.68	DeCoste and Scholkopf, MLJ 2002
V-SVM, 2-pixel jittered		deskewing	0.56	DeCoste and Scholkopf, MLJ 2002
2-layer NN, 300 HU, MSE		none	4.70	LeCun et al. 1998
2-layer NN, 300 HU, MSE,	Affine	none	3.60	LeCun et al. 1998
2-layer NN, 300 HU		deskewing	1.60	LeCun et al. 1998
3-layer NN, 500+150 HU		none	2.95	LeCun et al. 1998
3-layer NN, 500+150 HU	Affine	none	2.45	LeCun et al. 1998
3-layer NN, 500+300 HU, CE, reg		none	1.53	Hinton, unpublished, 2005
2-layer NN, 800 HU, CE		none	1.60	Simard et al., ICDAR 2003
2-layer NN, 800 HU, CE	Affine	none	1.10	Simard et al., ICDAR 2003
2-layer NN, 800 HU, MSE	Elastic	none	0.90	Simard et al., ICDAR 2003
2-layer NN, 800 HU, CE	Elastic	none	0.70	Simard et al., ICDAR 2003
Convolutional net LeNet-1		subsamp 16x16 pixels	1.70	LeCun et al. 1998
Convolutional net LeNet-4		none	1.10	LeCun et al. 1998
Convolutional net LeNet-5,		none	0.95	LeCun et al. 1998
Conv. net LeNet-5,	Affine	none	0.80	LeCun et al. 1998
Boosted LeNet-4	Affine	none	0.70	LeCun et al. 1998
Conv. net, CE	Affine	none	0.60	Simard et al., ICDAR 2003
Comv net, CE	Elastic	none	0.40	Simard et al., ICDAR 2003

Some Results on MNIST (from raw images: no preprocessing)

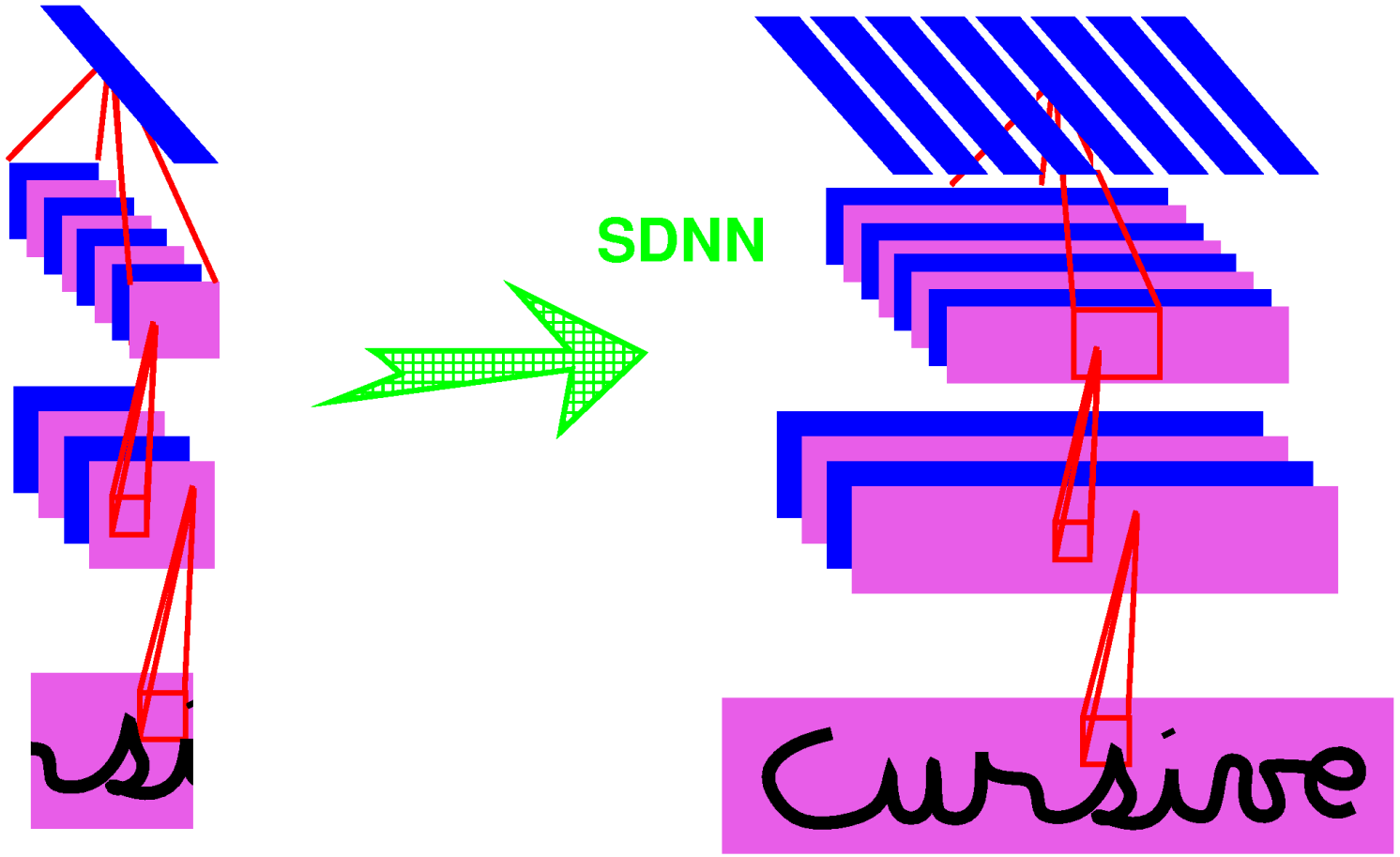
CLASSIFIER	DEFORMATION	ERROR	Reference
Knowledge-free methods (a fixed permutation of the pixels would make no difference)			
2-layer NN, 800 HU, CE		1.60	Simard et al., ICDAR 2003
3-layer NN, 500+300 HU, CE, reg		1.53	Hinton, in press, 2005
SVM, Gaussian Kernel		1.40	Cortes 92 + Many others
Convolutional nets			
Convolutional net LeNet-5,		0.80	Ranzato et al. NIPS 2006
Convolutional net LeNet-6,		0.70	Ranzato et al. NIPS 2006
Training set augmented with Affine Distortions			
2-layer NN, 800 HU, CE	Affine	1.10	Simard et al., ICDAR 2003
Virtual SVM deg-9 poly	Affine	0.80	Scholkopf
Convolutional net, CE	Affine	0.60	Simard et al., ICDAR 2003
Training et augmented with Elastic Distortions			
2-layer NN, 800 HU, CE	Elastic	0.70	Simard et al., ICDAR 2003
Convolutional net, CE	Elastic	0.40	Simard et al., ICDAR 2003

Note: some groups have obtained good results with various amounts of preprocessing such as deskewing (e.g. 0.56% using an SVM with smart kernels [deCoste and Schoelkopf]) hand-designed feature representations (e.g. 0.63% with "shape context" and nearest neighbor [Belongie]

Invariance and Robustness to Noise

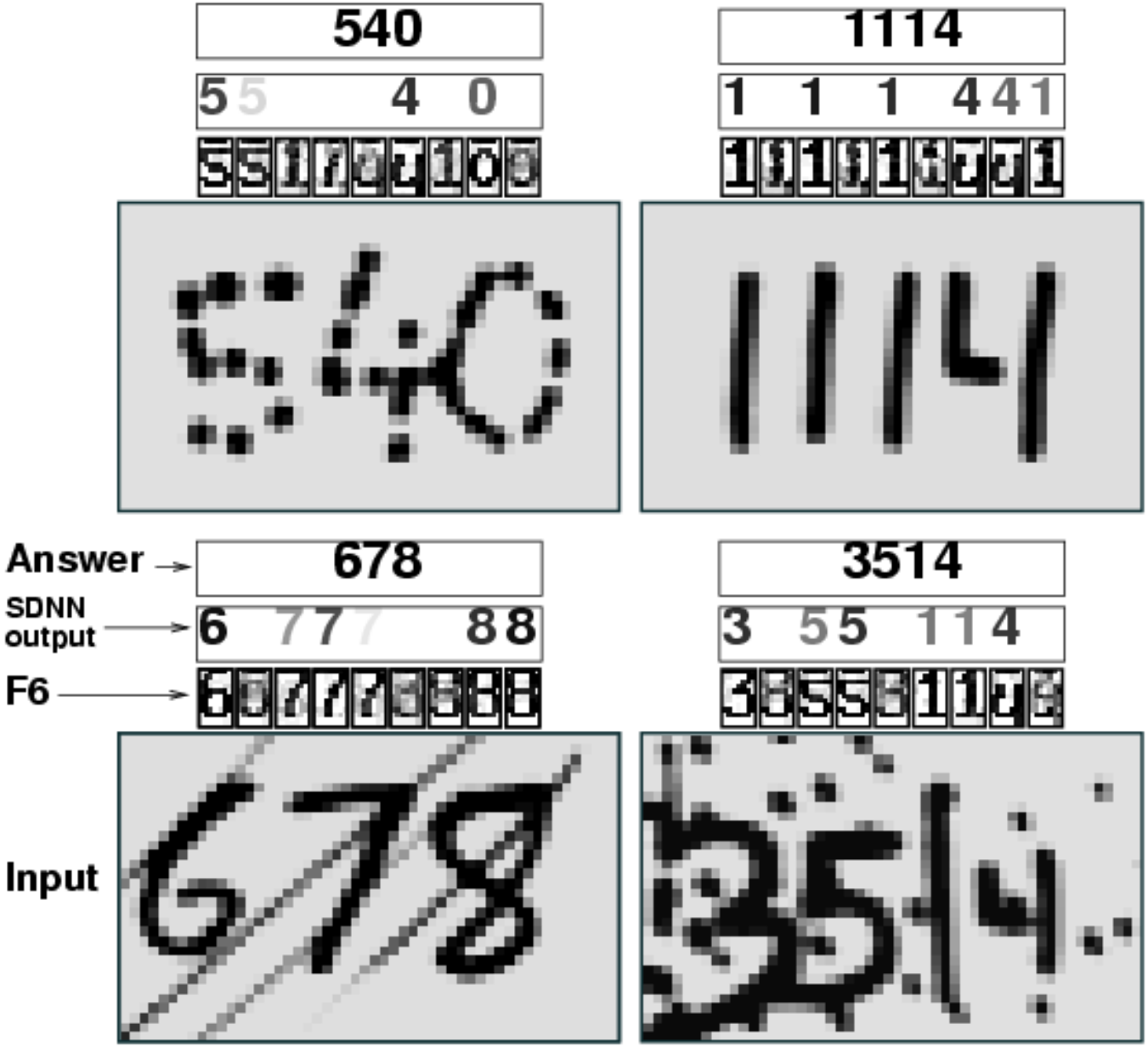


Recognizing Multiple Characters with Replicated Nets



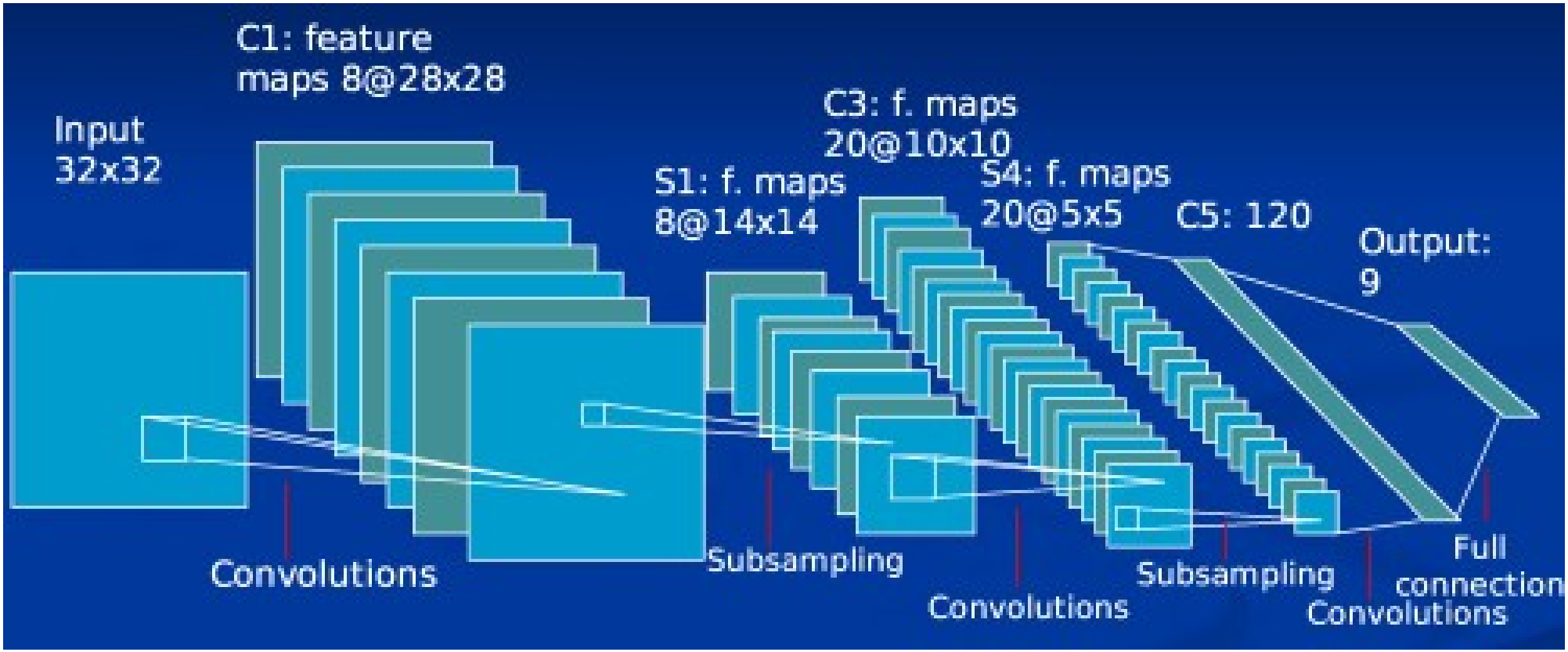
Recognizing Multiple Characters with Replicated Nets

Handwriting Recognition



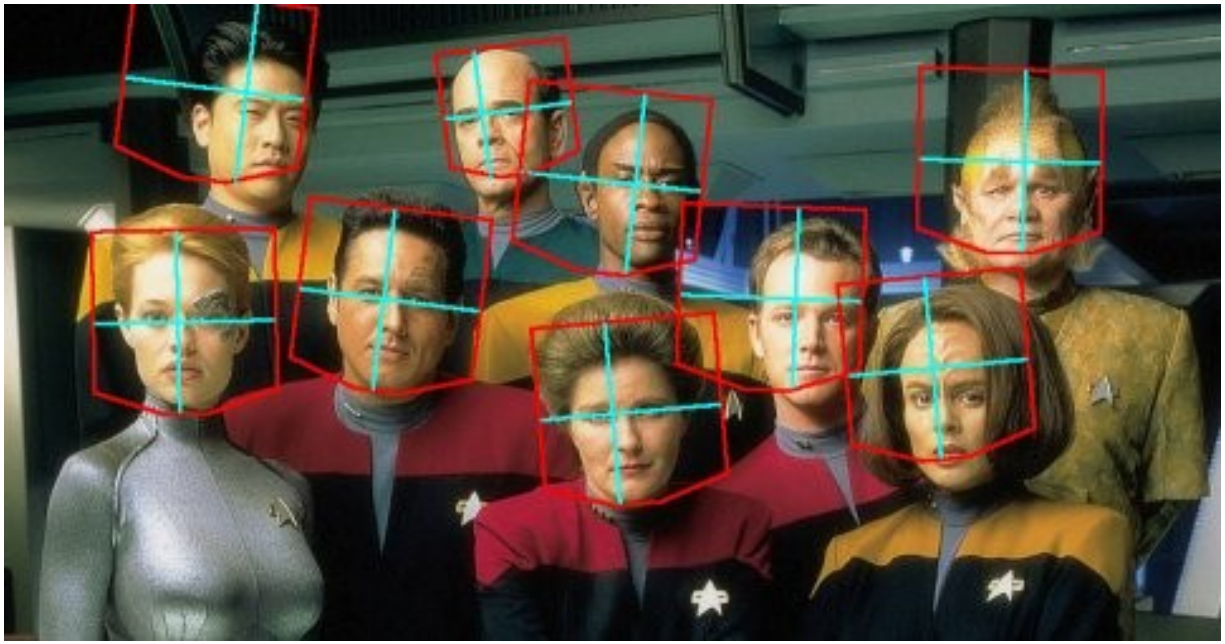
Face Detection and Pose Estimation with Convolutional Nets

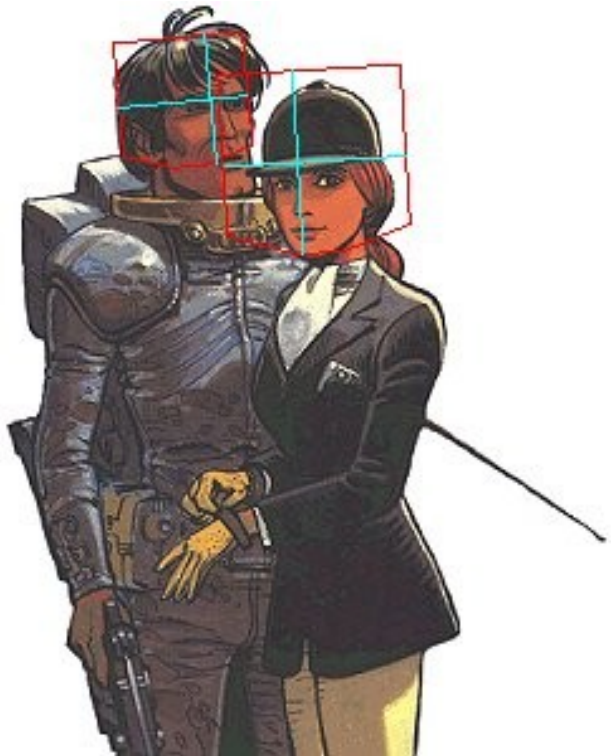
- **Training:** 52,850, 32x32 grey-level images of faces, 52,850 non-faces.
- **Each sample:** used 5 times with random variation in scale, in-plane rotation, brightness and contrast.
- **2nd phase:** half of the initial negative set was replaced by false positives of the initial version of the detector .



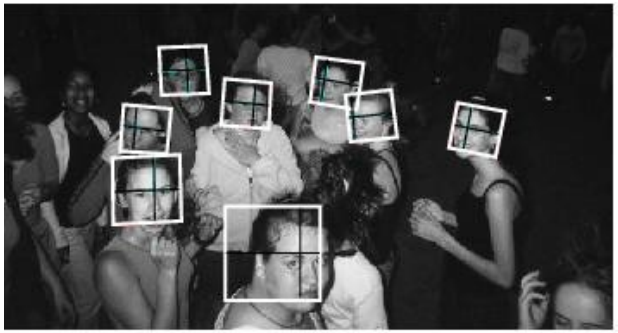
Face Detection: Results

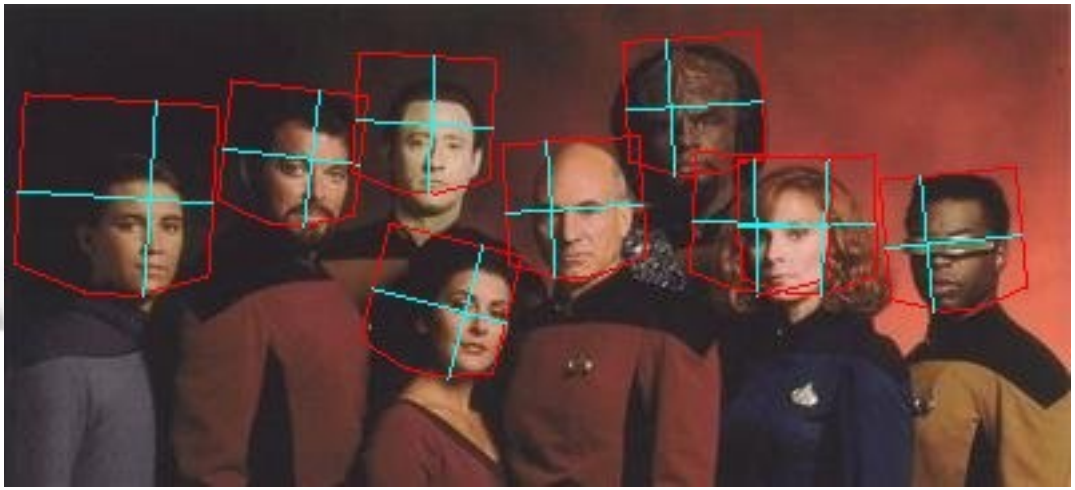
Data Set->	TILTED		PROFILE		MIT+CMU	
False positives per image->	4.42	26.9	0.47	3.36	0.5	1.28
Our Detector	90%	97%	67%	83%	83%	88%
Jones & Viola (tilted)	90%	95%	x		x	
Jones & Viola (profile)	x		70%	83%	x	

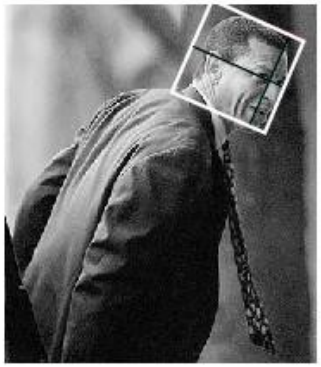


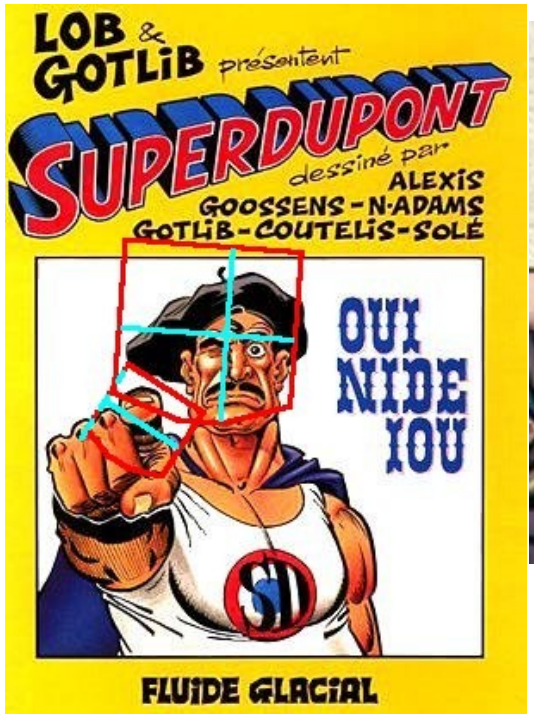


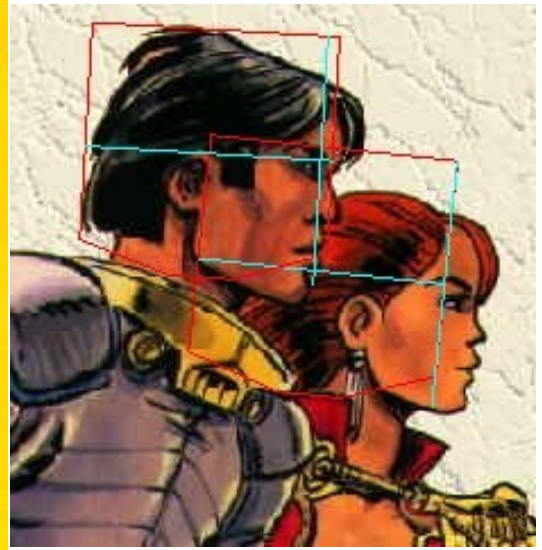
Face Detection and Pose Estimation: Results





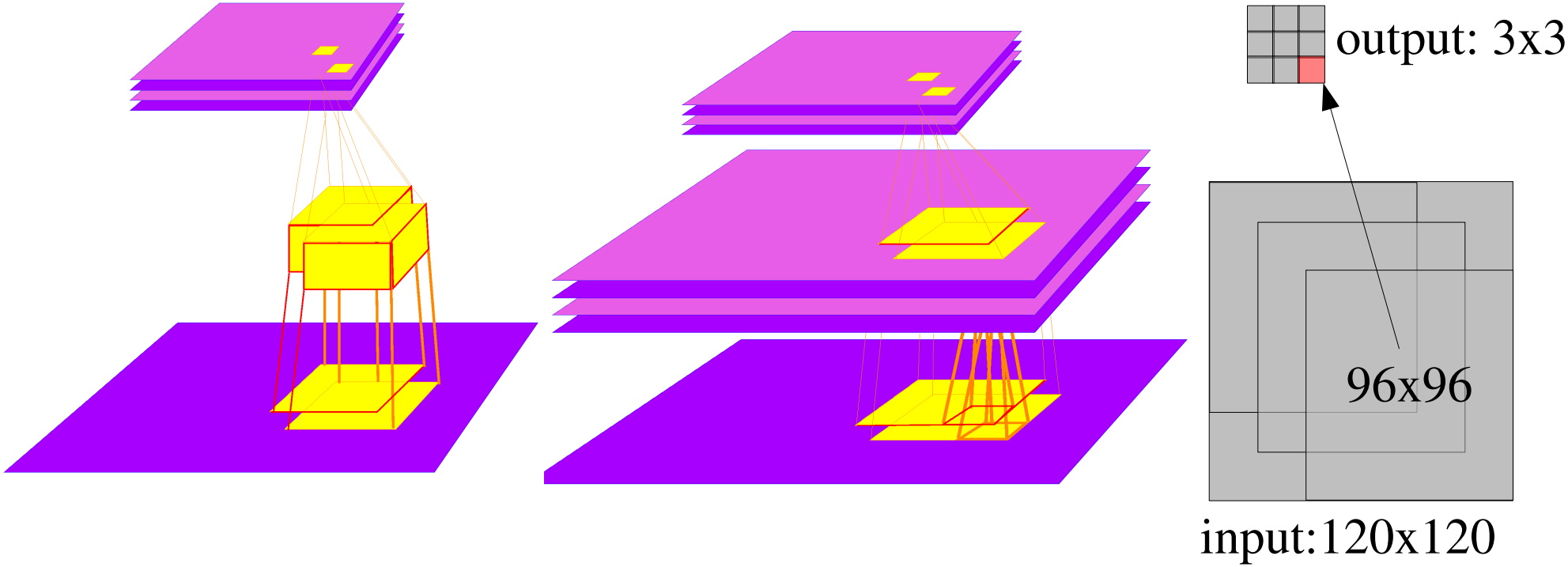






Face Detection with a Convolutional Net

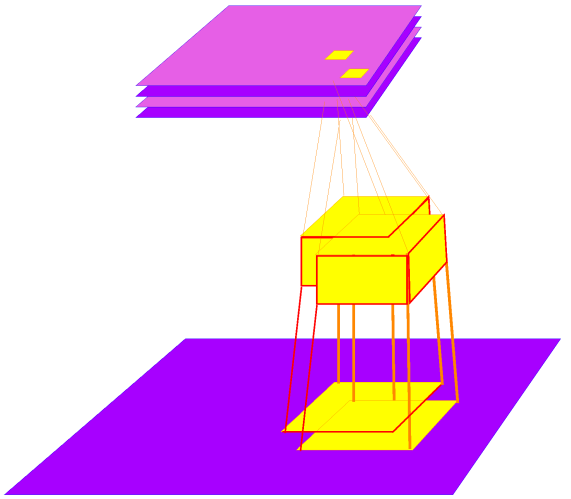
Applying a ConvNet on Sliding Windows is Very Cheap!

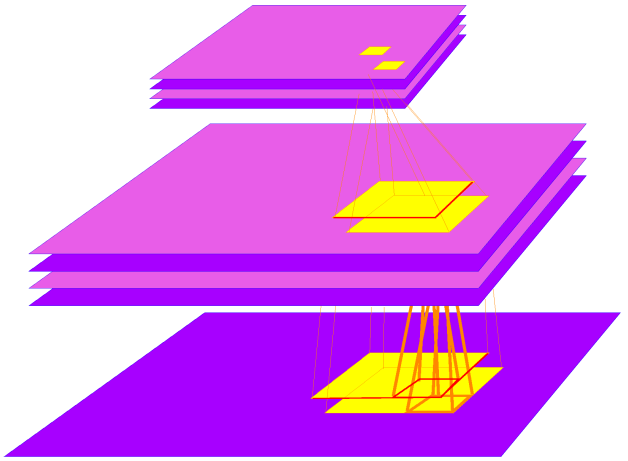


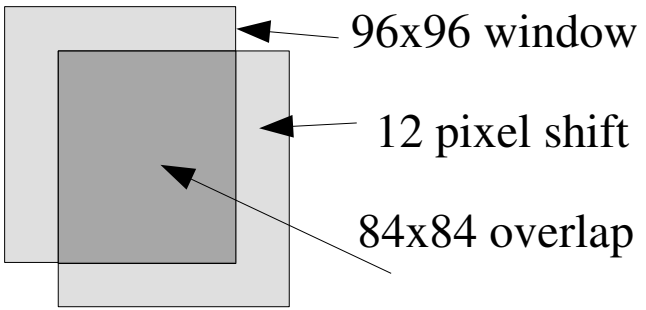
- Traditional Detectors/Classifiers must be applied to every location on a large input image, at multiple scales.
- Convolutional nets can replicated over large images very cheaply.
- The network is applied to multiple scales spaced by 1.5.

Building a Detector/Recognizer: Replicated Convolutional Nets

- Computational cost for replicated convolutional net:
  - 96x96 -> 4.6 million multiply-accumulate operations
  - 120x120 -> 8.3 million multiply-accumulate operations
  - 240x240 -> 47.5 million multiply-accumulate operations
  - 480x480 -> 232 million multiply-accumulate operations
- 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 operations
  - 480x480 -> 5,083 million multiply-accumulate operations



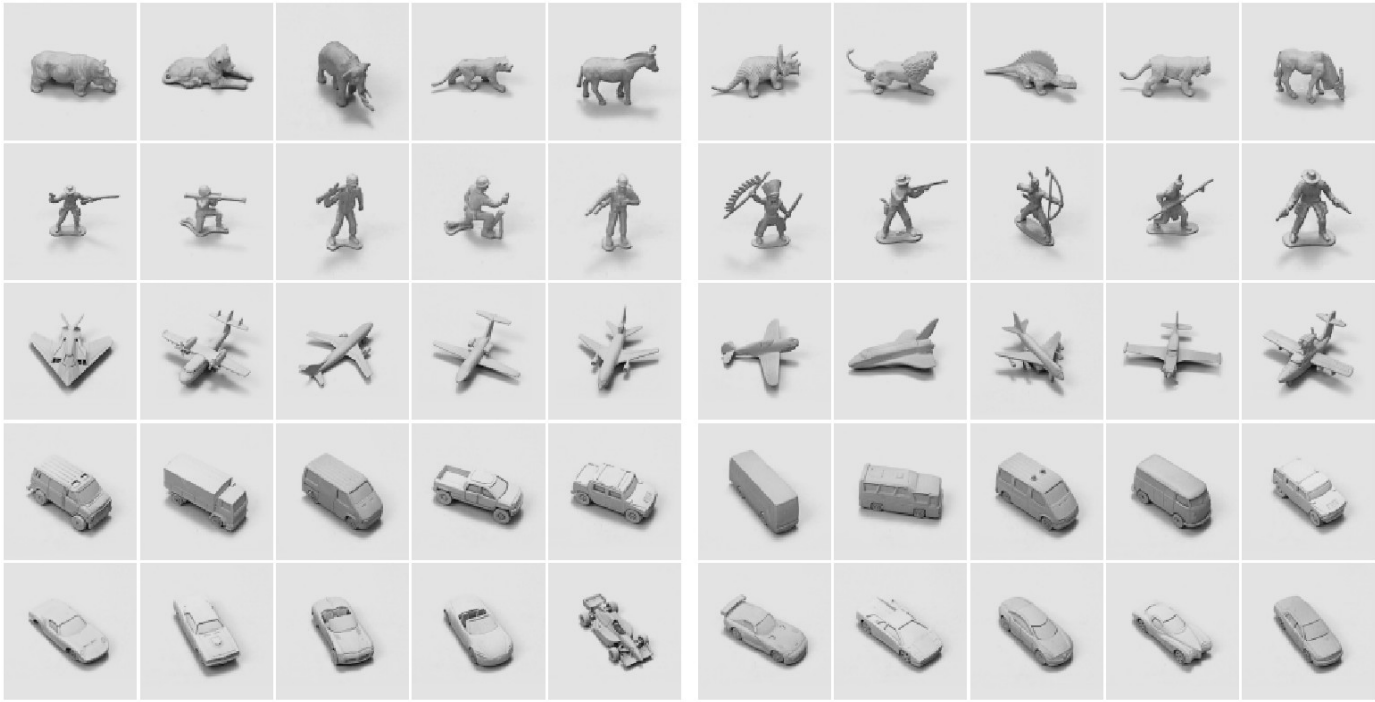




Generic Object Detection and Recognition with Invariance to Pose and Illumination

- **50** toys belonging to 5 categories: **animal, human figure, airplane, truck, car**
- **10** instance per category: 5 instances used for training, 5 instances for testing
- **Raw dataset: 972** stereo pair of each object instance. **48,600** image pairs total.

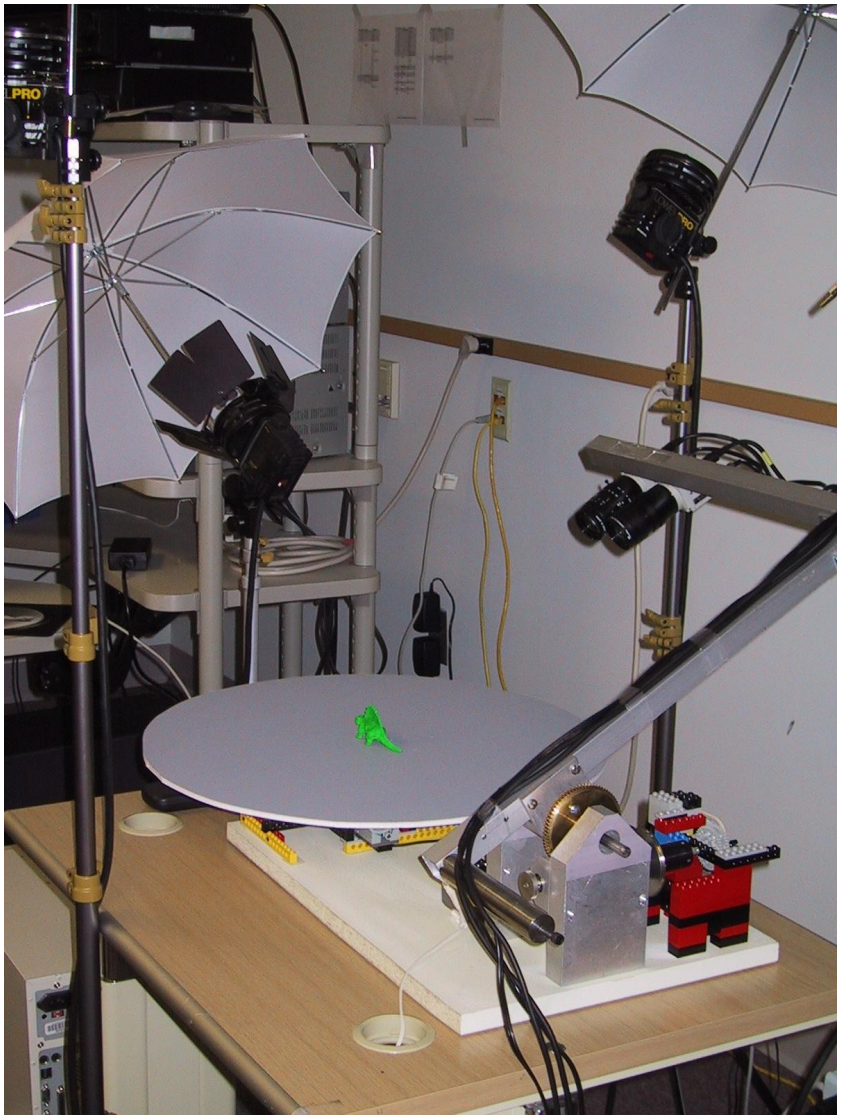
- **For each instance:**
- **18 azimuths**
  - 0 to 350 degrees every 20 degrees
- **9 elevations**
  - 30 to 70 degrees from horizontal every 5 degrees
- **6 illuminations**
  - on/off combinations of 4 lights
- **2 cameras (stereo)**
  - 7.5 cm apart
  - 40 cm from the object



Training instances **Test instances**

Data Collection, Sample Generation

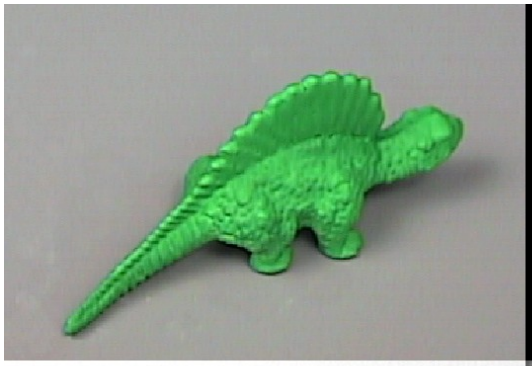
Image capture setup



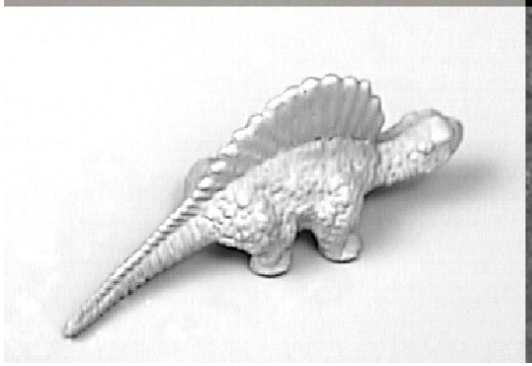
Objects are painted green so that:

- all features other than shape are removed
- objects can be segmented, transformed, and composited onto various backgrounds

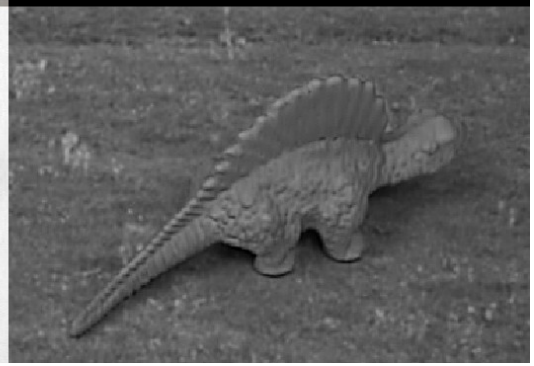
Original image



Object mask

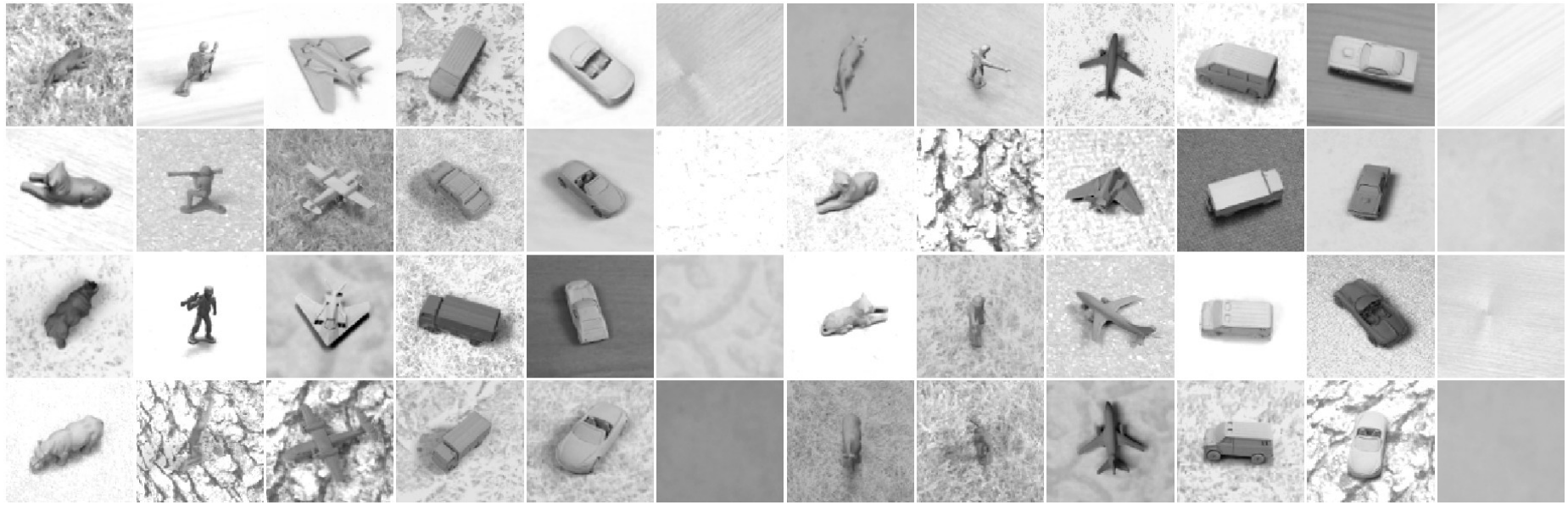


Shadow factor

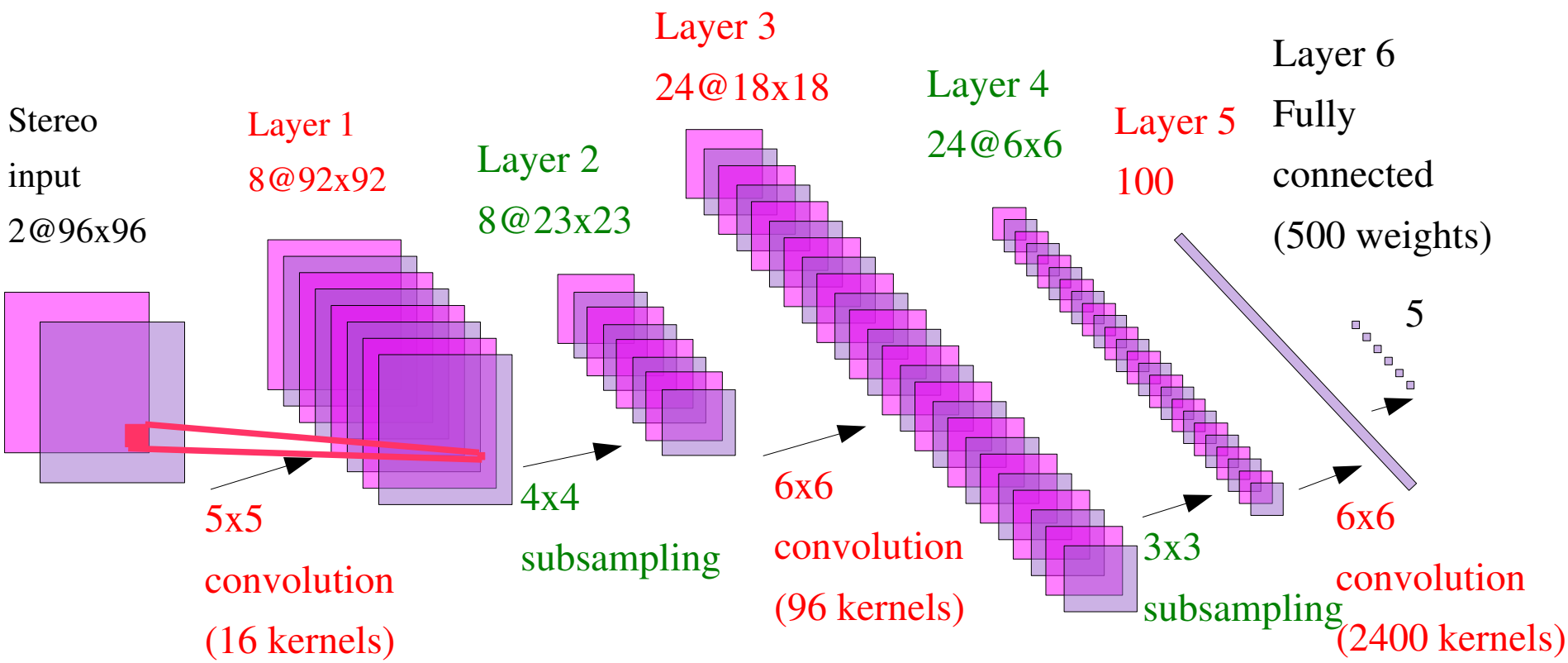


Composite image

Textured and Cluttered Datasets

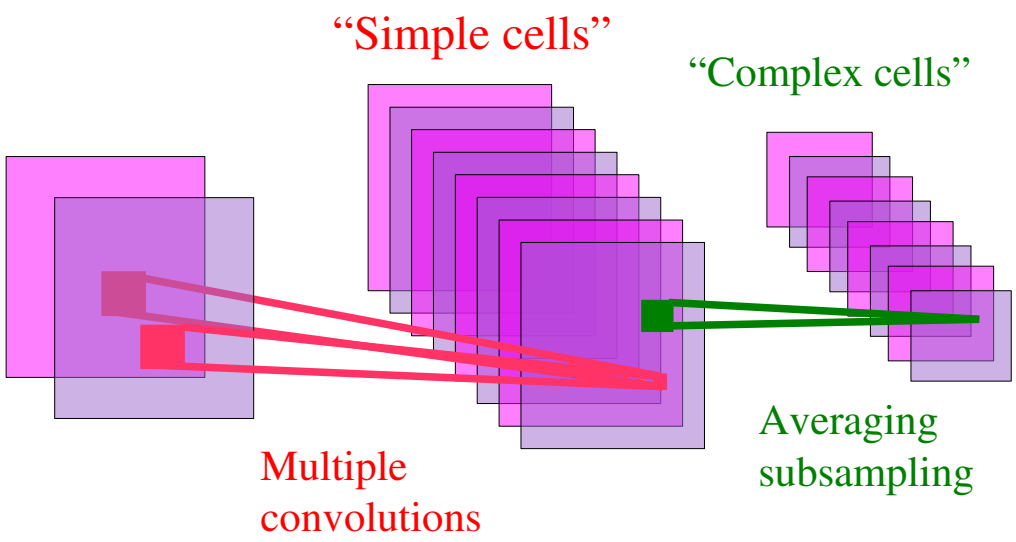


Convolutional Network

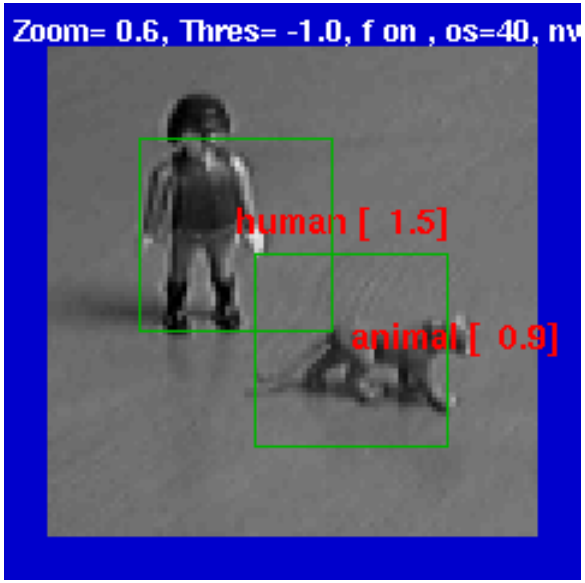


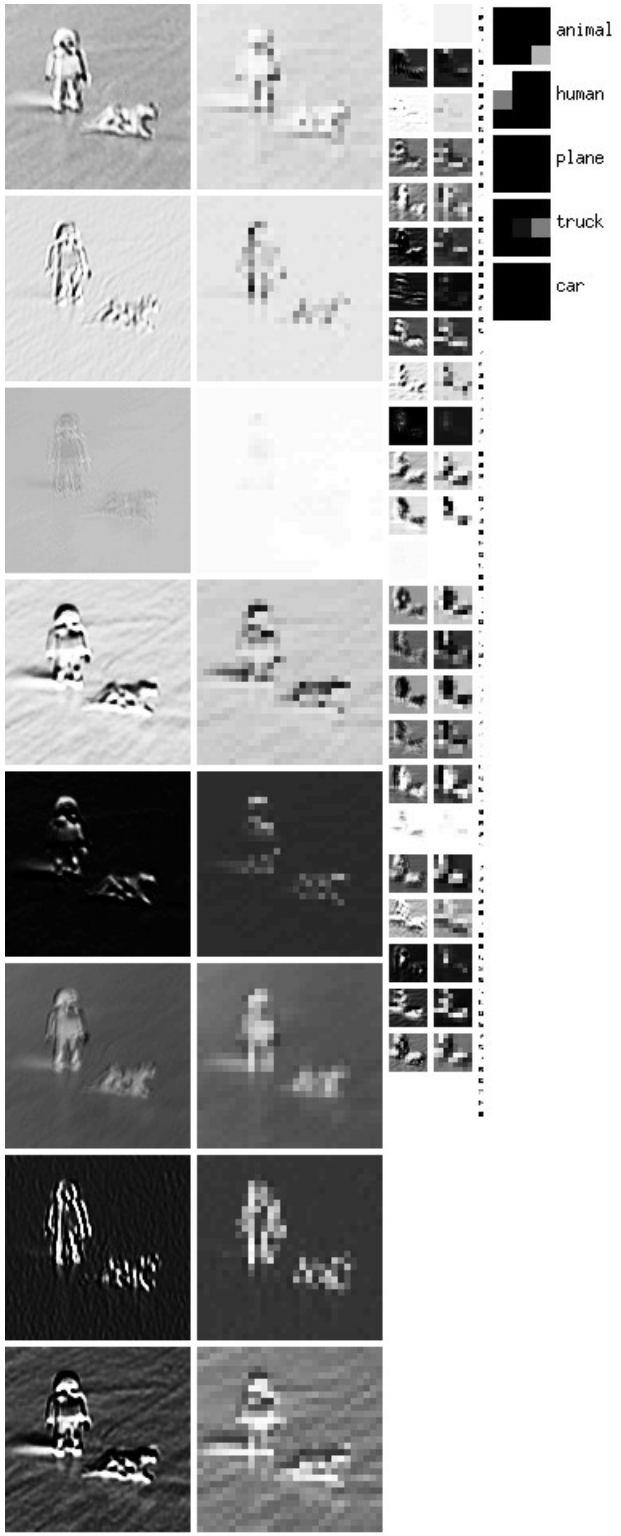
- **90,857 free parameters, 3,901,162 connections.**
- The architecture alternates convolutional layers (feature detectors) and subsampling layers (local feature pooling for invariance to small distortions).
- **The entire network is trained end-to-end** (all the layers are trained simultaneously).
- A gradient-based algorithm is used to minimize a supervised loss function.

Alternated Convolutions and Subsampling



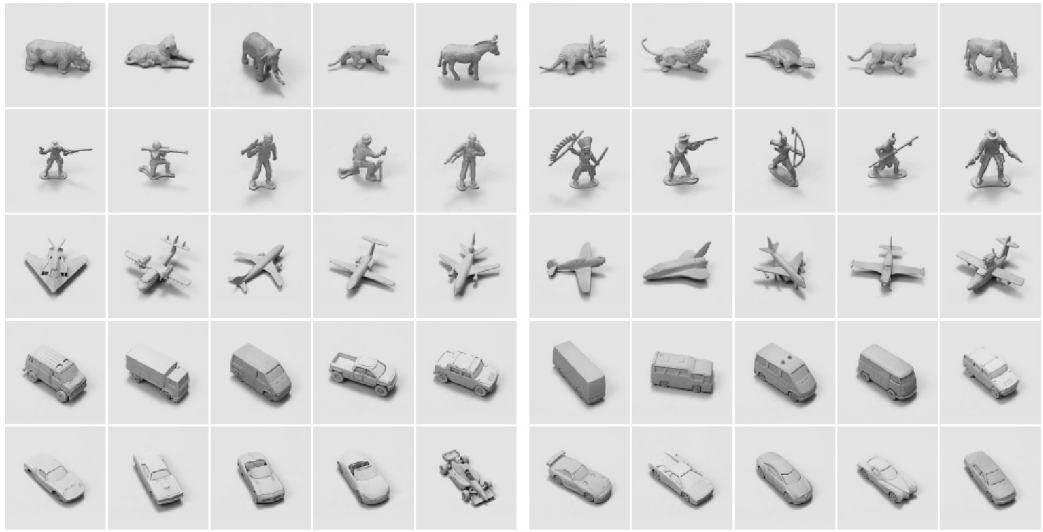
- Local features are extracted everywhere.
- averaging/subsampling layer builds robustness to variations in feature locations.
- Hubel/Wiesel'62, Fukushima'71, LeCun'89, Riesenhuber & Poggio'02, Ullman'02,....





Normalized-Uniform Set: Error Rates

- **Linear Classifier on raw stereo images:** 30.2% error.
- **K-Nearest-Neighbors on raw stereo images:** 18.4% error.
- **K-Nearest-Neighbors on PCA-95:** 16.6% error.
- **Pairwise SVM on 96x96 stereo images:** 11.6% error
- **Pairwise SVM on 95 Principal Components:** 13.3% error.
- **Convolutional Net on 96x96 stereo images:** 5.8% error.



Training instances Test instances

Normalized-Uniform Set: Learning Times

	SVM	Conv Net				SVM/Conv
test error	11.6%	10.4%	6.2%	5.8%	6.2%	5.9%
train time (min*GHz)	480	64	384	640	3,200	50+
test time per sample (sec*GHz)	0.95	0.03				0.04+
#SV	28%					28%
parameters	σ=2,000 C=40					dim=80 σ=5 C=0.01

SVM: using a parallel implementation by Graf, Durdanovic, and Cosatto (NEC Labs)

Chop off the last layer of the convolutional net and train an SVM on it

Jittered-Cluttered Dataset

- **Jittered-Cluttered Dataset:**
- **291,600** tereo pairs for training, **58,320** for testing
- Objects are jittered: position, scale, in-plane rotation, contrast, brightness, backgrounds, distractor objects,...
- Input dimension: 98x98x2 (approx 18,000)

Experiment 2: Jittered-Cluttered Dataset

- 291,600 training samples, 58,320 test samples
- SVM with Gaussian kernel — 43.3% error
- Convolutional Net with binocular input: — 7.8% error
- Convolutional Net + SVM on top: — 5.9% error
- Convolutional Net with monocular input: — 20.8% error
- Smaller mono net (DEMO): — 26.0% error
- Dataset available from http://www.cs.nyu.edu/~yann

Jittered-Cluttered Dataset

	SVM	Conv Net			SVM/Conv
test error	43.3%	16.38%	7.5%	7.2%	5.9%
train time (min*GHz)	10,944	420	2,100	5,880	330+
test time per sample (sec*GHz)	2.2	0.04			0.06+
#SV	5%				2%
parameters	σ=10^4 C=40				dim=100 σ=5 C=1

OUCH!

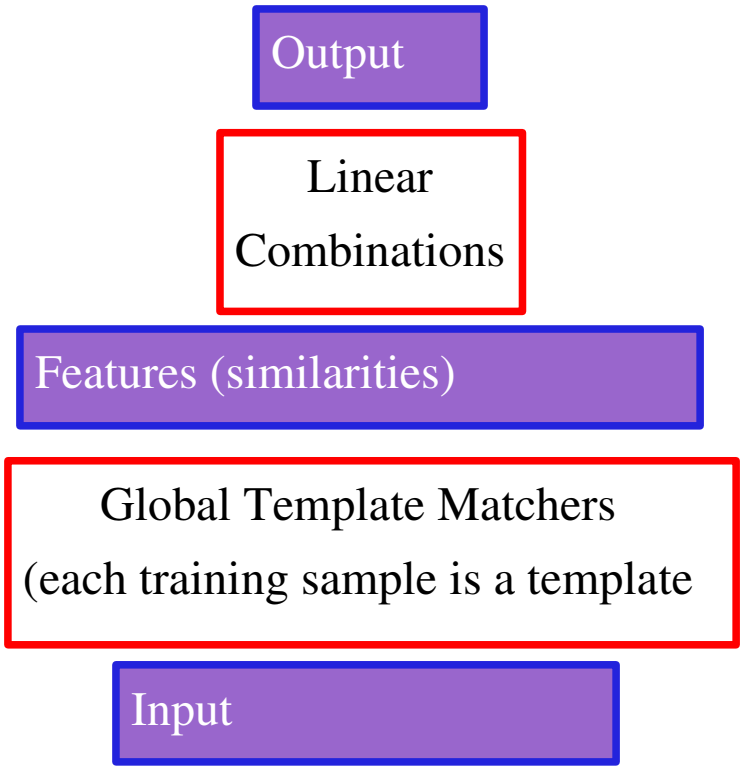
The convex loss, VC bounds and representers theorems don't seem to help

Chop off the last layer, and train an SVM on it it works!

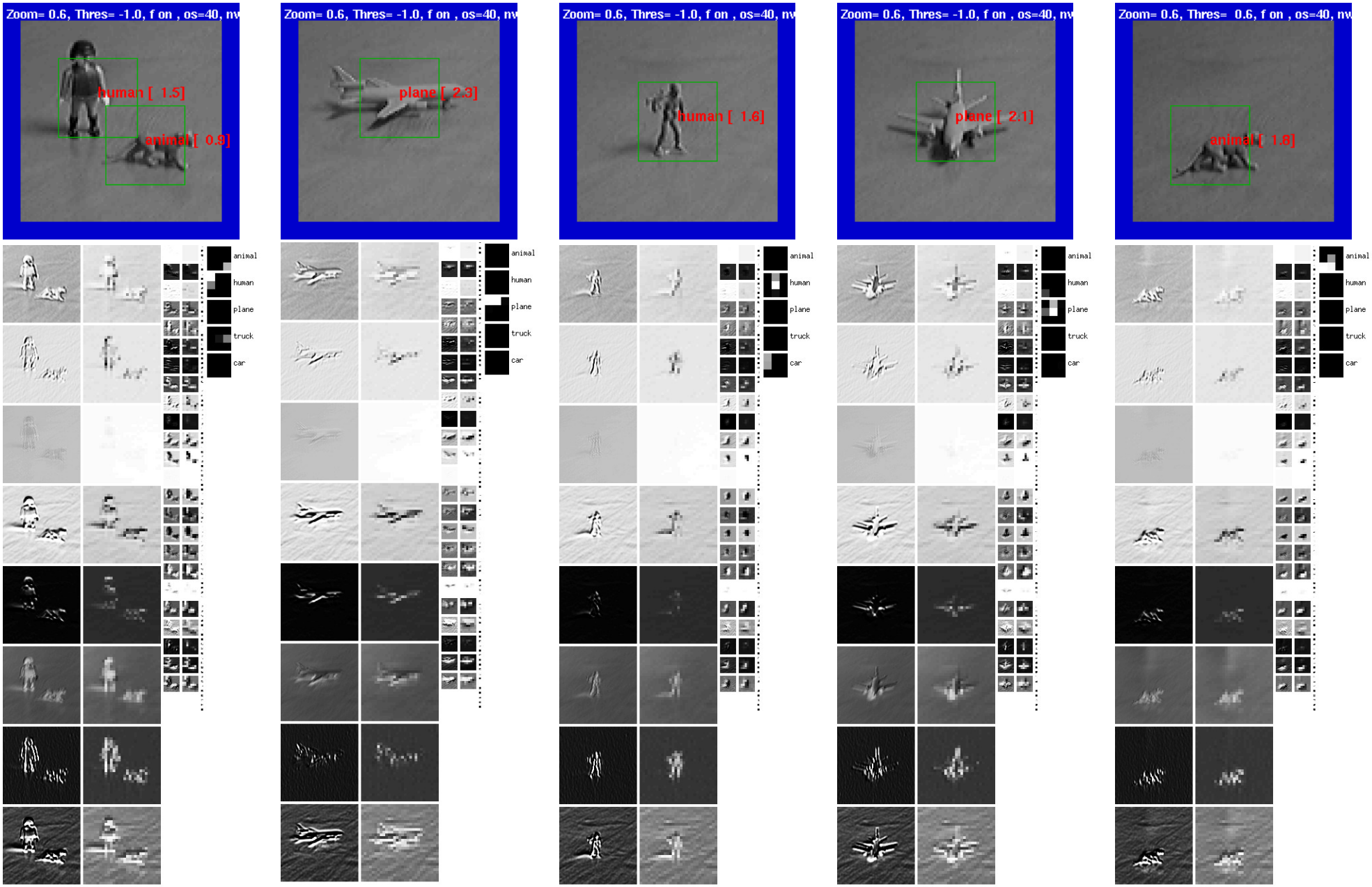
What's wrong with K-NN and SVMs?

- **K-NN and SVM with Gaussian kernels are based on matching global templates**
- **Both are "shallow" architectures**
- **There is now way to learn invariant recognition tasks with such naïve architectures (unless we use an impractically large number of templates).**

- **The number of necessary templates grows exponentially with the number of dimensions of variations.**
- **Global templates are in trouble when the variations include: category, instance shape, configuration (for articulated object), position, azimuth, elevation, scale, illumination, texture, albedo, in-plane rotation, background luminance, background texture, background clutter, ...**



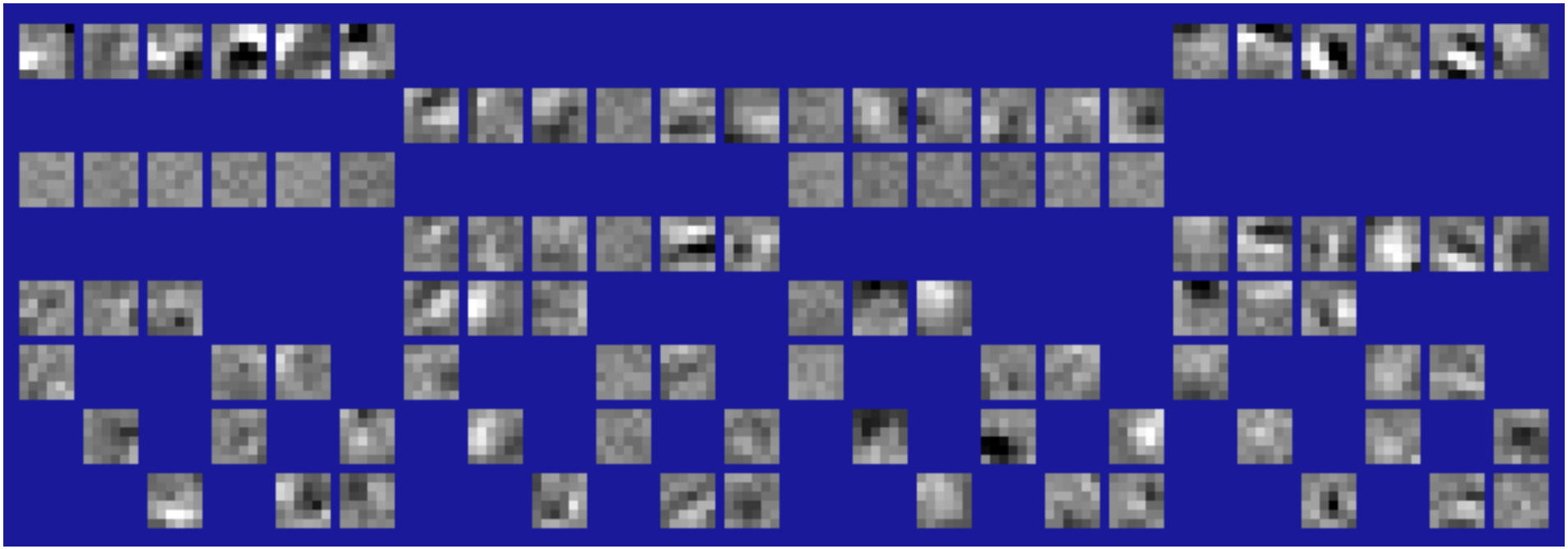
Examples (Monocular Mode)



Learned Features

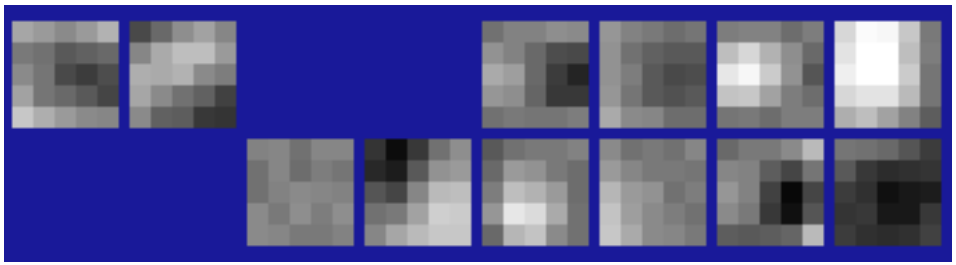
Layer 3

Layer 2

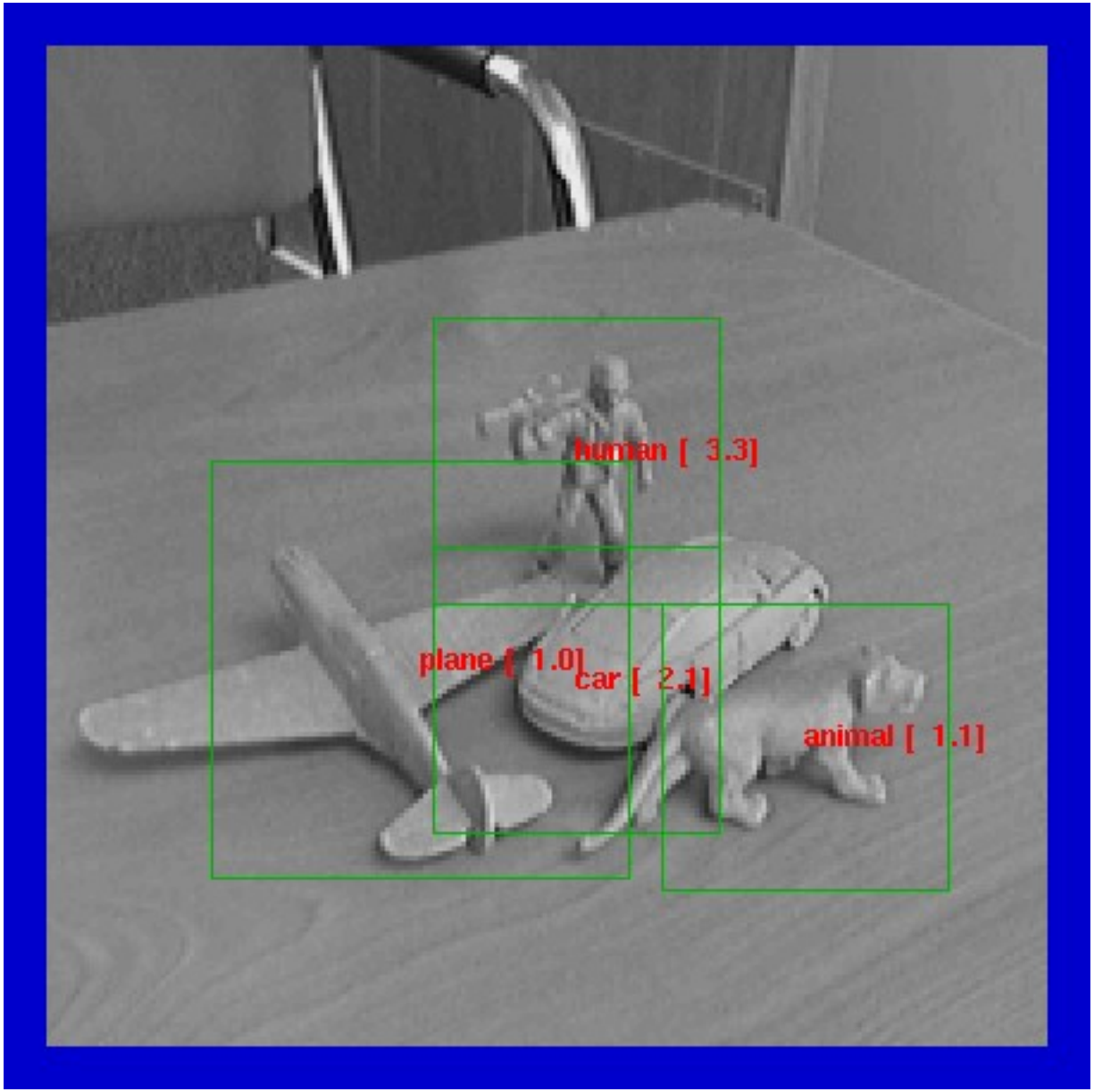


Layer 1

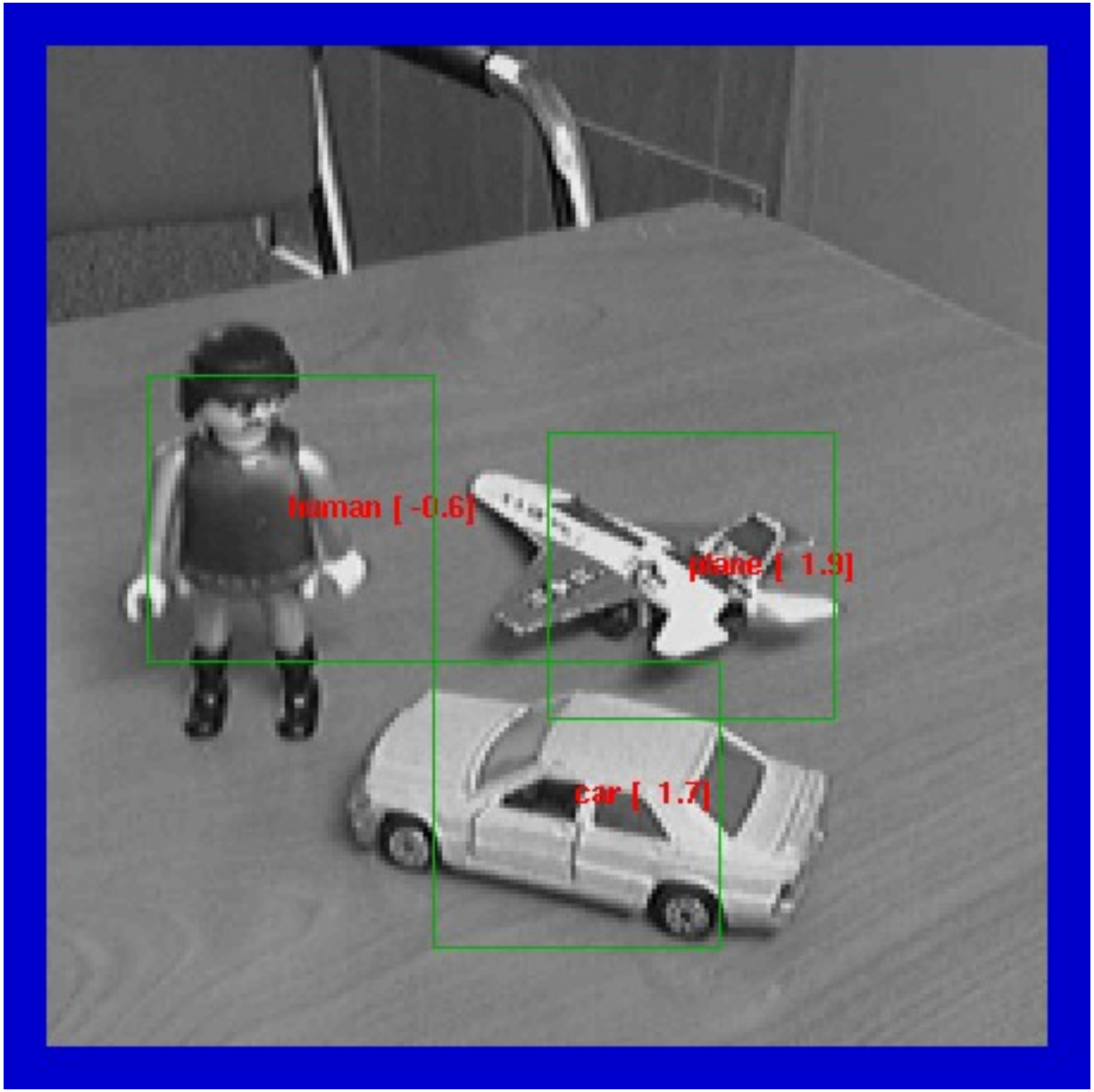
Input



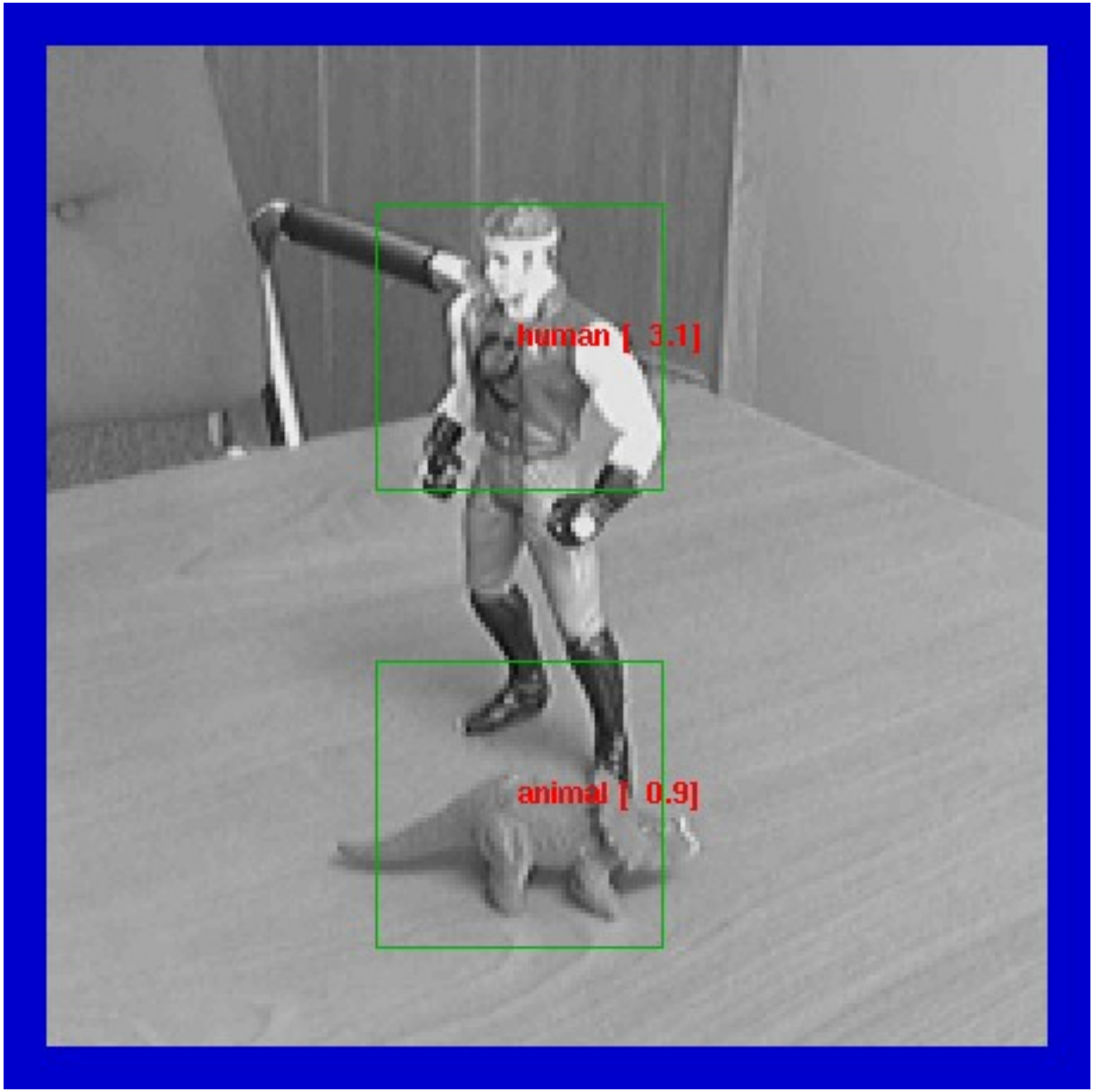
Examples (Monocular Mode)



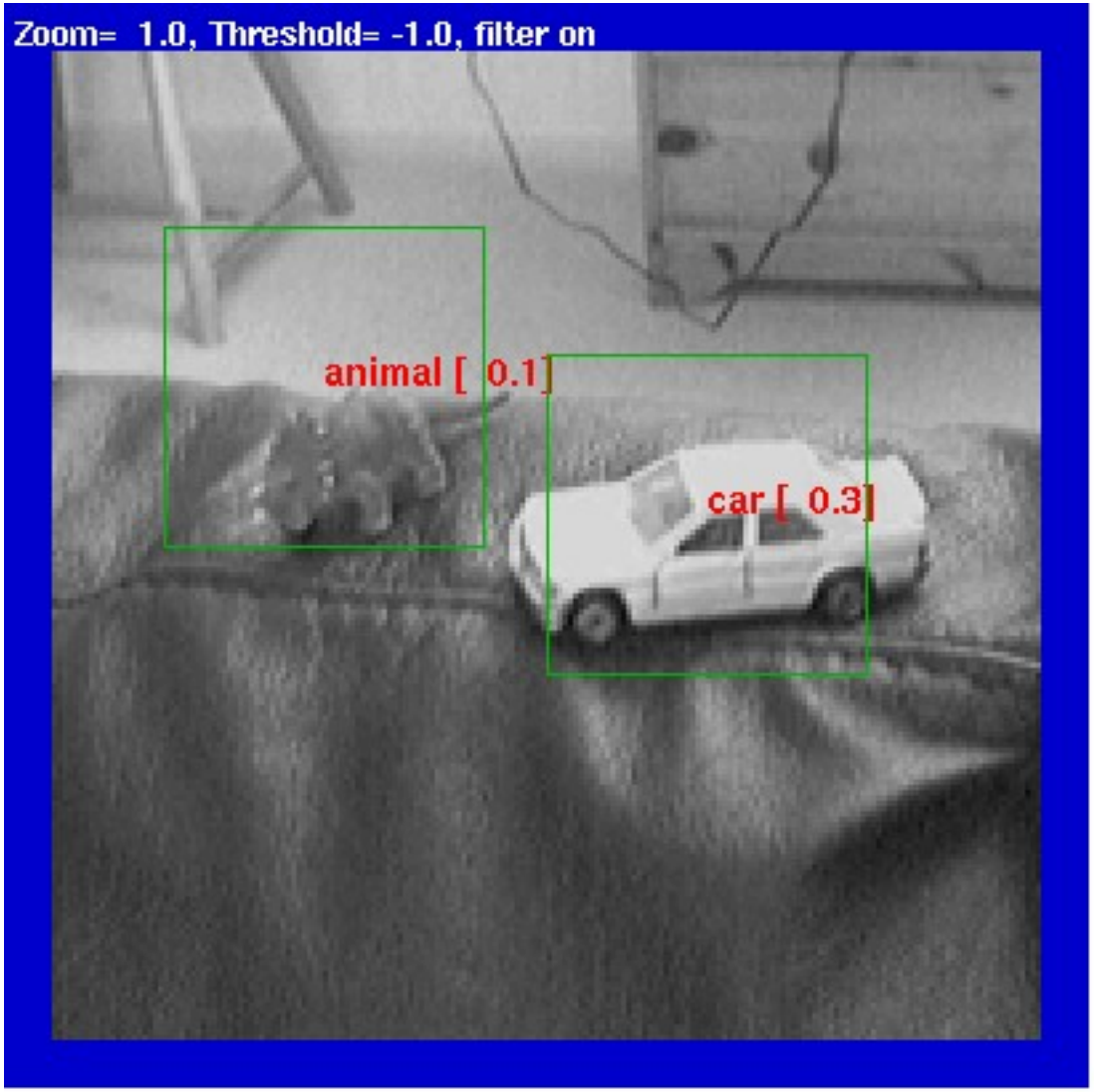
Examples (Monocular Mode)



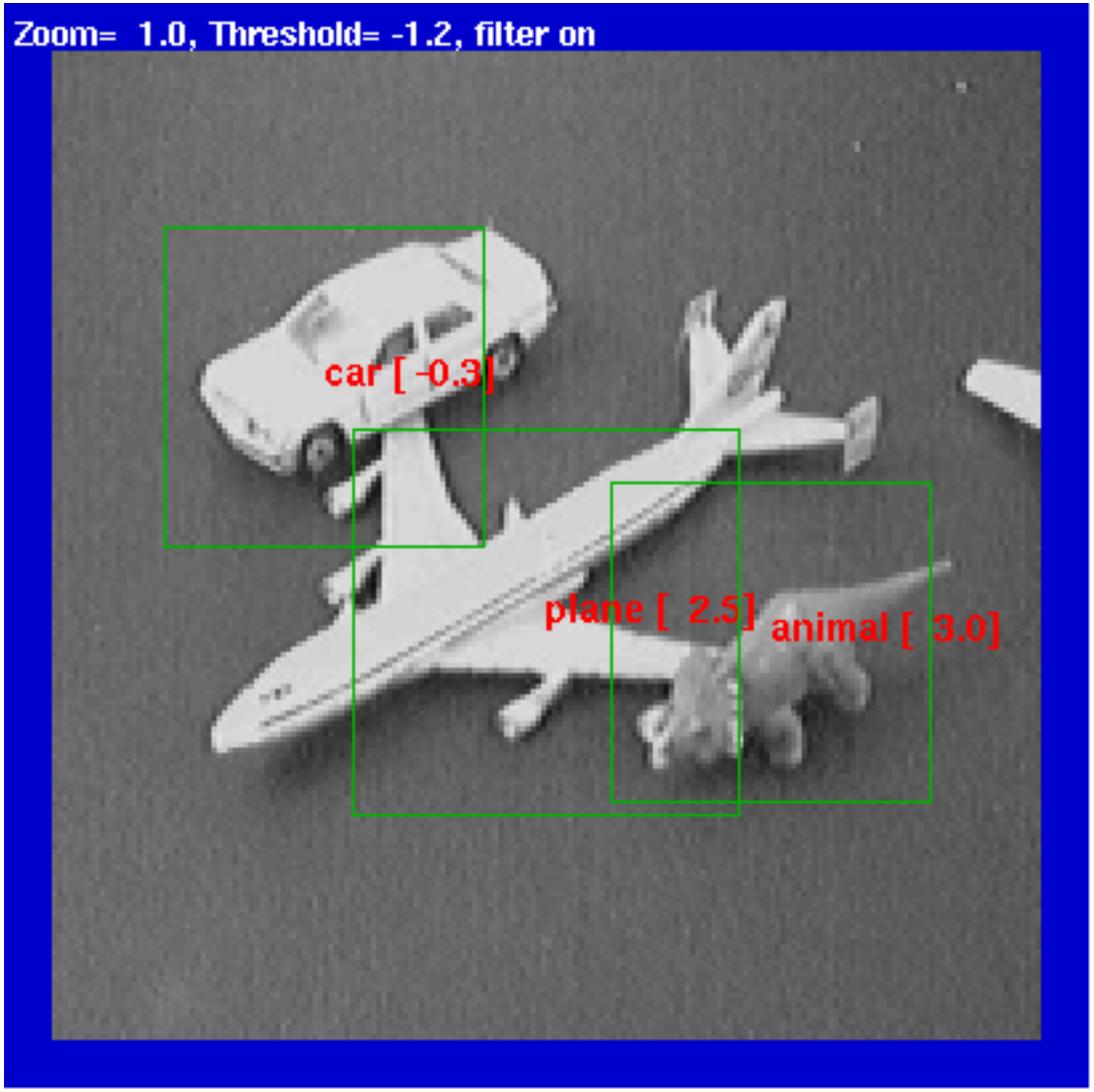
Examples (Monocular Mode)



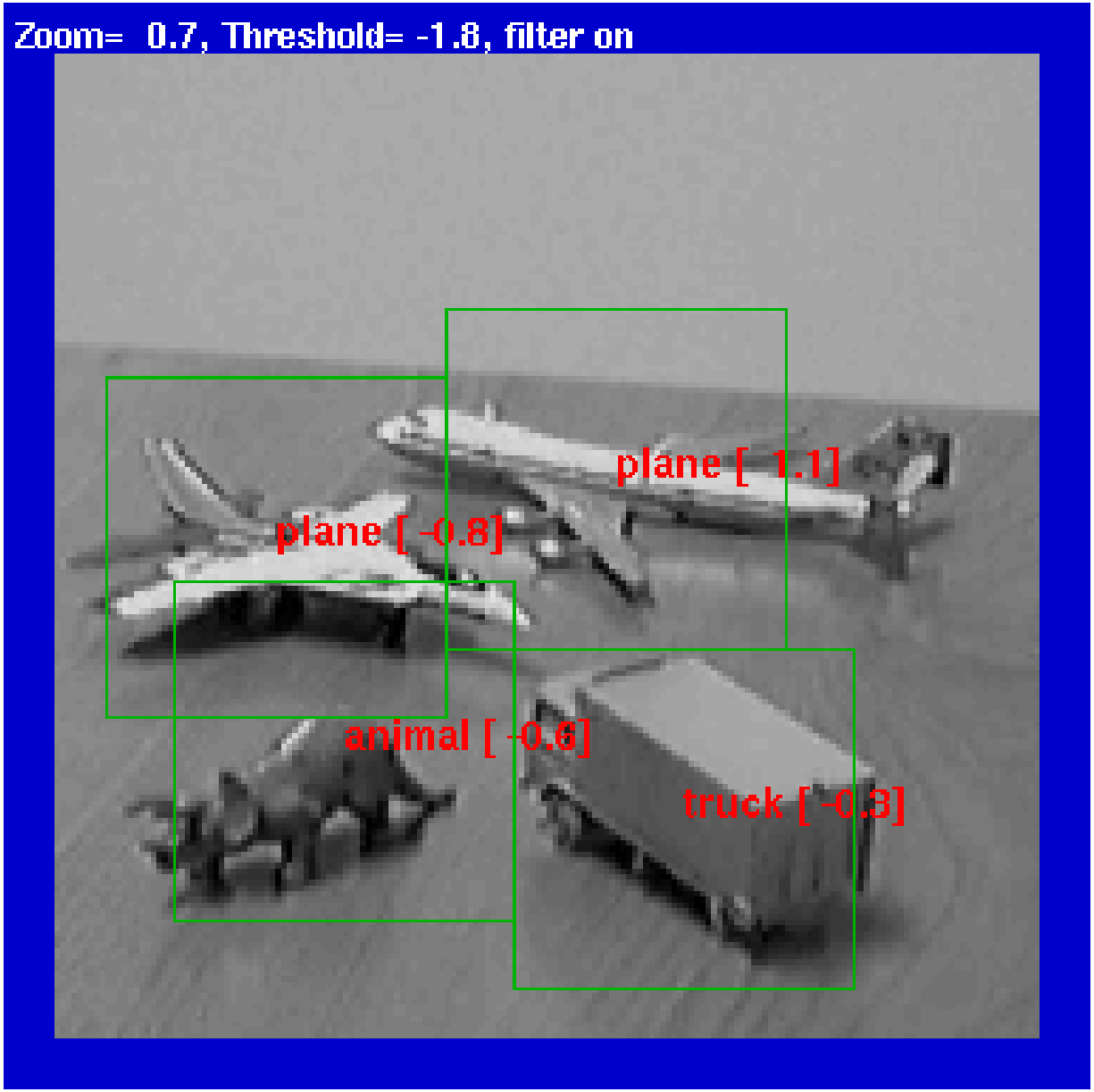
Examples (Monocular Mode)



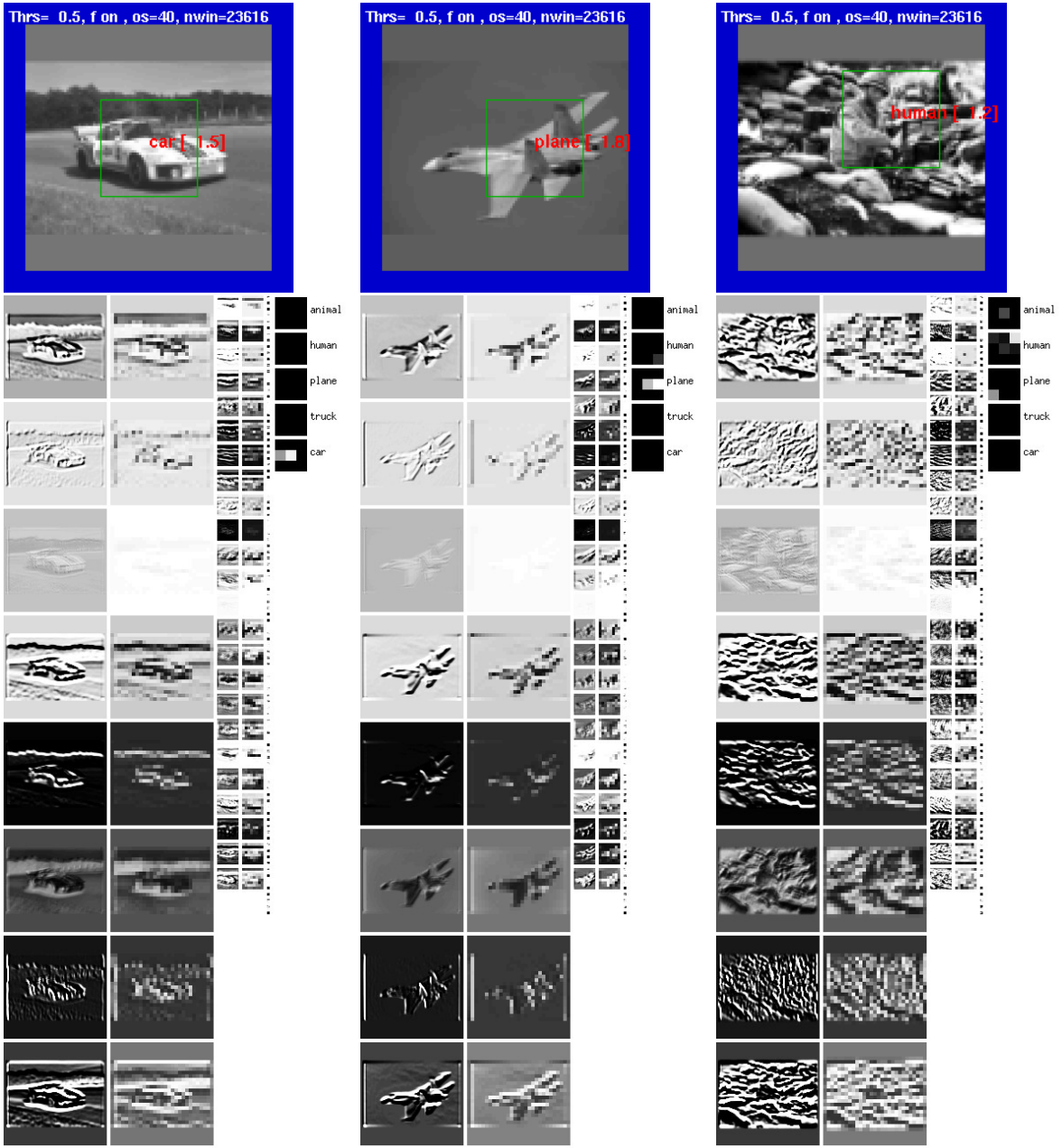
Examples (Monocular Mode)



Examples (Monocular Mode)

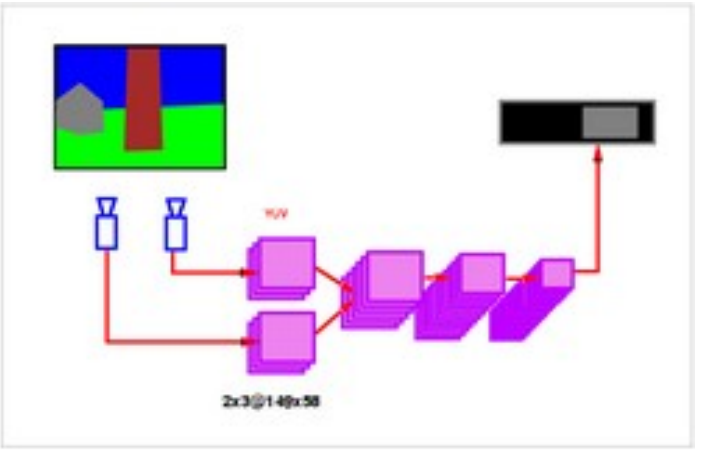


Natural Images (Monocular Mode)

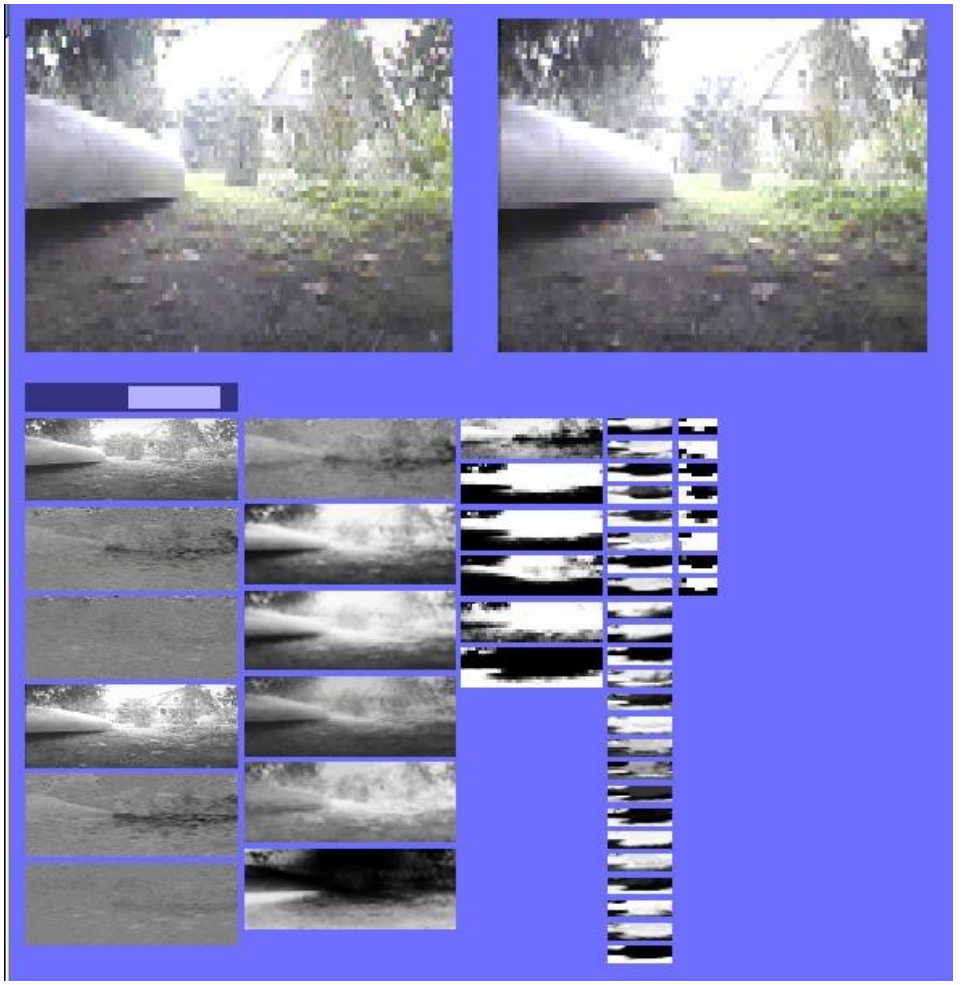


Visual Navigation for a Mobile Robot

[LeCun et al. NIPS 2005]



- Mobile robot with two cameras
- The convolutional net is trained to emulate a human driver from recorded sequences of video + human-provided steering angles.
- The network maps stereo images to steering angles for obstacle avoidance

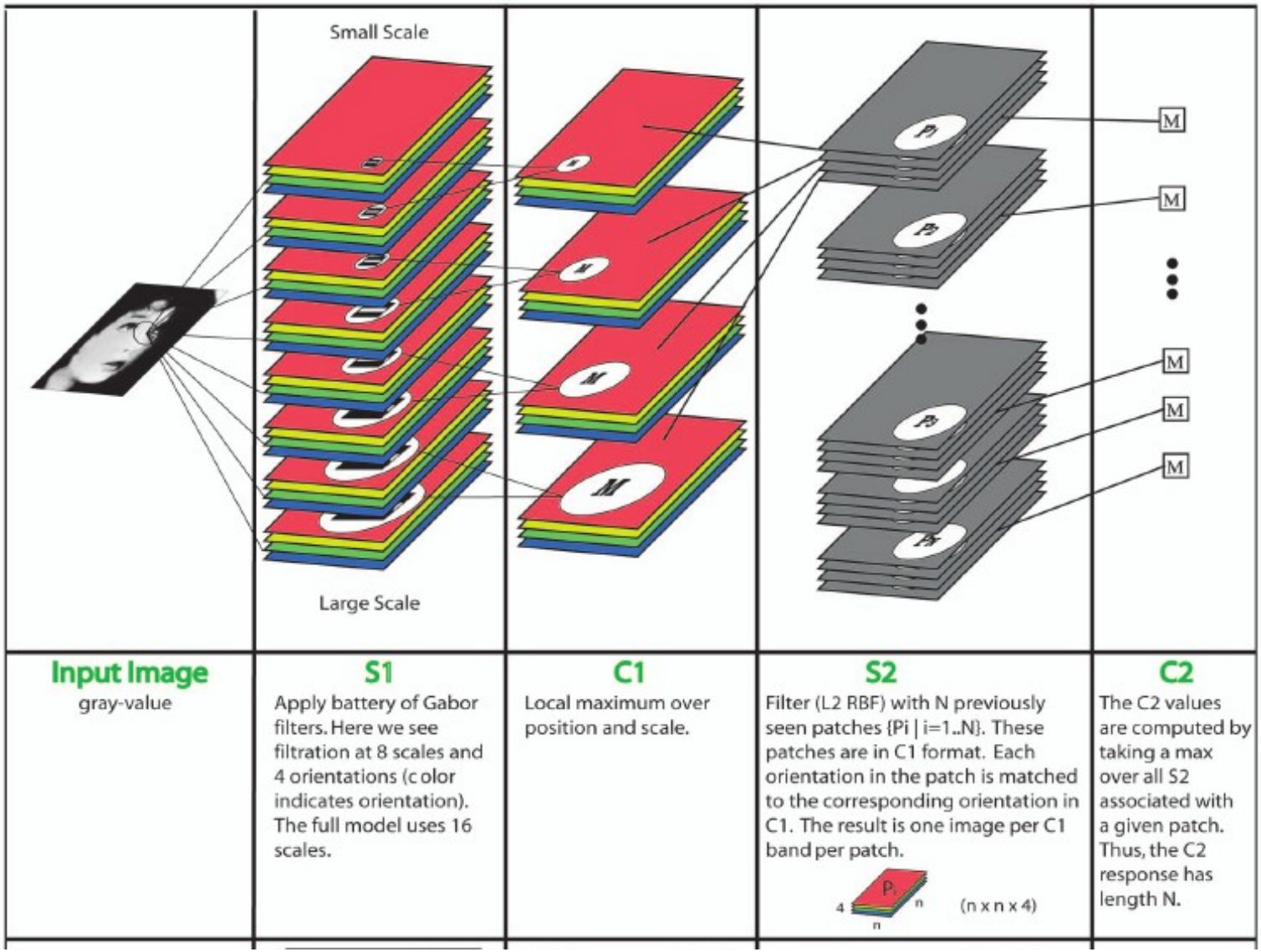


Supervised Convolutional Nets: Pros and Cons

- **Convolutional nets can be trained to perform a wide variety of visual tasks.**
  - Global supervised gradient descent can produce parsimonious architectures
- **BUT: they require lots of labeled training samples**
  - 60,000 samples for handwriting
  - 120,000 samples for face detection
  - 25,000 to 350,000 for object recognition
- **Since low-level features tend to be non task specific, we should be able to learn them unsupervised.**
- **Hinton has shown that layer-by-layer unsupervised "pre-training" can be used to initialize "deep" architectures**
  - [Hinton & Shalakhutdinov, Science 2006]
- **Can we use this idea to reduce the number of necessary labeled examples.**

Models Similar to ConvNets

- **HMAX**
  - [Poggio & Riesenhuber 2003]
  - [Serre et al. 2007]
  - [Mutch and Low CVPR 2006]
- **Difference?**
  - the features are not learned
- **HMAX is very similar to Fukushima's Neocognitron**



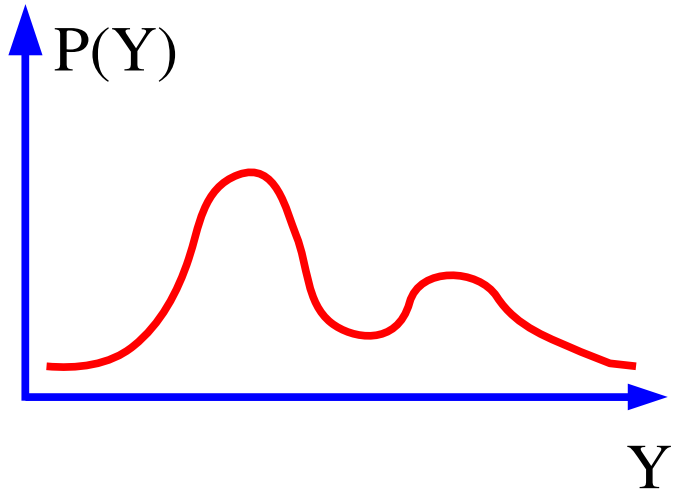
[from Serre et al. 2007]

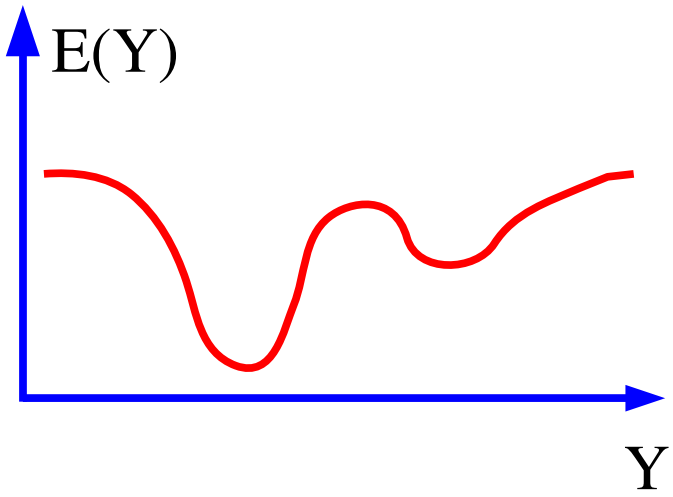
Part 2: Deep Unsupervised Learning for Dimensionality Reduction and Feature Extraction

- **Unsupervised Learning with "energy-based models"**
  - Contrastive Divergence, Product of Experts [Hinton, Neur Comp 2002]
  - Restricted Boltzmann Machines [Hinton et al 2003-]
  - General Encoder-Decoder model [Ranzato & al., AI-stats 2007]
- **Sparse-Overcomplete Feature Learning**
  - Sparse-Overcomplete Feature Learning [Olshausen & Fields 1997]
  - Feature Learning with PoE [Teh et al. JMLR 2003]
  - Sparse Features with Encoder-Decoder [Ranzato et al. NIPS 2006]
- **Convolutional image models**
  - Conditional Convolutional Product of Experts [Ning & al IEEE-TIP 2005]
  - Field of Experts [Roth and Black, CVPR 2005]

What is Unsupervised Learning?

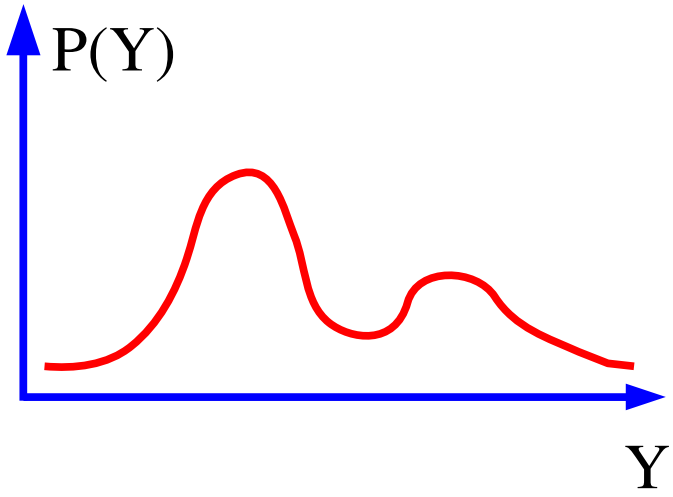
- **Probabilistic View:**
  - Produce a probability density function that:
  - has high value in regions of high sample density
  - has low value everywhere else (integral = 1).
- **Energy-Based View:**
  - produce an energy function E(Y) that:
  - has low value in regions of high sample density
  - has high(er) value everywhere else



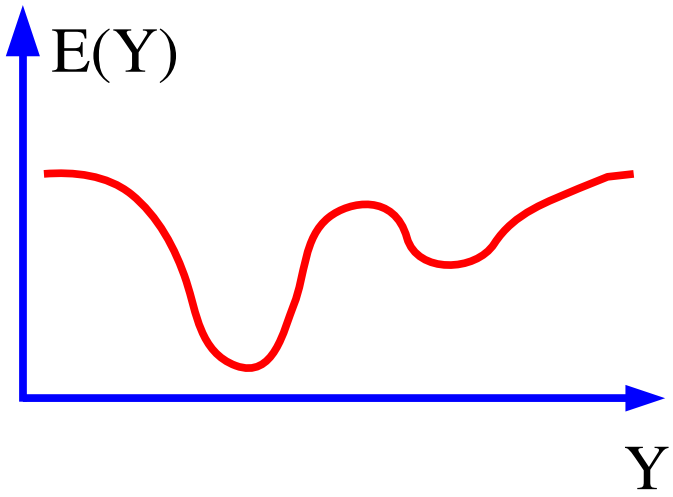


What is Unsupervised Learning?

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



$$E(Y,W) = -\log P(Y,W)$$



Training a Probabilistic Unsupervised Model

- **Minimize the negative log-likelihood of the data:**
  - Training set S = $\{Y^1, Y^2, Y^3, \ldots Y^P\}$

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

- **Gradient of the negative log-likelihood loss:**

$$\frac{\partial L(W, S)}{\partial W} = \frac{1}{P} \sum_{i=1}^{P} \frac{\partial E(Y^i, W)}{\partial W} - \int_y P(y, W) \frac{\partial E(y, W)}{\partial W}$$

Pushes down on the energy of the samples

Pushes up on the energy of everything

Training an Energy-Based Unsupervised Model

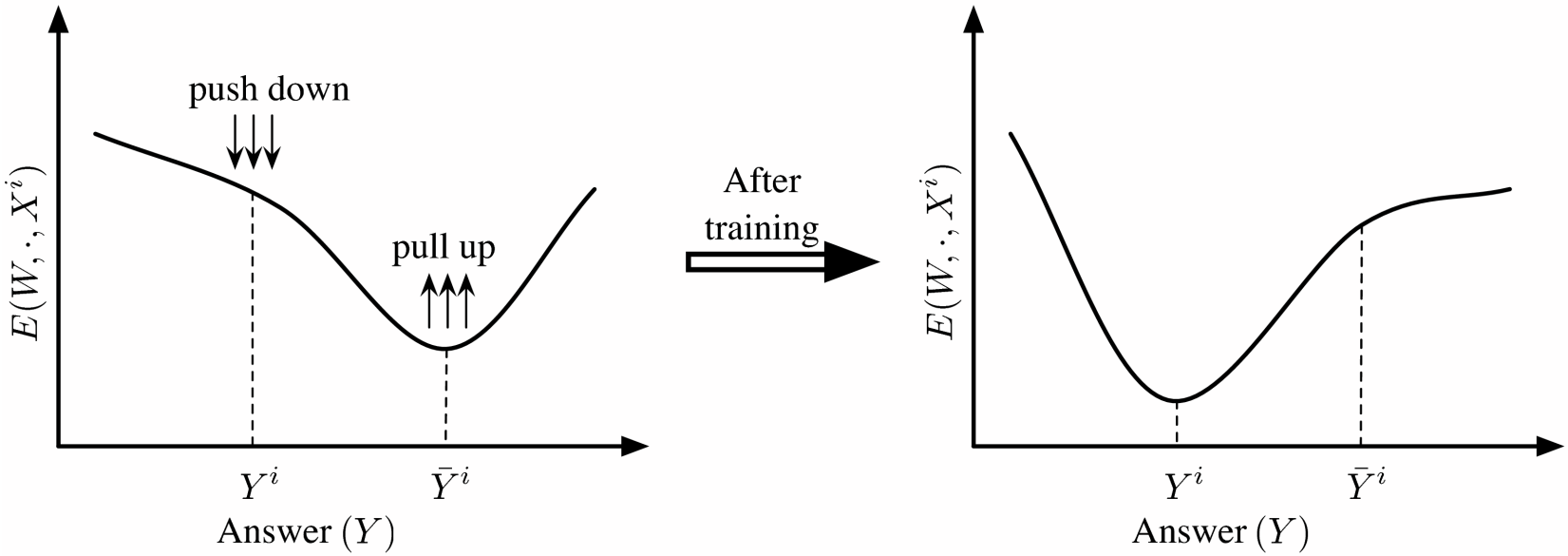
- **Design an architecture E(Y) and a loss function L(W,T) so that minimizing L(W,T) will:**
  - make the energy around training samples small
  - make the energy everywhere else larger
- **Question: how do make the energy everywhere else larger?**

Approximation to Minimize the neg-log-likelihood loss

- **Use an energy function such that contrastive term in the loss is either constant or easy to compute**
  - e.g. Energy is quadratic: convex (inference is easy), integral of exponential is easily computable or constant.
- **Approximate the derivative of the contrastive term in the loss with a variational approximation**
- **Simple sampling approximation:**
  - Pull down on the energy of the training samples
  - Pull up on the energies of other configurations that have low energy (that are threatening)
  - Question: how do we pick those configurations?
  - One idea: **contrastive Divergence**

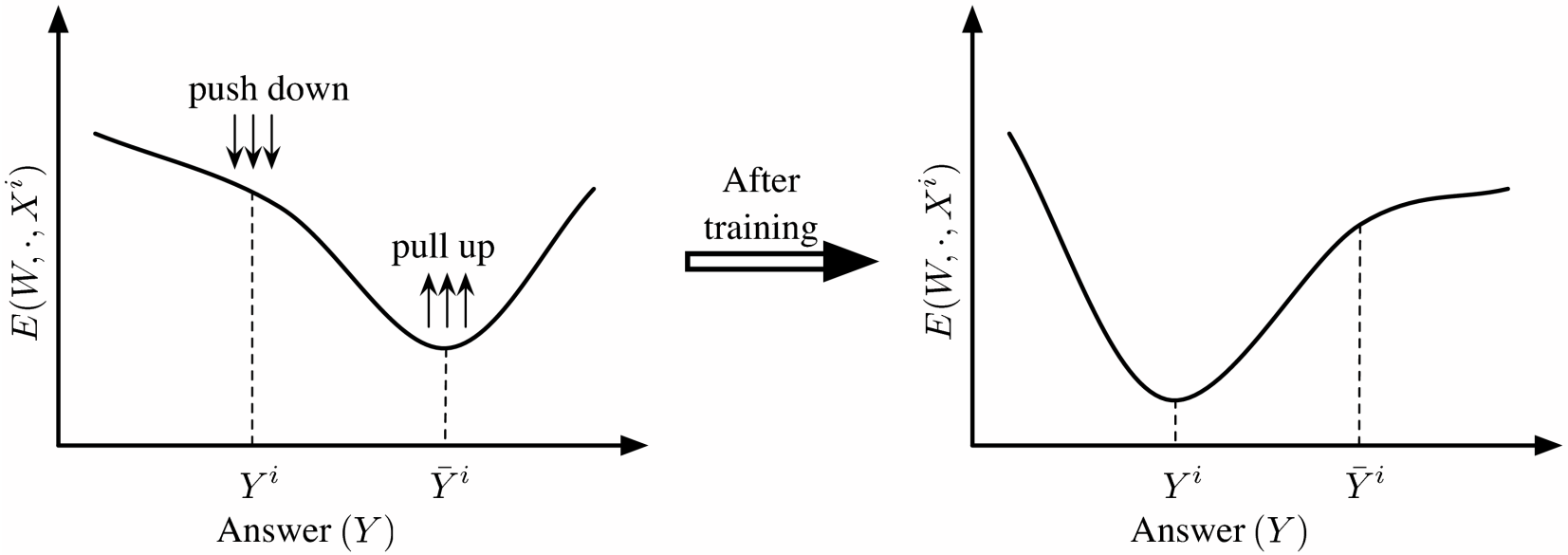
Contrastive Divergence [Hinton 2000]

- To generate the "bad" configurations:
- 1. Start from the correct value of Y
- 2. Pull down the energy of the correct value
- 3. To obtain a "bad" configuration, go down the energy surface with "some noise"
- 4. pull up the energy of the obtained configuration



Contrastive Divergence

- To generate the "bad" configurations:
- Hybrid Monte-Carlo Sampling: simulate a ball rolling down the energy surface in Y space.
- Kick the ball in the a random direction (with a random momentum), and run the simulation for a few iterations.
- The final configuration is quite likely to have lower energy than the starting point.



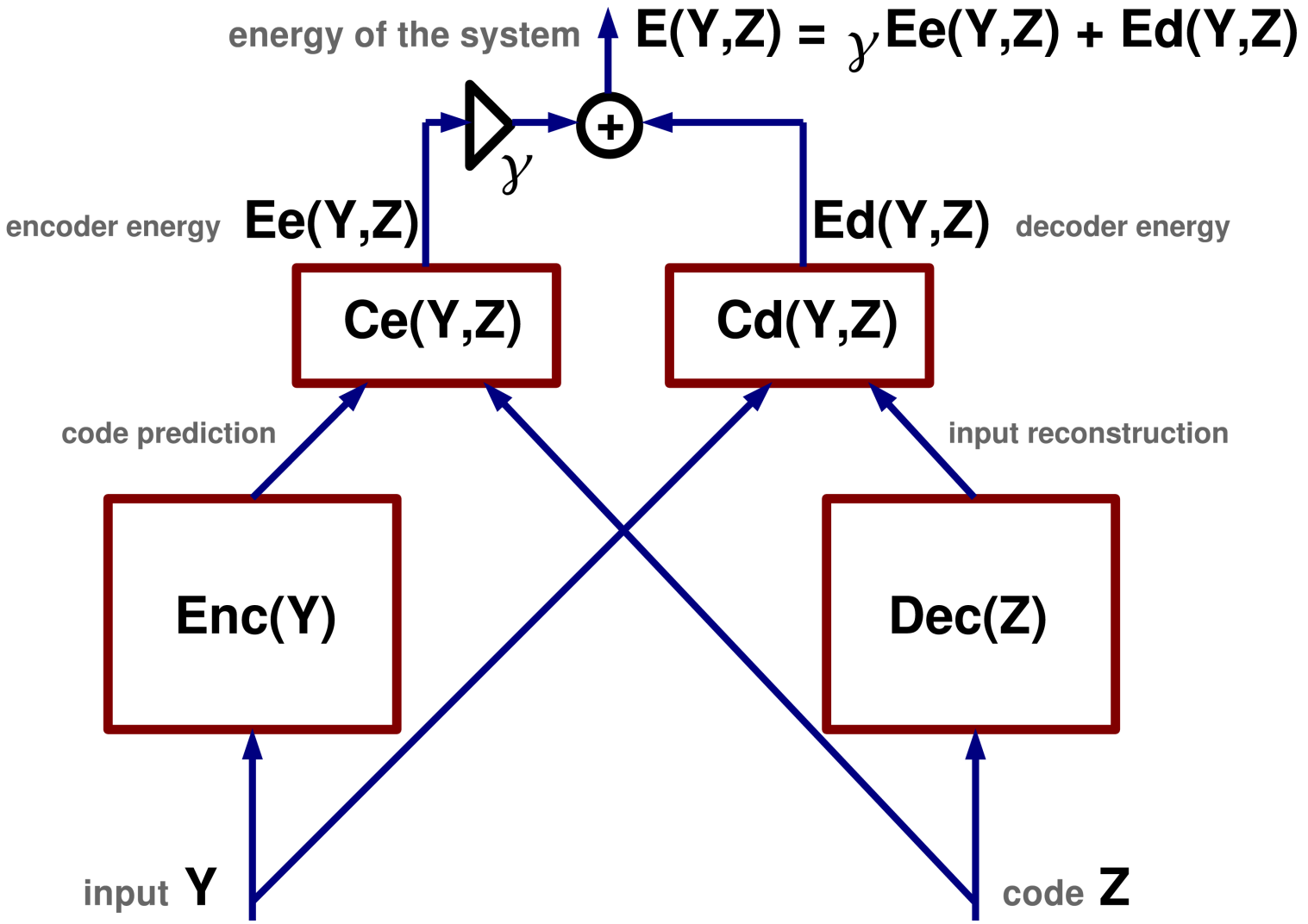
A common architecture: the encoder/decoder architecture

- **Many unsupervised learning algorithms use the encoder/decoder architecture.**
  - The energy has a hidden "code" variable Z:

$$E(Y, W) = min_Z E(Y, Z, W)$$

$$\bar{Z}_Y = argmin_Z E(Y, Z, W)$$

A common architecture: the encoder/decoder architecture



$$\bar{Z}_Y = argmin_Z E(Y, Z, W)$$

Encoder/decoder architecture: Examples

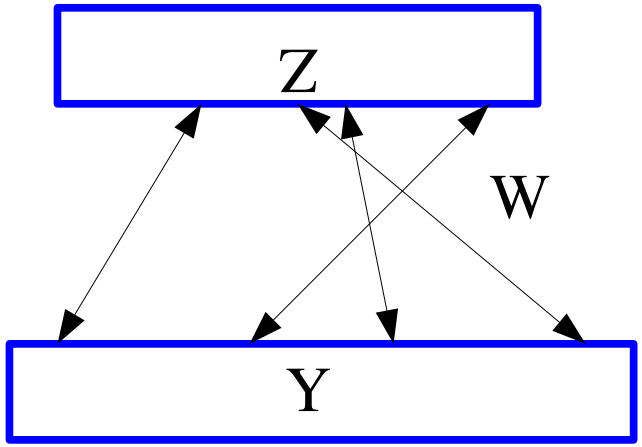
- **PCA**
  - encoder and decoder are linear projections
  - code is compact and dense
- **K-Means**
  - no encoder. Decoder is linear (matrix of prototypes times code vector)
  - code is 00001000, where the 1 is for the prototype that is closest to the input vector (very sparse binary code).
- **Auto-encoder neural net**
  - encoder and decoder can be non-linear
  - code is compact and dense
- **Sparse/Overcomplete representations (Olshausen, Pearlmutter...)**
  - no encoder. Decoder is linear (matrix of basis vectors times code vector)
  - code is sparse, and larger than the input

Restricted Boltzmann Machine [Hinton]

$$E(Y, Z, W) = \sum_{ij} -Y_i W_{ij} Z_j$$

$$E(Y, W) = -1/\beta \log \sum_Z \exp(-\beta E(Y, Z, W))$$

- **W is a symmetric matrix**
- **Z is a binary (stochastic) vector**
  - the distribution on Z can be approximated by sampling

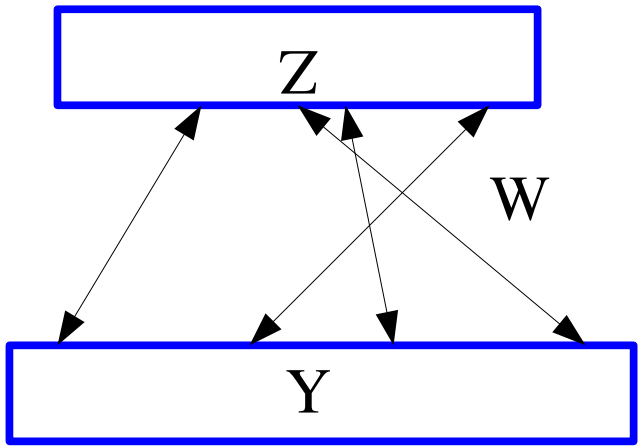


$$P(Z_j = 1/Y, W) = 1/(1 + \exp(\beta \sum_i W_{ij} Y_i))$$

$$P(Y_i = 1/Z, W) = 1/(1 + \exp(\beta \sum_j W_{ij} Z_j))$$

Restricted Boltzmann Machine: Learning Procedure

- 1. Clamp Y with observed data vector
- 2. Sample Z from $P(Z/Y,W)$
- 3. Sample $\bar{Y}$ from $P(Y/Z,W)$
- 4. Sample $\bar{Z}$ from $P(Z/\bar{Y},W)$
- update W with



$$W_{ij} \leftarrow W_{ij} + \eta(Y_i Z_j - \bar{Y}_i \bar{Z}_j)$$

The learning rule minimizes the loss:

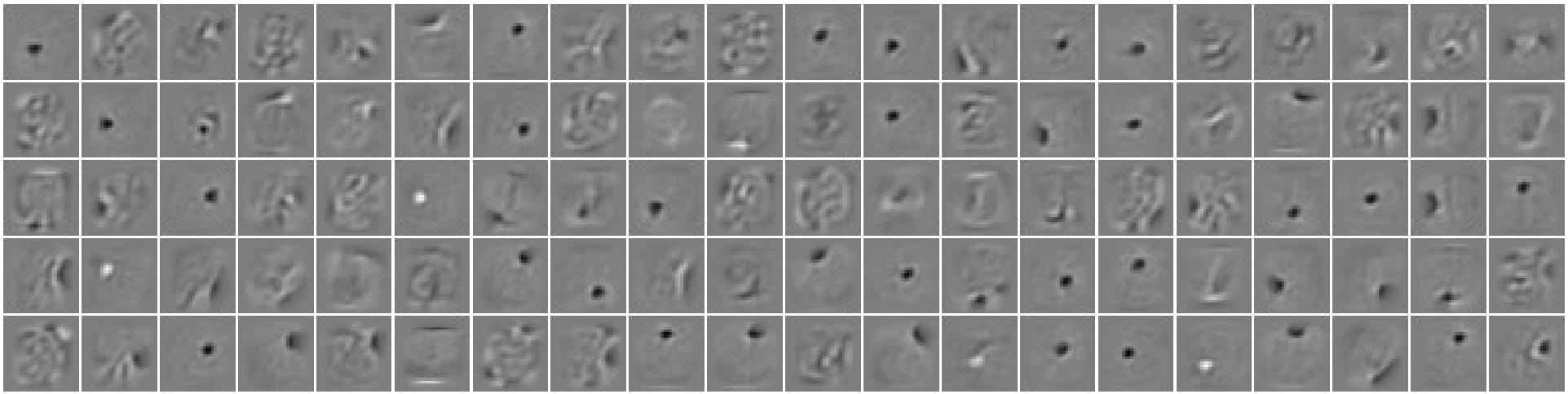
$$L(W,Y) = E(Y,Z,W) - E(\bar{Y},\bar{Z},W)$$

which can be seen a sampled approximation of

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

RBM: filters trained on MNIST

- "bubble" detectors



Common Feature of Encoder/Decoder Models

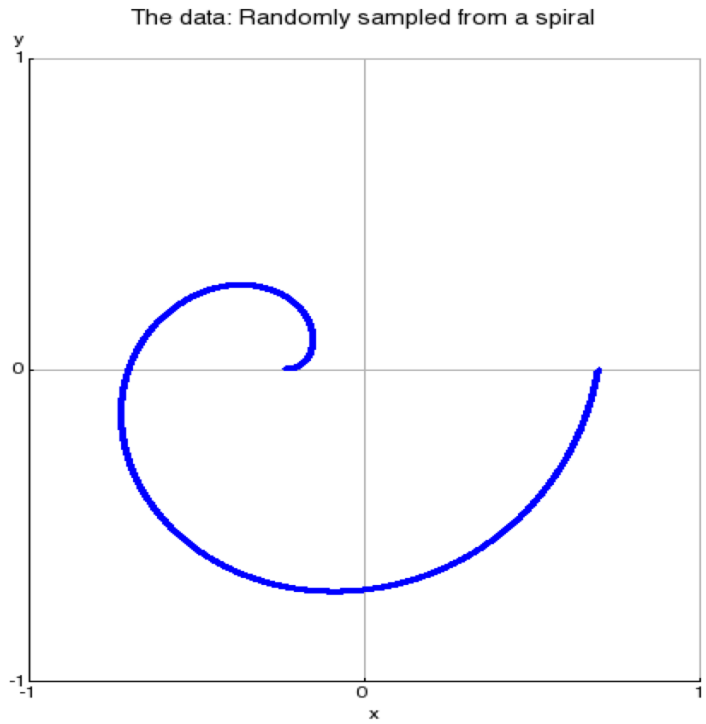
- **All encoder/decoder models have one thing in common: they restrict the information content of the code**
  - by making it compact (PCA, auto-encoders)
  - by making it sparse (K-Means, sparse/overcomplete bases)
  - by making it binary (K-Means)
- **When the information content in the code is restricted:**
  - there are fewer possible code configurations than input vectors
  - each code configuration produces a single reconstructed vector
  - an input vector is exactly reconstructed **only if** it is equal to one of those reconstructed code vectors.
  - every other vector is imperfectly reconstructed.
  - since the energy is the reconstruction error, this means that only a few (perfectly reconstructed) input vectors can have low energy. All others have high energy.
  - **Hence, restricting the information content of the code alleviates the need to push up of the energy of everything.**

Restricting the Information Content of the Code

- Restricting the information content of the code alleviates the need to push up of the energy of everything.
- Hence, we can happily use a simple loss function that simply pulls down on the energy of the training samples.
- We do not need a contrastive term that pulls up on the energy everywhere else.

Example: A Toy Problem. The Spiral

- Dataset
  - 10,000 random points along a spiral in a 2D plane
  - The spiral fits in a square with opposite corners (-1,1), (1,-1)
  - The spiral is designed so that no function can predict a single value of from
- Goal
  - Learn an energy surface with low energies along the spiral and high energy everywhere else



PCA (Principal Component Analysis)

- Can be seen as encoder-decoder architecture that minimizes mean square reconstruction error (energy loss)
- Optimal code is constrained to be equal to the value predicted by encoder
- Flat surfaces are avoided by using a code of low dimension

$$Enc(Y) = WY$$

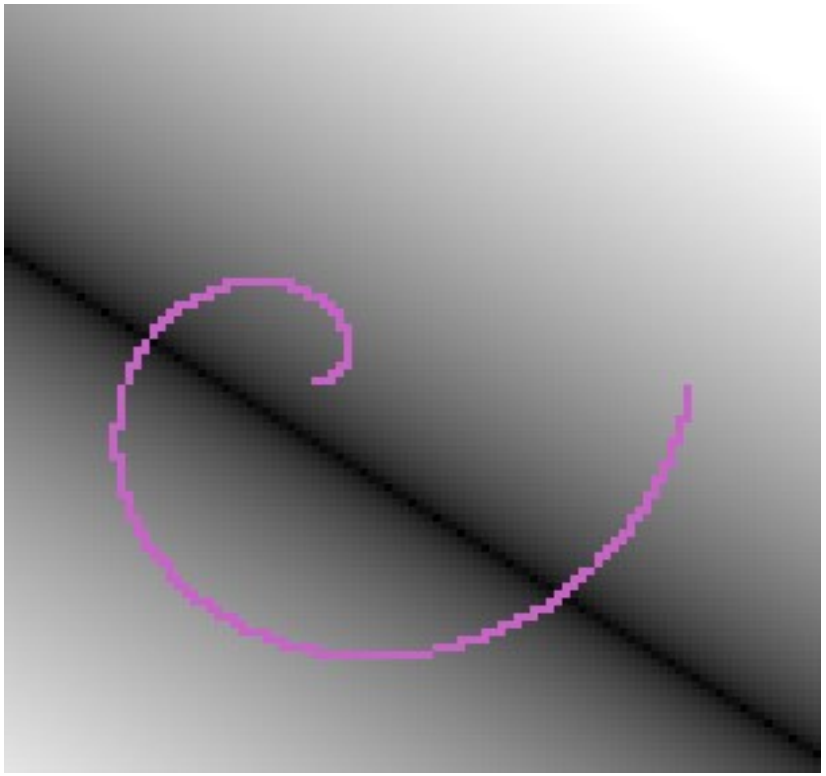
$$Dec(Z) = W^T Y \text{ , where } W \in \Re^{NxM}$$

$$C_e(Y, Z) = \|WY - Z\|^2$$

$$C_d(Y, Z) = \|W^T Z - Y\|^2$$

- For large value of γ energy reduces to

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



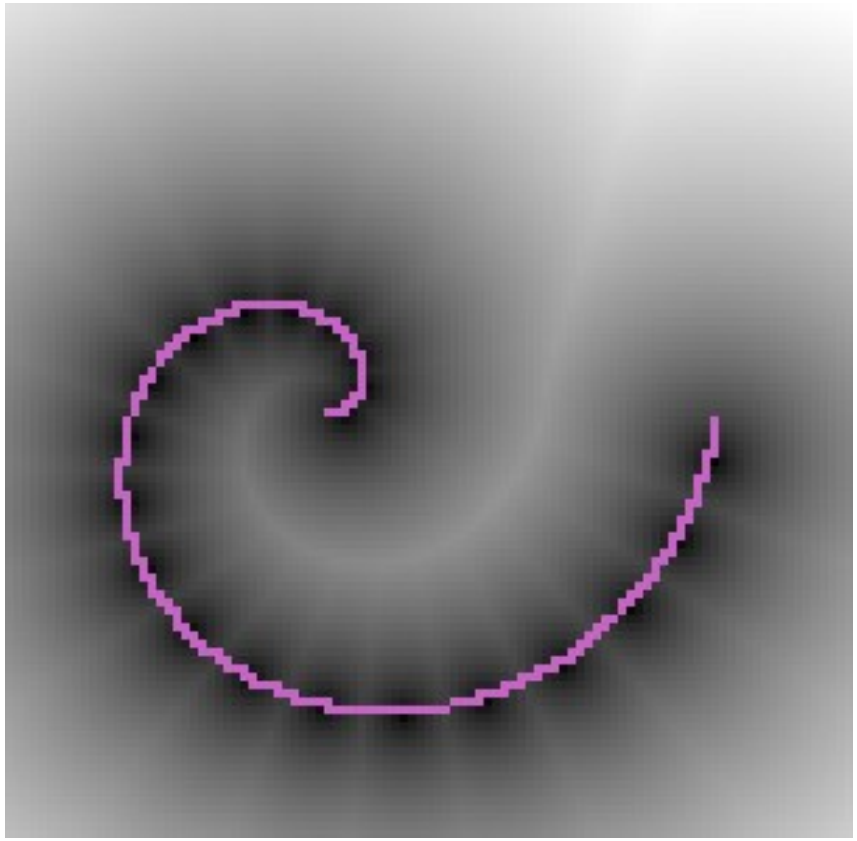
K-Means

- In this architecture the code Z is a binary vector of size N (N being the number of prototypes)
- For any sample Y, only one component is active (equal to 1) and all the others are inactive (equal to 0).
- This is a one-of-N sparse binary code
- The energy is:

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

W_i *is the* i^{th} *prototype*

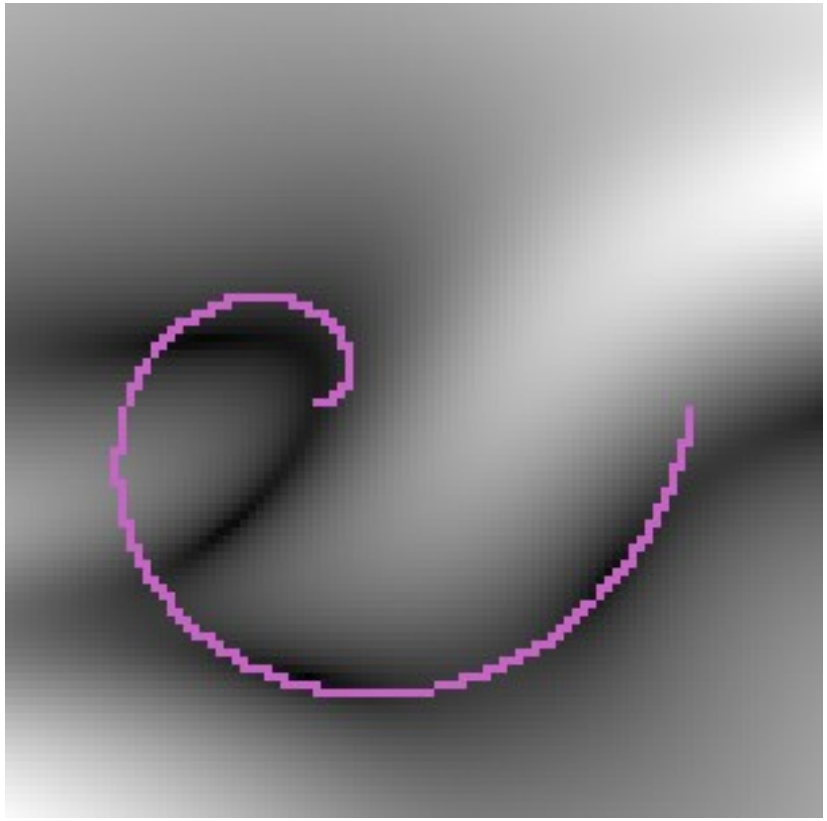
- Inference involves finding Z that minimizes the energy



Auto-encoder neural net (2-100-1-100-2 architecture)

- A neural network autoencoder that learns a low dimensional representation.
- Architecture used
  - Input layer: 2 units
  - First hidden layer: 100 units
  - Second hidden layer (the code): 1 unit
  - Third hidden layer: 100 units
  - Output layer: 2 units
- Similar to PCA but non-linear
- Energy is

$$E(Y) = |Dec(Enc(Y)) - Y|^2$$

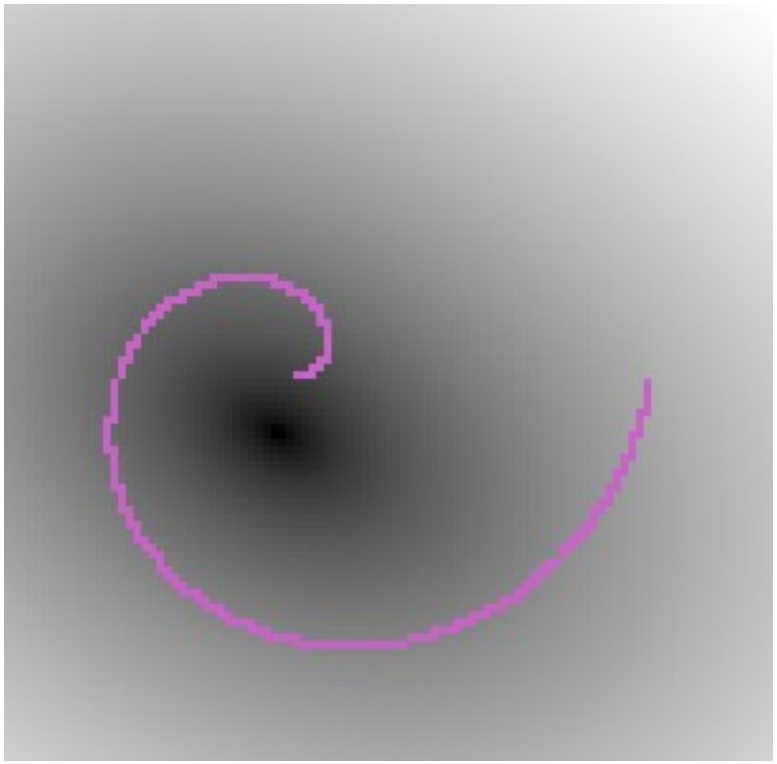


Wide auto-encoder neural net with energy loss

- The energy loss simply pulls down on the energy of the training samples (no contrastive term).
- Because the dimension of the code is larger than the input, nothing prevents the architecture from learning the identity function, which gives a very flat energy surface (a collapse): everything is perfectly reconstructed.
- Simplest example: a multi layer neural network with identical input and output layers and a large hidden layer.
- Architecture used
  - Input layer: 2 units
  - Hidden layer (the code): 20 units
  - Output layer: 2 units
- Energy loss leads to a collapse
- Tried a number of loss functions

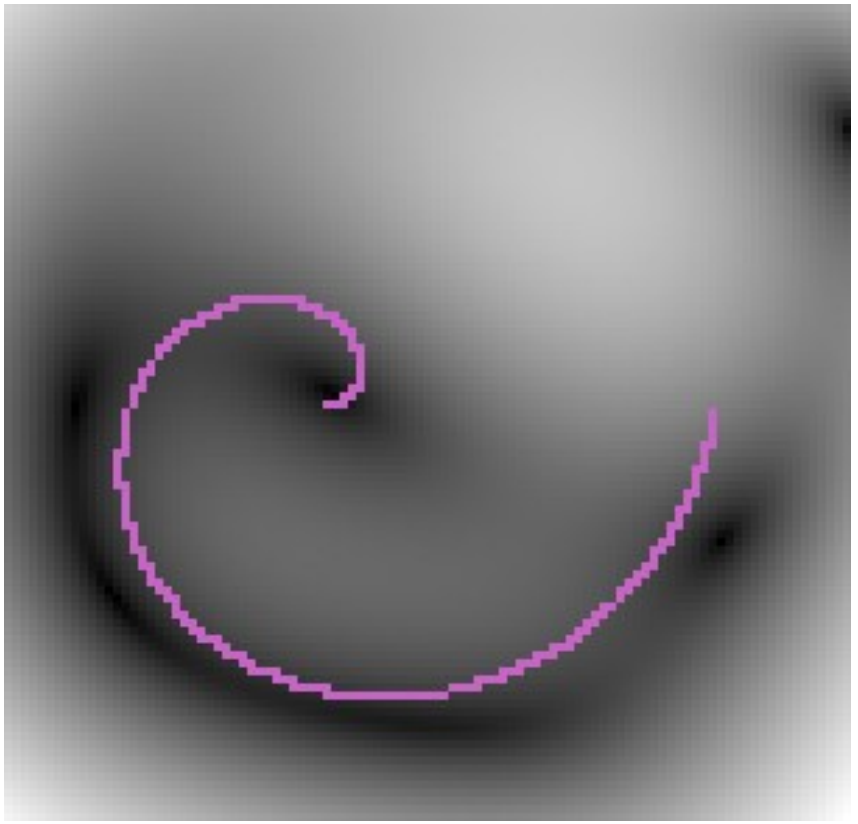
Wide auto-encoder with negative-log-likelihood loss

- Negative Log Likelihood Loss
  - Pull down on the energy of training (observed) samples
  - Pull up on the energies of all the other (unobserved) samples
  - Approximate the log of partition function through dense sampling.
  - Energy surface is very “stiff” because of small number of parameters.
  - Hence the energy surface is not perfect.



Wide auto-encoder with energy-based contrastive loss

- Linear-Linear Contrastive Loss
  - Avoid the cost associated with minimizing negative log likelihood
  - Idea is to pull up on unobserved points in the vicinity of training samples
  - We use Langevin dynamics to generate such points



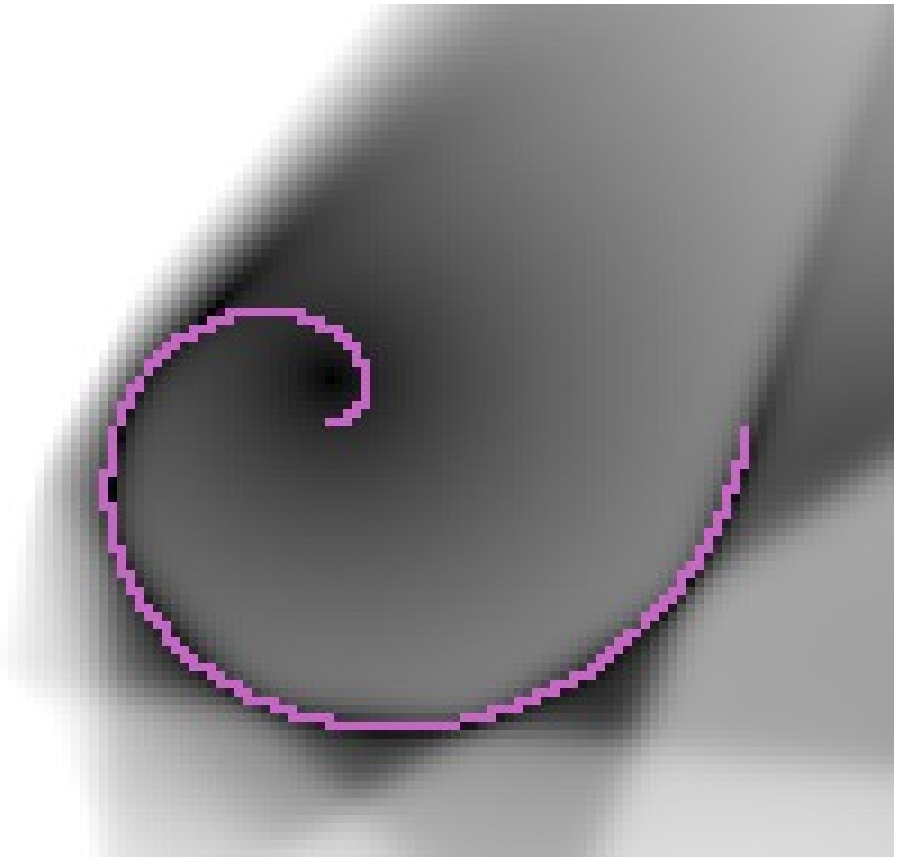
$$\bar{Y} \leftarrow \bar{Y} - \eta \frac{\delta E(\bar{Y})}{\delta Y} + \epsilon$$

  - The loss function is

$$L(Y, W) = \alpha E(Y, W) + max(0, m - E(\bar{Y}, W))$$

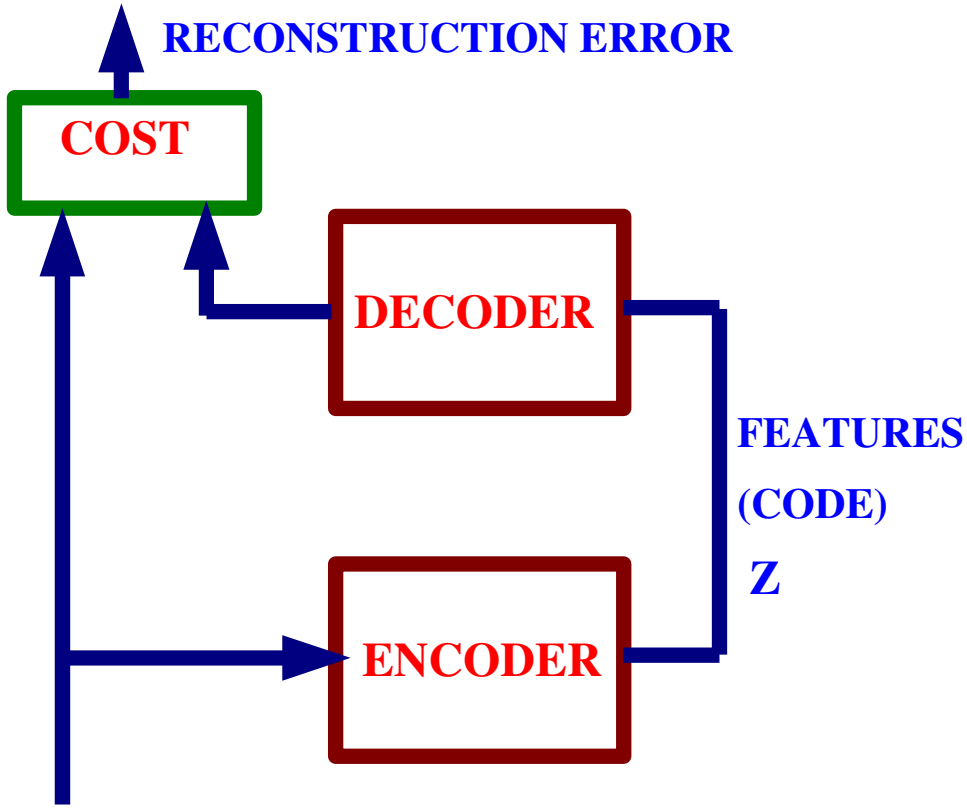
Wide auto-encoder with sparse code

- Sparse Codes
  - Limiting the information content of the code prevents flat energy surfaces, without the need to explicitly push up the bad points
  - Idea is to make the high dimensional code sparse by forcing each variable to be zero most of the time



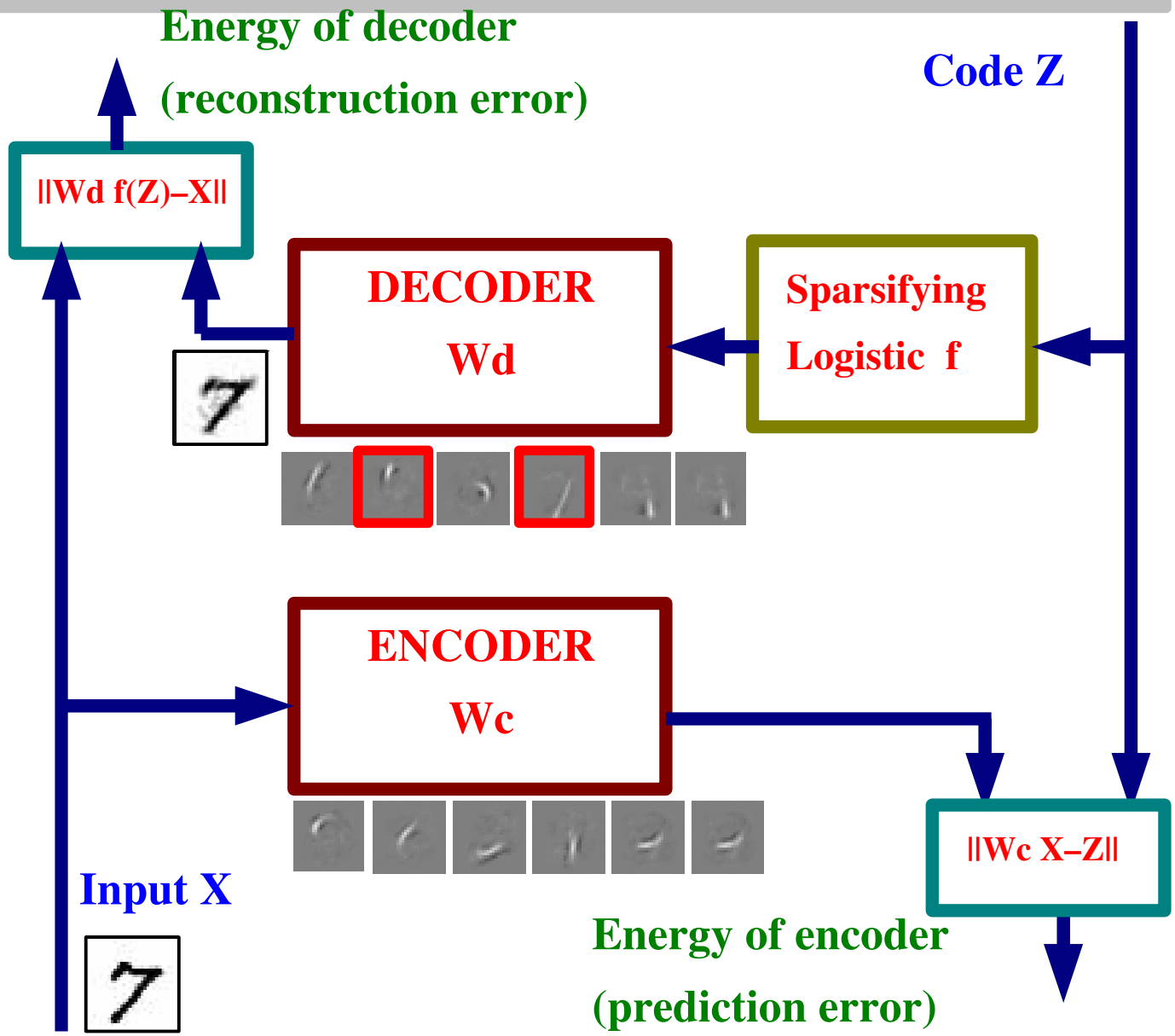
Encoder-Decoder Architecture for Unsupervised Training

- **A principle on which unsupervised algorithms can be built is reconstruction of the input from a code (feature vector)**
  - reconstruction from compact feature vectors (e.g. PCA).
  - reconstruction from sparse overcomplete feature vectors (Olshausen & Field 1997)



Encoder/Decoder Architecture for learning Sparse Feature Representations

- **Algorithm:**
  - 1. find the code Z that minimizes the reconstruction error AND is close to the encoder output
  - 2. Update the weights of the decoder to decrease the reconstruction error
  - 3. Update the weights of the encoder to decrease the prediction error

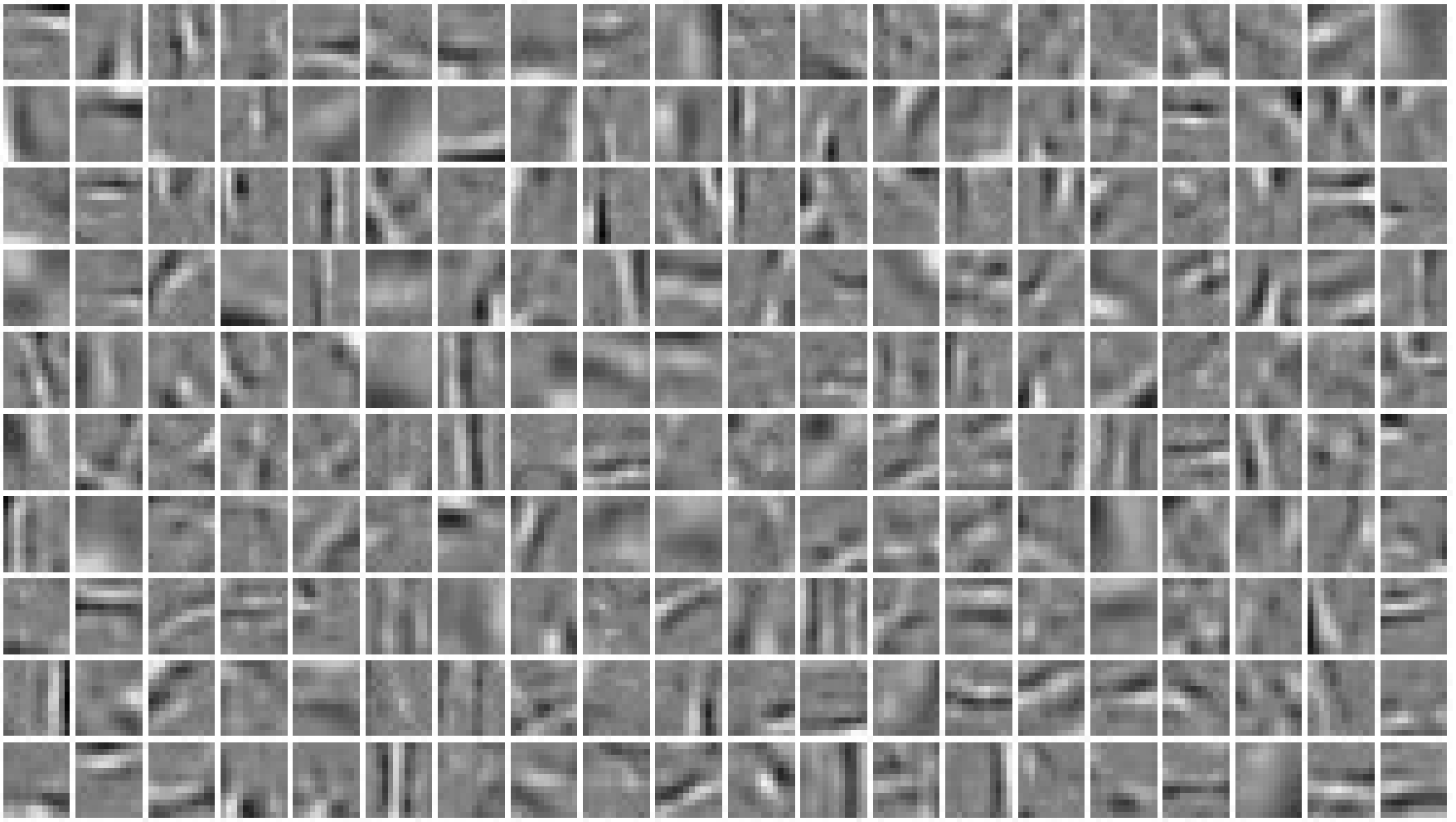


Training on natural image patches

Berkeley data set

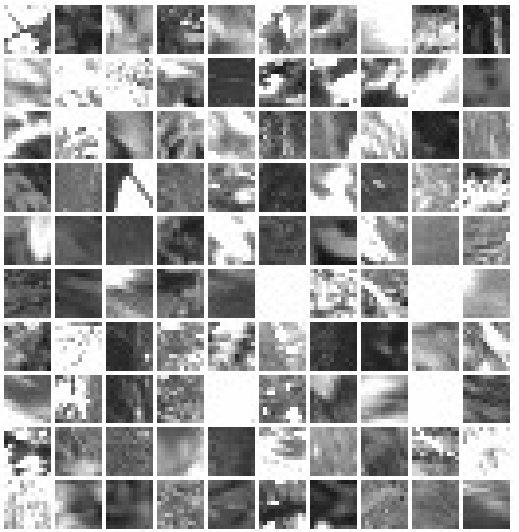
- 100,000 12x12 patches
- 200 units in the code
- η 0.02
- β 1
- learning rate 0.001
- L1 regularizer 0.001
- fast convergence: < 30min.

Natural image patches: Filters



200 decoder filters (reshaped columns of matrix $\mathbf{W_d}$)

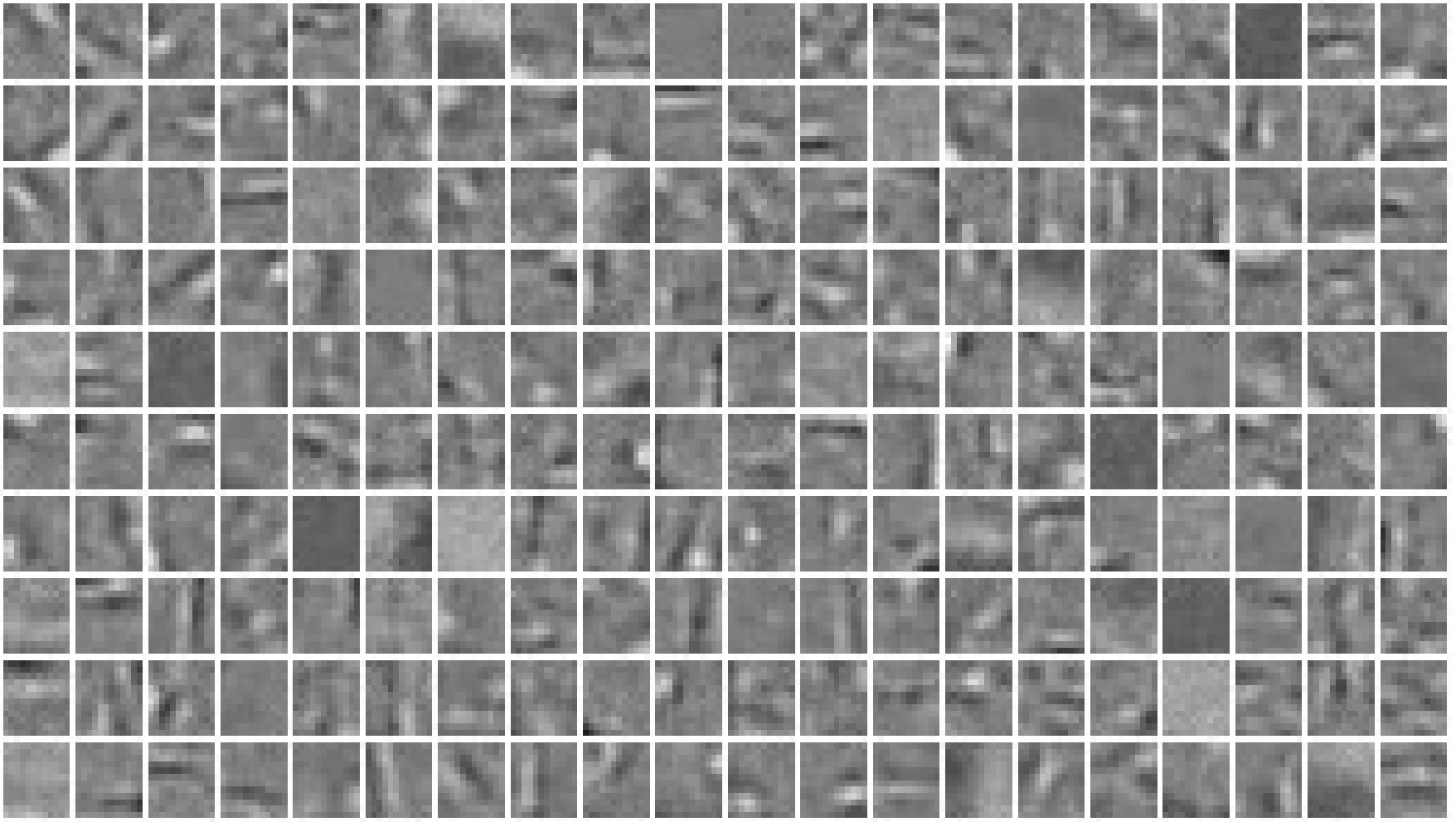
Natural image patches - Forest



Forest data set

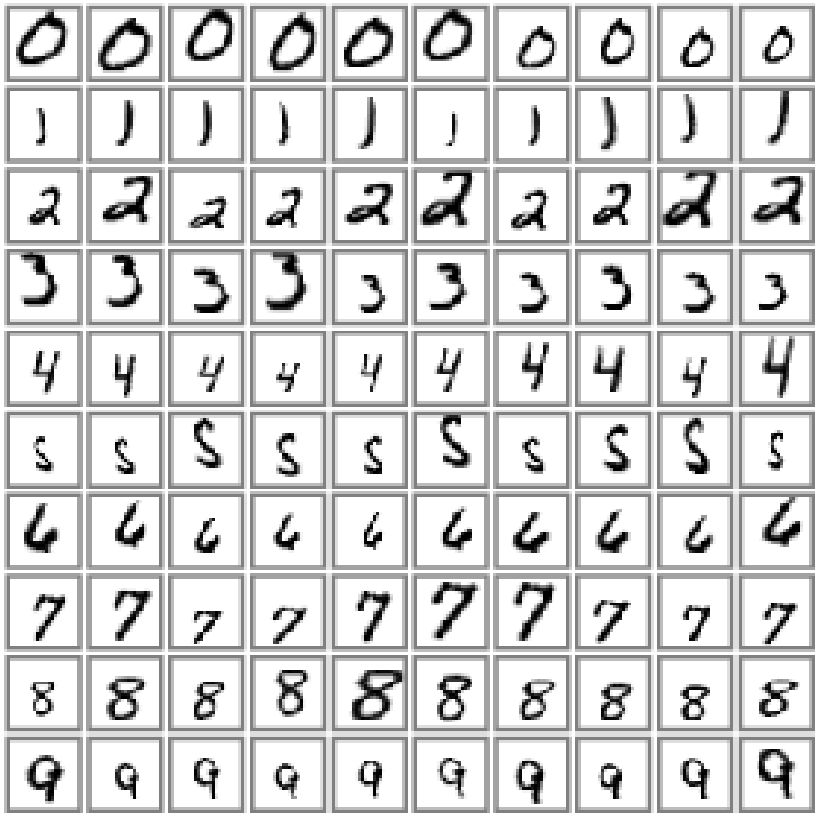
- 100,000 12x12 patches
- 200 units in the code
- η 0.02
- β 1
- learning rate 0.001
- L1, L2 regularizer 0.001
- fast convergence: < 30min.

Natural image patches - Forest



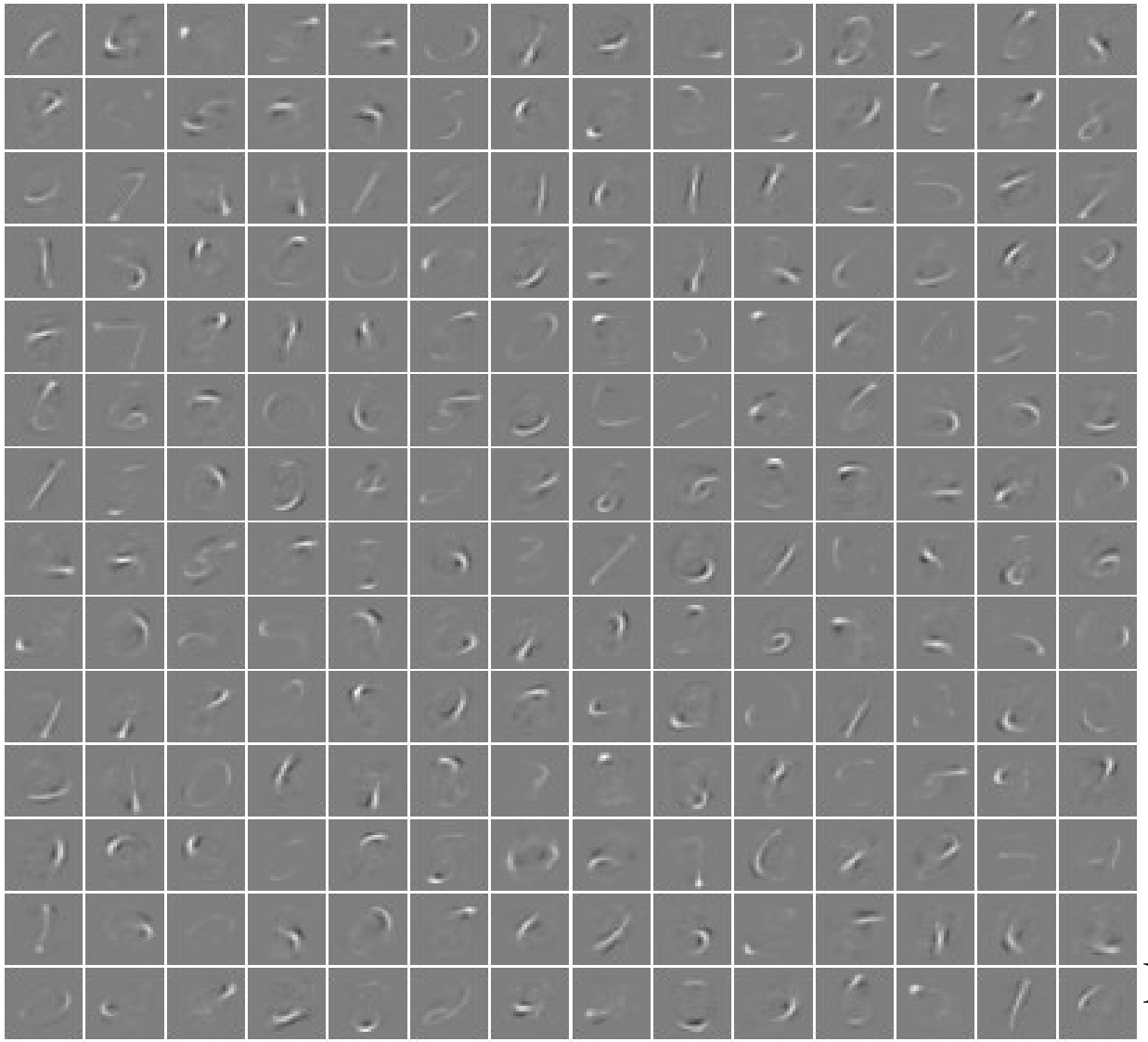
200 decoder filters (reshaped columns of matrix $\mathbf{W_d}$)

MNIST Dataset



- Handwritten Digit Dataset MNIST: 60,000 training samples, 10,000 test samples

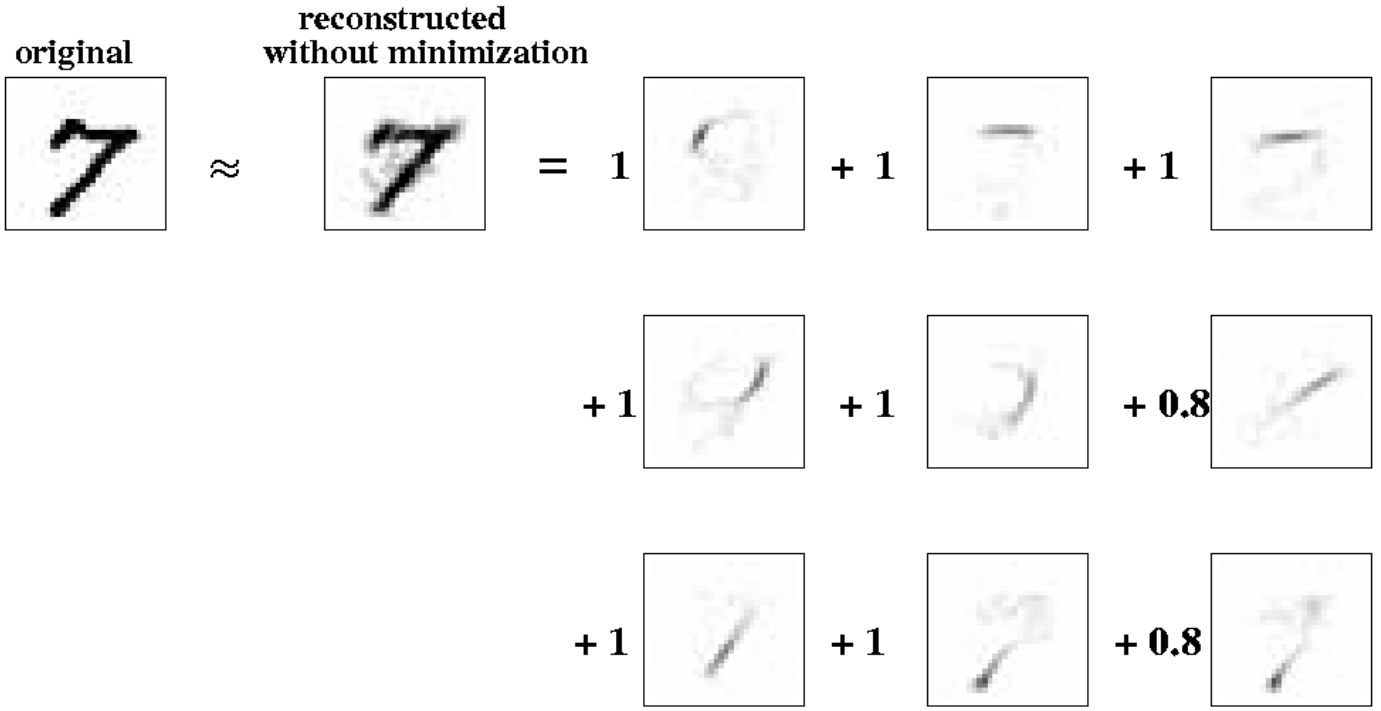
Training on handwritten digits

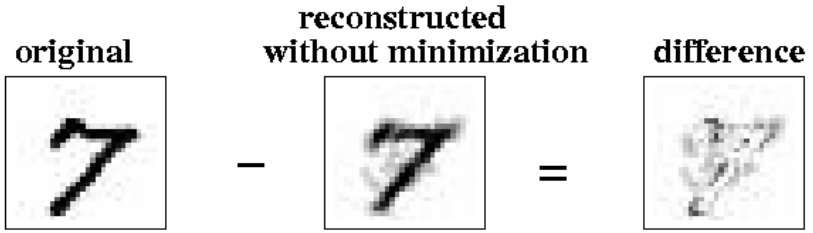


- 60,000 28x28 images
- 196 units in the code
- η 0.01
- β 1
- learning rate 0.001
- L1, L2 regularizer 0.005

Encoder *direct* filters

Handwritten digits - MNIST





forward propagation through encoder and decoder

after training there is no need to minimize in code space

Unsupervised Training of Convolutional Filters

CLASSIFICATION EXPERIMENTS

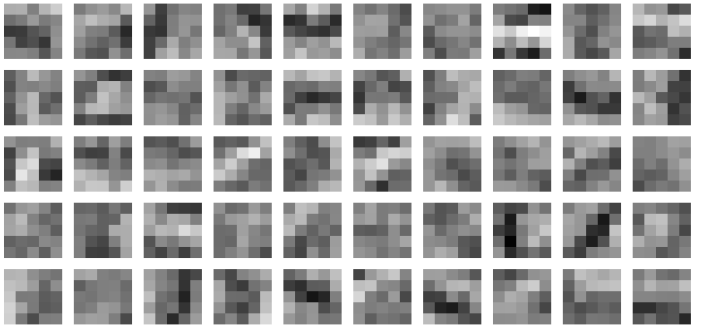
IDEA: improving supervised learning by pre-training with the unsupervised method (*)

sparse representations & *lenet6* (1->50->50->200->10)

- The baseline: *lenet6* initialized randomly

Test error rate: **0.70%**. Training error rate: 0.01%.

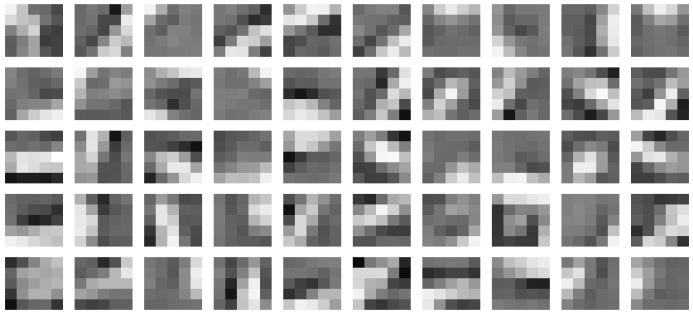
supervised filters in first conv. layer



- *Experiment 1*
  - Train on 5x5 patches to find 50 features
  - Use the scaled filters in the encoder to initialize the kernels in the first convolutional layer

Test error rate: 0.60%. Training error rate: 0.00%.

unsupervised filters in first conv. layer



- *Experiment 2*
  - Same as experiment 1, but training set augmented by elastically distorted digits (random initialization gives test error rate equal to **0.49%**).

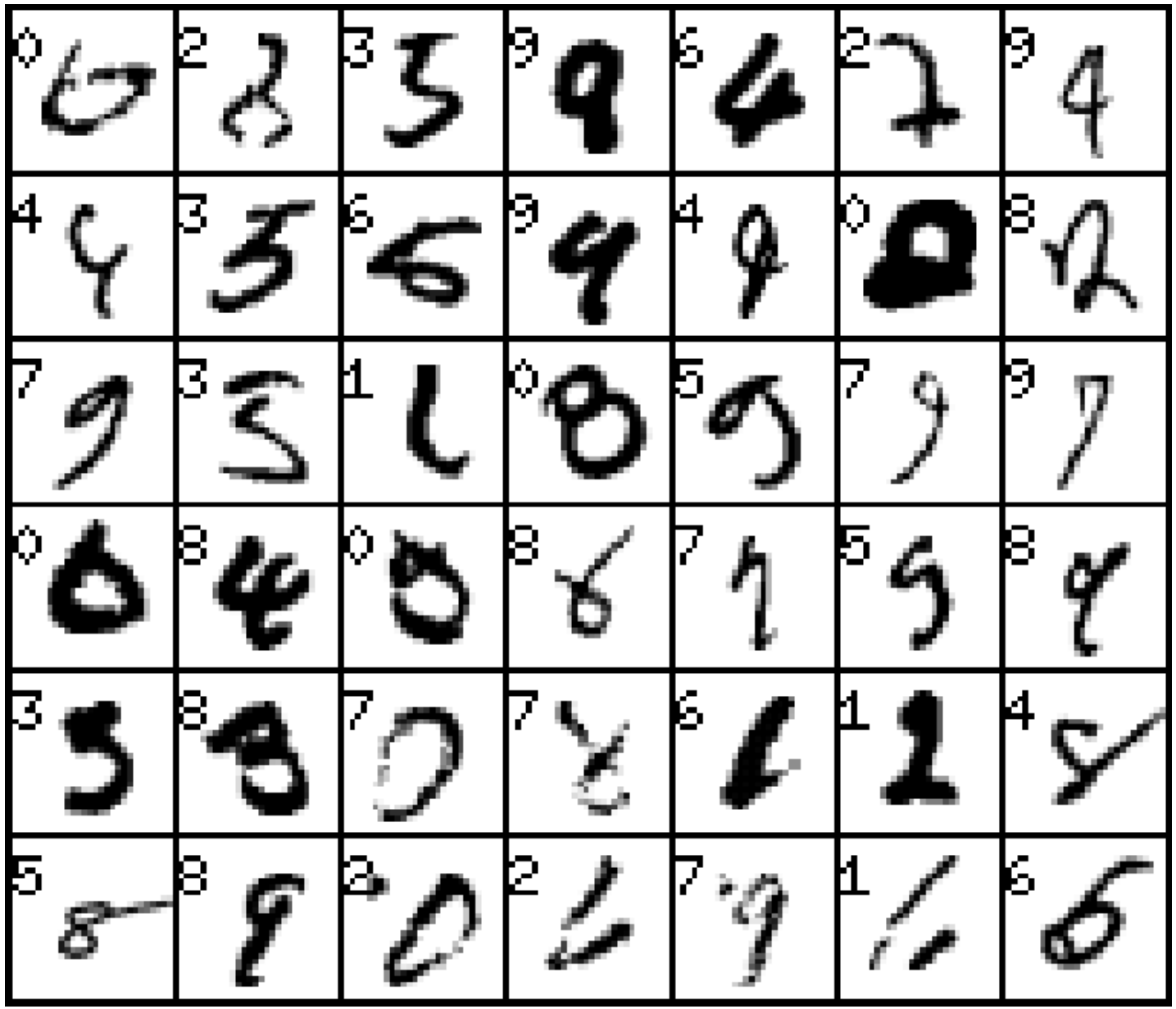
Test error rate: 0.39%. Training error rate: 0.23%.

(*)[Hinton, Osindero, Teh "*A fast learning algorithm for deep belief nets*" Neural Computaton 2006]

Best Results on MNIST (from raw images: no preprocessing)

CLASSIFIER	DEFORMATION	ERROR	Reference
Knowledge-free methods			
2-layer NN, 800 HU, CE		1.60	Simard et al., ICDAR 2003
3-layer NN, 500+300 HU, CE, reg		1.53	Hinton, in press, 2005
SVM, Gaussian Kernel		1.40	Cortes 92 + Many others
Unsupervised Stacked RBM + backprop		0.95	Hinton, Neur Comp 2006
Convolutional nets			
Convolutional net LeNet-5,		0.80	Ranzato et al. NIPS 2006
Convolutional net LeNet-6,		0.70	Ranzato et al. NIPS 2006
Conv. net LeNet-6- + unsup learning		0.60	Ranzato et al. NIPS 2006
Training set augmented with Affine Distortions			
2-layer NN, 800 HU, CE	Affine	1.10	Simard et al., ICDAR 2003
Virtual SVM deg-9 poly	Affine	0.80	Scholkopf
Convolutional net, CE	Affine	0.60	Simard et al., ICDAR 2003
Training et augmented with Elastic Distortions			
2-layer NN, 800 HU, CE	Elastic	0.70	Simard et al., ICDAR 2003
Convolutional net, CE	Elastic	0.40	Simard et al., ICDAR 2003
Conv. net LeNet-6- + unsup learning	Elastic	0.39	Ranzato et al. NIPS 2006

MNIST Errors (0.42% error)



Denoising

original image

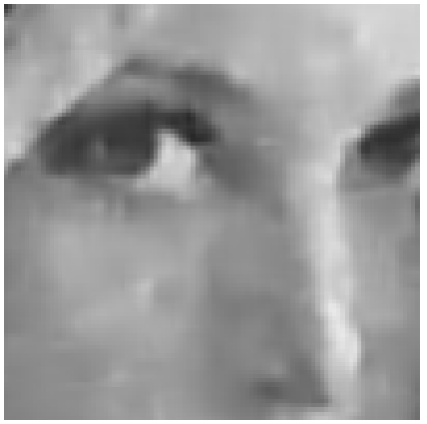
noisy imagePSNR 14.15dB

(std. dev. noise 50)

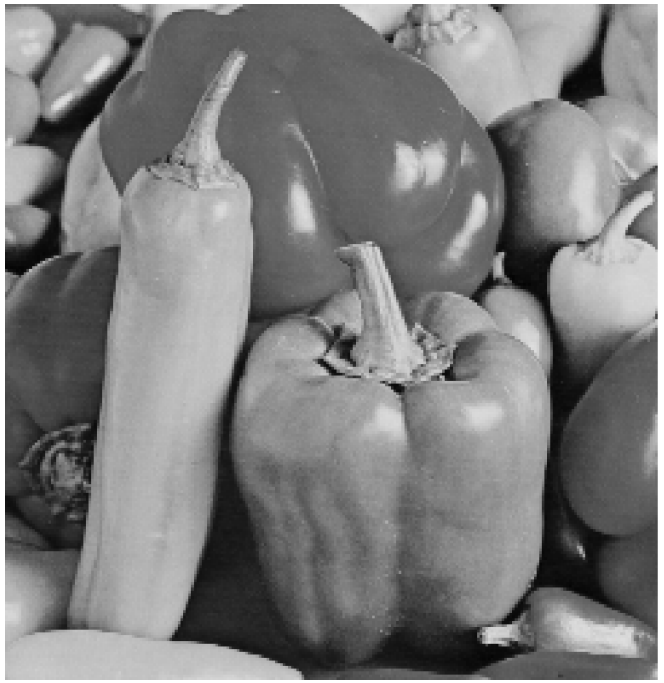
denoised image

PSNR 27.88dB

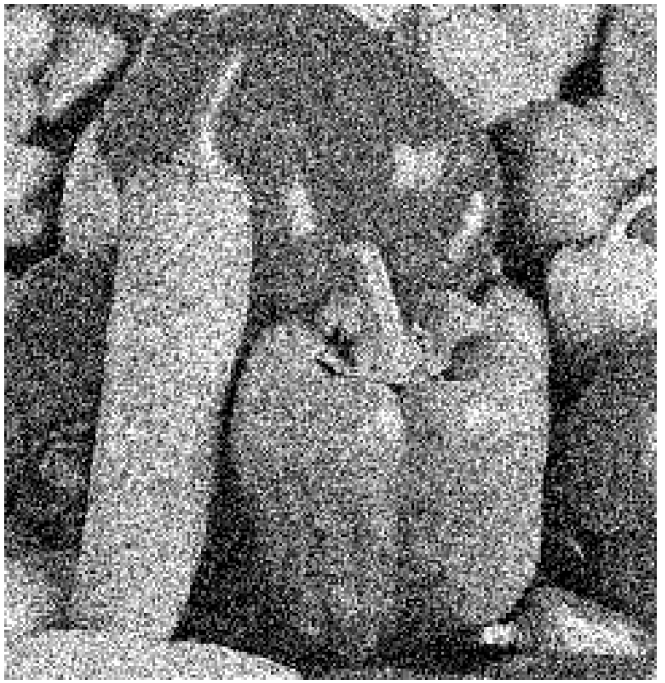
ZOOM ->



Denoising



original image



noisy image: PSNR 14.15dB

(std. dev. noise 50)

denoised image

PSNR 26.50dB

ZOOM ->

Denoising

s.d. / PSNR	*Lena*				*Barbara*				*Boat*				*House*				*Peppers*			
50 / 14.15	27.86	**28.61**	27.79	26.49	23.46	**25.48**	25.47	23.15	26.02	**26.38**	25.95	24.53	27.85	**28.26**	27.95	26.74	**26.35**	25.90	26.13	24.52
75 / 10.63	25.97	**26.84**	25.80	24.13	22.46	**23.65**	23.01	21.36	24.31	**24.79**	23.98	22.48	25.77	**26.41**	25.22	24.13	**24.56**	24.00	23.69	21.68
100 / 8.13	24.49	**25.64**	24.46	21.87	21.77	**22.61**	21.89	19.77	23.09	**23.75**	22.81	20.80	24.20	**25.11**	23.71	21.66	**23.04**	22.66	21.75	19.60

Comparison between:

- our method [first column]
- Portilla et al. IEEE Trans. Image Processing (2003) [second column]
- Elad and Aharon CVPR 2006 [third column]
- Roth and Black CVPR 2005 [fourth column]

Part 3: Training "Deep Belief Networks", Learning Invariant Feature Hierarchies

- **Why do we need Deep Learning?**
  - "scaling learning algorithms towards AI" [Bengio and LeCun 2007]
- **Deep Belief Networks, Deep Learning**
  - Stacked RBM [Hinton, Osindero, and Teh, Neural Comp 2006]
  - Stacked autoencoders [Bengio et al. NIPS 2006]
  - Stacked sparse features [Ranzato & al., NIPS 2006]
- **Unsupervised Learning of Invariant Feature Hierarchies**
  - learning features for Caltech-101 [Ranzato et al. CVPR 2006]

Why do we need "Deep" Architectures?

[Bengio & LeCun 2007]

- **Conjecture: we won't solve the perception problem without solving the problem of learning in deep architectures [Hinton]**
  - Neural nets with lots of layers
  - Deep belief networks
  - Factor graphs with a "Markov" structure
- **We will not solve the perception problem with kernel machines**
  - Kernel machines are glorified template matchers
  - You can't handle complicated invariances with templates (you would need too many templates)
- **Many interesting functions are "deep"**
  - Any function can be approximated with 2 layers (linear combination of non-linear functions)
  - But many interesting functions a more efficiently represented with multiple layers
  - Stupid examples: binary addition

Non-Linear Dimensionality Reduction

- [Hinton and Salakhutdinov, Science 2006]

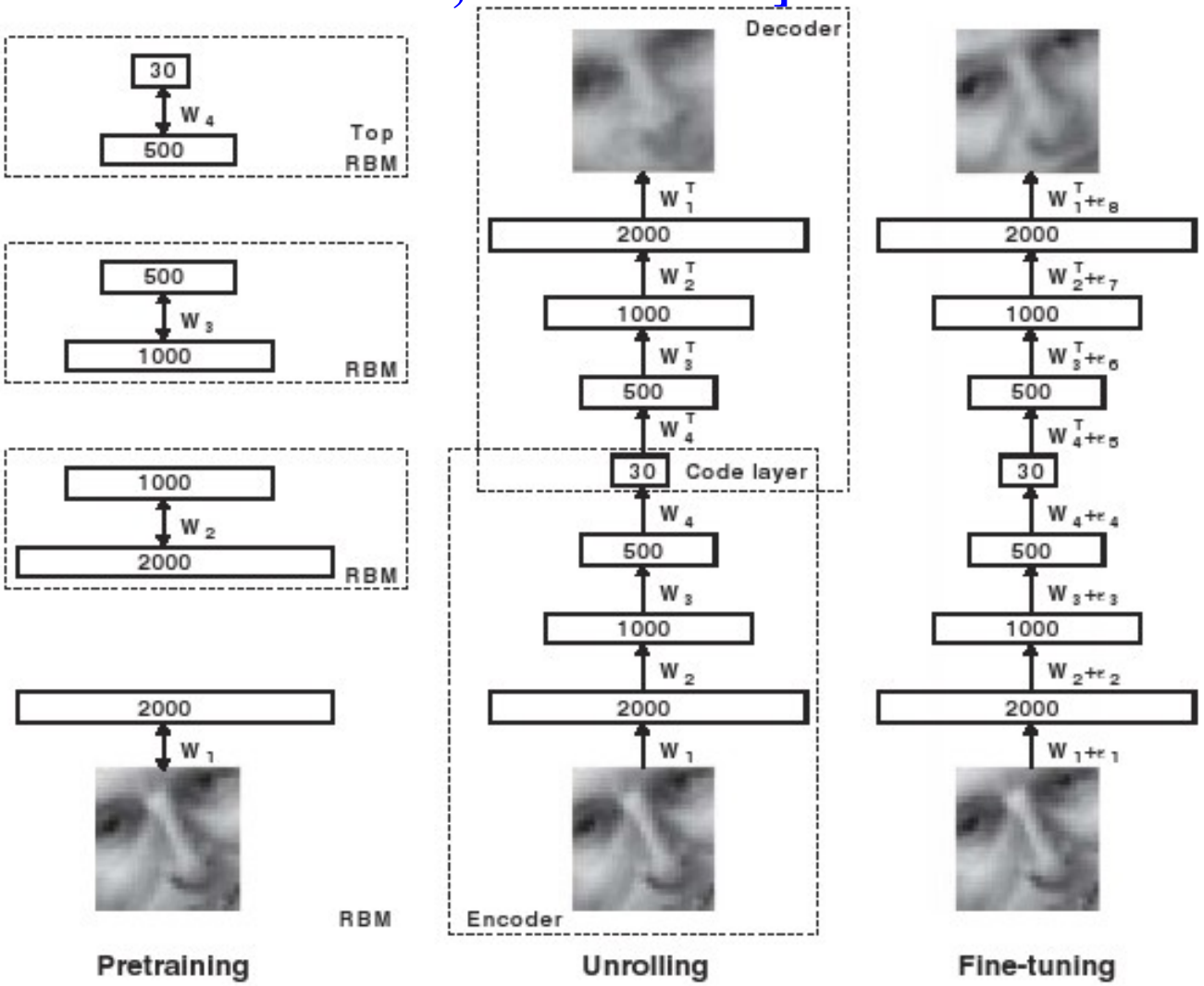


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

Non-Linear Dimensionality Reduction

- [Hinton and Salakhutdinov, Science 2006]

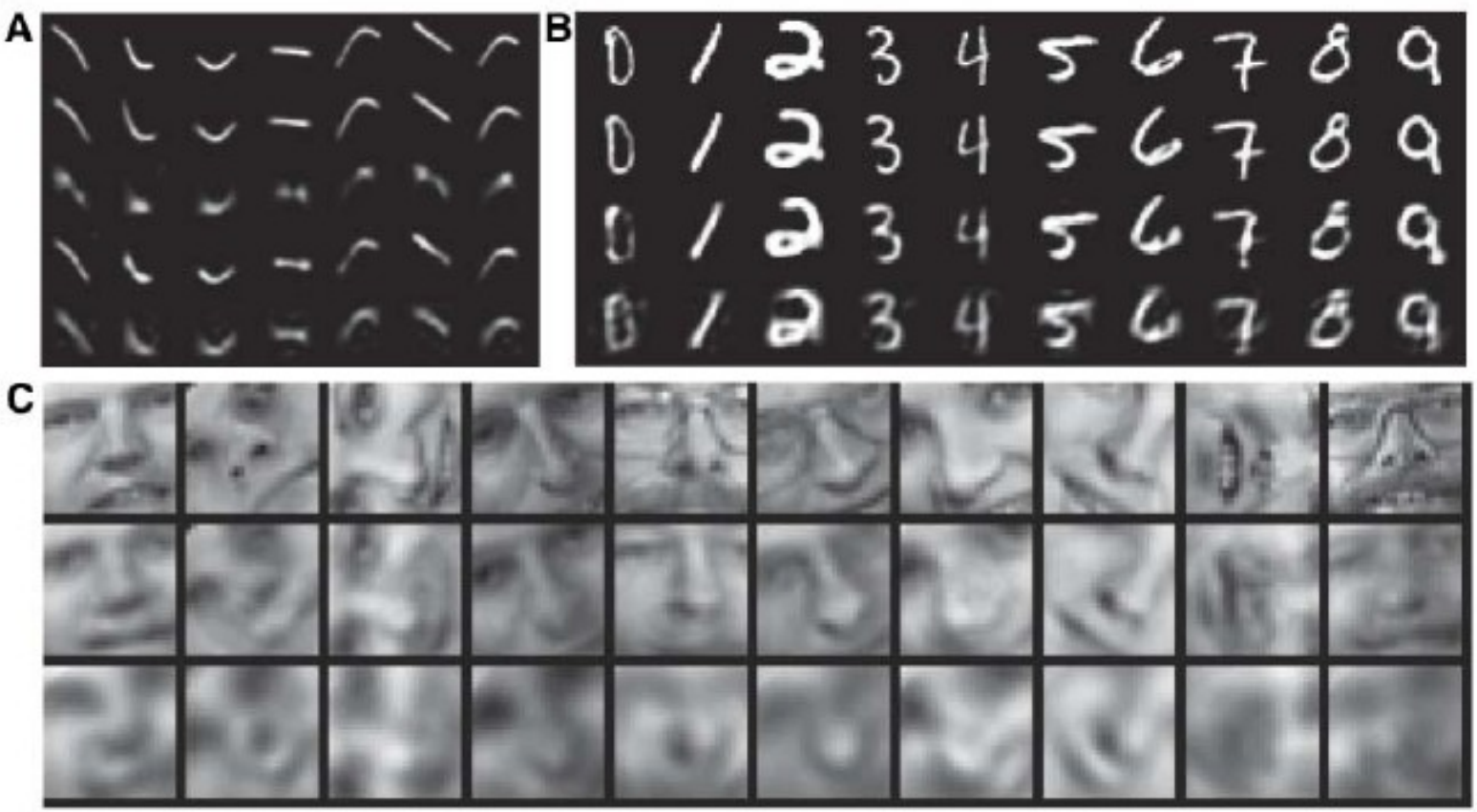
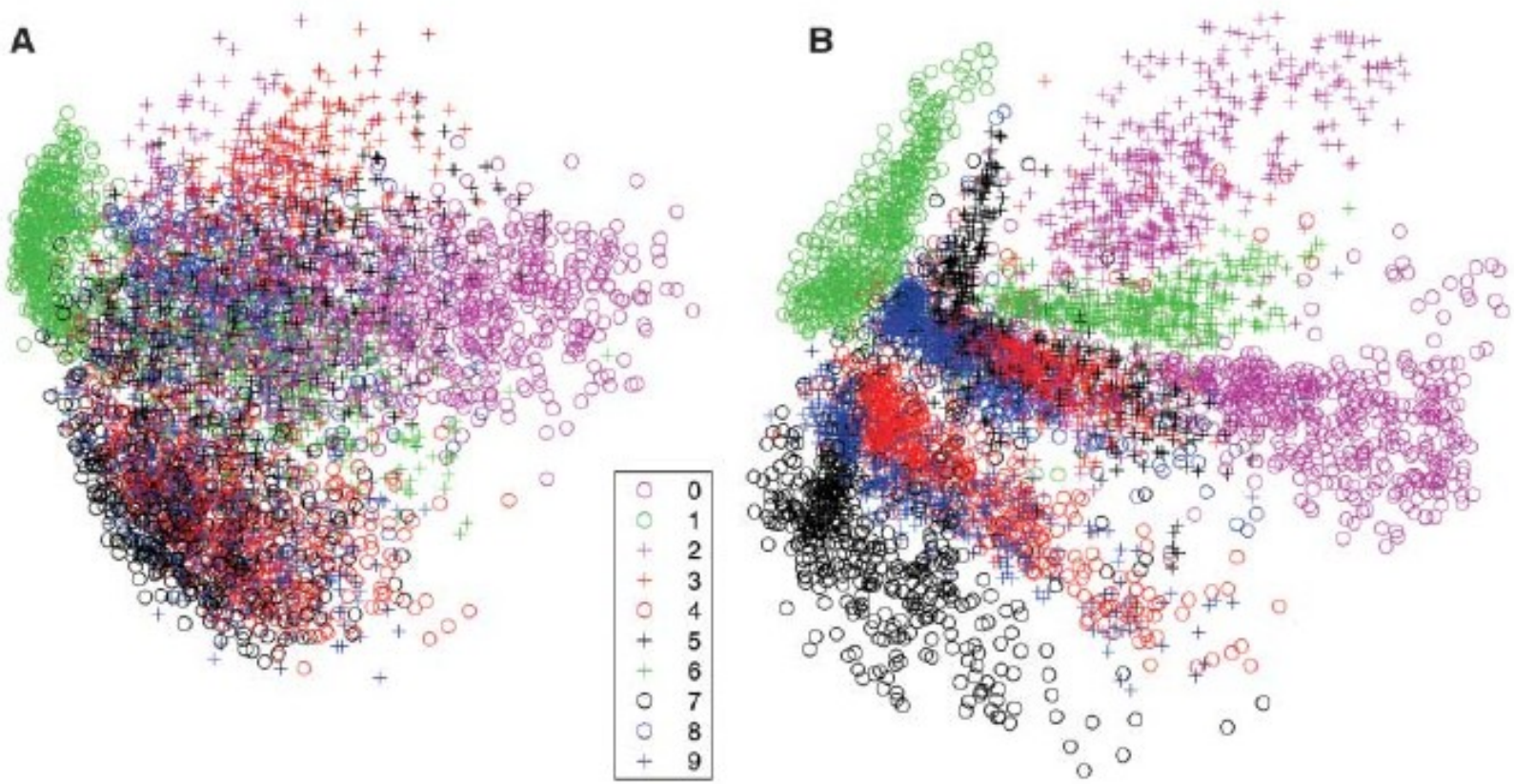


Fig. 2. (**A**) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (*8*) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (**B**) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87. (**C**) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.

Non-Linear Dimensionality Reduction

- [Hinton and Salakhutdinov, Science 2006]

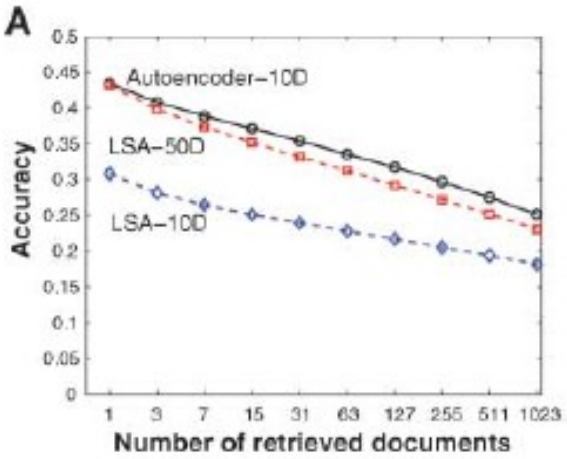
Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

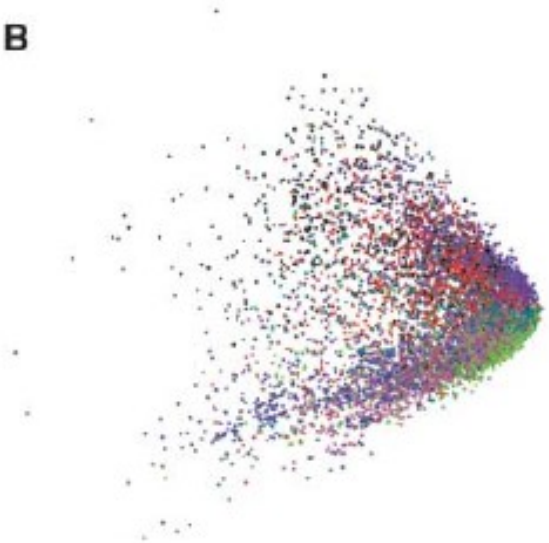


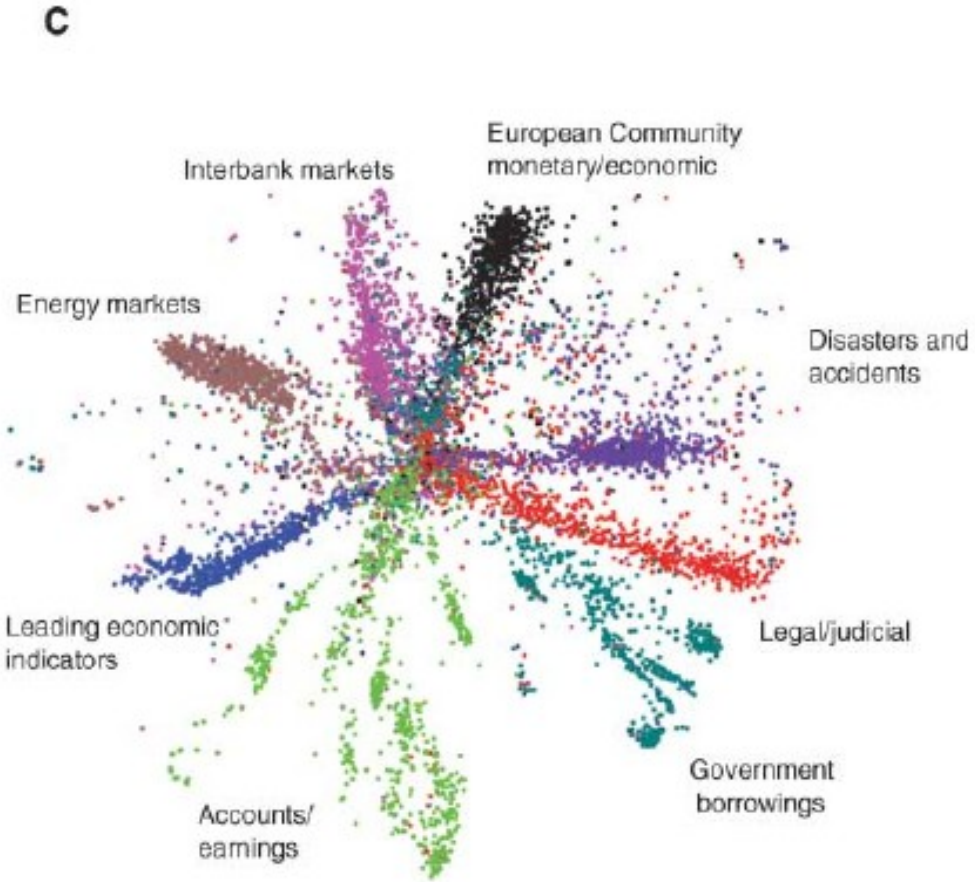
Non-Linear Dimensionality Reduction

- [Hinton and Salakhutdinov, Science 2006]

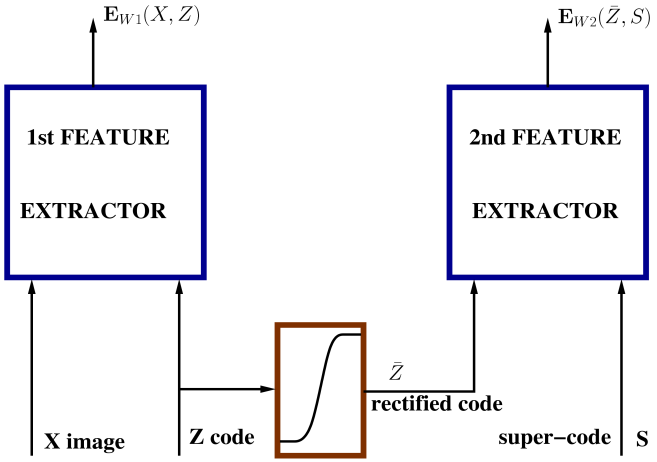
Fig. 4. (**A**) The fraction of retrieved documents in the same class as the query when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (**B**) The codes produced by two-dimensional LSA. (**C**) The codes produced by a 2000-500-250-125-2 autoencoder.







Training a Feature Hierarchy on MNIST

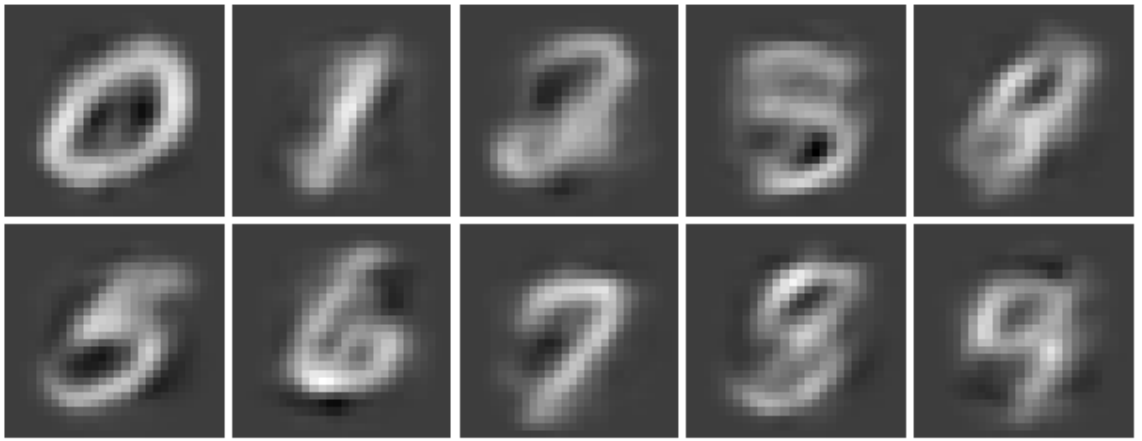


- Each super-code unit in S represents a digit prototype

NOTE: no label was used, fully unsupervised system

Q: What do the super-units in S represent?

reconstruction of super-code units



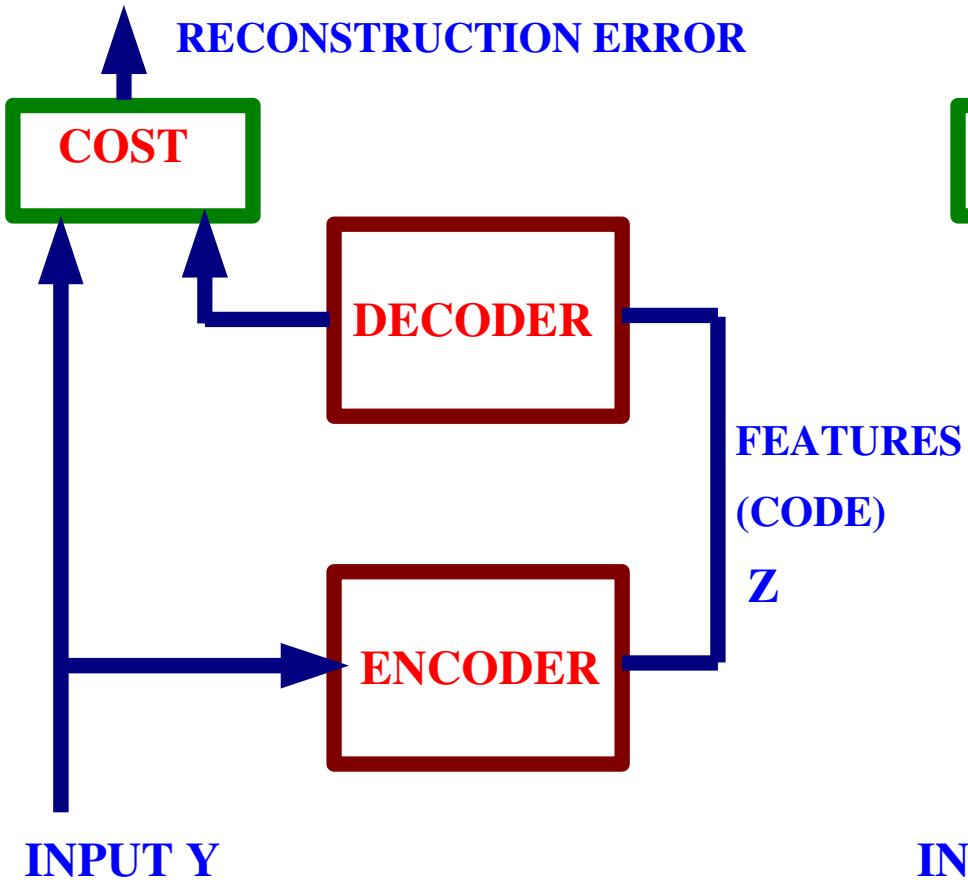
- activate only one unit in S
- propagate the code S through decoder in second machine -> get $\bar{Z}$
- propagate the code $\bar{Z}$ through decoder in first machine -> get image

Training The Layers of a Convolutional Net Unsupervised

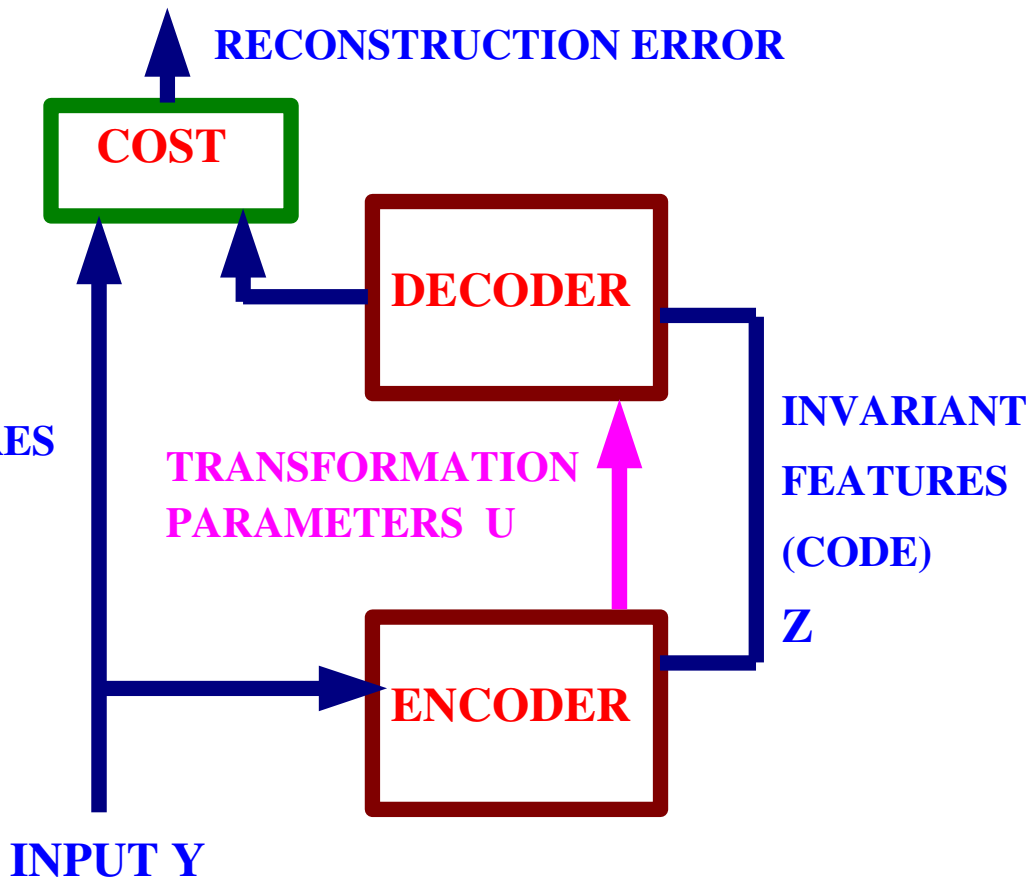
- Extract windows from the MNIST images
- Train the sparse encoder/decoder on those windows
- Use the resulting encoder weights as the convolution kernels of a convolution network
- Repeat the process for the second layer
- Train the resulting network supervised.

Learning Invariant Feature Hierarchies

Learning Shift Invariant Features



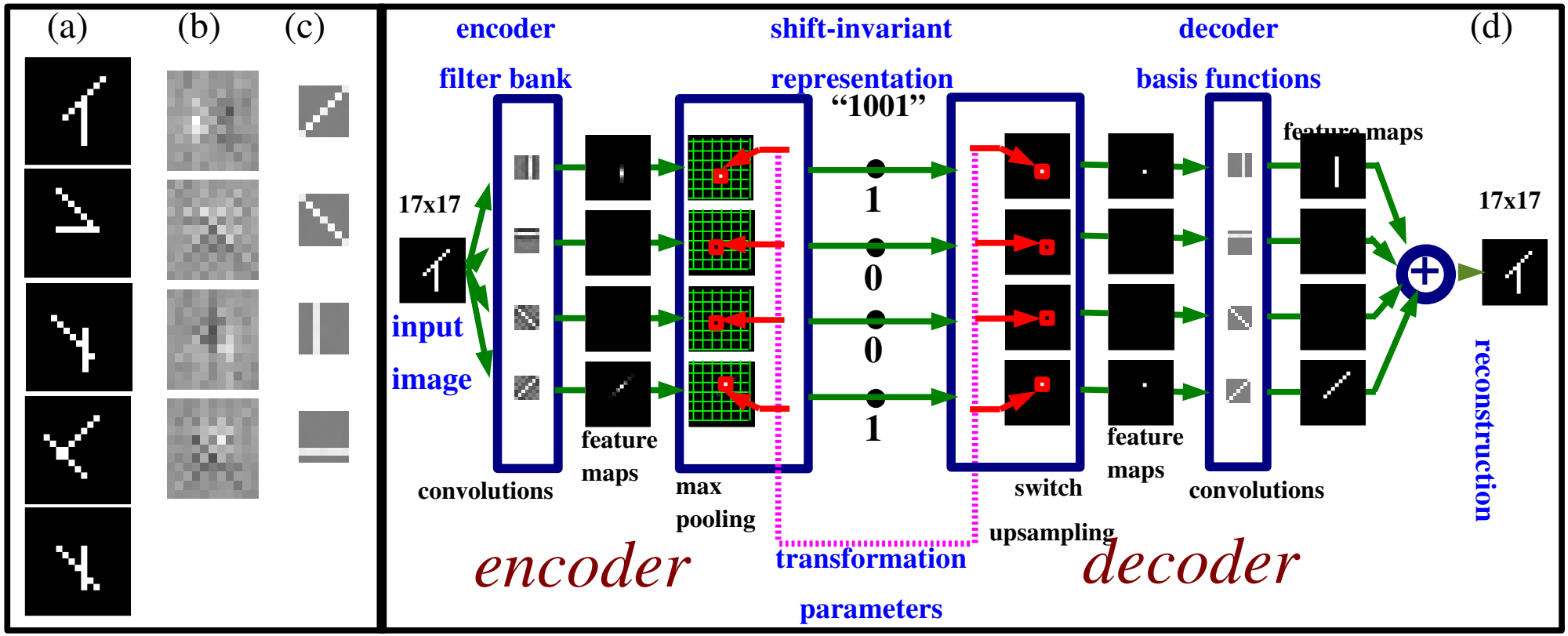
Standard Feature Extractor



Invariant Feature Extractor

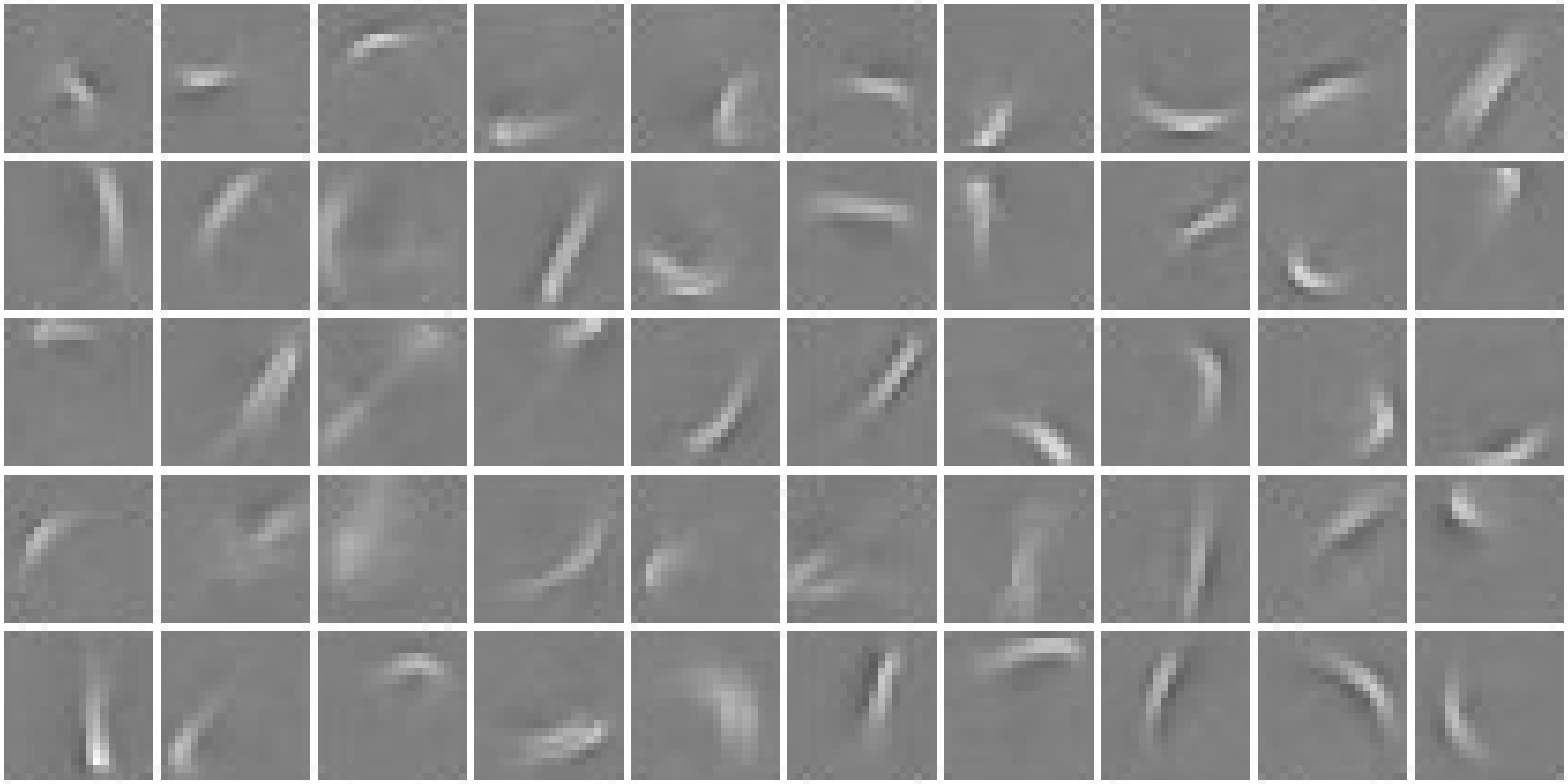
Learning Invariant Feature Hierarchies

- Learning Shift Invariant Features



Shift Invariant Global Features on MNIST

- **Learning 50 Shift Invariant Global Features on MNIST:**
  - 50 filters of size 20x20 movable in a 28x28 frame (81 positions)
  - movable strokes!

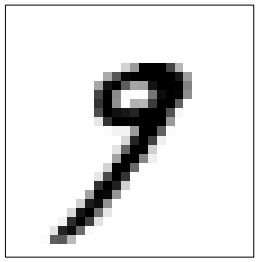
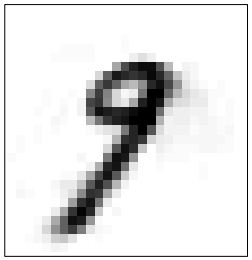


Example of Reconstruction

- Any character can be reconstructed as a linear combination of a small number of basis functions.

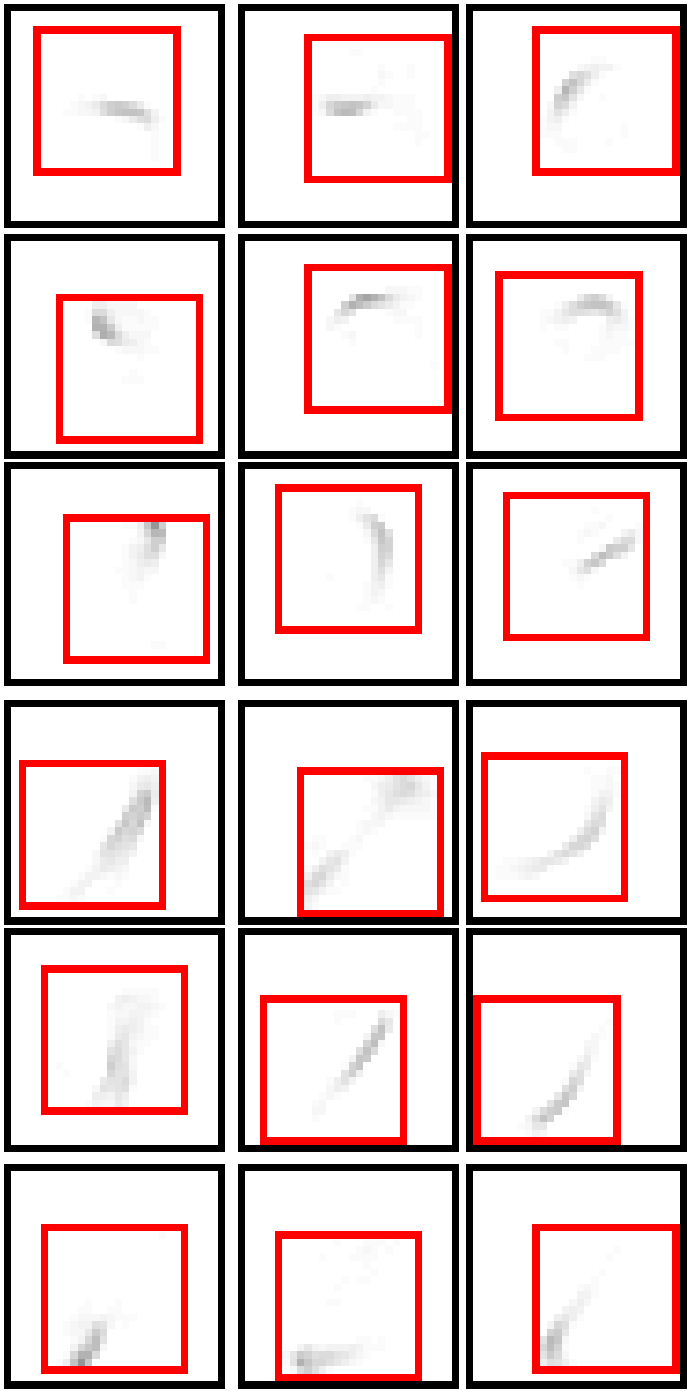
ORIGINAL DIGIT

RECONS-TRUCTION

$\sum$  $\cong$  $=$

ACTIVATED DECODER BASIS FUNCTIONS

(in feed-back layer)



red squares: decoder bases

Learning Invariant Filters in a Convolutional Net

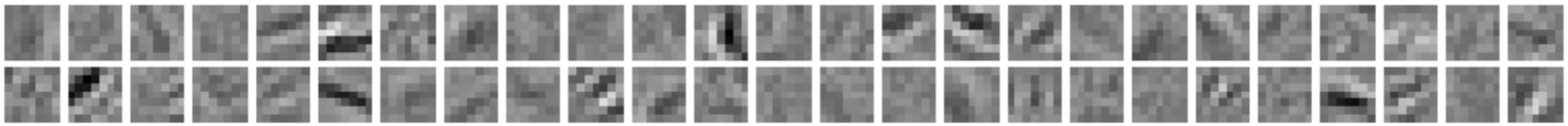


Figure 1: 50 7x7 filters in the first convolutional layer that were learned by the network trained supervised from *random* initial conditions with 600K digits.

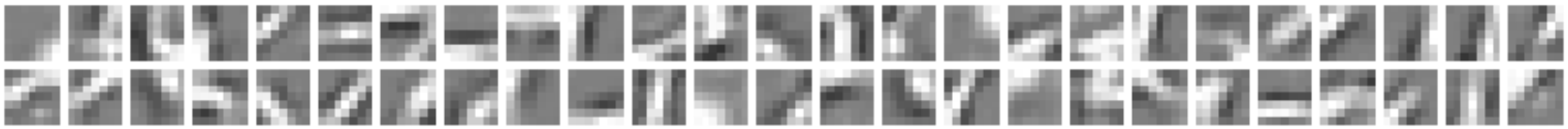


Figure 2: 50 7x7 filters that were learned by the unsupervised method (on 60K digits), and that are used to initialize the first convoltional layer of the network.

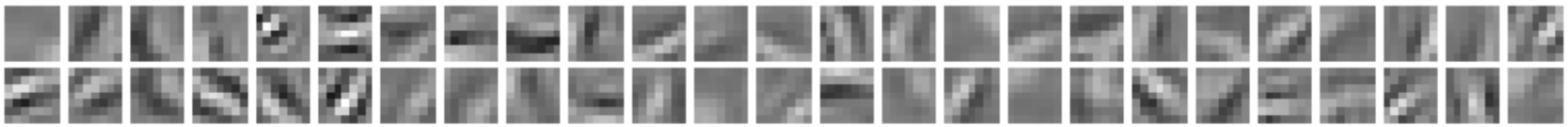
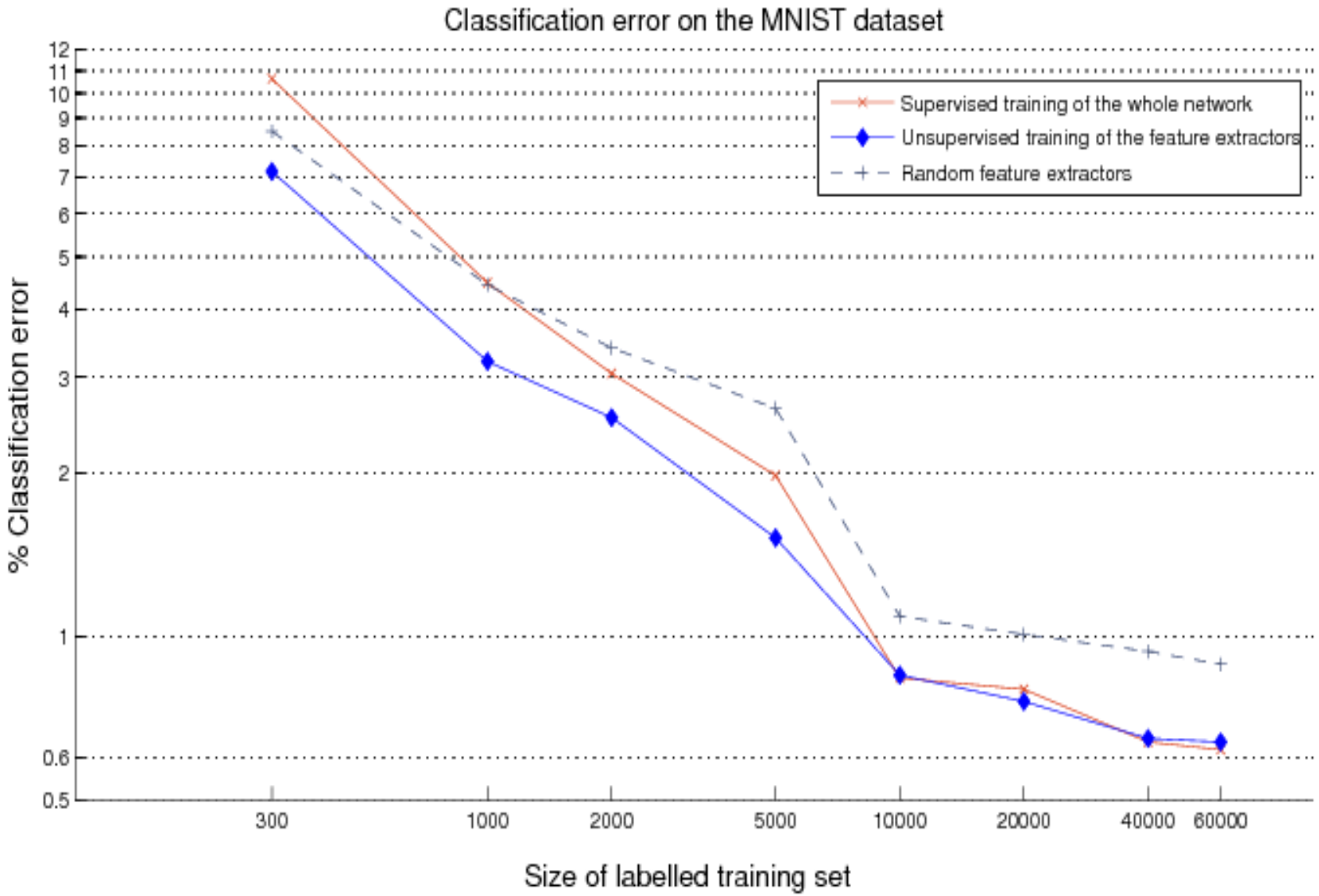


Figure 3: 50 7x7 filters in the first convolutional layer that were learned by the network trained supervised from the initial conditions given by the *unsupervised method* (see fig.2) with 600K digits.

Influence of Number of Training Samples

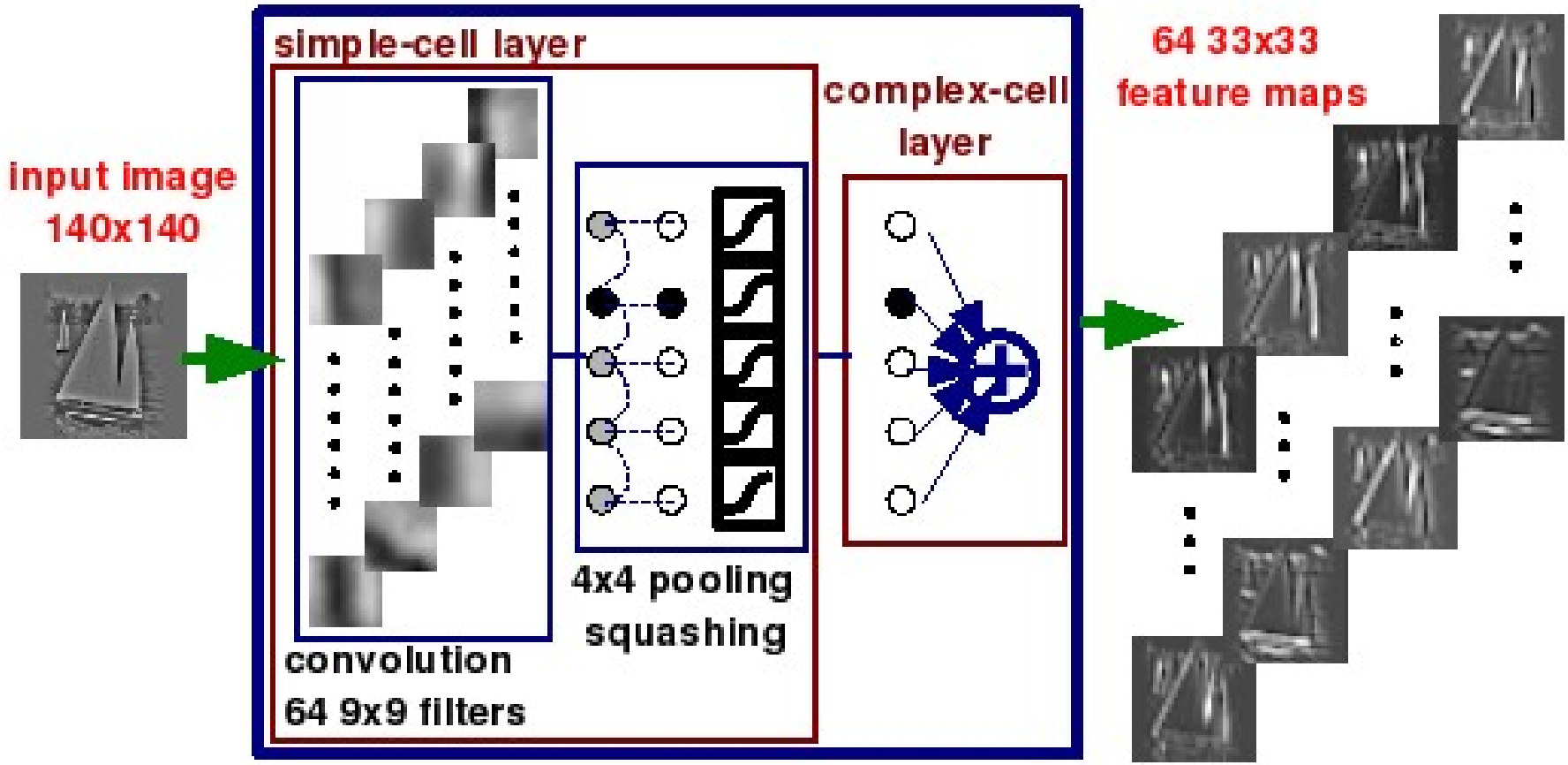


Generic Object Recognition: 101 categories + background

- **Caltech-101 dataset: 101 categories**
  - accordion airplanes anchor ant barrel bass beaver binocular bonsai brain brontosaurus buddha butterfly camera cannon car_side ceiling_fan cellphone chair chandelier cougar_body cougar_face crab crayfish crocodile crocodile_head cup dalmatian dollar_bill dolphin dragonfly electric_guitar elephant emu euphonium ewer Faces Faces_easy ferry flamingo flamingo_head garfield gerenuk gramophone grand_piano hawksbill headphone hedgehog helicopter ibis inline_skate joshua_tree kangaroo ketch lamp laptop Leopards llama lobster lotus mandolin mayfly menorah metronome minaret Motorbikes nautilus octopus okapi pagoda panda pigeon pizza platypus pyramid revolver rhino rooster saxophone schooner scissors scorpion sea_horse snoopy soccer_ball stapler starfish stegosaurus stop_sign strawberry sunflower tick trilobite umbrella watch water_lilly wheelchair wild_cat windsor_chair wrench yin_yang
- **Only 30 training examples per category!**
- **A convolutional net trained with backprop (supervised) gets 20% correct recognition.**
- **Training the filters with the sparse invariant unsupervised method**

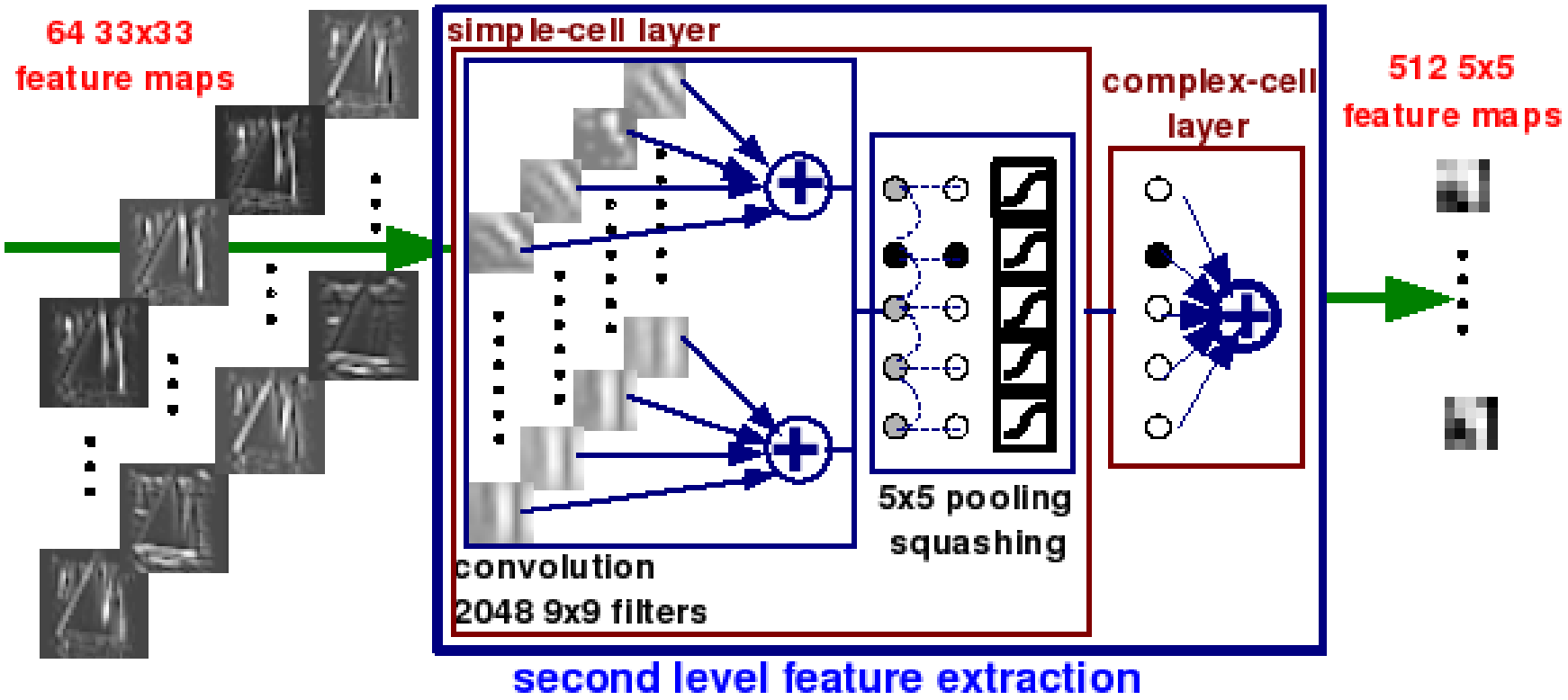
Training the 1st stage filters

- 12x12 input windows (complex cell receptive fields)
- 9x9 filters (simple cell receptive fields)
- 4x4 pooling



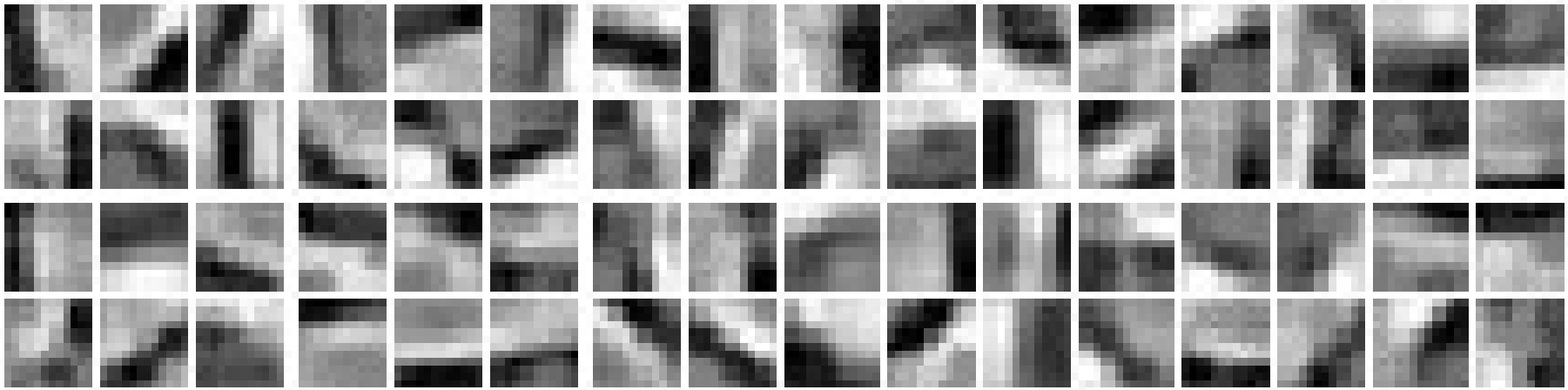
Training the 2nd stage filters

- 13x13 input windows (complex cell receptive fields on 1st features)
- 9x9 filters (simple cell receptive fields)
- Each output feature map combines 4 input feature maps
- 5x5 pooling

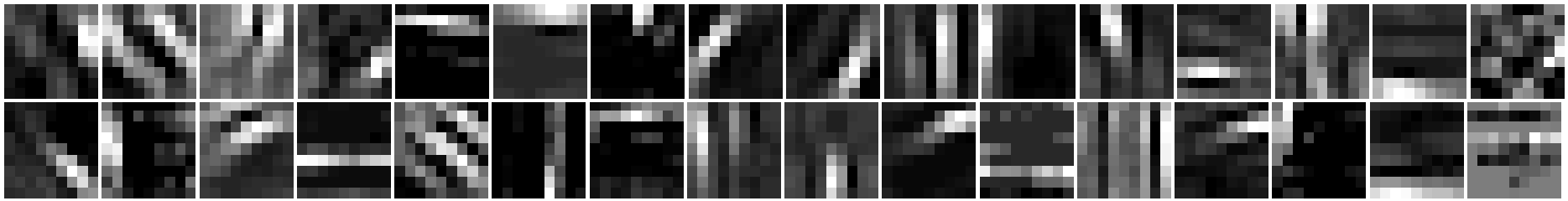


Generic Object Recognition: 101 categories + background

- 9x9 filters at the first level

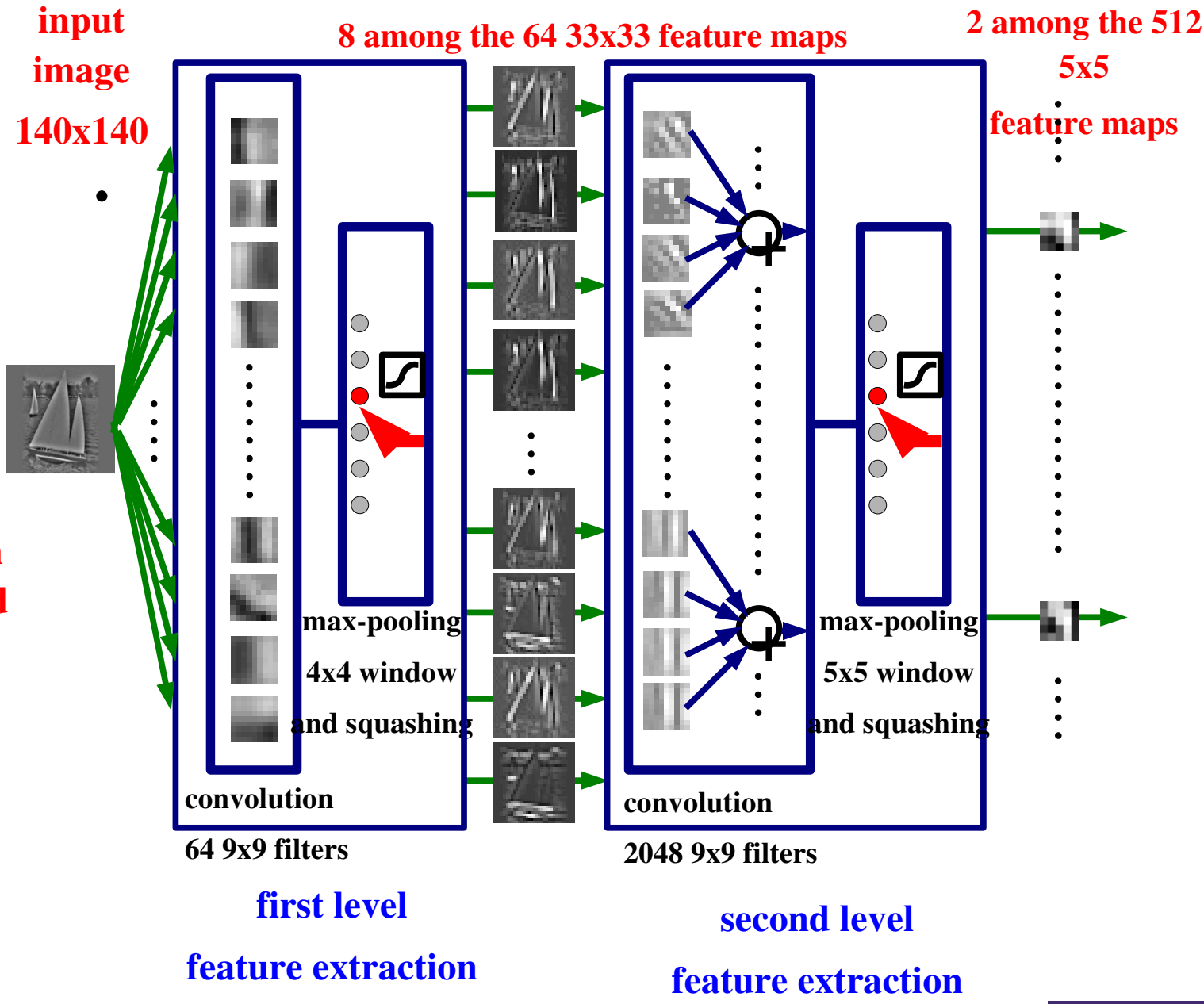


- 9x9 filters at the second level

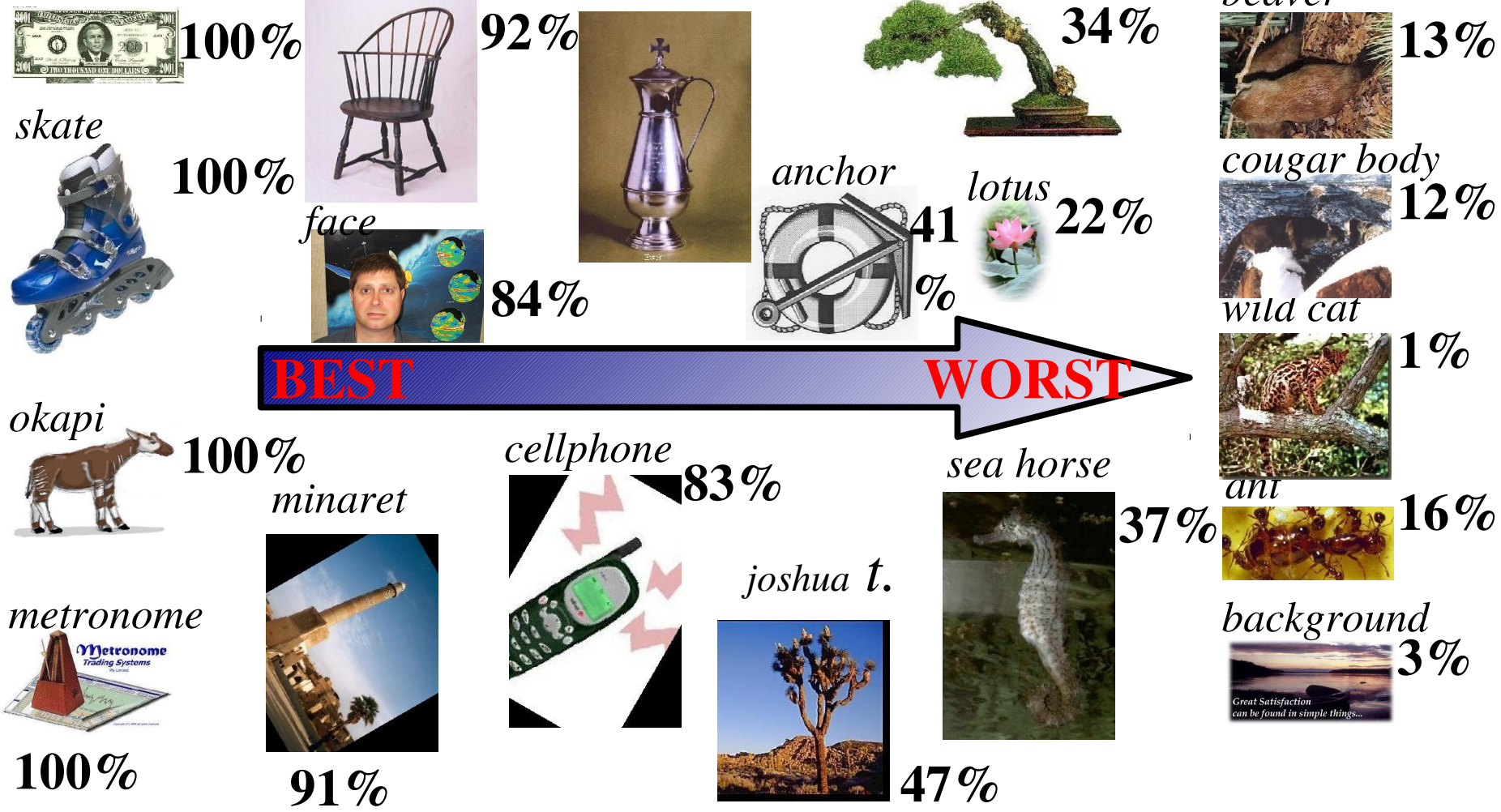


Shift-Invariant Feature Hierarchies on Caltech-101

- 2 layers of filters trained unsupervised
- supervised classifier on top.
- 54% correct on Caltech-101 with 30 examples per class
- 20% correct with purely supervised backprop



Recognition Rate on Caltech 101



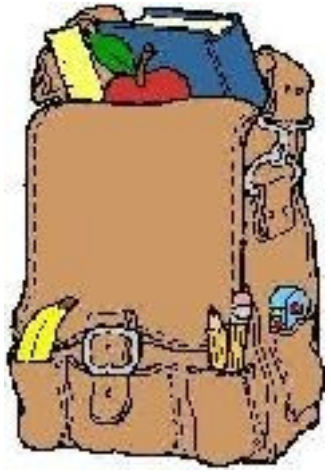
Caltech 256









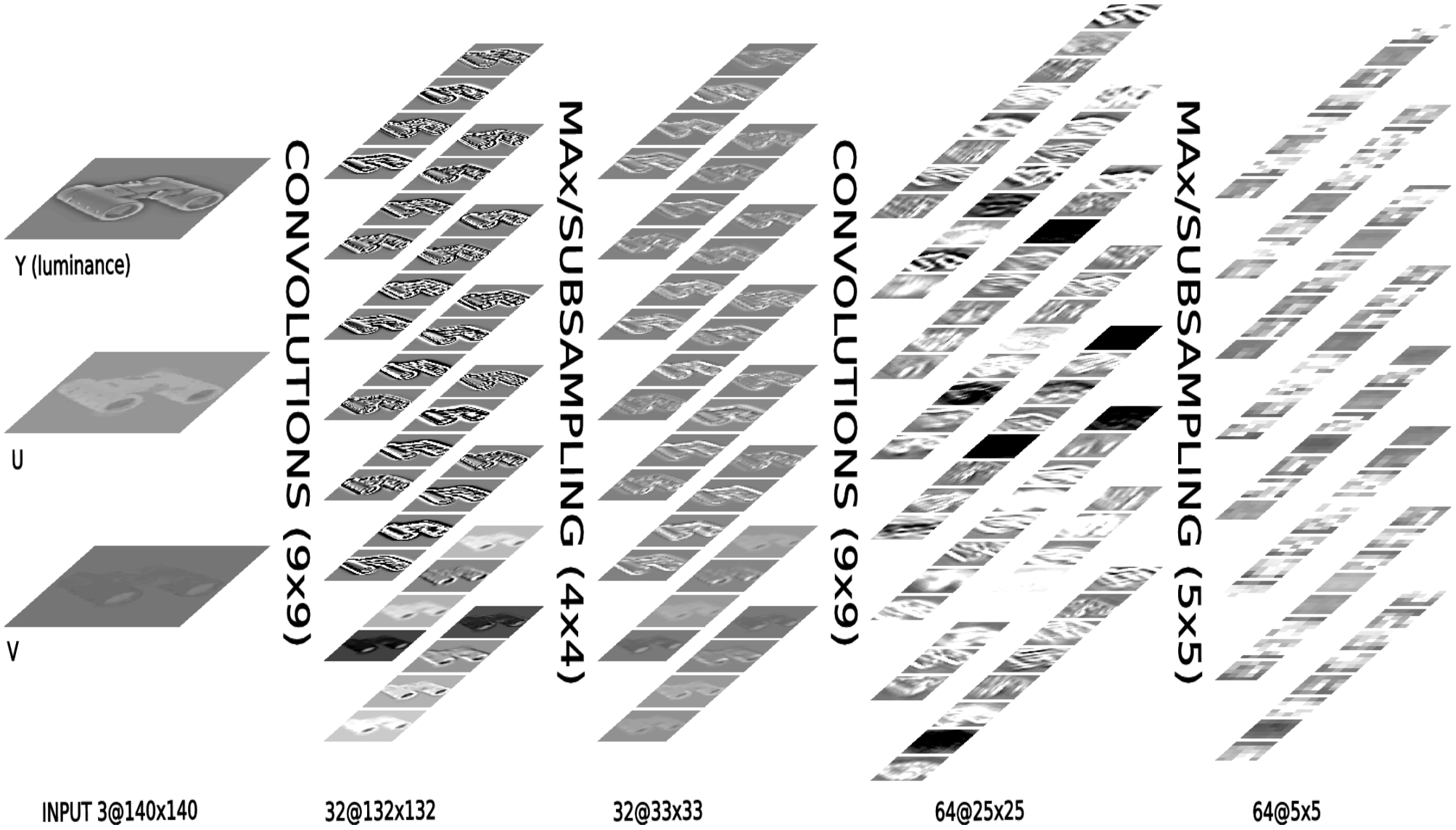




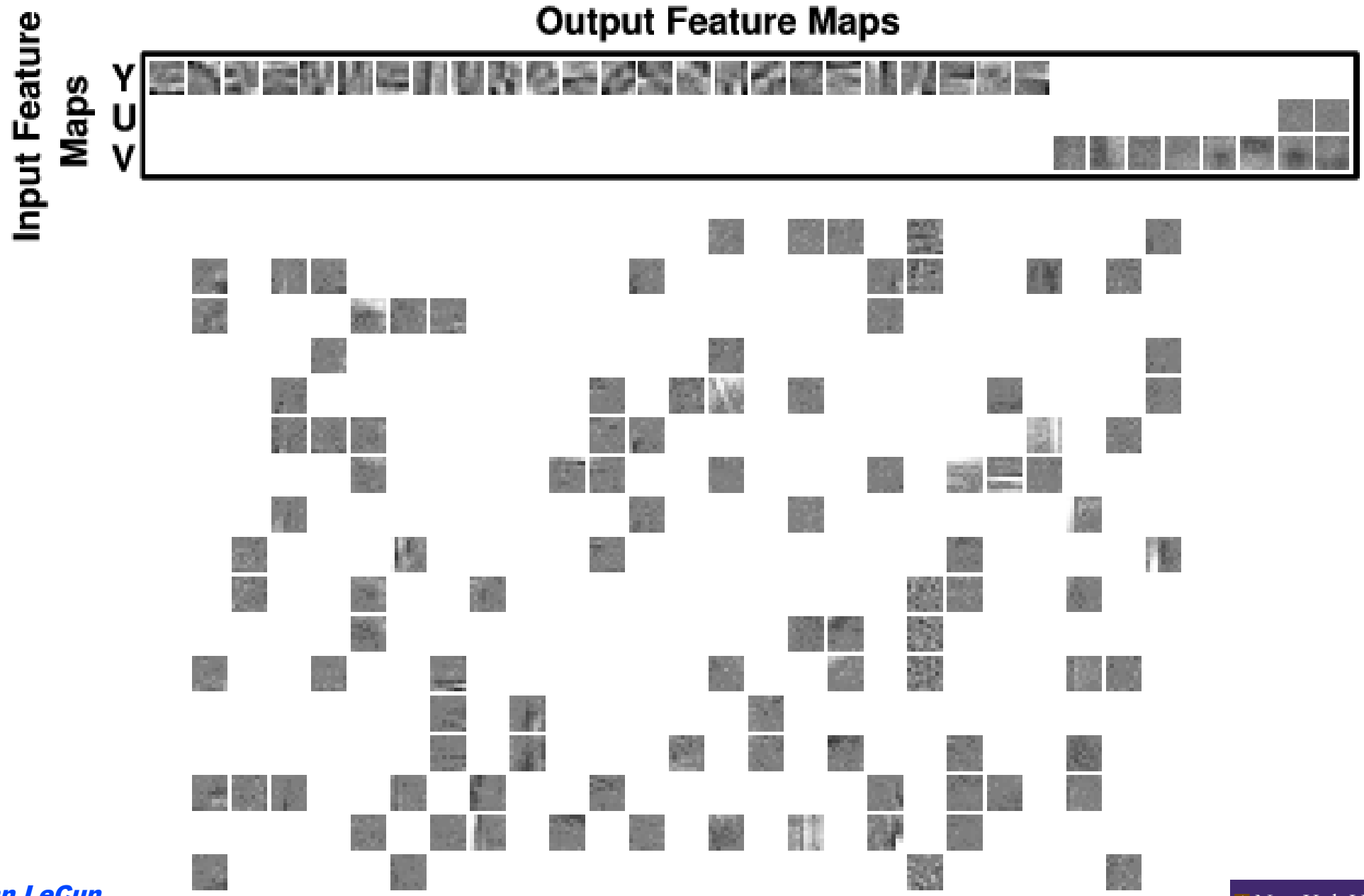





Network

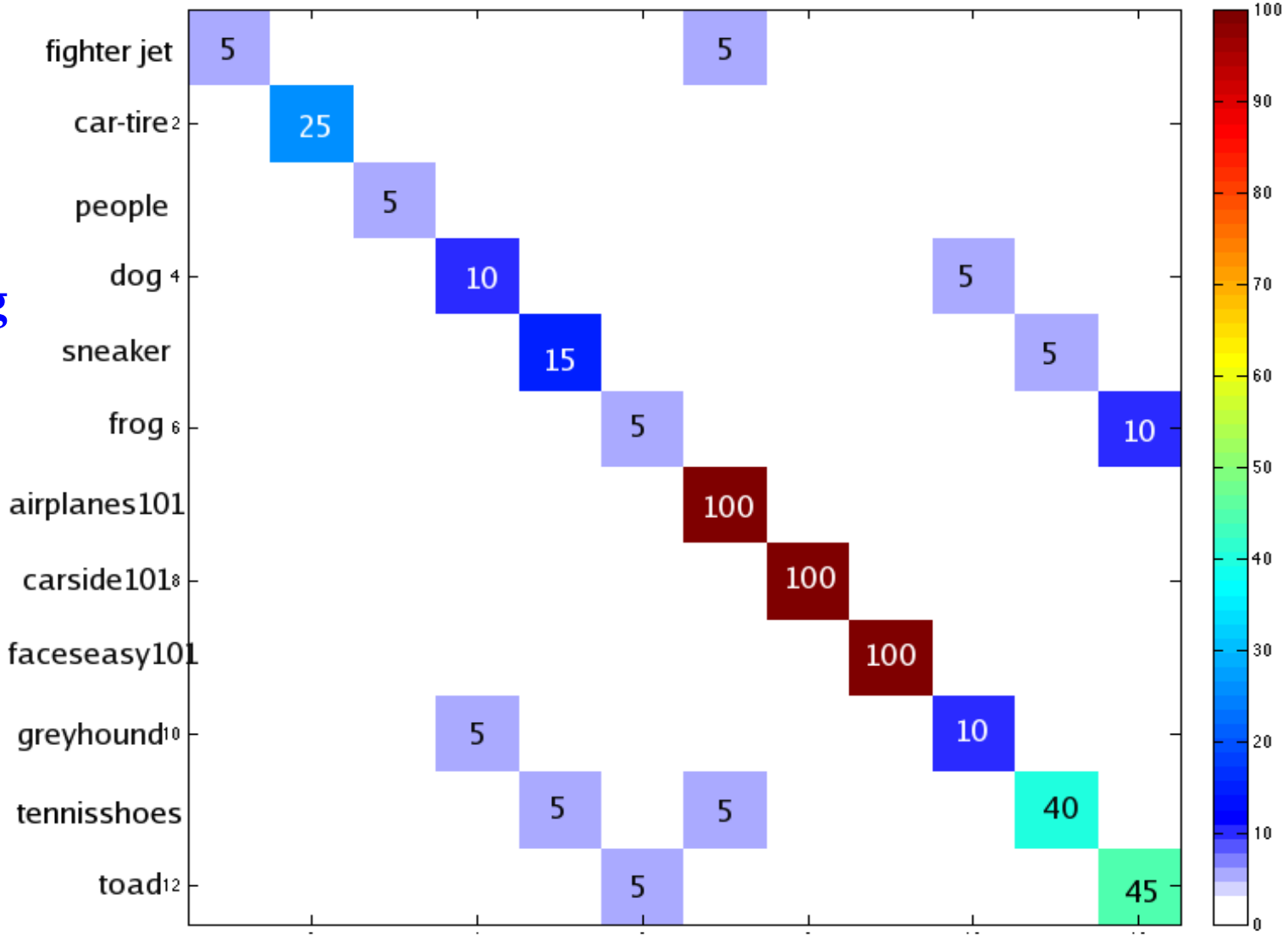

Y (luminance)
U
V
CONVOLUTIONS (9x9)
MAx/SUBSAMPLING (4x4)
CONVOLUTIONS (9x9)
MAx/SUBSAMPLING (5x5)
INPUT 3@140x140
32@132x132
32@33x33
64@25x25
64@5x5

Network: Learned Filters



Caltech 256: Results

31% correct recognition with 60 training samples per category



Conclusion

- **Unsupervised Learning methods for Energy-Based Model**
  - Encoder-only model with contrastive divergence
    - PoE, CCPoE, FoE
  - Decoder-only models with sparsity (Olshausen and Fields)
  - Encoder-Decoder model with Contrastive Divergence (RBM)
  - Encoder-Decoder model with sparsity (Ranzato et al)

Conclusion (continued)

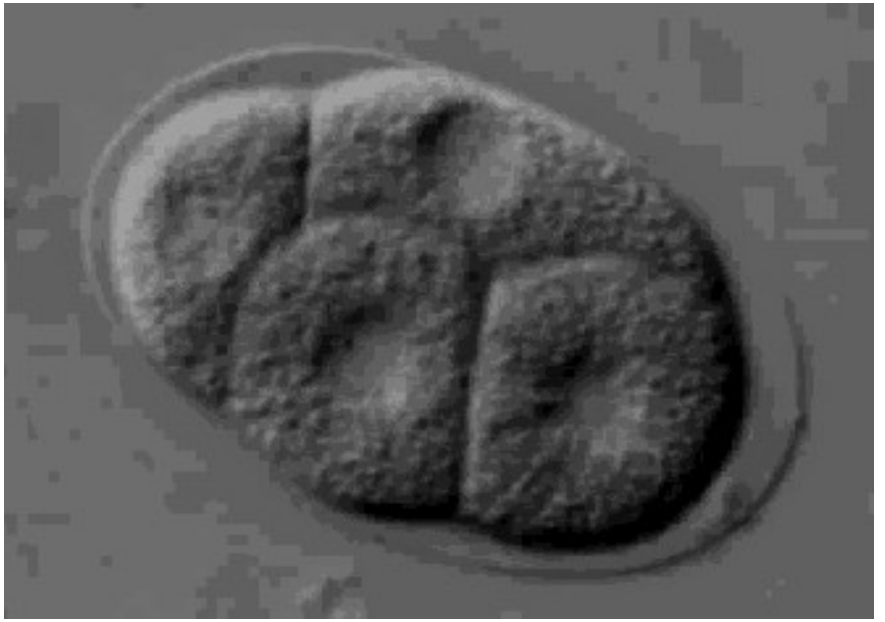
- **Invariant vision tasks require deep learning**
  - shallow models can't handle complicated invariances.
- **Deep Supervised Learning works well with lots of samples**
  - Convolutional nets have record accuracy on handwriting recognition and face detection.
- **Unsupervised Learning can reduce the need for labeled samples**
  - Stacks of sequentially-trained RBMs or sparse encoder-decoder layers learn good feature without requiring labeled samples
- **Learning invariant feature hierarchies**
  - Each encoder (and decoder) stage contains a layer of trainable feature extractors and a layer of feature pooling. The feature pooling separates the “what” from the “where”.

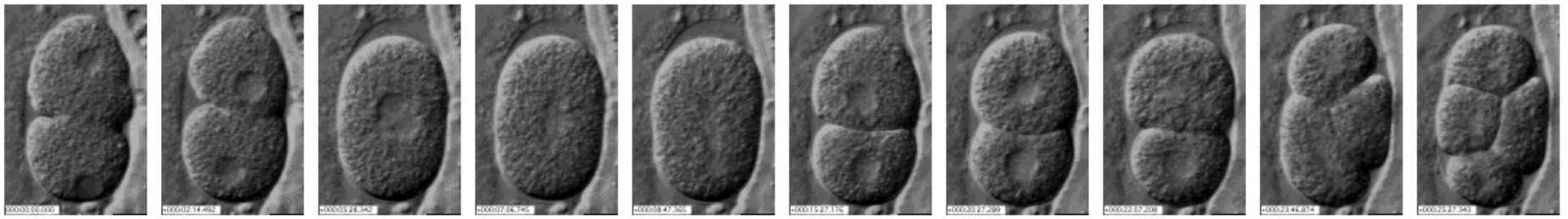
The End

C. Elegans Embryo Phenotyping

[Ning, Delhome, LeCun, Piano, Bottou, Barbano
IEEE Trans. Image Processing, October 2005]

- **Analyzing results for Gene Knock-Out Experiments**
- **Automatically determining if a roundworm embryo is developing normally after a gene has been knocked out.**





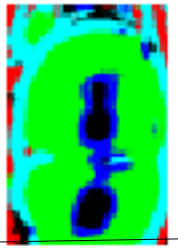
Time-lapse movie

Architecture

- Region Classification with a convolutional network
- Local Consistency with a Conditional Product of Experts
- Embryo classification with elastic model matching

Region Labeling
Convolutional
Network

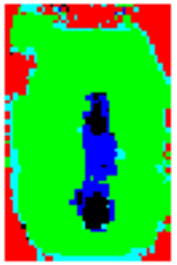
X



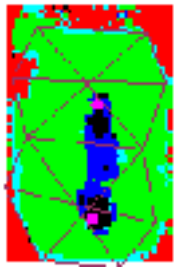
Local Consistency
Satisfaction
Conditional
Convolutional PoE

E(Y,X)

Y



Classification
Elastic Model
Matching



Elastic model
of embryo

Region Labeling with a Convolutional Net

- Supervised training fromhand-labeled images
- 5 categories:
  - nucleus, nuclear membrane, cytoplasm, cell wall, external medium

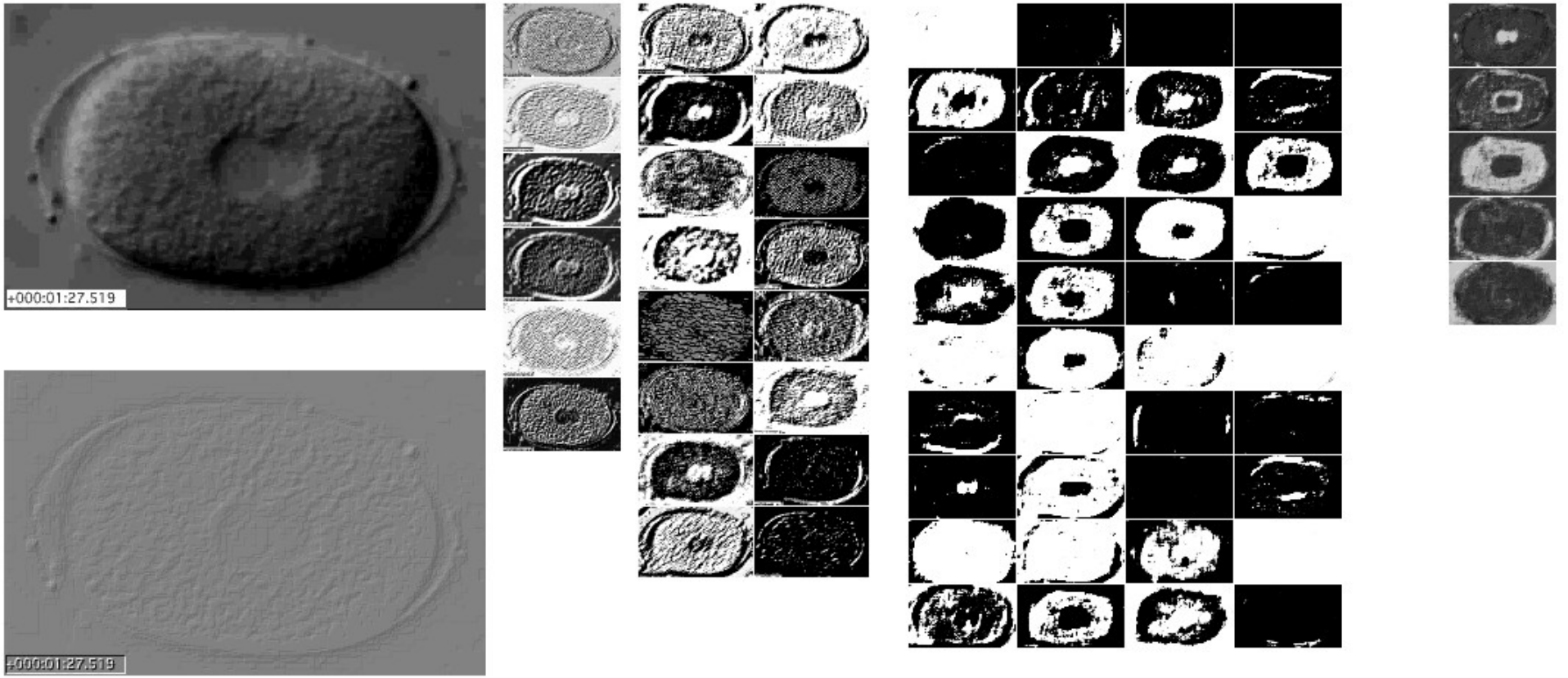
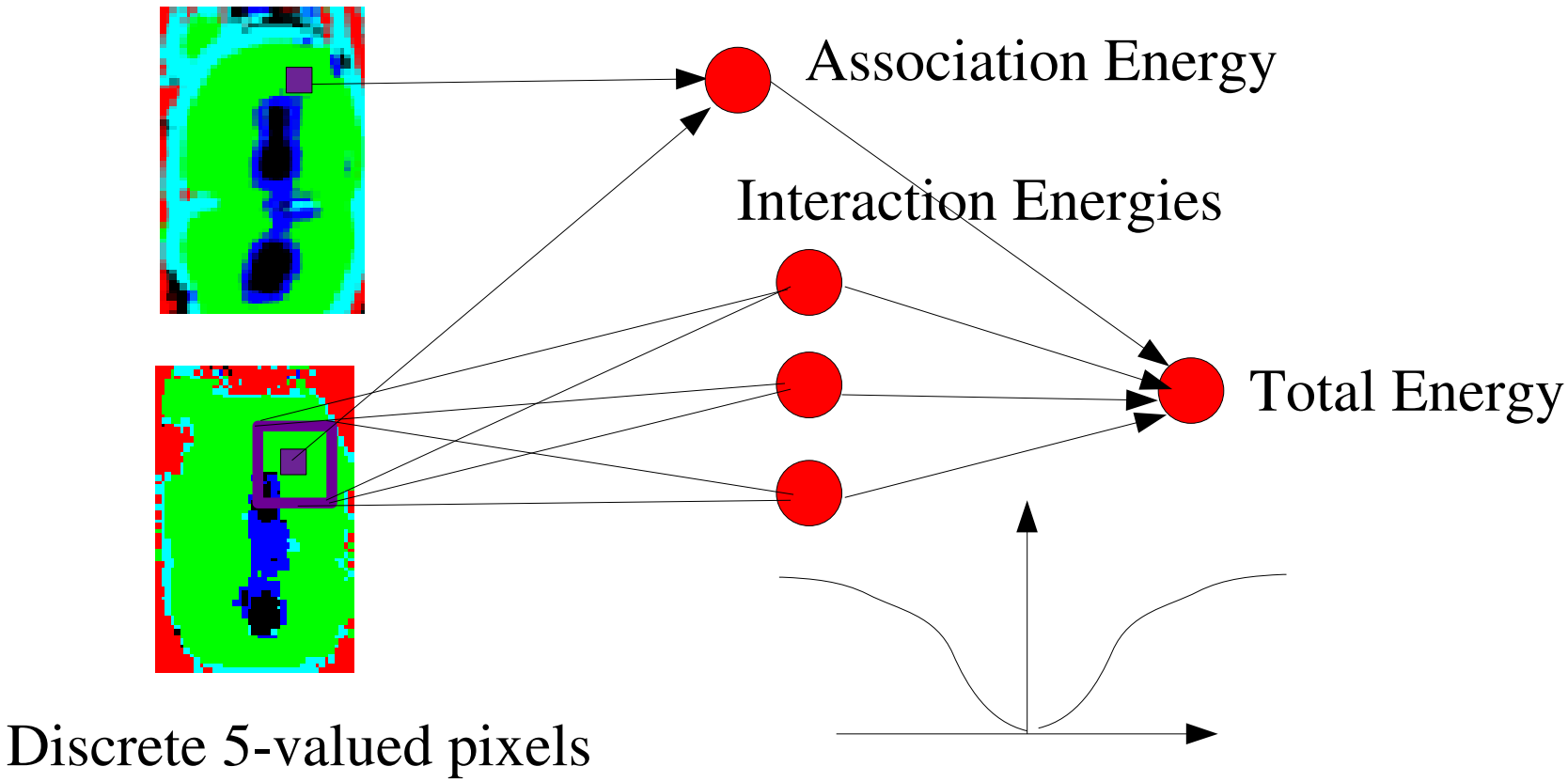


Image Segmentation with Local Consistency Constraints

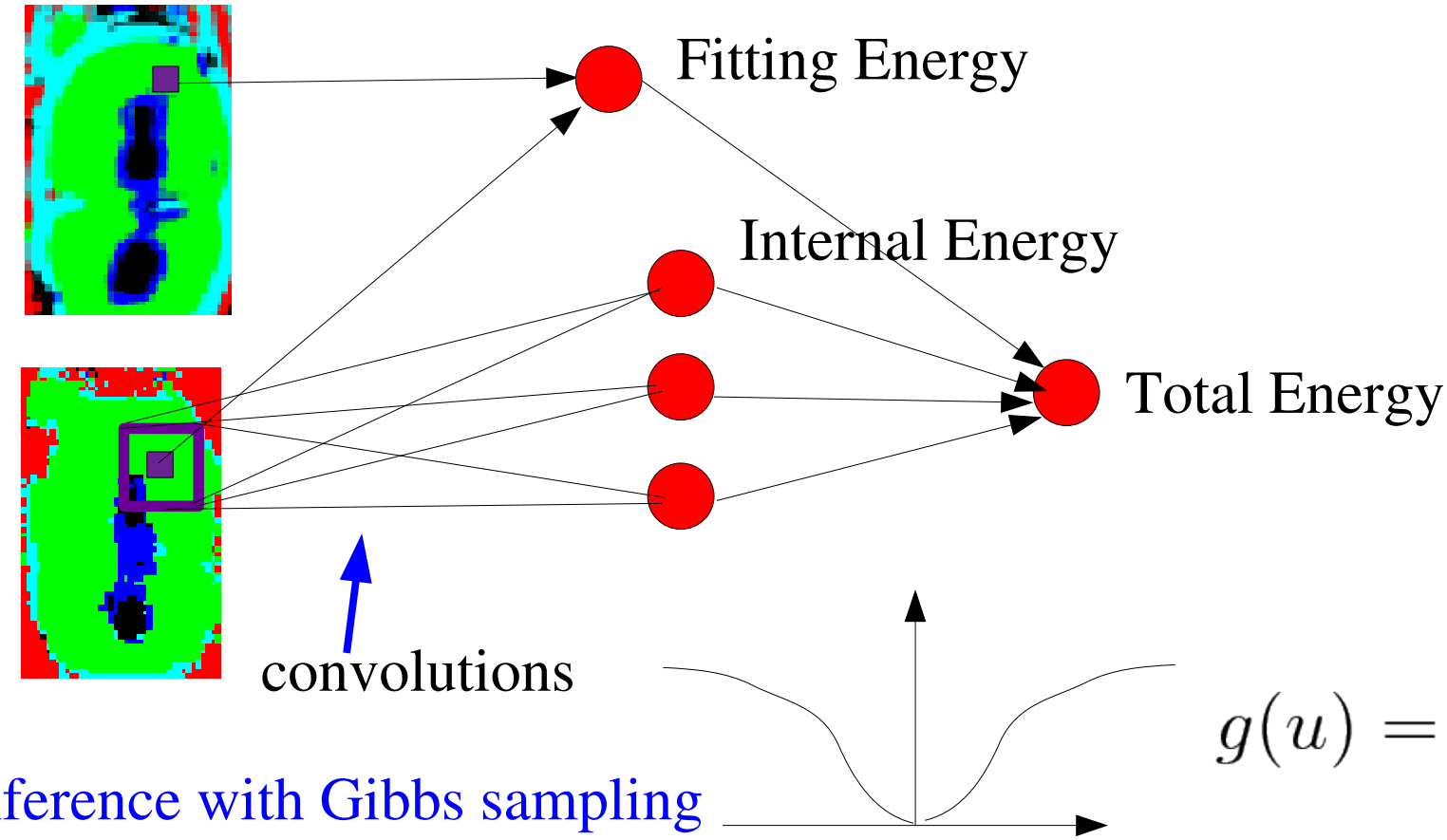
[Teh, Welling, Osindero, Hinton, 2001], [Kumar, Hebert 2003], [Zemel 2004]

- **Learn local consistency constraints with an Energy-Based Model so as to clean up images produced by the segmentor.**



Convolutional Conditional PoE

$$E(Y,X,W) = \sum_{ij} C_{Y_{ij},X_{ij}} + \sum_{k=1}^{10} \sum_{ij} g\left(\sum_{lpq=(1,-2,-2)}^{(5,+2,+2)} W_{klpq}.Y_{l,i-p,j-q} \right)$$



Inference with Gibbs sampling

$$g(u) = \frac{u^2}{1+u^2}$$

C. Elegans Embryo Phenotyping

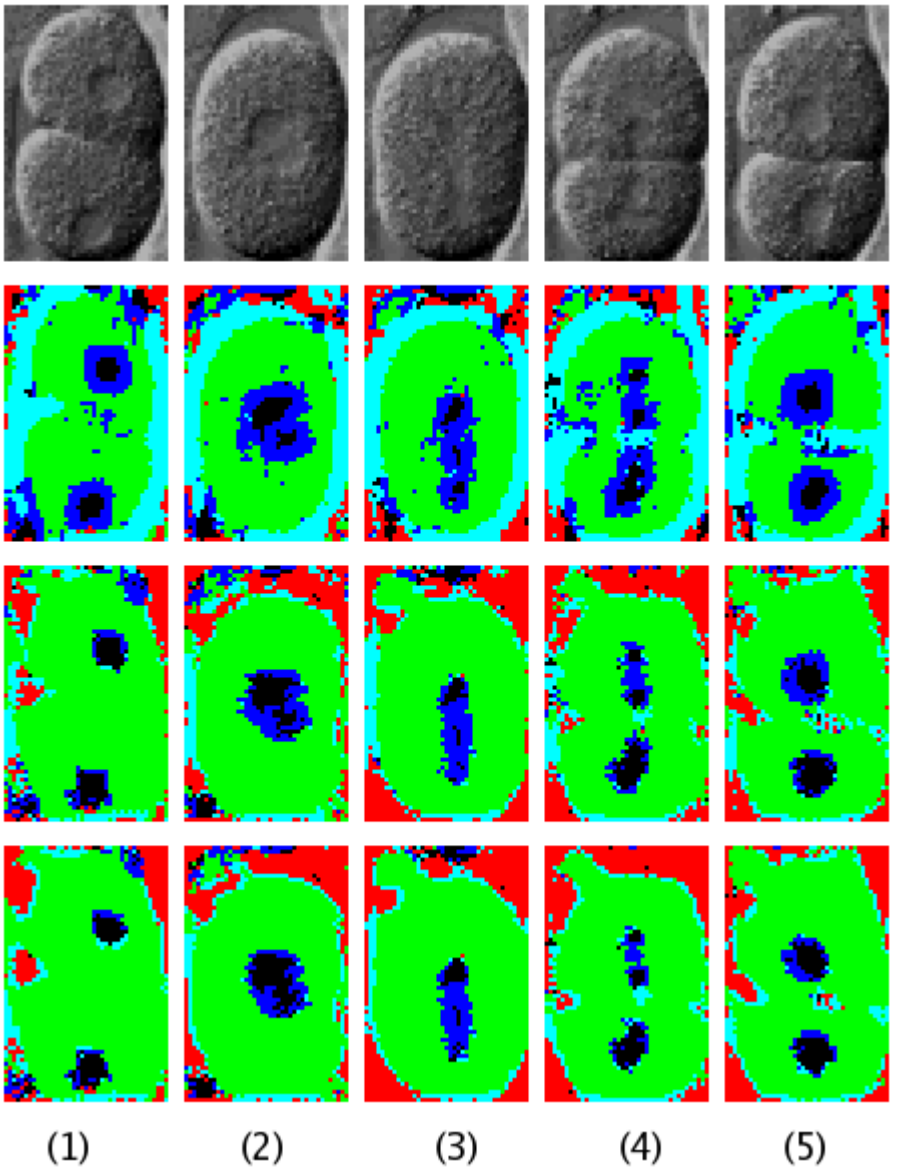
- **Analyzing results for Gene Knock-Out Experiments**

Original Images

Segmentation #1

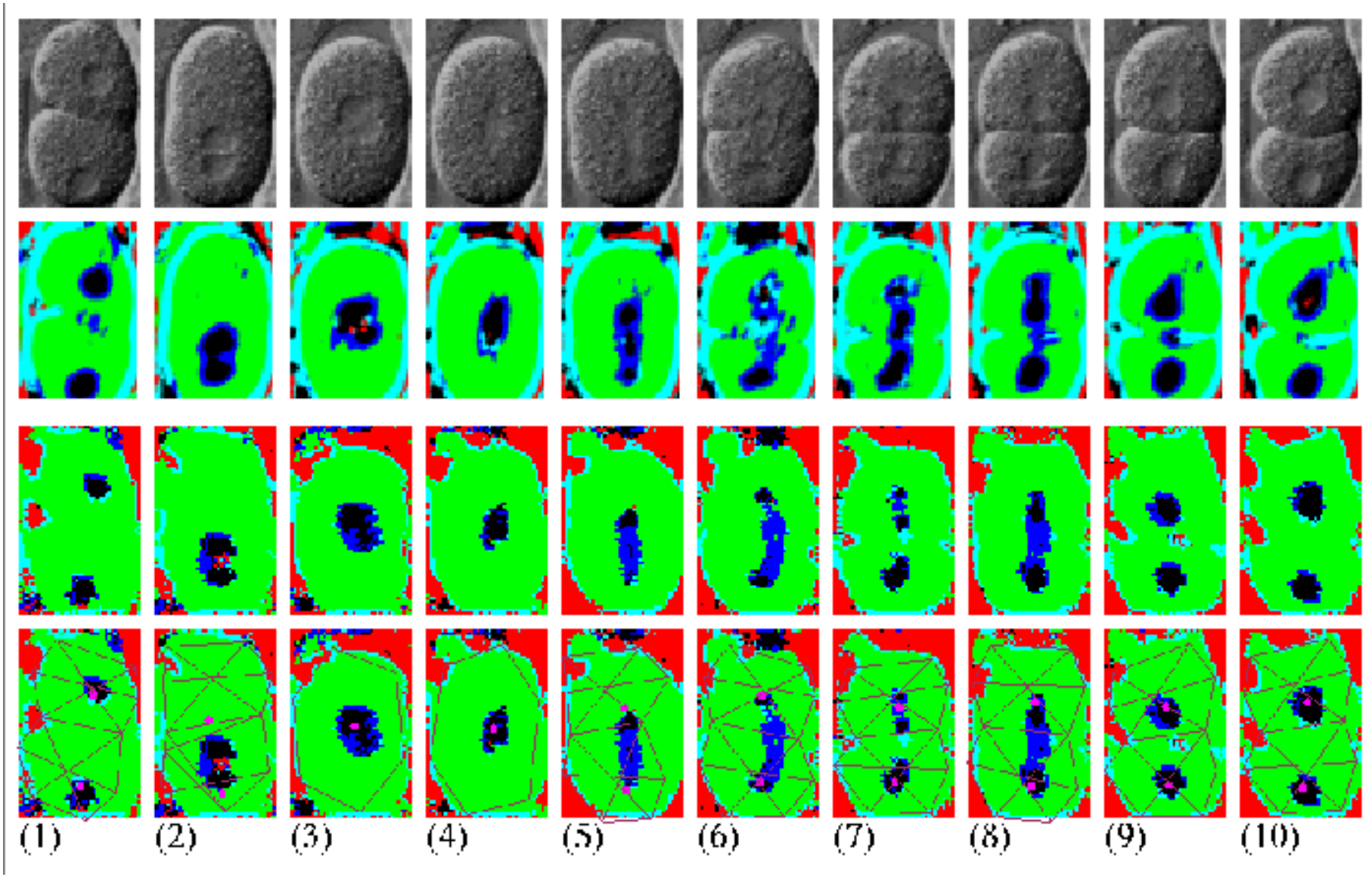
Segmentation #2

CCPoE Cleanup



C. Elegans Embryo Phenotyping

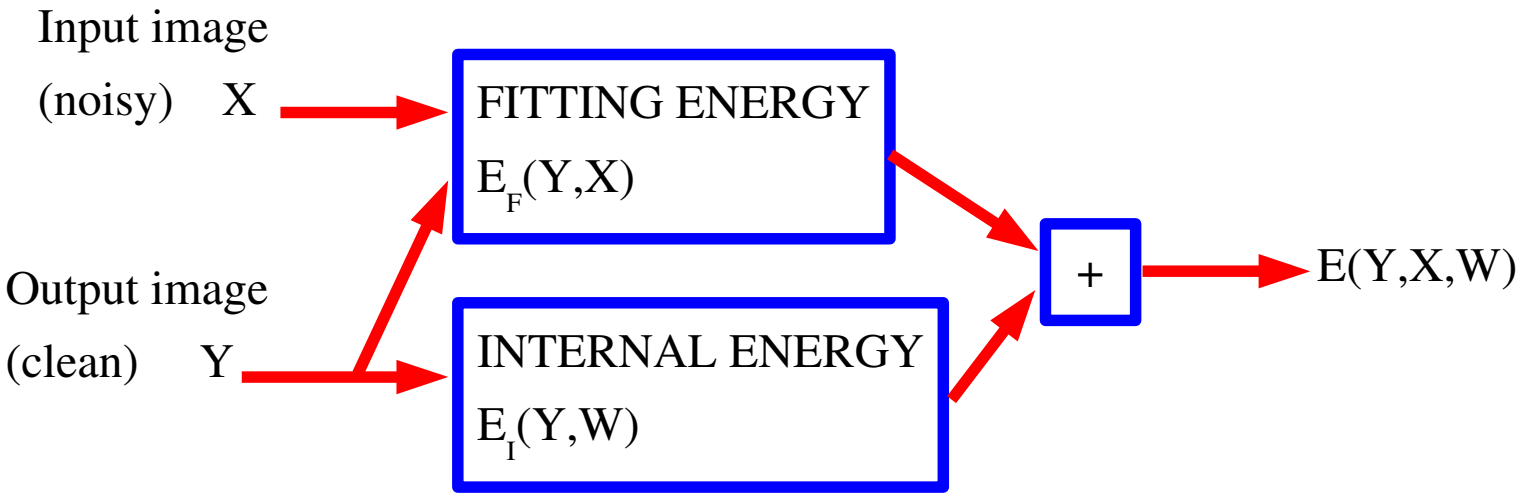
- Analyzing results for Gene Knock-Out Experiments



Example: Conditional Product of Experts

[Ning, Delhome, LeCun, Piano, Bottou, Barbano: "Toward Automatic Phenotyping of Developing Embryos from Videos" *IEEE Trans. Image Processing*, September 2005]

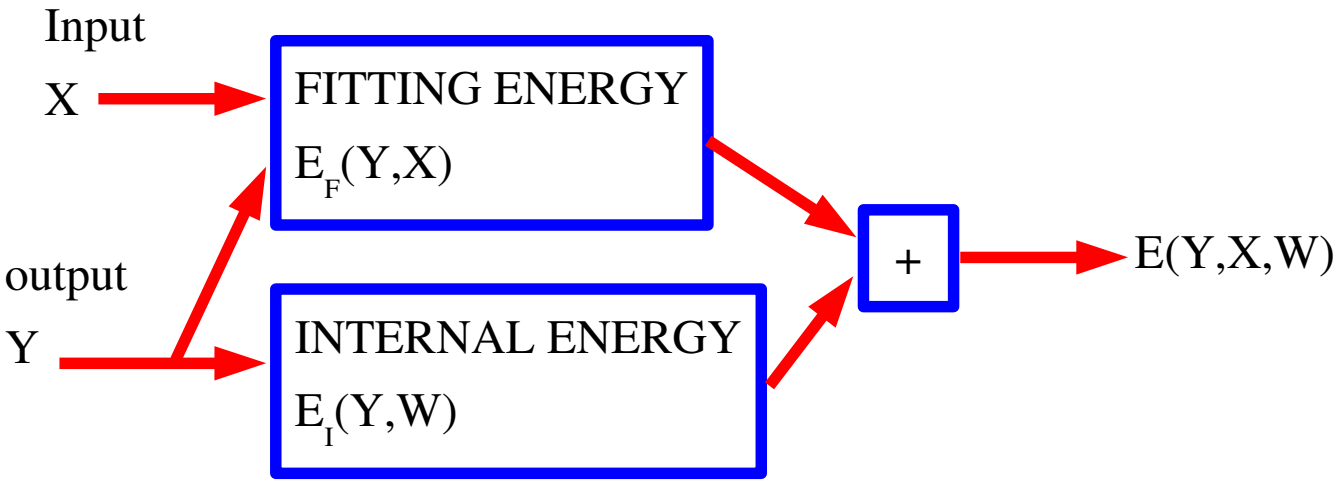
Using Energy-Based Models for image "cleanup" (segmentation, denoising,.....)



$$E(Y, X, W) = E_F(Y, X) + E_I(Y, W)$$

MAP Inference: clamp X and find a Y that minimizes E(Y,X,W)

Conditional PoE: Contrastive Divergence Training

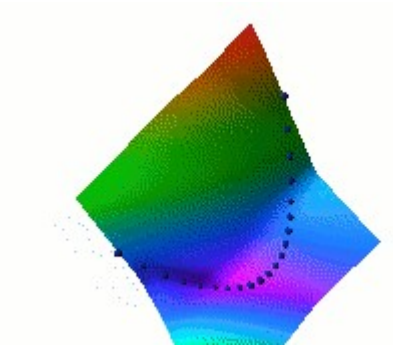


$$L_{\text{nll}}(Y^i, X^i, W) = E(Y^i, X^i, W) + \frac{1}{\beta} \log \left[\sum_y \exp(-\beta E(y, X^i, W)) \right]$$

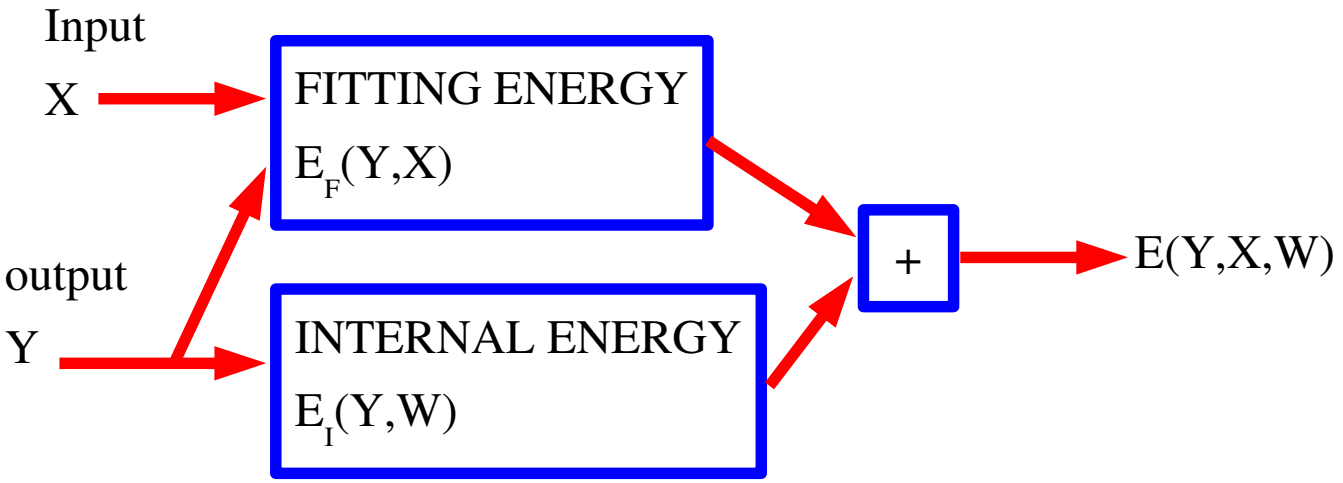
The negative log-likelihood loss has an intractable sum over all possible configurations of Y

Hinton's method:

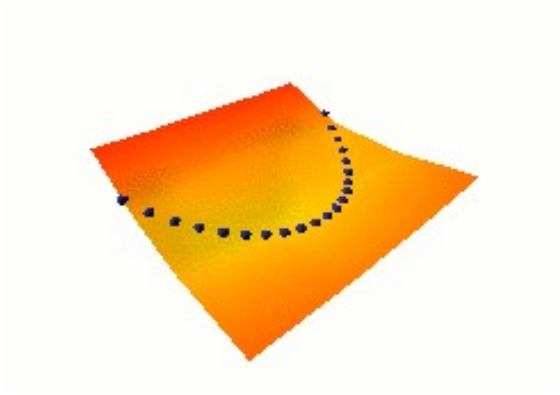
- use MCMC to approximate the derivative of the log partition function
- realize it takes too long. Get bored waiting.
- decide to cut the number of iterations of MCMC.
- realize that it's a sensible thing to do, and call is Contrastive Divergence
- come up with a complicated justification for it.

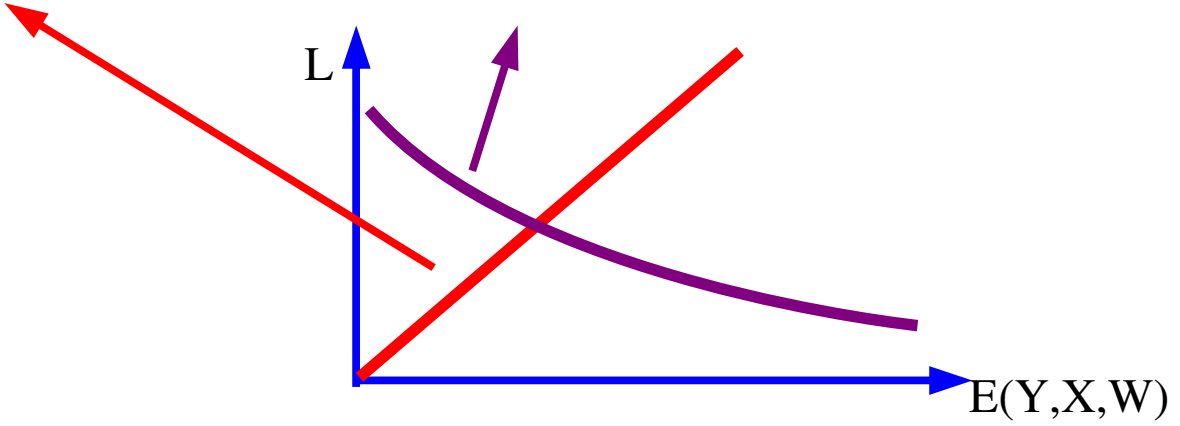


Conditional PoE: Training with the Linear-Exponential Loss



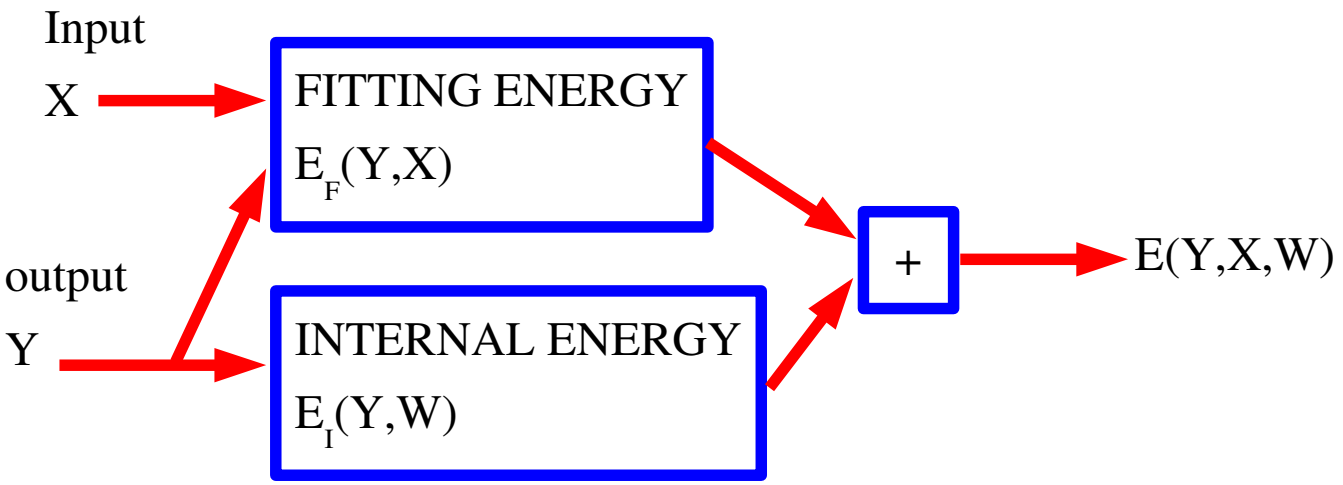
$$L(Y^i, X^i, W) = E(Y^i, X^i, W) + c.\exp(-\beta \min_{y,|y-Y^i|>\delta} E(y, X^i, W))$$





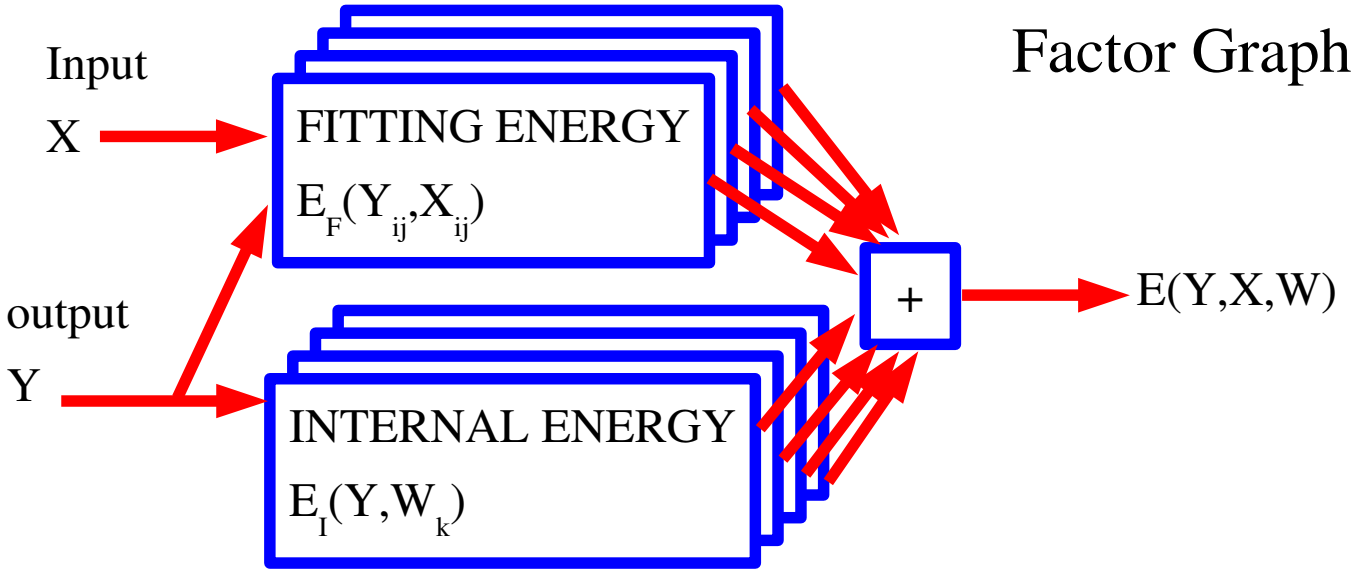
Energy-based loss: make the energy of the desired answer low, and make the energy of the most offending undesired answer high (forget about likelihoods altogether)

Conditional Product of Experts: Training



$$L(Y^i, X^i, W) = E(Y^i, X^i, W) - c.\exp(-\beta \min_{y,|y-Y^i|>\delta} E(y, X^i, W))$$

Conditional Product of Experts



$$E(Y,X,W) = \sum_{ij} E_F(Y_{ij},X_{ij}) + \sum_{k} E_I(Y,W_k)$$

Fitting energy: summed over “sites” (e.g. pixels)

Internal energy: summed over “experts” (e.g. Features) and sites.

Architecture

- Region Classification with a convolutional network
- Local Consistency with a Conditional Product of Experts
- Embryo classification with elastic model matching

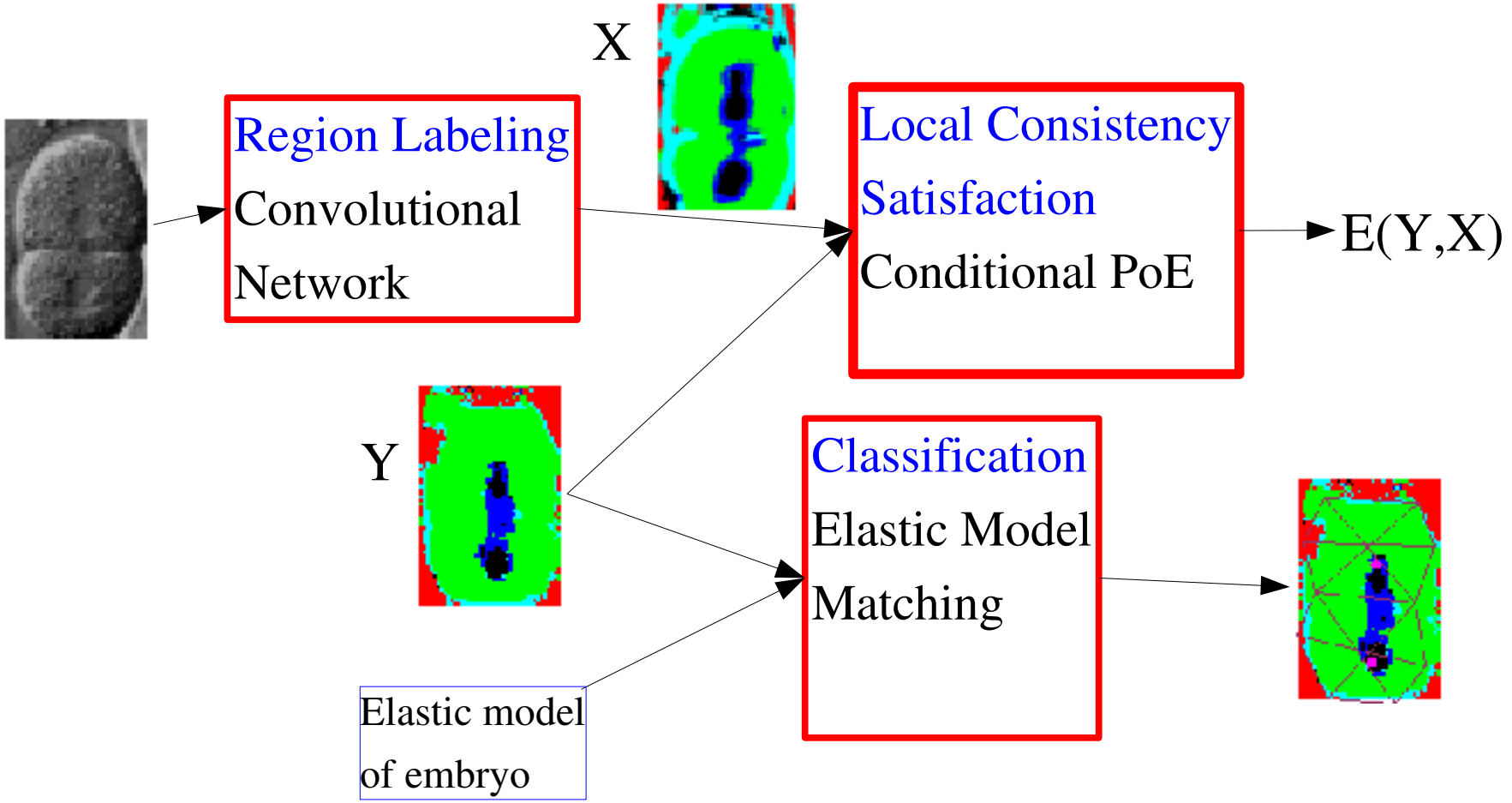
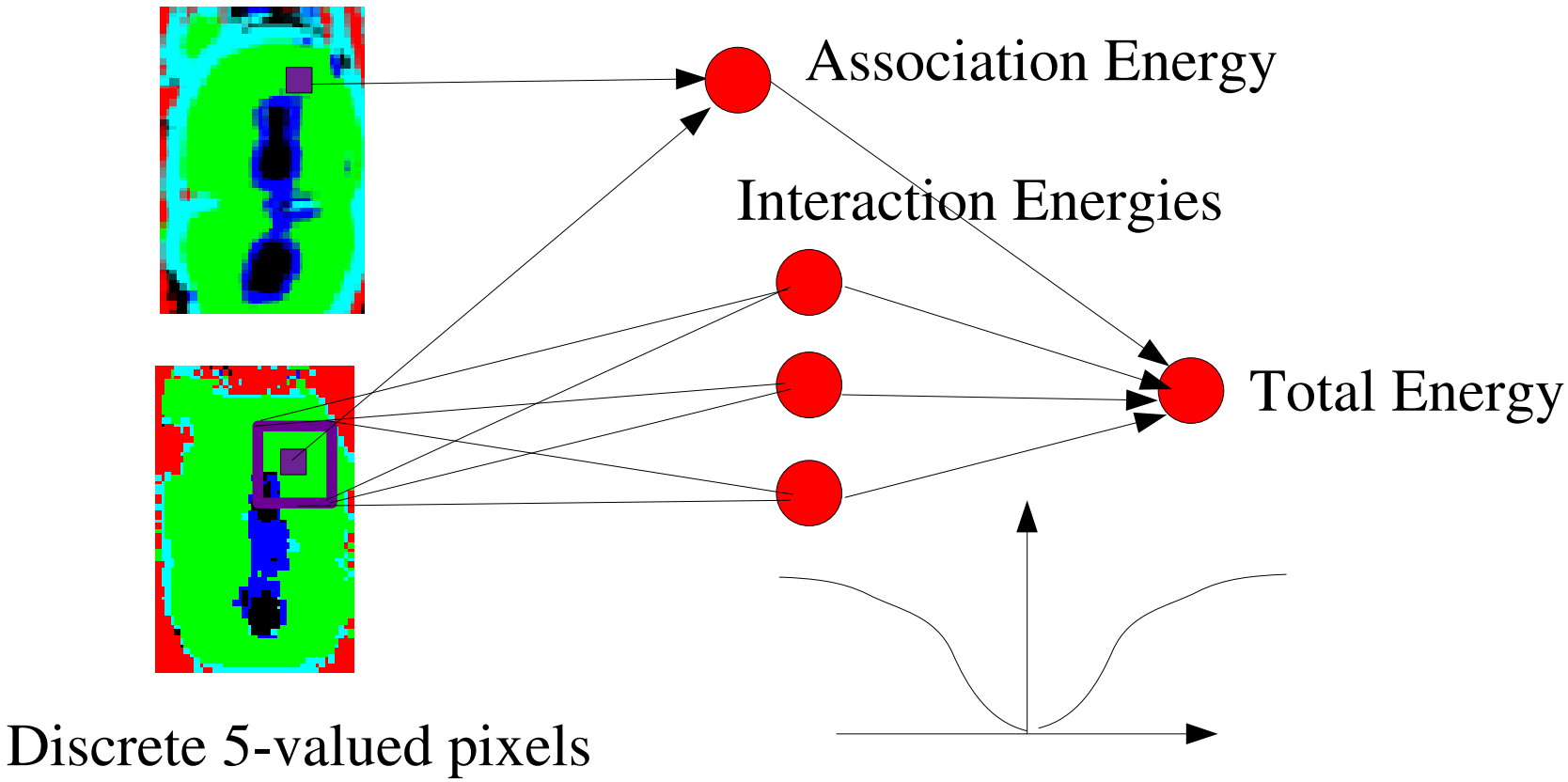


Image Segmentation with Local Consistency Constraints

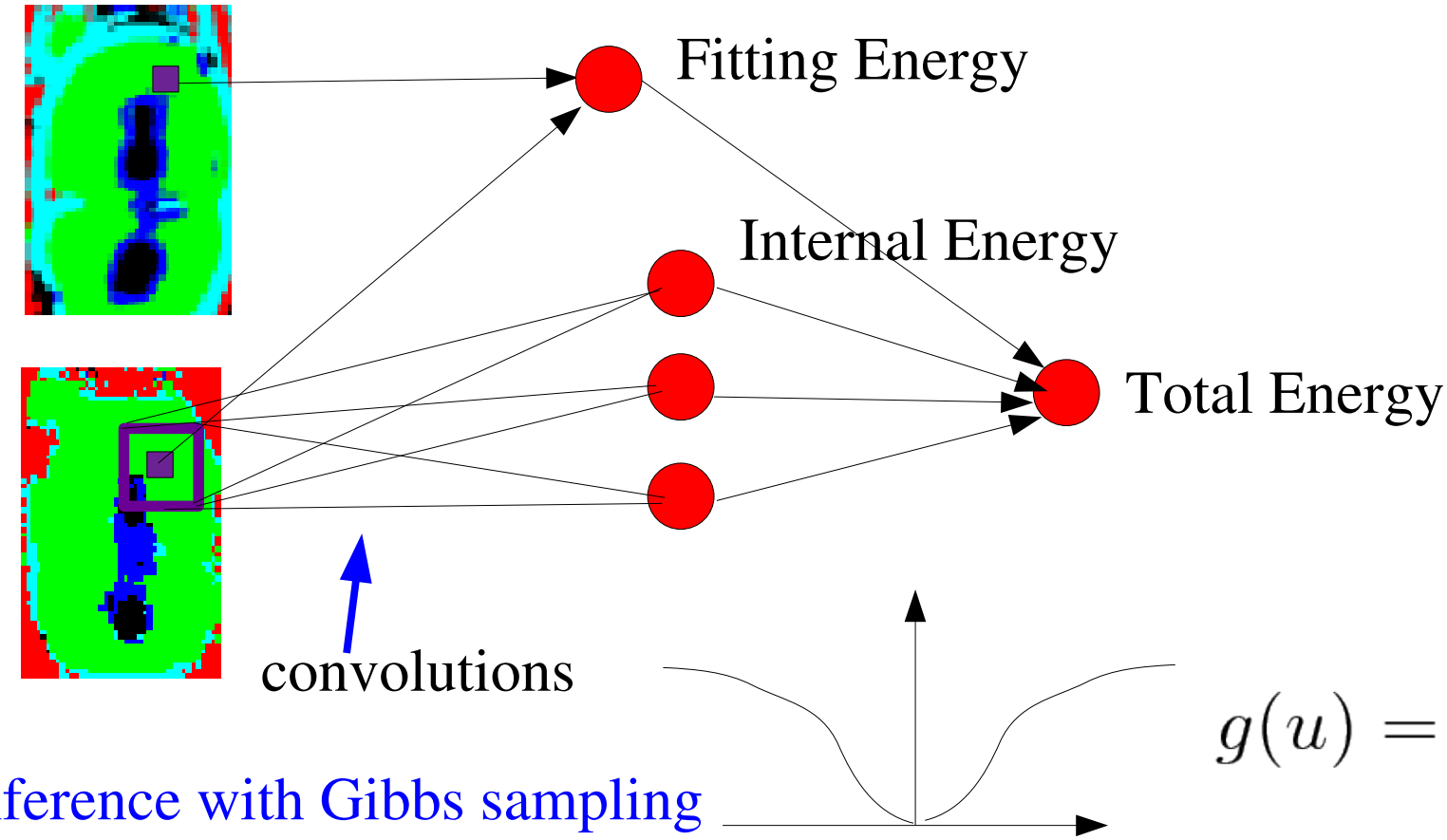
[Teh, Welling, Osindero, Hinton, 2001], [Kumar, Hebert 2003], [Zemel 2004]

- **Learn local consistency constraints with an Energy-Based Model so as to clean up images produced by the segmentor.**



Convolutional Conditional PoE

$$E(Y,X,W) = \sum_{ij} C_{Y_{ij},X_{ij}} + \sum_{k=1}^{10} \sum_{ij} g\left(\sum_{lpq=(1,-2,-2)}^{(5,+2,+2)} W_{klpq}.Y_{l,i-p,j-q} \right)$$



Inference with Gibbs sampling

$$g(u) = \frac{u^2}{1+u^2}$$

C. Elegans Embryo Phenotyping

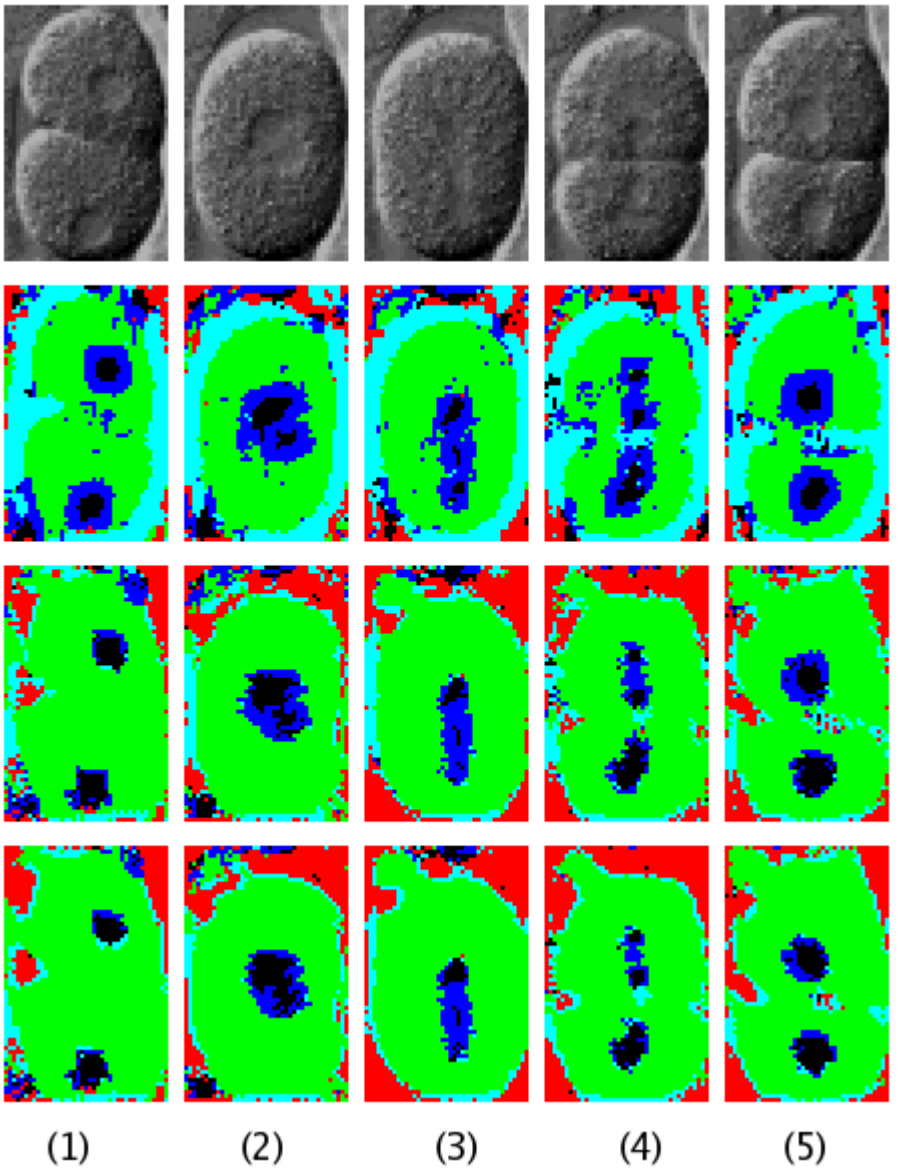
- **Analyzing results for Gene Knock-Out Experiments**

Original Images

Segmentation #1

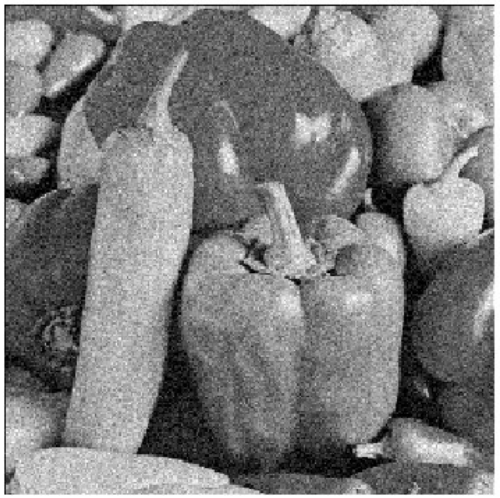
Segmentation #2

CCPoE Cleanup



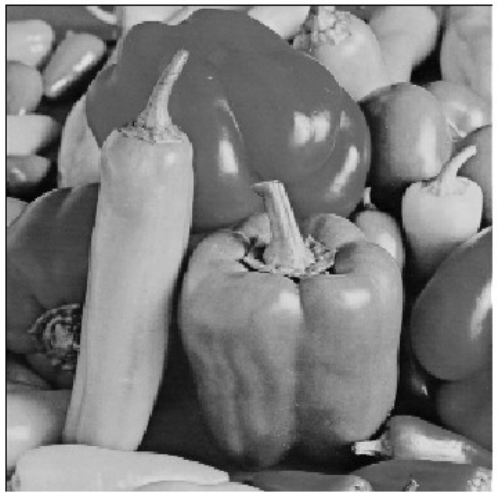
Convolutional Conditional PoE for Image Denoising

$$L(Y^i, X^i, W) = E(Y^i, X^i, W) + c.\exp(-\beta \min_{y,|y-Y^i|>\delta} E(y, X^i, W))$$



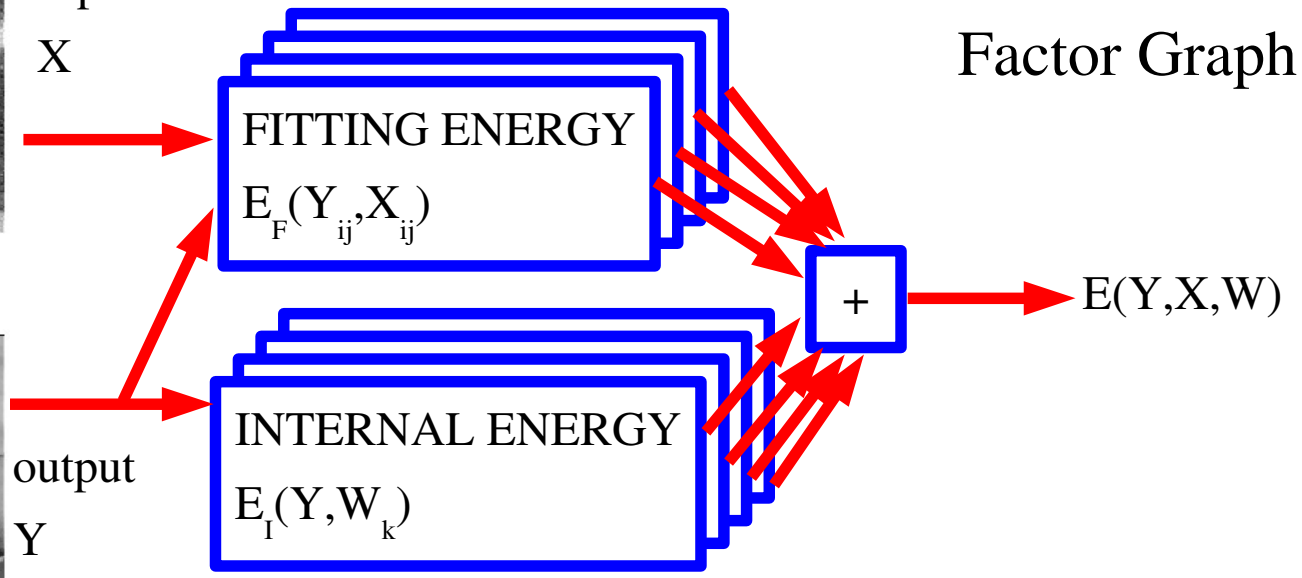
Input

X



output

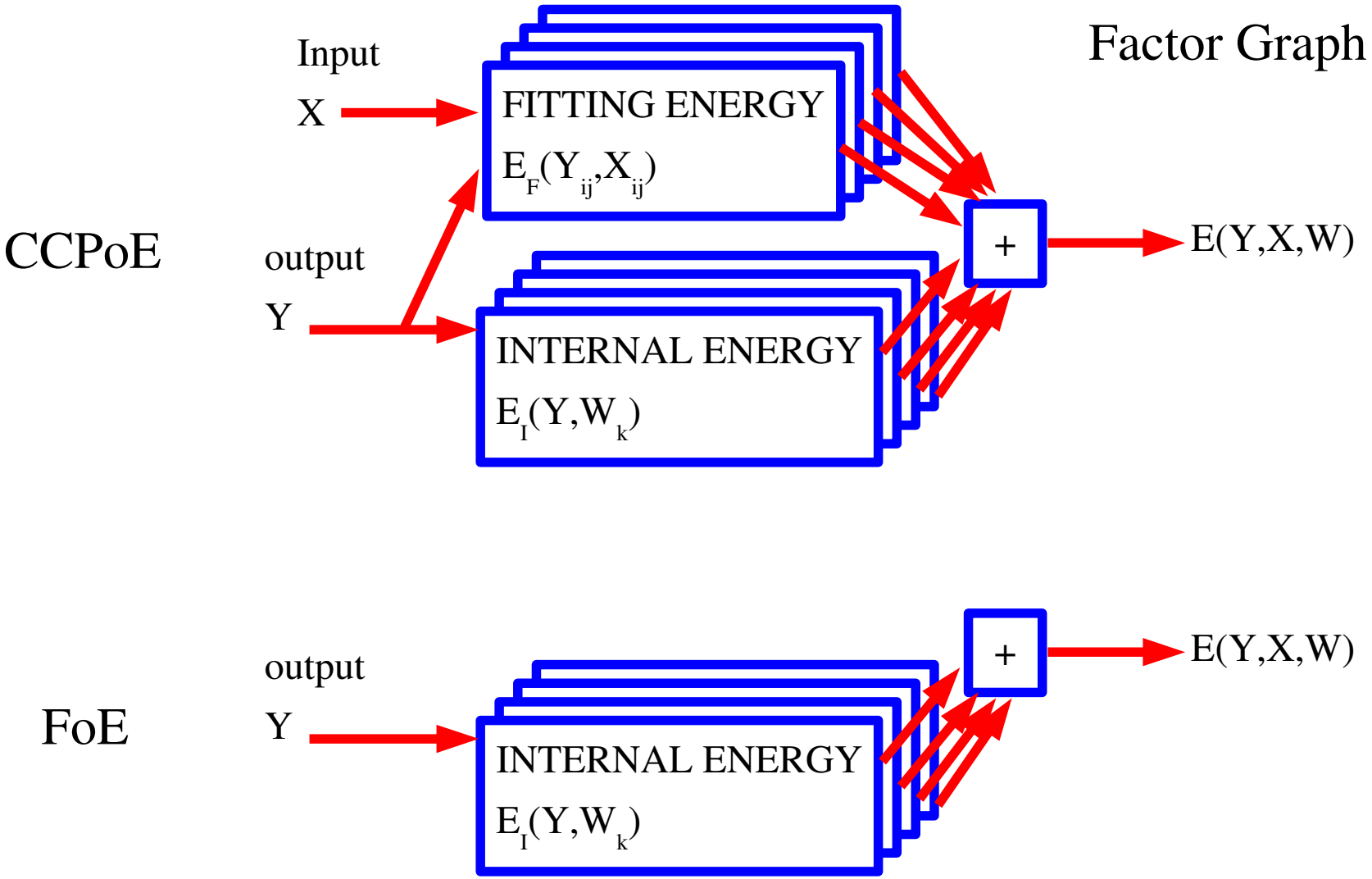
Y



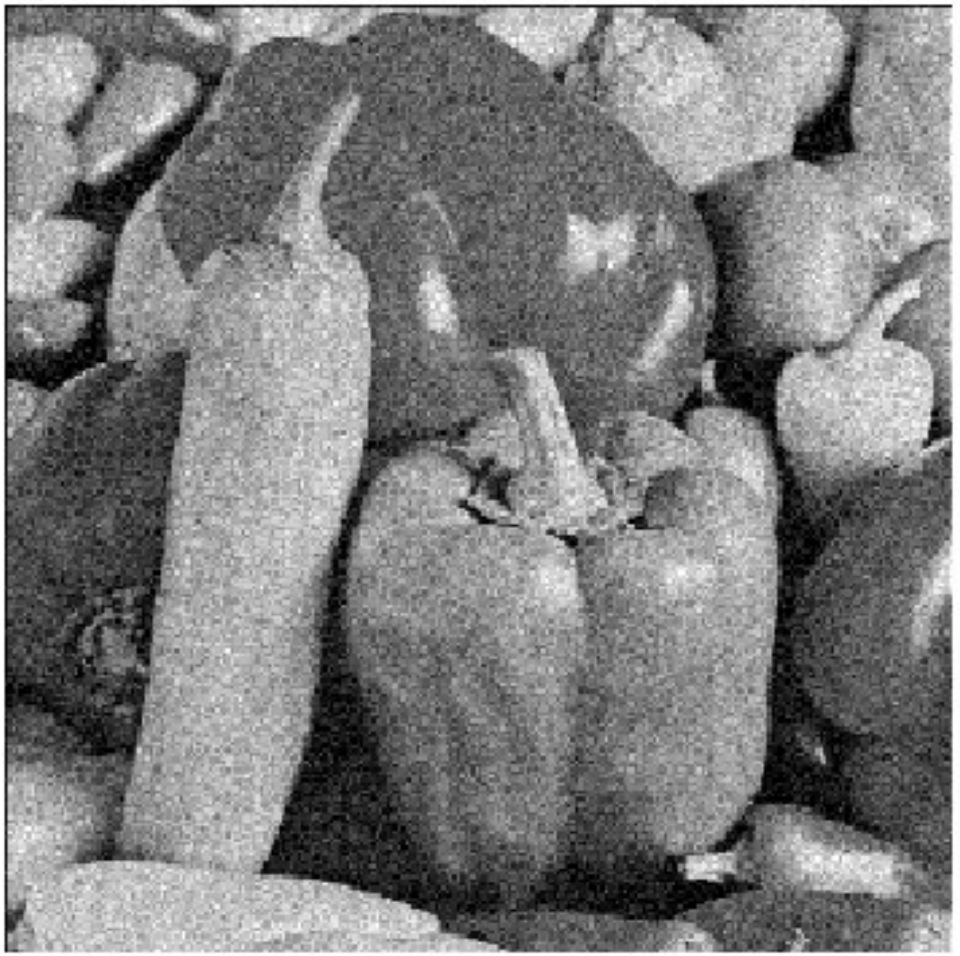
Factor Graph

Convolutional Conditional PoE for Image Denoising

- Somewhat similar to the Field of Experts [Roth & Black CVPR 2005]



Convolutional Conditional PoE for Image Denoising



Noisy peppers PSNR=22.10

CCPoE PSNR=30.40

Convolutional Conditional PoE for Image Denoising

FoE PSNR=30.41

(Roth & Black report 30.58)

CCPoE PSNR=30.40

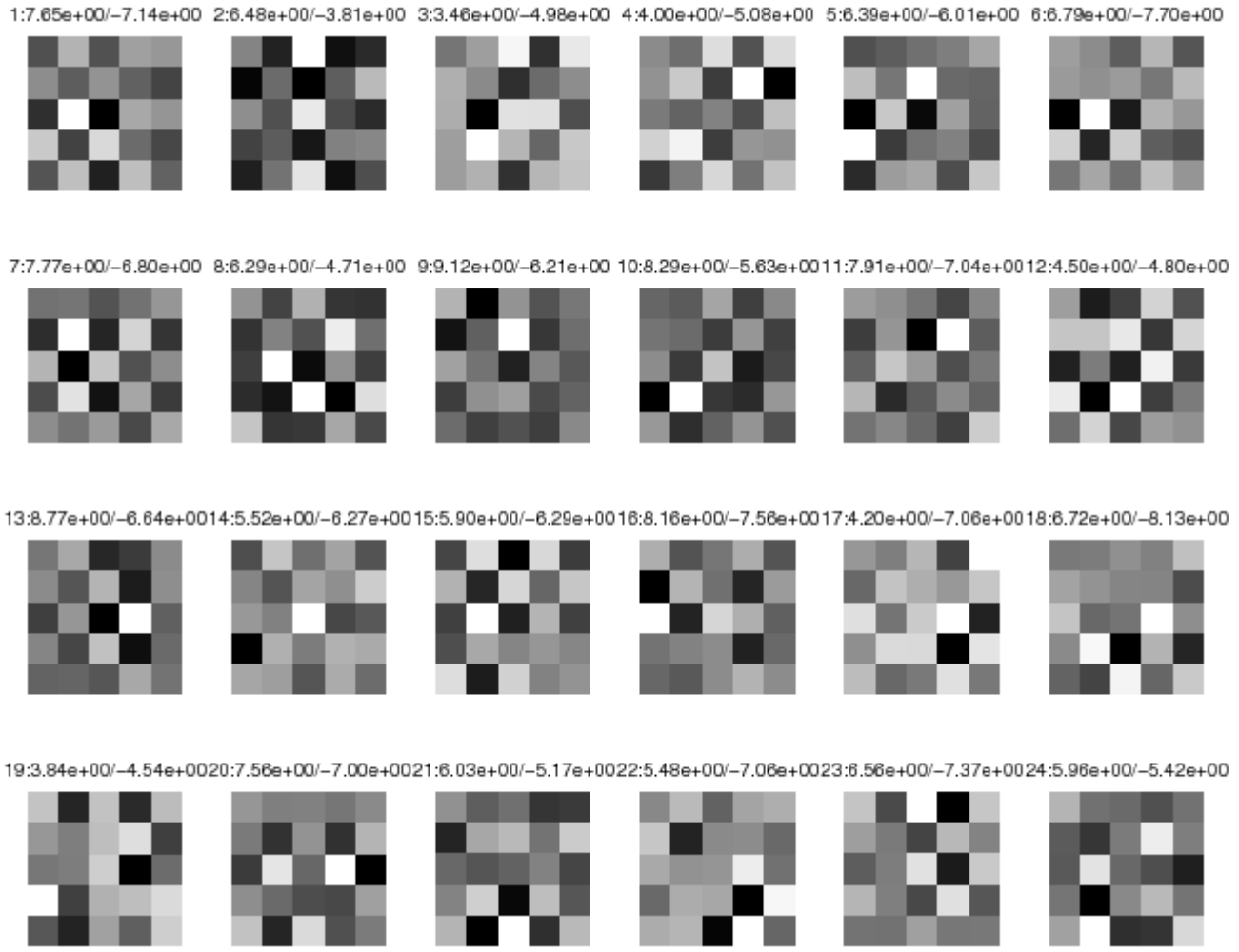
Convolutional Conditional PoE for Image Denoising

Random Kernels, PSNR= 29.70

(Gasp!)

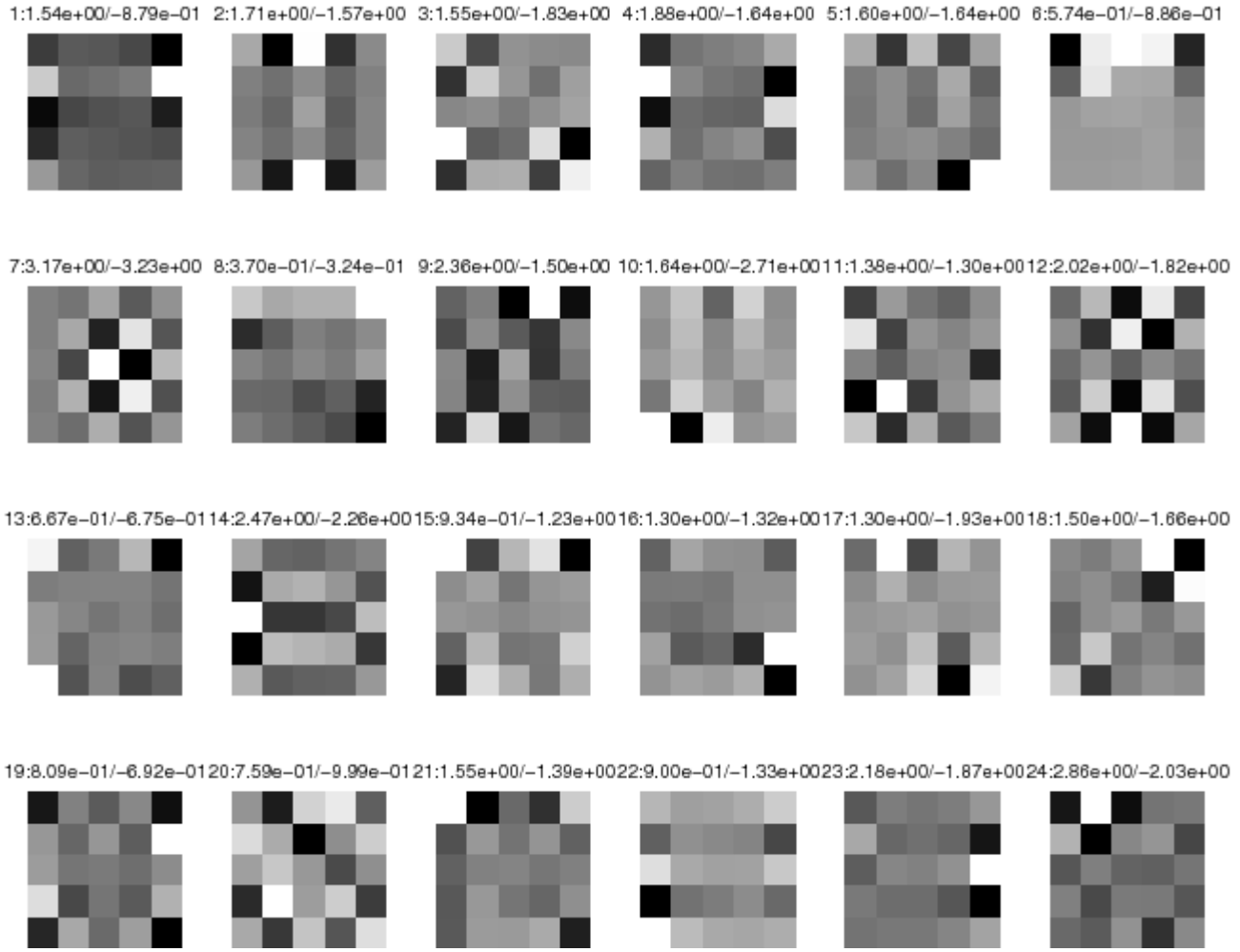
CCPoE PSNR=30.40

Convolutional Conditional PoE for Image Denoising



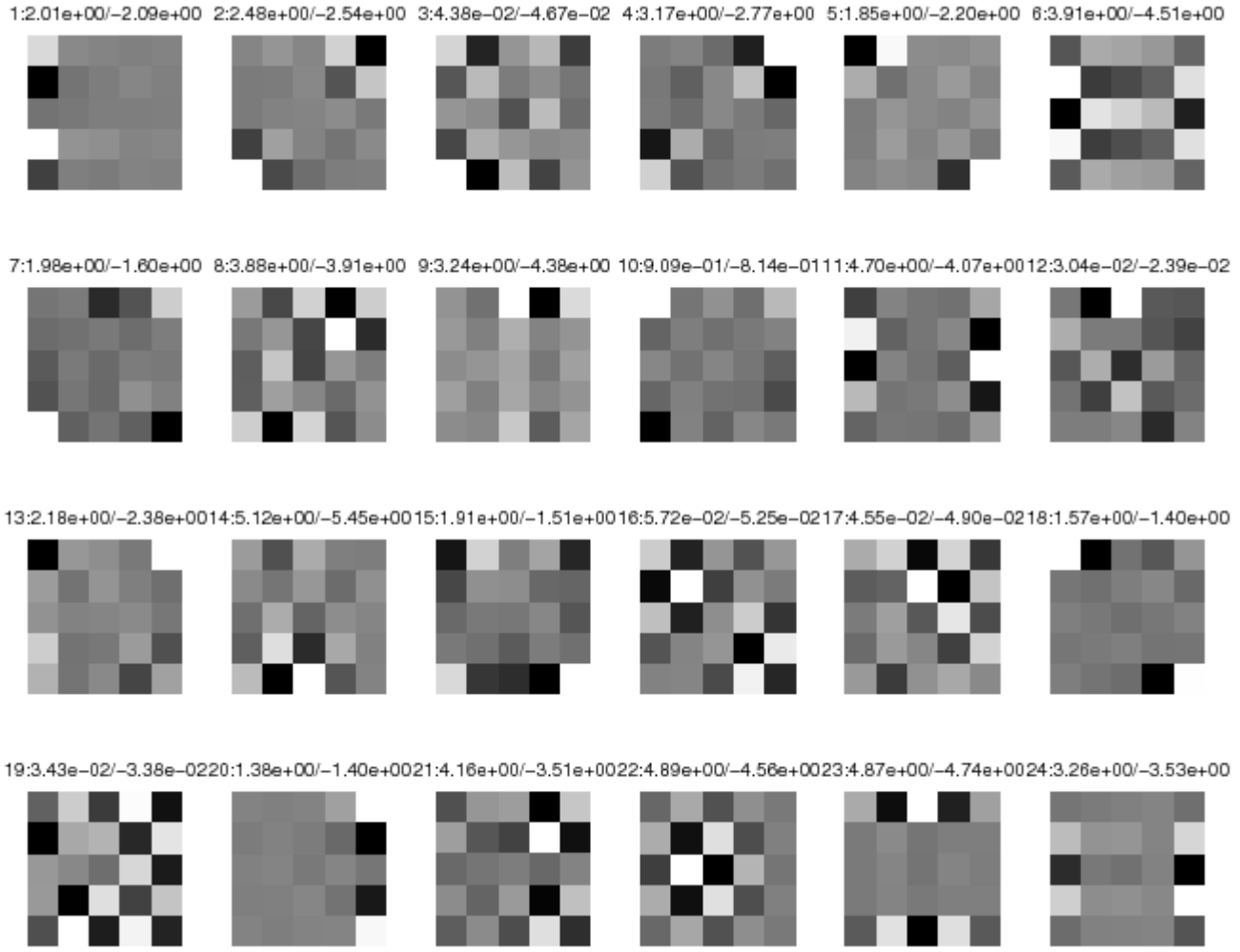
Random Kernels, PSNR= 29.70

Convolutional Conditional PoE for Image Denoising



Roth & Black Kernels, PSNR= 30.58

Convolutional Conditional PoE for Image Denoising



CCPoE Kernels, PSNR= 30.40