Learning Invariant Feature Hierarchies

Yann LeCun

The Courant Institute of Mathematical Sciences

Center For Neural Science

New York University

collaborators:

Y-Lan Boureau, Rob Fergus,

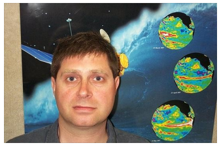
Karol Gregor, Kevin Jarrett,

Koray Kavukcuoglu, Marc'Aurelio Ranzato

Problem: supervised ConvNets don't work with few labeled samples

- **On recognition tasks with few labeled samples, deep supervised architectures don't do so well**
- **Example: Caltech-101 Object Recognition Dataset**
  - 101 categories of objects (gathered from the web)
  - Only 30 training samples per category!
- **Recognition rates (OUCH!):**
  - Supervised ConvNet: **29.0%**
  - SIFT features + Pyramid Match Kernel SVM: **64.6%**
    - [Lazebnik et al. 2006]
- **When learning the features, there are simply too many parameters to learn in purely supervised mode (or so we thought).**

face



beaver

wild cat

lotus

ant

dollar

metronome

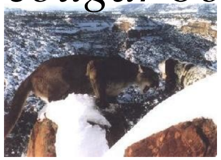
w. chair

minaret

cellphone

joshua t.

cougar body



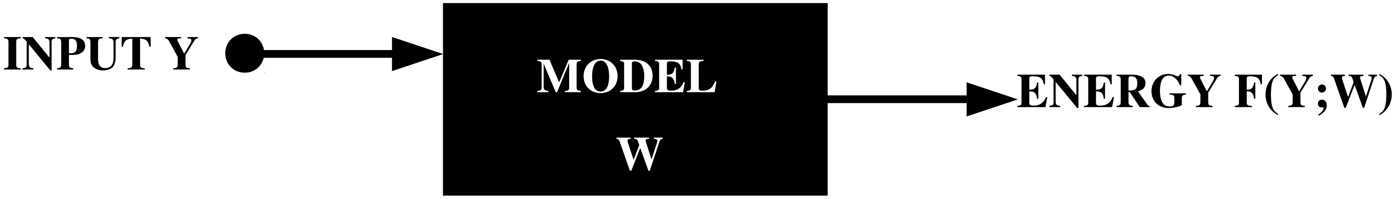
background

Unsupervised Deep Learning: Leveraging Unlabeled Data

[Hinton 05, Bengio 06, LeCun 06, Ng 07]

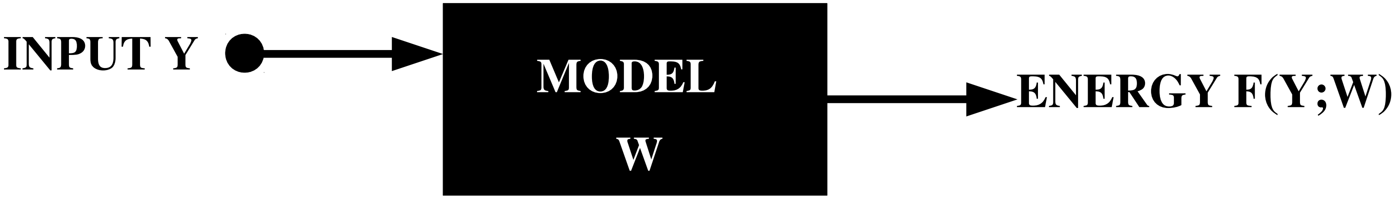
- **Unlabeled data is usually available in large quantity**
- **A lot can be learned about the world by just looking at it**
- **Unsupervised learning captures underlying regularities about the data**
- **The best way to capture underlying regularities is to learn good representations of the data**
- **The main idea of Unsupervised Deep Learning**
  - Learn each layer one at a time in unsupervised mode
  - Stick a supervised classifier on top
  - Optionally: refine the entire system in supervised mode
- **Unsupervised Learning view as Energy-Based Learning**

Energy-Based Framework for Unsupervised Learning



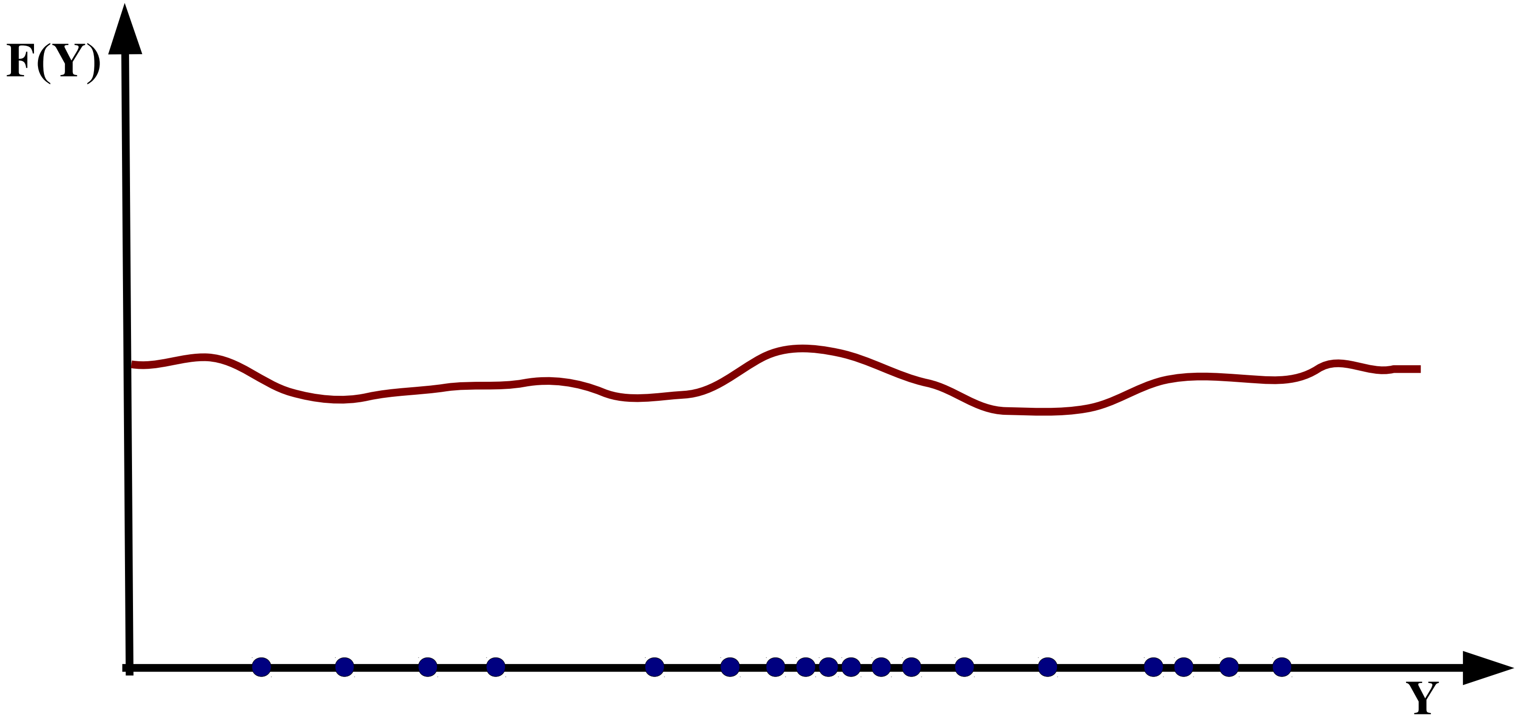
- **GOAL:** make F(Y,W) lower around areas of high data density

Energy-Based Framework for Unsupervised Learning

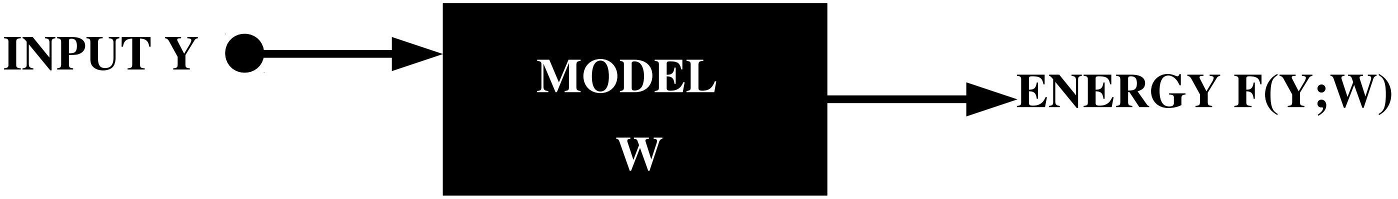


- **GOAL: make F(Y,W) lower around areas of high data density**

ENERGY BEFORE TRAINING



Energy-Based Framework for Unsupervised Learning


INPUT Y

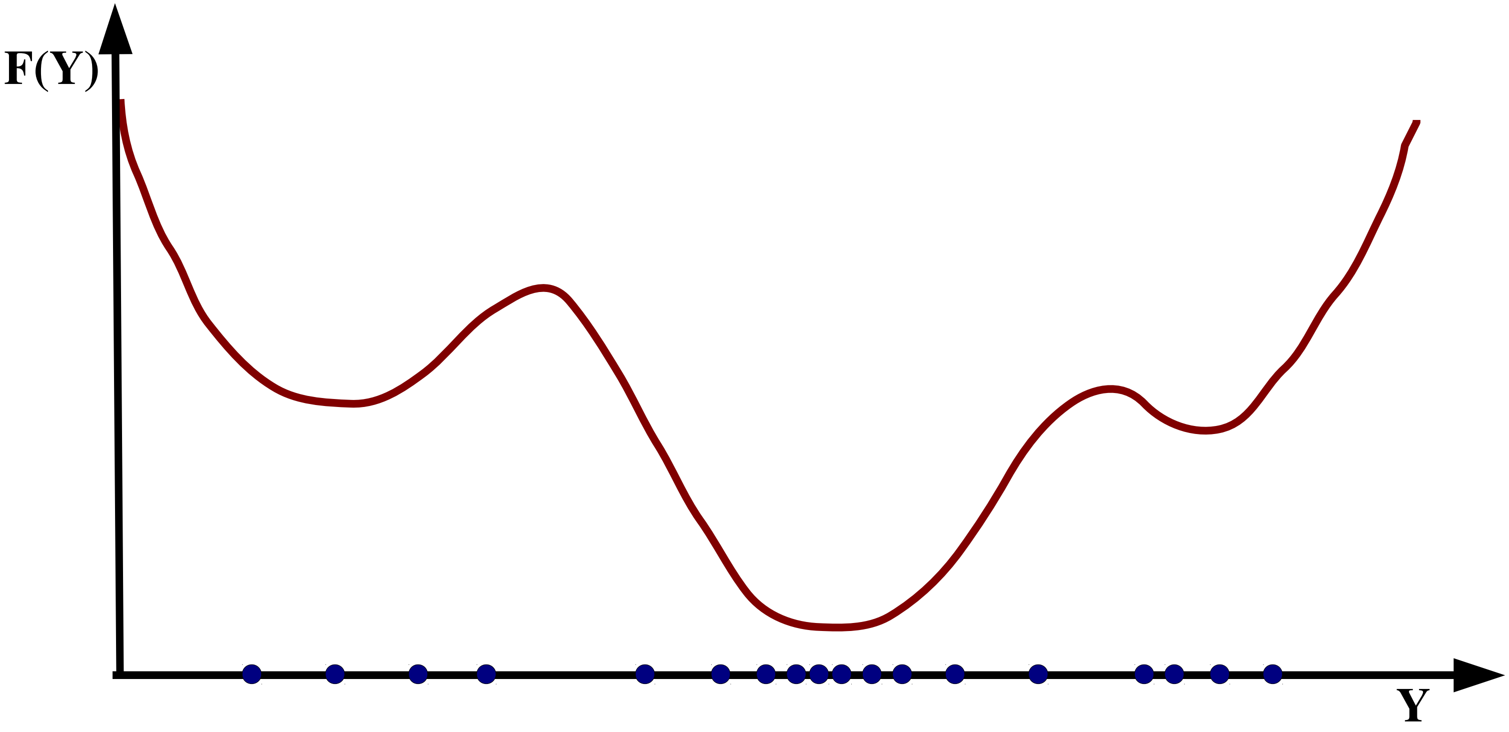
MODEL

W

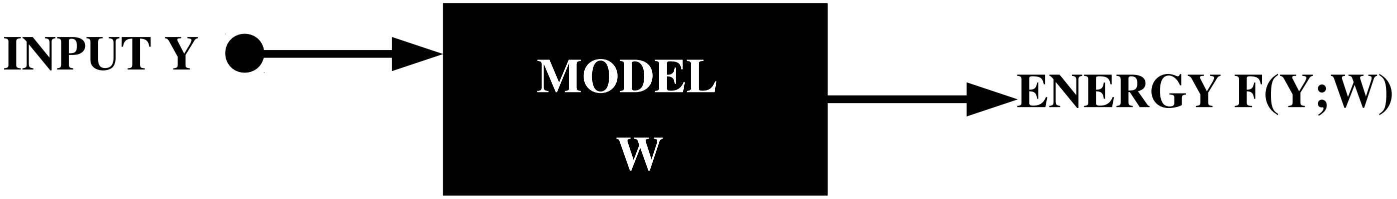
ENERGY F(Y;W)

- **GOAL:** make F(Y,W) lower around areas of high data density

ENERGY AFTER TRAINING

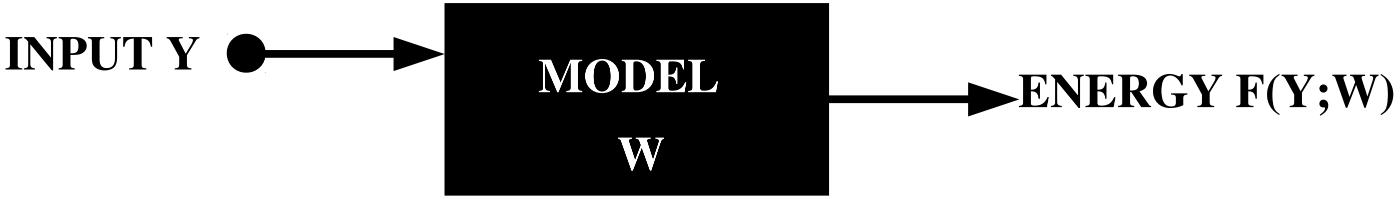


Energy-Based Framework for Unsupervised Learning



- GOAL: make F(Y,W) lower around areas of high data density
- Training the model by minimizing a loss functional L[F(. , W)]

Energy-Based Framework for Unsupervised Learning

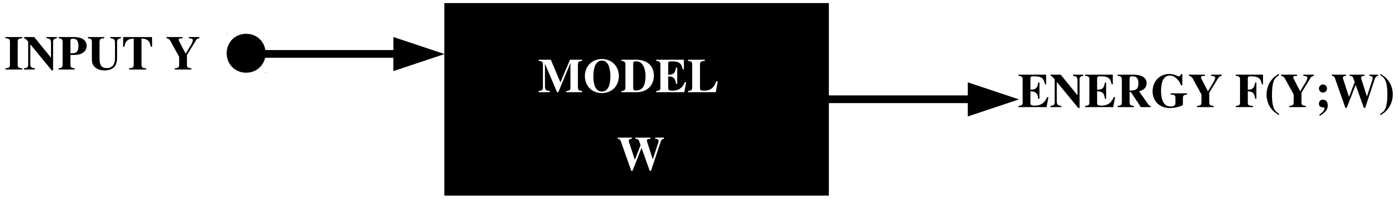


- **GOAL:** **make F(Y,W) lower around areas of high data density**
- **Contrastive loss**
  - Pushes down on the energy of data points
  - Pushes on the energy of everything else

$$L(W) = L(F(Y;W), F(\bar{Y};W))$$

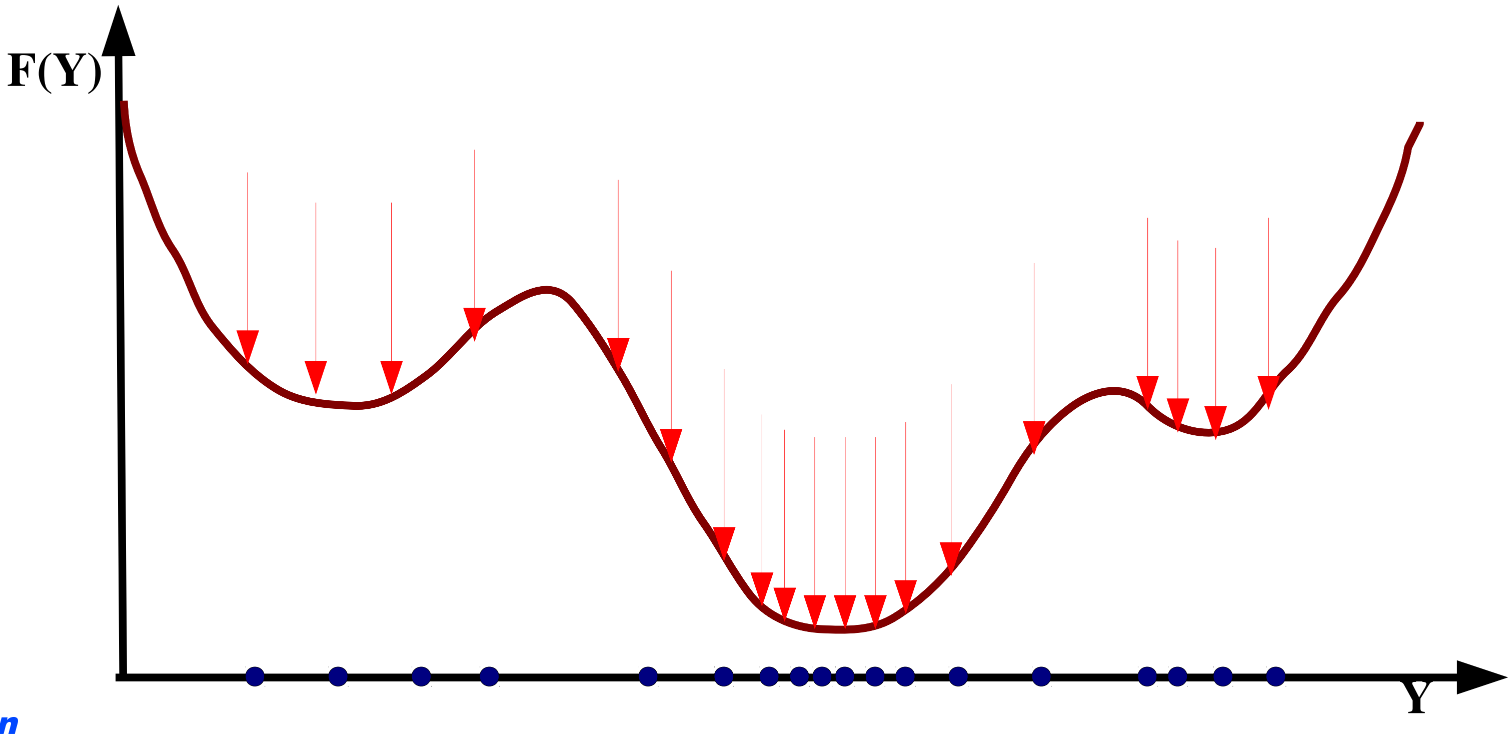
- **L(a,b): increasing function of a, decreasing function of b.**
- **Y: data point from the training set**
- **$\bar{Y}$: "fantasy" point outside of the region of high data density**

Energy-Based Framework for Unsupervised Learning

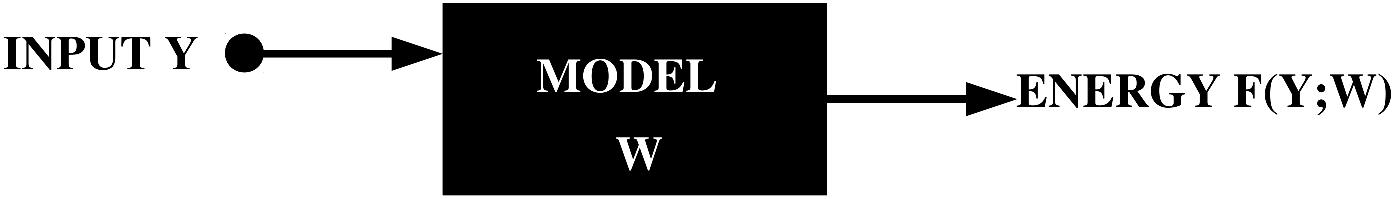


- Contrastive loss

$$L(W) = L(F(Y;W), F(\bar{Y};W))$$

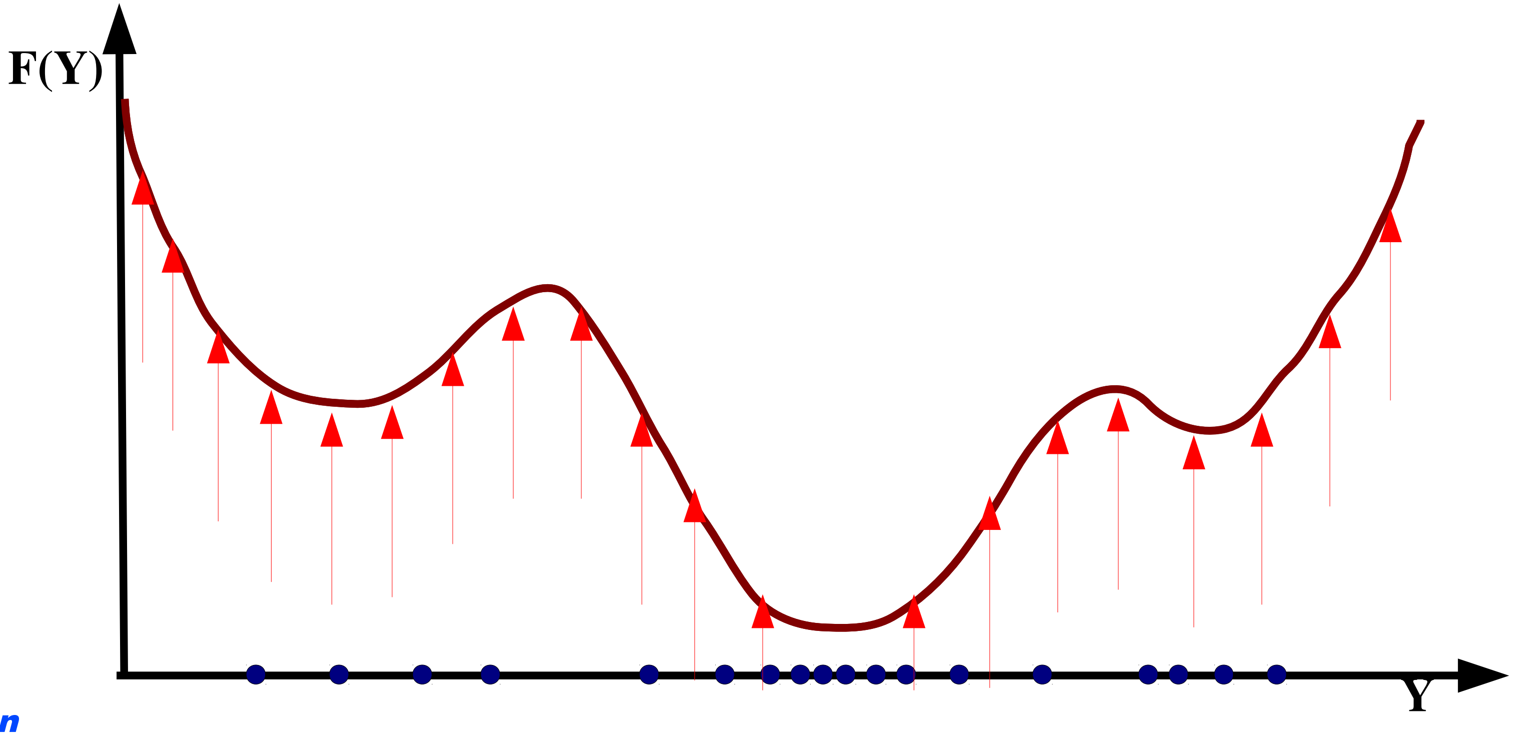


Energy-Based Framework for Unsupervised Learning



- Contrastive loss

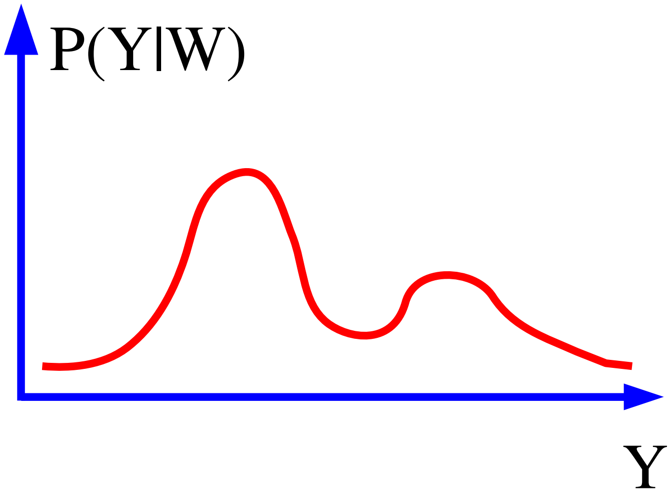
$$L(W)=L(F(Y;W),F(\bar{Y};W))$$



Each Stage is Trained as an Estimator of the Input Density

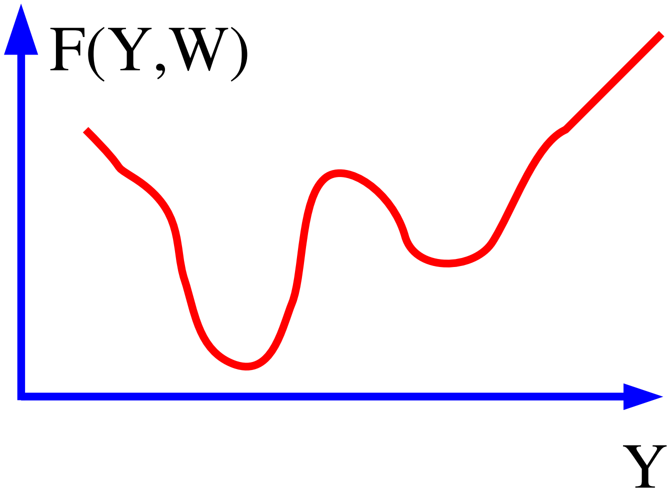
Probabilistic View:

- Produce a probability density function that:
- has high value in regions of high sample density
- has low value everywhere else (integral = 1).



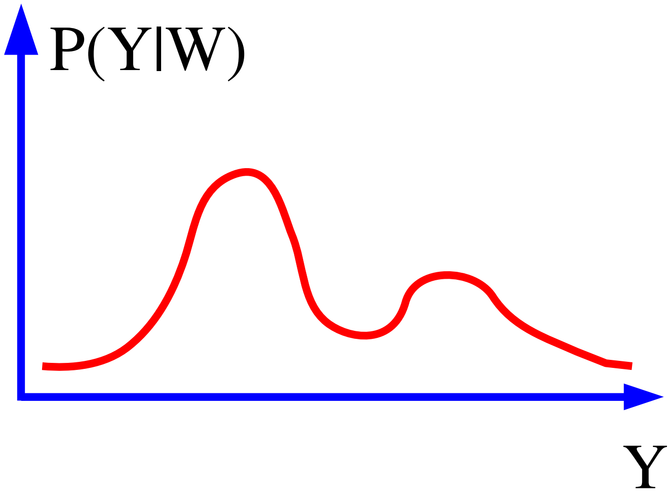
Energy-Based View:

- produce an energy function F(Y,W) that:
- has low value in regions of high sample density
- has high(er) value everywhere else

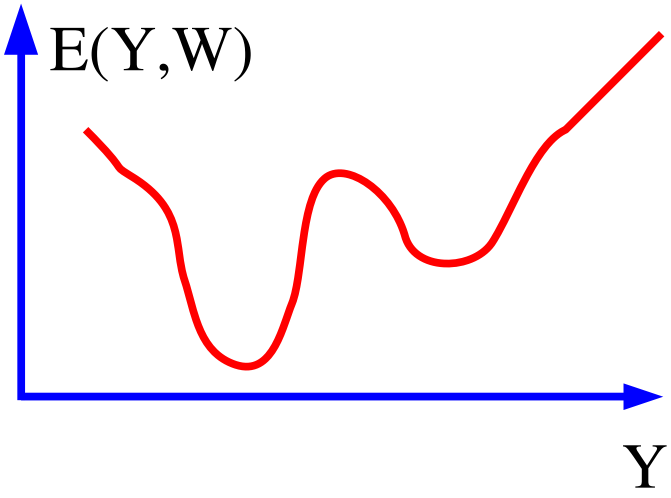


Energy <-> Probability

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

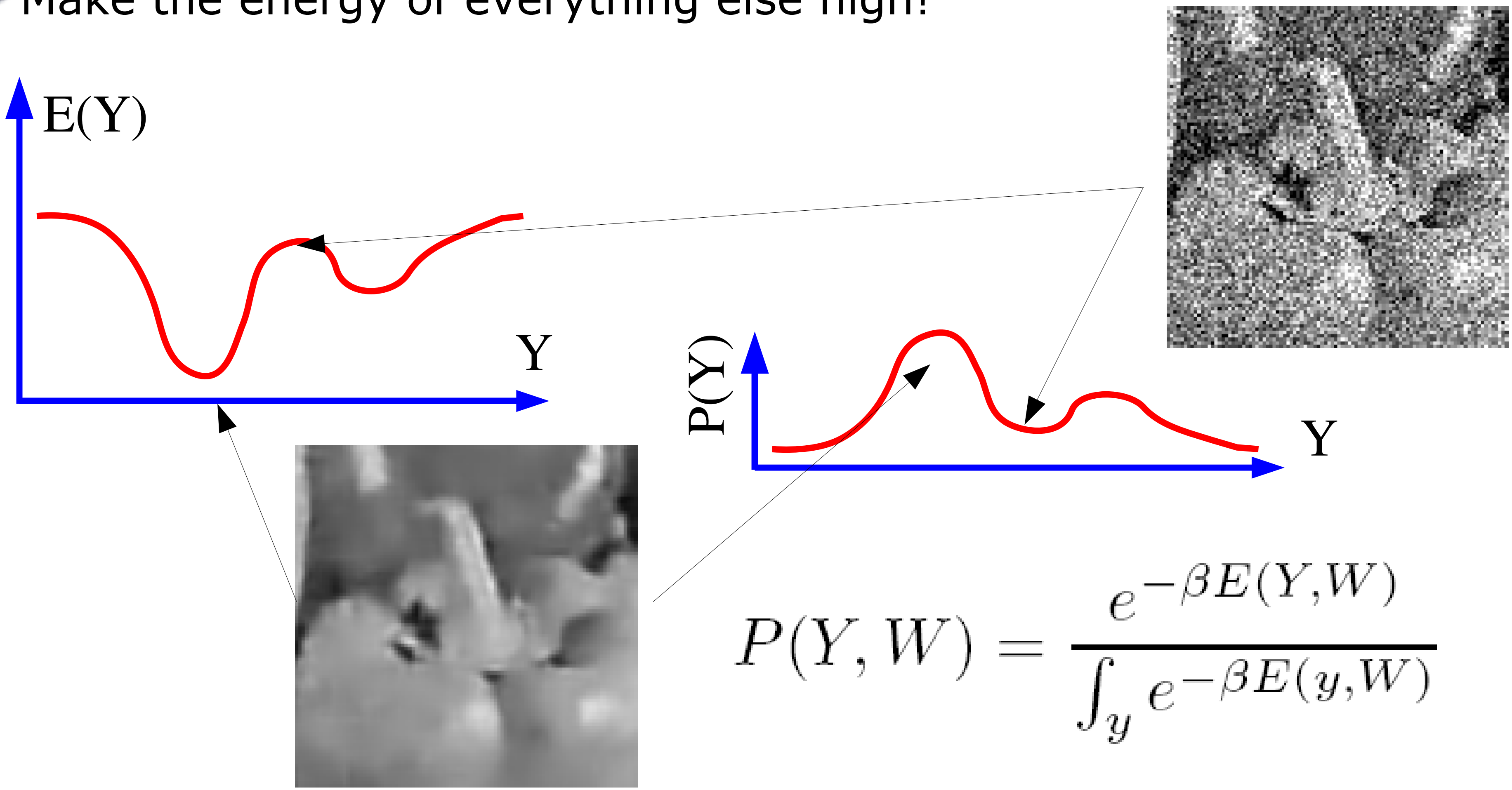


$$E(Y,W) \propto -\log P(Y|W)$$



The Intractable Normalization Problem

- **Example: Image Patches**
- **Learning:**
  - Make the energy of every “natural image” patch low
  - Make the energy of everything else high!



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

Training an Energy-Based Model to Approximate a Density

Maximizing P(Y|W) on training samples

make this big

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

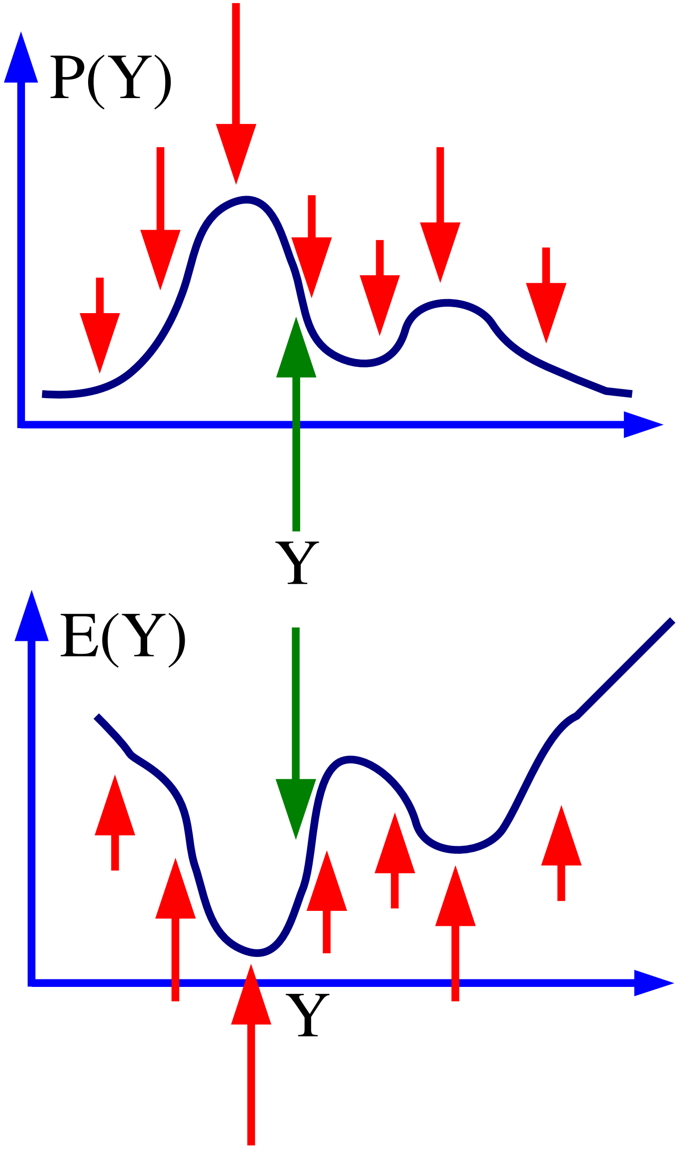
make this small

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

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

make this small

make this big



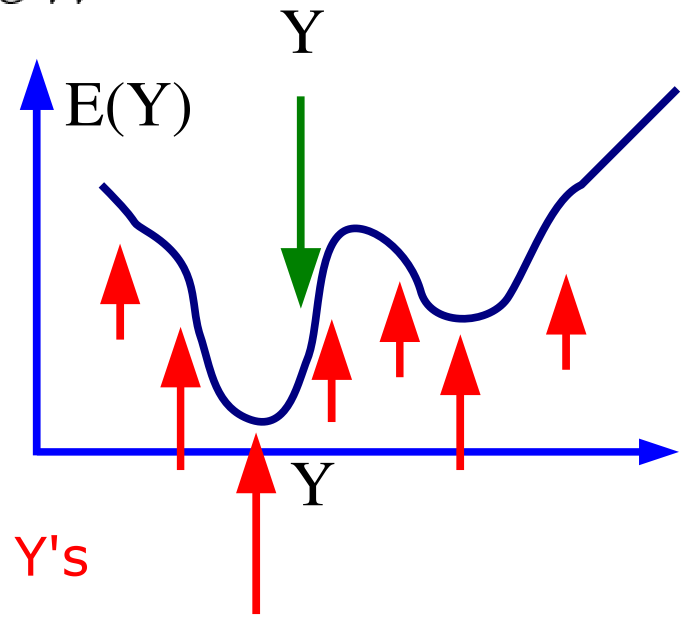
Training an Energy-Based Model with Gradient Descent

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

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

- **Gradient descent:**

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



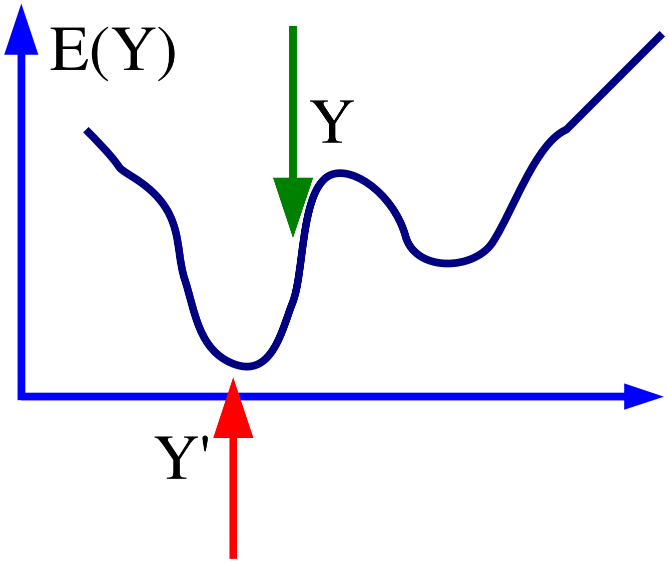
Pushes down on the energy of the samples

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

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

Contrastive Divergence Trick [Hinton 2000]

- **push down on the energy of the training sample Y**
- **Pick a sample of low energy Y' near the training sample, and pull up its energy**
  - this digs a trench in the energy surface around the training samples



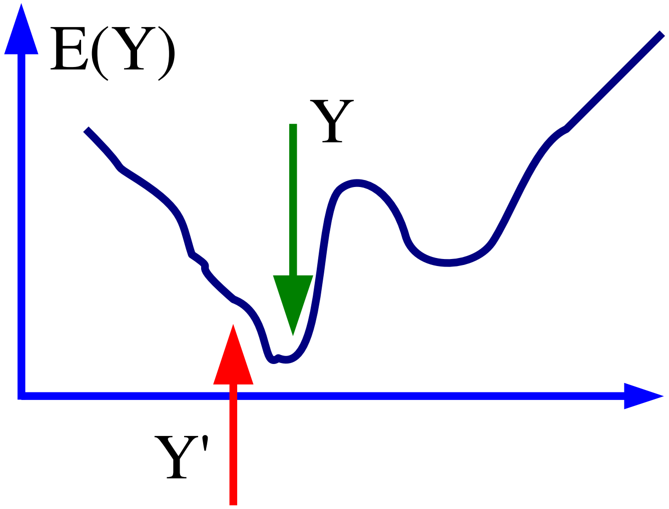
$$W \leftarrow W - \eta \frac{\partial E(Y,W)}{\partial W} + \eta \frac{\partial E(Y',W)}{\partial W}$$

Pushes down on the energy of the training sample Y

Pulls up on the energy Y'

Contrastive Divergence Trick [Hinton 2000]

- push down on the energy of the training sample Y
- Pick a sample of low energy Y' near the training sample, and pull up its energy
  - this digs a trench in the energy surface around the training samples

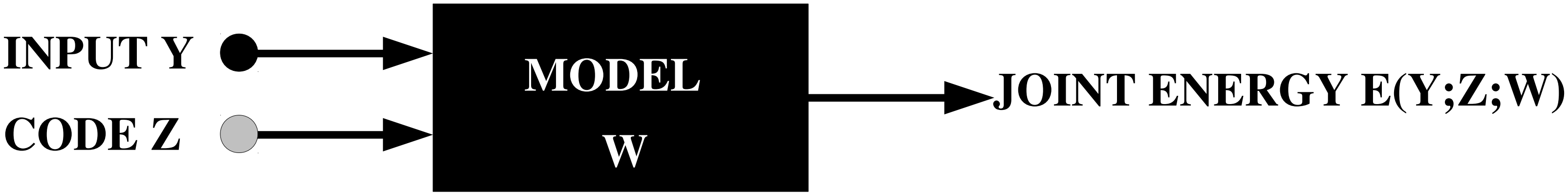


$$W \leftarrow W - \eta \frac{\partial E(Y,W)}{\partial W} + \eta \frac{\partial E(Y',W)}{\partial W}$$

Pushes down on the energy of the training sample Y

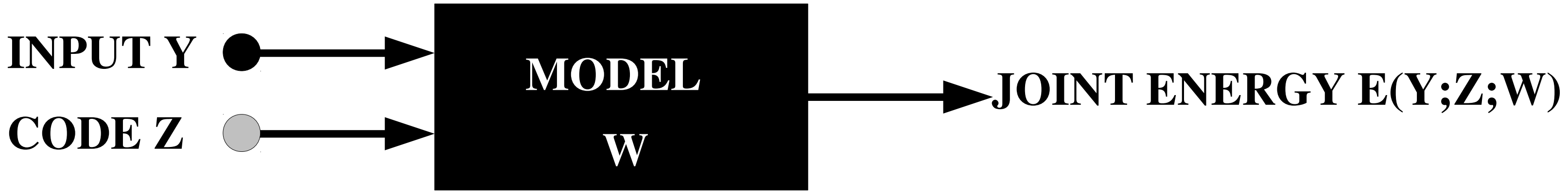
Pulls up on the energy Y'

Energy-Based Model Framework

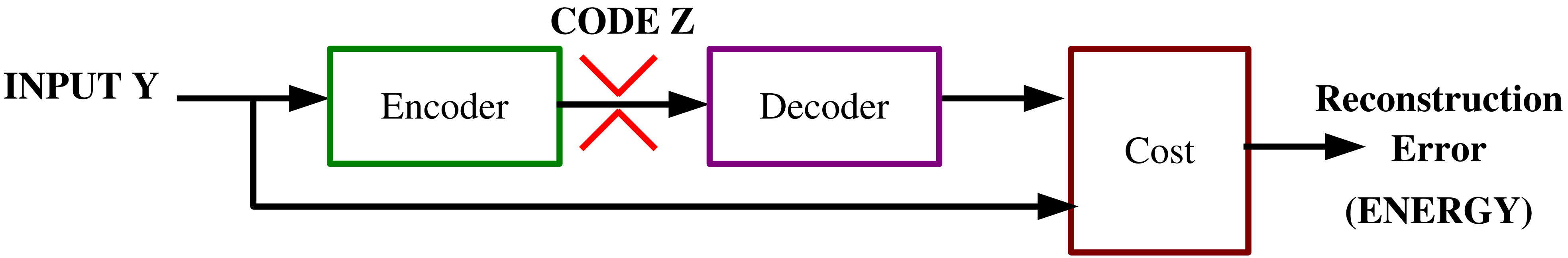


- Restrict **information content of internal representation**
  - assume that input is reconstructed from code
  - inference determines the value of Z and F(Y;W)

Getting Around The Intractability Problem



- **MAIN INSIGHT:**
- **Assume that the input is reconstructed from an internal code Z**
- **Assume that the energy measures the reconstruction error**
- **Restricting the information content of the code will automatically push up the energy outside of regions of high data density**

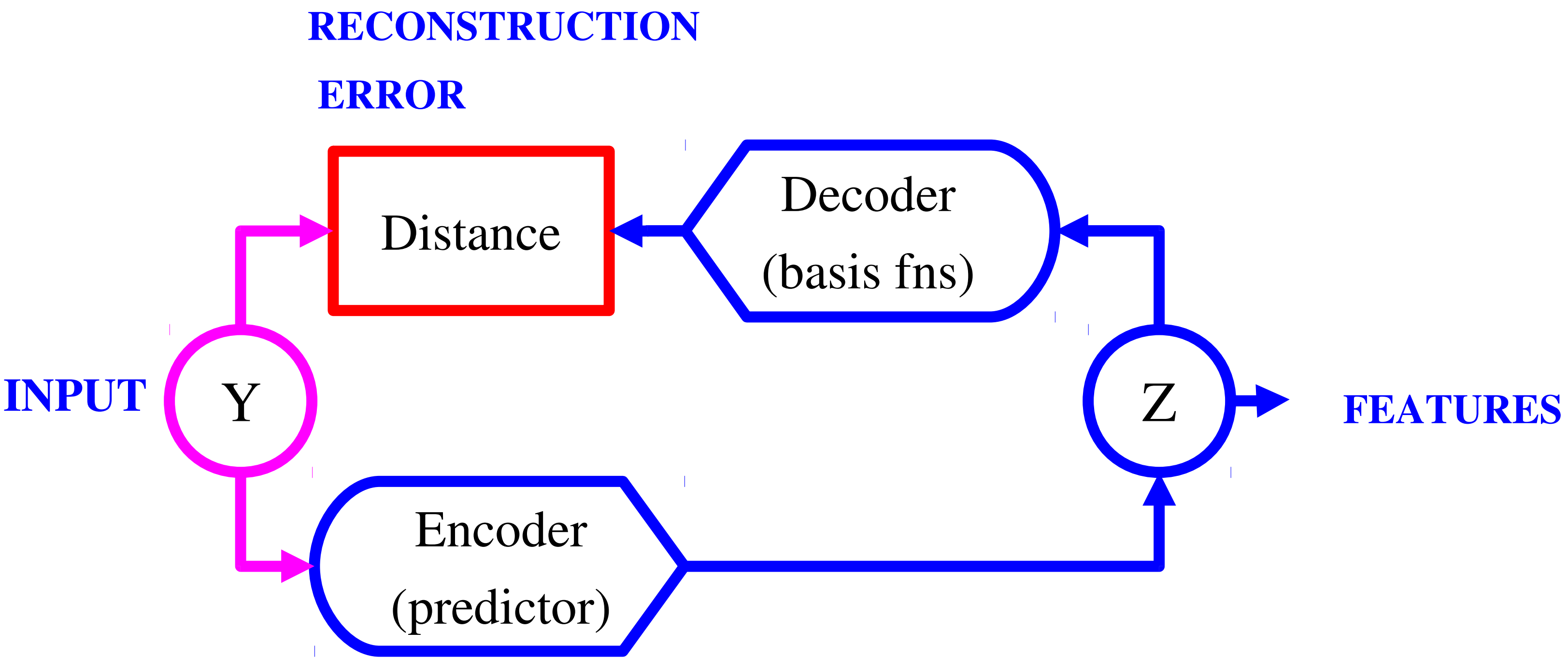


How do we push up on the energy of everything else?

- **Solution 1: contrastive divergence [Hinton 2000]**
  - Move away from a training sample a bit
  - Push up on that
- **Solution 2: score matching [Hyvarinen]**
  - On the training samples: minimize the gradient of the energy, and maximize the trace of its Hessian.
- **Solution 3: denoising auto-encoder [Vincent & Bengio 2008]**
  - Train the inference dynamics to map noisy samples to clean samples (not really energy based, but simple and efficient)
- **Solution 4: MAIN INSIGHT! [Ranzato, ..., LeCun AI-Stat 2007]**
  - **Restrict the information content of the code (features) Z**
  - If the code Z can only take a few different configurations, only a correspondingly small number of Ys can be perfectly reconstructed
  - Idea: impose a sparsity prior on Z
  - This is reminiscent of sparse coding [Olshausen & Field 1997]

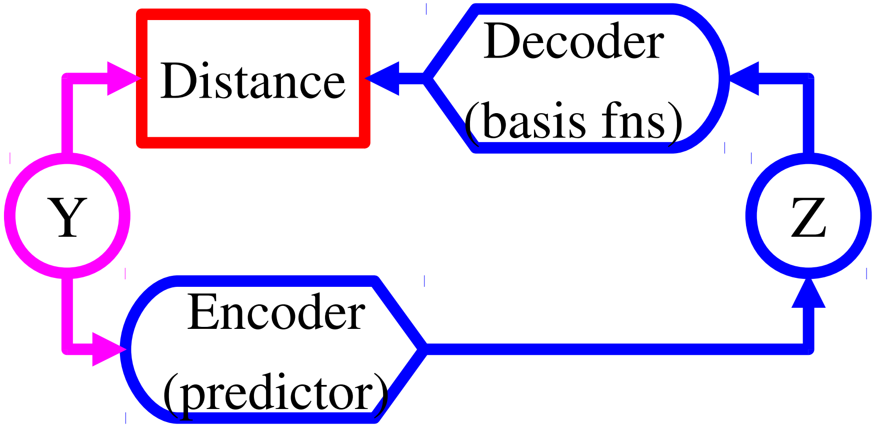
The Encoder/Decoder Architecture

- **Each stage is composed of** [Hinton 05, Bengio 06, LeCun 06, Ng 07]
  - an encoder that produces a feature vector from the input
  - a decoder that reconstruct the input from the feature vector
    - PCA is a special case (linear encoder and decoder)



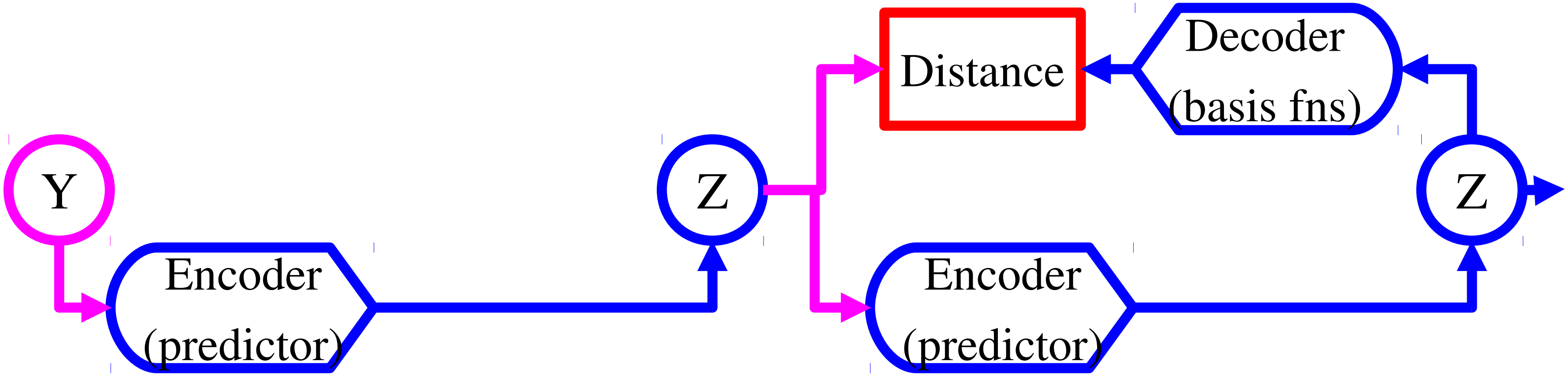
Deep Learning: Stack of Encoder/Decoders

- Train each stage one after the other
- 1. Train the first stage



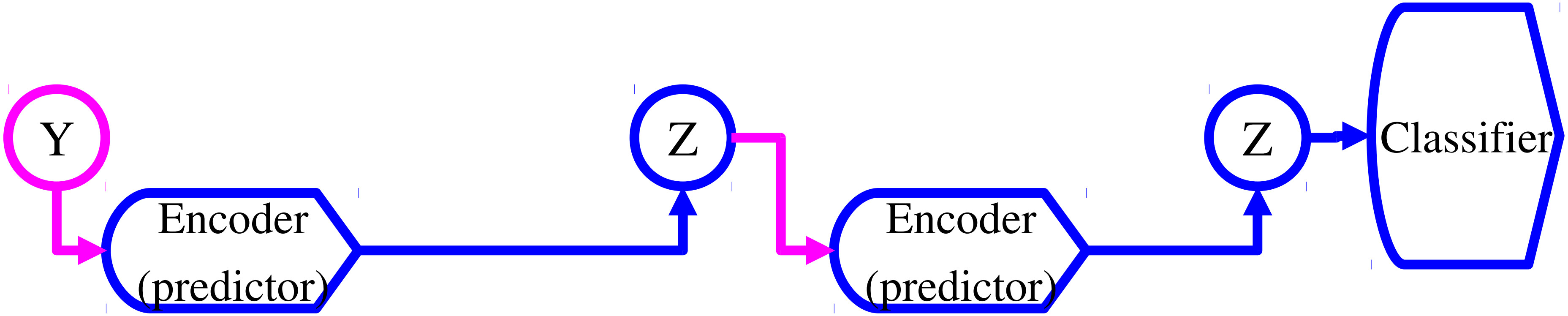
Deep Learning: Stack of Encoder/Decoders

- Train each stage one after the other
- 2. Remove the decoder, and train the second Stage



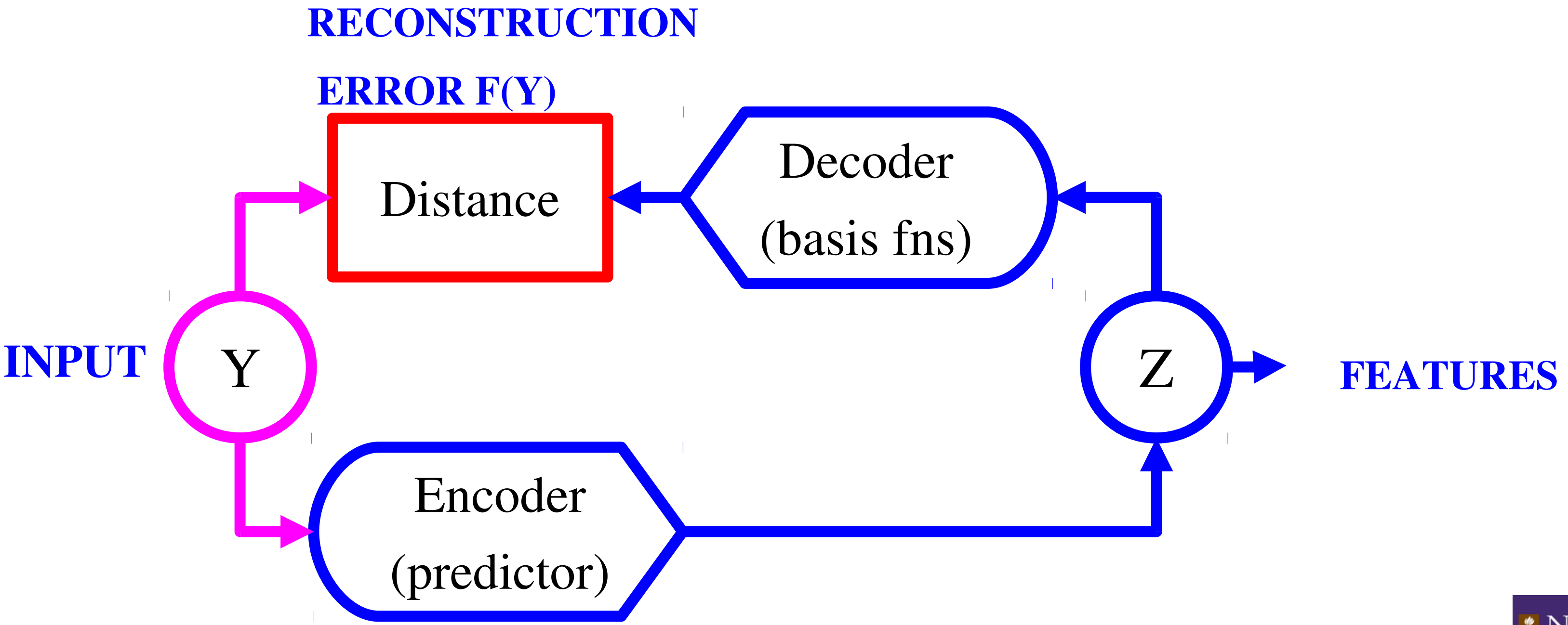
Deep Learning: Stack of Encoder/Decoders

- Train each stage one after the other
- 3. Remove the 2nd stage decoder, and train a supervised classifier on top
- 4. Refine the entire system with supervised learning
    - e.g. using gradient descent / backprop



Training an Encoder/Decoder Module

- **Define the Energy F(Y) as the reconstruction error**
  - Example: F(Y) = || Y – Decoder(Encoder(Y)) $||^2$
- **Probabilistic Training, given a training set (Y1, Y2.......)**
  - Interpret the energy F(Y) as a -log P(Y) (unnormalized)
  - Train the encoder/decoder to maximize the prob of the data
- **Train the encoder/decoder so that:**
  - F(Y) is small in regions of high data density (good reconstruction)
  - F(Y) is large in regions of low data density (bad reconstruction)



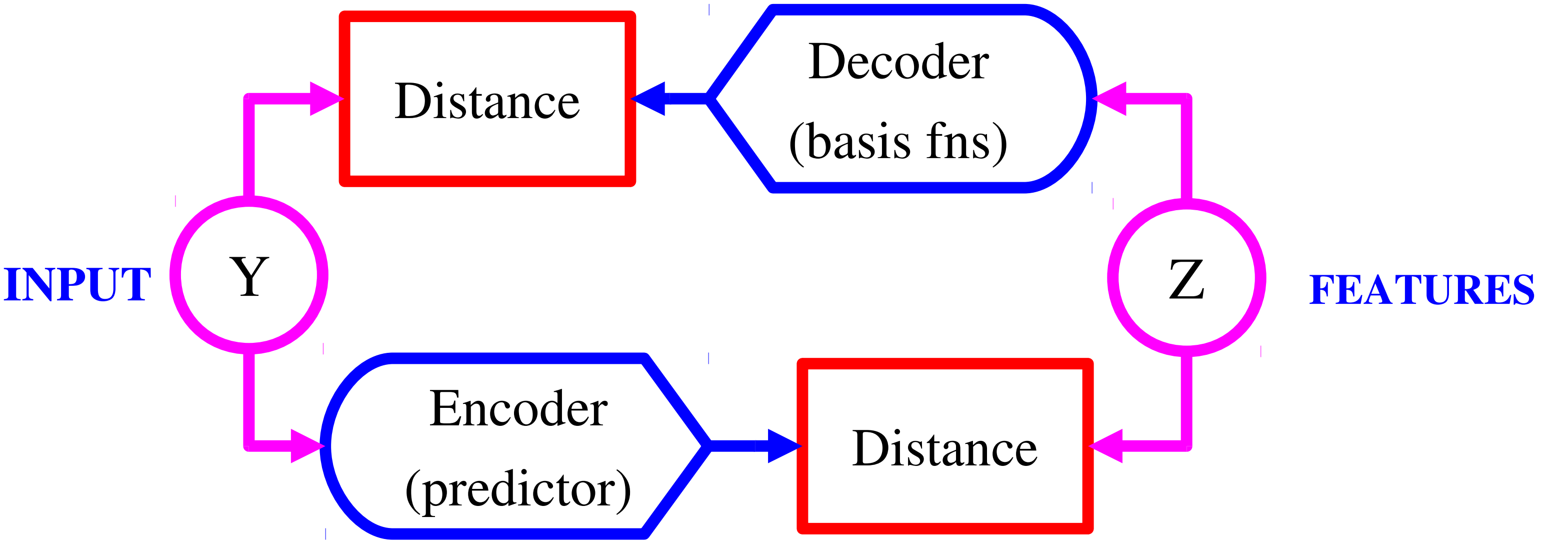
Encoder-Decoder: feature Z is a latent variable

- **Energy:**

$$E(Y,Z) = \mathrm{Dist}[Y, \mathrm{Dec}(Z)] + \mathrm{Dist}[Z, \mathrm{Enc}(Y)]$$

- **Inference through minimization or marginalization**

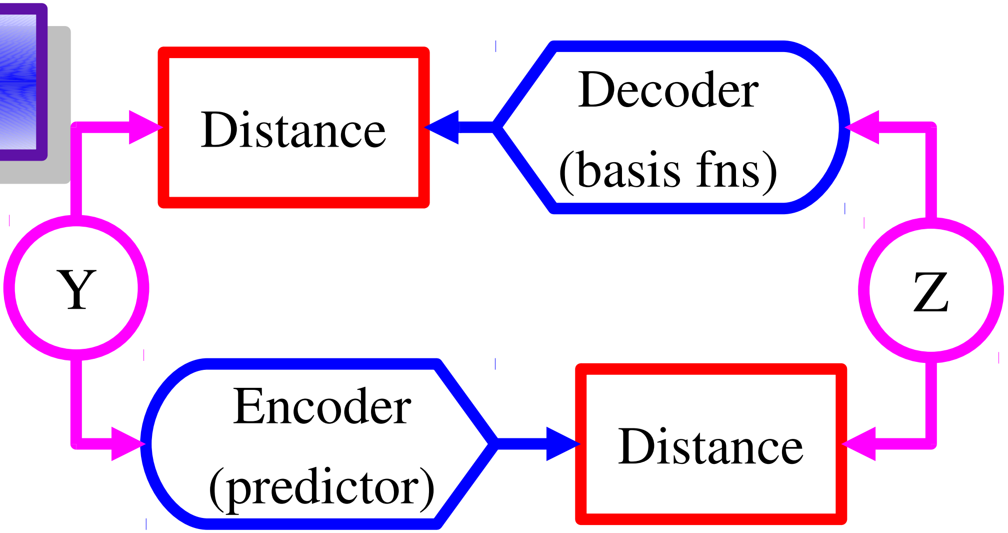
$$F(Y) = \min_z E(Y,z) \quad \text{or} \quad F(Y) = -\frac{1}{\beta}\log\int_z e^{-\beta E(Y,z)}$$



Restricted Boltzmann Machines

[Hinton & Salakhutdinov 2005]

- **Y and Z are binary**
- **Enc and Dec are linear**
- **Distance is negative dot product**



$$E(Y,Z) = \text{Dist}[Y, \text{Dec}(Z)] + \text{Dist}[Z, \text{Enc}(Y)]$$

$$\text{Enc}(Y) = -W.Y \quad \text{Dist}(Z, W.Y) = -\frac{1}{2} Z^T.W.Y$$

$$\text{Dec}(Y) = -W^T.Z \quad \text{Dist}(Y, E^T.Z) = -\frac{1}{2} Y^T.W^T.Z$$

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

Non-Linear Dimensionality Reduction with Stacked RBMs

- [Hinton and Salakhutdinov, Science 2006]

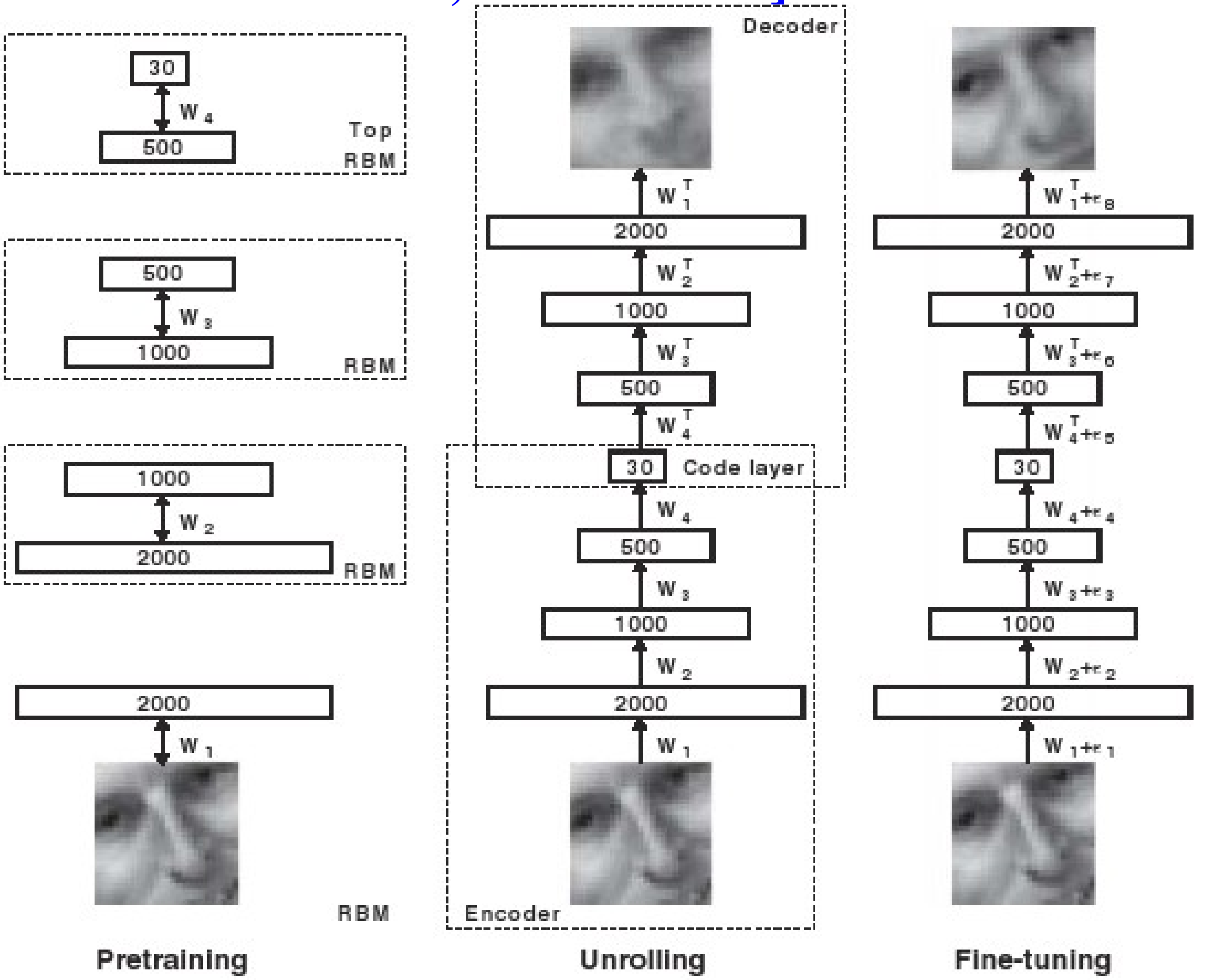


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create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

Non-Linear Dimensionality Reduction with Deep Learning

- [Hinton and Salakhutdinov, Science 2006]

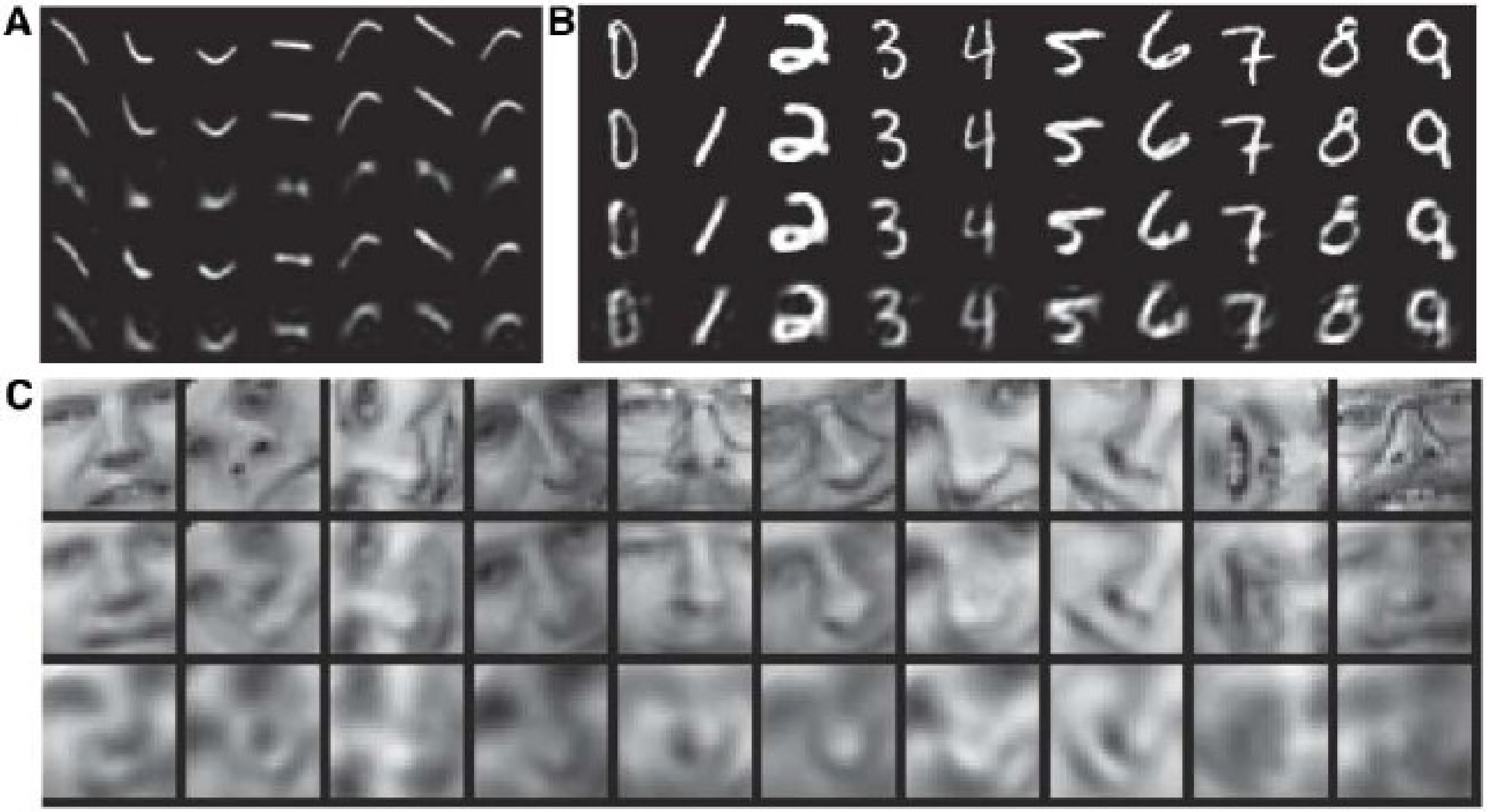


Fig. 2. (**A**) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (*8*) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error 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: MNIST

- [Hinton and Salakhutdinov, Science 2006]

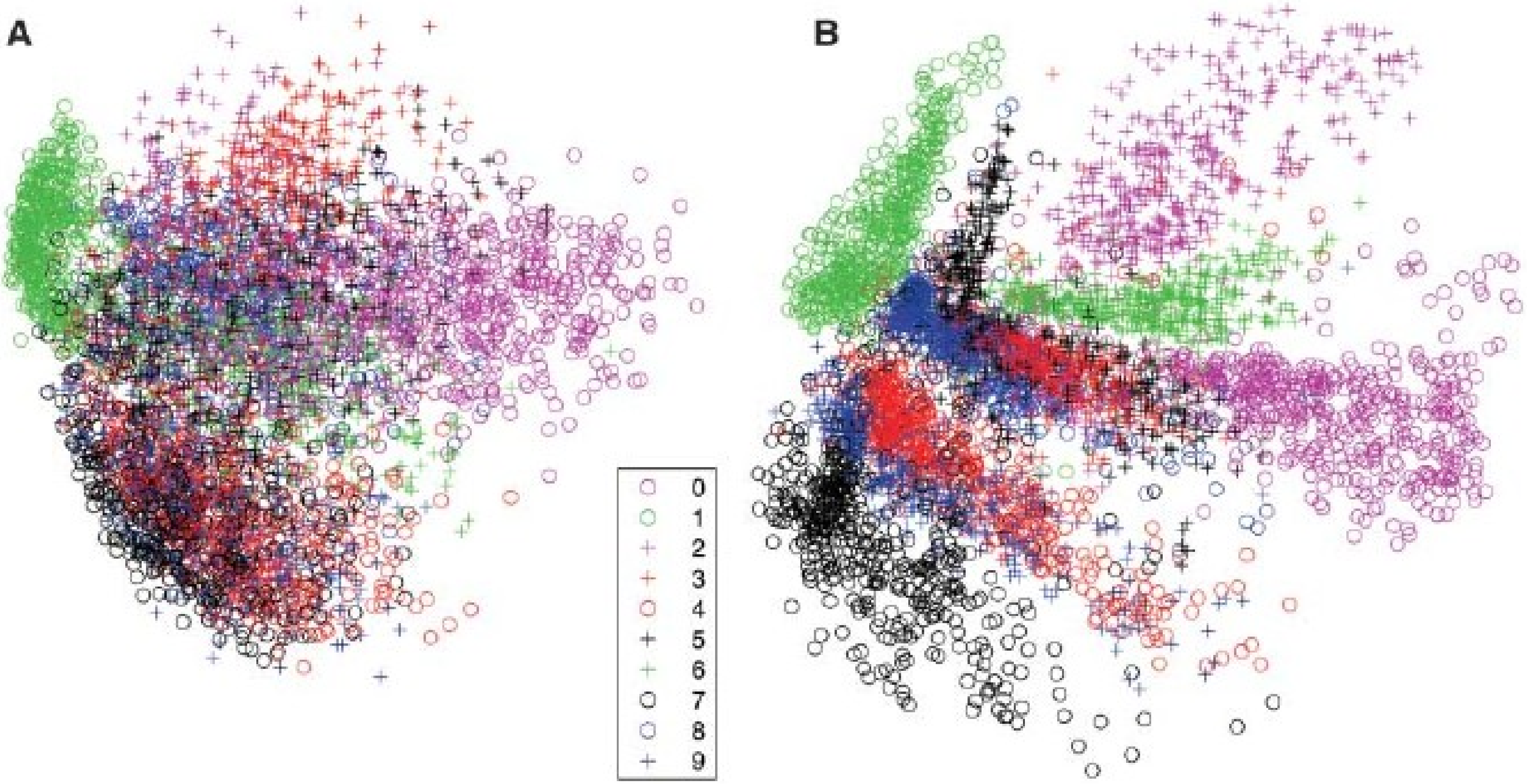
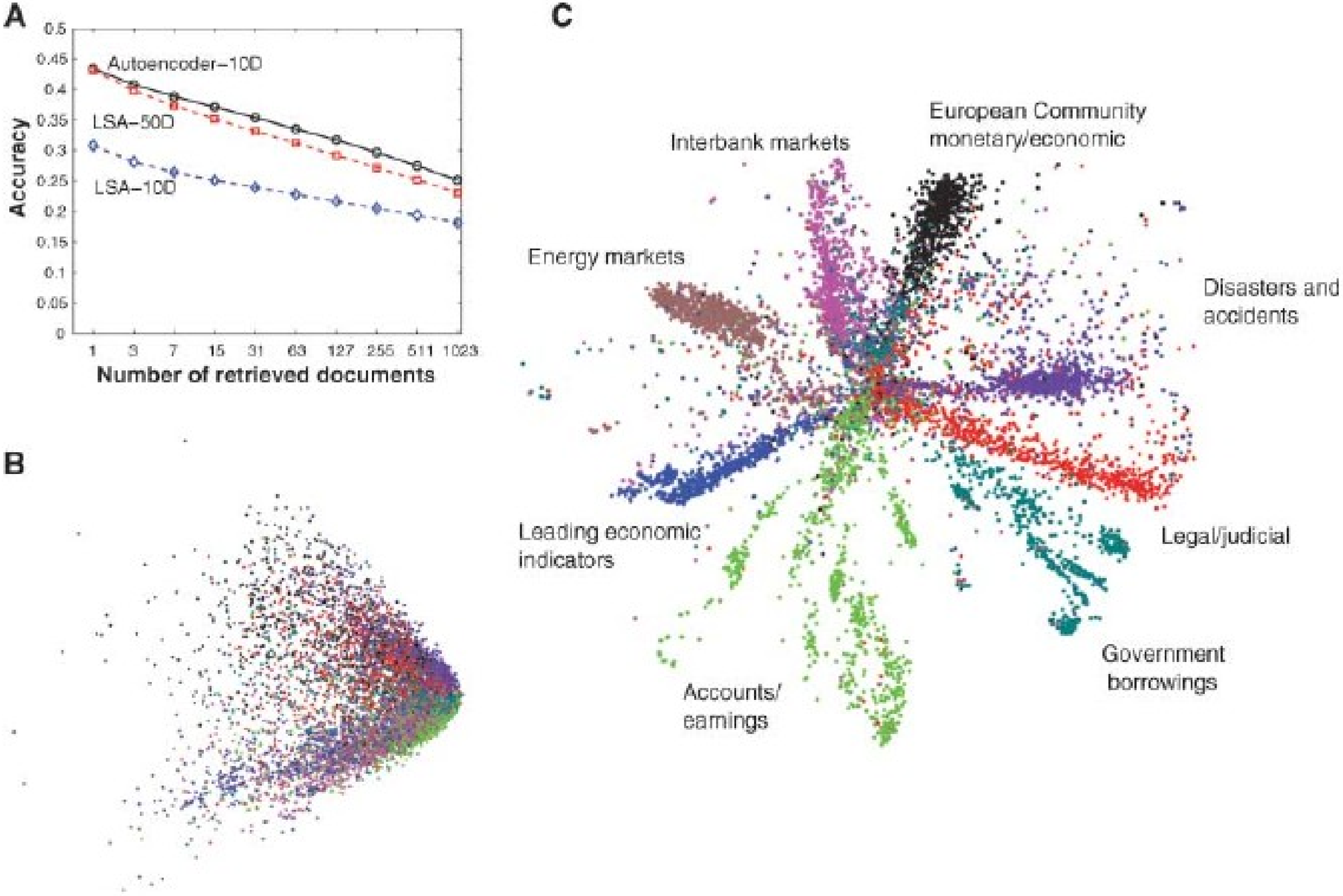


Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

Non-Linear Dimensionality Reduction: Text Retrieval

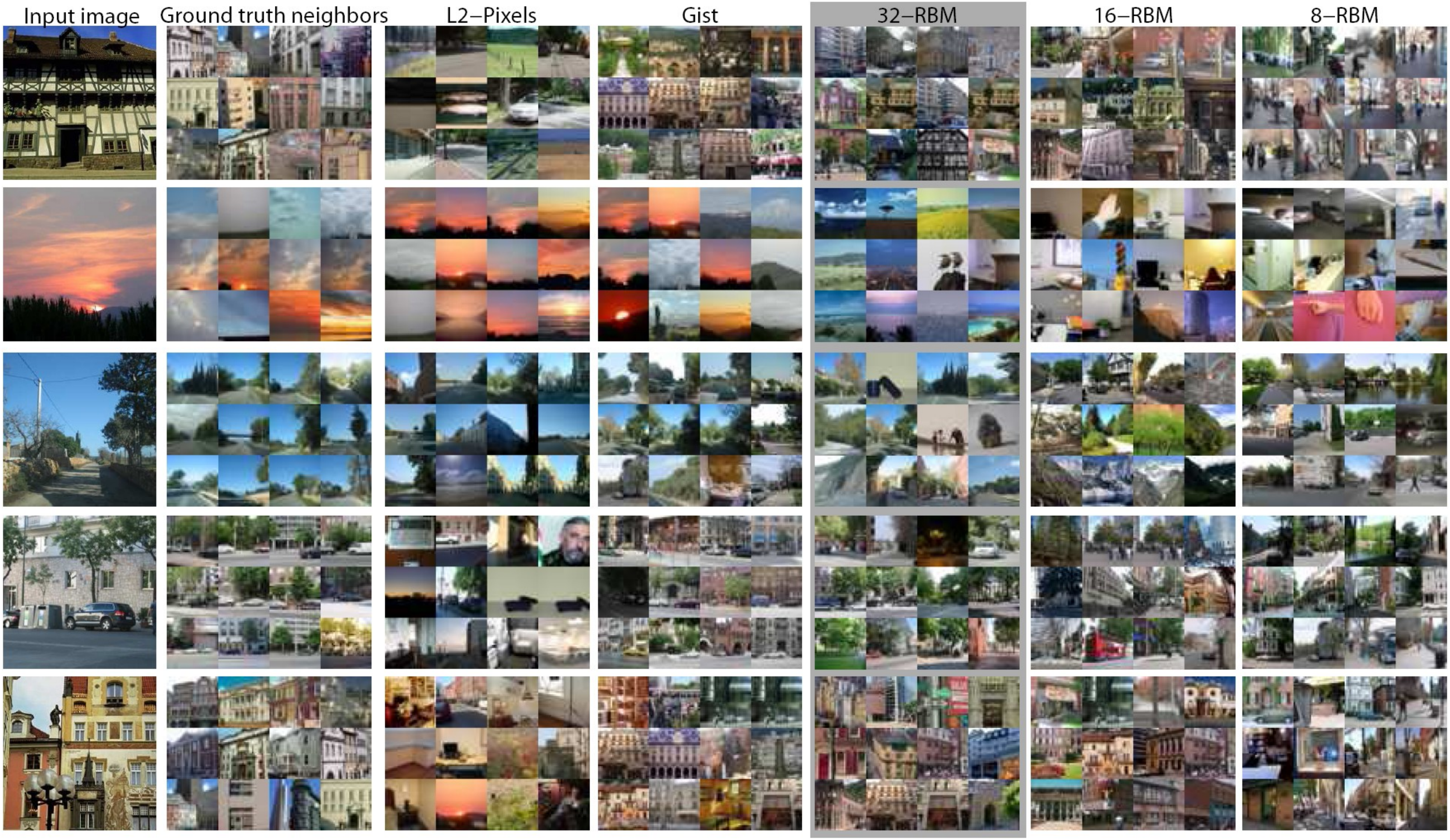
- [Hinton and Salakhutdinov, Science 2006]

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.

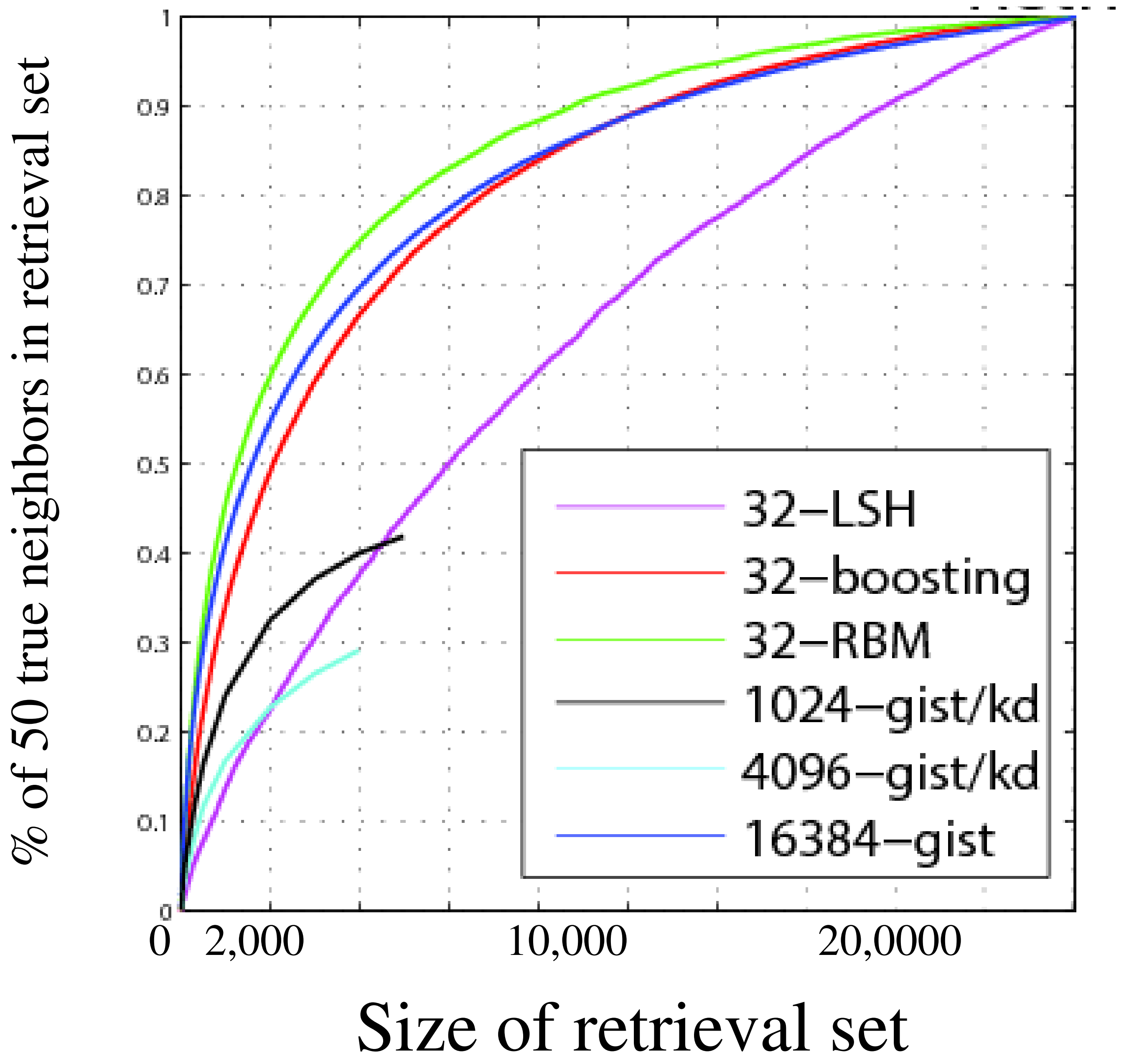


Examples of LabelMe retrieval using RBMs

- [Torralba, Fergus, Weiss, CVPR 2008]
- 12 closest neighbors under different distance metrics



LabelMe Retrieval Comparison of methods



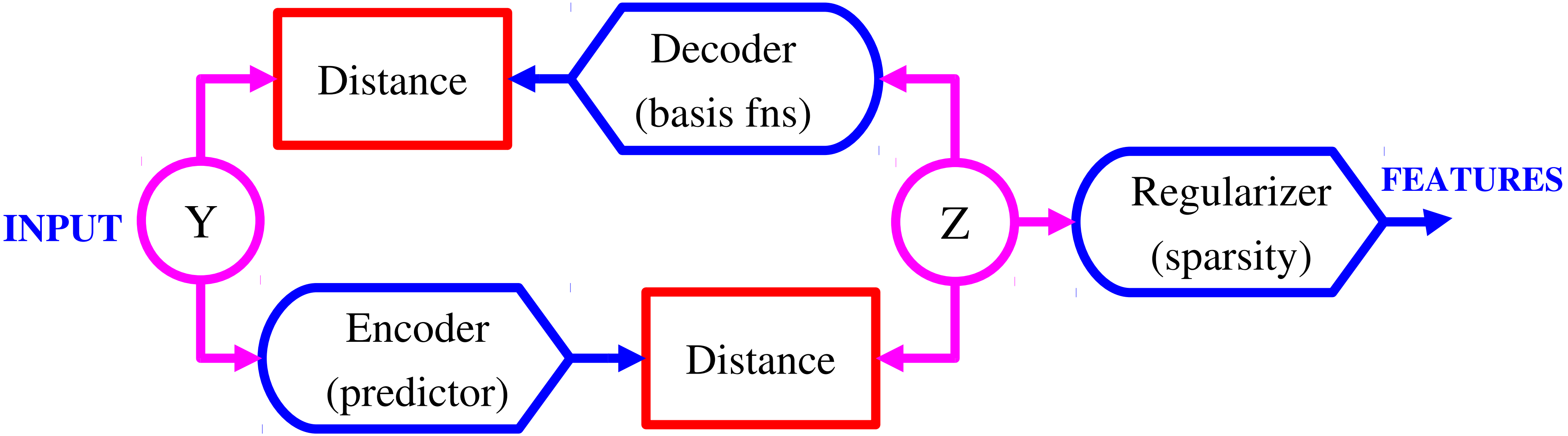
Encoder-Decoder with Sparsity

- **Energy:**

$$E(Y,Z) = \mathrm{Dist}[Y, \mathrm{Dec}(Z)] + \mathrm{Dist}[Z, \mathrm{Enc}(Y)] + \mathrm{Regularizer}(Z)$$

- **Inference through minimization or marginalization**

$$F(Y) = \min_z E(Y,z) \quad \text{or} \quad F(Y) = -\frac{1}{\beta}\log\int_z e^{-\beta E(Y,z)}$$

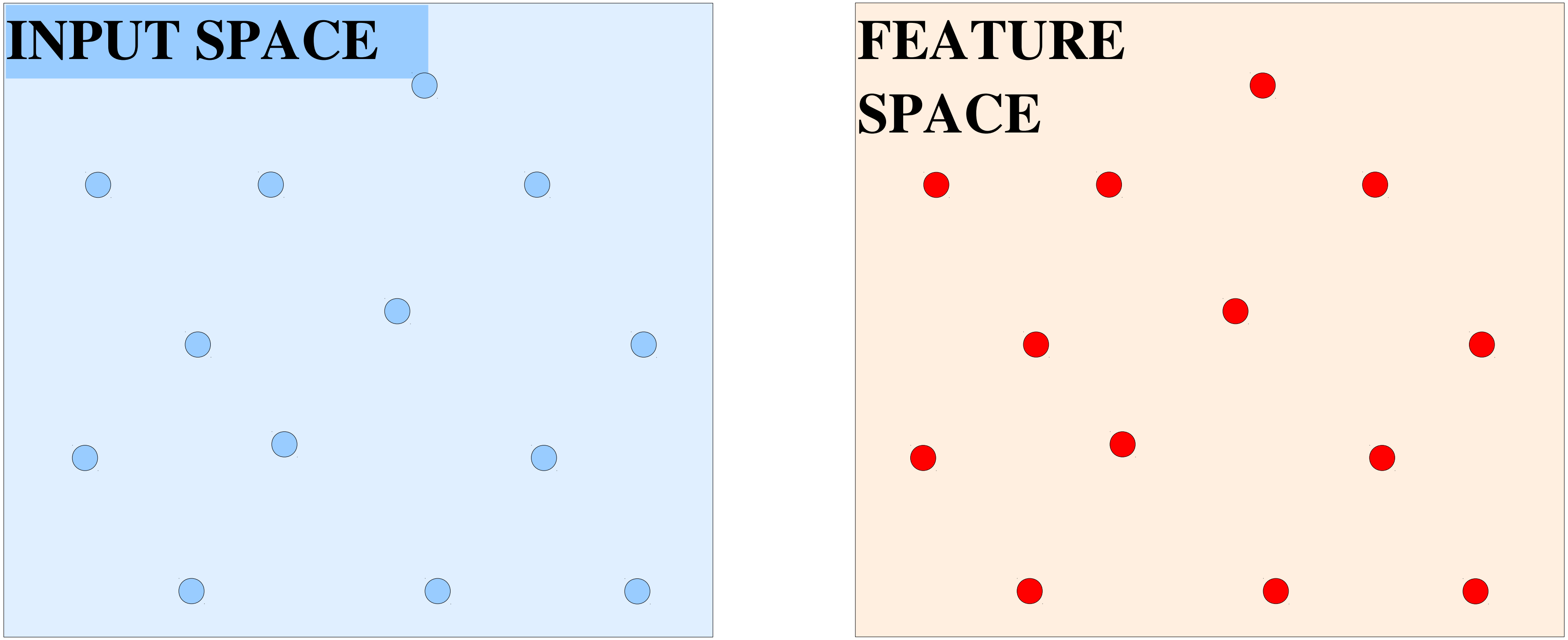


The Main Insight [Ranzato et al. AISTATS 2007]

- **If the information content of the feature vector is limited (e.g. by imposing sparsity constraints), the energy MUST be large in most of the space.**
  - pulling down on the energy of the training samples will necessarily make a groove
- **The volume of the space over which the energy is low is limited by the entropy of the feature vector**
  - Input vectors are reconstructed from feature vectors.
  - If few feature configurations are possible, few input vectors can be reconstructed properly

Why Limit the Information Content of the Code?

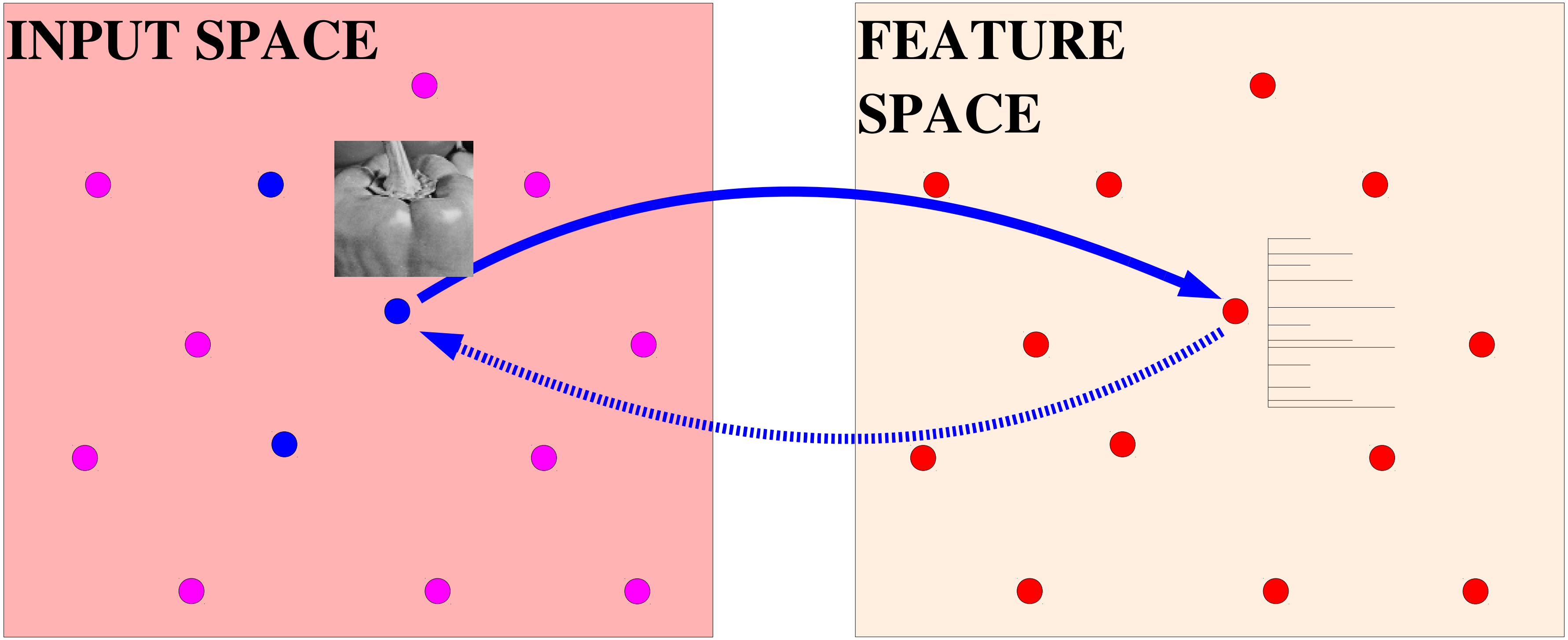
- ● Training sample
- ● Input vector which is NOT a training sample
- ● Feature vector



Why Limit the Information Content of the Code?

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

Training based on minimizing the reconstruction error over the training set



INPUT SPACE

FEATURE SPACE

Why Limit the Information Content of the Code?

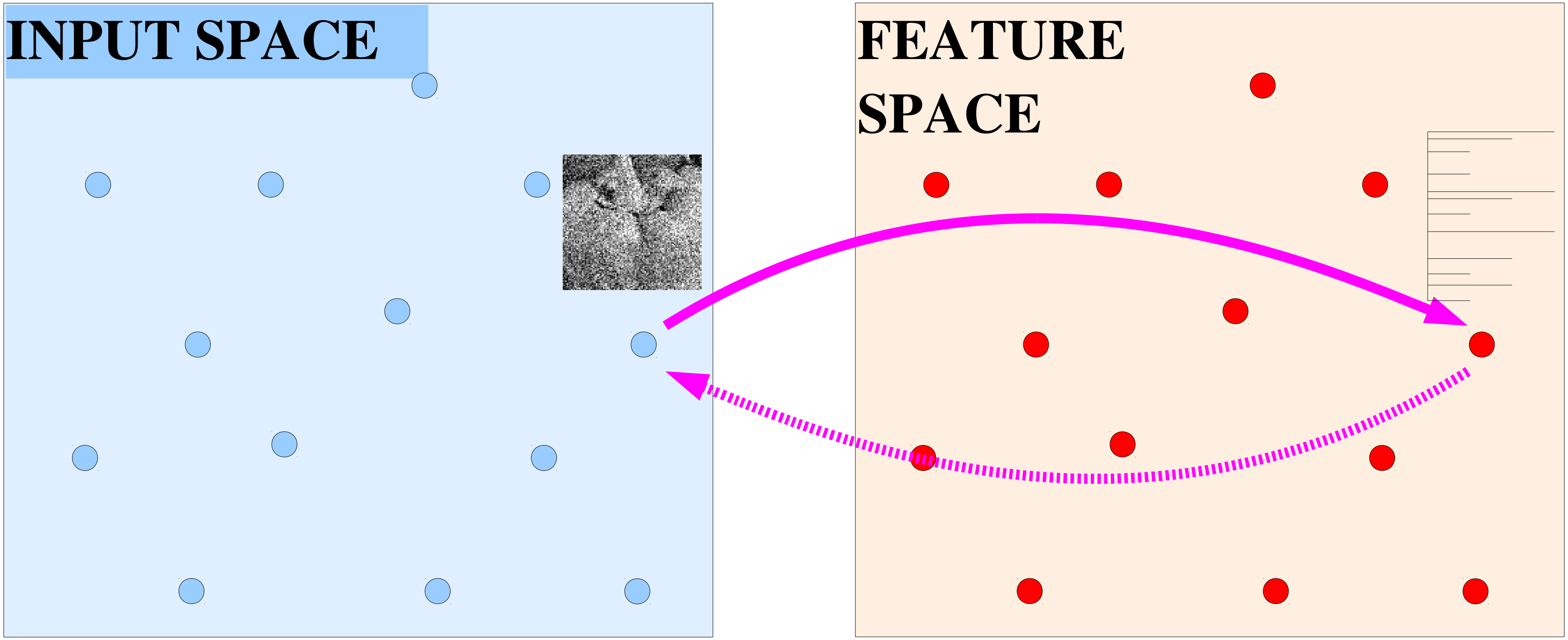
● Training sample

● Input vector which is NOT a training sample

● Feature vector

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

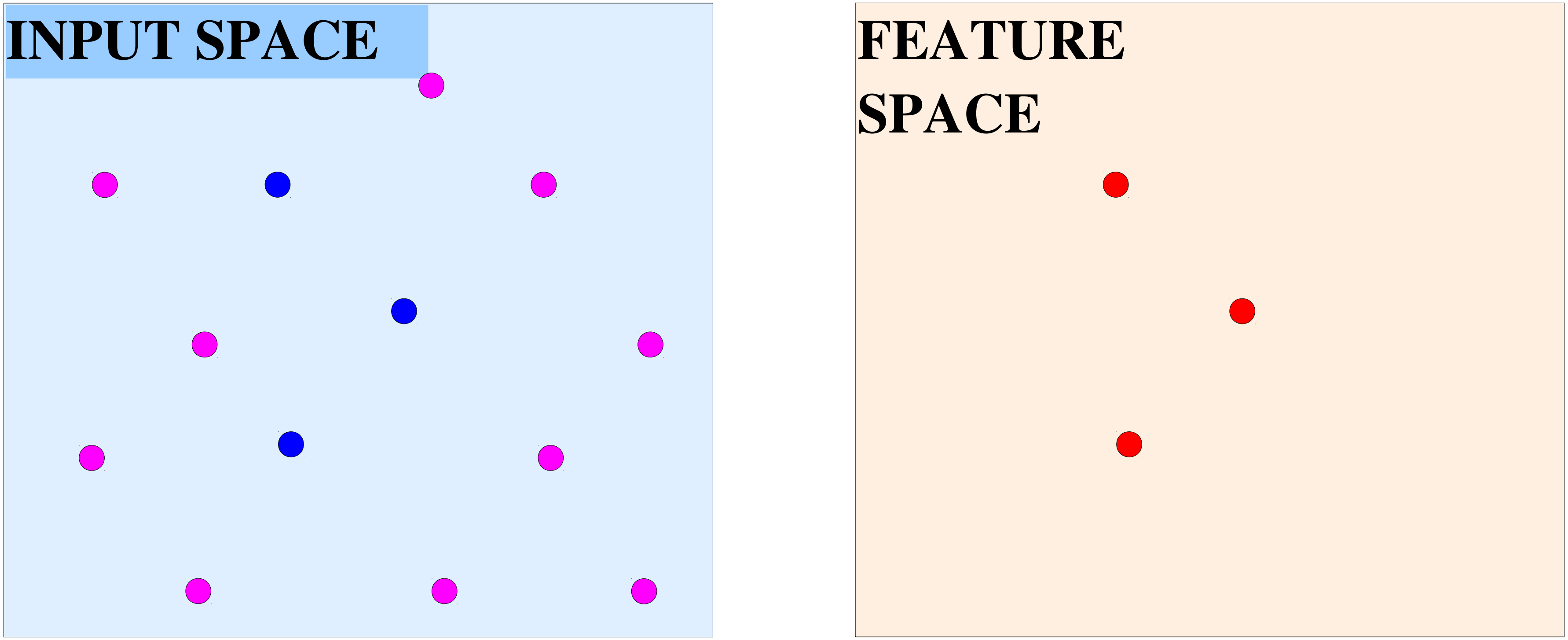
It just copies the data.



Why Limit the Information Content of the Code?

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

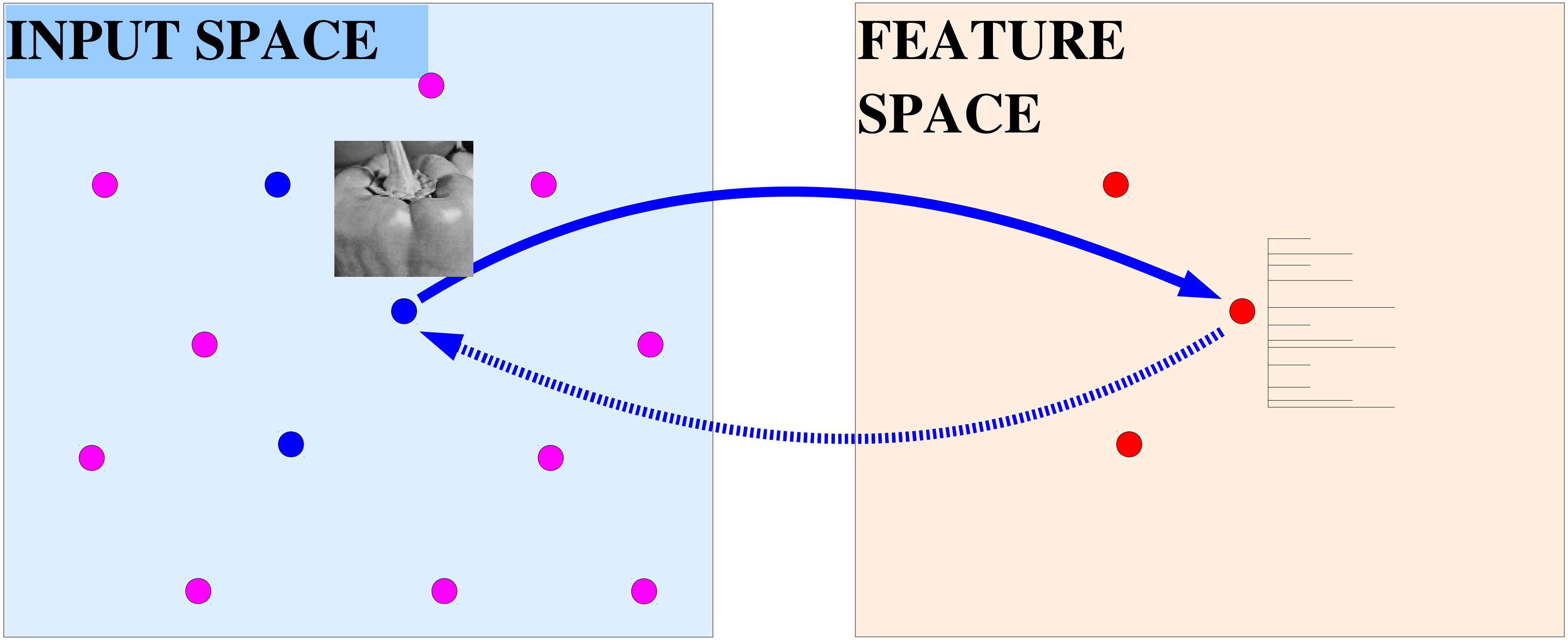
IDEA: reduce number of available codes.



Why Limit the Information Content of the Code?

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

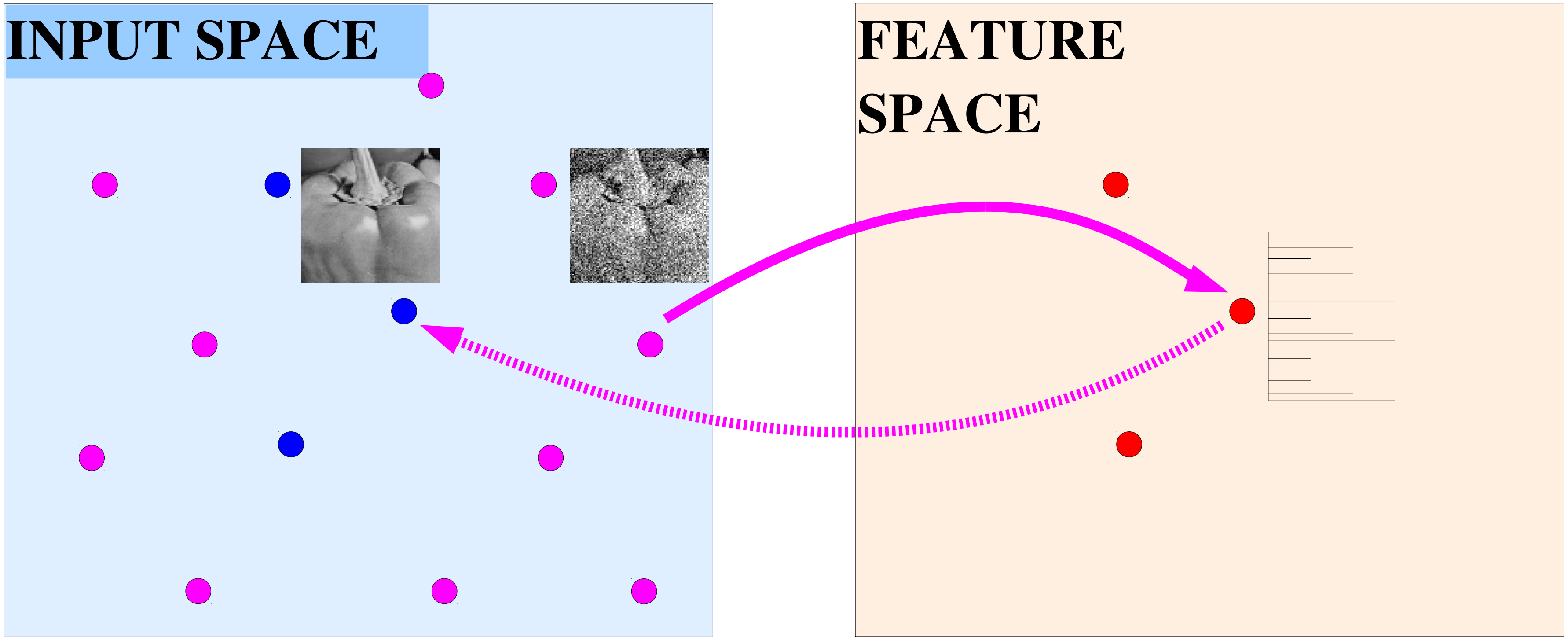
IDEA: reduce number of available codes.



Why Limit the Information Content of the Code?

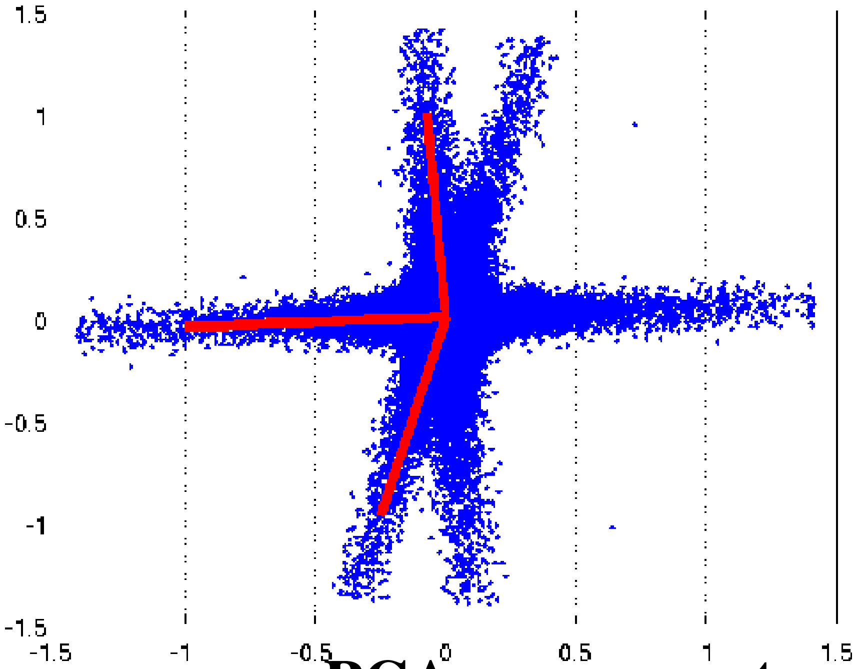
- ● Training sample
- ● Input vector which is NOT a training sample
- ● Feature vector

IDEA: reduce number of available codes.



Sparsity Penalty to Restrict the Code

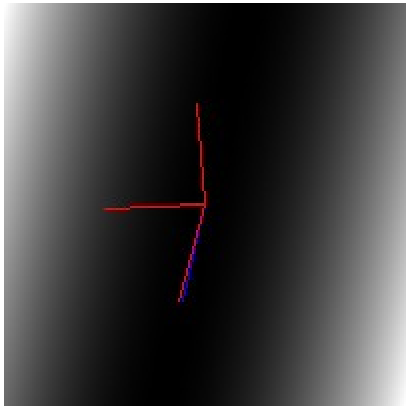
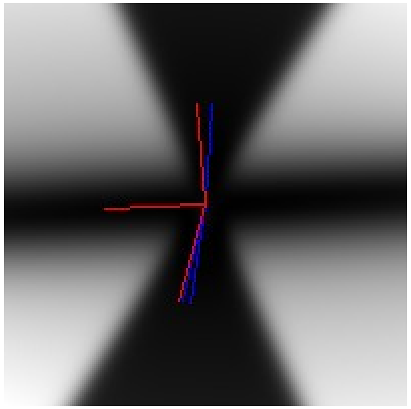
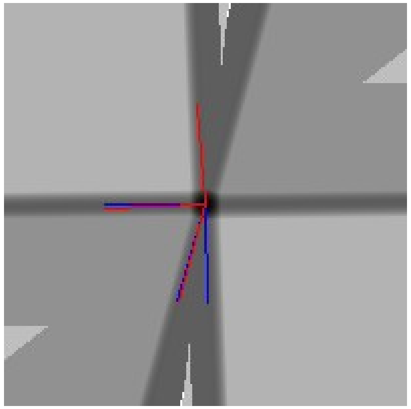
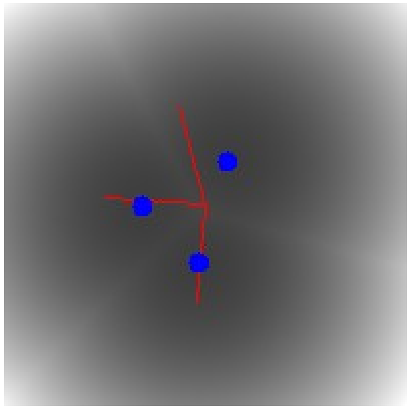
- **We are going to impose a sparsity penalty on the code to restrict its information content.**
- **We will allow the code to have higher dimension than the input**
- **Categories are more easily separable in high-dim sparse feature spaces**
  - This is a trick that SVM use: they have one dimension per sample
- **Sparse features are optimal when an active feature costs more than an inactive one (zero).**
  - e.g. neurons that spike consume more energy
  - The brain is about 2% active on average.

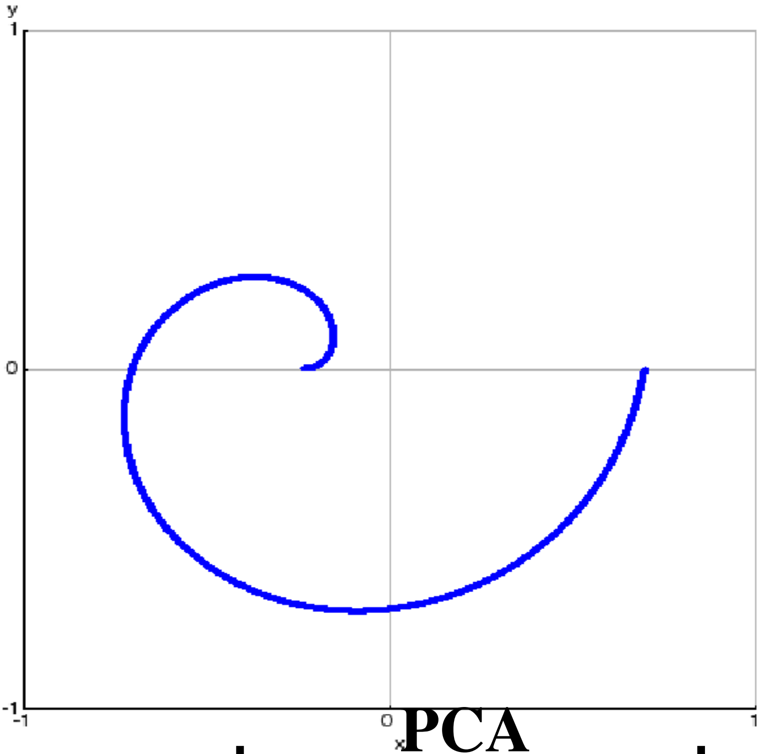


- 2 dimensional toy dataset
  - Mixture of 3 Cauchy distrib.
- Visualizing energy surface

(black = low, white = high)

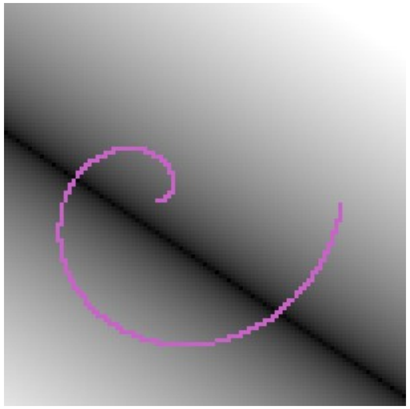
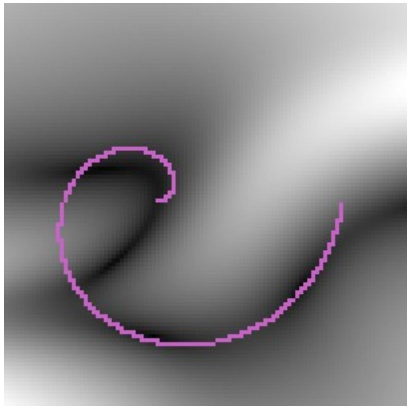
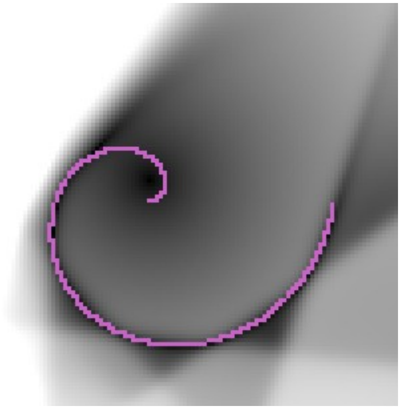
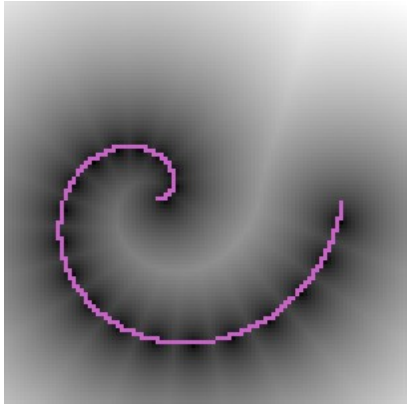
[Ranzato 's PhD thesis 2009]

	PCA **(1 code unit)**	**autoencoder** **(3 code units)**	**sparse coding** **(3 code units)**	**K-Means** **(3 code units)**
encoder	$W^T Y$	$\sigma(W_e Y)$	$-$	$-$
decoder	WZ	$W_d Z$	WZ	WZ
energy	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2+\lambda\|Z\|$	$\|\|Y-WZ\|\|^2$
loss	$F(Y)$	$F(Y)+\log\Gamma$	$F(Y)$	$F(Y)$
pull-up	dimens.	part. func.	sparsity	1-of-N code
				



- 2 dimensional toy dataset
  - spiral
- Visualizing energy surface

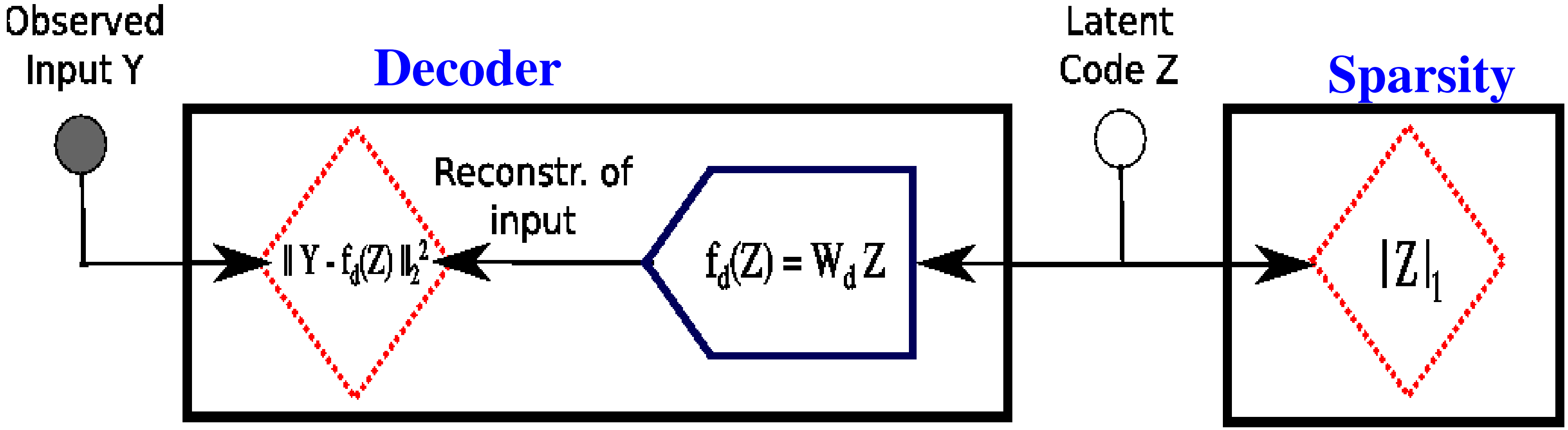
(black = low, white = high)

	PCA **(1 code unit)**	**autoencoder** **(1 code unit)**	**sparse coding** **(20 code units)**	**K-Means** **(20 code units)**
encoder	W^TY	$\sigma(W_eY)$	$\sigma(W_eZ)$	—
decoder	WZ	W_dZ	W_dZ	WZ
energy	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$
loss	$F(Y)$	$F(Y)$	$F(Y)$	$F(Y)$
pull-up	dimens.	dimens.	sparsity	1-of-N code
				

Sparse Decomposition with Linear Reconstruction

[Olshausen and Field 1997]

- **Energy(Input,Code) = || Input – Decoder(Code) ||^2 + Sparsity(Code)**
- **Energy(Input) = Min_over_Code[Energy(Input,Code)]**



- **Energy: minimize to infer Z**

$$E(Y^i, Z^i; W) = \|Y^i - W_d Z^i\|^2 + \lambda \sum_j |z_j^i|$$
$$F(Y^i; W) = min_z E(Y^i, z; W)$$

- **Loss: minimize to learn W (the columns of W are constrained to have norm 1)**

$$L(W) = \sum_i F(Y^i; W) = \sum_i (min_{Z^i} E(Y^i, Z^i; W))$$

Problem with Sparse Decomposition: It's slow

- **Inference: Optimal_Code = Arg_Min_over_Code[Energy(Input,Code)]**

$$E(Y^i, Z^i; W) = \|Y^i - W_d Z^i\|^2 + \lambda \sum_j |z^i_j|$$

$$F(Y^i; W) = min_z E(Y^i, z; W)$$

$$Z^i = argmin_z E(Y^i, z; W)$$

- **For each new Y, an optimization algorithm must be run to find the corresponding optimal Z**
- **This would be very slow for large scale vision tasks**
- **Also, the optimal Z are very unstable:**
  - A small change in Y can cause a large change in the optimal Z

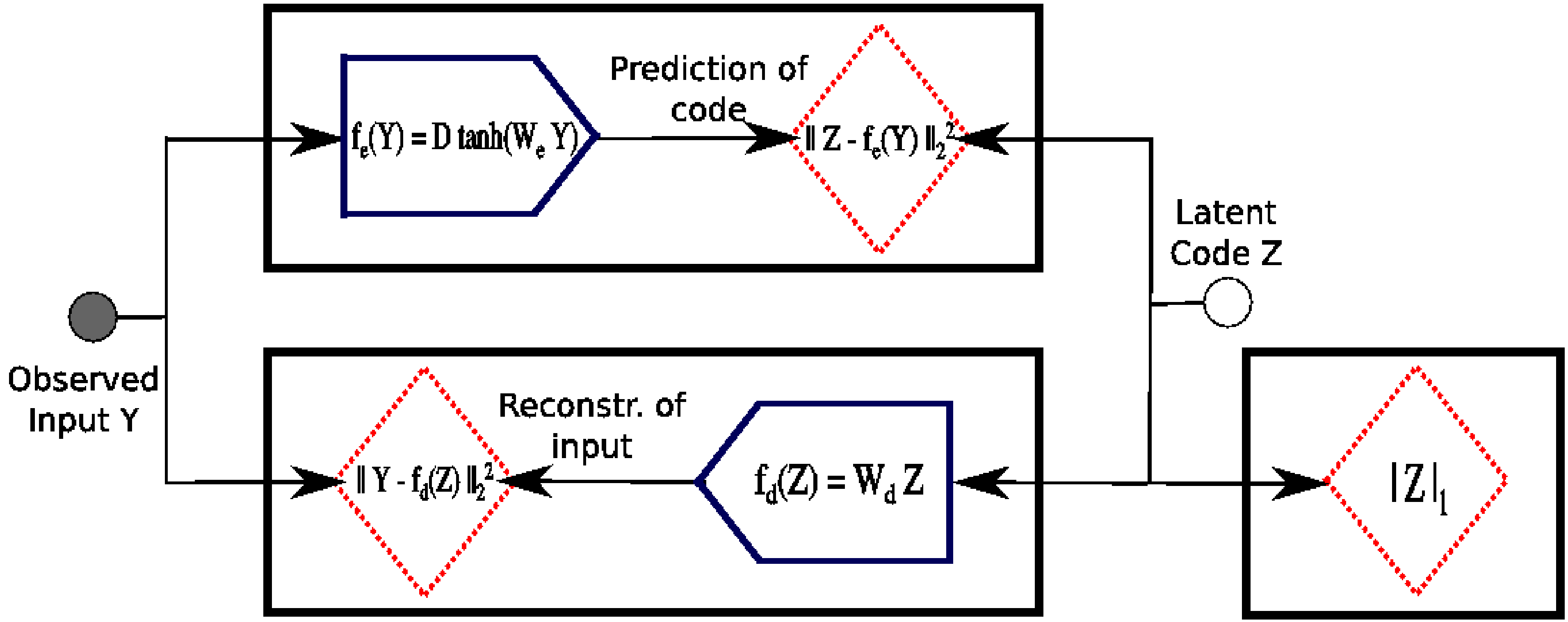
Solution: Predictive Sparse Decomposition (PSD)

[Kavukcuoglu, Ranzato, LeCun, 2009]

- **Prediction the optimal code with a trained encoder**
- **Energy = reconstruction_error + code_prediction_error + code_sparsity**

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

$$f_e(Y^i) = D \tanh(W_e Y)$$

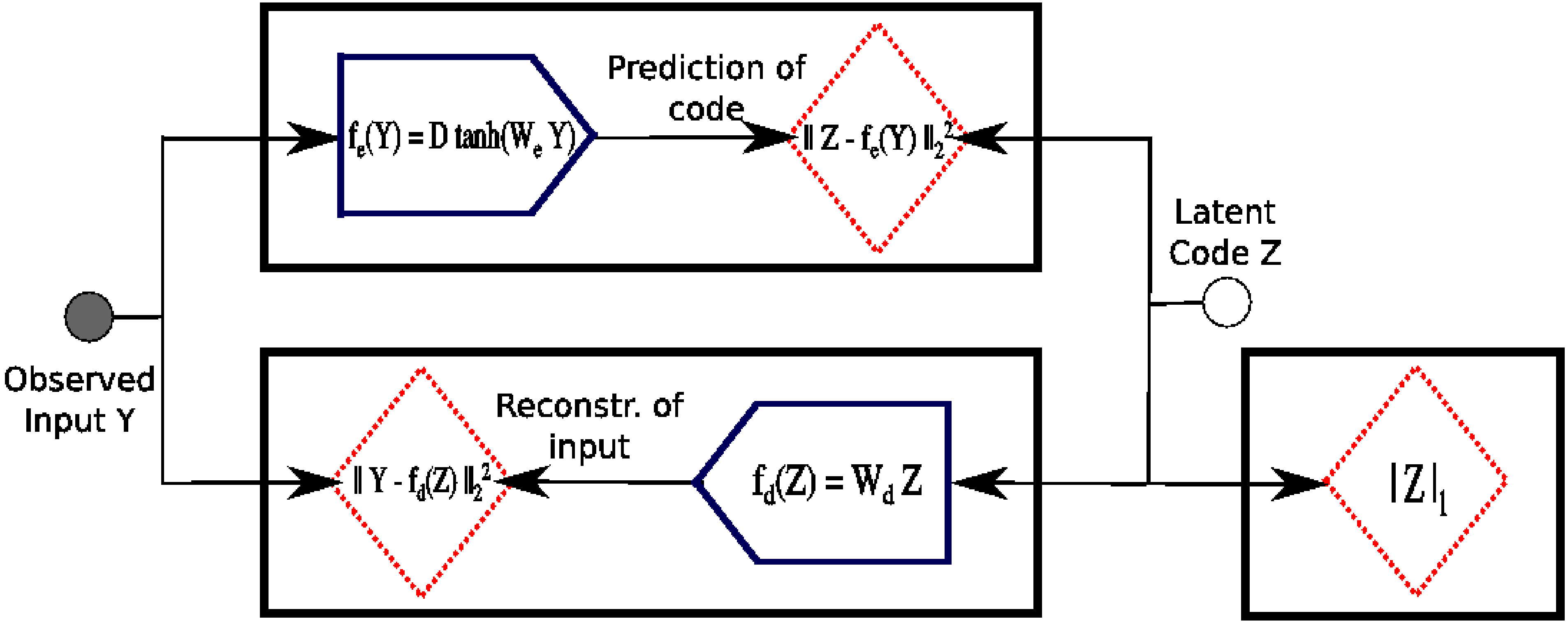


PSD: Inference

- **Inference by gradient descent starting from the encoder output**

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

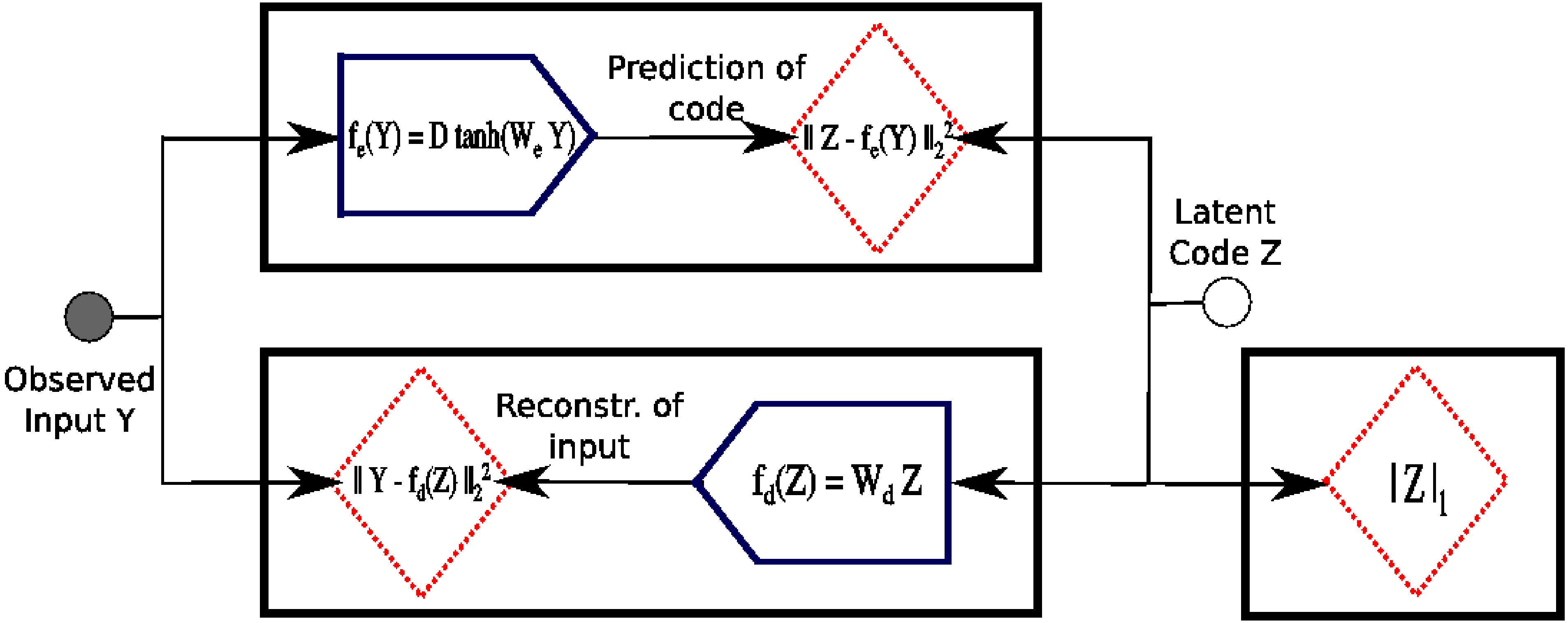
$$Z^i = argmin_z E(Y^i, z; W)$$



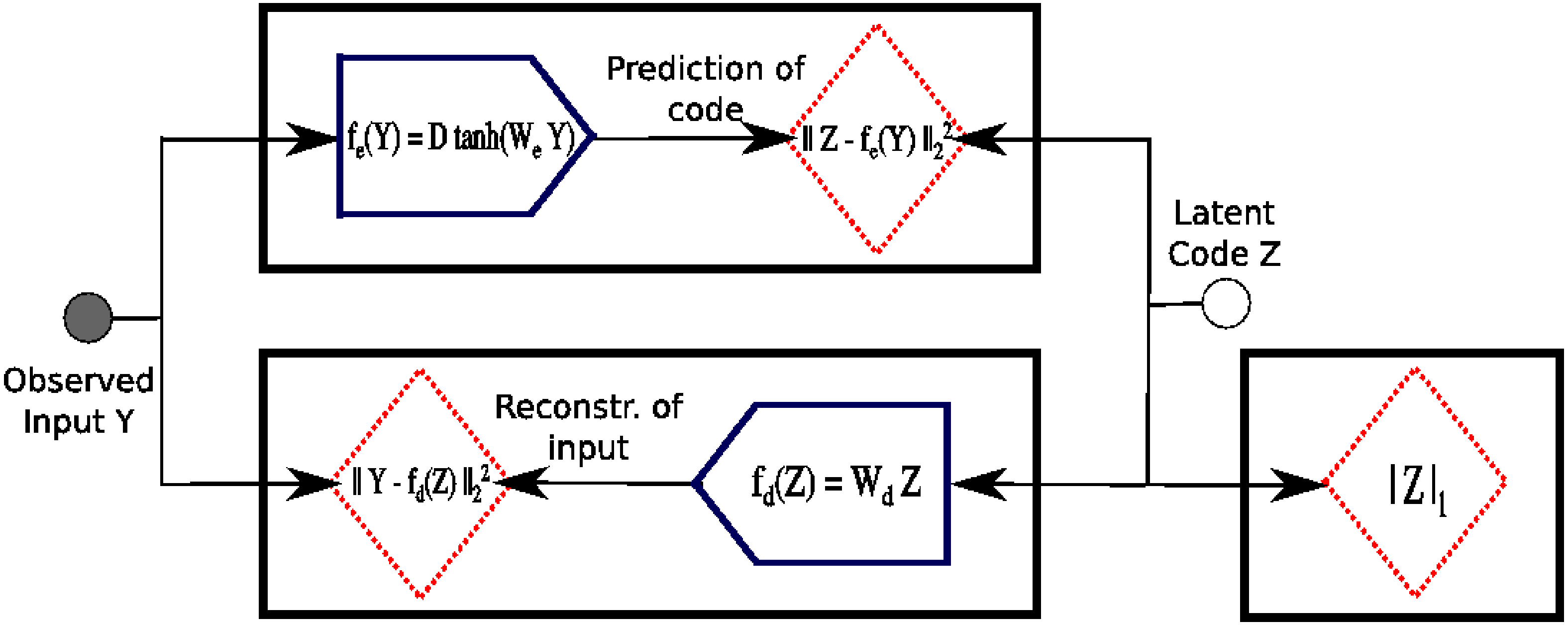
PSD: Learning [Kavukcuoglu et al. 2009]

- **Learning by minimizing the average energy of the training data with respect to Wd and We.**
- **Loss function:** $L(W)=\sum_i F(Y^i;W)$

$$F(Y^i;W)=min_z E(Y^i,z;W)$$



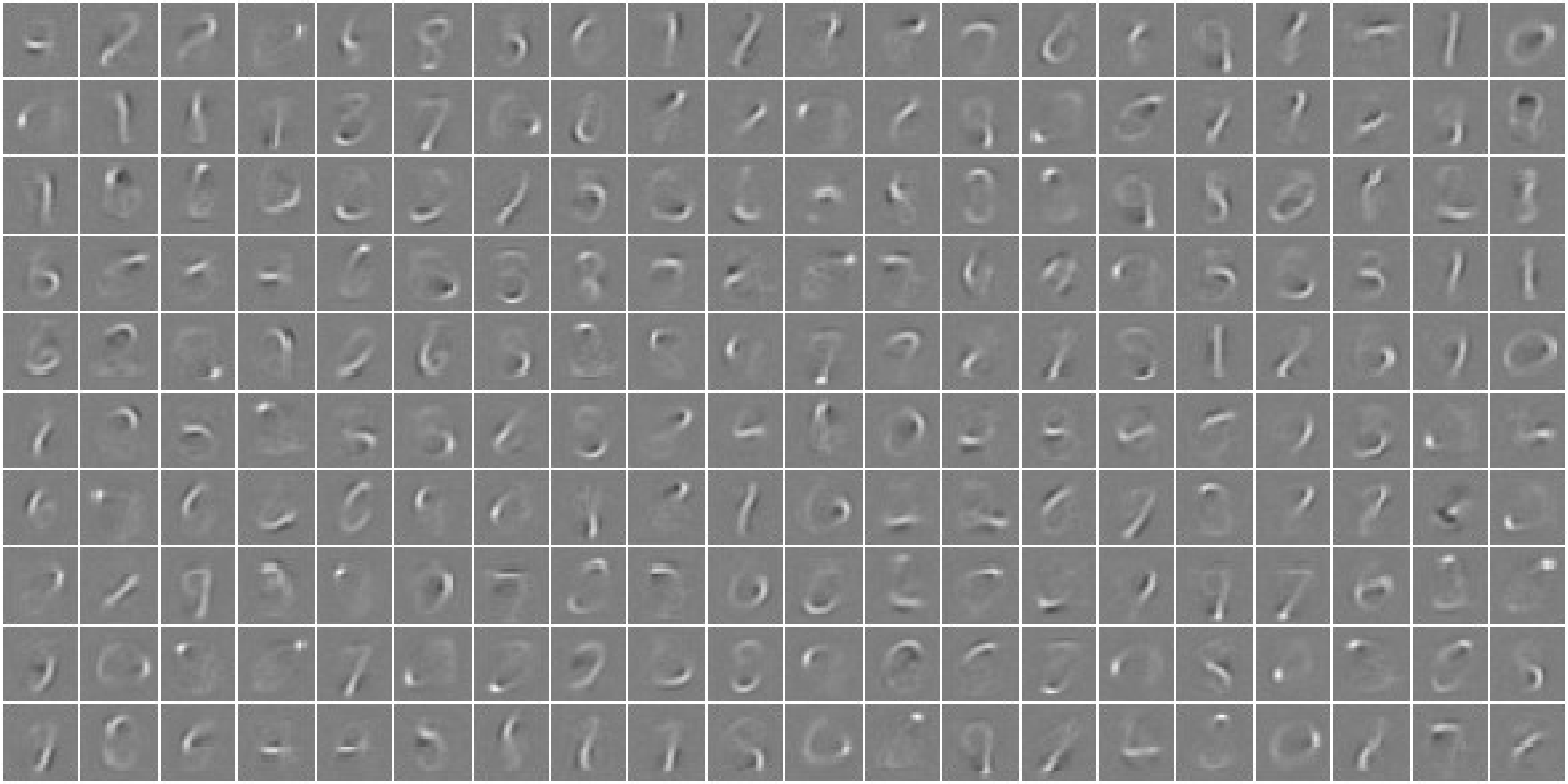
PSD: Learning Algorithm



- 1. Initialize Z = Encoder(Y)
- 2. Find Z that minimizes the energy function
- 3. Update the Decoder basis functions to reduce reconstruction error
- 4. Update Encoder parameters to reduce prediction error
- Repeat with next training sample

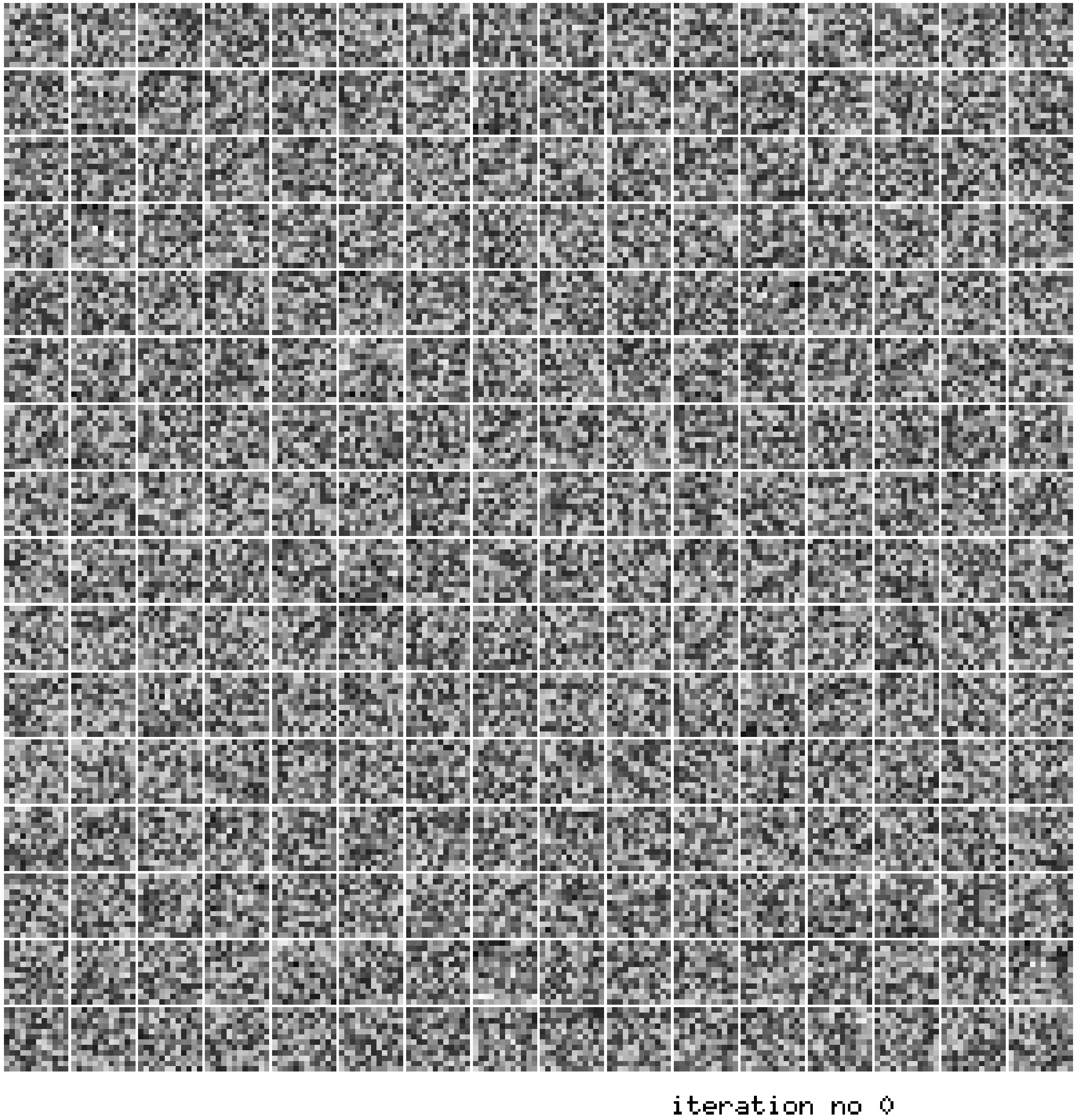
Decoder Basis Functions on MNIST

- **PSD trained on handwritten digits: decoder filters are "parts" (strokes).**
  - Any digit can be reconstructed as a linear combination of a small number of these "parts".



PSD Training on Natural Image Patches

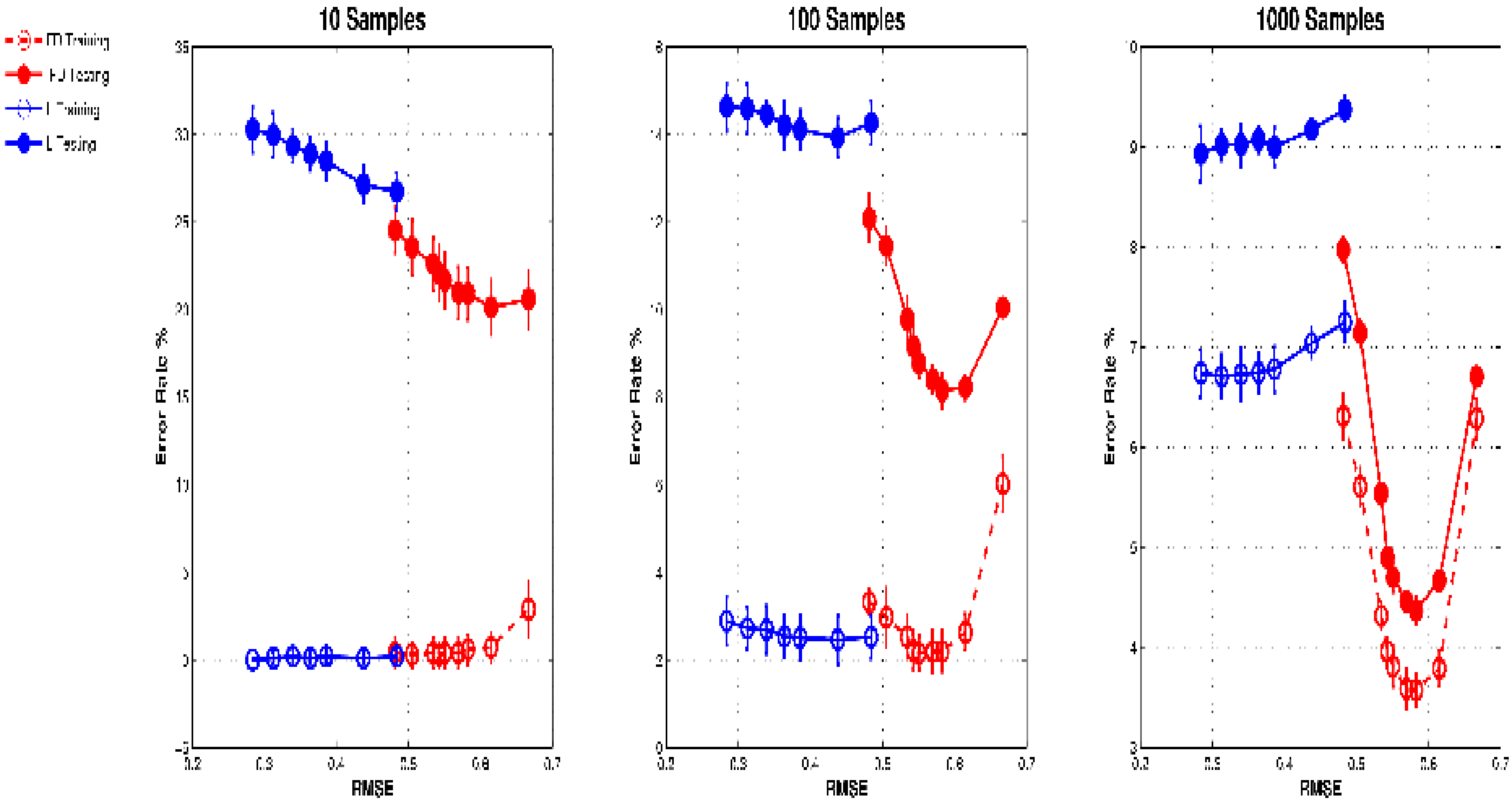
- Basis functions are like Gabor filters (like receptive fields in V1 neurons)



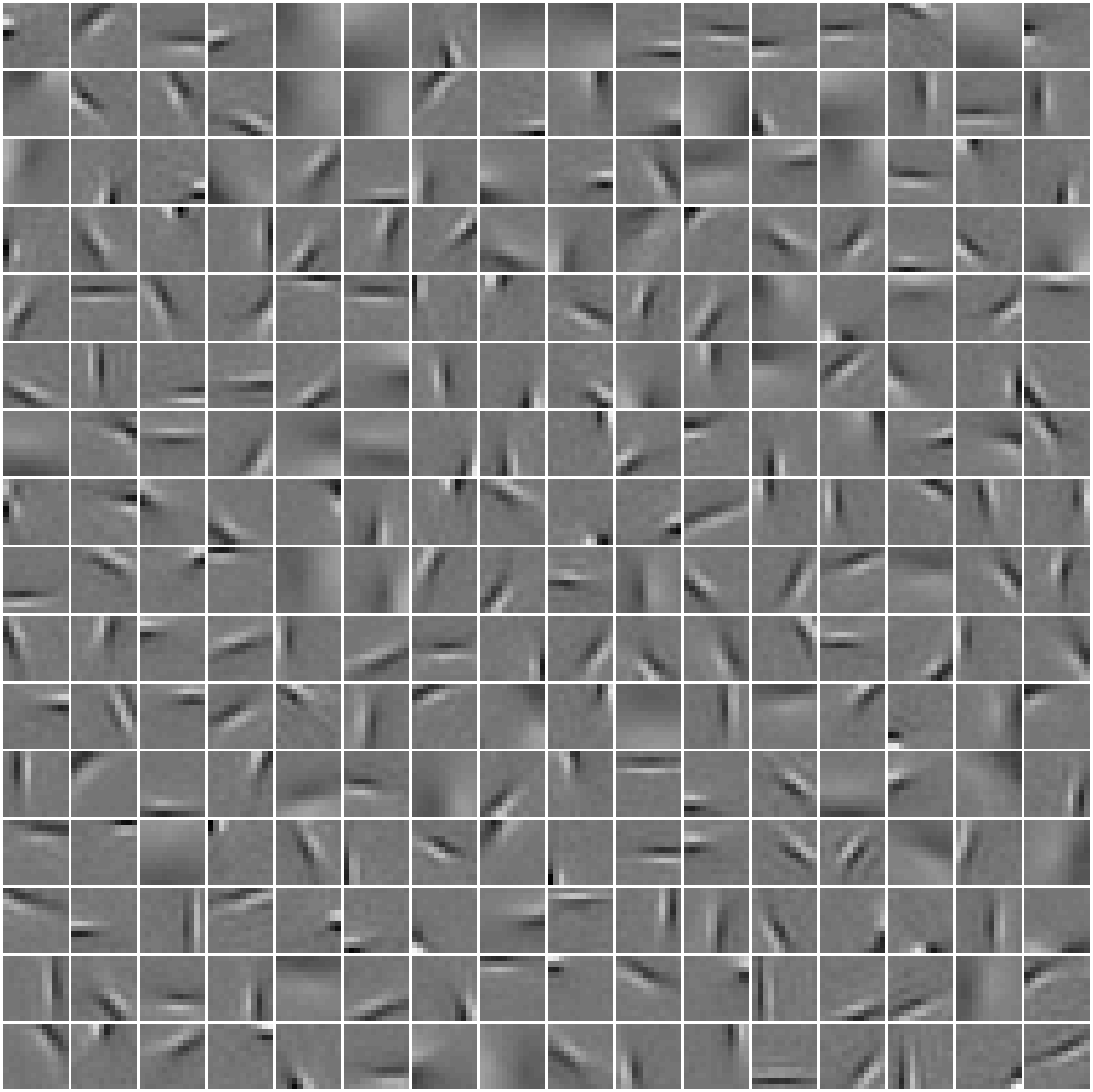
- 256 filters of size 12x12
- Trained on natural image patches from the Berkeley dataset
- Encoder is linear-tanh-diagonal

Classification Error Rate on MNIST

- **Supervised Linear Classifier trained on 200 trained sparse features**
  - Red: linear-tanh-diagonal encoder; Blue: linear encoder

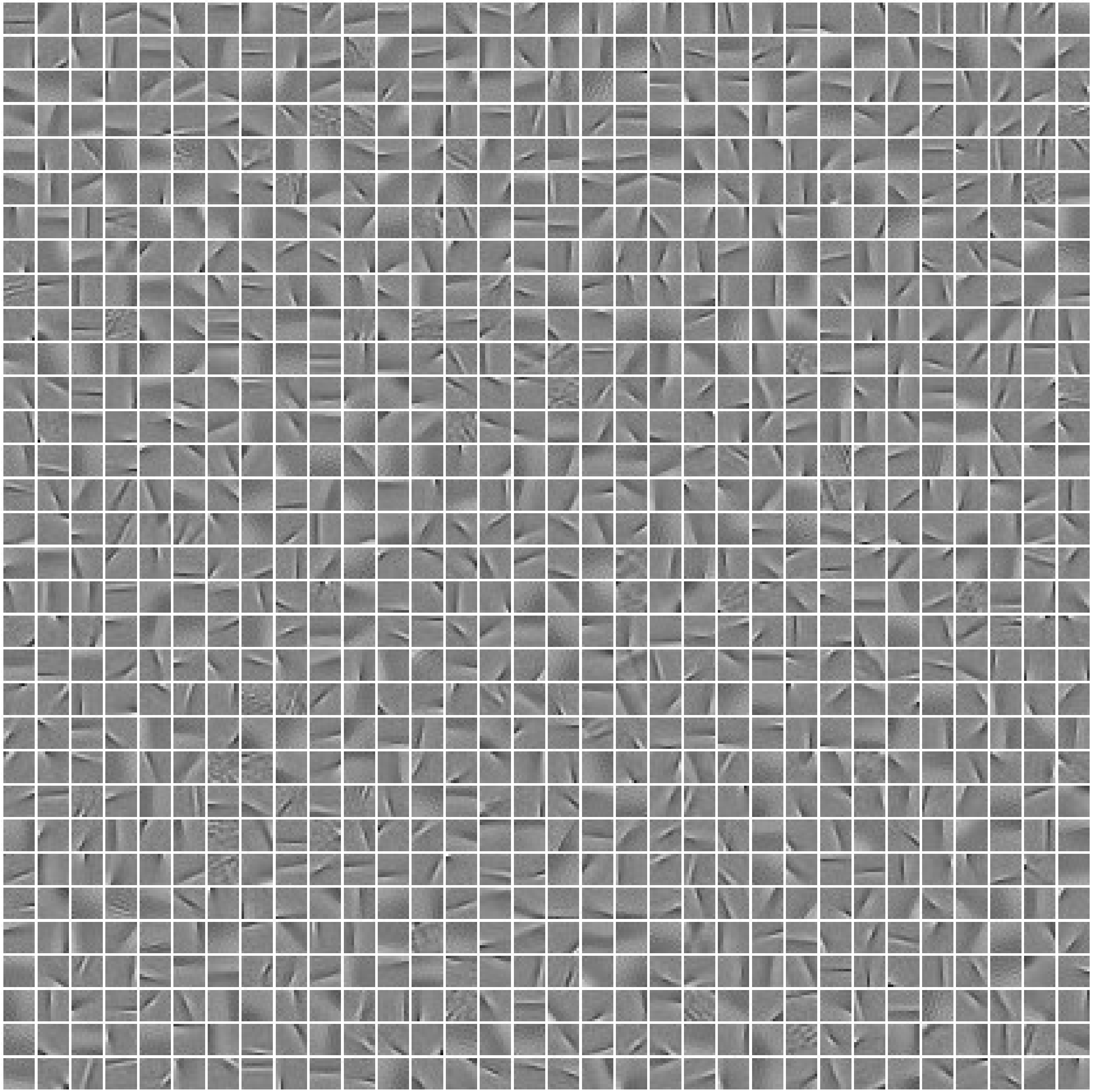


Learned Features on natural patches: V1-like receptive fields

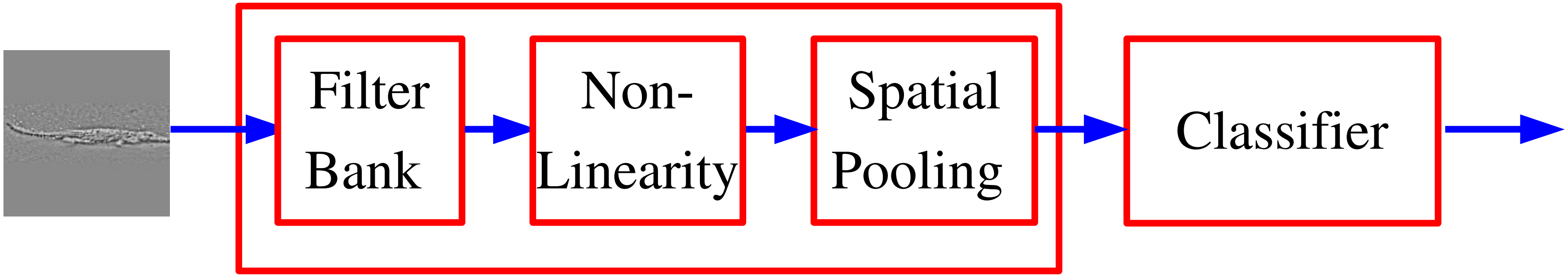


Learned Features: V1-like receptive fields

- **12x12 filters**
- **1024 filters**

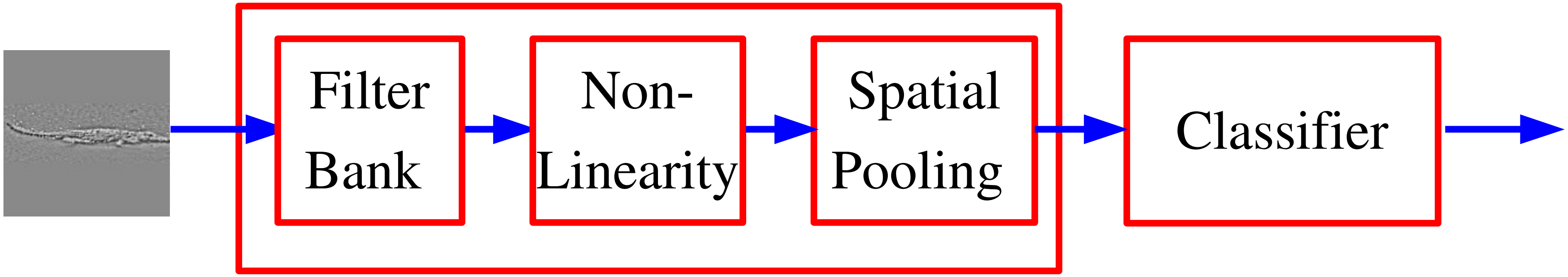


Using PSD to learn the features of an object recognition system



- Learning the filters of a ConvNet-like architecture with PSD
- 1. Train filters on images patches with PSD
- 2. Plug the filters into a ConvNet architecture
- 3. Train a supervised classifier on top

“Modern” Object Recognition Architecture in Computer Vision



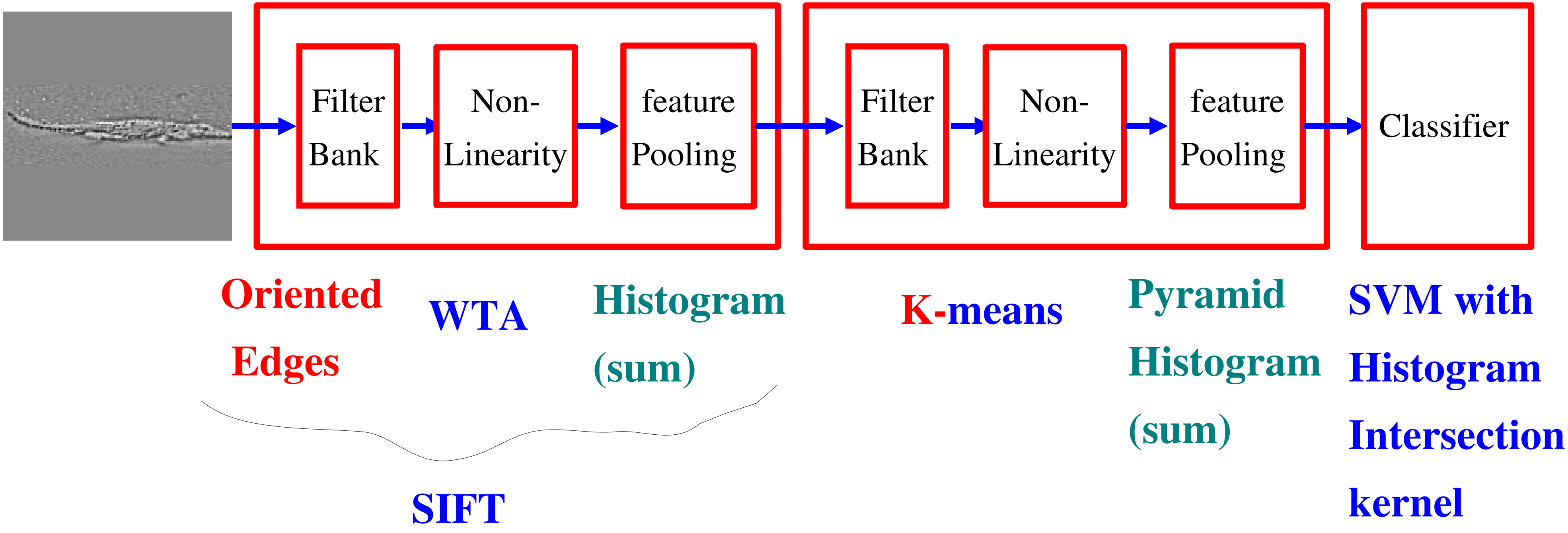
Oriented Edges
Gabor Wavelets
Other Filters...

Sigmoid
Rectification
Vector Quant.
Contrast Norm.

Averaging
Max pooling
VQ+Histogram
Geometric Blurr

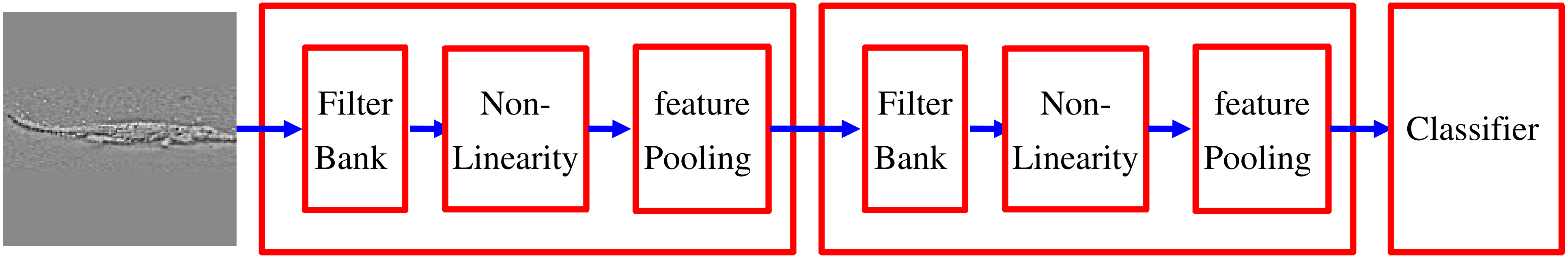
- **Example:**
  - Edges + Rectification + Histograms + SVM [Dalal & Triggs 2005]
  - SIFT + classification
- **Fixed Features + “shallow” classifier**

“State of the Art” architecture for object recognition



- **Example:**
  - SIFT features with Spatial Pyramid Match Kernel SVM [Lazebnik et al. 2006]
- **Fixed Features + unsupervised features + “shallow” classifier**

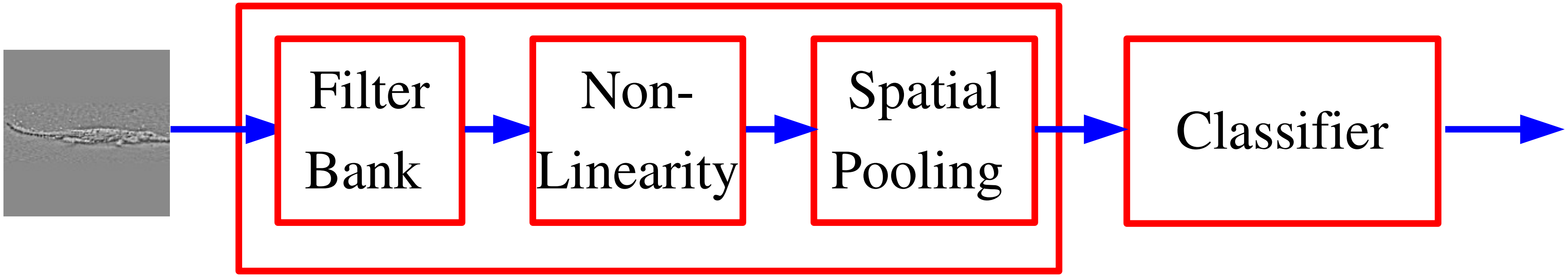
Can't we get the same results with (deep) learning?

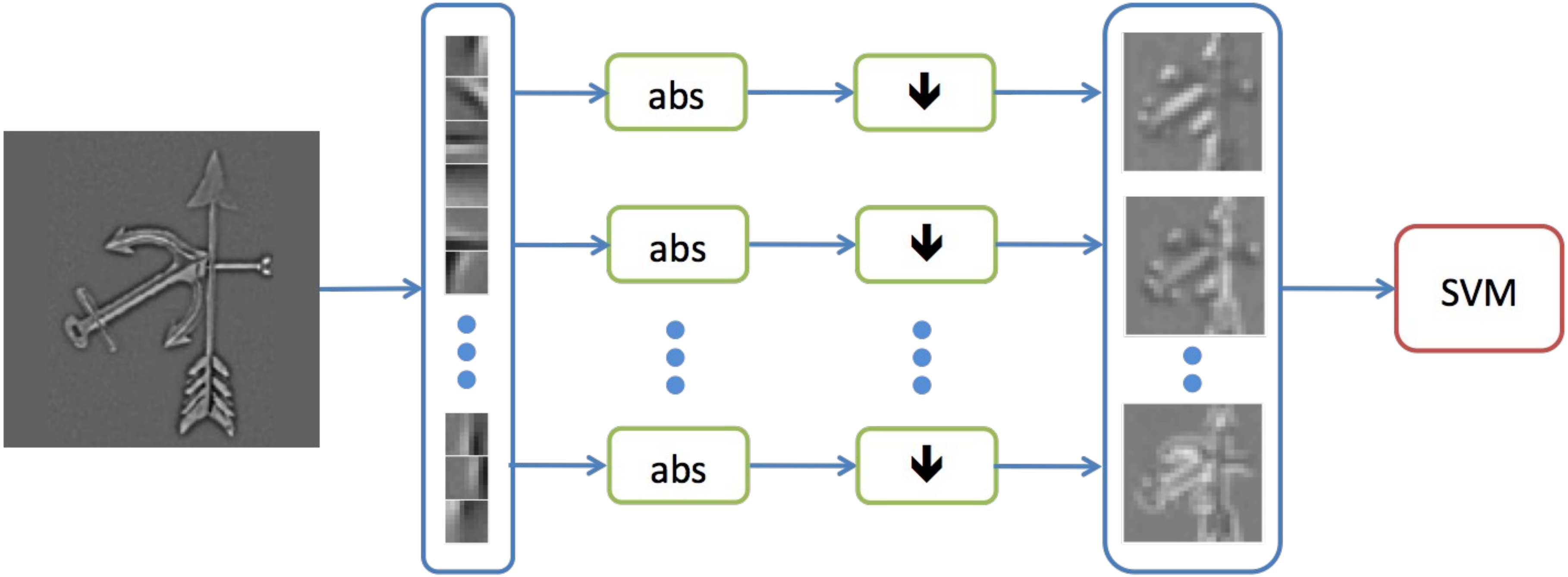


- **Stacking multiple stages of feature extraction/pooling.**
- **Creates a hierarchy of features**
- **ConvNets and SIFT+PMK-SVM architectures are conceptually similar**
- **Can deep learning make a ConvNet match the performance of SIFT+PNK-SVM?**

How well do PSD features work on Caltech-101?

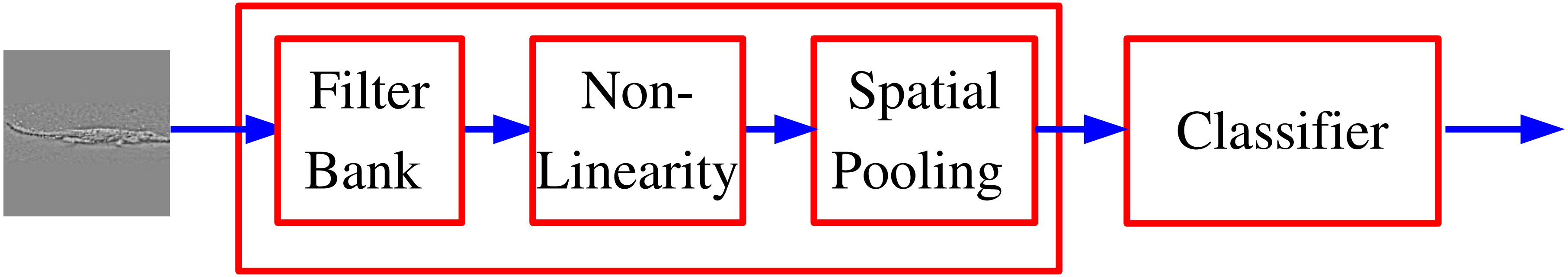
- **Recognition Architecture**





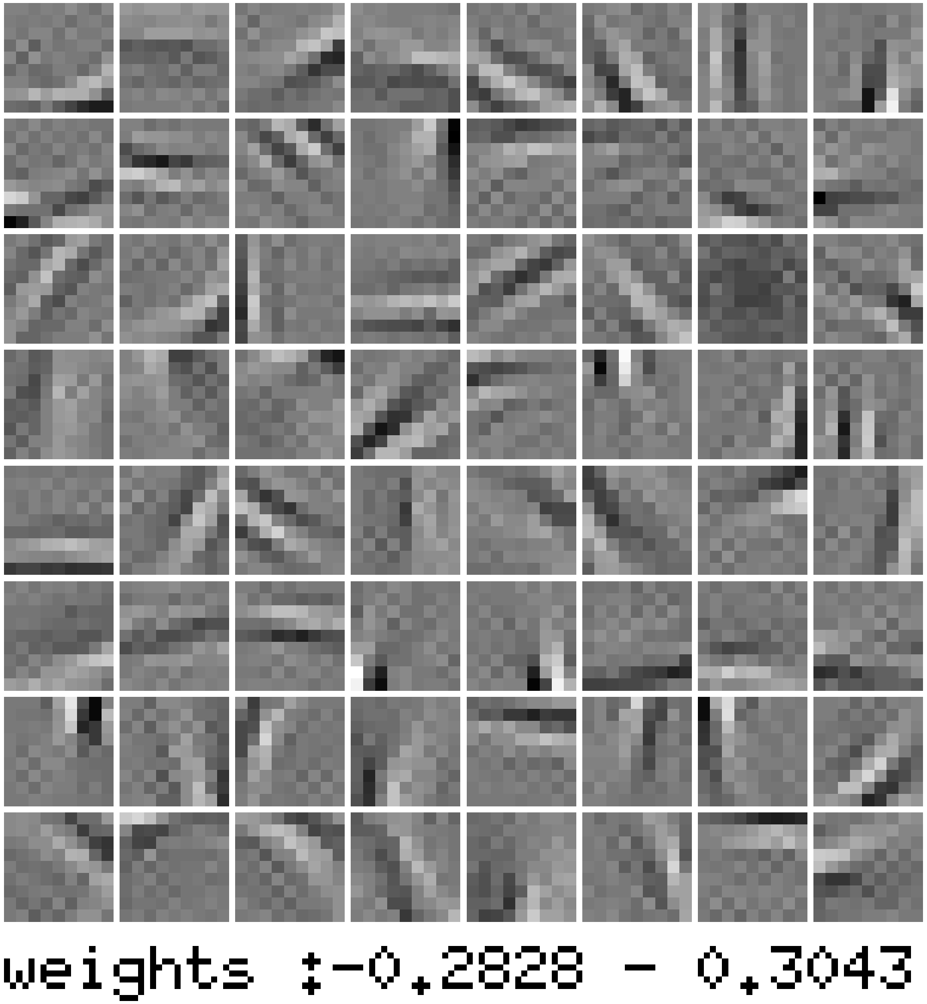
Procedure for a single-stage system

- **1. Pre-process images**
  - remove mean, high-pass filter, normalize contrast
- **2. Train encoder-decoder on 9x9 image patches**
- **3. use the filters in a recognition architecture**
  - Apply the filters to the whole image
  - Apply the tanh and D scaling
  - Add more non-linearities (rectification, normalization)
  - Add a spatial pooling layer
- **4. Train a supervised classifier on top**
  - Multinomial Logistic Regression or Pyramid Match Kernel SVM



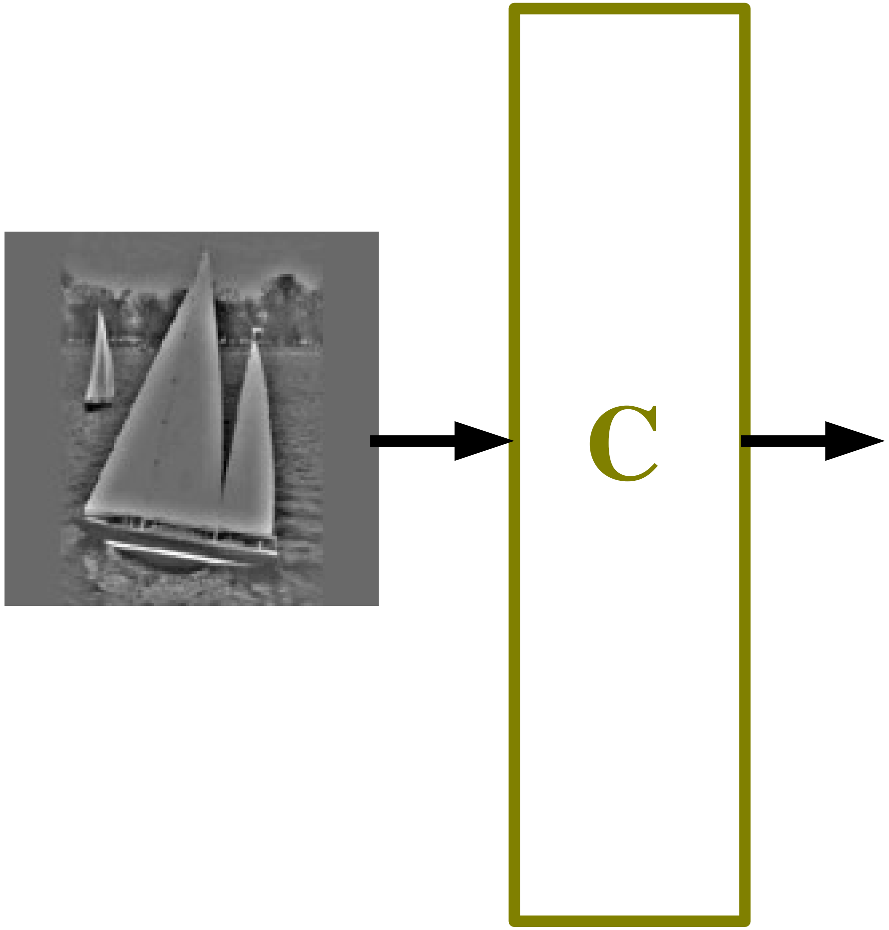
Using PSD Features for Recognition

- **64 filters on 9x9 patches trained with PSD**
  - with Linear-Sigmoid-Diagonal Encoder



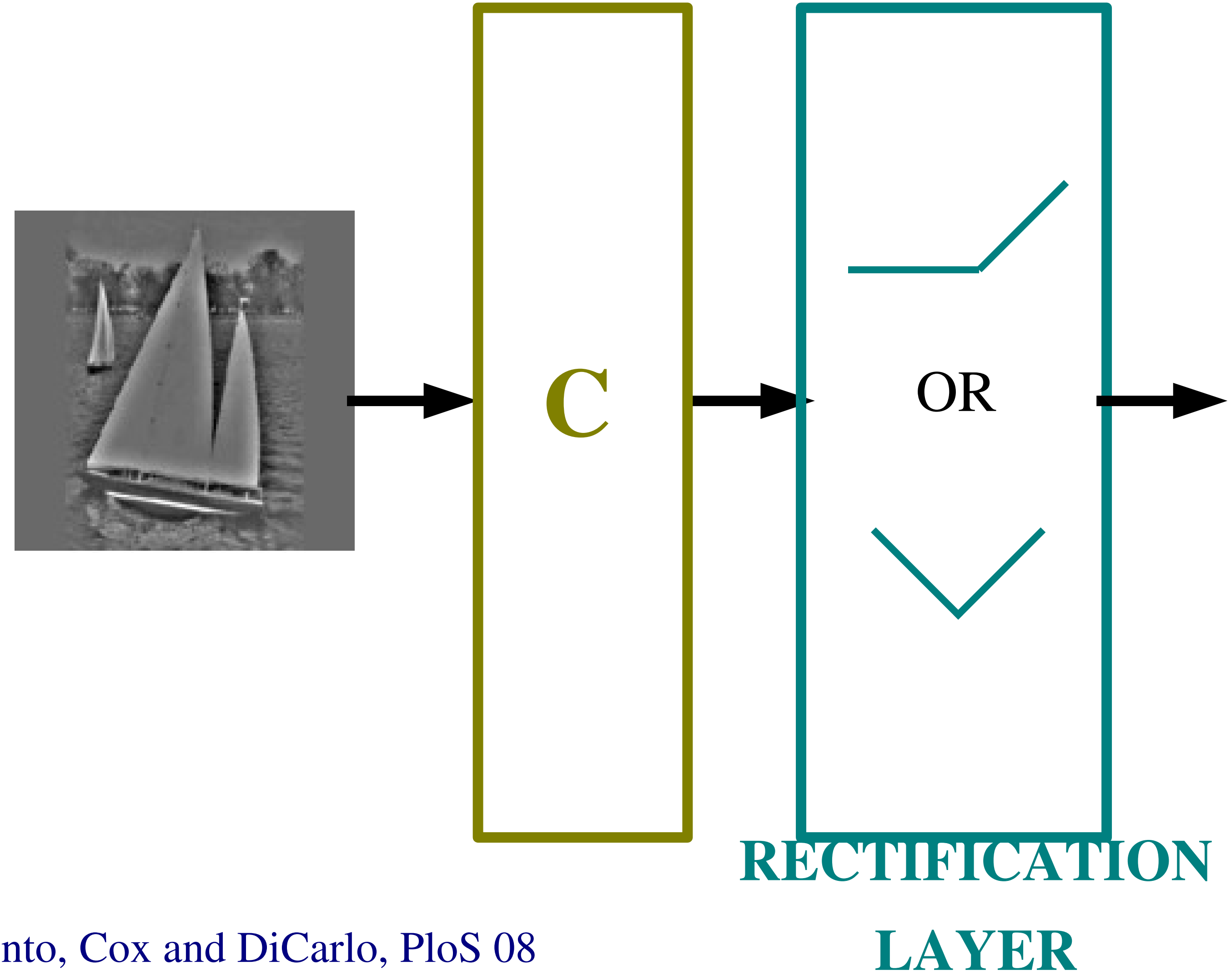
Feature Extraction

- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?



Feature Extraction

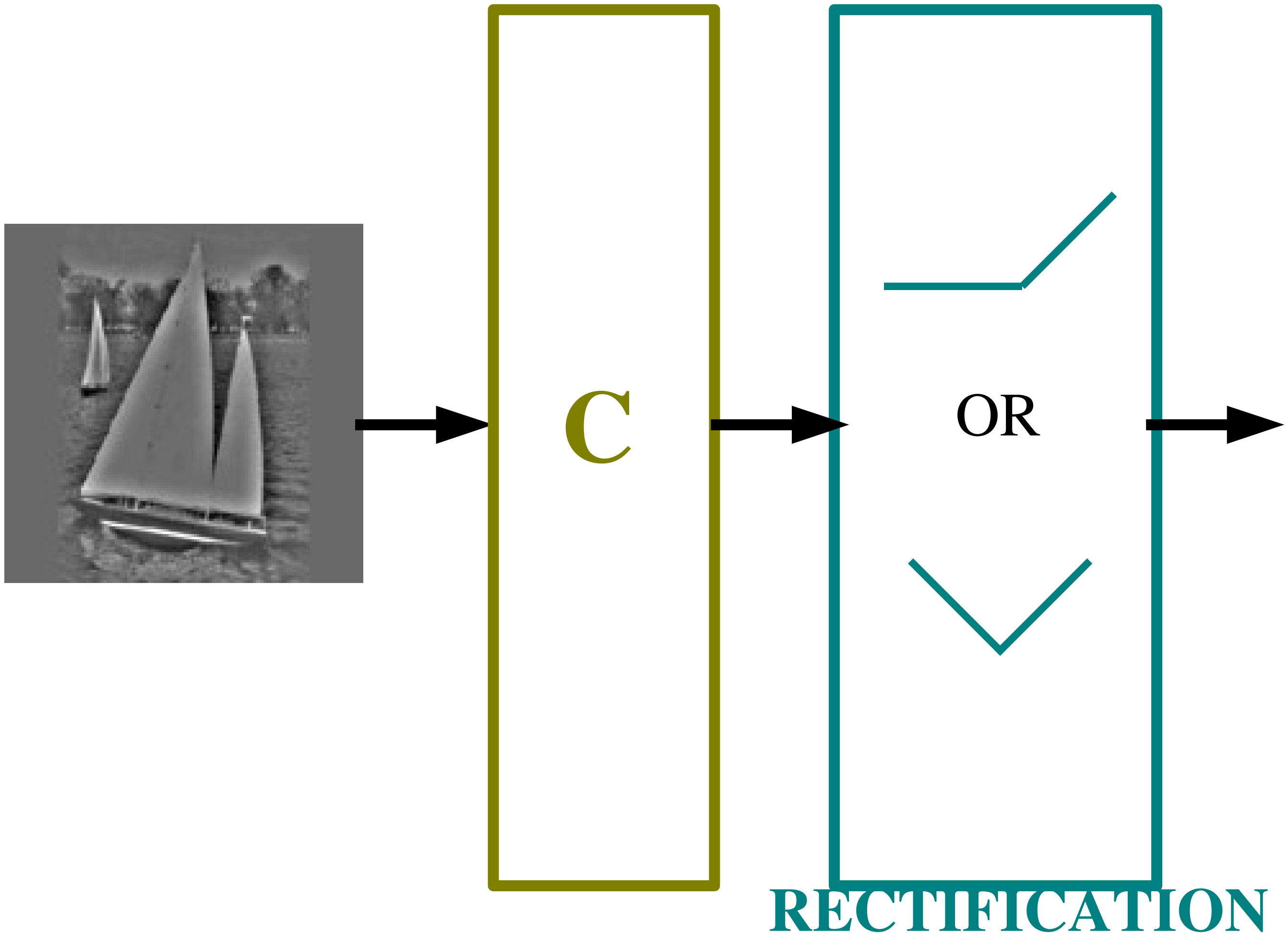
- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?



Pinto, Cox and DiCarlo, PloS 08

Feature Extraction

- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?



C

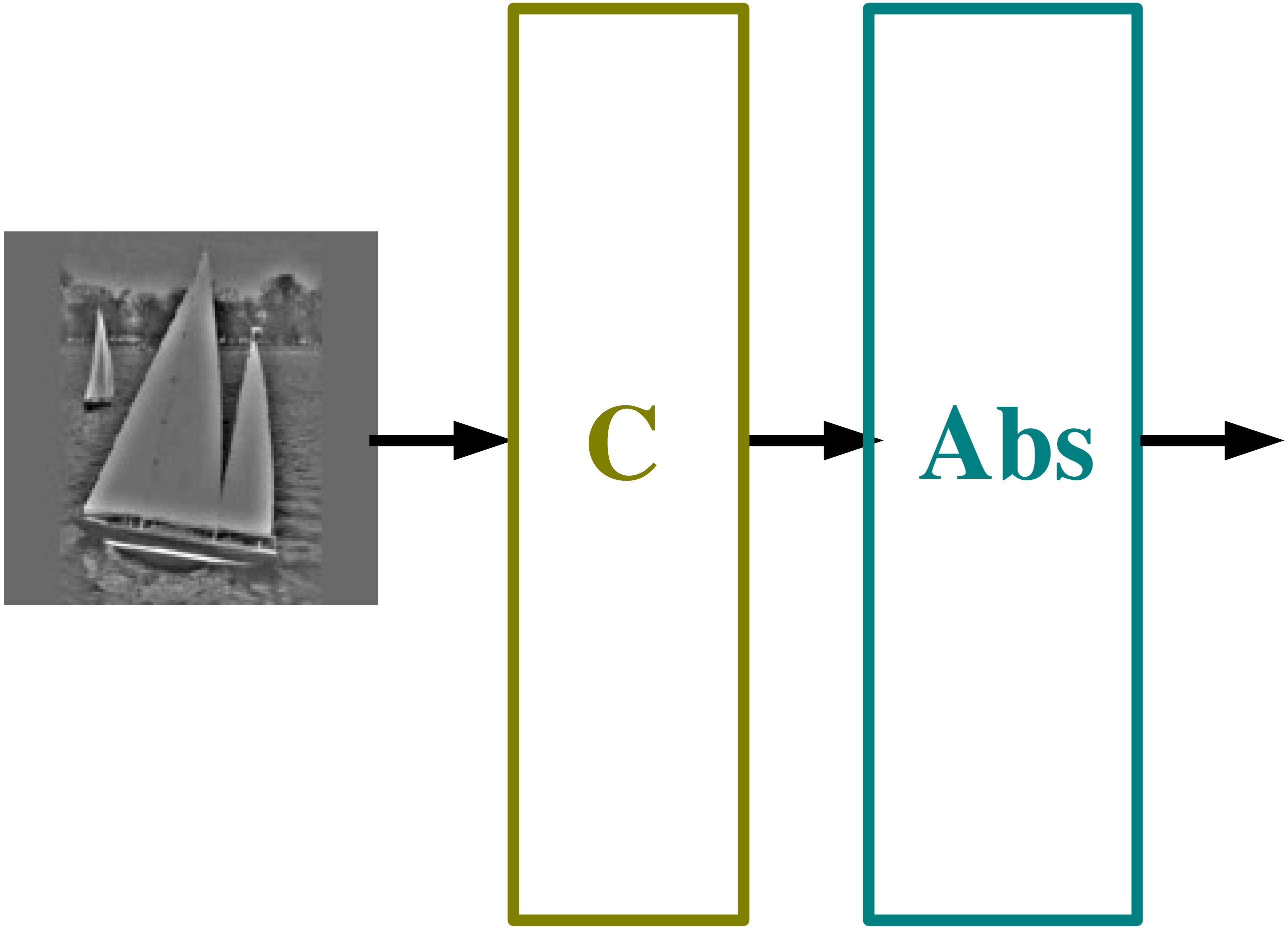
OR

RECTIFICATION LAYER

Pinto, Cox and DiCarlo, PloS 08

Feature Extraction

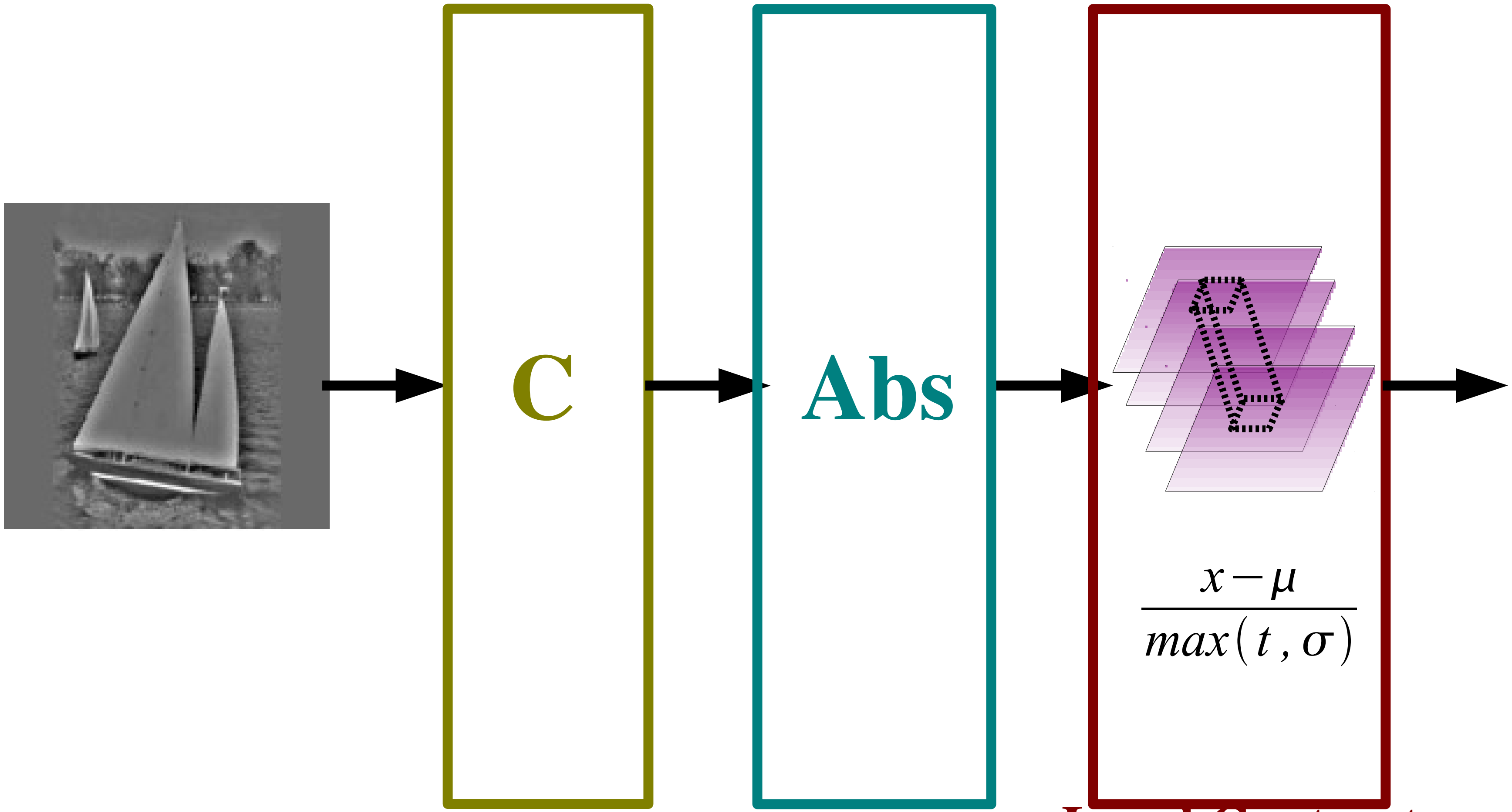
- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?



Pinto, Cox and DiCarlo, PloS 08

Feature Extraction

- C   Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?



C

Abs

$$\frac{x-\mu}{max(t,\sigma)}$$

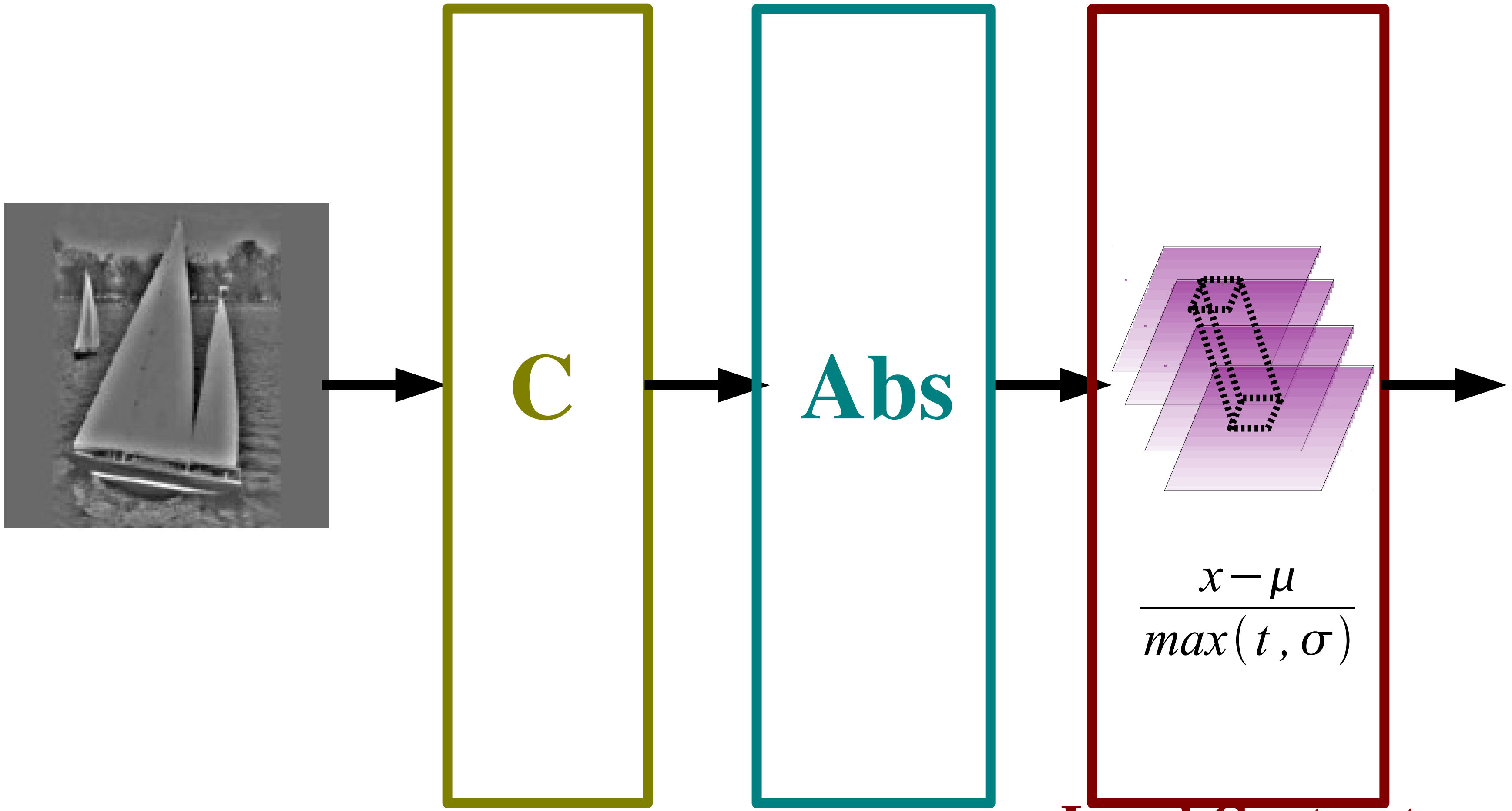
Local Contrast

Normalization Layer

Pinto, Cox and DiCarlo, PloS 08

Feature Extraction

- C   Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N   Normalization layer: needed?



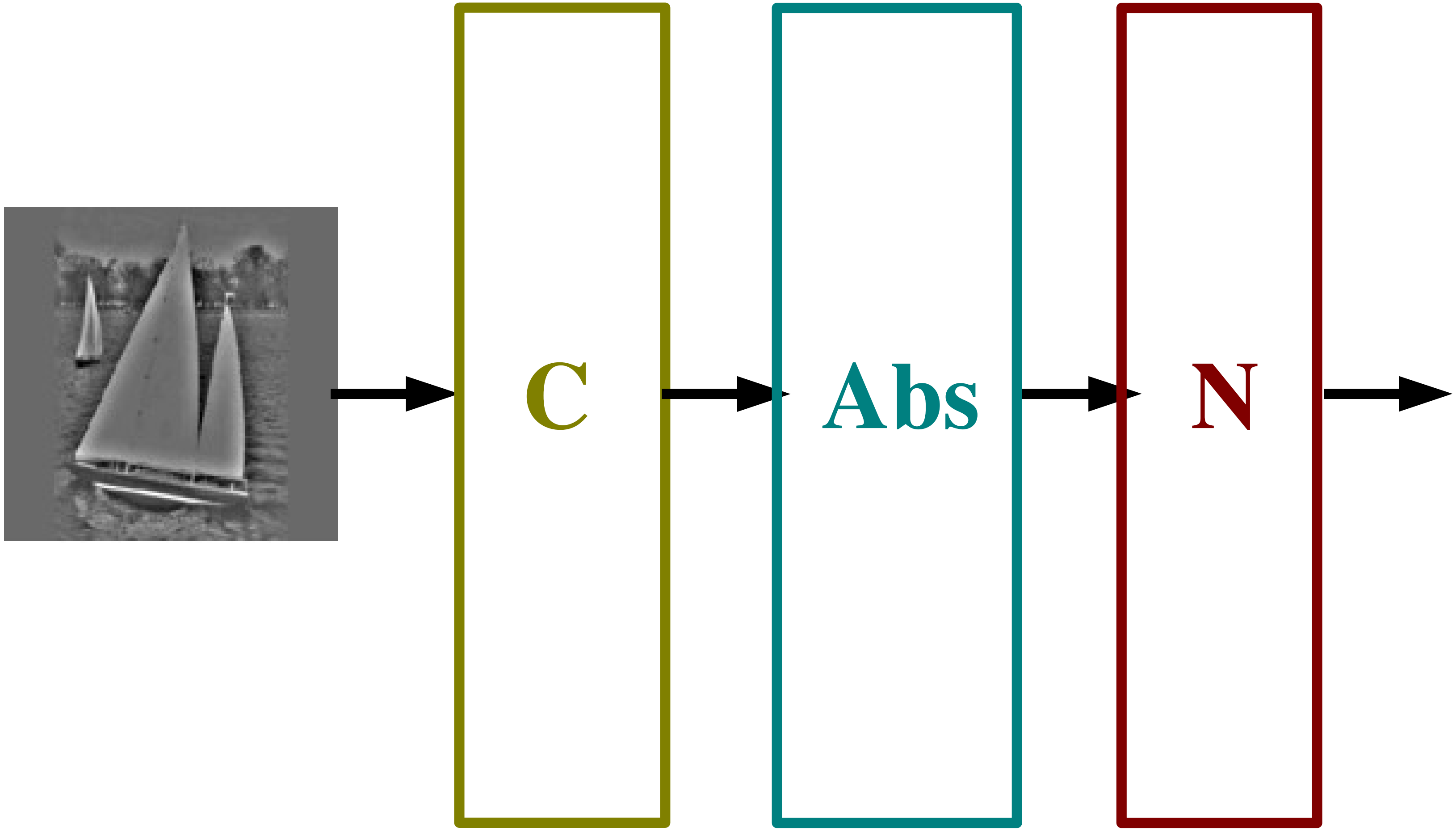
$$\frac{x-\mu}{max(t,\sigma)}$$

Local Contrast Normalization Layer

Pinto, Cox and DiCarlo, PloS 08

Feature Extraction

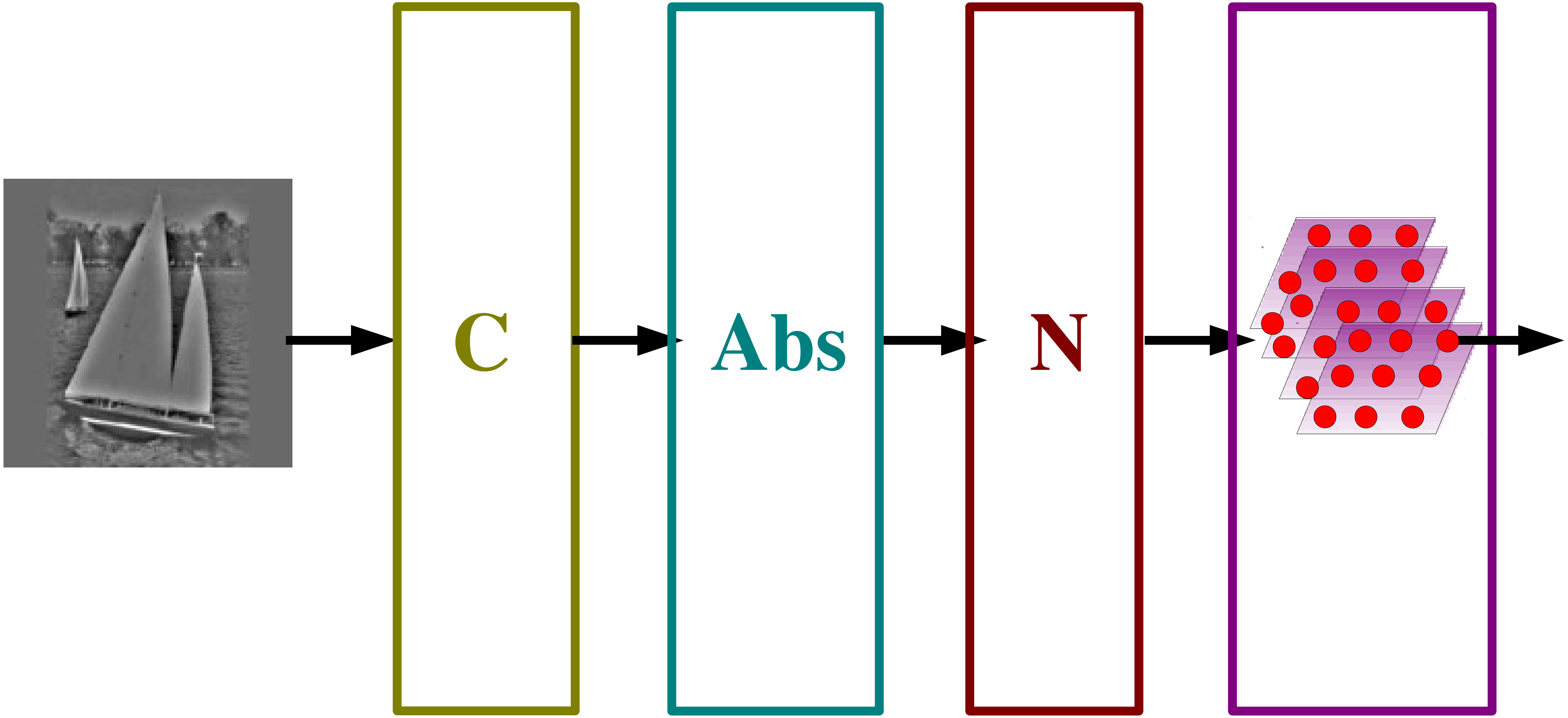
- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N Normalization layer: needed?



Pinto, Cox and DiCarlo, PloS 08

Feature Extraction

- C   Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N   Normalization layer: needed?



C

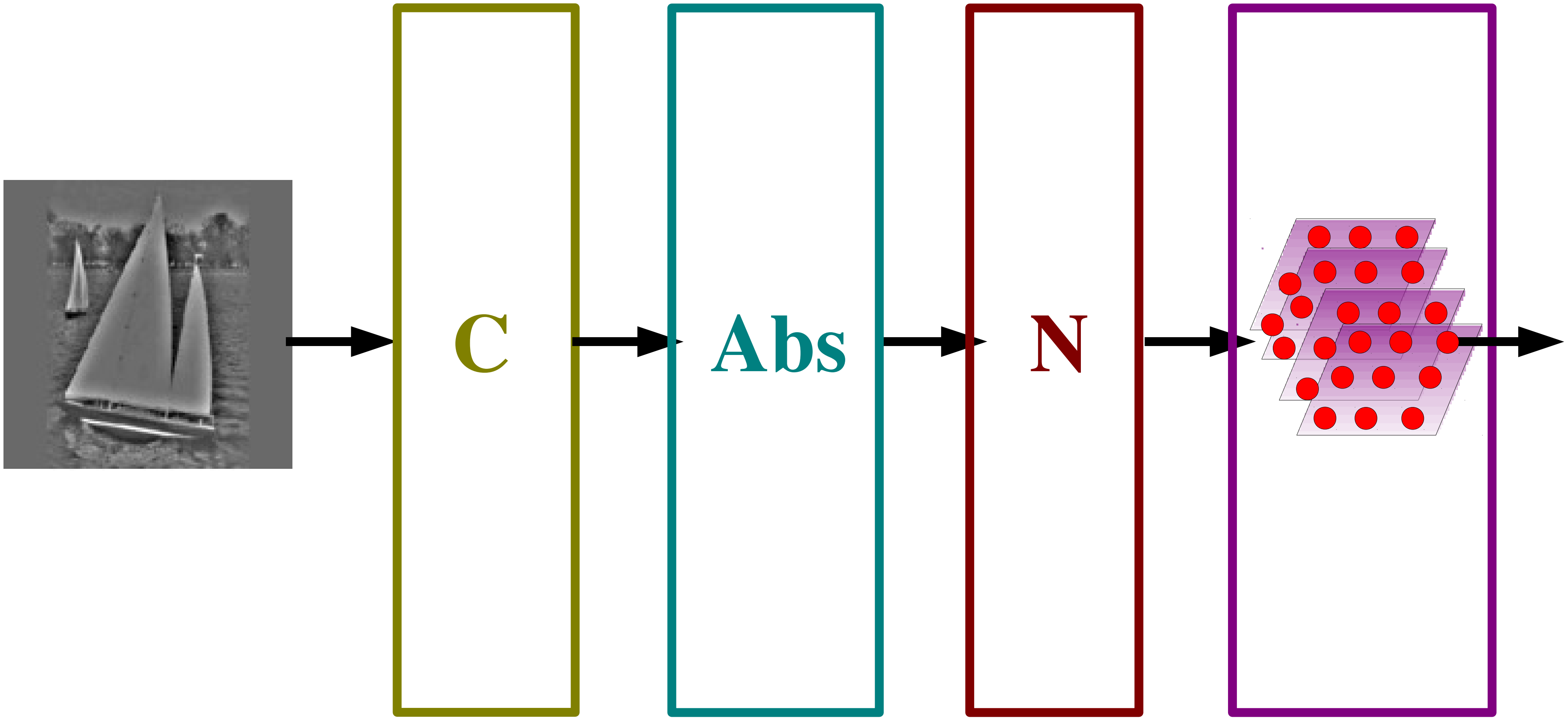
Abs

N

Pooling Down-Sampling Layer

Feature Extraction

- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N Normalization layer: needed?
- P Pooling down-sampling layer: average or max?



C

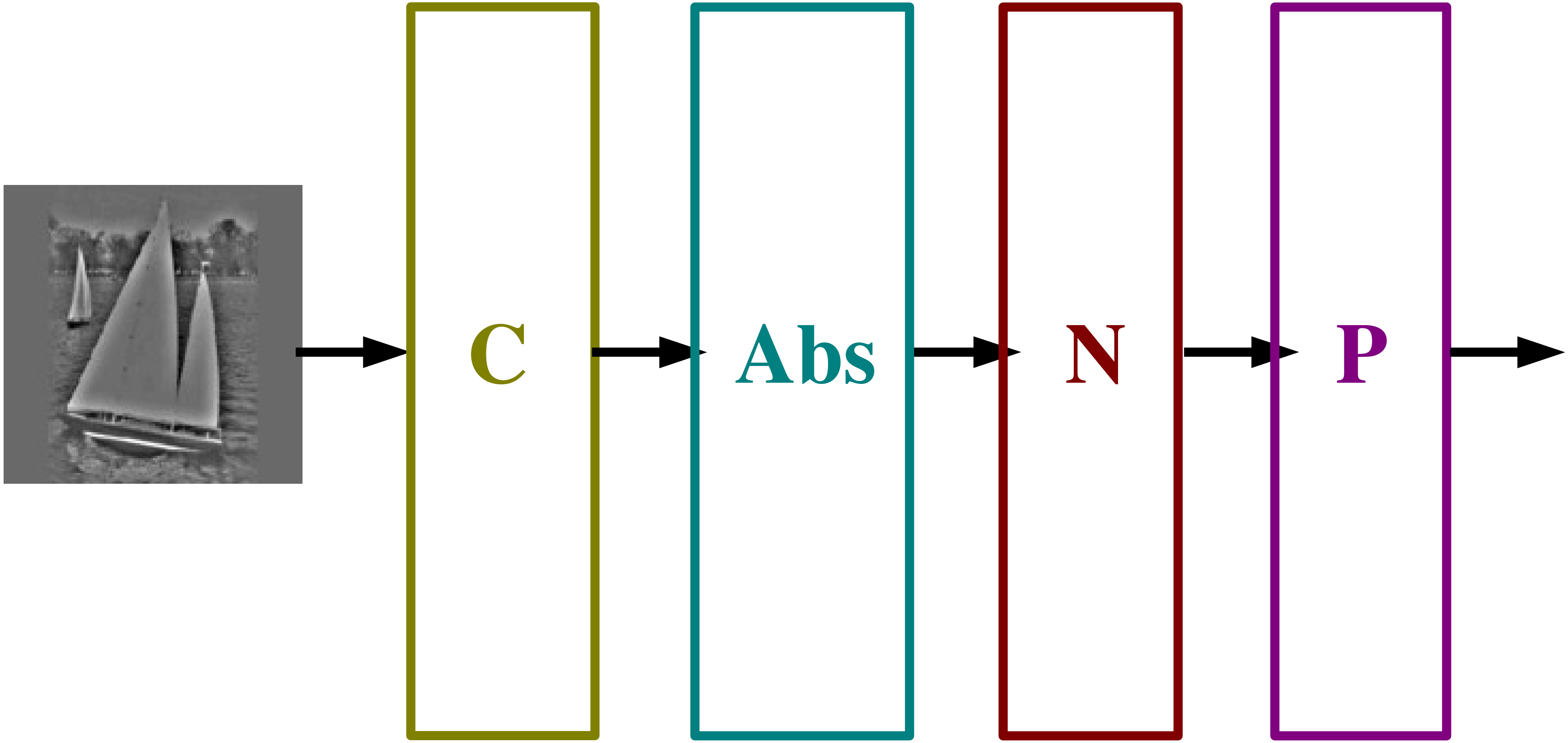
Abs

N

Pooling Down-Sampling Layer

Feature Extraction

- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N Normalization layer: needed?
- P Pooling down-sampling layer: average or max?



C

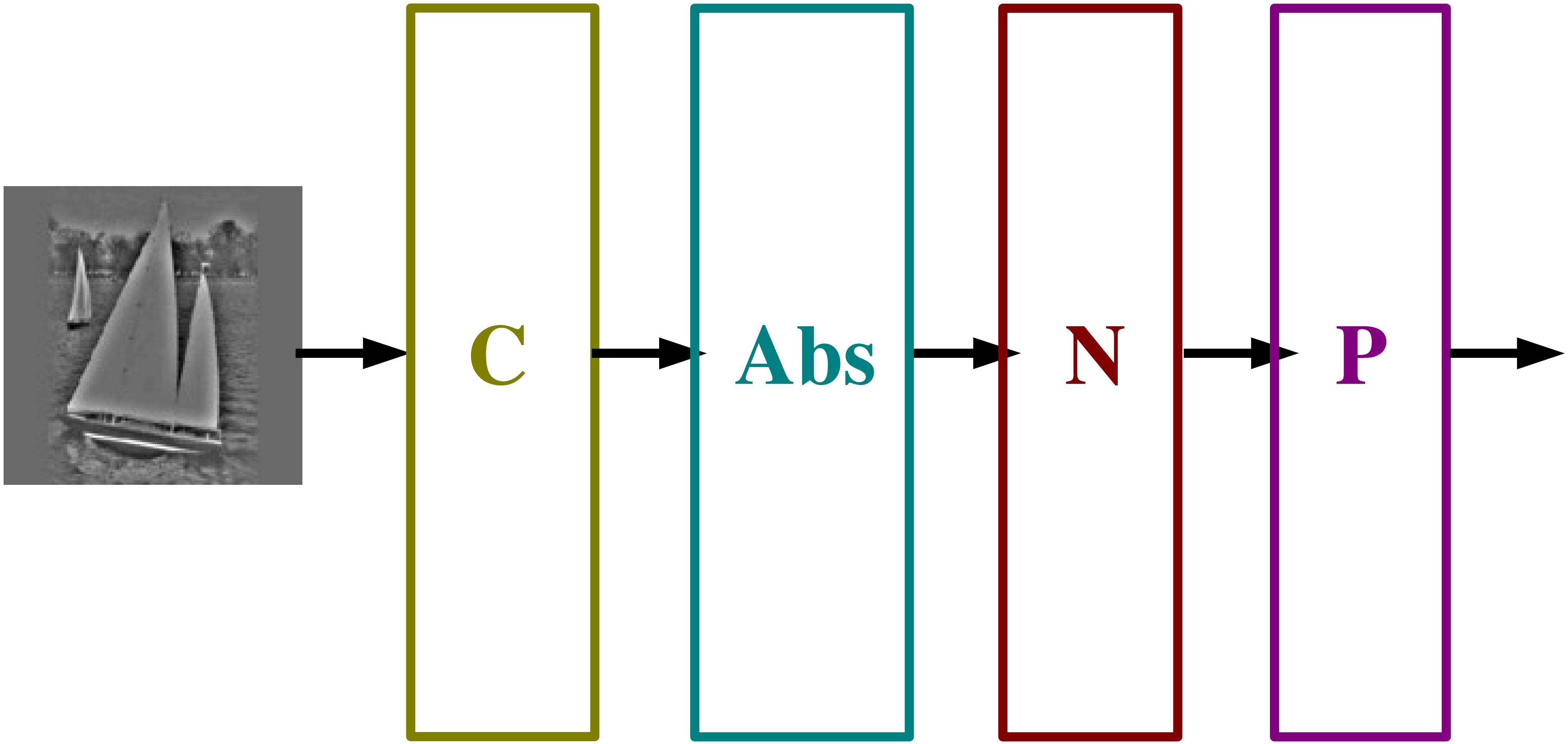
Abs

N

P

Feature Extraction

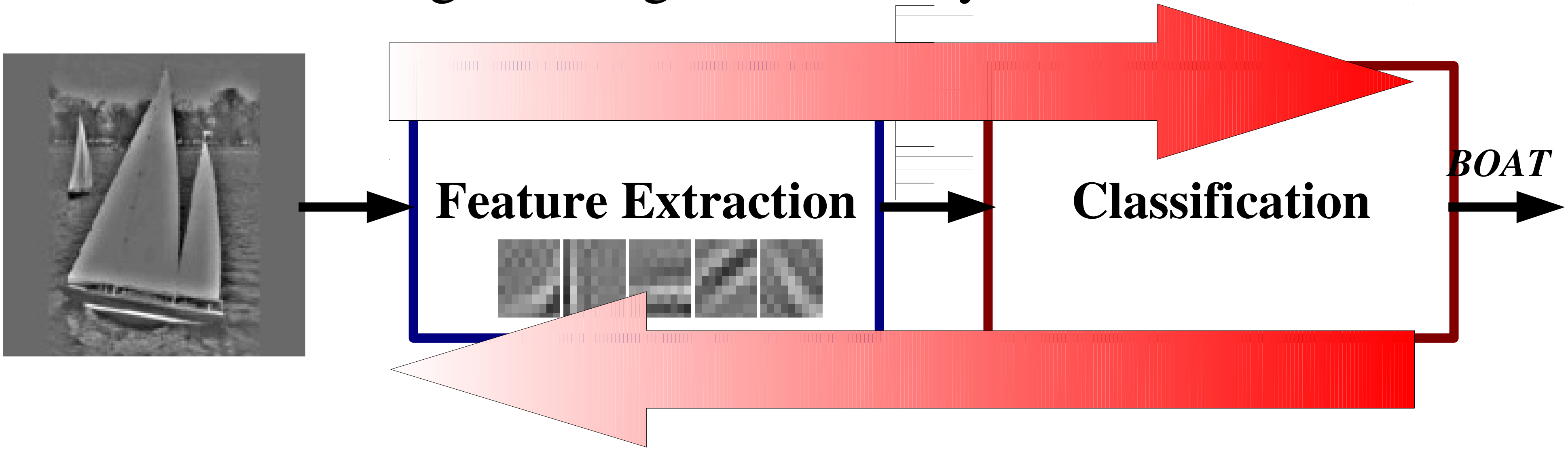
- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N Normalization layer: needed?
- P Pooling down-sampling layer: average or max?



THIS IS ONE STAGE OF FEATURE EXTRACTION

Training Protocol

- **Training**
  - Logistic Regression on Random Features: R
  - Logistic Regression on PSD features: U
  - Refinement of whole net from random with backprop: R^+
  - Refinement of whole net starting from PSD filters: U^+
- **Classifier**
  - Multinomial Logistic Regression or Pyramid Match Kernel SVM



Using PSD Features for Recognition

$[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ - 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)%
$[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ - PMK					
U	65.0%				

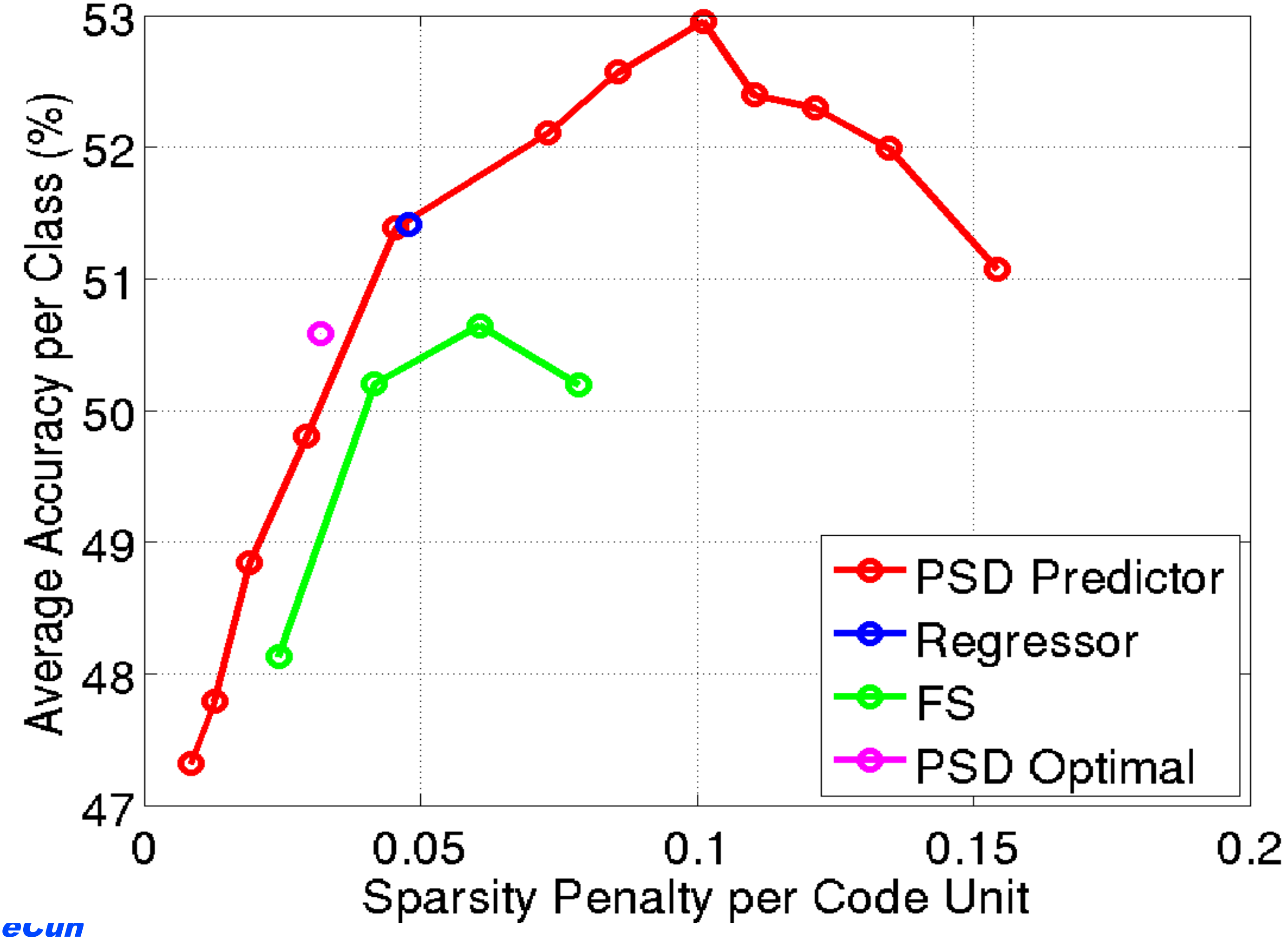
$[96.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ - PCA - lin_svm		
U	58.0%	
96.Gabors - PCA - lin_svm (Pinto and DiCarlo 2006)		
Gabors	59.0%	
SIFT - PMK (Lazebnik et al. CVPR 2006)		
Gabors	64.6%	

Using PSD Features for Recognition

- Rectification makes a huge difference:
  - 14.5% -> 50.0%, without normalization
  - 44.3% -> 54.2% with normalization
- Normalization makes a difference:
  - 50.0 → 54.2
- Unsupervised pretraining makes small difference
- PSD works just as well as SIFT
- Random filters work as well as anything!
  - If rectification/normalization is present
- PMK_SVM classifier works a lot better than multinomial log_reg on low-level features
  - 52.2% → 65.0%

Comparing Optimal Codes Predicted Codes on Caltech 101

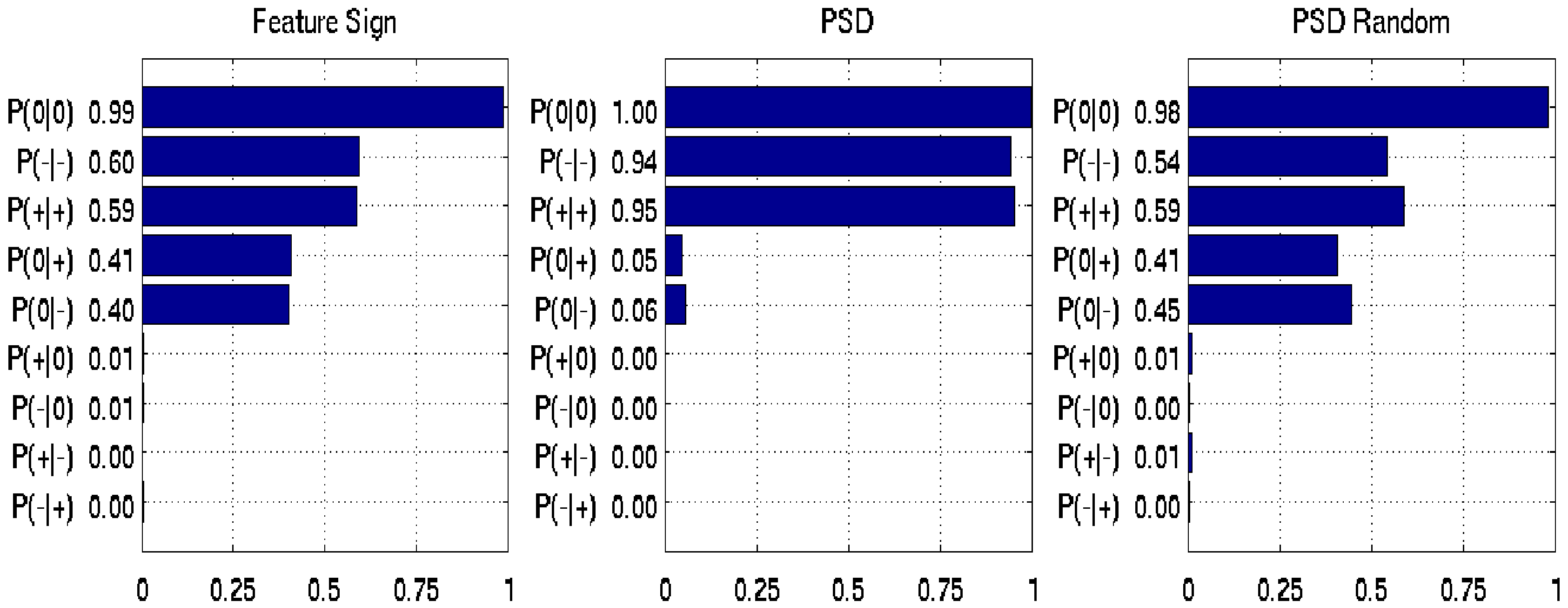
- **Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!**
  - PSD features are more stable.



Feature Sign (FS) is an optimization methods for computing sparse codes [Lee...Ng 2006]

PSD Features are more stable

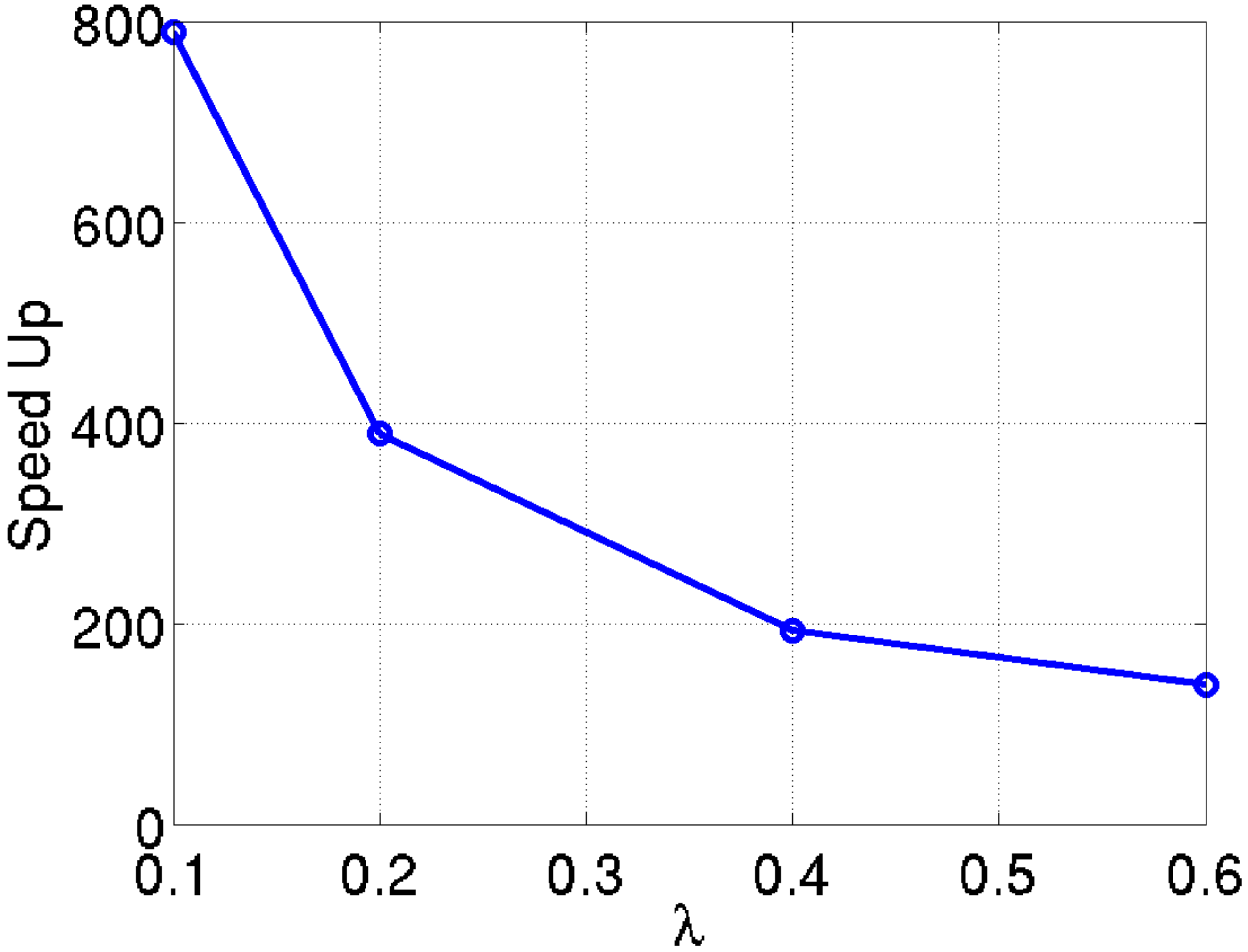
- **Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!**
- **Because PSD features are more stable. Feature obtained through sparse optimization can change a lot with small changes of the input.**



How many features change sign in patches from successive video frames (a,b), versus patches from random frame pairs (c)

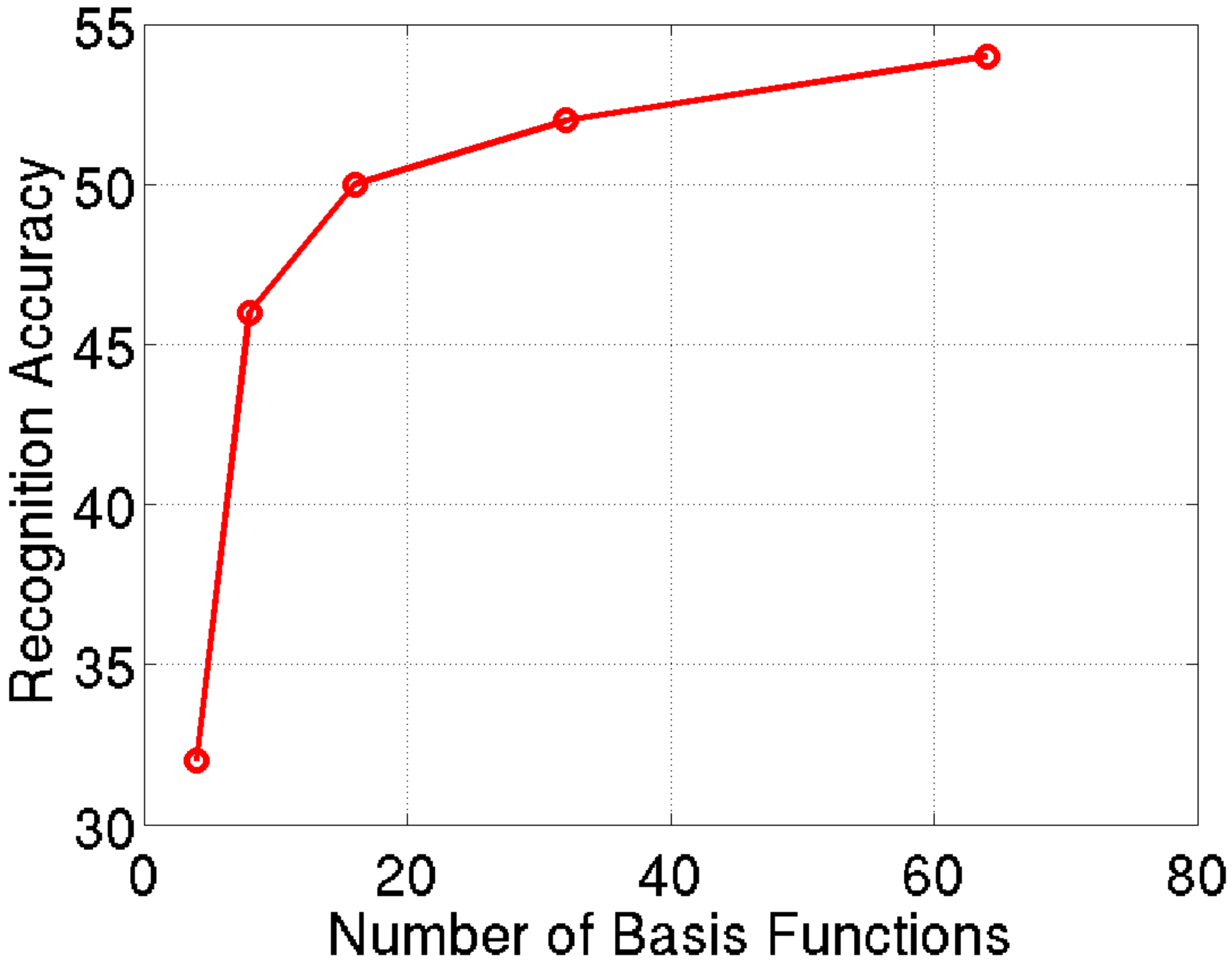
PSD features are much cheaper to compute

- **Computing PSD features is hundreds of times cheaper than Feature Sign.**

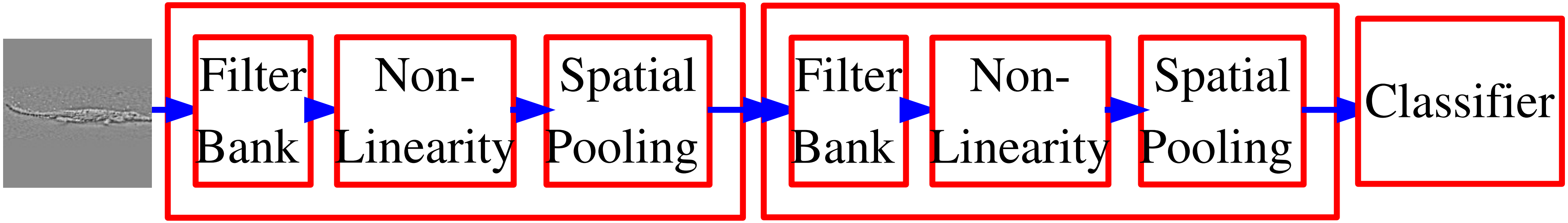


How Many 9x9 PSD features do we need?

- **Accuracy increases slowly past 64 filters.**

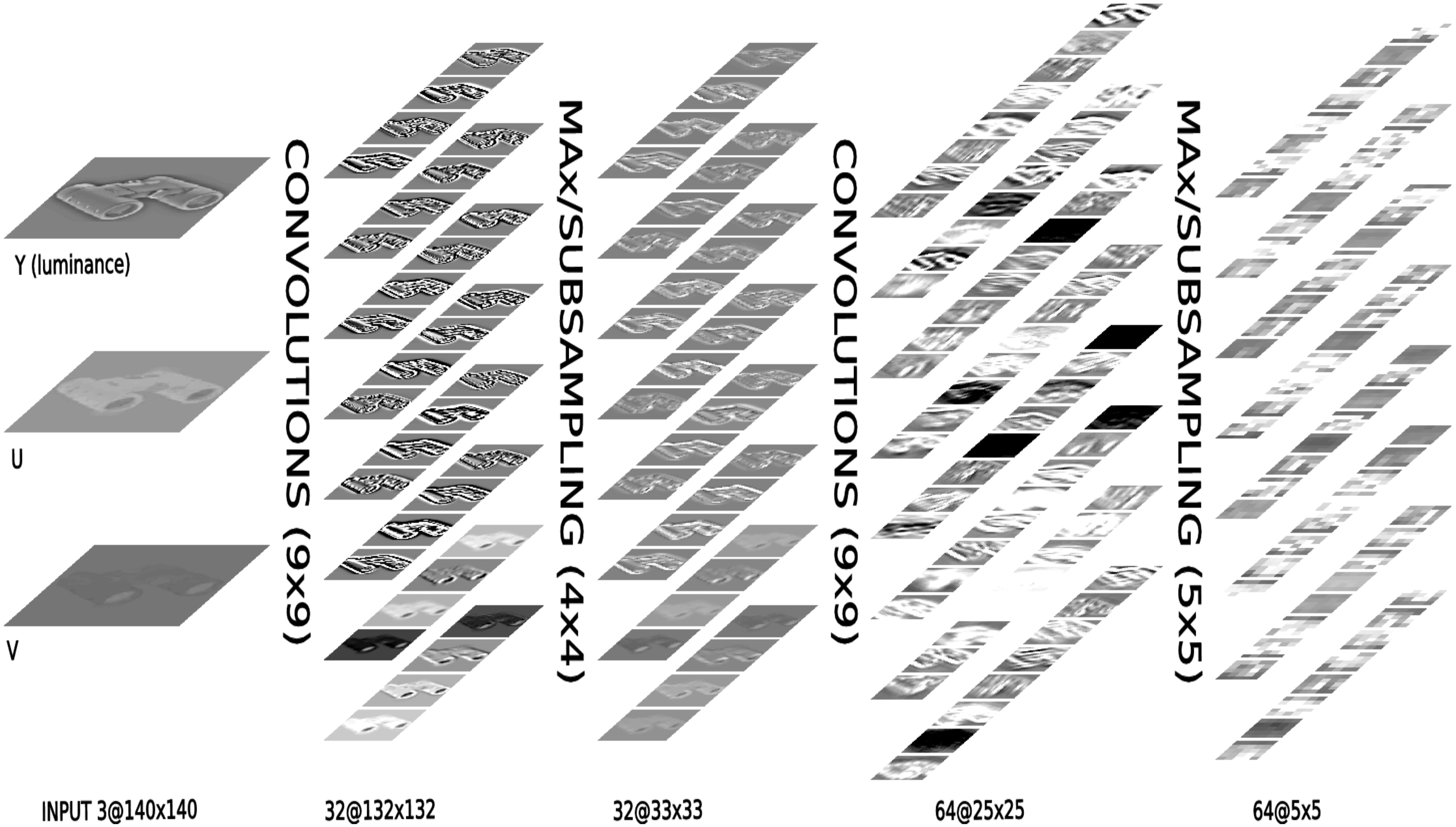


Training a Multi-Stage Hubel-Wiesel Architecture with PSD



- 1. Train stage-1 filters with PSD on patches from natural images
- 2. Compute stage-1 features on training set
- 3. Train state-2 filters with PSD on stage-1 feature patches
- 4. Compute stage-2 features on training set
- 5. Train linear classifier on stage-2 features
- 6. Refine entire network with supervised gradient descent
- What are the effects of the non-linearities and unsupervised pretraining?

Multistage Hubel-Wiesel Architecture on Caltech-101



Multistage Hubel-Wiesel Architecture

- **Image Preprocessing:**
  - High-pass filter, local contrast normalization (divisive)
- **First Stage:**
  - Filters: 64 9x9 kernels producing 64 feature maps
  - Pooling: 10x10 averaging with 5x5 subsampling
- **Second Stage:**
  - Filters: 4096 9x9 kernels producing 256 feature maps
  - Pooling: 6x6 averaging with 3x3 subsampling
  - Features: 256 feature maps of size 4x4 (4096 features)
- **Classifier Stage:**
  - Multinomial logistic regression
- **Number of parameters:**
  - Roughly 750,000

Multistage Hubel-Wiesel Architecture on Caltech-101

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%

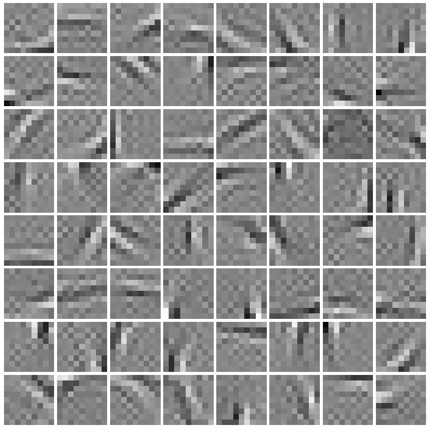
Two-Stage Result Analysis

- Second Stage + logistic regression = PMK_SVM
- Unsupervised pre-training doesn't help much :-(
- Random filters work amazingly well with normalization
- Supervised global refirnement helps a bit
- The best system is really cheap
- Either use rectification and average pooling or no rectification and max pooling.

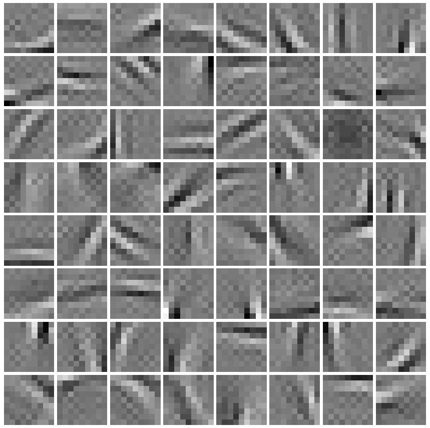
Multistage Hubel-Wiesel Architecture: Filters

- After PSD
- After supervised refinement

- Stage 1

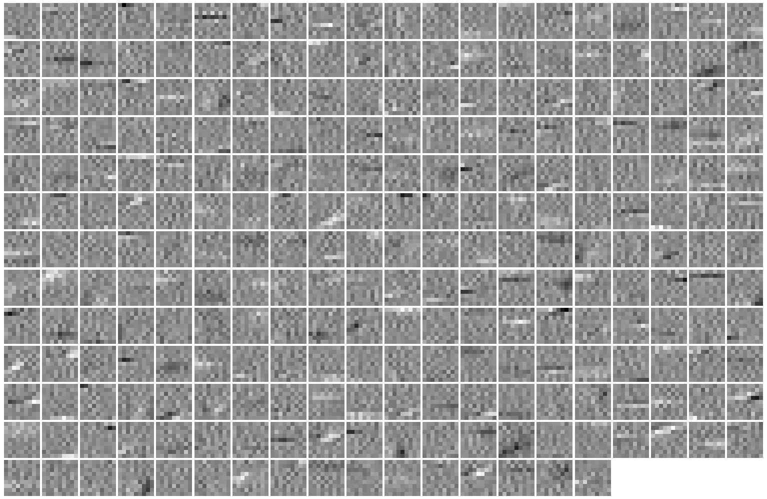


weights :-0.2232 - 0.2075

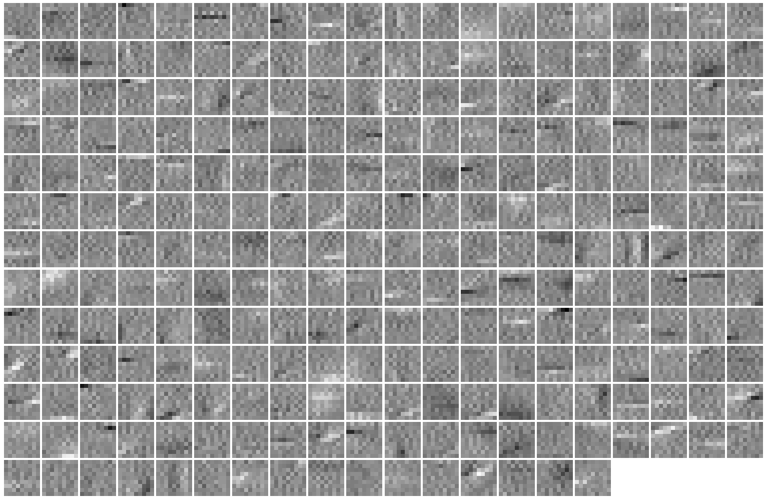


weights :-0.2828 - 0.3043

- Stage2



weights :-0.0778 - 0.064



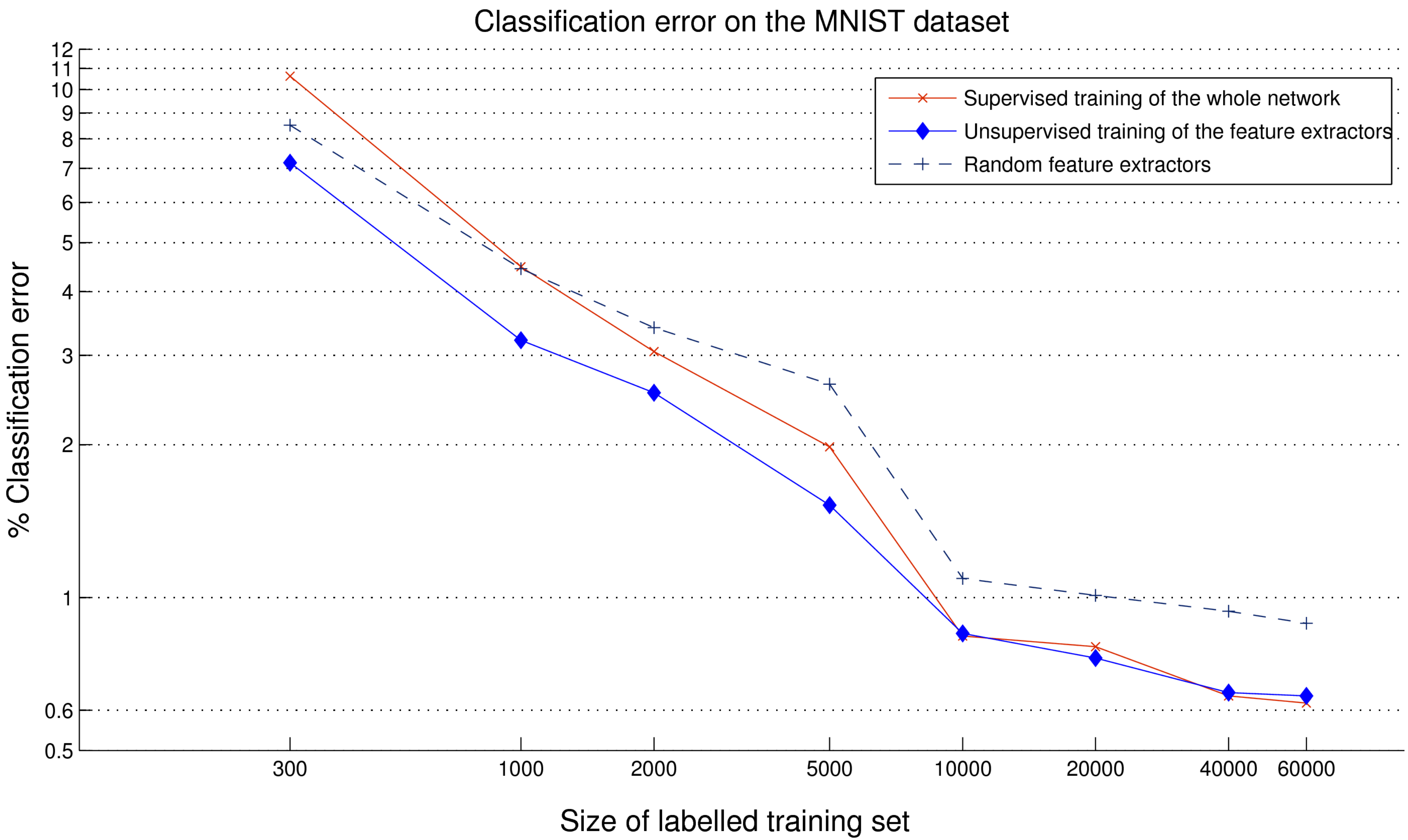
weights :-0.0929 - 0.0784

MNIST dataset

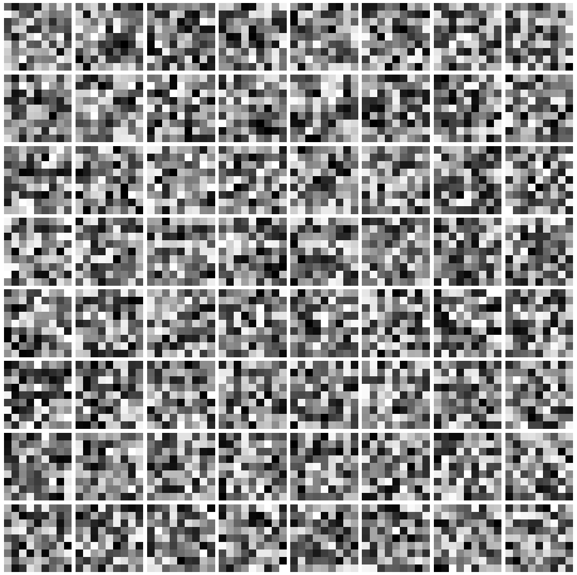
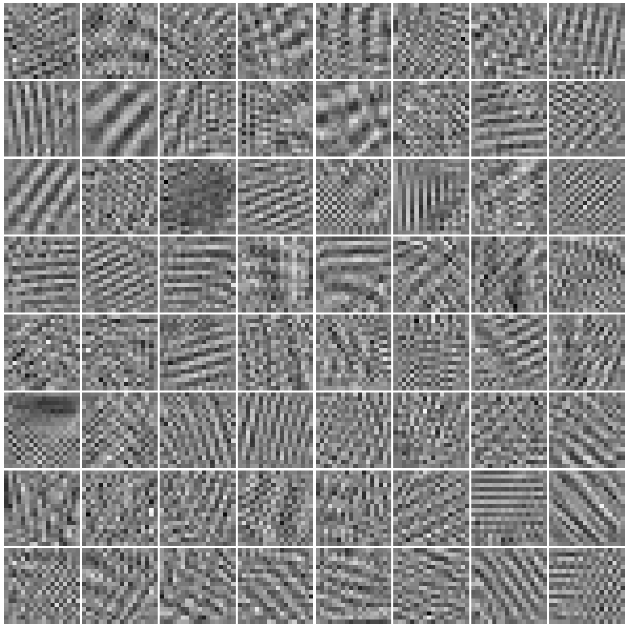
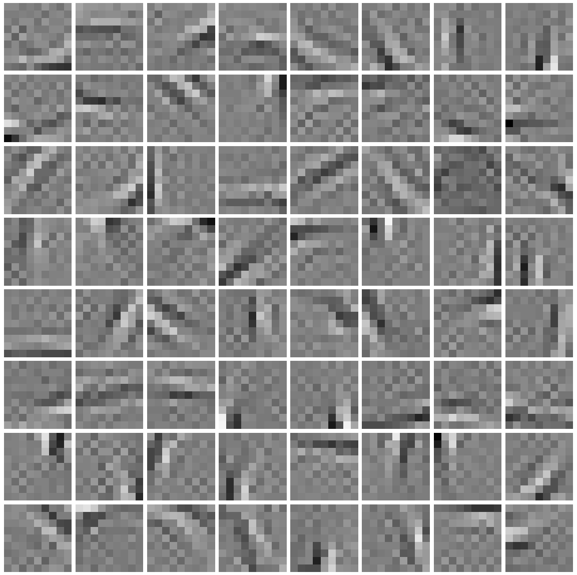
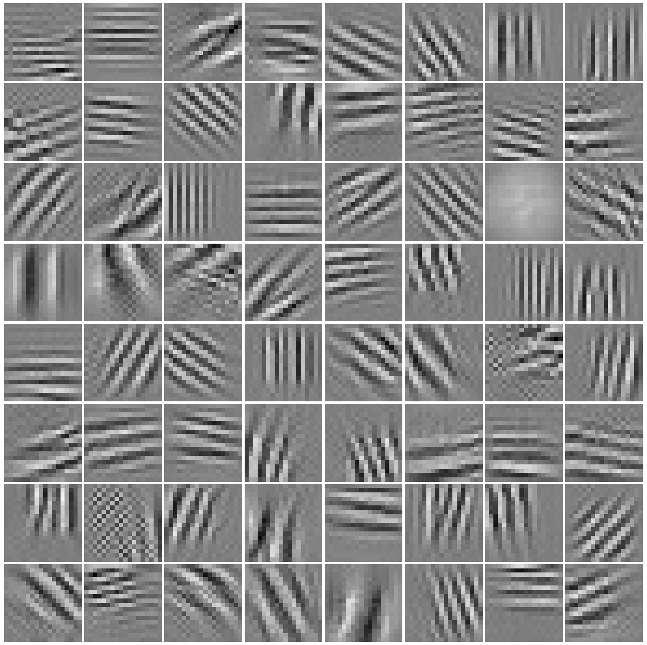
- 10 classes and up to 60,000 training samples per class

MNIST dataset

- **Architecture**
  - U^+U^+: 0.53% error (this is a record on the undistorted MNIST!)
  - Comparison: RR versus UU and R^+R^+

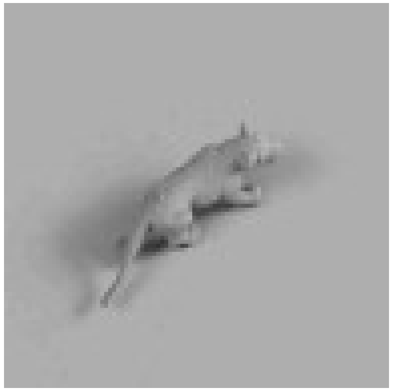


Why Random Filters Work?

Small NORB dataset

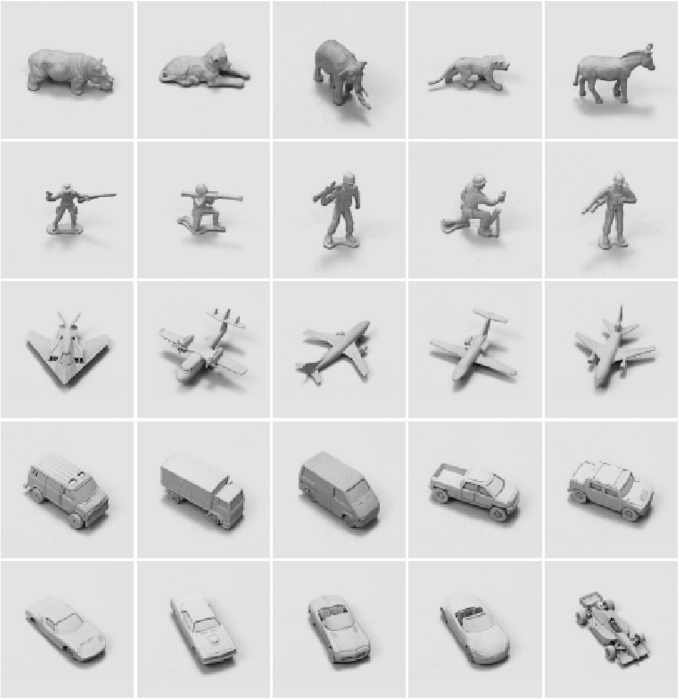
- 5 classes and up to 24,300 training samples per class



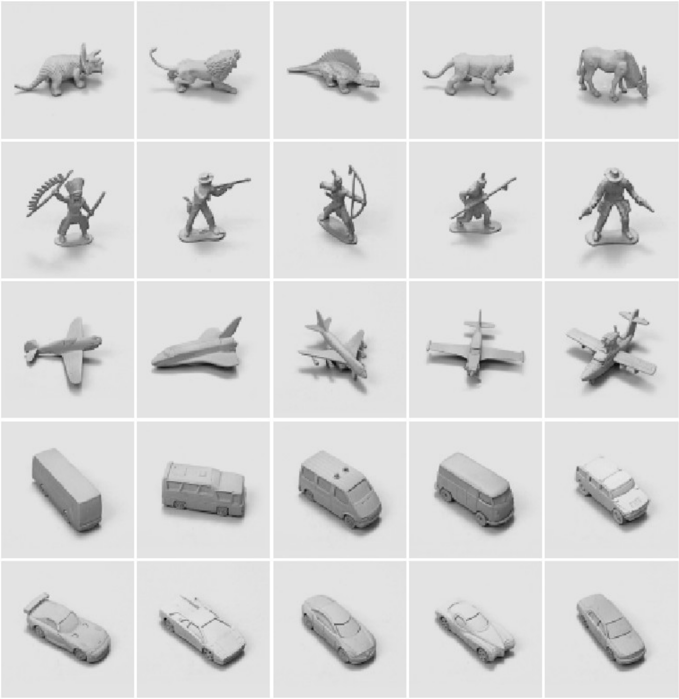
NORB Generic Object Recognition Dataset

- **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

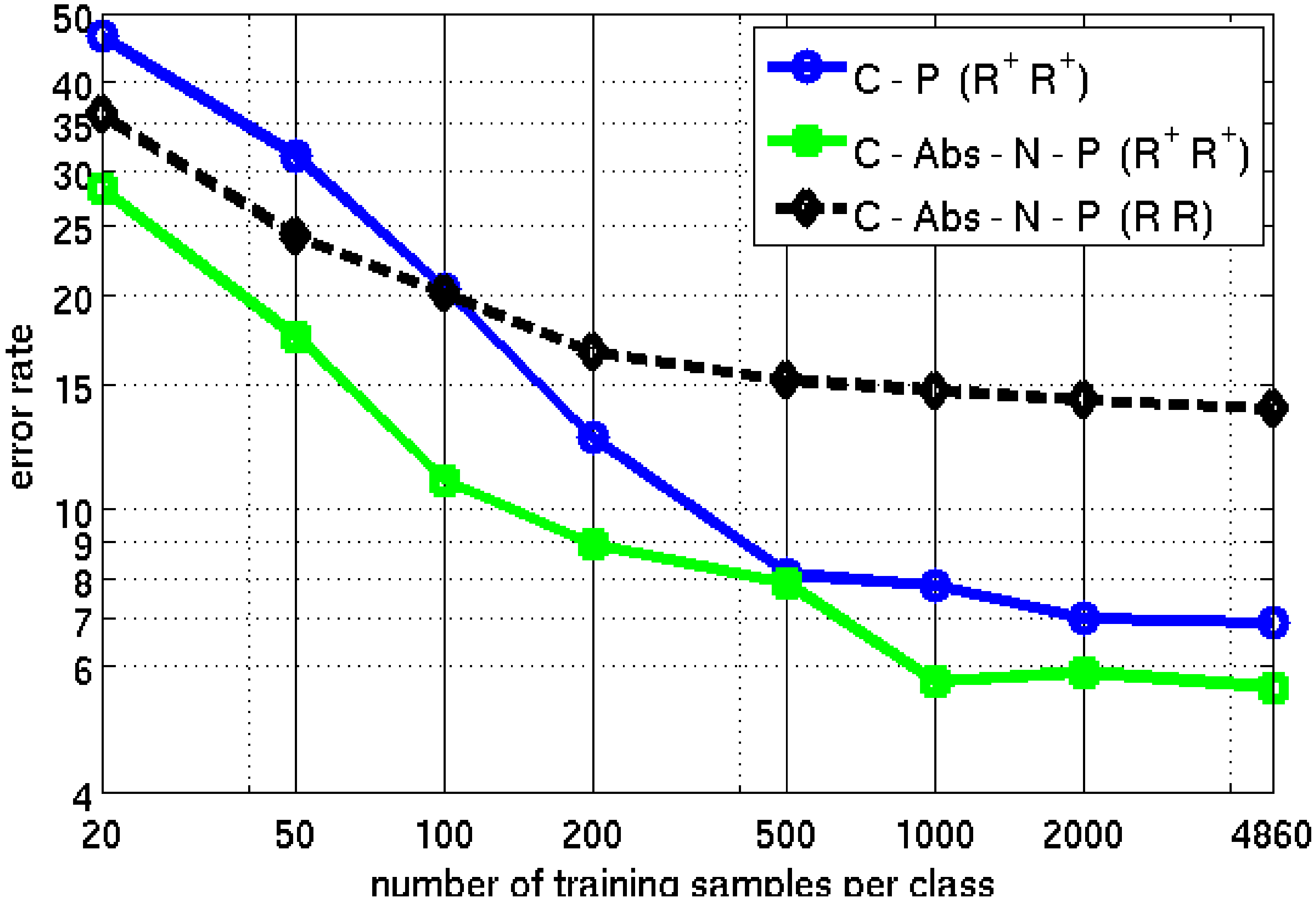


Test instances

Small NORB dataset

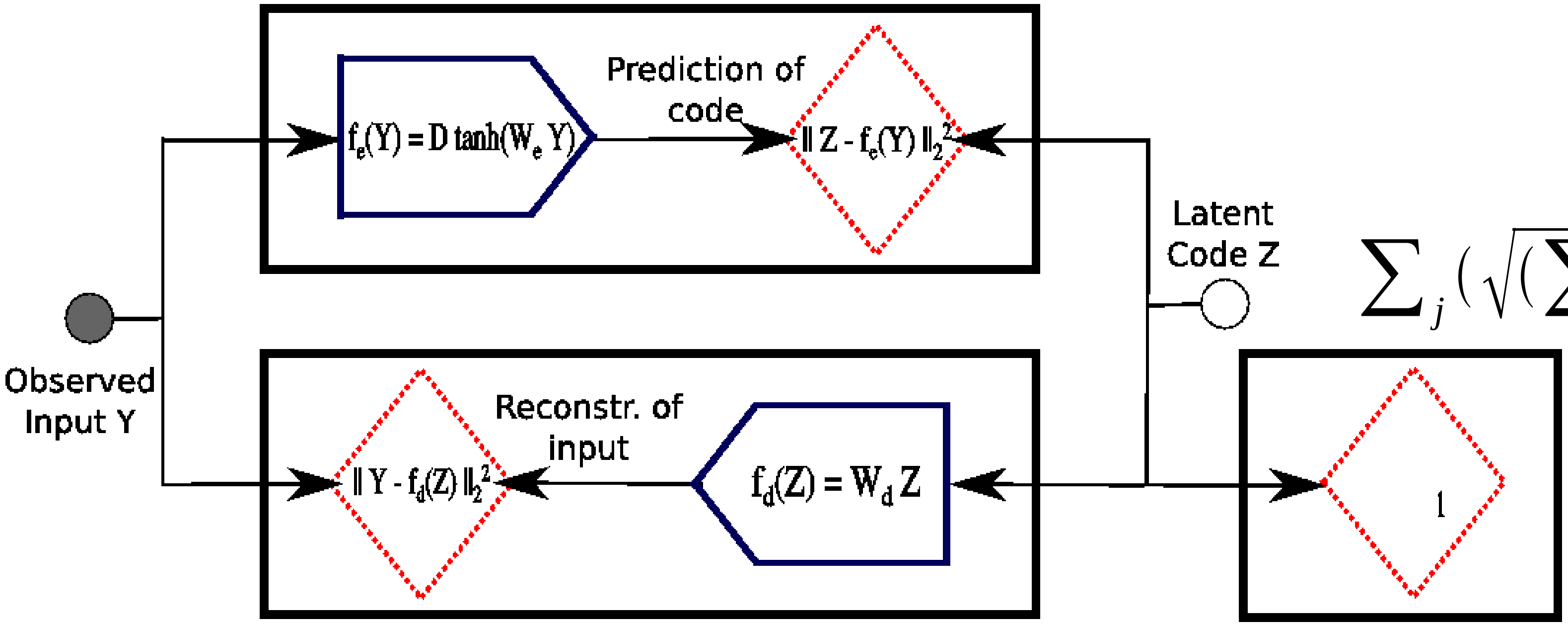
- **Architecture**
  - Two Stages

Error Rate (log scale) VS. **Number Training Samples** (log scale)



Learning Invariant Features [Kavukcuoglu et al. CVPR 2009]

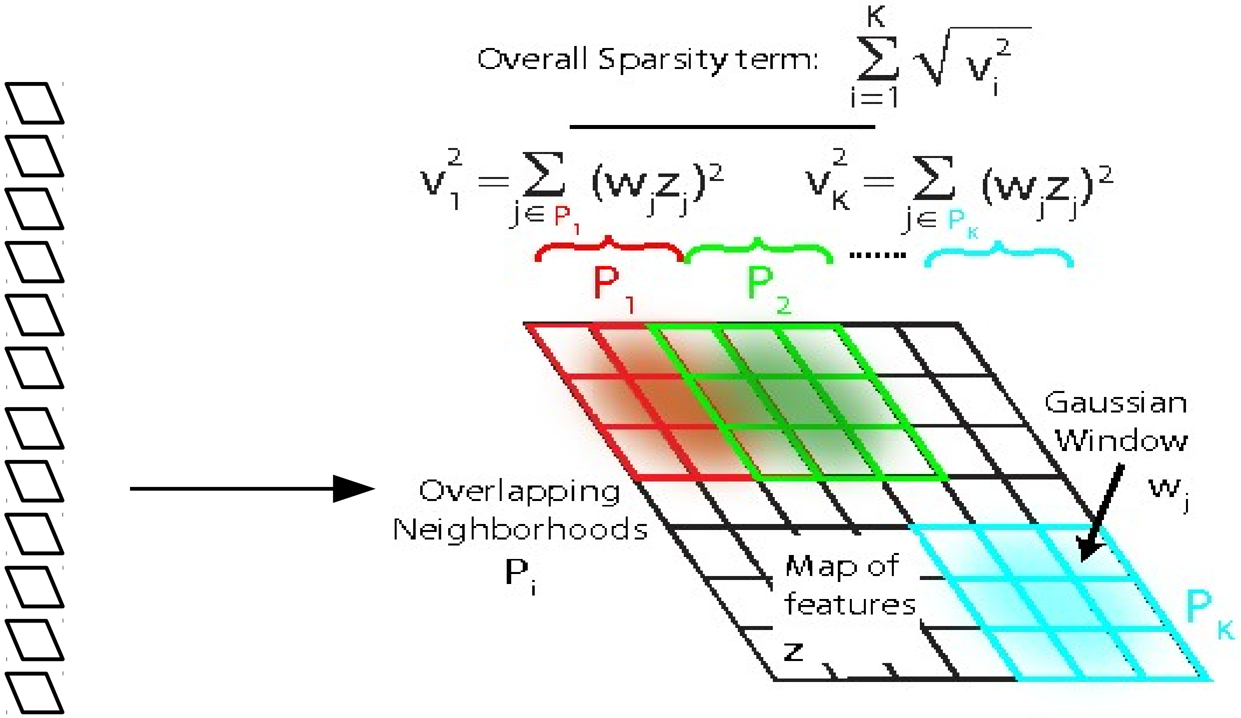
- **Unsupervised PSD ignores the spatial pooling step.**
- **Could we devise a similar method that learns the pooling layer as well?**
- **Idea [Hyvarinen & Hoyer 2001]: sparsity on pools of features**
  - Minimum number of pools must be non-zero
  - Number of features that are on within a pool doesn't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



$$\sum_j \left(\sqrt{\left(\sum_{(i \in P_j)} Z_i^2\right)}\right)$$

Learning the filters and the pools

- **Using an idea from Hyvarinen: topographic square pooling (subspace ICA)**
  - 1. Apply filters on a patch (with suitable non-linearity)
  - 2. Arrange filter outputs on a 2D plane
  - 3. square filter outputs
  - 4. minimize sqrt of sum of blocks of squared filter outputs

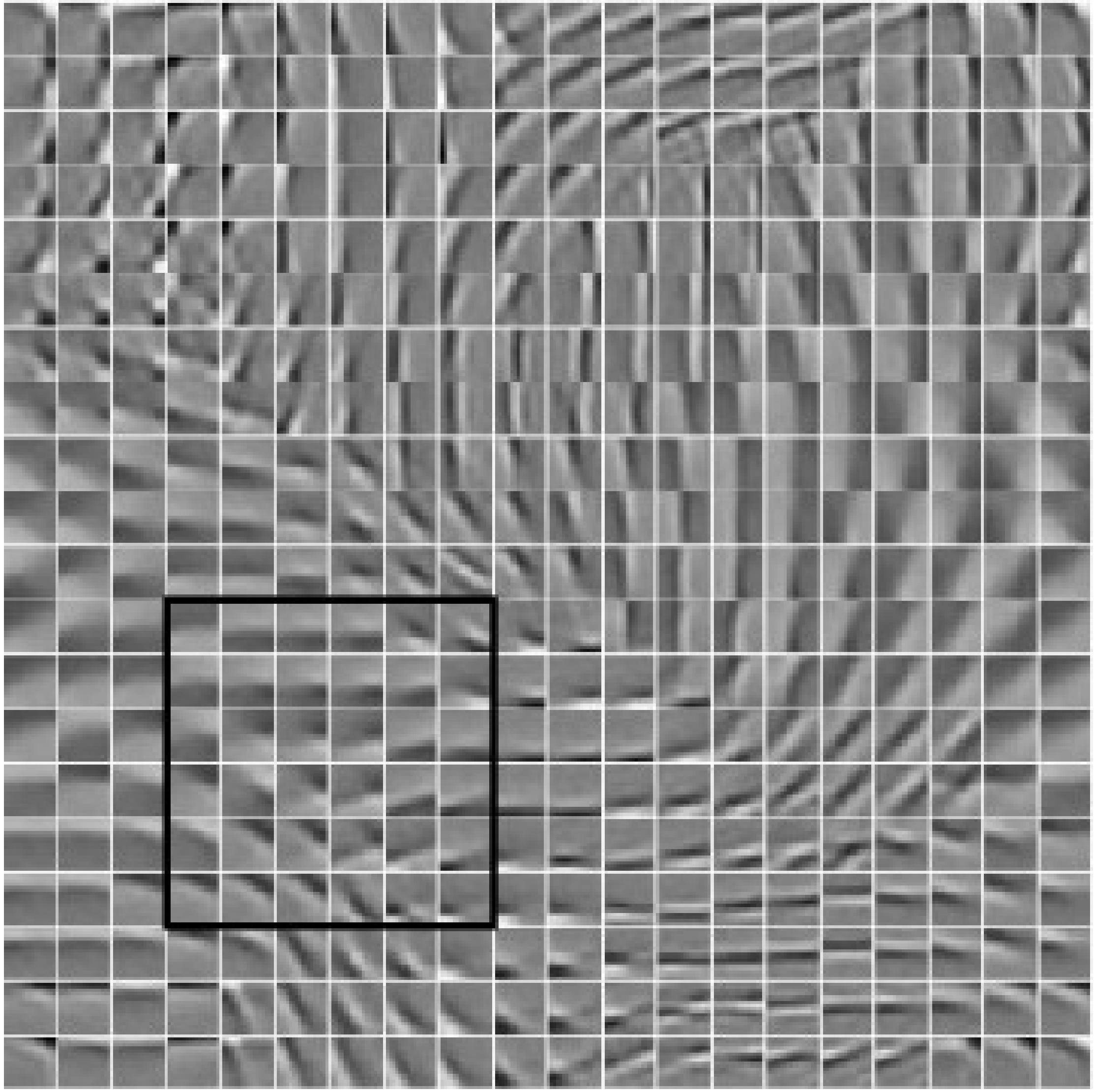

Overall Sparsity term: $\sum_{i=1}^{K} \sqrt{v_i^2}$
$v_1^2 = \sum_{j \in P_1} (w_j z_j)^2$
$v_K^2 = \sum_{j \in P_K} (w_j z_j)^2$
P_1
P_2
....
Gaussian Window w_j
Overlapping Neighborhoods P_i
Map of features
z
P_K

Units in the code Z

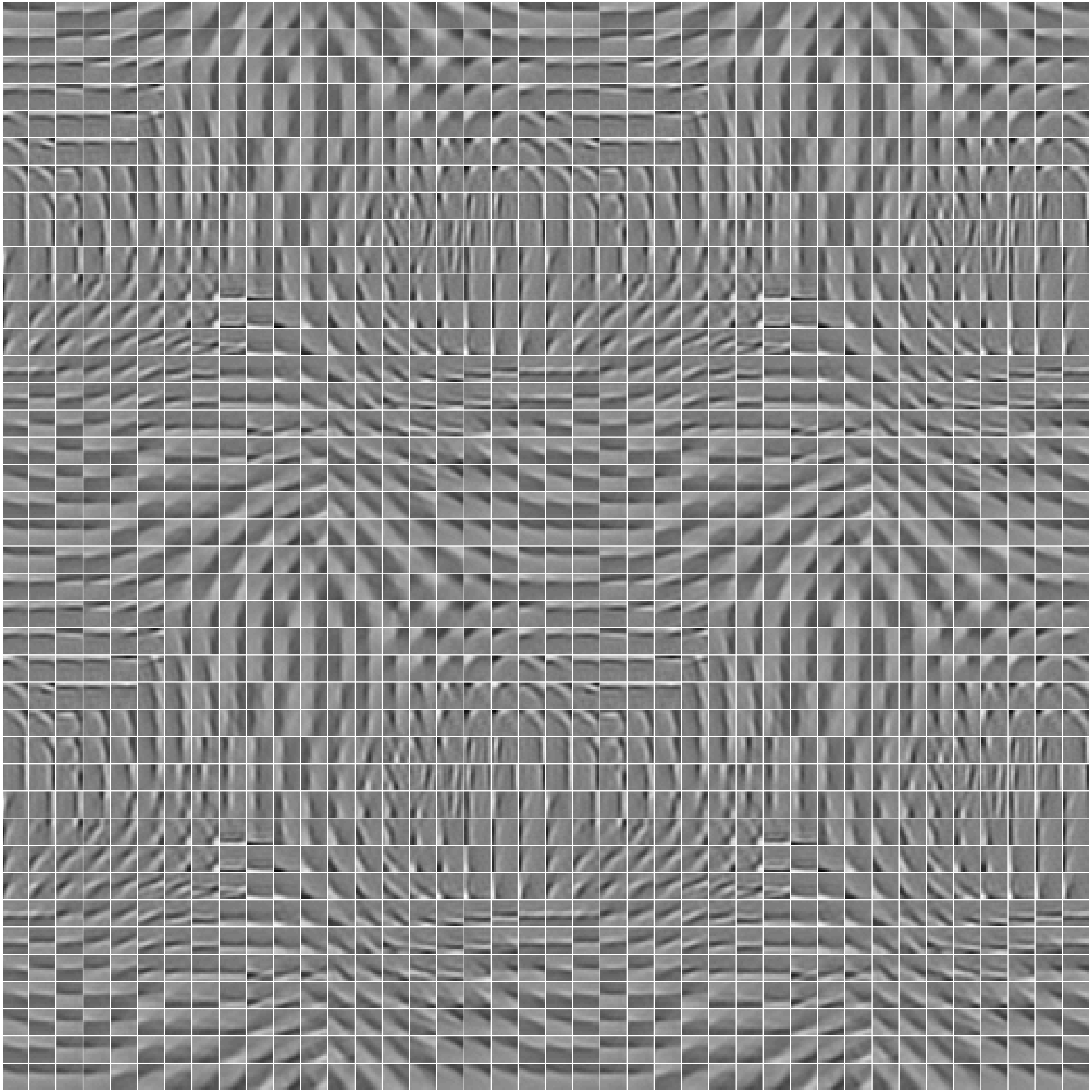
Define pools and enforce sparsity across pools

Learning the filters and the pools

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

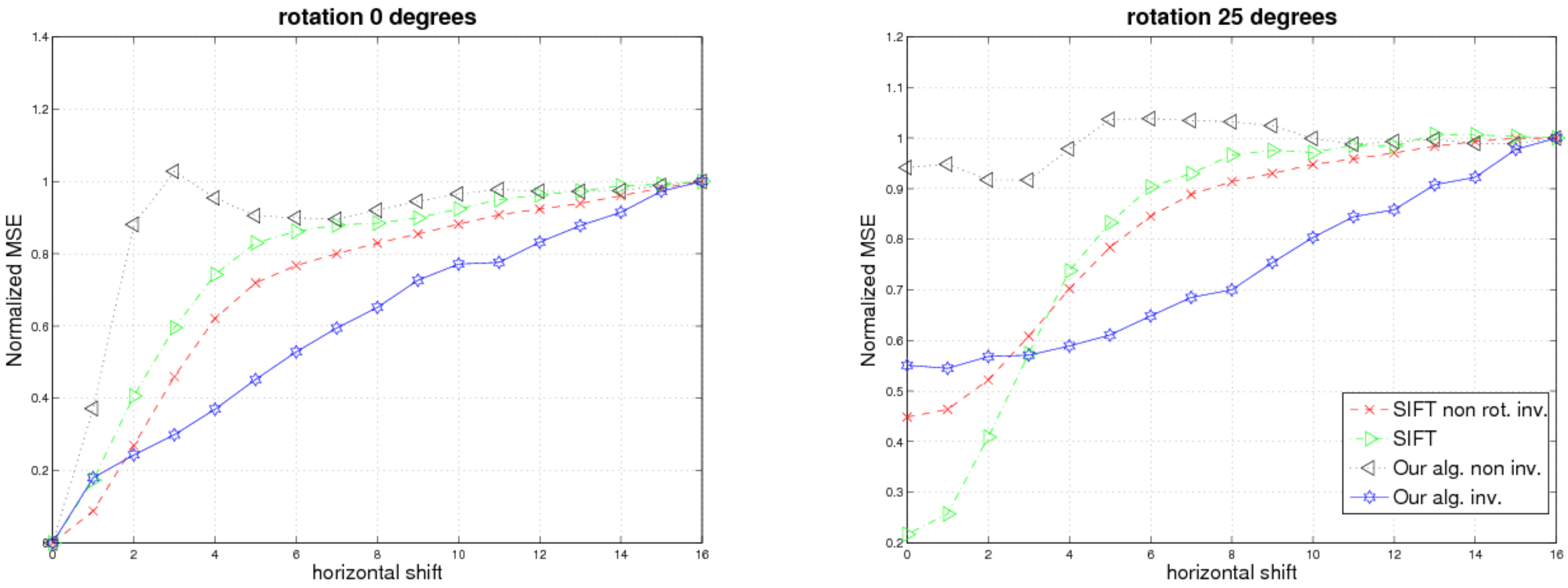


Pinwheels?



Invariance Properties Compared to SIFT

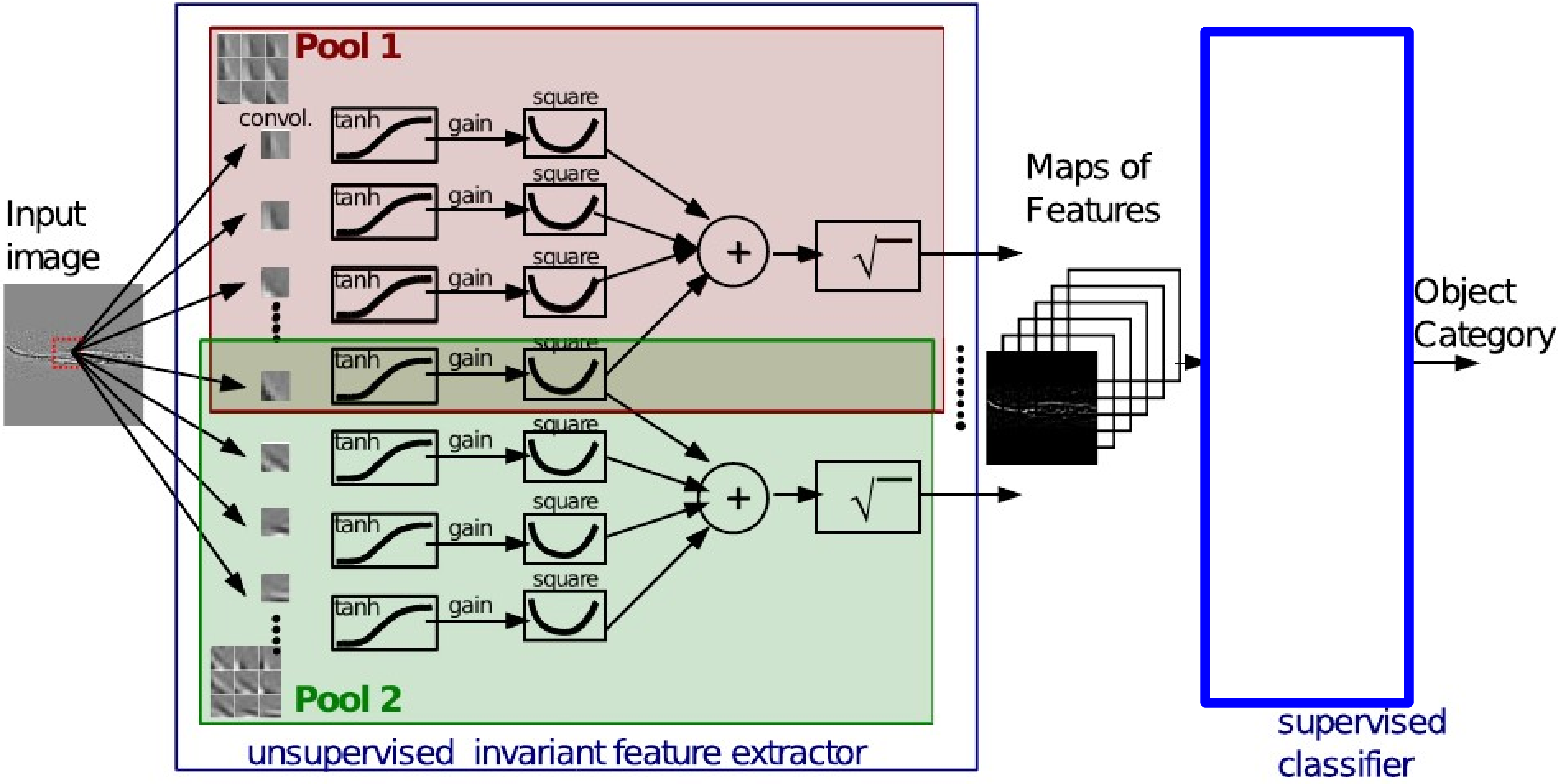
- **Measure distance between feature vectors (128 dimensions) of 16x16 patches from natural images**
  - Left: normalized distance as a function of translation
  - Right: normalized distance as a function of translation when one patch is rotated 25 degrees.
- **Topographic PSD features are more invariant than SIFT**



Learning Invariant Features

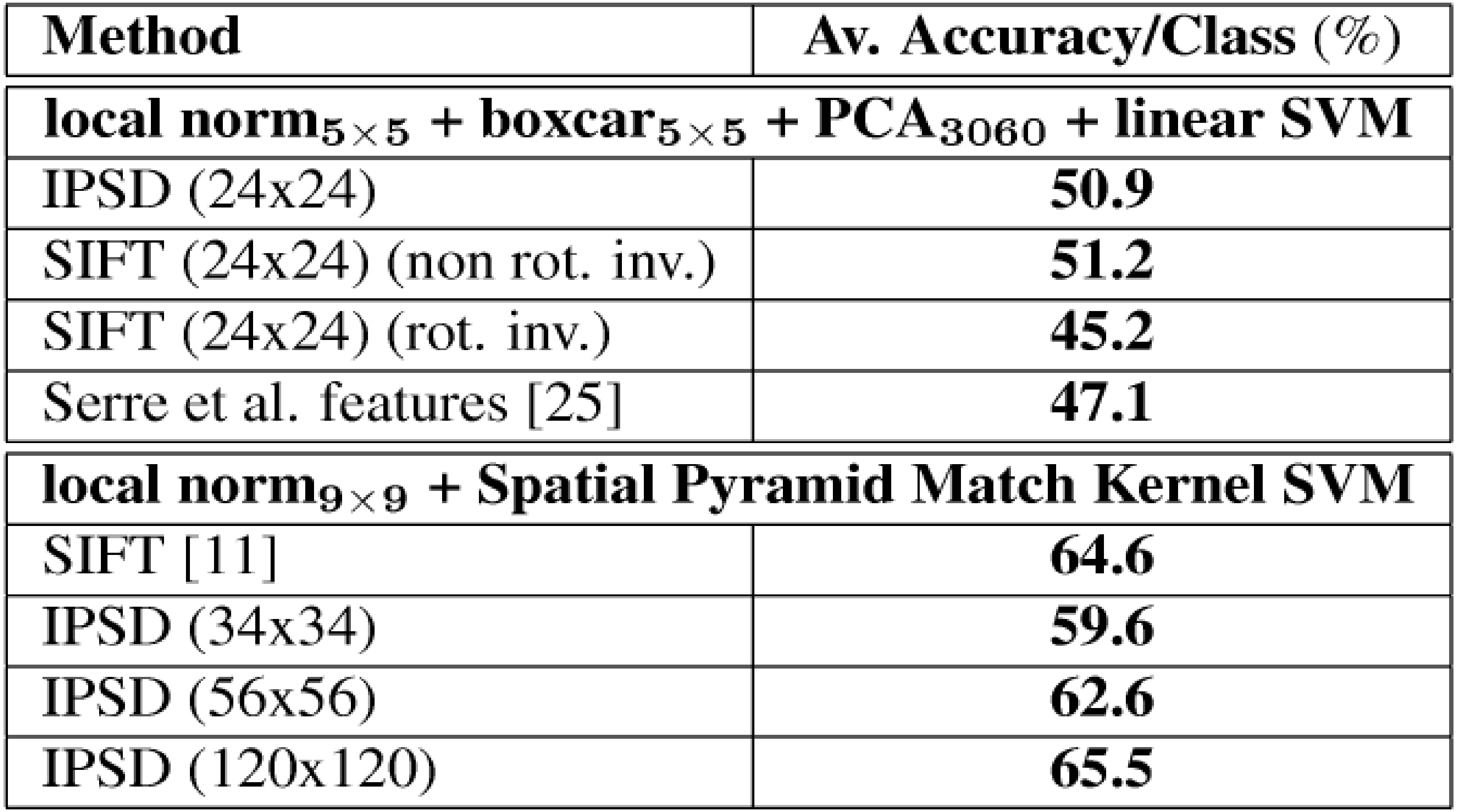
Recognition Architecture

- ->HPF/LCN->filters->tanh->sqr->pooling->sqrt->Classifier
- Block pooling plays the same role as rectification



Recognition Accuracy on Caltech 101

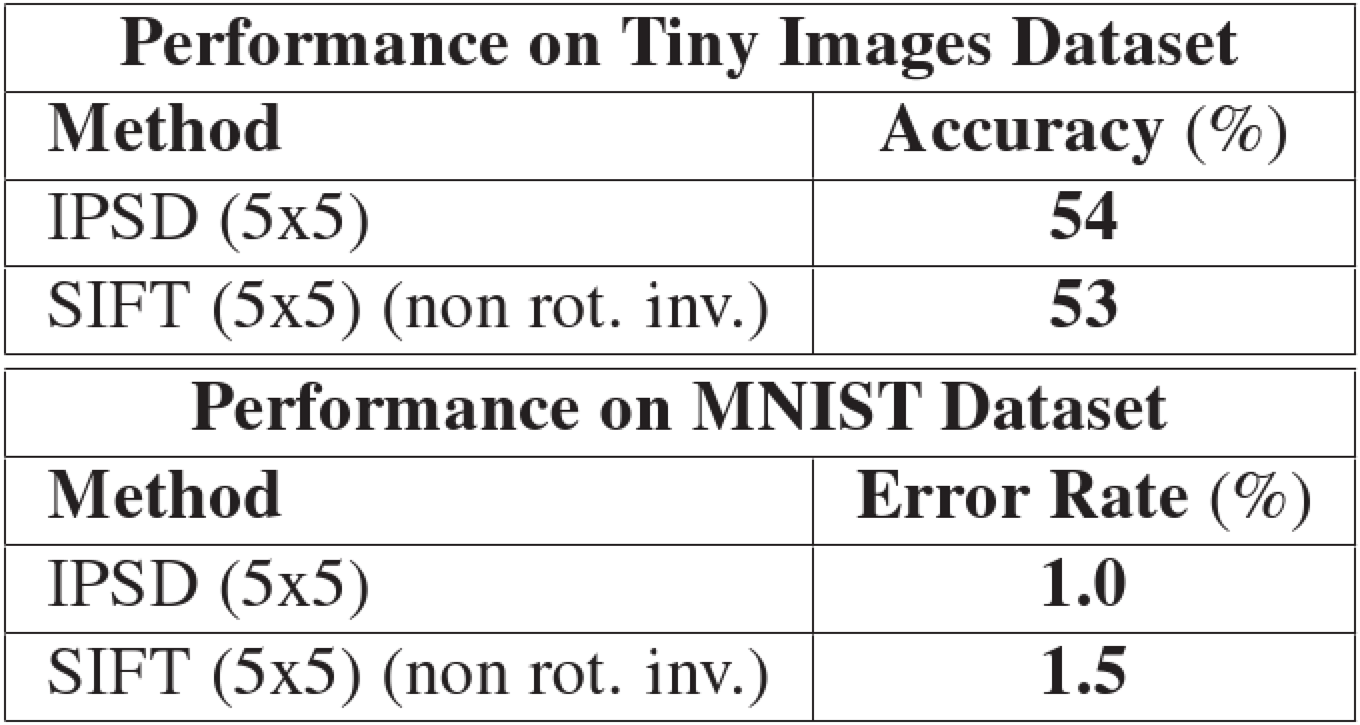
- A/B Comparison with SIFT (128x34x34 descriptors)
- 32x16 topographic map with 16x16 filters
- Pooling performed over 6x6 with 2x2 subsampling
- 128 dimensional feature vector per 16x16 patch
- Feature vector computed every 4x4 pixels (128x34x34 feature maps)
- Resulting feature maps are spatially smoothed



Method	Av. Accuracy/Class (%)
local norm$_{5\times5}$ + boxcar$_{5\times5}$ + PCA$_{3060}$ + linear SVM	
IPSD (24x24)	**50.9**
SIFT (24x24) (non rot. inv.)	**51.2**
SIFT (24x24) (rot. inv.)	**45.2**
Serre et al. features [25]	**47.1**
local norm$_{9\times9}$ + Spatial Pyramid Match Kernel SVM	
SIFT [11]	**64.6**
IPSD (34x34)	**59.6**
IPSD (56x56)	**62.6**
IPSD (120x120)	**65.5**

Recognition Accuracy on Tiny Images & MNIST

- A/B Comparison with SIFT (128x5x5 descriptors)
- 32x16 topographic map with 16x16 filters.



Performance on Tiny Images Dataset	
Method	**Accuracy** (%)
IPSD (5x5)	**54**
SIFT (5x5) (non rot. inv.)	**53**

Performance on MNIST Dataset	
Method	**Error Rate** (%)
IPSD (5x5)	**1.0**
SIFT (5x5) (non rot. inv.)	**1.5**

The End