Learning Feature Hierarchies for Vision

Yann LeCun

Courant Institute of Mathematical Sciences and

Center for Neural Science, New York University

Collaborators:

Rob Fergus,

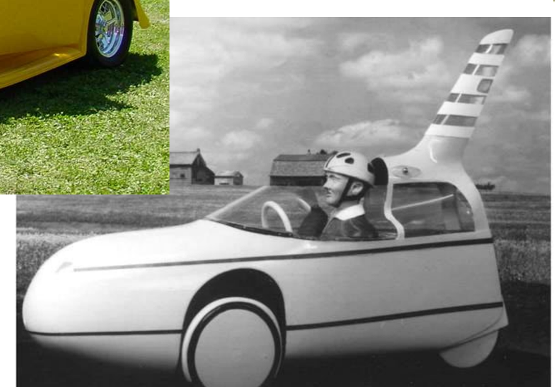
Karol Gregor, Arthur Szlam, Graham Taylor

Y-Lan Boureau, Benoit Corda, Clément Farabet

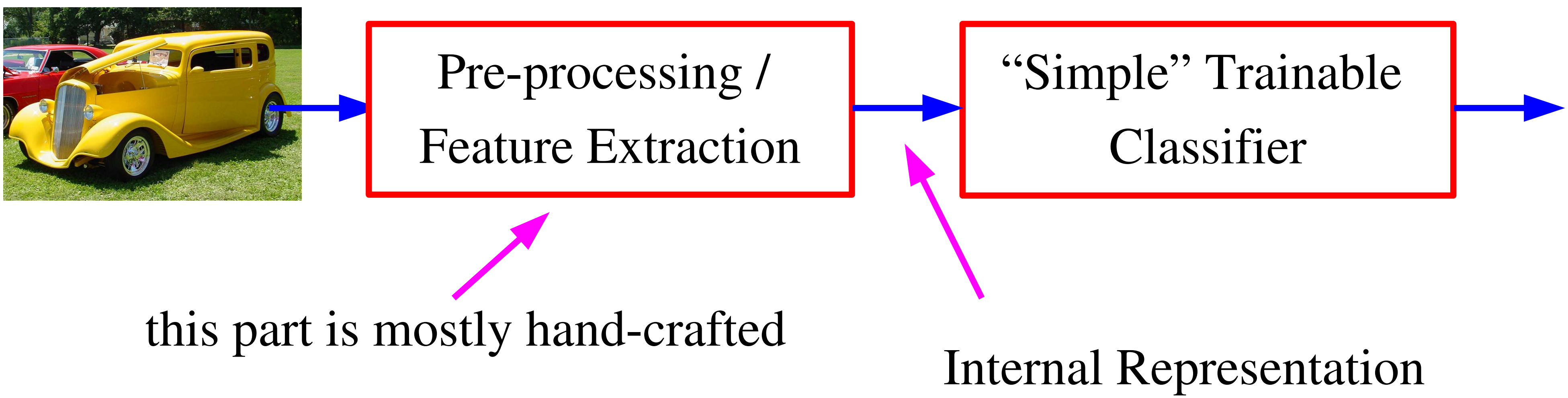
Kevin Jarrett, Koray Kavukcuoglu, Pierre Sermanet

The Next Challenge for AI, Robotics, and Neuroscience

- How do we learn perception (e.g. vision)?
- How do we learn representations of the perceptual world?
- How do we learn visual categories from just a few examples?



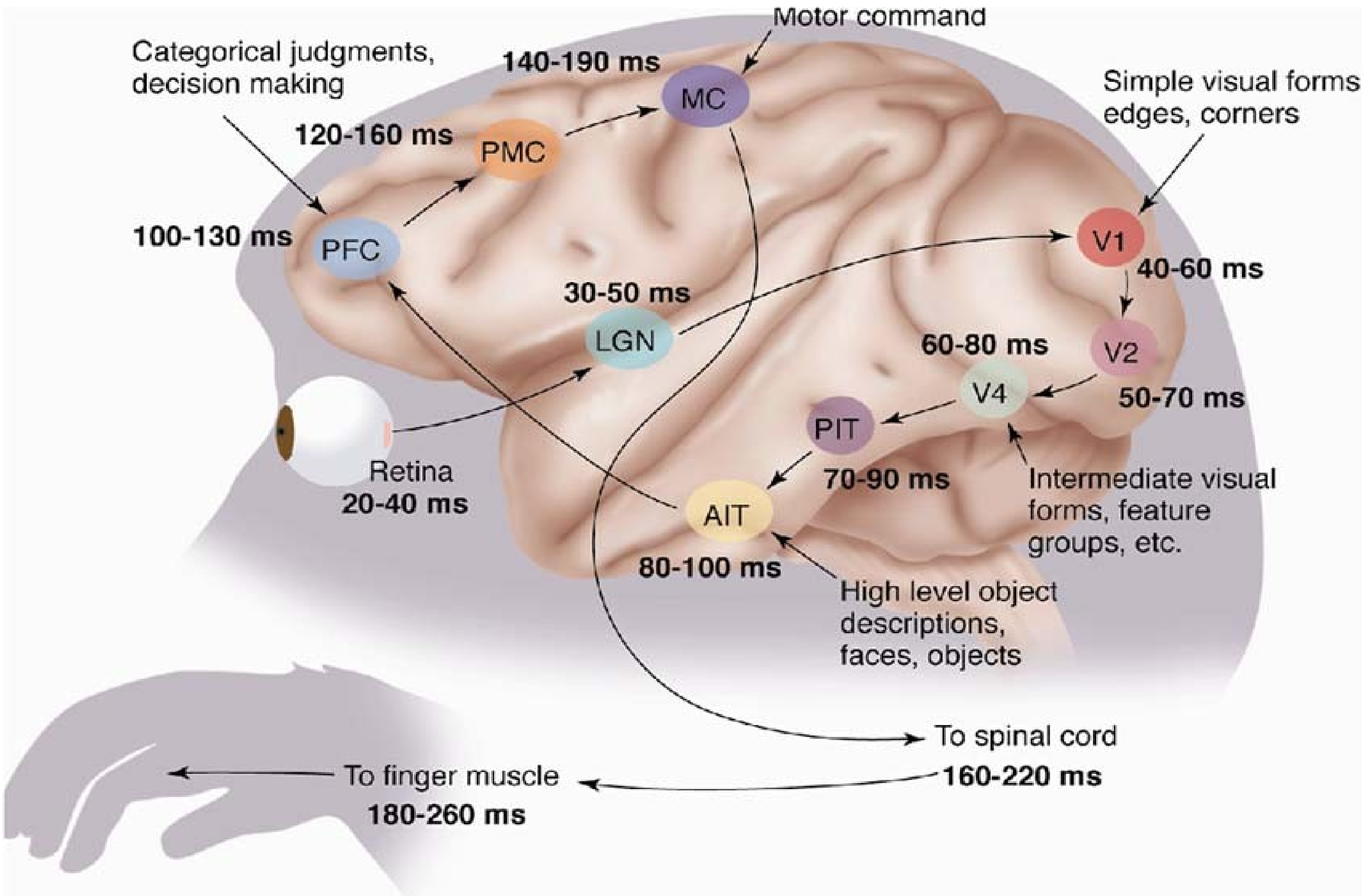
The Traditional "Shallow" Architecture for Recognition



- **The raw input is pre-processed through a hand-crafted feature extractor**
- **The features are not learned**
- **The trainable classifier is often generic (task independent), and "simple" (linear classifier, kernel machine, nearest neighbor,.....)**
- **The most common Machine Learning architecture: the Kernel Machine**

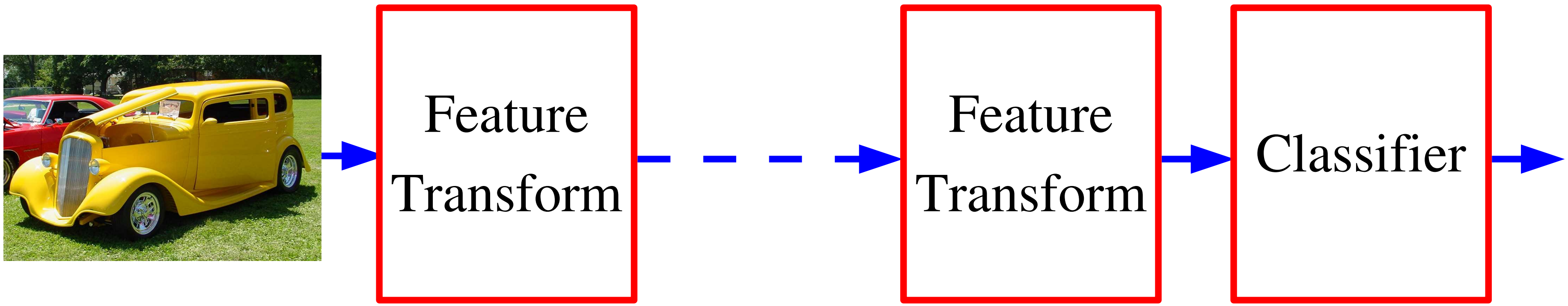
...But the Mammalian Visual Cortex is Hierarchical. Why?

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



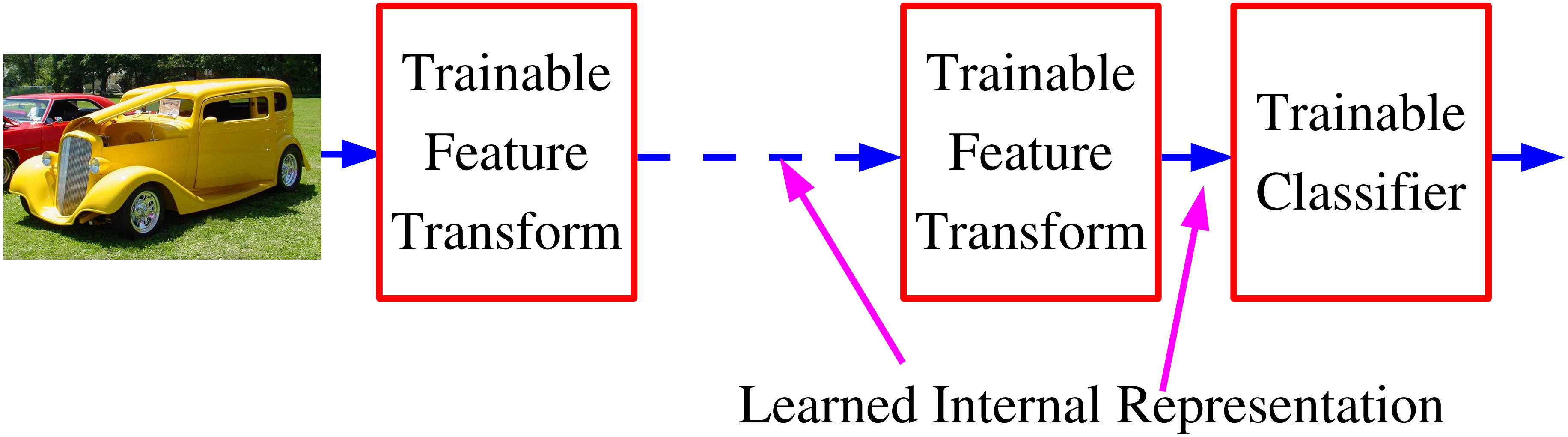
[picture from Simon Thorpe]

Good Internal Representations are Hierarchical



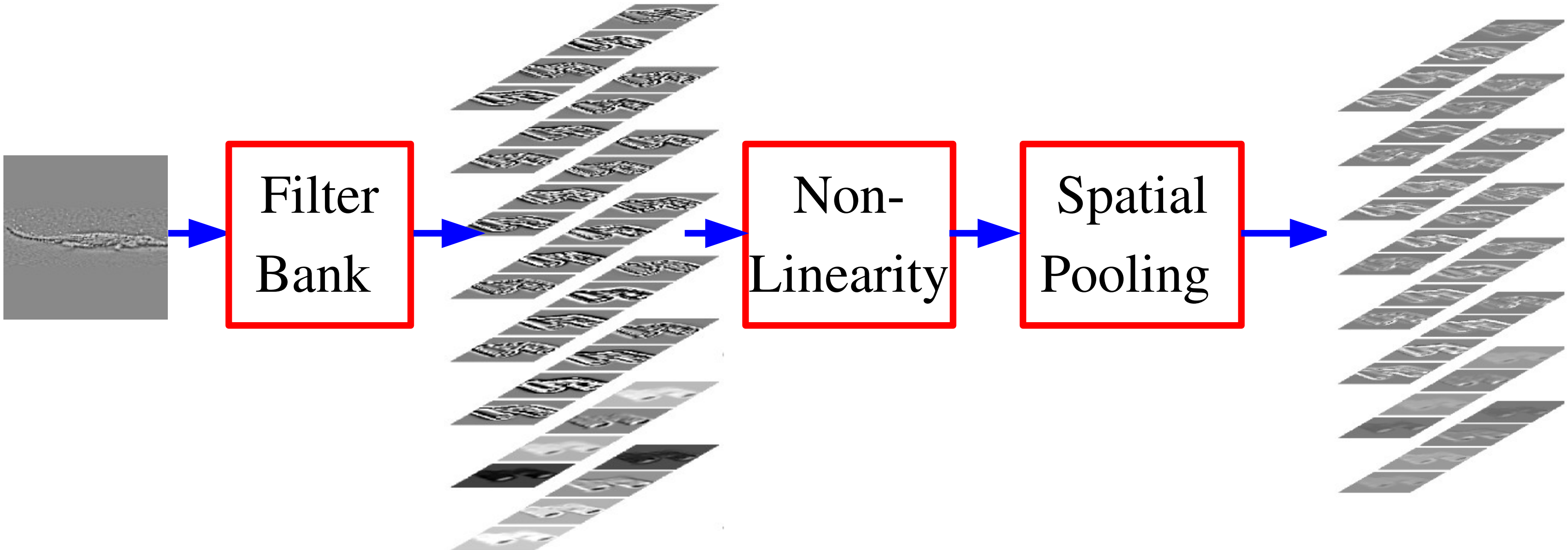
- **Low-level features - mid-level features - high-level features - categories**
- **Representations are increasingly abstract, global, and invariant.**
- **In Vision: part-whole hierarchy**
  - Pixels->Edges->Textons->Parts->Objects->Scenes
- **In Language: hierarchy in syntax and semantics**
  - Words->Parts of Speech->Sentences->Text
  - Objects,Actions,Attributes...-> Phrases -> Statements -> Stories

"Deep" Learning: Learning Hierarchical Representations



- **Deep Learning: learning a hierarchy of internal representations**
- **From low-level features to mid-level invariant representations, to object identities**
- **Representations are increasingly invariant as we go up the layers**
- **using multiple non-linear stages gets around the specificity/invariance dilemma [Mallat 2010]**

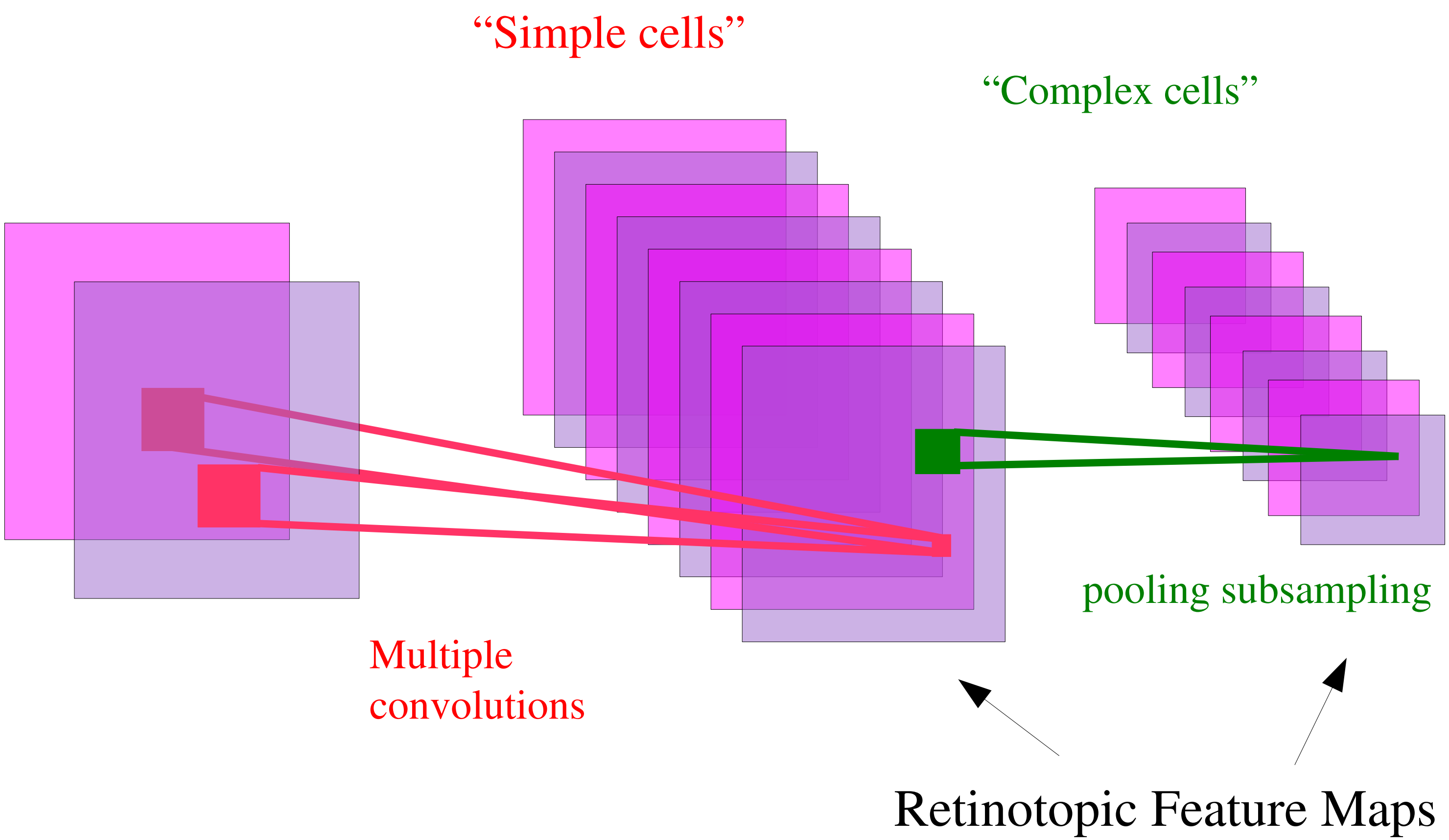
Feature Transform = Filter Bank + Non-Linearity + Pooling



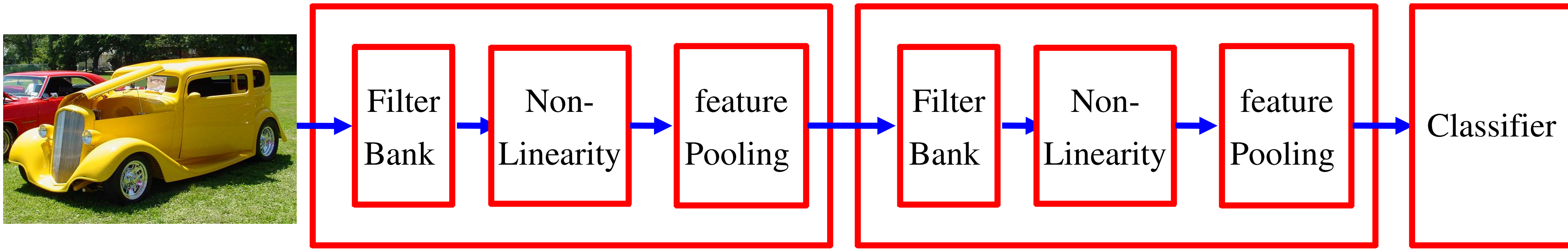
- **Biologically-inspired models of low-level feature extraction**
  - Inspired by [Hubel and Wiesel 1962]
  - Many feature extraction methods are based on this
  - SIFT, GIST, HoG, Convolutional networks.....

An Old Idea for Image Representation with Distortion Invariance

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

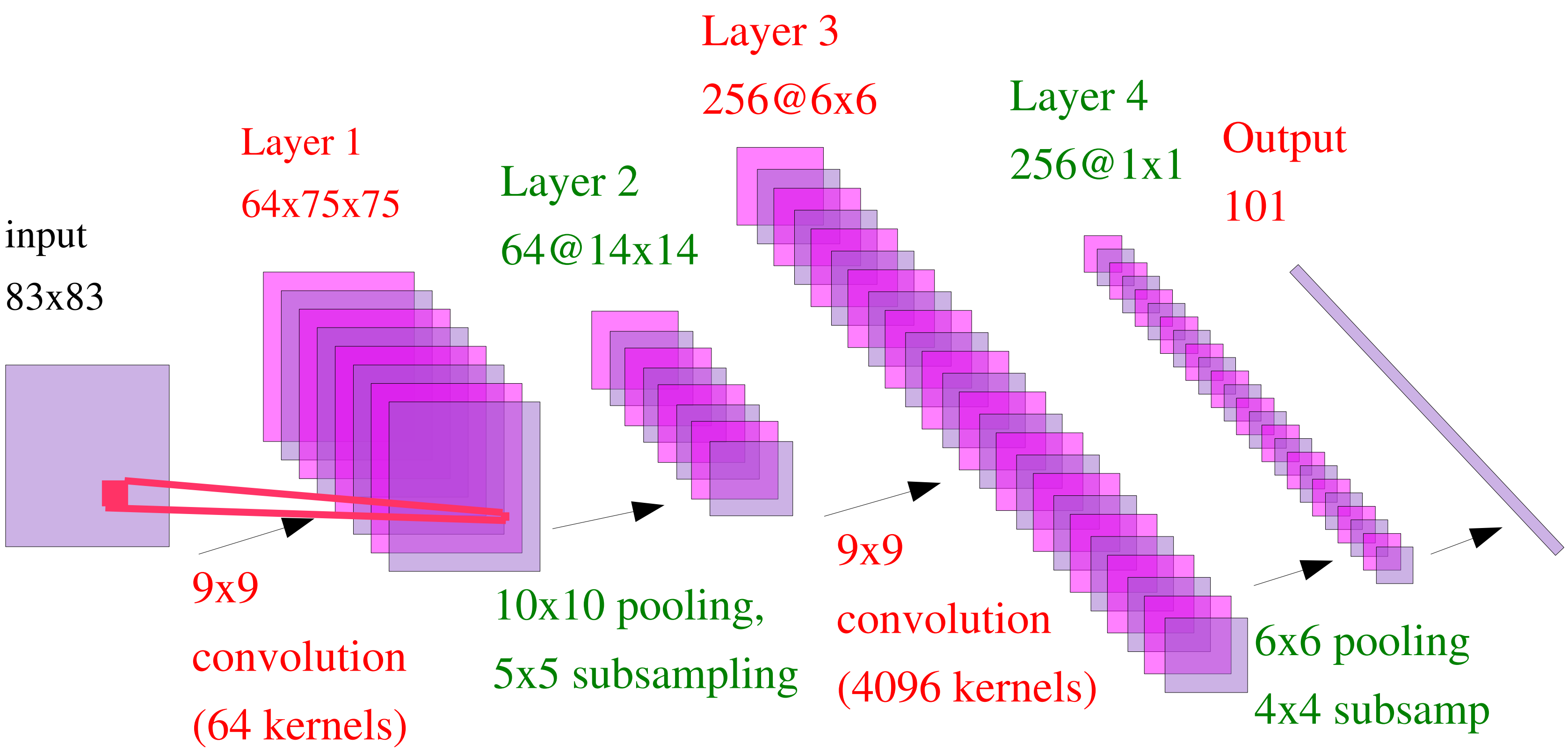


Vision: Multiple Stage of Feature Transform + Classifier



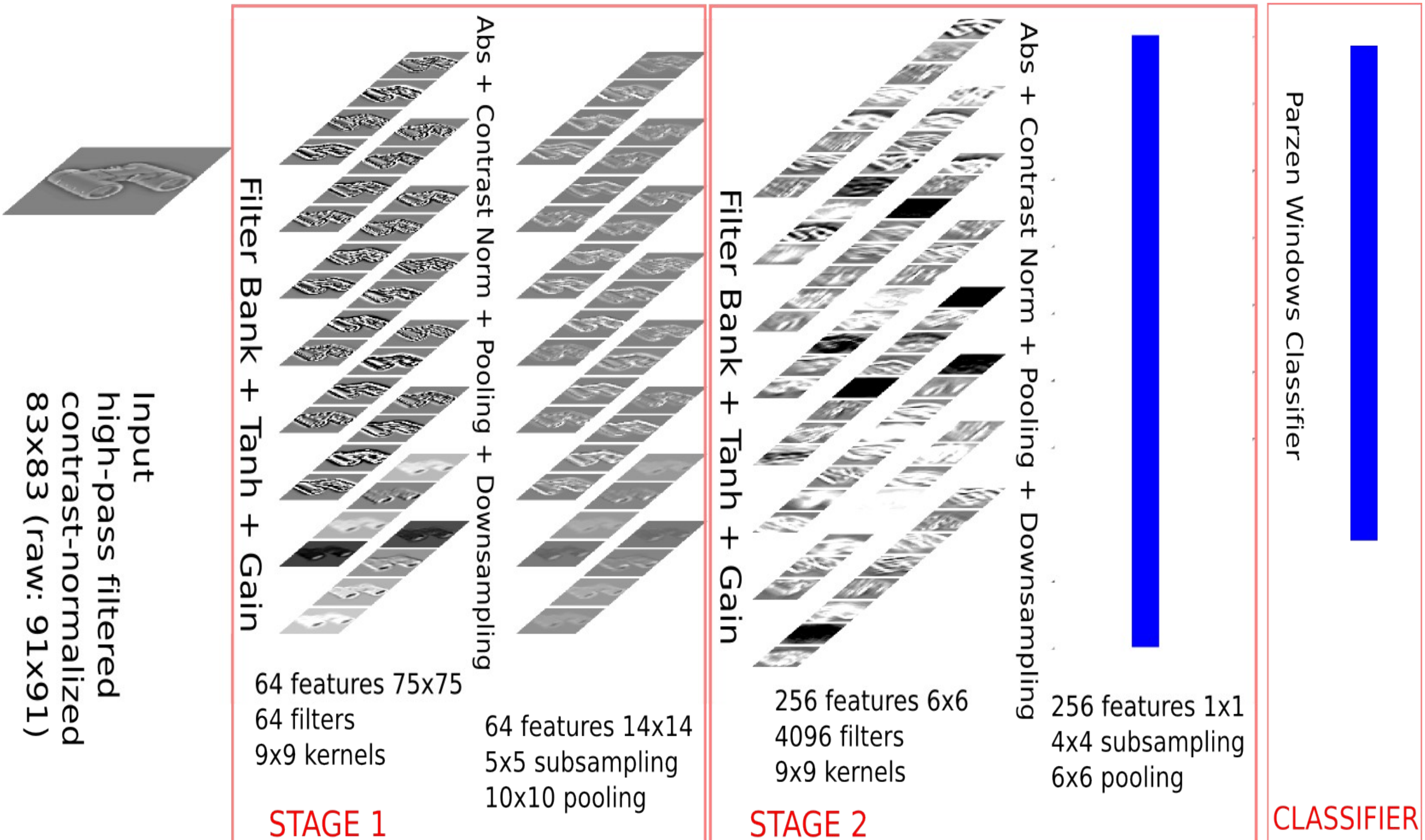
- **Stacking multiple stages of [Filter Bank + Non-Linearity + Pooling].**
- **Learning the filter banks at every layers**
- **Creating a hierarchy of features**
- **Basic elements are inspired by models of the visual cortex**
  - Simple Cell + Complex Cell model of [Hubel and Wiesel 1962]
  - Many “traditional” feature extraction methods are based on this
  - SIFT, GIST, HoG, Convolutional networks.....

Example of Architecture: Convolutional Network (ConvNet)



- **Non-Linearity: tanh, absolute value, shrinkage function, local whitening,..**
- **Pooling: average, max, Lp norm, ...**

Example of Architecture: Convolutional Network (ConvNet)



Training all the filters in a multi-stage ConvNet architecture

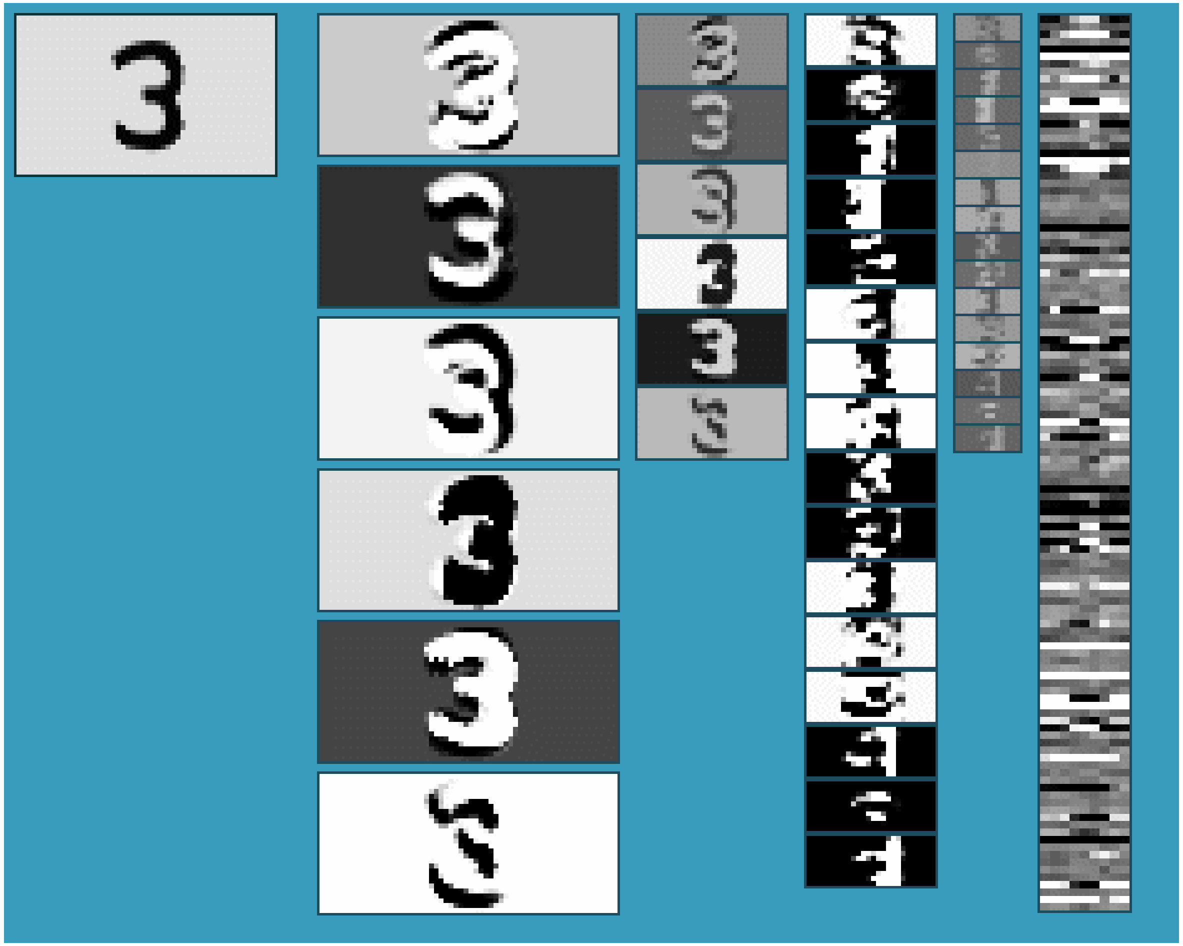
→ [Filter Bank → Non-Linearity → feature Pooling] → [Filter Bank → Non-Linearity → feature Pooling] → Classifier

- End-to-End Supervised Learning by Stochastic Gradient Descent (backprop)
- Layerwise Unsupervised Training with Sparse Coding ("deep learning")
- Supervised Refinement after Unsupervised pre-Training.
- Lots of people work on these architectures: Rob Fergus NYU), Geoff Hinton (Toronto), Andrew Ng (Stanford), David Lowe (UBC), Tommy Poggio (MIT), Larry Carin (Duke), Thomas Serre (Brown), Stéphane Mallat (Polytechnique), Sebastian Seung (MIT),
- Industry: Kai Yu, Ronan Collobert (NEC), T. Dean, J. Weston (Google), C. Garcia (France Telecom), P. Simard (Microsoft) + a number of startups...

Supervised Learning of Convolutional Nets

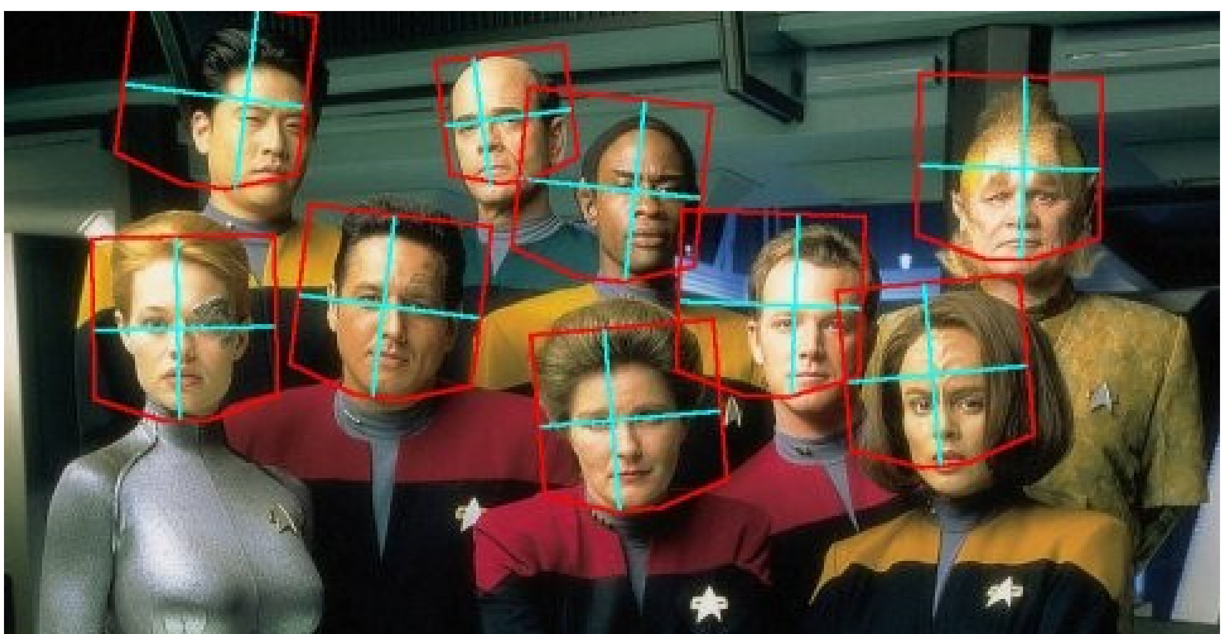
Supervised Learning of ConvNets

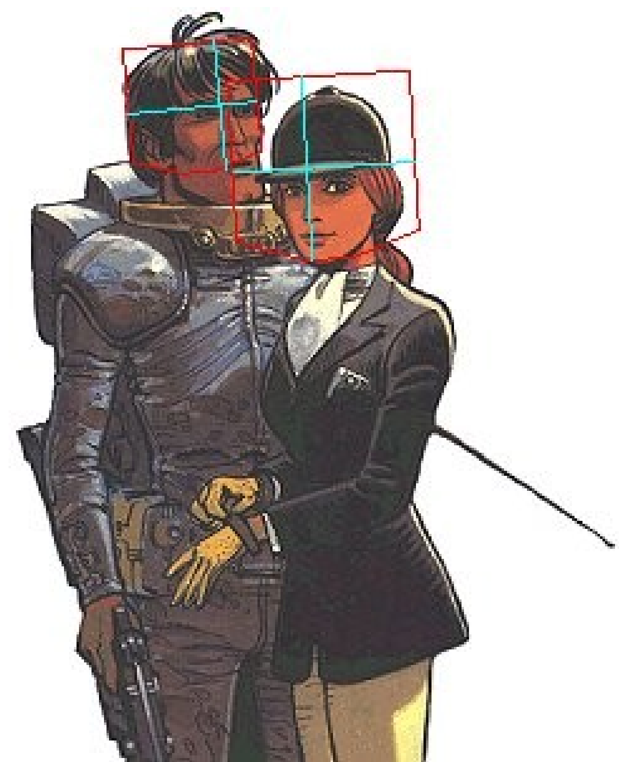
- Stochastic Gradient Descent
- Gradients computed using back-propagation (chain rule)
- Filters are initialized randomly



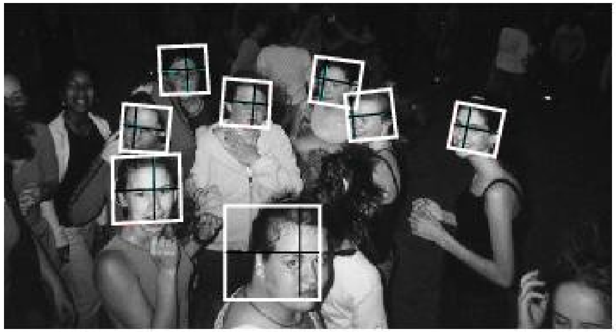
Face Detection: Results

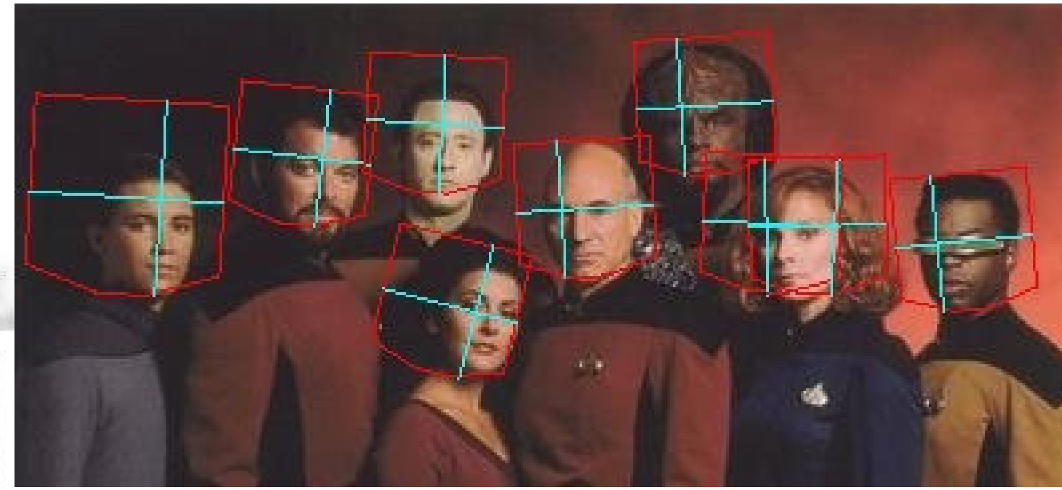
Data Set-> / *False positives per image->*	TILTED 4.42	TILTED 26.9	PROFILE 0.47	PROFILE 3.36	MIT+CMU 0.5	MIT+CMU 1.28
Our Detector	90%	97%	67%	83%	83%	88%
Jones & Viola (tilted)	90%	95%	x		x	
Jones & Viola (profile)	x		70%	83%	x	

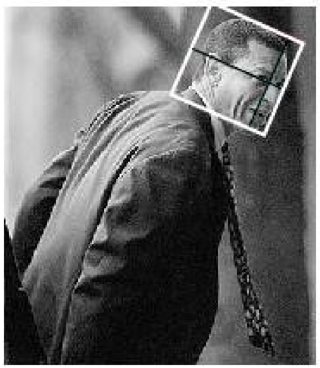


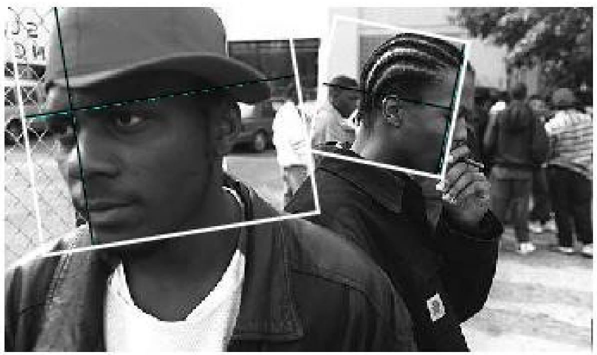


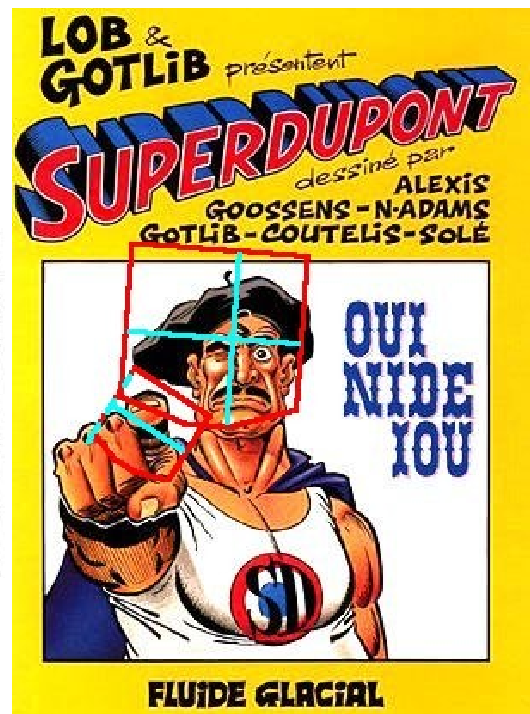
Face Detection and Pose Estimation: Results

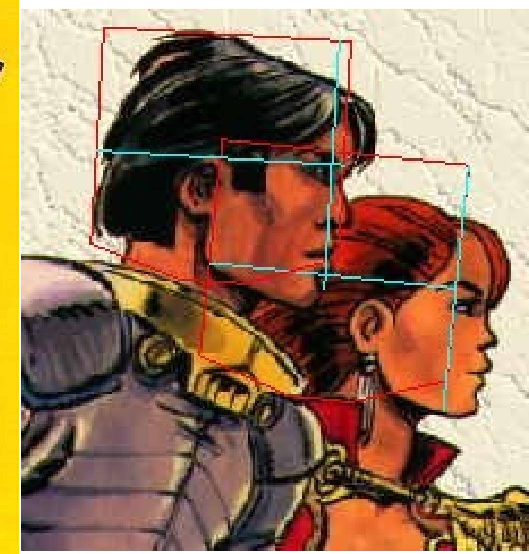


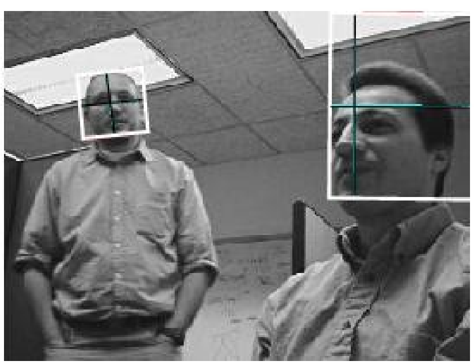


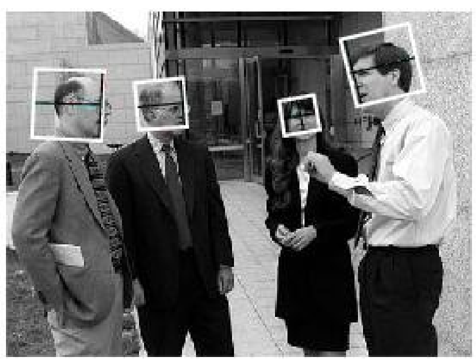




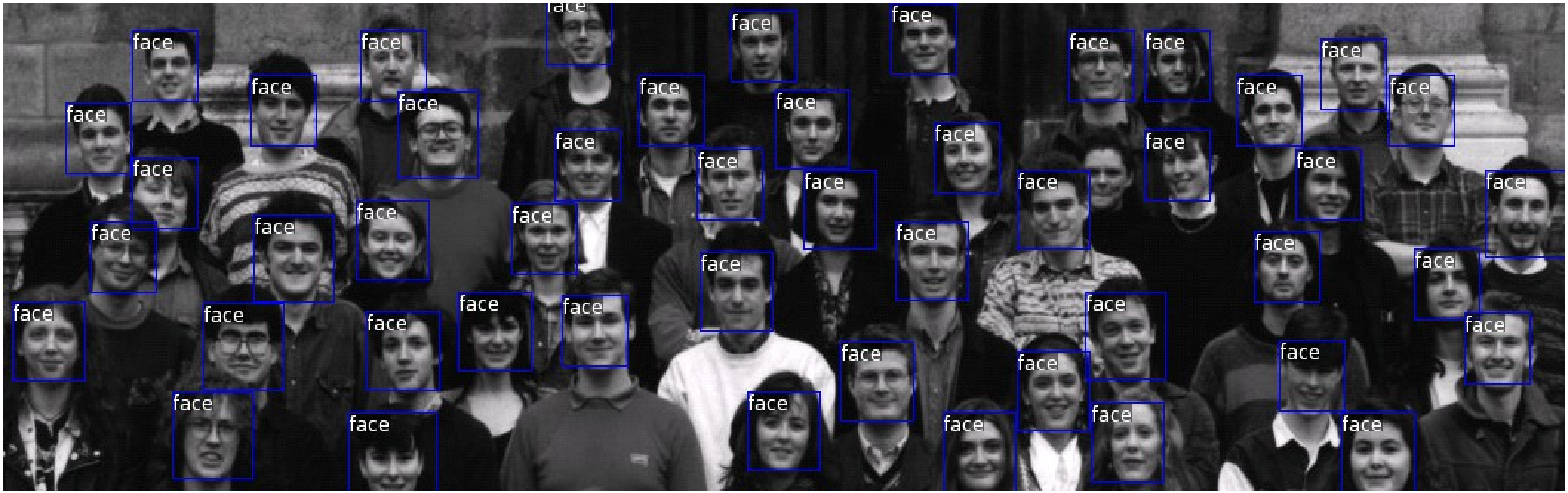








Face Detection with a ConvNet

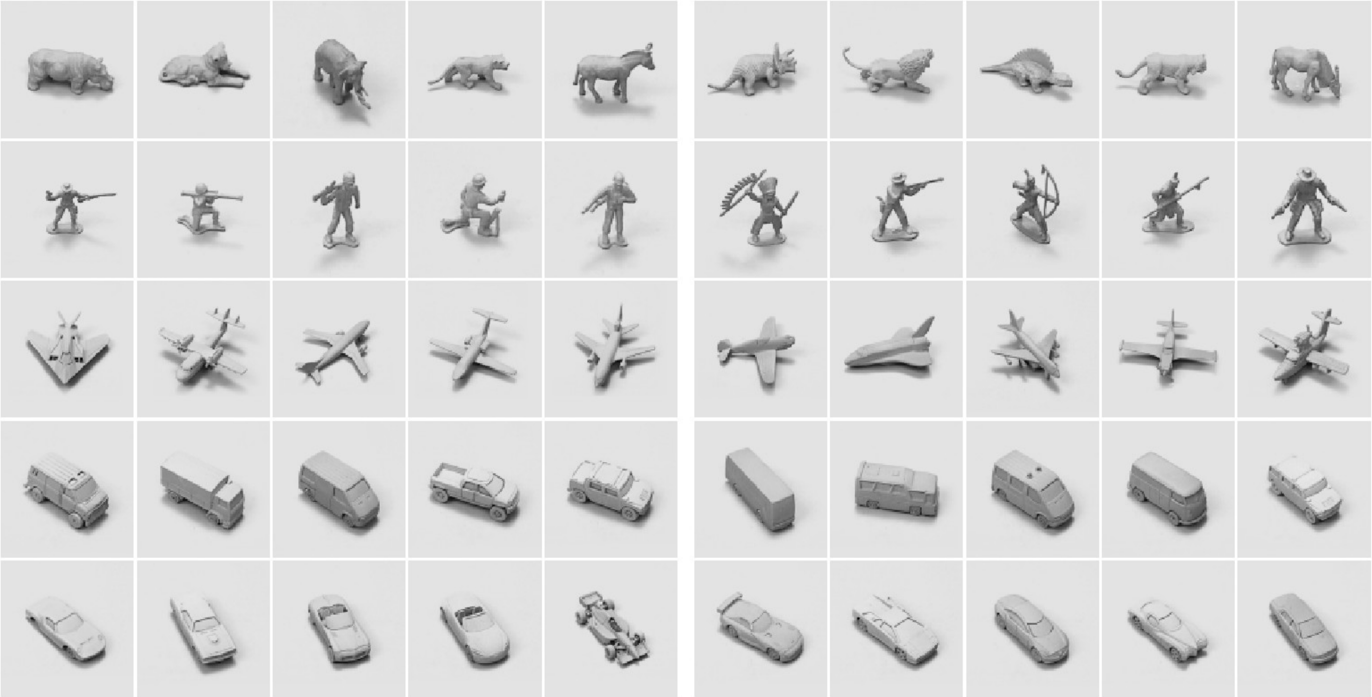


- **Demo produced with EBLearn open source package**
- **http://eblearn.sf.net**

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

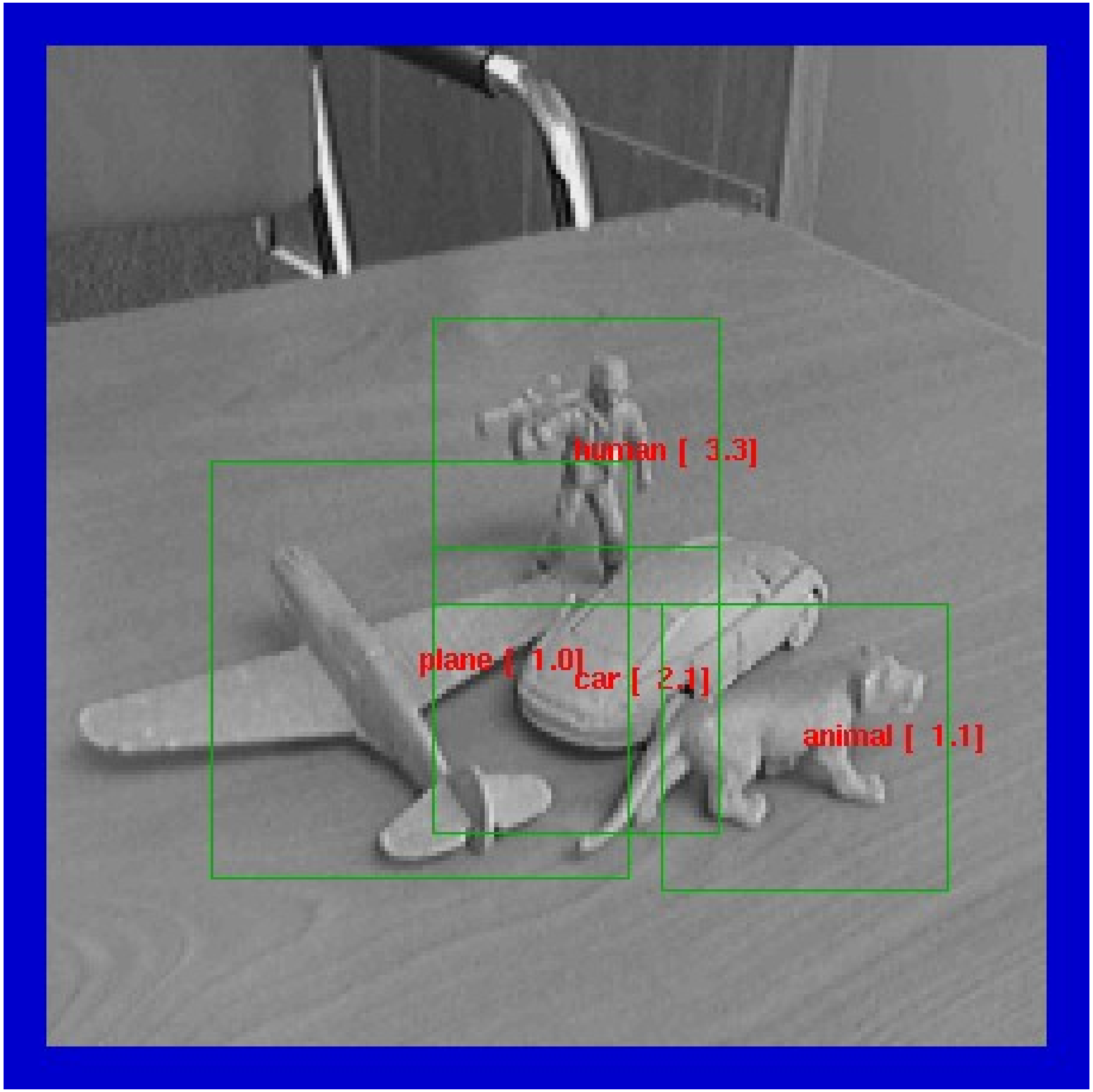


Training instances **Test instances**

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

Examples (Monocular Mode)



Road Sign Recognition Competition

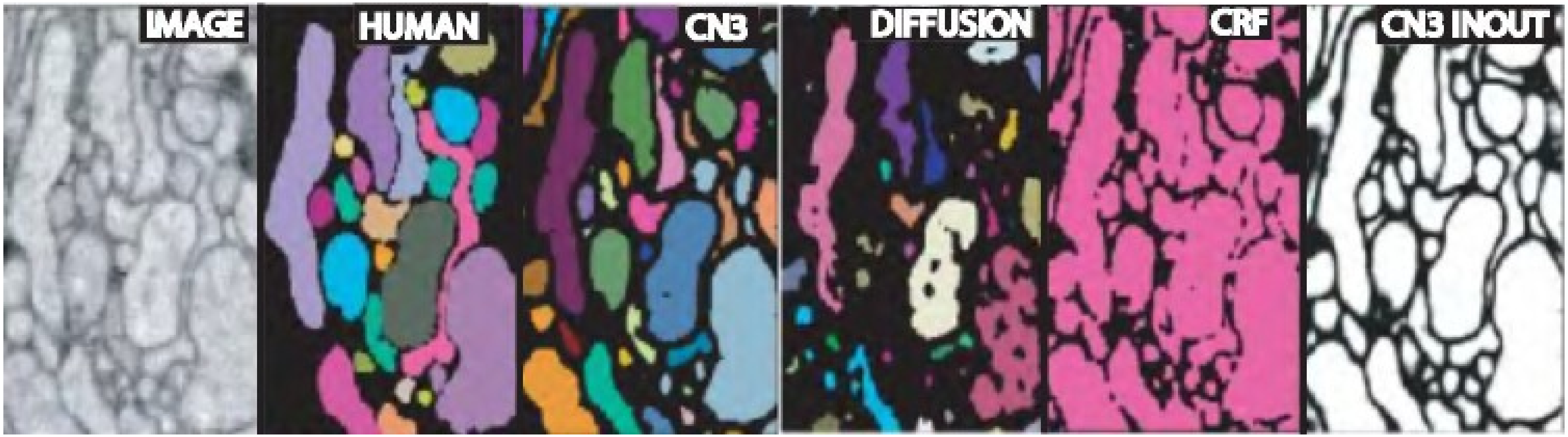
GTSRB Road Sign Recognition Competition (phase 1)

- 32x32 images
- The 13 of the top 14 entries are ConvNets, 6 from NYU, 7 from IDSIA
- No 6 is humans!

#	Team	Method	Accuracy
	sermanet	**EBLearn 2LConvNet ms 108 feats + 100-feats CF classifier + No color**	**99.17%**
197	IDSIA	cnn_hog3	98.98%
196	IDSIA	cnn_cnn_hog3	98.98%
178	**sermanet**	**EBLearn 2LConvNet ms 108 feats**	**98.97%**
195	IDSIA	cnn_cnn_hog3_haar	98.97%
187	**sermanet**	**EBLearn 2LConvNet ms 108 + val**	**98.89%**
199	INI-RTCV	Human performance	98.81%
170	IDSIA	CNN(IMG)_MLP(HOG3)	98.79%

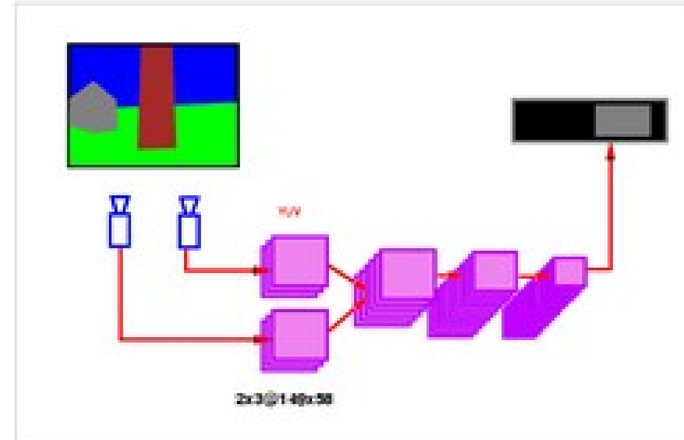
Convolutional Nets For Brain Imaging and Biology

- **Brain tissue reconstruction from slice images [Jain,....,Denk, Seung 2007]**
  - Sebastian Seung's lab at MIT.
  - 3D convolutional net for **image segmentation**
  - ConvNets Outperform MRF, Conditional Random Fields, Mean Shift, Diffusion,...[ICCV'07]

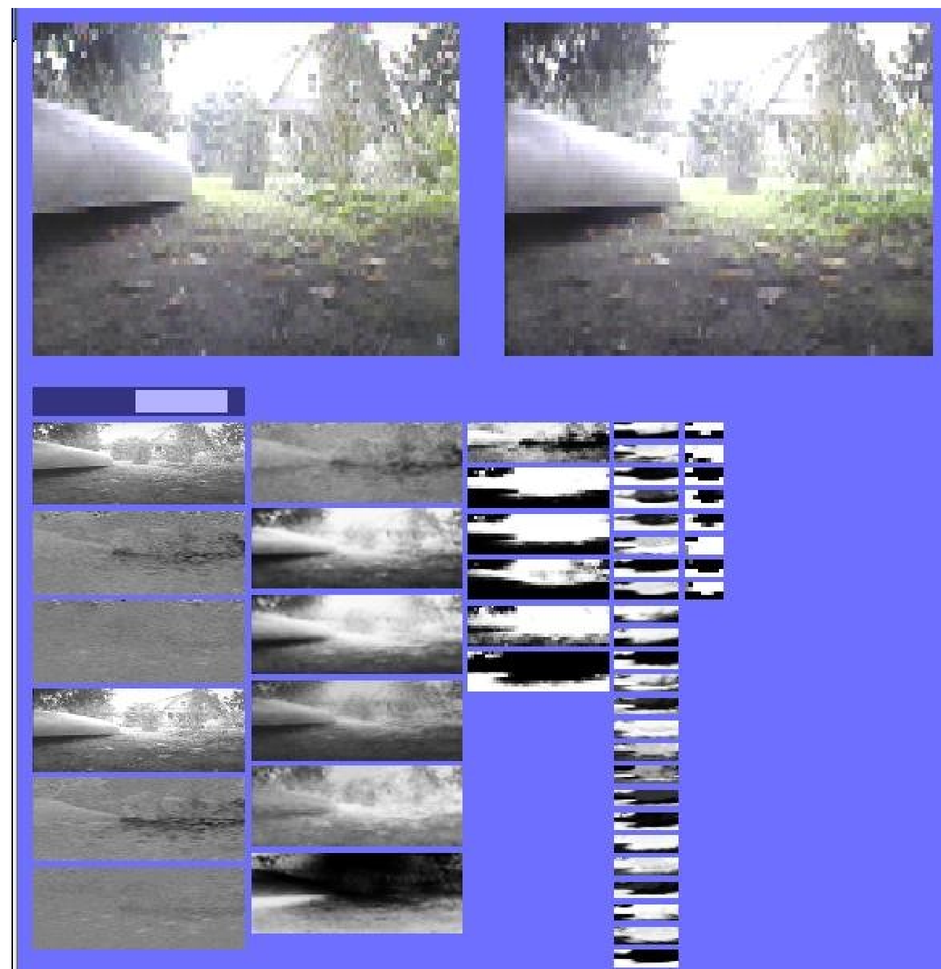


Visual Navigation for a Mobile Robot

[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



Industrial Applications of ConvNets

- **AT&T/Lucent/NCR**
  - Check reading, OCR, handwriting recognition (deployed 1996)
- **Vident Inc**
  - Vident Inc's "SmartCatch" system deployed in several airports and facilities around the US for detecting intrusions, tailgating, and abandoned objects (Vident is a spin-off of NEC)
- **NEC Labs**
  - Cancer cell detection, automotive applications, kiosks
- **Google**
  - Face and license plate removal from StreetView
- **Microsoft**
  - OCR, handwriting recognition, speech detection
- **France Telecom**
  - Face detection, HCI, cell phone-based applications
- **Other projects: HRL (3D vision)....**

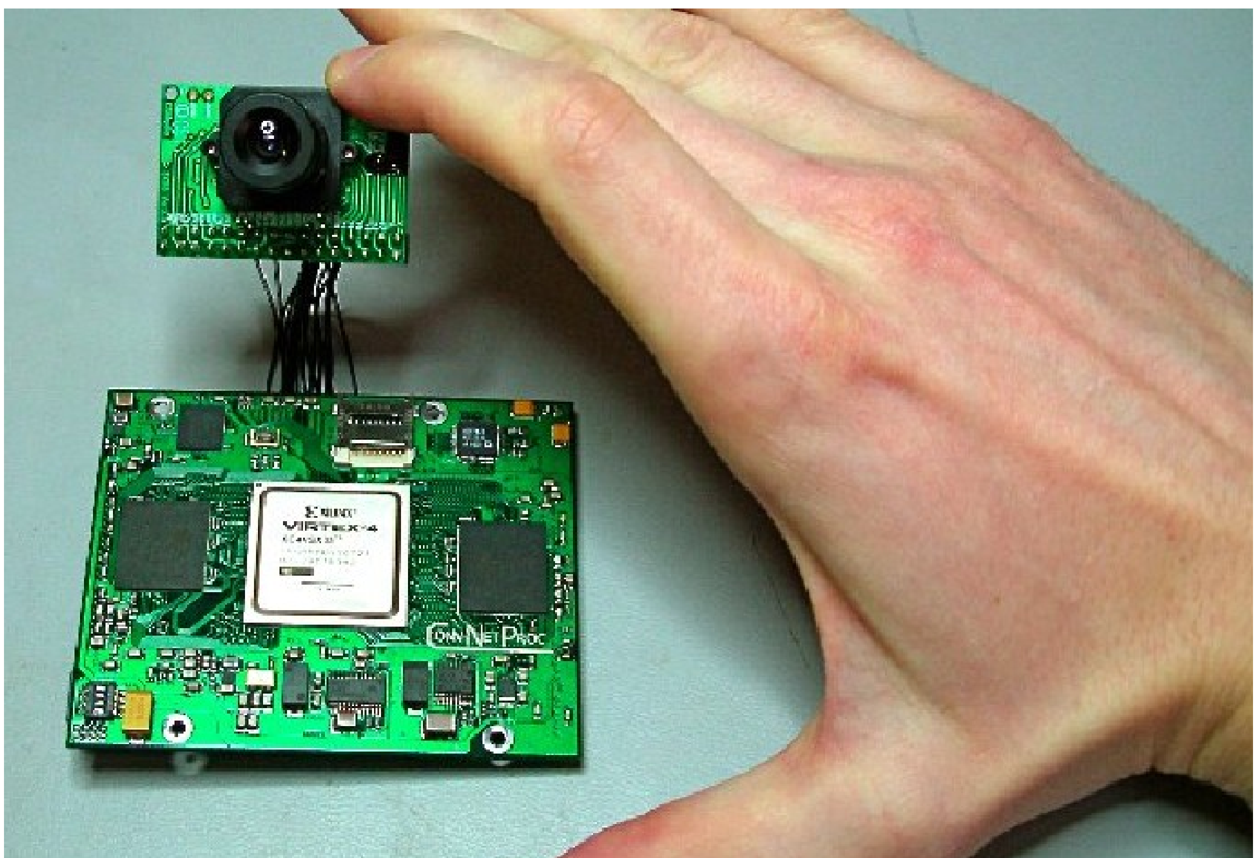
Embedded Hardware for Fast ConvNet

NeuFlow: a Dataflow Computer for Embedded Vision

- **High Peak Performance**
  - Current proven implementation: 92 GOP/sec on a Xilinx Virtex 6
  - Beta version: 200 GOP/sec on a Xilinx Virtex 6
  - Simulated version: 700 GOP/sec on an IBM 45nm process
- **High Actual Performance with a Custom Dataflow Compiler**
  - Takes a high-level description of a processing flow as an input (any typical image transform, as found in EbLearn/GbLearn/Torch)
  - Performs a multi-step analysis and generates optimized bytecode for NeuFlow, by minimizing memory bandwidth usage
  - A typical ConvNet is computed at an average of 80 to 90% (end-to-end) of the peak perf (GPU code rarely goes beyond 20/30%)

FPGA Custom Board: NYU ConvNet Processor

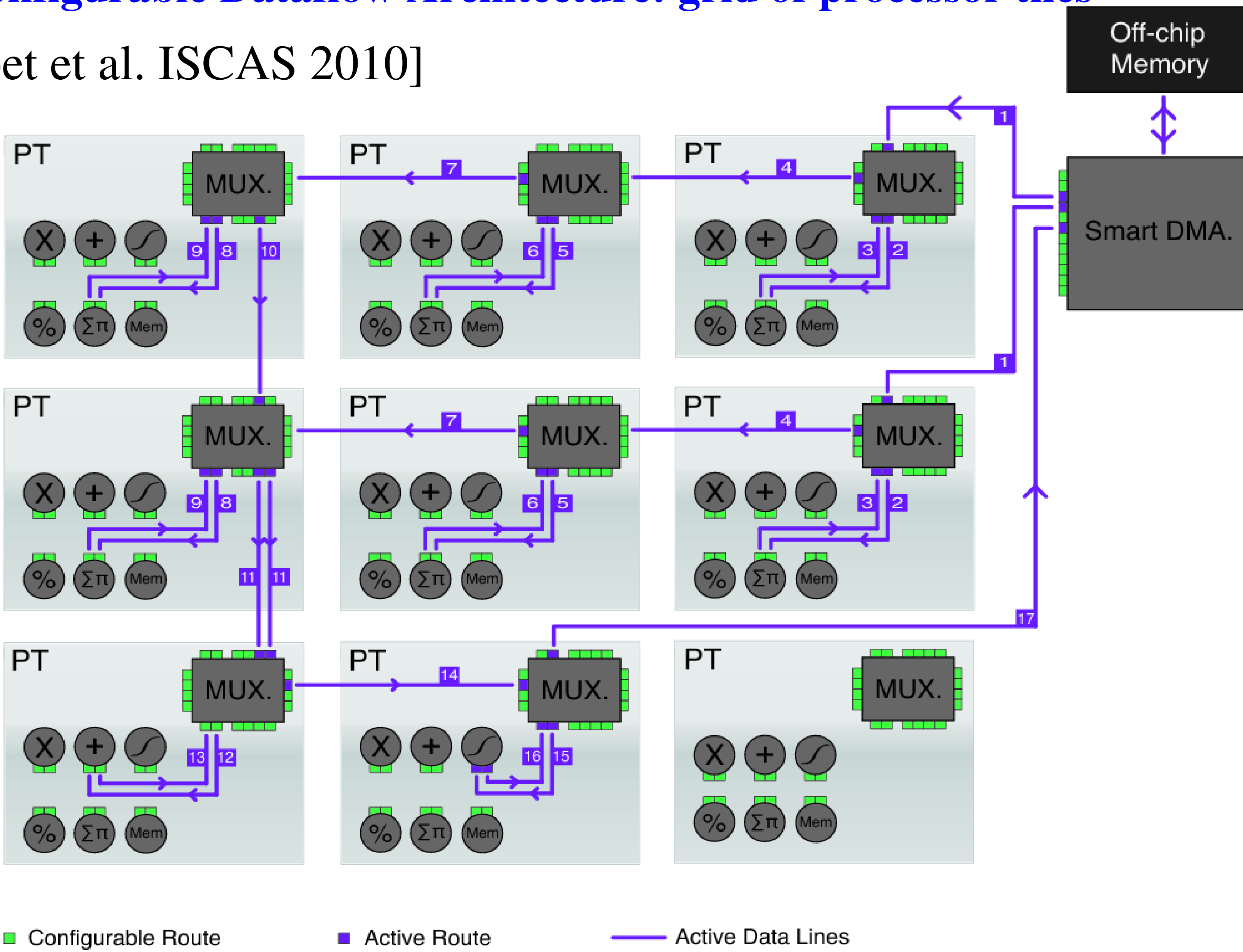
- **Xilinx Virtex 4 FPGA, 8x5 cm board** [Farabet et al. ISCAS 2009]
  - Dual camera port, Fast dual QDR RAM,
- **New version developed in collaboration with Eugenio Culurciello (Yale)**
  - Version for **Virtex 6 FPGA** development board **(operational!)**



NeuFlow: Dataflow architecture for ConvNet/Vision

- **Reconfigurable Dataflow Architecture: grid of processor tiles**

[Farabet et al. ISCAS 2010]



xFlow & LuaFlow: Language/Compiler for NeuFlow

- **Algorithm described as a graph of computing nodes (dataflow graph)**
- **Compiler generates instructions for NeuFlow processor (or CUDA)**

flow-graph model

↓ *graph parsing = {node reordering, node merging}*

memory map &
sequence of transforms

↓ *device mapping = {static scheduling, reconfiguration sequencing}*

sequence of reconfigurations
& memory transfers

↓ *compilation = {machine code generation}*

binary code

LuaFlow program example

- **16 convolutions, 9x9 kernels - > tanh - > fully-connected layer**

- initializing neuFLow:

```
neuFlow = NeuFlow{mode='runtime'}
```

- describing a neural net:

```
input_host = torch.Tensor(100,100)
net = nn.Sequential()
net:add(nn.SpatialConvolution(1,16,9,9))
net:add(nn.Tanh())
net:add(nn.SpatialLinear(16,4))
```

- elaborating the code for neuFlow:

```
neuFlow:beginLoop('main')
    input_nf = neuFlow:copyFromHost(input_host)
    output_nf = neuFlow:compile(net, input_nf)
    output_host = neuFlow:copyToHost(output_nf)
neuFlow:endLoop('main')
```

LuaFlow program example

16 convolutions, 9x9 kernels - > tanh - > fully-connected layer

- loading the bytecode on neuFlow:

```
neuFlow:loadBytecode()
-- at this point, neuFlow executes its new code
```

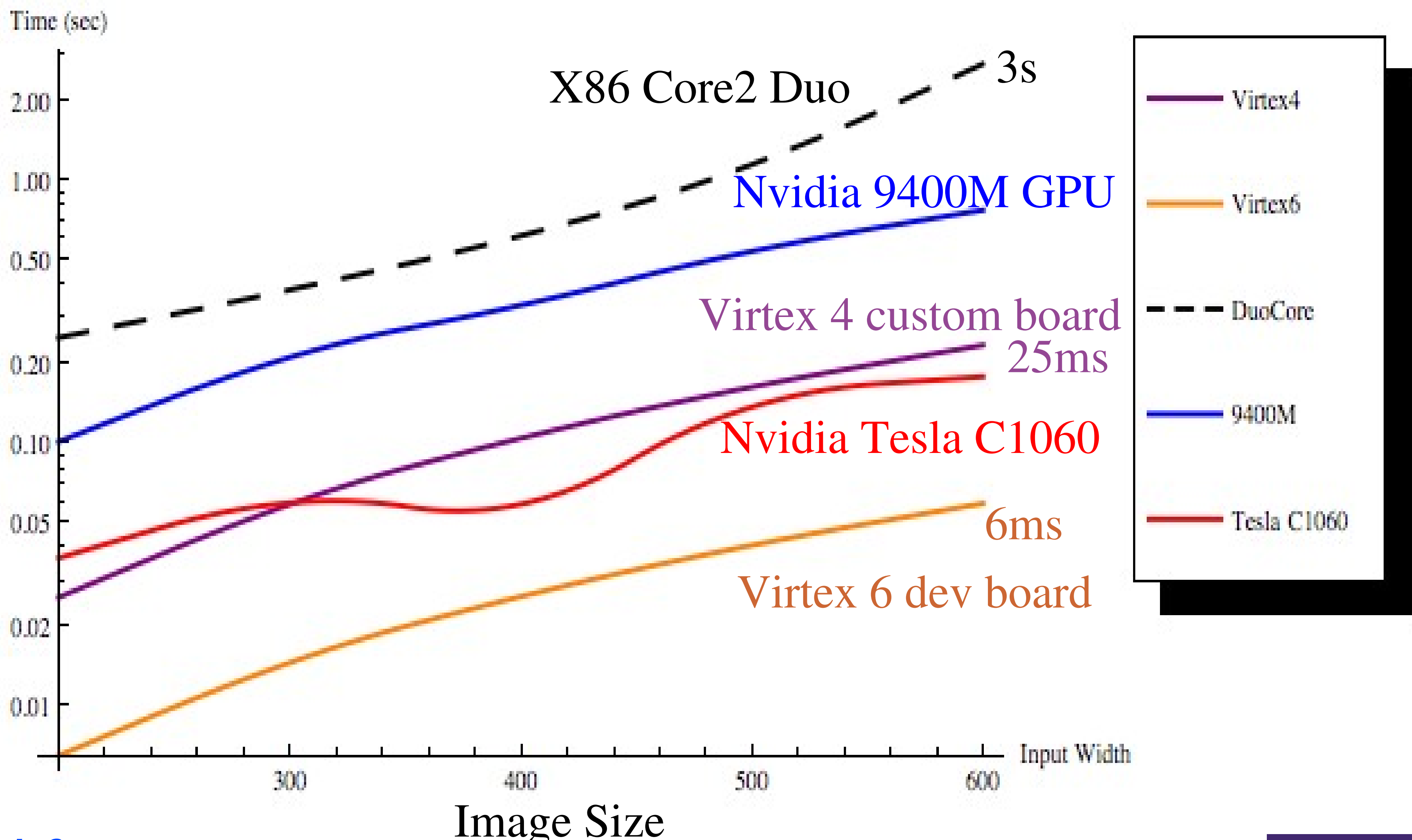
- now simply describe the host code:

```
while true do
     input_host = camera:getFrame()
     neuFlow:copyToDev(input_host)
     neuFlow:copyFromDev(output_host)
     result = soft_classifier:forward(output_host)
end
```

- at this point the code is running in a loop, neuFlow is computing the neural net, while the host computes a simple linear classifier on the results

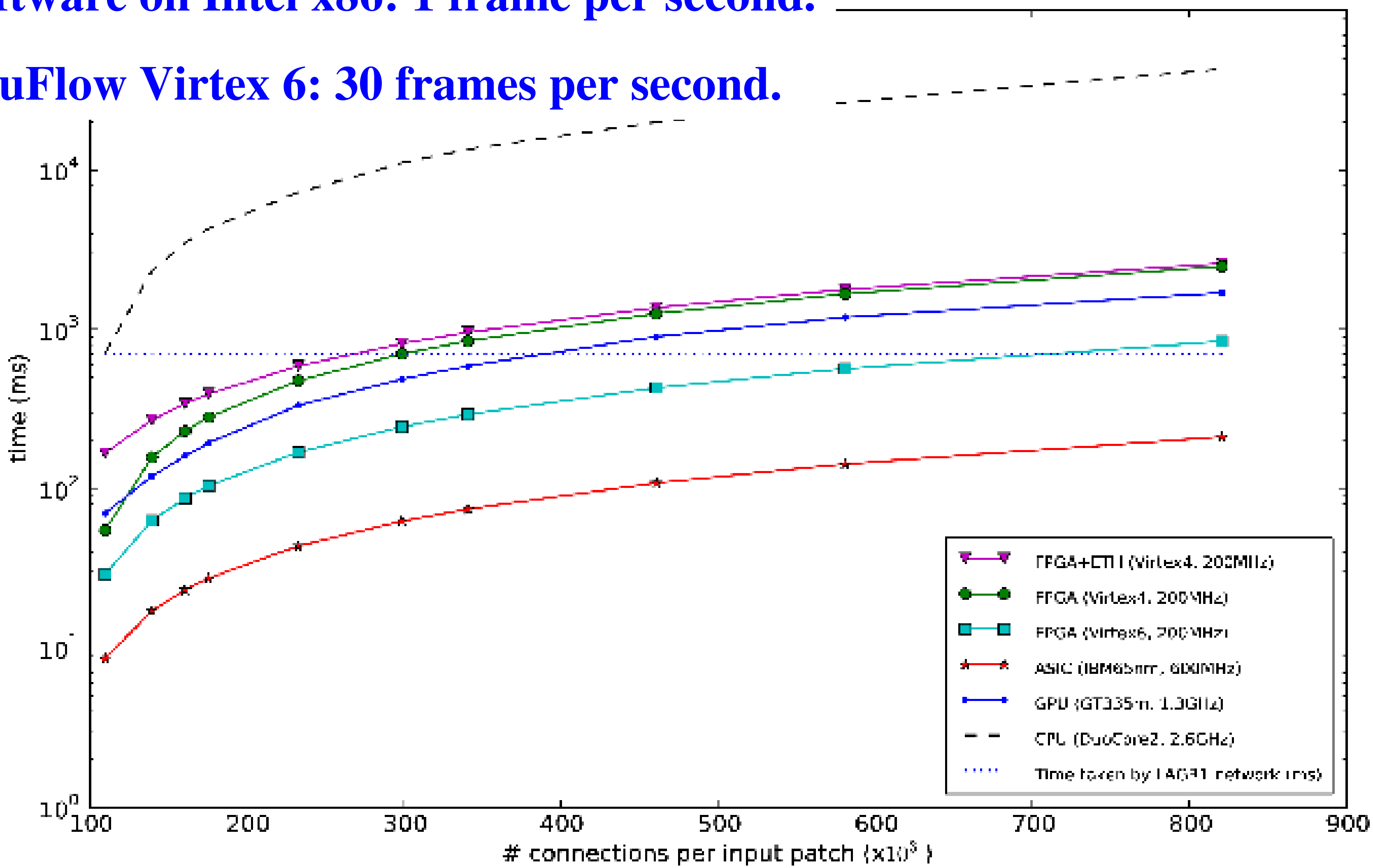
FPGA Performance

- **Seconds per frame for a robot vision task (log scale)** [Farabet et al. 2010]



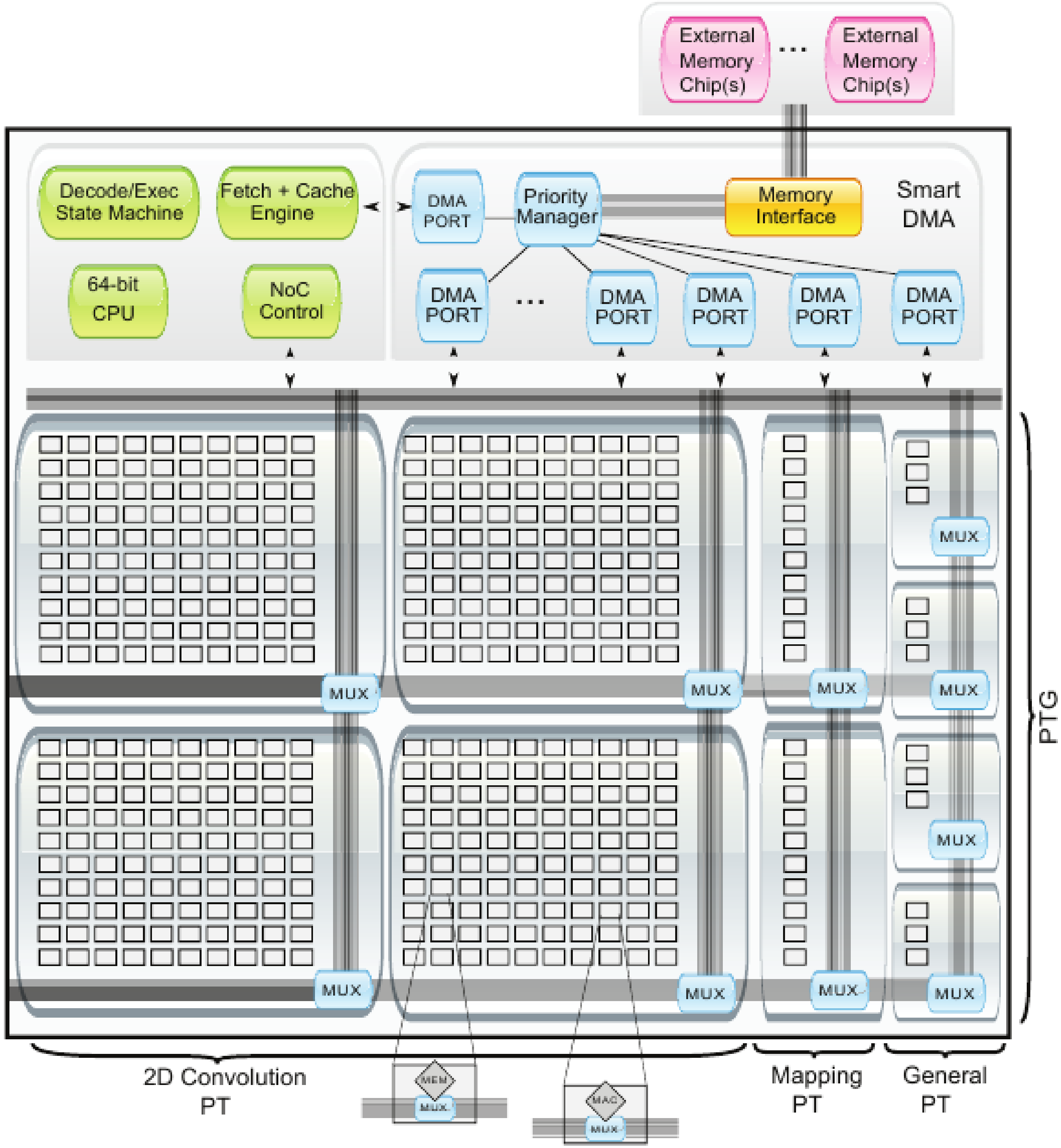
NeuFlow: Performance with the LAGR ConvNet

- Example: a typical ConvNet trained for obstacle detection (LAGR)
- Software on Intel x86: 1 frame per second.
- NeuFlow Virtex 6: 30 frames per second.



NeuFlow ASIC

- Collaboration with
  - e-Lab (Yale)
- Design in progress
- 45 nm technology
- 700 Gop/s
- < 3 Watts.
- 5x3 mm



NeuFlow: Performance Comparison

- **Example: a typical ConvNet**

	CPU Intel 2 cores	GPU NVidia GT335m	GPU NVidia GS1070	FPGA NeuFlow Virtex-4	FGPA NeuFlow Virtex-6	ASIC NeuFlow IBM 45nm
Peak Gop/sec	10	182	1000	40	160	700
Actual Gop/sec	1.1	54	290	37	147 99	600
FPS	1.4	67	360	46	182	700
Power (W)	30	30	220	10	15	3
Gop/s/W	0.037	1.8	1.32	3.7	9.8	200

"The Economist", October 21, 2010

The Economist

Computer vision

Eye robot

Poor eyesight remains one of the main obstacles to letting robots loose among humans. But it is improving, in part by aping natural vision

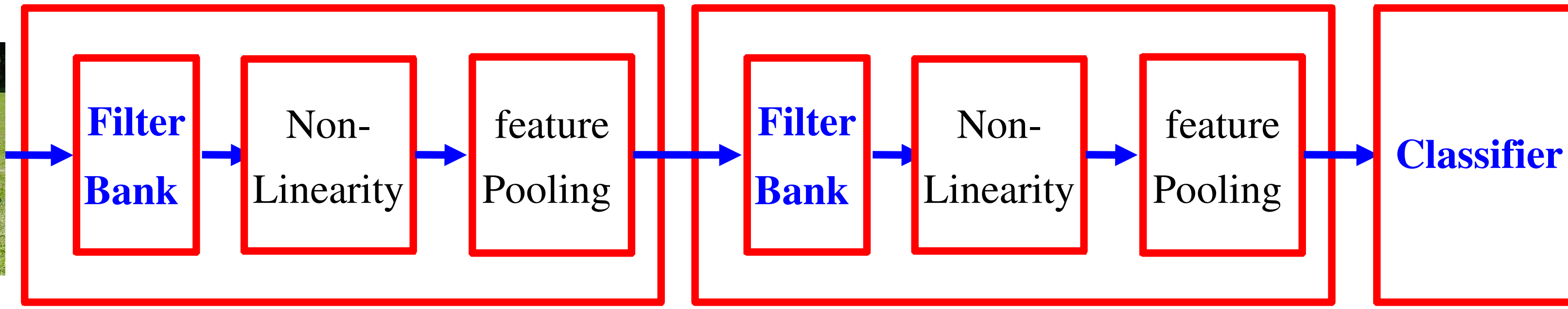
Oct 21st 2010

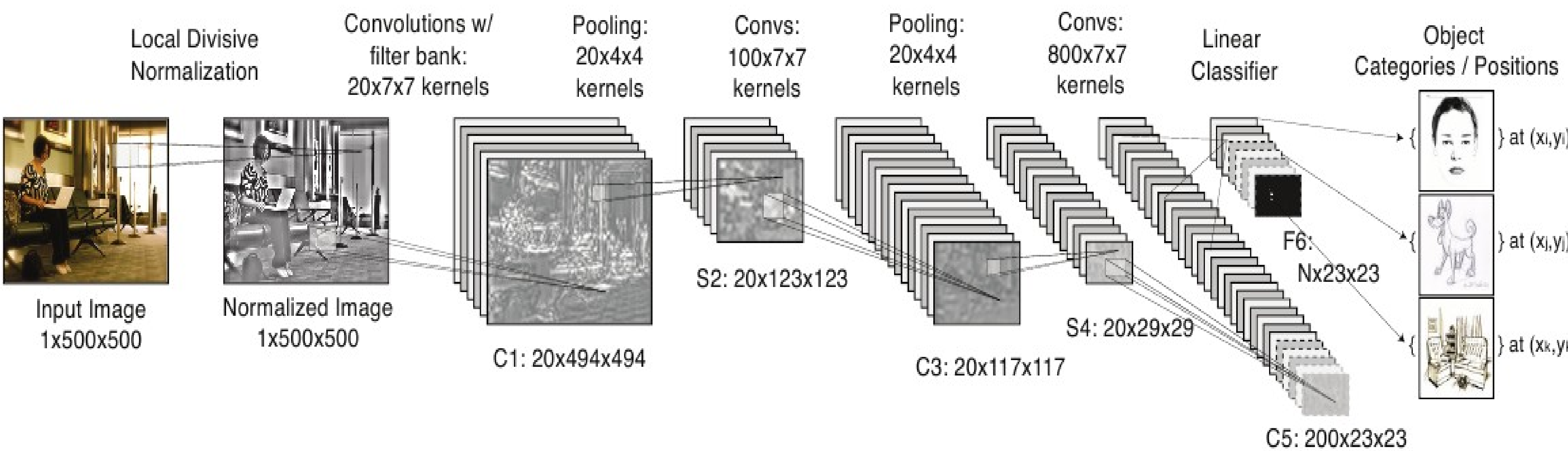
A ConvNet begins by swiping a number of software filters, each several pixels across, over the image, pixel by pixel. Like the brain's primary visual cortex, these filters look for simple features such as edges. The upshot is a set of feature maps, one for each filter, showing which patches of the original image contain the sought-after element. A series of transformations is then performed on each map in order to enhance it and improve the contrast. Next, the maps are swiped again, but this time rather than stopping at each pixel, the filter takes a snapshot every few pixels. That produces a new set of maps of lower resolution. These highlight the salient features while reining in computing power. The whole process is then repeated, with several hundred filters probing for more elaborate shapes rather than just a few scouring for simple ones. The resulting array of feature maps is run through one final set of filters. These classify objects into general categories, such as pedestrians or cars.

Many state-of-the-art computer-vision systems work along similar lines. The uniqueness of ConvNets lies in where they get their filters. Traditionally, these were simply plugged in one by one, in a laborious manual process that required an expert human eye to tell the machine what features to look for, in future, at each level. That made systems which relied on them good at spotting narrow classes of objects but inept at discerning anything else.

Dr LeCun's artificial visual cortex, by contrast, lights on the appropriate filters automatically as it is taught to distinguish the different types of object. When an

Object Recognition with ConvNets

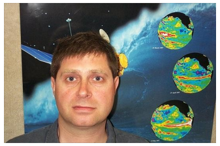




Problem: supervised ConvNets don't work with few labeled samples

- **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

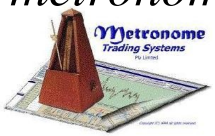
wild cat

lotus

ant

dollar

metronome



w. chair

minaret

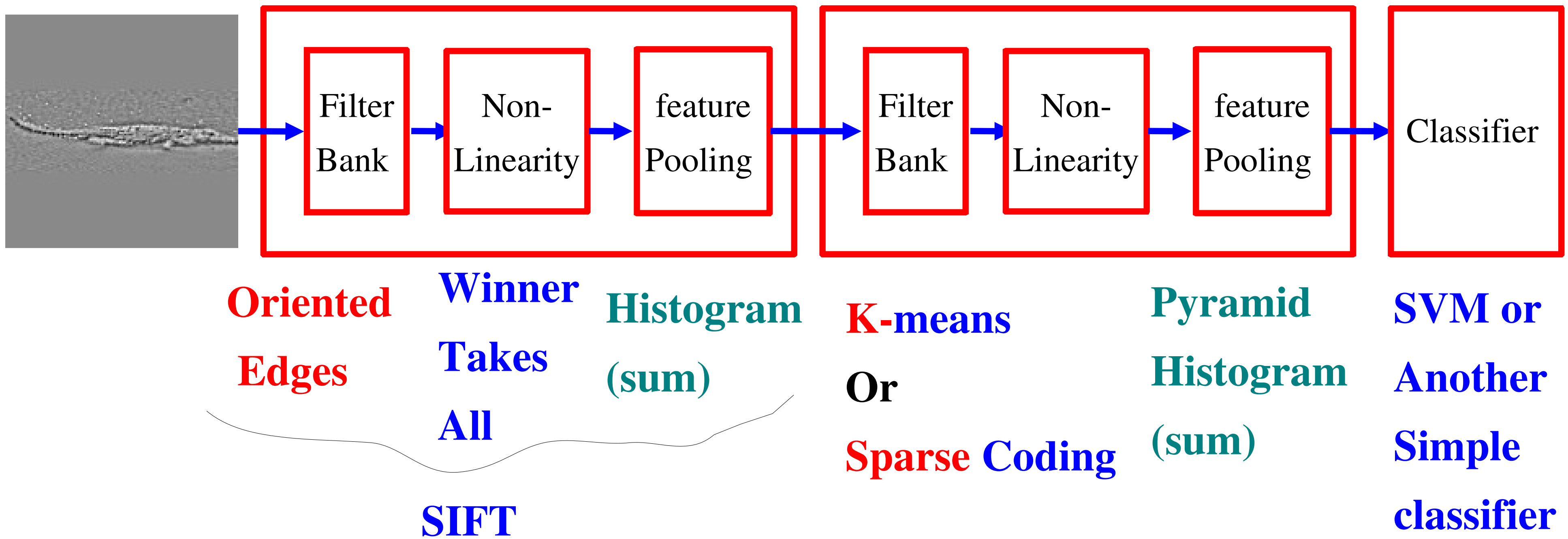
cellphone

joshua t.

cougar body

background

"Conventional" vision systems are similar to ConvNets



- **Fixed low-level Features + unsupervised mid-level features + simple classifier**
- **Example:**
  - SIFT + K-means + Pyramid pooling + SVM intersection kernel: **>65%**
    - [Lazebnik et al. CVPR 2006]
  - SIFT + Sparse coding + Pyramid pooling + SVM: **>73%**
    - [Yang et al. CVPR 2009, Boureau et al. CVPR 2010]

Unsupervised Deep Learning: Leveraging Unlabeled Data

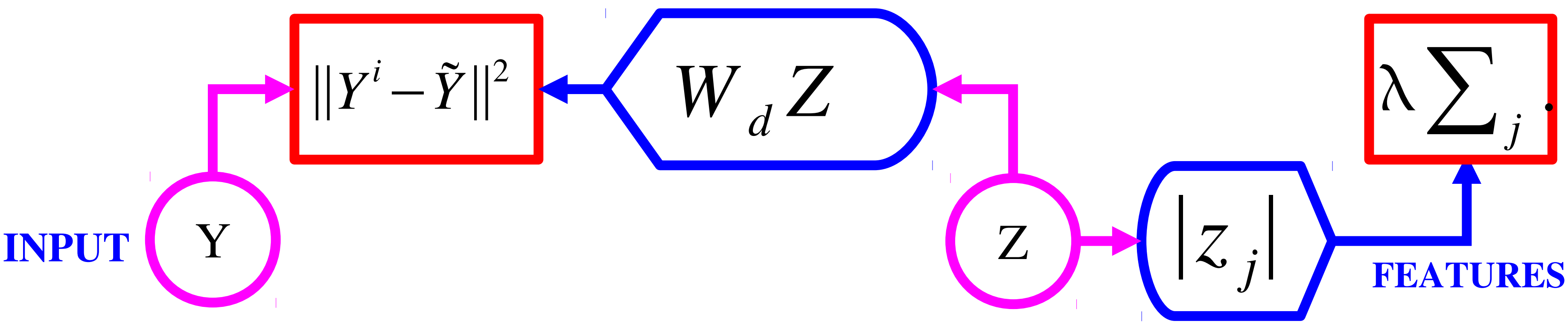
[Hinton 05, Bengio 06, LeCun 06, Ng 07]

- **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**

Unsupervised Feature Learning with Sparse Coding

[Olshausen & field 1997]

- **Find a dictionary of basis functions such that any input can be reconstructed of a sparse linear combination of them.**



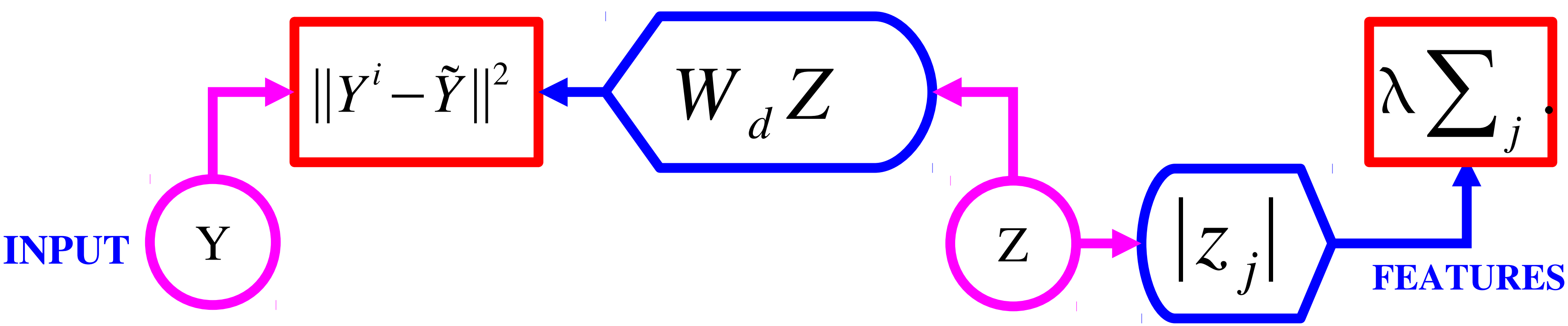
- **Energy:** $$E(Y^i, Z; W_d) = \|Y^i - W_d Z\|^2 + \lambda \sum_j |z_j|$$
- **Optimal Code** $$Z^i = argmin_z E(Y^i, z; W_d)$$
- **Free Energy:** $$F(Y^i; W_d) = F(Z^i) = min_z E(Y^i, z; W_d)$$

Unsupervised Feature Learning with Sparse Coding

- **The learning algorithm minimizes the loss function:**

$$L(W_d) = \sum_i F(Y^i; W_d) = \sum_i (\min_Z E(Y^i, Z; W_d))$$

- **The columns of Wd are normalized**



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

- **Free Energy:** $$F(Y^i; W_d) = F(Z^i) = \min_z E(Y^i, z; W_d)$$

Problem with Sparse Coding: Inference is slow

- **Inference: find Z that minimizes the energy for a given Y**

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

$$Z^i = \mathrm{argmin}_z\, E(Y^i, z; W_d)$$

- **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

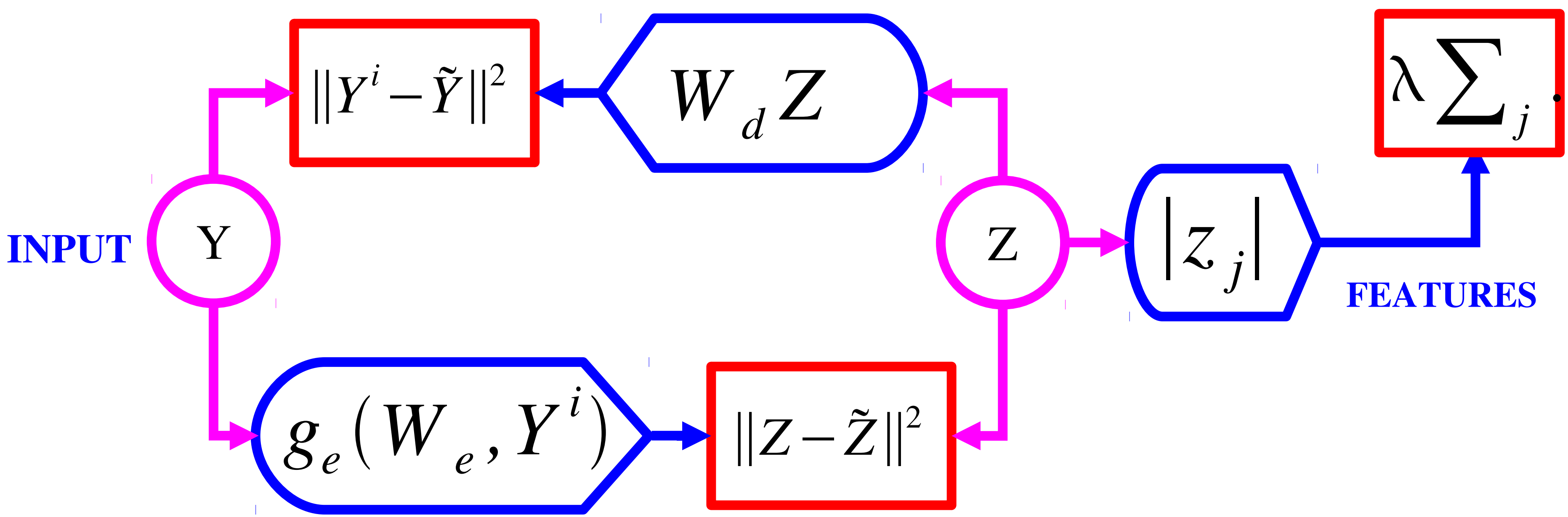
Solution: Predictive Sparse Decomposition (PSD)

[Kavukcuoglu, Ranzato, LeCun, 2009]

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

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

$$g_e(W_e, Y^i) = D \tanh(W_e Y)$$

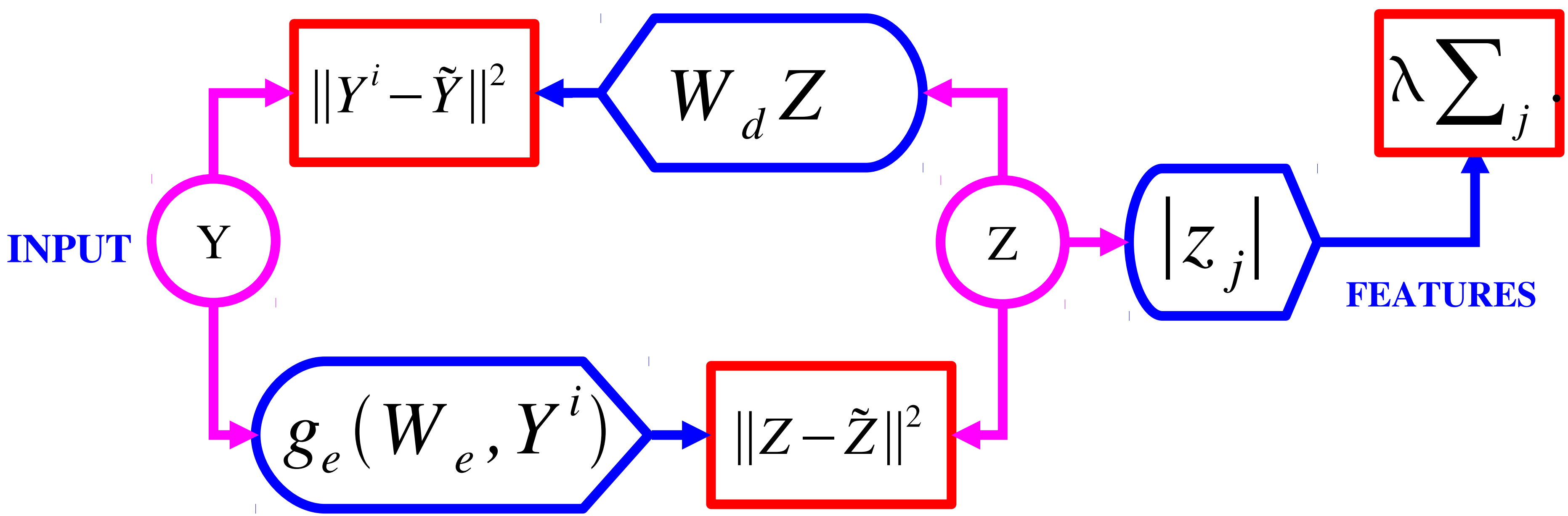


PSD: Inference

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

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

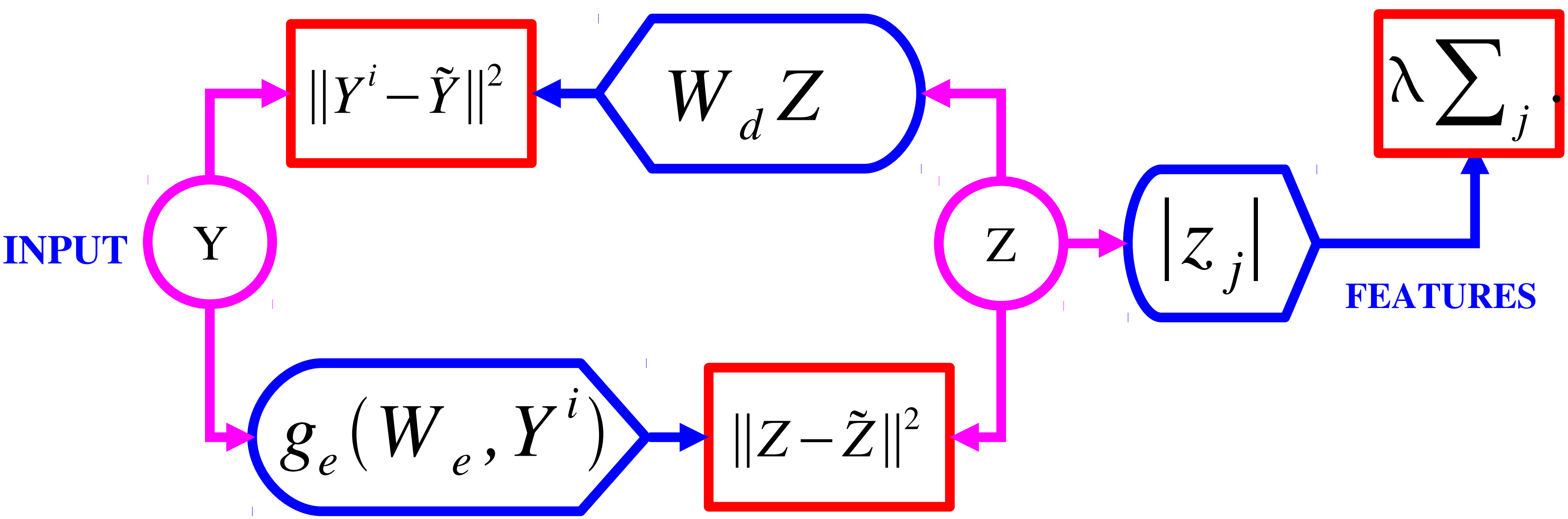
$$Z^i = argmin_z E(Y^i, z; W)$$



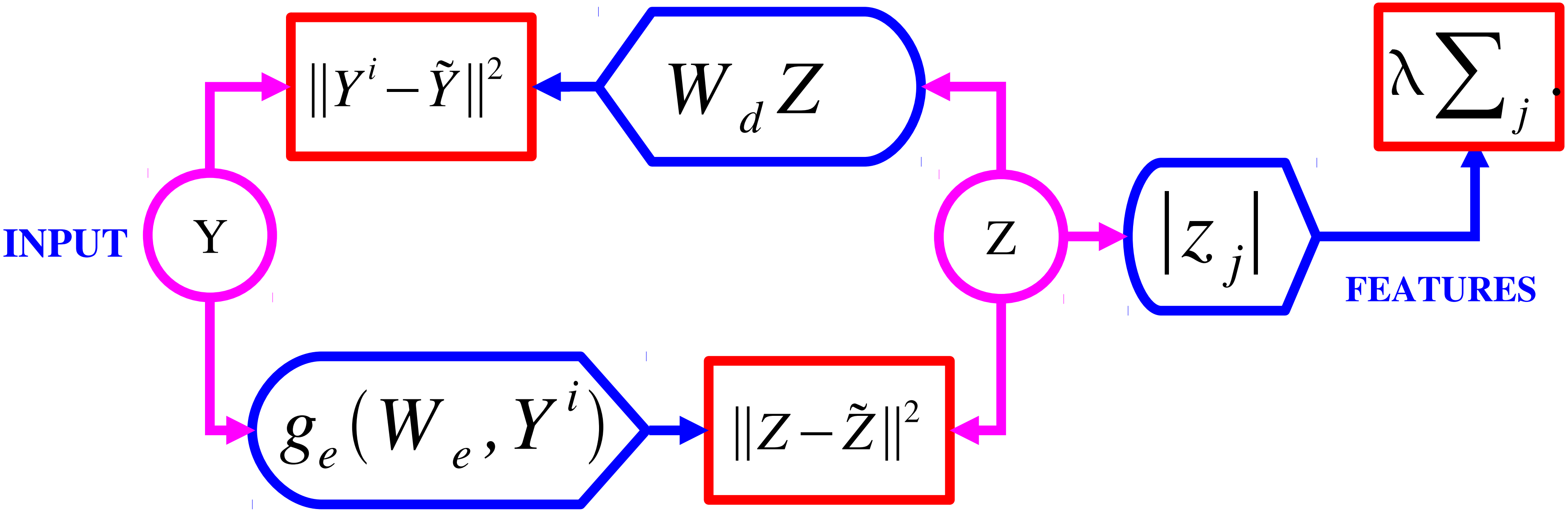
PSD: Learning [Kavukcuoglu et al. 2009]

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

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

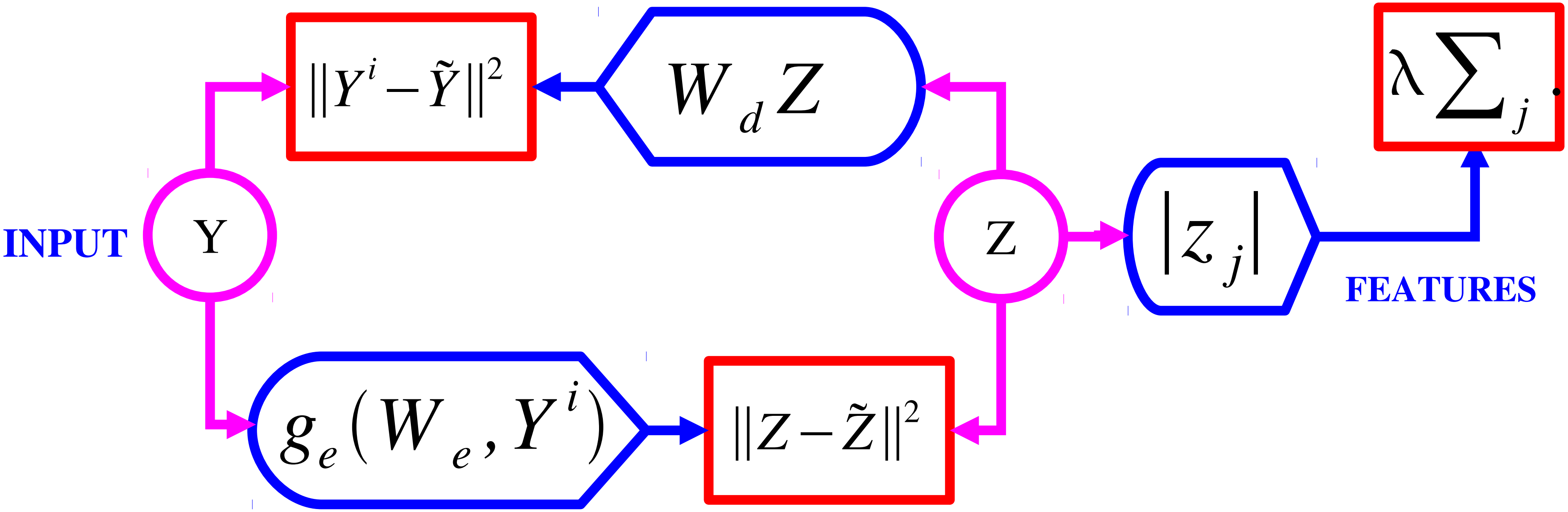


PSD: Learning Algorithm



- 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

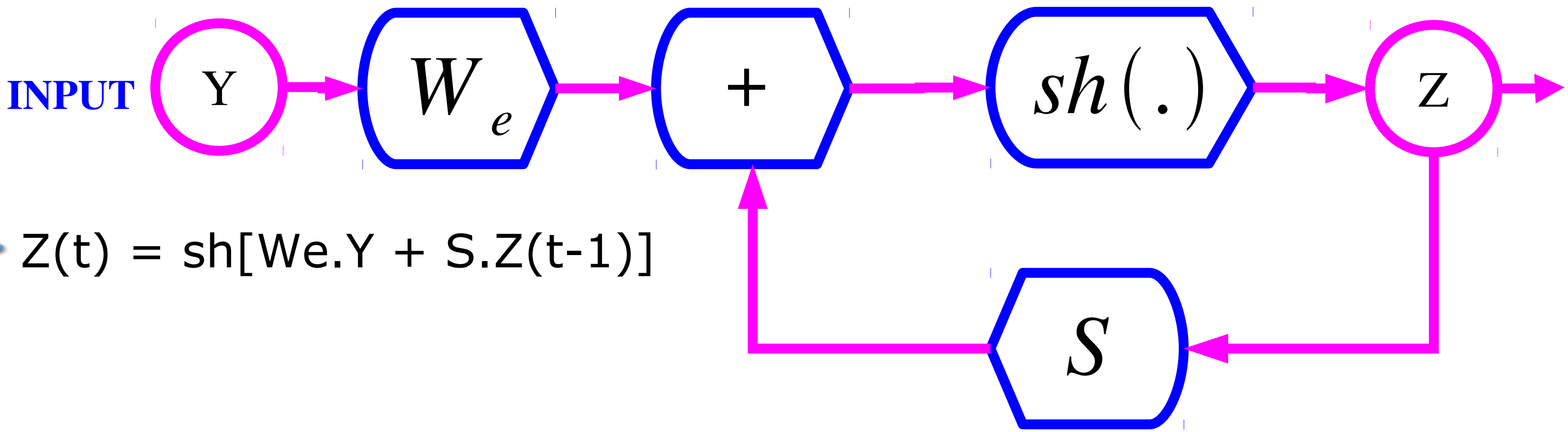
PSD: Encoder Architectures



- **Simple: D.tanh(We.Y)**
- **Sophisticated/Iterative: Z(t) = shrink(We.Y + S.Z(t-1))**
- **For a discussion of best encoders, see [Gregor & LeCun, ICML 2010]**

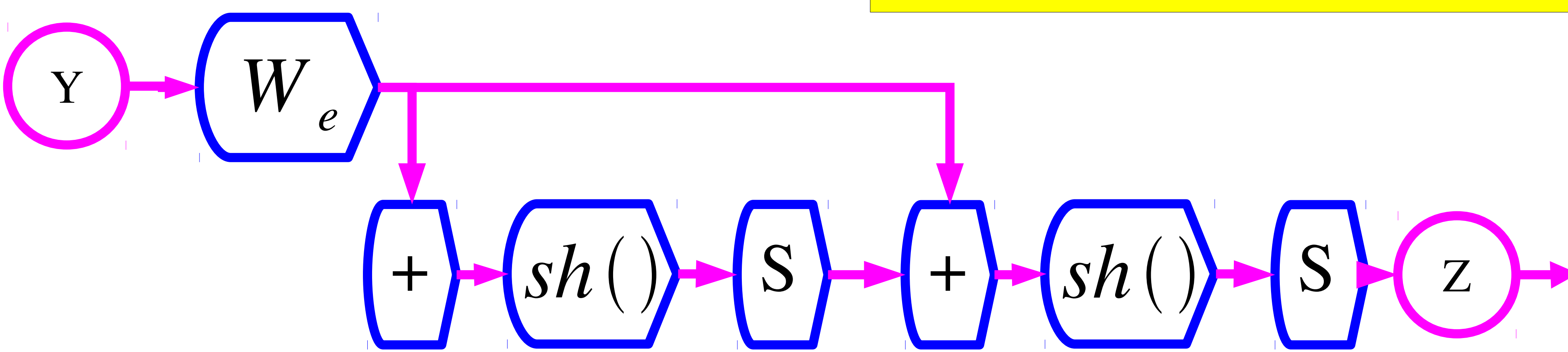
Encoder Architecture inspired by ISTA [Gregor, LeCun ICML 2010] (Iterative Shrinkage and Thresholding Algorithm)

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

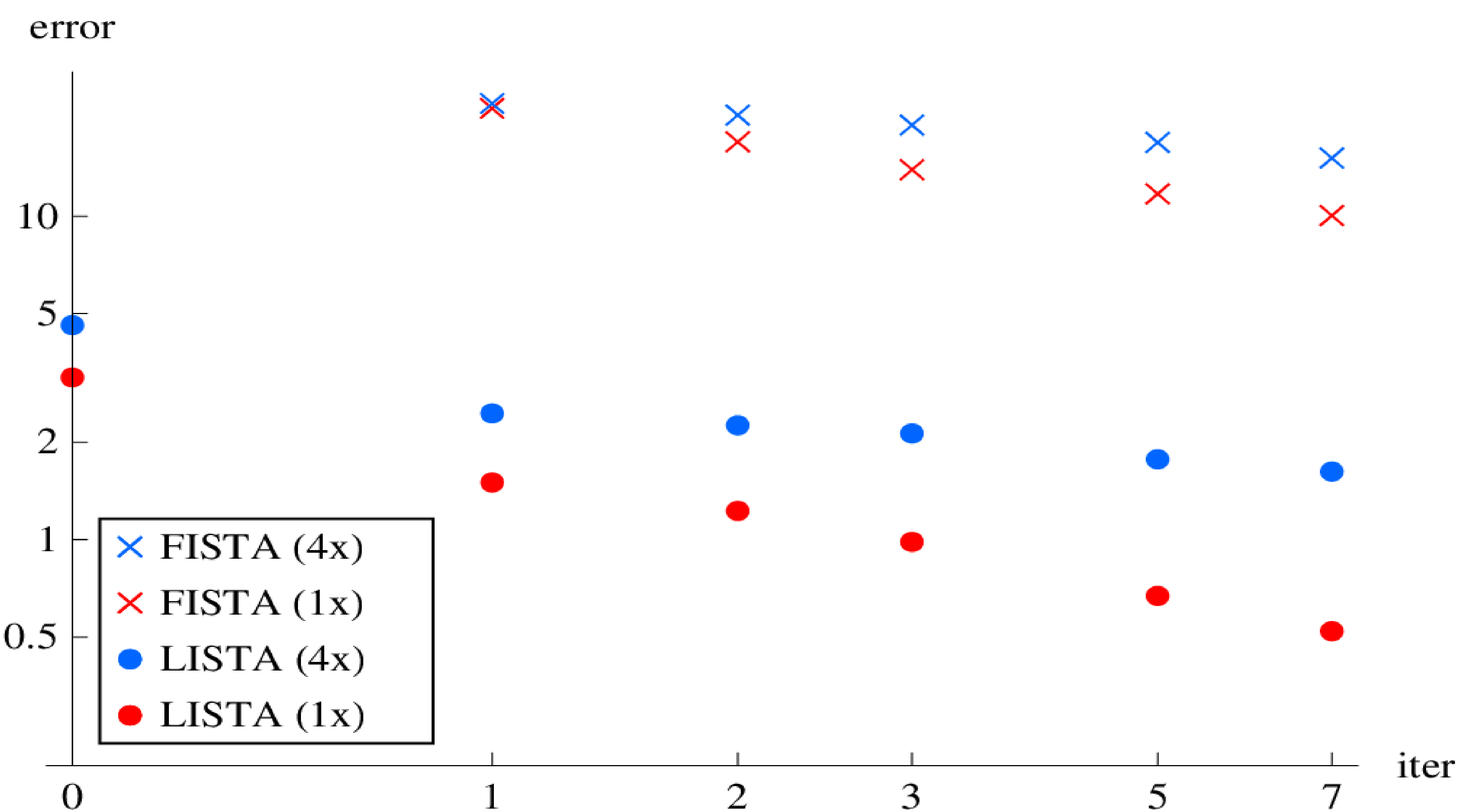


  - Z(t) = sh[We.Y + S.Z(t-1)]

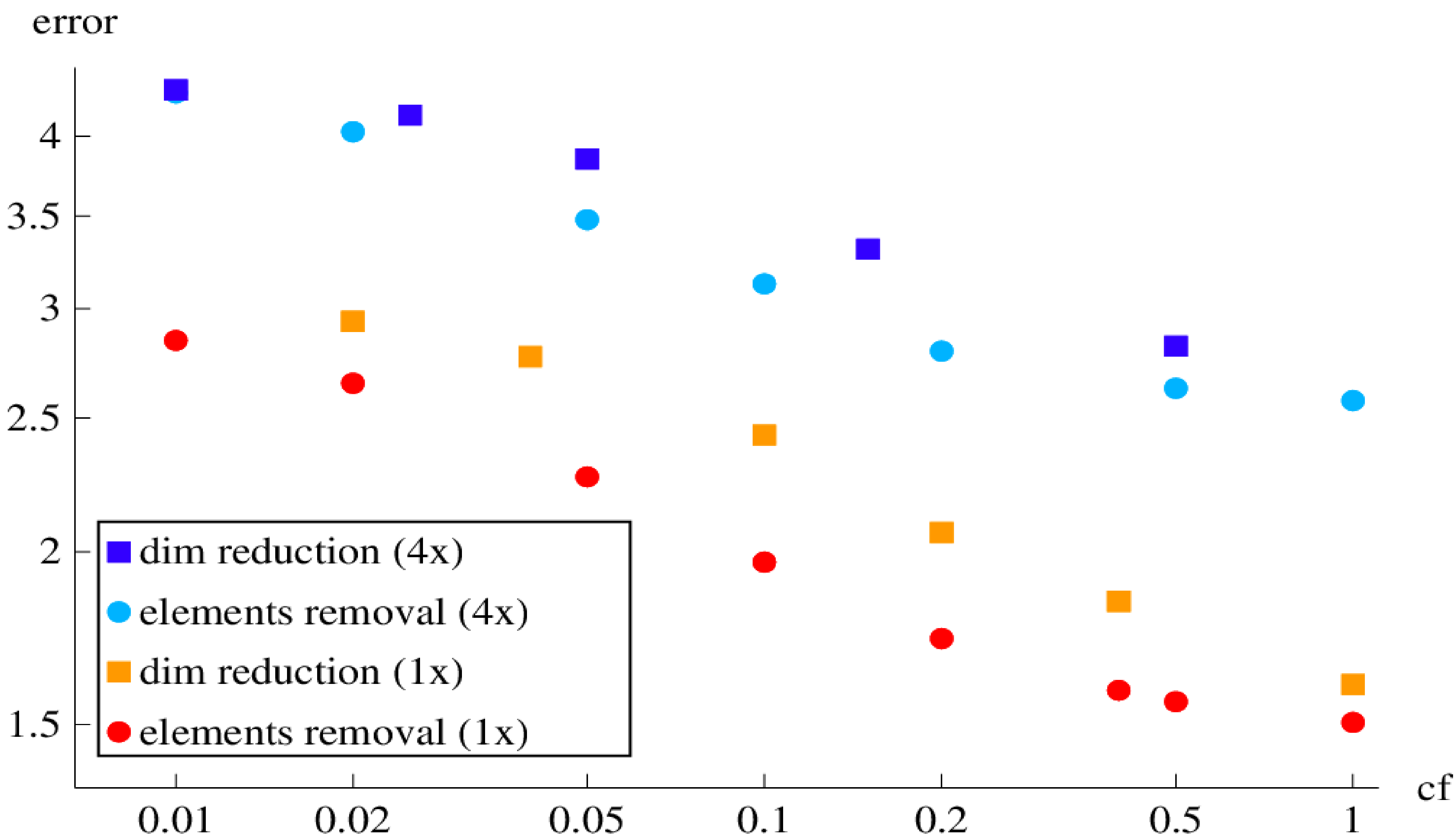
- **Time-Unfolded version of ISTA/FISTA. Idea: learn the We and S matrices**



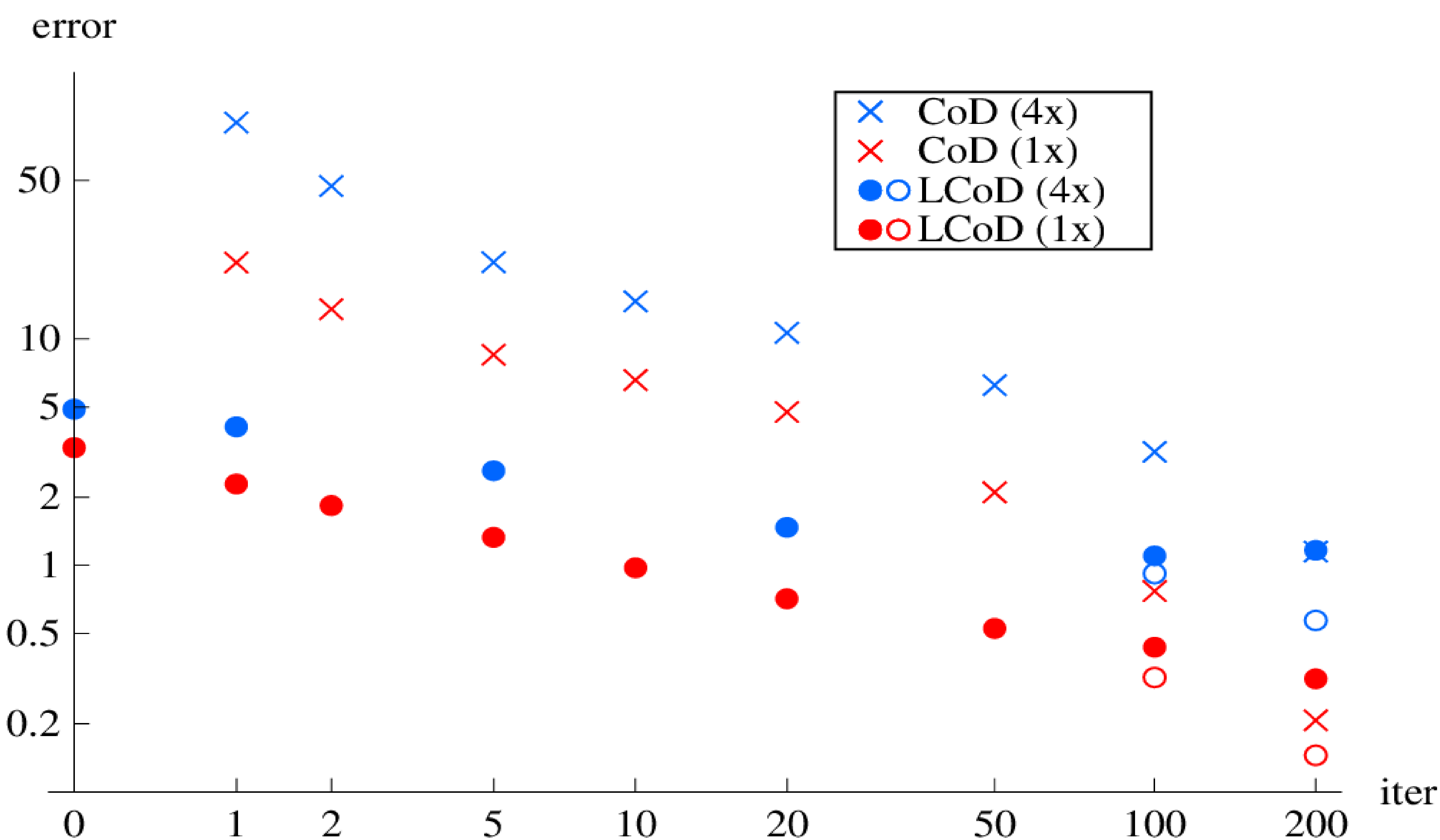
Learning ISTA (LISTA) vs ISTA/FISTA



LISTA with partial mutual inhibition matrix

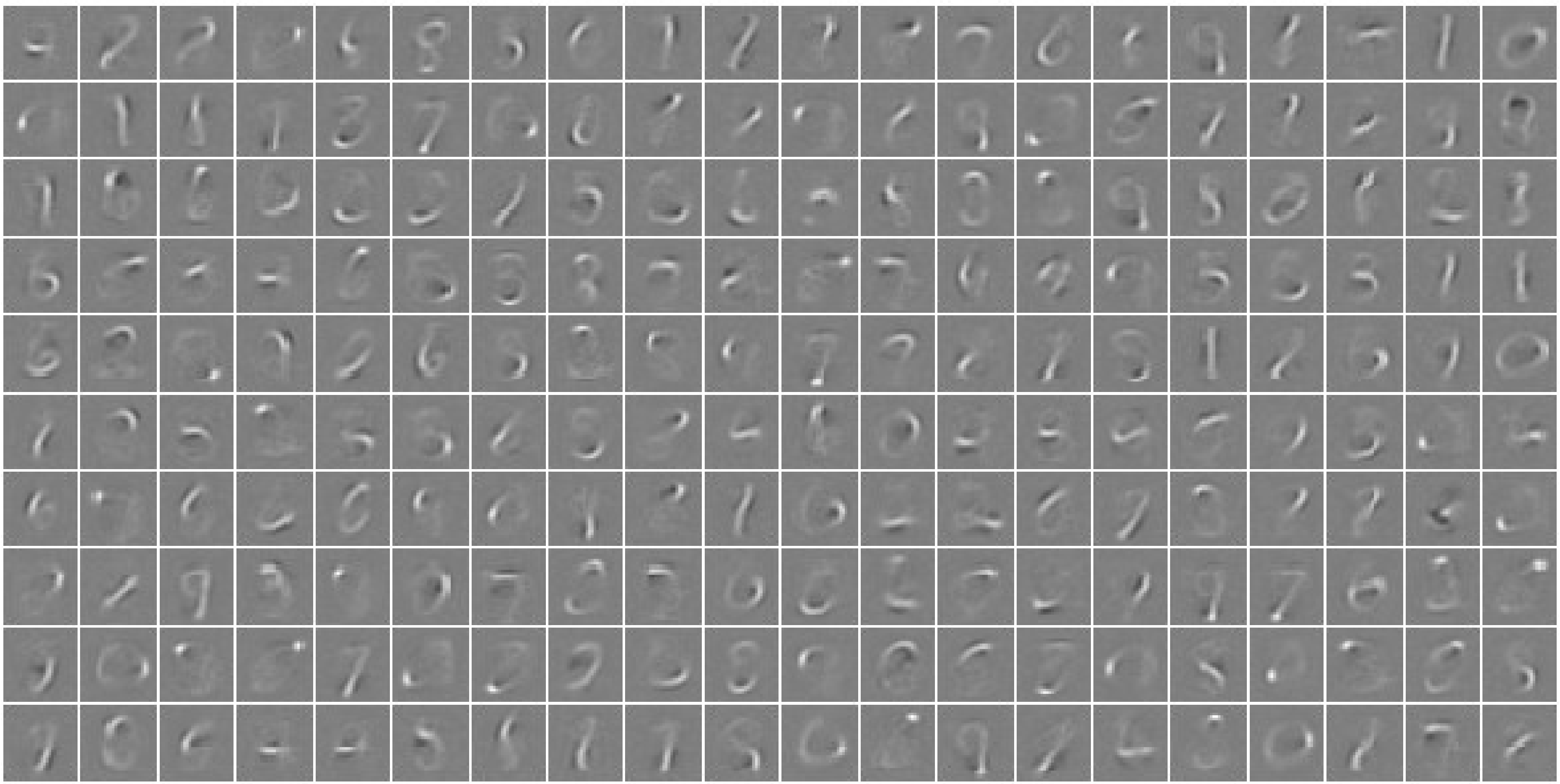


Learning Coordinate Descent (LcoD): faster than LISTA



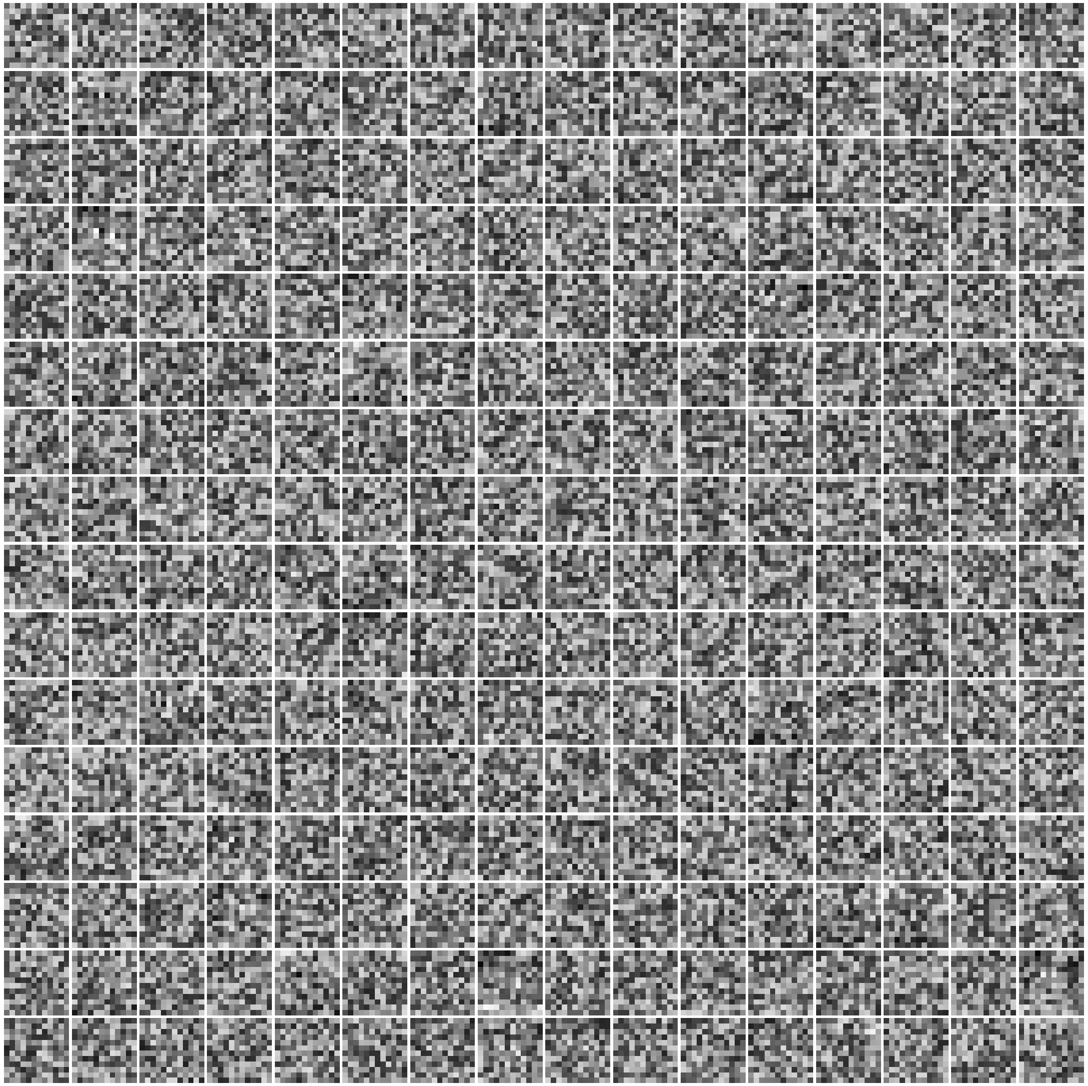
Decoder Basis Functions on MNIST

- **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".



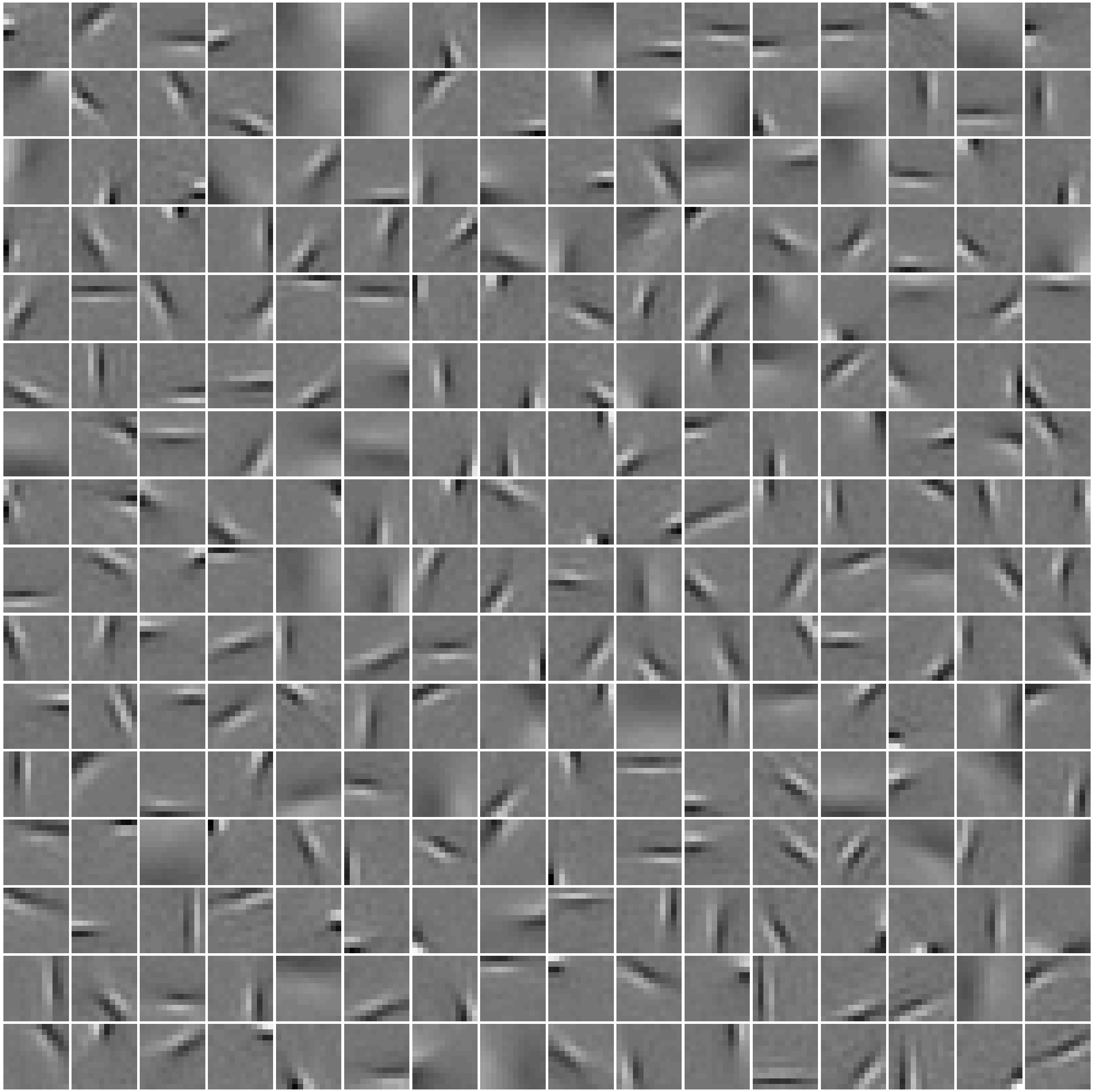
PSD Training on Natural Image Patches

- 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



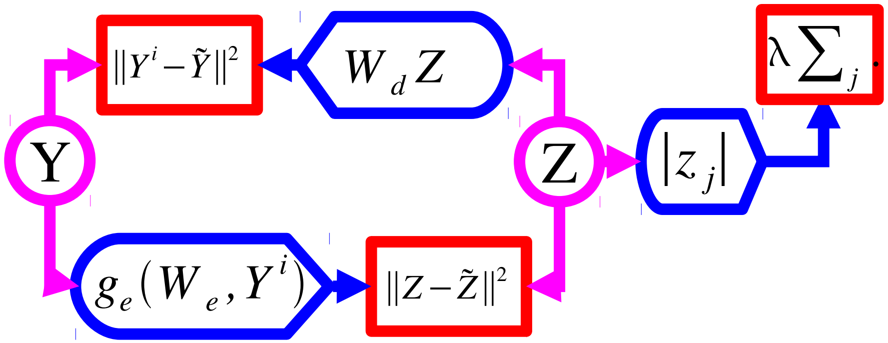
iteration no 0

Learned Features on natural patches: V1-like receptive fields



Using PSD to Train a Hierarchy of Features

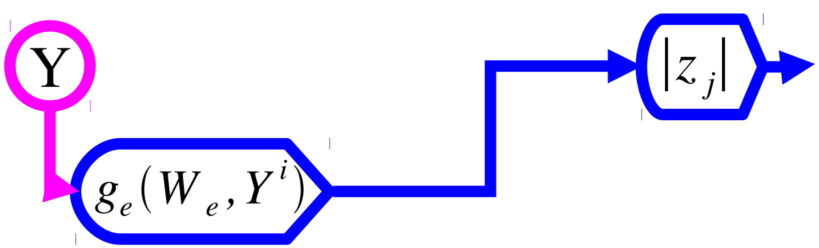
- **Phase 1: train first layer using PSD**



FEATURES

Using PSD to Train a Hierarchy of Features

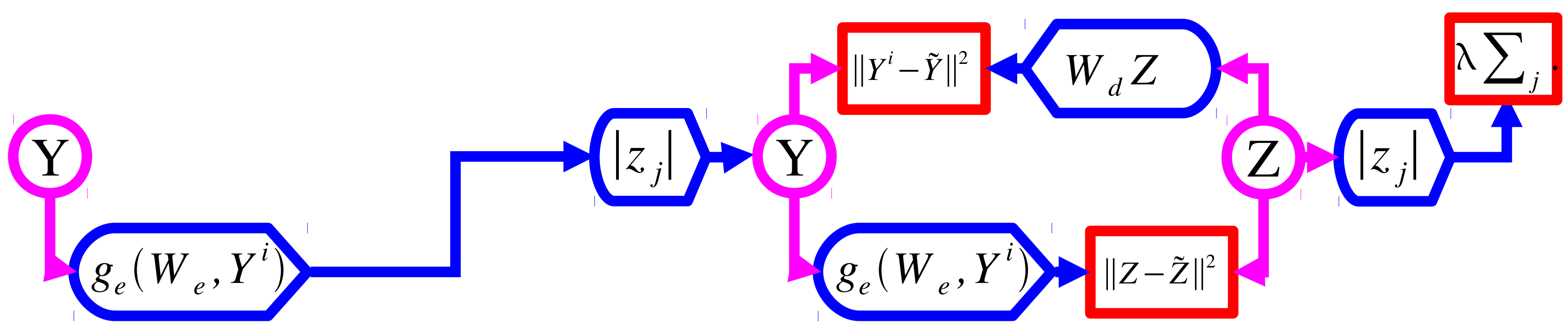
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor



FEATURES

Using PSD to Train a Hierarchy of Features

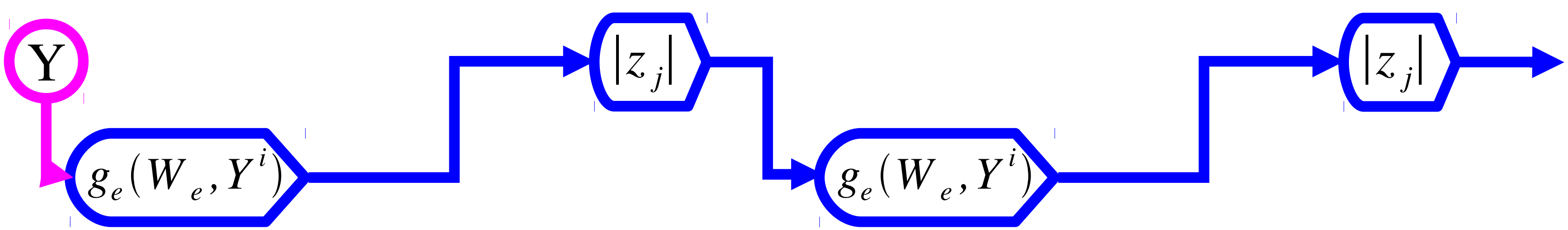
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD



FEATURES

Using PSD to Train a Hierarchy of Features

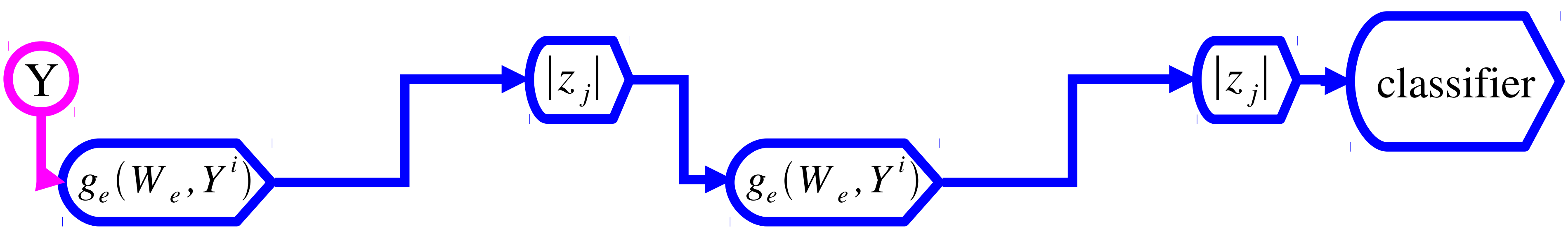
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor



FEATURES

Using PSD to Train a Hierarchy of Features

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



FEATURES

A General View of Unsupervised Learning

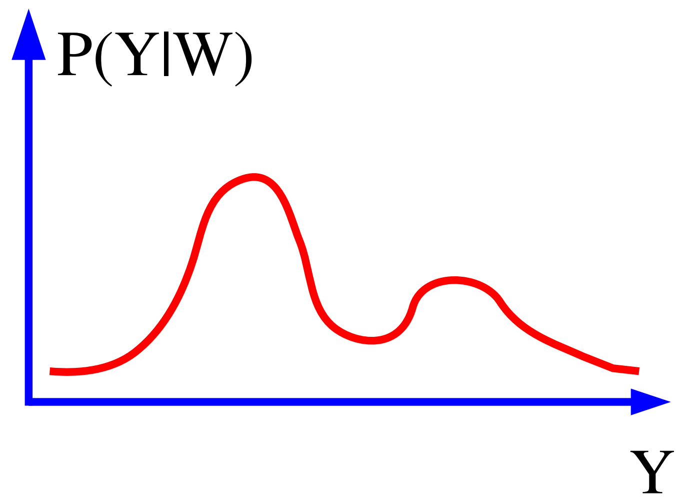
“Deep Learning”

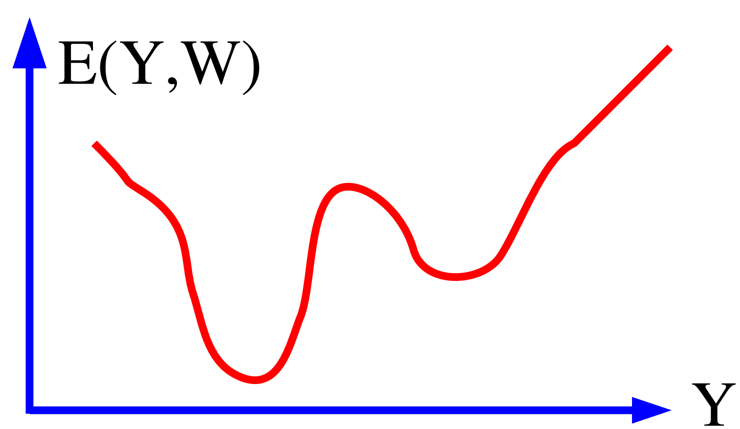
[Hinton 05, Bengio 06, LeCun 06, Ng 07]

- **The “deep learning” method was popularized by Hinton for training “deep belief networks”.**
  - DBN use a special kind of encoder-decoder architecture called Restricted Boltzmann Machines (RBM)
- **1. Train each layer in an unsupervised fashion, layer by layer**
- **2. Stick a supervised classifier on top, and refine the entire system with gradient descent (back-prop) on a supervised criterion.**

Unsupervised Learning: Capturing Dependencies Between Variables

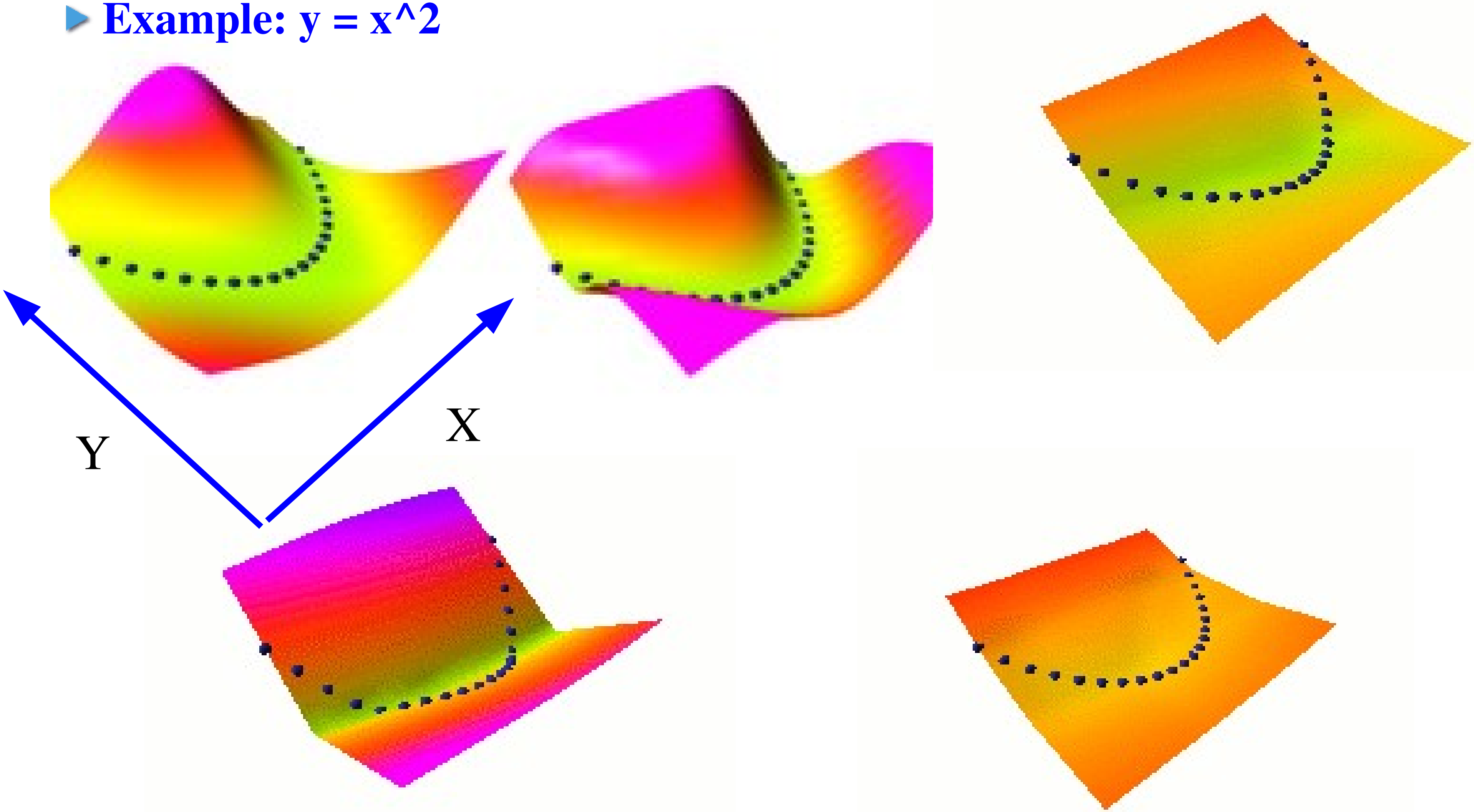
- **Energy function: viewed as a negative log probability density**
- **Probabilistic View:**
  - Produce a probability density function that:
  - has high value in regions of high sample density
  - has low value everywhere else (integral = 1).
- **Energy-Based View:**
  - produce an energy function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





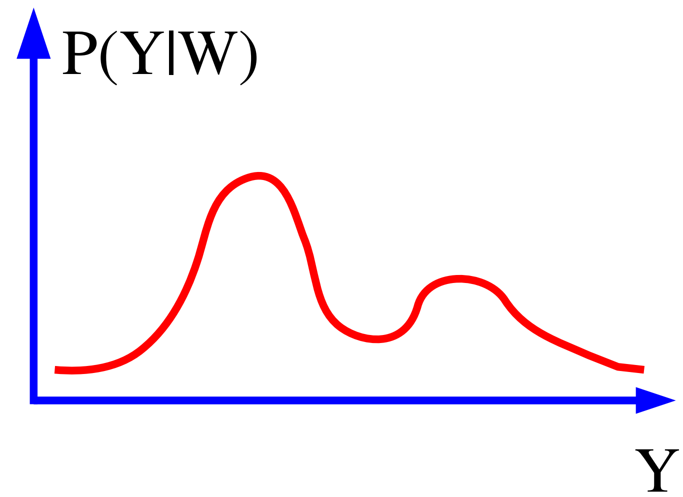
Unsupervised Learning: Capturing Dependencies Between Variables

- **Energy function viewed as a negative log density**
  - **Example: $y = x^2$**

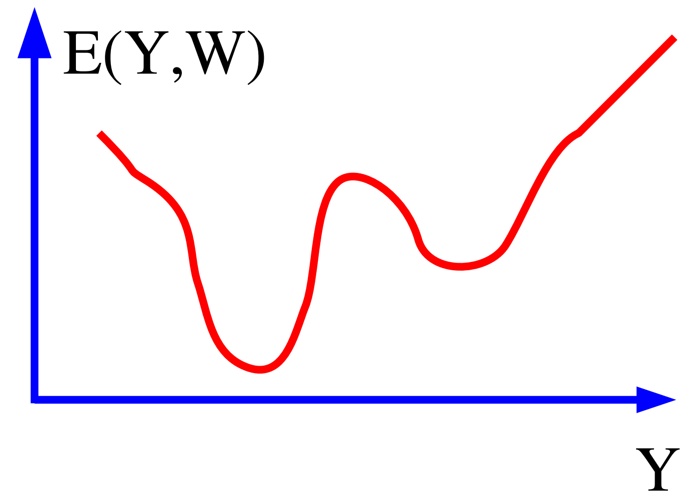


Energy <-> Probability

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

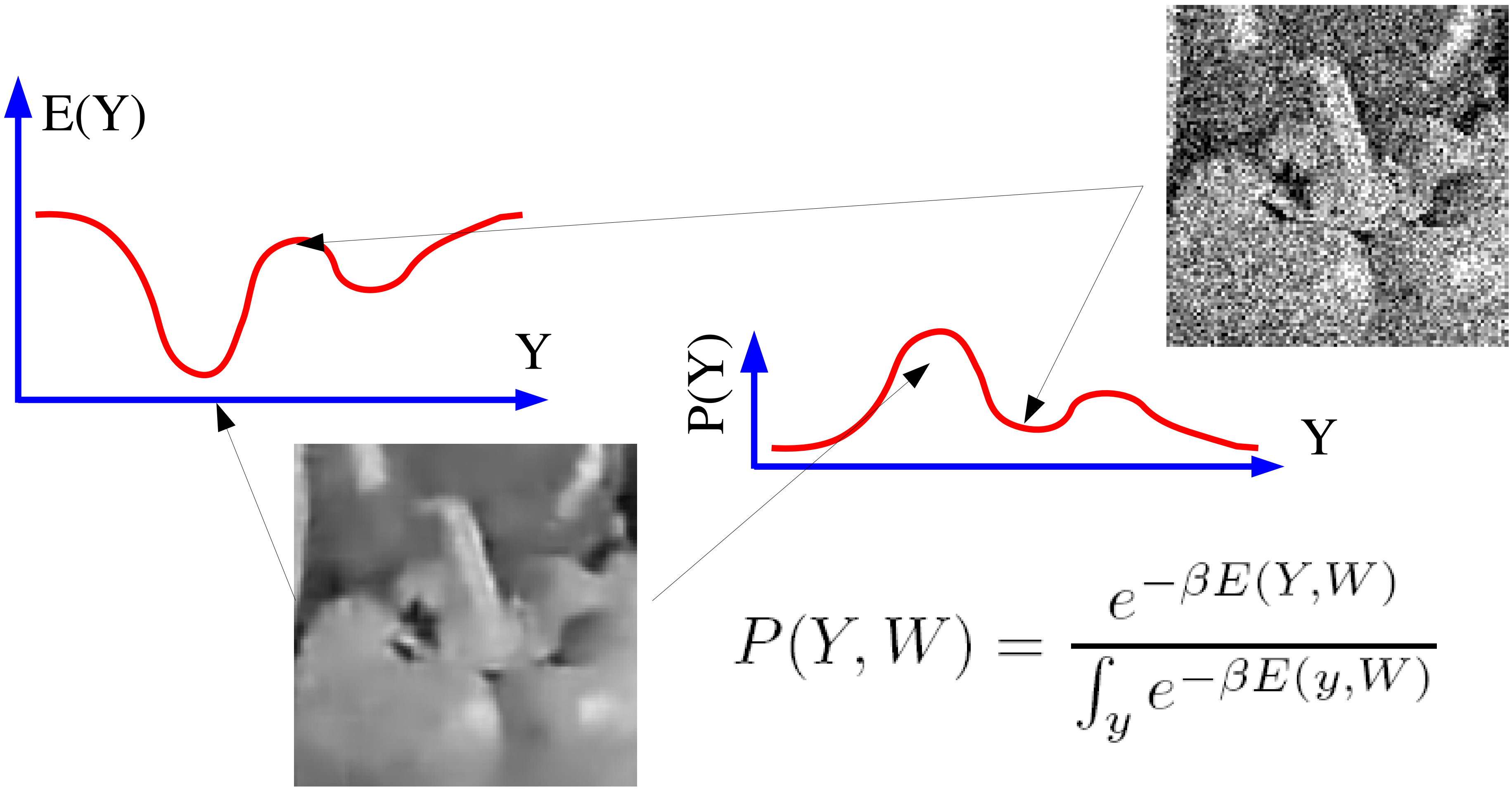


$$E(Y,W) \propto -\log P(Y|W)$$



Training an Energy-Based Model

- Make the energy around training samples low
- Make the energy everywhere else higher



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

Training an Energy-Based Model to Approximate a Density

Maximizing P(Y|W) on training samples

make this big

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

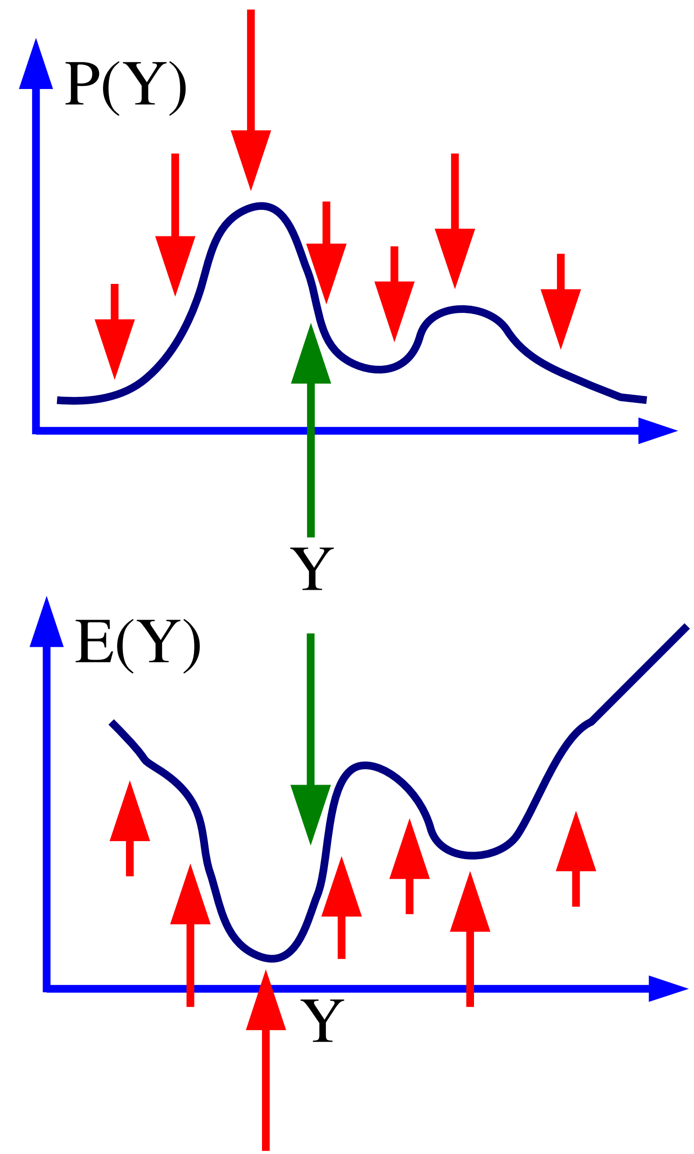
make this small

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



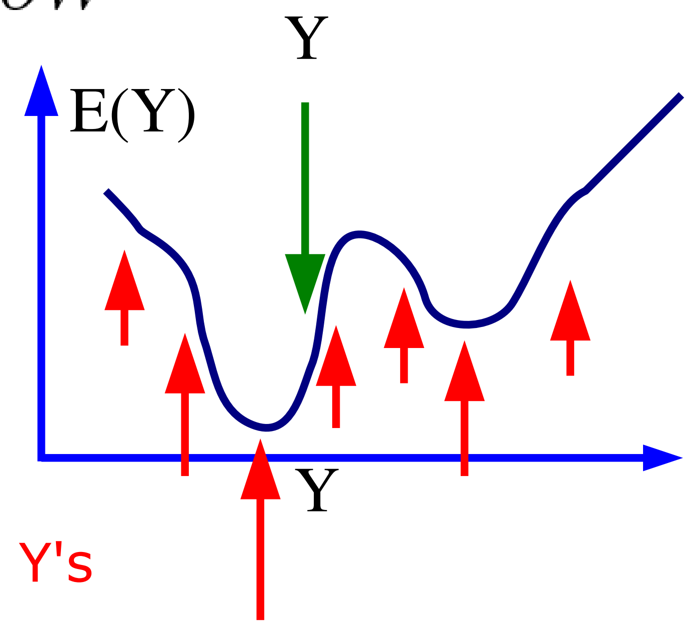
Training an Energy-Based Model with Gradient Descent

- **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

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

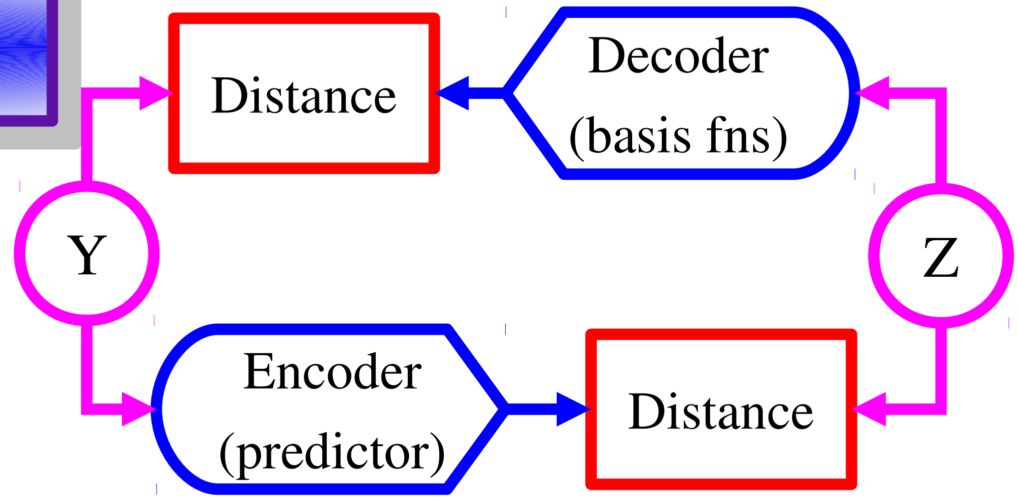
How do we push up on the energy of everything else?

- **Solution 1: contrastive divergence [Hinton 2000]**
  - Move away from a training sample a bit
  - Push up on that
- **Solution 2: score matching**
  - On the training samples: minimize the gradient of the energy, and maximize the trace of its Hessian.
- **Solution 3: denoising auto-encoder (not really energy-based)**
  - Train the inference dynamics to map noisy samples to clean samples
- **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]

Restricted Boltzmann Machines

[Hinton & Salakhutdinov 2005]

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



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

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

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

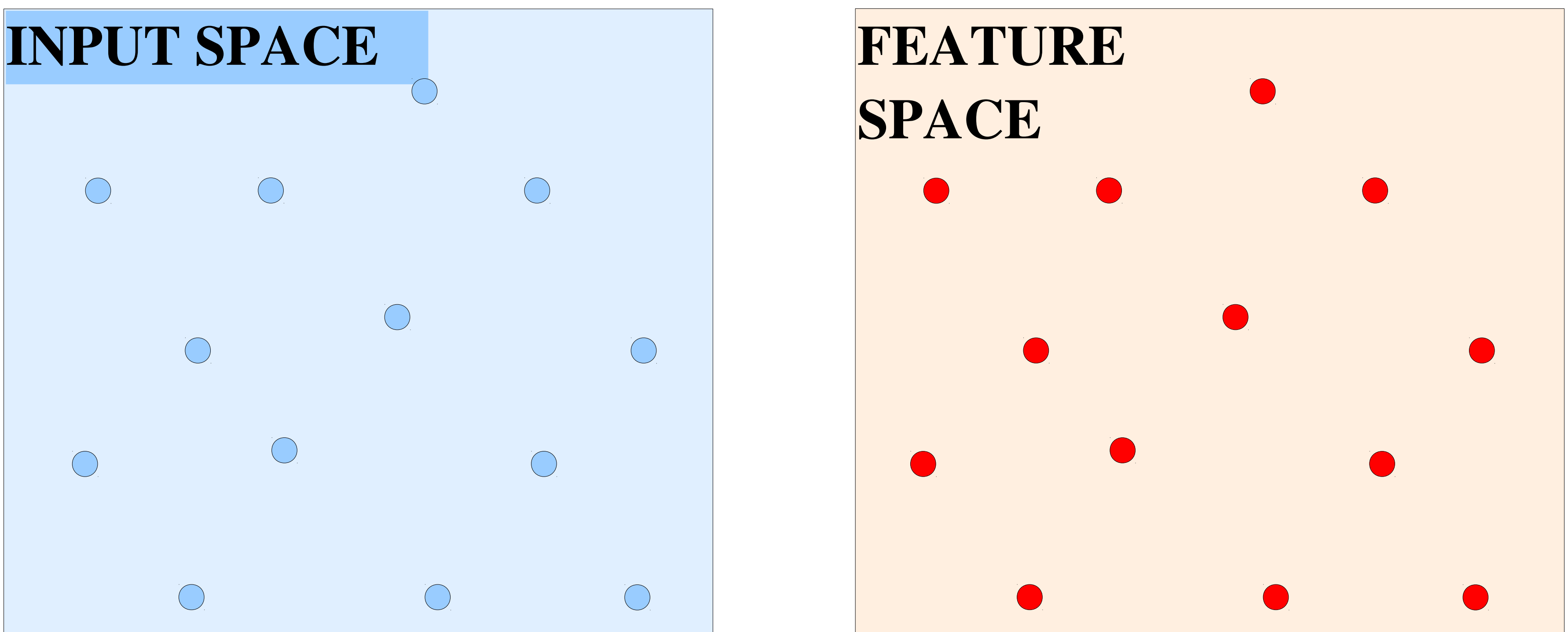
$$E(Y,Z) = -Z^T.W.Y \quad F(Y) = -\log \sum_z e^{Z^T.W.Y}$$

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?

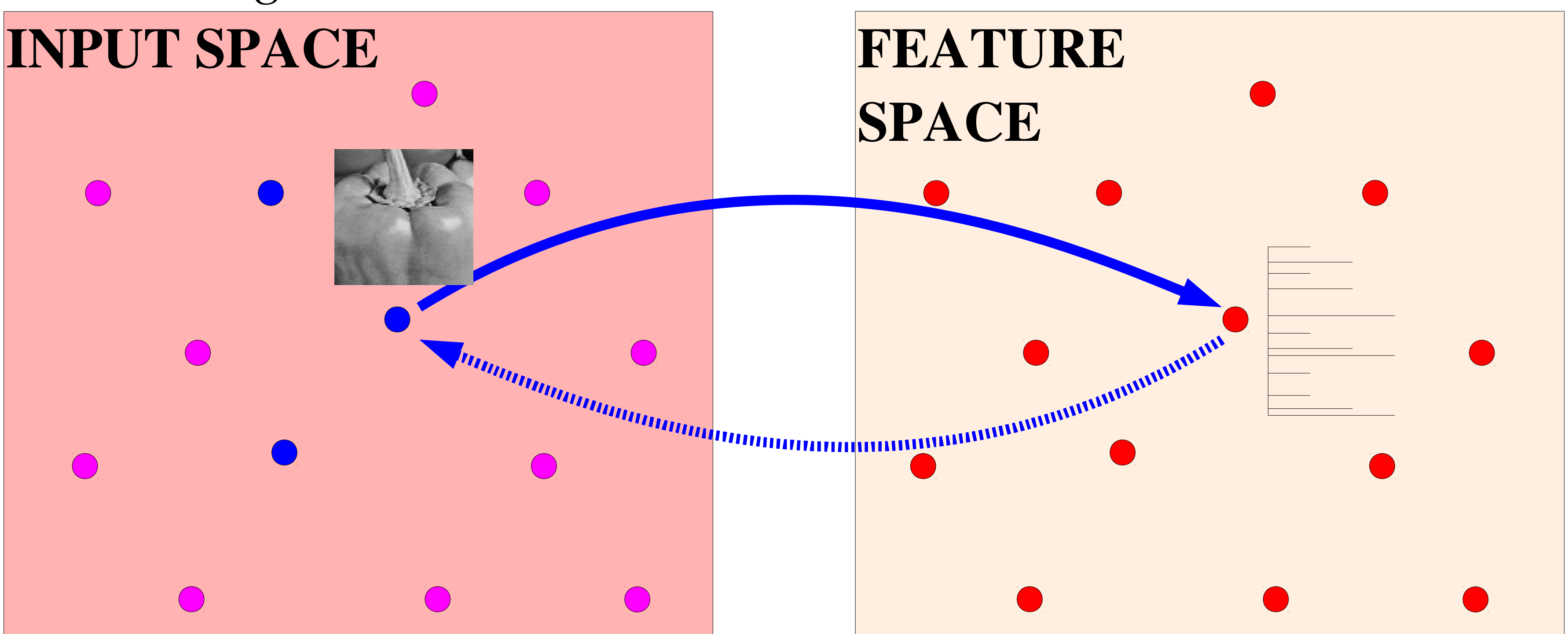
- Training sample
- Input vector which is NOT a training sample
- Feature vector



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

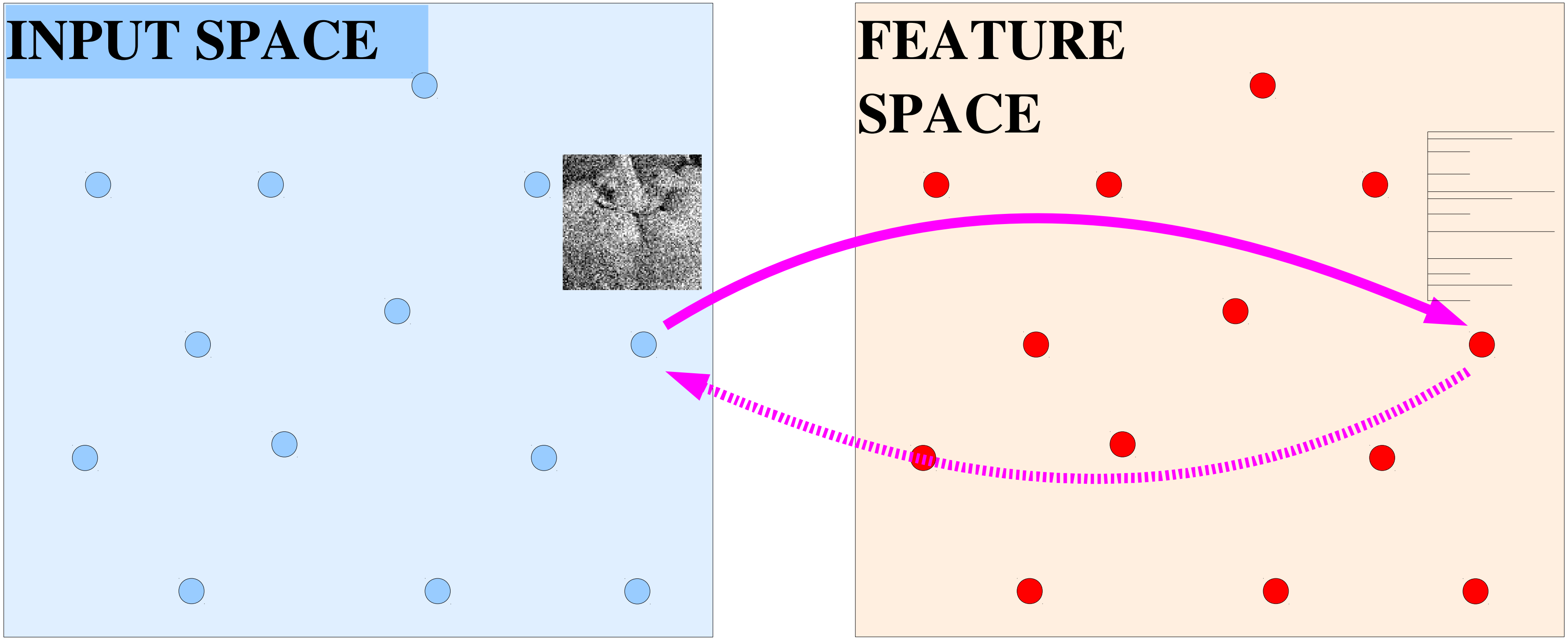


Why Limit the Information Content of the Code?

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

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

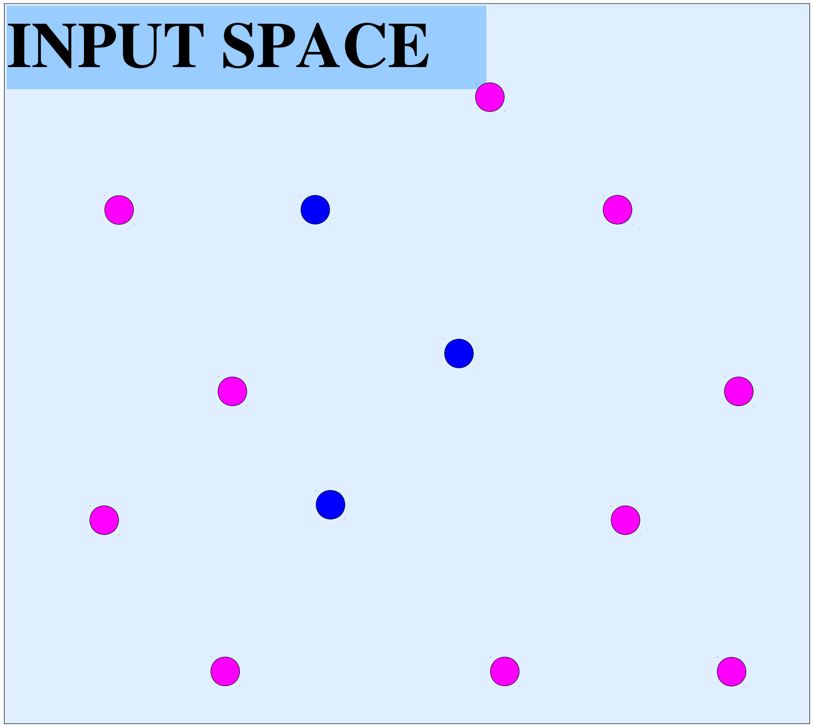
It just copies the data.

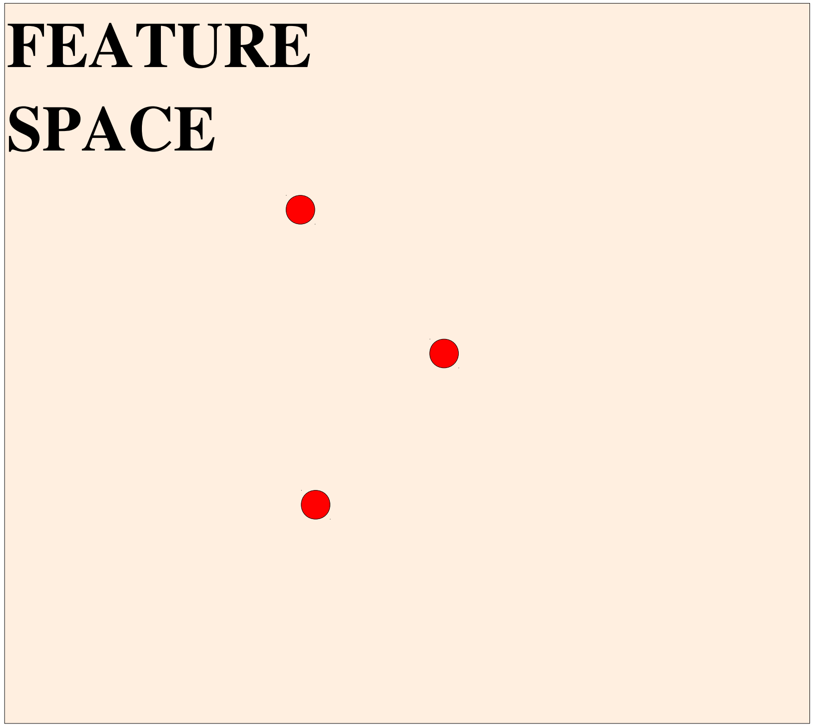


Why Limit the Information Content of the Code?

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

IDEA: reduce number of available codes.

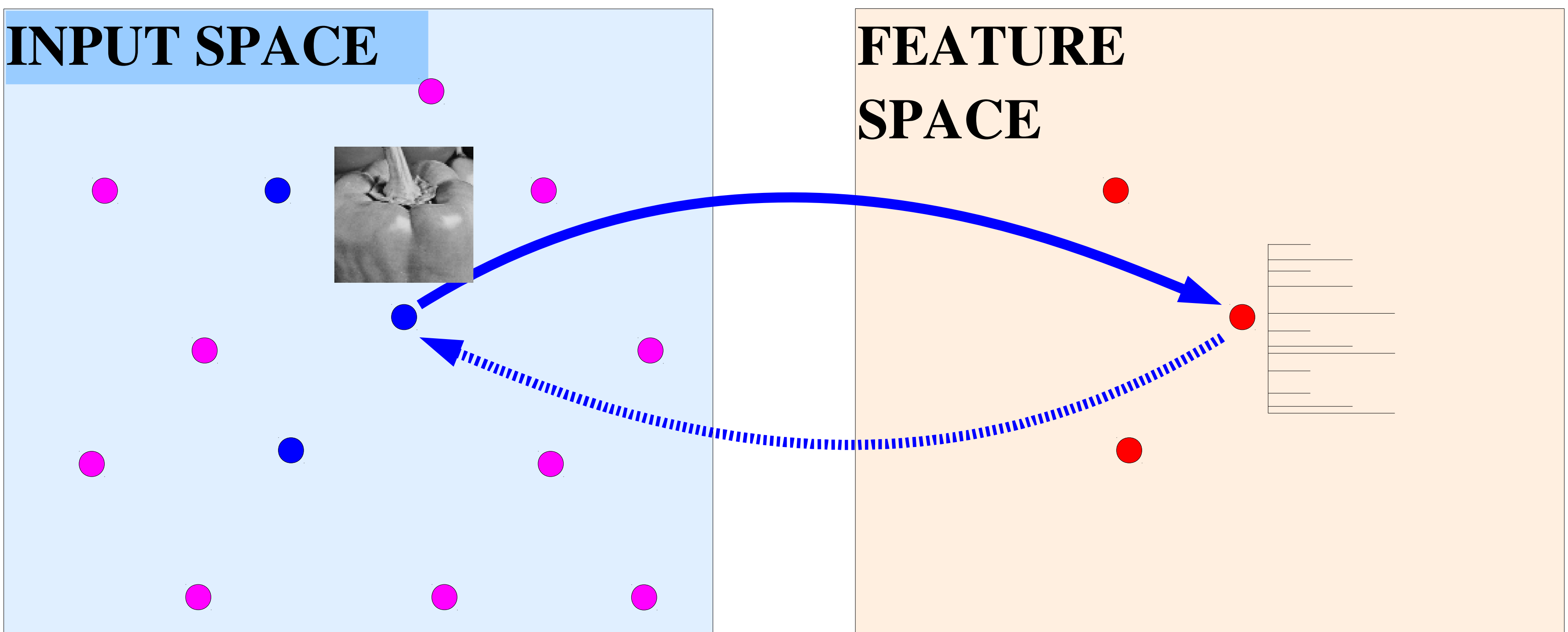




Why Limit the Information Content of the Code?

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

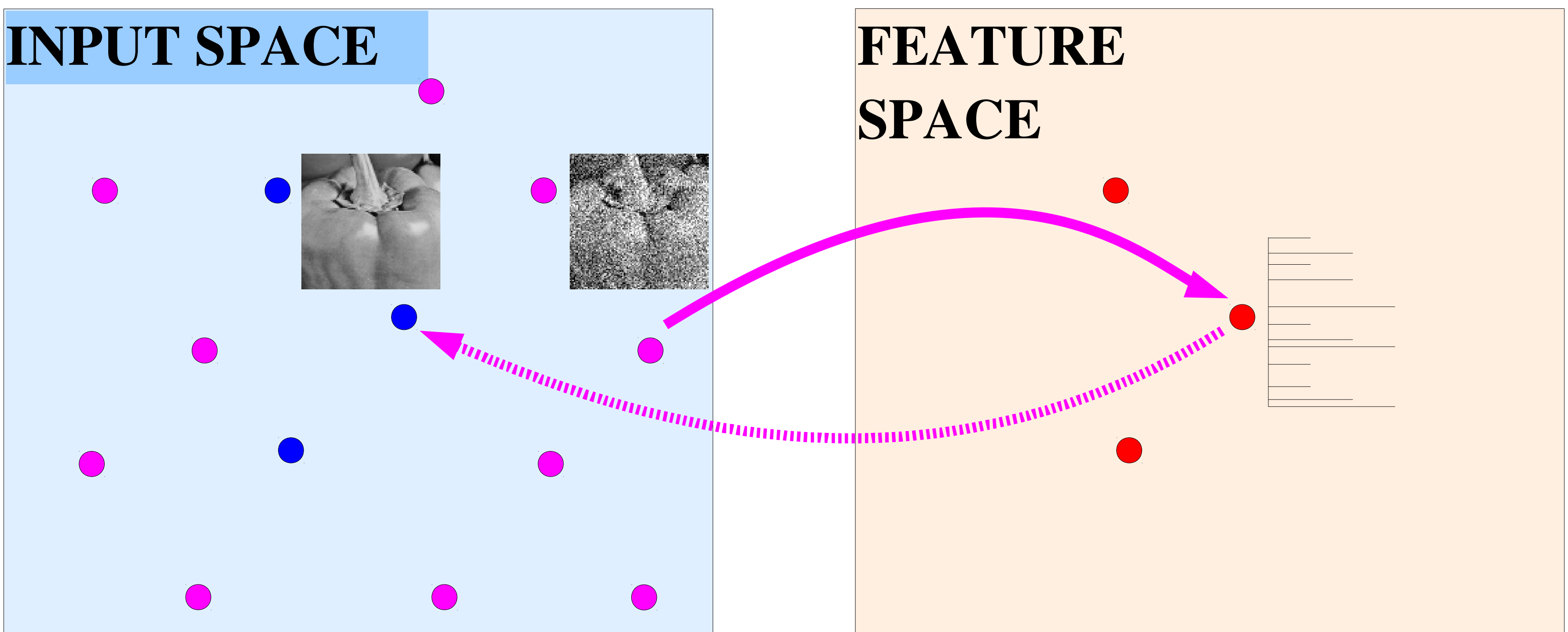
IDEA: reduce number of available codes.



Why Limit the Information Content of the Code?

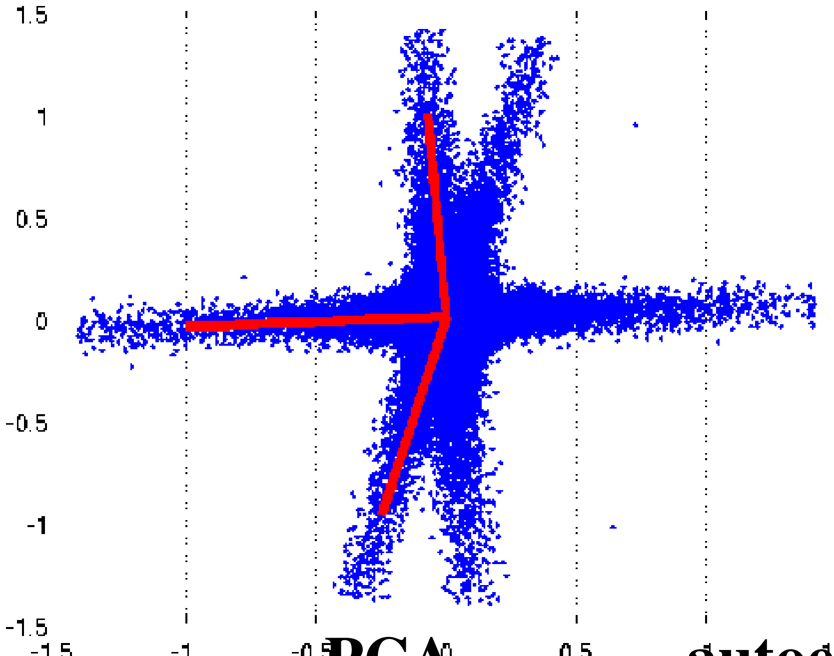
- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**

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.

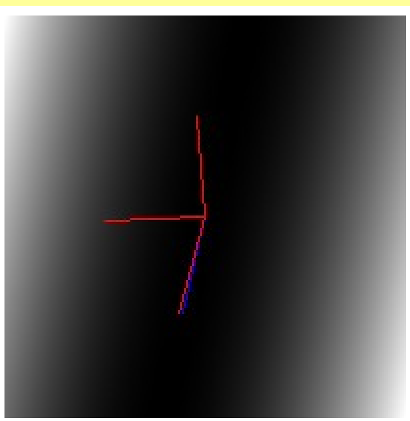


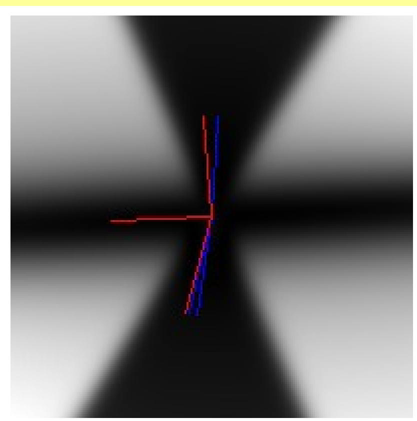
- 2 dimensional toy dataset
  - Mixture of 3 Cauchy distrib.
- Visualizing energy surface

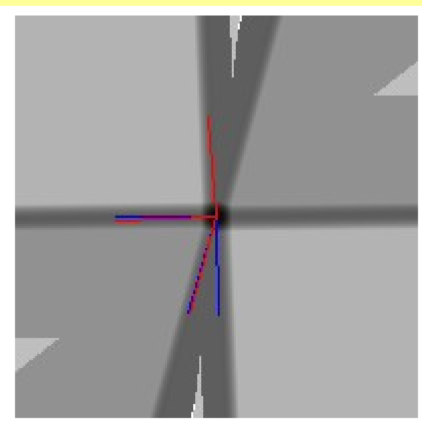
(black = low, white = high)

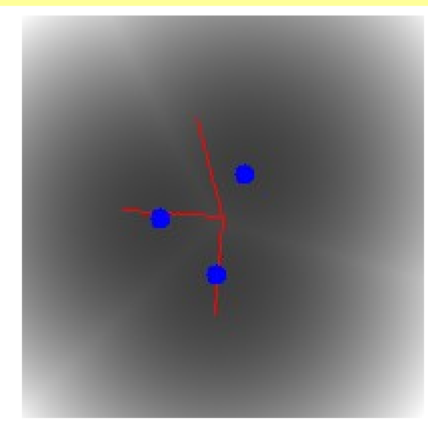
[Ranzato 's PhD thesis 2009]

	PCA (1 code unit)	**autoencoder (3 code units)**	**sparse coding (3 code units)**	**K-Means (3 code units)**
encoder	$W'Y$	$\sigma(W_e Y)$	$-$	$-$
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

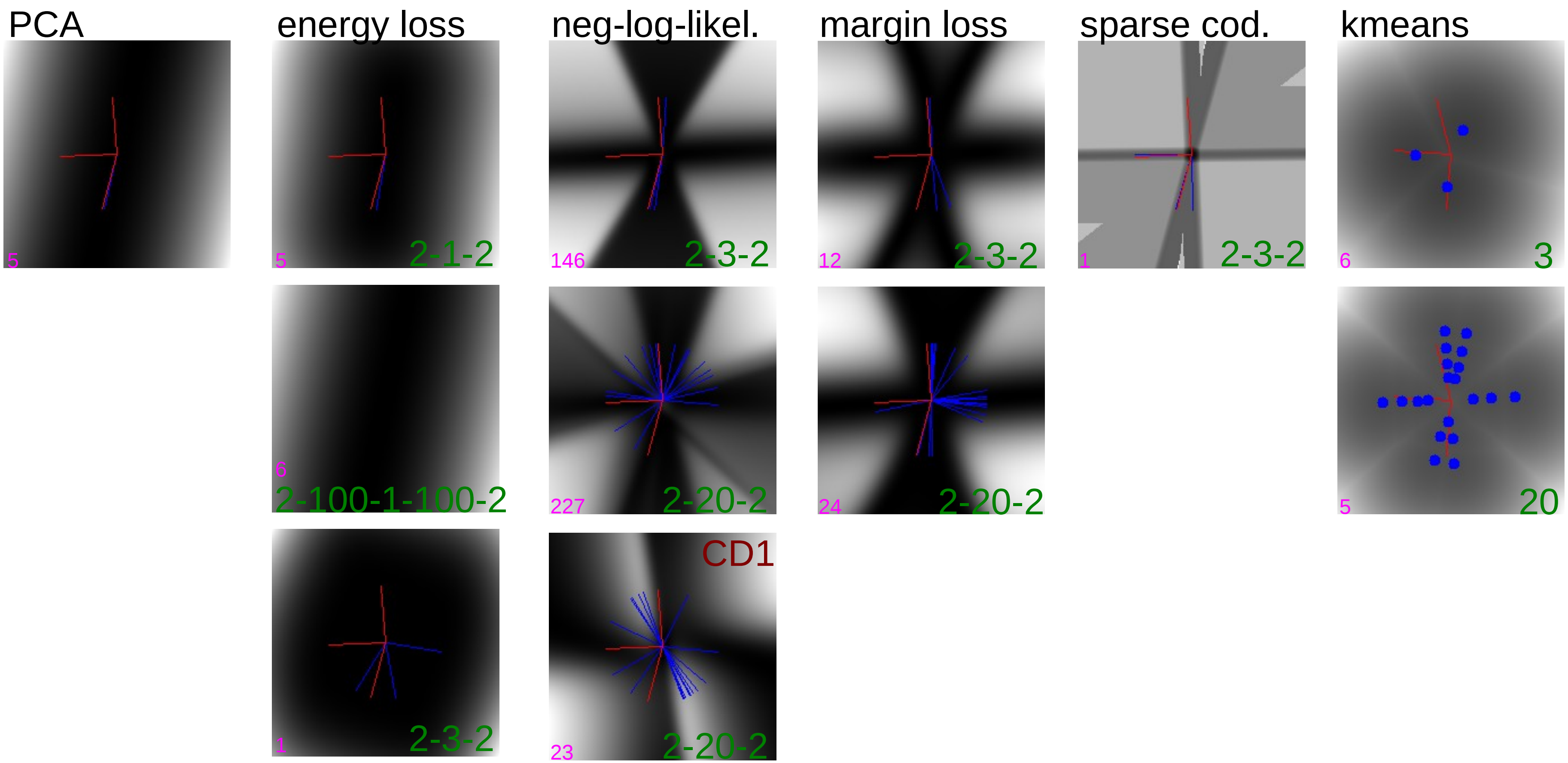






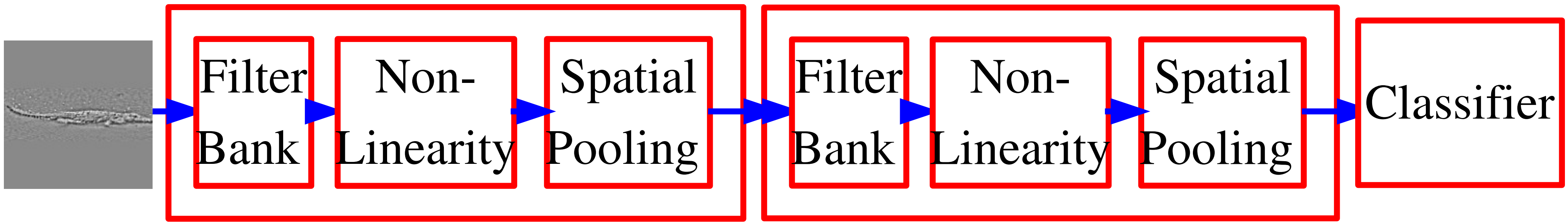


Energies Surfaces for Various Loss Functions



Back To Object Recognition

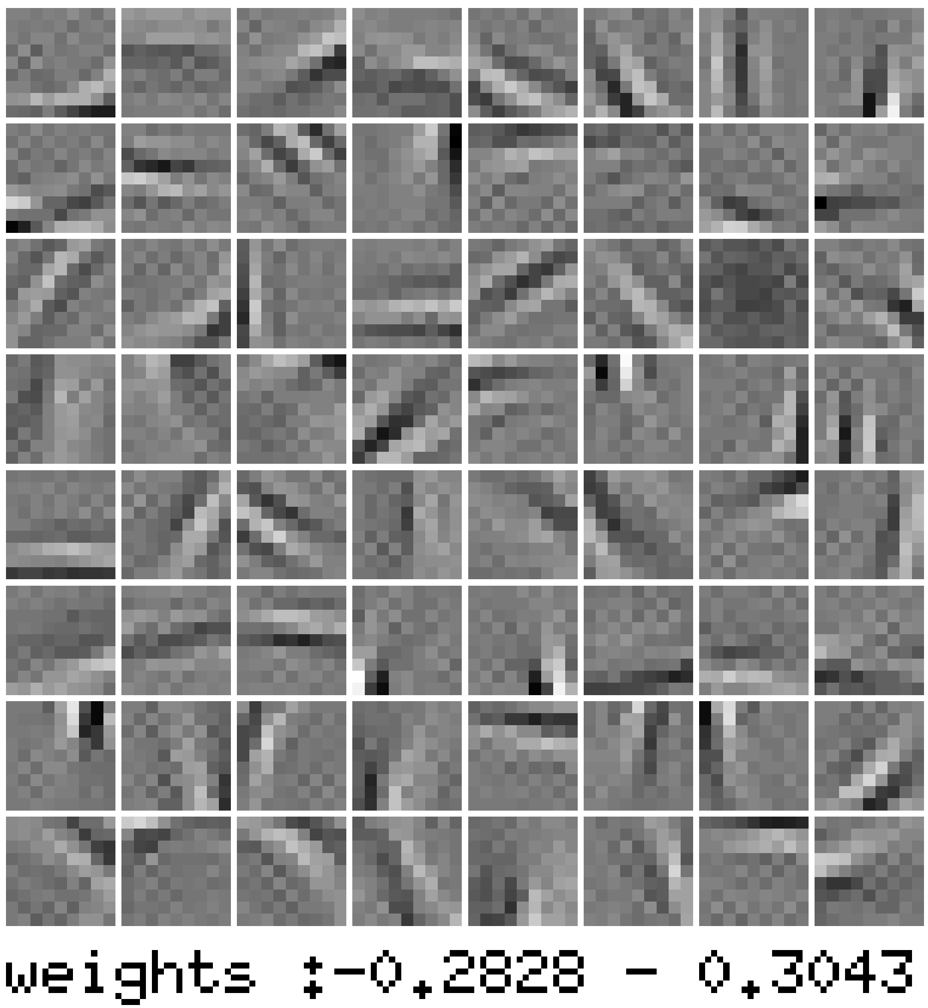
Training a Multi-Stage Hubel-Wiesel Architecture with PSD



- **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?**

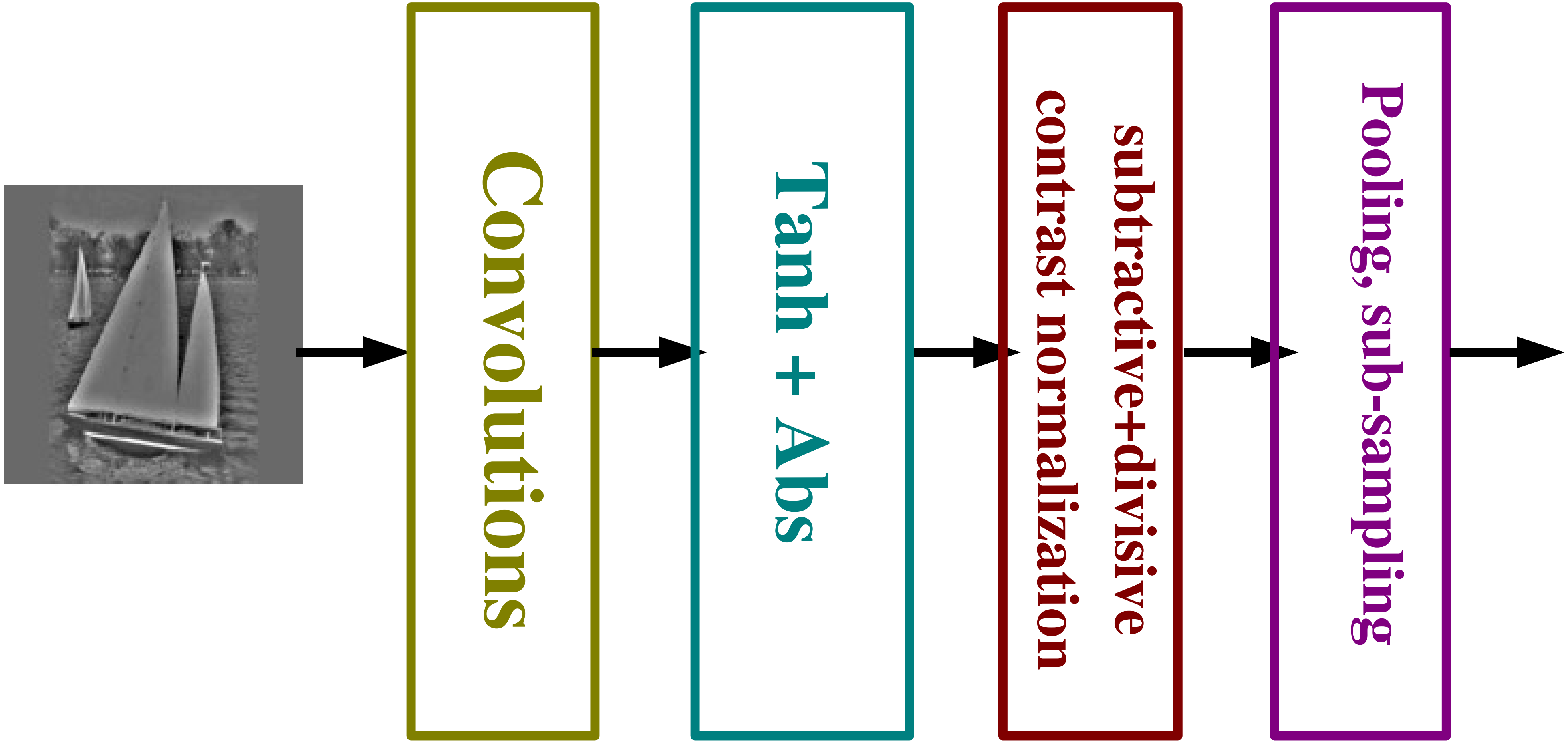
Using PSD Features for Object Recognition

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



Adding Rectification and Normalization Modules

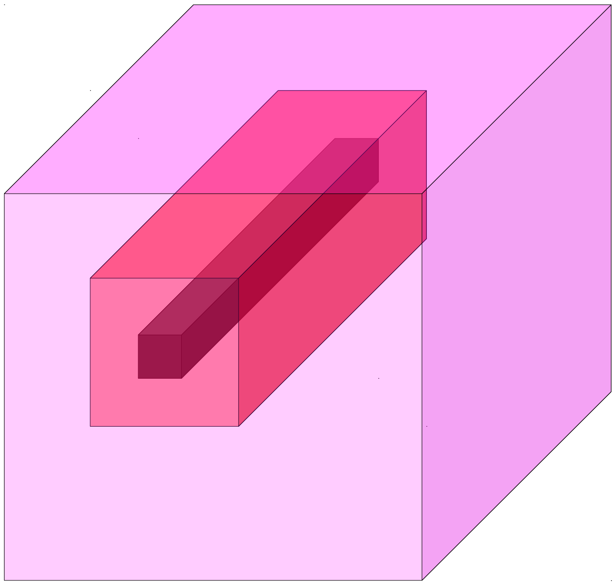
- **C** Convolutions (filter bank)
- **Tanh+Abs** tanh (sigmoid) + Absolute Value Rectification
- **N** Subtractive and Divisive Local Normalization
- **P** Pooling down-sampling layer: average or max?



THIS IS ONE STAGE OF THE CONVNET

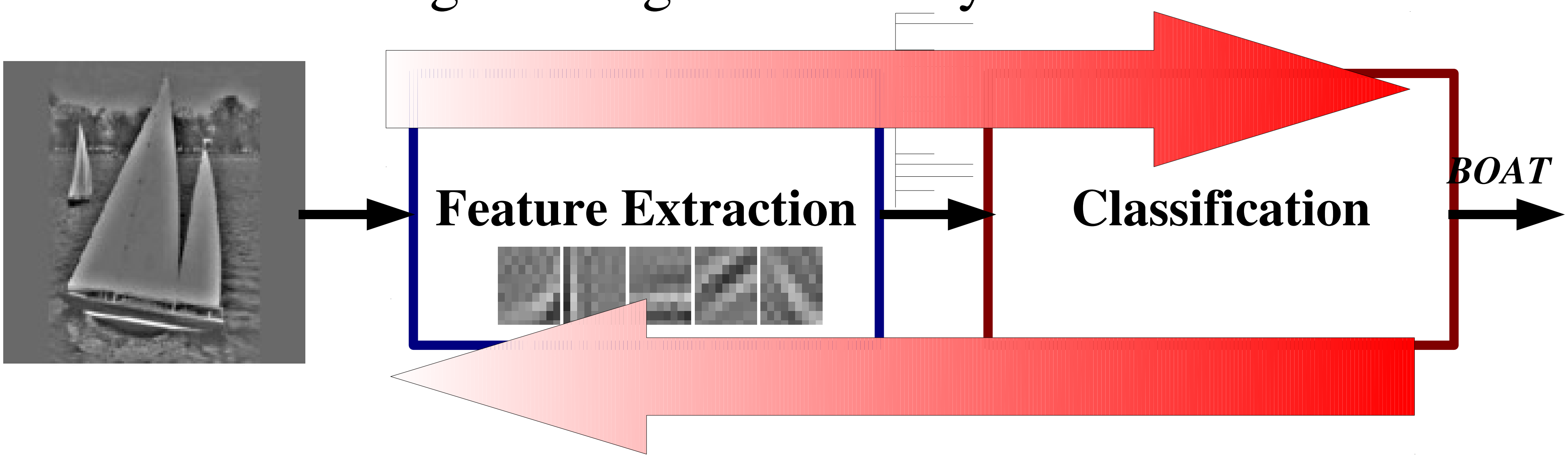
Local Contrast Normalization

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



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%

Multistage Hubel-Wiesel Architecture

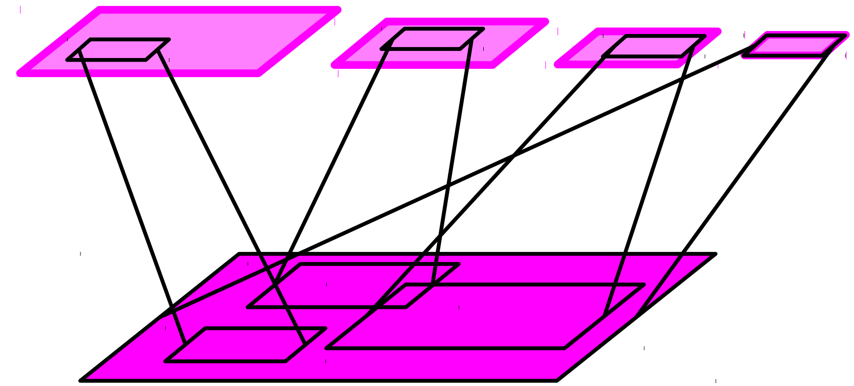
- **Image Preprocessing:**
  - High-pass filter, local contrast normalization (divisive)
- **First Stage:**
  - Filters: 64 9x9 kernels producing 64 feature maps
  - Pooling: 10x10 averaging with 5x5 subsampling
- **Second Stage:**
  - Filters: 4096 9x9 kernels producing 256 feature maps
  - Pooling: 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

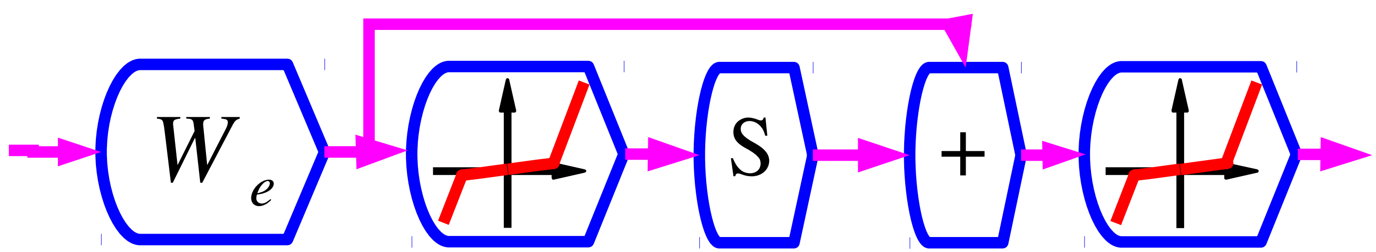
Single Stage System: $[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ **- log_reg**					
R/N/P	$R_{abs} - N - P_A$	$R_{abs} - P_A$	$N - P_M$	$N - P_A$	P_A
U^+	54.2%	50.0%	44.3%	18.5%	14.5%
R^+	54.8%	47.0%	38.0%	16.3%	14.3%
U	52.2%	43.3%(±1.6)	44.0%	17.2%	13.4%
R	53.3%	31.7%	32.1%	15.3%	12.1%(±2.2)
G	52.3%				
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%				

Using more ideas from biology

- **Pyramid Pooling**
  - Multi-scale pooling at the last stage



- **Threshold/Shrinkage Response Function + Lateral Inhibition Matrix**
  - Filter Bank - Shrinkage - Inhibition - Shrinkage



- **Discriminative term during pre-training (using label information)**
  - $$E(x,y,z,D,\theta) = \mathcal{C}(y, l(z,\theta)) + ||x - Dz||_2^2 + \lambda_1 ||z||_1$$
  -
  - [Mairal NIPS 09], [Boureau CVPR 10]

Using a few more tricks...

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

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

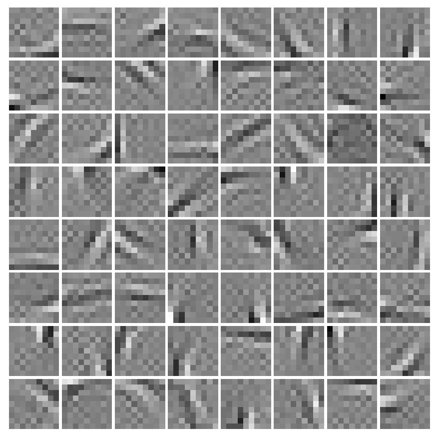
Latest Results and Analysis

- **Latest result on C-101: 70.8% correct**
  - Multi-scale pooling at the last layer (pyramid pooling)
  - Discriminative term in the sparse coding unsupervised learning
  - Different encoder architecture, with shrinkage function. And different sparse coding inference method (ISTA) [Gregor ICML 2010]
- **Second Stage + logistic regression = PMK_SVM**
- **Unsupervised pre-training doesn't help much :-(**
- **Random filters work amazingly well with normalization**
- **Supervised global refinement helps a bit**
- **The best system is really cheap**
- **Either use rectification and average pooling or no rectification and max pooling.**

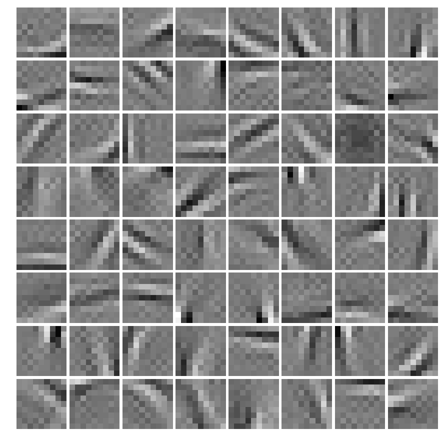
Multistage Hubel-Wiesel Architecture: Filters

- After PSD
- After supervised refinement

- Stage 1

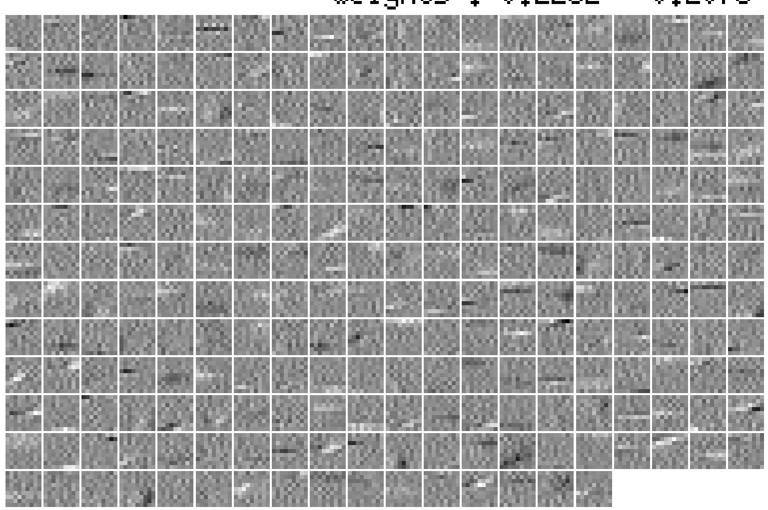


weights :-0.2232 - 0.2075

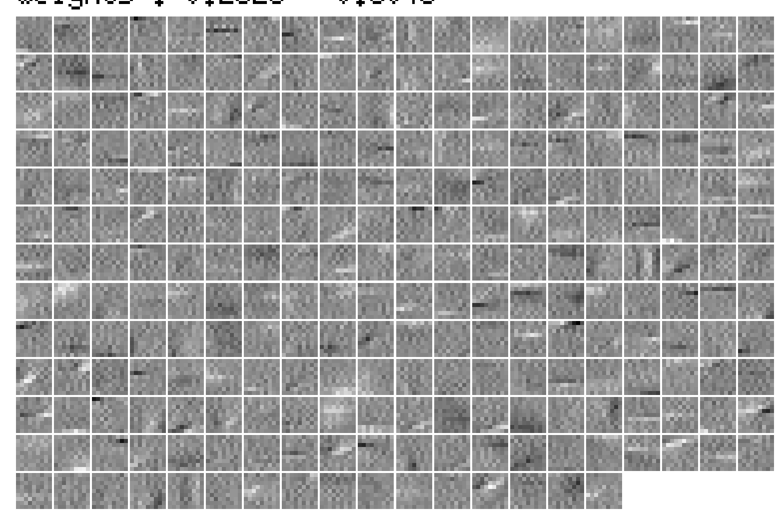


weights :-0.2828 - 0.3043

- Stage2

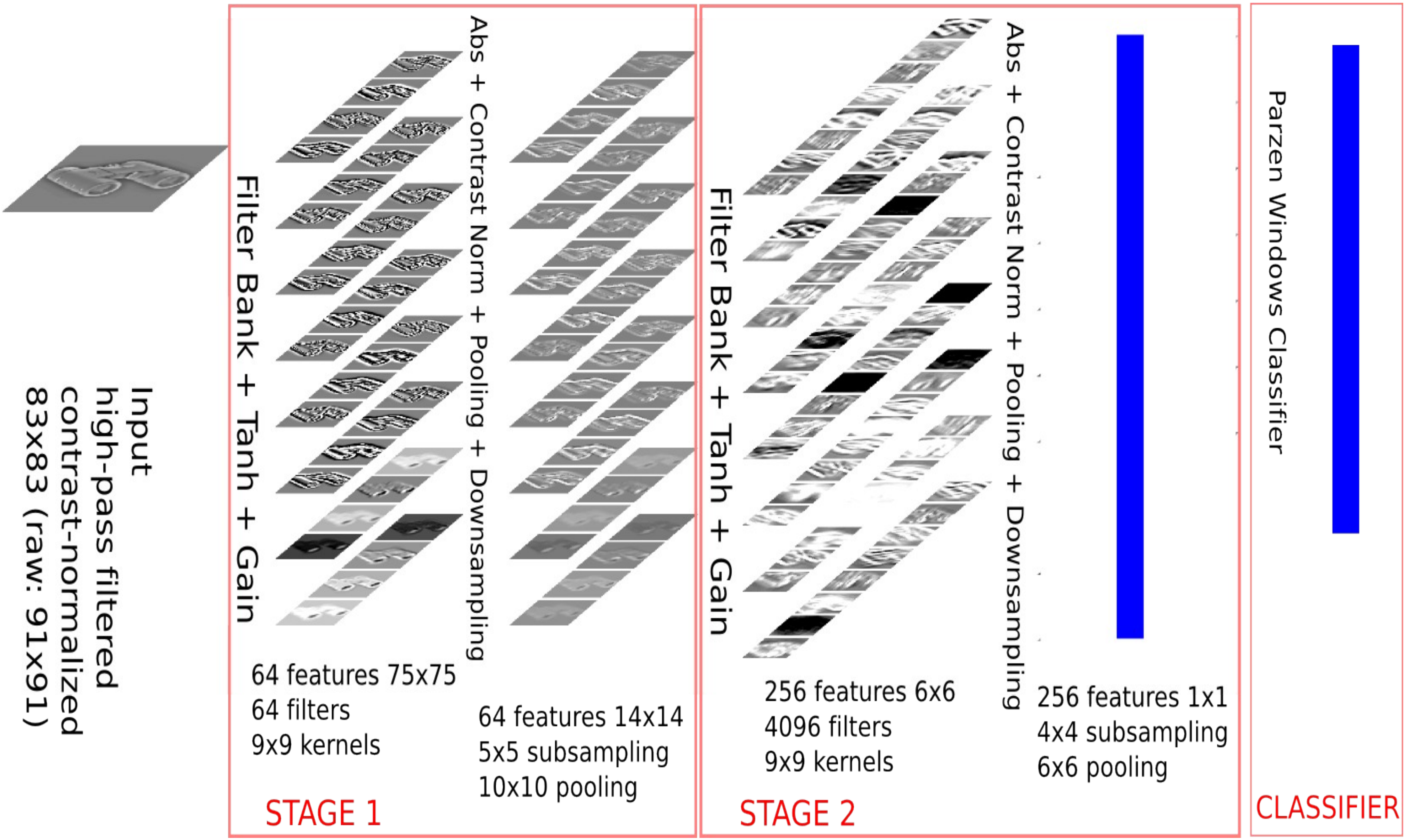


weights :-0.0778 - 0.064

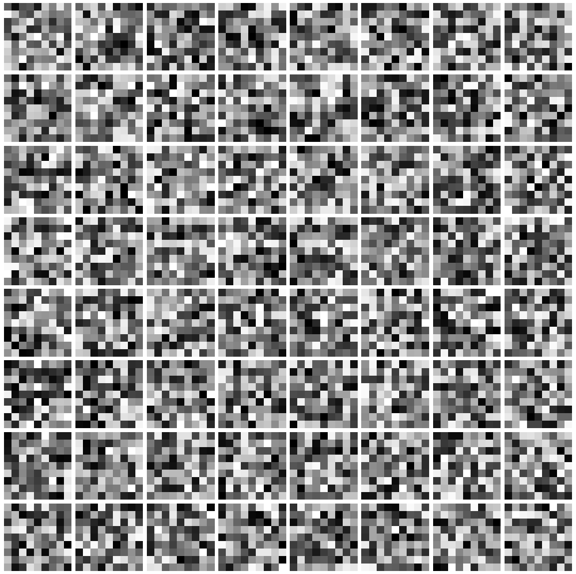


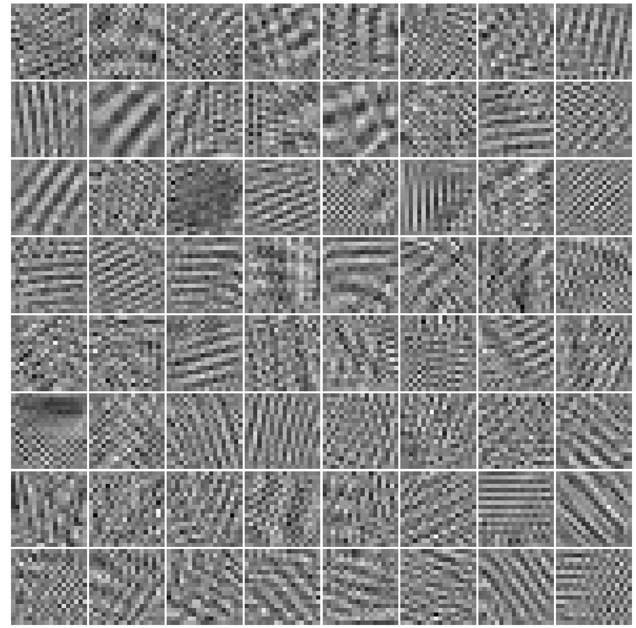
weights :-0.0929 - 0.0784

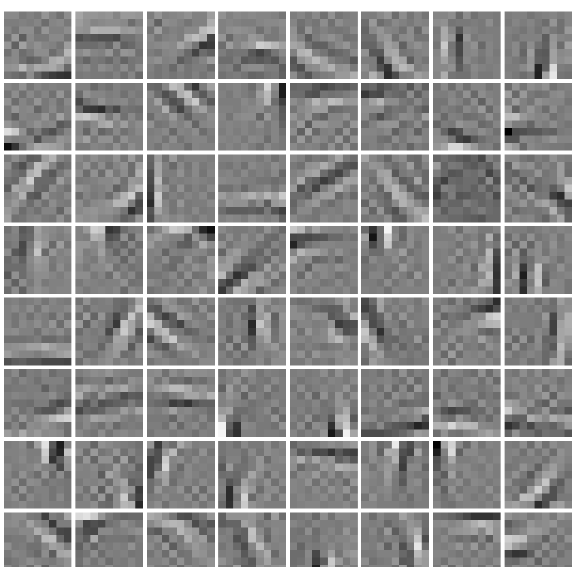
Demo: real-time learning of visual categories

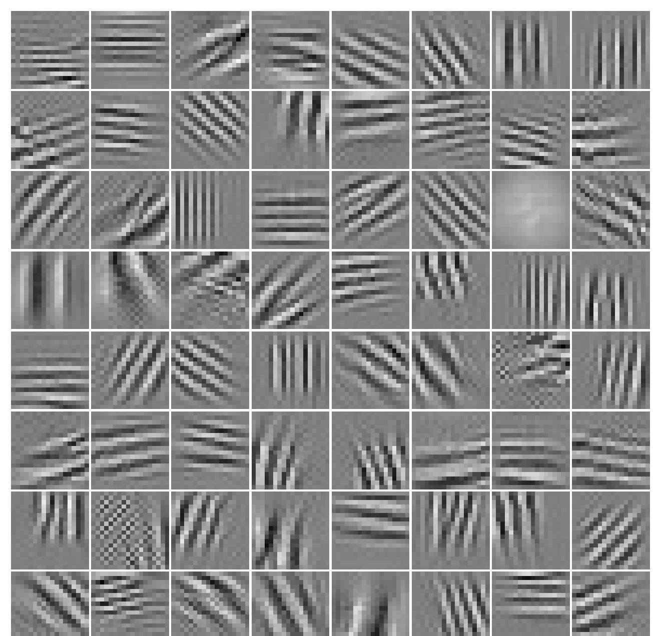


Why Random Filters Work?



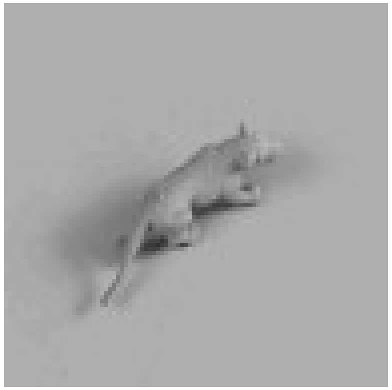






Small NORB dataset

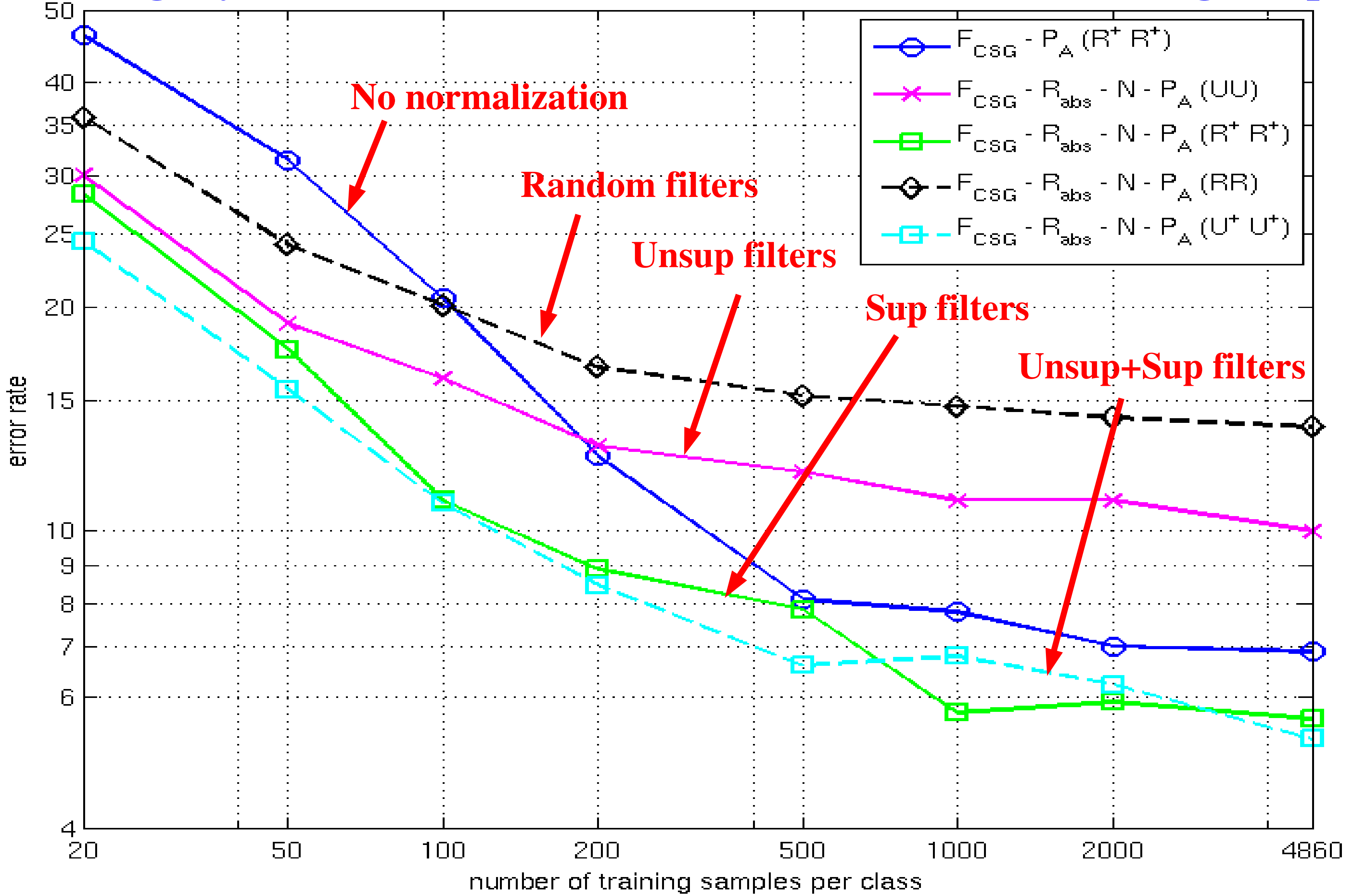
- 5 classes and up to 24,300 training samples per class



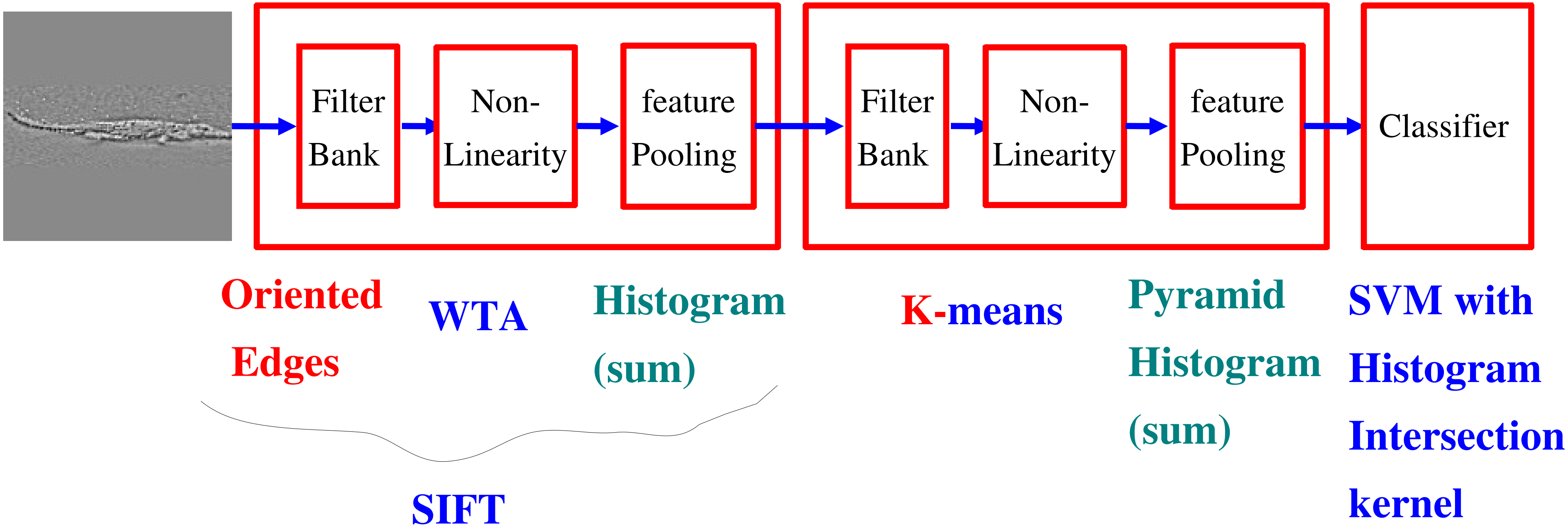



Small NORB dataset

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



ConvNets and "Conventional" Vision Architectures are Similar

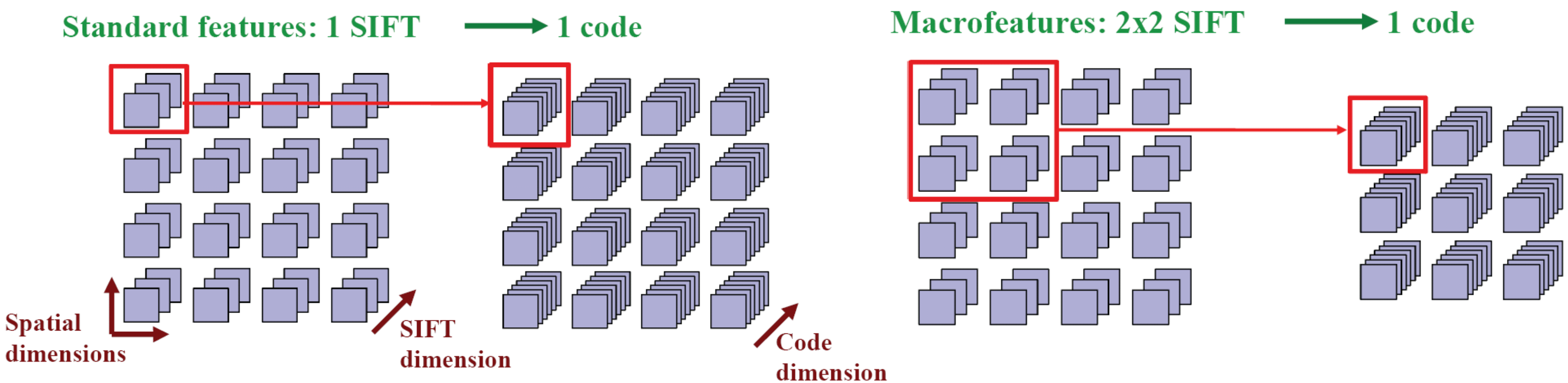


- Can't we use the same tricks as ConvNets to train the second stage of a "conventional vision architecture?
- Stage 1: SIFT
- Stage 2: discriminatibe sparse coding over neighborhoods + normalization + pooling

Using DL/ConvNet ideas in "conventional" recognition systems

[Boureau et al. CVPR 2010]

- **Adapting insights from ConvNets:**
  - Jointly encoding spatial neighborhoods instead of single points: increase spatial receptive fields for higher-level features



  - Use max pooling instead of average pooling
  - Train supervised dictionary for sparse coding
- **This yields state-of-the-art results:**
  - **75.7%** on Caltech-101 (+/-1.1%): record for single system
  - **85.6%** on 15-Scenes (+/- 0.2): record!

The Competition: SIFT + Sparse-Coding + PMK-SVM

- **Replacing K-means with Sparse Coding**
  - [Yang 2008] [Boureau, Bach, Ponce, LeCun 2010]

	Method	Caltech 15	Caltech 30	Scenes
Boiman et al. [1]	Nearest neighbor + spatial correspondence	65.00 ± 1.14	70.40	-
Jain et al. [8]	Fast image search for learned metrics	61.00	69.60	-
Lazebnik et al. [12]	Spatial Pyramid + hard quantization + kernel SVM	56.40	64.40 ± 0.80	81.40 ± 0.50
van Gemert et al. [24]	Spatial Pyramid + soft quantization + kernel SVM	–	64.14 ± 1.18	76.67 ± 0.39
Yang et al. [26]	SP + sparse codes + max pooling + linear	**67.00±0.45**	**73.2±0.54**	80.28 ± 0.93
Zhang et al. [27]	kNN-SVM	59.10 ± 0.60	66.20 ± 0.50	-
Zhou et al. [29]	SP + Gaussian mixture	–	–	**84.1 ± 0.5**
Baseline:	SP + hard quantization + avg pool + kernel SVM	56.74 ± 1.31	64.19 ± 0.94	80.89 ± 0.21
Unsupervised coding	SP + soft quantization + avg pool + kernel SVM	59.12 ± 1.51	66.42 ± 1.26	81.52 ± 0.54
1×1 features	SP + soft quantization + max pool + kernel SVM	63.61 ± 0.88	–	83.41 ± 0.57
8 pixel grid resolution	SP + sparse codes + avg pool + kernel SVM	62.85 ± 1.22	70.27 ± 1.29	83.15 ± 0.35
	SP + sparse codes + max pool + kernel SVM	64.62 ± 0.94	**71.81±0.96**	**84.25 ± 0.35**
	SP + sparse codes + max pool + linear	**64.71 ± 1.05**	71.52 ± 1.13	83.78 ± 0.53
Macrofeatures +	SP + sparse codes + max pool + kernel SVM	**69.03±1.17**	**75.72±1.06**	**84.60 ± 0.38**
Finer grid resolution	SP + sparse codes + max pool + linear	68.78 ± 1.09	75.14 ± 0.86	84.41 ± 0.26

The Competition: SIFT + Sparse-Coding + PMK-SVM

- **Splitting the Sparse Coding into Clusters**

[Boureau, et al. 2011]

p	1	4	16	64	1 + 4	1 + 16	1 + 64
			Caltech-101				
$k = 256$	70.5 ± 0.8	72.6 ± 1.0	74.0 ± 1.0	75.0 ± 0.8	72.5 ± 1.0	74.2 ± 1.1	75.6 ± 0.6
$k = 1024$	75.6 ± 0.9	76.0 ± 1.2	76.3 ± 1.1	76.2 ± 0.8	76.3 ± 1.2	$\mathbf{76.9 \pm 1.0}$	$\mathbf{77.3 \pm 0.6}$
			Scenes				
$k = 256$	78.8 ± 0.6	80.9 ± 0.7	81.5 ± 0.8	81.1 ± 0.5	80.8 ± 0.8	81.5 ± 0.8	81.9 ± 0.7
$k = 1024$	82.7 ± 0.7	83.0 ± 0.7	82.7 ± 0.9	81.4 ± 0.7	83.3 ± 0.8	83.3 ± 1.0	83.1 ± 0.7

Table 2. Results on Caltech 101 (30 training examples per class), and 15-Scenes (100 training examples per class) as a function of k: size of the codebook for sparse coding, and p: number of clusters extracted on the input data. Macrofeatures extracted every 4 pixels for Caltech, every 8 pixels for the Scenes

Convolutional Sparse Coding

[Kavukcuoglu et al. NIPS 2010]: convolutional PSD

[Zeiler, Krishnan, Taylor, Fergus, CVPR 2010]: Deconvolutional Network

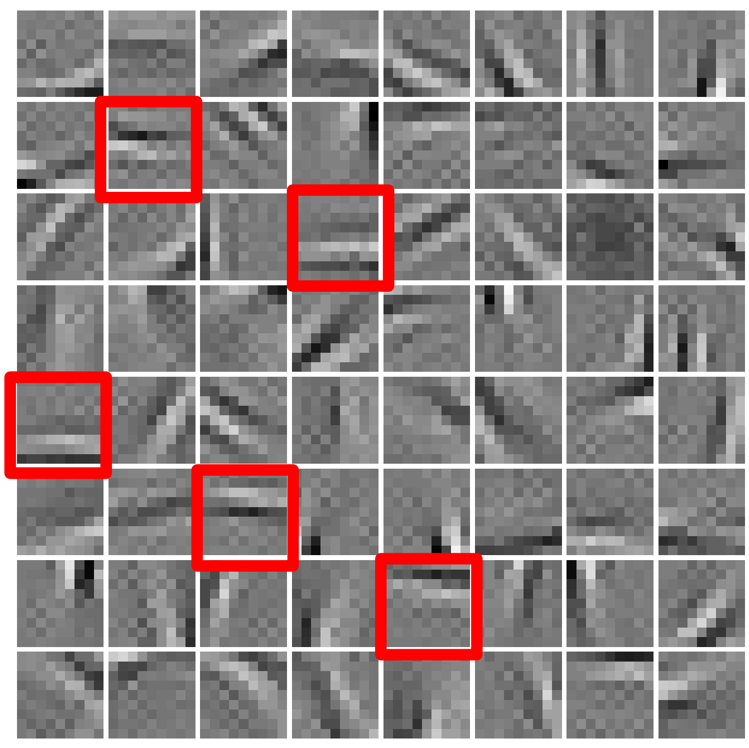
[Lee, Gross, Ranganath, Ng, ICML 2009]: Convolutional Boltzmann Machine

[Norouzi, Ranjbar, Mori, CVPR 2009]: Convolutional Boltzmann Machine

[Chen, Sapiro, Dunson, Carin, Preprint 2010]: Deconvolutional Network with automatic adjustment of code dimension.

Convolutional Training

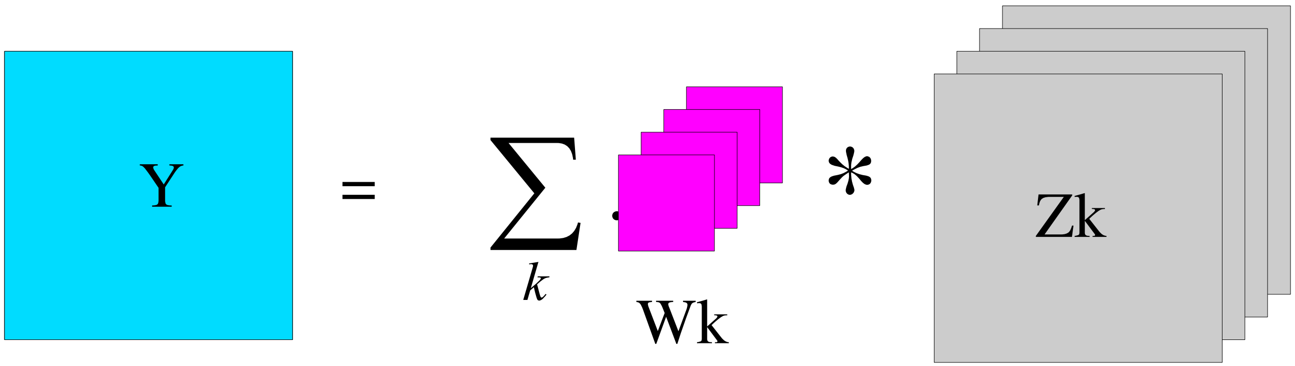
- **Problem:**
  - With patch-level training, the learning algorithm must reconstruct the entire patch with a single feature vector
  - But when the filters are used convolutionally, neighboring feature vectors will be highly redundant



Patch-level training produces lots of filters that are shifted versions of each other.

Convolutional Sparse Coding

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



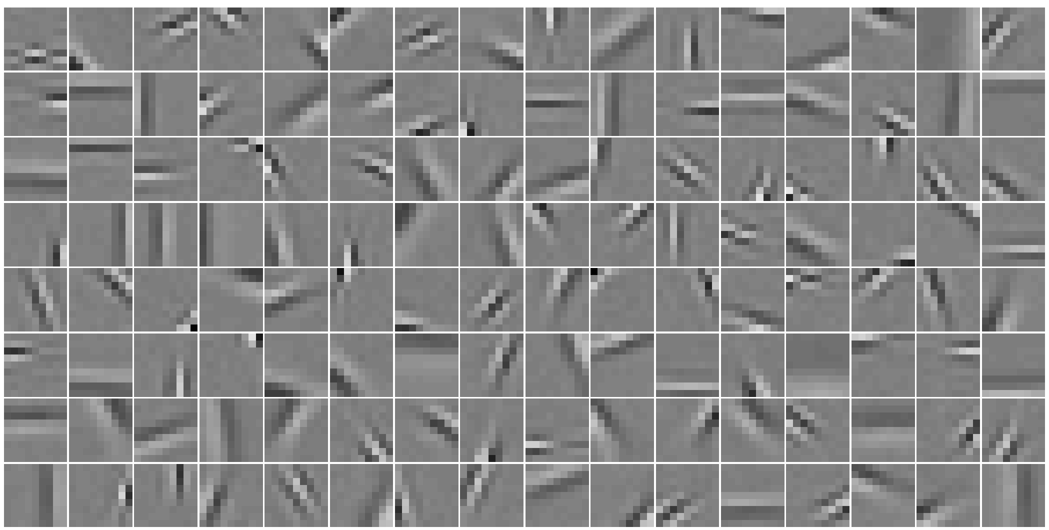
“deconvolutional networks” [Zeiler, Taylor, Fergus CVPR 2010]

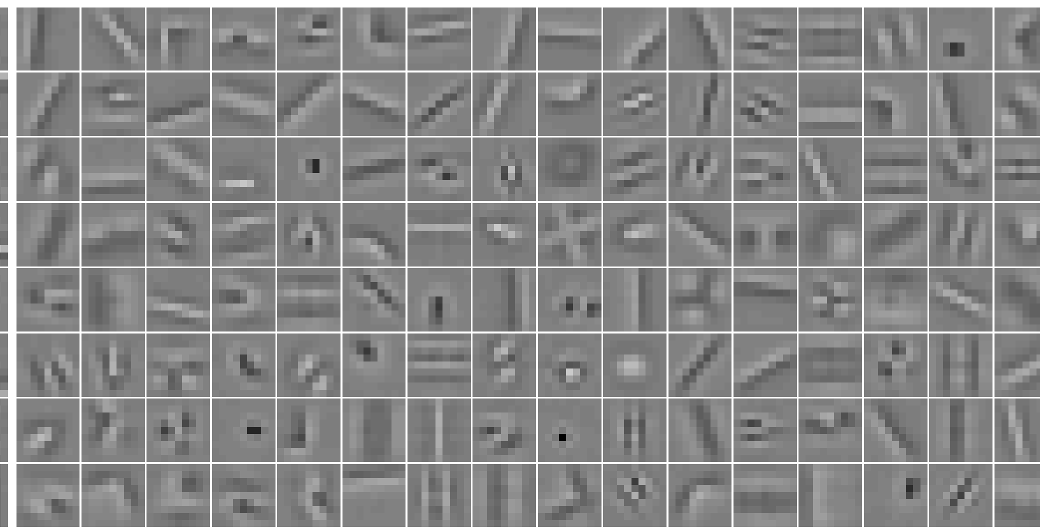
Convolutional PSD: Encoder with a soft sh() Function

Convolutional Formulation

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

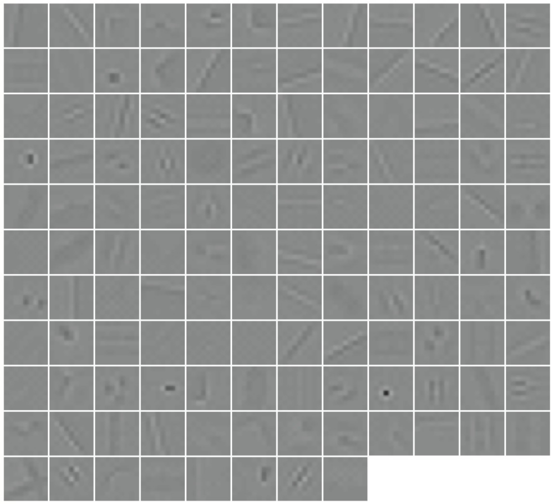
$$\mathcal{L}(x,z,\mathcal{D}) = \frac{1}{2}||x - \sum_{k=1}^{K} \mathcal{D}_k * z_k||_2^2 + \sum_{k=1}^{K} ||z_k - f(W^k * x)||_2^2 + |z|_1$$





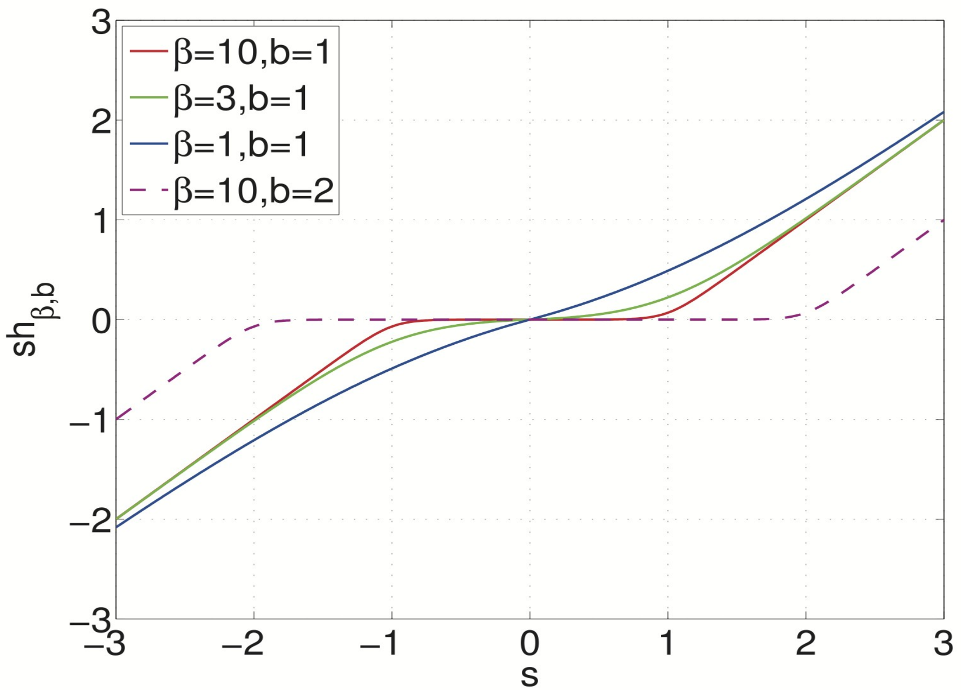
- ***PATCH*** based learning
- ***CONVOLUTIONAL*** learning

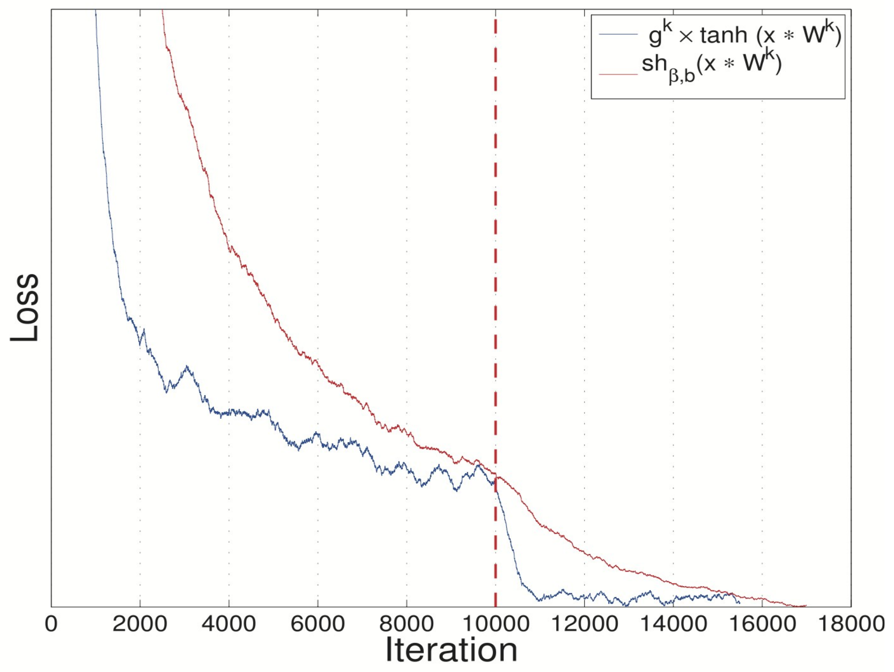
Convolutional PSD



Convolutional Formulation

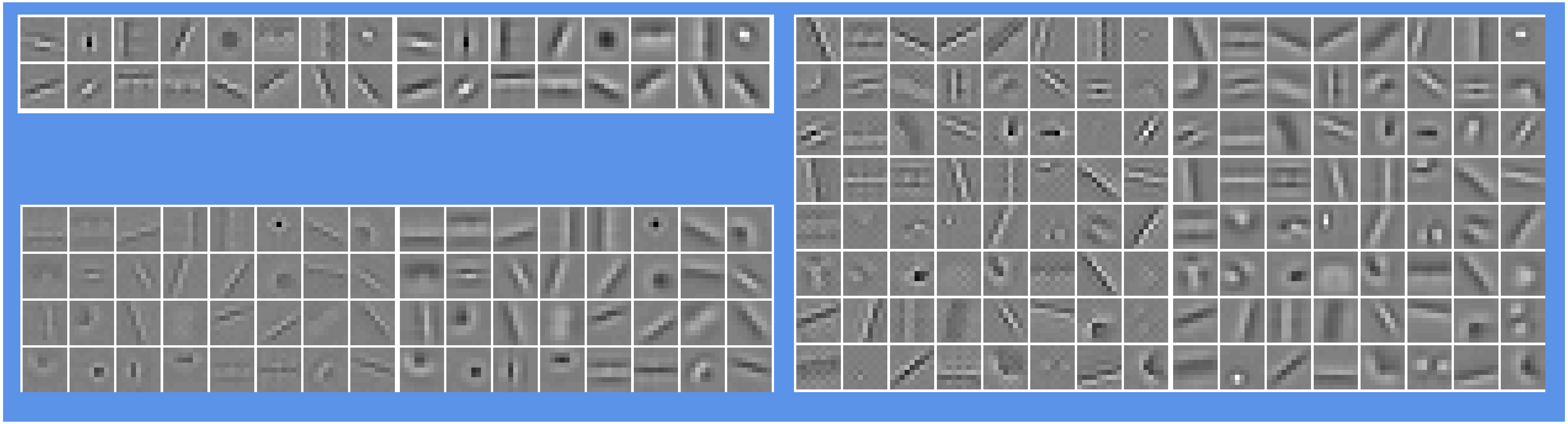
- Efficient Training using 2nd order derivative approximation
- Especially important for training an encoder function
- Encoder with smooth shrinkage non-linearity





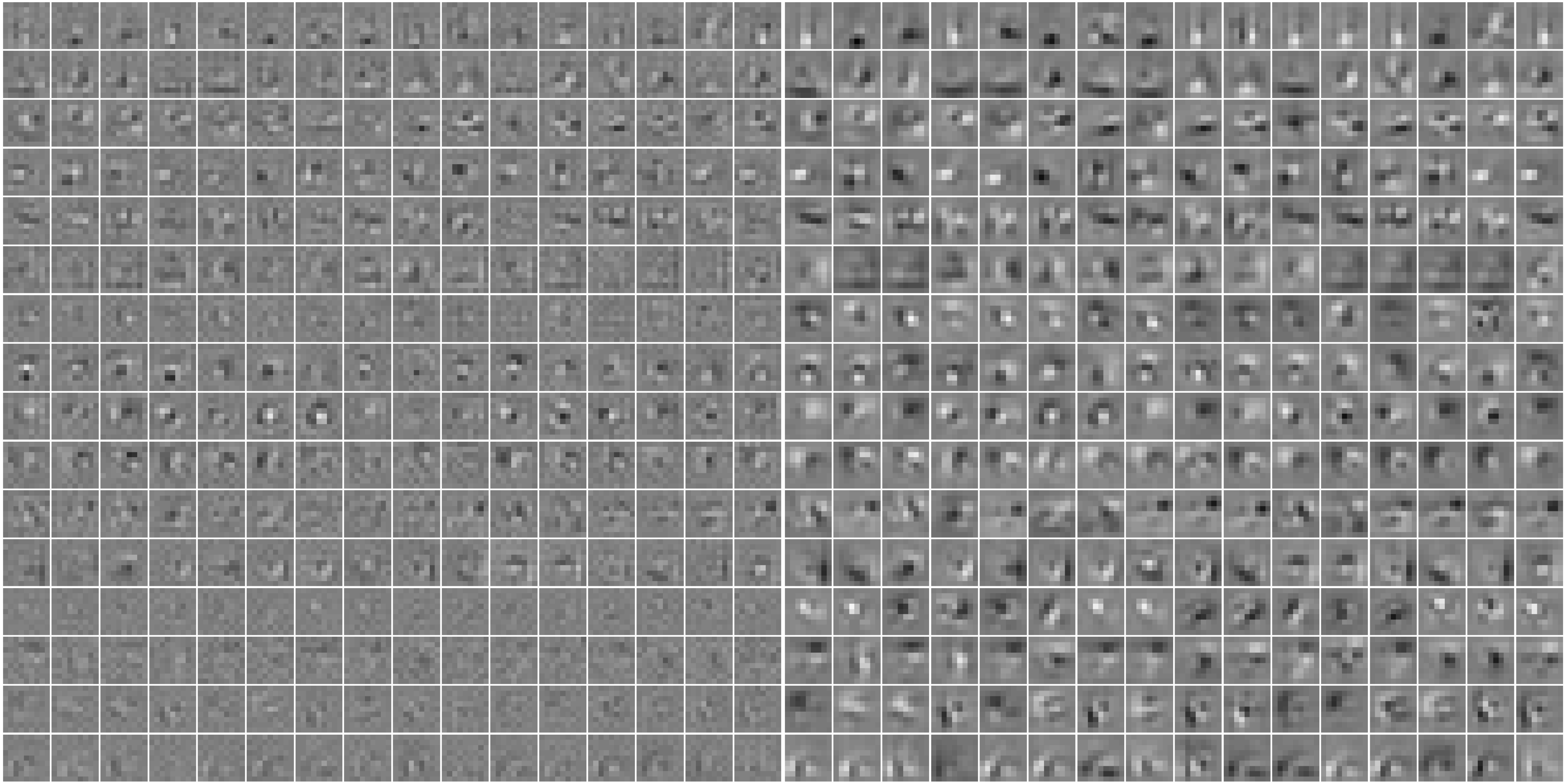
Convolutional Training

- **Filters and Basis Functions obtained with 16, 32, and 64 filters.**
  - Smooth shrinkage encoder, coordinate gradient descent inference



Convolutional PSD: Second Stage

- Second Stage Filters (encoder) and Basis Functions (decoder)



- ***ENCODER***
- ***DECODER***

Convolutional PSD: training the filters of a ConvNet

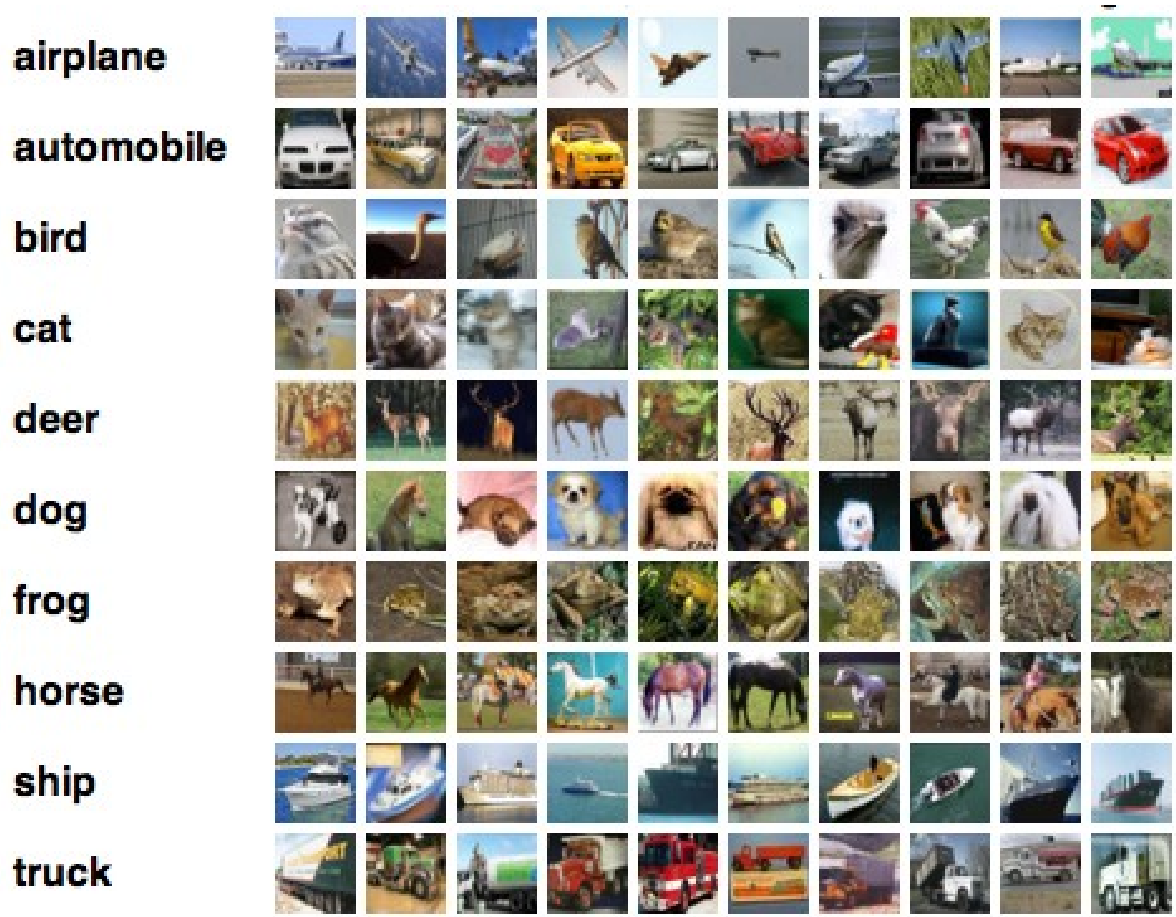
Performance on Caltech-101

- Significant Improvement on 1st layer from ***Patch*** to ***Convolutional***
- 1st layer is closest to input and convolutional training is most effective

	Patch	Convolutional
1 stage, unusp: U	52.2%	57.1%
1 stage, unsup+sup: U^+	54.2%	57.6%
2 stages, unsup: UU	63.7%	65.3%
2 stages, unsup+sup: U^+U^+	65.5%	66.3%

Cifar-10 Dataset

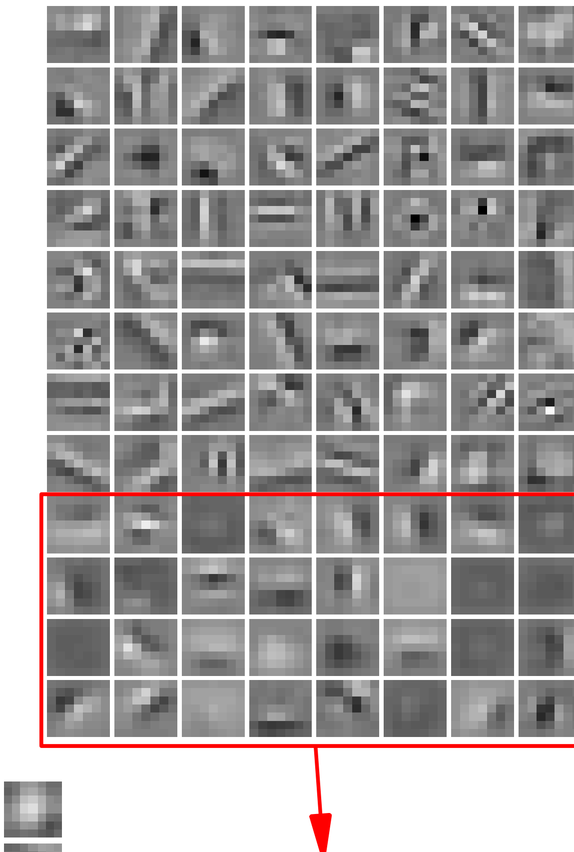
- **Dataset of tiny images**
  - Images are 32x32 color images
  - 10 object categories with 50000 training and 10000 testing
- **Example Images**



Architecture of Network

First Stage:

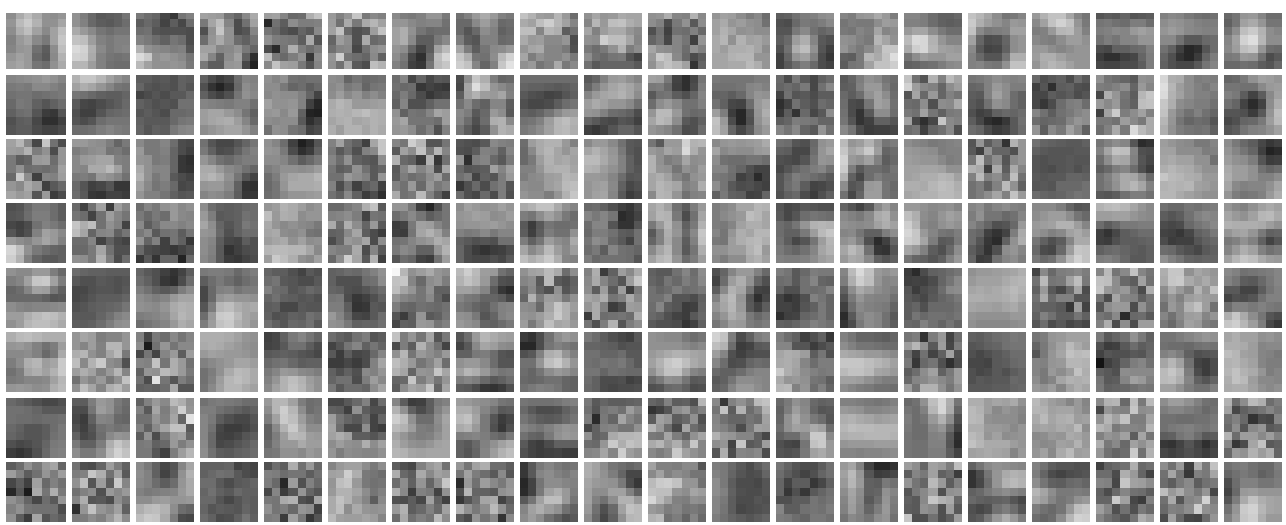
- Filters Y: 96 7x7 kernels: 64 Y, 16 U, and 16 V
- Pooling: 4x4 averaging with 2x2 subsample
- Features: 64 feature maps size 12x12
- Filters are learned <u>convolutionally</u> with DPSD



Chrominance Filters
(Cr, Cb)

Second Stage:

- Filters: 2048 7x7 kernels
- Pooling: 3x3 averaging no downsampling
- Features: 128 feature maps of size 4x4
- Filters are learned <u>convolutionally</u> with DPSD



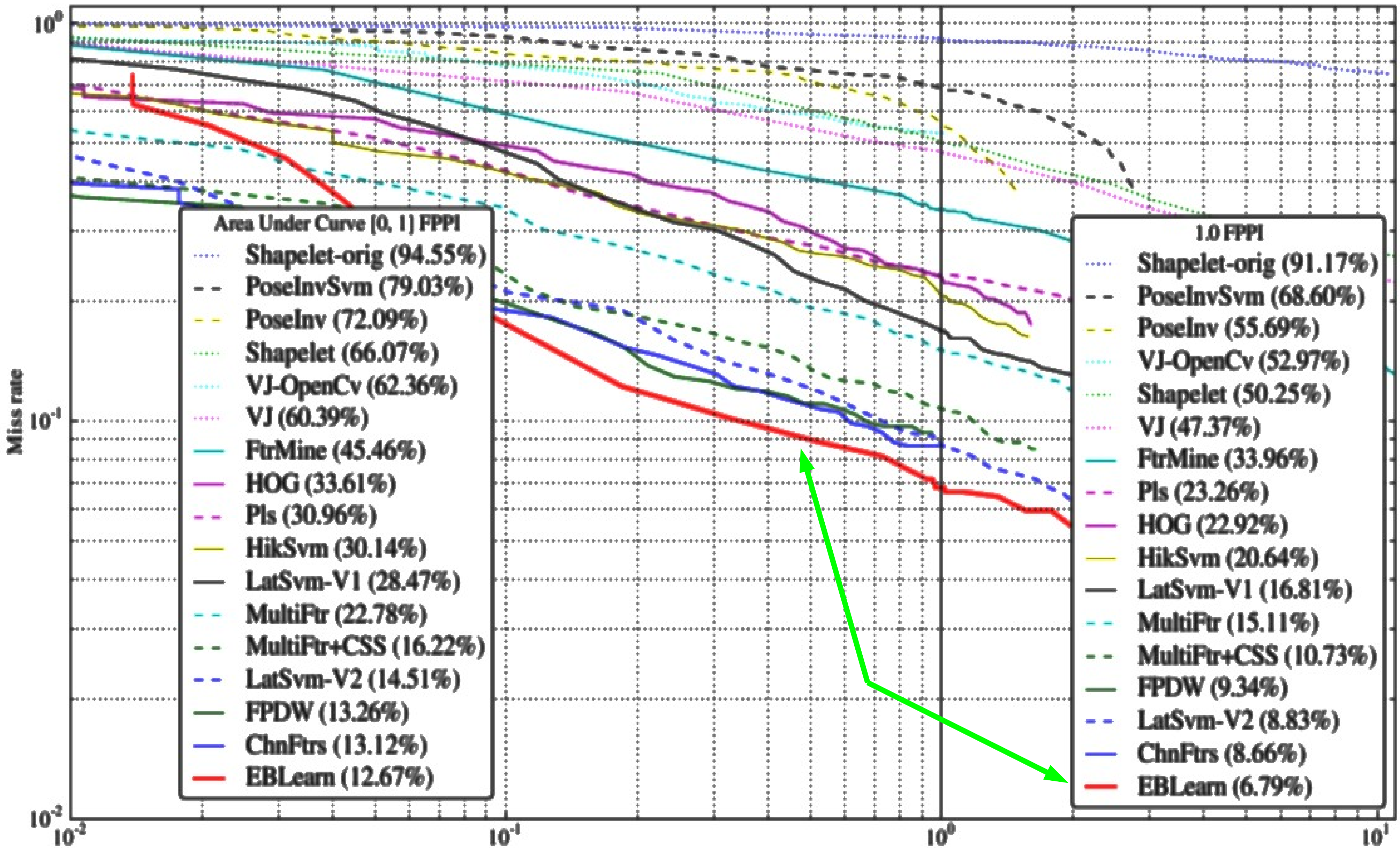
Comparative Results on Cifar-10 Dataset

10000 **linear combinations [25]**		36.0%
10k **GRBM, 1 layer with fine-tuning [10]**		64.8%
mcRBM-DBN(11025-8192-8192) [25]		71.0%
PCA(512)-iLCC(4096)-SVM [30]		**74.5%**
Architecture	**Protocol**	%
(1) $F_{si} - R_{abs} - P_M - N$	$\mathbf{RR}$	47.5%
(2) $F_{tanh} - R_{abs} - P_M - N$	$\mathbf{R^+R^+}$	70.0%
(3) $F_{si} - R_{abs} - P_M - N$	$\mathbf{R^+R^+}$	70.5%
(4) $F_{si} - R_{abs} - P_M - N$	$\mathbf{D_c^+}$	59.6%
(5) $F_{si} - R_{abs} - N - P_A$	$\mathbf{D_c^+}$	60.0%
(6) $F_{tanh} - R_{abs} - P_M - N$	$\mathbf{U_c^+U^+}$	74.7%
(7) $F_{si} - R_{abs} - P_M - N$	$\mathbf{U_c^+U^+}$	74.8%
(8) $F_{si} - R_{abs} - P_M - N$	$\mathbf{D^+D^+}$	74.4%
(9) $F_{si} - R_{abs} - N - P_A$	$\mathbf{D_c^+D^+}$	75.0%
(10) $F_{si} - R_{abs} - P_M - N$	$\mathbf{D_c^+D^+}$	**77.6%**

* Krizhevsky. Learning multiple layers of features from tiny images. Masters thesis, Dept of CS U of Toronto

**Ranzato and Hinton. Modeling pixel means and covariances using a factorized third order boltzmann machine. CVPR 2010

Pedestrian Detection (INRIA Dataset)



[Kavukcuoglu et al. NIPS 2010]

Pedestrian Detection: Examples

Yann LeCun

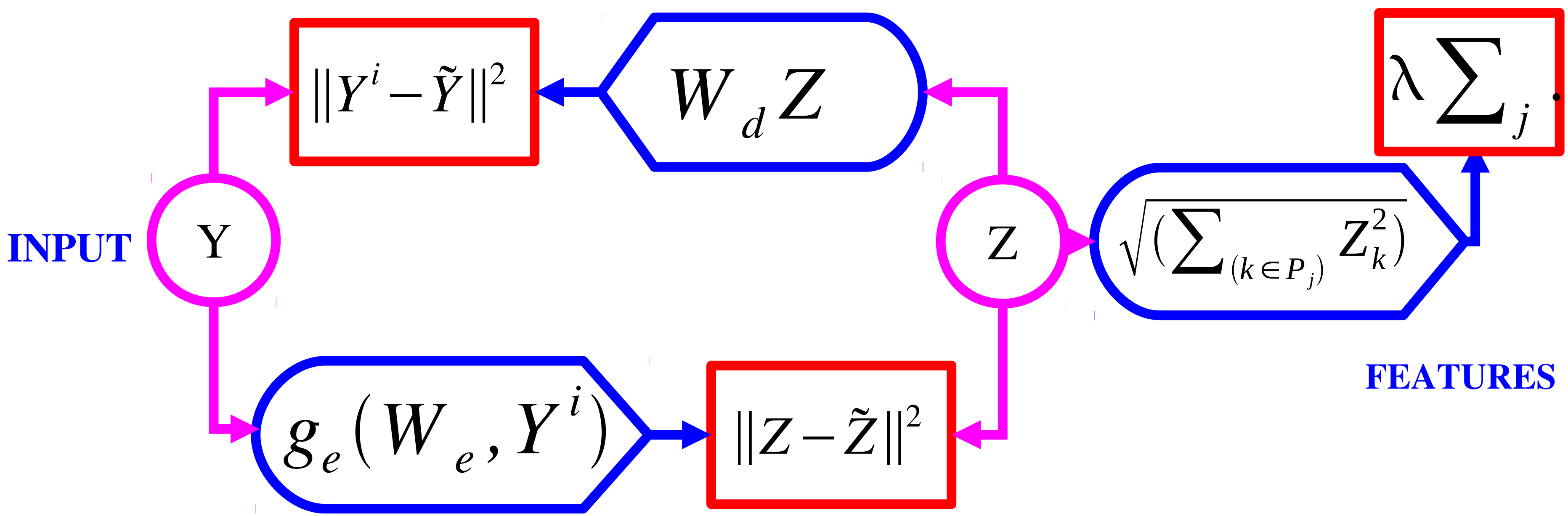
[Kavukcuoglu et al. NIPS 2010]

Learning Complex Cells with Invariance Properties Using Group Sparsity

[Kavukcuoglu et al. CVPR 2008]

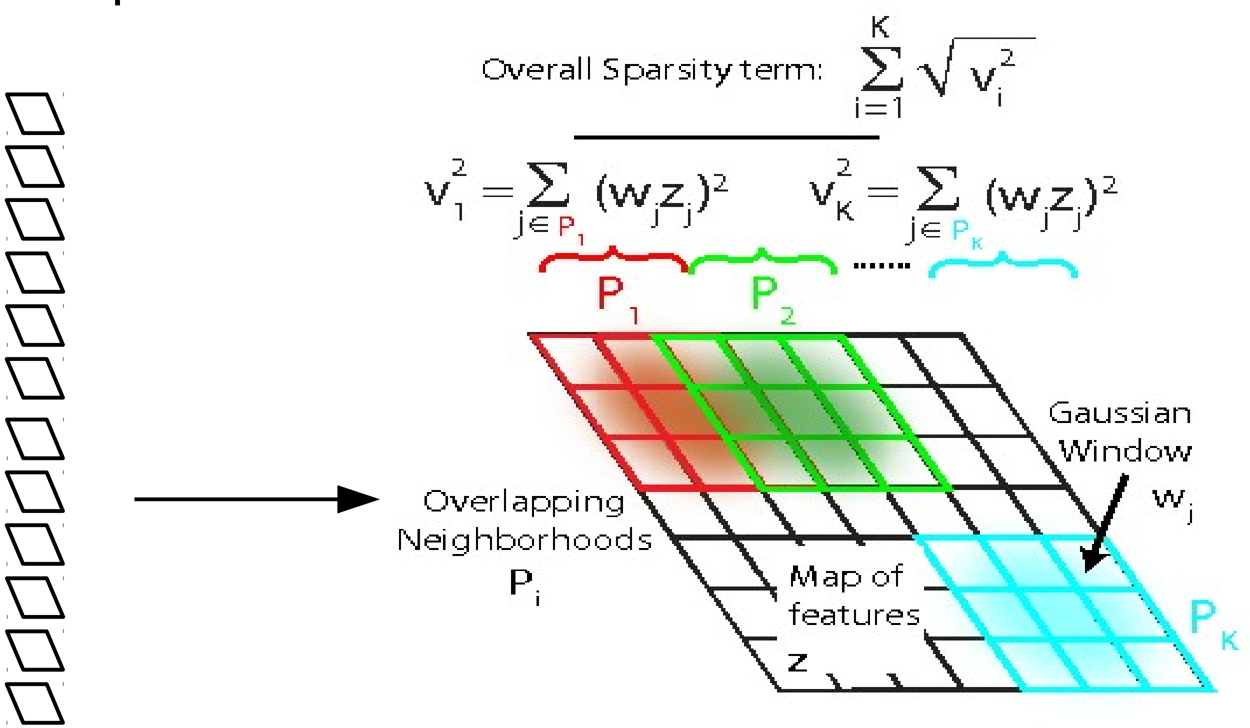
Learning Invariant Features [Kavukcuoglu et al. CVPR 2009]

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



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

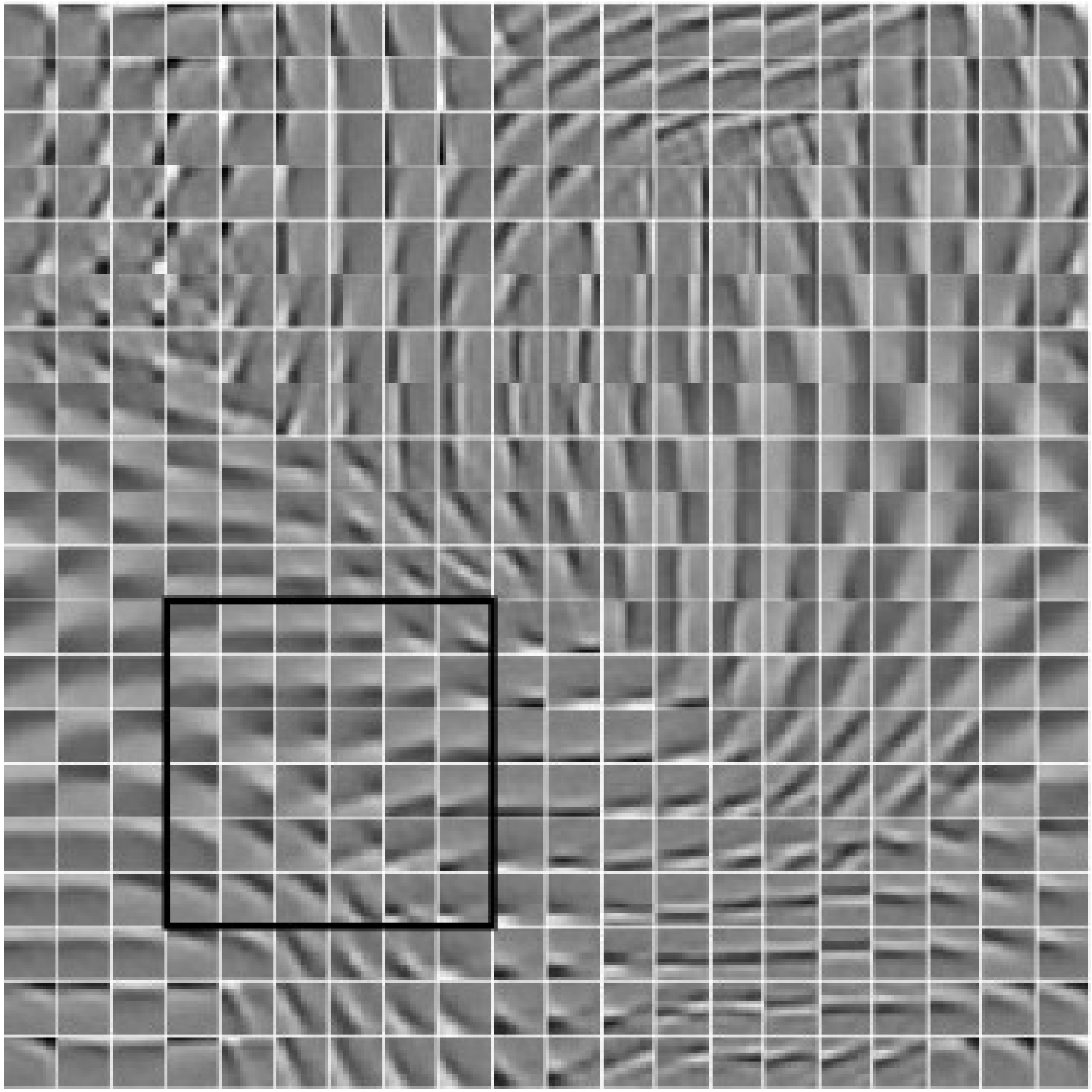


Units in the code Z

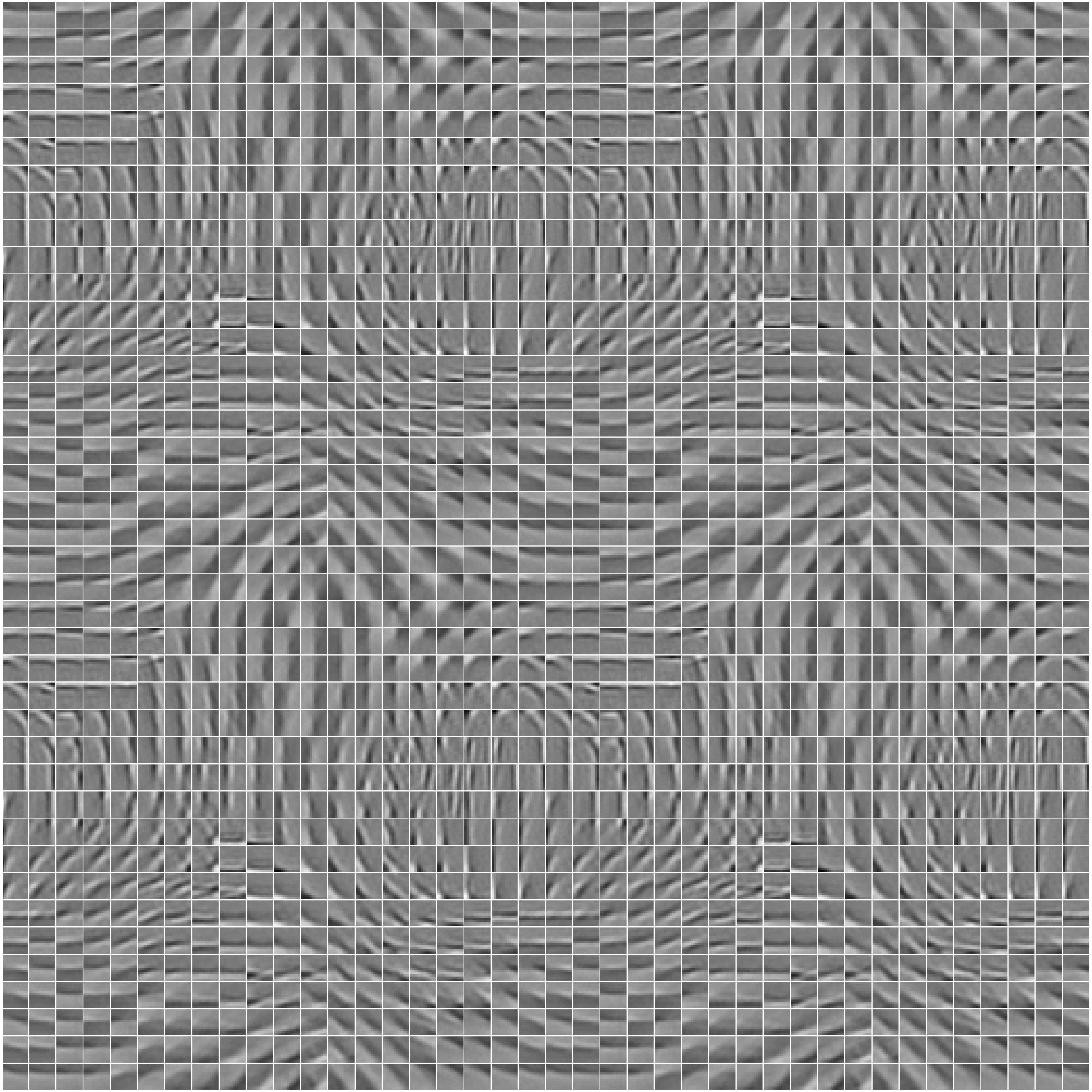
Define pools and enforce sparsity across pools

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.

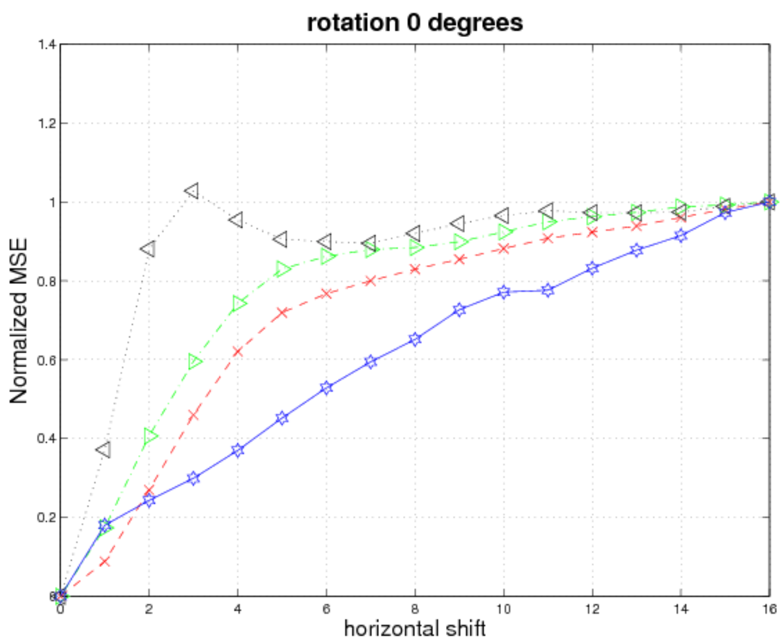


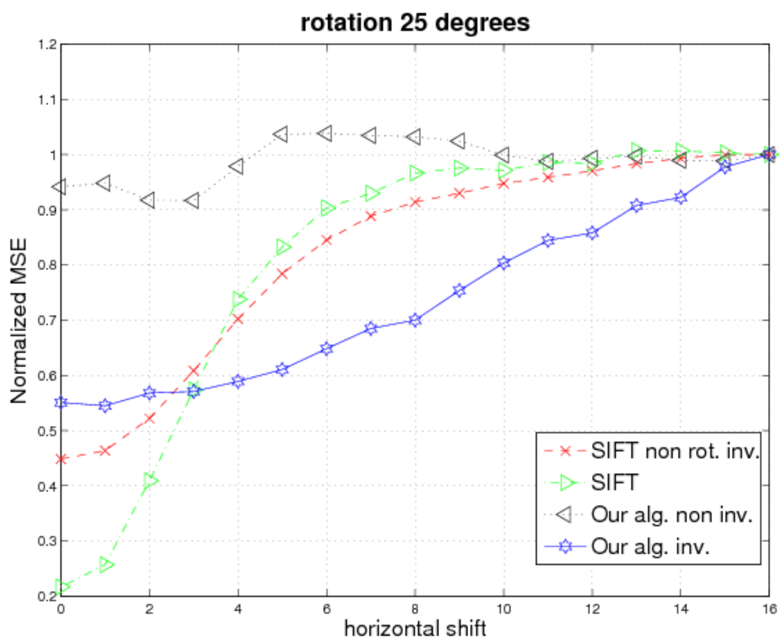
Pinwheels?



Invariance Properties Compared to SIFT

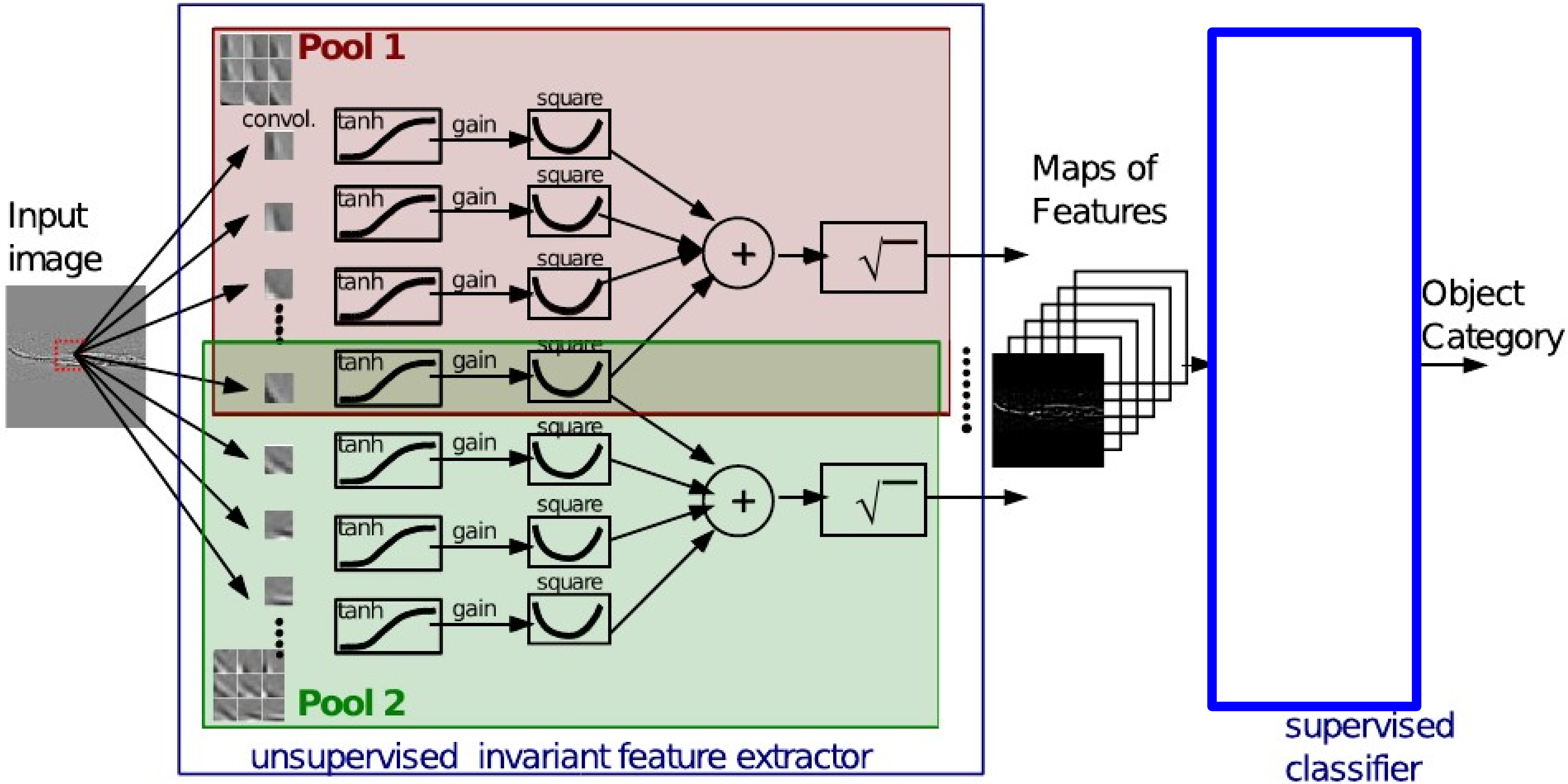
- **Measure distance between feature vectors (128 dimensions) of 16x16 patches from natural images**
  - Left: normalized distance as a function of translation
  - Right: normalized distance as a function of translation when one patch is rotated 25 degrees.
- **Topographic PSD features are more invariant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

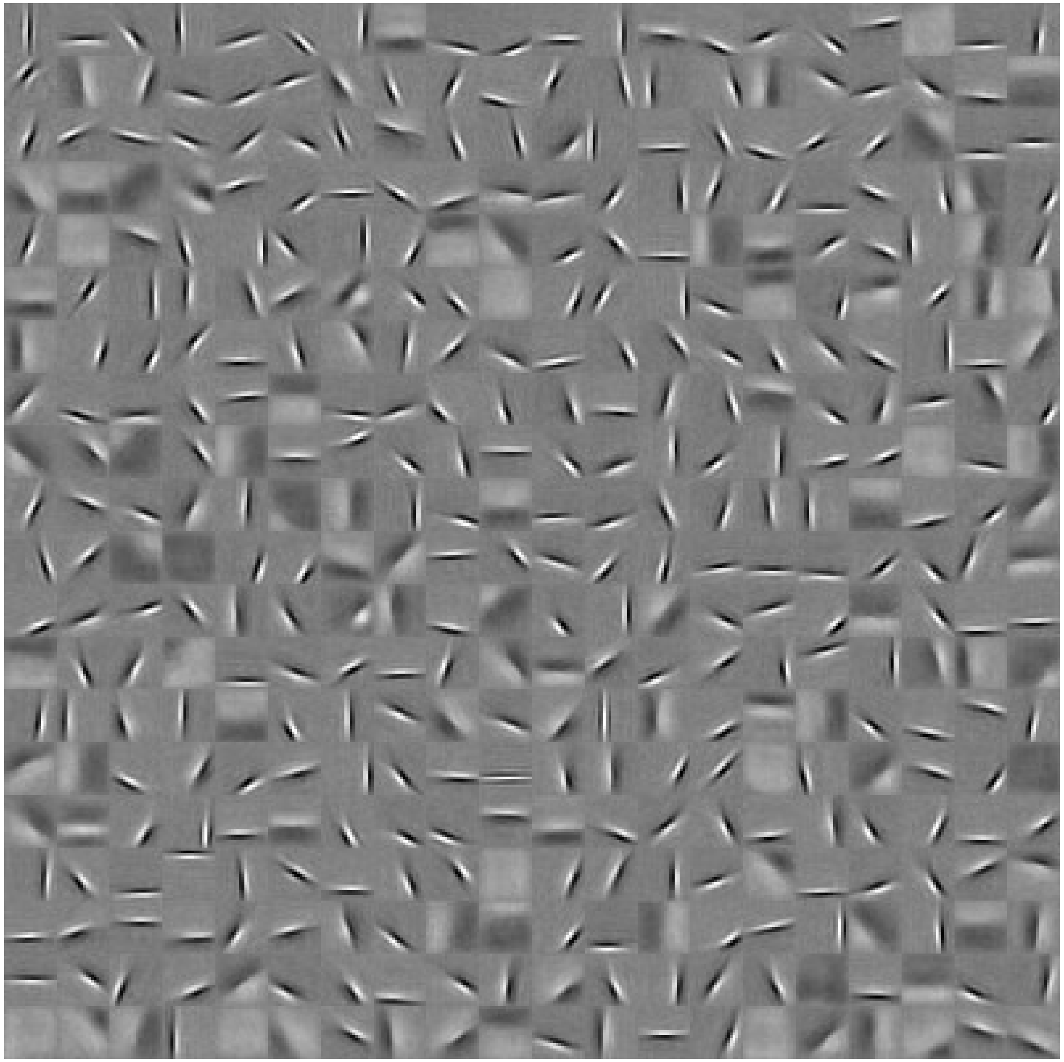
Performance on MNIST Dataset	
Method	**Error Rate (%)**
IPSD (5x5)	**1.0**
SIFT (5x5) (non rot. inv.)	**1.5**

Learning fields of Simple Cells and Complex Cells

[Gregor and LeCun, arXiv.org 2010]

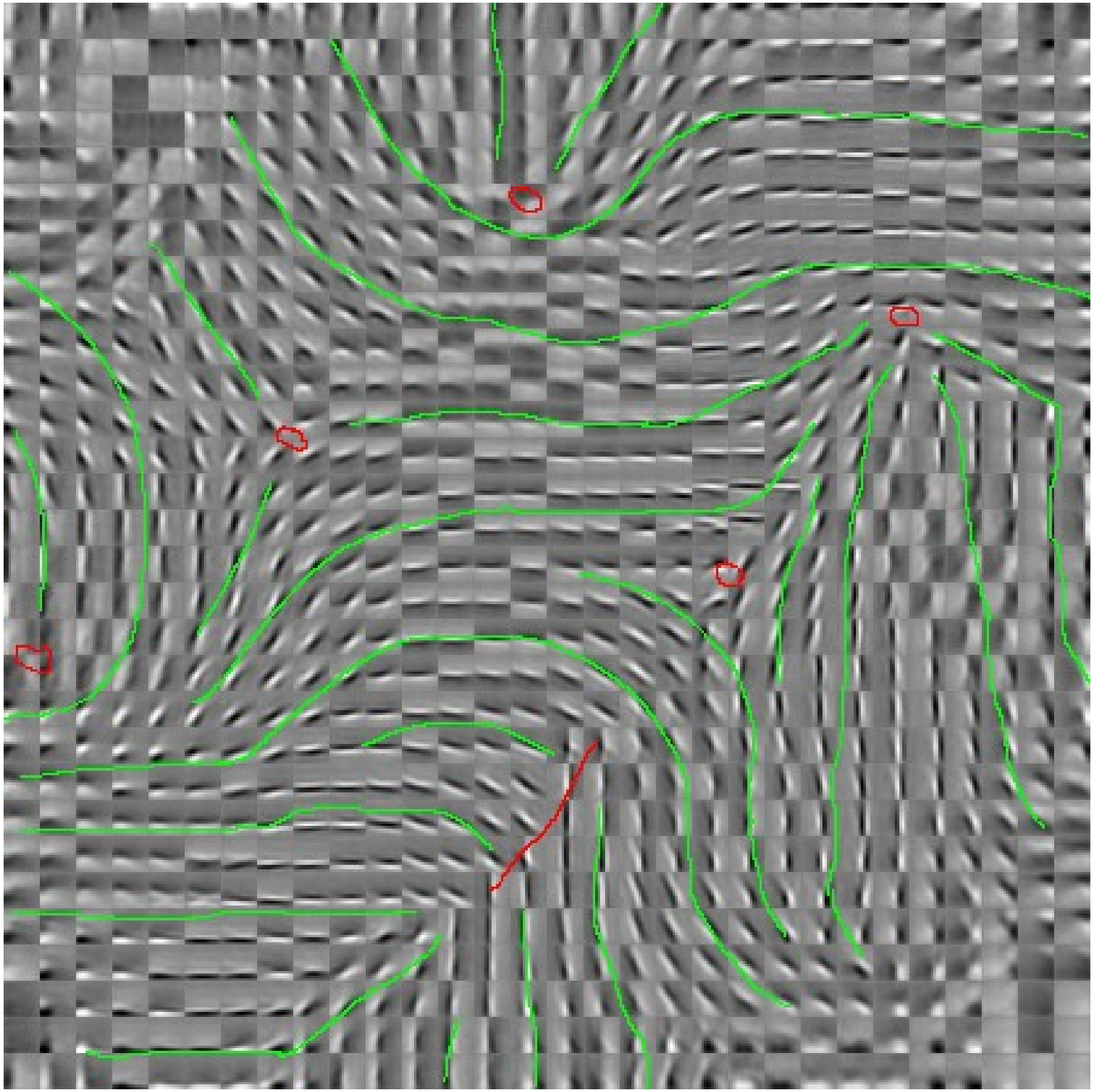
Training Simple Cells with Local Receptive Fields over Large Input Images

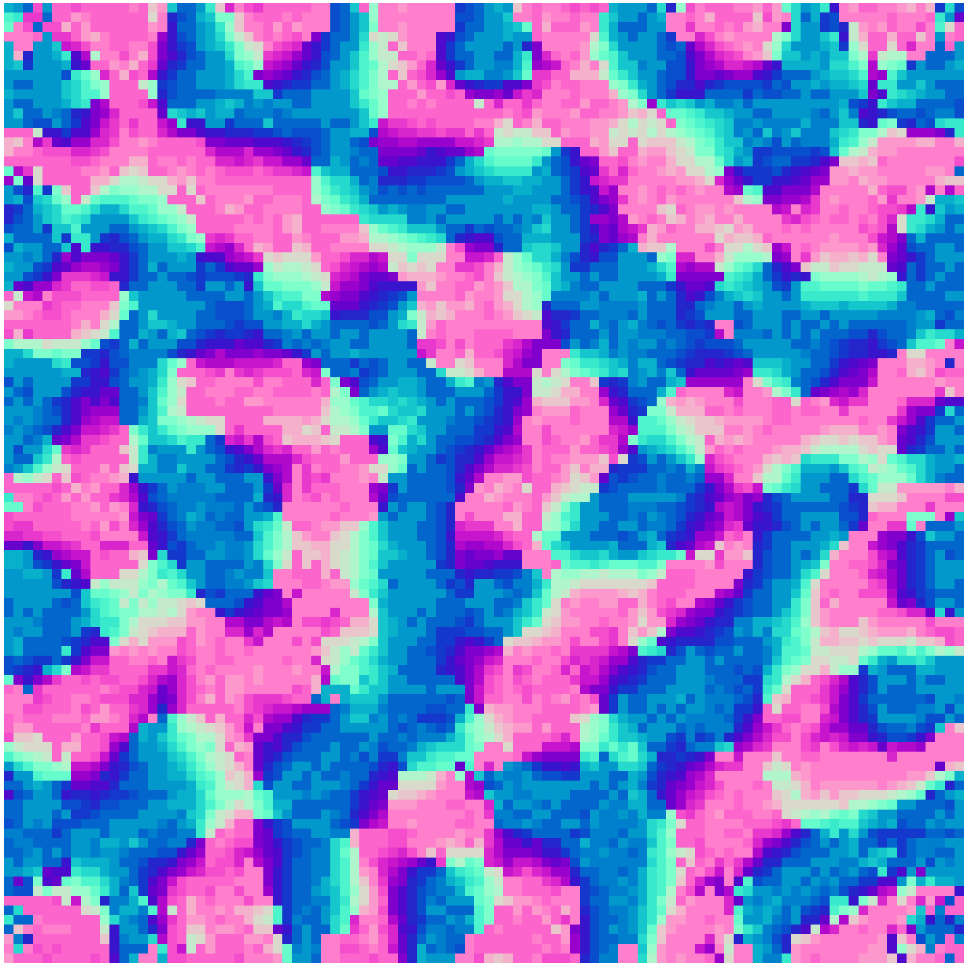
- Training on 115x115 images. Kernels are 15x15



Simple Cells + Complex Cells with Sparsity Penalty: Pinwheels

- Training on 115x115 images. Kernels are 15x15





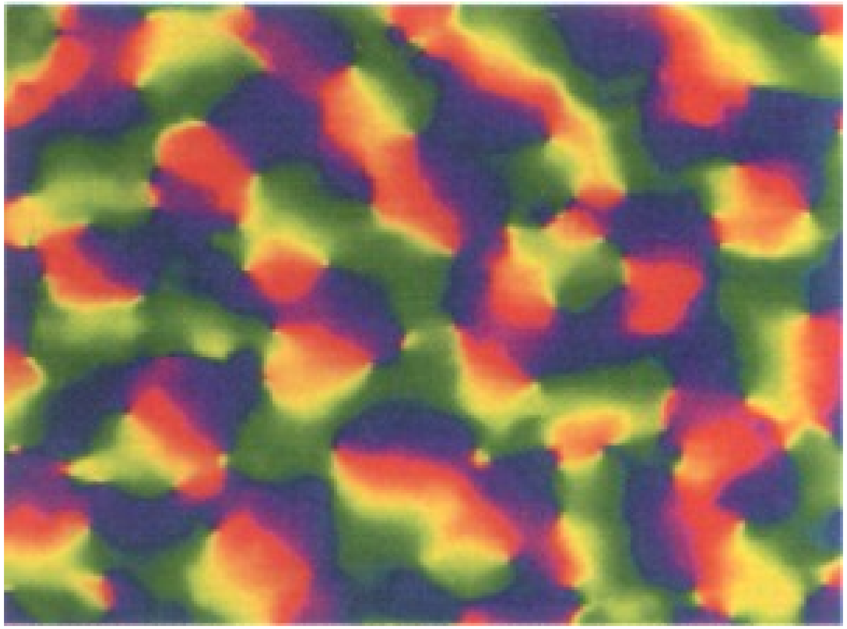
119x119 Image Input

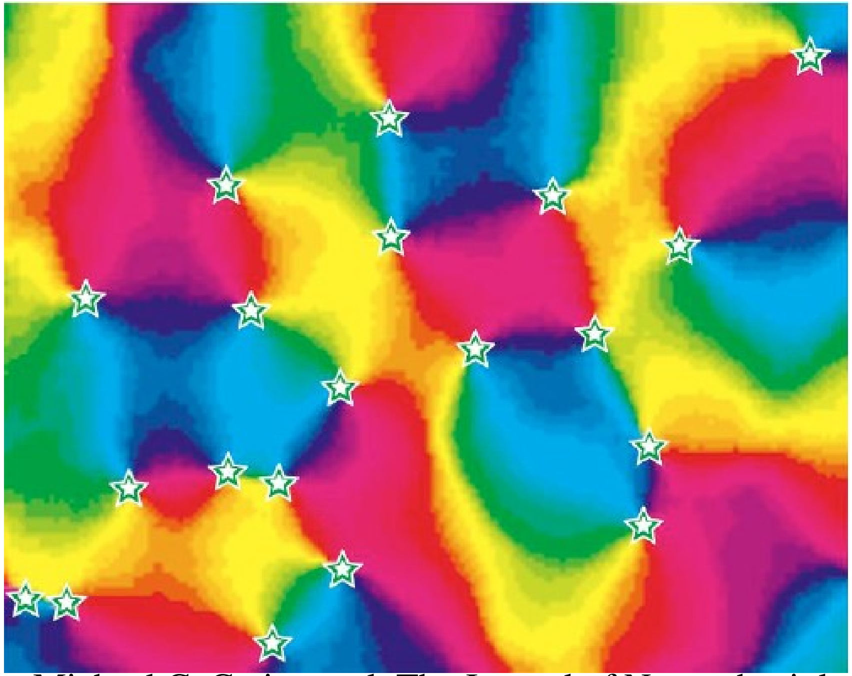
100x100 Code

20x20 Receptive field size

sigma=5

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

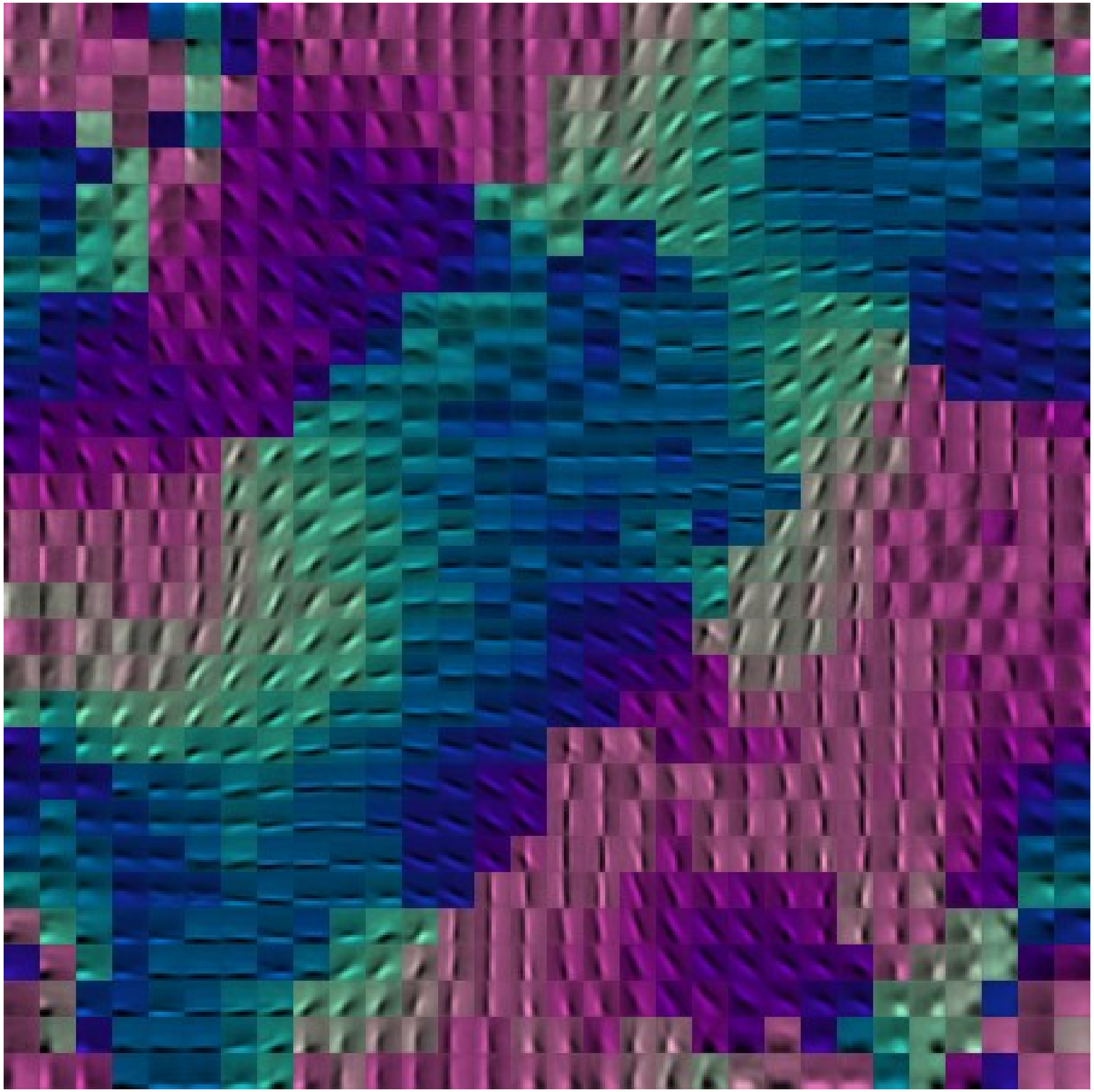




Michael C. Crair, et. al. The Journal of Neurophysiology Vol. 77 No. 6 June 1997, pp. 3381-3385 (**Cat**)

Same Method, withTraining at the Image Level (vs patch)

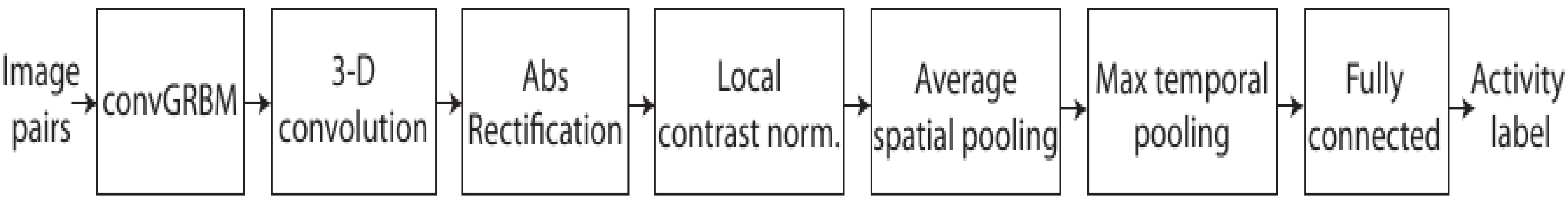
- Color indicates orientation (by fitting Gabors)



Recognizing Activities In Videos

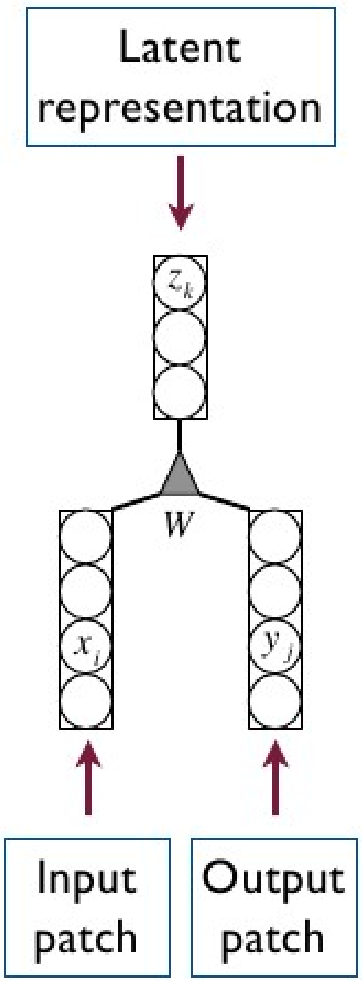
[Taylor, Fergus, LeCun, Bregler ECCV 2010]

Architecture



- **Convolutional Gated RBM: takes two successive frames as input and automatically learns motion features**
  - Feature encode the transformation from the first frame to the second frame
  - Trained in unsupervised mode to predict the second frame
- **The rest is a 3D (spatio-temporal) convolutional network**
  - Trained in supervised mode with sparse coding

Gated RBM



$$E(\mathbf{y}, \mathbf{z}; \mathbf{x}) = -\sum_{ijk} W_{ijk} x_i y_j z_k$$

$$p(\mathbf{y}, \mathbf{z} | \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(-E(\mathbf{y}, \mathbf{z}; \mathbf{x})\right)$$

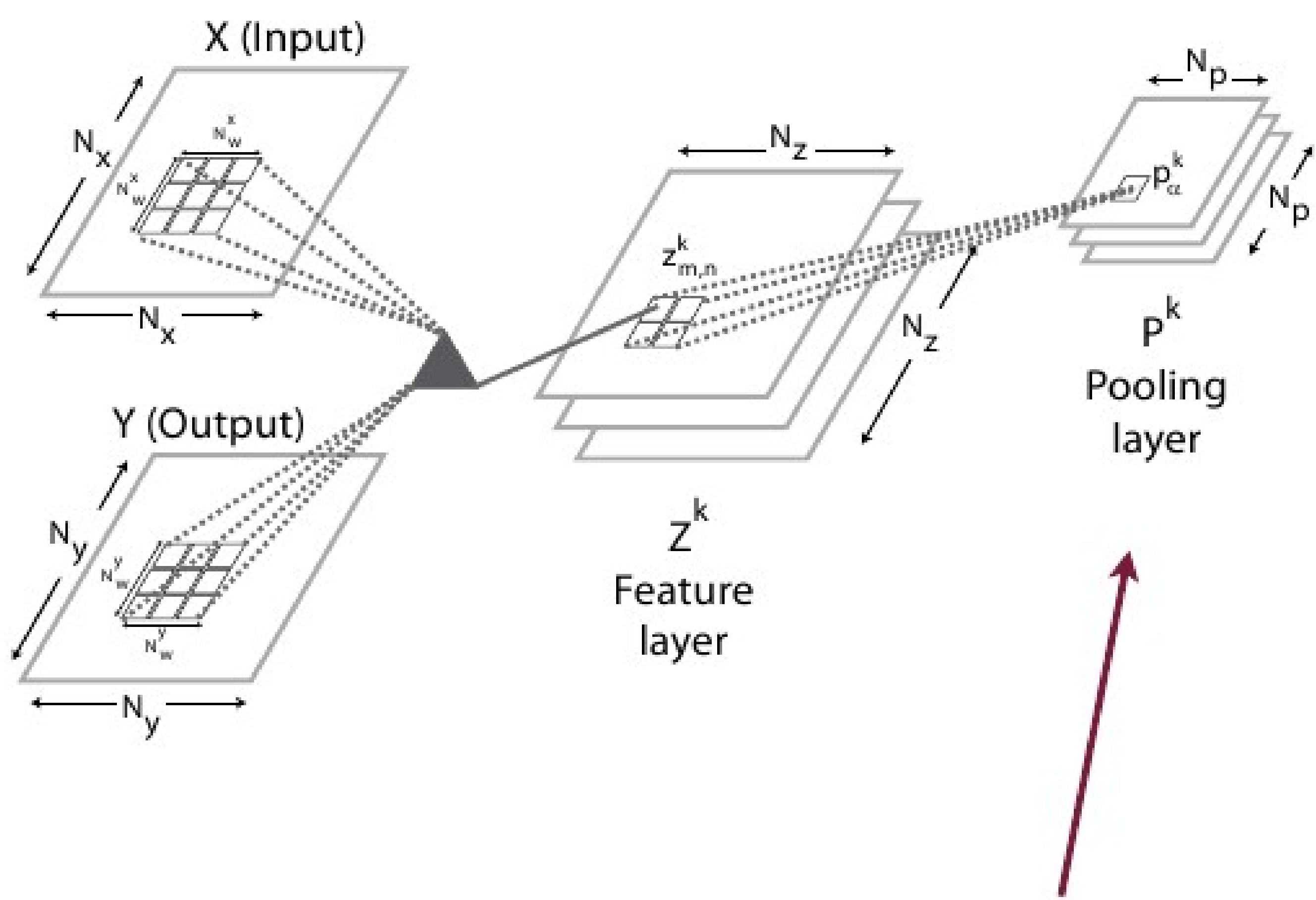
$$E(\mathbf{y}, \mathbf{z}; \mathbf{x}) = \frac{1}{2\sigma^2} \sum_j (y_j - a_j)^2 - \frac{1}{\sigma} \sum_{ijk} W_{ijk} x_i y_j z_k - \sum_k b_k z_k$$

In practice, we use this energy (real valued outputs, biases)

$$p(z_k = 1 | \mathbf{x}, \mathbf{y}) = \frac{1}{1 + \exp\left(-\sum_{ij} W_{ijk} x_i y_j\right)}$$

Inference

Convolutional Gated RBM (ConvGRBM)



Parameters (filter weights) are a 4D tensor

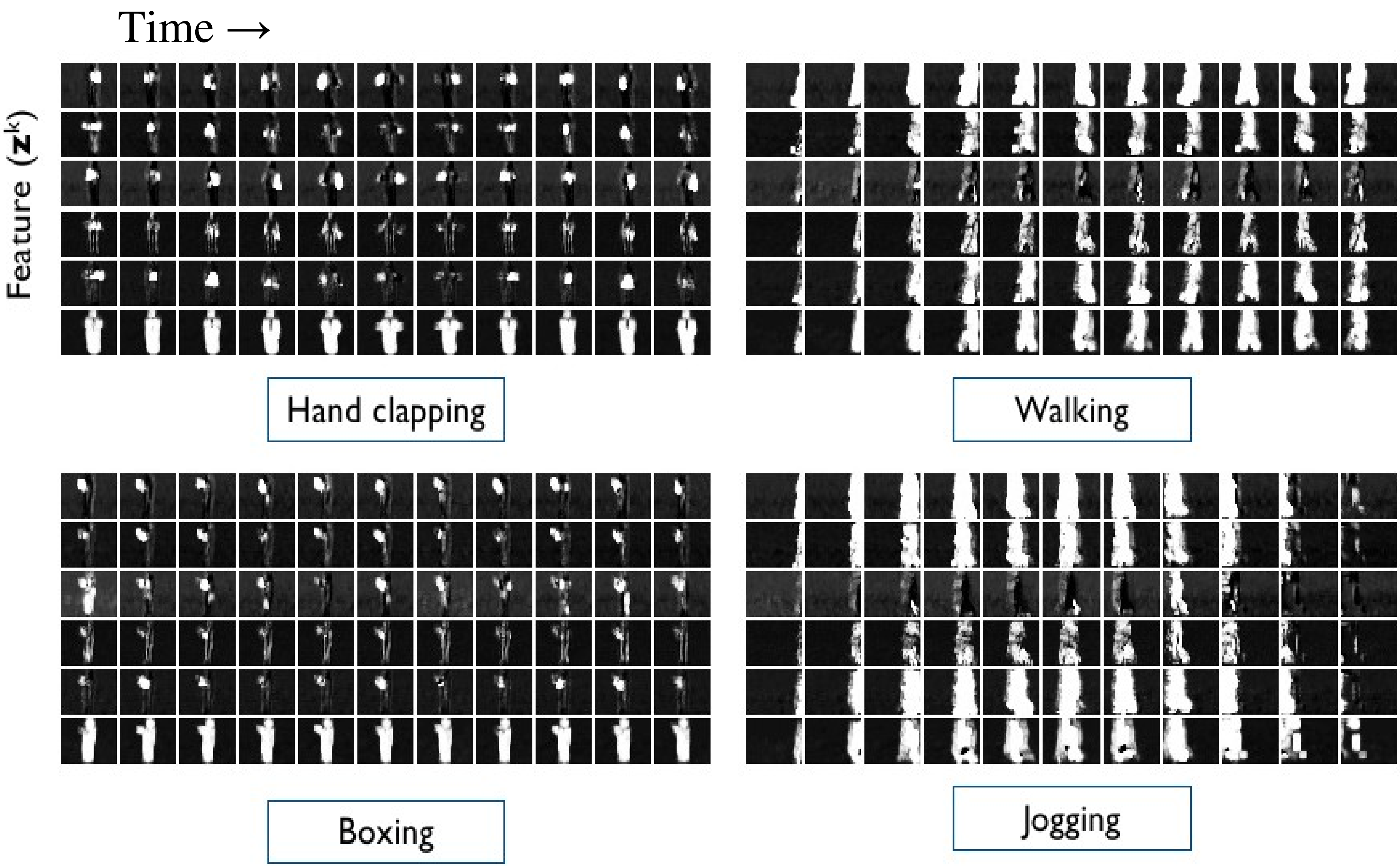
As in convolutional RBM (Lee et. al 2009)

KTH Action Dataset

- Available since 2004
- 160x120 greyscale
- 25 subjects performing 6 actions: walking, jogging, running, boxing, hand waving and hand clapping
- 4 scenarios: outdoors, outdoors with scale variation, outdoors with different clothes and indoors
- 2 popular evaluation schemes




Features Learned by ConvGRBM on KTH



Hollywood 2 Dataset

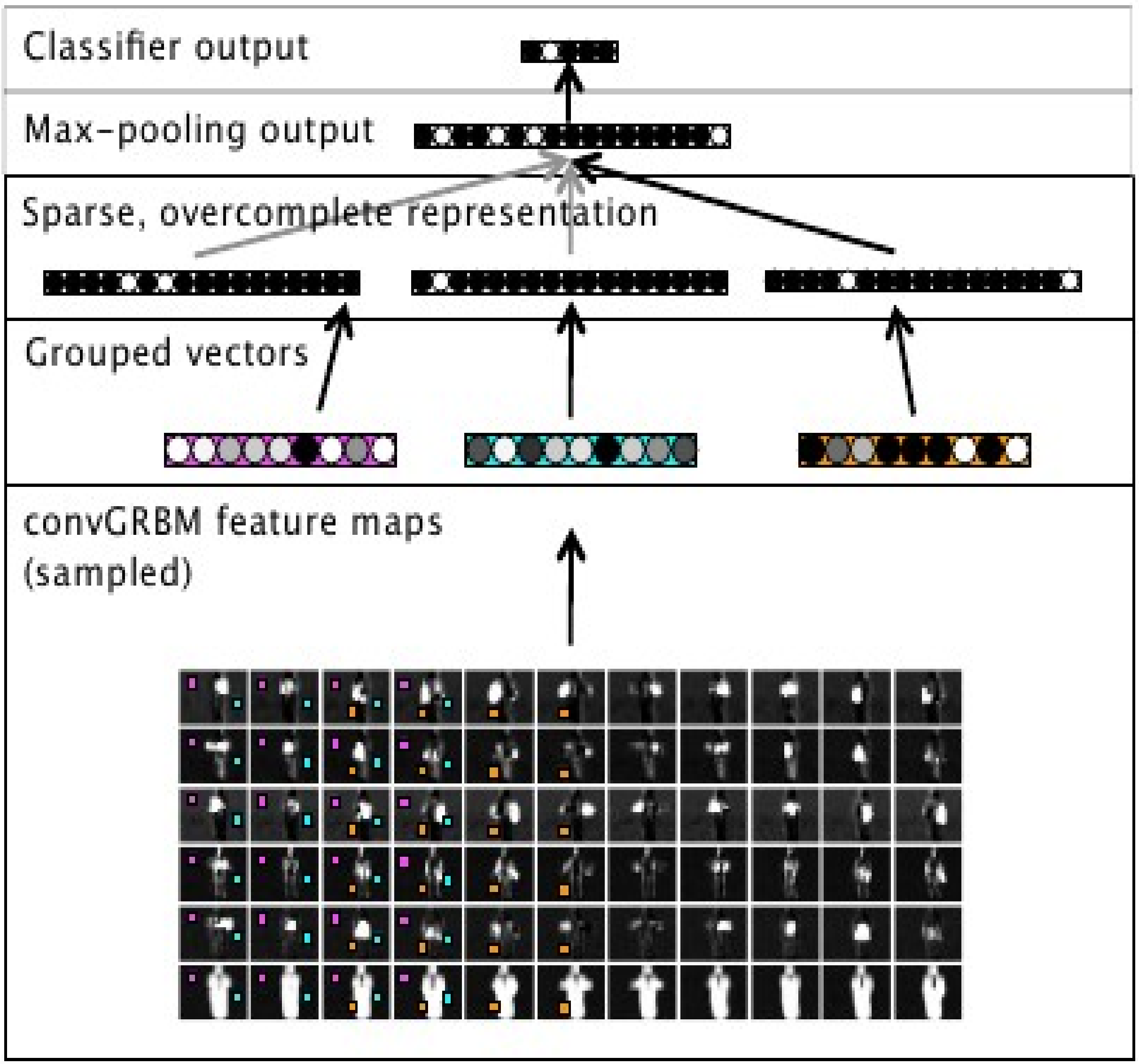
[Laptev 2008]

- 12 classes of human actions, 10 class of scenes
- 3669 video clips (30s-2m); 20.1h total video
- Captured from 69 Hollywood movies
- Samples may contain instances of several actions
- Contains an "automatic" and "human" labeled training sets

Actions			
	Training subset (clean)	Training subset (automatic)	Test subset (clean)
AnswerPhone	66	59	64
DriveCar	85	90	102
Eat	40	44	33
FightPerson	54	33	70
GetOutCar	51	40	57
HandShake	32	38	45
HugPerson	64	27	66
Kiss	114	125	103
Run	135	187	141
SitDown	104	87	108
SitUp	24	26	37
StandUp	132	133	146
All Samples	**823**	**810**	**884**

Hollywood 2 Architecture

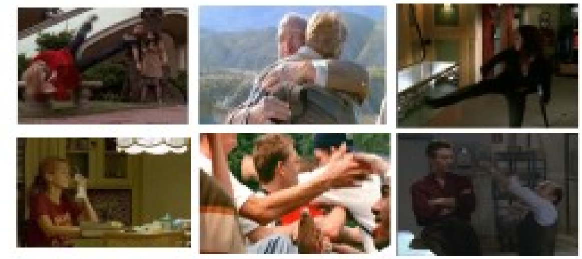
- ConvGRBM
- Sparse coding
- Max pooling
- SVM



Results

KTH actions

Hollywood2



Compared to other methods using dense sampling (no interest points):

Method	Accuracy
HOG3D-KM-SVM	85.3
HOG/HOF-KM-SVM	86.1
HOG-KM-SVM	79.0
HOF-KM-SVM	88.0
32GRBM-KM-SVM	88.3
32GRBM-SC-SVM	89.1
32convGRBM-3Dconvnet-LR	88.9
32convGRBM-3Dconvnet-MLP	**90.0**

State-of-the-art: 91.8% (Laptev et al. 2008)
Uses explicit interest-point detection

Dense sampling actually works better:

Method	Mean AP
HOG3D-KM-SVM	45.3
HOG/HOF-KM-SVM	**47.4**
HOG-KM-SVM	39.4
HOF-KM-SVM	45.5
convGRBM+SC+SVM	**46.8**

2nd place

We also outperform Cuboids (45.0%) and Willems et. al (38.2%)

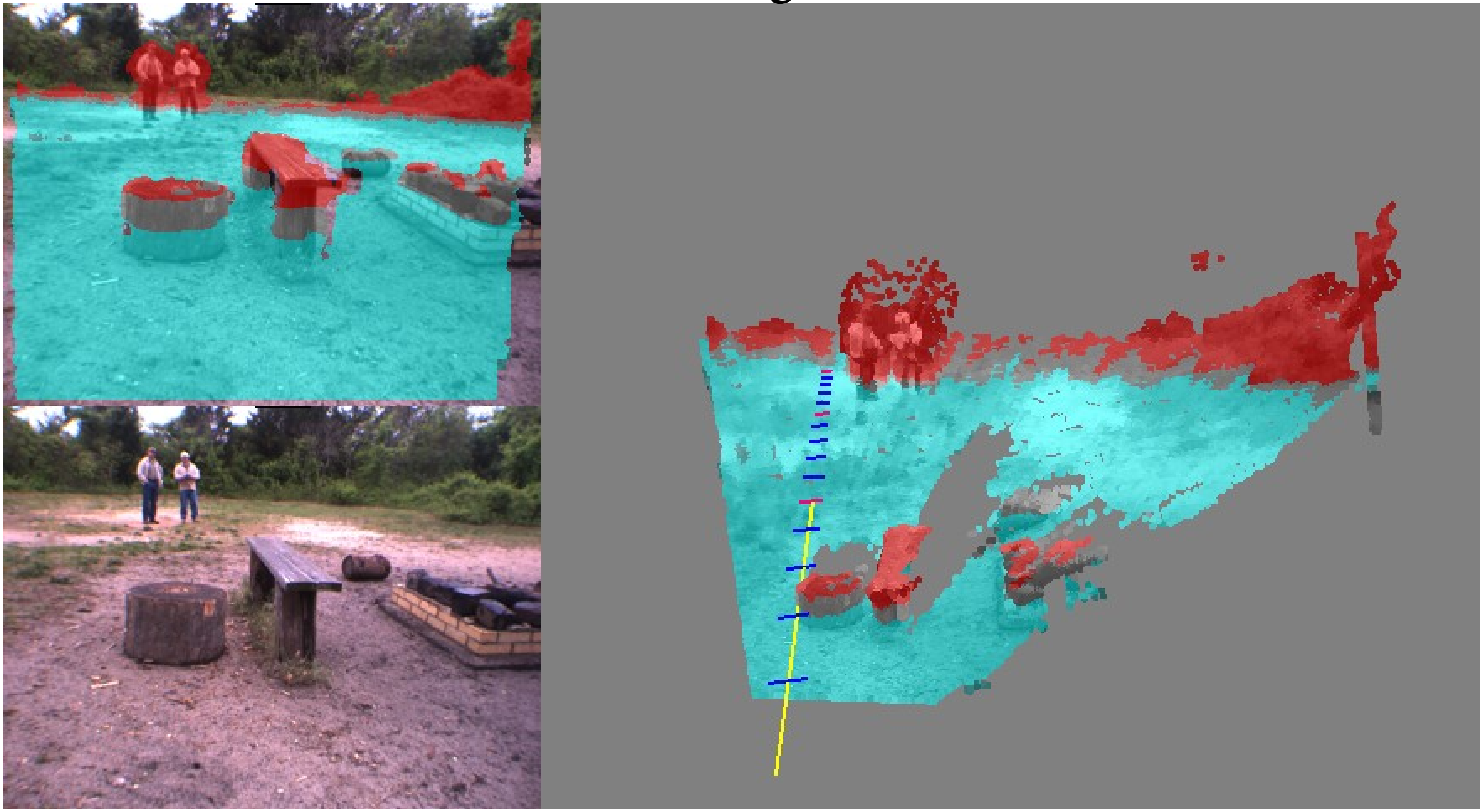
Deep Learning for Mobile Robot Vision

DARPA/LAGR: Learning Applied to Ground Robotics

- Getting a robot to drive autonomously in unknown terrain solely from vision (camera input).
- Our team (NYU/Net-Scale Technologies Inc.) was one of 8 participants funded by DARPA
- All teams received identical robots and can only modify the software (not the hardware)
- The robot is given the GPS coordinates of a goal, and must drive to the goal as fast as possible. The terrain is unknown in advance. The robot is run 3 times through the same course.
- Long-Range Obstacle Detection with on-line, self-trained ConvNet
- Uses temporal consistency!

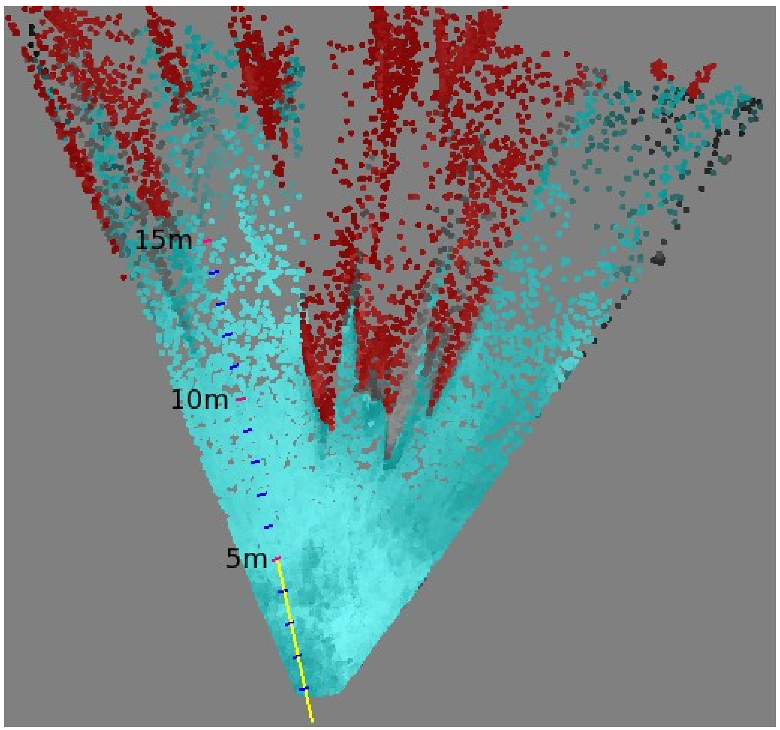
Obstacle Detection

Obstacles overlaid with camera image



Camera image Detected obstacles (red)

Navigating to a goal is hard...



stereo perspective

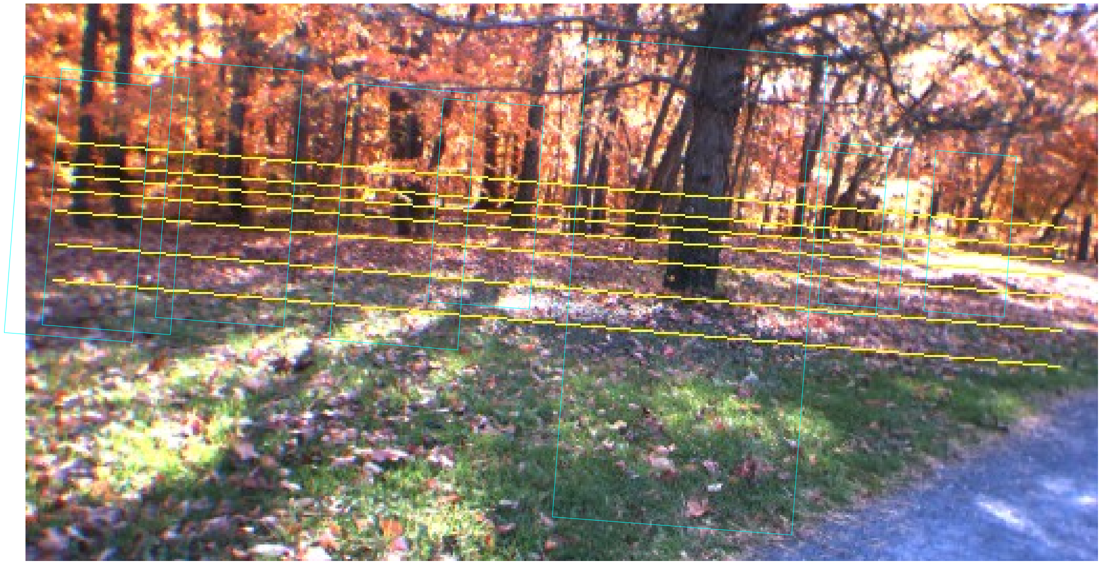
human perspective

especially in a snowstorm.

Self-Supervised Learning

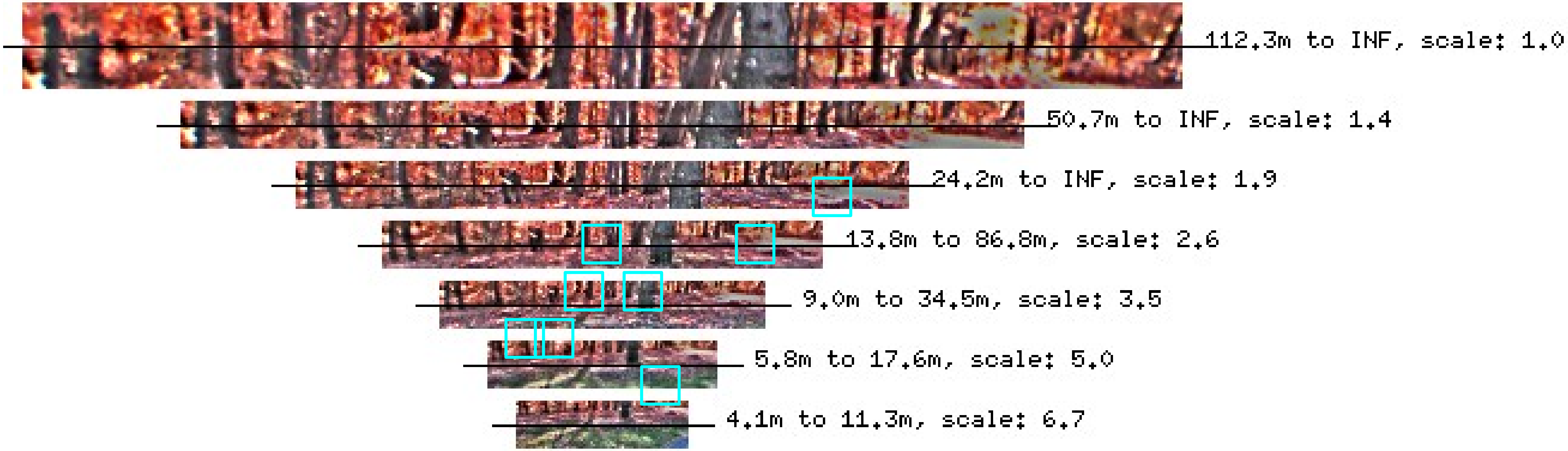
- Stereo vision tells us what nearby obstacles look like
- Use the labels (obstacle/traversible) produced by stereo vision to train a monocular neural network
- Self-supervised “near to far” learning

Long Range Vision: Distance Normalization



Pre-processing (125 ms)

- Ground plane estimation
- Horizon leveling
- Conversion to YUV + local contrast normalization
- Scale invariant pyramid of distance-normalized image "bands"

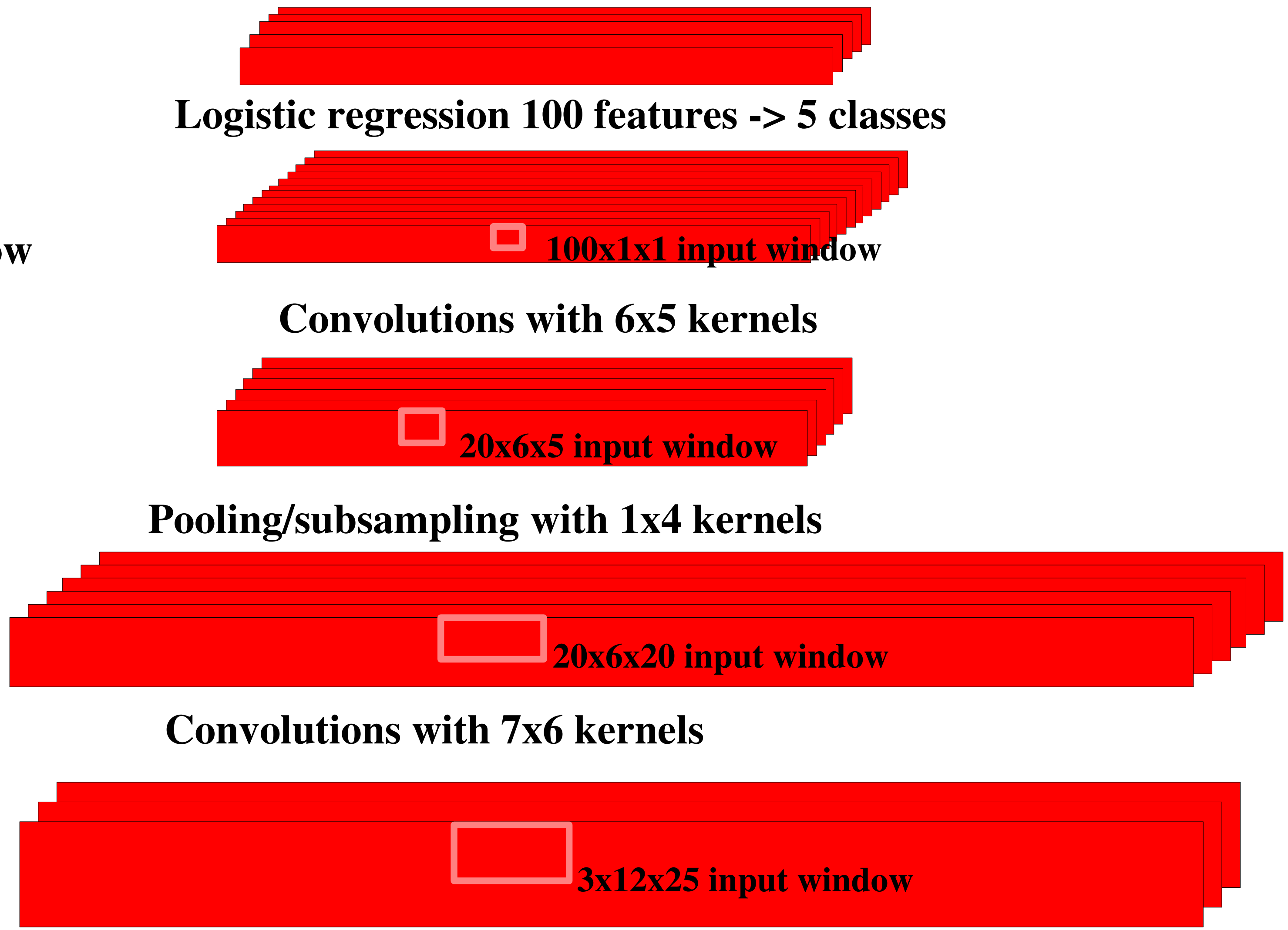


Convolutional Net Architecture

- Operates on 12x25 YUV windows from the pyramid

100 features per

3x12x25 input window

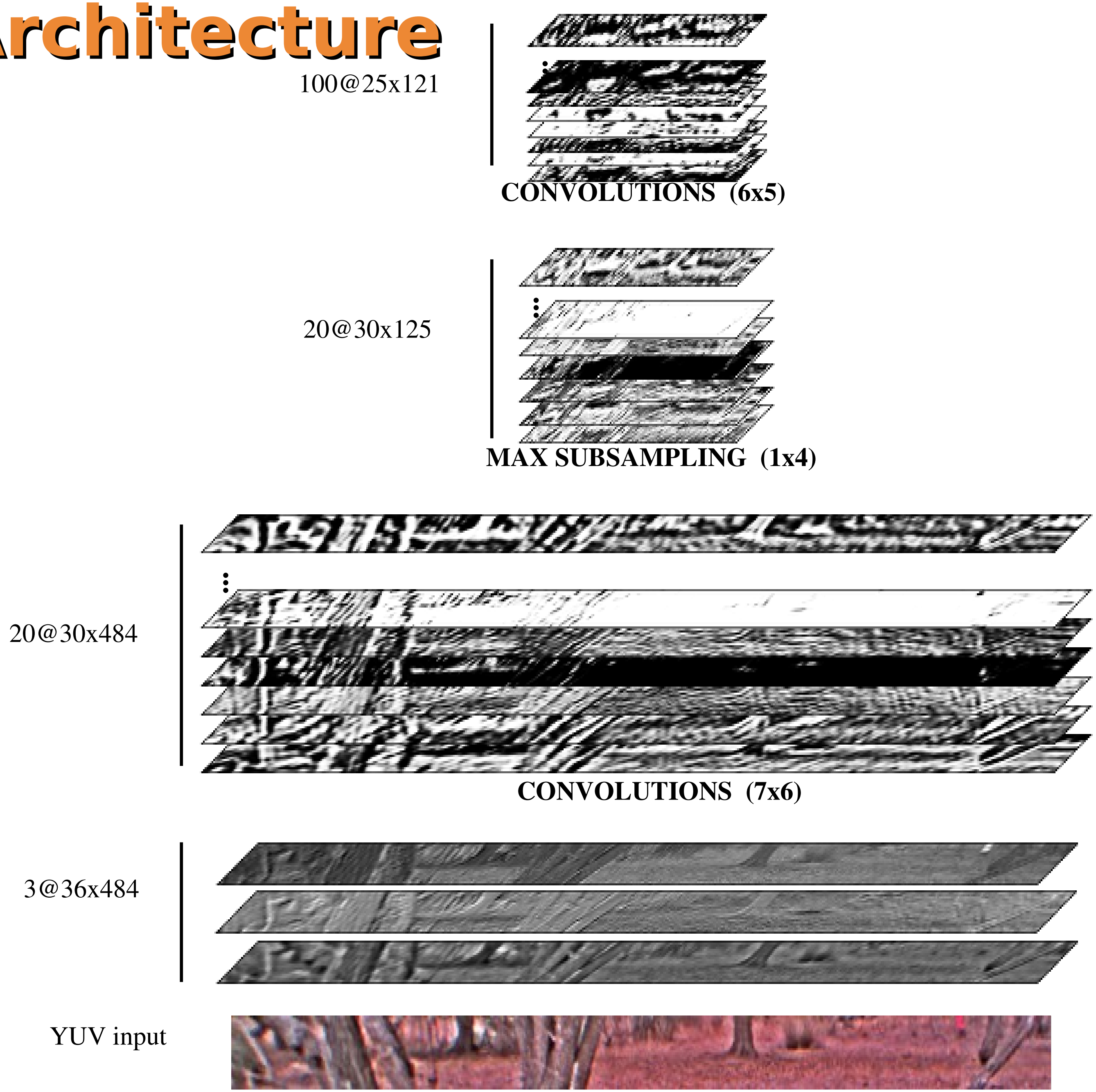


YUV image band

20-36 pixels tall,

36-500 pixels wide

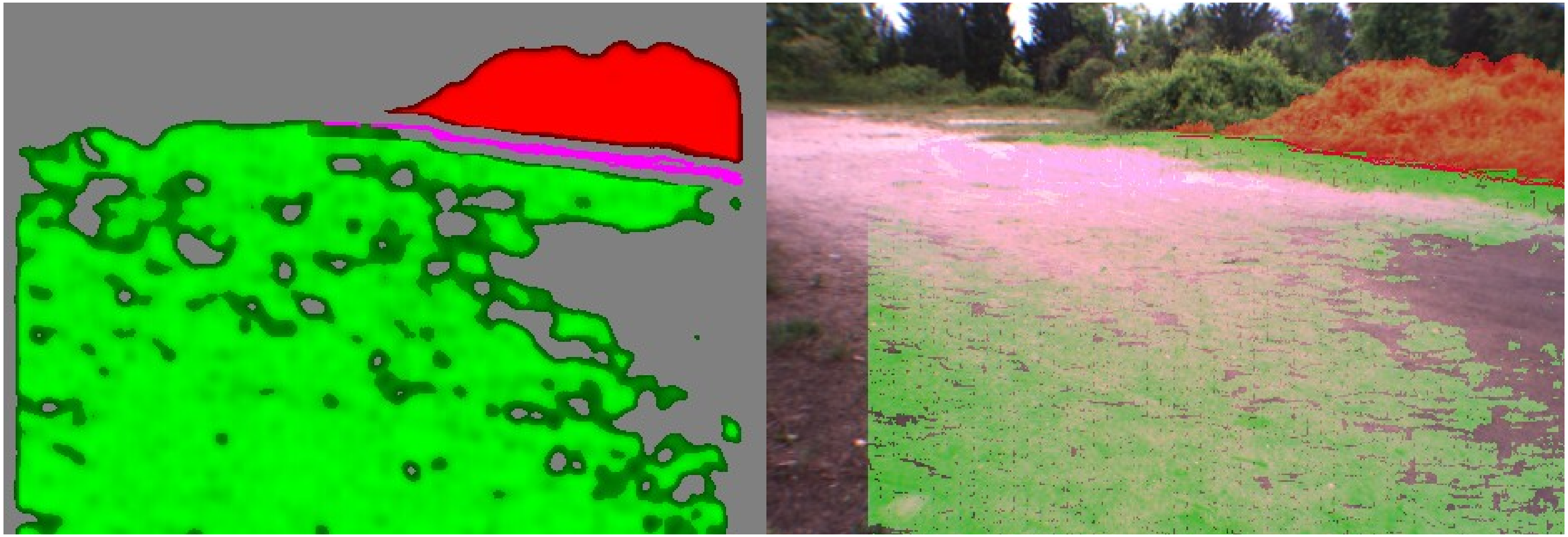
Convolutional Net Architecture



Long Range Vision: 5 categories

Online Learning (52 ms)

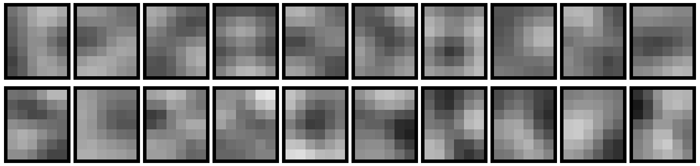
- Label windows using stereo information – 5 classes

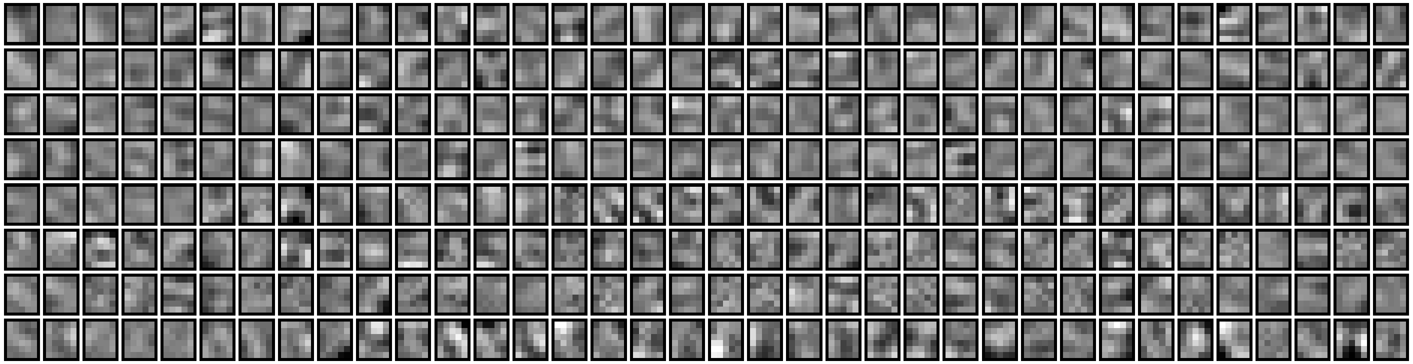


super-ground **ground** **footline** **obstacle** **super-obstacle**

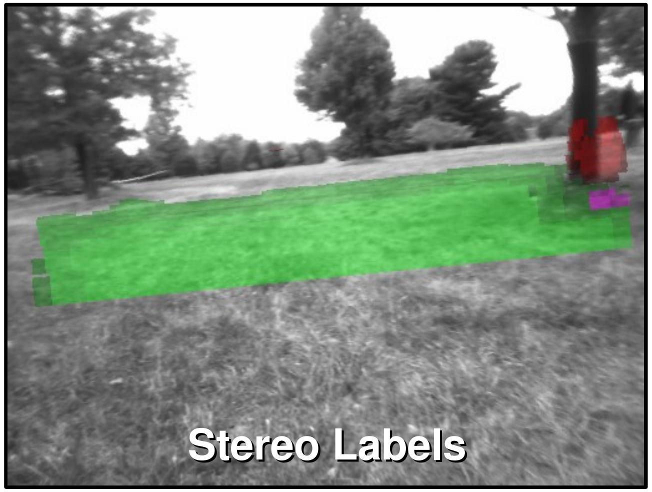
Trainable Feature Extraction

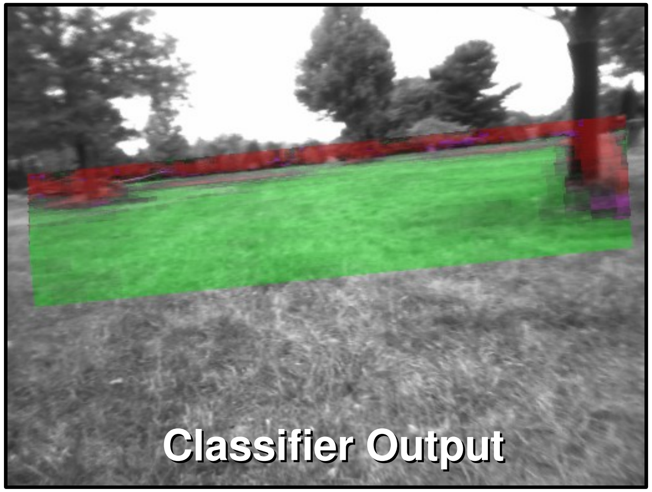
- "Deep belief net" approach to unsupervised feature learning
- Two stages are trained in sequence
  - each stage has a layer of convolutional filters and a layer of horizontal feature pooling.
  - Naturally shift invariant in the horizontal direction
- Filters of the convolutional net are trained so that the input can be reconstructed from the features
  - 20 filters at the first stage (layers 1 and 2)
  - 300 filters at the second stage (layers 3 and 4)
- Scale invariance comes from pyramid.
  - for near-to-far generalization

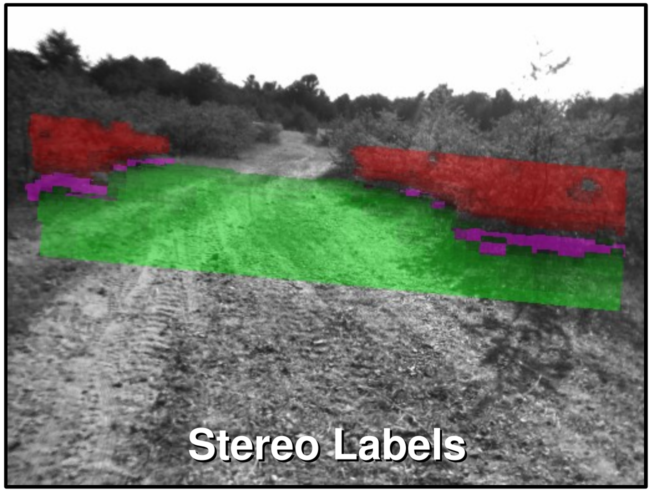


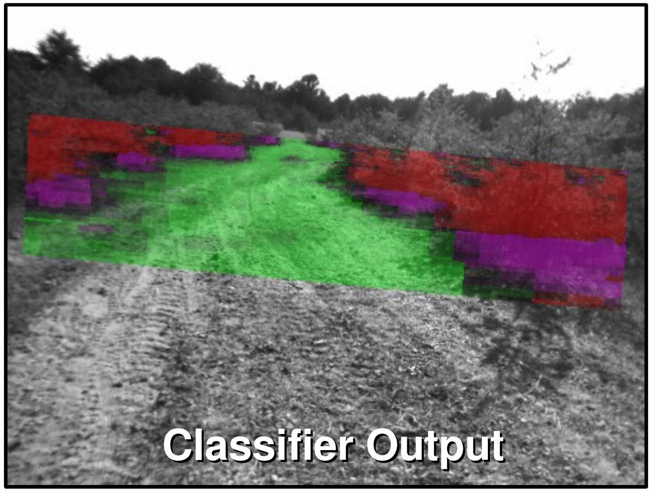


Long Range Vision Results

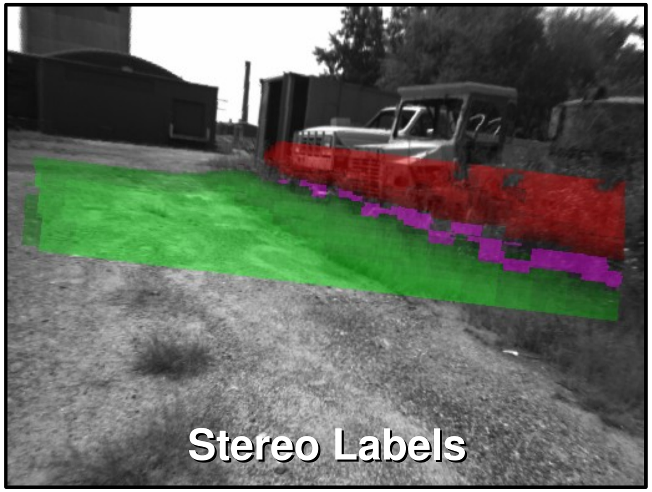


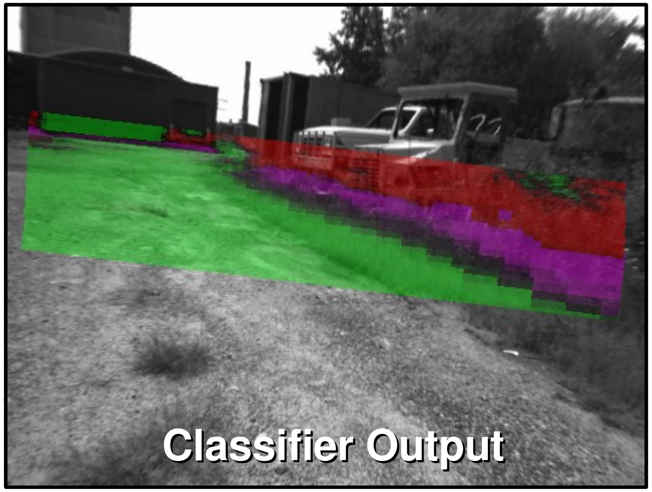


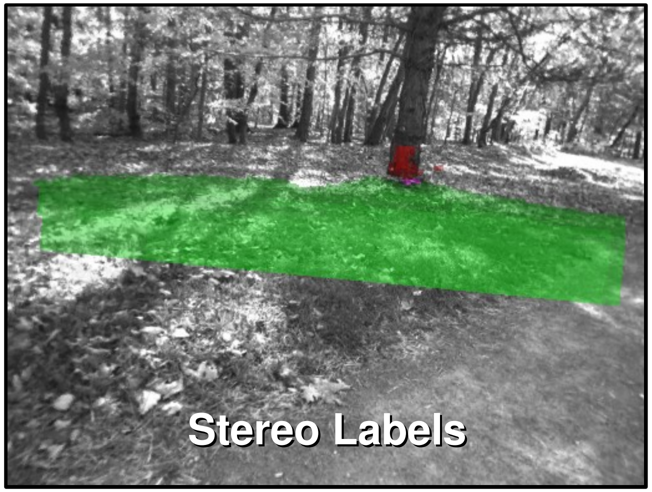


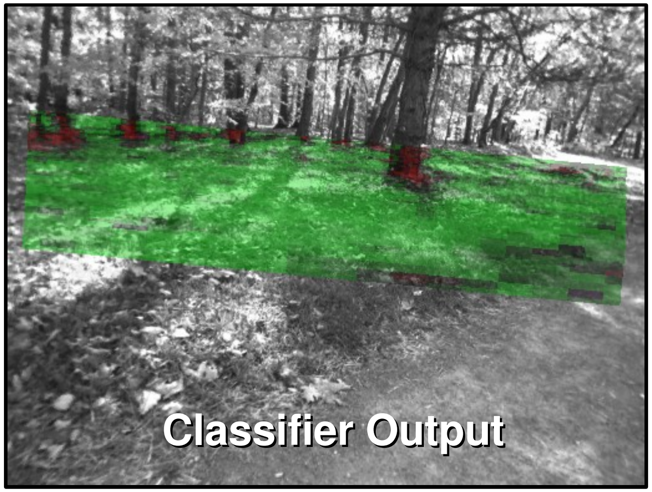


Long Range Vision Results

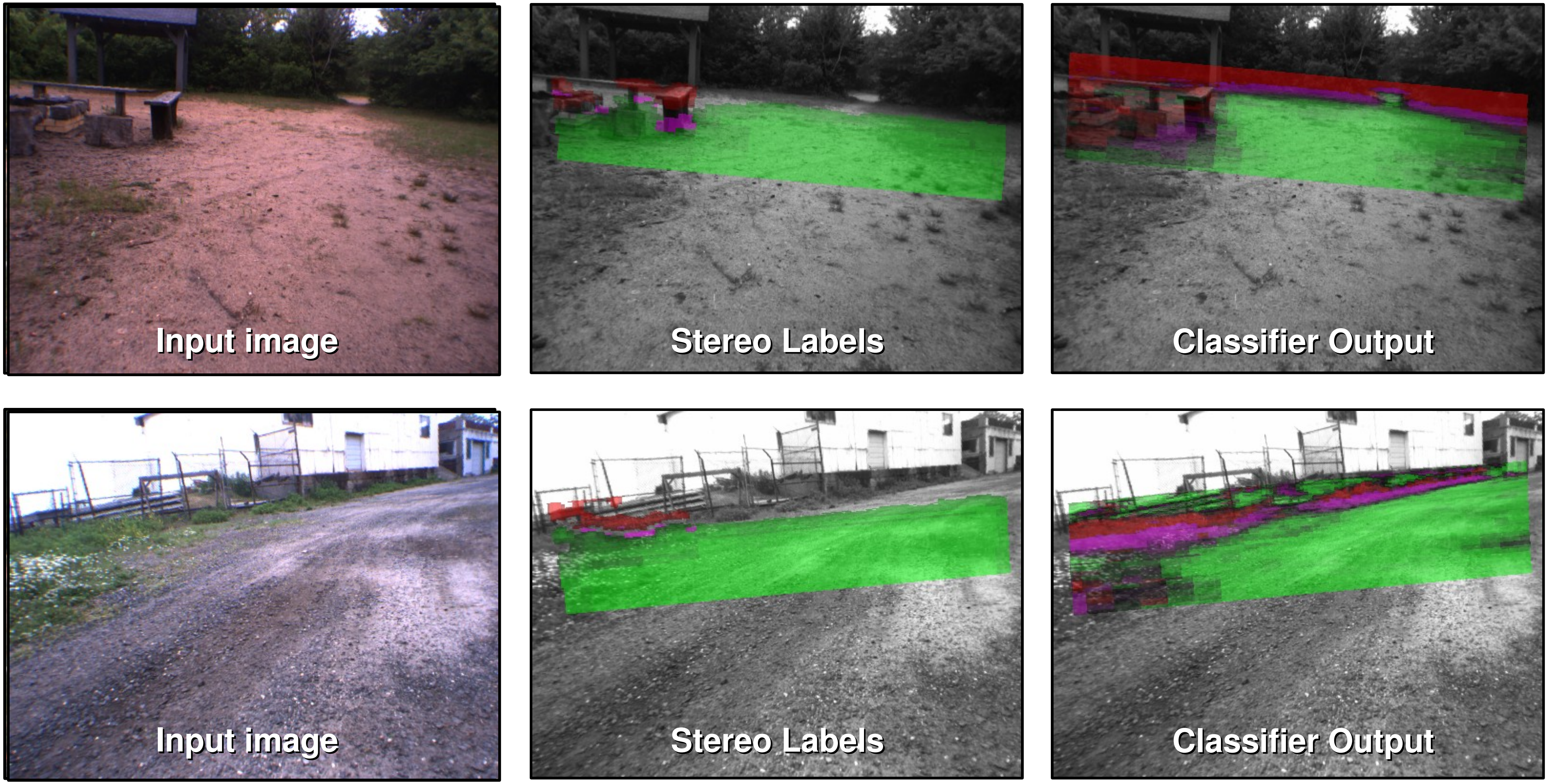


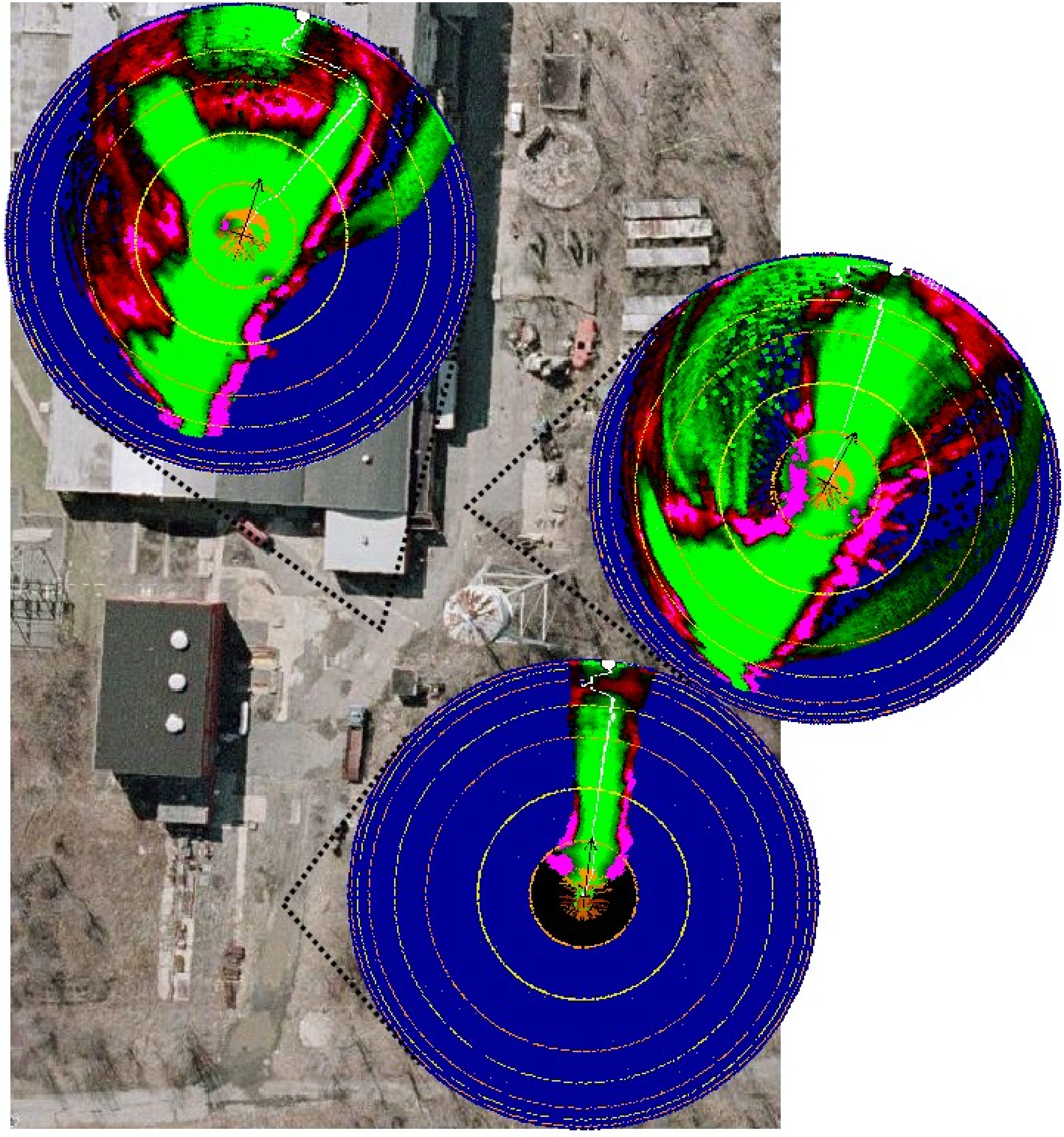






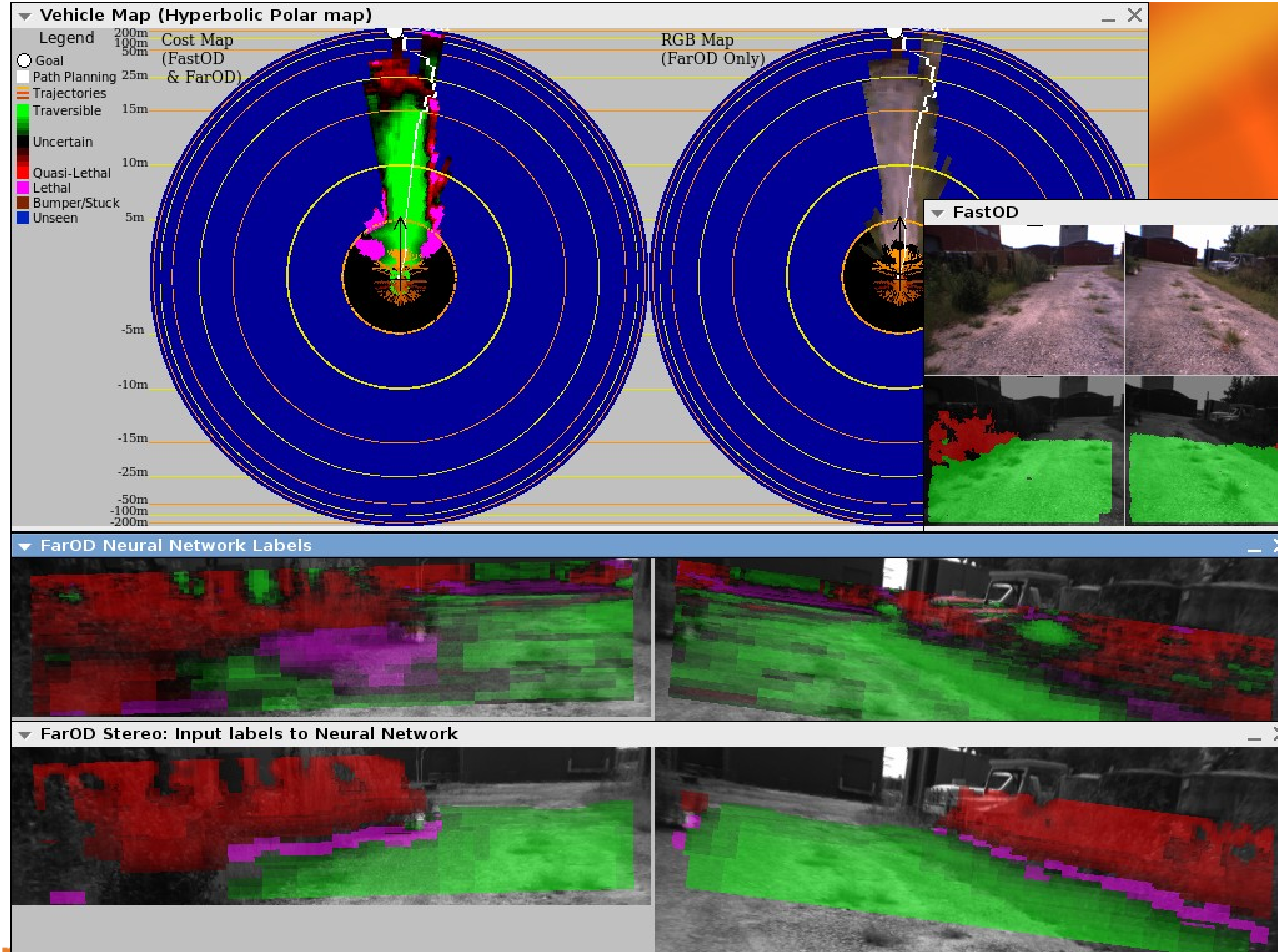
Long Range Vision Results

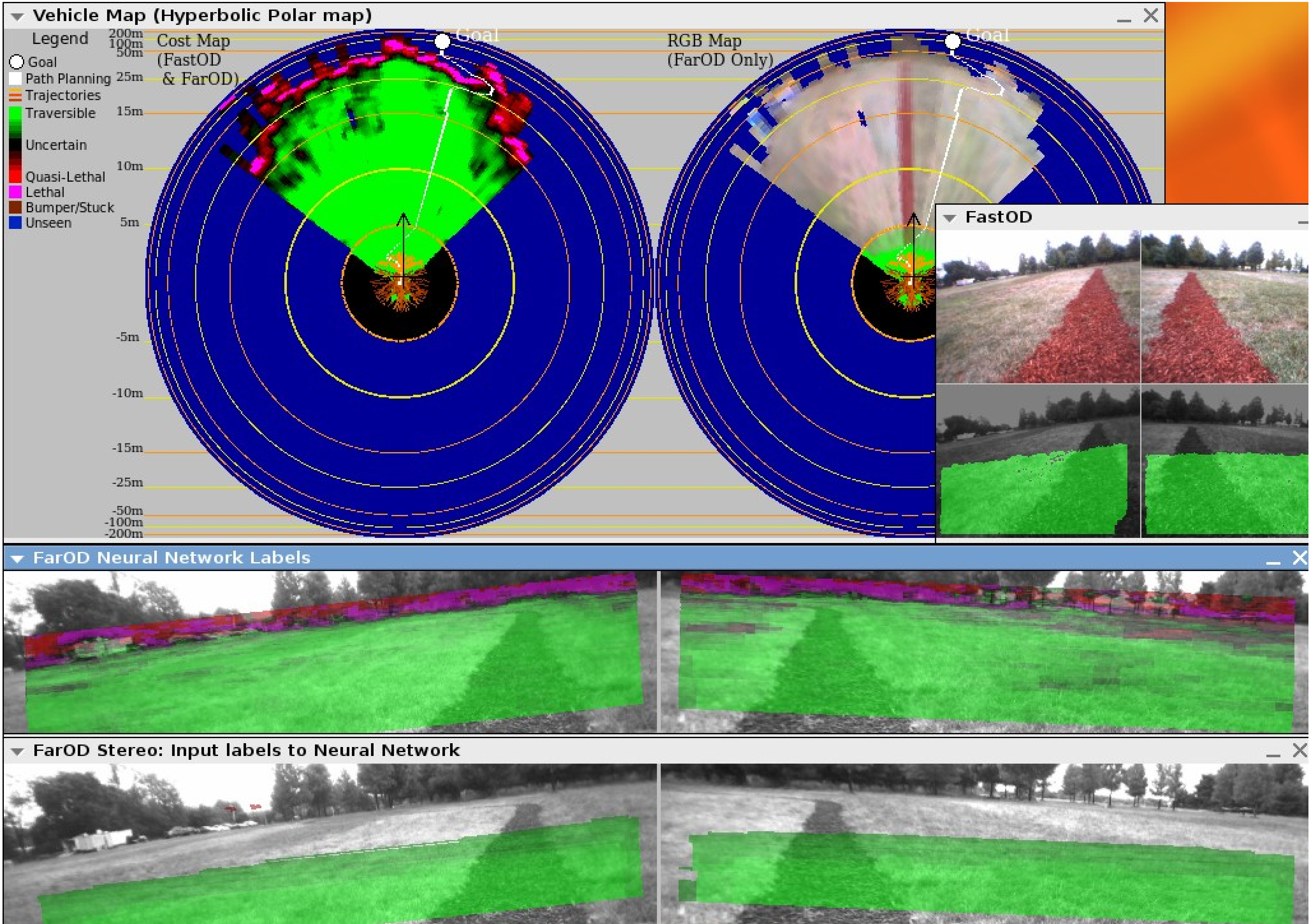




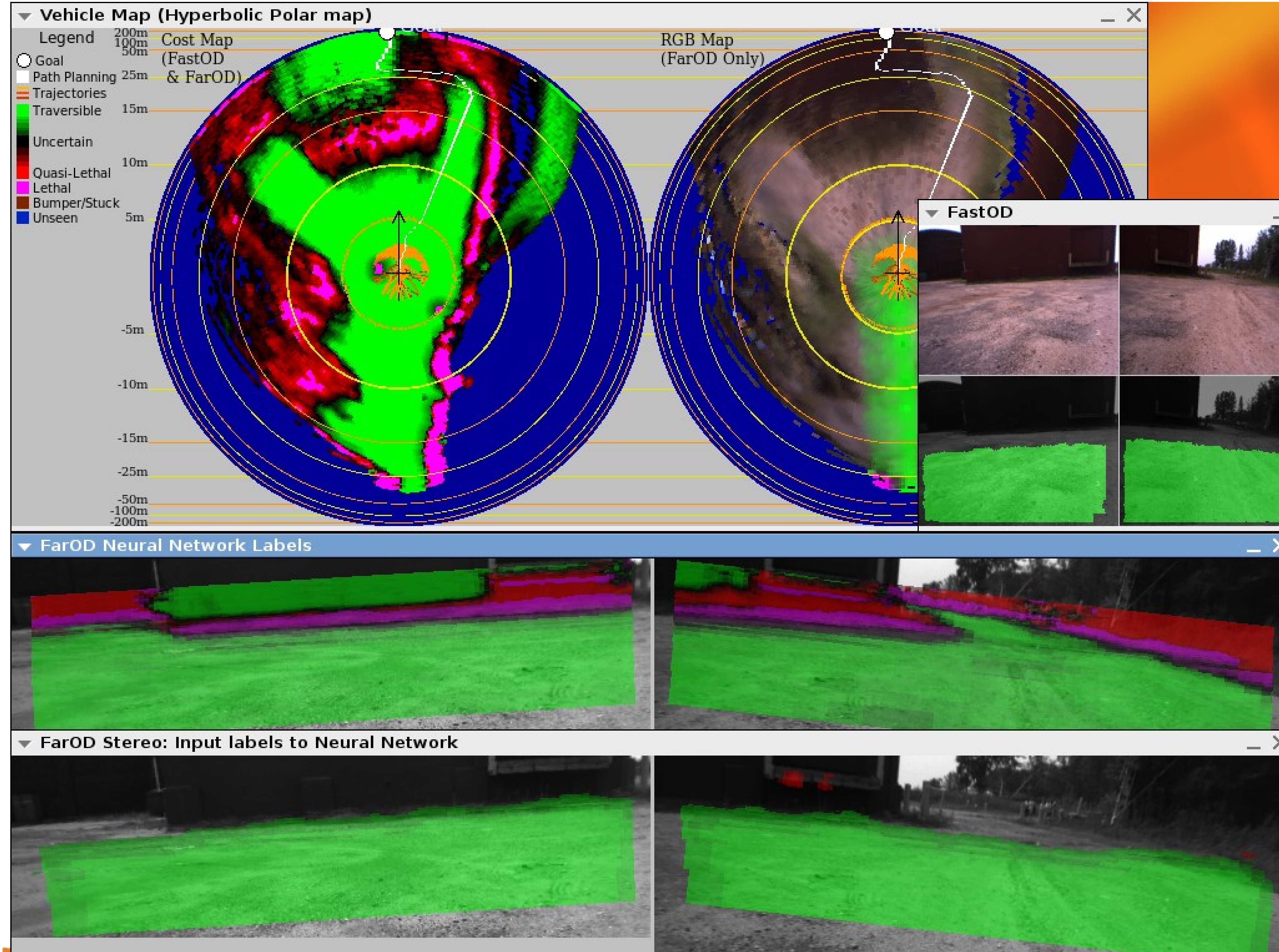
netSCALE Technologies, Inc.

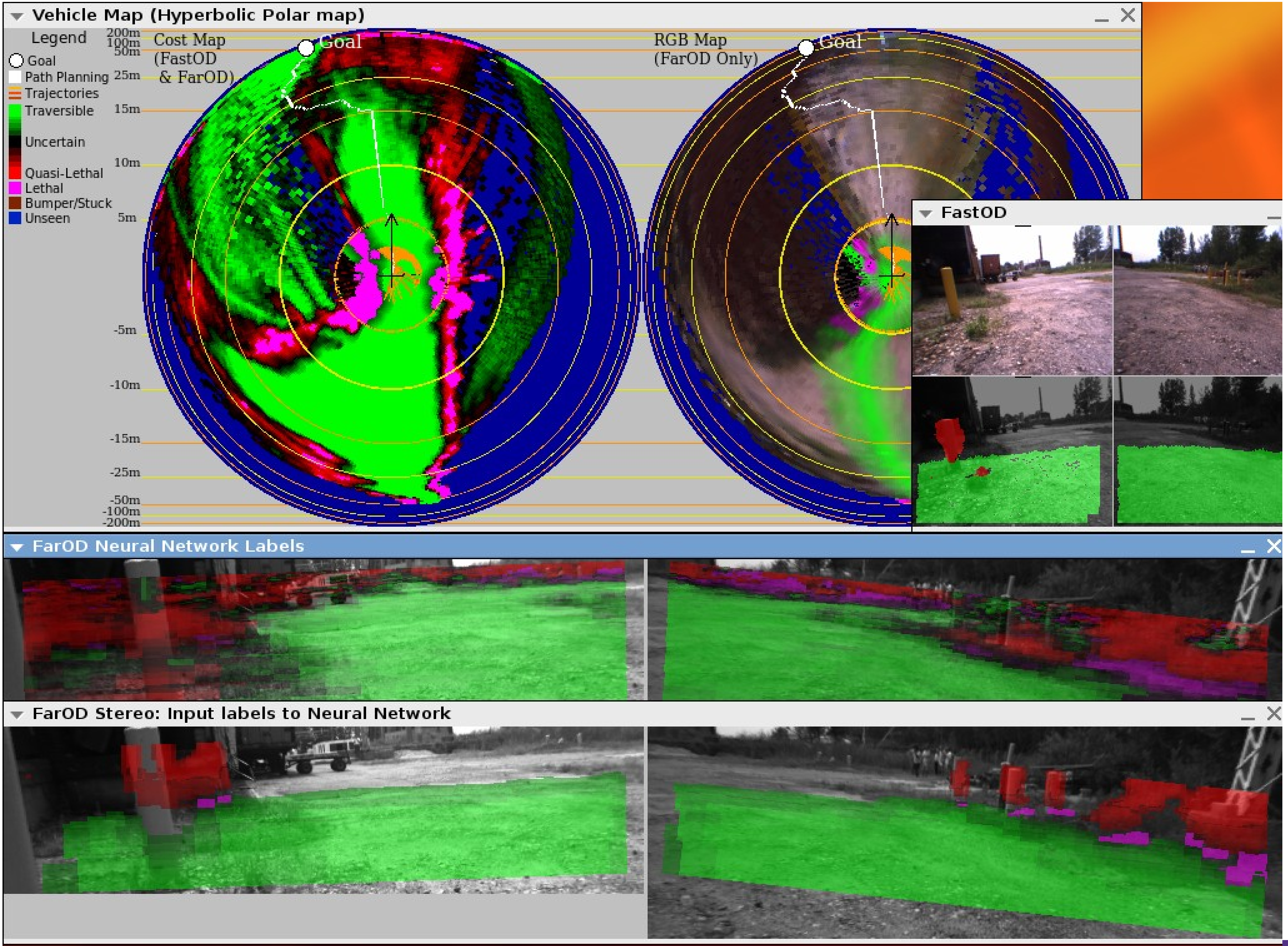
RSITY


Vehicle Map (Hyperbolic Polar map)
Legend
Goal
Path Planning
Trajectories
Traversible
Uncertain
Quasi-Lethal
Lethal
Bumper/Stuck
Unseen
200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m
Cost Map
(FastOD
& FarOD)
RGB Map
(FarOD Only)
FastOD
FarOD Neural Network Labels
FarOD Stereo: Input labels to Neural Network

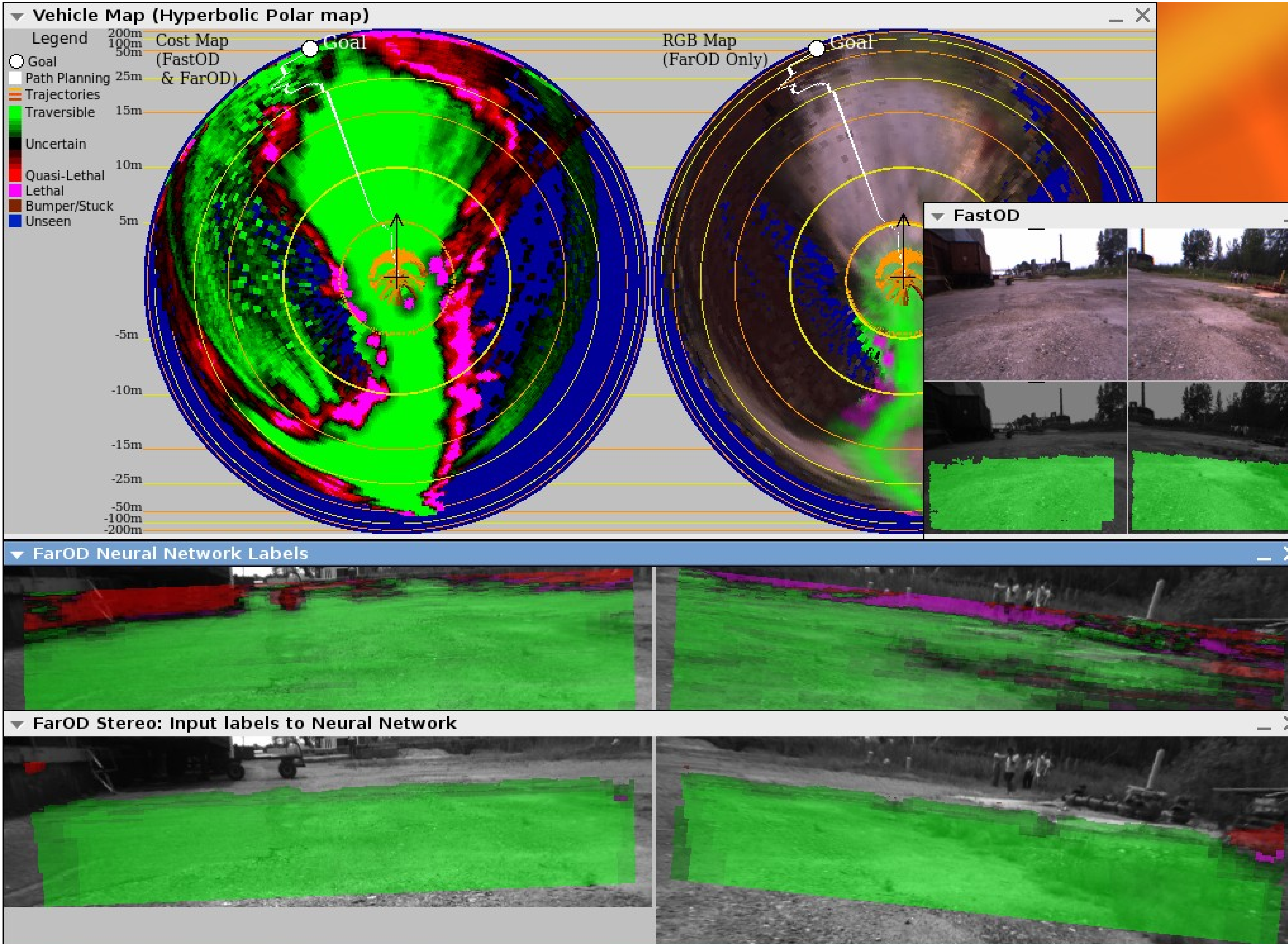


Vehicle Map (Hyperbolic Polar map)
Legend
Goal
Path Planning
Trajectories
Traversible
Uncertain
Quasi-Lethal
Lethal
Bumper/Stuck
Unseen
200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m
Cost Map
(FastOD
& FarOD)
Goal
RGB Map
(FarOD Only)
Goal
FastOD
FarOD Neural Network Labels
FarOD Stereo: Input labels to Neural Network


Vehicle Map (Hyperbolic Polar map)
Legend
Goal
Path Planning
Trajectories
Traversible
Uncertain
Quasi-Lethal
Lethal
Bumper/Stuck
Unseen
200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m
Cost Map (FastOD & FarOD)
RGB Map (FarOD Only)
FastOD
FarOD Neural Network Labels
FarOD Stereo: Input labels to Neural Network



Vehicle Map (Hyperbolic Polar map)
Legend
Goal
Path Planning
Trajectories
Traversible
Uncertain
Quasi-Lethal
Lethal
Bumper/Stuck
Unseen
200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m
Cost Map (FastOD & FarOD)
Goal
RGB Map (FarOD Only)
Goal
FastOD
FarOD Neural Network Labels
FarOD Stereo: Input labels to Neural Network

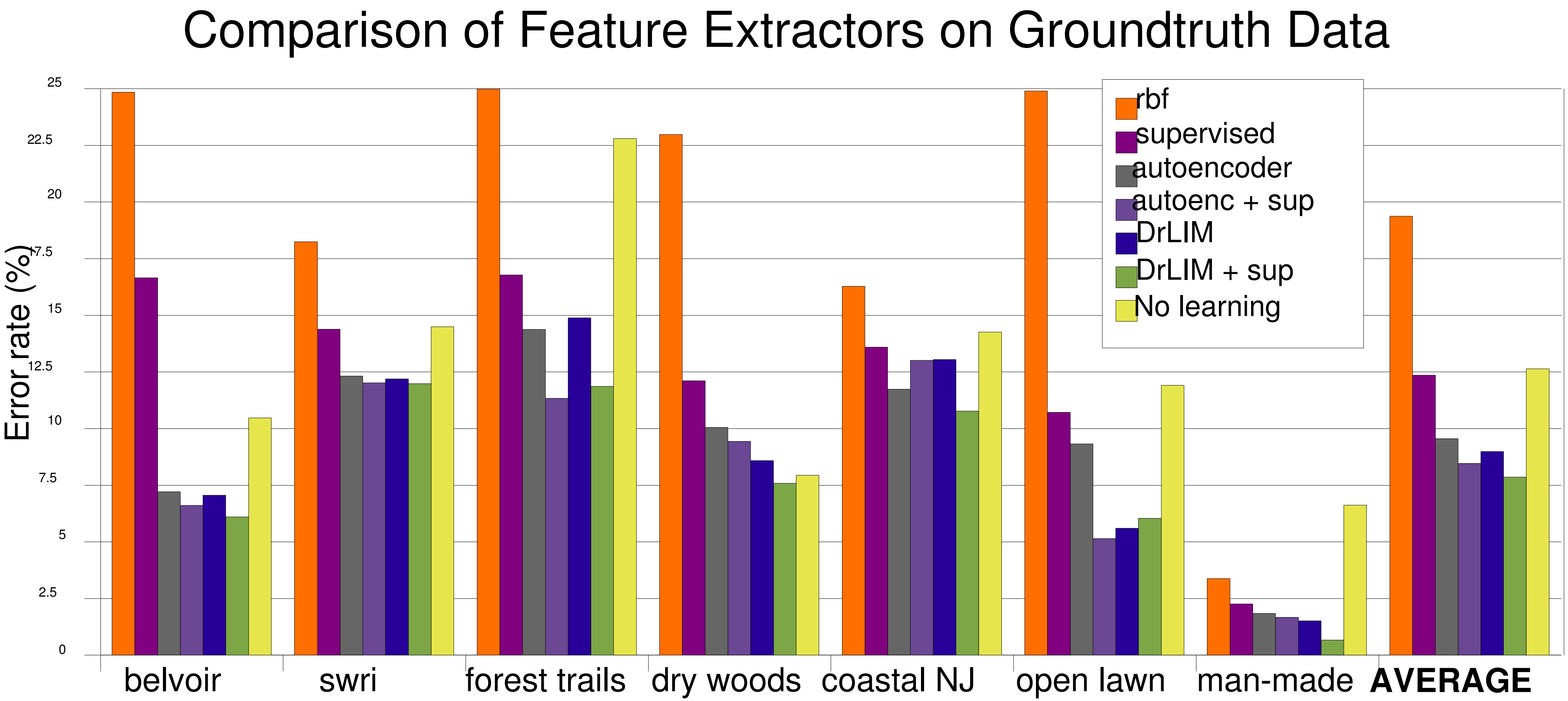

Vehicle Map (Hyperbolic Polar map)
Legend
Goal
Path Planning
Trajectories
Traversible
Uncertain
Quasi-Lethal
Lethal
Bumper/Stuck
Unseen
200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m
Cost Map
(FastOD
& FarOD)
Goal
RGB Map
(FarOD Only)
Goal
FastOD
FarOD Neural Network Labels
FarOD Stereo: Input labels to Neural Network

Feature Learning for traversability prediction (LAGR)

Comparing

- purely supervised
- stacked, invariant auto-encoders
- DrLIM invariant learning

Testing on hand-labeled groundtruth frames – binary labels



The End