

Learning Hierarchies of Invariant Visual Features

Yann LeCun

**The Courant Institute of Mathematical Sciences
New York University**

collaborators:

Y-Lan Boureau, Clément Farabet,

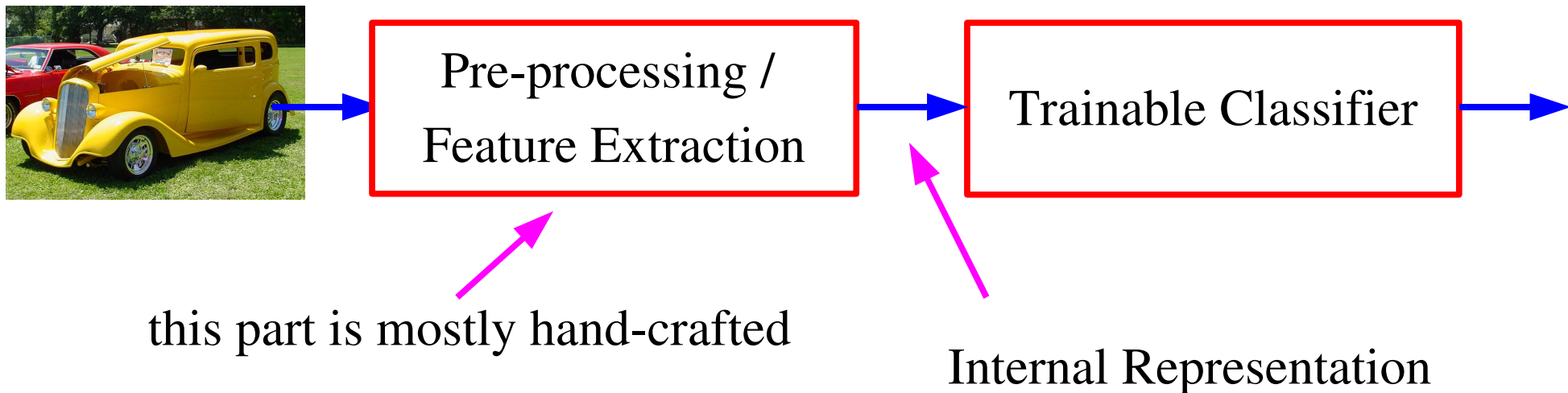
Rob Fergus, Kevin Jarrett, Karol Gregor,

Raia Hadsell, Koray Kavukcuoglu, Marc'Aurelio Ranzato

The Next Frontier in Machine Learning: Learning Representations

- **The big success of ML has been to learn classifiers from labeled data**
 - ▶ The representation of the input, and the metric to compare them are assumed to be “intelligently designed.”
 - ▶ Example: Support Vector Machines require a good input representation, and a good kernel function.
- **The next frontier is to “learn the features”**
 - ▶ The question: how can a machine learn good internal representations
 - ▶ In language, good representations are paramount.
 - What makes the words “cat” and “dog” semantically similar?
 - How can different sentences with the same meaning be mapped to the same internal representation?
- **How can we leverage unlabeled data (which is plentiful)?**

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)
- The most common Machine Learning architecture: the **Kernel Machine**

The Next Challenge of ML, Vision (and Neuroscience)

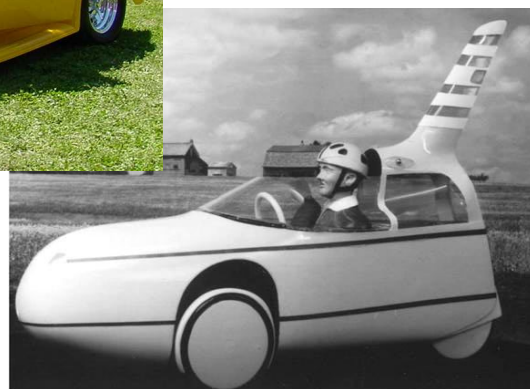
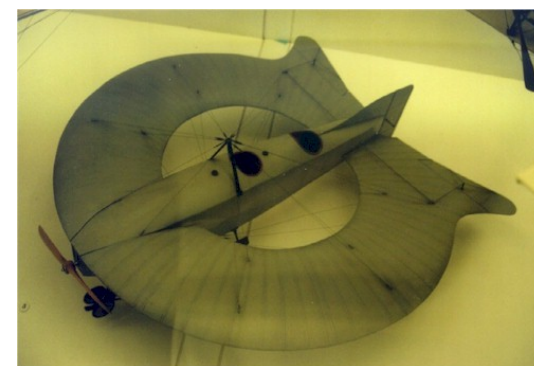
How do we learn invariant representations?

- ▶ From the image of an airplane, how do we extract a representation that is invariant to pose, illumination, background, clutter, object instance....
- ▶ How can a human (or a machine) learn those representations by just looking at the world?

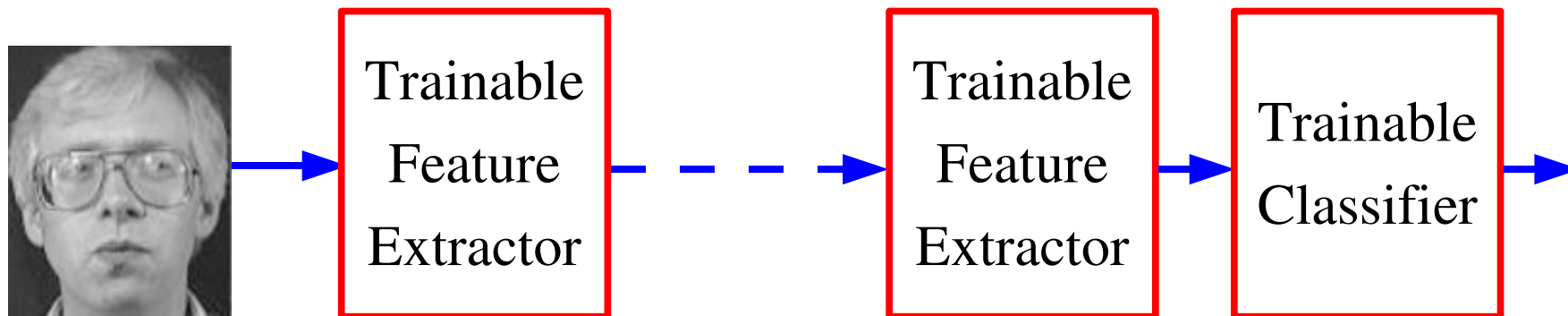


How can we learn visual categories from just a few examples?

- ▶ I don't need to see many airplanes before I can recognize every airplane (even really weird ones)



Good Representations are Hierarchical



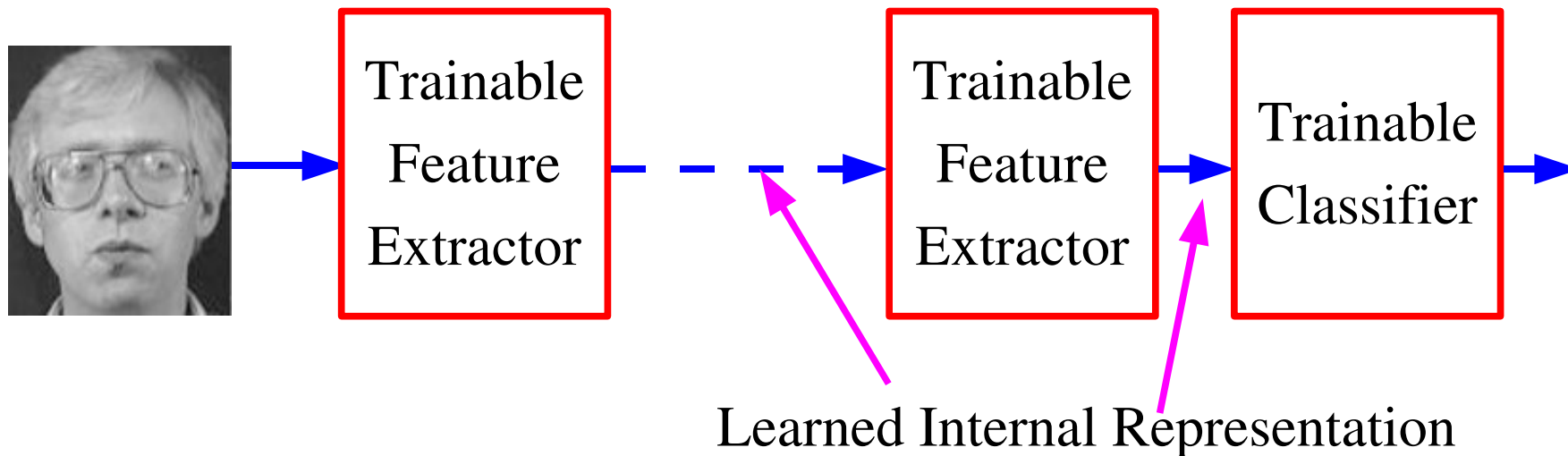
● In Language: hierarchy in syntax and semantics

- ▶ Words->Parts of Speech->Sentences->Text
- ▶ Objects,Actions,Attributes...-> Phrases -> Statements -> Stories

● In Vision: part-whole hierarchy

- ▶ Pixels->Edges->Textons->Parts->Objects->Scenes

“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 stages gets around the specificity/invariance dilemma

The Primate's Visual System is Deep

- **The recognition of everyday objects is a very fast process.**
 - ▶ The recognition of common objects is essentially “feed forward.”
 - ▶ But not all of vision is feed forward.
- **Much of the visual system (all of it?) is the result of learning**
 - ▶ How much prior structure is there?
- **If the visual system is deep and learned, what is the learning algorithm?**
 - ▶ What learning algorithm can train neural nets as “deep” as the visual system (10 layers?).
 - ▶ Unsupervised vs Supervised learning
 - ▶ What is the loss function?
 - ▶ What is the organizing principle?
 - ▶ Broader question (Hinton): what is the learning algorithm of the neo-cortex?

Do we really need deep architectures?

- We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?

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

- ▶ kernel machines and 2-layer neural net are “universal”.

- Deep learning machines

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

- Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition

- ▶ they can represent more complex functions with less “hardware”

- We need an efficient parameterization of the class of functions that are useful for “AI” tasks.

Why are Deep Architectures More Efficient?

[Bengio & LeCun 2007 “Scaling Learning Algorithms Towards AI”]

● A deep architecture trades space for time (or breadth for depth)

- ▶ more layers (more sequential computation),
- ▶ but less hardware (less parallel computation).
- ▶ Depth-Breadth tradoff

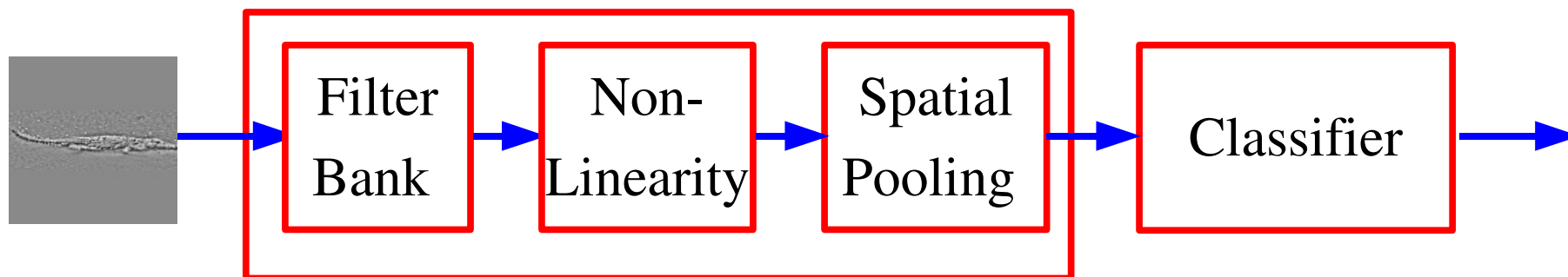
● Example1: N-bit parity

- ▶ requires $N-1$ XOR gates in a tree of depth $\log(N)$.
- ▶ requires an exponential number of gates if we restrict ourselves to 2 layers (DNF formula with exponential number of minterms).

● Example2: circuit for addition of 2 N-bit binary numbers

- ▶ Requires $O(N)$ gates, and $O(N)$ layers using N one-bit adders with ripple carry propagation.
- ▶ Requires lots of gates (some polynomial in N) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
- ▶ Bad news: almost all boolean functions have a DNF formula with an exponential number of minterms $O(2^N)$

Feature Extraction in Computer Vision



Oriented Edges

Gabor Wavelets

Other Filters...

Sigmoid

Rectification

Vector Quant.

Contrast Norm.

Averaging

Max pooling

VQ+Histogram

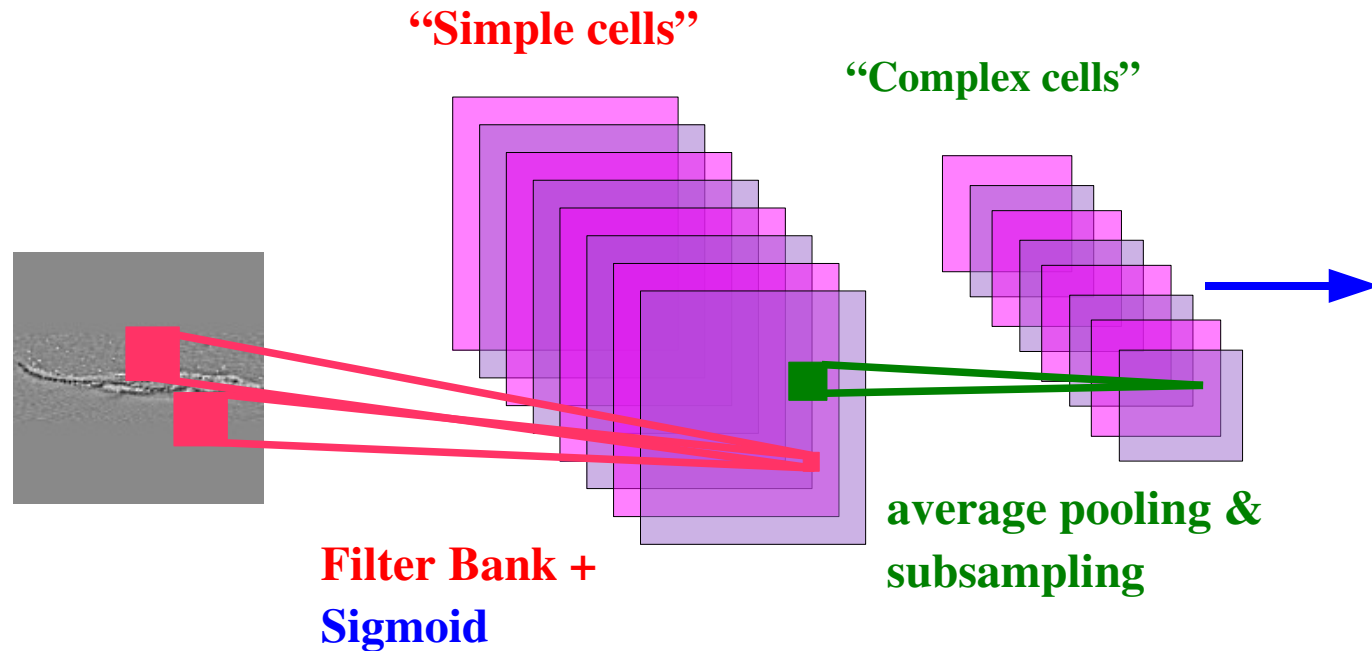
Geometric Blurr

Examples:

- ▶ SIFT features with Spatial Pyramid Matching Kernel SVM [Lazebnik et al. 2006]
- ▶ Edges + Rectification + Histograms + SVM [Dalal & Triggs 2005]

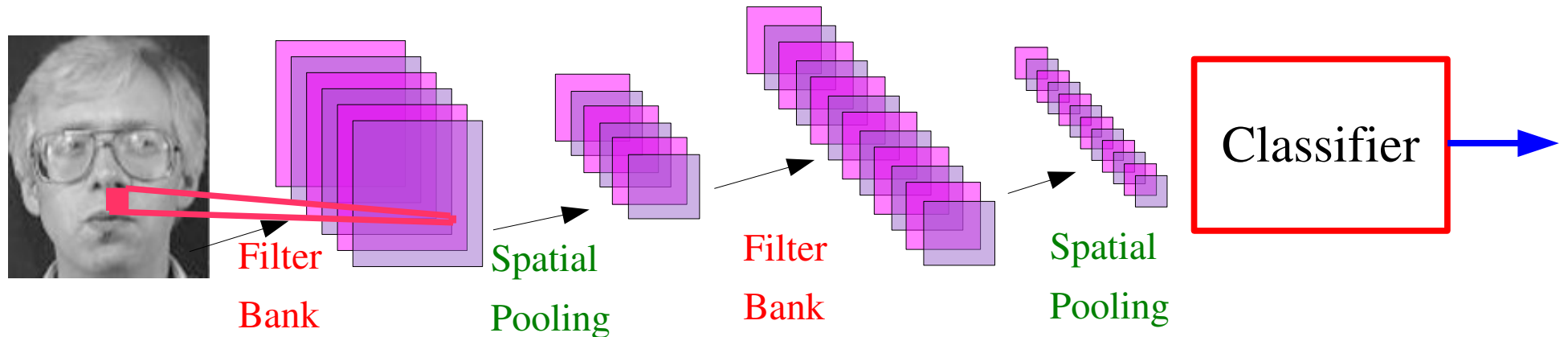
Fixed Features + “shallow” classifier

Trainable Feature Extraction: Hubel-Wiesel Stage



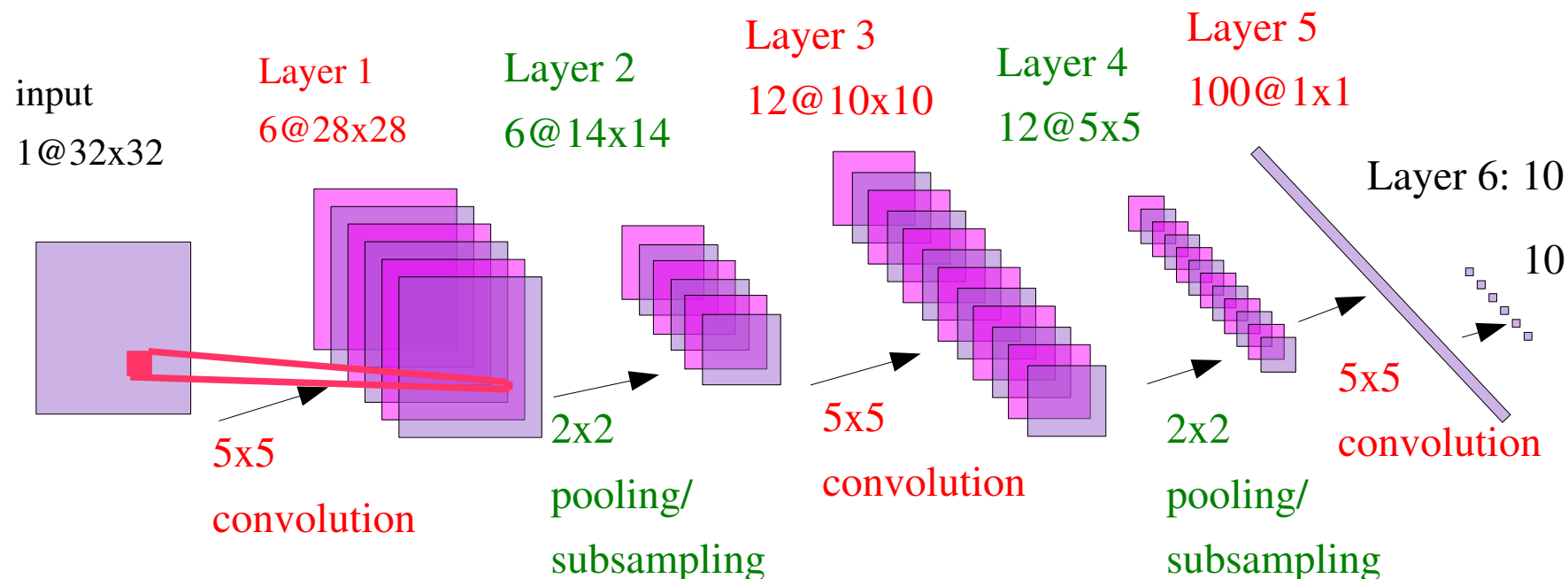
- Filter Bank -> Sigmoid -> Average Pooling & Subsampling
- Filter Bank + Sigmoid similar to “simple cells” in the visual cortex
- Pooling & Subsampling similar to “complex cells” in the visual cortex
 - ▶ [Hubel & Wiesel 1962]

Deep Architecture: The Multi-stage Hubel-Wiesel Architecture



- **Stacking multiple stages of simple cell / complex cell layer pairs**
- **We can't build the second layer features by hand!**
- **Neocognitron [Fukushima 1971-1982]**
 - ▶ simple unsupervised/competitive feature learning
- **Convolutional Nets [LeCun 1988-2007, Garcia 2004, Yu 2008,...]**
 - ▶ fully supervised feature learning
 - ▶ **A rare example of successful supervised deep learning**
- **HMAX & friends [Poggio's group 2002-2006, Lowe 2006]**
 - ▶ simple feature learning (fixed Gabor filters + stored templates)

Convolutional Net: Supervised Multi-Stage Hubel-Wiesel Arch.



Convolutional Net: supervised multi-stage Hubel-Wiesel Architecture

- ▶ Convolutional Layers: detect local motifs
- ▶ Pooling/Subsampling: builds local invariance to distortions

Training by supervised gradient descent (using back-propagation)

- ▶ Every coefficient of every filter is learned simultaneously
- ▶ “end-to-end learning”

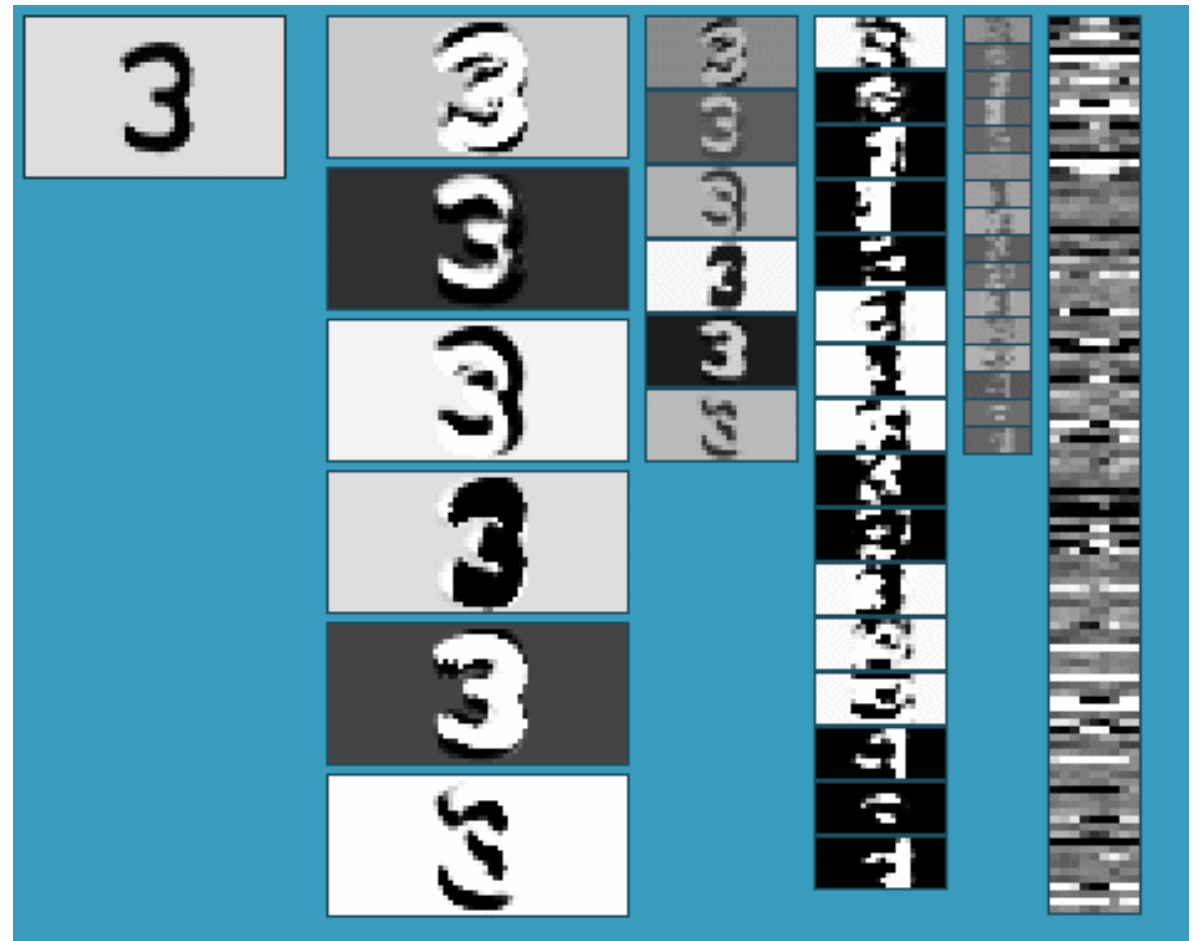
The architecture is biologically inspired, but not the learning algorithm.

Supervised Training of Convolutional Network

End-to-End Supervised Training of Convolutional Nets:

- ▶ Gradient-based learning algorithm (similar to back-propagation)
- ▶ Every filter at every layer is learned.

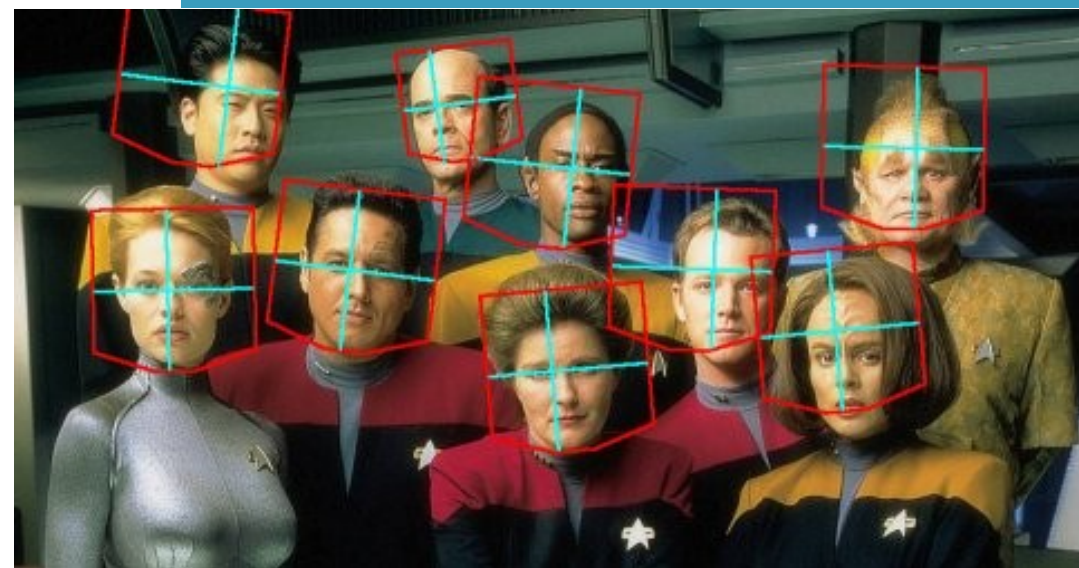
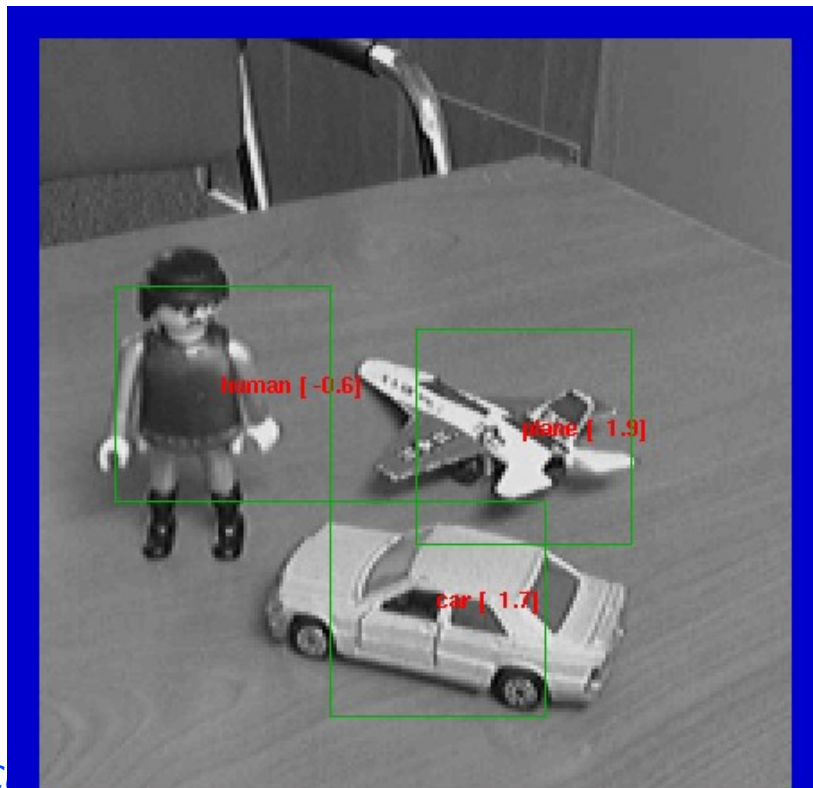
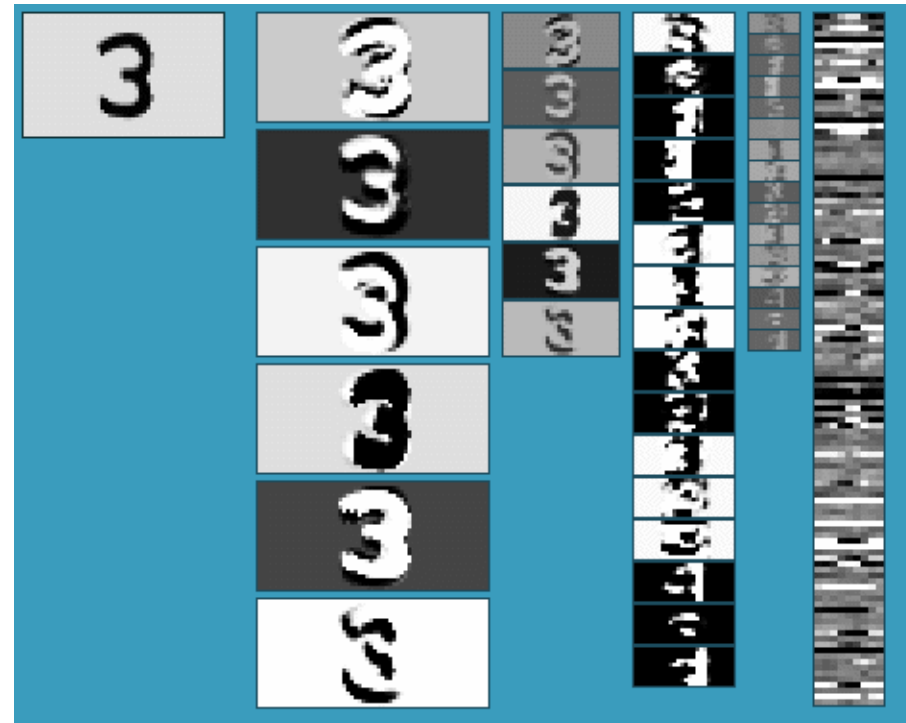
- This training method works very well but it requires many labeled training samples.
- It is the record-holding method for hand-writing recognition
- It is used commercially by NCR for check reading machines, and Microsoft for OCR.



Supervised Convolutional Nets learn well with lots of data

Supervised Convolutional nets work very well for:

- ▶ handwriting recognition
 - Holds the record on MNIST!
- ▶ face detection
- ▶ object recognition with few classes and lots of training samples

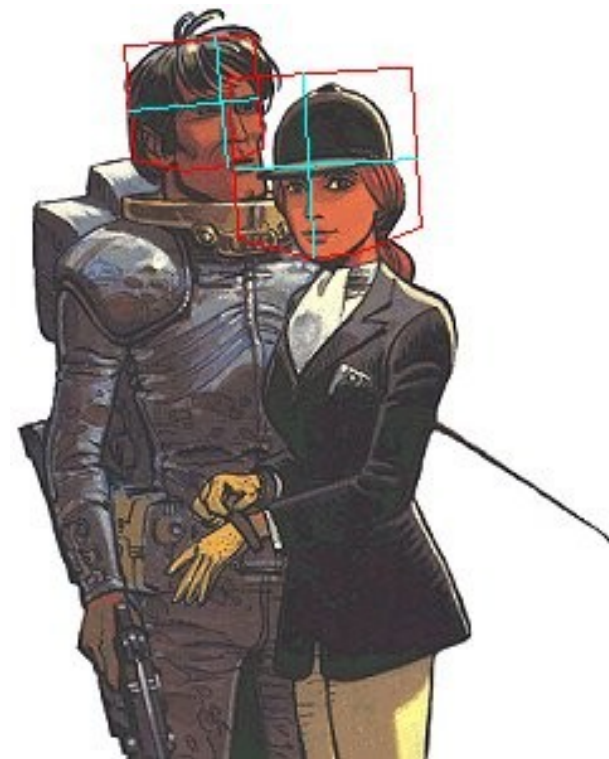
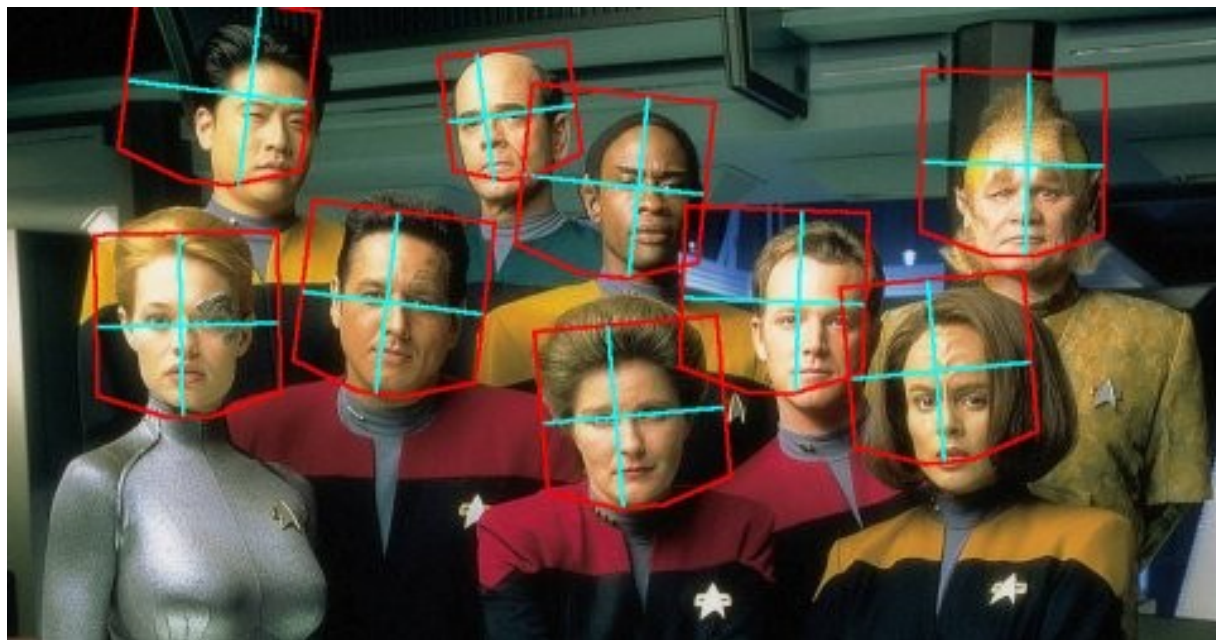


Deep Supervised ConvNets Work (with lots of labeled data)

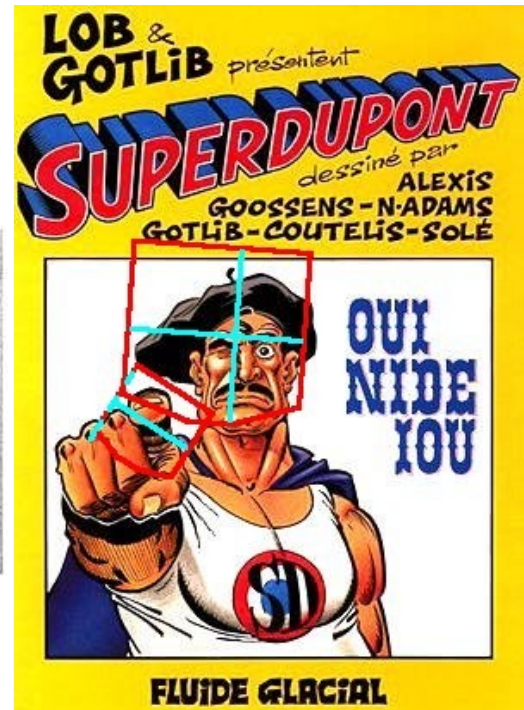
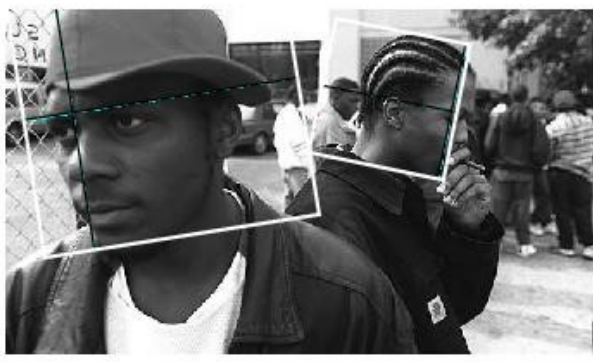
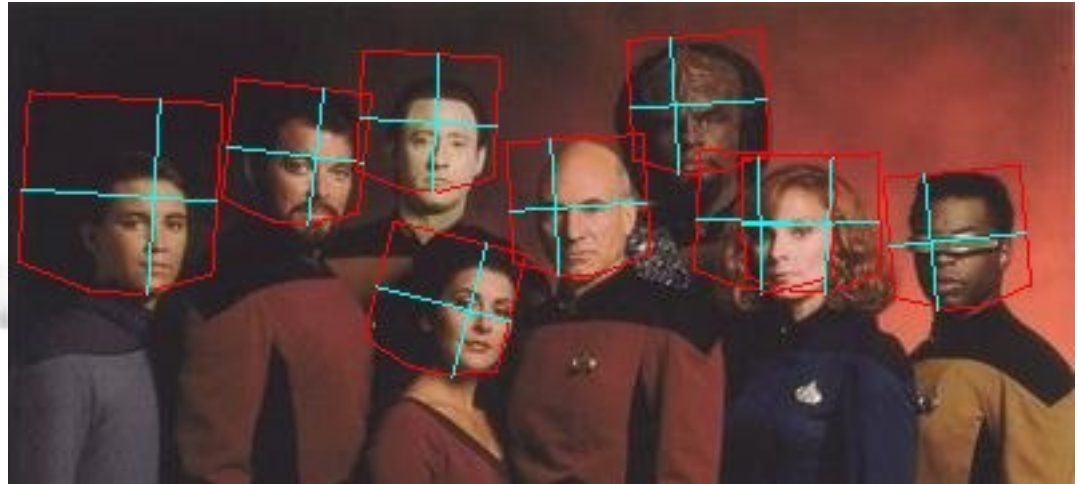
- On recognition tasks **with lots of training samples**, deep supervised architecture outperform shallow architectures in speed and accuracy
- Handwriting Recognition: ConvNets hold the record**
 - raw MNIST: 0.62% for convolutional nets [Ranzato 07]
 - raw MNIST: 1.40% for SVMs [Cortes 92]
 - distorted MNIST: 0.40% for conv nets [Simard 03, Ranzato 06]
 - distorted MNIST: 0.67% for SVMs [Bordes 07]
- Object Recognition: beats SVMs**
 - small NORB: 6.0% for conv nets [Huang 05]
 - small NORB: 11.6% for SVM [Huang 05]
 - big NORB: 7.8% for conv nets [Huang 06]
 - big NORB: 43.3% for SVM [Huang 06]
- Face Detection: ConvNets beat Viola-Jones**
 - [Vaillant 93,94][Garcia & Delakis PAMI 05][Osadchy JMLR 07]

Face Detection: Results

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



Face Detection and Pose Estimation: Results



Face Detection with a Convolutional Net



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

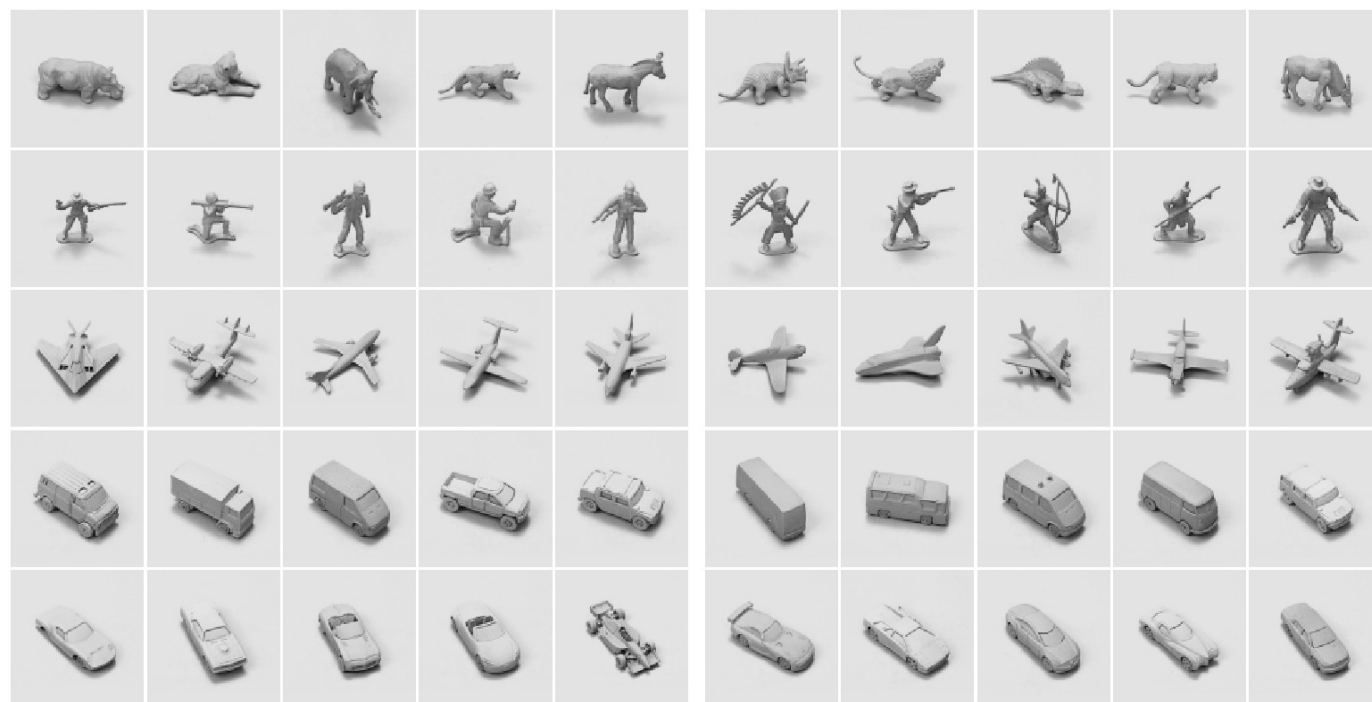
6 illuminations

on/off combinations of 4 lights

2 cameras (stereo)

7.5 cm apart

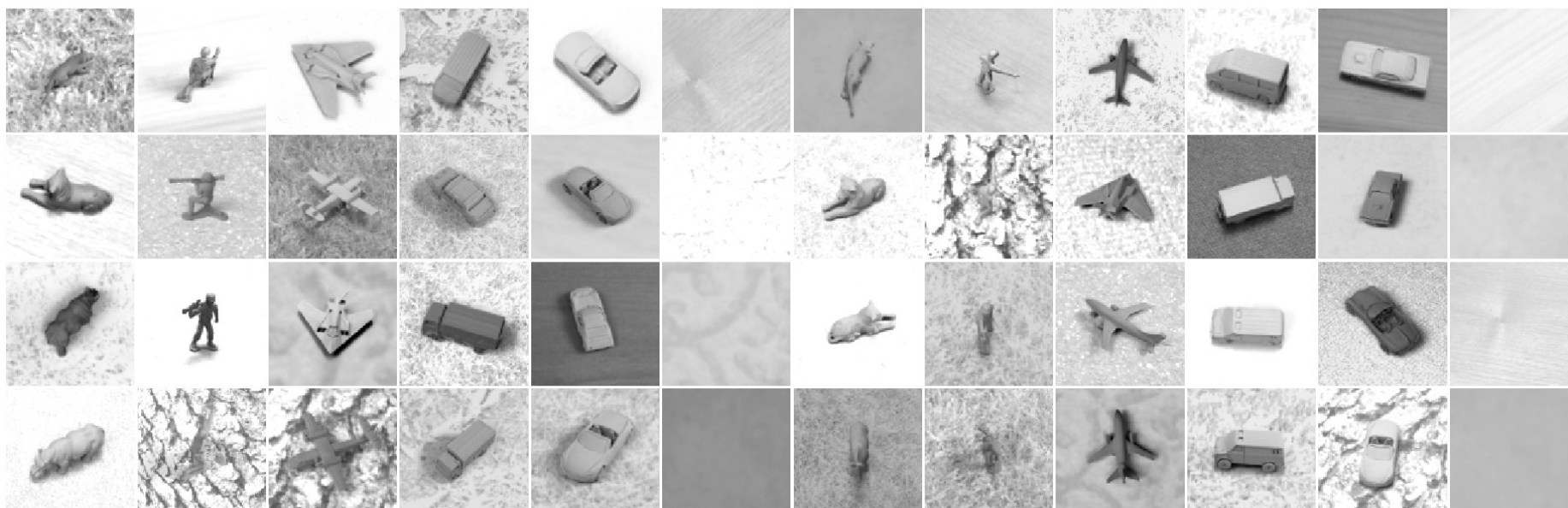
40 cm from the object



Training instances

Test instances

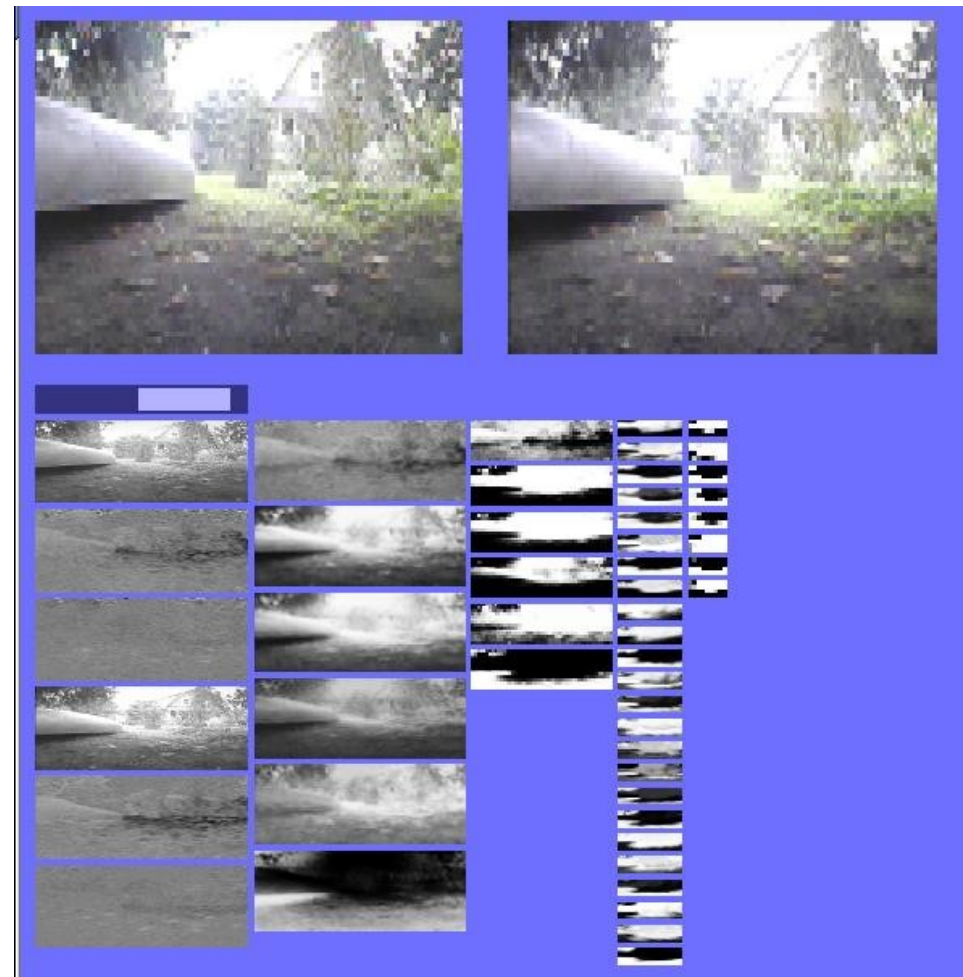
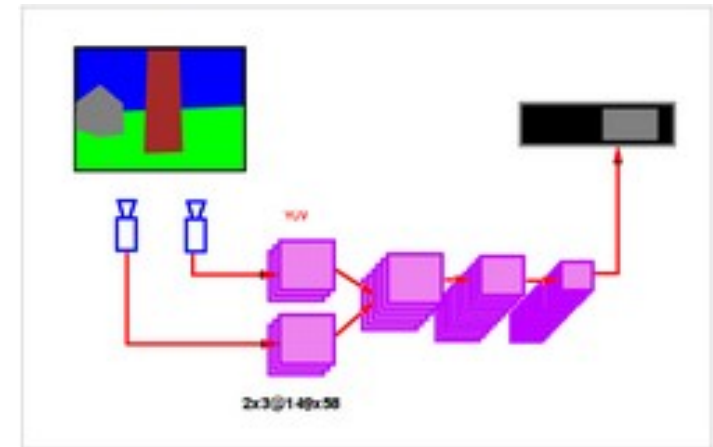
Textured and Cluttered Datasets



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)

● Vidient Inc

- ▶ Vidient Inc's "SmartCatch" system deployed in several airports and facilities around the US for detecting intrusions, tailgating, and abandoned objects (Vidient is a spin-off of NEC)

● NEC Labs

- ▶ Cancer cell detection, automotive applications, kiosks

● Google

- ▶ OCR, ???

● Microsoft

- ▶ OCR, handwriting recognition, speech detection

● France Telecom

- ▶ Face detection, HCI, cell phone-based applications

● Other projects: HRL (3D vision)....

Problem: ConvNets don't work when labeled samples are scarce

- 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: 26.0%
- ▶ SIFT features + spatial pyramid kernel SVM: 66.2%
- [Lazebnik et al. 2006]

- When learning the features, there are simply too many parameters to learn in purely supervised mode (or so we thought).**

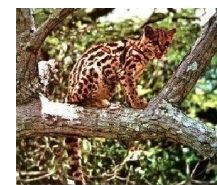
face



beaver



wild cat



lotus



ant



dollar



metronome



w. chair



minaret



cellphone



joshua t.



cougar body



background



Great Satisfaction
can be found in simple things...

How Can We Use Unlabeled Data to Learn Features?

- We need **unsupervised** learning methods that can learn invariant feature hierarchies
- **“Deep Belief Networks” strategy [Hinton 2005]**
 - ▶ train each stage unsupervised one after the other.
 - ▶ Hinton uses Restricted Boltzmann Machines for each stage.

[Hinton et al. “A fast learning algorithm for DBNs” 06]

[Hinton et al. “Reducing the dimensionality of data with neural nets” 06]

[Bengio et al. “Greedy layer-wise training of deep nets” 07]

[Ranzato et al. “Efficient learning of sparse representations with energy-based models” 07]

[Lee et al. 07]

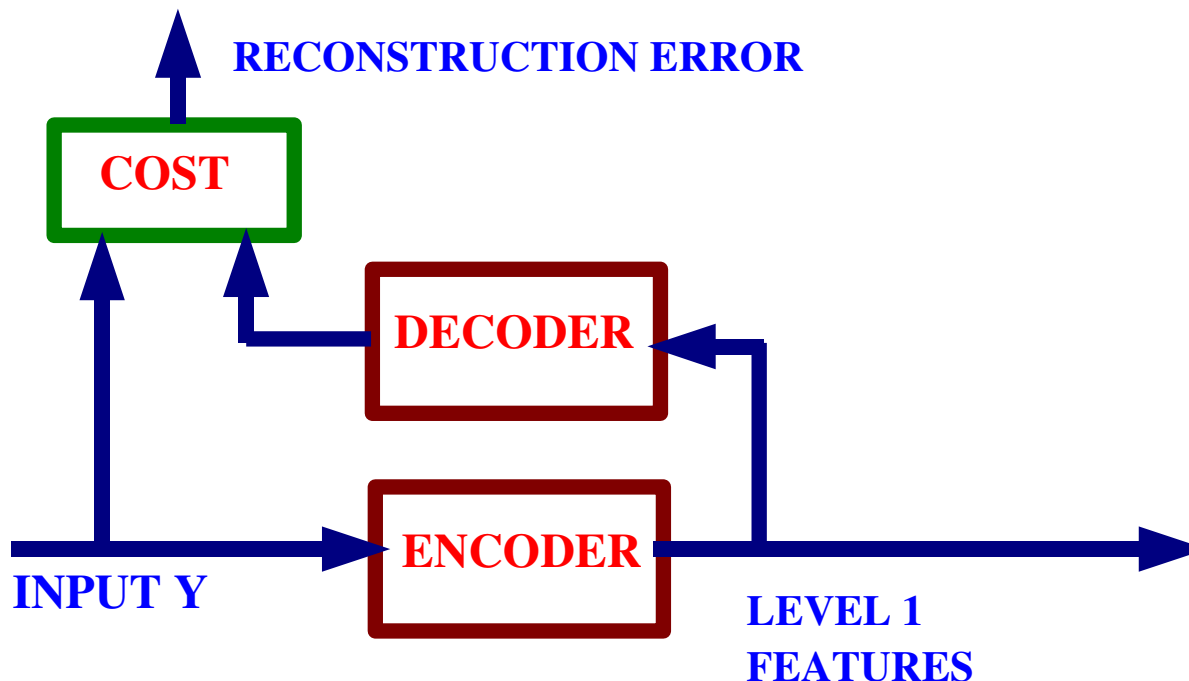
The Deep Encoder/Decoder Architecture

- **Each stage is composed of** [Bengio 06, LeCun 06]
 - ▶ an encoder that produces a feature vector from the input
 - ▶ a decoder that reconstruct the input from the feature vector
 - Hinton's Restricted Boltzmann Machines are a special case
- **Each stage is trained one after the other in a greedy fashion**
 - ▶ the input to stage $k+1$ is the feature vector of stage k .



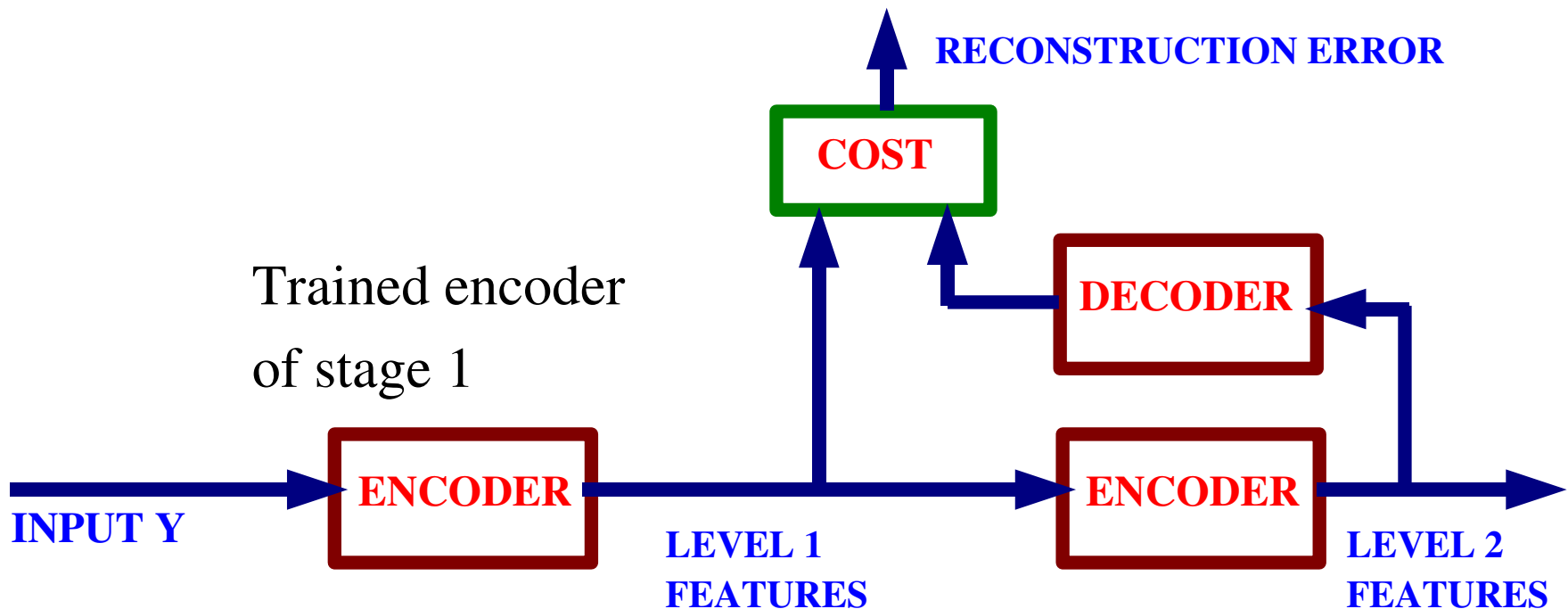
The Deep Encoder/Decoder Architecture

- 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
 - Hinton's Restricted Boltzmann Machines are a special case
- Each stage is trained one after the other
 - Training stage 1



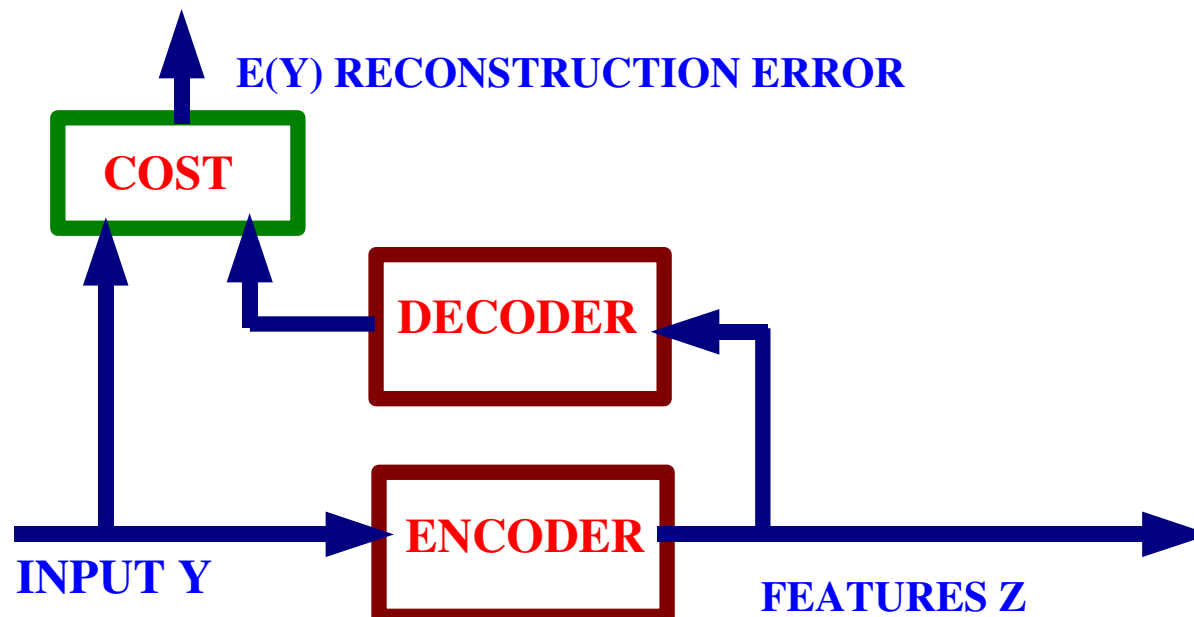
The Deep Encoder/Decoder Architecture

- 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
 - Hinton's Restricted Boltzmann Machines are a special case
- Each stage is trained one after the other
 - Training stage 2



Training an Encoder/Decoder Module

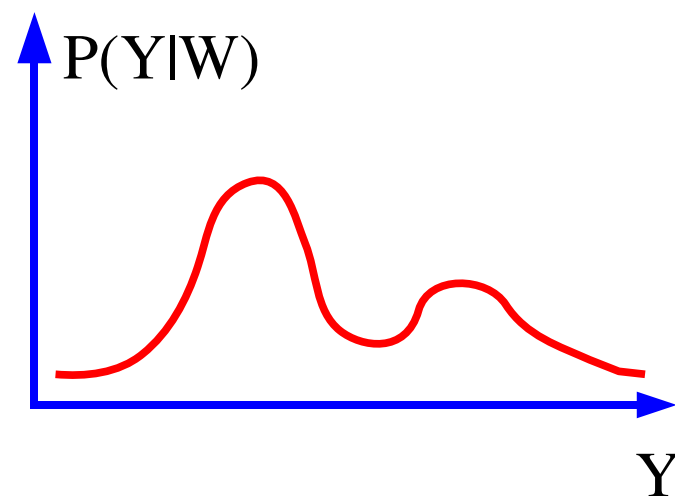
- Define the Energy $E(Y)$ as the reconstruction error
 - Example: $E(Y) = || Y - \text{Decoder}(\text{Encoder}(Y)) ||^2$
- Probabilistic Training, given a training set (Y_1, Y_2, \dots)
 - Interpret the energy $E(Y)$ as a $-\log P(Y)$ (unnormalized)
 - Train the encoder/decoder to maximize the prob of the data
- Train the encoder/decoder so that:
 - $E(Y)$ is small in regions of high data density (good reconstruction)
 - $E(Y)$ is large in regions of low data density (bad reconstruction)



Each Stage is Trained as an Estimator of the Input Density

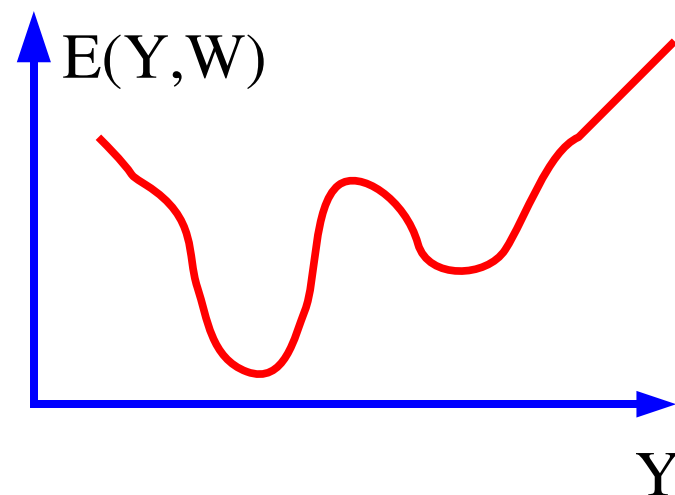
Probabilistic View:

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



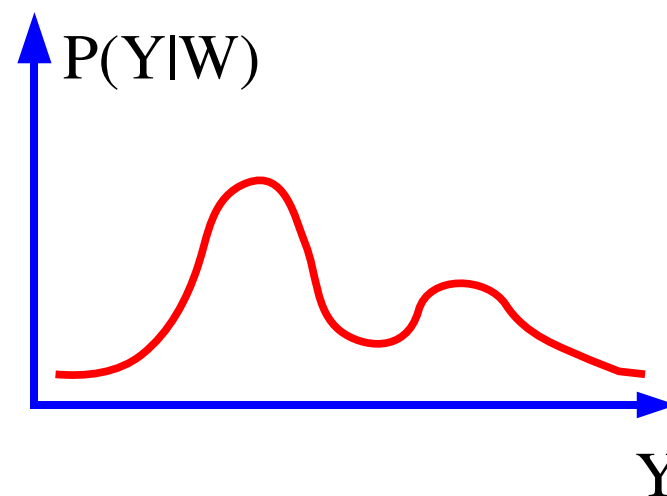
Energy-Based View:

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

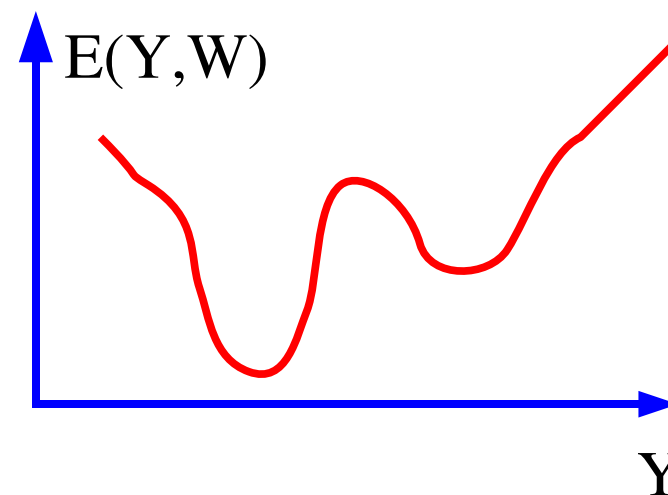


Energy <-> Probability

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



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

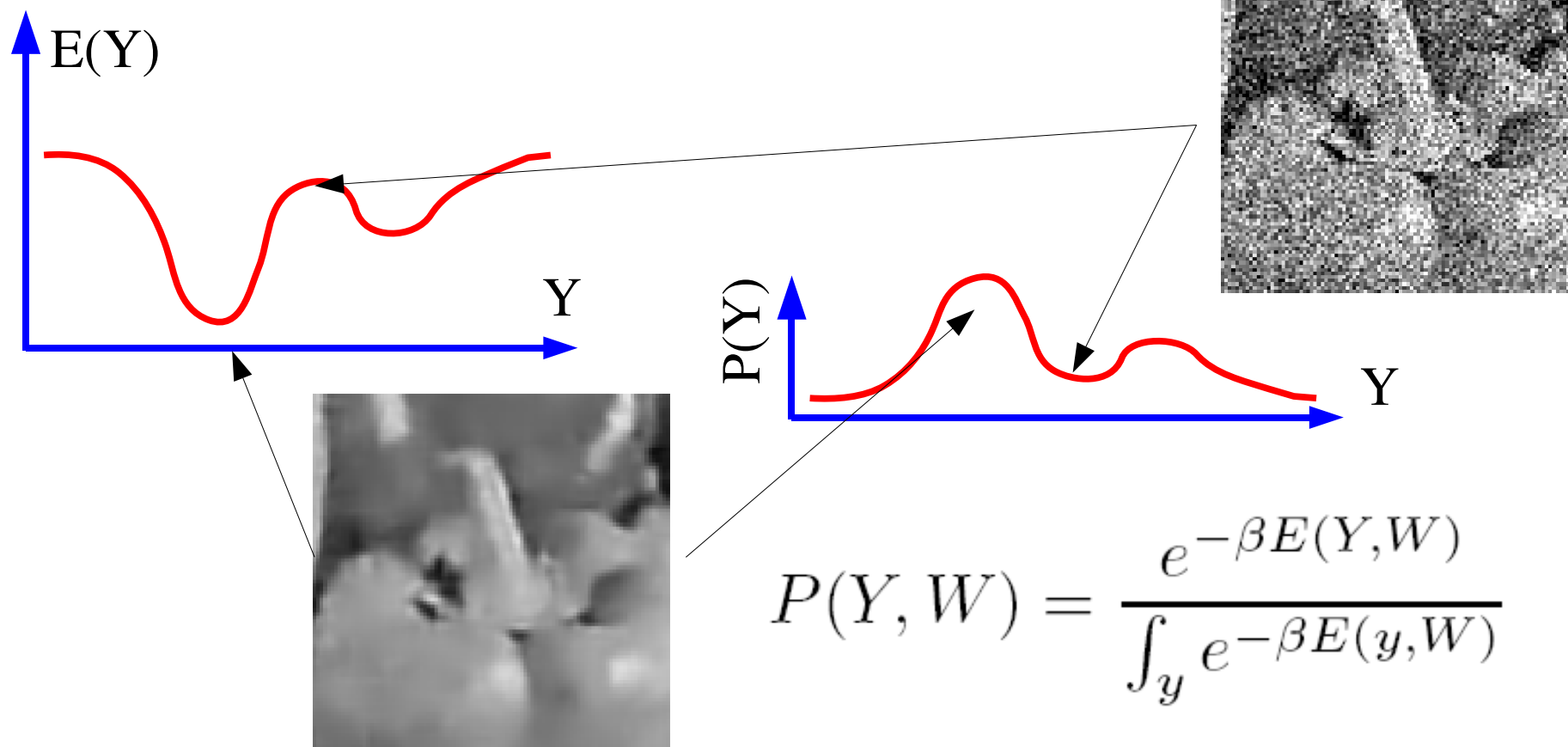


The Intractable Normalization Problem

Example: Image Patches

Learning:

- ▶ Make the energy of every “natural image” patch low
- ▶ Make the energy of everything else high!



Training an Energy-Based Model to Approximate a Density

Maximizing $P(Y|W)$ on training samples

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

make this big

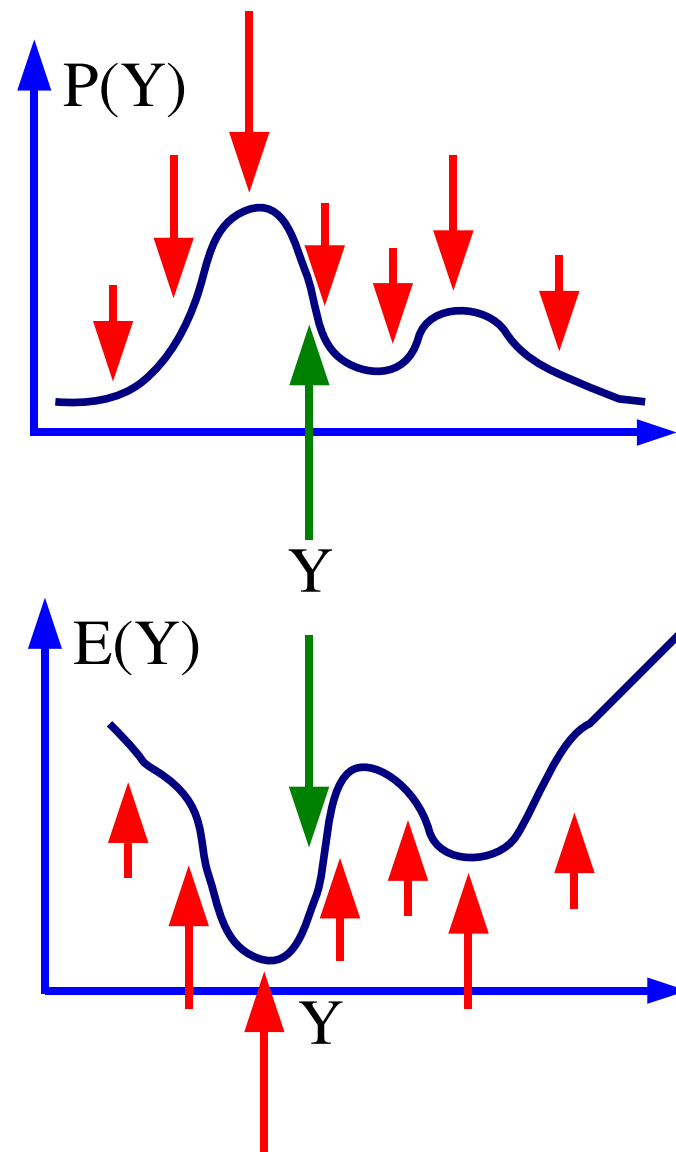
make this small

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 :

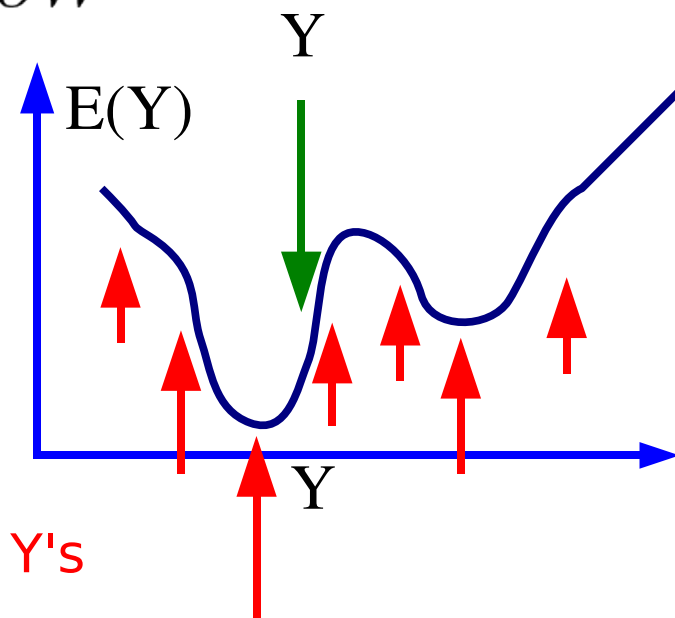
$$\frac{\partial L(Y, W)}{\partial W} = \frac{\partial E(Y, W)}{\partial W} - \int_y P(y|W) \frac{\partial E(y, W)}{\partial W}$$

- 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



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

Solving The Intractable Normalization problem

• Probabilistic unsupervised learning is hard

- ▶ Pushing up on the energy of every points in regions of low data density is often impractical.

• Solution 1: contrastive divergence [Hinton 2000]

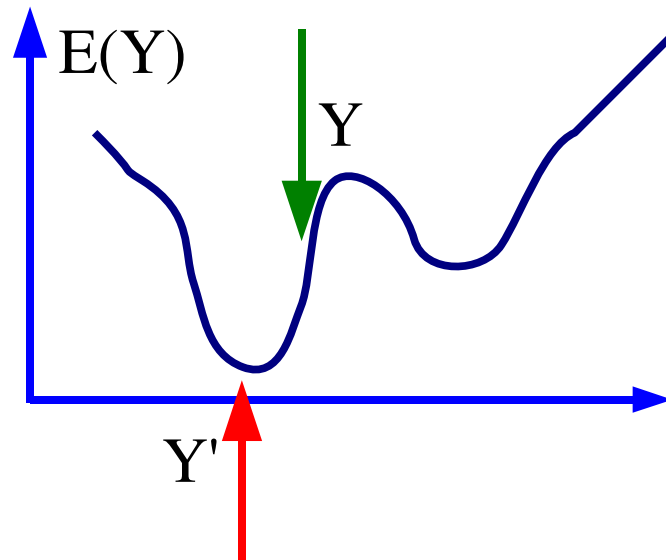
- ▶ Only push up on points that are not too far from the training samples, and only on those points that have low energy. These points are obtained from the training samples through MCMC.
- ▶ This makes a “groove” in the energy surface around the data manifold.

• Solution 2: **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 Y s can be perfectly reconstructed**
- ▶ Idea: impose a sparsity prior on Z
- ▶ This is reminiscent of sparse coding [Olshausen & Field 1997]

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



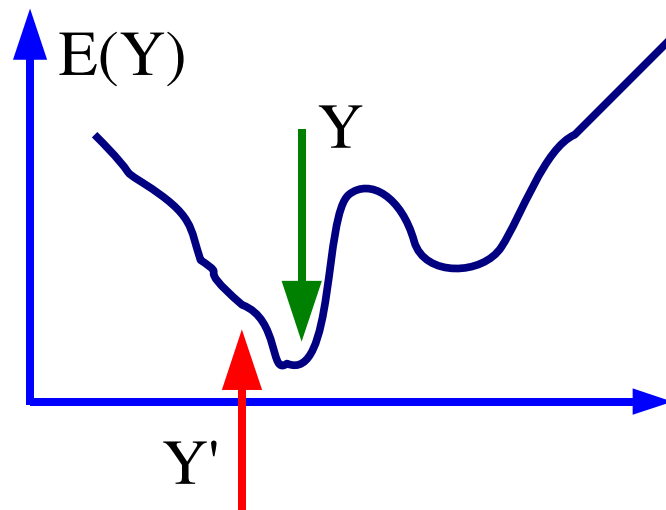
$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \frac{\partial E(Y', W)}{\partial W}$$

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

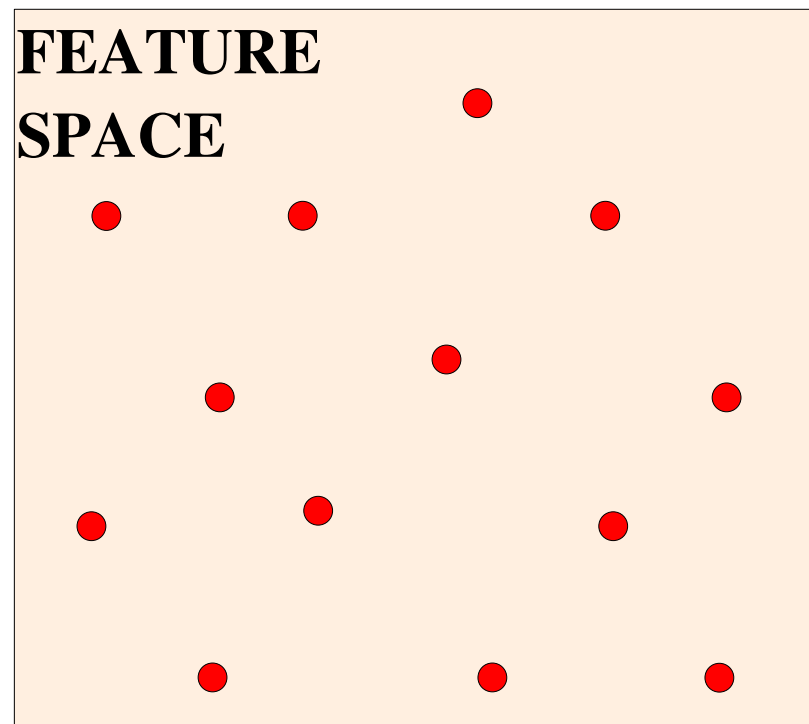
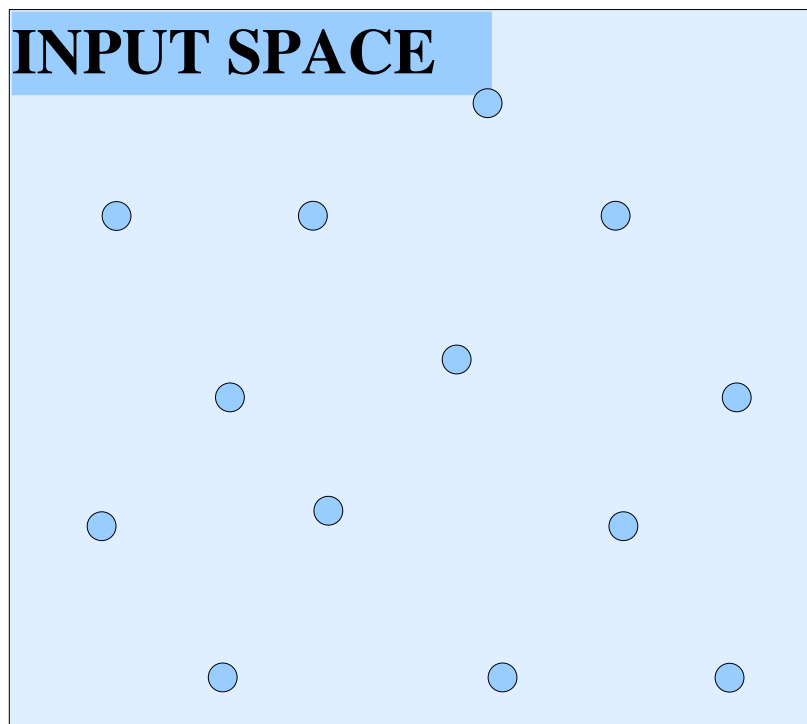
Pulls up on the energy Y'

The Main Insight [Ranzato et al. 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?

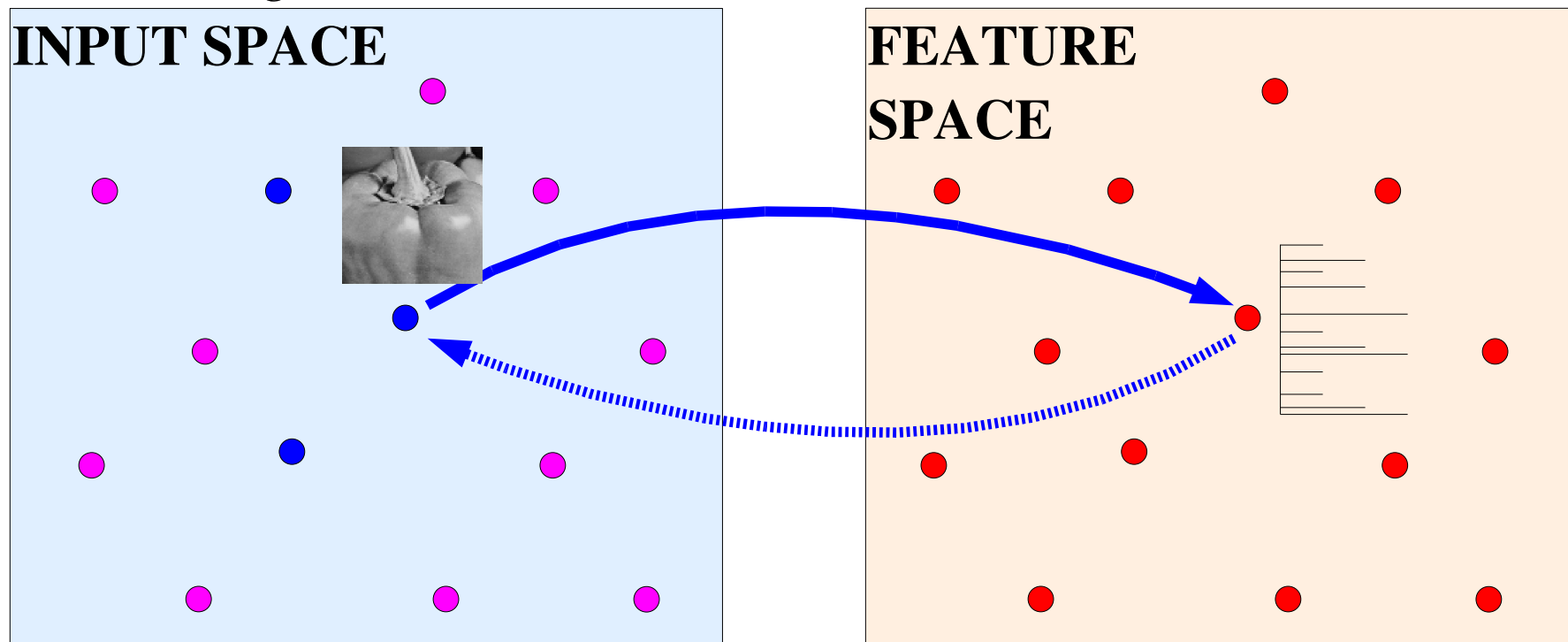
- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**



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

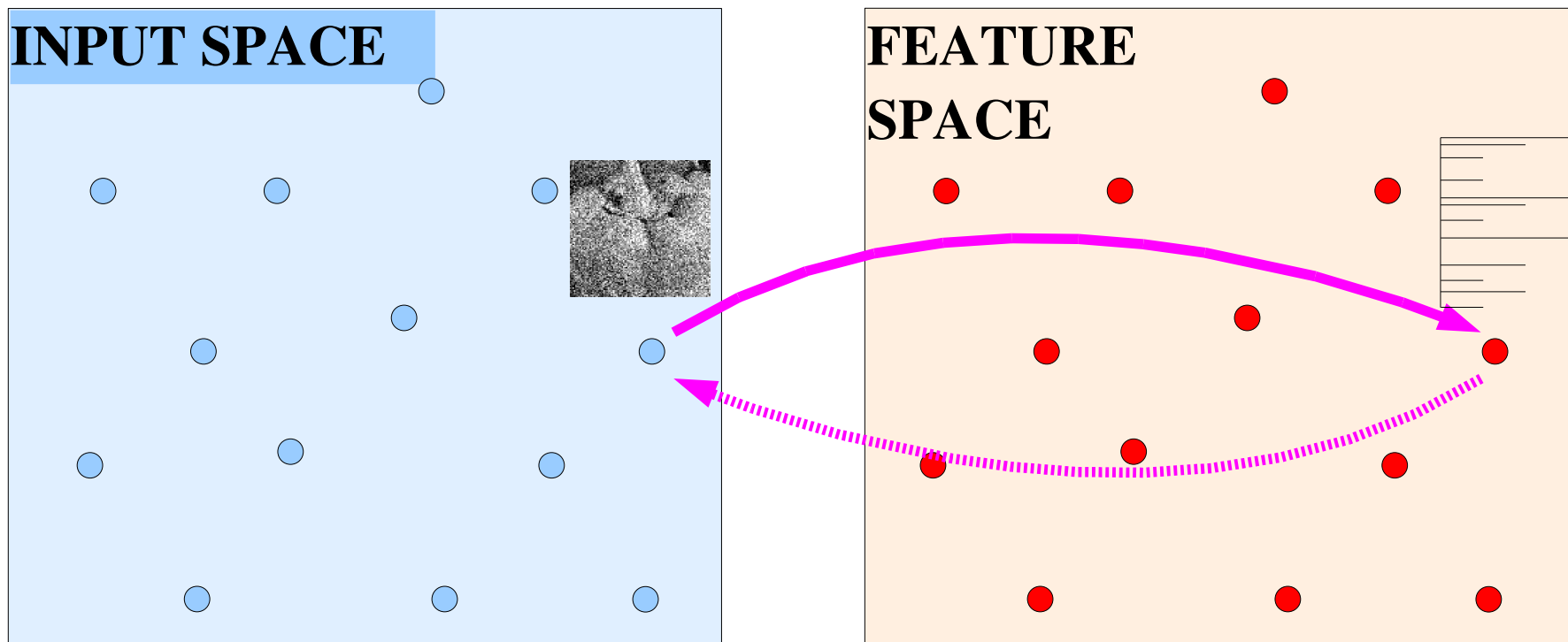


Why Limit the Information Content of the Code?

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

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

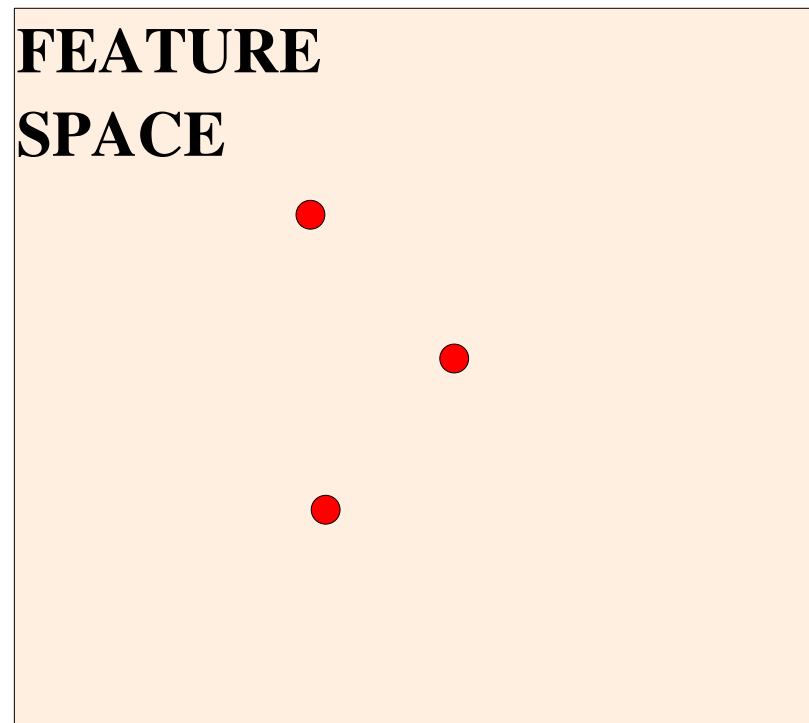
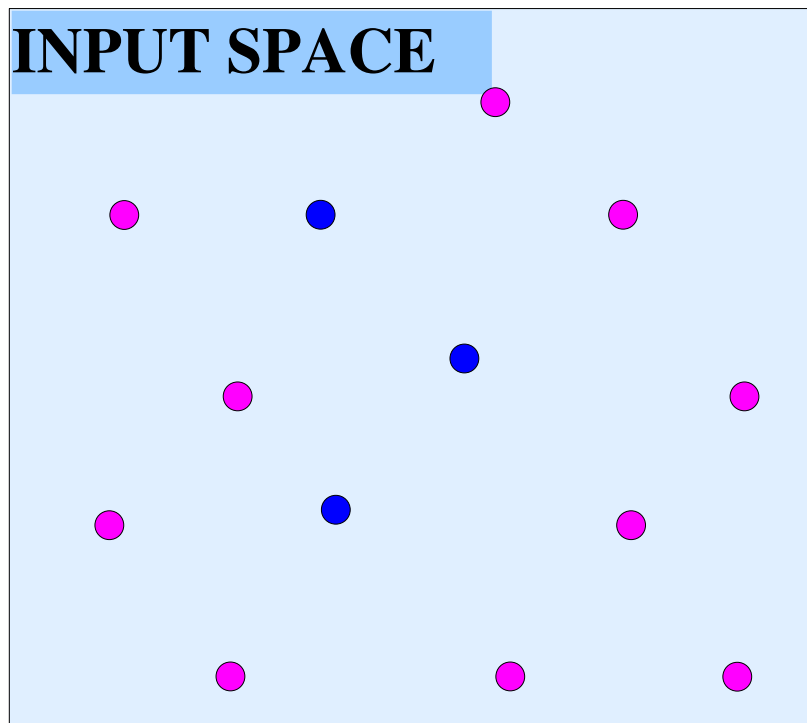
It just copies the data.



Why Limit the Information Content of the Code?

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

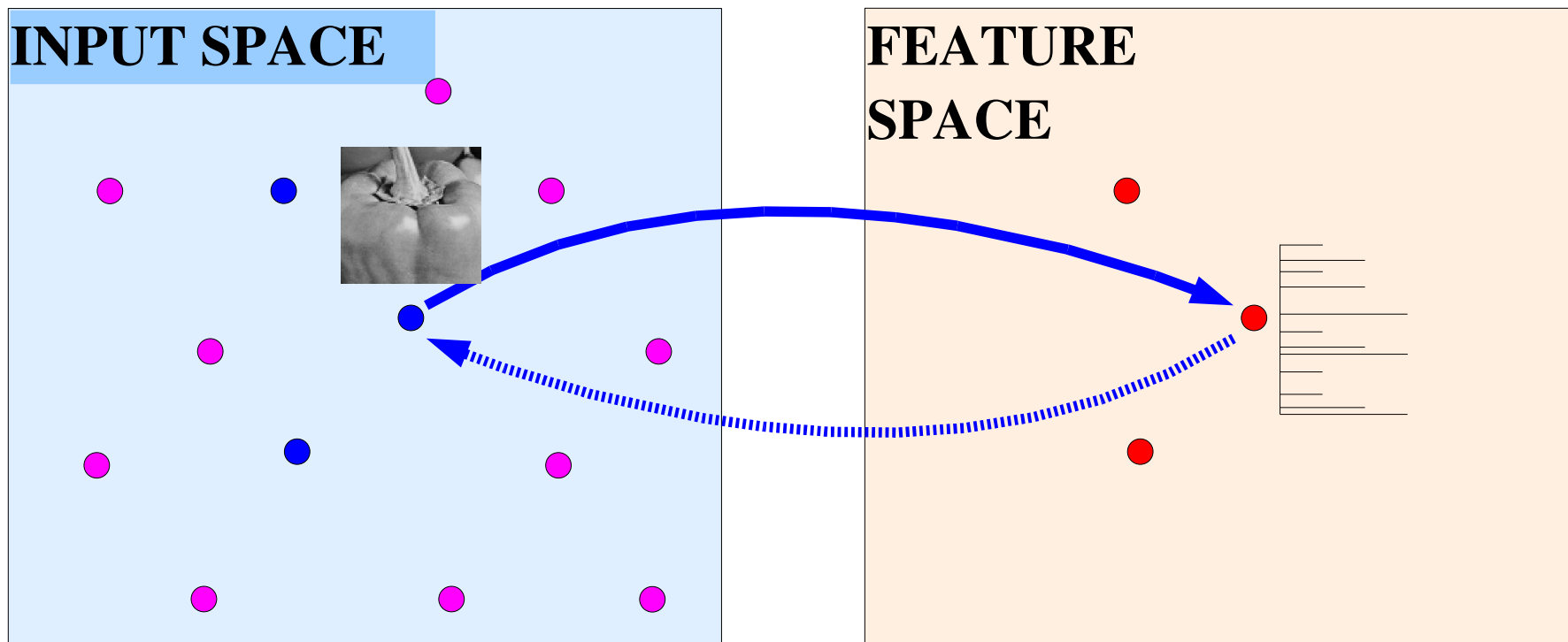
IDEA: reduce number of available codes.



Why Limit the Information Content of the Code?

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

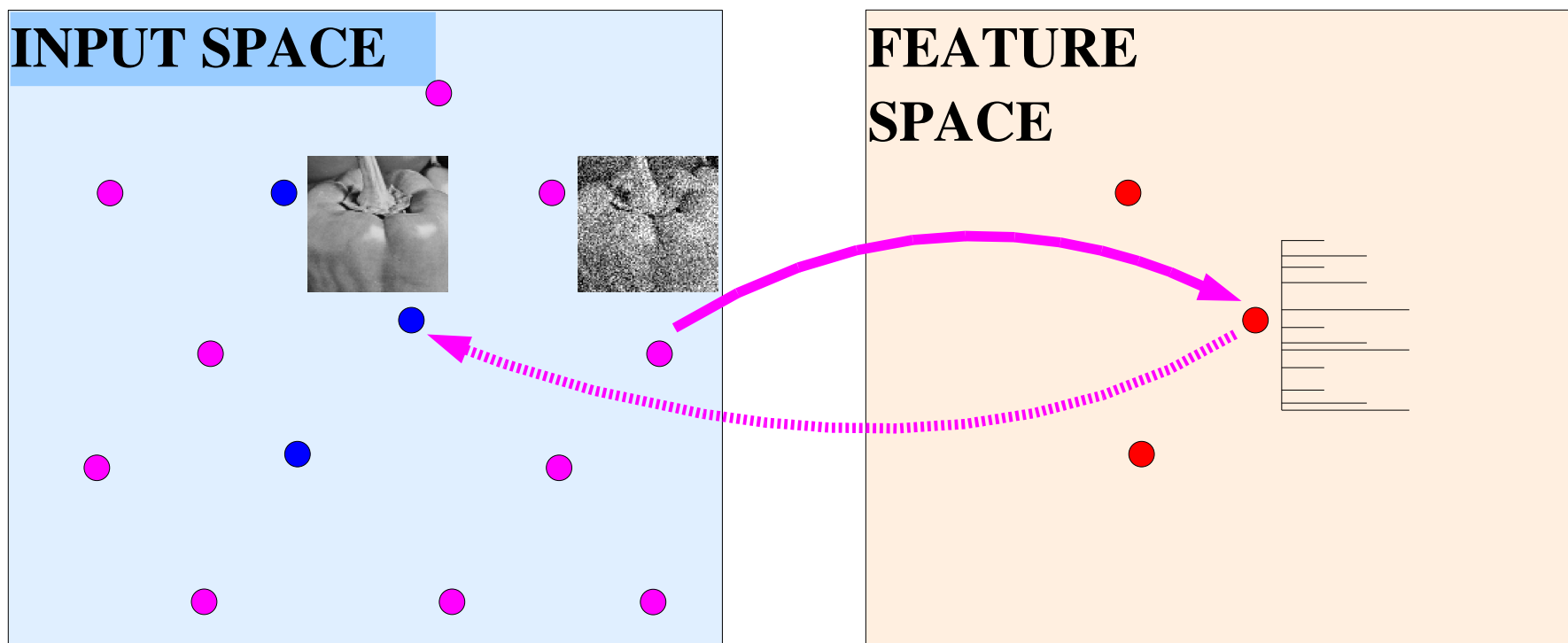
IDEA: reduce number of available codes.



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.

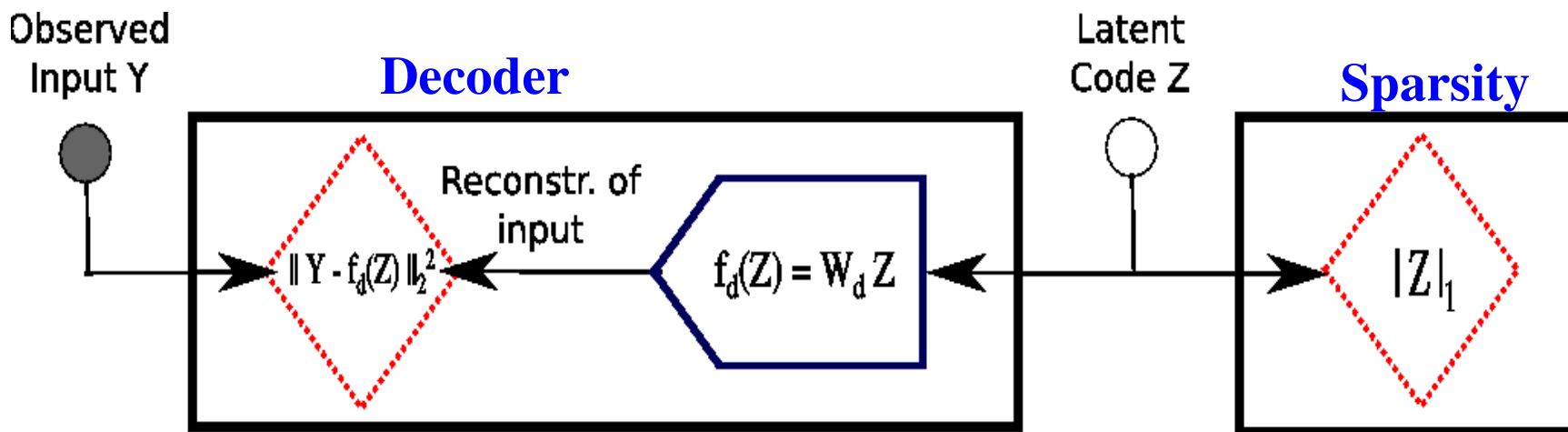


Sparse Codes are Good

- **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.

Sparse Decomposition with Linear Reconstruction

- **Energy**(Input,Code) = $\| \text{Input} - \text{Decoder}(\text{Code}) \|^2 + \text{Sparsity}(\text{Code})$
- **Energy**(Input) = $\text{Min_over_Code}[\text{Energy}(\text{Input},\text{Code})]$



- **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)]**

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

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

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

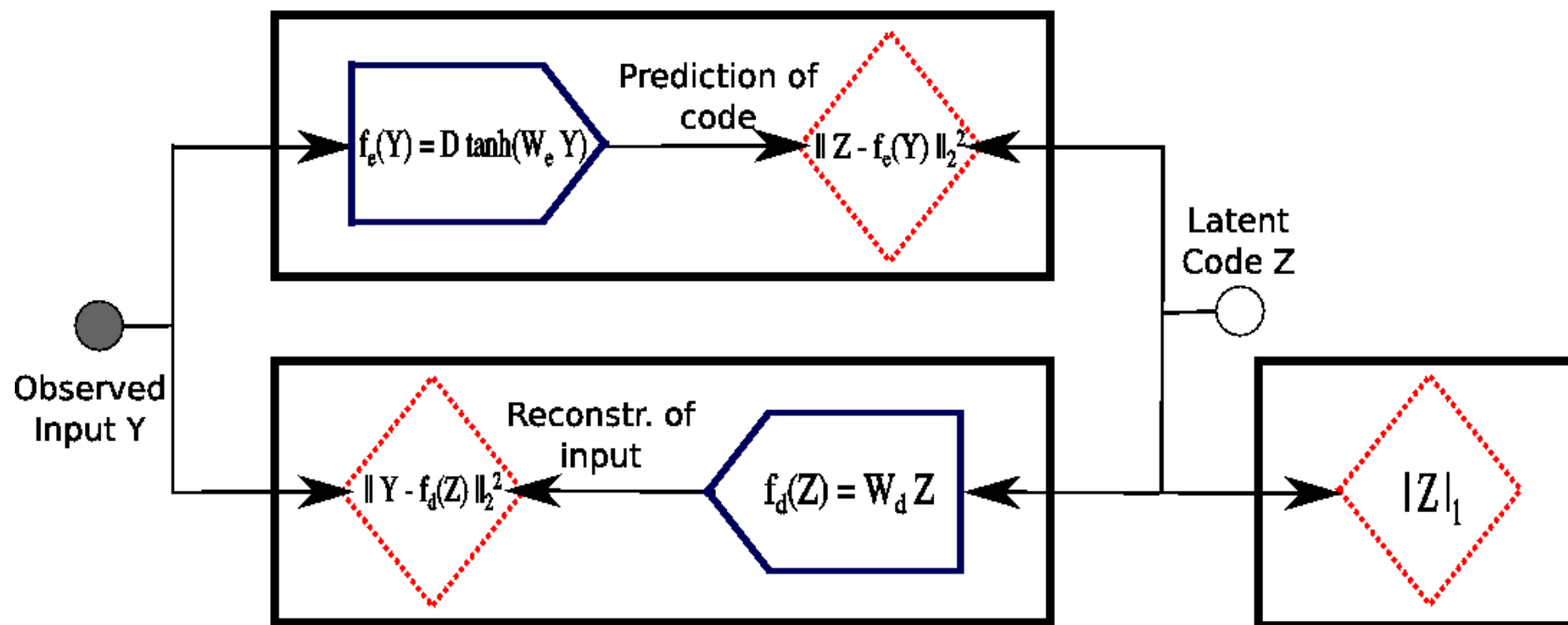
- **For each new Y, an optimization algorithm must be run to find the corresponding optimal Z**
- **This would be very slow for large scale vision tasks**
- **Also, the optimal Z are very unstable:**
 - ▶ A small change in Y can cause a large change in the optimal Z

Solution: Predictive Sparse Decomposition (PSD)

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

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

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

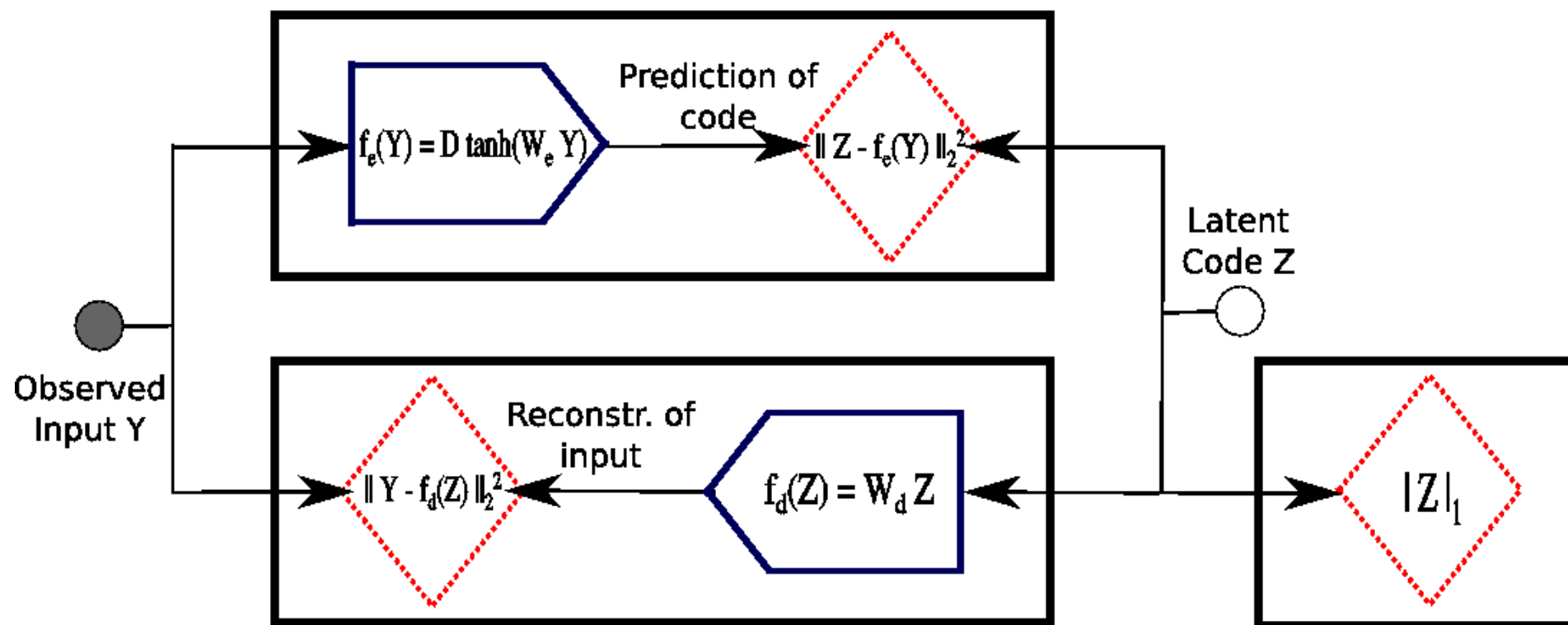


PSD: Inference

- Inference by gradient descent starting from the encoder output

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

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

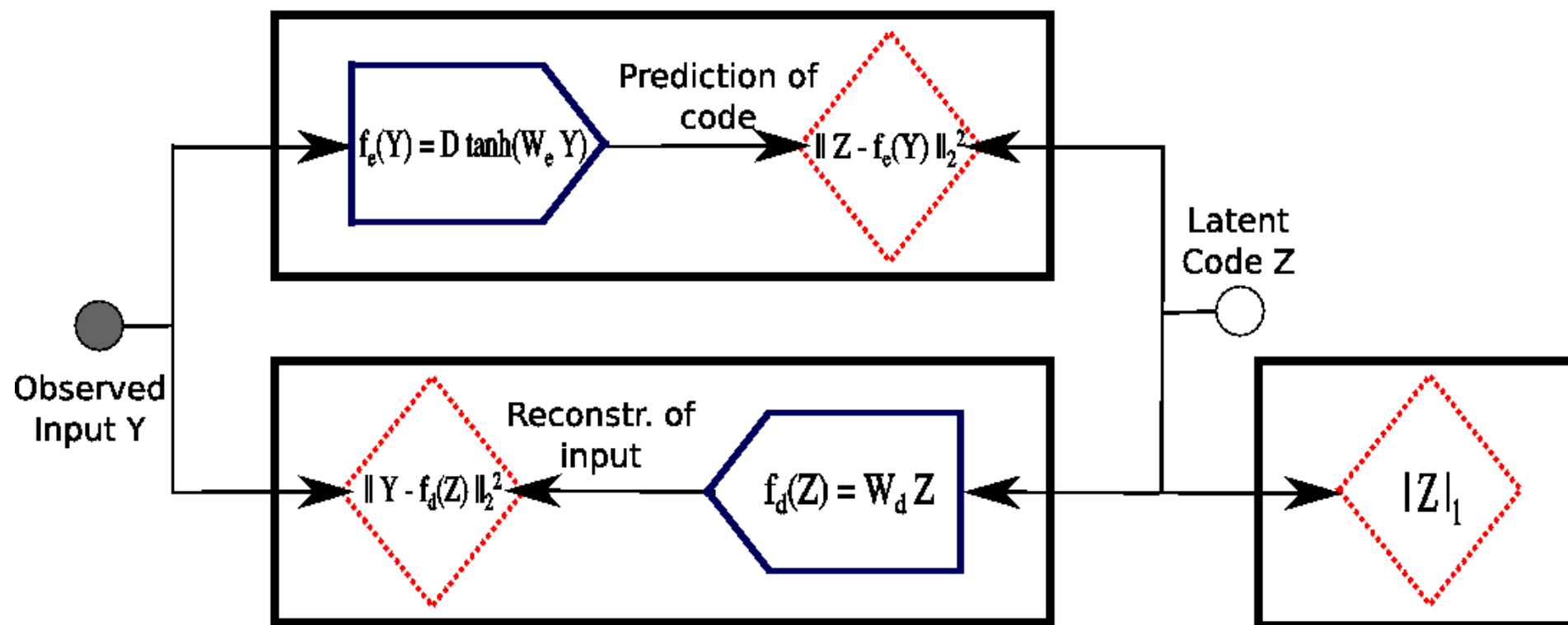


PSD: Learning [Kavukcuoglu et al. 2009]

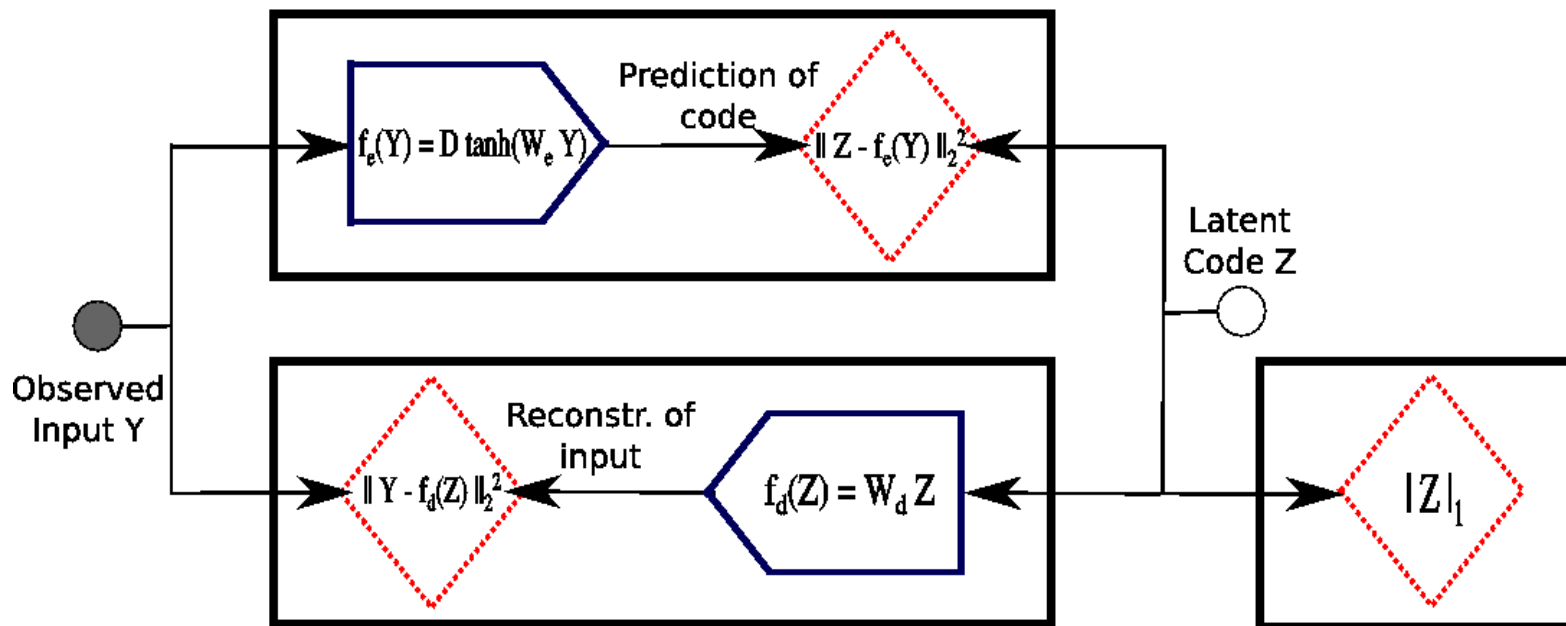
- Learning by minimizing the average energy of the training data with respect to W_d and W_e .

- Loss function:** $L(W) = \sum_i F(Y^i; W)$

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



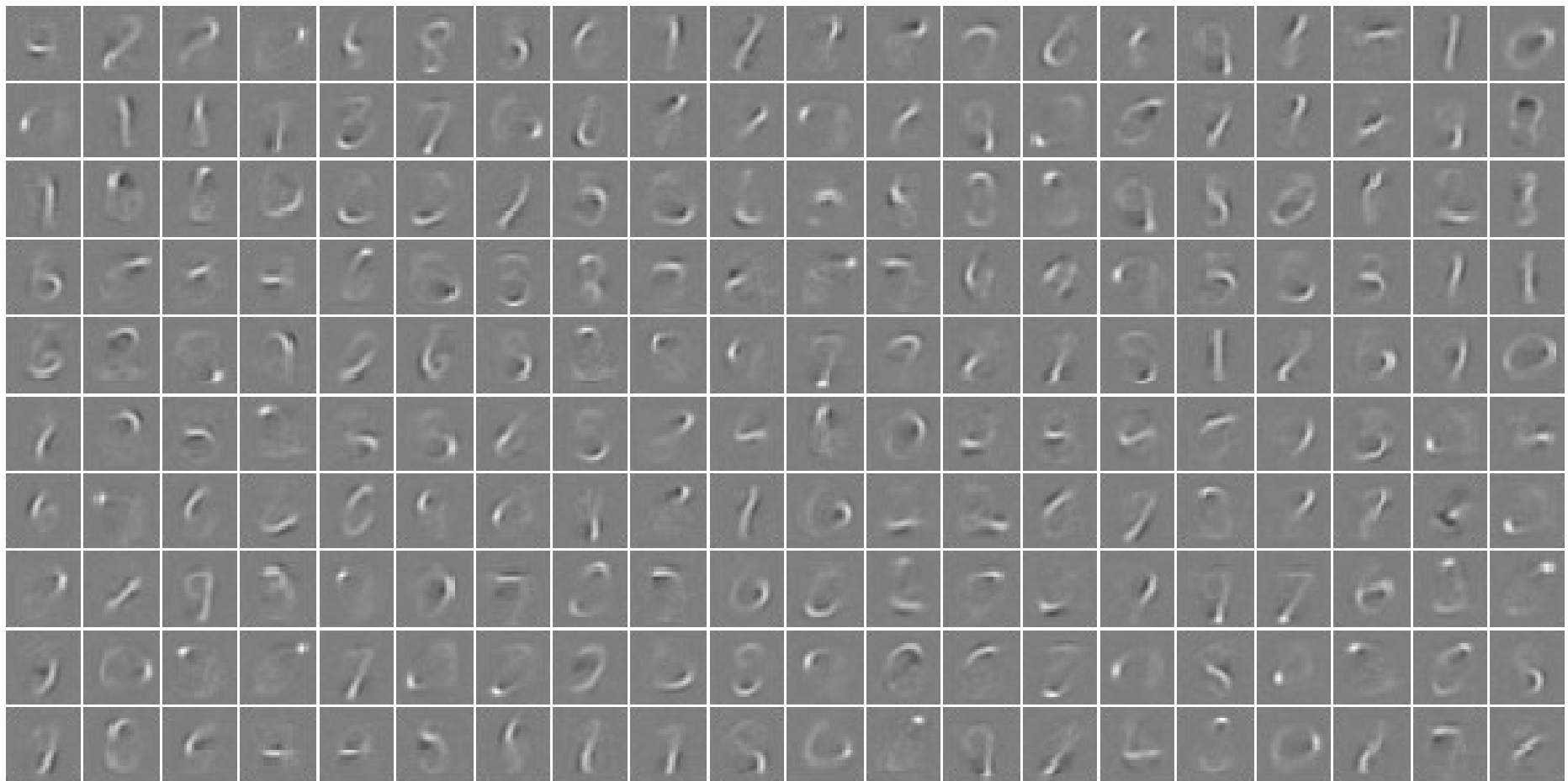
PSD: Learning Algorithm



1. Initialize $Z = \text{Encoder}(Y)$
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

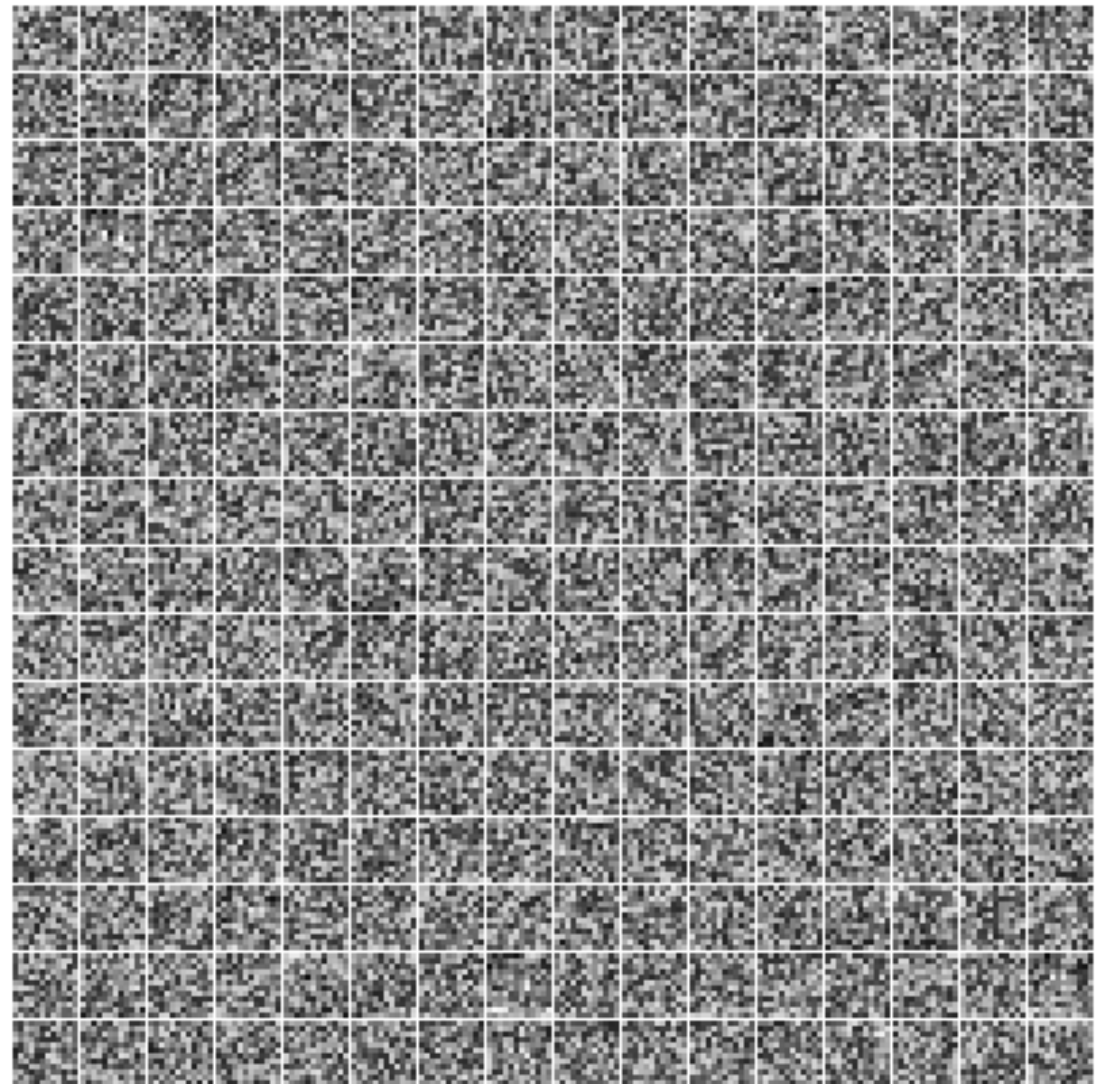
Decoder Basis Functions on MNIST

- 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

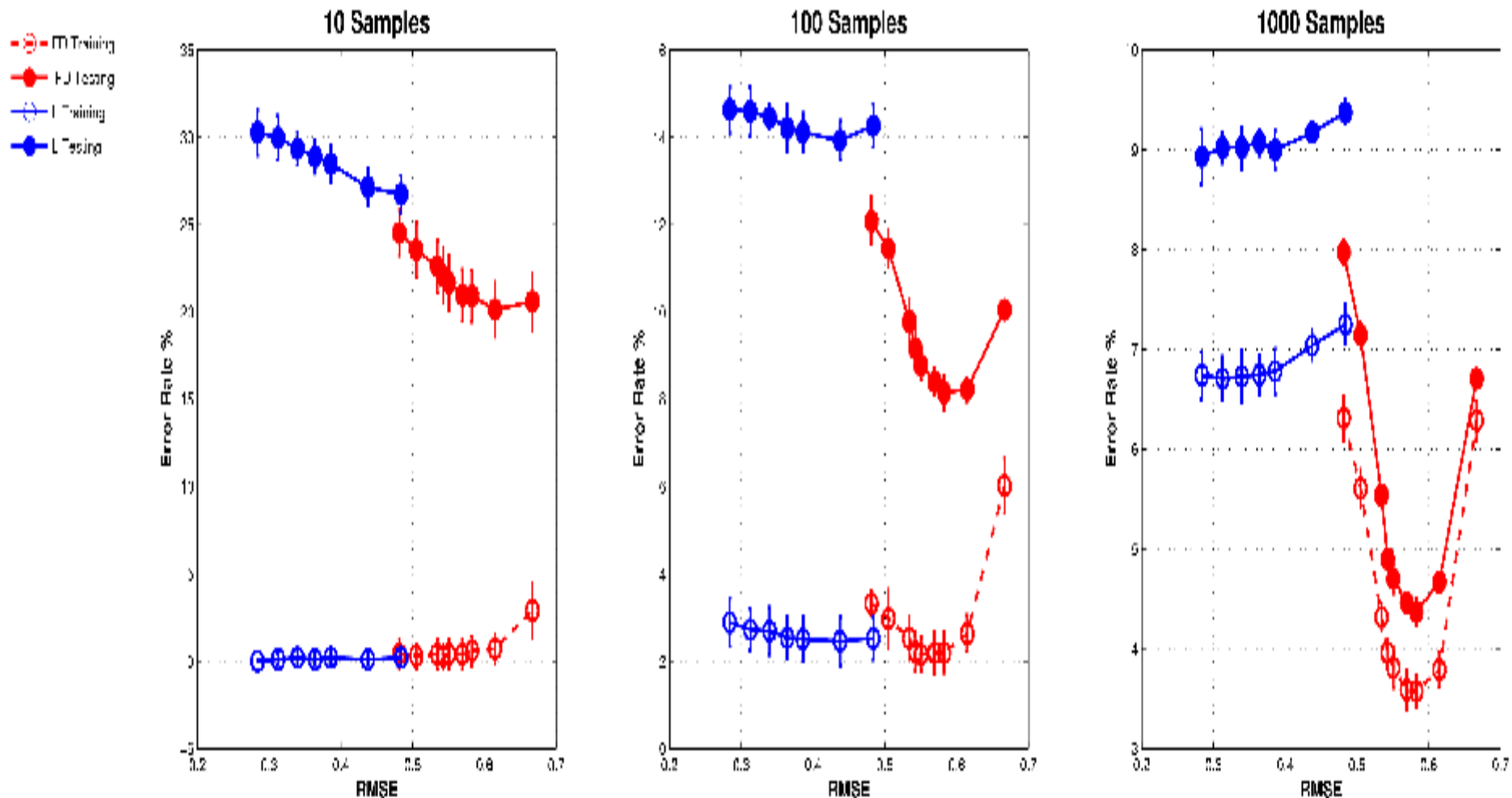


iteration no 0

Classification Error Rate on MNIST

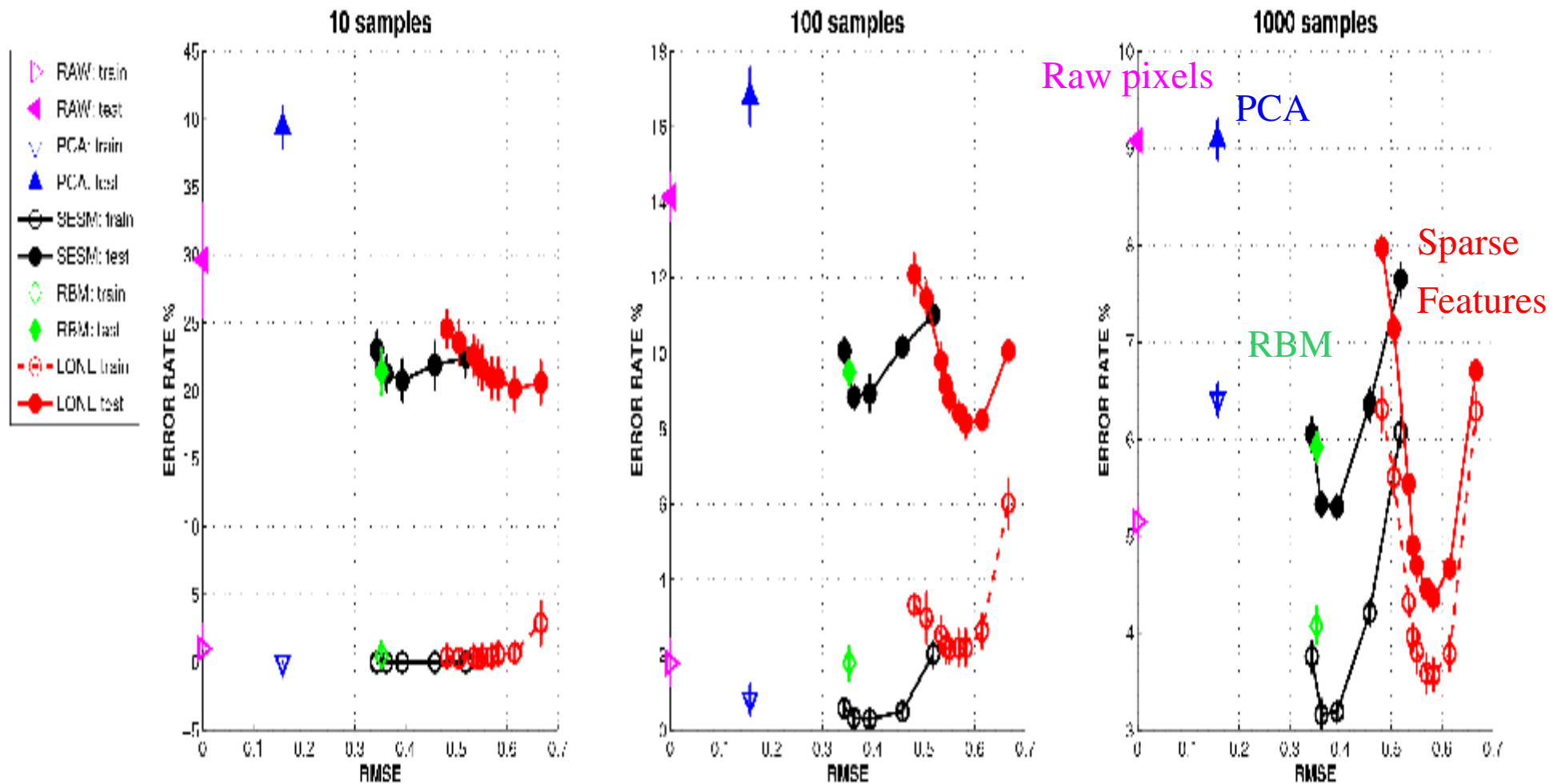
Supervised Linear Classifier trained on 200 trained sparse features

► Red: linear-tanh-diagonal encoder; Blue: linear encoder

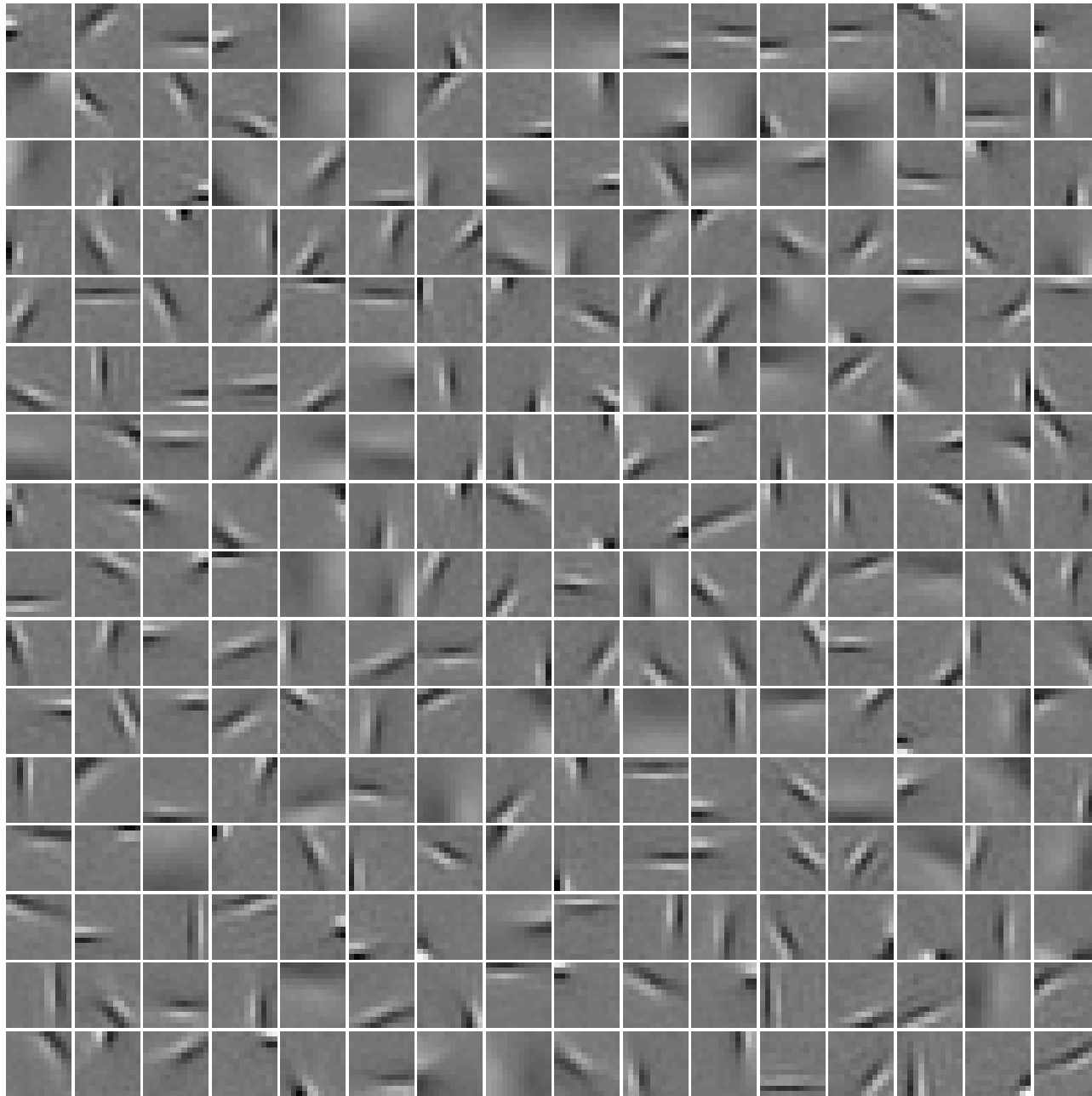


Classification Error Rate on MNIST

Supervised Linear Classifier trained on 200 trained sparse features

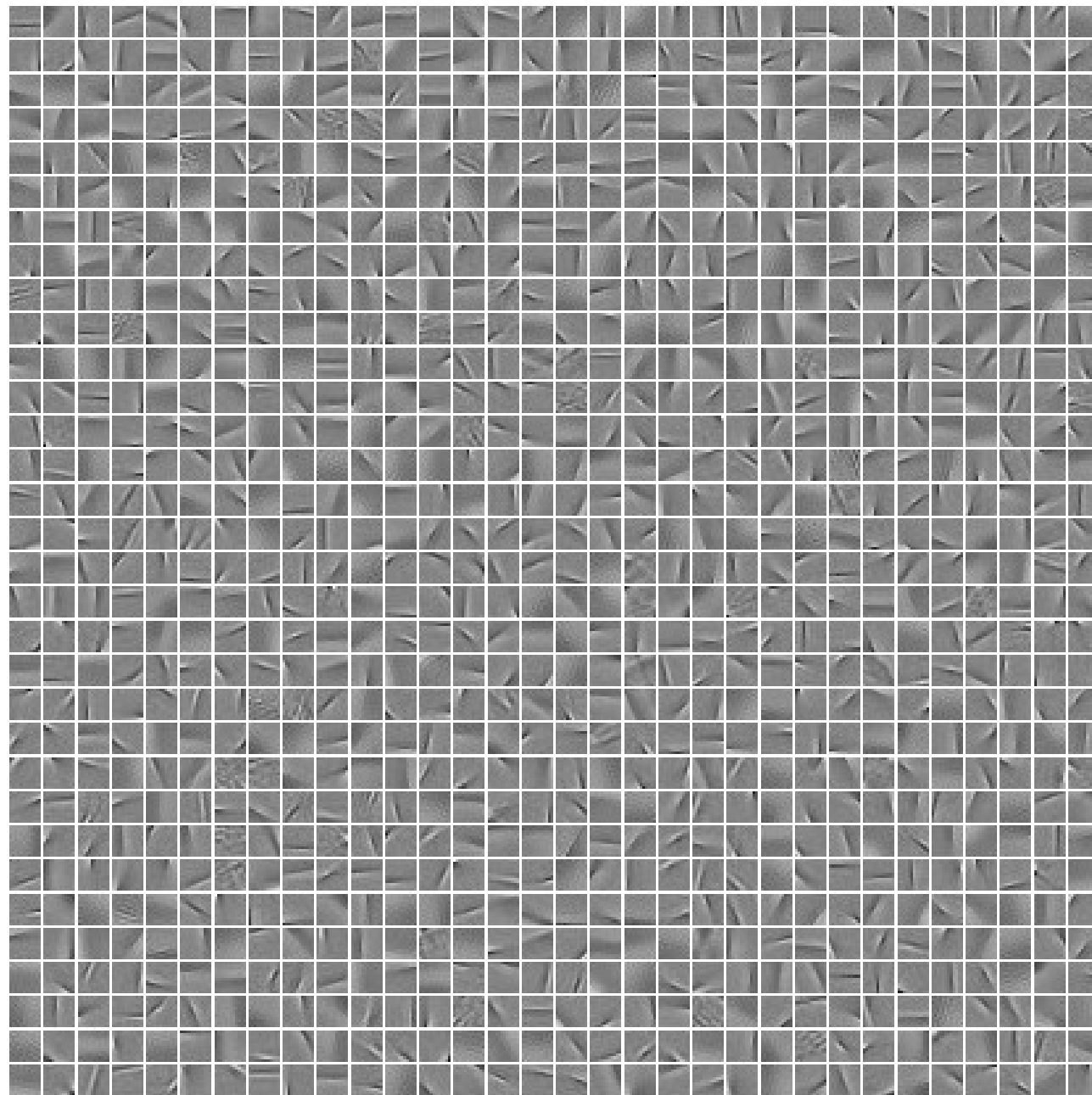


Learned Features on natural patches: V1-like receptive fields



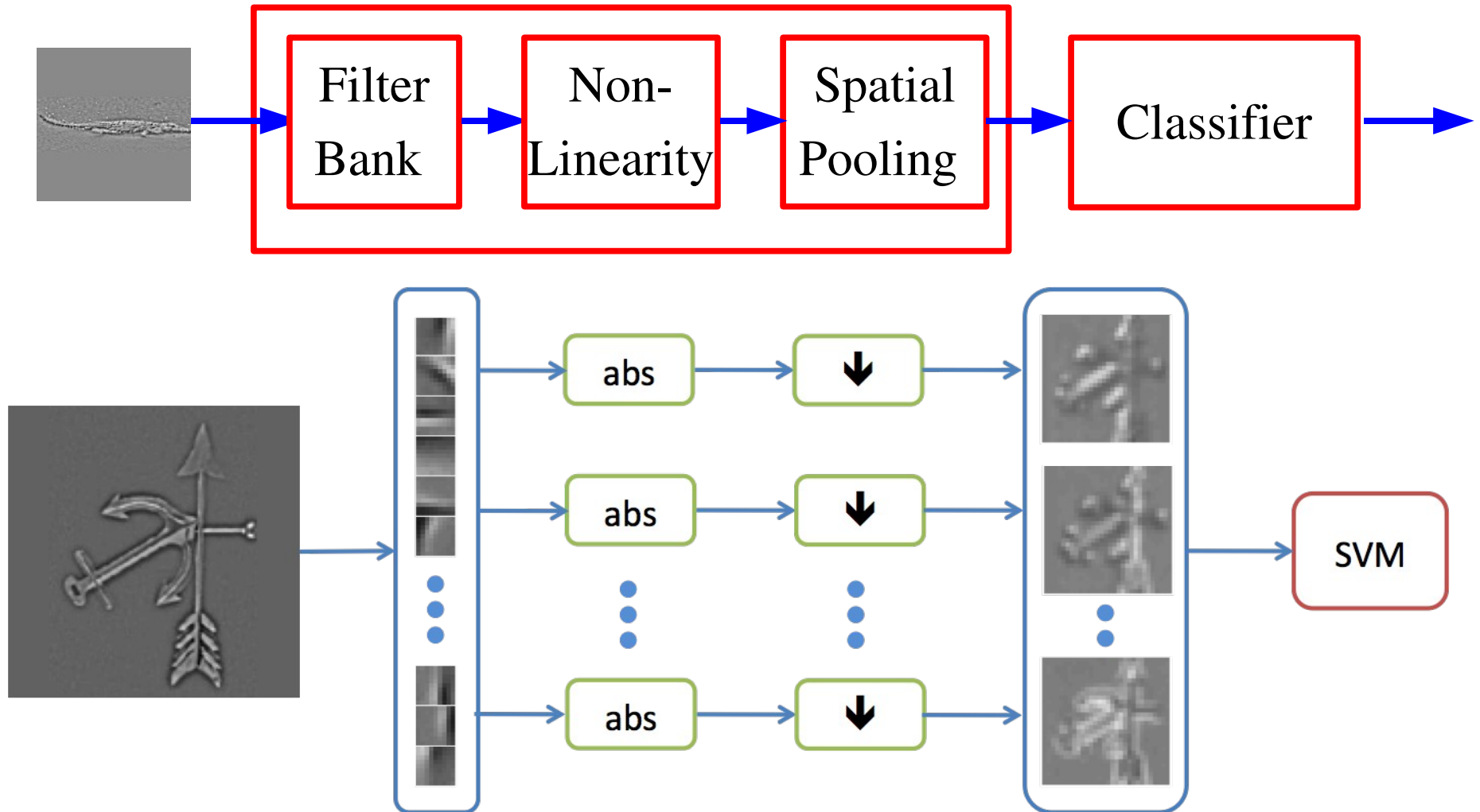
Learned Features: V1-like receptive fields

- 12x12 filters
- 1024 filters



How well do PSD features work on Caltech-101?

Recognition Architecture



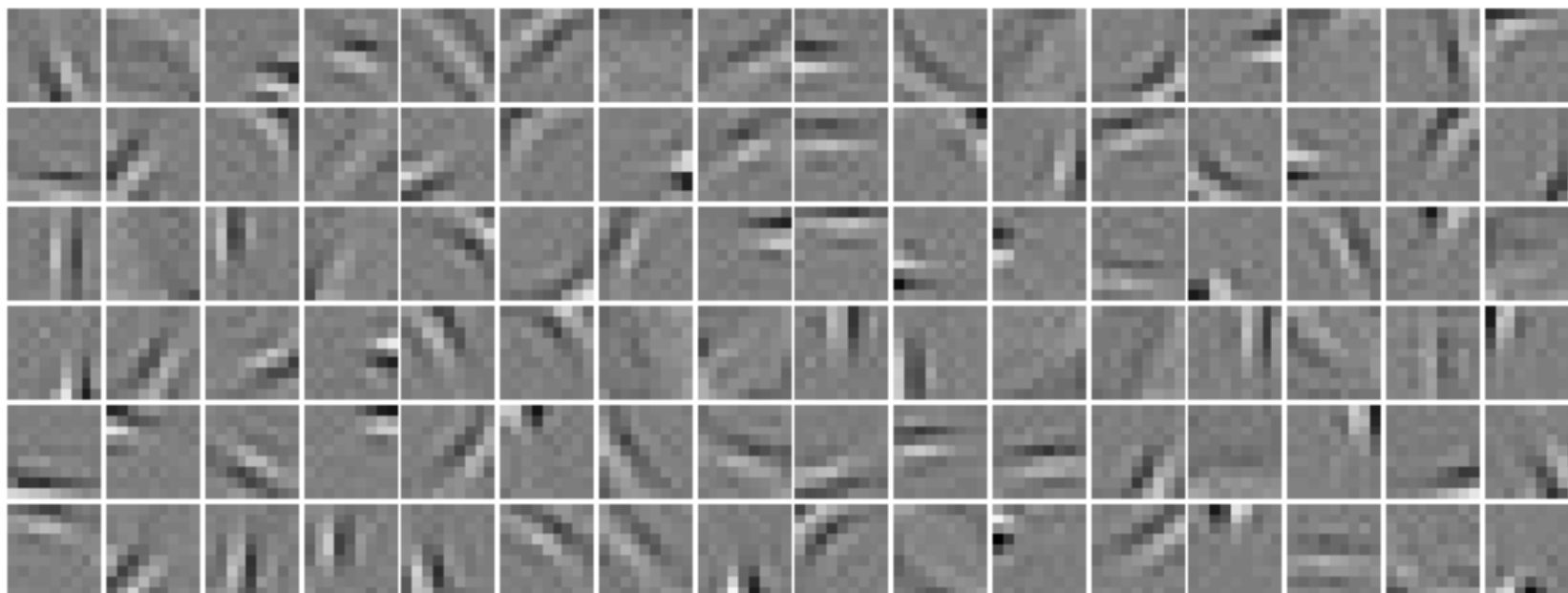
Using PSD Features for Recognition

96 filters on 9x9 patches trained with PSD

- ▶ with Linear-Sigmoid-Diagonal Encoder

Recognition:

- ▶ Normalized_Image -> Learned_Filters -> Rectification -> Local_Normalization -> Spatial_Pooling -> PCA -> Linear_Classifier
- ▶ What is the effect of rectification and normalization?



weights $\pm 0.9275 - 0.8688$

PSD: Caltech-101 Recognition Rate

Learning 96 Filters with PSD

- ▶ Filters->Sigmoid->AvPool->PCA->LinSVM 16%
- ▶ Filters->Sigmoid+Abs->AvPool->PCA->LinSVM 51%
- ▶ LCN->Filters->Sigmoid+Abs->AvPool->PCA->LinSVM 56%
- ▶ LCN->Filters->Sigm+Abs->LCN->AvPool->PCA->LinSVM 58%

LCN = Local Contrast Normalization (division by std dev of neighbors)

AvPool = Average pooling using boxcar filter and subsampling

PCA = PCA to 3000 components; LinSVM: Linear SVM classifier.

[Pinto&DiCarlo 2008] V1 model with 96 Gabor Filters (16 orientations, 4 scales) and half rectification

- ▶ LCN->G.Filters->Half-Rectif.->LCN->AvPool->PCA->LinSVM
- ▶ Caltech-101 recognition rate 59%

Adding a rectification makes a huge difference: 16%->51%

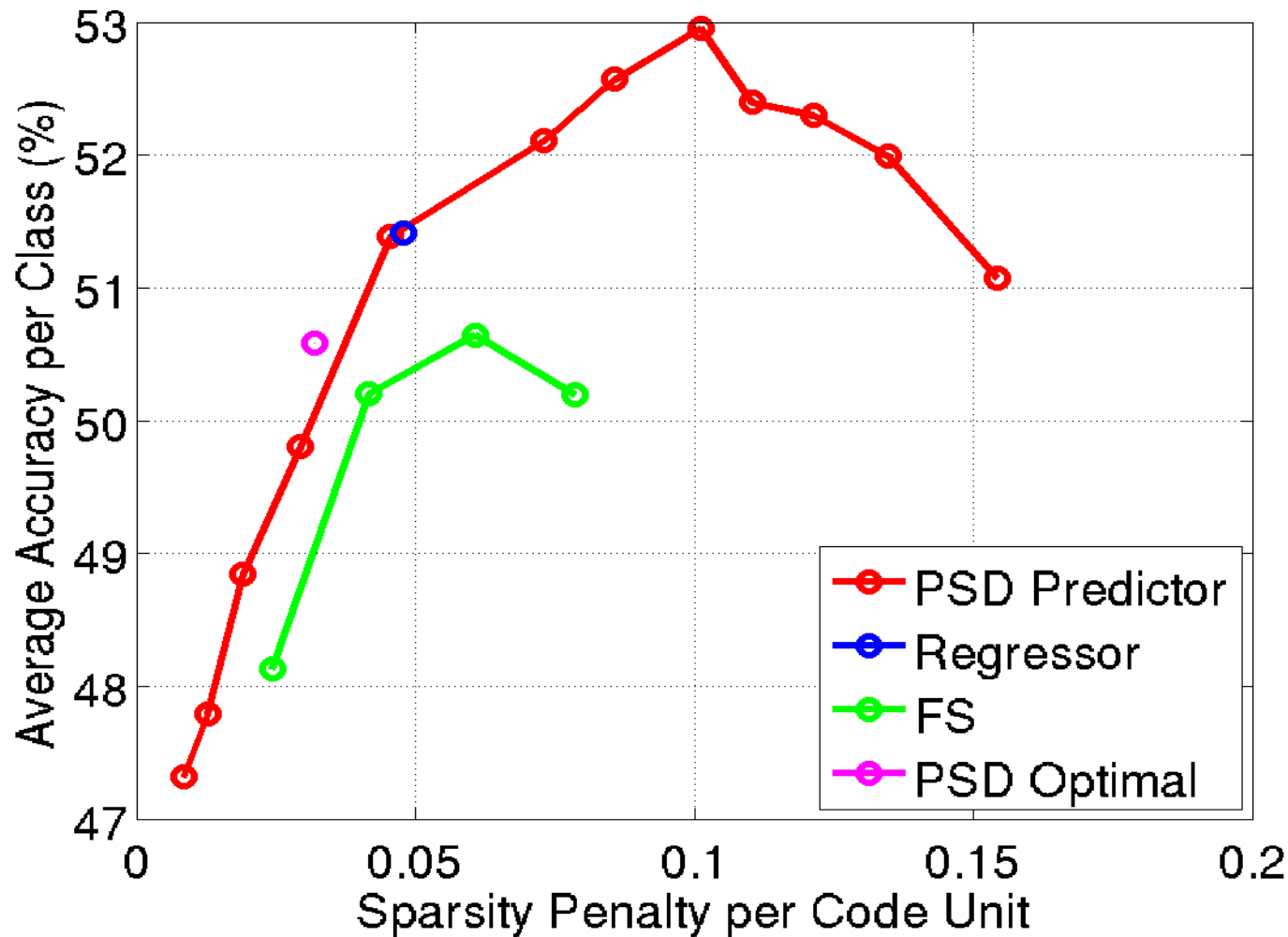
- ▶ The right features are not the output of the encoder
- ▶ The right features are the output of the sparsification function

Learning the filters with PSD gives the same results as multiscale Gabors

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!**

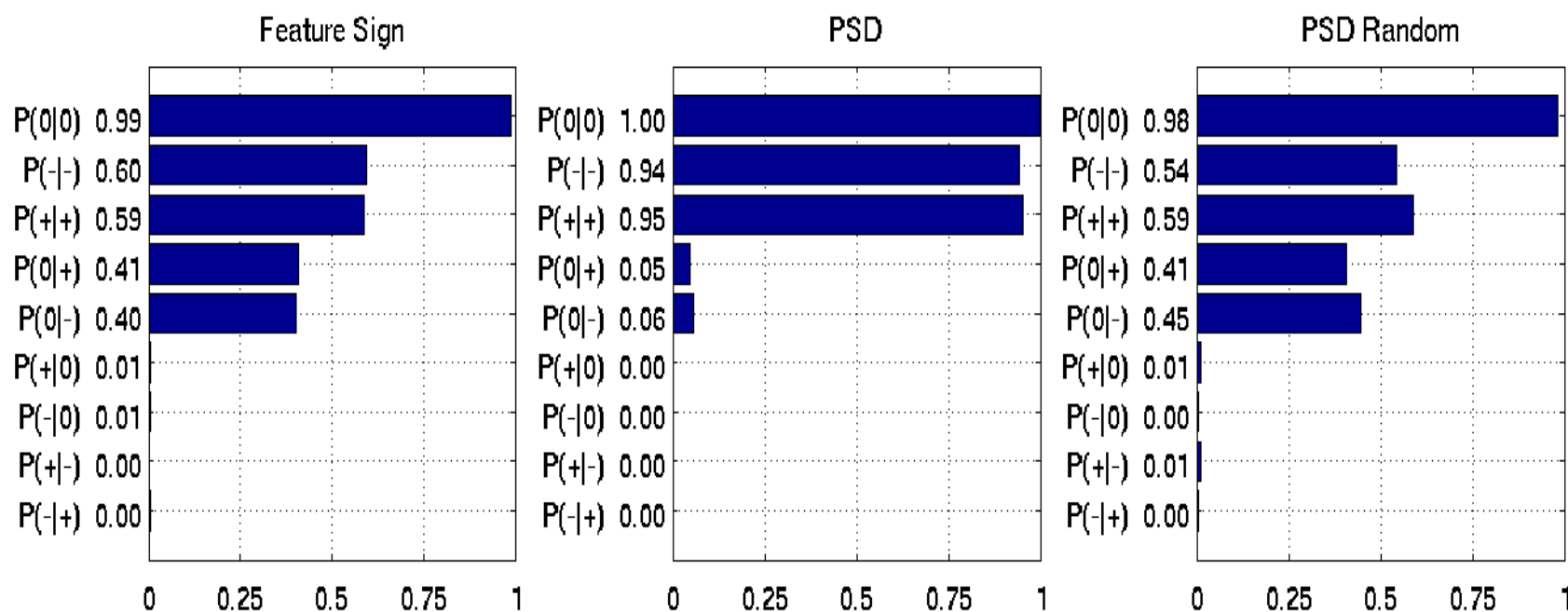
▶ PSD features are more stable.



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

PSD Features are more stable

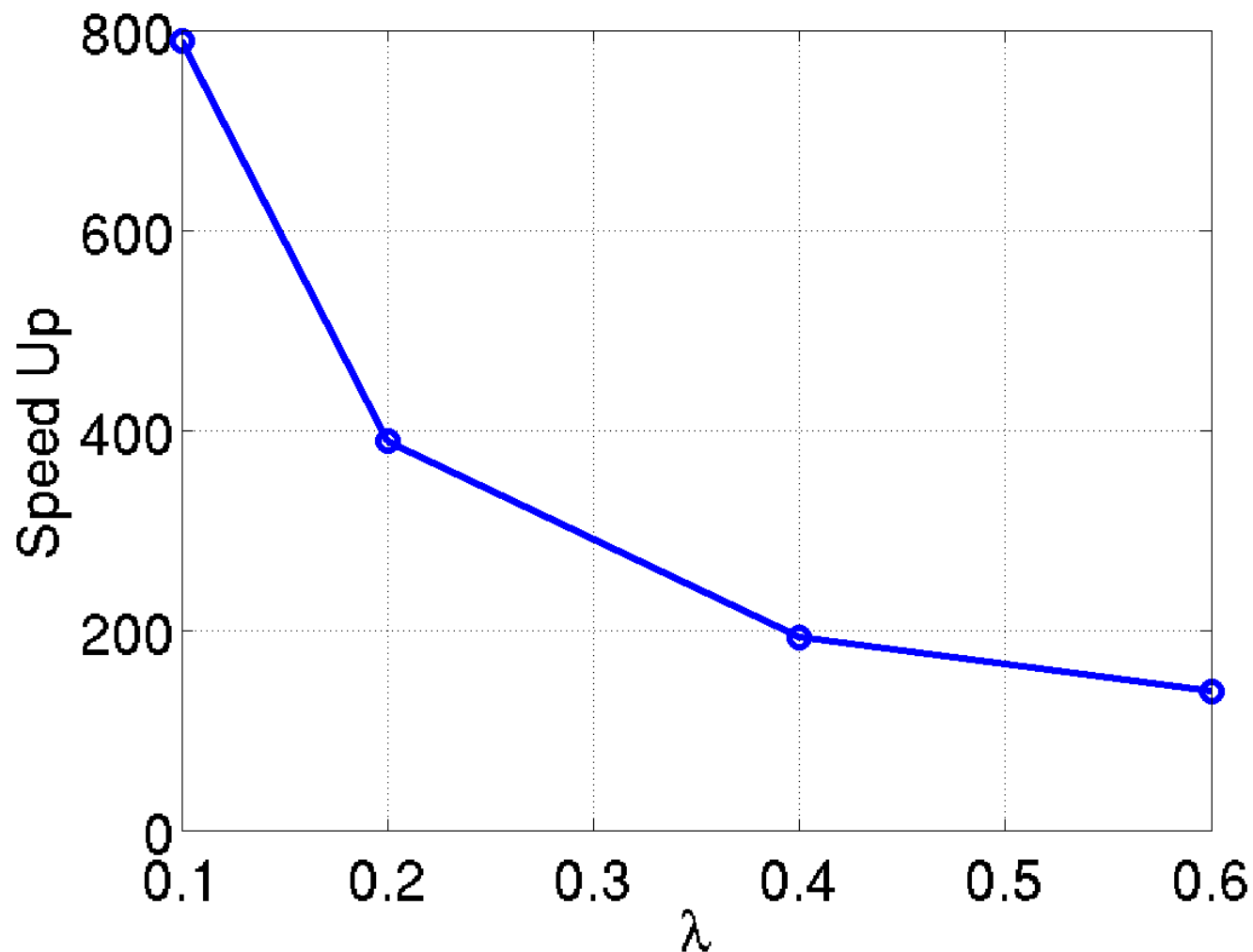
- Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!
- Because PSD features are more stable. Feature obtained through sparse optimization can change a lot with small changes of the input.



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

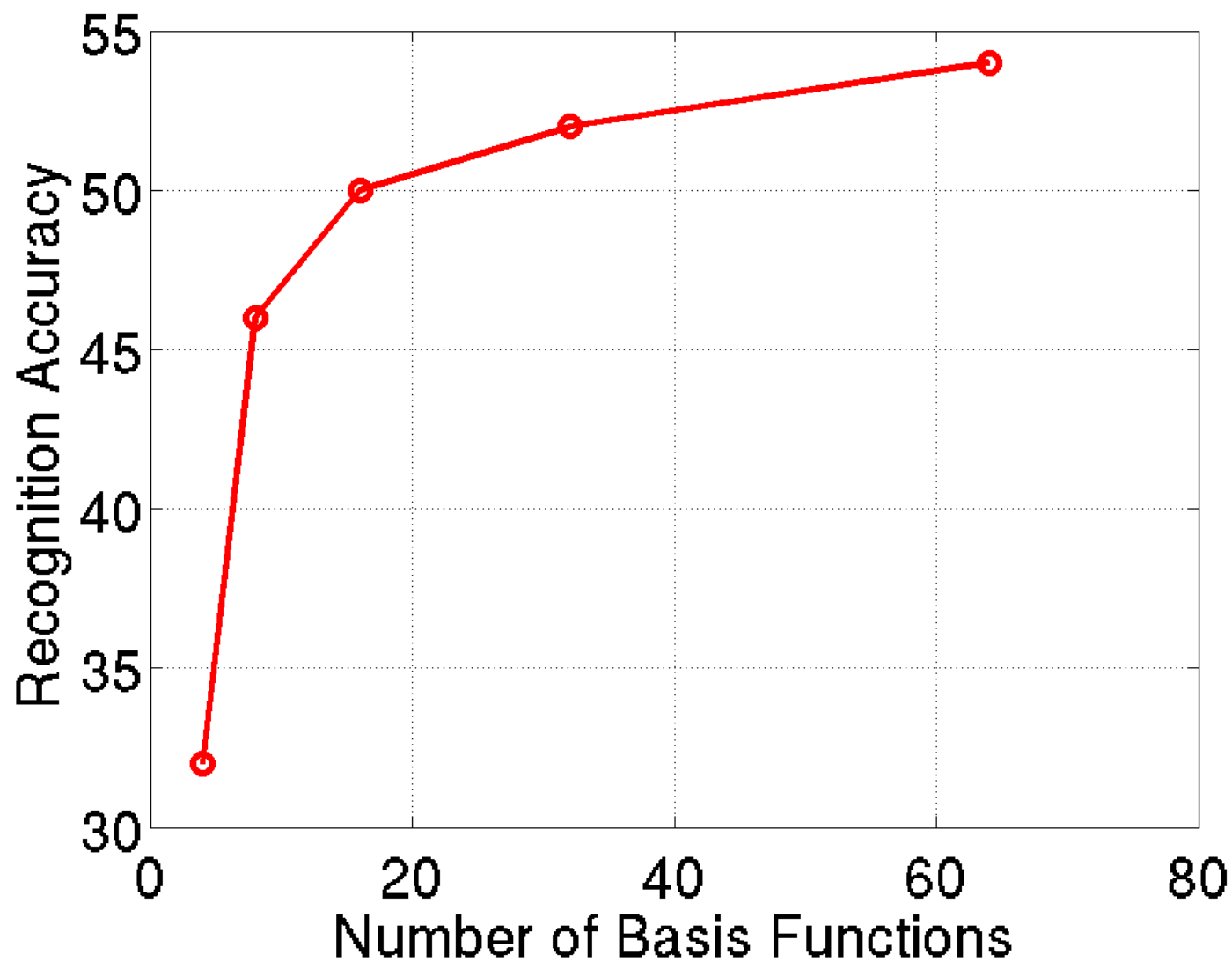
PSD features are much cheaper to compute

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

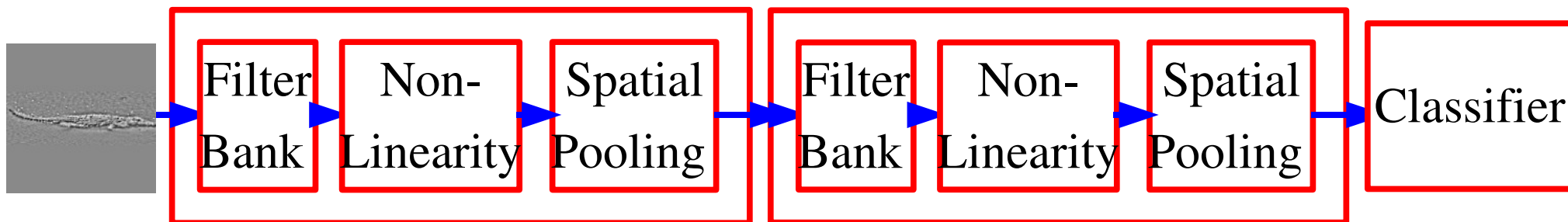


How Many 9x9 PSD features do we need?

- Accuracy increases slowly past 64 filters.

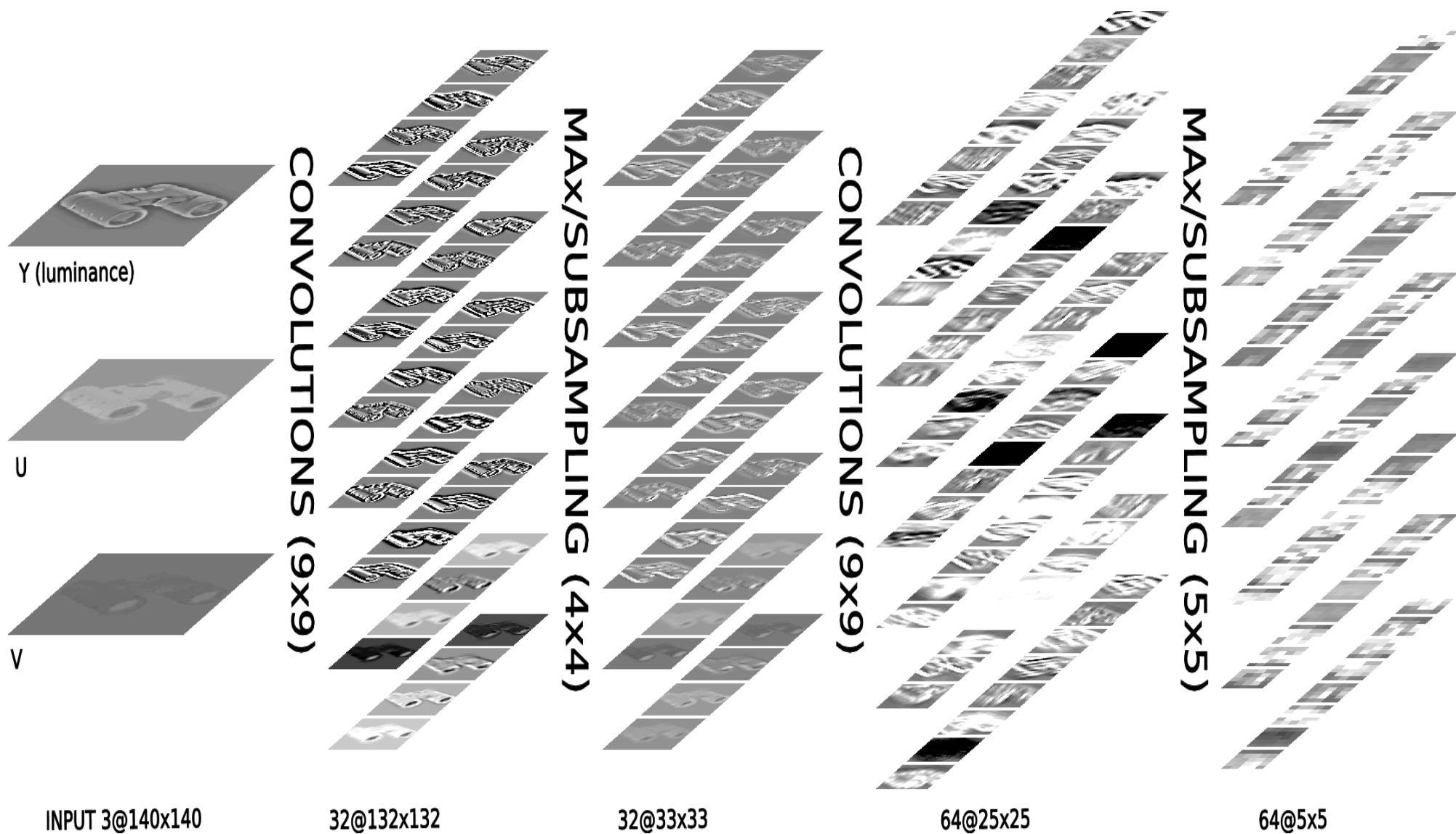


Training a Multi-Stage Hubel-Wiesel Architecture with PSD



1. Train stage-1 filters with PSD on patches from natural images
2. Compute stage-1 features on training set
3. Train stage-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 on Caltech-101

- Each architecture has two stages of feature extraction and pooling, plus a supervised linear classifier on top.
- Convolutional Net: purely supervised training**
 - ▶ Stage: Filters->Tanh->AveragePooling->Tanh
 - ▶ Architecture: Stage->Stage->LinClassifier 26%
- Convolutional Net: unsupervised training of stages with SESM, and supervised training of top layer [Ranzato et al. CVPR 07]**
 - ▶ Stage: Filters->Tanh->MaxPooling
 - ▶ Architecture: Stage->Stage->LinClassifier 54%
- HMAX [Serre 05] -> [Mutch&Lowe 06]**
 - ▶ Fixed Gabors at stage-1, simple learning algo for stage-2 (storing random templates)
 - ▶ Stage: MultiscaleFilters->Sigmoid->Scale/Space Pooling
 - ▶ Architecture: Stage->Stage->LinClassifier 56%

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

Various non-linearities and training protocols

- ▶ R: random initialization + supervised training
- ▶ P: PSD training (frozen)
- ▶ A: PSD training + supervised adjustment
- ▶ Tanh: sigmoid non-linearity
- ▶ Abs: sigmoid+absolute value non-linearity
- ▶ Cnorm: local contrast normalization

Id	Accuracy (%)	Protocol	Machine
Traditional ConvNet Architecture			
1	26.0%	RR	Tanh, 64 features
2	30.0%	AA	Tanh, CNorm, 64 features
With Absolute Value Non-Linearity			
3	58.0%	RR	Abs, 64 features
Abs and Contrast Normalization			
4	60.0%	RR	Abs, CNorm, 64 features
5	62.0%	AR	Abs, CNorm, 64 features
6	62.9%	PP	Abs, CNorm, 64 features
7	63.0%	PA	Abs, CNorm, 64 features
8	67.2%	AA	Abs, CNorm, 64 features
Smaller net with Abs and CNorm			
9	59.8%	PP	Abs, CNorm, 16 features
10	65.2%	AA	Abs, CNorm, 16 features

Multistage Hubel-Wiesel Architecture on Caltech-101

• Our Best result is **67.2%**

- ▶ comparable to the **66%** of [Lazebnik 2006], obtained with SIFT features, VQ, and an SVM with spatial pyramid matching kernel.
- ▶ Comparable to the **67.2%** of [Ahmad, Yu et al 2008], which was a ConvNet trained with a “pseudo-task” with extra output produced by an HMAX-like model.
- ▶ The result is below [Varma 2007] which uses a large number of hand-designed features and a learned linear combination of kernels.

• Our system is considerably cheaper/faster, and simpler

- ▶ Smaller net with only 16 features at stage-1 yield **65.2%**.

• The crucial ingredient seems to be the absolute value rectification

- ▶ A purely supervised system with rectification and contrast normalization yields **60%**, despite the enormous number of parameters compared to the number of training samples!

• Global supervised refinement is essential

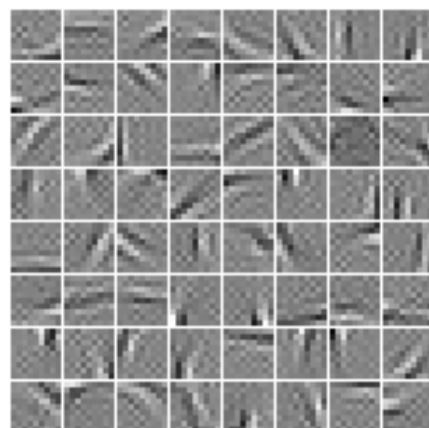
- ▶ The 2nd stage features need supervised refinement.

Multistage Hubel-Wiesel Architecture: Filters

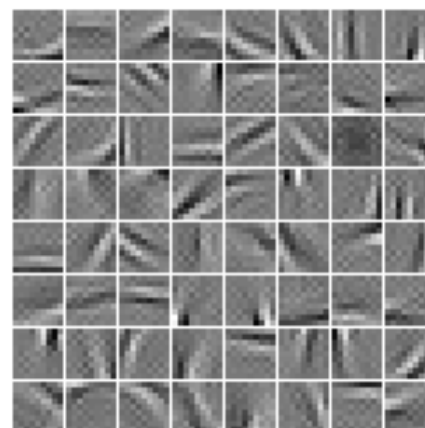
● After PSD

● After supervised refinement

● Stage 1

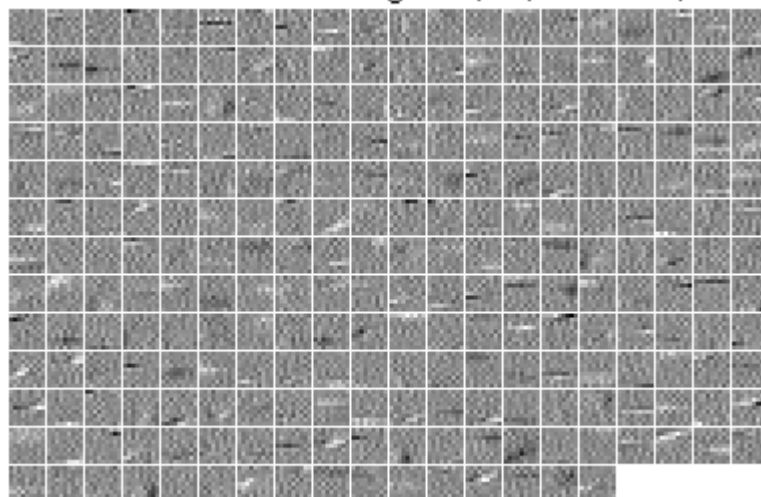


weights $\pm 0.2232 - 0.2075$

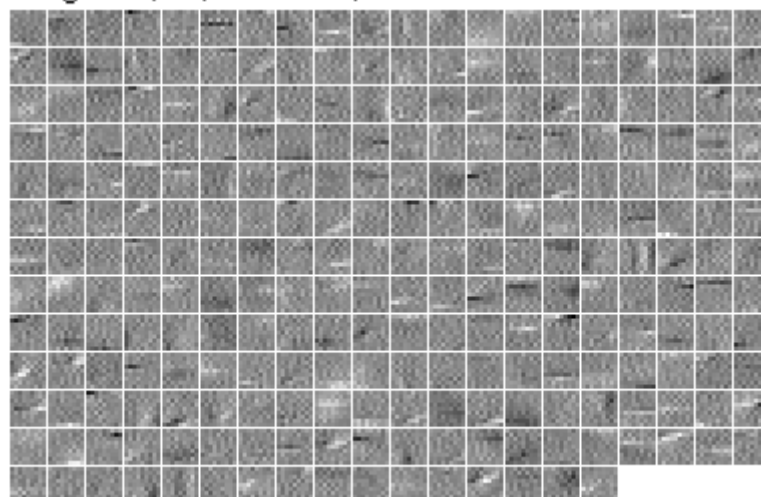


weights $\pm 0.2828 - 0.3043$

● Stage 2



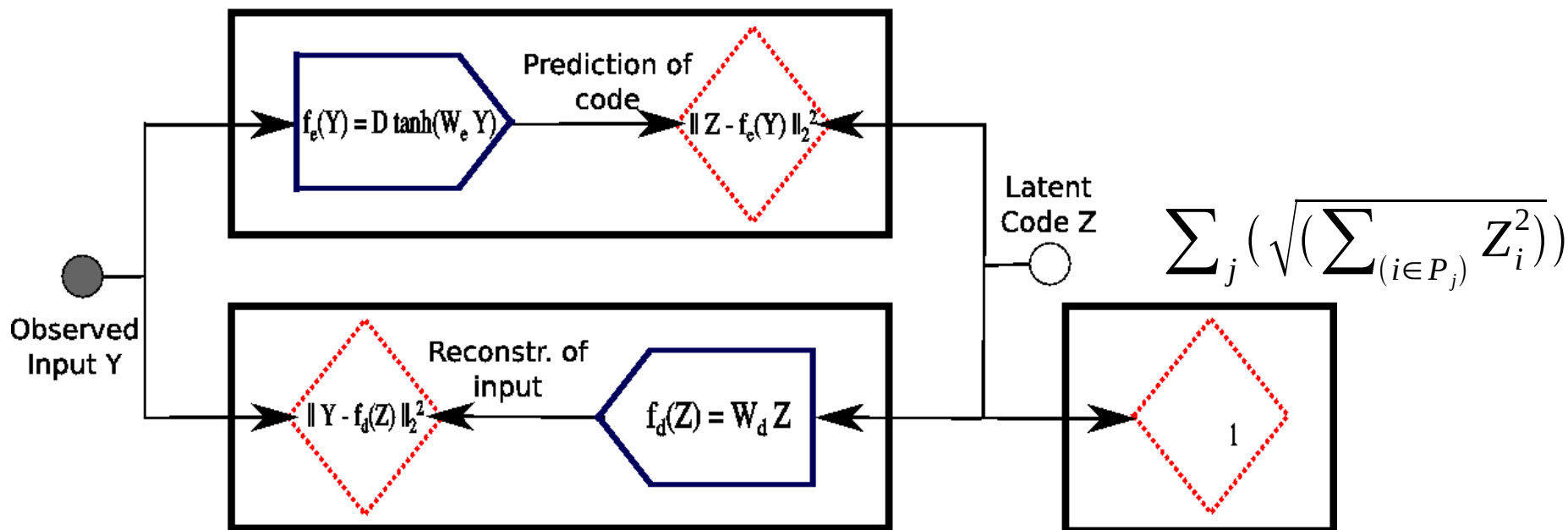
weights $\pm 0.0778 - 0.064$



weights $\pm 0.0929 - 0.0784$

Learning Invariant Features [Kavukcuoglu et al. CVPR 2009]

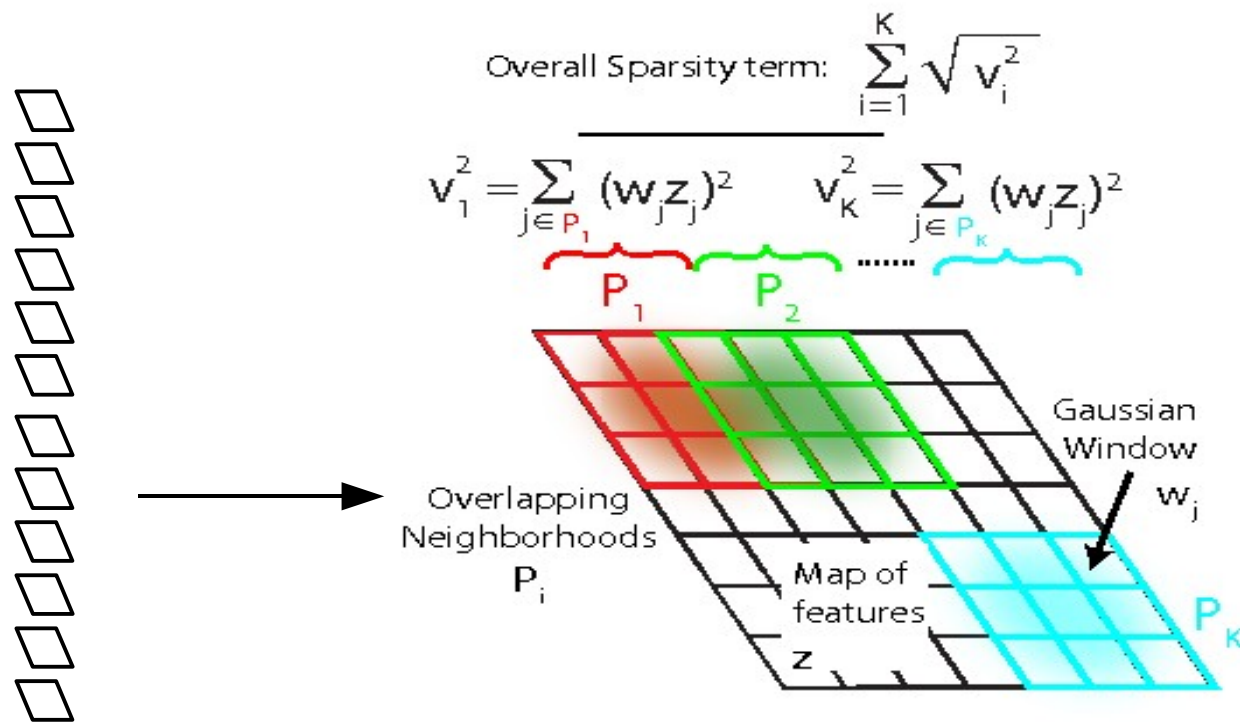
- Unsupervised PSD ignores the spatial pooling step.
- Could we devise a similar method that learns the pooling layer as well?
- Idea [Hyvarinen & Hoyer 2001]: sparsity on pools of features
 - ▶ Minimum number of pools must be non-zero
 - ▶ Number of features that are on within a pool doesn't matter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

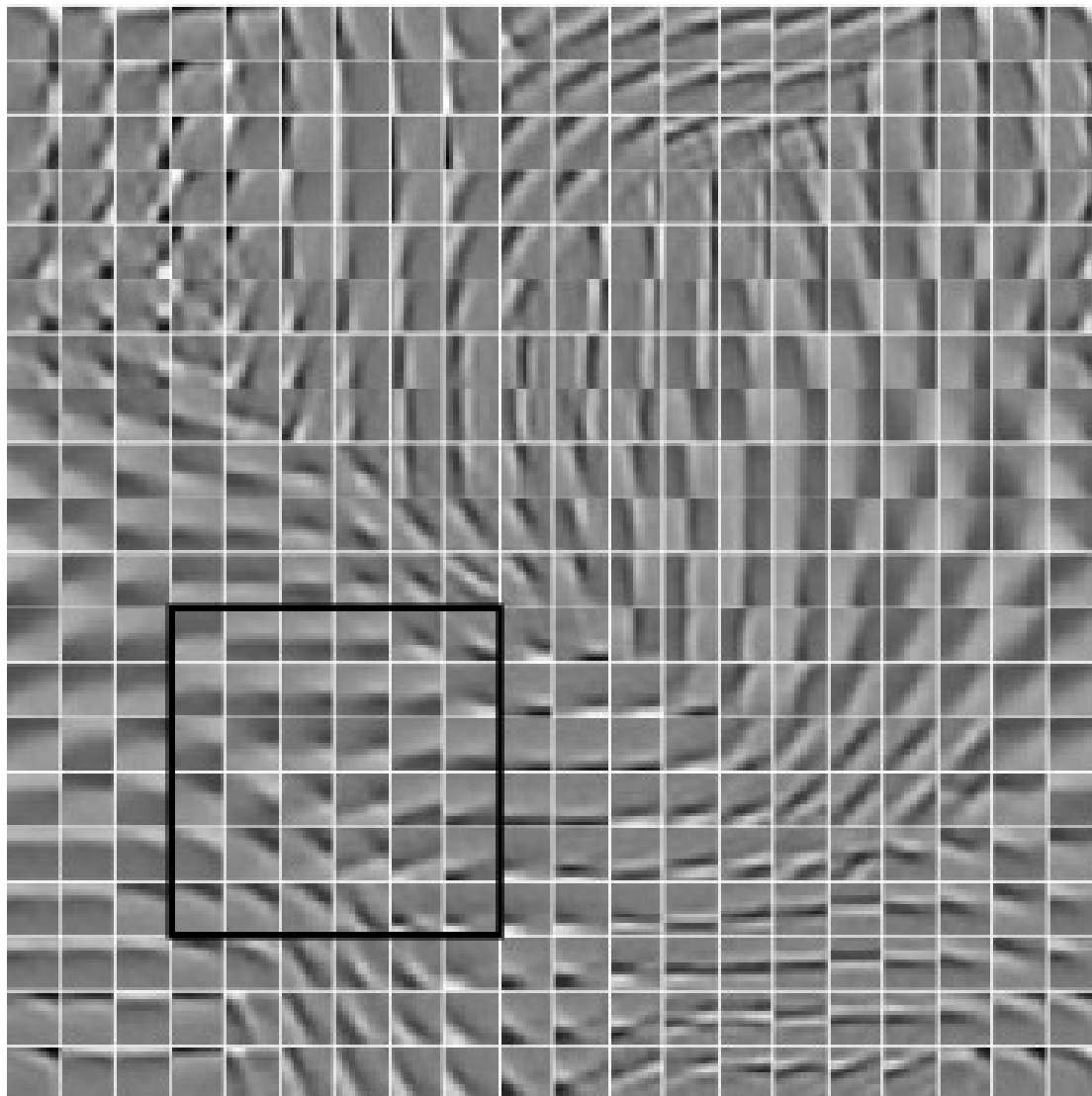


Units in the code Z

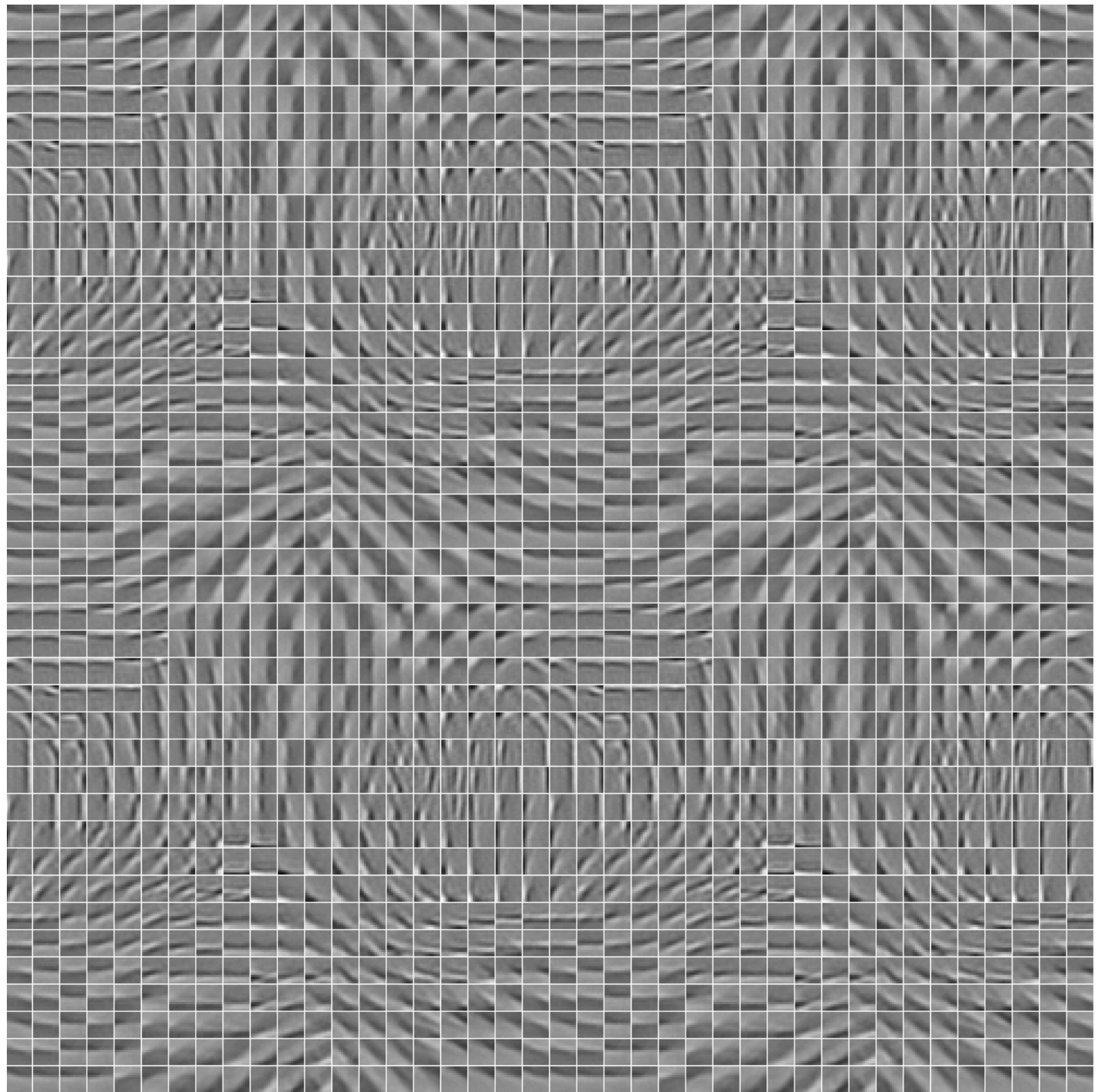
Define pools and enforce sparsity across pools

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?

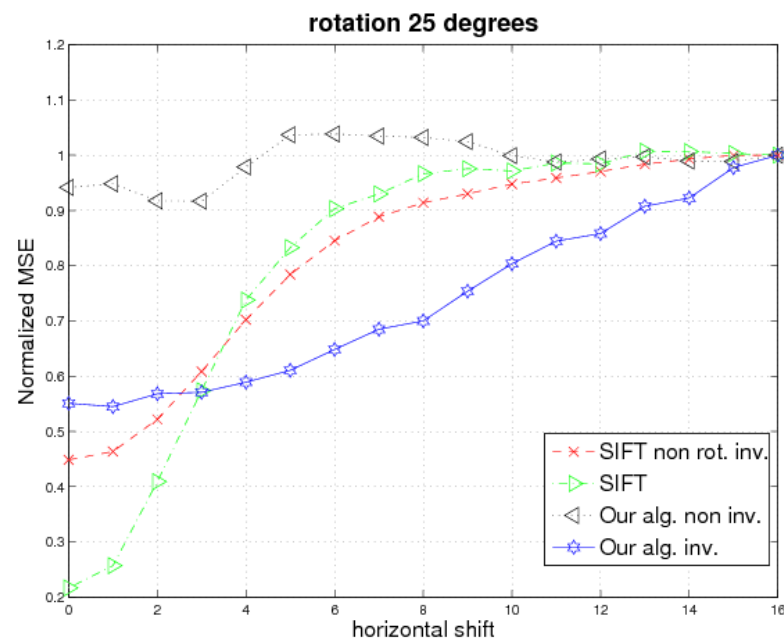
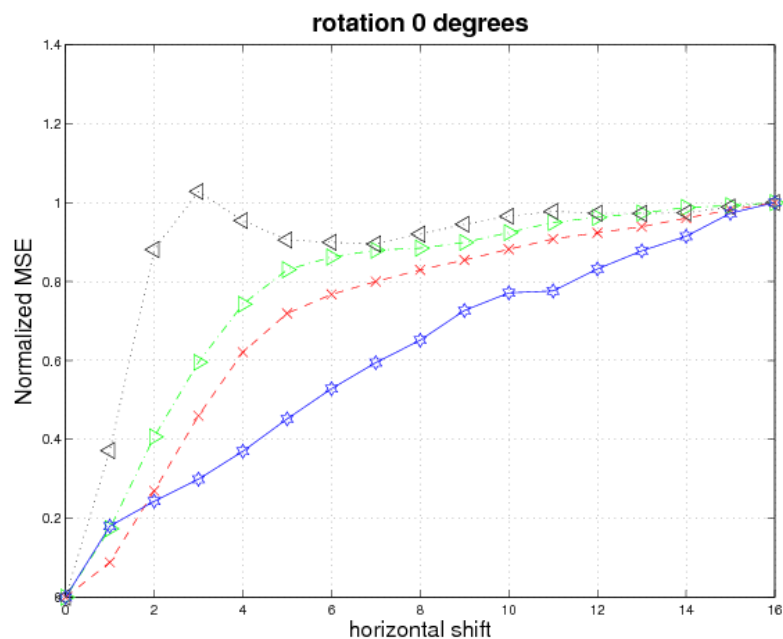


Invariance Properties Compared to SIFT

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.

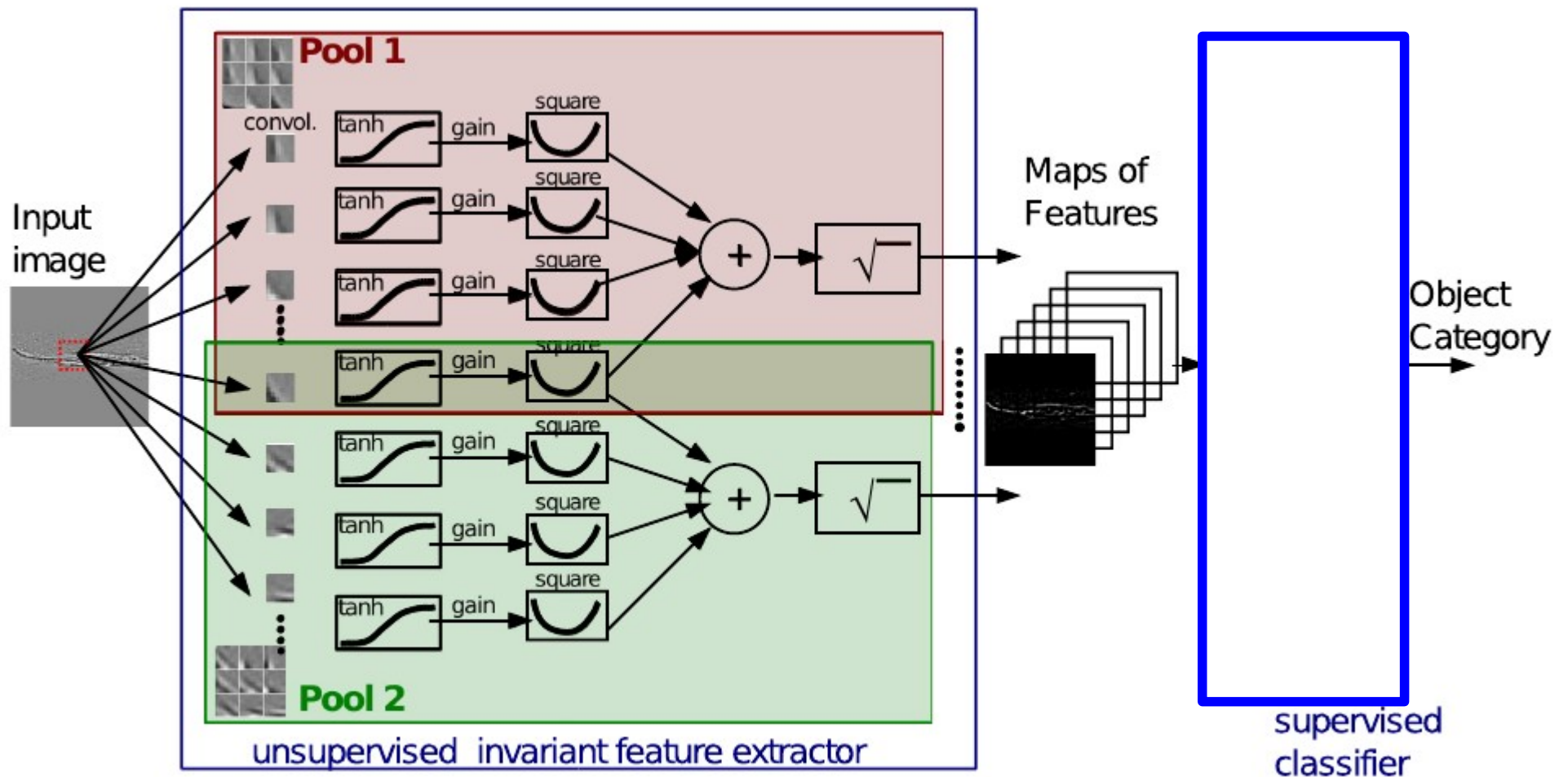
Topographic PSD features are more invariant than SIFT



Learning Invariant Features

Recognition Architecture

- ▶ ->HPF/LCN->filters->tanh->sq- >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	Avrge Accuracy/Class (%)
classifier: PCA + linear SVM	
Proposed Method	50.9
SIFT (not rotation invariant)	51.2
SIFT (rotation invariant)	45.2
Serre et al. features [26]	47.1
classifier: Spatial Pyramid Matching Kernel SVM	
Proposed Method	59.6
SIFT	65

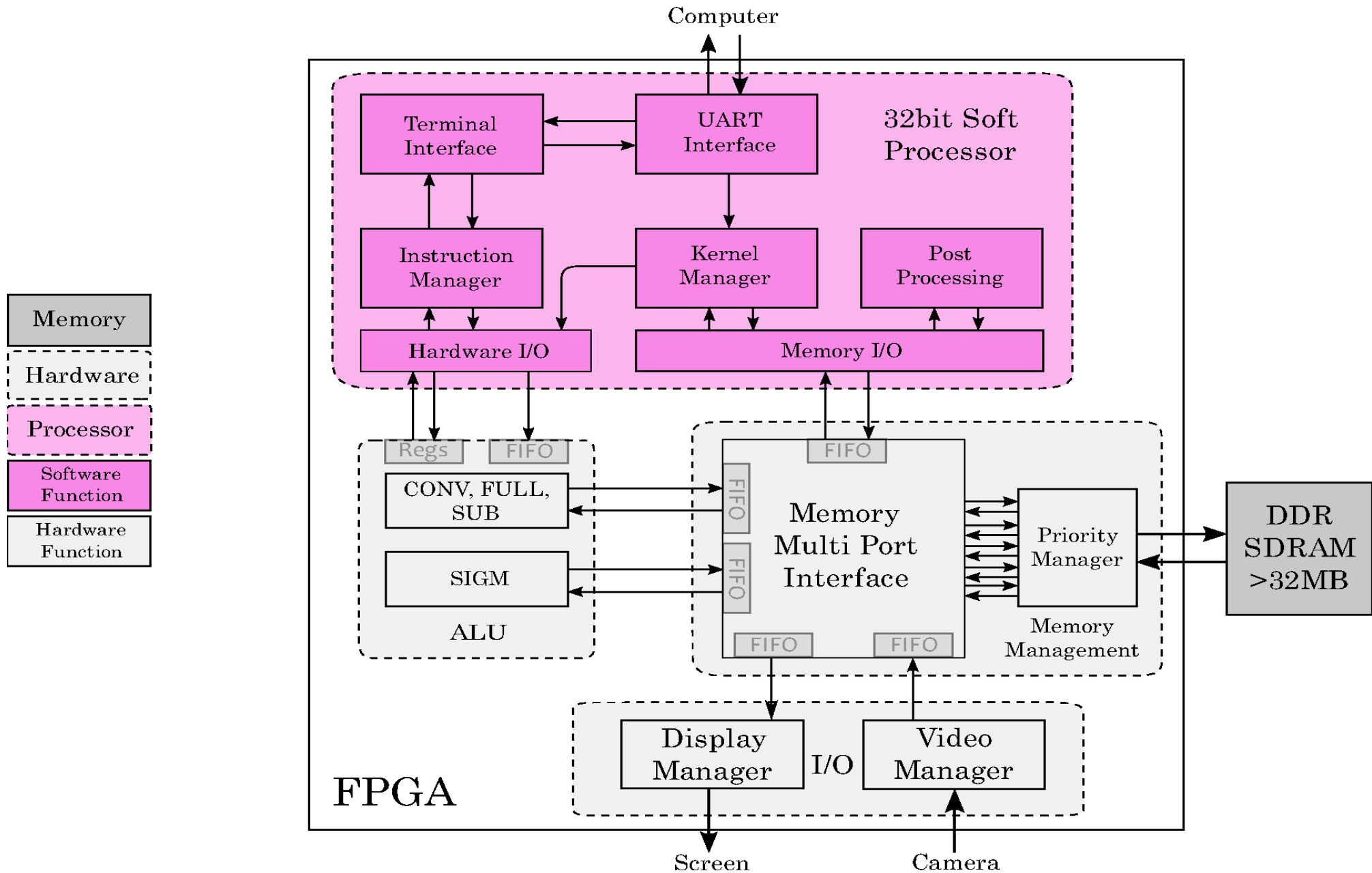
CNP: FPGA Implementation of ConvNets

Implementation on low-end Xilinx FPGA

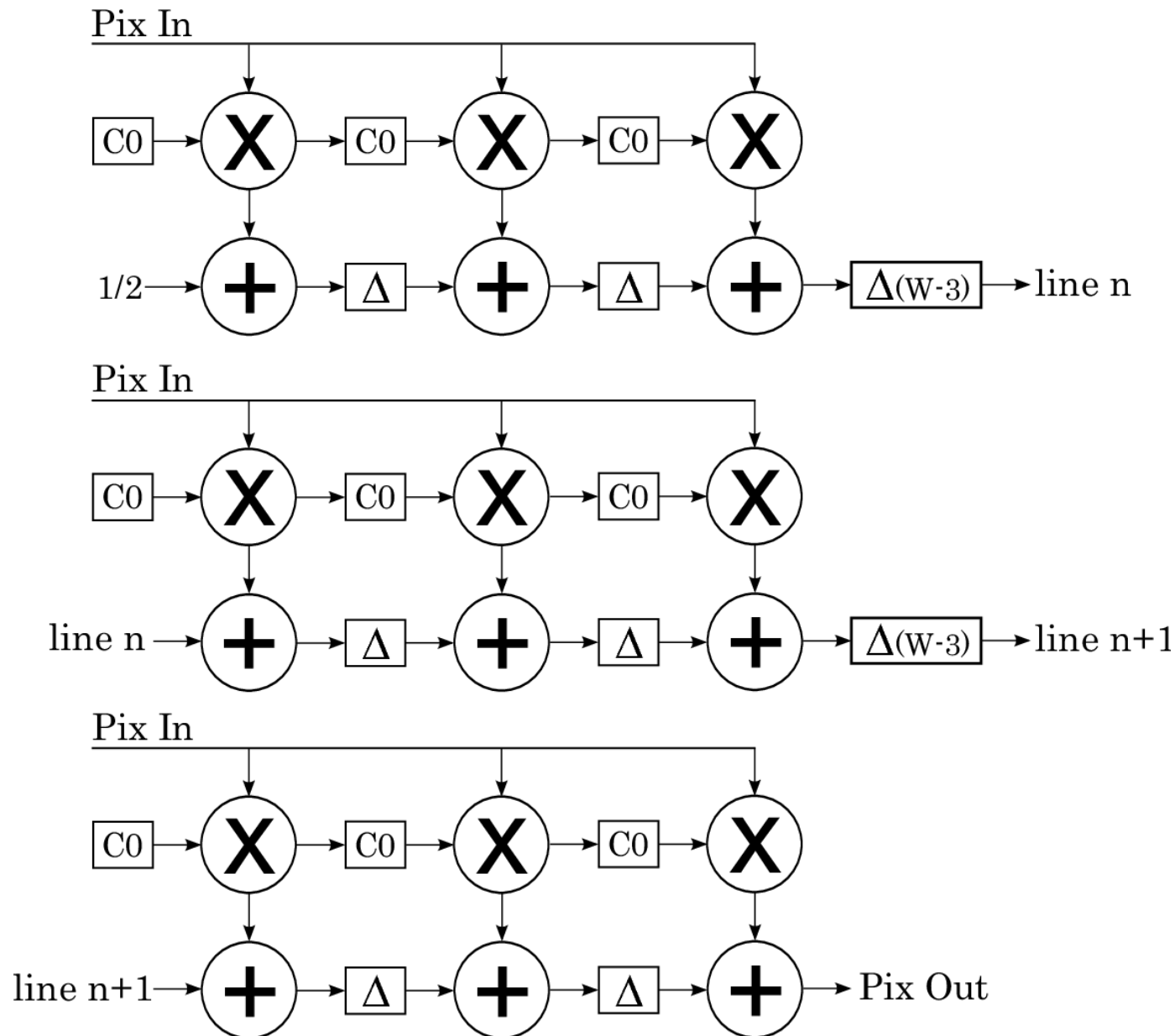
- ▶ Xilinx Spartan3A-DSP: 250MHz, 126 multipliers.
- ▶ **Face detector ConvNet** at 640x480: 5e8 connections
- ▶ 8fps with 200MHz clock: 4Gcps effective
 - Prototype runs at lower speed b/c of narrow memory bus on dev board
- ▶ Very lightweight, very low power
 - Custom board the size of a matchbox (4 chips: FPGA + 3 RAM chips)
 - good for micro UAVs vision-based navigation.
- ▶ High-End FPGA could deliver very high speed: 1024 multipliers at 500MHz: 500Gcps peak perf.



CNP Architecture



Systolic Convolver: 7x7 kernel in 1 clock cycle



Design

• Soft CPU used as micro-sequencer

- ▶ Micro-program is a C program on soft CPU

• 16x16 fixed-point multipliers

- ▶ Weights on 16 bits, neuron states on 8 bits.

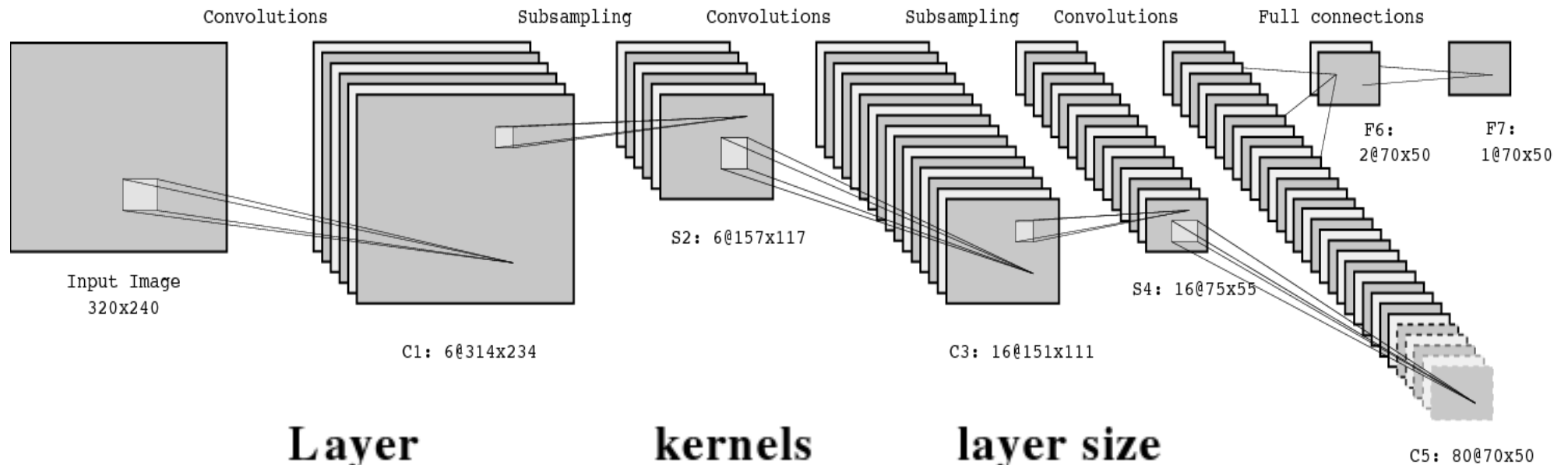
• Instruction set includes:

- ▶ Convolve X with kernel K result in Y, with sub-sampling ratio S
- ▶ Sigmoid X to Y
- ▶ Multiply/Divide X by Y (for contrast normalization)

• Microcode generated automatically from network description in Lush

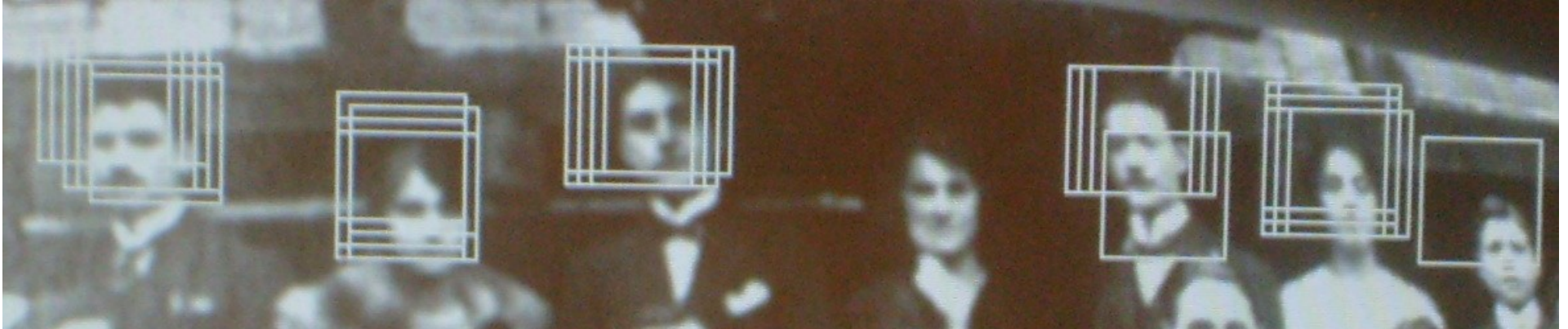
Entity	Occupancy	
I/Os	135 out of 469	28%
DCMs	2 out of 8	25%
Mult/Accs	56 out of 126	44%
Bloc Rams	100 out of 126	84%
Slices	16790 out of 23872	70%

Face detector on CNP



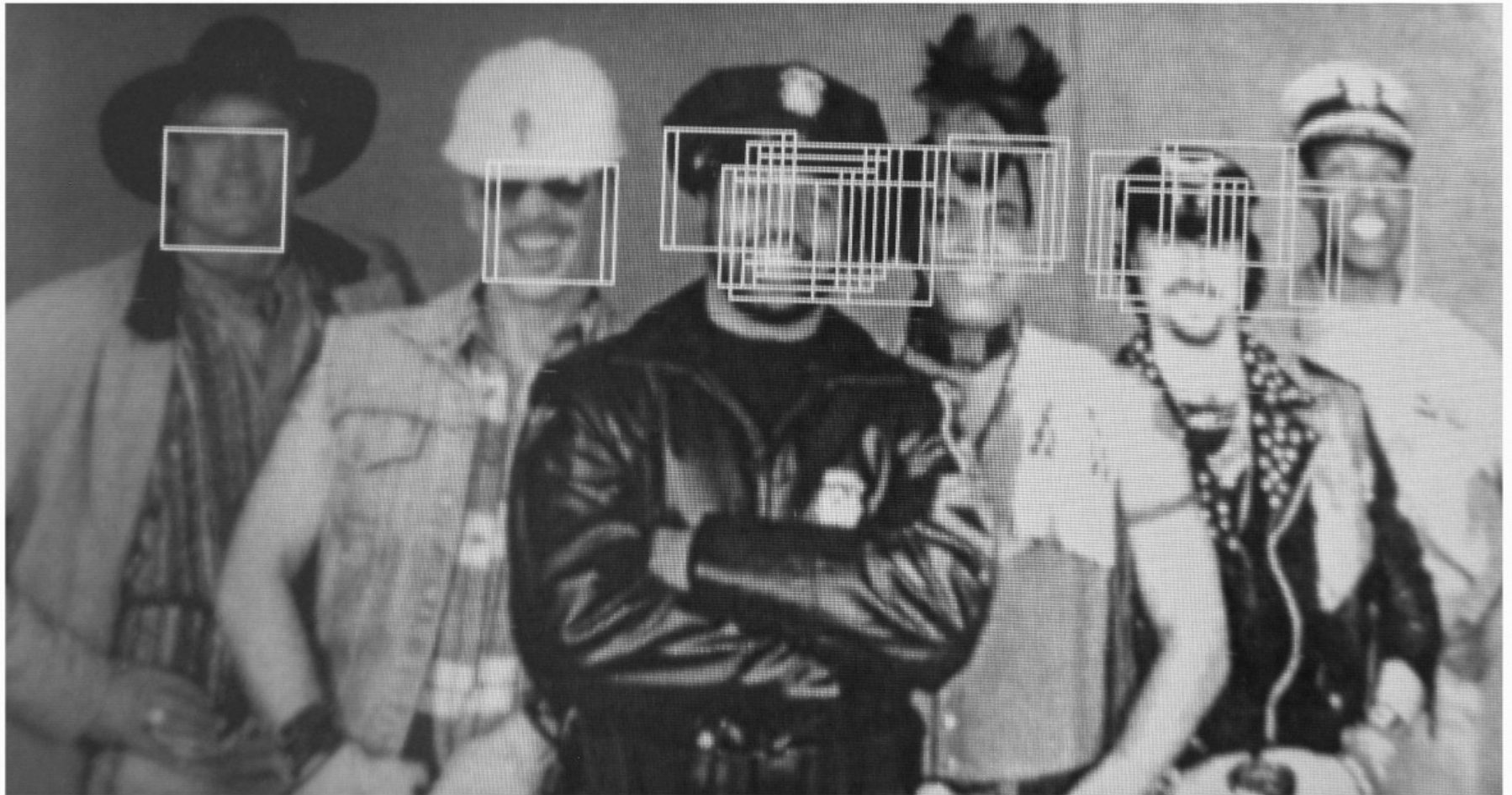
Layer	kernels	layer size
Input image		1@42 × 42
C1 (Conv)	[6@7 × 7]	6@36 × 36
S2 (Pool)	[6@2 × 2]	6@18 × 18
C3 (Conv)	[61@7 × 7]	16@12 × 12
S4 (Pool)	[16@2 × 2]	16@6 × 6
C5 (Conv)	[305@6 × 6]	80@1 × 1
F6 (Dotp)	[160@1 × 1]	2@1 × 1

Results



- **Clock speed limited by low memory bandwidth on the development board**
 - ▶ Dev board uses a single DDR with 32 bit bus
 - ▶ Custom board will use 128 bit memory bus
- **Currently uses a single 7x7 convolver**
 - ▶ We have space for 2, but the memory bandwidth limits us
- **Current Implementation: 5fps at 512x384**
- **Custom board will yield 30fps at 640x480**
 - ▶ 4e10 connections per second peak.

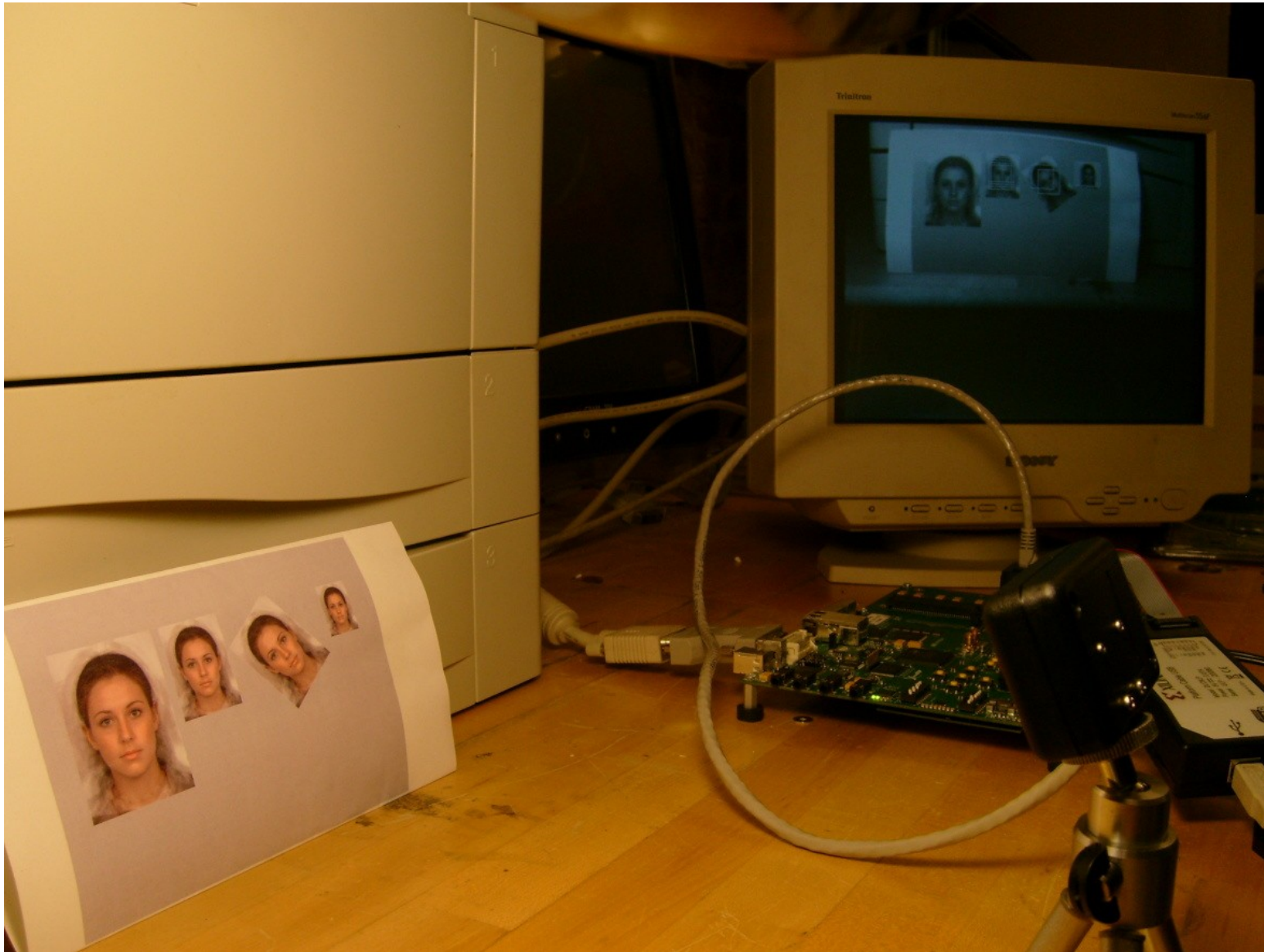
Results



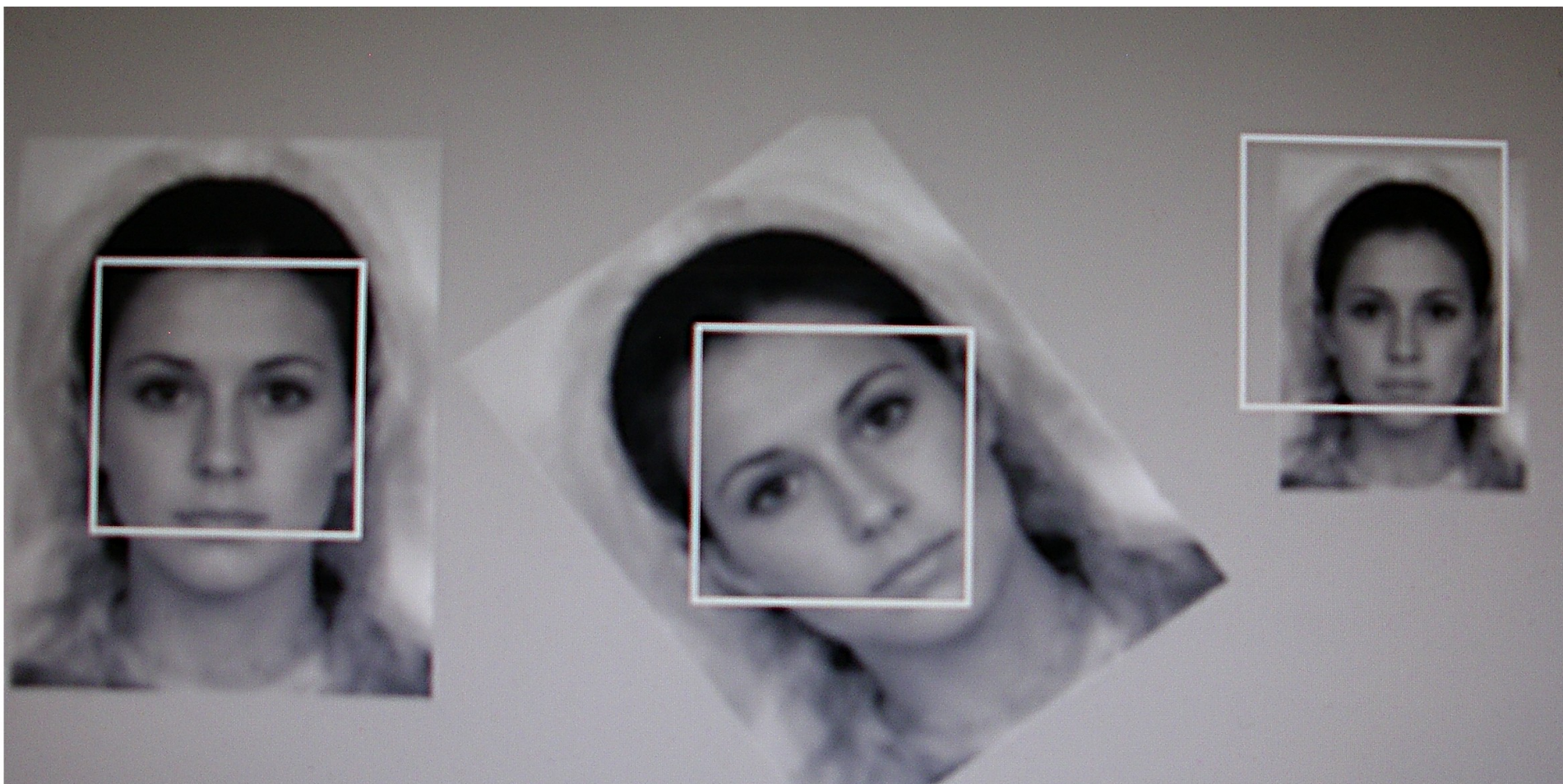
Results



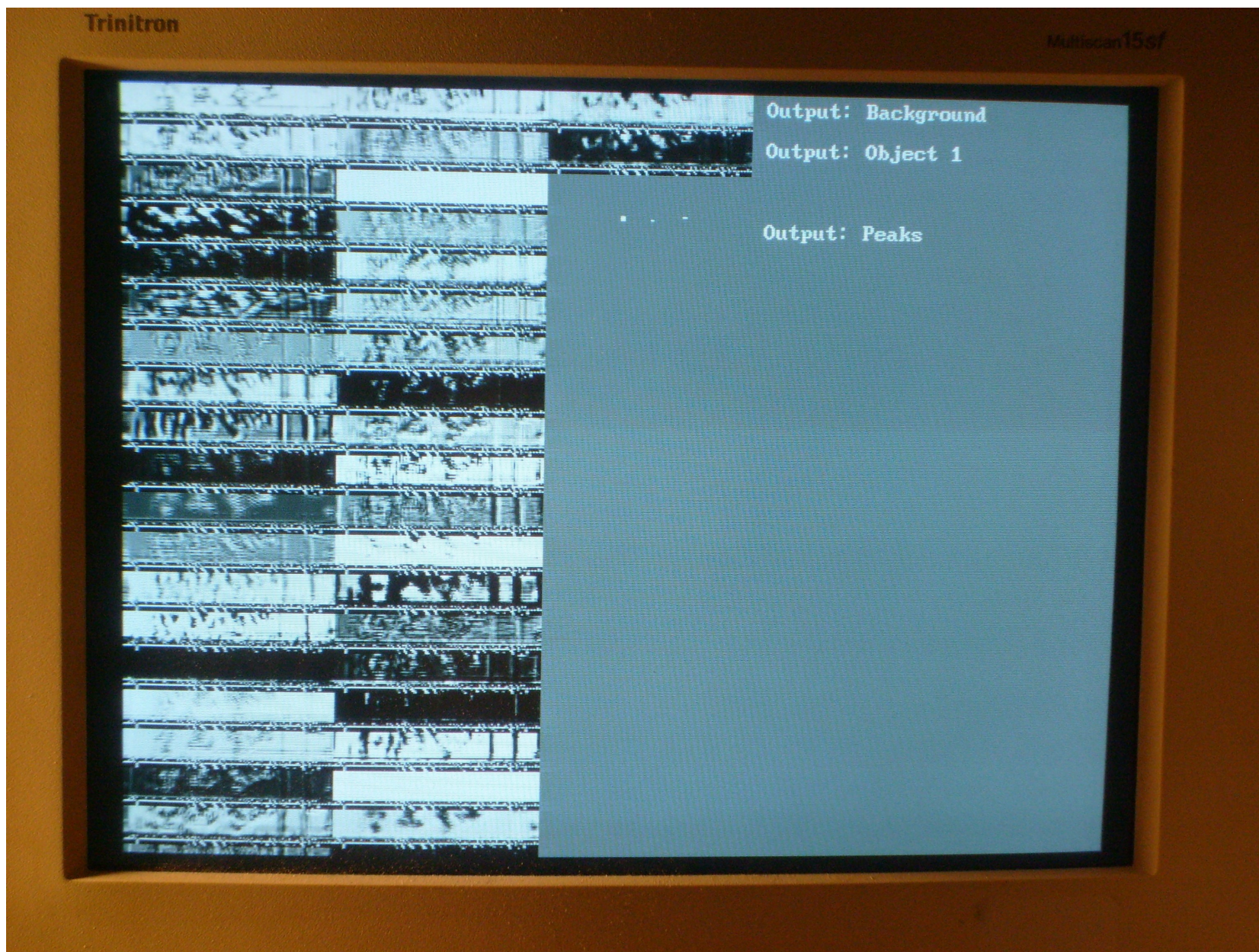
Results



Results



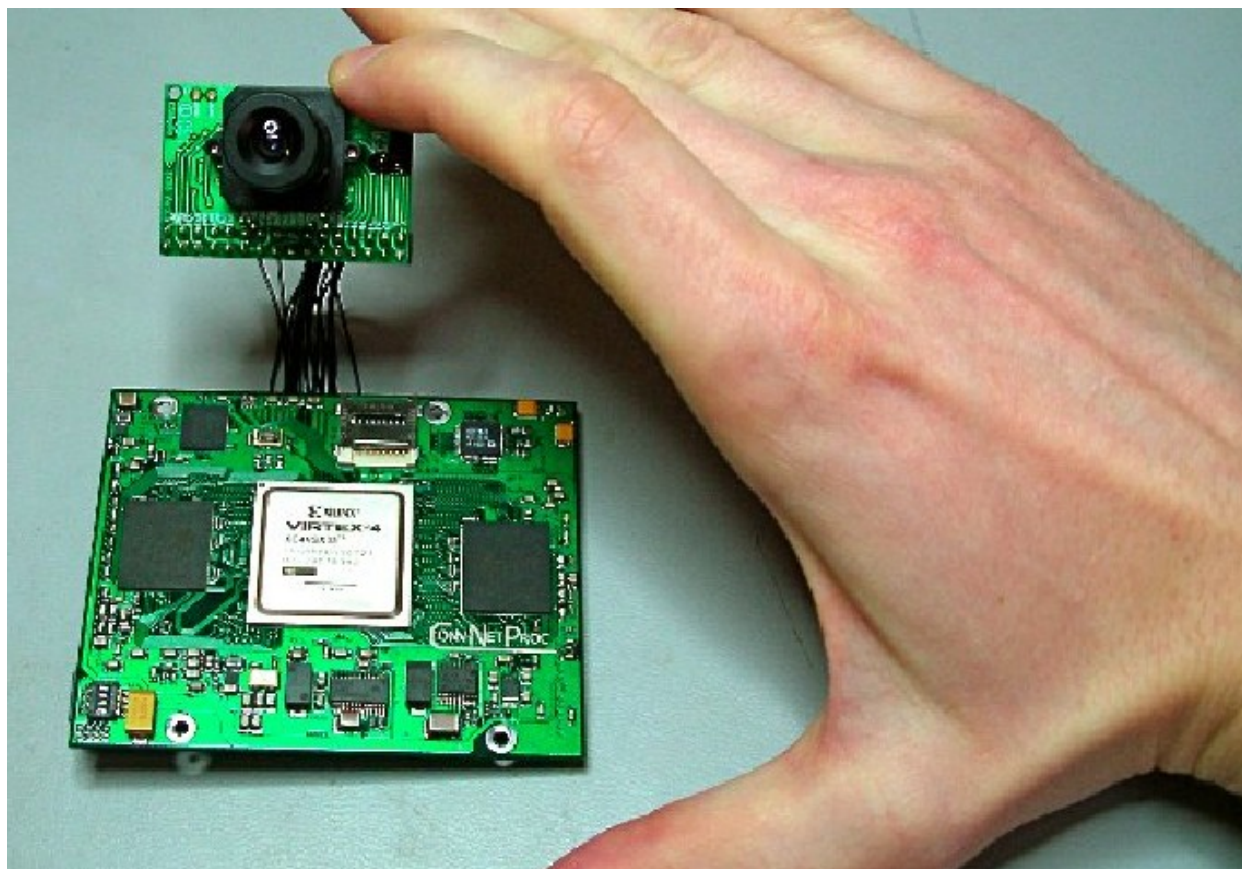
Results



FPGA Custom Board: NYU ConvNet Proc

● Xilinx Virtex 4 FPGA, 8x5 cm board

- ▶ Dual camera port, expansion and I/O port
- ▶ Dual QDR RAM for fast memory bandwidth
- ▶ MicroSD port for easy configuration
- ▶ DVI output
- ▶ Serial communication to optional host

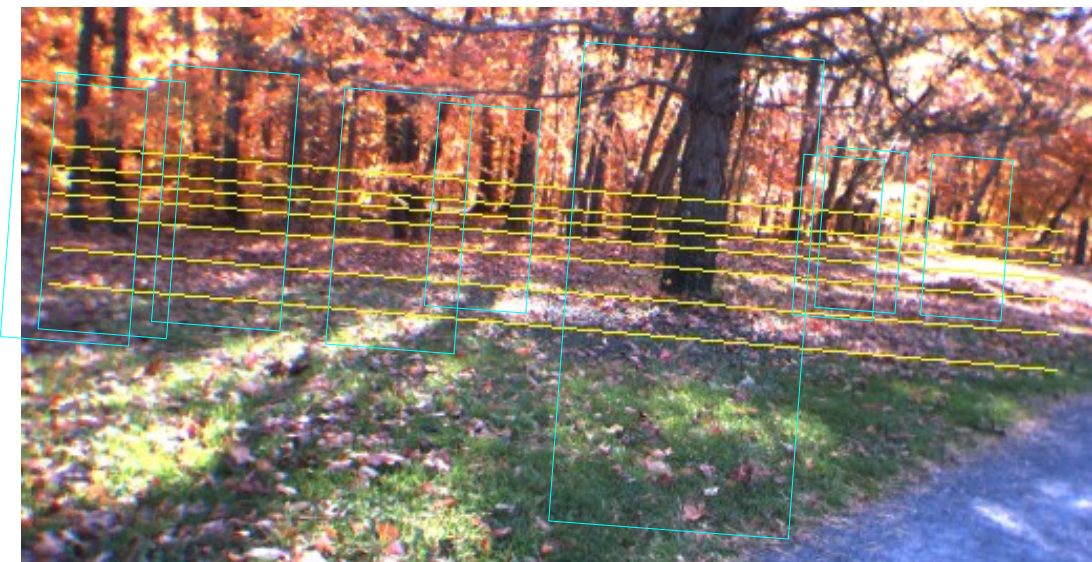


DARPA/LAGR: Learning Applied to Ground Robotics

- 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!**

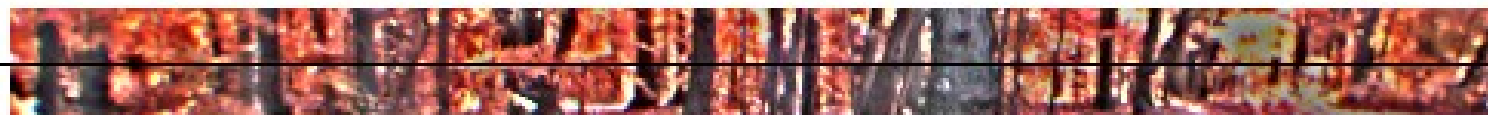


Long Range Vision: Distance Normalization



Pre-processing (125 ms)

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



112.3m to INF, scale: 1.0



50.7m to INF, scale: 1.4



24.2m to INF, scale: 1.9



13.8m to 86.8m, scale: 2.6



9.0m to 34.5m, scale: 3.5



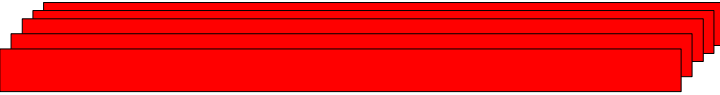
5.8m to 17.6m, scale: 5.0



4.1m to 11.3m, scale: 6.7

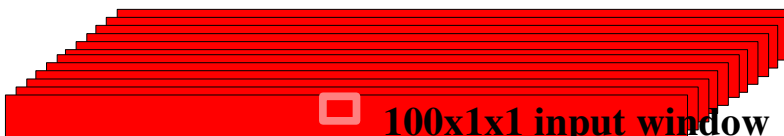
Convolutional Net Architecture

- Operates on 12x25 YUV windows from the pyramid



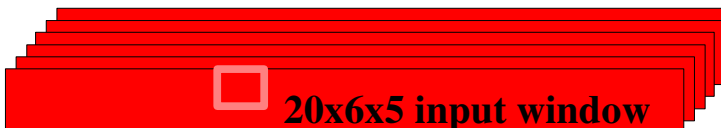
Logistic regression 100 features -> 5 classes

100 features per
3x12x25 input window



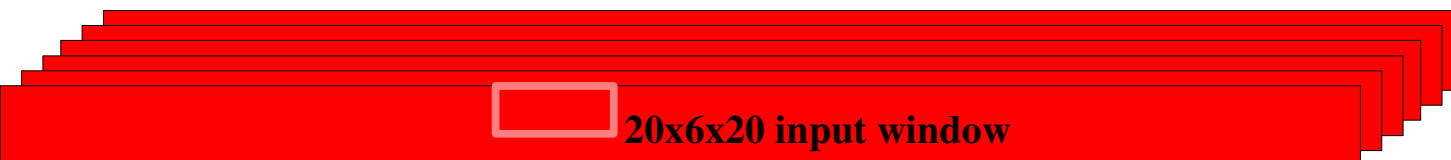
100x1x1 input window

Convolutions with 6x5 kernels



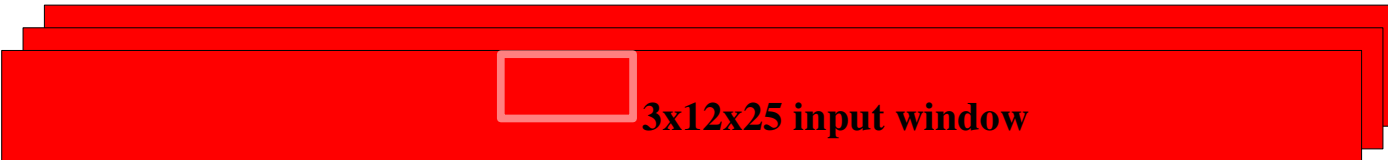
20x6x5 input window

Pooling/subsampling with 1x4 kernels



20x6x20 input window

Convolutions with 7x6 kernels

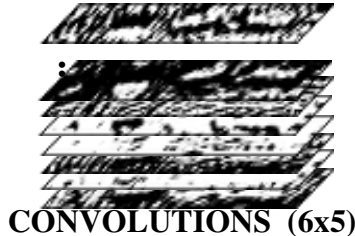


3x12x25 input window

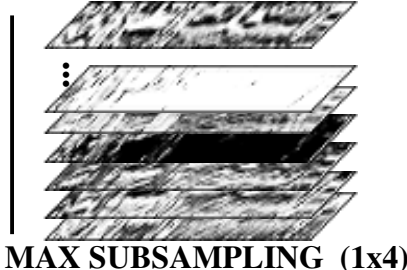
YUV image band
20-36 pixels tall,
36-500 pixels wide

Convolutional Net Architecture

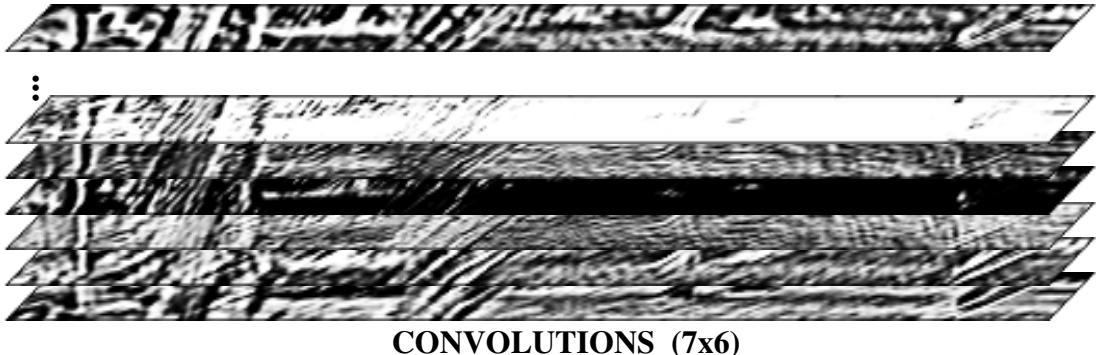
100@25x121



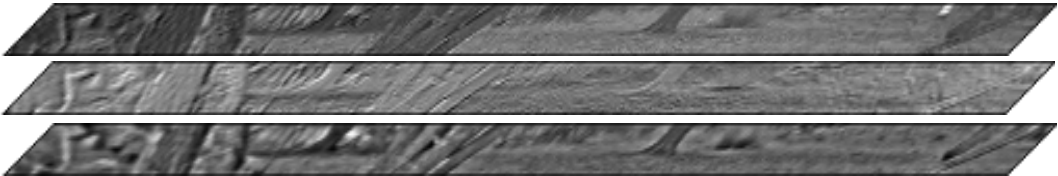
20@30x125



20@30x484



3@36x484



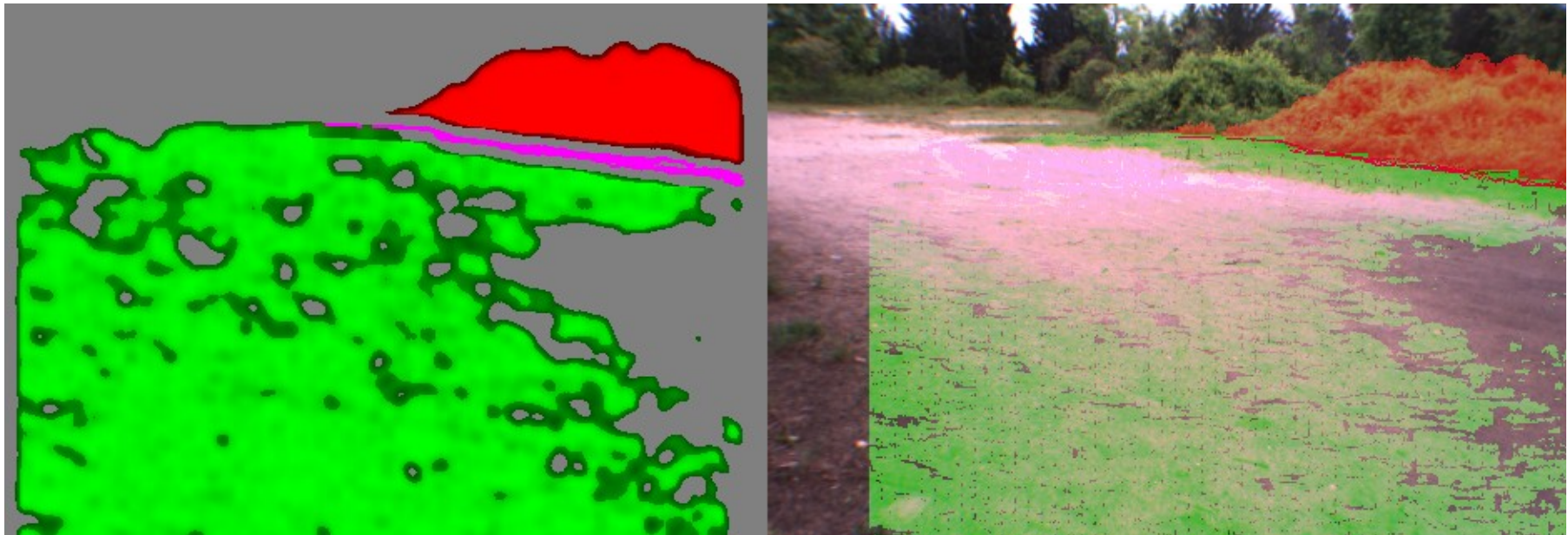
YUV input



Long Range Vision: 5 categories

Online Learning (52 ms)

- Label windows using stereo information – 5 classes



super-ground



ground



footline



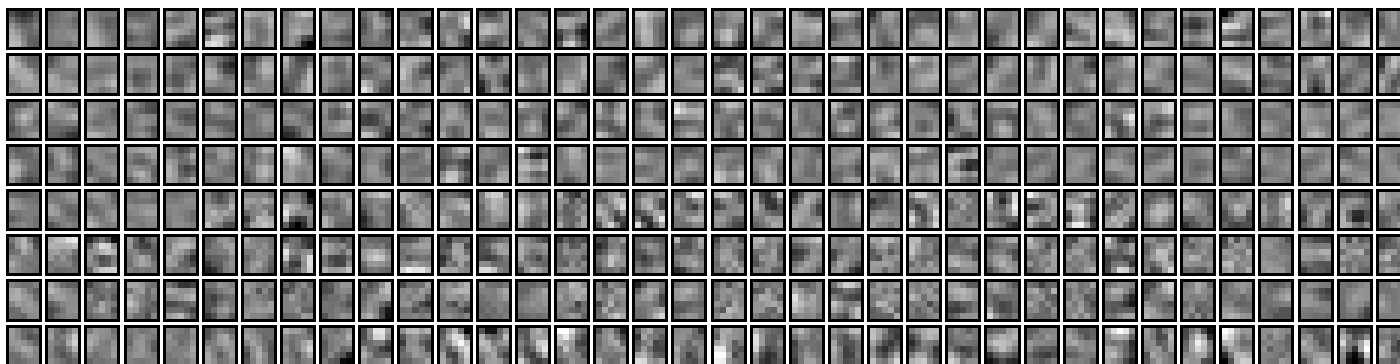
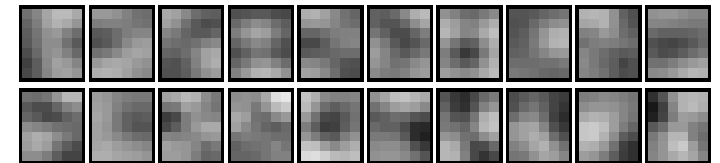
obstacle



super-obstacle

Trainable Feature Extraction

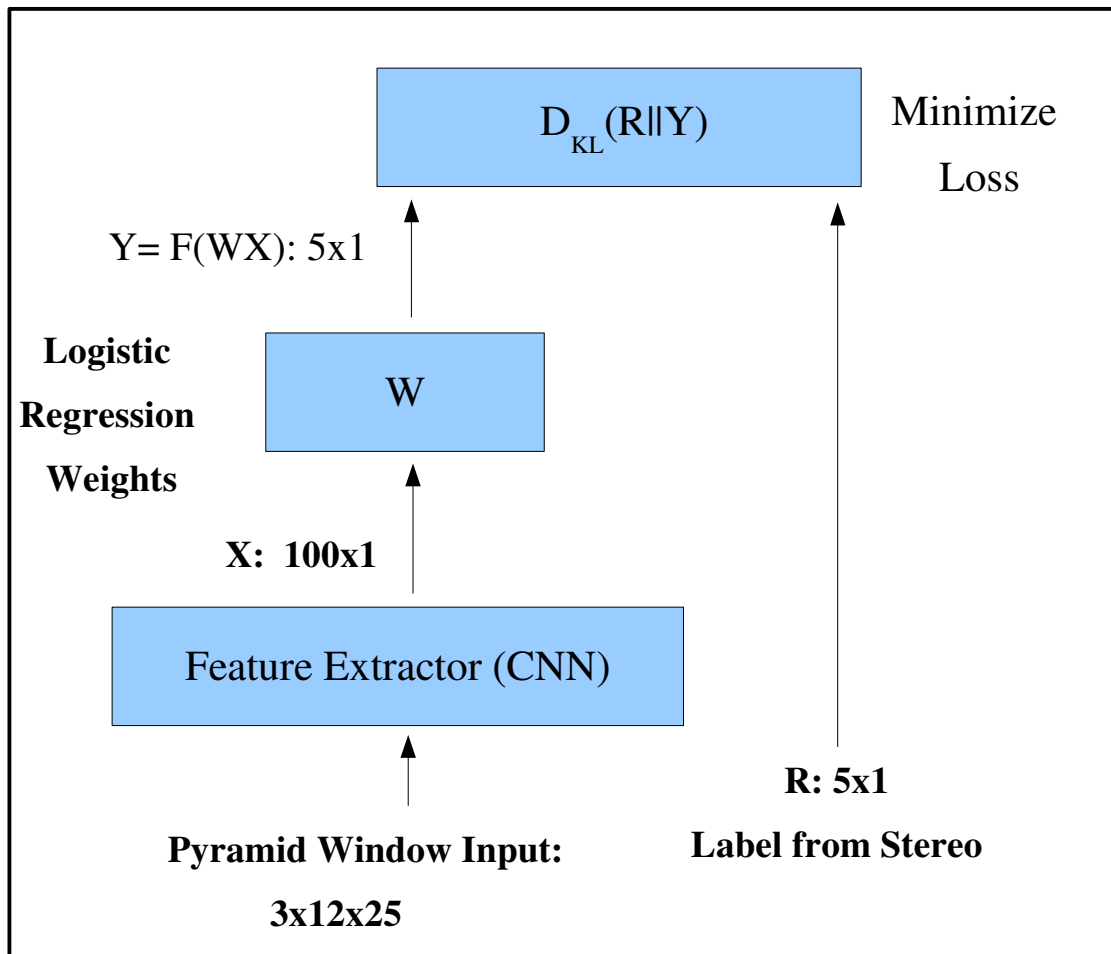
- “Deep belief net” approach to unsupervised feature learning
- Two stages are trained in sequence
 - each stage has a layer of convolutional filters and a layer of horizontal feature pooling.
 - Naturally shift invariant in the horizontal direction
- Filters of the convolutional net are trained so that 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: the Classifier

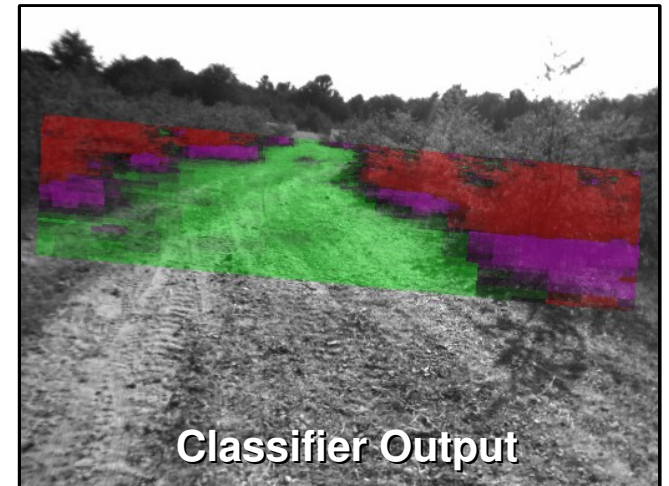
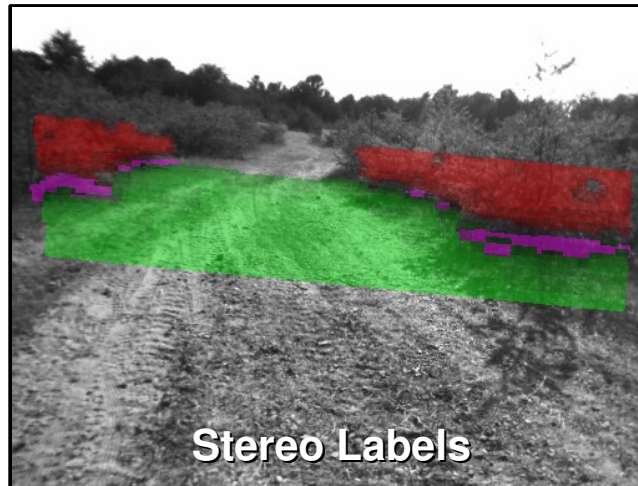
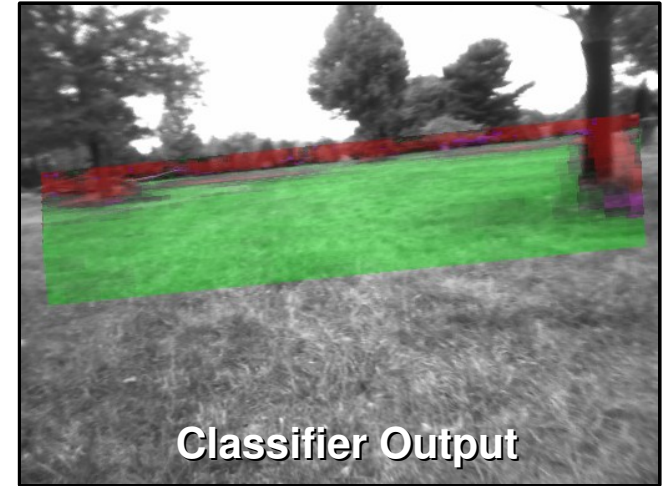
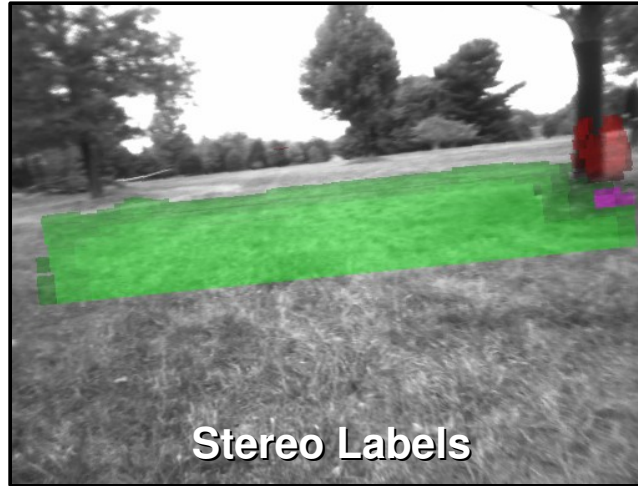
Online Learning (52 ms)

- Train a logistic regression on every frame, with cross entropy loss function

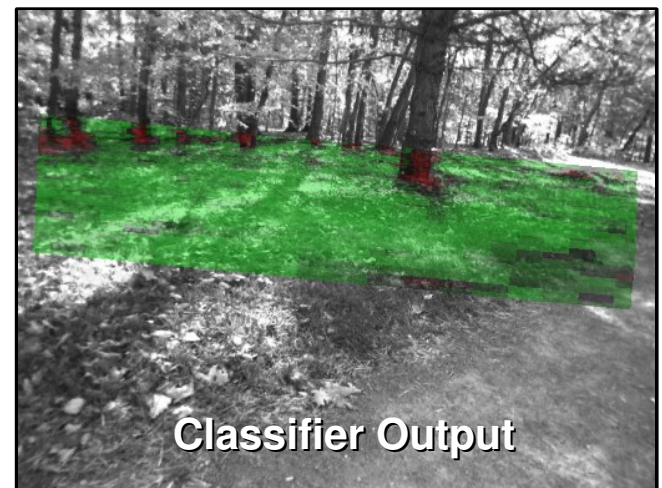
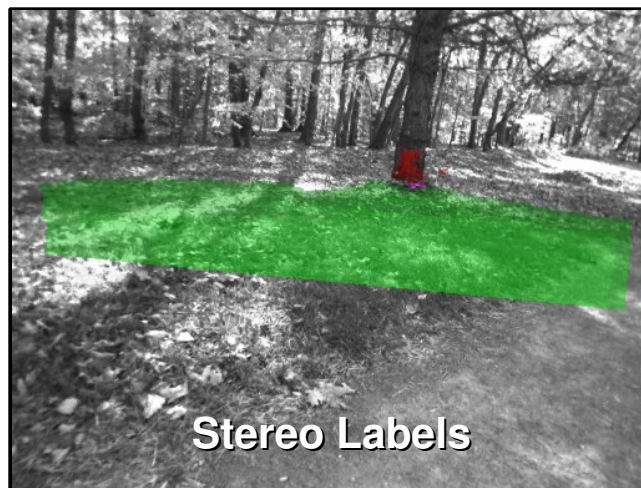
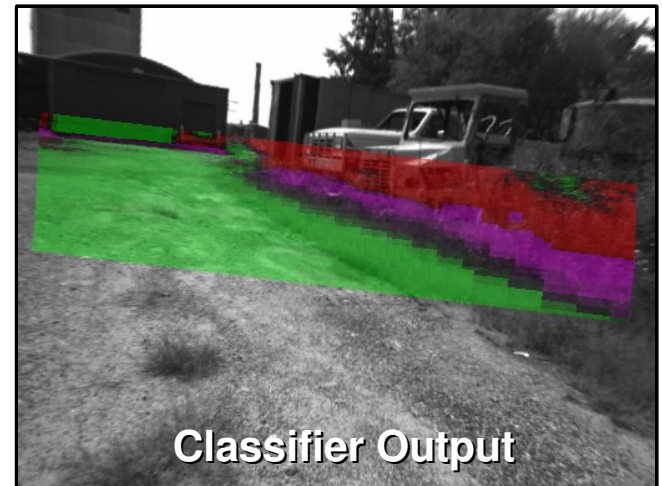
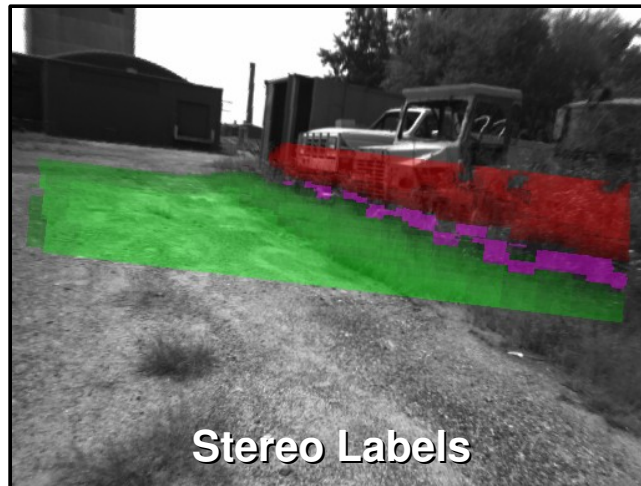


- 5 categories are learned
- 750 samples of each class are kept in a ring buffer: short term memory.
- Learning “snaps” to new environment in about 10 frames
- Weights are trained with stochastic gradient descent
- Regularization by decay to default weights

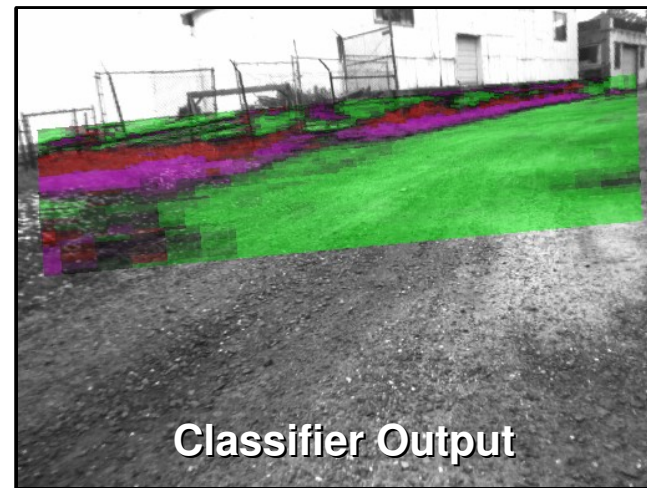
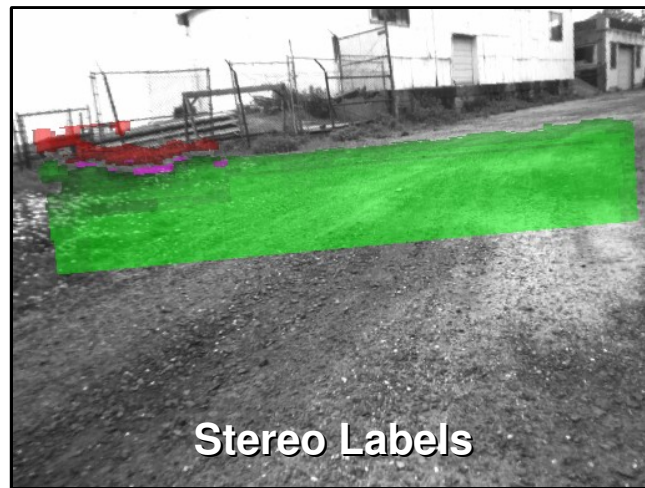
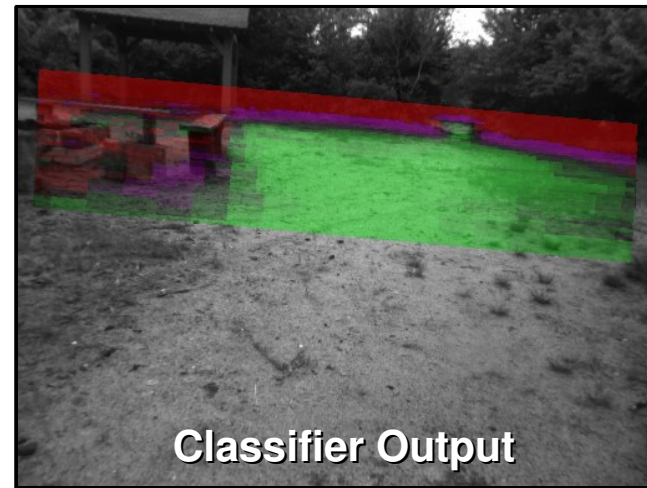
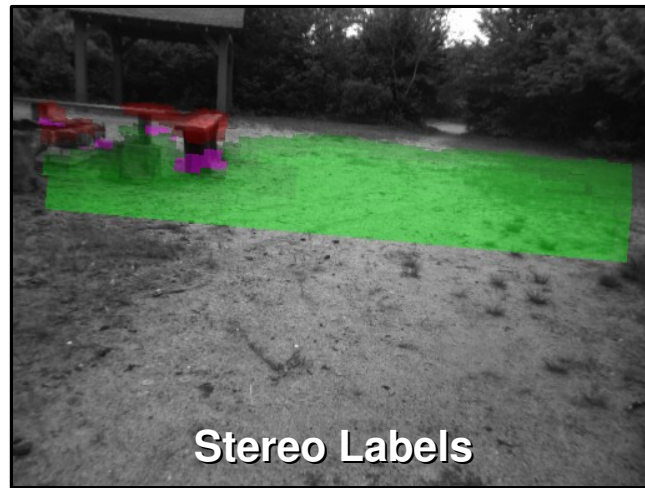
Long Range Vision Results

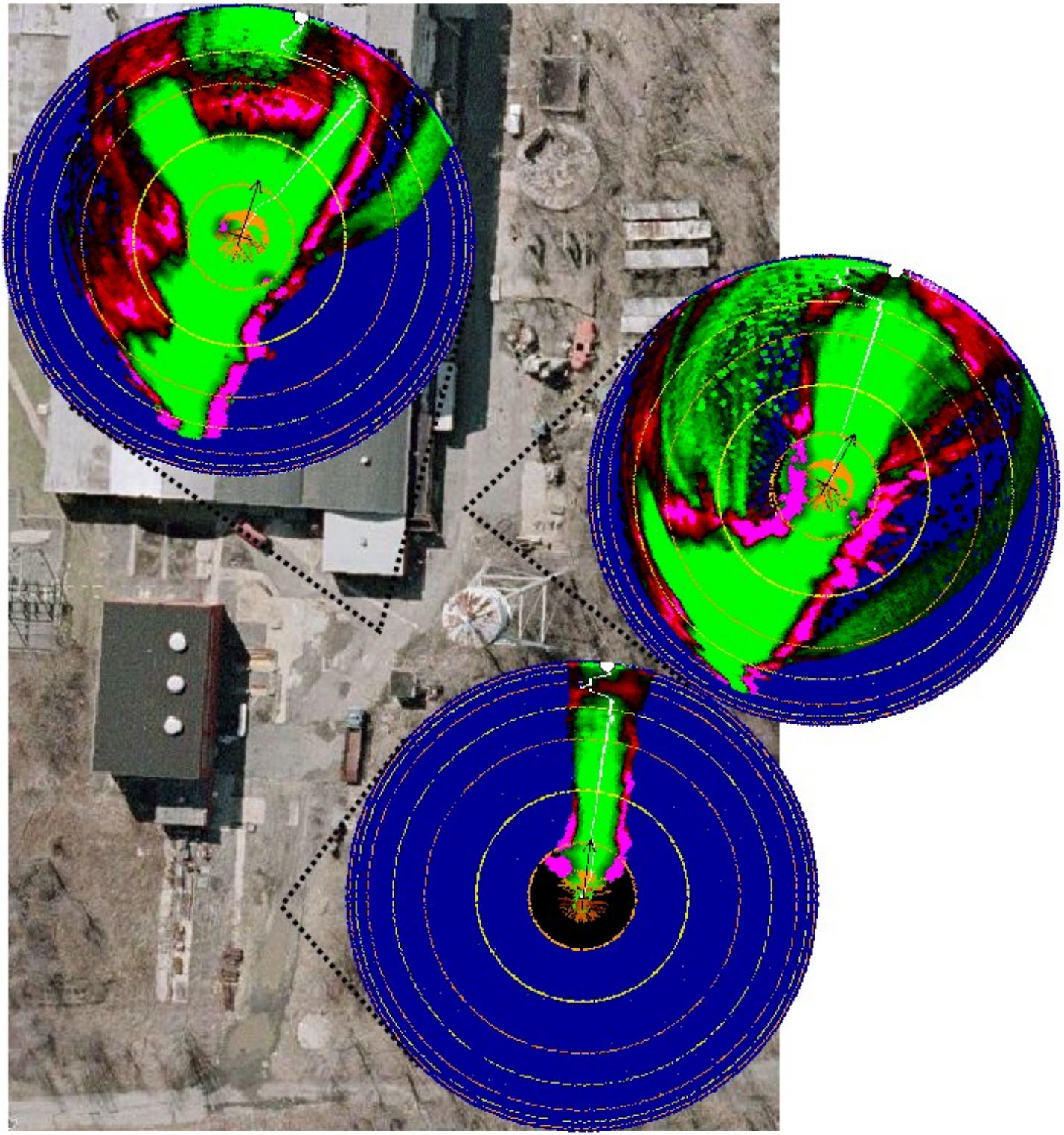


Long Range Vision Results



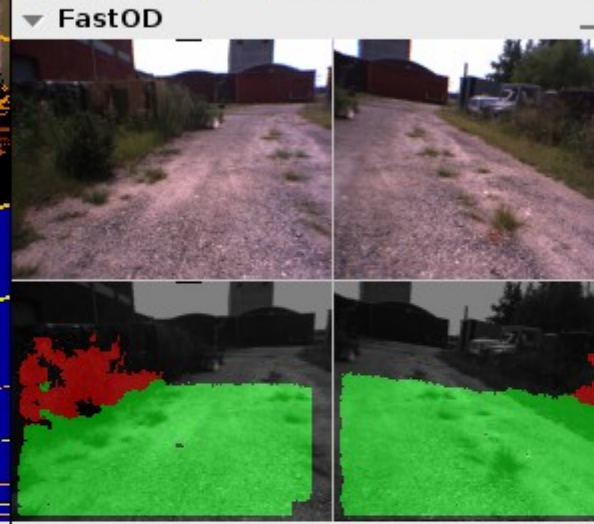
Long Range Vision Results



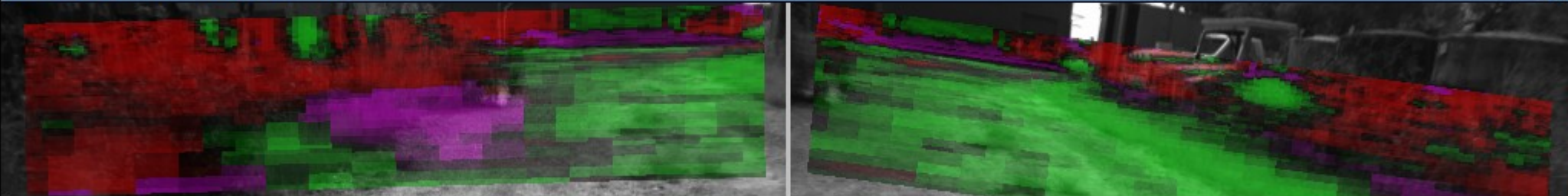


Vehicle Map (Hyperbolic Polar map)

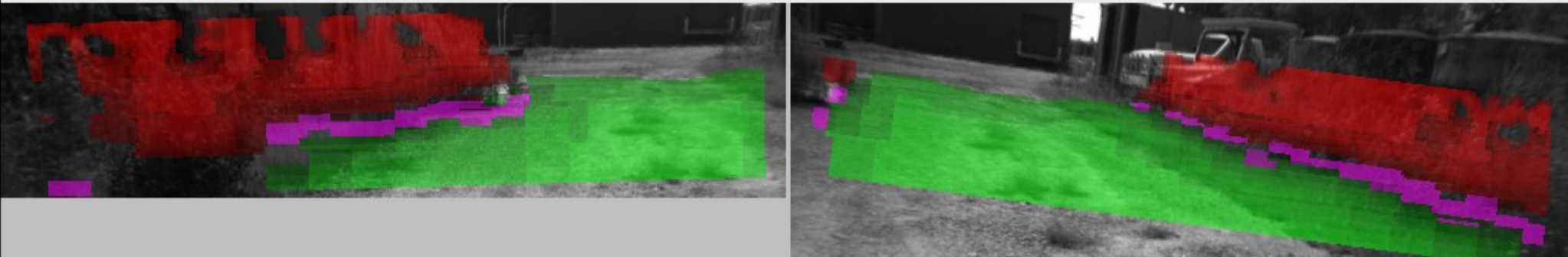
- Legend
 - Goal
 - Path Planning
 - ▬ Trajectories
 - ▬ Traversable
 - ▬ Uncertain
 - ▬ Quasi-Lethal
 - ▬ Lethal
 - ▬ Bumper/Stuck
 - ▬ Unseen
- 200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m



FarOD Neural Network Labels

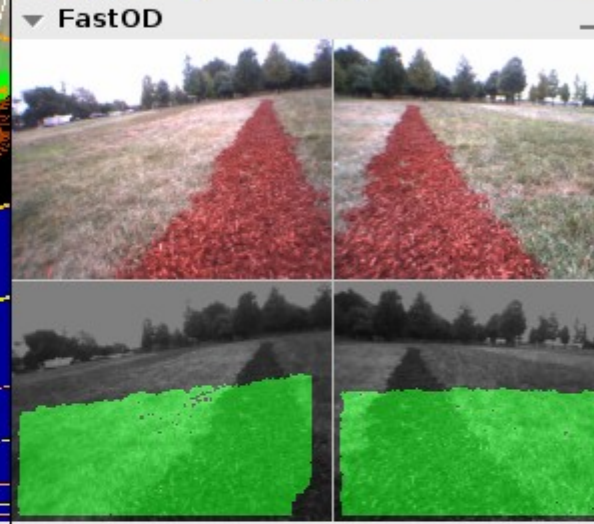
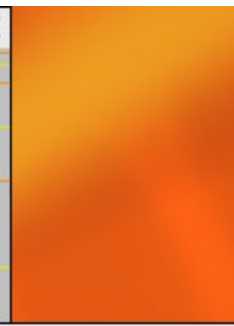
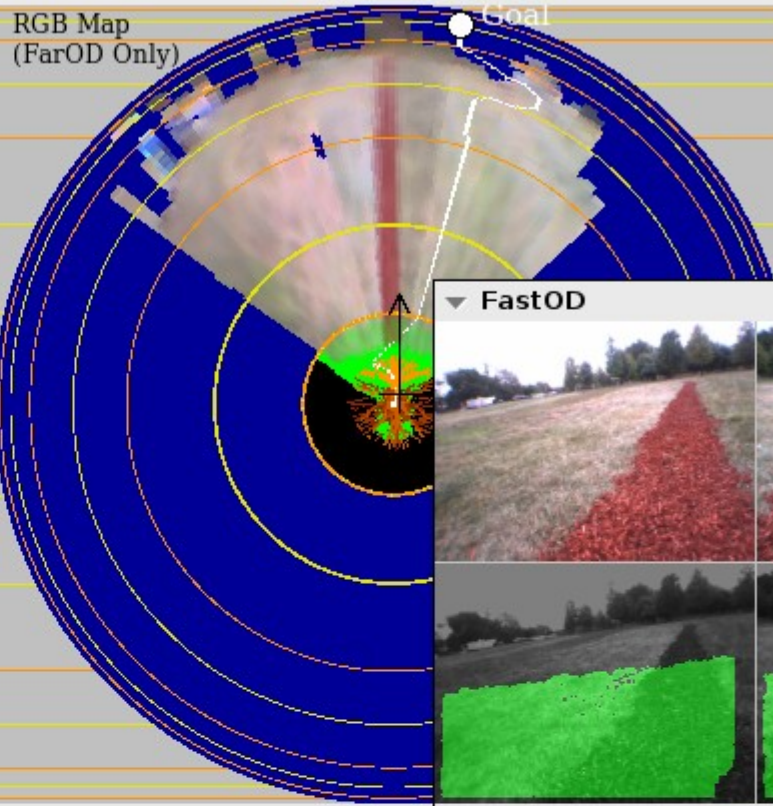
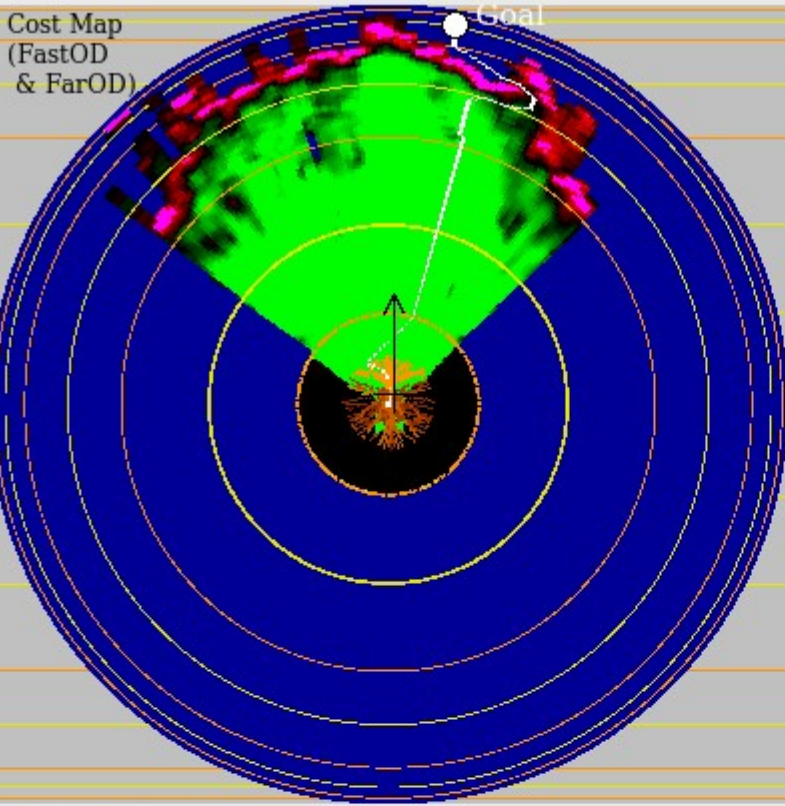


FarOD Stereo: Input labels to Neural Network

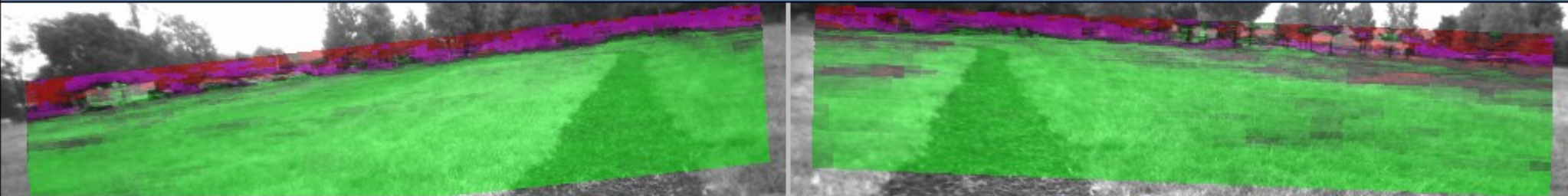


Vehicle Map (Hyperbolic Polar map)

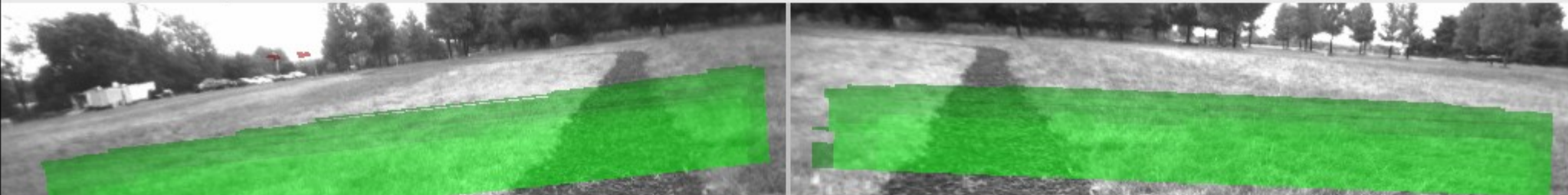
- Legend
- 200m
- 100m
- 50m
- Goal
- Path Planning
- Trajectories
- Traversable
- Uncertain
- Quasi-Lethal
- Lethal
- Bumper/Stuck
- Unseen
- 25m
- 15m
- 10m
- 5m
- 5m
- 10m
- 15m
- 25m
- 50m
- 100m
- 200m



FarOD Neural Network Labels

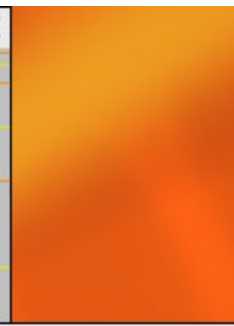
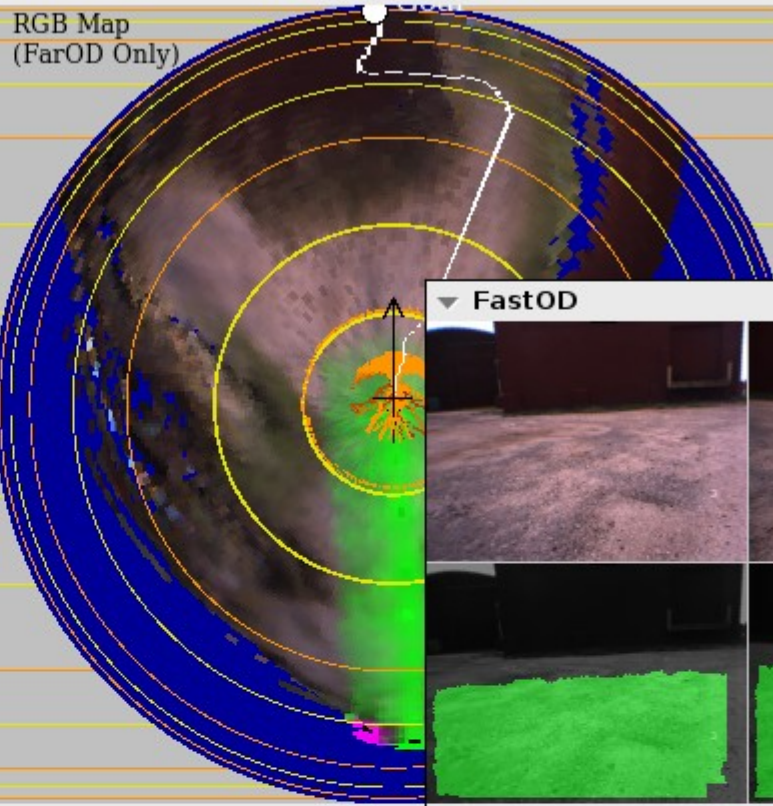
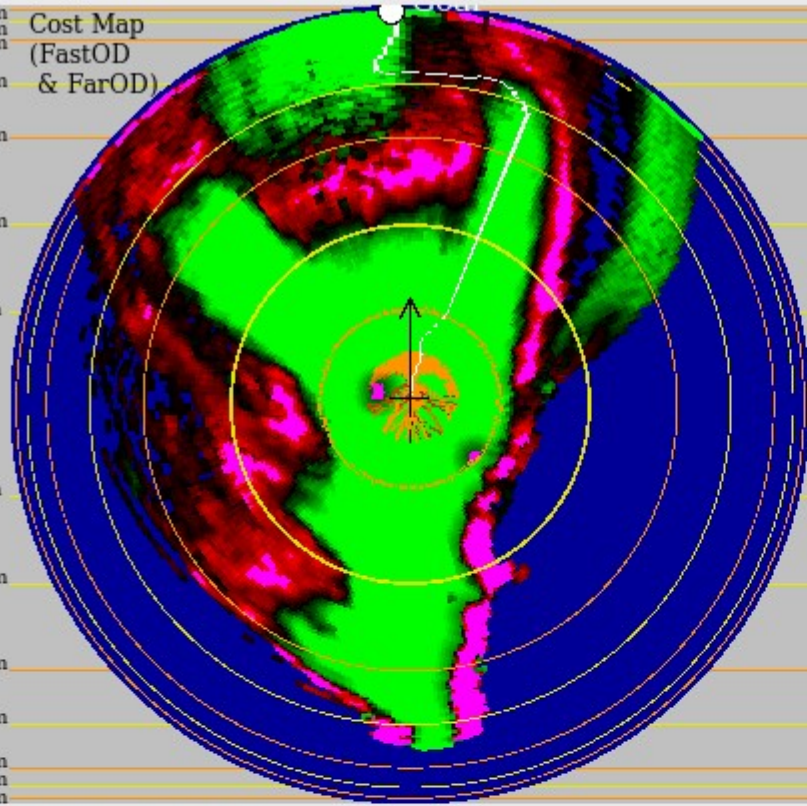


FarOD Stereo: Input labels to Neural Network

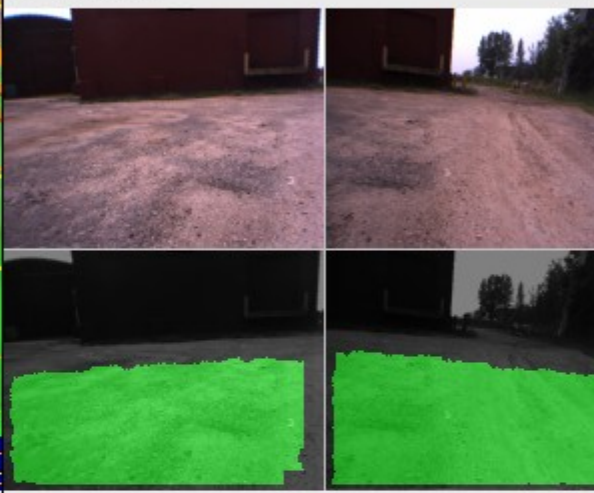


Vehicle Map (Hyperbolic Polar map)

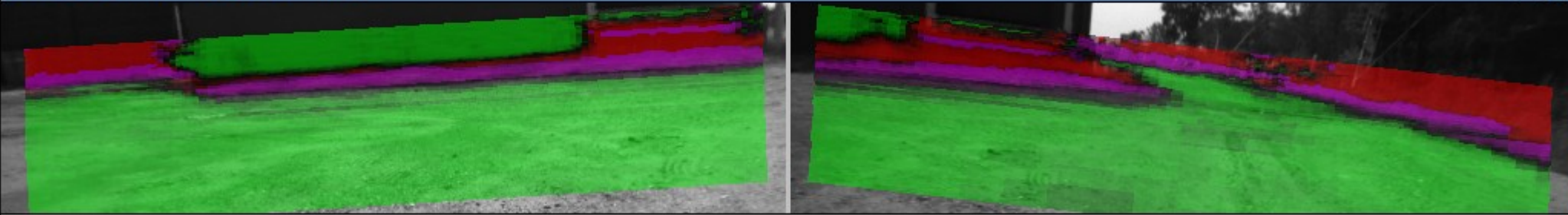
- Legend
- 200m
- 100m
- 50m
- Cost Map (FastOD & FarOD)
- Goal
- Path Planning
- Trajectories
- Traversable
- Uncertain
- Quasi-Lethal
- Lethal
- Bumper/Stuck
- Unseen
- 5m
- 10m
- 15m
- 25m
- 50m
- 100m
- 200m



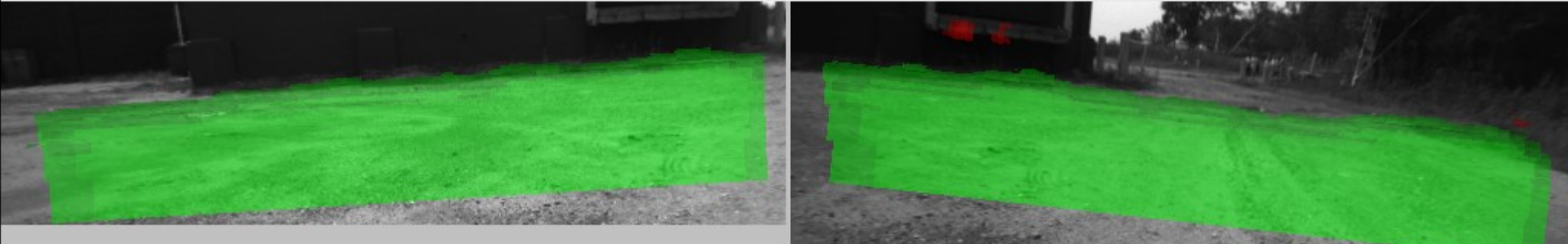
FastOD



FarOD Neural Network Labels

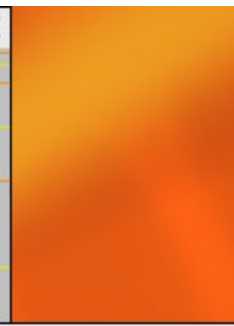
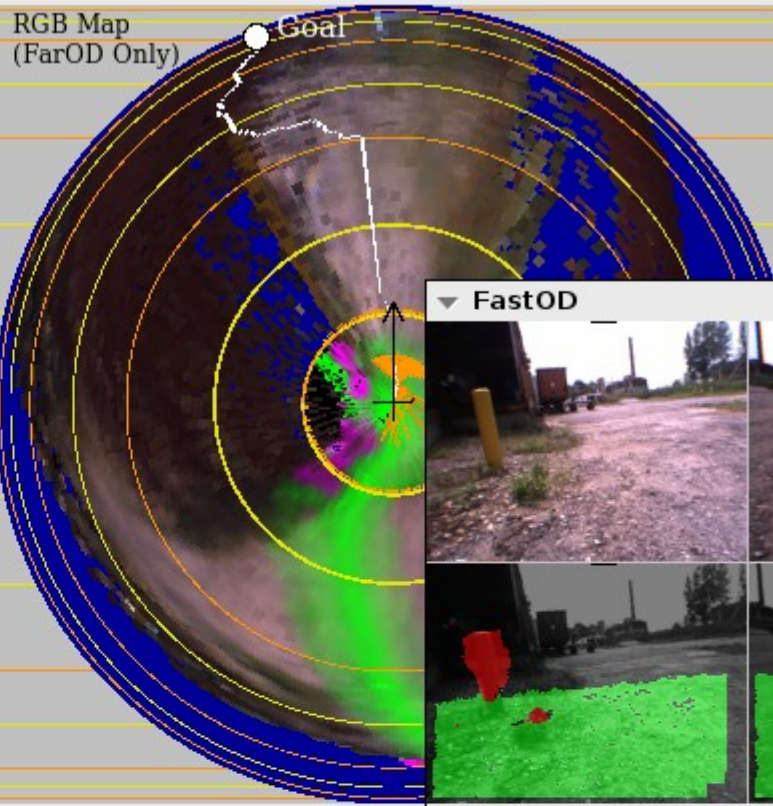
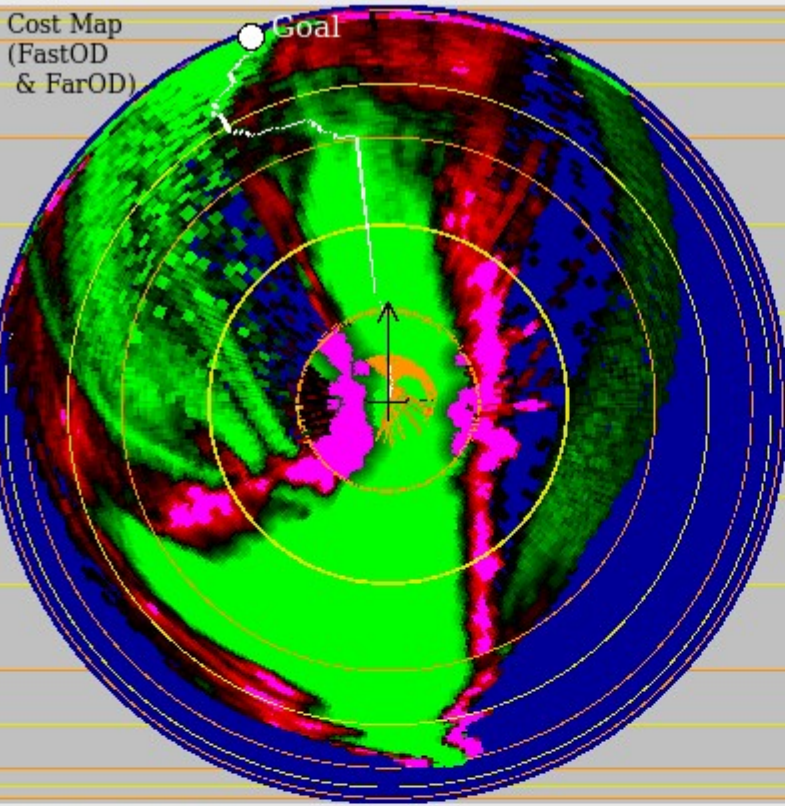


FarOD Stereo: Input labels to Neural Network

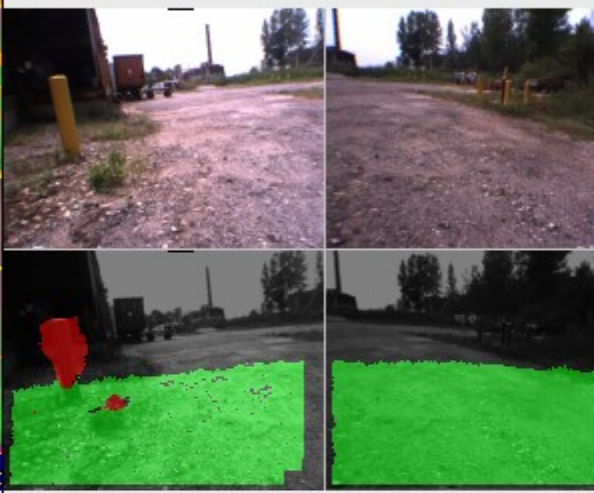


Vehicle Map (Hyperbolic Polar map)

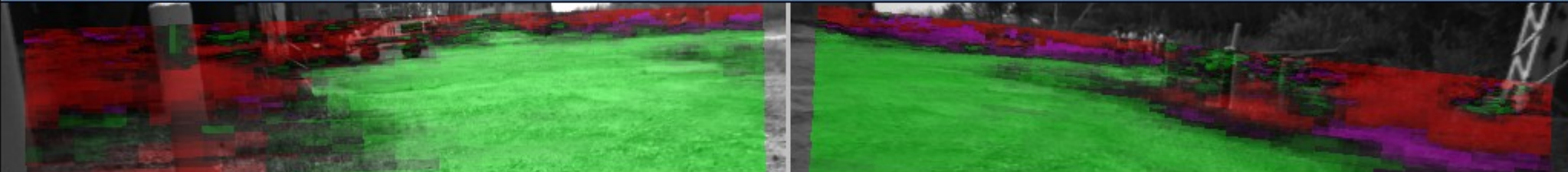
- Legend
- 200m
- 100m
- 50m
- Goal
- Path Planning
- Trajectories
- Traversable
- Uncertain
- Quasi-Lethal
- Lethal
- Bumper/Stuck
- Unseen
- 25m
- 15m
- 10m
- 5m
- 5m
- 10m
- 15m
- 25m
- 50m
- 100m
- 200m



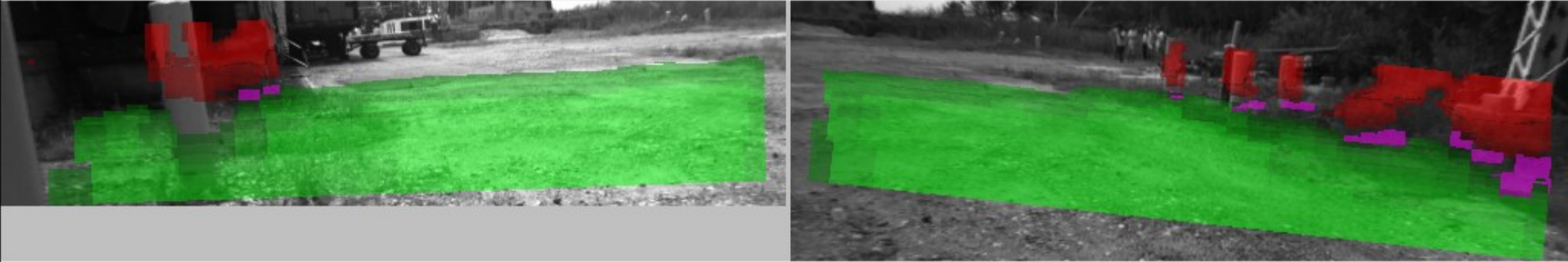
FastOD



FarOD Neural Network Labels

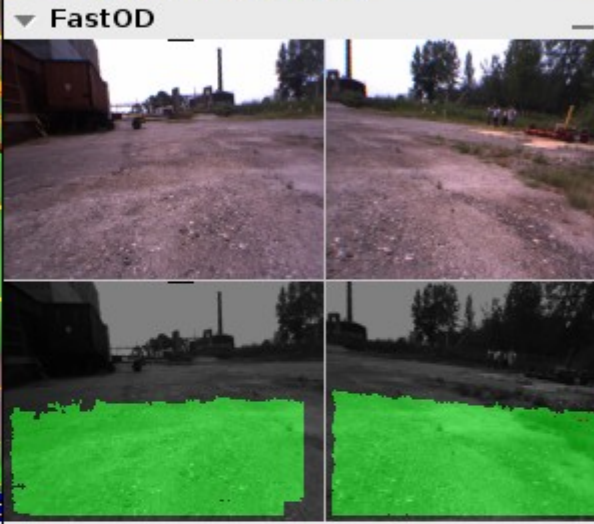
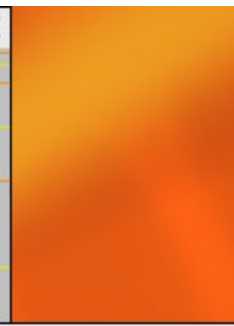
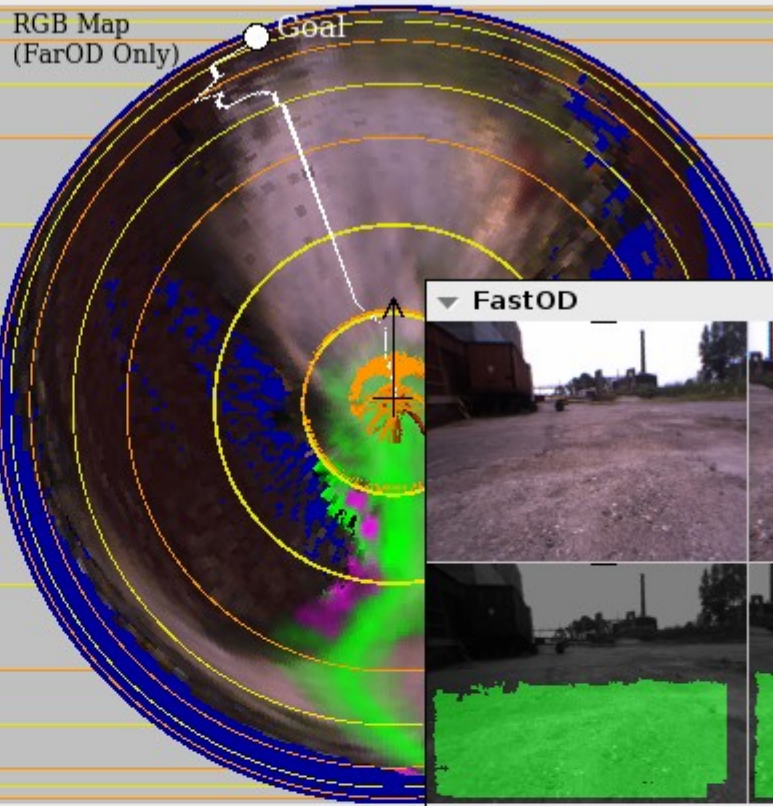
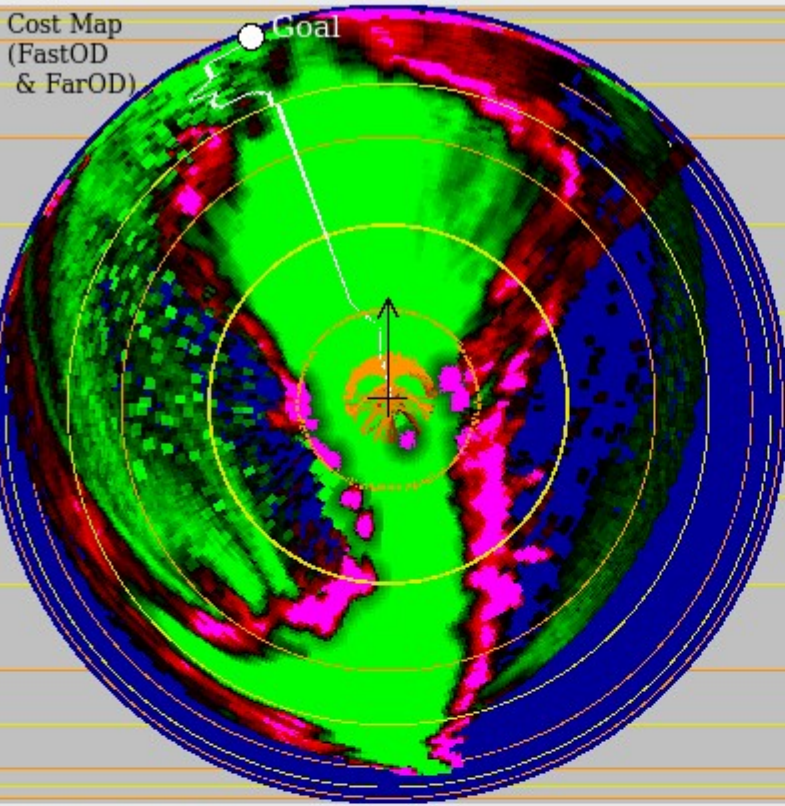


FarOD Stereo: Input labels to Neural Network

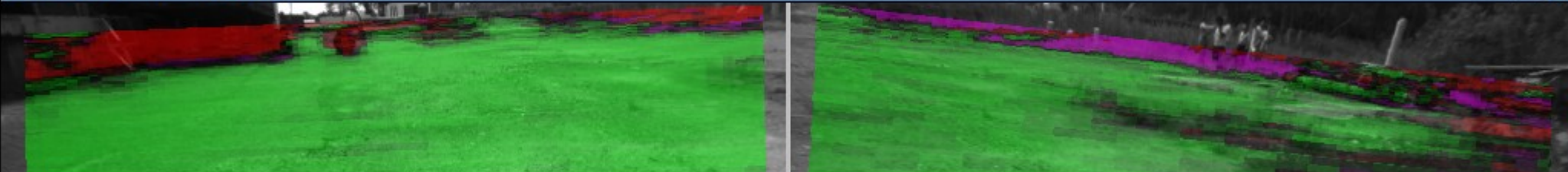


Vehicle Map (Hyperbolic Polar map)

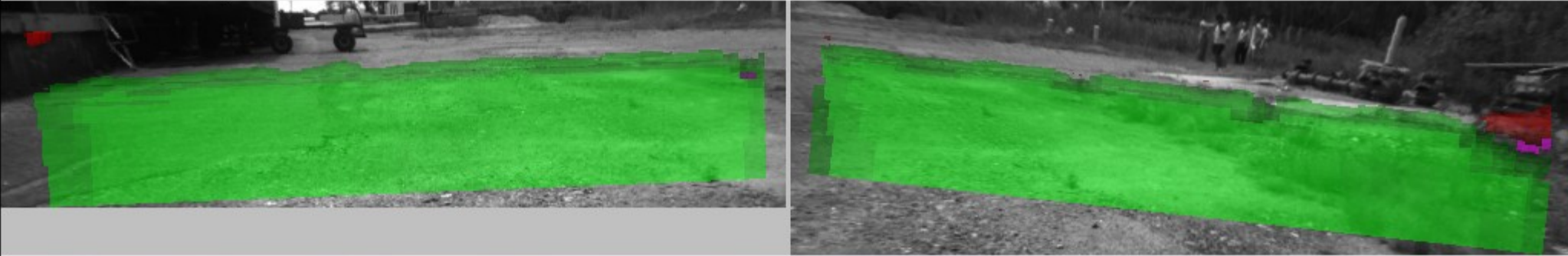
- Legend
 - Goal
 - Path Planning
 - Trajectories
 - Traversable
 - Uncertain
 - Quasi-Lethal
 - Lethal
 - Bumper/Stuck
 - Unseen
- 200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m



FarOD Neural Network Labels



FarOD Stereo: Input labels to Neural Network

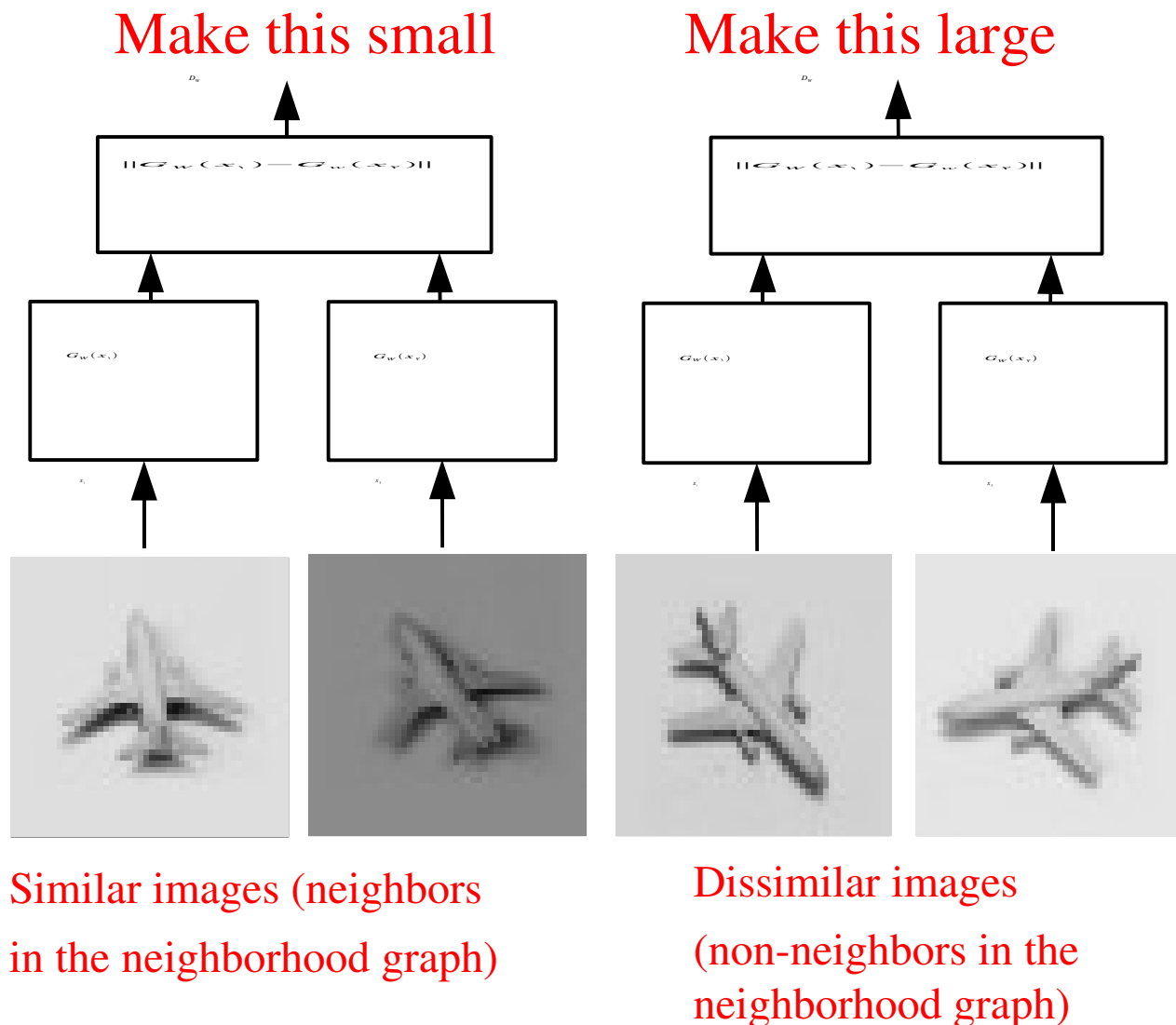


Another way to Learn Deep Invariant Features: DrLIM

Hadsell, Chopra, LeCun CVPR 06], also [Weston & Collobert ICML 08 for language models]

Loss function:

- ▶ Outputs corresponding to input samples that are neighbors in the neighborhood graph should be nearby
- ▶ Outputs for input samples that are not neighbors should be far away from each other



Learning Deep Invariant Features with DrLIM

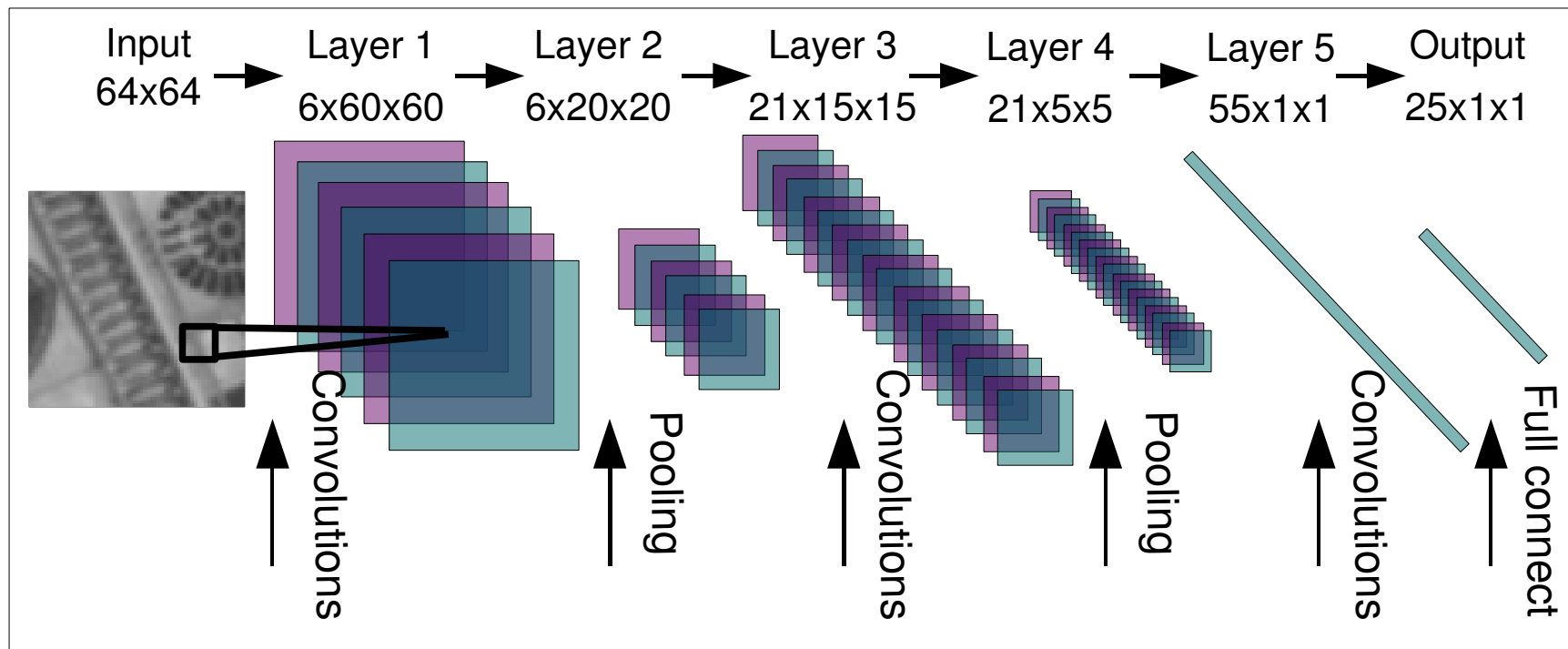
Co-location patch data

- multiple tourist photos
- 3d reconstruction
- groundtruth matches



Uses temporal consistency

- ▶ Pull together outputs for same patch
- ▶ Push away outputs for different patches



Feature Learning for traversability prediction (LAGR)

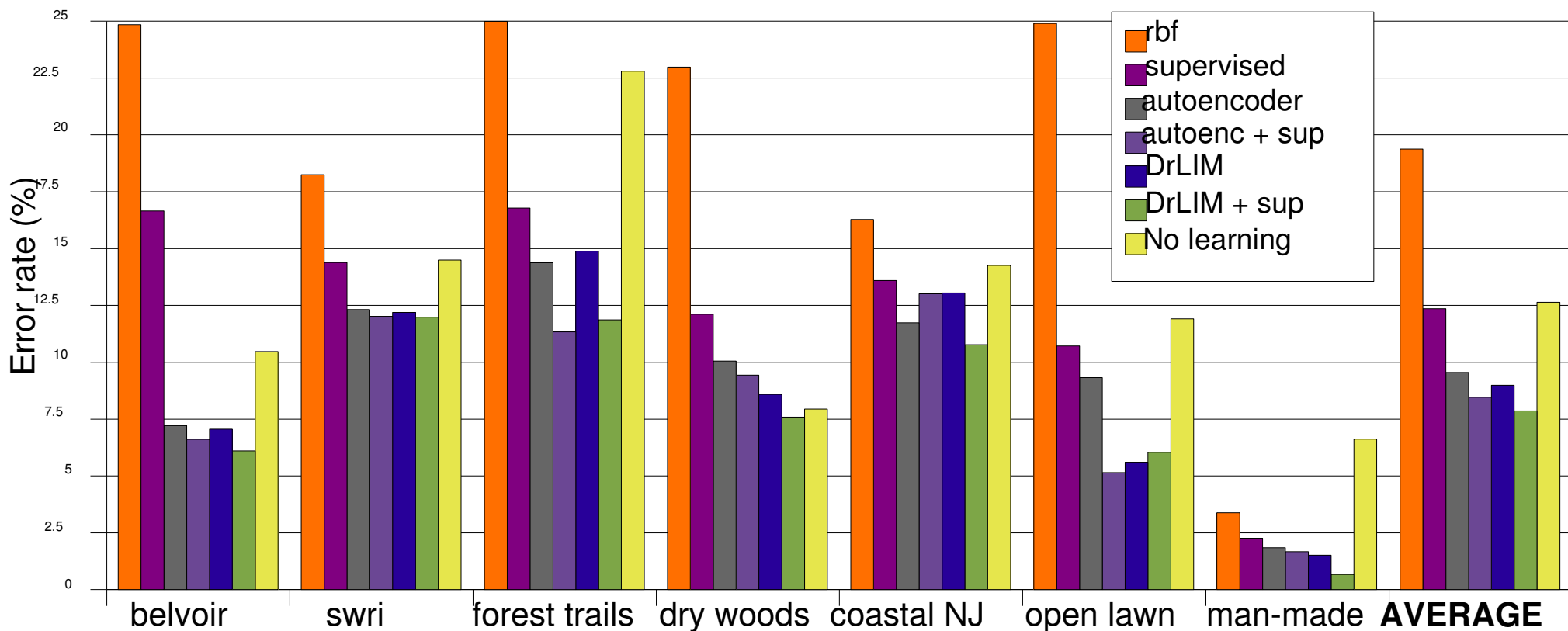
Comparing

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



Testing on hand-labeled groundtruth frames – binary labels

Comparison of Feature Extractors on Groundtruth Data



The End