Deep Learning

Yann LeCun

The Courant Institute of Mathematical Sciences

New York University

joint work with: Marc'Aurelio Ranzato,

Y-Lan Boureau, Koray Kavackuoglu,

Fu-Jie Huang, Raia Hadsell

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 netd 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?

The Traditional "Shallow" Architecture for Recognition

Pre-processing / Feature Extraction

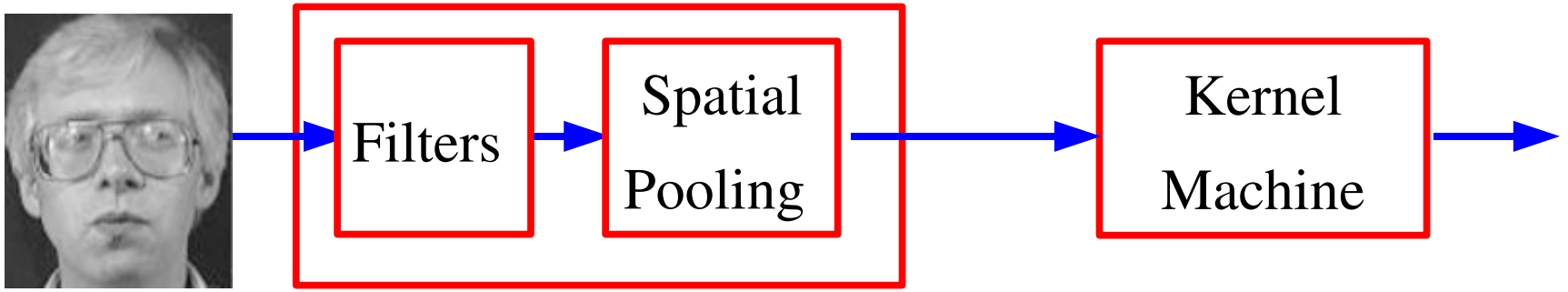
Trainable Classifier

this part is mostly hand-crafted

Internal Representation

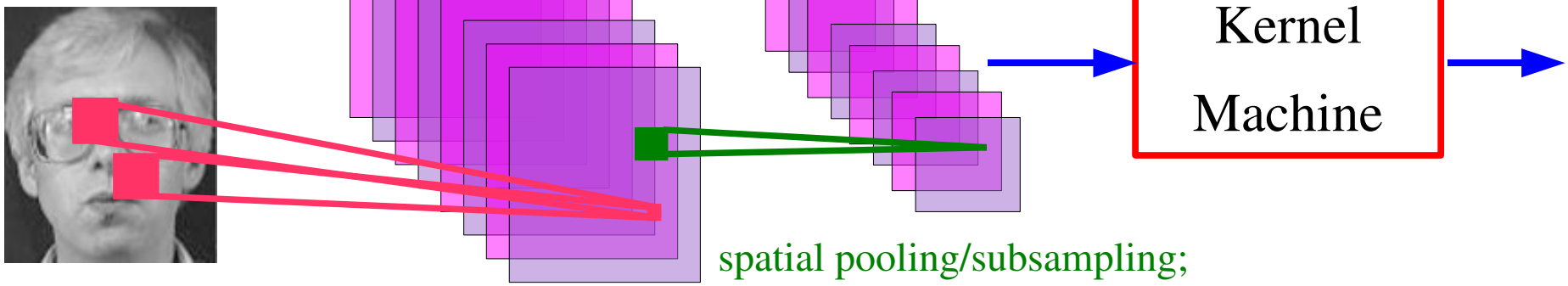
- **The raw input is pre-processed through a hand-crafted feature extractor**
- **The trainable classifier is often generic (task independent)**
- **The most common Machine Learning architecture: the Kernel Machine**
  - kernel machines are shallow

Most Common Architecture Today:
Fixed Dense Features + Spatial Pooling + Kernel Classifier



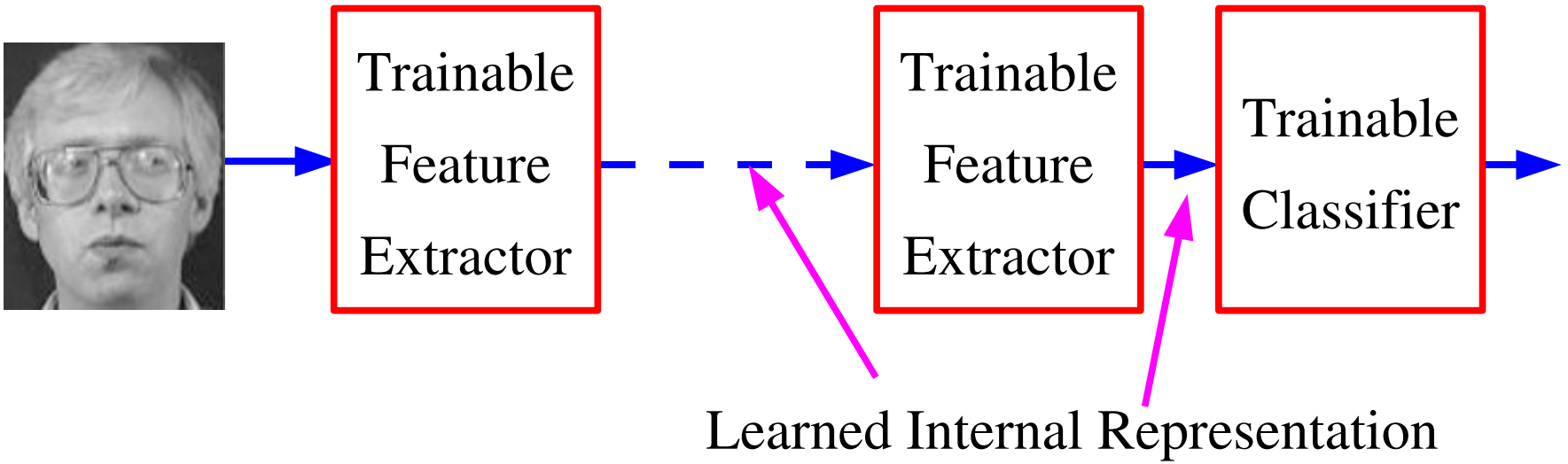
- It's shallow
- The features are not learned
- But it's the only architecture that can work with 30 training samples/class

The Most Common Recognition Architecture Today:
Fixed Dense Features + Spatial Pooling + Kernel Classifier



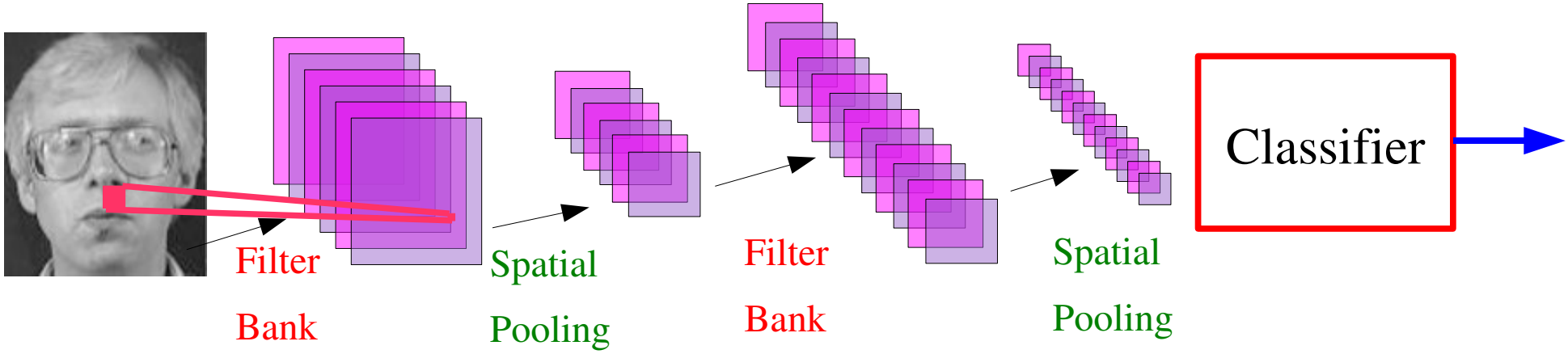
- **It's shallow**
- **The features are not learned**
- **But it's the only architecture that can work with 30 training samples/class**

"Deep" Learning: Learning Internal 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 solves the specificity/invariance dilemma**

Multi-stage Hubel-Wiesel Architecture



- **Stacking multiple 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]**
  - fully supervised feature learning
- **HMAX & friends [Poggio's group 2002-2006, Lowe 2006]**
  - no real feature learning (just storing templates)

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**

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

- **Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition**
  - they can represent more complex functions with less "hardware"

- **We need an efficient parameterization of the class of functions that we need to build intelligent machines (the "AI-set")**

Why would Deep Architectures be more efficient?

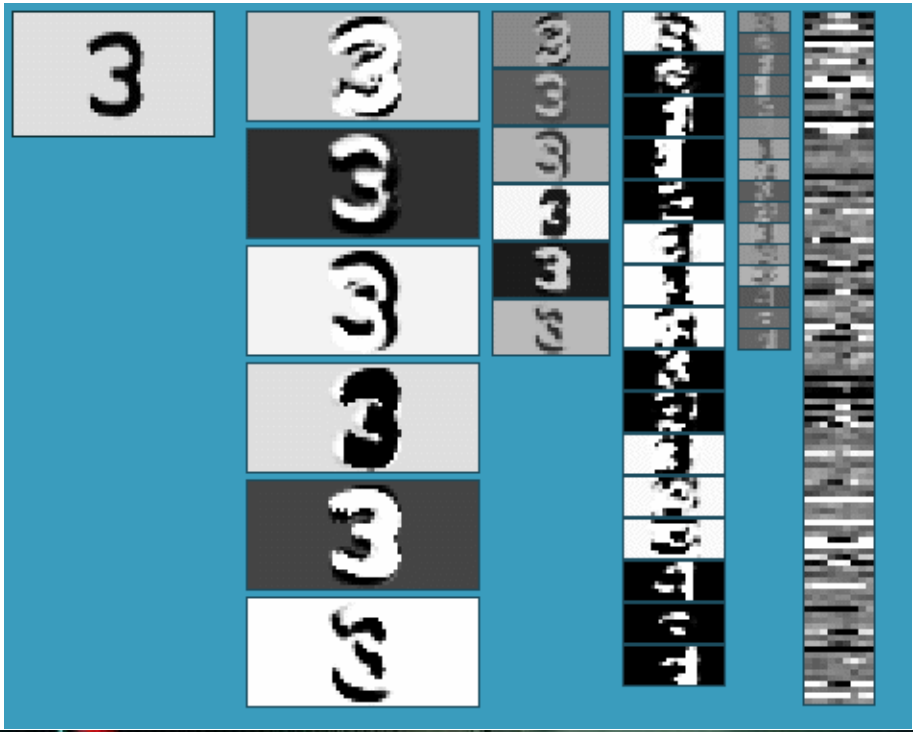
- **A deep architecture trades space for time**
  - more layers (more sequential computation),
  - but less hardware (less parallel computation).
  - Depth-Breadth tradoff
- **Example1: N-bit parity**
  - requires N-1 XOR gates in a tree of depth log(N).
  - requires an exponential number of gates of we restrict ourselves to 2 layers (DNF formula with exponential number of minterms).
- **Example2: circuit for addition of 2 N-bit binary numbers**
  - Requires O(N) gates, and O(N) layers using N one-bit adders with ripple carry propagation.
  - Requires lots of gates (some polynomial in N) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
  - Bad news: almost all boolean functions have a DNF formula with an exponential number of minterms O(2^N).....

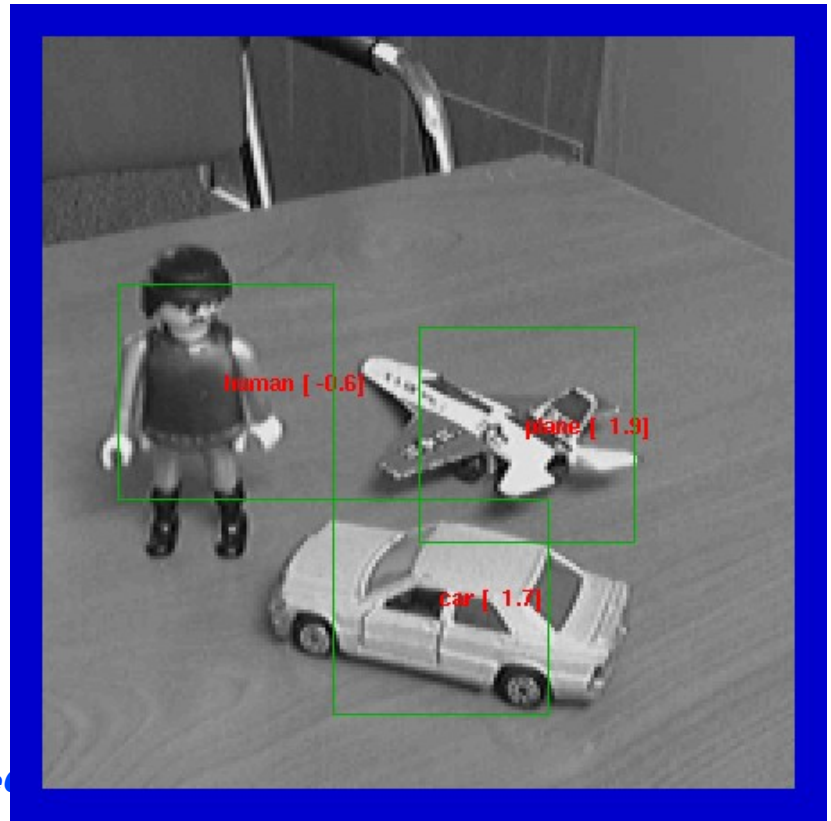
Strategies (after Hinton 2007)

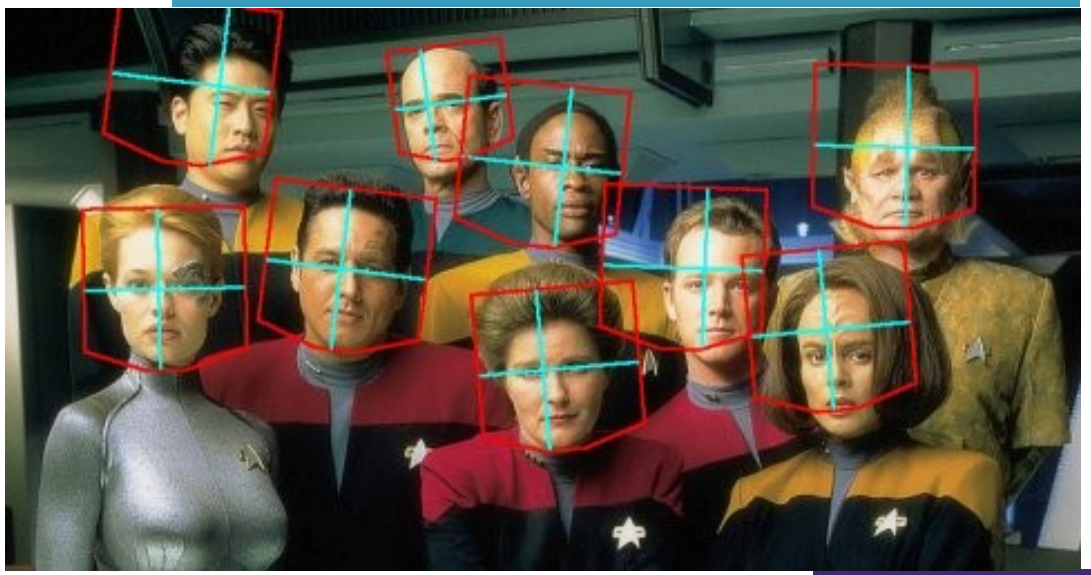
- **Defeatism: since no good parameterization of the AI-set is available, let's parameterize a much smaller set for each specific task through careful engineering (preprocessing, kernel....).**
- **Denial: kernel machines can approximate anything we want, and the VC-bounds guarantee generalization. Why would we need anything else?**
  - unfortunately, kernel machines with common kernels can only represent a tiny subset of functions efficiently
- **Optimism: Let's look for learning models that can be applied to the largest possible subset of the AI-set, while requiring the smallest amount of task-specific knowledge for each task.**
  - There is a parameterization of the AI-set with neurons.
  - Is there an efficient parameterization of the AI-set with computer technology?
- **Today, the ML community oscillates between defeatism and denial.**

Supervised Convolutional Nets learn well with lots of data

- **Supervised Convolutional nets work very well for:**
  - handwriting recognition(winner on MNIST)
  - face detection
  - object recognition with few classes and lots of training samples







Supervised Deep Learning works well with lots of samples

- **On recognition tasks with lots of training samples, deep supervised architecture outperform shallow architectures in speed and accuracy**

- **Handwriting Recognition: conv. nets 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]

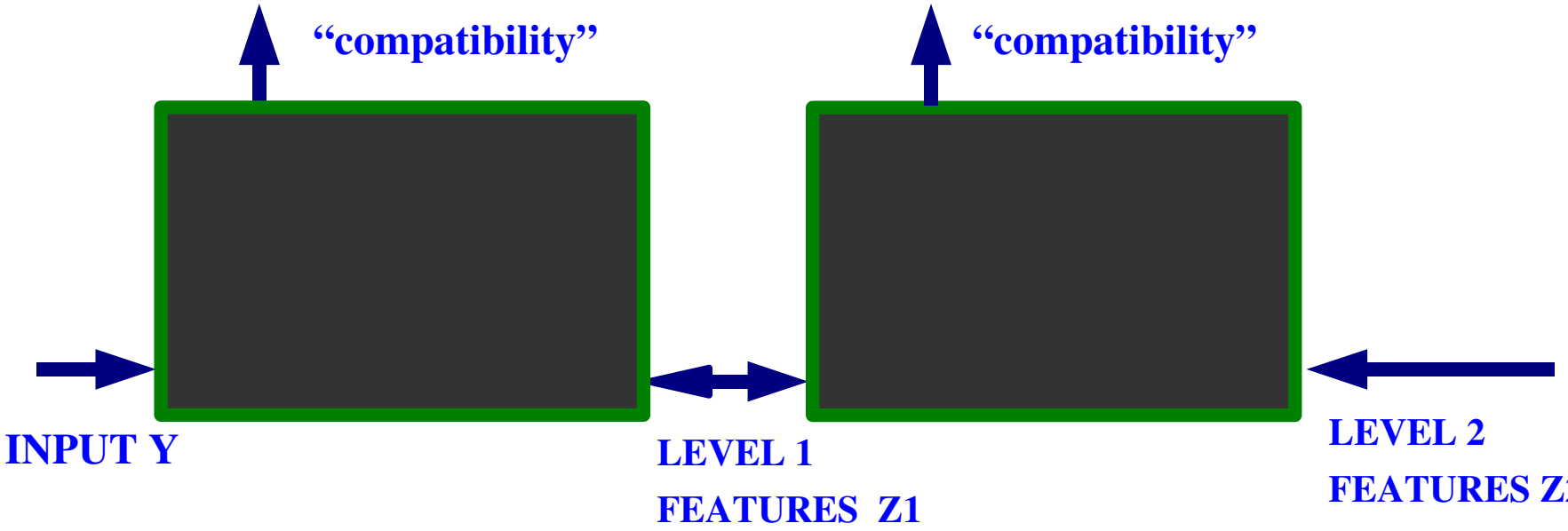
- **Face Detection: beats Viola-Jones**
  - [Vaillant 93,94][Garcia & Delakis PAMI 05][Osadchy JMLR 07]

Unsupervised Learning of Deep Feature Hierarchies

- **On recognition tasks with few labeled samples, deep supervised architectures don't do so well**
  - a purely supervised convolutional net gets only 20% correct on Caltech-101 with 30 training samples/class
- **We need unsupervised learning methods that can learn invariant feature hierarchies**
- **"Deep Belief Networks" [Hinton 2005]**
  - train each stage unsupervised one after the other.
  - if necessary, refine with backprop

The Basic Idea for Training Deep Feature Hierarchies

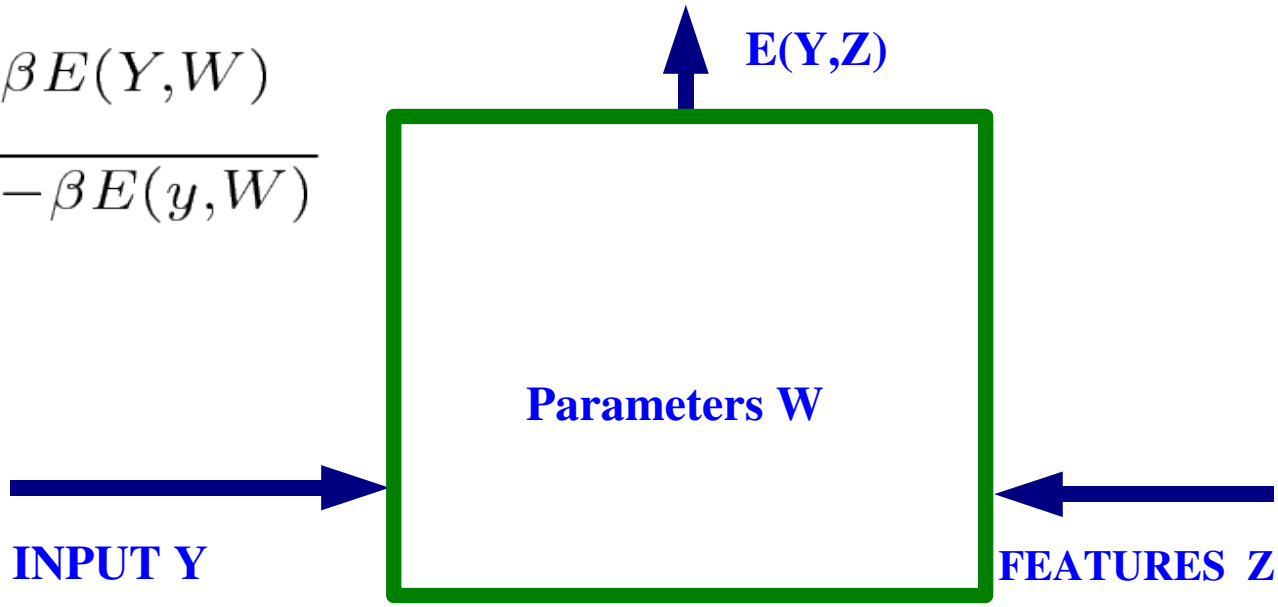
- **Stage k measures the "compatibity" between features at level k-1 (Z_{k-1}) and features at level k (Z_k).**
  - compatibility = -log likelihood = energy = $E(Z_{k-1},Z_k,W_k)$
- **Inference: Find the Z's that minimize the total energy.**
- **The stages are trained one after the other**
  - the input to stage k+1 is the feature vector of stage k.



Unsupervised Feature Learning as Density Estimation

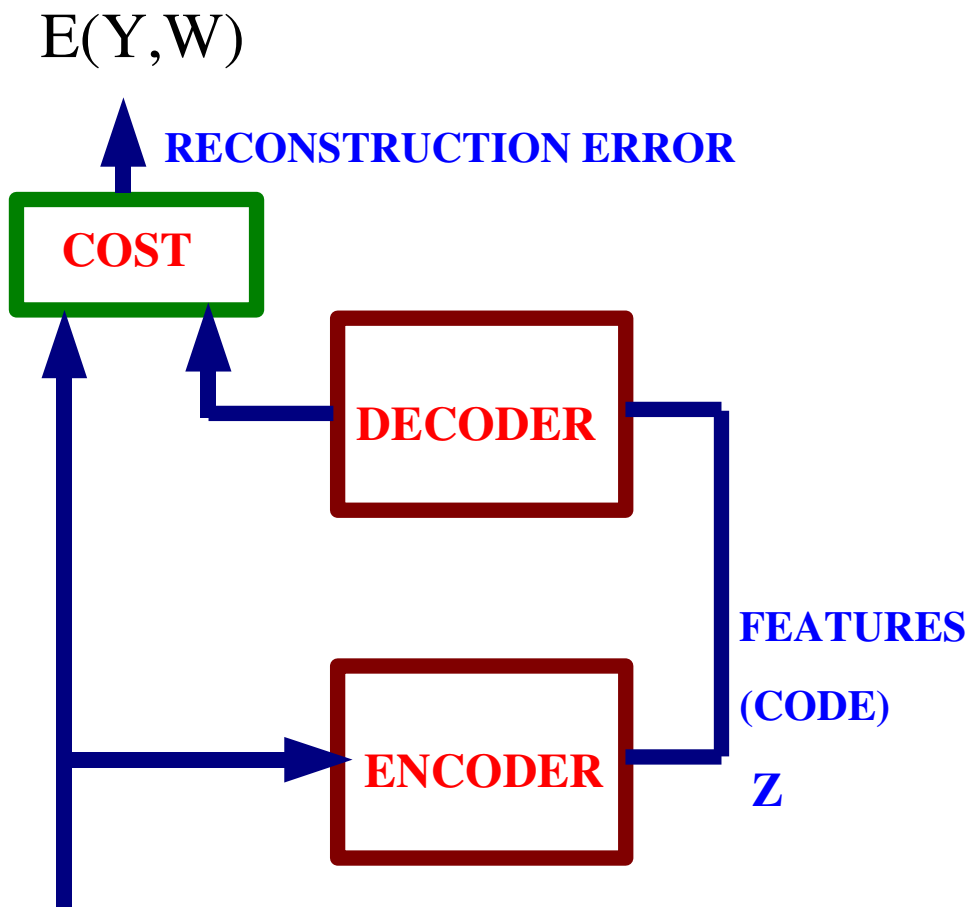
- **Energy function:**
  - E(Y,W) = MIN$_Z$ E(Y,Z,W) or E(Y,W) = -log SUM$_Z$ exp[-E(Y,Z,W)]
  - Y: input
  - Z: "feature" vector, representation, latent variables
  - W: parameters of the model (to be learned)
  - Maximum A Posteriori approximation or marginalization over Z
- **Density function P(Y|W)**
  - Learn W so as to maximize the likelihood of the training data

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



Unsupervised Feature Learning: Encoder/Decoder Architecture

- **Learns a probability density function of the training data**
- **Generates Features in the process**
- **The feature space is akin to an embedding of the manifold containing regions of high-density of data.**
- **Learning Algorithms:**
  - contrastive divergence
  - constraints on the information content of the features

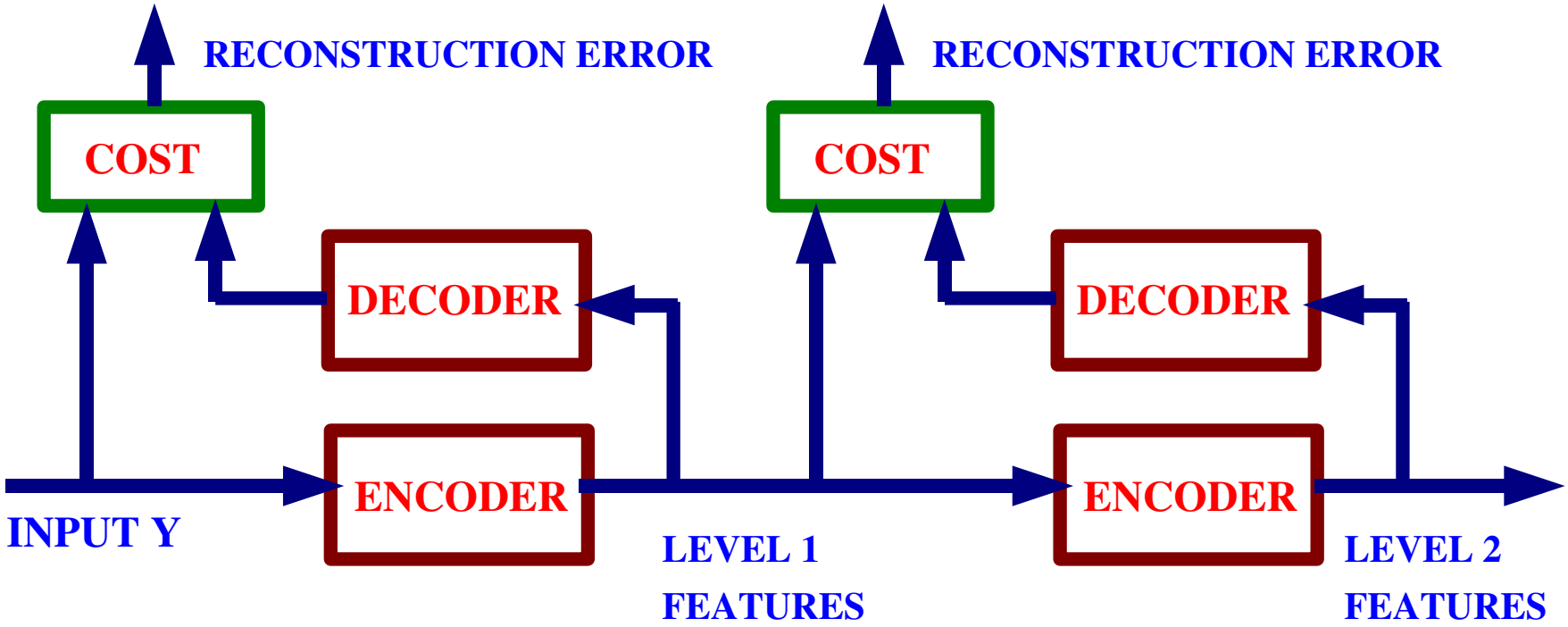


$$P(Y,W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

The Deep Encoder/Decoder Architecture

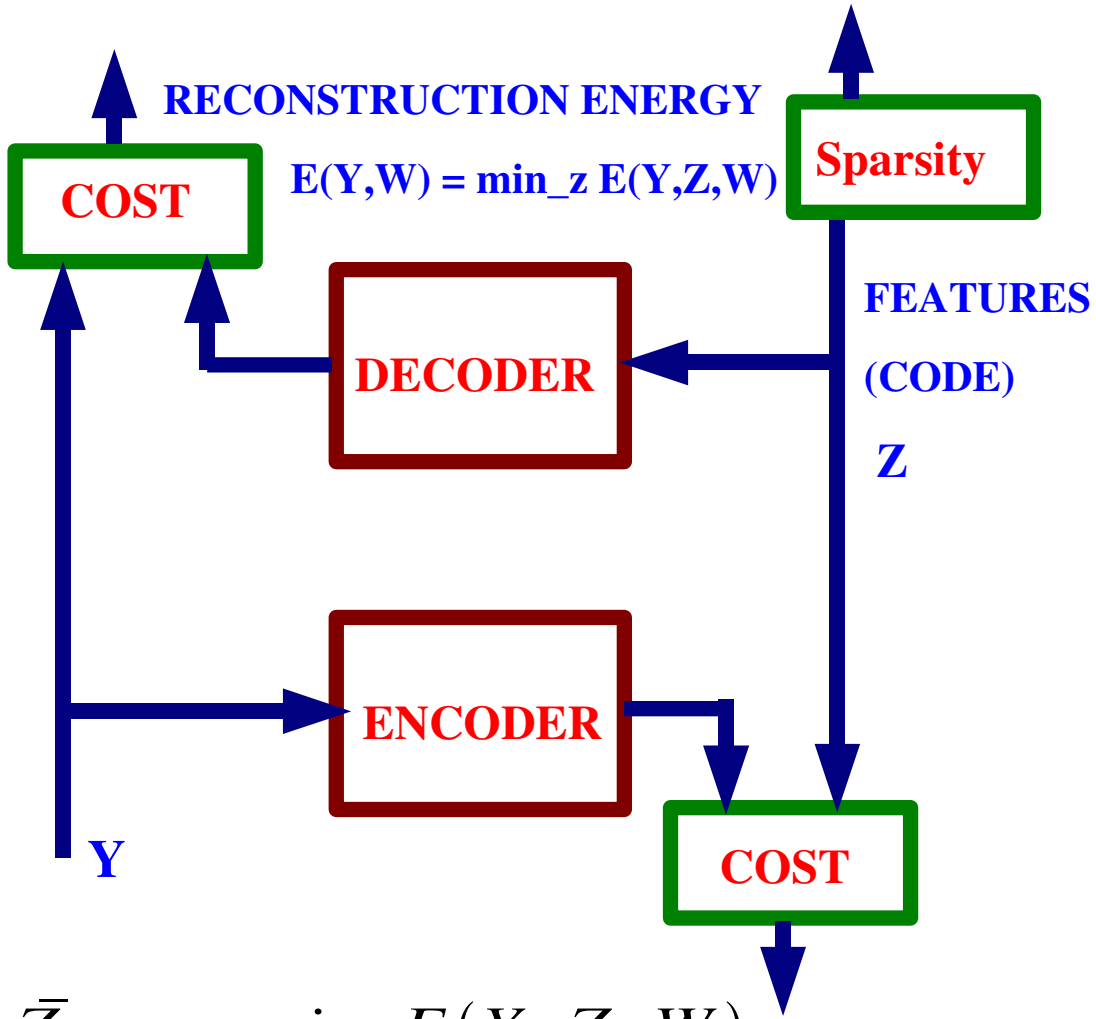
[Hinton 2005, Bengio 2006, LeCun 2006]

- **Each stage is composed of**
  - an encoder that produces a feature vector from the input
  - a decoder that reconstruct the input from the feature vector
    - (Restricted Boltzmann Machines are a special case)
- **Each stage is trained one after the other**
  - the input to stage k+1 is the feature vector of stage k.



General Encoder/Decoder Architecture

- **Decoder:**
  - Linear
- **Optional encoders of different types:**
  - None
  - Linear
  - Linear-Sigmoid-Scaling
  - Linear-Sigmoid-Linear
- **Optional sparsity penalty**
  - None, L1, Log Student-T
- **Feature Vector Z**
  - continuous
  - binary stochastic
  - discrete (e.g. 1-of-N)



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

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

Non-Linear Dimensionality Reduction with DBNs

- [Hinton and Salakhutdinov, Science 2006]

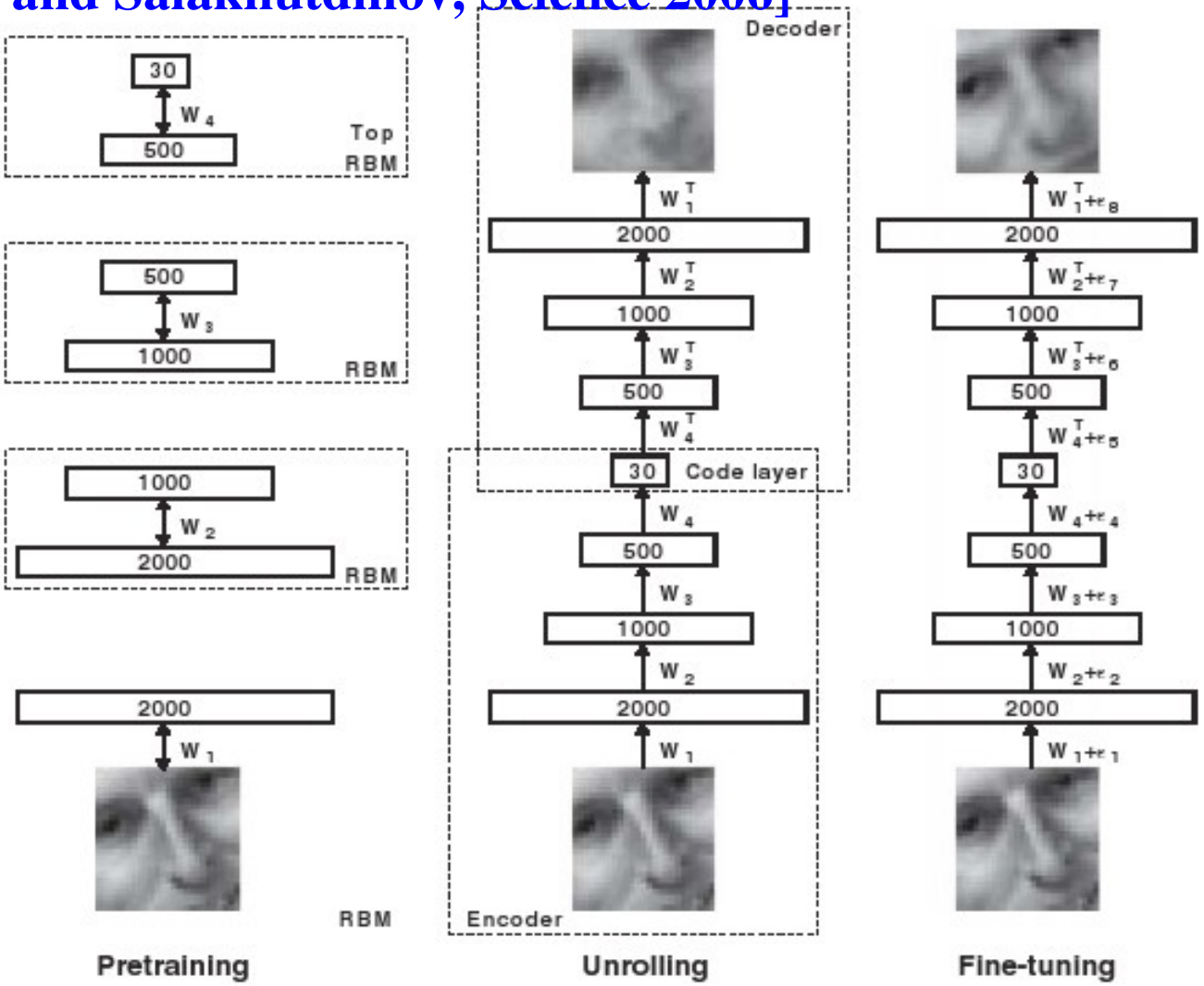


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create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

Non-Linear Dimensionality Reduction

- [Hinton and Salakhutdinov, Science 2006]

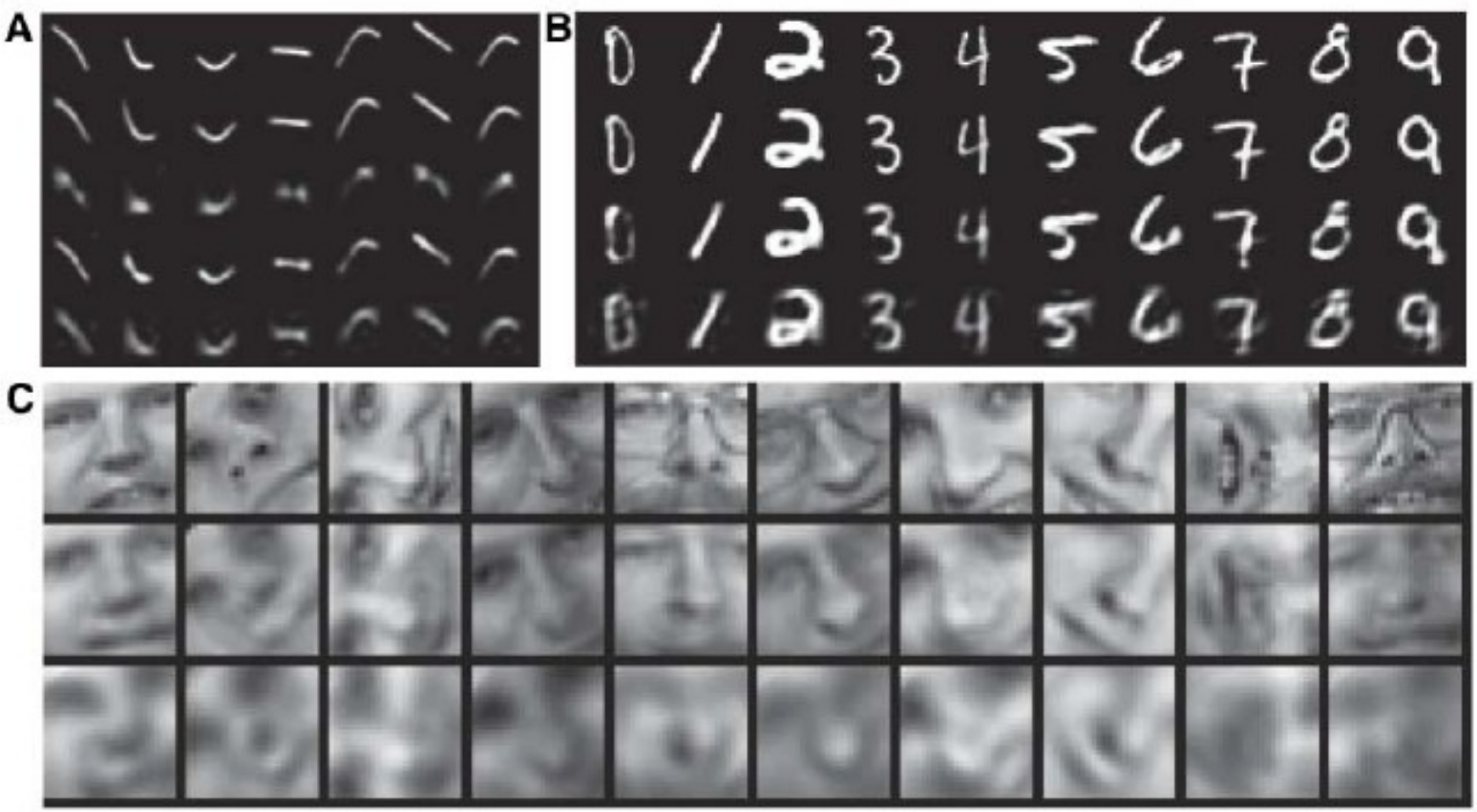
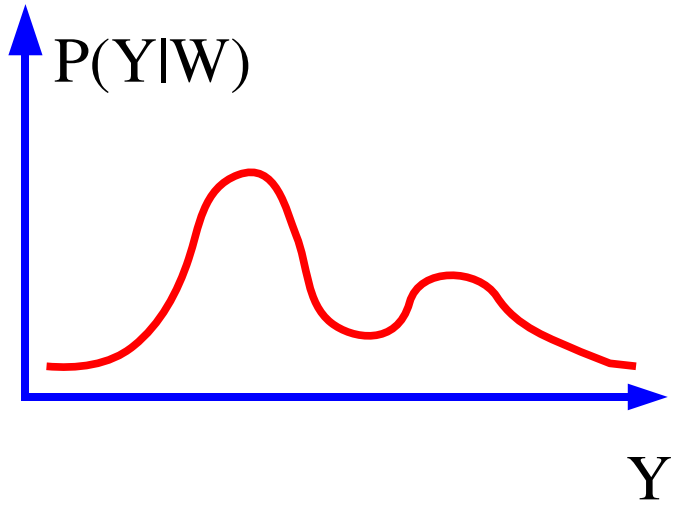


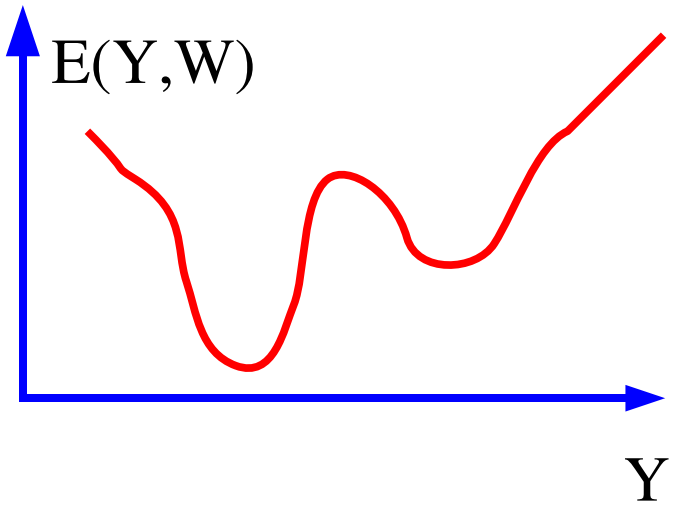
Fig. 2. (**A**) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (*8*) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error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

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).

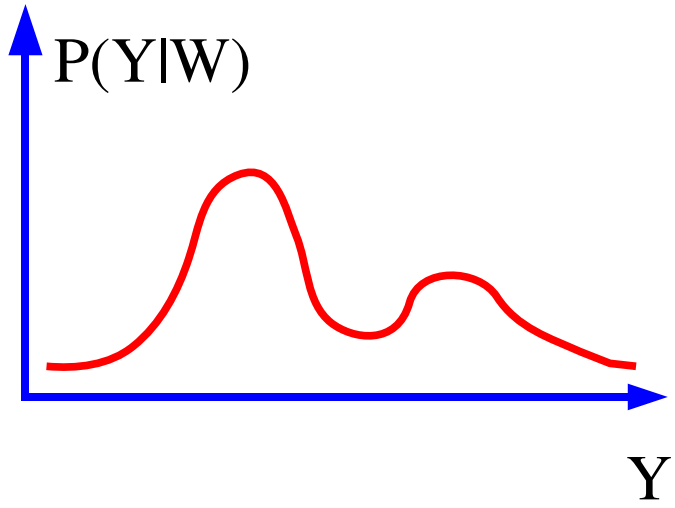


- **Energy-Based View:**
  - produce an energy function E(Y,W) that:
  - has low value in regions of high sample density
  - has high(er) value everywhere else

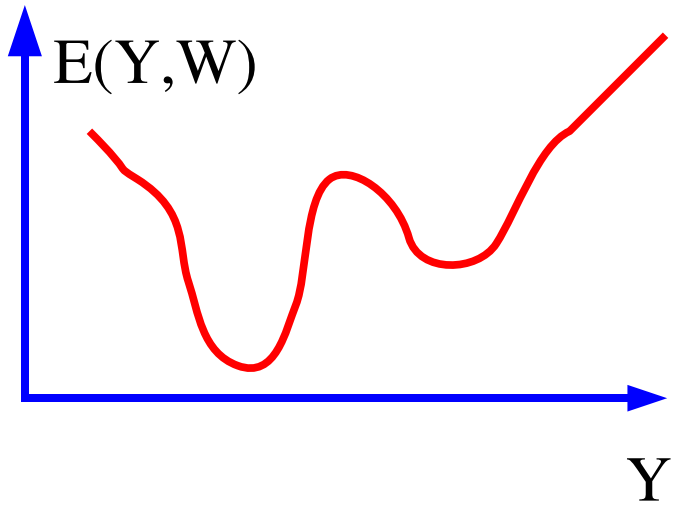


Energy <-> Probability

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



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



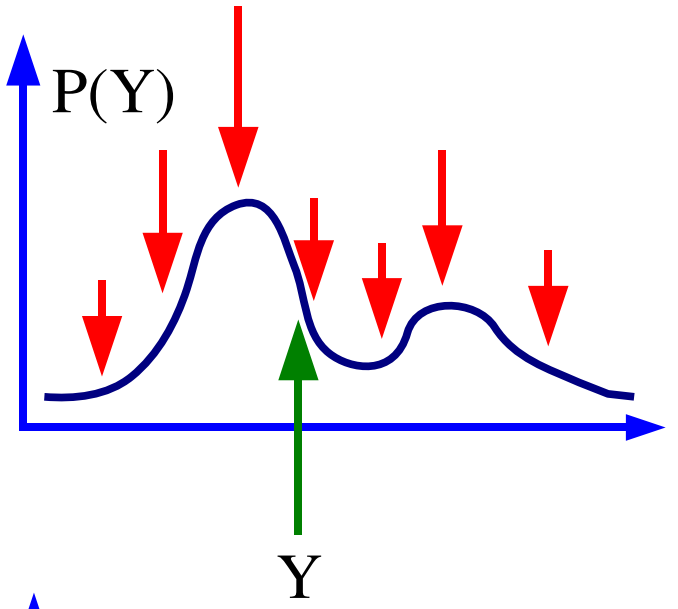
Training an Energy-Based Model to Approximate a Density

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

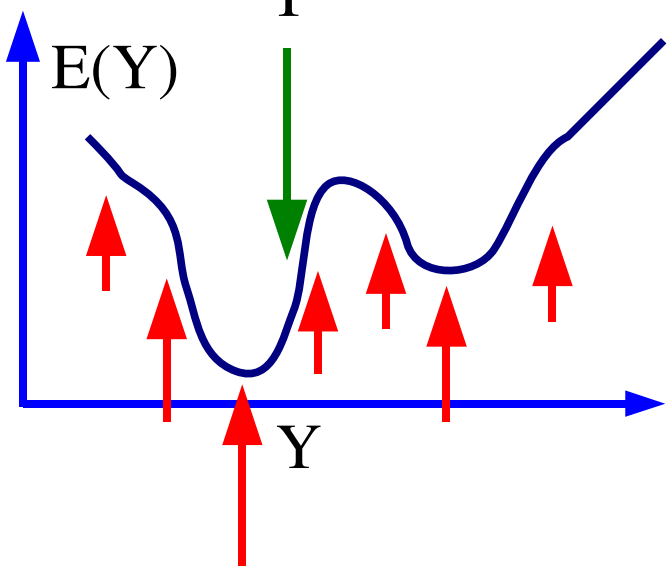


Minimizing -log P(Y,W) on training samples

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

make this small

make this big



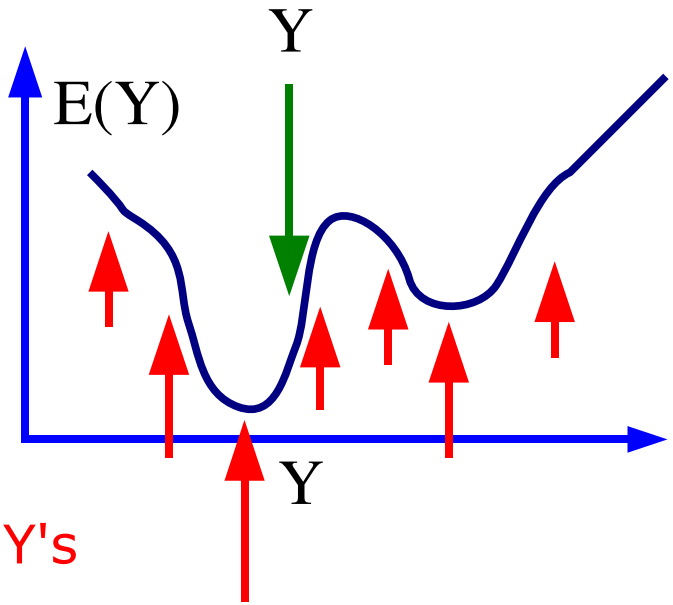
Training an Energy-Based Model with Gradient Descent

- **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

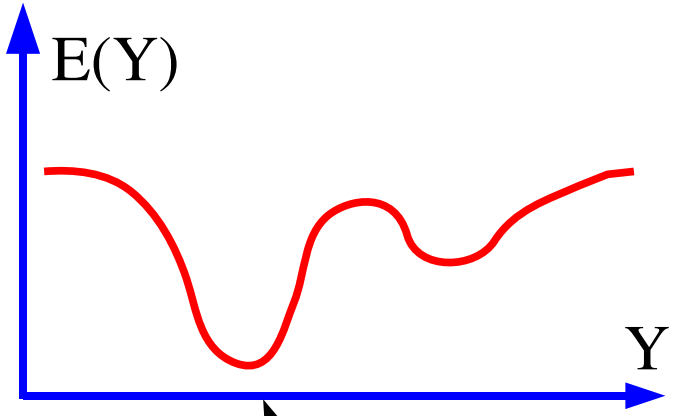
$$W \leftarrow W - \eta \frac{\partial E(Y,W)}{\partial W} + \eta \int_y P(y|W) \frac{\partial E(y,W)}{\partial W}$$

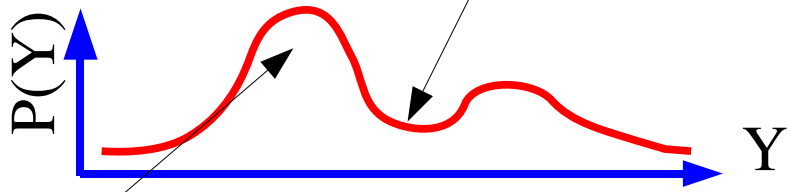
The Normalization problem

- **The "Intractable Partition Function Problem" a.k.a. the Normalization Problem**
  - Give high probability (or low energy) to training samples
  - Give low probability (or high energy) to everything else
  - There are too many "everything else"!
  - The normalization constant of probabilistic models is a sum over too many terms.
  - **Making the energies of everthing else large is very hard**

The Intractable Partition Function Problem

- **Example: Image Denoising**
- **Learning:**
  - push down on the energy of training samples
  - push up on the energy of everything else



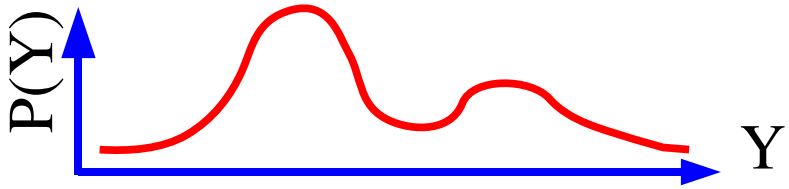


$$P(Y,W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

The Two Biggest Challenges in Machine Learning

1. The "Intractable Partition Function Problem"

- Complicated probability distributions in high dimensional spaces are difficult to normalize



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

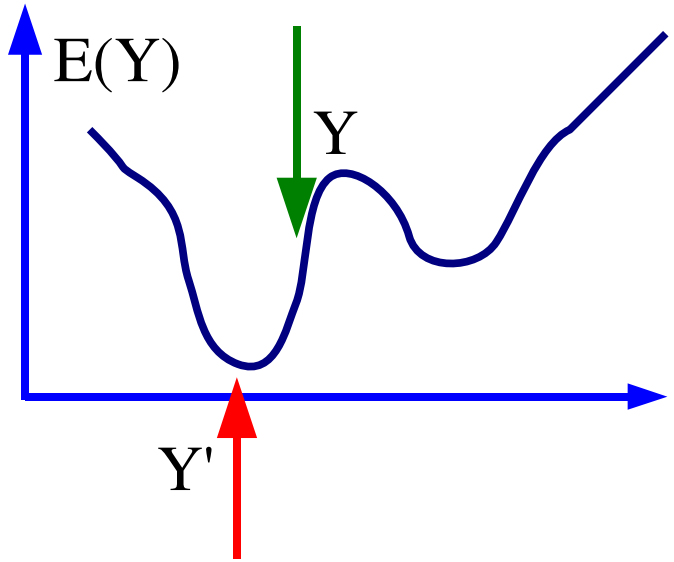
- Example: what is the PDF of natural images?
- Question: how do we get around this problem?

2. The "Deep Learning Problem"

- Complex tasks (vision, audition, natural language understanding....) require appropriate internal representations.
- With most current approaches to learning, the internal representation (and the similarity metric on it) are assumed to be given (or hand-engineered).
- Question: how do we learning internal representations?

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



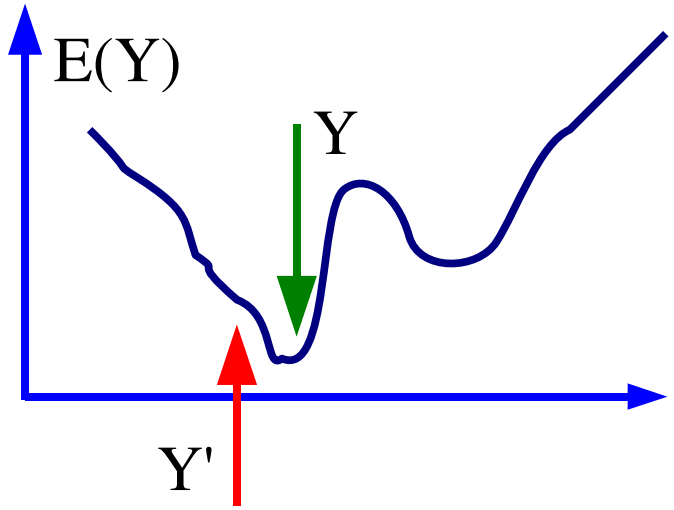
$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \frac{\partial E(Y', W)}{\partial W}$$

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

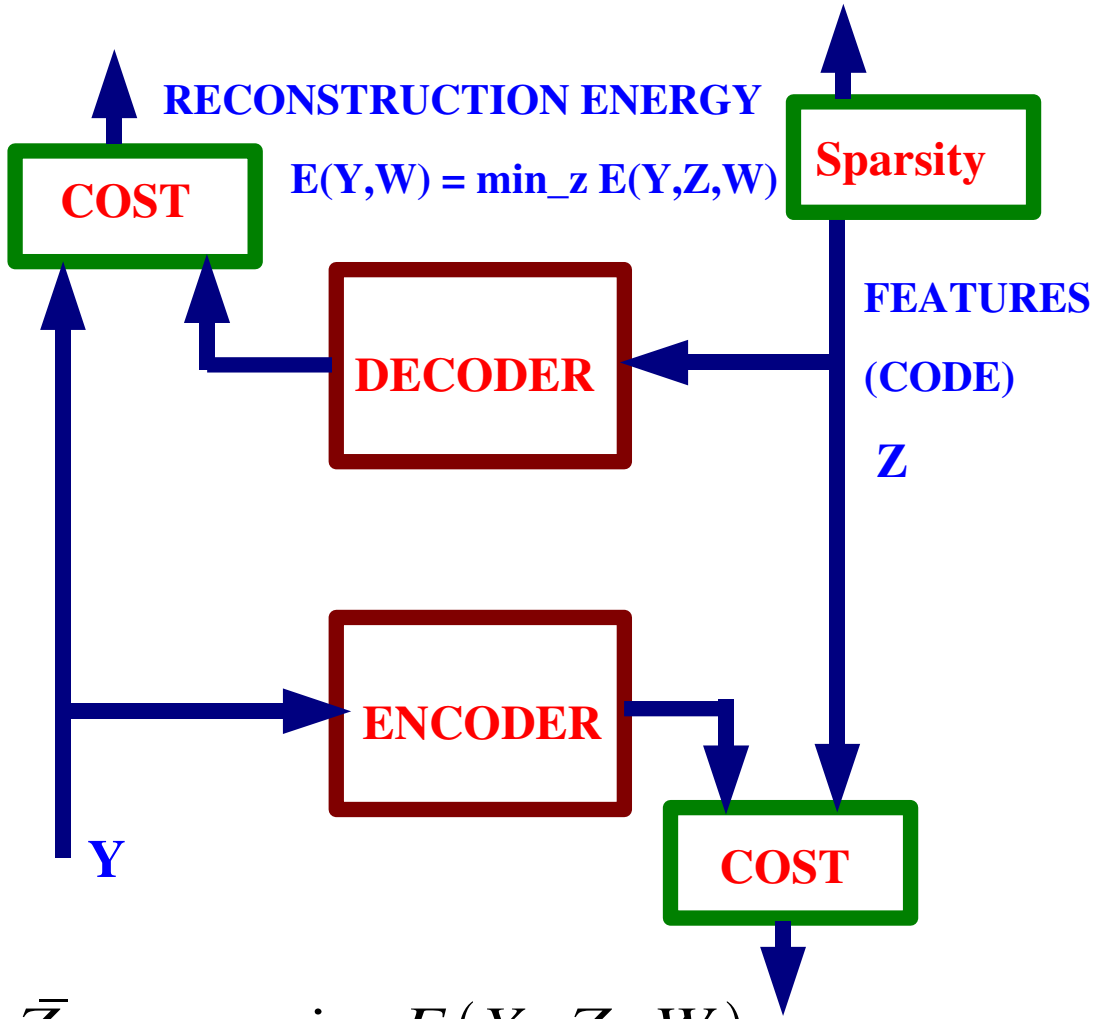
Pulls up on the energy Y'

Problems with Contrastive Divergence in High Dimension

- Contrastive Divergence is a "boutique" way of making the energy surface non-flat
- If the energy surface is highly flexible and high-dimensional, there are simply too many points whose energy needs to be pulled up.
- It becomes very difficult to make it non-flat.
- We need a more "wholesale" way of making the energy surface non-flat.
- The main idea of this talk: restricting the information content of the code makes the energy surface non-flat.
- Information restriction through sparsification

General Encoder/Decoder Architecture with a sparsity term

- **Decoder:**
  - Linear
- **Optional encoders of different types:**
  - None
  - Linear
  - Linear-Sigmoid-Scaling
  - Linear-Sigmoid-Linear
- **Optional sparsity penalty**
  - None, L1, Log Student-T
- **Feature Vector Z**
  - continuous
  - binary stochastic
  - discrete (e.g. 1-of-N)

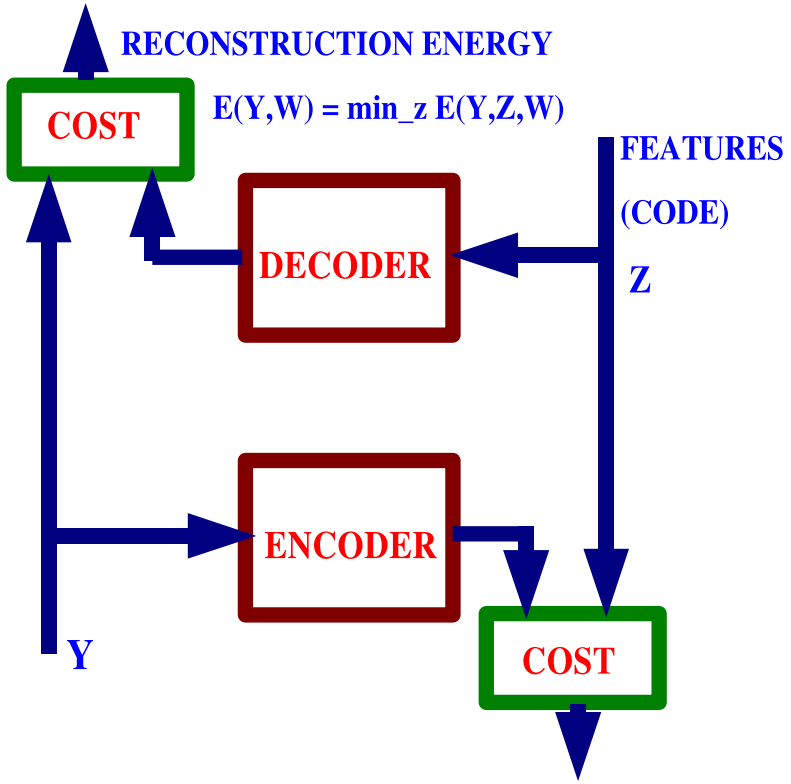


$$\bar{Z}_Y = argmin_Z E(Y, Z, W)$$

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

The Main New Idea in This Talk!

- **Contrastive divergence doesn't work too well when the dimension of the input is very large (> a few hundred)**
  - because the space of "everything else" is too large
- **We need a more efficient way to ensure that the energy surface takes the right shape**
  - with a groove around the manifold containing the training samples
- **Main Idea: Restrict the information content in the feature vector Z**
  - **by making it sparse**
  - by making it low dimensional
  - by making it binary
  - by making it noisy



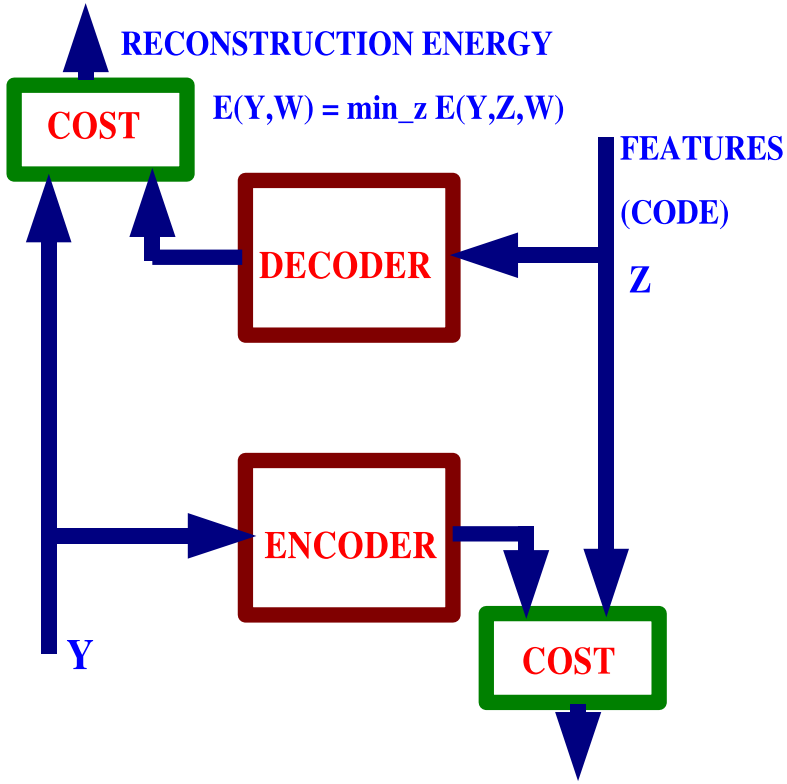
$$\bar{Z}_Y = argmin_Z E(Y, Z, W)$$

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

[Ranzato et al. AI-Stats 2007]

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



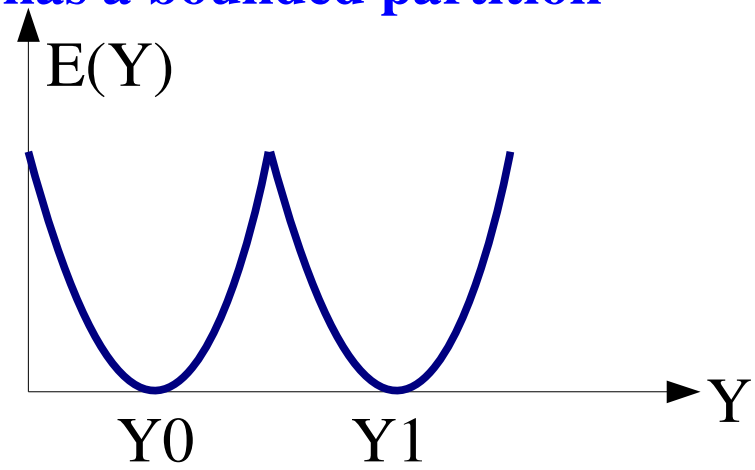
$$\bar{Z}_Y = argmin_Z E(Y, Z, W)$$

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

Why sparsity puts an upper bound on the partition function

- **Imagine the code has no restriction on it**
  - The energy (or reconstruction error) can be zero everywhere, because every Y can be perfectly reconstructed. The energy is flat, and the partition function is unbounded
- **Now imagine that the code is binary (Z=0 or Z=1), and that the reconstruction cost is quadratic E(Y) = ||Y-Dec(Z)||^2**
  - Only two input vectors can be perfectly reconstructed:
  - Y0=Dec(0) and Y1=Dec(1).
  - All other vectors have a higher reconstruction error
- **The corresponding probabilistic model has a bounded partition function:**

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

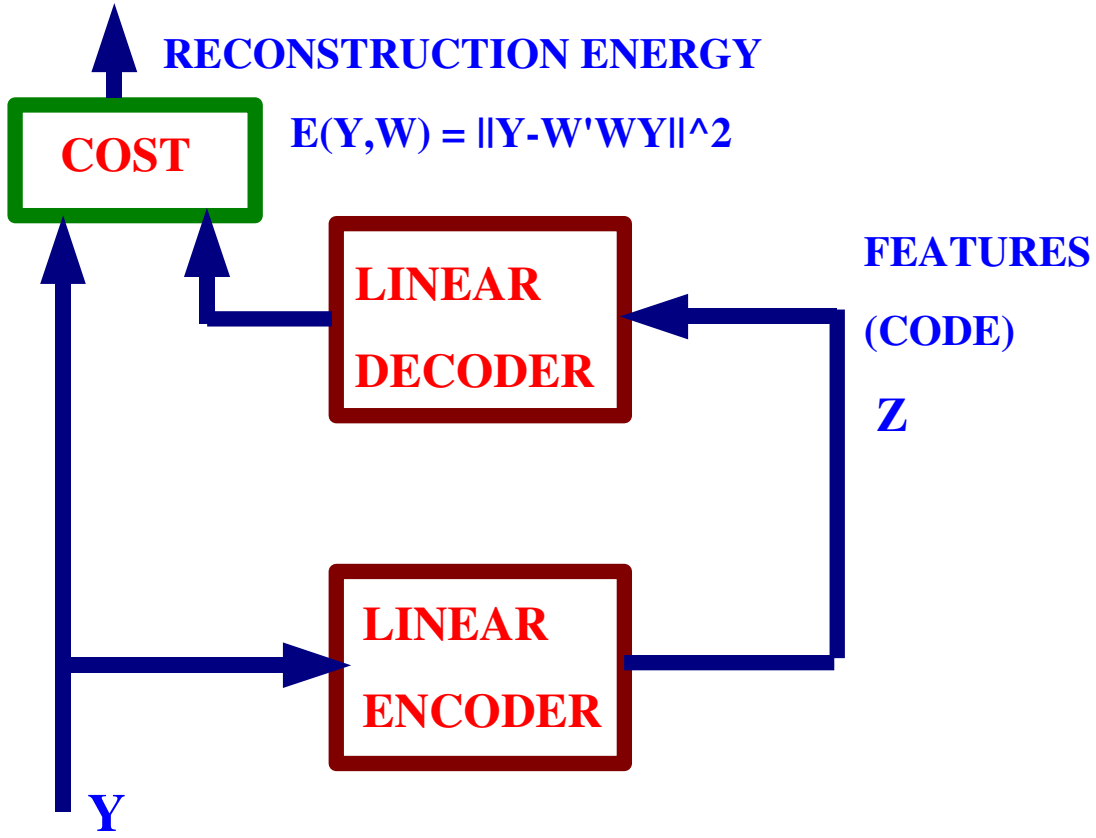


Restricting the Information Content of the Code

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

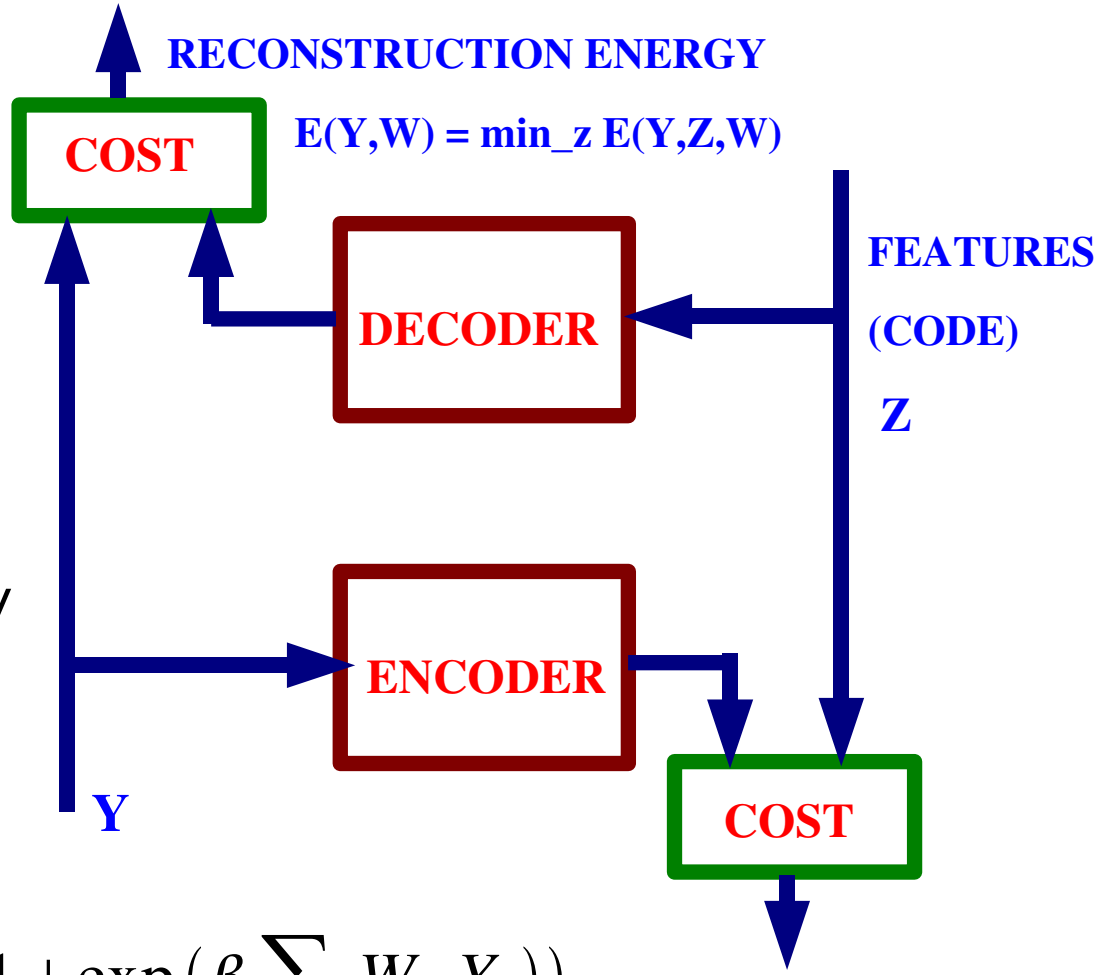
Encoder/Decoder Architecture: PCA

- **PCA:**
  - linear encoder and decoder
  - no sparsity
  - low-dimensional code Z
  - $E(Y) = ||Y-W'WY||^2$



Encoder/Decoder Architecture: RBM

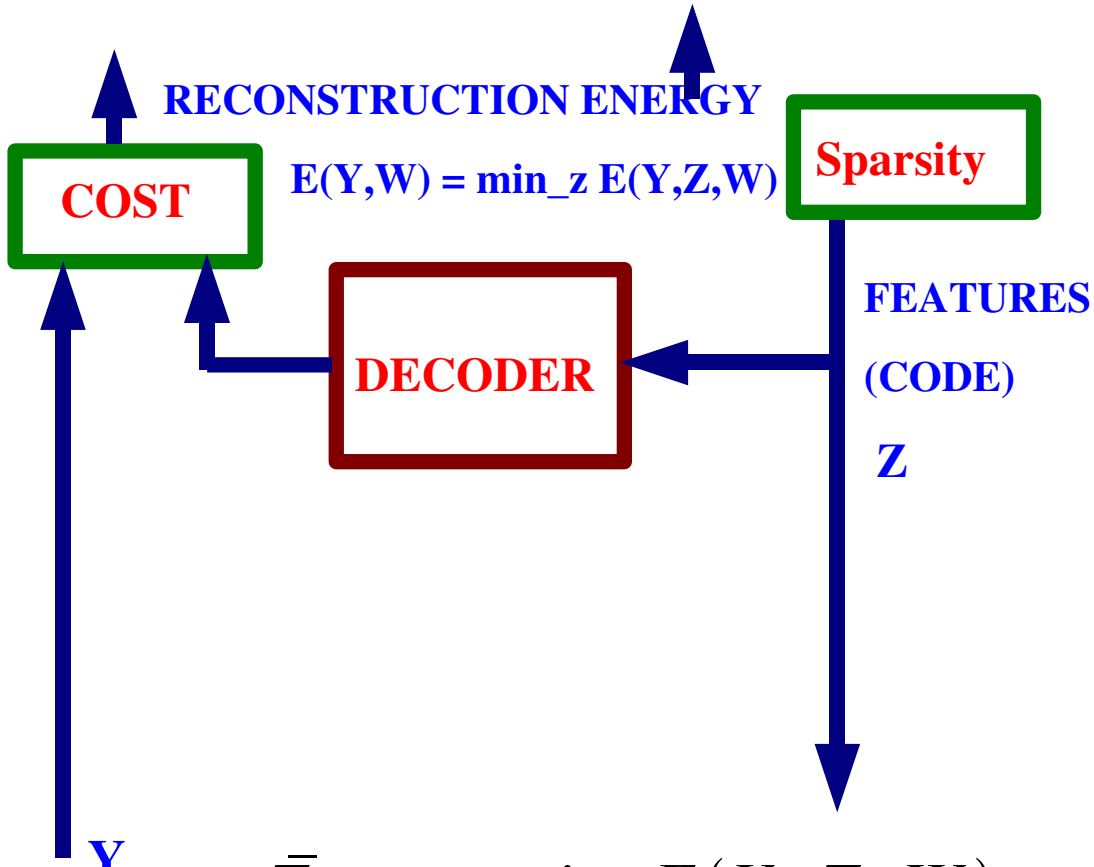
- **Restricted Boltzmann Machines:**
  - [Hinton et al. 2005]
  - symmetric encoder/decoder
  - E(Y,Z,W)=-Y'WZ
  - Z: binary stochastic vector
  - Learning: contrastive Divergence
  - It seems that the energy surface becomes non flat because Z is binary and noisy (not just because of contrastive divergence).
- **Sampling is expensive**



$$P(Z_j=1/Y,W)=1/(1+\exp(\beta\sum_i W_{ij}Y_i))$$
$$P(Y_i=1/Z,W)=1/(1+\exp(\beta\sum_j W_{ij}Z_j))$$
$$E(Y,W)=-1/\beta\log\sum_Z \exp(-\beta E(Y,Z,W))$$

Unsupervised Feature Learning: Linear Decoder Architecture

- **K-Means:**
  - no encoder, linear decoder
  - Z is a one-of-N binary code
  - $E(Y) = ||Y-ZW||^2$
- **Sparse Overcomplete Bases:**
  - [Olshausen & Field]
  - no encoder
  - linear decoder
  - log Student-T sparsity
- **Learned Basis Pursuit**
  - [Chen & Donoho]
  - no encoder
  - linear decoder
  - L1 sparsity



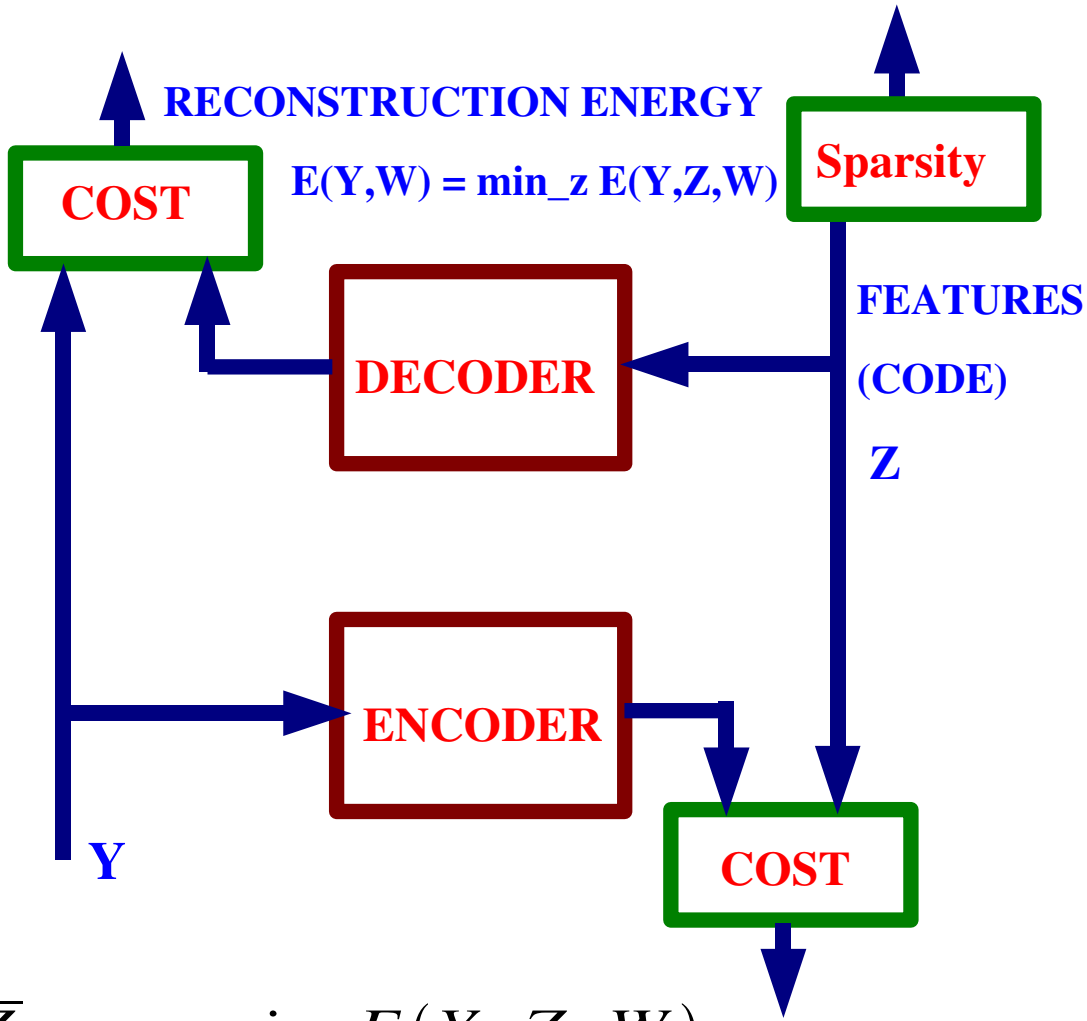
$$\bar{Z}_Y = argmin_Z E(Y, Z, W)$$

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

- **Problem: computing Z from Y involves running a minimization algorithm**

Sparse Features with "Predictable Basis Pursuit"

- Linear Decoder with normalized basis functions
- L1 Sparsity penalty
- Encoder of different types
  - Linear
  - Linear-Sigmoid-Scaling
  - Linear-Sigmoid-Linear
- The decoder+sparsity is identical to Chen & Donoho's basis pursuit
- But the encoder learns to "predicts" the optimal feature codes

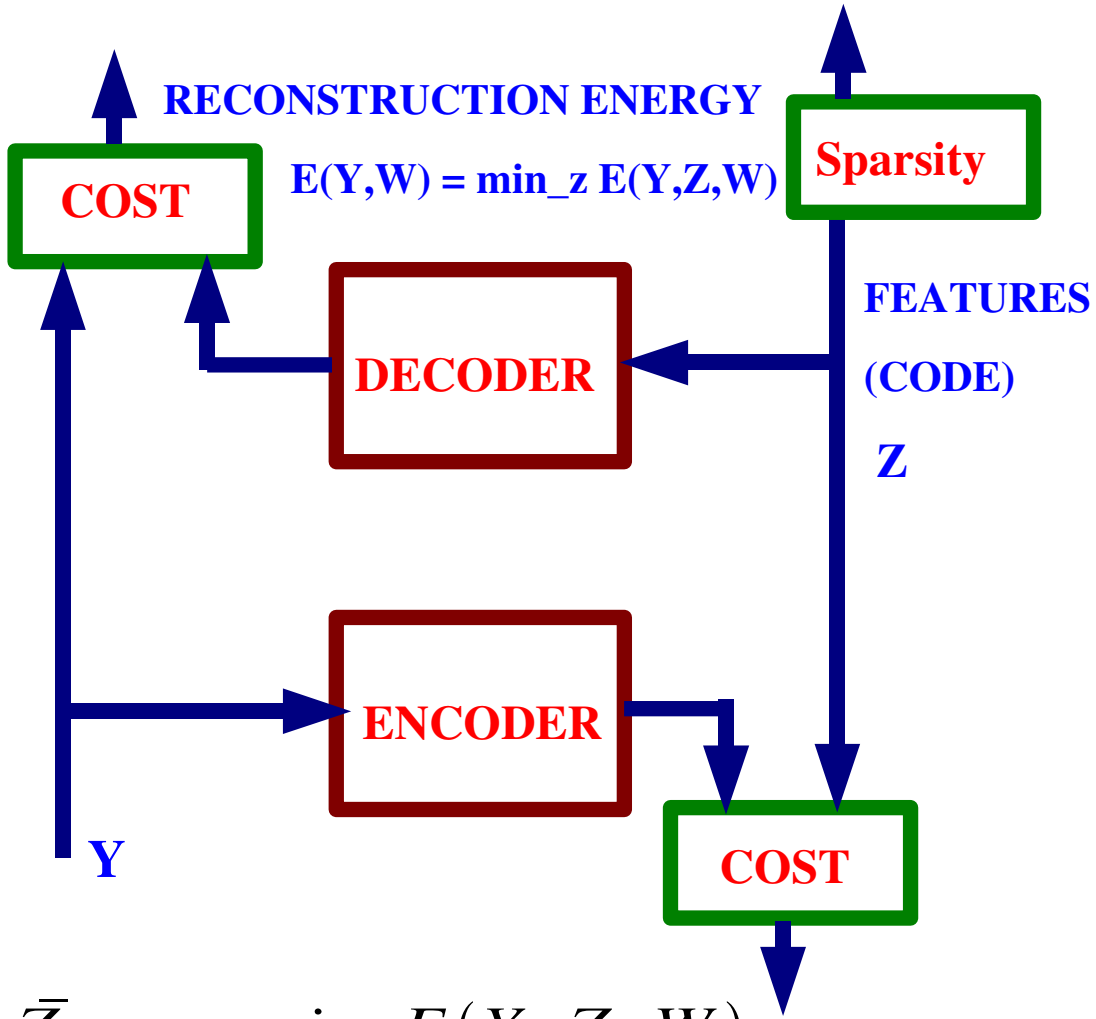


$$\bar{Z}_Y = argmin_Z E(Y, Z, W)$$

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

Encoder/Decoder: Predictable Basis Pursuit

- **Decoder:**
  - Linear $\|y - \Phi z\|_2^2 + \alpha_z \|z\|_1$
- **Encoders of different types:**
  - None
  - Linear
  - Linear-Sigmoid-Scaling
  - Linear-Sigmoid-Linear
- **Sparsity penalty**
  - L1
- **Main Idea:**
  - find basis functions such that the coefficients that reconstruct any vector can be predicted by the encoder.

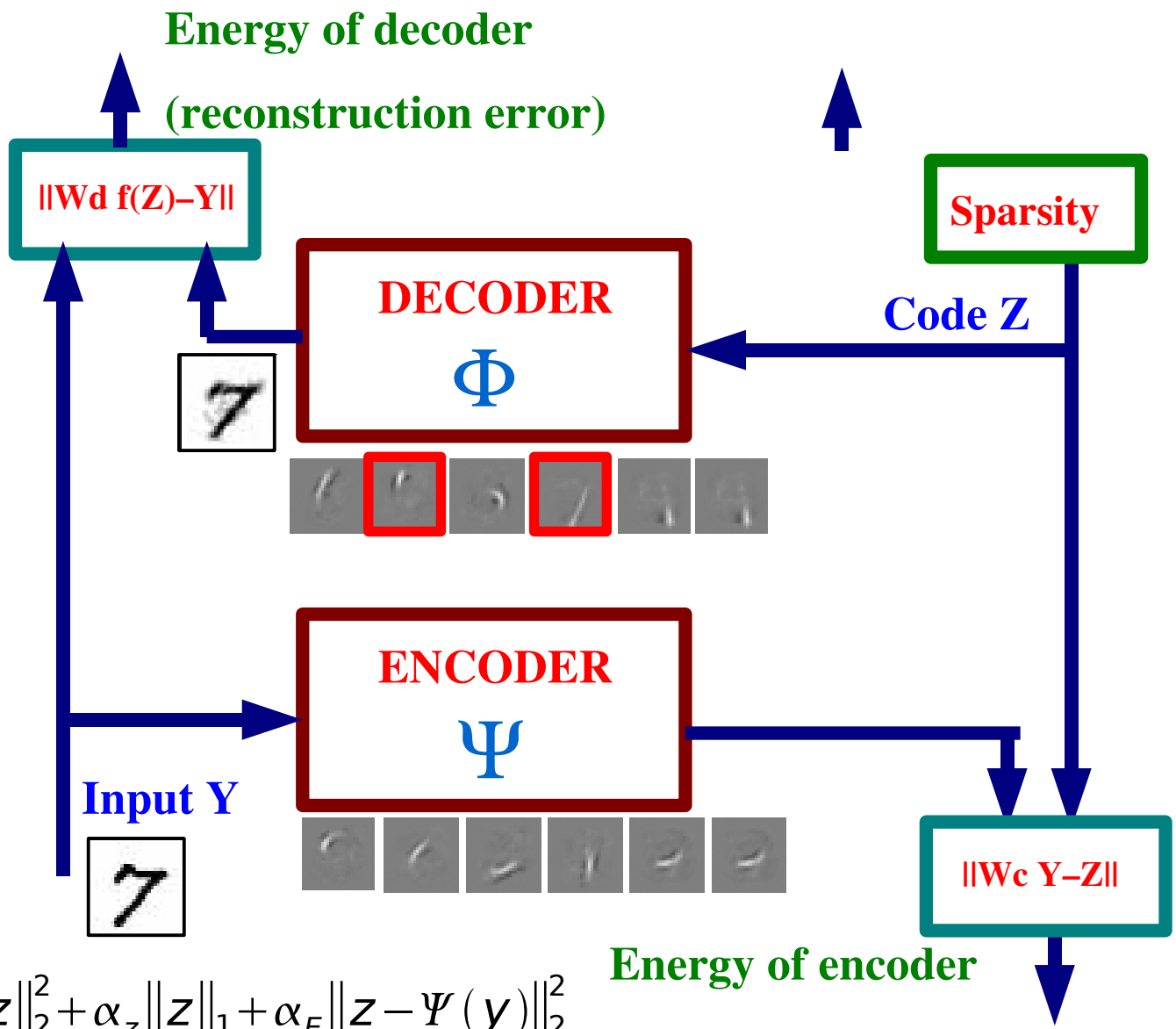


$$\bar{Z}_Y = argmin_Z E(Y, Z, W)$$

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

Training The Predictable Basis Pursuit Model

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

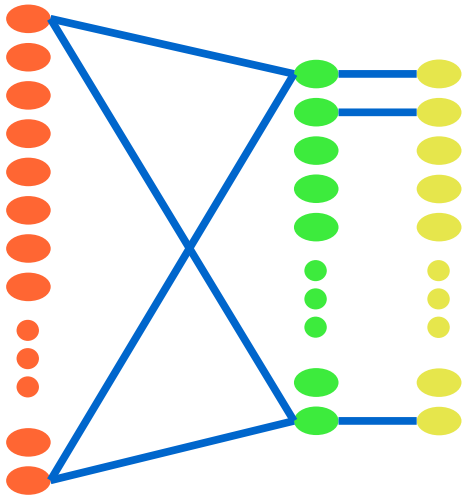


$$L(y, z, \Phi, \Psi) = \|y - \Phi z\|_2^2 + \alpha_z \|z\|_1 + \alpha_E \|z - \Psi(y)\|_2^2$$

Encoder Architectures

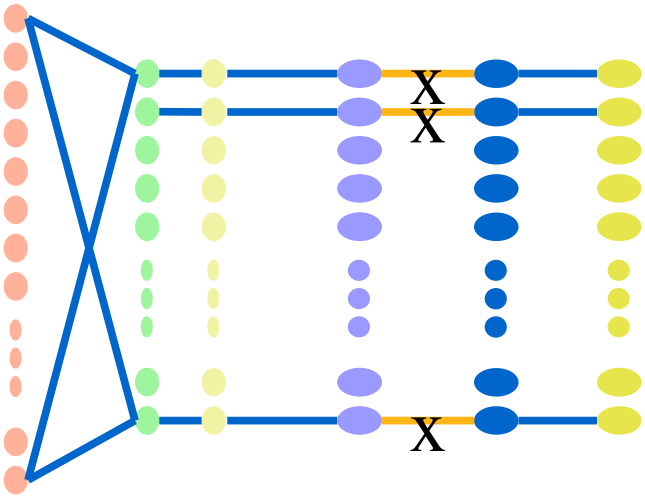
- L: Linear
- FD: Linear + Sigmoid + Gain + Bias
- FL: Linear + Sigmoid + Linear

Linear (L)



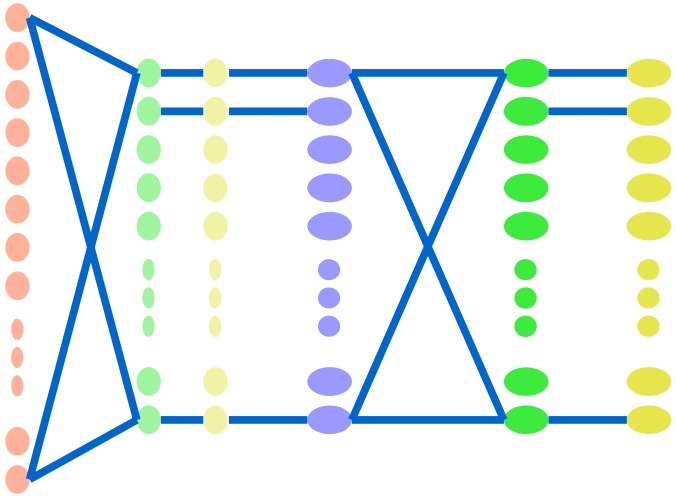
$$Z' = MY + b$$

Non-Linear – Individual gains (FD)



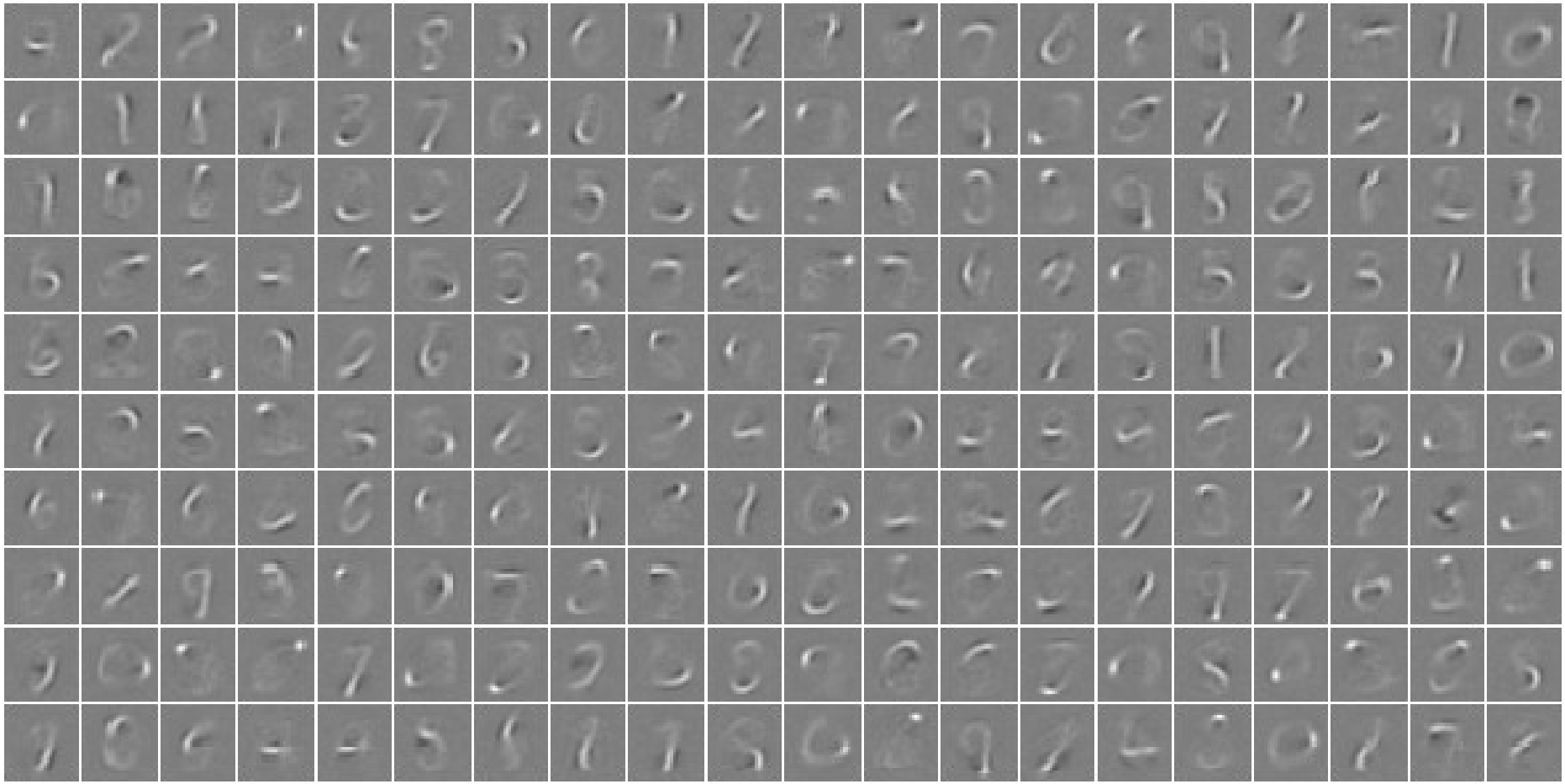
$$Z' = \sigma(MY + b_1) \times diag(g) + b_2$$

Non-Linear – 2 Layer NN (FL)



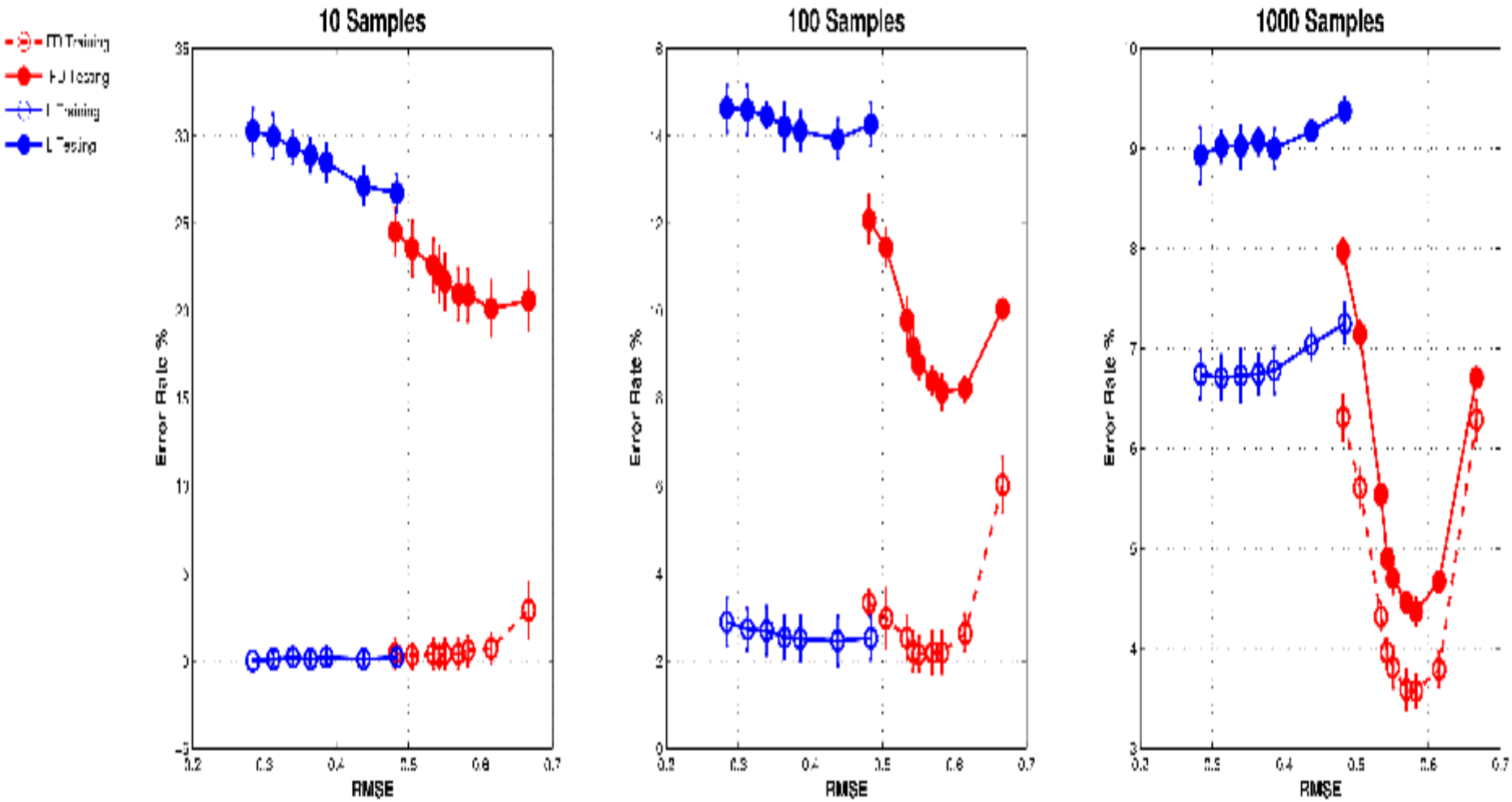
$$Z' = M_2 \sigma(M_1 Y + b_1) + b_2$$

Decoder Basis Functions on MNIST



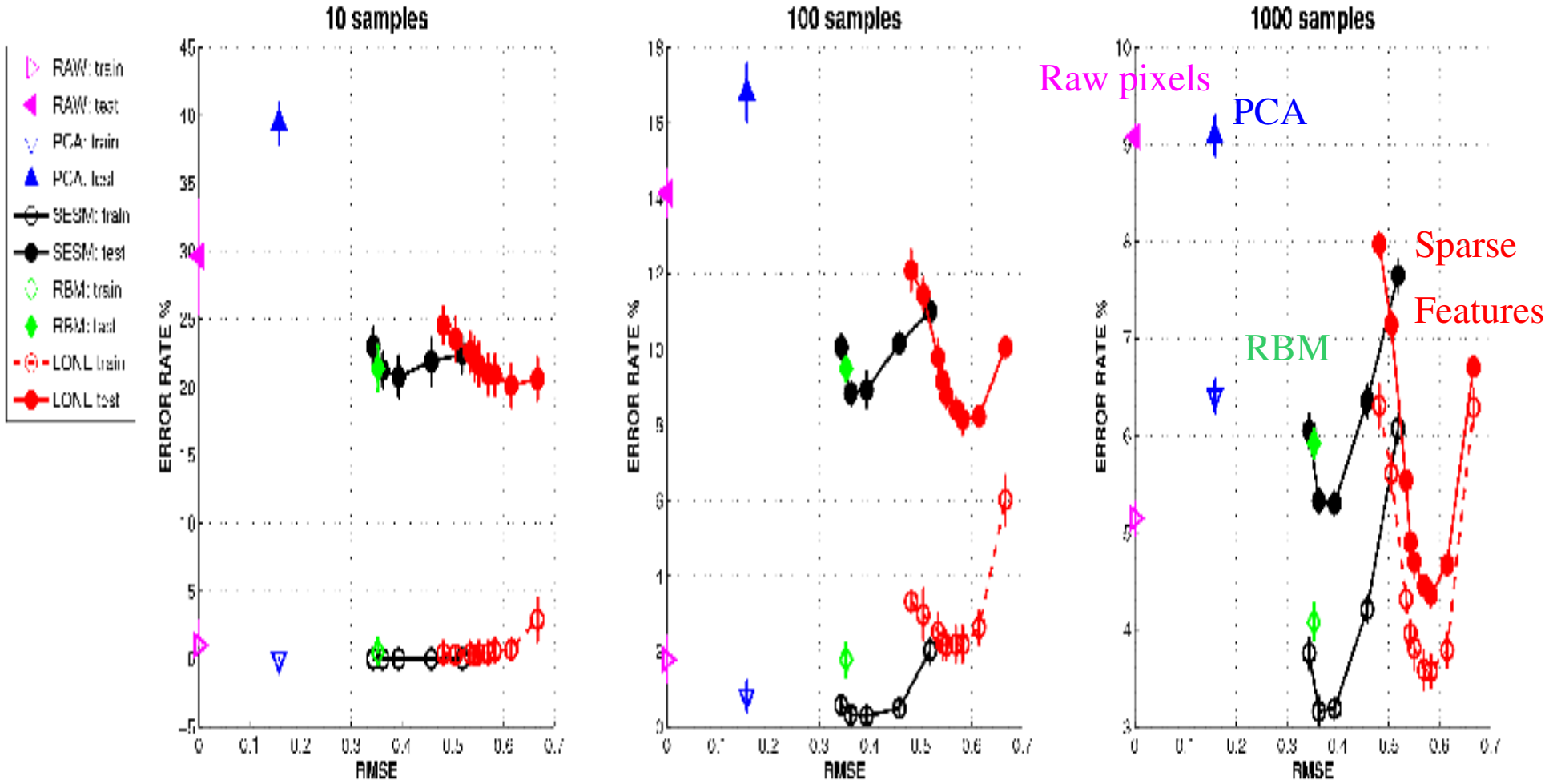
Classification Error Rate on MNIST

- Supervised Linear Classifier trained on 200 trained sparse features

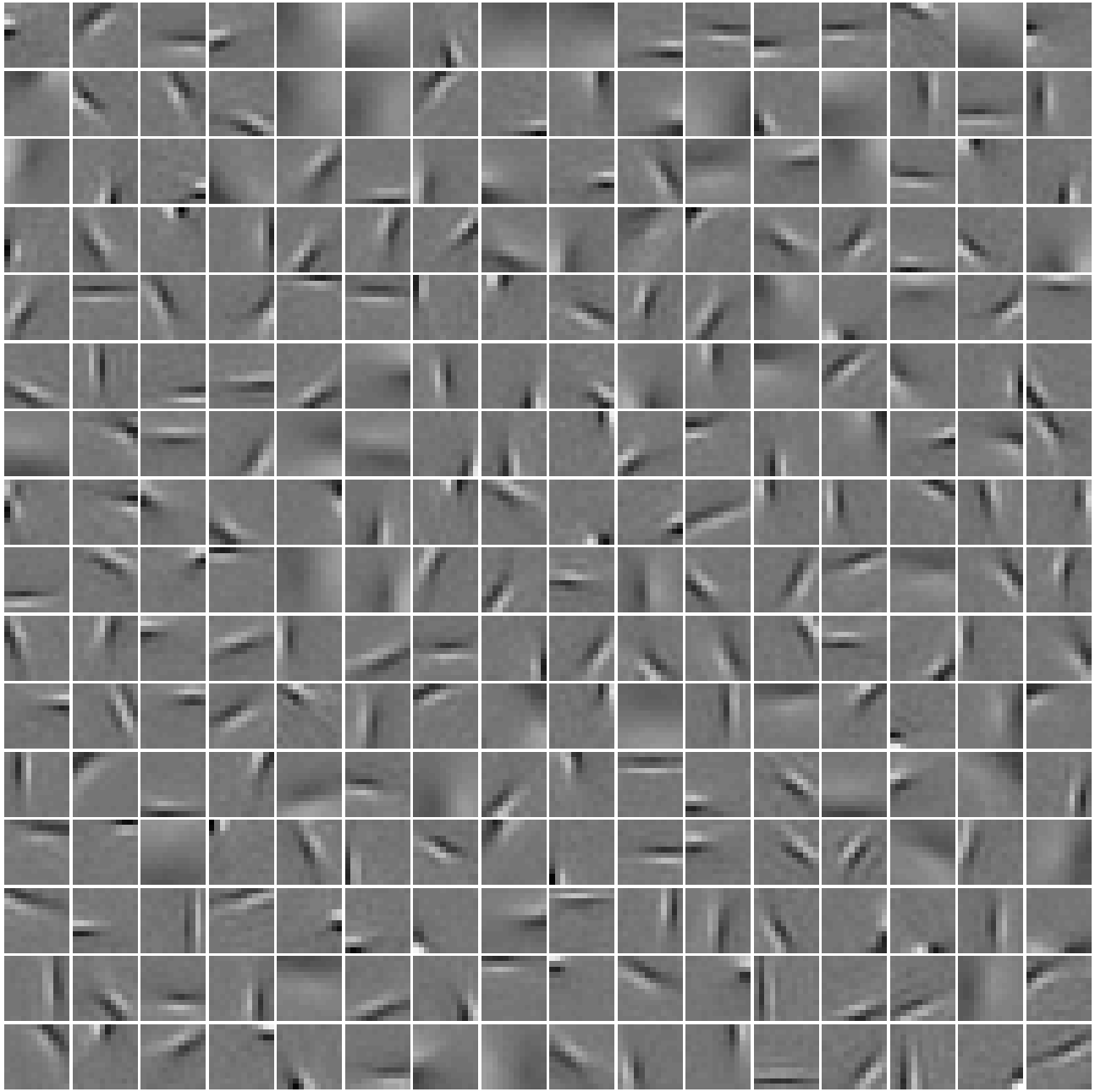


Classification Error Rate on MNIST

- **Supervised Linear Classifier trained on 200 trained sparse features**

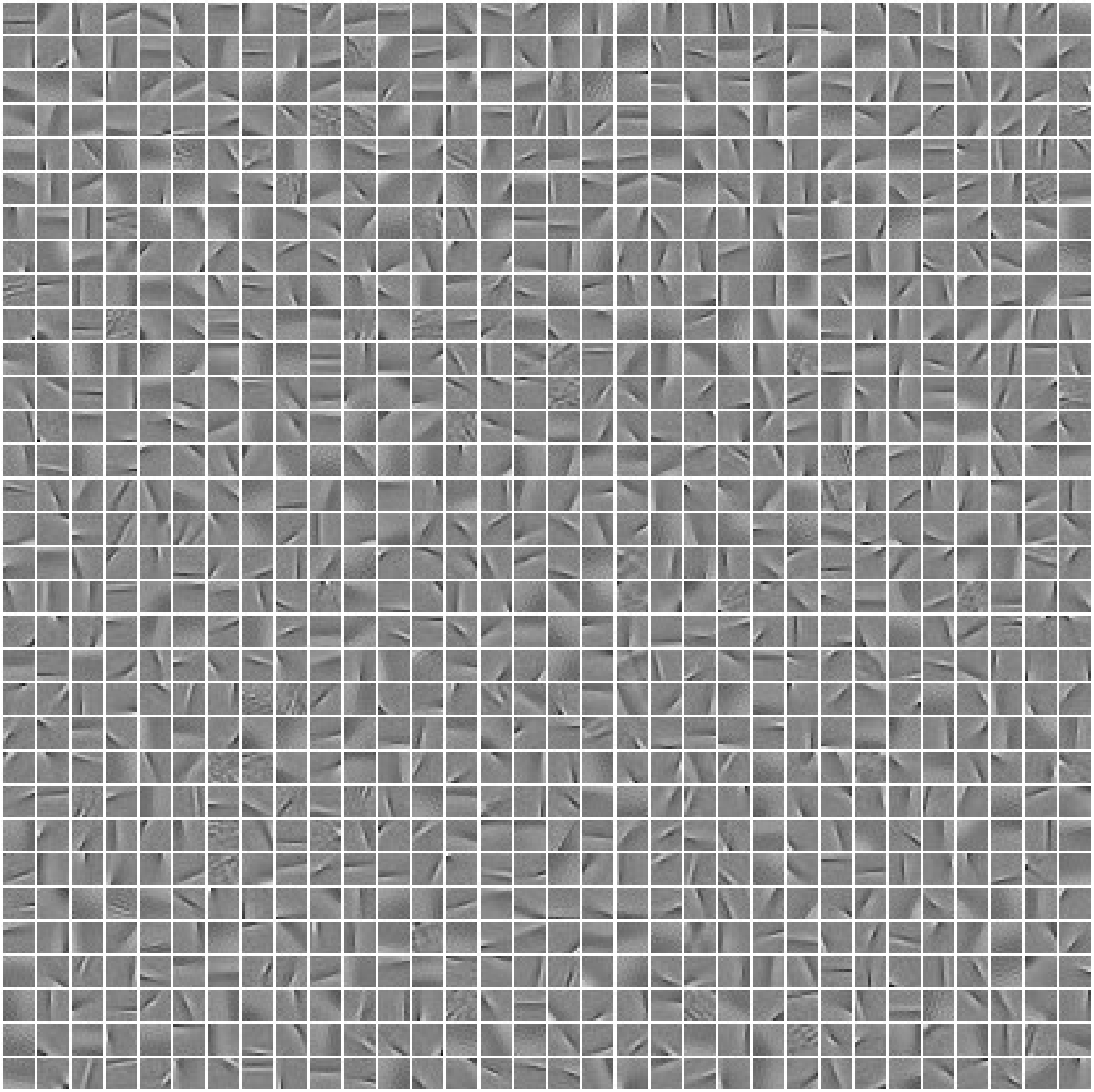


Learned Features on natural patches: V1-like receptive fields



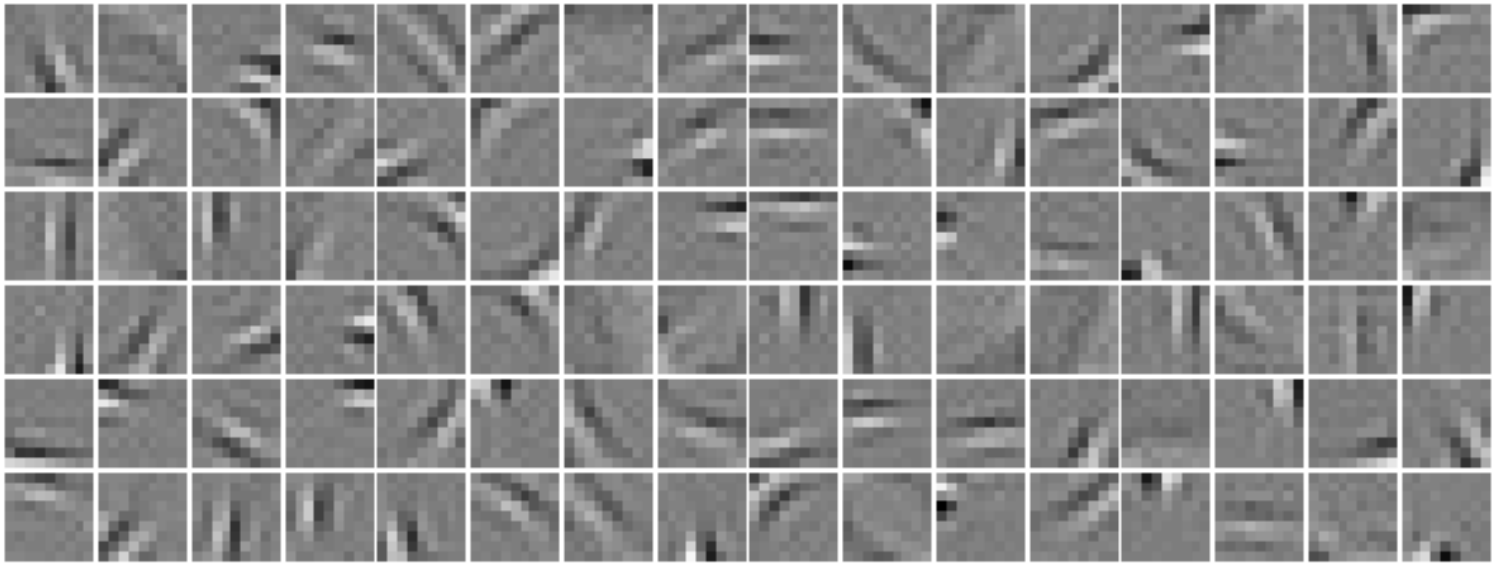
Learned Features: V1-like receptive fields

- 12x12 filters
- 1024 filters

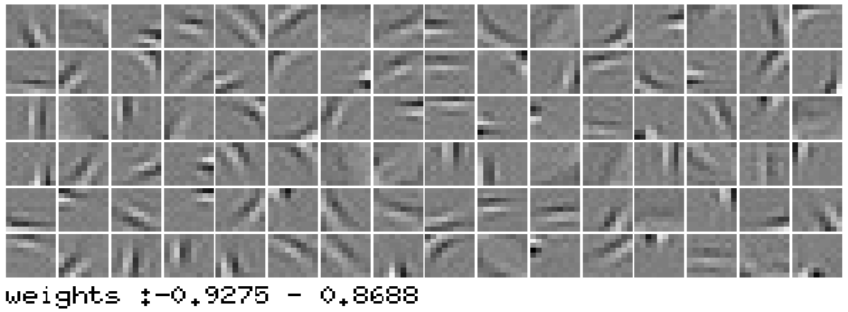


Using Predictable Basis Pursuit Features for Recognition

- **96 filters on 9x9 patches trained with PBP**
  - with Linear-Sigmoid-Gain Encoder
- **Recognition:**
  - Normalized_Image -> Learned_Filters -> Rectification -> Local_Normalization -> Spatial_Pooling -> PCA -> Linear_Classifier
  - What is the effect of rectification and normalization?



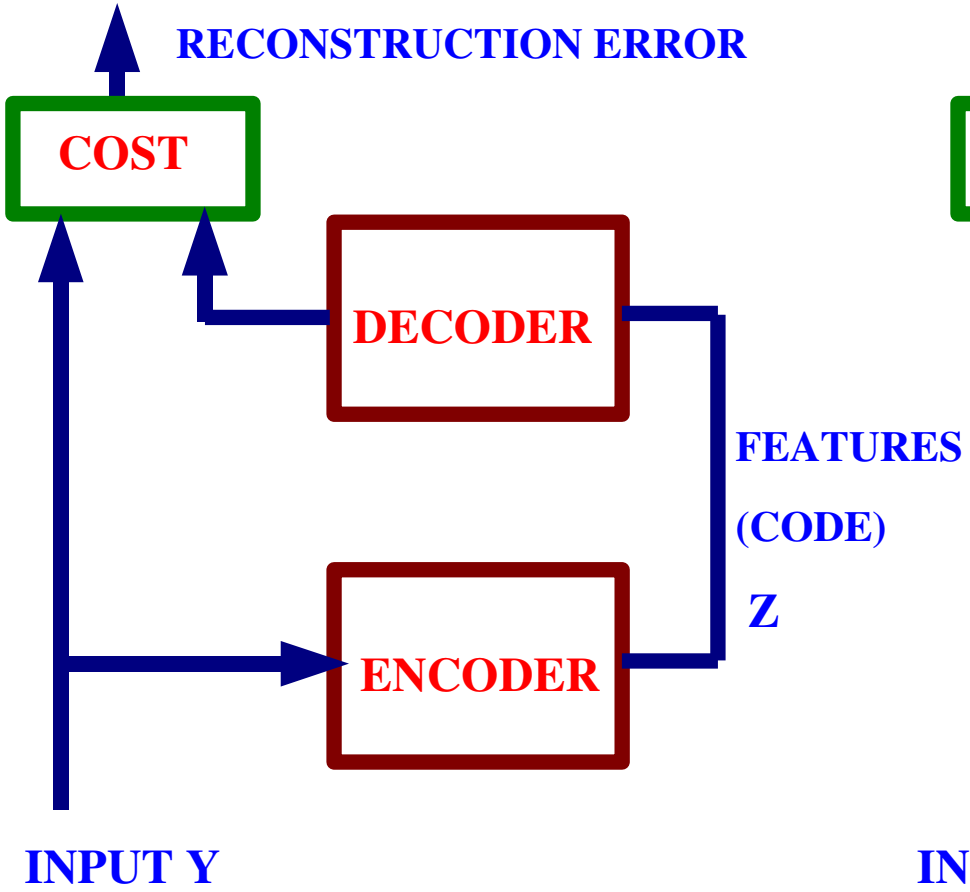
Caltech-101 Recognition Rate



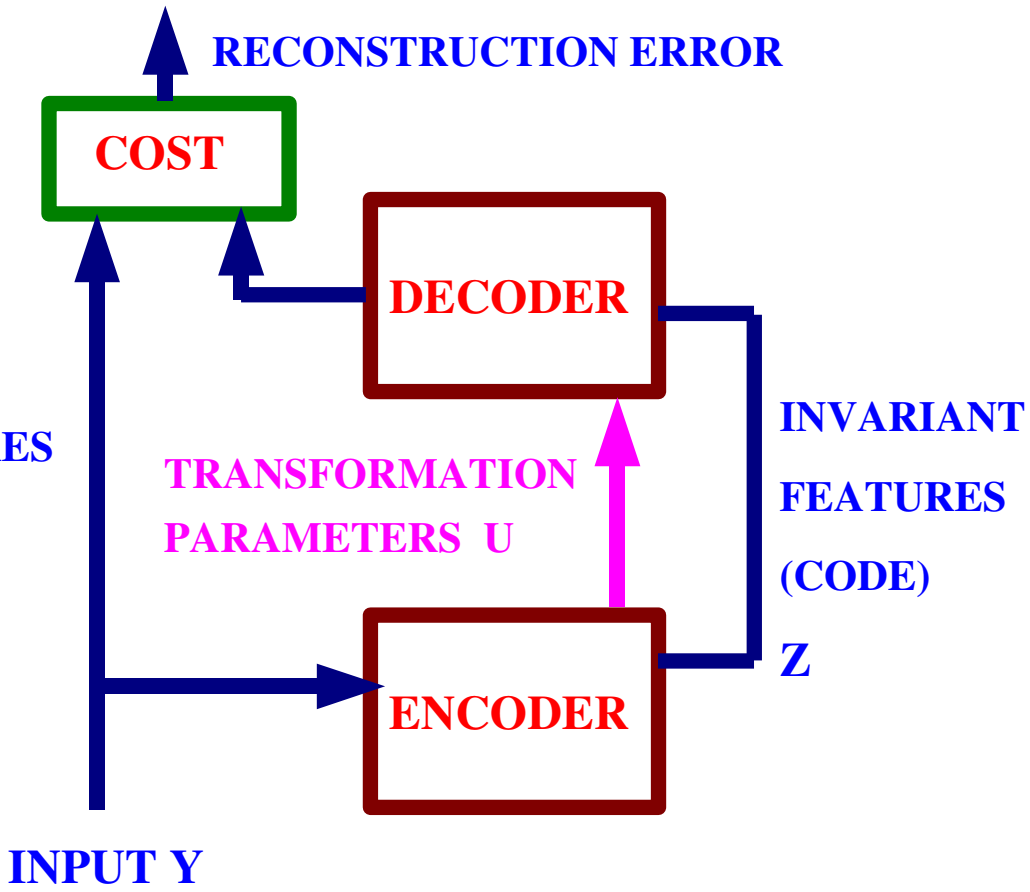
- **[96_Filters->Rectification]->Pooling->PCA->Linear_Classifier**
  - [Filters->Sigmoid] 16%
  - [Filters->Absolute_Value] 51%
  - [Local_Norm->Filters->Absolute_Value] 56%
  - [Local_Norm->Filters->Absolute_Value->Local_Norm] 58%
- **Multi-Scale Filters->Rectification->Pooling->PCA->Linear_Classifier**
  - LN->Gabor_Filters->Rectif->LN (Pinto&diCarlo 08) 59%
- **Unsupervised Convolutional Net**
  - Filt->Sigm->Pooling->Filt->Sigm->Pooling->Classifier 54%
- **Supervised Convolutional Net**
  - Filt->Sigm->Pooling->Filt->Sigm->Pooling->Classifier 20%
- **HMAX (Serre -> Mutch&Lowe 06)**
  - Filt->Sigm->Pooling->Filt->Sigm->Pooling->Classifier 56%

Learning Invariant Features [Ranzato et al CVPR 07]

- Separating the "what" from the "where"



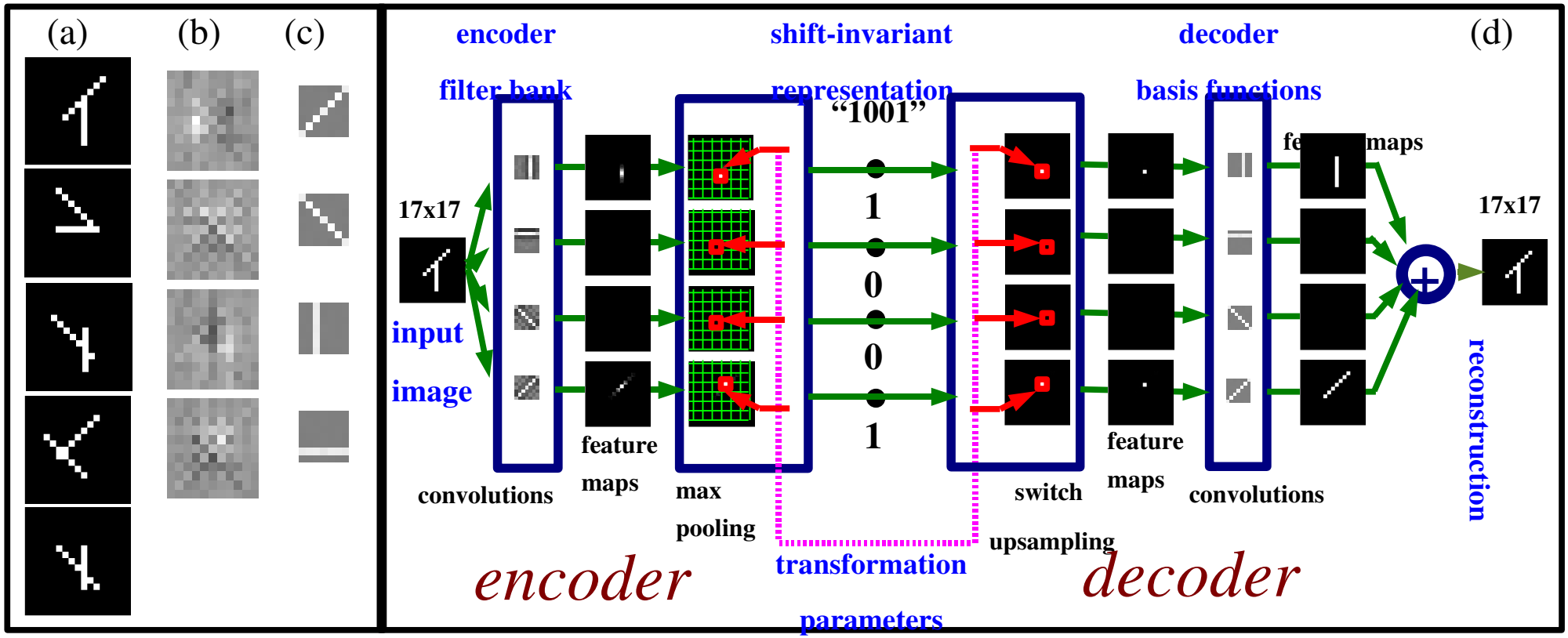
Standard Feature Extractor



Invariant Feature Extractor

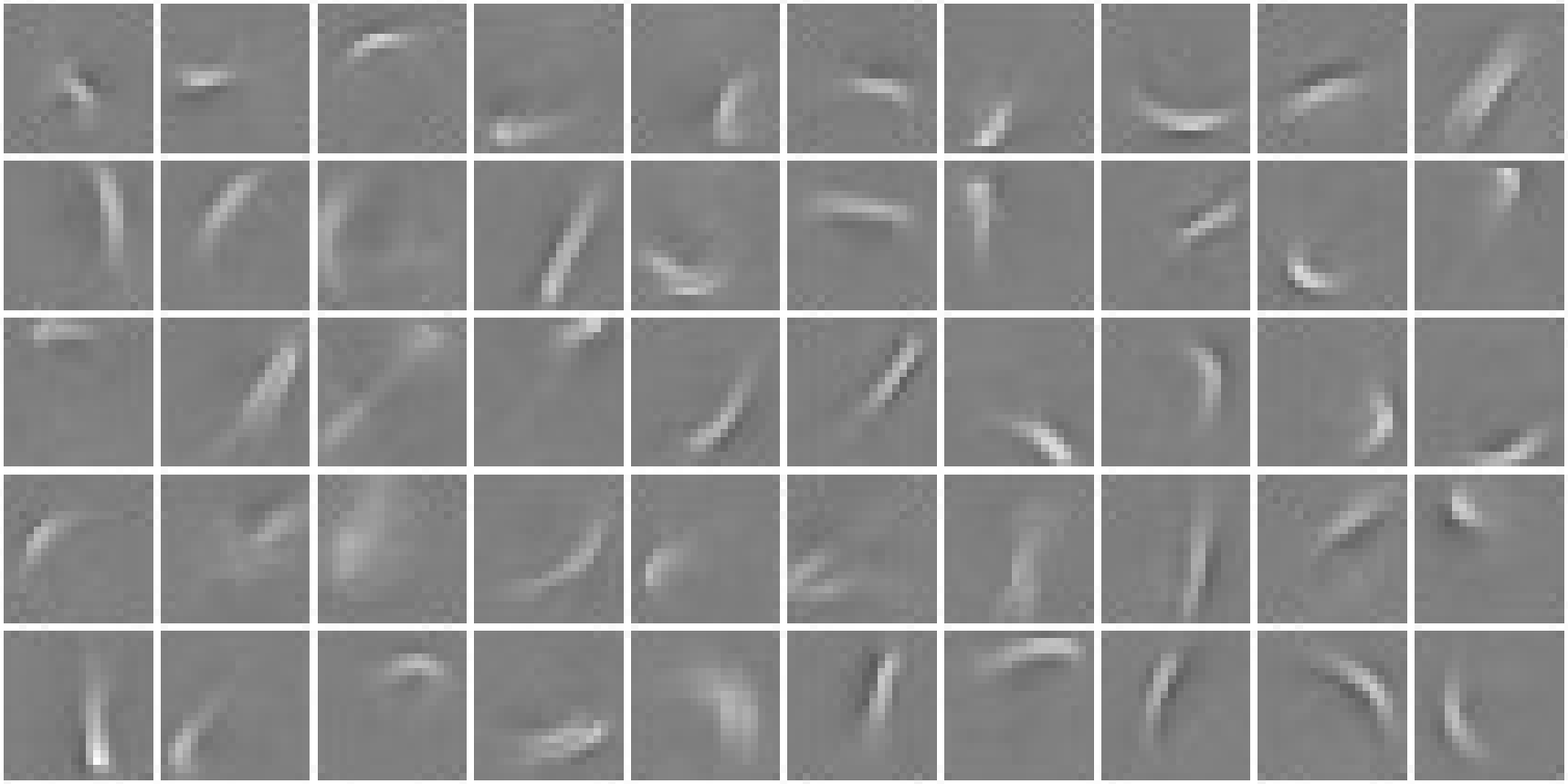
Learning Invariant Feature Hierarchies

- Learning Shift Invariant Features
- image->filters->pooling->switches->bases->reconstruction



Shift Invariant Global Features on MNIST

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

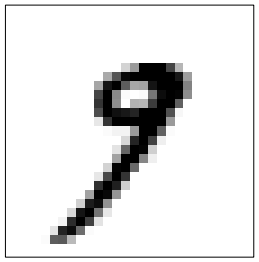
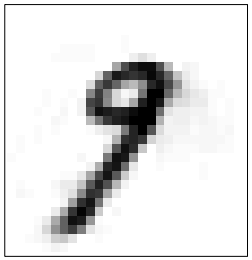


Example of Reconstruction

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

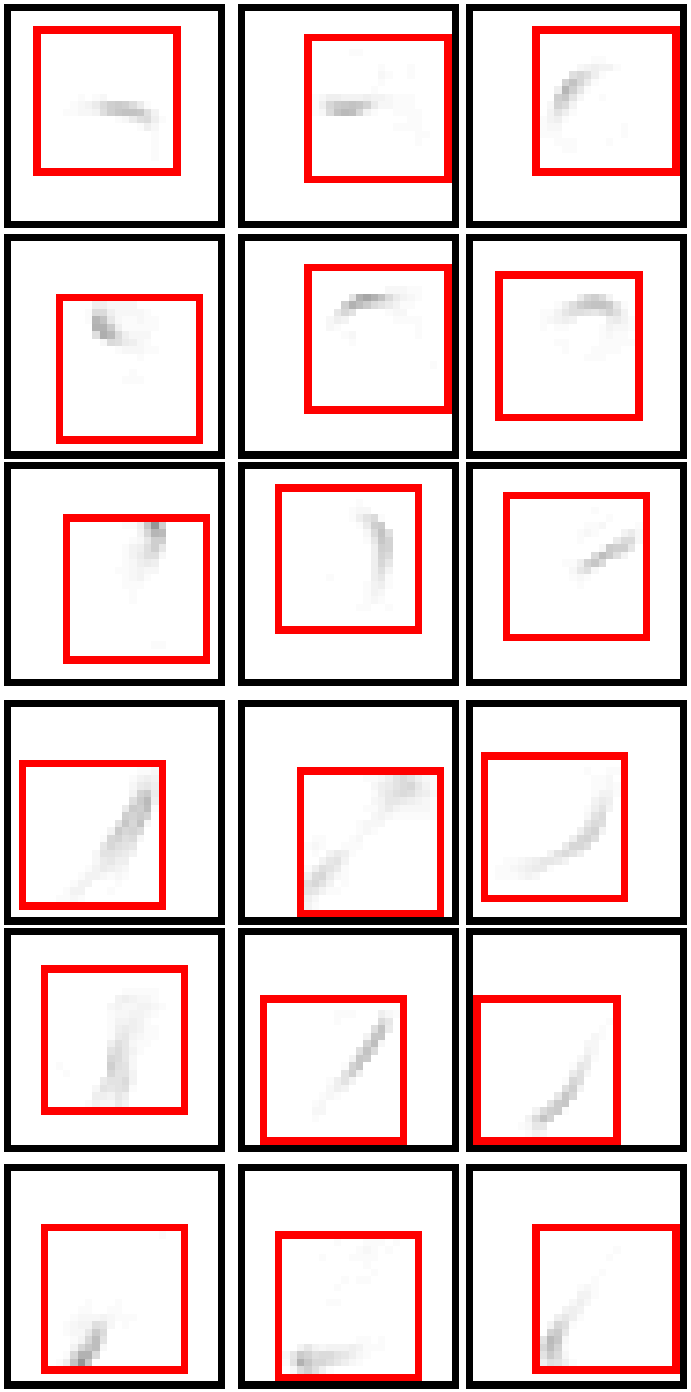
ORIGINAL DIGIT

RECONS-TRUCTION

$\sum$  $\cong$  $=$

ACTIVATED DECODER BASIS FUNCTIONS

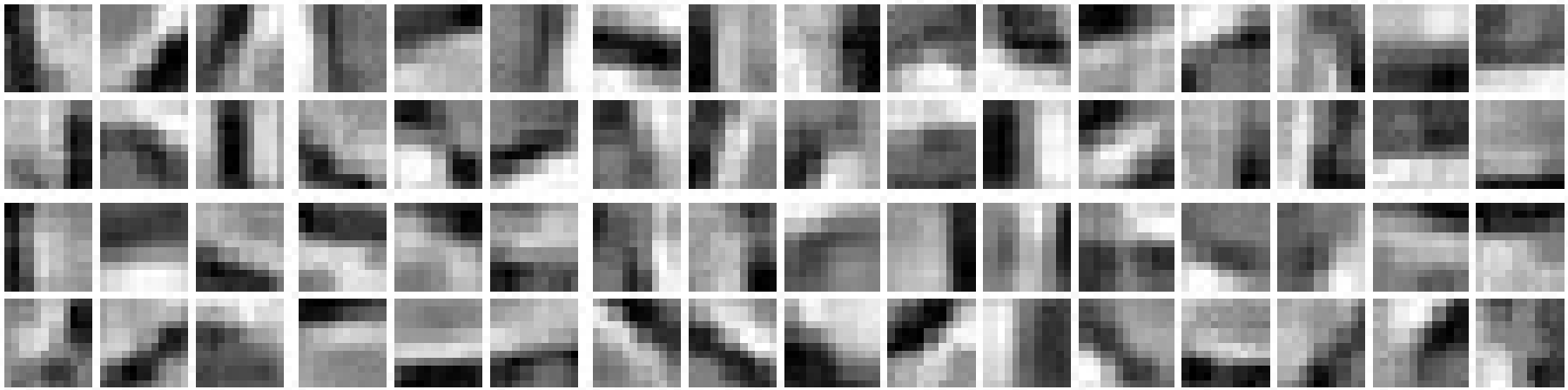
(in feed-back layer)



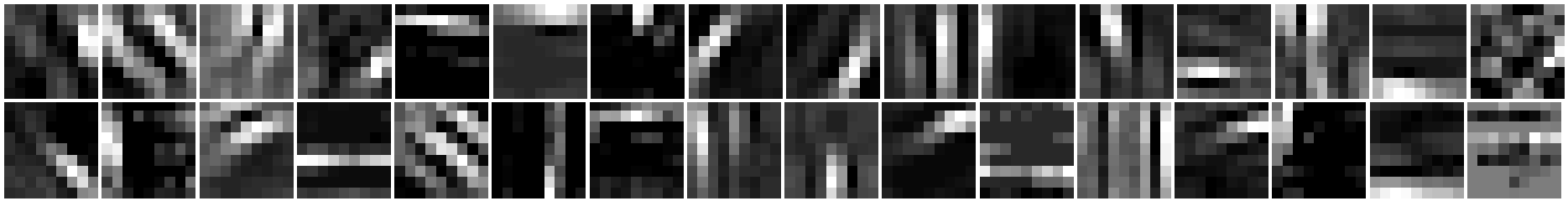
red squares: decoder bases

Sparse Enc/Dec on Object Images

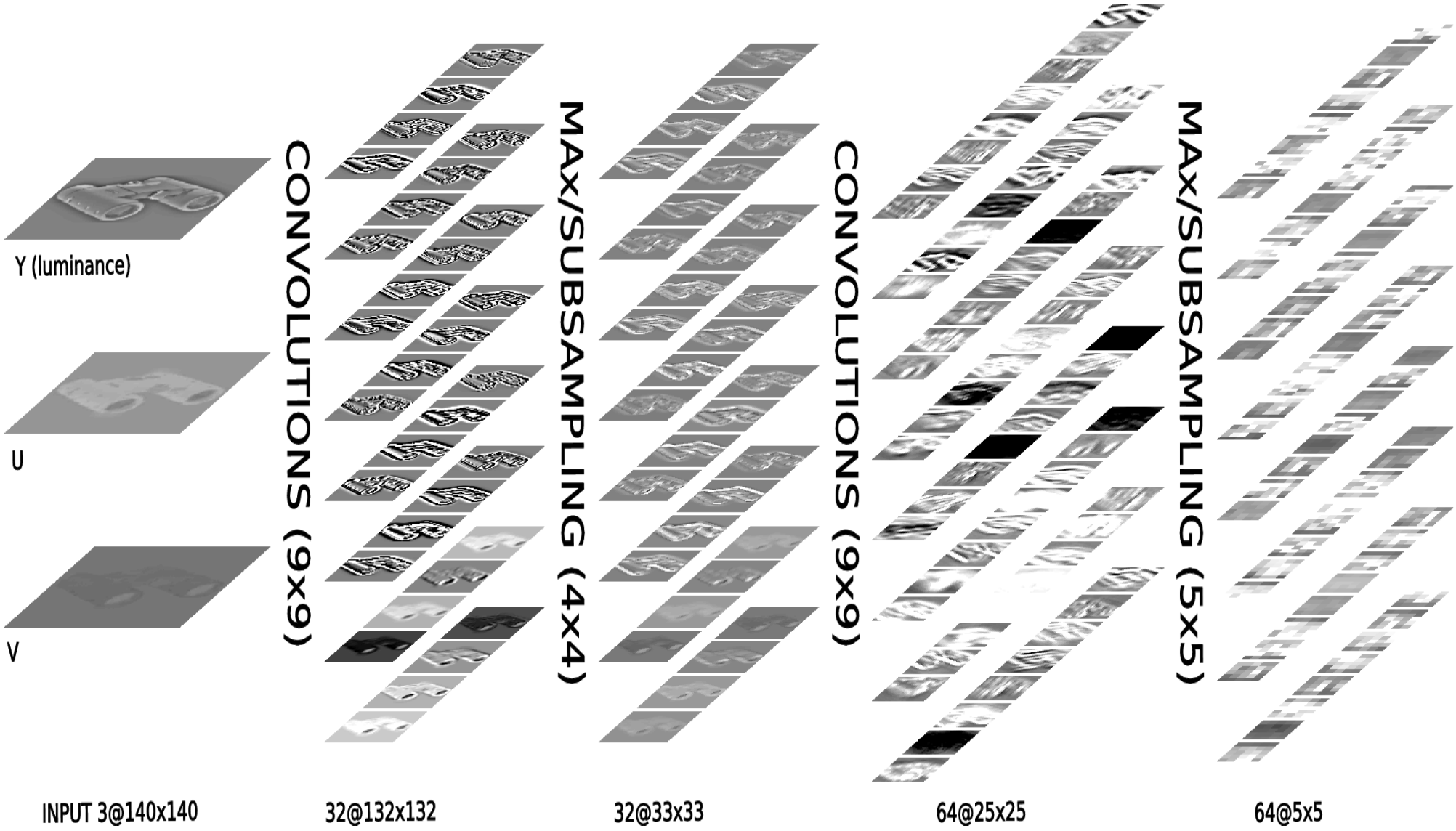
- 9x9 filters at the first level



- 9x9 filters at the second level (like V4?)

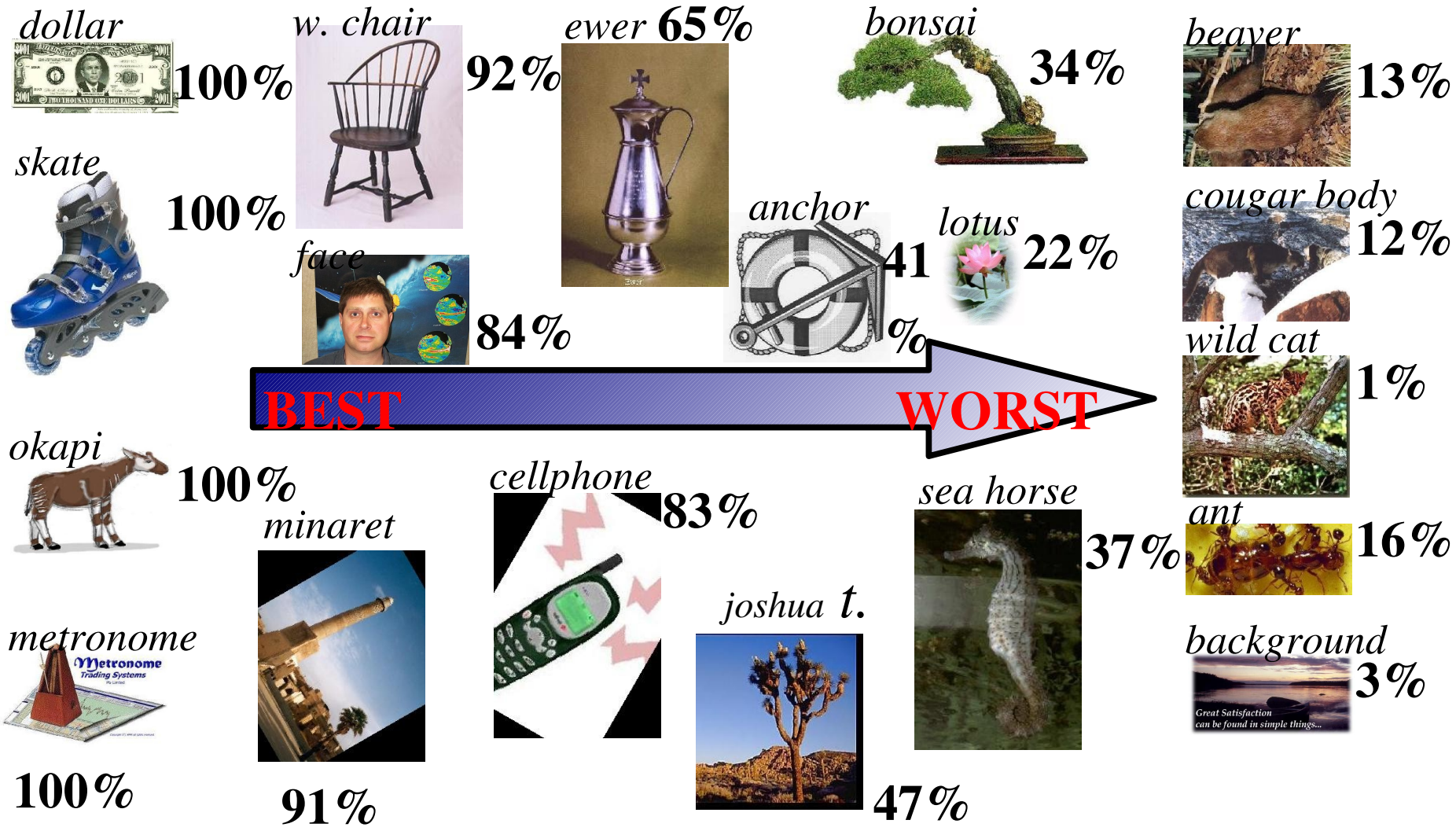


Feature Hierarchy for Object Recognition



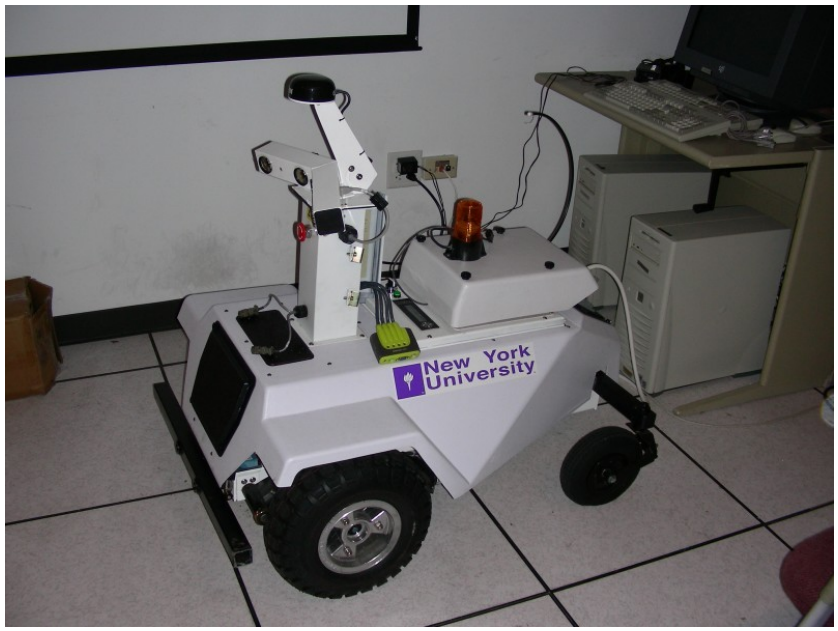
Recognition Rate on Caltech 101

- 54% on Caltech-101 (only 20% with purely supervised backprop)

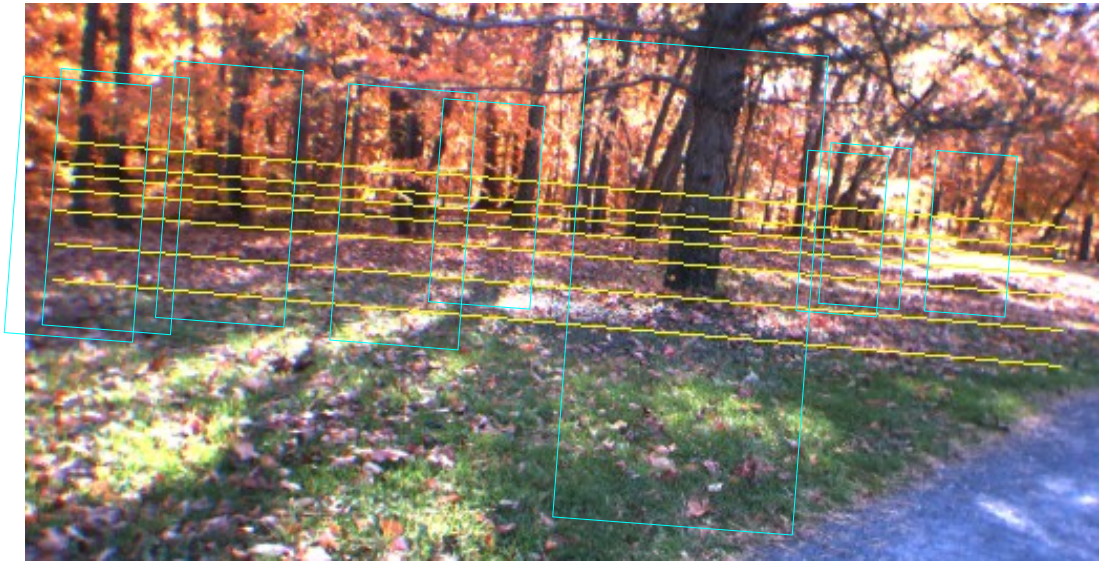


LAGR: Learning Applied to Ground Robotics

- Getting a robot to drive autonomously in unknown terrain solely from vision (camera input).
- Our team (NYU/Net-Scale Technologies Inc.) is 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.

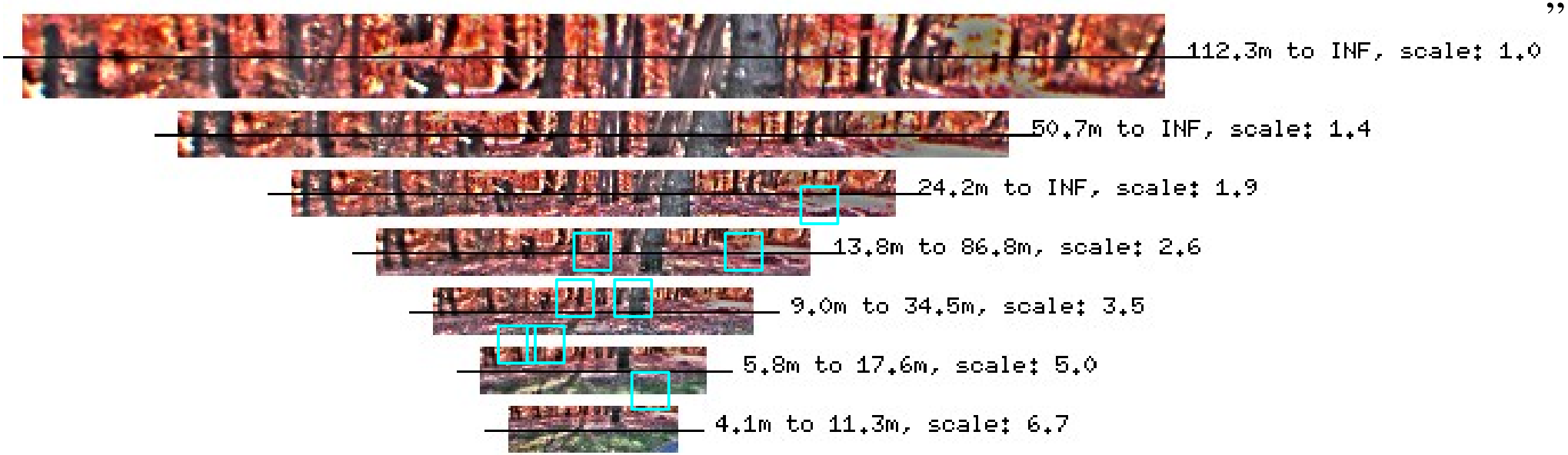


Long Range Vision: Distance Normalization



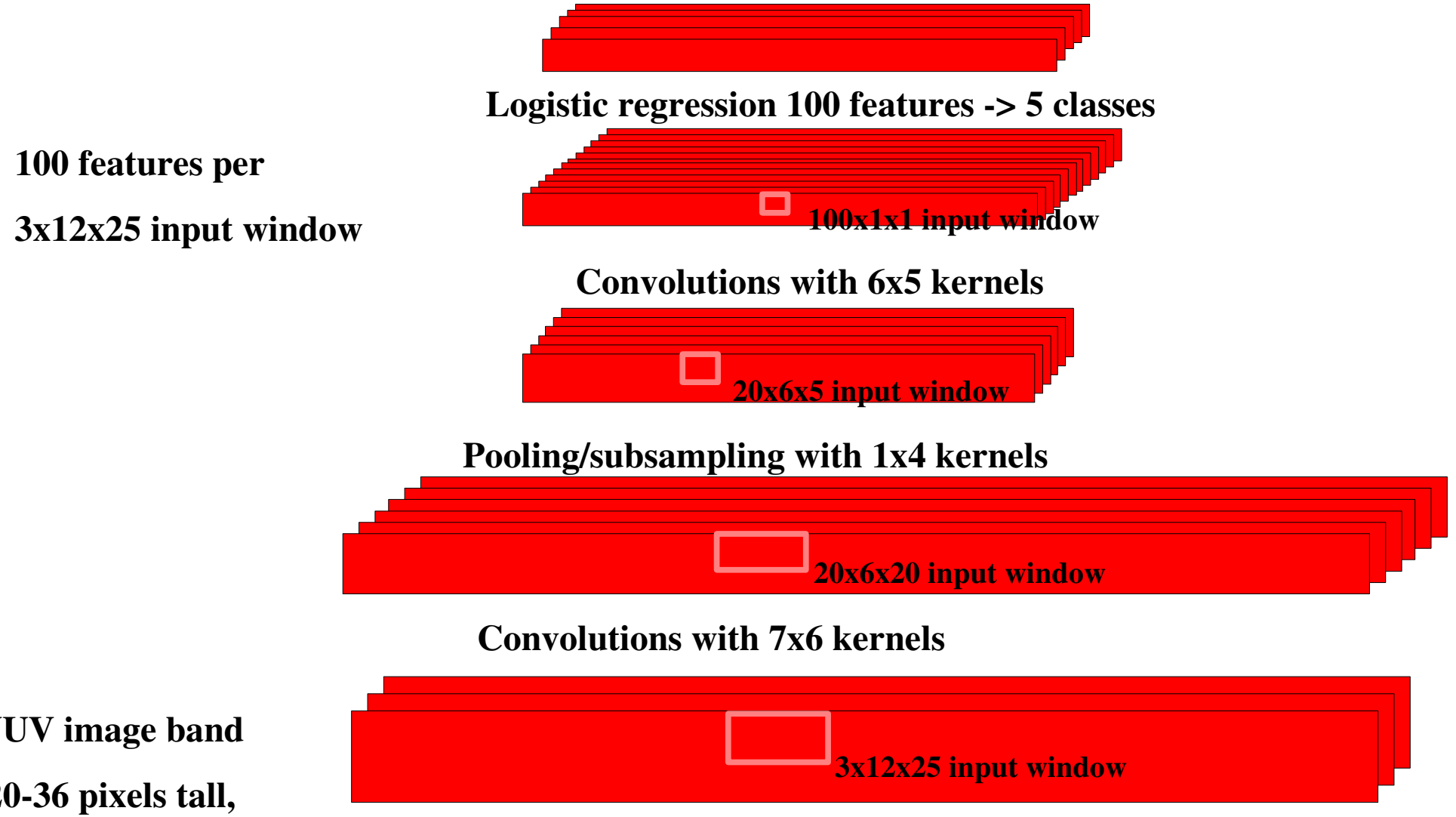
Pre-processing (125 ms)

- Ground plane estimation
- Horizon leveling
- Conversion to YUV + local contrast normalization
- Scale invariant pyramid of ”

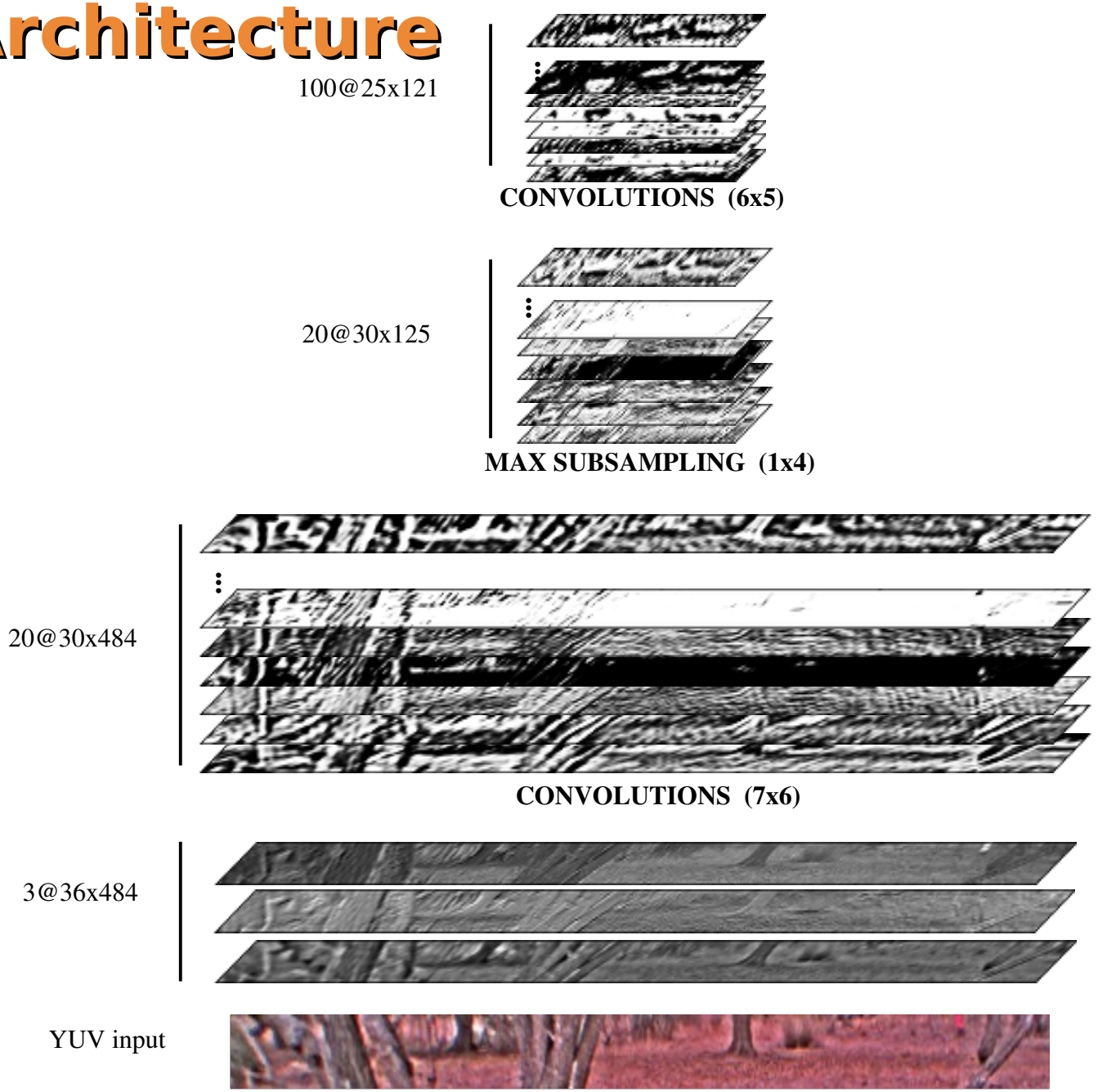


Convolutional Net Architecture

- Operates on 12x25 YUV windows from the pyramid



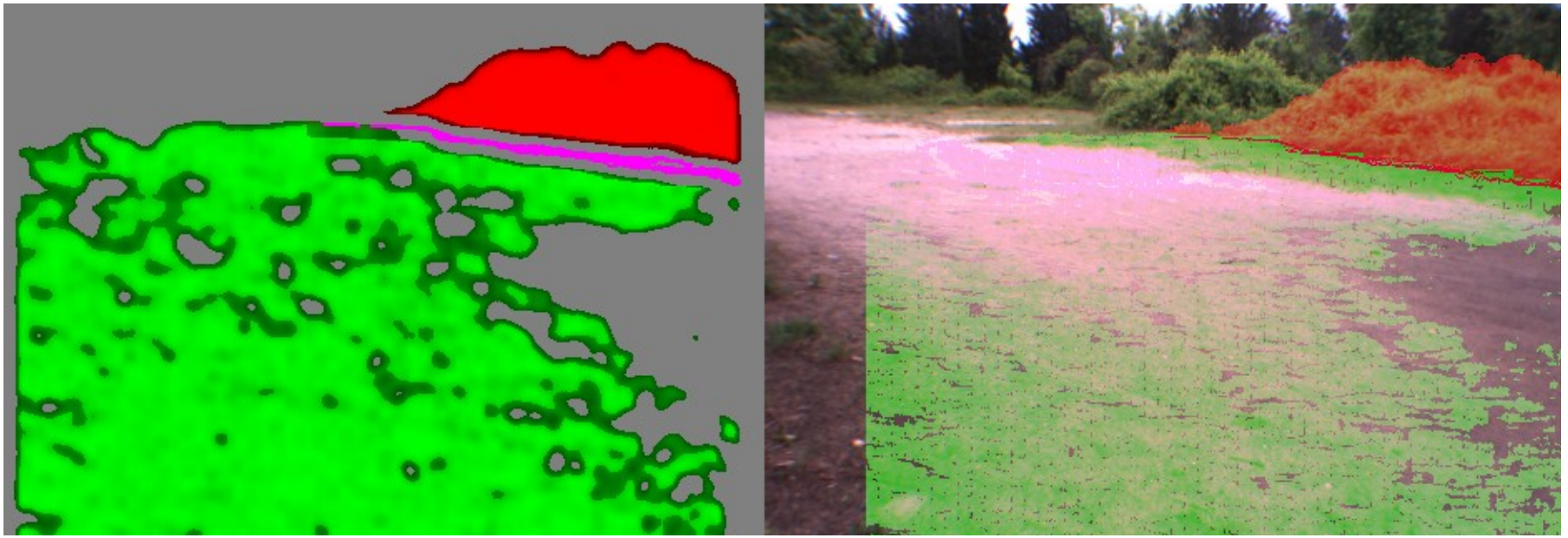
Convolutional Net Architecture



Long Range Vision: 5 categories

Online Learning (52 ms)

- Label windows using stereo information – 5 classes



super-ground

ground

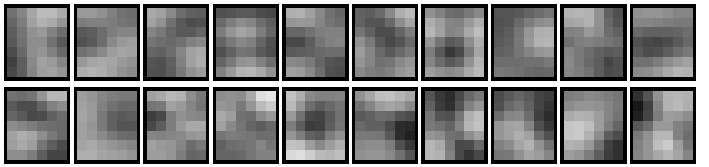
footline

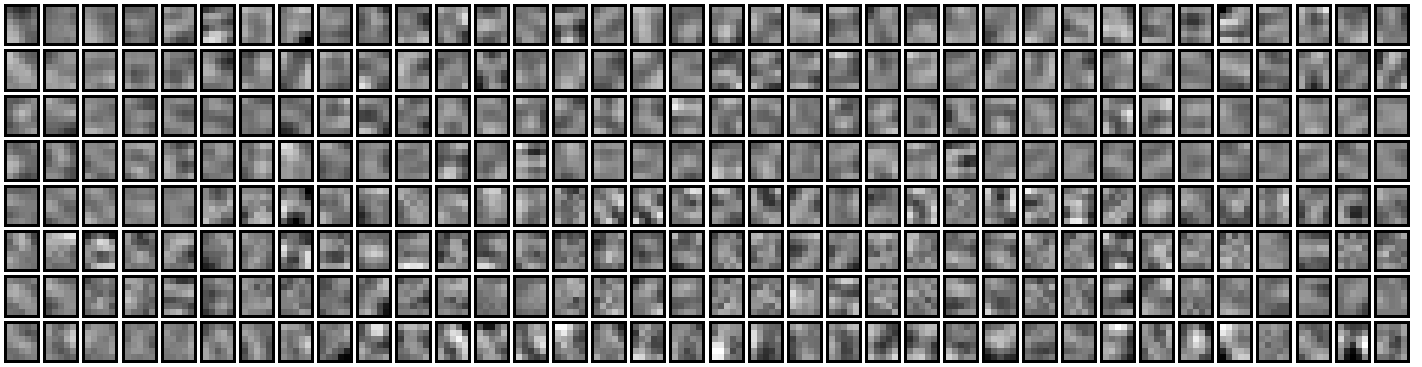
obstacle

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

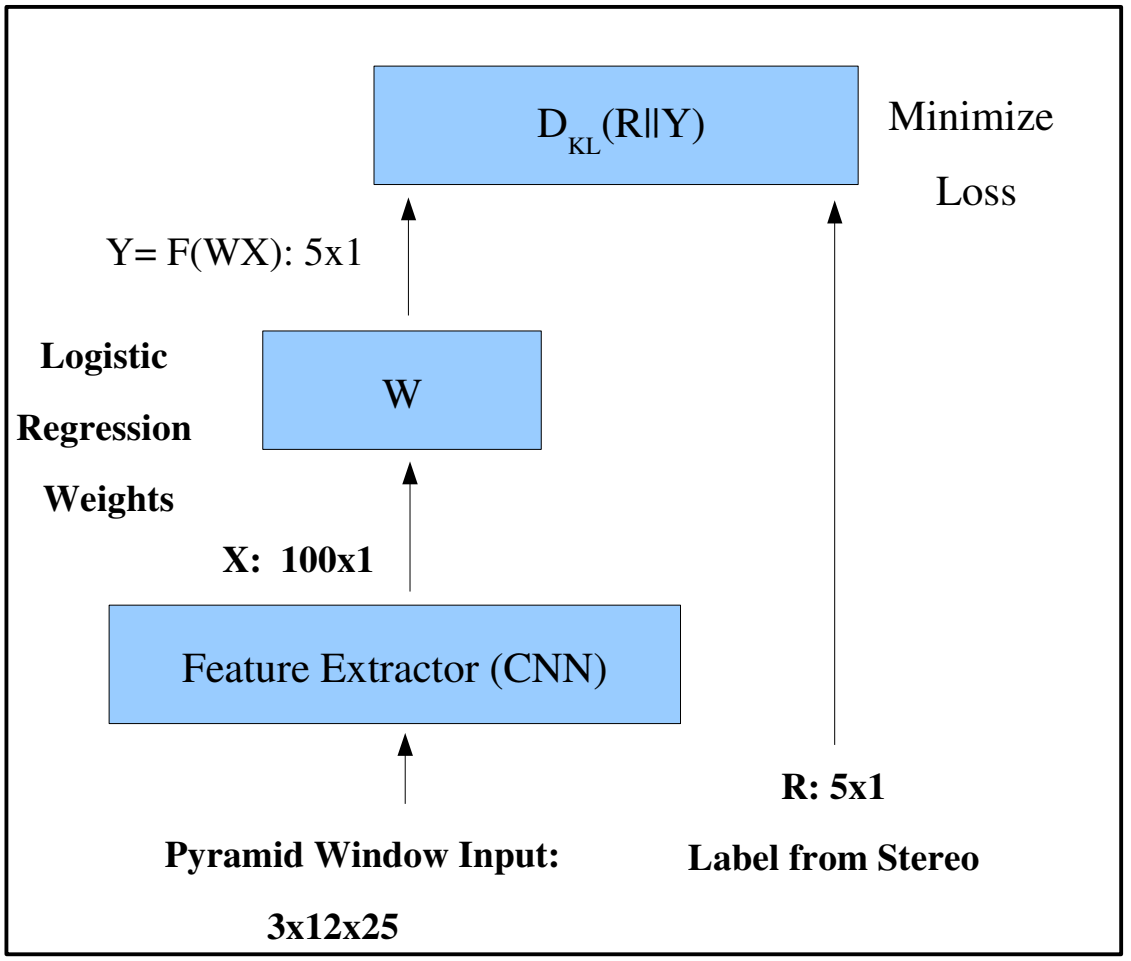




Long Range Vision: the Classifier

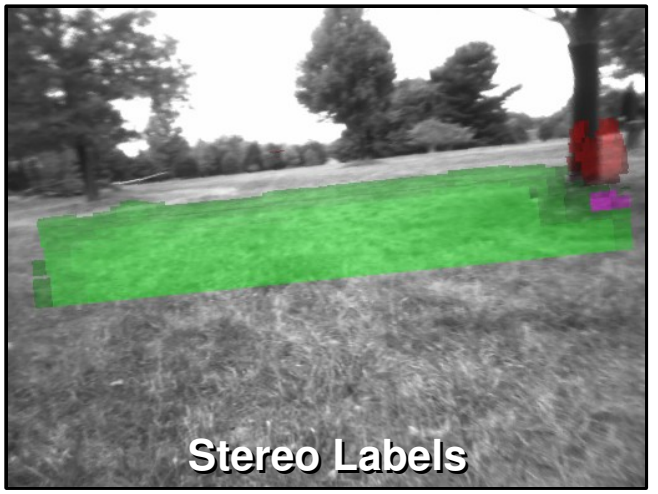
Online Learning (52 ms)

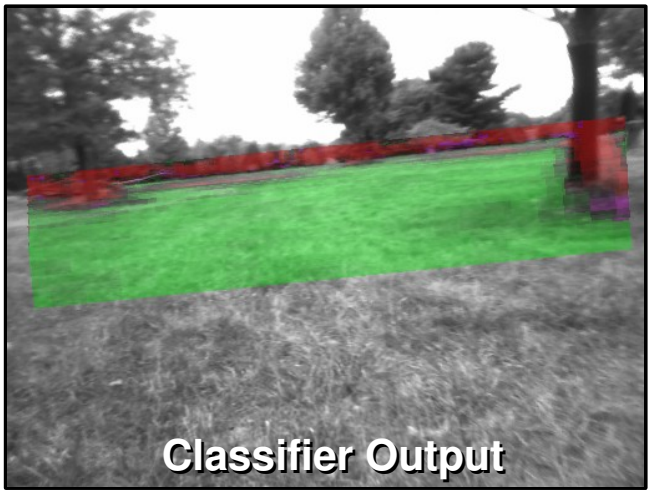
- Train a logistic regression on every frame, with cross entropy loss function

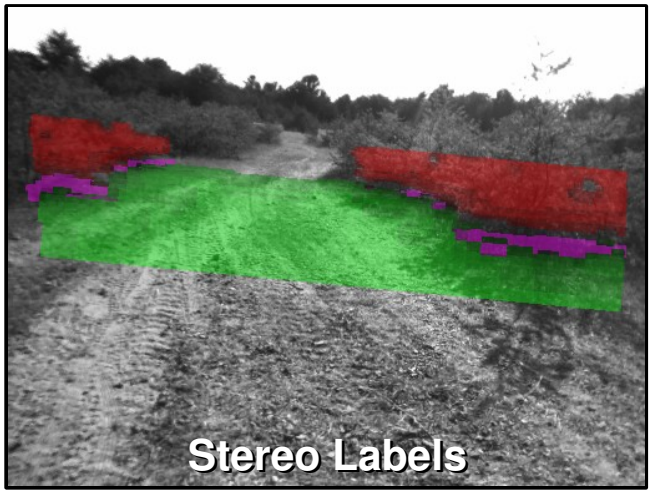


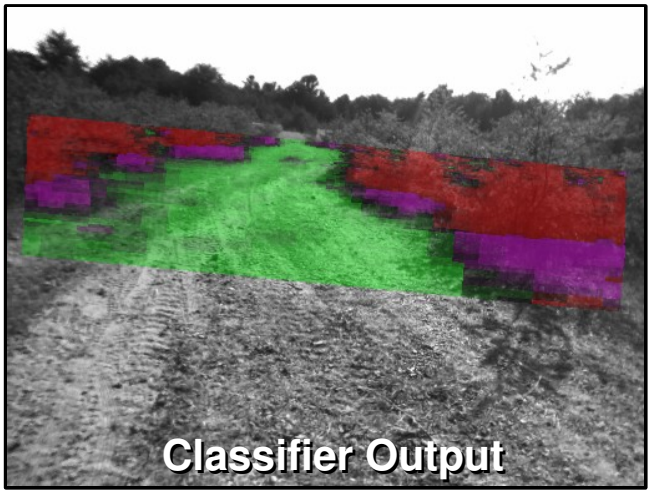
- 5 categories are learned
- 750 samples of each class are kept in a ring buffer: short term memory.
- Learning "snaps" to new environment in about 10 frames
- Weights are trained with stochastic gradient descent
- Regularization by decay to default weights

Long Range Vision Results

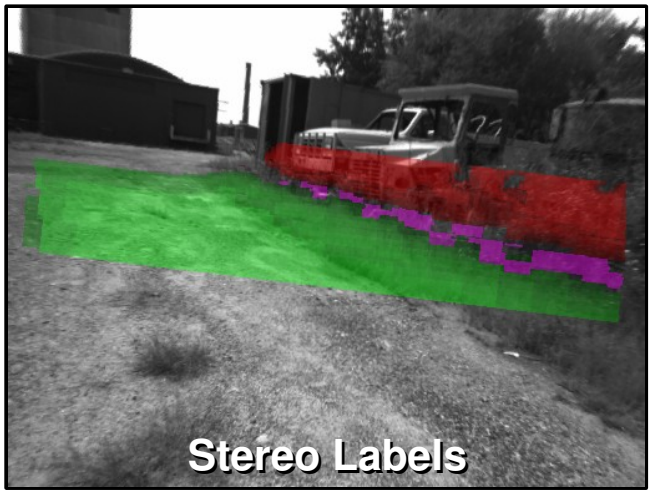


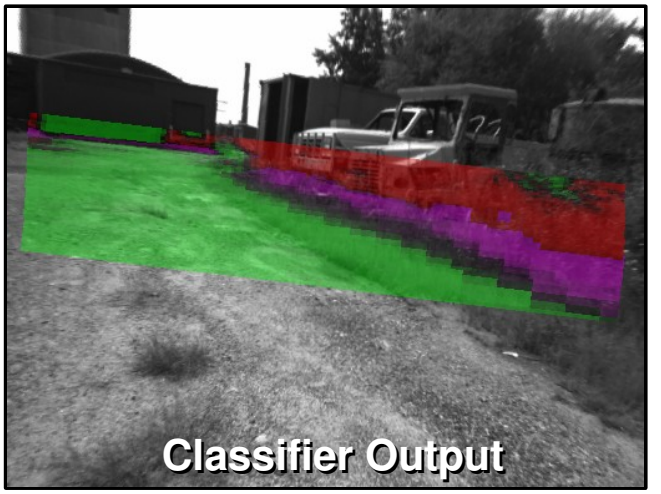


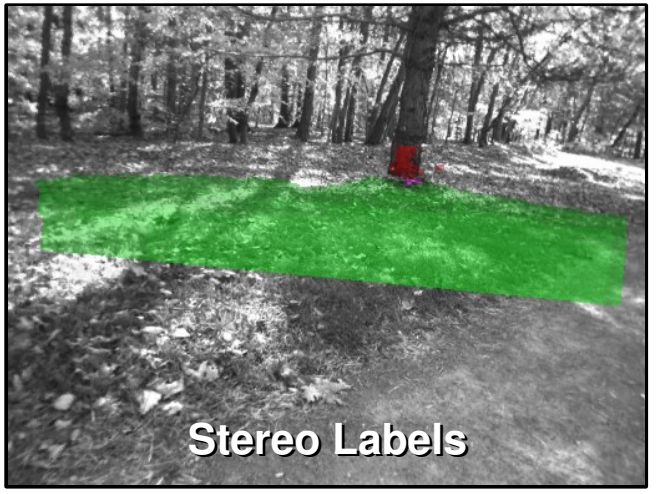


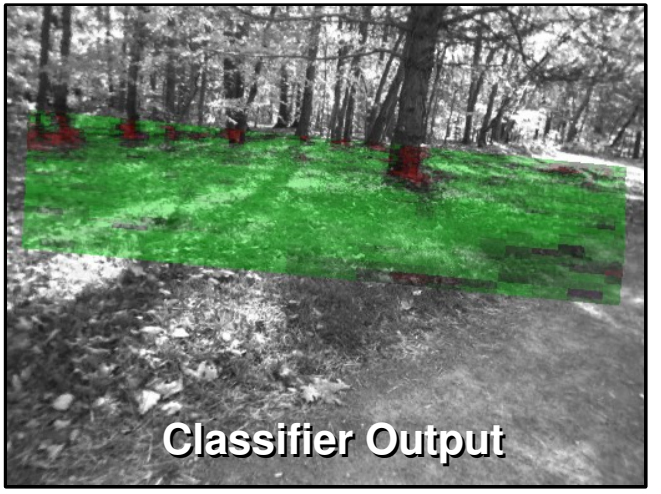


Long Range Vision Results

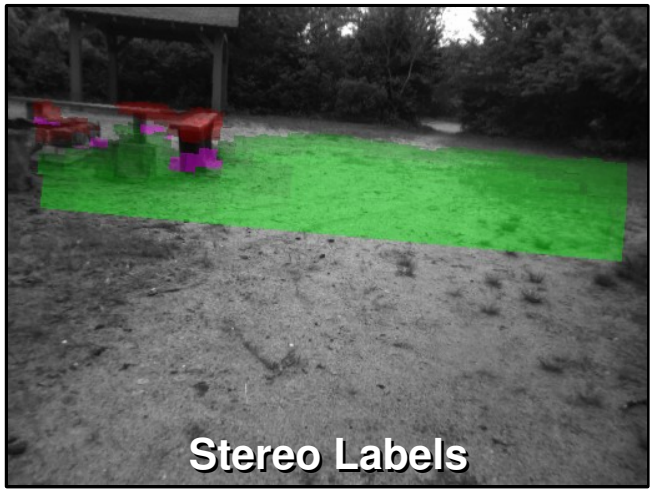


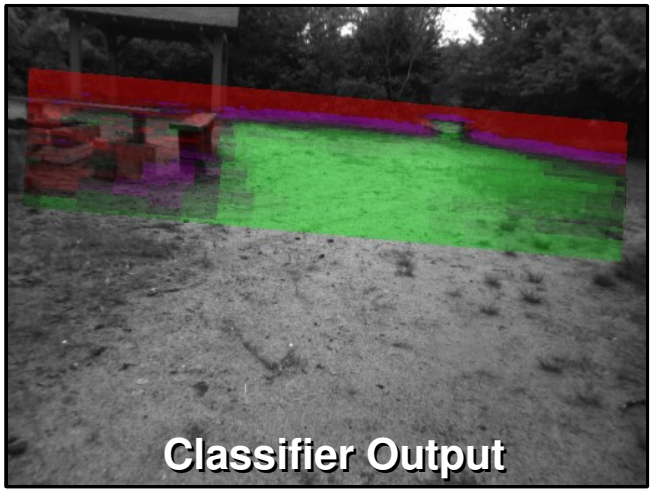


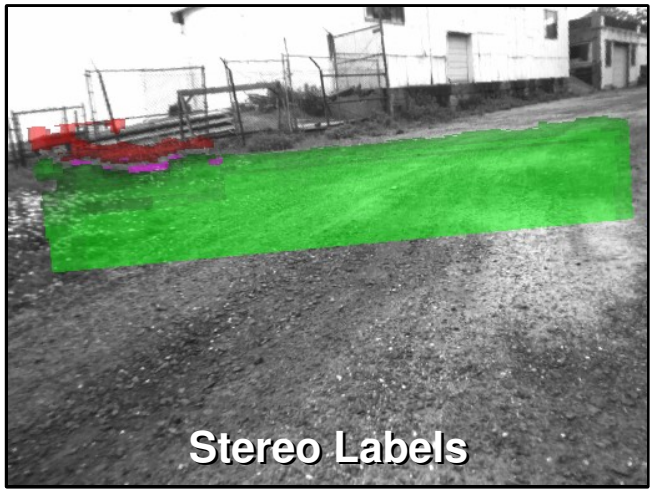


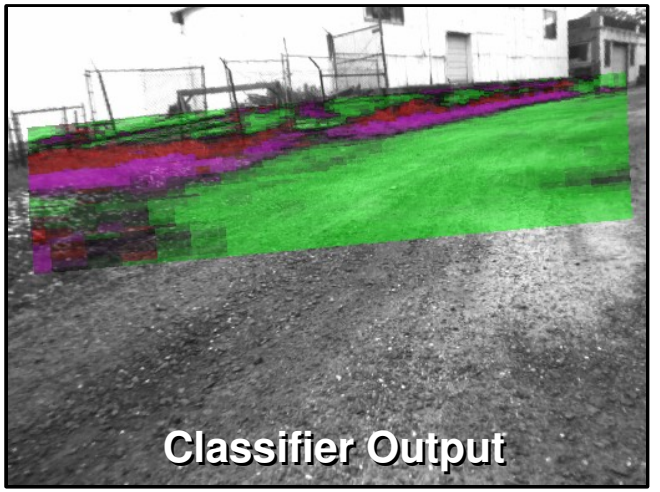


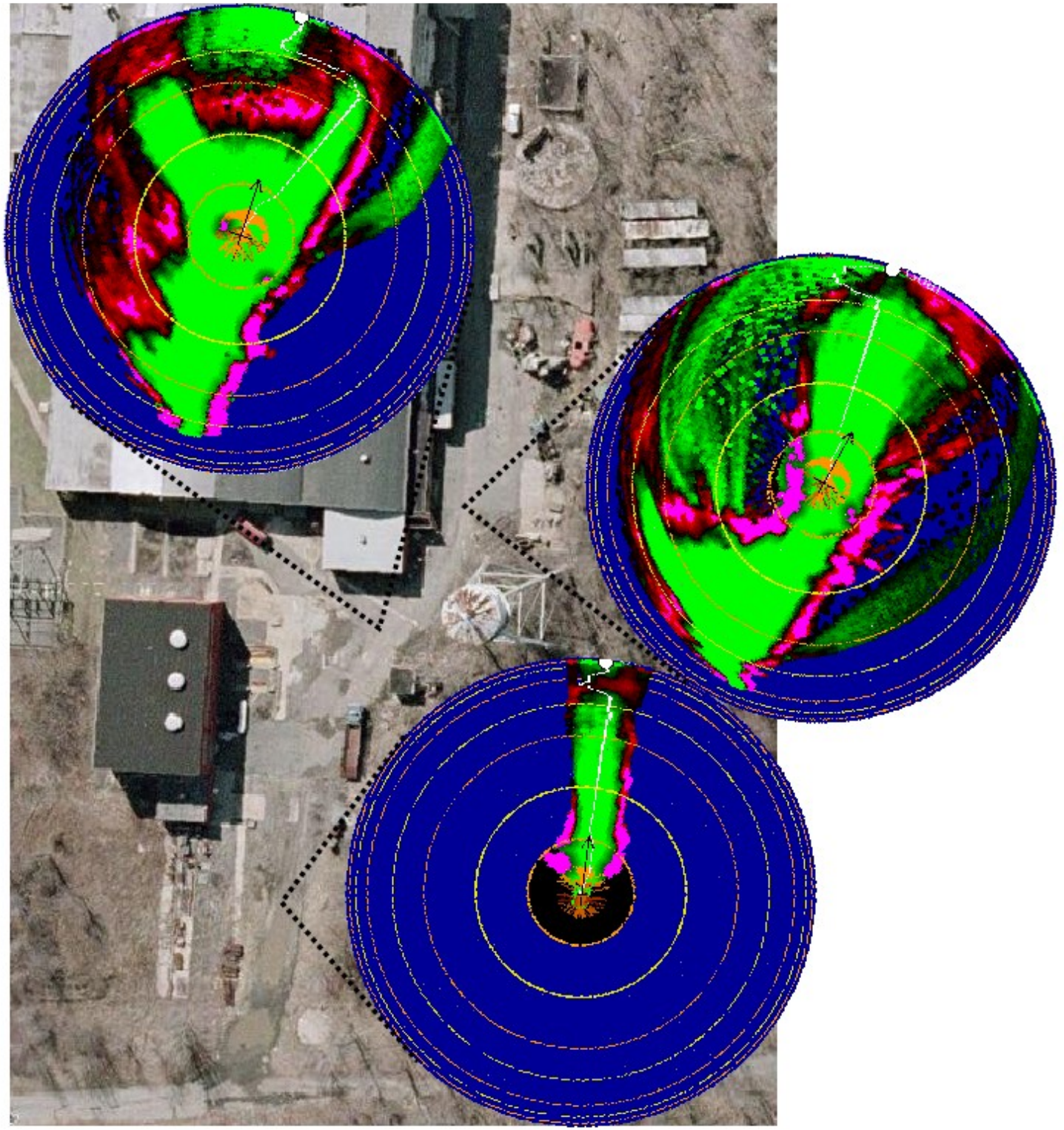
Long Range Vision Results

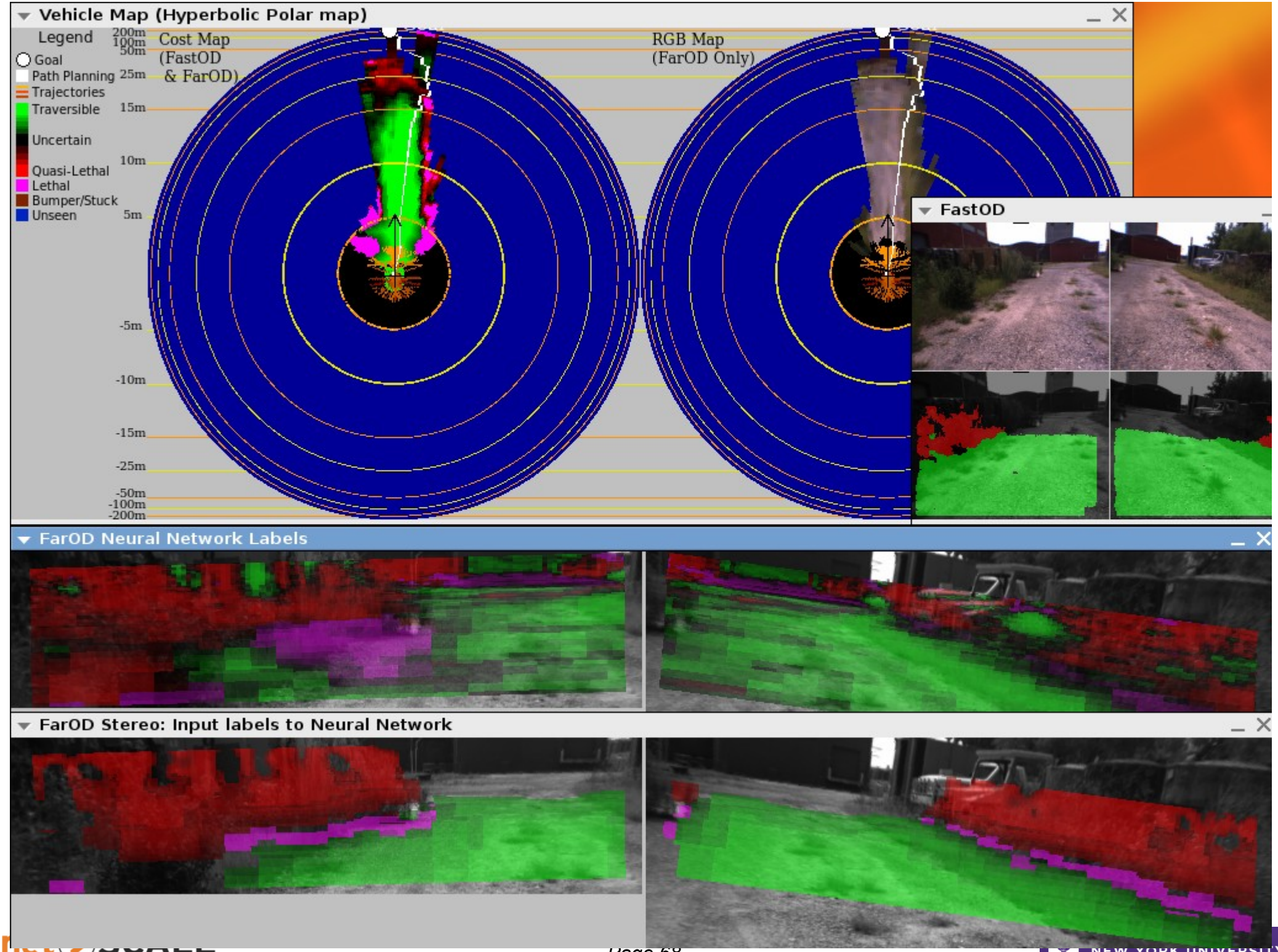


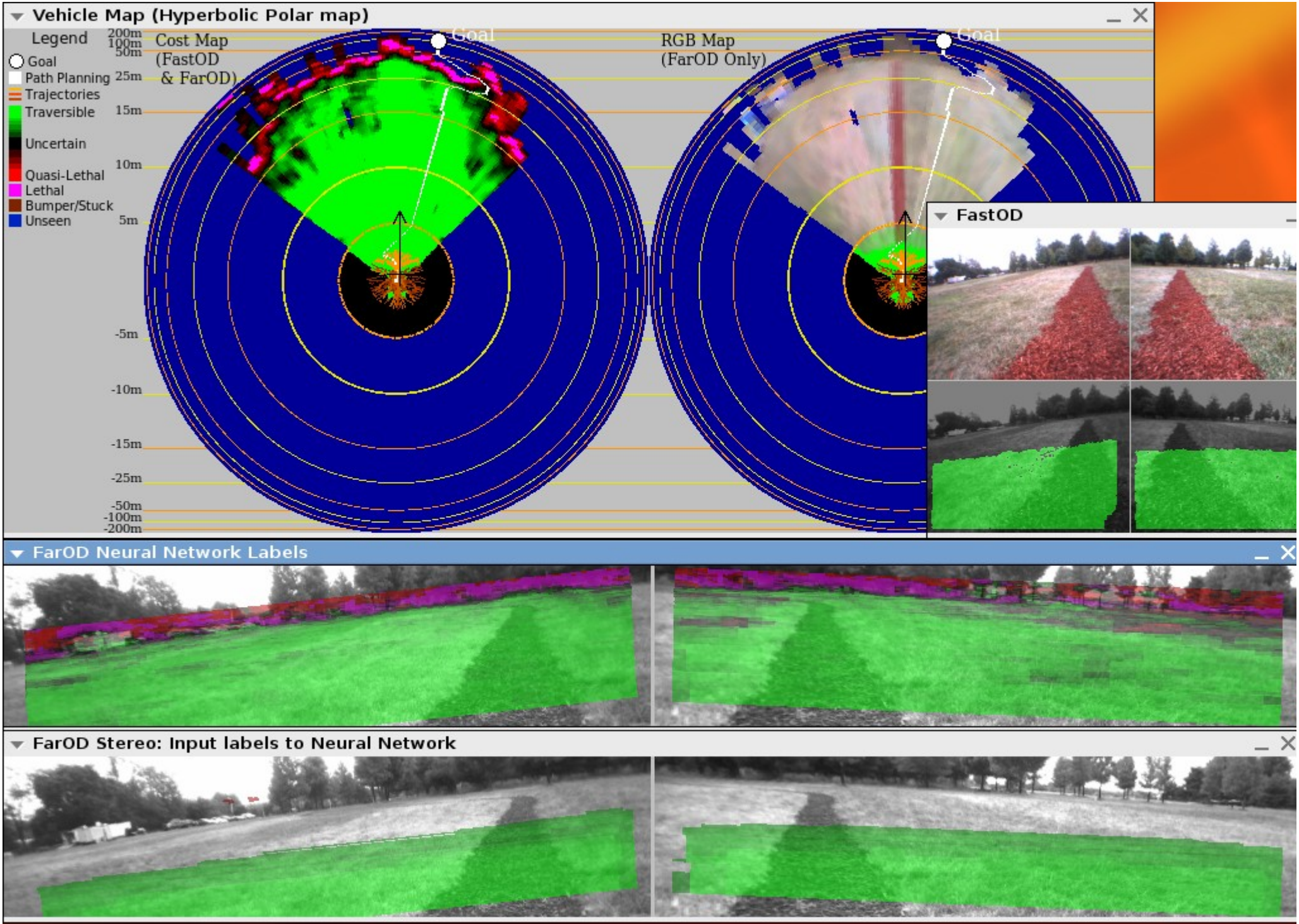


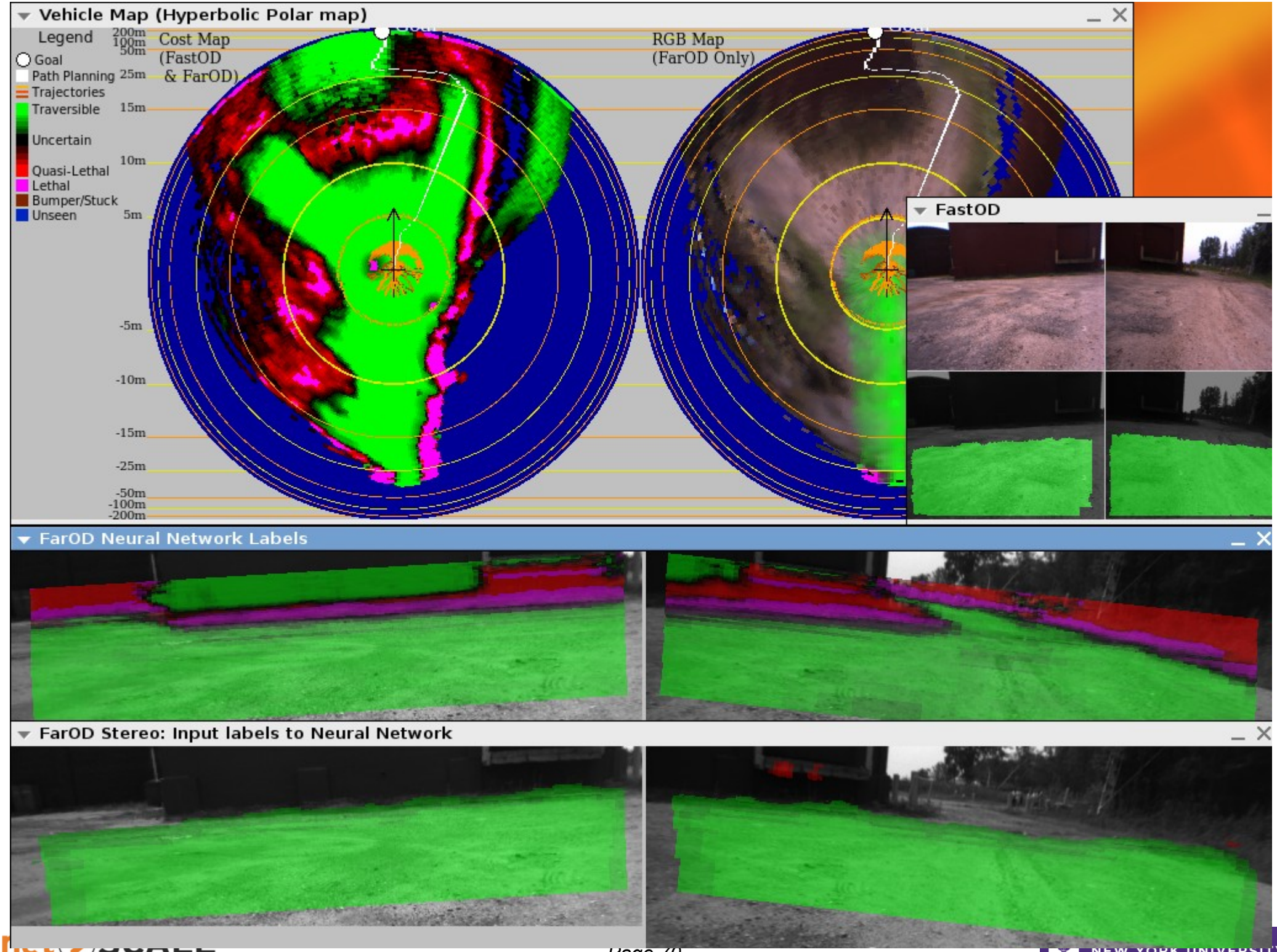


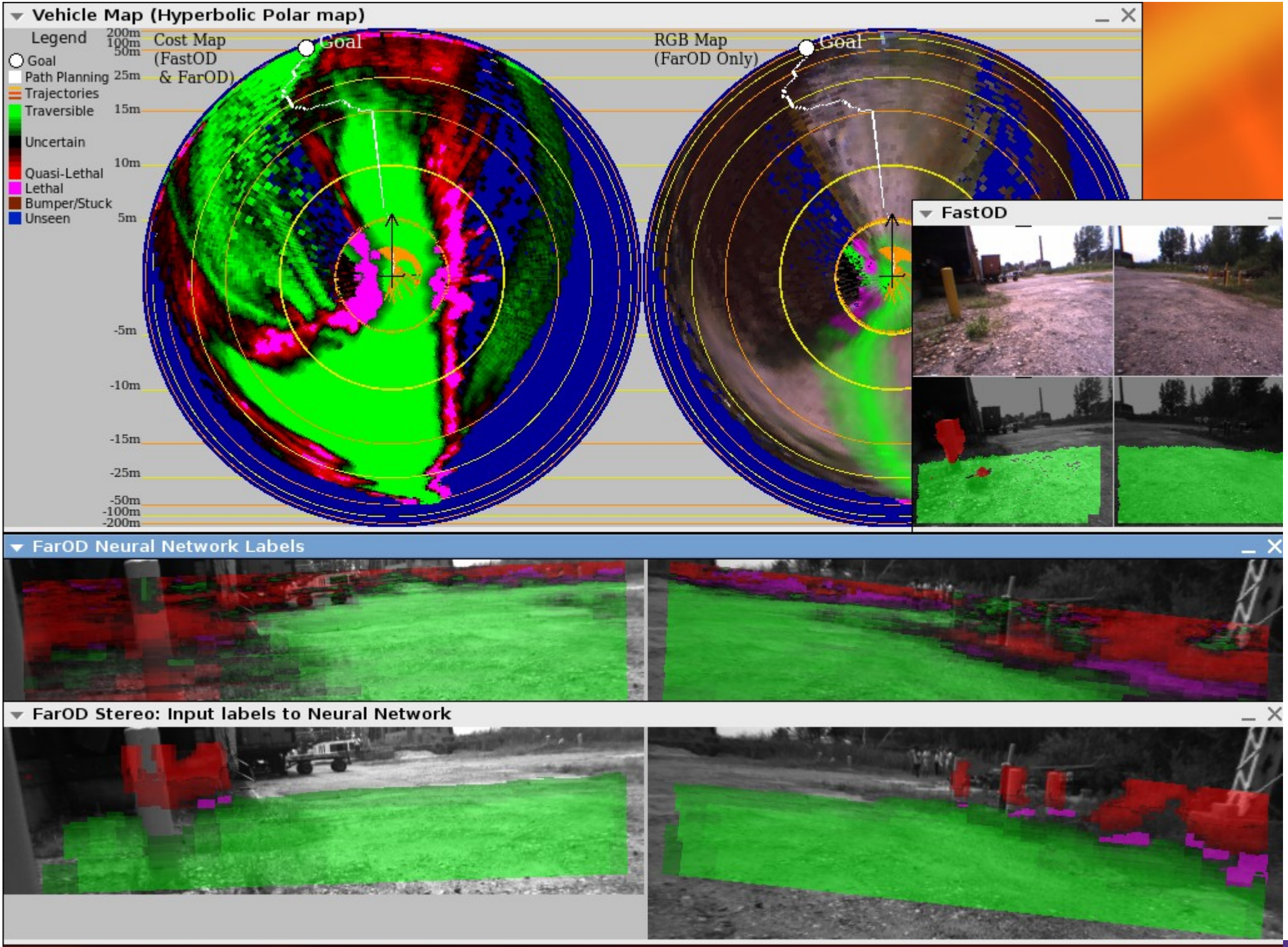


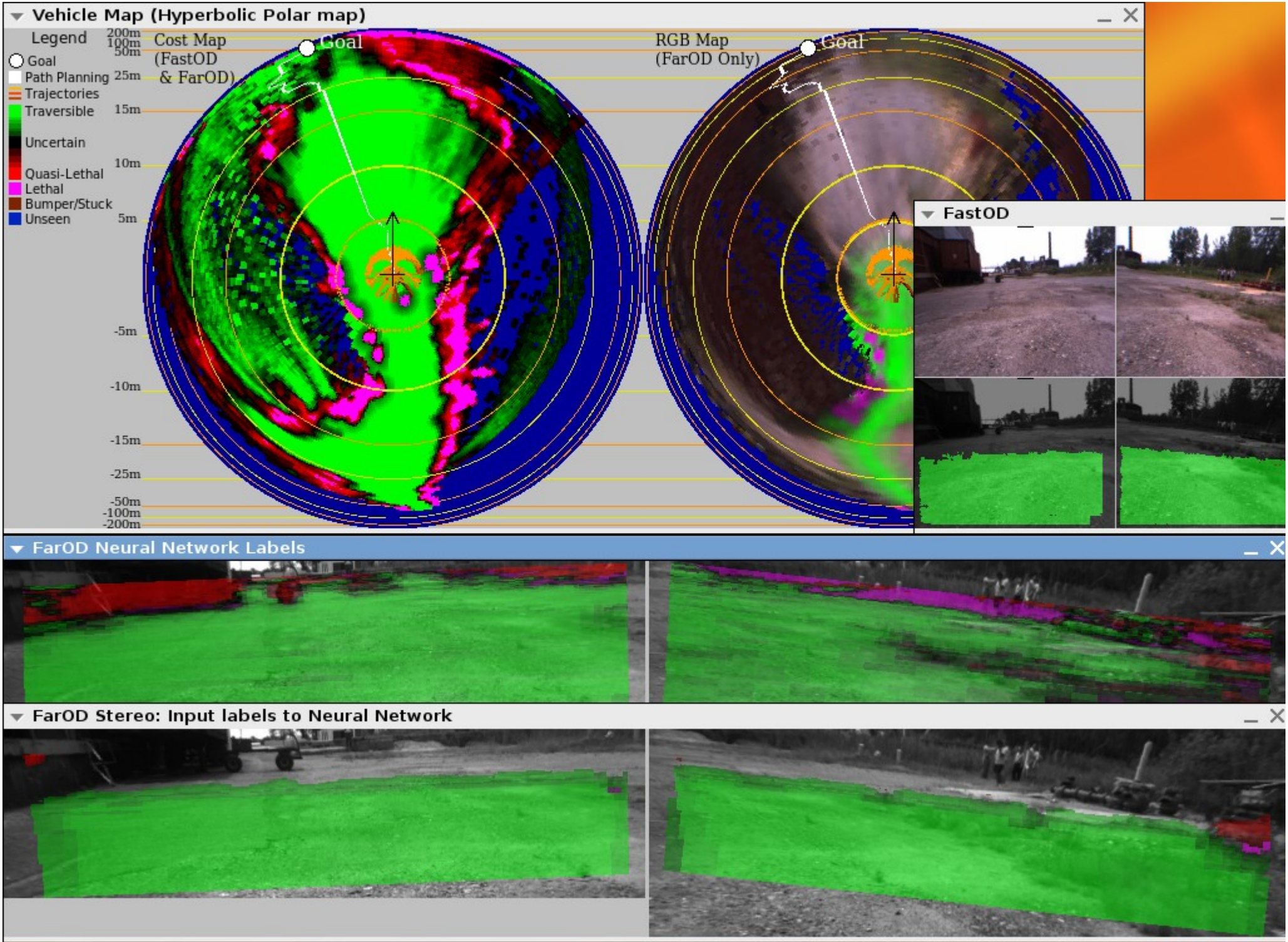



Vehicle Map (Hyperbolic Polar map)
Legend
Goal
Path Planning
Trajectories
Traversible
Uncertain
Quasi-Lethal
Lethal
Bumper/Stuck
Unseen
200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m
Cost Map
(FastOD
& FarOD)
RGB Map
(FarOD Only)
FastOD
FarOD Neural Network Labels
FarOD Stereo: Input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

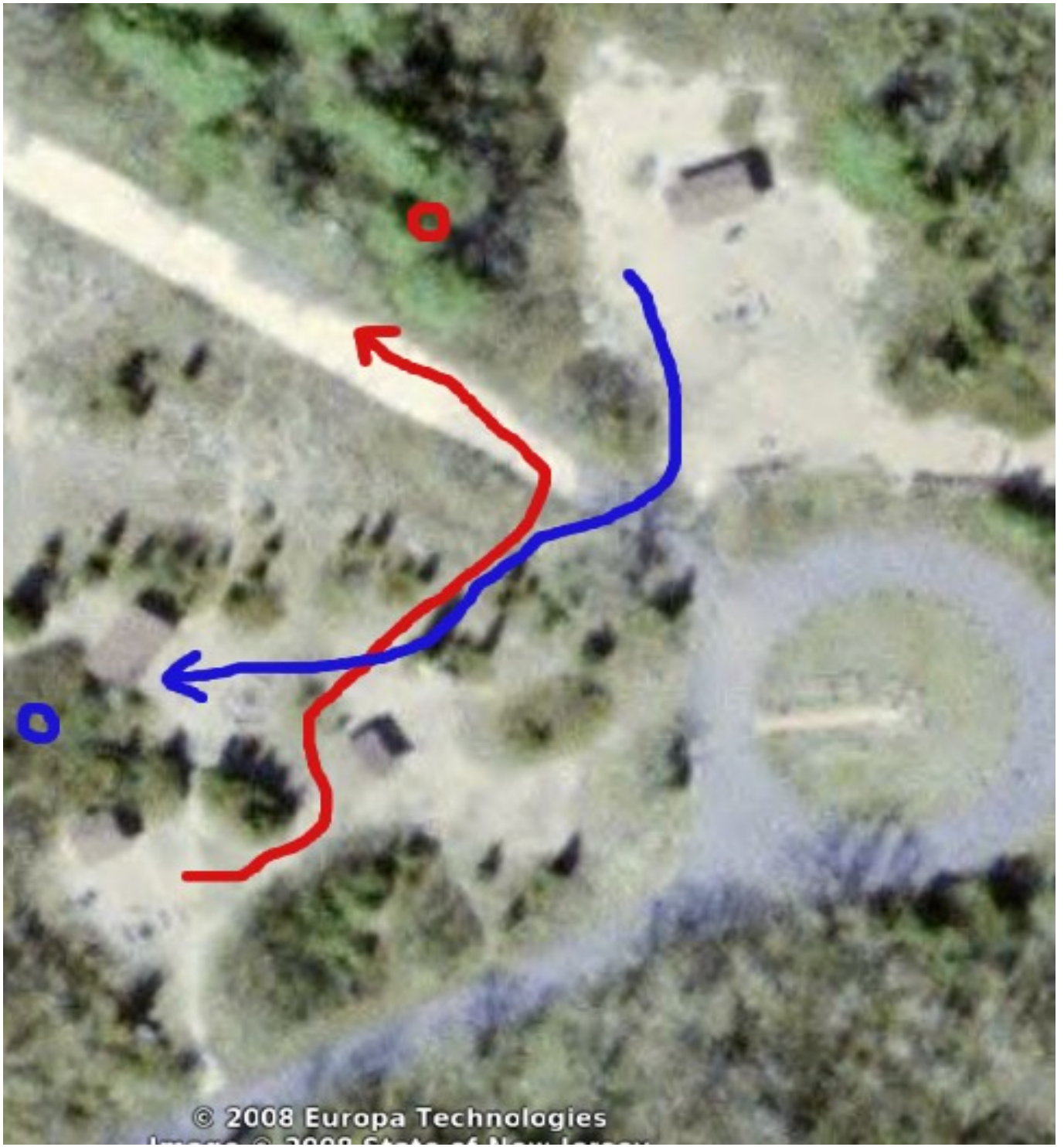

Vehicle Map (Hyperbolic Polar map)
Legend
Goal
Path Planning
Trajectories
Traversible
Uncertain
Quasi-Lethal
Lethal
Bumper/Stuck
Unseen
200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m
Cost Map
(FastOD
& FarOD)
Goal
RGB Map
(FarOD Only)
Goal
FastOD
FarOD Neural Network Labels
FarOD Stereo: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

Videos



The End