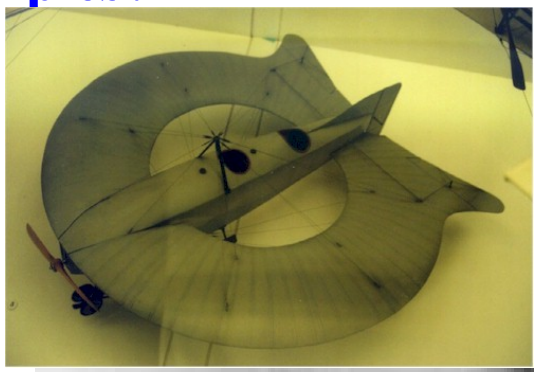
# Learning Hierarchies of Visual Features

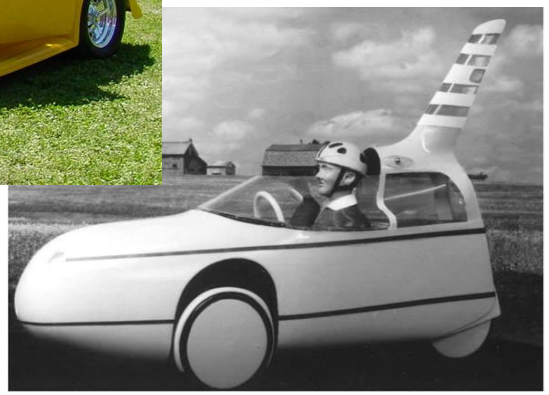
Yann LeCun
The Courant Institute of Mathematical Sciences
And Center for Neural Science
New York University

# The Next Challenge for AI, Robotics, and Neuroscience

- **How do we learn vision and perception?**
  - 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?

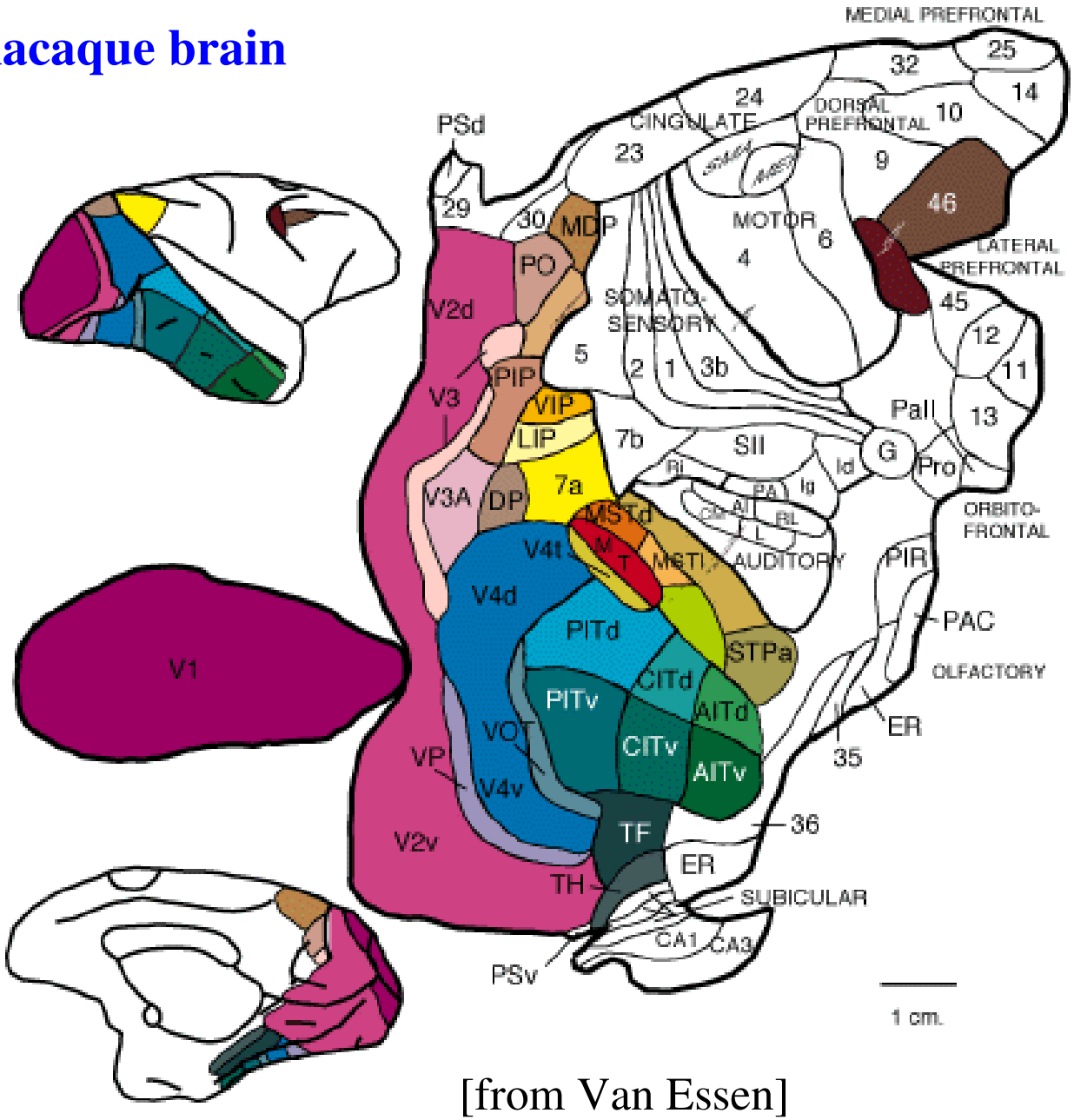
- **How can we learn visual categories from just a few examples?**
  - I don't need to see many airplanes before I can recognize every airplane (even really weird ones)





# Vision occupies a big chunk of our brains

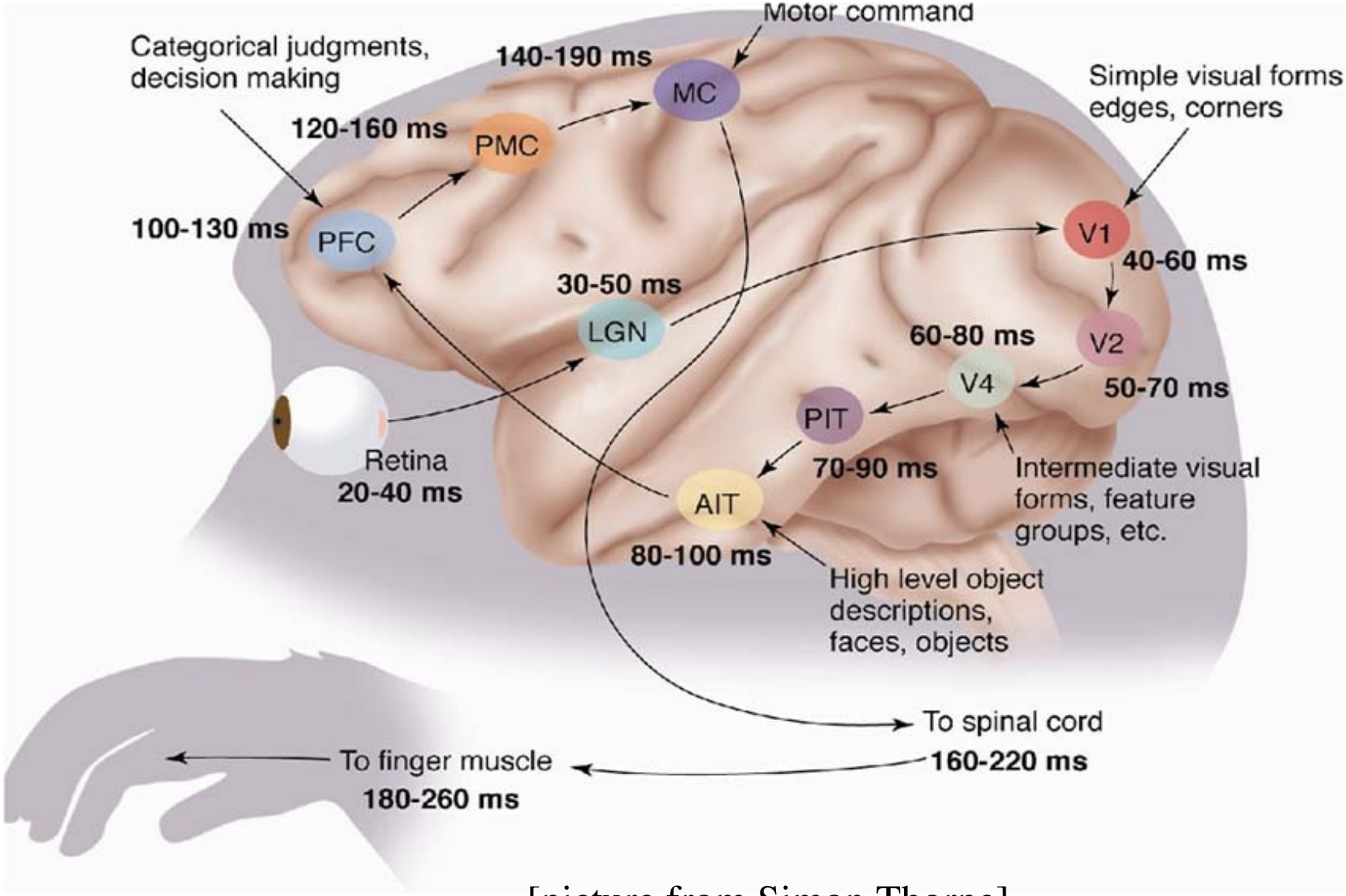
- 1/3 of the macaque brain




[from Van Essen]

# Vision is very fast and the visual cortex is hierarchical

- The ventral (recognition) pathway in the visual cortex




[picture from Simon Thorpe]

# The Primate's Visual System is Deep (LGN->V1->V2->V4->IT)

- **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 (around 10 layers) and learned**
- **what is the learning algorithm of the visual cortex?**
  - 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?

# The Broader Challenge of Machine Learning and AI

- **Can we devise learning algorithms to train a "deep" artificial visual system, and other artificial perception systems.**
- **How can we learn the structure of the world?**
  - How can we build/learn internal representations of the world that allow us to discover its hidden structure?
  - How can we learn internal representations that capture the relevant information and eliminates irrelevant variabilities?
- **How can a human or a machine learn internal representations by just looking at the world?**
- **Can we find learning methods that solve really complex problems end-to-end, such as vision, natural language, speech....?**

# The Traditional "Shallow" Architecture for Recognition

Pre-processing / Feature Extraction

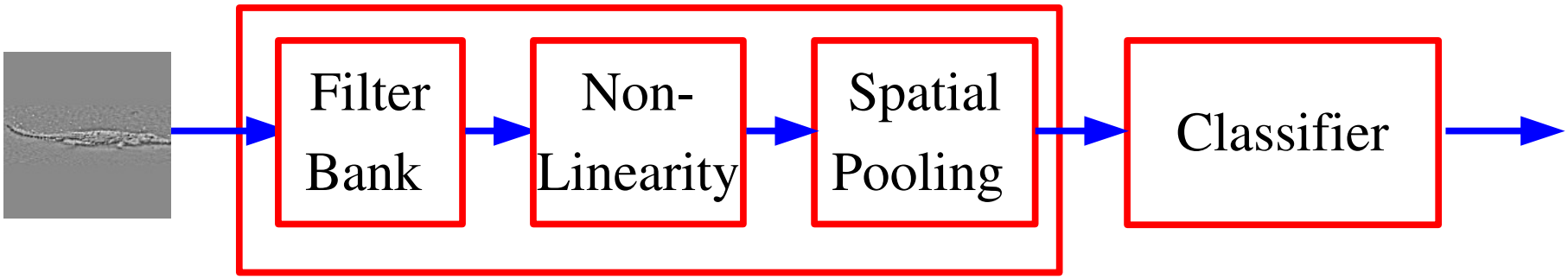
"Simple" Trainable Classifier

this part is mostly hand-crafted

Internal Representation

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

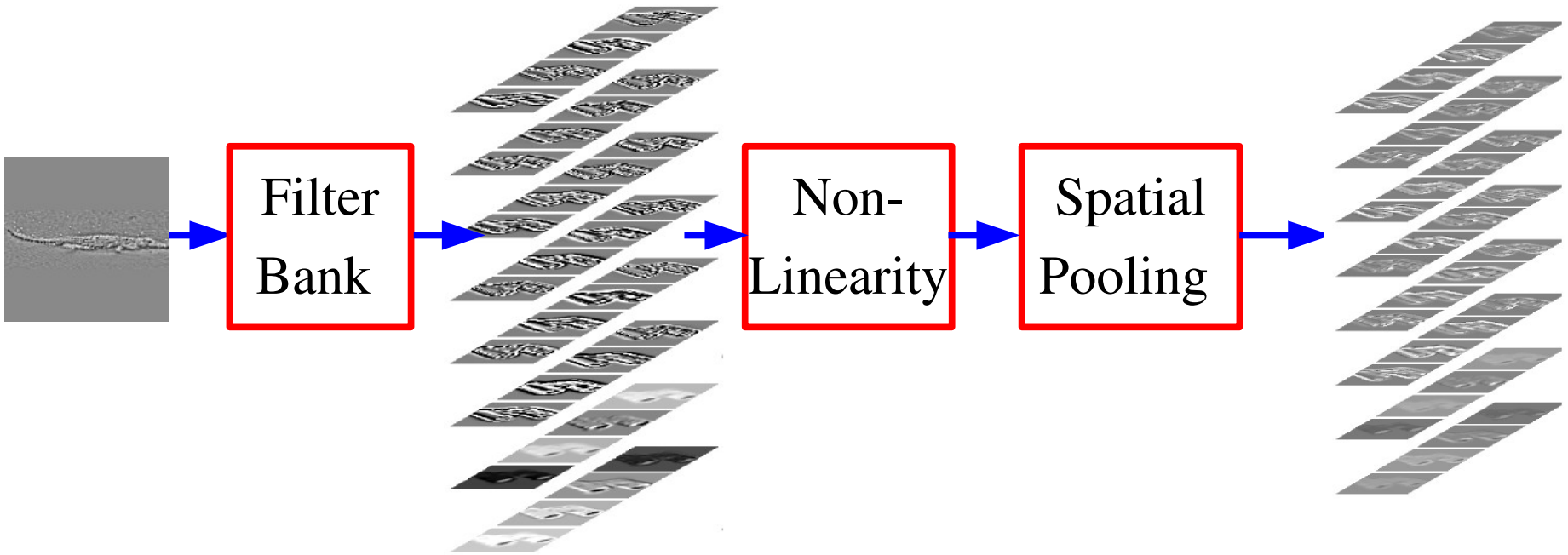
# "Modern" Object Recognition Architecture in Computer Vision




| | | |
|---|---|---|
| Oriented Edges | Sigmoid | Averaging |
| Gabor Wavelets | Rectification | Max pooling |
| Other Filters... | Vector Quant. | VQ+Histogram |
| | Contrast Norm. | Geometric Blurr |

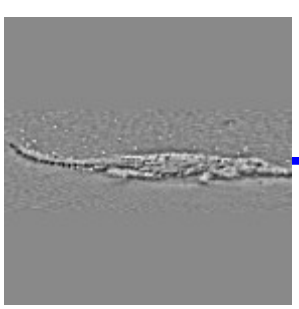
- **Example:**
  - Edges + Rectification + Histograms + SVM [Dalal & Triggs 2005]
  - SIFT + classification
- **Fixed Features + "shallow" classifier**

# Feature Extraction by Filtering and Pooling



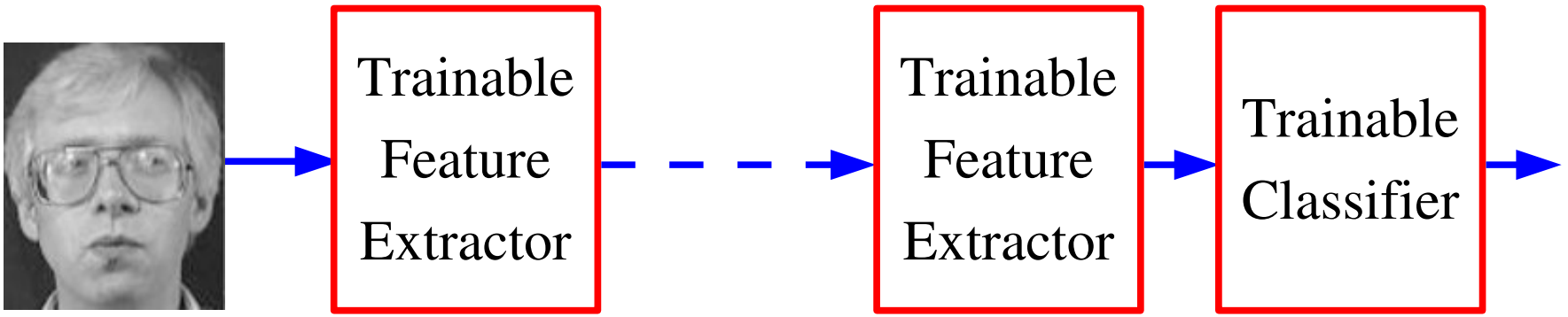

- **Biologically-inspired models of low-level feature extraction**
  - Inspired by [Hubel and Wiesel 1962]
  - Many feature extraction methods are based on this
  - SIFT, GIST, HoG, Convolutional networks.....

# "State of the Art" architecture for object recognition



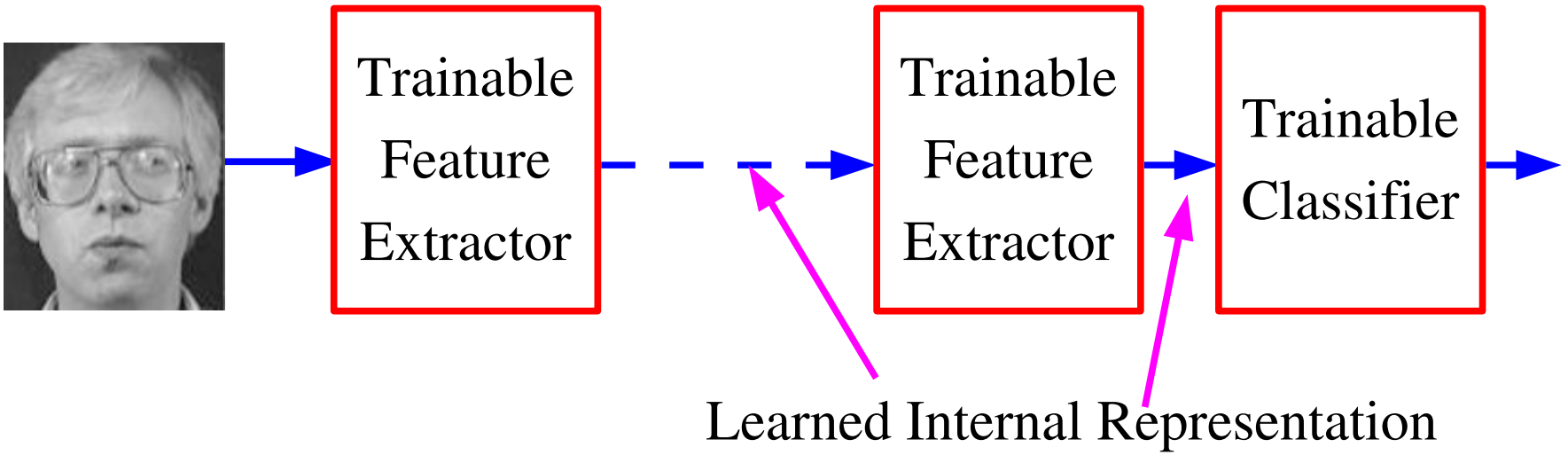

- **Example:**
  - SIFT features with Spatial Pyramid Match Kernel SVM [Lazebnik et al. 2006]
- **Fixed Features + unsupervised features + "shallow" classifier**

# 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**

# Do we really need deep architectures?

- **We can approximate any function as close as we want with shallow architecture (e.g. a kernel machine). Why would we need deep ones?**

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

  - kernel machines and 2-layer neural net are "universal".
- **Deep learning machines**

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

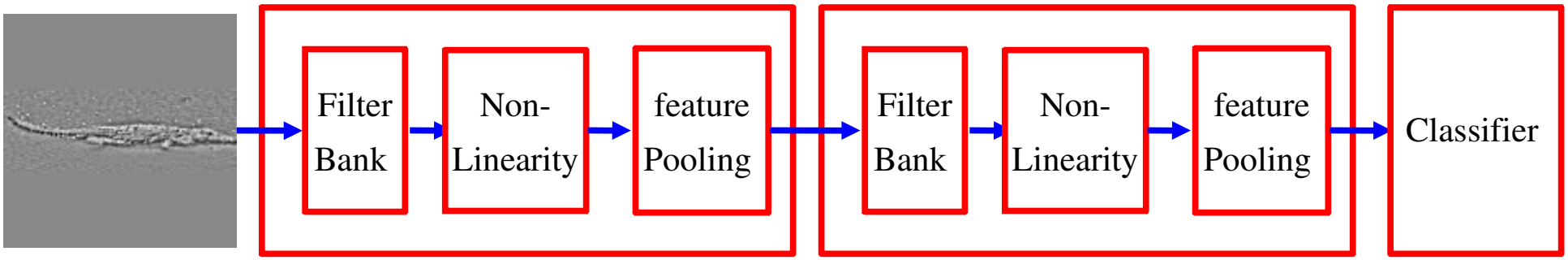
- **Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition**
  - they can represent more complex functions with less "hardware"
- **We need an efficient parameterization of the class of functions that are useful for "AI" tasks.**

# Why are Deep Architectures More Efficient?

[Bengio & LeCun 2007 "Scaling Learning Algorithms Towards AI"]

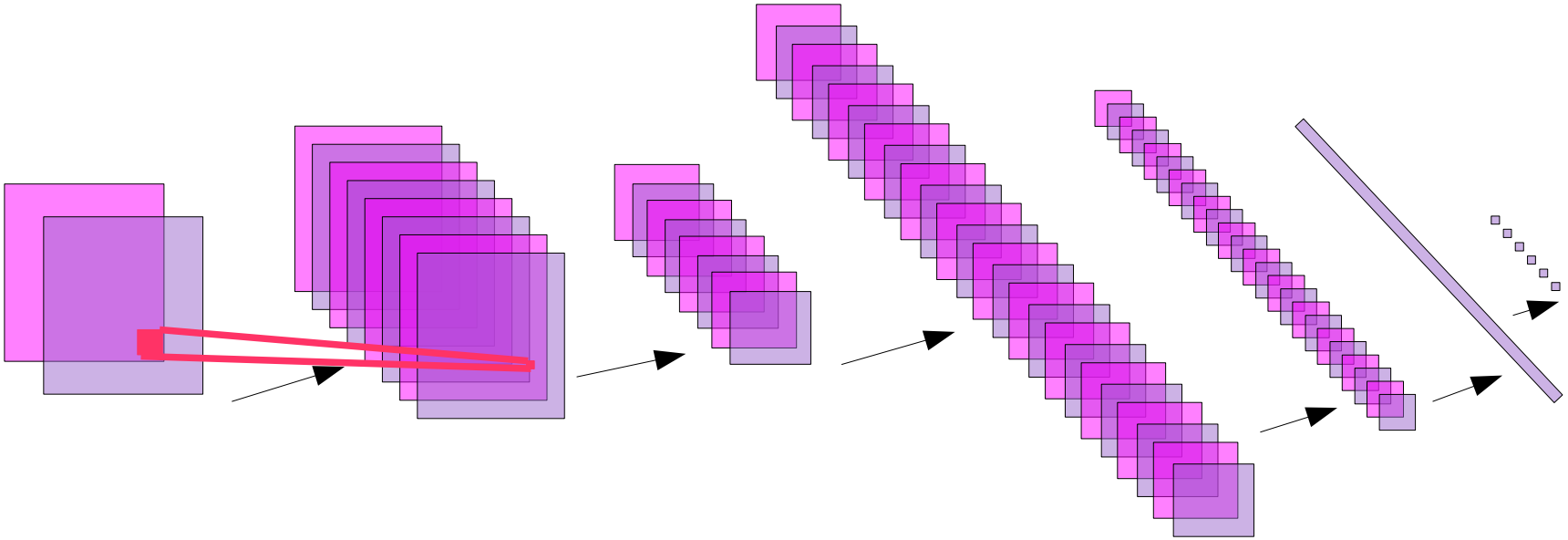
- **A deep architecture trades space for time (or breadth for depth)**
  - more layers (more sequential computation),
  - but less hardware (less parallel computation).
  - Depth-Breadth tradoff
- **Example1: N-bit parity**
  - requires N-1 XOR gates in a tree of depth log(N).
  - requires an exponential number of gates 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).....

# Can't we train multi-stage vision architectures?




- Stacking multiple stages of feature extraction/pooling.
- Creates a hierarchy of features

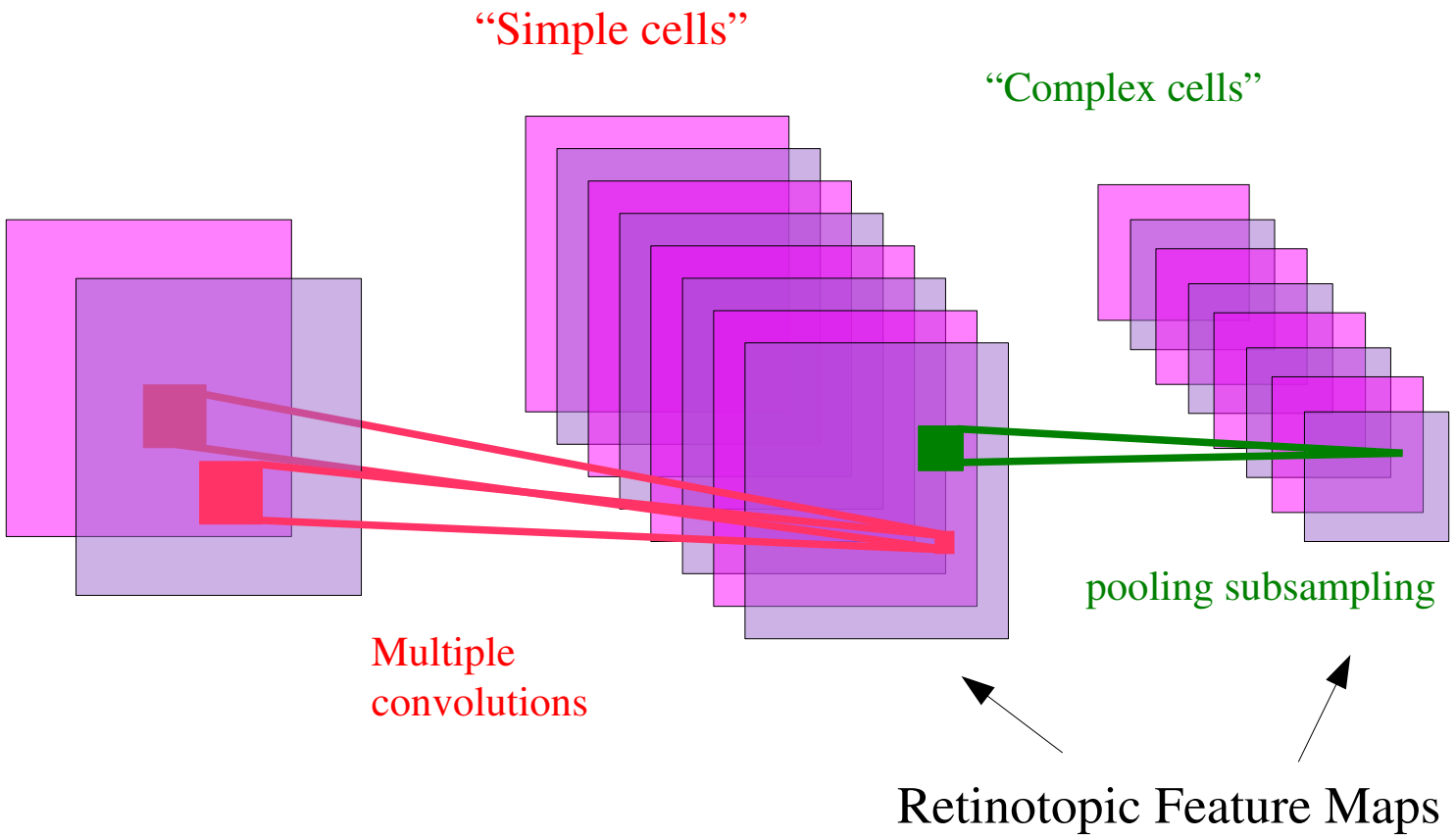
# Convolutional Network



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

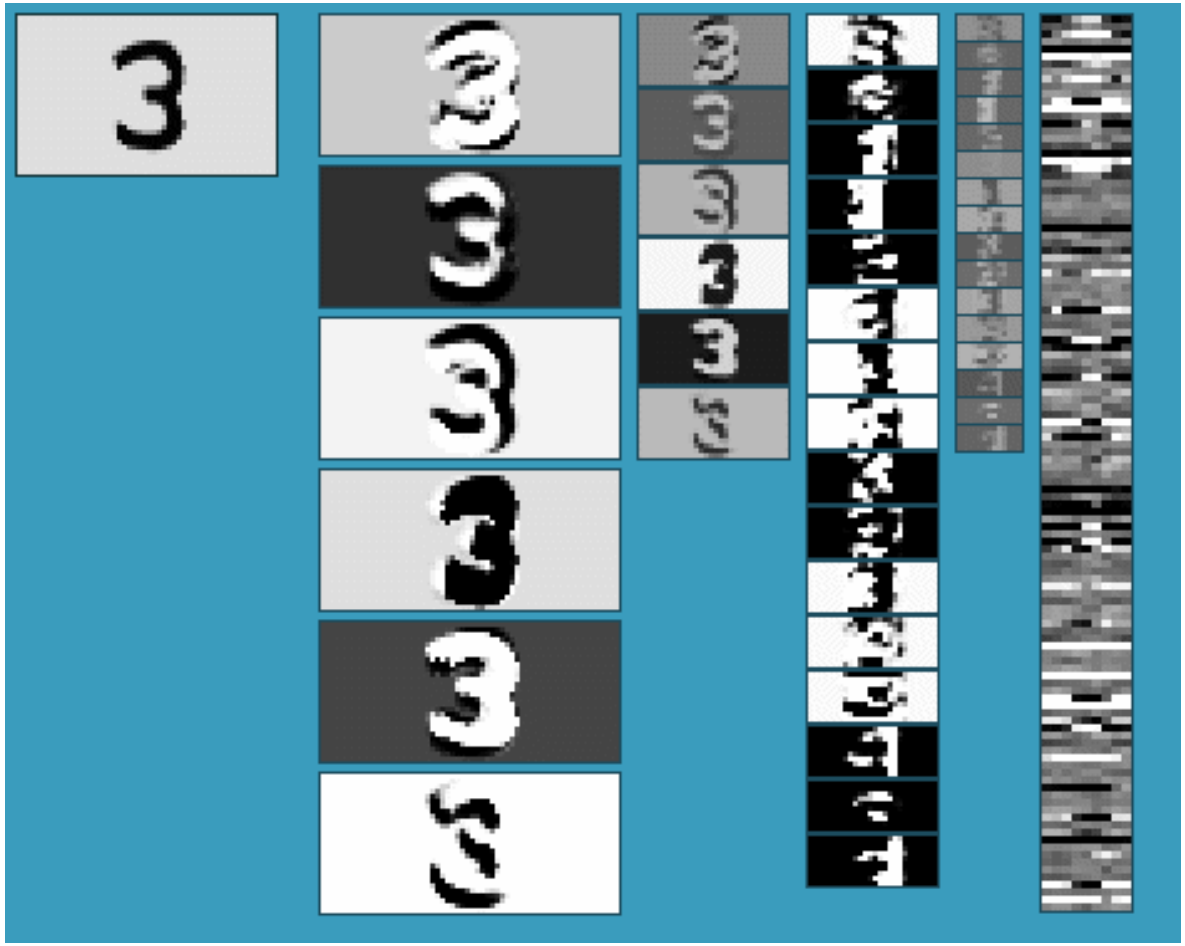
# An Old Idea for Local Shift Invariance

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

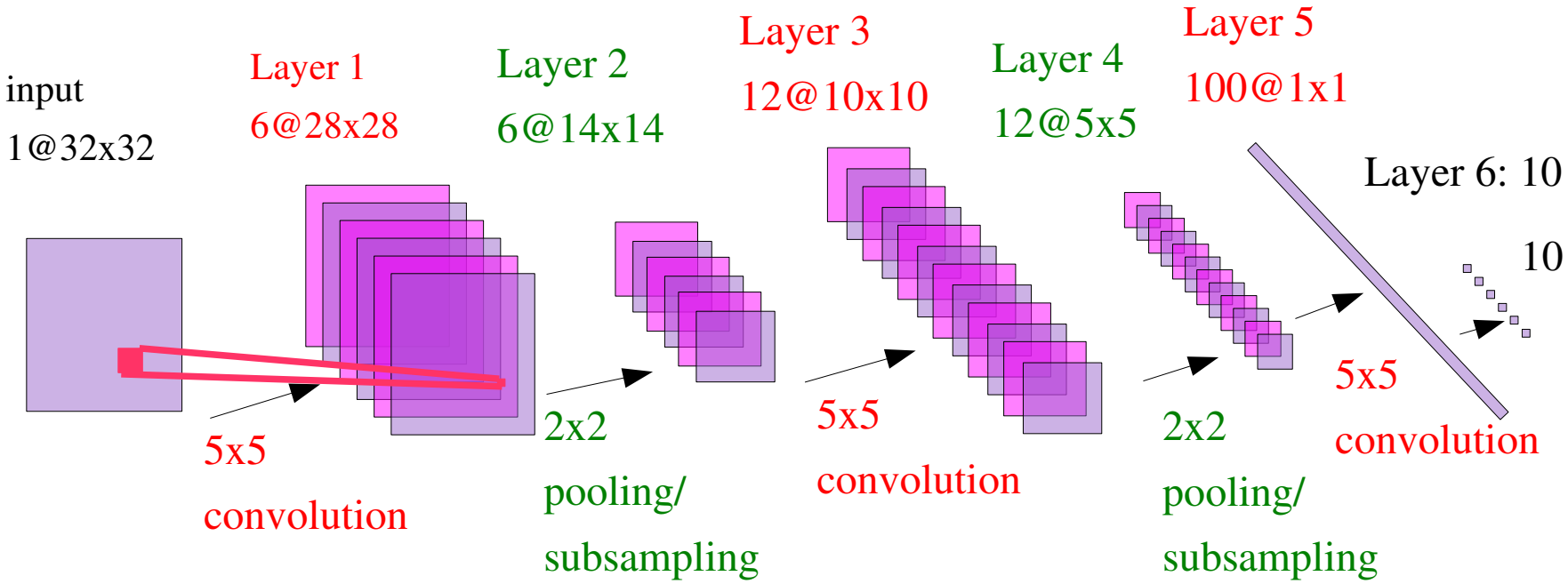


# The Multistage Hubel-Wiesel Architecture

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



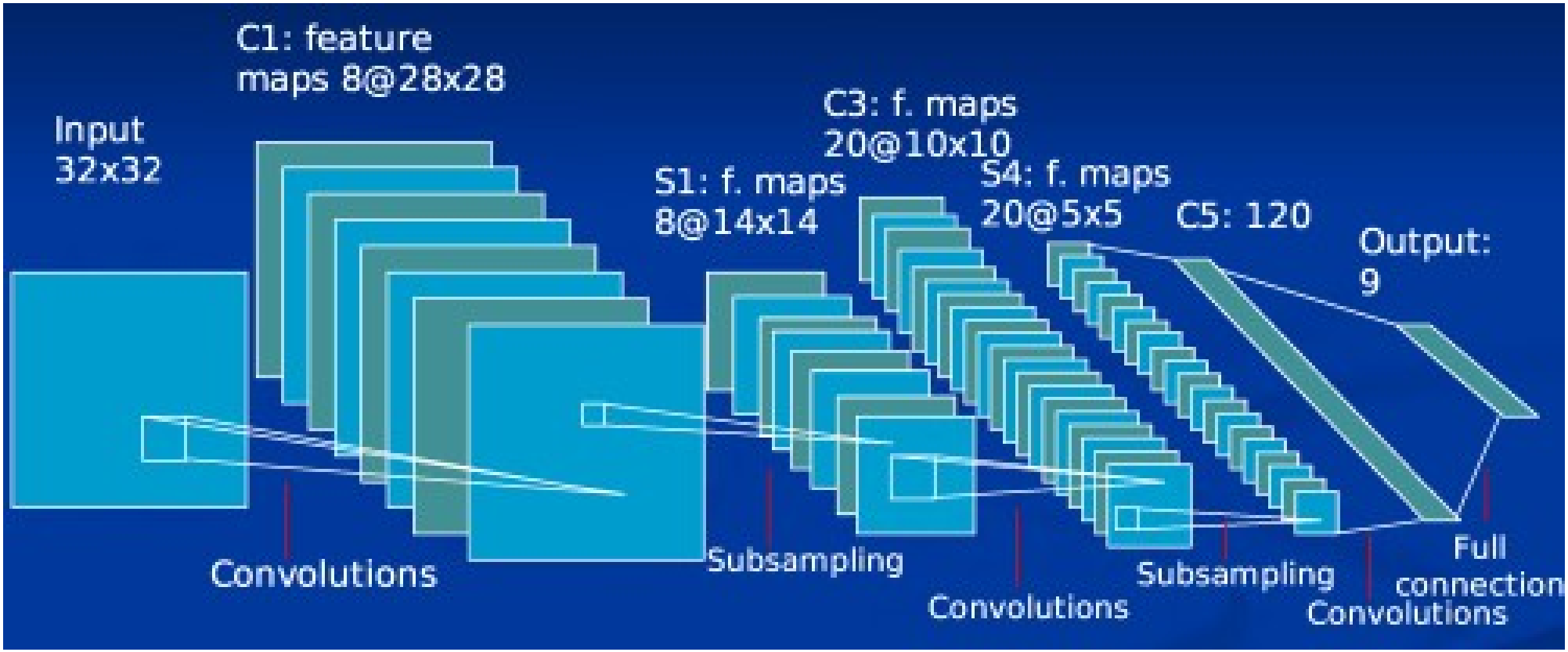
# Convolutional Net Architecture




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

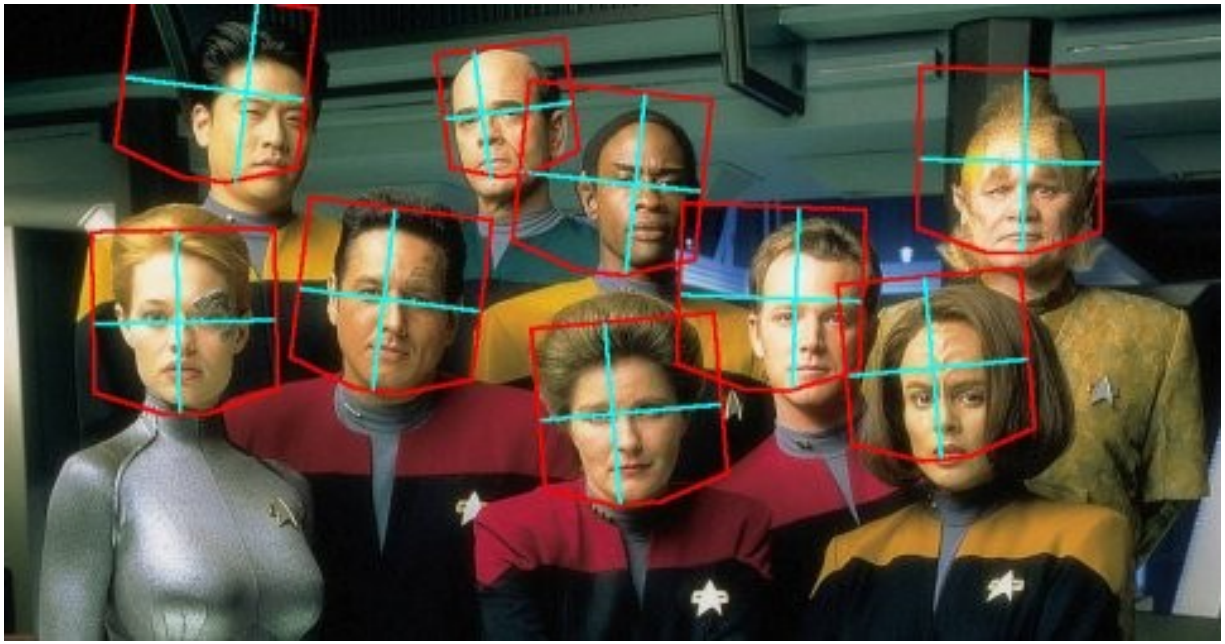
# Face Detection and Pose Estimation with Convolutional Nets

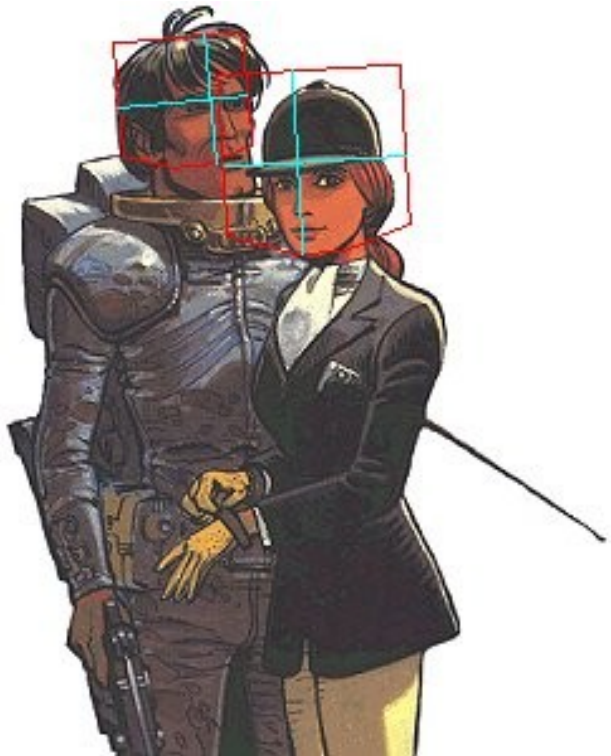
- **Training:** 52,850, 32x32 grey-level images of faces, 52,850 non-faces.
- **Each sample:** used 5 times with random variation in scale, in-plane rotation, brightness and contrast.
- **2nd phase:** half of the initial negative set was replaced by false positives of the initial version of the detector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



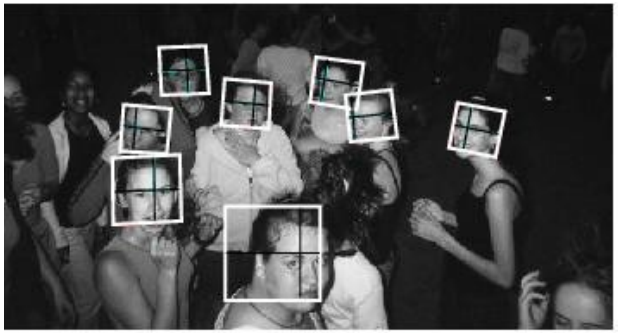
# Face Detection: Results

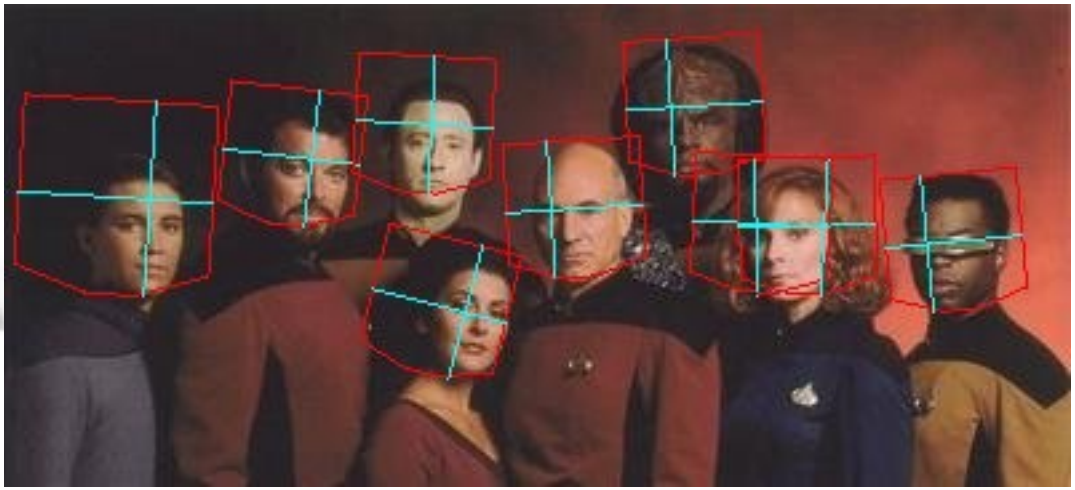
| *Data Set->* | TILTED | | PROFILE | | MIT+CMU | |
|---|---|---|---|---|---|---|
| *False positives per image->* | 4.42 | 26.9 | 0.47 | 3.36 | 0.5 | 1.28 |
| Our Detector | 90% | 97% | 67% | 83% | 83% | 88% |
| Jones & Viola (tilted) | 90% | 95% | x | | x | |
| Jones & Viola (profile) | x | | 70% |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

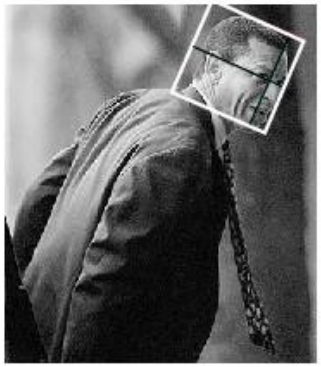


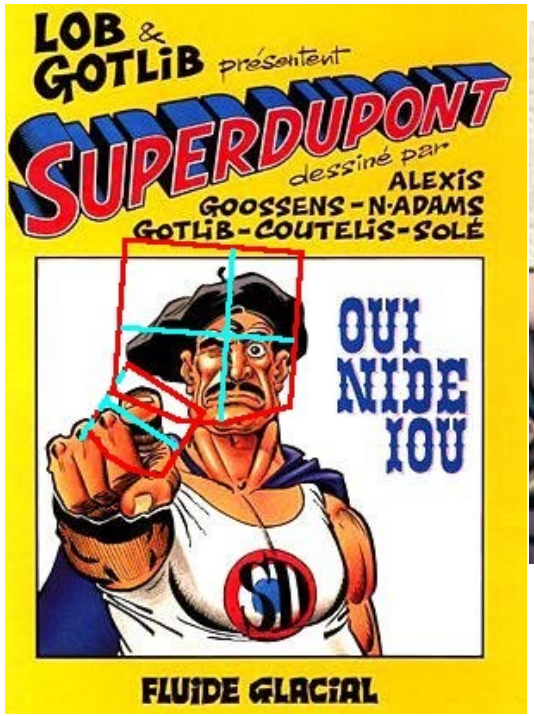


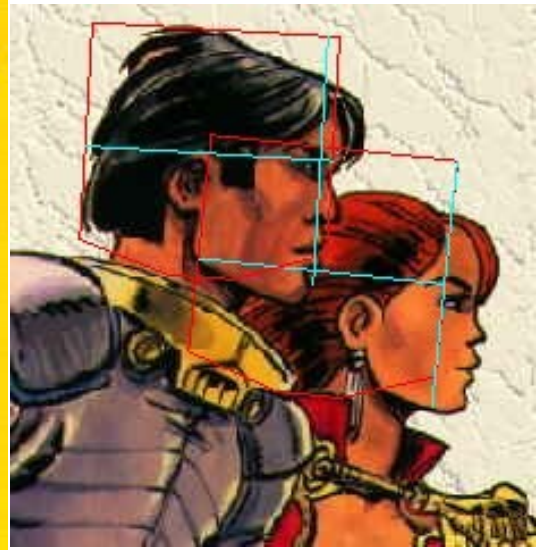
# Face Detection and Pose Estimation: Results





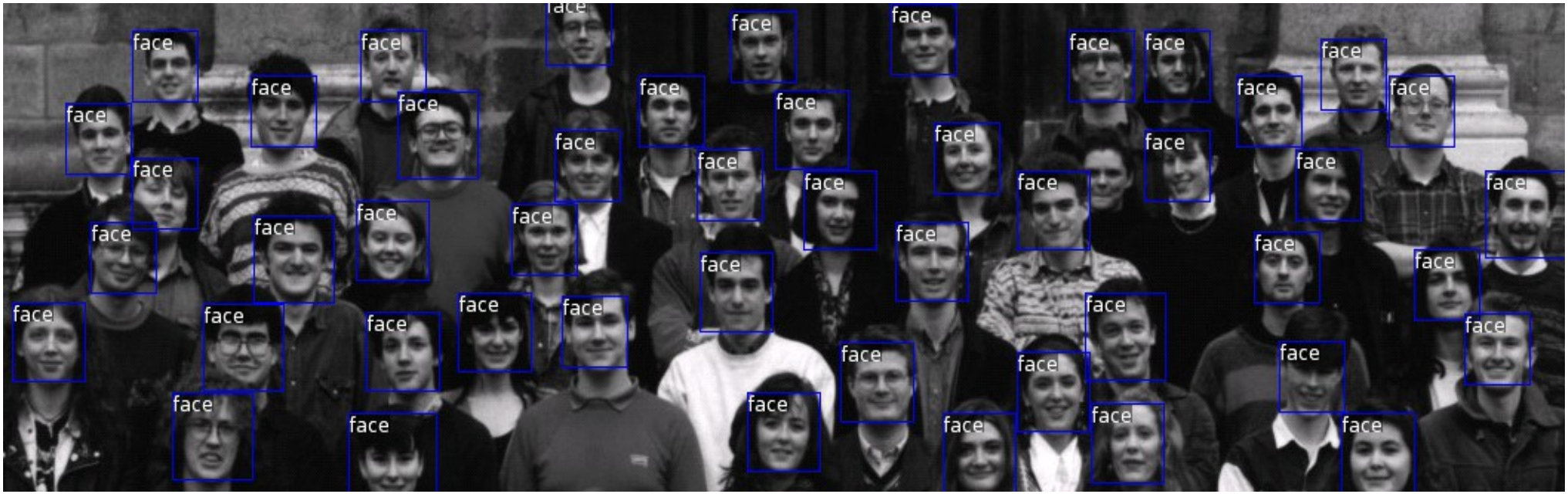




# Face Detection with a Convolutional Net

# Face Detection with a ConvNet

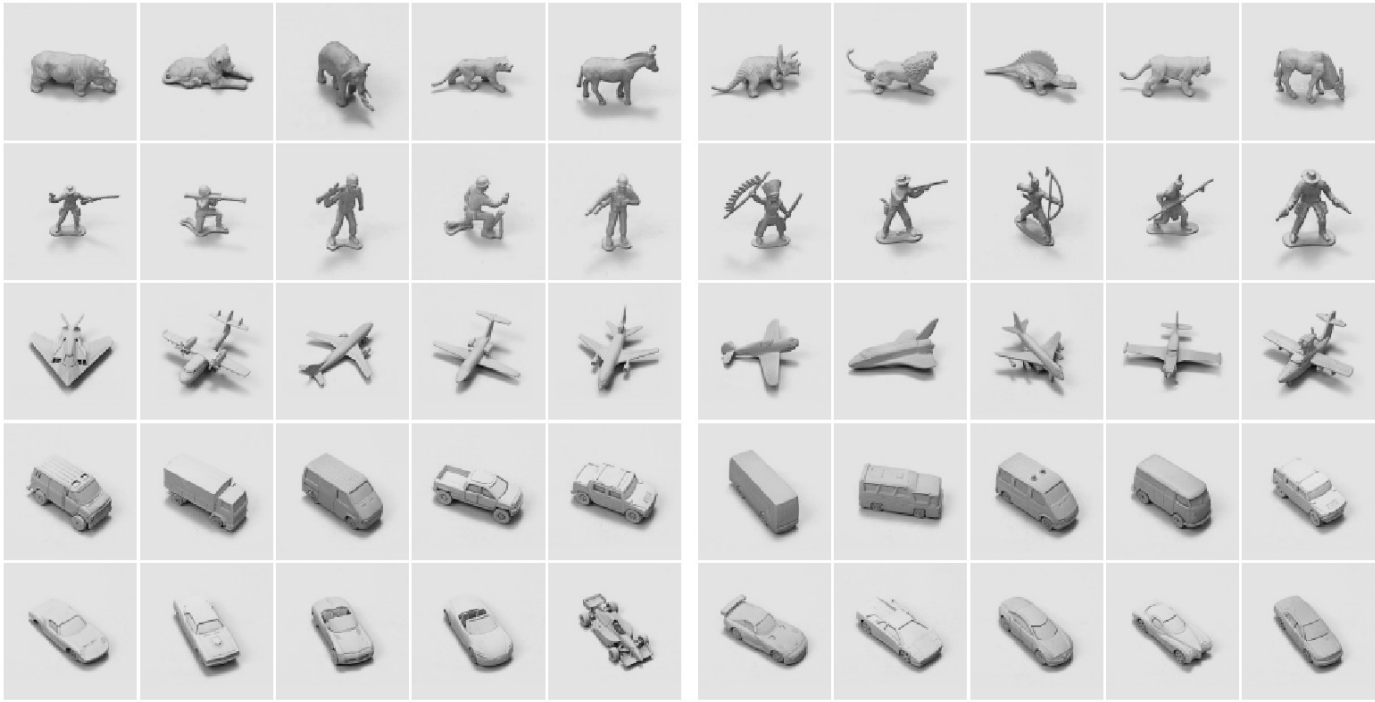



- Demo produced with EBLearn open source package
- http://eblearn.sf.net

# Generic Object Detection and Recognition with Invariance to Pose and Illumination

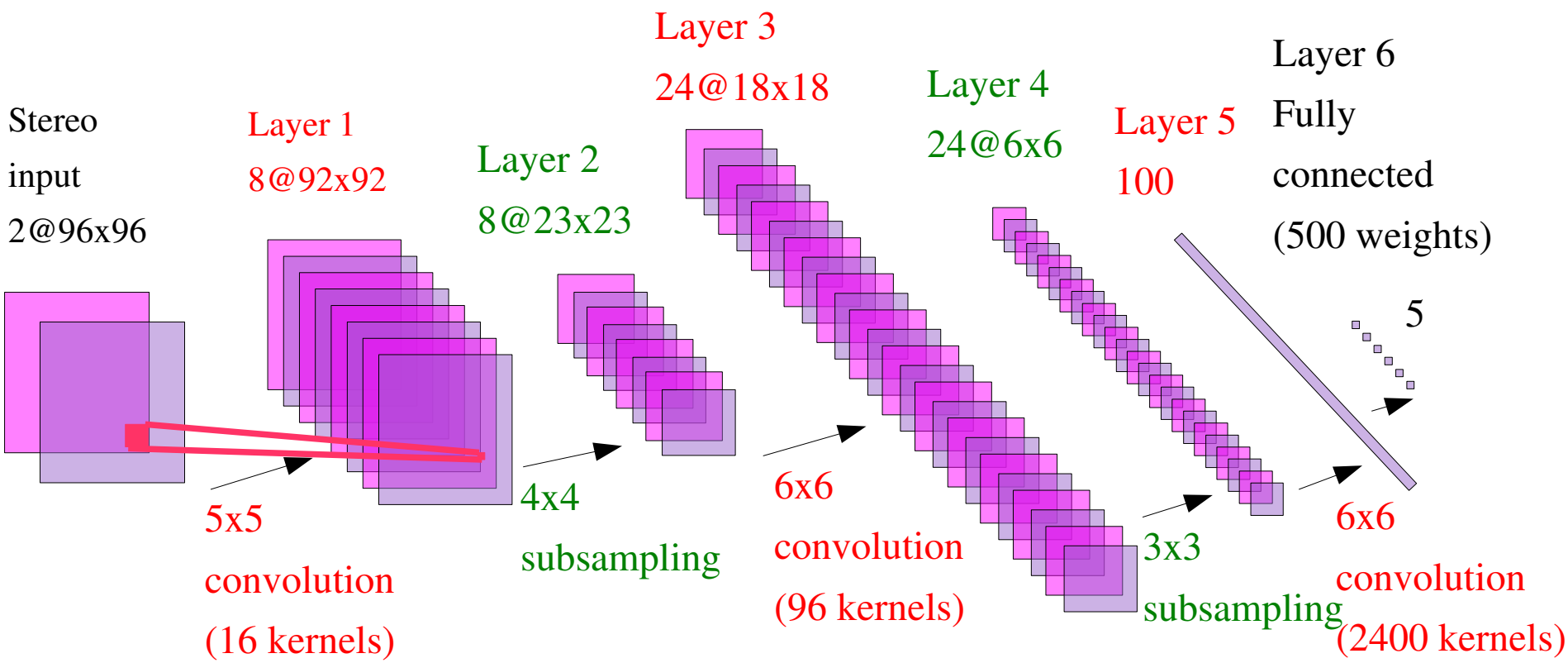
- **50** toys belonging to 5 categories: **animal, human figure, airplane, truck, car**
- **10** instance per category: 5 instances used for training, 5 instances for testing
- **Raw dataset: 972** stereo pair of each object instance. **48,600** image pairs total.

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



**Training instances** **Test instances**

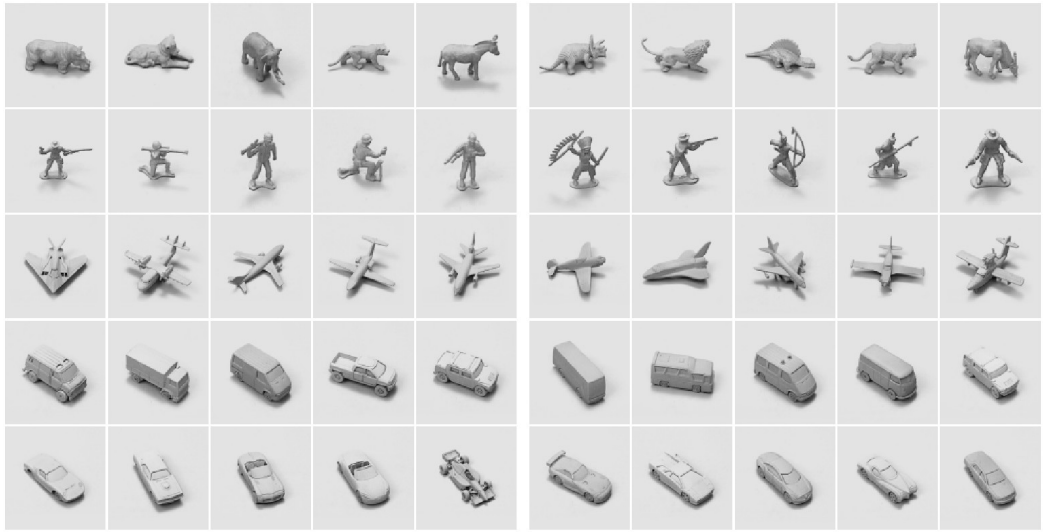
# Convolutional Network




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

# Normalized-Uniform Set: Error Rates

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



Training instances Test instances

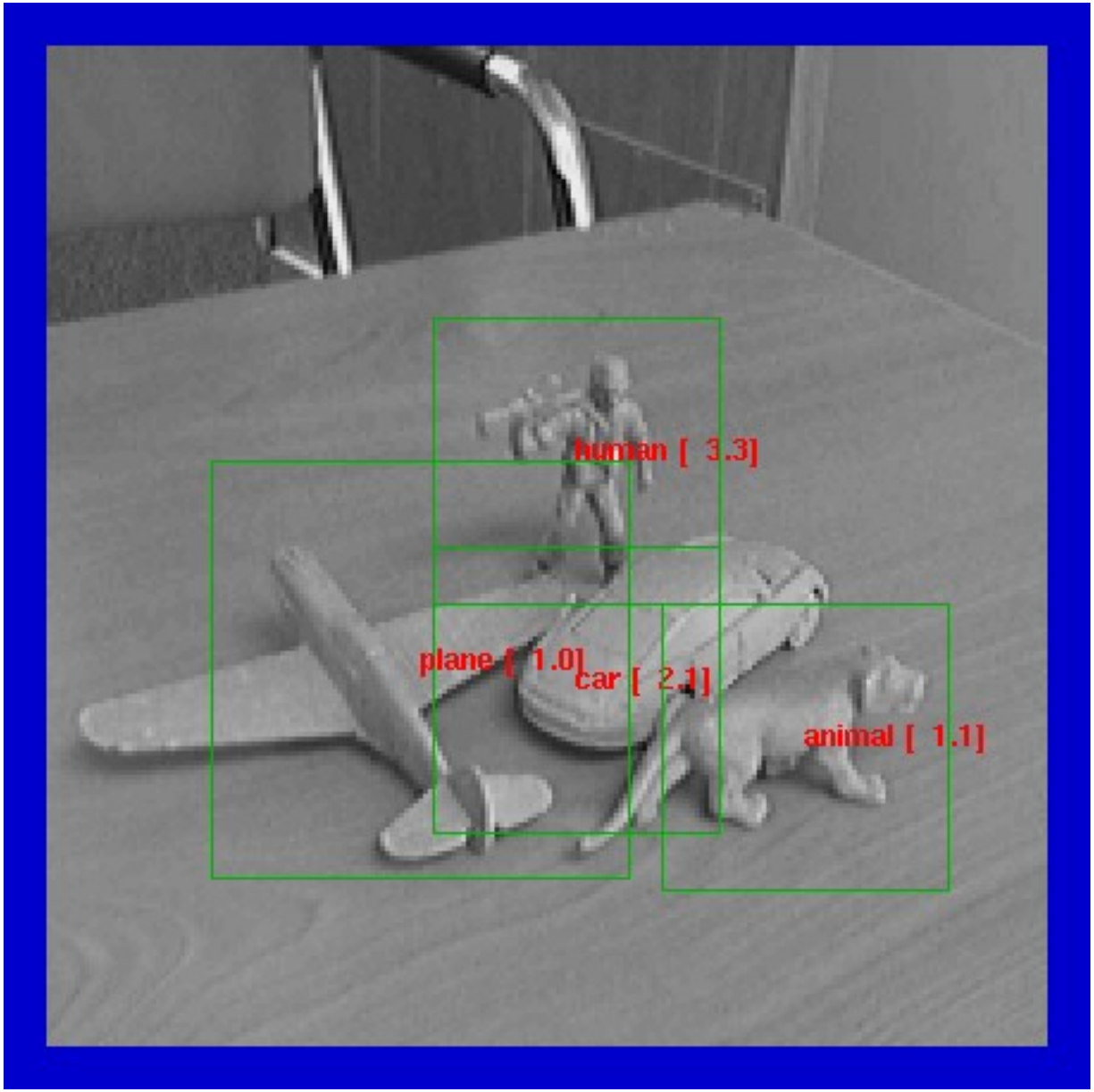
# Jittered-Cluttered Dataset

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

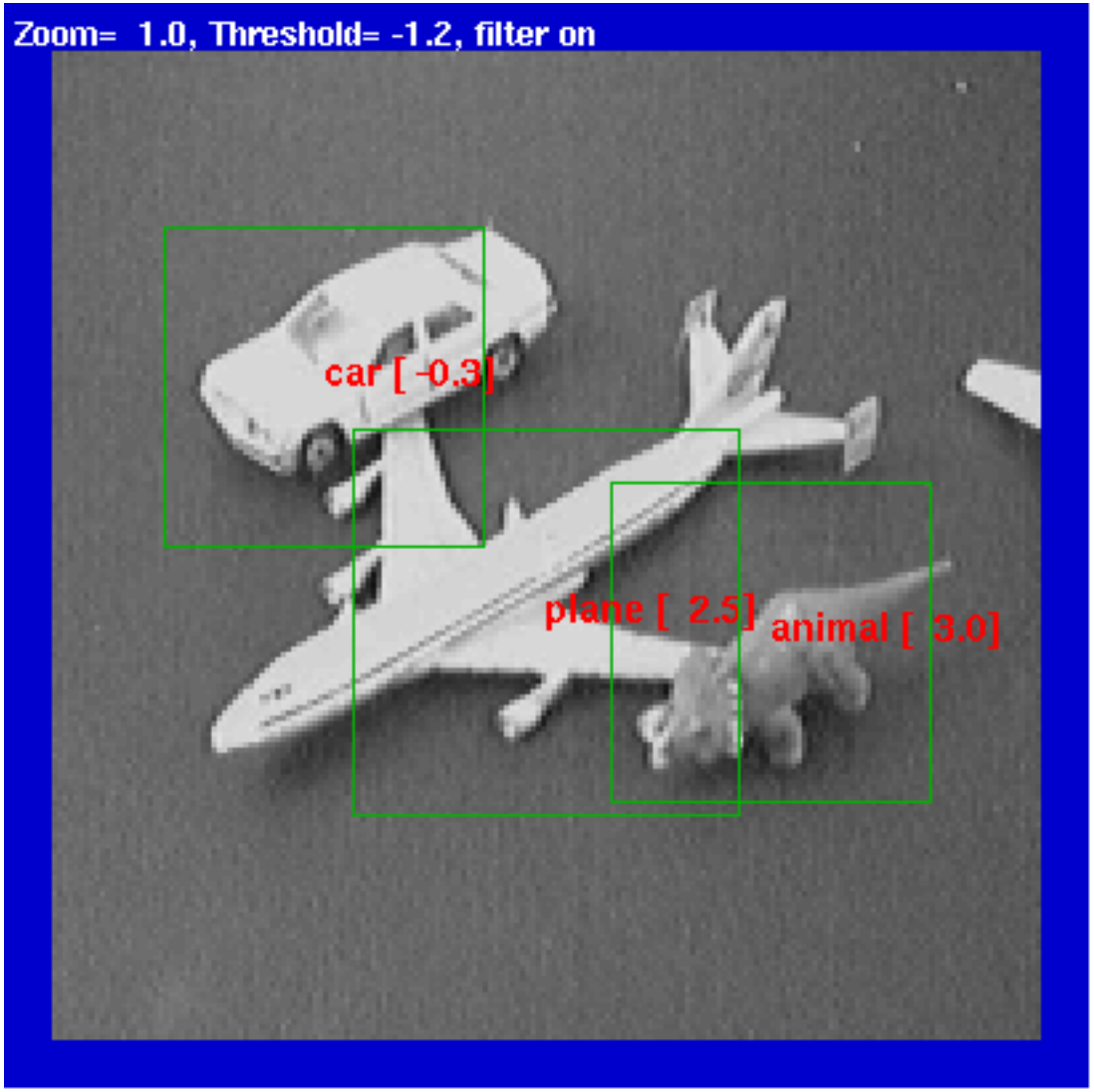
# Experiment 2: Jittered-Cluttered Dataset

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

# Examples (Monocular Mode)

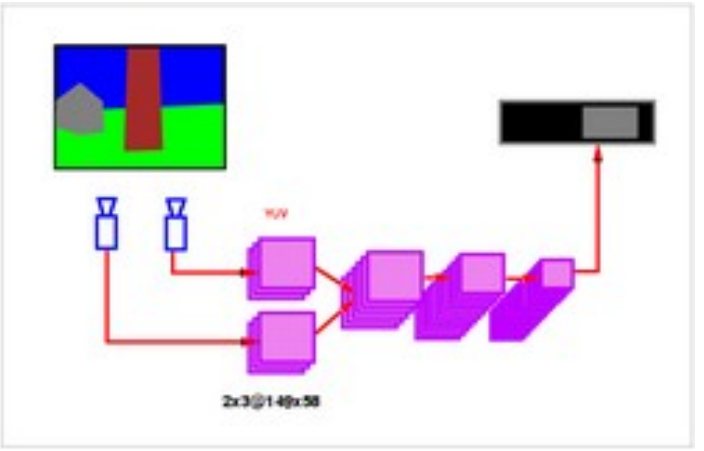


# Examples (Monocular Mode)

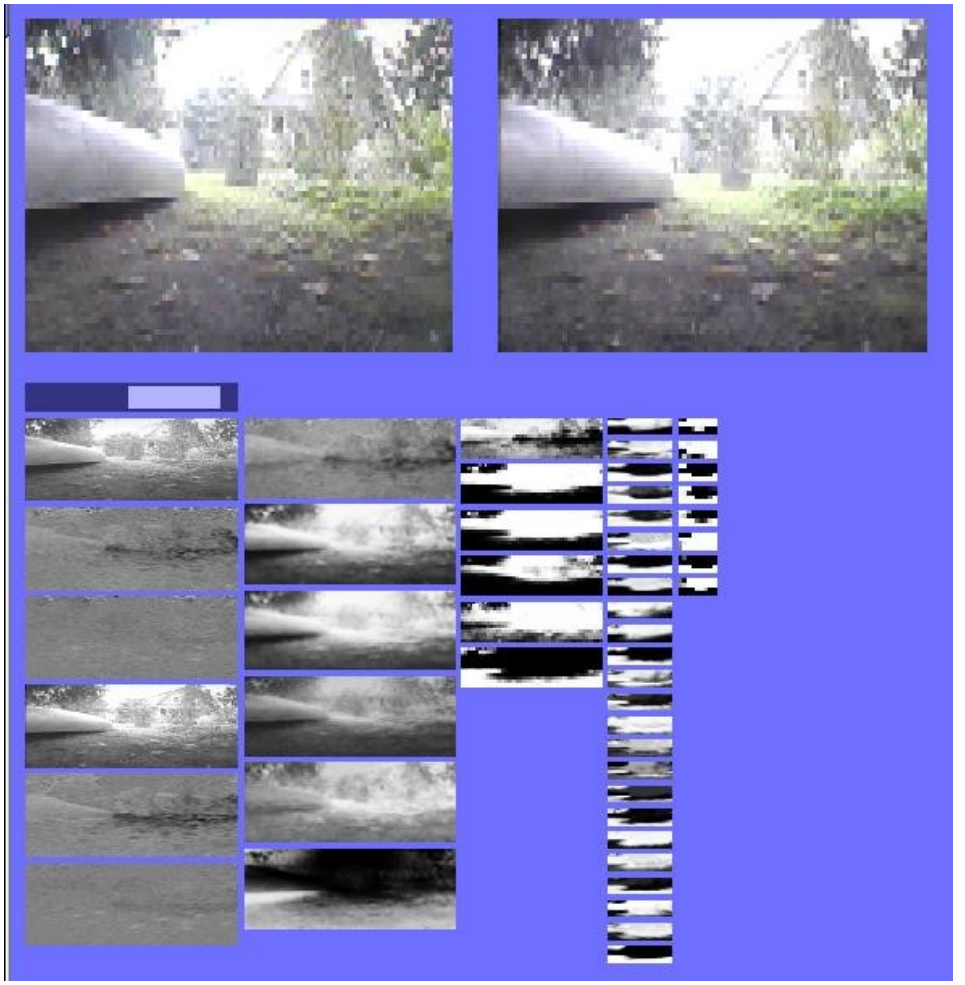


# Visual Navigation for a Mobile Robot

[LeCun et al. NIPS 2005]

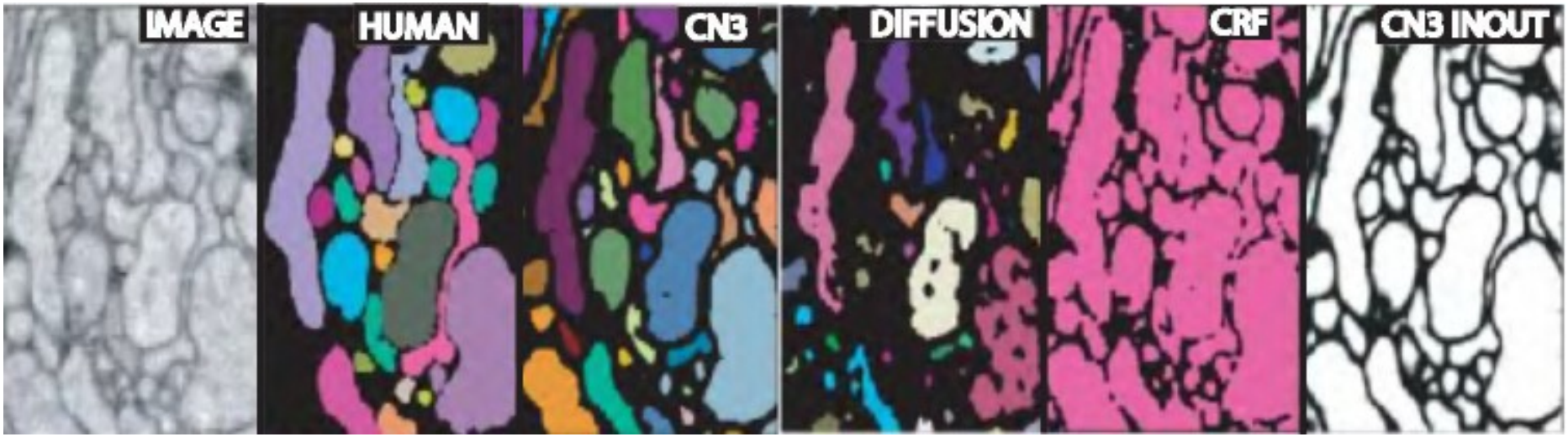


- Mobile robot with two cameras
- The convolutional net is trained to emulate a human driver from recorded sequences of video + human-provided steering angles.
- The network maps stereo images to steering angles for obstacle avoidance



# Convolutional Nets For Brain Imaging and Biology

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

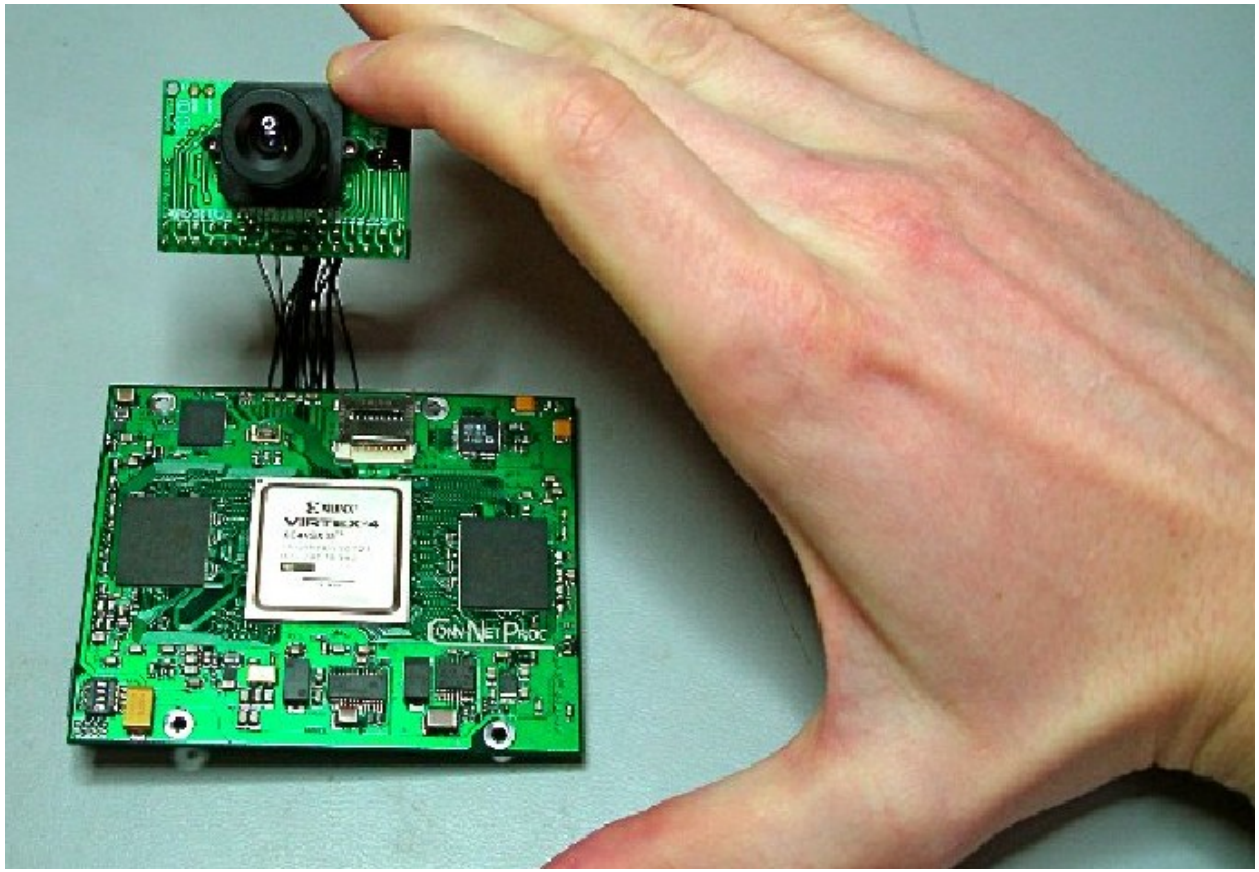


# Industrial Applications of ConvNets

- **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, face and license plate removal from StreetView
- **Microsoft**
  - OCR, handwriting recognition, speech detection
- **France Telecom**
  - Face detection, HCI, cell phone-based applications
- **Other projects: HRL (3D vision)....**

# FPGA Custom Board: NYU ConvNet Processor

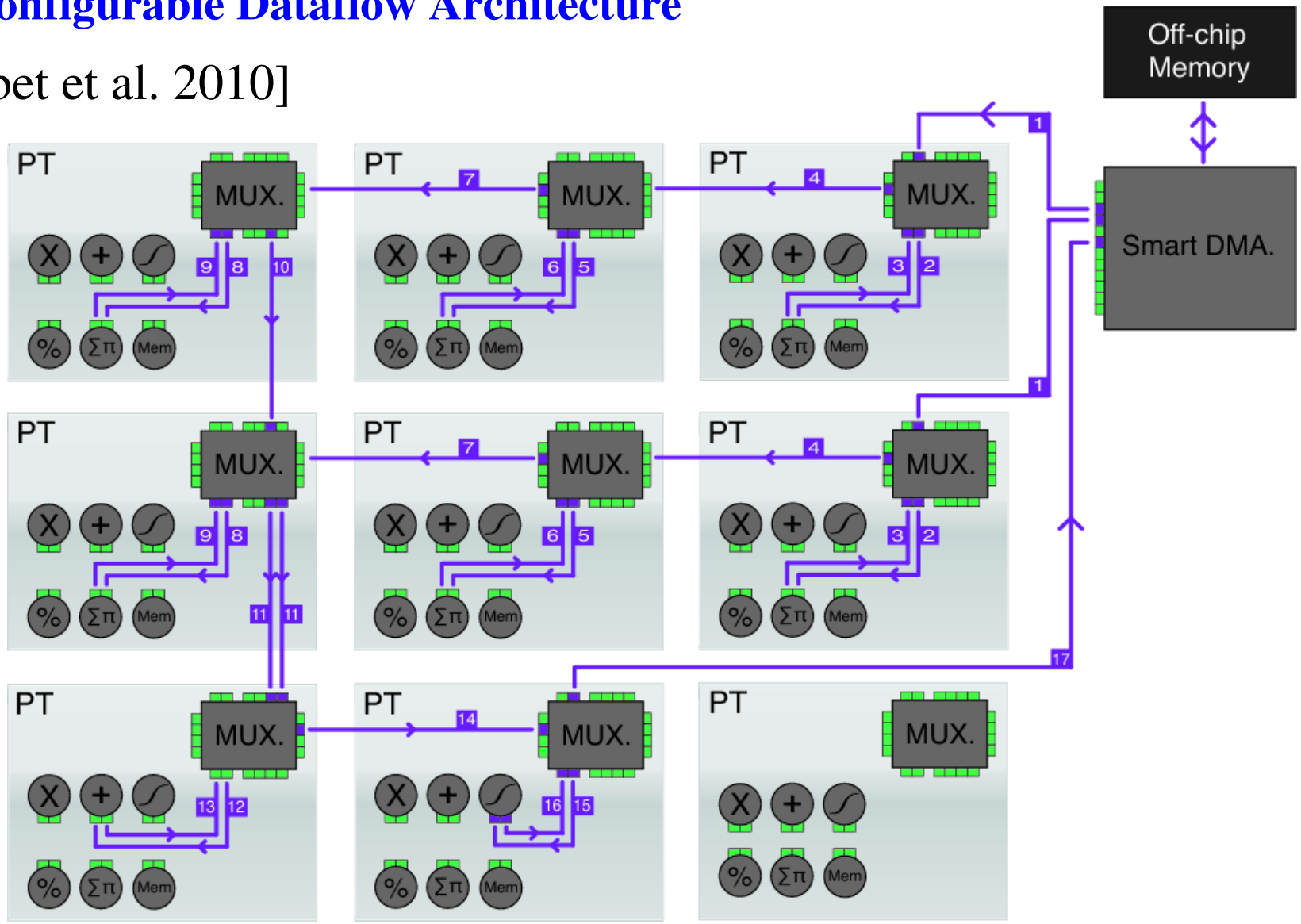
- **Xilinx Virtex 4 FPGA, 8x5 cm board** [Farabet et al. 2009]
  - Dual camera port, Fast dual QDR RAM,
- **New version being developed with Eugenio Culurciello (Yale EE)**
  - Full custom chip
  - Version for Virtex 6 FPGA



# ConvNet/Vision Processor (FPGA and ASIC)

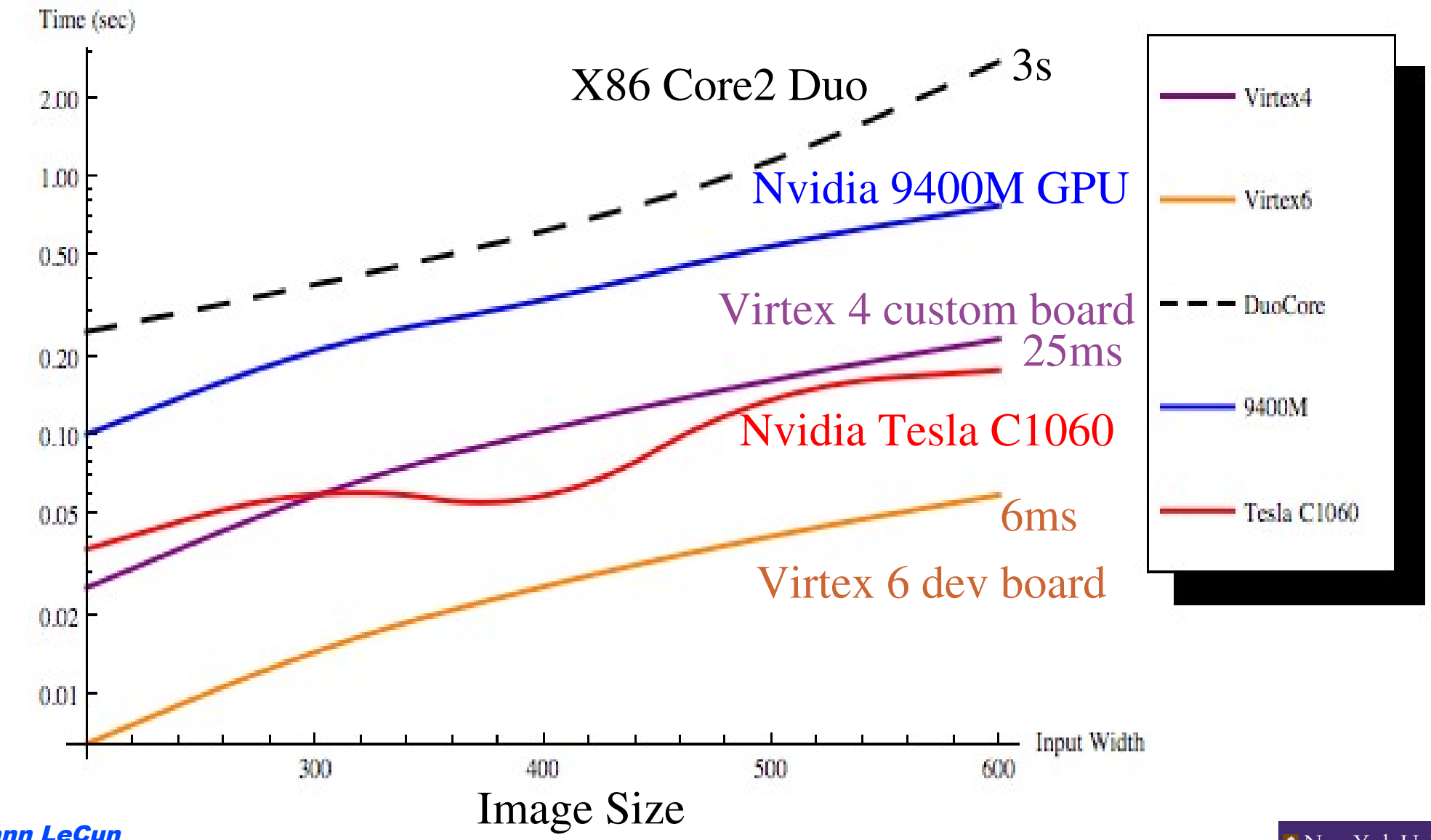
## Reconfigurable Dataflow Architecture

[Farabet et al. 2010]



# FPGA Performance

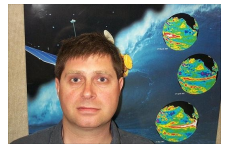
- **Seconds per frame for a robot vision task (log scale)** [Farabet et al. 2010]



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

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

*face*



*beaver*

*wild cat*

*lotus*

*ant*

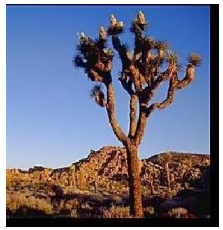
*dollar*

*w. chair*

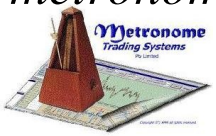
*minaret*

*cellphone*

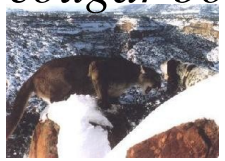
*joshua t.*



*metronome*




*cougar body*



*background*

# Unsupervised Deep Learning: Leveraging Unlabeled Data

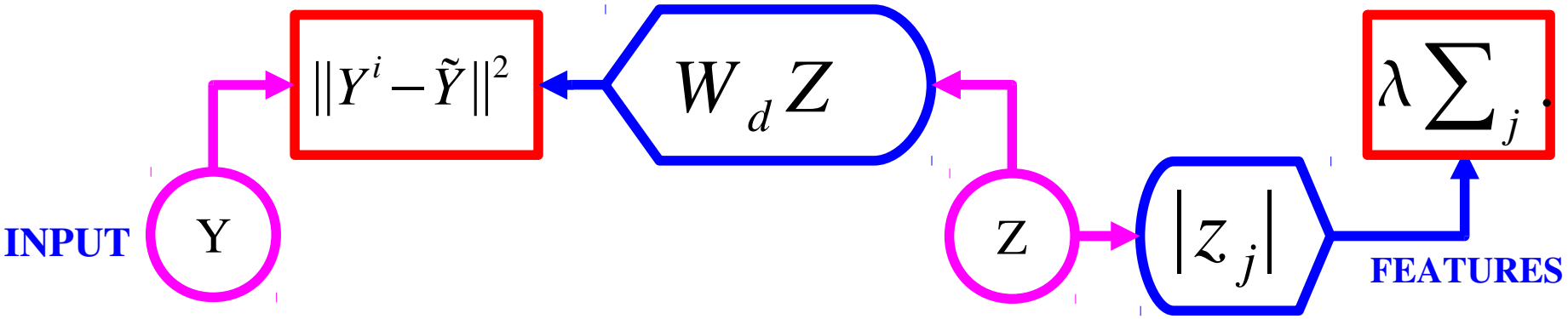
[Hinton 05, Bengio 06, LeCun 06, Ng 07]

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

# Unsupervised Feature Learning with Sparse Coding

[Olshausen & field 1997]

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



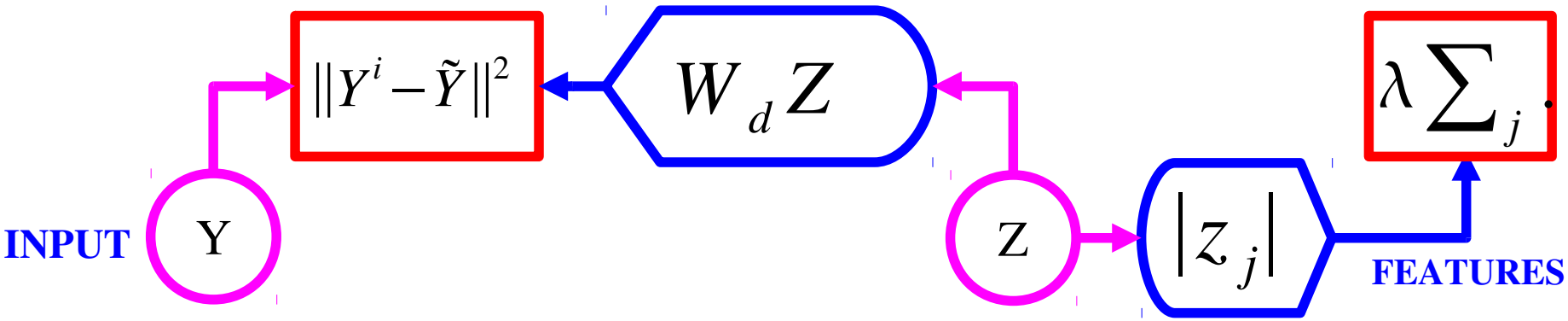

- **Energy:** $E(Y^i, Z; W_d) = \|Y^i - W_d Z\|^2 + \lambda \sum_j |z_j|$
- **Optimal Code** $Z^i = argmin_z E(Y^i, z; W_d)$
- **Free Energy:** $F(Y^i; W_d) = F(Z^i) = min_z E(Y^i, z; W_d)$

# Unsupervised Feature Learning with Sparse Coding

- The learning algorithm minimizes the loss function:

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

- The columns of Wd are normalized




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

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

# Problem with Sparse Coding: Inference is slow

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

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

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

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

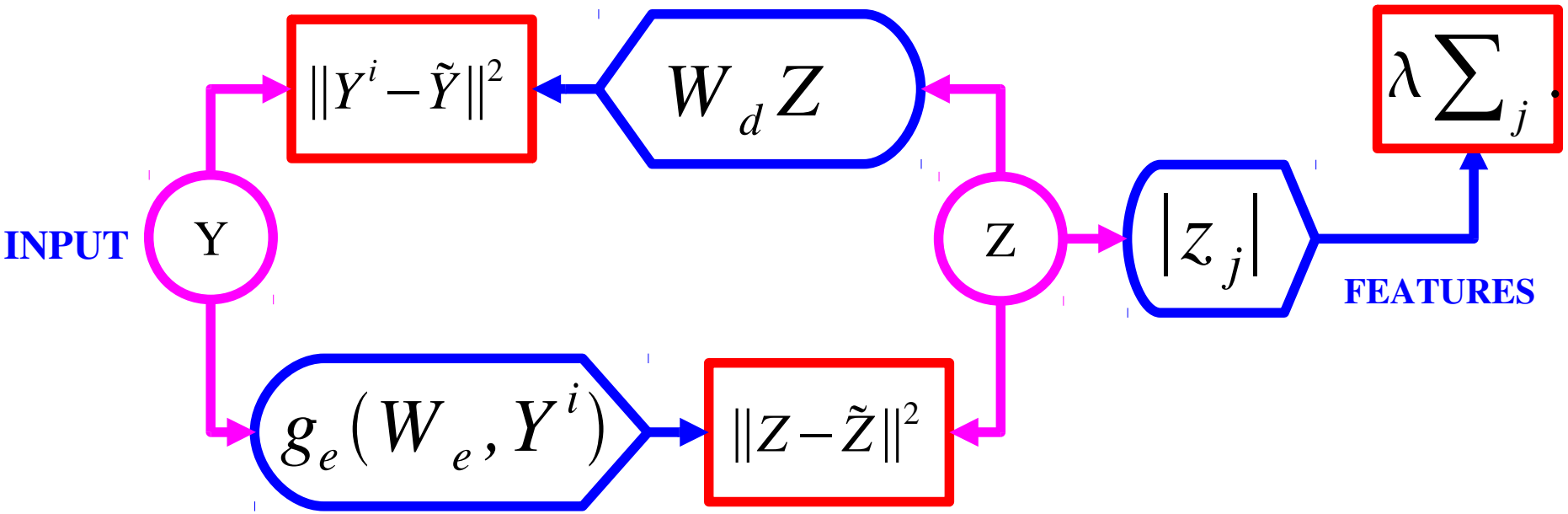
# Solution: Predictive Sparse Decomposition (PSD)

[Kavukcuoglu, Ranzato, LeCun, 2009]

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

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

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

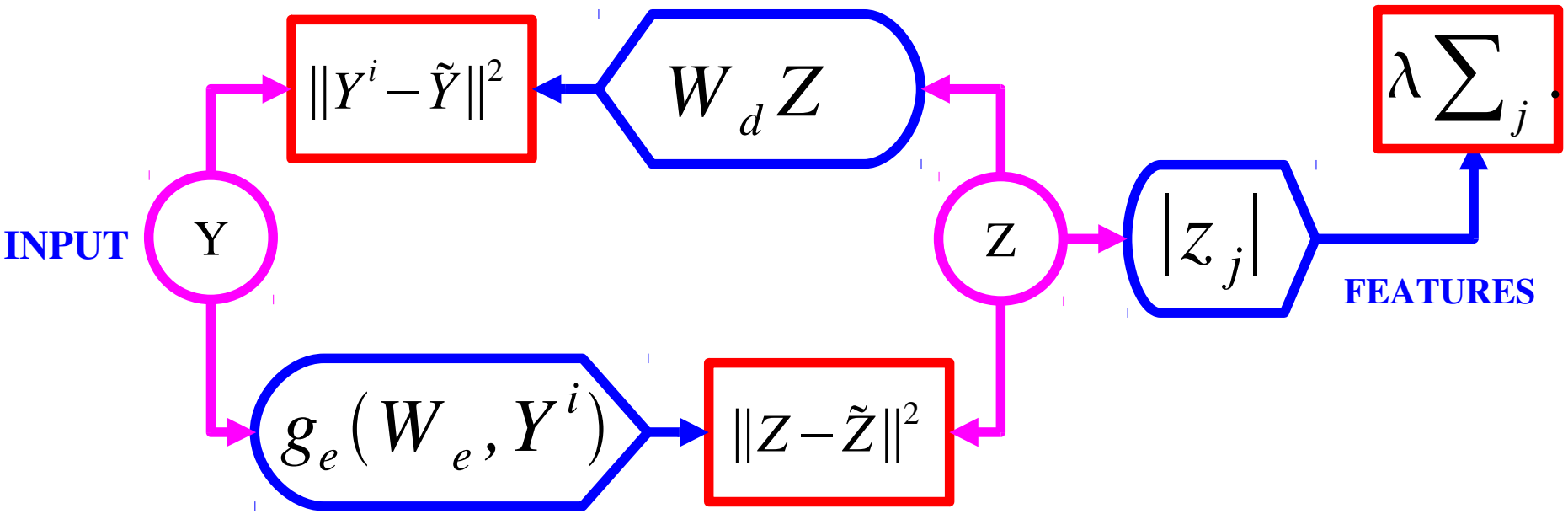


# PSD: Inference

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

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

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

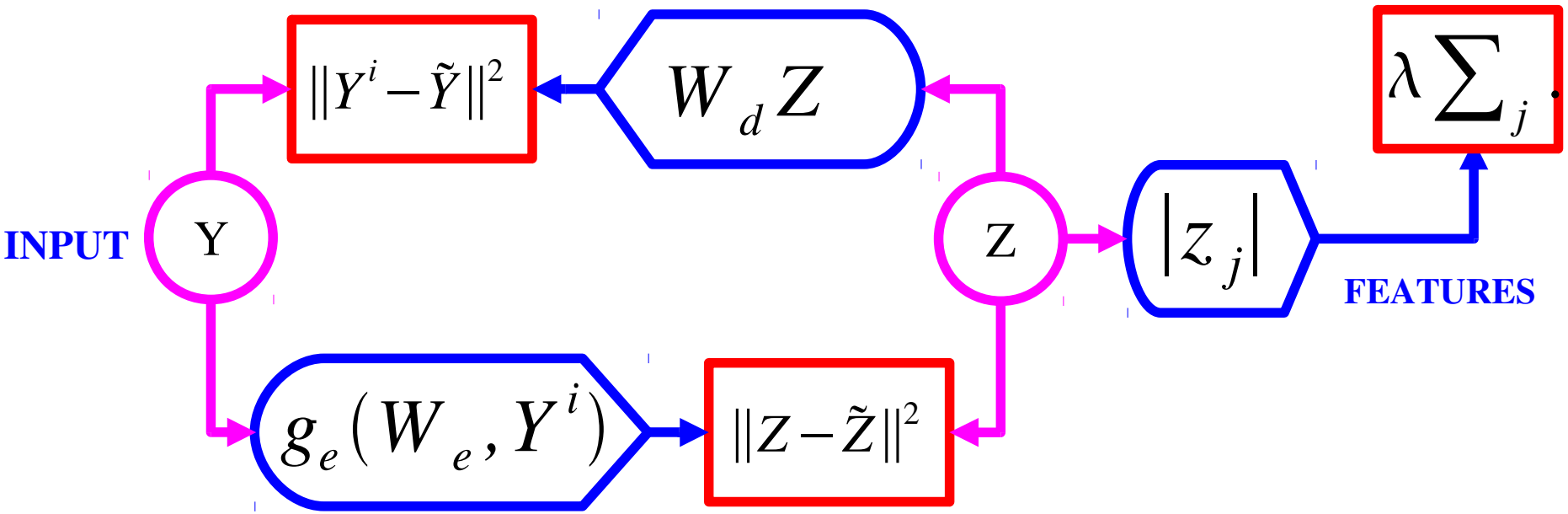


# PSD: Learning [Kavukcuoglu et al. 2009]

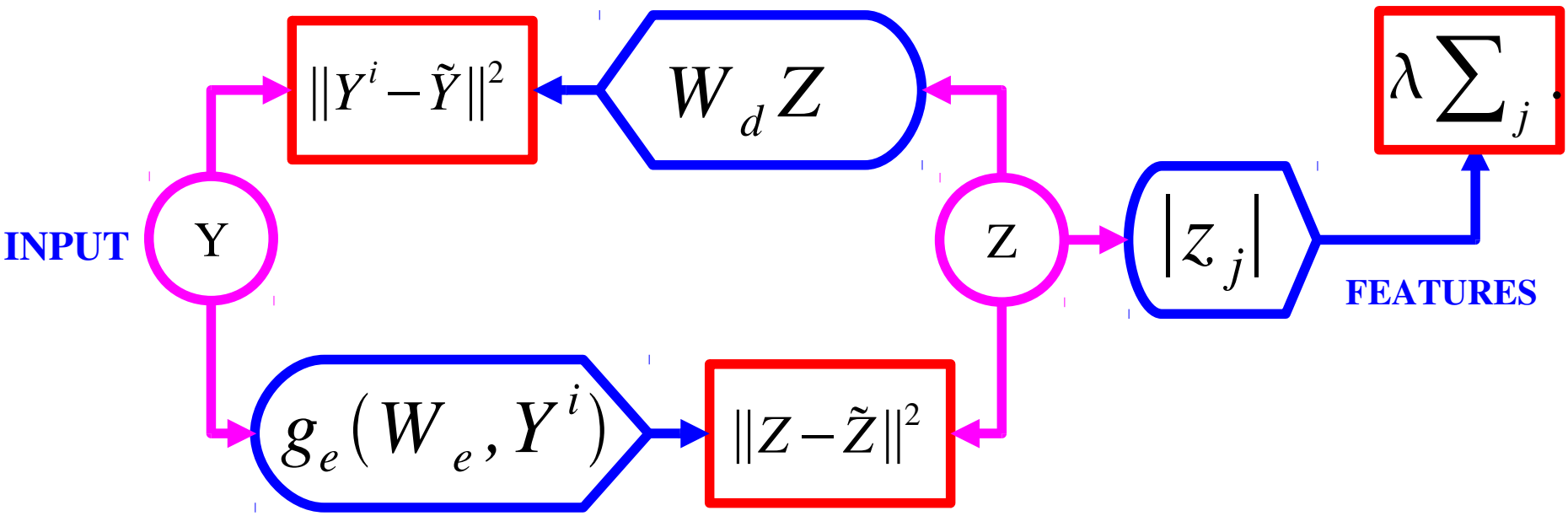
- Learning by minimizing the average energy of the training data with respect to Wd and We.
- Loss function:

$$L(W_d, W_e) = \sum_i F(Y^i; W_d, W_e)$$

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

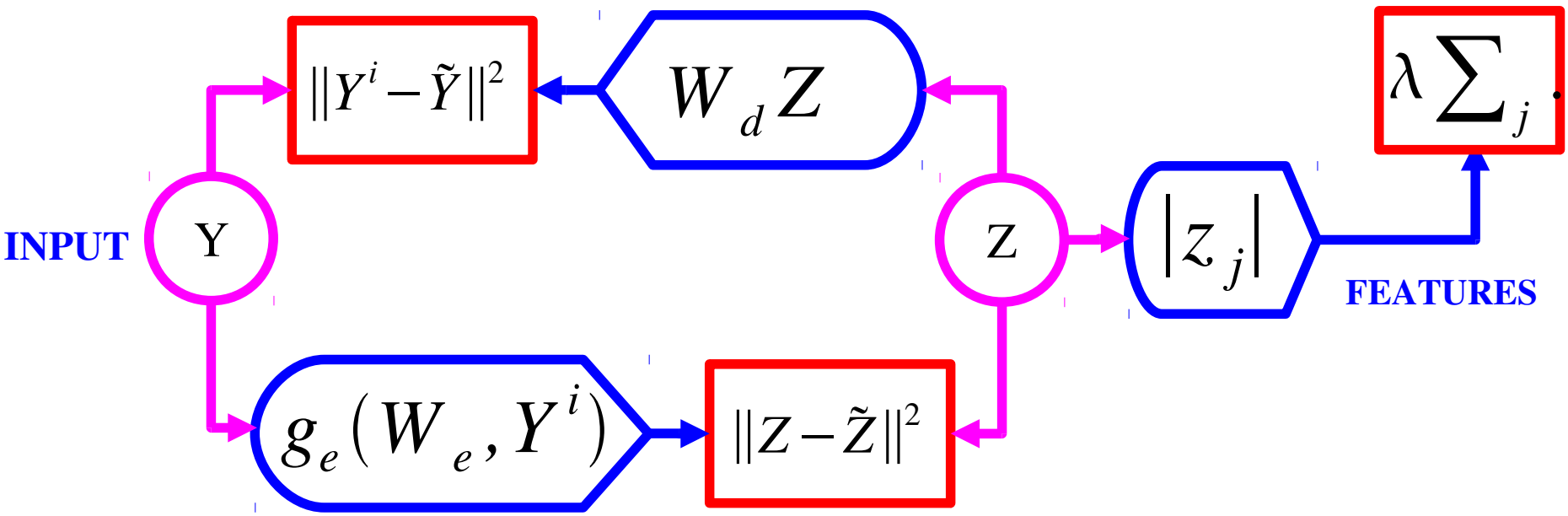


# PSD: Learning Algorithm




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

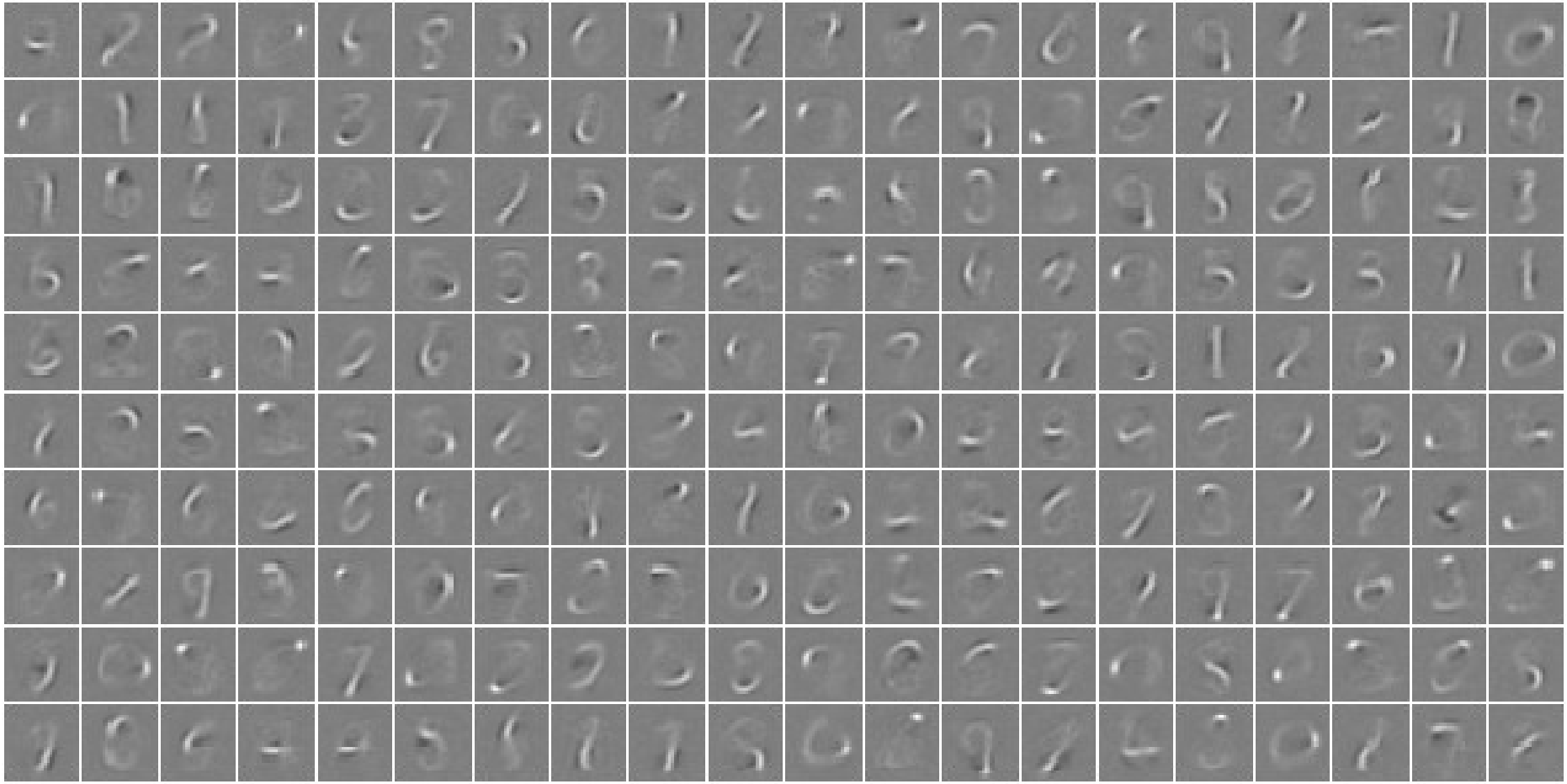
# PSD: Encoder Architectures




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

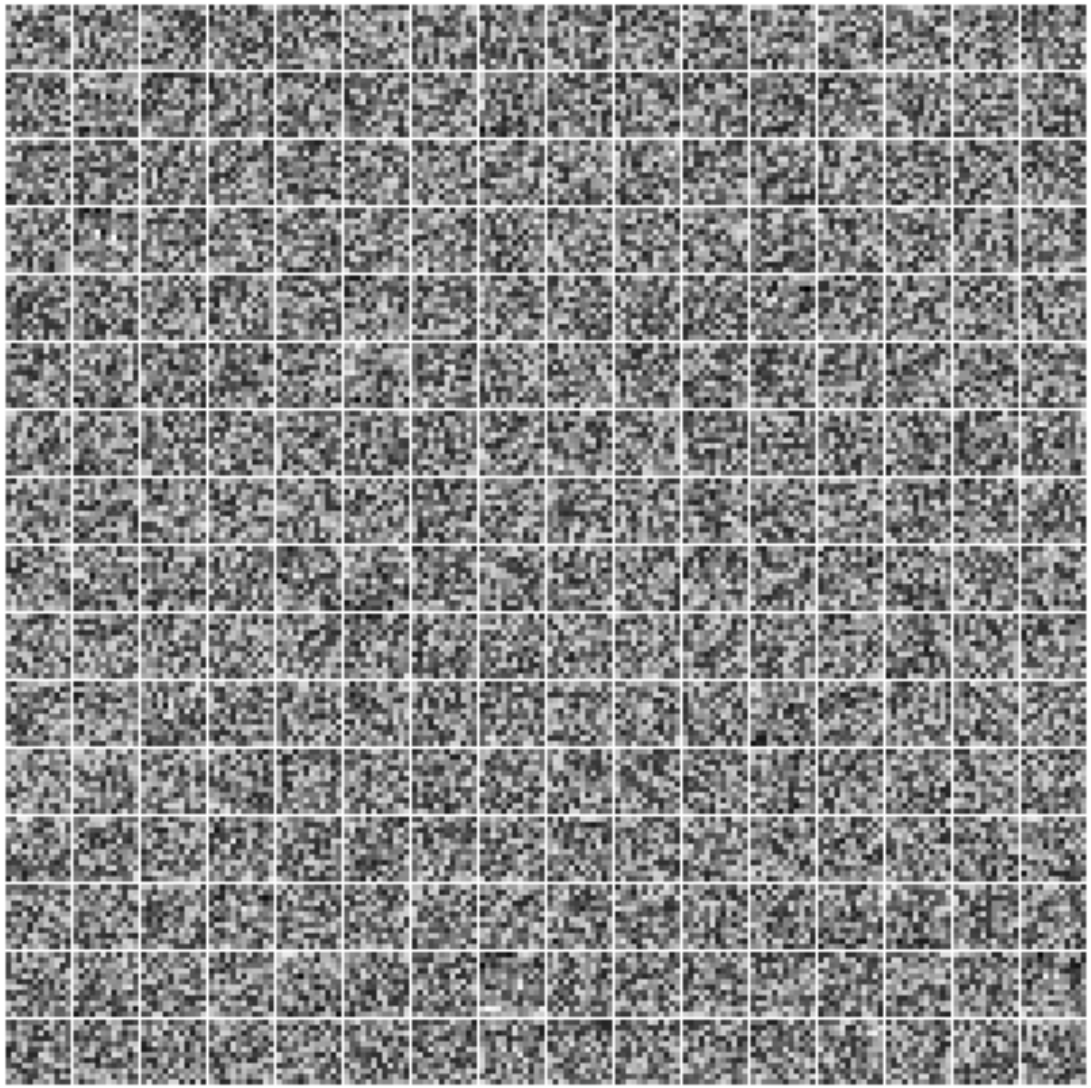
# Decoder Basis Functions on MNIST

- **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".

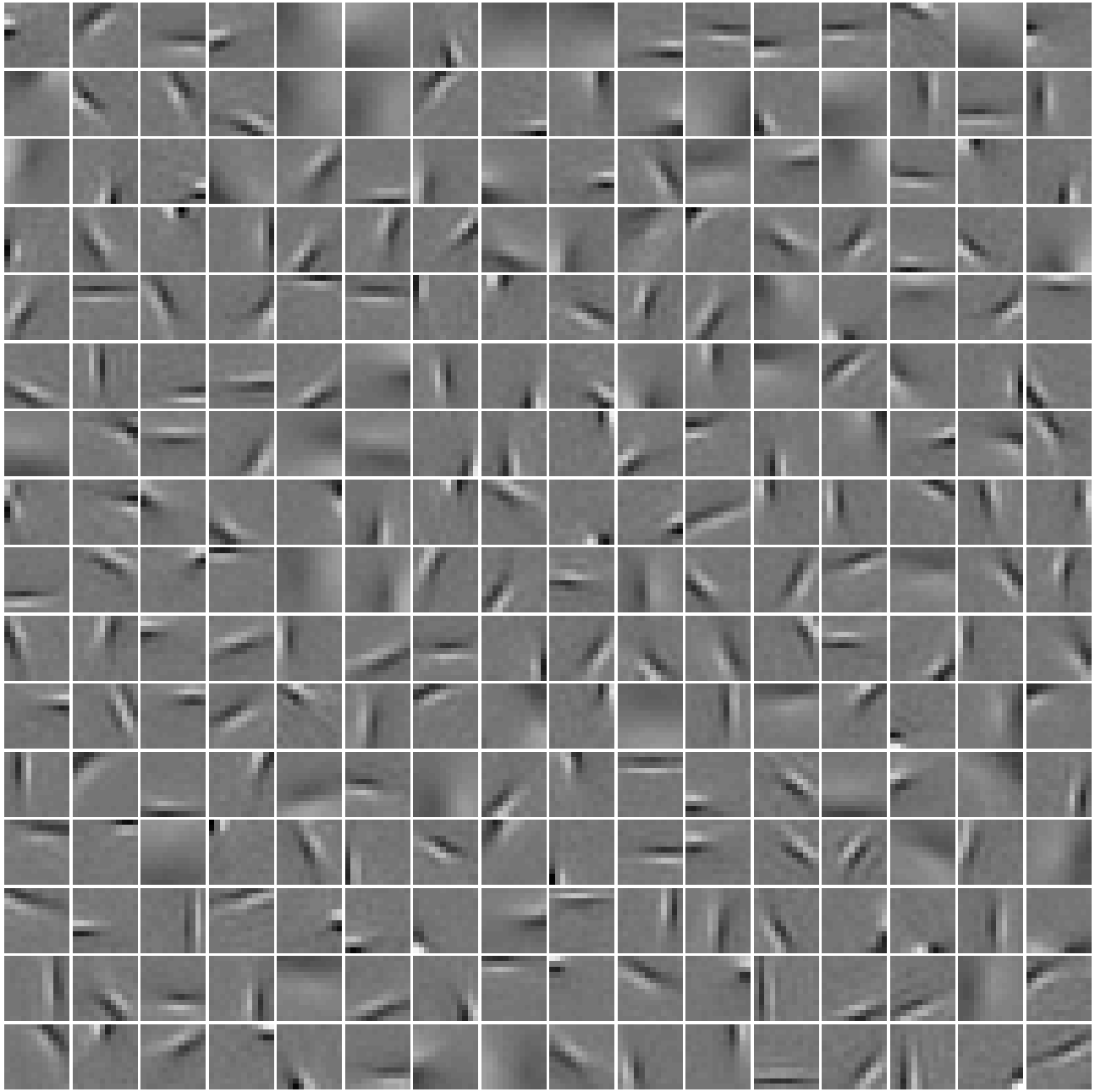


# PSD Training on Natural Image Patches

- 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

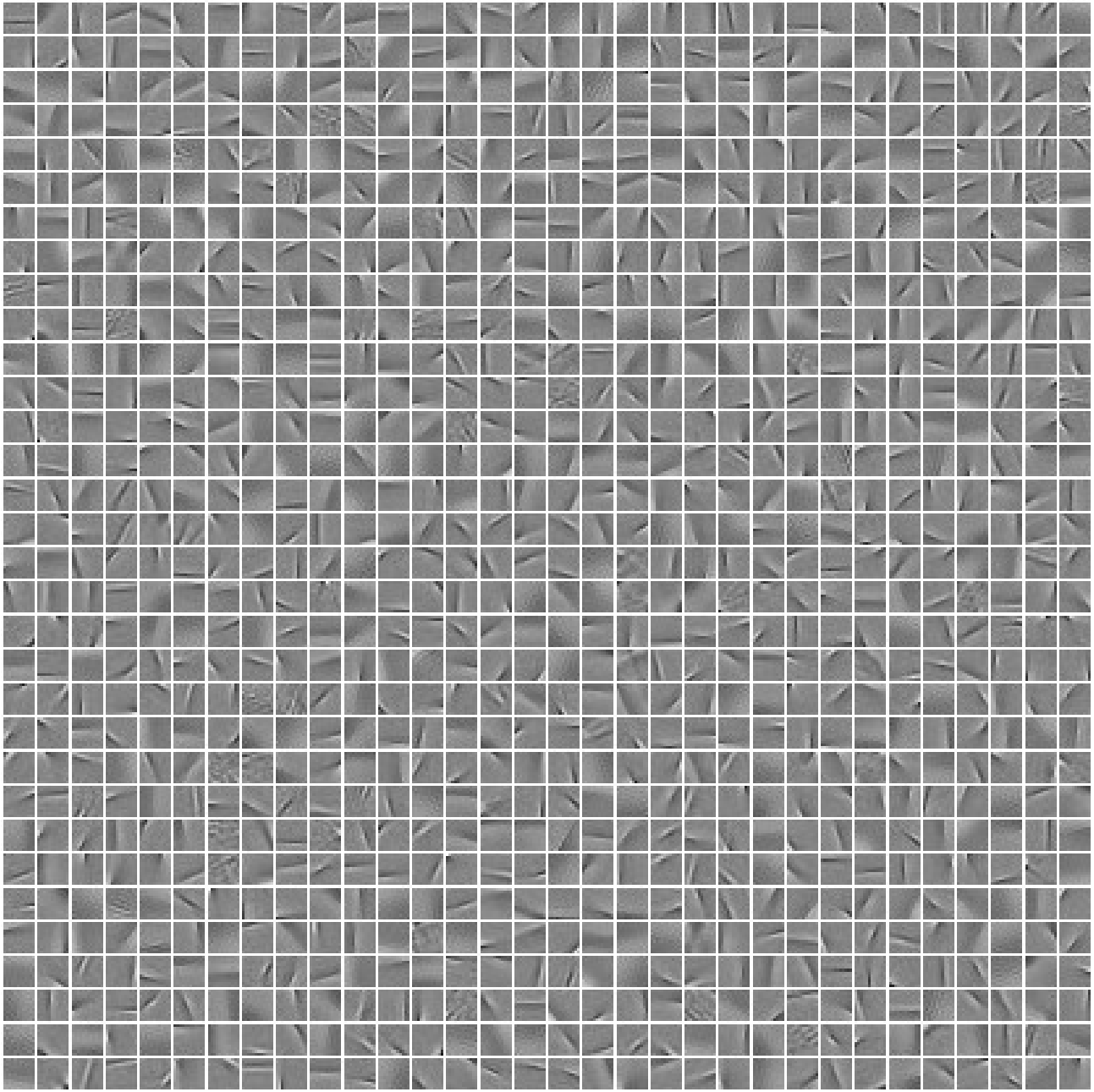


# Learned Features on natural patches: V1-like receptive fields



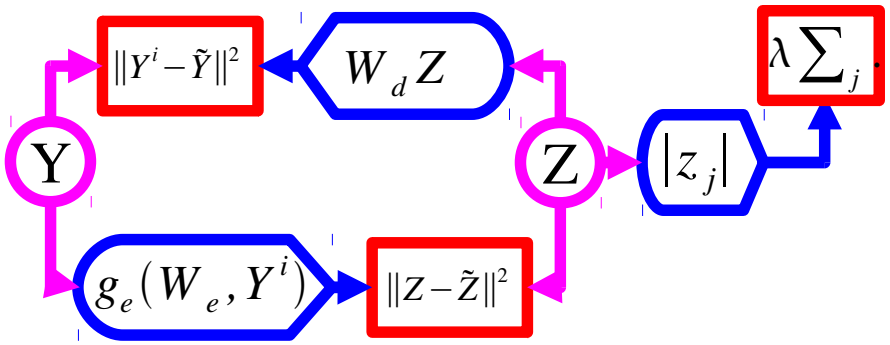
# Learned Features: V1-like receptive fields

- 12x12 filters
- 1024 filters



# Using PSD to Train a Hierarchy of Features

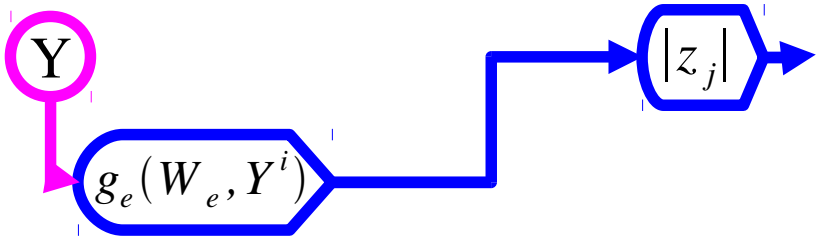
- **Phase 1: train first layer using PSD**




**FEATURES**

# Using PSD to Train a Hierarchy of Features

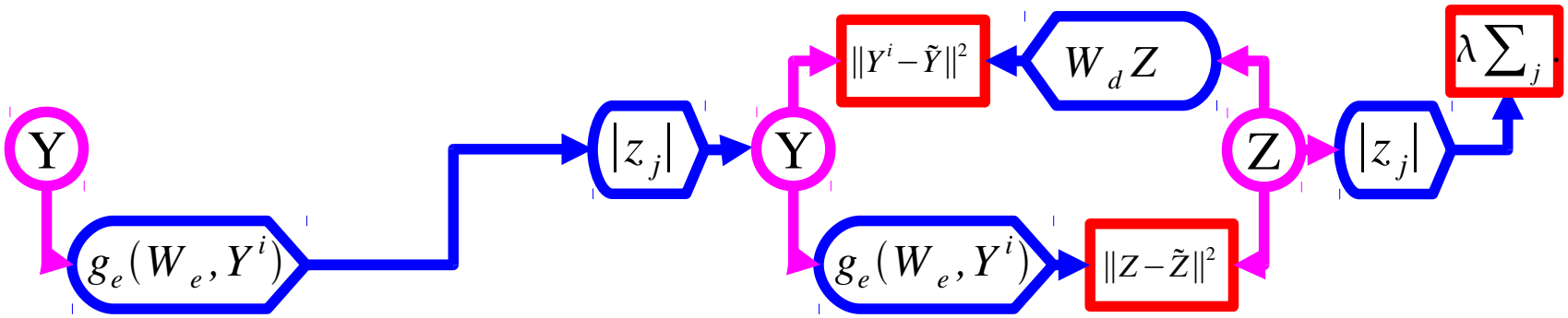
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor




FEATURES

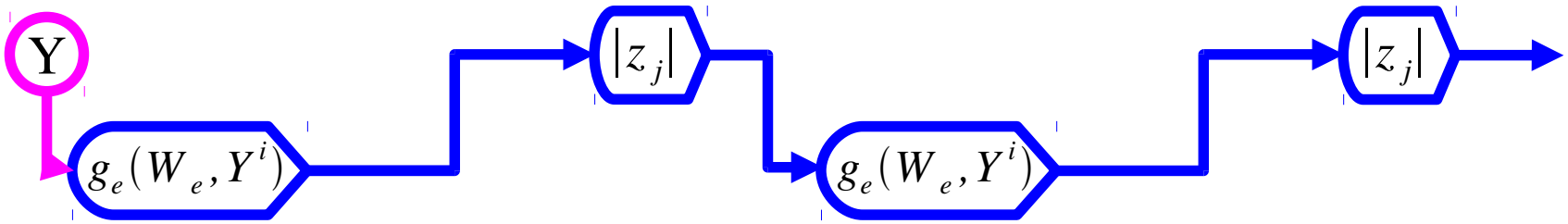
# Using PSD to Train a Hierarchy of Features

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



# Using PSD to Train a Hierarchy of Features

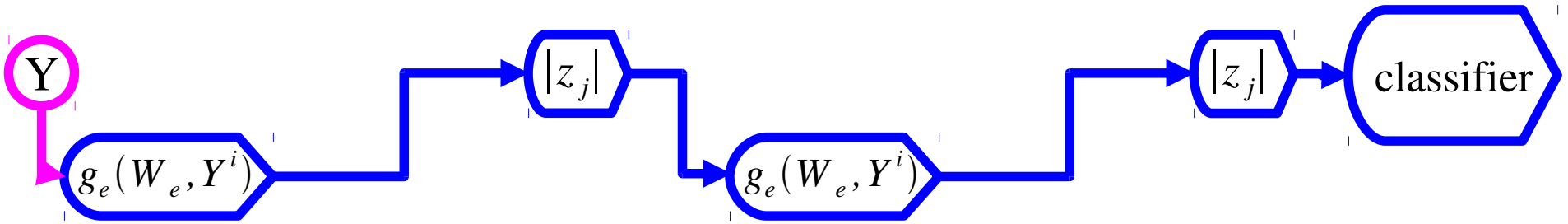
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor




FEATURES

# Using PSD to Train a Hierarchy of Features

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



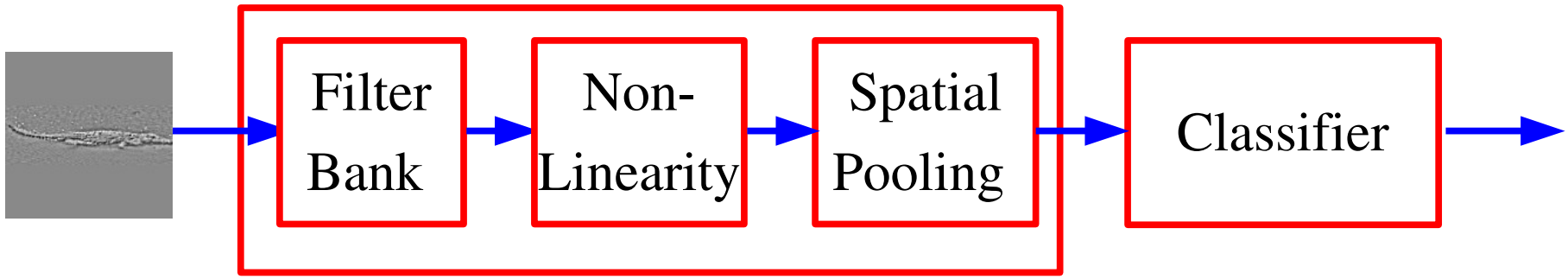

FEATURES

# "Deep Learning"

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

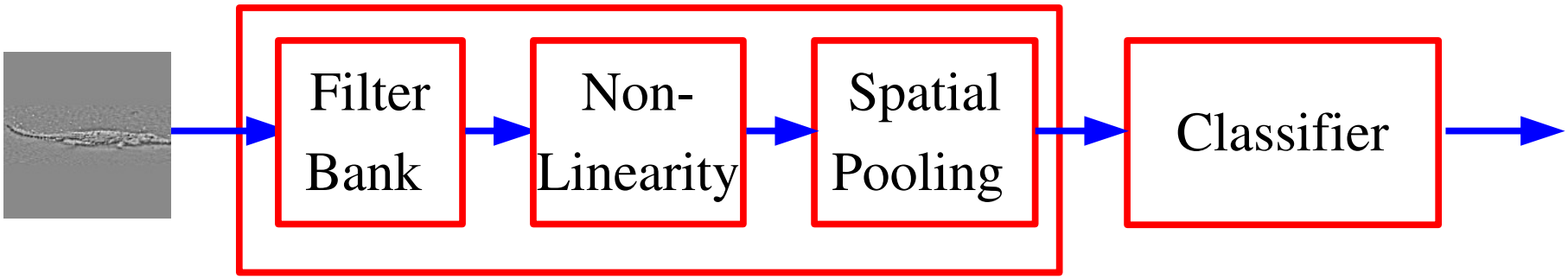
- **The "deep learning" method was popularized by Hinton for training "deep belief networks".**
  - DBN use a special kind of encoder-decoder architecture called Restricted Boltzmann Machines (RBM)
- **1. Train each layer in an unsupervised fashion, layer by layer**
- **2. Stick a supervised classifier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

# Using PSD to learn the features of an object recognition system




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

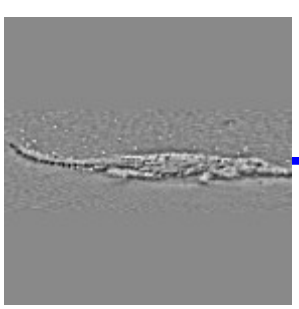
# "Modern" Object Recognition Architecture in Computer Vision




| | | |
|---|---|---|
| Oriented Edges | Sigmoid | Averaging |
| Gabor Wavelets | Rectification | Max pooling |
| Other Filters... | Vector Quant. | VQ+Histogram |
| | Contrast Norm. | Geometric Blurr |

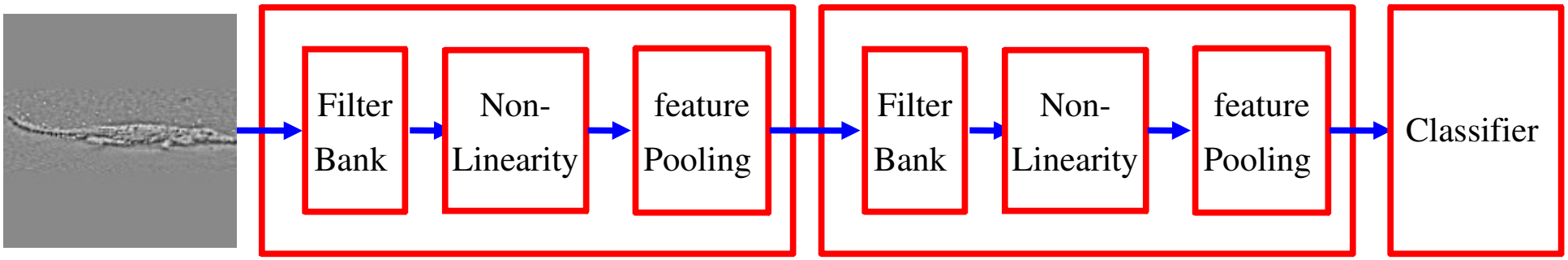
- **Example:**
  - Edges + Rectification + Histograms + SVM [Dalal & Triggs 2005]
  - SIFT + classification
- **Fixed Features + "shallow" classifier**

# "State of the Art" architecture for object recognition




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

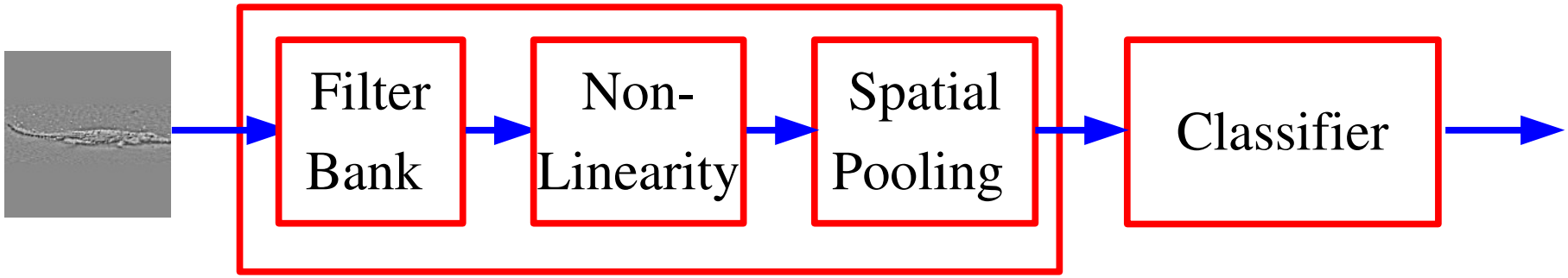
# Can't we get the same results with (deep) learning?




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

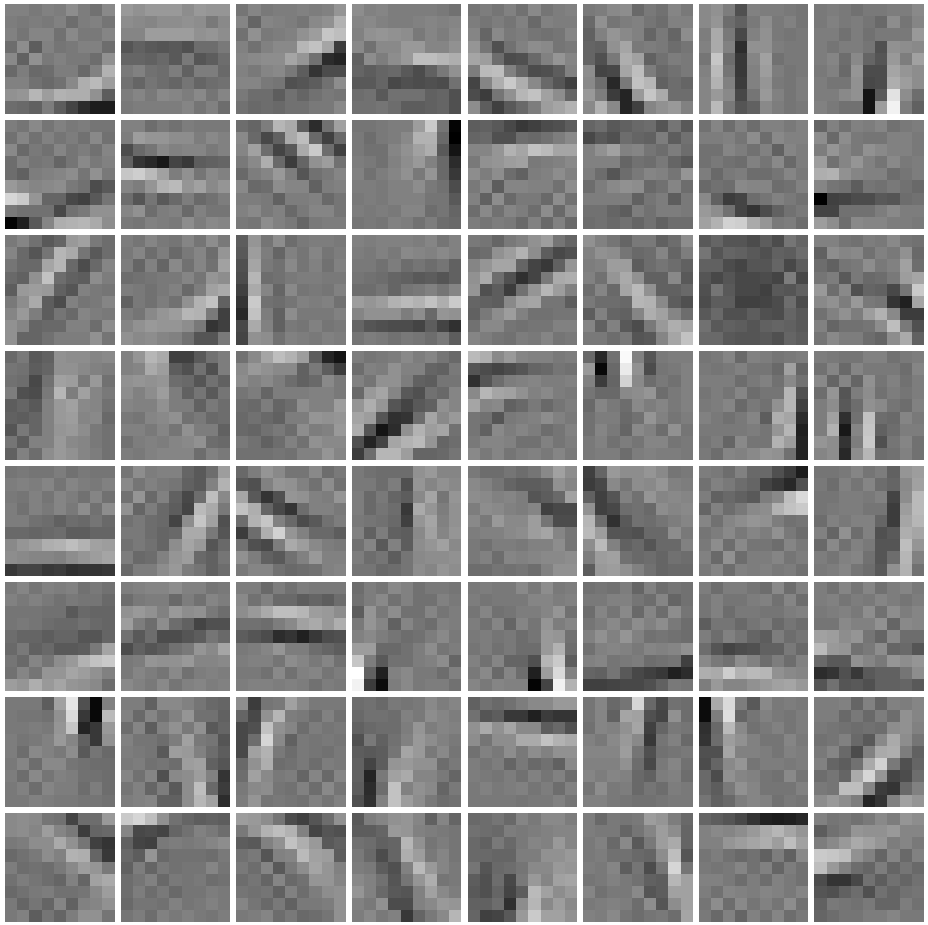
# Procedure for a single-stage system

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



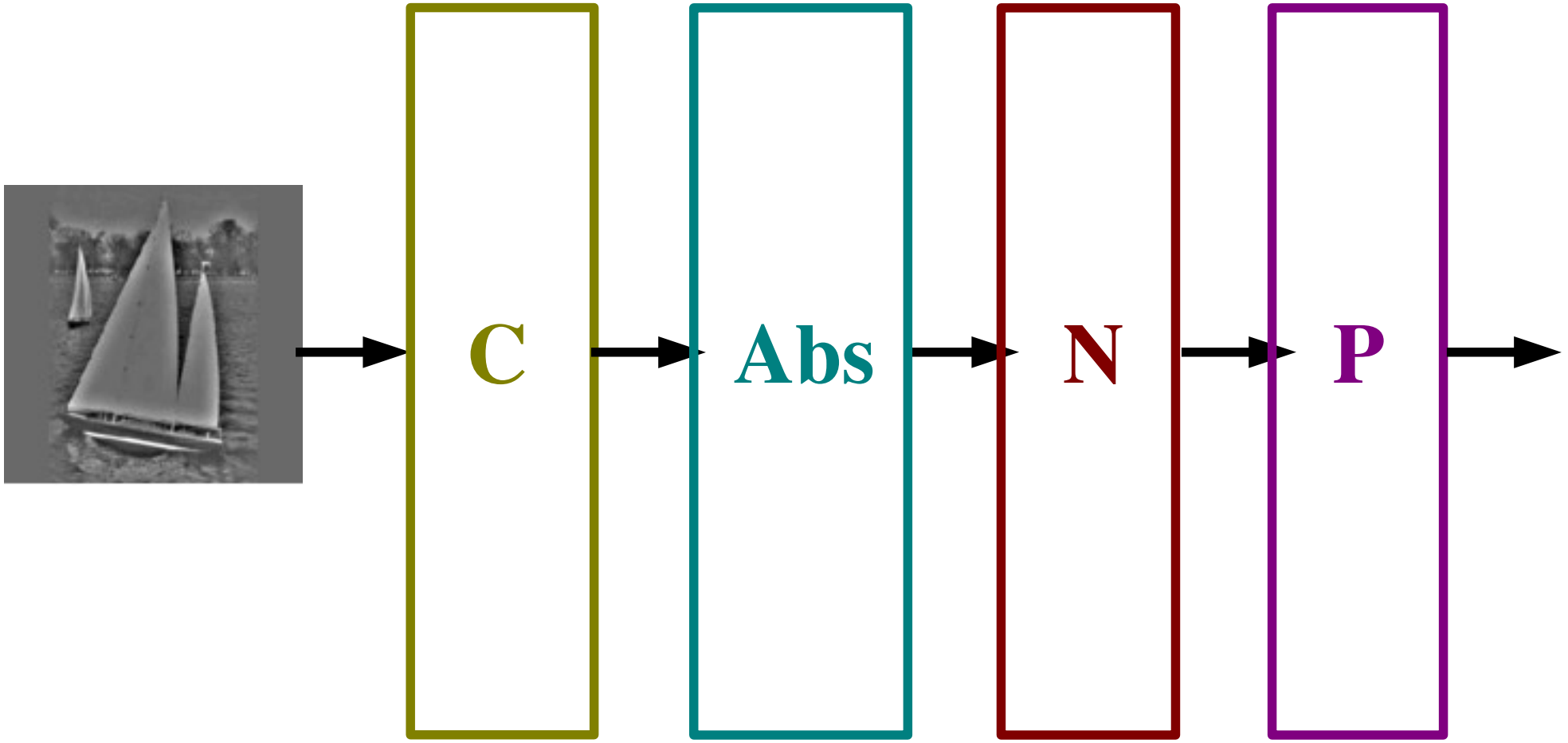
# Using PSD Features for Recognition

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



# Feature Extraction

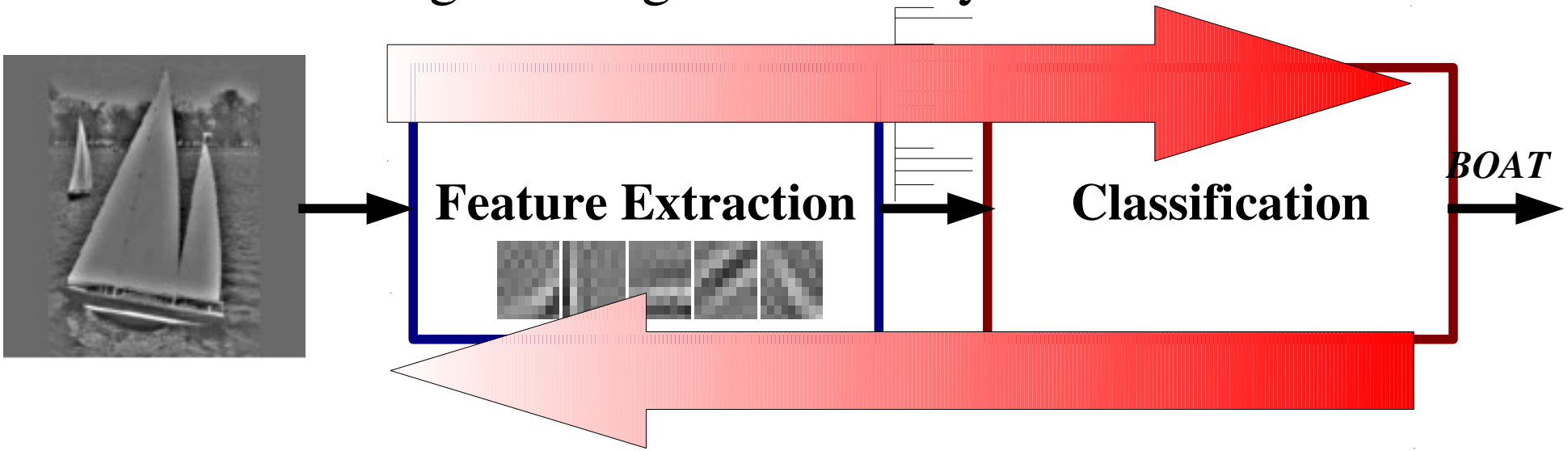
- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?
- **P** Pooling down-sampling layer: average or max?




**THIS IS ONE STAGE OF FEATURE EXTRACTION**

# Training Protocol

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



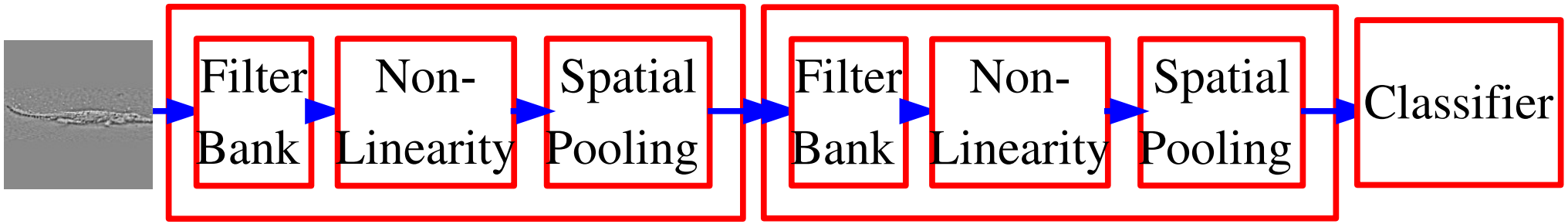
# Using PSD Features for Recognition

| $[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ - log_reg | | | | | |
|---|---|---|---|---|---|
| R/N/P | $R_{abs} - N - P_A$ | $R_{abs} - P_A$ | $N - P_M$ | $N - P_A$ | $P_A$ |
| $U^+$ | 54.2% | 50.0% | 44.3% | 18.5% | 14.5% |
| $R^+$ | 54.8% | 47.0% | 38.0% | 16.3% | 14.3% |
| U | 52.2% | 43.3(±1.6)% | 44.0% | 17.2% | 13.4% |
| R | 53.3% | 31.7% | 32.1% | 15.3% | 12.1(±2.2)% |
| $[64.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ - PMK | | | | | |
| U | 65.0% | | | | |
| $[96.F^{9\times9}_{CSG} - R/N/P^{5\times5}]$ - PCA - lin_svm | | | | | |
| U | 58.0% | | | | |
| 96.Gabors - PCA - lin_svm (Pinto and DiCarlo 2006) | | | | | |
| Gabors | 59.0% | | | | |
| SIFT - PMK (Lazebnik et al. CVPR 2006) | | | | | |
| Gabors | 64.6% | | | | |

# Using PSD Features for Recognition

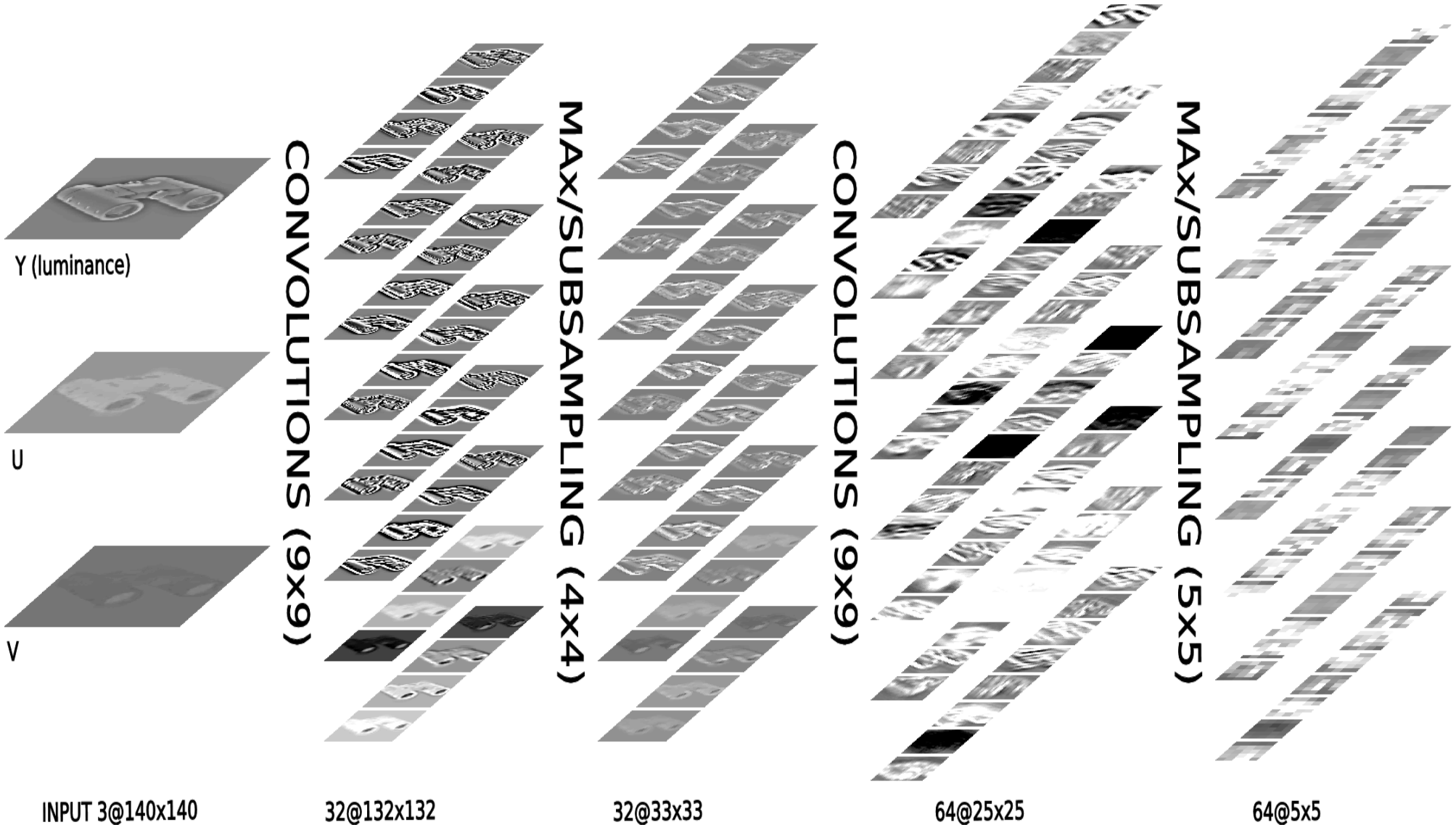
- **Rectification makes a huge difference:**
  - 14.5% -> 50.0%, without normalization
  - 44.3% -> 54.2% with normalization
- **Normalization makes a difference:**
  - 50.0 → 54.2
- **Unsupervised pretraining makes small difference**
- **PSD works just as well as SIFT**
- **Random filters work as well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

# Training a Multi-Stage Hubel-Wiesel Architecture with PSD




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

# Multistage Hubel-Wiesel Architecture on Caltech-101



# Multistage Hubel-Wiesel Architecture

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

# Multistage Hubel-Wiesel Architecture on Caltech-101

| **Single Stage System:** $[64.F_{CSG}^{9\times9} - R/N/P^{5\times5}]$ **- log_reg** | | | | | |
|---|---|---|---|---|---|
| R/N/P | $R_{abs} - N - P_A$ | $R_{abs} - P_A$ | $N - P_M$ | $N - P_A$ | $P_A$ |
| $U^+$ | 54.2% | 50.0% | 44.3% | 18.5% | 14.5% |
| $R^+$ | 54.8% | 47.0% | 38.0% | 16.3% | 14.3% |
| U | 52.2% | 43.3%(±1.6) | 44.0% | 17.2% | 13.4% |
| R | 53.3% | 31.7% | 32.1% | 15.3% | 12.1%(±2.2) |
| G | 52.3% | | | | |
| **Two Stage System:** $[64.F_{CSG}^{9\times9} - R/N/P^{5\times5}] - [256.F_{CSG}^{9\times9} - R/N/P^{4\times4}]$ **- log_reg** | | | | | |
| R/N/P | $R_{abs} - N - P_A$ | $R_{abs} - P_A$ | $N - P_M$ | $N - P_A$ | $P_A$ |
| $U^+U^+$ | 65.5% | 60.5% | 61.0% | 34.0% | 32.0% |
| $R^+R^+$ | 64.7% | 59.5% | 60.0% | 31.0% | 29.7% |
| UU | 63.7% | 46.7% | 56.0% | 23.1% | 9.1% |
| RR | 62.9% | 33.7%(±1.5) | 37.6%(±1.9) | 19.6% | 8.8% |
| GT | 55.8% | ← like HMAX model | | | |
| **Single Stage:** $[64.F_{CSG}^{9\times9} - R/N/P^{5\times5}]$ **- PMK-SVM** | | | | | |
| U | 64.0% | | | | |
| **Two Stages:** $[64.F_{CSG}^{9\times9} - R/N/P^{5\times5}] - [256.F_{CSG}^{9\times9} - R/N]$ **- PMK-SVM** | | | | | |
| UU | 52.8% | | | | |

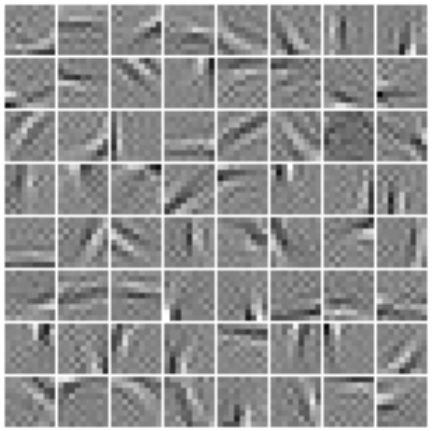
# Two-Stage Result Analysis

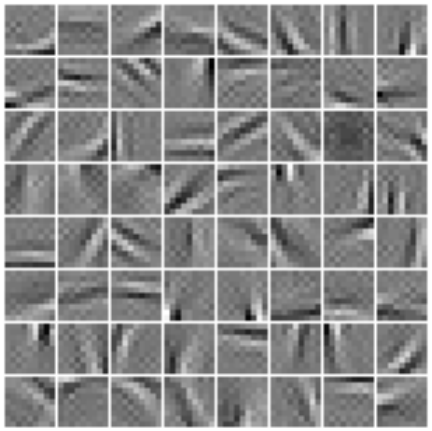
- Latest result: 69.7% correct
- Second Stage + logistic regression = PMK_SVM
- Unsupervised pre-training doesn't help much :-(
- Random filters work amazingly well with normalization
- Supervised global refinement helps a bit
- The best system is really cheap
- Either use rectification and average pooling or no rectification and max pooling.

# Multistage Hubel-Wiesel Architecture: Filters

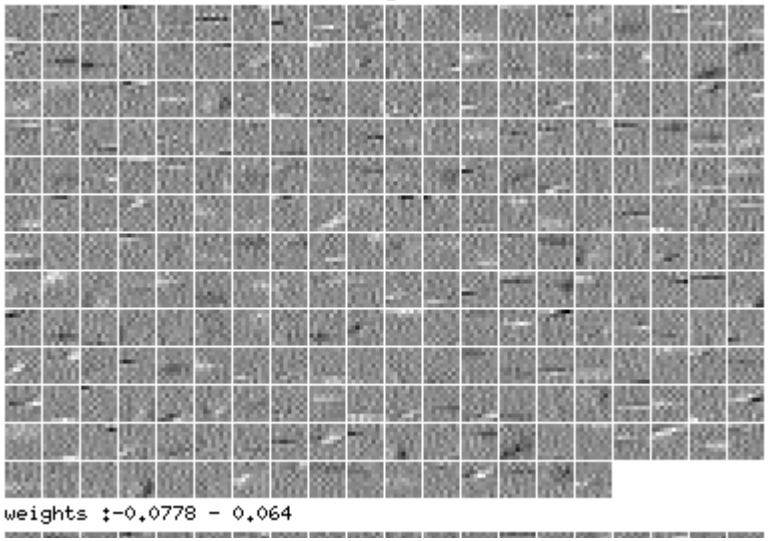
- After PSD
- After supervised refinement

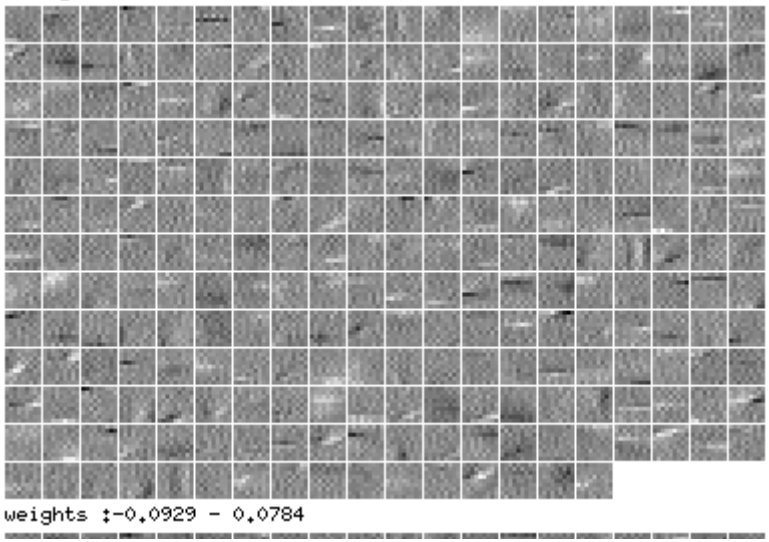
- Stage 1



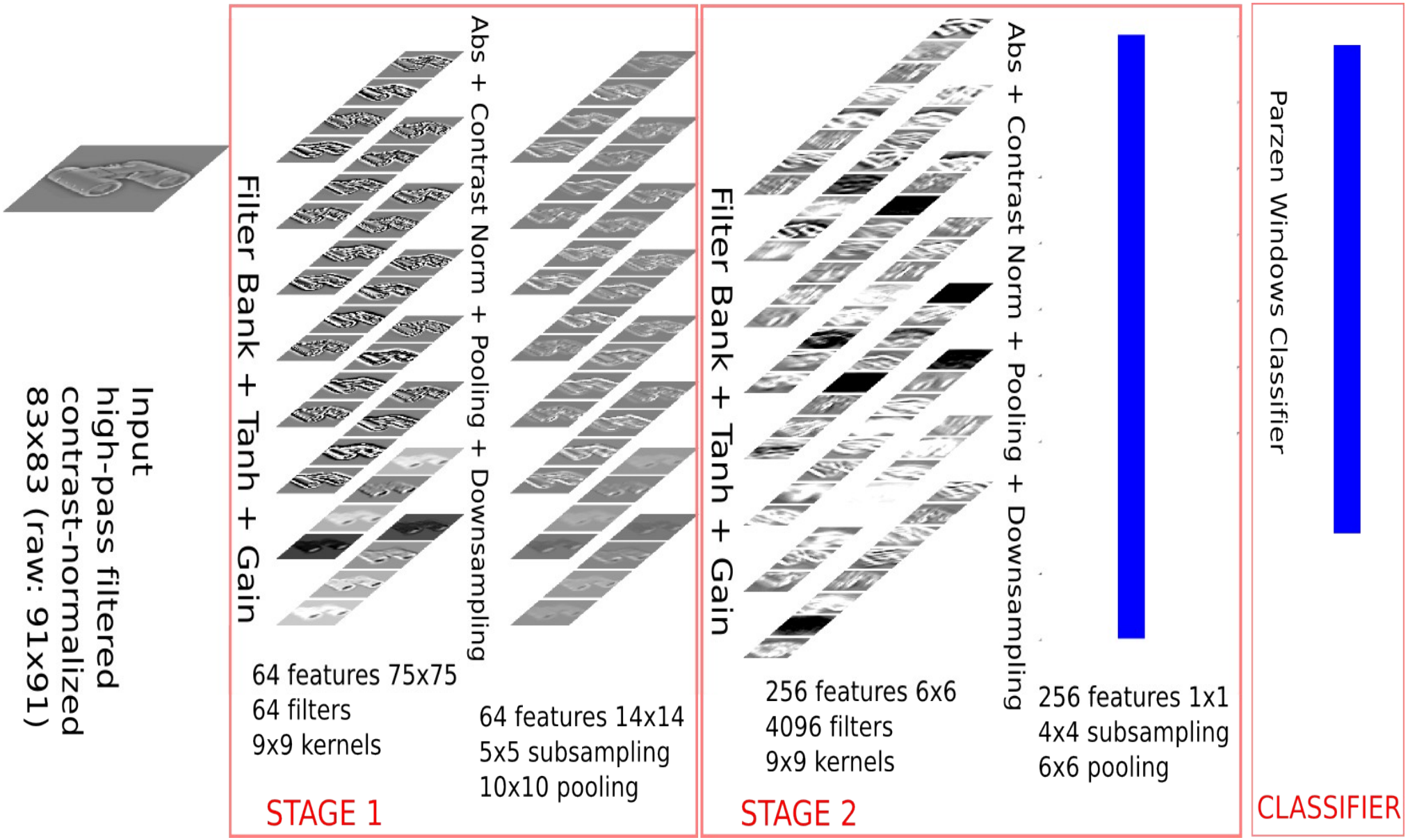




- Stage2

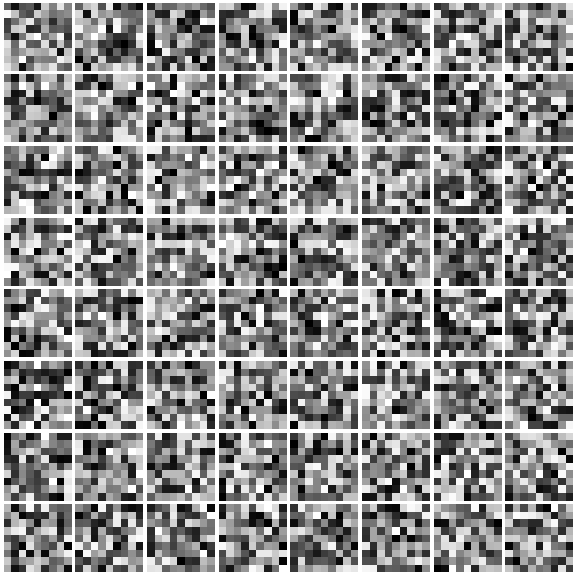


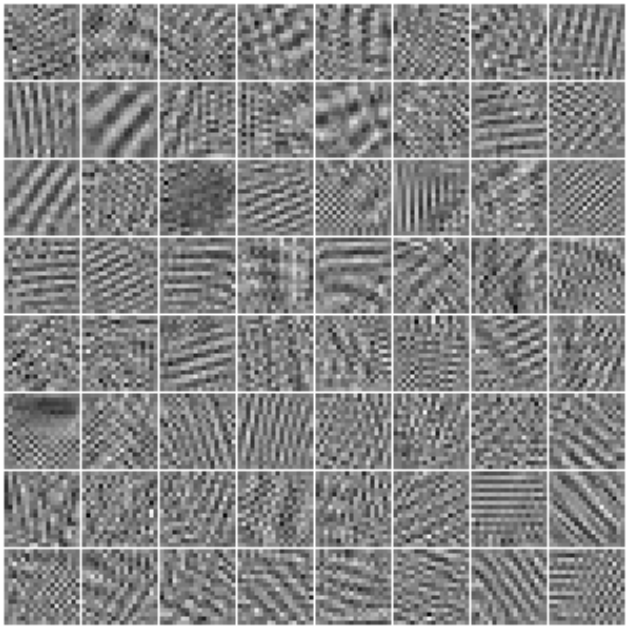



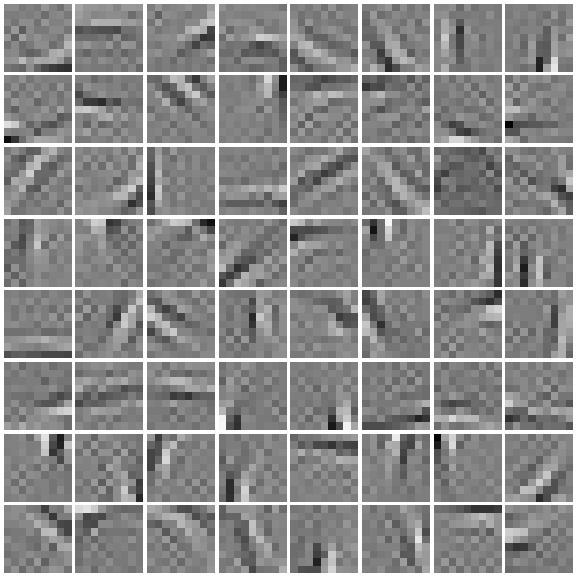
# Demo: real-time learning of visual categories

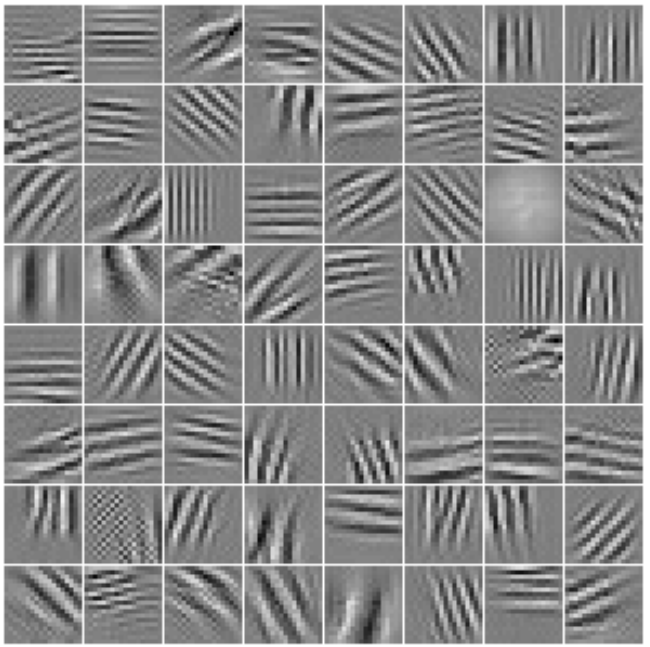


# Why Random Filters Work?









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

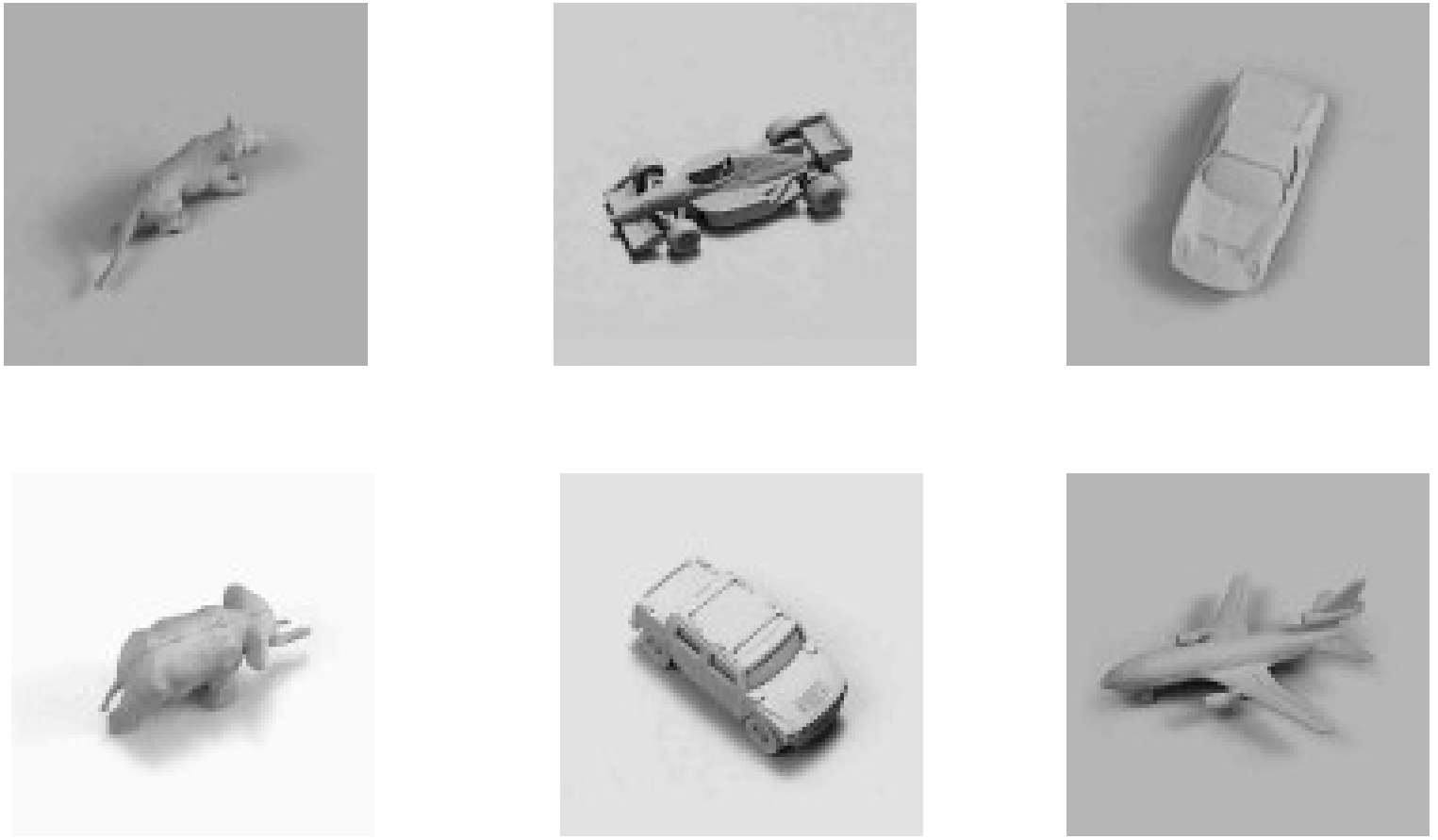
## Replacing K-means with Sparse Coding

- [Yang 2008] [Boureau, Bach, Ponce, LeCun 2010]

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

# Small NORB dataset

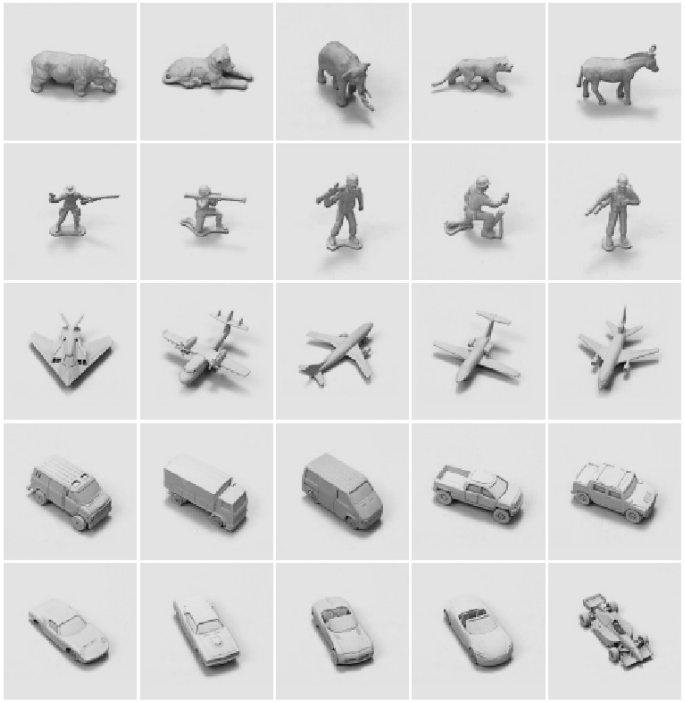
- 5 classes and up to 24,300 training samples per class



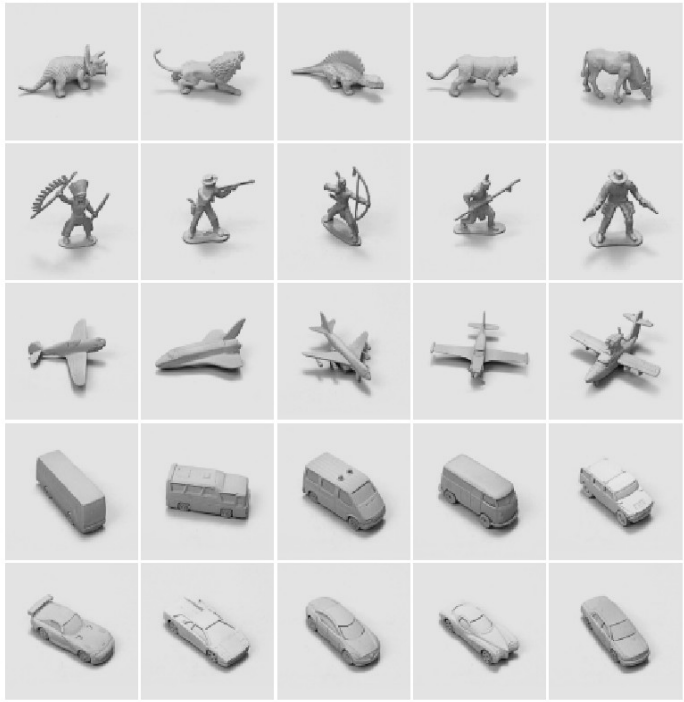
# NORB Generic Object Recognition Dataset

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

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



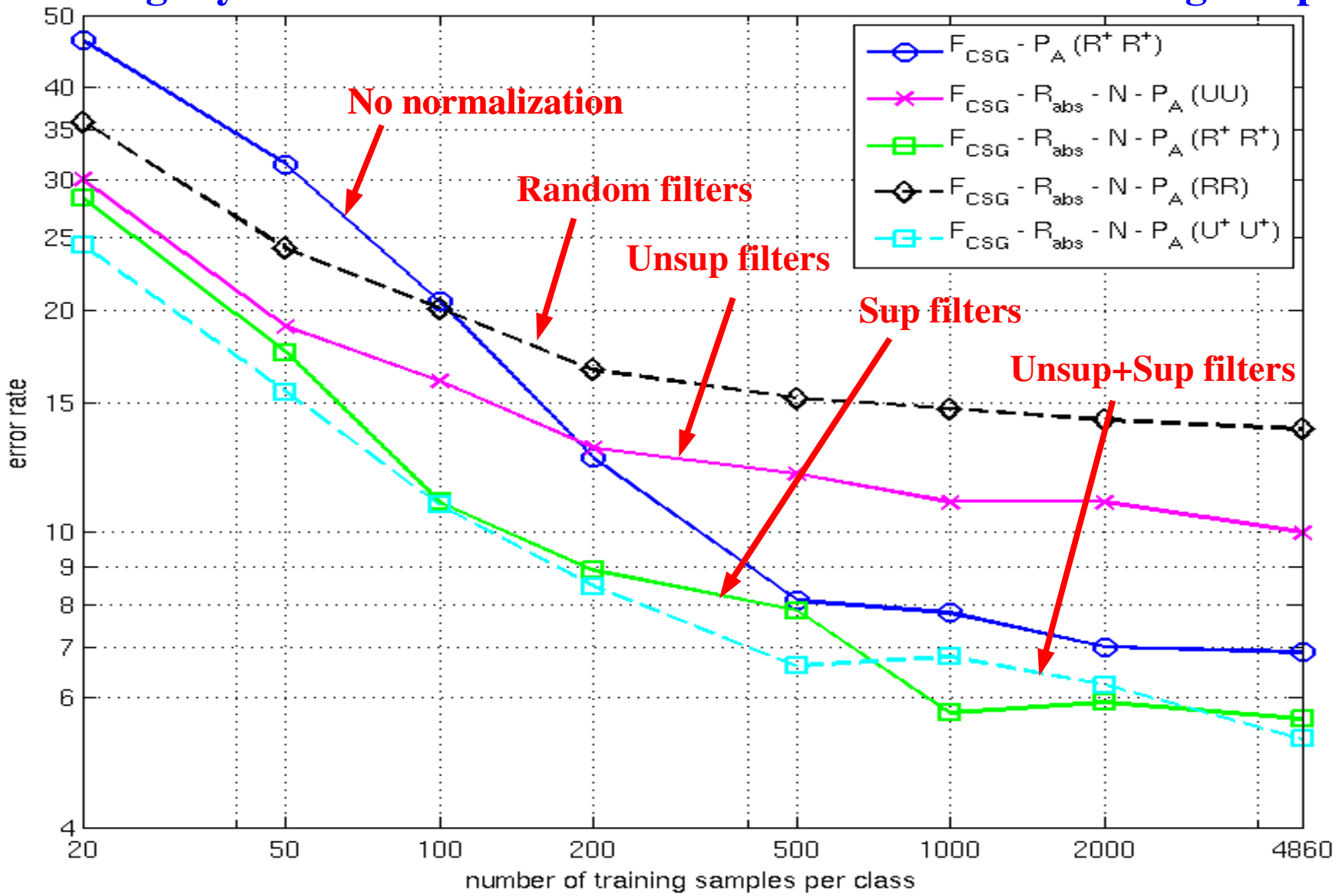
Training instances



Test instances

# Small NORB dataset

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

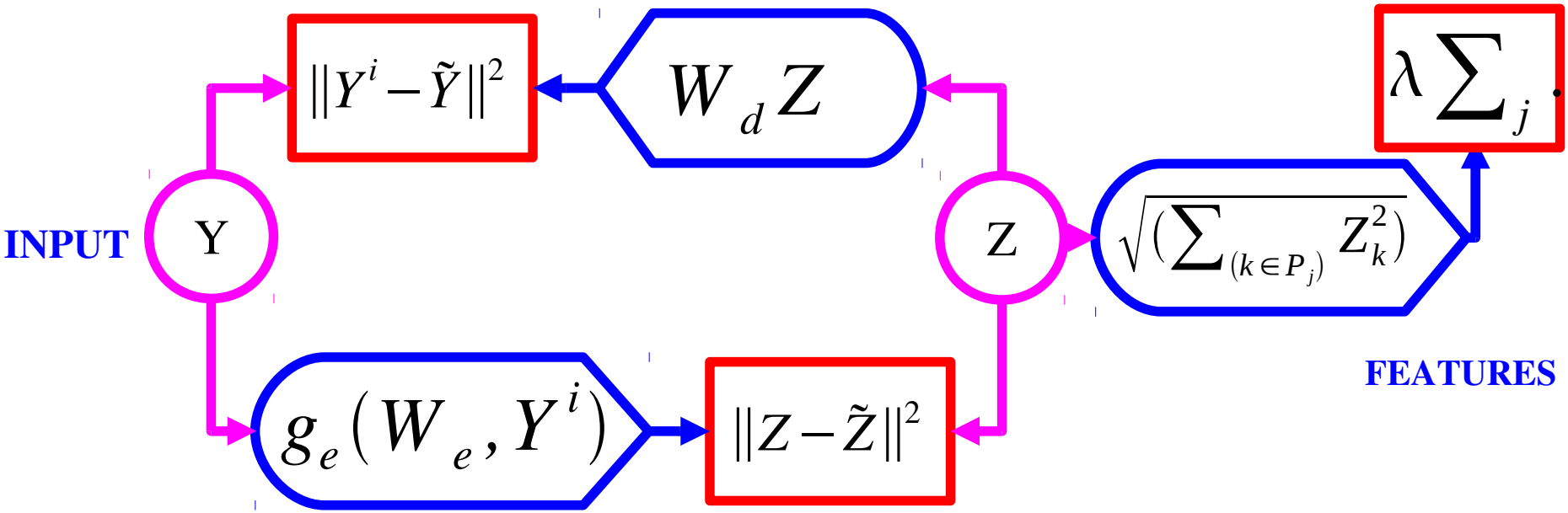


# Learning Complex Cells with Invariance Properties

[Kavukcuoglu et al. CVPR 2008]

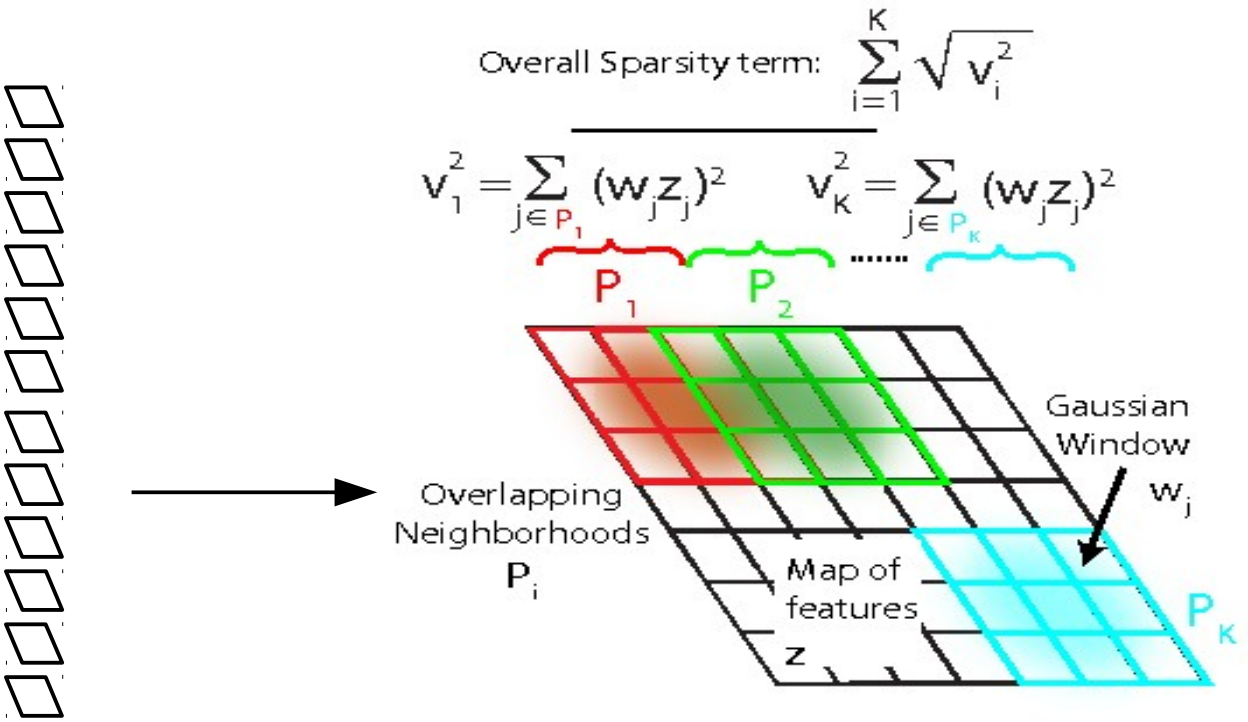
# Learning Invariant Features [Kavukcuoglu et al. CVPR 2009]

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



# Learning the filters and the pools

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

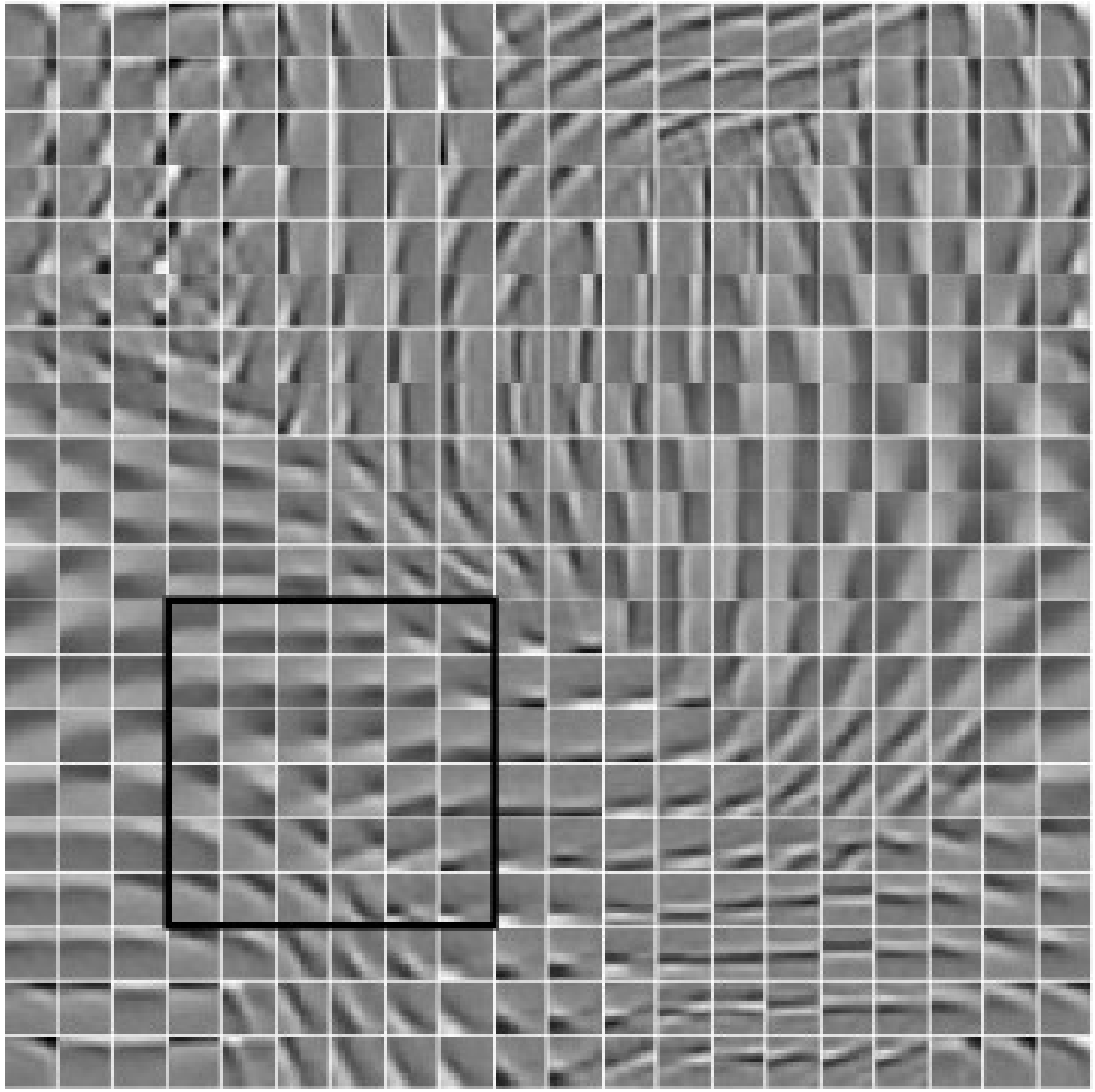



Units in the code Z

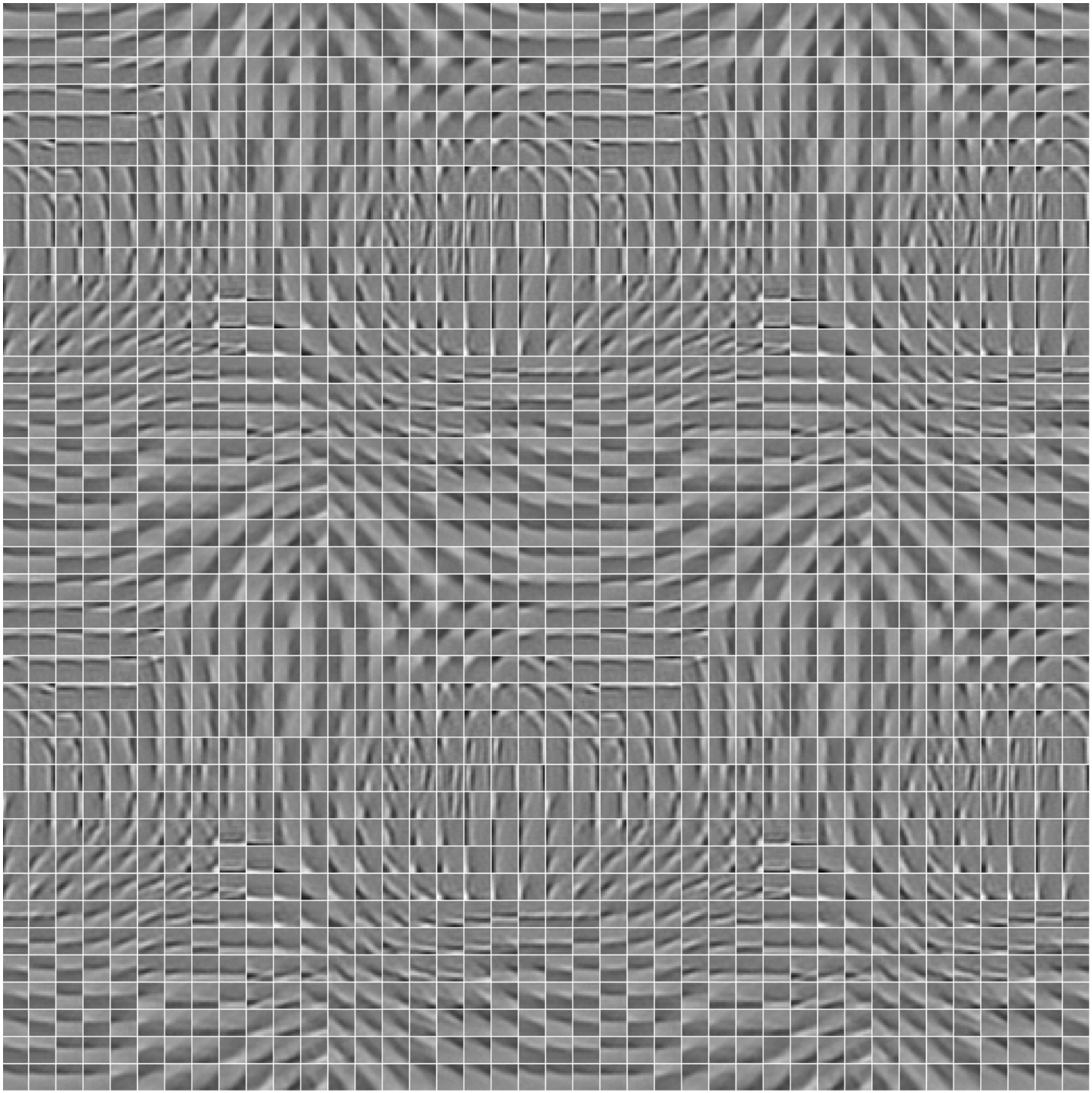
Define pools and enforce sparsity across pools

# Learning the filters and the pools

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

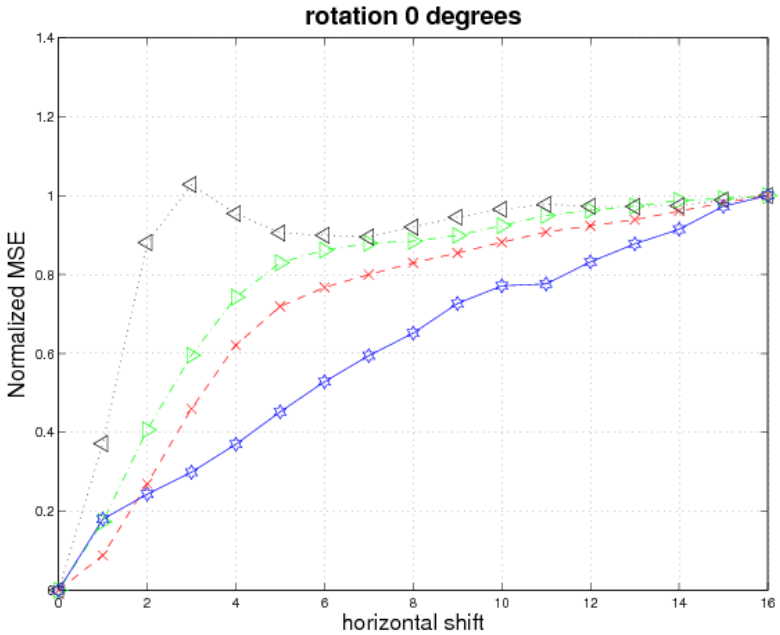


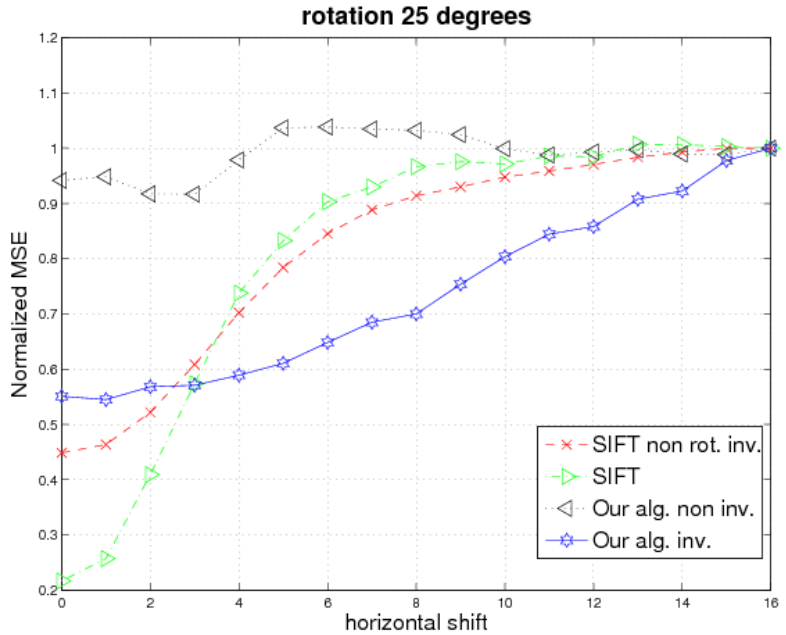
# Pinwheels?



# Invariance Properties Compared to SIFT

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

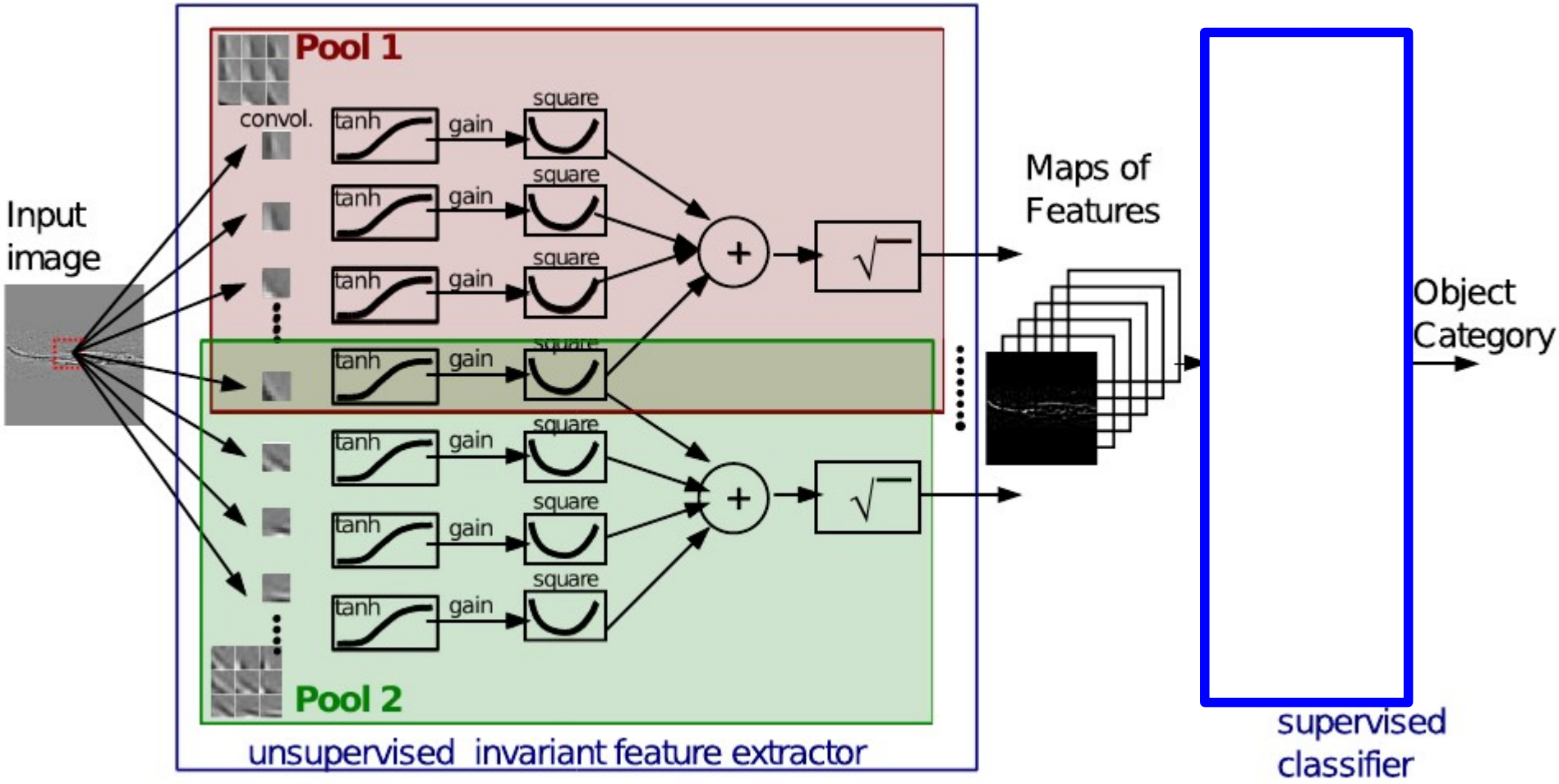


# Learning Invariant Features

## Recognition Architecture

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



# Recognition Accuracy on Caltech 101

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

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

# Recognition Accuracy on Tiny Images & MNIST

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

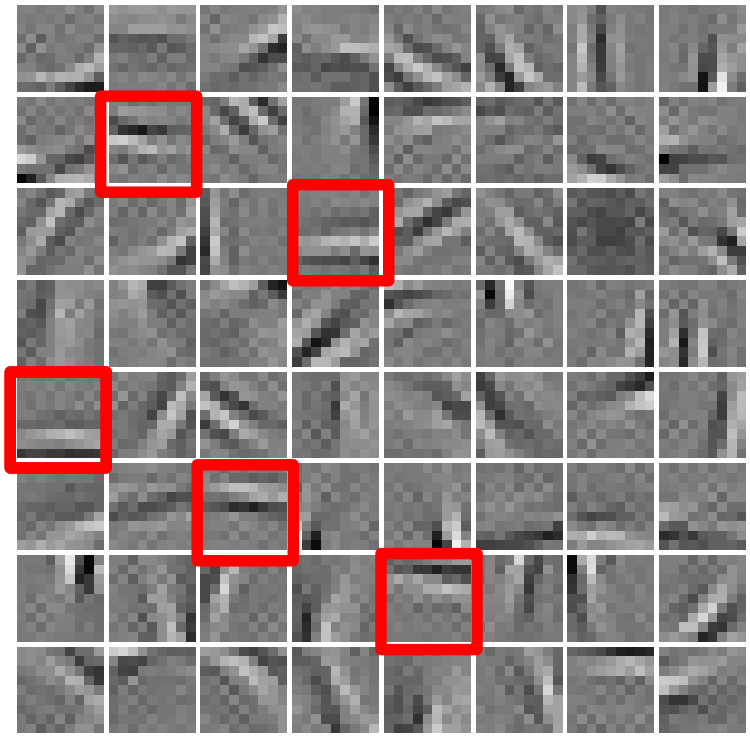
| Performance on Tiny Images Dataset | |
|---|---|
| **Method** | **Accuracy (%)** |
| IPSD (5x5) | **54** |
| SIFT (5x5) (non rot. inv.) | **53** |

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

# Learning fields of Convolutional Filters

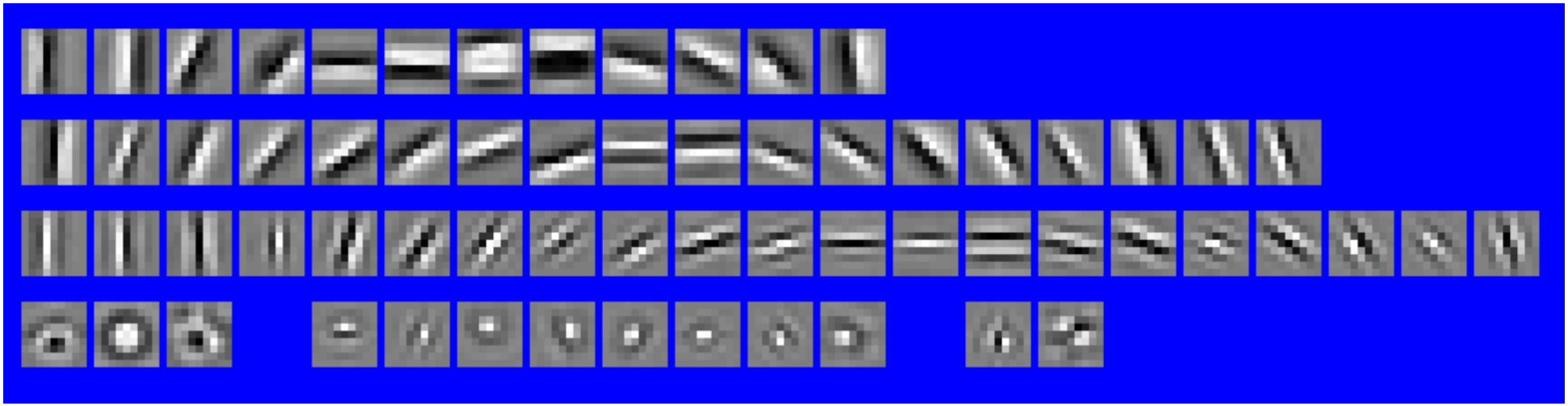
# Convolutional Training

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



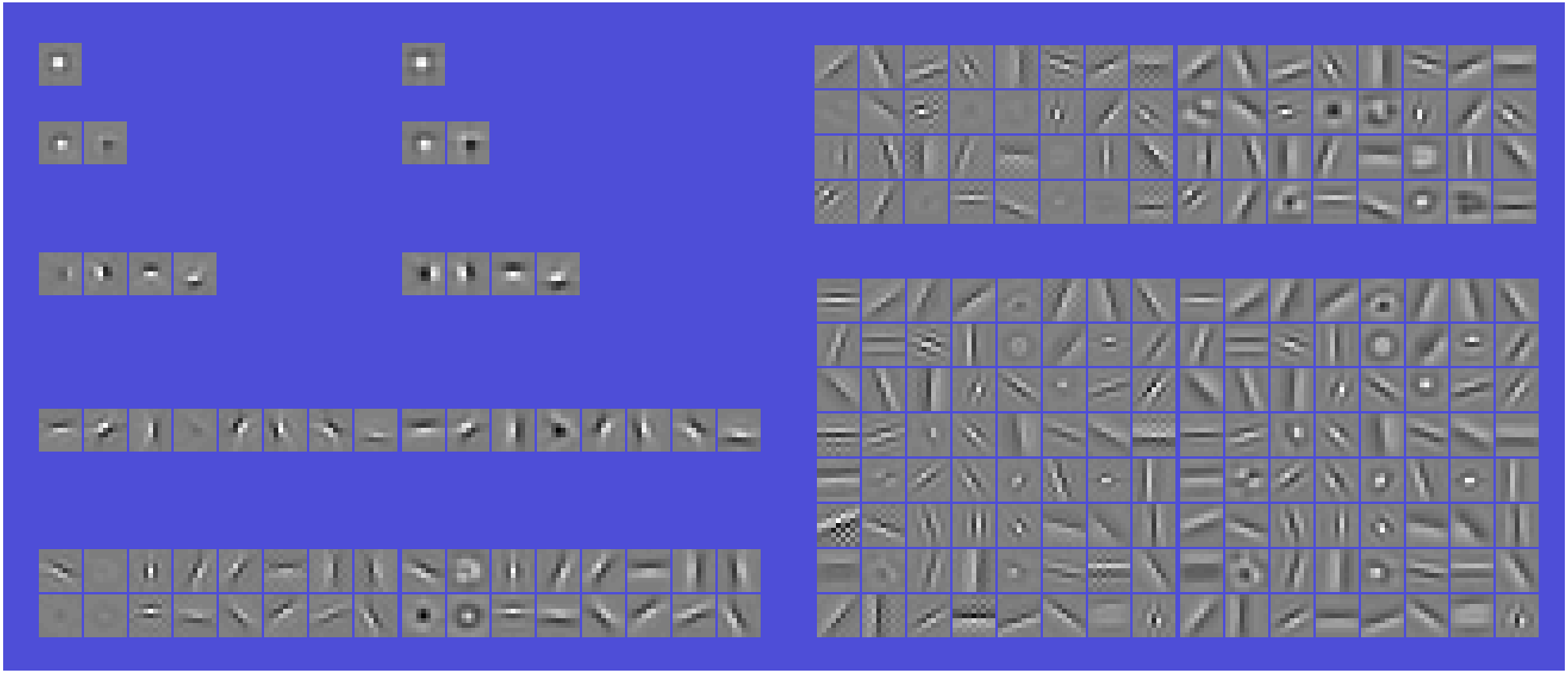
# Convolutional Training

- Problem with patch-based training: high correlation between outputs of filters from overlapping receptive fields.



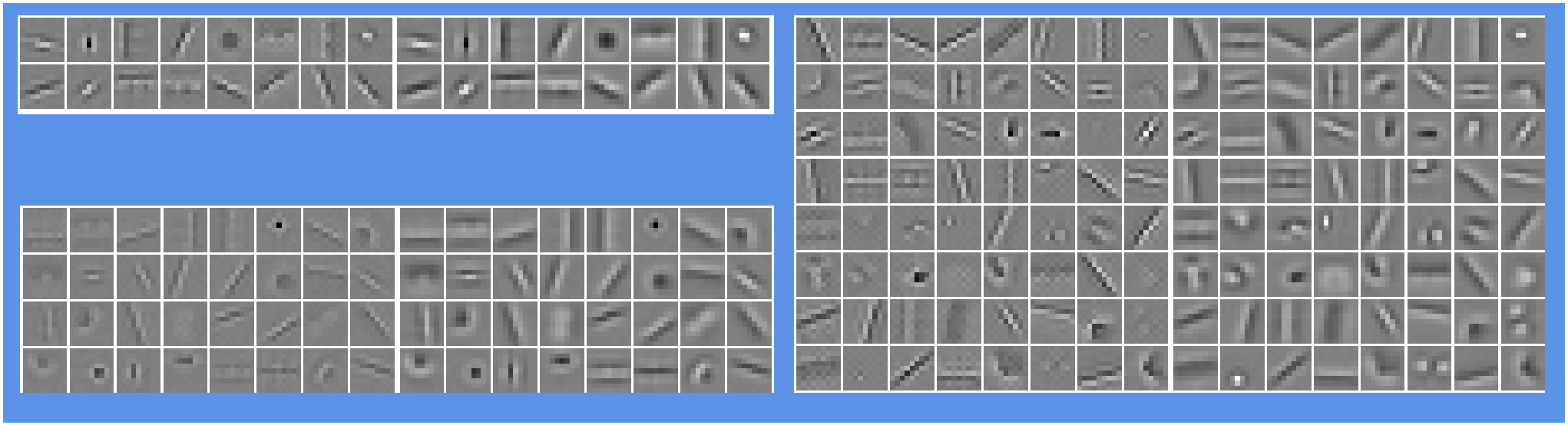
# Convolutional Training

- Filters and Basis Functions obtained with 1, 2, 4, 8, 16, 32, and 64 filters.



# Convolutional Training

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



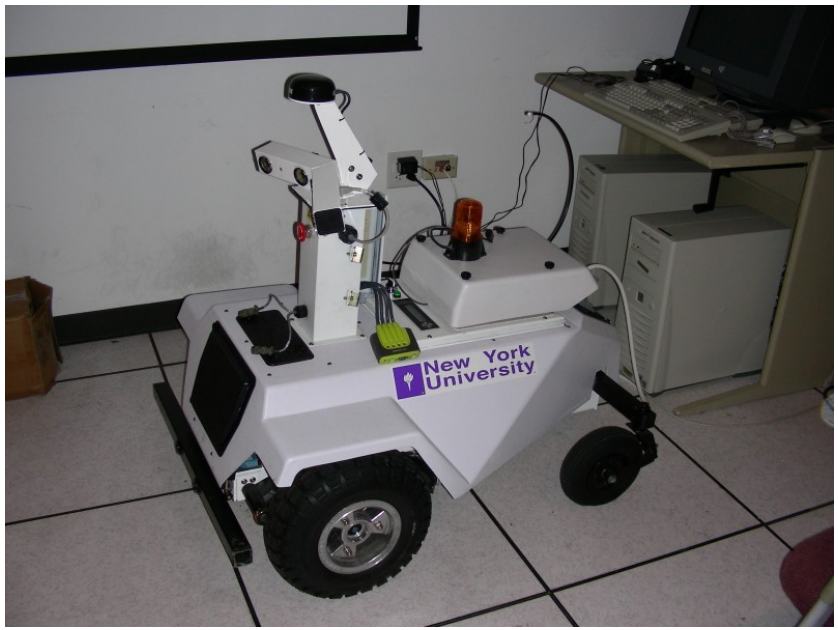
# Preliminary Results on C-101

- C-101, 30 training samples/category
- Single Stage: 64 filters, 9x9 → tanh or shrinkage → abs → local contrast normalization → 10x10 pooling → 5x5 subsampling → multinomial logistic regression
- With patch-level unsupervised training (tanh): 52.2%
- With convolutional unsupervised training (tanh): 56.0%
- With patch-level unsupervised training (shrinkage): 53.0%
- With convolutional unsupervised training (shrinkage): 57.0%
- Two Stages:
- With patch-level unsupervised training (tanh): 65.5%
- With convolutional unsupervised training (tanh): 69.7%

# Deep Learning for Mobile Robot Vision

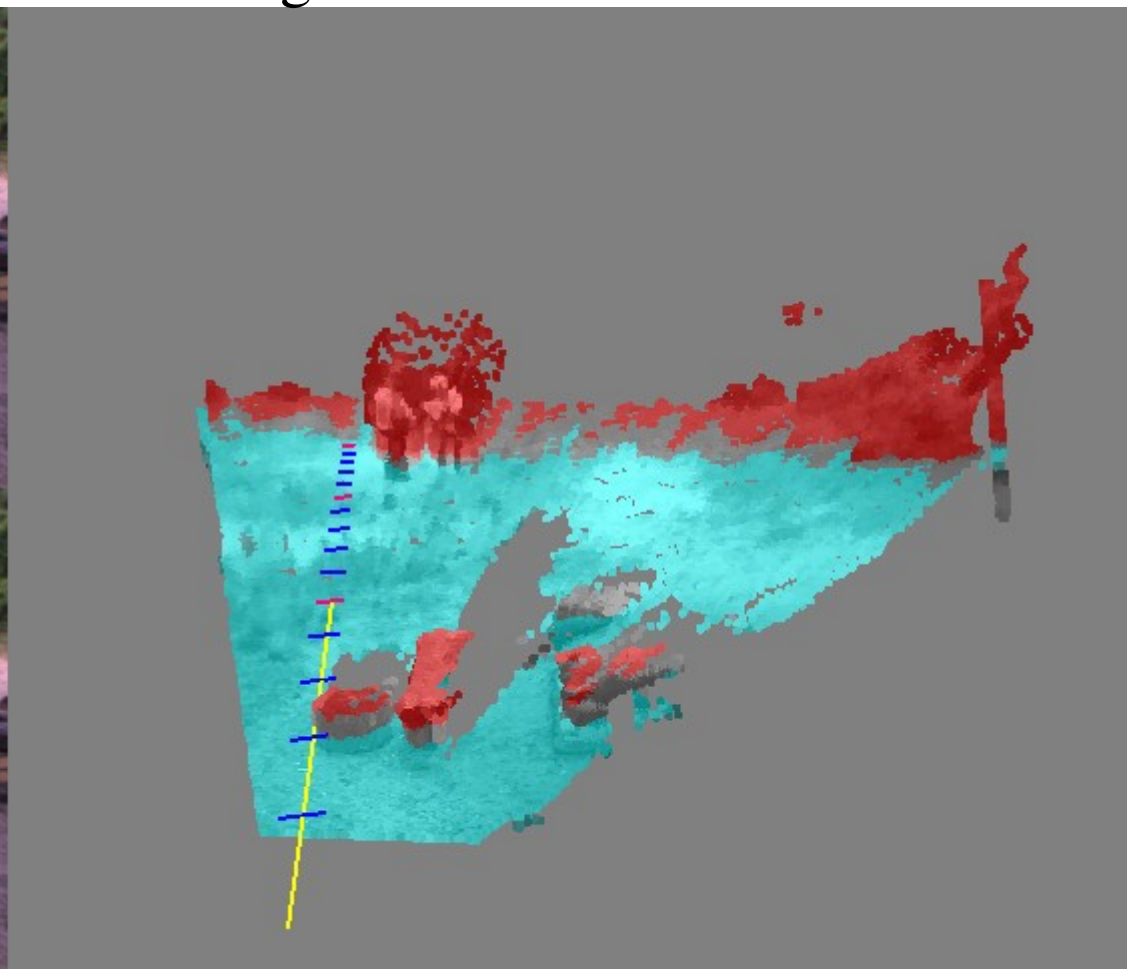
# DARPA/LAGR: Learning Applied to Ground Robotics

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




# Obstacle Detection

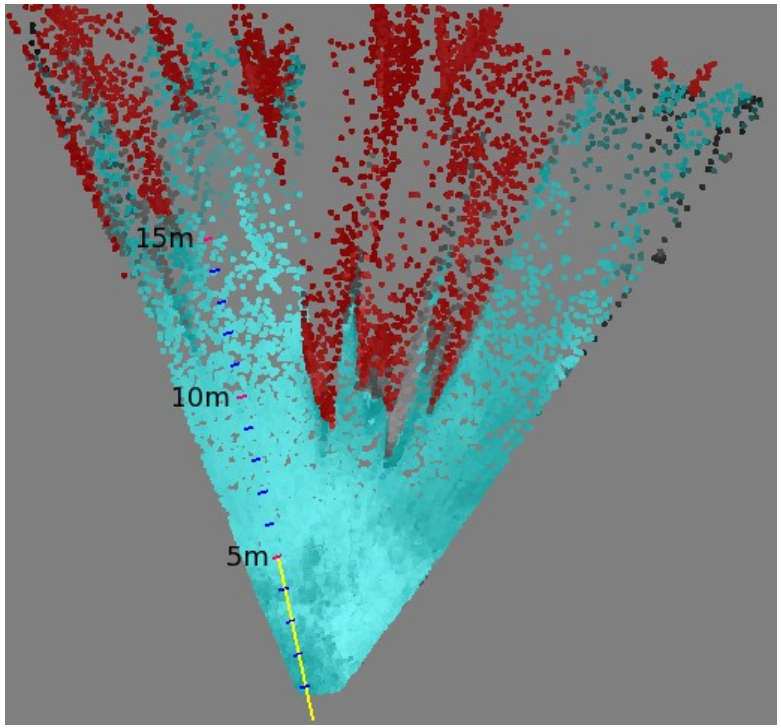
Obstacles overlaid with camera image



Camera image

Detected obstacles (red)

# Navigating to a goal is hard...



stereo perspective

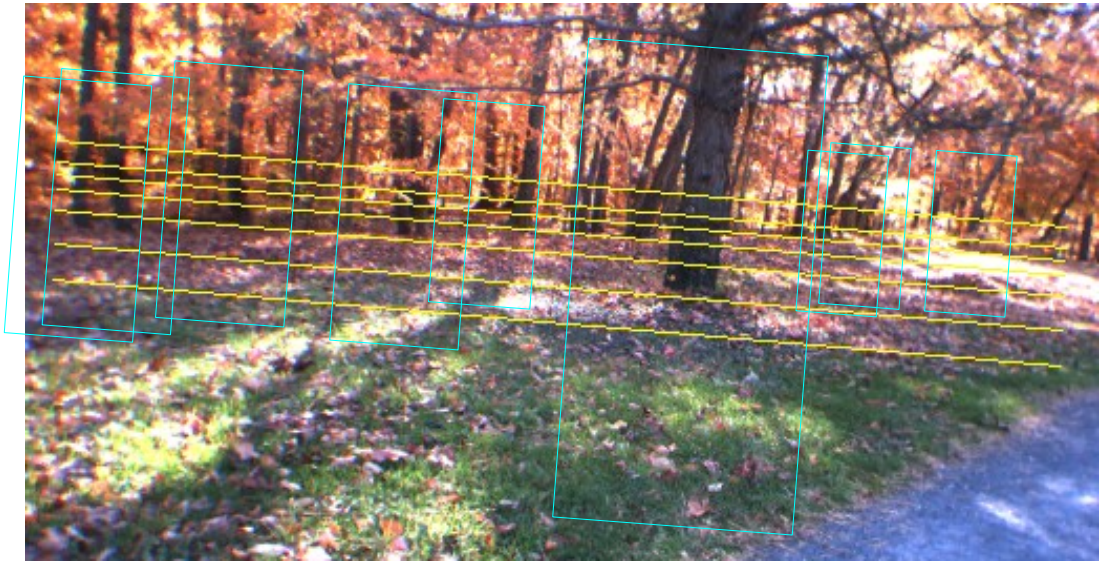
human perspective

**especially in a snowstorm.**

# Self-Supervised Learning

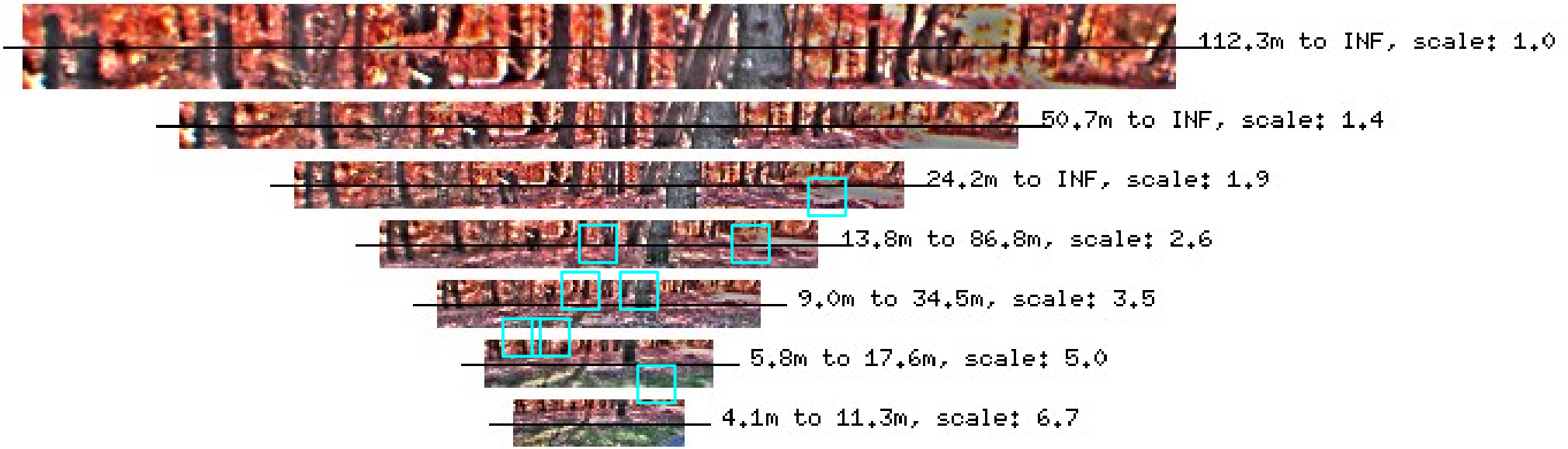
- Stereo vision tells us what nearby obstacles look like
- Use the labels (obstacle/traversible) produced by stereo vision to train a monocular neural network
- Self-supervised "near to far" learning

# Long Range Vision: Distance Normalization



**Pre-processing** (125 ms)

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



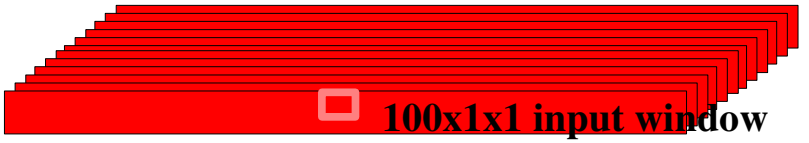
# Convolutional Net Architecture

- Operates on 12x25 YUV windows from the pyramid

Logistic regression 100 features -> 5 classes

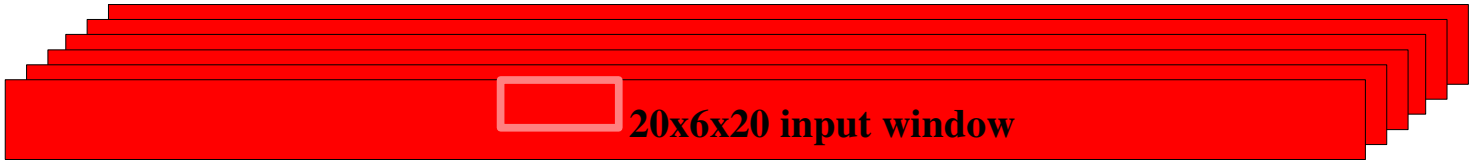
100 features per

3x12x25 input window




Convolutions with 6x5 kernels


Pooling/subsampling with 1x4 kernels

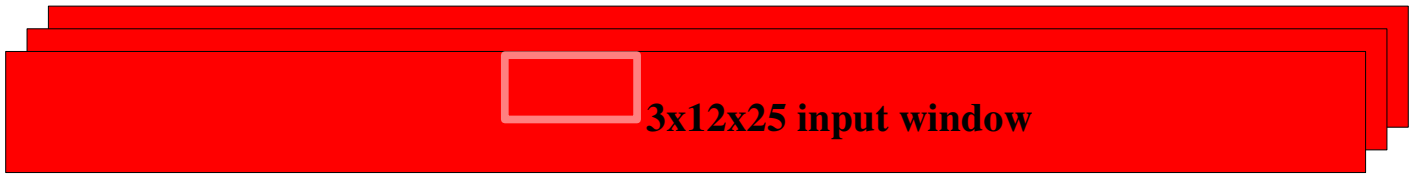



Convolutions with 7x6 kernels

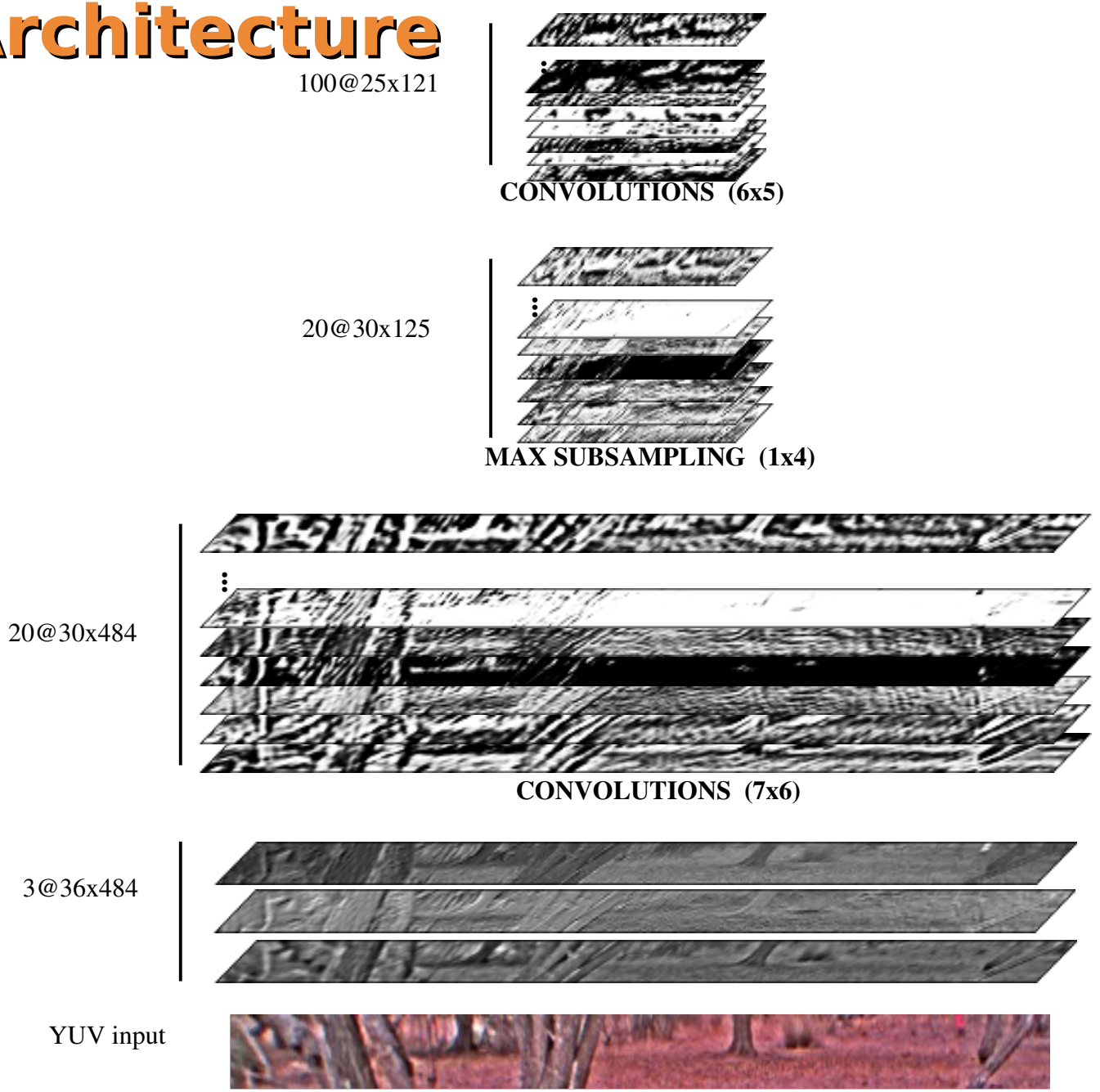
YUV image band

20-36 pixels tall,

36-500 pixels wide



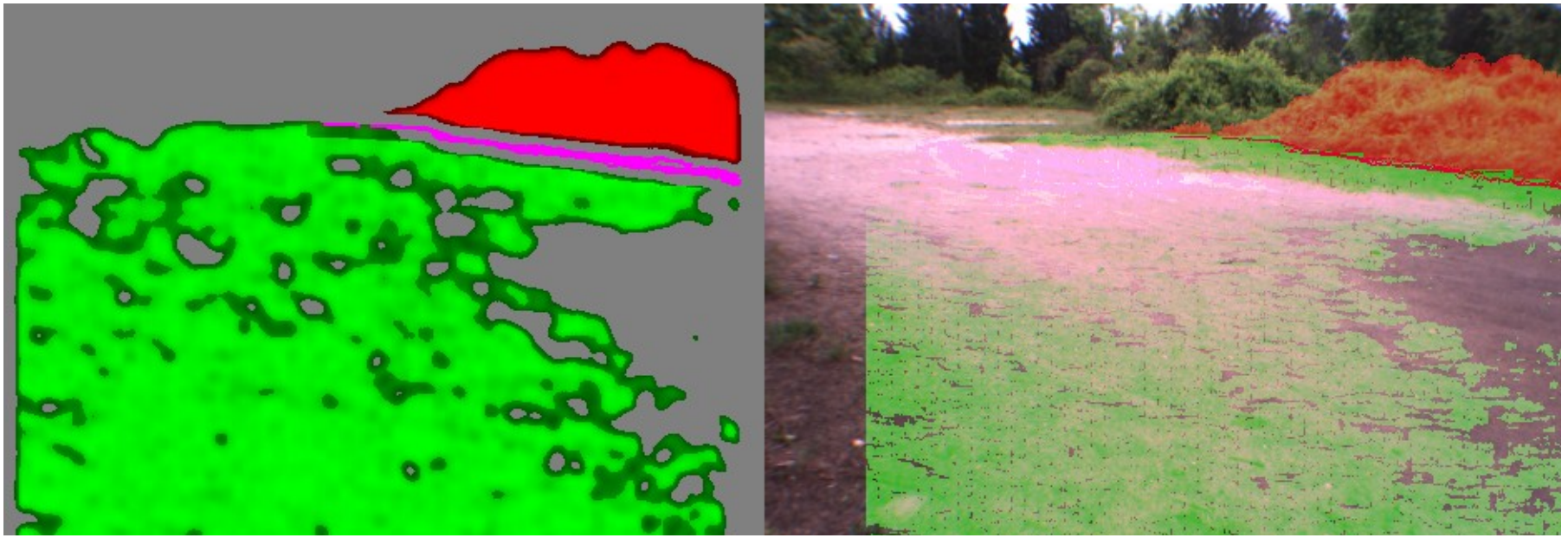
# Convolutional Net Architecture

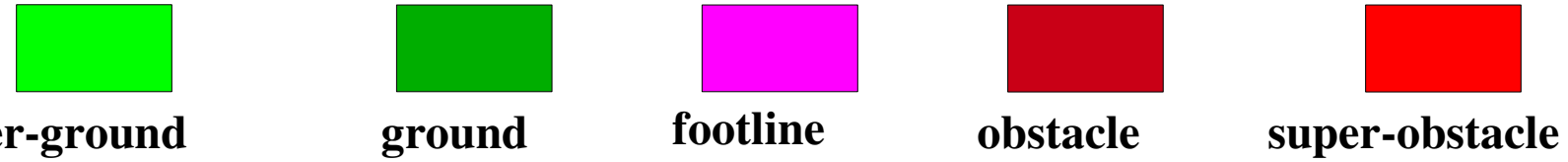


# Long Range Vision: 5 categories

## Online Learning (52 ms)

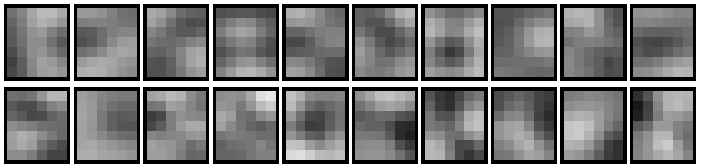
- Label windows using stereo information – 5 classes

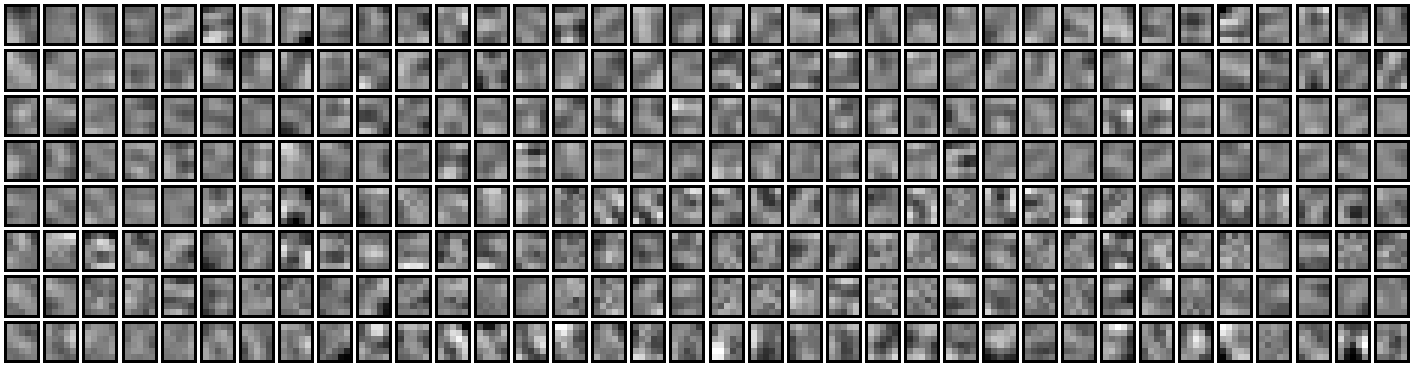




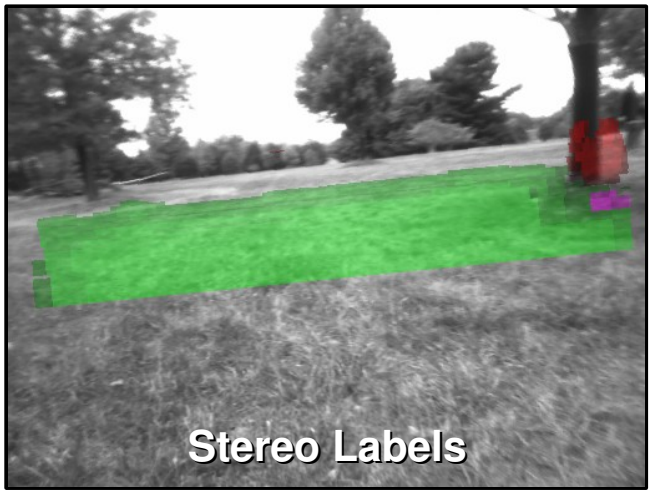
# Trainable Feature Extraction

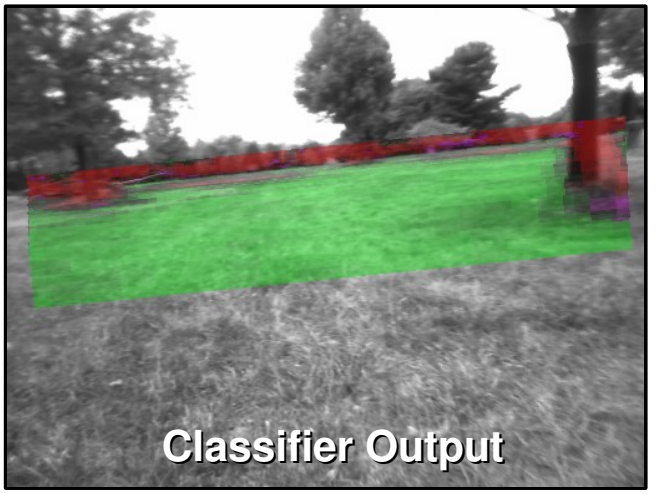
- "Deep belief net" approach to unsupervised feature learning
- Two stages are trained in sequence
  - each stage has a layer of convolutional filters and a layer of horizontal feature pooling.
  - Naturally shift invariant in the horizontal direction
- Filters of the convolutional net are trained so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



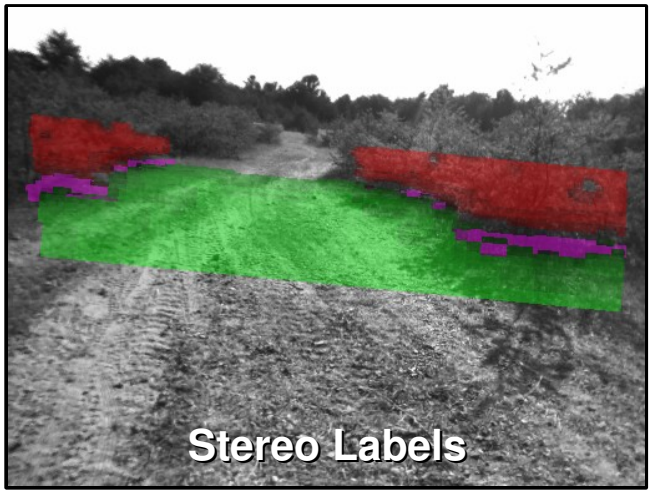


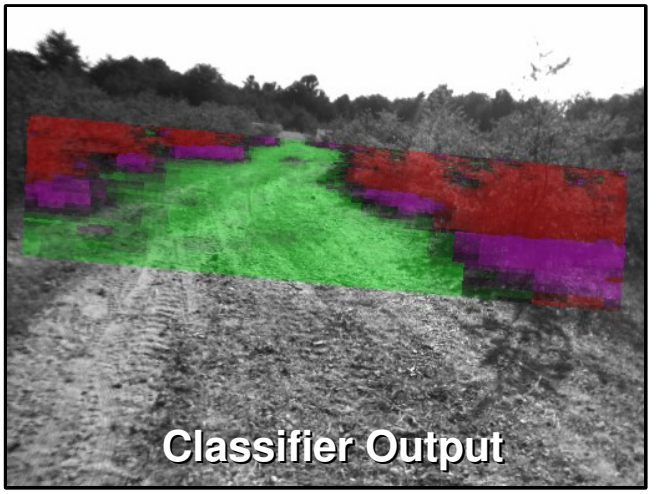
# Long Range Vision Results

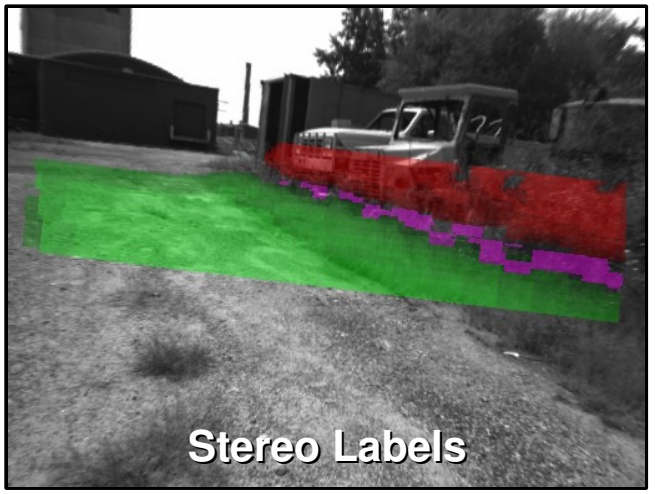



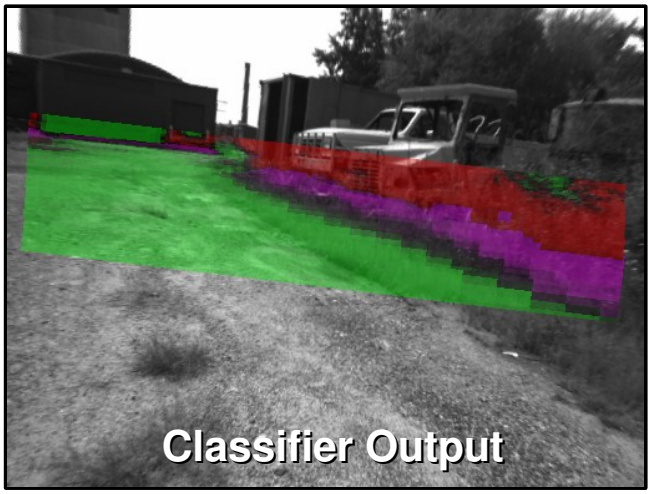




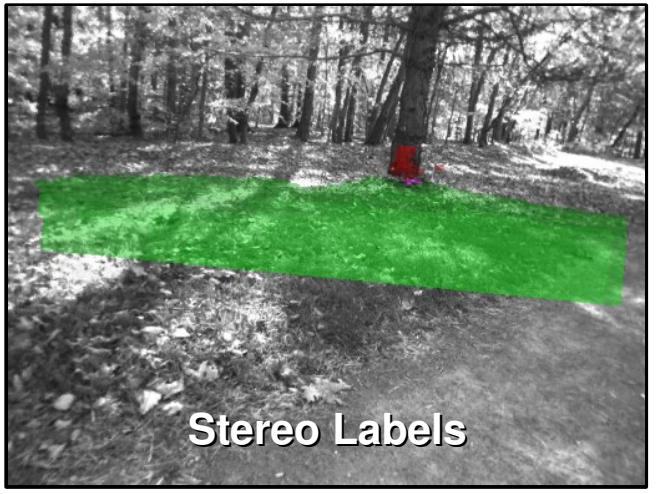



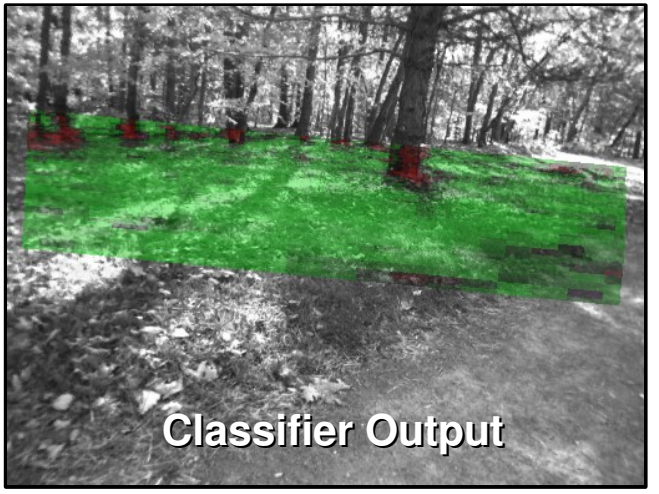
# Long Range Vision Results

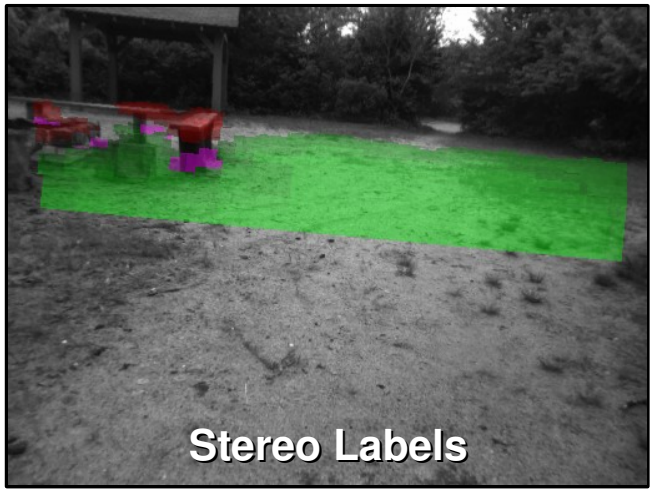



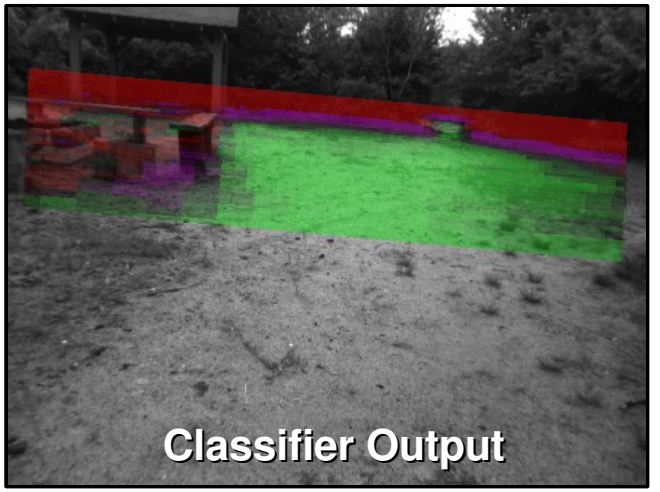




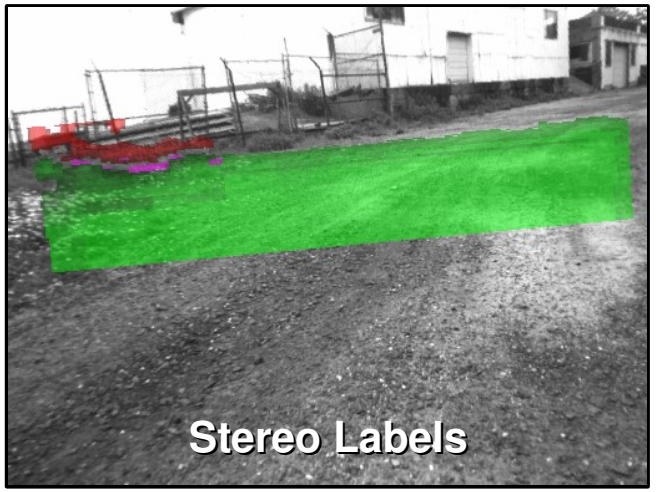



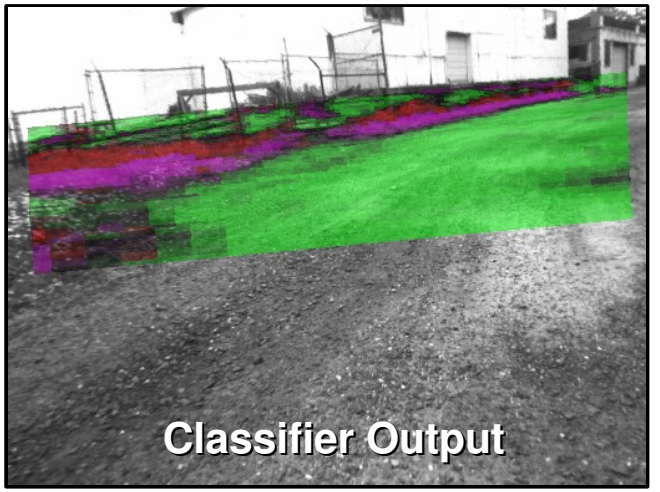
# Long Range Vision Results

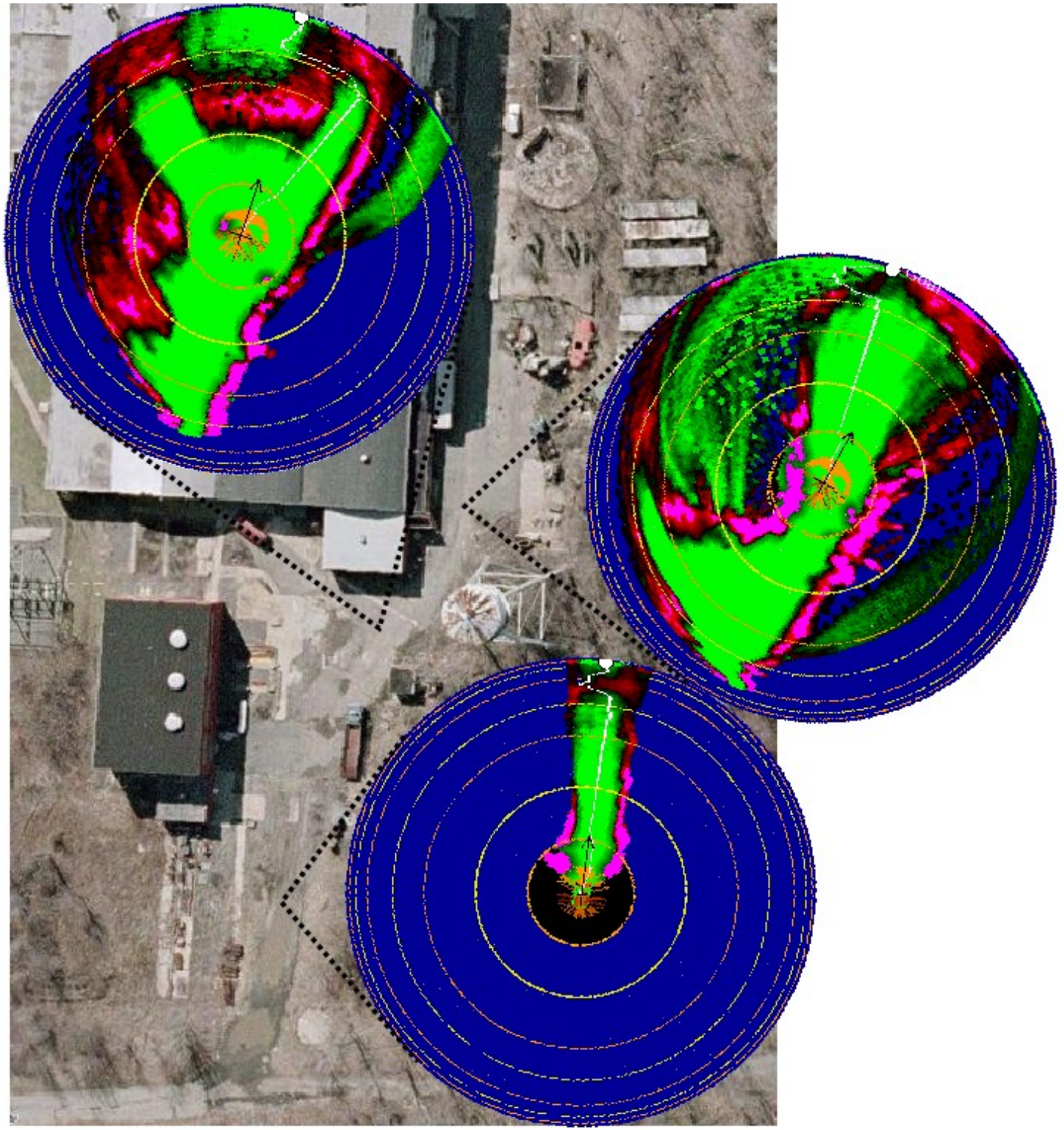



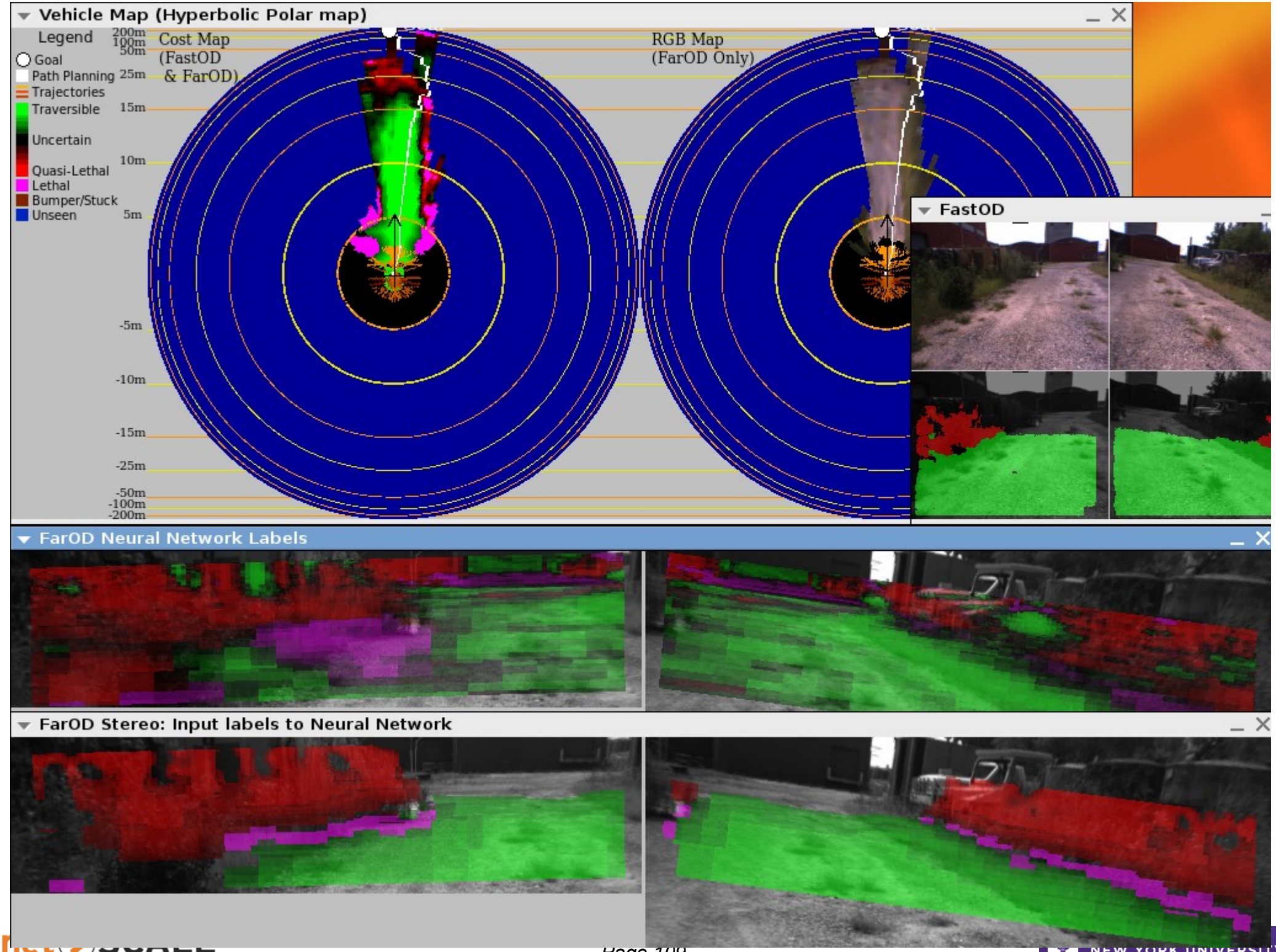



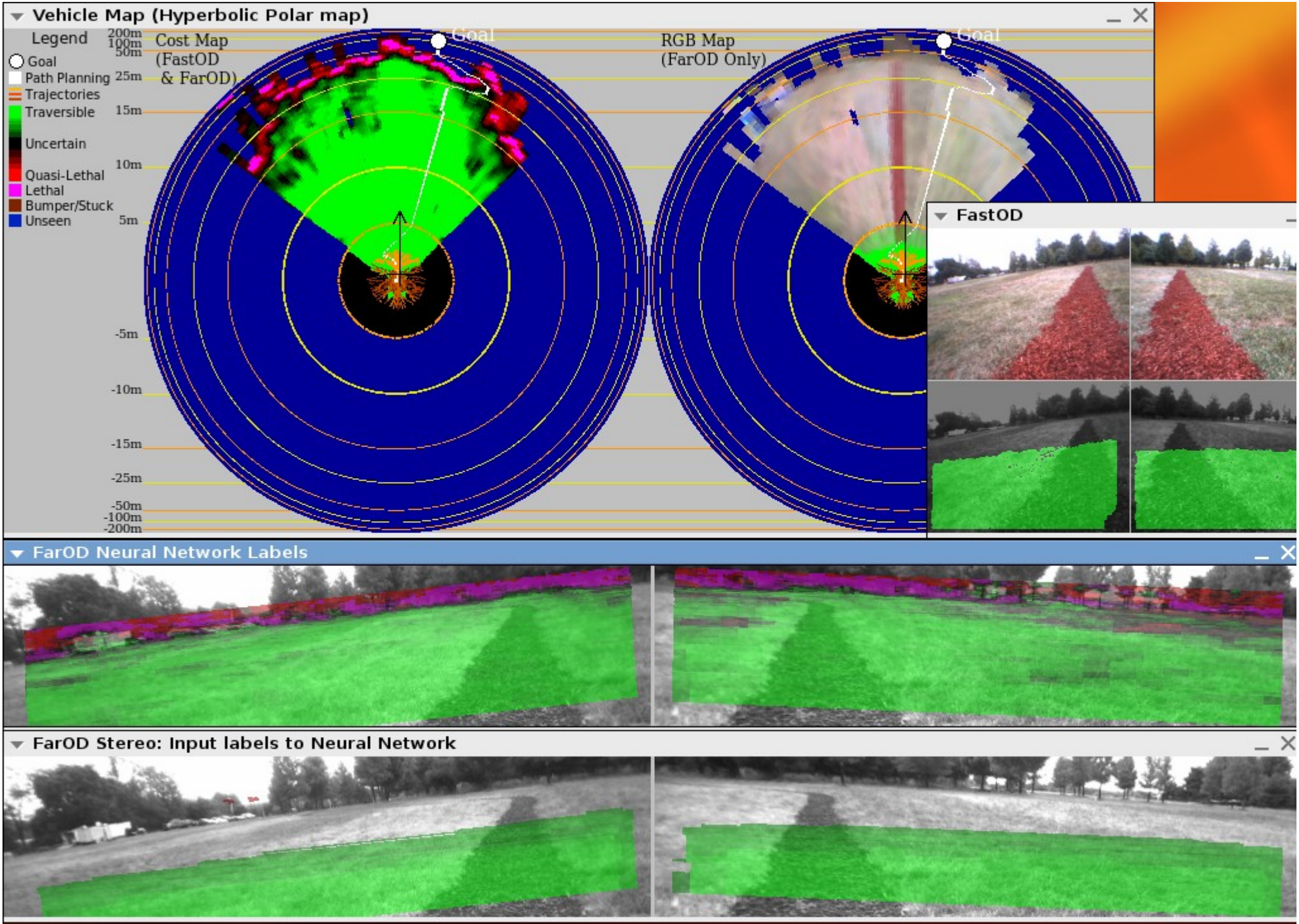


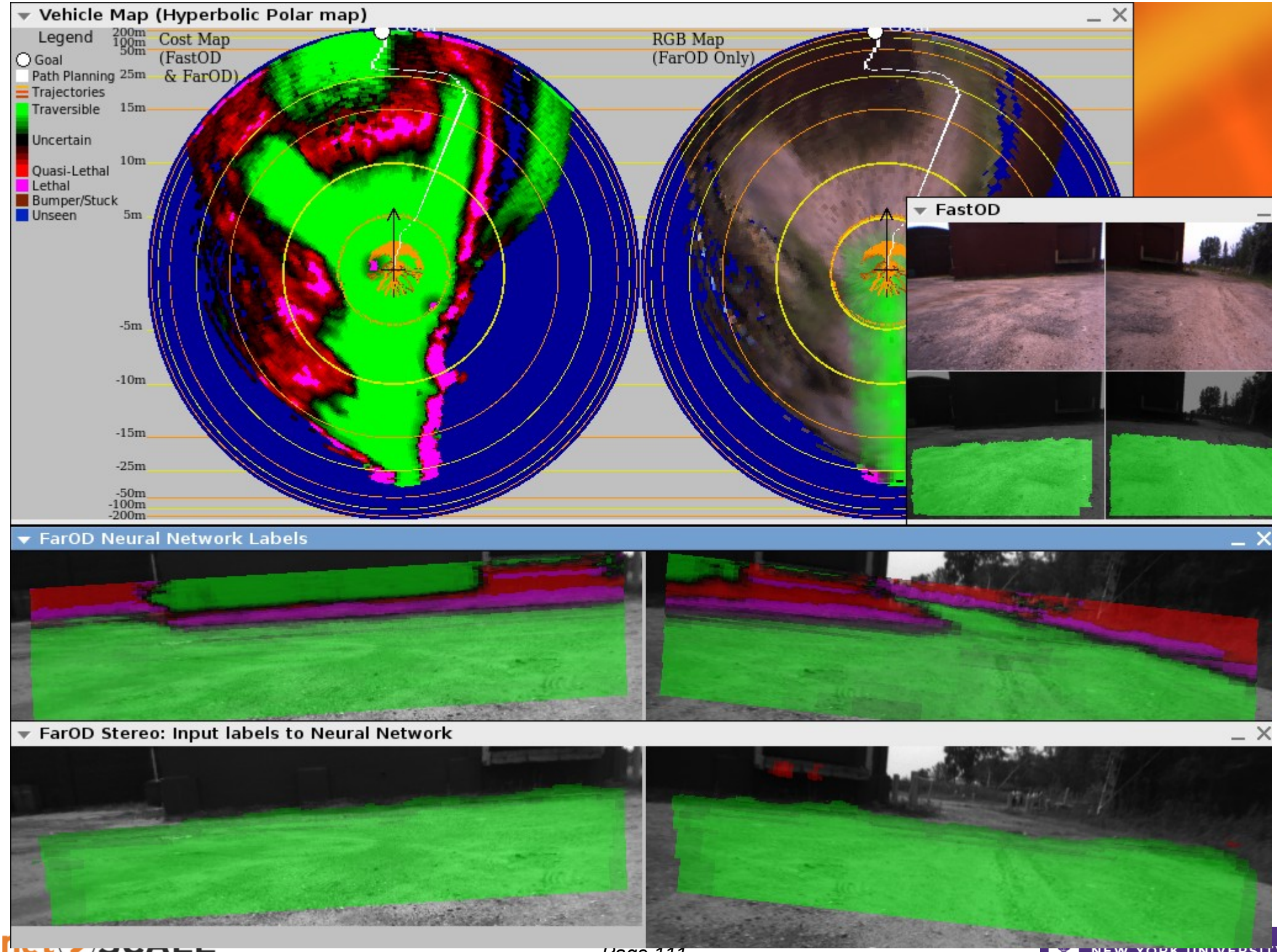


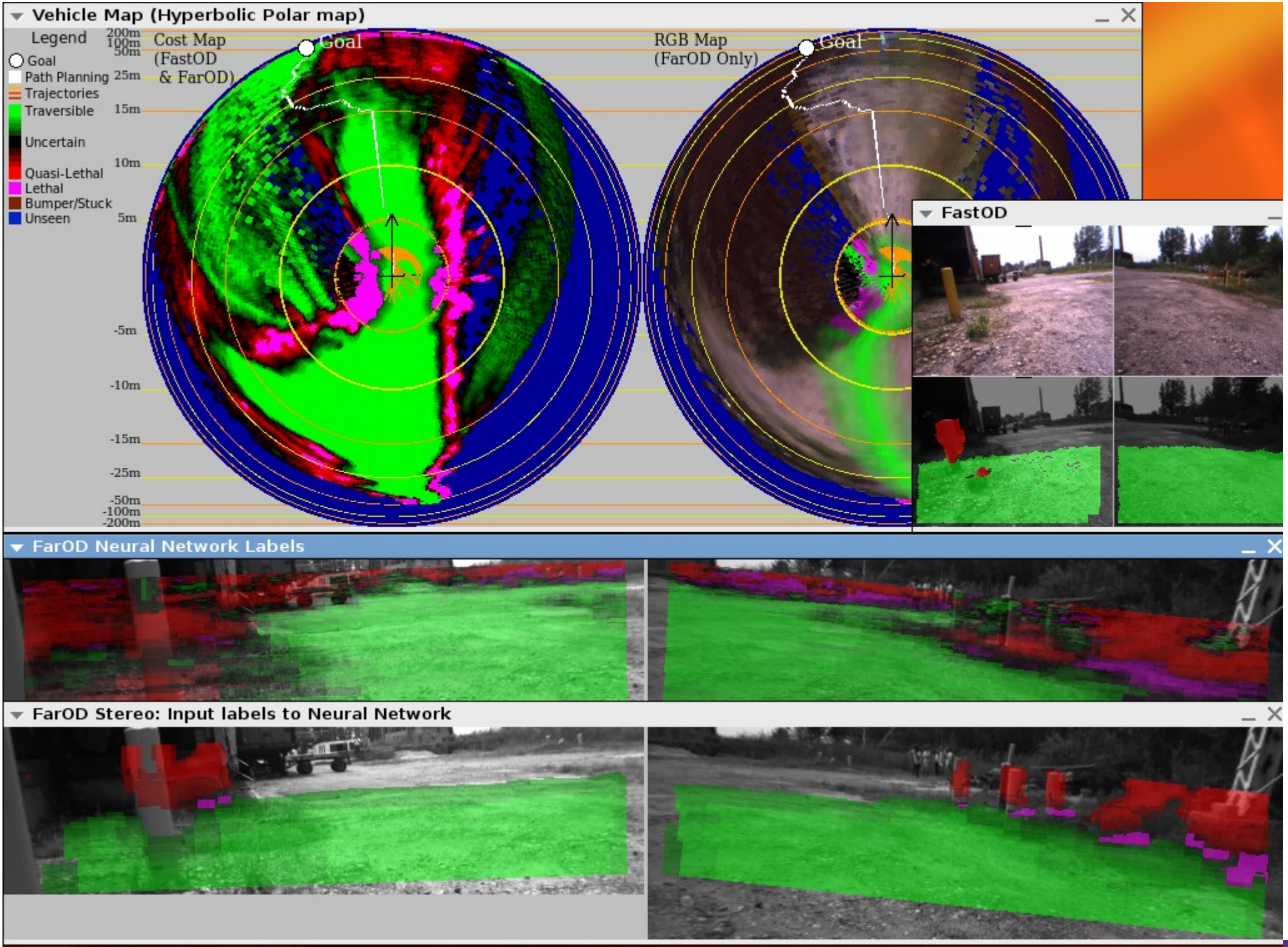



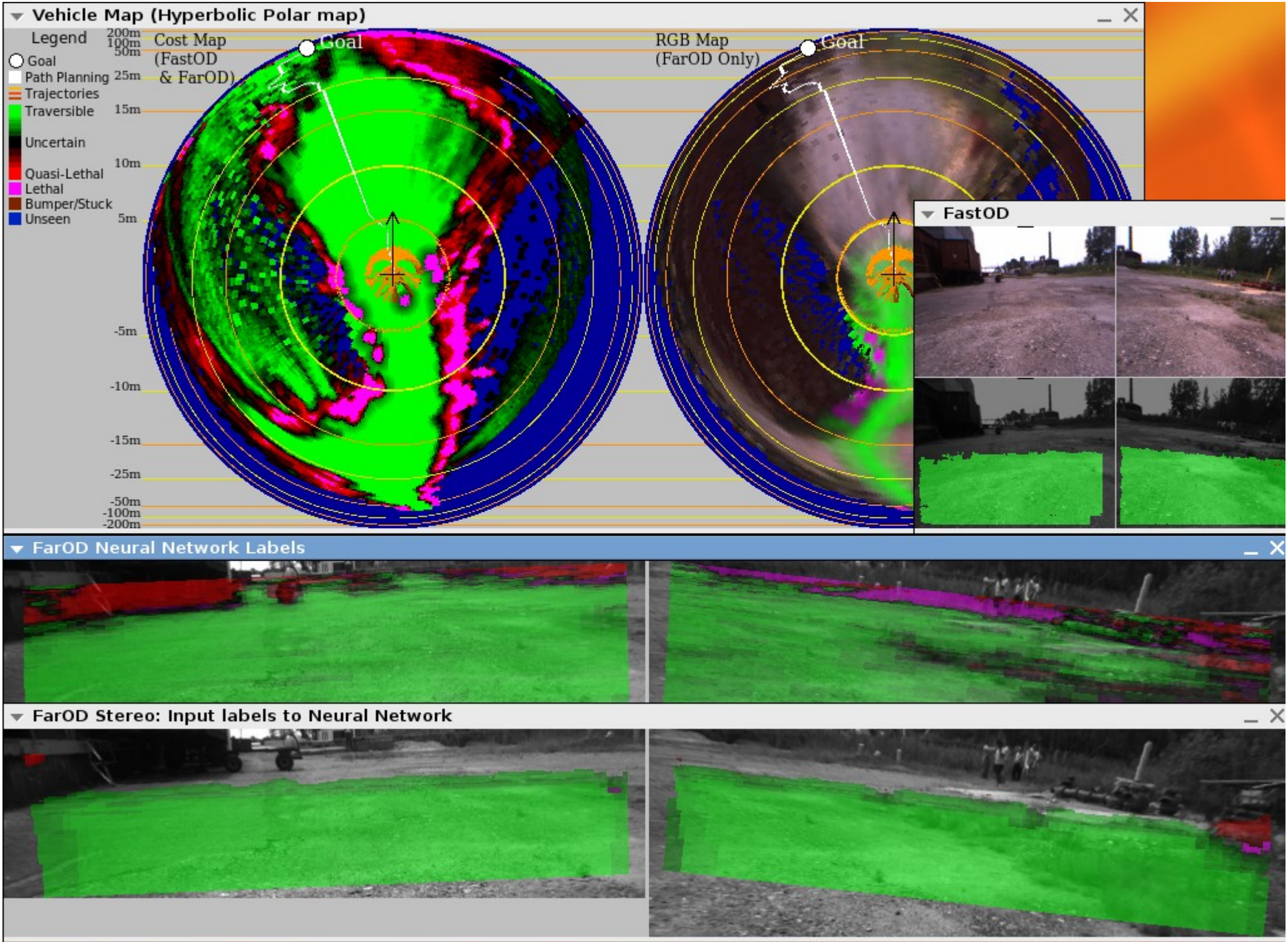



Vehicle Map (Hyperbolic Polar map)
Legend
Goal
Path Planning
Trajectories
Traversible
Uncertain
Quasi-Lethal
Lethal
Bumper/Stuck
Unseen
200m
100m
50m
25m
15m
10m
5m
-5m
-10m
-15m
-25m
-50m
-100m
-200m
Cost Map
(FastOD
& FarOD)
RGB Map
(FarOD Only)
FastOD
FarOD Neural Network Labels
FarOD Stereo: Input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


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


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


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


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

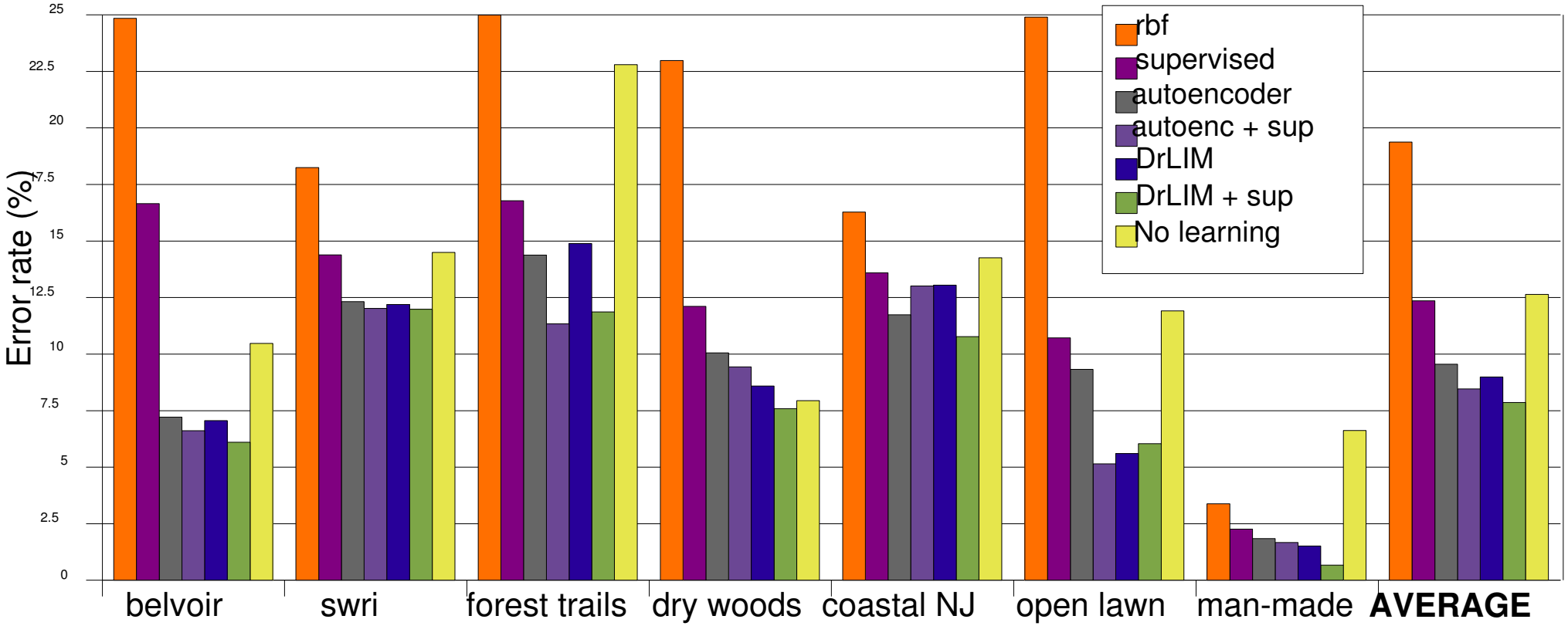
# Feature Learning for traversability prediction (LAGR)

**Comparing**

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

**Testing on hand-labeled groundtruth frames – binary labels**

Comparison of Feature Extractors on Groundtruth Data



# Collaborators

- **Current PhD students:**
  - Y-Lan Boureau, Koray Kavukcuoglu, Pierre Sermanet
- **Former PhD students:**
  - Raia Hadsell, Fu-Jie Huang, Marc'Aurelio Ranzato
- **Postdocs and Research Scientists**
  - Clément Farabet, Karol Gregor, Marco Scoffier
- **Senior Collaborators**
  - Rob Fergus (NYU): invariant feature learning
  - Eugenio Culurciello (Yale): FPGA/ASIC design
  - Yoshua Bengio (U. Montreal): deep learning
  - Leon Bottou (NEC Labs): handwriting recognition
  - Jean Ponce (ENS/INRIA), Francis Bach (ENS/INRIA): sparse coding.

# The End