

Learning Hierarchies of Invariant Features

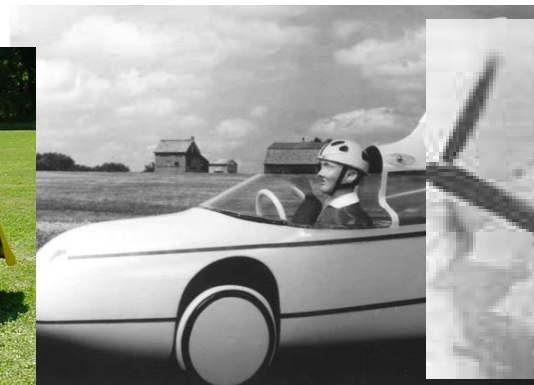
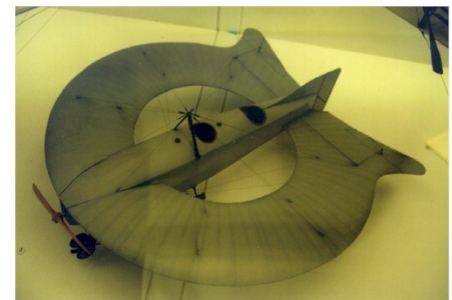
Yann LeCun

Courant Institute of Mathematical Sciences
and

Center for Neural Science,
New York University

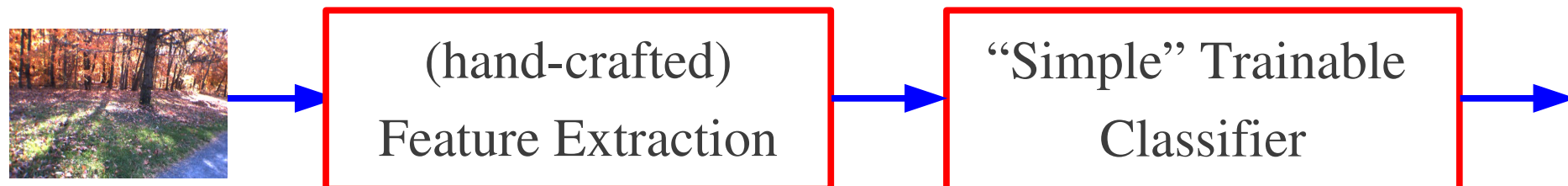
Challenges for Machine Learning, Vision, Signal Processing, AI, Neuroscience

- How can learning build a perceptual system?
- How do we learn representations of the perceptual world?
- In ML/CV/ASR/MIR: How do we learn features (not just classifiers)?
- With good representations, we can learn categories from just a few examples.
- ML has neglected the question of learning representations, relying instead on domain expertise to engineer features and kernels.
- Deep Learning addresses the problem of learning representations
- Goal 1: biologically-plausible methods for deep learning
- Goal 2: representation learning for computer perception

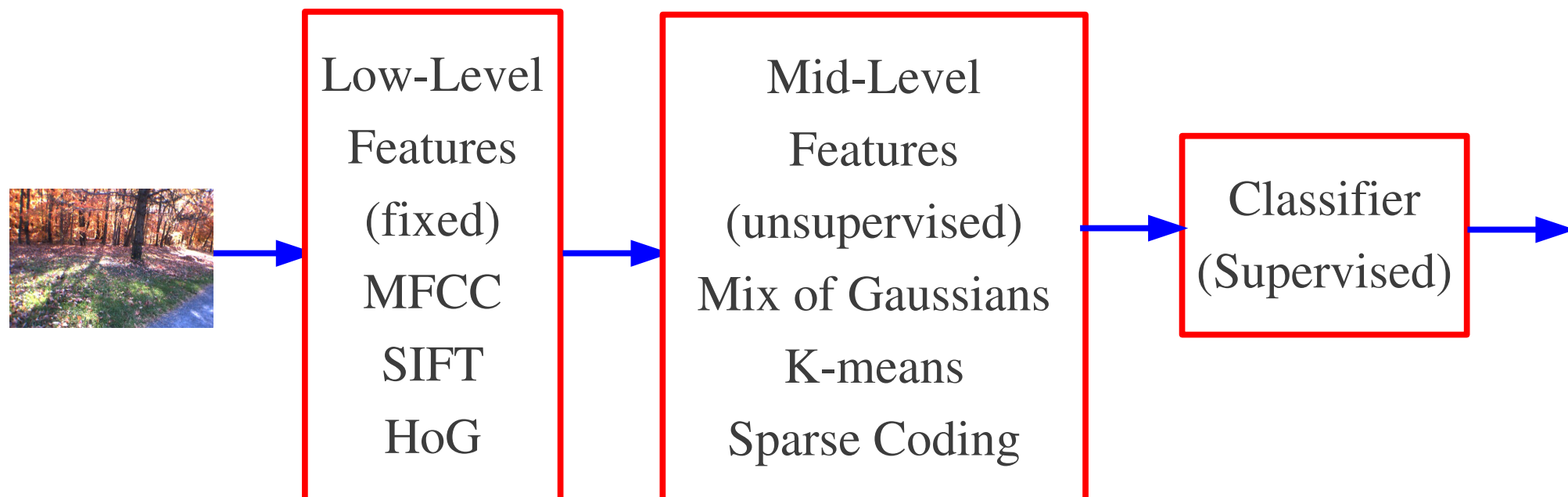


Architecture of “Mainstream” Systems

- Traditional way: handcrafted features + classifier

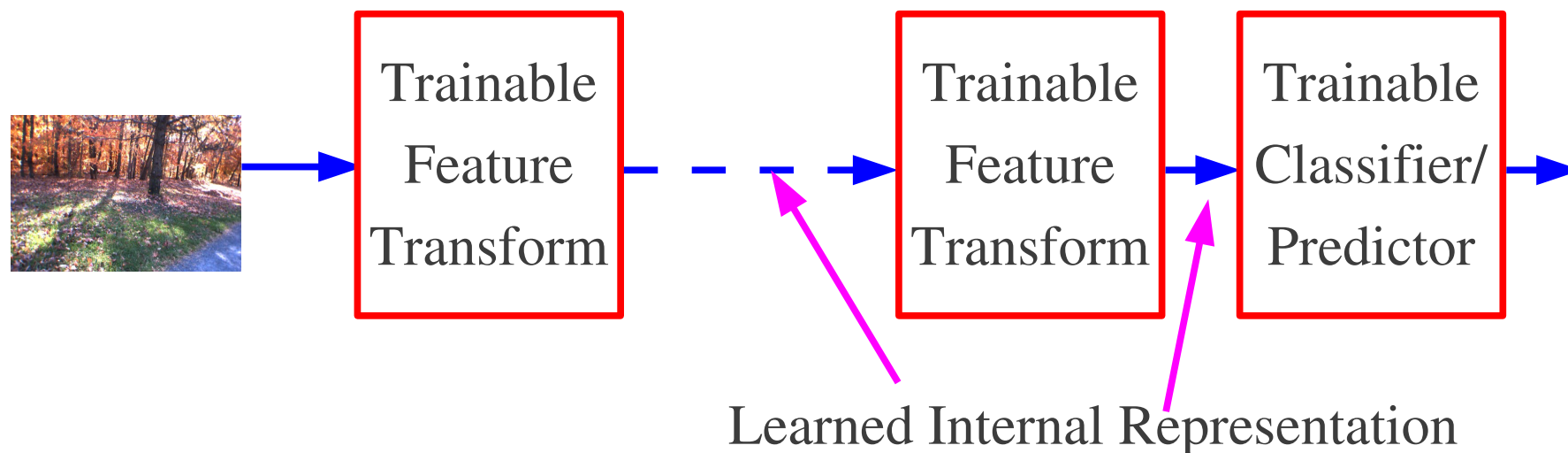


- Mainstream Approaches to Image and Speech Recognition



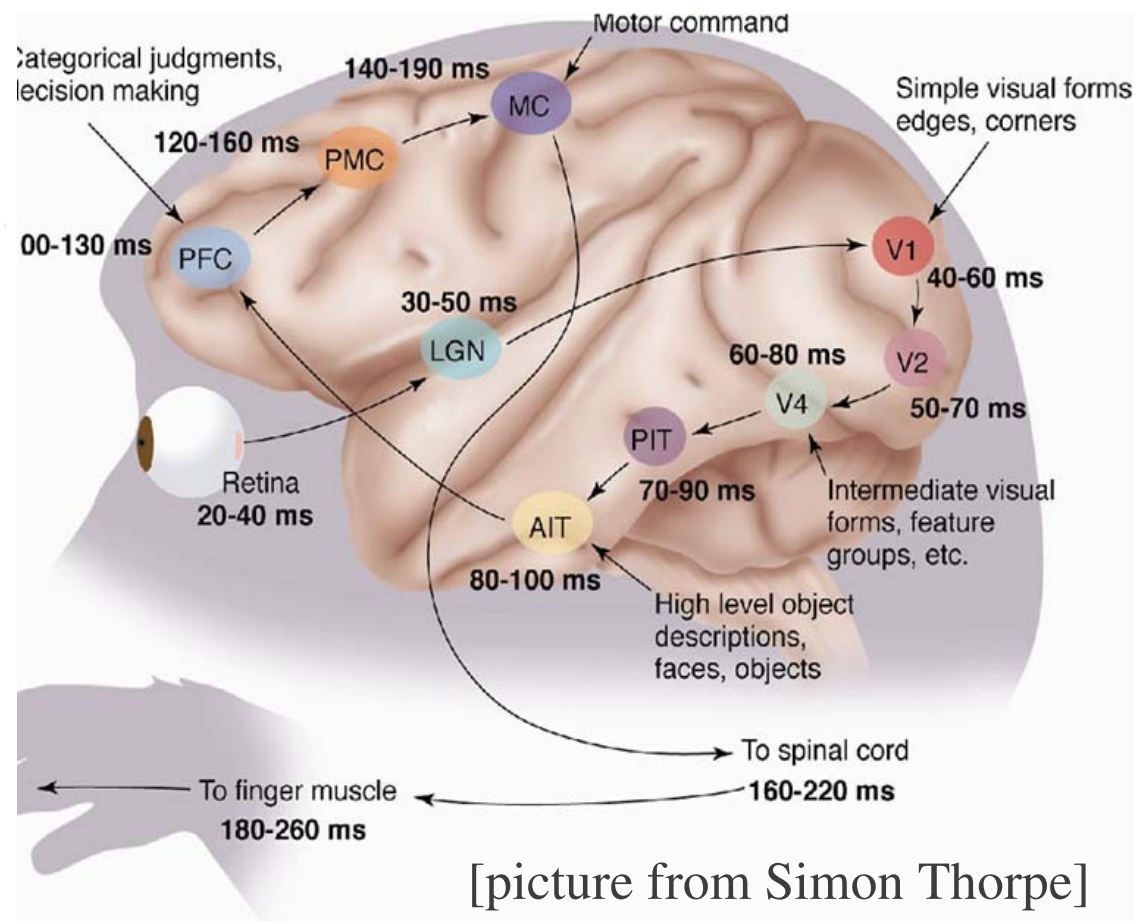
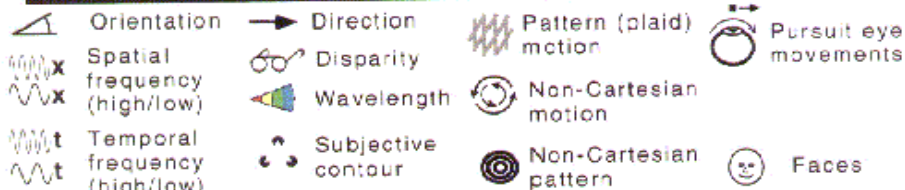
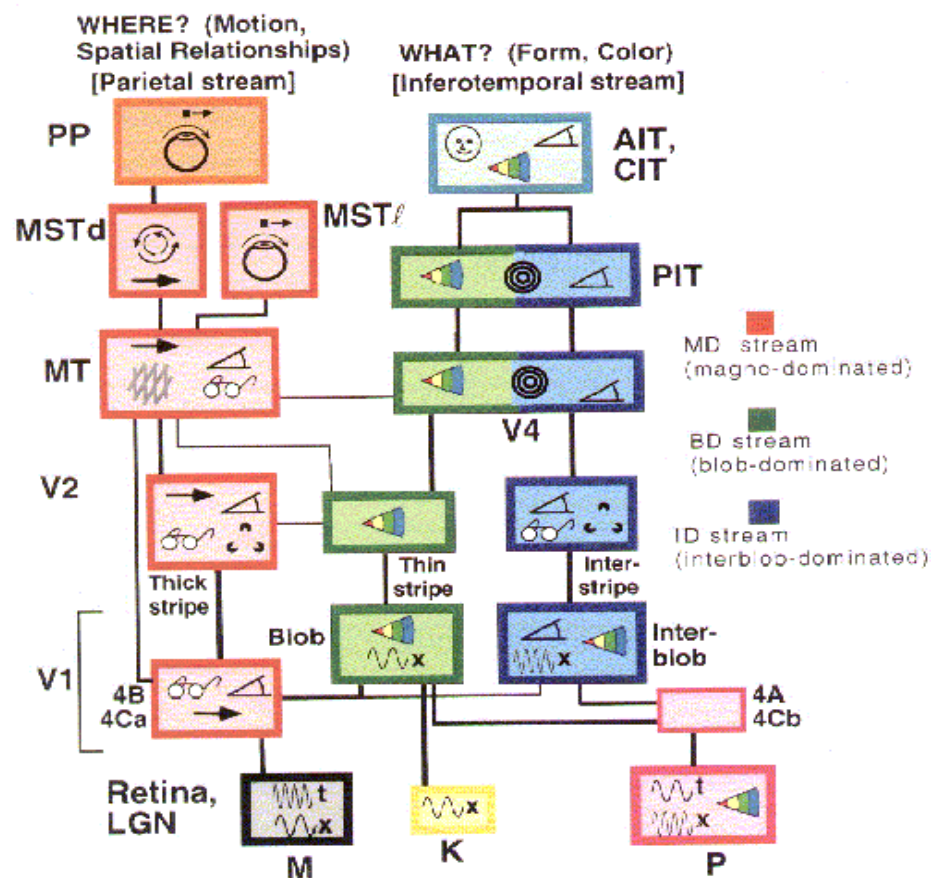
Trainable Feature Hierarchies

- Why can't we make all the modules trainable?
- Proposed way: hierarchy of trained features



The Mammalian Visual Cortex is Hierarchical

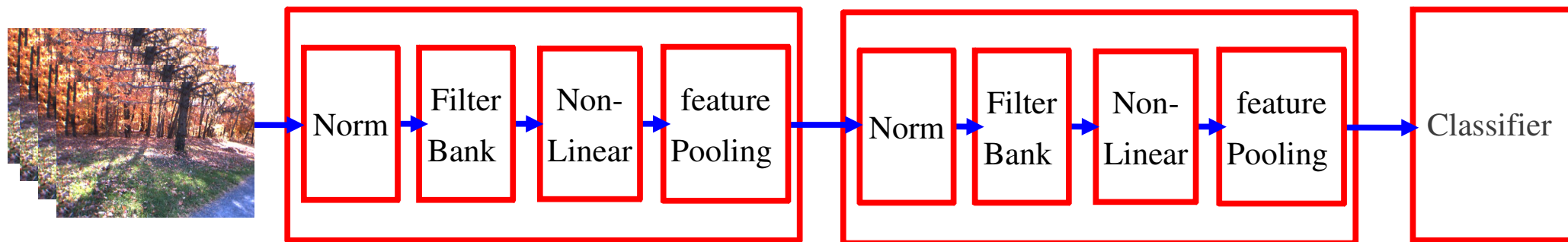
- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT



[picture from Simon Thorpe]

[Gallant & Van Essen]

Feature Transform = Normalization → Filter Bank → Non-Linearity → Pooling



Stacking multiple stages of

▶ [Normalization → Filter Bank → Non-Linearity → Pooling].

Normalization: variations on whitening

- ▶ Subtractive: average removal, high pass filtering
- ▶ Divisive: local contrast normalization, variance normalization

Filter Bank: dimension expansion, projection on overcomplete basis

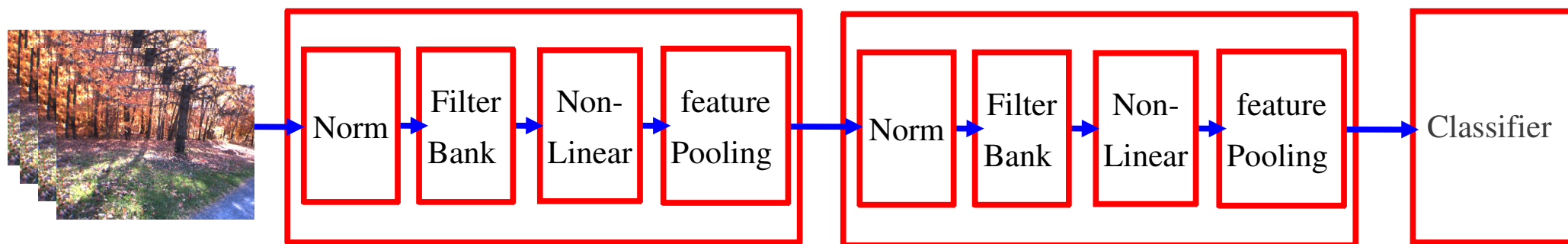
Non-Linearity: sparsification, saturation, lateral inhibition....

- ▶ Component-wise shrinkage or tanh, winner-takes-all

Pooling: aggregation over space or feature type, subsampling

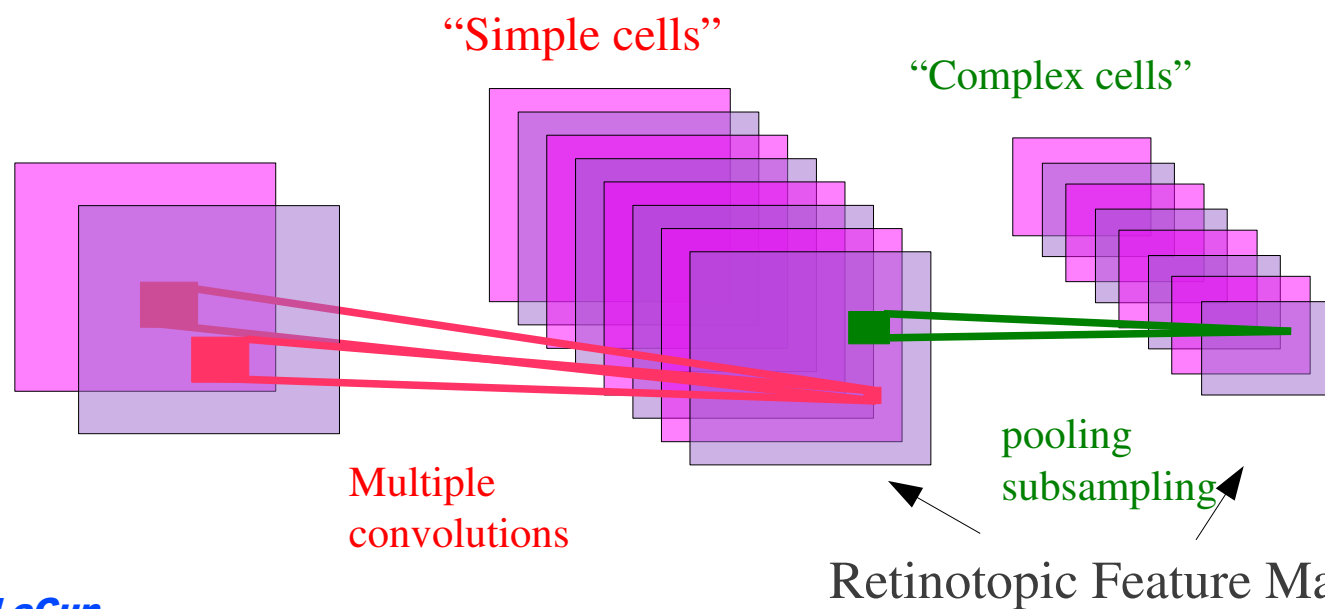
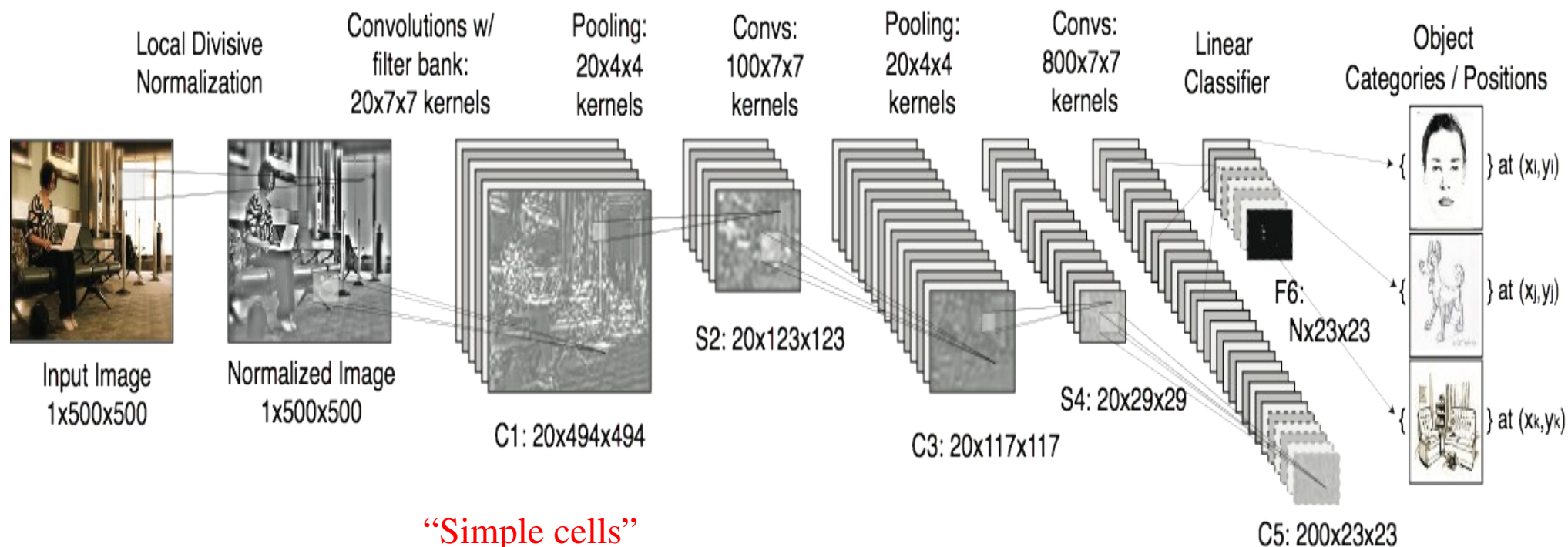
- ▶ $AVERAGE: \frac{1}{K} \sum_i X_i$; $MAX: \max_i X_i$; $L_p: \sqrt[p]{X_i^p}$; $PROB: \frac{1}{b} \log \left(\sum_i e^{b X_i} \right)$

Feature Transform = Normalization → Filter Bank → Non-Linearity → Pooling



- Filter Bank → Non-Linearity = Non-linear embedding in high dimension
- Feature Pooling = contraction, dimensionality reduction, smoothing
- Learning the filter banks at every stage
- Creating a hierarchy of features
- Basic elements are inspired by models of the visual (and auditory) cortex
 - Simple Cell + Complex Cell model of [Hubel and Wiesel 1962]
 - Many “traditional” feature extraction methods are based on this
 - SIFT, GIST, HoG, Convolutional networks.....
- [Fukushima 1974-1982], [LeCun 1988-now], [Poggio 2005-now], [Ng 2006-now], many others....

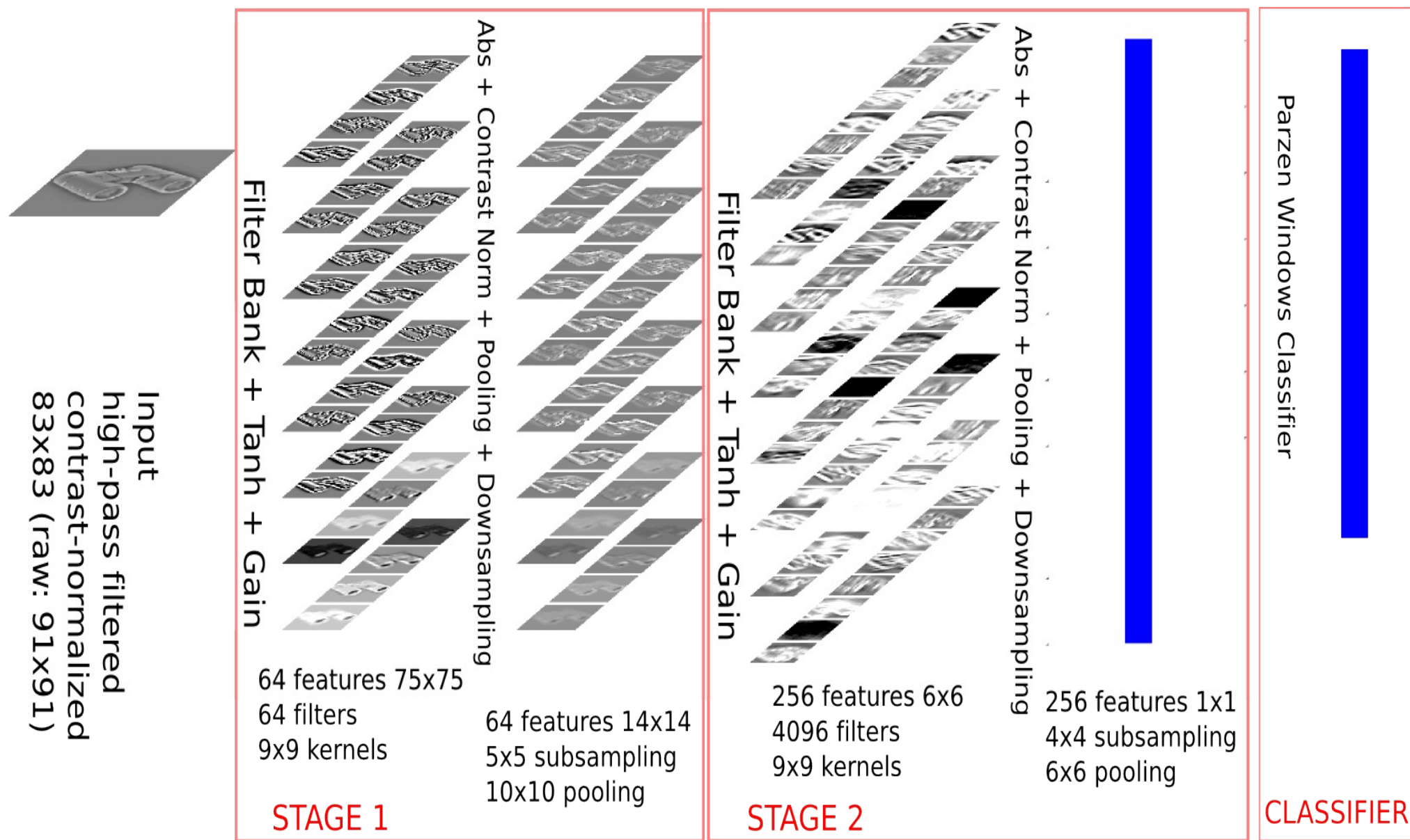
Basic Convolutional Network Architecture



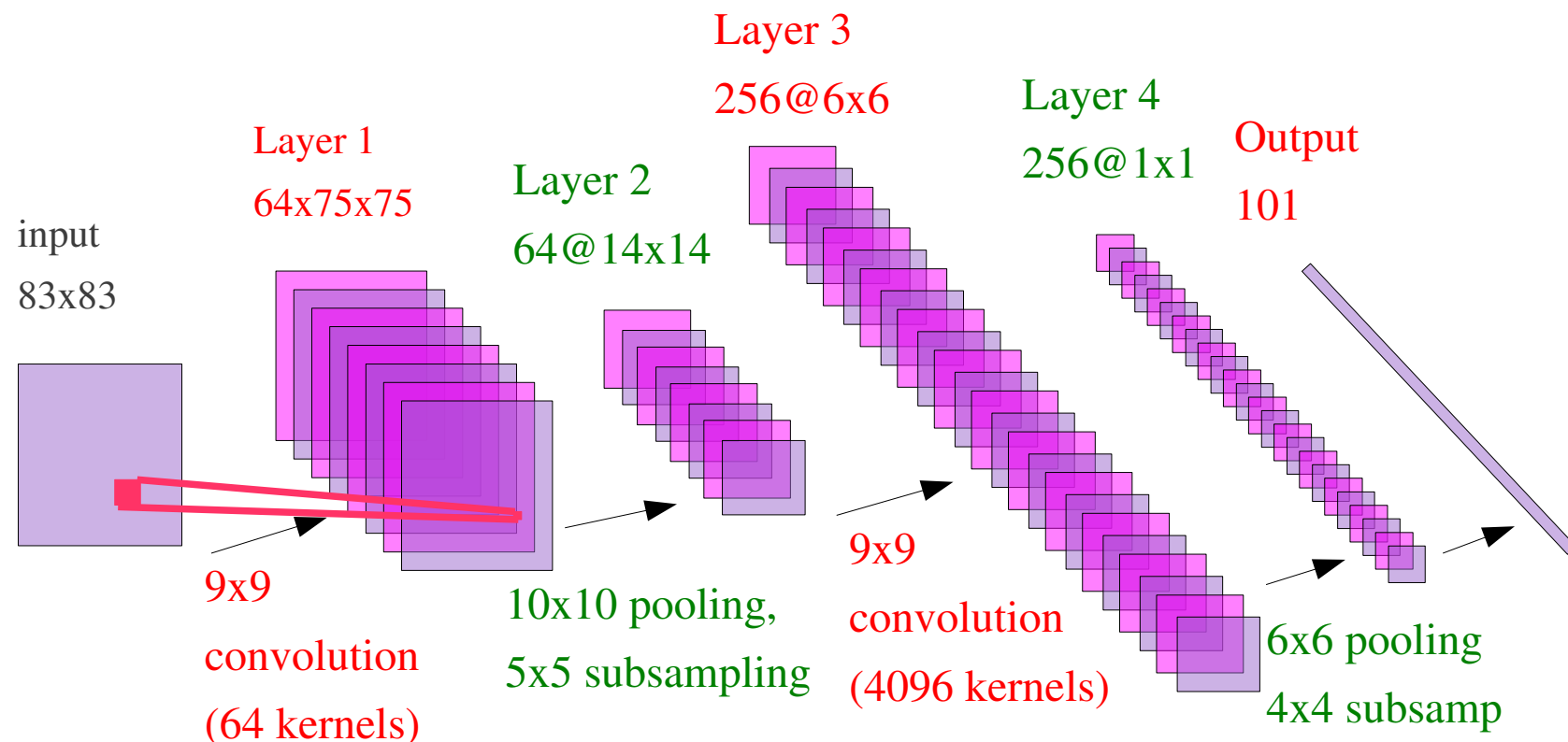
[LeCun et al. 89]

[LeCun et al. 98]

Convolutional Network Architecture



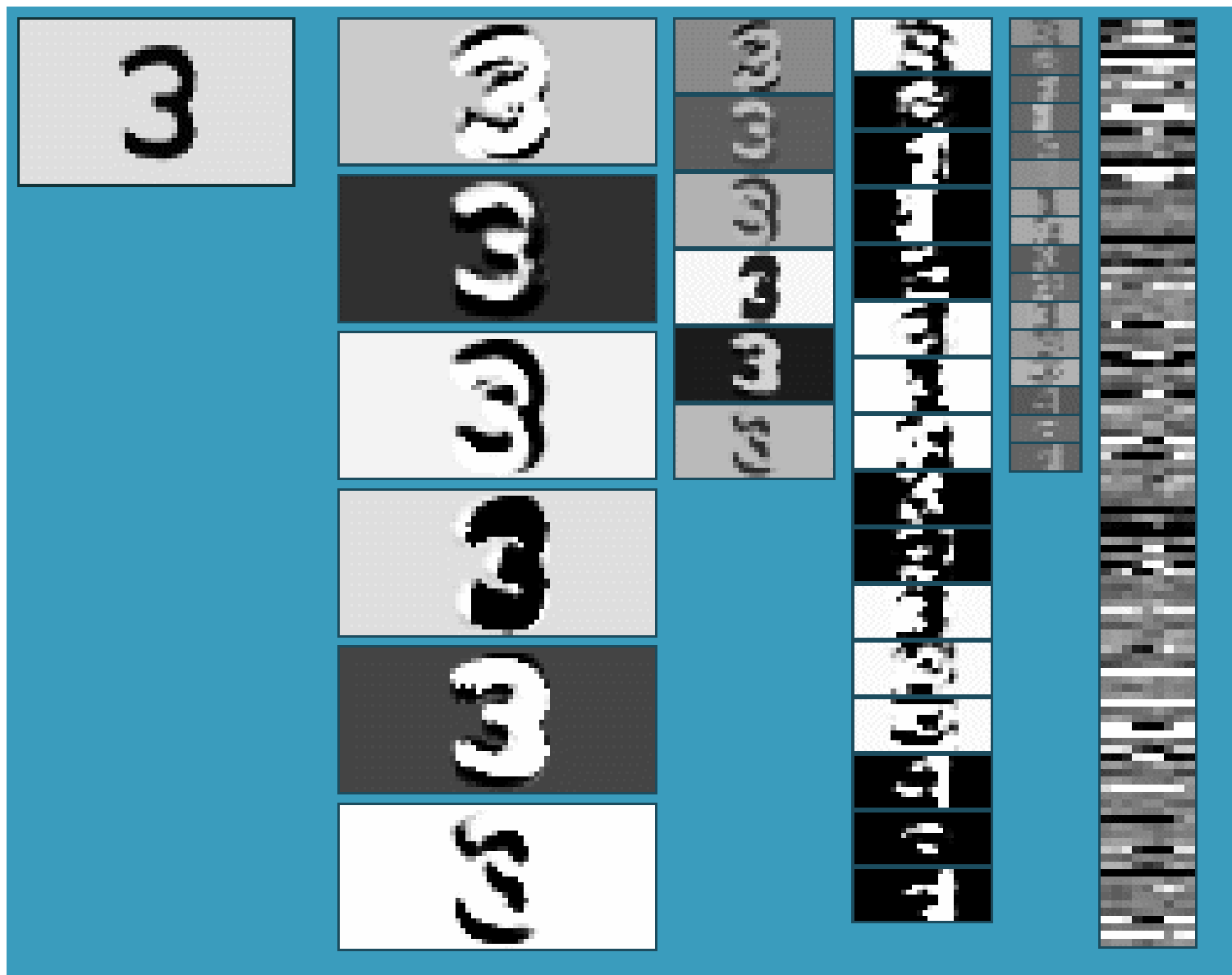
Convolutional Network (ConvNet)



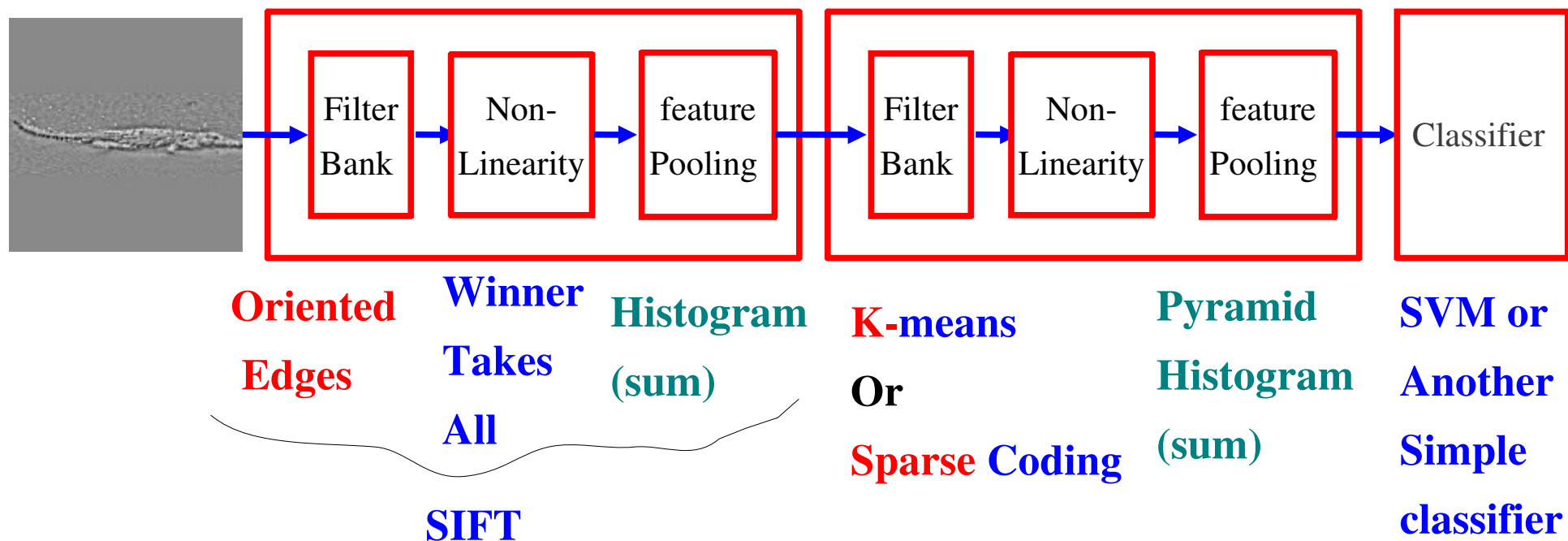
- **Non-Linearity**: shrinkage function, tanh
- **Pooling**: L2, average, max, average→tanh
- **Training**: Supervised (1988-2006), Unsupervised+Supervised (2006-now)

Convolutional Network (vintage 1990)

filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh



"Mainstream" object recognition pipeline 2006-2010: similar to ConvNets



● Fixed low-level Features + unsupervised mid-level features + simple classifier

● Example (on Caltech 101 dataset):

▶ SIFT + Vector Quantization + Pyramid pooling + SVM: **>65%**

● [Lazebnik et al. CVPR 2006]

▶ SIFT + Local Sparse Coding Macrofeat. + Pyr/ pooling + SVM: **>77%**

● [Boureau et al. ICCV 2011]

Other Applications with State-of-the-Art Performance

● Traffic Sign Recognition (GTSRB)

- ▶ German Traffic Sign Reco Bench
- ▶ 97.2% accuracy



● House Number Recognition (Google)

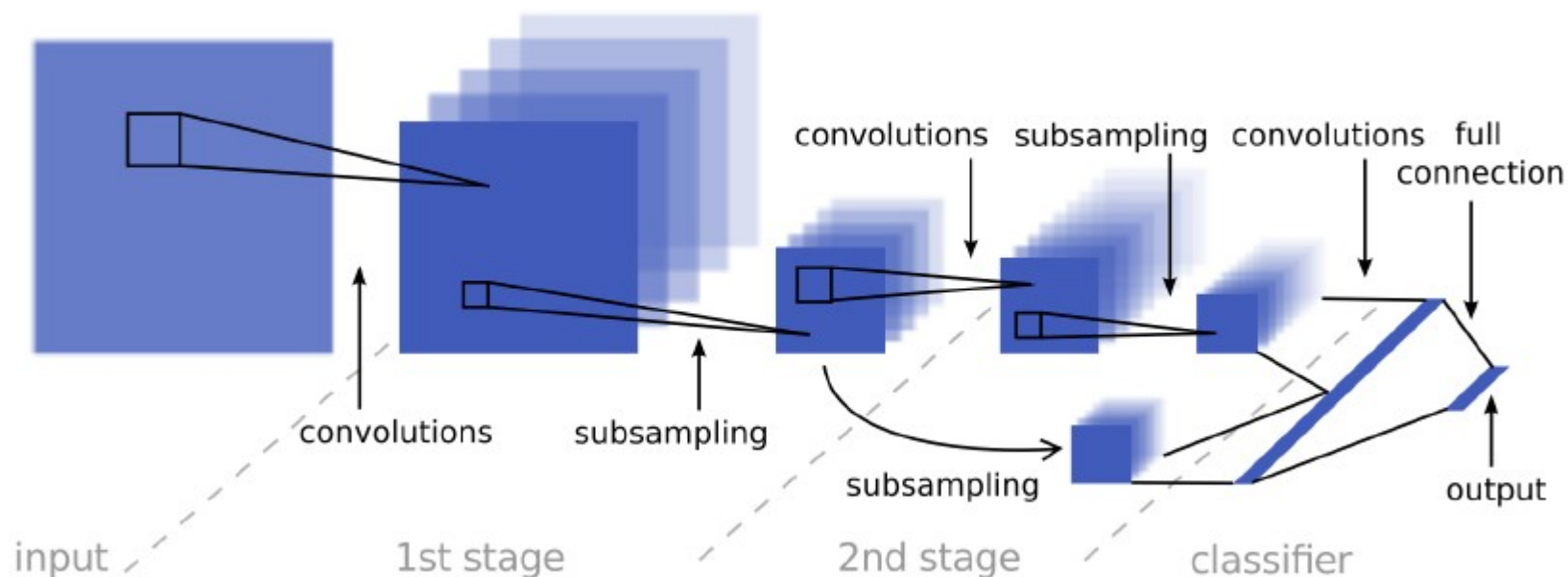
- ▶ Street View House Numbers
- ▶ 94.8% accuracy



ConvNet Architecture with Multi-Stage Features

- Feature maps from all stages are pooled/subsampled and sent to the final classification layers

- Pooled low-level features: good for textures and local motifs
- High-level features: good for “gestalt” and global shape



Task	Single-Stage features	Multi-Stage features	Improvement %
Pedestrians detection (INRIA) [9]	14.26%	9.85%	31%
Traffic Signs classification (GTSRB) [11]	1.80%	0.83%	54%
House Numbers classification (SVHN)	5.72%	5.67%	0.9%

[Sermanet, Chintala, LeCun ArXiv:1204.3968, 2012]

Industrial Applications of ConvNets

• AT&T/Lucent/NCR

- ▶ Check reading, OCR, handwriting recognition (deployed 1996)

• NEC

- ▶ Intelligent vending machines and advertizing posters, cancer cell detection, automotive applications

• Google

- ▶ Face and license plate removal from StreetView images

• Microsoft

- ▶ Handwriting recognition, speech detection

• Orange

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

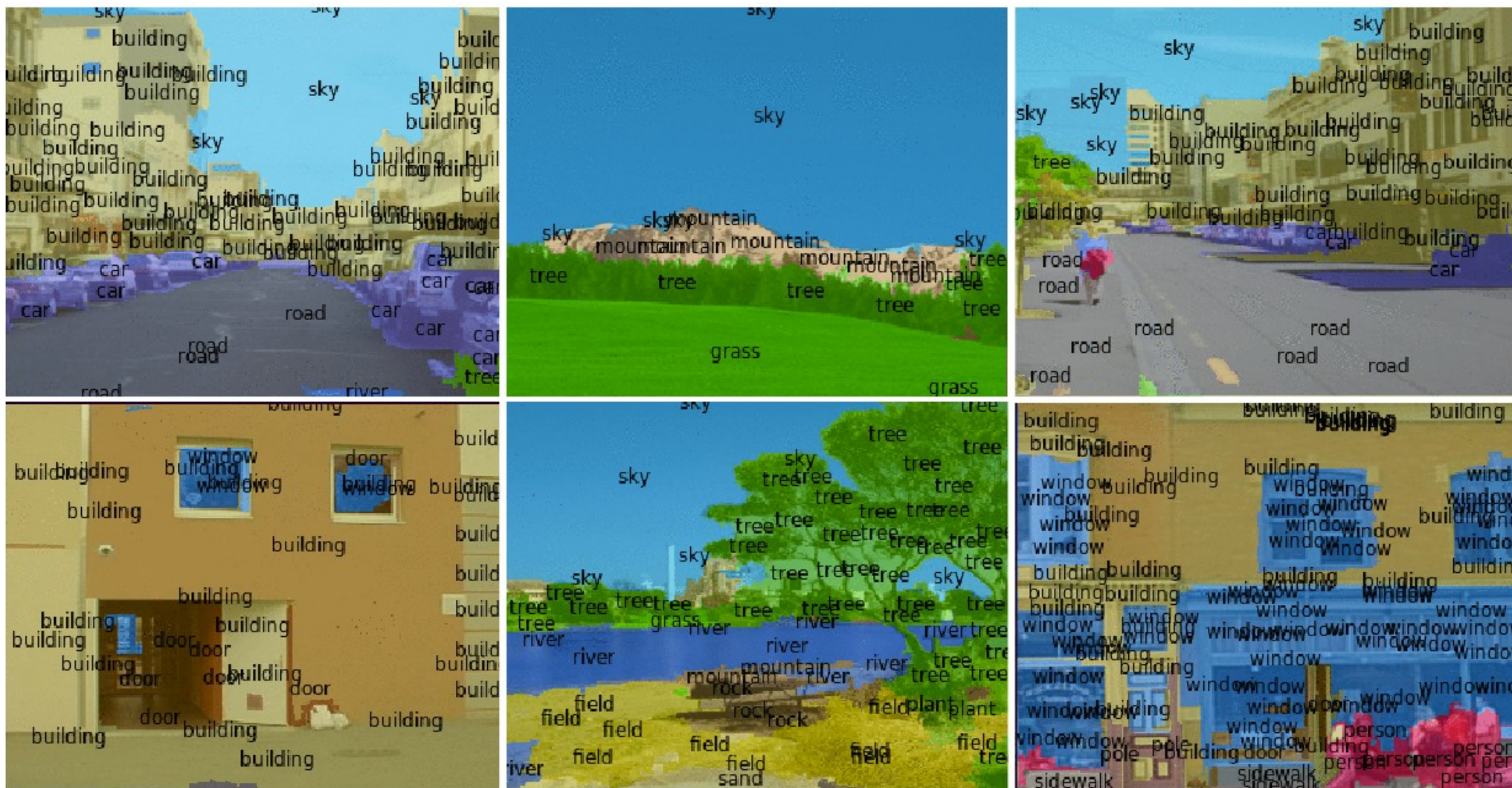
• Startups, other companies...

Fast Scene Parsing

[Farabet, Couprie, Najman, LeCun, ICML 2012]

Labeling every pixel with the object it belongs to

- Would help identify obstacles, targets, landing sites, dangerous areas
- Would help line up depth map with edge maps

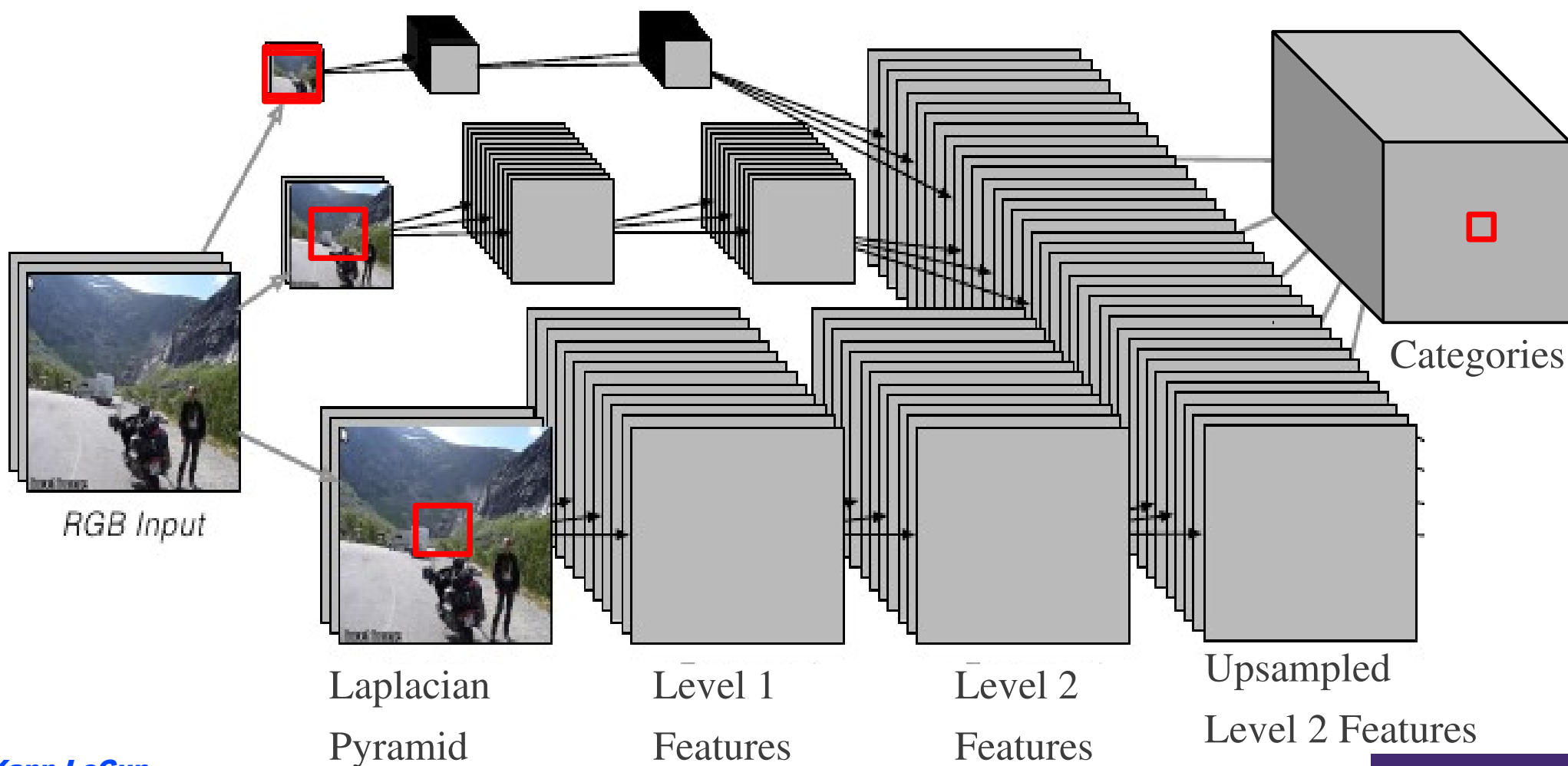


[Farabet et al. ICML 2012]

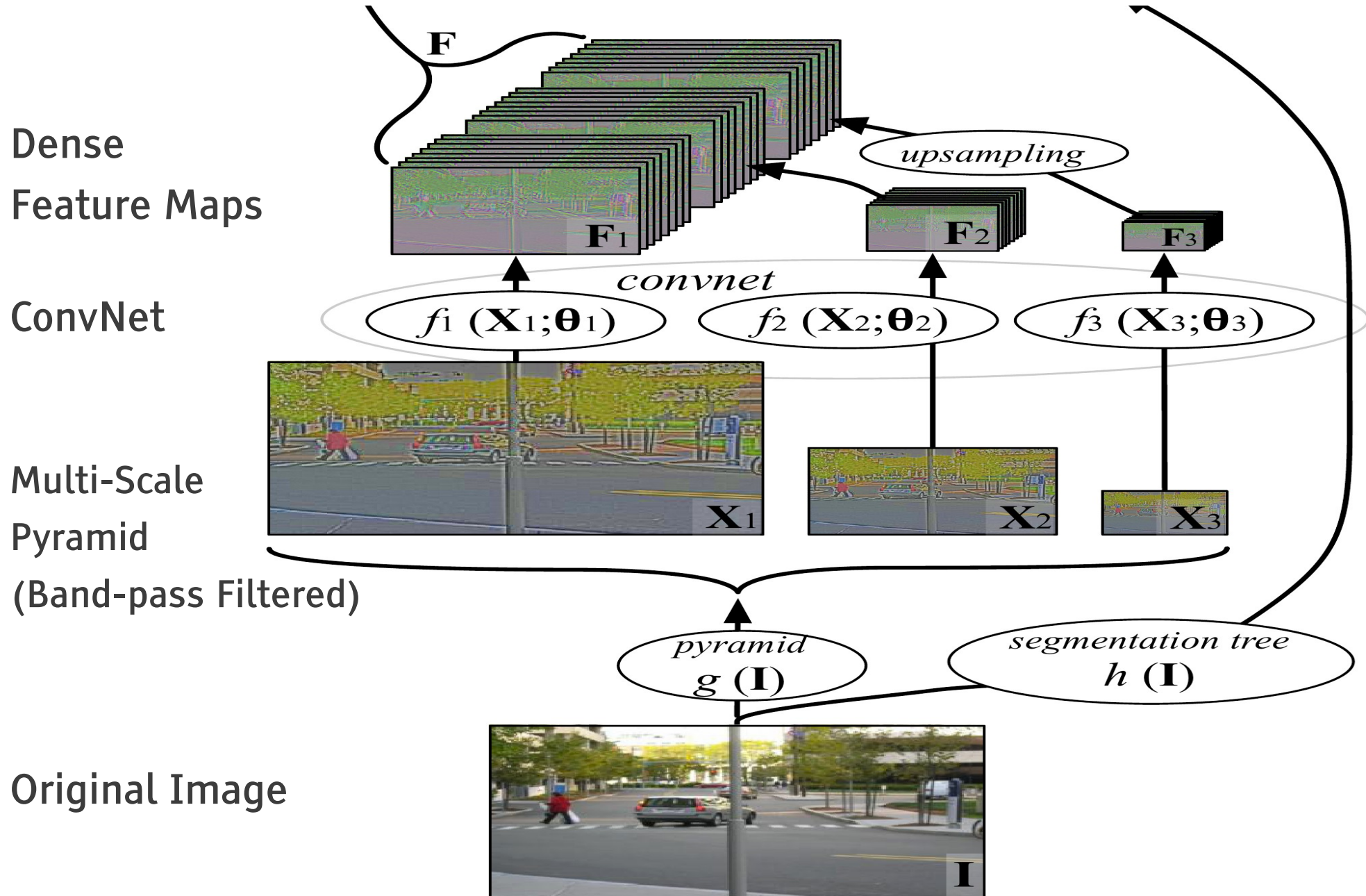
Scene Parsing/Labeling: ConvNet Architecture

Each output sees a large input context:

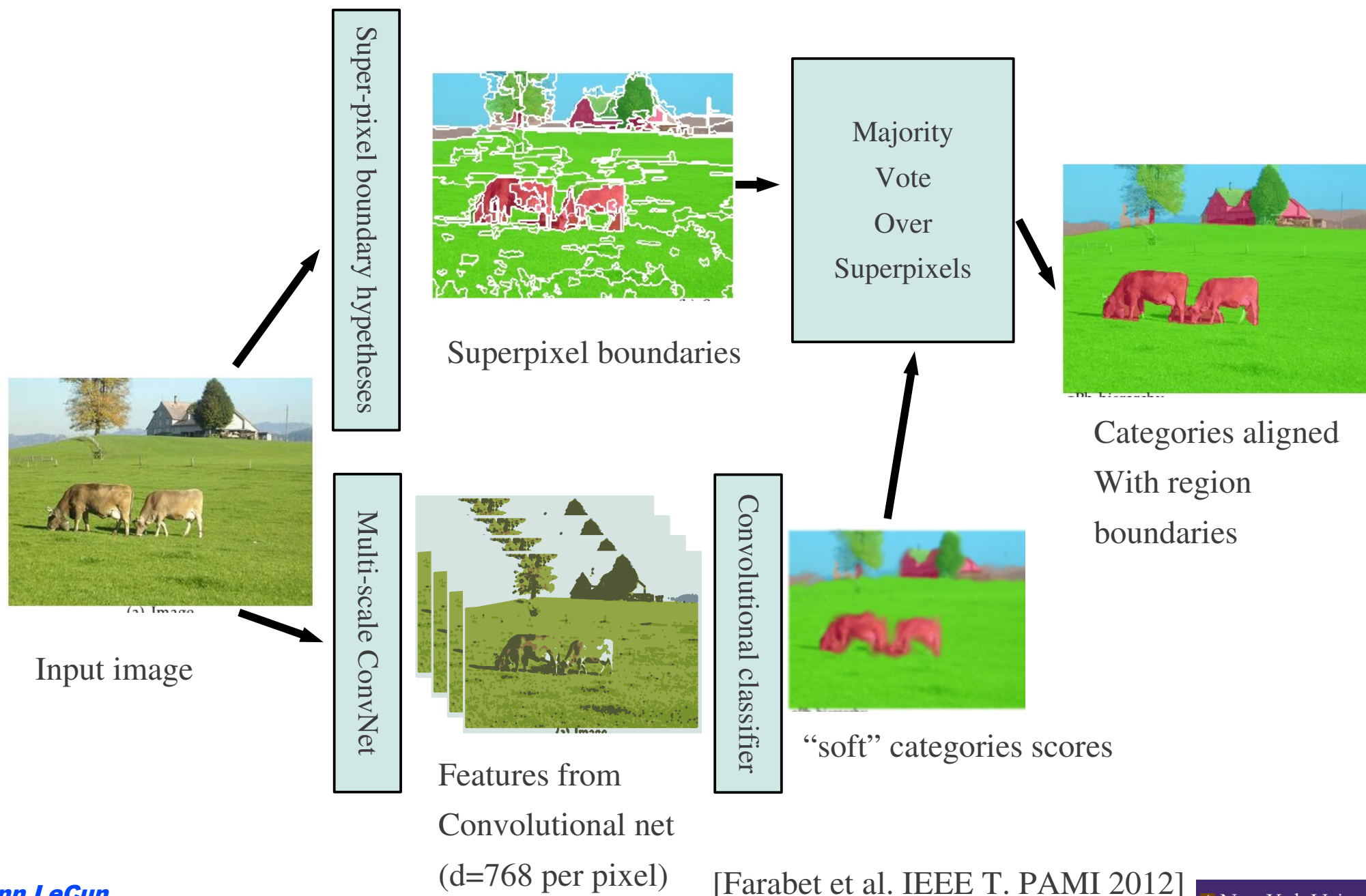
- ▶ **46x46** window at full rez; **92x92** at $\frac{1}{2}$ rez; **184x184** at $\frac{1}{4}$ rez
- ▶ [7x7conv]->[2x2pool]->[7x7conv]->[2x2pool]->[7x7conv]->
- ▶ Trained supervised on fully-labeled images



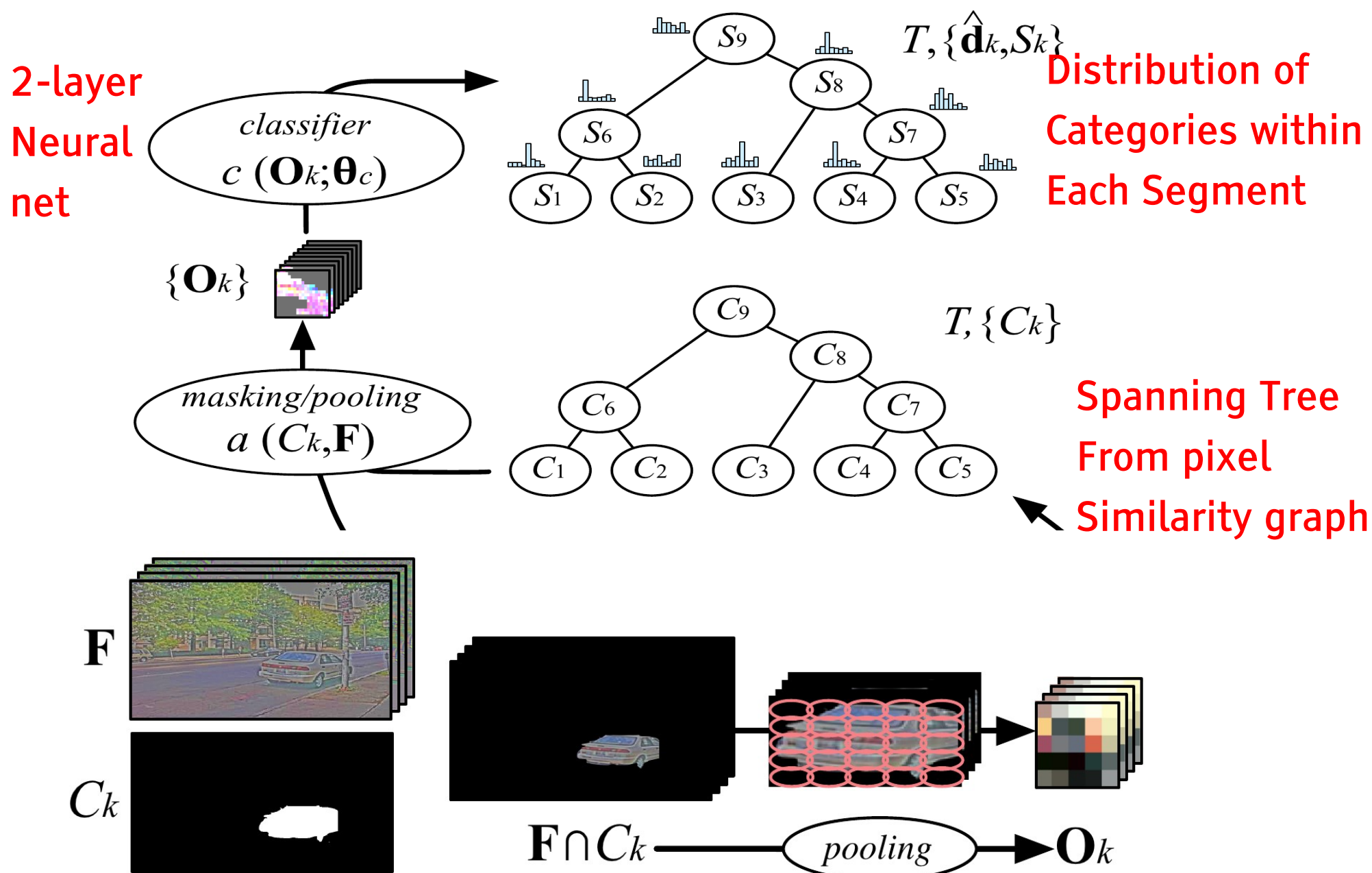
Scene Parsing/Labeling: System Architecture



Method 1: majority over super-pixel regions



Method 2: optimal cover of purity tree



Scene Parsing/Labeling: Performance

Stanford Background Dataset [Gould 1009]: 8 categories

	Pixel Acc.	Class Acc.	CT (sec.)
Gould <i>et al.</i> 2009 [14]	76.4%	-	10 to 600s
Munoz <i>et al.</i> 2010 [32]	76.9%	66.2%	12s
Tighe <i>et al.</i> 2010 [46]	77.5%	-	10 to 300s
Socher <i>et al.</i> 2011 [45]	78.1%	-	?
Kumar <i>et al.</i> 2010 [22]	79.4%	-	< 600s
Lempitzky <i>et al.</i> 2011 [28]	81.9%	72.4%	> 60s
singlescale convnet	66.0 %	56.5 %	0.35s
multiscale convnet	78.8 %	72.4%	0.6s
multiscale net + superpixels	80.4%	74.56%	0.7s
multiscale net + gPb + cover	80.4%	75.24%	61s
multiscale net + CRF on gPb	81.4%	76.0%	60.5s

SIFT Flow dataset [Liu 2009]: 33 categories

	Pixel Acc.	Class Acc.
Liu <i>et al.</i> 2009 [29]	74.75%	-
Tighe <i>et al.</i> 2010 [46]	76.9%	29.4%
multiscale net + cover ¹	78.5%	29.6%
multiscale net + cover ²	74.2%	46.0%

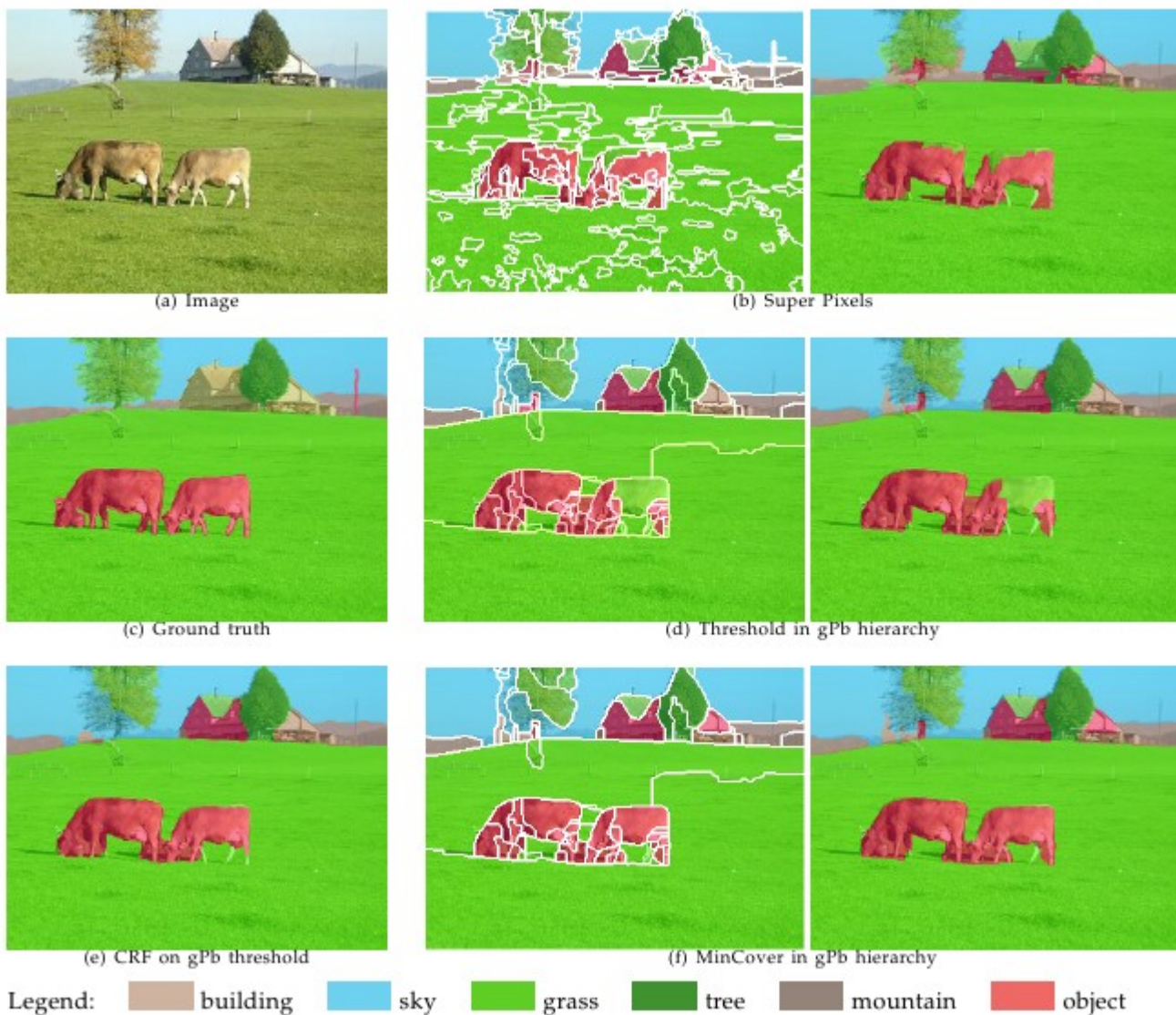
Barcelona dataset [Tighe 2010]: 170 categories.

	Pixel Acc.	Class Acc.
Tighe <i>et al.</i> 2010 [46]	66.9%	7.6%
multiscale net + cover ¹	67.8%	%
multiscale net + cover ²	39.1%	10.7%

[Farabet et al. IEEE T. PAMI 2012]

Scene Parsing/Labeling: Results

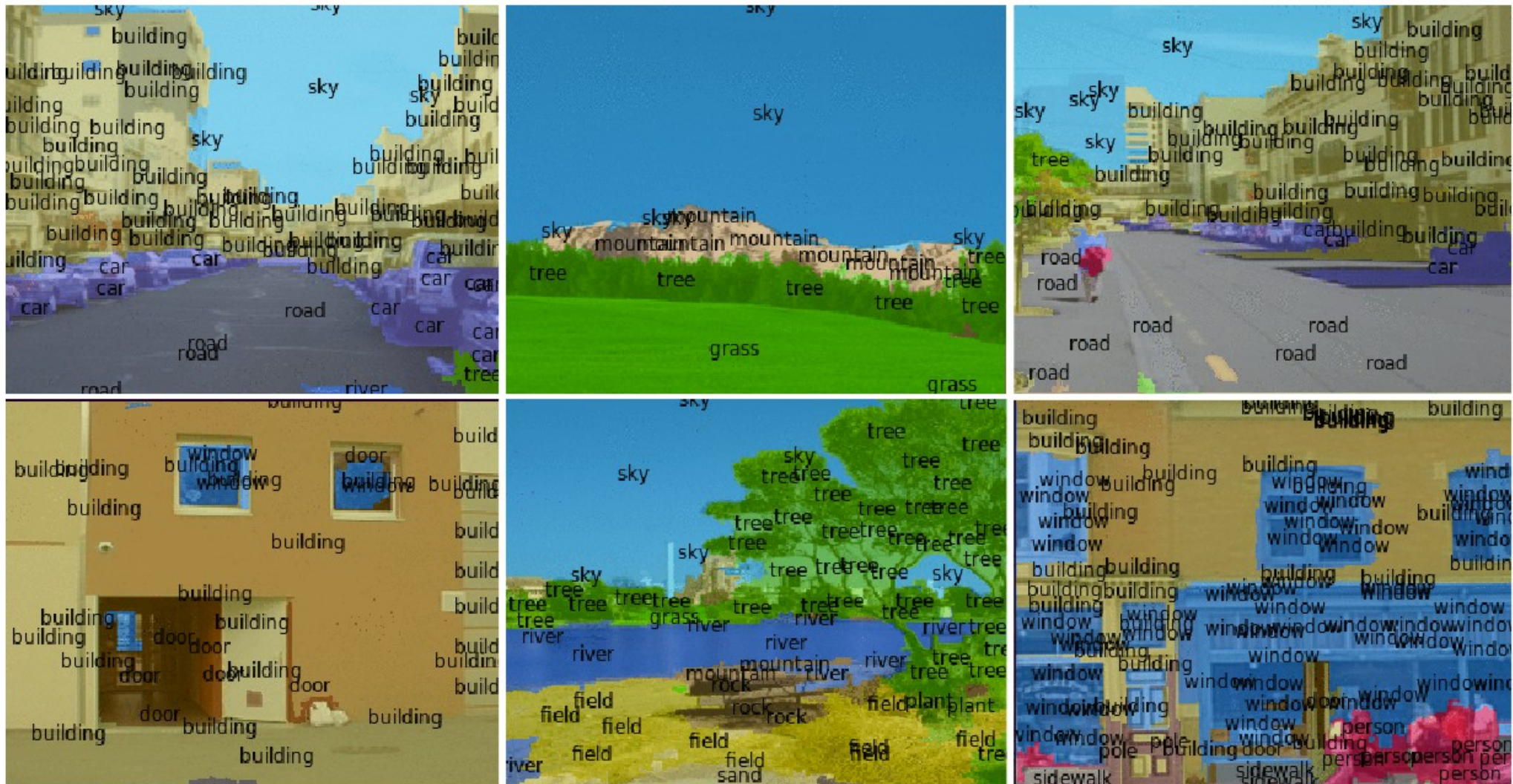
● Samples from the SIFT-Flow dataset (Liu)



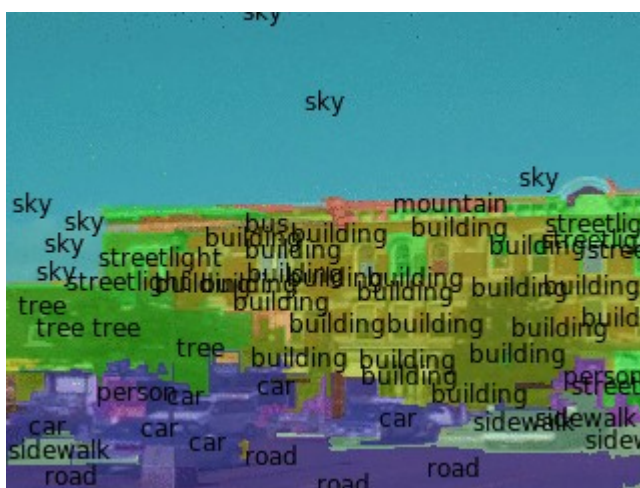
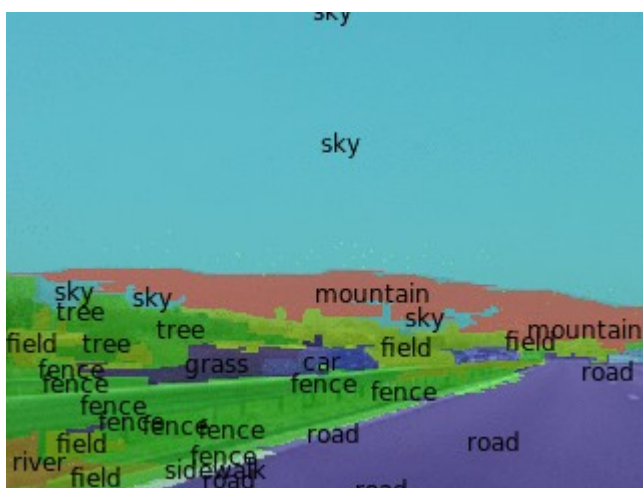
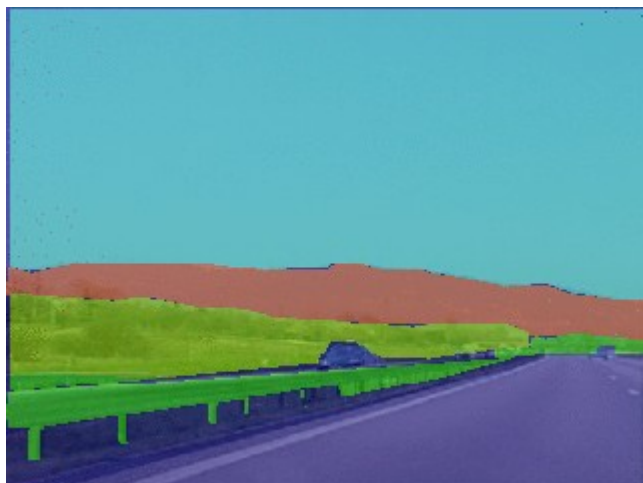
[Farabet et al. 2012]

Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

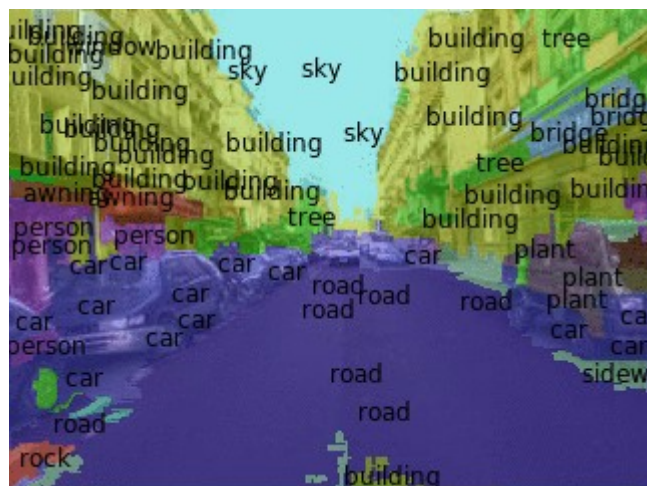
● Samples from the SIFT-Flow dataset (Liu)



Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

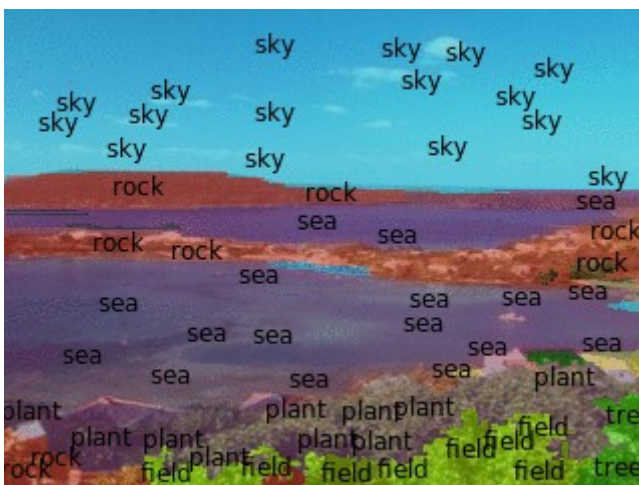
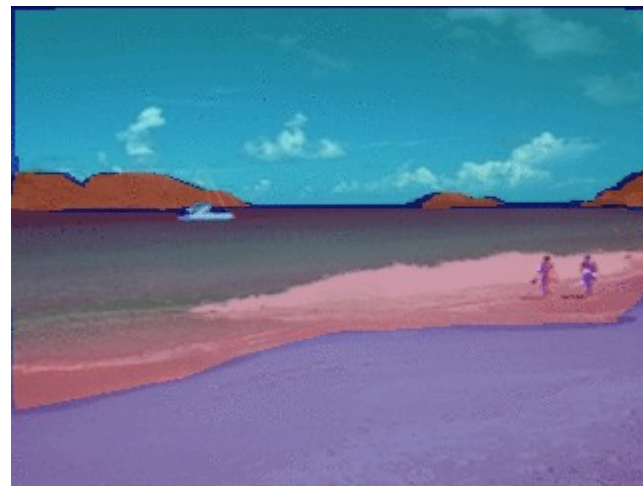
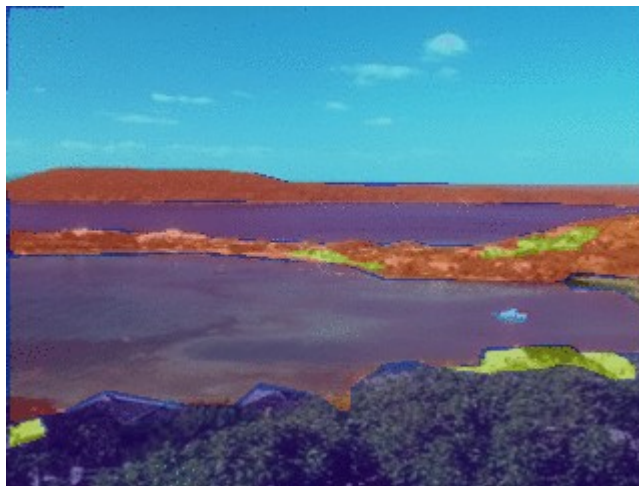


Scene Parsing/Labeling



[Farabet et al. ICML 2012]

Scene Parsing/Labeling



[Farabet et al. ICML 2012]

Scene Parsing/Labeling



[Farabet et al. 2012]

Scene Parsing/Labeling



[Farabet et al. 2012]

Scene Parsing/Labeling



- No post-processing
- Frame-by-frame
- ConvNet runs at 50ms/frame on Virtex-6 FPGA hardware
 - ▶ But communicating the features over ethernet limits system perf.

Scene Parsing/Labeling: Temporal Consistency



- Majority Vote on Spatio-Temporal Super-Pixels
- Reset every second

Unsupervised Feature Learning: variations on the sparse auto-encoder theme

Learning Features with Unsupervised Pre-Training

- Supervised learning requires lots of labeled samples
- Most available data is unlabeled
- **Models need to be large** to “understand” the task
- But large models have many parameters and require many labeled samples
- **Unsupervised learning** can be used to **pre-train** the system before supervised refinement
- Unsupervised pre-training “consumes” degrees of freedom while placing the system in a favorable region of parameter space.
- Supervised refinement merely find the closest local minimum within the attractor found by unsupervised pre-training.
- **Unsupervised feature learning through sparse/overcomplete auto-encoders**
- **With high-dimensional and sparse representations, the data manifold is “flattened”** (any collection of points is flatter in higher dimension)

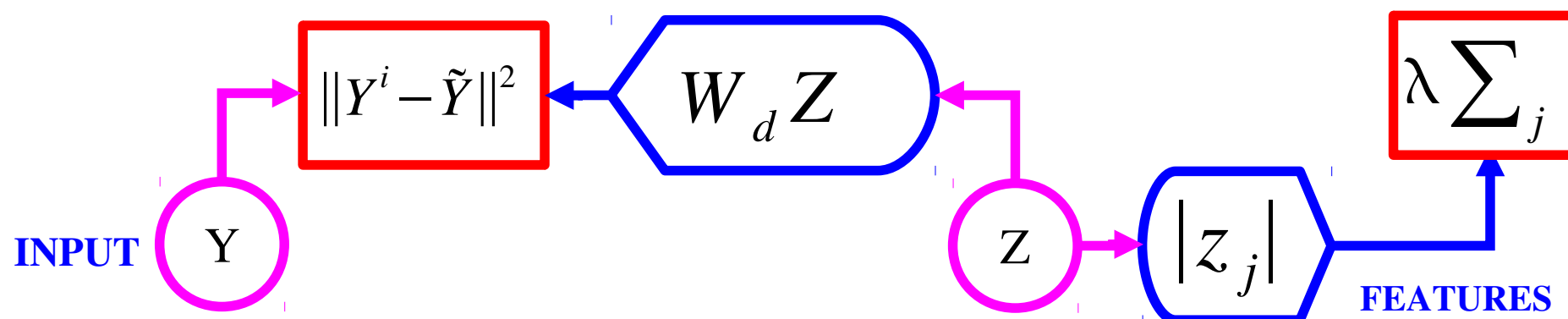
Sparse Coding & Sparse Modeling

[Olshausen & Field 1997]

- Sparse linear reconstruction

- Energy = reconstruction_error + code_prediction_error + code_sparsity

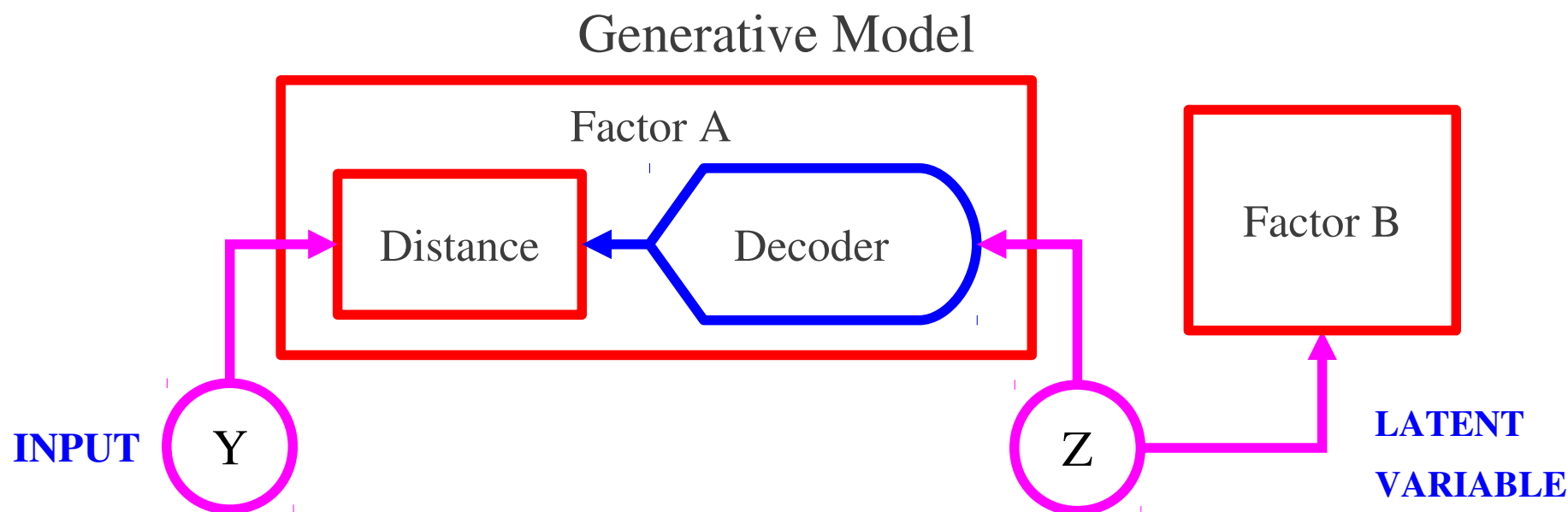
$$E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \lambda \sum_j |z_j|$$



- Inference is slow $Y \rightarrow \hat{Z} = \operatorname{argmin}_Z E(Y, Z)$

How to Speed Up Inference in a Generative Model

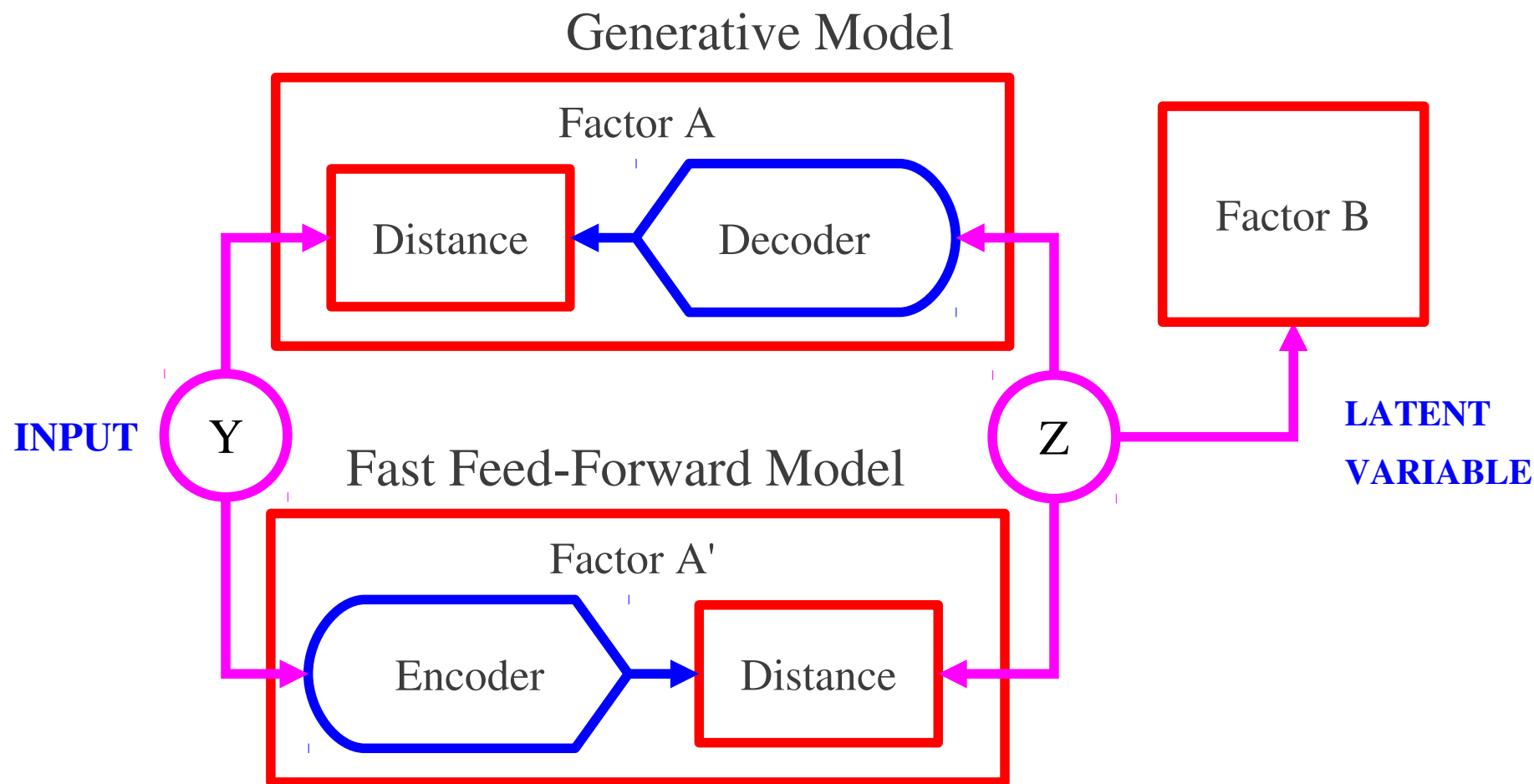
- Factor Graph with an asymmetric factor
- Inference $Z \rightarrow Y$ is easy
 - Run Z through deterministic decoder, and sample Y
- Inference $Y \rightarrow Z$ is hard, particularly if Decoder function is many-to-one
 - MAP: minimize sum of two factors with respect to Z
 - $Z^* = \operatorname{argmin}_z \text{Distance}[\text{Decoder}(Z), Y] + \text{FactorB}(Z)$



Idea: Train a “simple” function to approximate the solution

[Kavukcuoglu, Ranzato, LeCun, rejected by every conference, 2008-2009]

- Train a “simple” feed-forward function to predict the result of a complex optimization on the data points of interest



- 1. Find optimal Z_i for all Y_i ; 2. Train Encoder to predict Z_i from Y_i

Predictive Sparse Decomposition (PSD): sparse auto-encoder

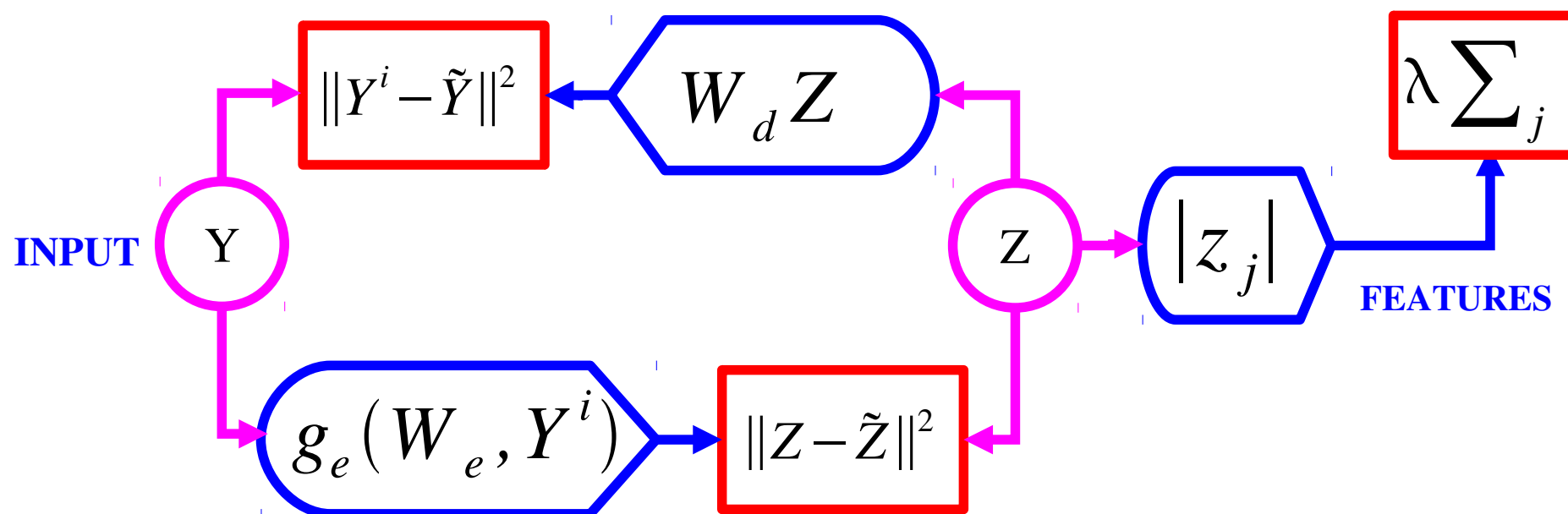
[Kavukcuoglu, Ranzato, LeCun, 2008 → arXiv:1010.3467],

• Prediction the optimal code with a **trained encoder**

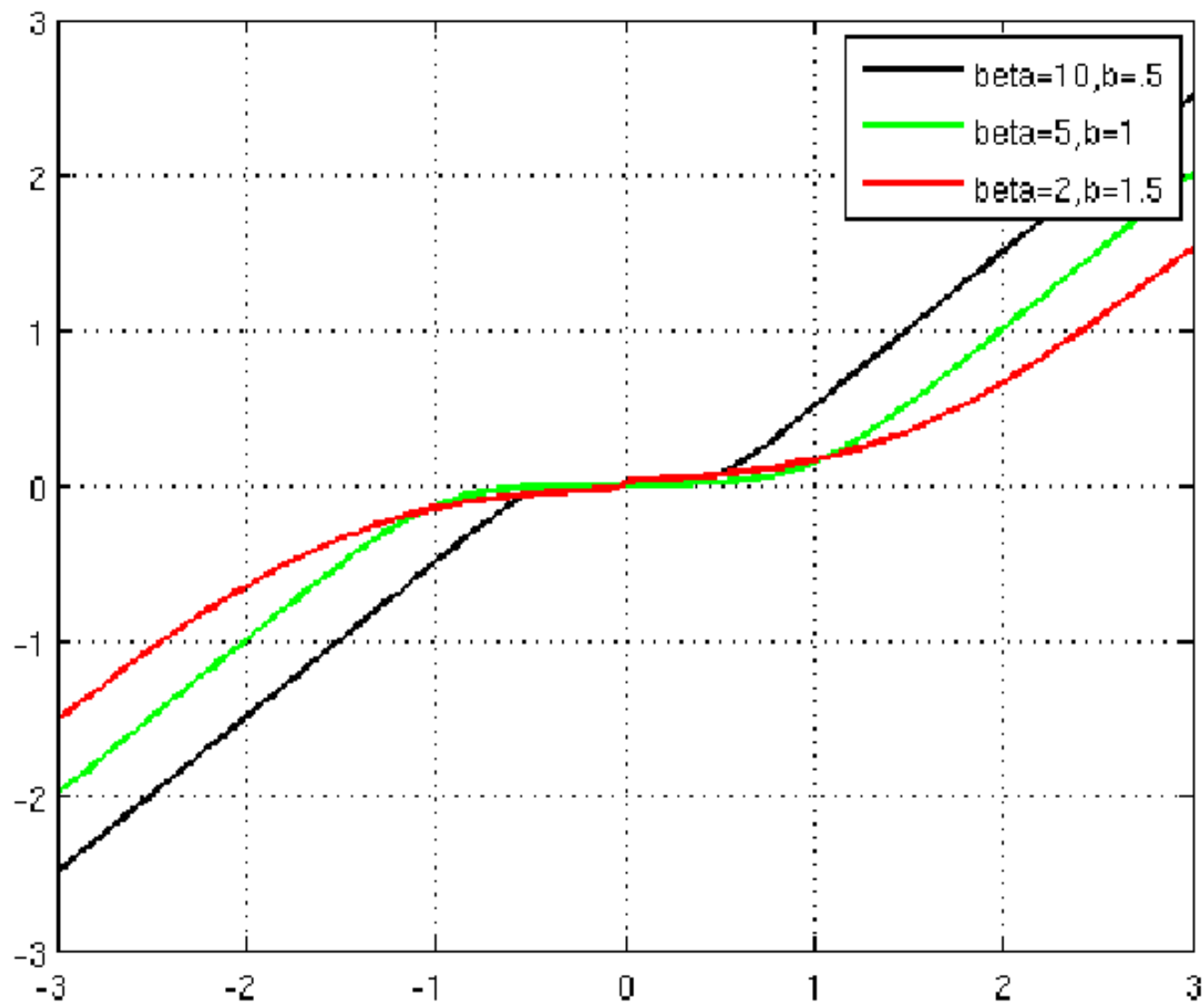
• Energy = reconstruction_error + code_prediction_error + code_sparsity

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

$$g_e(W_e, Y^i) = \text{shrinkage}(W_e Y^i)$$

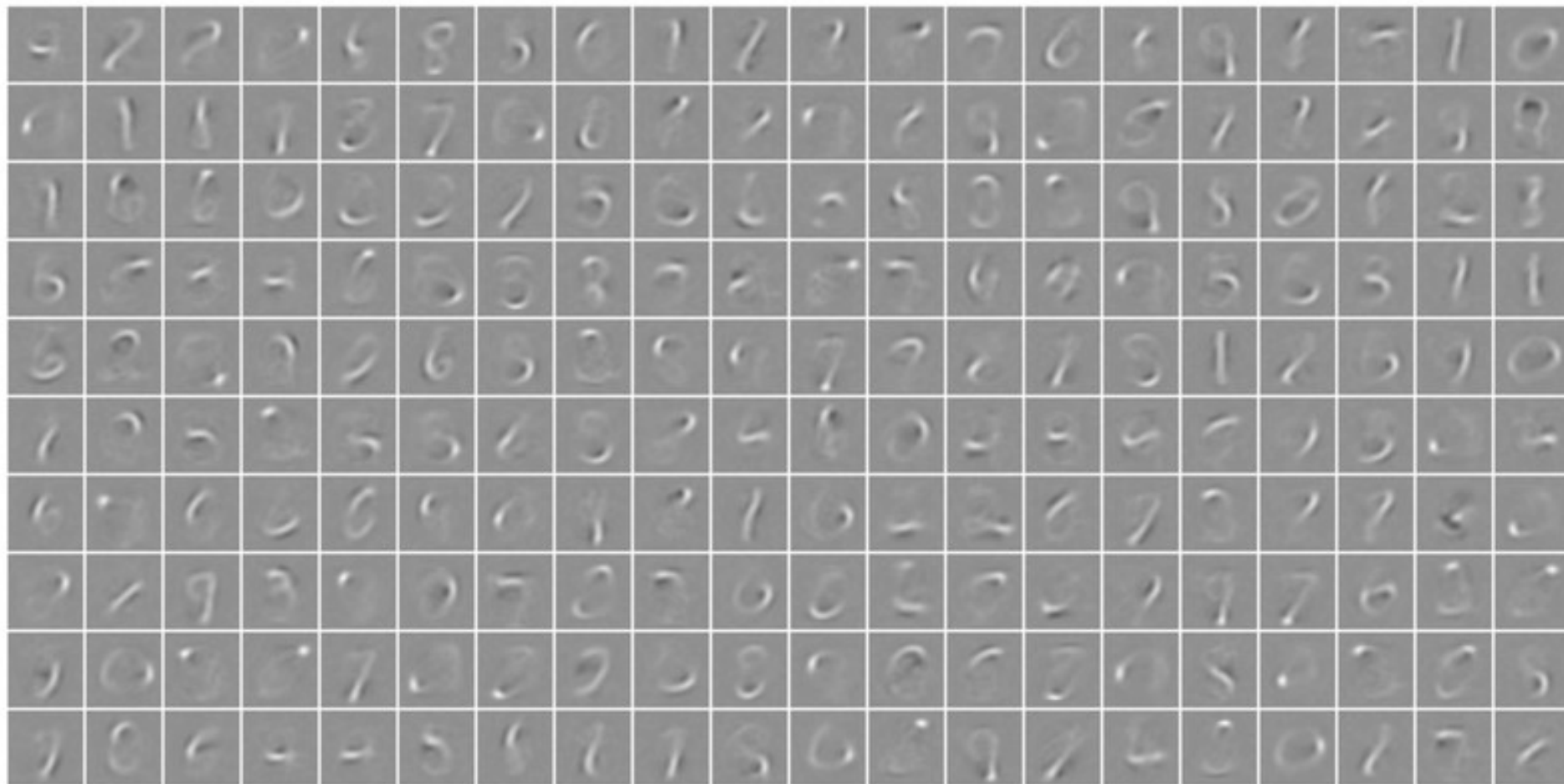


Soft Shrinkage Non-Linearity



PSD: Basis Functions on MNIST

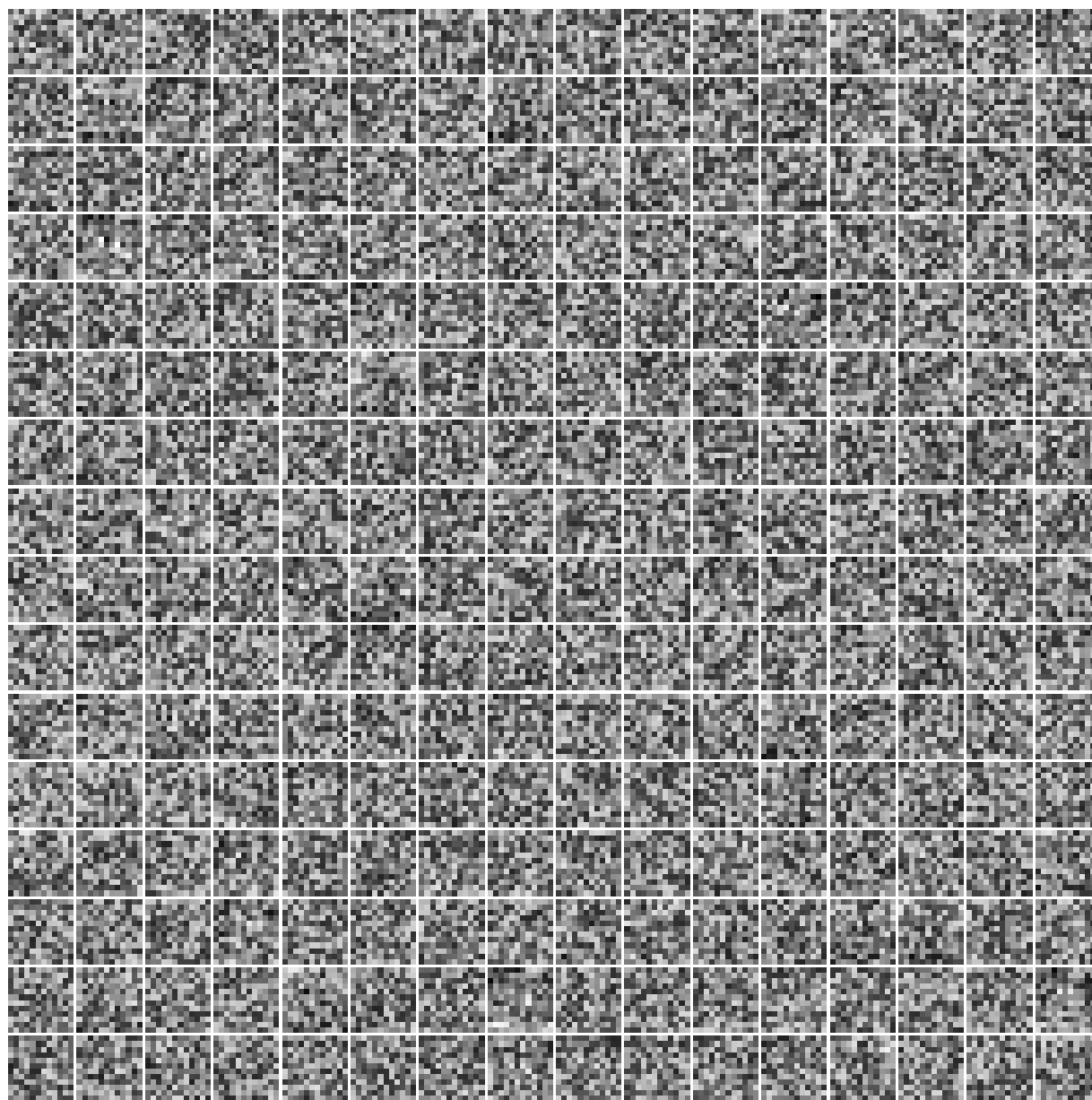
- Basis functions (and encoder matrix) are digit parts



Predictive Sparse Decomposition (PSD): Training

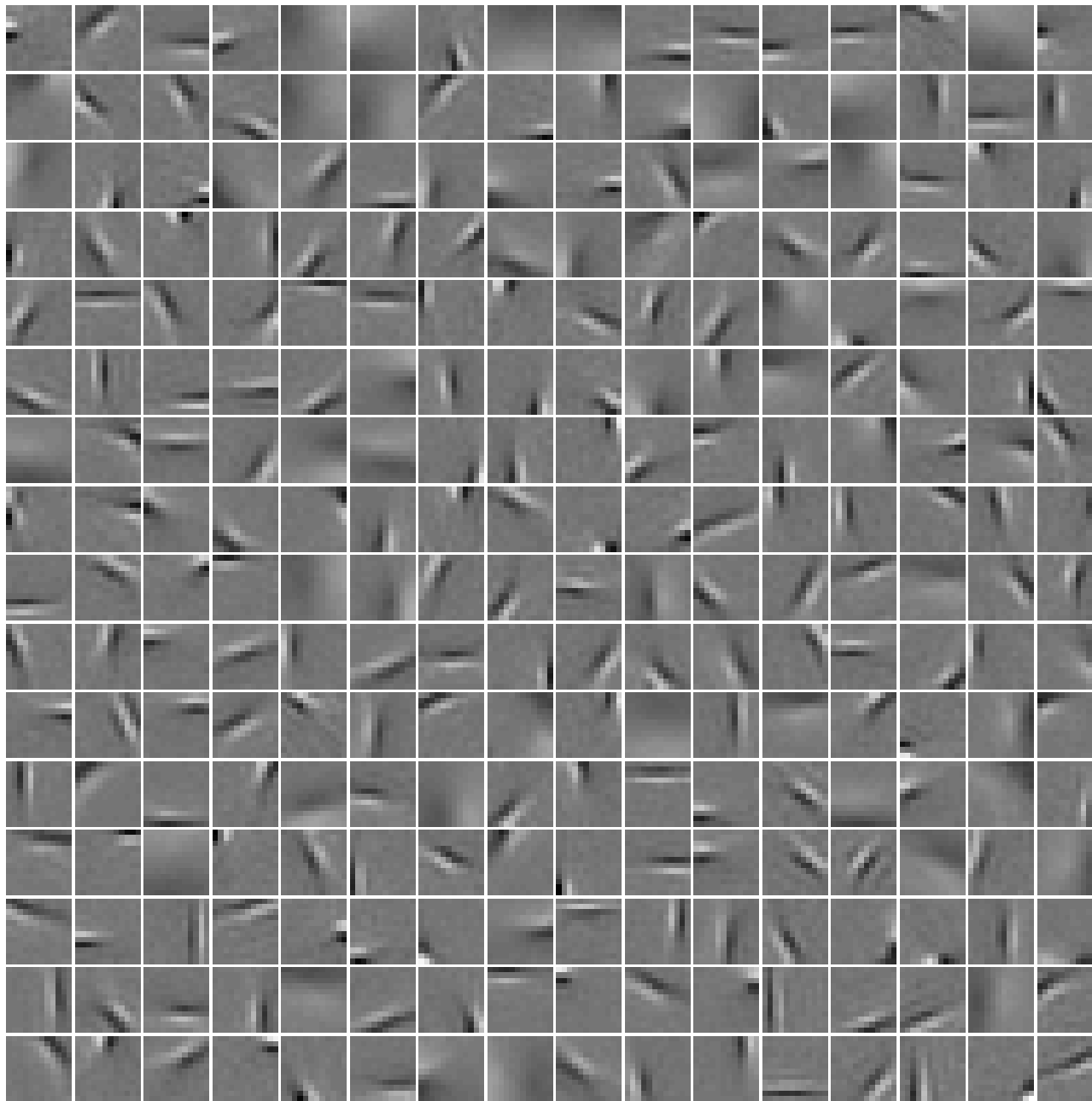
- Training on natural images patches.

- ▶ 12X12
- ▶ 256 basis functions



iteration no 0

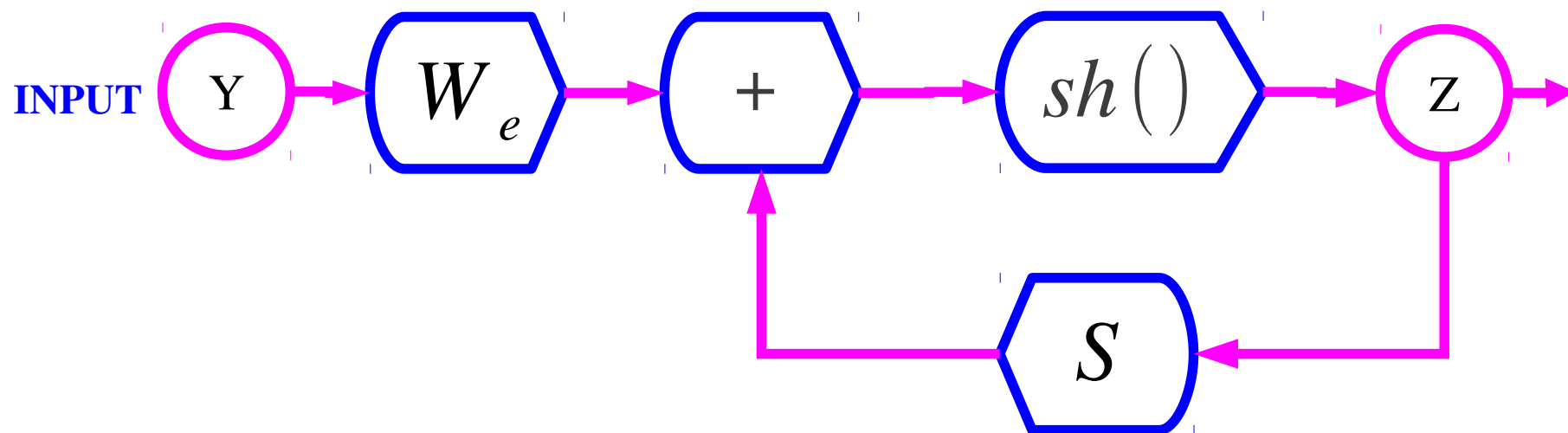
Learned Features on natural patches: V1-like receptive fields



Better Idea: Give the “right” structure to the encoder

[Gregor & LeCun, ICML 2010], [Bronstein et al. ICML 2012]

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

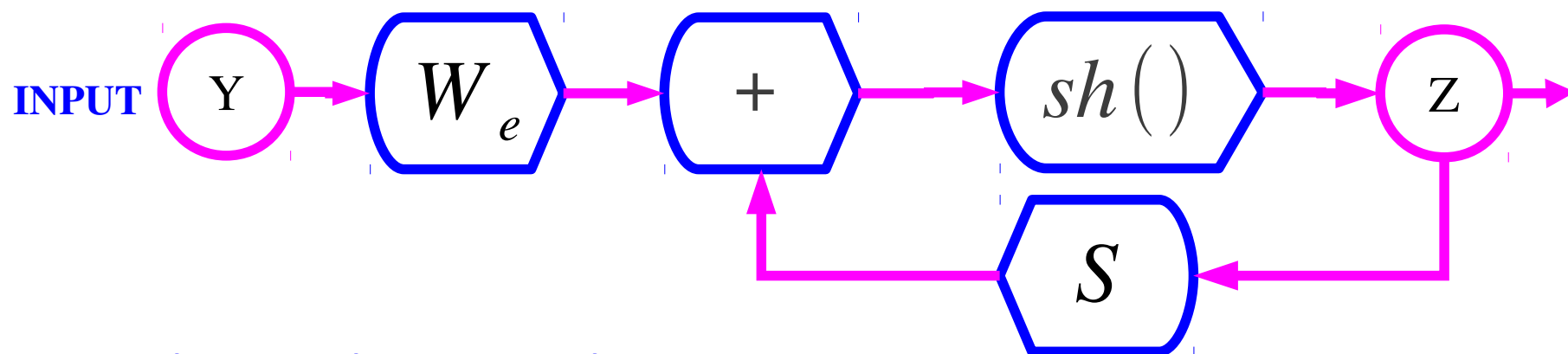


$$Z(t+1) = \text{Shrinkage}_{\lambda/L} \left[Z(t) - \frac{1}{L} W_d^T (W_d Z(t) - Y) \right]$$

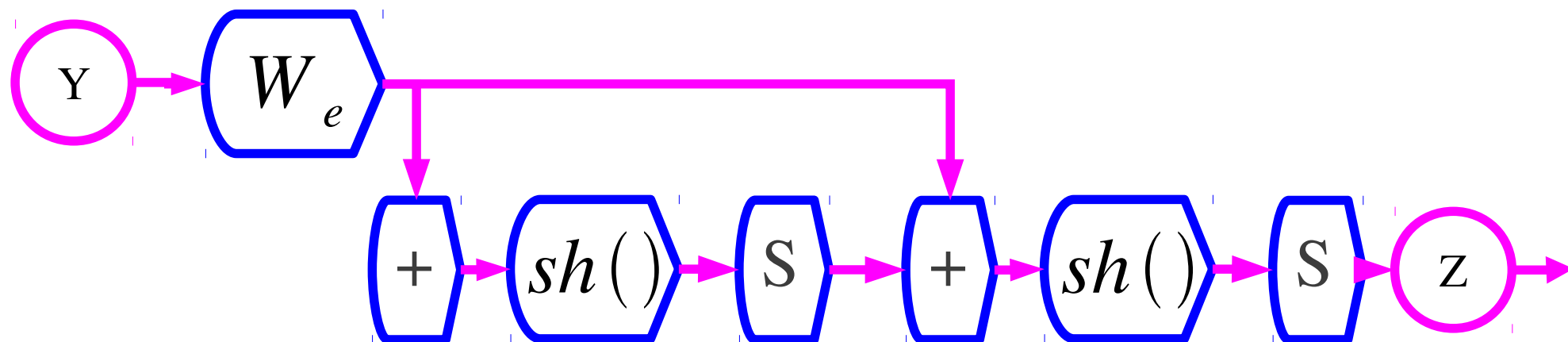
$$Z(t+1) = \text{Shrinkage}_{\lambda/L} [W_e^T Y + S Z(t)]; \quad W_e = \frac{1}{L} W_d; \quad S = I - \frac{1}{L} W_d^T W_d$$

LISTA: Train W_e and S matrices to give a good approximation quickly

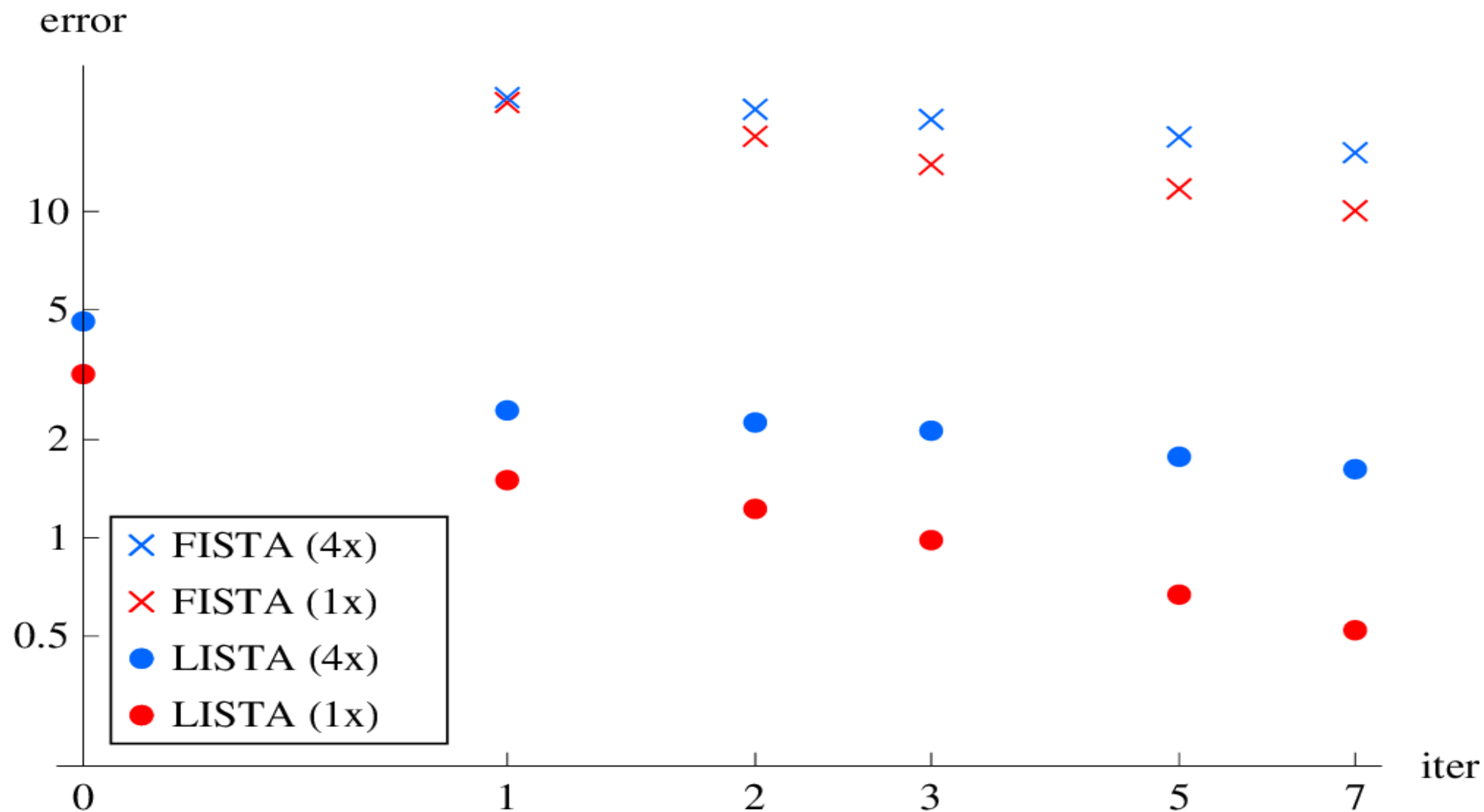
- Think of the FISTA flow graph as a recurrent neural net where W_e and S are trainable parameters



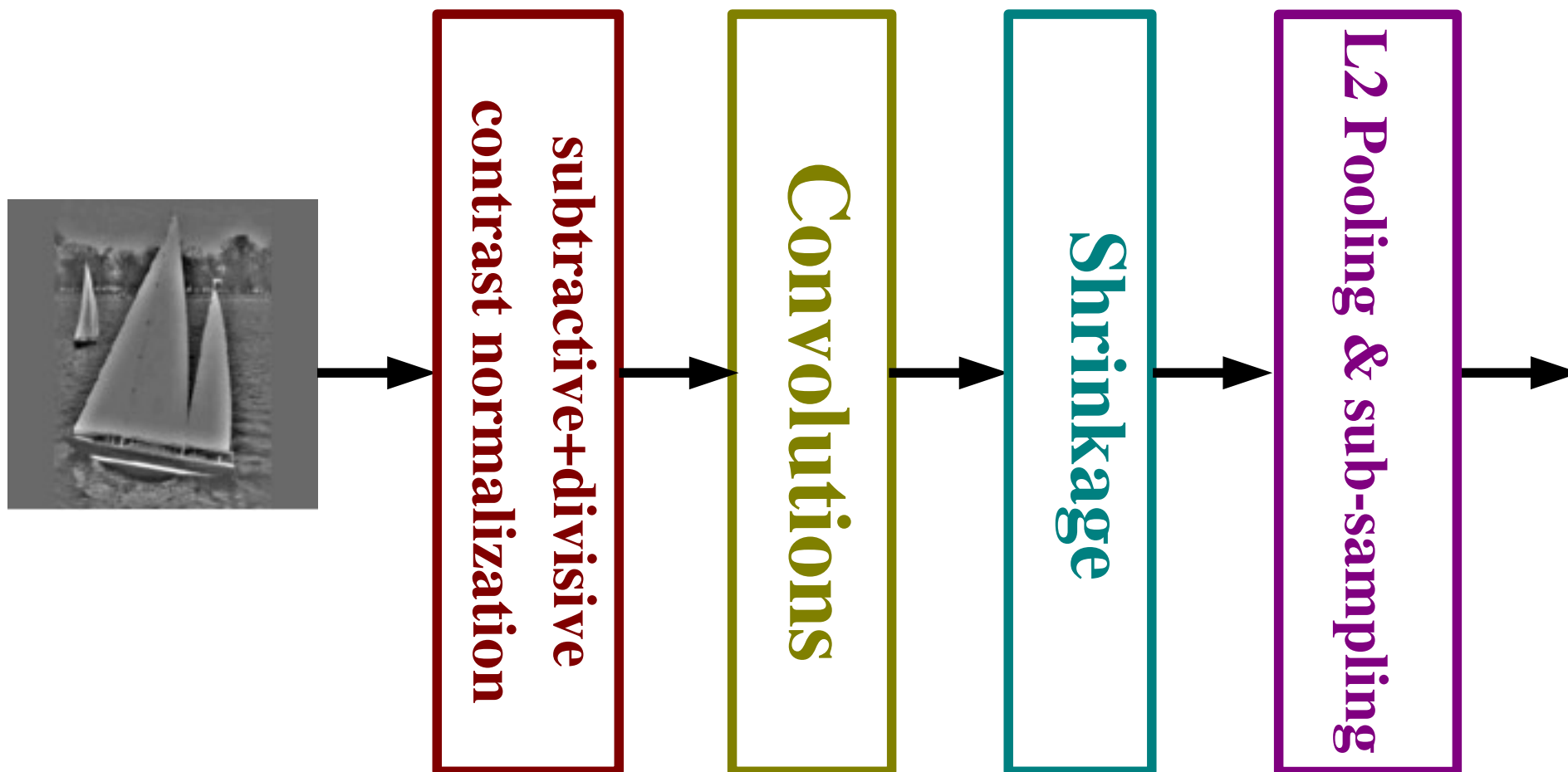
- Time-Unfold the flow graph for K iterations
- Learn the W_e and S matrices with “backprop-through-time”
- Get the best approximate solution within K iterations



Learning ISTA (LISTA) vs ISTA/FISTA



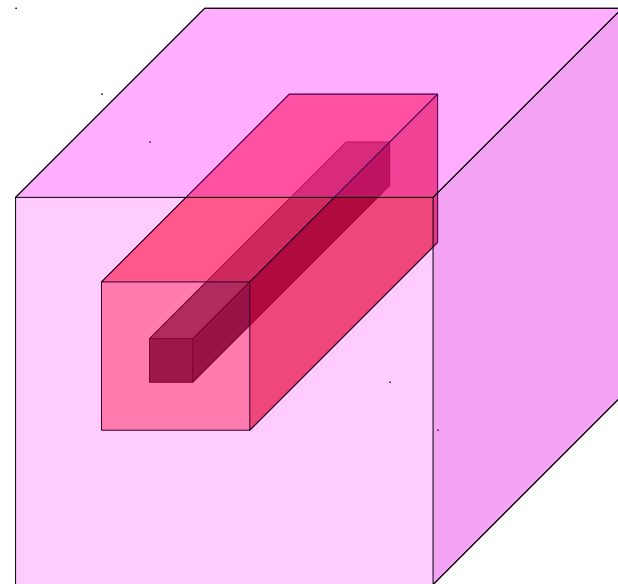
One Stage: filter → Shrinkage → L2 Pooling → Contrast Norm



THIS IS **ONE STAGE** OF THE CONVNET

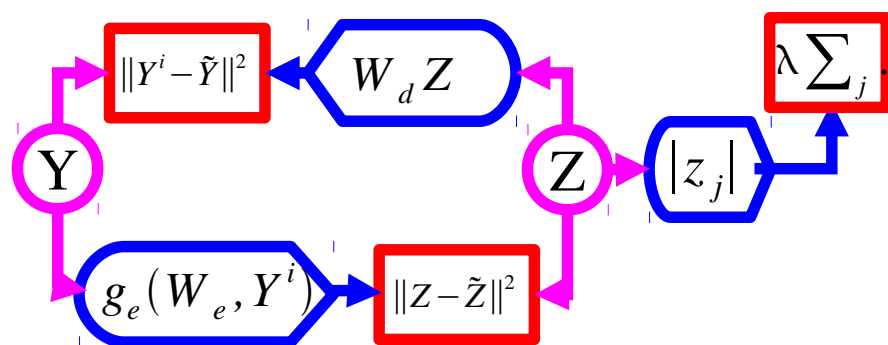
Local Contrast Normalization

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



Using PSD to Train a Hierarchy of Features

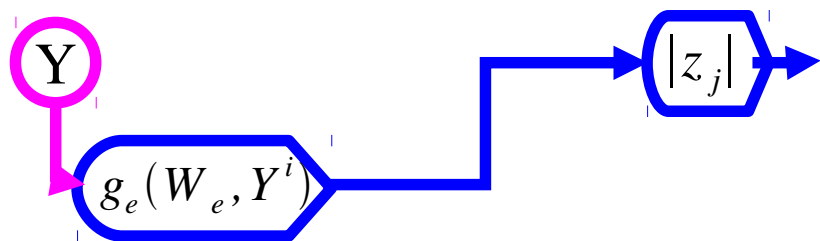
Phase 1: train first layer using PSD



FEATURES

Using PSD to Train a Hierarchy of Features

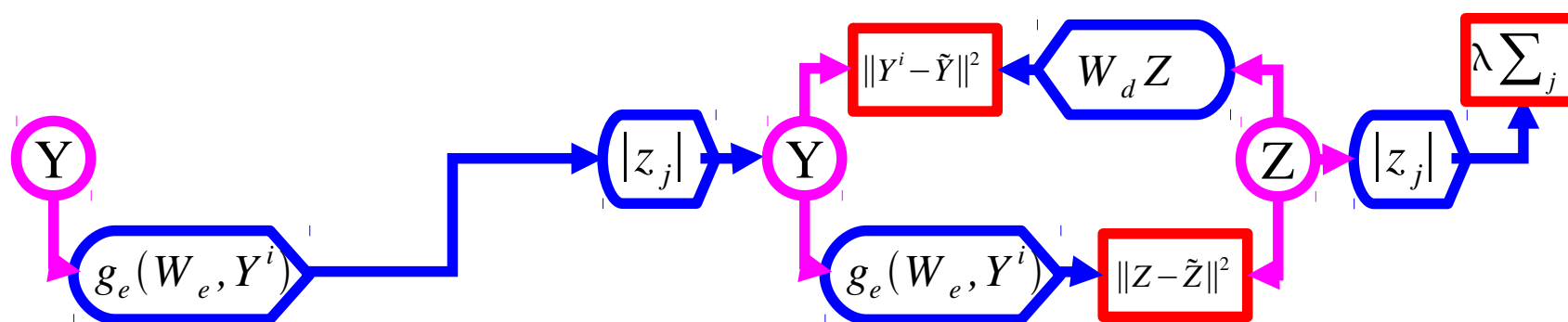
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor



FEATURES

Using PSD to Train a Hierarchy of Features

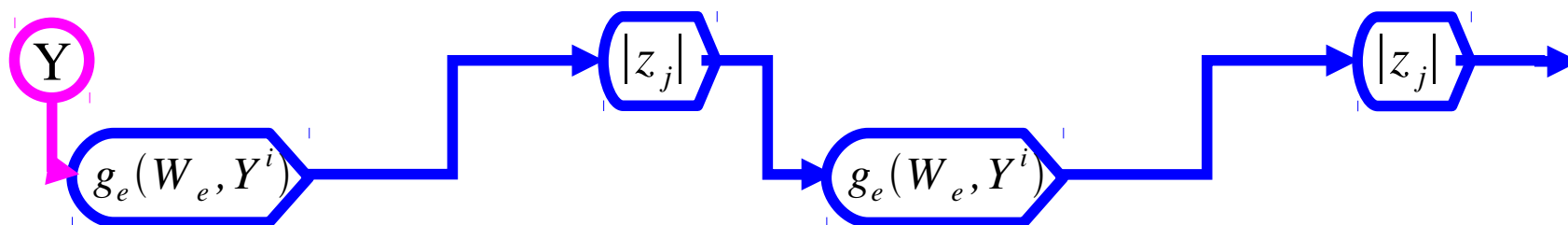
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD



FEATURES

Using PSD to Train a Hierarchy of Features

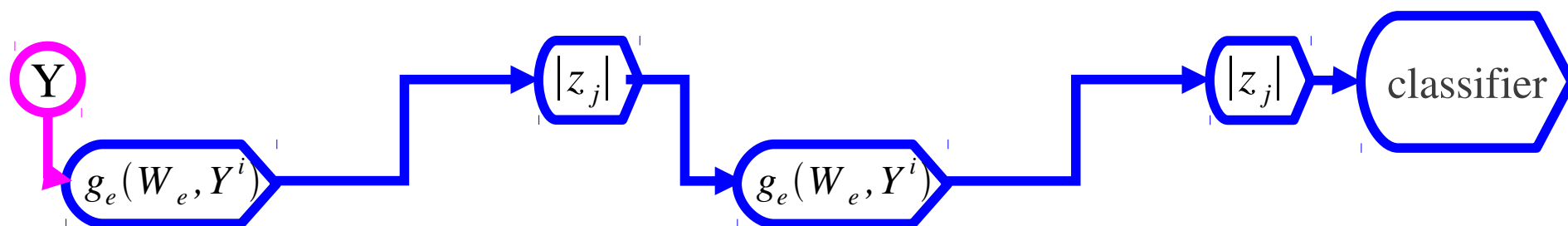
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor



FEATURES

Using PSD to Train a Hierarchy of Features

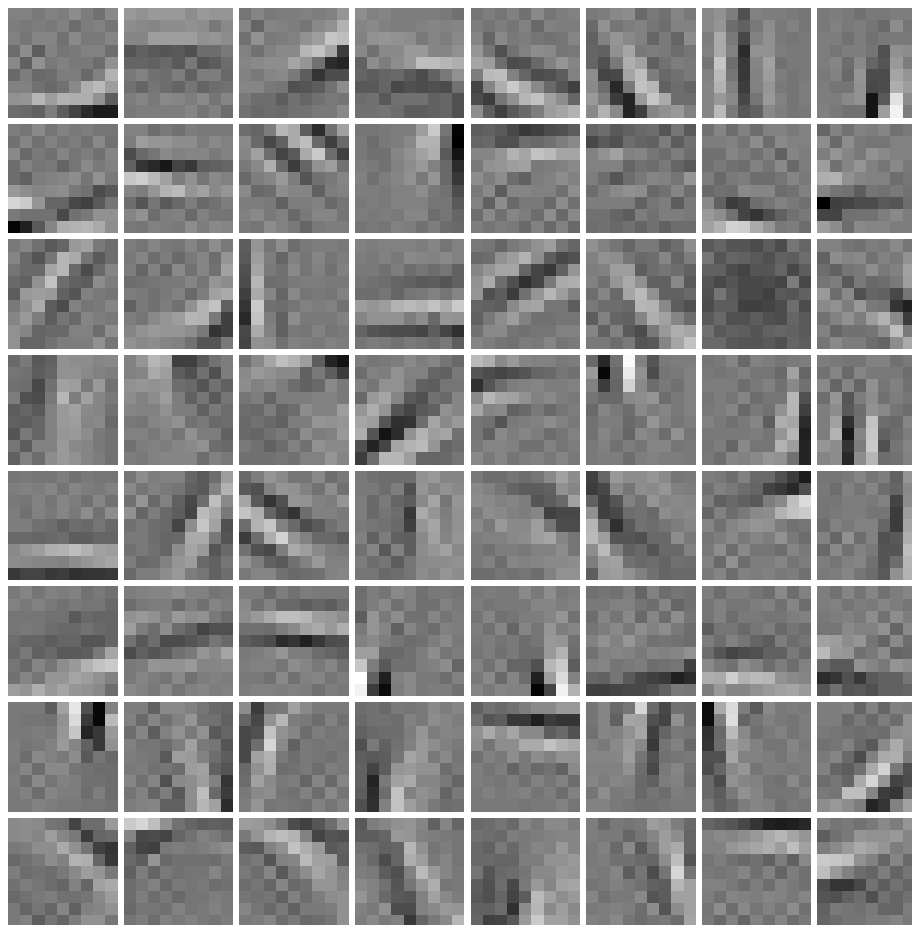
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor
- Phase 5: train a supervised classifier on top
- Phase 6 (optional): train the entire system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



FEATURES

Using PSD Features for Object Recognition

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



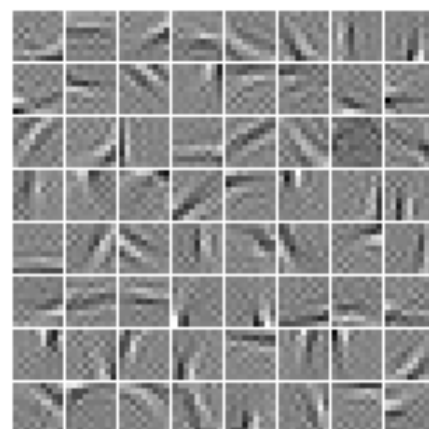
weights $\pm 0.2828 - 0.3043$

Multistage Hubel-Wiesel Architecture: Filters

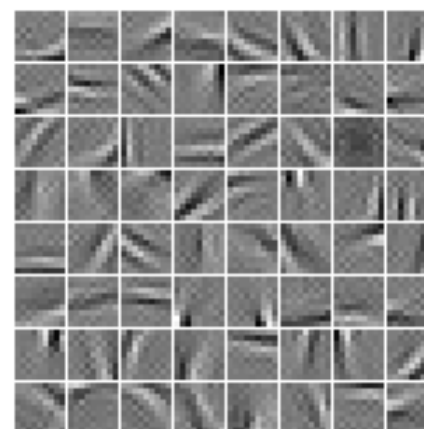
After PSD

After supervised refinement

Stage 1

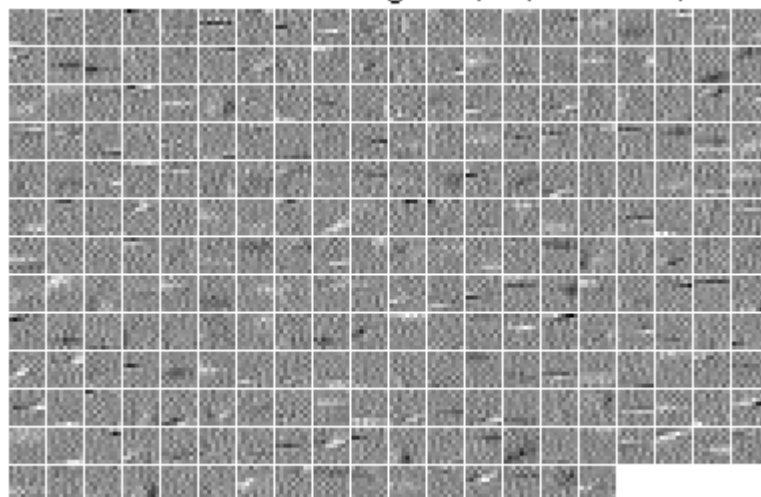


weights $\pm 0.2232 - 0.2075$

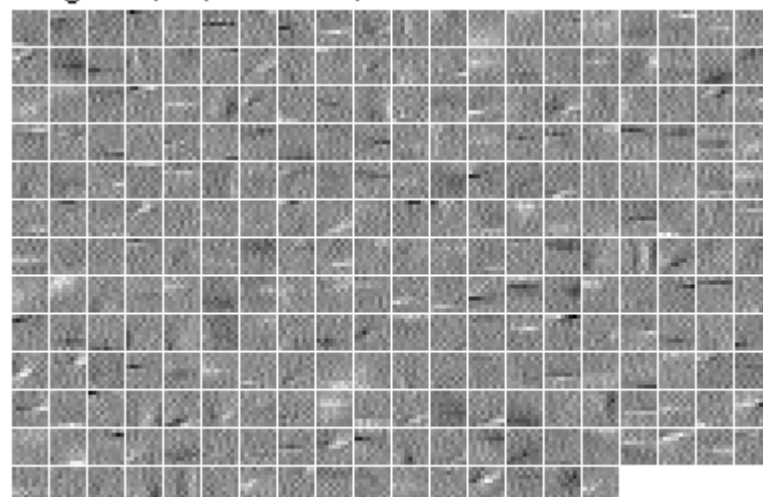


weights $\pm 0.2828 - 0.3043$

Stage 2



weights $\pm 0.0778 - 0.064$



weights $\pm 0.0929 - 0.0784$

Results on Caltech101 with sigmoid non-linearity

Single Stage System: $[64.F_{CSG}^{9 \times 9} - R/N/P^{5 \times 5}] - \log_reg$

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: $[64.F_{CSG}^{9 \times 9} - R/N/P^{5 \times 5}] - [256.F_{CSG}^{9 \times 9} - R/N/P^{4 \times 4}] - \log_reg$

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: $[64.F_{CSG}^{9 \times 9} - R/N/P^{5 \times 5}] - PMK-SVM$

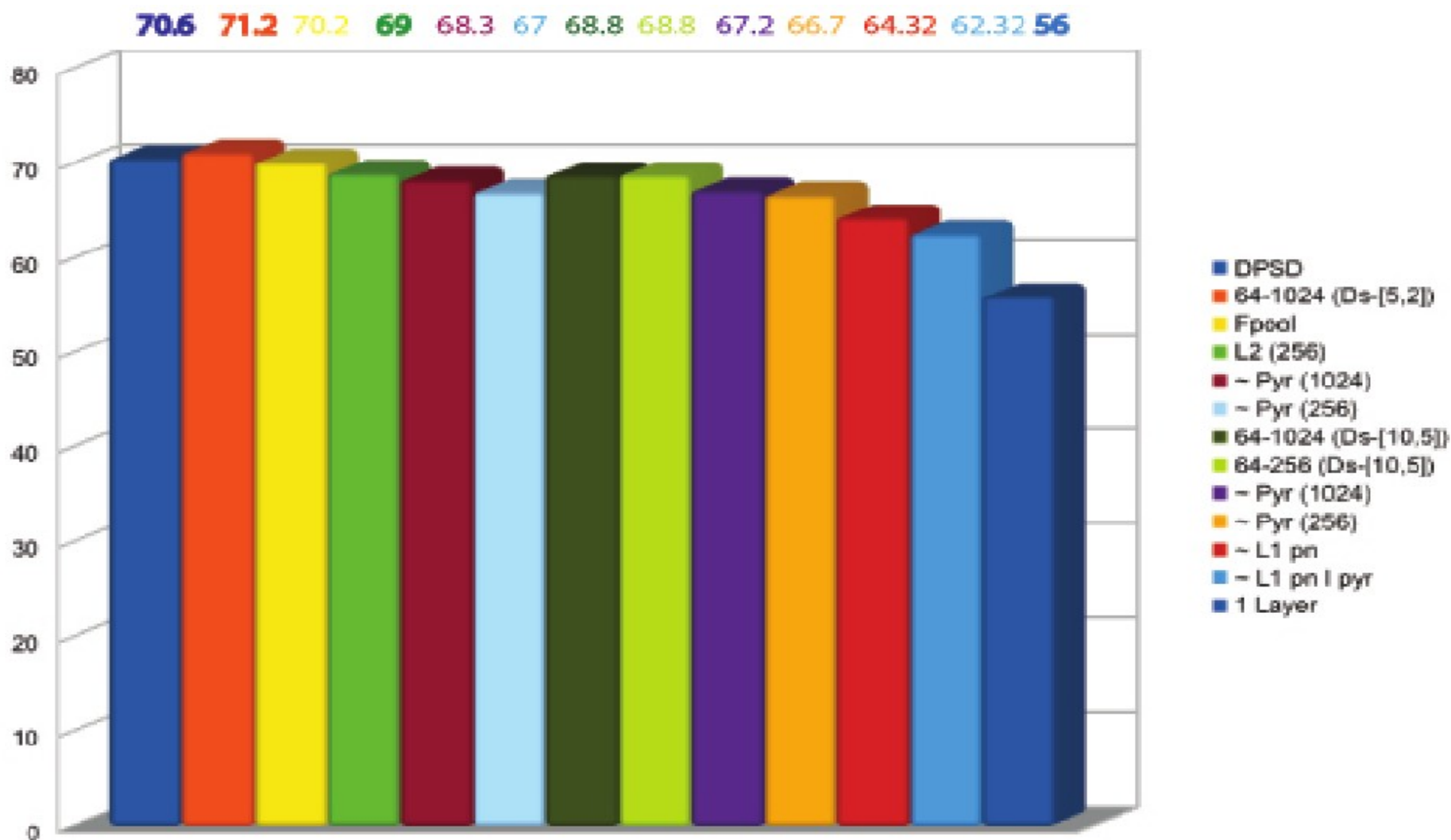
U	64.0%	
---	-------	--

Two Stages: $[64.F_{CSG}^{9 \times 9} - R/N/P^{5 \times 5}] - [256.F_{CSG}^{9 \times 9} - R/N] - PMK-SVM$

UU	52.8%	
----	-------	--

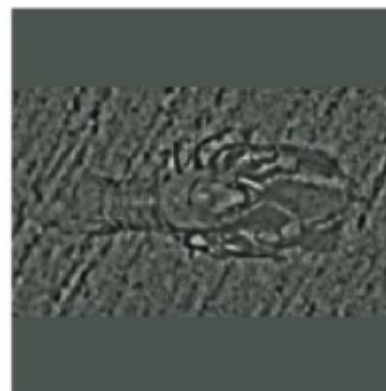
Results on Caltech101: purely supervised with soft-shrink, L2 pooling, contrast normalization

- Supervised learning with soft-shrinkage non-linearity, L2 complex cells, and sparsity penalty on the complex cell outputs: **71%**

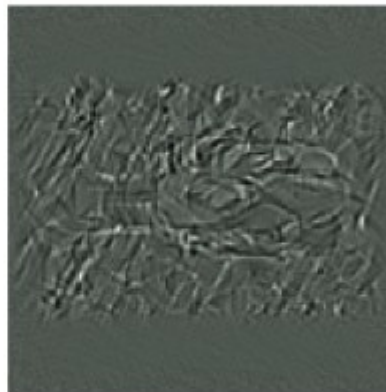
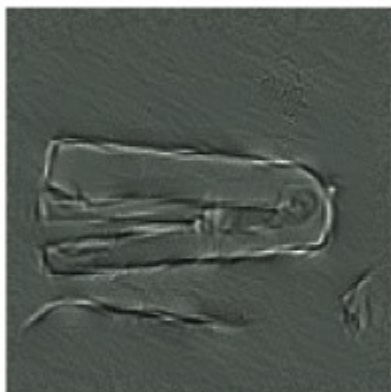


What does Local Contrast Normalization Do?

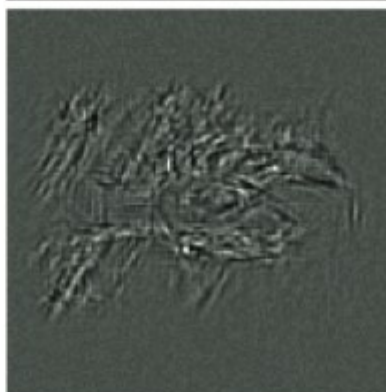
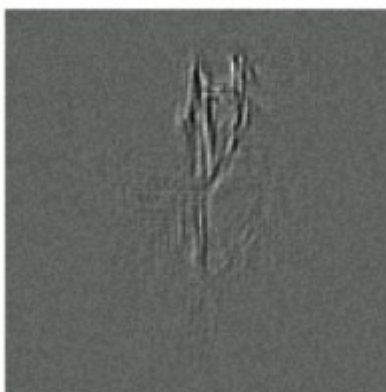
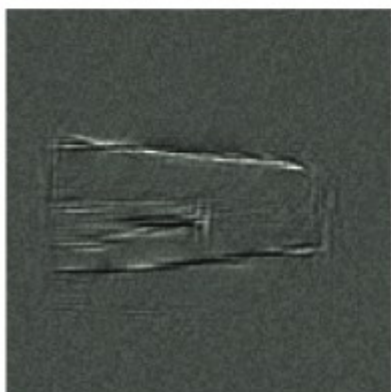
Original



Reconstruction
With LCN

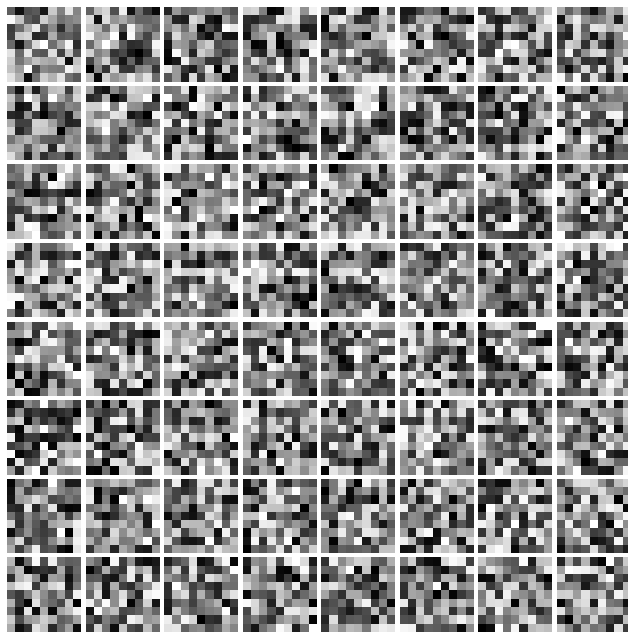


Reconstruction
Without LCN

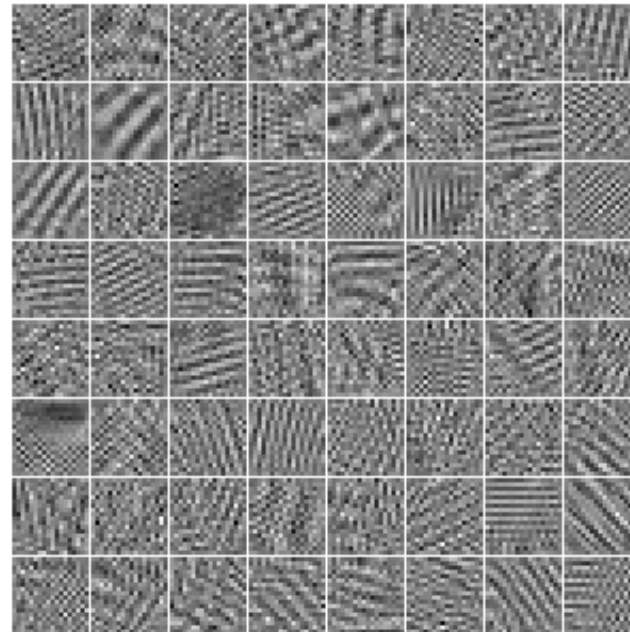
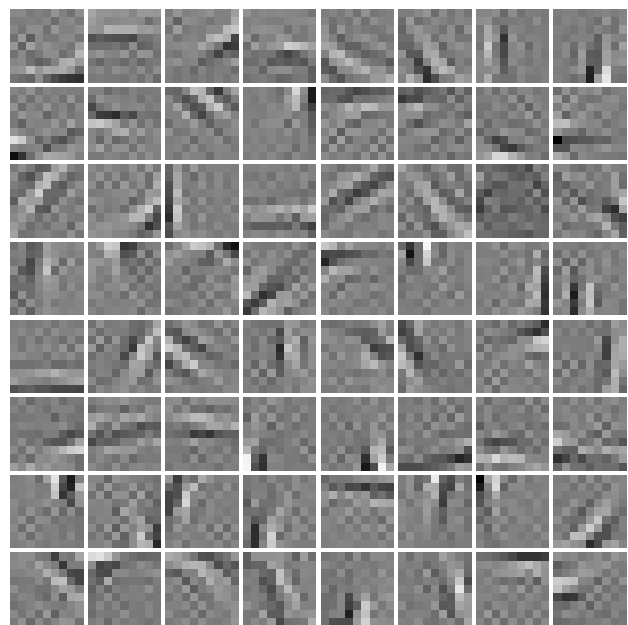


Why Do Random Filters Work?

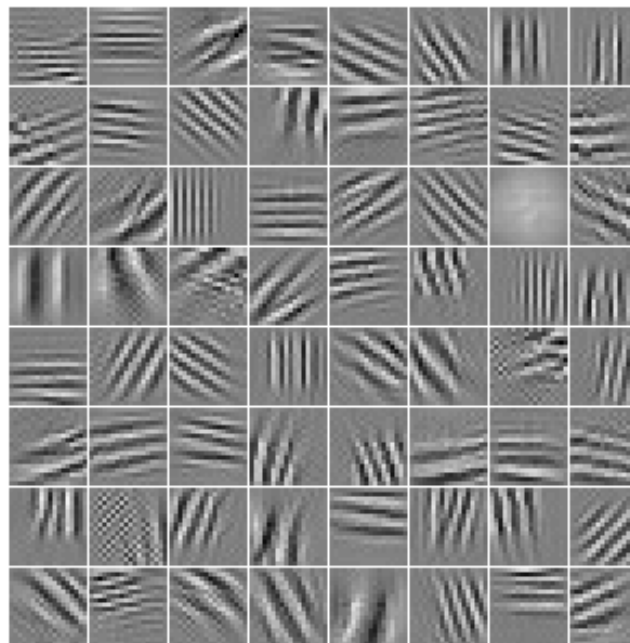
Random
Filters
For
Simple
Cells



Trained
Filters
For
Simple
Cells

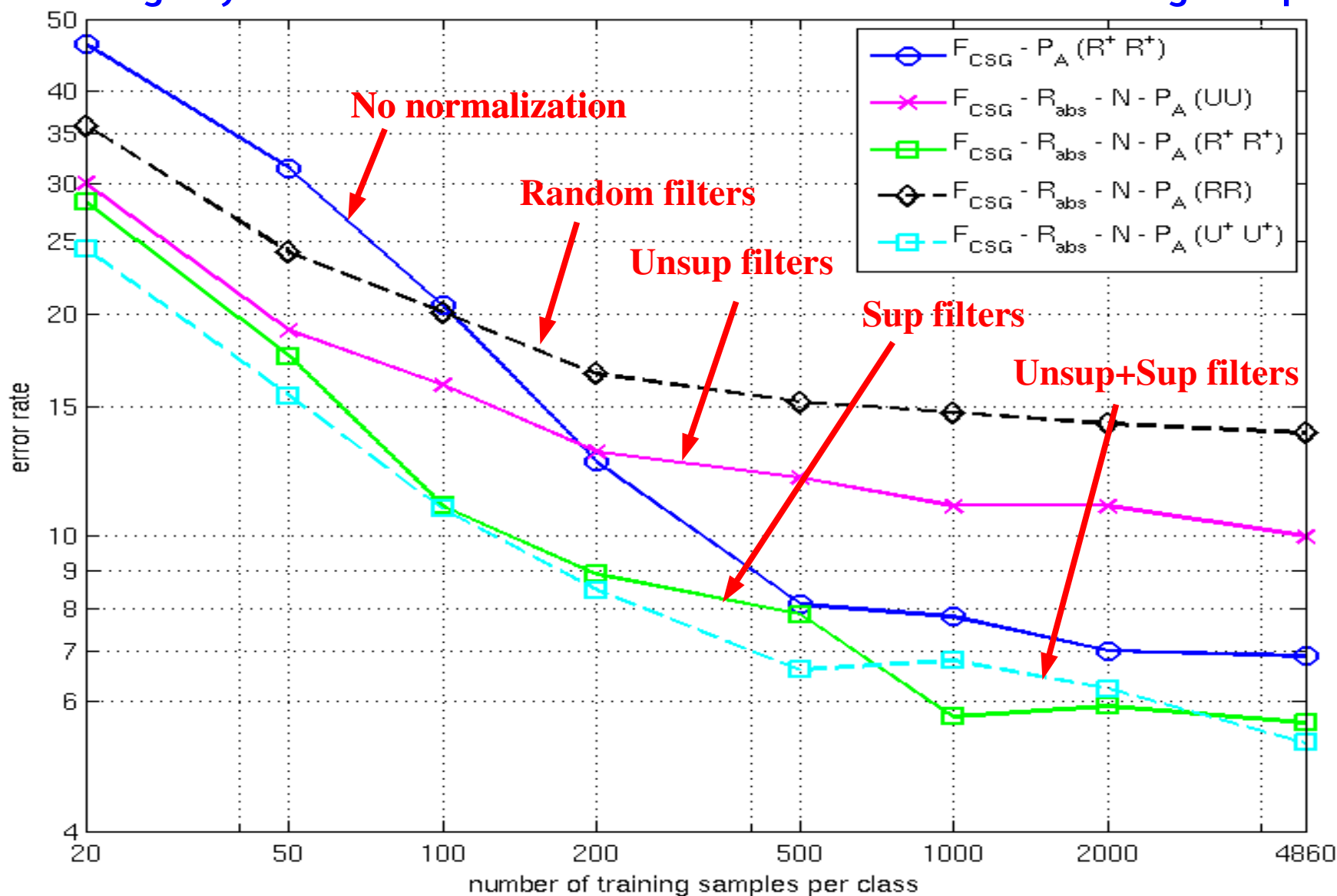


Optimal
Stimuli
for each
Complex
Cell



Small NORB dataset

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



Convolutional Sparse Coding

Convolutional PSD

[Kavukcuoglu, Sermanet, Boureau, Mathieu, LeCun. NIPS 2010]: convolutional PSD

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

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

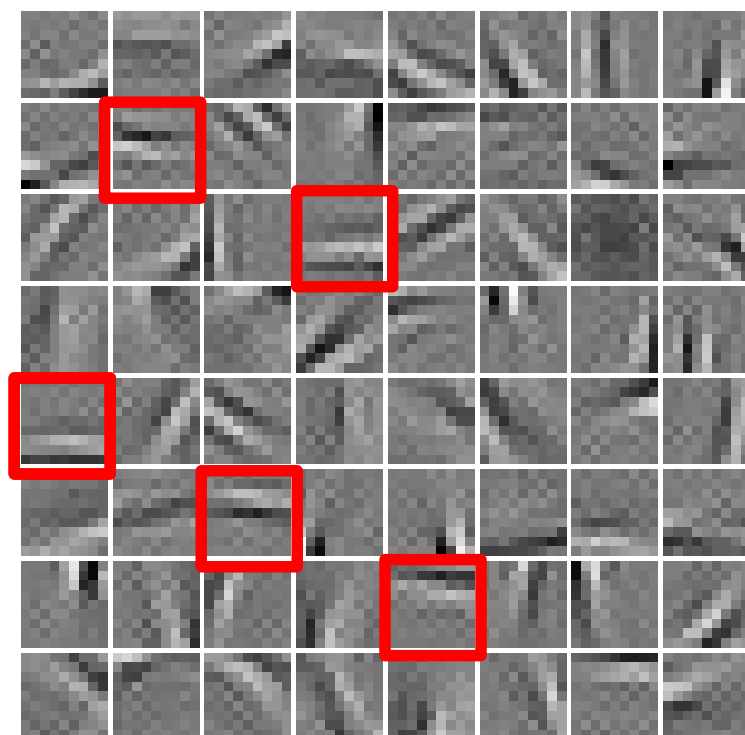
[Norouzi, Ranjbar, Mori, CVPR 2009]: Convolutional Boltzmann Machine

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

Convolutional Training

Problem:

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



weights $[-0.2828 \quad -0.3043$

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

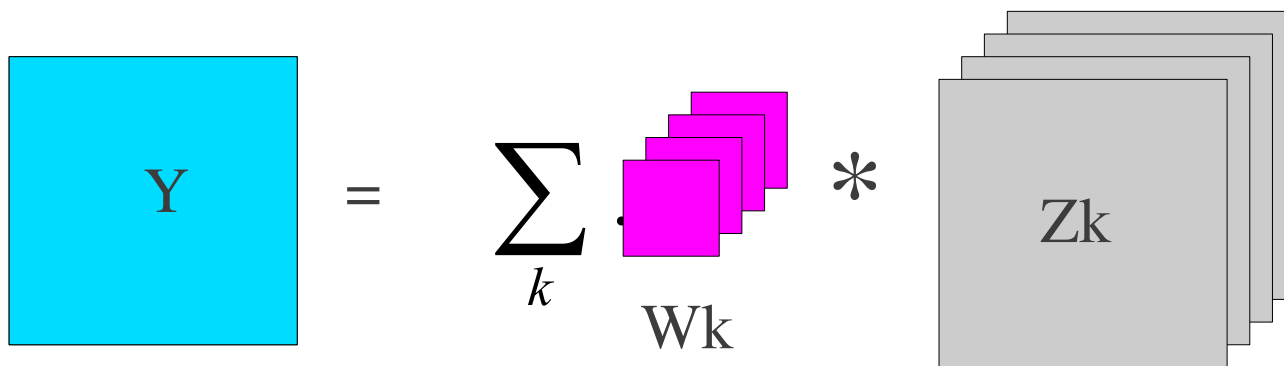
Convolutional Sparse Coding

- Replace the dot products with dictionary element by convolutions.

- ▶ Input Y is a full image
- ▶ Each code component Z_k is a feature map (an image)
- ▶ Each dictionary element is a convolution kernel

- Regular sparse coding** $E(Y, Z) = \|Y - \sum_k W_k Z_k\|^2 + \alpha \sum_k |Z_k|$

- Convolutional S.C.** $E(Y, Z) = \|Y - \sum_k W_k * Z_k\|^2 + \alpha \sum_k |Z_k|$



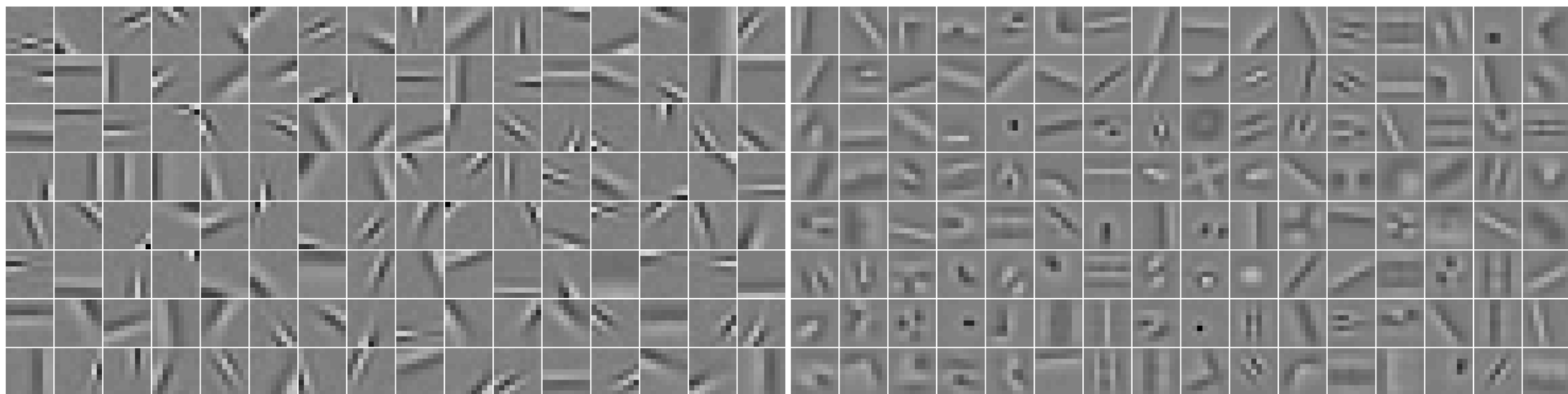
“deconvolutional networks” [Zeiler, Taylor, Fergus CVPR 2010]

Convolutional PSD: Encoder with a soft sh() Function

Convolutional Formulation

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

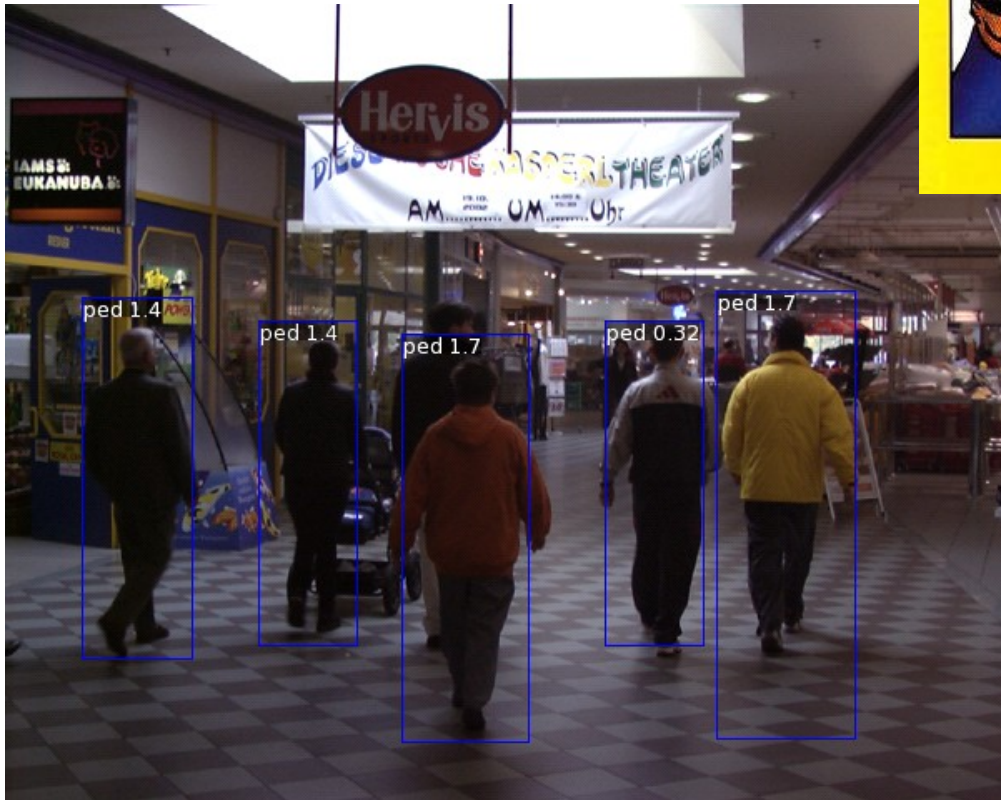
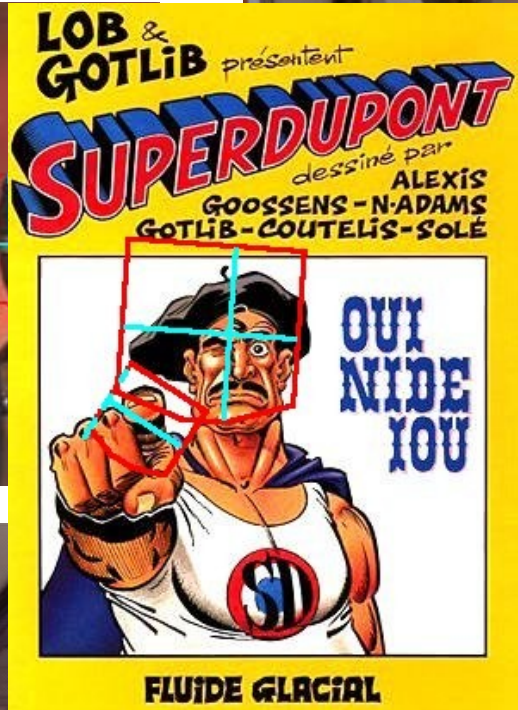
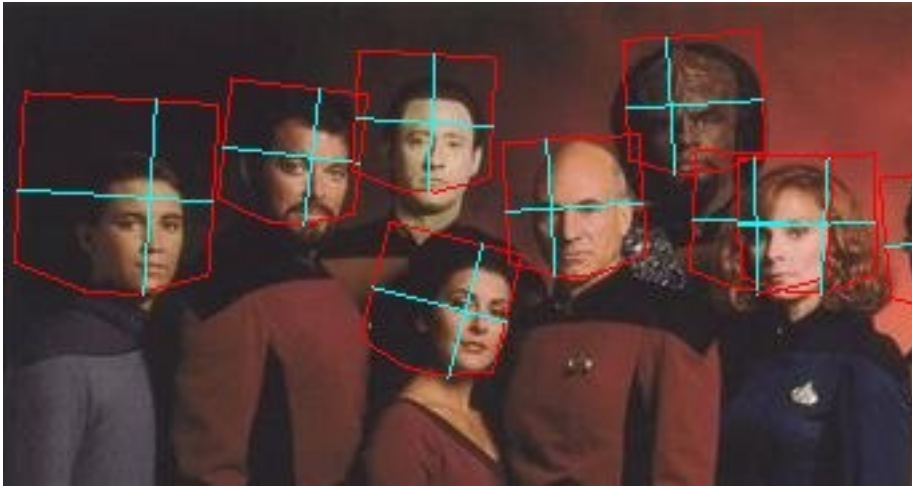
$$\mathcal{L}(x, z, \mathcal{D}) = \frac{1}{2} \left\| x - \sum_{k=1}^K \mathcal{D}_k * z_k \right\|_2^2 + \sum_{k=1}^K \left\| z_k - f(W^k * x) \right\|_2^2 + |z|_1$$



- ▶ **PATCH** based learning

- ▶ **CONVOLUTIONAL** learning

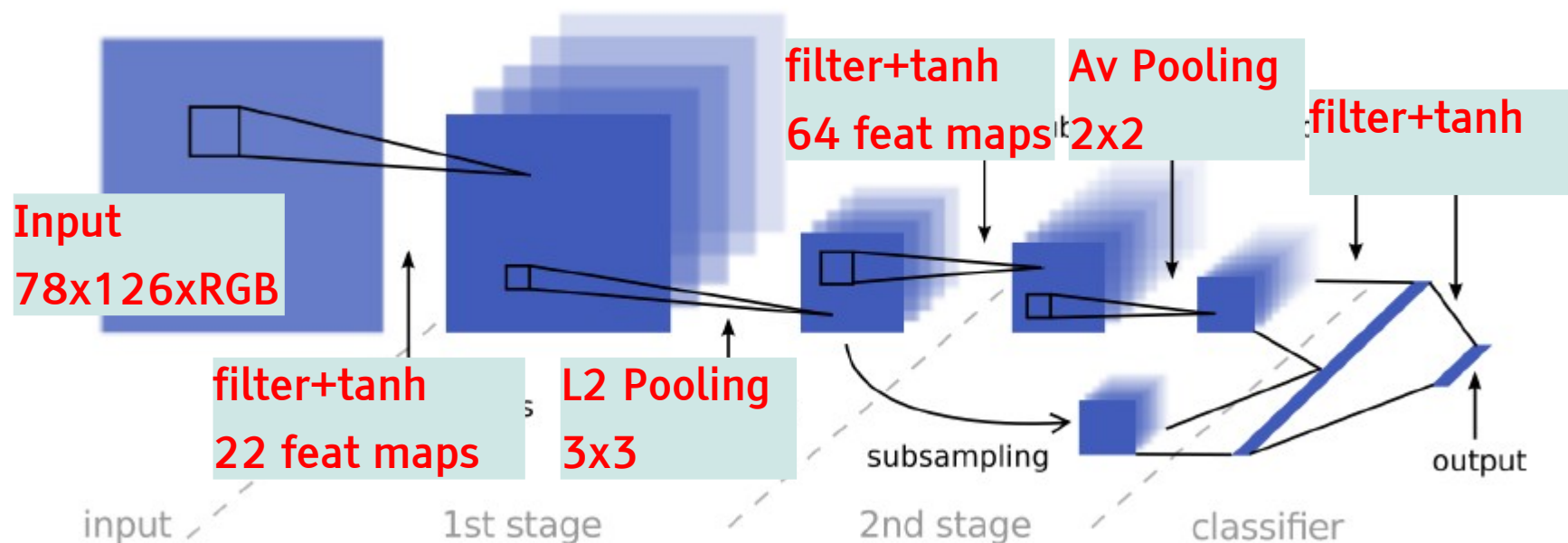
Pedestrian Detection, Face Detection



ConvNet Architecture with Multi-Stage Features

- Feature maps from all stages are pooled/subsampled and sent to the final classification layers

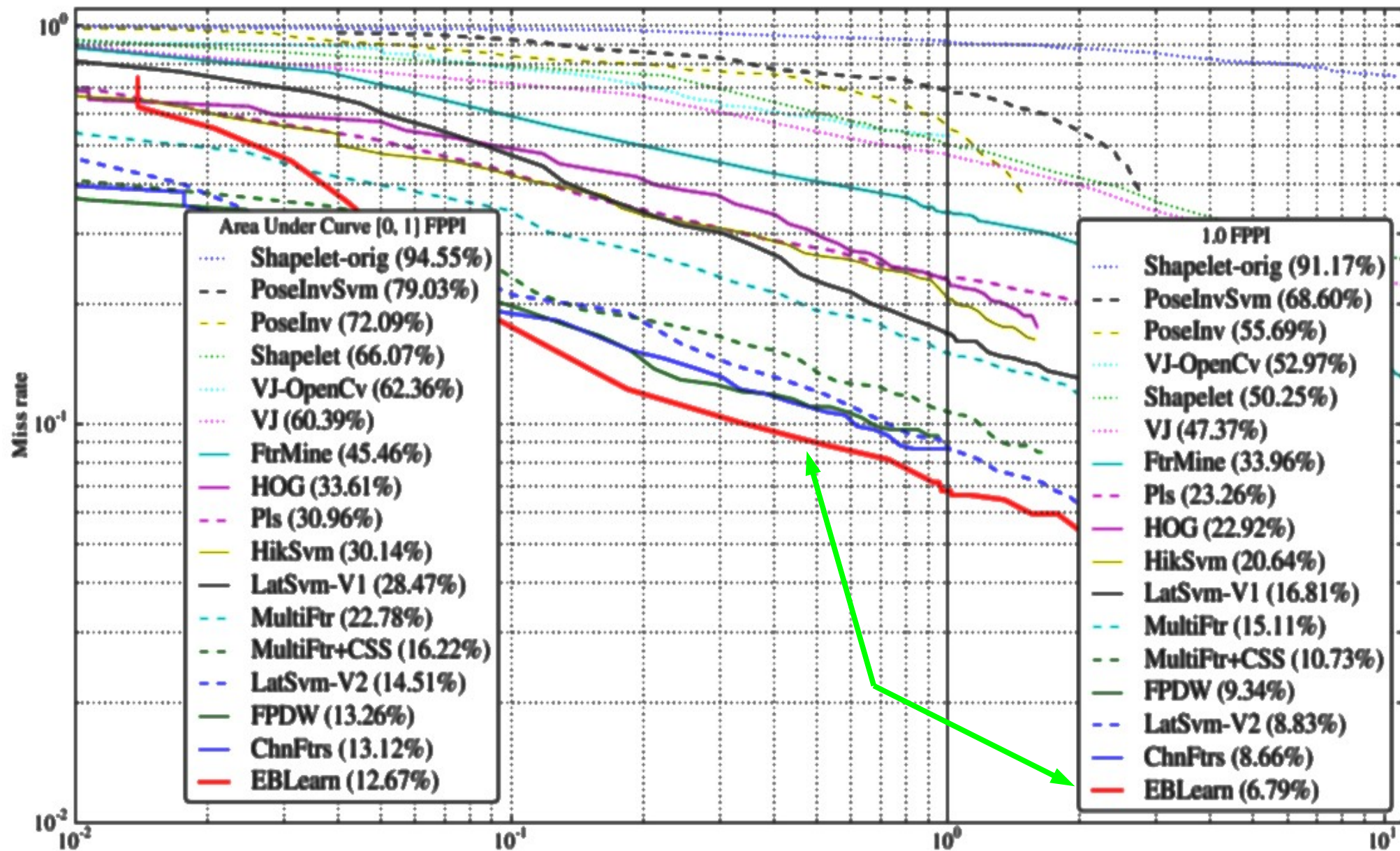
- Pooled low-level features: good for textures and local motifs
- High-level features: good for “gestalt” and global shape



Task	Single-Stage features	Multi-Stage features	Improvement %
Pedestrians detection (INRIA) [9]	14.26%	9.85%	31%
Traffic Signs classification (GTSRB) [11]	1.80%	0.83%	54%
House Numbers classification (SVHN)	5.72%	5.67%	0.9%

[Sermanet, Chintala, LeCun ArXiv:1204.3968, 2012]

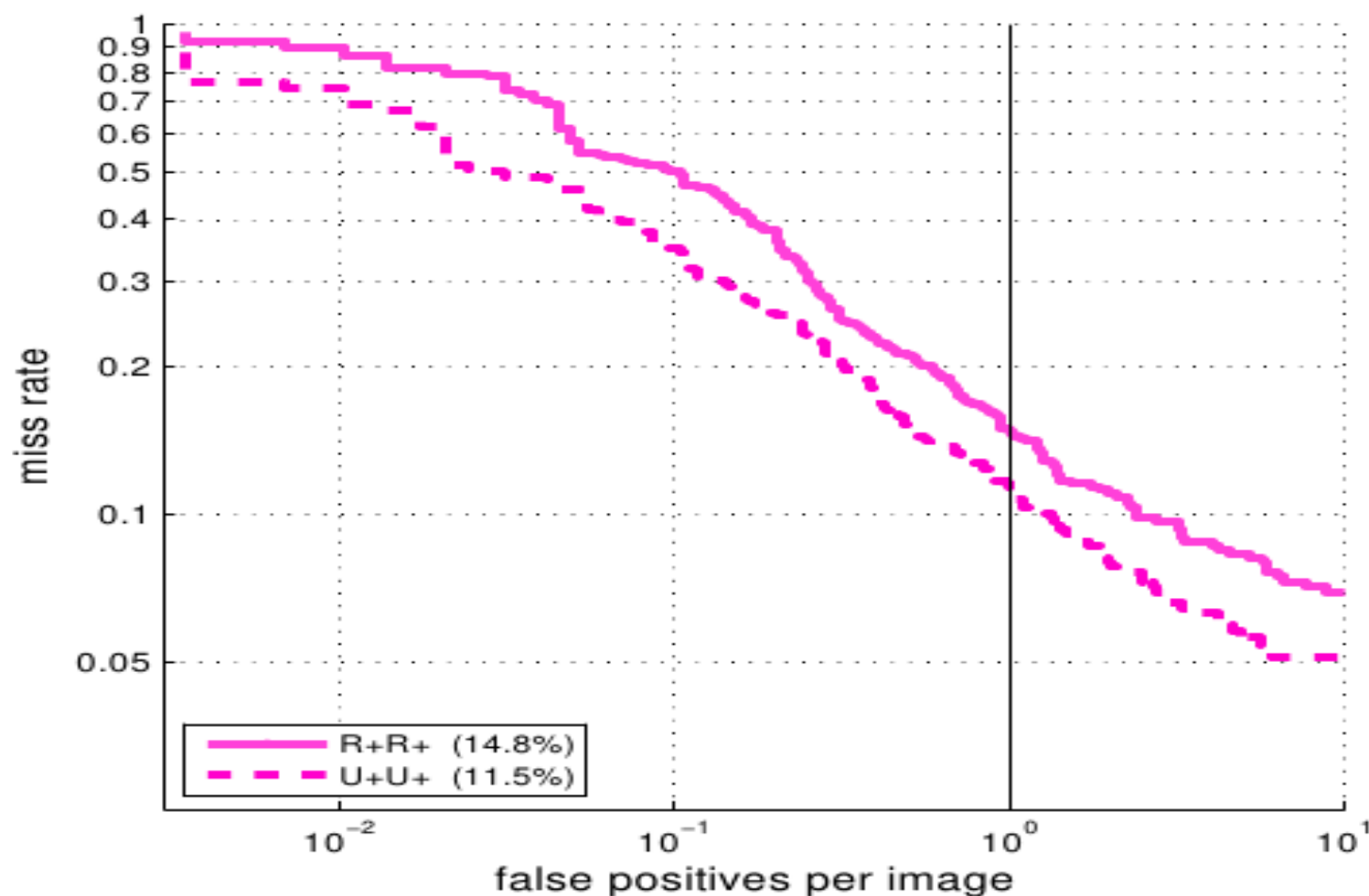
Pedestrian Detection (INRIA Dataset)



[Kavukcuoglu et al. NIPS 2010]

Convolutional PSD pre-training for pedestrian detection

- ConvPSD pre-training improves the accuracy of pedestrian detection over purely supervised training from random initial conditions.

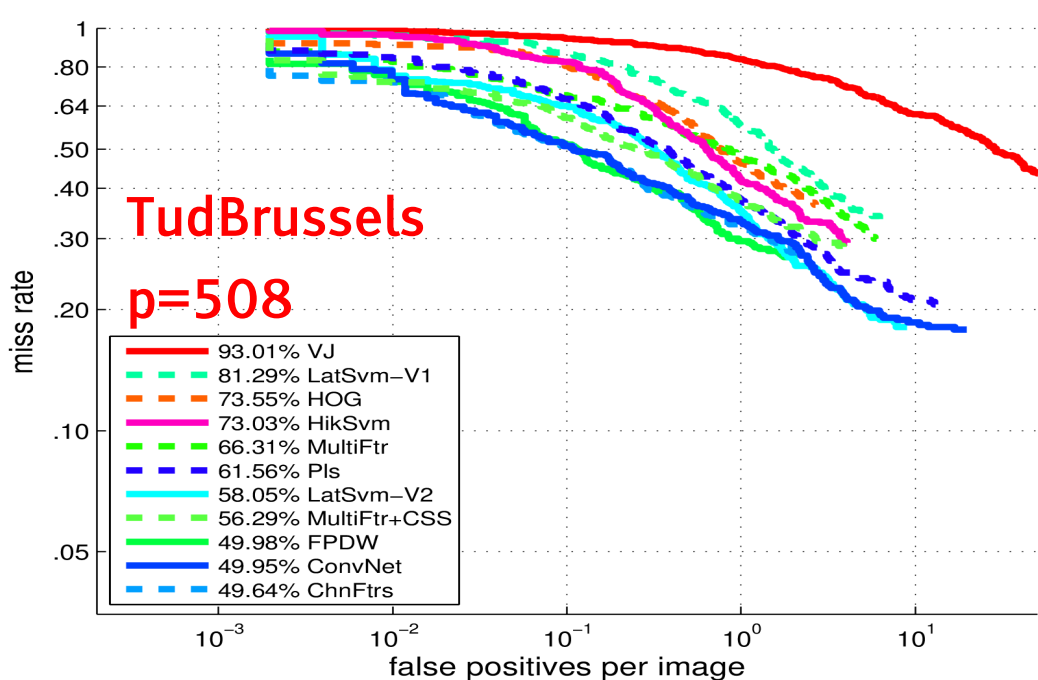
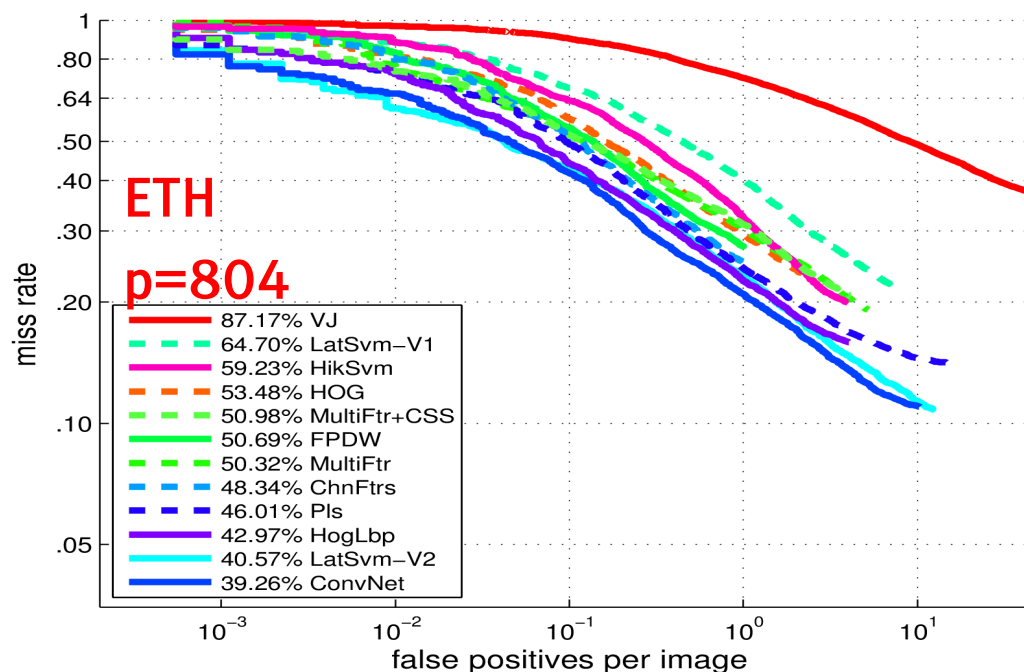
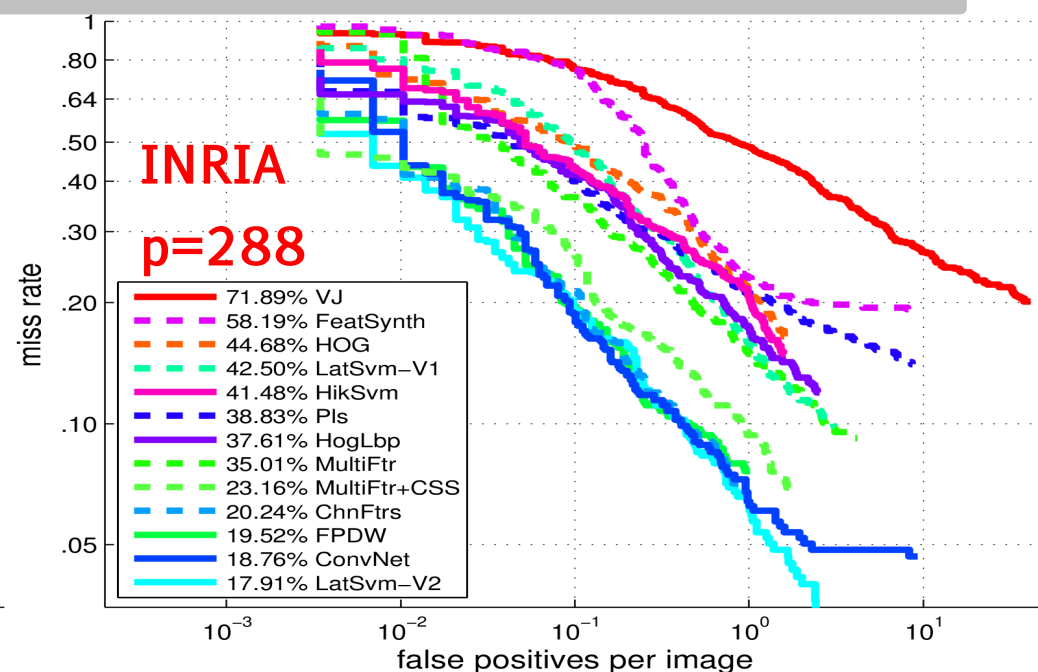
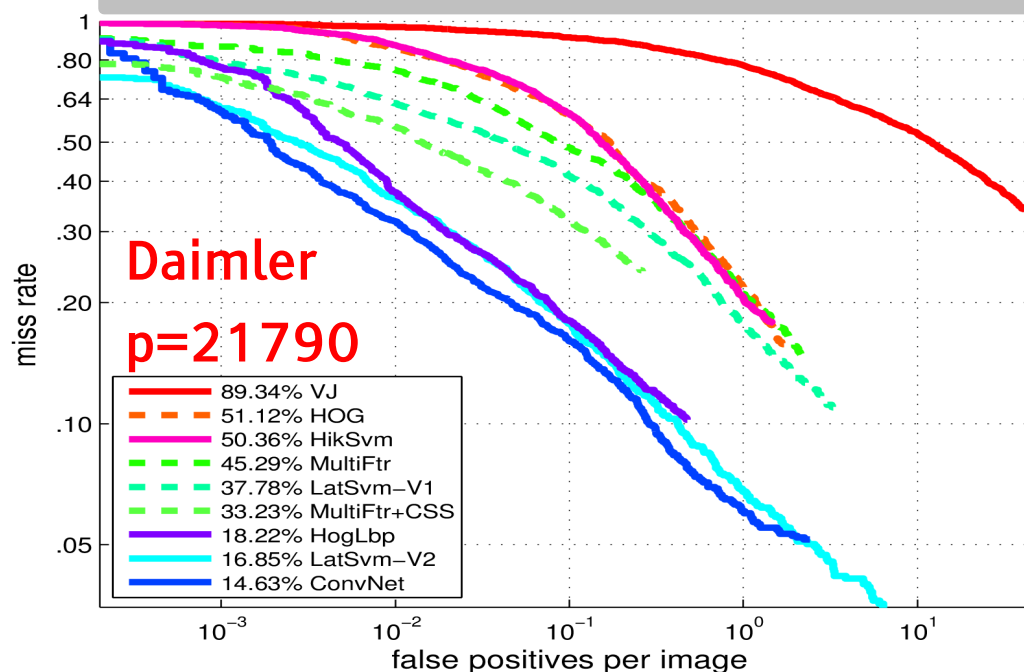


Convolutional PSD pre-training for pedestrian detection

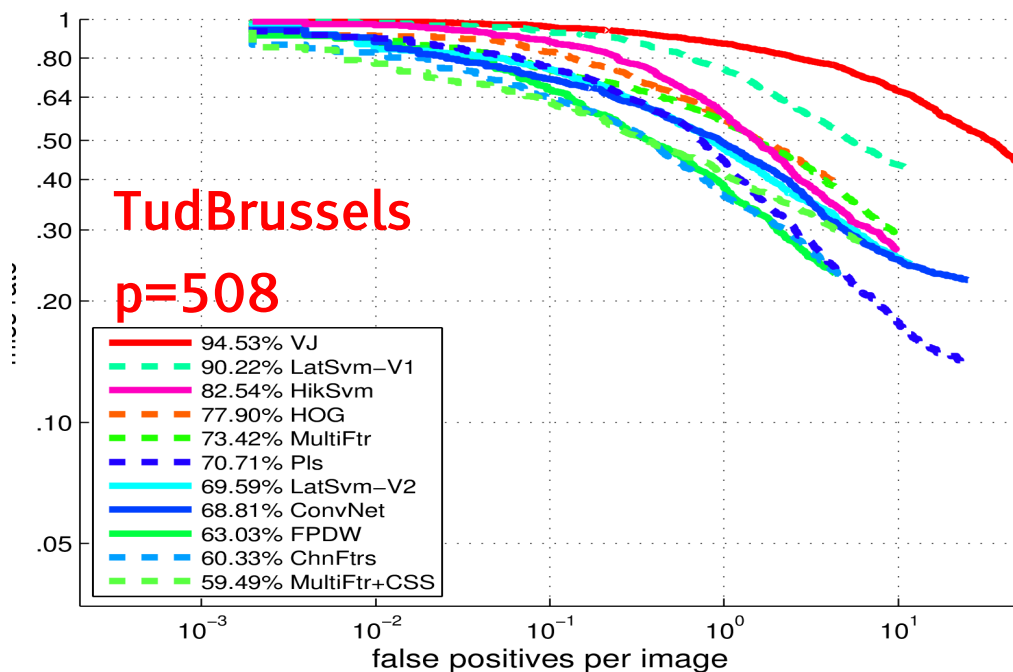
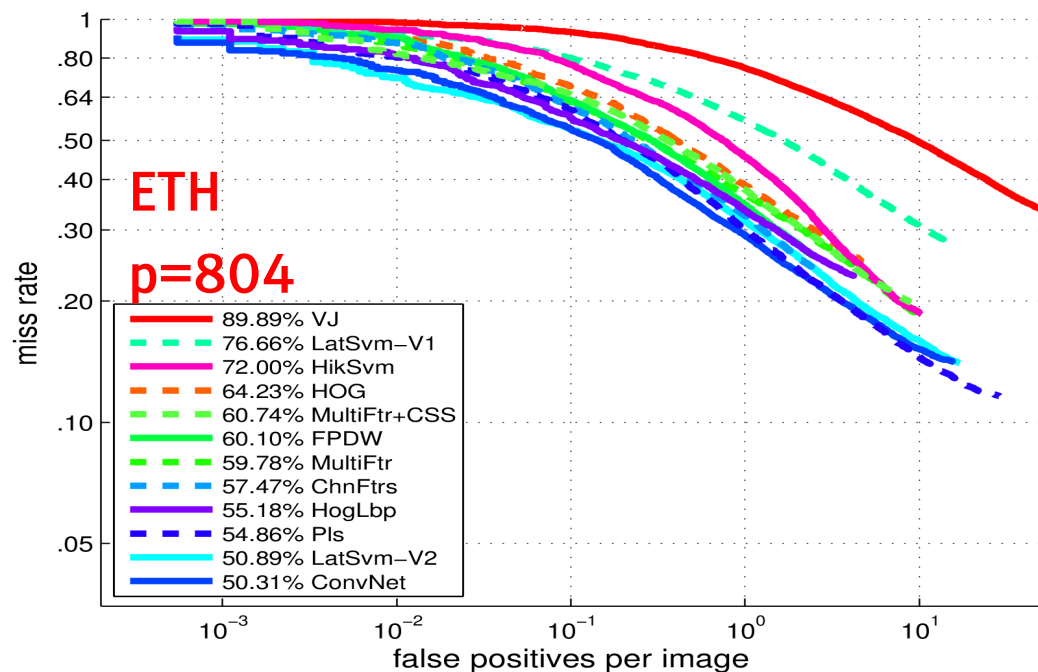
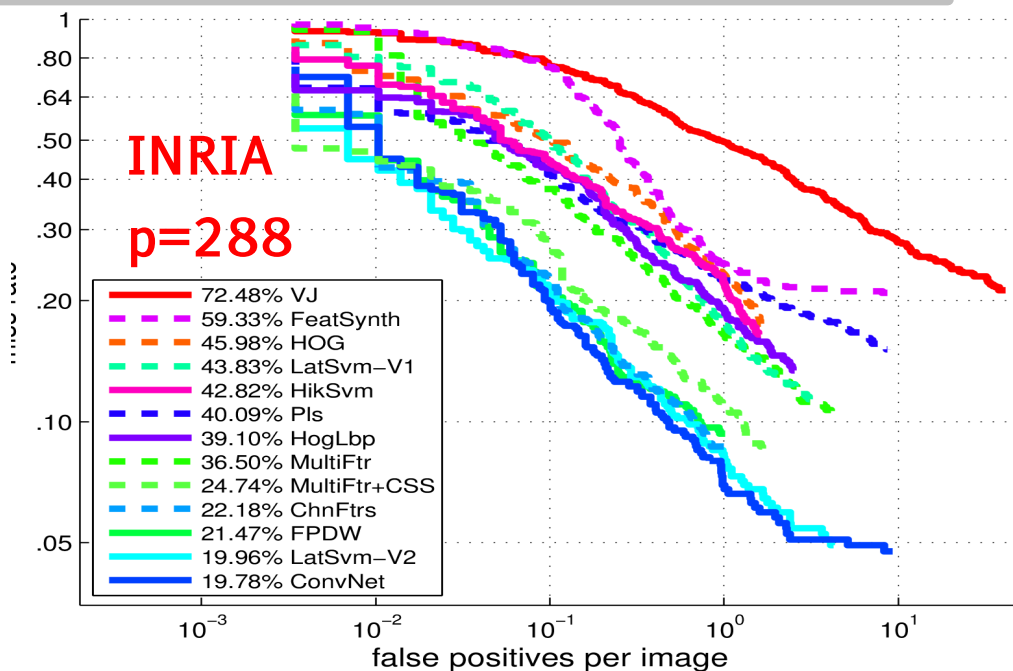
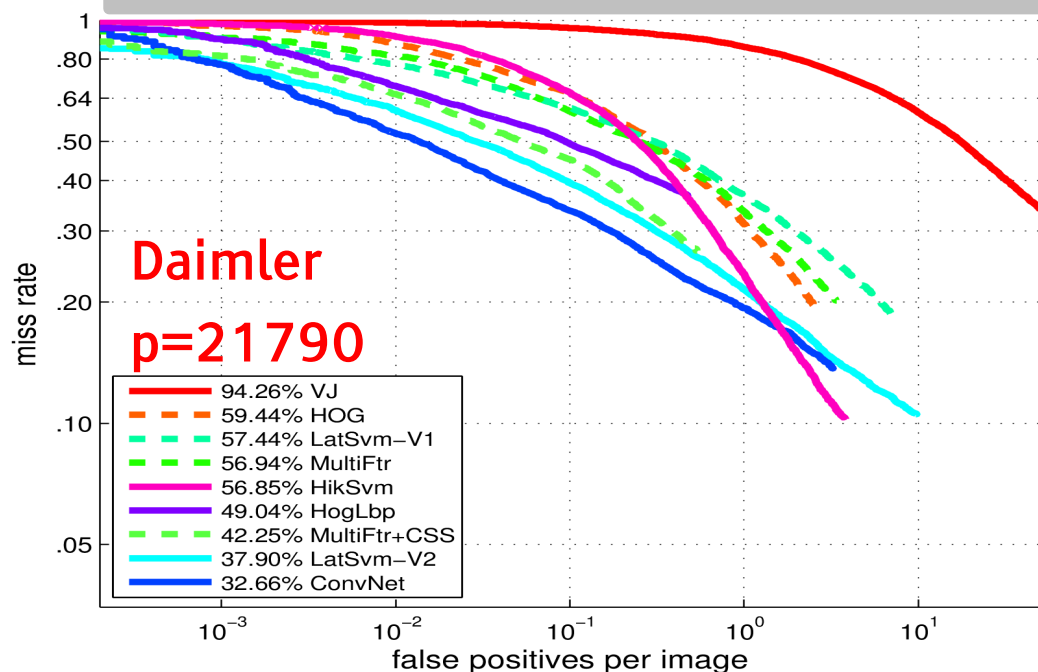
- Trained on INRIA. Tested on INRIA, Daimler, TUD-Brussels, ETH
- Same testing protocol as in [Dollar et al. T.PAMI 2011]

	INRIA (all positives labeled)		INRIA (missing positive labels)	Caltech	Daimler	TUD	ETH
# test images	288		288	4024	21790	508	1804
Error Percentage type	1FPPI	AUC	AUC	AUC	AUC	AUC	AUC
Reasonable: at least 50 pixels tall + no or partial occlusion							
ConvNet's rank			1st	12th	1st	4th	1st
ConvNet			19.78 %	77.2 %	32.66 %	68.81 %	50.31 %
LatSvm-V2			19.96 %	63.26 %	37.9 %	69.59 %	50.89 %
ChnFtrs			22.18 %	56.34 %	-	60.33 %	57.47 %
MultiFtr+CSS			24.74 %	60.89 %	42.25 %	59.49 %	60.74 %
FPDW			21.47 %	57.40 %	-	63.03 %	60.10 %
Near: at least 80 pixels tall							
ConvNet's rank	1st	1st	2nd	4th	1st	2nd	1st
ConvNet	5.77 %	11.26 %	18.76 %	36.68 %	14.63 %	49.95 %	39.26 %
LatSvm-V2	8.83 %	14.51 %	17.91 %	34.27 %	16.85 %	58.05 %	40.57 %
ChnFtrs	8.66 %	13.12 %	20.24 %	35.12 %	-	49.64 %	48.34 %
HogLpb	-	-	37.61 %	30.79 %	18.22 %	-	42.97 %
Large: at least 100 pixels tall							
ConvNet's rank			2nd	1st	2nd	2nd	1st
ConvNet			17.27 %	21.37 %	11.81 %	41.15 %	34.56 %
LatSvm-V2			16.31 %	28.23 %	11.52 %	54.16 %	35.27 %
HogLpb			37.71 %	22.67 %	20.2 %	-	40.84 %

Results on "Near Scale" Images (>80 pixels tall, no occlusions)



Results on "Reasonable" Images (>50 pixels tall, few occlusions)



Musical Genre Recognition

Same Architecture, Different Data

[Henaff et al. ISMIR 2011]

Convolutional PSD Features on Time-Frequency Signals

Input: “Constant Q Transform” over 46.4ms windows (1024 samples)

- ▶ 96 filters, with frequencies spaced every quarter tone (4 octaves)

Architecture:

- ▶ Input: sequence of contrast-normalized CQT vectors
- ▶ 1: PSD features, 512 trained filters
- ▶ 2: shrinkage function → rectification
- ▶ 3: pooling over 5 seconds
- ▶ 4: linear SVM classifier
- ▶ 5: pooling of SVM categories over 30 seconds

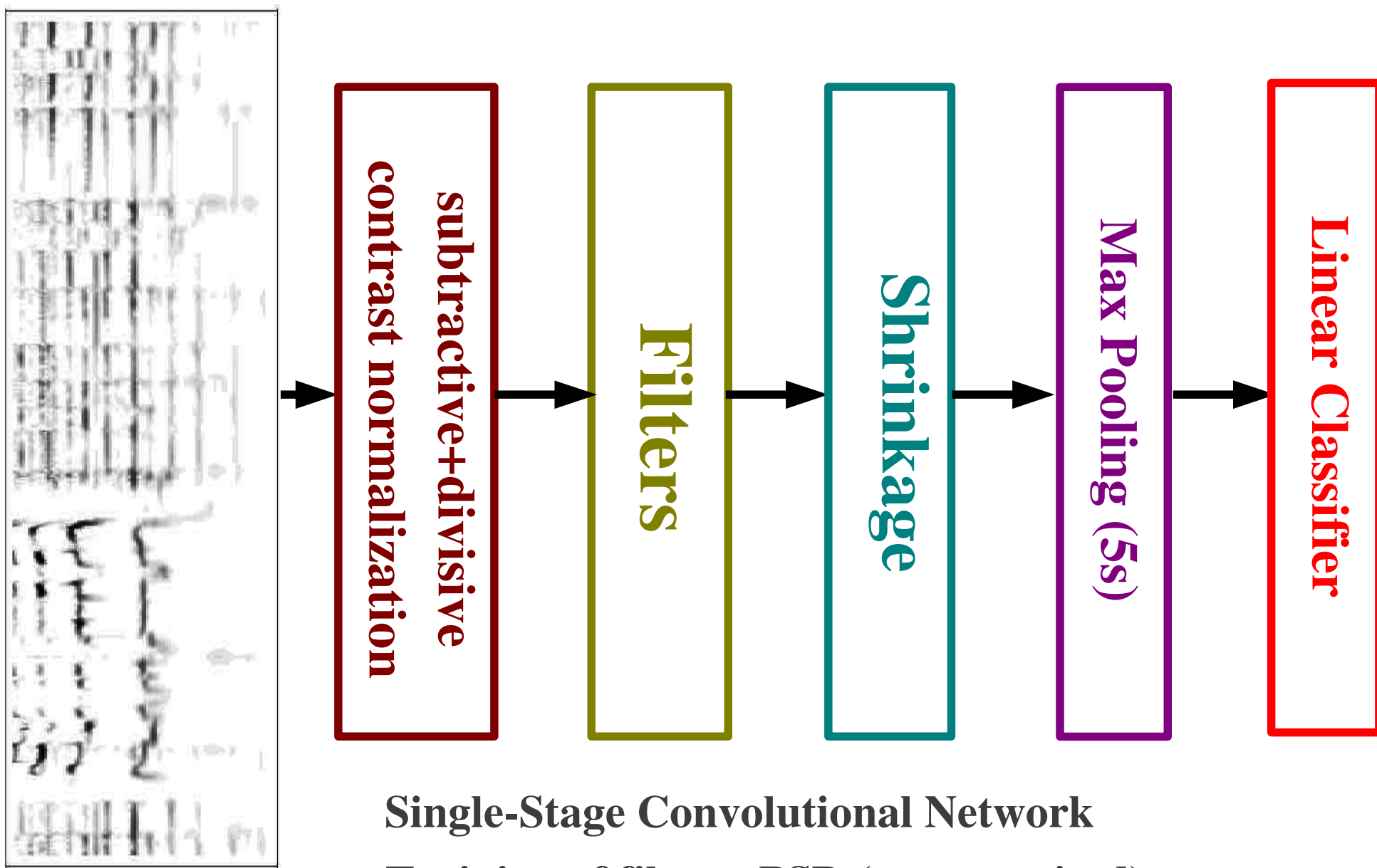
GTZAN Dataset

- ▶ 1000 clips, 30 second each
- ▶ 10 genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and rock.

Results

- ▶ 84% correct classification
- ▶ (state of the art is at 92% with many features)

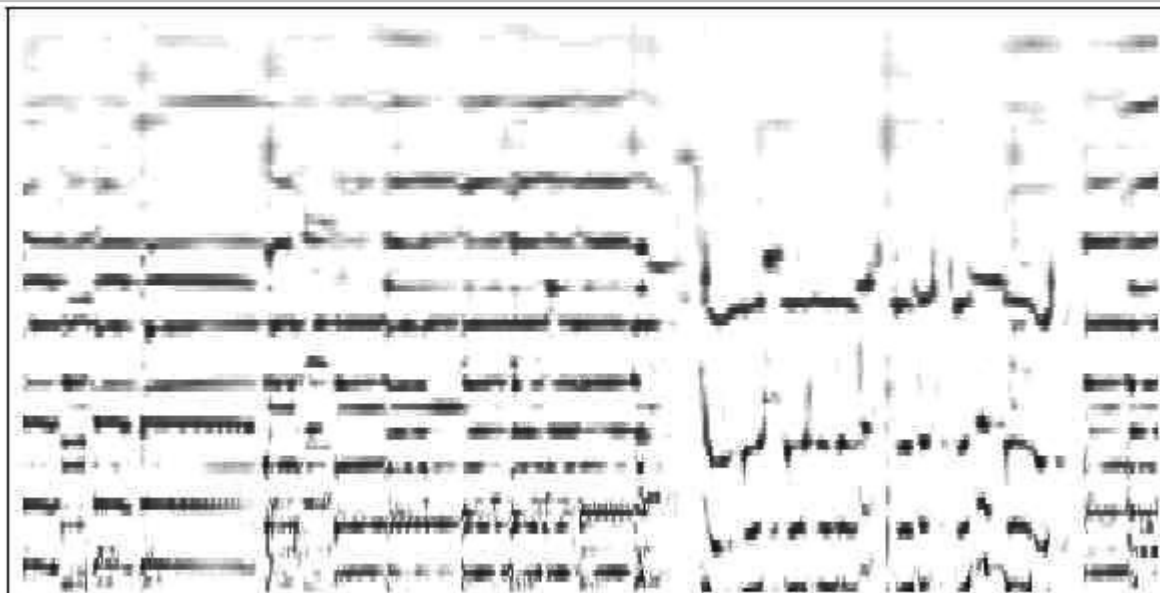
Architecture: contrast norm $\rightarrow$ filters $\rightarrow$ shrink $\rightarrow$ max pooling



Single-Stage Convolutional Network

Training of filters: PSD (unsupervised)

Constant Q Transform over 46.4 ms → Contrast Normalization

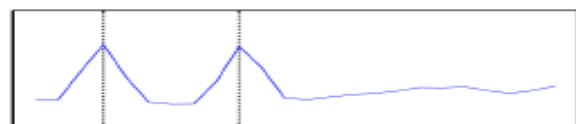


subtractive+divisive contrast normalization

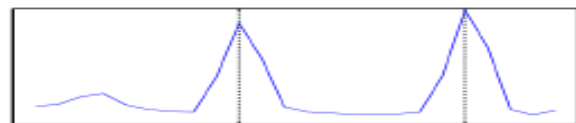


Convolutional PSD Features on Time-Frequency Signals

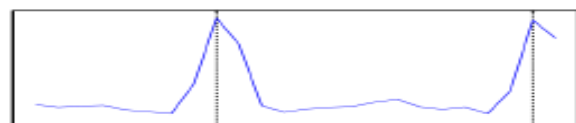
Octave-wide features



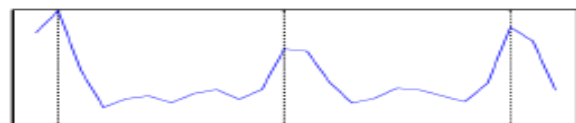
(a)



(b)



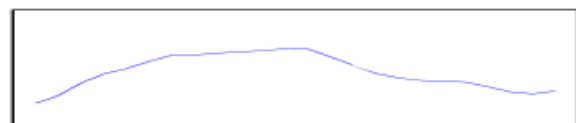
(c)



(d)



(e)



(f)

Minor 3rd

Perfect 4th

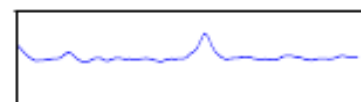
Perfect 5th

Quartal chord

Major triad

transient

full 4-octave features

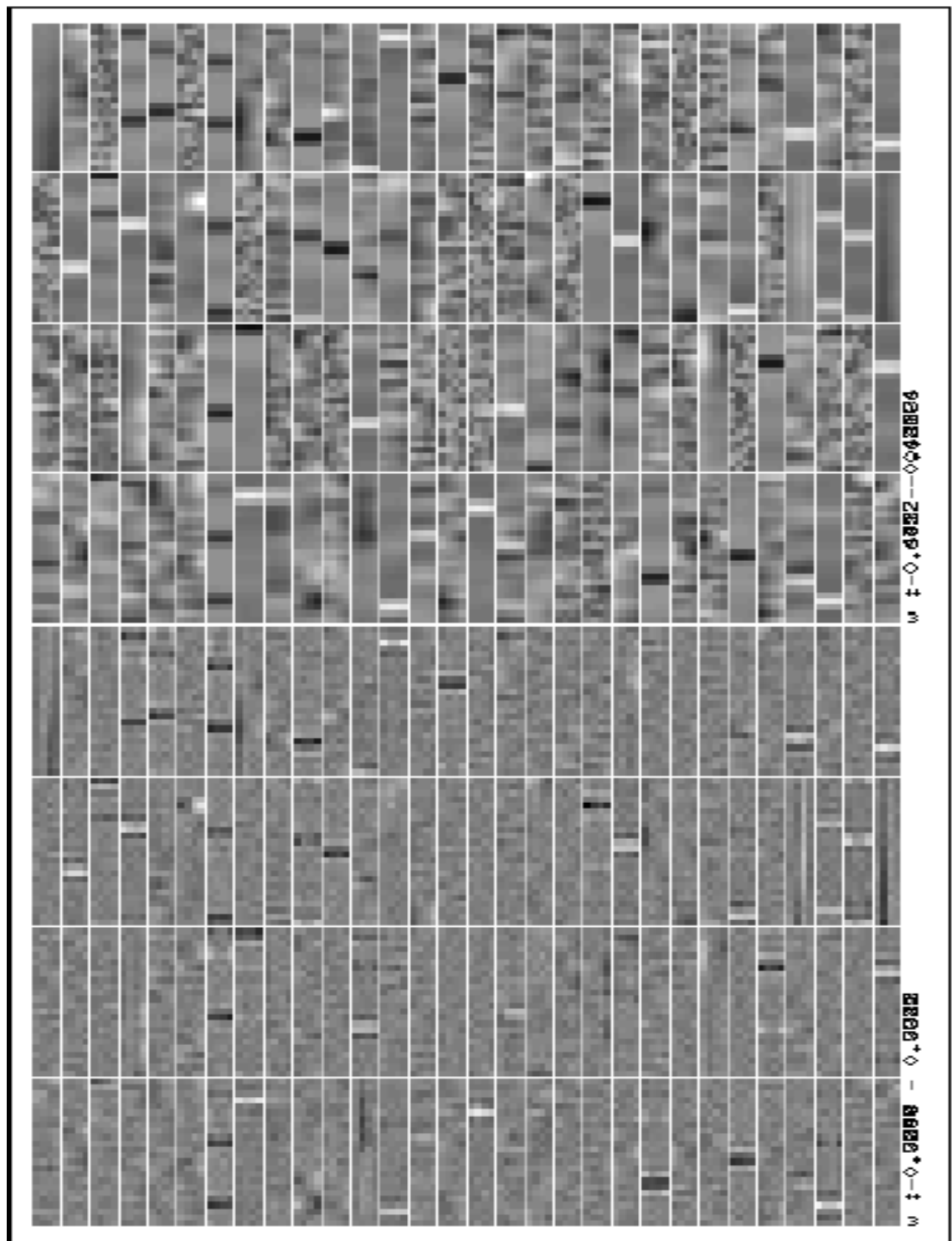


PSD Features on Constant-Q Transform

● Octave-wide features

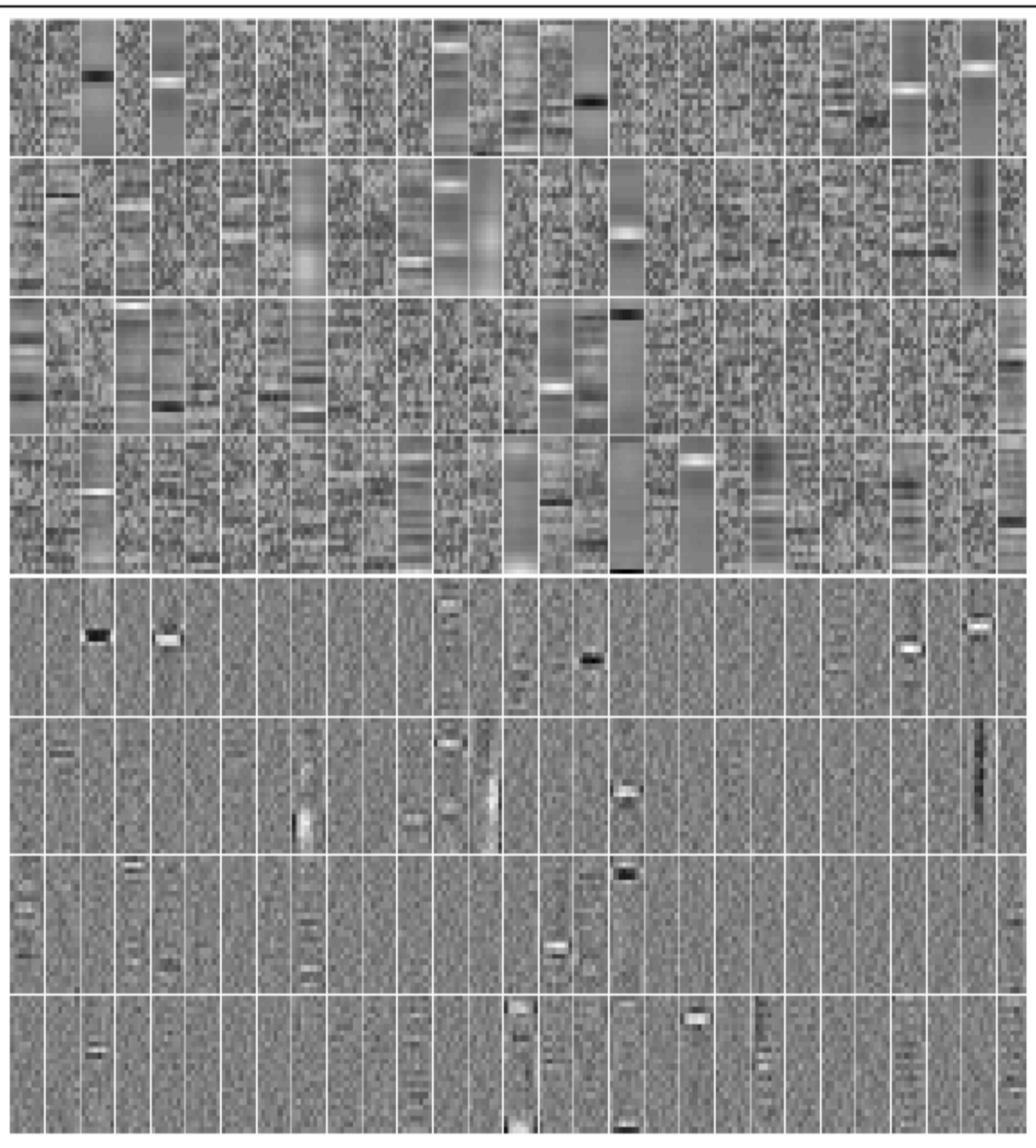
► Encoder basis functions

► Decoder basis functions



Time-Frequency Features

- Octave-wide features on 8 successive acoustic vectors
 - Almost no temporal structure in the filters!



Accuracy on GTZAN dataset (small, old, etc...)

● Accuracy: 83.4%. State of the Art: 84.3%

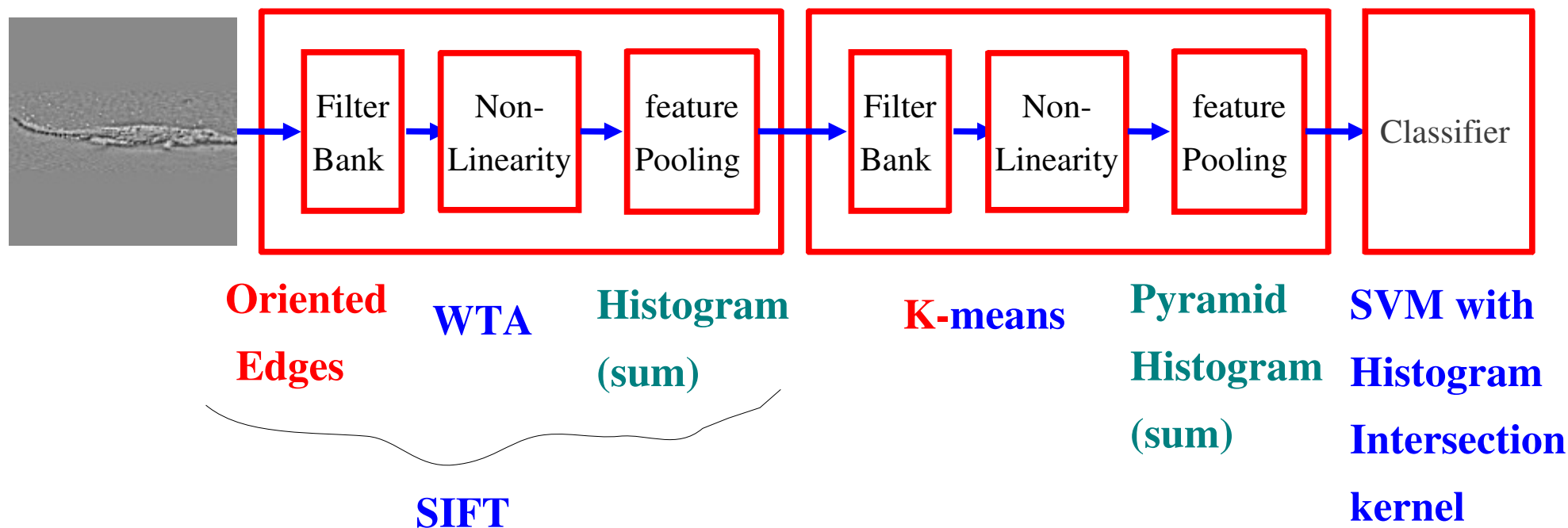
● Very fast

Classifier	Features	Acc. (%)
CSC	Many features [6]	92.7
SRC	Auditory cortical feat. [25]	92
RBF-SVM	Learned using DBN [12]	84.3
Linear SVM	Learned using PSD on octaves	83.4 ± 3.1
AdaBoost	Many features [2]	83
Linear SVM	Learned using PSD on frames	79.4 ± 2.8
SVM	Daubechies Wavelets [19]	78.5
Log. Reg.	Spectral Covariance [3]	77
LDA	MFCC + other [18]	71
Linear SVM	Auditory cortical feat. [25]	70
GMM	MFCC + other [29]	61

Learning Mid-Level Features

[Boureau et al., CVPR 2010, ICML 2010, CVPR 2011]

ConvNets and “Conventional” Vision Architectures are Similar



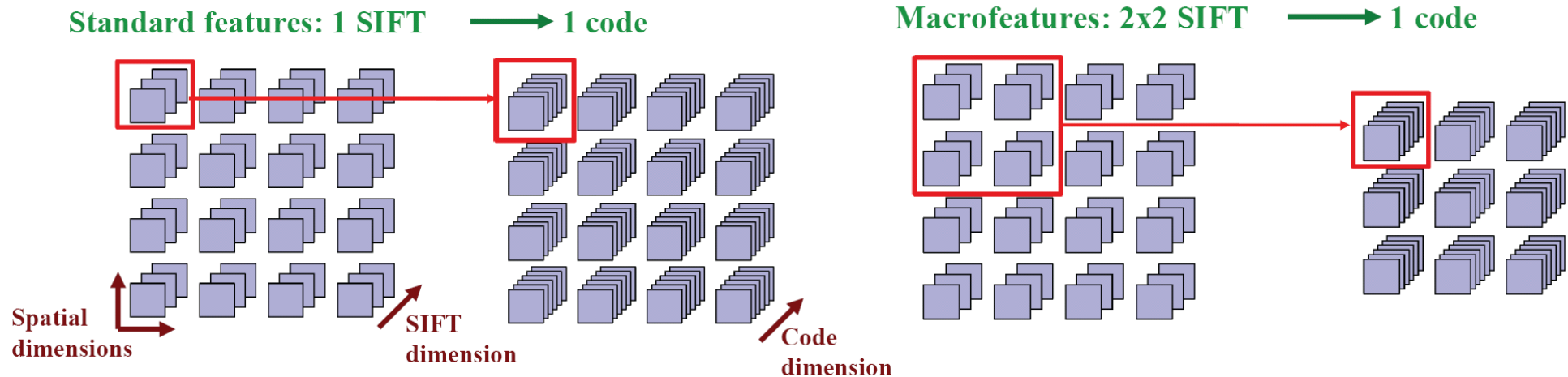
- Can't we use the same tricks as ConvNets to train the second stage of a “conventional vision architecture?”
- Stage 1: SIFT
- Stage 2: sparse coding over neighborhoods + pooling

Using DL/ConvNet ideas in “conventional” recognition systems

[Boureau et al. CVPR 2010]

Adapting insights from ConvNets:

- ▶ Jointly encoding spatial neighborhoods instead of single points: increase spatial receptive fields for higher-level features



- ▶ Use max pooling instead of average pooling
- ▶ Train supervised dictionary for sparse coding

This yields state-of-the-art results:

- ▶ **75.7%** on Caltech-101 (+/-1.1%): record for single system
- ▶ **85.6%** on 15-Scenes (+/- 0.2): record!

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

Replacing K-means with Sparse Coding

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

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

Sparse Coding within Clusters

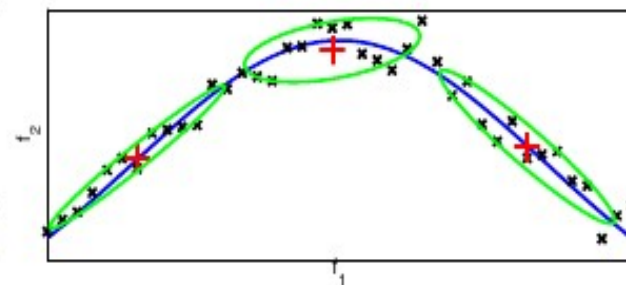
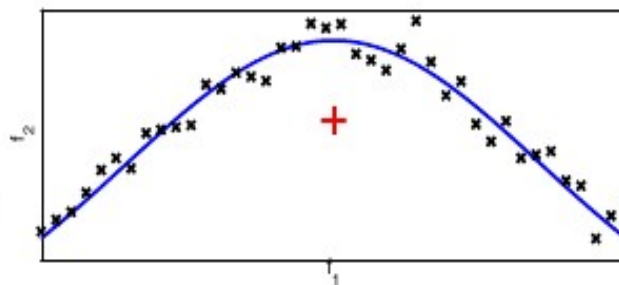
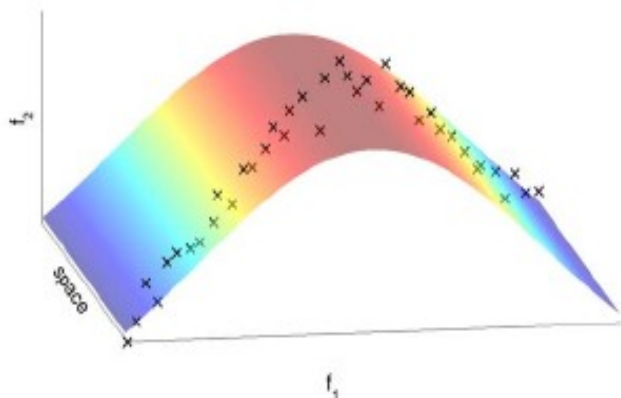
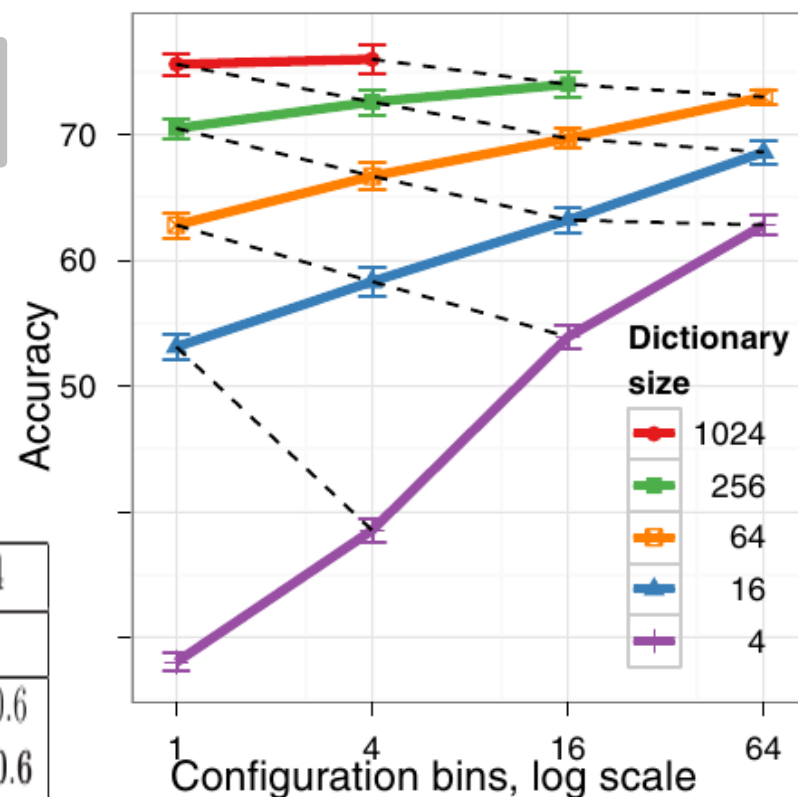
Splitting the Sparse Coding into Clusters

- only similar things get pooled together

[Boureau, et al. CVPR 2011]

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

Results on Caltech 101 (30 training examples per class), and 15-Scenes (100 training examples per class) as a func



Learning Invariant Features (learning complex cells)

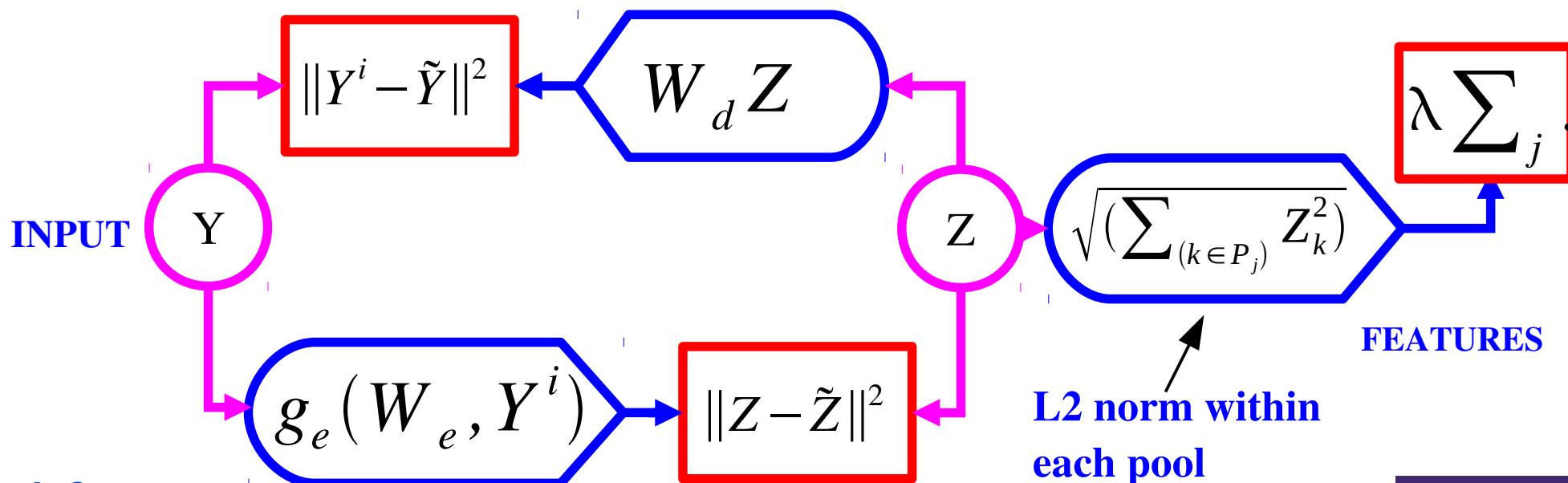
[Kavukcuoglu, Ranzato, Fergus, LeCun, CVPR 2009]

[Gregor & LeCun 2010]

Learning Invariant Features with L2 Group Sparsity

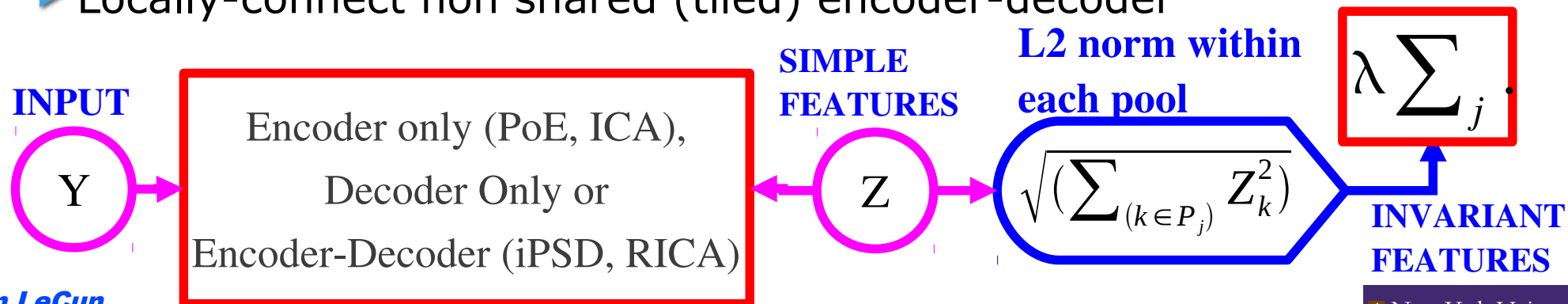
- Unsupervised PSD ignores the spatial pooling step.
- Could we devise a similar method that learns the pooling layer as well?
- Idea [Hyvarinen & Hoyer 2001]: **group sparsity** on pools of features
 - Minimum number of pools must be non-zero
 - Number of features that are on within a pool doesn't matter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

$$E(Y, Z) = \|Y - W_d Z\|^2 + \|Z - g_e(W_e, Y)\|^2 + \sum_j \sqrt{\sum_{k \in P_j} Z_k^2}$$



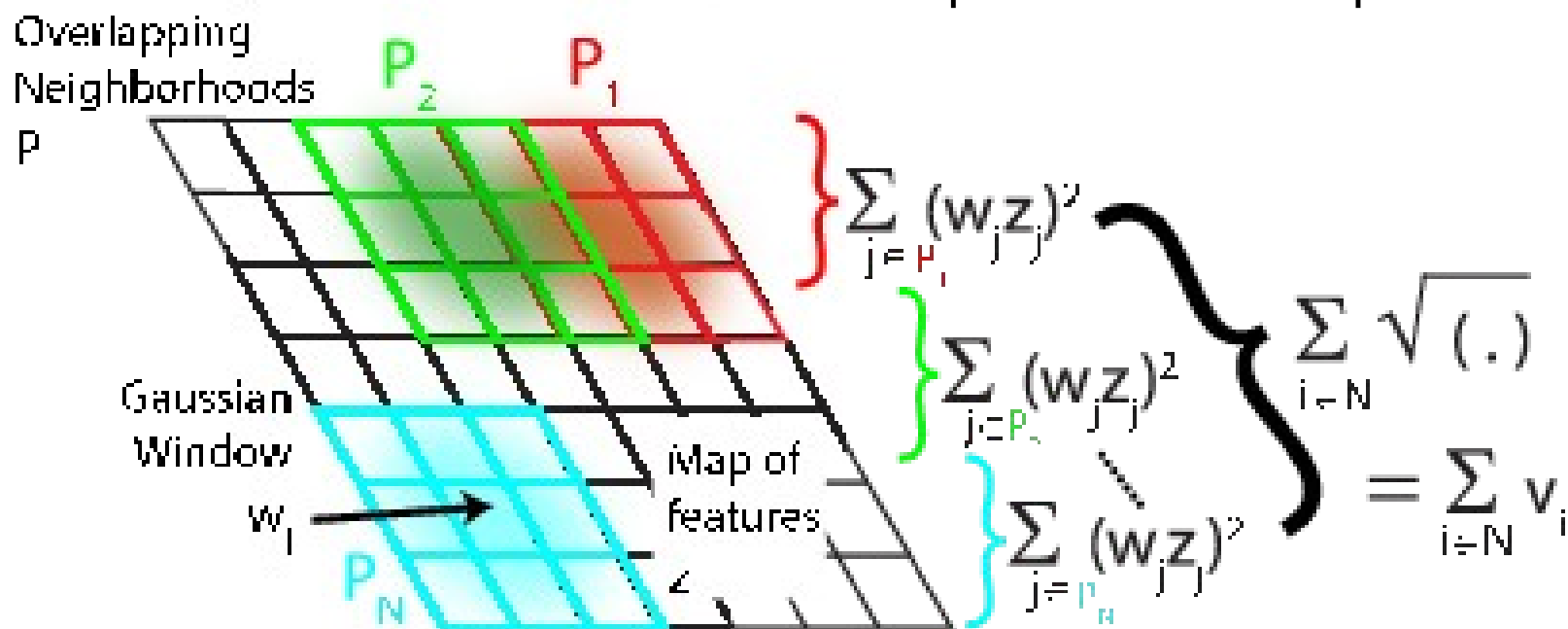
Learning Invariant Features with L2 Group Sparsity

- Idea: features are pooled in group.
 - ▶ Sparsity: sum over groups of L2 norm of activity in group.
- [Hyvärinen Hoyer 2001]: “subspace ICA”
 - ▶ decoder only, square
- [Welling, Hinton, Osindero NIPS 2002]: pooled product of experts
 - ▶ encoder only, overcomplete, log student-T penalty on L2 pooling
- [Kavukcuoglu, Ranzato, Fergus LeCun, CVPR 2010]: Invariant PSD
 - ▶ encoder-decoder (like PSD), overcomplete, L2 pooling
- [Le et al. NIPS 2011]: Reconstruction ICA
 - ▶ Same as [Kavukcuoglu 2010] with linear encoder and tied decoder
- [Gregor & LeCun arXiv:1006:0448, 2010] [Le et al. ICML 2012]
 - ▶ Locally-connect non shared (tiled) encoder-decoder



Pooling Similar Features using Group Sparsity

- A sparse-overcomplete version of Hyvarinen's subspace ICA
- Decoder ensures reconstruction (unlike ICA which requires orthonogonal matrix)
 - ▶ 1. Apply filters on a patch (with suitable non-linearity)
 - ▶ 2. Arrange filter outputs on a 2D plane
 - ▶ 3. square filter outputs
 - ▶ 4. minimize sort of sum of blocks of squared filter outputs



[Kavukcuoglu, Ranzato, Fergus, LeCun, CVPR 2009]

[Jenatton, Obozinski, Bach AISTATS 2010] [Le et al. NIPS2011]

Groups are local in a 2D Topographic Map

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

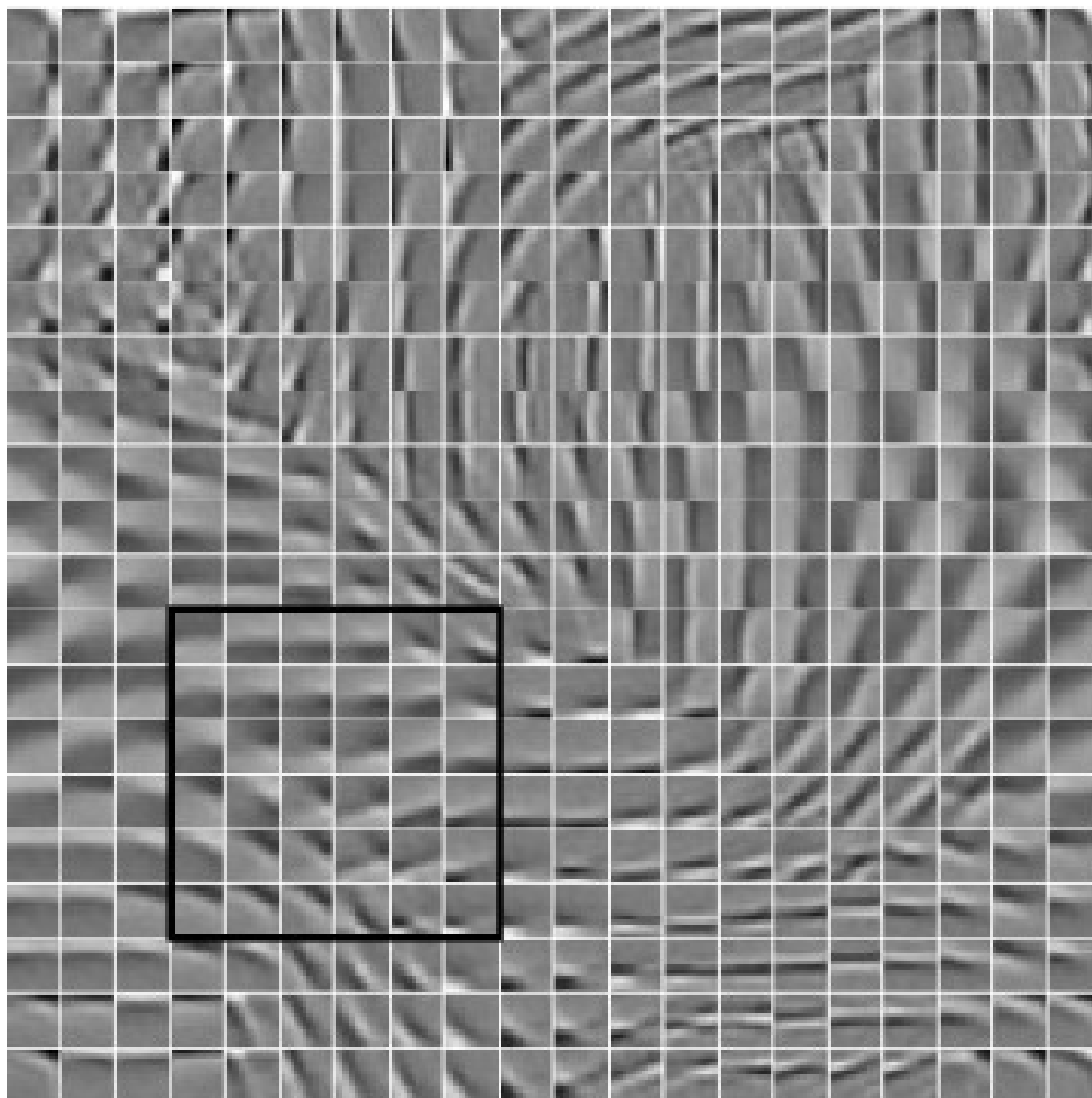


Image-level training, local filters but no weight sharing

Training on 115×115 images. Kernels are 15×15 (not shared across space!)

- ▶ [Gregor & LeCun 2010]
- ▶ Local receptive fields
- ▶ No shared weights
- ▶ 4x overcomplete
- ▶ L2 pooling
- ▶ Group sparsity over pools

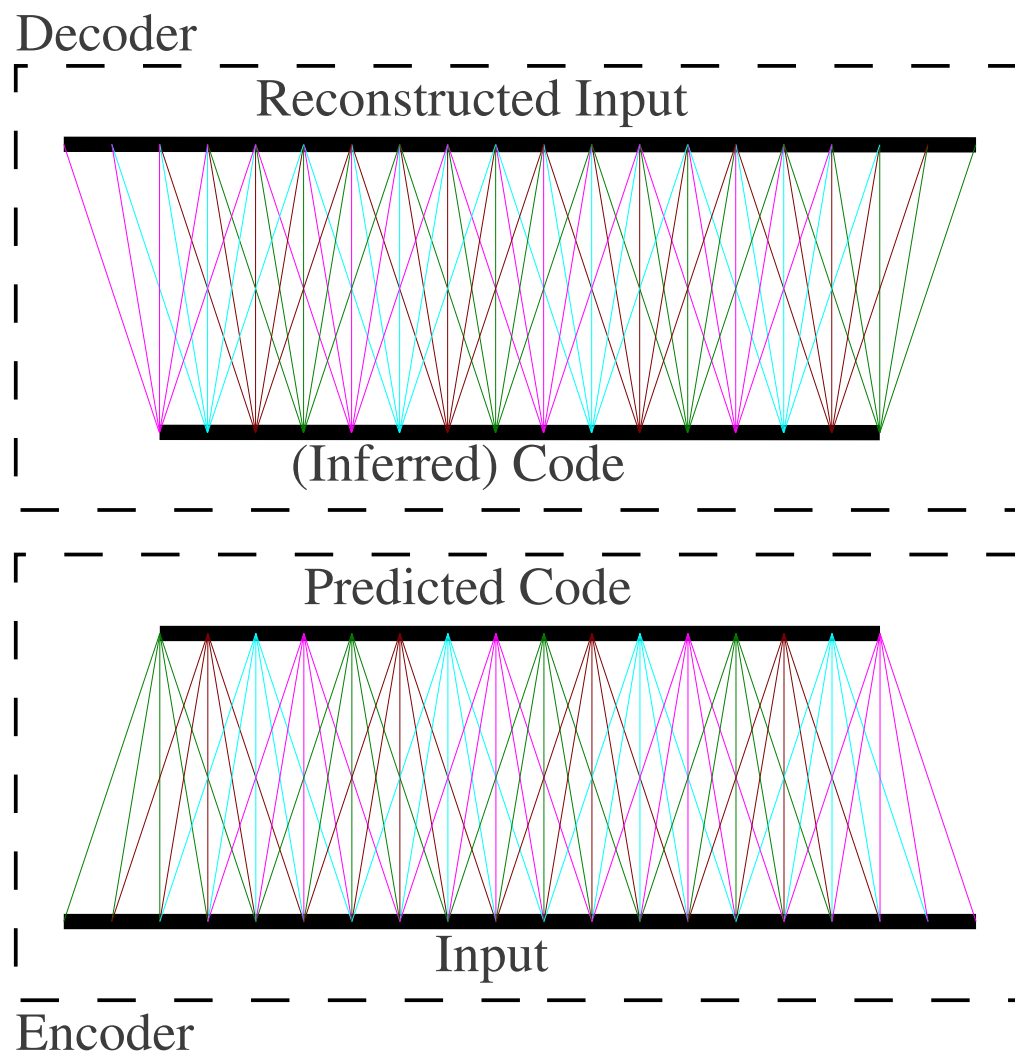


Image-level training, local filters but no weight sharing

- Topographic maps of continuously-varying features
 - Local overlapping pools are invariant complex cells
-
- ▶ [Gregor & LeCun
arXiv:1006.0448]
(double tanh encoder)
 - ▶ [Le et al. ICML'12]
(linear encoder)

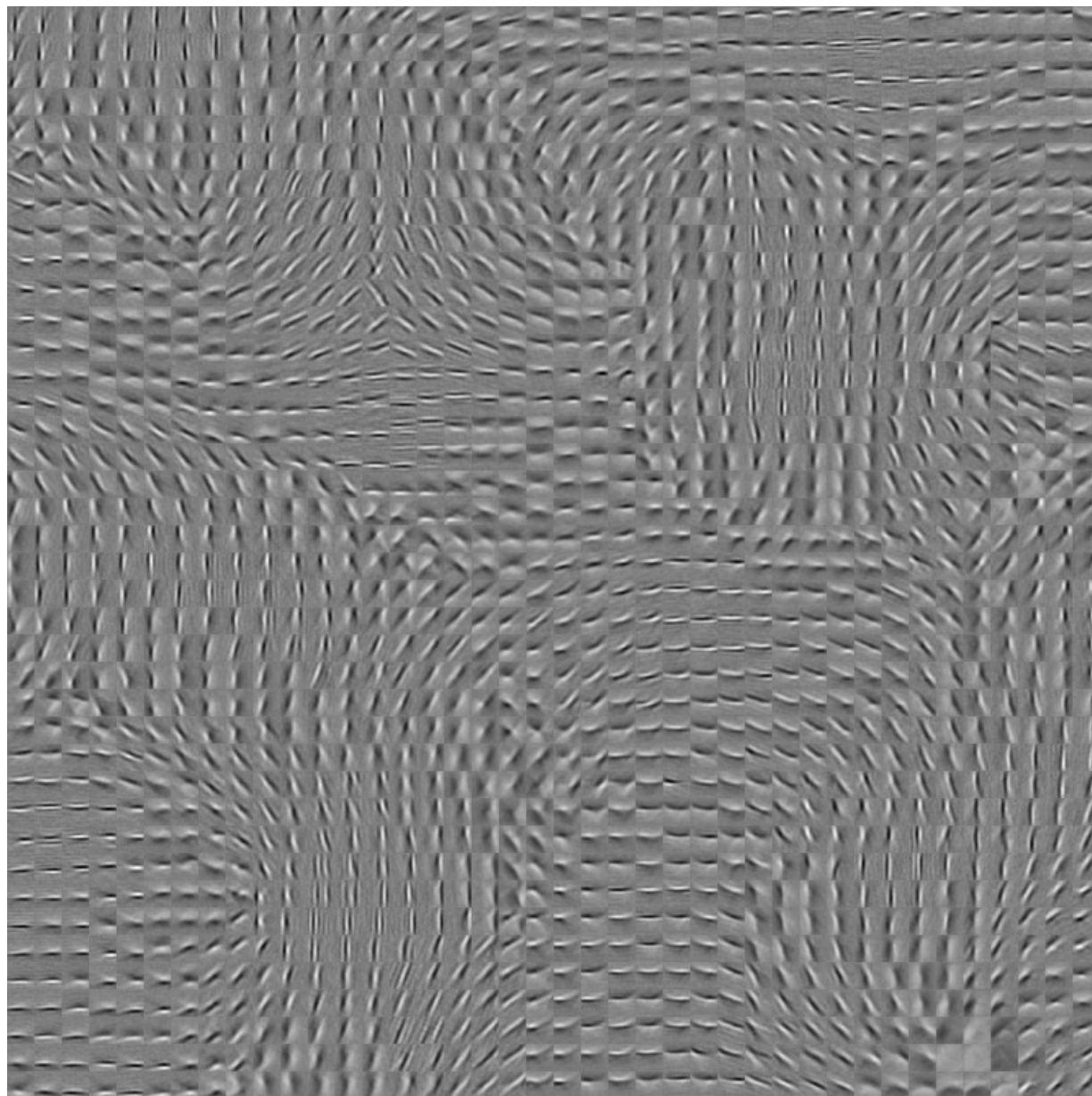
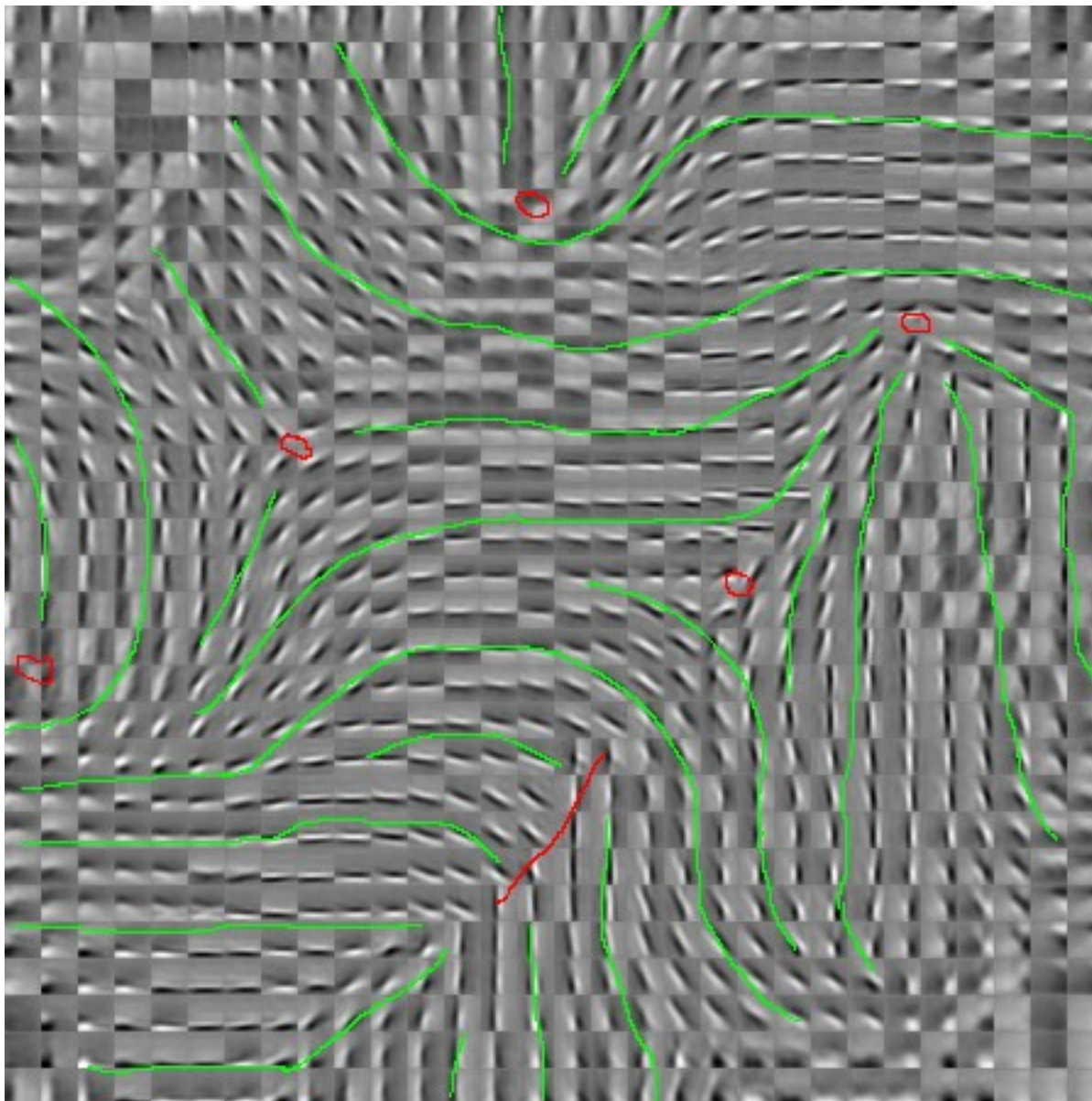
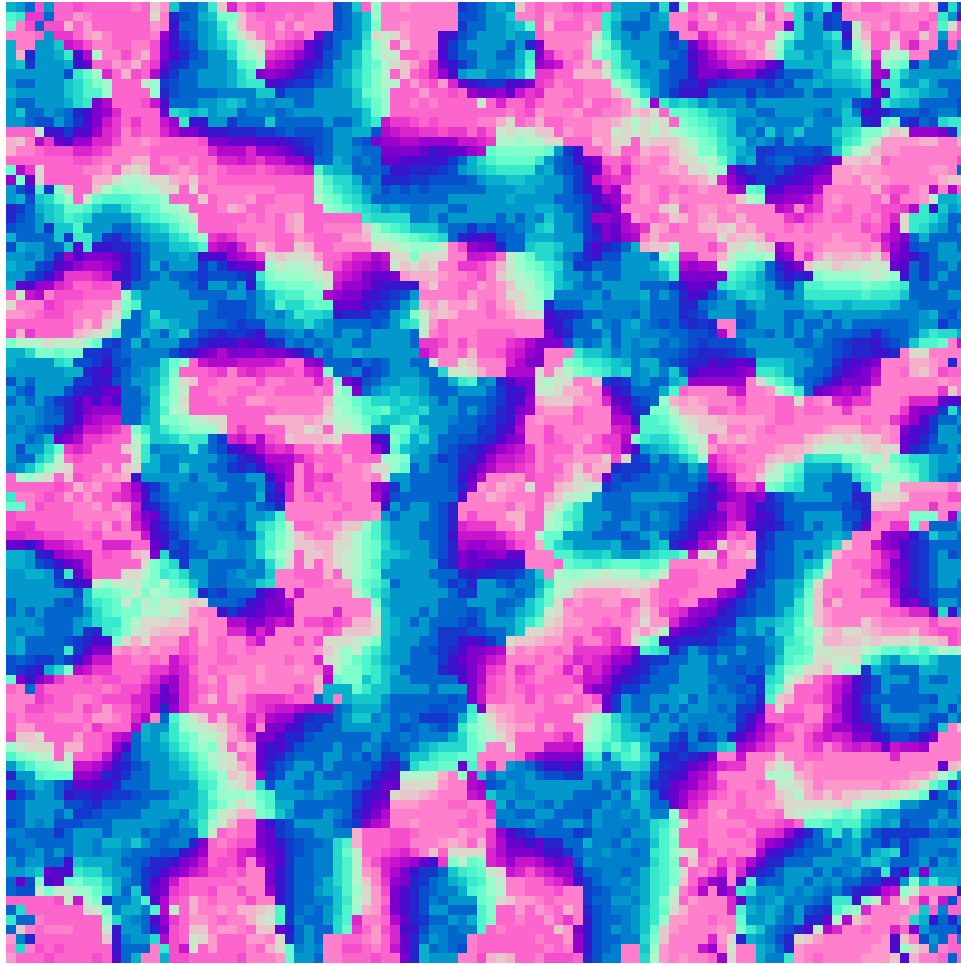


Image-level training, local filters but no weight sharing

- Training on 115×115 images. Kernels are 15×15 (not shared across space!)





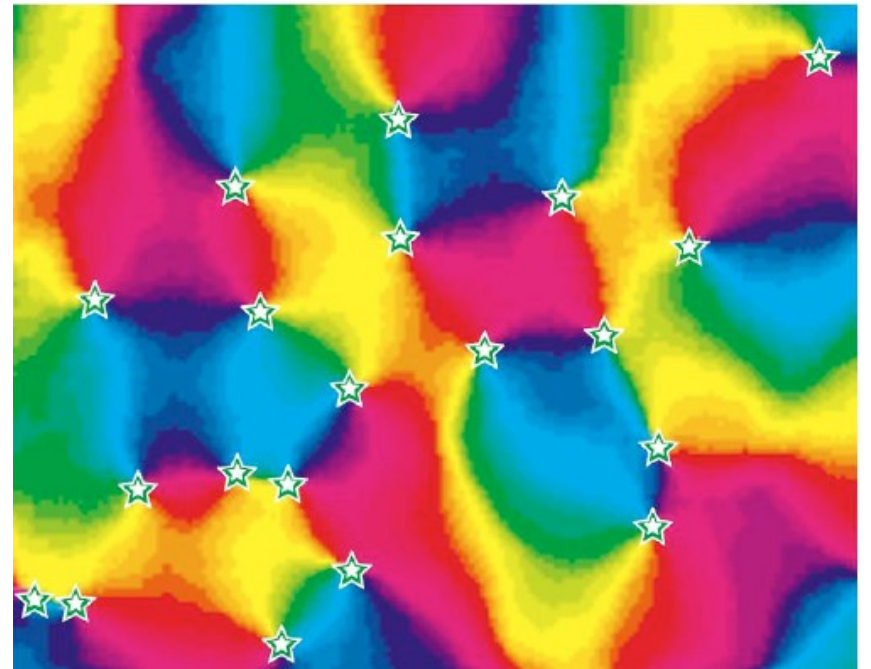
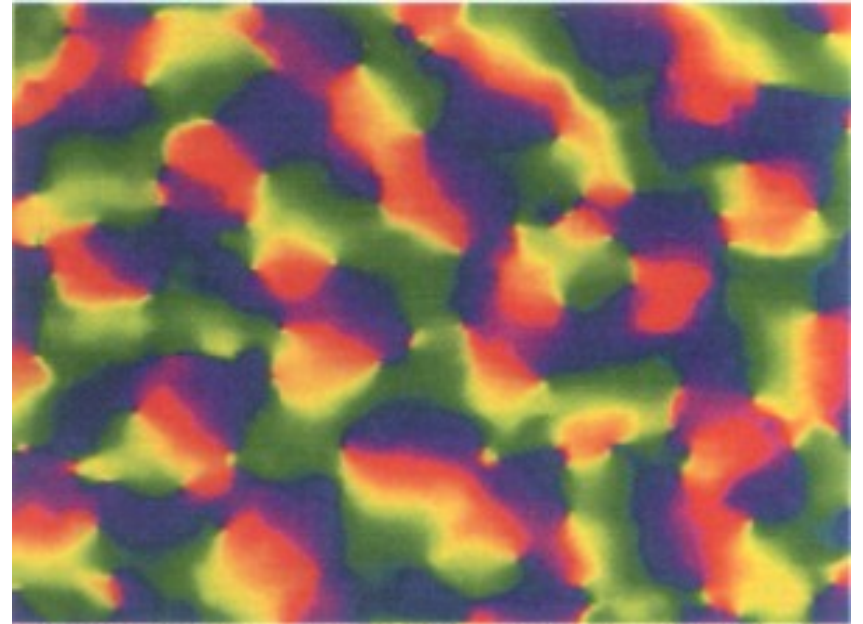
119x119 Image Input

100x100 Code

20x20 Receptive field size

$\sigma=5$

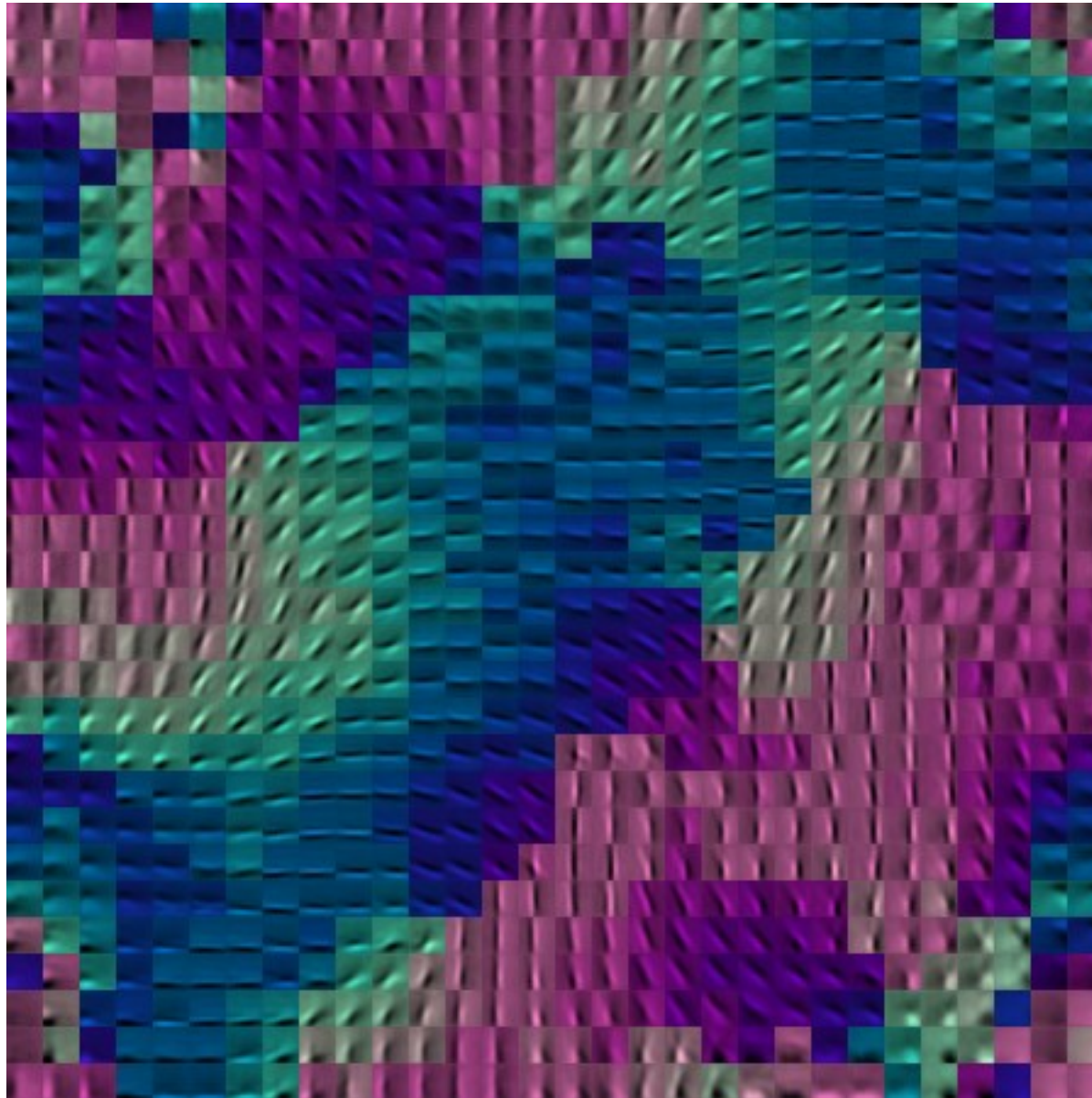
K Obermayer and GG Blasdel, Journal of Neuroscience, Vol 13, 4114-4129 (**Monkey**)



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

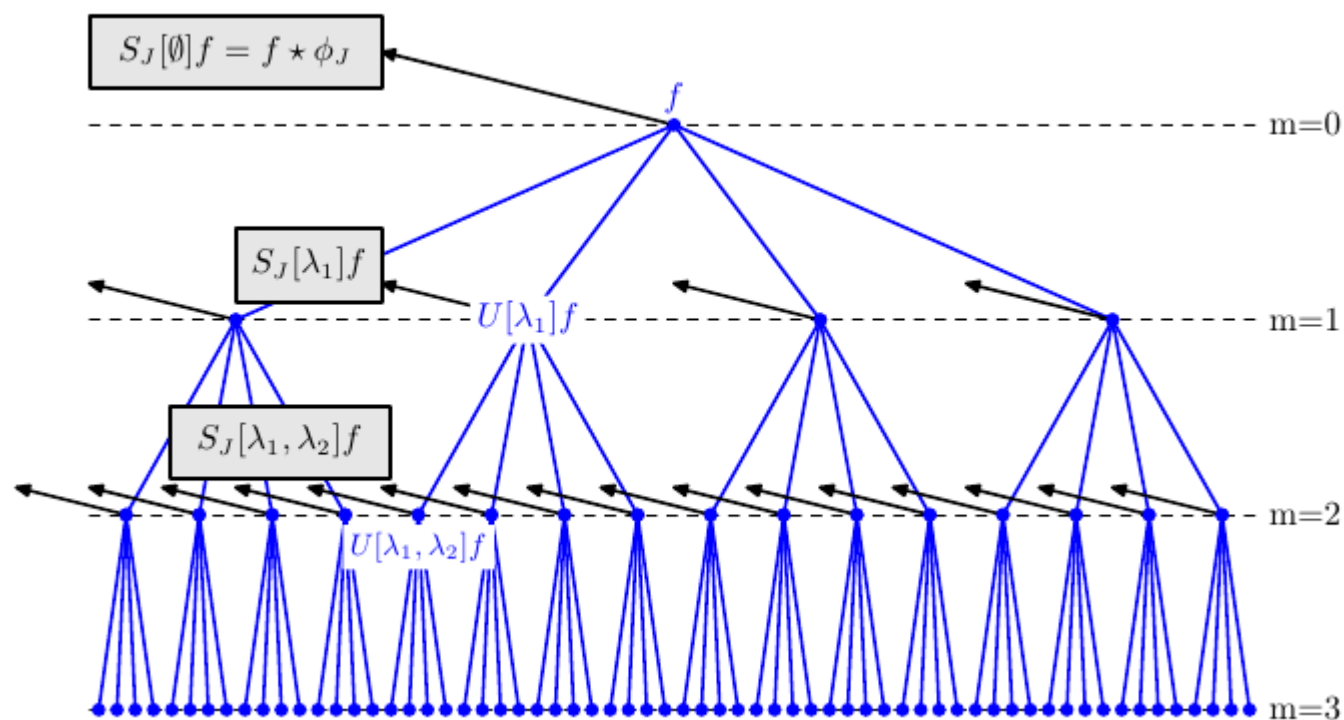
Image-level training, local filters but no weight sharing

- Color indicates orientation (by fitting Gabors)



Theory of Repeated [Filter Bank $\rightarrow$ L2 Pooling $\rightarrow$ Average Pooling]

- Stéphane Mallat's "Scattering Transform": Theory of ConvNet-like architectures
- [Mallat & Bruna CVPR 2011] Classification with Scattering Operators
- [Mallat & Bruna, arXiv:1203.1513 2012] Invariant Scattering Convolution Networks
- [Mallat CPAM 2012] Group Invariant Scattering



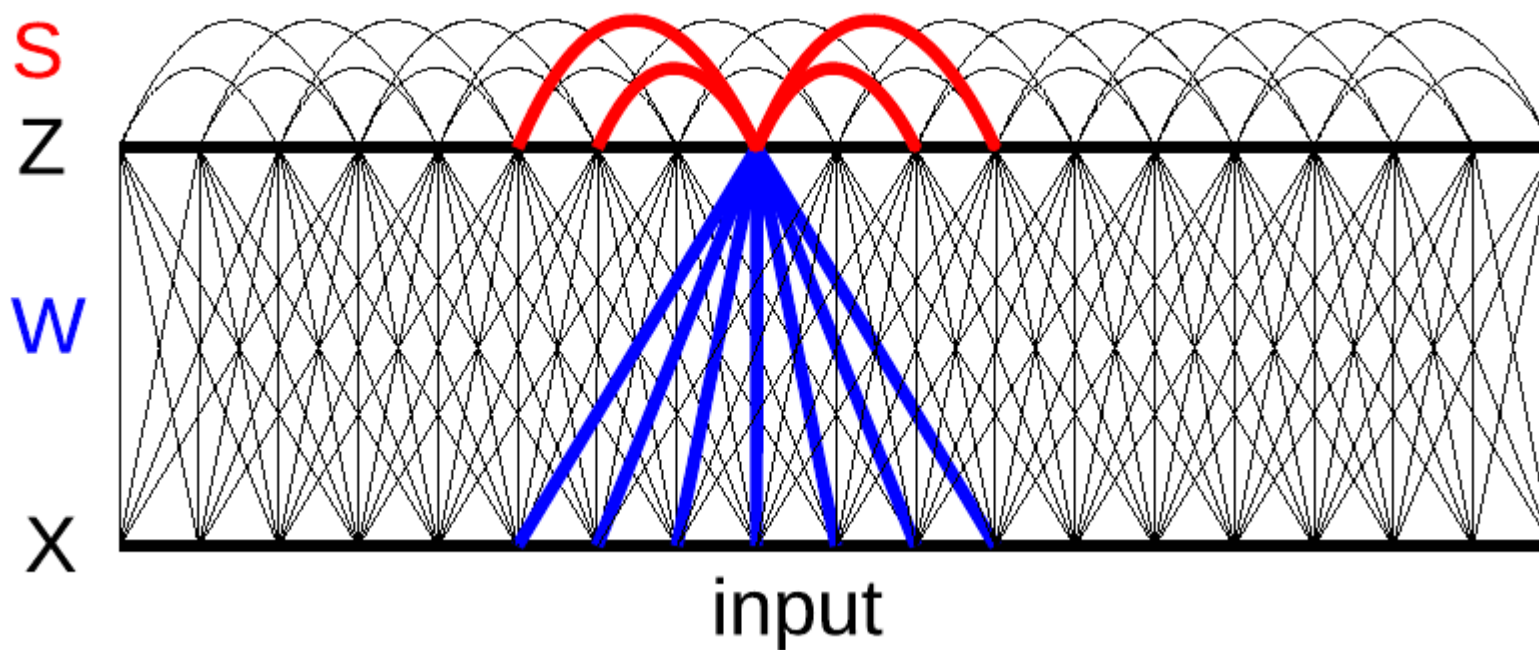
Sparse Coding Using Lateral Inhibition

[Gregor, Szlam, LeCun, NIPS 2011]

Invariant Features Lateral Inhibition

- Replace the L1 sparsity term by a lateral inhibition matrix
- Easy way to impose some structure on the sparsity

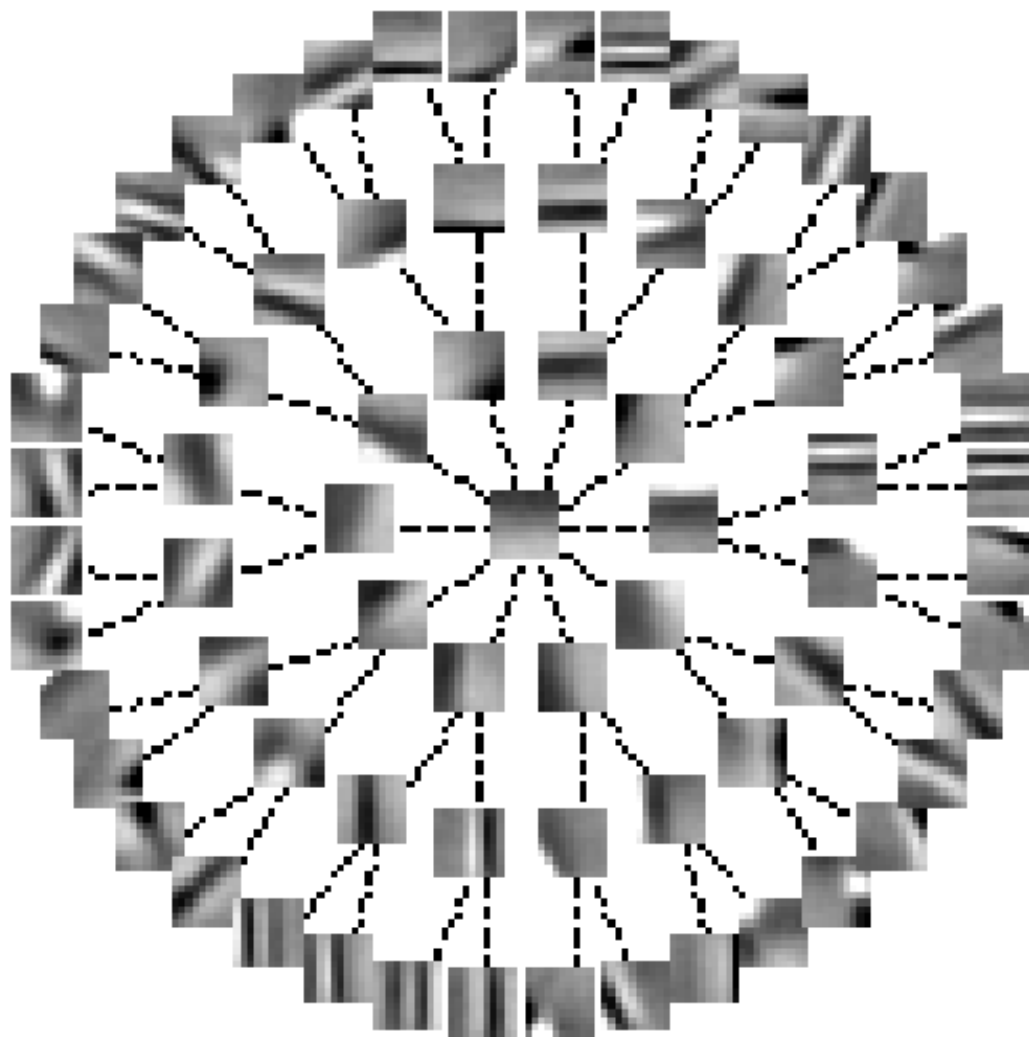
$$\min_{W, Z} \sum_{x \in X} ||Wz - x||^2 + |z|^T S |z|$$

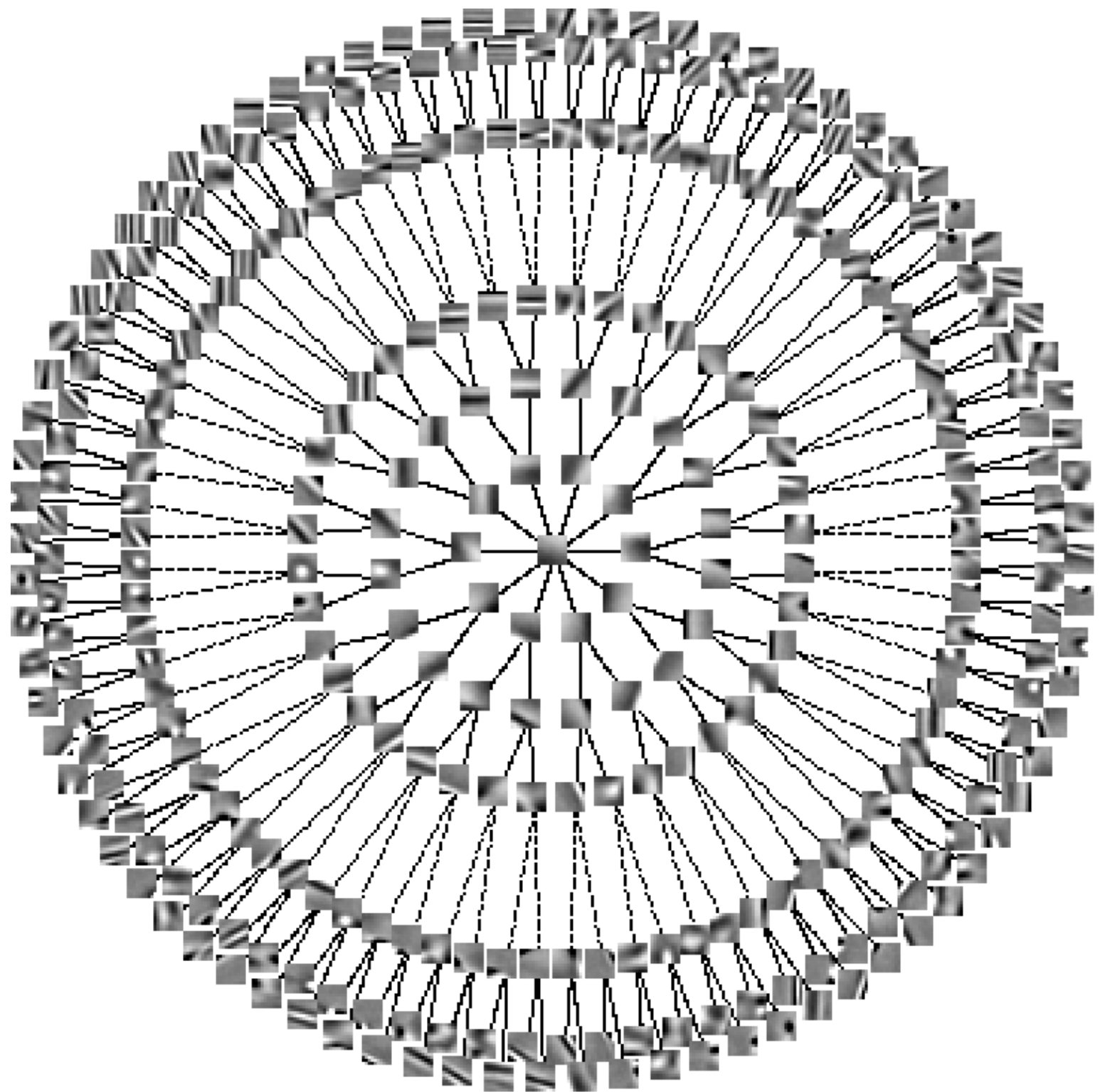


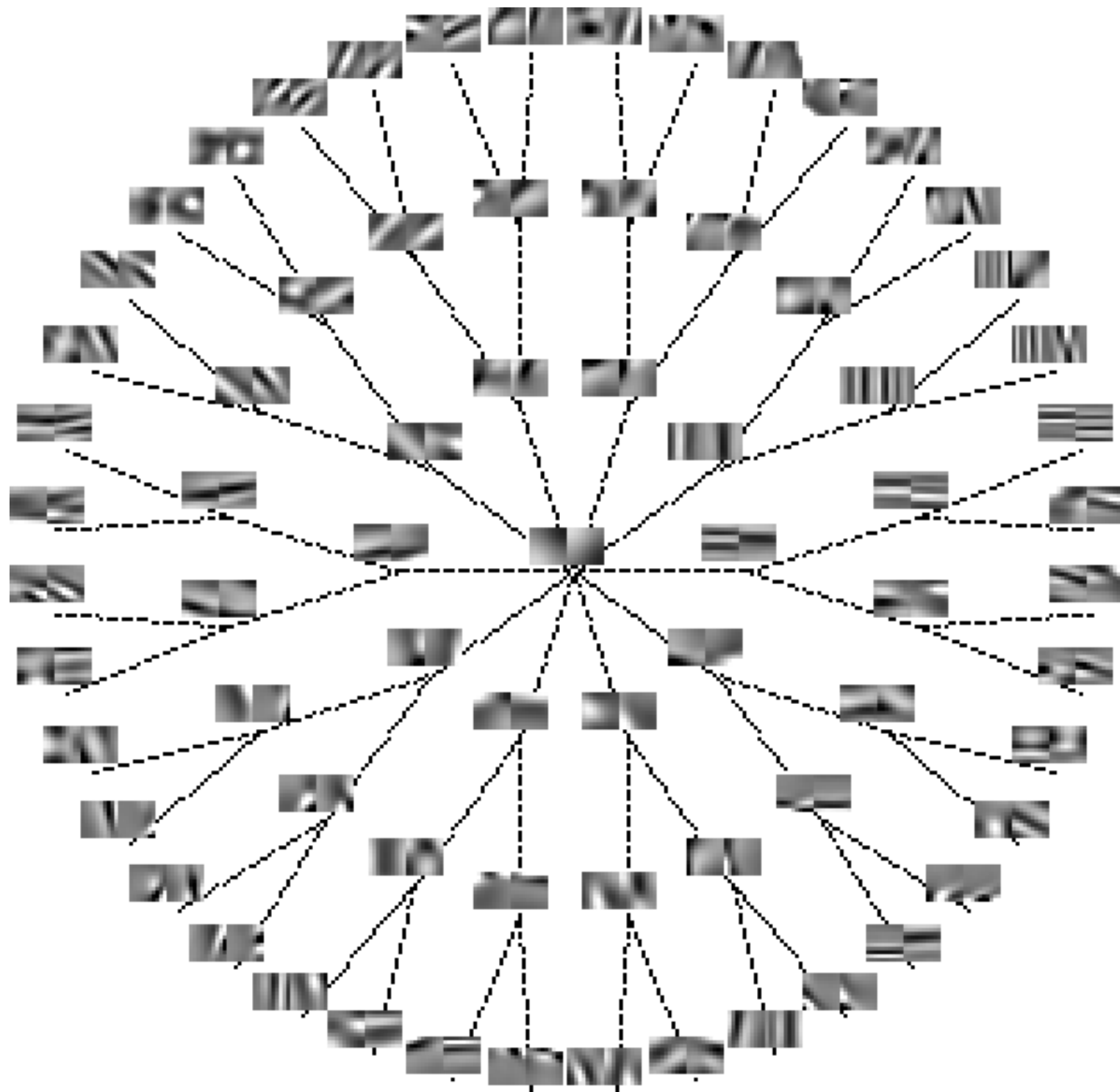
[Gregor, Szlam, LeCun NIPS 2011]

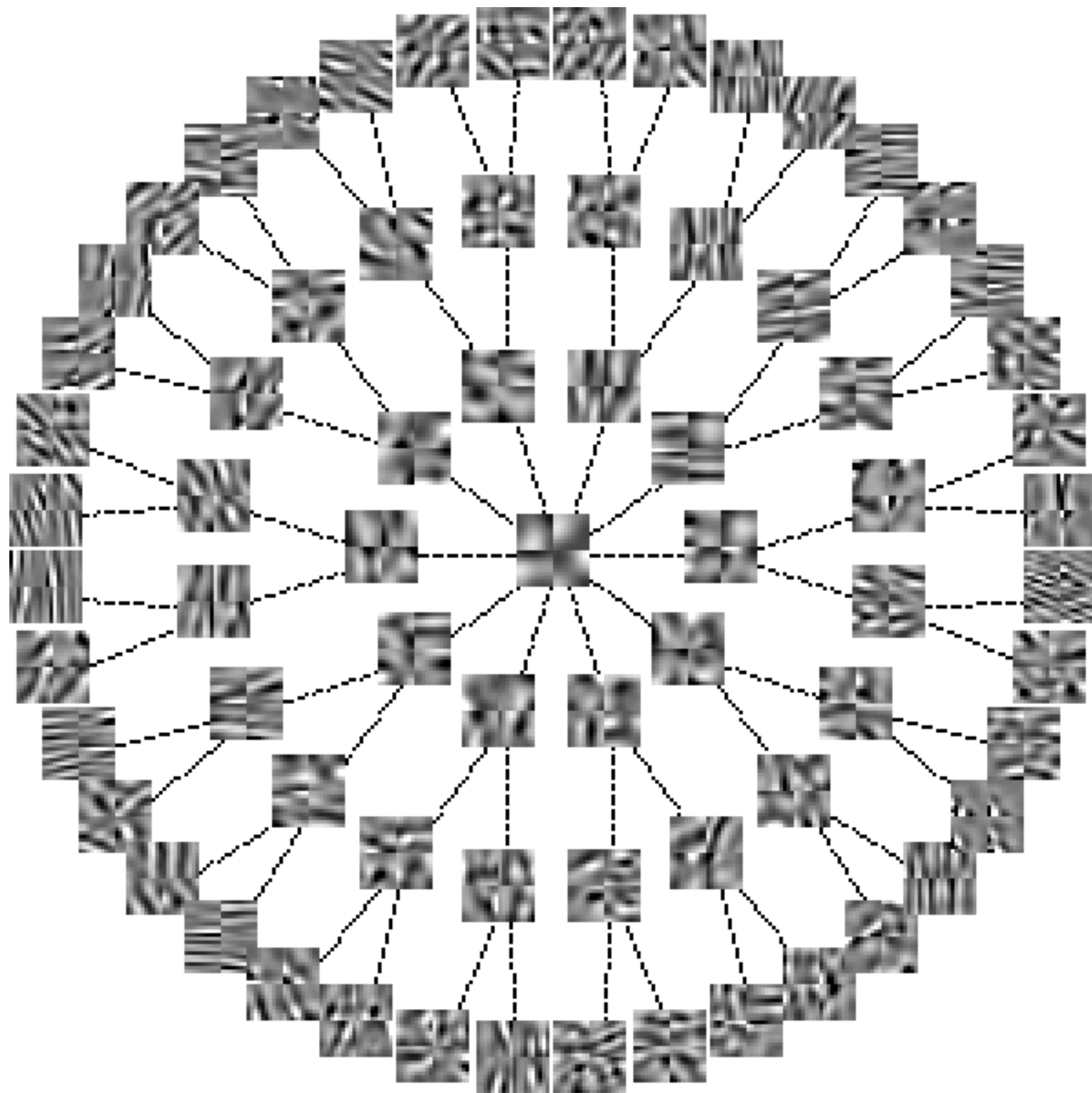
Invariant Features via Lateral Inhibition: Structured Sparsity

- Each edge in the tree indicates a zero in the S matrix (no mutual inhibition)
- S_{ij} is larger if two neurons are far away in the tree



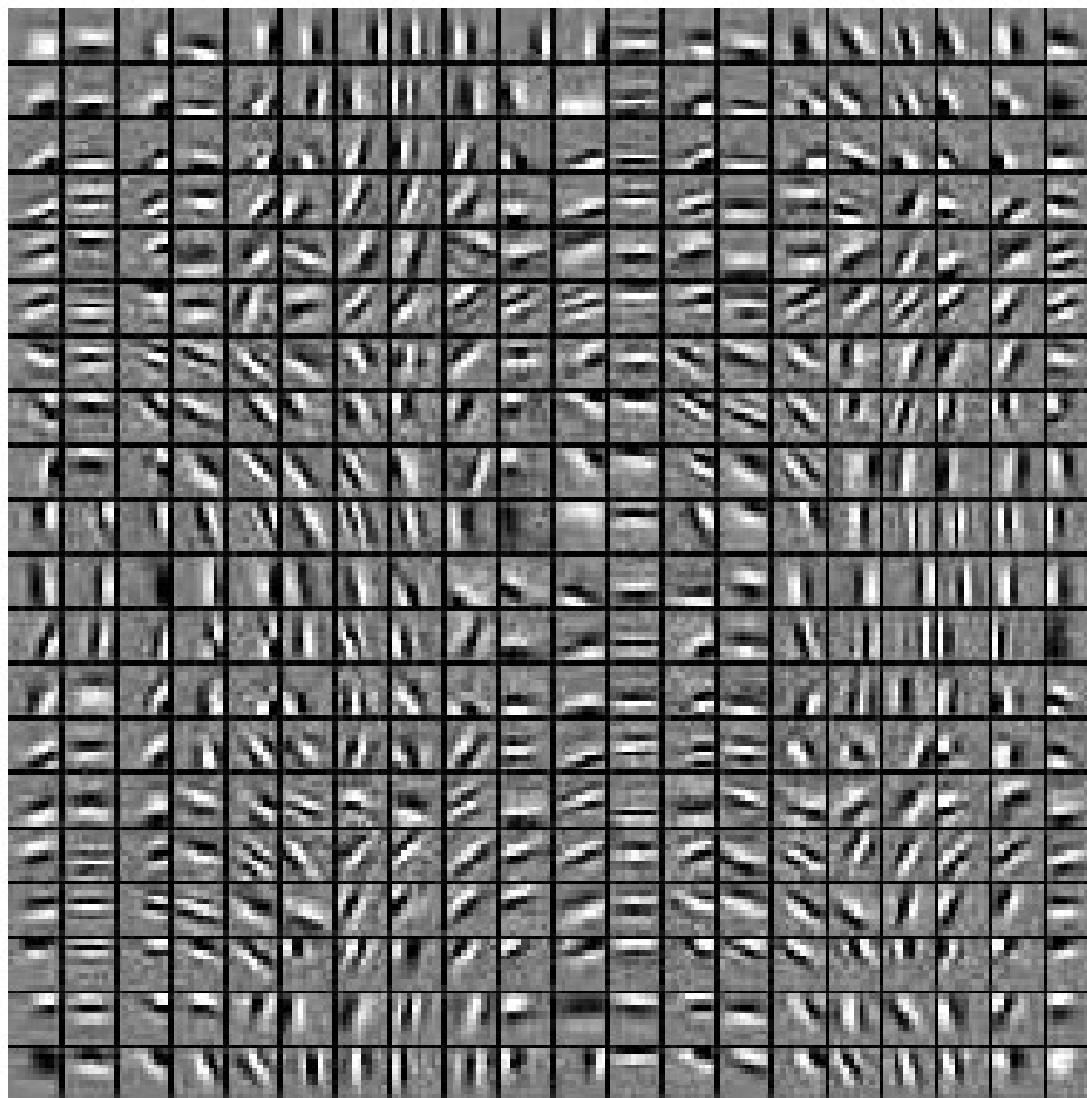






Invariant Features via Lateral Inhibition: Topographic Maps

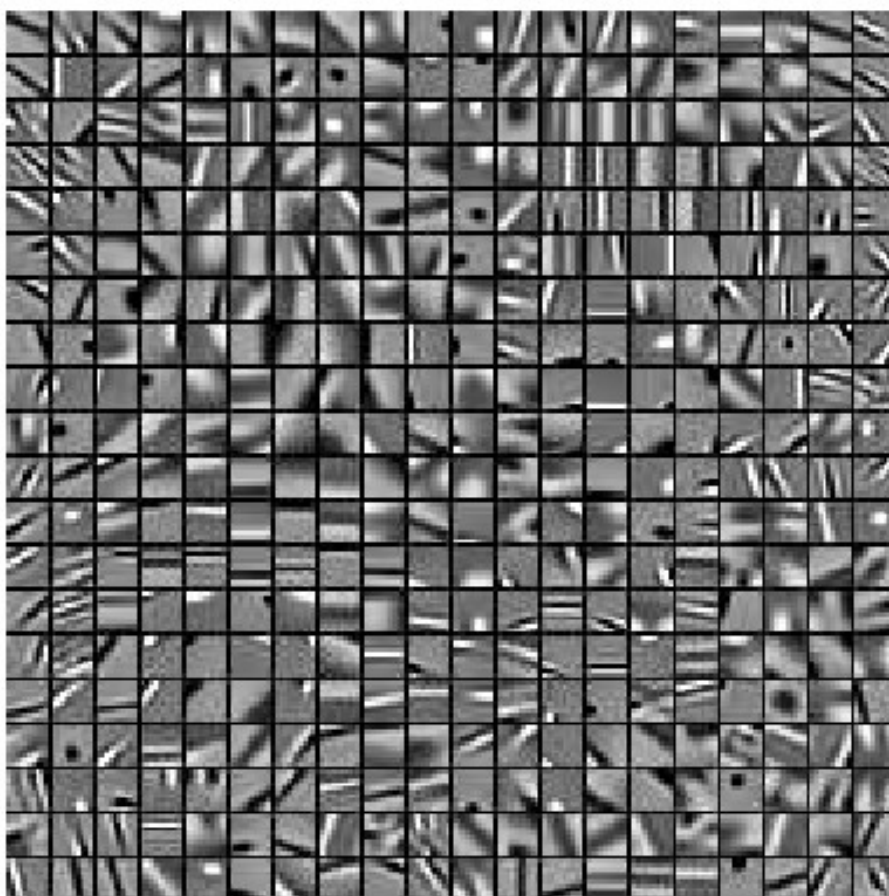
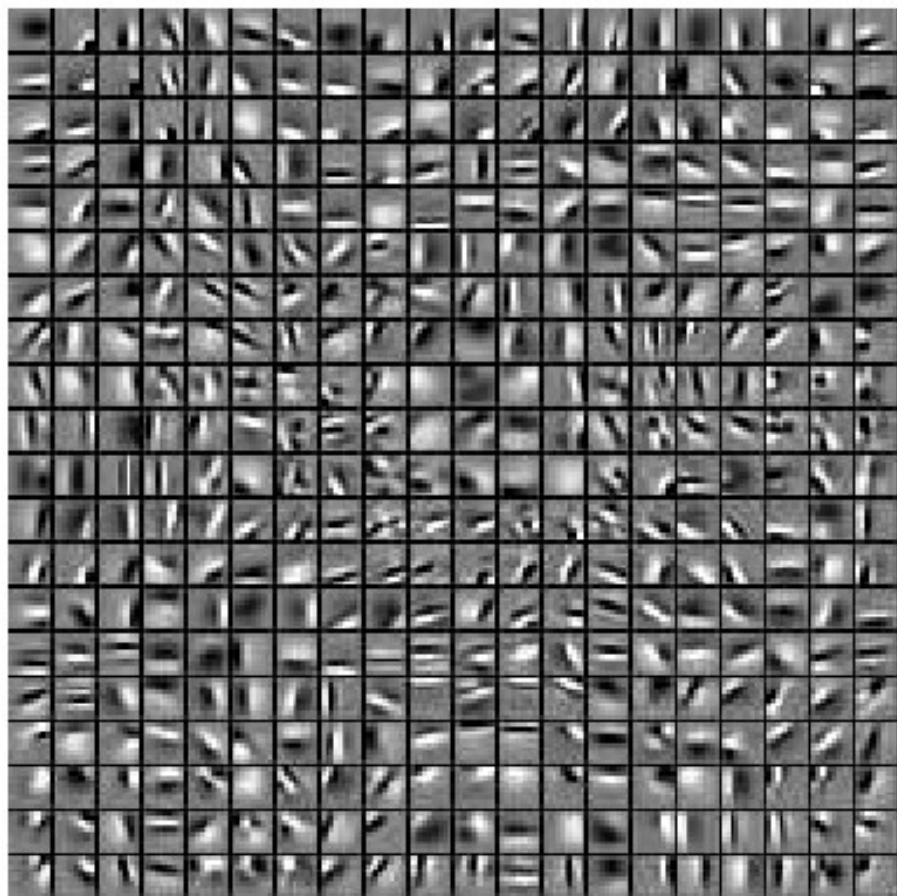
- Non-zero values in S form a ring in a 2D topology
 - ▶ Input patches are high-pass filtered



Invariant Features via Lateral Inhibition: Topographic Maps

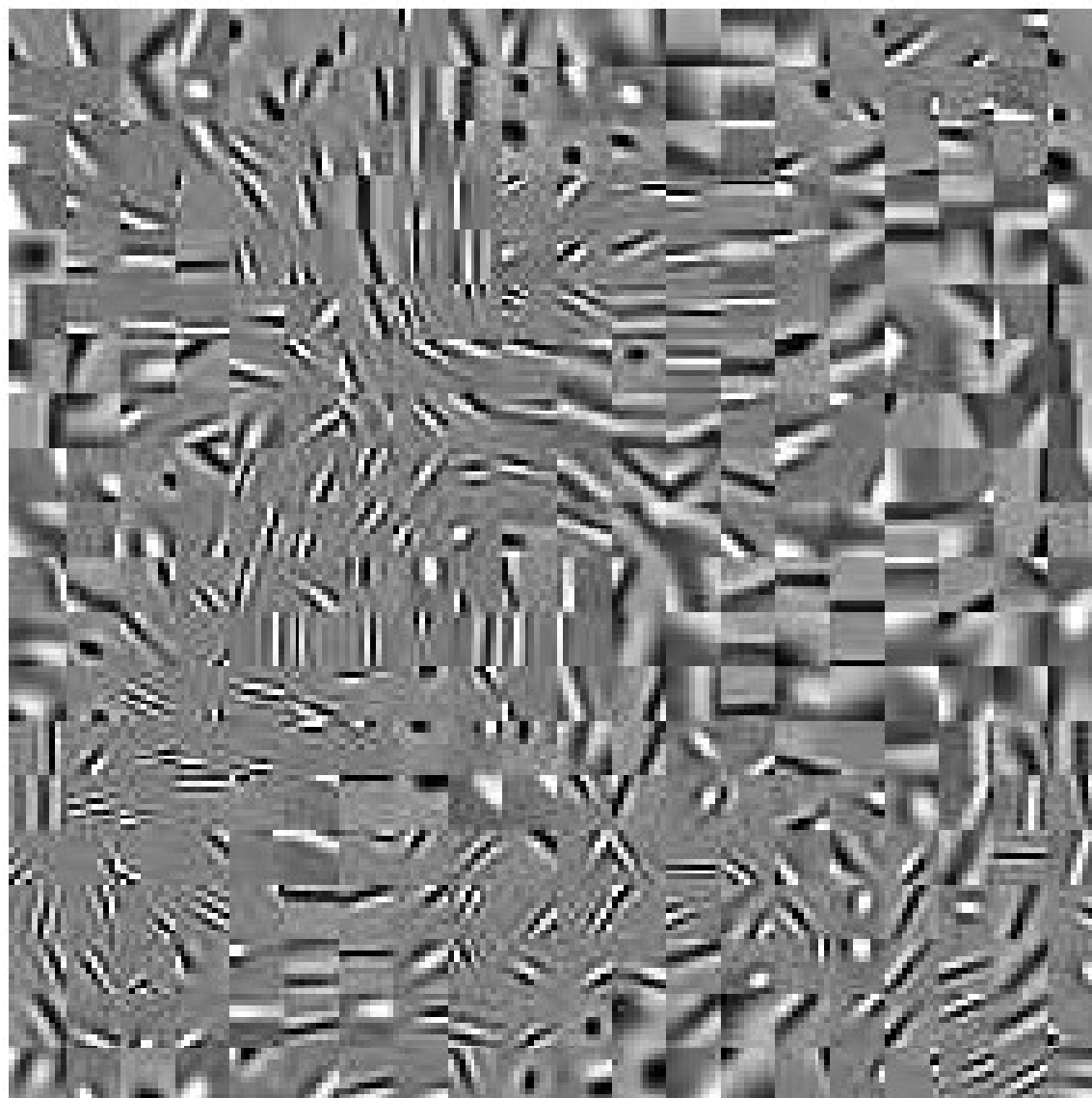
• Non-zero values in S form a ring in a 2D topology

- ▶ Left: no high-pass filtering of input
- ▶ Right: patch-level mean removal



Invariant Features via Lateral Excitation: Topographic Maps

- Short-range lateral excitation + L1 sparsity

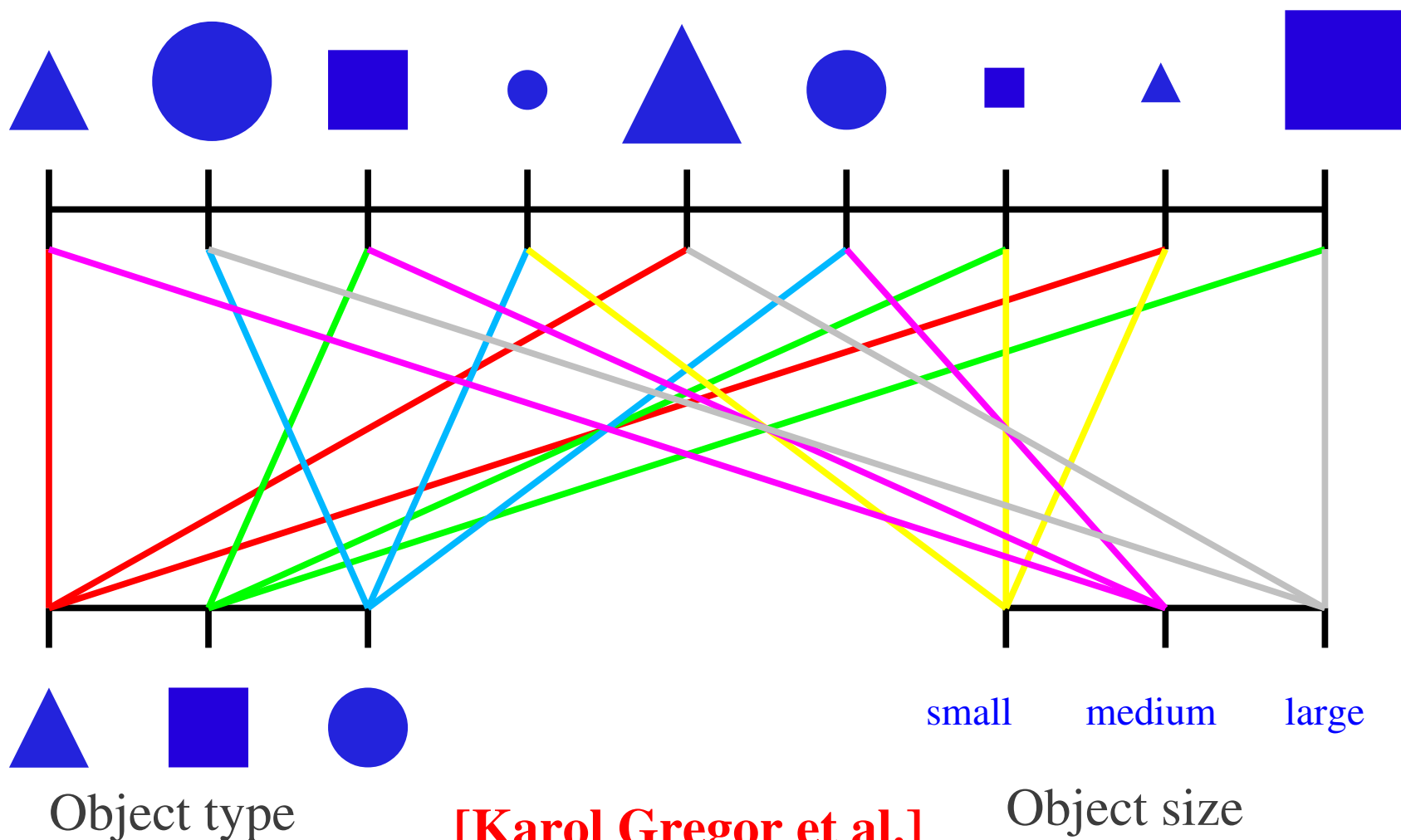


Learning What/Where Features with Temporal Constancy

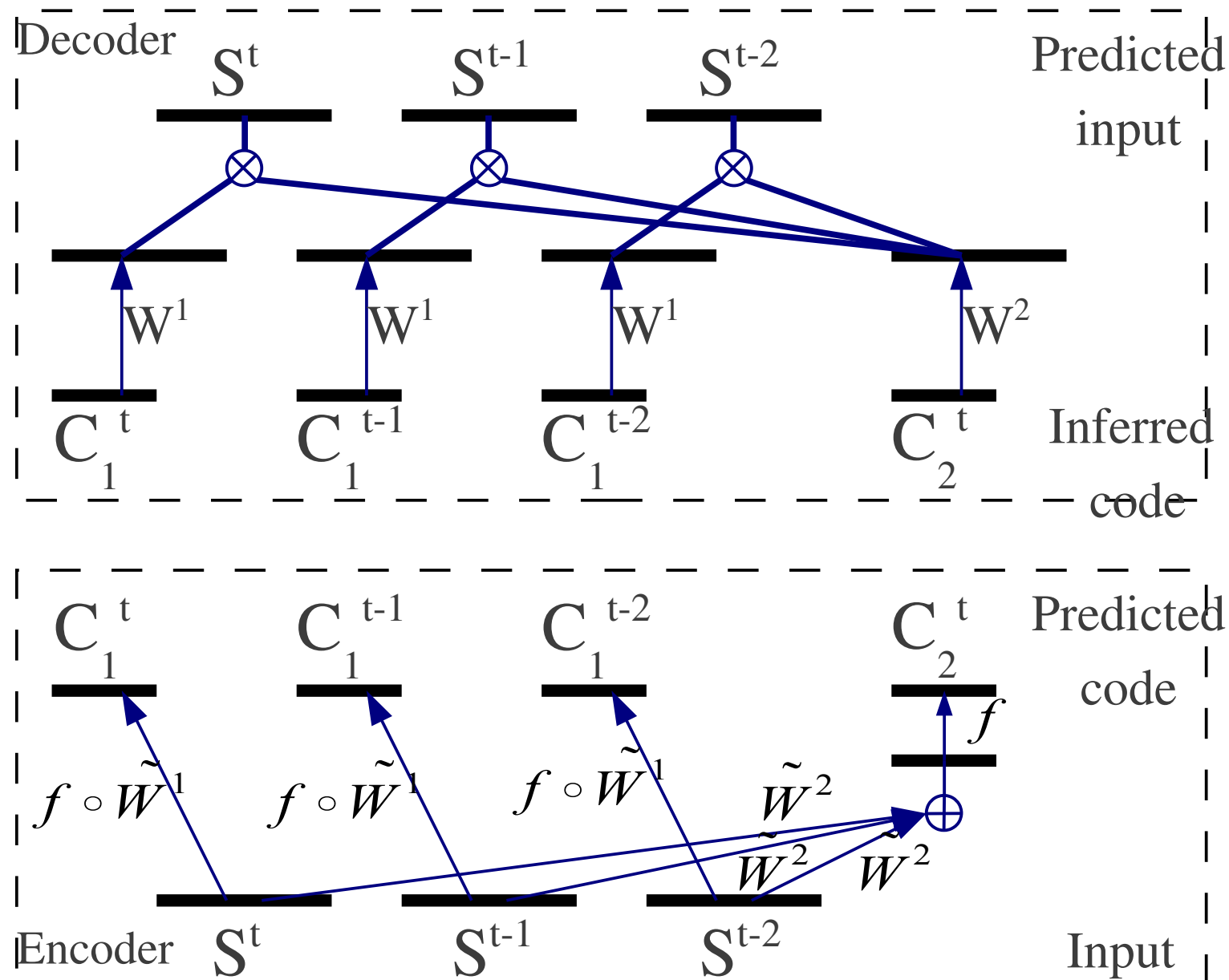
[Gregor & LeCun arXiv:1006.0448, 2010]

Invariant Features through Temporal Constancy

- Object is **cross-product** of object type and instantiation parameters
 - Mapping units [Hinton 1981], capsules [Hinton 2011]

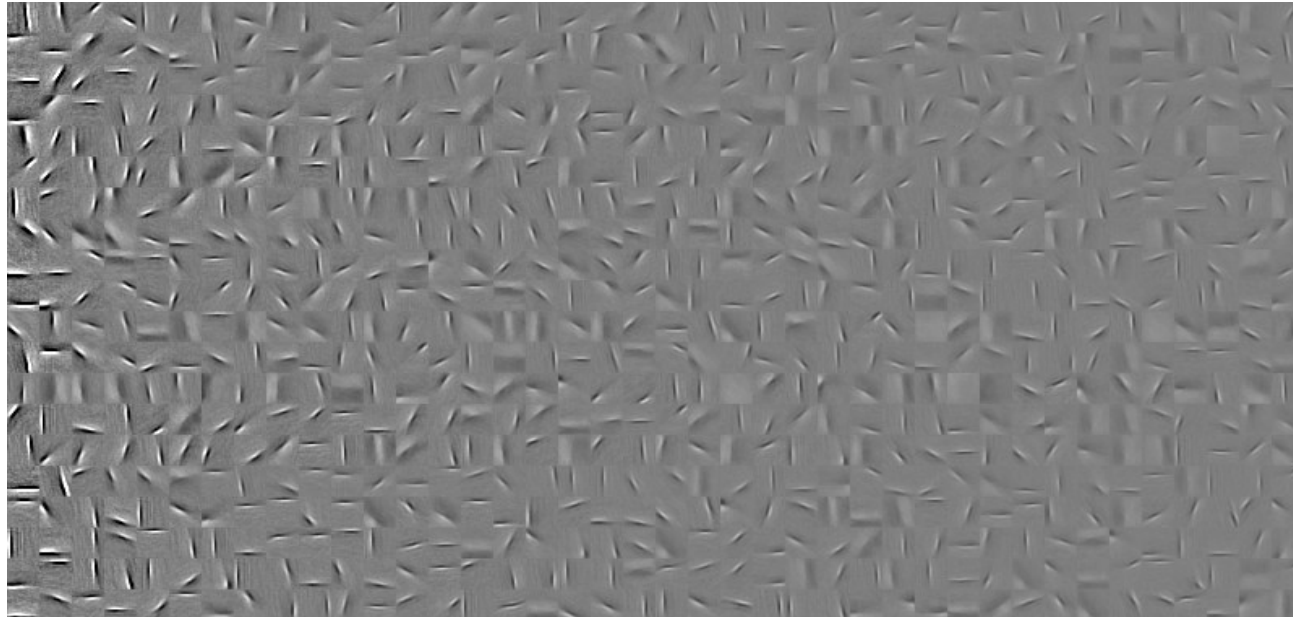


What-Where Auto-Encoder Architecture

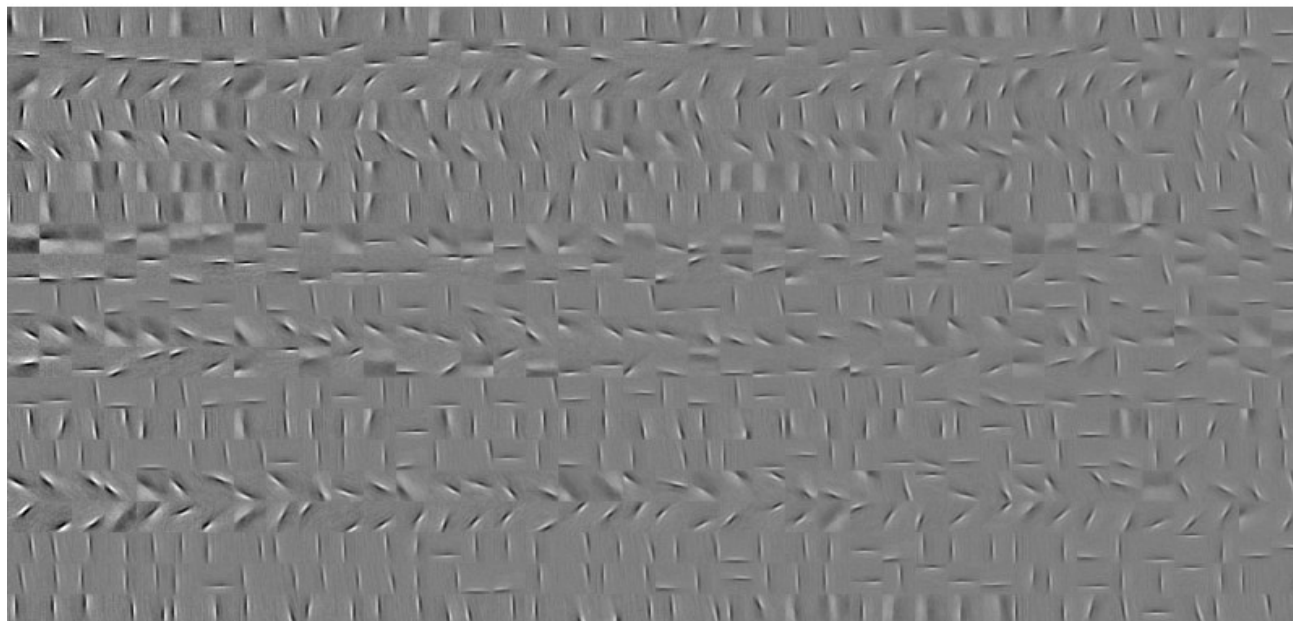


Low-Level Filters Connected to Each Complex Cell

C1
(where)

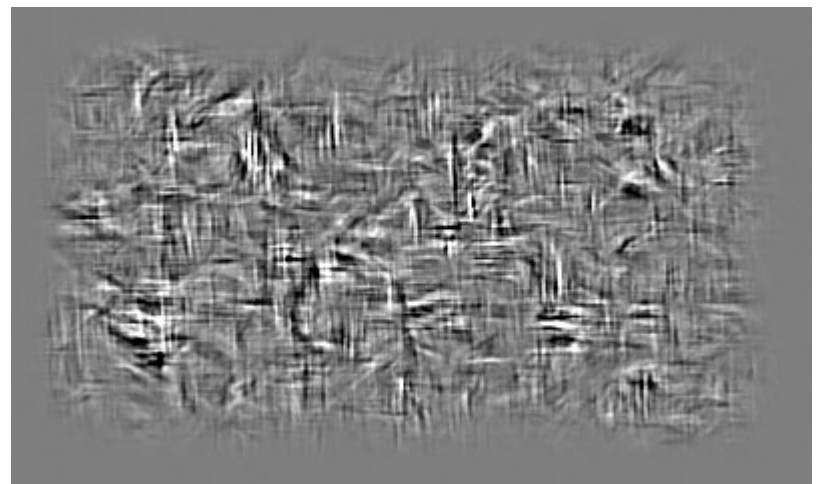
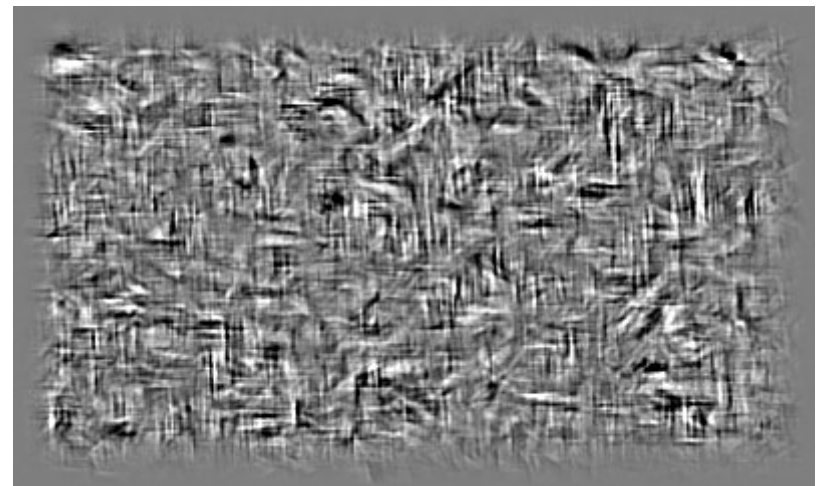
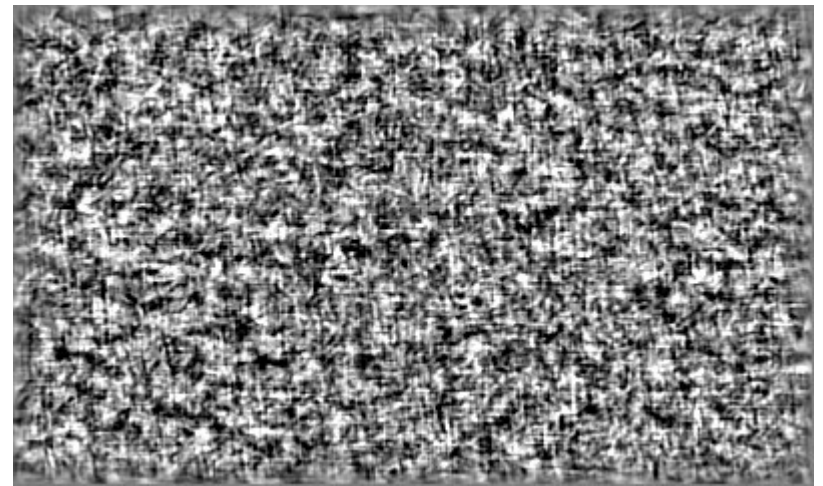
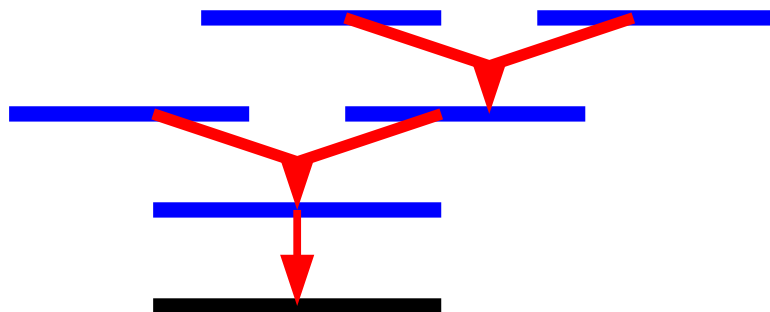
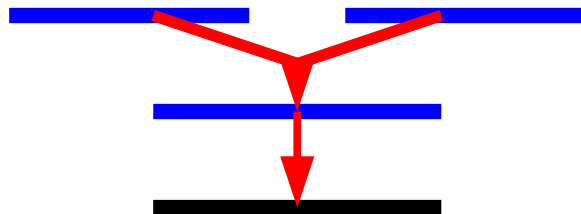
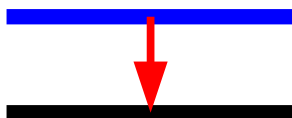


C2
(what)



Generating from the Network

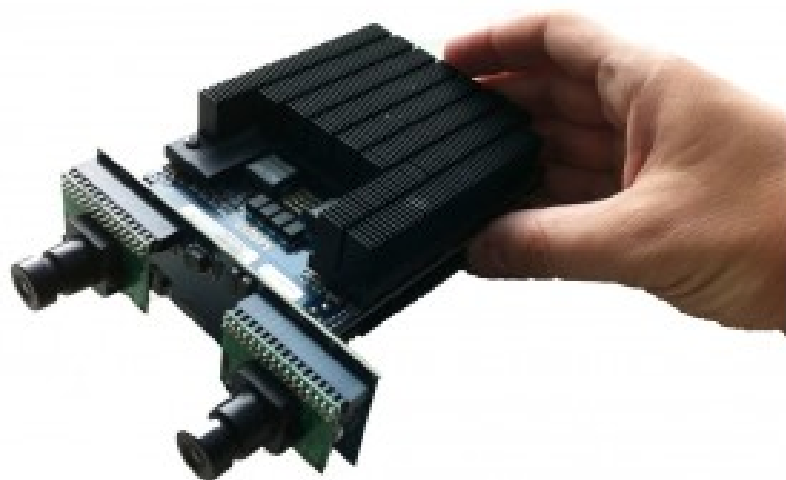
Input



Hardware Implementations

Higher End: FPGA with NeuFlow architecture

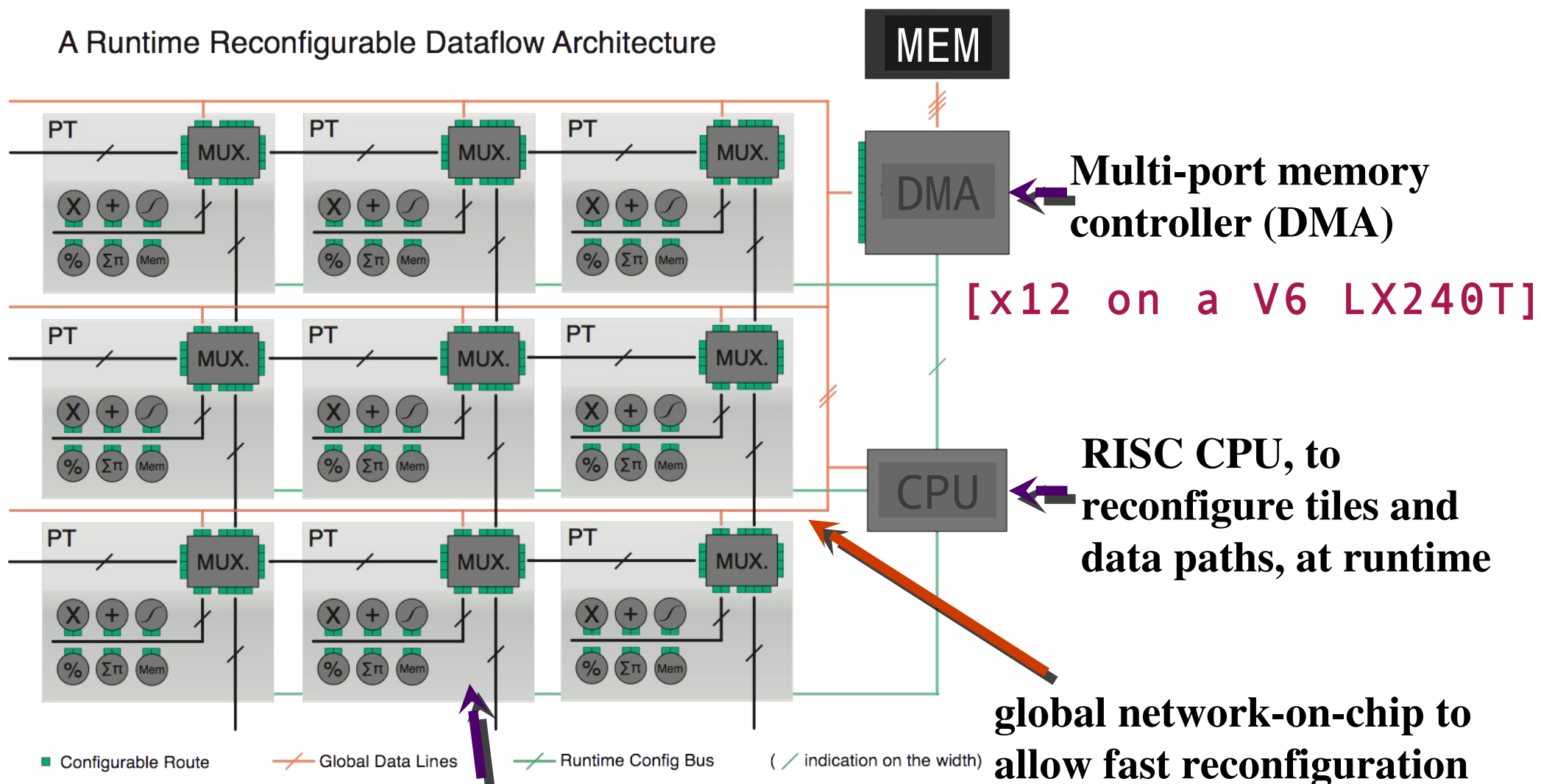
- Now Running on Picocomputing 8x10cm high-performance FPGA board
 - ▶ Virtex 6 LX240T: 680 MAC units, 20 neuflow tiles
- Full scene labeling at 20 frames/sec (50ms/frame) at 320x240



New board with Virtex-6

NewFlow: Architecture

A Runtime Reconfigurable Dataflow Architecture



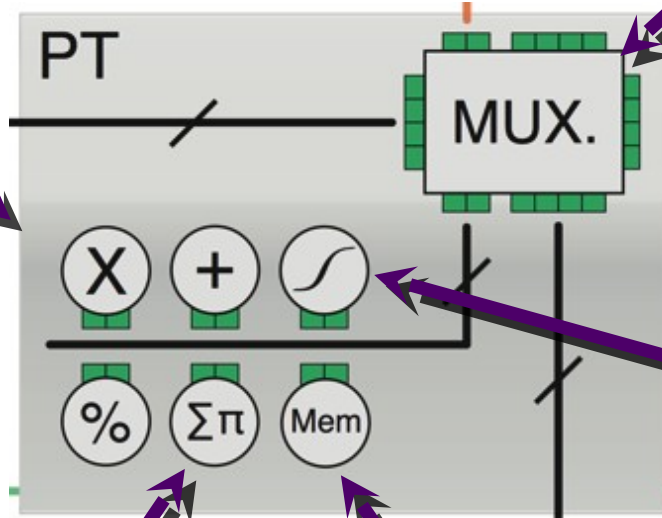
grid of passive processing tiles (PTs)

[x20 on a Virtex6 LX240T]

NewFlow: Processing Tile Architecture

Term-by-term
streaming operators
(MUL,DIV,ADD,SUB,MAX)

[x8, 2 per tile]



configurable router,
to stream data in
and out of the tile, to
neighbors or DMA
ports

[x20]

configurable piece-wise
linear or quadratic
mapper

[x4]

full 1/2D parallel convolver
with 100 MAC units

[x4]

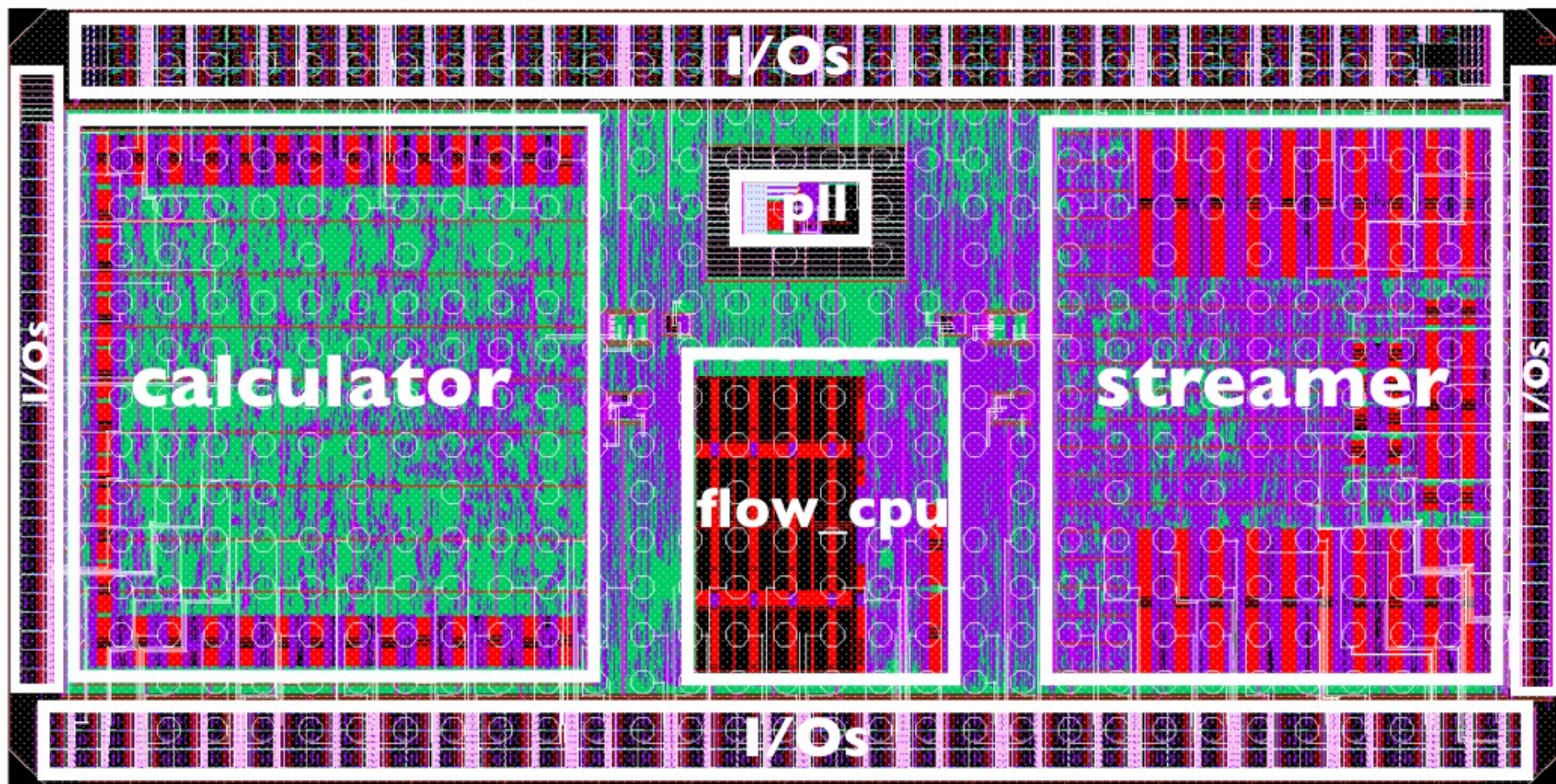
configurable bank of
FIFOs , for stream
buffering, up to 10kB
per PT

[x8]

[Virtex6 LX240T]

NewFlow ASIC: 2.5x5 mm, 45nm, 0.6Watts, 160GOPS

- Collaboration NYU-Purdue (Eugenio Culurciello's group)
- Suitable for vision-enabled embedded and mobile devices
- Status: waiting for first samples end of 2012



NewFlow: Performance

	Intel I7 4 cores	neuFlow Virtex4	neuFlow Virtex 6	nVidia GT335m	neuFlow IBM 45nm	nVidia GTX480
Peak GOP/sec	40	40	160	182	160	1350
Actual GOP/sec	12	37	147	54	147	294
FPS	14	46	182	67	182	374
Power (W)	50	10	10	30	0.6	220
Embed? (GOP/s/W)	0.24	3.7	14.7	1.8	245	1.34

Software Platforms: Torch7

<http://www.torch.ch>

[Collobert, Kavukcuoglu, Farabet 2011]

Torch7

● ML/Vision development environment

- ▶ Collaboration between IDIAP, NYU, and NEC Labs
- ▶ Uses the Lua interpreter/compiler as a front-end
- ▶ Very simple and efficient scripting language
- ▶ Ultra simple and natural syntax
- ▶ It's like Scheme underneath, but with a “normal” syntax.
- ▶ Lua is widely used in the video game and web industries

● Torch7 extend Lua with Numerical, ML, Vision libraries

- ▶ Powerful Vector/Matrix/Tensor Engine (developed by us)
- ▶ Interfaces to numerical libraries, OpenCV, etc
- ▶ Back-ends for SSE, OpenMP, CUDA, ARM-Neon,...
- ▶ Qt-based GUI and graphics
- ▶ Open source: <http://www.torch.ch/>
- ▶ First released in early 2012

Torch7

ML/Vision development environment

- ▶ As simple as Matlab, but faster and better designed as a language.

2D CONVOLUTION

```
x = torch.rand(100,100)
k = torch.rand(10,10)
res1 = torch.conv2(x,k)
```

TRAINING A NEURAL NET

```
mlp = nn.Sequential()
mlp:add( nn.Linear(10, 25) ) -- 10 input, 25 hidden units
mlp:add( nn.Tanh() ) -- some hyperbolic tangent transfer function
mlp:add( nn.Linear(25, 1) ) -- 1 output
criterion = nn.MSECriterion() -- Mean Squared Error criterion
trainer = nn.StochasticGradient(mlp, criterion)
trainer:train(dataset) -- train using some examples
```

MATRIX and VECTOR Operations

```
> M = torch.Tensor(2,2):fill(2)
> N = torch.Tensor(2,4):fill(3)
> x = torch.Tensor(2):fill(4)
> y = torch.Tensor(2):fill(5)
> = x*y -- dot product
40
> = M*x --- matrix-vector
16
16
[torch.Tensor of dimension 2]
```

Acknowledgements

• Y-Lan Boureau



• Kevin Jarrett



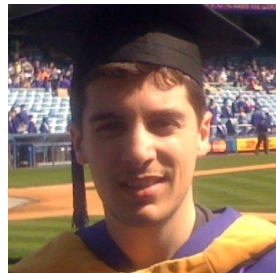
• Koray Kavukcuoglu



• Marc'Aurelio Ranzato



• Pierre Sermanet



• Camille Couprie



• Karol Gregor



• Clément Farabet



• Arthur Szlam



• Rob Fergus



• Laurent Najman (ESIEE)



• Eugenio Culurciello
(Purdue)



The End