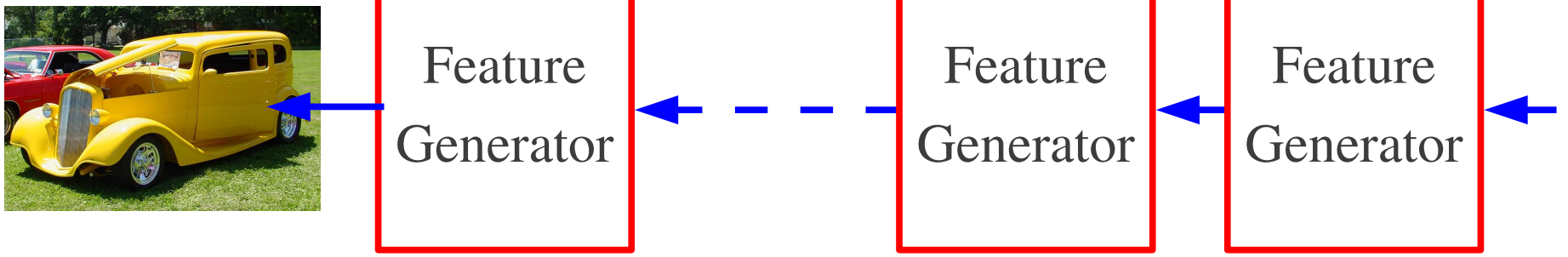
NPB Hype or Hope?

Yann LeCun
Courant Institute of Mathematical Sciences
and
Center for Neural Science,
New York University

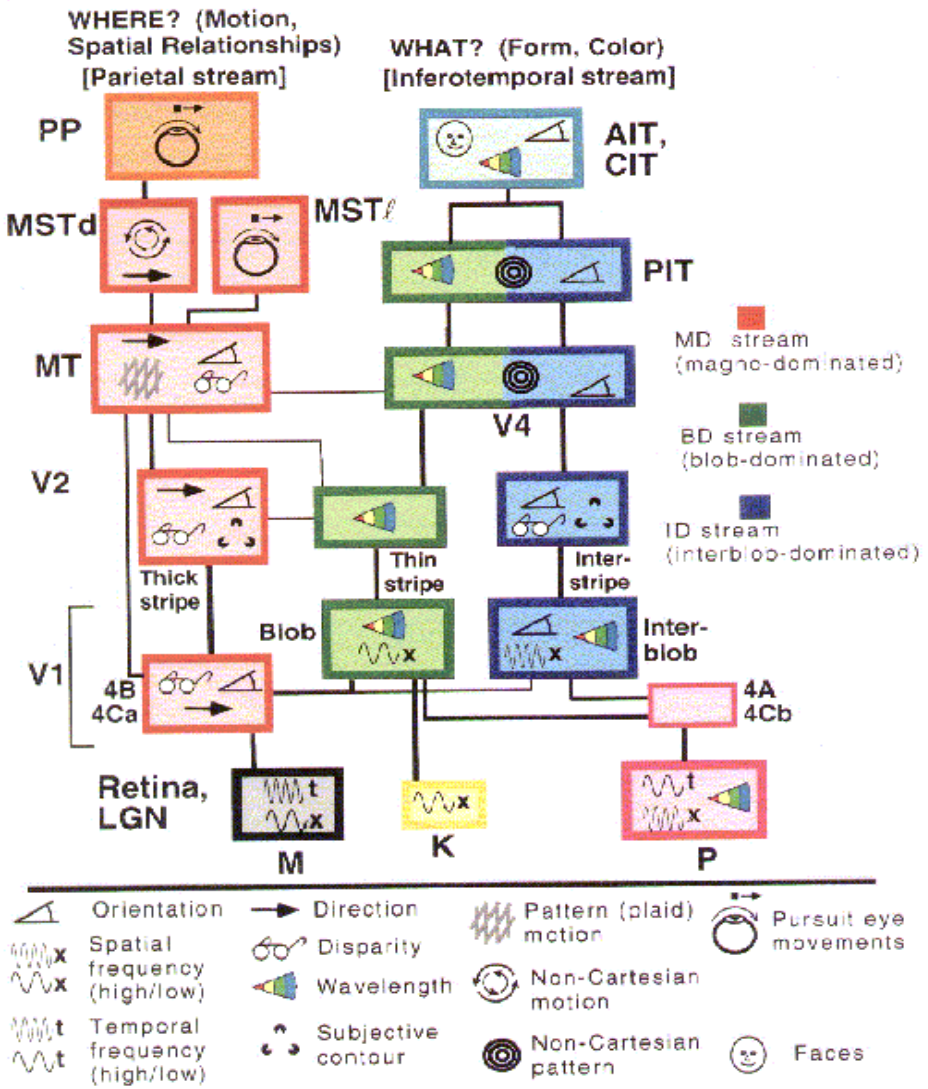
Hierarchical Feed-Back Processing

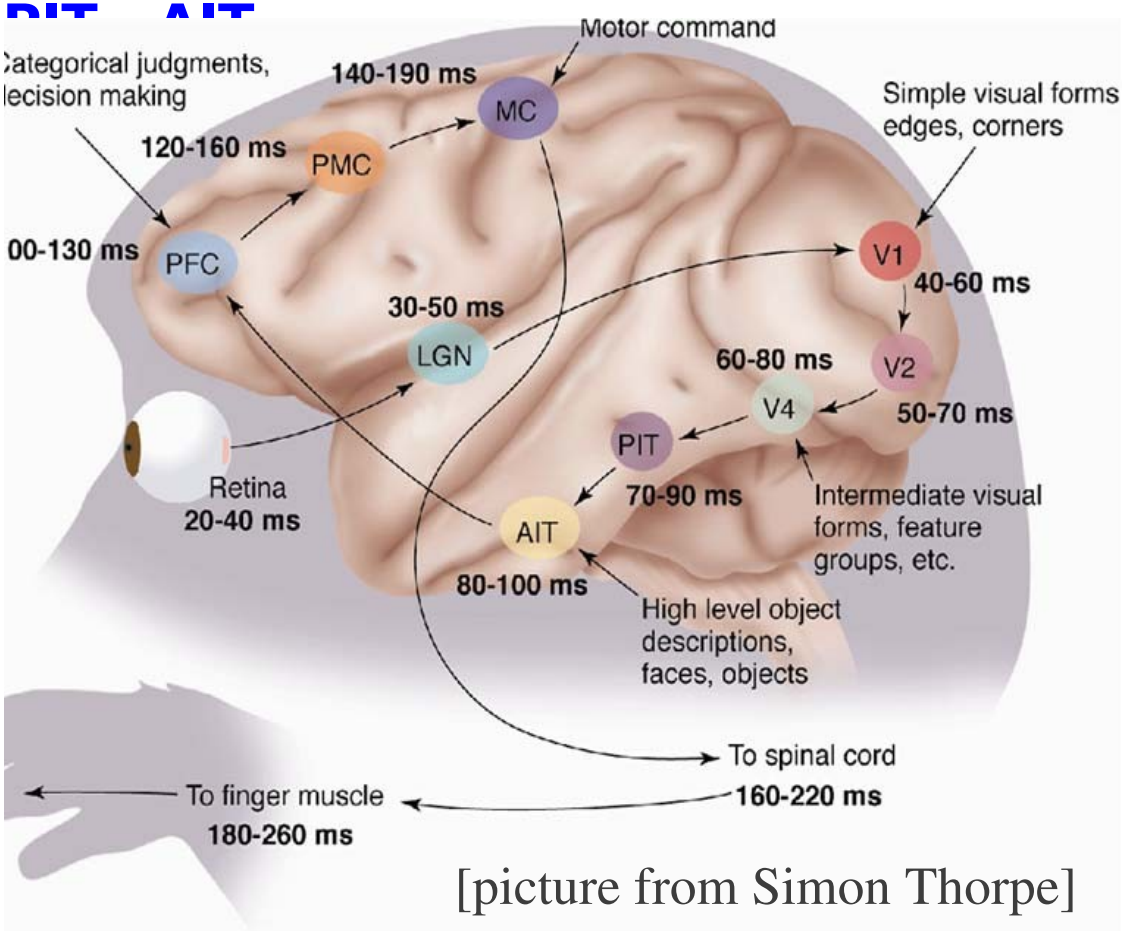


- **Another example of Analysis by synthesis**
- **Many-to-one mapping → must be probabilistic**
- **Don't know how big to make you model? → make it potentially infinite.**
- **Analysis by synthesis is slow. NPB is even slower.**

Does the brain use NPB?

- **The ventral (recognition) pathway in the visual cortex has multiple stages**

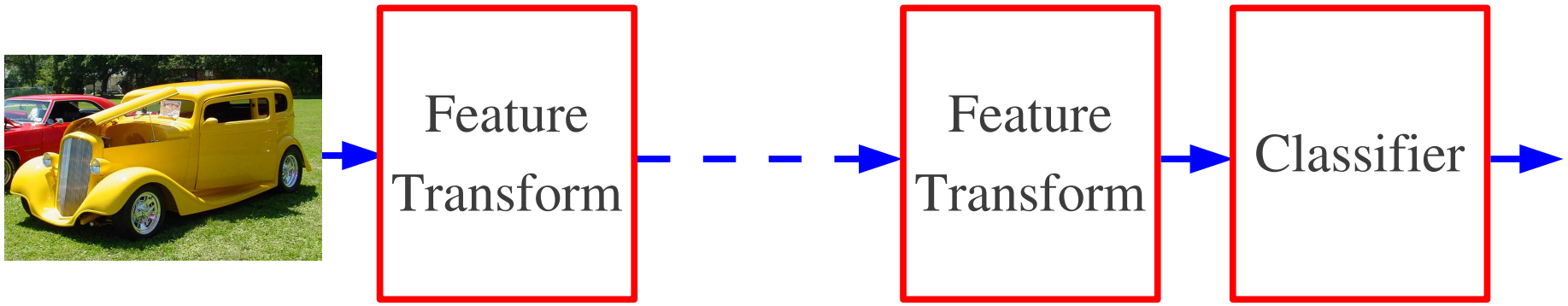




[picture from Simon Thorpe]

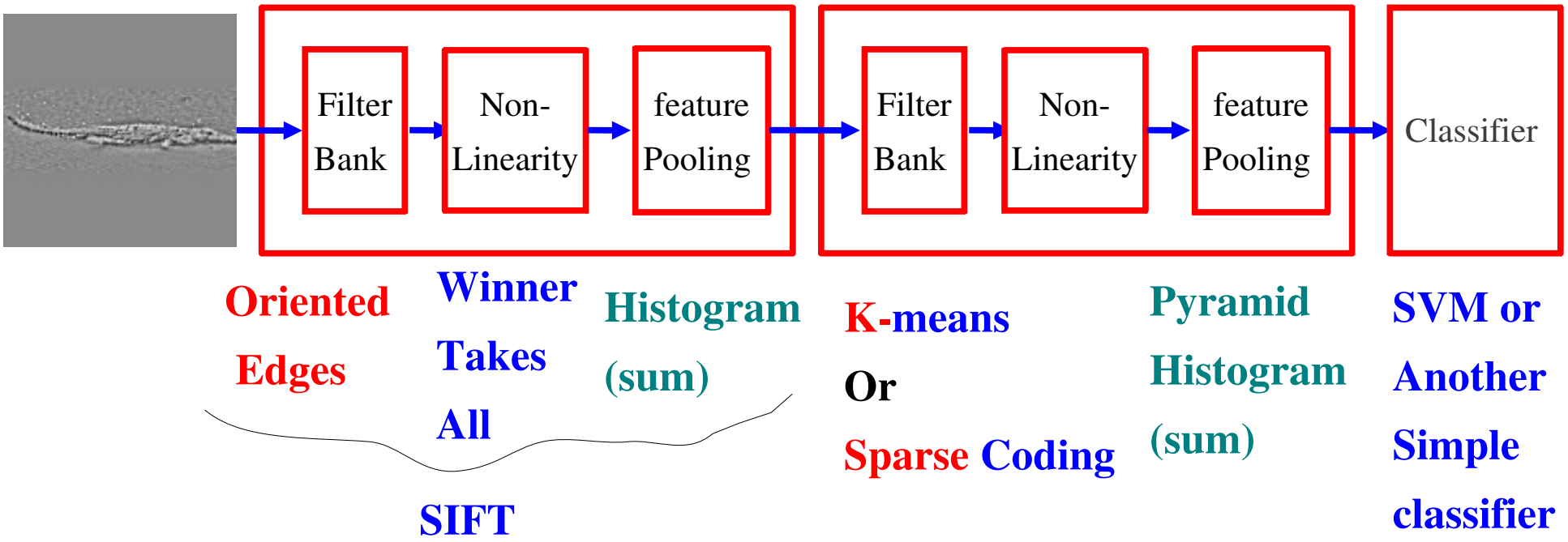
[Gallant & Van Essen]

Hierarchical Feed-Forward Processing



- **Most scenes are completely non-ambiguous**

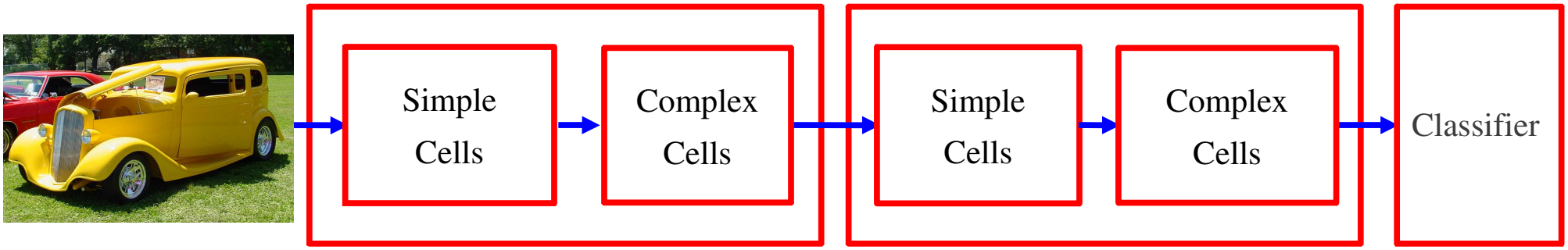
The dominant model for object recognition



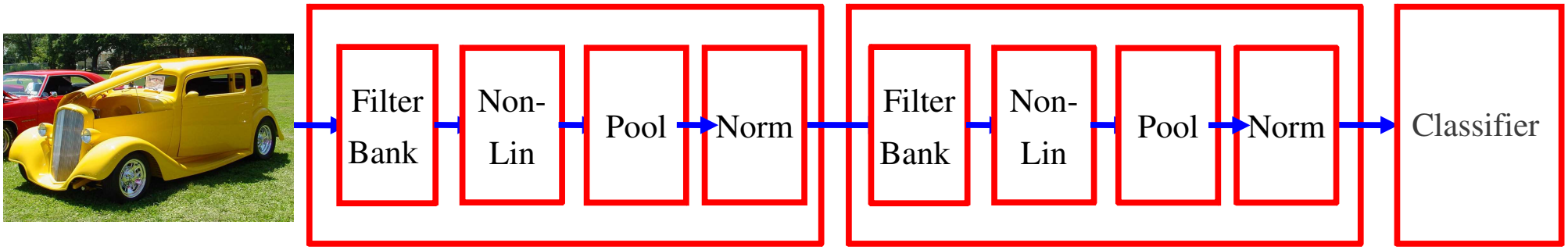
- **Fixed low-level Features + unsupervised mid-level features + simple classifier**
- **Example:**
  - SIFT + K-means + Pyramid pooling + SVM intersection kernel
    - [Lazebnik et al. CVPR 2006]
  - SIFT + Sparse coding or LCC + Pyramid pooling + SVM:
    - [Yann et al. CVPR 2009, Boureau et al. CVPR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

The dominant model is (indirectly) inspired by (half of) neuroscience

- **Simple Cells / Complex Cell: [Hubel & Wiesel 1962]**

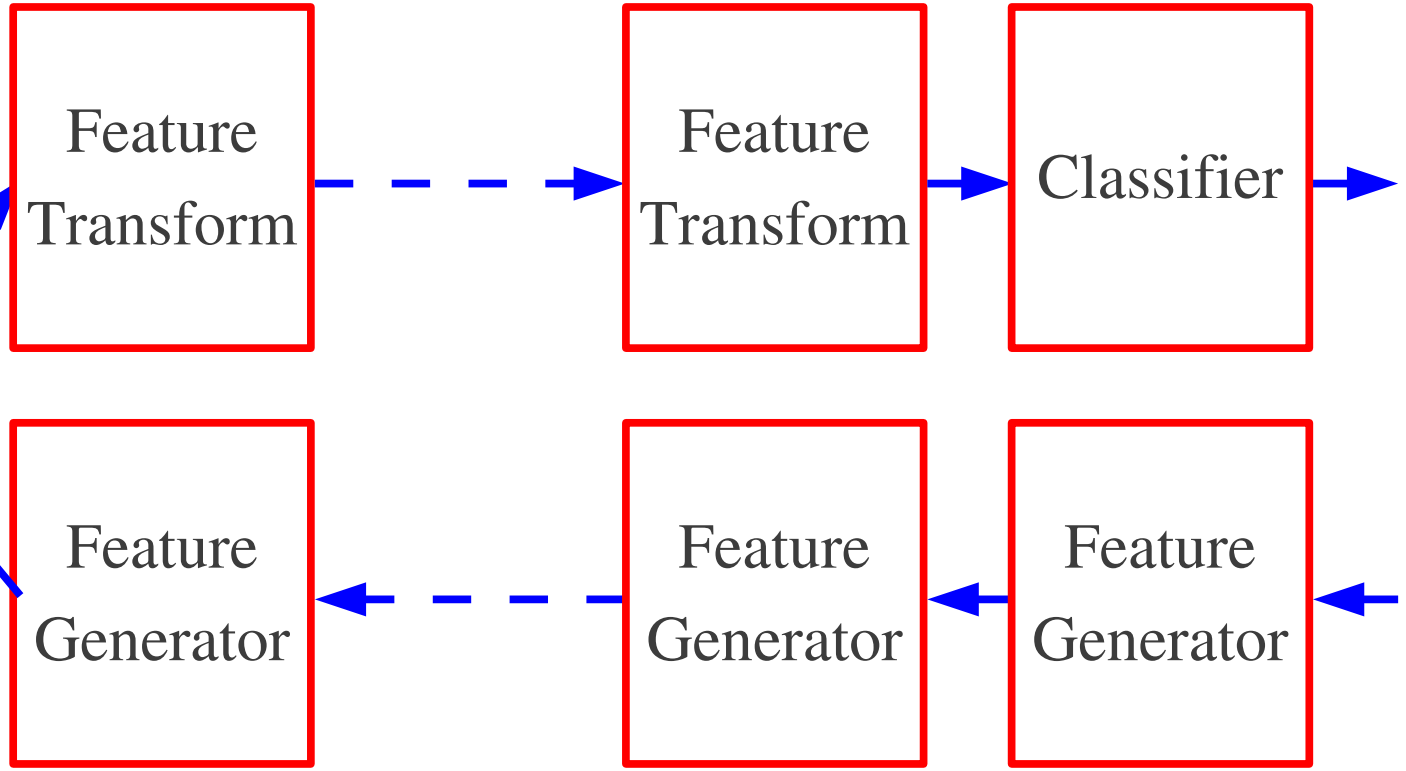


- **Basic Module: Filtering → Non-Linearity → Pooling → Normalization**



- **Computational Models: [Fukushima 1974-1982], [LeCun 1988-now], [Poggio 2005-now], [Ng 2006-now]**

Hierarchical Processing



- **Feed forward for fast inference**
- **Feedback for complicated/unusual cases**
- **Feedback path produces targets to train the feed-forward model**

Feed-Forward – Feed-Back Architecture for Learning

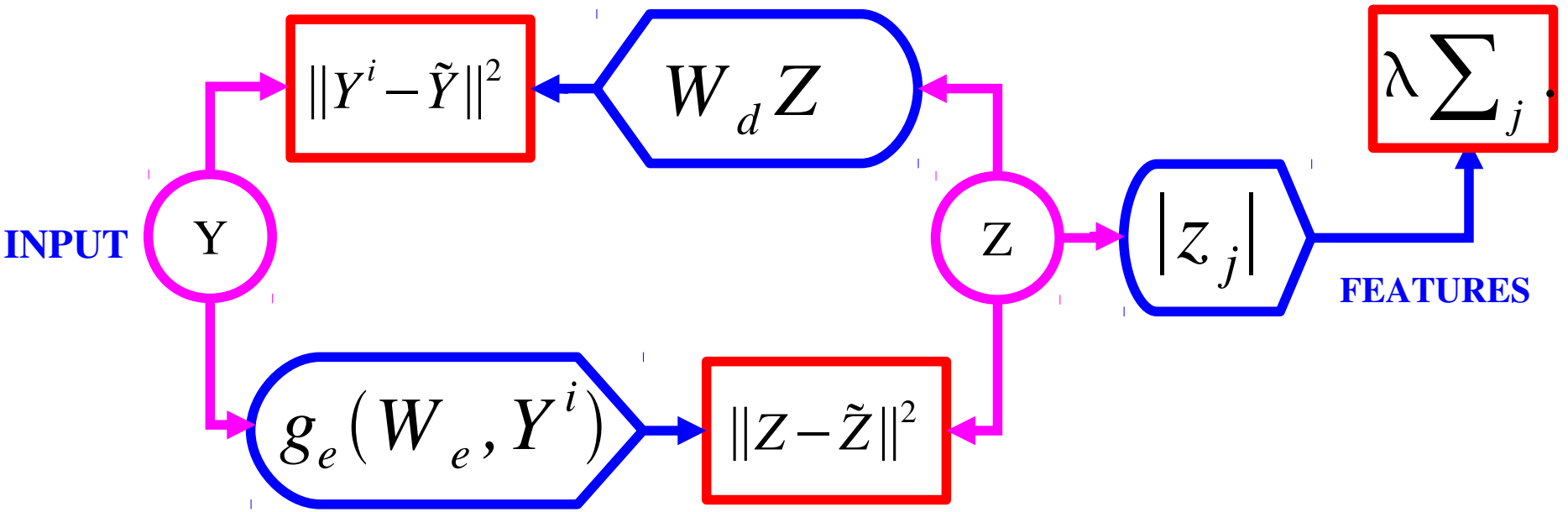
Predictive Sparse Decomposition (PSD)

[Kavukcuoglu, Ranzato, LeCun, 2009]

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

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

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



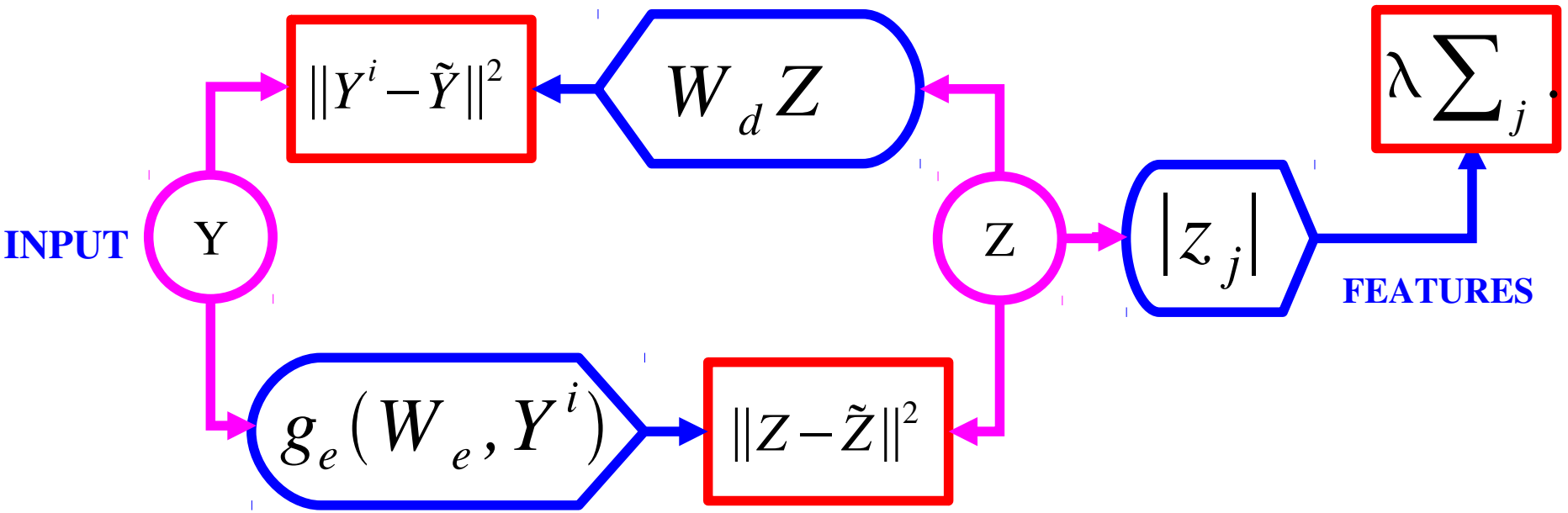
New York University

PSD: Inference

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

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

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

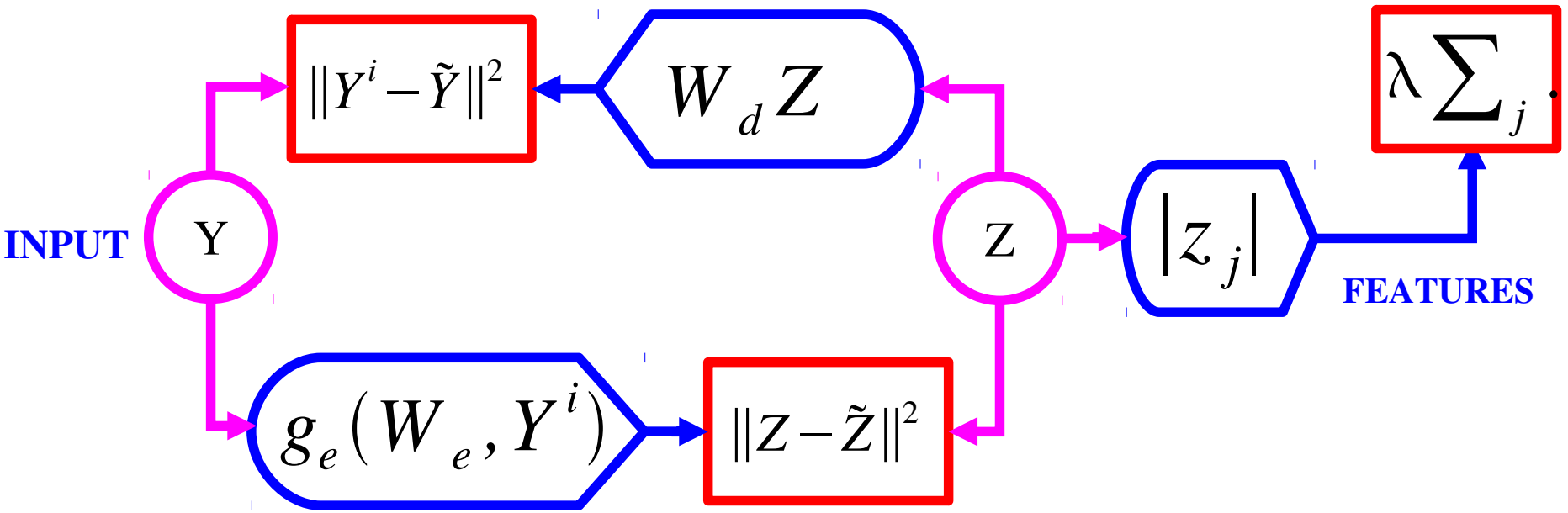


PSD: Learning [Kavukcuoglu et al. 2009]

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

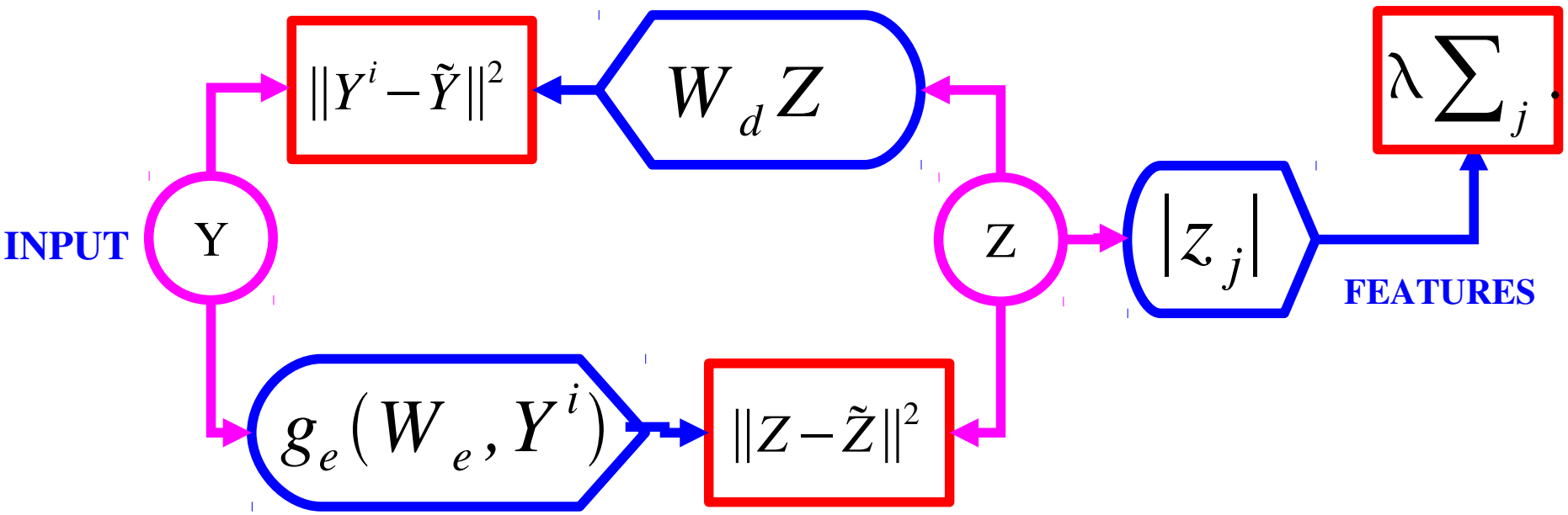
$$L(W_d, W_e) = \sum_i F(Y^i; W_d, W_e)$$

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



New York University

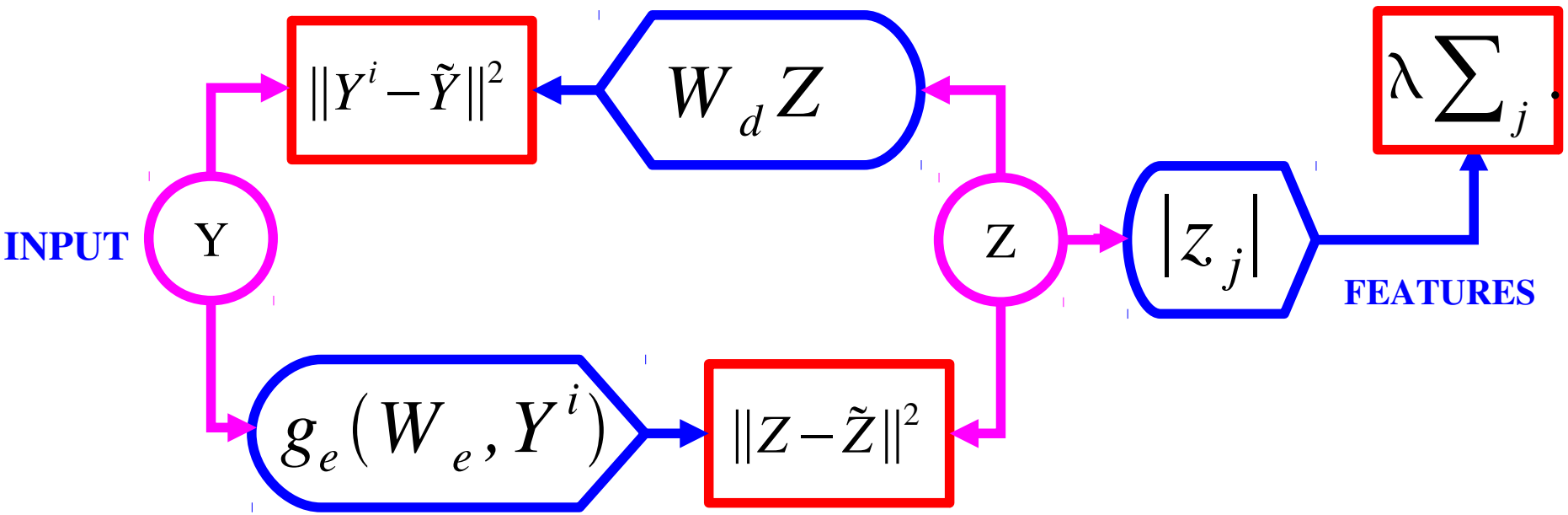
PSD: Learning Algorithm



- 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

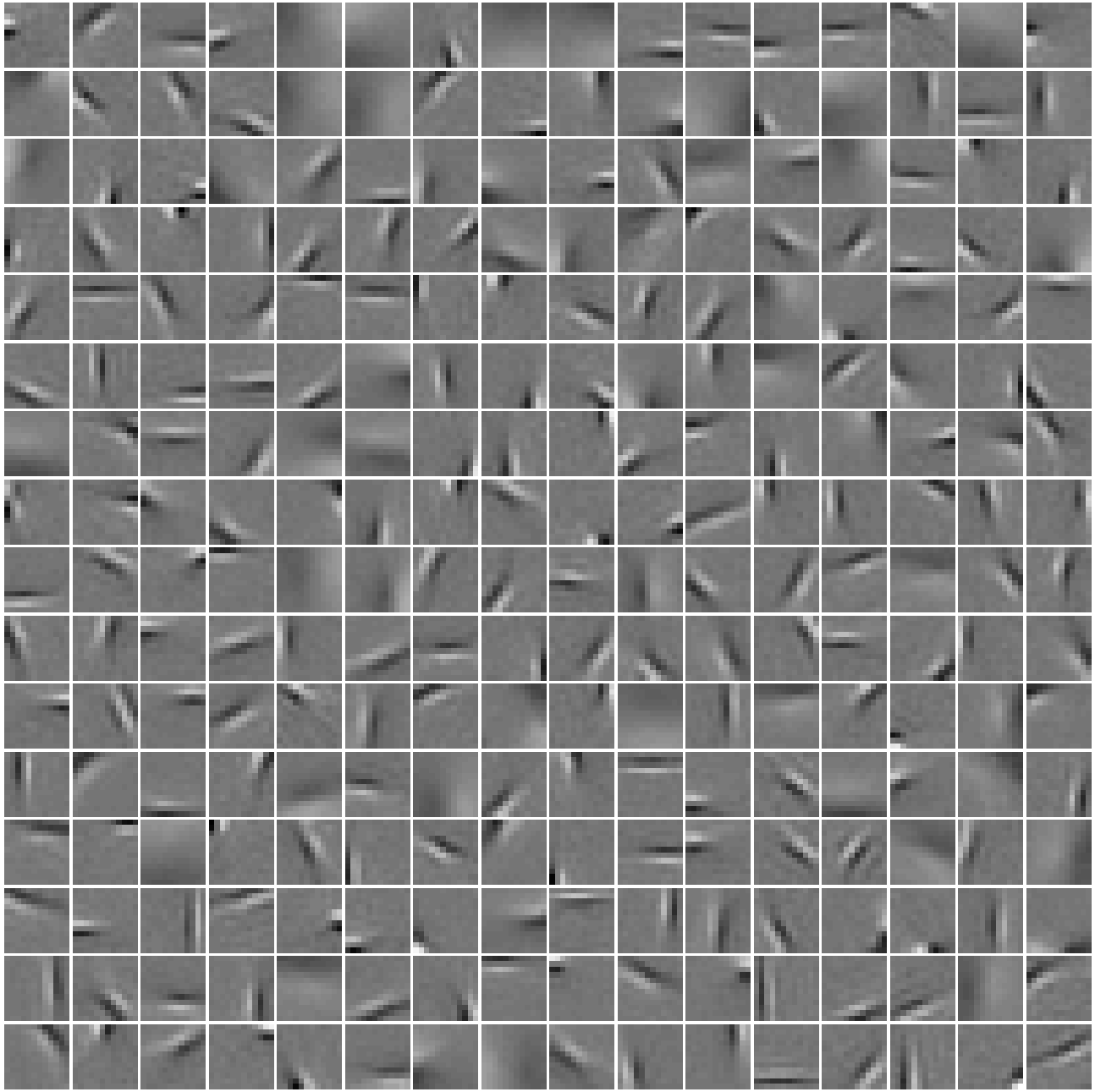
New York University

PSD: Encoder Architectures



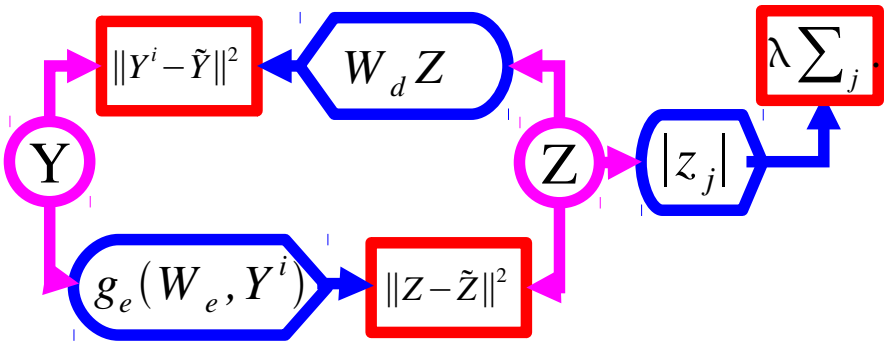
- **Simple: D.tanh(We.Y)**
- **Sophisticated/Iterative: Z(t) = shrink(We.Y + S.Z(t-1))**
- **For a discussion of best encoders, see [Gregor & LeCun, ICML 2010]**

Learned Features on natural patches: V1-like receptive fields



Using PSD to Train a Hierarchy of Features

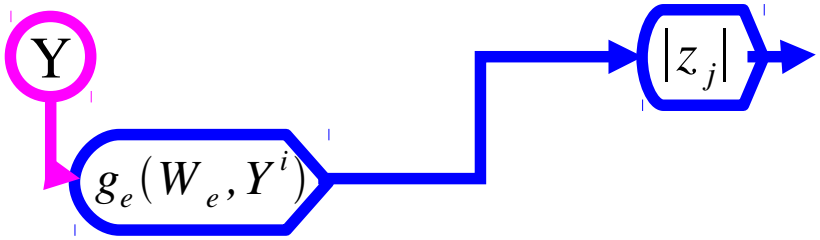
- **Phase 1: train first layer using PSD**



FEATURES

Using PSD to Train a Hierarchy of Features

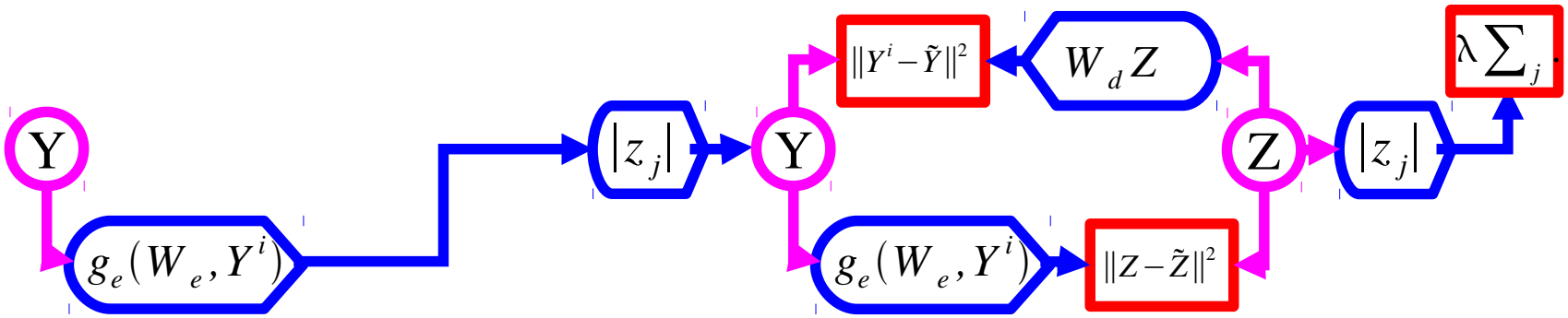
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor



FEATURES

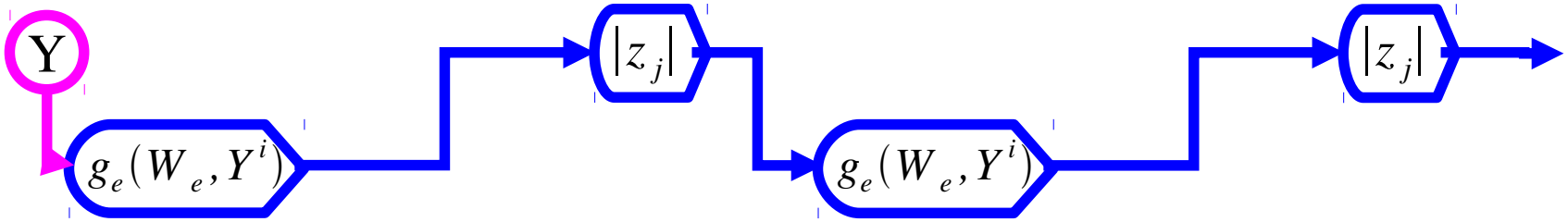
Using PSD to Train a Hierarchy of Features

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



Using PSD to Train a Hierarchy of Features

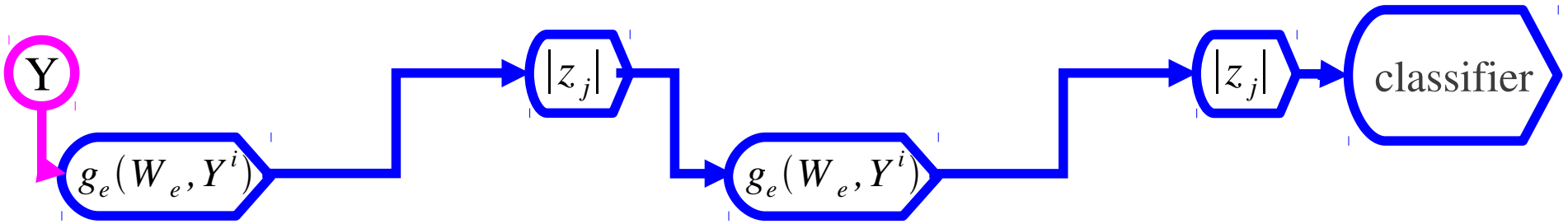
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor



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

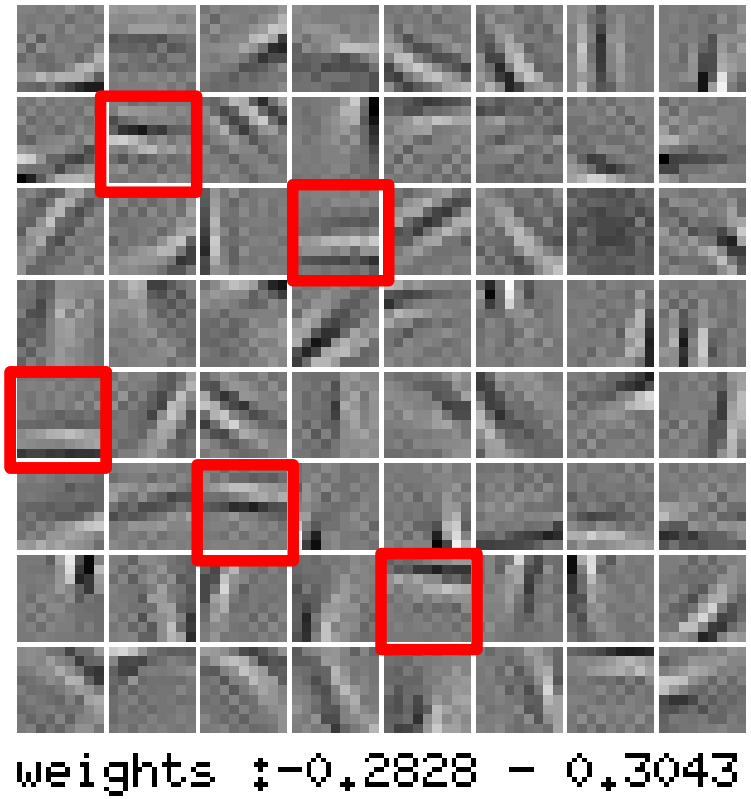


FEATURES

New York University

Convolutional Training

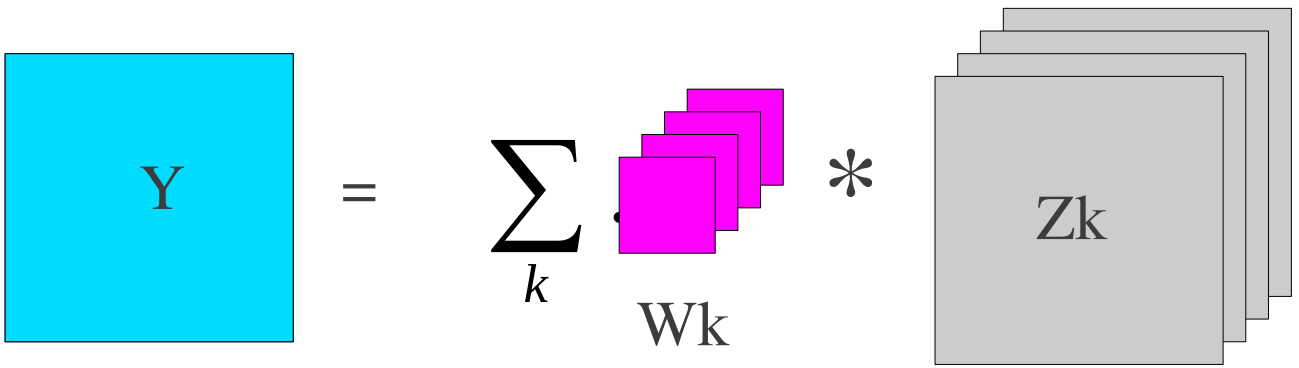
- **Problem:**
  - With patch-level training, the learning algorithm must reconstruct the entire patch with a single feature vector
  - But when the filters are used convolutionally, neighboring feature vectors will be highly redundant



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

Convolutional Sparse Coding

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



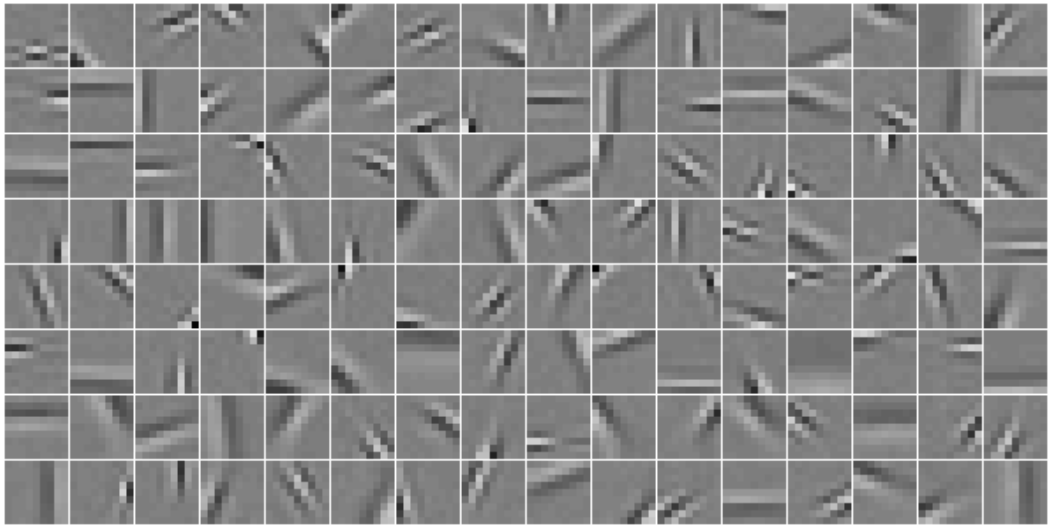
"deconvolutional networks" [Zeiler, Taylor, Fergus CVPR 2010]

Convolutional PSD: Encoder with a soft sh() Function

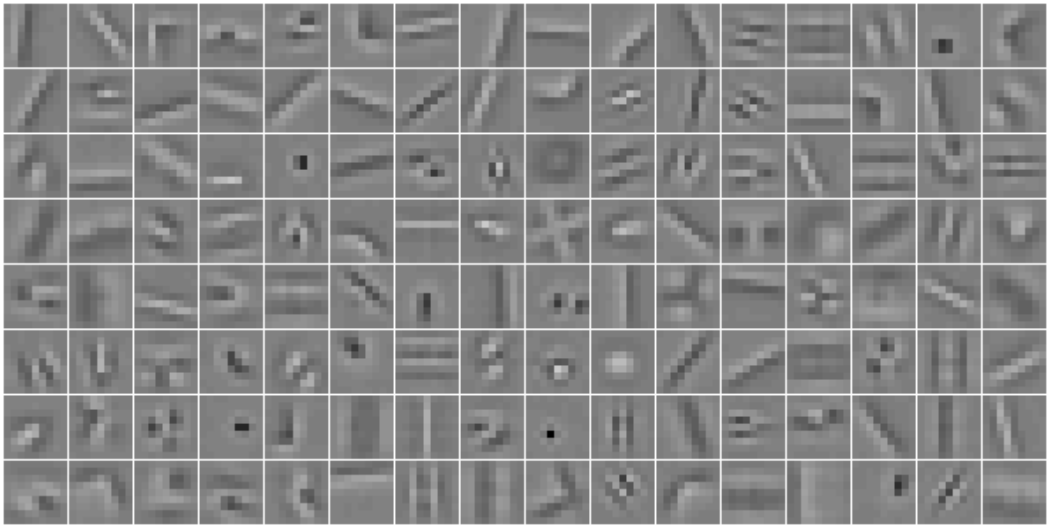
Convolutional Formulation

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

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

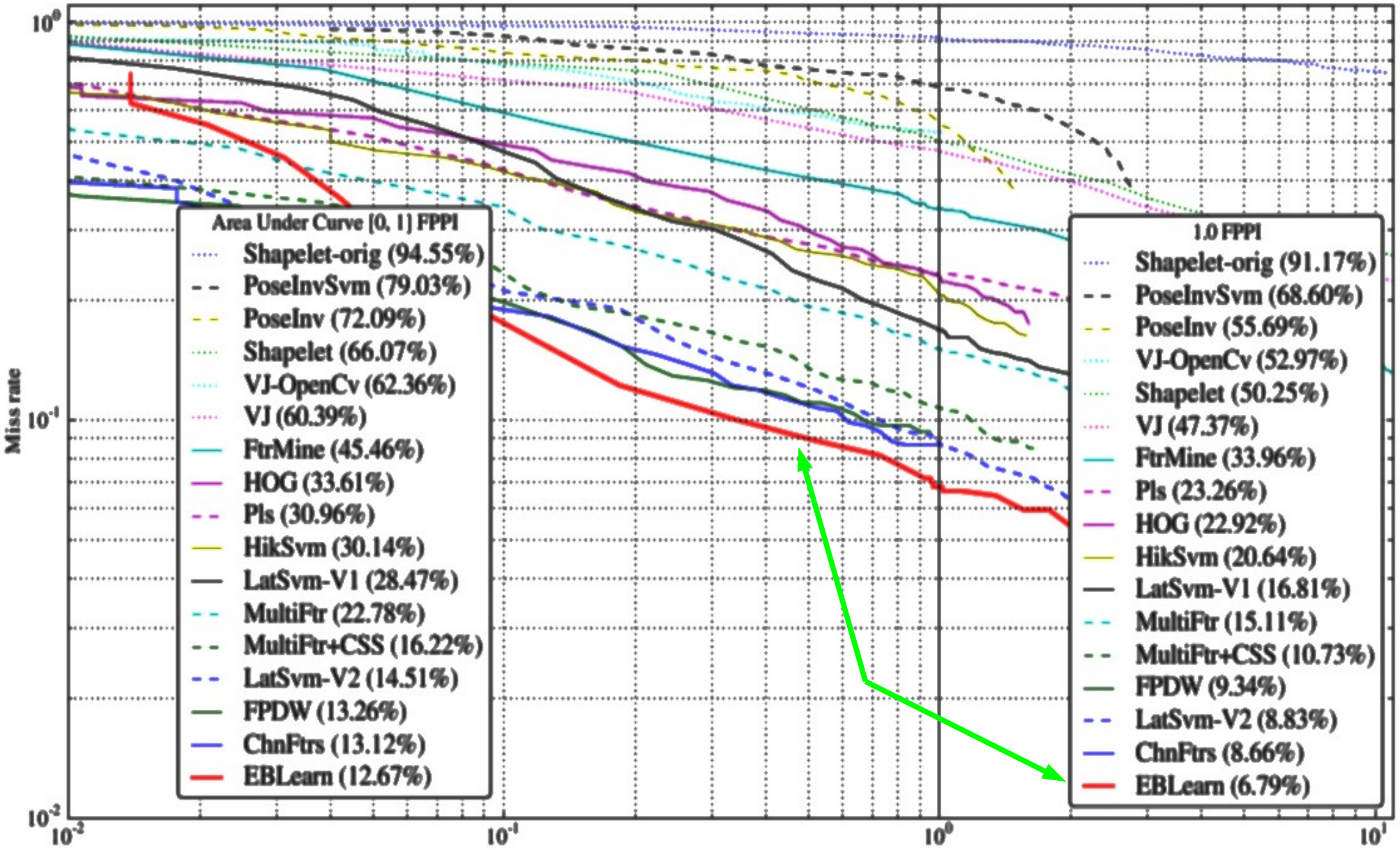


- ***PATCH*** based learning



- ***CONVOLUTIONAL*** learning

Pedestrian Detection (INRIA Dataset)



[Kavukcuoglu et al. NIPS 2010]

Pedestrian Detection: Examples

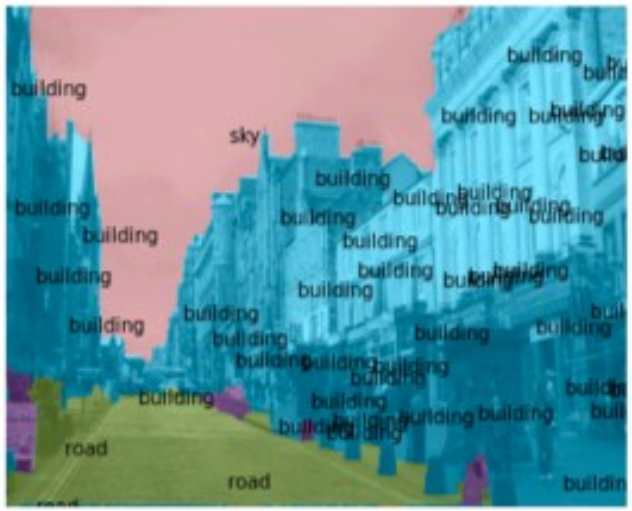
[Kavukcuoglu et al. NIPS 2010]

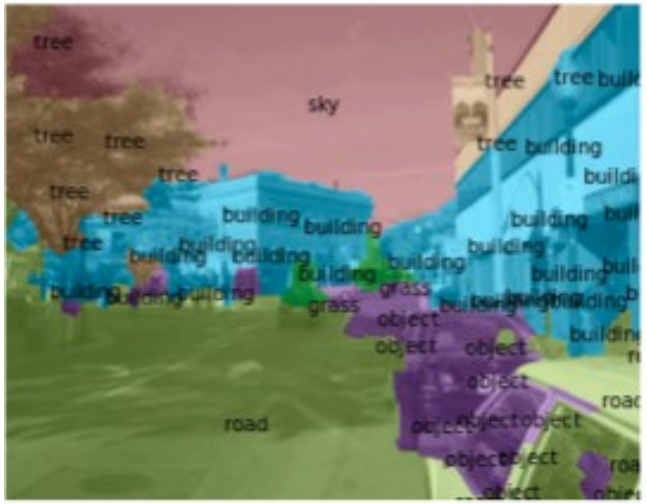
Multi-Scale ConvNet: Application to Full-Scene Labeling

[Farabet et al. 2012, under review]

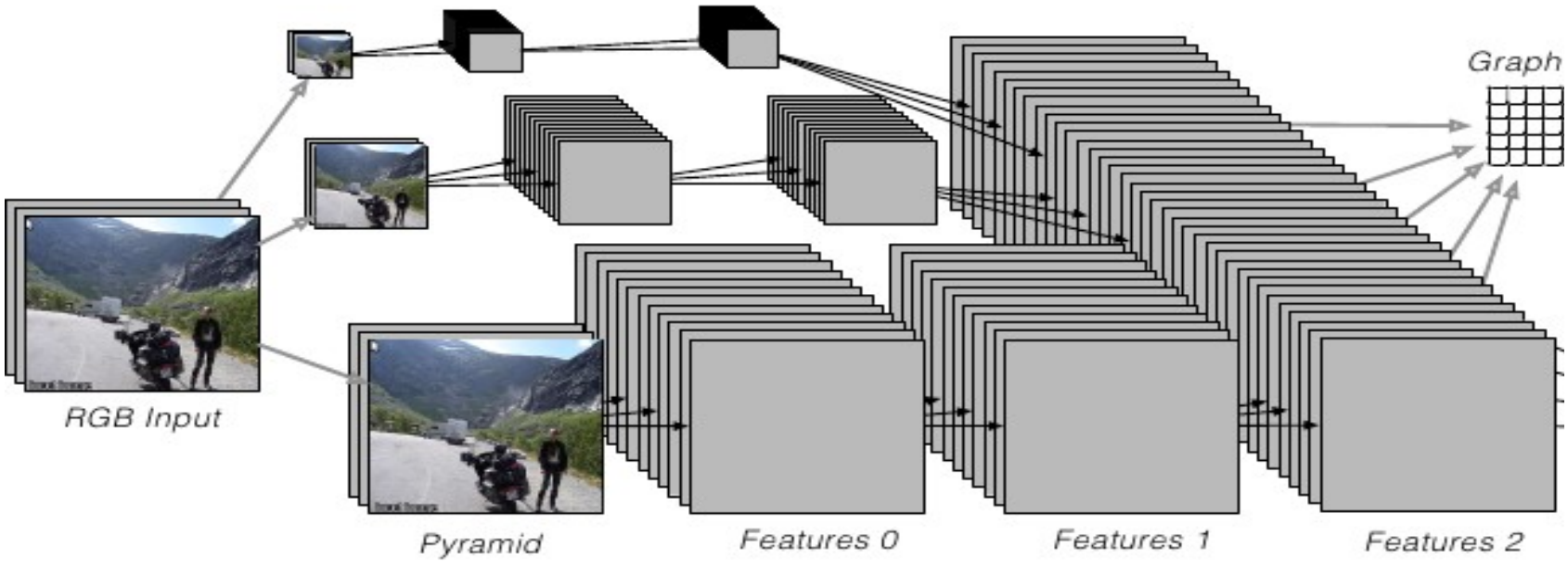
Labeling Every Pixel in the Image

- **Full-Scene category labeling ("scene parsing")**



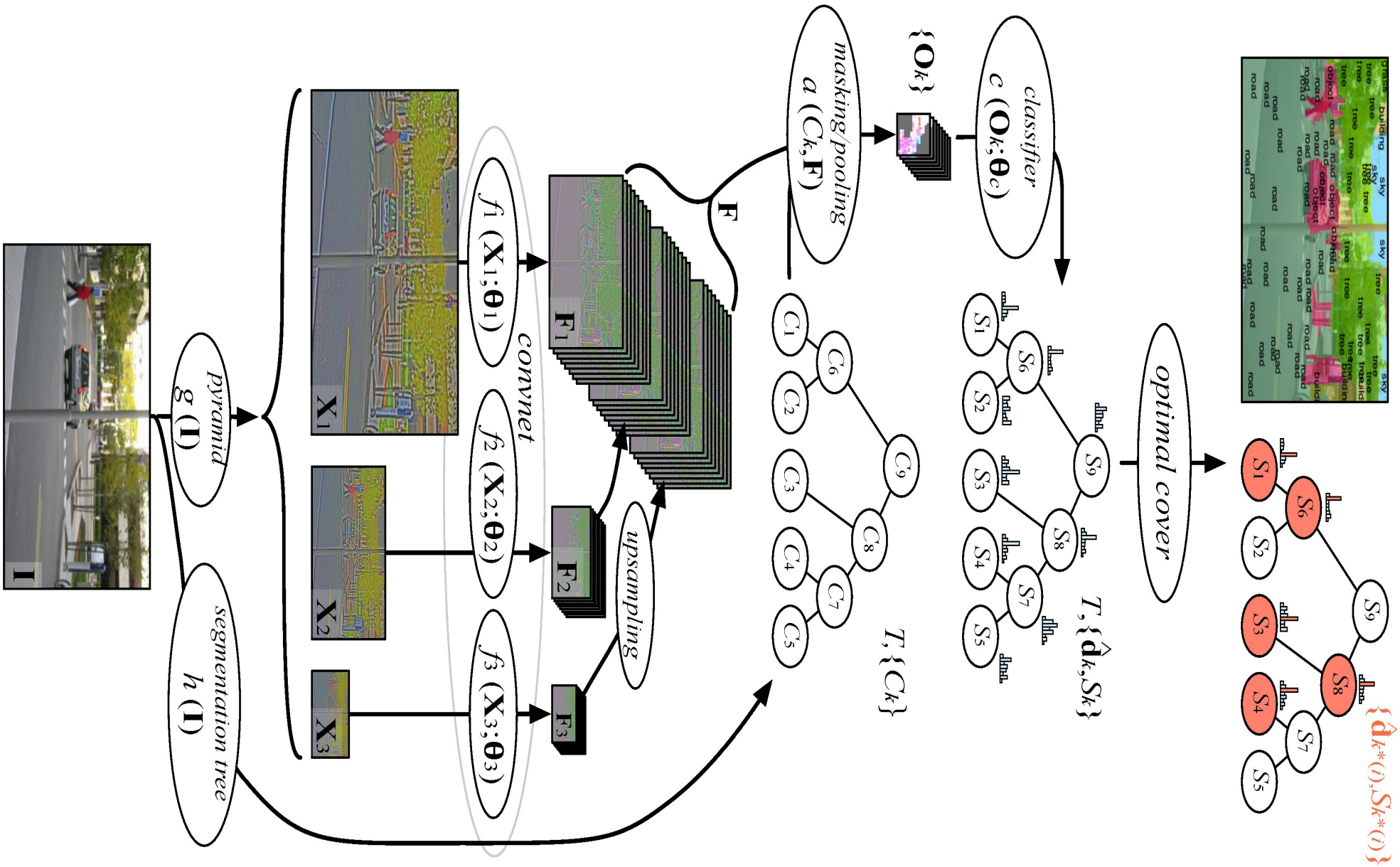


Scene Parsing/Labeling: ConvNet Architecture



[Farabet et al. 2012]

Scene Parsing/Labeling: System Architecture



[Farabet et al. 2011]

Scene Parsing/Labeling: Performance

- **Stanford Background Dataset [Gould 1009]: 8 categories**

	P / C	CT
Gould *et al.* [6]	76.4% / -	10s to 10min
Munoz *et al.* [17]	76.9% / 66.2%	12s
Tighe *et al.* [24]	77.5% / -	10s to 5min
Socher *et al.* [23]	78.1% / -	?
Kumar *et al.* [13]	79.4% / -	10s to 10min
multiscale net	77.5 % / 70.0%	0.5s
multiscale net + cover	**79.5% / 74.3%**	1s

- **SIFT Flow dataset [Liu 2009]: 33 categories**

	P / C
Liu *et al.* [15]	74.75 % / -
Tighe *et al.* [24]	76.9 % / 29.4 %
multiscale net + cover[1]	**78.5 % / 29.6 %**
multiscale net + cover[2]	74.2 % / **46.0 %**

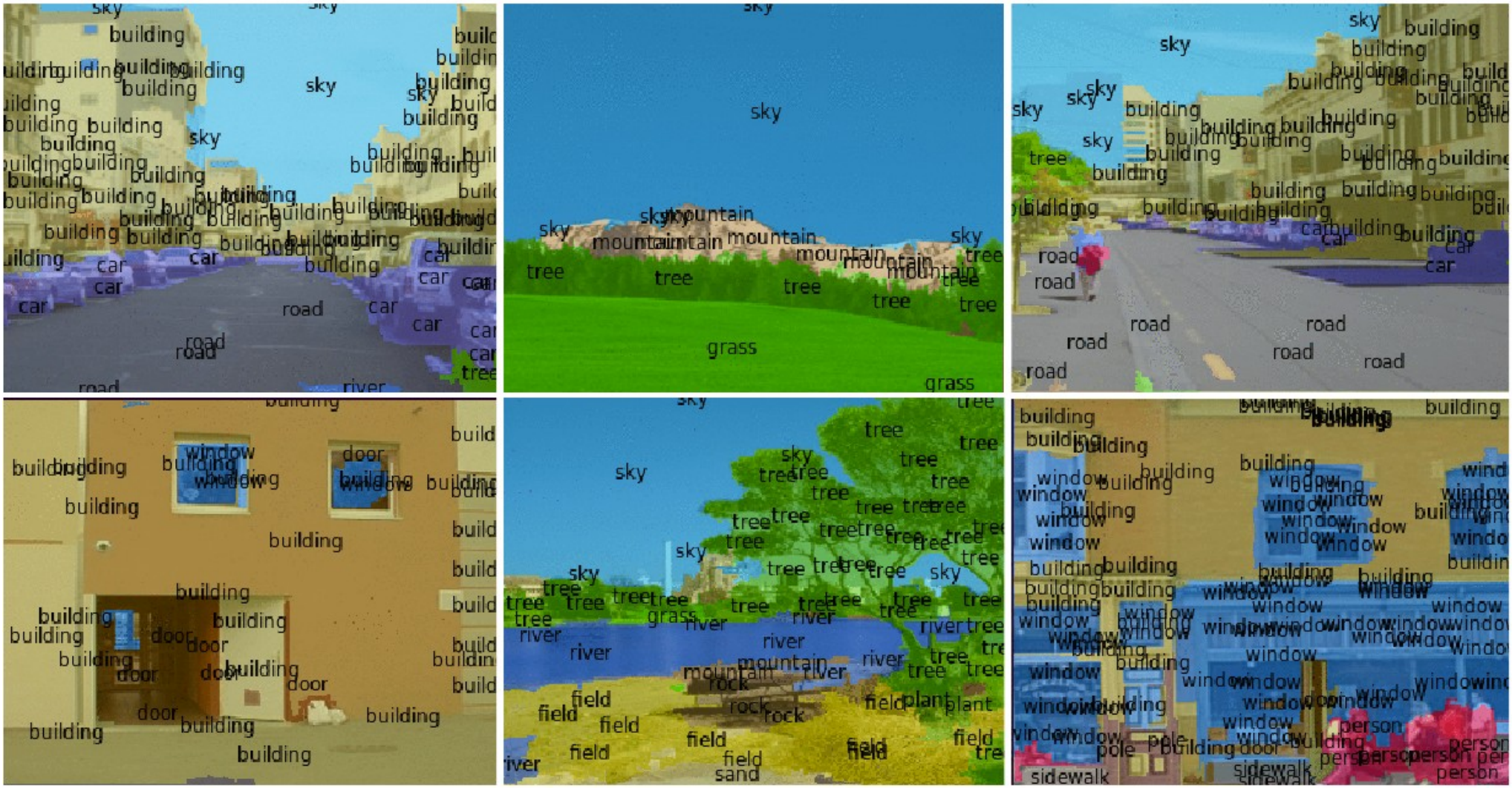
- **Barcelona dataset[Tighe 2010]: 170 categories.**

	P / C
Tighe *et al.* [24]	66.9 % / 7.6 %
multiscale net + cover[1]	**67.8 % / 9.5 %**
multiscale net + cover[2]	39.1 % **10.7** %

[Farabet et al. 2012]

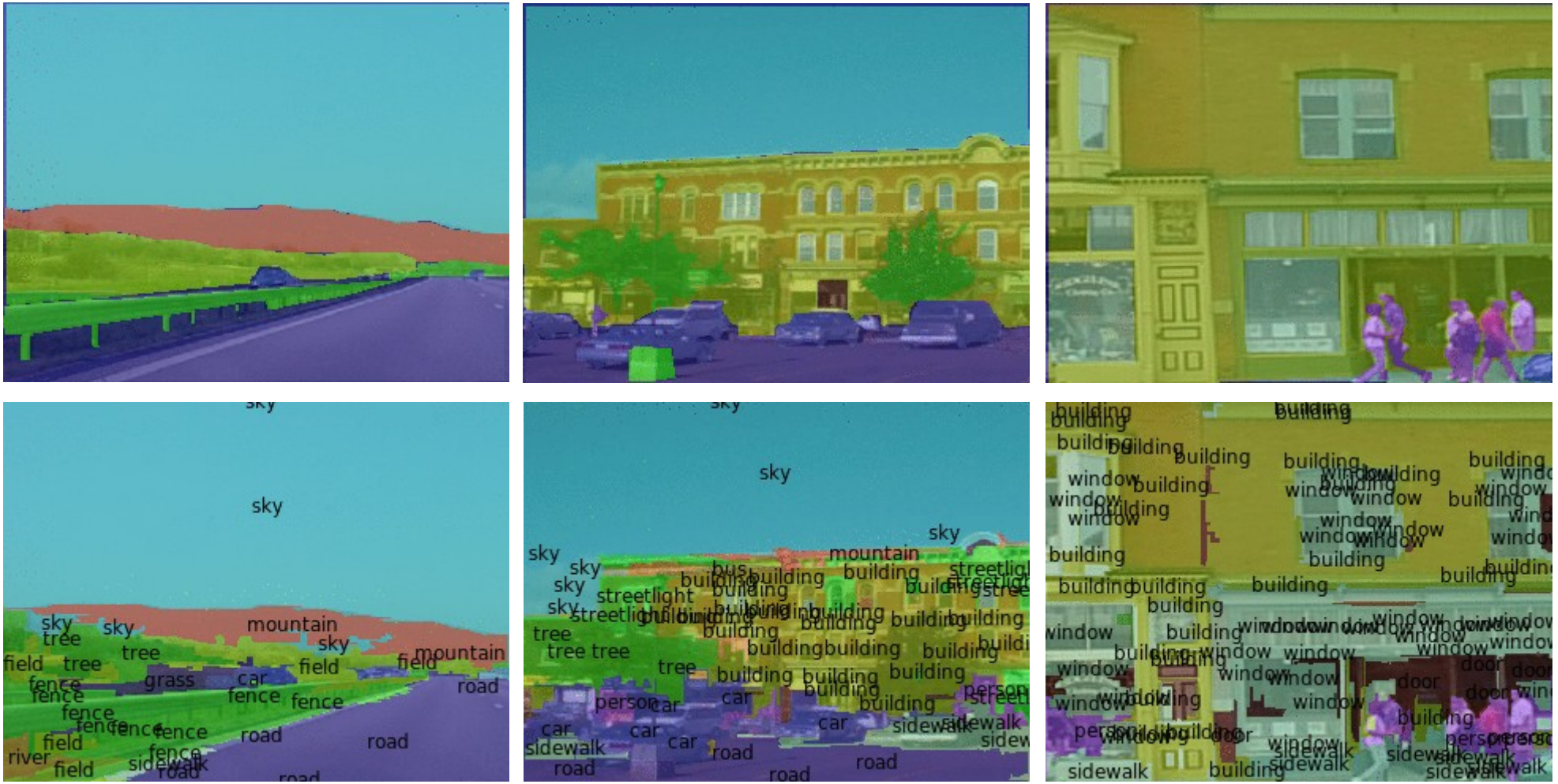
Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

- **Samples from the SIFT-Flow dataset (Liu)**



[Farabet et al. 2012]

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

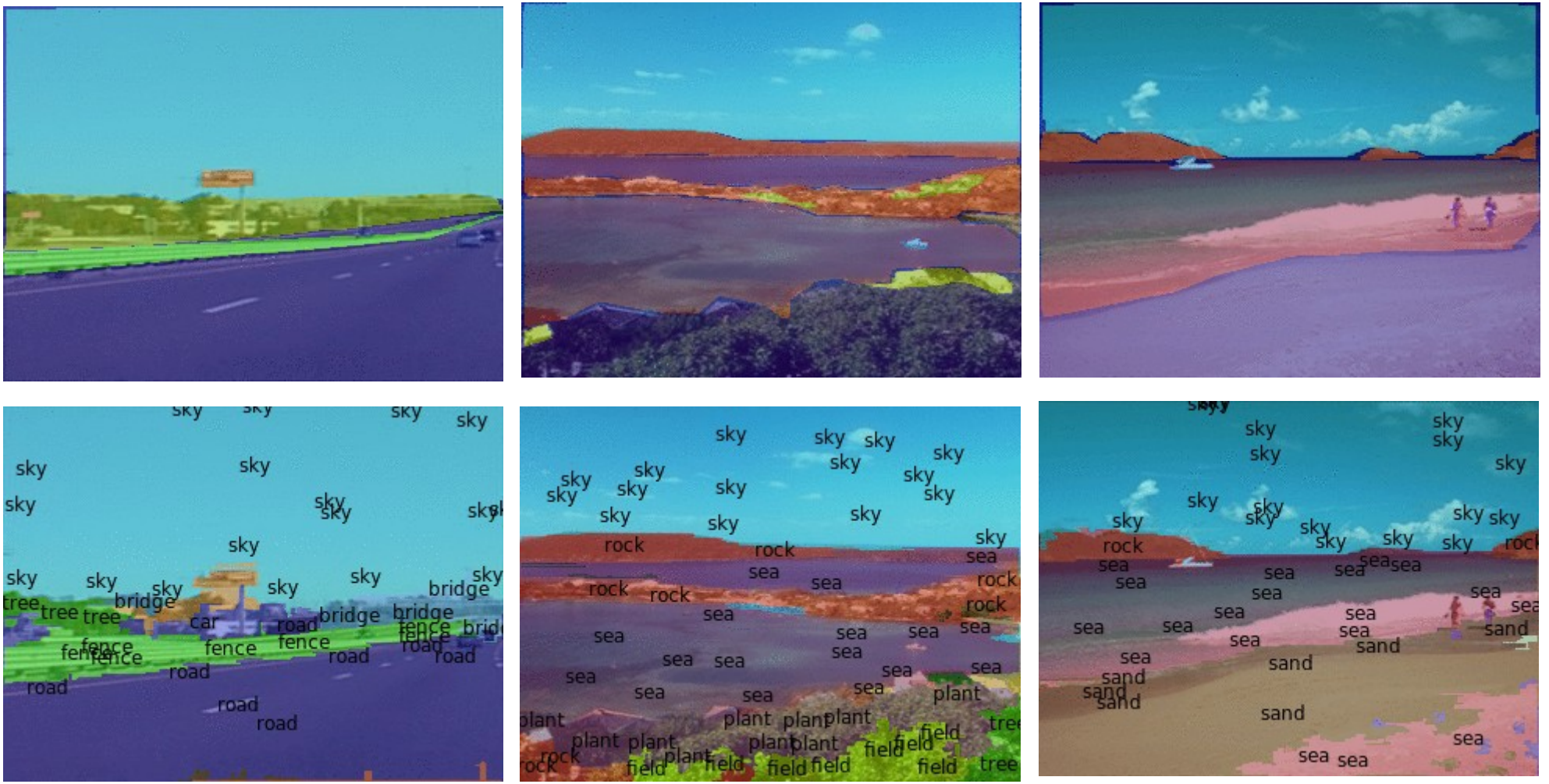


[Farabet et al. 2012]

Scene Parsing/Labeling

[Farabet et al. 2012]

Scene Parsing/Labeling



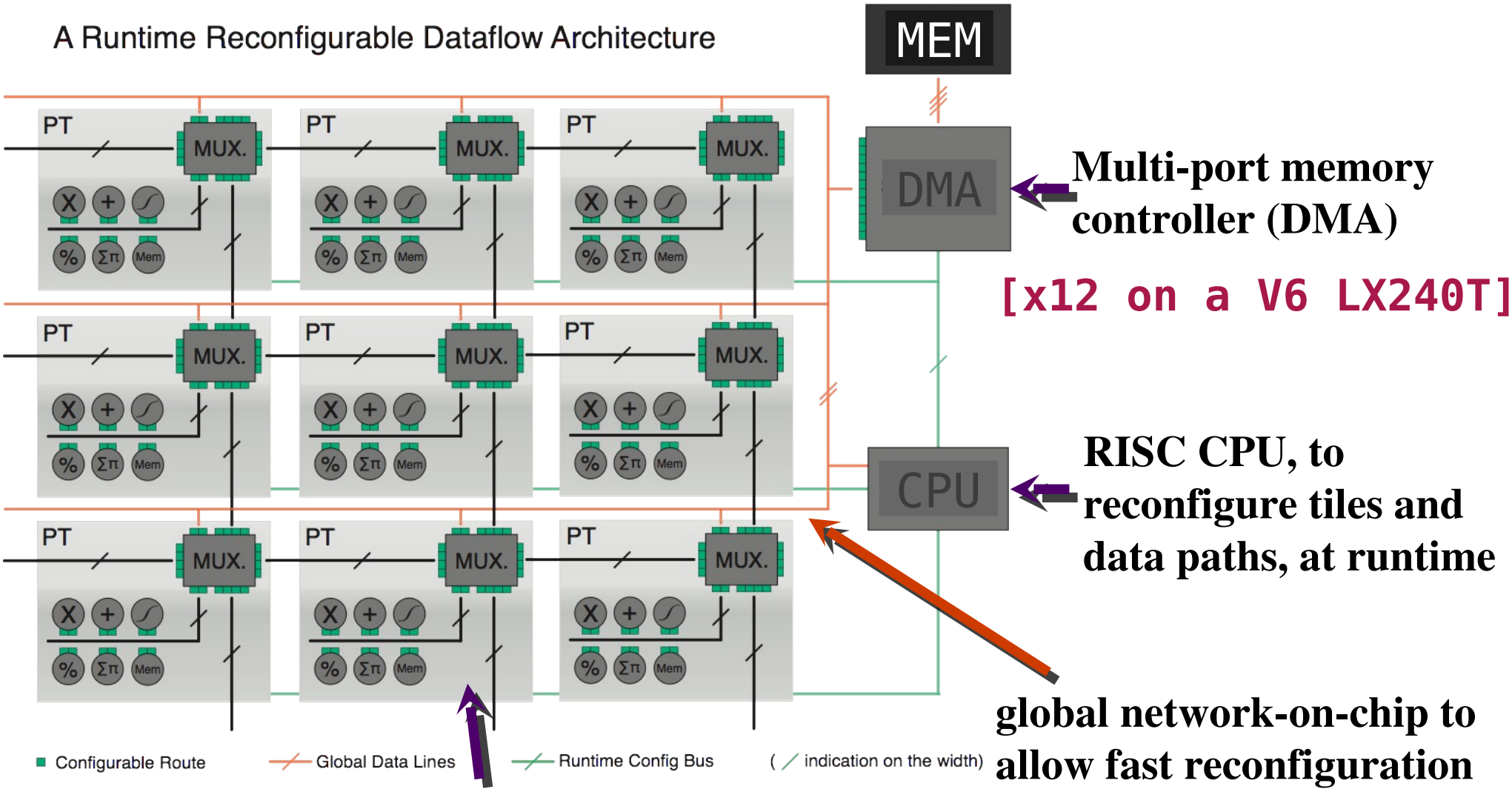
[Farabet et al. 2012]

Hardware Implementations in FPGAs and chips

Clément Farabet & Yann LeCun

Collaboration with Eugenio Culurciello's lab (Purdu)e

NewFlow: Architecture



Multi-port memory controller (DMA)

[x12 on a V6 LX240T]

RISC CPU, to reconfigure tiles and data paths, at runtime

global network-on-chip to allow fast reconfiguration

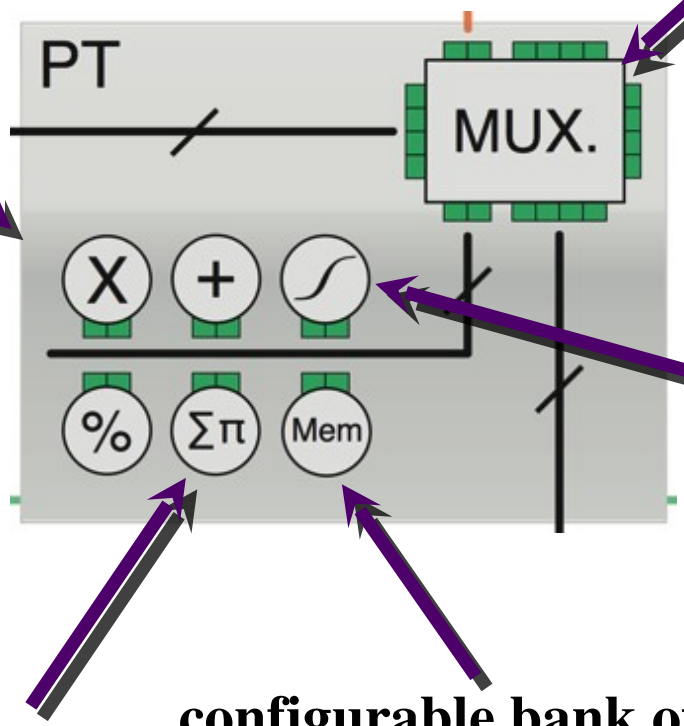
grid of passive processing tiles (PTs)

[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]

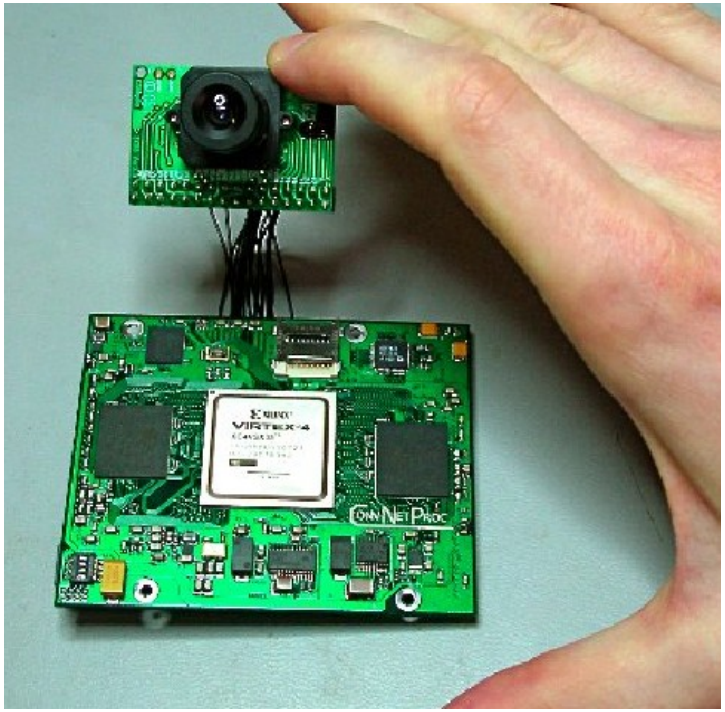
configurable bank of FIFOs , for stream buffering, up to 10kB per PT

[x8]

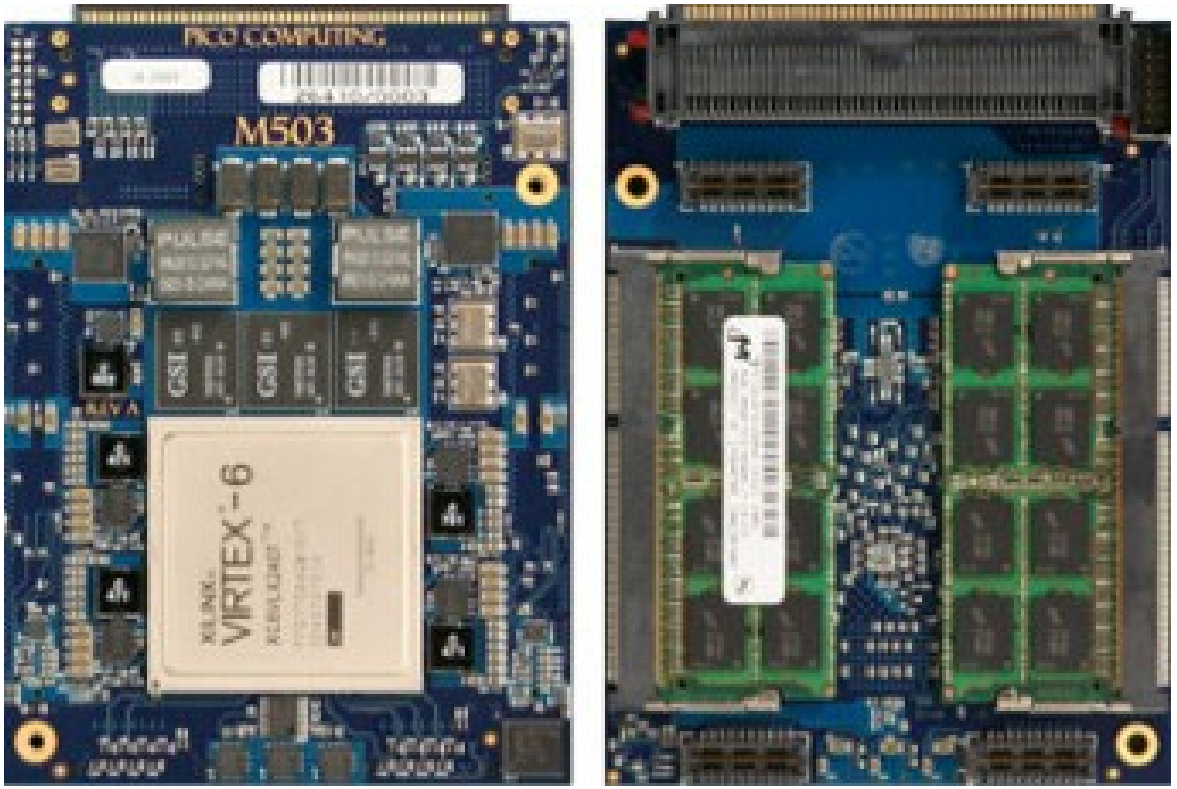
[Virtex6 LX240T]

NewFlow: Latest Hardware

- **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**



Old board with Virtex-4



New board with Virtex-6

NewFlow: Performance

	Intel 2Core	neuFlow Virtex4	neuFlow Virtex 6	nVidia GT335m	neuFlow IBM 45nm	nVidia GTX480
Peak GOP/sec	10?	40	160	182	1280	1350
Actual GOP/sec	1.1	37	147	54	1164	294
FPS	1.4	46	182	67	1456	374
Power (W)	30	10	10	30	5	220
Embed? (GOP/s/W)	0.04	3.7	14.7	1.8	232.8	1.34

Labeling Every Pixel in the Image