

# Deep Learning

Yann Le Cun

The Courant Institute of Mathematical Sciences

New York University

<http://yann.lecun.com>

# The Challenges of Machine Learning

## ● How can we use learning to progress towards AI?

- ▶ Can we find learning methods that scale?
- ▶ Can we find learning methods that solve **really complex problems** end-to-end, such as vision, natural language, speech....?

## ● How can we learn the structure of the world?

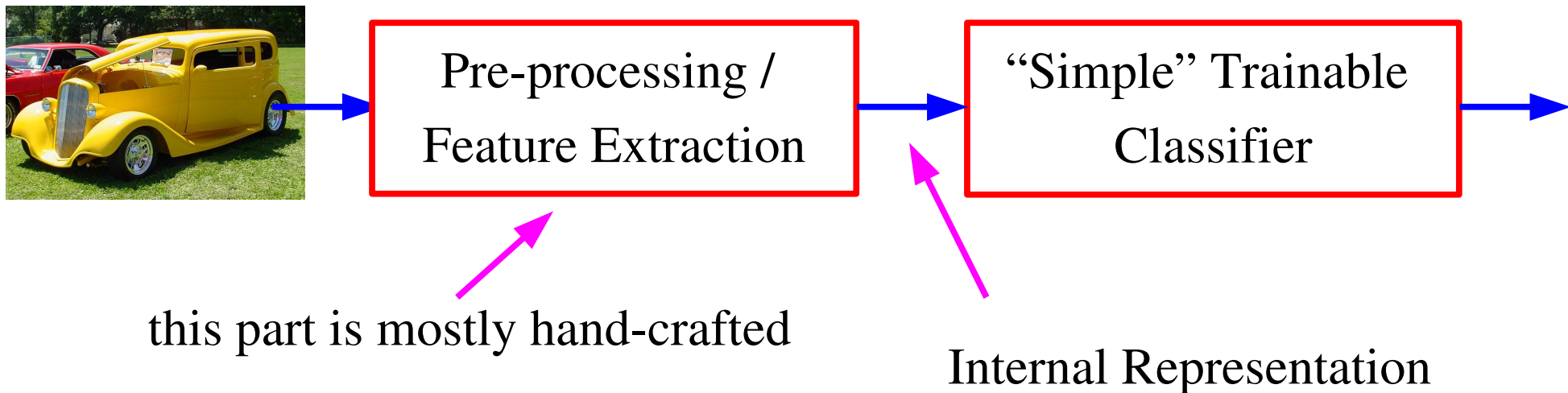
- ▶ How can we build/learn internal representations of the world that allow us to discover its hidden structure?
- ▶ How can we learn internal representations that capture the relevant information and eliminates irrelevant variabilities?

## ● How can a human or a machine learn internal representations by just looking at the world?

# The Next Frontier in Machine Learning: Learning Representations

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

# The Traditional “Shallow” Architecture for Recognition



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

# The Next Challenge of ML, Vision (and Neuroscience)

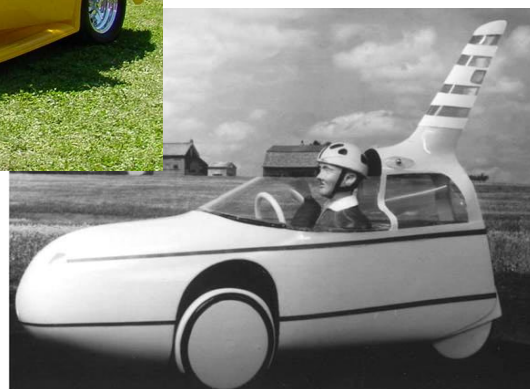
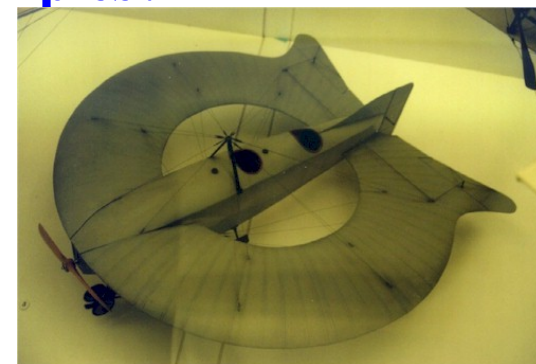
## How do we learn invariant representations?

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

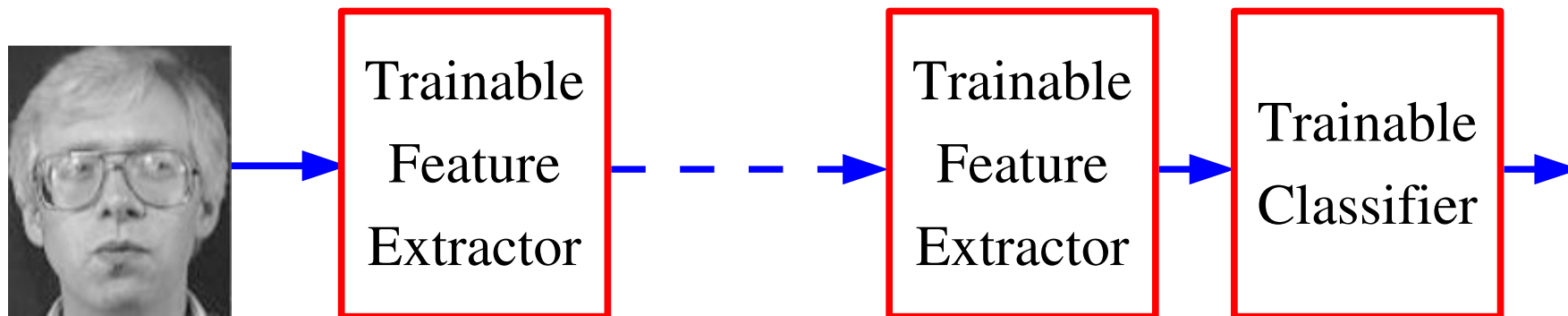


## How can we learn visual categories from just a few examples?

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



# Good Representations are Hierarchical



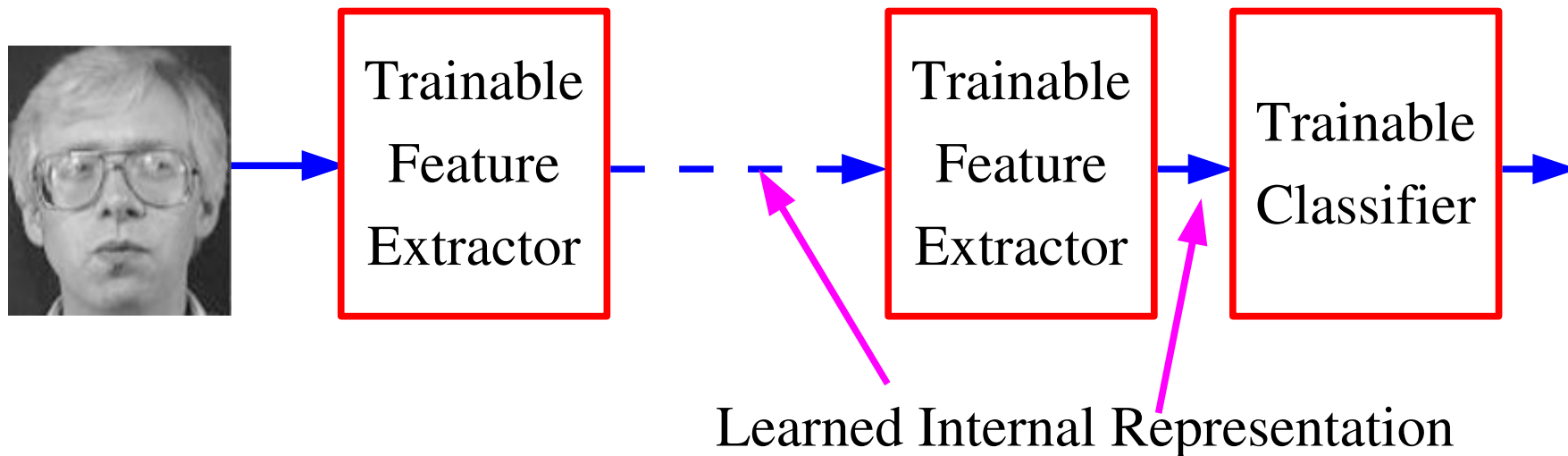
## ● In Language: hierarchy in syntax and semantics

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

## ● In Vision: part-whole hierarchy

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

# “Deep” Learning: Learning Hierarchical Representations



- **Deep Learning:** learning a hierarchy of internal representations
- From low-level features to mid-level invariant representations, to object identities
- Representations are increasingly invariant as we go up the layers
- using multiple stages gets around the specificity/invariance dilemma

# The Primate's Visual System is Deep

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



## Do we really need deep architectures?

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

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

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

- Deep learning machines

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

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

# Why are Deep Architectures More Efficient?

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

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

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

## • Example1: N-bit parity

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

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

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

## Strategies (a parody of [Hinton 2007])

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

# Supervised Deep Learning, The Convolutional Network Architecture

## • Convolutional Networks:

- ▶ [LeCun et al., Neural Computation, 1988]
- ▶ [LeCun et al., Proc IEEE 1998] (handwriting recognition)

## • Face Detection and pose estimation with convolutional networks:

- ▶ [Vaillant, Monrocq, LeCun, IEE Proc Vision, Image and Signal Processing, 1994]
- ▶ [Osadchy, Miller, LeCun, JMLR vol 8, May 2007]

## • Category-level object recognition with invariance to pose and lighting

- ▶ [LeCun, Huang, Bottou, CVPR 2004]
- ▶ [Huang, LeCun, CVPR 2006]

## • autonomous robot driving

- ▶ [LeCun et al. NIPS 2005]

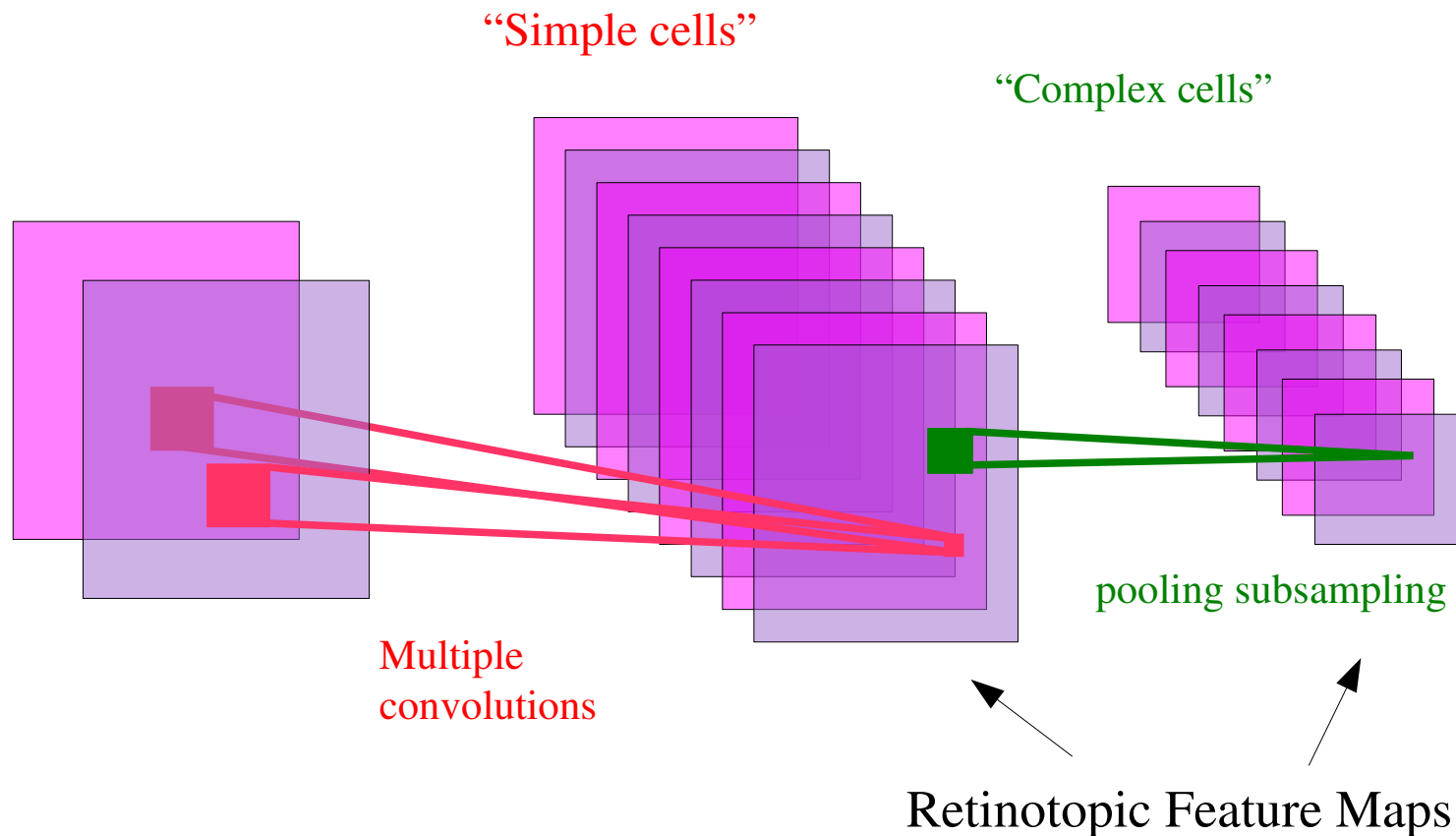
# Deep Supervised Learning is Hard

- **The loss surface is non-convex, ill-conditioned, has saddle points, has flat spots.....**
- **For large networks, it will be horrible! (not really, actually)**
- **Back-prop doesn't work well with networks that are tall and skinny.**
  - ▶ Lots of layers with few hidden units.
- **Back-prop works fine with short and fat networks**
  - ▶ But over-parameterization becomes a problem without regularization
  - ▶ Short and fat nets with fixed first layers aren't very different from SVMs.
- **For reasons that are not well understood theoretically, back-prop works well when they are highly structured**
  - ▶ e.g. convolutional networks.

# 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.

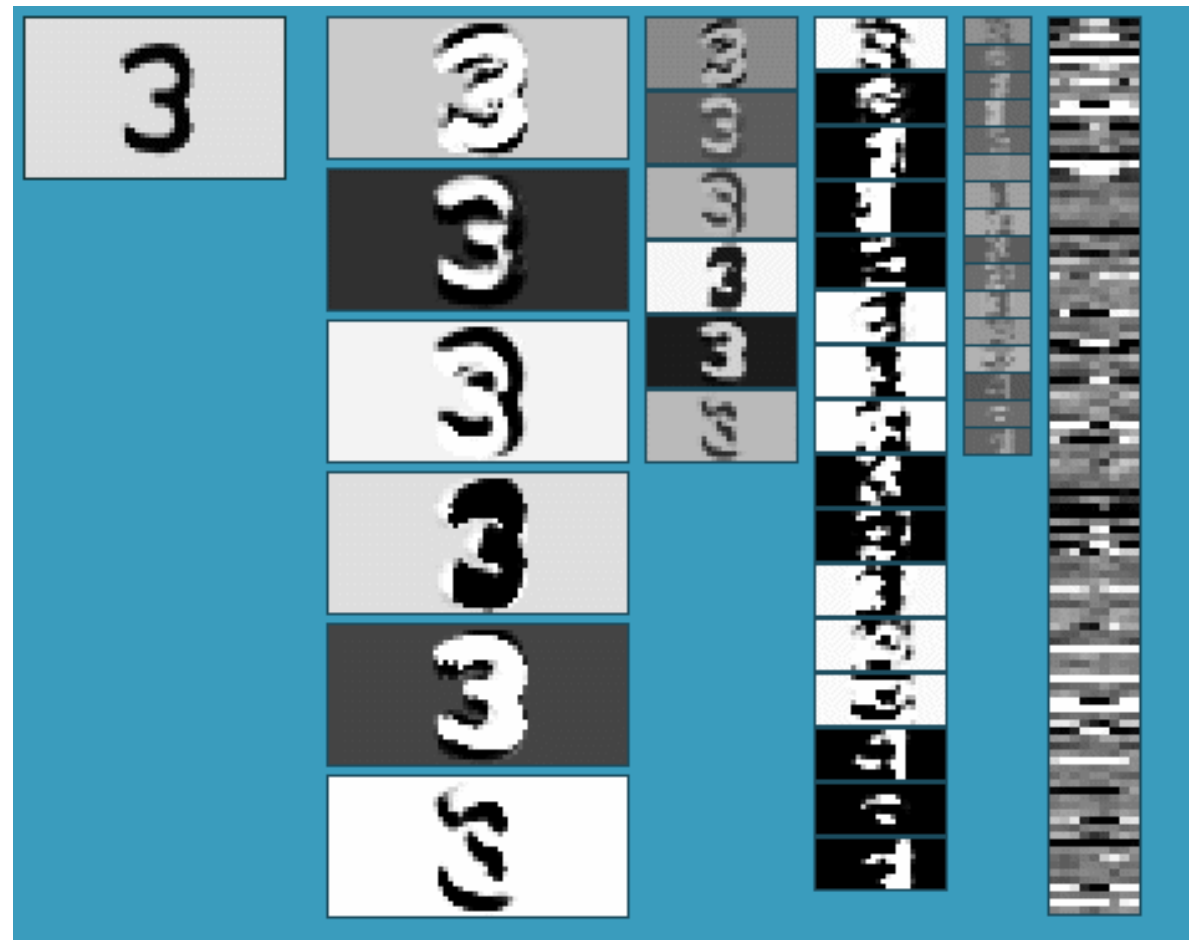


# The Multistage Hubel-Wiesel Architecture

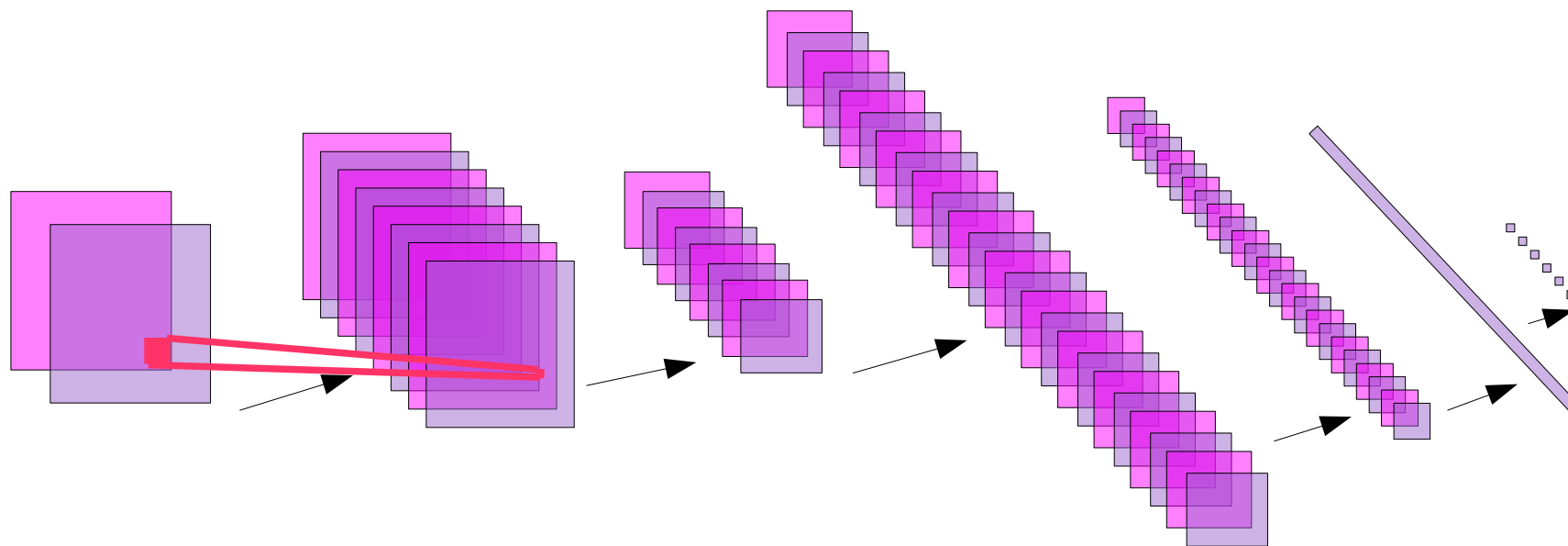
## Building a complete artificial vision system:

- ▶ 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?**



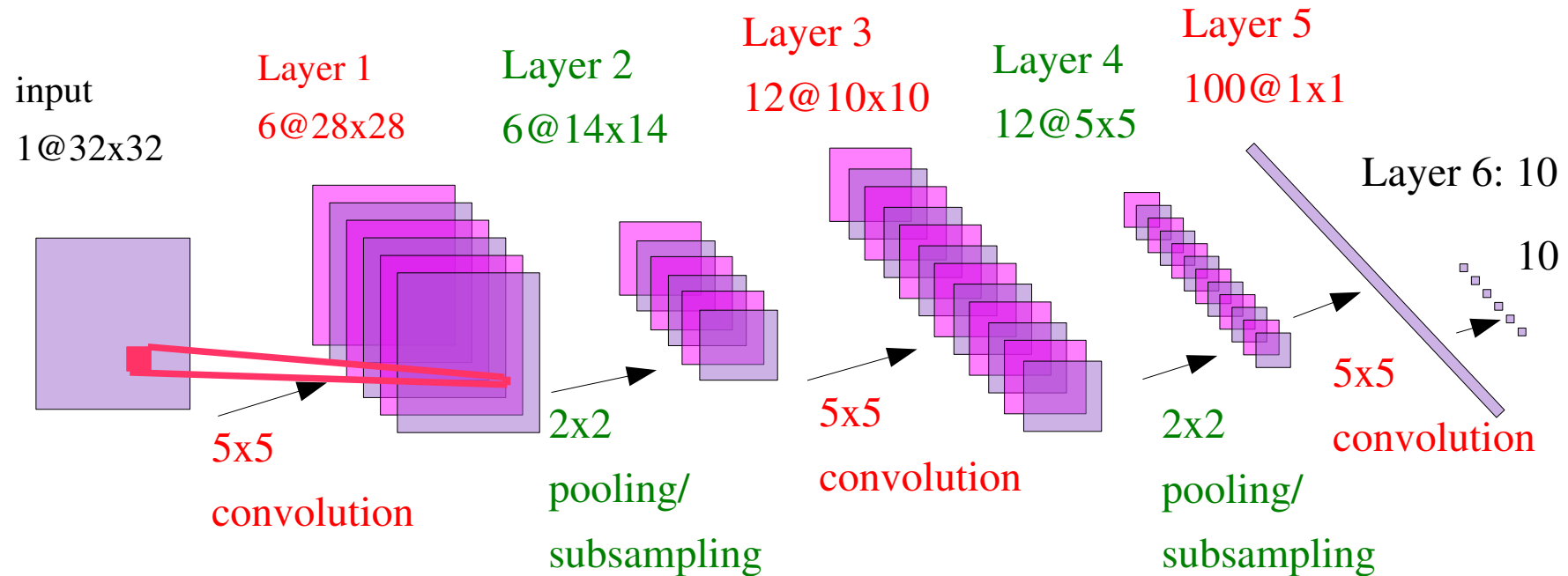
# Getting Inspiration from Biology: Convolutional Network



- **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.**

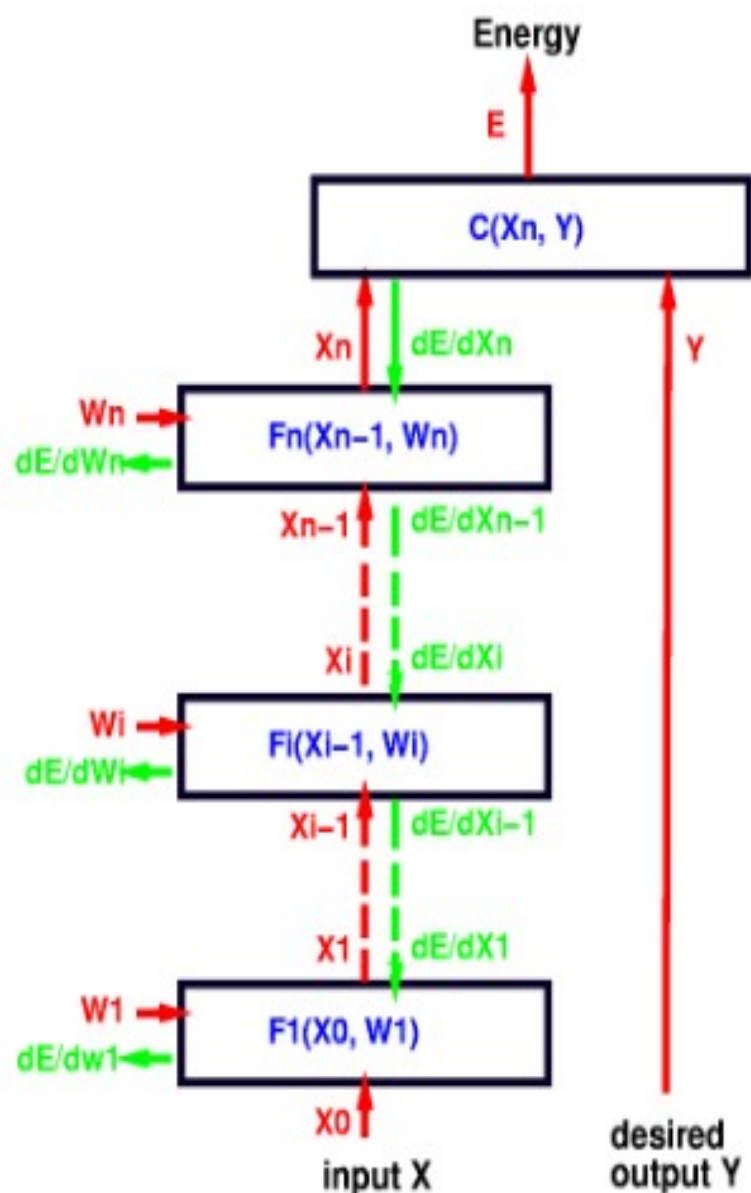


# Convolutional Net Architecture



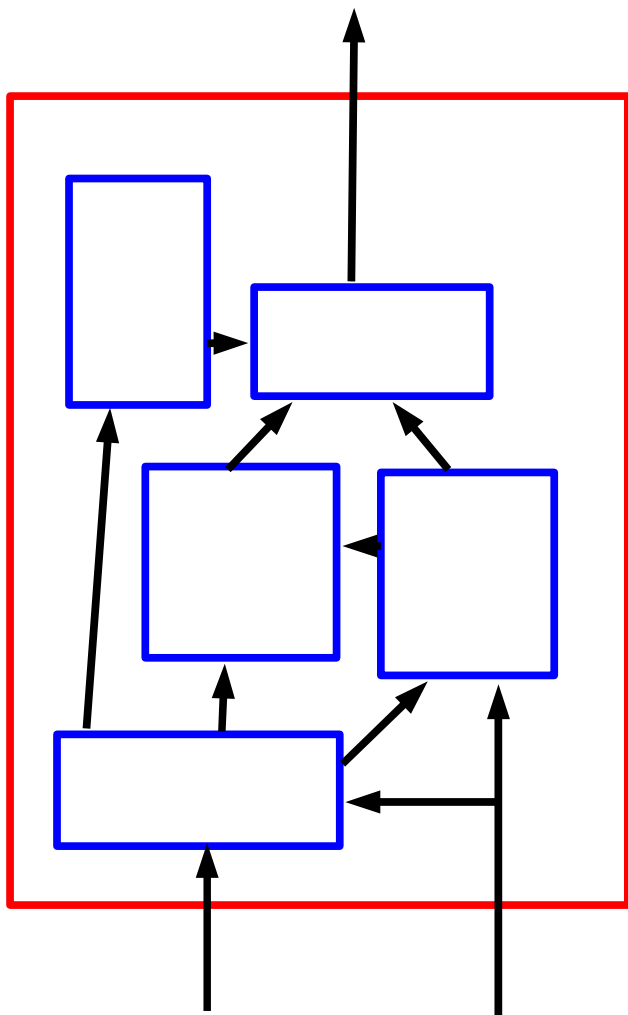
- **Convolutional net for handwriting recognition** (400,000 synapses)
- **Convolutional layers** (simple cells): all units in a feature plane share the same weights
- **Pooling/subsampling layers** (complex cells): for invariance to small distortions.
- **Supervised gradient-descent learning using back-propagation**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

# Back-propagation: deep supervised gradient-based learning



- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the  $\frac{\partial E}{\partial W_i}$  for  $i \in [1, n]$ .

# Any Architecture works



- **Any connection is permissible**

- ▶ Networks with loops must be “unfolded in time”.

- **Any module is permissible**

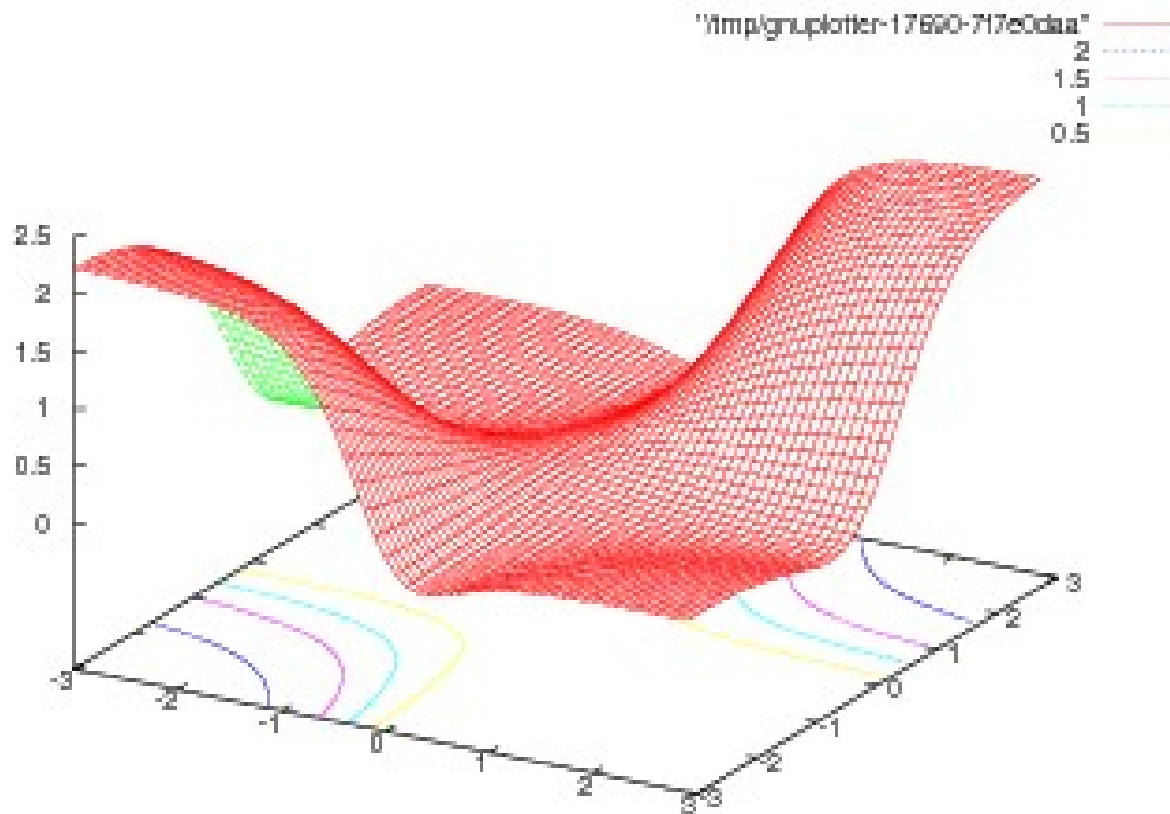
- ▶ As long as it is continuous and differentiable almost everywhere with respect to the parameters, and with respect to non-terminal inputs.

# Deep Supervised Learning is Hard

## Example: what is the loss function for the simplest 2-layer neural net ever

- Function: 1-1-1 neural net. Map 0.5 to 0.5 and -0.5 to -0.5 (identity function) with quadratic cost:


$$y = \tanh(W_1 \tanh(W_0 \cdot x)) \quad L = (0.5 - \tanh(W_1 \tanh(W_0 \cdot 0.5)))^2$$



# MNIST Handwritten Digit Dataset

3 6 8 1 7 9 6 6 4 1  
6 7 5 7 8 6 3 4 8 5  
2 1 7 9 7 1 2 8 4 5  
4 8 1 9 0 1 8 8 9 4  
7 6 1 8 6 4 1 5 6 0  
7 5 9 2 6 5 8 1 9 7  
2 2 2 2 2 3 4 4 8 0  
0 2 3 8 0 7 3 8 5 7  
0 1 4 6 4 6 0 2 4 3  
7 1 2 8 7 6 9 8 6 1

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9

 Handwritten Digit Dataset MNIST: 60,000 training samples, 10,000 test samples

# Results on MNIST Handwritten Digits

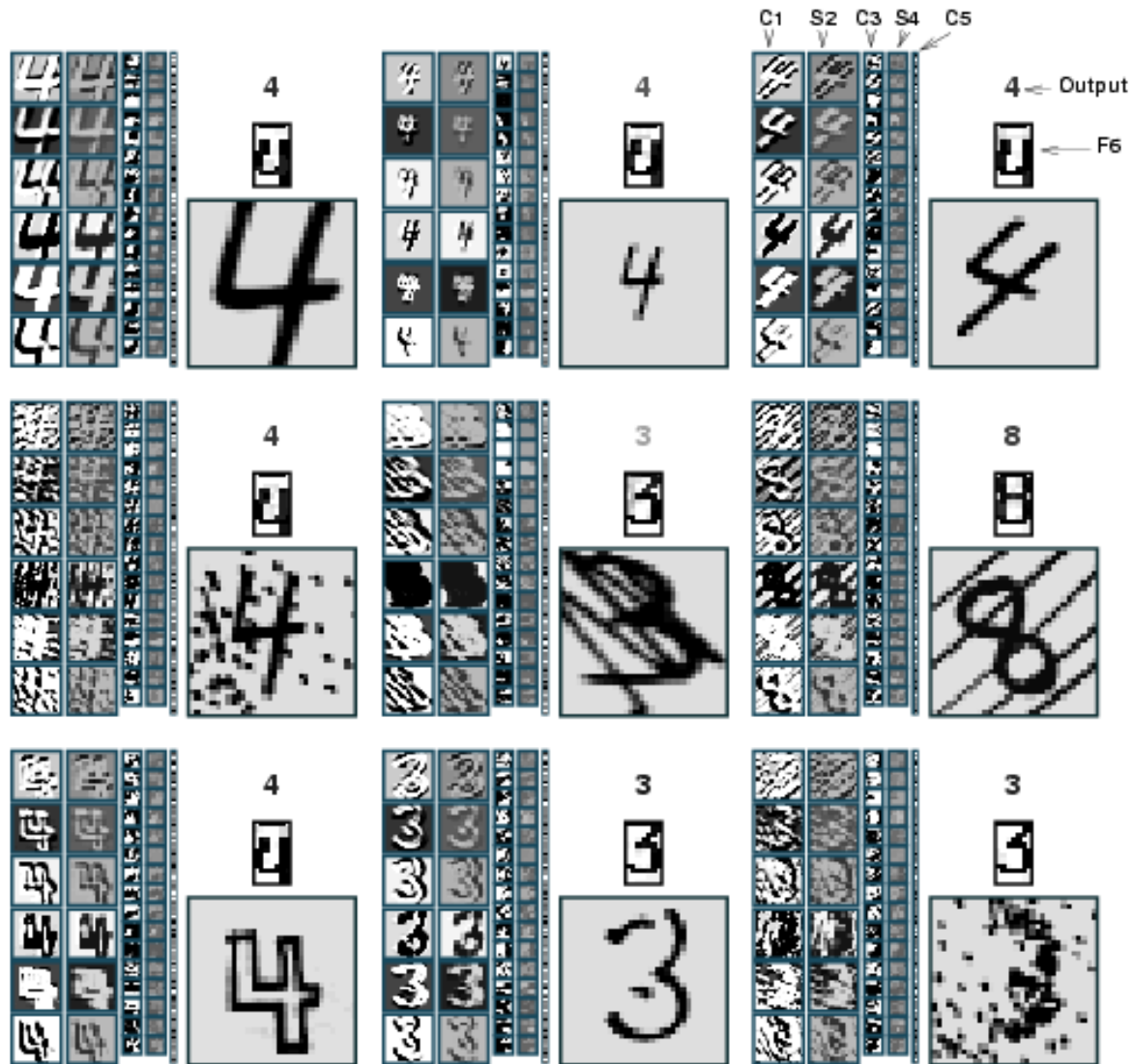
CLASSIFIER	DEFORMATION	PREPROCESSING	ERROR (%)	Reference
linear classifier (1-layer NN)		none	12.00	LeCun et al. 1998
linear classifier (1-layer NN)		deskewing	8.40	LeCun et al. 1998
pairwise linear classifier		deskewing	7.60	LeCun et al. 1998
K-nearest-neighbors, (L2)		none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, (L2)		deskewing	2.40	LeCun et al. 1998
K-nearest-neighbors, (L2)		deskew, clean, blur	1.80	Kenneth Wilder, U. Chicago
K-NN L3, 2 pixel jitter		deskew, clean, blur	1.22	Kenneth Wilder, U. Chicago
<b>K-NN, shape context matching</b>		<b>shape context feature</b>	<b>0.63</b>	<b>Belongie et al. IEEE PAMI 2002</b>
40 PCA + quadratic classifier		none	3.30	LeCun et al. 1998
1000 RBF + linear classifier		none	3.60	LeCun et al. 1998
K-NN, Tangent Distance		subsamp 16x16 pixels	1.10	LeCun et al. 1998
SVM, Gaussian Kernel		none	1.40	
SVM deg 4 polynomial		deskewing	1.10	LeCun et al. 1998
Reduced Set SVM deg 5 poly		deskewing	1.00	LeCun et al. 1998
Virtual SVM deg-9 poly	Affine	none	0.80	LeCun et al. 1998
V-SVM, 2-pixel jittered		none	0.68	DeCoste and Scholkopf, MLJ 2002
<b>V-SVM, 2-pixel jittered</b>		<b>deskewing</b>	<b>0.56</b>	<b>DeCoste and Scholkopf, MLJ 2002</b>
2-layer NN, 300 HU, MSE		none	4.70	LeCun et al. 1998
2-layer NN, 300 HU, MSE,	Affine	none	3.60	LeCun et al. 1998
2-layer NN, 300 HU		deskewing	1.60	LeCun et al. 1998
3-layer NN, 500+150 HU		none	2.95	LeCun et al. 1998
3-layer NN, 500+150 HU	Affine	none	2.45	LeCun et al. 1998
3-layer NN, 500+300 HU, CE, reg		none	1.53	Hinton, unpublished, 2005
2-layer NN, 800 HU, CE		none	1.60	Simard et al., ICDAR 2003
2-layer NN, 800 HU, CE	Affine	none	1.10	Simard et al., ICDAR 2003
2-layer NN, 800 HU, MSE	Elastic	none	0.90	Simard et al., ICDAR 2003
<b>2-layer NN, 800 HU, CE</b>	<b>Elastic</b>	<b>none</b>	<b>0.70</b>	<b>Simard et al., ICDAR 2003</b>
Convolutional net LeNet-1		subsamp 16x16 pixels	1.70	LeCun et al. 1998
Convolutional net LeNet-4		none	1.10	LeCun et al. 1998
Convolutional net LeNet-5,		none	0.95	LeCun et al. 1998
<b>Conv. net LeNet-5,</b>	<b>Affine</b>	<b>none</b>	<b>0.80</b>	<b>LeCun et al. 1998</b>
Boosted LeNet-4	Affine	none	0.70	LeCun et al. 1998
<b>Conv. net, CE</b>	<b>Affine</b>	<b>none</b>	<b>0.60</b>	<b>Simard et al., ICDAR 2003</b>
<b>Conv net, CE</b>	<b>Elastic</b>	<b>none</b>	<b>0.40</b>	<b>Simard et al., ICDAR 2003</b>

# Some Results on MNIST (from raw images: no preprocessing)

CLASSIFIER	DEFORMATION	ERROR	Reference
<b>Knowledge-free methods</b>			
(no permutation of the pixels would make no difference)			
2-layer NN, 800 HU, CE		1.60	Simard et al., ICDAR 2003
3-layer NN, 500+300 HU, CE, reg		1.53	Hinton, in press, 2005
SVM, Gaussian Kernel		1.40	Cortes 92 + Many others
<b>Convolutional nets</b>			
Convolutional net LeNet-5,		0.80	Ranzato et al. NIPS 2006
Convolutional net LeNet-6,		0.70	Ranzato et al. NIPS 2006
<b>Training set augmented with Affine Distortions</b>			
2-layer NN, 800 HU, CE	Affine	1.10	Simard et al., ICDAR 2003
Virtual SVM deg-9 poly	Affine	0.80	Scholkopf
Convolutional net, CE	Affine	0.60	Simard et al., ICDAR 2003
<b>Training set augmented with Elastic Distortions</b>			
2-layer NN, 800 HU, CE	Elastic	0.70	Simard et al., ICDAR 2003
Convolutional net, CE	Elastic	0.40	Simard et al., ICDAR 2003

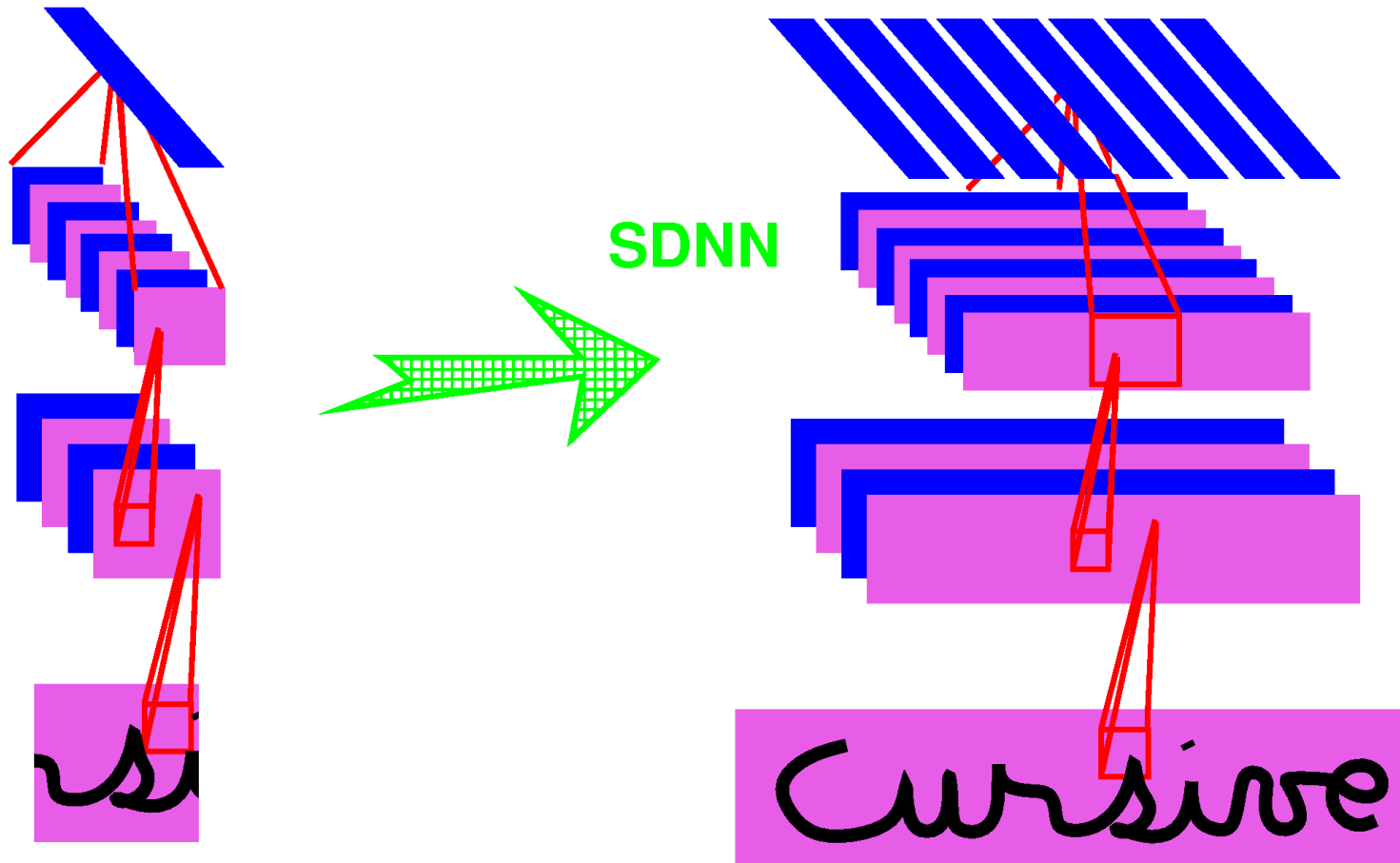
Note: some groups have obtained good results with various amounts of preprocessing such as deskewing (e.g. 0.56% using an SVM with smart kernels [deCoste and Schoelkopf]) hand-designed feature representations (e.g. 0.63% with “shape context” and nearest neighbor [Belongie])

# Invariance and Robustness to Noise





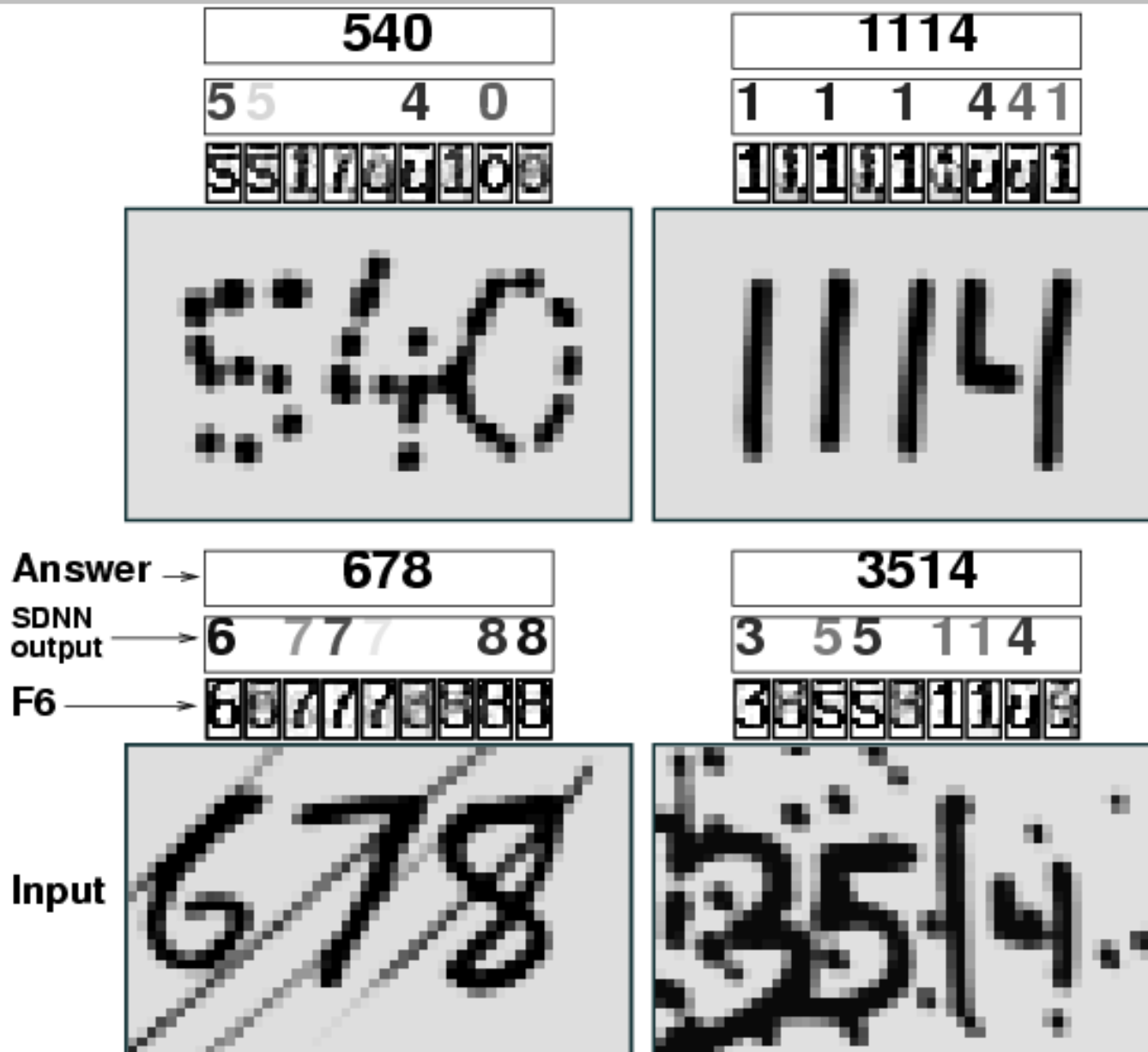
# Recognizing Multiple Characters with Replicated Nets



# Recognizing Multiple Characters with Replicated Nets

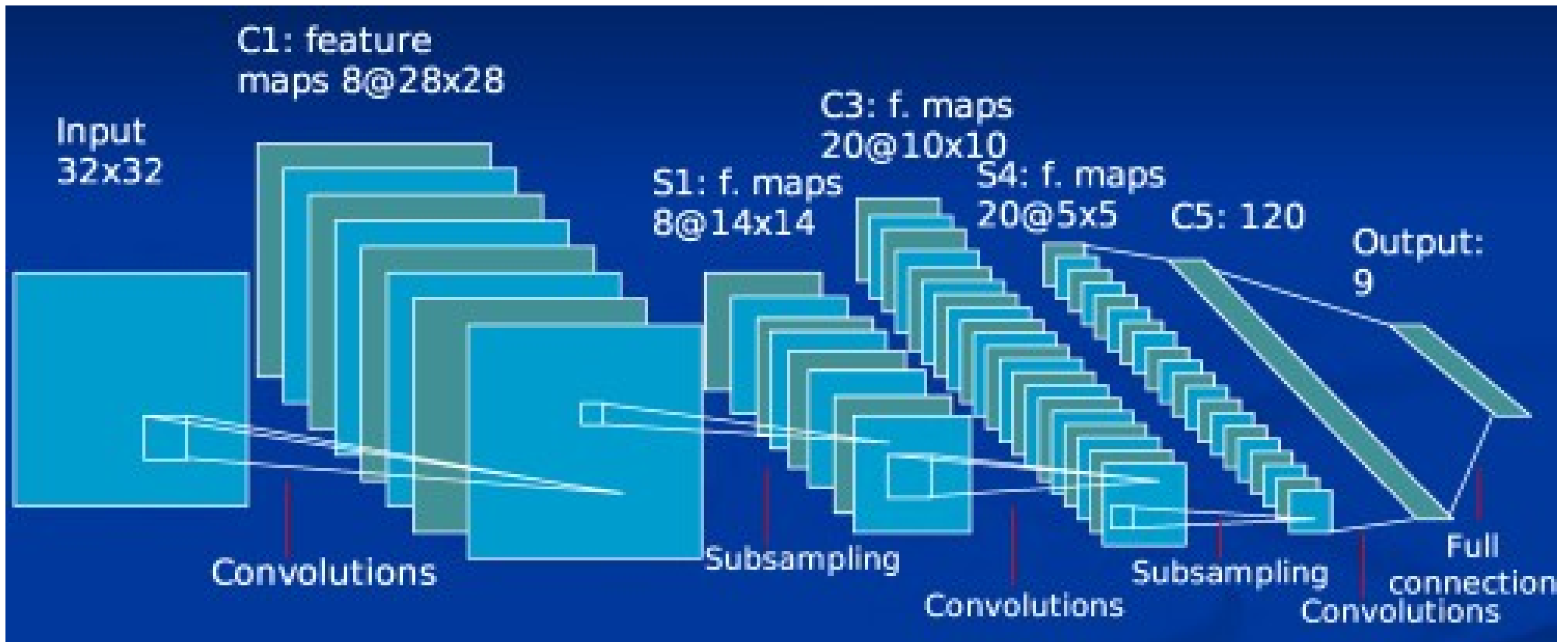


# Handwriting Recognition



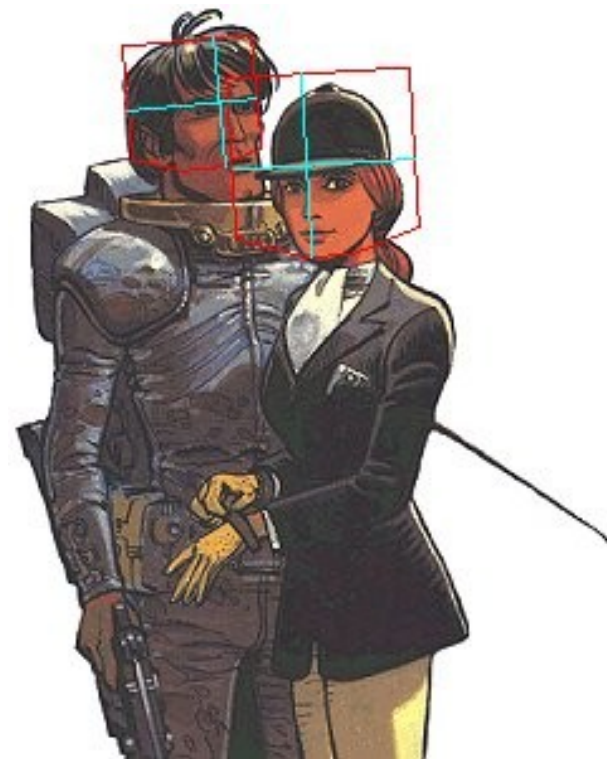
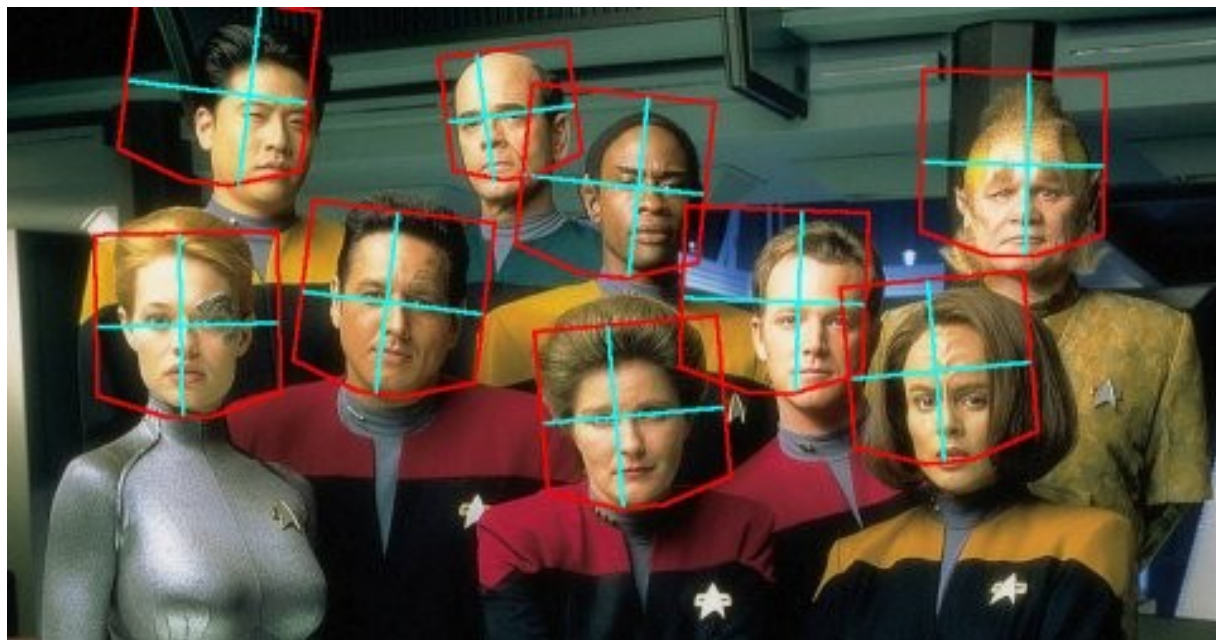
# Face Detection and Pose Estimation with Convolutional Nets

- **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.
- **2<sup>nd</sup> phase:** half of the initial negative set was replaced by false positives of the initial version of the detector .

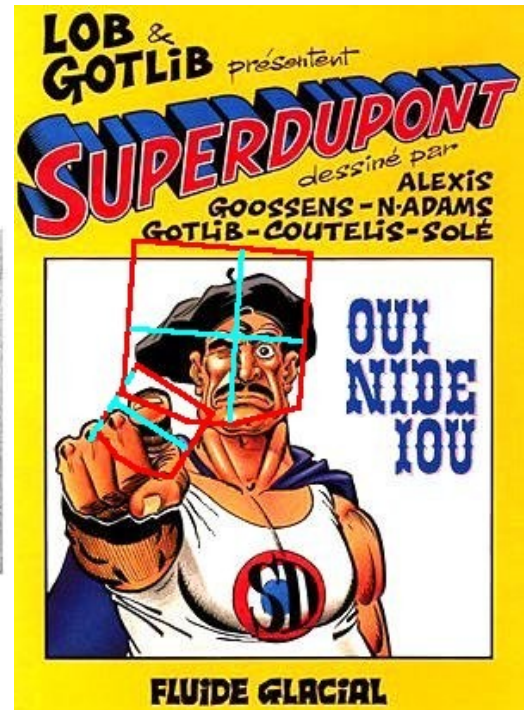
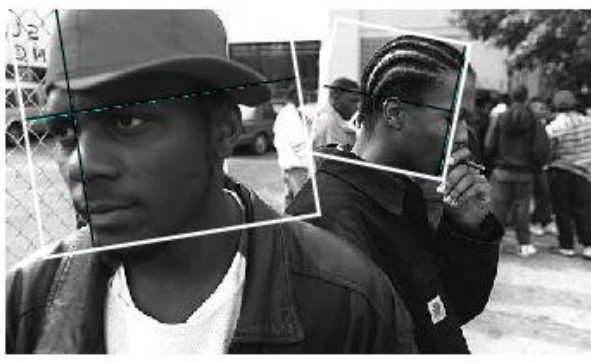
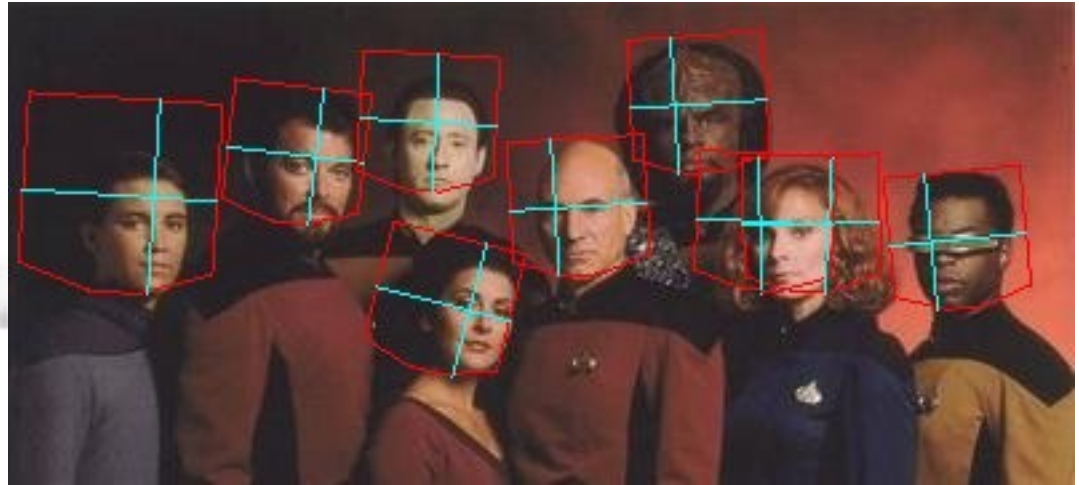
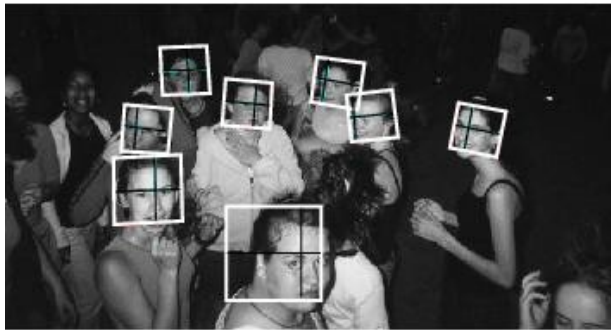


# Face Detection: Results

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



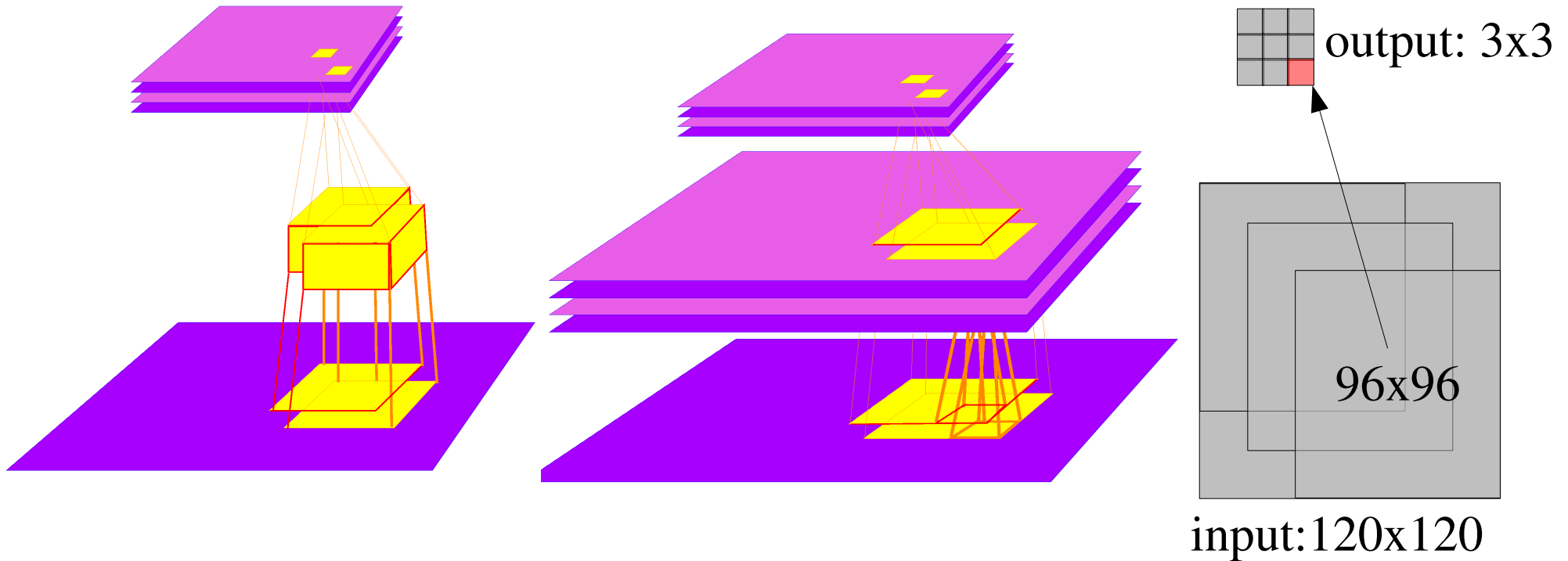
# Face Detection and Pose Estimation: Results



# Face Detection with a Convolutional Net



# Applying a ConvNet on Sliding Windows is Very Cheap!

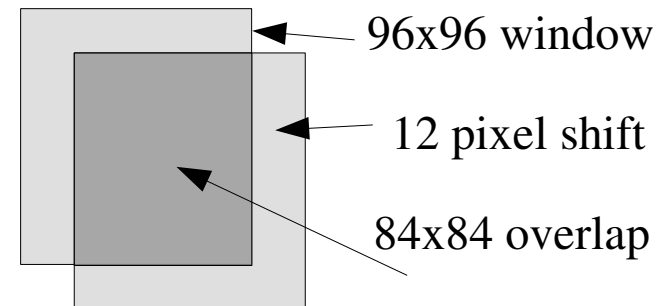
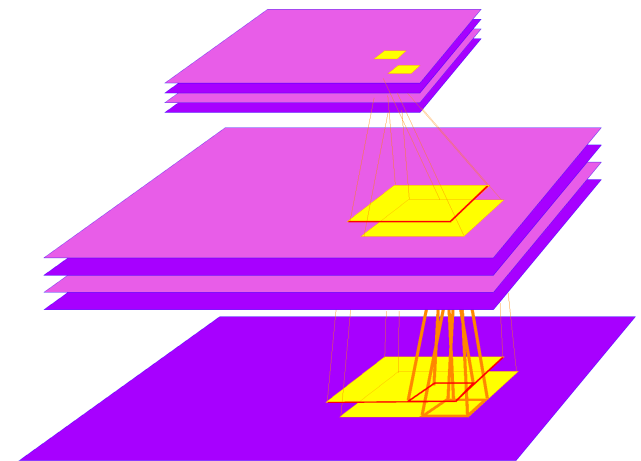
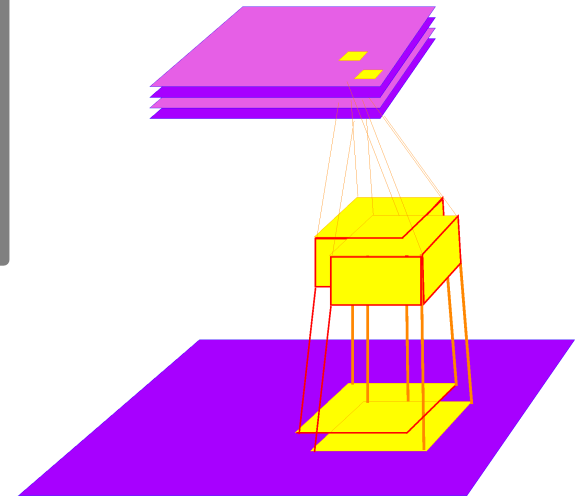


- Traditional Detectors/Classifiers must be applied to every location on a large input image, at multiple scales.
- Convolutional nets can be replicated over large images very cheaply.
- The network is applied to multiple scales spaced by 1.5.



# Building a Detector/Recognizer: Replicated Convolutional Nets

- Computational cost for replicated convolutional net:
  - 96x96 -> 4.6 million multiply-accumulate operations
  - 120x120 -> 8.3 million multiply-accumulate operations
  - 240x240 -> 47.5 million multiply-accumulate operations
  - 480x480 -> 232 million multiply-accumulate operations
- Computational cost for a non-convolutional detector of the same size, applied every 12 pixels:
  - 96x96 -> 4.6 million multiply-accumulate operations
  - 120x120 -> 42.0 million multiply-accumulate operations
  - 240x240 -> 788.0 million multiply-accumulate operations
  - 480x480 -> 5,083 million multiply-accumulate operations



# 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

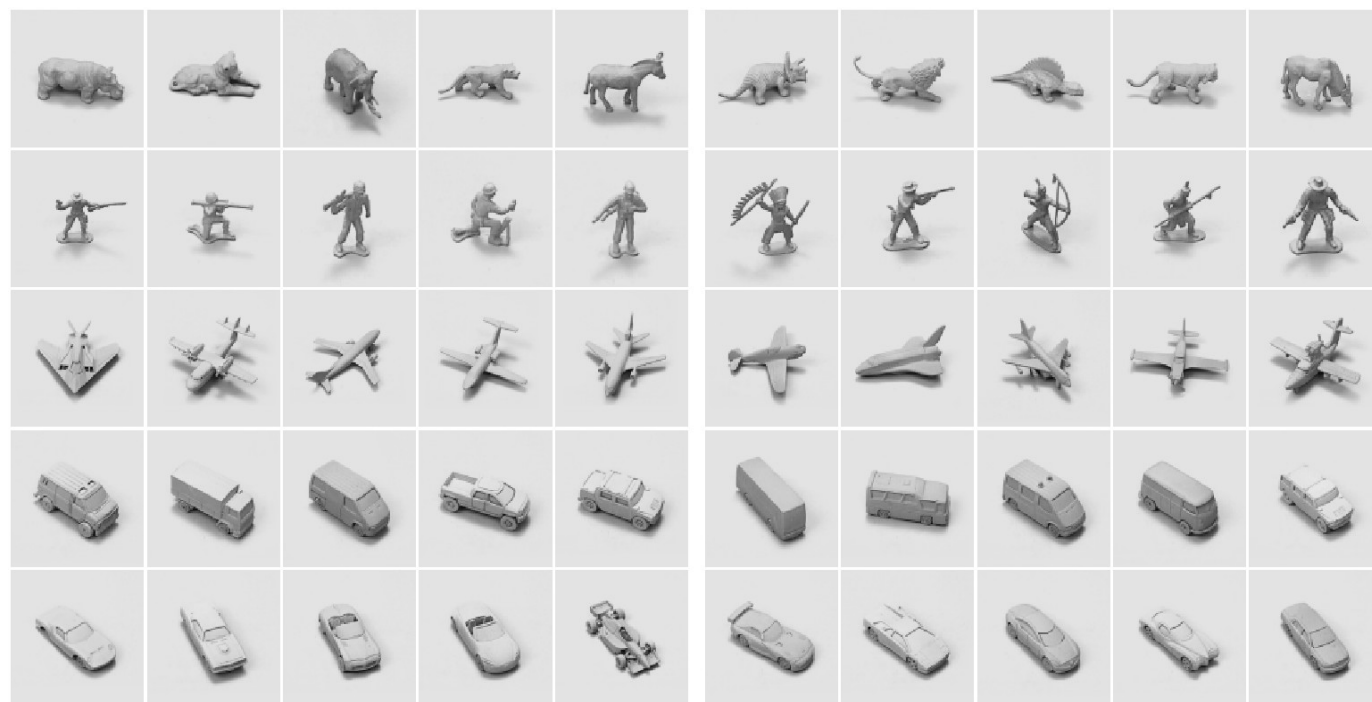
**6 illuminations**

on/off combinations of 4 lights

**2 cameras (stereo)**

7.5 cm apart

40 cm from the object

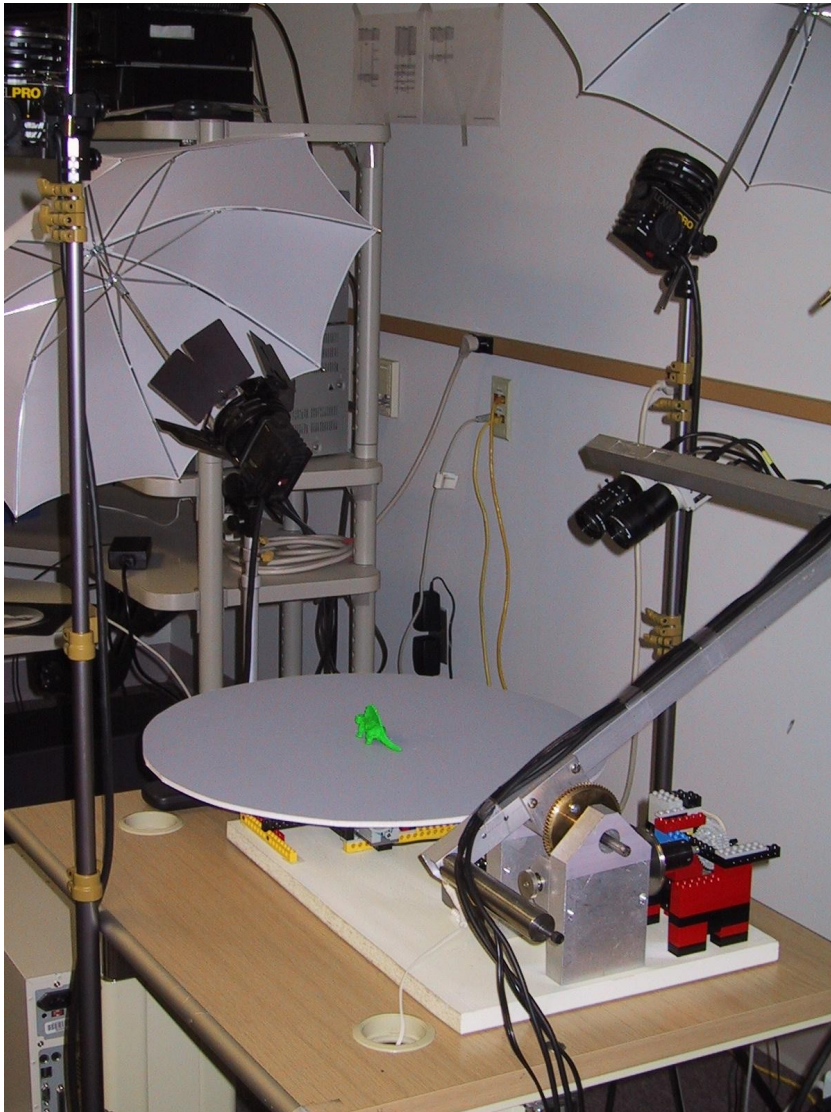


Training instances

Test instances

# Data Collection, Sample Generation

## Image capture setup



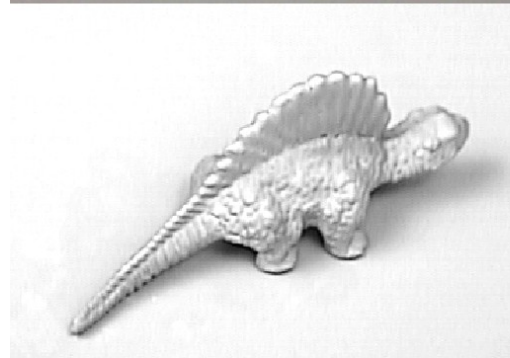
Objects are painted green so that:

- all features other than shape are removed
- objects can be segmented, transformed, and composited onto various backgrounds

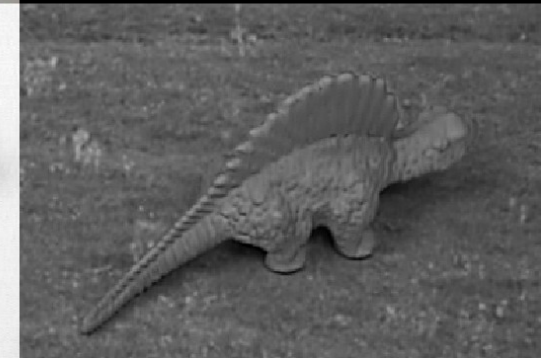
Original image



Object mask

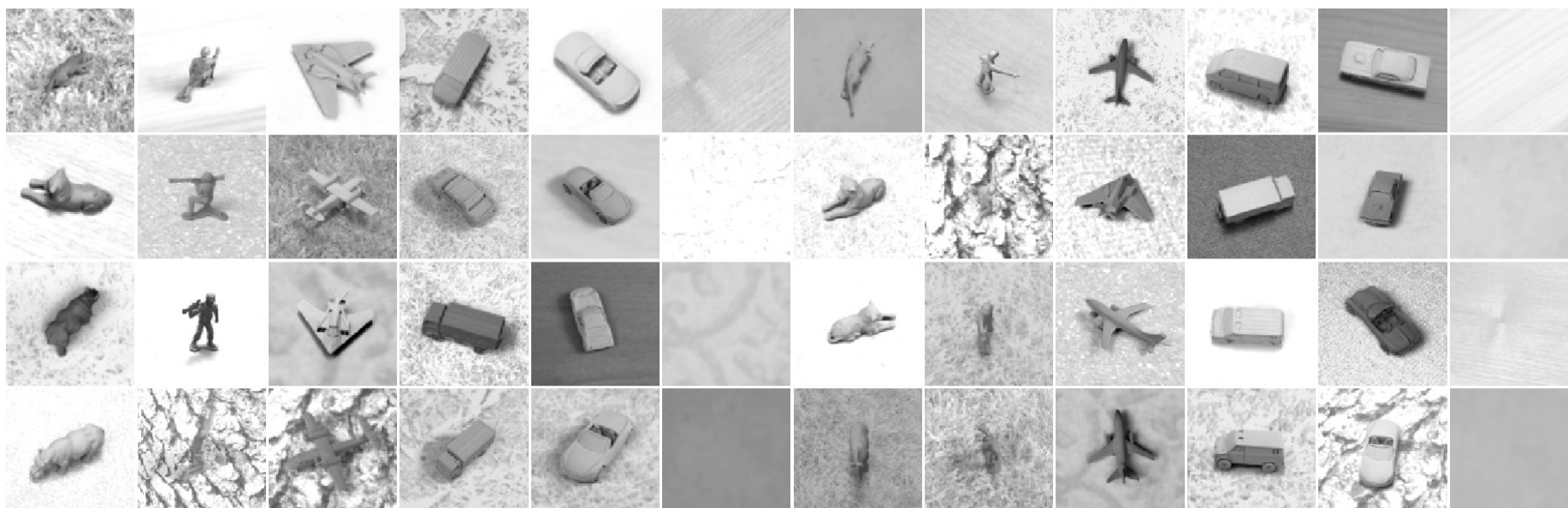


Shadow factor



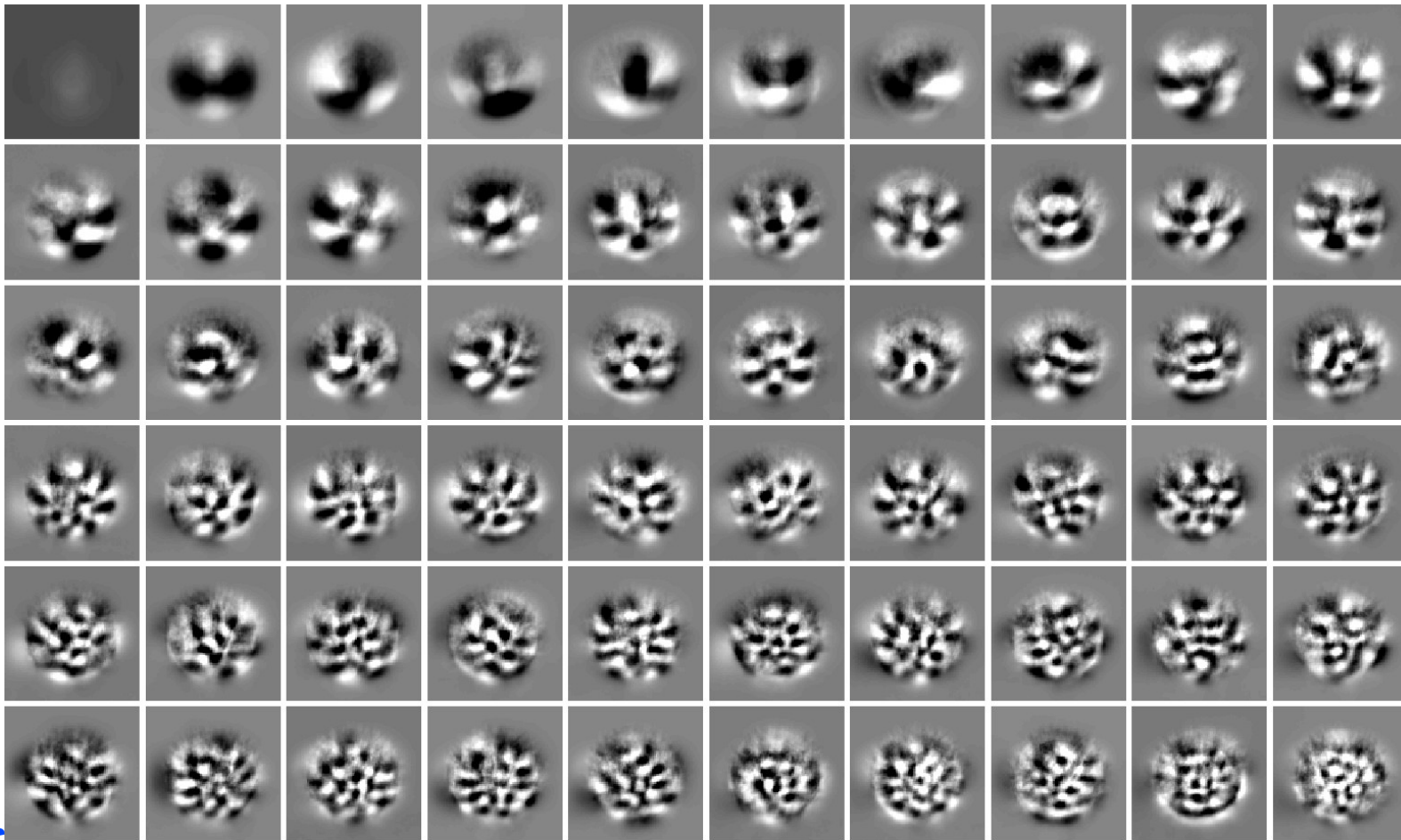
Composite image

# Textured and Cluttered Datasets



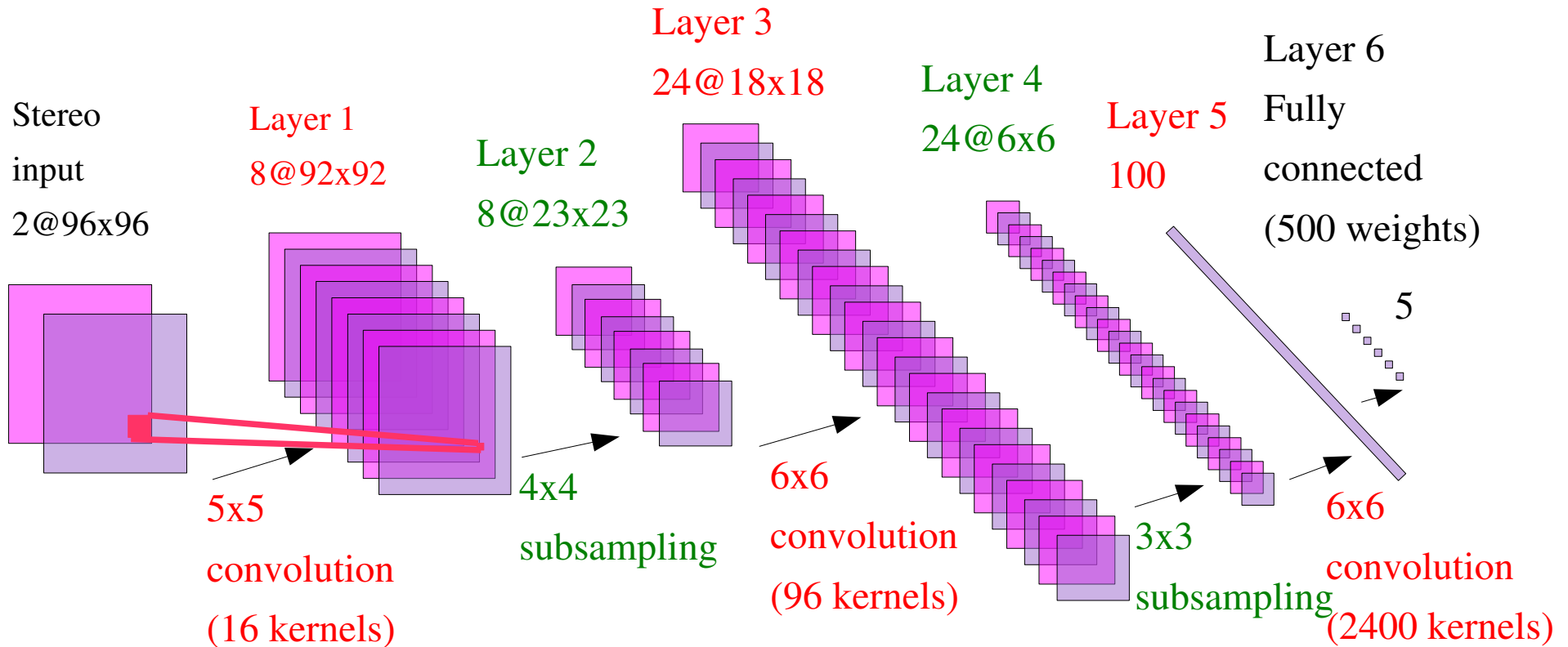
# Experiment 1: Normalized-Uniform Set: Representations

- 1 - Raw Stereo Input: 2 images 96x96 pixels **input dim. = 18432**
- 2 - Raw Monocular Input: 1 image, 96x96 pixels **input dim. = 9216**
- 3 - Subsampled Mono Input: 1 image, 32x32 pixels **input dim = 1024**
- 4 - PCA-95 (EigenToys): First 95 Principal Components **input dim. = 95**



First 60 eigenvectors (EigenToys)

# Convolutional Network



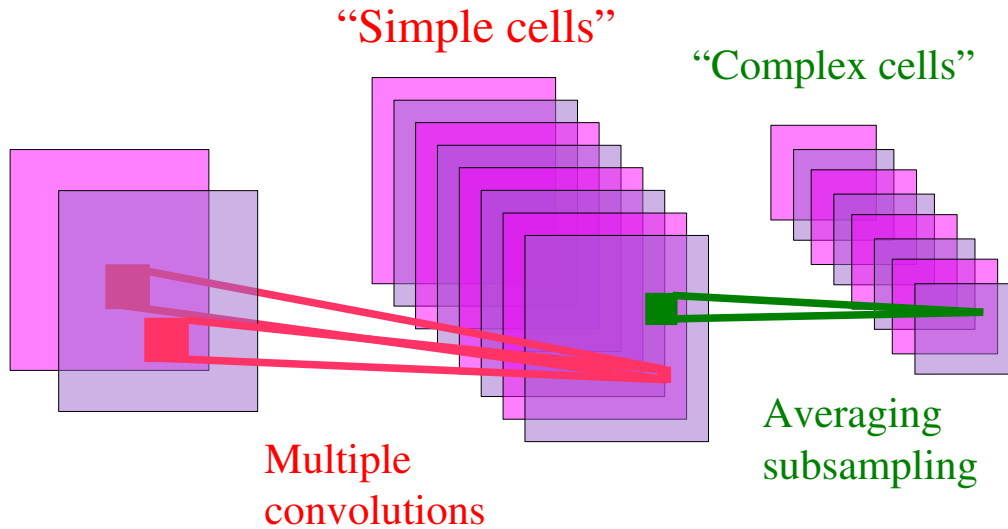
90,857 free parameters, 3,901,162 connections.

The architecture alternates **convolutional layers** (feature detectors) and **subsampling layers** (local feature pooling for invariance to small distortions).

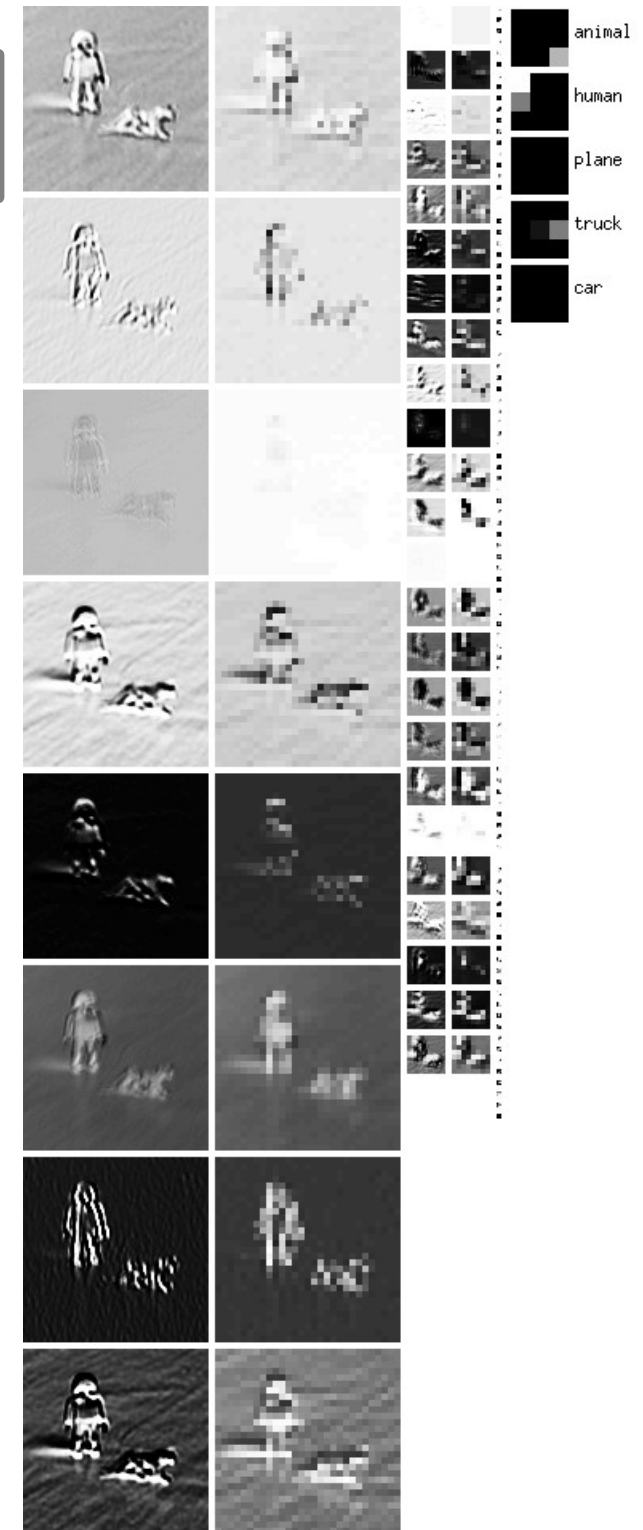
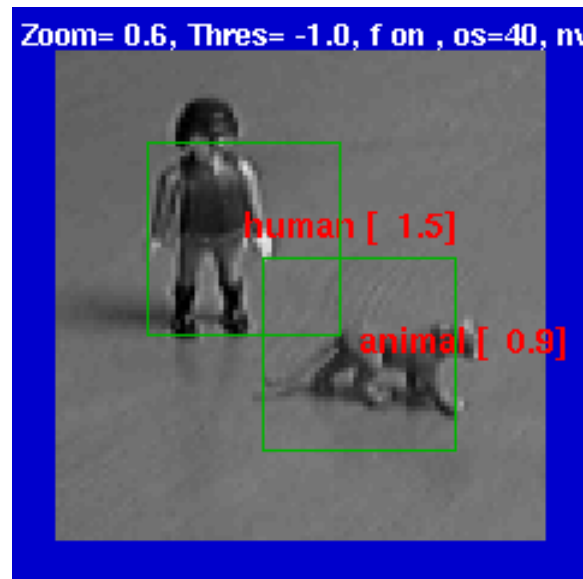
**The entire network is trained end-to-end** (all the layers are trained simultaneously).

A gradient-based algorithm is used to minimize a supervised loss function.

# Alternated Convolutions and Subsampling

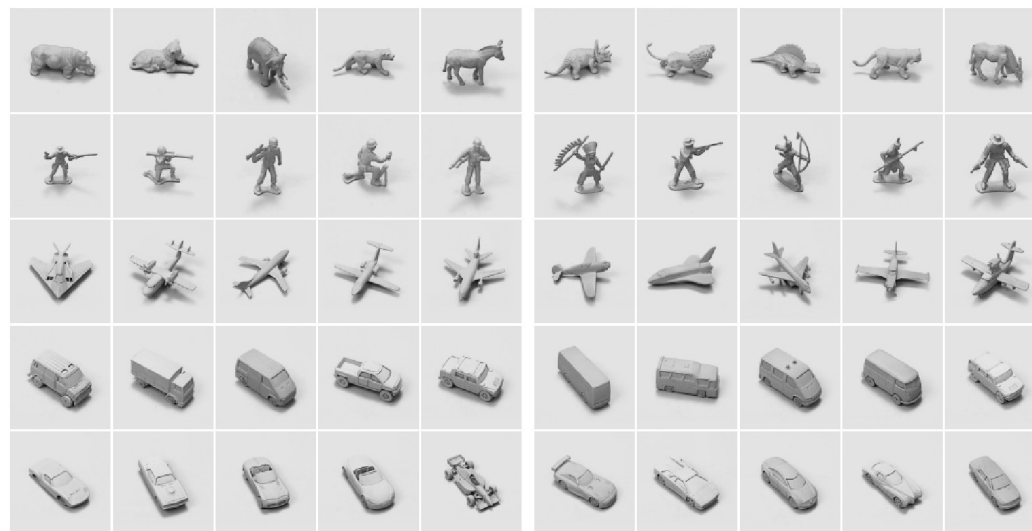


- Local features are extracted everywhere.
- averaging/subsampling layer builds robustness to variations in feature locations.
- Hubel/Wiesel'62, Fukushima'71, LeCun'89, Riesenhuber & Poggio'02, Ullman'02,....



# 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**



## Normalized-Uniform Set: Learning Times

	SVM	Conv Net				SVM/Conv
test error	11.6%	10.4%	6.2%	5.8%	6.2%	5.9%
train time (min*GHz)	480	64	384	640	3,200	50+
test time per sample (sec*GHz)	0.95	0.03				0.04+
#SV	28%					28%
parameters	$\sigma=2,000$ $C=40$					dim=80 $\sigma=5$ $C=0.01$

SVM: using a parallel implementation by Graf, Durdanovic, and Cosatto (NEC Labs)

Chop off the last layer of the convolutional net and train an SVM on it

# Jittered-Cluttered Dataset



## ■ Jittered-Cluttered Dataset:

■ **291,600** stereo 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>

# Jittered-Cluttered Dataset

	SVM	Conv Net			SVM/Conv
test error	43.3%	16.38%	7.5%	7.2%	5.9%
train time (min*GHz)	10,944	420	2,100	5,880	330+
test time per sample (sec*GHz)	2.2	0.04			0.06+
#SV	5%				2%
parameters	$\sigma=10^4$ $C=40$				dim=100 $\sigma=5$ $C=1$

**OUCH!**

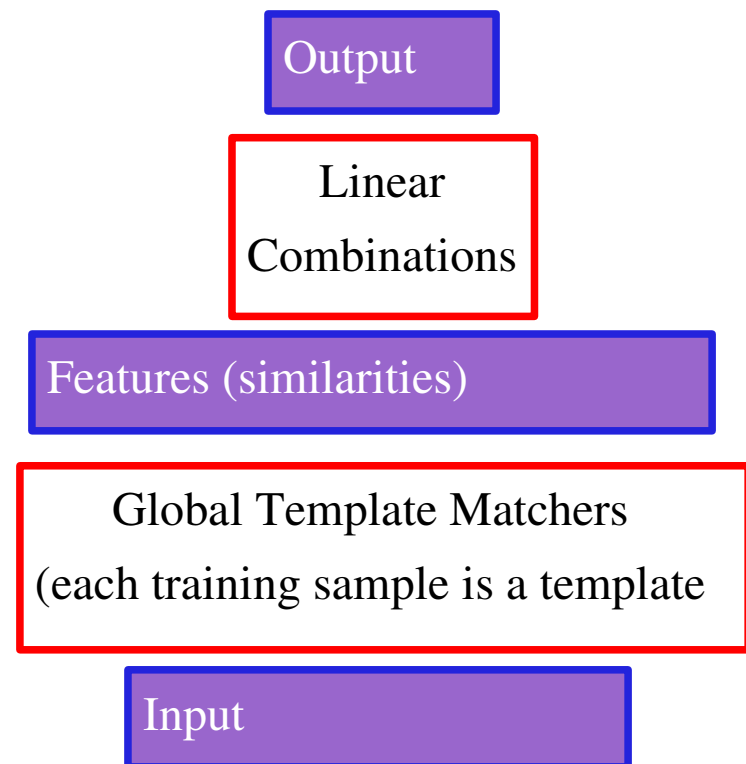
The convex loss, VC bounds  
and representers theorems  
don't seem to help

Chop off the last layer,  
and train an SVM on it  
it works!

# What's wrong with K-NN and SVMs?

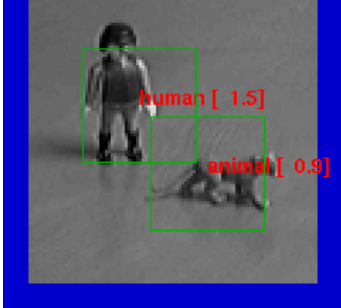
- K-NN and SVM with Gaussian kernels are based on **matching global templates**
- Both are “shallow” architectures
- There is now way to learn invariant recognition tasks with such naïve architectures (unless we use an impractically large number of templates).

- The number of necessary templates grows **exponentially** with the number of dimensions of variations.
- Global templates are in trouble when the variations include: category, instance shape, configuration (for articulated object), position, azimuth, elevation, scale, illumination, texture, albedo, in-plane rotation, background luminance, background texture, background clutter, .....

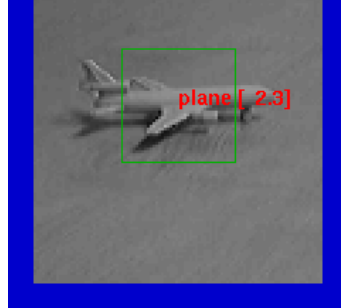


# Examples (Monocular Mode)

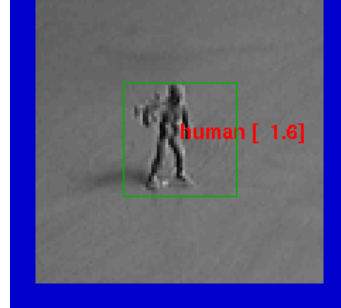
Zoom= 0.6, Thres= -1.0, f on , os=40, nv



Zoom= 0.6, Thres= -1.0, f on , os=40, nv



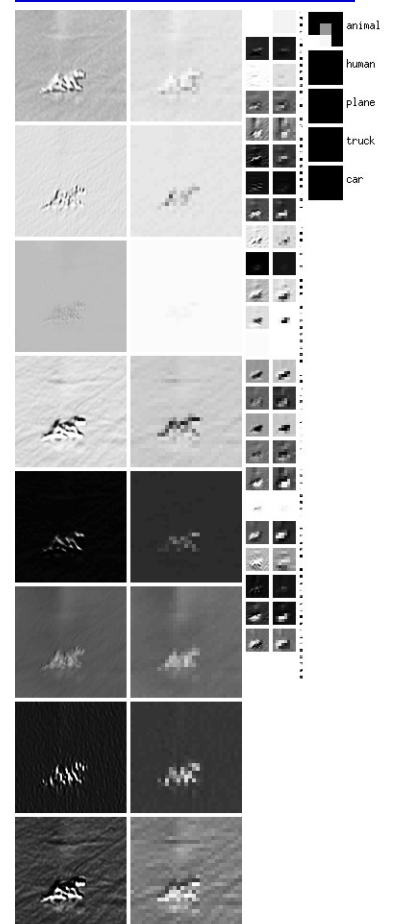
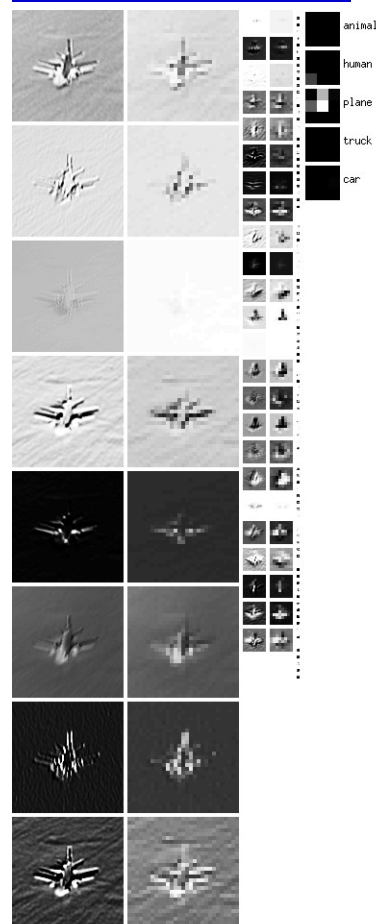
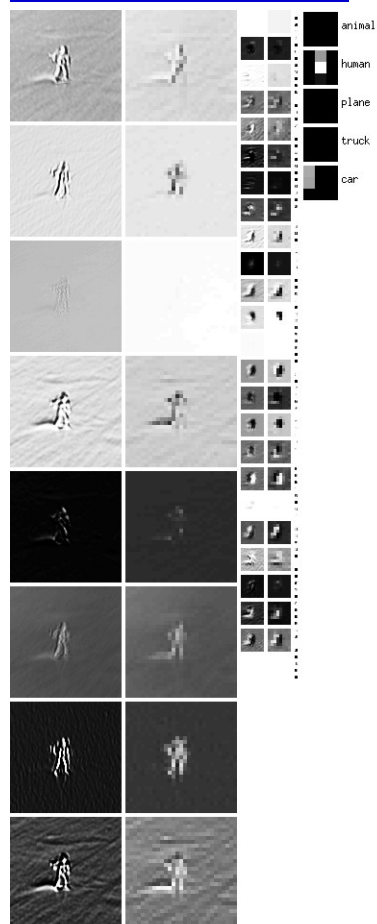
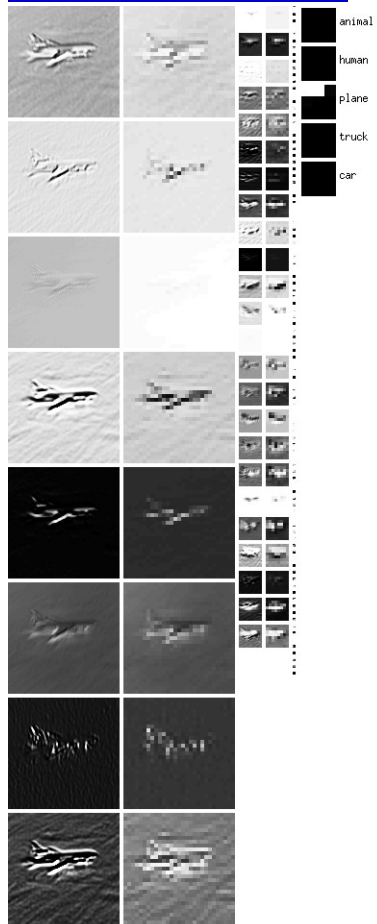
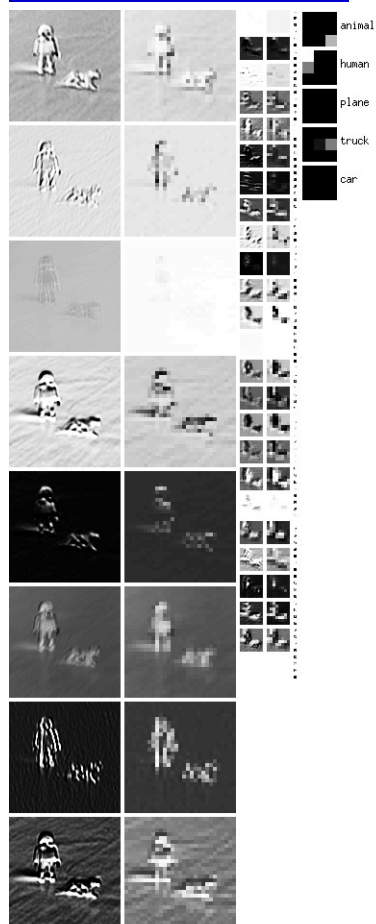
Zoom= 0.6, Thres= -1.0, f on , os=40, nv



Zoom= 0.6, Thres= -1.0, f on , os=40, nv



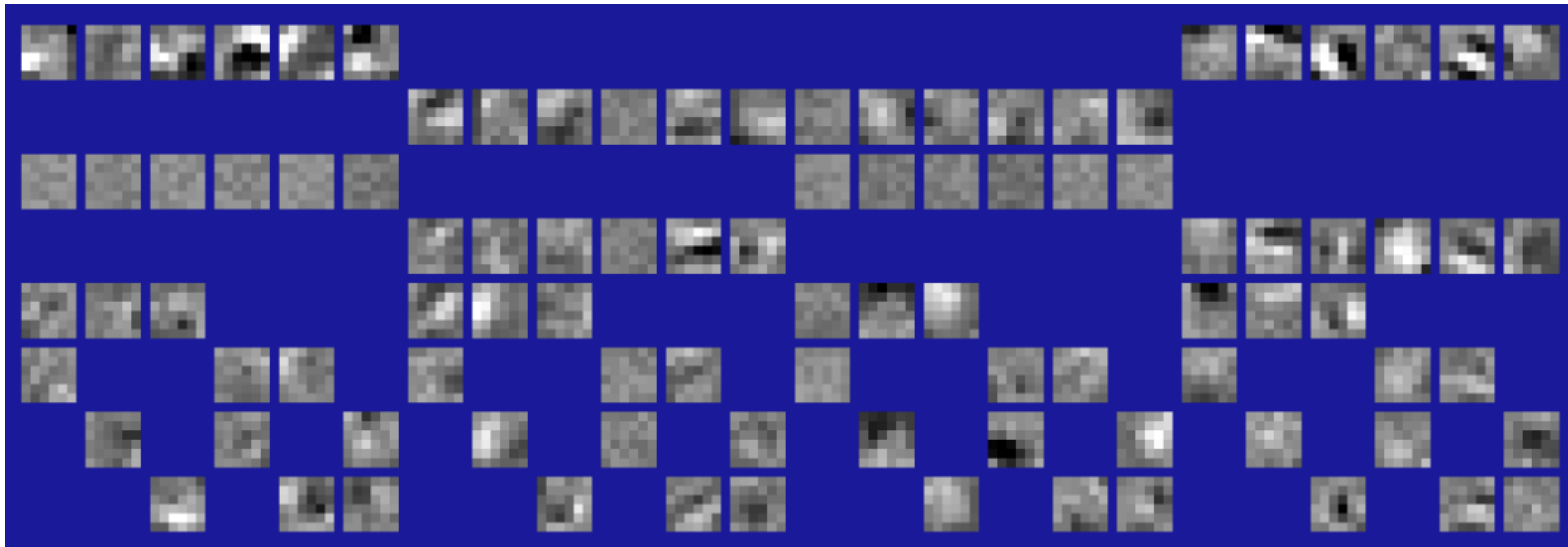
Zoom= 0.6, Thres= 0.5, f on , os=40, nv



# Learned Features

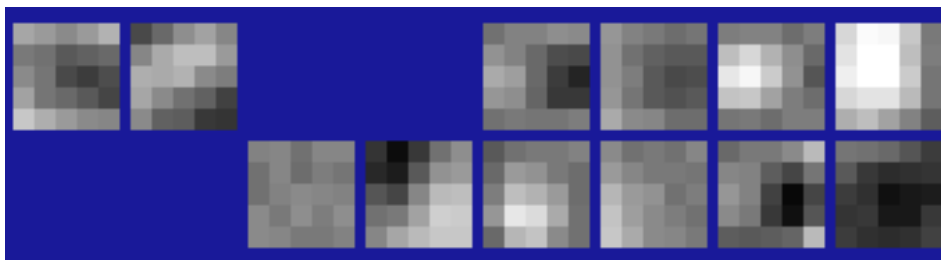
Layer 3

Layer 2

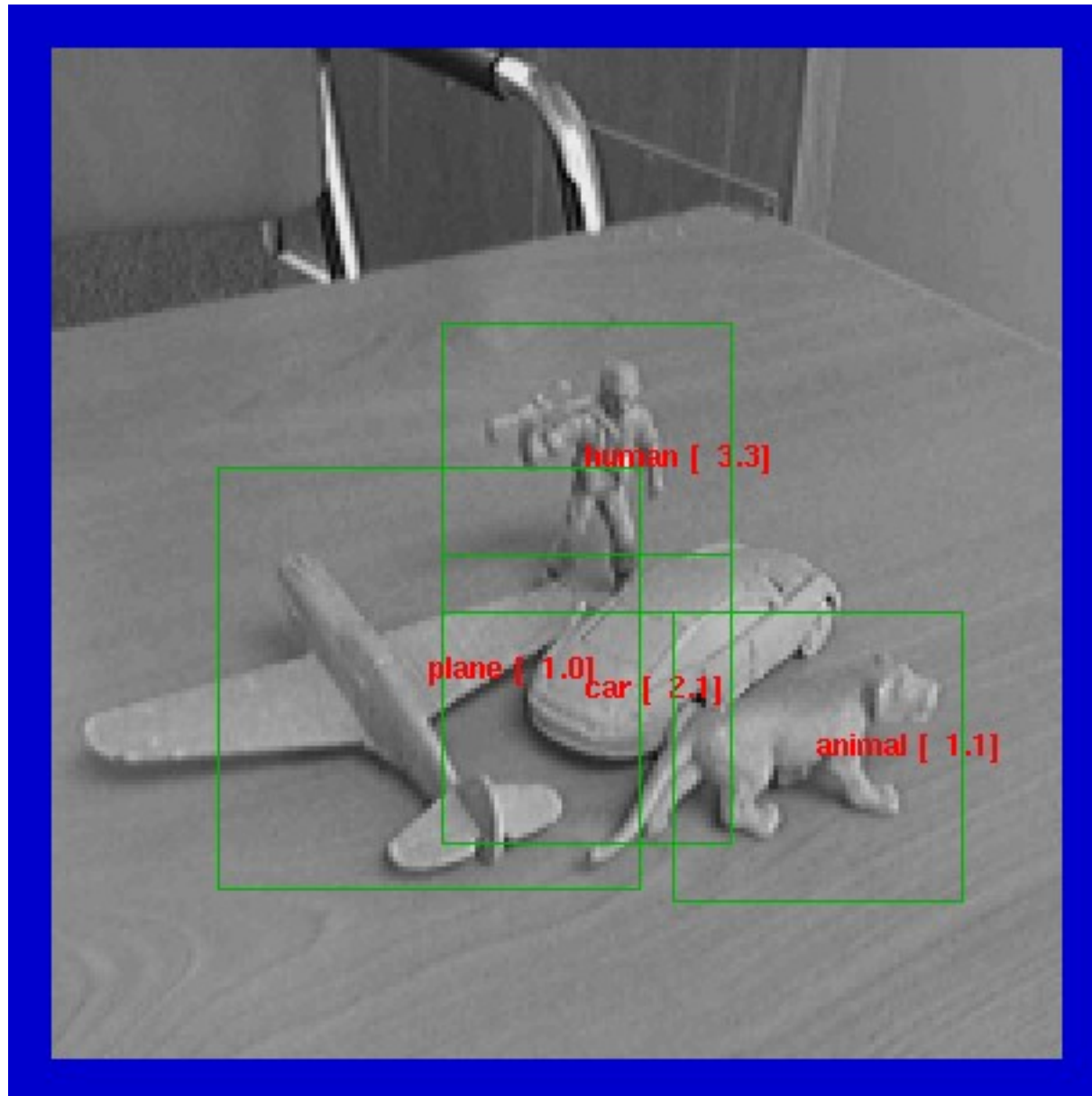


Layer 1

Input

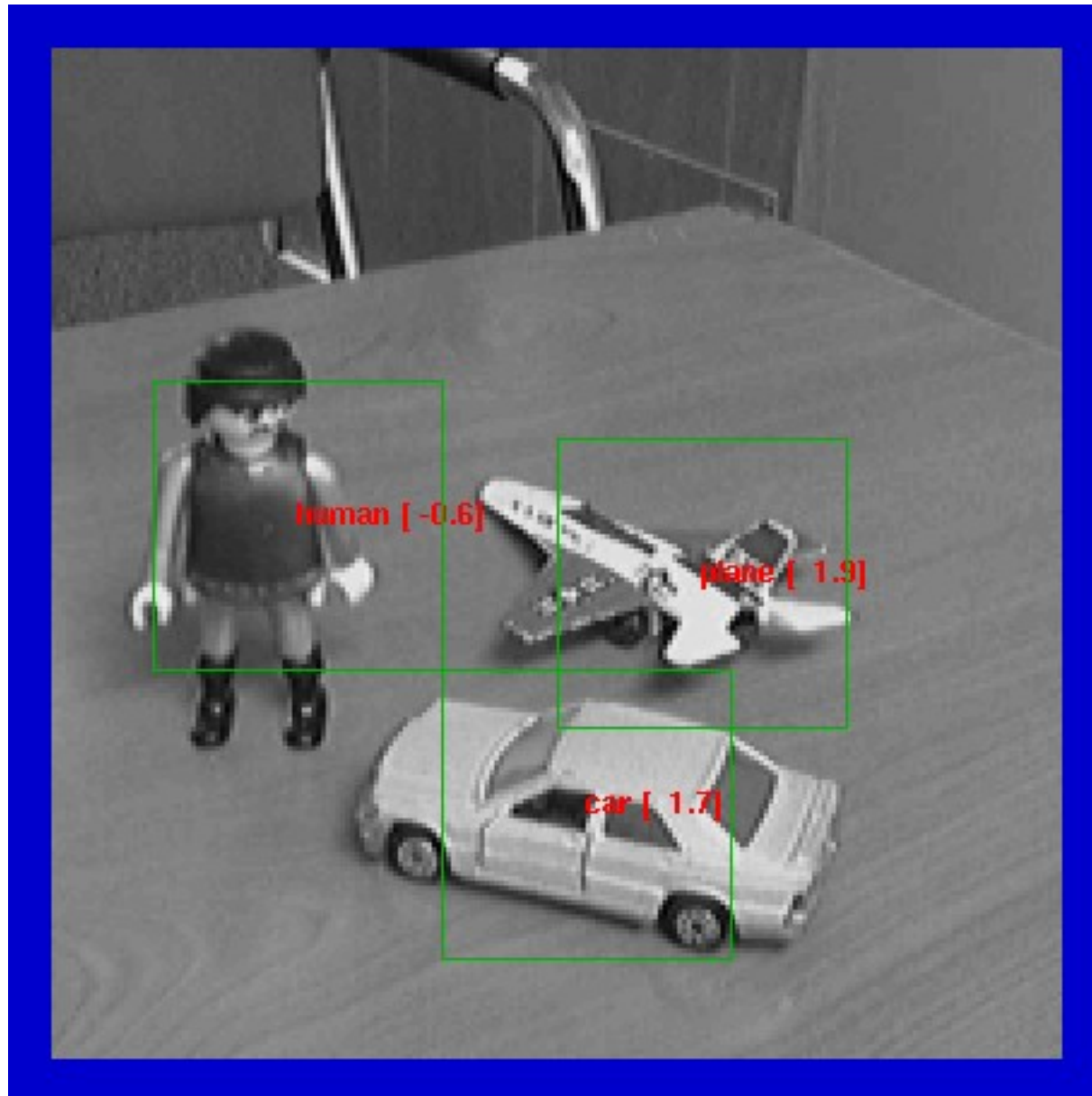


# Examples (Monocular Mode)

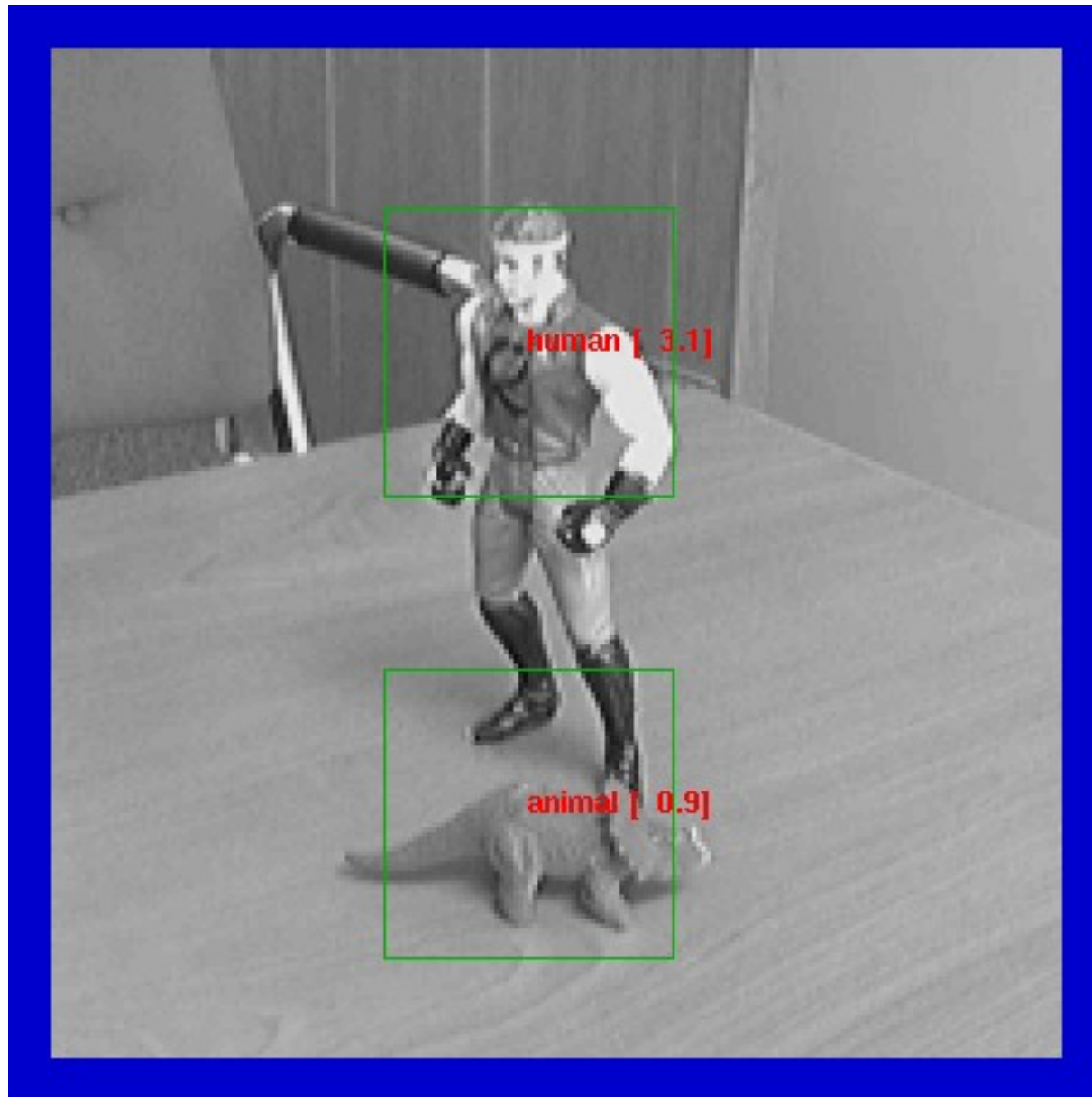




# Examples (Monocular Mode)

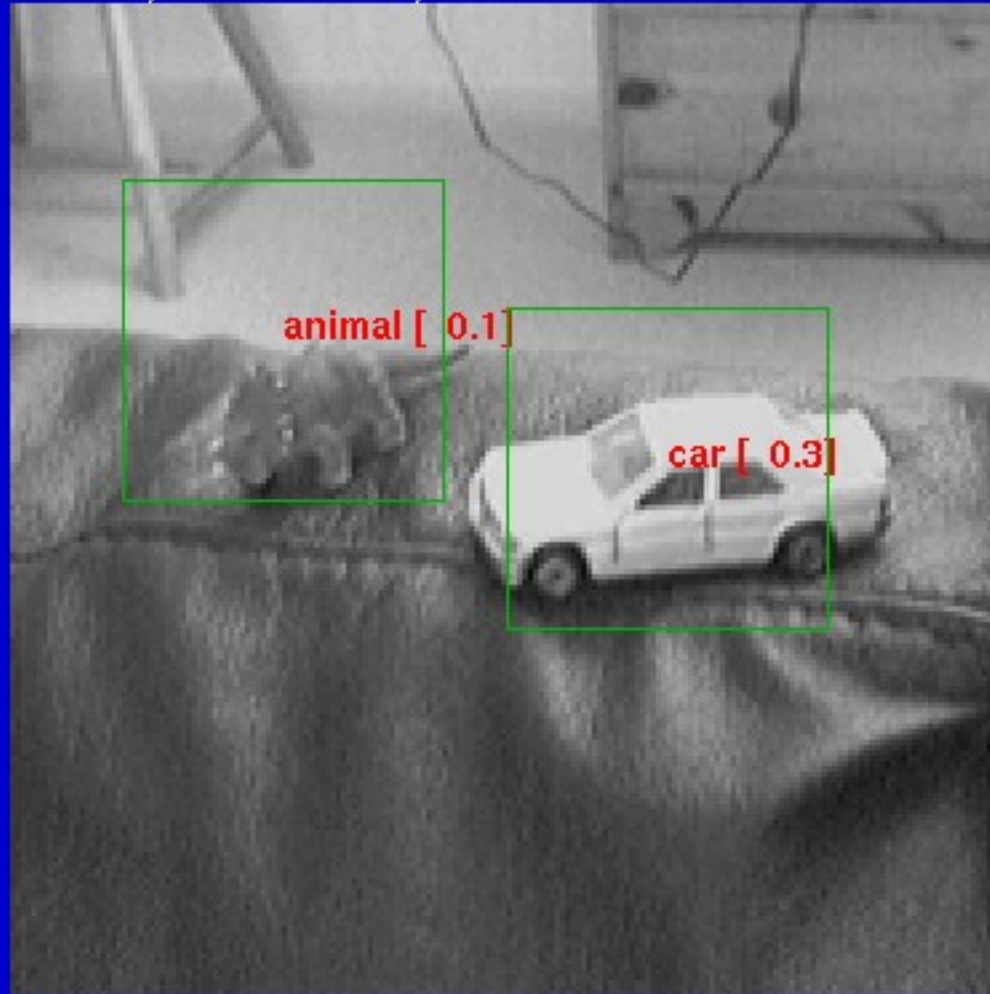


# Examples (Monocular Mode)



# Examples (Monocular Mode)

Zoom= 1.0, Threshold= -1.0, filter on



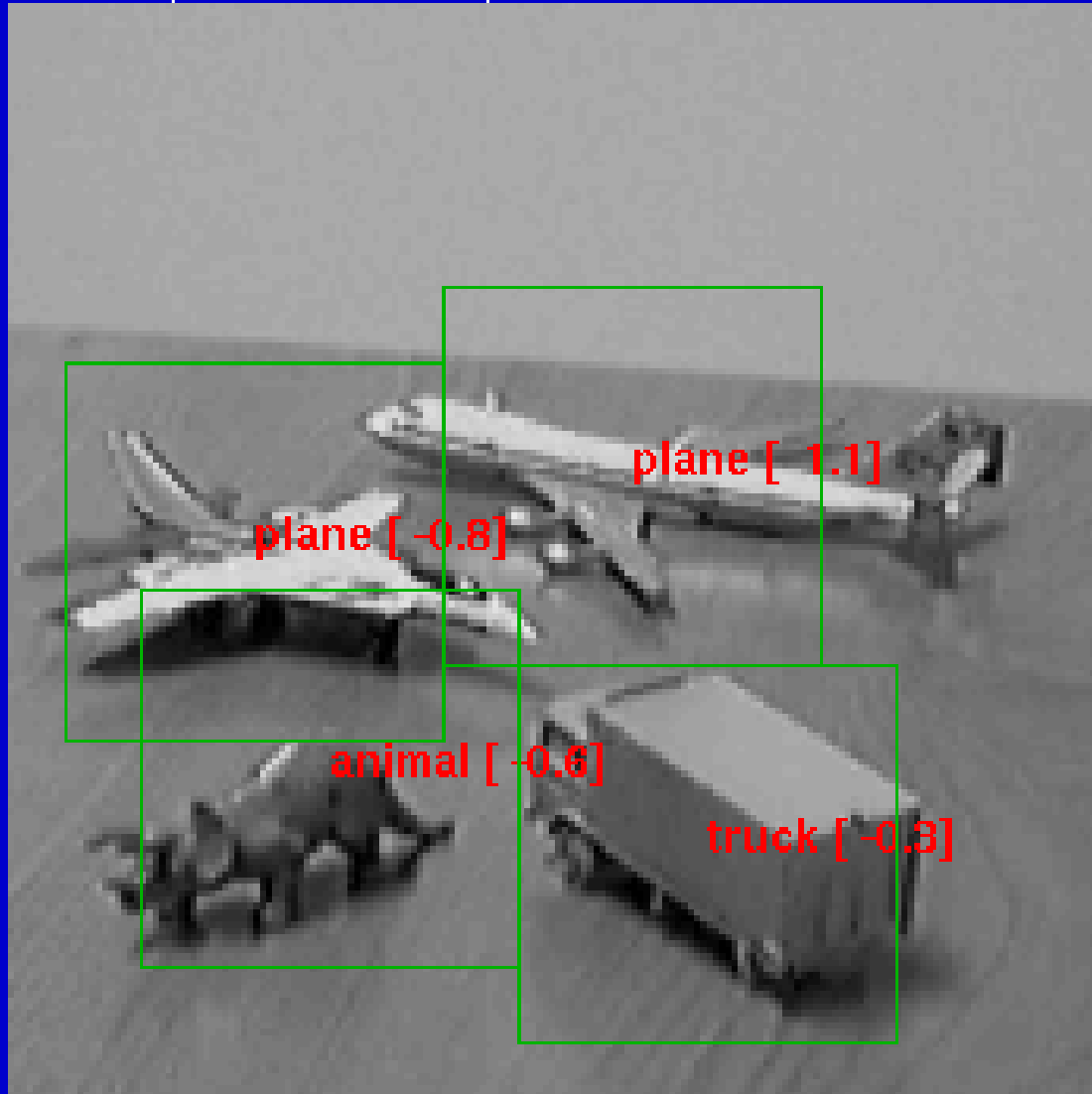
# Examples (Monocular Mode)

Zoom= 1.0, Threshold= -1.2, filter on

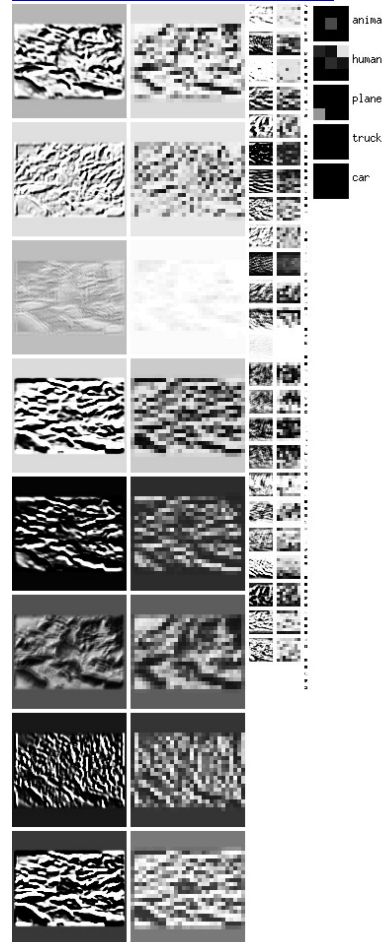
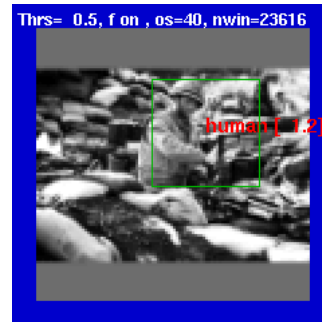
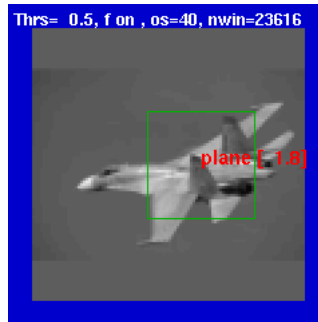
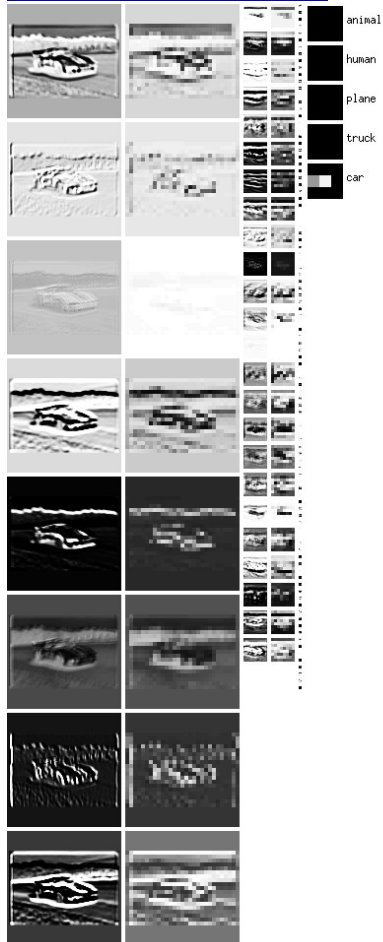
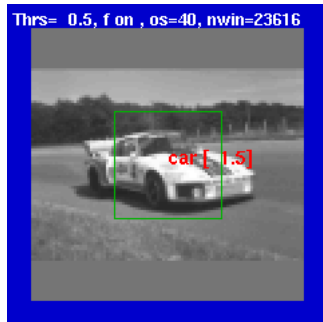


# Examples (Monocular Mode)

Zoom= 0.7, Threshold= -1.8, filter on



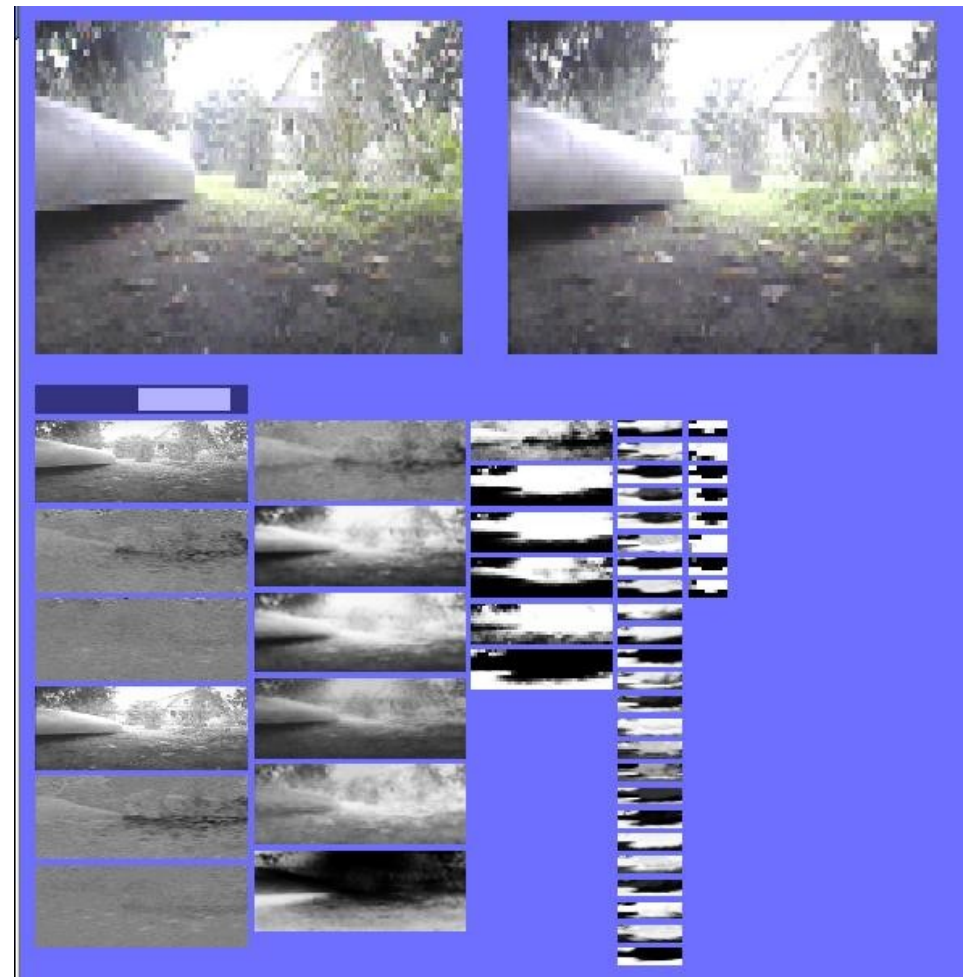
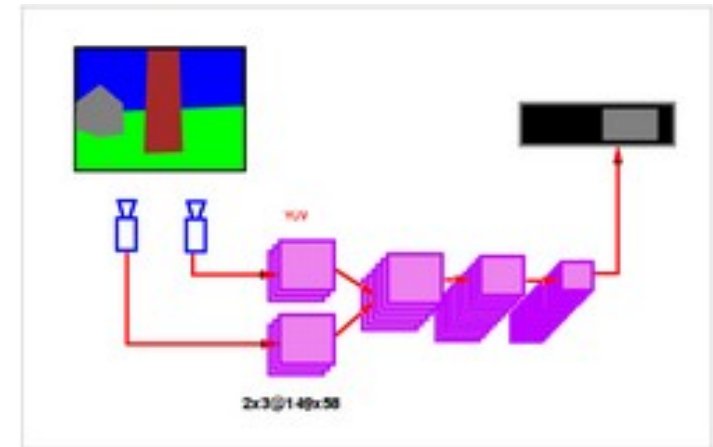
# Natural Images (Monocular Mode)



# Visual Navigation for a Mobile Robot

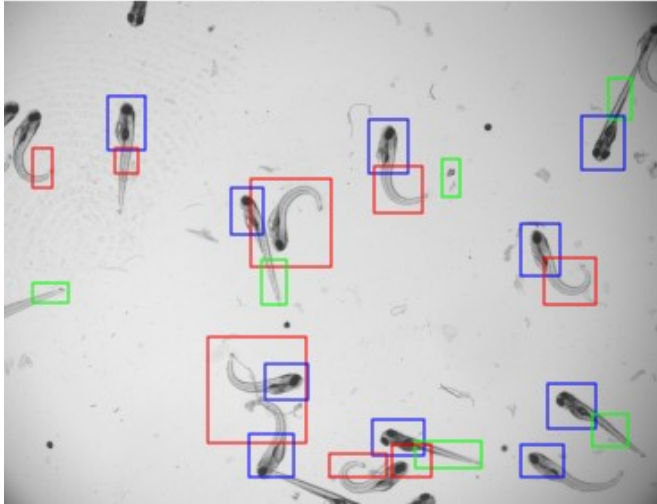
[LeCun et al. NIPS 2005]

- 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

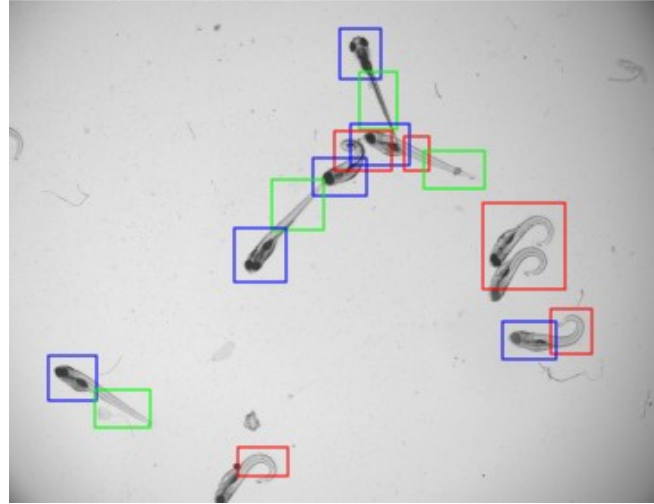


# Convolutional Nets for Counting/Classifying Zebra Fish

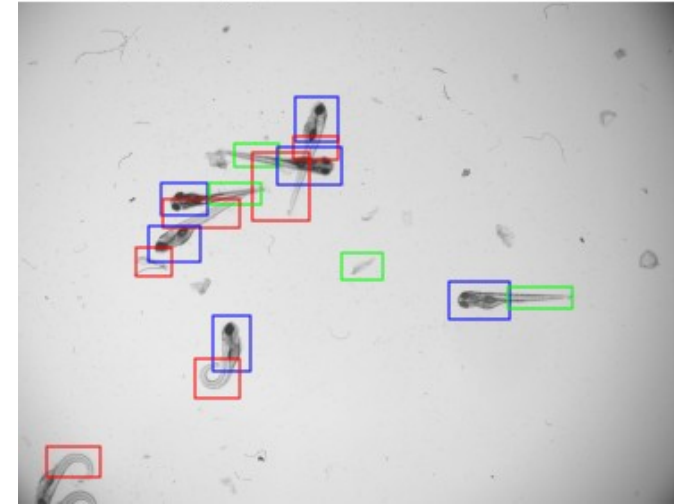
10 head, 6 straight, 8 curved



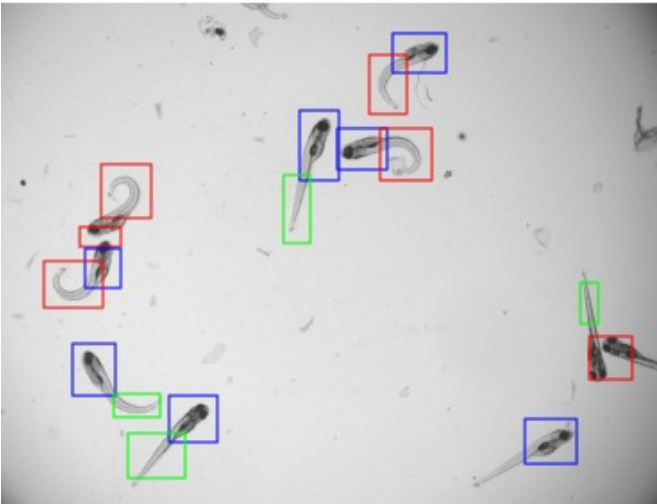
6 head, 4 straight, 5 curved



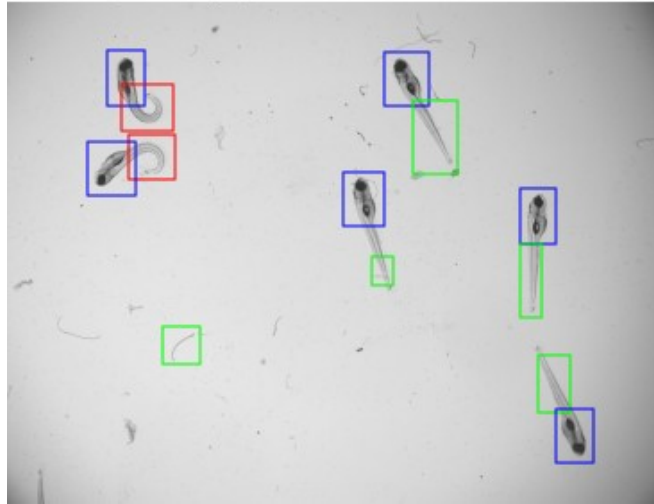
6 head, 4 straight, 6 curved



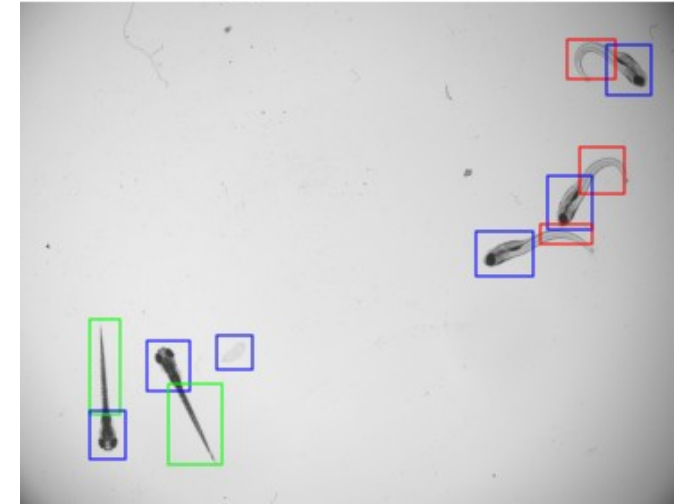
7 head, 4 straight, 6 curved



6 head, 5 straight, 2 curved



6 head, 2 straight, 3 curved



Head – Straight Tail – Curved Tail



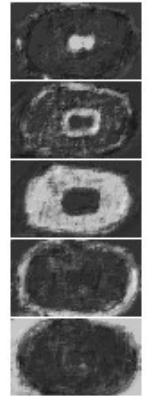
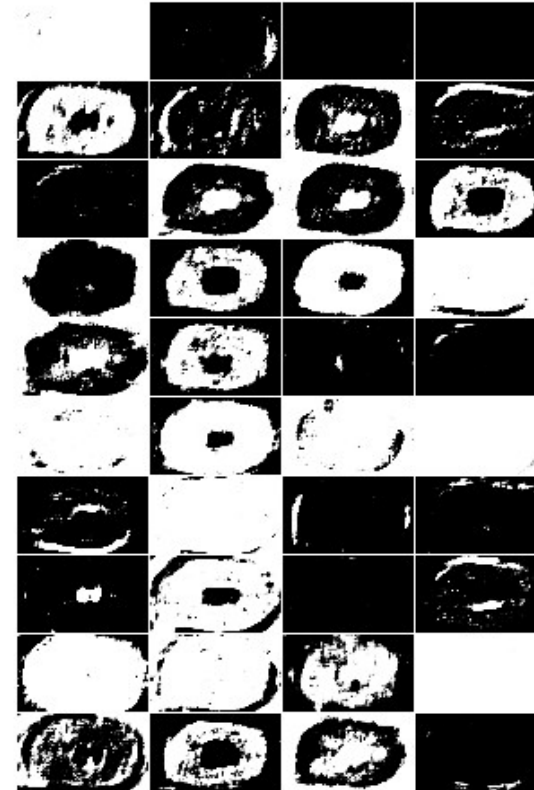
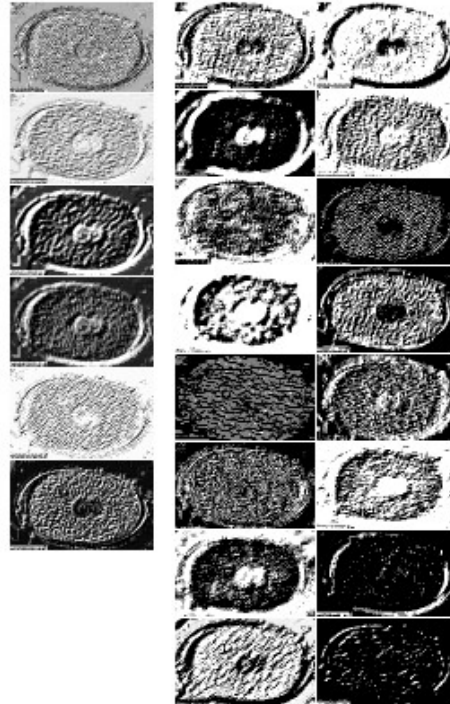
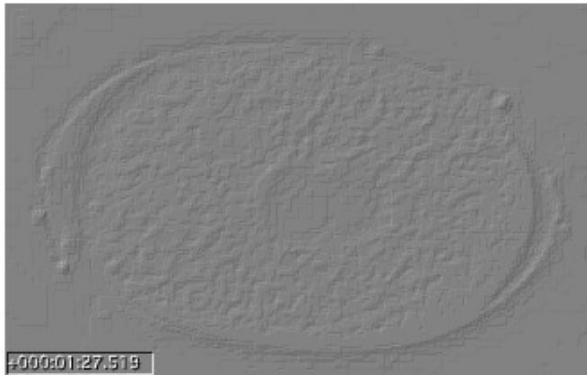
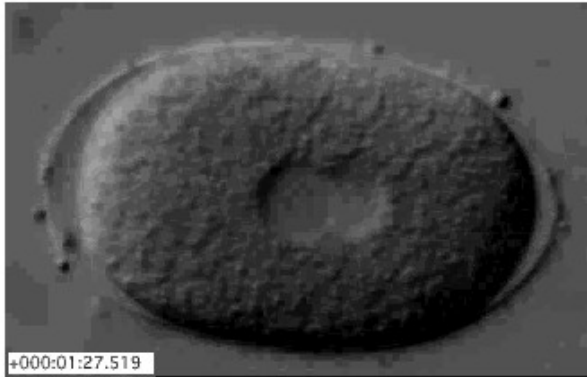
# C. Elegans Embryo Phenotyping

## Analyzing results for Gene Knock-Out Experiments



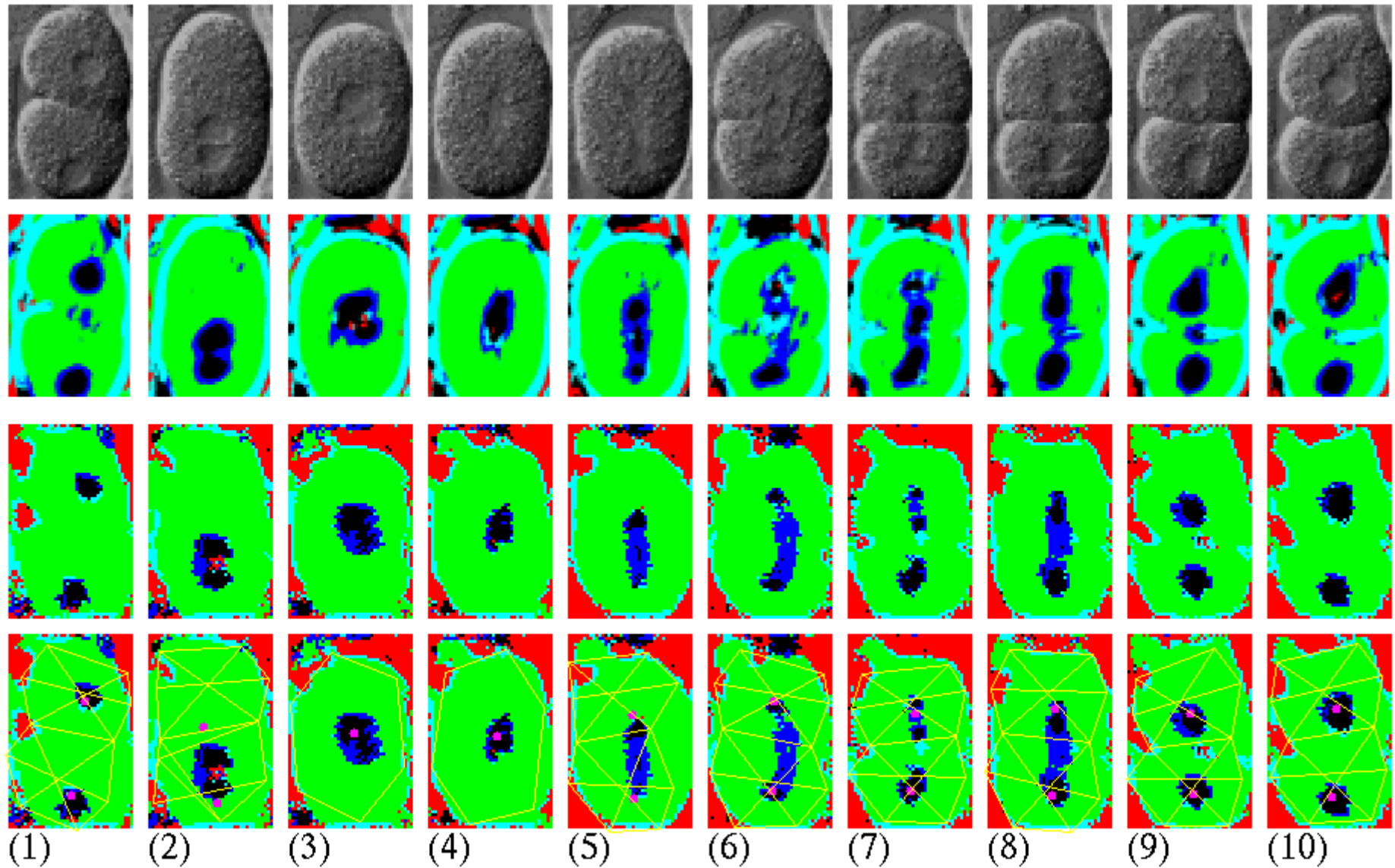
# C. Elegans Embryo Phenotyping

## Analyzing results for Gene Knock-Out Experiments



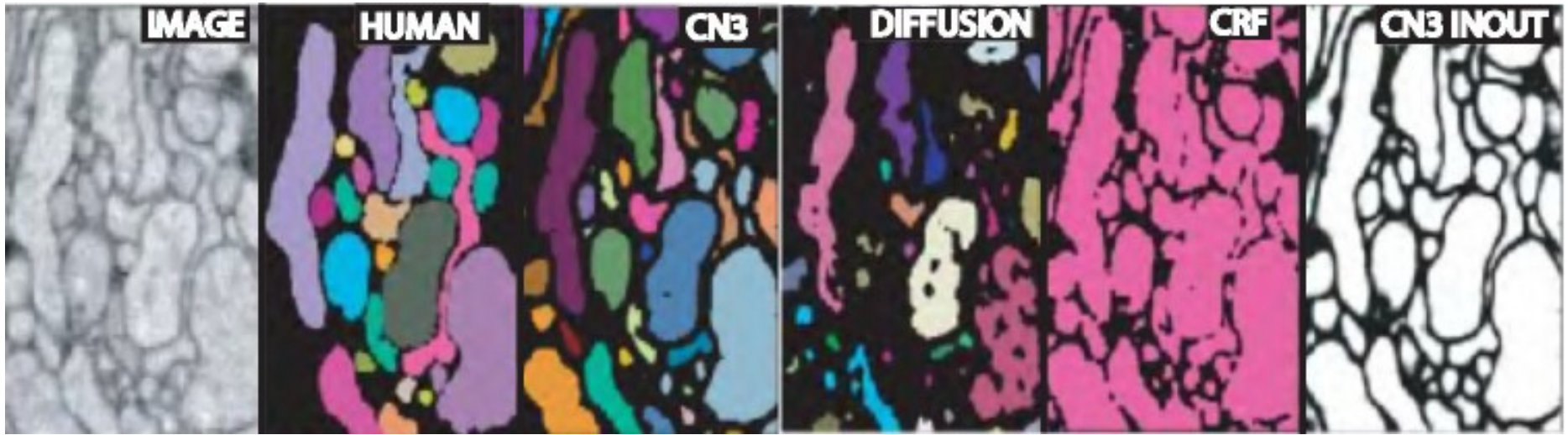
# C. Elegans Embryo Phenotyping

## Analyzing results for Gene Knock-Out Experiments



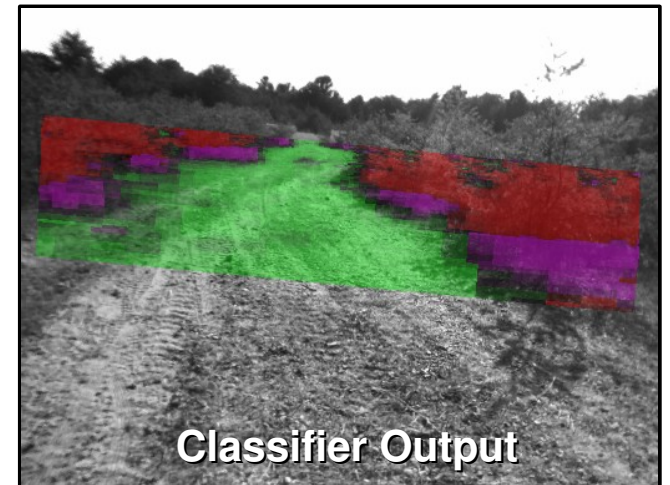
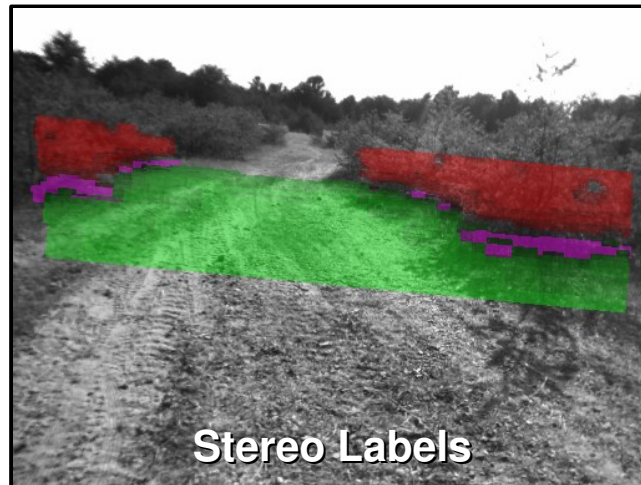
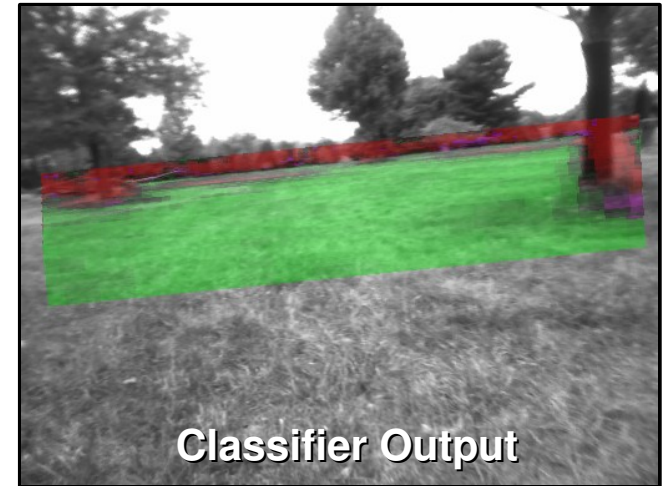
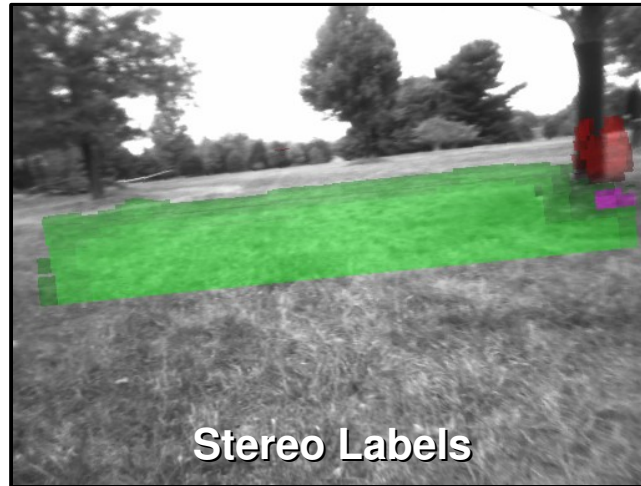
# Convolutional Nets For Brain Imaging and Biology

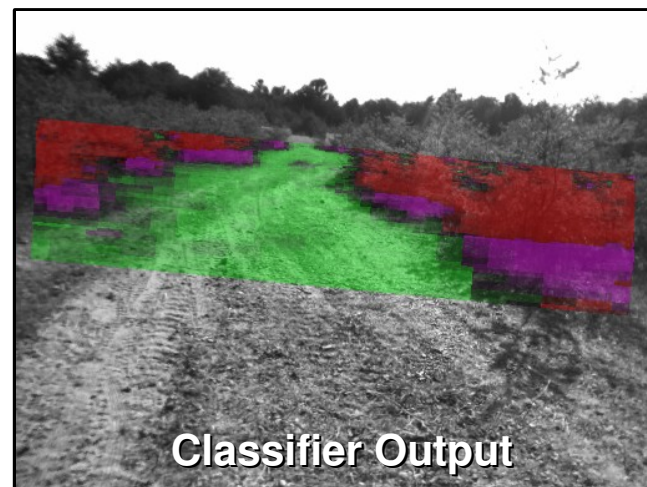
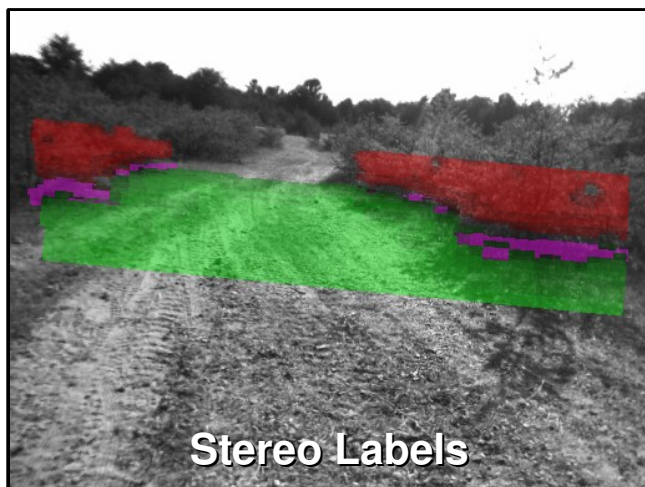
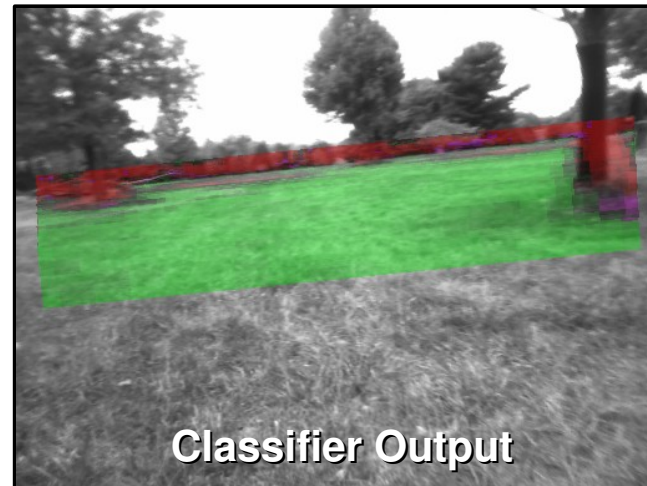
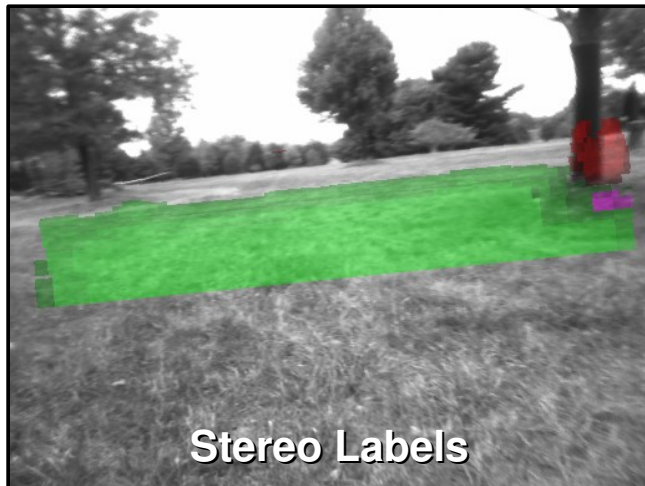
- **Brain tissue reconstruction from slice images [Jain,....,Denk, Seung 2007]**
  - ▶ Sebastian Seung's lab at MIT.
  - ▶ 3D convolutional net for image segmentation
  - ▶ ConvNets Outperform MRF, Conditional Random Fields, Mean Shift, Diffusion,...[ICCV'07]



# Convolutional Nets for Image Region Labeling

- Long-range obstacle labeling for vision-based mobile robot navigation
  - (more on this later....)





# Industrial Applications of ConvNets

## ● AT&T/Lucent/NCR

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

## ● Vidient Inc

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

## ● NEC Labs

- ▶ Cancer cell detection, automotive applications, kiosks

## ● Google

- ▶ OCR, ???

## ● Microsoft

- ▶ OCR, handwriting recognition, speech detection

## ● France Telecom

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

## ● Other projects: HRL (3D vision)....

# CNP: FPGA Implementation of ConvNets

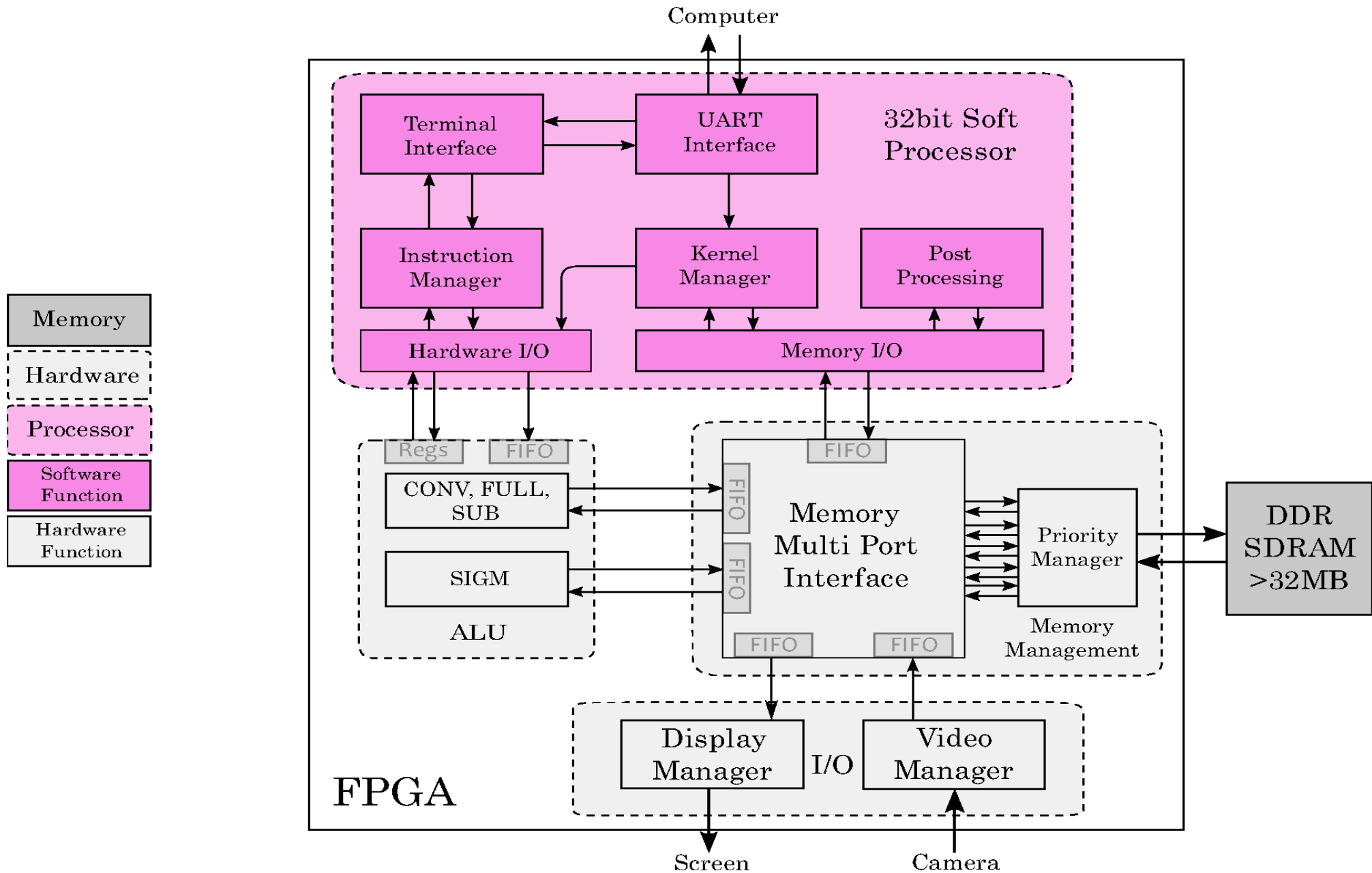
## Implementation on low-end Xilinx FPGA

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

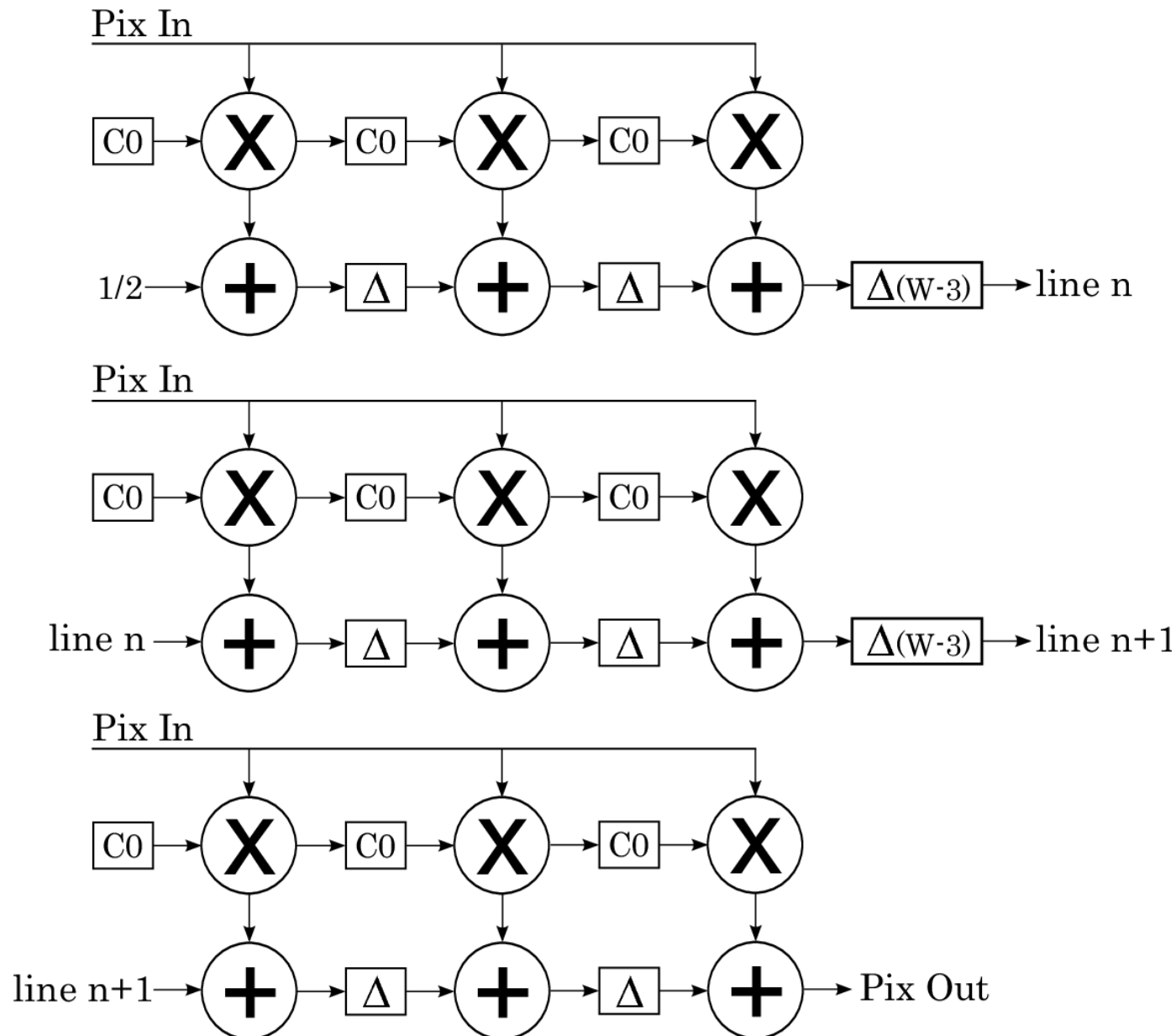




# CNP Architecture



# Systolic Convolver: 7x7 kernel in 1 clock cycle



# Design

## • **Soft CPU used as micro-sequencer**

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

## • **16x16 fixed-point multipliers**

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

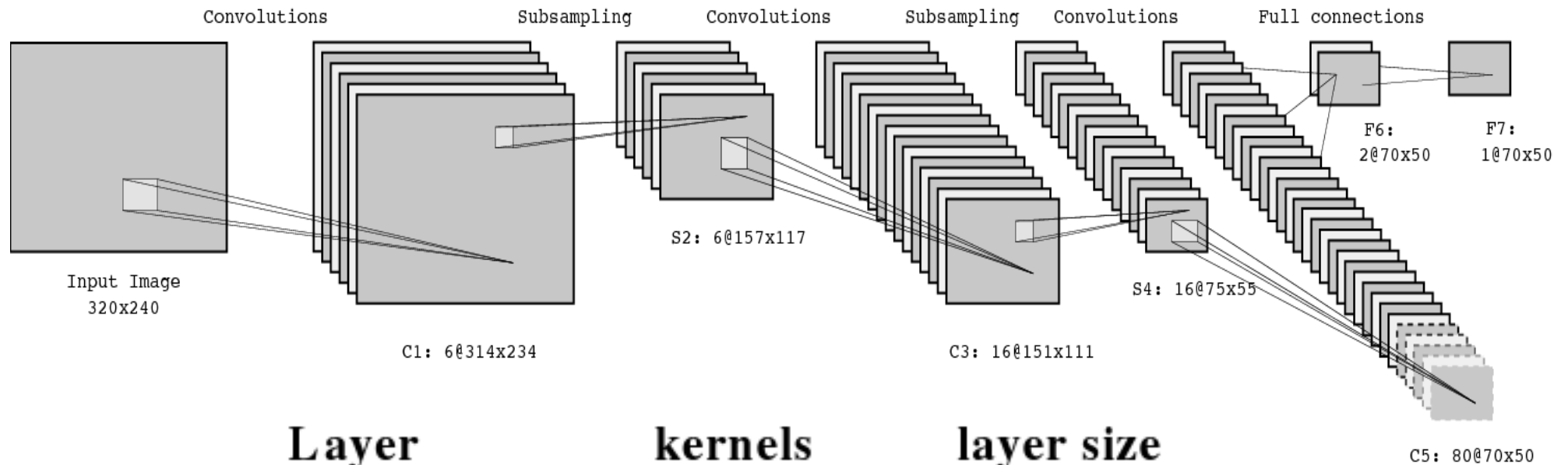
## • **Instruction set includes:**

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

## • **Microcode generated automatically from network description in Lush**

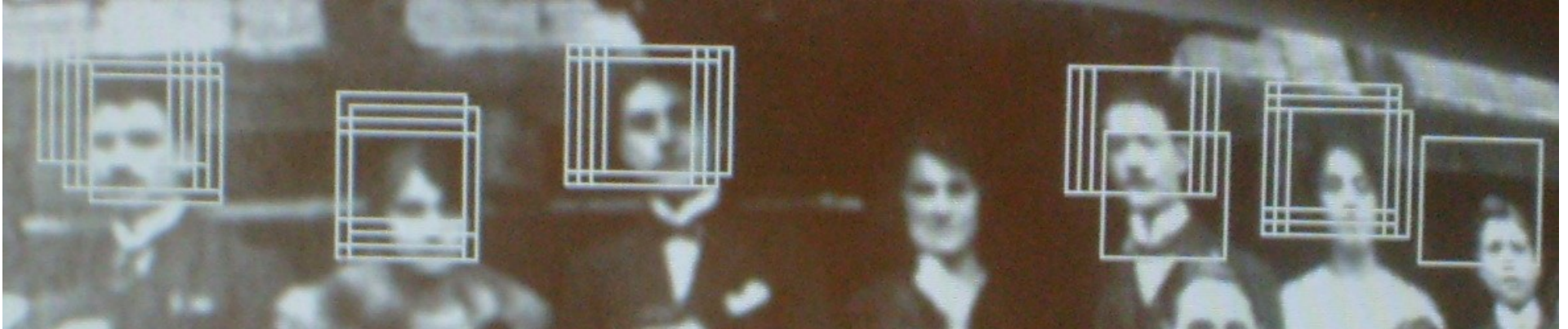
<b>Entity</b>	<b>Occupancy</b>	
I/Os	135 out of 469	28%
DCMs	2 out of 8	25%
Mult/Accs	56 out of 126	44%
Bloc Rams	100 out of 126	84%
Slices	16790 out of 23872	70%

# Face detector on CNP



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

# Results



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

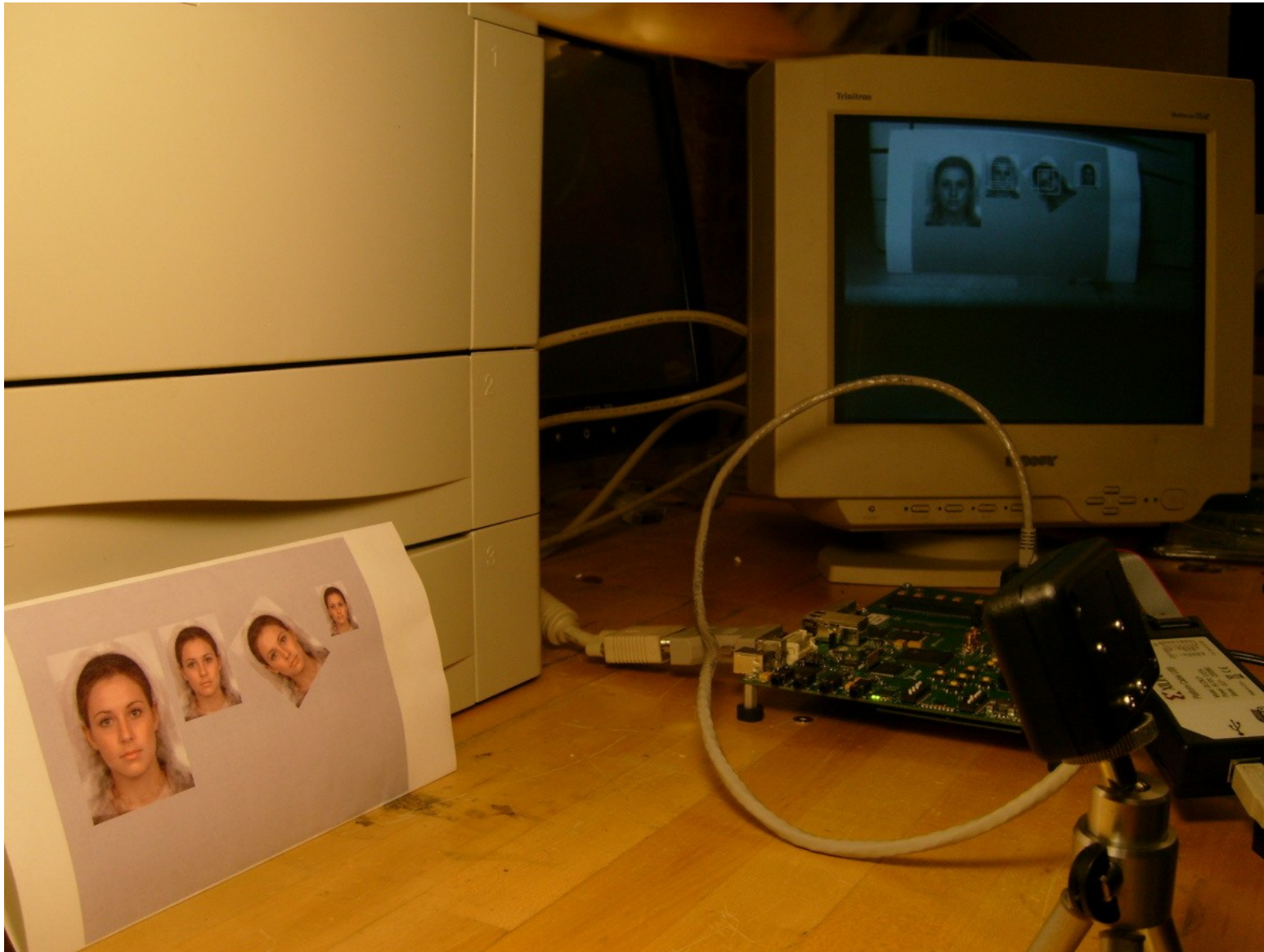
# Results



# Results

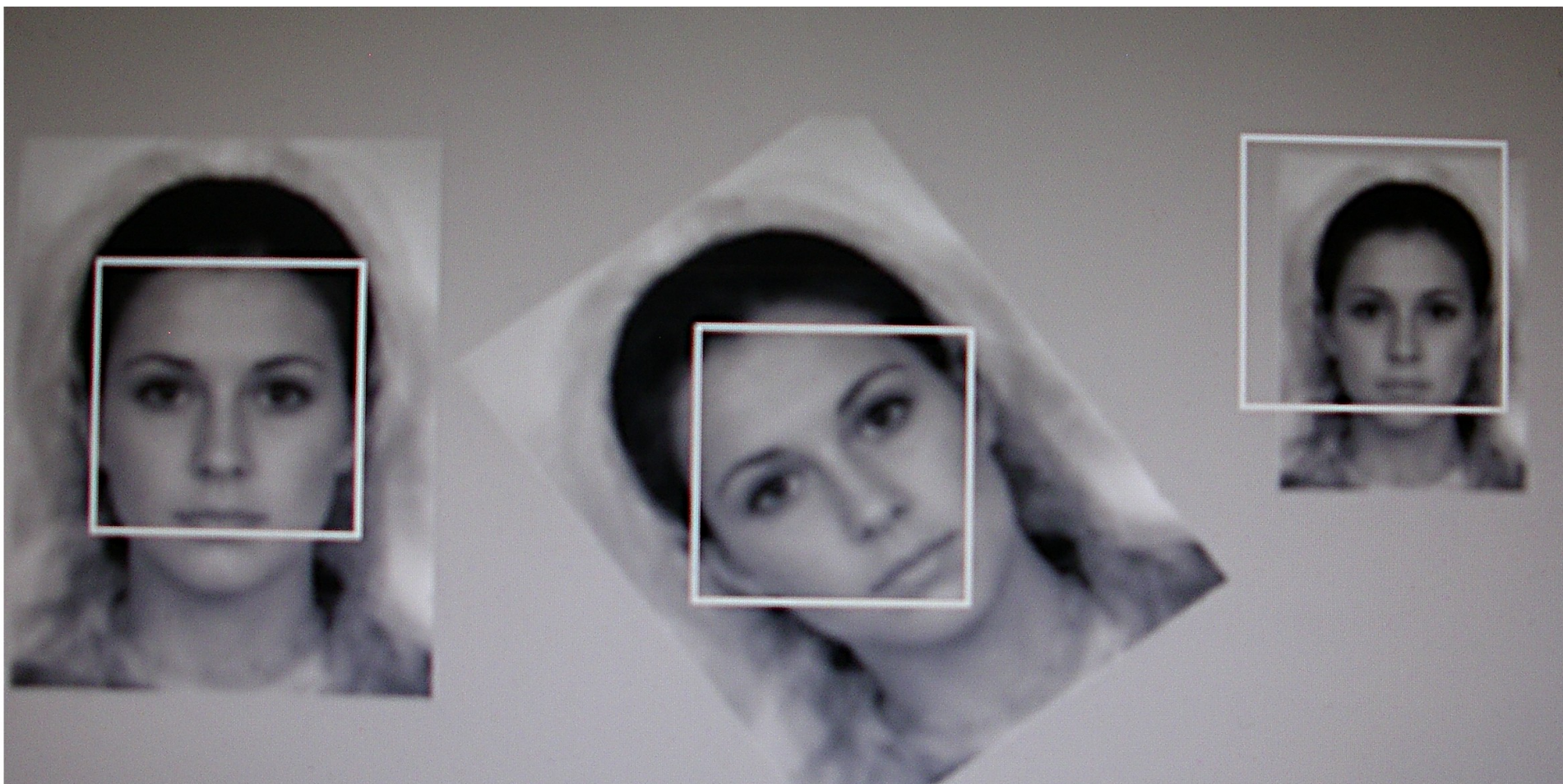


# Results

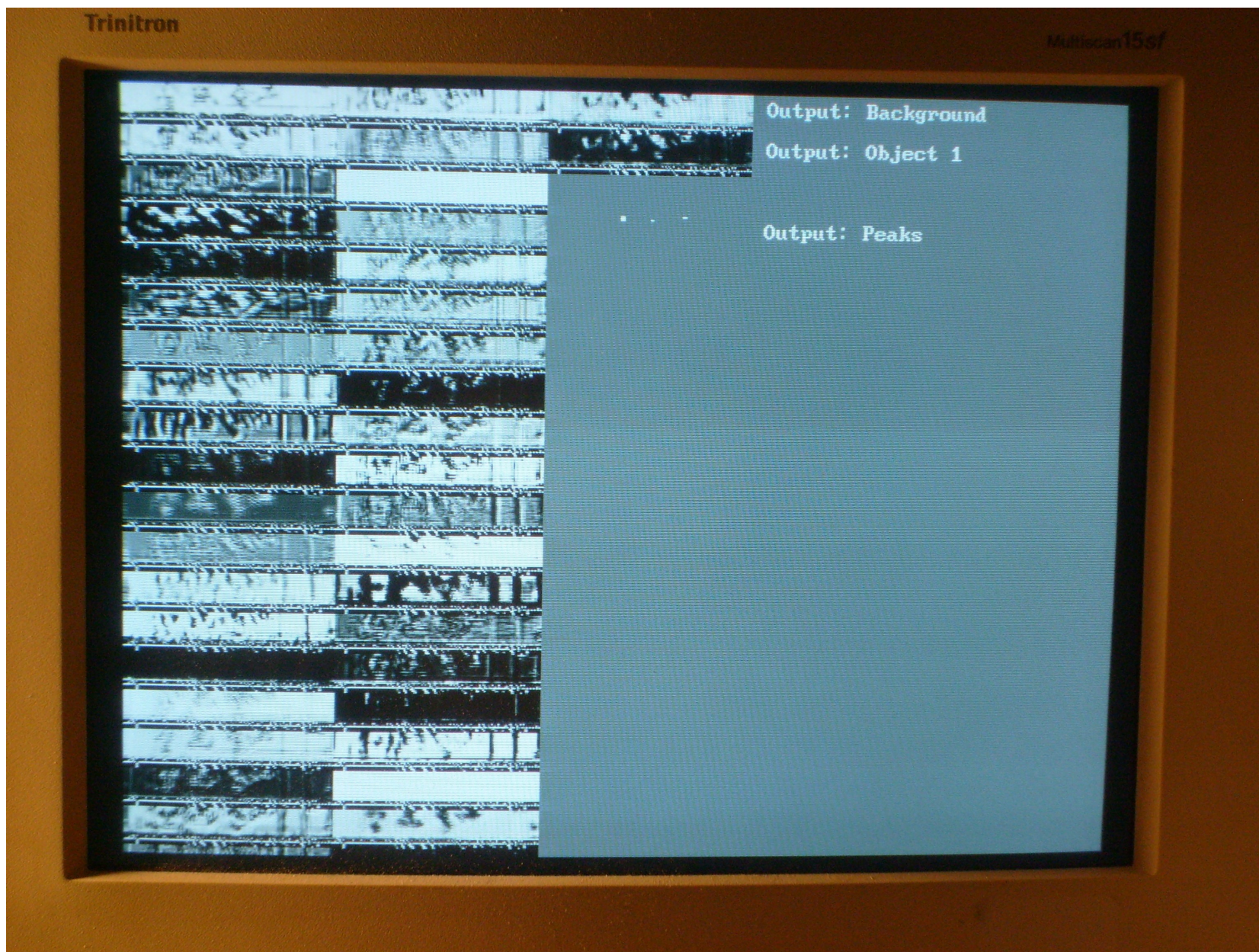




# Results



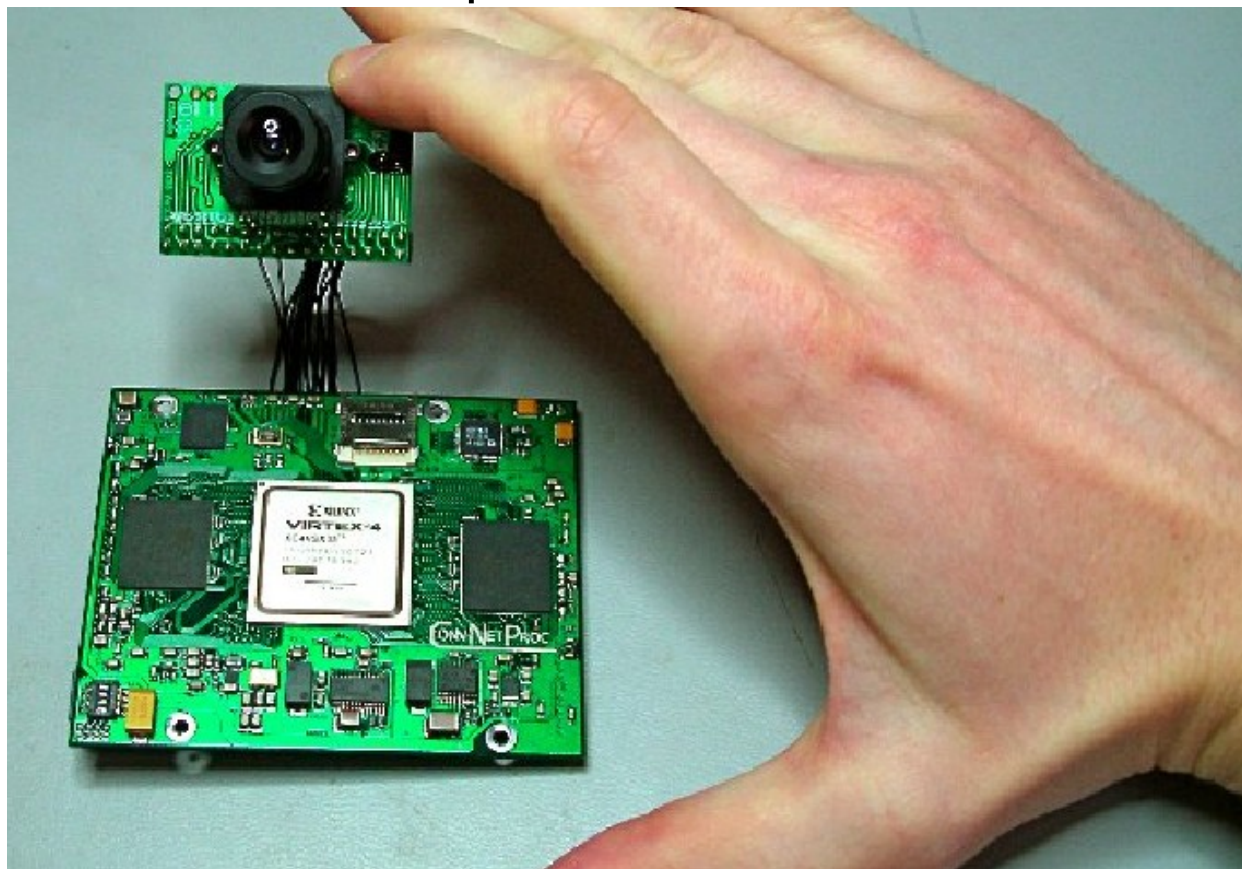
# Results



# FPGA Custom Board: NYU ConvNet Proc

## ● Xilinx Virtex 4 FPGA, 8x5 cm board

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



# Models Similar to ConvNets

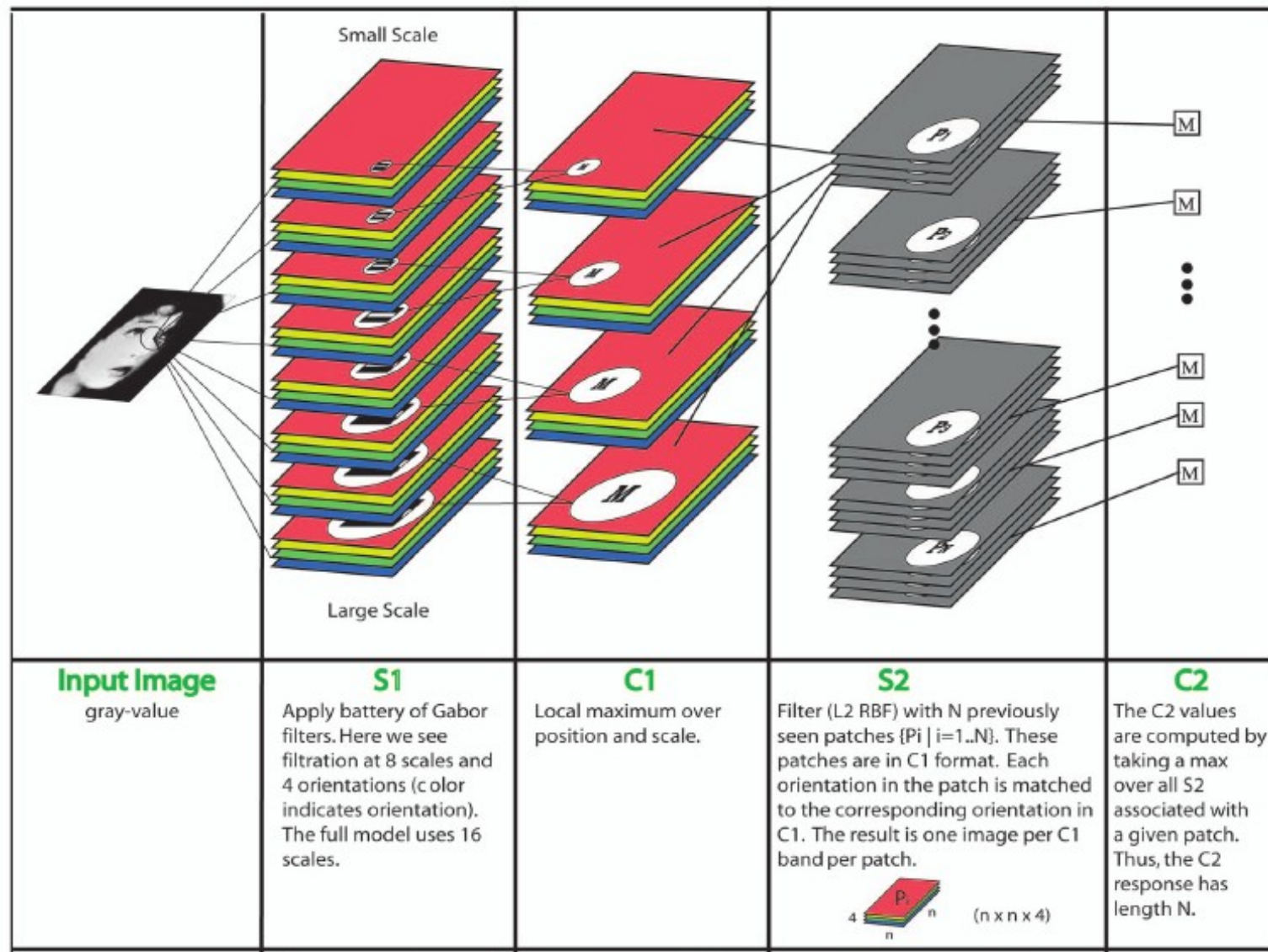
## HMAX

- ▶ [Poggio & Riesenhuber 2003]
- ▶ [Serre et al. 2007]
- ▶ [Mutch and Lowe CVPR 2006]

## Difference?

- ▶ the features are not learned

## HMAX is very similar to Fukushima's Neocognitron



[from Serre et al. 2007]