

# End-to-End Learning

Yann LeCun

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 those representations by just looking at the world?

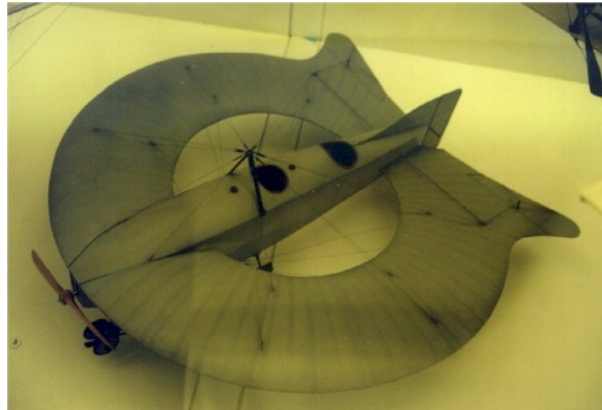
# Challenges of Artificial Perception (& Neuroscience)

## 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)



# What is Intelligence?

- **Most wetware cycles in higher animals are devoted to perception, and most of the rest to motor control.**
  - ▶ 20% of our brain does vision
- **Intelligence includes the ability to derive complex behavior from massive amounts of sensory information.**
  - ▶ Intelligence requires making sense complex sensor input (including proprioception).
- **Intelligence is modeling, prediction, and evaluation**
  - ▶ The more intelligent the organism, the better it can predict the world, predict the consequences of its actions (including rewards), and pick the “best” action.
- **How can an intelligent agent learn to predict the world?**
  - ▶ Not the whole world, only the part of the world which is relevant to the purpose of its existence, including its own actions.



# What is Intelligence?

- **In the past we thought that intelligence was what smart humans could do**
  - ▶ Speech, logical reasoning, playing chess, computing integrals.....
- **But those things turned out to be pretty simple computationally**
  - ▶ It turns out that the complicated things are perception, intuition and common sense
  - ▶ Things that even a mouse can do much better than any existing robot.
- **What about rat-level intelligence?**
  - ▶ We will achieve rat-level intelligence before we achieve human-level, so let's work on rat-level intelligence
- **It is likely that intelligent agents cannot be “intelligently designed” ;-)**
  - ▶ They have to build themselves through learning (or evolution)
  - ▶ How much prior structure is required?
  - ▶ The more intelligent, the less prior structure.

# Where are we now?

- **We are still quite far from rat-level intelligence**
  - ▶ We don't really have robots that can run around without bumping into things or falling into ditches using vision only.
  - ▶ We know that the methods currently being developed are hacks.
- **Most NIPS/ICML papers do not take us any closer to rat-level intelligence**
  - ▶ In fact, some of them take us backward.
- **What problems should we work on, what type of new methods should we develop?**
- **Which methods currently being worked on the community, while being valuable, will not take us to rat-level intelligence?**
  - ▶ How do we avoid the trap of building ever so taller ladders when our goal is to reach the Moon?

# Questions?

## • Is there a magic bullet?

- ▶ Is there a general principle for learning/AI, or is it just a bunch of tricks? (see Gary Marcus's book "Kluge").
- ▶ Is there a **universal learning algorithm/architecture** which, given a small amount of appropriate prior structure, can produce intelligent agents?
- ▶ Or do we need to accumulate a large repertoire of "modules" to solve each specific problem an intelligent agent must solve. How would we assemble those modules?

## • Let's face it, our only working example uses neurons.

- ▶ Does that mean rat-level intelligence will be achieved with simulated neurons?
- ▶ Airplanes don't flap their wings
- ▶ Yes, but they exploit the same aerodynamical properties as birds (and they are not as efficient at it)
- ▶ What is the analog of aerodynamics for intelligence?

# Questions?

- **Every reasonable learning algorithm we know minimizes some sort of loss function?**
  - ▶ Does the brain minimize a loss function?
  - ▶ If yes, what is it, and how does it do it?
  - ▶ Is there any other way to build learning systems?
- **If we accept the loss function hypothesis:**
  - ▶ Can the loss function possibly be convex?
  - ▶ How is it parameterized?
    - What is the architecture?
  - ▶ How is it minimized
    - direct solution (like quadratic problems),
    - gradient-based iterative procedures
    - perturbation/trial and error
    - all of the above?

# What Do We Need?

## • Learning

- ▶ Supervised, unsupervised, reinforcement

## • Efficient reasoning with large numbers of variables

- ▶ e.g. For image segmentation/labeling...

## • Very fast inference for high-dimensional complex tasks

- ▶ People and animal can recognize common visual categories in less than 100ms. There is no time for complex reasoning/relaxation/inference.

## • Emotions

- ▶ Emotion, in the restricted sense of reward prediction.



# What Do We Need?

## ● Learning algorithms that scale

- ▶ Sub-linearly with the number of training samples
- ▶ Not much more than linearly with the size of the learning system
- ▶ Non-exponentially with the size of the action space and state space

## ● “Deep” Learning

- ▶ Intelligent inference requires lots of elementary decisions (non-linear steps):
  - pixels->low-level feature->high-level features->categories
  - Global goal->macro-action sequence->action sequence->motor commands

## ● A framework with which to build large-scale “deep” learning machines with millions of parameters

- ▶ A framework that allows us to specify prior knowledge

## ● How little prior knowledge can we get away with?

# What classes of methods will NOT take us there?

## • Template matching

- ▶ Fast nearest-neighbor methods, kernel methods, and other “glorified template matching methods” are very useful, but they won't take us there.

## • “Shallow” learning

- ▶ Linear combinations of **fixed** basis functions won't take us there
- ▶ Examples: SVM, generalized linear models, boosting with simple weak learners, Gaussian processes.....Those methods are useful, just not for our purpose
- ▶ Shallow architectures are inefficient: most complex functions are more efficiently implemented with many layers of non-linear decisions
- ▶ We need “hierarchical” models that learn high-level representations from low-level perceptions, features, macro features.....

# What classes of methods will NOT take us there?

## ❶ Convex Optimization with a practical number of variables

- ▶ If intelligence could be reduced to convex optimization, the order in which we learn things would not matter: we would go to the same minimum no matter what.

## ❷ Purely generative methods, purely supervised methods, purely unsupervised methods.

## ❸ Fully probabilistic methods

- ▶ Because we can't normalize complicated distributions in high dimension

## ❹ Purely discriminative methods

- ▶ Because not everything comes down to classification
- ▶ We need to model the world

# What problems should we solve?

## ■ Integrating reasoning with learning

- ▶ Graphical models (factor graphs in particular) are a good avenue, but we need to free ourselves from the “partition function problem”
- ▶ How do we build non-probabilistic factor graphs that can be trained in supervised, unsupervised, and reinforcement mode.

## ■ Invariant Representations

- ▶ How do we learn complex invariances in vision?
- ▶ We can pre-engineer some of it, but ultimately, we need a scheme to learn features and invariant representation automatically.
- ▶ Optimization algorithms (learning) will become better at this than human engineer. It is already the case for handwriting recognition systems and speech recognition systems.

# What problems should we solve?

## Deep learning

- ▶ Ultimately, we need to think again about learning in deep structures with many layers on non-linear decisions.
- ▶ We have partial solutions that are not entirely satisfactory
- ▶ Pure back-prop can't handle more than a few layers and is very inefficient for unsupervised learning
- ▶ Probabilistic belief nets have some of the right ingredients (e.g. Boltzmann machines-like algorithms), but they are plagued by the “partition function problem”: learning involves estimating intractable integrals.



# What Project should we work on?

## ● Vision

- ▶ Generic object recognition, object detection, and such are some of the most challenging perceptual tasks for learning
- ▶ We can make progress with clever as-hoc preprocessing combined with simple (linear) learning methods or generative models, but ultimately, good performance will be achieved when we come up with efficient “deep” learning algorithms that can learn the whole task **end-to-end** (with the minimum amount of prior knowledge)

## ● Robotics

- ▶ The best way to integrate perception, action, and reinforcement.

# The visual system is “deep” and learned

## ● The primate's visual system is “deep”

- ▶ It has 10-20 layers of neurons from the retina to the infero-temporal cortex (where object categories are encoded).
- ▶ How does it train itself by just looking at the world?.

## ● Is there a magic bullet for visual learning?

- ▶ The neo-cortex is pretty much the same all over
- ▶ The “learning algorithm” it implements is not specific to a modality (what works for vision works for audition)
- ▶ There is evidence that everything is learned, down to low-level feature detectors in V1
- ▶ Is there a **universal learning algorithm/architecture** which, given a small amount of appropriate prior structure, can produce an intelligent vision system?
- ▶ Or do we have to keep accumulating a large repertoire of pre-engineered “modules” to solve every specific problem an intelligent vision system must solve?

# Can we learn everything end-to-end?

- **There are simple situations where we have been able to demonstrate end-to-end learning in vision.**
  - ▶ Learning to recognize handwritten words from pixels to labels with no preprocessing and minimal prior knowledge (NIPS 1990-1998)
  - ▶ Learning to detect faces (NIPS 2004)
  - ▶ Learning to detect and recognize generic object categories from raw pixels to labels (CVPR 2003)
  - ▶ Training a robot to avoid obstacles from raw left/right image pairs to steering angles (NIPS 2005)

# Two Big Problems in Machine Learning and Computer Vision

## 2. The “Deep Learning Problem”

- ▶ Training “Deep Belief Networks” is a necessary step towards solving the invariance problem in vision (and perception in general).
- ▶ How do we train deep architectures with lots of non-linear stages?

## 1. The “Intractable Partition Function Problem”

- ▶ Give high probability (or low energy) to good answers
- ▶ Give low probability (or high energy) to bad answers
- ▶ There are too many bad answers!
- ▶ The normalization constant of probabilistic models is a sum over too many terms.

## 1. Deep Supervised Learning, Gradient-Based Learning

- ▶ Simple gradient-based learning
- ▶ Learning invariant feature hierarchies for visual recognition
- ▶ Convolutional networks

## 2. Energy-Based Models. Structured Output Models.

- ▶ Circumventing the “Intractable Partition Function Problem”
- ▶ Discriminative learning for structure output models
- ▶ Energy-Based factor graphs
- ▶ CRF, Max Margin Markov nets, Graph Transformer Networks

## 3. EBM for Manifold Learning and Feature Extraction

- ▶ Learning Similarity Metrics
- ▶ Neighborhood Component Analysis, DrLIM

## 4. Deep Unsupervised Learning

- ▶ Unsupervised learning: the Energy-Based approach.
- ▶ Restricted Boltzmann Machines, Contrastive Divergence.
- ▶ Sparse feature models.



# Supervised Deep Learning

# Deep Supervised Learning for Vision: The Convolutional Network Architecture

## Convolutional Networks:

- ▶ [LeCun et al., Neural Computation, 1988]
- ▶ [LeCun et al., Proc IEEE 1998]

## 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]

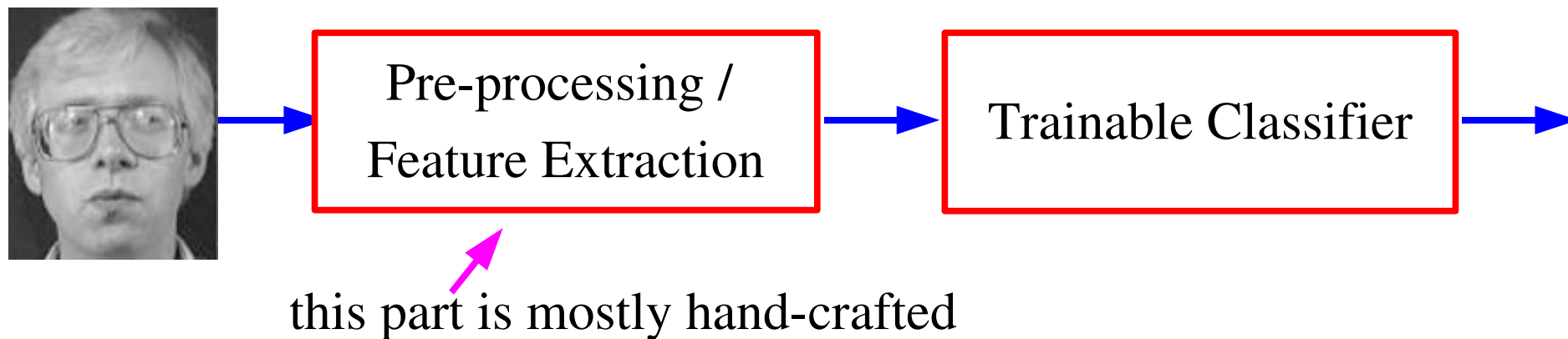
## Category-level object recognition with invariance to pose and lighting

- ▶ [LeCun, Huang, Bottou, CVPR 2004]
- ▶ [Huang, LeCun, CVPR 2005]

## autonomous robot driving

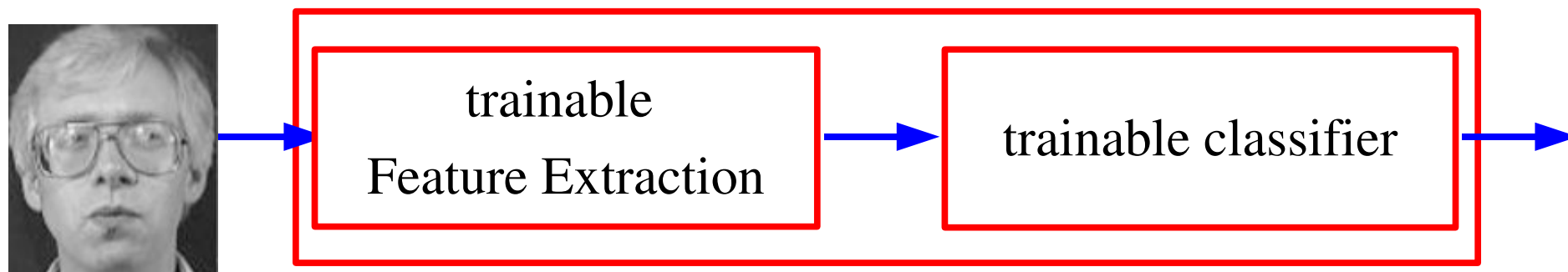
- ▶ [LeCun et al. NIPS 2005]

# The Traditional Architecture for Recognition



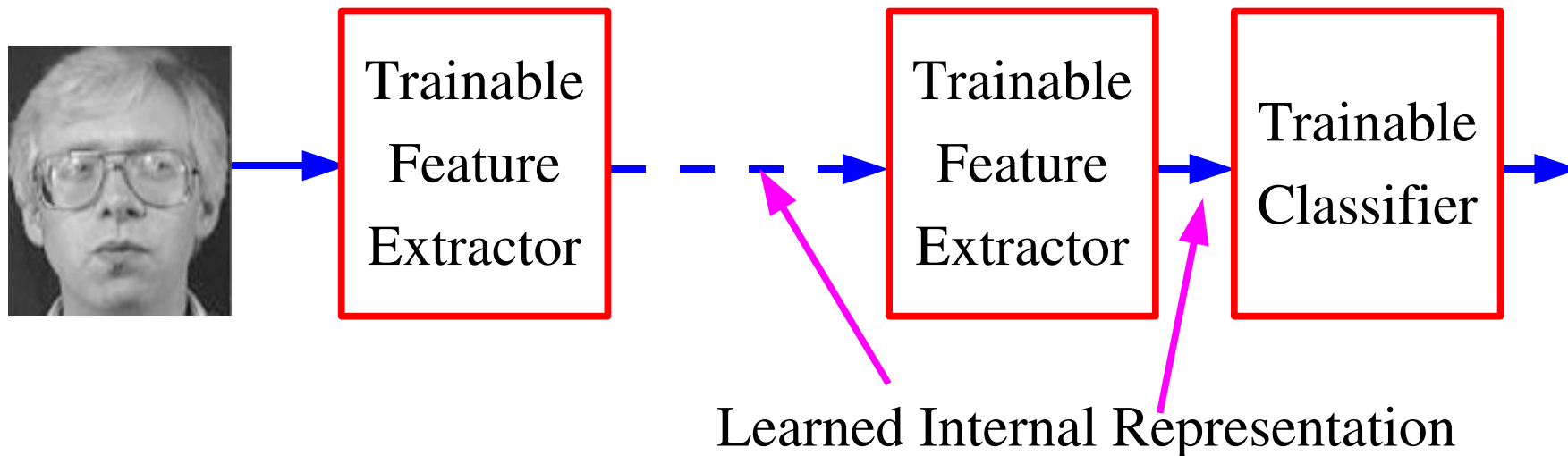
- The raw input is pre-processed through a hand-crafted feature extractor
- The trainable classifier is often generic (task independent)

# End-to-End Learning



- The entire system is integrated and trainable “end-to-end”.
- In some of the models presented here, there will be no discernible difference between the feature extractor and the classifier.
- We can embed general prior knowledge about images into the architecture of the system.

# “Deep” Learning: Learning Internal Representations



- **Deep Learning: learning a hierarchy of internal representations**
- **From low-level features to mid-level invariant representations, to object identities**



## 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 we need to build intelligent machines (the “AI-set”)

# Why should Deep Architectures be more Efficient?

## • A deep architecture trades space for time

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

## • Example1: N-bit parity

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

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

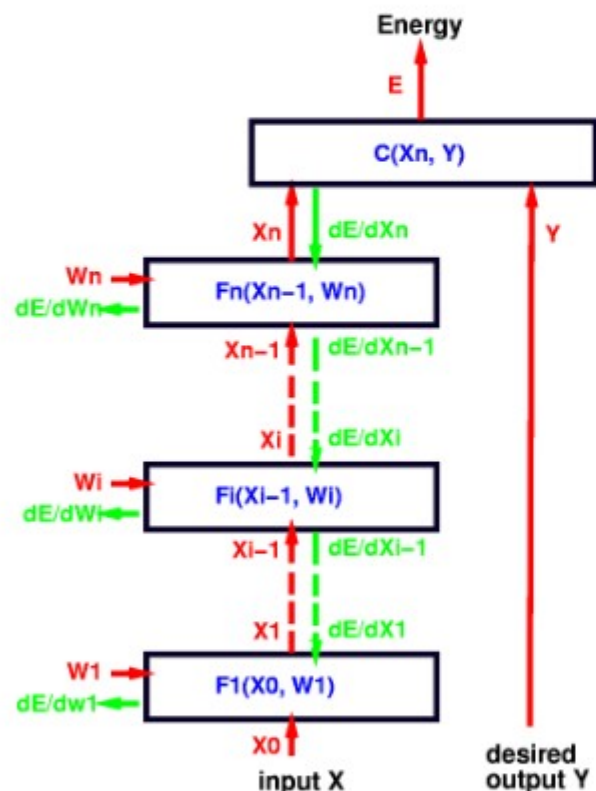
- ▶ Requires  $O(N)$  gates, and  $O(N)$  layers using  $N$  one-bit adders with ripple carry propagation.
- ▶ Requires lots of gates (some polynomial in  $N$ ) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
- ▶ Bad news: almost all boolean functions have a DNF formula with an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

## Strategies (after Hinton 2007)

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

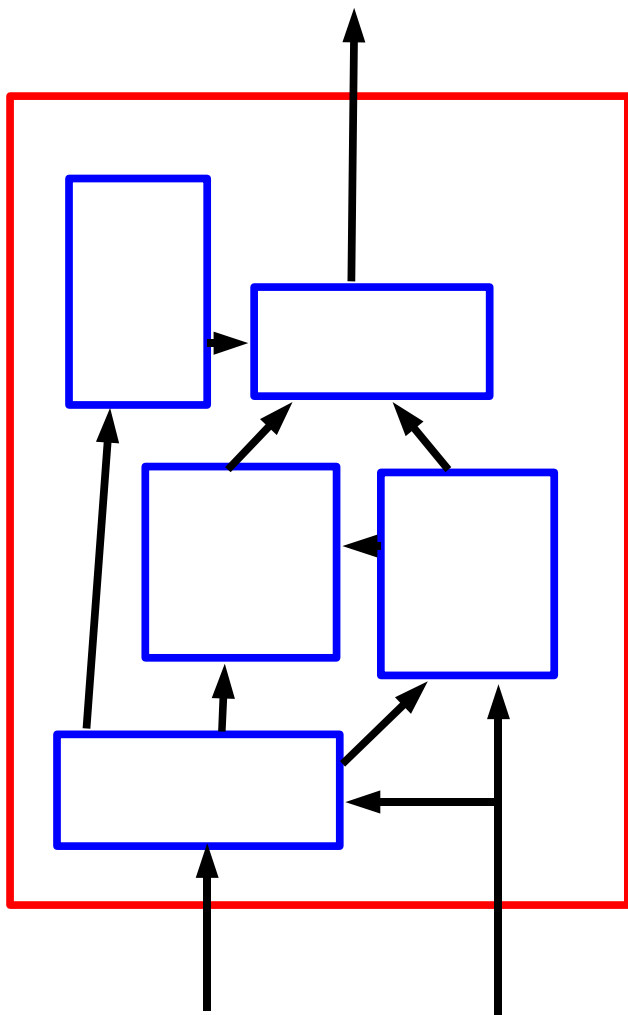
# Back-propagation: deep supervised gradient-based learning

To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for  $\frac{\partial E}{\partial X_i}$



- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....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**

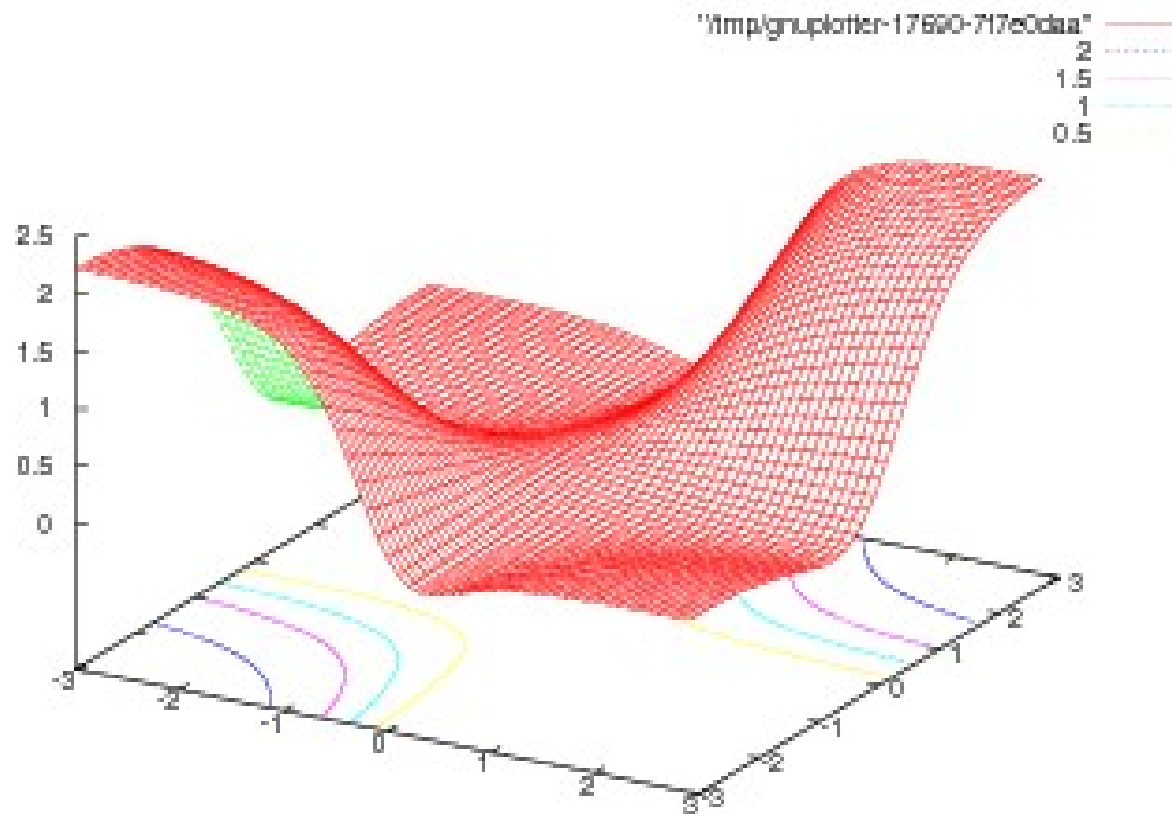
- ▶ As long as it is continuous and differentiable almost everywhere with respect to the parameters, and with respect to non-terminal inputs.

# 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:

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



# Deep Supervised Learning is Hard

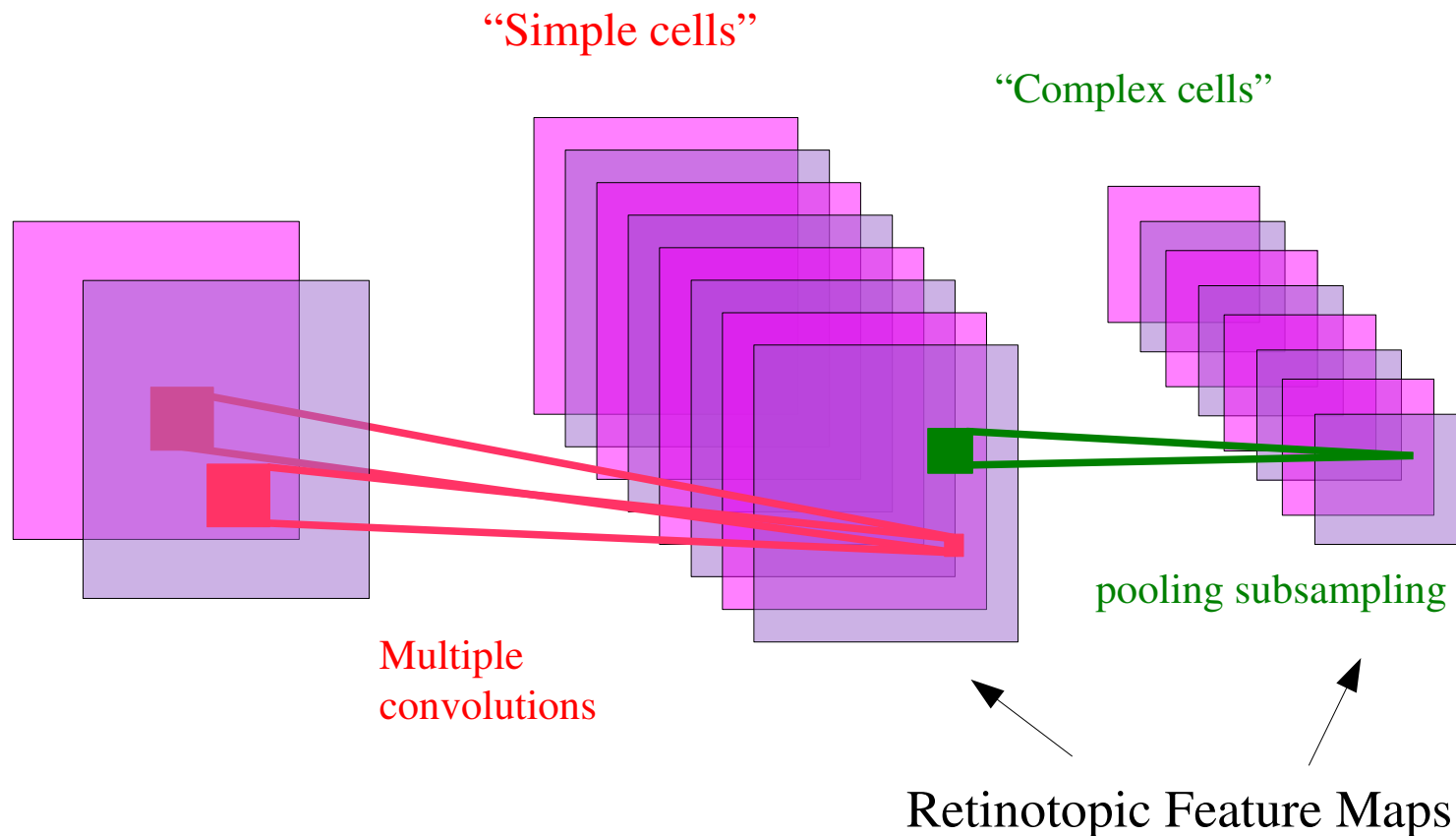
- **The loss surface is non-convex, ill-conditioned, has saddle points, has flat spots.....**
- **For large networks, it will be horrible!**
- **Back-prop doesn't work well with networks that are tall and skinny.**
  - ▶ Lots of layers with few hidden units.
- **Back-prop works fine with short and fat 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.



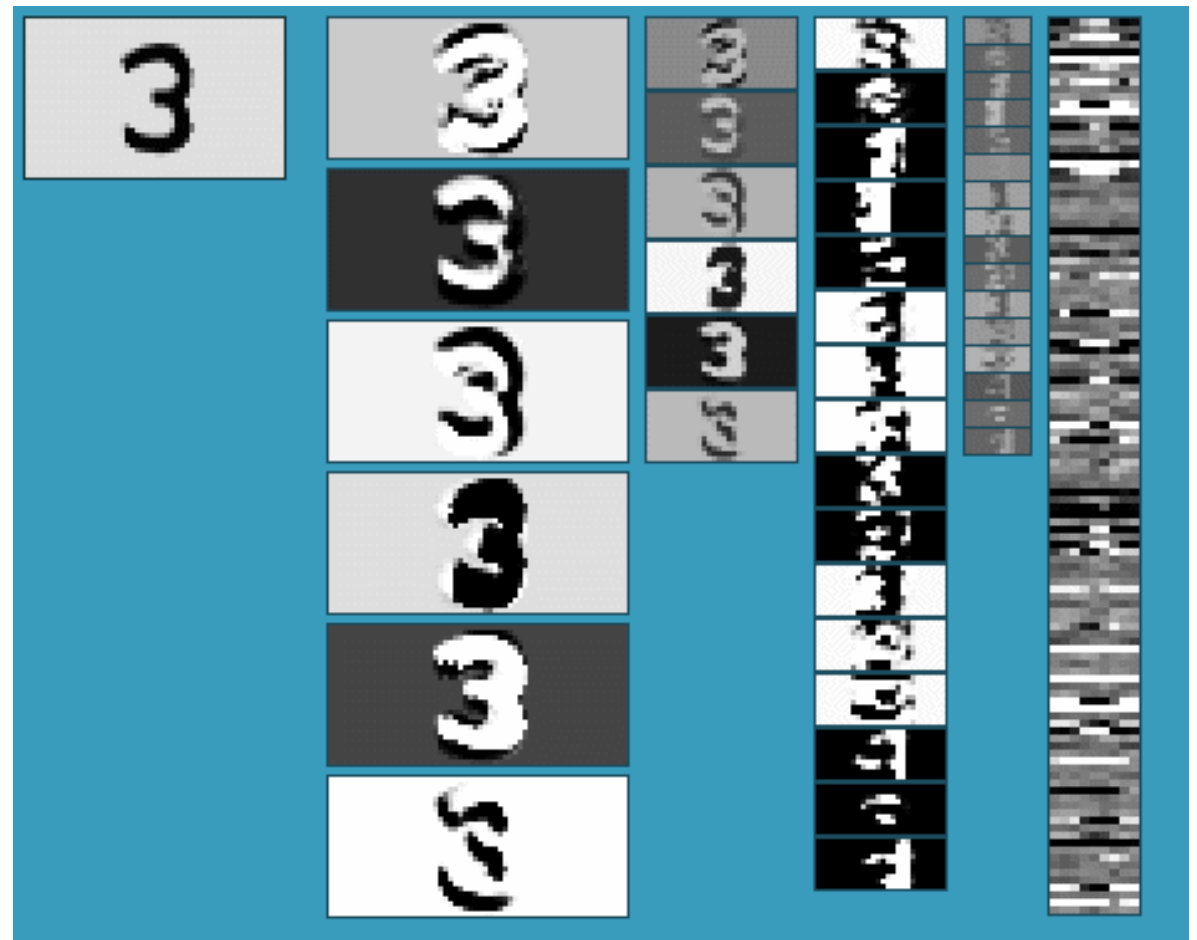


# The Multistage Hubel-Wiesel Architecture

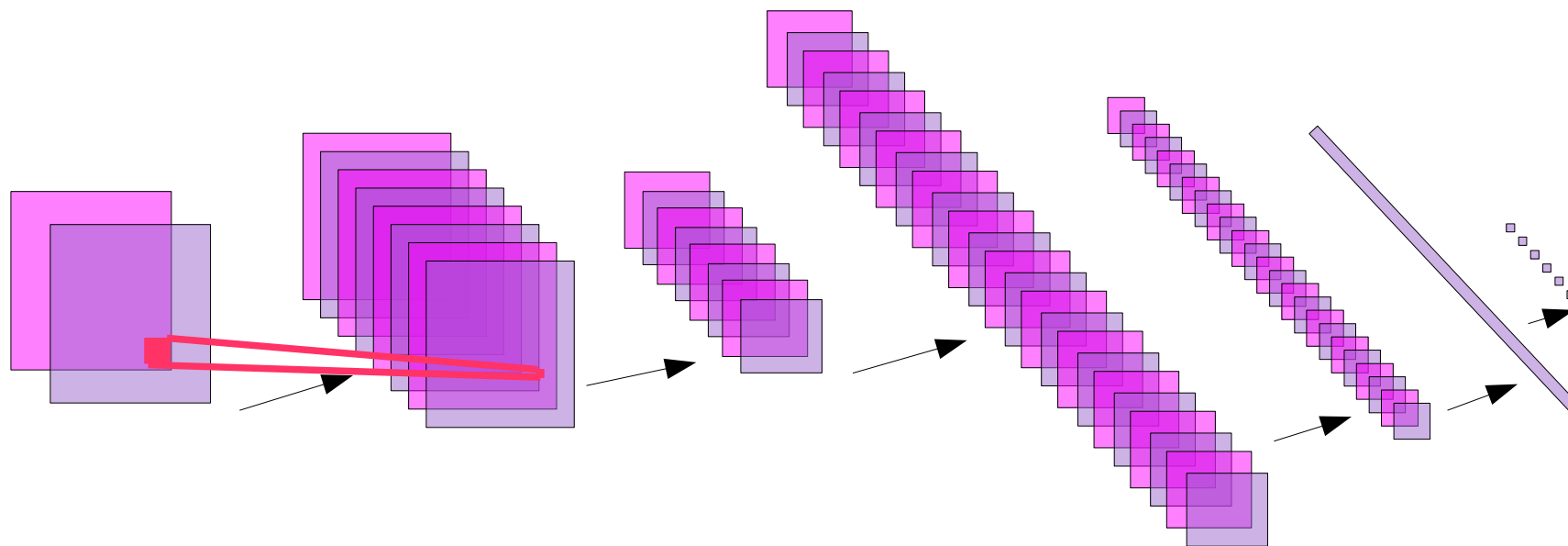
## Building a complete artificial vision system:

- ▶ 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?**

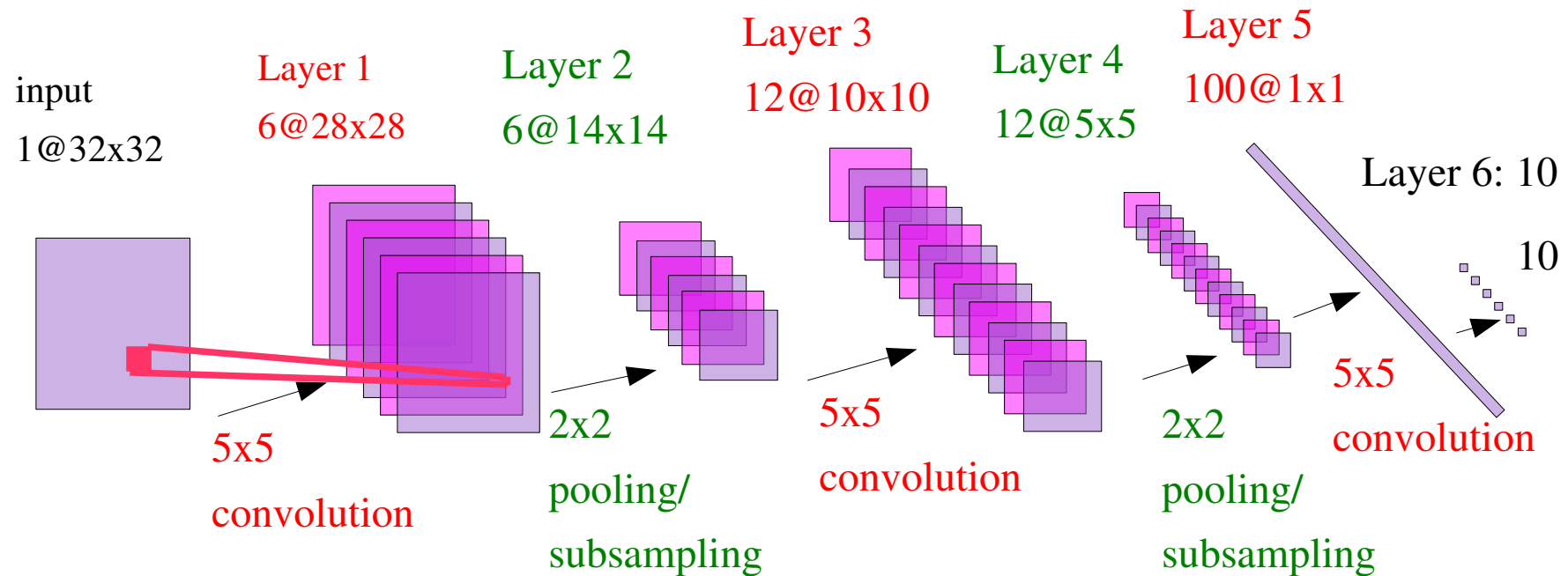


# Getting Inspiration from Biology: Convolutional Network



- **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.

# 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

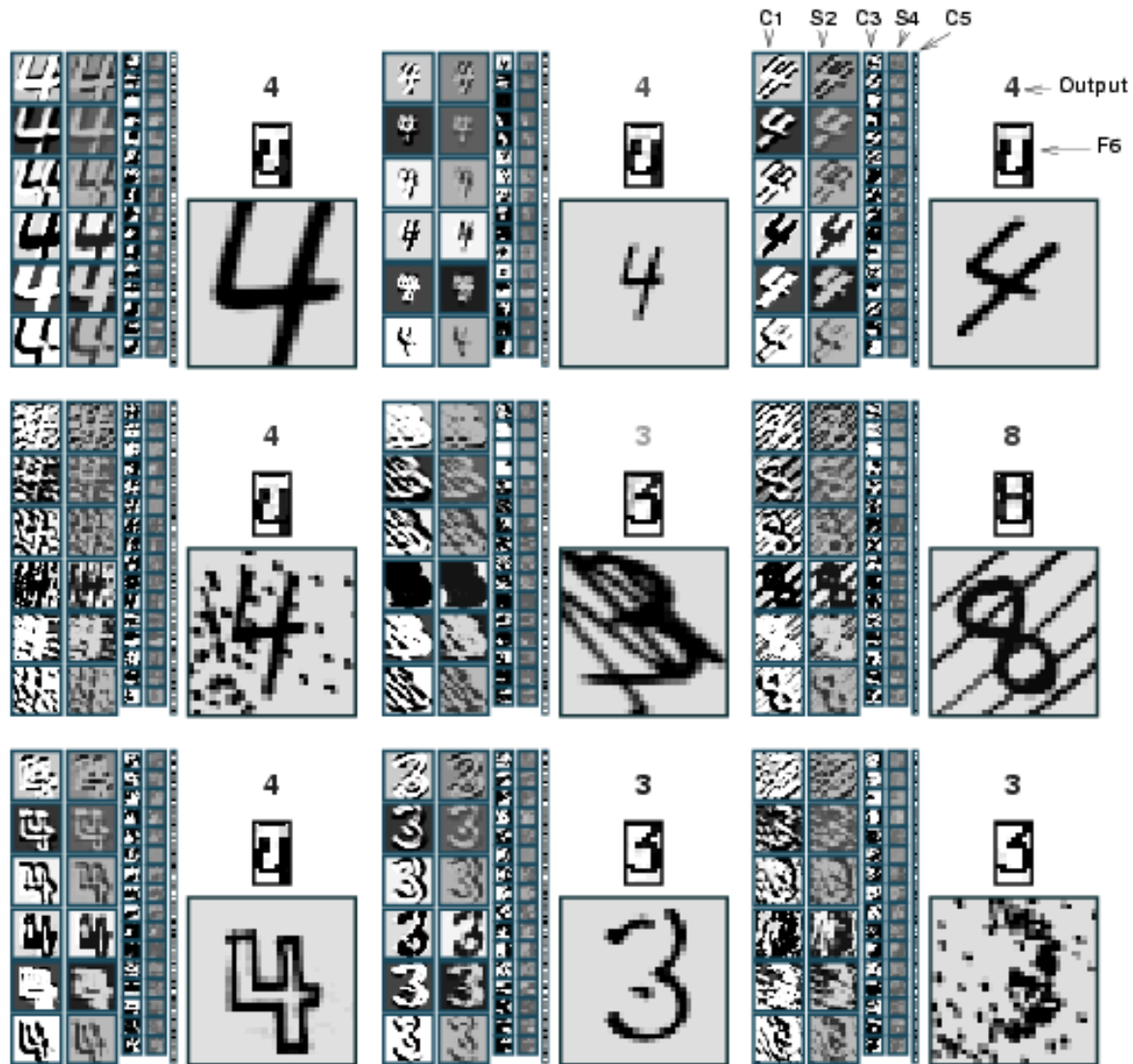
CLASSIFIER	DEFORMATION	PREPROCESSING	ERROR (%)	Reference
linear classifier (1-layer NN)		none	12.00	LeCun et al. 1998
linear classifier (1-layer NN)		deskewing	8.40	LeCun et al. 1998
pairwise linear classifier		deskewing	7.60	LeCun et al. 1998
K-nearest-neighbors, (L2)		none	3.09	Kenneth Wilder, U. Chicago
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> (a fixed 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

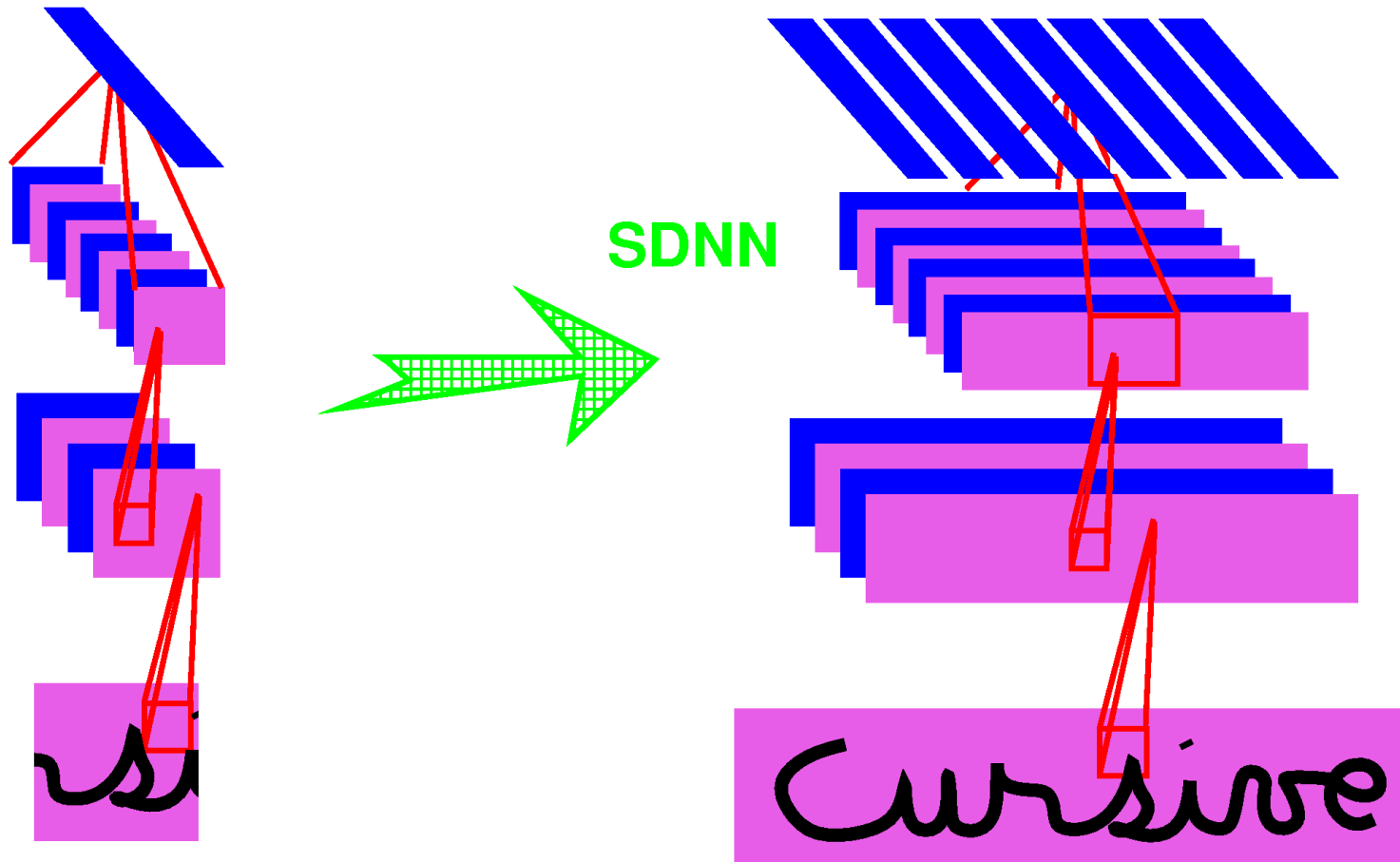
Note: some groups have obtained good results with various amounts of preprocessing such as deskewing (e.g. 0.56% using an SVM with smart kernels [deCoste and Schoelkopf]) hand-designed feature representations (e.g. 0.63% with “shape context” and nearest neighbor [Belongie])

# Invariance and Robustness to Noise





# Recognizing Multiple Characters with Replicated Nets

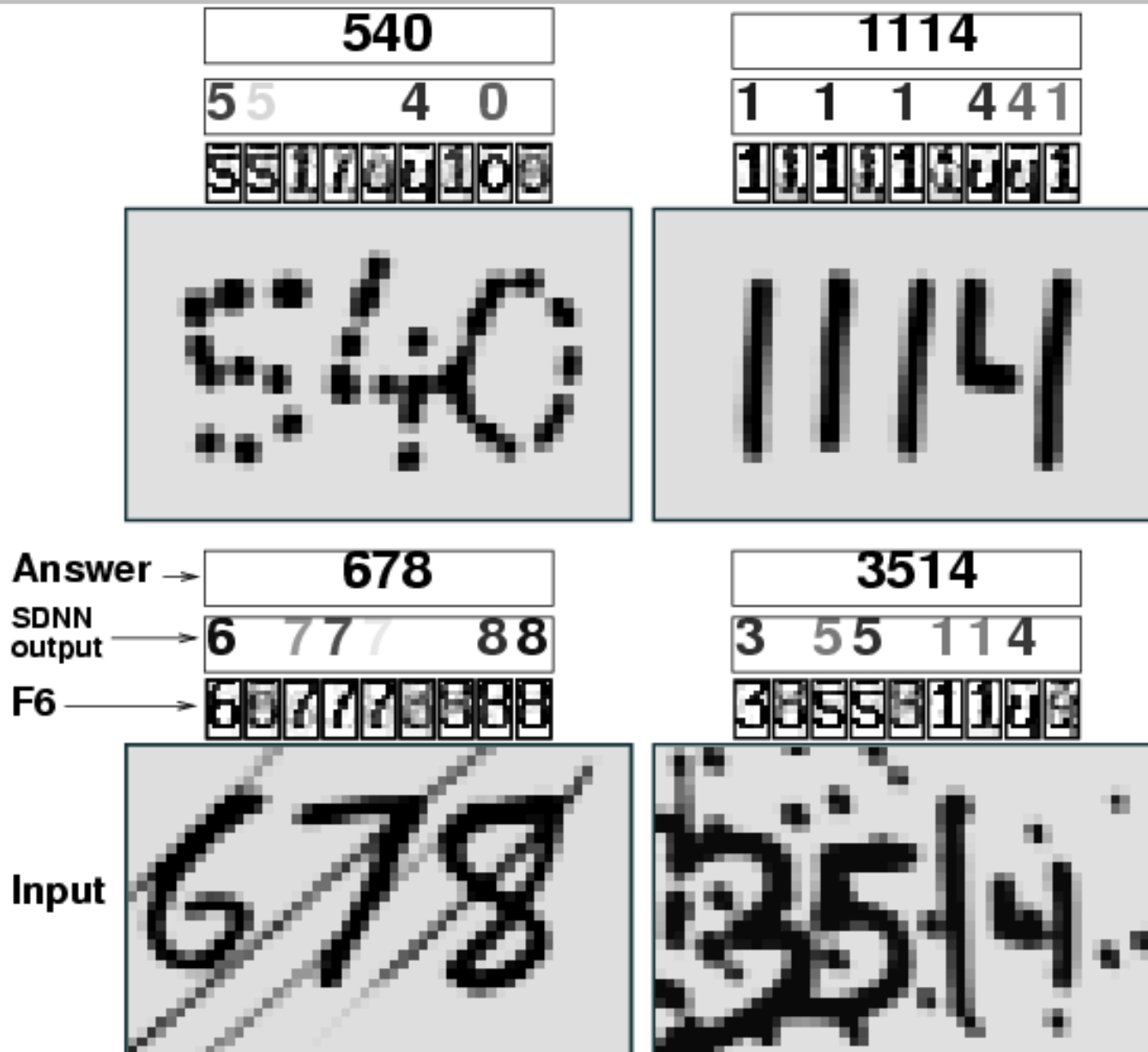




# Recognizing Multiple Characters with Replicated Nets

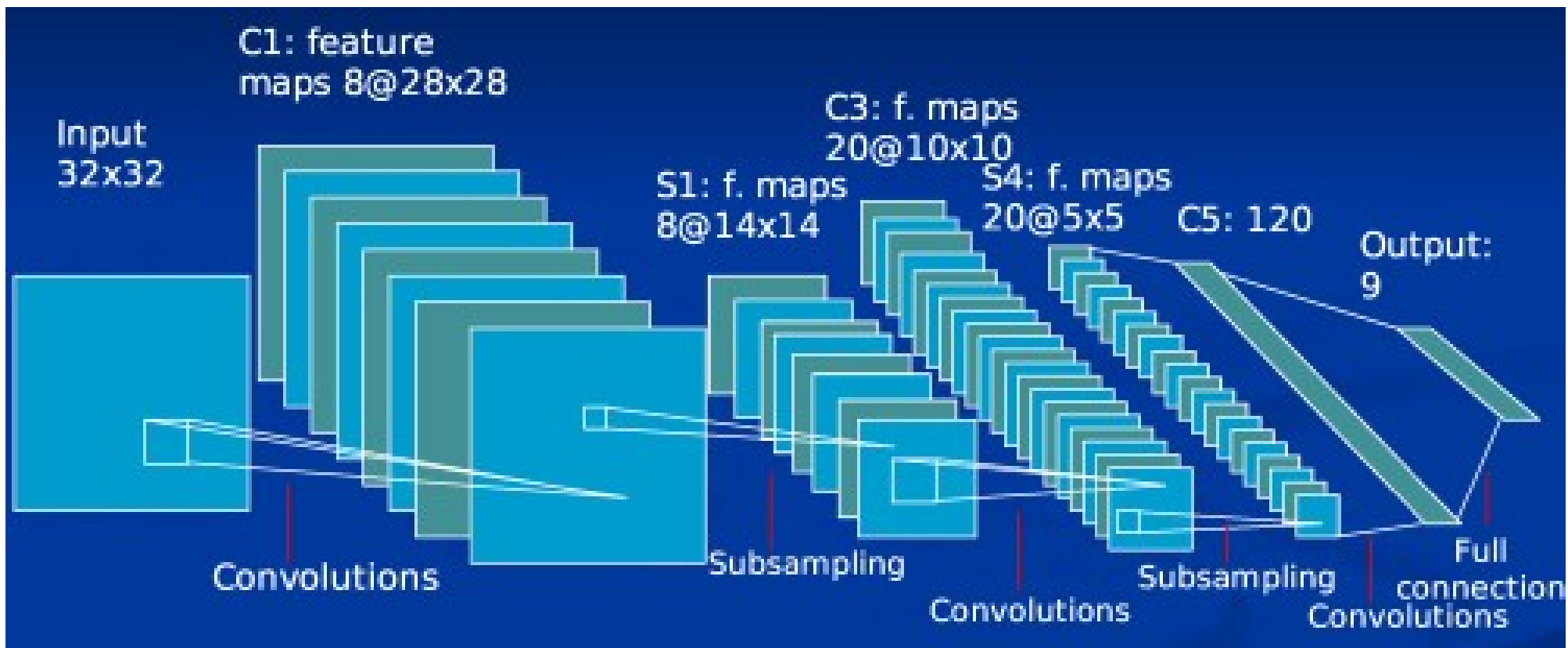


# Handwriting Recognition



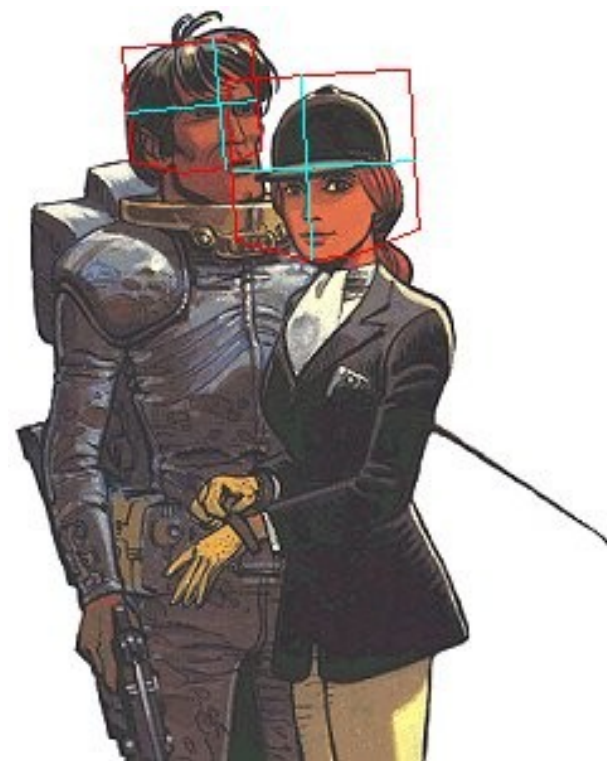
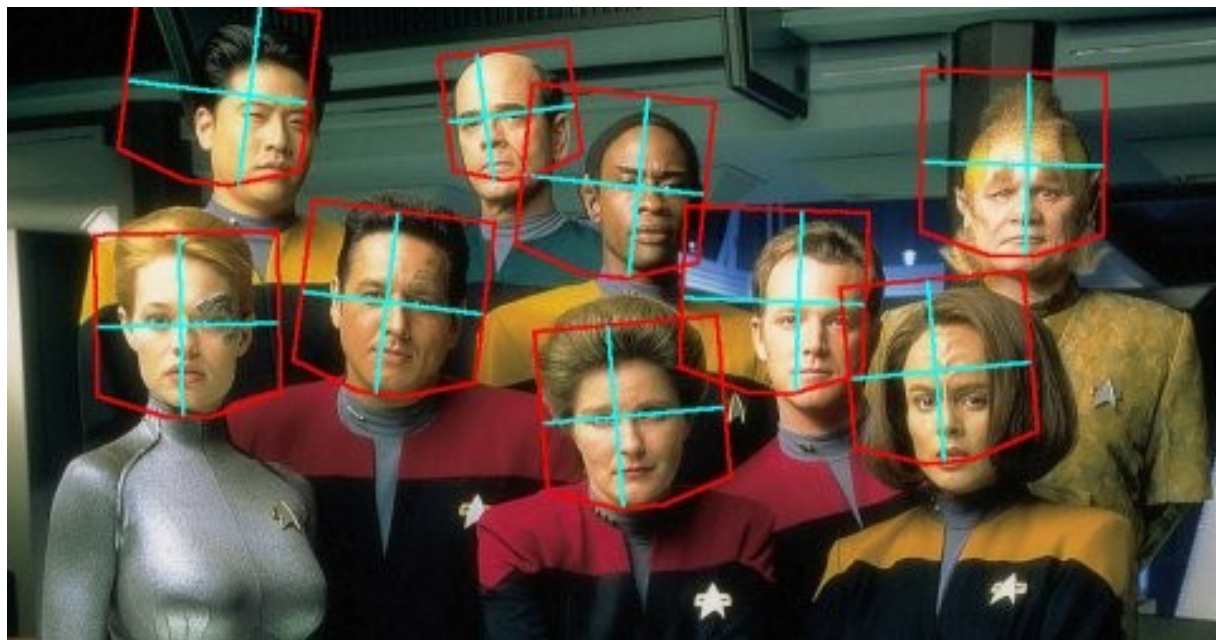
# Face Detection and Pose Estimation with Convolutional Nets

- **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 .



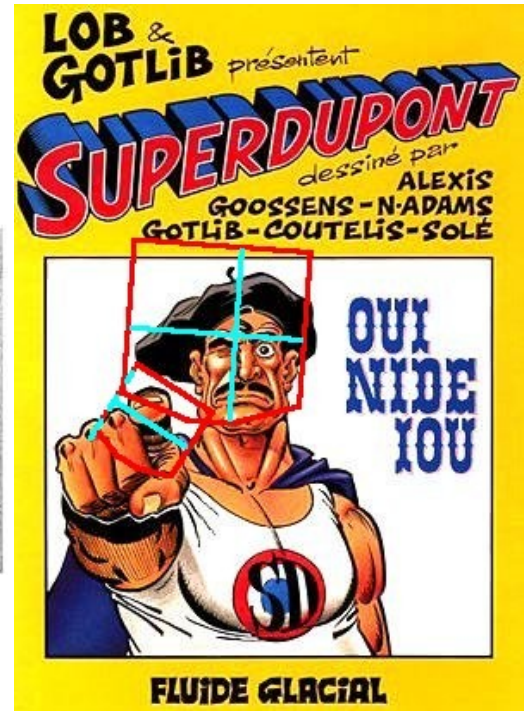
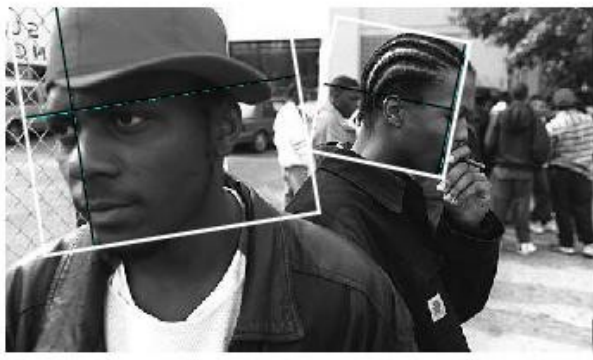
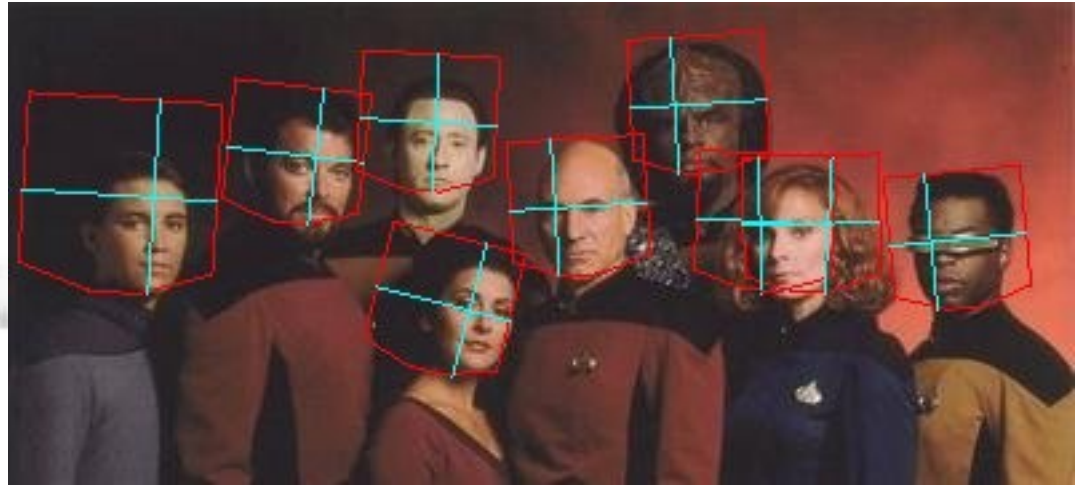
# Face Detection: Results

<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



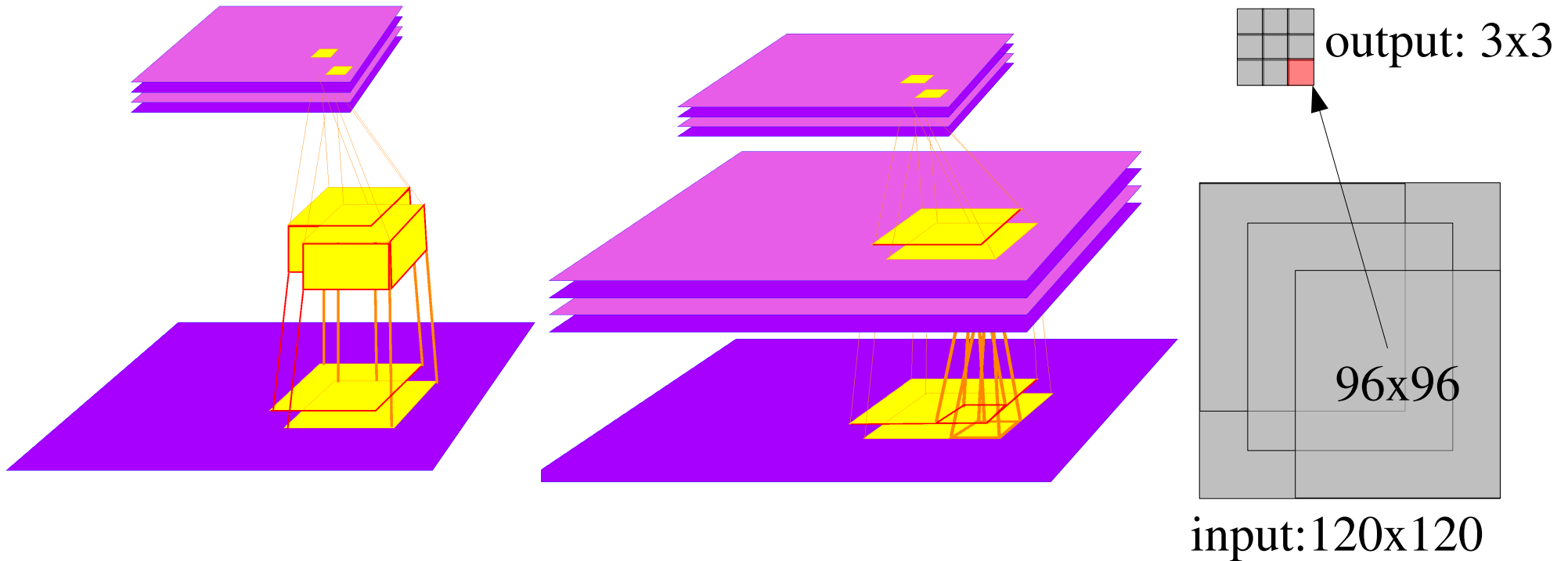


# Face Detection with a Convolutional Net





# 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

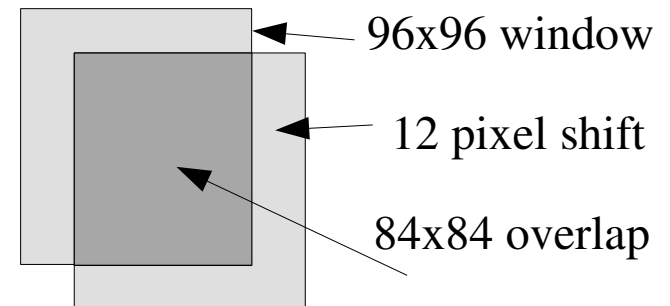
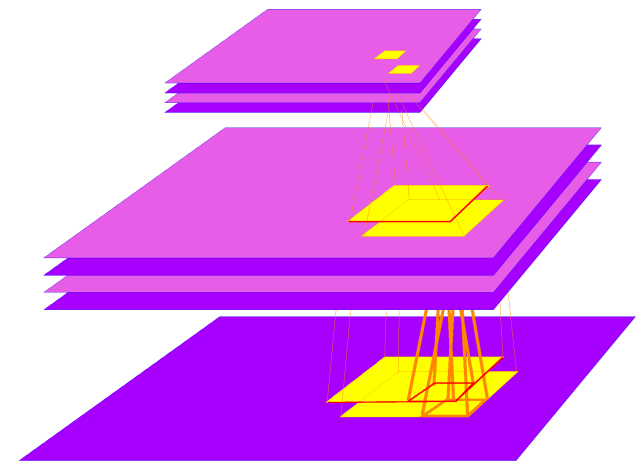
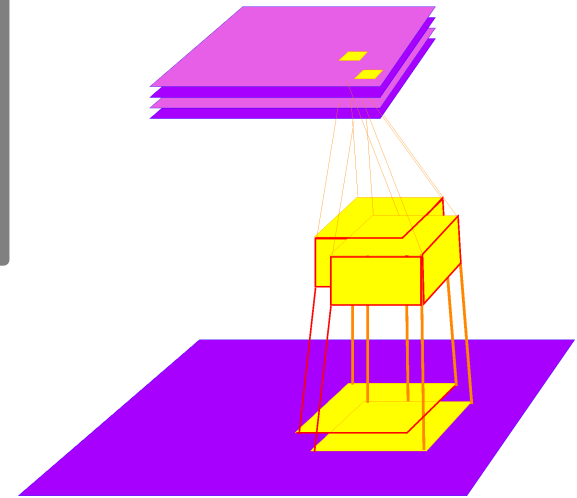
● Computational cost for a non-convolutional detector of the same size, applied every 12 pixels:

● 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

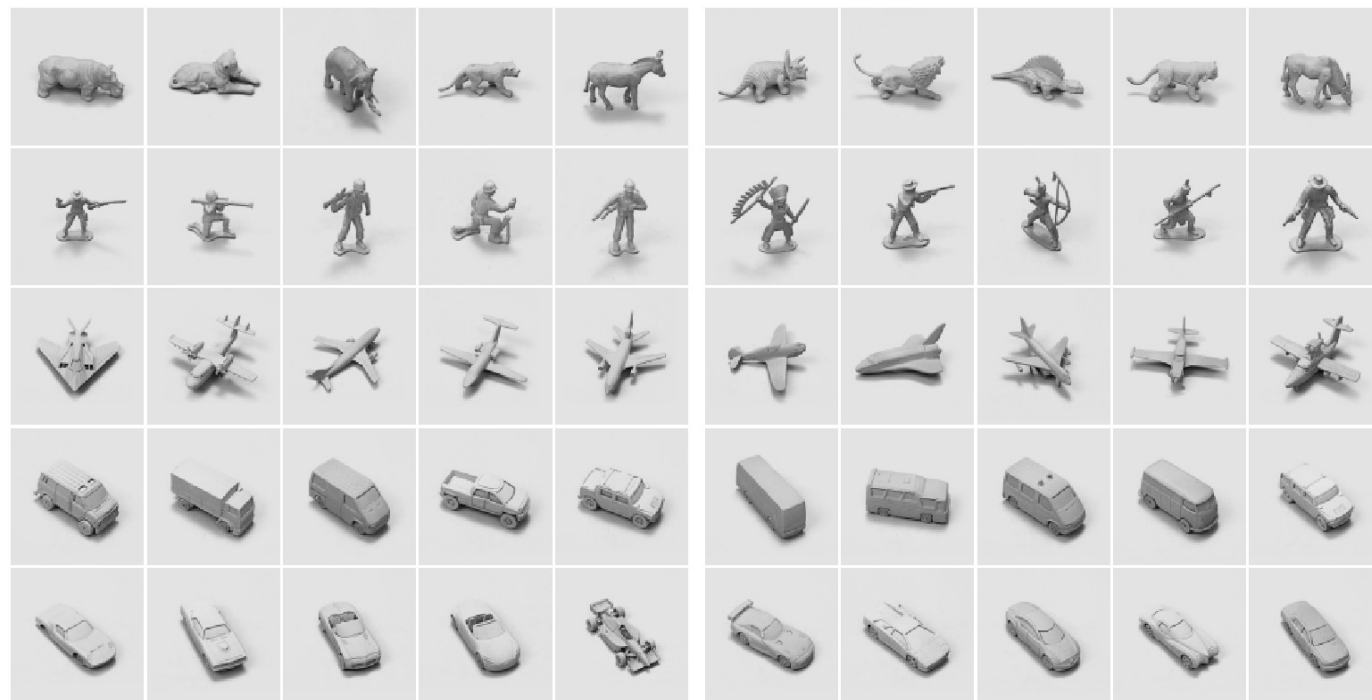
**6 illuminations**

on/off combinations of 4 lights

**2 cameras (stereo)**

7.5 cm apart

40 cm from the object

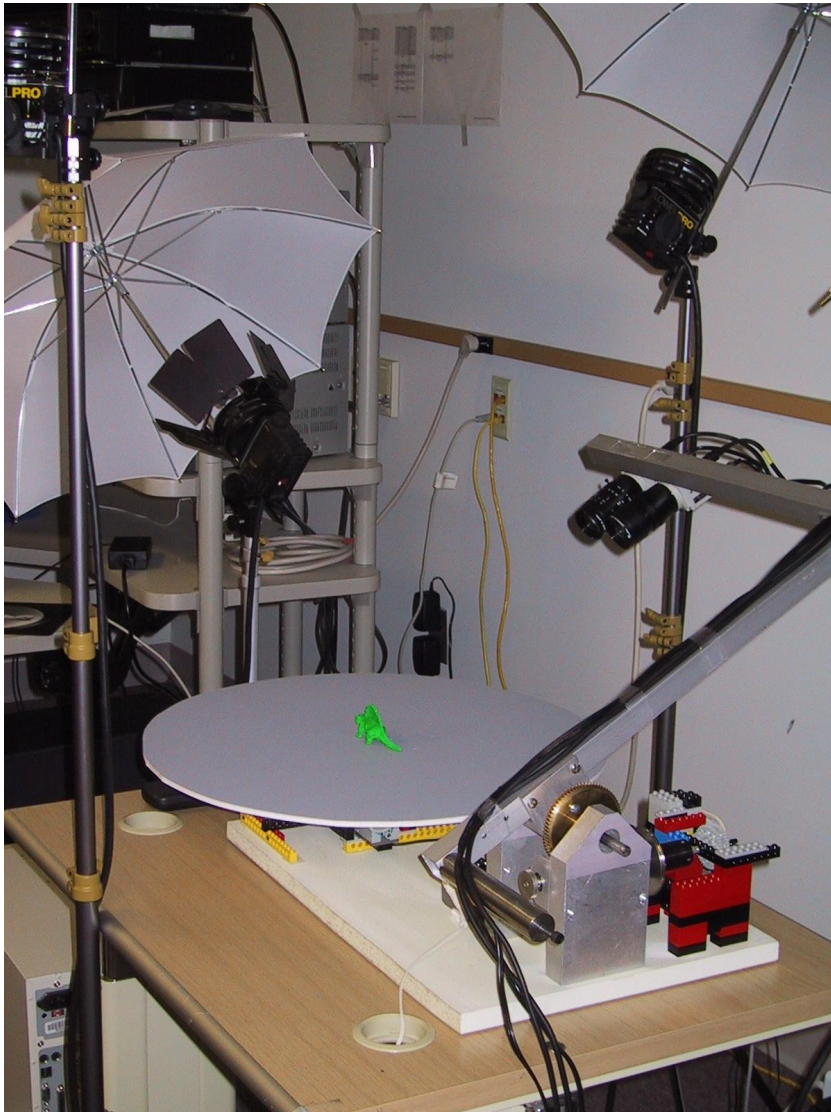


Training instances

Test instances

# Data Collection, Sample Generation

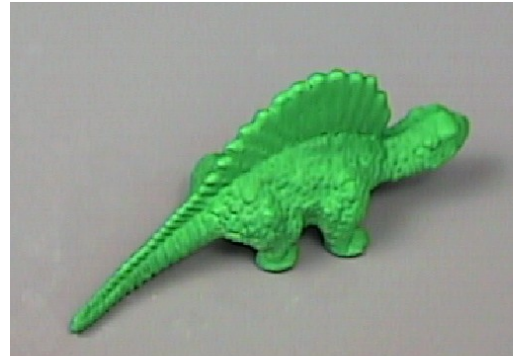
## Image capture setup



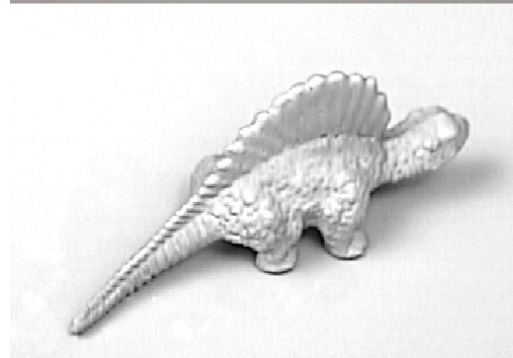
Objects are painted green so that:

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

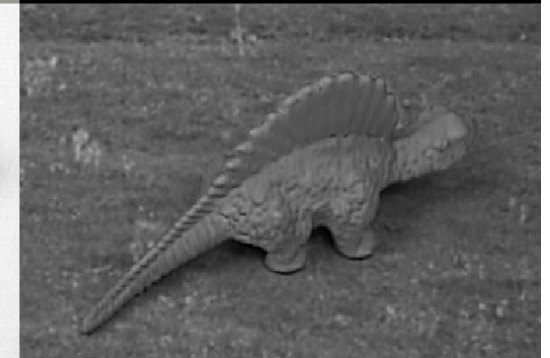
Original image



Object mask



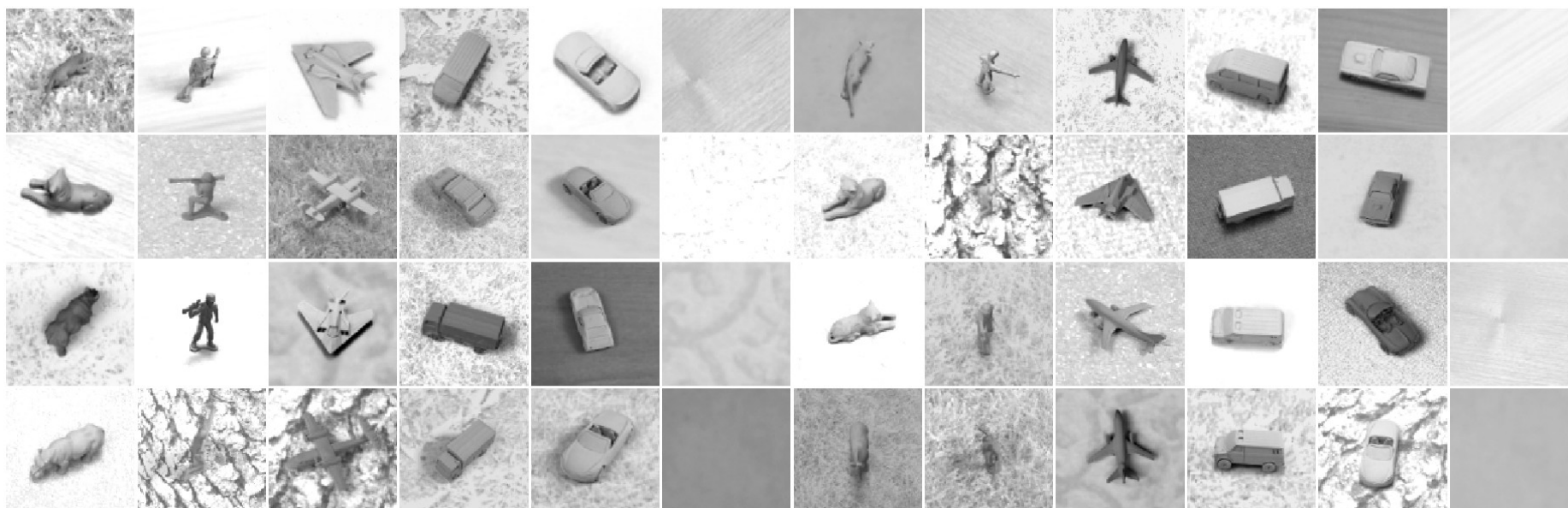
Shadow factor



Composite image

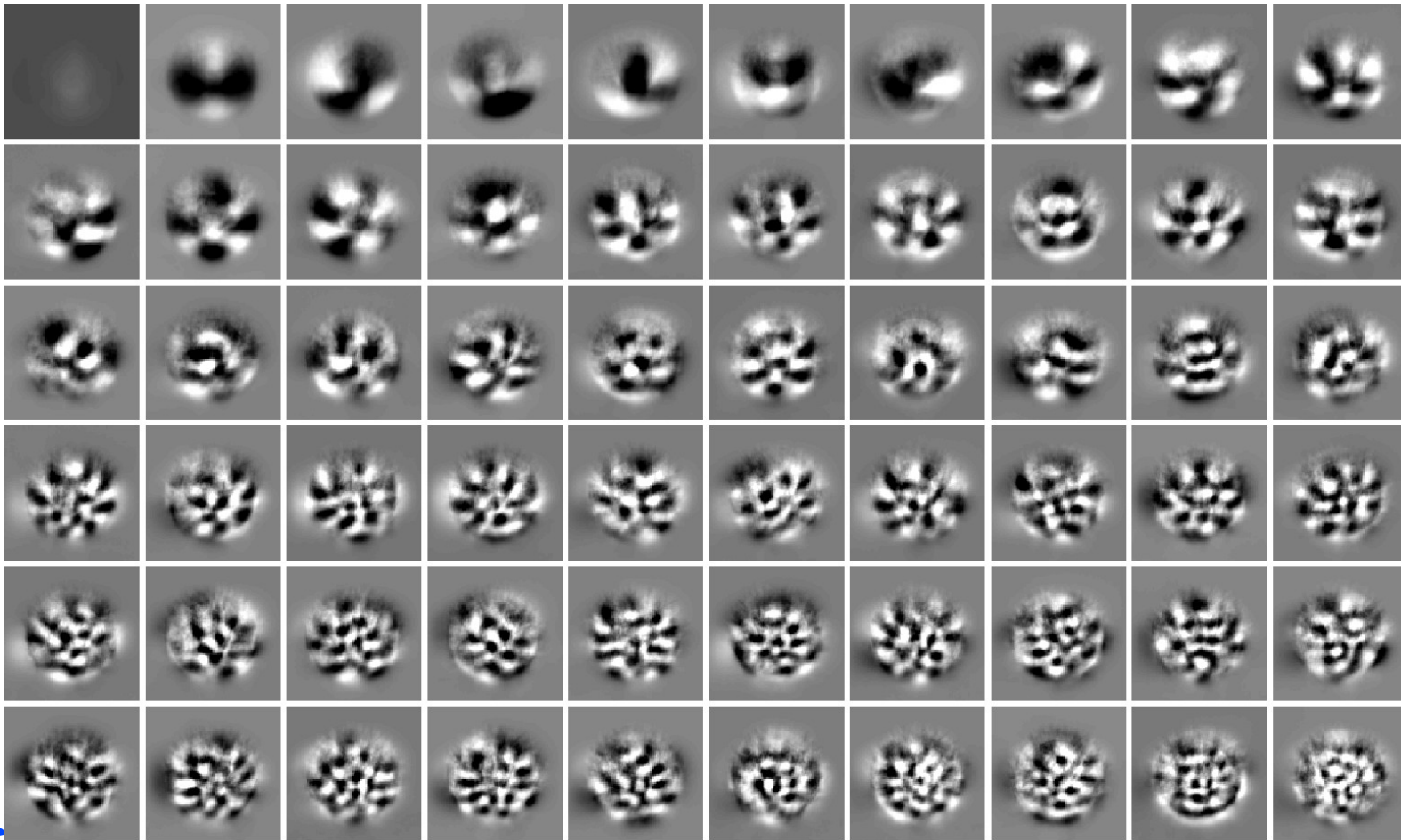


# Textured and Cluttered Datasets



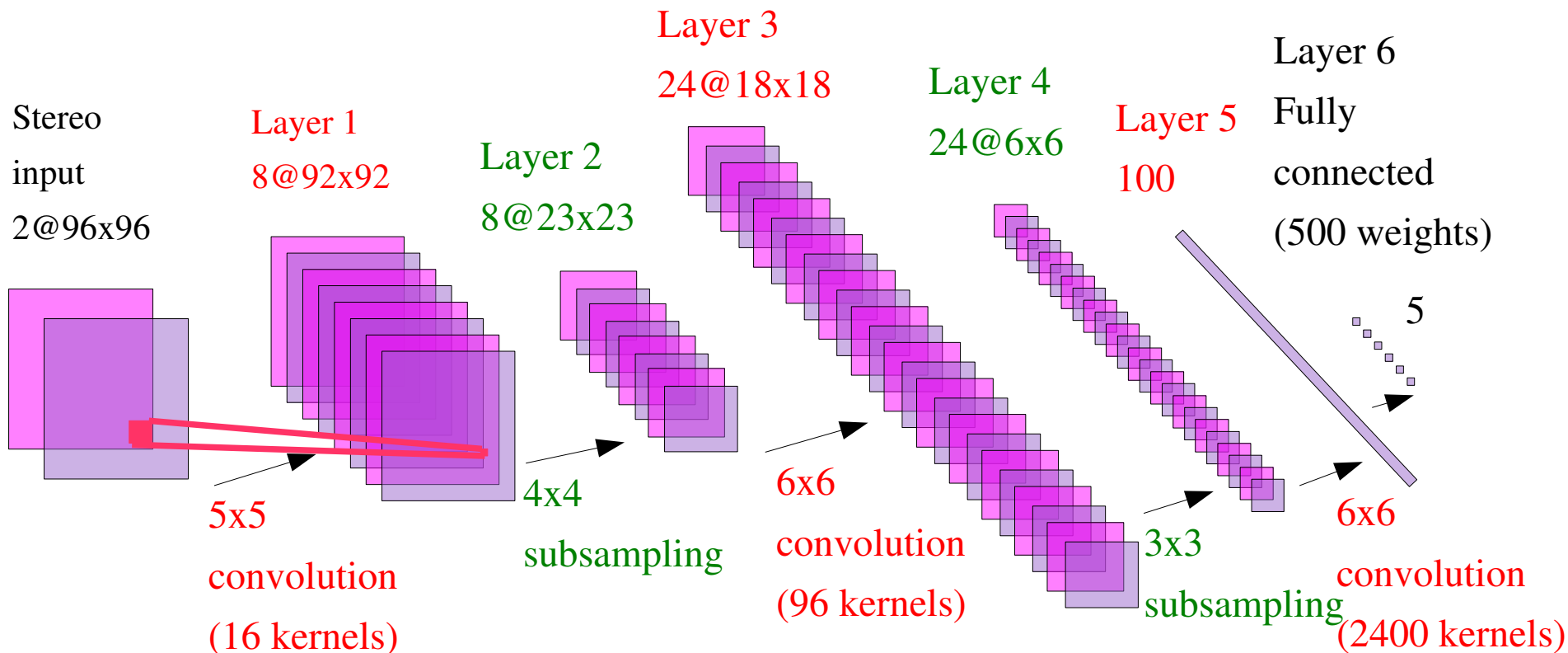
# Experiment 1: Normalized-Uniform Set: Representations

- 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.

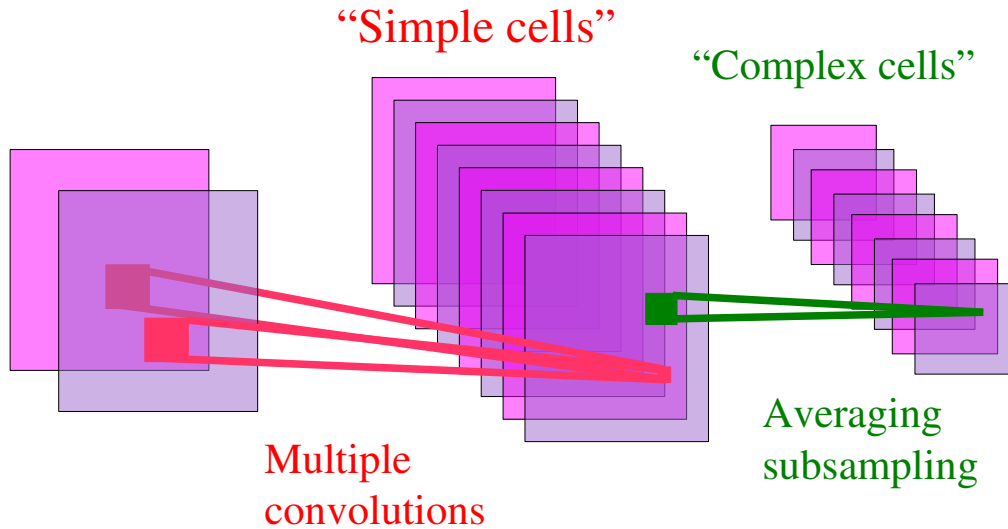
The architecture alternates **convolutional layers** (feature detectors) and **subsampling layers** (local feature pooling for invariance to small distortions).

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

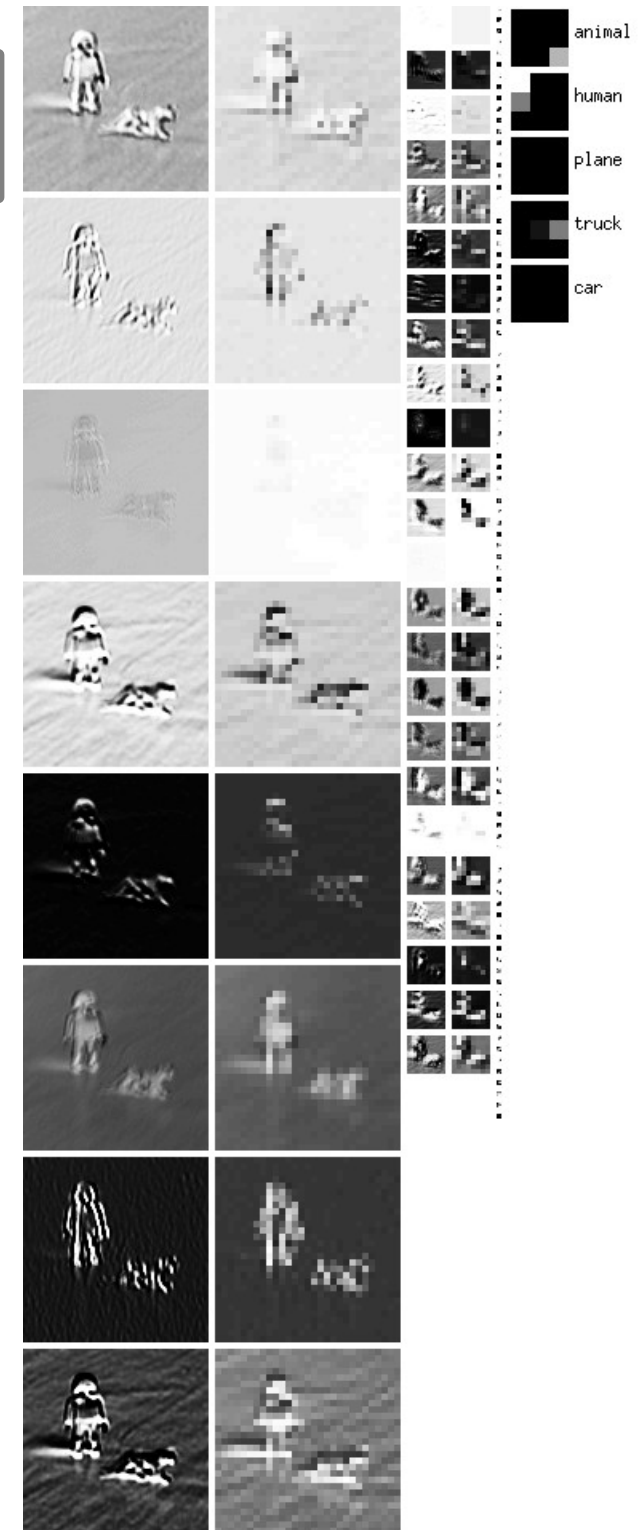
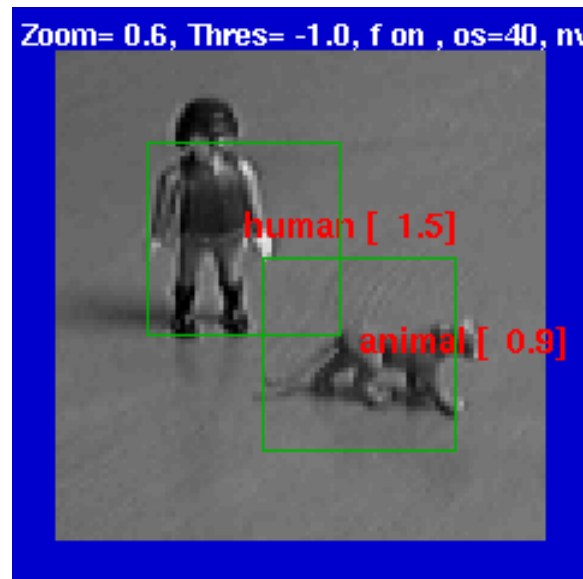
A gradient-based algorithm is used to minimize a supervised loss function.



# Alternated Convolutions and Subsampling

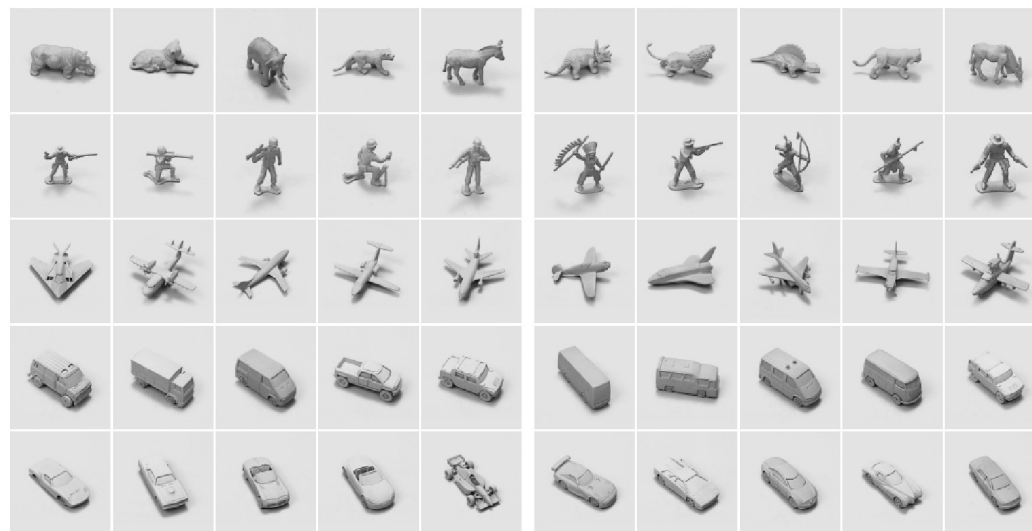


- Local features are extracted everywhere.
- averaging/subsampling layer builds robustness to variations in feature locations.
- Hubel/Wiesel'62, Fukushima'71, LeCun'89, Riesenhuber & Poggio'02, Ullman'02,....



# 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:  $98 \times 98 \times 2$  (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

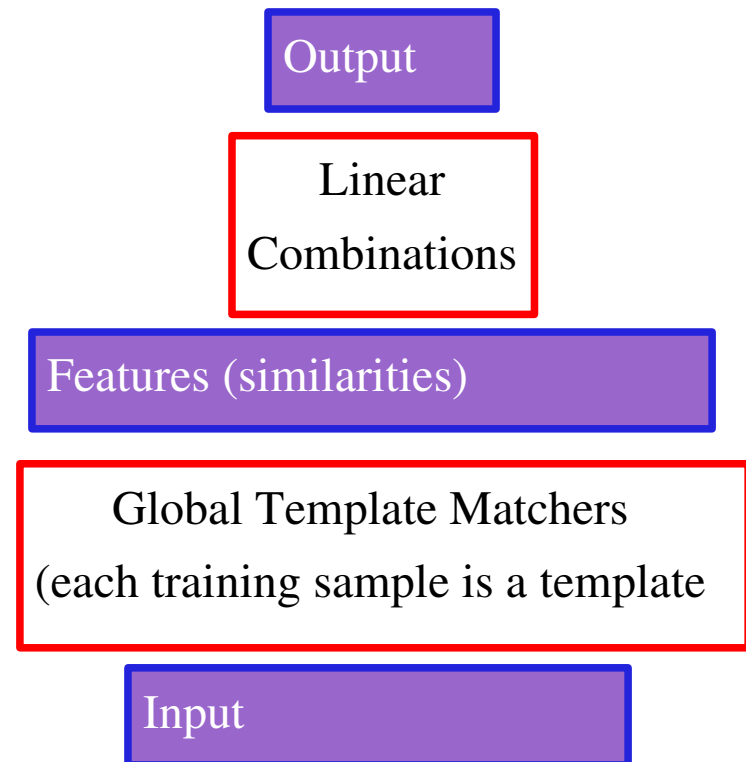
Chop off the last layer,  
and train an SVM on it  
it works!

# 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.

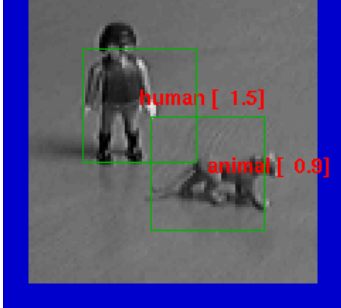
- Global templates are in trouble when the variations include: category, instance shape, configuration (for articulated object), position, azimuth, elevation, scale, illumination, texture, albedo, in-plane rotation, background luminance, background texture, background clutter, .....



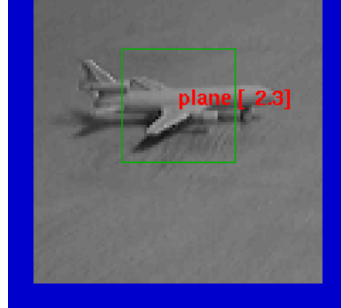


# Examples (Monocular Mode)

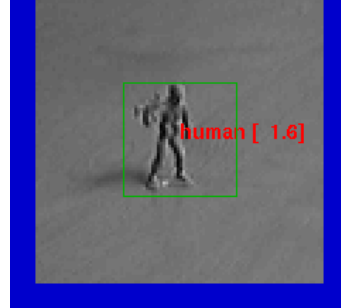
Zoom= 0.6, Thres= -1.0, f on , os=40, nv



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



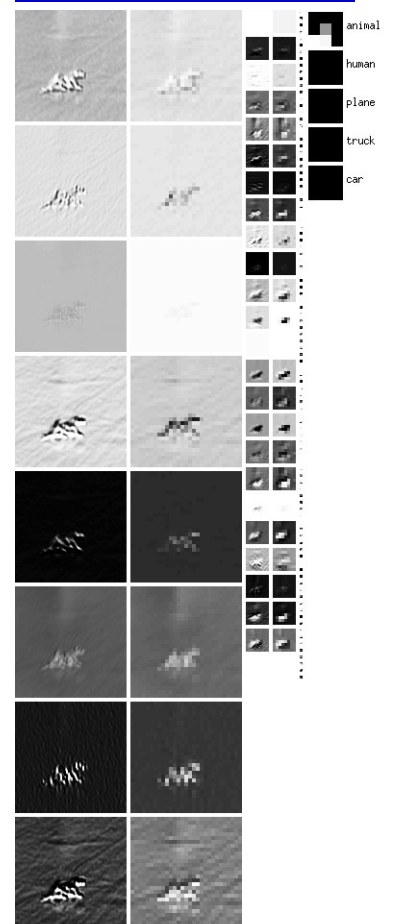
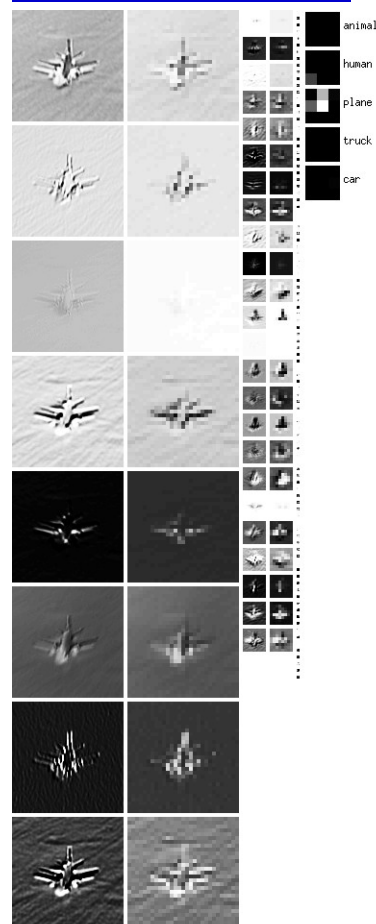
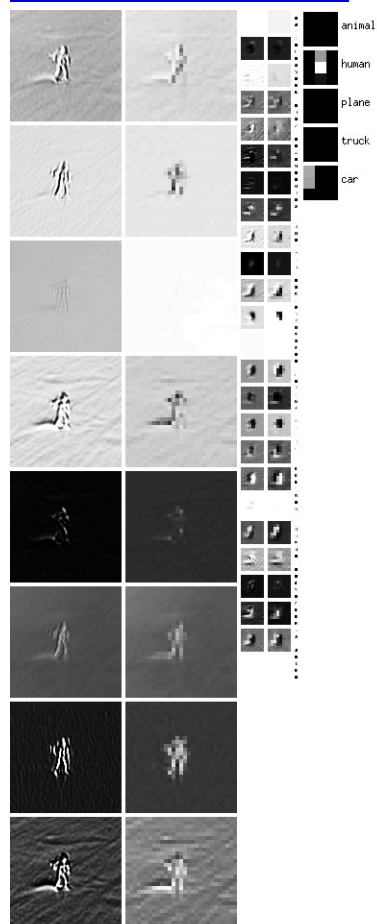
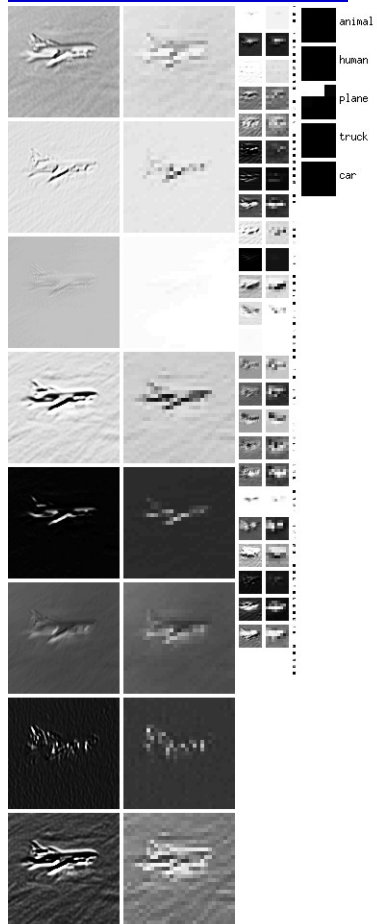
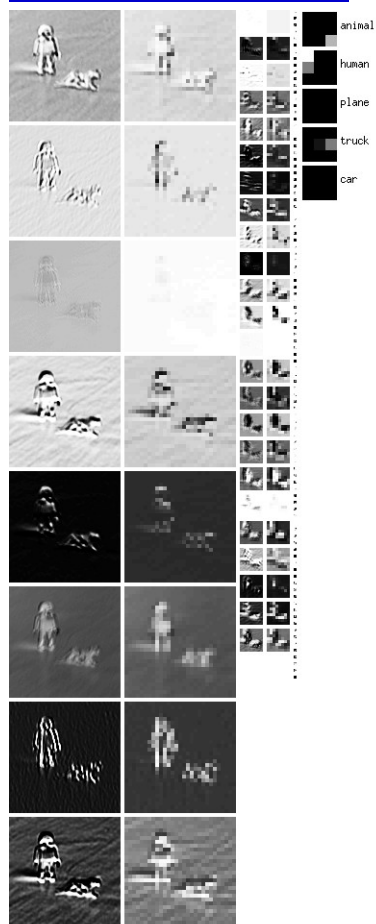
Zoom= 0.6, Thres= -1.0, f on , os=40, nv



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



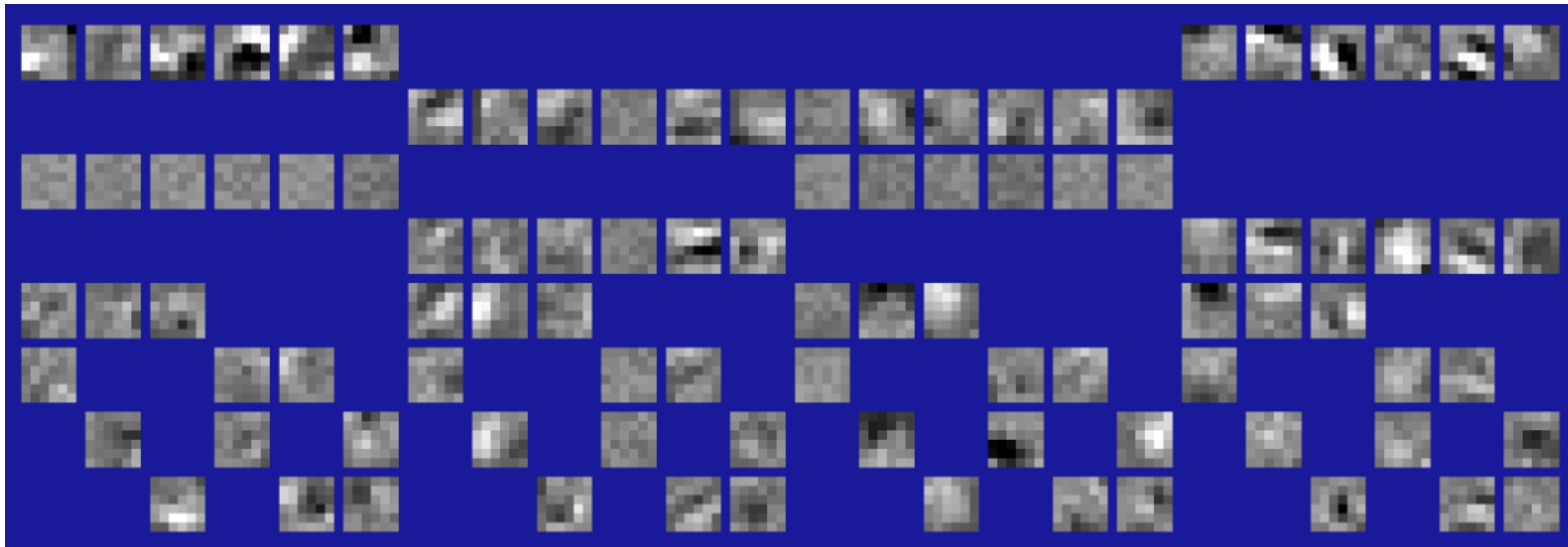
Zoom= 0.6, Thres= 0.5, f on , os=40, nv



# Learned Features

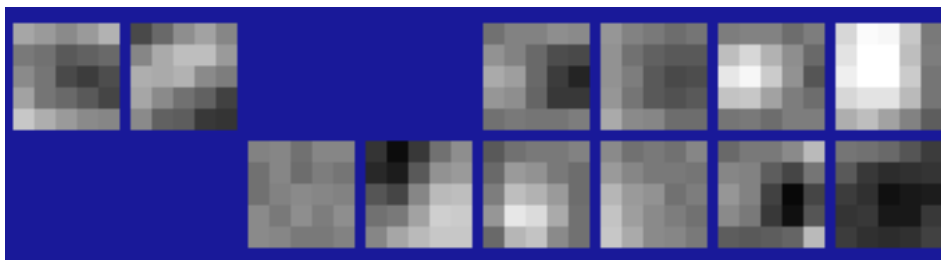
Layer 3

Layer 2



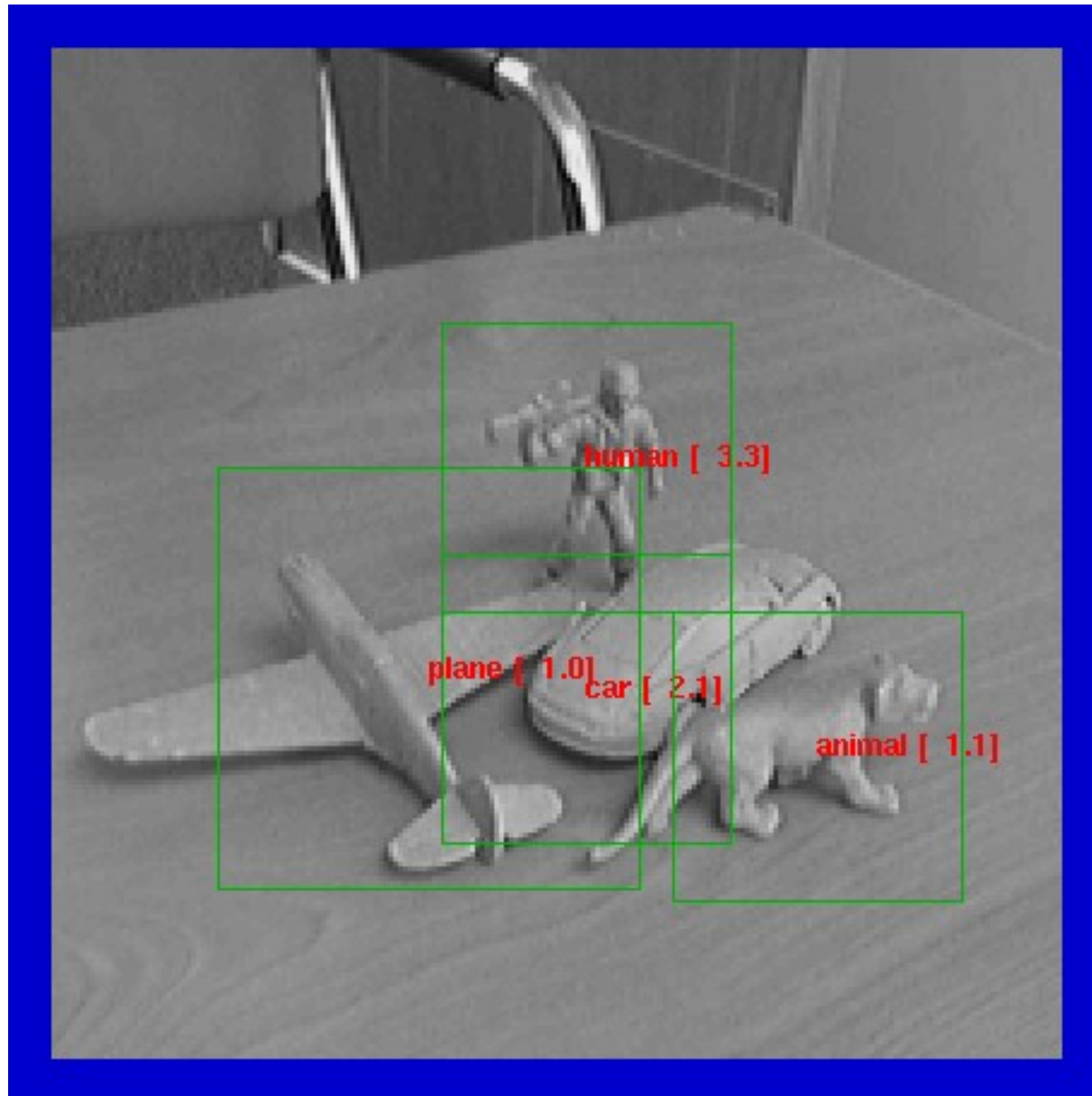
Layer 1

Input

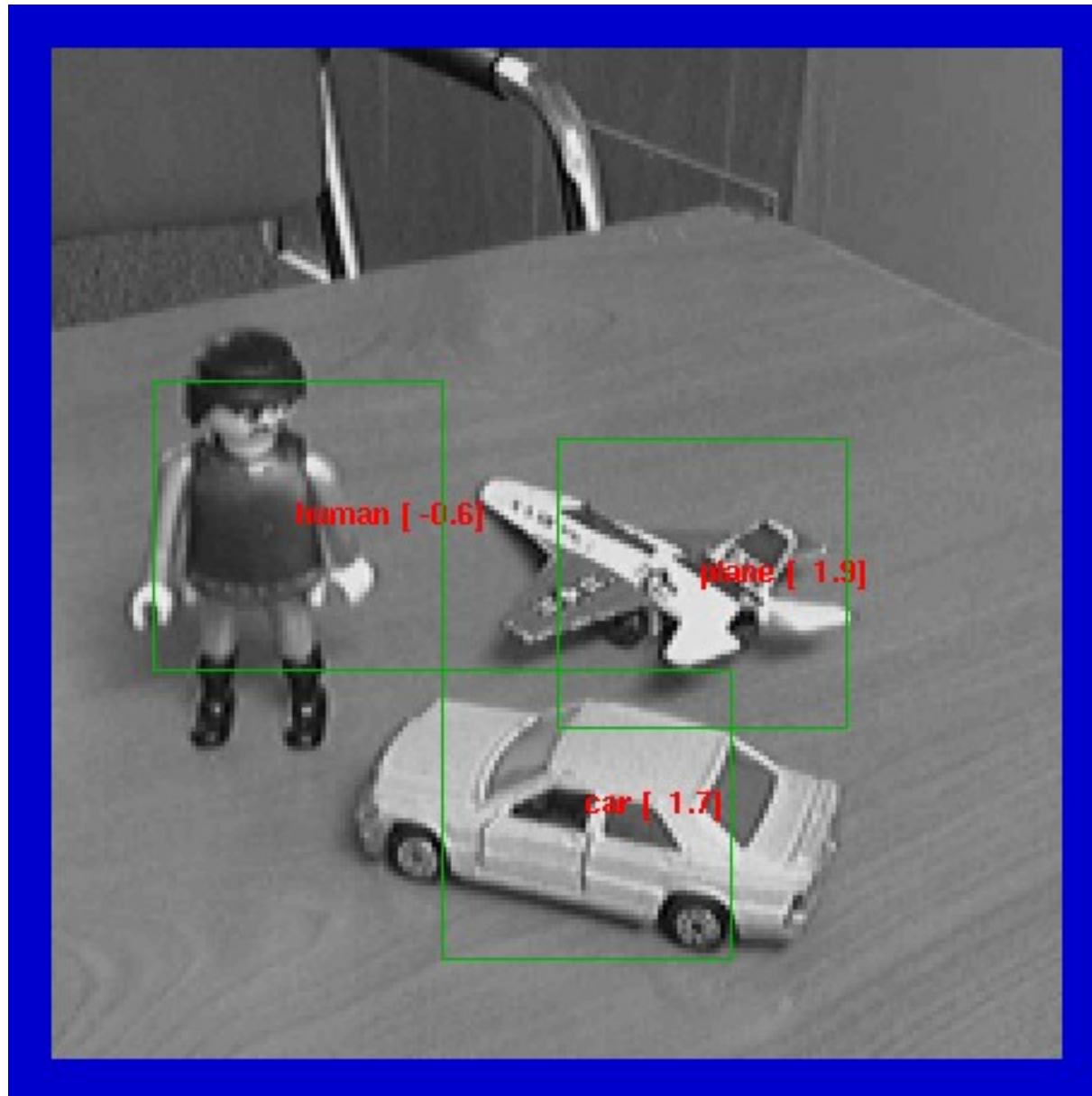




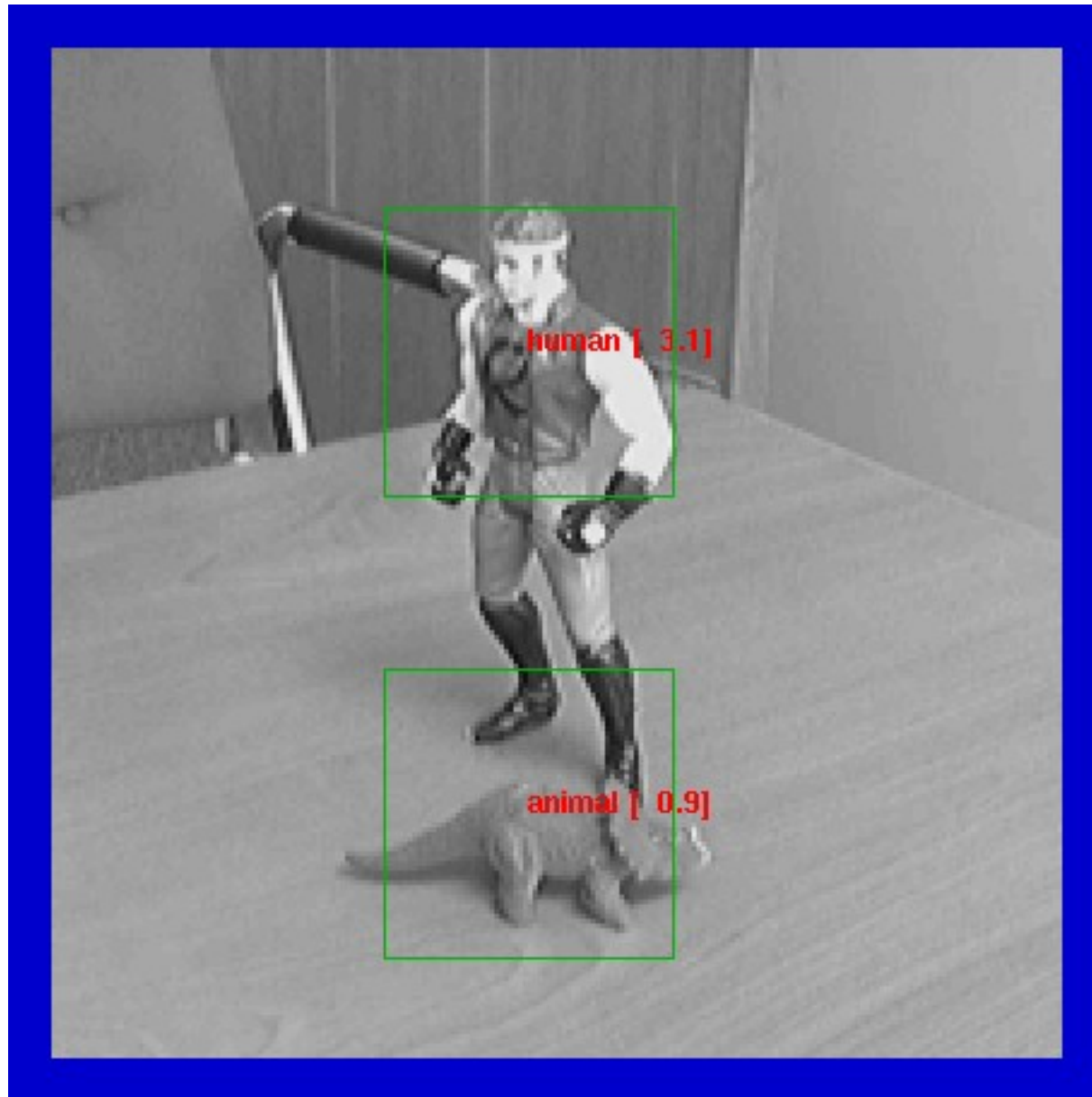
# Examples (Monocular Mode)



# Examples (Monocular Mode)

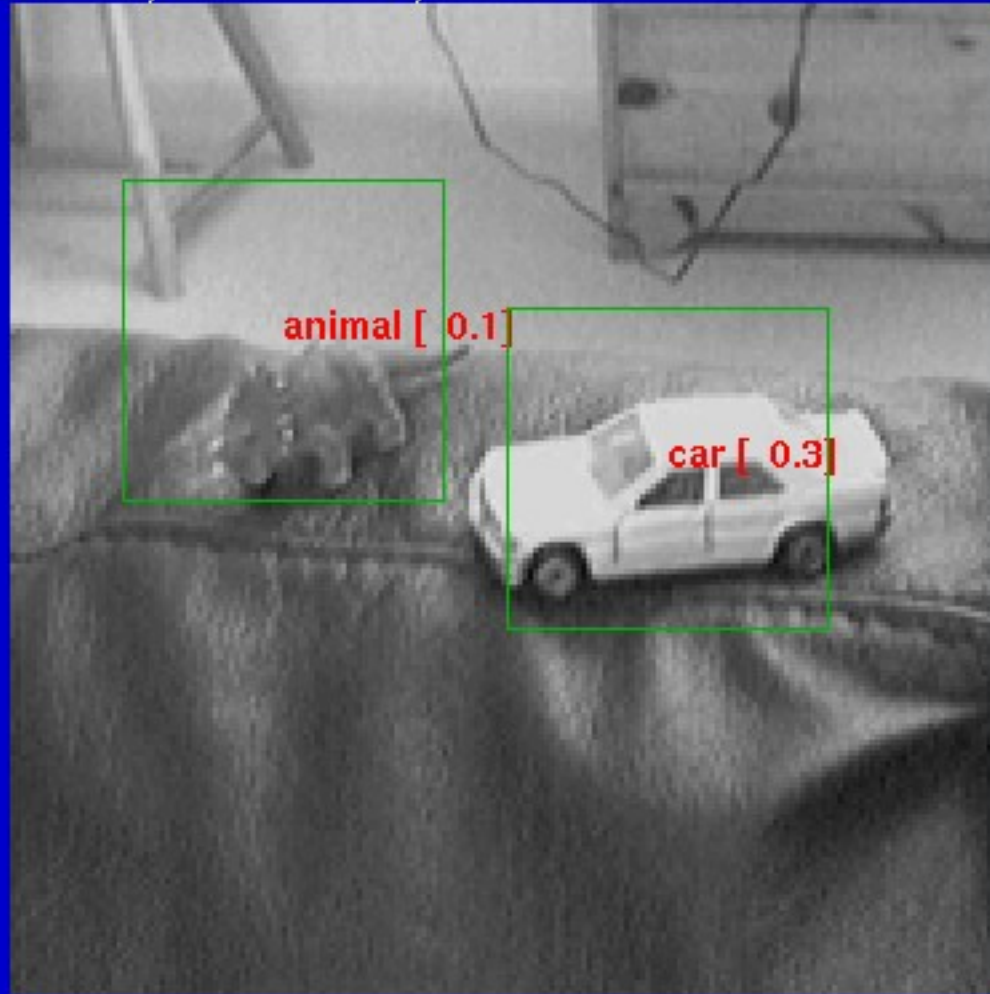


# Examples (Monocular Mode)



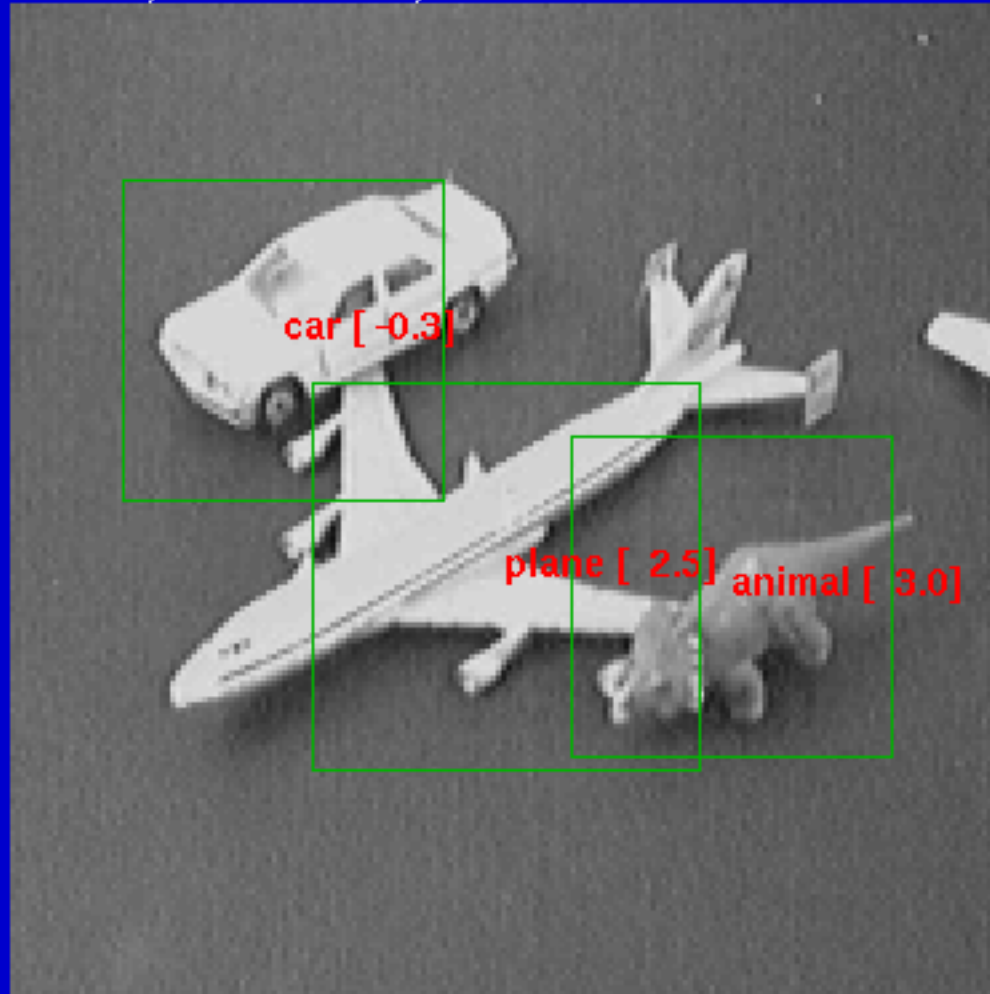
# Examples (Monocular Mode)

Zoom= 1.0, Threshold= -1.0, filter on



# Examples (Monocular Mode)

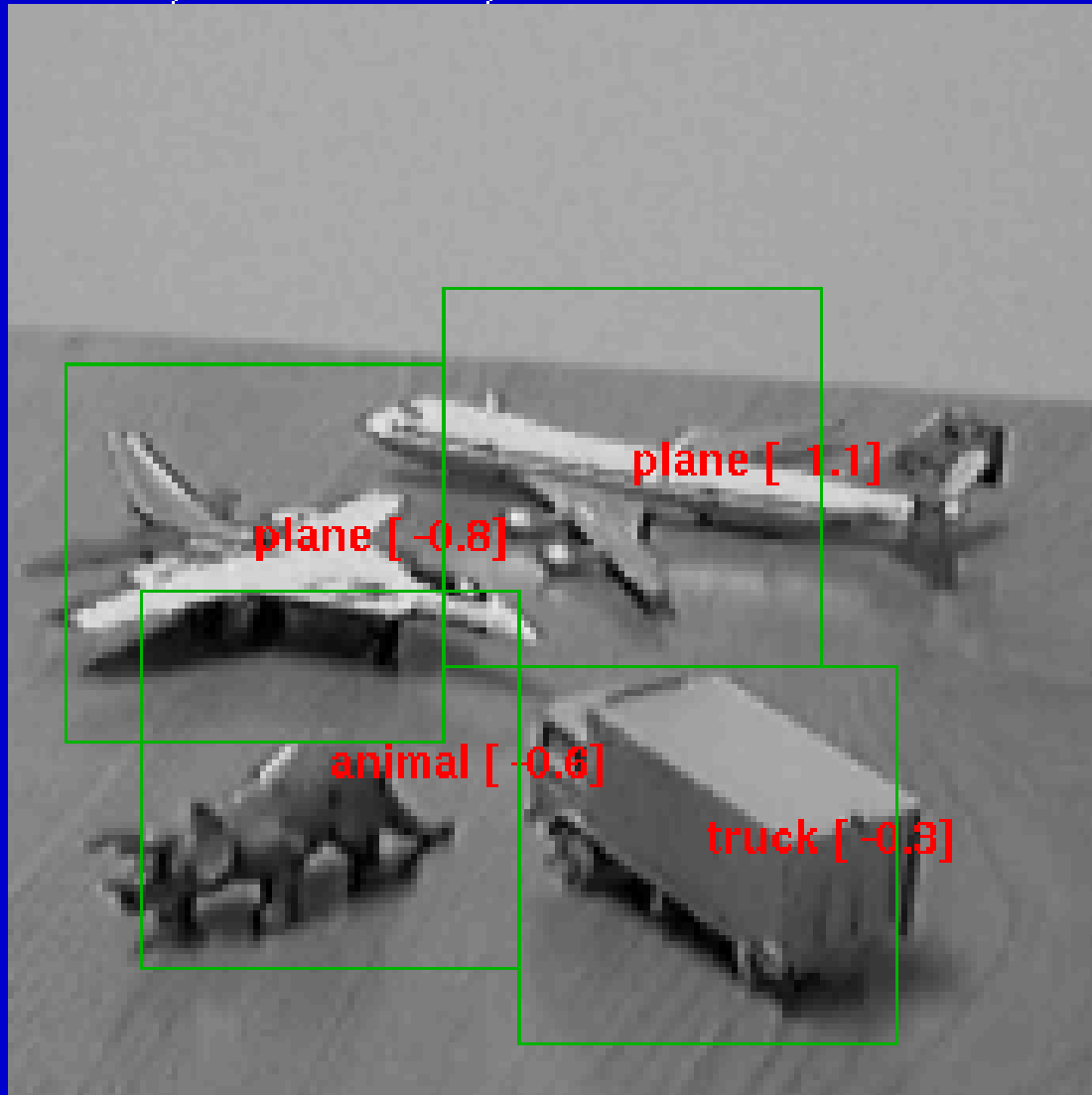
Zoom= 1.0, Threshold= -1.2, filter on



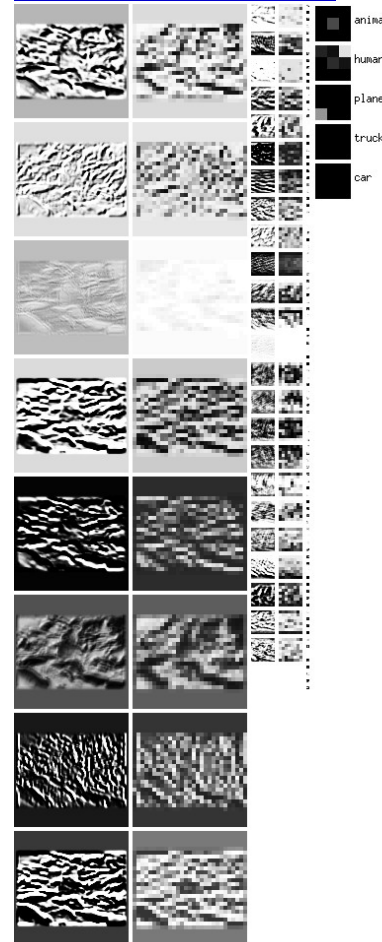
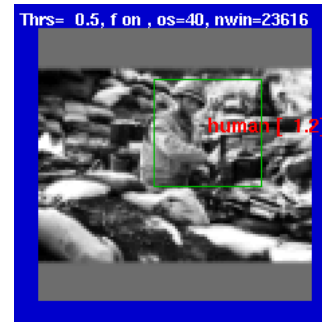
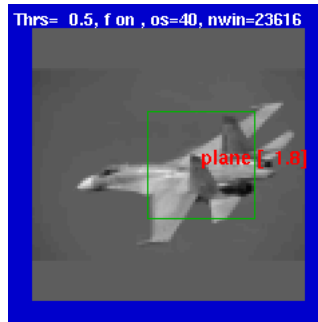
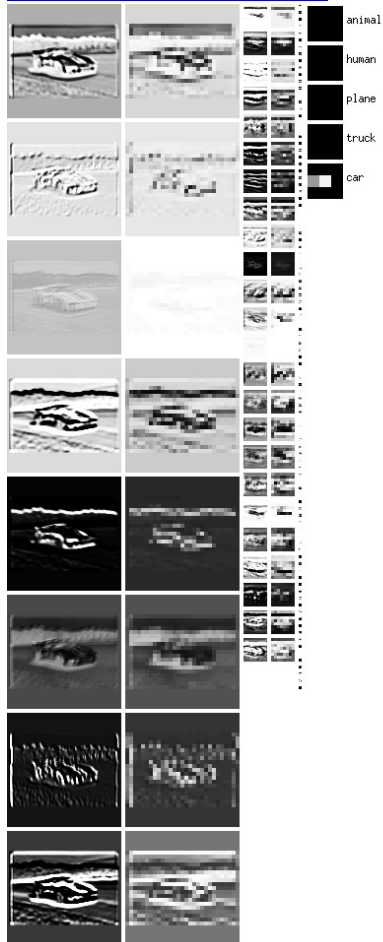
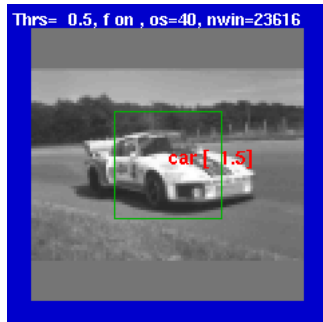


# Examples (Monocular Mode)

Zoom= 0.7, Threshold= -1.8, filter on



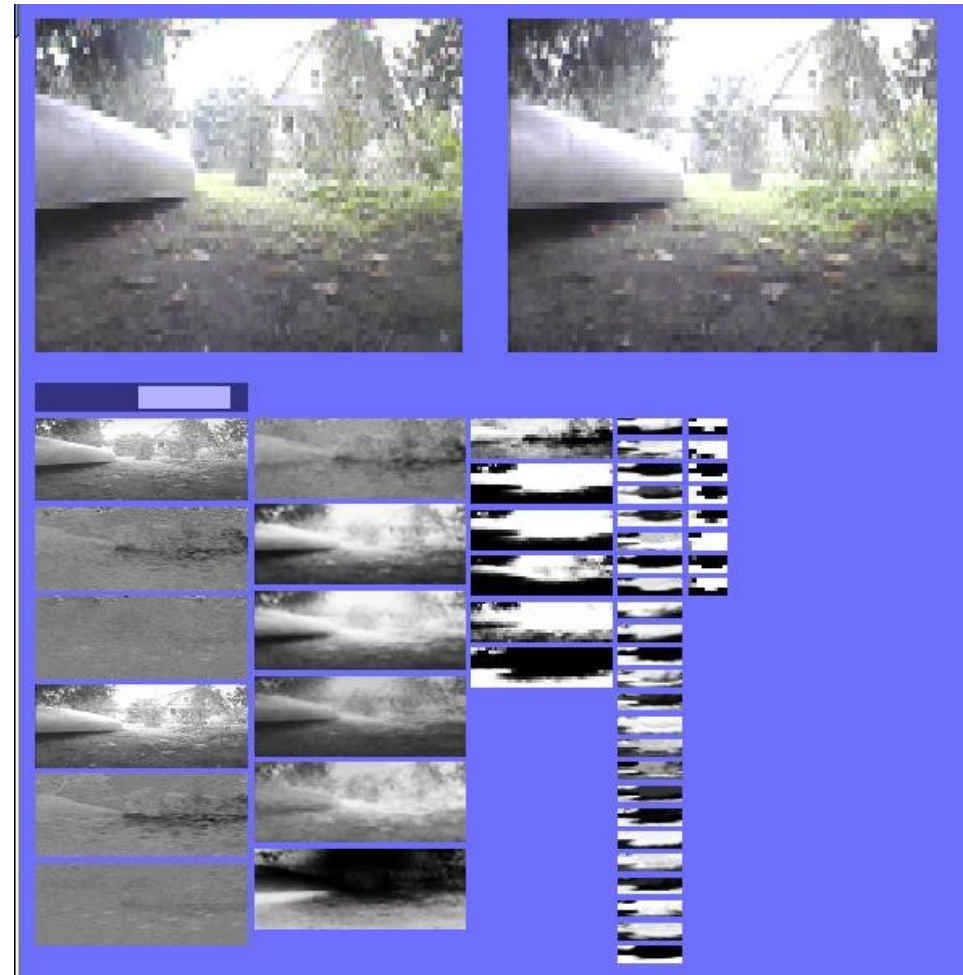
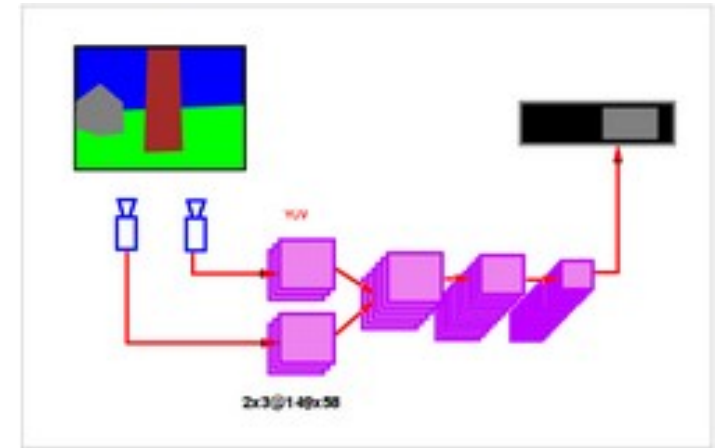
# Natural Images (Monocular Mode)



# 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



# Supervised Convolutional Nets: Pros and Cons

- Convolutional nets can be trained to perform a wide variety of visual tasks.
  - ▶ Global supervised gradient descent can produce parsimonious architectures
- **BUT: they require lots of labeled training samples**
  - ▶ 60,000 samples for handwriting
  - ▶ 120,000 samples for face detection
  - ▶ 25,000 to 350,000 for object recognition
- **Since low-level features tend to be non task specific, we should be able to learn them unsupervised.**
- Hinton has shown that layer-by-layer unsupervised “pre-training” can be used to initialize “deep” architectures
  - ▶ [Hinton & Shalakhutdinov, Science 2006]
- **Can we use this idea to reduce the number of necessary labeled examples.**



# Models Similar to ConvNets

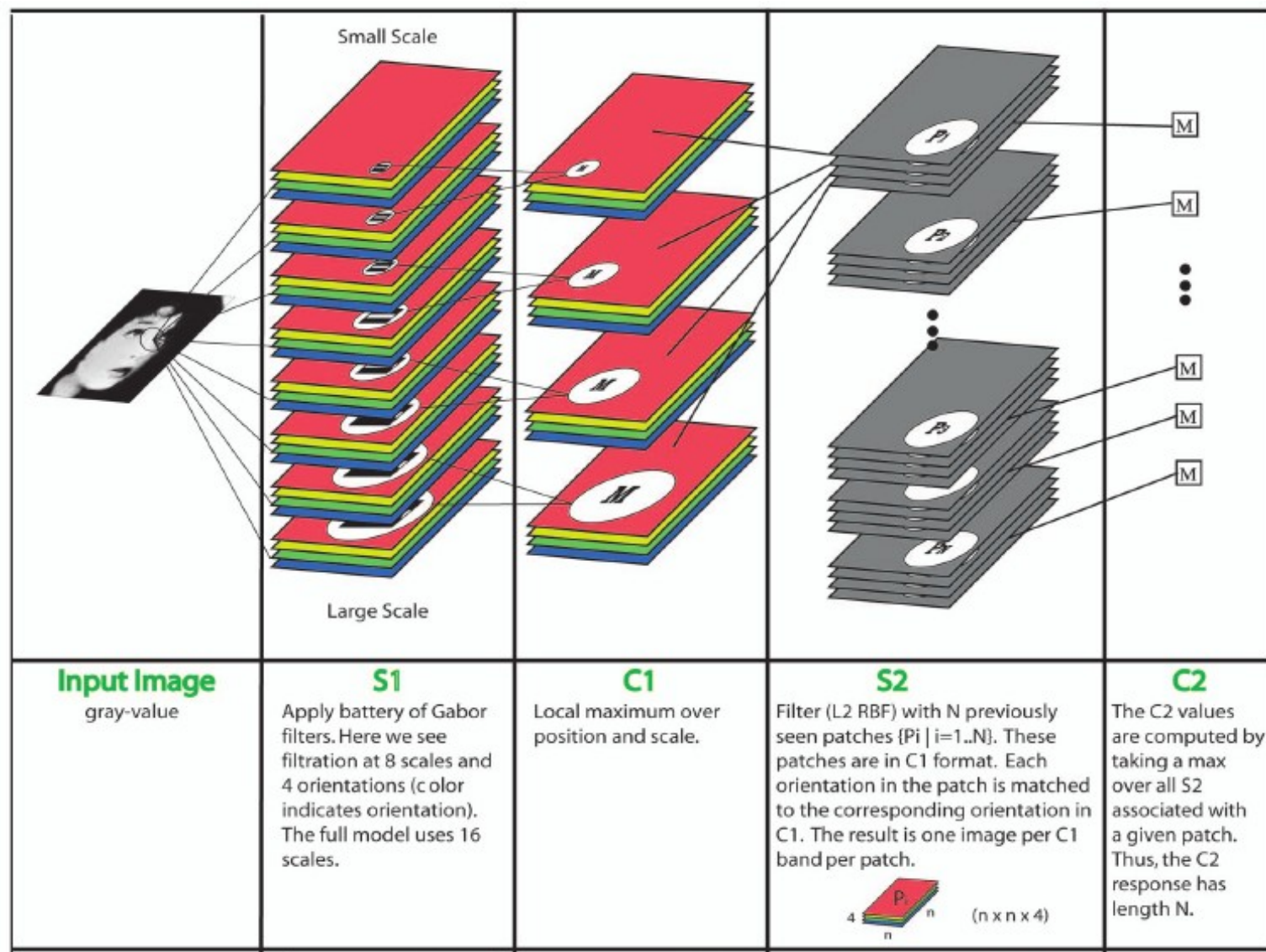
## HMAX

- ▶ [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]