Deep Learning:
A Cornucopia of Applications and Mathematical Mysteries

Yann Le Cun
Facebook AI Research,
Center for Data Science, NYU
Courant Institute of Mathematical Sciences, NYU
http://yann.lecun.com

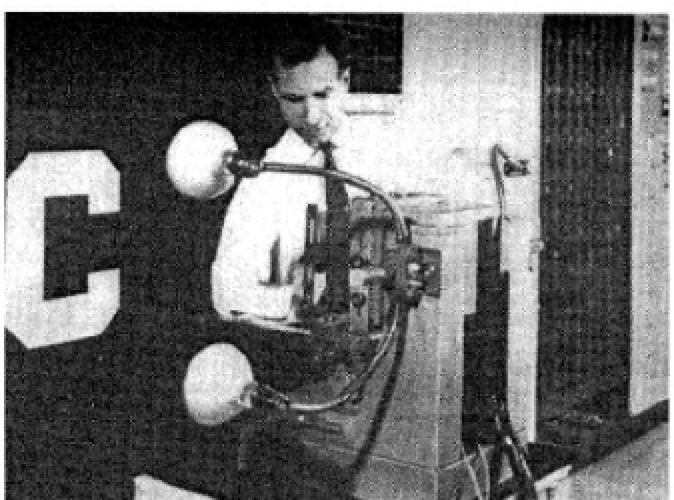
The Traditional Model of Pattern Recognition

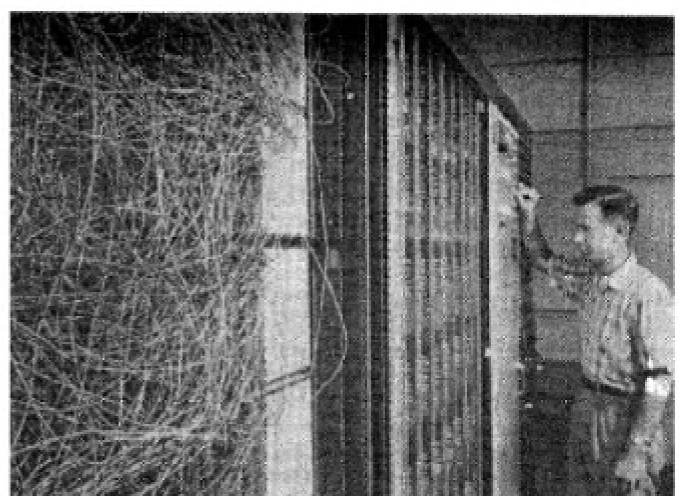
Y LeCun

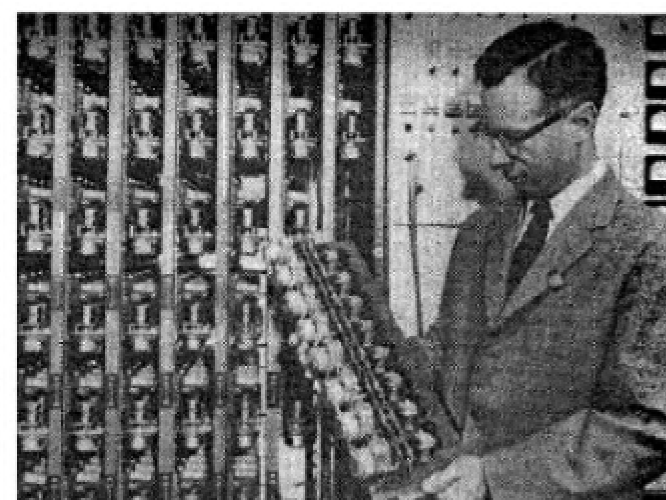
- The traditional model of pattern recognition (since the late 50's)
  - Fixed/engineered features (or fixed kernel) + trainable classifier

hand-crafted Feature Extractor → "Simple" Trainable Classifier →

- Perceptron (Cornell University, 1957)

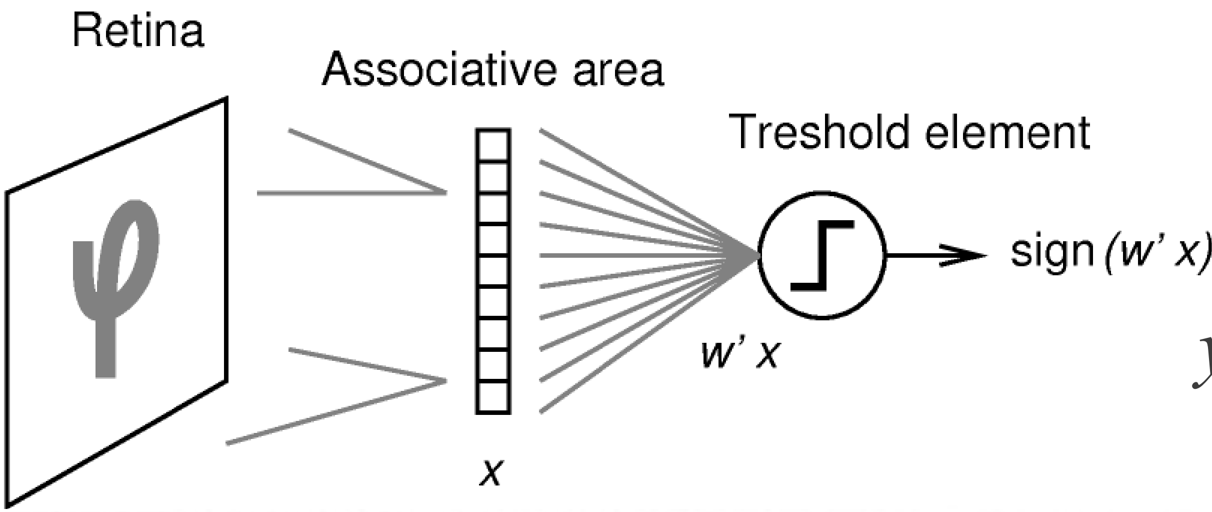




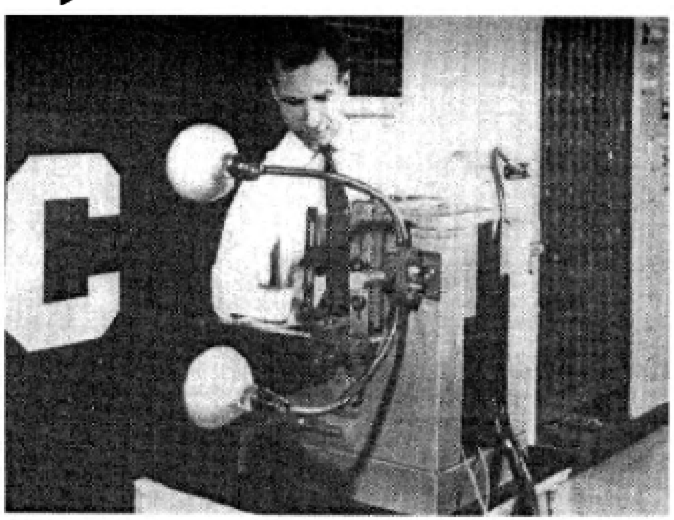


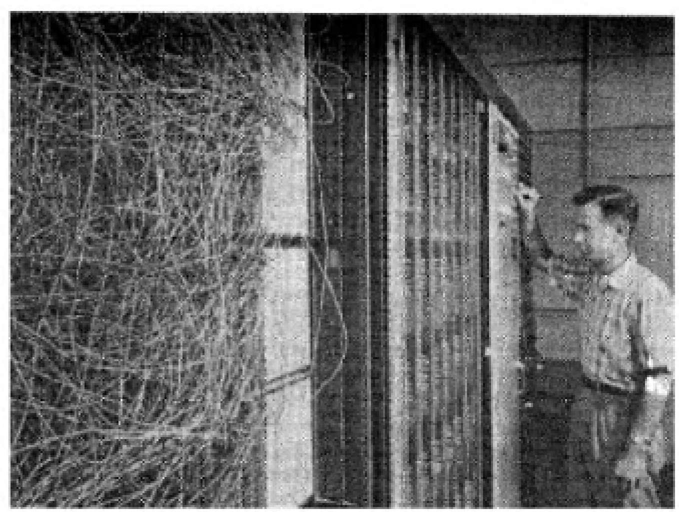
1957: The Perceptron (the first learning machine)

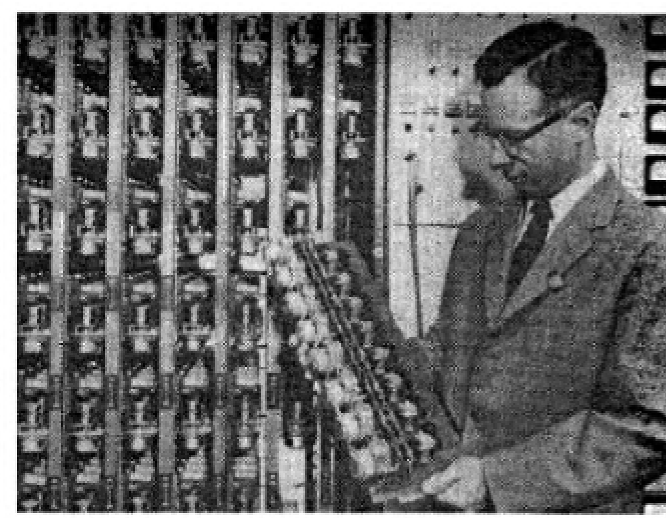
- A simple simulated neuron with adaptive "synaptic weights"
  - Computes a weighted sum of inputs
  - Output is +1 if the weighted sum is above a thresold, -1 otherwise.



$$y = sign\left(\sum_{i=1}^{N} W_i X_i + b\right)$$

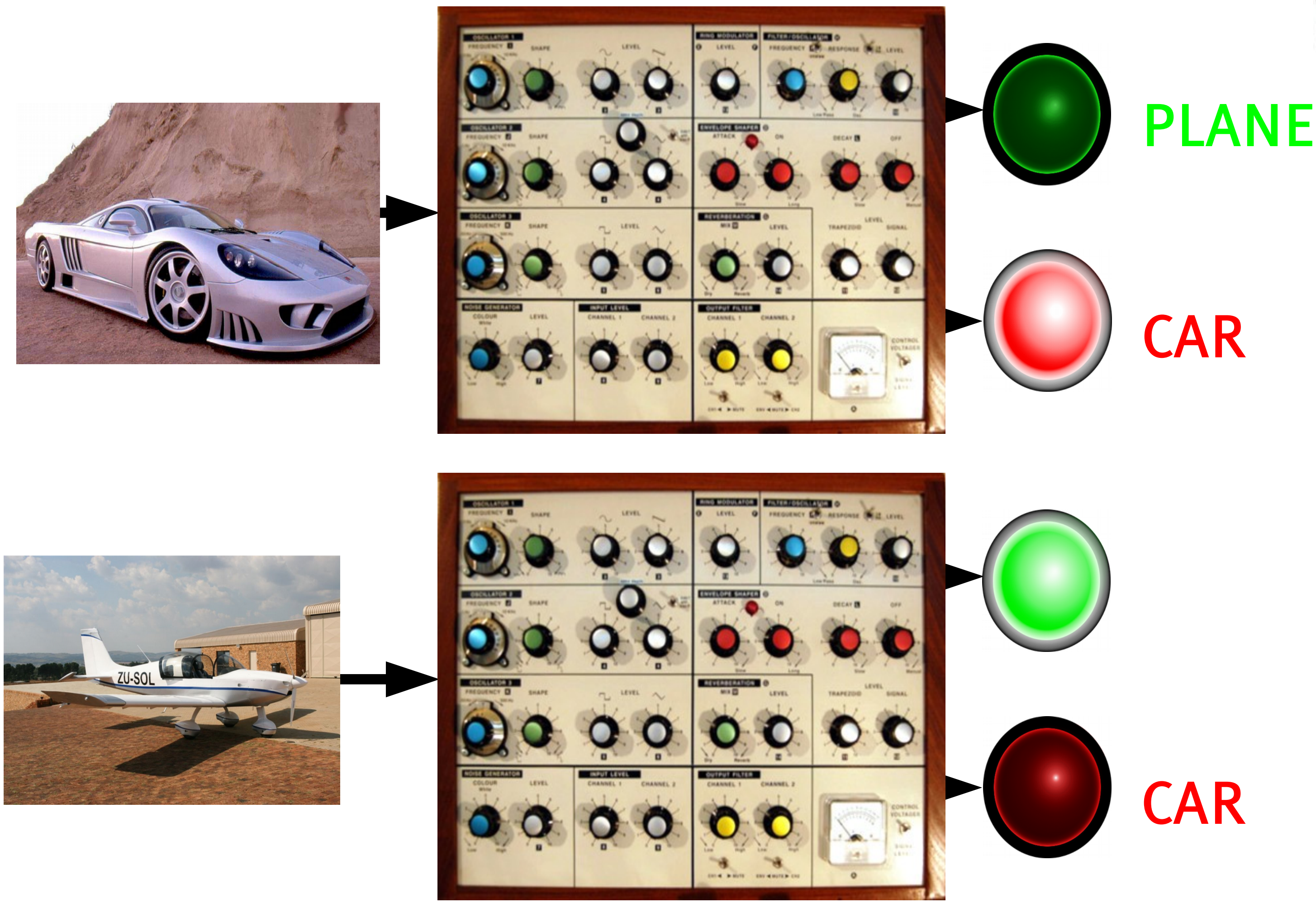






Supervised Learning

- We can train a machine on lots of examples of tables, chairs, dog, cars, and people
- But will it recognize table, chairs, dogs, cars, and people it has never seen before?



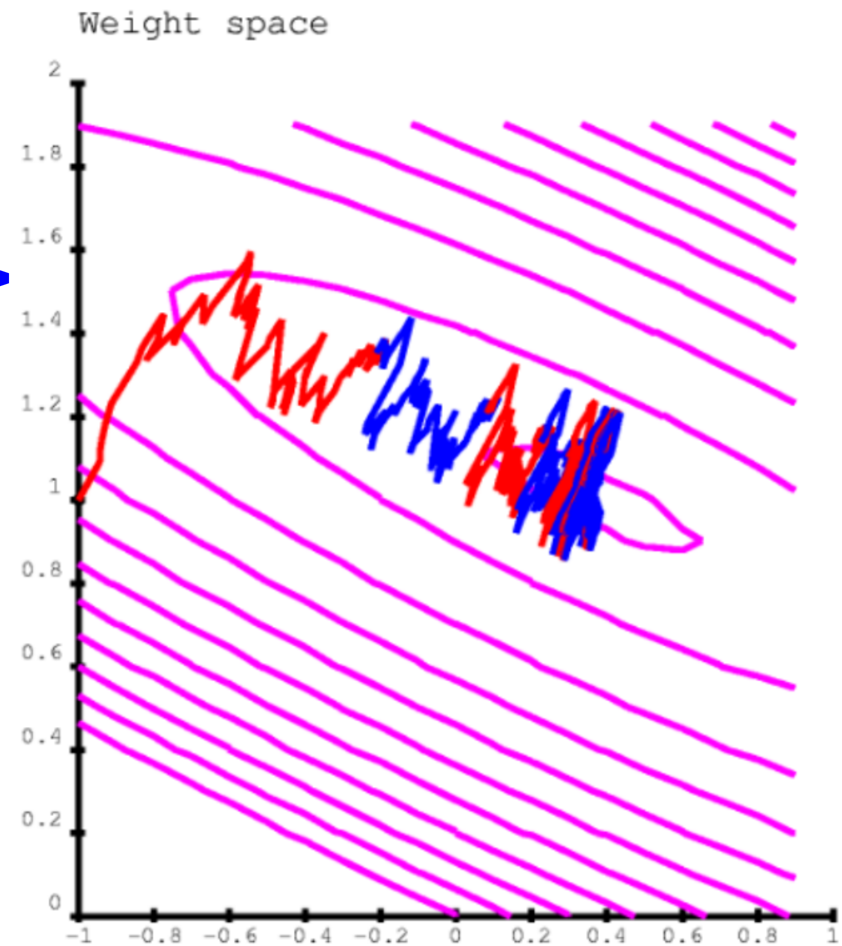
Supervised Machine Learning = Function Optimization

Y LeCun

Function with adjustable parameters

Error Function

traffic light: -1



- It's like walking in the mountains in a fog and following the direction of steepest descent to reach the village in the valley
- But each sample gives us a noisy estimate of the direction. So our path is a bit random.

$$W_i \leftarrow W_i - \eta \frac{\partial L(W,X)}{\partial W_i}$$

- Stochastic Gradient Descent (SGD)

Large-Scale Machine Learning: the reality

- Hundreds of millions of "knobs" (parameters)
- Thousands of categories
- Millions of training samples
- Recognizing each sample may take billions of operations
  - But these operations are simple multiplications and additions

Deep Learning = The Entire Machine is Trainable

- **Traditional Pattern Recognition:** Fixed/Handcrafted Feature Extractor

Feature Extractor → Trainable Classifier →

- **Mainstream Modern Pattern Recognition:** Unsupervised mid-level features

Feature Extractor → Mid-Level Features → Trainable Classifier →

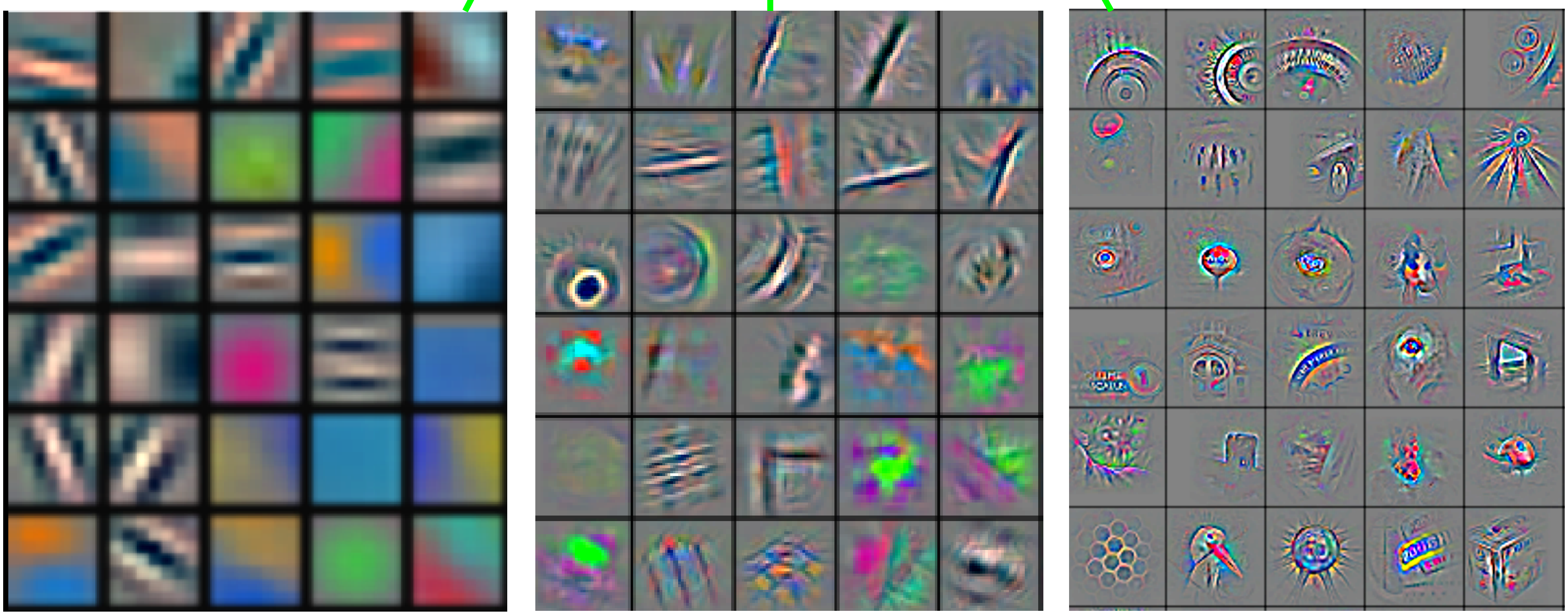
- **Deep Learning:** Representations are hierarchical and trained

Low-Level Features → Mid-Level Features → High-Level Features → Trainable Classifier →

Deep Learning = Learning Hierarchical Representations

- It's deep if it has more than one stage of non-linear feature transformation

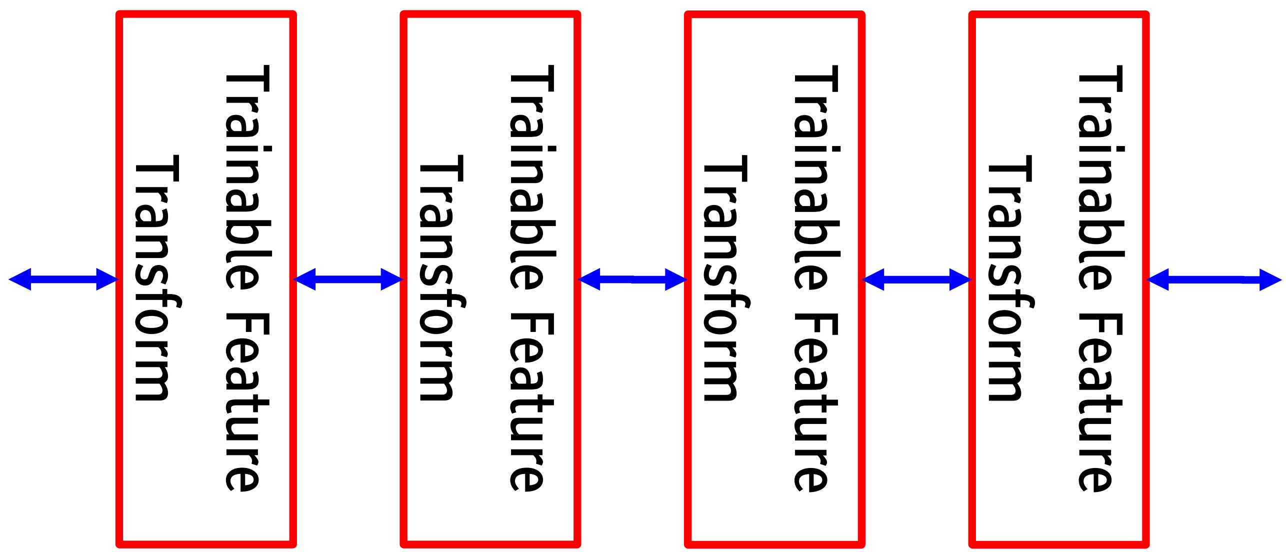
Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier →



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Trainable Feature Hierarchy

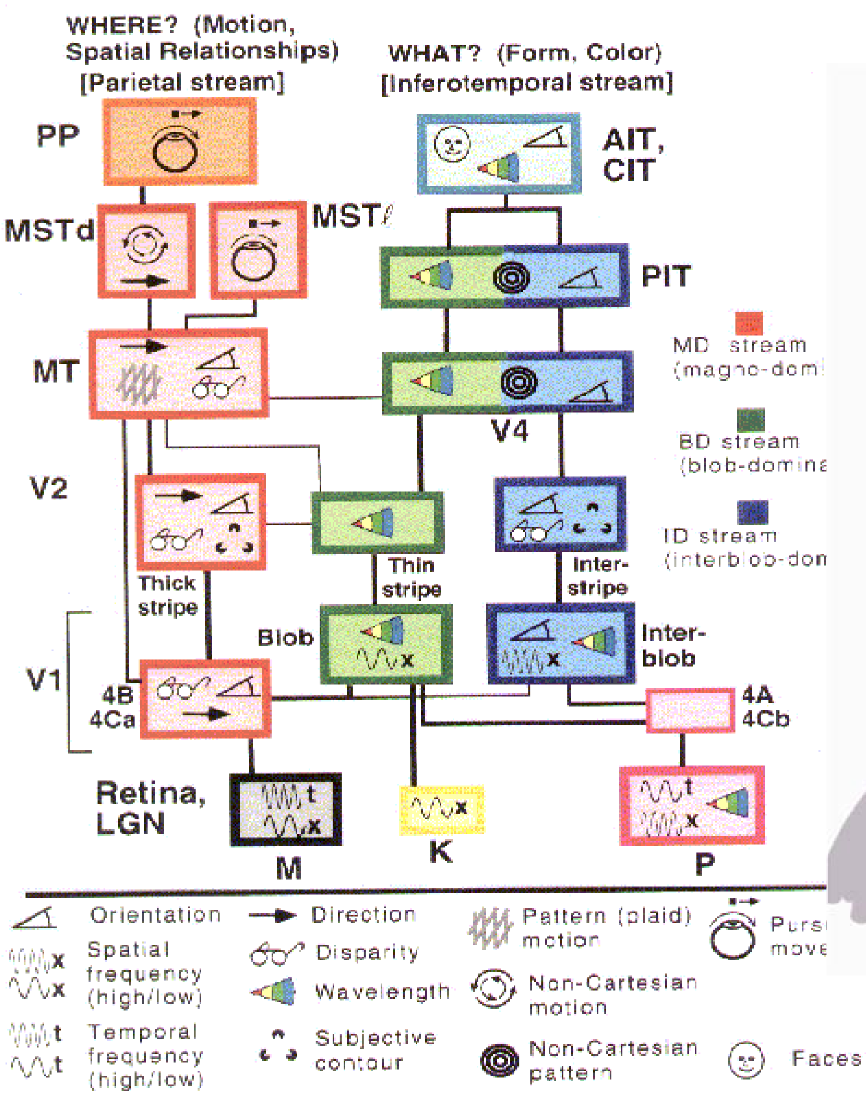
- **Hierarchy of representations with increasing level of abstraction**
- **Each stage is a kind of trainable feature transform**
- **Image recognition**
  - Pixel → edge → texton → motif → part → object
- **Text**
  - Character → word → word group → clause → sentence → story
- **Speech**
  - Sample → spectral band → sound → … → phone → phoneme → word

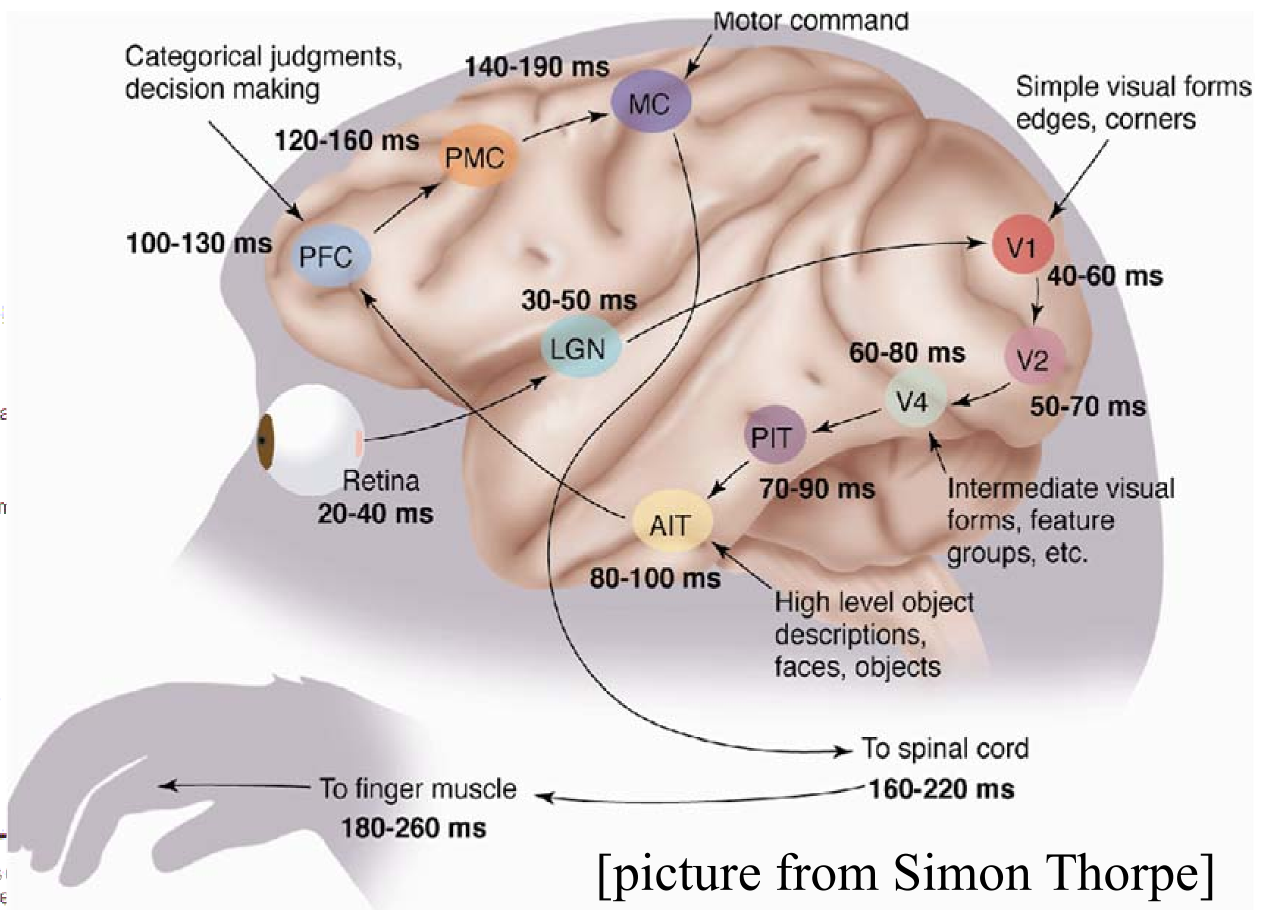


How does the brain interprets images?

Y LeCun

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





[picture from Simon Thorpe]

[Gallant & Van Essen]

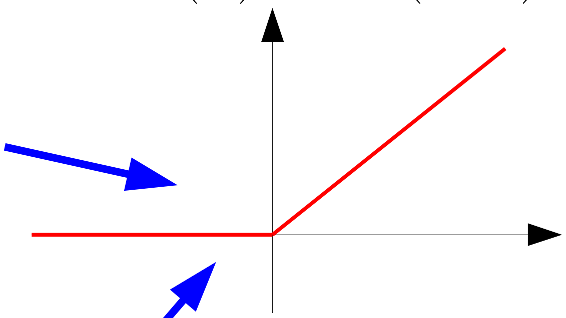
Multi-Layer Neural Networks

Multi-Layer Neural Nets

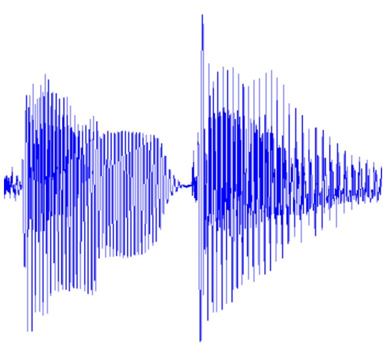
Y LeCun

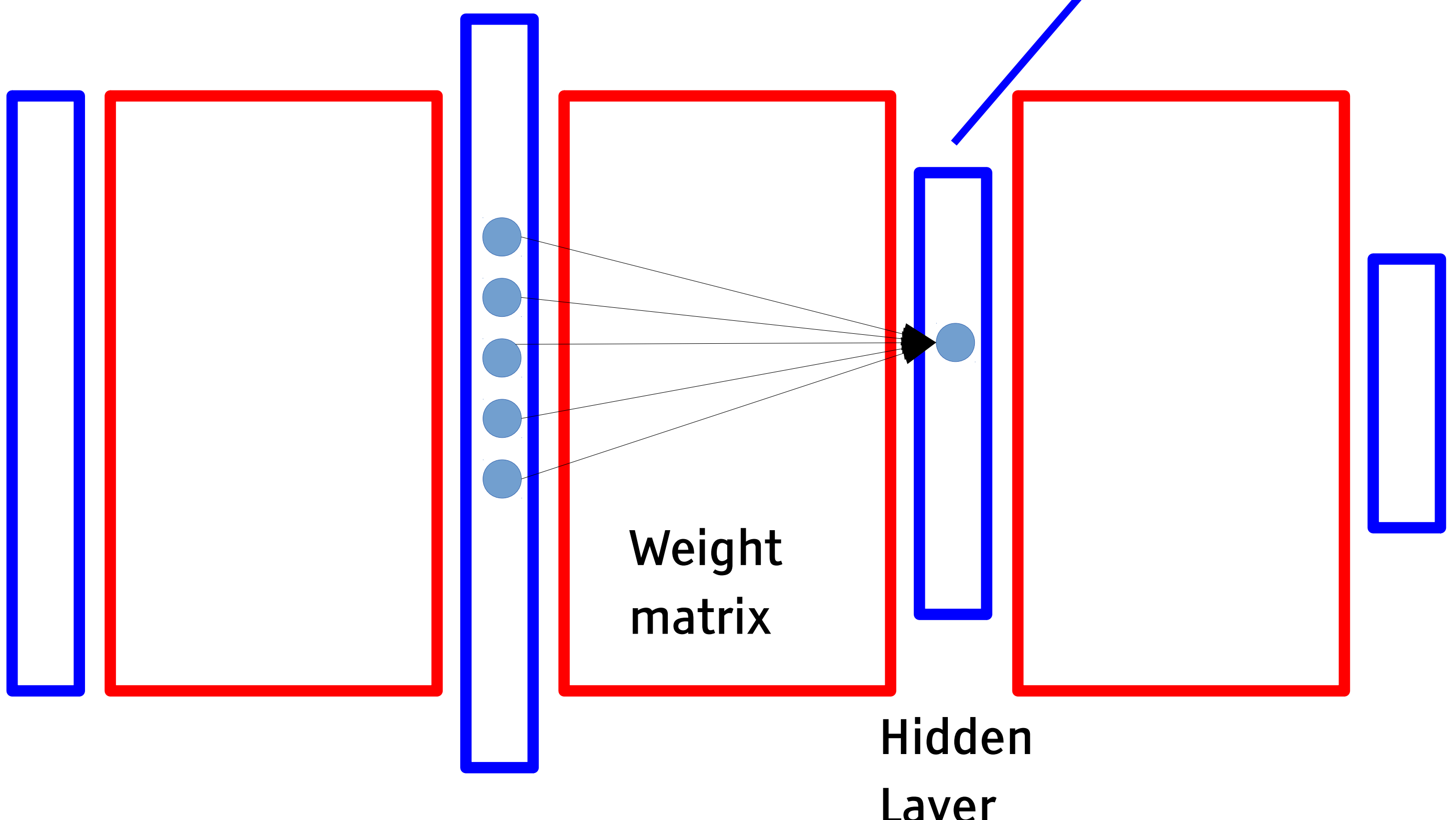
- Multiple Layers of simple units
- Each units computes a weighted sum of its inputs
- Weighted sum is passed through a non-linear function
- The learning algorithm changes the weights

$$ReLU(x) = max(x, 0)$$



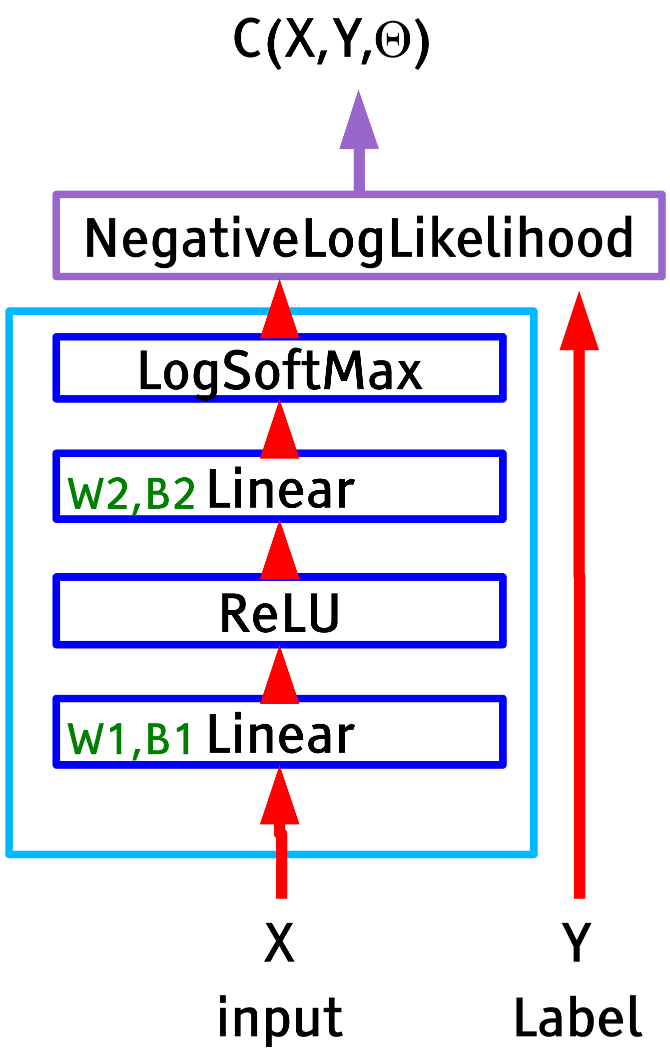
Ceci est une voiture





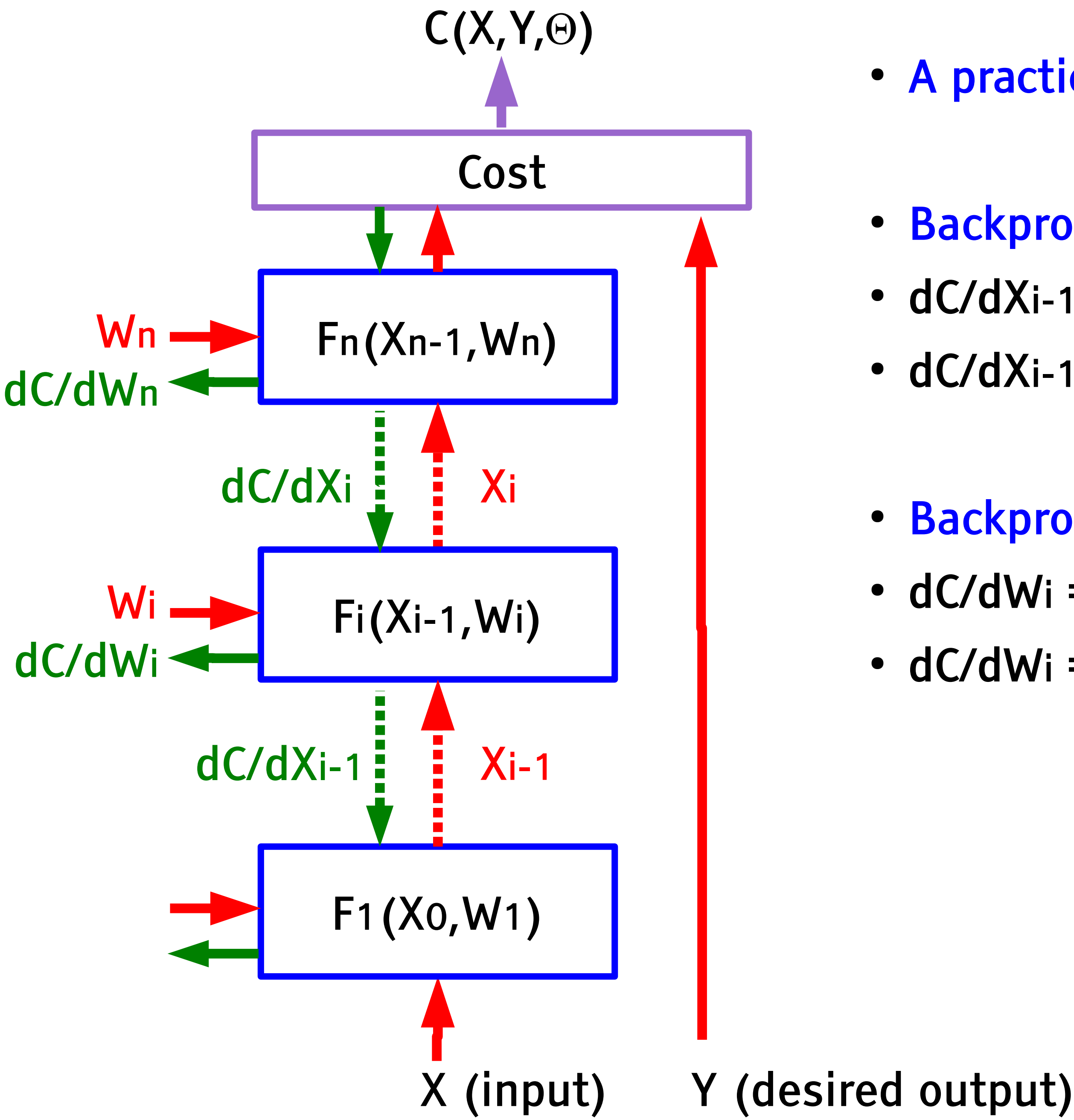
Building a Network by Assembling Modules. With Automatic Differentiation

- All major deep learning frameworks use modules (inspired by SN/Lush, 1991)
  - Torch7, Theano, TensorFlow....




```
-- sizes
ninput = 28*28   -- e.g. for MNIST
nhidden1 = 1000
noutput = 10

-- network module
net = nn.Sequential()
net:add(nn.Linear(ninput, nhidden))
net:add(nn.Threshold())
net:add(nn.Linear(nhidden, noutput))
net:add(nn.LogSoftMax()))

-- cost module
cost = nn.ClassNLLCriterion()

-- get a training sample
input = trainingset.data[k]
target = trainingset.labels[k]

-- run through the model
output = net:forward(input)
c = cost:forward(output, target)
```

Computing Gradients by Back-Propagation

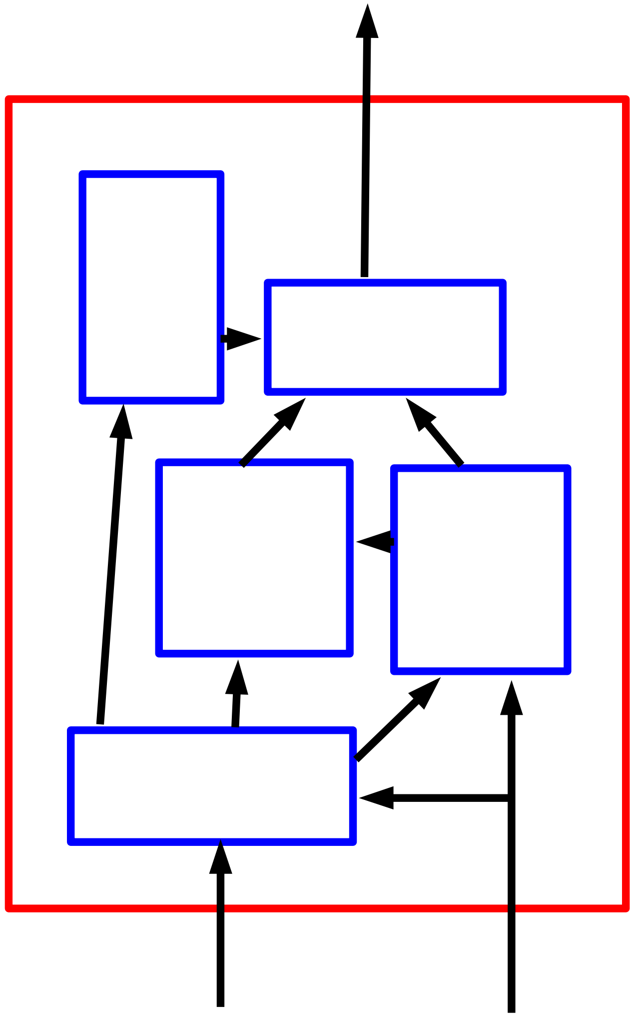


- A practical Application of Chain Rule

- Backprop for the state gradients:
- $dC/dX_{i-1} = dC/dX_i \,.\, dX_i/dX_{i-1}$
- $dC/dX_{i-1} = dC/dX_i \,.\, dF_i(X_{i-1},W_i)/dX_{i-1}$

- Backprop for the weight gradients:
- $dC/dW_i = dC/dX_i \,.\, dX_i/dW_i$
- $dC/dW_i = dC/dX_i \,.\, dF_i(X_{i-1},W_i)/dW_i$

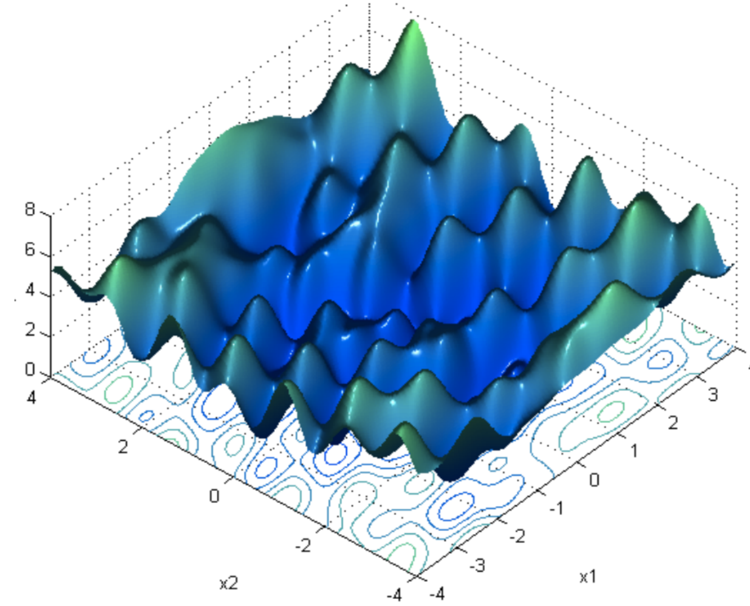
Any Architecture works



- **Any connection graph is permissible**
  - Directed acyclic graphs (DAG)
  - 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.
- **Most frameworks provide automatic differentiation**
  - Theano, Torch7+autograd,...
  - Programs are turned into computation DAGs and automatically differentiated.

The Objective Function of Multi-layer Nets is Non Convex

Y LeCun

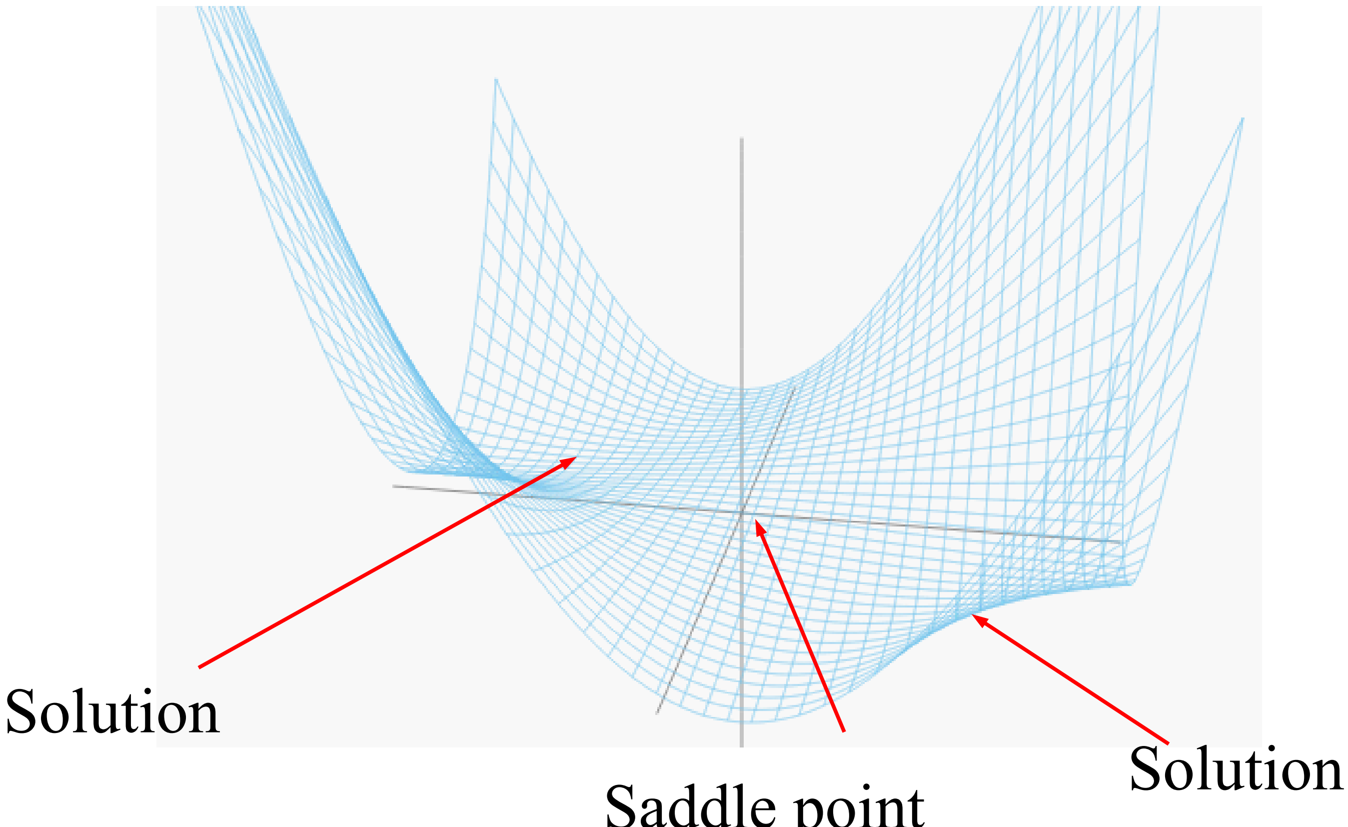


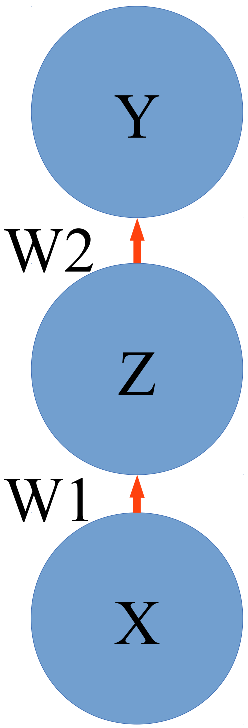
1-1-1 network

- Y = W1*W2*X

Objective: identity function with quadratic loss

One sample: X=1, Y=1 L(W) = (1-W1*W2)^2





Backprop in Practice

- Use ReLU non-linearities
- Use cross-entropy loss for classification
- Use Stochastic Gradient Descent on minibatches
- Shuffle the training samples (← very important)
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 or L2 regularization on the weights (or a combination)
  - But it's best to turn it on after a couple of epochs
- Use “dropout” for regularization
- Lots more in [LeCun et al. “Efficient Backprop” 1998]
- Lots, lots more in “Neural Networks, Tricks of the Trade” (2012 edition) edited by G. Montavon, G. B. Orr, and K-R Müller (Springer)

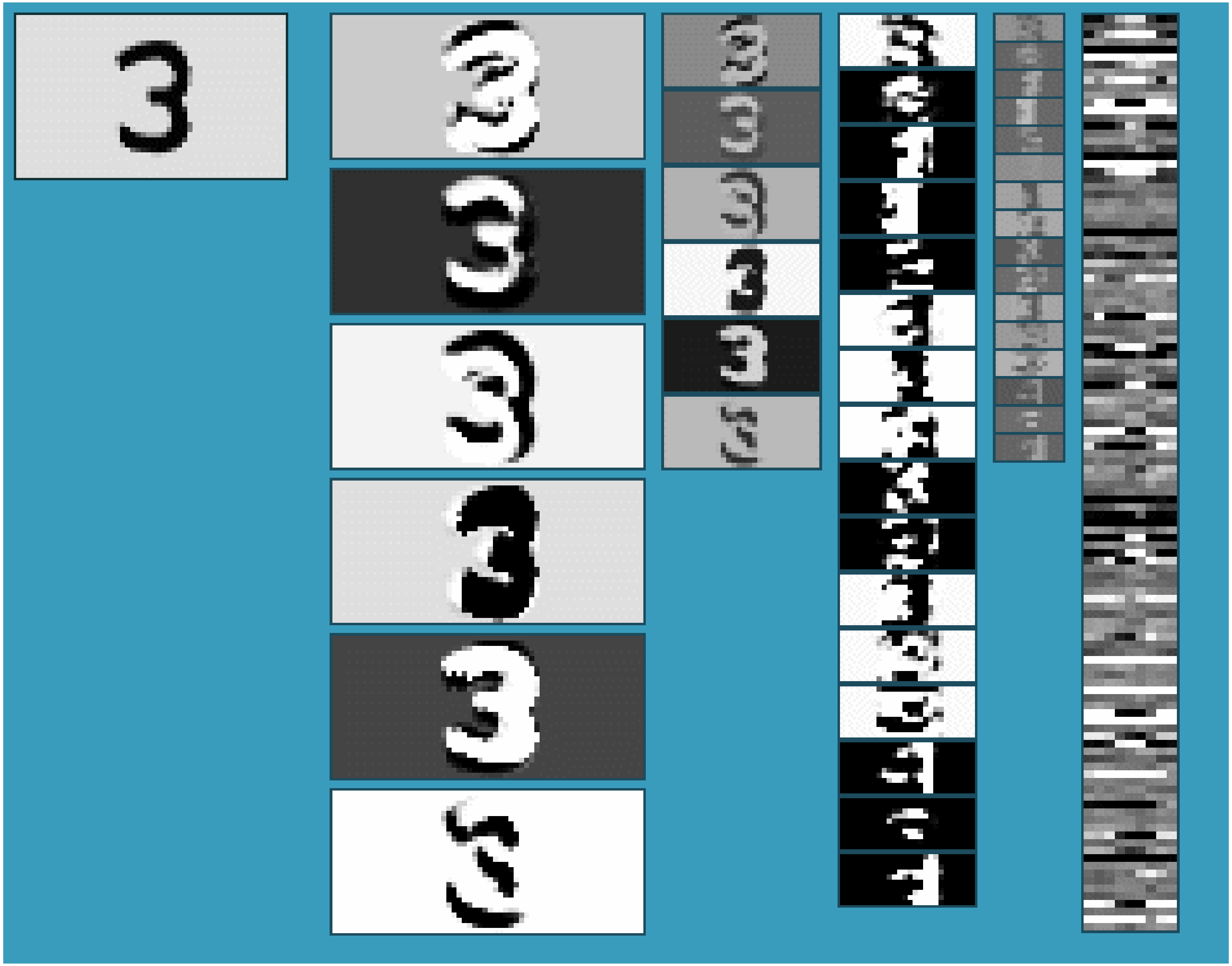
Convolutional Networks

(ConvNet or CNN)

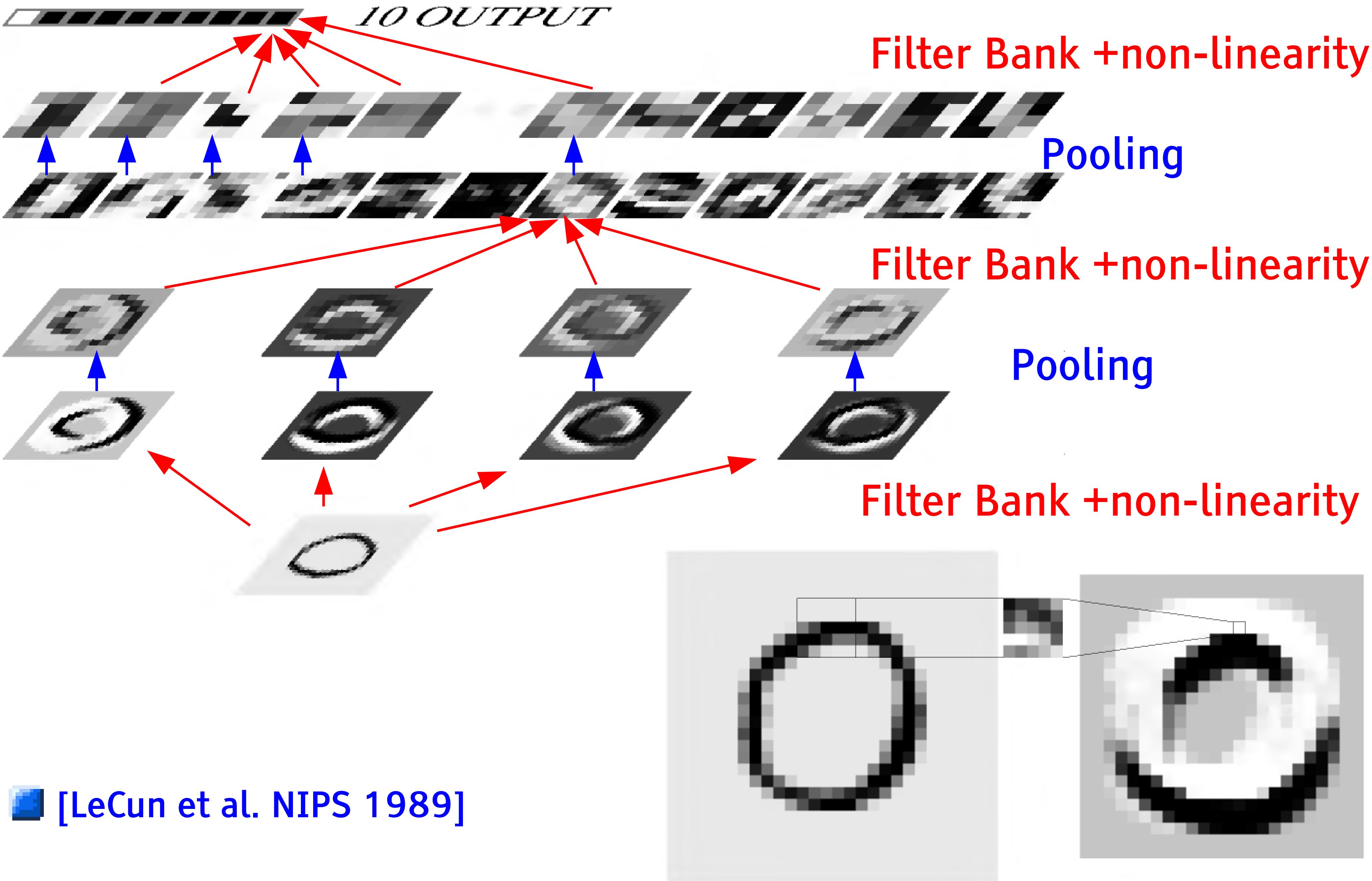
Convolutional Network (vintage 1990)

Y LeCun

- Filters-tanh → pooling → filters-tanh → pooling → filters-tanh



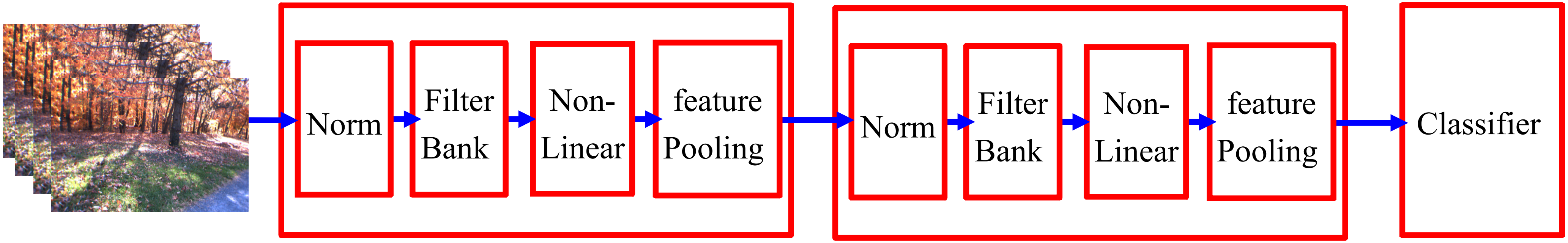
Convolutional Network Architecture



- [LeCun et al. NIPS 1989]

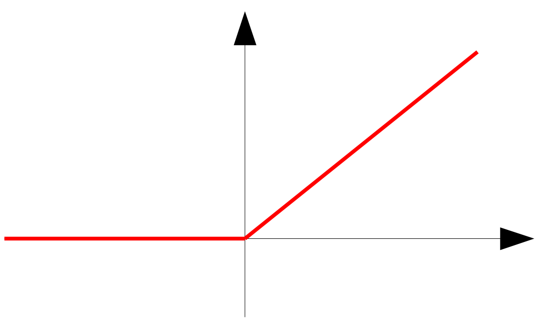
Overall Architecture: multiple stages of Normalization → Filter Bank → Non-Linearity → Pooling

Y LeCun



- **Normalization:** variation on whitening (optional)
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank:** dimension expansion, projection on overcomplete basis
- **Non-Linearity:** sparsification, saturation, lateral inhibition....
  - Rectification (ReLU), Component-wise shrinkage, tanh,..

$$ReLU(x) = max(x, 0)$$



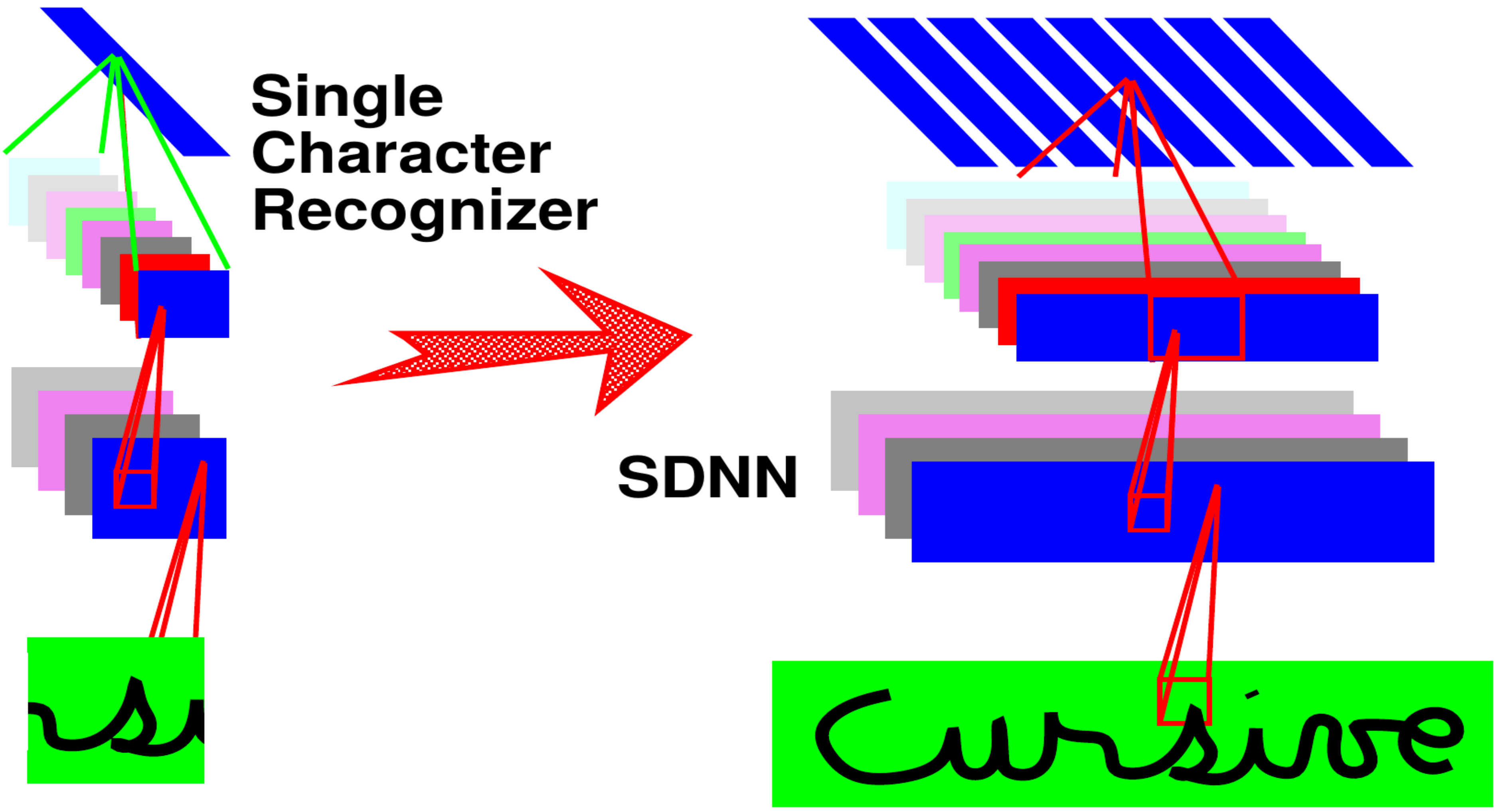
- **Pooling:** aggregation over space or feature type
  - Max, Lp norm, log prob.

$$MAX: Max_i(X_i); \quad L_p: \sqrt[p]{X_i^p}; \quad PROB: \frac{1}{b} \log\left(\sum_i e^{bX_i}\right)$$

Multiple Character Recognition [Matan et al 1992]

Y LeCun

- Every layer is a convolution



Sliding Window ConvNet + Weighted Finite-State Machine

Sliding Window ConvNet + Weighted FSM

Check Reader
(Bell Labs, 1995)

Graph transformer network trained to read check amounts.

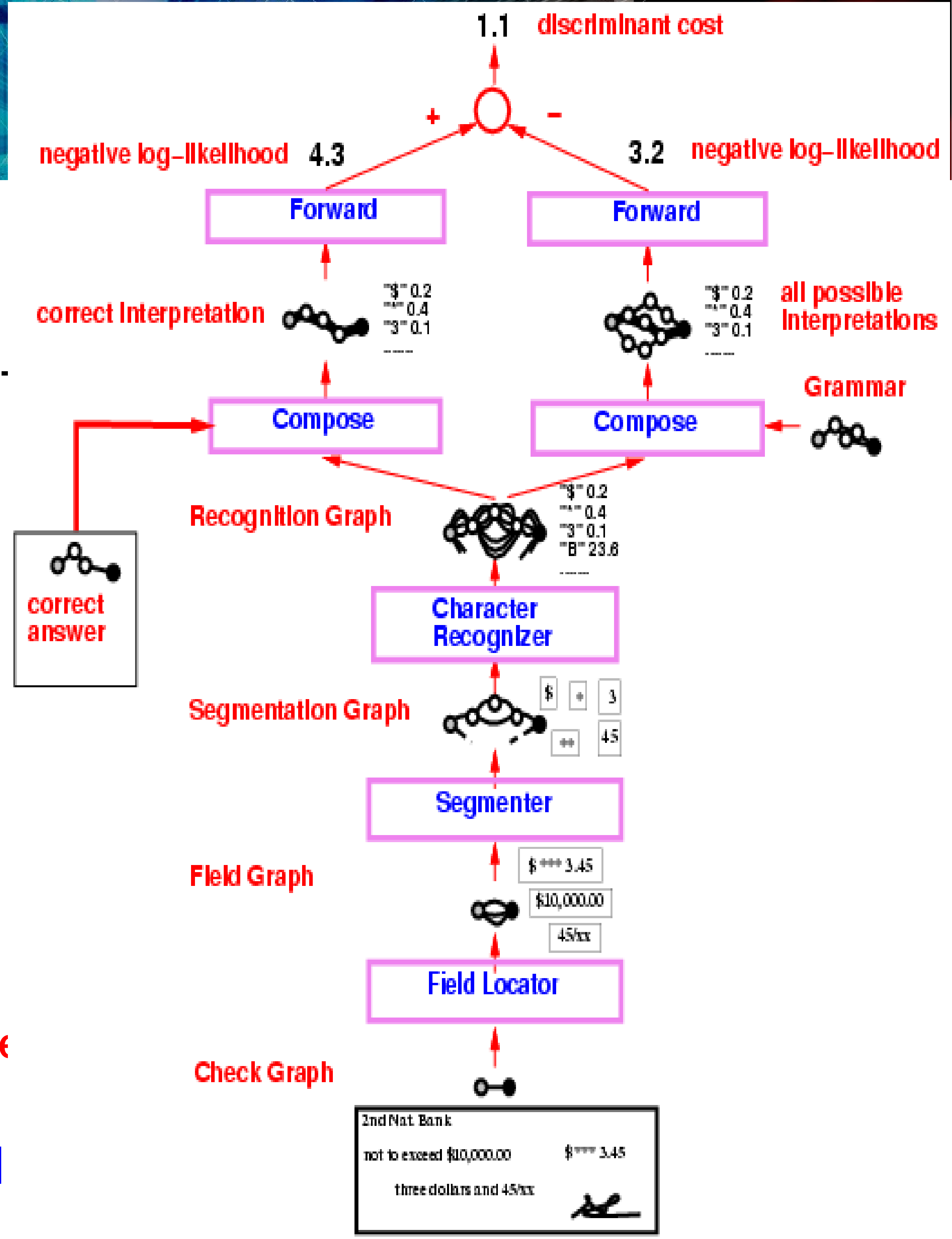
Trained globally with Negative-Log-Likelihood loss.

50% percent correct, 49% reject, 1% error (detectable later in the process).

Fielded in 1996, used in many banks in the US and Europe.

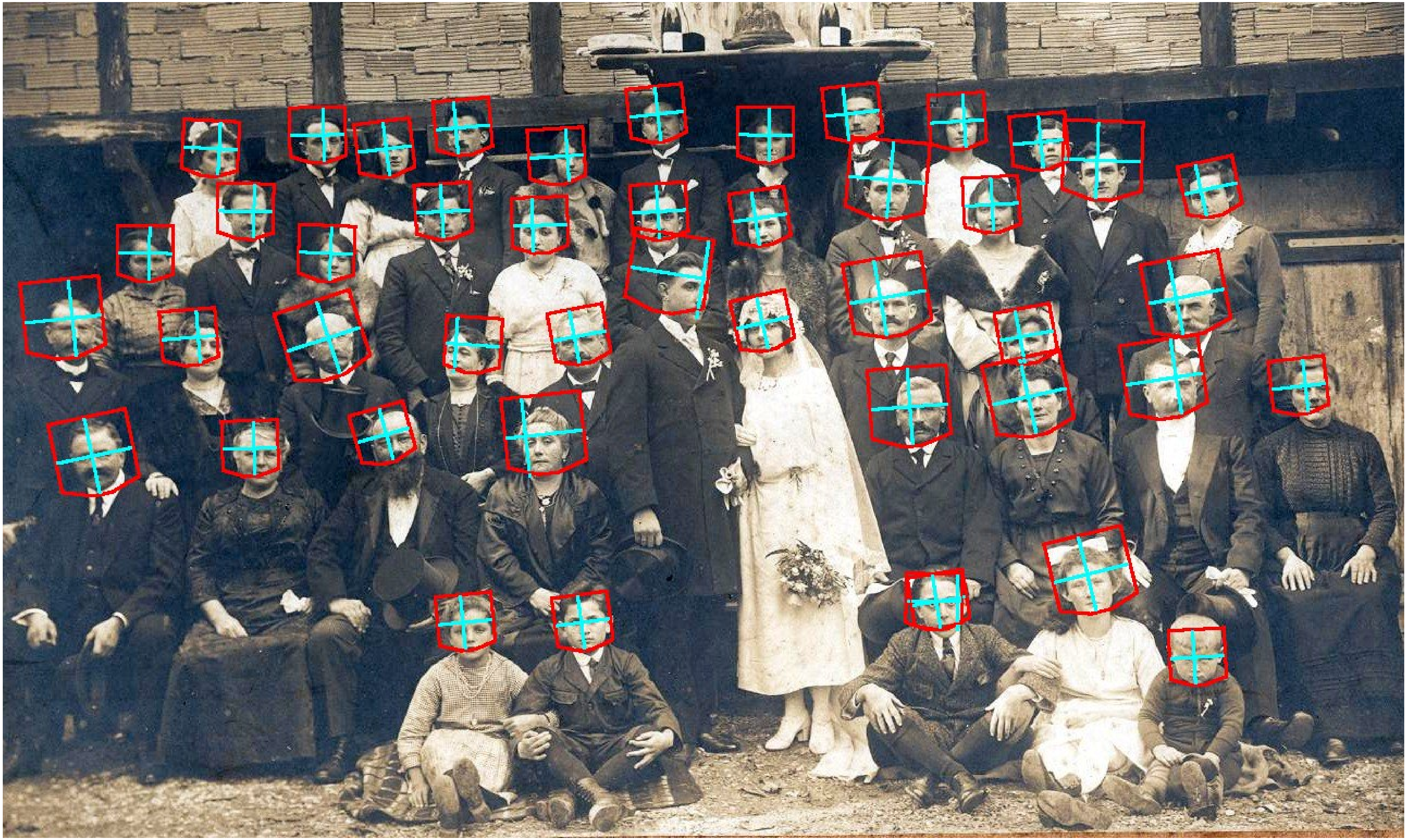
Processed an estimated 10% to 20% of all the checks written in the US in the early 2000s.

[LeCun, Bottou, Bengio, Haffner 1998]

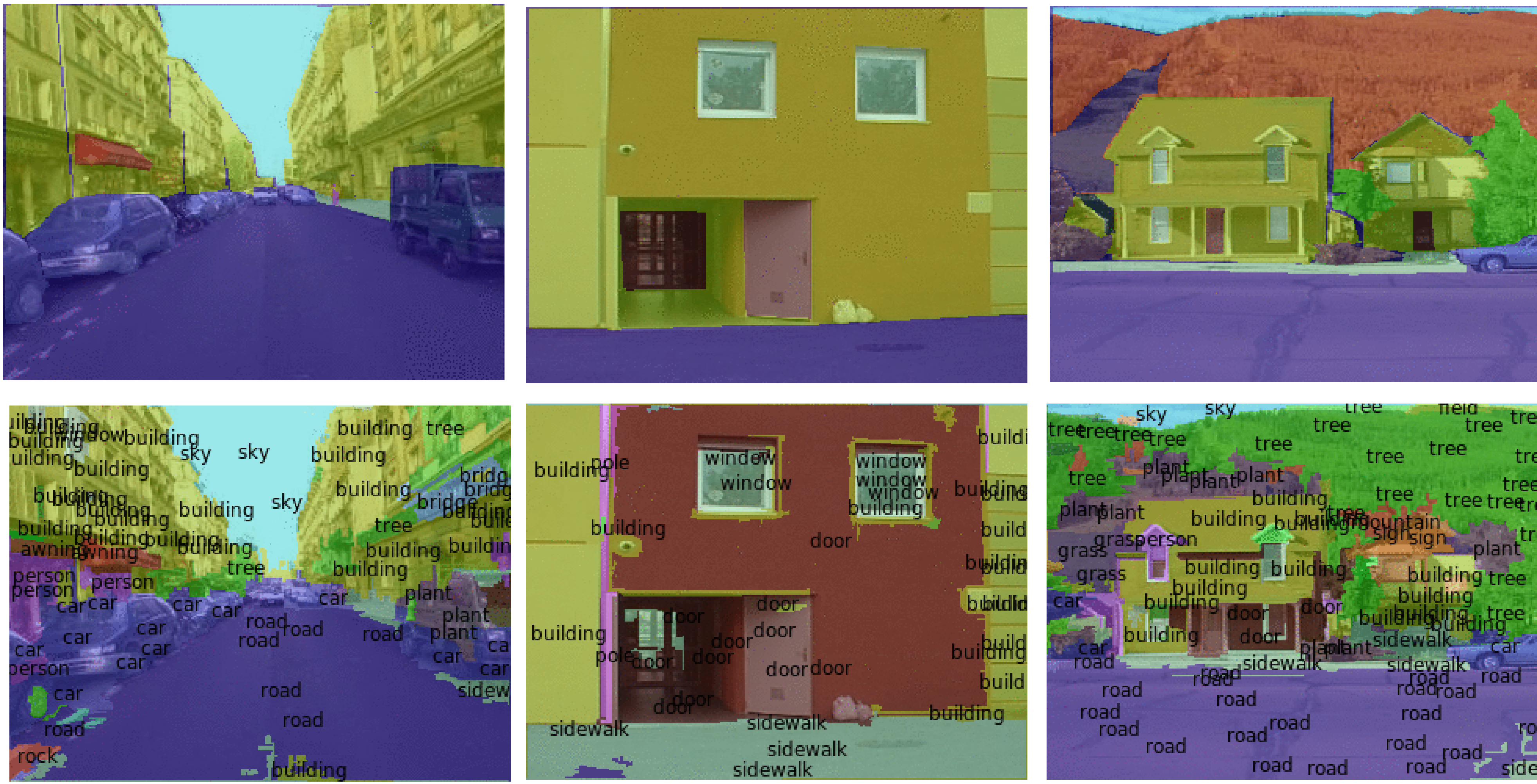


Simultaneous face detection and pose estimation

Y LeCun



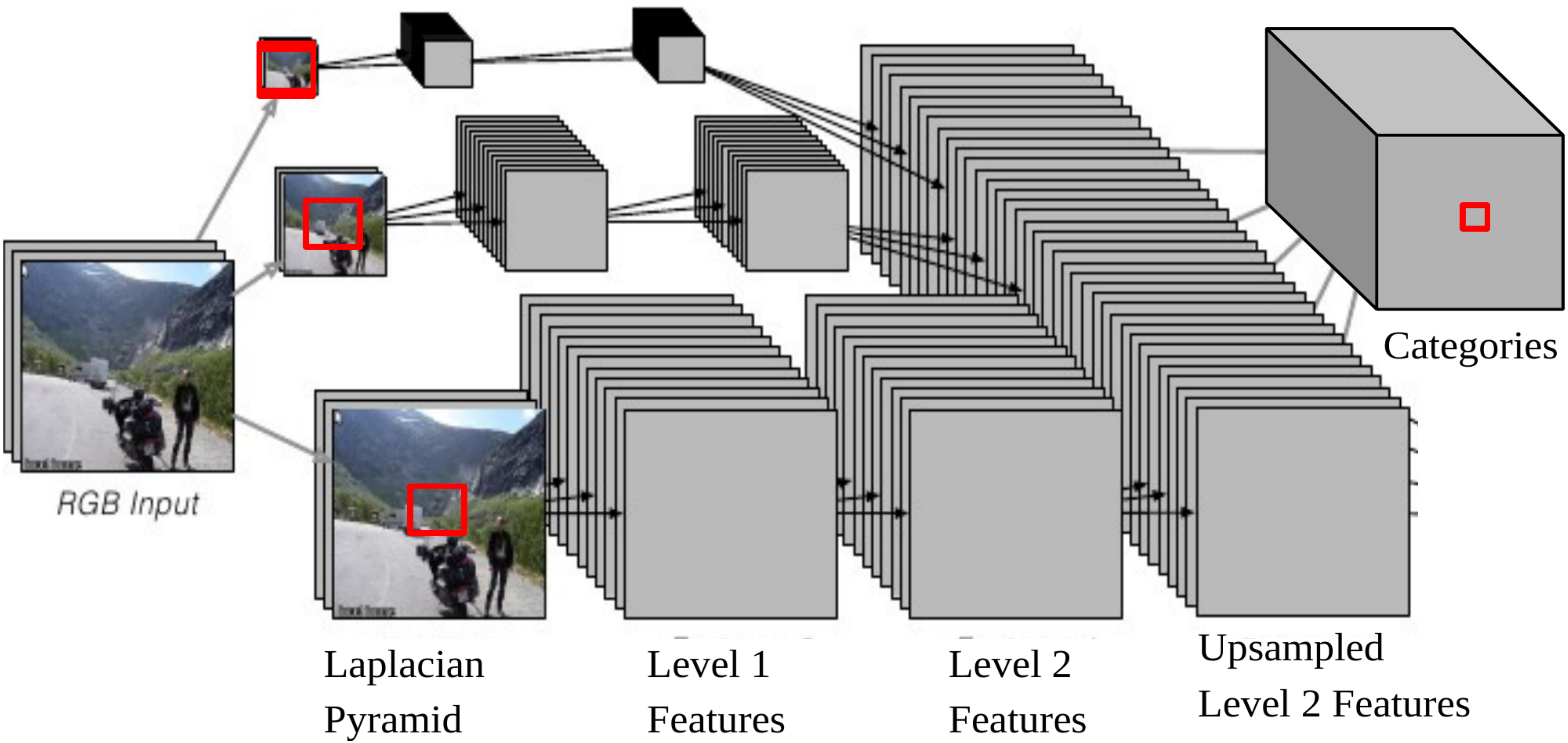
Scene Parsing/Labeling



[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling: Multiscale ConvNet Architecture

- **Each output sees a large input context:**
  - **46x46** window at full rez; **92x92** at ½ rez; **184x184** at ¼ rez
  - [7x7conv]->[2x2pool]->[7x7conv]->[2x2pool]->[7x7conv]->
  - Trained supervised on fully-labeled images



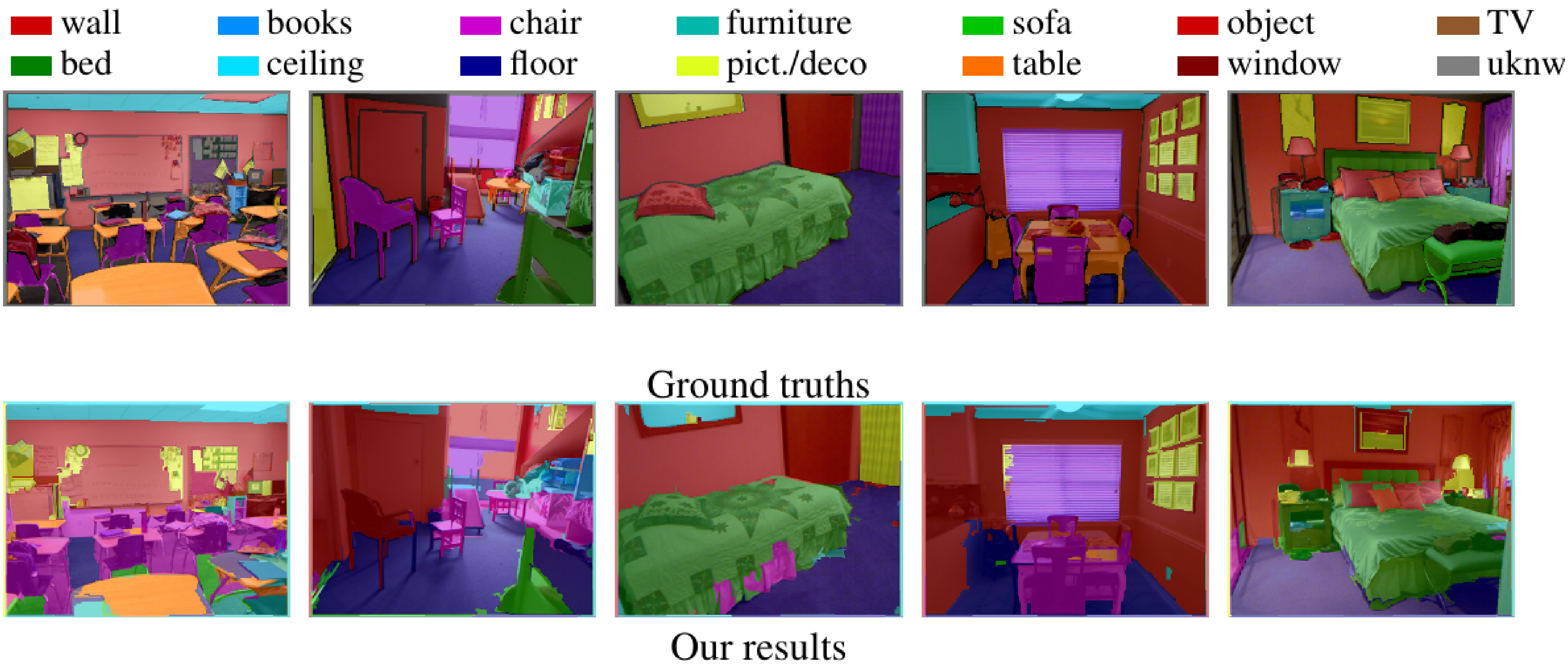
Scene Parsing/Labeling

VIDEO: SCENE PARSING

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

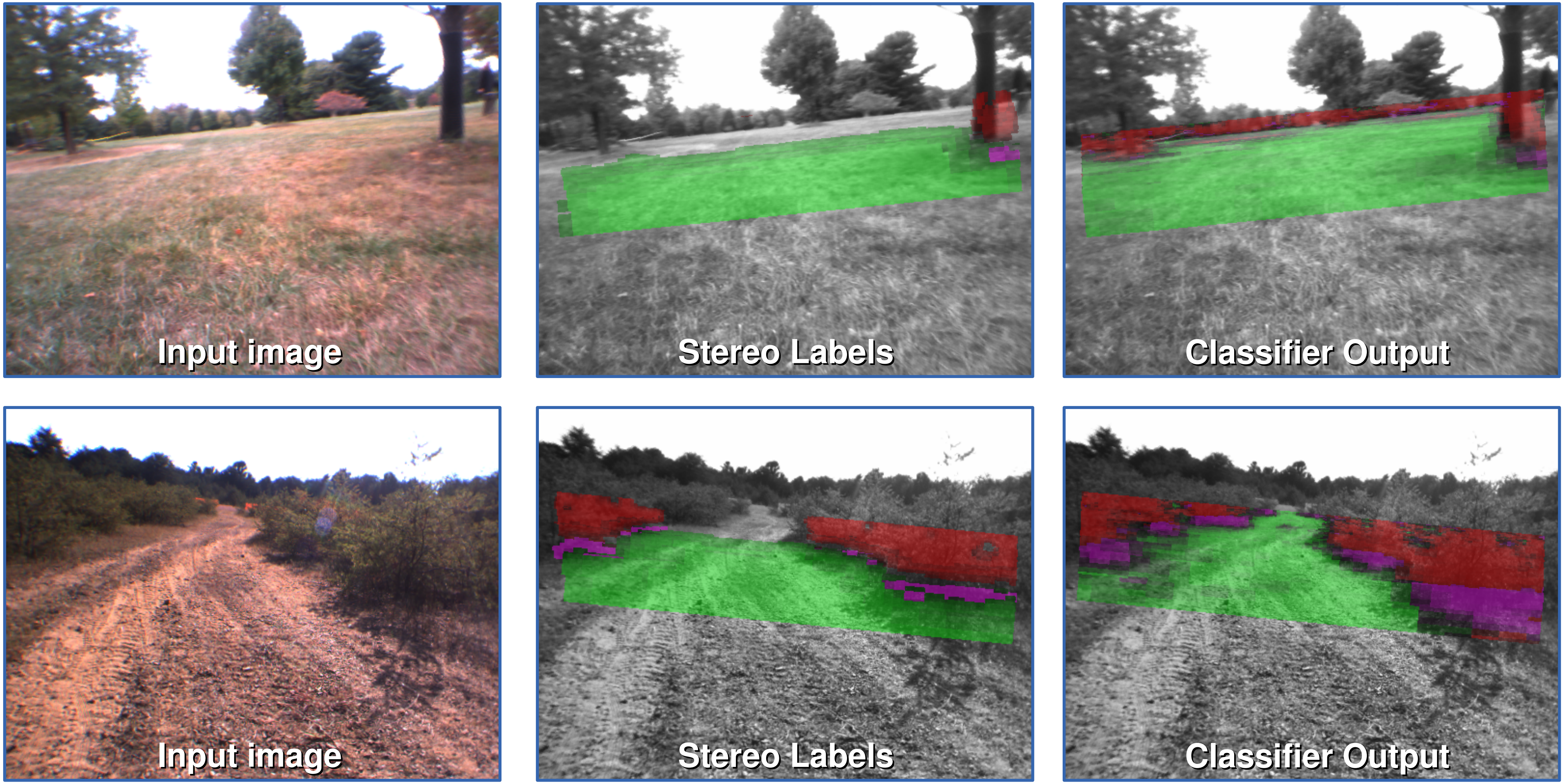
[Farabet et al. ICML 2012, PAMI 2013]

Scene Parsing/Labeling on RGB+Depth Images



[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]

ConvNet for Long Range Adaptive Robot Vision (DARPA LAGR program 2005-2008)



Then in 2012 two things happened...

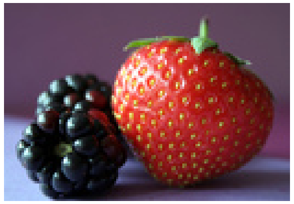
- **The ImageNet dataset [Fei-Fei et al. 2012]**
  - 1.2 million training samples
  - 1000 categories
- **Fast & Programmable General-Purpose GPUs**
  - NVIDIA CUDA
  - Capable of over 1 trillion operations/second

Matchstick

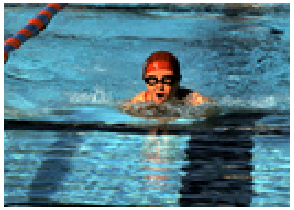
Sea lion

Flute

Strawberry



Bathing cap

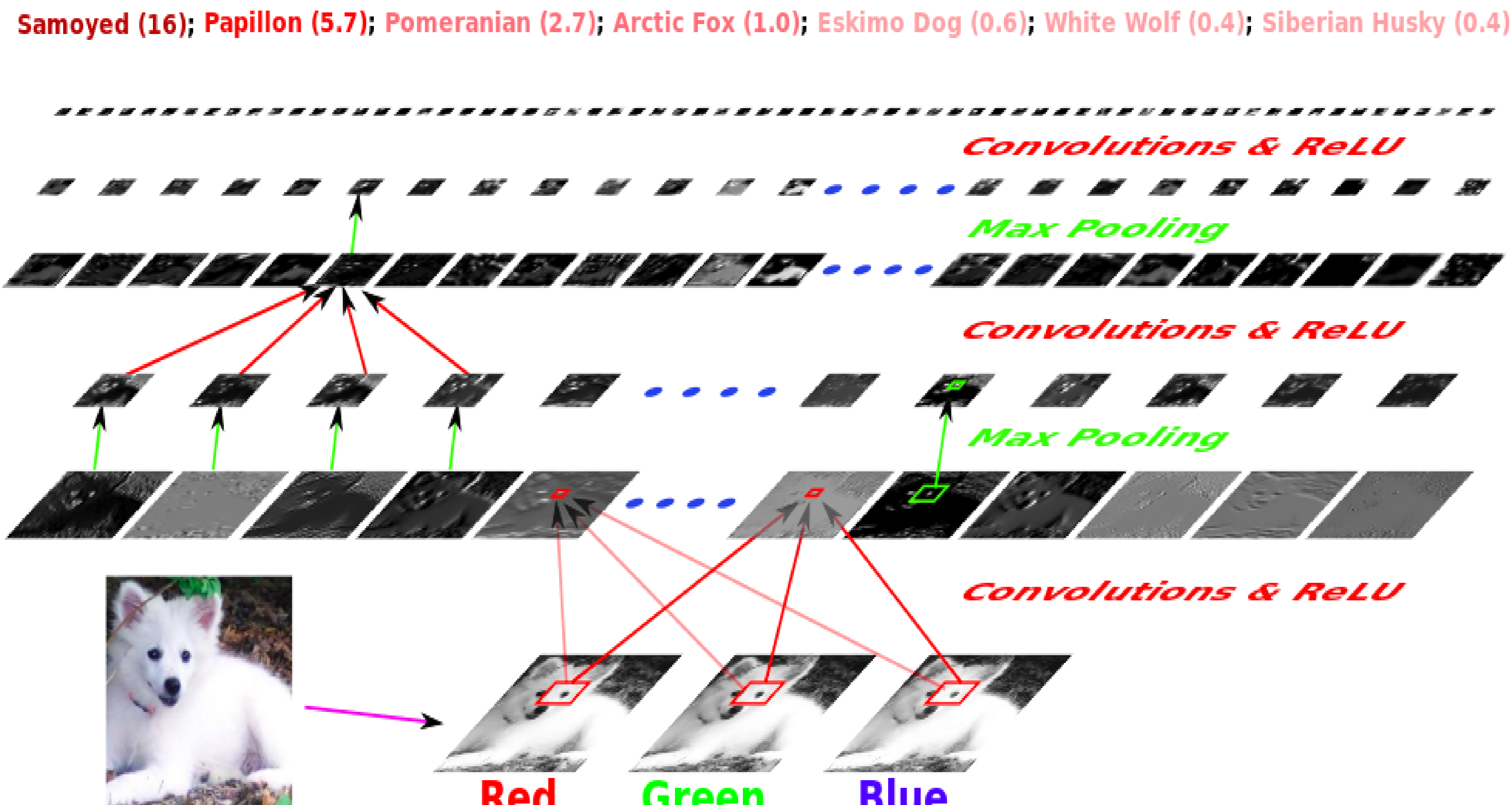


Backpack

Racket

Very Deep ConvNet for Object Recognition

- **1 to 10 billion connections, 10 million to 1 billion parameters, 8 to 20 layers.**



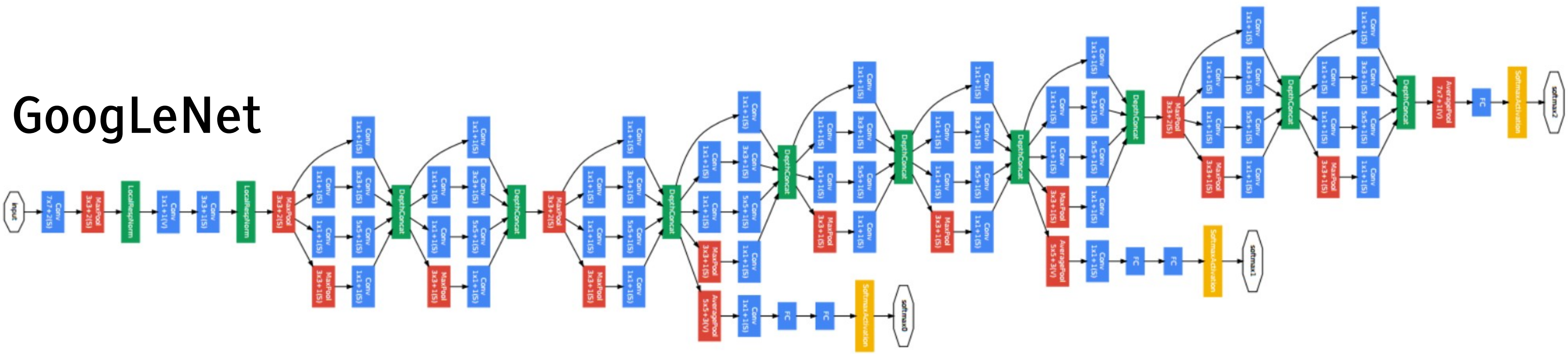
Very Deep ConvNet Architectures

Y LeCun

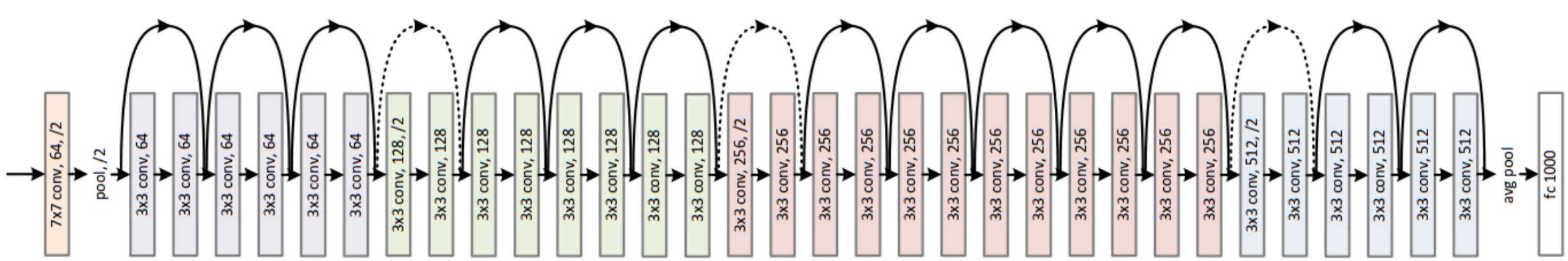
- Small kernels, not much subsampling (fractional subsampling).

VGG

GoogLeNet



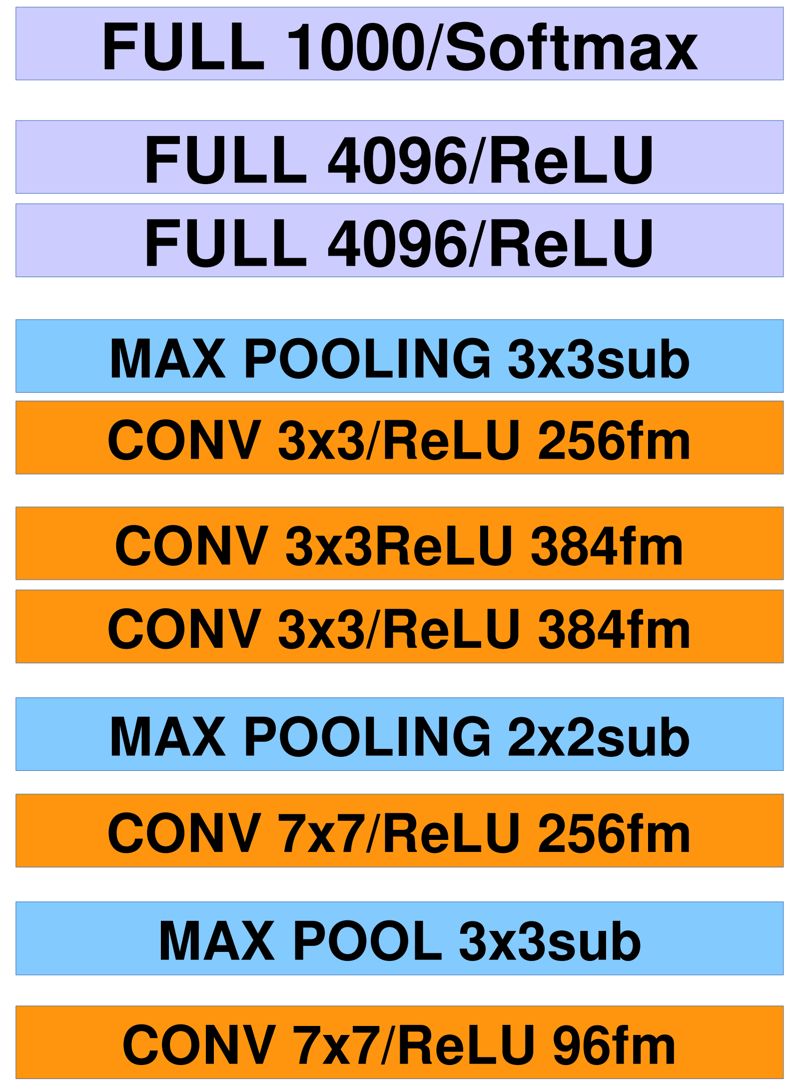
ResNet



Very Deep ConvNets Trained on GPU

- AlexNet [Krizhevski, Sutskever, Hinton 2012]
  - 15% top-5 error on ImageNet
- OverFeat [Sermanet et al. 2013]
  - 13.8%
- VGG Net [Simonyan, Zisserman 2014]
  - 7.3%
- GoogLeNet [Szegedy et al. 2014]
  - 6.6%
- ResNet [He et al. 2015]
  - 5.7%

- http://torch.ch
- https://github.com/torch/torch7/wiki/Cheatsheet



Learning in Action

- **How the filters in the first layer learn**

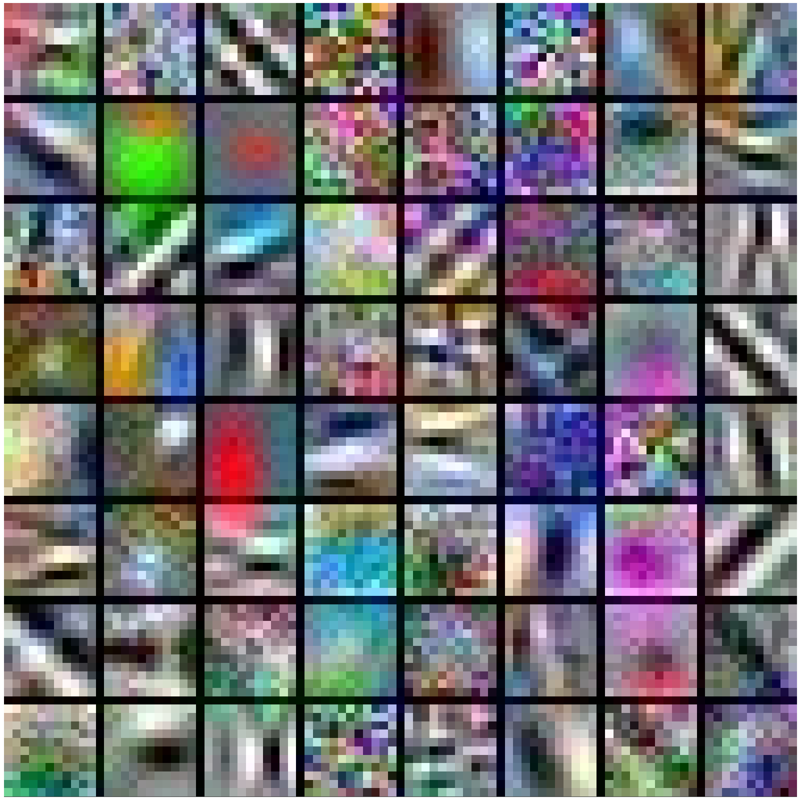


Image captioning: generating a descriptive sentence

Y LeCun

[Lebret, Pinheiro, Collobert 2015][Kulkarni 11][Mitchell 12][Vinyals 14][Mao 14]

A man riding skis on a snow covered ski slope.

NP: a man, skis, the snow, a person, a woman, a snow covered slope, a slope, a snowboard, a skier, man.
VP: wearing, riding, holding, standing on, skiing down.
PP: on, in, of, with, down.

A man wearing skis on the snow.

A man is doing skateboard tricks on a ramp.

NP: a skateboard, a man, a trick, his skateboard, the air, a skateboarder, a ramp, a skate board, a person, a woman.
VP: doing, riding, is doing, performing, flying through.
PP: on, of, in, at, with.

A man riding a skateboard on a ramp.

The girl with blue hair stands under the umbrella.

NP: a woman, an umbrella, a man, a person, a girl, umbrellas, that, a little girl, a cell phone.
VP: holding, wearing, is holding, holds, carrying.
PP: with, on, of, in, under.

A woman is holding an umbrella.

A slice of pizza sitting on top of a white plate.

NP: a plate, a white plate, a table, pizza, it, a pizza, food, a sandwich, top, a close.
VP: topped with, has, is, sitting on, is on.
PP: of, on, with, in, up.

A table with a plate of pizza on a white plate.

A baseball player swinging a bat on a field.

NP: the ball, a game, a baseball player, a man, a tennis court, a ball, home plate, a baseball game, a batter, a field.
VP: swinging, to hit, playing, holding, is swinging.
PP: on, during, in, at, of.

A baseball player swinging a bat on a baseball field.

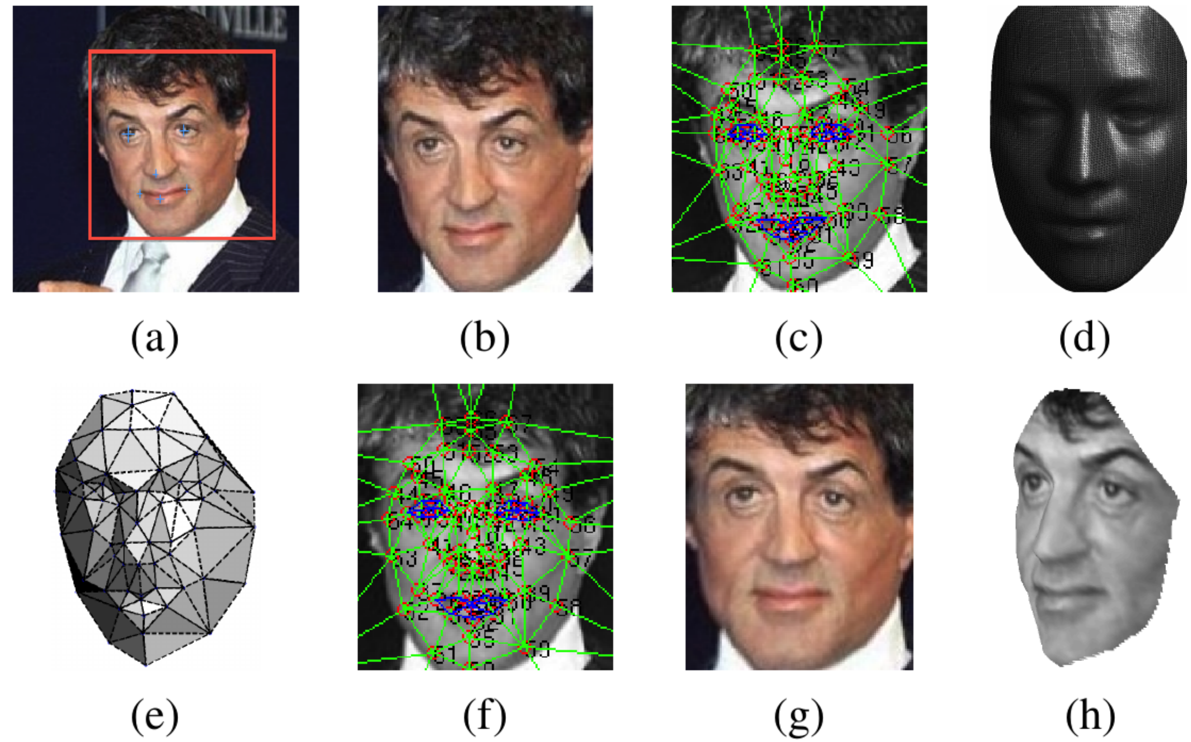
A bunch of kites flying in the sky on the beach.

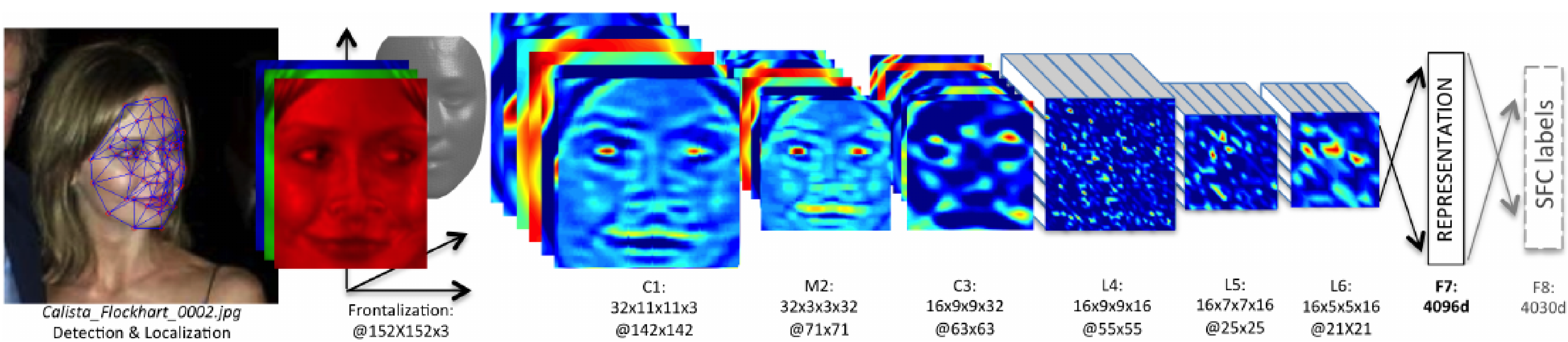
NP: the beach, a beach, a kite, kites, the ocean, the water, the sky, people, a sandy beach, a group.
VP: flying, flies, is flying, flying in, are.
PP: on, of, with, in, at.

People flying kites on the beach.

Deep Face

- **[Taigman et al. CVPR 2014]**
  - Alignment
  - ConvNet
  - Metric Learning
- **Deployed at Facebook for Auto-tagging**
  - 800 million photos per day





Person Detection and Pose Estimation

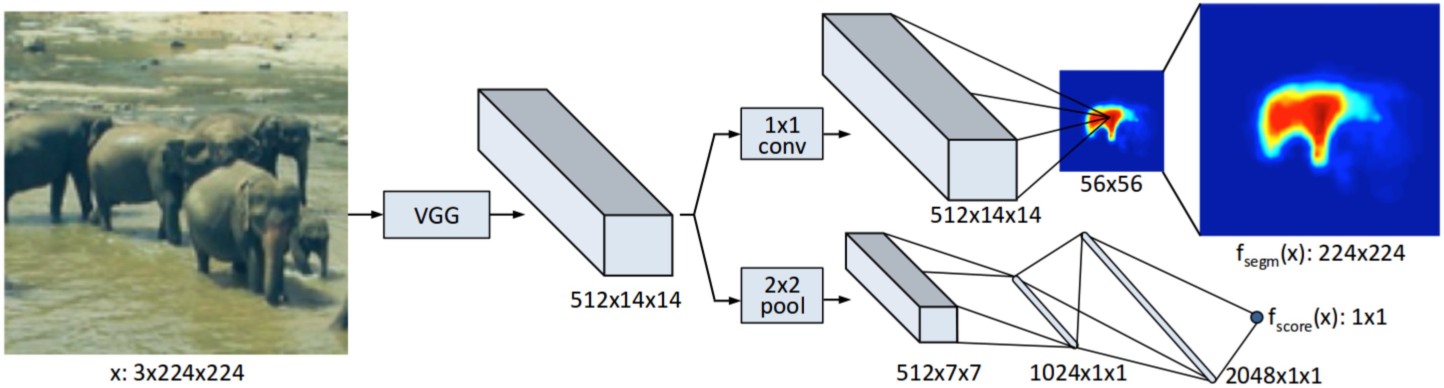
Y LeCun

Tompson, Goroshin, Jain, LeCun, Bregler arXiv:1411.4280 (2014)

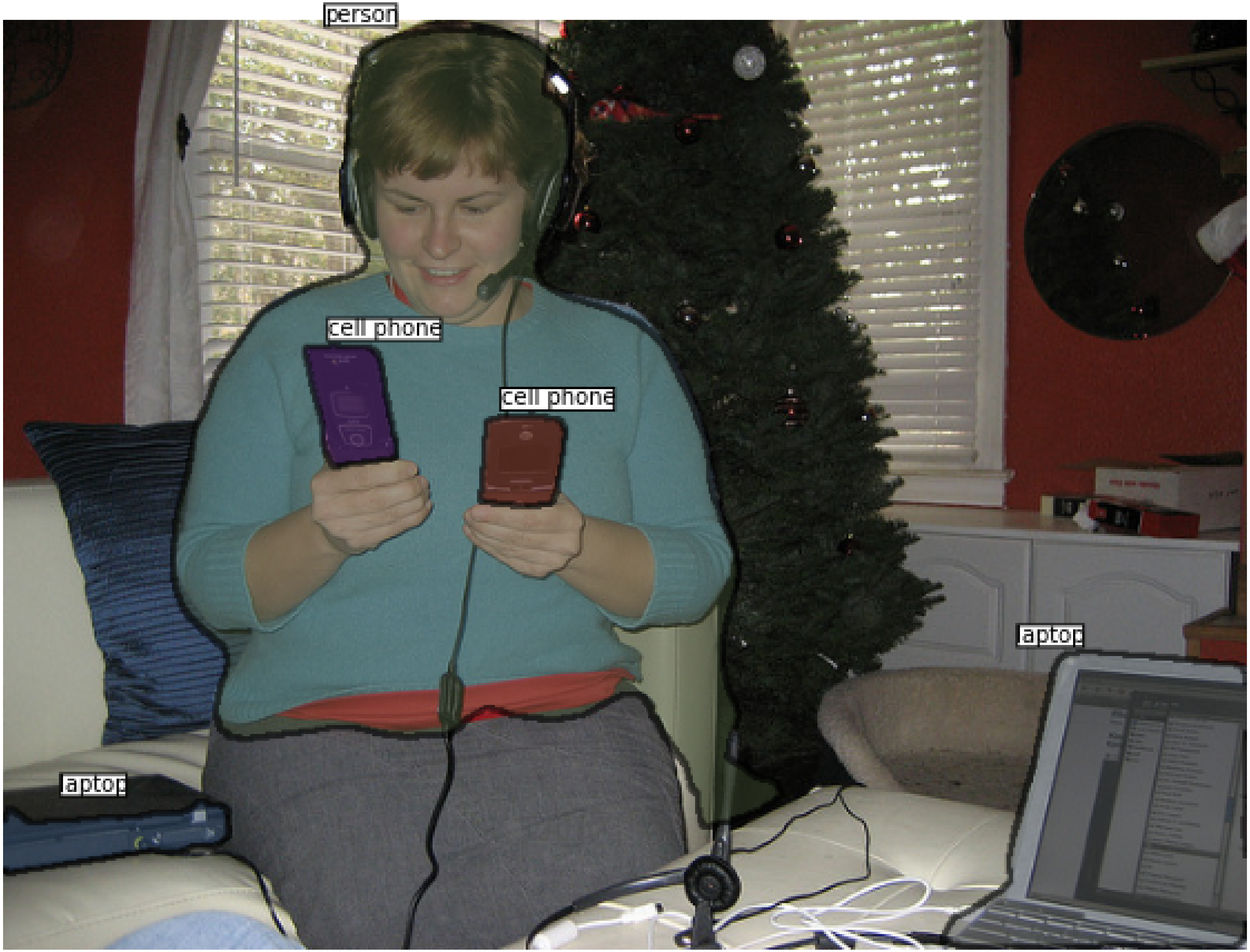
Segmenting and Localizing Objects (DeepMask)

Y LeCun

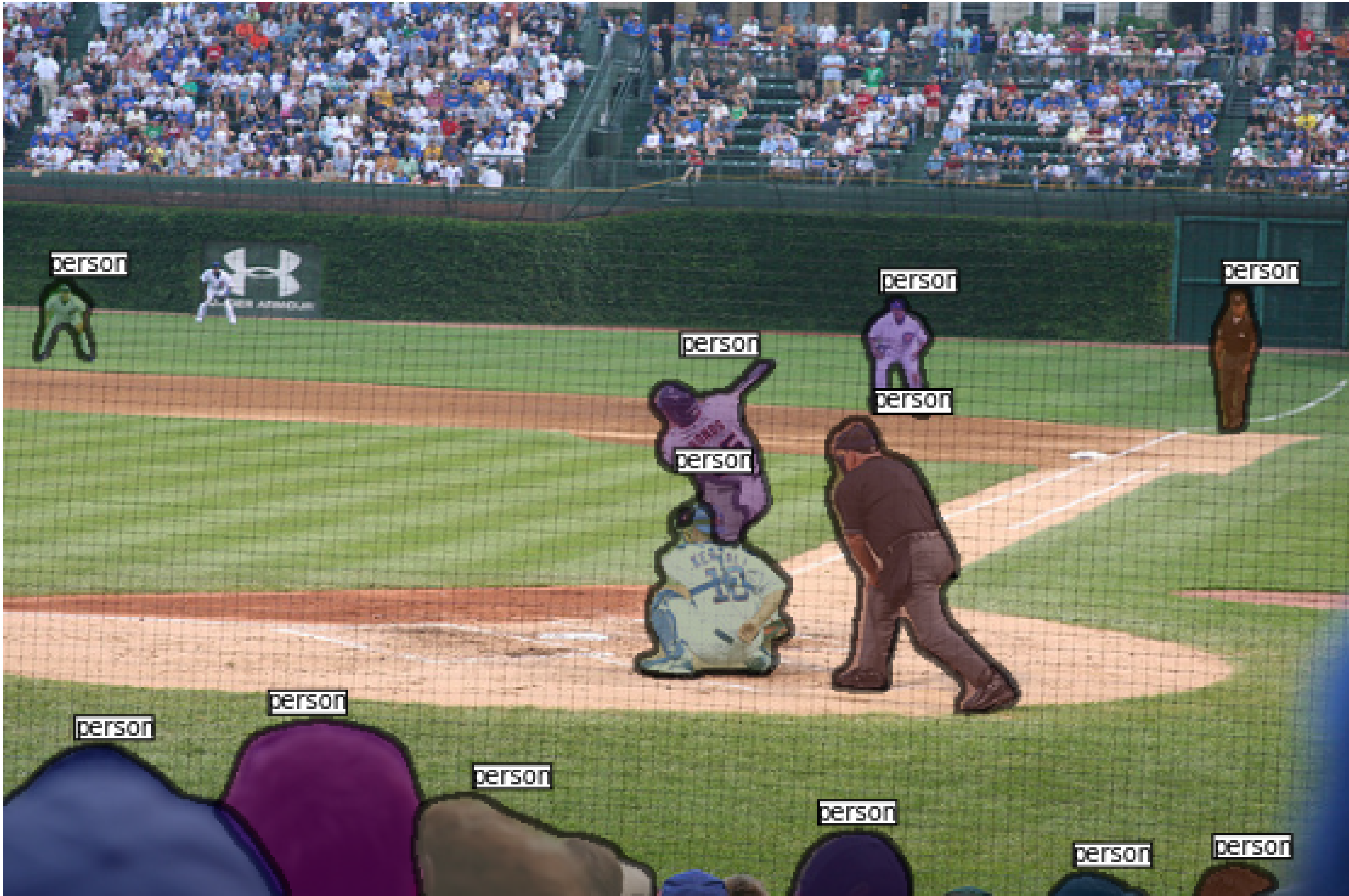
- [Pinheiro, Collobert, Dollar ICCV 2015]
  - ConvNet produces object masks



Results

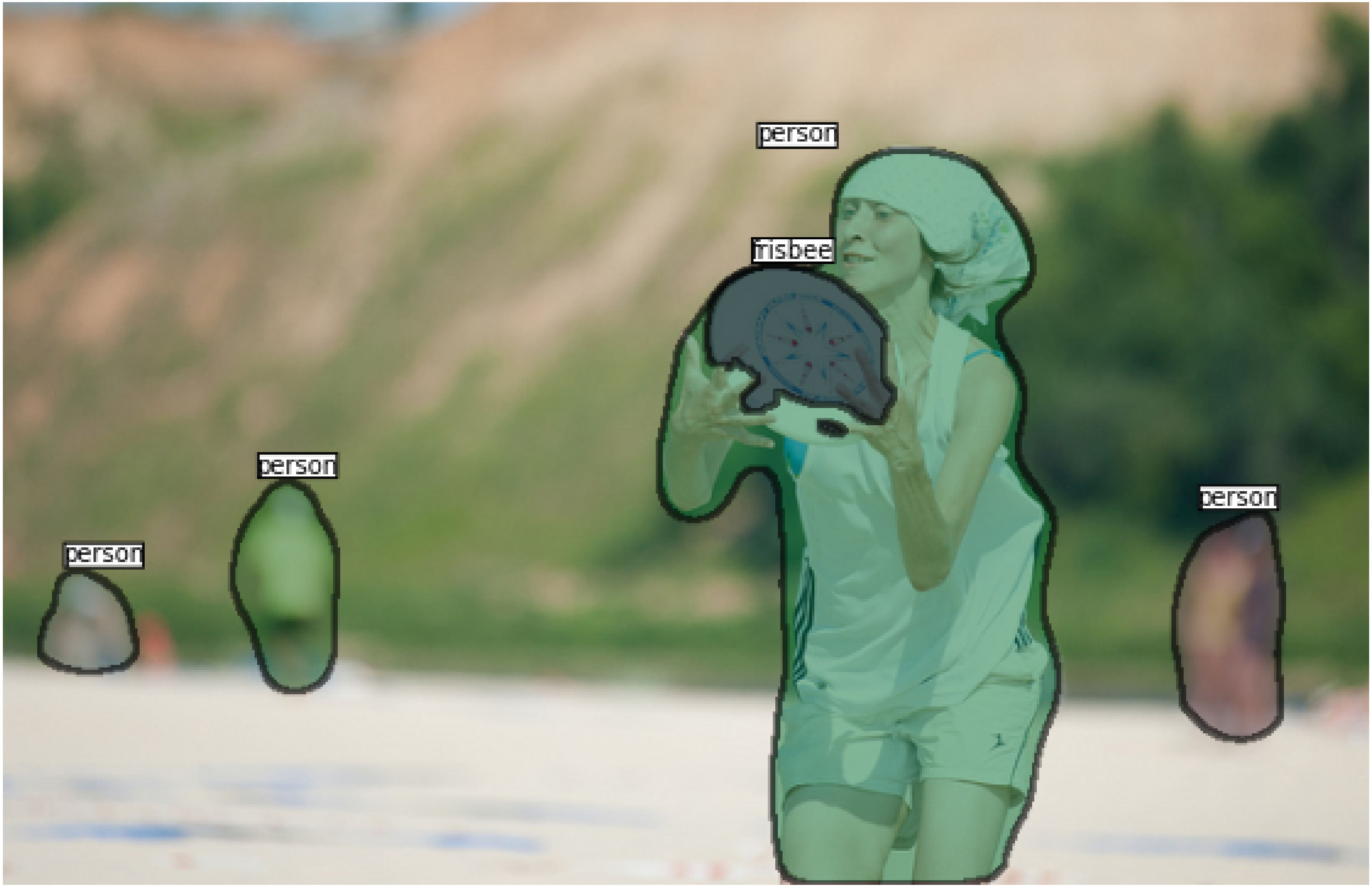


Results

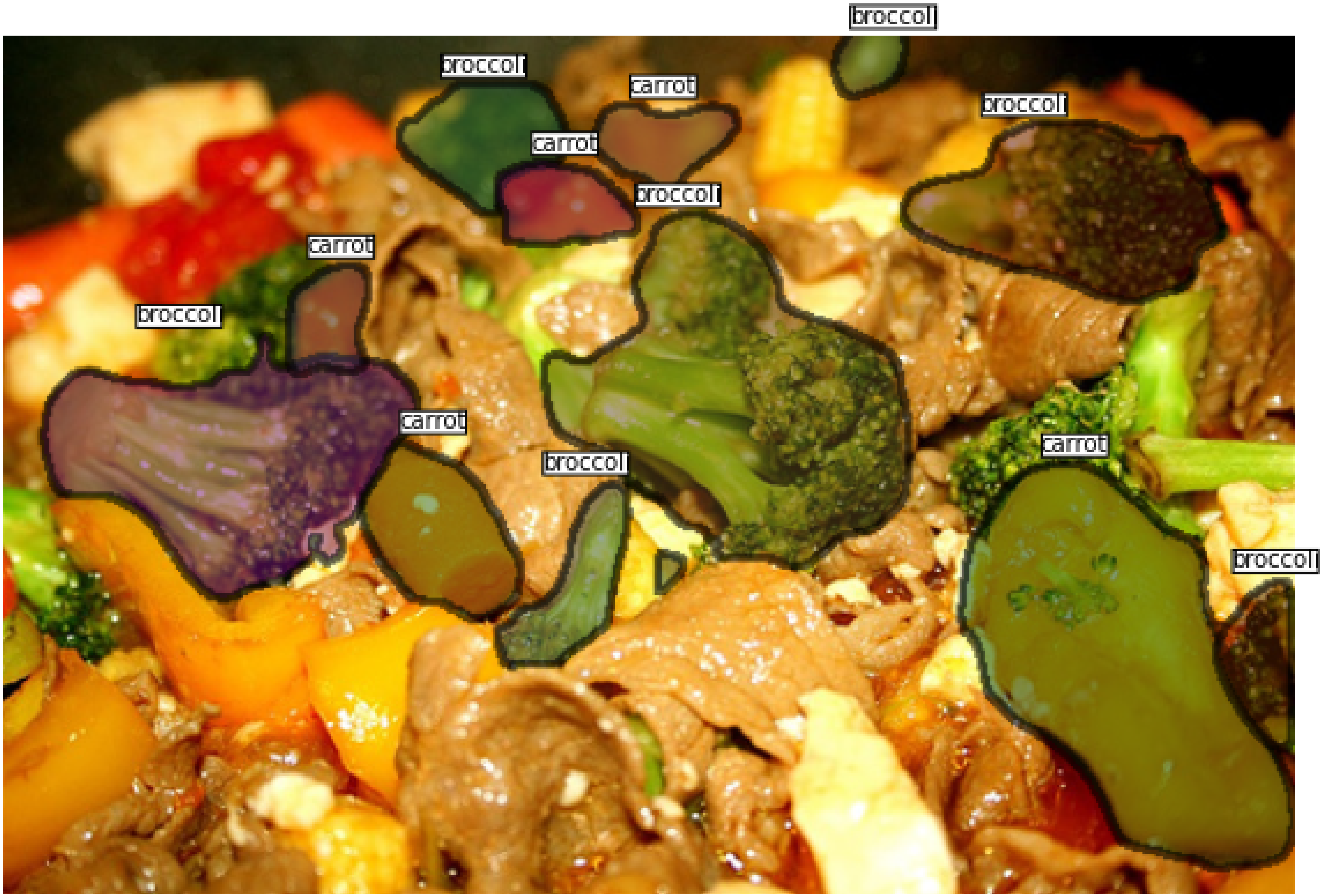

person
person
person
person
person
person
person
person
person
person
person
person

Results

Results

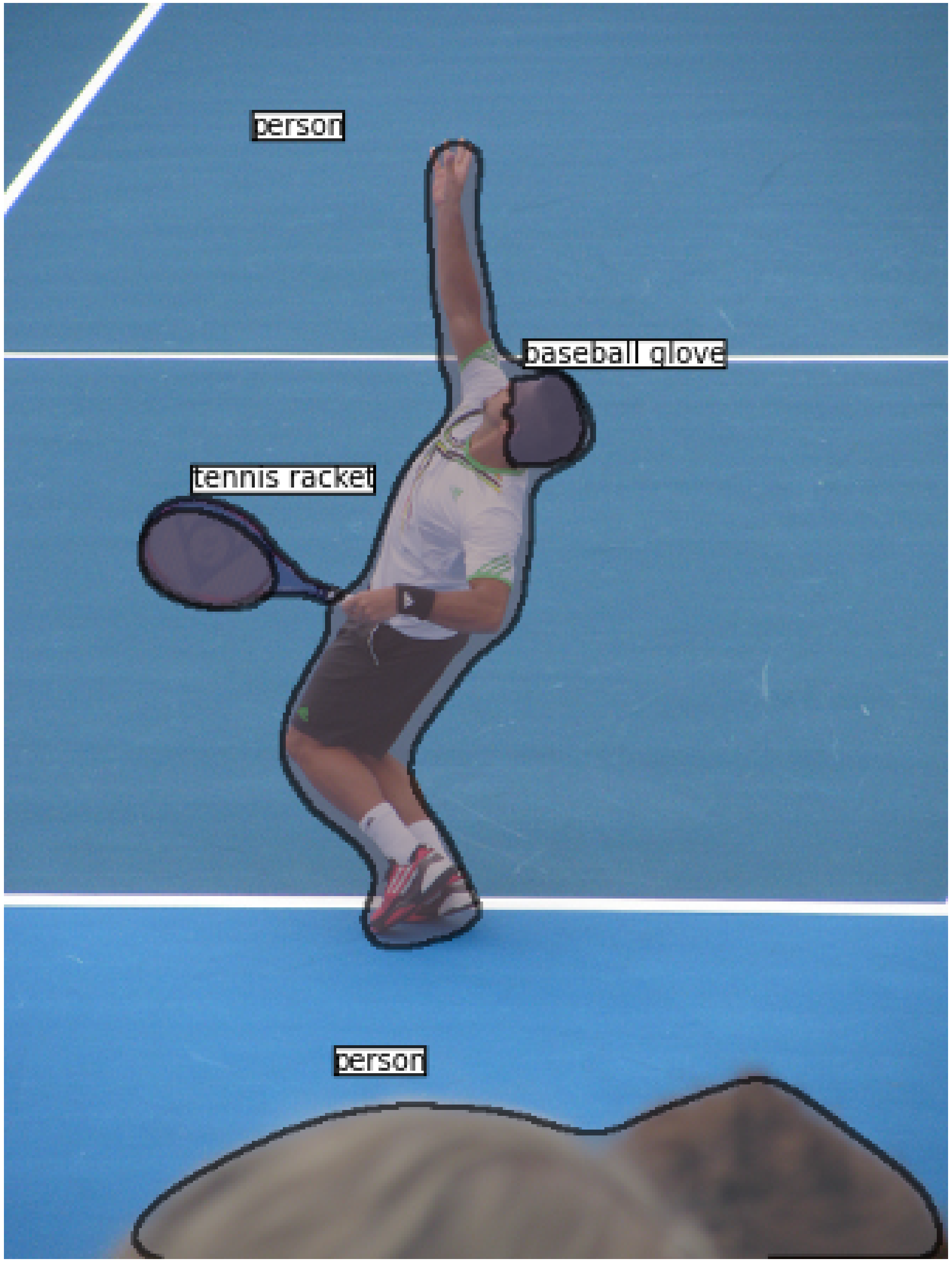


Results


broccol
broccol
carrot
broccol
carrot
broccol
carrot
broccol
carrot
broccol
carrot
broccol

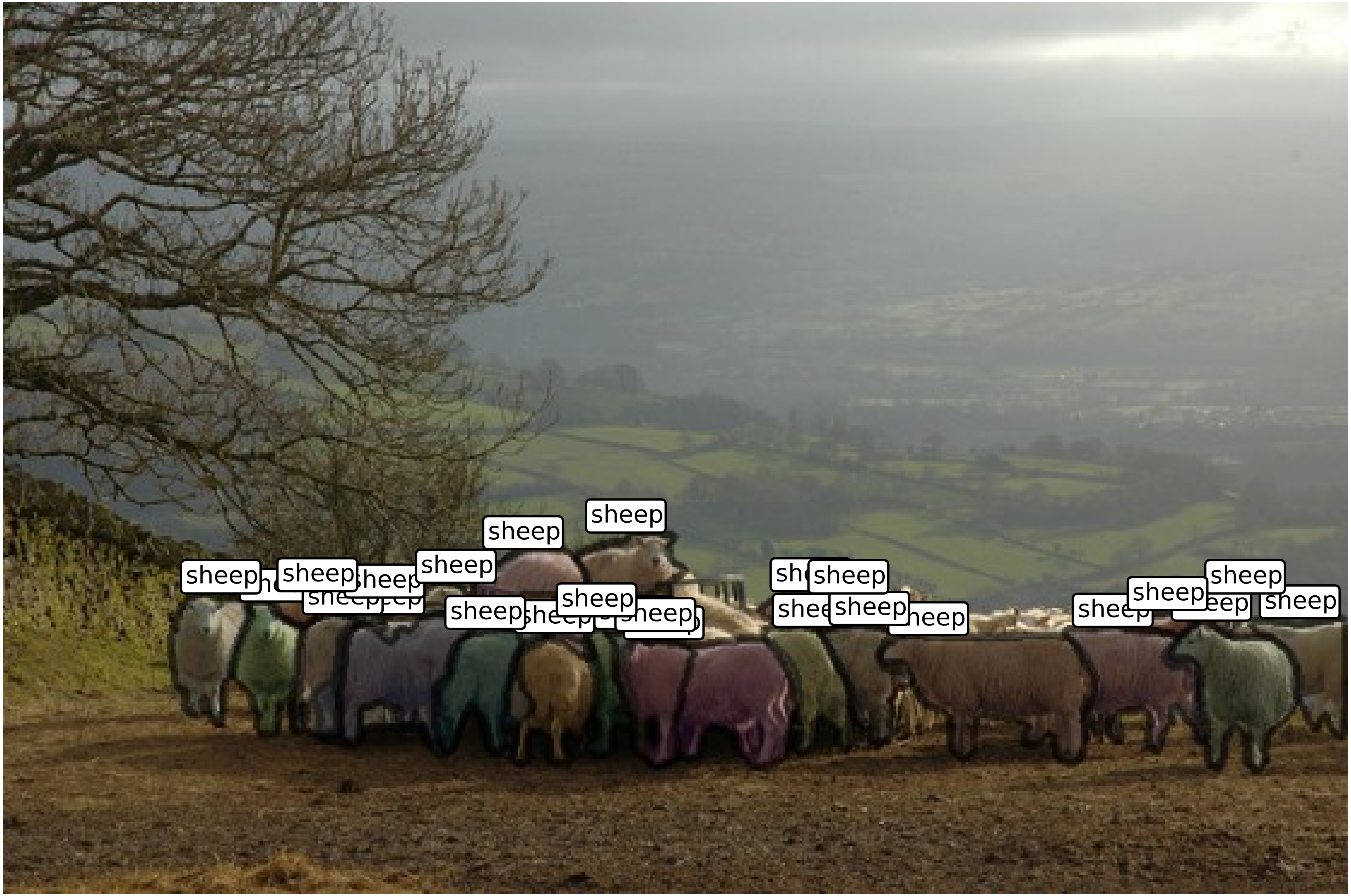
Mistakes

Y LeCun



Results

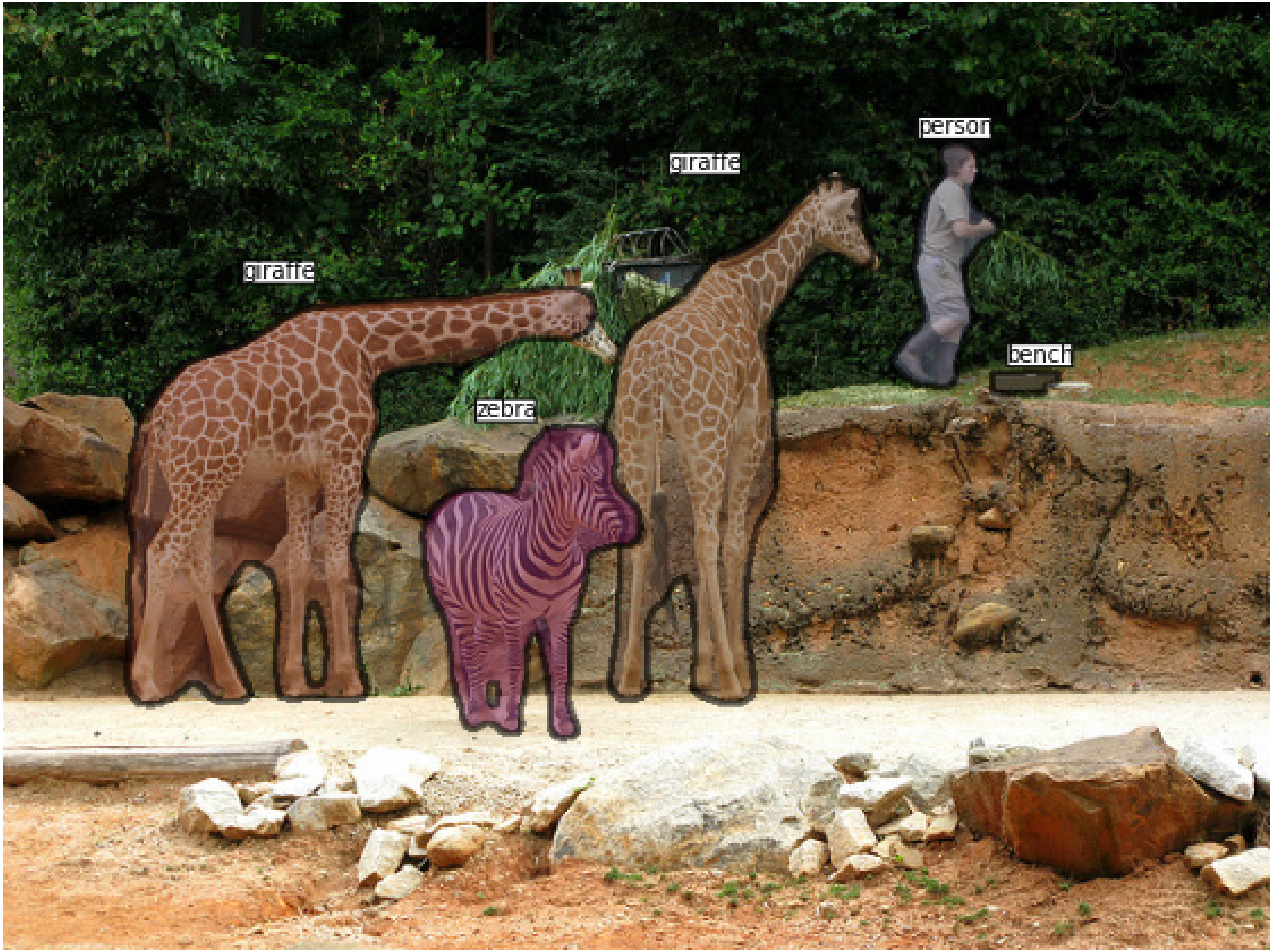
Results


sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep

Results

Y LeCun

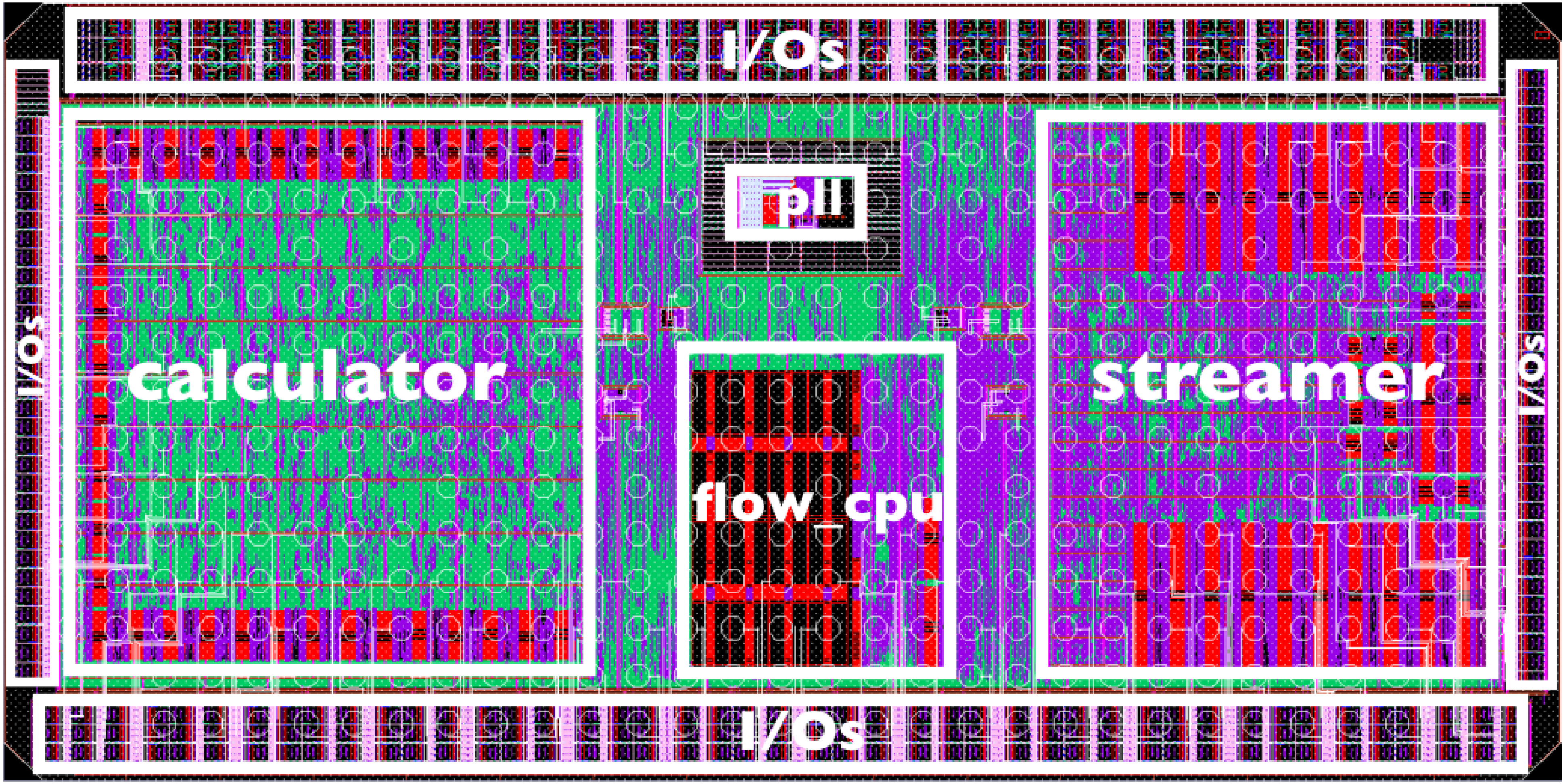
Results



ConvNets are Everywhere
(or soon will be)

ConvNet Chips

- Currently in development at NVIDIA, Intel, Mobileye, Qualcomm, Samsung
- Many startups: Movidius, Teradeep, Nervana….
- Soon, a ConvNet chip will drive your car.



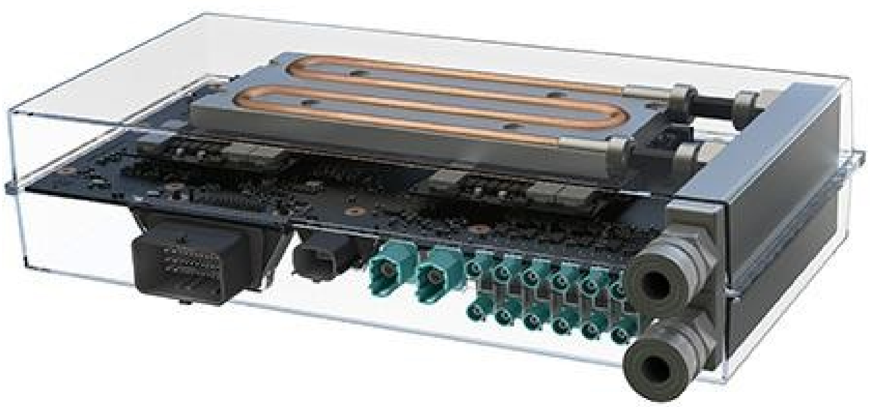
NeuFlow chip[Pham, Jelaca, Farabet, Martini, LeCun, Culurciello 2012]

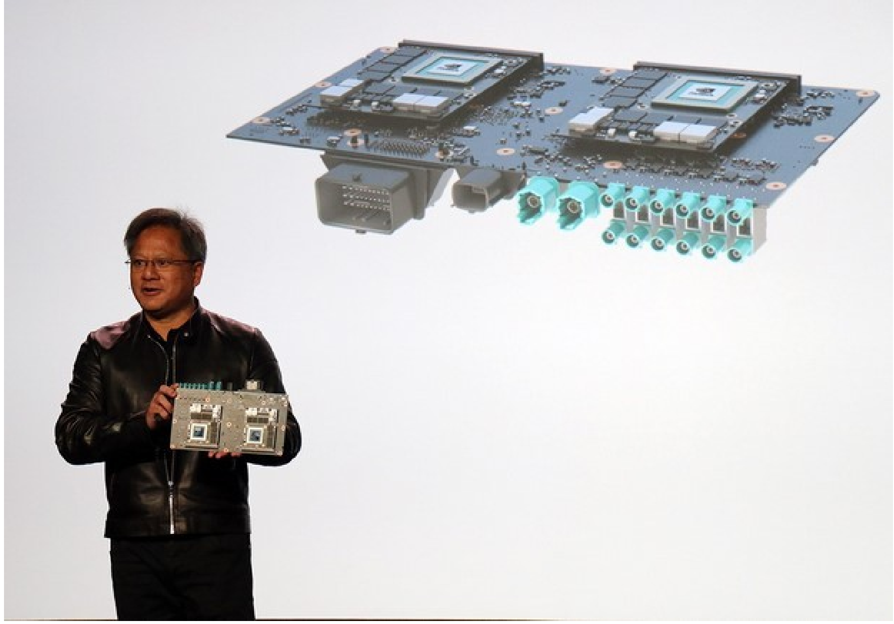
NVIDIA: ConvNet-Based Driver Assistance

Drive-PX2: Open Platform for Driver Assistance

Embedded Super-Computer: 42 TOPS

- (=150 Macbook Pros)

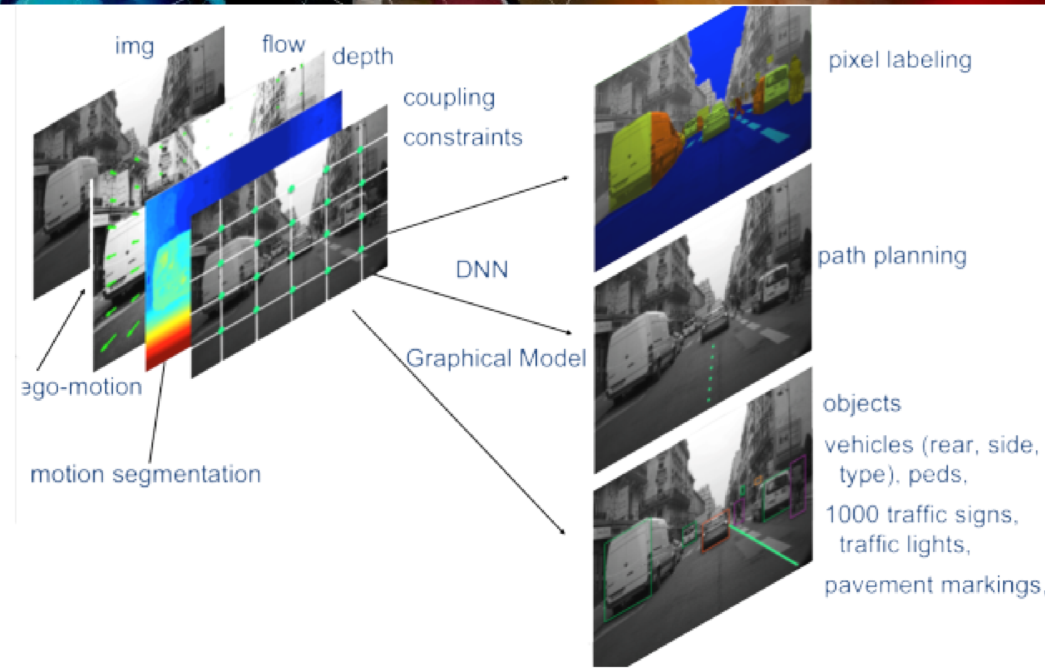


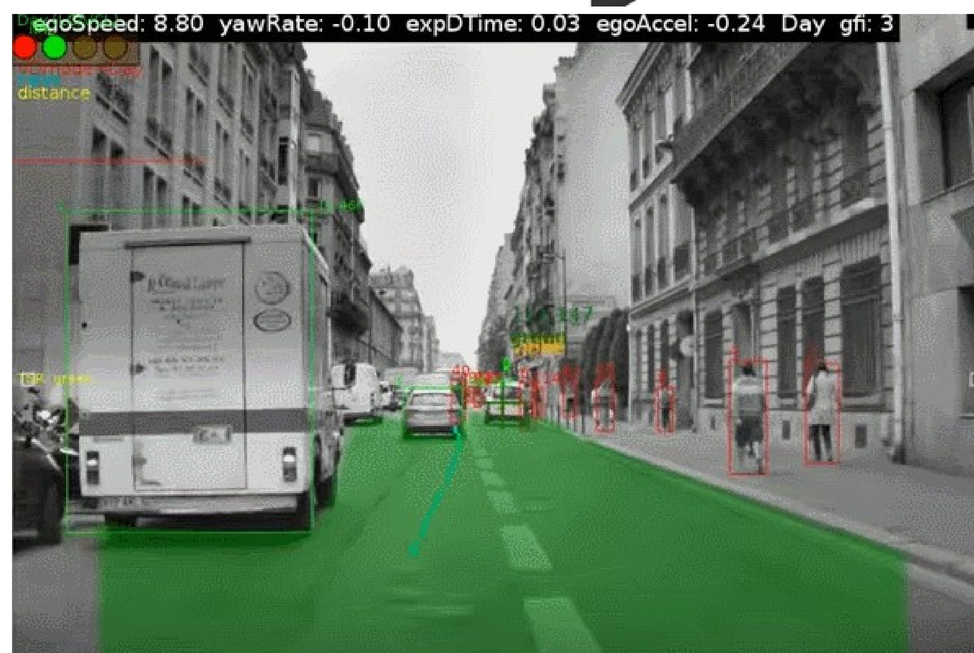


MobilEye: ConvNet-Based Driver Assistance

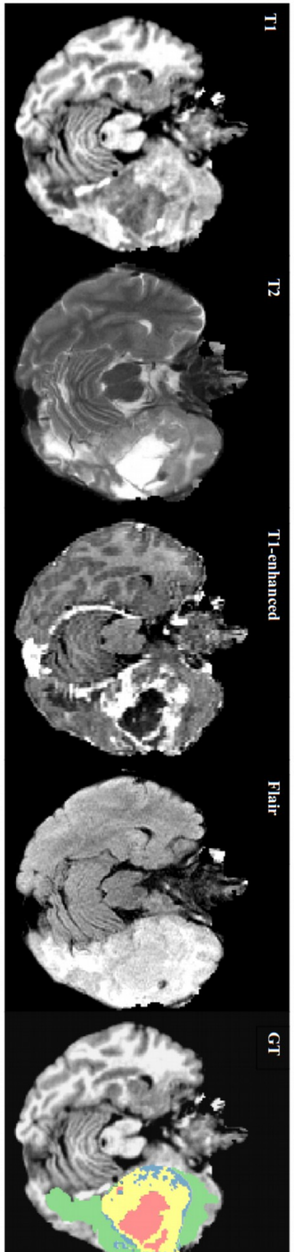
Deployed in the latest

Tesla Model S and Model X

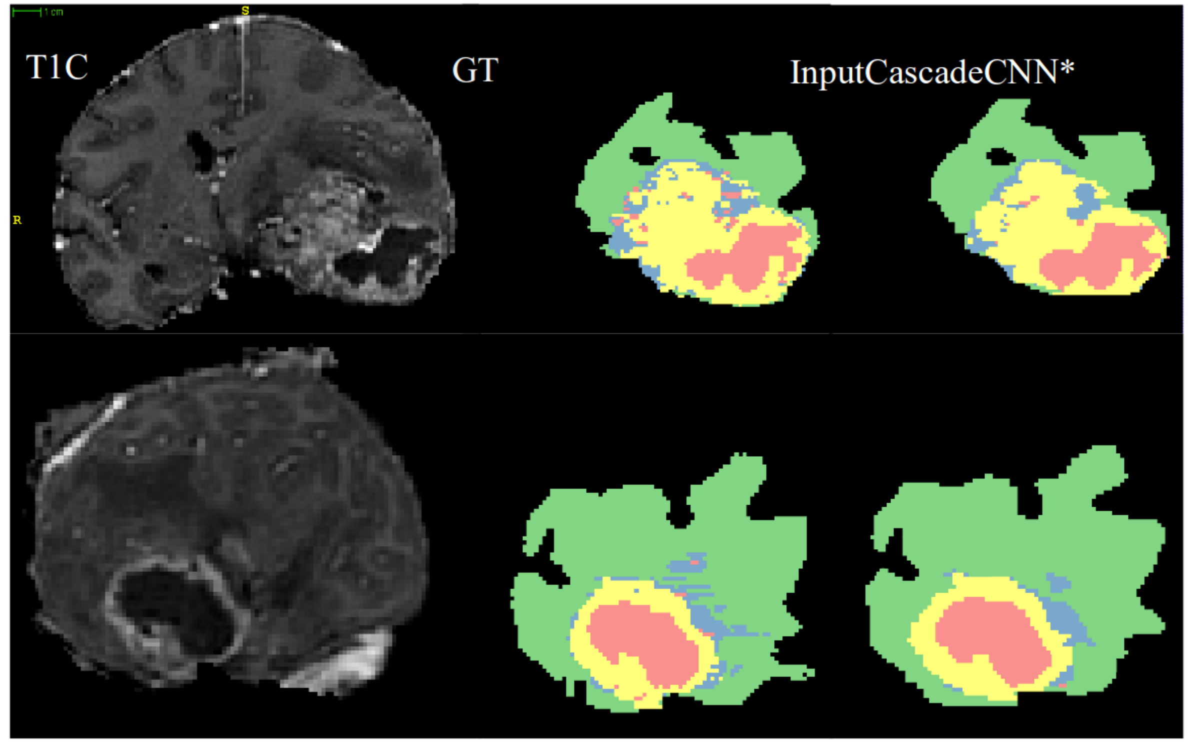




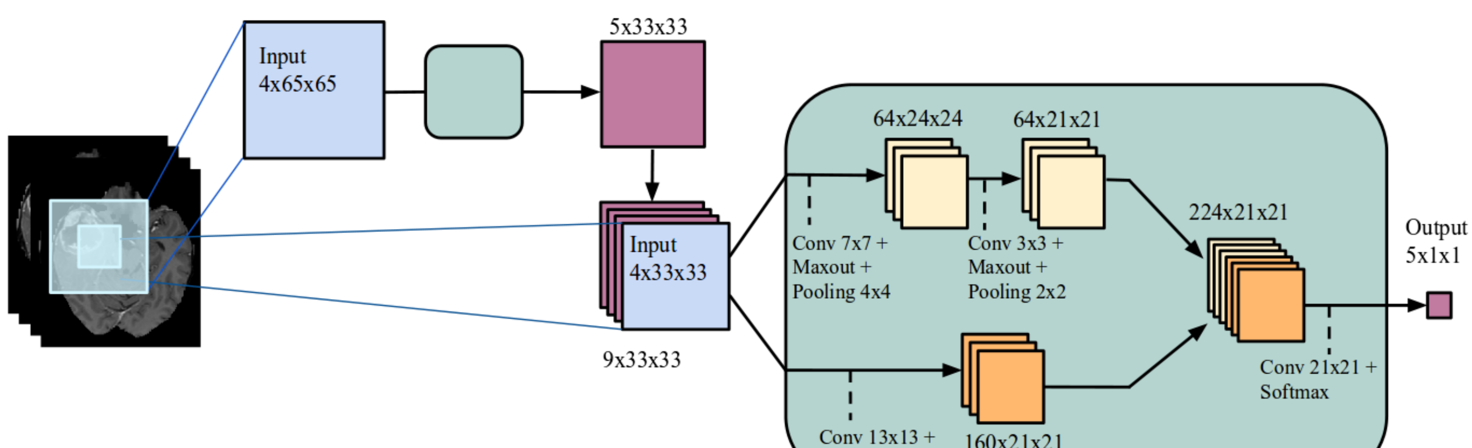
Brain Tumor Detection



- [Havaei et al. 2015]
  - Arxiv:1505.03540
- InputCascadeCNN architecture
  - 802,368 parameters
- Trained on 30 patients.
- State of the art results on BRAT2013



edema, enhanced tumor, necrosis, non-enhanced tumor.



Parameters 802,368

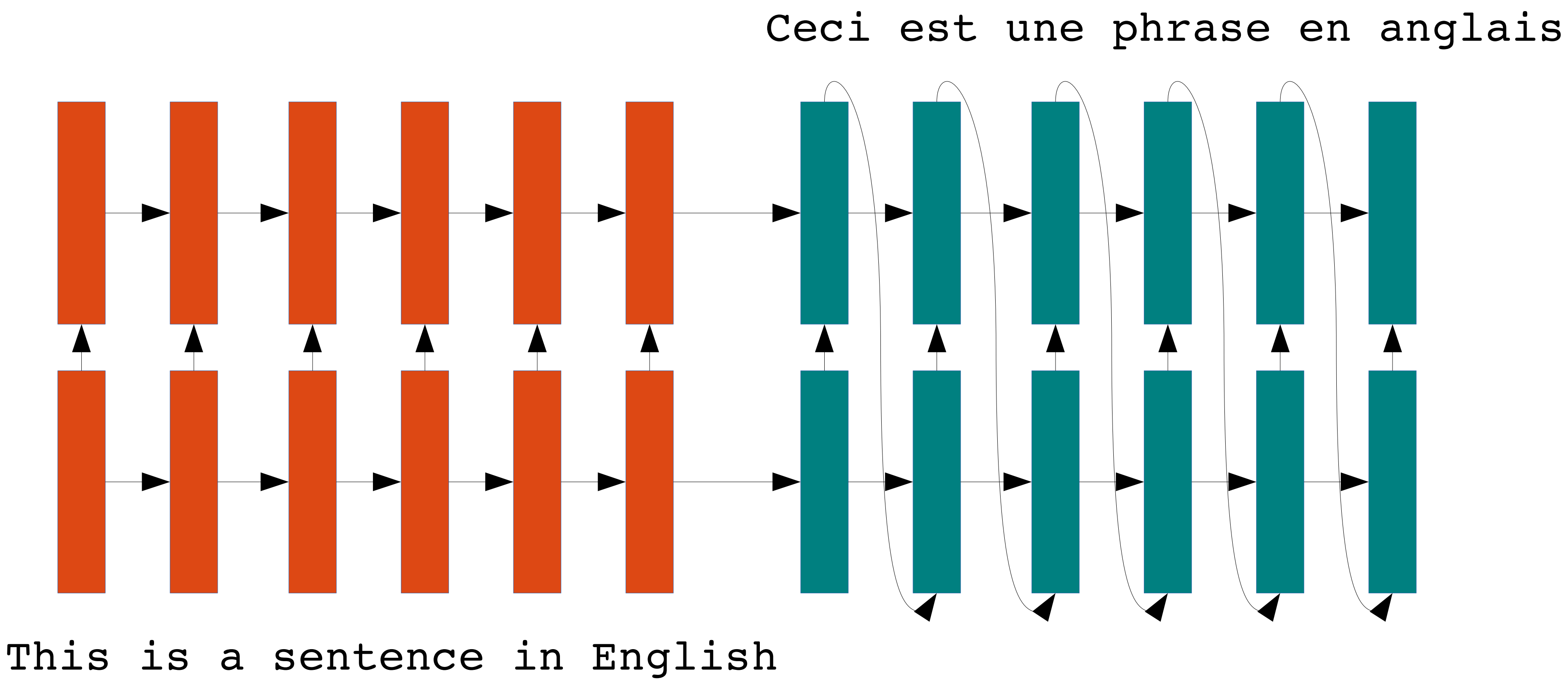
Deep Learning is Everywhere (ConvNets are Everywhere)

- **Lots of applications at Facebook, Google, Microsoft, Baidu, Twitter, IBM…**
  - Image recognition for photo collection search
  - Image/Video Content filtering: spam, nudity, violence.
  - Search, Newsfeed ranking
- **People upload one billion photos on Facebook every day**
  - (over 2 billion photos per day if we count Instagram, Messenger and Whatsapp)
- **Each photo on Facebook goes through two ConvNets within 2 seconds**
  - One for image recognition/tagging
  - One for face recognition (not activated in Europe).
- **Soon ConvNets will really be everywhere:**
  - self-driving cars, medical imaging, augmented reality, mobile devices, smart cameras, robots, toys…..

Natural Language Understanding

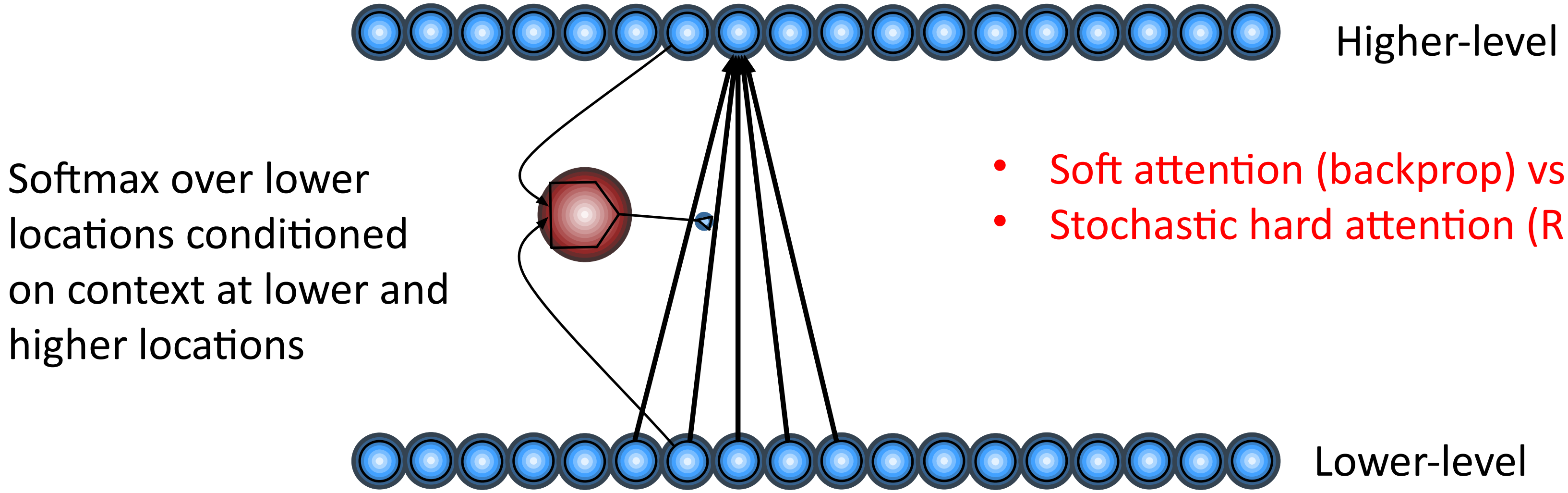
Language Translation with LSTM networks

- [Sutskever et al. NIPS 2014]
  - Multiple layers of very large LSTM recurrent modules
  - English sentence is read in and encoded
  - French sentence is produced after the end of the English sentence
  - Accuracy is very close to state of the art.



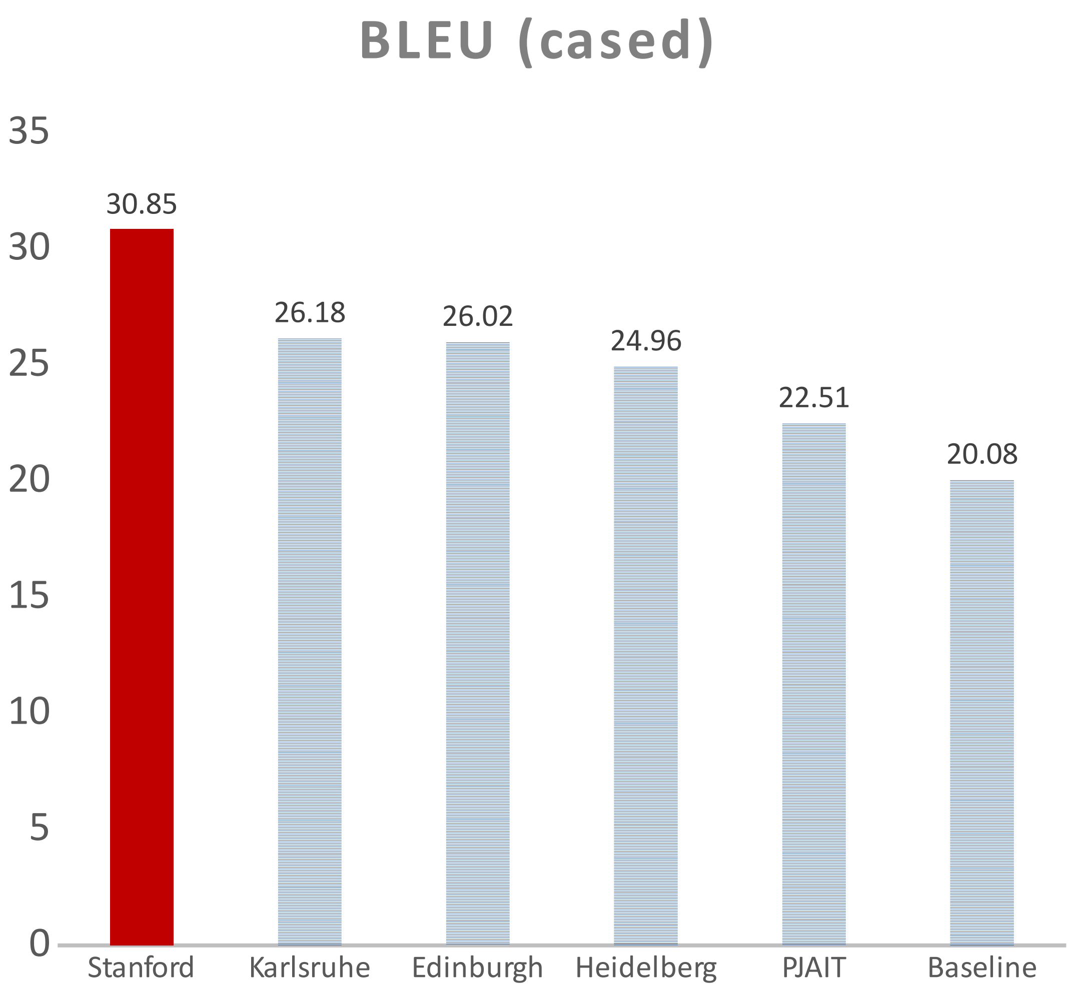
Gating and Attention

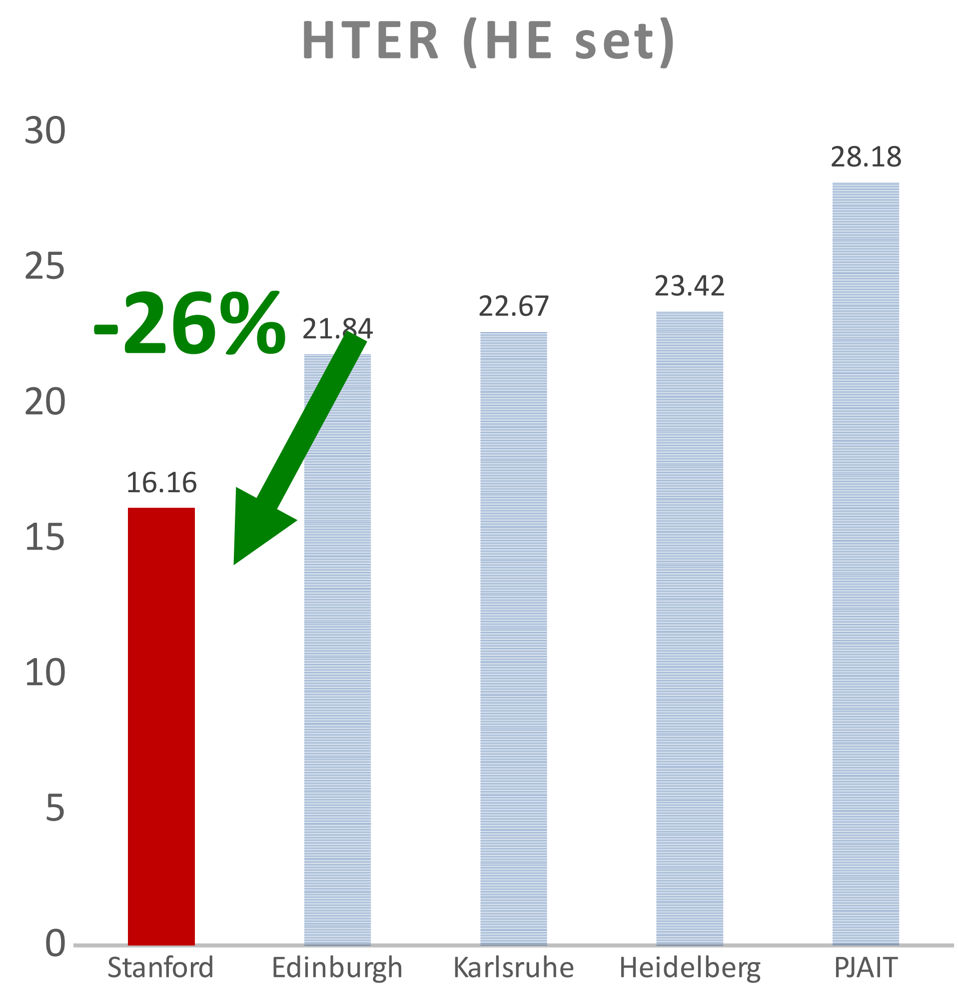
- Connections at activated depending on context
  - (Bahdanau, Cho & Bengio, arXiv sept. 2014)
  - following up on (Graves 2013) and (Larochelle & Hinton NIPS 2010)
- Input of a unit is selected among several by the softmax output of a sub-network
  - The unit “pays attention” to a particular location



- Soft attention (backprop) vs
- Stochastic hard attention (RL)

IWSLT 2015 – *Luong & Manning (2015)*
TED talk MT, English-German

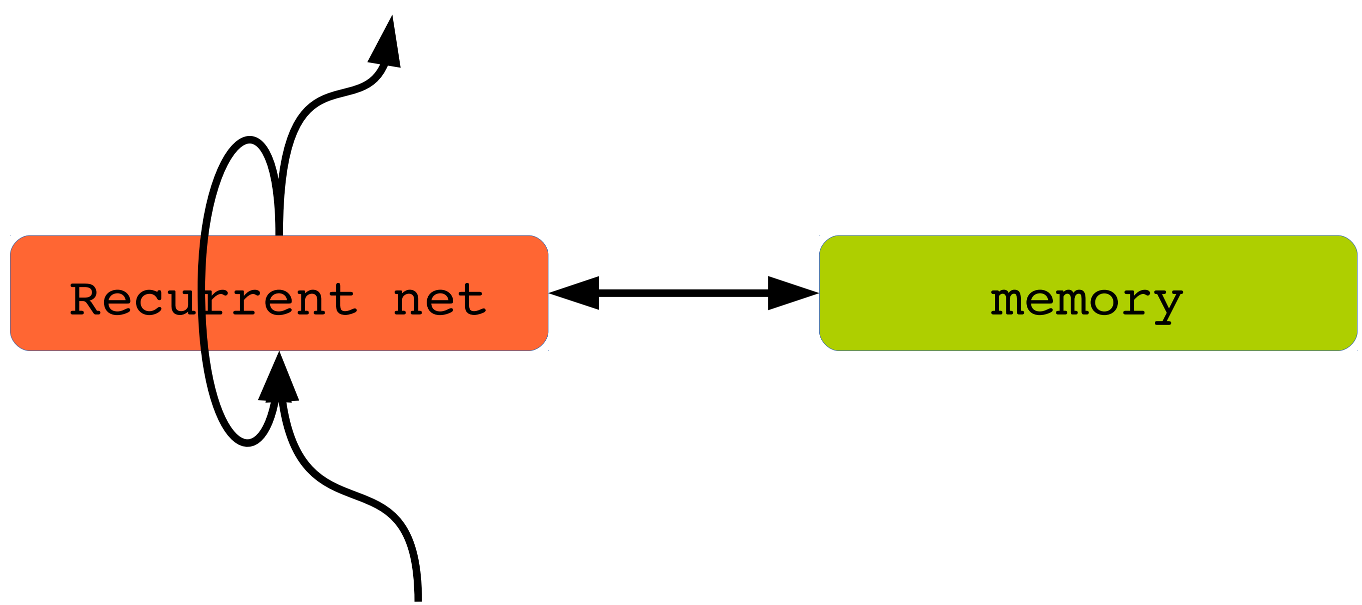




[From Bengio&LeCun tutorial NIPS 2015]

But How can Neural Nets Remember Things?

- **Recurrent networks cannot remember things for very long**
  - The cortex only remember things for 20 seconds
- **We need a “hippocampus” (a separate memory module)**
  - LSTM [Hochreiter 1997], registers
  - **Memory networks** [Weston et 2014] (FAIR), associative memory
  - **Stacked-Augmented Recurrent Neural Net** [Joulin & Mikolov 2014] (FAIR)
  - NTM [DeepMind 2014], “tape”.

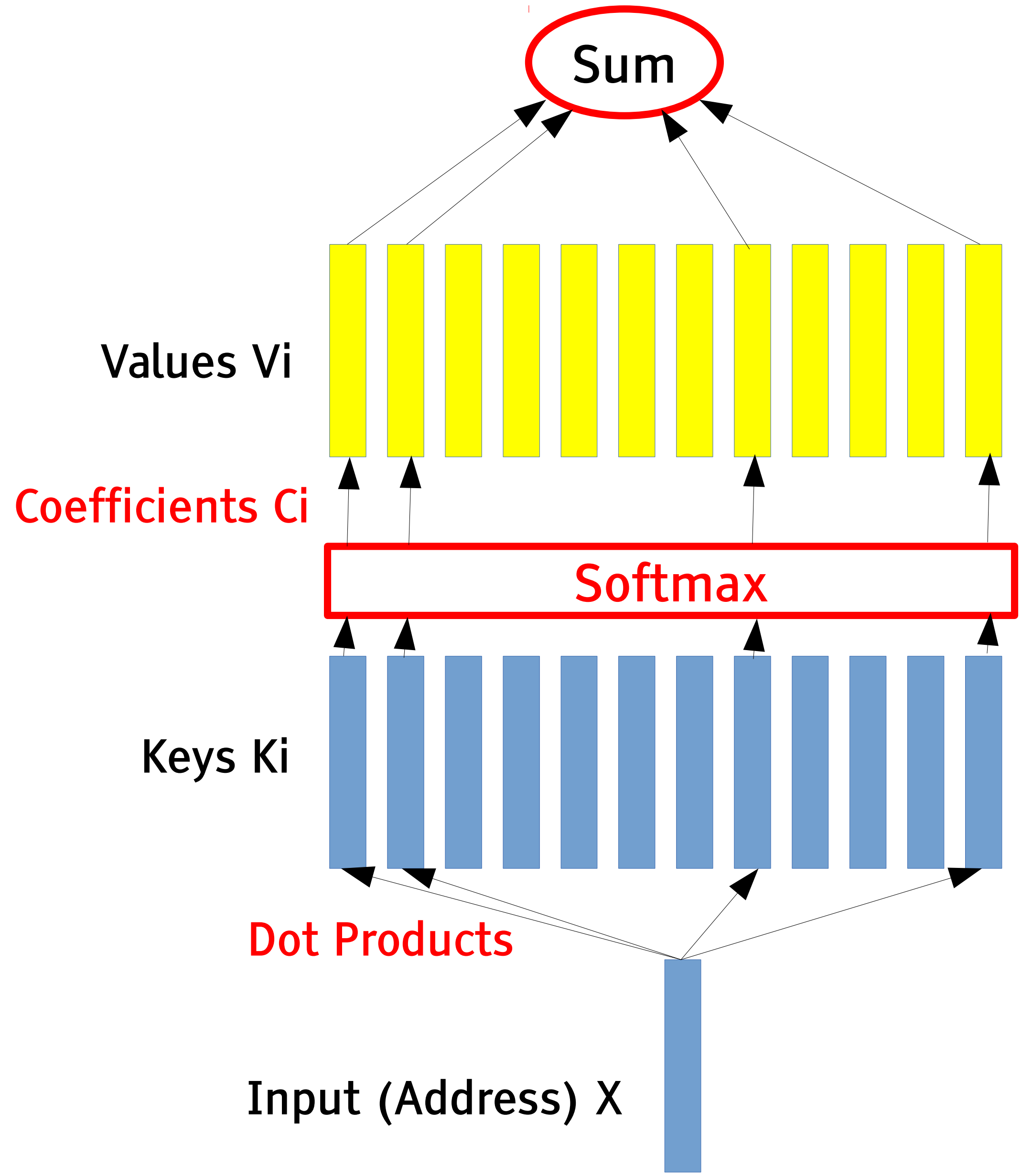


Differentiable Memory

- Like a “soft” RAM circuit
- Or a “soft” hash table
- Stores Key-Value pairs (Ki,Vi)

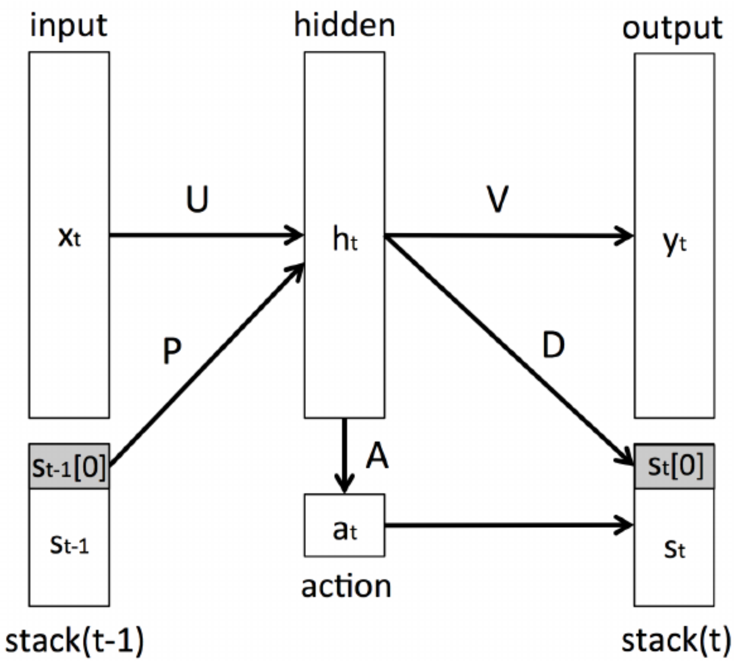
$$C_i = \frac{e^{K_i^T X}}{\sum_j e^{K_j^T X}}$$

$$Y = \sum_i C_i V_i$$

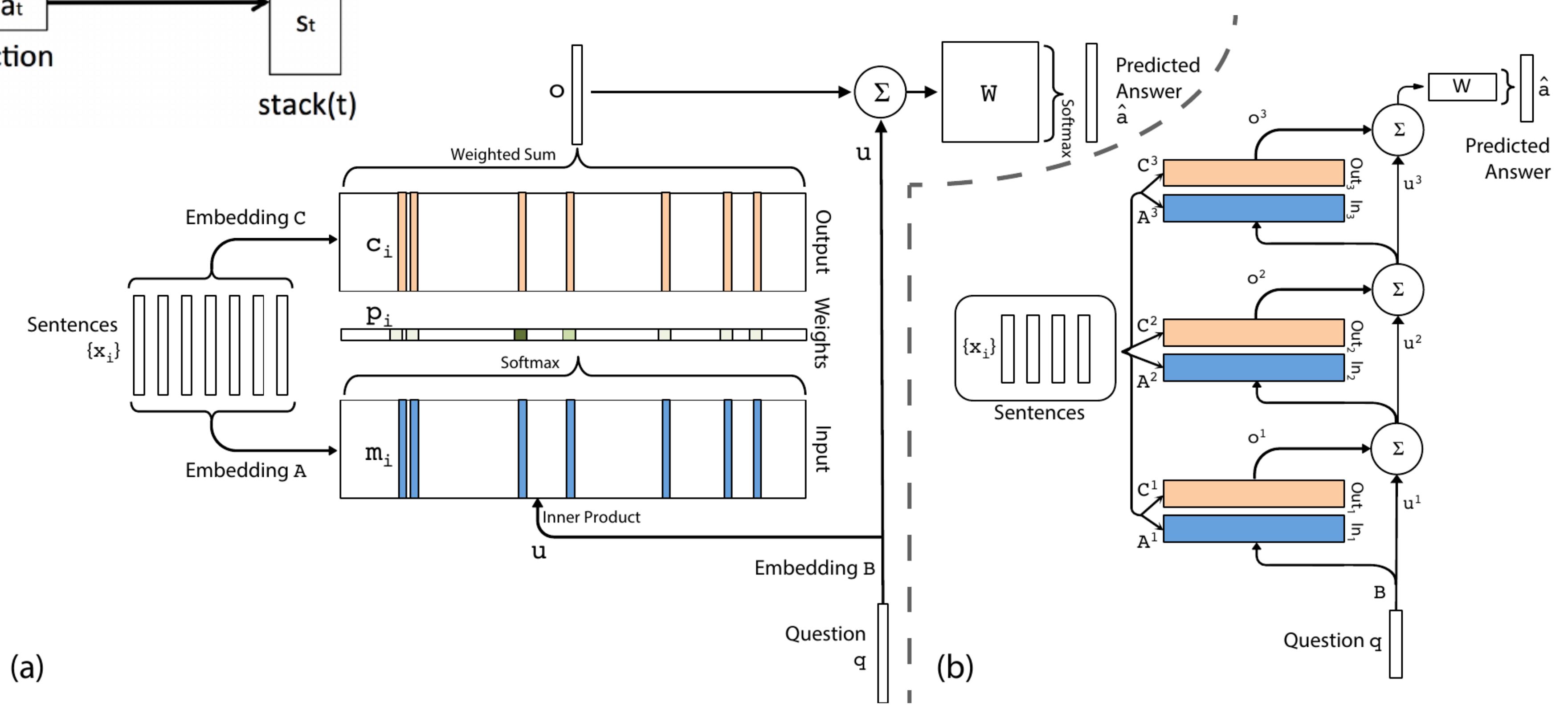


Memory/Stack-Augmented Recurrent Nets

Y LeCun



- [Joulin & Mikolov, ArXiv:1503.01007]
  - Stack-augmented RNN
- [Sukhbataar, Szlam, Weston, Fergus NIPS 2015]
  - ArXiv:1503.08895]
- Weakly-supervised MemNN:
  - discovers which memory location to use.



Memory Network [Weston, Chopra, Bordes 2014]

Add a short-term memory to a network

http://arxiv.org/abs/1410.3916

I: (input feature map) – converts the incoming input to the internal feature representation.
G: (generalization) – updates old memories given the new input.
O: (output feature map) – produces a new output (in the feature representation space), given the new input and the current memory.
R: (response) – converts the output into the response format desired. For example, a textual response or an action.

Method	F1
(Fader et al., 2013) [4]	0.54
(Bordes et al., 2014) [3]	0.73
MemNN	0.71
MemNN (with BoW features)	0.79

Results on Question Answering Task

Bilbo travelled to the cave.
Gollum dropped the ring there.
Bilbo took the ring.
Bilbo went back to the Shire.
Bilbo left the ring there.
Frodo got the ring.
Frodo journeyed to Mount-Doom.
Frodo dropped the ring there.
Sauron died.
Frodo went back to the Shire.
Bilbo travelled to the Grey-havens.
The End.
Where is the ring? A: Mount-Doom
Where is Bilbo now? A: Grey-havens
Where is Frodo now? A: Shire

Fig. 2. An example story with questions correctly answered by a MemNN. The MemNN was trained on the simulation described in Section 4.2 and had never seen many of these words before, e.g. Bilbo, Frodo and Gollum.

End-to-End Memory Network on bAbI tasks [Weston 2015]

Y LeCun

```
Sam walks into the kitchen.
Sam picks up an apple.
Sam walks into the bedroom.
Sam drops the apple.
Q: Where is the apple?
A. Bedroom
```

```
Brian is a lion.
Julius is a lion.
Julius is white.
Bernhard is green.
Q: What color is Brian?
A. White
```

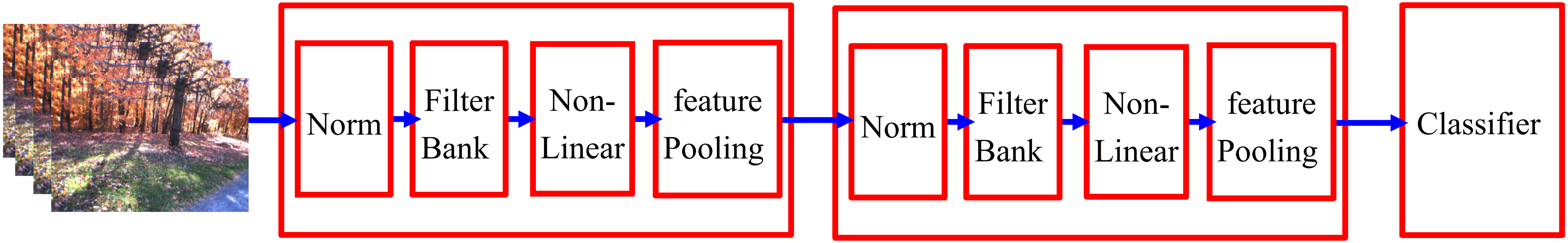
```
Mary journeyed to the den.
Mary went back to the kitchen.
John journeyed to the bedroom.
Mary discarded the milk.
Q: Where was the milk before the den?
A. Hallway
```

	Baseline			MemN2N								
Task	Strongly Supervised MemNN [21]	LSTM [21]	MemNN WSH	BoW	PE	PE LS	PE LS RN	1 hop PE LS joint	2 hops PE LS joint	3 hops PE LS joint	PE LS RN joint	PE LS LW joint
1: 1 supporting fact	0.0	50.0	0.1	0.6	0.1	0.2	0.0	0.8	0.0	0.1	0.0	0.1
2: 2 supporting facts	0.0	80.0	42.8	17.6	21.6	12.8	8.3	62.0	15.6	14.0	11.4	18.8
3: 3 supporting facts	0.0	80.0	76.4	71.0	64.2	58.8	40.3	76.9	31.6	33.1	21.9	31.7
4: 2 argument relations	0.0	39.0	40.3	32.0	3.8	11.6	2.8	22.8	2.2	5.7	13.4	17.5
5: 3 argument relations	2.0	30.0	16.3	18.3	14.1	15.7	13.1	11.0	13.4	14.8	14.4	12.9
6: yes/no questions	0.0	52.0	51.0	8.7	7.9	8.7	7.6	7.2	2.3	3.3	2.8	2.0
7: counting	15.0	51.0	36.1	23.5	21.6	20.3	17.3	15.9	25.4	17.9	18.3	10.1
8: lists/sets	9.0	55.0	37.8	11.4	12.6	12.7	10.0	13.2	11.7	10.1	9.3	6.1
9: simple negation	0.0	36.0	35.9	21.1	23.3	17.0	13.2	5.1	2.0	3.1	1.9	1.5
10: indefinite knowledge	2.0	56.0	68.7	22.8	17.4	18.6	15.1	10.6	5.0	6.6	6.5	2.6
11: basic coreference	0.0	38.0	30.0	4.1	4.3	0.0	0.9	8.4	1.2	0.9	0.3	3.3
12: conjunction	0.0	26.0	10.1	0.3	0.3	0.1	0.2	0.4	0.0	0.3	0.1	0.0
13: compound coreference	0.0	6.0	19.7	10.5	9.9	0.3	0.4	6.3	0.2	1.4	0.2	0.5
14: time reasoning	1.0	73.0	18.3	1.3	1.8	2.0	1.7	36.9	8.1	8.2	6.9	2.0
15: basic deduction	0.0	79.0	64.8	24.3	0.0	0.0	0.0	46.4	0.5	0.0	0.0	1.8
16: basic induction	0.0	77.0	50.5	52.0	52.1	1.6	1.3	47.4	51.3	3.5	2.7	51.0
17: positional reasoning	35.0	49.0	50.9	45.4	50.1	49.0	51.0	44.4	41.2	44.5	40.4	42.6
18: size reasoning	5.0	48.0	51.3	48.1	13.6	10.1	11.1	9.6	10.3	9.2	9.4	9.2
19: path finding	64.0	92.0	100.0	89.7	87.4	85.6	82.8	90.7	89.9	90.2	88.0	90.6
20: agent's motivation	0.0	9.0	3.6	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.2
Mean error (%)	6.7	51.3	40.2	25.1	20.3	16.3	13.9	25.8	15.6	13.3	12.4	15.2
Failed tasks (err. $> 5\%$)	4	20	18	15	13	12	11	17	11	11	11	10
On 10k training data												
Mean error (%)	3.2	36.4	39.2	15.4	9.4	7.2	6.6	24.5	10.9	7.9	7.5	11.0
Failed tasks (err. $> 5\%$)	2	16	17	9	6	4	4	16	7	6	6	6

Non-Convex Objective

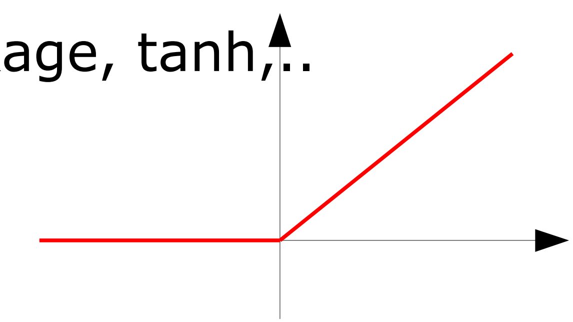
Overall Architecture: multiple stages of Normalization → Filter Bank → Non-Linearity → Pooling

Y LeCun



- **Normalization:** variation on whitening (optional)
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank:** dimension expansion, projection on overcomplete basis
- **Non-Linearity:** sparsification, saturation, lateral inhibition....
  - Rectification (ReLU), Component-wise shrinkage, tanh,..

$$ReLU(x) = max(x, 0)$$



- **Pooling:** aggregation over space or feature type
  - Max, Lp norm, log prob.

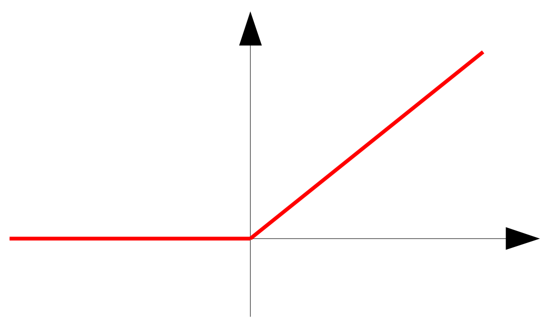
$$MAX: Max_i(X_i);\quad L_p: \sqrt[p]{X_i^p};\quad PROB: \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$$

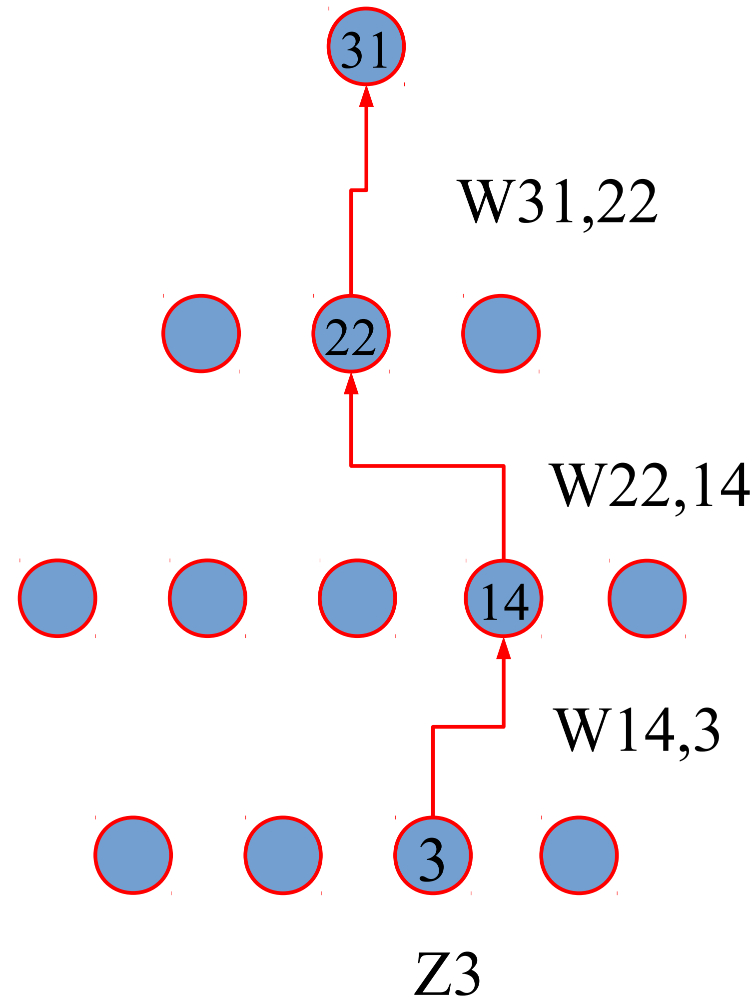
Deep Nets with ReLUs and Max Pooling

Y LeCun

- Stack of linear transforms interspersed with Max operators
- Point-wise ReLUs:

$$ReLU(x) = max(x, 0)$$





- Max Pooling
  - "switches" from one layer to the next

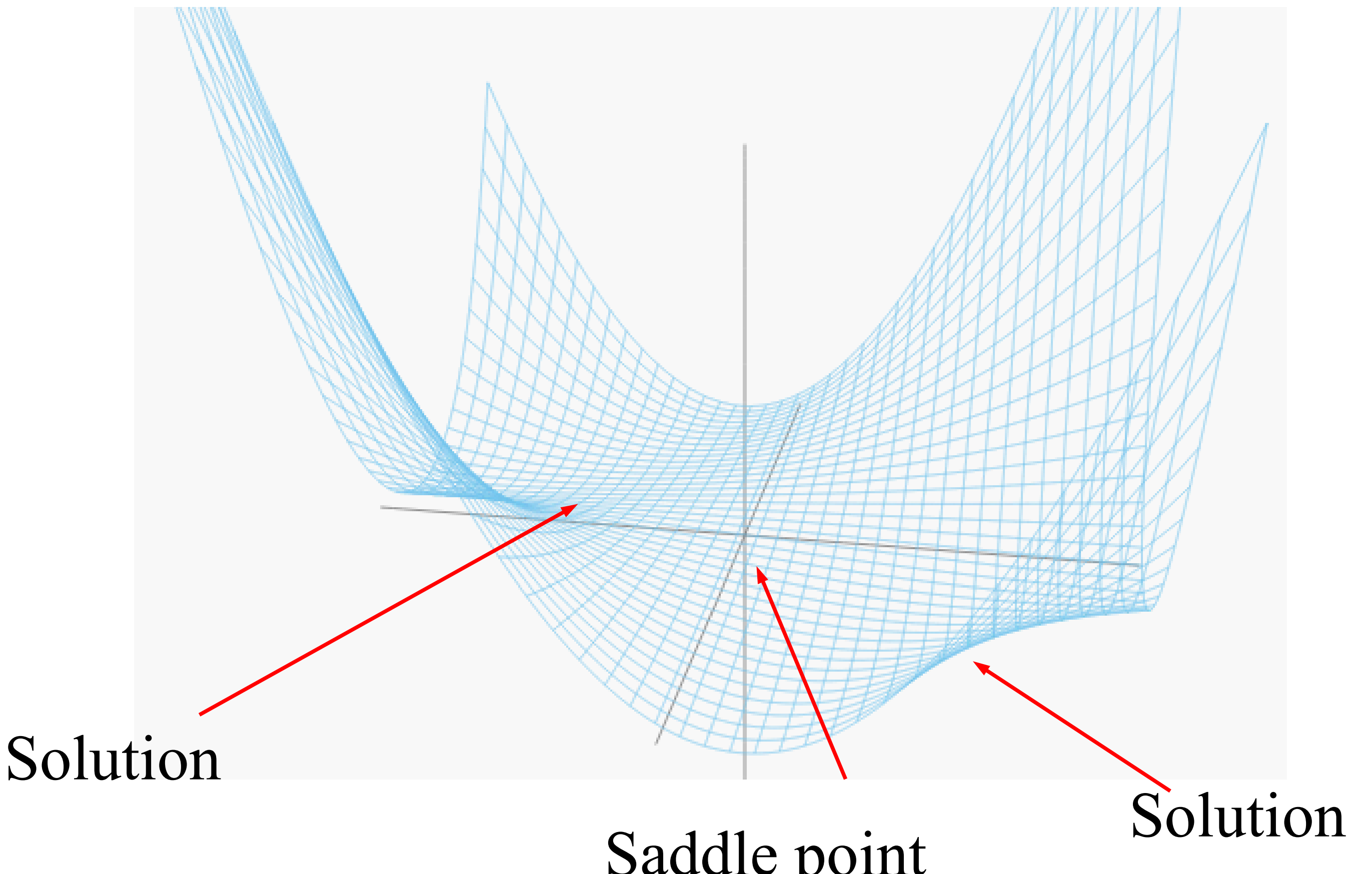
Loss Function for a simple network

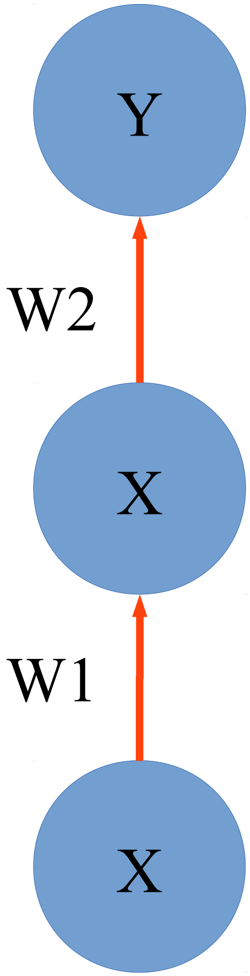
1-1-1 network

– Y = W1*W2*X

trained to compute the identity function with quadratic loss

– Single sample X=1, Y=1 L(W) = (1-W1*W2)^2





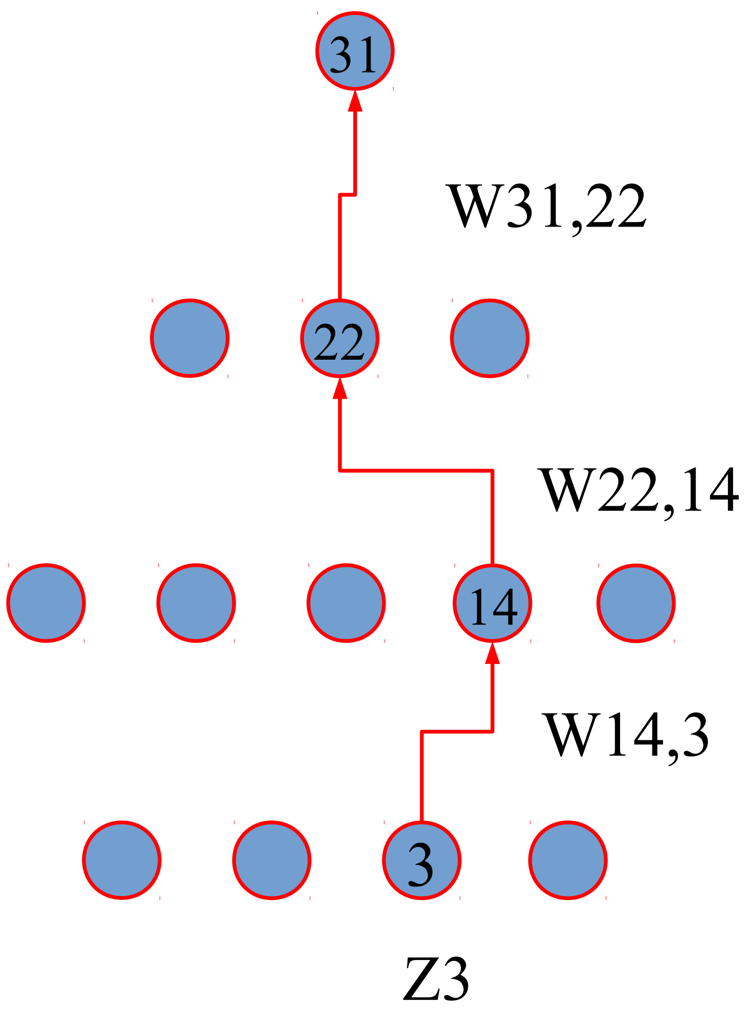
- Single output:

$$\hat{Y} = \sum_P \delta_P(W,X) \Big(\prod_{(ij)\in P} W_{ij} \Big) X_{P_{start}}$$

- Wij: weight from j to i
- P: path in network from input to output
  - P=(3,(14,3),(22,14),(31,22))
- di: 1 if ReLU i is linear, 0 if saturated.
- Xpstart: input unit for path P.

$$\hat{Y} = \sum_P \delta_P(W,X) \Big(\prod_{(ij)\in P} W_{ij} \Big) X_{P_{start}}$$

- Dp(W,X): 1 if path P is "active", 0 if inactive
- Input-output function is piece-wise linear
- Polynomial in W with random coefficients



Deep Convolutional Nets (and other deep neural nets)

- Training sample: (Xi,Yi) k=1 to K
- Objective function (with margin-type loss = ReLU)

$$L(W)=\sum_k ReLU(1-Y^k\sum_P \delta_P(W,X^k)(\prod_{(ij)\in P} W_{ij})X^k_{P_{start}})$$

$$L(W)=\sum_k \sum_P (X^k_{P_{start}} Y^k)\delta_P(W,X^k)(\prod_{(ij)\in P} W_{ij})$$

$$L(W)=\sum_P [\sum_k (X^k_{P_{start}} Y^k)\delta_P(W,X^k)](\prod_{(ij)\in P} W_{ij})$$

$$L(W)=\sum_P C_p(X,Y,W)(\prod_{(ij)\in P} W_{ij})$$

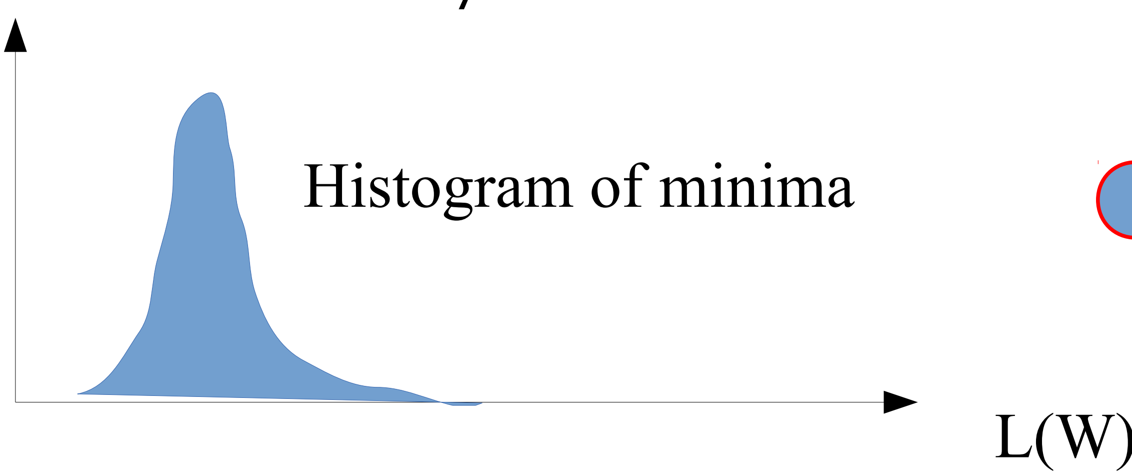
- Polynomial in W of degree l (number of adaptive layers)
- Continuous, piece-wise polynomial with “switched” and partially random coefficients
  - Coefficients are switched in an out depending on W

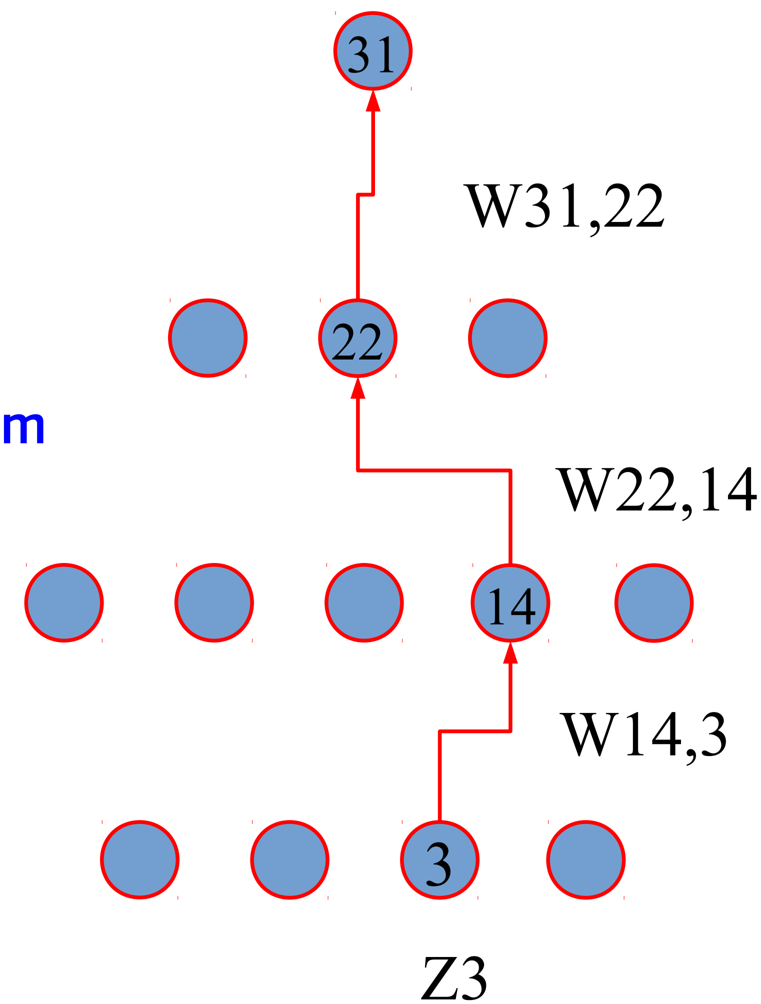
Deep Nets with ReLUs: Objective Function is Piecewise Polynomial

- If we use a hinge loss, delta now depends on label Yk:

$$L(W)=\sum_{P} C_p(X,Y,W)\Big(\prod_{(ij)\in P} W_{ij}\Big)$$

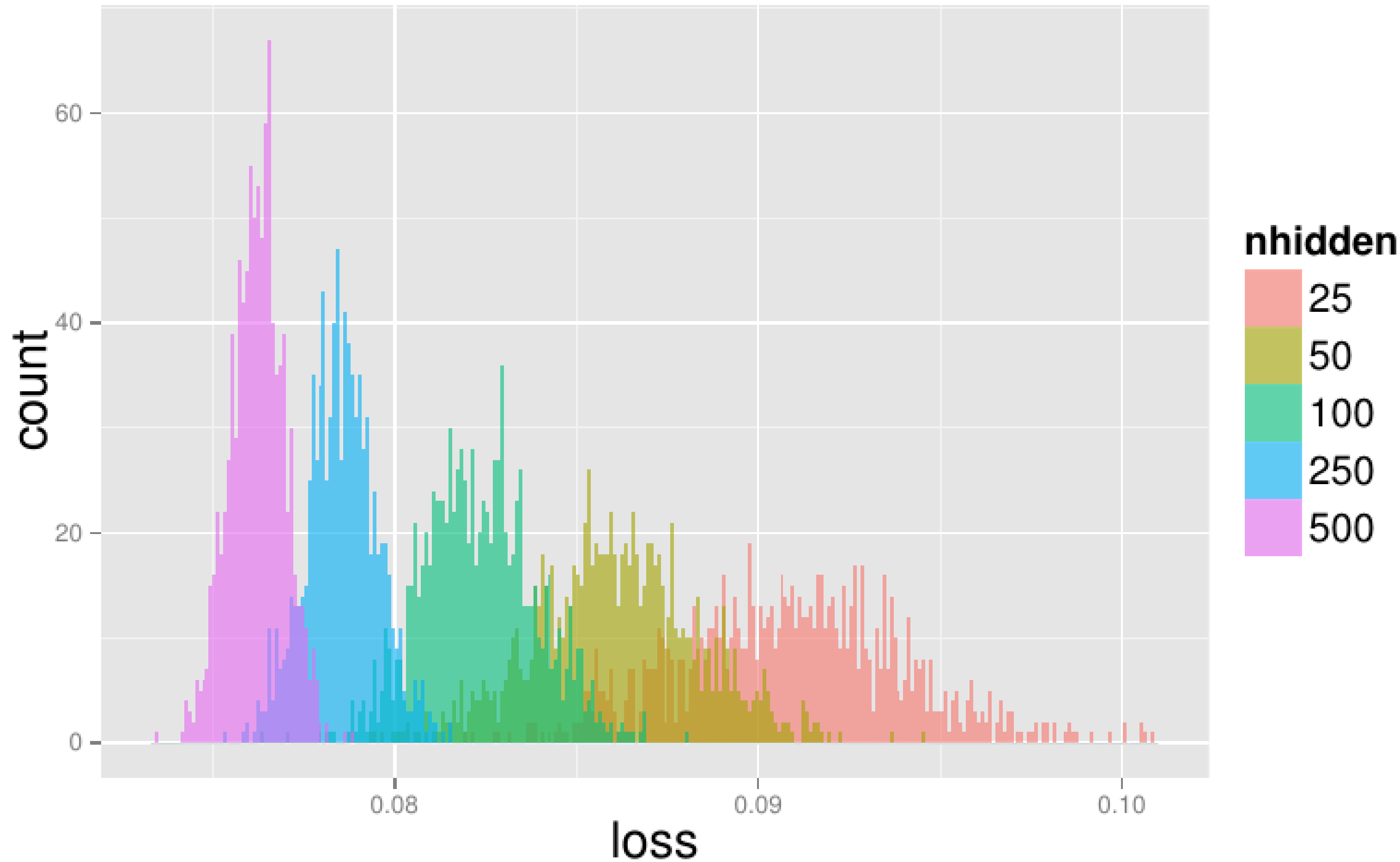
- Piecewise polynomial in W with random coefficients
- A lot is known about the distribution of critical points of polynomials on the sphere with random (Gaussian) coefficients [Ben Arous et al.]
  - High-order spherical spin glasses
  - Random matrix theory





Deep Nets with ReLUs: Objective Function is Piecewise Polynomial

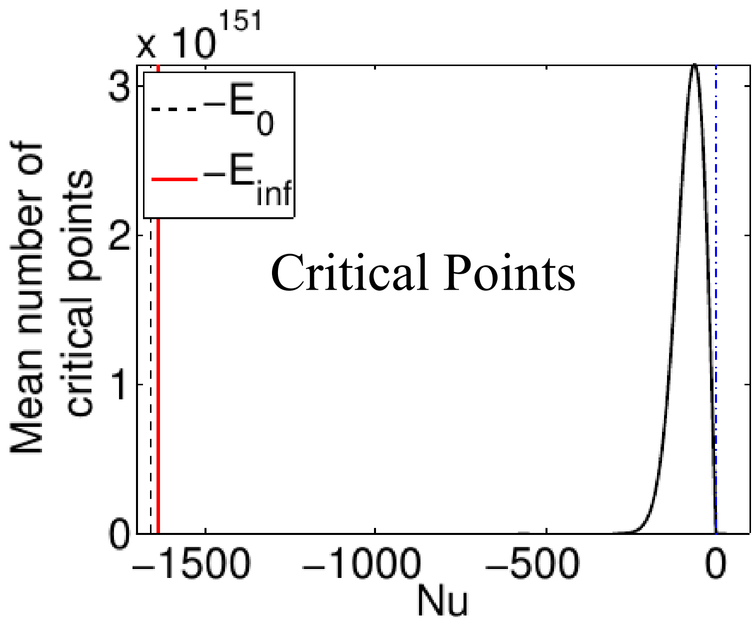
- Train 2-layer nets on scaled-down MNIST (10x10) from multiple initial conditions. Measure loss on test set.

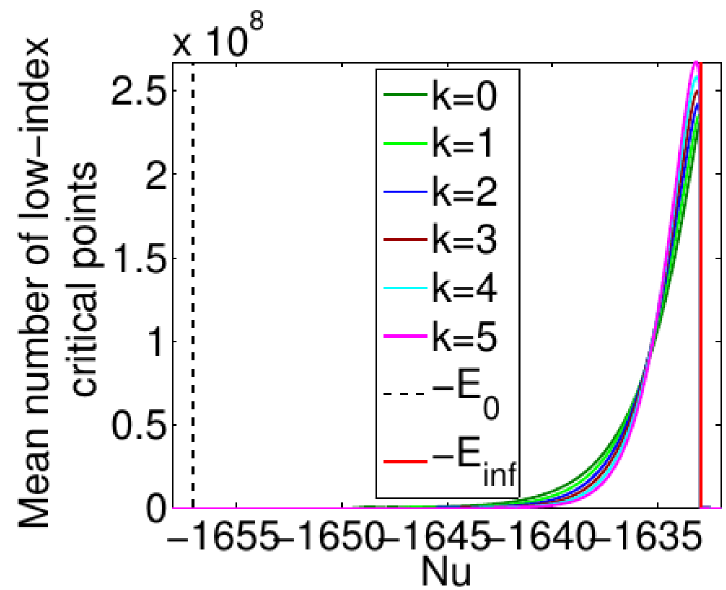


[Choromanska, Henaff, Mathieu, Ben Arous, LeCun 2015]

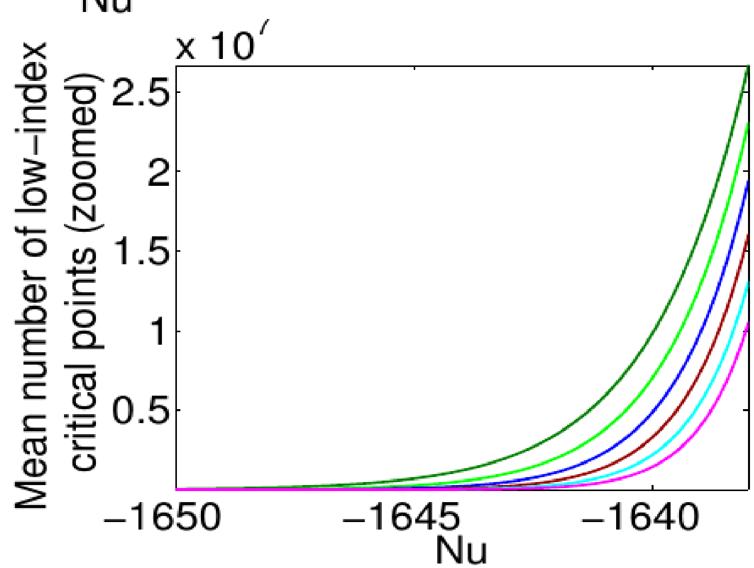
Distribution of critical points (saddle points, minima, maxima)

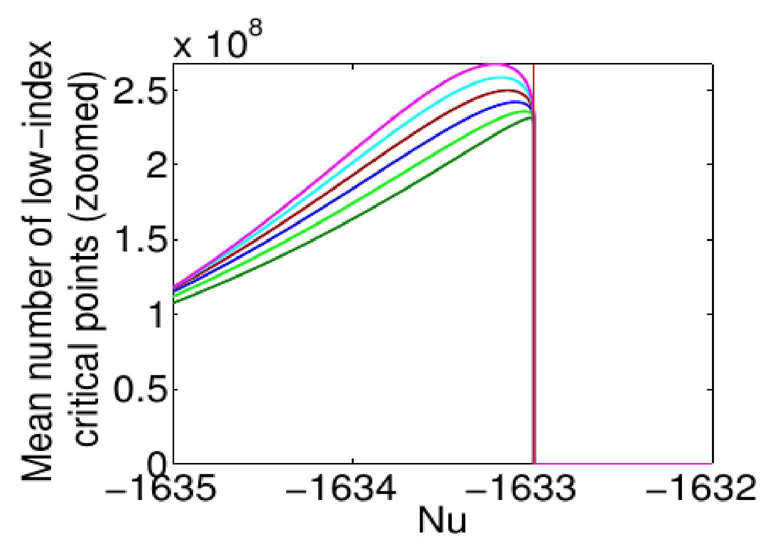
- K=number of negative eigenvalues of Hessian (K=0 → minimum)





Zoomed:





Elastic Average SGD

Distributing SGD over multiple CPU/GPU nodes

Deep Learning withElastic Average SGD: [Zhang, Choromanska, LeCun arXiv:1412.6651]

- Expected loss

$$\min_{x} F(x) := \mathbb{E}[f(x,\xi)],$$

- Distributed form

$$\min_{x^1,\dots,x^p,\tilde{x}} \sum_{i=1}^{p} \mathbb{E}[f(x^i,\xi^i)] + \frac{\rho}{2}\|x^i - \tilde{x}\|^2$$

- Update formulas

$$\begin{aligned} x^i_{t+1} &= x^i_t - \eta(g^i_t(x^i_t) + \rho(x^i_t - \tilde{x}_t)) \\ \tilde{x}_{t+1} &= \tilde{x}_t + \eta\sum_{i=1}^{p}\rho(x^i_t - \tilde{x}_t), \end{aligned}$$

- Reparameterization:

$\alpha = \eta\rho$ and $\beta = p\alpha$

$$\begin{aligned} x^i_{t+1} &= x^i_t - \eta g^i_t(x^i_t) - \alpha(x^i_t - \tilde{x}_t) \\ \tilde{x}_{t+1} &= (1-\beta)\tilde{x}_t + \beta\left(\frac{1}{p}\sum_{i=1}^{p} x^i_t\right) \end{aligned}$$

Elastic Average SGD & Elastic Average Momentum SGD

- Asynchronous algorithms. Sync between node every Tau updates.
  - Every Tau steps: move workers toward center, and vice versa
  - Momentum form uses Nesterov accelerated gradient

Algorithm 1: Asynchronous EASGD: Processing by worker i and the master

Input: learning rate η, moving rate α, communication period $\tau \in \mathbb{N}$
Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $t^i = 0$

Repeat
$x \leftarrow x^i$
if (τ divides t^i) **then**
a) $x^i \leftarrow x^i - \alpha(x - \tilde{x})$
b) $\tilde{x} \leftarrow \tilde{x} + \alpha(x - \tilde{x})$
end
$x^i \leftarrow x^i - \eta g^i_{t^i}(x)$
$t^i \leftarrow t^i + 1$
Until forever

Algorithm 2: Asynchronous EAMSGD: Processing by worker i and the master

Input: learning rate η, moving rate α, communication period $\tau \in \mathbb{N}$, momentum term δ
Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $v^i = 0$, $t^i = 0$

Repeat
$x \leftarrow x^i$
if (τ divides t^i) **then**
a) $x^i \leftarrow x^i - \alpha(x - \tilde{x})$
b) $\tilde{x} \leftarrow \tilde{x} + \alpha(x - \tilde{x})$
end
$v^i \leftarrow \delta v^i - \eta g^i_{t^i}(x + \delta v^i)$
$x^i \leftarrow x^i + v^i$
$t^i \leftarrow t^i + 1$
Until forever

Downpour algorithm: send gradient, receive parameter vector

- **Asynchronous algorithm. Sync between node every Tau updates.**
  - Every Tau steps:
    - node sends accumulated gradient to server
    - Server sends updated parameter vector to node.
  - Momentum form uses Nesterov accelerated gradient

Algorithm 3: DOWNPOUR: Processing by worker i and the master

Input: learning rate η, communication period $\tau \in \mathbb{N}$

Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $v^i = 0$, $t^i = 0$

Repeat

    if (τ divides t^i) **then**

        $\tilde{x} \leftarrow \tilde{x} + v^i$

        $x^i \leftarrow \tilde{x}$

        $v^i \leftarrow 0$

    end

    $x^i \leftarrow x^i - \eta g^i_{t^i}(x^i)$

    $v^i \leftarrow v^i - \eta g^i_{t^i}(x^i)$

    $t^i \leftarrow t^i + 1$

Until forever

Like ADMM without the constraint term

- **But ADMM is unstable under a round robin scheme for synchronization**

$$\max_{\lambda^1,\ldots,\lambda^p} \min_{x^1,\ldots,x^p,\tilde{x}} \sum_{i=1}^{p} F(x^i) - \lambda^i (x^i - \tilde{x}) + \frac{\rho}{2} \|x^i - \tilde{x}\|^2$$

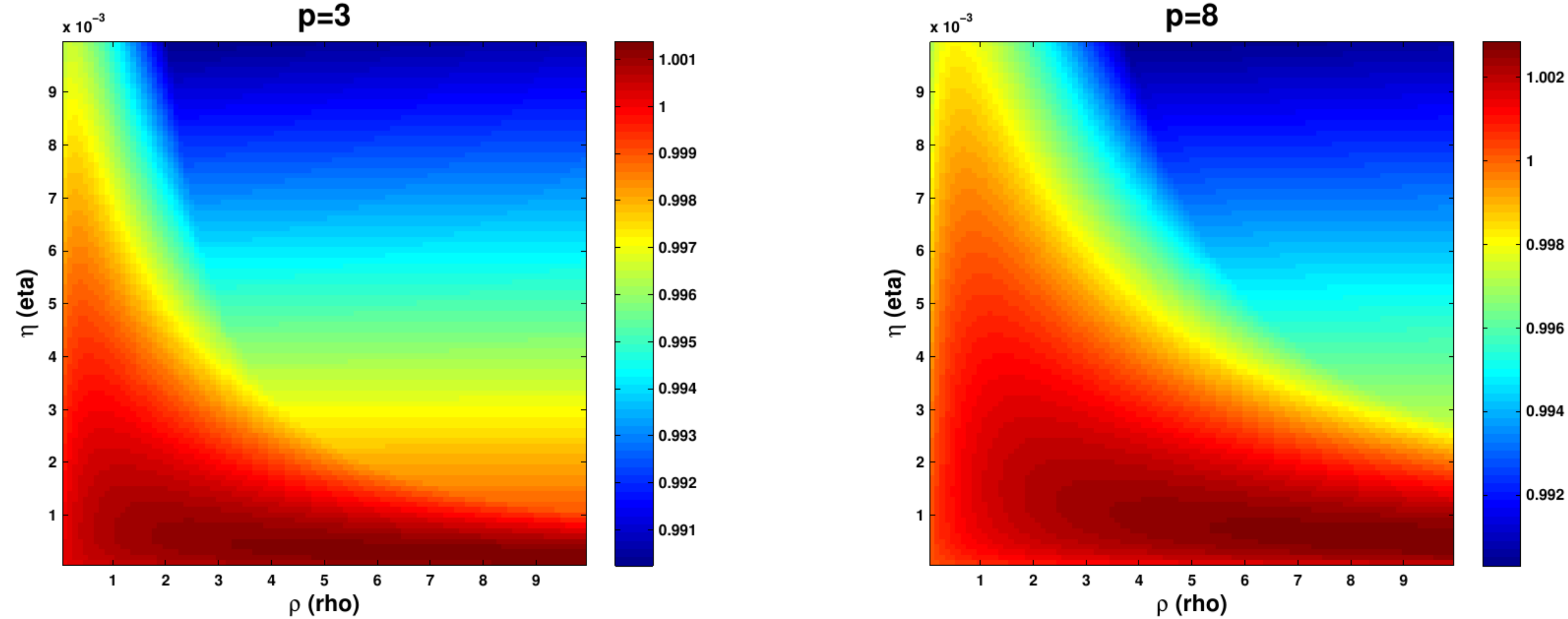
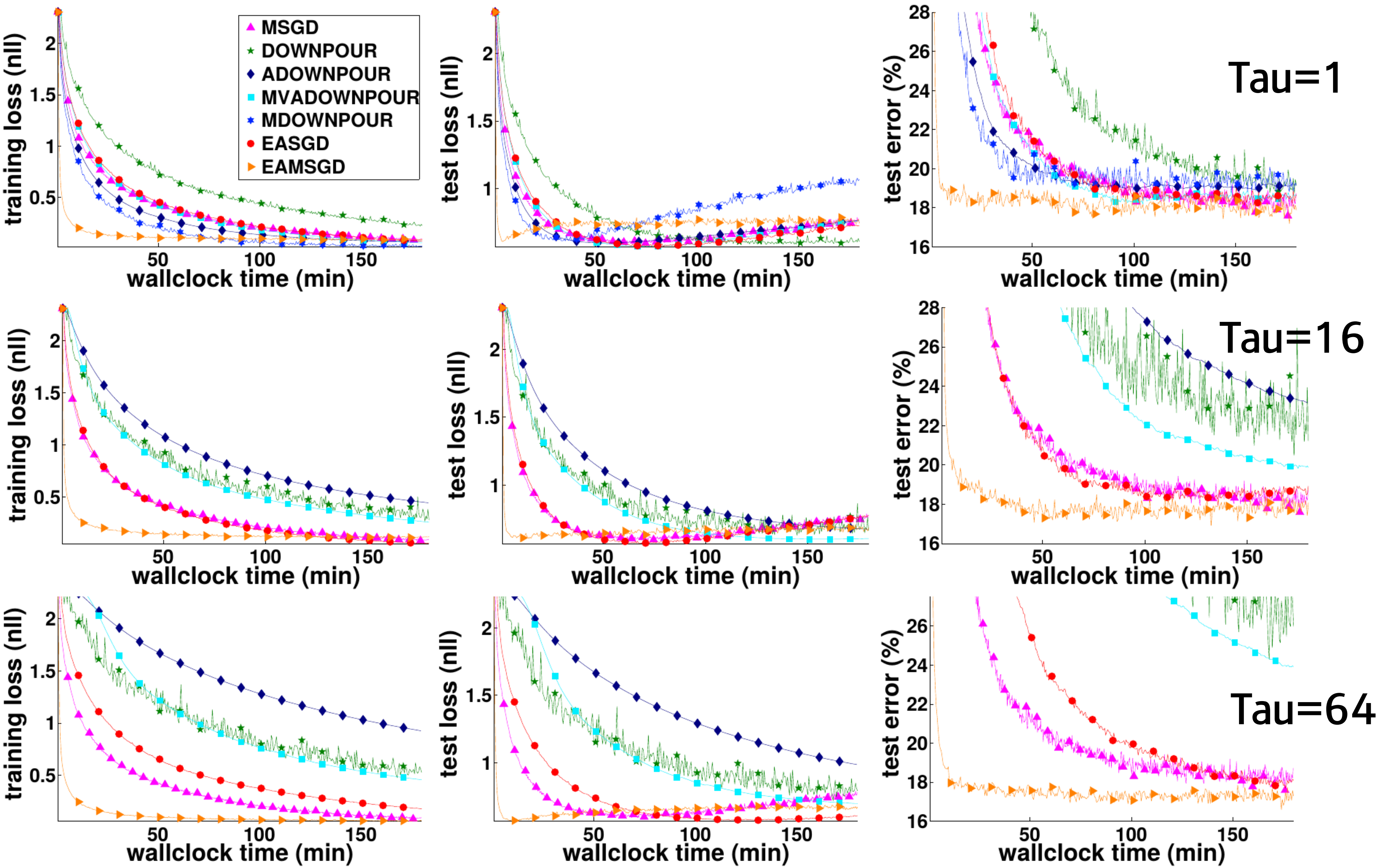


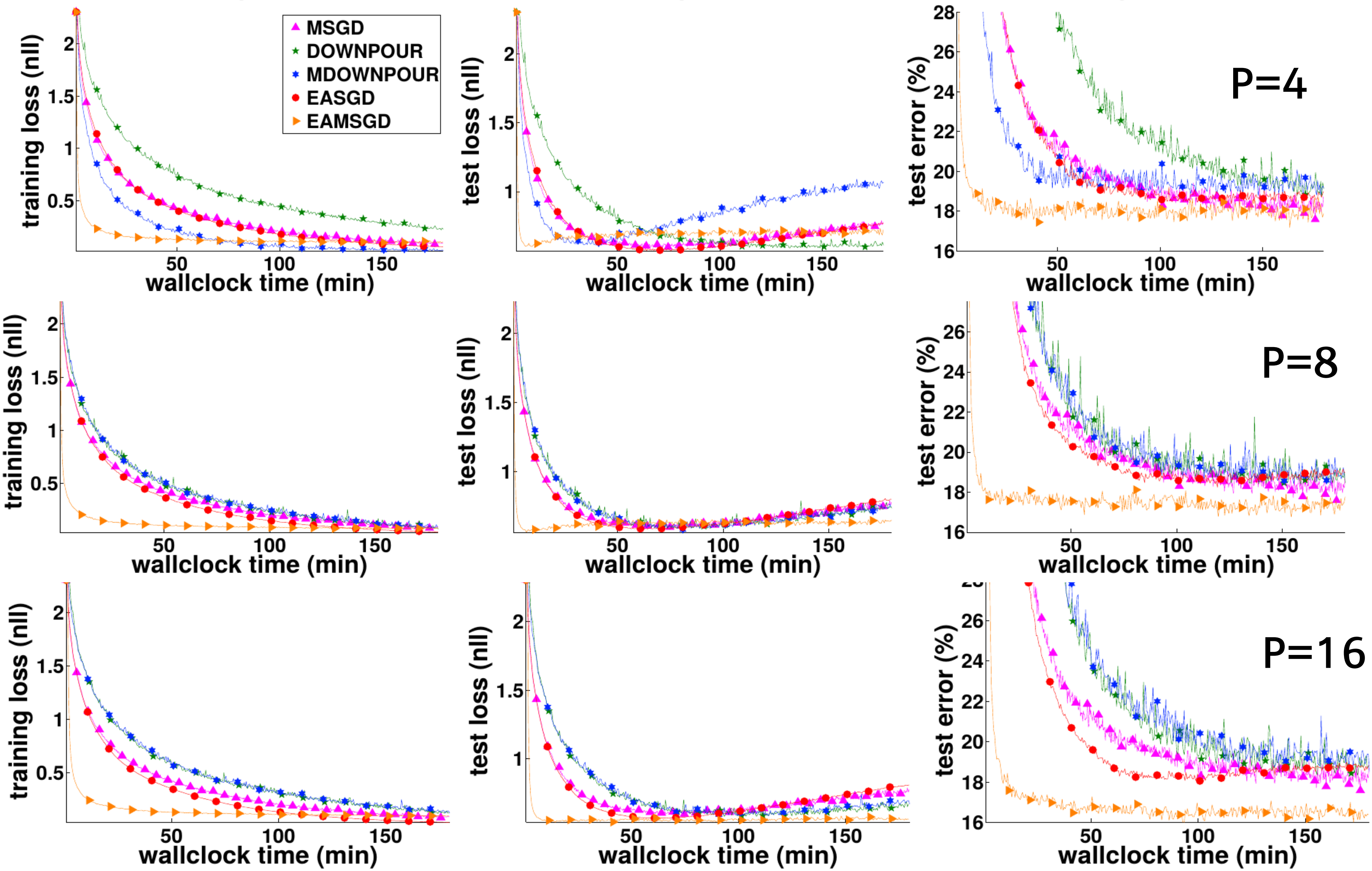
Figure 1: The largest absolute eigenvalue of the linear map $\mathcal{F} = F_3^p \circ F_2^p \circ F_1^p \circ \ldots \circ F_3^1 \circ F_2^1 \circ F_1^1$ as a function of $\eta \in (0, 10^{-2})$ and $\rho \in (0, 10)$ when $p = 3$ and $p = 8$. To simulate the chaotic behavior of the *ADMM* algorithm, one may pick $\eta = 0.001$ and $\rho = 2.5$ and initialize the state s_0 either randomly or with $\lambda_0^1 = 0, x_0^1 = 1000, \lambda_0^2 = 0, x_0^2 = 1000, \lambda_0^3 = 0, x_0^3 = 1000, \tilde{x}_0 = 1000$. *Figure should be read in color.*

Results on CIFAR-10 dataset , 7-layer ConvNet, 4 nodes

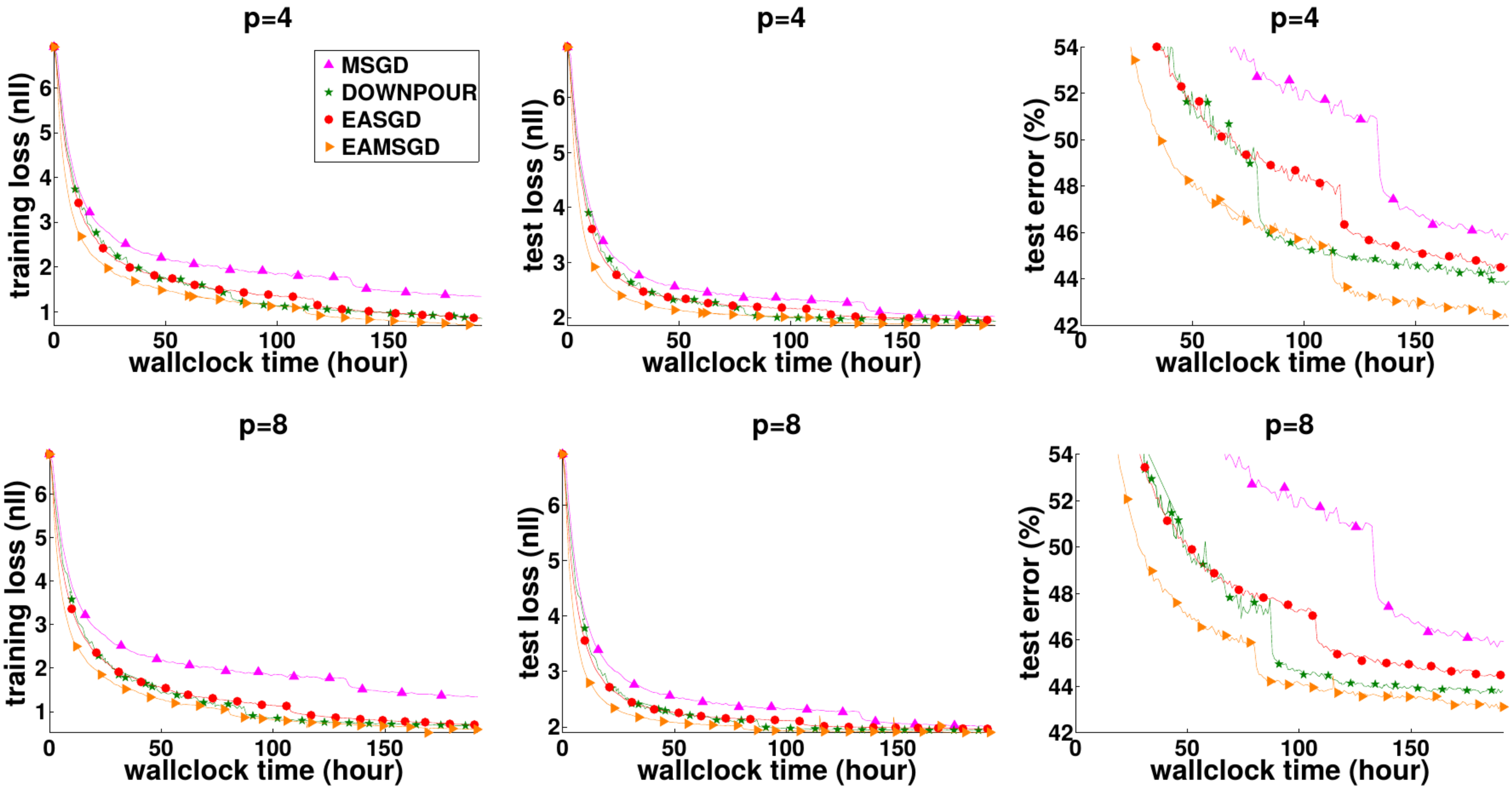
Y LeCun



Results: CIFAR-10, 7-layer ConvNet, Tau=10 (Tau=1 for Downpour)



Results: ImageNet, ConvNet, Tau=10 (Tau=1 for Downpour)



Results: Performance comparison on CIFAR-10 and ImageNet

Y LeCun

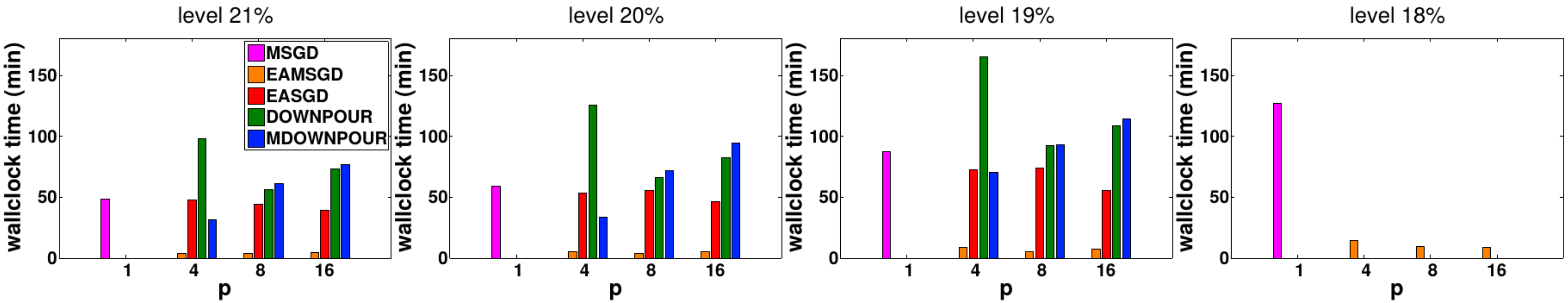


Figure 10: The wall clock time needed to achieve the same level of the test error thr as a function of the number of local workers p on the *CIFAR* dataset. From left to right: $thr = \{21\%, 20\%, 19\%, 18\%\}$. Missing bars denote that the method never achieved specified level of test error.

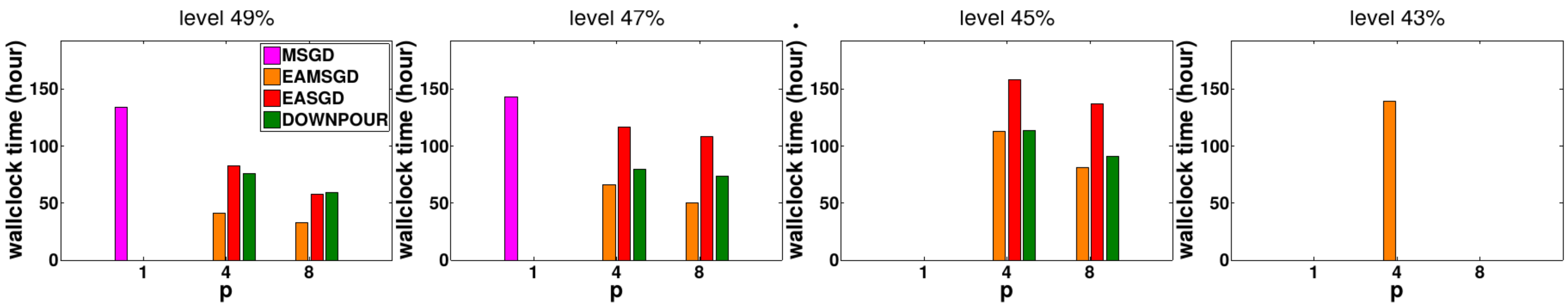


Figure 11: The wall clock time needed to achieve the same level of the test error thr as a function of the number of local workers p on the *ImageNet* dataset. From left to right: $thr = \{49\%, 47\%, 45\%, 43\%\}$. Missing bars denote that the method never achieved specified level of test error.

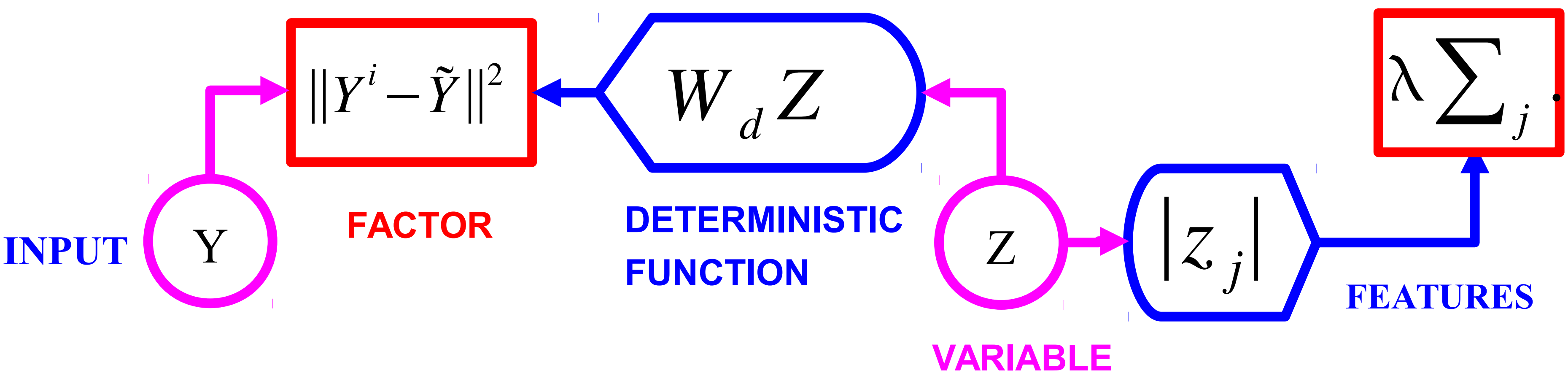
How Learning Can Help With Optimization: LISTA

Sparse Modeling: Sparse Coding + Dictionary Learning

[Olshausen & Field 1997]

- Sparse linear reconstruction
- Energy = reconstruction_error + code_prediction_error + code_sparsity

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

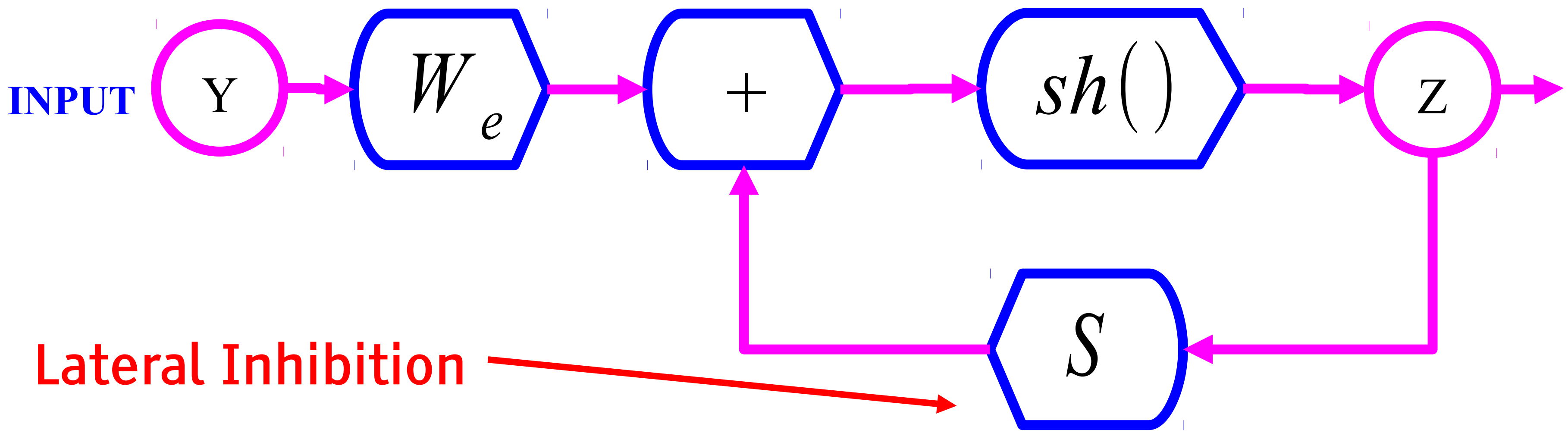


- Inference is expensive: ISTA/FISTA, CGIHT, coordinate descent....

$$Y \rightarrow \hat{Z} = argmin_Z E(Y, Z)$$

Better Idea: Give the "right" structure to the encoder

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



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

- ISTA/FISTA reparameterized:

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

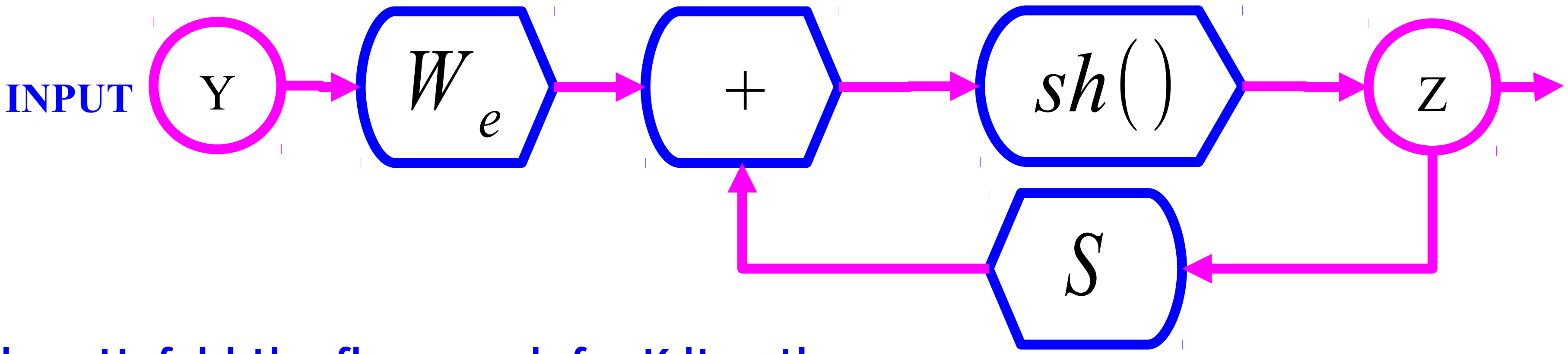
- LISTA (Learned ISTA): learn the We and S matrices to get fast solutions

[Gregor & LeCun, ICML 2010], [Bronstein et al. ICML 2012], [Rolfe & LeCun ICLR 2013]

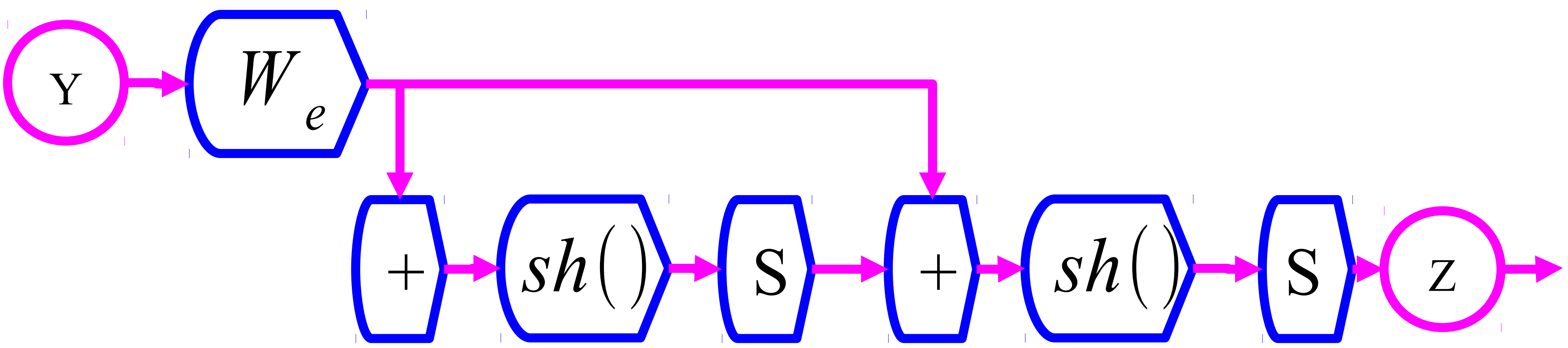
LISTA: Train the We and S matrices to give a good approximation quickly

Y LeCun

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

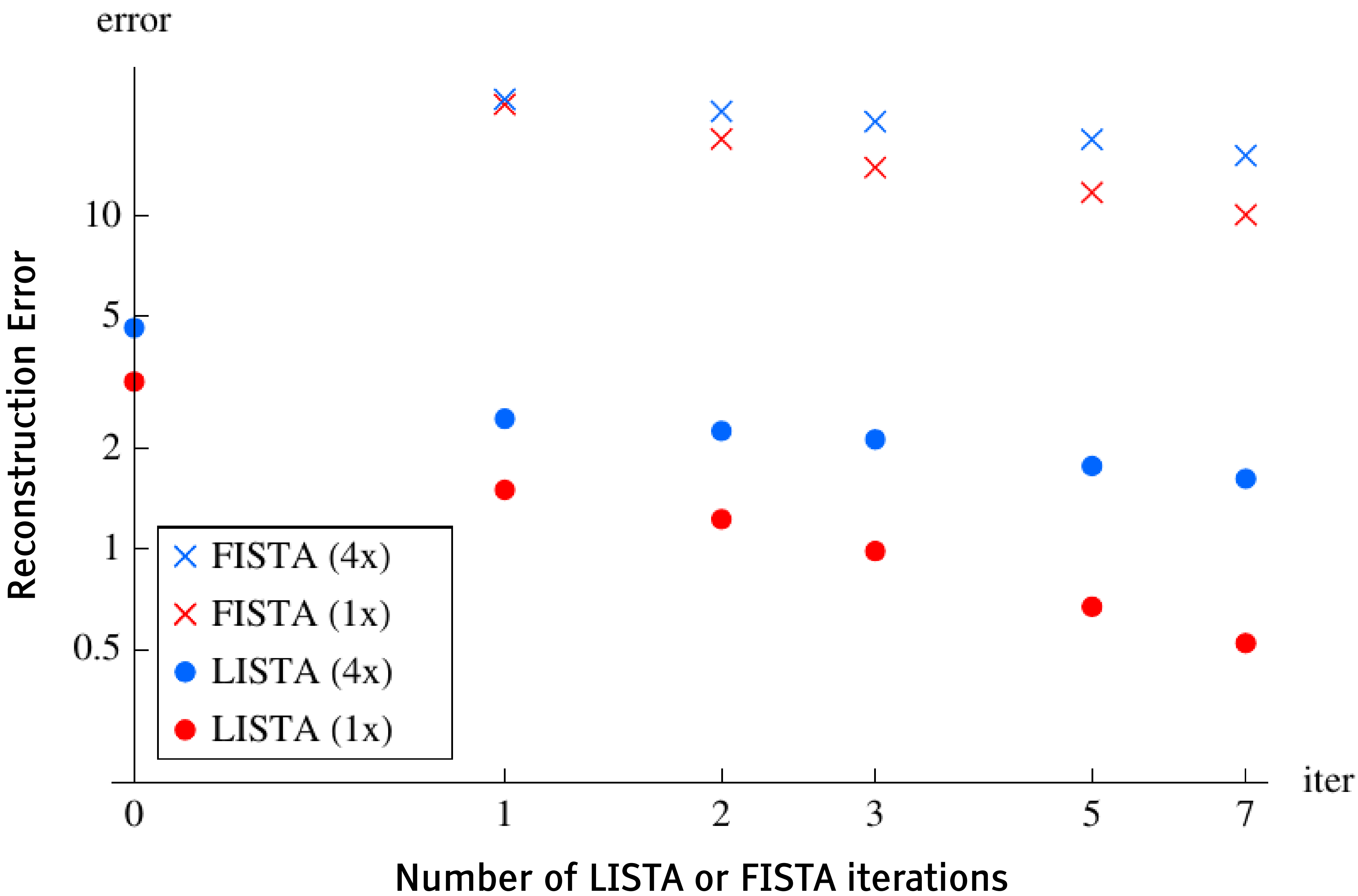


- Time-Unfold the flow graph for K iterations
- Learn the We and S matrices with “backprop-through-time”
- Get the best approximate solution within K iterations



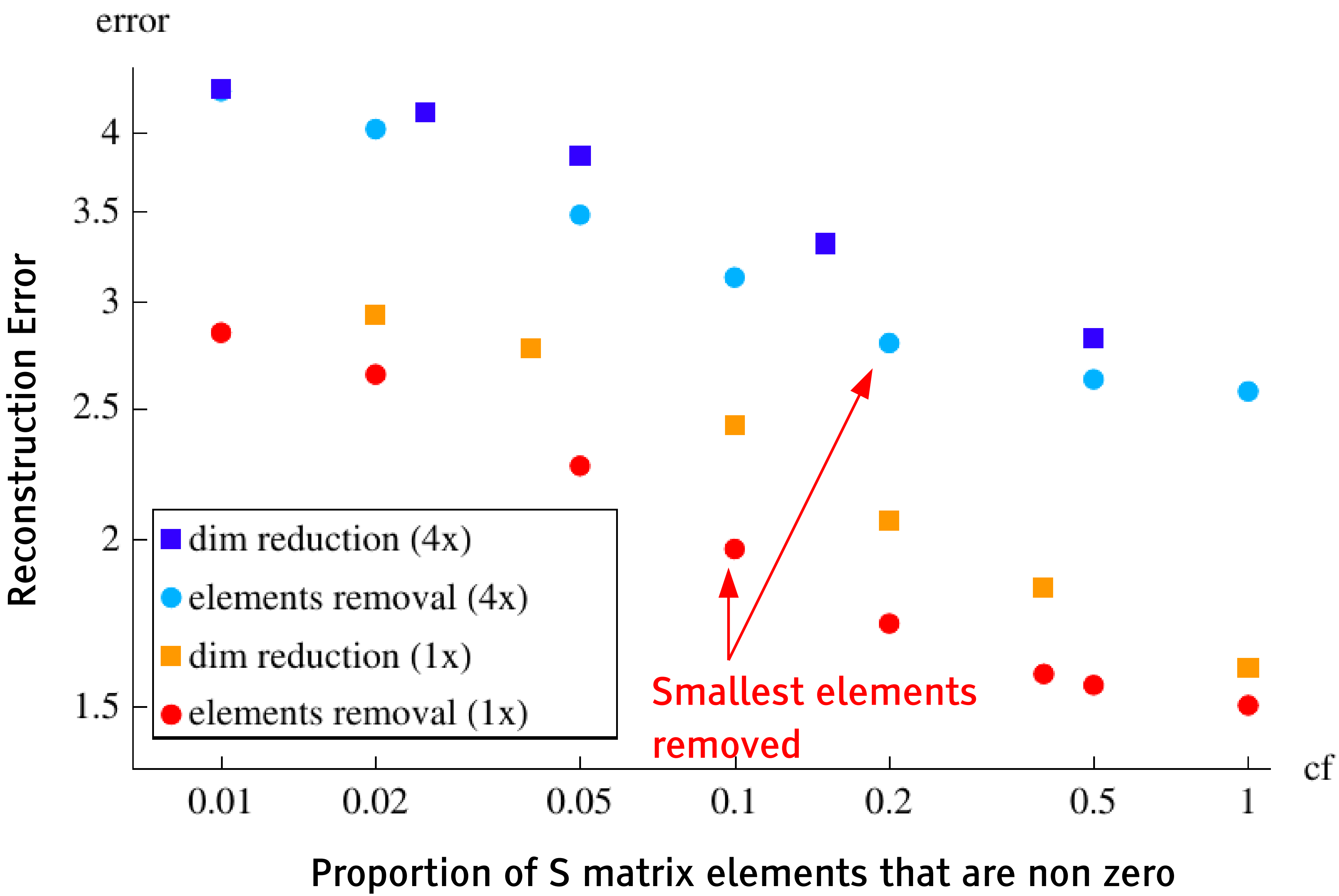
Learning ISTA (LISTA) vs ISTA/FISTA

Y LeCun



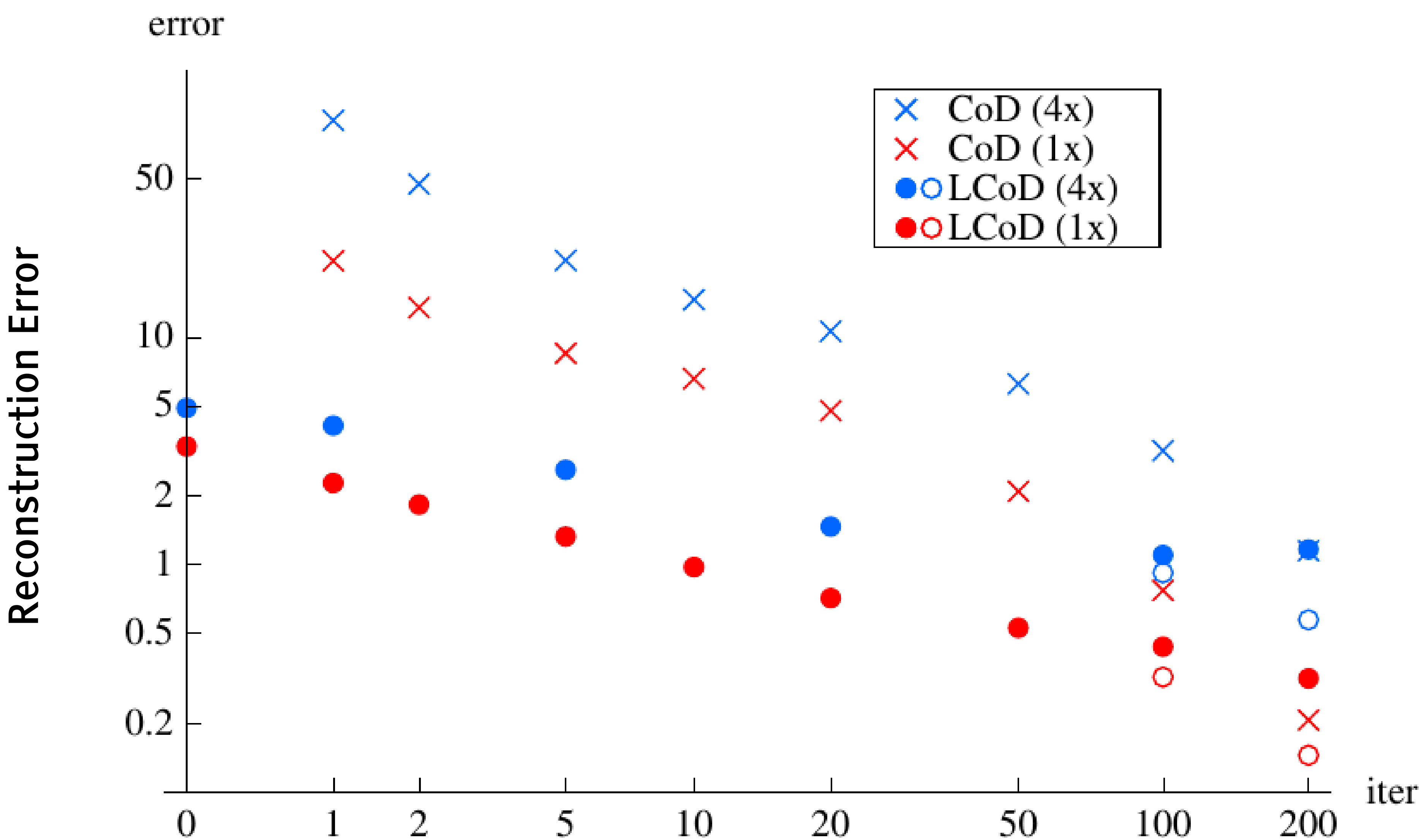
LISTA with partial mutual inhibition matrix

Y LeCun



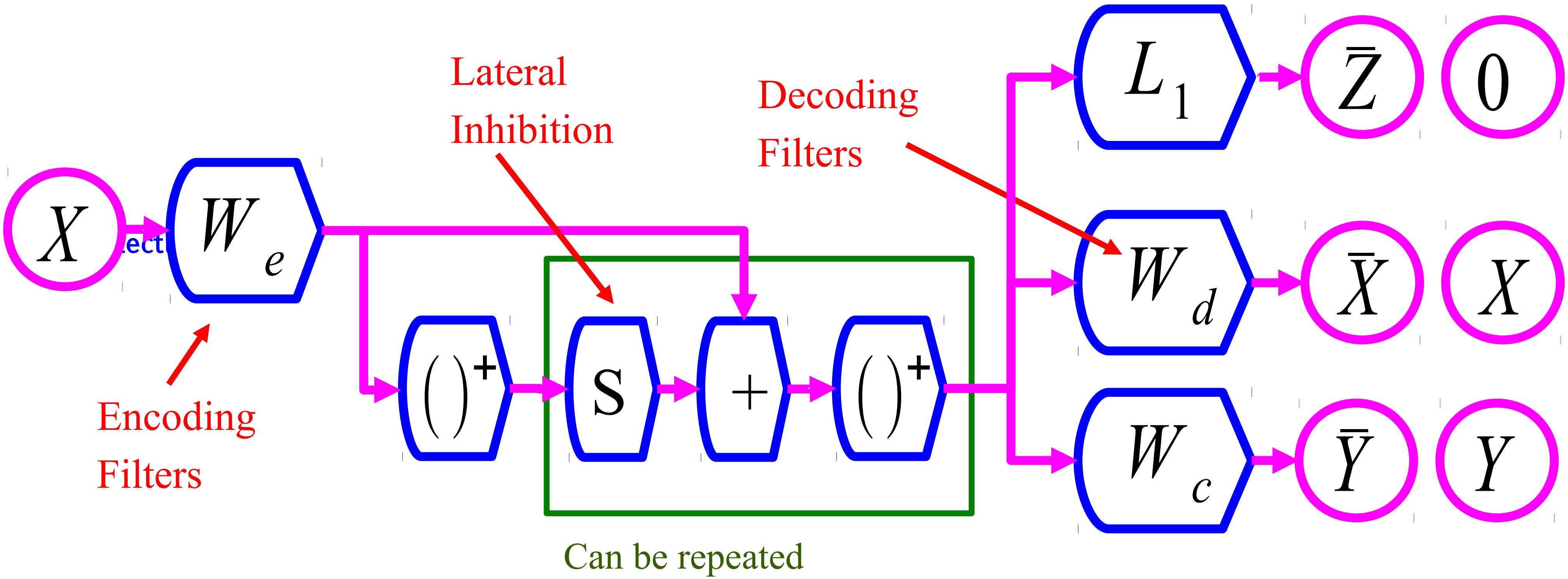
Learning Coordinate Descent (LcoD): faster than LISTA

Y LeCun



Number of LISTA or FISTA iterations

Discriminative Recurrent Sparse Auto-Encoder (DrSAE)

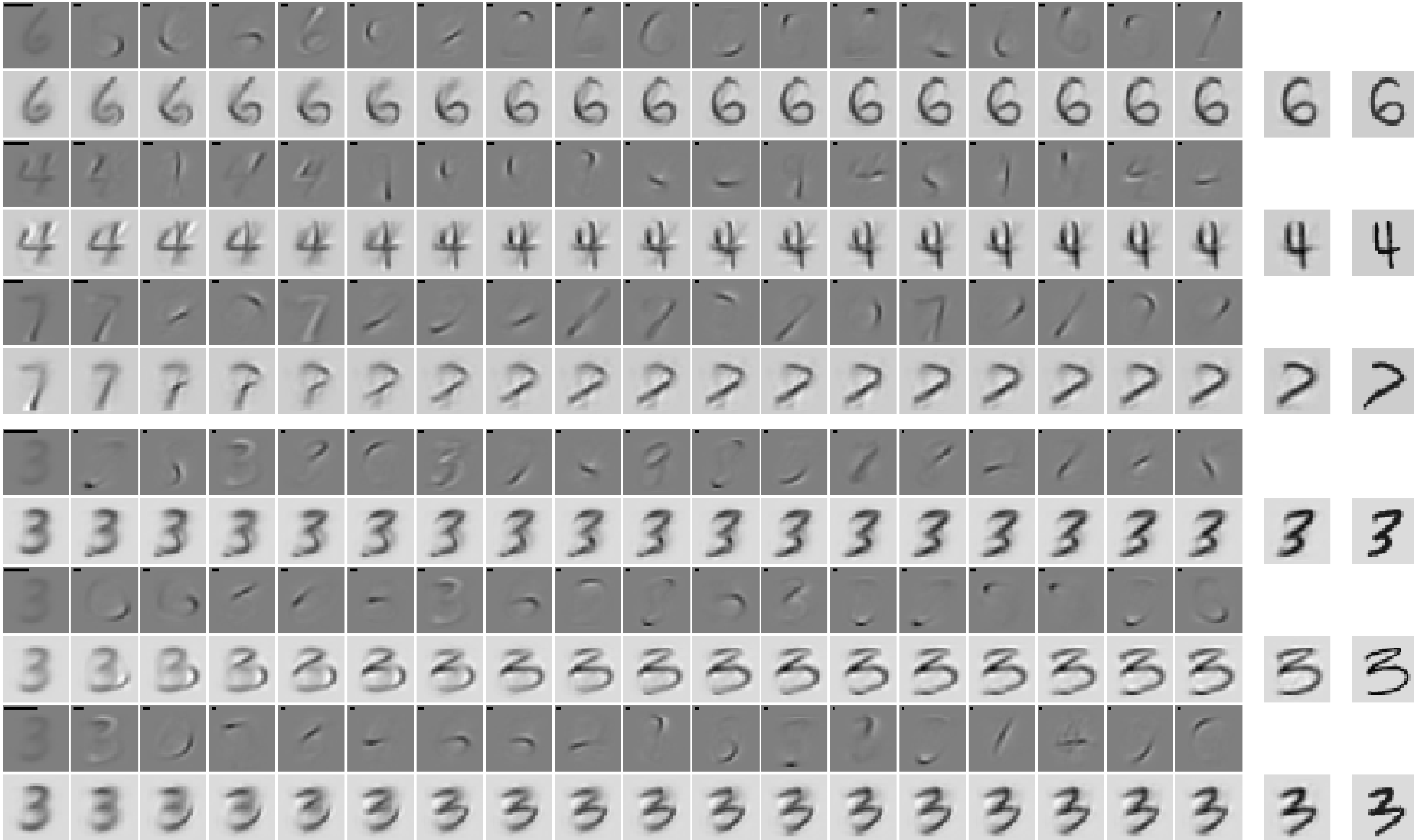


[Rolfe & LeCun ICLR 2013]

- Rectified linear units
- Classification loss: cross-entropy
- Reconstruction loss: squared error
- Sparsity penalty: L1 norm of last hidden layer
- Rows of Wd and columns of We constrained in unit sphere

DrSAE Discovers manifold structure of handwritten digits

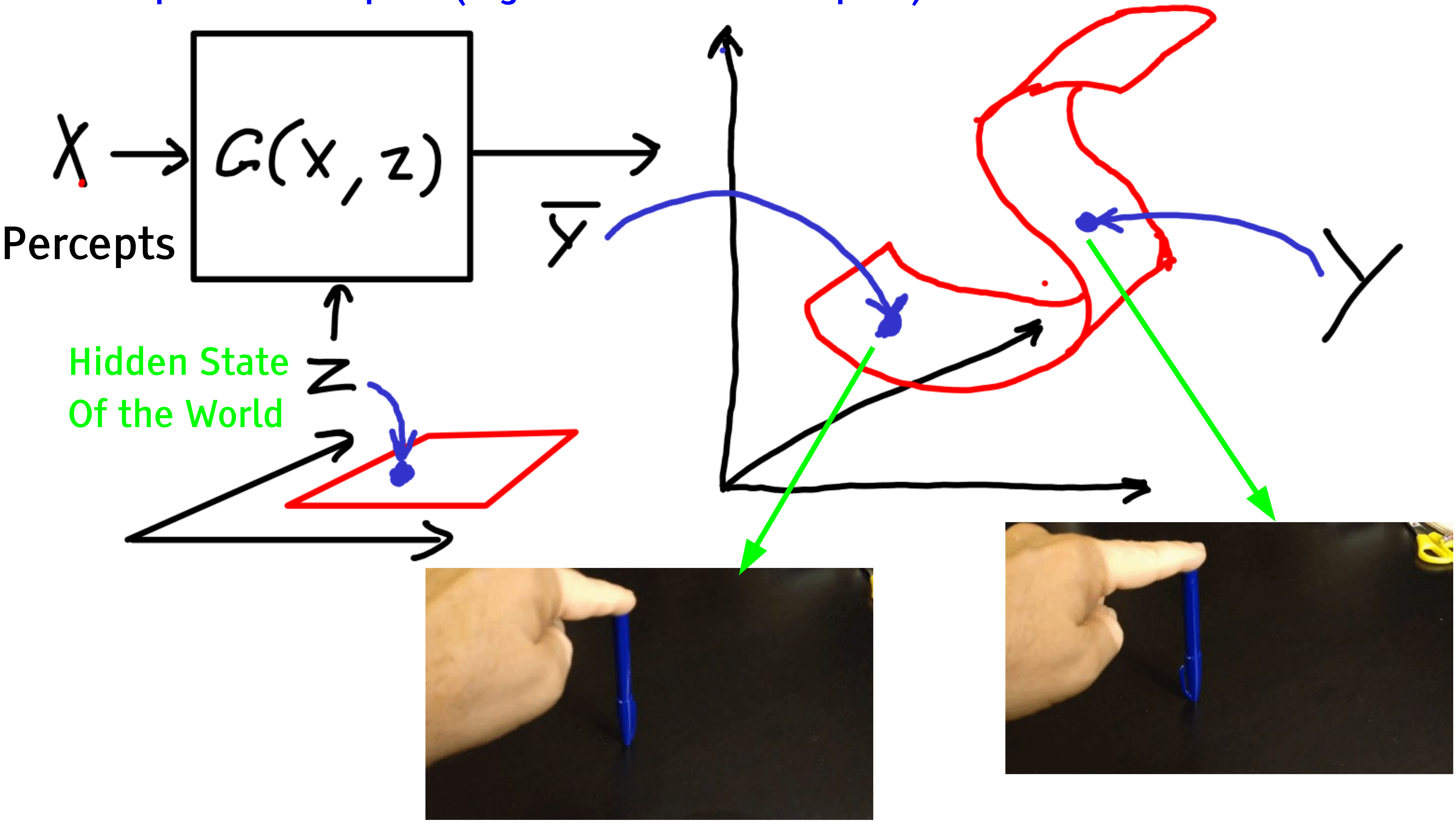
- Image = prototype + sparse sum of “parts” (to move around the manifold)



Learning in the Presence of Uncertainty: Adversarial Training

The Hard Part: Prediction Under Uncertainty

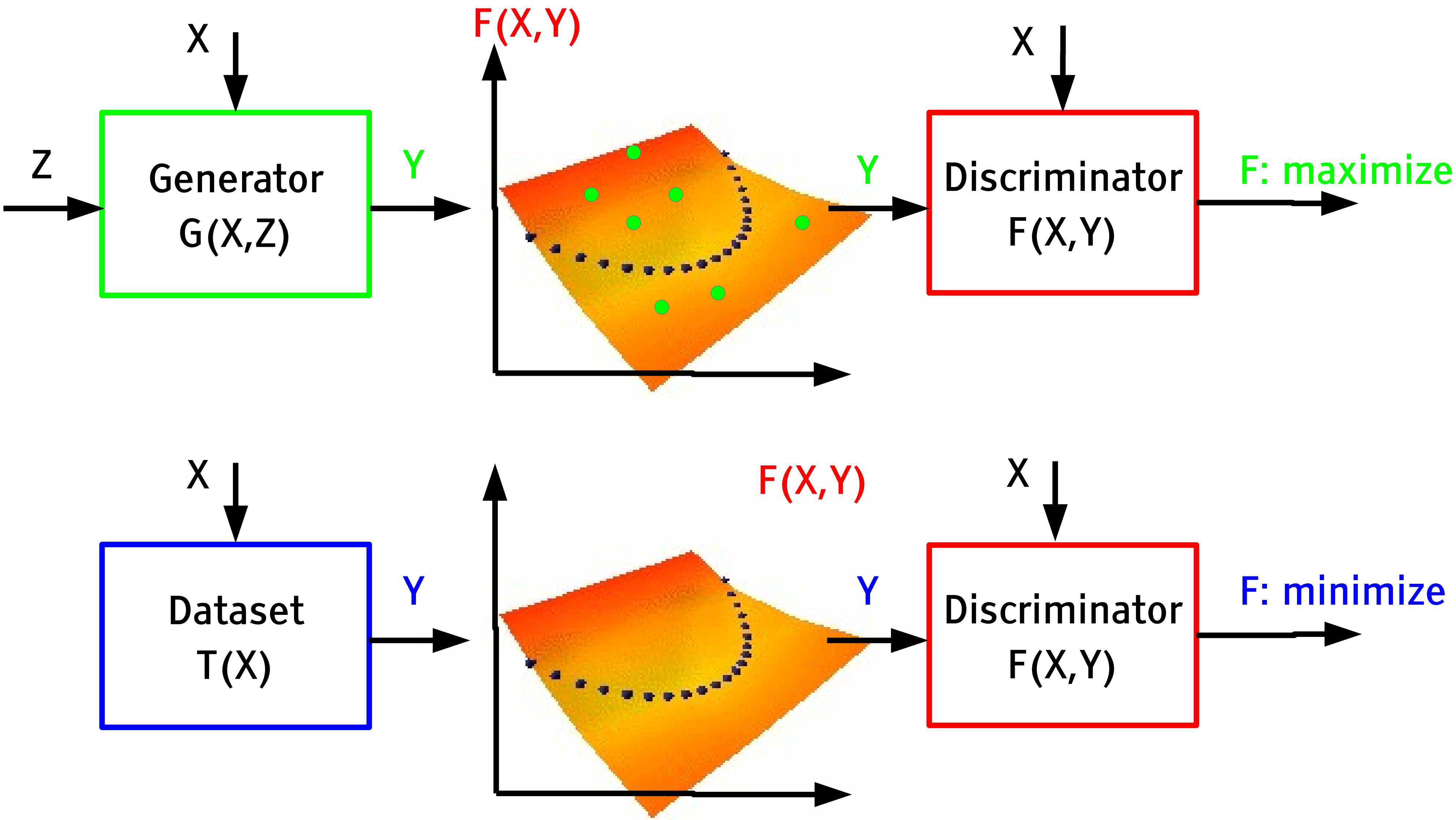
Invariant prediction: The training samples are merely representatives of a whole set of possible outputs (e.g. a manifold of outputs).



Adversarial Training: A Trainable Objective Function

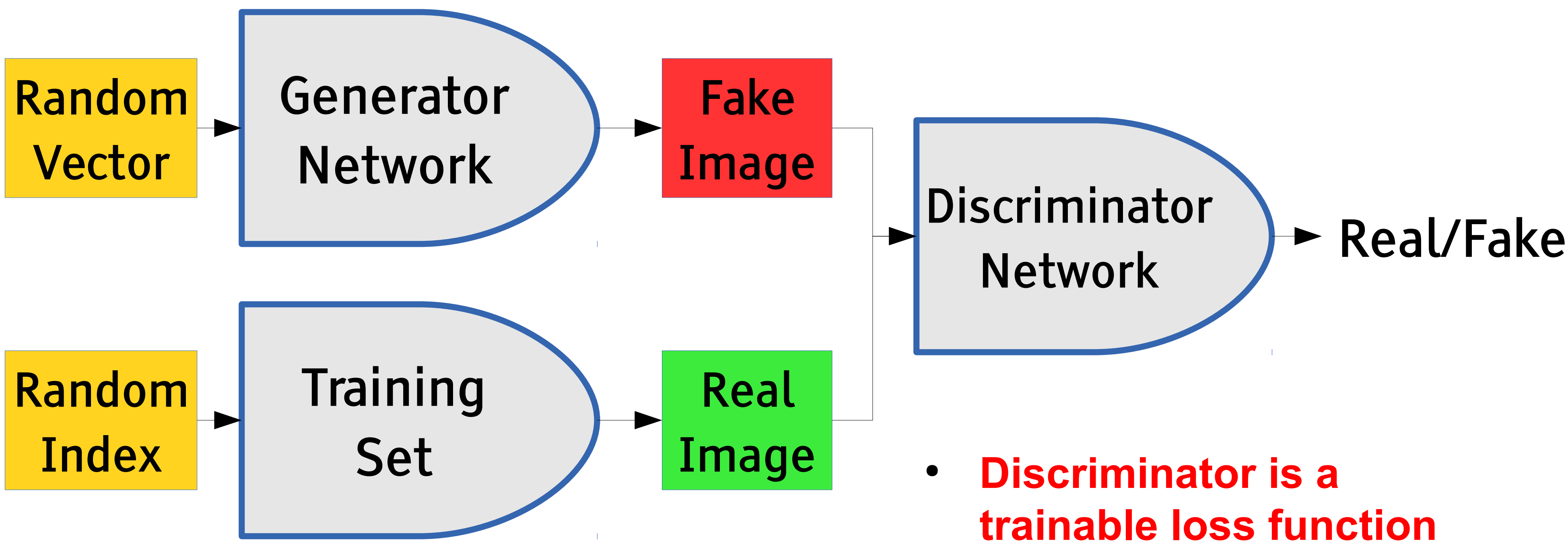
Adversarial Training [Goodfellow et al. NIPS 2014]

Energy-based view of adversarial training: generator picks points to push up



Generative Adversarial Networks

- [Goodfellow et al. NIPS 2014]
- Generator net maps random numbers to image
- Discriminator learns to tell real from fake images.
- Generator can cheat: it knows the gradient of the output of the discriminator with respect to its input



- Discriminator is a trainable loss function

DCGAN: adversarial training to generate images.

[Radford, Metz, Chintala 2015]

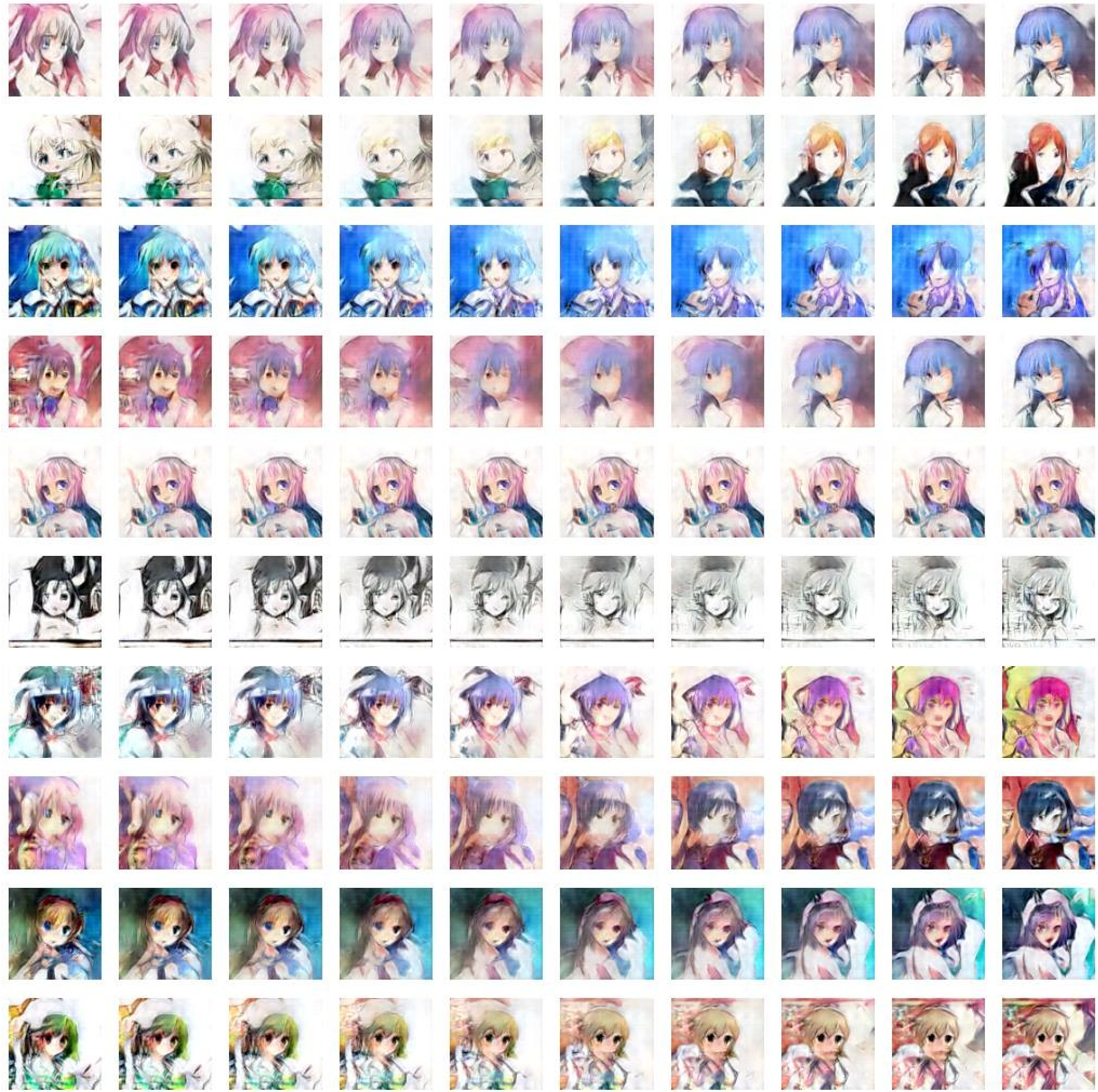
– Input: random numbers; output: bedrooms.

Navigating the Manifold

DCGAN: adversarial training to generate images.

Trained on Manga characters

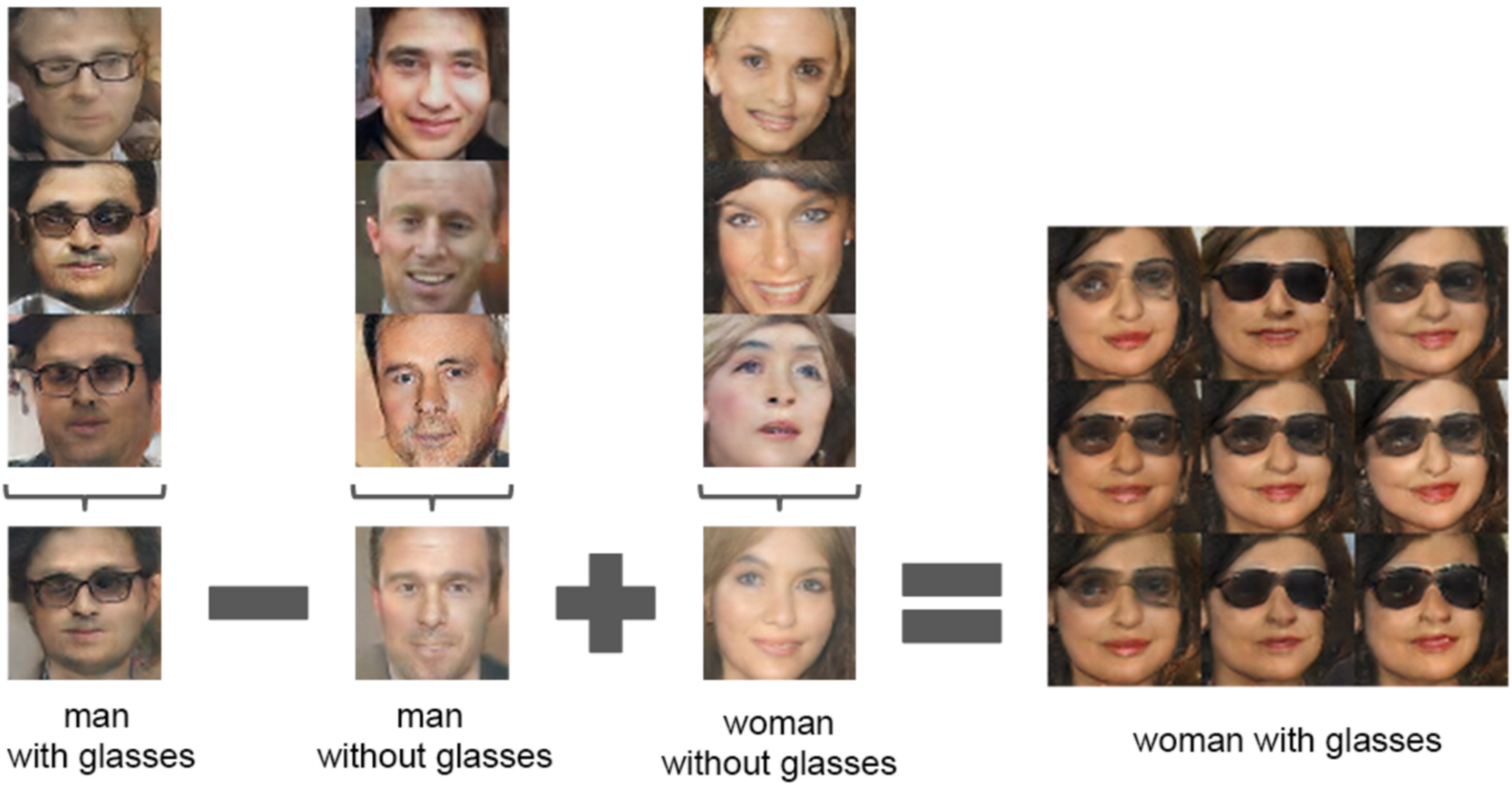
Interpolates between characters



Face Algebra (in DCGAN space)

DCGAN: adversarial training to generate images.

- [Radford, Metz, Chintala 2015]



Video Prediction
(with adversarial training)

[Mathieu, Couprie, LeCun ICLR 2016]
arXiv:1511:05440

Unsupervised Learning is the “Dark Matter” of AI

Y LeCun

Unsupervised learning is the only form of learning that can provide enough information to train large neural nets with billions of parameters.

– Supervised learning would take too much labeling effort

– Reinforcement learning would take too many trials

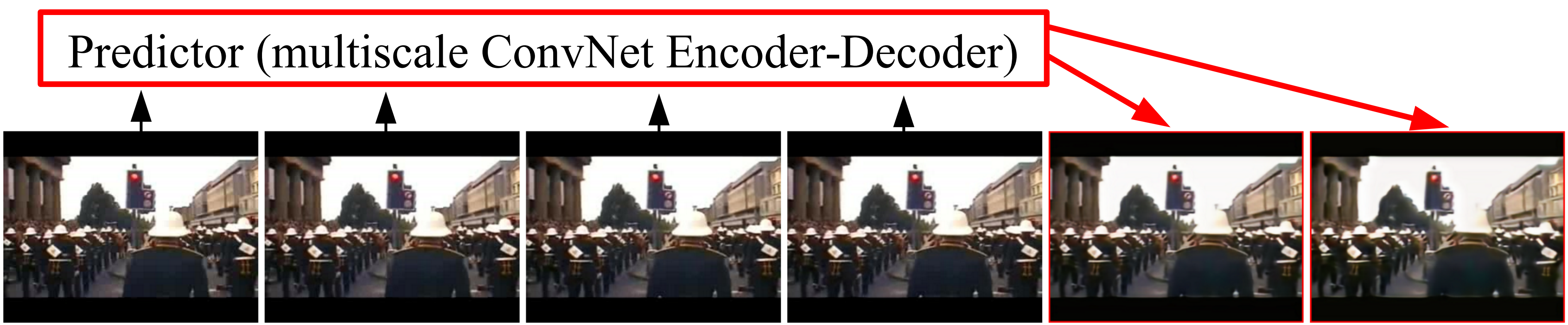
But we don't know how to do unsupervised learning (or even formulate it)

– We have lots of ideas and methods

– They just don't work that well yet.

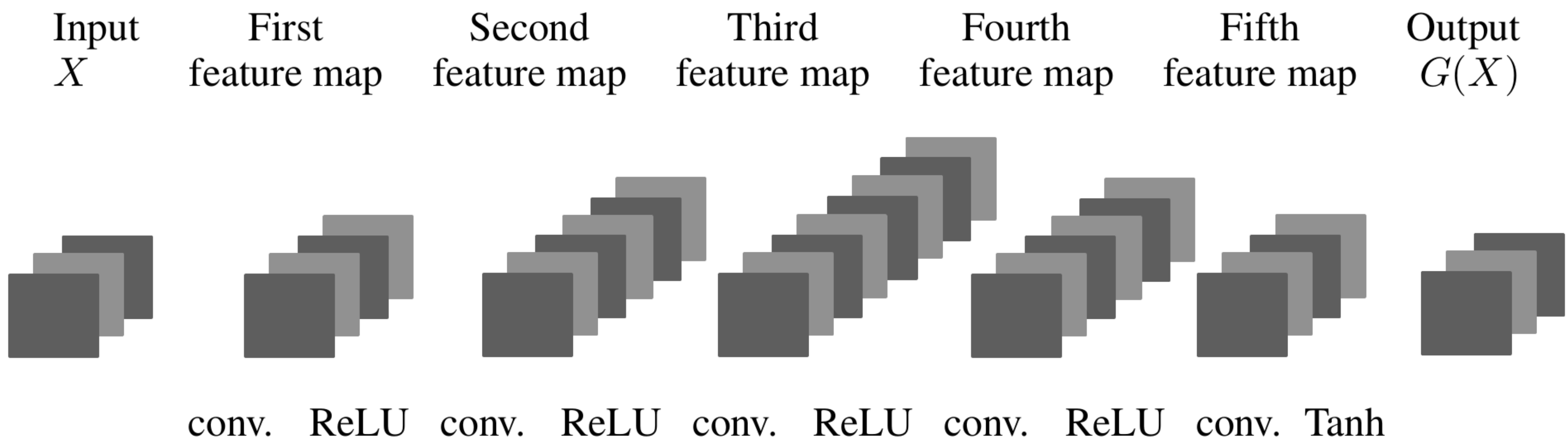
Why is it so hard? The world is unpredictable!

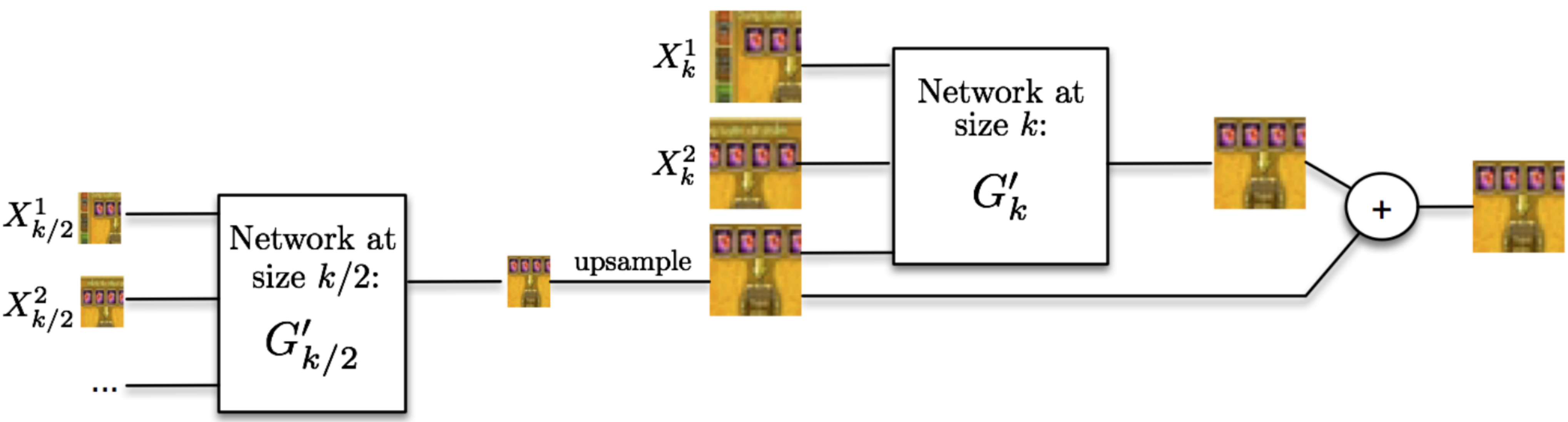
– Predictors produce an average of all possible futures → **Blurry image.**



Multi-Scale ConvNet for Video Prediction

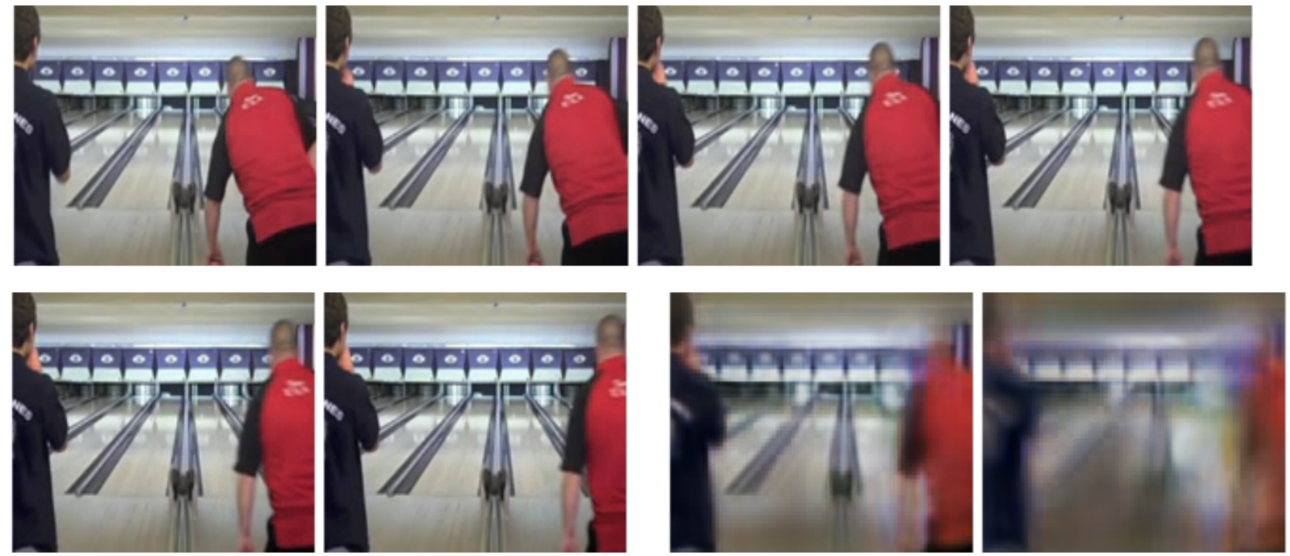
- **4 to 8 frames input → ConvNet with no pooling → 1 to 8 frames output**





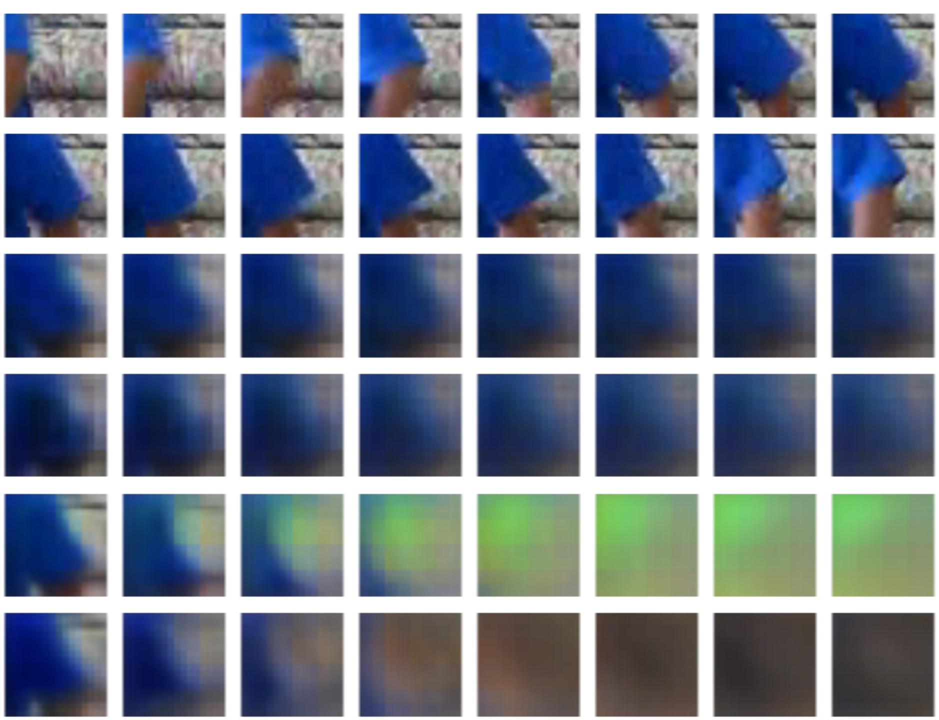
Can't Use Squared Error: blurry predictions

- **The world is unpredictable**
- **MSE training predicts the average of possible futures: blurry images.**



Ground truth

ℓ_2 result



Input

Ground truth

ℓ_1

ℓ_2

ℓ_1 recursive

ℓ_2 recursive

Multi-Scale ConvNet for Video Prediction

Architectures

Models 4 frames in input – 1 frame in output

Generative network scales	G_1	G_2	G_3	G_4
Number of feature maps	128, 256, 128	128, 256, 128	128, 256, 512, 256, 128	128, 256, 512, 256, 128
Conv. kernel size	3, 3, 3, 3	5, 3, 3, 5	5, 3, 3, 3, 5	7, 5, 5, 5, 5, 7
Adversarial network scales	D_1	D_2	D_3	D_4
Number of feature maps	64	64, 128, 128	128, 256, 256	128, 256, 512, 128
Conv. kernel size (no padding)	3	3, 3, 3	5, 5, 5	7, 7, 5, 5
Fully connected	512, 256	1024, 512	1024, 512	1024, 512

Models 8 frames in input – 8 frames in output

Generative network scales	G_1	G_2	G_3	G_4
Number of feature maps	16, 32, 64	16, 32, 64	32, 64, 128	32, 64, 128, 128
Conv. kernel size	3, 3, 3, 3	5, 3, 3, 3	5, 5, 5, 5	7, 5, 5, 5, 5
Adversarial network scales	D_1	D_2	D_3	D_4
Number of feature maps	16	16, 32, 32	32, 64, 64	32, 64, 128, 128
Conv. kernel size (no padding)	3	3, 3, 3	5, 5, 5	7, 7, 5, 5
Fully connected	128, 64	256, 128	256, 128	256, 128

Multi-Scale ConvNet for Video Prediction

Results on UCF101 (10% of test images)

- 8 frames input → 8 frames output

	1st frame prediction scores		8th frame prediction scores	
	PSNR	Sharpness	PSNR	Sharpness
ℓ_2	18.3	0.47	16.4	0.36
Adv	21.0	0.65	18.9	0.54
ℓ_1	21.2	0.57	18.3	0.51
GDL ℓ_1	**21.8**	**0.87**	**19.2**	**0.79**

Results on UCF101 (10% of test images)

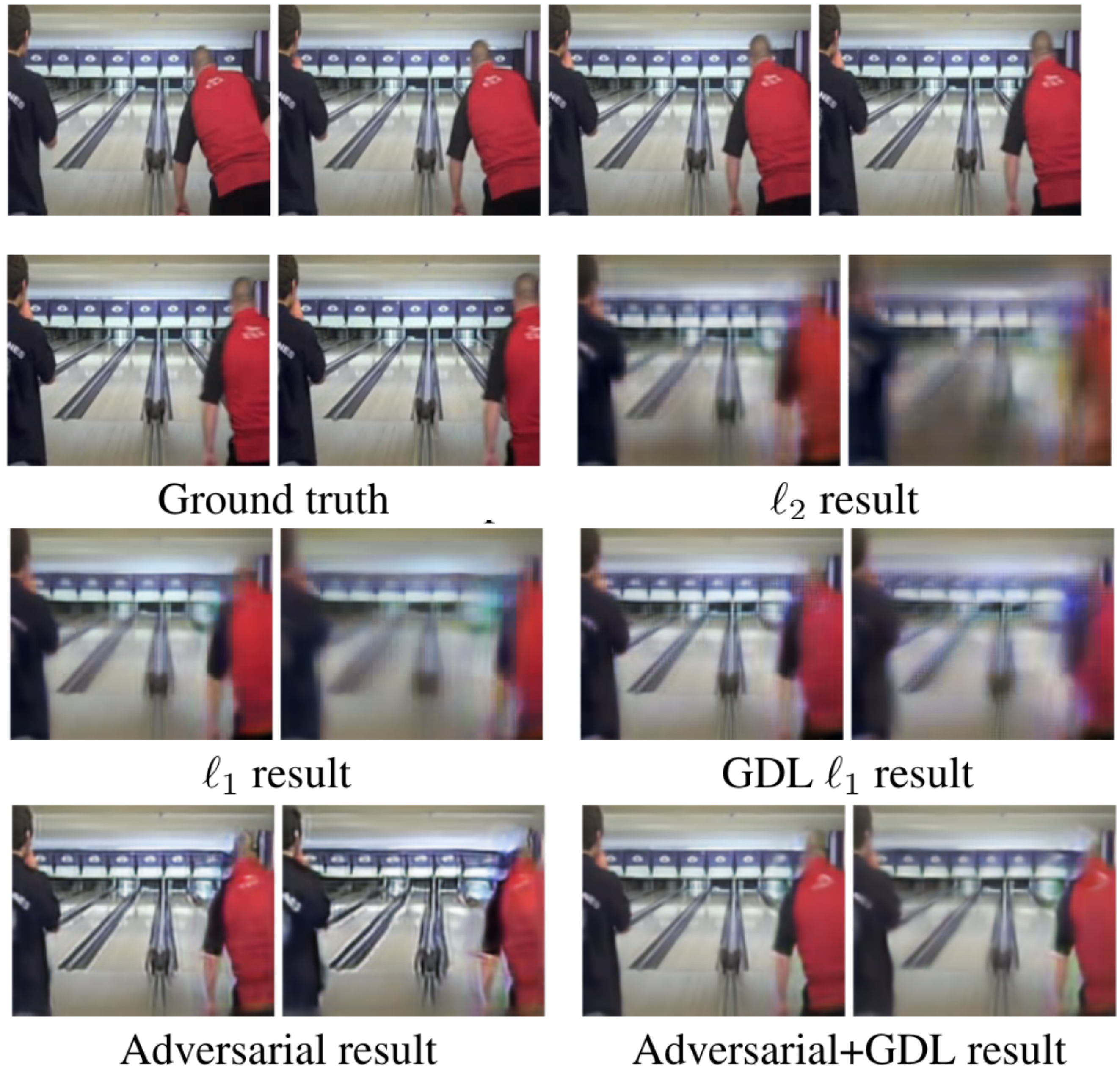
- 4 frames input → 1 frames output

	1st frame prediction scores		2nd frame prediction scores	
	PSNR	Sharpness	PSNR	Sharpness
ℓ_2	20.1	0.48	14.1	0.29
ℓ_1	22.2	0.58	16.0	0.48
GDL ℓ_1	23.9	0.69	18.5	0.60
Adv	24.4	**0.95**	18.9	**1.00**
Adv+GDL	**27.2**	0.77	**22.6**	0.68

Multi-Scale ConvNet for Video Prediction

Examples

Input frames

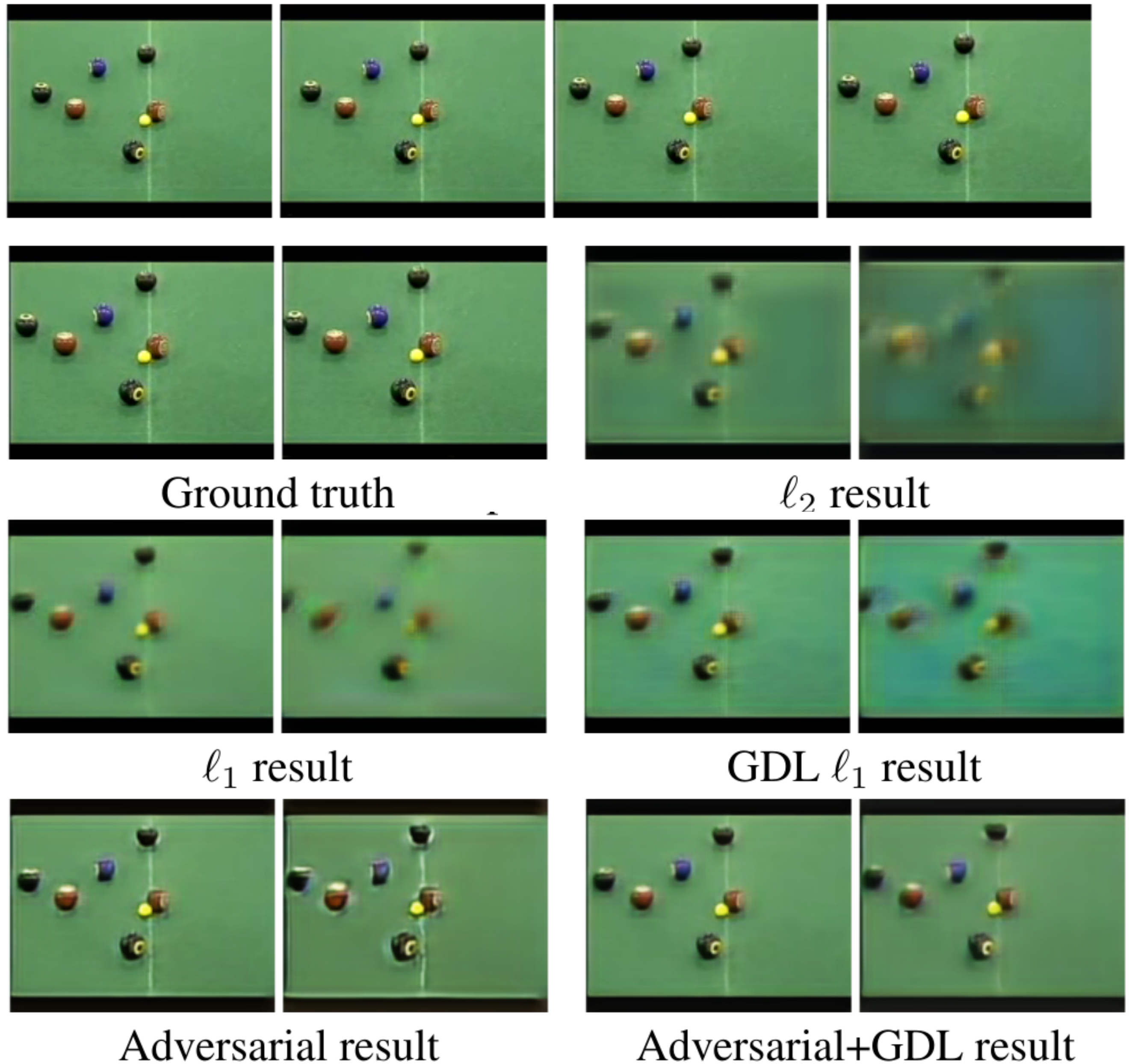


Multi-Scale ConvNet for Video Prediction

Y LeCun

Examples

Input frames



Predictive Learning: Video Prediction

Our brains are “prediction machines”

Can we train machines to predict the future?

Some success with “adversarial training”

- [Mathieu, Couprie, LeCun ICLR’16, arXiv:1511:05440]

But we are far from a complete solution.