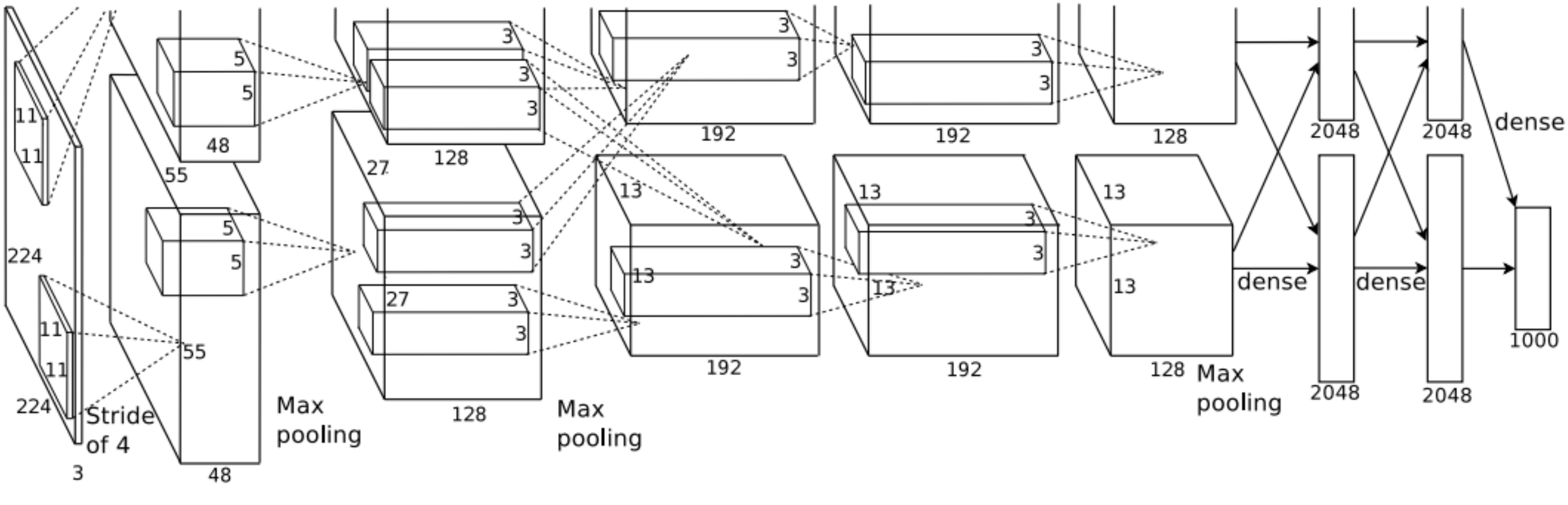
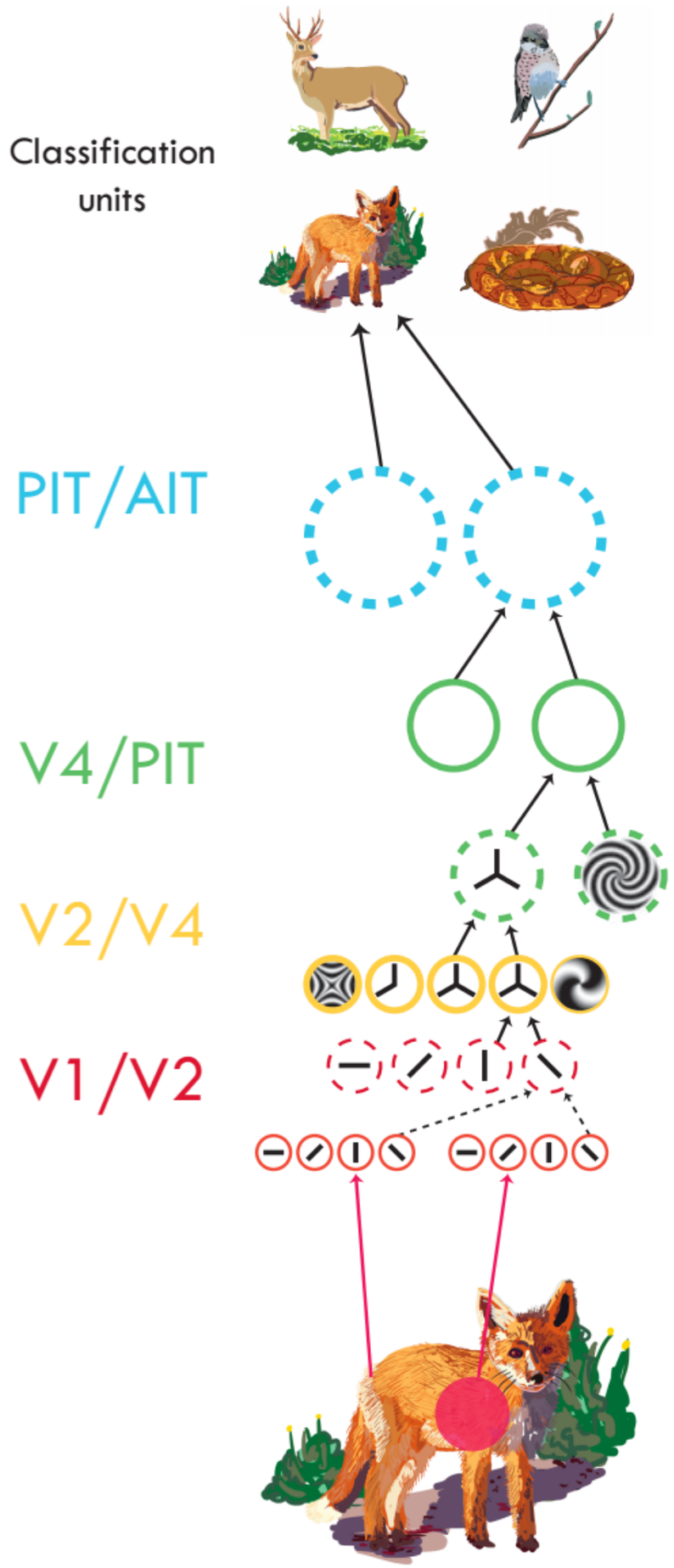


Using and Understanding Deep Neural Nets

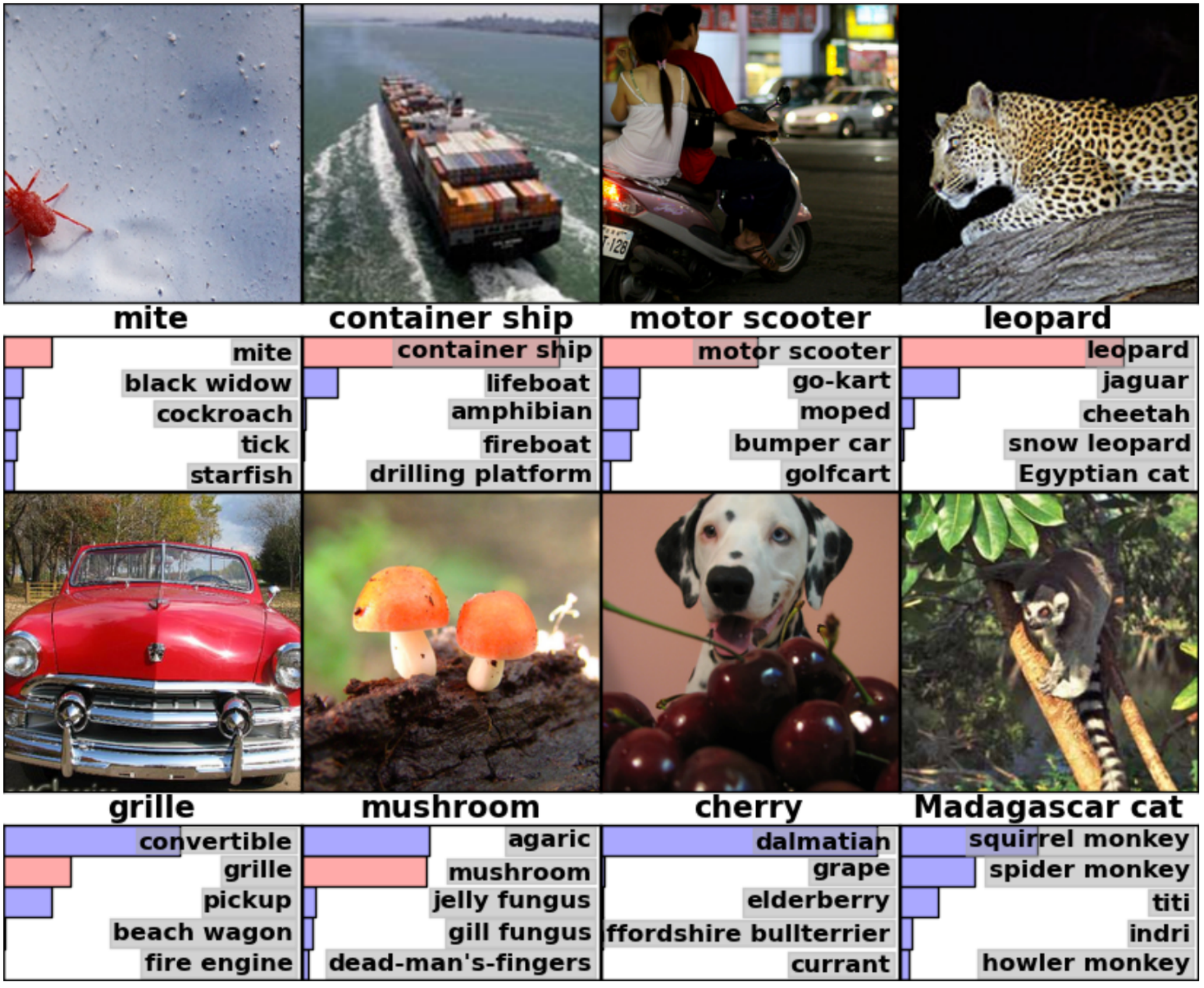
Phillip Isola

Tutorial Series in Computational topics for BCS

6/17/15



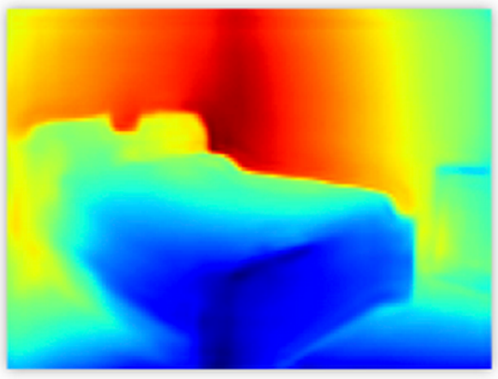
Serre, 2014



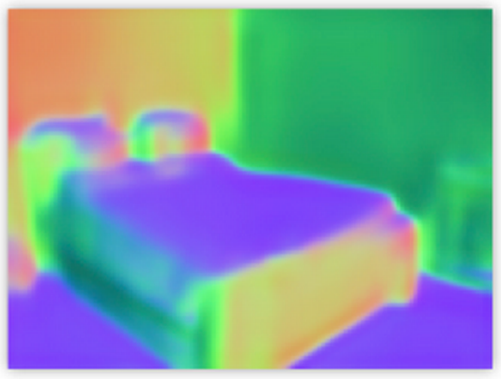
Krizhevsky et al. NIPS 2012

a pot of broccoli on a stove

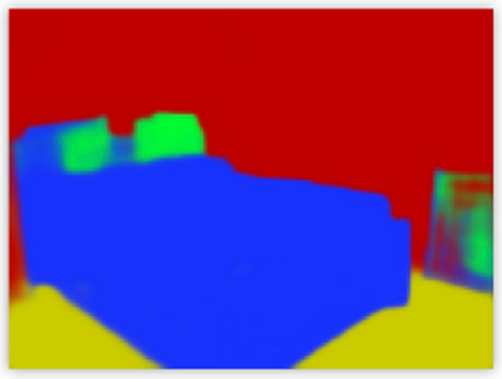
Input Image



Depth

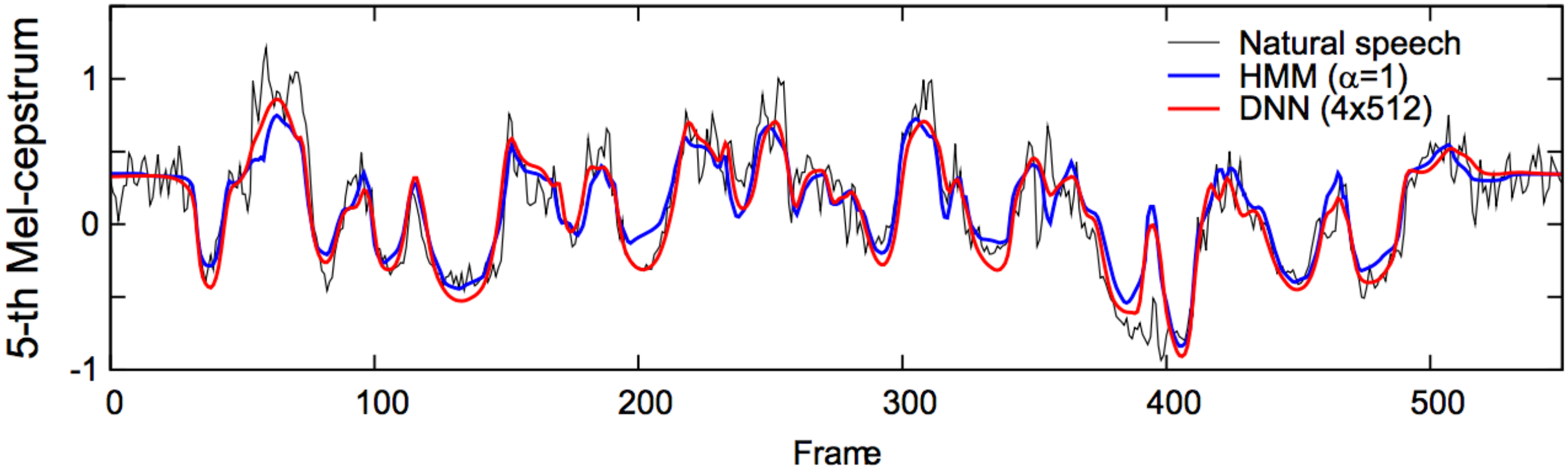


Normals



Labels

Eigen & Fergus 2014



Zen et al. 2013

Outline

1. How do they work?
2. What do they learn?
3. Practical use

Outline

1. **How do they work?**
2. What do they learn?
3. Practical use

Basic idea

Brain/Machine

“clown fish”

Object recognition

Edges

"Textons"

Colors

Segments

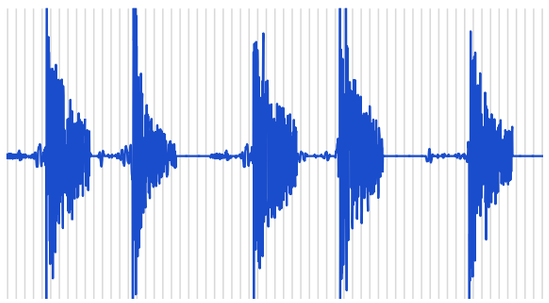
Parts

"clown fish"

Feature extractors

Classifier

Music recognition



Spectrogram

Notes

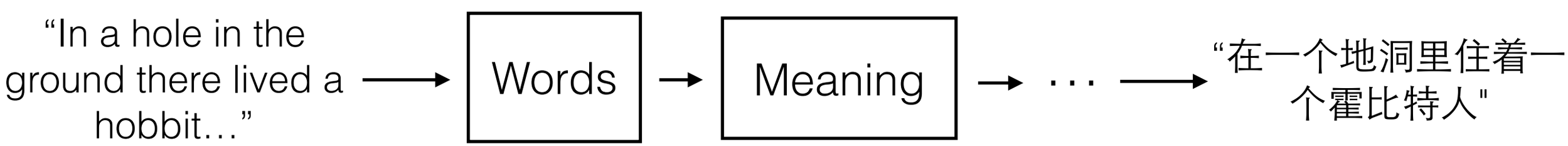
…

Bach Cello Suite #1

Feature extractors

Classifier

Text translation



Object recognition

Learned

Edges

"Textons"

Colors

Segments

Parts

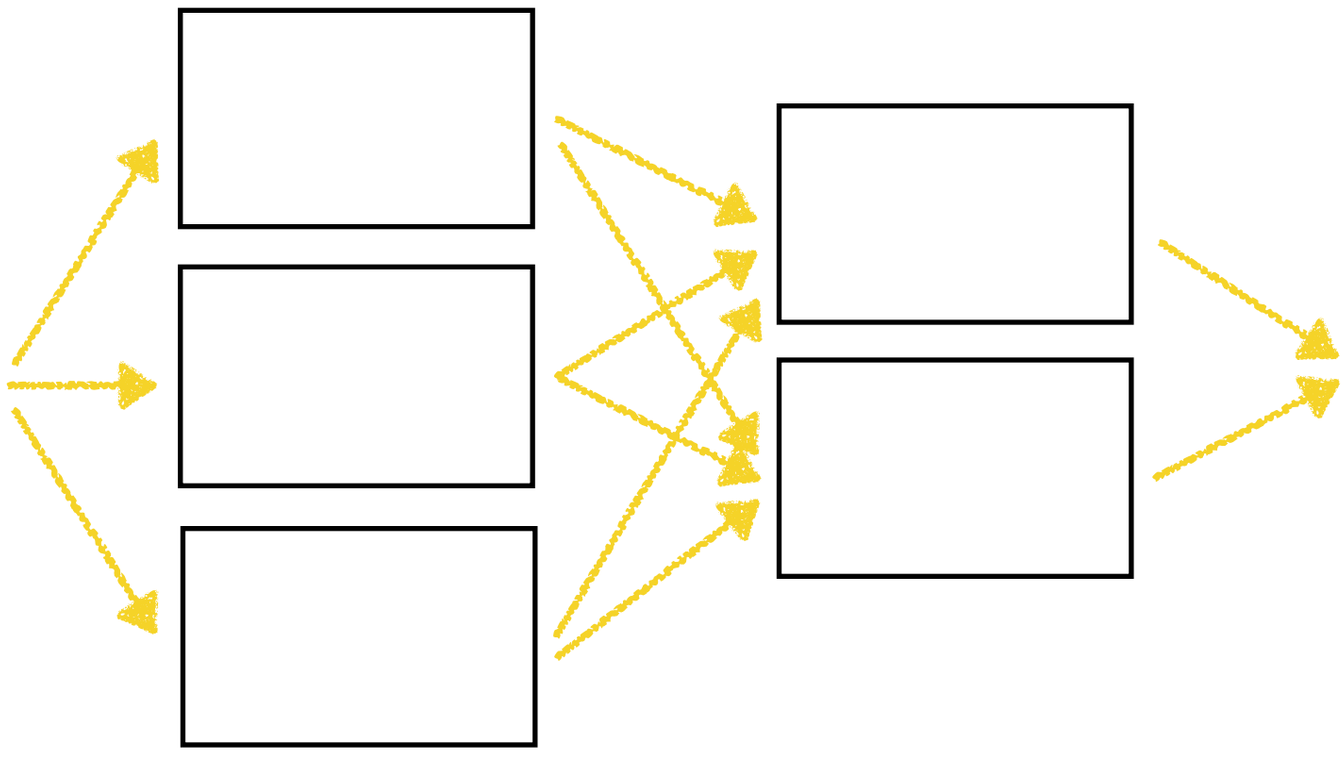
"clown fish"

Feature extractors

Classifier

Neural network

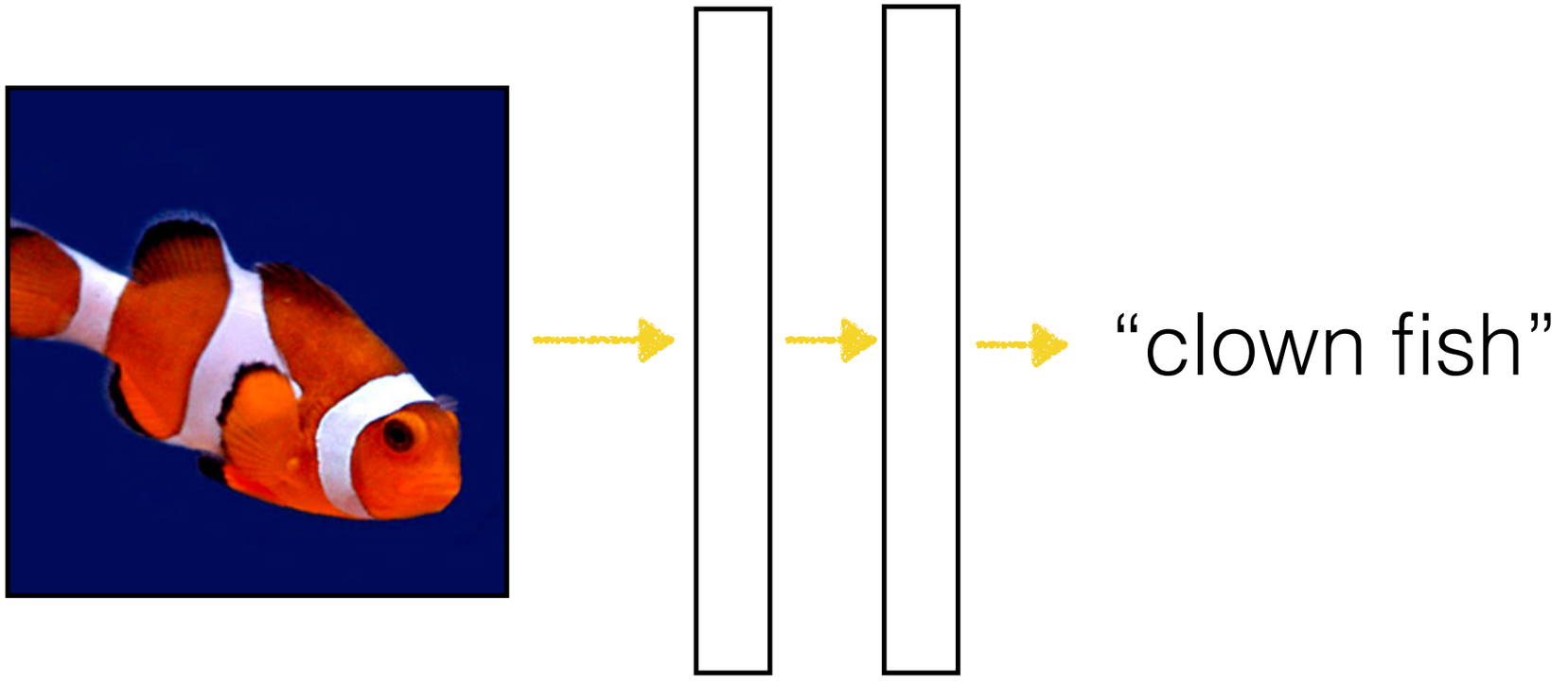
Learned



"clown fish"

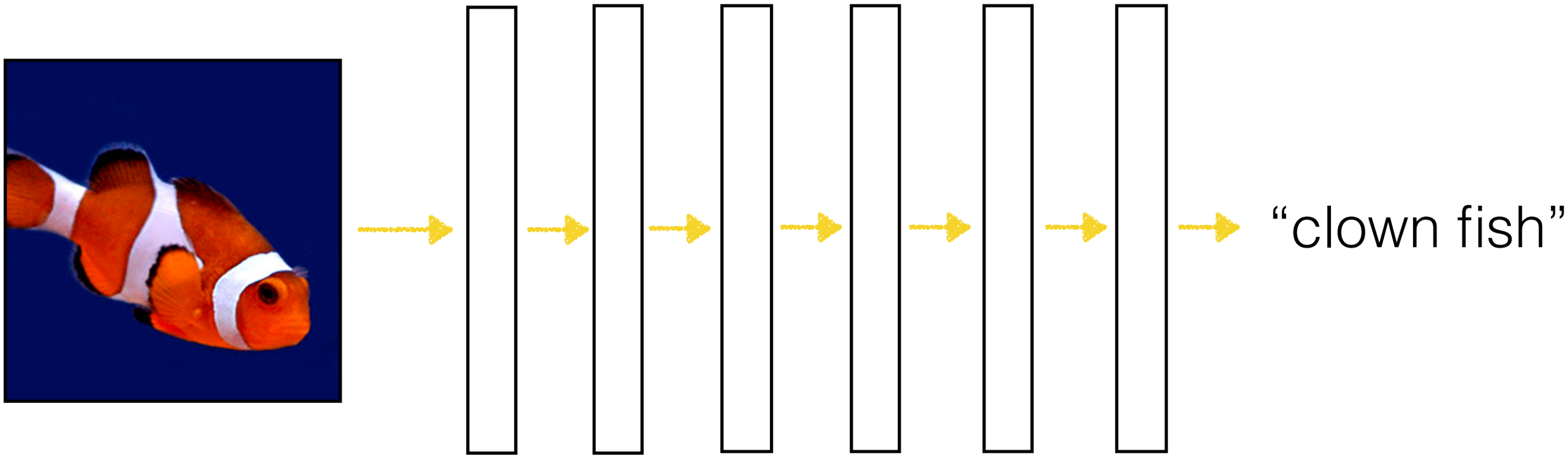
Neural network

Learned



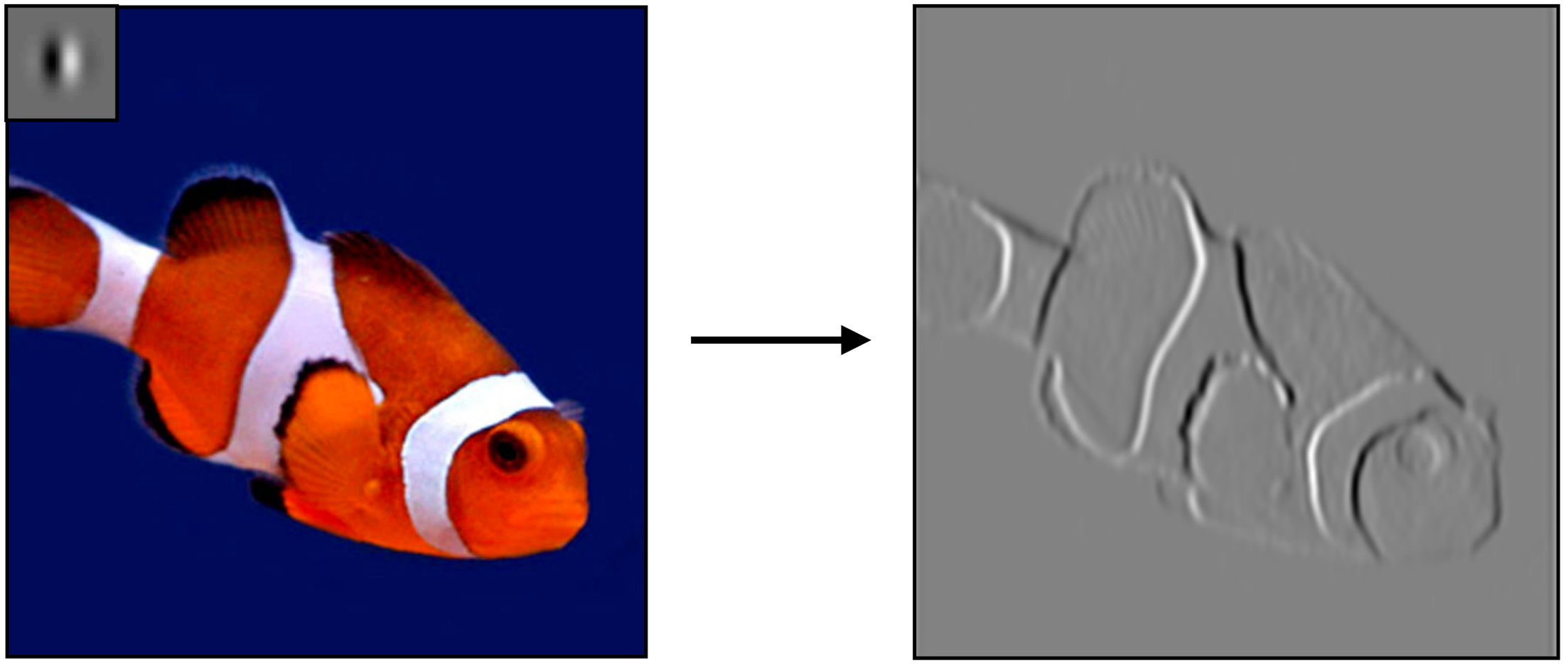
Deep neural network

Learned

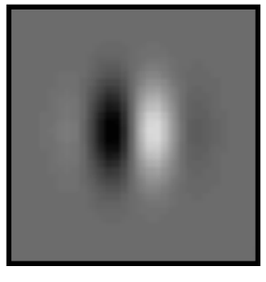


Convolutional Neural Nets

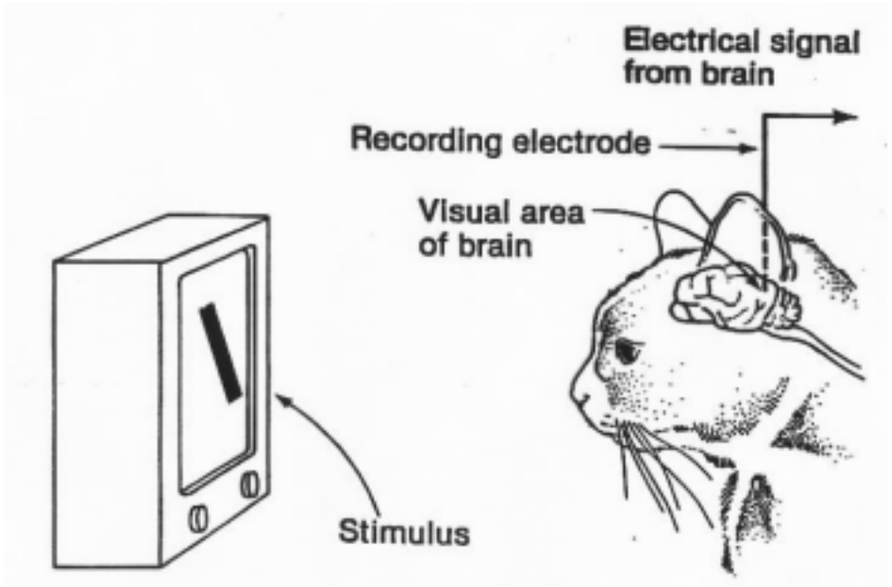
Convolution

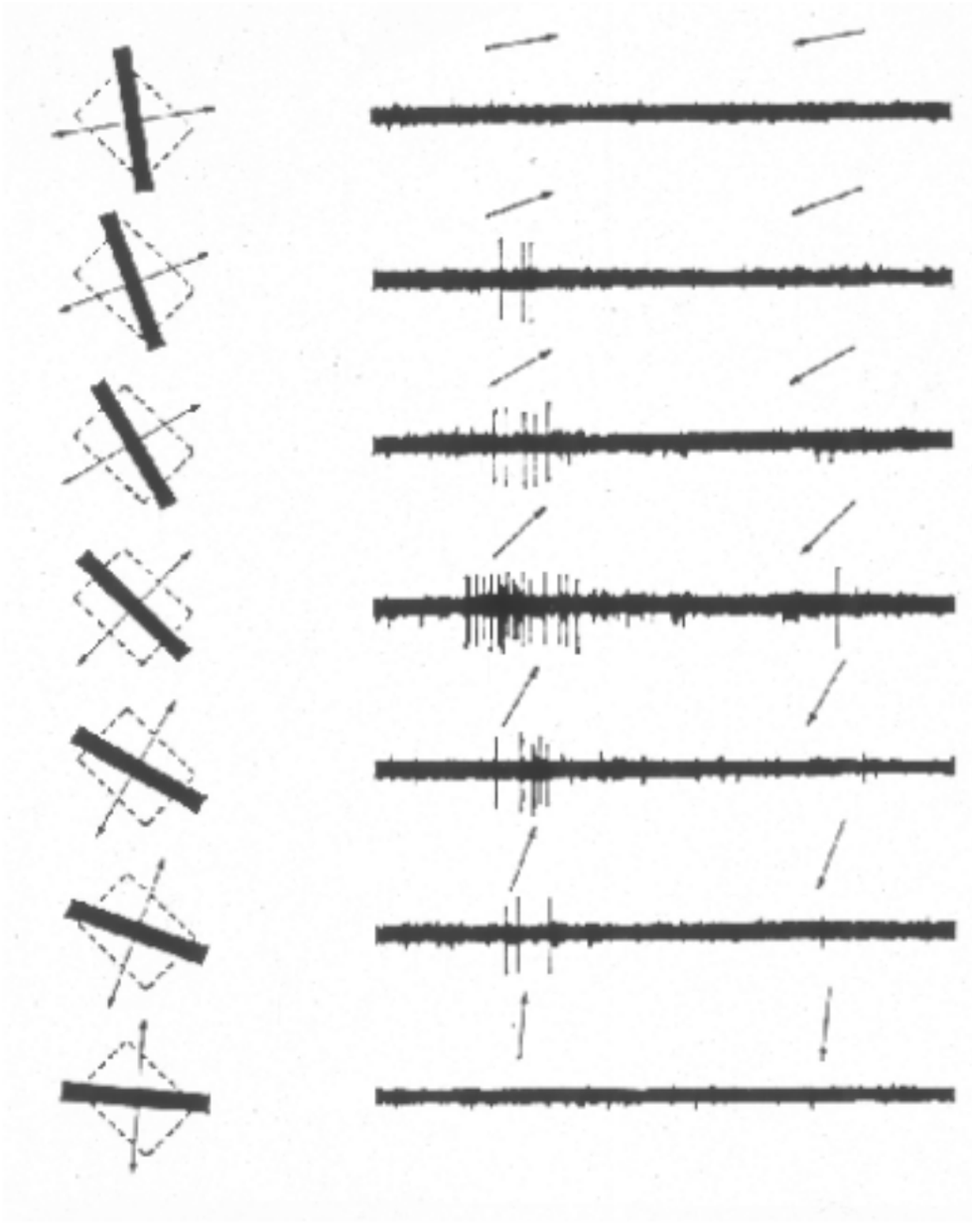


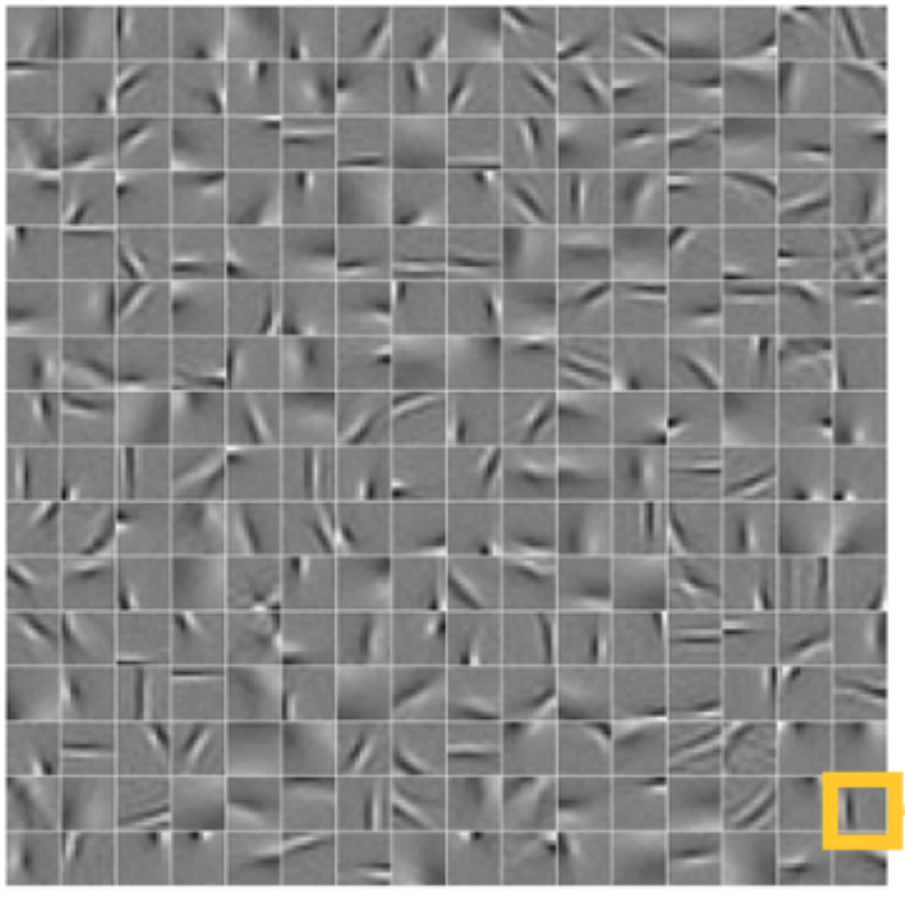
filter

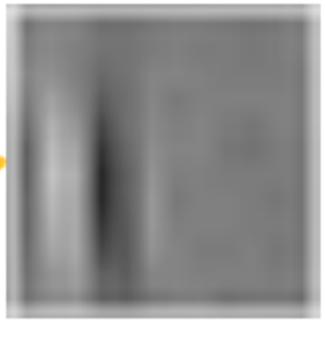


[Hubel and Wiesel 59]





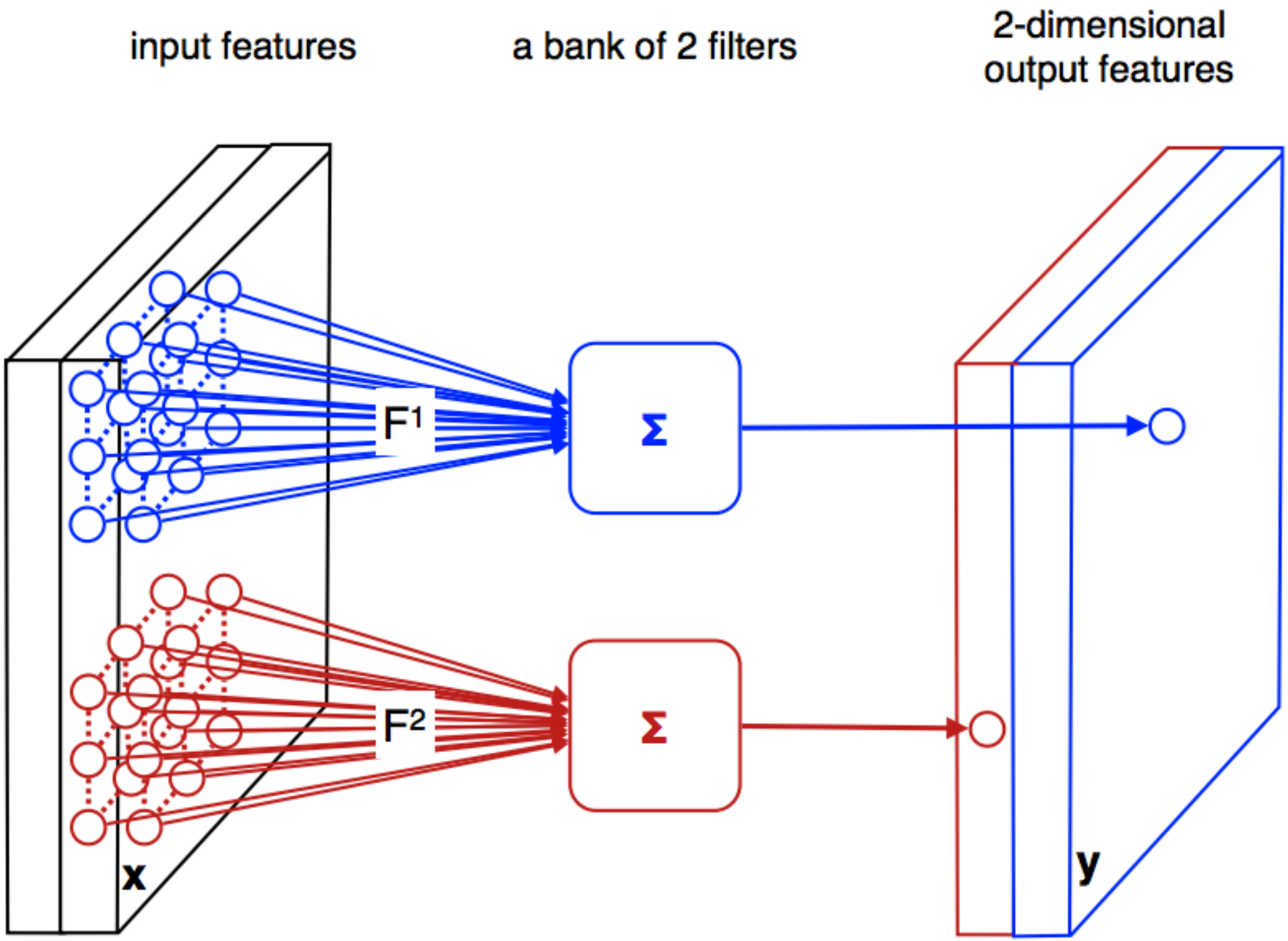




oriented filter

Slide from Andrea Vedaldi

Convolutional Neural Nets

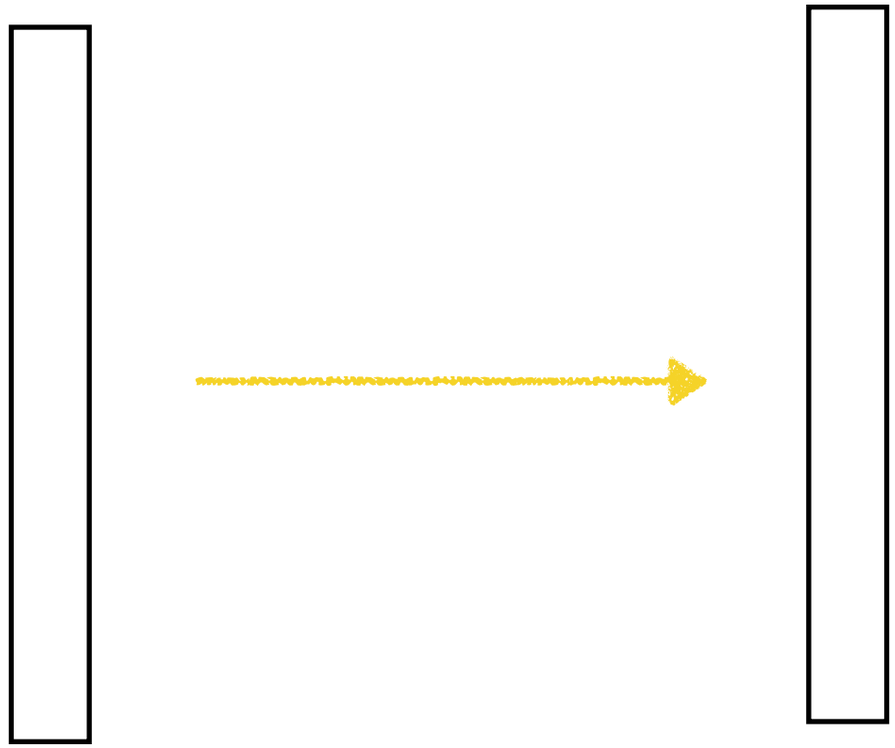


Slide from Andrea Vedaldi

Computation in a neural net

Input representation

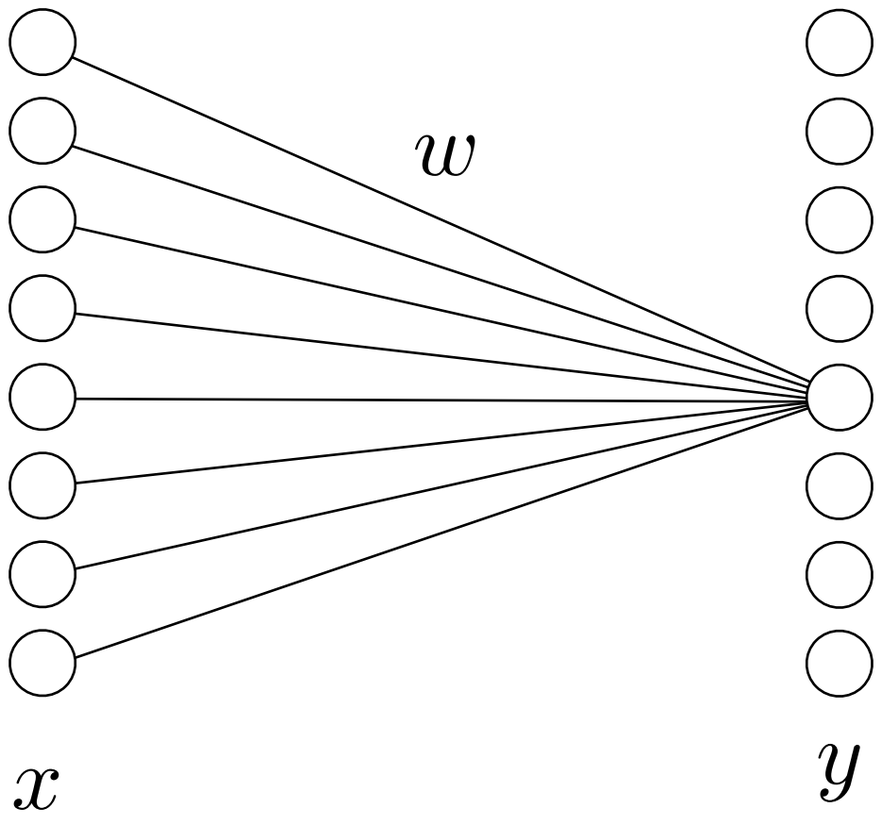
Output representation



Computation in a neural net

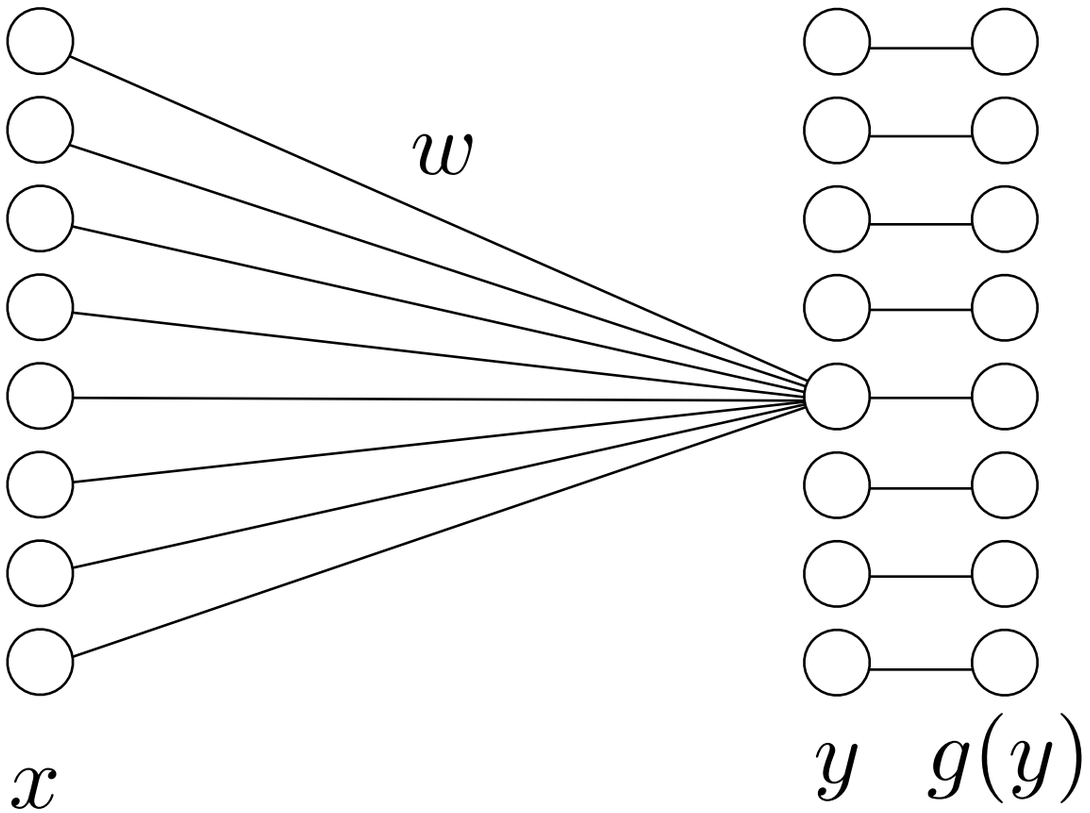
Input representation

Output representation

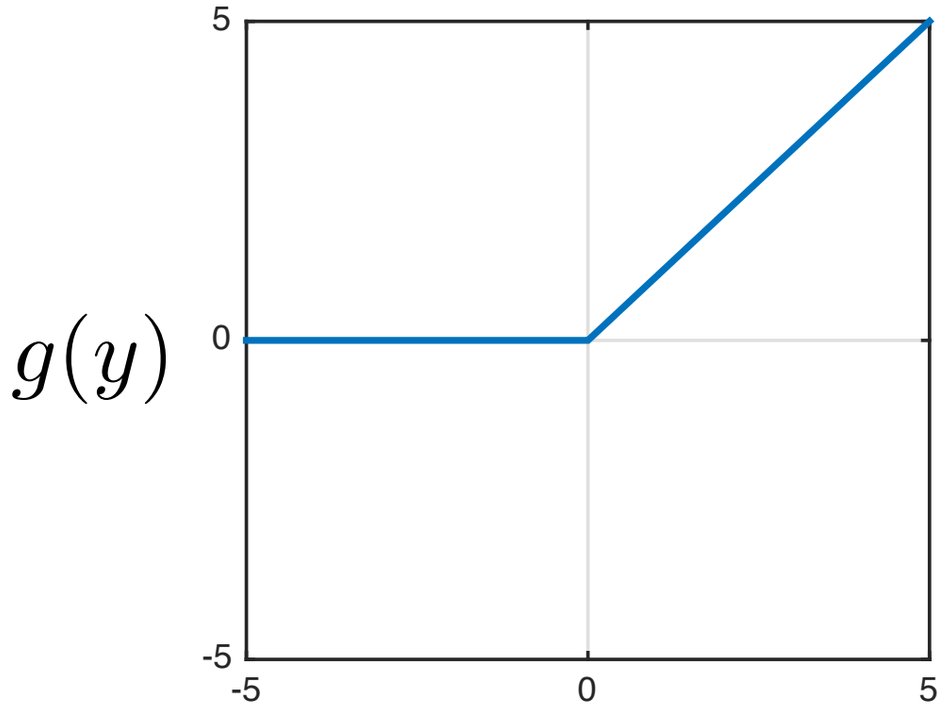


$$y_j = \sum_i w_{ij} x_i$$

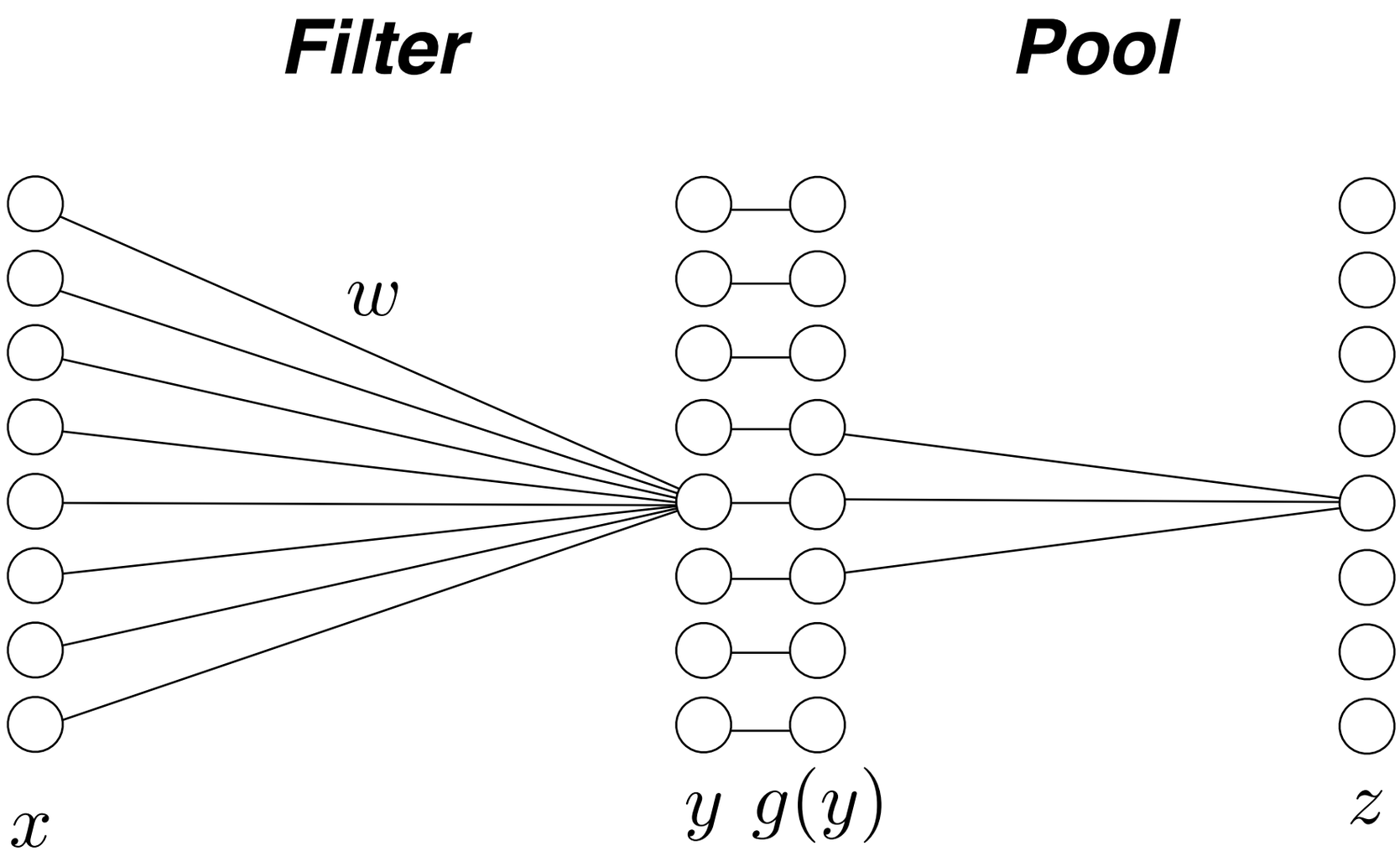
Computation in a neural net



Rectified linear unit (ReLU)

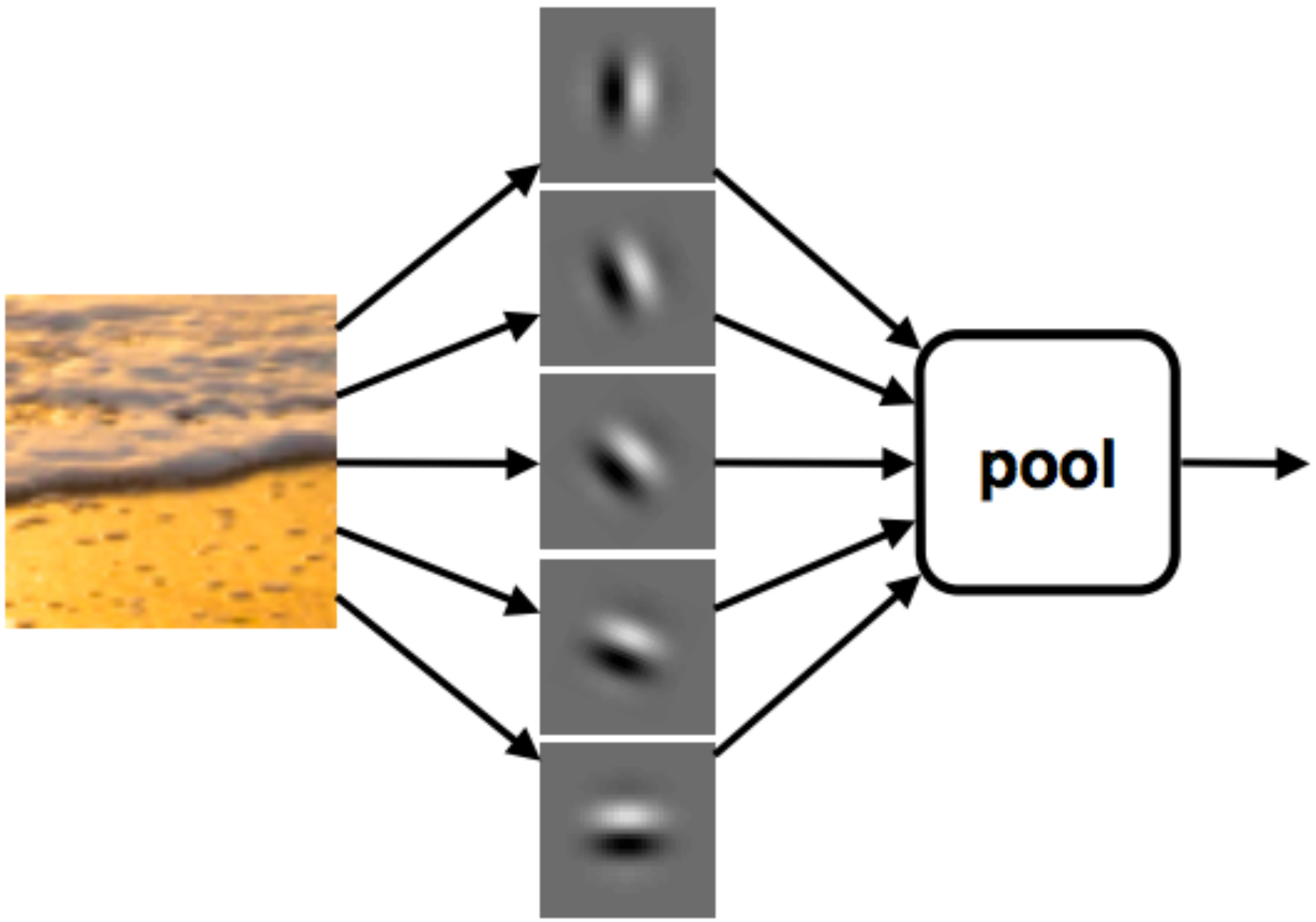


Computation in a neural net



$$y_j = \sum_i w_{ij} x_i \qquad\qquad z_k = \max_{j \in \mathcal{N}(j)} g(y_j)$$

Pooling across feature channels (filter outputs) can achieve invariance.



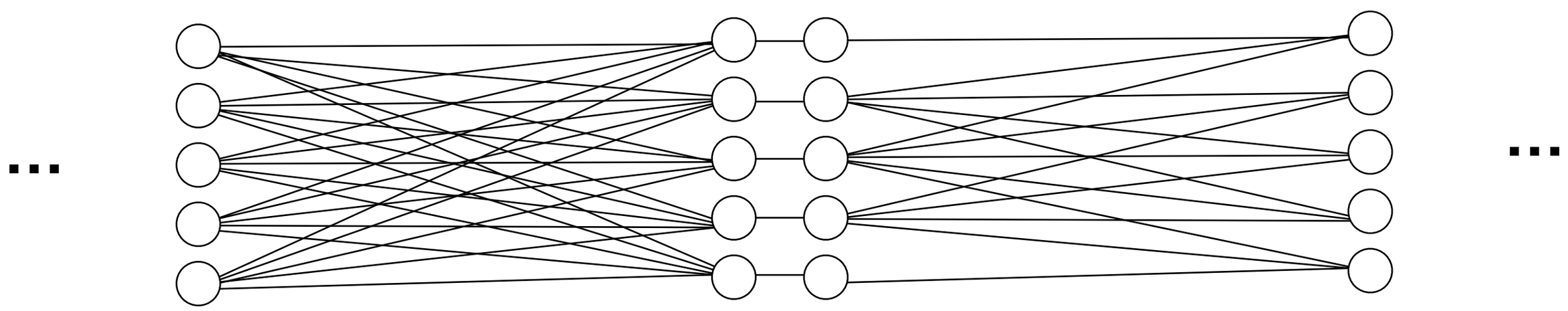
large response for any edge, regardless of its orientation

Slide from Andrea Vedaldi

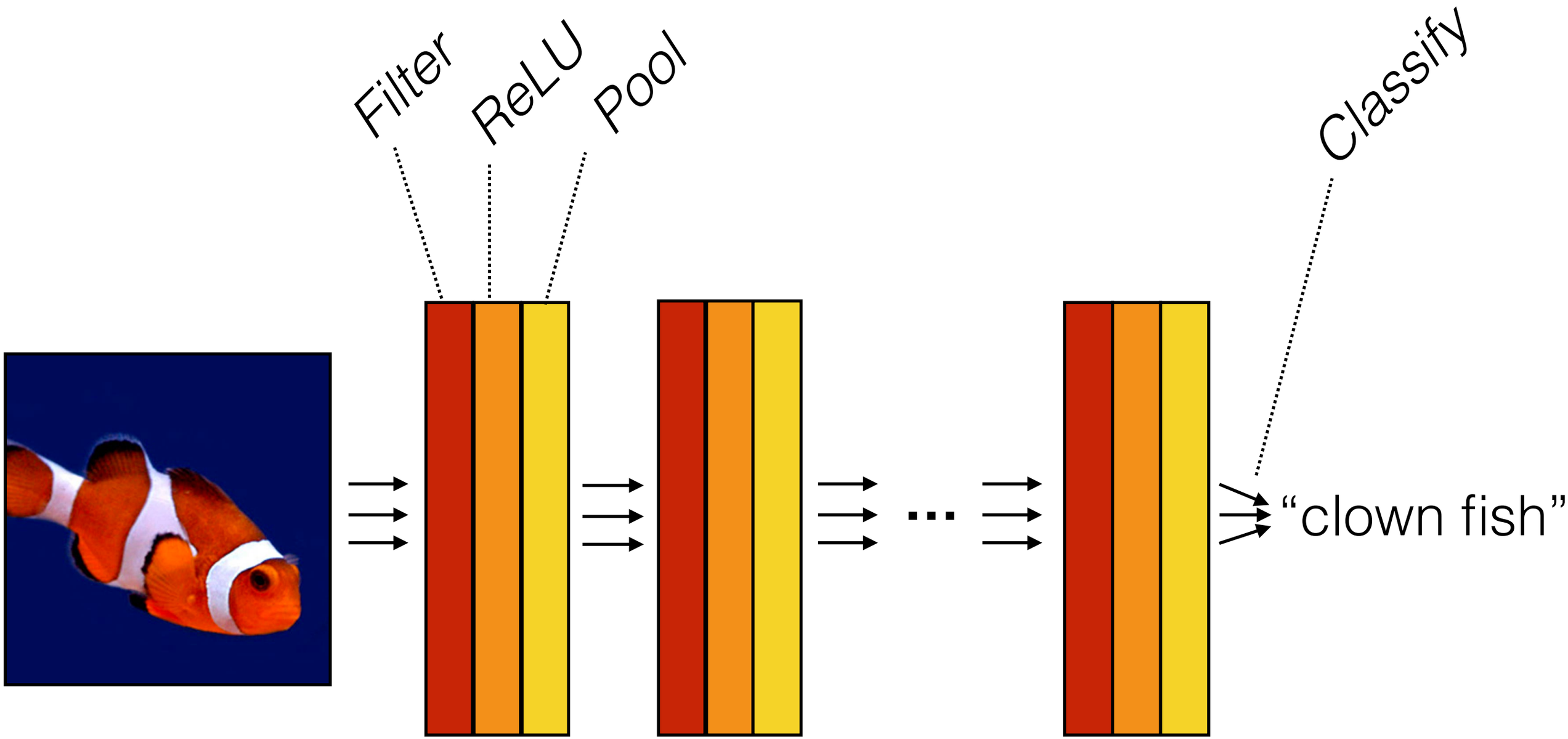
Computation in a neural net

One “layer” of a CNN

Filter ***ReLU*** ***Pool***



Computation in a neural net



$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

Computation in a neural net

Ingredients

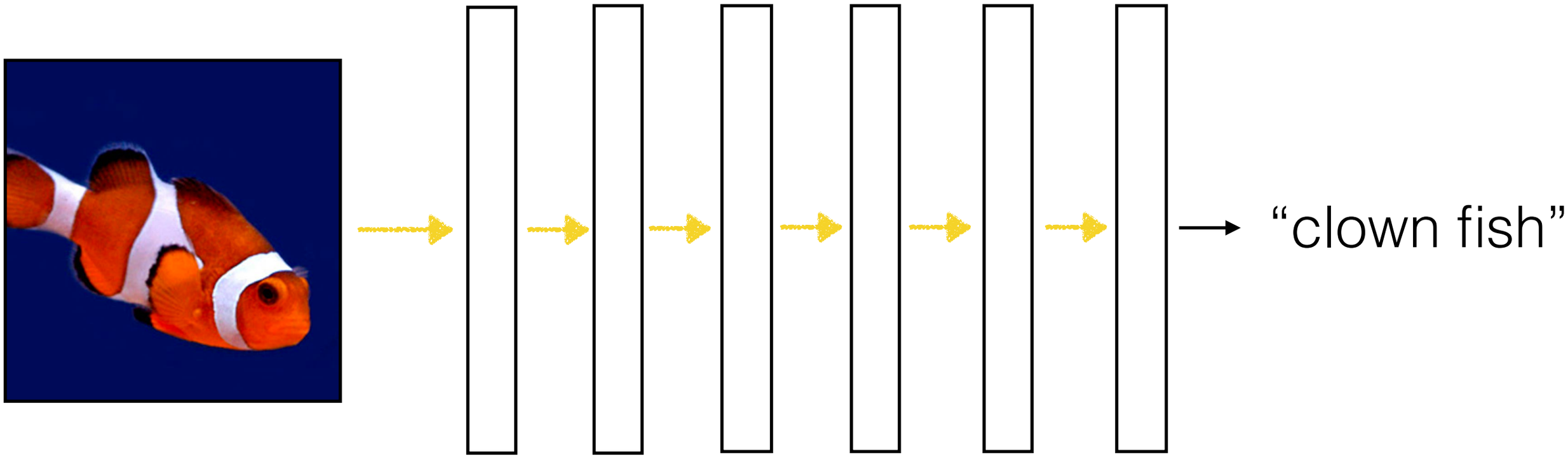
Select important features of the data
— linear filters
— pointwise nonlinearity

Group features that all indicate the same thing
— pooling

Repeat to achieve greater abstraction

Learning with deep nets

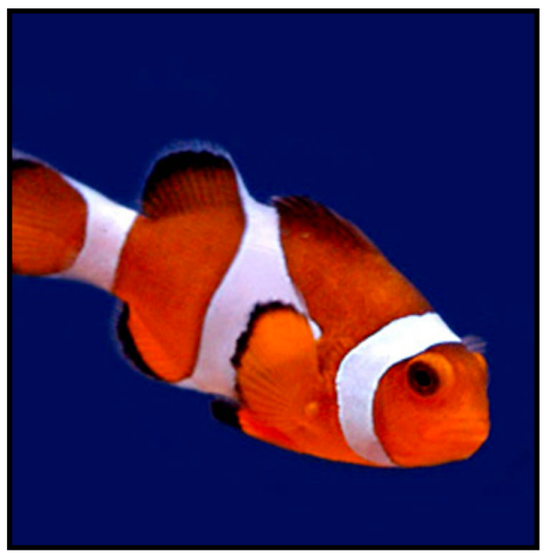
Learned

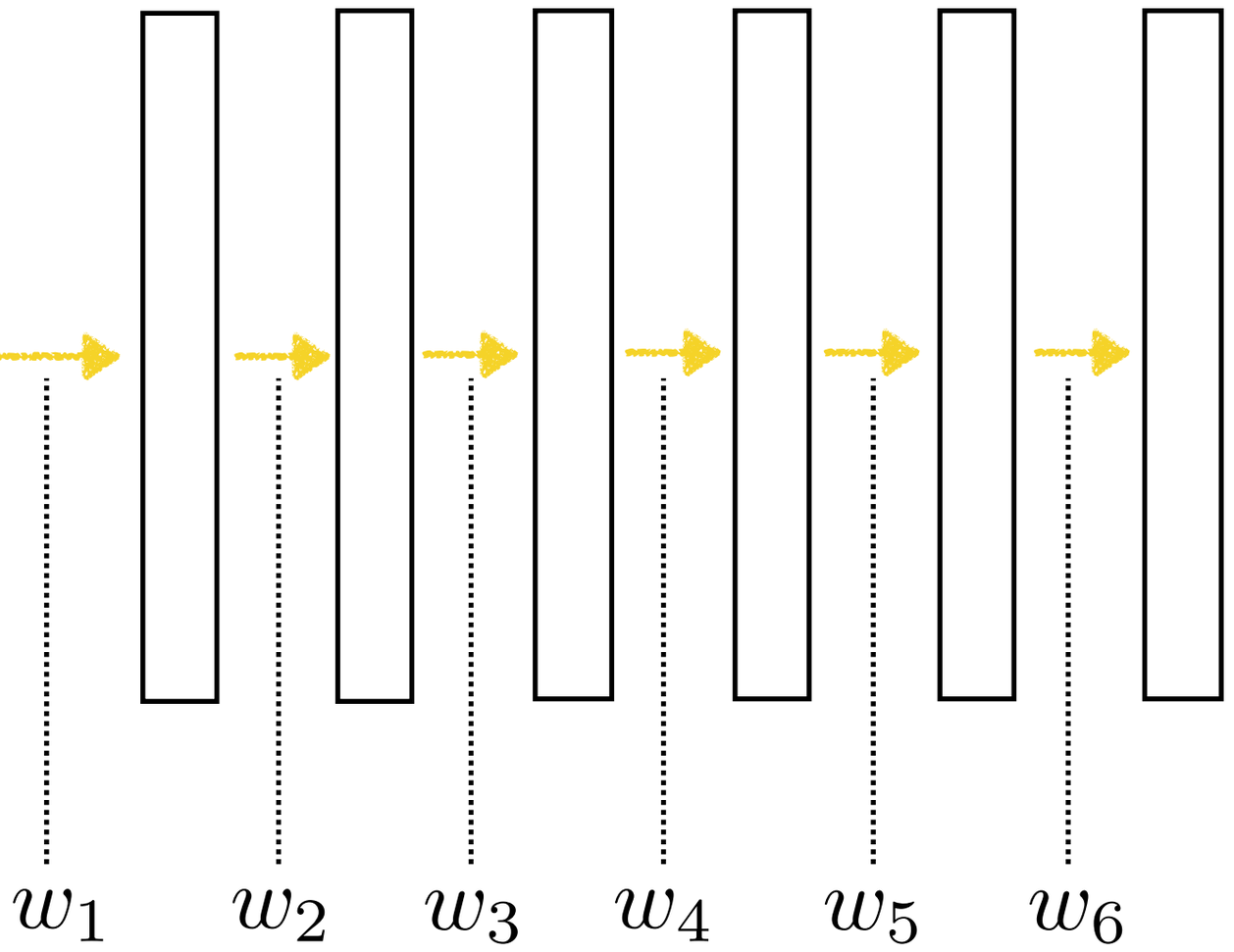


Learning with deep nets

Learned

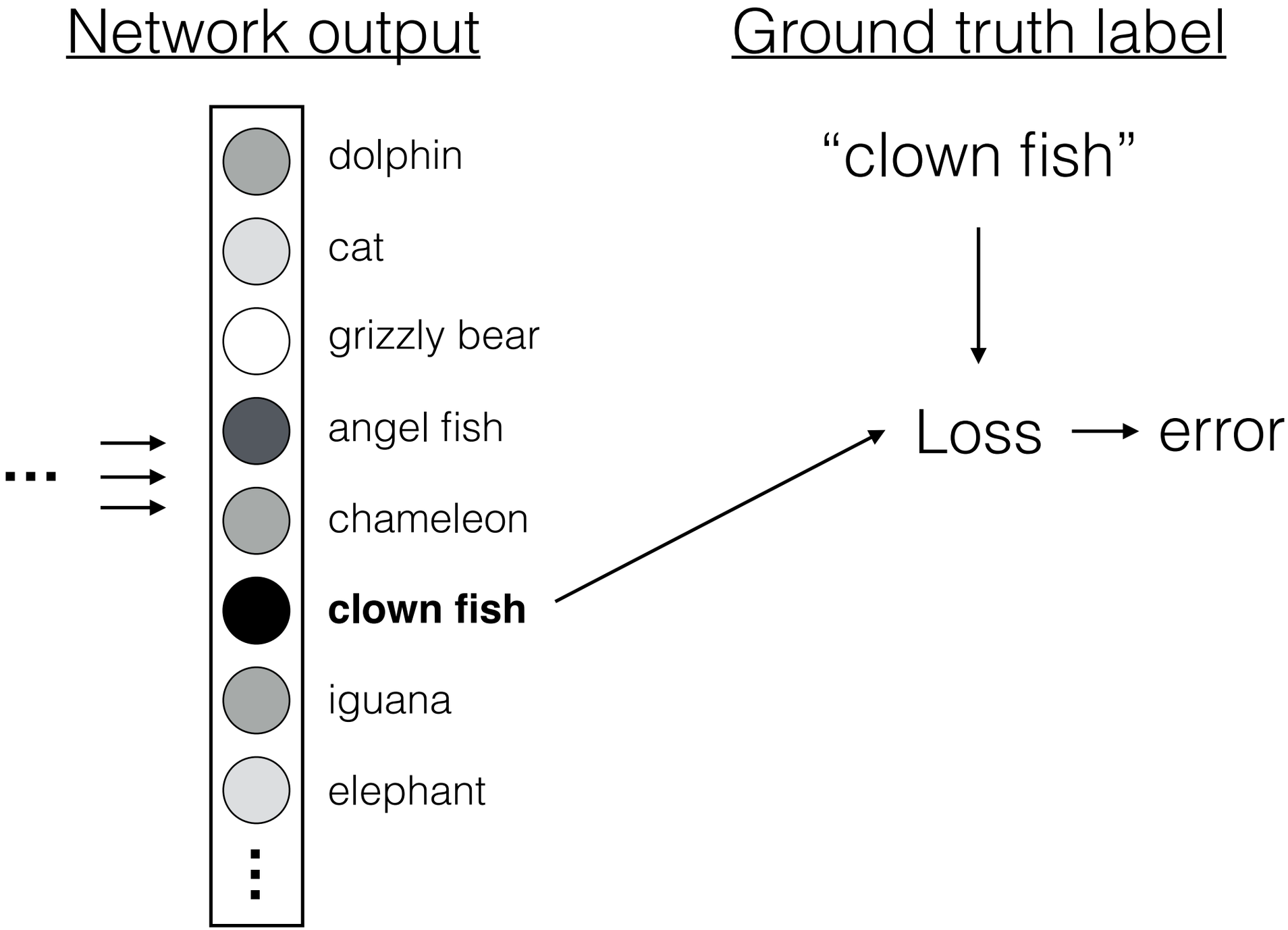
"clown fish"



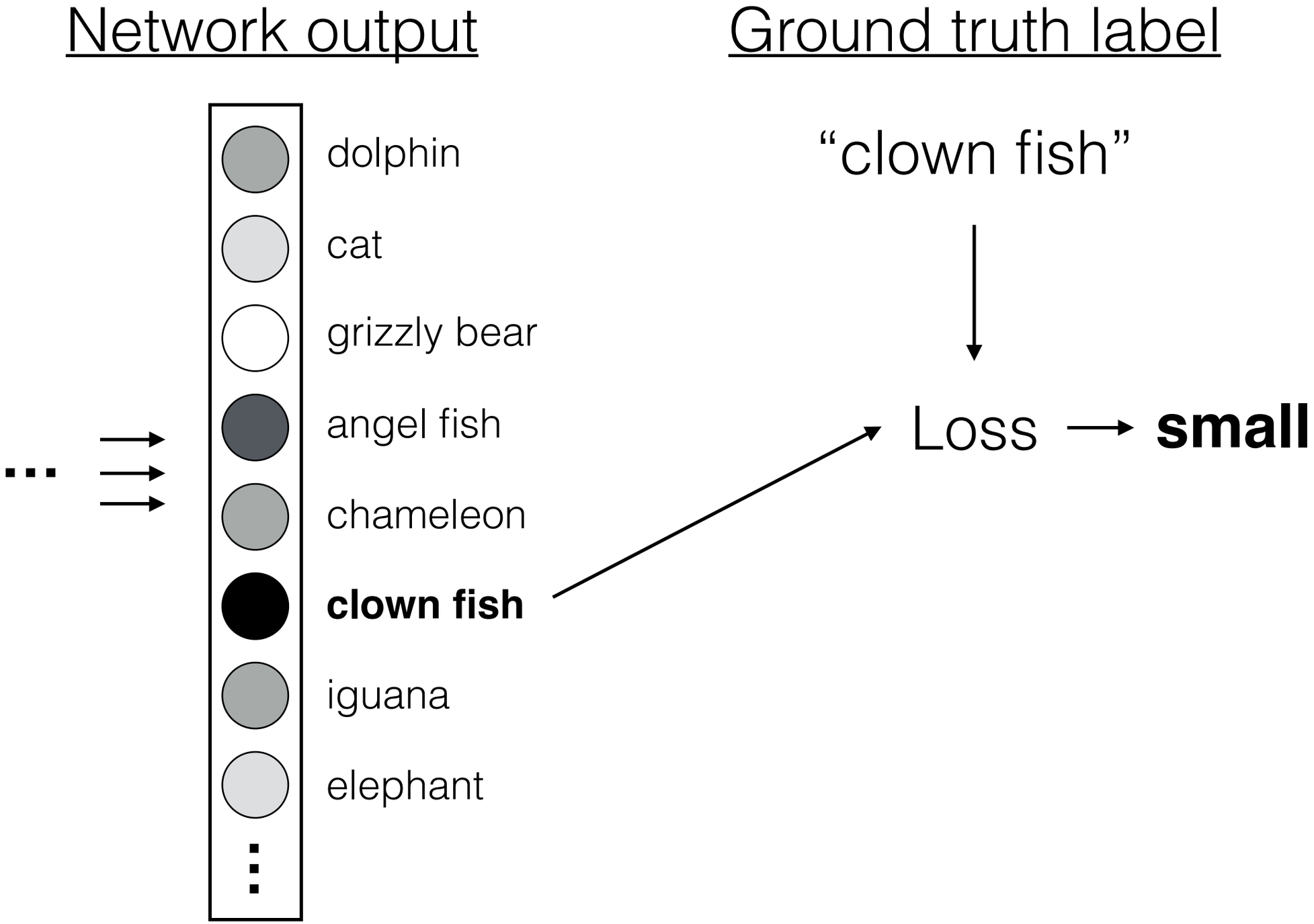


$$\underset{\mathbf{w}}{\text{argmin}} \quad L(w_1, \ldots, w_6)$$

Loss function



Loss function



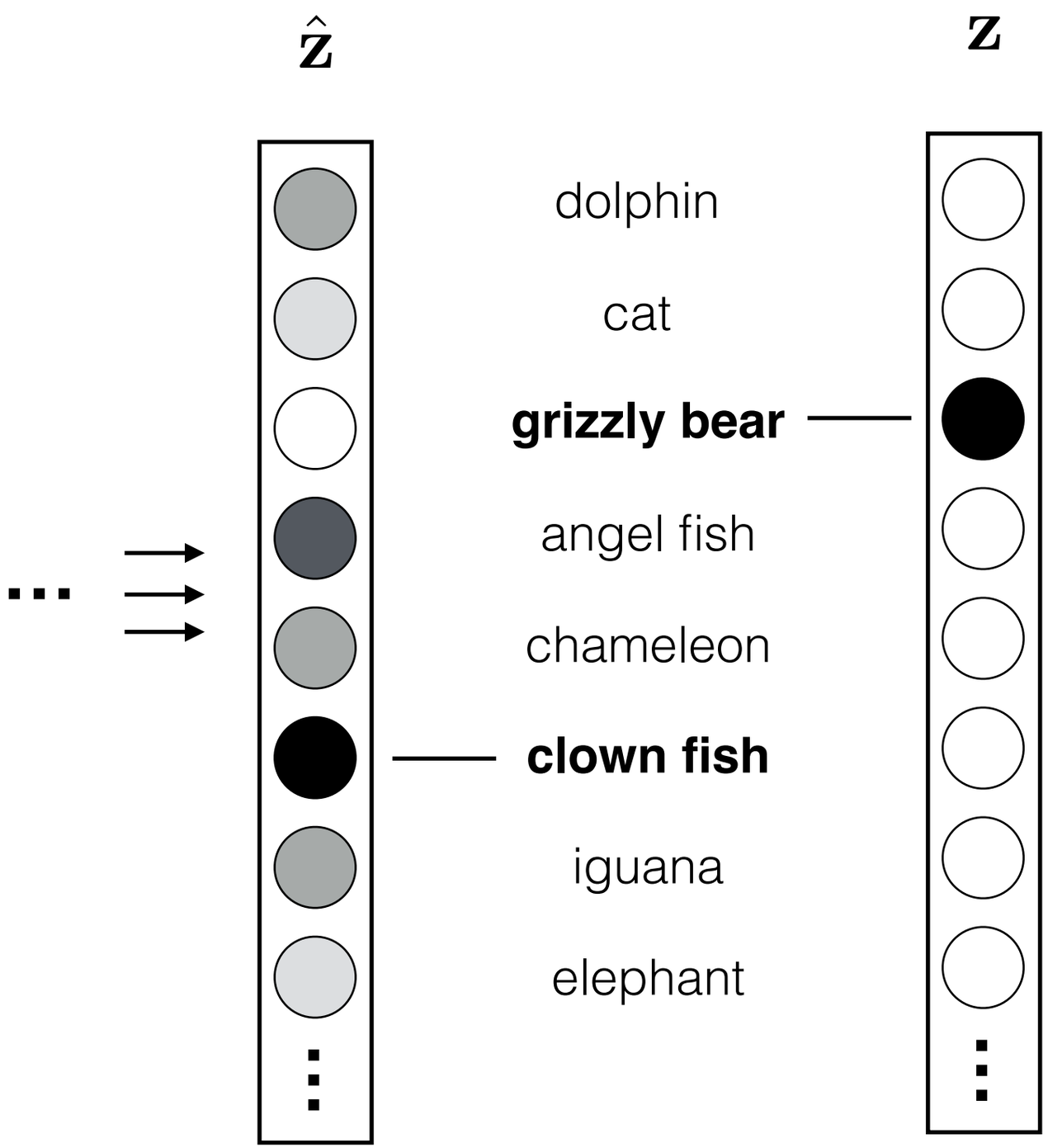
Loss function

Network output

Ground truth label

Loss function for classification

Network output Ground truth label



Cross-entropy loss

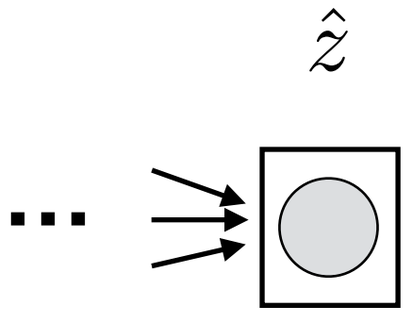
$$H(\hat{\mathbf{z}}, \mathbf{z}) = -\sum_c \hat{\mathbf{z}}_c \log \mathbf{z}_c$$

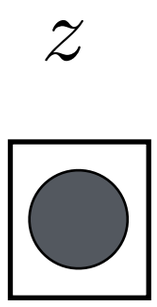
Loss function for regression

Task: *How much force should I apply to open this door?*

Network output: "100 Newtons"

Ground truth value: "10 Newtons"





Euclidean loss

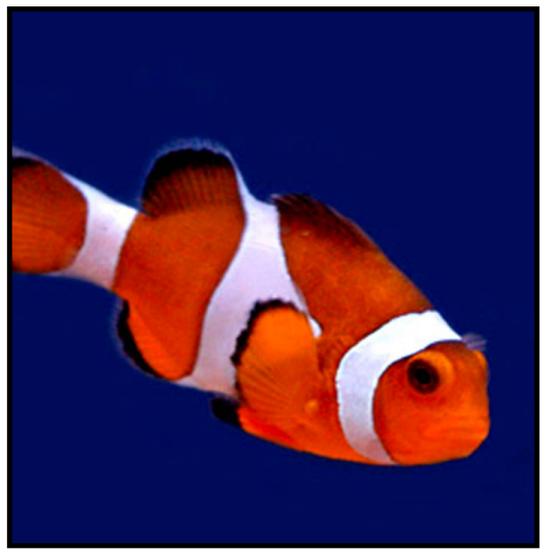
$$\|\hat{z} - z\|_2^2$$

Learning with deep nets

Learned

$\mathbf{z}_1$
"clown fish"

$\mathbf{x}_1$



Loss

$\ell(\mathbf{z}_1, f(\mathbf{x}_1; \mathbf{w}))$

w_1 w_2 w_3 w_4 w_5 w_6

Learning with deep nets

Learned

$\mathbf{z}_2$

"grizzly bear"

$\mathbf{x}_2$

Loss

$\ell(\mathbf{z}_2, f(\mathbf{x}_2; \mathbf{w}))$

w_1 w_2 w_3 w_4 w_5 w_6

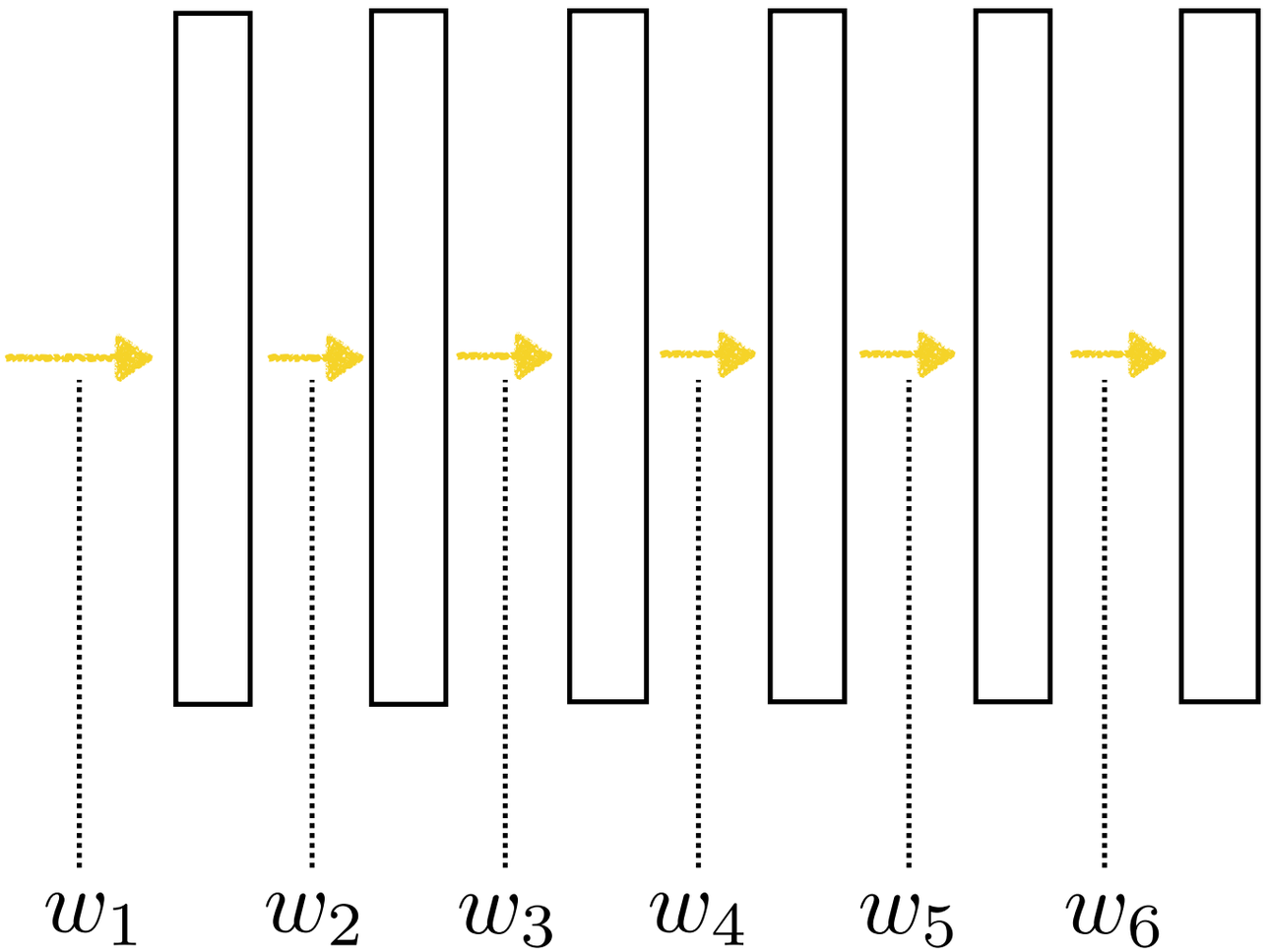
Learning with deep nets

Learned

$\mathbf{z}_i$

"chameleon"

$\mathbf{x}_i$



Loss

$\ell(\mathbf{z}_i, f(\mathbf{x}_i; \mathbf{w}))$

$$\underset{\mathbf{w}}{\text{argmin}} \quad \sum_i \ell(\mathbf{z}_i, f(\mathbf{x}_i; \mathbf{w}))$$

Gradient descent

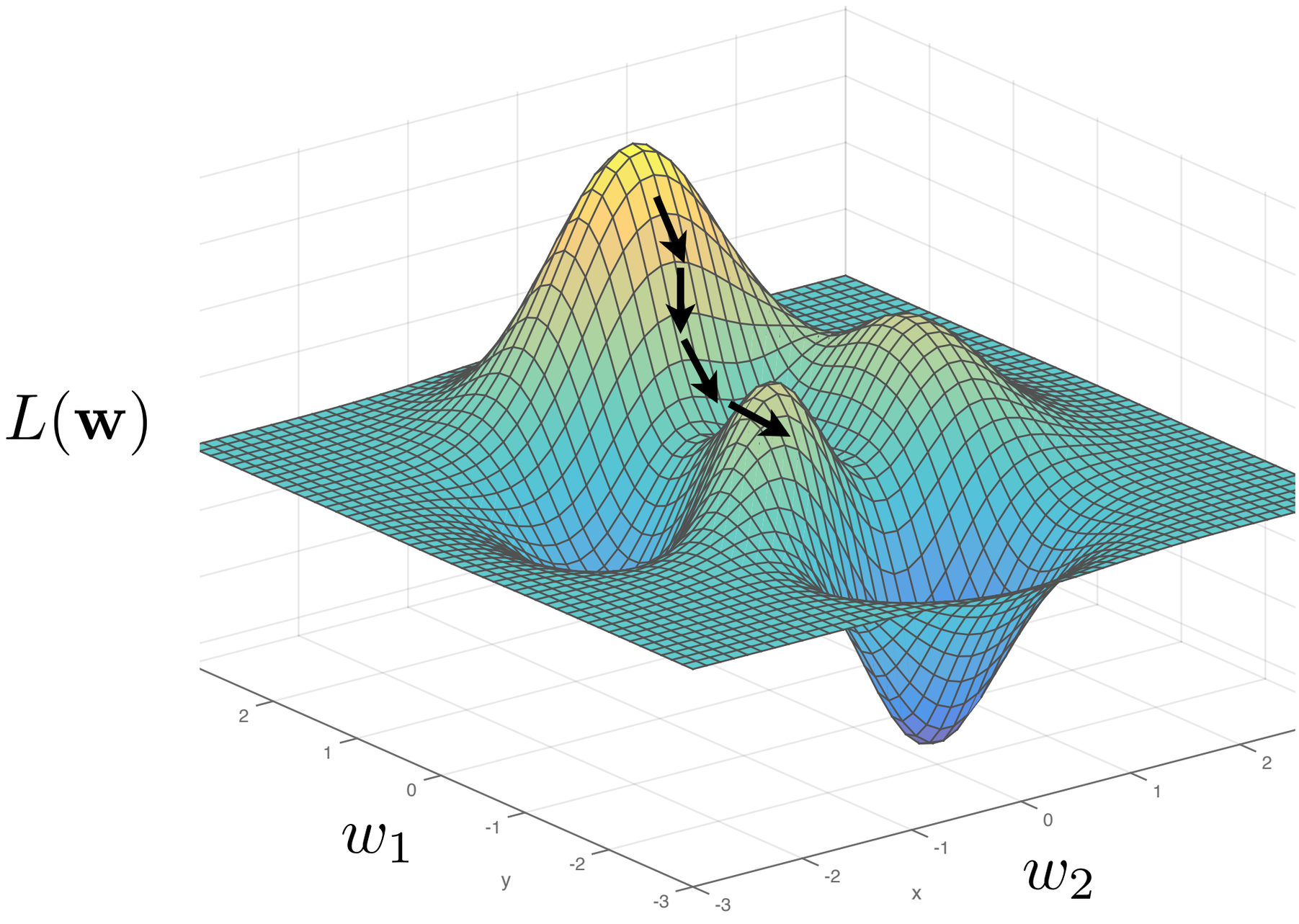
$$\underset{\mathbf{w}}{\operatorname{argmin}} \quad \sum_i \ell(\mathbf{z}_i, f(\mathbf{x}_i; \mathbf{w})) = L(\mathbf{w})$$

One iteration of gradient descent:

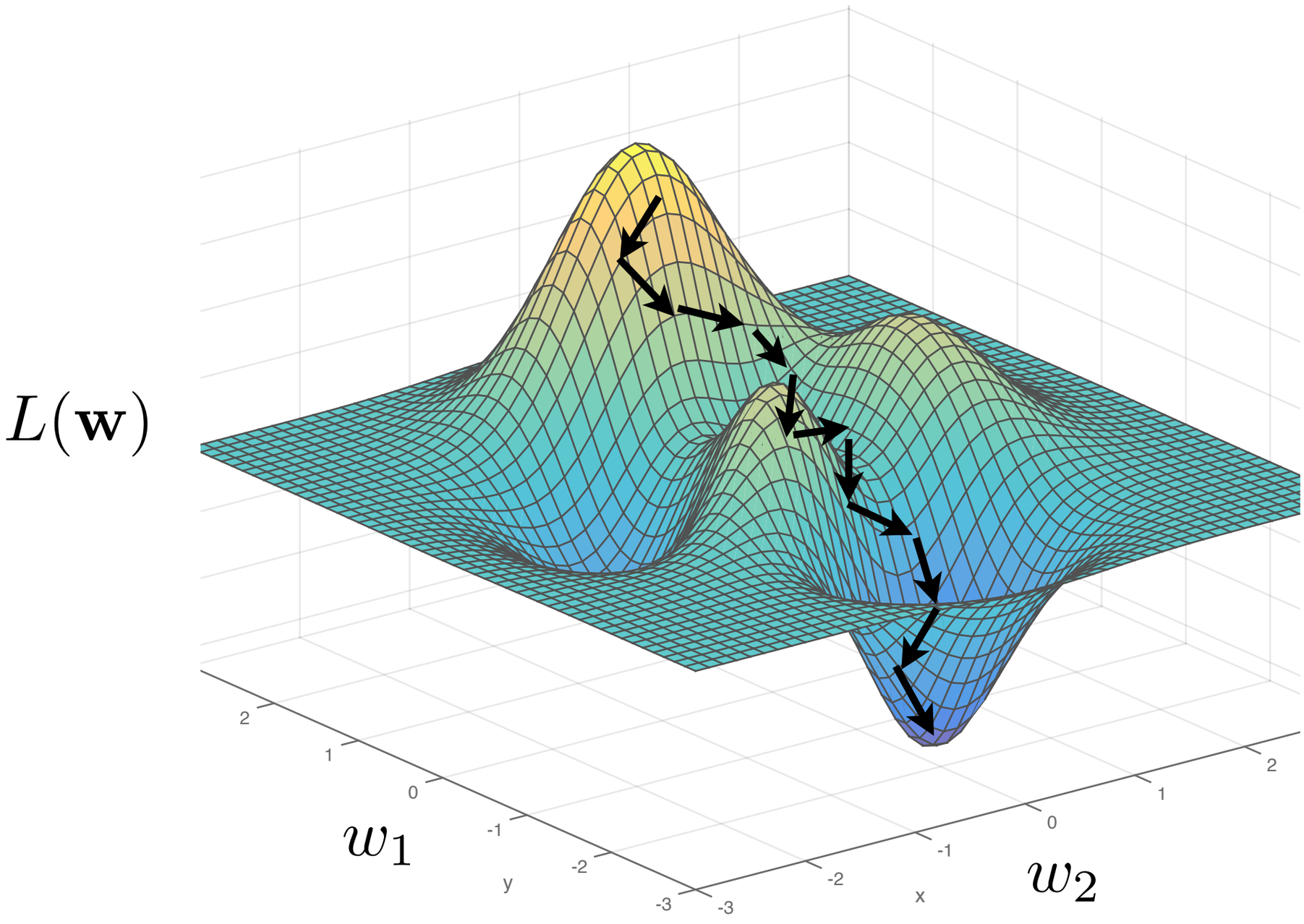
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \frac{\partial L(\mathbf{w^t})}{\partial \mathbf{w}}$$

learning rate

Gradient descent



Stochastic gradient descent



Computing the gradients

$$L(\mathbf{x}, \mathbf{w}) = f_L(\dots f_2(f_1(\mathbf{x}; w_1); w_2) \dots)$$

Chain rule

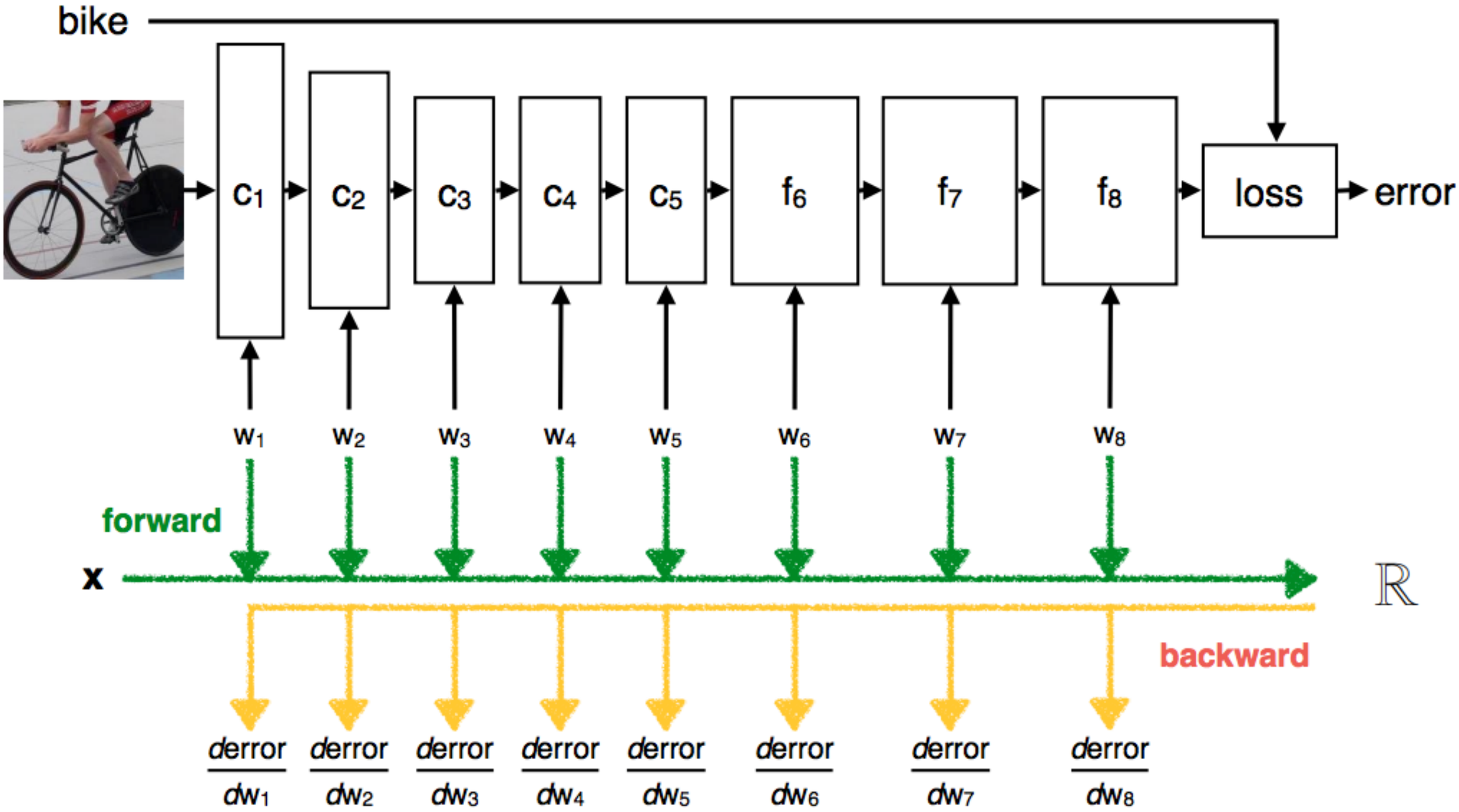
$$\frac{\partial L}{\partial \mathbf{w}_i} = \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial f_{L-2}} \cdots \frac{\partial f_i}{\partial \mathbf{w}_i}$$

This can be computed efficiently using **back-propagation**.

First run the net forward to see what kinds of errors it makes.

Then propagate errors back, using the chain rule, to tell each layer how to adjust its weights to reduce the error.

Back-propagation



Slide from Andrea Vedaldi

How to avoid overfitting?

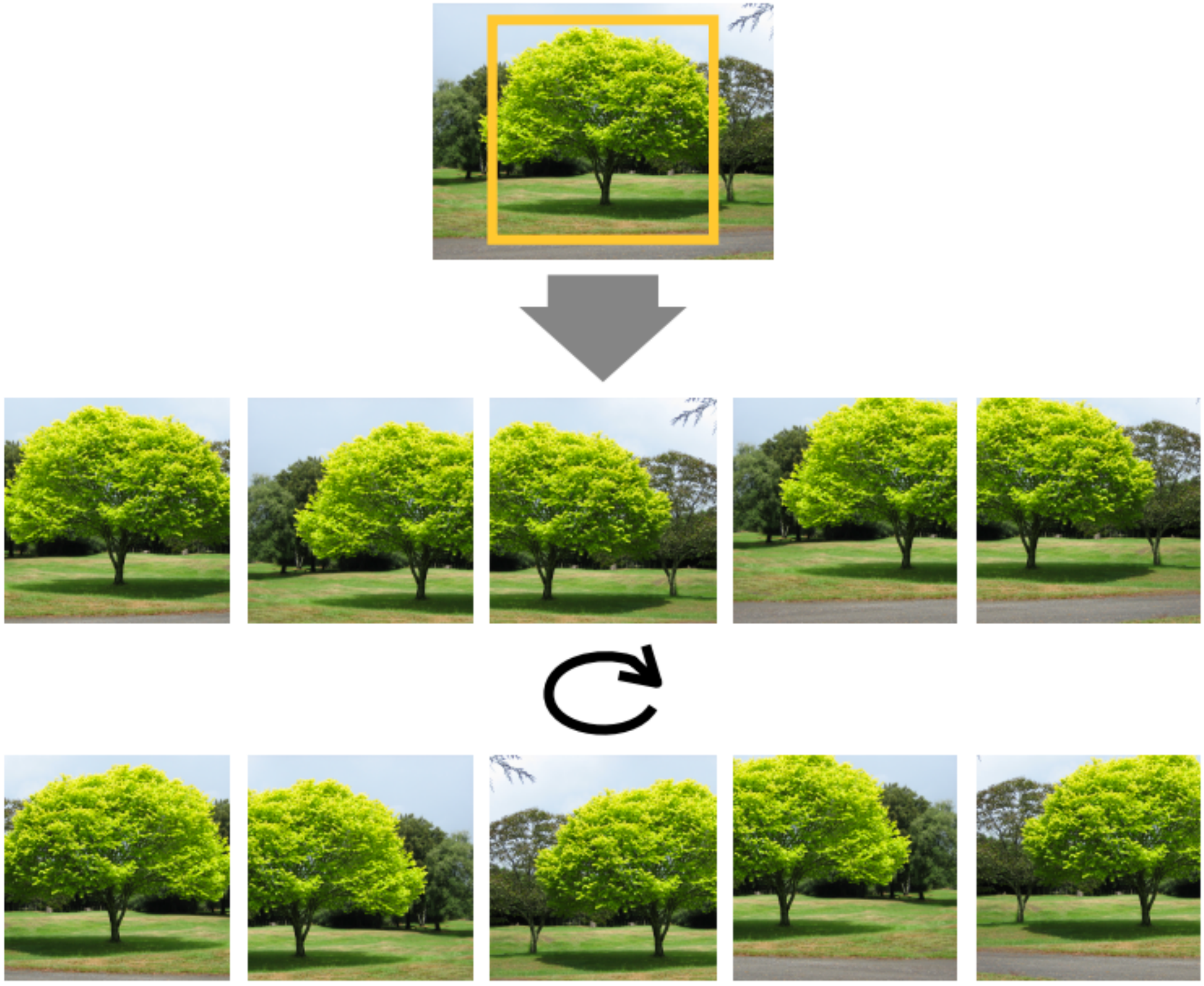
1. Network architecture as a prior
2. Data augmentation
3. Dropout

Network architecture as a prior

Convolutional nets use the prior that stuff in the world does not change identity as it translates.

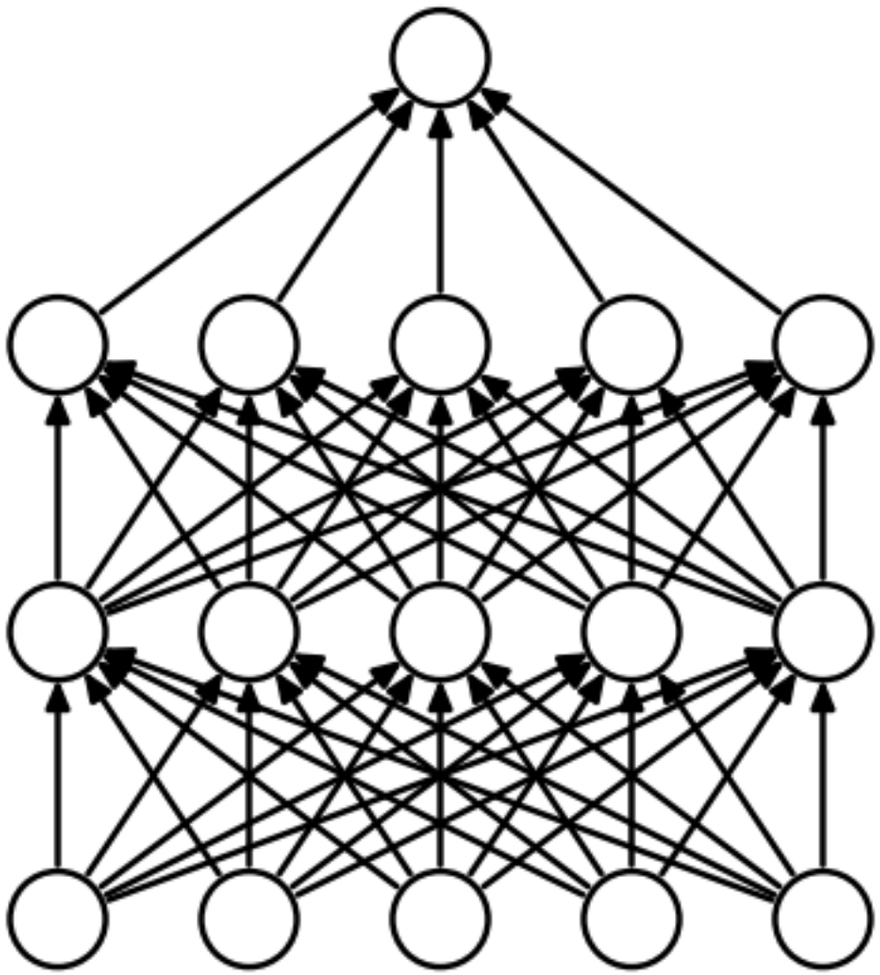
Small layers require that the representation be parsimonious — complexity is penalized.

Augment the training data by adding jittered versions of each image

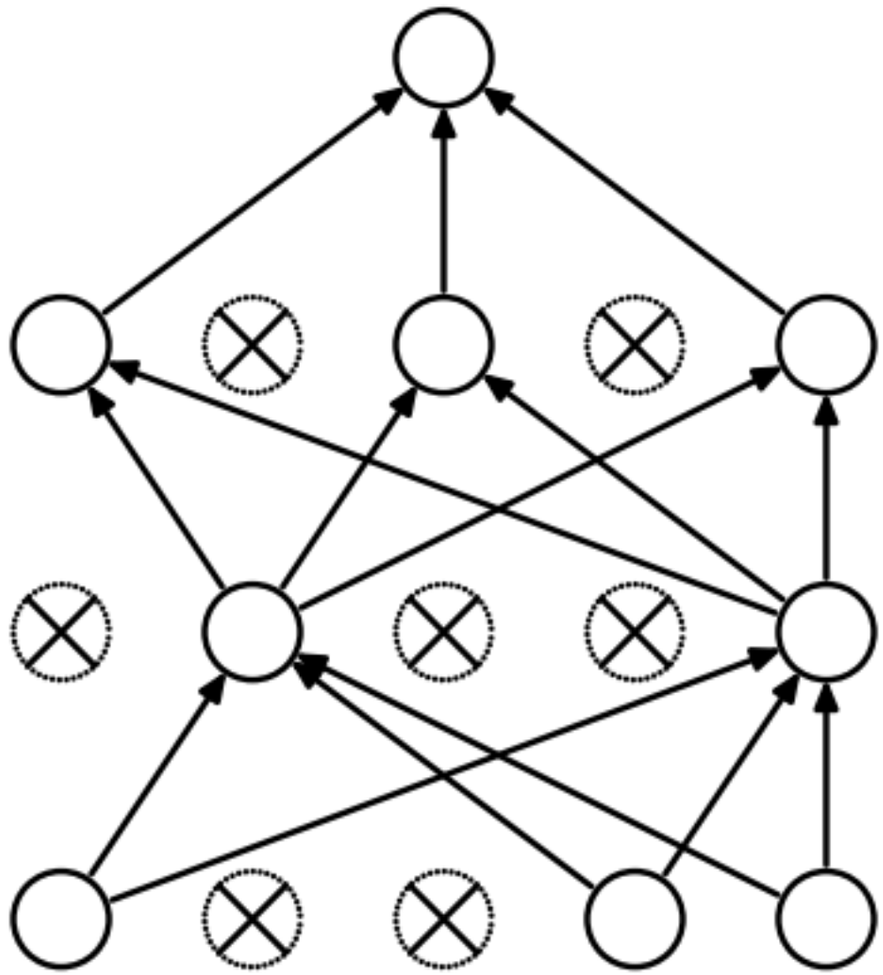


Slide from Andrea Vedaldi

Dropout



(a) Standard Neural Net



(b) After applying dropout.

Srivastava et al., JMLR 2014

Optimal brain damage, Le Cun et al. 1990

How do deep neural nets work?

1. Hierarchy of simple, repeated computations

2. Sift through data by filtering it

3. Build up invariance by pooling alike features

4. Can be learned with vanilla SGD

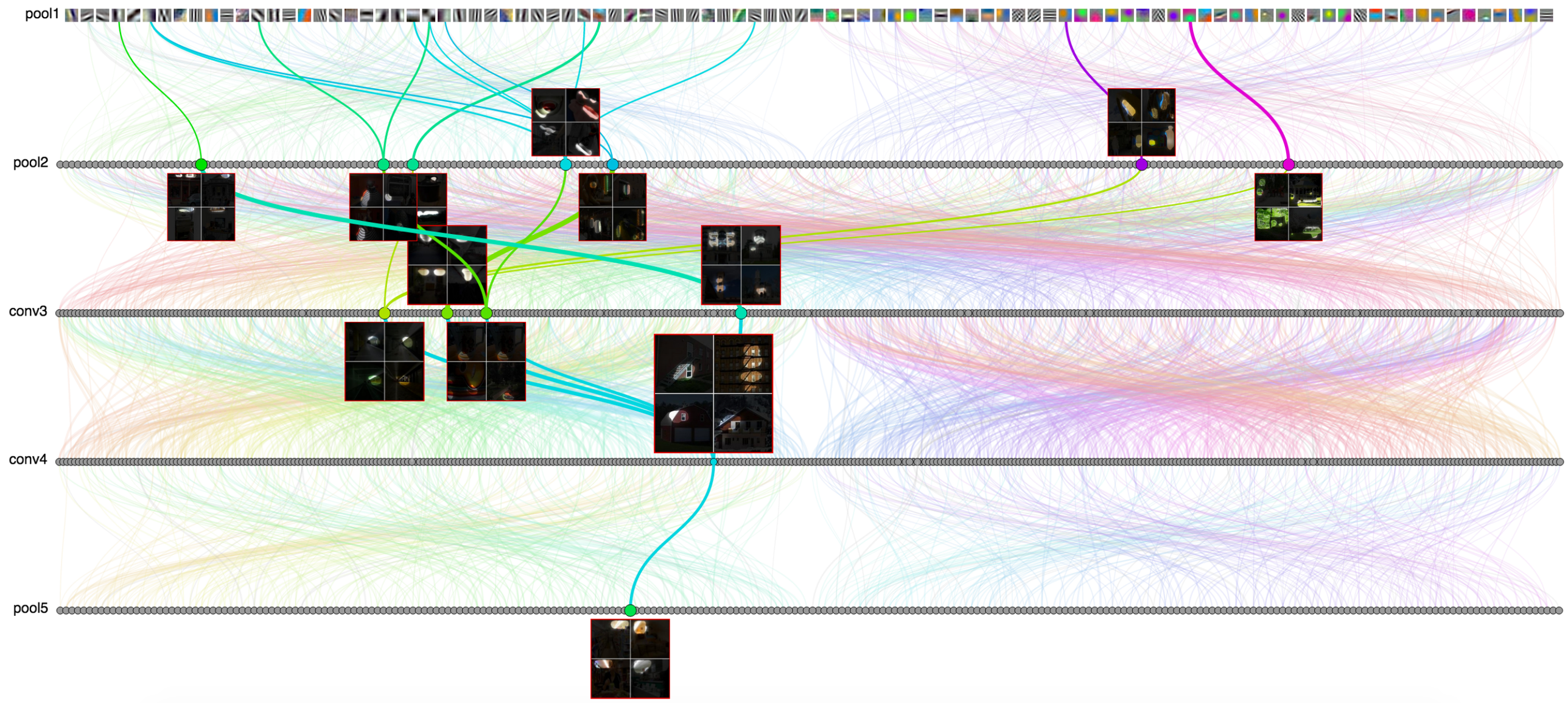
Outline

1. How do they work?

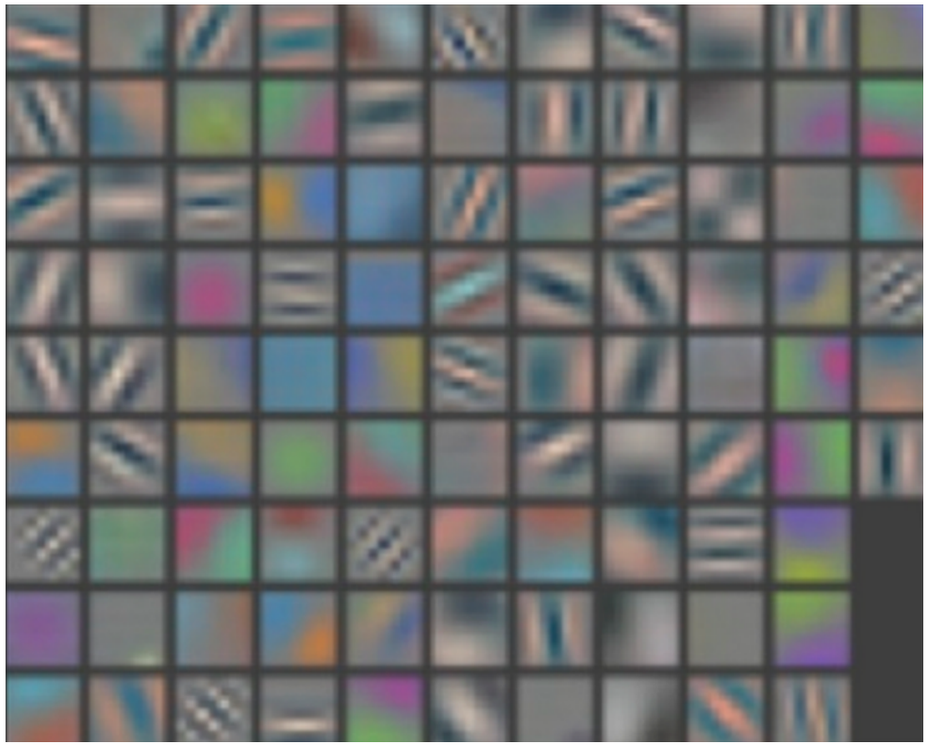
2. **What do they learn?**

3. Practical use

Visualizing what is learned

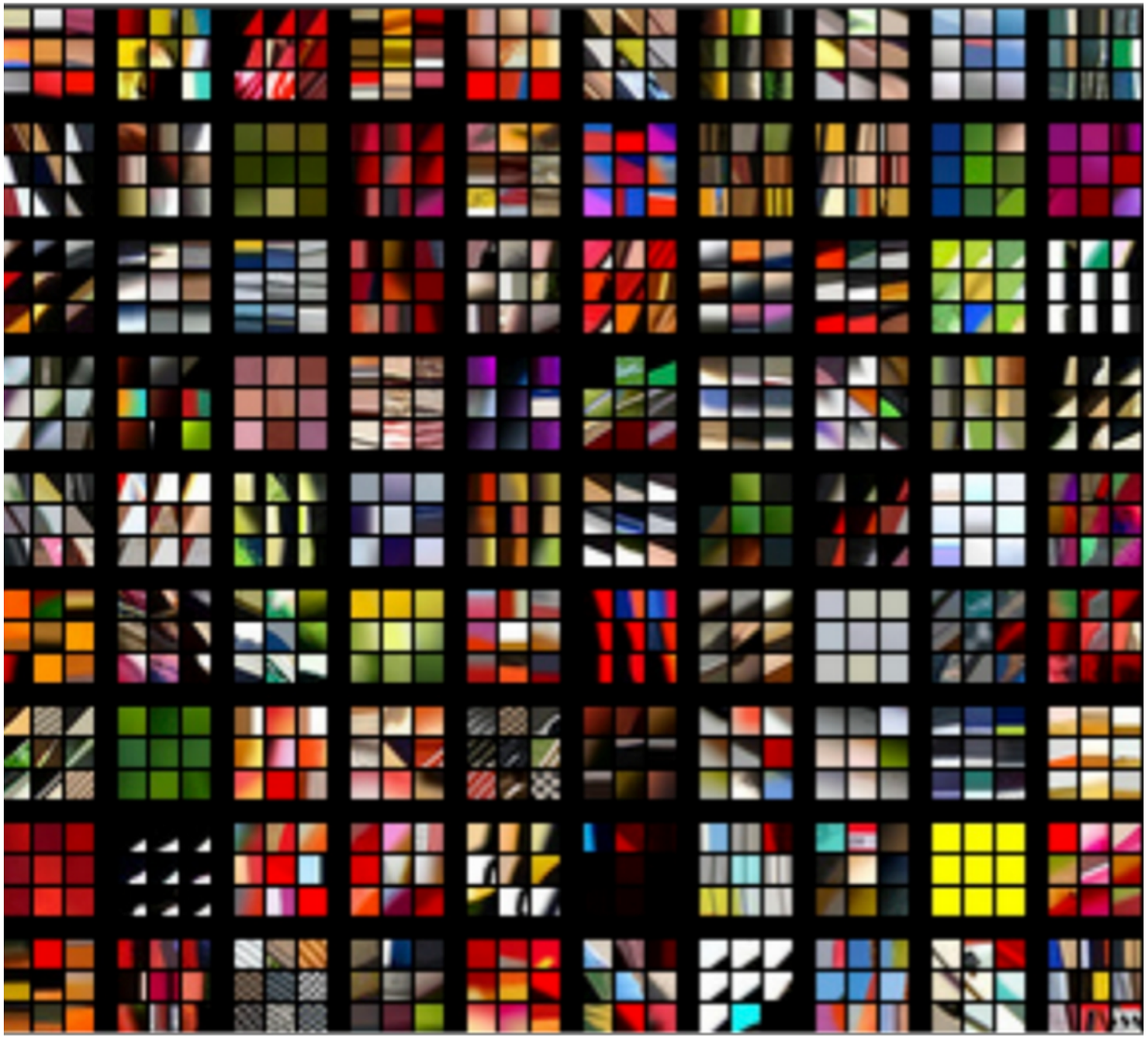


http://people.csail.mit.edu/torralba/research/drawCNN/drawNet.html



1st layer filters

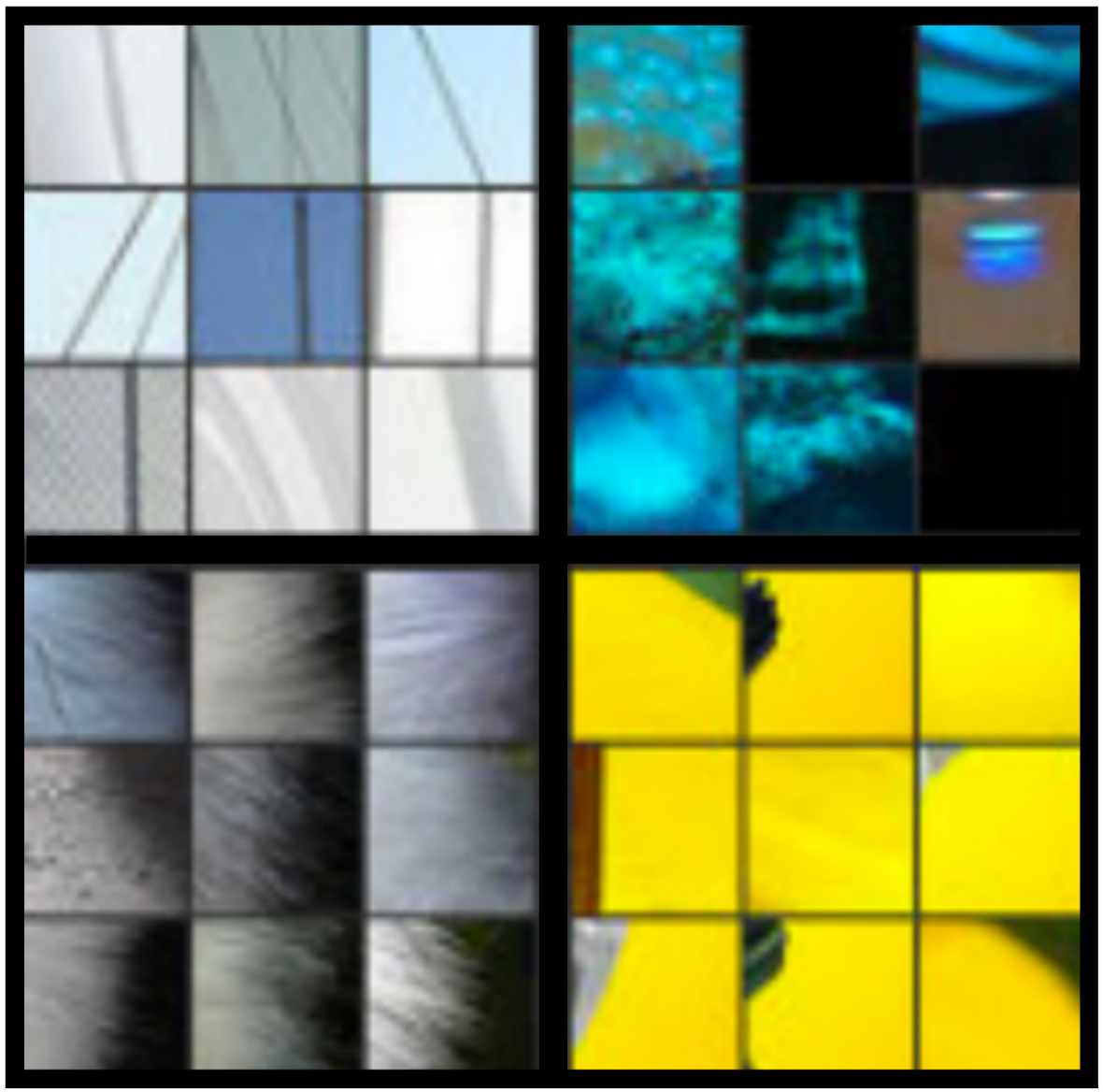
c.f. Olshausen & Field, Nature 1996



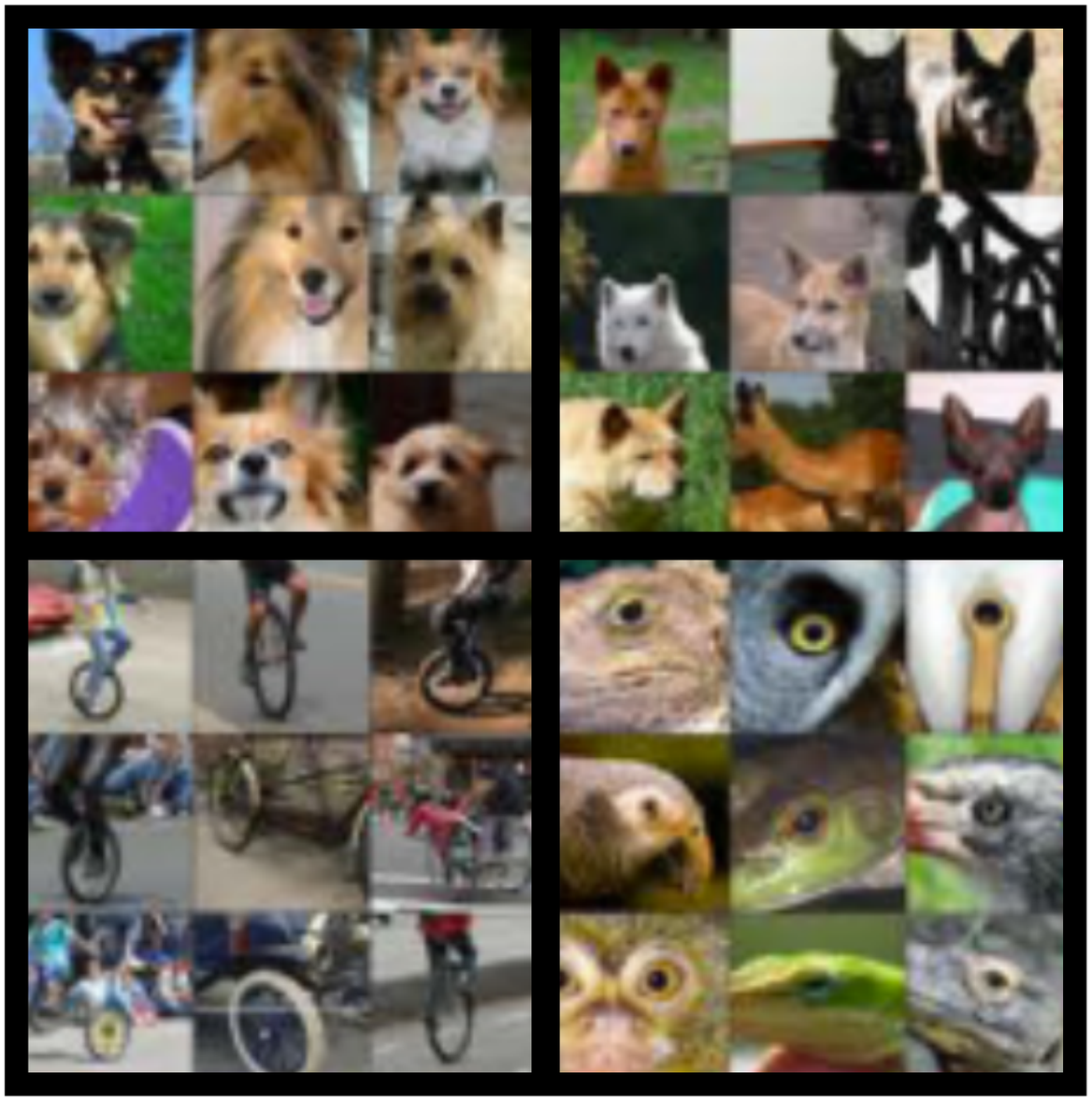
patches that strongly activate first layer filters

Zeiler & Fergus, ECCV 2014

Layer 2



Layer 5



patches that strongly activate neurons on specified layer

Zeiler & Fergus, ECCV 2014

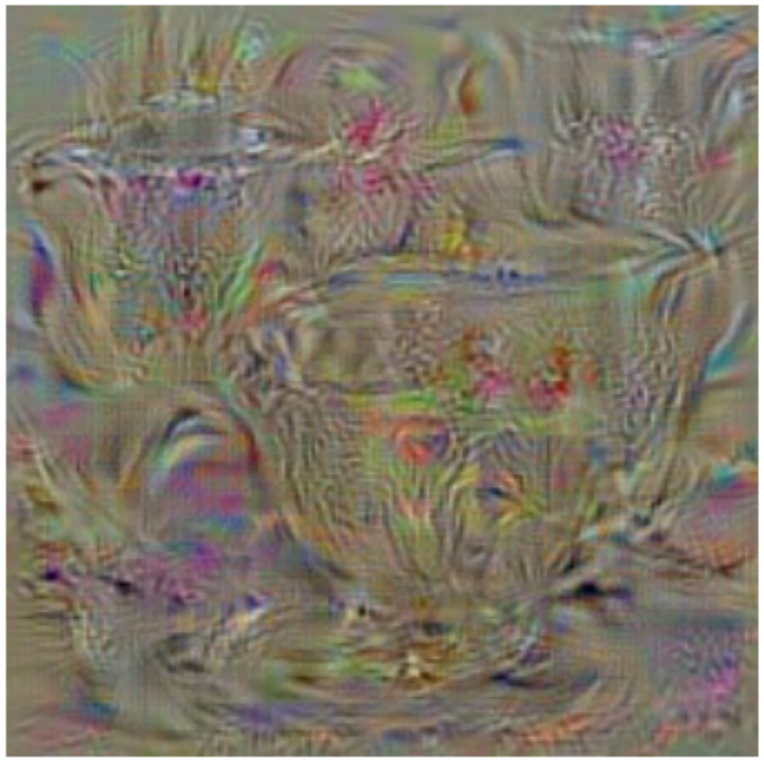
Optimizing the input

How much visual structure does the net actually encode?

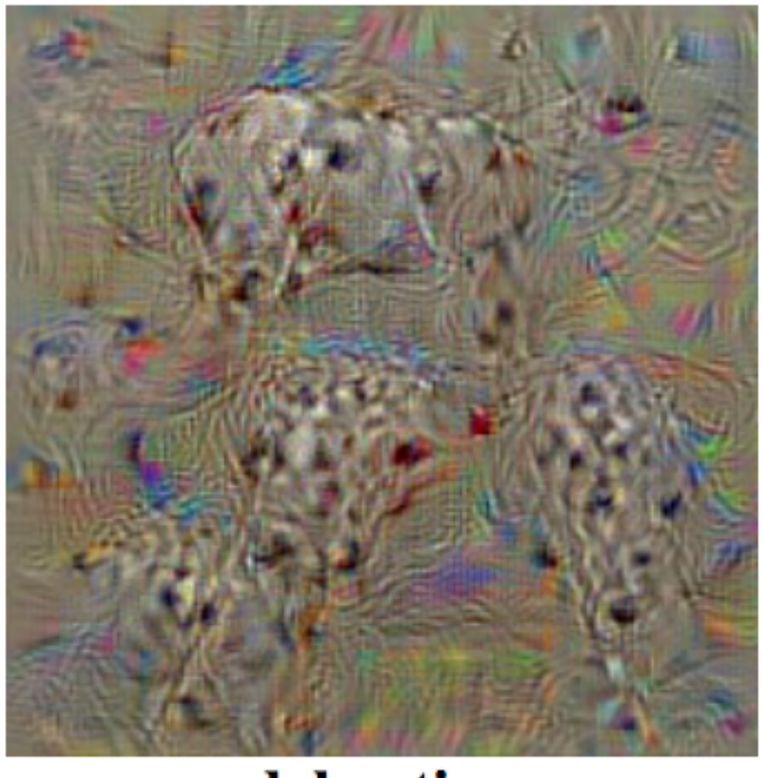
Invert a CNN by finding the stimulus that maximizes the output of a class.

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

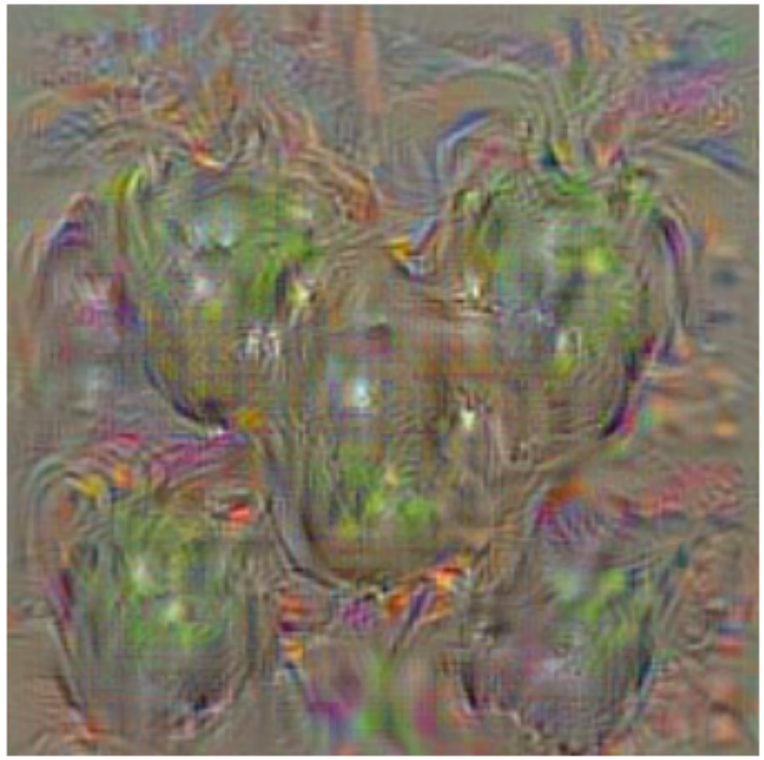
dumbbell



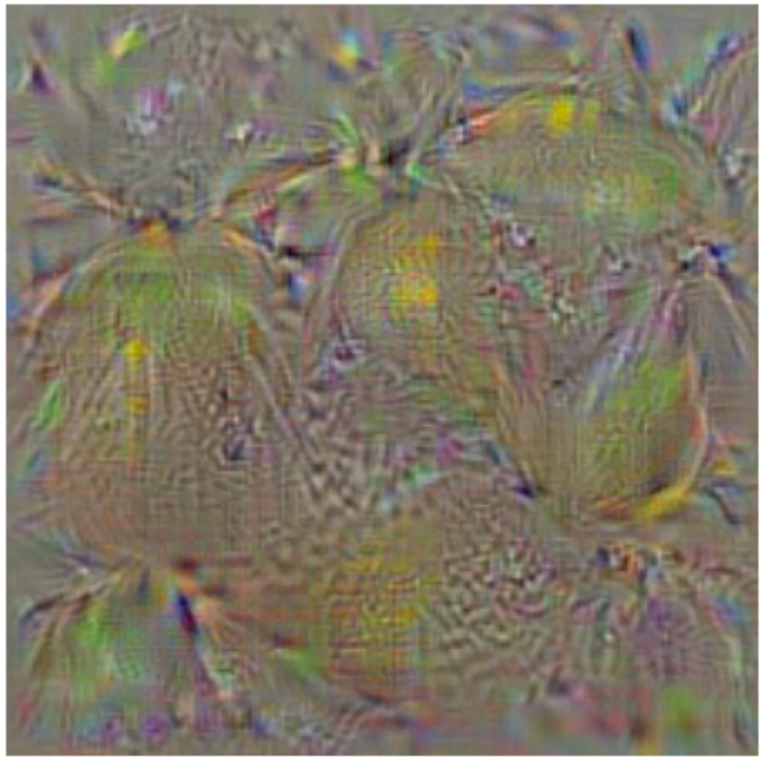
cup



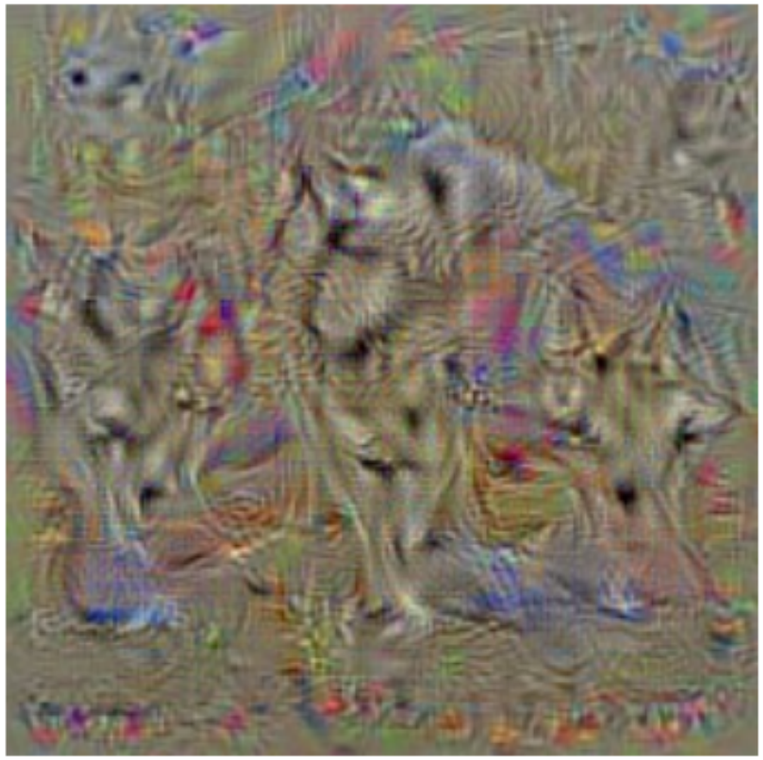
dalmatian



bell pepper

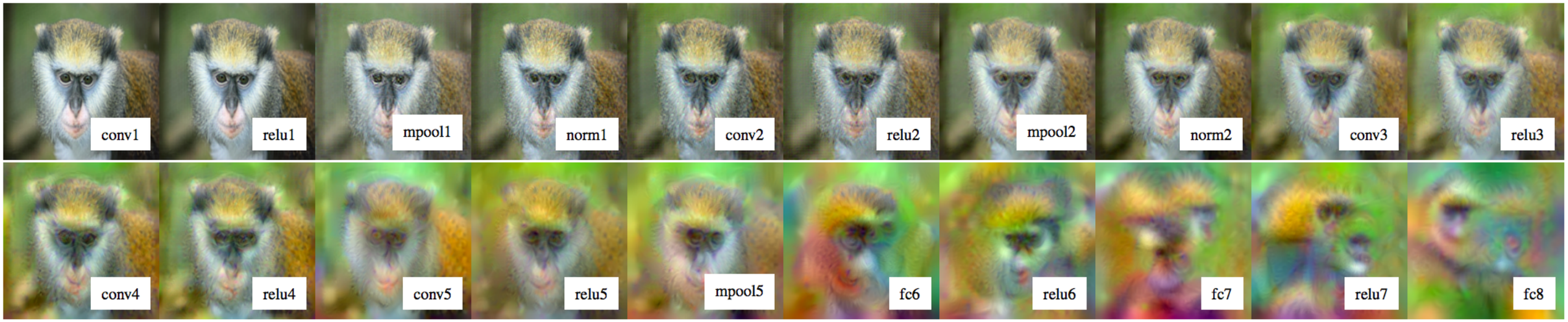


lemon



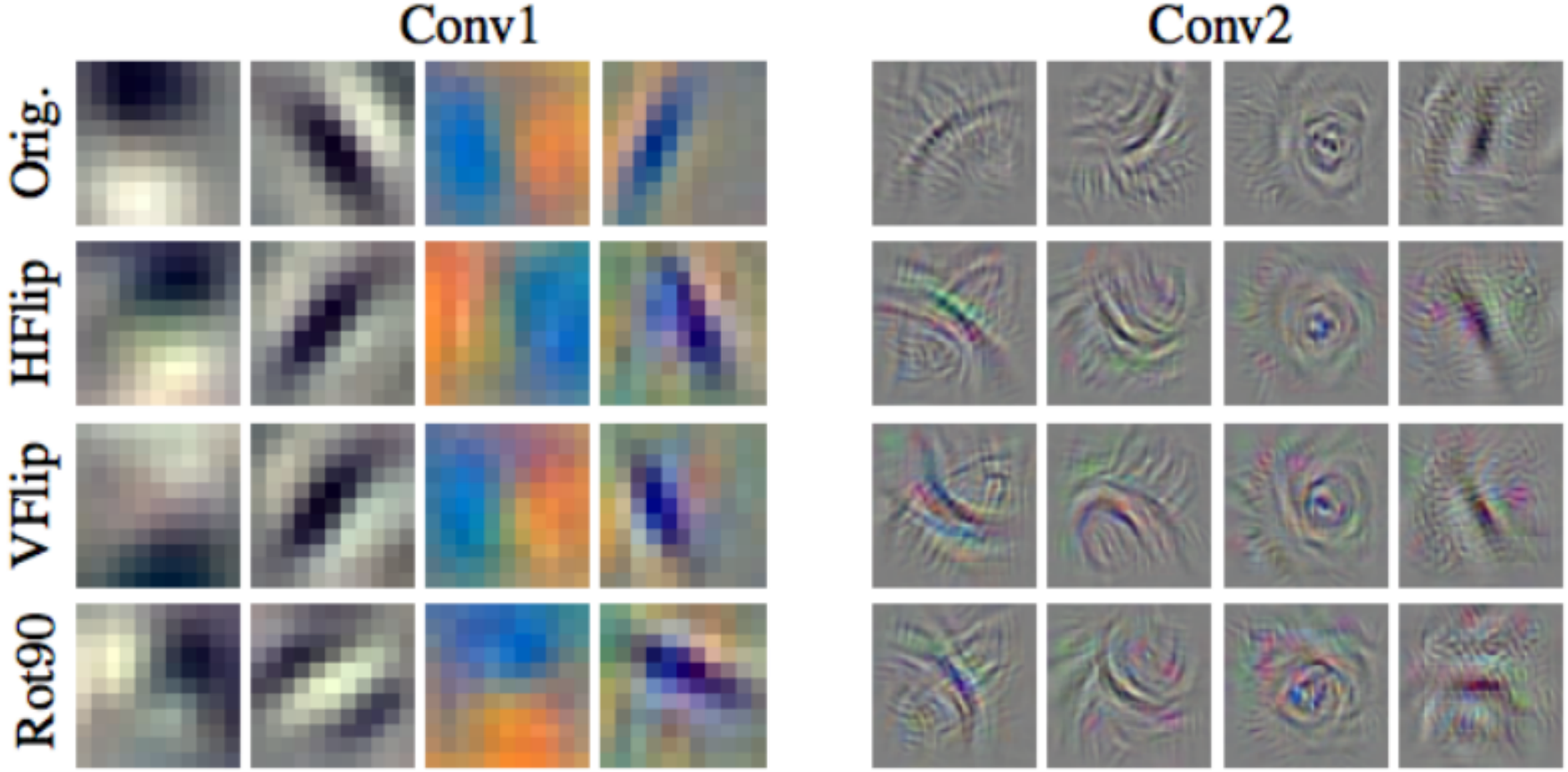
husky

Simonyan et al. ICLR 2014



Mahendran & Vedaldi CVPR 2015

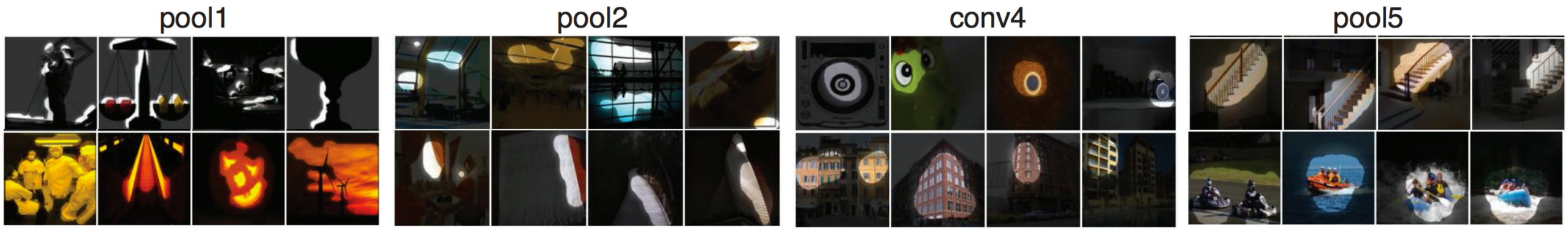
Invariance and Equivariance

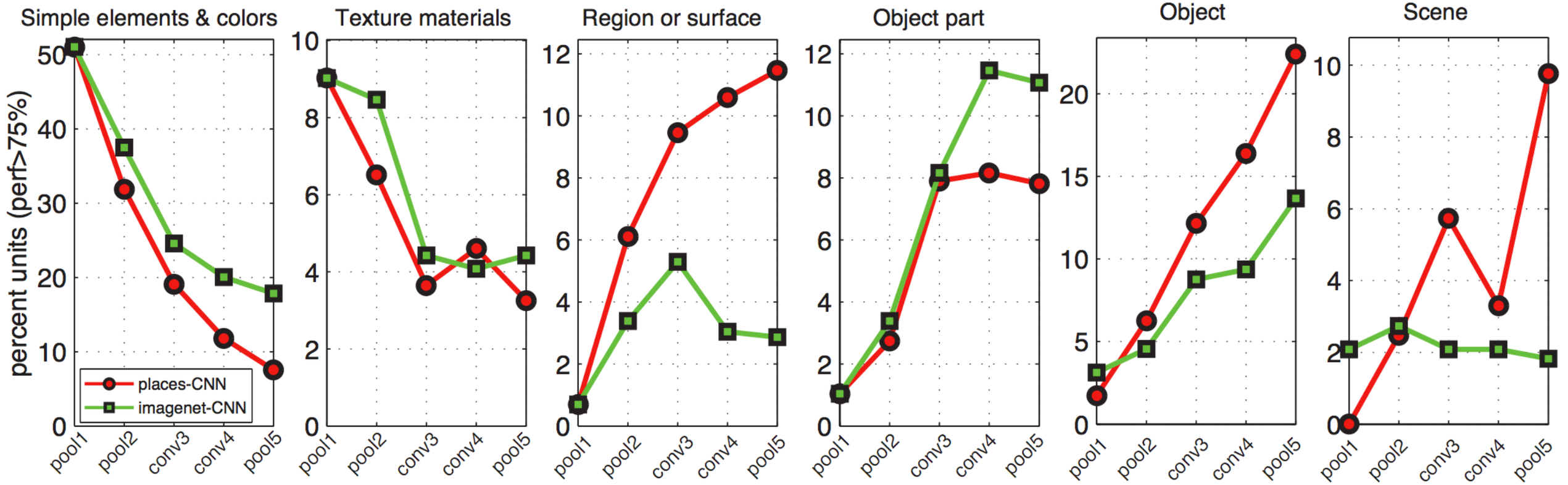


Layer	Horiz. Flip		Vert. Flip		Sc. $2^{-\frac{1}{2}}$		Rot. 90°	
	Num	%	Num	%	Num	%	Num	%
Conv1	52	54.17	53	55.21	95	98.96	42	43.75
Conv2	131	51.17	45	17.58	69	26.95	27	10.55
Conv3	238	61.98	132	34.38	295	76.82	120	31.25
Conv4	343	89.32	124	32.29	378	98.44	101	26.30
Conv5	255	99.61	47	18.36	252	98.44	56	21.88

Lenc & Vedaldi, CVPR 2015

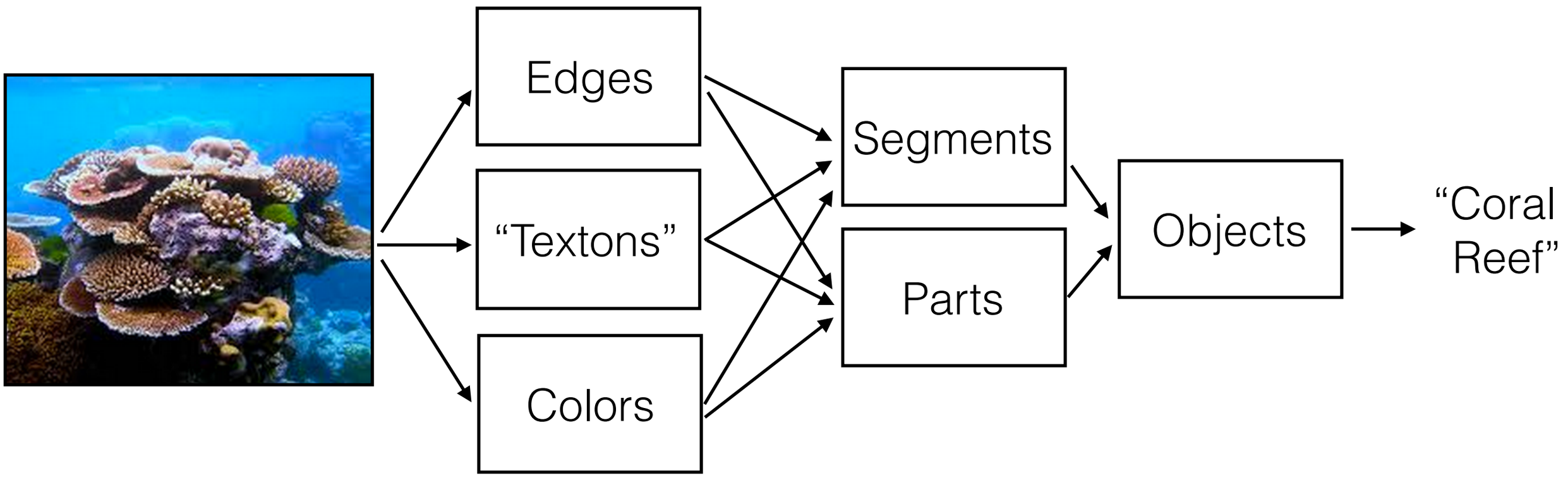
“Object Detectors Emerge in Deep Scene CNNs”





Zhou et al. ICLR 2014

Vision



CNNs appear to learn the classical visual processing hierarchy.

What do they learn?

1. Huge capacity to absorb data

2. Learn something like the classical visual hierarchy

3. Increasing invariance as we go deeper

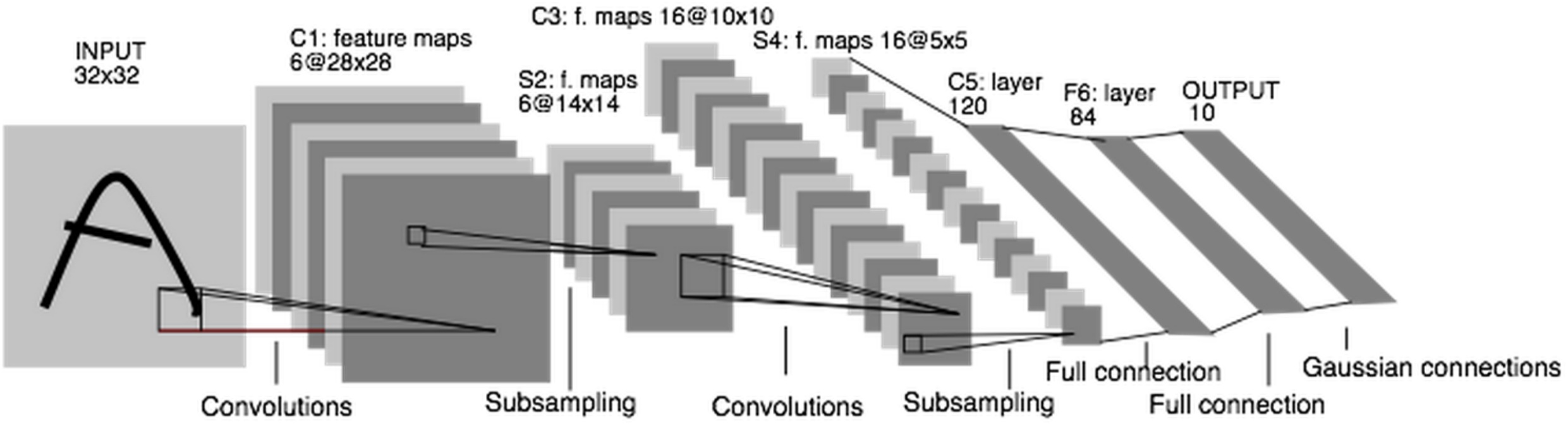
4. But still retain a lot of low-level stuff (e.g., spatial layout) even pretty deep (conv5)

Outline

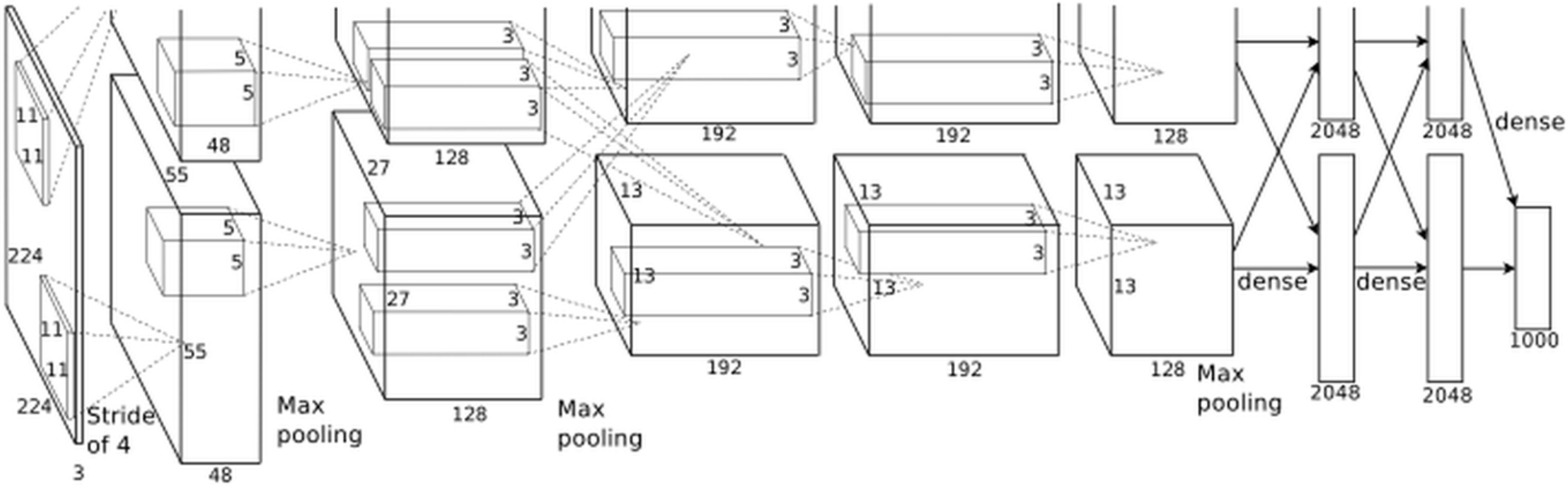
1. Basic theory
2. What do they learn?

3. **Practical use**

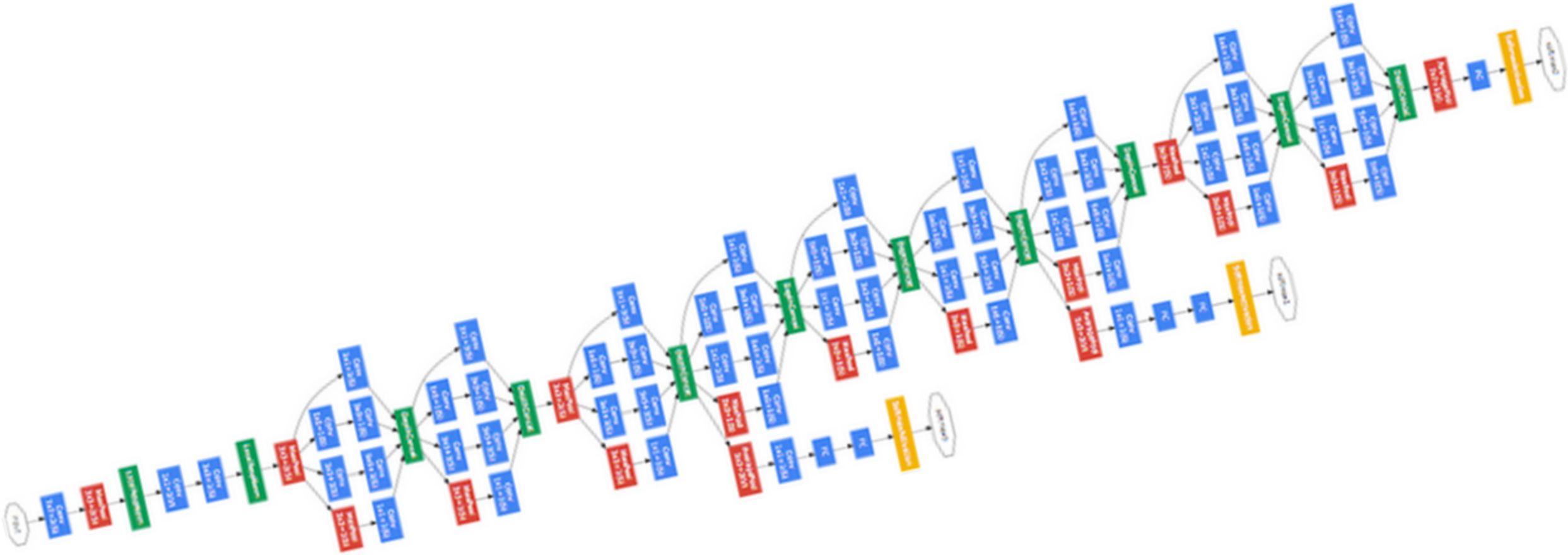
1989 LeNet

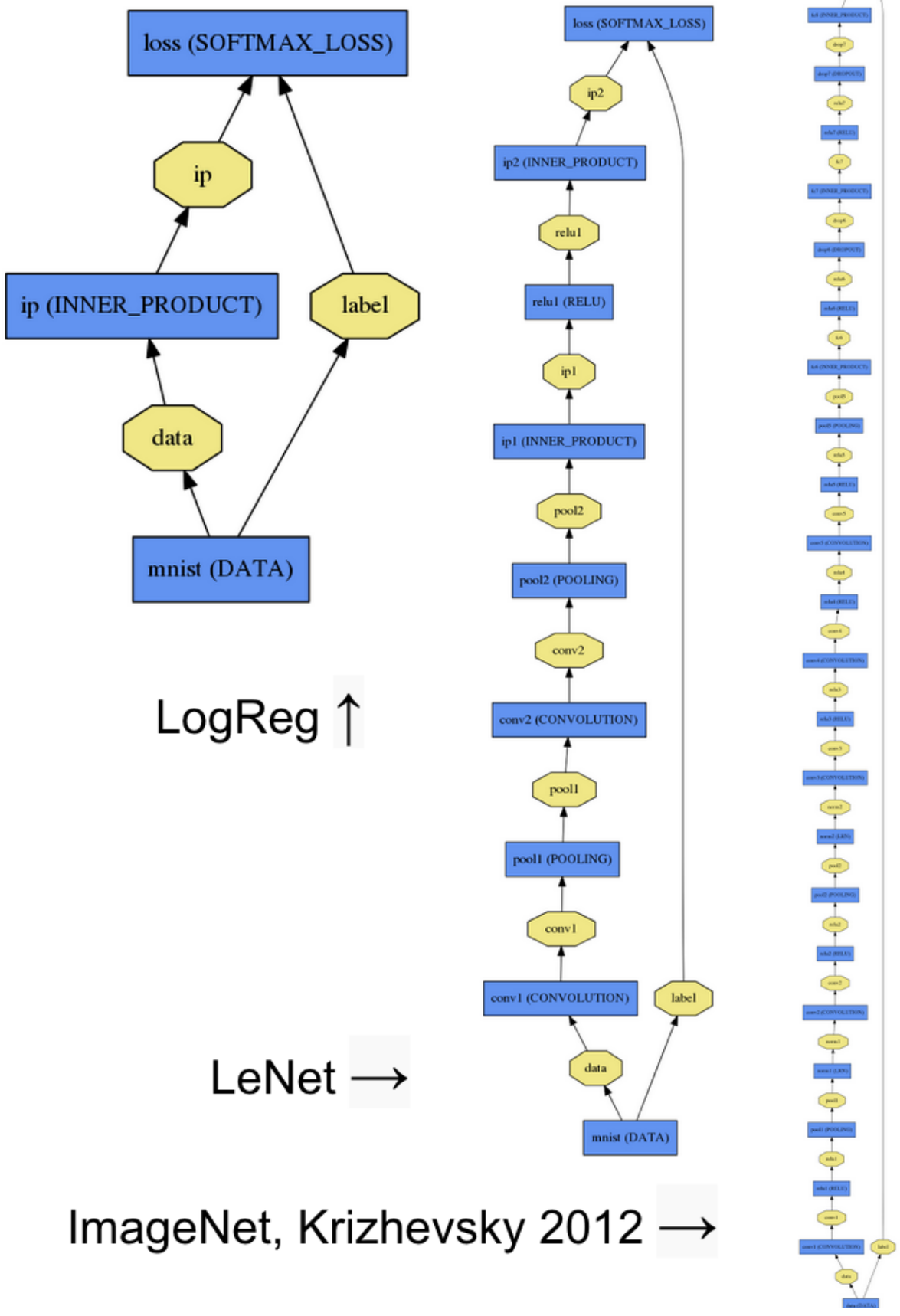

INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

2012 AlexNet



2014 GoogleNet





Slide from caffe.berkeleyvision.org

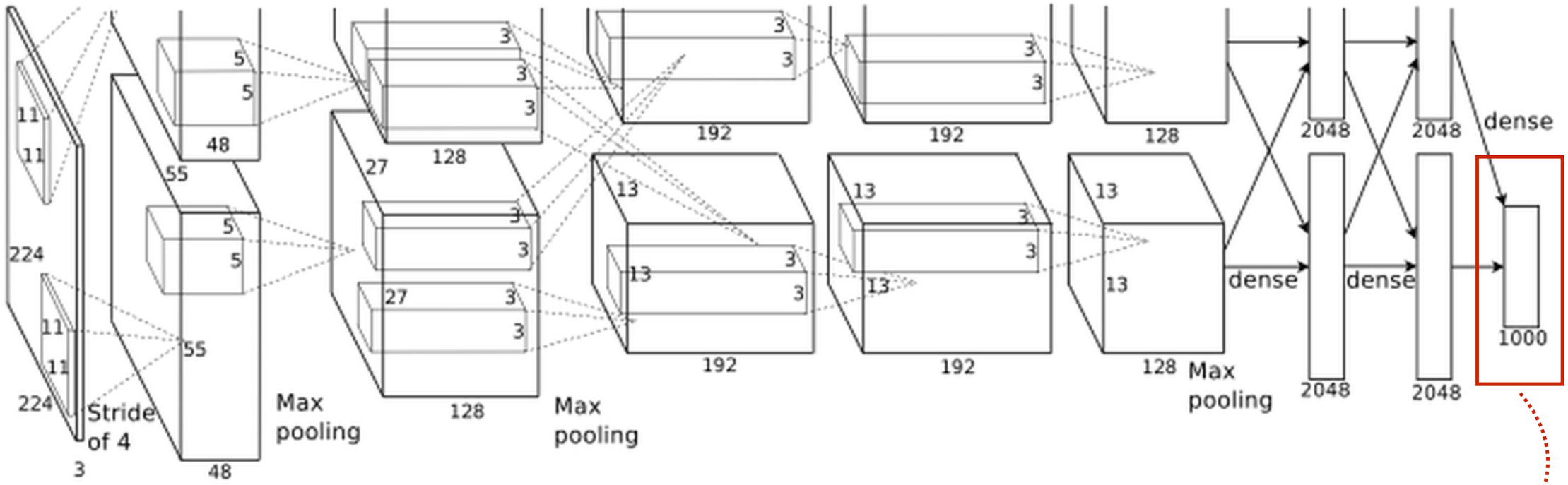
Frameworks

1. **Caffe**
   + Fast and popular
   - Hard to use
   C++ with limited Matlab and Python interfaces
2. **Theano**
   + Symbolic computation and automatic differentiation
   Python
3. **Torch**
   Lua
4. **MatConvNet**
   + Easy to use
   - Might be slightly slower than alternatives
   Matlab

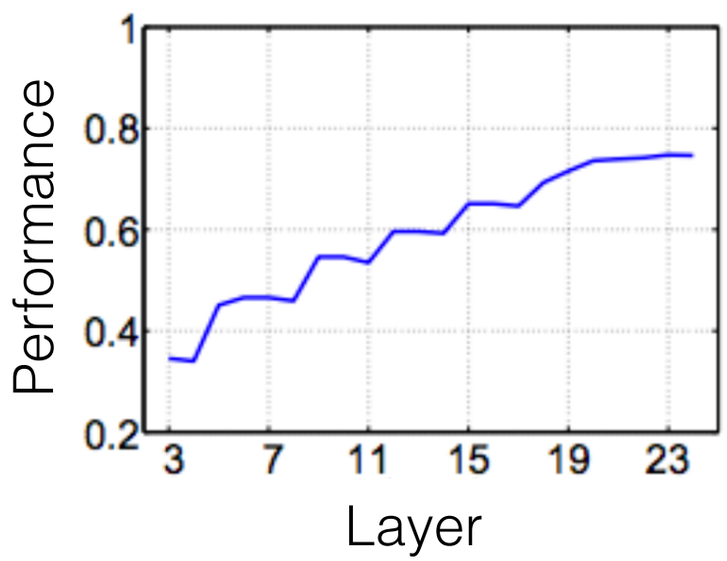
Three levels of usage

1. Use a pre-trained CNN as a feature extractor
2. Fine-tune on limited data
3. Train from scratch on big data

Using CNN features



Use this vector as a generically useful image representation



CNN Features off-the-shelf: an Astounding Baseline for Recognition, Razavian et al. 2014

Finetuning CNNs

1. Download a pre-trained network

2. Replace loss layer with a new one for your specific problem

3. Run SGD

How much data do you need?

State-of-the-art object recognition is trained on ~1 million labeled images.

But you can get somewhat close with ~10k labeled images.

Agrawal et al. 2014

And you can also get very close with unlabeled data.

Doersch et al. 2015
Wang & Gupta 2015
Chen & Gupta 2015

What hardware do you need

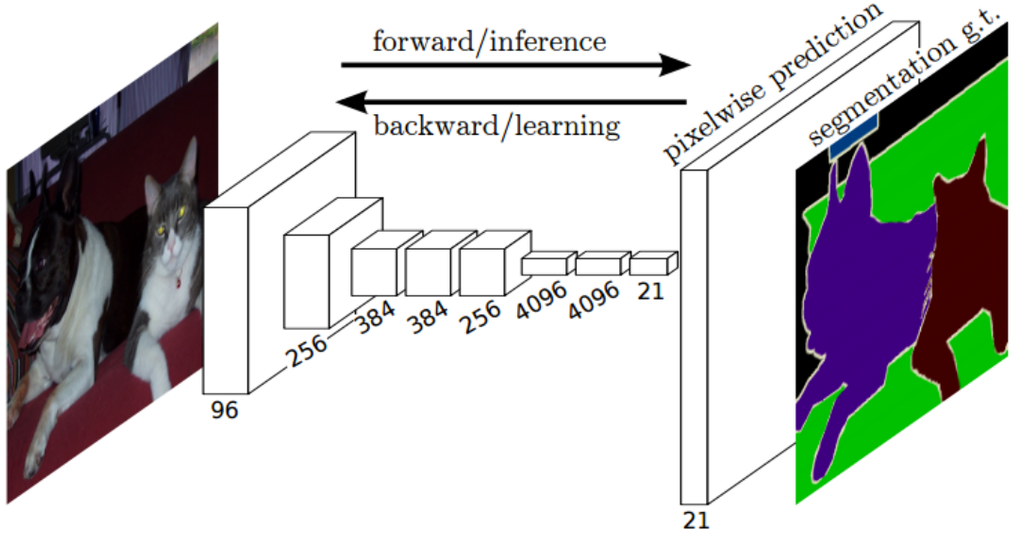
For many problems, a laptop CPU is enough.

For big problems, a single powerful GPU (e.g., Tesla) is enough.

Can run with GPUs on openmind.

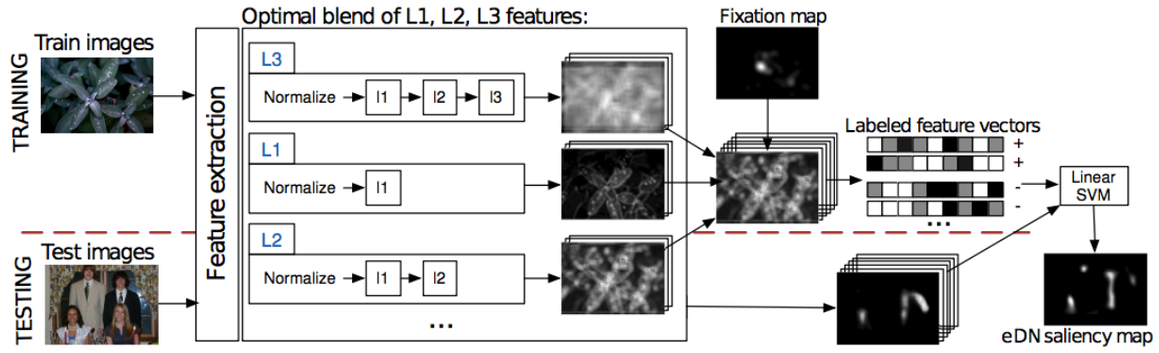
What can you do with them?

Parse images



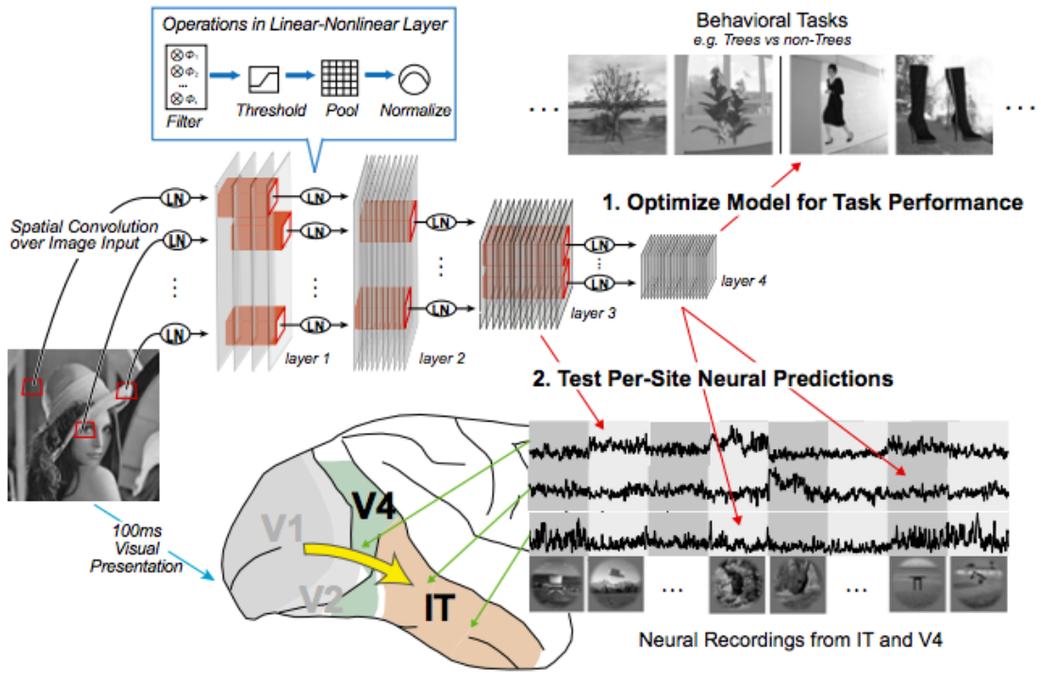
Long et al., CVPR 2015

Model perception



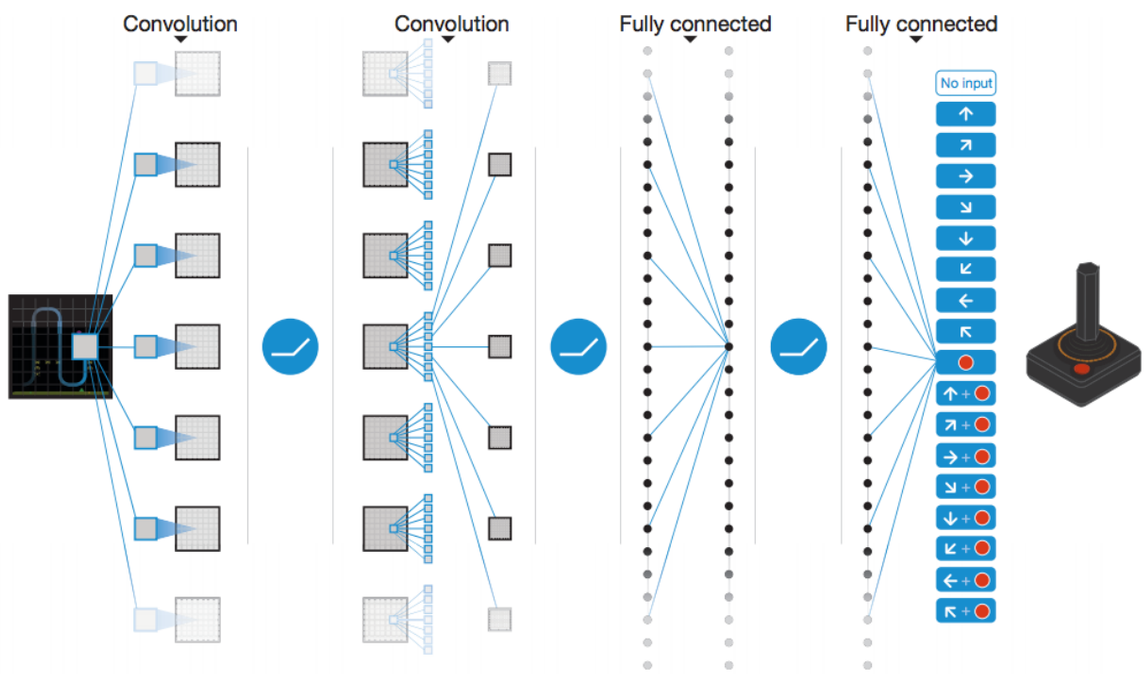
Vig et al., CVPR 2014

Model the brain



Yamins et al., PNAS 2014

Beat humans at Atari games



Mnih et al., Nature 2015

Practical use

1. Slow to learn, very fast at inference time

2. Require a lot of data

3. But can avoid this by starting with a pre-trained network

4. Achieve great performance

5. Many easy to use implementations

Thanks!