EECS 4422/5323 Computer Vision

Unit 8: ConvNets & Learning

This presentation includes slides and figures from
R. Duda et al., R. Fergus, S. Lazebnik and Y. LeCun.

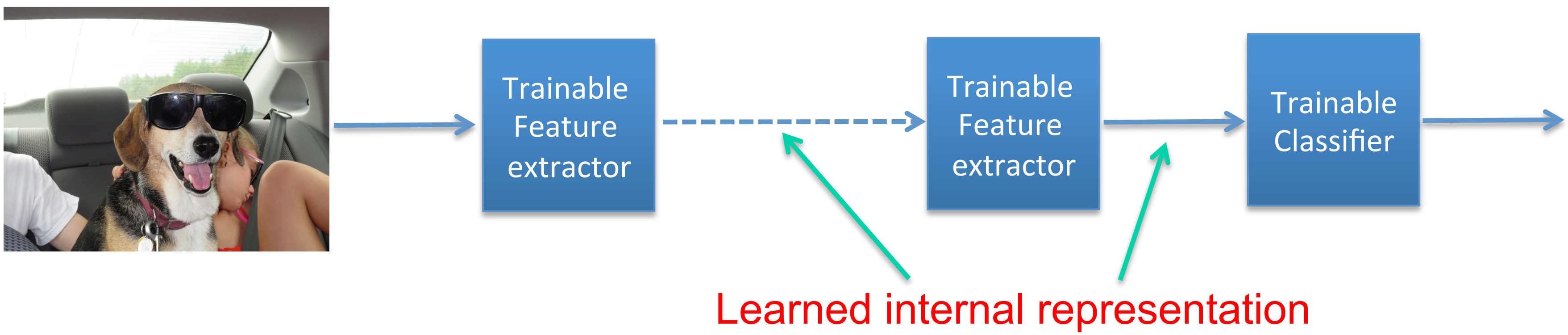
Outline

- Introduction
- Background
- Architecture
- Examples
- Summary

Outline

- **Introduction**
- **Background**
- **Architecture**
- **Examples**
- **Summary**

Introduction: Convolutional networks

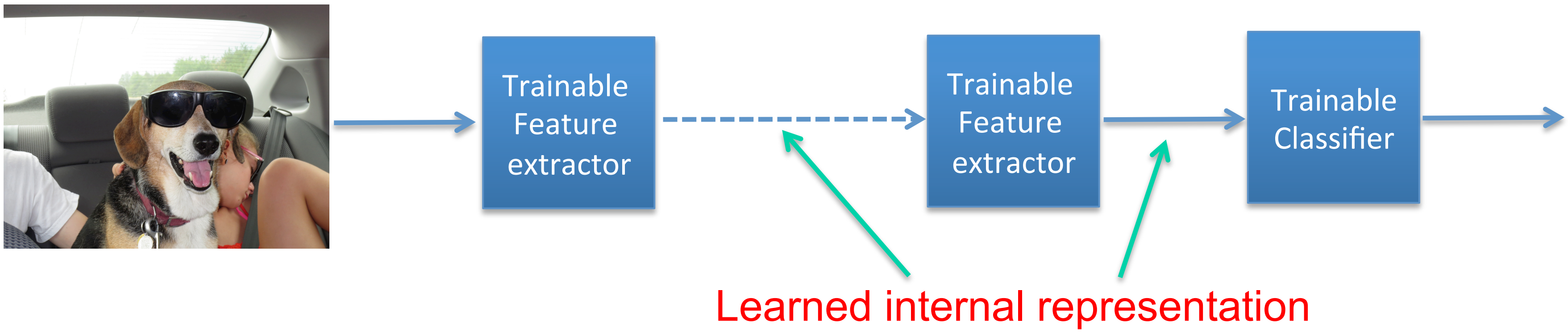


Key ideas

- Build a hierarchy of representations: From primitive features to mid-level abstractions to object identity.
- Invariance to irrelevant aspects of data increases as we go up the layers.
- Efficiency results as far fewer parameters than a fully connected network with same number of elemental units.
- Deep learning: Learn the hierarchy of internal representations.

Introduction: Convolutional networks

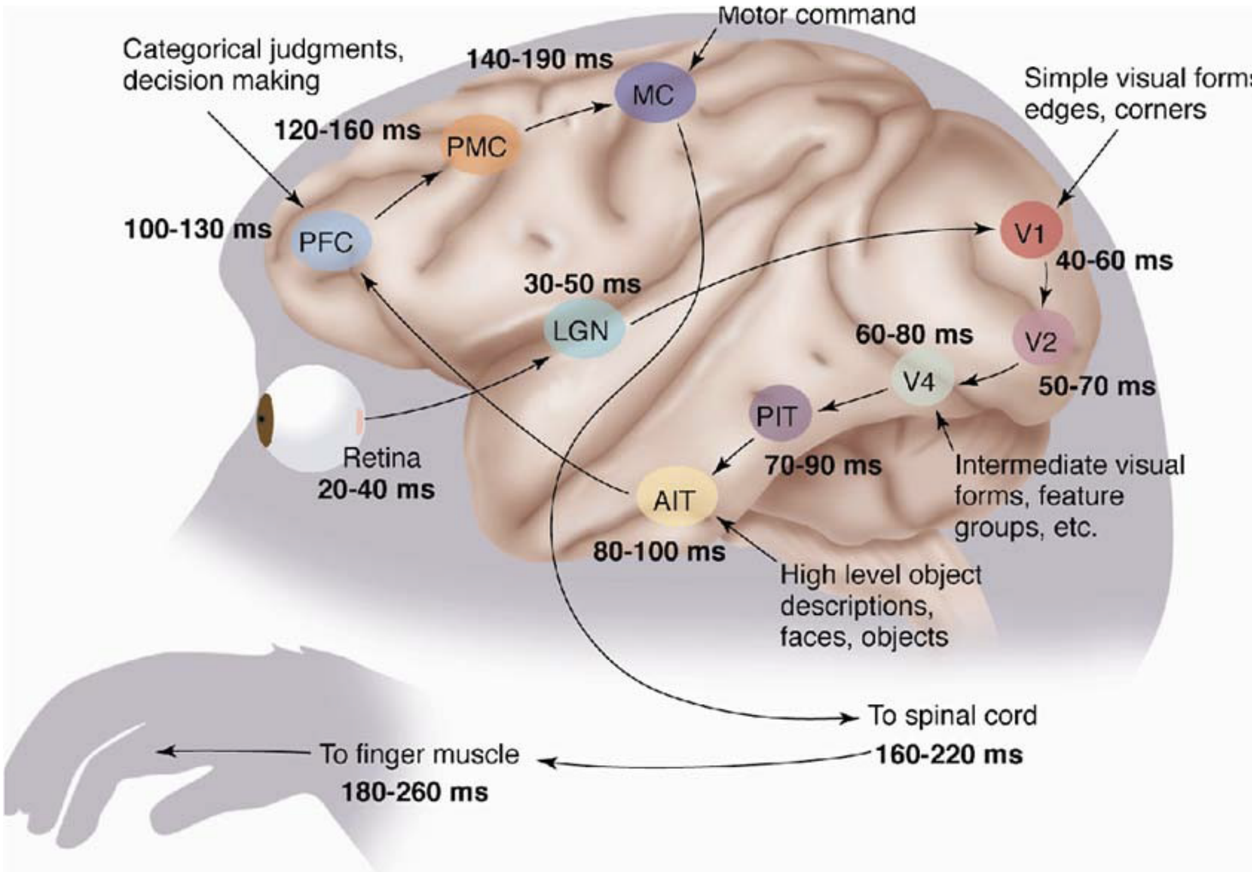
Also called *ConvNets*, *Convolutional Neural Networks* & *CNNs*



Key ideas

- Build a hierarchy of representations: From primitive features to mid-level abstractions to object identity.
- Invariance to irrelevant aspects of data increases as we go up the layers.
- Efficiency results as far fewer parameters than a fully connected network with same number of elemental units.
- Deep learning: Learn the hierarchy of internal representations.

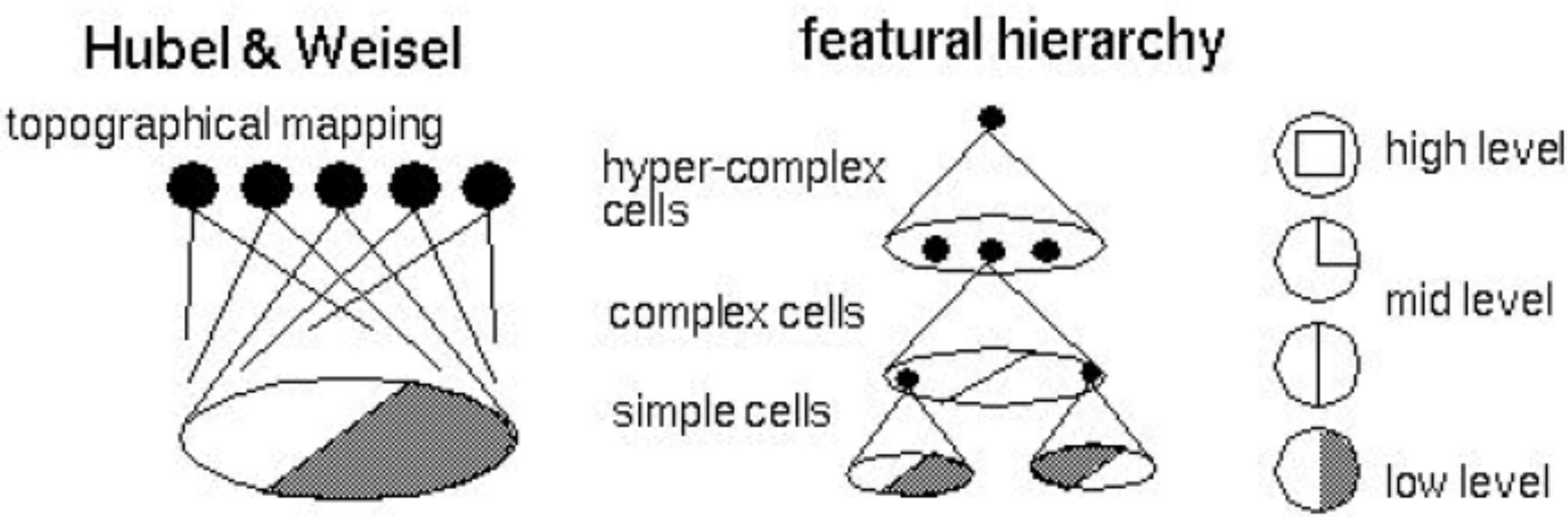
Introduction: Inspiration



Mammalian visual cortex

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

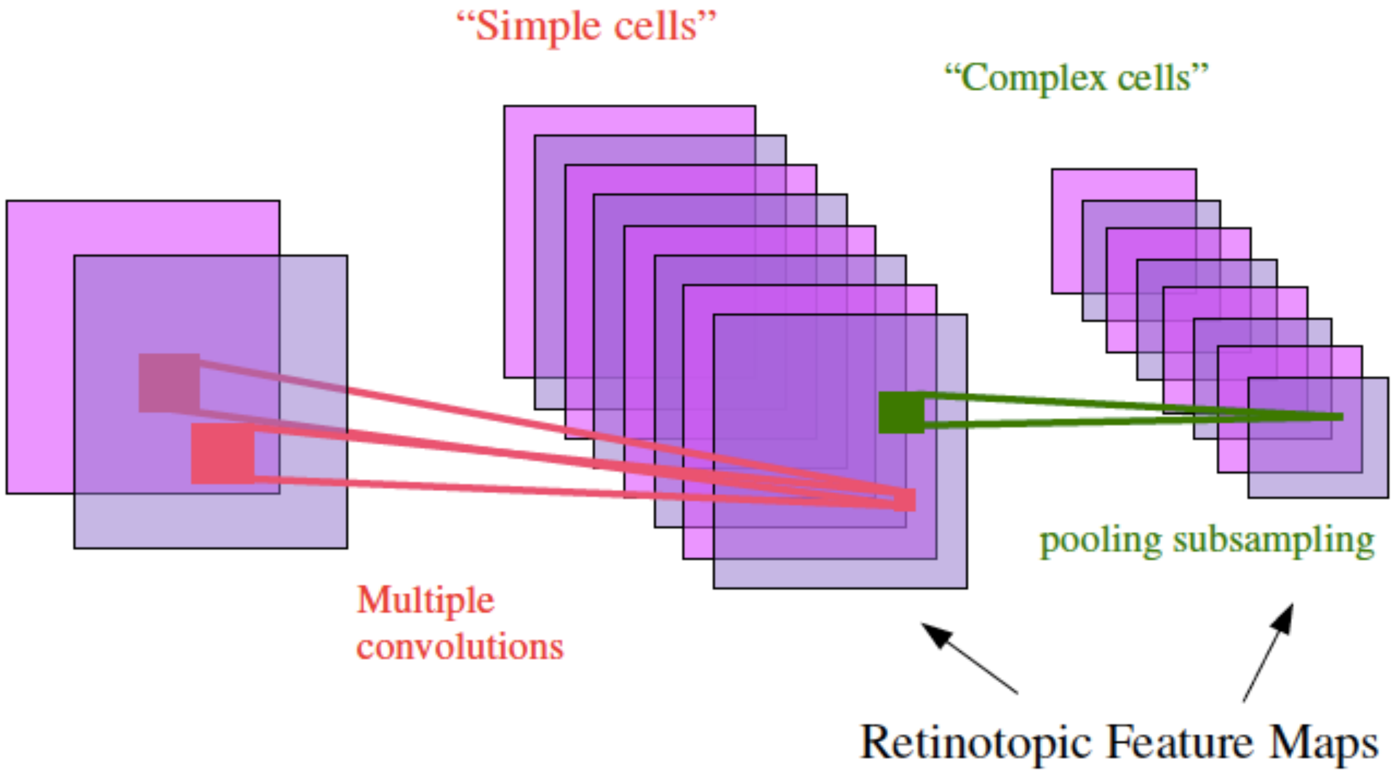
Introduction: Hubel/Wiesel visual cortex model



D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)

- Visual cortex consists of …
  - a hierarchy of simply complex and hypercomplex cells ..
  - with retinotopic organization.
- Based physiological recordings in cat cortex.
- D. Hubel & T. Wiesel (1959) Receptive fields of single neurons in the cat's striate cortex. Journal of Physiology 148 (3), 574-591.
- D. Hubel & T. Wiesel (1962) Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. Journal of Physiology 160 (1), 106-154.

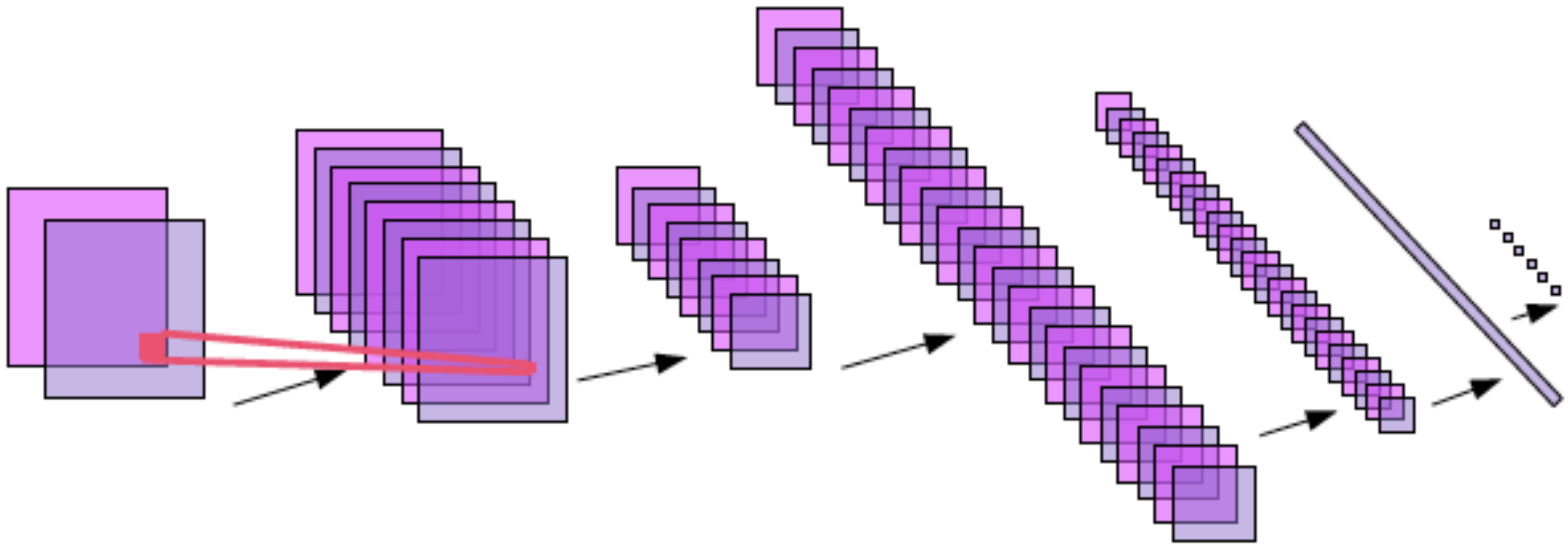
Introduction: An old idea for shift invariance



Hubel & Wiesel features + pooling

- Simple cells detect local features.
- Complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.

Introduction: Repeat



Convolutional network

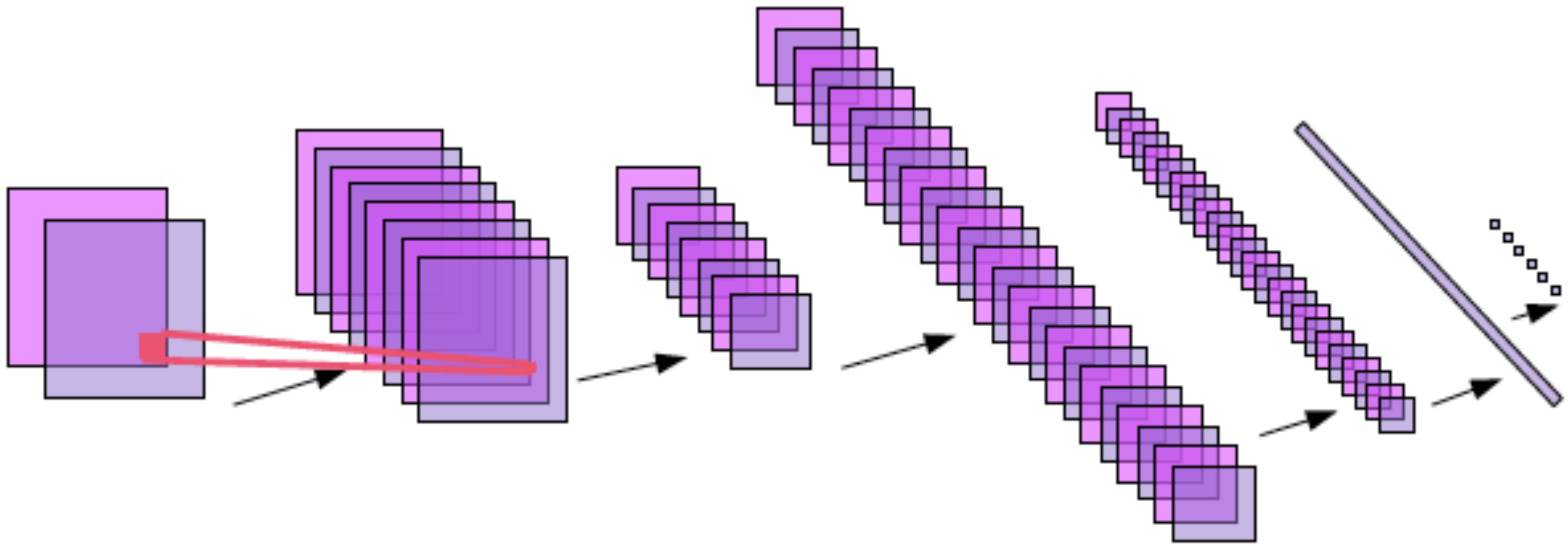
- Hierarchical/multilayer: Features get progressively global, invariant and numerous.
- Dense features: Feature detectors applied everywhere (no interest points).
- Broadly tuned: Toward invariance.
- Complete recognition system: Integrates segmentation, feature extraction and classification.

Introduction: Where do the features come from?

What about learning the features?

- Learn a feature hierarchy all the way from bottom to top.
  - In Vision: Pixels → Edges → Textons →Parts → Objects → Scenes
  - In language: Audio → phonemes → Words → Parts of Speech → Sentences → Narratives
- Each layer extracts features from the output of the previous layer.
- Train all layers jointly, end-to-end to minimize a global loss function.
- Use a gradient based optimization algorithm.

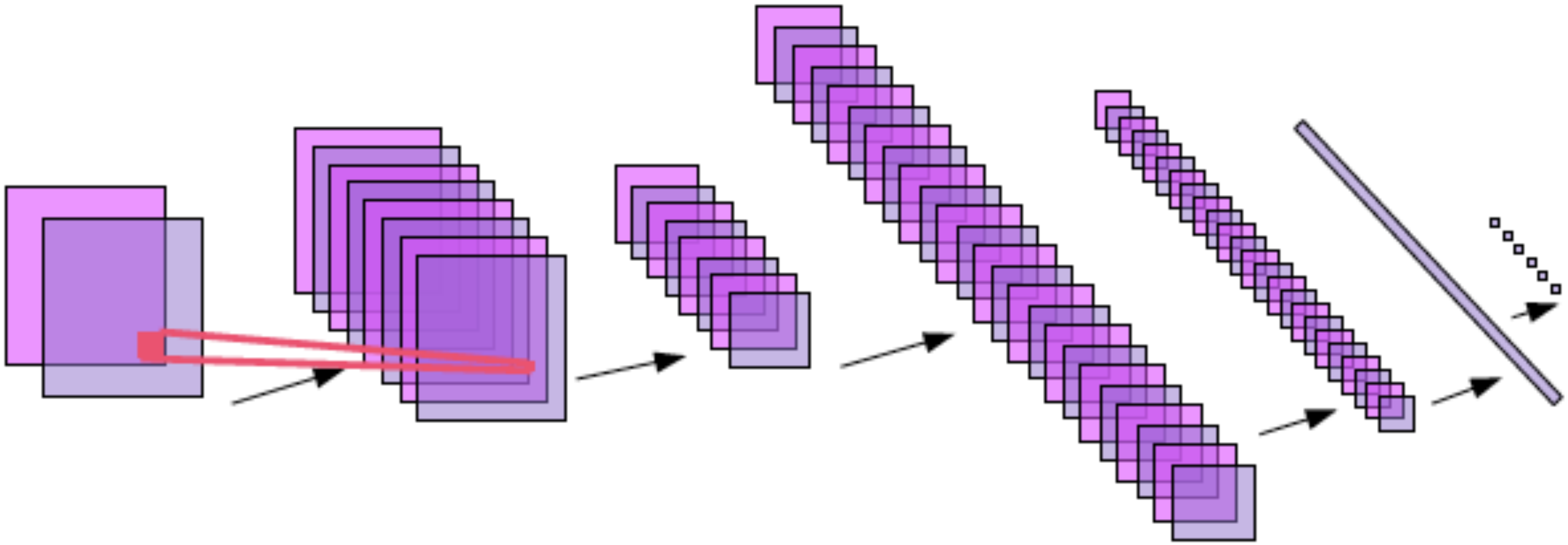
Introduction: Repeat



Convolutional network

- Hierarchical/multilayer: Features get progressively global, invariant and numerous.
- Dense features: Feature detectors applied everywhere (no interest points).
- Broadly tuned: Toward invariance.
- Complete recognition system: Integrates segmentation, feature extraction and classification.

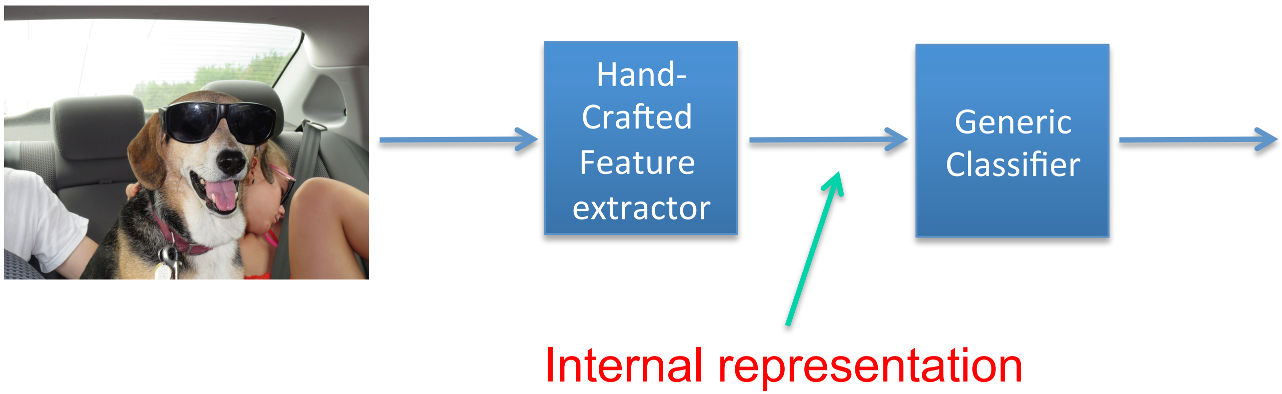
Introduction: Repeat with learning



Convolutional network

- Hierarchical/multilayer: Features get progressively global, invariant and numerous.
- Dense features: Feature detectors applied everywhere (no interest points).
- Broadly tuned: Toward invariance.
- Complete recognition system: Integrates segmentation, feature extraction and classification.
- Global discriminant training: Train whole system end-to-end, e.g., with a gradient based optimization algorithm to minimize a global loss function.

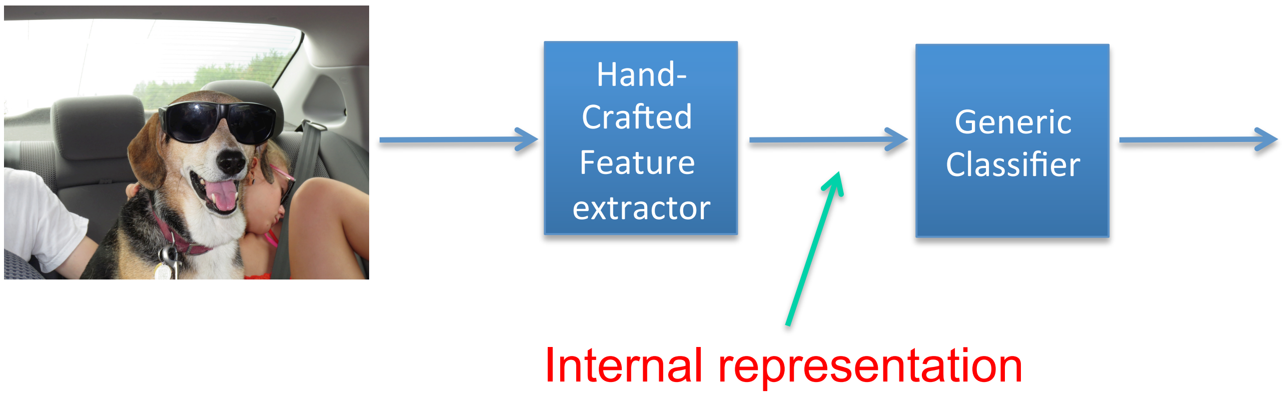
Introduction: The "traditional" approach



Key contrasting ideas

- Raw input is processed with a hand-crafted feature extractor.
- Features not learned.
- Classifier is "generic" (e.g., Nearest Neighbor, SVM, …).

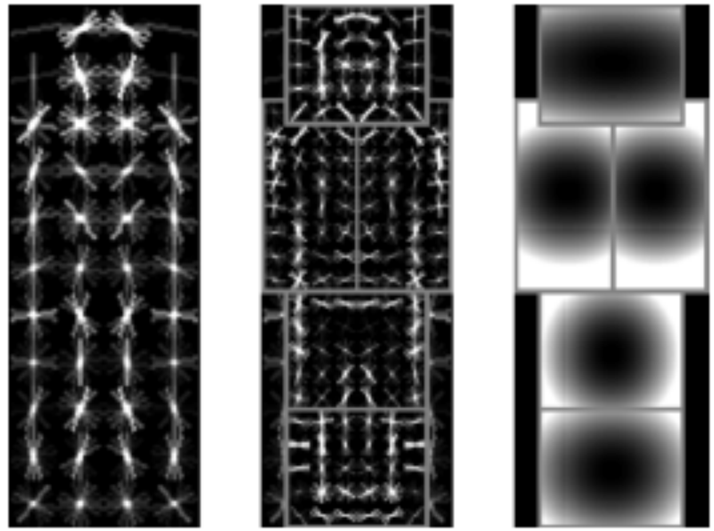
Introduction: The "traditional" approach



Key contrasting ideas

- Raw input is processed with a hand-crafted feature extractor.
- Features not learned.
- Classifier is "generic" (e.g., Nearest Neighbor, SVM, …).
- **Remark:** As with ConvNets, there likely are multiple stages of internal representation, but they are hand-crafted.

Introduction: The "traditional" features

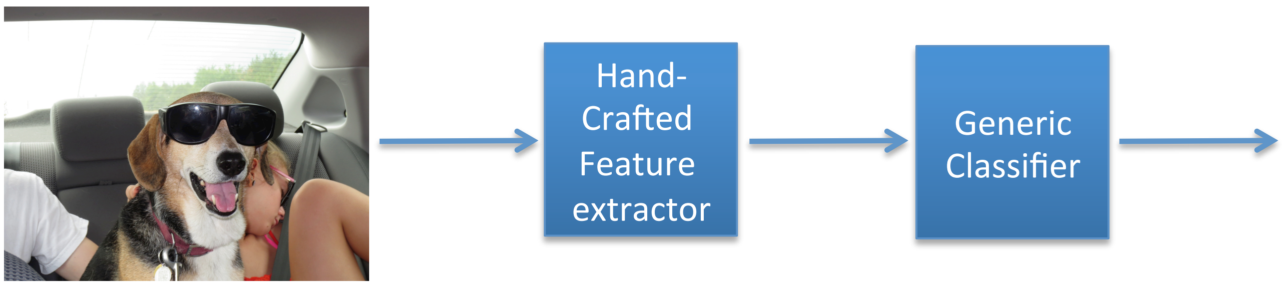


Key ideas

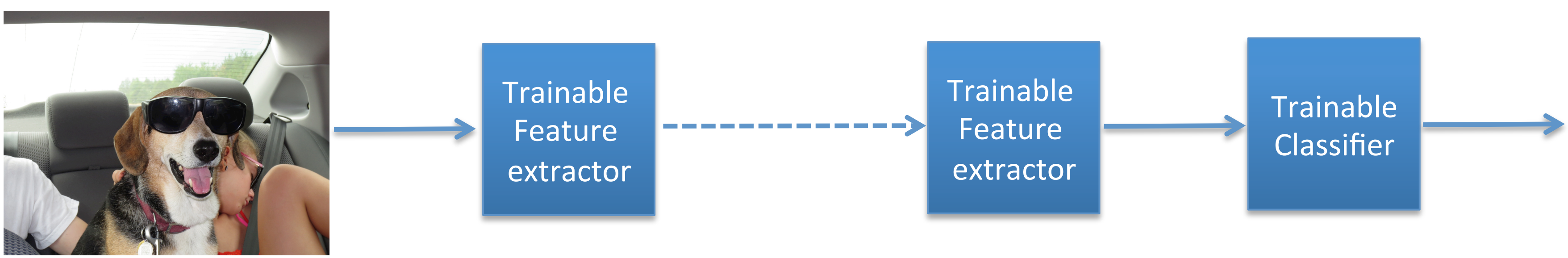
- Features are key to recent progress in recognition.
- Multiple of hand-designed features currently in use.
  - Edges
  - Corners
  - SOE
- What should be the next step?
  - Build ever better features?
  - Leverage better classifiers?

Introduction: Shallow vs. deep architectures

Traditional "shallow" architecture



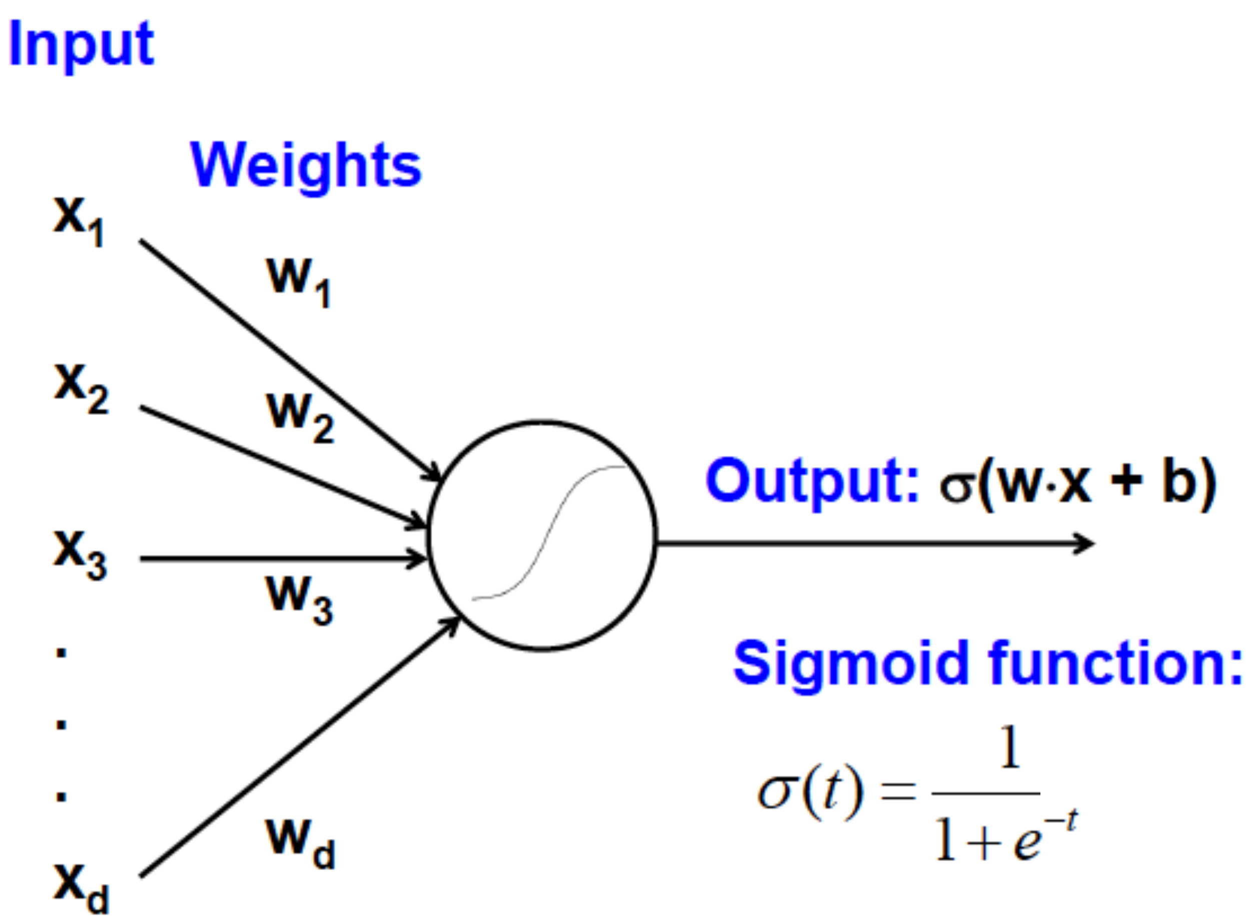
New (not really) "deep" architecture



Outline

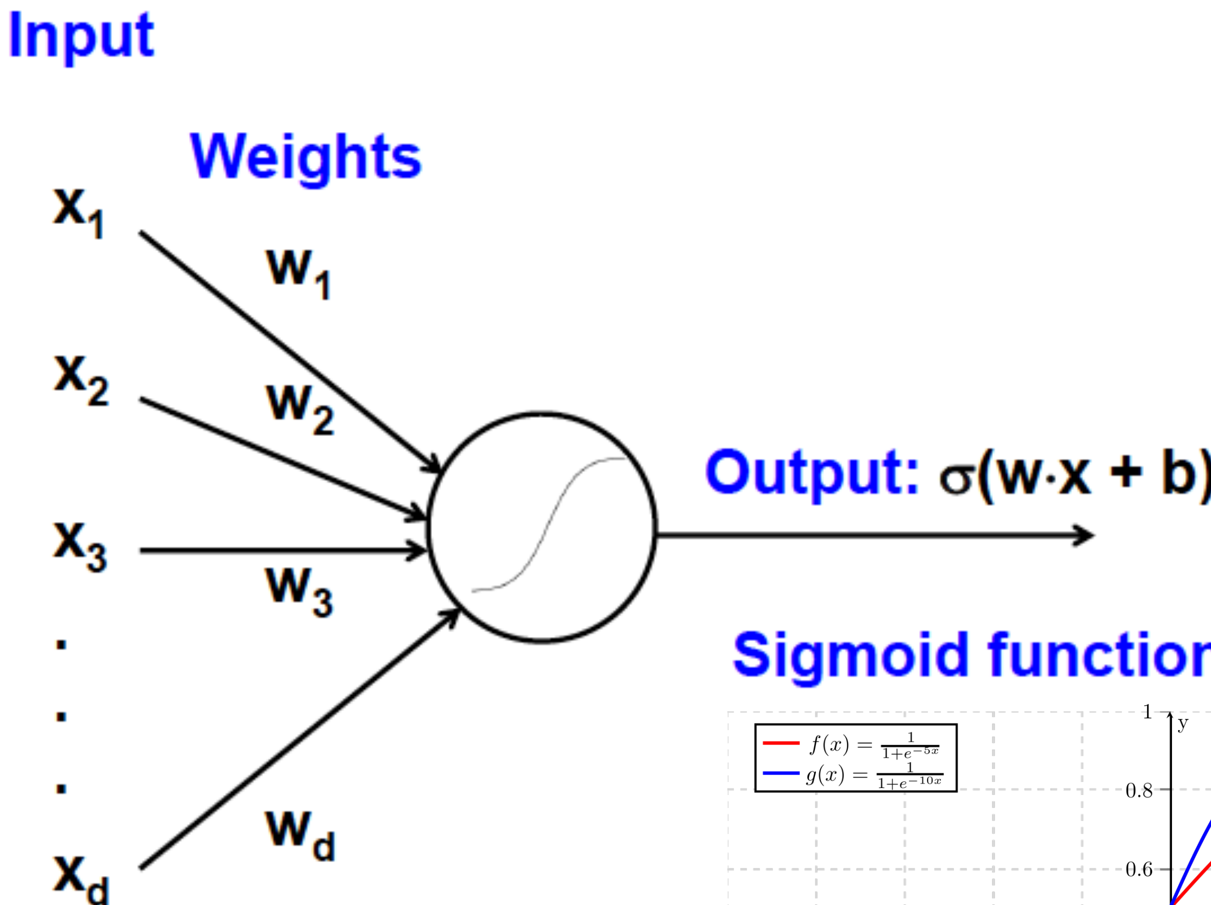
- Introduction
- **Background**
- Architecture
- Examples
- Summary

Background: Perceptron (Rosenblatt 1957)

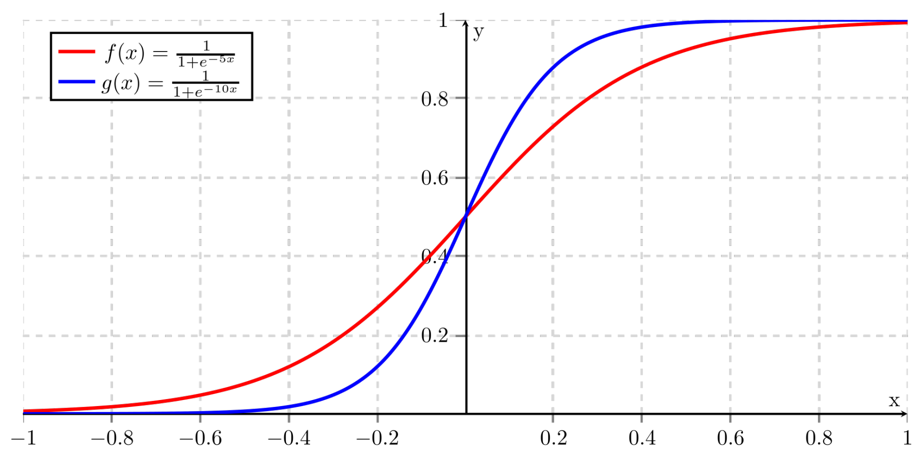


Rosenblatt, F. (1957) The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

Background: Perceptron (Rosenblatt 1957)

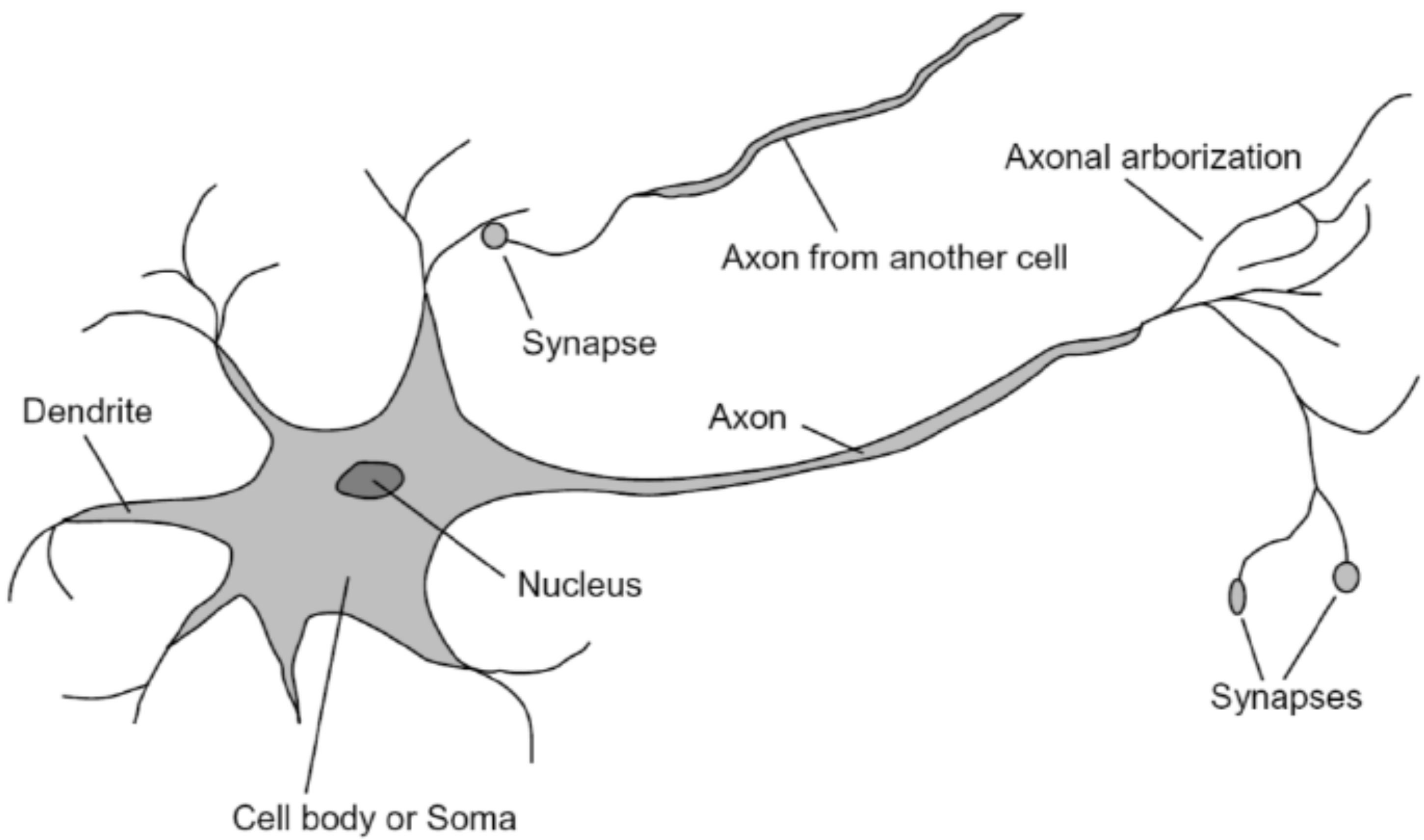


Sigmoid function:

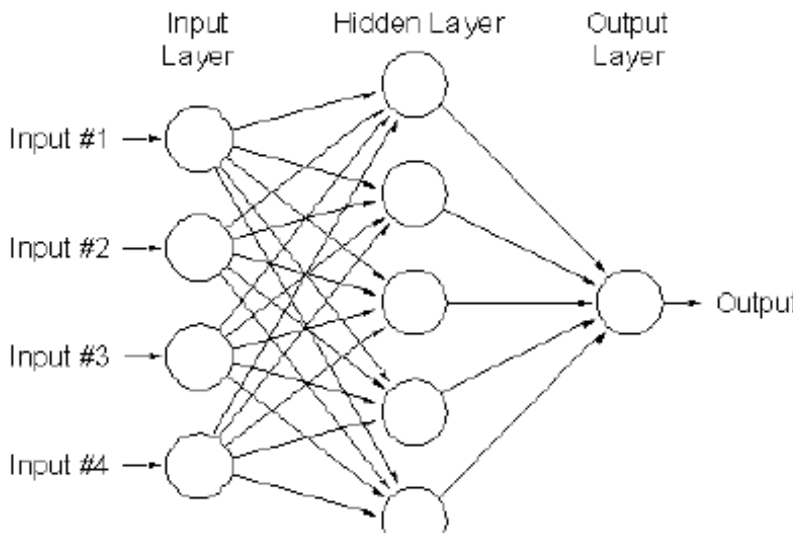


Rosenblatt, F. (1957) The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

Background: Inspiration from neurons



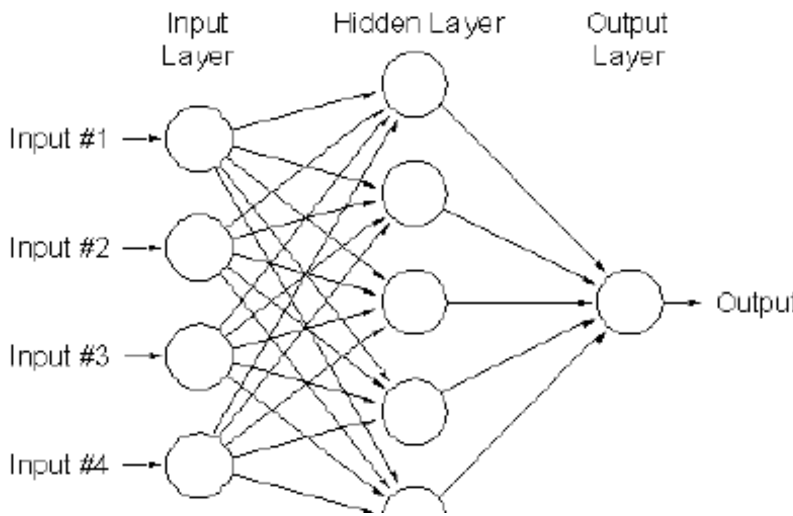
Background: Multilayer neural networks



Rosenblatt (1962): 3 layer perceptron

- Multilayer perceptron for classification.
- Input and output layers
- Hidden-layer, not seen by input nor output, connected between the two.
- Rosenblatt, F. (1962) Principles of Neurodynamics. Washington, DC:Spartan Books.

Background: Multilayer neural networks



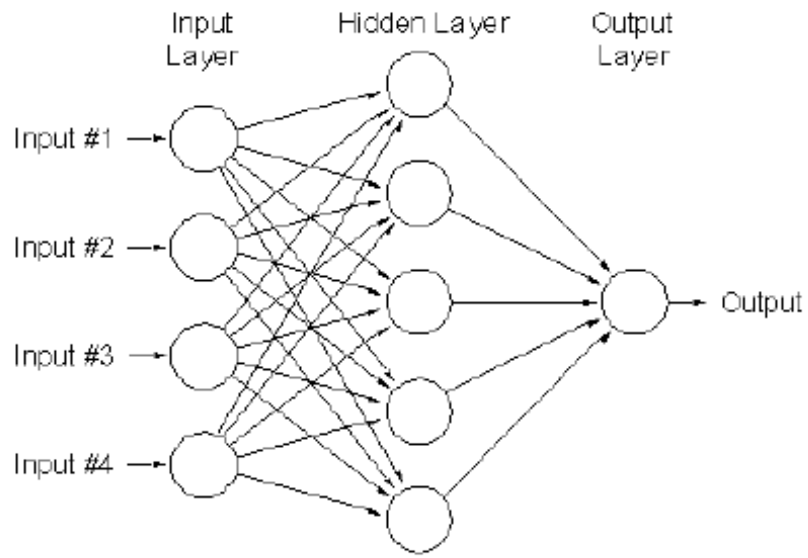
Where do the connection weights come from?

- **Training:** find network weights **w** to minimize the error between true training labels y_i and estimated labels $f_{\mathbf{w}}(\mathbf{x}_i)$:

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - f_{\mathbf{w}}(\mathbf{x}_i)\right)^2$$

- Minimization can be done by gradient descent provided f is differentiable
  - This training method is called **back-propagation**

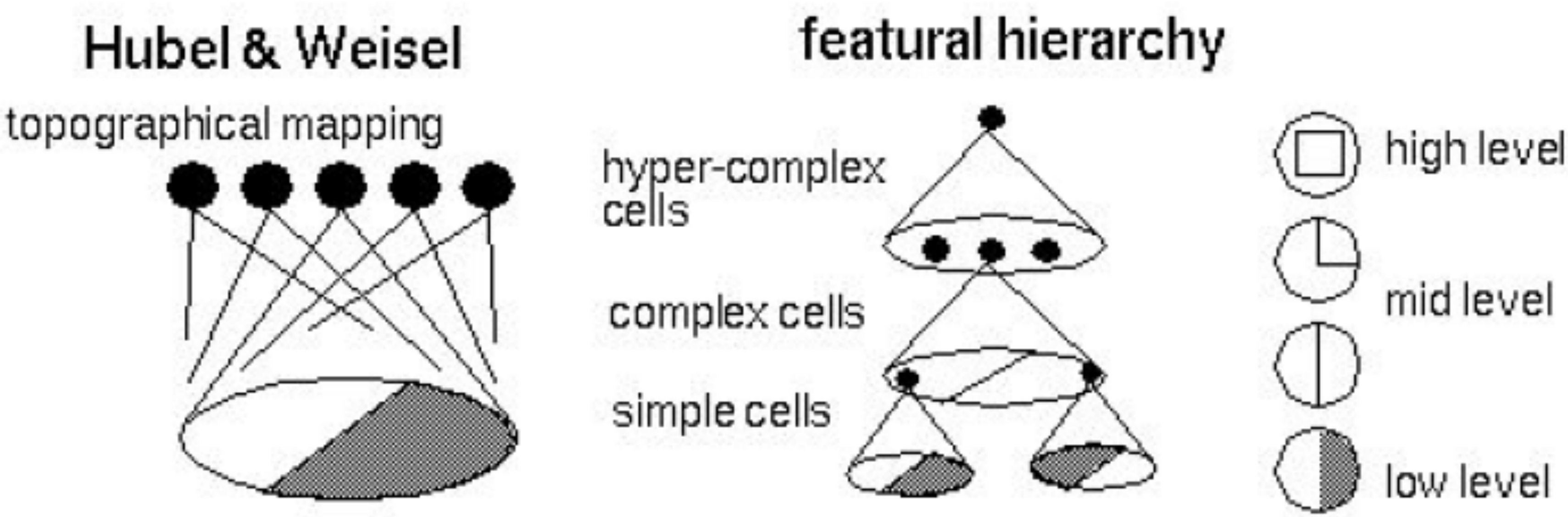
Background: Multilayer neural networks



Historical remark

- Back-propagation originally proposed in Bryson, Deham, Dreyfus (1963) Optimal programming problems with inequality constraints, AIAA J. 1 (11), 2544-2550.
- Subsequently applied to NN by Werbos (1970) in his Harvard PhD thesis, New Tools for Prediction and Analysis in the Behavioural Sciences.
- Popularized by Rumelhart, Hinton & Williams (1986) Learning representations by back-propagating errors, Nature 323, 533-536.

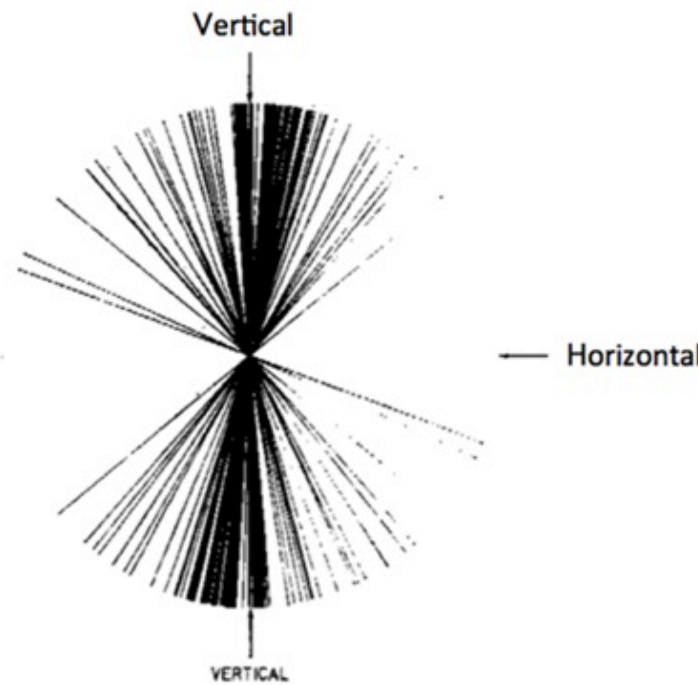
Background: Hubel/Wiesel visual cortex model



D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)

- Visual cortex consists of …
  - a hierarchy of simply complex and hypercomplex cells ..
  - with retinotopic organization.
- Based physiological recordings in cat cortex.
- D. Hubel & T. Wiesel (1959) Receptive fields of single neurons in the cat's striate cortex. Journal of Physiology 148 (3), 574-591.
- D. Hubel & T. Wiesel (1962) Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. Journal of Physiology 160 (1), 106-154.

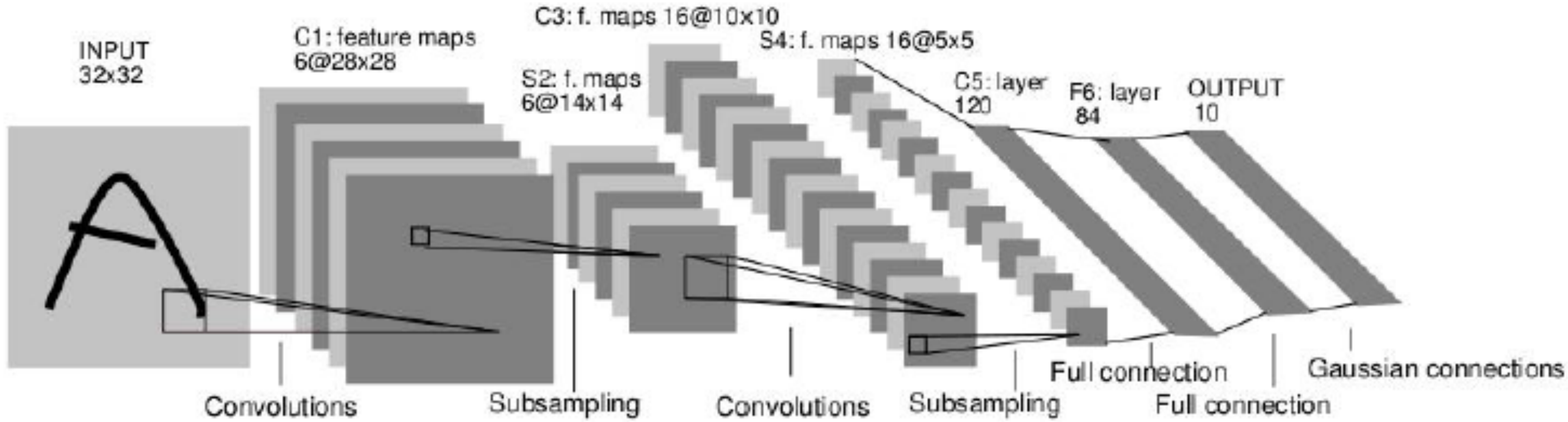
Background: Blakemore/Cooper



Blakemore & Cooper (1970)

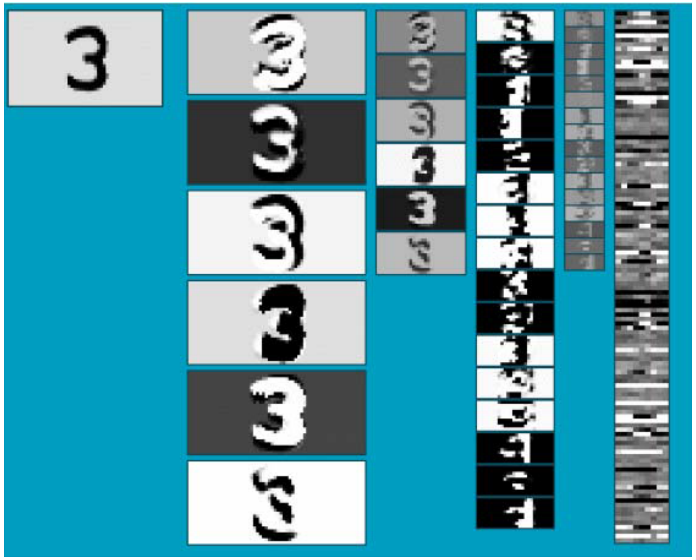
- Cats raised in environment consisting of lines of only one orientation…
- … hand no cortical neurons responding to the orthogonal orientation.
- Suggests role of stimulus driven learning in neural development.
- C. Blakemore & G. Cooper (1970), Development of the brain depends on visual environment. Nature 228, 447-448.

Background: From Hubel/Wiesel to ConvNets

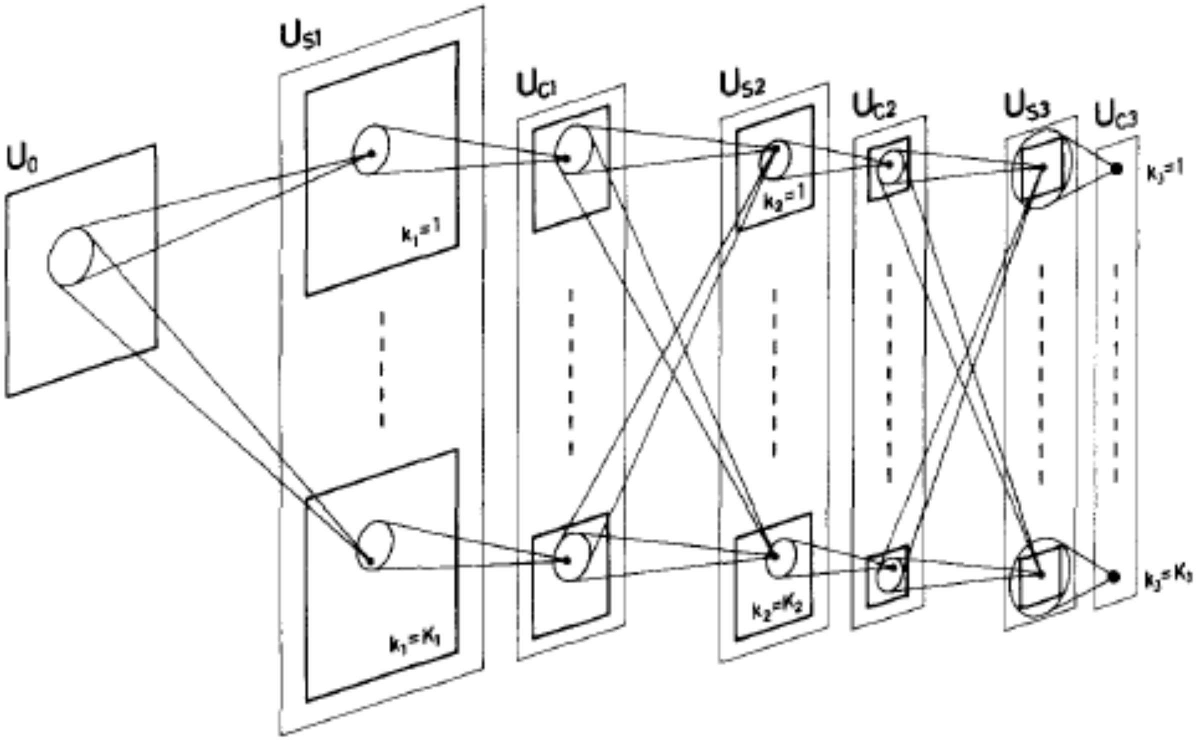


LeCun et al. 1998

- Neural network with special connectivity structure.
- Stack multiple layers of feature extractors.
- Higher layers extract more global and invariant descriptors.
- Classification at the end.
- Supervised learning via back-propagation.



Background: Prehistory of ConvNets



Neocognitron (Fukushima 1980)

- Similar architectures were proposed earlier.
- Indeed, they had arguably more sophisticated learning capabilities (non-supervised)…
- … as well as recurrent connections that enabled selective attention.
- They were even applied to the same problems.

Background: Backpropagation

Feedforward operation

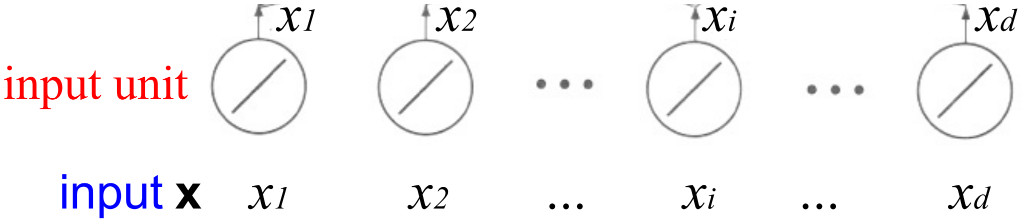
- A d-dimensional input **x** is presented to the input layer.

input **x** x_1 x_2 … x_i … x_d

Background: Backpropagation

Feedforward operation

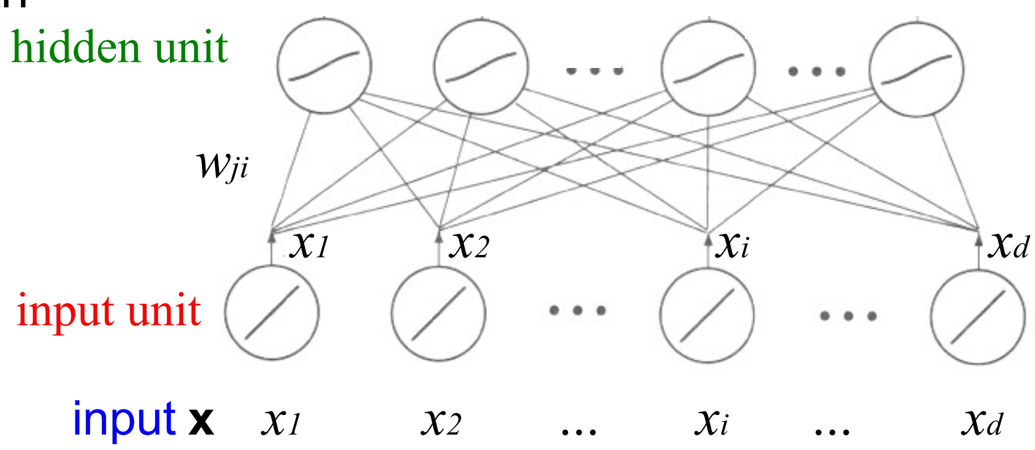
- A d-dimensional input **x** is presented to the input layer.
- Each input unit emits a corresponding component x_i



Background: Backpropagation

Feedforward operation

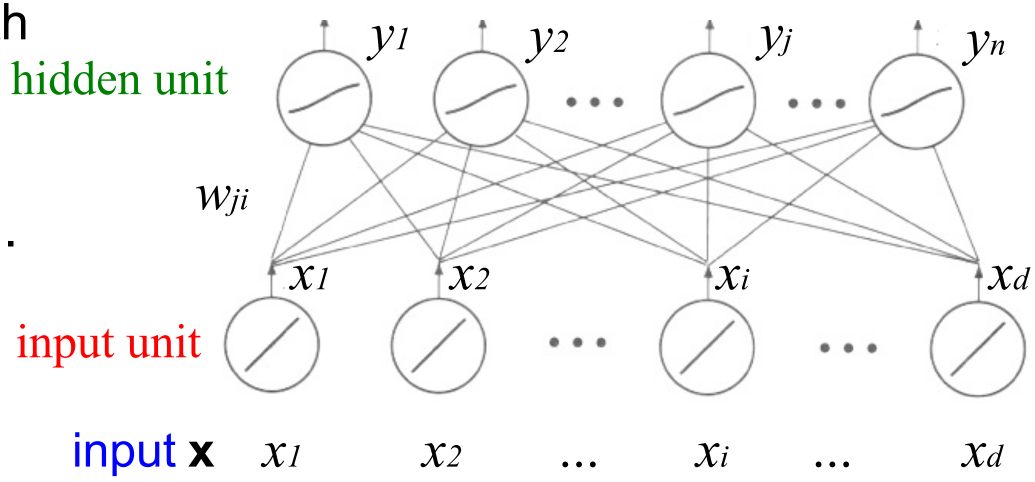
- A d-dimensional input **x** is presented to the input layer.
- Each input unit emits a corresponding component x_i
- Each of the n hidden units computes its net activation net_j as the inner product of the input layer signals with weights w_{ji} at the hidden unit.



Background: Backpropagation

Feedforward operation

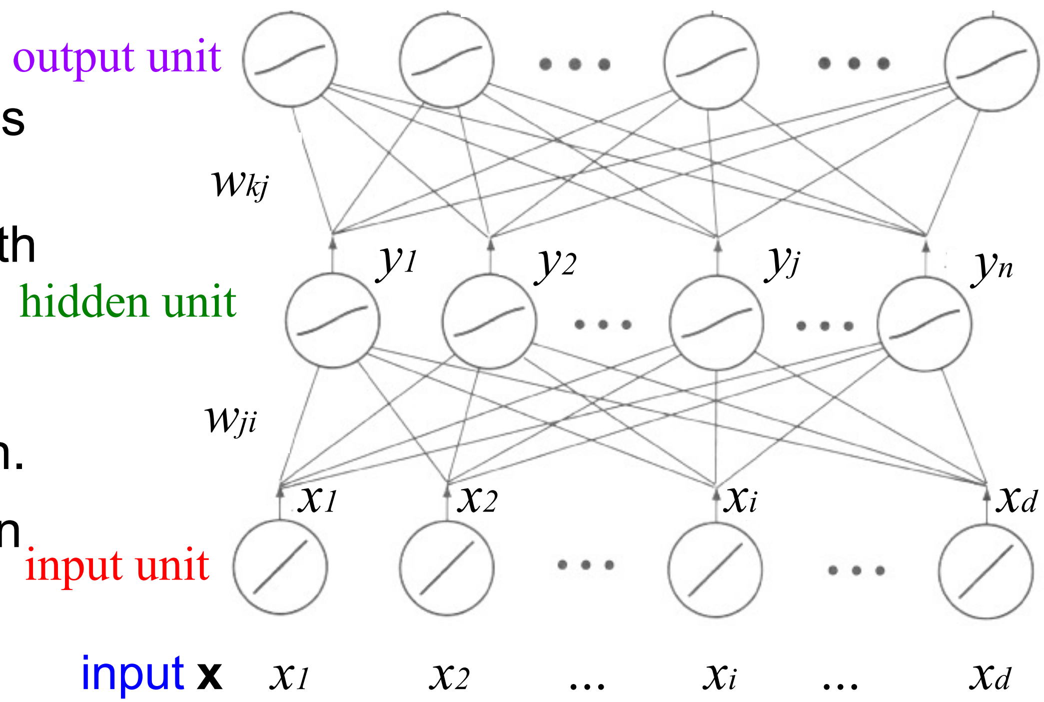
- A d-dimensional input $\mathbf{x}$ is presented to the input layer.
- Each input unit emits a corresponding component x_i
- Each of the n hidden units computes its net activation net_j as the inner product of the input layer signals with weights w_{ji} at the hidden unit.
- The hidden unit emits $y_j = \mathrm{f}(net_j)$, with f a nonlinear activation function.



Background: Backpropagation

Feedforward operation

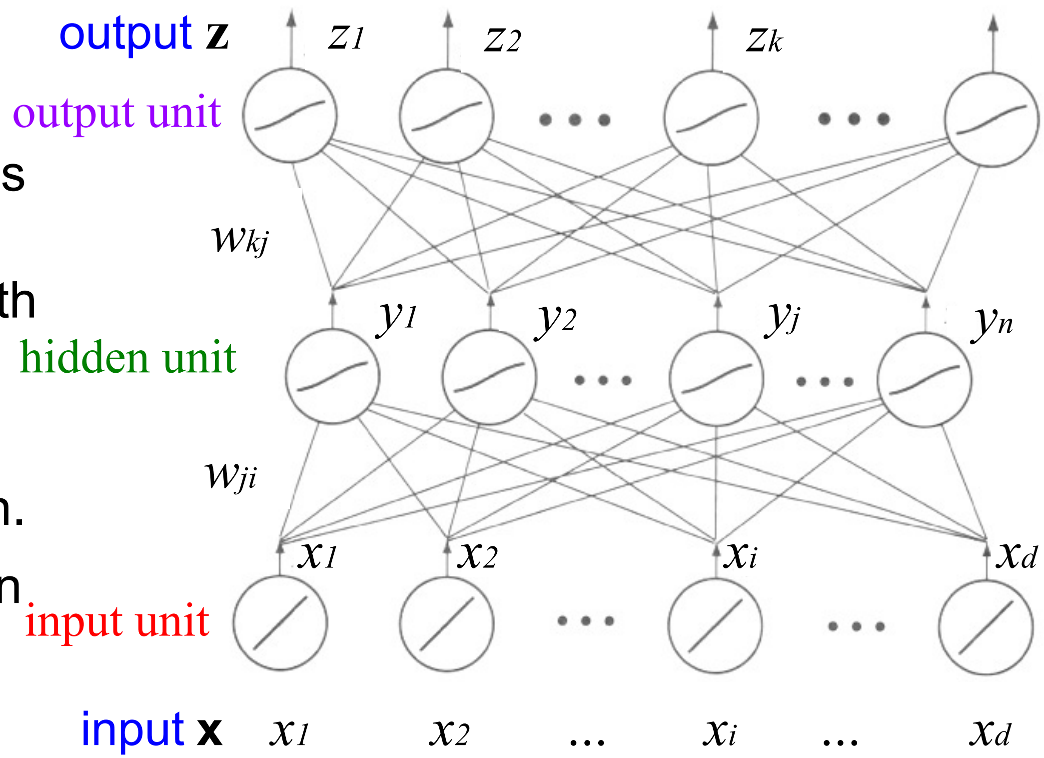
- A d-dimensional input **x** is presented to the input layer.
- Each input unit emits a corresponding component x_i
- Each of the n hidden units computes its net activation net_j as the inner product of the input layer signals with weights w_{ji} at the hidden unit.
- The hidden unit emits $y_j = \mathrm{f}(net_j)$, with f a nonlinear activation function.
- Each of the c output units function in the same way as the hidden units.



Background: Backpropagation

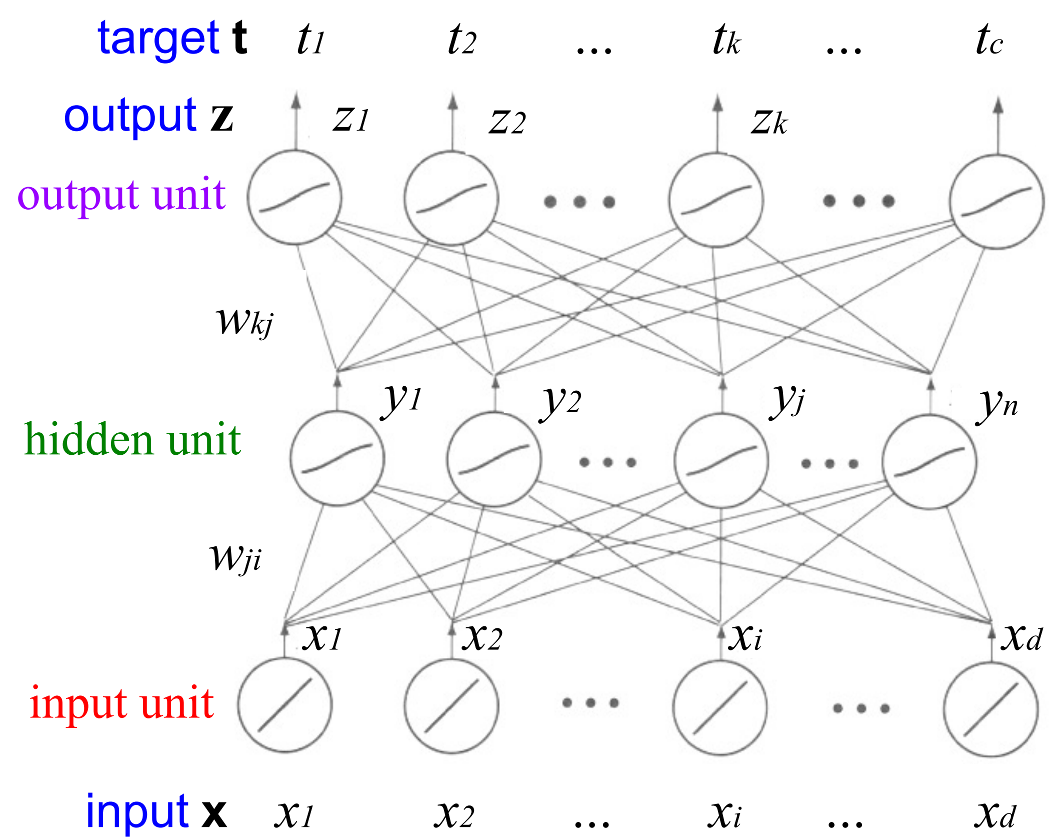
Feedforward operation

- A d-dimensional input **x** is presented to the input layer.
- Each input unit emits a corresponding component x_i
- Each of the n hidden units computes its net activation net_j as the inner product of the input layer signals with weights w_{ji} at the hidden unit.
- The hidden unit emits $y_j = \mathrm{f}(net_j)$, with f a nonlinear activation function.
- Each of the c output units function in the same way as the hidden units.
- The final emitted signals, $z_k = \mathrm{f}(net_k)$, are used as discriminant functions for classification.



Background: Backpropagation

Training error



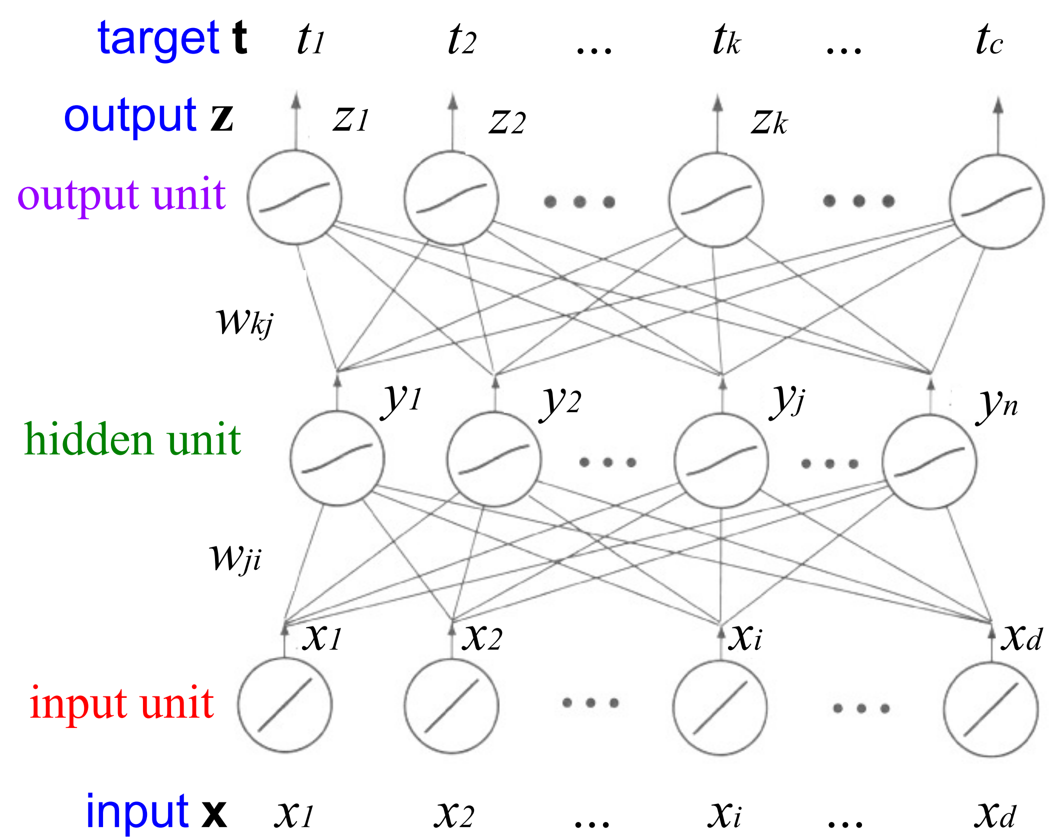
Background: Backpropagation

Training error

- Let the training error on a pattern be the sum over output units of the squared difference between desired output t_k given by a teacher and the actual output z_k

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

- With c the length of the target and network output vectors and $\mathbf{w}$ all the weights in the network.



Background: Backpropagation

Learning

- Initialize weights to random variables.

Background: Backpropagation

Learning

- Initialize weights to random variables.
- Change weights in a direction that reduces the error

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

Background: Backpropagation

Learning

- Initialize weights to random variables.
- Change weights in a direction that reduces the error

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

- In component form

$$\Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}}$$

with η the learning rate that indicates the relative size of change in the weights.

Background: Backpropagation

Learning

- Initialize weights to random variables.
- Change weights in a direction that reduces the error

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

- In component form

$$\Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}}$$

  with η the learning rate that indicates the relative size of change in the weights.
- An iterative algorithm results

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta \mathbf{w}(m)$$

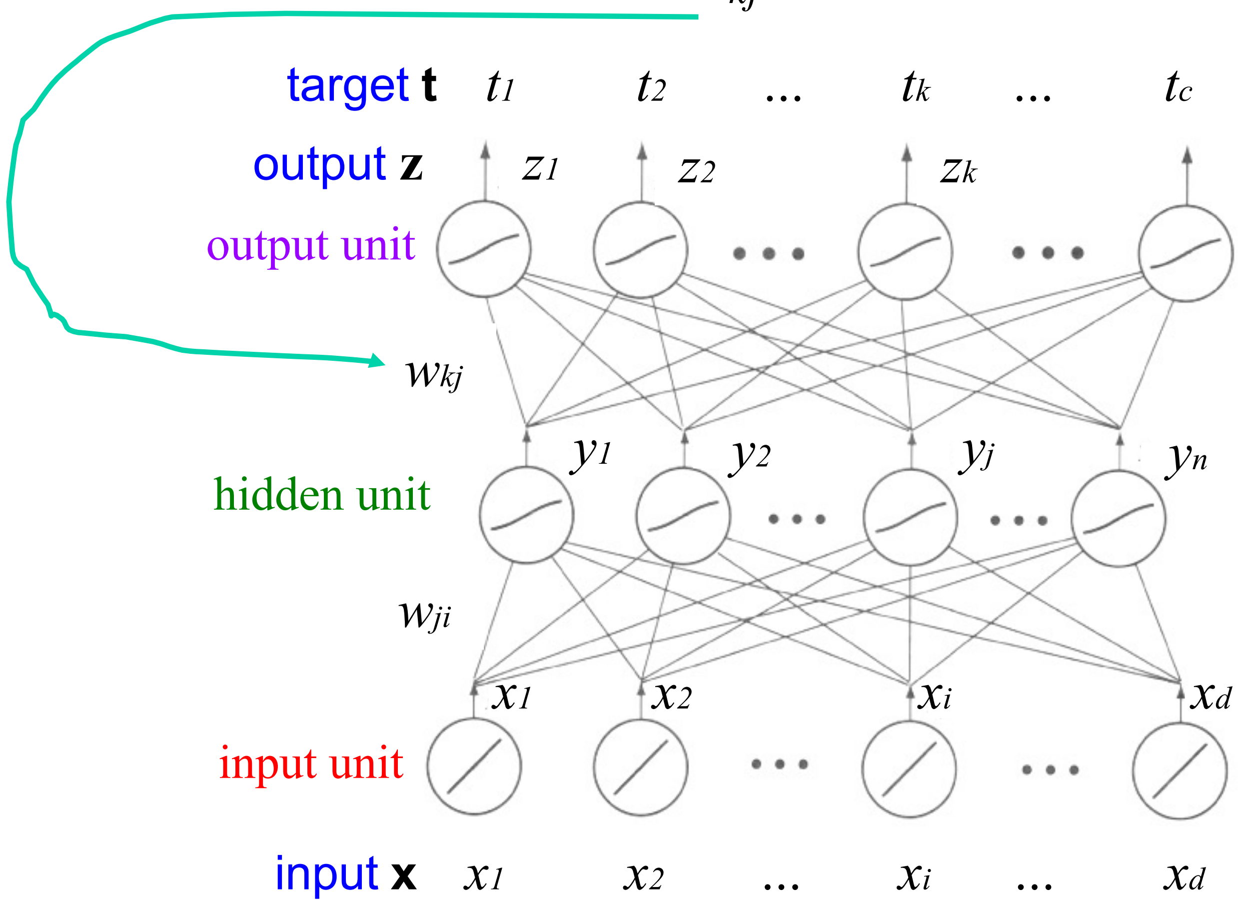
with m the particular pattern presented.

Background: Backpropagation

Hidden-to-output weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$



Background: Backpropagation

Hidden-to-output weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$

- Since the error does not depend explicitly on w_{kj}, we have a problem

Background: Backpropagation

Hidden-to-output weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$

- Since the error does not depend explicitly on w_{kj}, we have a problem

$$\frac{\partial J}{\partial w_{kj}}$$

Background: Backpropagation

Hidden-to-output weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$

- Since the error does not depend explicitly on w_{kj}, we have a problem

$$\frac{\partial J}{\partial w_{kj}}$$

$$J = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

Background: Backpropagation

Hidden-to-output weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$

- Since the error does not depend explicitly on w_{kj}, we have a problem

$$\frac{\partial J}{\partial w_{kj}}$$

$$J = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

$$z_k = f(net_k)$$

Background: Backpropagation

Hidden-to-output weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$

- Since the error does not depend explicitly on w_{kj}, we have a problem

$$\frac{\partial J}{\partial w_{kj}}$$

$$J = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

$$z_k = f(net_k)$$

$$net_k = \sum_j w_{kj} y_j$$

Background: Backpropagation

Hidden-to-output weight update

Recall: The "chain rule",
let
h(x)=g[f(x)]
then
h'(x)=g'[f(x)] f'(x)

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$

- Since the error does not depend explicitly on w_{kj}, we use the chain rule

$$\frac{\partial J}{\partial w_{kj}}$$

$$J = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

$$z_k = f(net_k)$$

$$net_k = \sum_j w_{kj} y_j$$

Background: Backpropagation

Hidden-to-output weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$

Recall: The "chain rule", let

$h(x)=g[f(x)]$

then

$h'(x)=g'[f(x)]\,f'(x)$

- Since the error does not depend explicitly on w_{kj}, we use the chain rule

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}}$$

$$J = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

$$z_k = f(net_k)$$

$$net_k = \sum_j w_{kj} y_j$$

Background: Backpropagation

Hidden-to-output weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{kj}}$$

- Since the error does not depend explicitly on w_{kj}, we use the chain rule

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

with

$$\delta_k = -\frac{\partial J}{\partial net_k}$$

the sensitivity of unit k and describes the overall error change as a function of the unit's net activation.

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k}$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k}$$

$$J = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

$$z_k = f(net_k)$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k}$$

$$J = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

$$z_k = f(net_k)$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k}$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

$$-\frac{\partial}{\partial z_k}\frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

$$\frac{\partial}{\partial net_k} f(net_k)$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

- Next we evaluate the second component of the error

$$\frac{\partial J}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

- Next we evaluate the second component of the error

$$\frac{\partial J}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

as

$$net_k = \sum_j w_{kj} y_j$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

- Next we evaluate the second component of the error

$$\frac{\partial J}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

as

$$net_k = \sum_j w_{kj} y_j$$

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

Background: Backpropagation

Hidden-to-output weight update

- Assuming the activation function f is differentiable

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

- Next we evaluate the second component of the error

$$\frac{\partial J}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

as

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

- So that the weight update is

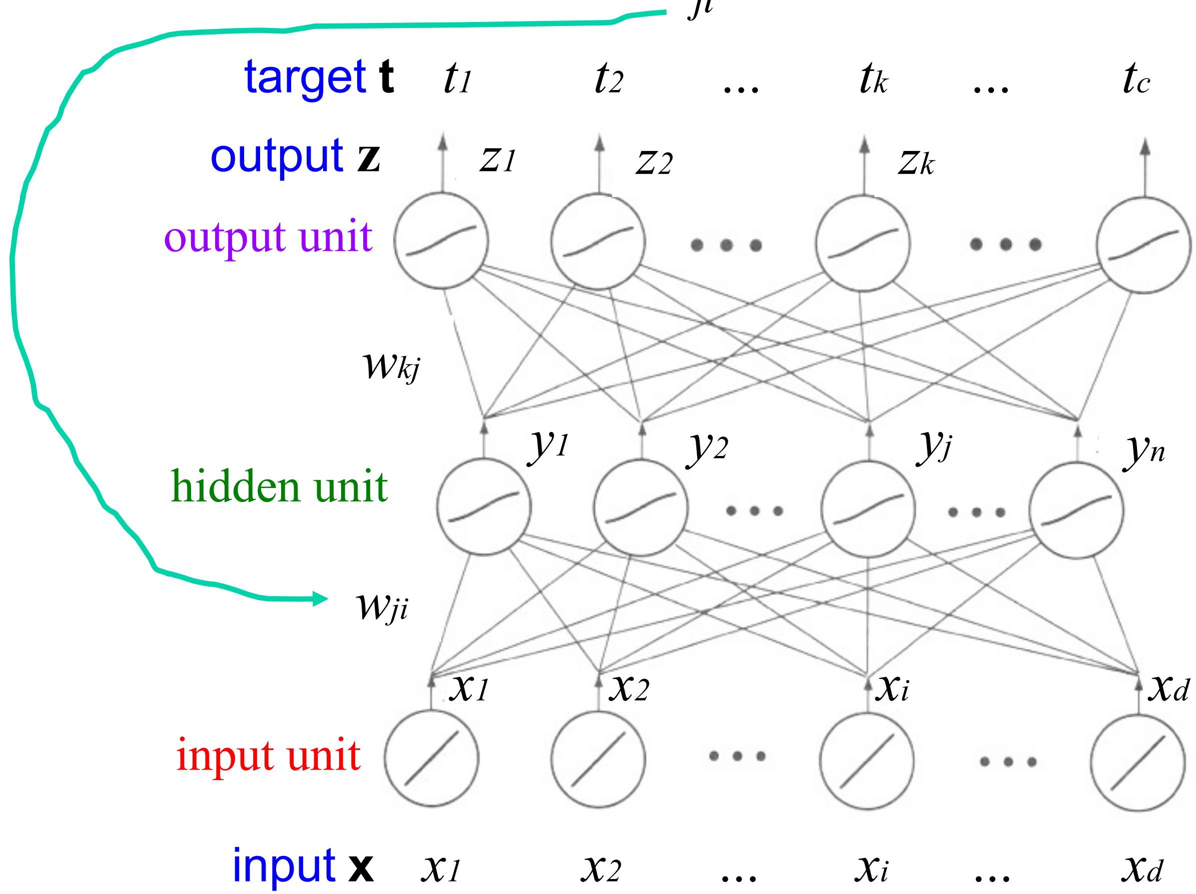
$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

Background: Backpropagation

Input-to-hidden weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{ji}}$$



Background: Backpropagation

Input-to-hidden weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{ji}}$$

- Since the error does not depend explicitly on w_{ji}, we use the chain rule

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

Background: Backpropagation

Input-to-hidden weight update

- We wish to evaluate

$$\frac{\partial J}{\partial w_{ji}}$$

- Since the error does not depend explicitly on w_{ji}, we use the chain rule

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

- The first term on the RHS involves all the weights

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2\right)$$

because each z_k depends on all y_j

Background: Backpropagation

Input-to-hidden weight update

- Now we evaluate $$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2\right)$$

Background: Backpropagation

Input-to-hidden weight update

- Now we evaluate

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2\right)$$

$$= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial y_j}$$

Background: Backpropagation

Input-to-hidden weight update

- Now we evaluate

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right)$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$

$$z_k = f(net_k)$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$$

$$net_k = \sum_j w_{kj} y_j$$

Background: Backpropagation

Input-to-hidden weight update

- Now we evaluate

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2\right)$$

$$= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial y_j}$$

$$z_k = f(net_k)$$

$$= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$$

$$net_k = \sum_j w_{kj} y_j$$

$$= -\sum_{k=1}^{c}\left(t_k - z_k\right) f'(net_k) w_{kj}$$

Background: Backpropagation

Input-to-hidden weight update

- Now we evaluate

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right)$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{kj}$$

Background: Backpropagation

Input-to-hidden weight update

- Now we evaluate

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2\right)$$

$$= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}\left(t_k - z_k\right)f'(net_k)w_{kj}$$

$$= \sum_{k=1}^{c}\delta_k w_{kj}$$

Background: Backpropagation

Input-to-hidden weight update

- Now we evaluate

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right)$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}\underbrace{(t_k - z_k)f'(net_k)}w_{kj}$$

$$= \sum_{k=1}^{c}\delta_k w_{kj}$$

- The sum over output units expresses how the hidden unit y_j affects error at each output unit.

Background: Backpropagation

Input-to-hidden weight update

- We still need to evaluate the remaining two terms on the RHS of the error

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

Background: Backpropagation

Input-to-hidden weight update

- We still need to evaluate the remaining two terms on the RHS of the error

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

- They yield as

$$y_j = f(net_j)$$

$$\frac{\partial y_j}{\partial net_j} = f'(net_j)$$

$$net_j = \sum_i w_{ji} x_i$$

$$\frac{\partial net_j}{\partial w_{ji}} = x_i$$

Background: Backpropagation

Input-to-hidden weight update

- We still need to evaluate the remaining two terms on the RHS of the error

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

- They yield as

$$\frac{\partial y_j}{\partial net_j} = f'(net_j)$$

$$\frac{\partial net_j}{\partial w_{ji}} = x_i$$

- Pulling all together, we have

$$\Delta w_{ji} = \eta \left(\sum_{k=1}^{c} w_{kj} \delta_k \right) f'(net_j) x_i$$

Background: Backpropagation

Input-to-hidden weight update

- We can further interpret the update

$$\Delta w_{ji} = \eta \left(\sum_{k=1}^{c} w_{kj} \delta_k \right) f'(net_j) x_i$$

$$= \eta x_i \delta_j$$

- Here, δ_j is the sensitivity for given (hidden) unit
  - The sum of the individual sensitivities of the output units
  - Weighted by the hidden-to-output weights
  - All modulated by the derivative of the activation function, f

Background: Backpropagation

Recapitulation

- We seek to minimize the training error

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2$$

as a function of the all the weights, $\mathbf{w}$, in the network.
- To do so, we employ gradient descent.
- The chain rule serves to push the error derivatives through the network.
- While we have only explicitly derived the weight updates for a 3 layer network, the same methodology works for ever more layers.

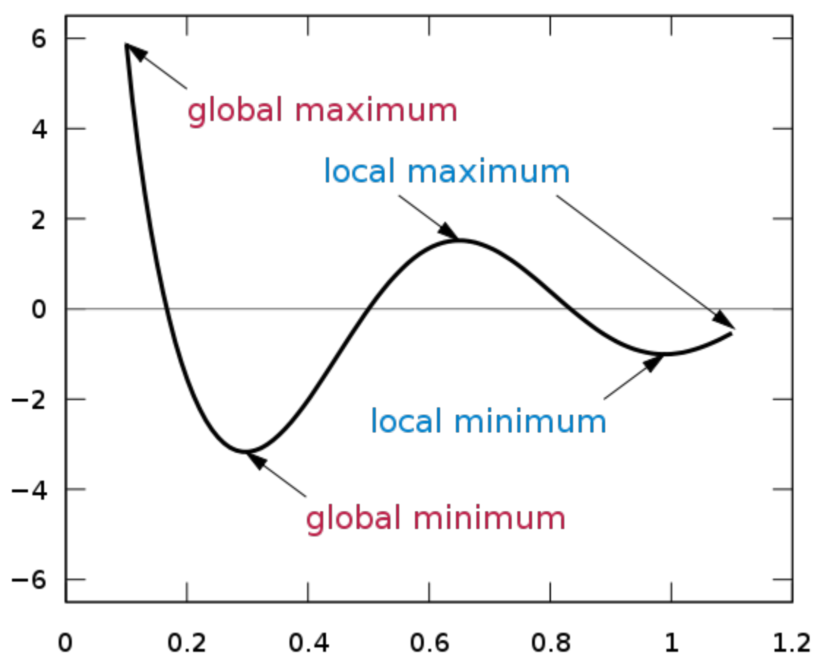
Background: Backpropagation

Caveat

- Gradient descent, e.g.,

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m), \quad \Delta\mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

only finds local minima!



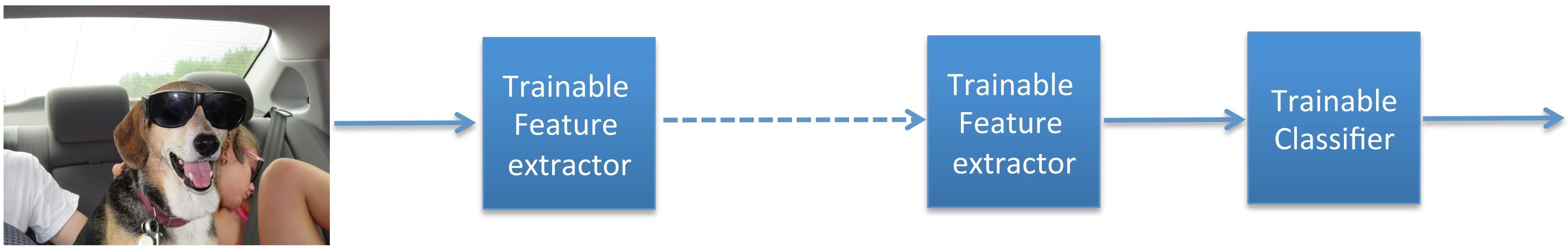
Outline

- Introduction
- Background
- **Architecture**
- Examples
- Summary

Architecture: Feedforward extraction

Looking under the hood

- Our overall architecture is



Architecture: Feedforward extraction

Looking under the hood

- Our overall architecture is

- But, what is inside each Feature Extractor?

Architecture: Feedforward extraction

Looking under the hood

- Our overall architecture is

Trainable Feature extractor → Trainable Feature extractor → Trainable Classifier →

- But, what is inside each Feature Extractor?

Trainable Feature extractor

Convolution (learned) → Nonlinearity → Spatial pooling → Normalization

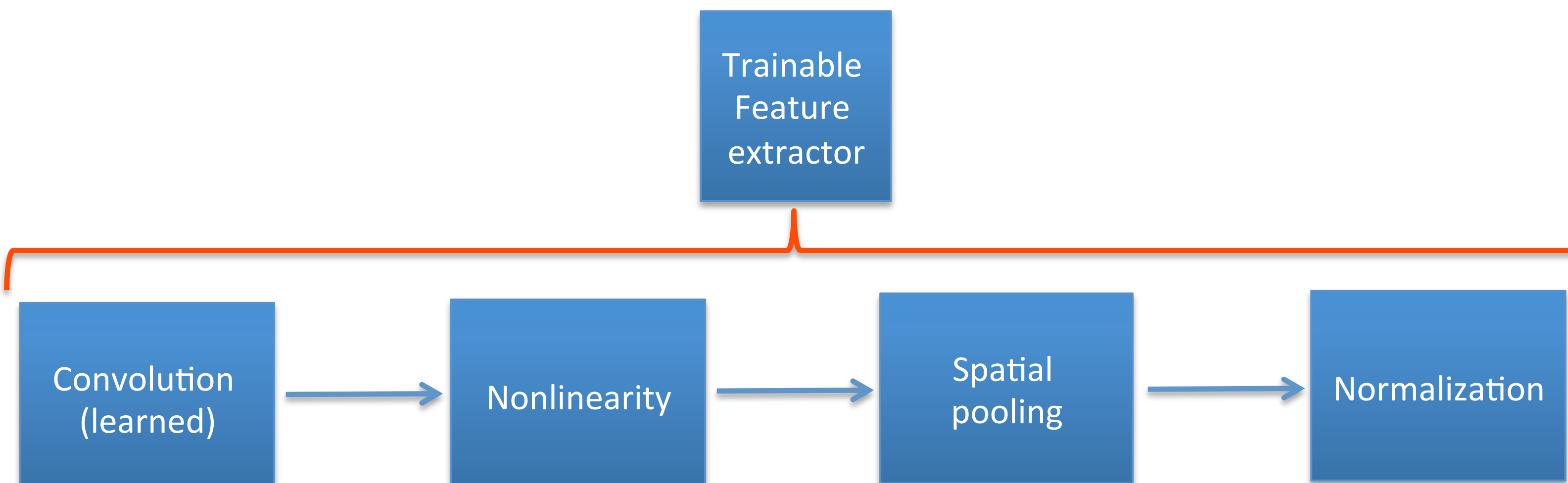
Architecture: Feedforward extraction

Looking under the hood

- Our overall architecture is

Remarks

- But, what is inside each Feature Extractor?



Architecture: Feedforward extraction

Looking under the hood

- Our overall architecture is

Trainable Feature extractor → ... → Trainable Feature extractor → Trainable Classifier →

- But, what is inside each Feature Extractor?

Trainable Feature extractor

Convolution (learned) → Nonlinearity → Spatial pooling → Normalization

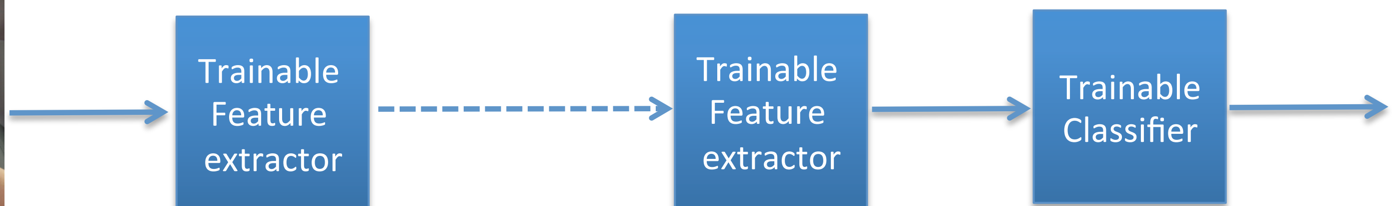
Remarks

- Inspiration is, once again, taken from biology.

Architecture: Feedforward extraction

Looking under the hood

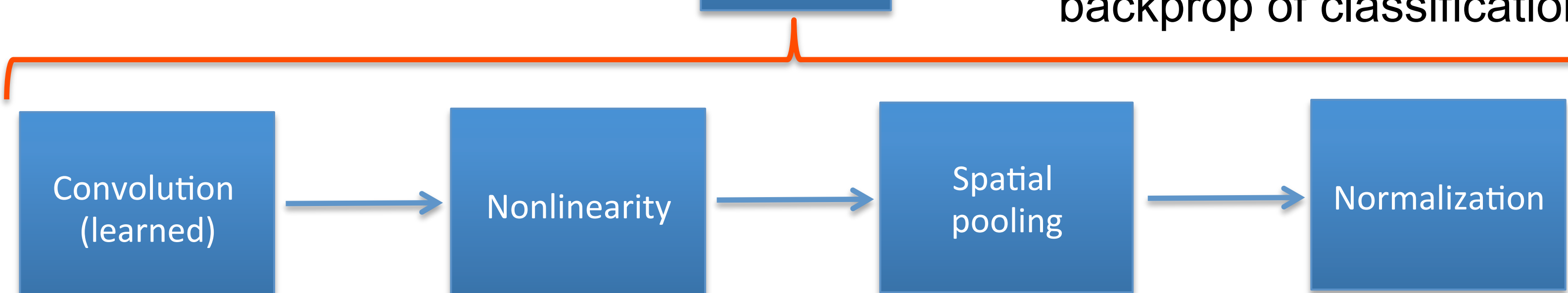
- Our overall architecture is



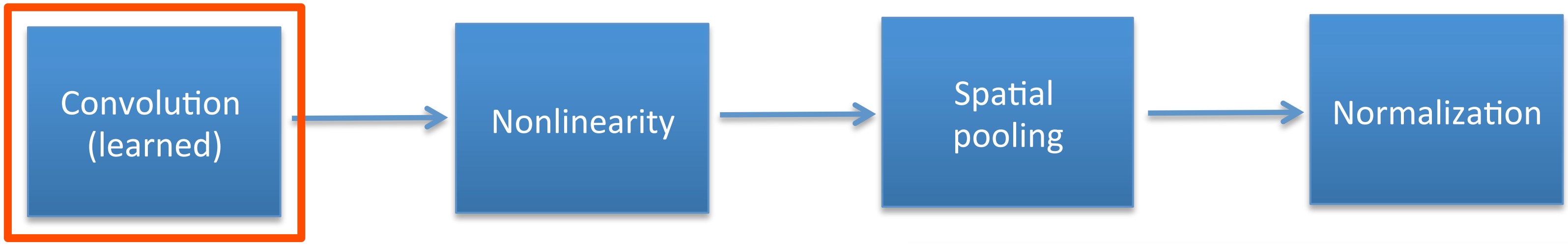
- But, what is inside each Feature Extractor?

Remarks

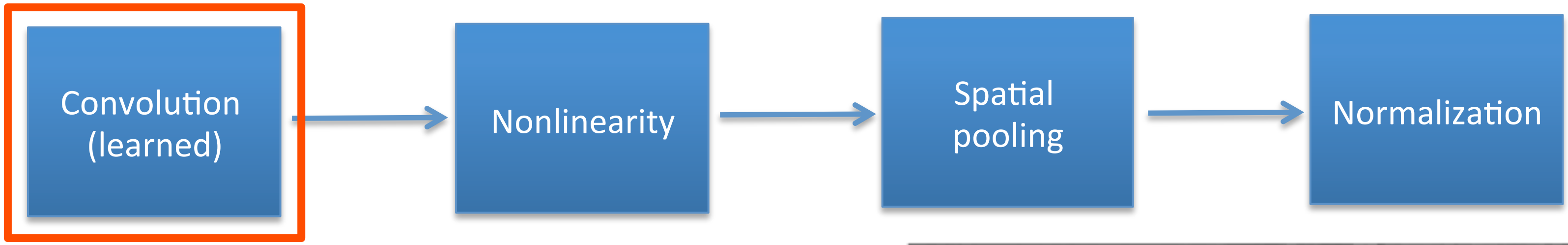
- Inspiration is, once again, taken from biology.
- Convolution filters learned via supervised training with backprop of classification error.

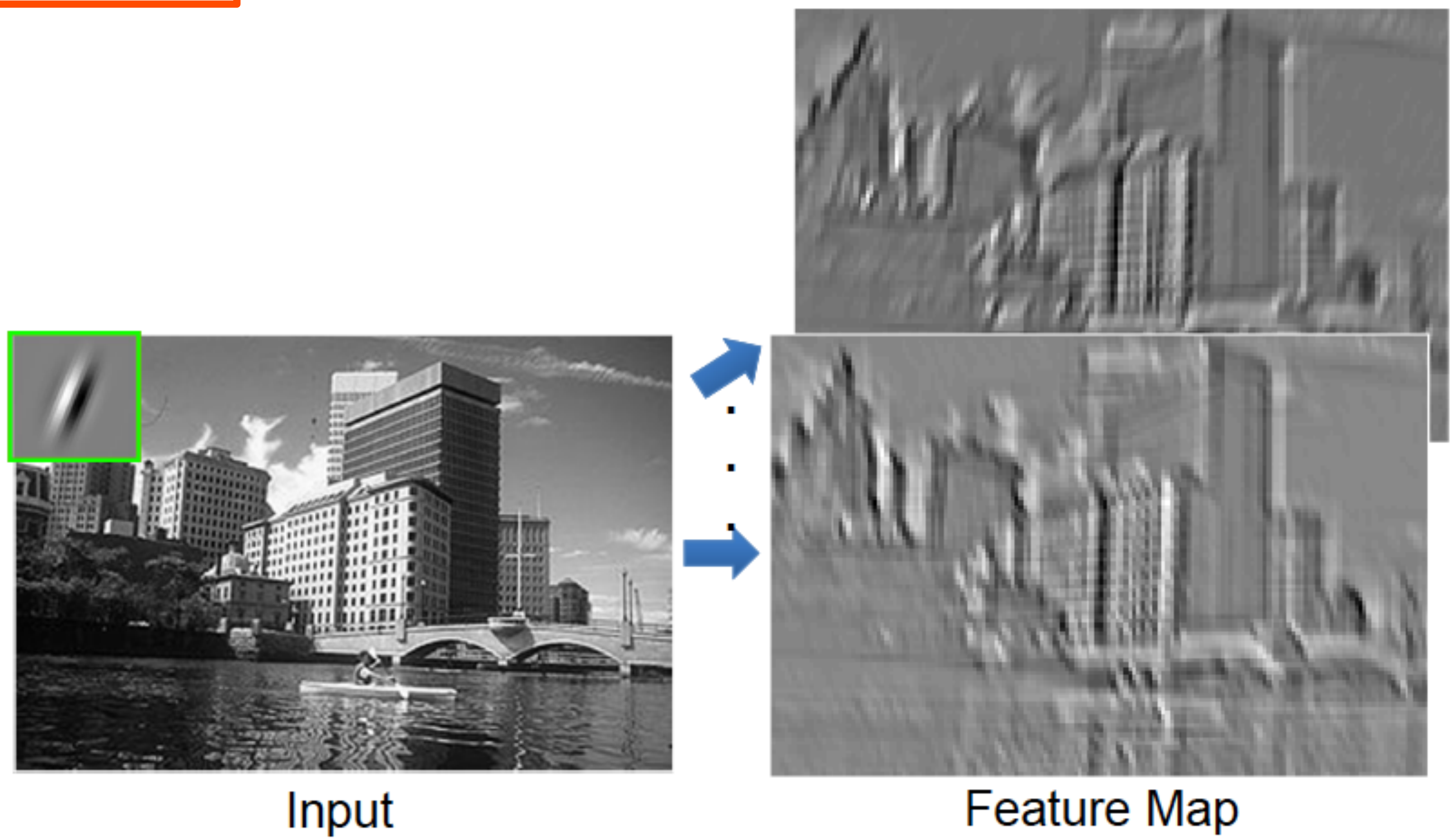


Architecture: Under the hood

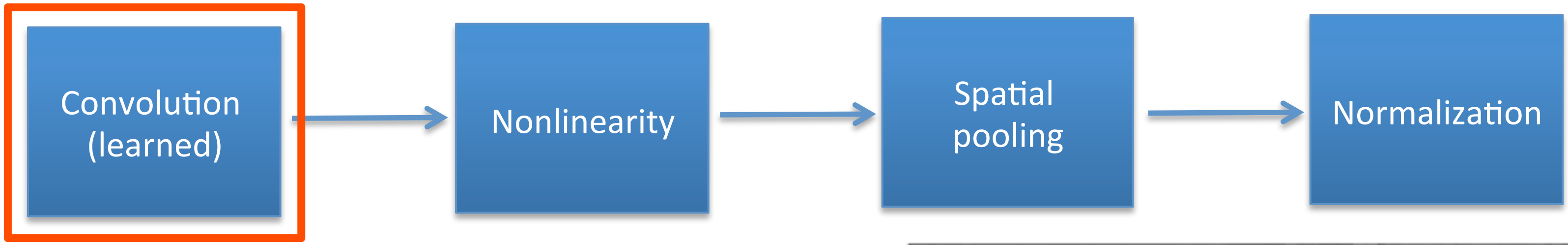


Architecture: Under the hood



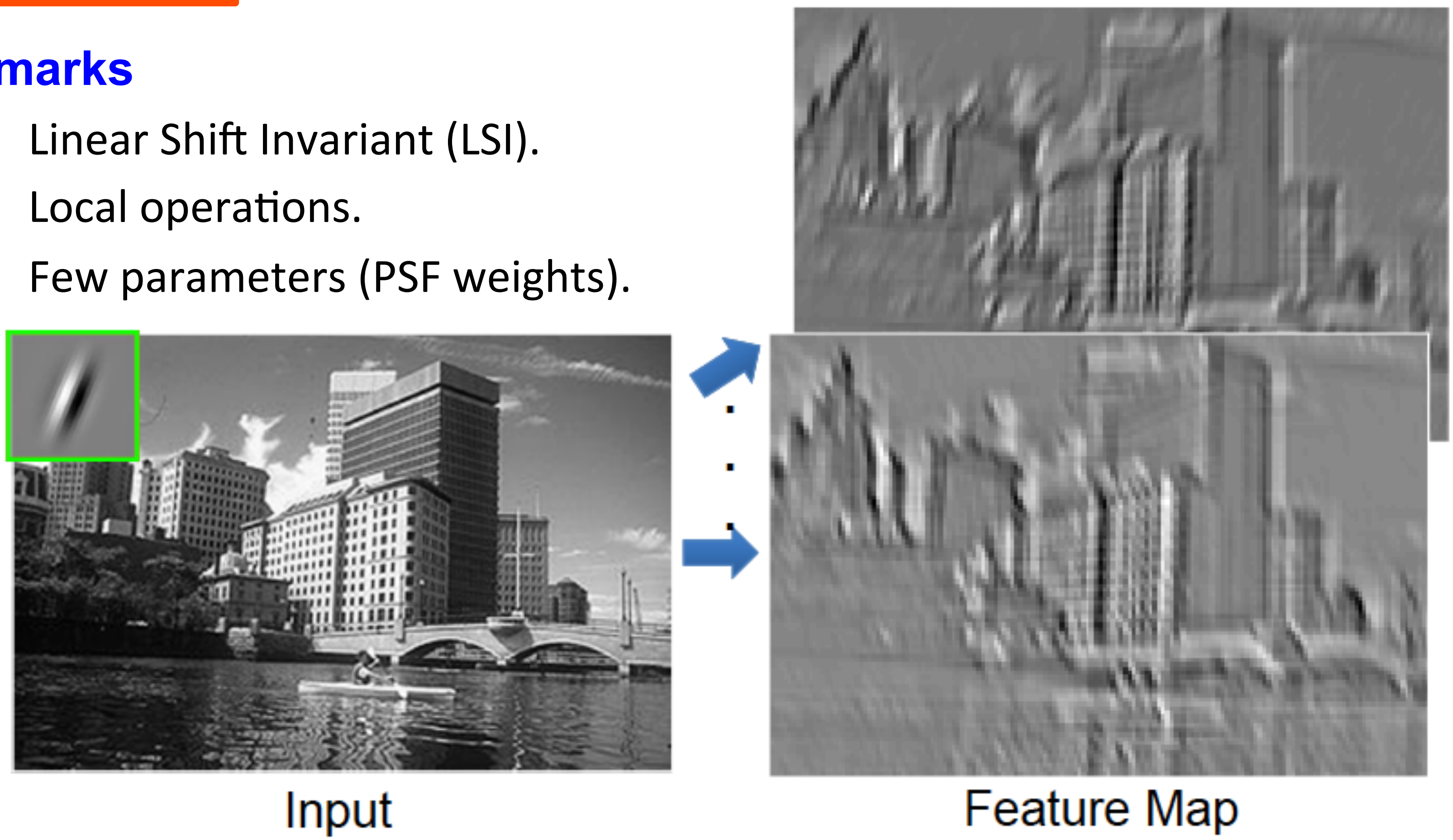


Architecture: Under the hood

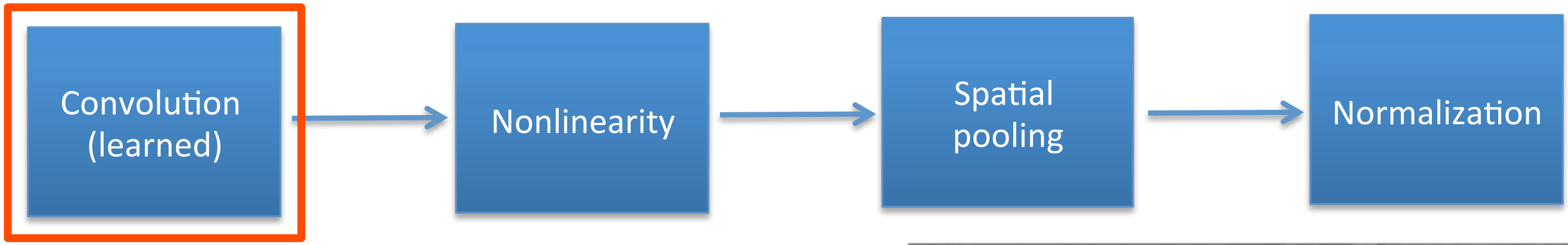


Remarks

- Linear Shift Invariant (LSI).
- Local operations.
- Few parameters (PSF weights).



Architecture: Under the hood

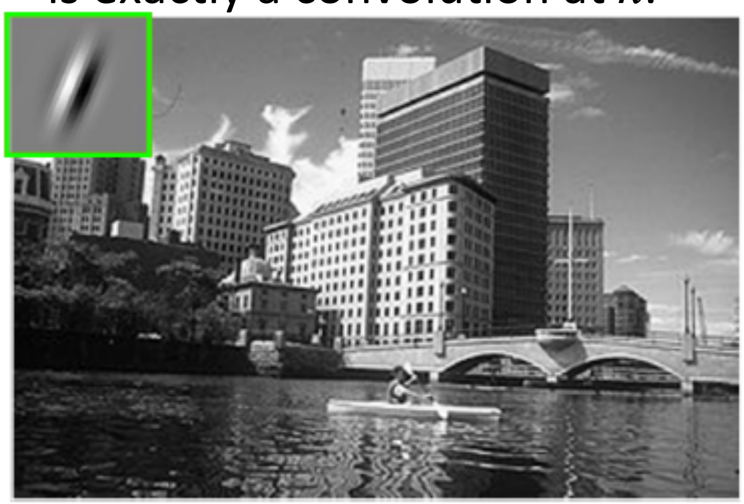


Recall (from our recent past)

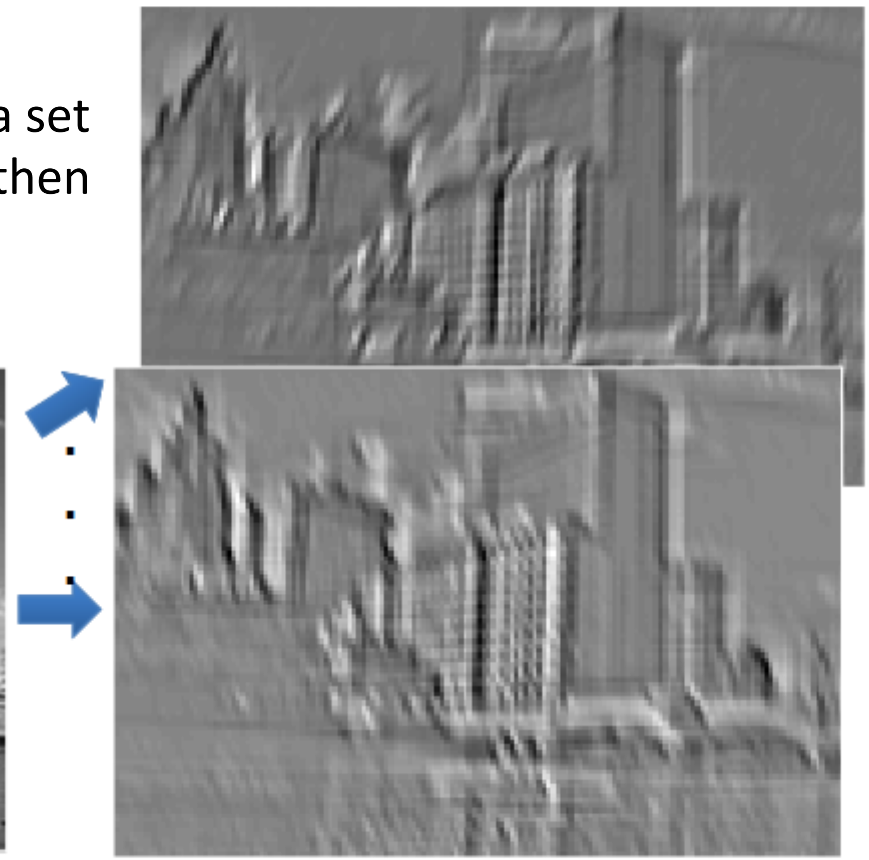
- If we multiply by and sum over a set of weights, w_{kj} at year point, y_j, then

$$net_k = \sum_j w_{kj} y_j$$

is exactly a convolution at k.

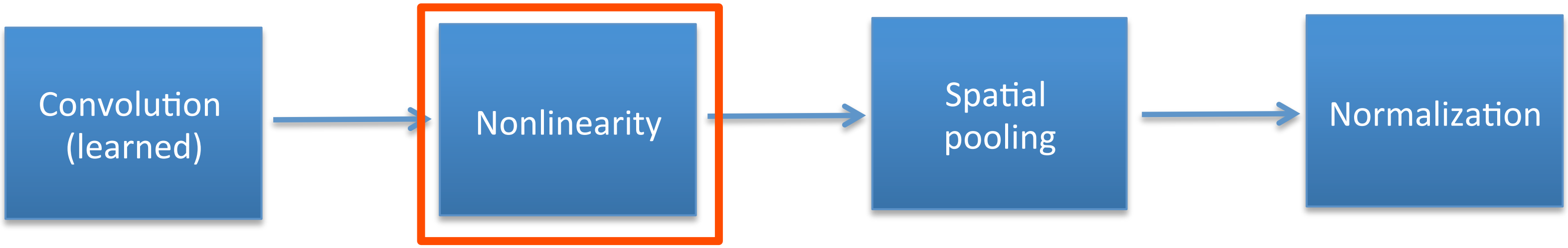


Input

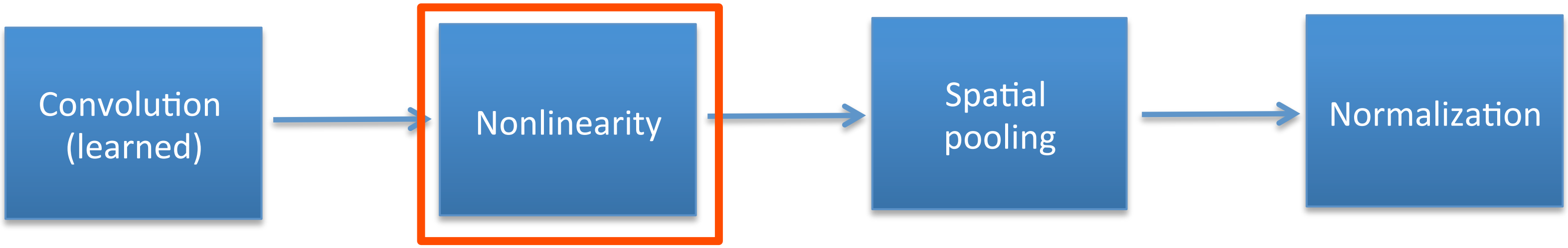


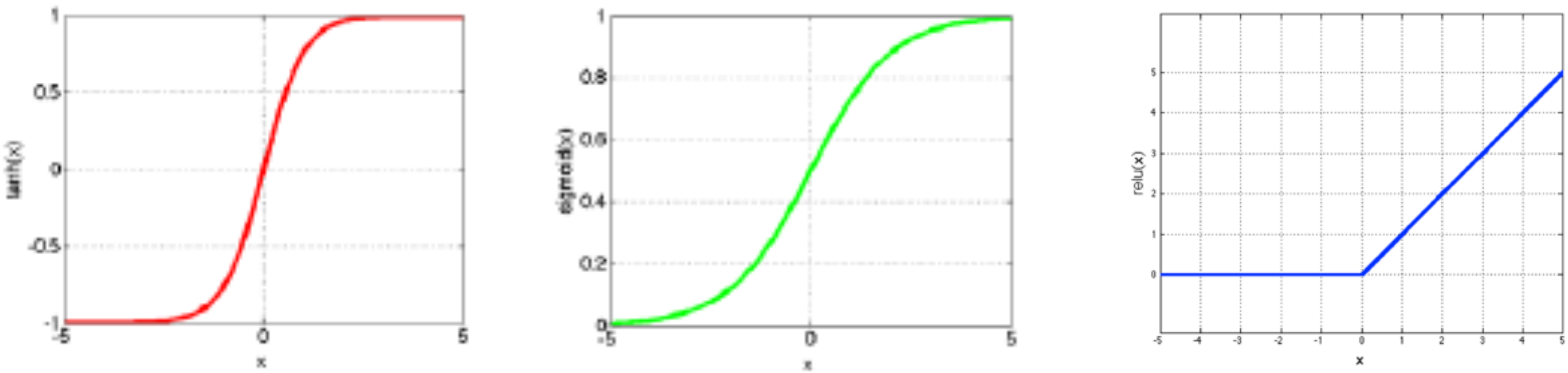
Feature Map

Architecture: Under the hood

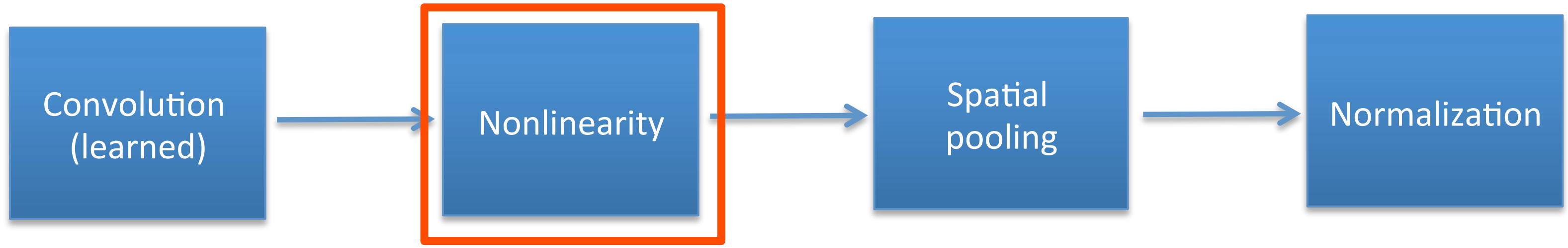


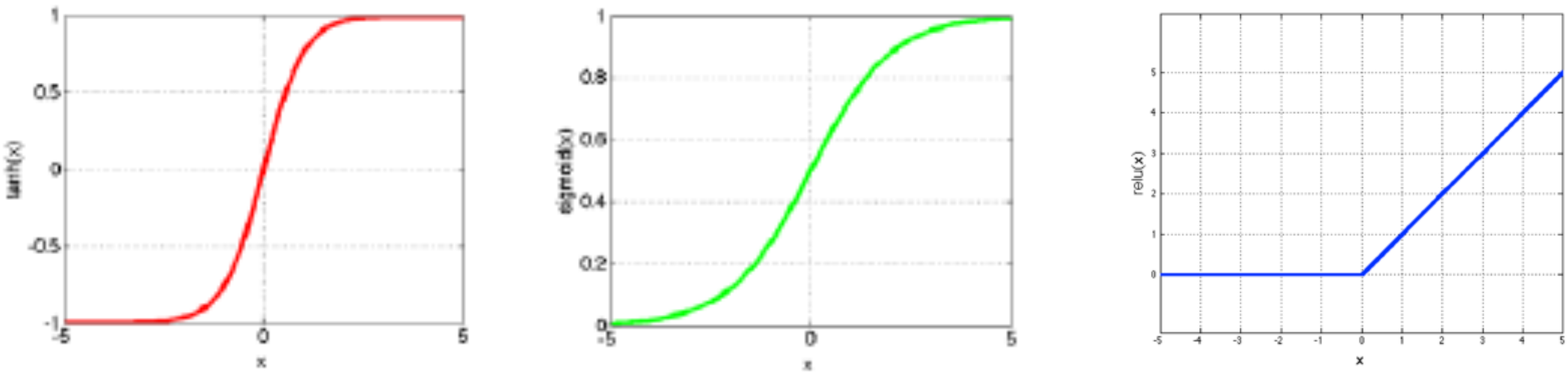
Architecture: Under the hood





Architecture: Under the hood

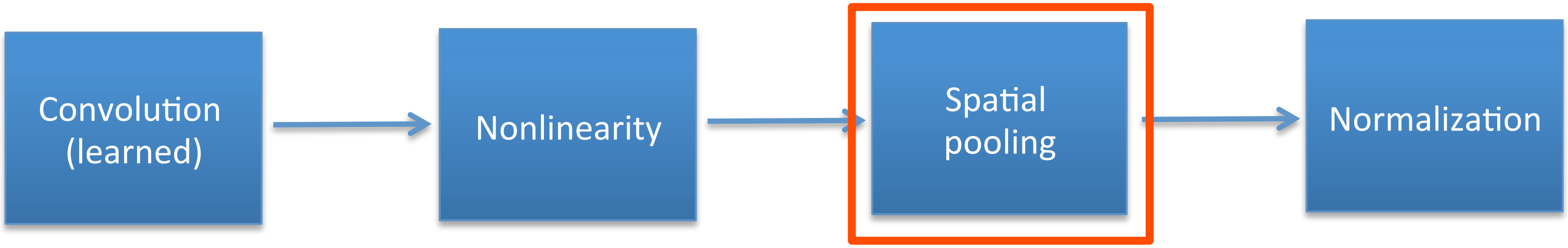




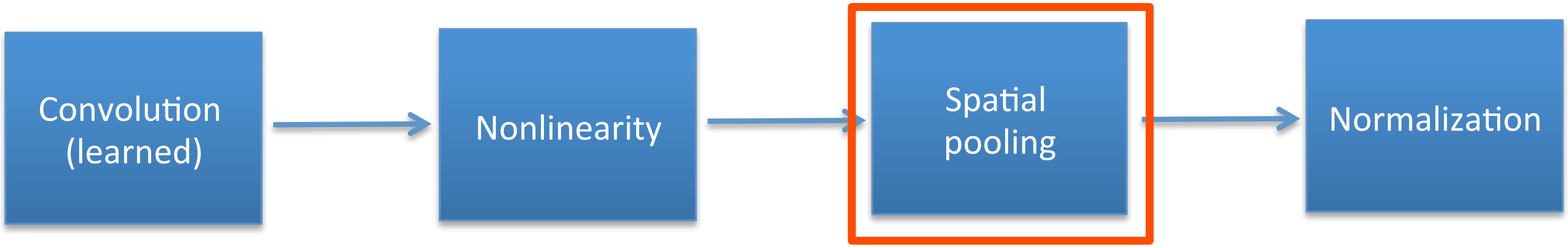
Remarks

- Applied independently per element.
- Enhances strong responses at expense of weak responses.
- Popular choices: tanh, sigmoid, rectified linear unit (ReLU).

Architecture: Under the hood



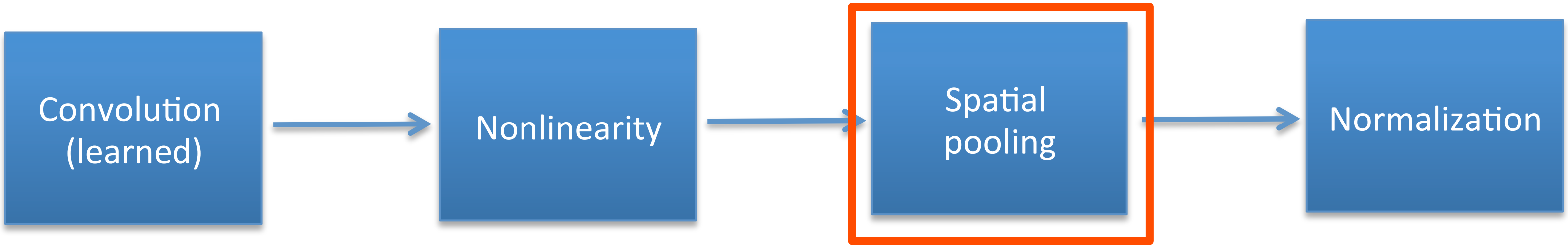
Architecture: Under the hood



Remarks

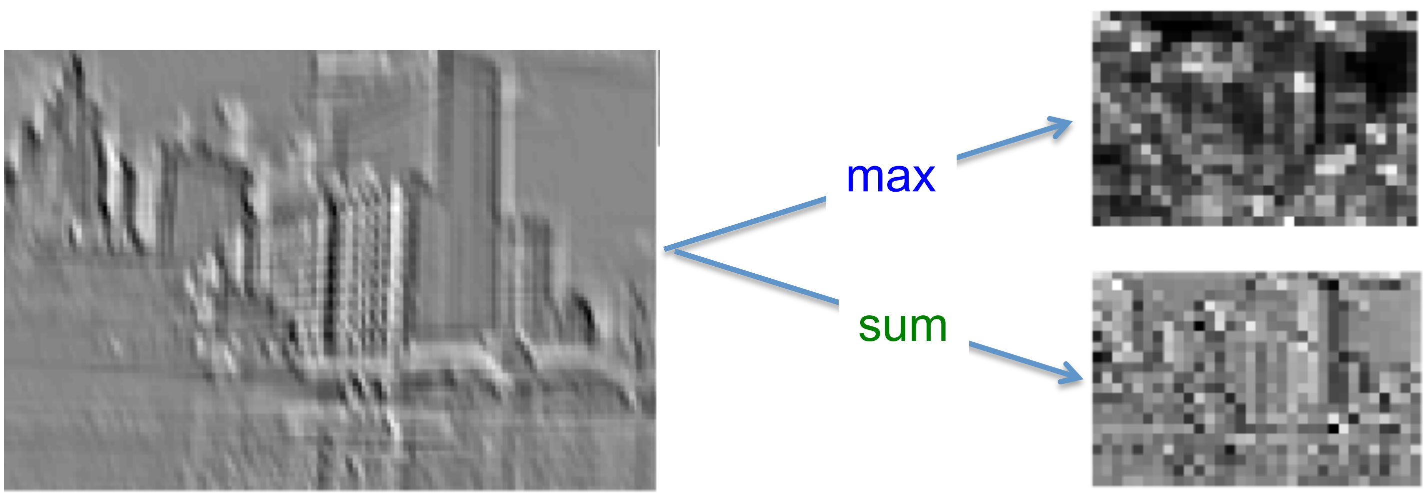
- Role of pooling.
  - Invariance to small transformations.
  - Larger support regions “see” more of input.
- Can be overlapped or not.
- Popular choices: sum and max.

Architecture: Under the hood

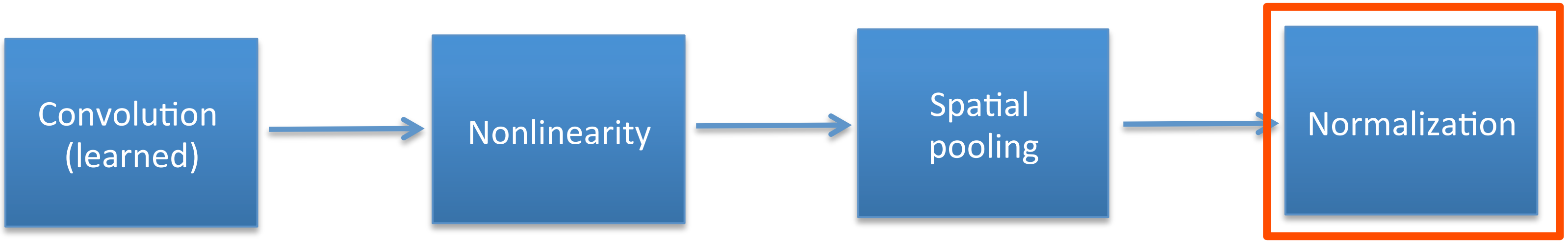


Remarks

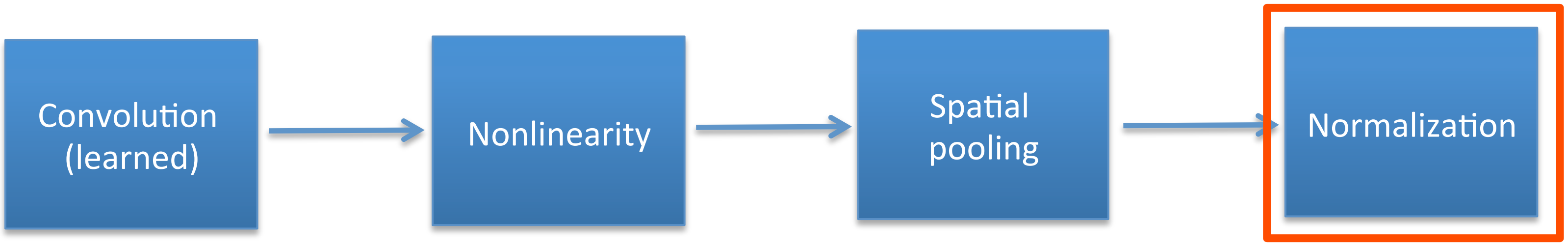
- Role of pooling.
  - Invariance to small transformations.
  - Larger support regions “see” more of input.
- Can be overlapped or not.
- Popular choices: sum and max.



Architecture: Under the hood



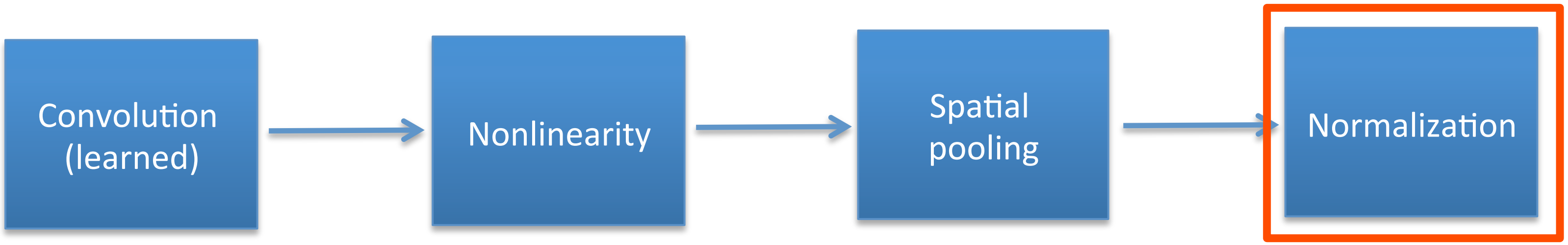
Architecture: Under the hood



Remarks

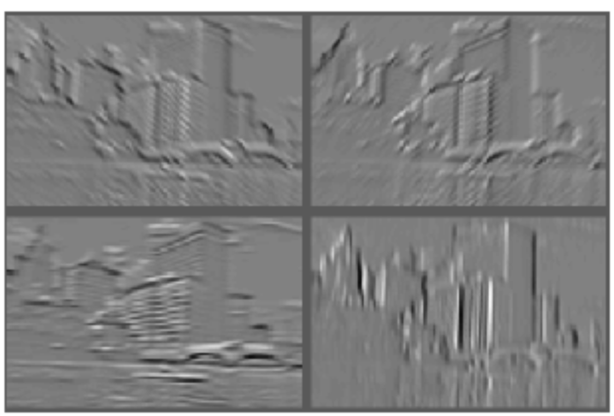
- Role of normalization:
  - Additional photometric invariance.
  - Increased contrast.
- Within and/or across feature maps.
  - Subtractive: Subtract from every value in feature map a weighted average of its neighbors.
  - Divisive: Divide every value in feature map by the sum (or standard deviation) of all feature maps.
- Before or after pooling.

Architecture: Under the hood

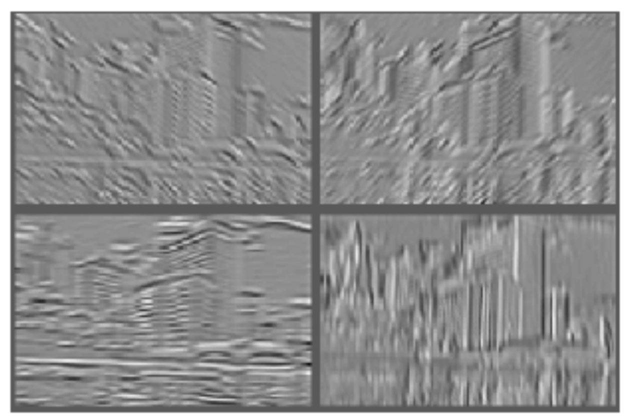


Remarks

- Role of normalization:
    - Additional photometric invariance.
    - Increased contrast.
- Within and/or across feature maps.
- Before or after pooling.



Feature Maps



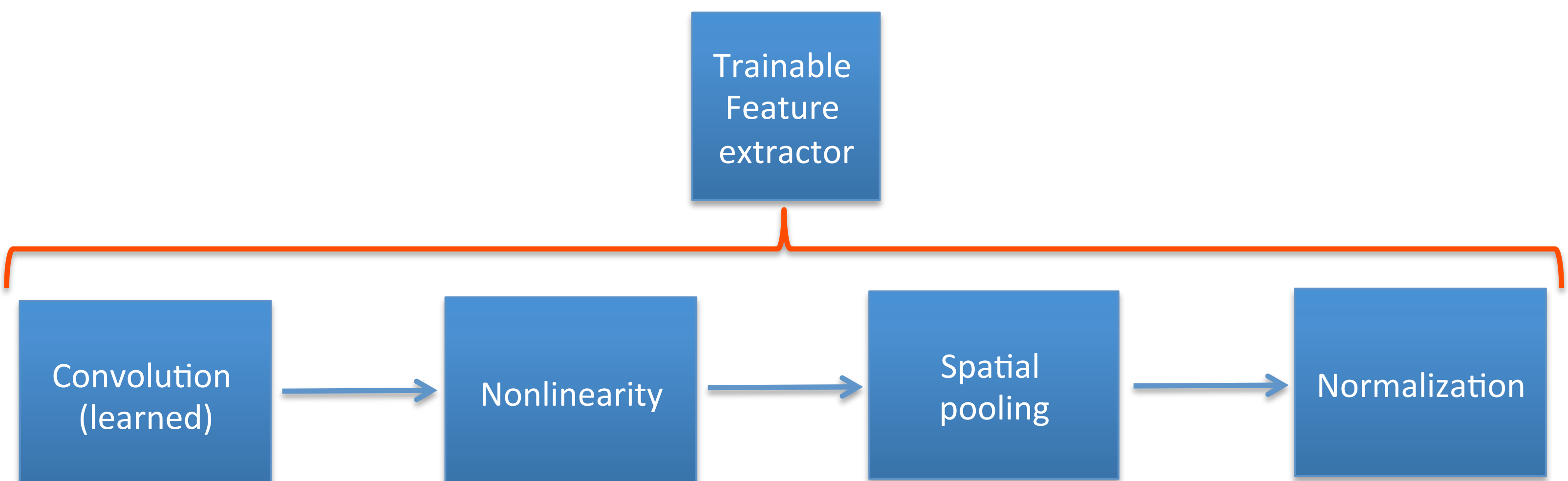
Feature Maps
After Contrast Normalization

Architecture: Feedforward extraction

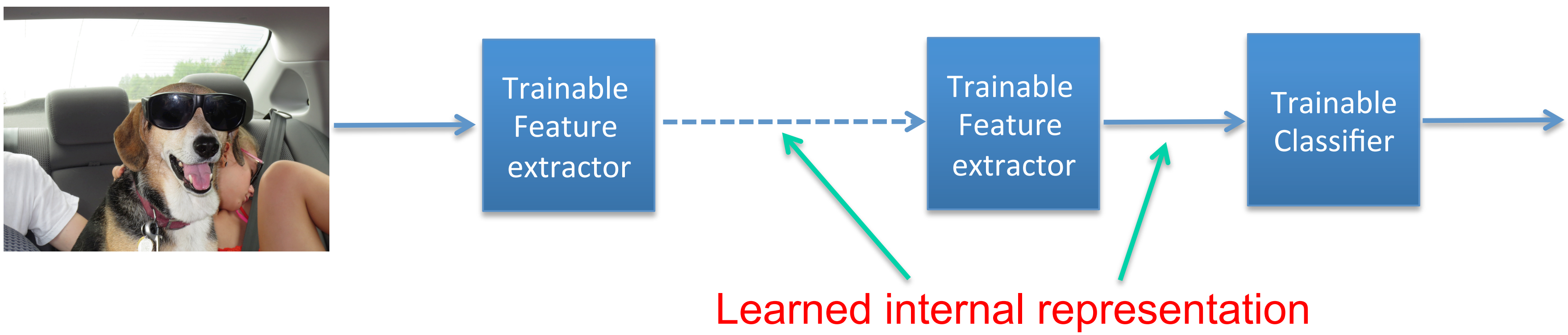
Looking under the hood

- Our overall architecture is

- What is inside each Feature Extractor.



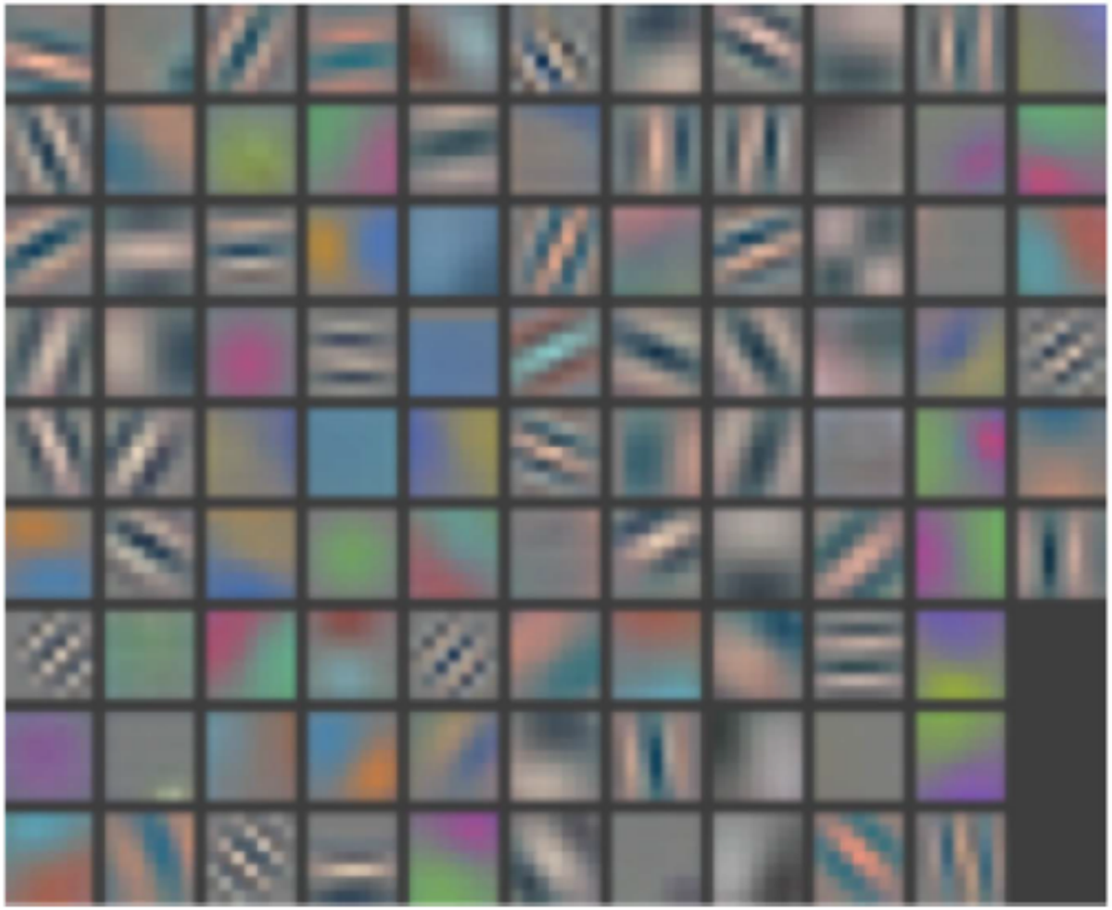
Architecture: Visualizing ConvNets



Questions

- What can be said about the nature of the learned representations?
- What types of information do they capture?

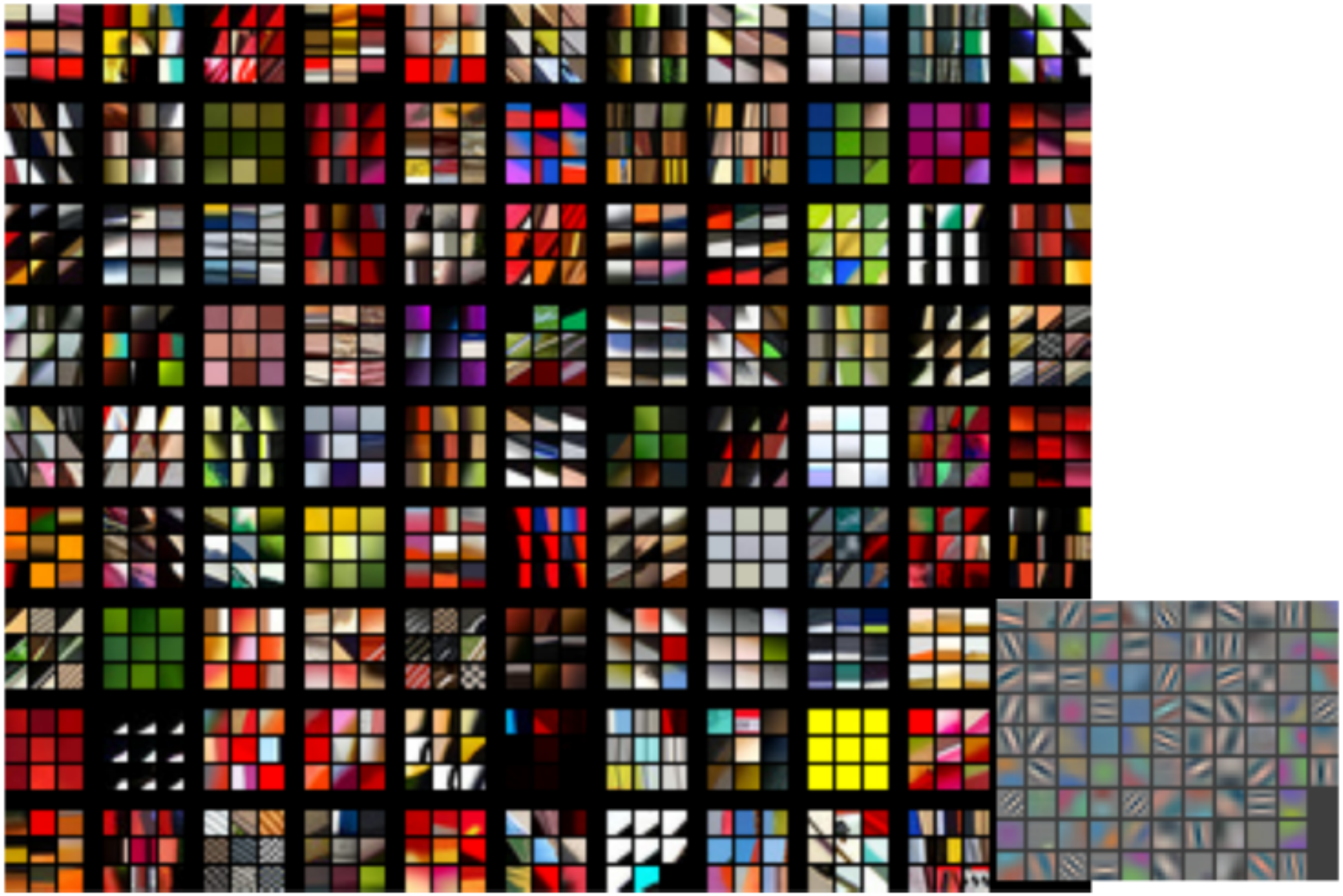
Architecture: Visualizing ConvNets



Layer 1

- Learned PSFs appear as oriented bandpass kernels.

Architecture: Visualizing ConvNets



Layer 1

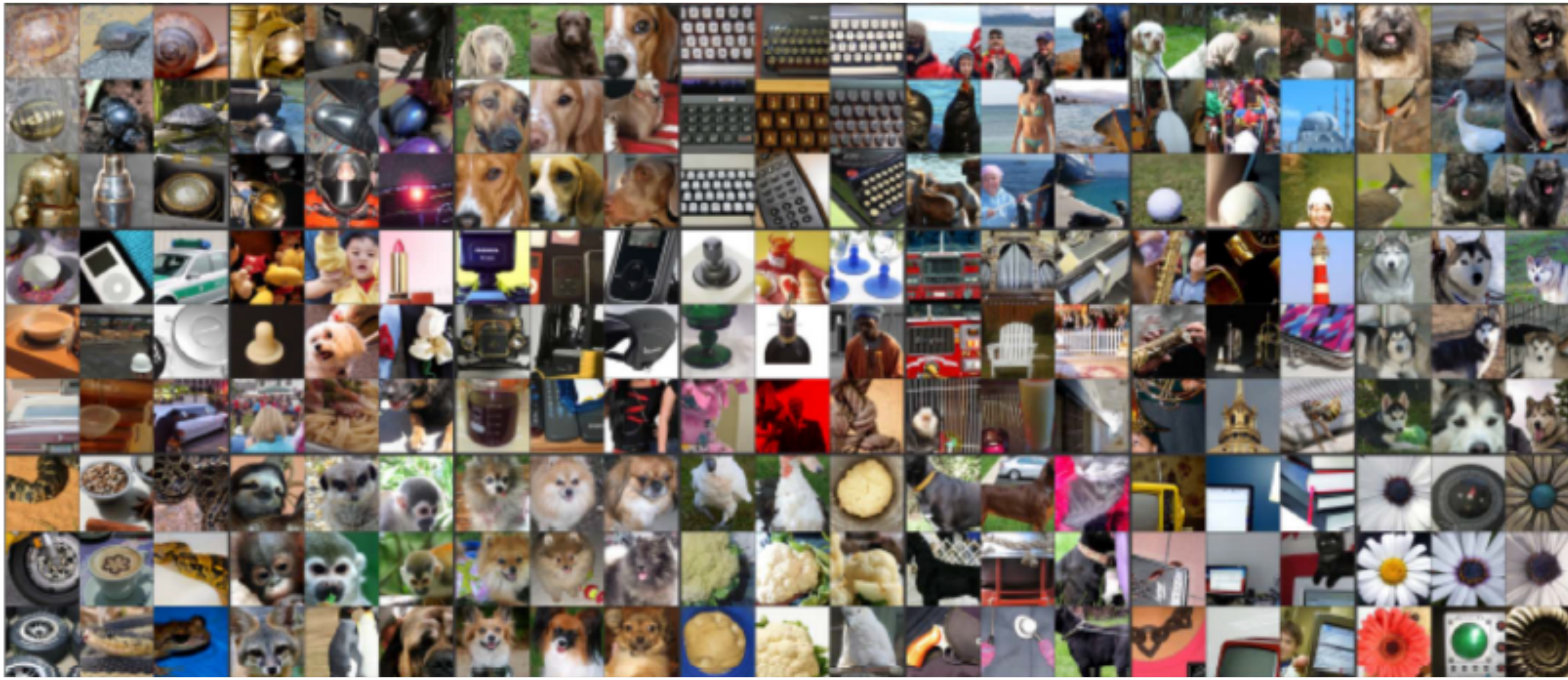
- Top nine patches from images giving maximal response for a filter following training.

Architecture: Visualizing ConvNets

Layer 2

- Top nine patches from images giving maximal response for a filter following training.

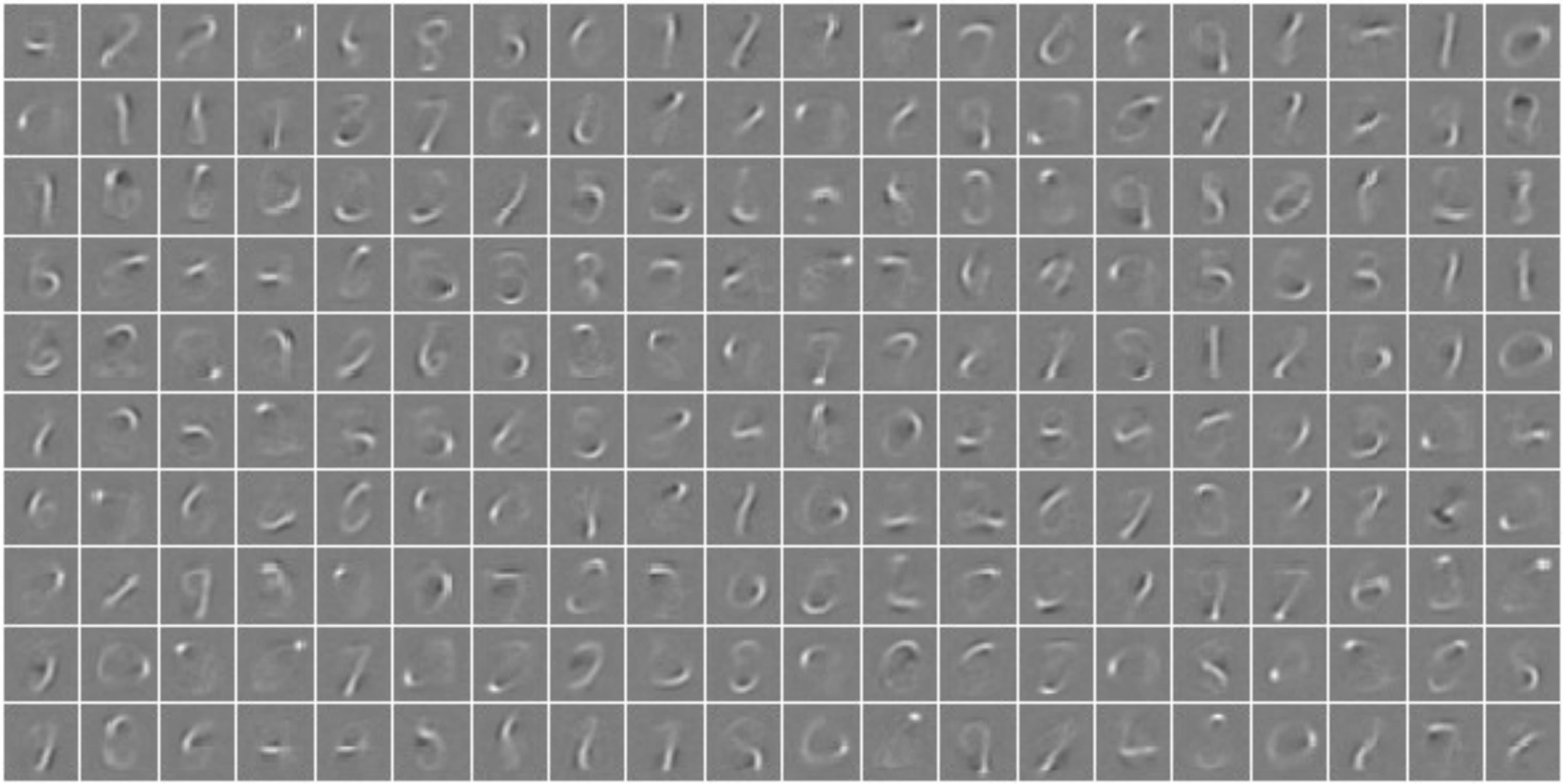
Architecture: Visualizing ConvNets



Layer 5

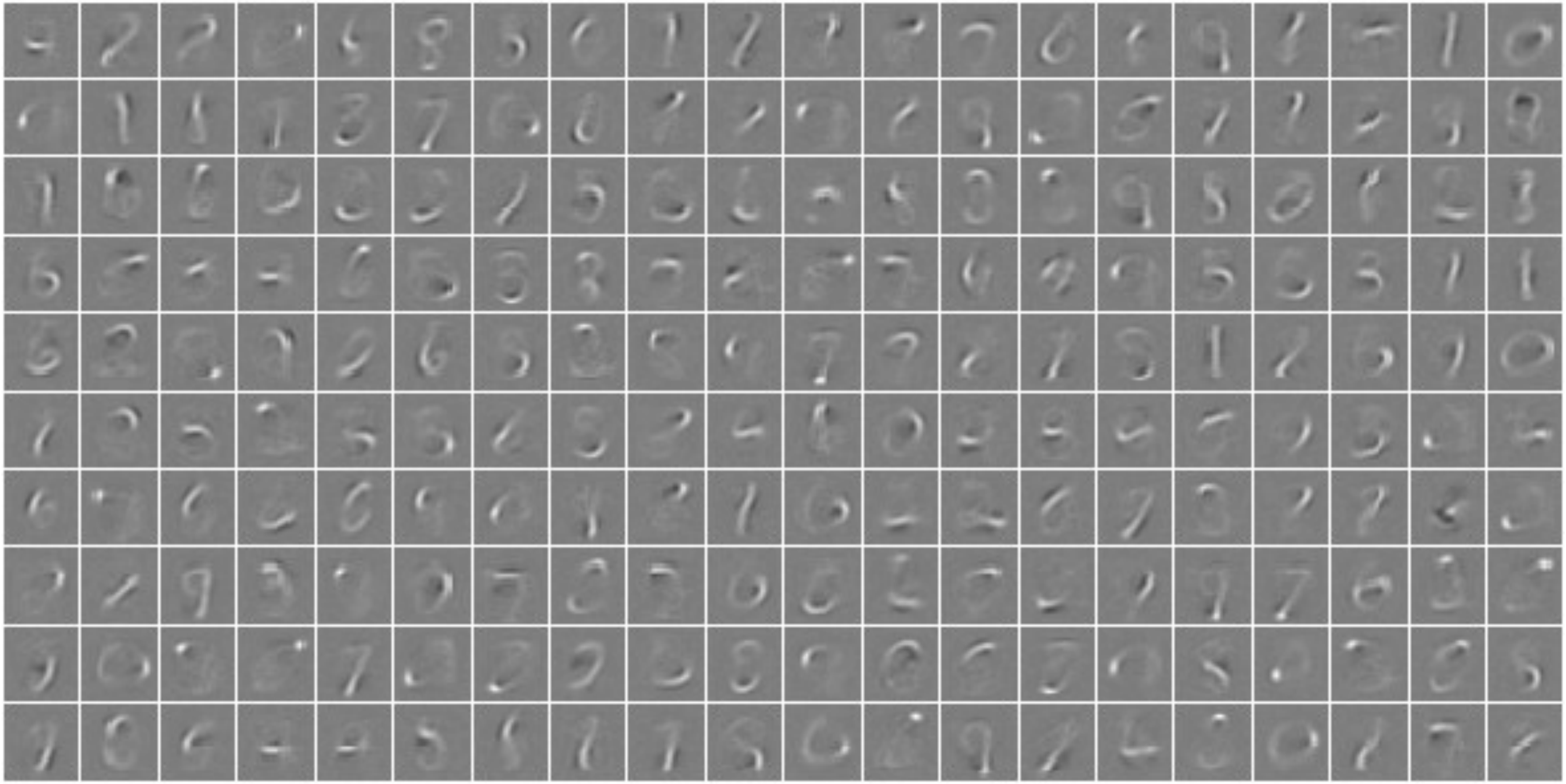
- Top nine patches from images giving maximal response for a filter following training.

Architecture: Visualizing ConvNets



Learned feature maps depend on training data

Architecture: Visualizing ConvNets

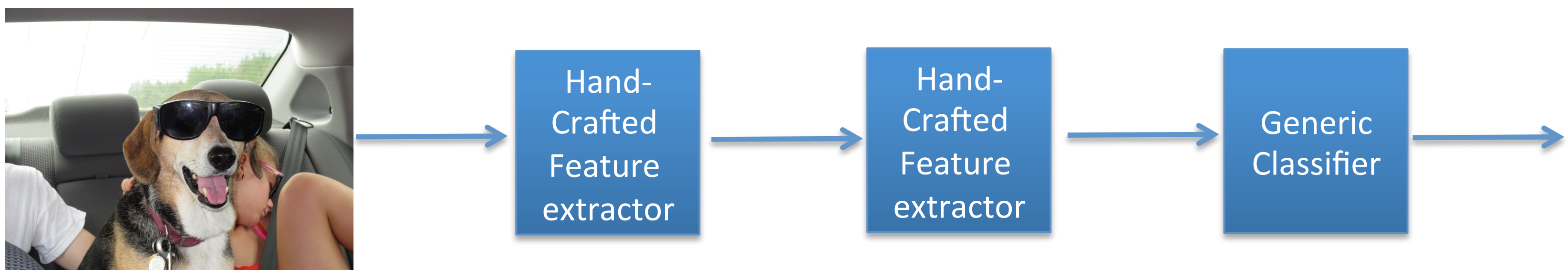


Learned feature maps depend on training data

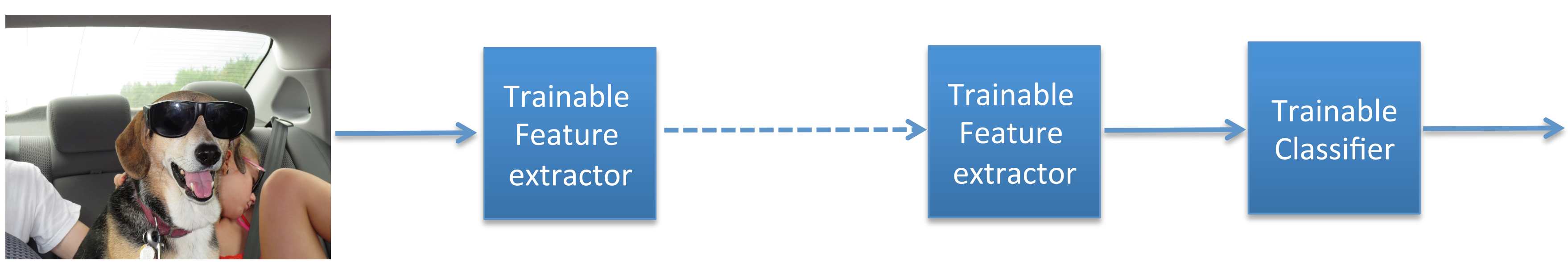
- Maps trained on handwritten digits.

Architecture: ConvNets vs. traditional

Traditional architecture



ConvNet architecture



Outline

- Introduction
- Background
- Architecture
- **Examples**
- Summary

Examples

Sampling of ConvNet success stories

- Handwritten digits
- Simple recognition
- Generic object recognition
- Face detection
- Driving

Examples: ConvNet success stories

Handwritten digit recognition

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

Examples: ConvNet success stories

Handwritten digit recognition

- MNIST Handwritten Digit Dataset
- 60,000 training samples
- 10,000 test samples
- Ciresan et al. 2011: 0.17% error

Examples: ConvNet success stories

Traffic sign recognition

- Human error rate: 1.16%

Examples: ConvNet success stories

Traffic sign recognition

- Human error rate: 1.16%
- Ciresan et al. 2011: 0.56% error

Examples: ConvNet success stories

Traffic sign recognition

- Human error rate: 1.16%
- Ciresan et al. 2011: 0.56% error

Strong performance in simple recognition tasks

- But less good with more complicated datasets (e.g., Caltech-101)

Examples: ConvNet success stories

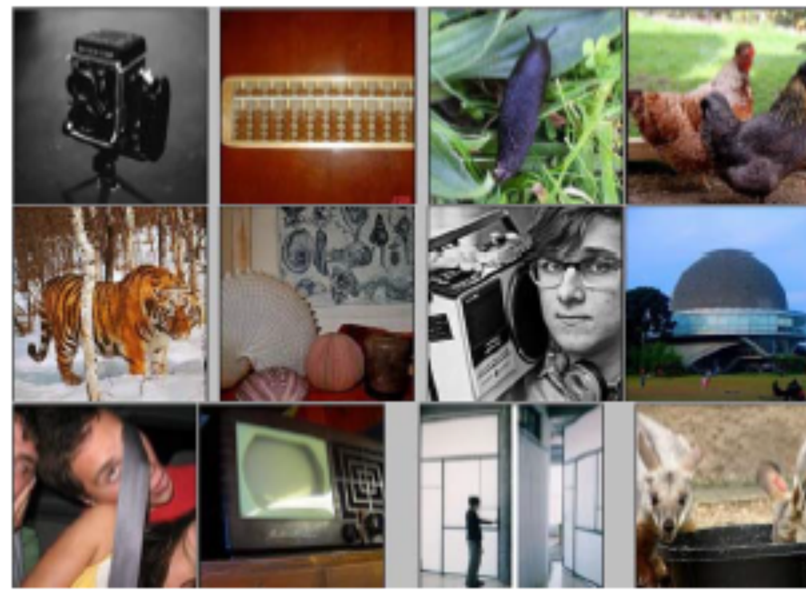
Traffic sign recognition

- Human error rate: 1.16%
- Ciresan et al. 2011: 0.56% error

Strong performance in simple recognition tasks

- But less good with more complicated datasets (e.g., Caltech-101)
- Until recently…

IMAGENET

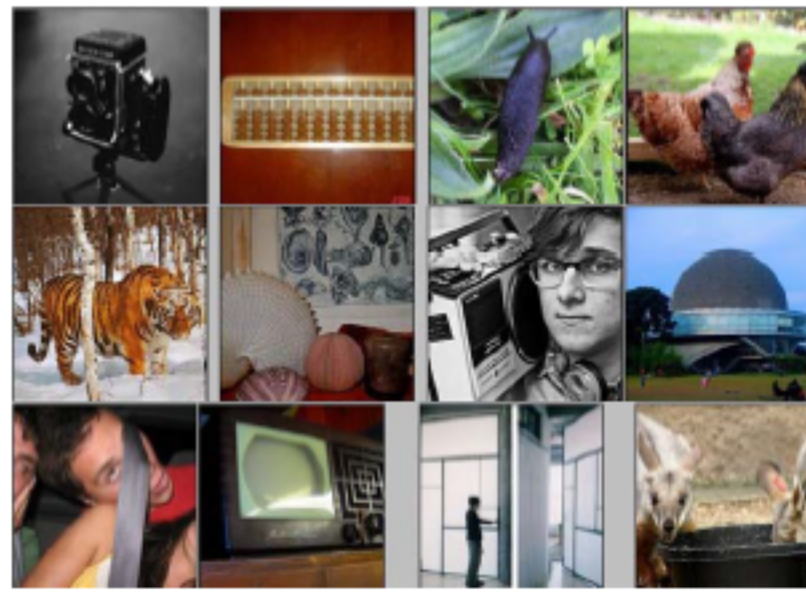


[Deng et al. CVPR 2009]

Examples: ConvNet success stories

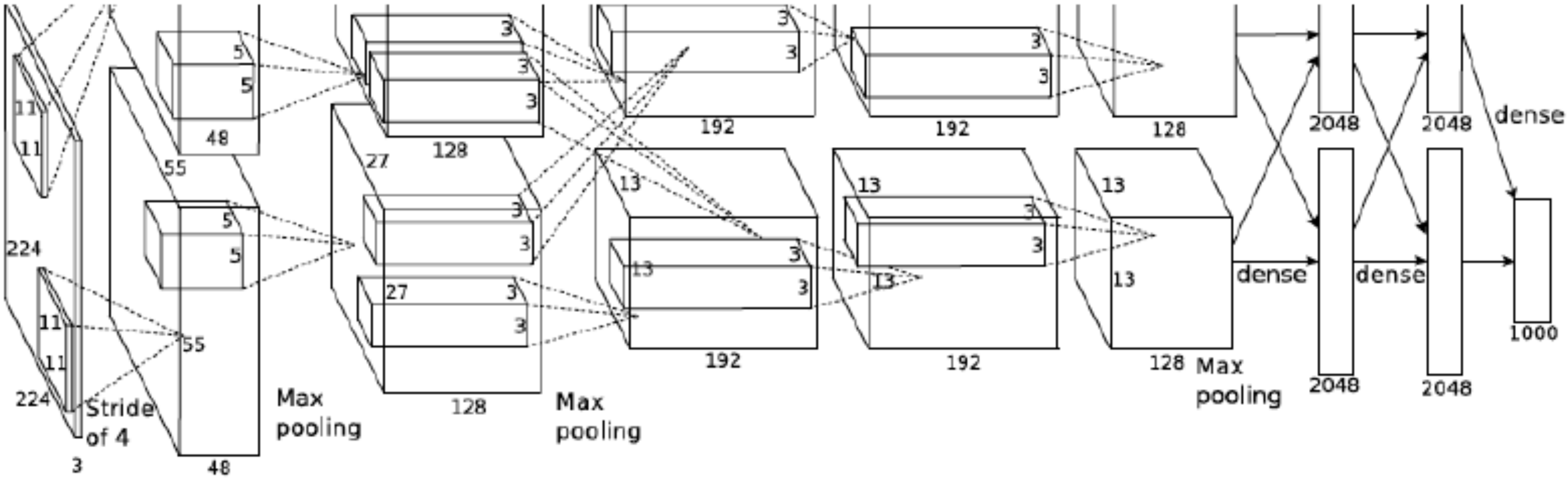
ImageNet Challenge 2012

- ~14 million labeled image; 20K classes
- Gathered from Internet
- Human labels via Amazon Turk
- Challenge: 1.2 million test images; 1000 classes



[Deng et al. CVPR 2009]

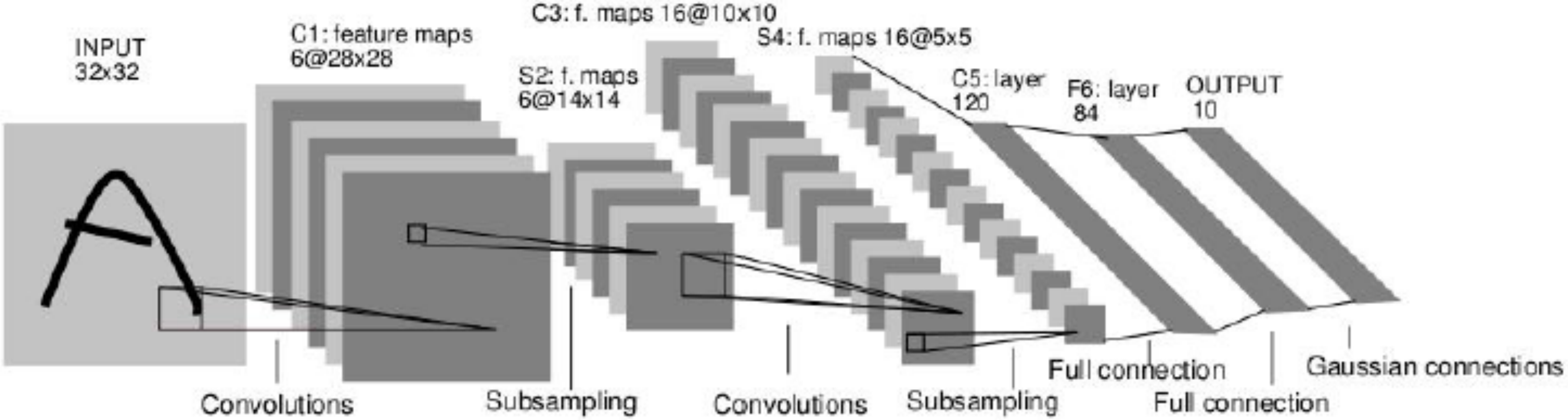
Examples: ConvNet success stories



Krizhevsky, Sutskever & Hinton 2012

- Similar to LeCun et al. 1998, but …

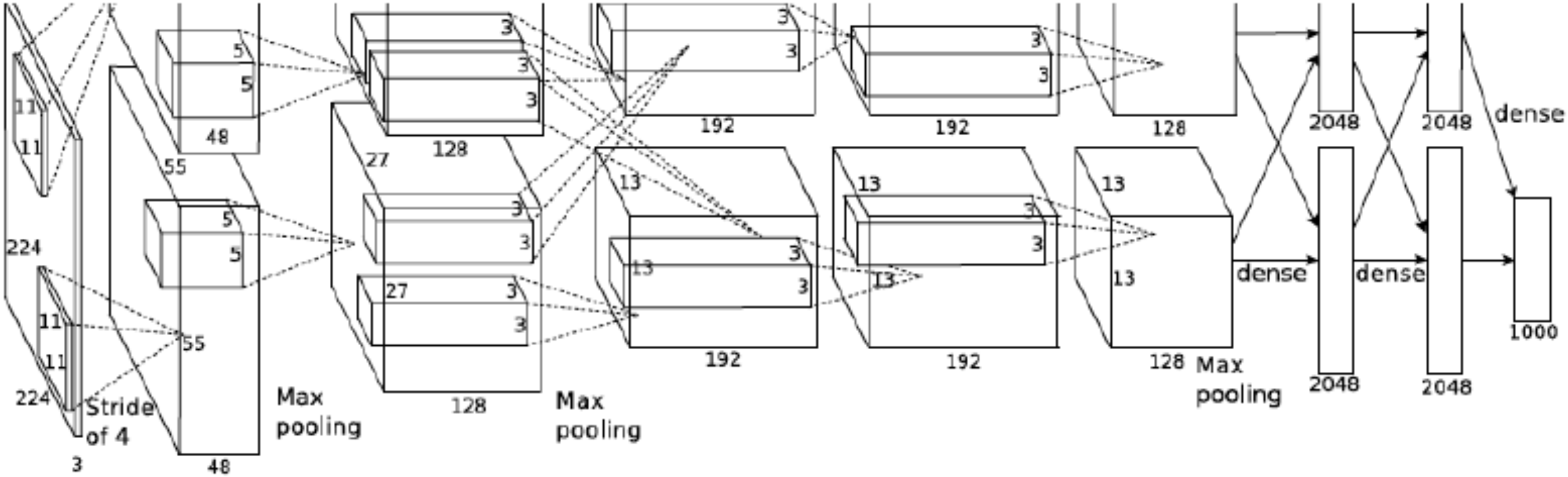
Examples: ConvNet success stories



Krizhevsky, Sutskever & Hinton 2012

- Similar to LeCun et al. 1998, but …

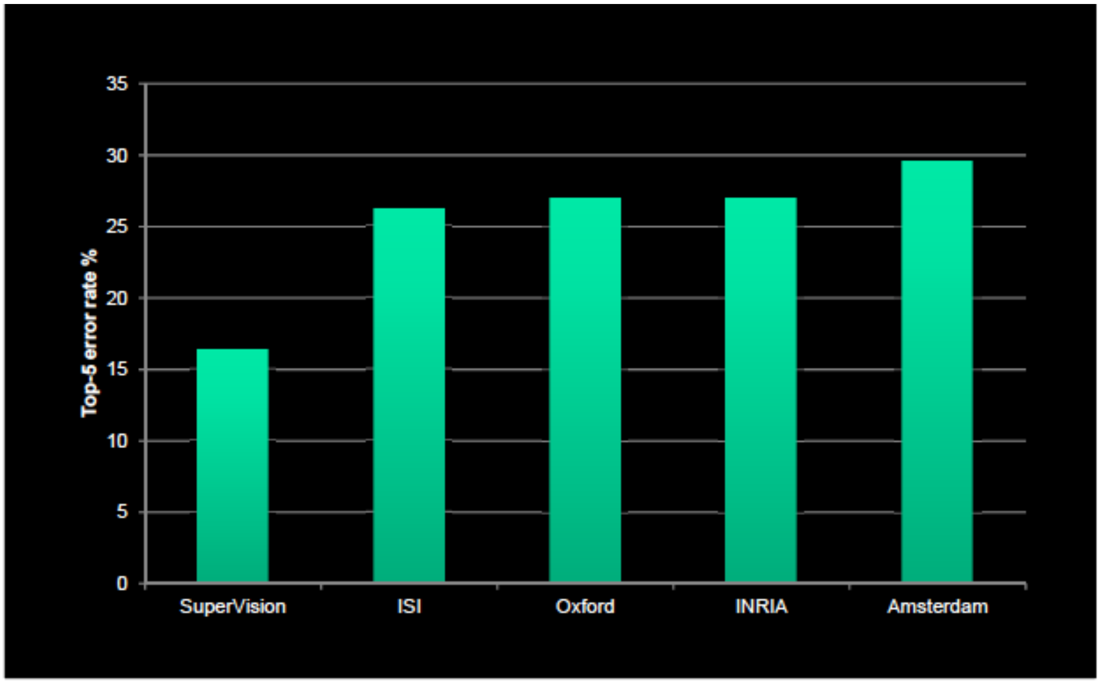
Examples: ConvNet success stories



Krizhevsky, Sutskever & Hinton 2012

- Similar to LeCun et al. 1998, but …
- Bigger model: 7 hidden layers, 650,000 units, 60,000,000 params
- More training data: 10^6 vs. 10^3 images

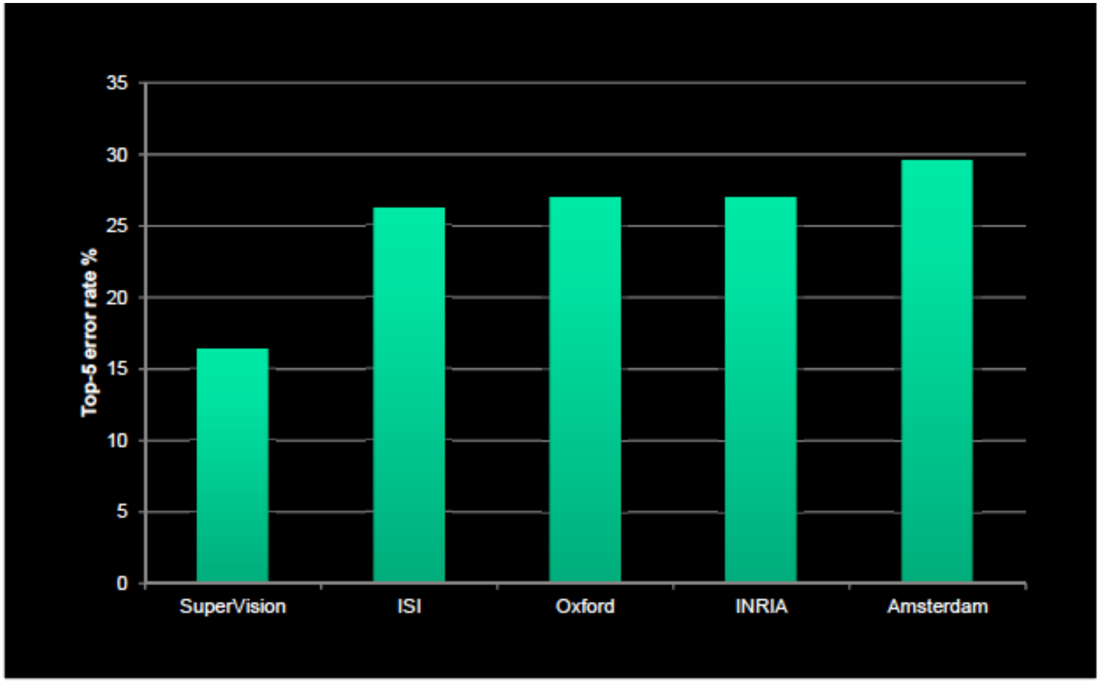
Examples: ConvNet success stories



ImageNet Challenge 2012: Results

- Top performer: Krizhevsky et al: 16.4% error.
- Next best (non-ConvNet): 26.2% error.

Examples: ConvNet success stories



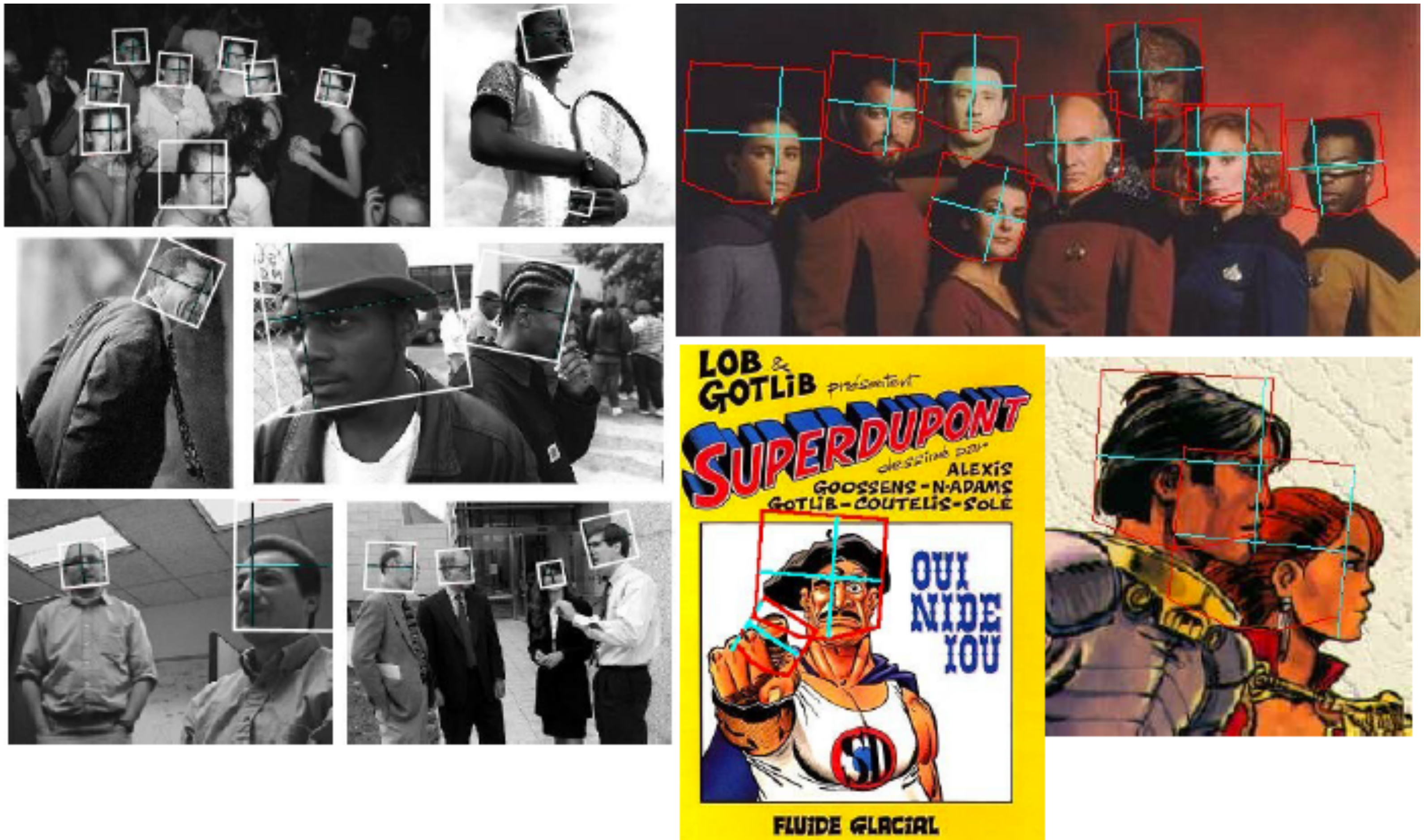
ImageNet Challenge 2012: Results

- Top performer: Krizhevsky et al: 16.4% error.
- Next best (non-ConvNet): 26.2% error.

Why?

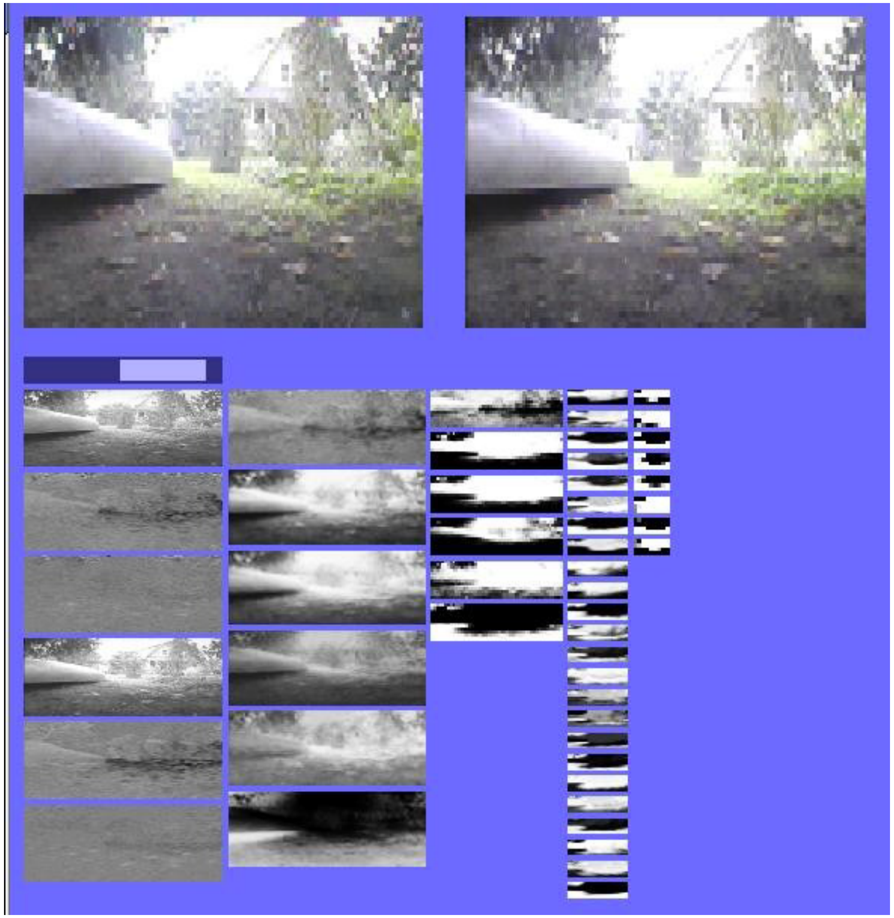
- More training data; more computational resources.

Examples: ConvNet success stories



Face detection & pose recognition: Osadchy et al., 2007

Examples: ConvNet success stories



Driving: LeCun et al. 2005

- Mobile platform with two cameras
- Network trained from recorded stereo video + human steering angles.
- Result maps stereo images to steering angles to avoid obstacles.

Examples: Industry

Industry labs actively pursuing ConvNets include

- Facebook: Face and object recognition
- France Telcom: Face detection, HCI, handheld apps
- Google: OCR, face & license plate removal from StreetView
- Microsoft: OCR, handwriting and speech recognition
- NEC: Cancer cell detection, automotive apps
- Vident: Video surveillance

Outline

- Introduction
- Background
- Architecture
- Examples
- **Summary**

Summary

ConvNets provide

- Hierarchical representation for invariant pattern recognition that is
  - learning based
  - biologically inspired
  - recently enabled through increased computation power and large amounts of training data

Summary

ConvNets provide

- Hierarchical representation for invariant pattern recognition that is
  - learning based
  - biologically inspired
  - recently enabled through increased computation power and large amounts of training data
- Success on a variety of tasks
  - Here we concentrate on vision, …
  - … but many others as well (e.g., speech recognition, medical, ...)

Summary

ConvNets provide

- Hierarchical representation for invariant pattern recognition that is
  - learning based
  - biologically inspired
  - recently enabled through increased computation power and large amounts of training data
- Success on a variety of tasks
  - Here we concentrate on vision, …
  - … but many others as well (e.g., speech recognition, medical, ...)

Current limitations include

- Need for large amounts of training data and computational resources
- Little ability to learn without supervision
- Lack of (short term) memory
- Lack of reasoning mechanisms
- Lack of theoretical understanding on what they represent

Outline

- Introduction
- Background
- Architecture
- Examples
- Summary