Convolutional networks

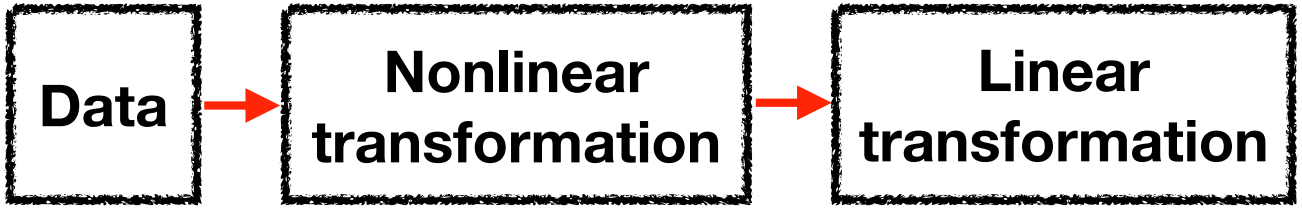
Laurens van der Maaten

TU Delft

Deep learning

- *Deep learning* is what people called “neural networks” in the 80s and 90s

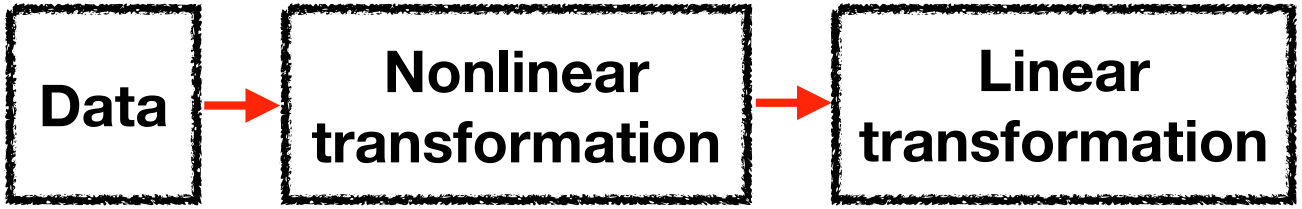
- If you think about a kernel SVM, it basically does the following:



Deep learning

- *Deep learning* is what people called “neural networks” in the 80s and 90s

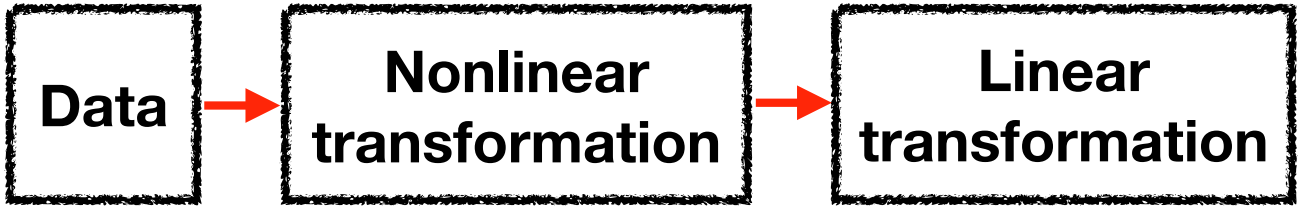
- If you think about a kernel SVM, it basically does the following:



  - This is a *shallow* model: there’s only one latent feature representation

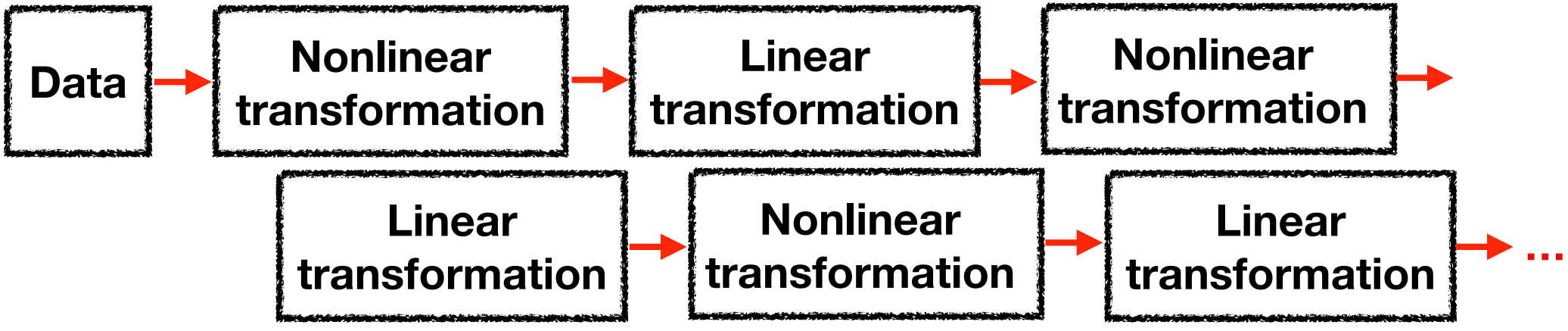
Deep learning

- *Deep learning* is what people called "neural networks" in the 80s and 90s
- If you think about a kernel SVM, it basically does the following:



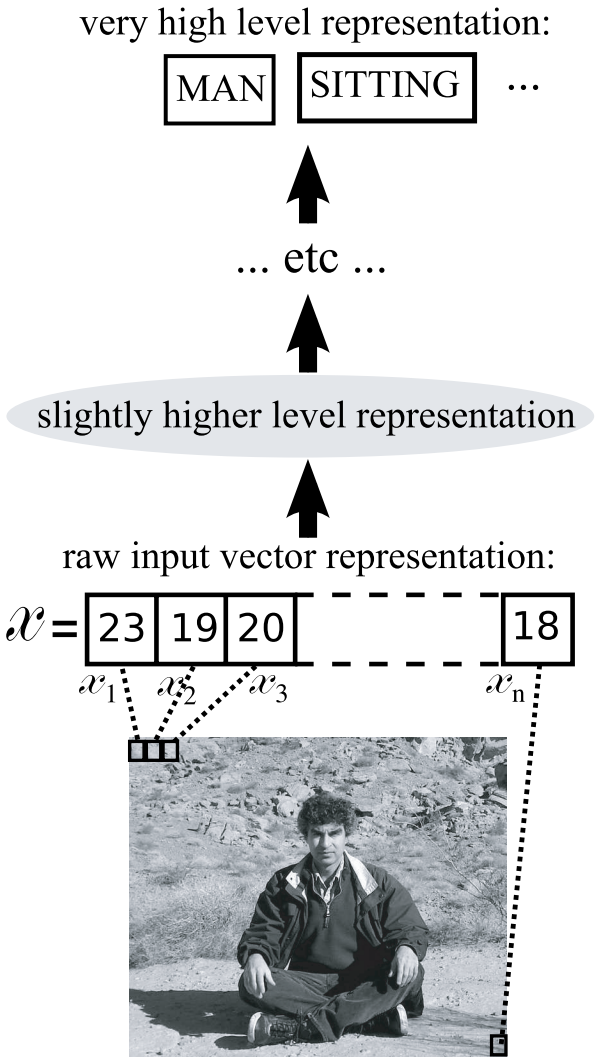
  - This is a *shallow* model: there's only one latent feature representation

- A *deep* model learns multiple, increasingly high-level feature representations:



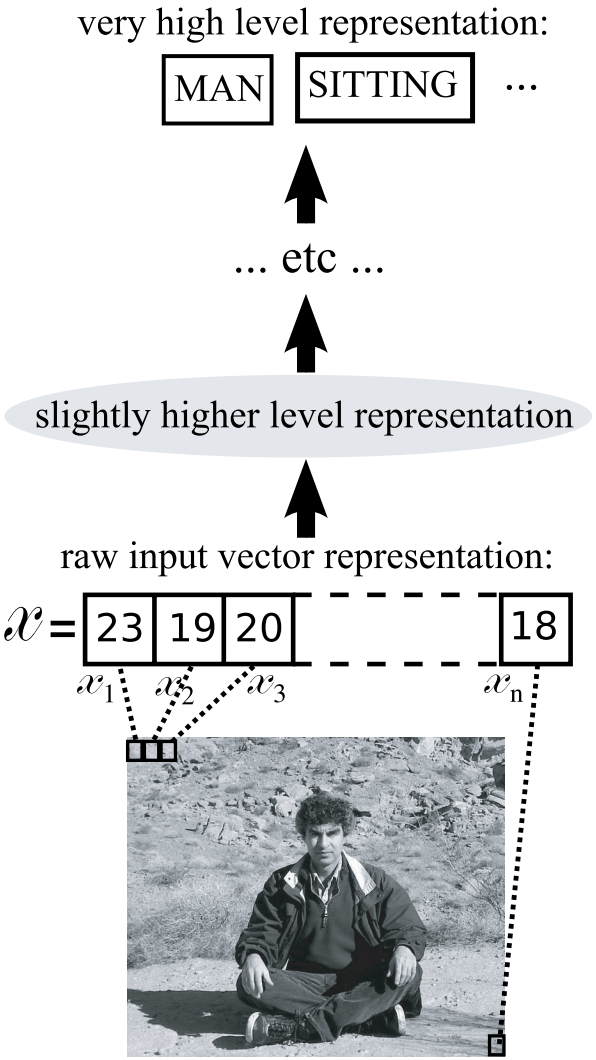
Deep learning

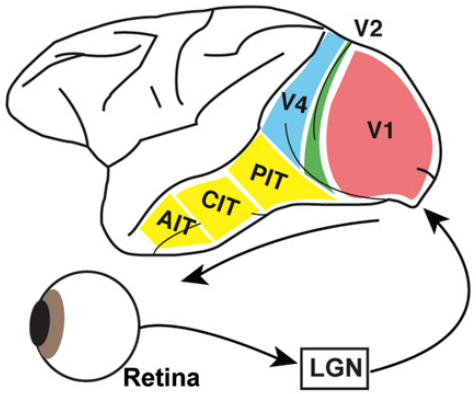
- Central idea: build increasingly high-level *representations* of the data

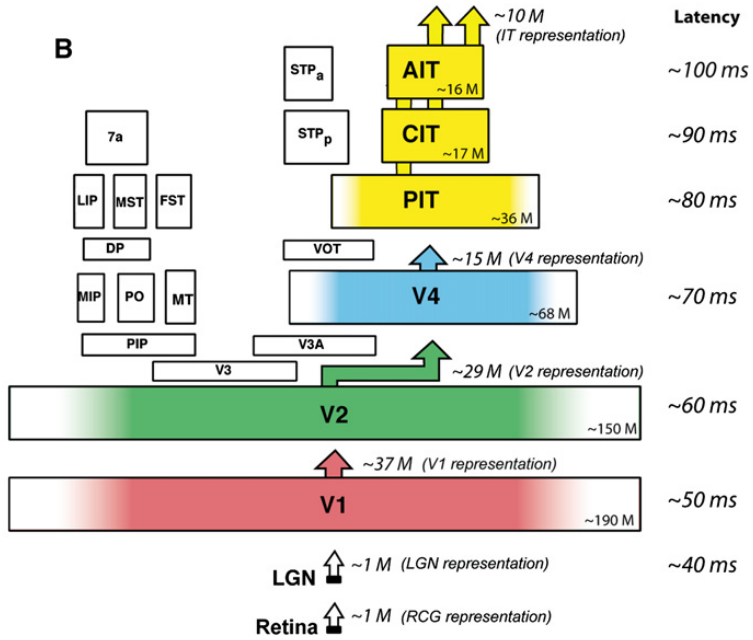


Deep learning

- Central idea: build increasingly high-level *representations* of the data

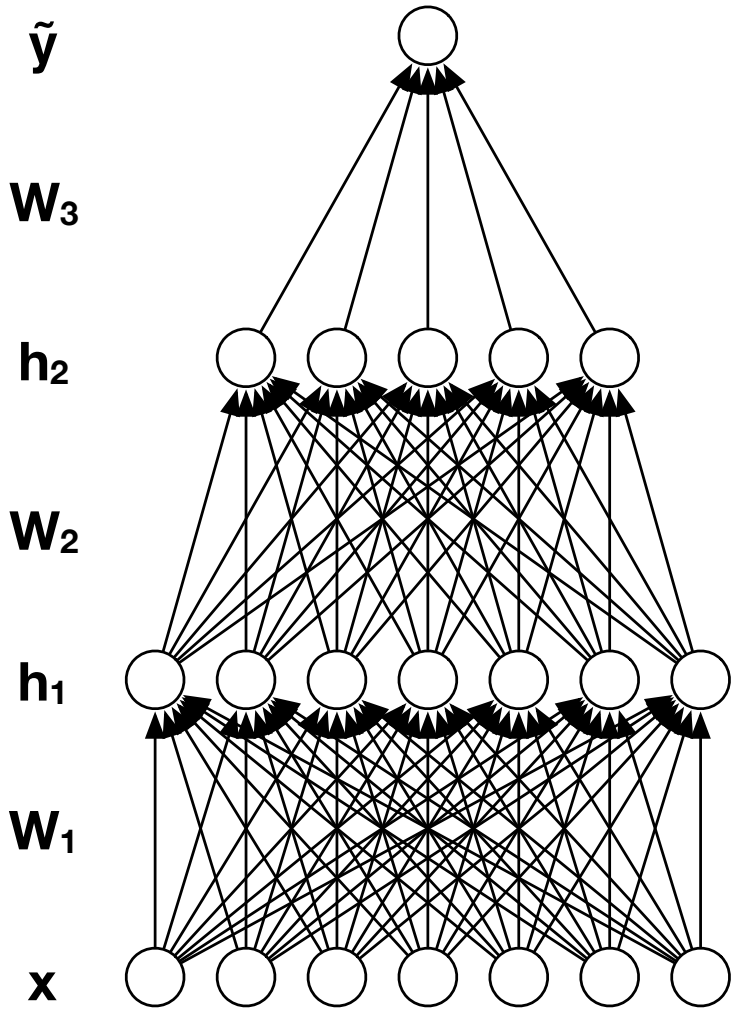






Feedforward network

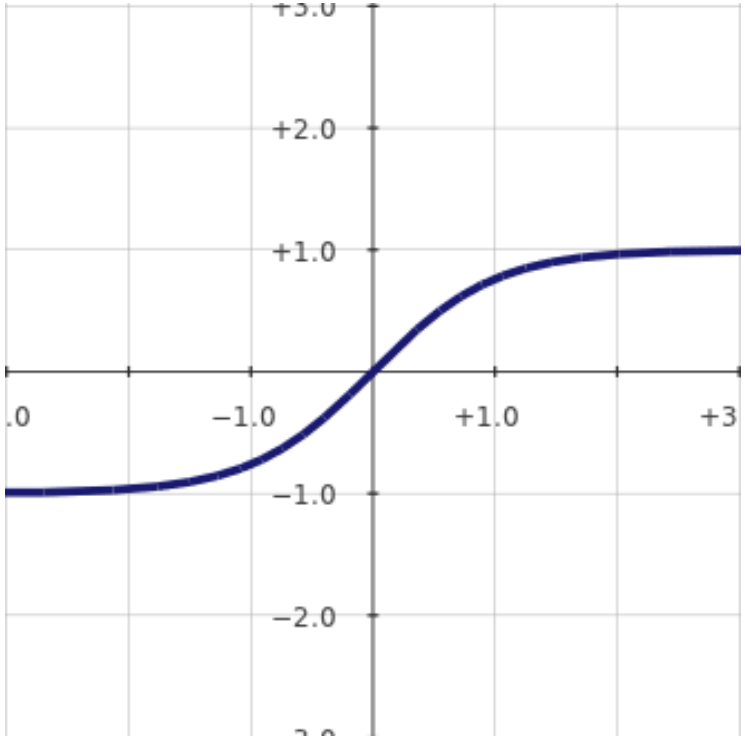
- Simple feedforward networks have the following general structure:

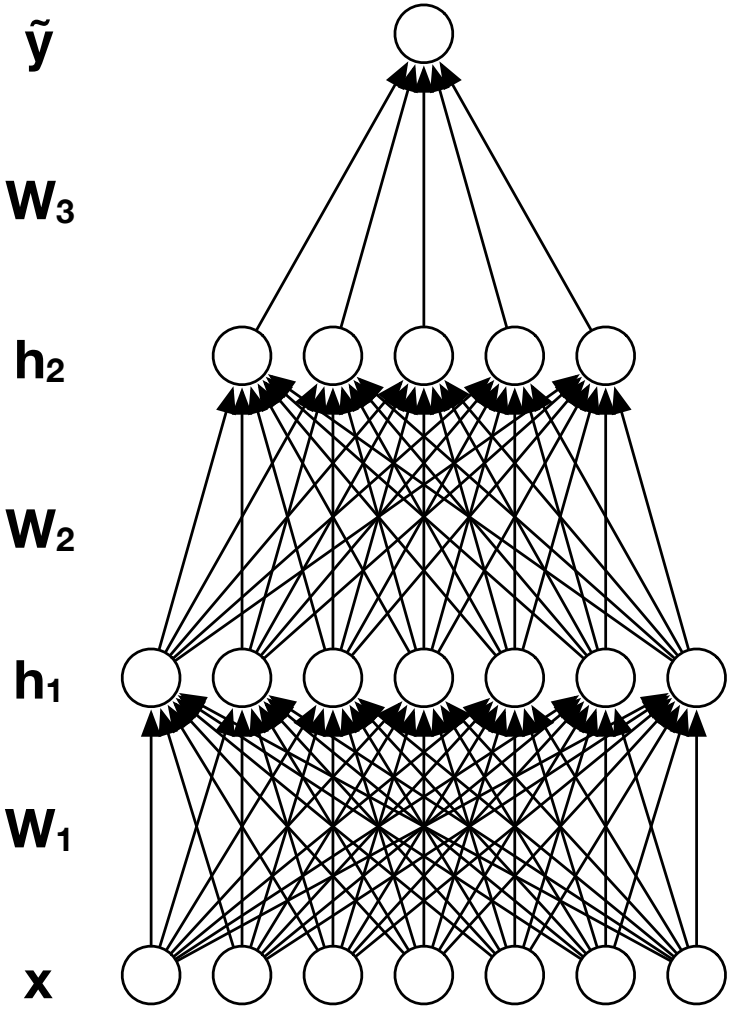


Feedforward network

- Simple feedforward networks have the following general structure:

- At every layer, there is some non-linear *activation function*

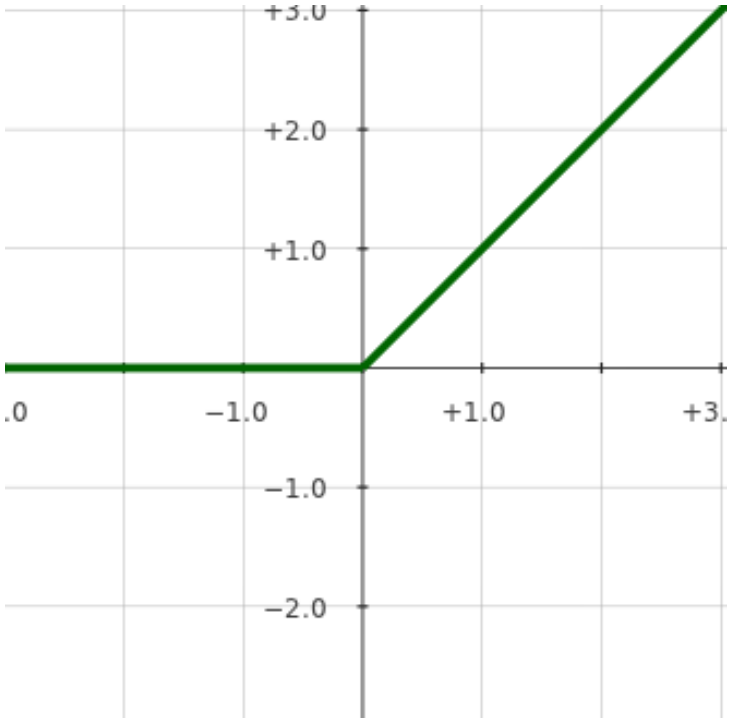


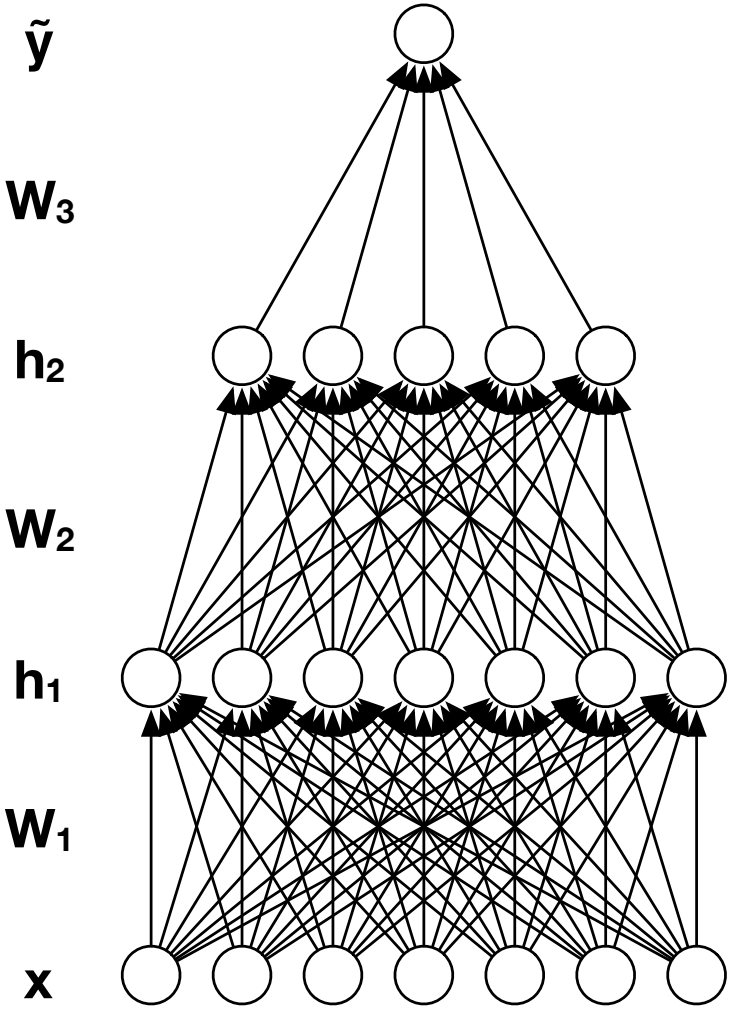


Feedforward network

- Simple feedforward networks have the following general structure:

- At every layer, there is some non-linear *activation function*





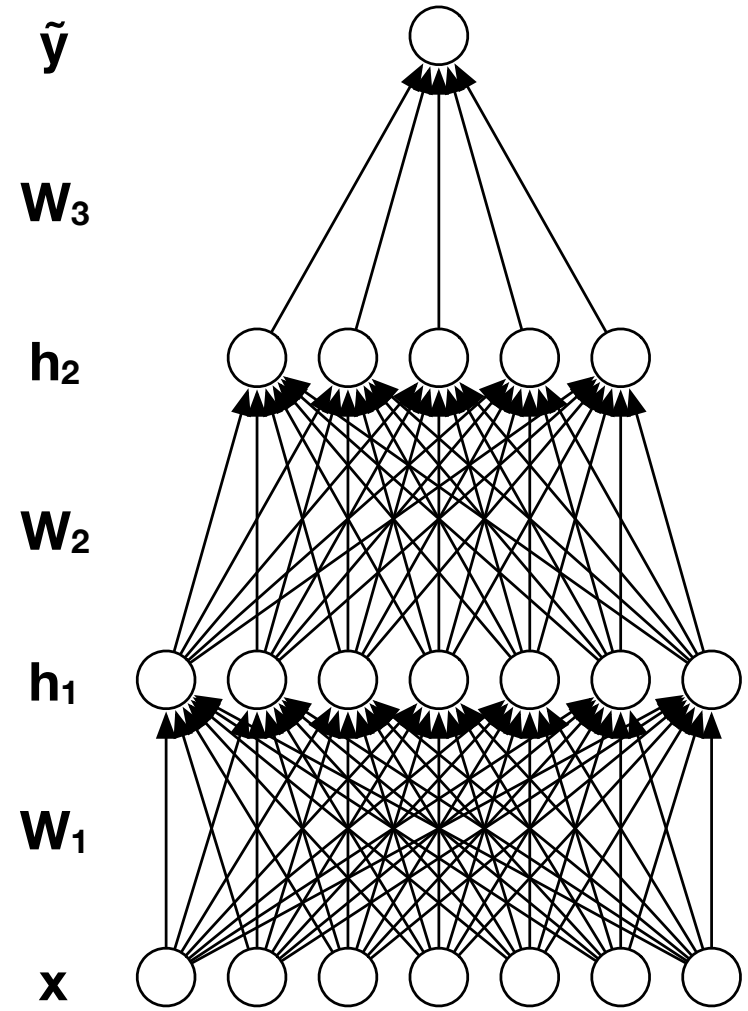
* Nair & Hinton, 2010

Feedforward network

- Simple feedforward networks have the following general structure:

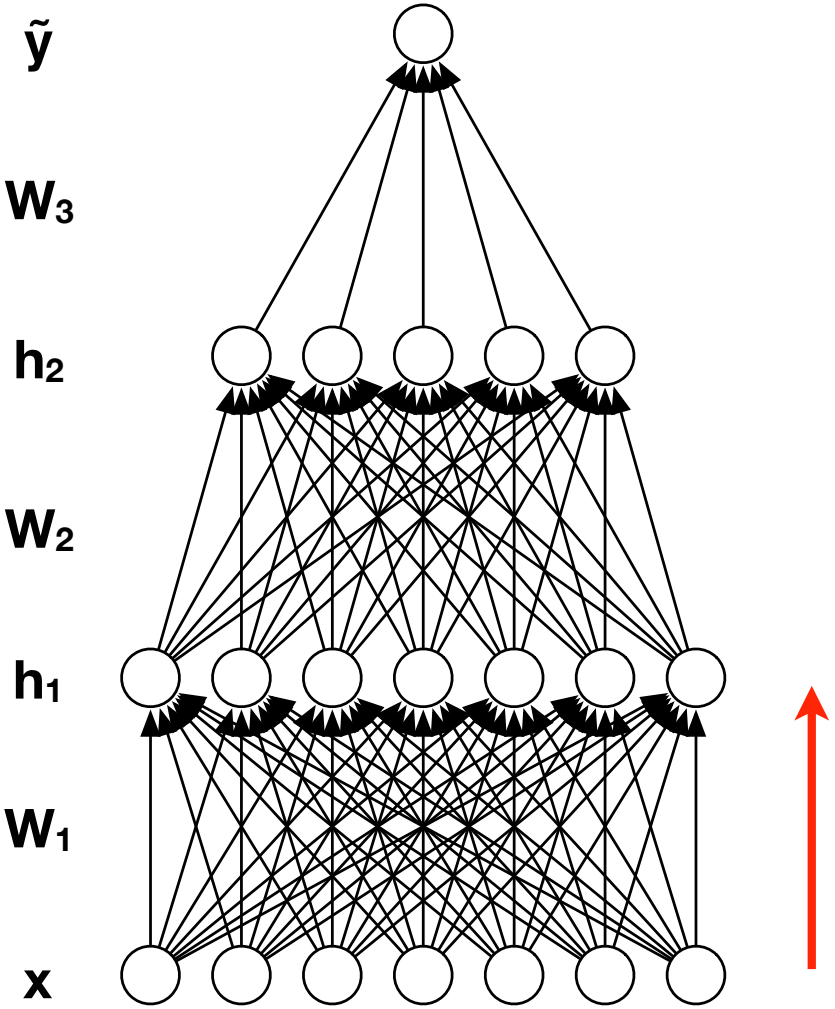
- Network is trained to minimize some *loss* between the output and the target:

$$C = \frac{1}{2}(y - \tilde{y})^2$$



Backpropagation

- Backpropagation performs *gradient descent* w.r.t. the network weights:

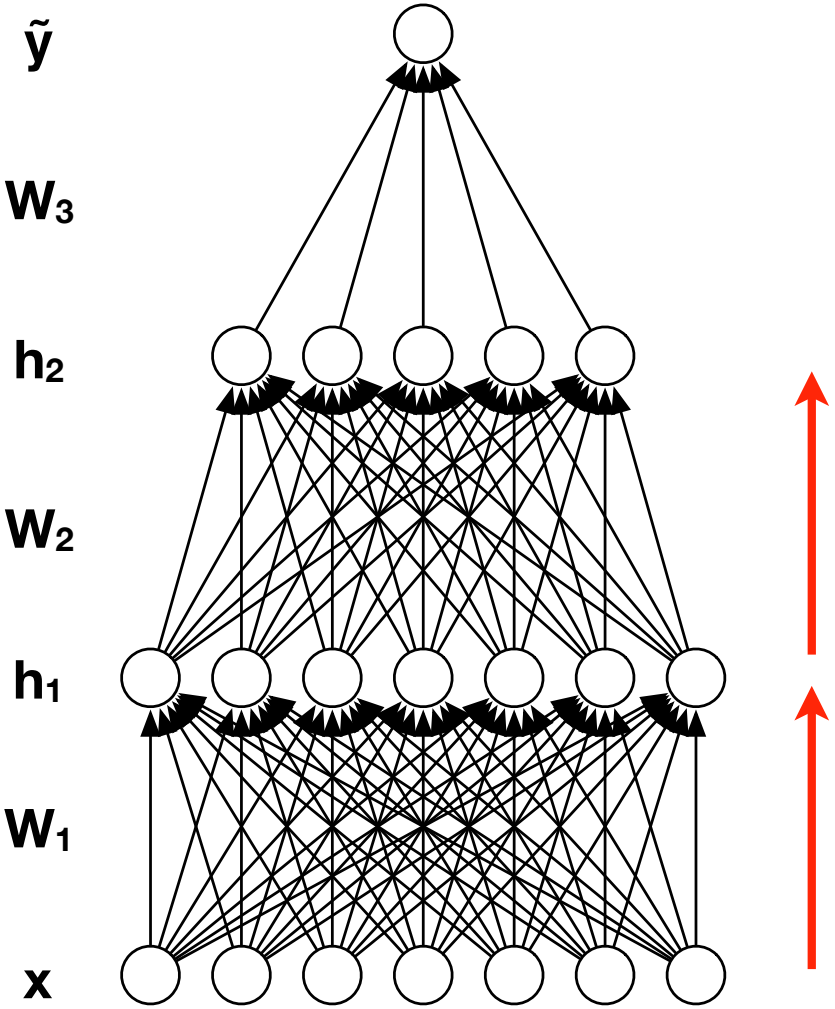


$$\mathbf{h}_1 = f_1(\mathbf{W}_1^{\mathrm{T}}\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{W}_1^{\mathrm{T}}\mathbf{x})}$$

* Rumelhart *et al.*, 1986

Backpropagation

- Backpropagation performs *gradient descent* w.r.t. the network weights:



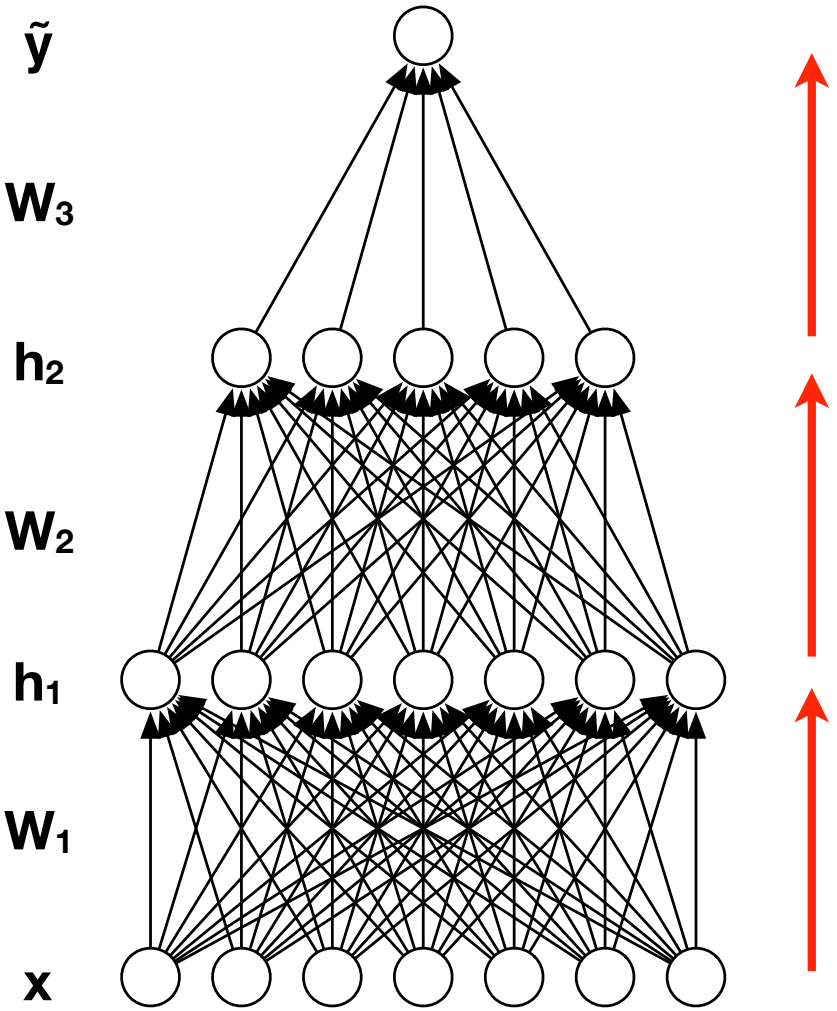
$$\mathbf{h}_2 = f_2(\mathbf{W}_2^{\mathrm{T}}\mathbf{h}_1) = \frac{1}{1+\exp(-\mathbf{W}_2^{\mathrm{T}}\mathbf{h}_1)}$$

$$\mathbf{h}_1 = f_1(\mathbf{W}_1^{\mathrm{T}}\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{W}_1^{\mathrm{T}}\mathbf{x})}$$

* Rumelhart *et al.*, 1986

Backpropagation

- Backpropagation performs *gradient descent* w.r.t. the network weights:



$$\tilde{y} = f_3(\mathbf{W}_3^{\mathrm{T}}\mathbf{h}_2) = \frac{1}{1+\exp(-\mathbf{W}_3^{\mathrm{T}}\mathbf{h}_2)}$$

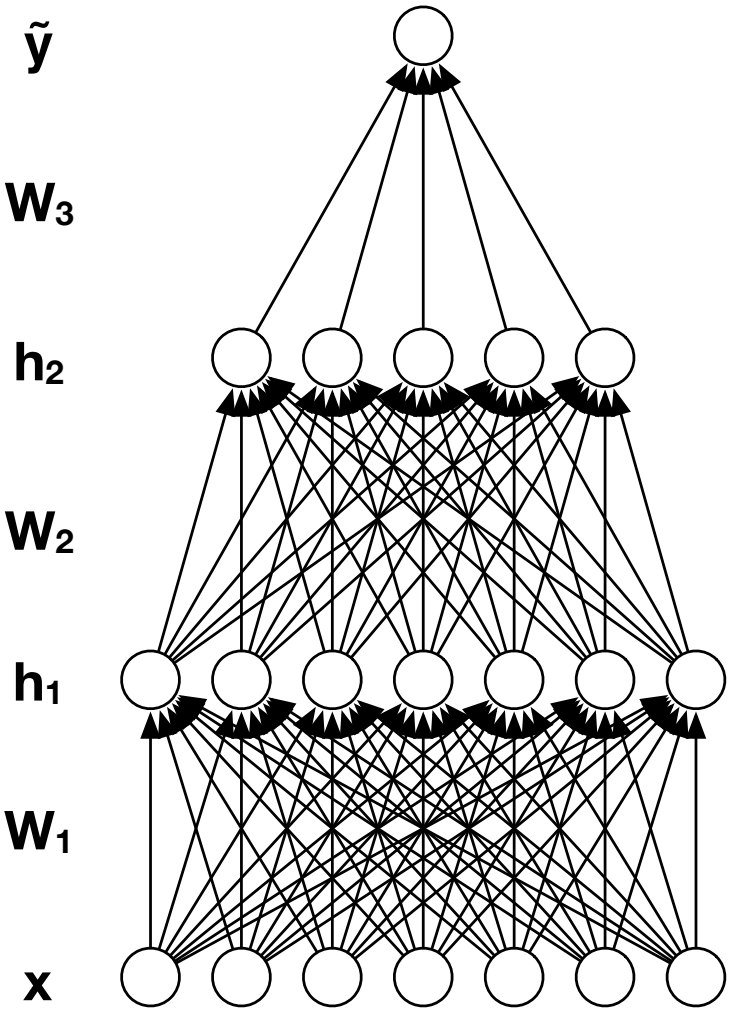
$$\mathbf{h}_2 = f_2(\mathbf{W}_2^{\mathrm{T}}\mathbf{h}_1) = \frac{1}{1+\exp(-\mathbf{W}_2^{\mathrm{T}}\mathbf{h}_1)}$$

$$\mathbf{h}_1 = f_1(\mathbf{W}_1^{\mathrm{T}}\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{W}_1^{\mathrm{T}}\mathbf{x})}$$

* Rumelhart *et al.*, 1986

Backpropagation

- Often, we want the network to minimize the *sum of squared errors*:



$$C = \frac{1}{2}(y - \tilde{y})^2$$

- *Backpropagation* proceeds in two steps:

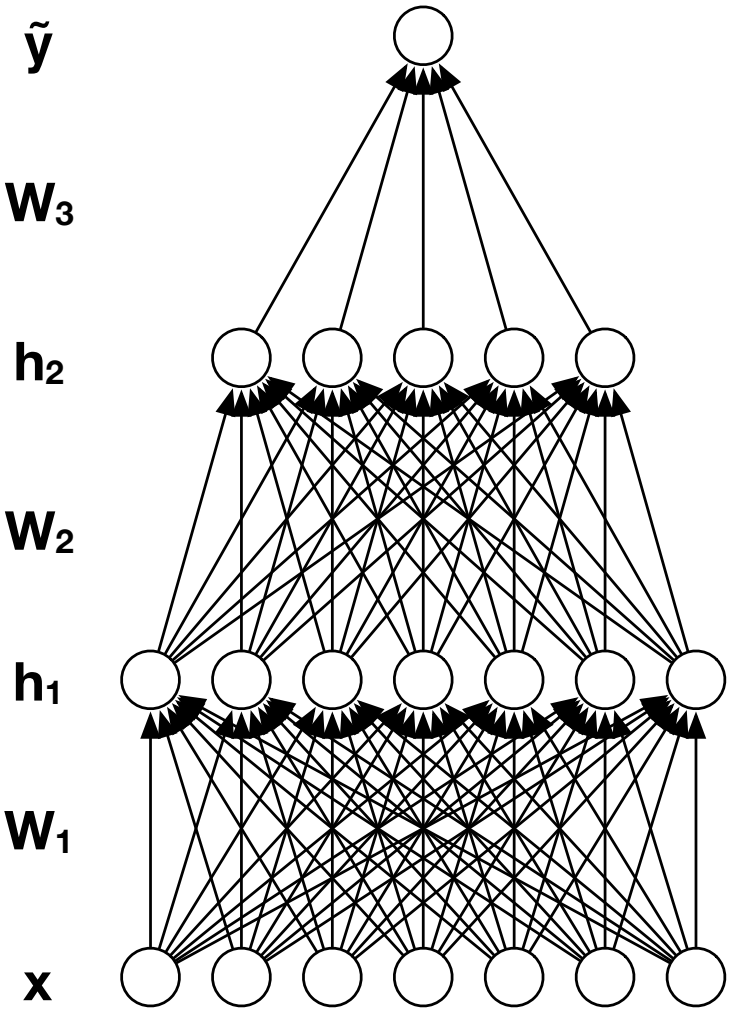
1. Propagate the error signal over weights

2. Compute the associated weight update

* Rumelhart *et al.*, 1986

Backpropagation

- Backpropagation performs *gradient descent* w.r.t. the network weights:



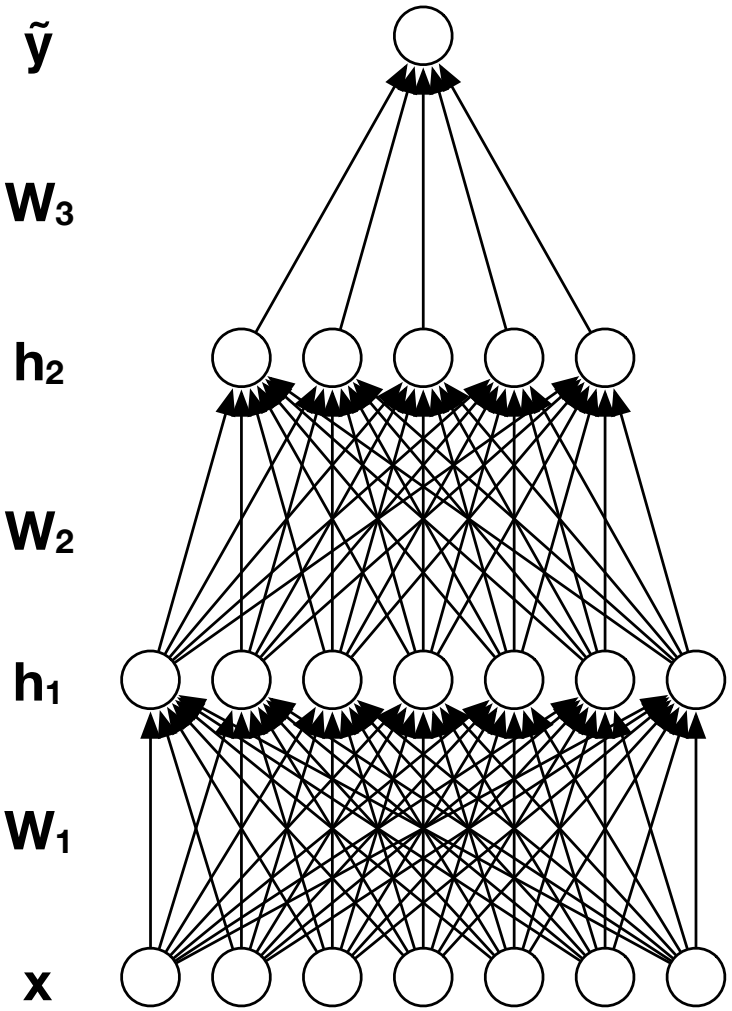
$$e_3 = (y - \tilde{y}) \frac{\partial f_3}{\partial \mathbf{W}_3} = (y - \tilde{y})\tilde{y}(1 - \tilde{y})$$

$$\frac{\partial C}{\partial \mathbf{W}_3} = e_3 \mathbf{h}_2$$

* Rumelhart *et al.*, 1986

Backpropagation

- Backpropagation performs *gradient descent* w.r.t. the network weights:



$$e_3 = (y - \tilde{y}) \frac{\partial f_3}{\partial \mathbf{W}_3} = (y - \tilde{y})\tilde{y}(1 - \tilde{y})$$

$$\frac{\partial C}{\partial \mathbf{W}_3} = e_3 \mathbf{h}_2$$

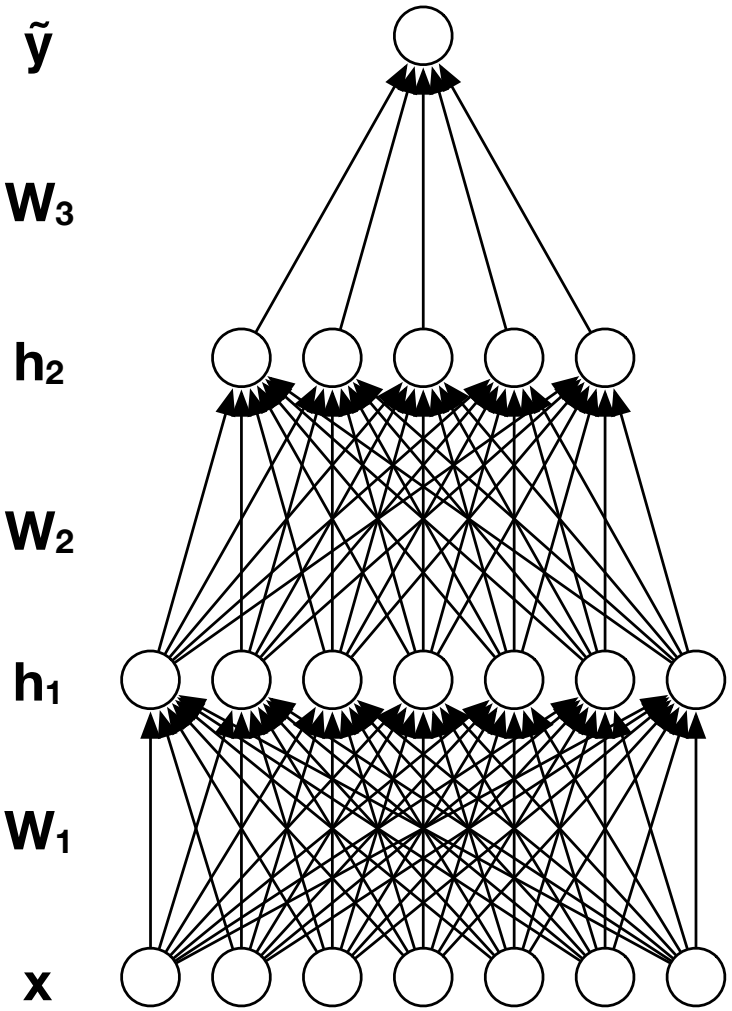
$$\mathbf{e}_2 = \mathbf{h}_2 \circ (1 - \mathbf{h}_2) \circ (\mathbf{W}_3 e_3)$$

$$\frac{\partial C}{\partial \mathbf{W}_2} = \mathbf{e}_2^{\mathrm{T}} \mathbf{h}_1$$

* Rumelhart *et al.*, 1986

Backpropagation

- Backpropagation performs *gradient descent* w.r.t. the network weights:



$$e_3 = (y - \tilde{y})\frac{\partial f_3}{\partial \mathbf{W}_3} = (y - \tilde{y})\tilde{y}(1 - \tilde{y})$$

$$\frac{\partial C}{\partial \mathbf{W}_3} = e_3 \mathbf{h}_2$$

$$\mathbf{e}_2 = \mathbf{h}_2 \circ (1 - \mathbf{h}_2) \circ (\mathbf{W}_3 e_3)$$

$$\frac{\partial C}{\partial \mathbf{W}_2} = \mathbf{e}_2^{\mathrm{T}} \mathbf{h}_1$$

$$\mathbf{e}_1 = \mathbf{h}_1 \circ (1 - \mathbf{h}_1) \circ (\mathbf{W}_2 \mathbf{e}_2)$$

$$\frac{\partial C}{\partial \mathbf{W}_1} = \mathbf{e}_1^{\mathrm{T}} \mathbf{x}$$

* Rumelhart *et al.*, 1986

Convolutional network

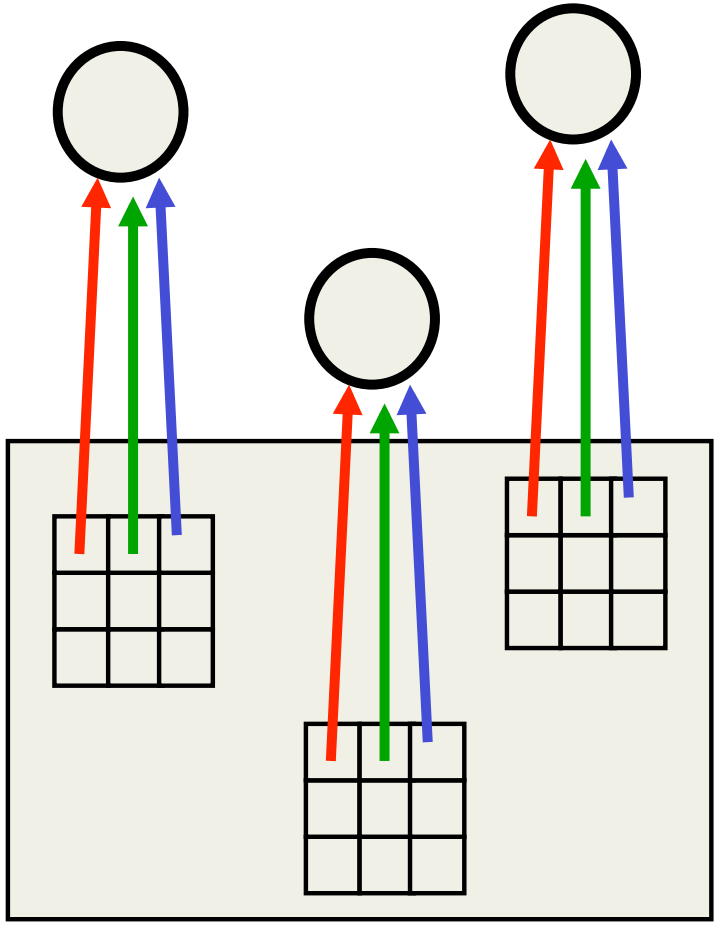
- Training a feedforward net on large images leads to huge numbers of weights
    - Do you expect to see some structure in those weights?

Convolutional network

- Training a feedforward net on large images leads to huge numbers of weights
    - Do you expect to see some structure in those weights? Yes!

Convolutional network

- Training a feedforward net on large images leads to huge numbers of weights
    - Do you expect to see some structure in those weights? Yes!
- *Convolutional networks* extract the same features from different image parts
- This restricts the number of weights we need to learn



The red connections all have the same weight.

* LeCun *et al.*, 1989

Convolutional network

- Local *pooling* of features increases invariance to small image variations

* LeCun *et al.*, 1989

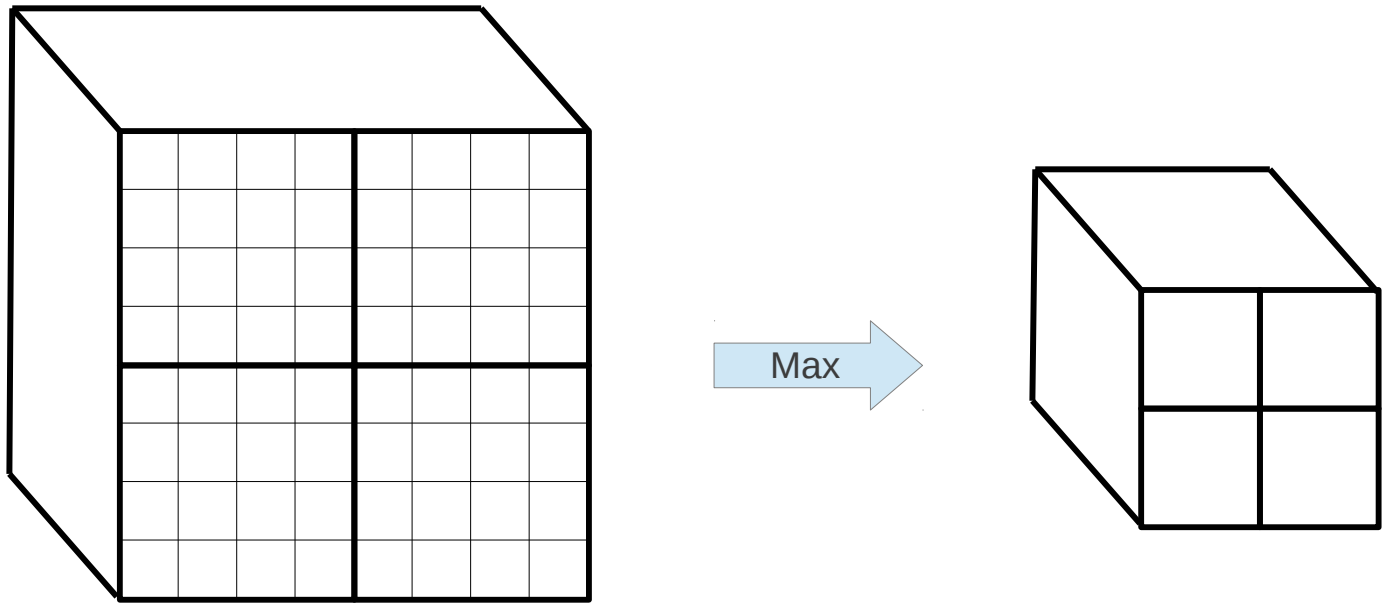
Convolutional network

- Local *pooling* of features increases invariance to small image variations
    - Such local pooling also happens in SIFT, HOG, BoW, *etc.*

* LeCun *et al.*, 1989

Convolutional network

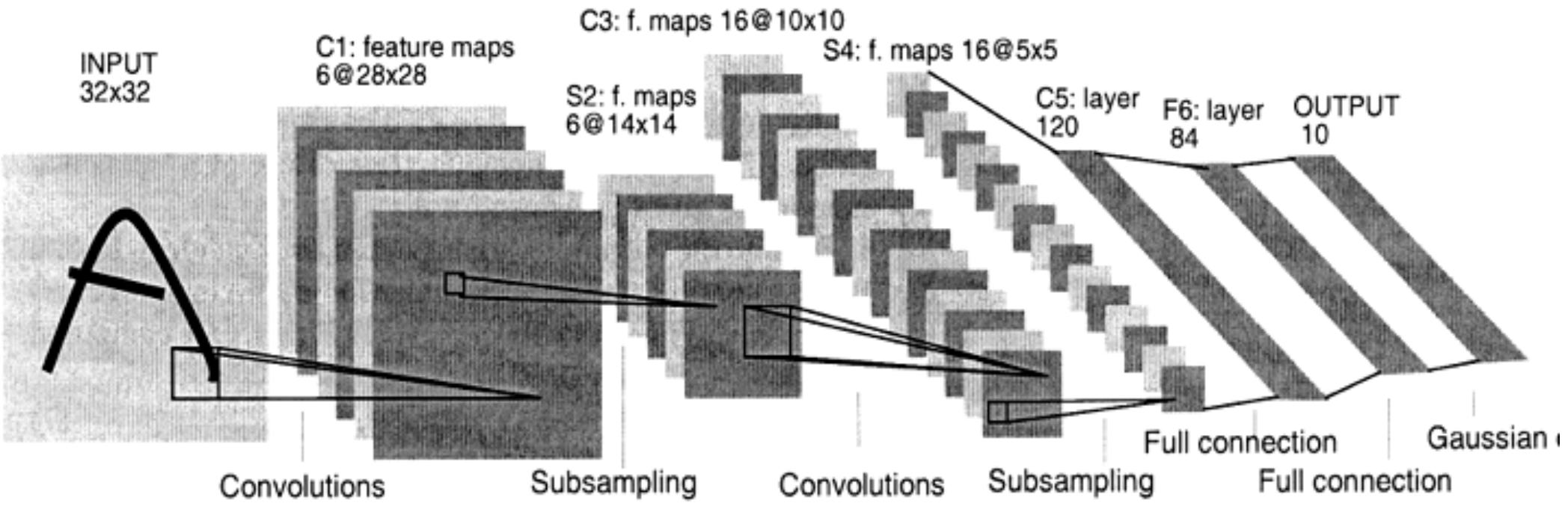
- Local *pooling* of features increases invariance to small image variations
  - Such local pooling also happens in SIFT, HOG, BoW, *etc.*
- Convolutional networks generally also include such *pooling layers*:



* LeCun *et al.*, 1989

Convolutional network

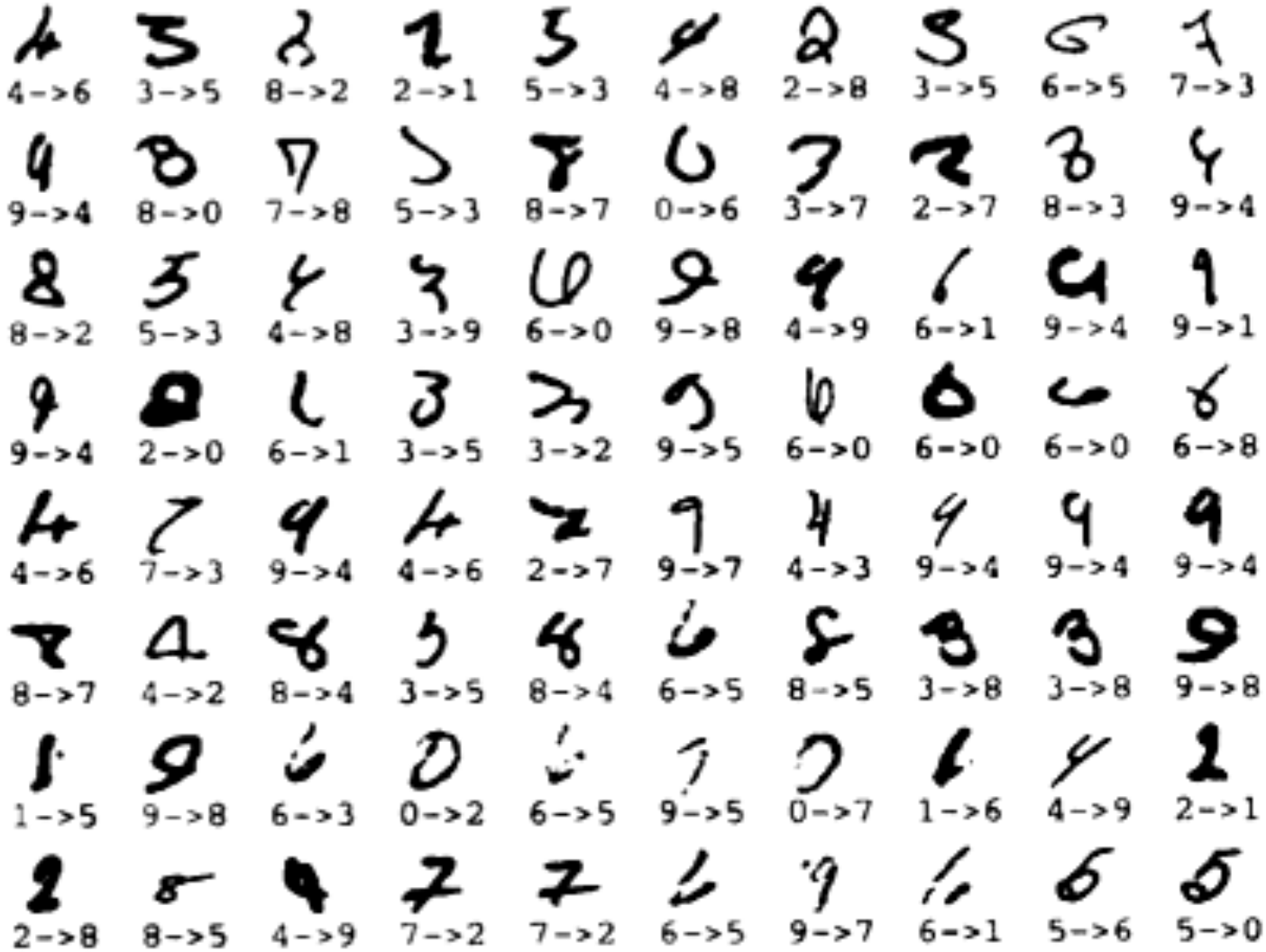
- LeNet5 was one of the first convolutional nets (for reading cheques):



* LeCun *et al.*, 1989

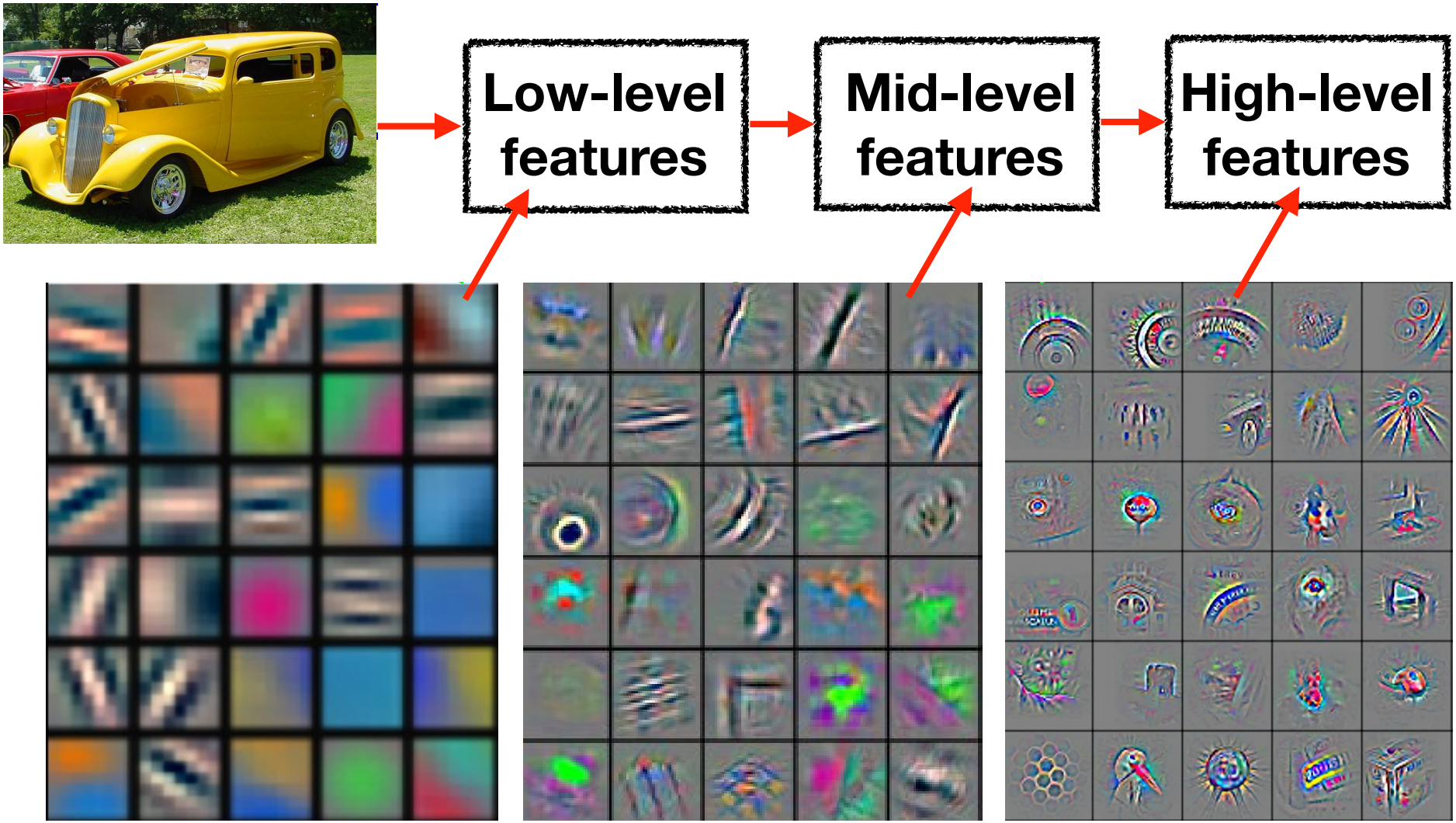
Convolutional network

- Erroneously recognized digits by LeNet5:



Convolutional network

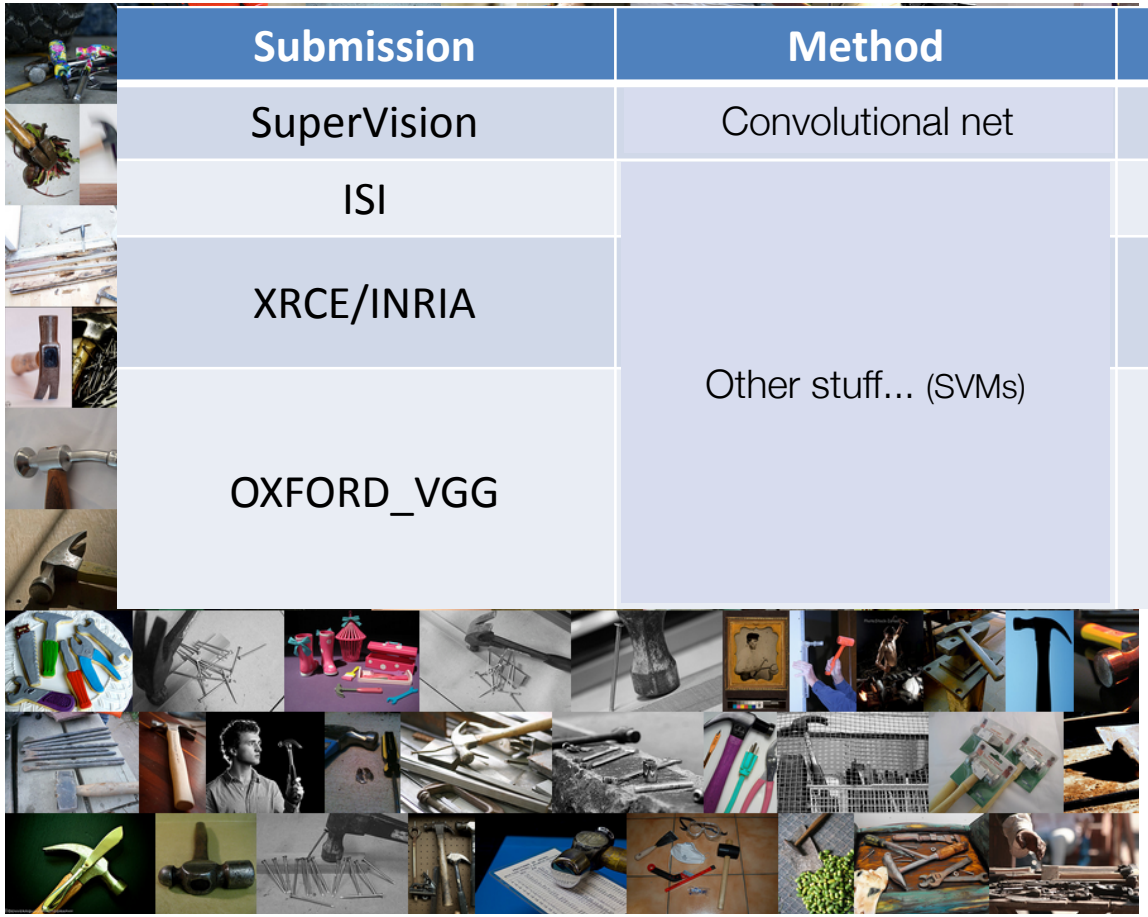
- Convolutional networks indeed appear to learn increasingly complex features:



The ILSVRC-2012 Competition

- Substantial improvement of the state-of-the-art in object recognition:

ImageNet Challenge 2012

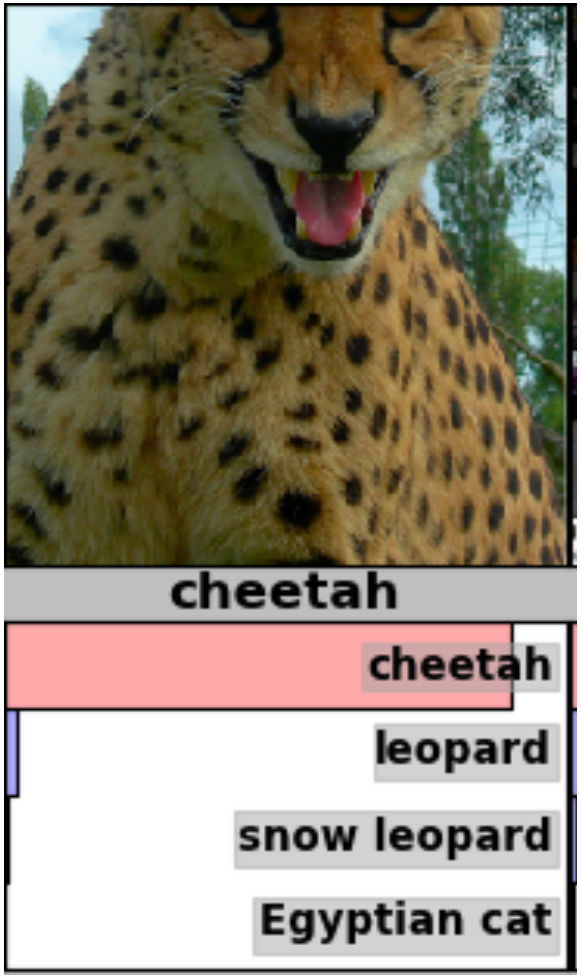


Submission	Method	Error rate
SuperVision	Convolutional net	0.16422
ISI	Other stuff... (SVMs)	0.26172
XRCE/INRIA		0.27058
OXFORD_VGG		0.27302

9.8%

1.1%

Recognition examples

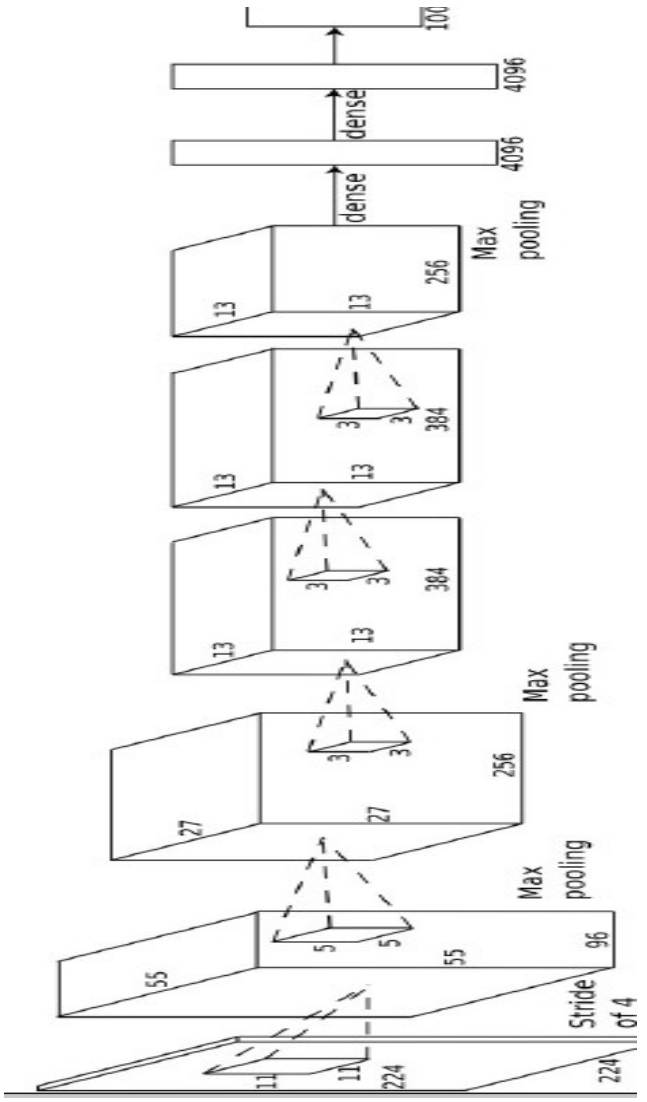


The ILSVRC-2012 Competition

- The winning model was a single, large convolutional network:
  - Seven hidden layers not counting the max-pooling layers
  - First five layers were convolutional; the last two fully connected
  - Rectified linear units were used as activation function everywhere
  - Model used a *competitive normalization* trick: suppress hidden activities when nearby units have stronger activities to deal with intensity variations
  - Total number of 80 million parameters

The ILSVRC-2012 Competition

4M	**FULL CONNECT**	4Mflop
16M	**FULL 4096/ReLU**	16M
37M	**FULL 4096/ReLU**	37M
	MAX POOLING	
442K	**CONV 3x3/ReLU 256fm**	74M
1.3M	**CONV 3x3ReLU 384fm**	224M
884K	**CONV 3x3/ReLU 384fm**	149M
	MAX POOLING 2x2sub	
	LOCAL CONTRAST NORM	
307K	**CONV 11x11/ReLU 256fm**	223M
	MAX POOL 2x2sub	
	LOCAL CONTRAST NORM	
35K	**CONV 11x11/ReLU 96fm**	105M



Regularization

- Convolutional networks have tons of parameters: prone to *overfitting*

Regularization

- Convolutional networks have tons of parameters: prone to *overfitting*
- Simple solution is to use *weight decay*: penalize L2-norm of network weights

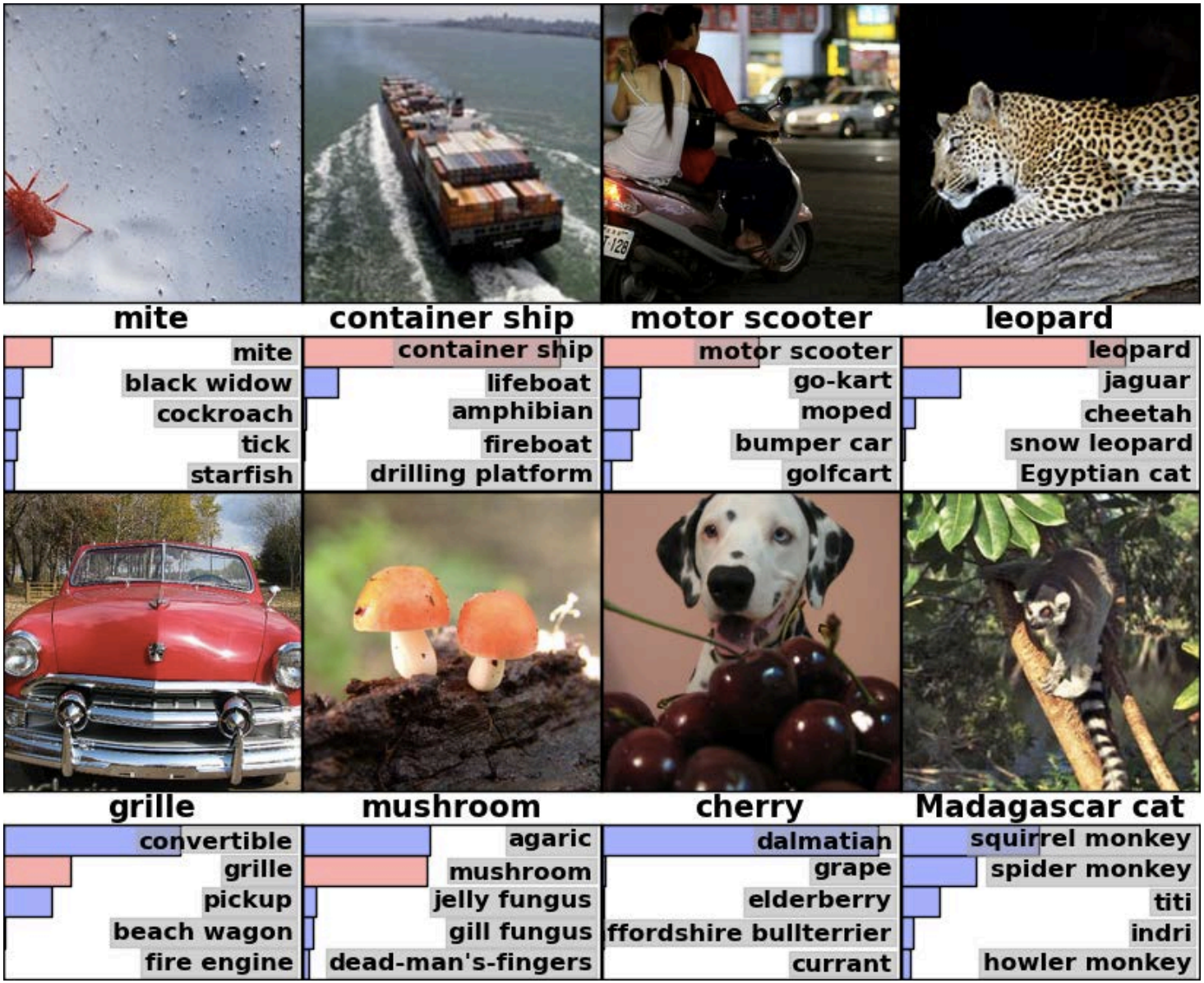
Regularization

- Convolutional networks have tons of parameters: prone to *overfitting*
- Simple solution is to use *weight decay*: penalize L2-norm of network weights
- Alternative, better way to regularize such networks is via *dropout*:
  - Randomly switch off some hidden units during forward pass

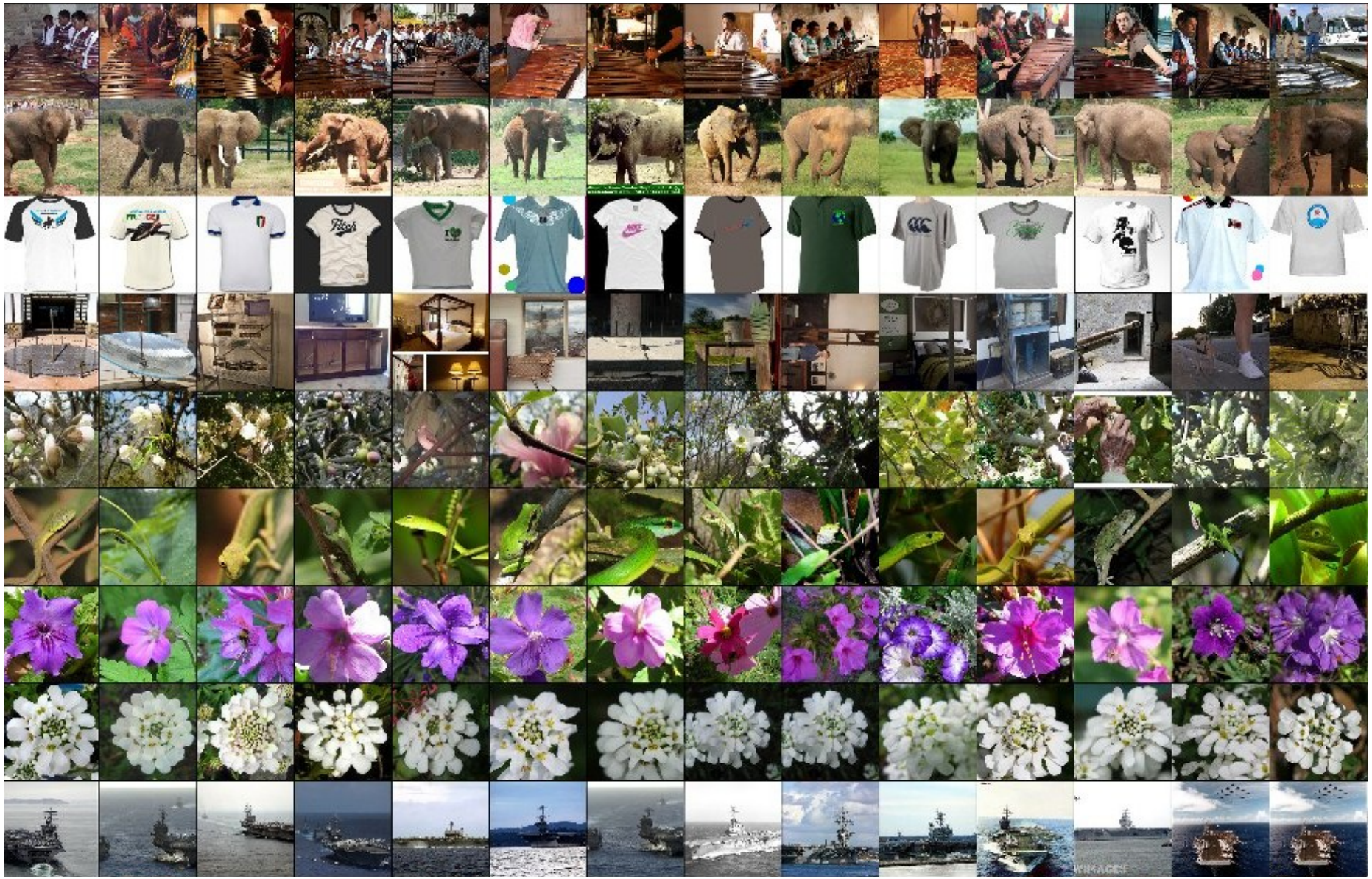
Regularization

- Convolutional networks have tons of parameters: prone to *overfitting*
- Simple solution is to use *weight decay*: penalize L2-norm of network weights
- Alternative, better way to regularize such networks is via *dropout*:
  - Randomly switch off some hidden units during forward pass
- Moreover, train model on slightly warped positive examples:

Some more examples



Retrieval using convolutional net



Implementing a convolutional net

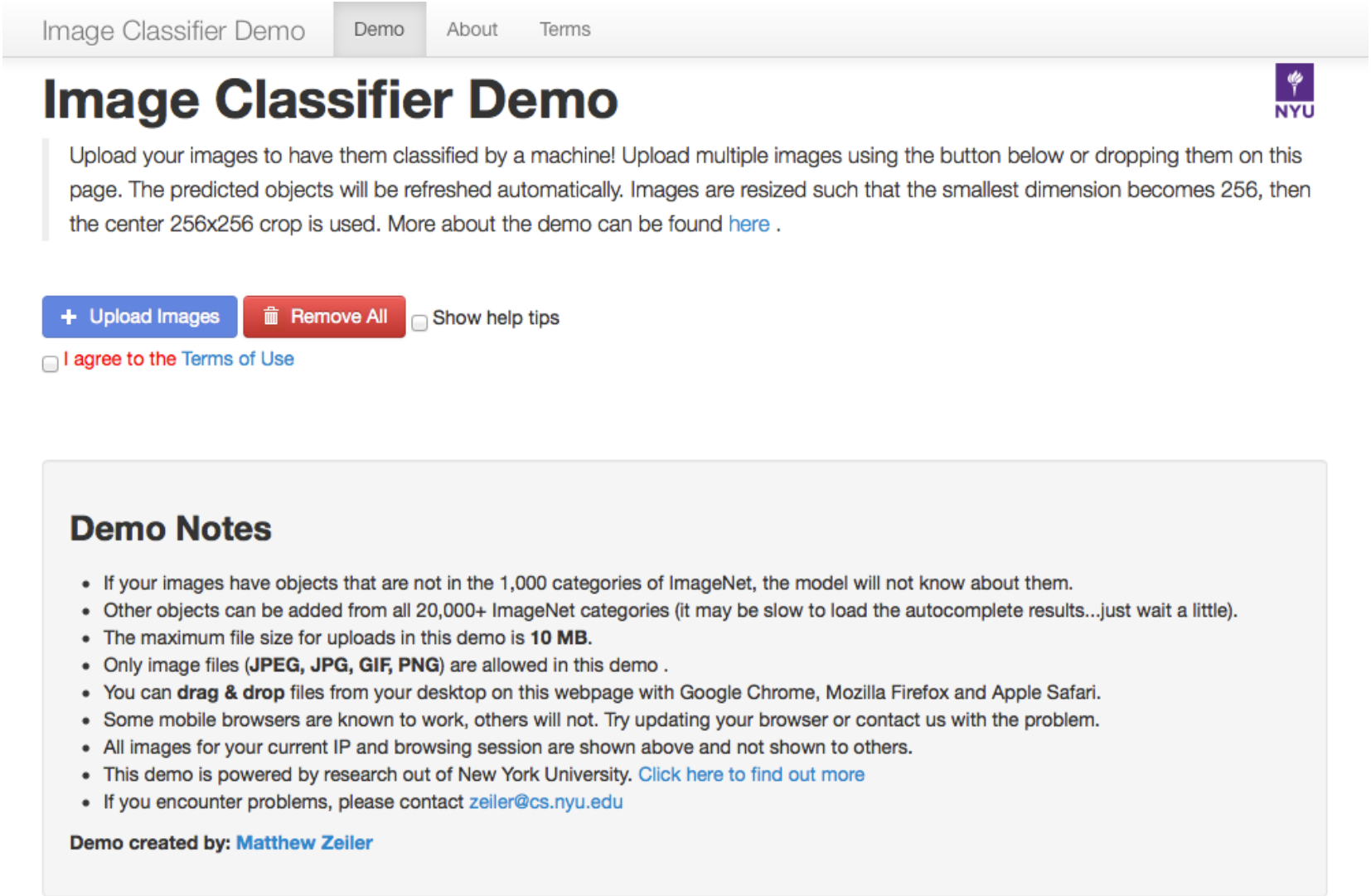
- Convolutional networks perform many convolutions and matrix multiplications
    - These are very fast on GPUs (up to 100 times faster)
- GPU implementations (and Big Data) are essential in the recent successes of deep learning and convolutional nets

Implementing a convolutional net

- Convolutional networks perform many convolutions and matrix multiplications
    - These are very fast on GPUs (up to 100 times faster)
- GPU implementations (and Big Data) are essential in the recent successes of deep learning and convolutional nets
- Libraries for building deep (convolutional) nets: `Torch7, eblearn, Theano, Caffe`

Demonstration

- http://horatio.cs.nyu.edu



Questions?