Image Classification with Deep Networks

Ronan Collobert

Facebook AI Research

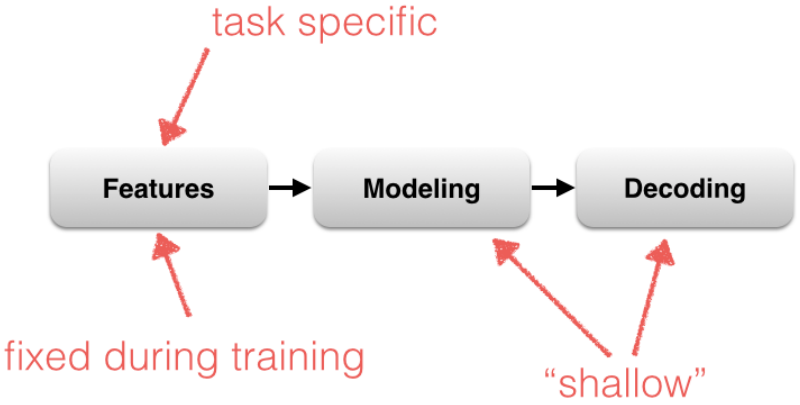
Feb 11, 2015

- **Origins of Deep Learning**
  - Shallow vs Deep
  - Perceptron
  - Multi Layer Perceptrons
- Going **Deeper**
  - Why?
  - Issues (and fix)?
- **Convolutional Neural Networks**
- **Fancier** Architectures
- **Applications**

Acknowledgement

Part of these slides have been cut-and-pasted from
Marc'Aurelio Ranzato's original presentation

Shallow vs Deep



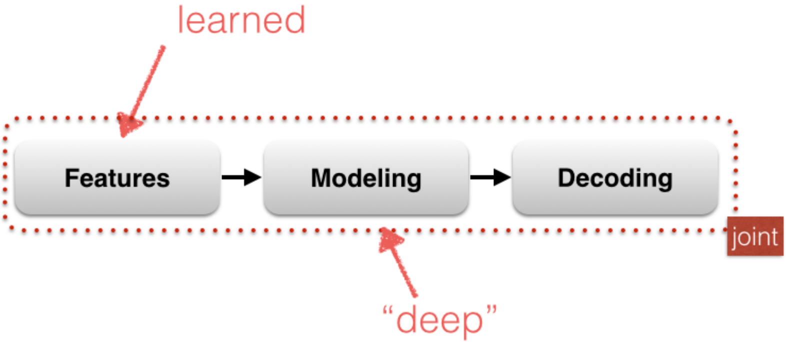
Task-specific features:
Simple machine learning model

Typical example

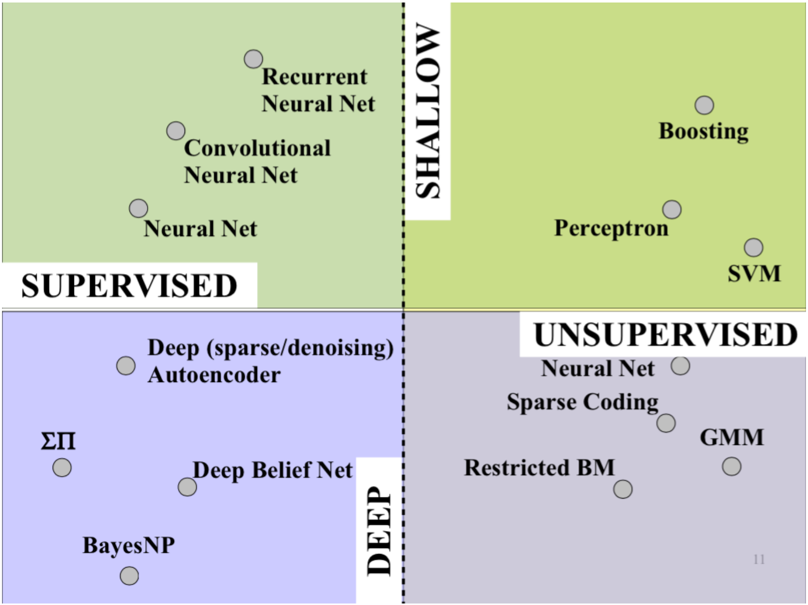
Features color and texture, region-based segmentation, shape, pyramid histogram of oriented gradients, percentage pixels above horizontal, SIFT

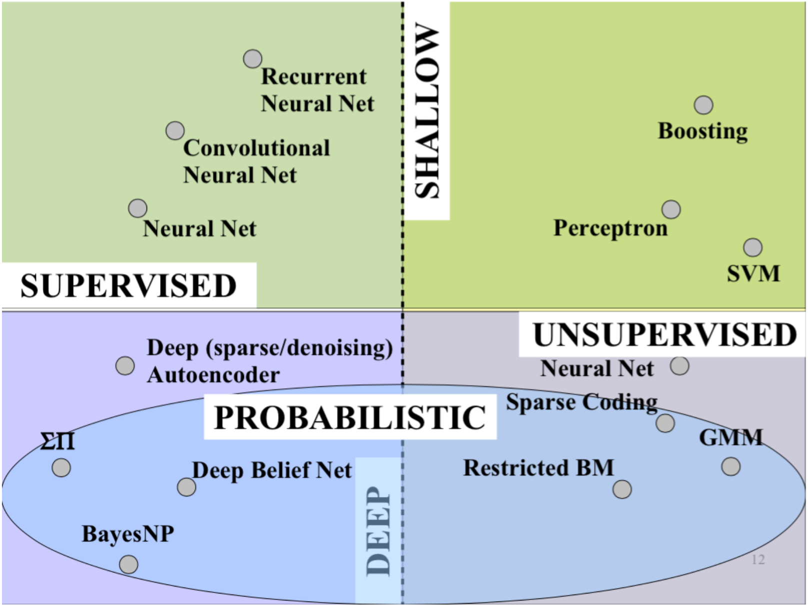
Modeling neural network, simple probabilistic model, linear programming, trees…

Decoding Markov Random Field



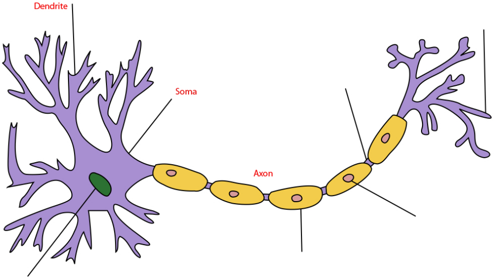
Flexible:
Complex machine learning algorithm

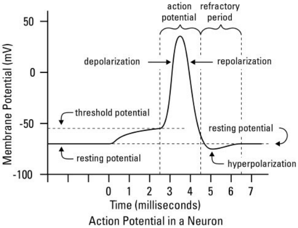

SHALLOW
SUPERVISED
Recurrent Neural Net
Convolutional Neural Net
Neural Net
Boosting
Perceptron
SVM
UNSUPERVISED
Deep (sparse/denoising) Autoencoder
ΣΠ
Deep Belief Net
BayesNP
DEEP
Neural Net
Sparse Coding
GMM
Restricted BM
11


SHALLOW
Recurrent Neural Net
Convolutional Neural Net
Neural Net
SUPERVISED
Boosting
Perceptron
SVM
UNSUPERVISED
Deep (sparse/denoising) Autoencoder
Neural Net
Sparse Coding
PROBABILISTIC
ΣΠ
GMM
Deep Belief Net
Restricted BM
BayesNP
DEEP
12

Perceptrons
(shallow)

Biological Neuron

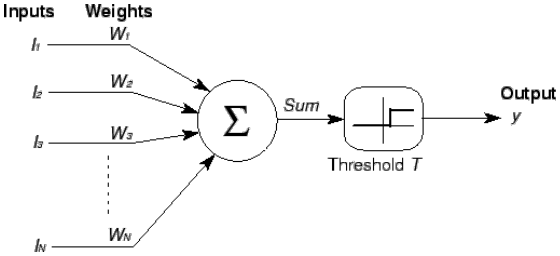




- **Dendrites** connected to other neurons through **synapses**
- **Excitatory** and **inhibitory** signals are integrated
- If stimulus reaches a threshold, neuron **fires** along the **axon**

McCulloch and Pitts (1943)

- Neuron as **linear threshold units**



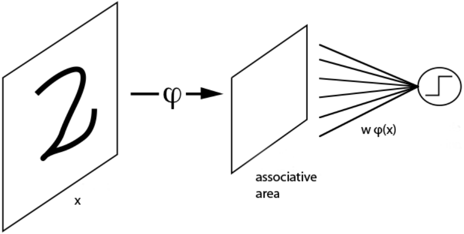
- **Binary** inputs $x \in \{0, 1\}^d$, **binary** output, vector of **weights** $w \in \mathrm{R}^d$

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x > T \\ 0 & \text{otherwise} \end{cases}$$

- A unit can perform OR and AND operations
- Combine these units to **represent any boolean function**

- **How to train them**?

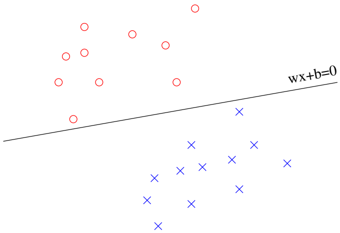
Perceptron: Rosenblatt (1957)



- Input: **retina** $x \in \mathrm{R}^n$
- **Associative area**: any kind of (fixed) function $\varphi(x) \in \mathrm{R}^d$
- **Decision** function:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot \varphi(x) > 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron: Rosenblatt (1957)



- **Training update rule**: given $(x_t,\ y_t) \in \mathrm{R}^d \times \{-1, 1\}$

$$w_{t+1} = w_t + \begin{cases} y_t\, \varphi(x_t) & \text{if } y_t\, w \cdot \varphi(x_t) \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

  - Note that

$$w_{t+1} \cdot \varphi(x_t) = w_t \cdot \varphi(x_t) + y_t \underbrace{||\varphi(x_t)||^2}_{>0}$$

- Corresponds to **minimizing**

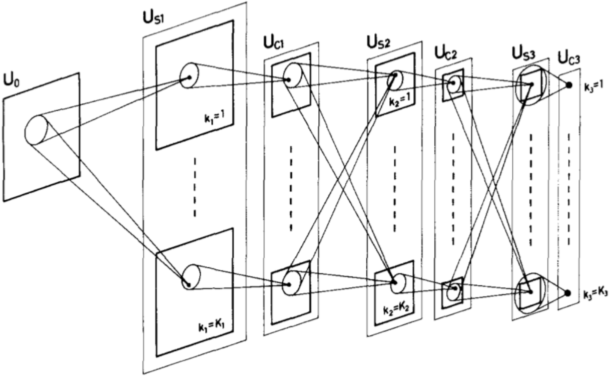
$$w \mapsto \sum_t \max(0, -y_t\, w \cdot \varphi(x_t))$$

Multi Layer Perceptrons
(deeper)

- How to **train** a "good" $\varphi(\cdot)$ in

$$w \cdot \varphi(x) \ ?$$

- Many attempts have been tried!
- Neocognitron (Fukushima, 1980)



- Madaline: Winter & Widrow, 1988

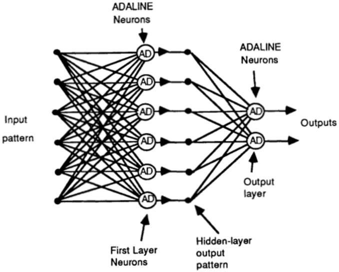
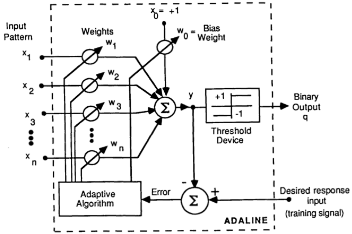


Figure 1: Layered feed-forward ADALINE network.



- **Multi Layer Perceptron**

$$x \longrightarrow \boxed{W^1 \times \bullet} \longrightarrow \boxed{\tanh(\bullet)} \longrightarrow \boxed{W^2 \times \bullet} \longrightarrow \text{score}$$

  - **Matrix-vector multiplications** interleaved with **non-linearities**
  - Each row of W^1 corresponds to a **hidden unit**
  - The number of hidden units must be chosen carefully

Universal Approximator (Cybenko, 1989)

- Any function

$$g : \mathrm{R}^d \longrightarrow \mathrm{R}$$

can be approximated (on a compact) by a **two-layer neural network**

$$x \longrightarrow \boxed{W^1 \times \bullet} \longrightarrow \boxed{\tanh(\bullet)} \longrightarrow \boxed{W^2 \times \bullet} \longrightarrow \text{score}$$

- Note:
  - It does not say **how to train it**
  - It does not say anything on the generalization capabilities

Training a Neural Network

- Given a network $f_w(\cdot)$ with parameters W, "input" examples x_t and "targets" y_t, we want to minimize a loss

$$W \mapsto \sum_{(x_t, y_t)} C(f_W(x_t), y_t)$$

- View the network+loss as a **"stack"** of layers

$$x \longrightarrow \boxed{f_1(\bullet)} \longrightarrow \boxed{f_2(\bullet)} \longrightarrow \boxed{f_3(\bullet)} \longrightarrow \boxed{f_4(\bullet)}$$

$$f(x) = f_L(f_{L-1}(\ldots f_1(x)))$$

- Optimization problem: use some sort of **gradient descent**

$$W_l \longleftarrow W_l - \lambda \frac{\partial f}{\partial w_l} \qquad \forall l$$

$\longrightarrow$ How to compute $\frac{\partial f}{\partial w_l}$ $\forall l$?

- In the neural network field: (Rumelhart et al, 1986)
- However, previous possible references exist, including (Leibniz, 1675) and (Newton, 1687)

- E.g., in the Adaline $L = 2$

$$x \longrightarrow \boxed{w^1 \times \bullet} \longrightarrow \boxed{\tfrac{1}{2}(y - \bullet)}$$

  - $f_1(x) = w_1 \cdot x$
  - $f_2(f_1) = \frac{1}{2}(y - f_1)^2$

$$\frac{\partial f}{\partial w_1} = \underbrace{\frac{\partial f_2}{\partial f_1}}_{=y-f_1} \underbrace{\frac{\partial f_1}{\partial w_1}}_{=x}$$

Gradient Backpropagation (2/2)

$$x \rightarrow \boxed{f_1(\bullet)} \rightarrow \boxed{f_2(\bullet)} \rightarrow \boxed{f_3(\bullet)} \rightarrow \boxed{f_4(\bullet)}$$

- Chain rule:

$$\frac{\partial f}{\partial w_l} = \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial f_{L-2}} \cdots \frac{\partial f_{l+1}}{\partial f_l} \frac{\partial f_l}{\partial w_l} = \frac{\partial f}{\partial f_l} \frac{\partial f_l}{\partial w_l}$$

- In the backprop way, each module $f_l()$
  - **Receive** the gradient w.r.t. its own outputs f_l
  - **Computes** the gradient w.r.t. its own input f_{l-1} (**backward**)
  - **Computes** the gradient w.r.t. its own parameters w_l (if any)

$$\frac{\partial f}{\partial f_{l-1}} = \frac{\partial f}{\partial f_l} \frac{\partial f_l}{\partial f_{l-1}}$$

$$\frac{\partial f}{\partial w_l} = \frac{\partial f}{\partial f_l} \frac{\partial f_l}{\partial w_l}$$

Examples Of Modules

- We denote
  - x the input of a module
  - z target of a loss module
  - y the output of a module $f_l(x)$
  - $\tilde{y}$ the gradient w.r.t. the output of each module

Module	**Forward**	**Backward**	**Gradient**
Linear	$y = W\,x$	$W^T\,\tilde{y}$	$\tilde{y}\,x^T$
Tanh	$y = \tanh(x)$	$\tilde{y}\,(1 - y^2)$	
Sigmoid	$y = 1/(1 + e^{-x})$	$\tilde{y}\,(1 - y)\,y$	
ReLU	$y = \max(0,\, x)$	$\tilde{y}\,1_{x \geq 0}$	
Perceptron Loss	$y = \max(0,\, -z\,x)$	$-1_{z \cdot x \leq 0}$	
MSE Loss	$y = \frac{1}{2}\,(x - z)^2$	$x - z$	

Typical Classification Loss (euh, Likelihood)

- Given a set of examples $(x_t, y_t) \in \mathrm{R}^d \times \mathrm{N}$, $t = 1 \dots T$ we want to maximize the (log-)likelihood

$$\log \prod_{t=1}^{T} p(y_t|x_t) = \sum_{t=1}^{T} \log p(y_t|x_t)$$

- The network outputs a **score** $f_y(x)$ per class y
- Interpret scores as **conditional probabilities** using a **softmax**:

$$p(y|x) = \frac{e^{f_y(x)}}{\sum_i e^{f_i(x)}}$$

- In practice we consider only log-probabilites:

$$\log p(y|x) = f_y(x) - \log \left[\sum_i e^{f_i(x)} \right]$$

Optimization Techniques

Minimize

$$W \mapsto \sum_{(x_t, y_t)} C(f_W(x), y)$$

- Gradient descent ("batch")

$$W \longleftarrow W - \lambda \sum_{(x_t, y_t)} \frac{\partial C(f_W(x_t), y_t)}{\partial W}$$

- **Stochastic** gradient descent

$$W \longleftarrow W - \lambda \frac{\partial C(f_W(x_t), y_t)}{\partial W}$$

- Many variants, including second order techniques (where the Hessian is approximated)

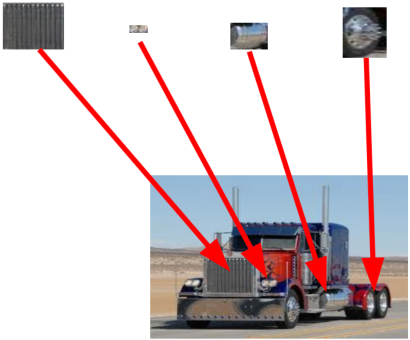
Going Deeper

Deeper: What is the Point? (1/3)

$$x \rightarrow \boxed{f_1(\bullet)} \rightarrow \boxed{f_2(\bullet)} \rightarrow \boxed{f_3(\bullet)} \rightarrow \boxed{f_4(\bullet)}$$

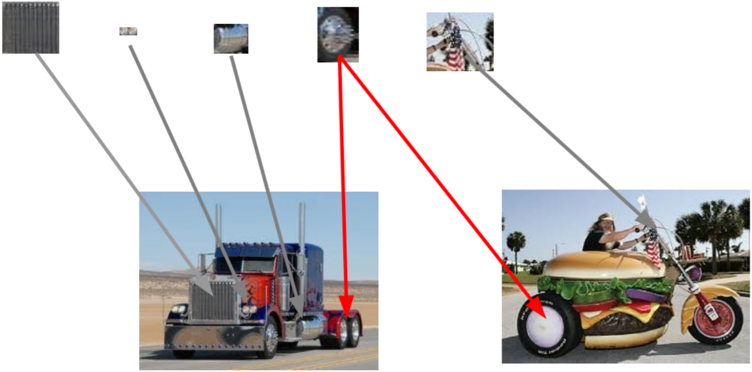
- **Share features** across the “deep” hierarchy
- **Compose** these features
- **Efficiency**: intermediate computations are **re-used**

[0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 . . .] truck feature

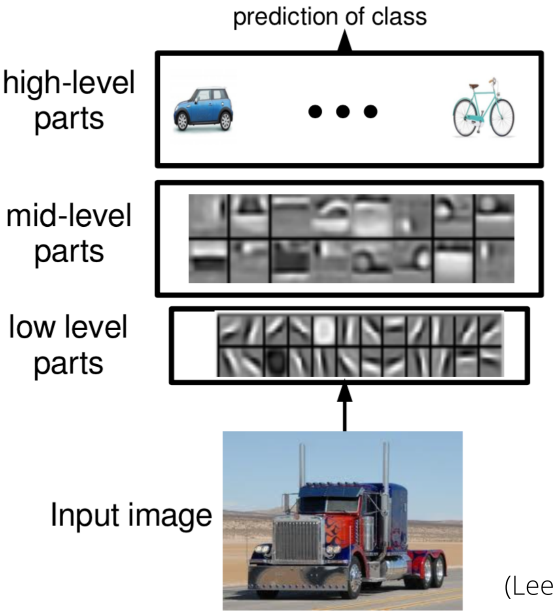


Sharing

[1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 . . .] motorbike
[0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 . . .] truck



Composing



(Lee et al., 2009)

Vanishing Gradients

- Chain rule:

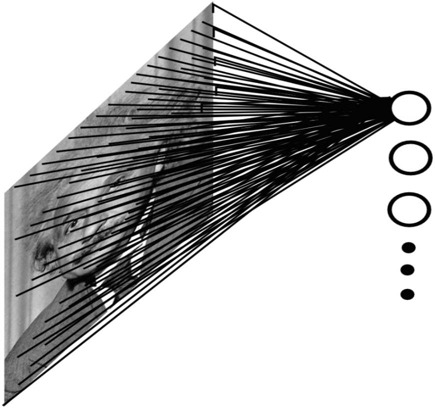
$$\frac{\partial f}{\partial w_l} = \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial f_{L-2}} \cdots \frac{\partial f_{l+1}}{\partial f_l} \frac{\partial f_l}{\partial w_l}$$

- Because transfer function non-linearities, some $\frac{\partial f_{l+1}}{\partial f_l}$ will be very small, or zero, when back-propagating
- E.g. with ReLU

$$y = \max(0, x) \qquad \frac{\partial y}{\partial x} = 1_{x \geq 0}$$

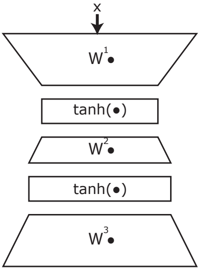
Number of Parameters

- A 200×200 image with 1000 hidden units leads to $40B$ parameters
- We would need a **lot** of training examples
- Spatial correlation is **local** anyways



Fix Vanishing Gradient Issue with Unsupervised Training (1/2)

- Leverage **unlabeled data** (when there is no y)?
  - Popular way to **pretrain** each layer
- "**Auto-encoder/bottleneck**" network

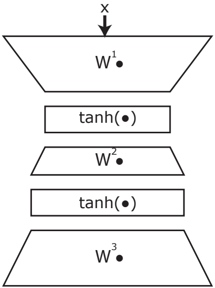


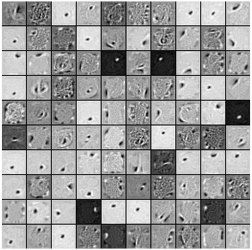
- Learn to **reconstruct** the input

$$||f(x) - x||^2$$

- Caveats:
  - **PCA** if no W^2 layer (Bourlard & Kamp, 1988)
  - Projected intermediate space must be of **lower dimension**

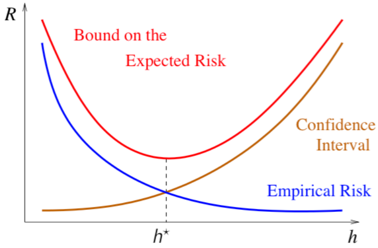
Fix Vanishing Gradient Issue with Unsupervised Training (2/2)

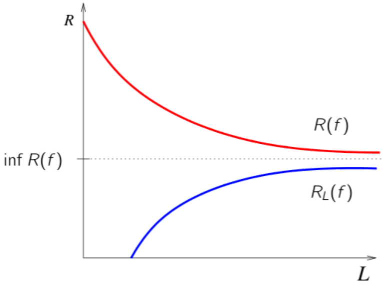




- Possible improvements:
  - No W^2 layer, $W^3 = \left[W^1\right]^T$ (Bengio et al., 2006)
  - **Noise injection** in x
    reconstruct the true x (Bengio et al., 2008)
  - Impose **sparsity constraints**
    on the projection (Kavukcuoglu et al., 2008)

- Capacity h is too large? Find more training examples L!

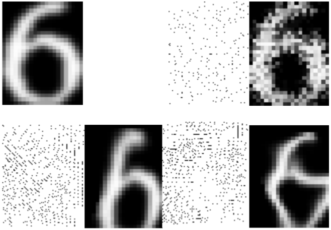




Fix Number of Parameters Issue by Generating Examples (2/2)

- Concrete example: digit recognition
- Add an (infinite) number of **random deformations**

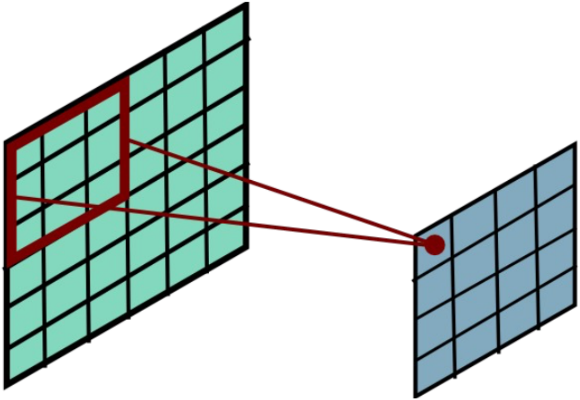
(Simard et al, 2003)



- State-of-the-art with 9 layers with 1000 hidden units and... a GPU (Ciresan et al, 2010)
- In general, **data augmentation** includes
  - random translation or rotation
  - random left/right flipping
  - random scaling

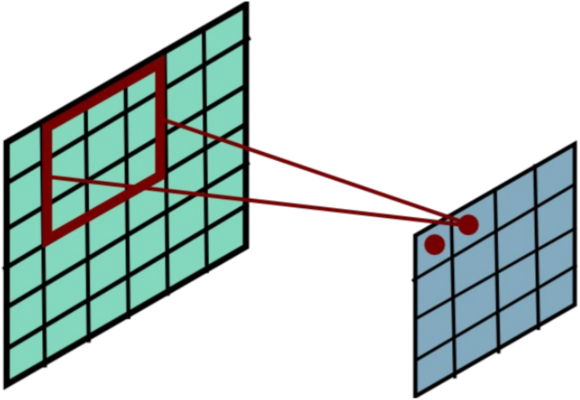
Convolutional Neural Networks

- **Share parameters across different locations**



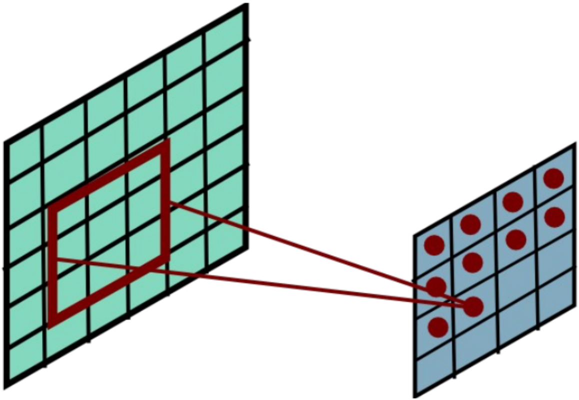
(Fukushima, 1980)
(LeCun, 1987)

- **Share parameters across different locations**



(Fukushima, 1980)
(LeCun, 1987)

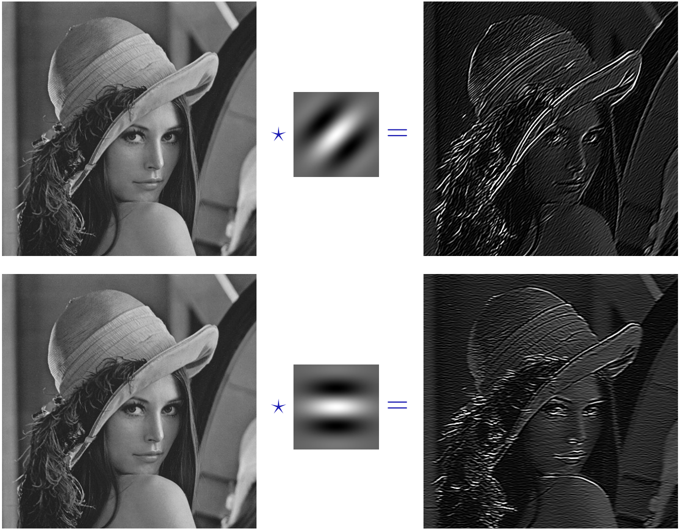
- **Share parameters across different locations**



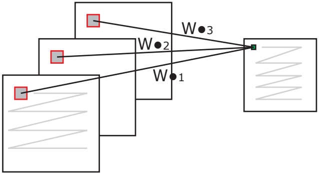
(Fukushima, 1980)
(LeCun, 1987)

2D Convolutions (2/4)

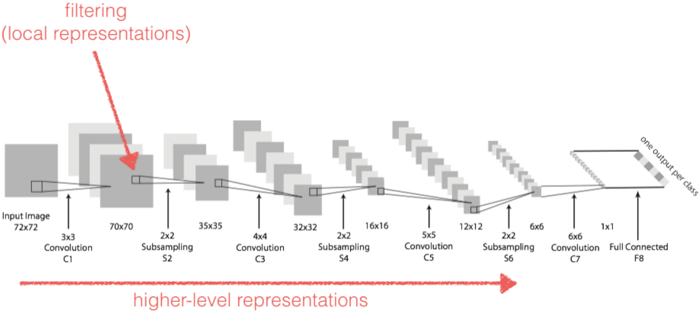
- It is like applying a filter to the image...
- ...but the filter is **trained**



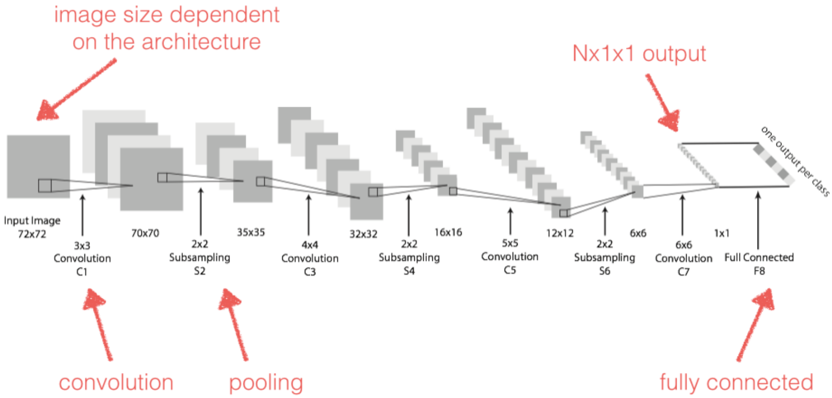
- It is again a **matrix-vector operation**, but where weights are **spatially "shared"**



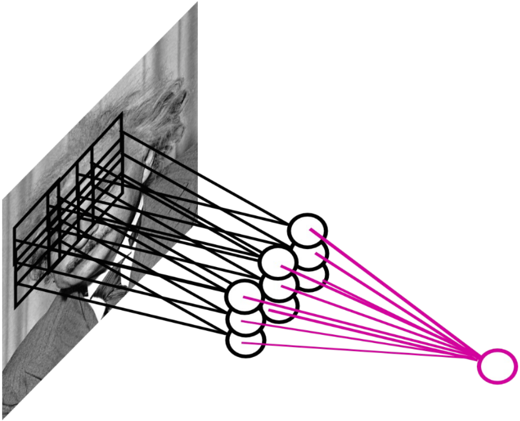
- As for normal linear layers, can be **stacked for higher-level representations**



2D Convolutions (4/4)

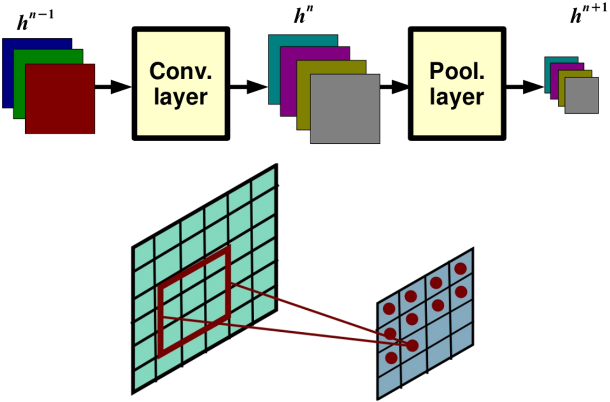


- "Pooling" (e.g. with a max() operation) **increases robustness** w.r.t. spatial location



Controls the capacity

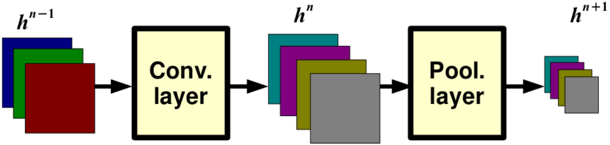
- A unit will see "more" of the image, for the same number of parameters

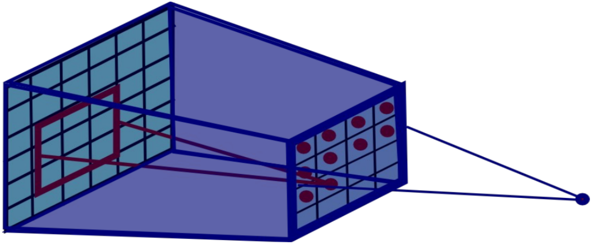


- adding pooling decreases the size of subsequent fully connected layers!

Controls the capacity

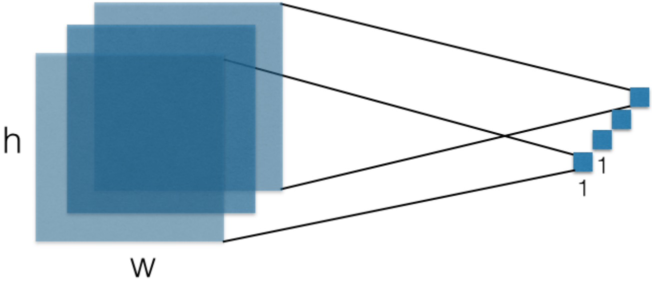
- A unit will see “more” of the image, for the same number of parameters





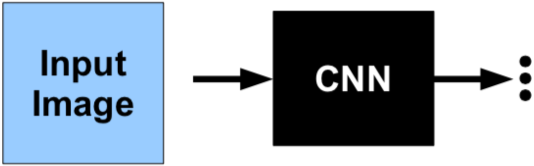
- adding pooling decreases the size of subsequent fully connected layers!

Fully connected layers are a particular type of **convolutions** (with a $w \times h$ kernel)

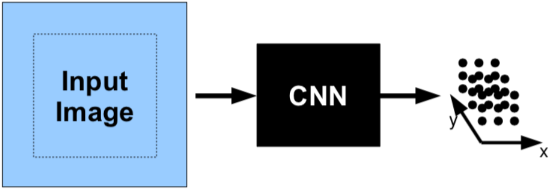


Training vs Testing

- Training Phase

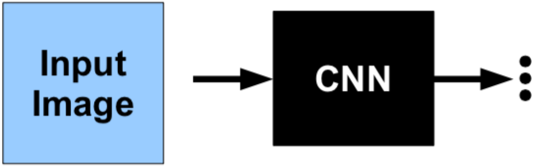


- Testing Phase

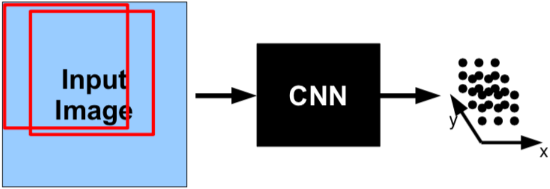


Training vs Testing

- Training Phase

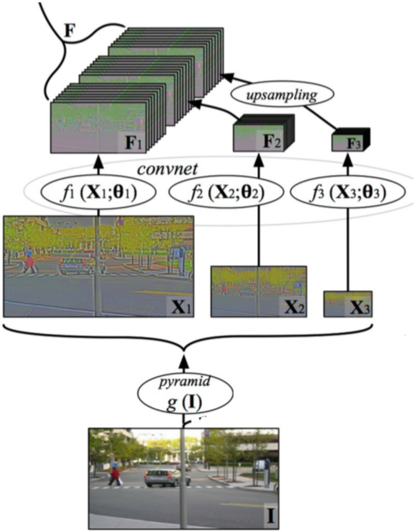


- Testing Phase
    - convolutions are naturally applied to larger input images
    - much **faster** than sliding windows



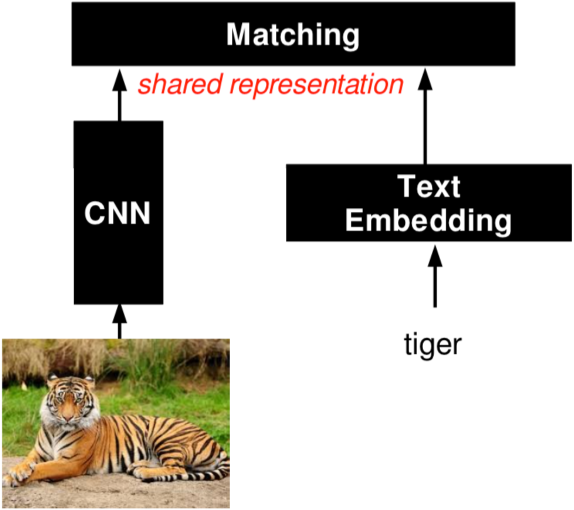
Fancier Architectures

Multi-Scale



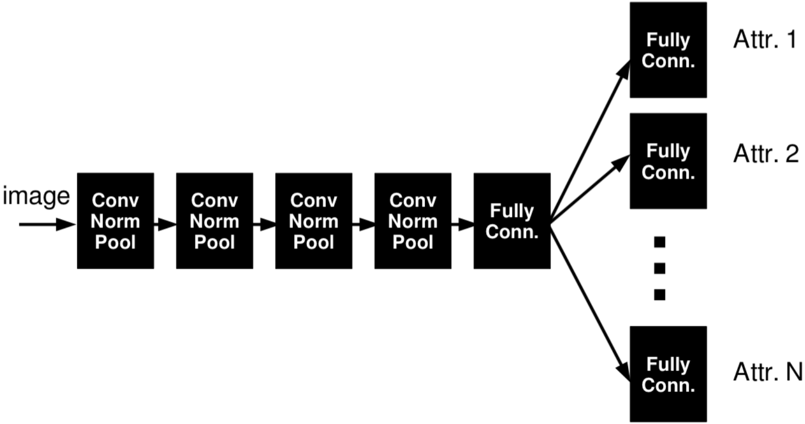
(Farabet et al., 2013) “Learning hierarchical features for scene labeling”

Multi-Modal



(Frome et al., 2013) “Devise: a deep visual semantic embedding model”

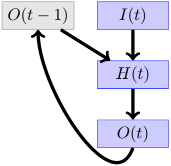
Multi-Task



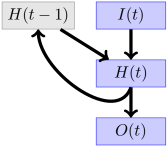
(Zhang et al., 2014) "PANDA"

Recurrent Neural Networks (1/3)

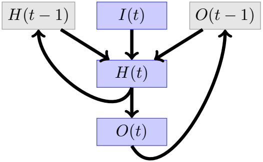
- Leverage previous output label scores



- Leverage previous hidden representations



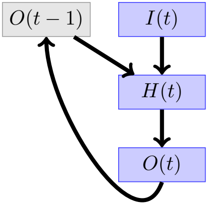
- Both

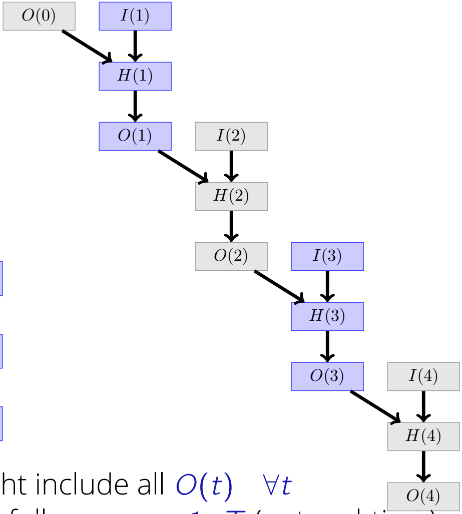


(Jordan, 1986) (Elman, 1990)

Recurrent Neural Networks (2/3)

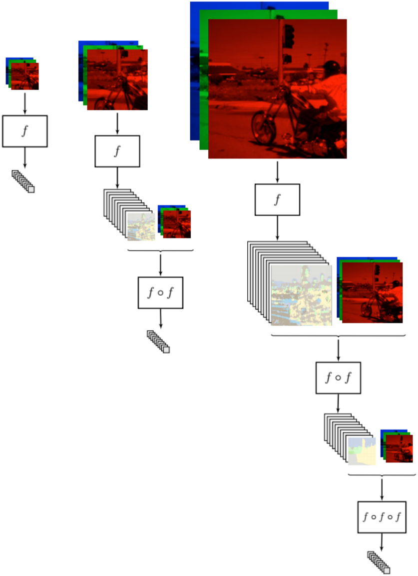
- Training: **unfold** network through time
  - Weights are **shared through time**
  - Standard backpropagation applies





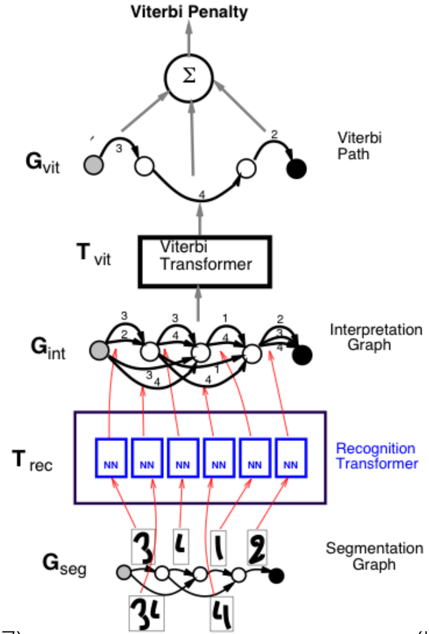
- Note: the loss might include all $O(t) \quad \forall t$
- Must consider the full sequence $1..T$ (not real-time)

Recurrent Neural Networks (3/3)



(Pinheiro et al., 2014) “Recurrent Convolutional Neural Networks for Scene Labeling”

Graph Transformer Networks



(Bottou et al., 1997) (Lecun et al., 1998)

Applications

Digit Recognition (1/2)

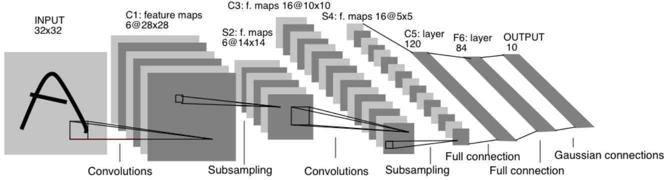
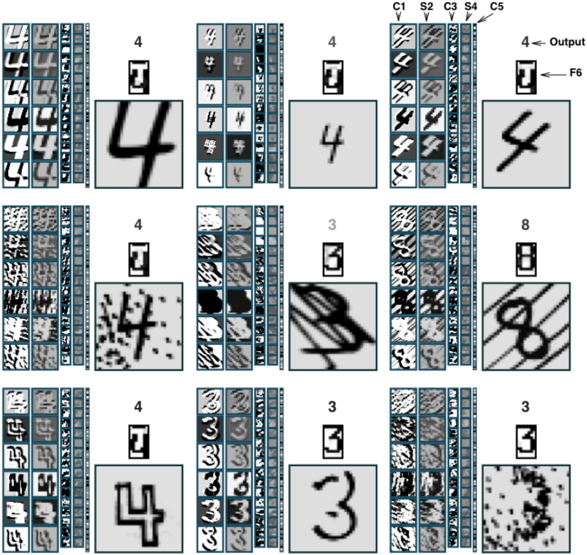


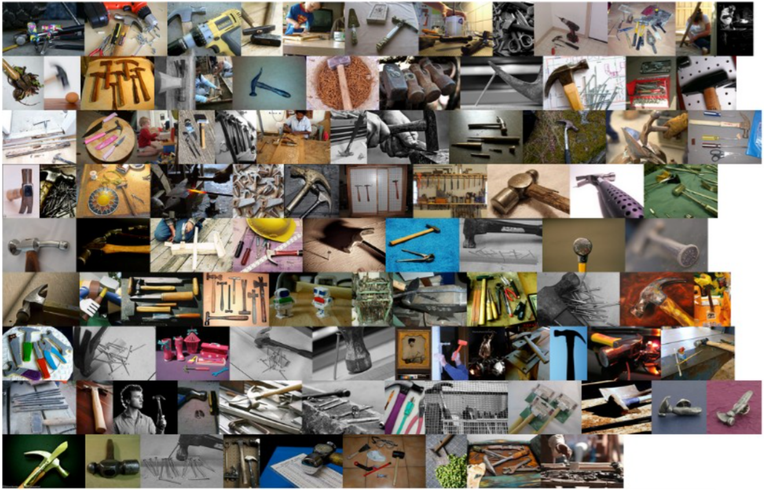
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Fig. 4. Size-normalized examples from the MNIST database.

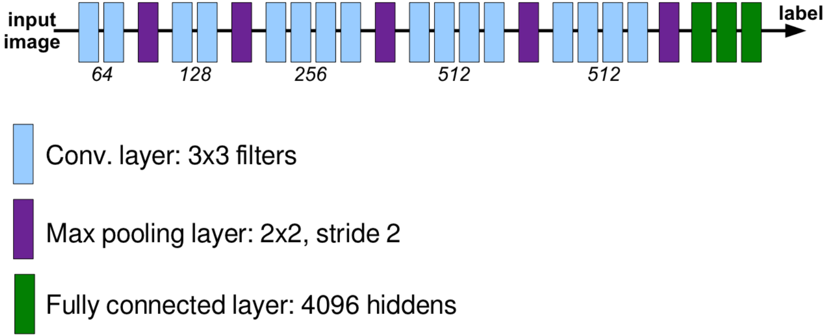
	Err. rate (%)
Gaussian SVM	1.4
1000 HU NN (MSE)	4.5
800 HU NN	1.6
CNN	0.8
CNN + distortions	0.4
9 layers NN + distortions	0.4



(Lecun et al., 1998)

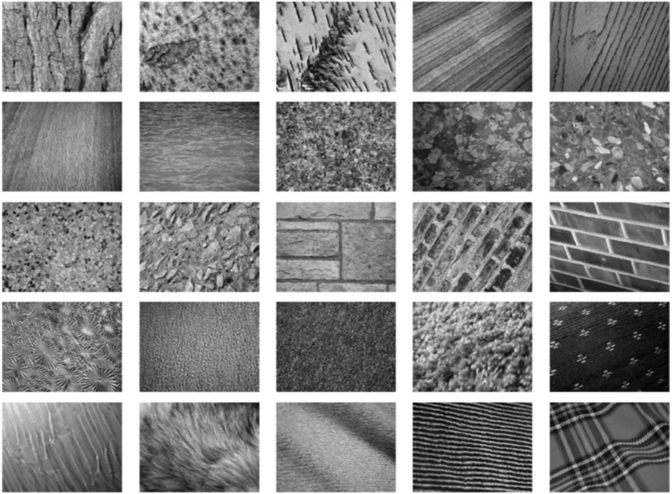


(Deng et al., 2009) “Imagenet: a large scale hierarchical image database”



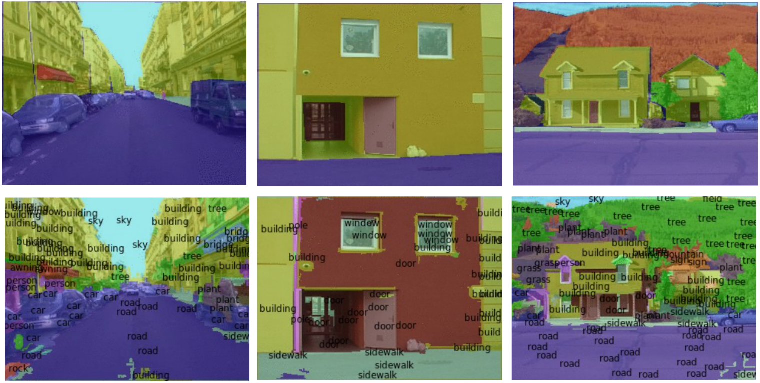
(Krizhevsky et al., 2012) "ImageNet Classification with deep CNNs"

Texture Classification



(Sifre et al., 2013) “Rotation, scaling and deformation invariant scattering for texture discrimination”

Object Segmentation

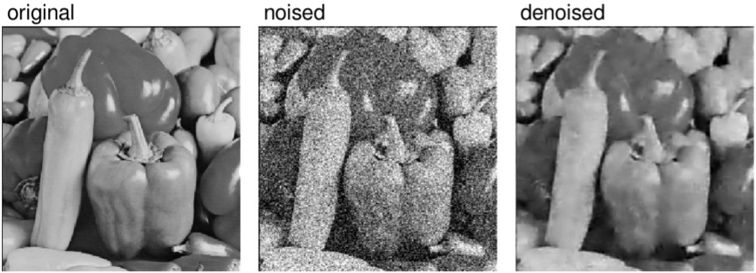


(Farabet et al., 2013) “Learning hierarchical features for scene labeling”
(Pinheiro et al., 2014) “Recurrent CNN for scene parsing”

Action Recognition in Videos

(Taylor et al., 2010) "Convolutional learning of spatio-temporal features"
(Karpathy et al, 2014) "Large-scale video classification with CNNs"

Denoising



(Burger et al., 2012) “Can plain NNs compete with BM3D?”

Toolboxes

- Torch7 `http://torch7.org`
- Theano `http://deeplearning.net/software/theano`
- Cuda Convnet `http://code.google.com/p/cuda-convnet`
- Caffe `http://caffe.berkeleyvision.org`
- NVIDIA Kernels `https://developer.nvidia.com/cuDNN`