Deep Learning & Convolutional Networks In Vision
VRML, Paris 2013-07-23

Yann LeCun
Center for Data Science & Courant Institute, NYU
yann@cs.nyu.edu
http://yann.lecun.com
Deep Learning = Learning Representations/Features

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

End-to-end learning / Feature learning / Deep learning
- Trainable features (or kernel) + trainable classifier
This Basic Model has not evolved much since the 50's

- The first learning machine: the Perceptron
  - Built at Cornell in 1960
- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor requires considerable efforts by experts.

$$y = \text{sign} \left( \sum_{i=1}^{N} W_i F_i(X) + b \right)$$
Modern architecture for pattern recognition

Speech recognition: early 90's – 2011

- MFCC (fixed)
- Mix of Gaussians (unsupervised)
- Classifier (supervised)

Object Recognition: 2006 - 2012

- SIFT (fixed)
- HoG (fixed)
- K-means
- Sparse Coding (unsupervised)
- Pooling
- Classifier (supervised)
Deep Learning = Learning Hierarchical Representations

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

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

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform

**Image recognition**
- Pixel → edge → texton → motif → part → object

**Text**
- Character → word → word group → clause → sentence → story

**Speech**
- Sample → spectral band → sound → ... → phone → phoneme → word
How do we learn representations of the perceptual world?

- How can a perceptual system build itself by looking at the world?
- How much prior structure is necessary

ML/AI: how do we learn features or feature hierarchies?

- What is the fundamental principle? What is the learning algorithm? What is the architecture?

Neuroscience: how does the cortex learn perception?

- Does the cortex “run” a single, general learning algorithm? (or a small number of them)

CogSci: how does the mind learn abstract concepts on top of less abstract ones?

Deep Learning addresses the problem of learning hierarchical representations with a single algorithm or perhaps with a few algorithms
The Mammalian Visual Cortex is Hierarchical

- The ventral (recognition) pathway in the visual cortex has multiple stages:
  - Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations
Let's be inspired by nature, but not too much

- It's nice to imitate Nature,
- But we also need to understand
  - How do we know which details are important?
  - Which details are merely the result of evolution, and the constraints of biochemistry?
- For airplanes, we developed aerodynamics and compressible fluid dynamics.
  - We figured that feathers and wing flapping weren't crucial

QUESTION: What is the equivalent of aerodynamics for understanding intelligence?

L'Avion III de Clément Ader, 1897
(Musée du CNAM, Paris)
His “Eole” took off from the ground in 1890, 13 years before the Wright Brothers, but you probably never heard of it (unless you are French).
A hierarchy of trainable feature transforms

- Each module transforms its input representation into a higher-level one.
- High-level features are more global and more invariant
- Low-level features are shared among categories

How can we make all the modules trainable and get them to learn appropriate representations?
Three Types of Deep Architectures

- **Feed-Forward**: multilayer neural nets, convolutional nets

- **Feed-Back**: Stacked Sparse Coding, Deconvolutional Nets [Zeiler et al.]

- **Bi-Directional**: Deep Boltzmann Machines, Stacked Auto-Encoders
Three Types of Training Protocols

Purely Supervised
- Initialize parameters randomly
- Train in supervised mode
  - typically with SGD, using backprop to compute gradients
- Used in most practical systems for speech and image recognition

Unsupervised, layerwise + supervised classifier on top
- Train each layer unsupervised, one after the other
- Train a supervised classifier on top, keeping the other layers fixed
- Good when very few labeled samples are available

Unsupervised, layerwise + global supervised fine-tuning
- Train each layer unsupervised, one after the other
- Add a classifier layer, and retrain the whole thing supervised
- Good when label set is poor (e.g. pedestrian detection)

Unsupervised pre-training often uses regularized auto-encoders
Do we really need deep architectures?

Theoretician's dilemma: “We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?”

\[ y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \quad y = F(W^1 . F(W^0 . X)) \]

Kernel machines (and 2-layer neural nets) are “universal”.

Deep learning machines

\[ y = F(W^K . F(W^{K-1} . F(\ldots F(W^0 . X)\ldots))) \]

Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition.

They can represent more complex functions with less “hardware”.

We need an efficient parameterization of the class of functions that are useful for “AI” tasks (vision, audition, NLP...).
Why would deep architectures be more efficient?
[Bengio & LeCun 2007 “Scaling Learning Algorithms Towards AI”]

A deep architecture trades space for time (or breadth for depth)
► more layers (more sequential computation),
► but less hardware (less parallel computation).

Example 1: N-bit parity
► requires N-1 XOR gates in a tree of depth log(N).
► Even easier if we use threshold gates
► requires an exponential number of gates of we restrict ourselves to 2 layers (DNF formula with exponential number of minterms).

Example 2: circuit for addition of 2 N-bit binary numbers
► Requires O(N) gates, and O(N) layers using N one-bit adders with ripple carry propagation.
► Requires lots of gates (some polynomial in N) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
► Bad news: almost all boolean functions have a DNF formula with an exponential number of minterms O(2^N).....
Shallow vs Deep == lookup table vs multi-step algorithm

- “shallow & wide” vs “deep and narrow” == “more memory” vs “more time”
- Look-up table vs algorithm
- Few functions can be computed in two steps without an exponentially large lookup table
- Using more than 2 steps can reduce the “memory” by an exponential factor.

Step 1 (look up table/templates)

- Step 1
- Step 2
- Step 3
- Step 4
Which Models are Deep?

- 2-layer models are not deep (even if you train the first layer)
  - Because there is no feature hierarchy

- Neural nets with 1 hidden layer are not deep

- SVMs and Kernel methods are not deep
  - Layer1: kernels; layer2: linear
  - The first layer is “trained” in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
  - “glorified template matching”

- Classification trees are not deep
  - No hierarchy of features. All decisions are made in the input space
There is no opposition between graphical models and deep learning.
  ► Many deep learning models are formulated as factor graphs
  ► Some graphical models use deep architectures inside their factors

Graphical models can be deep (but most are not).

Factor Graph: sum of energy functions
  ► Over inputs $X$, outputs $Y$ and latent variables $Z$. Trainable parameters: $W$

$$-\log P(X, Y, Z / W) \propto E(X, Y, Z, W) = \sum_i E_i(X, Y, Z, W_i)$$

Each energy function can contain a deep network

The whole factor graph can be seen as a deep network
Deep Learning involves non-convex loss functions

- With non-convex losses, all bets are off
- Then again, every speech recognition system ever deployed has used non-convex optimization (GMMs are non convex).

But to some of us all “interesting” learning is non convex

- Convex learning is invariant to the order in which sample are presented (only depends on asymptotic sample frequencies).
- Human learning isn't like that: we learn simple concepts before complex ones. The order in which we learn things matter.
No generalization bounds?
- Actually, the usual VC bounds apply: most deep learning systems have a finite VC dimension
- We don't have tighter bounds than that.
- But then again, how many bounds are tight enough to be useful for model selection?

It's hard to prove anything about deep learning systems
- Then again, if we only study models for which we can prove things, we wouldn't have speech, handwriting, and visual object recognition systems today.
Deep Learning is about representing high-dimensional data

- There has to be interesting theoretical questions there
- What is the geometry of natural signals?
- Is there an equivalent of statistical learning theory for unsupervised learning?
- What are good criteria on which to base unsupervised learning?

Deep Learning Systems are a form of latent variable factor graph

- Internal representations can be viewed as latent variables to be inferred, and deep belief networks are a particular type of latent variable models.
- The most interesting deep belief nets have intractable loss functions: how do we get around that problem?

Lots of theory at the 2012 IPAM summer school on deep learning

- Wright's parallel SGD methods, Mallat's “scattering transform”, Osher's “split Bregman” methods for sparse modeling, Morton's “algebraic geometry of DBN”,...
Deep Learning has been the hottest topic in speech recognition in the last 2 years
- A few long-standing performance records were broken with deep learning methods
- Microsoft and Google have both deployed DL-based speech recognition system in their products
- Microsoft, Google, IBM, Nuance, AT&T, and all the major academic and industrial players in speech recognition have projects on deep learning

Deep Learning is the hottest topic in Computer Vision
- Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
- But the record holders on ImageNet and Semantic Segmentation are convolutional nets

Deep Learning is becoming hot in Natural Language Processing

Deep Learning/Feature Learning in Applied Mathematics
- The connection with Applied Math is through sparse coding, non-convex optimization, stochastic gradient algorithms, etc...
In Many Fields, Feature Learning Has Caused a Revolution (methods used in commercially deployed systems)

- **Speech Recognition I (late 1980s)**
  - Trained mid-level features with Gaussian mixtures (2-layer classifier)

- **Handwriting Recognition and OCR (late 1980s to mid 1990s)**
  - Supervised convolutional nets operating on pixels

- **Face & People Detection (early 1990s to mid 2000s)**
  - Haar features generation/selection (Viola-Jones 2001)

- **Object Recognition I (mid-to-late 2000s: Ponce, Schmid, Yu, YLC....)**
  - Trainable mid-level features (K-means or sparse coding)

- **Low-Res Object Recognition: road signs, house numbers (early 2010's)**
  - Supervised convolutional net operating on pixels

- **Speech Recognition II (circa 2011)**
  - Deep neural nets for acoustic modeling

- **Object Recognition III, Semantic Labeling (2012, Hinton, YLC,...)**
  - Supervised convolutional nets operating on pixels
In Several Fields, Feature Learning Has Caused Revolutions: Speech Recognition, Handwriting Recognition

- U = unsupervised, S = supervised, X = unsupervised + supervised
- Low-level feat. → mid-level feat. → classifier → contextual post-proc

### Speech Recognition
- Early 1980s: Dyn. time Warping
- Late 1980s: Gaussian Mix. Model
- 1990s: discriminative GMM
- 2010: deep neural nets

### Handwriting Recognition and OCR
- Early 80's: features + classifier
- Late 80's: supervised convnet
- Mid 90's: convnet + CRF
In Several Fields, Feature Learning Has Caused Revolutions: Object Detection, Object Recognition, Scene Labeling

- **Face & People Detection (1993-now)**
  - Supervised ConvNet on pixels (93, 94, 05, 07)
  - Selected Haar features + Adaboost (2001)
  - Unsup+Sup ConvNet on raw pixels (2011)

- **Object Recognition**
  - SIFT/HoG+sparse code+pool+SVM (06)
  - unsup+sup convnet (07,10)
  - supervised convnet (2012)

- **Semantic Segmentation / scene labeling**
  - unsup mid-lvl, CRF (2009, 10, 11, 12)
  - supervised convnet (2008, 12, 13)
What Are Good Feature?
Discovering the Hidden Structure in High-Dimensional Data

The manifold hypothesis

- **Learning Representations of Data:**
  - **Discovering & disentangling the independent explanatory factors**

- **The Manifold Hypothesis:**
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data...
Example: all face images of a person

- 1000x1000 pixels = 1,000,000 dimensions
- But the face has 3 cartesian coordinates and 3 Euler angles
- And humans have less than about 50 muscles in the face
- Hence the manifold of face images for a person has <56 dimensions

The perfect representations of a face image:

- Its coordinates on the face manifold
- Its coordinates away from the manifold

We do not have good and general methods to learn functions that turns an image into this kind of representation

Ideal Feature Extractor

| 1.2 |
| -3  |
| 0.2 |
| -2... |

Face/not face
Pose
Lighting
Expression
Data Manifold & Invariance: Some variations must be eliminated

Azimuth-Elevation manifold. Ignores lighting. [Hadsell et al. CVPR 2006]
Basic Idea for Invariant Feature Learning

Embed the input **non-linearly** into a high(er) dimensional space
- In the new space, things that were non separable may become separable

Pool regions of the new space together
- Bringing together things that are semantically similar. Like pooling.

- Input
- Non-Linear Function
- Pooling Or Aggregation
- Stable/invariant features

- high-dim
- Unstable/non-smooth features
Non-Linear Expansion → Pooling

Entangled data manifolds

Non-Linear Dim Expansion, Disentangling

Pooling, Aggregation
Use clustering to break things apart, pool together similar things
Overall Architecture:
Normalization → Filter Bank → Non-Linearity → Pooling

- Stacking multiple stages of
  - [Normalization → Filter Bank → Non-Linearity → Pooling].

- **Normalization:** variations on whitening
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization

- **Filter Bank:** dimension expansion, projection on overcomplete basis

- **Non-Linearity:** sparsification, saturation, lateral inhibition...
  - Rectification (ReLU), Component-wise shrinkage, tanh, winner-takes-all

- **Pooling:** aggregation over space or feature type

\[
X_i; \quad L_p: \sqrt[p]{X_i^p}; \quad PROB: \frac{1}{b} \log \left( \sum_i e^{bX_i} \right)
\]
Deep Supervised Learning (modular approach)
Multimodule Systems: Cascade

Complex learning machines can be built by assembling modules into networks

Simple example: sequential/layered feed-forward architecture (cascade)

Forward Propagation:

let $X = X_0$,

$$X_i = F_i(X_{i-1}, W_i) \quad \forall i \in [1, n]$$

$$E(Y, X, W) = C(X_n, Y)$$
Multimodule Systems: Implementation

- Each module is an object
  - Contains trainable parameters
  - Inputs are arguments
  - Output is returned, but also stored internally
  - Example: 2 modules m1, m2

- Torch7 (by hand)
  - hid = m1:forward(in)
  - out = m2:forward(hid)

- Torch7 (using the nn.Sequential class)
  - model = nn.Sequential()
  - model:add(m1)
  - model:add(m2)
  - out = model:forward(in)
To train a multi-module system, we must compute the gradient of $E$ with respect to all the parameters in the system (all the $W_i$).

Let’s consider module $i$ whose fprop method computes $X_i = F_i(X_{i-1}, W_i)$.

Let’s assume that we already know $\frac{\partial E}{\partial X_i}$, in other words, for each component of vector $X_i$ we know how much $E$ would wiggle if we wiggled that component of $X_i$. 
We can apply chain rule to compute $\frac{\partial E}{\partial W_i}$ (how much $E$ would wiggle if we wiggled each component of $W_i$):

$$\frac{\partial E}{\partial W_i} = \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i}$$

$$[1 \times N_w] = [1 \times N_x].[N_x \times N_w]$$

$\frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i}$ is the Jacobian matrix of $F_i$ with respect to $W_i$.

$$\left[ \frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i} \right]_{kl} = \frac{\partial [F_i(X_{i-1}, W_i)]_k}{\partial [W_i]_l}$$

Element $(k, l)$ of the Jacobian indicates how much the $k$-th output wiggles when we wiggle the $l$-th weight.
Using the same trick, we can compute $\frac{\partial E}{\partial X_{i-1}}$. Let’s assume again that we already know $\frac{\partial E}{\partial X_i}$, in other words, for each component of vector $X_i$ we know how much $E$ would wiggle if we wiggled that component of $X_i$.

- We can apply chain rule to compute $\frac{\partial E}{\partial X_{i-1}}$ (how much $E$ would wiggle if we wiggled each component of $X_{i-1}$):

$$\frac{\partial E}{\partial X_{i-1}} = \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}}$$

- $\frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}}$ is the Jacobian matrix of $F_i$ with respect to $X_{i-1}$.

- $F_i$ has two Jacobian matrices, because it has to arguments.

- Element $(k, l)$ of this Jacobian indicates how much the $k$-th output wiggles when we wiggle the $l$-th input.

- The equation above is a recurrence equation!
Derivatives with respect to a column vector are line vectors (dimensions: $[1 \times N_{i-1}] = [1 \times N_i] \ast [N_i \times N_{i-1}]$)

\[
\frac{\partial E}{\partial X_{i-1}} = \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}}
\]

(dimensions: $[1 \times N_{wi}] = [1 \times N_i] \ast [N_i \times N_{wi}]$):

\[
\frac{\partial E}{\partial W_i} = \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial W}
\]

We may prefer to write those equation with column vectors:

\[
\frac{\partial E'}{\partial X_{i-1}} = \frac{\partial F_i(X_{i-1}, W_i)'}{\partial X_{i-1}} \frac{\partial E'}{\partial X_i}
\]

\[
\frac{\partial E'}{\partial W_i} = \frac{\partial F_i(X_{i-1}, W_i)'}{\partial W} \frac{\partial E'}{\partial X_i}
\]
To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$

- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- **etc., until we reach the first module.**
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$. 
Multimodule Systems: Implementation

**Backpropagation through a module**
- Contains trainable parameters
- Inputs are arguments
- Gradient with respect to input is returned.
- Arguments are input and gradient with respect to output

**Torch7 (by hand)**
- \( \text{hidg} = \text{m2:backward(hid, outg)} \)
- \( \text{ing} = \text{m1:backward(in, hidg)} \)

**Torch7 (using the nn.Sequential class)**
- \( \text{ing} = \text{model:backward(in, outg)} \)
The input vector is multiplied by the weight matrix.

- **fprop:** \( X_{\text{out}} = WX_{\text{in}} \)
- **bprop to input:**
  \[
  \frac{\partial E}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} \frac{\partial X_{\text{out}}}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} W
  \]
- **by transposing, we get column vectors:**
  \[
  \frac{\partial E}{\partial X_{\text{in}}} = W' \frac{\partial E}{\partial X_{\text{out}}}
  \]
- **bprop to weights:**
  \[
  \frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial X_{\text{out}}} \frac{\partial X_{\text{outi}}}{\partial W_{ij}} = X_{\text{inj}} \frac{\partial E}{\partial X_{\text{outi}}}
  \]
- **We can write this as an outer-product:**
  \[
  \frac{\partial E'}{\partial W} = \frac{\partial E'}{\partial X_{\text{out}}} X_{\text{in}}'
  \]
Tanh module (or any other pointwise function)

- fprop: \((X_{out})_i = \tanh((X_{in})_i + B_i)\)

- bprop to input:
  \[
  (\frac{\partial E}{\partial X_{in}})_i = (\frac{\partial E}{\partial X_{out}})_i \tanh'(((X_{in})_i + B_i)
  \]

- bprop to bias:
  \[
  \frac{\partial E}{\partial B_i} = (\frac{\partial E}{\partial X_{out}})_i \tanh'((X_{in})_i + B_i)
  \]

- \(\tanh(x) = \frac{2}{1+\exp(-x)} - 1 = \frac{1-\exp(-x)}{1+\exp(-x)}\)
Euclidean Distance Module

- fprop: $X_{\text{out}} = \frac{1}{2} \| X_{\text{in}} - Y \|^2$
- bprop to $X$ input: $\frac{\partial E}{\partial X_{\text{in}}} = X_{\text{in}} - Y$
- bprop to $Y$ input: $\frac{\partial E}{\partial Y} = Y - X_{\text{in}}$
Any Architecture works

- **Any connection is permissible**
  - Networks with loops must be “unfolded in time”.

- **Any module is permissible**
  - As long as it is continuous and differentiable almost everywhere with respect to the parameters, and with respect to non-terminal inputs.
Module-Based Deep Learning with Torch7

**Torch7 is based on the Lua language**
- Simple and lightweight scripting language, dominant in the game industry
- Has a native just-in-time compiler (fast!)
- Has a simple foreign function interface to call C/C++ functions from Lua

**Torch7 is an extension of Lua with**
- A multidimensional array engine with CUDA and OpenMP backends
- A machine learning library that implements multilayer nets, convolutional nets, unsupervised pre-training, etc
- Various libraries for data/image manipulation and computer vision
- A quickly growing community of users

**Single-line installation on Ubuntu and Mac OSX:**
- `curl -s https://raw.github.com/clementfarabet/torchinstall/master/install-all | bash`

**Torch7 Machine Learning Tutorial (neural net, convnet, sparse auto-encoder):**
Example: building a Neural Net in Torch7

Net for SVHN digit recognition
10 categories
Input is 32x32 RGB (3 channels)
1500 hidden units

Creating a 2-layer net
Make a cascade module
Reshape input to vector
Add Linear module
Add tanh module
Add Linear Module
Add log softmax layer
Create loss function module

Noutputs = 10;
nfeats = 3; Width = 32; height = 32
ninputs = nfeats*width*height
nhiddens = 1500

-- Simple 2-layer neural network
model = nn.Sequential()
model:add(nn.Reshape(ninputs))
model:add(nn.Linear(ninputs,nhiddens))
model:add(nn.Tanh())
model:add(nn.Linear(nhiddens,noutputs))
model:add(nn.LogSoftMax())
criterion = nn.ClassNLLCriterion()

for t = 1, trainData:size(), batchSize do
    inputs, outputs = getNextBatch()
    local feval = function(x)
        parameters: copy(x)
        gradParameters: zero()
        local f = 0
        for i = 1, #inputs do
            local output = model: forward(inputs[i])
            local err = criterion: forward(output, targets[i])
            f = f + err
            local df_do = criterion: backward(output, targets[i])
            model: backward(inputs[i], df_do)
        end
        gradParameters: div(#inputs)
        f = f / #inputs
        return f, gradParameters
    end  — of feval
    optim.sgd(feval, parameters, optimState)
end  — one epoch over training set
Get next batch of samples
Create a “closure” feval(x) that takes the parameter vector as argument and returns the loss and its gradient on the batch.
Run model on batch
backprop
Normalize by size of batch
Return loss and gradient
call the stochastic gradient optimizer
Example: what is the loss function for the simplest 2-layer neural net ever

Function: 1-1-1 neural net. Map 0.5 to 0.5 and -0.5 to -0.5 (identity function) with quadratic cost:

\[ y = \tanh(W_1 \tanh(W_0.x)) \quad L = (0.5 - \tanh(W_1 \tanh(W_00.5))^2 \]
Use ReLU non-linearities (tanh and logistic are falling out of favor)
Use cross-entropy loss for classification
Use Stochastic Gradient Descent on minibatches
Shuffle the training samples
Normalize the input variables (zero mean, unit variance)
Schedule to decrease the learning rate
Use a bit of L1 or L2 regularization on the weights (or a combination)
  But it's best to turn it on after a couple of epochs
Use “dropout” for regularization
Lots more in [LeCun et al. “Efficient Backprop” 1998]
Convolutional Networks
Convolutional Nets

- Are deployed in many practical applications
  - Image reco, speech reco, Google's and Baidu's photo taggers

- Have won several competitions
  - ImageNet, Kaggle Facial Expression, Kaggle Multimodal Learning, German Traffic Signs, Connectomics, Handwriting....

- Are applicable to array data where nearby values are correlated
  - Images, sound, time-frequency representations, video, volumetric images, RGB-Depth images,.....

- One of the few models that can be trained purely supervised

```
input
83x83

Layer 1
64x75x7
5

9x9 convolution
(64 kernels)

Layer 2
64@14x14

10x10 pooling, 5x5 subsampling

Layer 3
256@6x6

9x9 convolution
(4096 kernels)

Layer 4
256@1x1

Output
101

6x6 pooling
4x4 subsamp
```
Example: 200x200 image
- Fully-connected, 400,000 hidden units = 16 billion parameters
- Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
- Local connections capture local dependencies
Features that are useful on one part of the image and probably useful elsewhere.

All units share the same set of weights

Shift equivariant processing:
- When the input shifts, the output also shifts but stays otherwise unchanged.

Convolution
- with a learned kernel (or filter)
- Non-linearity: ReLU (rectified linear)

\[ A_{ij} = \sum_{kl} W_{kl} X_{i+j, k+l} \]

The filtered “image” Z is called a feature map

\[ Z_{ij} = \max(0, A_{ij}) \]

Example: 200x200 image
- 400,000 hidden units with 10x10 fields = 1000 params
- 10 feature maps of size 200x200, 10 filters of size 10x10
Multiple Convolutions with Different Kernels

- Detects multiple motifs at each location
- The collection of units looking at the same patch is akin to a feature vector for that patch.
- The result is a 3D array, where each slice is a feature map.

Multiple convolutions
Early Hierarchical Feature Models for Vision

[Hubel & Wiesel 1962]:

- **simple cells** detect local features
- **complex cells** “pool” the outputs of simple cells within a retinotopic neighborhood.

Cognitron & Neocognitron [Fukushima 1974-1982]
The Convolutional Net Model
(Multistage Hubel-Wiesel system)

Input Image
1x500x500

Normalized Image
1x500x500

Local Divisive Normalization

Convolutions w/ filter bank:
20x7x7 kernels

Pooling:
20x4x4 kernels

Convs:
20x494x494

S2: 20x123x123

C1: 20x494x494

“Simple cells”

“Complex cells”

Multiple convolutions

pooling subsampling

Retinotopic Feature Maps

Retinotopic Feature Maps

Convs:
100x7x7 kernels

Pooling:
20x4x4 kernels

Convs:
800x7x7 kernels

Linear Classifier

Object Categories / Positions

{ at (x,y) }

F6:
Nx23x23

S4: 20x29x29

C3: 20x117x117

C5: 200x23x23

Training is supervised
With stochastic gradient descent

[LeCun et al. 89]
[LeCun et al. 98]
Feature Transform:
Normalization → Filter Bank → Non-Linearity → Pooling

Stacking multiple stages of
[Normalization → Filter Bank → Non-Linearity → Pooling].

Normalization: variations on whitening
- Subtractive: average removal, high pass filtering
- Divisive: local contrast normalization, variance normalization

Filter Bank: dimension expansion, projection on overcomplete basis

Non-Linearity: sparsification, saturation, lateral inhibition....
- Rectification, Component-wise shrinkage, tanh, winner-takes-all

Pooling: aggregation over space or feature type, subsampling
- $X_i; \quad L_p: \sqrt[p]{X_i^p}; \quad \text{PROB}: \frac{1}{b} \log \left( \sum_i e^{bX_i} \right)$
Feature Transform:
Normalization → Filter Bank → Non-Linearity → Pooling

**Filter Bank → Non-Linearity** = Non-linear embedding in high dimension

**Feature Pooling** = contraction, dimensionality reduction, smoothing

Learning the filter banks at every stage

Creating a hierarchy of features

Basic elements are inspired by models of the visual (and auditory) cortex

- Simple Cell + Complex Cell model of [Hubel and Wiesel 1962]
- Many “traditional” feature extraction methods are based on this
  - SIFT, GIST, HoG, SURF...

- [Fukushima 1974-1982], [LeCun 1988-now],
  - since the mid 2000: Hinton, Seung, Poggio, Ng,...
Convolutional Network (ConvNet)

- **Non-Linearity**: half-wave rectification, shrinkage function, sigmoid
- **Pooling**: average, L1, L2, max
Convolutional Network Architecture

Input high-pass filtered
contrast-normalized 83x83 (raw: 91x91)

Filter Bank + Tanh + Gain
64 features 75x75
64 filters
9x9 kernels

Abs + Contrast Norm + Pooling + Downsampling
64 features 14x14
5x5 subsampling
10x10 pooling

STAGE 1

Filter Bank + Tanh + Gain
256 features 6x6
4096 filters
9x9 kernels

Abs + Contrast Norm + Pooling + Downsampling
256 features 1x1
4x4 subsampling
6x6 pooling

STAGE 2

Parzen Windows Classifier

CLASSIFIER
Convolutional Network (vintage 1990)

filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh

Curved manifold

Flatter manifold
“Mainstream” object recognition pipeline 2006-2012: somewhat similar to ConvNets

- **Fixed Features**
  - SIFT + Vector Quantization + Pyramid pooling + SVM
    - [Lazebnik et al. CVPR 2006]
  - SIFT + Local Sparse Coding Macrofeatures + Pyramid pooling + SVM
    - [Boureau et al. ICCV 2011]
  - SIFT + Fisher Vectors + Deformable Parts Pooling + SVM
    - [Perronin et al. 2012]

- **Unsupervised**
  - K-means Sparse Coding + Spatial Max Or average

- **Supervised**
  - Any simple classifier
Tasks for Which Deep Convolutional Nets are the Best

- **Handwriting recognition** MNIST (many), Arabic HWX (IDSIA)
- **OCR in the Wild [2011]**: StreetView House Numbers (NYU and others)
- **Traffic sign recognition [2011]** GTSRB competition (IDSIA, NYU)
- **Pedestrian Detection [2013]**: INRIA datasets and others (NYU)
- **Volumetric brain image segmentation [2009]** connectomics (IDSIA, MIT)
- **Human Action Recognition [2011]** Hollywood II dataset (Stanford)
- **Object Recognition [2012]** ImageNet competition
- **Scene Parsing [2012]** Stanford bgd, SiftFlow, Barcelona (NYU)
- **Scene parsing from depth images [2013]** NYU RGB-D dataset (NYU)
- **Speech Recognition [2012]** Acoustic modeling (IBM and Google)
- **Breast cancer cell mitosis detection [2011]** MITOS (IDSIA)

The list of perceptual tasks for which ConvNets hold the record is growing.
Most of these tasks (but not all) use purely supervised convnets.
Ideas from Neuroscience and Psychophysics

- The whole architecture: simple cells and complex cells
- Local receptive fields
- Self-similar receptive fields over the visual field (convolutions)
- Pooling (complex cells)
- Non-Linearity: Rectified Linear Units (ReLU)
- LGN-like band-pass filtering and contrast normalization in the input
- Divisive contrast normalization (from Heeger, Simoncelli....)
  - Lateral inhibition
- Sparse/Overcomplete representations (Olshausen-Field....)
- Inference of sparse representations with lateral inhibition
- Sub-sampling ratios in the visual cortex
  - between 2 and 3 between V1-V2-V4
- Crowding and visual metamers give cues on the size of the pooling areas
Simple ConvNet Applications with State-of-the-Art Performance

- **Traffic Sign Recognition (GTSRB)**
  - German Traffic Sign Recognition Bench
  - 99.2% accuracy

- **House Number Recognition (Google)**
  - Street View House Numbers
  - 94.3% accuracy
One Stage: Contrast Norm → Filter Bank → Shrinkage → L2 Pooling

THIS IS ONE STAGE OF THE CONVNET
Results on Caltech101 with sigmoid non-linearity

<table>
<thead>
<tr>
<th>Single Stage System:</th>
<th>[64.F_{C_{SG}}^{9\times9} - R/N/P_{5\times5}] - log_reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/N/P</td>
<td>R_{abs} - N - P_A</td>
</tr>
<tr>
<td>U+</td>
<td>54.2%</td>
</tr>
<tr>
<td>R+</td>
<td>54.8%</td>
</tr>
<tr>
<td>U</td>
<td>52.2%</td>
</tr>
<tr>
<td>R</td>
<td>53.3%</td>
</tr>
<tr>
<td>G</td>
<td>52.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Two Stage System:</th>
<th>[64.F_{C_{SG}}^{9\times9} - R/N/P_{5\times5}] - [256.F_{C_{SG}}^{9\times9} - R/N/P_{4\times4}] - log_reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/N/P</td>
<td>R_{abs} - N - P_A</td>
</tr>
<tr>
<td>U+U+</td>
<td>65.5%</td>
</tr>
<tr>
<td>R+R+</td>
<td>64.7%</td>
</tr>
<tr>
<td>UU</td>
<td>63.7%</td>
</tr>
<tr>
<td>RR</td>
<td>62.9%</td>
</tr>
<tr>
<td>GT</td>
<td>55.8%</td>
</tr>
</tbody>
</table>

← like HMAX model

<table>
<thead>
<tr>
<th>Single Stage:</th>
<th>[64.F_{C_{SG}}^{9\times9} - R/N/P_{5\times5}] - PMK-SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>64.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Two Stages:</th>
<th>[64.F_{C_{SG}}^{9\times9} - R/N/P_{5\times5}] - [256.F_{C_{SG}}^{9\times9} - R/N] - PMK-SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>UU</td>
<td>52.8%</td>
</tr>
</tbody>
</table>
Local Contrast Normalization

- Performed on the state of every layer, including the input

- **Subtractive Local Contrast Normalization**
  - Subtracts from every value in a feature a Gaussian-weighted average of its neighbors (high-pass filter)

- **Divisive Local Contrast Normalization**
  - Divides every value in a layer by the standard deviation of its neighbors over space and over all feature maps

- **Subtractive + Divisive LCN performs a kind of approximate whitening.**
The Effect of Architectural Elements

- Pyramid pooling on last layer: 1% improvement over regular pooling
- Shrinkage non-linearity + lateral inhibition: 1.6% improvement over tanh
- Discriminative term in sparse coding: 2.8% improvement

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Protocol</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) $F_{\text{tanh}} - R_{\text{abs}} - N - P_A^{\text{pyr}}$</td>
<td>$\mathbf{R}^+\mathbf{R}^+$</td>
<td>65.4 ± 1.0</td>
</tr>
<tr>
<td>(2) $F_{\text{tanh}} - R_{\text{abs}} - N - P_A^{\text{pyr}}$</td>
<td>$\mathbf{U}^+\mathbf{U}^+$</td>
<td>66.2 ± 1.0</td>
</tr>
<tr>
<td>(3) $F_{si} - R_{\text{abs}} - N - P_A$</td>
<td>$\mathbf{R}^+\mathbf{R}^+$</td>
<td>63.3 ± 1.0</td>
</tr>
<tr>
<td>(4) $F_{si} - R_{\text{abs}} - N - P_A$</td>
<td>$\mathbf{U}\mathbf{U}$</td>
<td>60.4 ± 0.6</td>
</tr>
<tr>
<td>(5) $F_{si} - R_{\text{abs}} - N - P_A$</td>
<td>$\mathbf{U}^+\mathbf{U}^+$</td>
<td>66.4 ± 0.5</td>
</tr>
<tr>
<td>(6) $F_{si} - R_{\text{abs}} - N - P_A^{\text{pyr}}$</td>
<td>$\mathbf{U}^+\mathbf{U}^+$</td>
<td>67.8 ± 0.4</td>
</tr>
<tr>
<td>(7) $F_{si} - R_{\text{abs}} - N - P_A$</td>
<td>$\mathbf{D}\mathbf{D}$</td>
<td>66.0 ± 0.3</td>
</tr>
<tr>
<td>(8) $F_{si} - R_{\text{abs}} - N - P_A$</td>
<td>$\mathbf{D}^+\mathbf{D}^+$</td>
<td>68.7 ± 0.2</td>
</tr>
<tr>
<td>(9) $F_{si} - R_{\text{abs}} - N - P_A^{\text{pyr}}$</td>
<td>$\mathbf{D}^+\mathbf{D}^+$</td>
<td>70.6 ± 0.3</td>
</tr>
</tbody>
</table>
Results on Caltech101: purely supervised with soft-shrink, L2 pooling, contrast normalization

- Supervised learning with soft-shrinkage non-linearity, L2 complex cells, and sparsity penalty on the complex cell outputs: 71%
- Caltech101 is pathological, biased, too small, etc...

![Bar chart showing classification accuracies for different models and datasets on Caltech101.](chart-image)
What does Local Contrast Normalization Do?
Why Do Random Filters Work?

Random Filters For Simple Cells

Trained Filters For Simple Cells

Optimal Stimuli for each Complex Cell
Small NORB dataset

Two-stage system: error rate versus number of labeled training samples

- No normalization
- Random filters
- Unsup filters
- Sup filters
- Unsup+Sup filters
Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

Won the 2012 ImageNet LSVRC. 60 Million parameters, 832M MAC ops

<table>
<thead>
<tr>
<th>Layer Description</th>
<th>Parameters</th>
<th>MAC Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL CONNECT</td>
<td>4M</td>
<td>4Mflop</td>
</tr>
<tr>
<td>FULL 4096/ReLU</td>
<td>16M</td>
<td></td>
</tr>
<tr>
<td>FULL 4096/ReLU</td>
<td>37M</td>
<td></td>
</tr>
<tr>
<td>MAX POOLING</td>
<td>442K</td>
<td>74M</td>
</tr>
<tr>
<td>CONV 3x3/ReLU 256fm</td>
<td>1.3M</td>
<td>224M</td>
</tr>
<tr>
<td>CONV 3x3/ReLU 384fm</td>
<td>884K</td>
<td>149M</td>
</tr>
<tr>
<td>MAX POOLING 2x2sub</td>
<td>307K</td>
<td>223M</td>
</tr>
<tr>
<td>LOCAL CONTRAST NORM</td>
<td>35K</td>
<td>105M</td>
</tr>
<tr>
<td>CONV 11x11/ReLU 256fm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONV 3x3/ReLU 384fm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONV 11x11/ReLU 96fm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ImageNet Large Scale Visual Recognition Challenge
1000 categories, 1.5 Million labeled training samples
Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

- **Method: large convolutional net**
  - 650K neurons, 832M synapses, 60M parameters
  - Trained with backprop on GPU
  - Trained “with all the tricks Yann came up with in the last 20 years, plus dropout” (Hinton, NIPS 2012)
  - Rectification, contrast normalization,...

- **Error rate: 15% (whenever correct class isn't in top 5)**
- **Previous state of the art: 25% error**

**A REVOLUTION IN COMPUTER VISION**

- Acquired by Google in Jan 2013
- Deployed in Google+ Photo Tagging in May 2013
Object Recognition [Krizhevsky, Sutskever, Hinton 2012]
Object Recognition [Krizhevsky, Sutskever, Hinton 2012]
ConvNet-Based Google+ Photo Tagger

🔍 Searched my personal collection for “bird”

Samy Bengio

???
Another ImageNet-trained ConvNet
[Zeiler & Fergus 2013]

- Convolutional Net with 8 layers, input is 224x224 pixels
  - conv-pool-conv-pool-conv-conv-full-full-full
  - Rectified-Linear Units (ReLU): $y = \max(0, x)$
  - Divisive contrast normalization across features [Jarrett et al. ICCV 2009]

- Trained on ImageNet 2012 training set
  - 1.3M images, 1000 classes
  - 10 different crops/flips per image

- Regularization: Dropout
  - [Hinton 2012]
  - zeroing random subsets of units

- Stochastic gradient descent
  - for 70 epochs (7-10 days)
  - With learning rate annealing
Object Recognition on-line demo [Zeiler & Fergus 2013]

http://horatio.cs.nyu.edu

Image Classifier Demo

Upload your images to have them classified by a machine! Upload multiple images using the button below or dropping them on this page. The predicted objects will be refreshed automatically. Images are resized such that the smallest dimension becomes 256, then the center 256x256 crop is used. More about the demo can be found here.

Upload Images  Remove All  Show help tips

I agree to the Terms of Use

Demo Notes

- If your images have objects that are not in the 1,000 categories of ImageNet, the model will not know about them.
- Other objects can be added from all 20,000+ ImageNet categories (it may be slow to load the autocomplete results...just wait a little).
- The maximum file size for uploads in this demo is 10 MB.
- Only image files (JPEG, JPG, GIF, PNG) are allowed in this demo.
- You can drag & drop files from your desktop on this webpage with Google Chrome, Mozilla Firefox and Apple Safari.
- Some mobile browsers are known to work, others will not. Try updating your browser or contact us with the problem.
- All images for your current IP and browsing session are shown above and not shown to others.
- This demo is powered by research out of New York University. Click here to find out more
- If you encounter problems, please contact zeiler@cs.nyu.edu

Demo created by: Matthew Zeiler
<table>
<thead>
<tr>
<th>Error %</th>
<th>Val Top-1</th>
<th>Val Top-5</th>
<th>Test Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deng et al. SIFT + FV [7]</td>
<td>---</td>
<td>---</td>
<td>26.2</td>
</tr>
<tr>
<td>Krizhevsky et al. [12], 1 convnet</td>
<td>40.7</td>
<td>18.2</td>
<td>---</td>
</tr>
<tr>
<td>Krizhevsky et al. [12], 5 convnets</td>
<td>38.1</td>
<td>16.4</td>
<td>16.4</td>
</tr>
<tr>
<td>*Krizhevsky et al. [12], 1 convnets</td>
<td>39.0</td>
<td>16.6</td>
<td>---</td>
</tr>
<tr>
<td>*Krizhevsky et al. [12], 7 convnets</td>
<td>36.7</td>
<td>15.4</td>
<td>15.3</td>
</tr>
<tr>
<td>Our replication of [12], 1 convnet</td>
<td>41.7</td>
<td>19.0</td>
<td>---</td>
</tr>
<tr>
<td>1 convnet - our model</td>
<td>38.4 ± 0.05</td>
<td>16.5 ± 0.05</td>
<td>---</td>
</tr>
<tr>
<td>5 convnets - our model (a)</td>
<td>36.7</td>
<td>15.3</td>
<td>15.3</td>
</tr>
<tr>
<td>1 convnet - tweaked model (b)</td>
<td>37.5</td>
<td>16.0</td>
<td>16.1</td>
</tr>
<tr>
<td>6 convnets, (a) &amp; (b) combined</td>
<td>36.0</td>
<td>14.7</td>
<td>14.8</td>
</tr>
</tbody>
</table>
Features are generic: Caltech 256

Network first trained on ImageNet.

Last layer chopped off

Last layer trained on Caltech 256, first layers N-1 kept fixed.

State of the art with only 6 training examples

State of the art accuracy with only 6 training samples/class

<table>
<thead>
<tr>
<th># Train</th>
<th>Acc % 15/class</th>
<th>Acc % 30/class</th>
<th>Acc % 45/class</th>
<th>Acc % 60/class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sohn et al. [16]</td>
<td>35.1</td>
<td>42.1</td>
<td>45.7</td>
<td>47.9</td>
</tr>
<tr>
<td>Bo et al. [3]</td>
<td>40.5 ± 0.4</td>
<td>48.0 ± 0.2</td>
<td>51.9 ± 0.2</td>
<td>55.2 ± 0.3</td>
</tr>
<tr>
<td>Non-pretr.</td>
<td>9.0 ± 1.4</td>
<td>22.5 ± 0.7</td>
<td>31.2 ± 0.5</td>
<td>38.8 ± 1.4</td>
</tr>
<tr>
<td>ImageNet-pretr.</td>
<td>65.7 ± 0.2</td>
<td>70.6 ± 0.2</td>
<td>72.7 ± 0.4</td>
<td>74.2 ± 0.3</td>
</tr>
</tbody>
</table>

Features are generic: PASCAL VOC 2012

- Network first trained on ImageNet.
- Last layer trained on Pascal VOC, keeping N-1 first layers fixed.

<table>
<thead>
<tr>
<th>Category</th>
<th>Acc % [15]</th>
<th>Acc % [19]</th>
<th>Acc % Ours</th>
<th># won</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airplane</td>
<td>92.0</td>
<td>97.3</td>
<td>96.0</td>
<td>0</td>
</tr>
<tr>
<td>Bicycle</td>
<td>74.2</td>
<td>84.2</td>
<td>77.1</td>
<td>15</td>
</tr>
<tr>
<td>Bird</td>
<td>73.0</td>
<td>80.8</td>
<td>88.4</td>
<td>5</td>
</tr>
<tr>
<td>Boat</td>
<td>77.5</td>
<td>85.3</td>
<td>85.5</td>
<td></td>
</tr>
<tr>
<td>Bottle</td>
<td>54.3</td>
<td>60.8</td>
<td>55.8</td>
<td></td>
</tr>
<tr>
<td>Bus</td>
<td>85.2</td>
<td>89.9</td>
<td>85.8</td>
<td></td>
</tr>
<tr>
<td>Car</td>
<td>81.9</td>
<td>86.8</td>
<td>78.6</td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td>76.4</td>
<td>89.3</td>
<td>91.2</td>
<td></td>
</tr>
<tr>
<td>Chair</td>
<td>65.2</td>
<td>75.4</td>
<td>65.0</td>
<td></td>
</tr>
<tr>
<td>Cow</td>
<td>63.2</td>
<td>77.8</td>
<td>74.4</td>
<td></td>
</tr>
<tr>
<td>Dining table</td>
<td>63.2</td>
<td>77.8</td>
<td>67.7</td>
<td></td>
</tr>
<tr>
<td>Dog</td>
<td>68.9</td>
<td>83.0</td>
<td>87.8</td>
<td></td>
</tr>
<tr>
<td>Horse</td>
<td>78.2</td>
<td>87.5</td>
<td>86.0</td>
<td></td>
</tr>
<tr>
<td>Motorbike</td>
<td>81.0</td>
<td>90.1</td>
<td>85.1</td>
<td></td>
</tr>
<tr>
<td>Person</td>
<td>91.6</td>
<td>95.0</td>
<td>90.9</td>
<td></td>
</tr>
<tr>
<td>Potted plant</td>
<td>55.9</td>
<td>57.8</td>
<td>52.2</td>
<td></td>
</tr>
<tr>
<td>Sheep</td>
<td>69.4</td>
<td>79.2</td>
<td>83.6</td>
<td></td>
</tr>
<tr>
<td>Sofa</td>
<td>65.4</td>
<td>73.4</td>
<td>61.1</td>
<td></td>
</tr>
<tr>
<td>Train</td>
<td>86.7</td>
<td>94.5</td>
<td>91.8</td>
<td></td>
</tr>
<tr>
<td>Tv/monitor</td>
<td>77.4</td>
<td>80.7</td>
<td>76.1</td>
<td></td>
</tr>
</tbody>
</table>

Traditional Detectors/Classifiers must be applied to every location on a large input image, at multiple scales.

Convolutional nets can replicated over large images very cheaply.

The network is applied to multiple scales spaced by 1.5.
Computational cost for replicated convolutional net:
- 96x96 -> 4.6 million multiply-accumulate operations
- 120x120 -> 8.3 million multiply-accumulate ops
- 240x240 -> 47.5 million multiply-accumulate ops
- 480x480 -> 232 million multiply-accumulate ops

Computational cost for a non-convolutional detector of the same size, applied every 12 pixels:
- 96x96 -> 4.6 million multiply-accumulate operations
- 120x120 -> 42.0 million multiply-accumulate operations
- 240x240 -> 788.0 million multiply-accumulate ops
- 480x480 -> 5,083 million multiply-accumulate ops
ConvNets for Image Segmentation

- **Biological Image Segmentation**
  - [Ning et al. IEEE-TIP 2005]

- **Pixel labeling with large context using a convnet**

- **Cleanup using a CRF**
  - Similar to a field of expert
3D ConvNet
Volumetric Images

Each voxel labeled as "membrane" or "non-membrane" using a 7x7x7 voxel neighborhood
**Image Labeling for Off-Road Robots [Hadsell JFR 2008]**

- ConvNet labels pixels as one of 3 categories
- Traversable/flat (green), non traversible (red), foot of obstacle (purple)
- Labels obtained from stereo vision and SLAM
Pedestrian Detection, Face Detection

[Osadchy, Miller LeCun JMLR 2007], [Kavukcuoglu et al. NIPS 2010] [Sermanet et al. CVPR 2013]
Feature maps from all stages are pooled/subsampled and sent to the final classification layers

- Pooled low-level features: good for textures and local motifs
- High-level features: good for “gestalt” and global shape

**ConvNet Architecture with Multi-Stage Features**

<table>
<thead>
<tr>
<th>Task</th>
<th>Single-Stage features</th>
<th>Multi-Stage features</th>
<th>Improvement %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pedestrians detection (INRIA)</td>
<td>14.26%</td>
<td>9.85%</td>
<td>31%</td>
</tr>
<tr>
<td>Traffic Signs classification (GTSRB) [33]</td>
<td>1.80%</td>
<td>0.83%</td>
<td>54%</td>
</tr>
<tr>
<td>House Numbers classification (SVHN) [32]</td>
<td>5.54%</td>
<td>5.36%</td>
<td>3.2%</td>
</tr>
</tbody>
</table>

[Sermanet, Chintala, LeCun CVPR 2013]
Pedestrian Detection: INRIA Dataset. Miss rate vs false positives

- ConvNet Color+Skip Supervised
- ConvNet Color+Skip Unsup+Sup
- ConvNet B&W Supervised
- ConvNet B&W Unsup+Sup

[_area under curve (0,1) FPP vs Miss rate
- Shapelet-orig (94.71%)
- PoseInvSvm (79.04%)
- PoseInv (72.02%)
- Shapelet (65.09%)
- Vj-OpenCv (52.35%)
- Vj (57.94%)
- FtrMine (44.36%)
- HOG (33.52%)
- Pls (30.49%)
- HikSvm (30.13%)
- LatSvm-V1 (28.20%)
- ConvNet-Supervised (26.05%)
- MultiFtr (22.76%)
- ConvNet-MRC-Supervised (20.43%)
- ConvNet-Unsup (17.81%)
- MultiFtr+CSS (16.18%)
- LatSvm-V2 (14.39%)
- FPDW (13.17%)
- ChnFtrs (12.92%)
- ConvNet-MRC-Unsup (11.05%)

[1.0 FPPI
- Shapelet-orig (91.13%)
- PoseInvSvm (88.76%)
- PoseInv (55.01%)
- Vj-OpenCv (52.97%)
- Shapelet (50.25%)
- Vj (47.37%)
- FtrMine (33.96%)
- Pls (23.26%)
- HOG (22.58%)
- HikSvm (20.54%)
- LatSvm-V1 (16.81%)
- MultiFtr (15.11%)
- ConvNet-Supervised (14.26%)
- MultiFtr+CSS (10.70%)
- ConvNet-Unsup (10.19%)
- ConvNet-MRC-Supervised (9.85%)
- FPDW (9.34%)
- LatSvm-V2 (8.66%)
- ChnFtrs (8.66%)
- ConvNet-MRC-Unsup (6.62%)

[Kavukcuoglu et al. NIPS 2010] [Sermanet et al. ArXiv 2012]
Results on “Near Scale” Images (>80 pixels tall, no occlusions)

Daimler
\( p=21790 \)

ETH
\( p=804 \)

TudBrussels
\( p=508 \)
Results on “Reasonable” Images (>50 pixels tall, few occlusions)

Daimler
p=21790

INRIA
p=288

ETH
p=804

TudBrussels
p=508
128 stage-1 filters on Y channel.

Unsupervised training with convolutional predictive sparse decomposition
Stage 2 filters.

Unsupervised training with convolutional predictive sparse decomposition
Semantic Labeling: Labeling every pixel with the object it belongs to

- Would help identify obstacles, targets, landing sites, dangerous areas
- Would help line up depth map with edge maps

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling: ConvNet Architecture

- Each output sees a large input context:
  - 46x46 window at full rez; 92x92 at ½ rez; 184x184 at ¼ rez
  - [7x7conv]—->[2x2pool]—->[7x7conv]—->[2x2pool]—->[7x7conv]—>
  - Trained supervised on fully-labeled images
Method 1: majority over super-pixel regions

- Input image
- Superpixel boundaries
- Features from Convolutional net (d=768 per pixel)
- Majority Vote Over Superpixels
- Categories aligned With region boundaries
- “soft” categories scores

[Farabet et al. IEEE T. PAMI 2013]
Method 2: optimal cover of purity tree

2-layer Neural net

2-layer Neural net

classifier

\( c (O_k; \theta_c) \)

\( \{O_k\} \)

masking/pooling

\( a (C_k, F) \)

\( F \)

\( C_k \)

Distribution of Categories within Each Segment

Spanning Tree From pixel Similarity graph

[Farabet et al. ICML 2012]
### Stanford Background Dataset [Gould 1009]: 8 categories

<table>
<thead>
<tr>
<th>Method</th>
<th>Pixel Acc.</th>
<th>Class Acc.</th>
<th>CT (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gould et al. 2009 [14]</td>
<td>76.4%</td>
<td>-</td>
<td>10 to 600s</td>
</tr>
<tr>
<td>Munoz et al. 2010 [32]</td>
<td>76.9%</td>
<td>66.2%</td>
<td>12s</td>
</tr>
<tr>
<td>Tighe et al. 2010 [46]</td>
<td>77.5%</td>
<td>-</td>
<td>10 to 300s</td>
</tr>
<tr>
<td>Socher et al. 2011 [45]</td>
<td>78.1%</td>
<td>-</td>
<td>?</td>
</tr>
<tr>
<td>Kumar et al. 2010 [22]</td>
<td>79.4%</td>
<td>-</td>
<td>&lt; 600s</td>
</tr>
<tr>
<td>Lempitzky et al. 2011 [28]</td>
<td>81.9%</td>
<td>72.4%</td>
<td>&gt; 60s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Approach</th>
<th>Pixel Acc.</th>
<th>Class Acc.</th>
<th>CT (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>singlescale convnet</td>
<td>66.0 %</td>
<td>56.5 %</td>
<td>0.35s</td>
</tr>
<tr>
<td>multiscale convnet</td>
<td>78.8 %</td>
<td>72.4%</td>
<td>0.6s</td>
</tr>
<tr>
<td>multiscale net + superpixels</td>
<td>80.4%</td>
<td>74.56%</td>
<td>0.7s</td>
</tr>
<tr>
<td>multiscale net + gPb + cover</td>
<td>80.4%</td>
<td>75.24%</td>
<td>61s</td>
</tr>
<tr>
<td>multiscale net + CRF on gPb</td>
<td>81.4%</td>
<td>76.0%</td>
<td>60.5s</td>
</tr>
</tbody>
</table>

[Farabet et al. IEEE T. PAMI 2013]
Scene Parsing/Labeling: Performance

<table>
<thead>
<tr>
<th></th>
<th>Pixel Acc.</th>
<th>Class Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liu et al. 2009 [31]</td>
<td>74.75%</td>
<td>-</td>
</tr>
<tr>
<td>Tighe et al. 2010 [44]</td>
<td>76.9%</td>
<td>29.4%</td>
</tr>
<tr>
<td>raw multiscale net¹</td>
<td>67.9%</td>
<td>45.9%</td>
</tr>
<tr>
<td>multiscale net + superpixels¹</td>
<td>71.9%</td>
<td>50.8%</td>
</tr>
<tr>
<td>multiscale net + cover¹</td>
<td>72.3%</td>
<td>50.8%</td>
</tr>
<tr>
<td>multiscale net + cover²</td>
<td>78.5%</td>
<td>29.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pixel Acc.</th>
<th>Class Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tighe et al. 2010 [44]</td>
<td>66.9%</td>
<td>7.6%</td>
</tr>
<tr>
<td>raw multiscale net¹</td>
<td>37.8%</td>
<td>12.1%</td>
</tr>
<tr>
<td>multiscale net + superpixels¹</td>
<td>44.1%</td>
<td>12.4%</td>
</tr>
<tr>
<td>multiscale net + cover¹</td>
<td>46.4%</td>
<td>12.5%</td>
</tr>
<tr>
<td>multiscale net + cover²</td>
<td>67.8%</td>
<td>9.5%</td>
</tr>
</tbody>
</table>

- SIFT Flow Dataset
- [Liu 2009]:
- 33 categories
- Barcelona dataset
- [Tighe 2010]:
- 170 categories.

[Farabet et al. IEEE T. PAMI 2012]
Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

Samples from the SIFT-Flow dataset (Liu)

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

[Y LeCun]

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

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

Causal method for temporal consistency

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

- Spatio-Temporal Super-Pixel segmentation
  - [Couprie et al ICIP 2013]
  - [Couprie et al JMLR under review]
  - Majority vote over super-pixels

Independent segmentations $S'_1, S'_2$ and $S'_3$

Temporally consistent segmentations $S_1(= S'_1), S_2$, and $S_3$
NYU RGB-Depth Indoor Scenes Dataset

- 407024 RGB-D images of apartments
- 1449 labeled frames, 894 object categories

[Silberman et al. 2012]
Captured with a Kinect on a steadycam
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bed 4.4%</td>
<td>30.3</td>
<td>38.1</td>
</tr>
<tr>
<td>objects 7.1%</td>
<td>10.9</td>
<td>8.7</td>
</tr>
<tr>
<td>chair 3.4%</td>
<td>44.4</td>
<td>34.1</td>
</tr>
<tr>
<td>furnit. 12.3%</td>
<td>28.5</td>
<td>42.4</td>
</tr>
<tr>
<td>ceiling 1.4%</td>
<td>33.2</td>
<td>62.6</td>
</tr>
<tr>
<td>floor 9.9%</td>
<td>68.0</td>
<td>87.3</td>
</tr>
<tr>
<td>deco. 3.4%</td>
<td>38.5</td>
<td>40.4</td>
</tr>
<tr>
<td>sofa 3.2%</td>
<td>25.8</td>
<td>24.6</td>
</tr>
<tr>
<td>table 3.7%</td>
<td>18.0</td>
<td>10.2</td>
</tr>
<tr>
<td>wall 24.5%</td>
<td>89.4</td>
<td>86.1</td>
</tr>
<tr>
<td>window 5.1%</td>
<td>37.8</td>
<td>15.9</td>
</tr>
<tr>
<td>books 2.9%</td>
<td>31.7</td>
<td>13.7</td>
</tr>
<tr>
<td>TV 1.0%</td>
<td>18.8</td>
<td>6.0</td>
</tr>
<tr>
<td>unkn. 17.8%</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

| Avg. Class Acc.   | -                                             | 35.8                        | 36.2                        |
| Pixel Accuracy (mean) | -                                             | 51.0                        | 52.4                        |
| Pixel Accuracy (median) | -                                             | 51.7                        | 52.9                        |
| Pixel Accuracy (std. dev.) | -                                             | 15.2                        | 15.2                        |
Results

Depth helps a bit
  - Helps a lot for floor and props
  - Helps surprisingly little for structures, and hurts for furniture

<table>
<thead>
<tr>
<th></th>
<th>Ground</th>
<th>Furniture</th>
<th>Props</th>
<th>Structure</th>
<th>Class Acc.</th>
<th>Pixel Acc.</th>
<th>Comput. time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silberman et al. (2012)</td>
<td>68</td>
<td>70</td>
<td>42</td>
<td>59</td>
<td>59.6</td>
<td>58.6</td>
<td>&gt;3</td>
</tr>
<tr>
<td>Cadena and Kosecka (2013)</td>
<td>87.9</td>
<td>64.1</td>
<td>31.0</td>
<td>77.8</td>
<td>65.2</td>
<td>66.9</td>
<td>1.7</td>
</tr>
<tr>
<td>Multiscale convnet</td>
<td>68.1</td>
<td>51.1</td>
<td>29.9</td>
<td>87.8</td>
<td>59.2</td>
<td>63.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Multiscale+depth convnet</td>
<td>87.3</td>
<td>45.3</td>
<td>35.5</td>
<td>86.1</td>
<td>63.5</td>
<td>64.5</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Architecture for indoor RGB-D Semantic Segmentation

- Similar to outdoors semantic segmentation method
  - Convnet with 4 input channels
  - Vote over superpixels
Scene Parsing/Labeling on RGB+Depth Images

[Ground truths]

[Our results]

[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]
Scene Parsing/Labeling on RGB+Depth Images

Ground truths

Our results

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

(a) Output of the Multiscale convnet trained using depth information - frame by frame

(b) Results smoothed temporally using Couprie et al. (2013a)

[Couprie, Farabet, Najman, LeCun ICLR 2013]
[Couprie, Farabet, Najman, LeCun ICIP 2013]
[Couprie, Farabet, Najman, LeCun submitted to JMLR]
Semantic Segmentation on RGB+D Images and Videos

[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]
model = nn.Sequential()
-- stage 1 : filter bank -> squashing -> L2 pooling -> normalization
model:add(nn.SpatialConvolutionMM(nfeats, nstates[1], filtsiz, filtsiz))
model:add(nn.Tanh())
model:add(nn.SpatialLPPooling(nstates[1],2,poolsiz,poolsiz,poolsiz,poolsiz))
model:add(nn.SpatialSubtractiveNormalization(nstates[1], normkernel))
-- stage 2 : filter bank -> squashing -> L2 pooling -> normalization
model:add(nn.SpatialConvolutionMM(nstates[1],nstates[2],filtsiz,filtsiz))
model:add(nn.Tanh())
model:add(nn.SpatialLPPooling(nstates[2],2,poolsiz,poolsiz,poolsiz,poolsiz))
model:add(nn.SpatialSubtractiveNormalization(nstates[2], normkernel))
-- stage 3 : 2 fully-connected layers
model:add(nn.Reshape(nstates[2]*filtsize*filtsize))
model:add(nn.Linear(nstates[2]*filtsize*filtsize, nstates[3]))
model:add(nn.Tanh())
model:add(nn.Linear(nstates[3], noutputs))

- http://www.torch.ch (Torch7: Lua-based dev environment for ML, CV....)
- http://eblearn.sf.net (C++ Library with convnet support by P. Sermanet)
Use ReLU non-linearities (tanh and logistic are falling out of favor)
Use cross-entropy loss for classification
Use Stochastic Gradient Descent on minibatches
Shuffle the training samples
Normalize the input variables (zero mean, unit variance)
Schedule to decrease the learning rate
Use a bit of L1 or L2 regularization on the weights (or a combination)
  But it's best to turn it on after a couple of epochs
Use “dropout” for regularization
Lots more in [LeCun et al. “Efficient Backprop” 1998]
Lots, lots more in “Neural Networks, Tricks of the Trade” (2012 edition)