Can Robots Learn to See?

Yann LeCun
Silver Professor of Computer Science and Neural Science
The Courant Institute of Mathematical Sciences
And Center for Neural Science
New York University
How do we learn vision and perception?
- From the image of an airplane, how do we extract a representation that is invariant to pose, illumination, background, clutter, object instance....
- How can a human (or a machine) learn those representations by just looking at the world?

How can we learn visual categories from just a few examples?
- I don't need to see many airplanes before I can recognize every airplane (even really weird ones)
Vision occupies a big chunk of our brains

1/3 of the macaque brain

[from Van Essen]
Vision is very fast and the visual cortex is hierarchical.

The ventral (recognition) pathway in the visual cortex

[picture from Simon Thorpe]
The Primate's Visual System is Deep (LGN->V1->V2->V4->IT)

- The recognition of everyday objects is a very fast process.
  - The recognition of common objects is essentially “feed forward.”
  - But not all of vision is feed forward.

- Much of the visual system (all of it?) is the result of learning
  - How much prior structure is there?

- If the visual system is deep (around 10 layers) and learned

- What is the learning algorithm of the visual cortex?
  - What learning algorithm can train neural nets as “deep” as the visual system (10 layers?).
  - Unsupervised vs Supervised learning
  - What is the loss function?
  - What is the organizing principle?
  - Broader question (Hinton): what is the learning algorithm of the neo-cortex?
The Broader Challenge of Machine Learning and AI

- Can we devise learning algorithms to train a “deep” artificial visual system, and other artificial perception systems.

- **How can we learn the structure of the world?**
  - How can we build/learn internal representations of the world that allow us to discover its hidden structure?
  - How can we learn internal representations that capture the relevant information and eliminates irrelevant variabilities?

- How can a human or a machine learn internal representations by just looking at the world?

- Can we find learning methods that solve really complex problems end-to-end, such as vision, natural language, speech....?
The Traditional “Shallow” Architecture for Recognition

The raw input is pre-processed through a hand-crafted feature extractor

The features are not learned

The trainable classifier is often generic (task independent), and “simple” (linear classifier, kernel machine, nearest neighbor,.....)

The most common Machine Learning architecture: the Kernel Machine
“Modern” Object Recognition Architecture in Computer Vision

- Filter Bank
- Non-Linearity
- Spatial Pooling
- Classifier

Oriented Edges
Gabor Wavelets
Other Filters...
Sigmoid
Rectification
Vector Quant.
Contrast Norm.
Averaging
Max pooling
VQ+Histogram
Geometric Blurr

Example:
- Edges + Rectification + Histograms + SVM [Dalal & Triggs 2005]
- SIFT + classification

Fixed Features + “shallow” classifier
Feature Extraction by Filtering and Pooling

- Filter Bank
- Non-Linearity
- Spatial Pooling

Biologically-inspired models of low-level feature extraction

Inspired by [Hubel and Wiesel 1962]
“State of the Art” architecture for object recognition

Example:
- SIFT features with Spatial Pyramid Match Kernel SVM [Lazebnik et al. 2006]
- Fixed Features + unsupervised features + “shallow” classifier
Good Representations are Hierarchical

In Language: hierarchy in syntax and semantics
- Words -> Parts of Speech -> Sentences -> Text
- Objects, Actions, Attributes... -> Phrases -> Statements -> Stories

In Vision: part-whole hierarchy
- Pixels -> Edges -> Textons -> Parts -> Objects -> Scenes
"Deep" Learning: Learning Hierarchical Representations

Deep Learning: learning a hierarchy of internal representations

From low-level features to mid-level invariant representations, to object identities

Representations are increasingly invariant as we go up the layers

using multiple stages gets around the specificity/invariance dilemma
Do we really need deep architectures?

We can approximate any function as close as we want with shallow architecture (e.g. a kernel machine). Why would we need deep ones?

\[ y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \]

kernel machines and 2-layer neural net are “universal”.

Deep learning machines

\[ y = F(W^1 . F(W^0 . X)) \]

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.
Why are Deep Architectures 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).
- Depth-Breadth tradoff

Example 1: N-bit parity

- requires N-1 XOR gates in a tree of depth log(N).
- 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).....
Deep Supervised Learning is Hard

- The loss surface is non-convex, ill-conditioned, has saddle points, has flat spots.....

- For large networks, it will be horrible! (not really, actually)

- Back-prop doesn't work well with networks that are tall and skinny.
  - Lots of layers with few hidden units.

- Back-prop works fine with short and fat networks
  - But over-parameterization becomes a problem without regularization
  - Short and fat nets with fixed first layers aren't very different from SVMs.

- For reasons that are not well understood theoretically, back-prop works well when they are highly structured
  - e.g. convolutional networks.
Can't we train multi-stage vision architectures?

Stacking multiple stages of feature extraction/pooling.

Creates a hierarchy of features.
Hierarchical/multilayer: features get progressively more global, invariant, and numerous

Dense features: features detectors applied everywhere (no interest point)

Broadly tuned (possibly invariant) features: sigmoid units are on half the time.

Global discriminative training: The whole system is trained “end-to-end” with a gradient-based method to minimize a global loss function

 Integrates segmentation, feature extraction, and invariant classification in one fell swoop.

Yann LeCun
[Hubel & Wiesel 1962]:

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

"Simple cells"  
"Complex cells"  

pooling subsampling  

Multiple convolutions  

Retinotopic Feature Maps
Building a complete artificial vision system:
- Stack multiple stages of simple cells / complex cells layers
- Higher stages compute more global, more invariant features
- Stick a classification layer on top

[Fukushima 1971-1982]
- neocognitron

[LeCun 1988-2007]
- convolutional net

[Poggio 2002-2006]
- HMAX

[Ullman 2002-2006]
- fragment hierarchy

[Lowe 2006]
- HMAX

QUESTION: How do we find (or learn) the filters?
Convolutional Net Architecture

Convolutional net for handwriting recognition  (400,000 synapses)

Convolutional layers  (simple cells): all units in a feature plane share the same weights

Pooling/subsampling layers (complex cells): for invariance to small distortions.

Supervised gradient-descent learning using back-propagation

The entire network is trained end-to-end.  All the layers are trained simultaneously.
**Face Detection and Pose Estimation with Convolutional Nets**

**Training:** 52,850, 32x32 grey-level images of faces, 52,850 non-faces.

**Each sample:** used 5 times with random variation in scale, in-plane rotation, brightness and contrast.

**2nd phase:** half of the initial negative set was replaced by false positives of the initial version of the detector.

---

![Convolutional Network Diagram](image-url)
## Face Detection: Results

<table>
<thead>
<tr>
<th>Data Set</th>
<th>TILTED</th>
<th>PROFILE</th>
<th>MIT+CMU</th>
</tr>
</thead>
<tbody>
<tr>
<td>False positives per image</td>
<td>4.42</td>
<td>0.47</td>
<td>0.5</td>
</tr>
<tr>
<td>Our Detector</td>
<td>90%</td>
<td>67%</td>
<td>83%</td>
</tr>
<tr>
<td></td>
<td>97%</td>
<td>83%</td>
<td>88%</td>
</tr>
<tr>
<td>Jones &amp; Viola (tilted)</td>
<td>90%</td>
<td>95%</td>
<td>x</td>
</tr>
<tr>
<td>Jones &amp; Viola (profile)</td>
<td>x</td>
<td>70%</td>
<td>x</td>
</tr>
</tbody>
</table>

MIT+CMU PROFILE TILTED

**False positives per image:**

Mit's detector was the best on the profile data, with 0.5 false positives per image. Our detector was the best on the tilted data, with 0.47 false positives per image.
Face Detection and Pose Estimation: Results
Face Detection with a Convolutional Net
Demo produced with EBLearn open source package

http://eblearn.sf.net
50 toys belonging to 5 categories: animal, human figure, airplane, truck, car

10 instance per category: 5 instances used for training, 5 instances for testing

Raw dataset: 972 stereo pair of each object instance. 48,600 image pairs total.

For each instance:

18 azimuths
- 0 to 350 degrees every 20 degrees

9 elevations
- 30 to 70 degrees from horizontal every 5 degrees

6 illuminations
- On/off combinations of 4 lights

2 cameras (stereo)
- 7.5 cm apart
- 40 cm from the object

Training instances

Test instances
90,857 free parameters, 3,901,162 connections.

The architecture alternates convolutional layers (feature detectors) and subsampling layers (local feature pooling for invariance to small distortions).

The entire network is trained end-to-end (all the layers are trained simultaneously).

A gradient-based algorithm is used to minimize a supervised loss function.
Normalized-Uniform Set: Error Rates

- Linear Classifier on raw stereo images: 30.2% error.
- K-Nearest-Neighbors on raw stereo images: 18.4% error.
- K-Nearest-Neighbors on PCA-95: 16.6% error.
- Pairwise SVM on 96x96 stereo images: 11.6% error.
- Pairwise SVM on 95 Principal Components: 13.3% error.
- Convolutional Net on 96x96 stereo images: 5.8% error.

Training instances  Test instances
**Jittered-Cluttered Dataset:**

- **291,600** stereo pairs for training, **58,320** for testing
- Objects are jittered: position, scale, in-plane rotation, contrast, brightness, backgrounds, distractor objects,...
- Input dimension: 98x98x2 (approx 18,000)
Experiment 2: Jittered-Cluttered Dataset

291,600 training samples, 58,320 test samples

- SVM with Gaussian kernel: 43.3% error
- Convolutional Net with binocular input: 7.8% error
- Convolutional Net + SVM on top: 5.9% error
- Convolutional Net with monocular input: 20.8% error
- Smaller mono net (DEMO): 26.0% error

Dataset available from http://www.cs.nyu.edu/~yann
Examples (Monocular Mode)
Examples (Monocular Mode)

Zoom = 1.0, Threshold = -1.2, filter on

- car [-0.3]
- plane [2.2]
- animal [3.0]
Mobile robot with two cameras

The convolutional net is trained to emulate a human driver from recorded sequences of video + human-provided steering angles.

The network maps stereo images to steering angles for obstacle avoidance

[LeCun et al. NIPS 2005]
Brain tissue reconstruction from slice images [Jain,....,Denk, Seung 2007]

- Sebastian Seung's lab at MIT.
- 3D convolutional net for image segmentation
- ConvNets Outperform MRF, Conditional Random Fields, Mean Shift, Diffusion,...[ICCV'07]
Industrial Applications of ConvNets

- **AT&T/Lucent/NCR**
  - Check reading, OCR, handwriting recognition (deployed 1996)

- **Vidient Inc**
  - Vidient Inc's "SmartCatch" system deployed in several airports and facilities around the US for detecting intrusions, tailgating, and abandoned objects (Vidient is a spin-off of NEC)

- **NEC Labs**
  - Cancer cell detection, automotive applications, kiosks

- **Google**
  - OCR, face and license plate removal from StreetView

- **Microsoft**
  - OCR, handwriting recognition, speech detection

- **France Telecom**
  - Face detection, HCI, cell phone-based applications

- **Other projects: HRL (3D vision)....**
FPGA Custom Board: NYU ConvNet Processor

- Xilinx Virtex 4 FPGA, 8x5 cm board
  - Dual camera port, Fast dual QDR RAM,
- New version being developed with Eugenio Culurciello (Yale EE)
  - Full custom chip
  - Version for Virtex 6 FPGA

[Farabet et al. 2009]
Reconfigurable Dataflow Architecture

[Farabet et al. 2010]
FPGA Performance

Seconds per frame for a robot vision task (log scale)  [Farabet et al. 2010]

- X86 Core2 Duo: 3s
- Nvidia 9400M GPU: 25ms
- Virtex 4 custom board: 6ms
- Nvidia Tesla C1060: 6ms
- Virtex 6 dev board:
Problem: supervised ConvNets don't work with few labeled samples

On recognition tasks with few labeled samples, deep supervised architectures don't do so well

Example: Caltech-101 Object Recognition Dataset
- 101 categories of objects (gathered from the web)
- Only 30 training samples per category!

Recognition rates (OUCH!):
- Supervised ConvNet: 29.0%
- SIFT features + Pyramid Match Kernel SVM: 64.6%
  
  [Lazebnik et al. 2006]

When learning the features, there are simply too many parameters to learn in purely supervised mode (or so we thought).
Unlabeled data is usually available in large quantity

A lot can be learned about the world by just looking at it

Unsupervised learning captures underlying regularities about the data

The best way to capture underlying regularities is to learn good representations of the data

The main idea of Unsupervised Deep Learning

- Learn each layer one at a time in unsupervised mode
- Stick a supervised classifier on top
- Optionally: refine the entire system in supervised mode

Unsupervised Learning view as Energy-Based Learning
Unsupervised Feature Learning with Sparse Coding

[Olshausen & field 1997]

Find a dictionary of basis functions such that any input can be reconstructed of a sparse linear combination of them.

\[ E(Y^i, Z; W_d) = \|Y^i - W_dZ\|^2 + \lambda \sum_j |z_j| \]

**Energy:**

**Optimal Code**

\[ Z^i = \text{argmin}_z E(Y^i, z; W_d) \]

**Free Energy:**

\[ F(Y^i; W_d) = F(Z^i) = \text{min}_z E(Y^i, z; W_d) \]
Unsupervised Feature Learning with Sparse Coding

The learning algorithm minimizes the loss function:

$$L(W_d) = \sum_i F(Y^i; W_d) = \sum_i (\text{min}_z E(Y^i, Z; W_d))$$

The columns of $W_d$ are normalized.

Energy:

$$E(Y^i, Z; W_d) = \|Y^i - W_d Z\|^2 + \lambda \sum_j |z_j|$$

Free Energy:

$$F(Y^i; W_d) = F(Z^i) = \text{min}_z E(Y^i, z; W_d)$$
Problem with Sparse Coding: Inference is slow

Inference: find $Z$ that minimizes the energy for a given $Y$

$$E(Y^i, Z^i; W_d) = \|Y^i - W_d Z^i\|^2 + \lambda \sum_j |z^i_j|$$

$$Z^i = \text{argmin}_z E(Y^i, z; W_d)$$

- For each new $Y$, an optimization algorithm must be run to find the corresponding optimal $Z$
- This would be very slow for large scale vision tasks
- Also, the optimal $Z$ are very unstable:
  - A small change in $Y$ can cause a large change in the optimal $Z$
Solution: Predictive Sparse Decomposition (PSD)

Prediction the optimal code with a trained encoder

Energy = reconstruction_error + code_prediction_error + code_sparsity

\[ E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \|Z - g_e(W_e, Y^i)\|^2 + \lambda \sum_j |z_j| \]

\[ g_e(W_e, Y^i) = D \tanh(W_e Y) \]
Inference by gradient descent starting from the encoder output

\[ E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \|Z - g_e(W_e, Y^i)\|^2 + \lambda \sum_j |z_j| \]

\[ Z^i = \text{argmin}_z E(Y^i, z; W) \]
Learning by minimizing the average energy of the training data with respect to \( W_d \) and \( W_e \).

**Loss function:**

\[
L(W_d, W_e) = \sum_i F(Y^i; W_d, W_e) \\
F(Y^i; W_d, W_e) = \min_z E(Y^i, z; W_d, W_e)
\]
1. Initialize $Z = \text{Encoder}(Y)$
2. Find $Z$ that minimizes the energy function
3. Update the Decoder basis functions to reduce reconstruction error
4. Update Encoder parameters to reduce prediction error

Repeat with next training sample
PSD trained on handwritten digits: decoder filters are “parts” (strokes).

- Any digit can be reconstructed as a linear combination of a small number of these “parts”.
Basis functions are like Gabor filters (like receptive fields in V1 neurons)

- 256 filters of size 12x12
- Trained on natural image patches from the Berkeley dataset
- Encoder is linear-tanh-diagonal
Learned Features on natural patches: V1-like receptive fields
Learned Features: V1-like receptive fields

- 12x12 filters
- 1024 filters
Classification Error Rate on MNIST

- Supervised Linear Classifier trained on 200 trained sparse features
  - Red: linear-tanh-diagonal encoder; Blue: linear encoder

![Graphs showing error rate vs. RMSE for different sample sizes: 10, 100, and 1000 samples.](image)
Using PSD to Train a Hierarchy of Features

**Phase 1: train first layer using PSD**

\[
\|Y^i - \hat{Y}\|^2 \quad W_d Z \\
\|Z - \hat{Z}\|^2 \\
g_e(W_e, Y^i) \\
Y \\
\lambda \sum_j |z_j|
\]
Using PSD to Train a Hierarchy of Features

- **Phase 1:** train first layer using PSD
- **Phase 2:** use encoder + absolute value as feature extractor

![Diagram](attachment:image.png)
Using PSD to Train a Hierarchy of Features

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD

\[ g_e(W_e, Y^i) \]

\[ |z_j| \rightarrow Y \]

\[ W_d Z \]

\[ \|y^i - \hat{y}\|^2 \]

\[ g_e(W_e, Y^i) \]

\[ |z_j| \rightarrow Z \]

\[ \|z - \hat{z}\|^2 \]

\[ \lambda \sum_j \]

FEATURES
Using PSD to Train a Hierarchy of Features

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2\textsuperscript{nd} feature extractor

\[
g_e(W_e, Y^i) \rightarrow |z_j| \rightarrow g_e(W_e, Y^i) \rightarrow |z_j|
\]
Using PSD to Train a Hierarchy of Features

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2\textsuperscript{nd} feature extractor
- Phase 5: train a supervised classifier on top
- Phase 6 (optional): train the entire system with supervised back-propagation
The “deep learning” method was popularized by Hinton for training “deep belief networks”.

- DBN use a special kind of encoder-decoder architecture called Restricted Boltzmann Machines (RBM)

1. Train each layer in an unsupervised fashion, layer by layer

2. Stick a supervised classifier on top, and refine the entire system with gradient descent (back-prop) on a supervised criterion.
Unsupervised Learning: Capturing Dependencies Between Variables

**Energy function:** viewed as a negative log probability density

**Probabilistic View:**
- Produce a probability density function that:
  - has high value in regions of high sample density
  - has low value everywhere else (integral = 1).

**Energy-Based View:**
- produce an energy function $E(Y, W)$ that:
  - has low value in regions of high sample density
  - has high(er) value everywhere else
Energy function viewed as a negative log density

Example: \( y = x^2 \)
Energy <-> Probability

\[ P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}} \]

\[ E(Y, W) \propto -\log P(Y|W) \]
Training an Energy-Based Model

- Make the energy around training samples low
- Make the energy everywhere else higher

\[ P(Y, W) = \frac{e^{-\beta E(Y, W)}}{\int_y e^{-\beta E(y, W)}} \]
Training an Energy-Based Model to Approximate a Density

Maximizing $P(Y|W)$ on training samples

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

Minimizing $-\log P(Y,W)$ on training samples

$$L(Y, W) = E(Y, W) + \frac{1}{\beta} \log \int_y e^{-\beta E(y,W)}$$
Training an Energy-Based Model with Gradient Descent

Gradient of the negative log-likelihood loss for one sample $Y$:

$$ \frac{\partial L(Y, W)}{\partial W} = \frac{\partial E(Y, W)}{\partial W} - \int_y P(y|W) \frac{\partial E(y, W)}{\partial W} $$

Gradient descent:

$$ W \leftarrow W - \eta \frac{\partial L(Y, W)}{\partial W} $$

- Pushes down on the energy of the samples
- Pulls up on the energy of low-energy $Y$'s

$$ W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \int_y P(y|W) \frac{\partial E(y, W)}{\partial W} $$
How do we push up on the energy of everything else?

**Solution 1: contrastive divergence [Hinton 2000]**
- Move away from a training sample a bit
- Push up on that

**Solution 2: score matching**
- On the training samples: minimize the gradient of the energy, and maximize the trace of its Hessian.

**Solution 3: denoising auto-encoder (not really energy-based)**
- Train the inference dynamics to map noisy samples to clean samples

**Solution 4: MAIN INSIGHT! [Ranzato, ..., LeCun AI-Stat 2007]**
- Restrict the information content of the code (features) $Z$
- If the code $Z$ can only take a few different configurations, only a correspondingly small number of Ys can be perfectly reconstructed
- Idea: impose a sparsity prior on $Z$
- This is reminiscent of sparse coding [Olshausen & Field 1997]
Prediction the optimal code with a trained encoder

Energy = reconstruction_error + code_prediction_error + code_sparsity

\[ E(Y^i, Z) = \| Y^i - W_d Z \|^2 + \| Z - g_e(W_e, Y^i) \|^2 + \lambda \sum_j |z_j| \]

\[ g_e(W_e, Y^i) = D \tanh(W_e Y) \]
The Main Insight [Ranzato et al. AISTATS 2007]

- If the information content of the feature vector is limited (e.g. by imposing sparsity constraints), the energy MUST be large in most of the space.
  - pulling down on the energy of the training samples will necessarily make a groove

- The volume of the space over which the energy is low is limited by the entropy of the feature vector
  - Input vectors are reconstructed from feature vectors.
  - If few feature configurations are possible, few input vectors can be reconstructed properly
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

*Training based on minimizing the reconstruction error over the training set*
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

BAD: machine does not learn structure from training data!!

It just copies the data.
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

IDEA: reduce number of available codes.
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

*IDEA: reduce number of available codes.*
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

**IDEA:** reduce number of available codes.
We are going to impose a sparsity penalty on the code to restrict its information content.

We will allow the code to have higher dimension than the input.

Categories are more easily separable in high-dim sparse feature spaces.
- This is a trick that SVM use: they have one dimension per sample.

Sparse features are optimal when an active feature costs more than an inactive one (zero).
- e.g. neurons that spike consume more energy.
- The brain is about 2% active on average.
- 2 dimensional toy dataset
- Mixture of 3 Cauchy distrib.
- Visualizing energy surface (black = low, white = high)

[Ranzato's PhD thesis 2009]

<table>
<thead>
<tr>
<th>Method</th>
<th>Encoder</th>
<th>Decoder</th>
<th>Energy</th>
<th>Loss</th>
<th>Pull-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>$W Y$</td>
<td>$W Z$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>dimens.</td>
</tr>
<tr>
<td>autoencoder</td>
<td>$\sigma(W e Y)$</td>
<td>$W_d Z$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y) + \log \Gamma$</td>
<td>part. func.</td>
</tr>
<tr>
<td>sparse coding</td>
<td>--</td>
<td>--</td>
<td>$|Y - WZ|^2 + \lambda</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td>--</td>
<td>--</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>1-of-N code</td>
</tr>
</tbody>
</table>
2 dimensional toy dataset
- spiral

Visualizing energy surface (black = low, white = high)

<table>
<thead>
<tr>
<th>Method</th>
<th>Encoder</th>
<th>Decoder</th>
<th>Energy</th>
<th>Loss</th>
<th>Pull-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>$W^1_Y$</td>
<td>$WZ$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>dimens.</td>
</tr>
<tr>
<td>autoencoder</td>
<td>$\sigma(W^e_Y)$</td>
<td>$W^d_Z$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>dimens.</td>
</tr>
<tr>
<td>sparse coding</td>
<td>$\sigma(W^e_Z)$</td>
<td>$W^d_Z$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>sparsity</td>
</tr>
<tr>
<td>K-Means</td>
<td>-</td>
<td>$WZ$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>1-of-N code</td>
</tr>
</tbody>
</table>
Learning the filters of a ConvNet-like architecture with PSD

1. Train filters on images patches with PSD

2. Plug the filters into a ConvNet architecture

3. Train a supervised classifier on top
“Modern” Object Recognition Architecture in Computer Vision

Example:
- Edges + Rectification + Histograms + SVM [Dalal & Triggs 2005]
- SIFT + classification

Fixed Features + “shallow” classifier
“State of the Art” architecture for object recognition

Example:
- SIFT features with Spatial Pyramid Match Kernel SVM [Lazebnik et al. 2006]
- Fixed Features + unsupervised features + “shallow” classifier
Can't we get the same results with (deep) learning?

Stacking multiple stages of feature extraction/pooling.

Creates a hierarchy of features

ConvNets and SIFT+PMK-SVM architectures are conceptually similar

Can deep learning make a ConvNet match the performance of SIFT+PNK-SVM?
How well do PSD features work on Caltech-101?

Recognition Architecture

Filter Bank -> Non-Linearity -> Spatial Pooling -> Classifier

abs -> SVM
Procedure for a single-stage system

1. Pre-process images
   - remove mean, high-pass filter, normalize contrast

2. Train encoder-decoder on 9x9 image patches

3. use the filters in a recognition architecture
   - Apply the filters to the whole image
   - Apply the tanh and D scaling
   - Add more non-linearities (rectification, normalization)
   - Add a spatial pooling layer

4. Train a supervised classifier on top
   - Multinomial Logistic Regression or Pyramid Match Kernel SVM
64 filters on 9x9 patches trained with PSD

with Linear-Sigmoid-Diagonal Encoder

weights \( \pm 0.2828 - 0.3043 \)
Feature Extraction

Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
Feature Extraction

Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?

```
\[ x - \mu \over \max(t, \sigma) \]
```

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C**: Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs**: Rectification layer: needed?
- **N**: Normalization layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?

Diagram:

1. **C** - Convolution/sigmoid layer
2. **Abs** - Rectification layer
3. **N** - Normalization layer
4. **Pooling Down-Sampling Layer**
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?
- **P** Pooling down-sampling layer: average or max?
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?
- **P** Pooling down-sampling layer: average or max?
Feature Extraction

- C Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- Abs Rectification layer: needed?
- N Normalization layer: needed?
- P Pooling down-sampling layer: average or max?

THIS IS ONE STAGE OF FEATURE EXTRACTION
Training Protocol

Training

- Logistic Regression on Random Features: \( R \)
- Logistic Regression on PSD features: \( U \)
- Refinement of whole net from random with backprop: \( R^+ \)
- Refinement of whole net starting from PSD filters: \( U^+ \)

Classifier

- Multinomial Logistic Regression or Pyramid Match Kernel SVM
Using PSD Features for Recognition

\[
\left[ 64.F_{\text{CSG}}^{9 \times 9} - R/N/P^{5 \times 5} \right] - \text{log\_reg}
\]

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>( R_{\text{abs}} - N - P_A )</th>
<th>( R_{\text{abs}} - P_A )</th>
<th>( N - P_M )</th>
<th>( N - P_A )</th>
<th>( P_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U^+ )</td>
<td>54.2%</td>
<td>50.0%</td>
<td>44.3%</td>
<td>18.5%</td>
<td>14.5%</td>
</tr>
<tr>
<td>( R^+ )</td>
<td>54.8%</td>
<td>47.0%</td>
<td>38.0%</td>
<td>16.3%</td>
<td>14.3%</td>
</tr>
<tr>
<td>( U )</td>
<td>52.2%</td>
<td>43.3(\pm1.6)%</td>
<td>44.0%</td>
<td>17.2%</td>
<td>13.4%</td>
</tr>
<tr>
<td>( R )</td>
<td>53.3%</td>
<td>31.7%</td>
<td>32.1%</td>
<td>15.3%</td>
<td>12.1(\pm2.2)%</td>
</tr>
</tbody>
</table>

\[
\left[ 64.F_{\text{CSG}}^{9 \times 9} - R/N/P^{5 \times 5} \right] - \text{PMK}
\]

| \( U \) | 65.0\% |

\[
\left[ 96.F_{\text{CSG}}^{9 \times 9} - R/N/P^{5 \times 5} \right] - \text{PCA - lin\_svm}
\]

| \( U \) | 58.0\% |

96.Gabors - PCA - lin\_svm (Pinto and DiCarlo 2006)

| Gabors | 59.0\% |

SIFT - PMK (Lazebnik et al. CVPR 2006)

| Gabors | 64.6\% |
Rectification makes a huge difference:
- 14.5% -> 50.0%, without normalization
- 44.3% -> 54.2% with normalization

Normalization makes a difference:
- 50.0 → 54.2

Unsupervised pretraining makes small difference

PSD works just as well as SIFT

Random filters work as well as anything!
- If rectification/normalization is present

PMK_SVM classifier works a lot better than multinomial log_reg on low-level features
- 52.2% → 65.0%
Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!

PSD features are more stable.

Feature Sign (FS) is an optimization methods for computing sparse codes [Lee...Ng 2006]
PSD Features are more stable

Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!

Because PSD features are more stable. Feature obtained through sparse optimization can change a lot with small changes of the input.

How many features change sign in patches from successive video frames (a,b), versus patches from random frame pairs (c)
PSD features are much cheaper to compute

Computing PSD features is hundreds of times cheaper than Feature Sign.
How Many 9x9 PSD features do we need?

Accuracy increases slowly past 64 filters.
1. Train stage-1 filters with PSD on patches from natural images
2. Compute stage-1 features on training set
3. Train state-2 filters with PSD on stage-1 feature patches
4. Compute stage-2 features on training set
5. Train linear classifier on stage-2 features
6. Refine entire network with supervised gradient descent

What are the effects of the non-linearities and unsupervised pretraining?
Multistage Hubel-Wiesel Architecture on Caltech-101
Multistage Hubel-Wiesel Architecture

**Image Preprocessing:**
- High-pass filter, local contrast normalization (divisive)

**First Stage:**
- Filters: 64 9x9 kernels producing 64 feature maps
- Pooling: 10x10 averaging with 5x5 subsampling

**Second Stage:**
- Filters: 4096 9x9 kernels producing 256 feature maps
- Pooling: 6x6 averaging with 3x3 subsampling
- Features: 256 feature maps of size 4x4 (4096 features)

**Classifier Stage:**
- Multinomial logistic regression

**Number of parameters:**
- Roughly 750,000
### Multistage Hubel-Wiesel Architecture on Caltech-101

#### Single Stage System:

\[ 64.F_{CSG}^{9 \times 9} - R/N/P^{5 \times 5} \] - log_reg

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>( R_{abs} - N - P_A )</th>
<th>( R_{abs} - P_A )</th>
<th>( N - P_M )</th>
<th>( N - P_A )</th>
<th>( P_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U^+ )</td>
<td>54.2%</td>
<td>50.0%</td>
<td>44.3%</td>
<td>18.5%</td>
<td>14.5%</td>
</tr>
<tr>
<td>( R^+ )</td>
<td>54.8%</td>
<td>47.0%</td>
<td>38.0%</td>
<td>16.3%</td>
<td>14.3%</td>
</tr>
<tr>
<td>( U )</td>
<td>52.2%</td>
<td>43.3% (±1.6)</td>
<td>44.0%</td>
<td>17.2%</td>
<td>13.4%</td>
</tr>
<tr>
<td>( R )</td>
<td>53.3%</td>
<td>31.7%</td>
<td>32.1%</td>
<td>15.3%</td>
<td>12.1% (±2.2)</td>
</tr>
<tr>
<td>( G )</td>
<td>52.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Two Stage System:

\[ 64.F_{CSG}^{9 \times 9} - R/N/P^{5 \times 5} \] – \[ 256.F_{CSG}^{9 \times 9} - R/N/P^{4 \times 4} \] - log_reg

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>( R_{abs} - N - P_A )</th>
<th>( R_{abs} - P_A )</th>
<th>( N - P_M )</th>
<th>( N - P_A )</th>
<th>( P_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U^+U^+ )</td>
<td>65.5%</td>
<td>60.5%</td>
<td>61.0%</td>
<td>34.0%</td>
<td>32.0%</td>
</tr>
<tr>
<td>( R^+R^+ )</td>
<td>64.7%</td>
<td>59.5%</td>
<td>60.0%</td>
<td>31.0%</td>
<td>29.7%</td>
</tr>
<tr>
<td>( UU )</td>
<td>63.7%</td>
<td>46.7%</td>
<td>56.0%</td>
<td>23.1%</td>
<td>9.1%</td>
</tr>
<tr>
<td>( RR )</td>
<td>62.9%</td>
<td>33.7% (±1.5)</td>
<td>37.6% (±1.9)</td>
<td>19.6%</td>
<td>8.8%</td>
</tr>
<tr>
<td>( GT )</td>
<td>55.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

← like HMAX model

#### Single Stage:

\[ 64.F_{CSG}^{9 \times 9} - R/N/P^{5 \times 5} \] - PMK-SVM

| U | 64.0% |

#### Two Stages:

\[ 64.F_{CSG}^{9 \times 9} - R/N/P^{5 \times 5} \] – \[ 256.F_{CSG}^{9 \times 9} - R/N \] - PMK-SVM

| UU | 52.8% |
Two-Stage Result Analysis

- Second Stage + logistic regression = PMK_SVM
- Unsupervised pre-training doesn't help much :-(
- Random filters work amazingly well with normalization
- Supervised global refinement helps a bit
- The best system is really cheap
- Either use rectification and average pooling or no rectification and max pooling.
Multistage Hubel-Wiesel Architecture: Filters

Stage 1

After PSD

weights : -0.2232 - 0.2075

weights : -0.0778 - 0.064

Stage 2

After supervised refinement

weights : -0.2828 - 0.3043

weights : -0.0929 - 0.0784
Demo: real-time learning of visual categories

Input: 83x83 (raw: 91x91)

Contrast-normalized high-pass filtered

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

Yann LeCun

New York University
MNIST dataset

- 10 classes and up to 60,000 training samples per class
**Architecture**

- $U^+U^+$: 0.53% error (this is a record on the undistorted MNIST!)

**Comparison:** $RR$ versus $UU$ and $R^+R^+$

**Diagram:**

Classification error on the MNIST dataset

- **Orange line with crosses:** Supervised training of the whole network
- **Blue line with diamonds:** Unsupervised training of the feature extractors
- **Dashed line with plus signs:** Random feature extractors

Size of labelled training set

% Classification error
Why Random Filters Work?
## The Competition: SIFT + Sparse-Coding + PMK-SVM

### Replacing K-means with Sparse Coding

- [Yang 2008] [Boureau, Bach, Ponce, LeCun 2010]

<table>
<thead>
<tr>
<th>Method</th>
<th>Method</th>
<th>Caltech 15</th>
<th>Caltech 30</th>
<th>Scenes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Boiman et al. [1]] Nearest neighbor + spatial correspondence</td>
<td>65.00 ± 1.14</td>
<td>70.40</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>[Jain et al. [8]] Fast image search for learned metrics</td>
<td>61.00</td>
<td>69.60</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>[Lazebnik et al. [12]] Spatial Pyramid + hard quantization + kernel SVM</td>
<td>56.40</td>
<td>64.40 ± 0.80</td>
<td>81.40 ± 0.50</td>
<td></td>
</tr>
<tr>
<td>[van Gemert et al. [24]] Spatial Pyramid + soft quantization + kernel SVM</td>
<td>-</td>
<td>64.14 ± 1.18</td>
<td>76.67 ± 0.39</td>
<td></td>
</tr>
<tr>
<td>[Yang et al. [26]] SP + sparse codes + max pooling + linear</td>
<td>67.00 ± 0.45</td>
<td>73.2 ± 0.54</td>
<td>80.28 ± 0.93</td>
<td></td>
</tr>
<tr>
<td>[Zhang et al. [27]] $k$NN-SVM</td>
<td>59.10 ± 0.60</td>
<td>66.20 ± 0.50</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>[Zhou et al. [29]] SP + Gaussian mixture</td>
<td>-</td>
<td>-</td>
<td>84.1 ± 0.5</td>
<td></td>
</tr>
<tr>
<td>Baseline:</td>
<td>SP + hard quantization + avg pool + kernel SVM</td>
<td>56.74 ± 1.31</td>
<td>64.19 ± 0.94</td>
<td>80.89 ± 0.21</td>
</tr>
<tr>
<td>Unsupervised coding</td>
<td>SP + soft quantization + avg pool + kernel SVM</td>
<td>59.12 ± 1.51</td>
<td>66.42 ± 1.26</td>
<td>81.52 ± 0.54</td>
</tr>
<tr>
<td>1x1 features</td>
<td>SP + soft quantization + max pool + kernel SVM</td>
<td>63.61 ± 0.88</td>
<td>-</td>
<td>83.41 ± 0.57</td>
</tr>
<tr>
<td>8 pixel grid resolution</td>
<td>SP + sparse codes + avg pool + kernel SVM</td>
<td>62.85 ± 1.22</td>
<td>70.27 ± 1.29</td>
<td>83.15 ± 0.35</td>
</tr>
<tr>
<td></td>
<td>SP + sparse codes + max pool + kernel SVM</td>
<td>64.62 ± 0.94</td>
<td>71.81 ± 0.96</td>
<td>84.25 ± 0.35</td>
</tr>
<tr>
<td></td>
<td>SP + sparse codes + max pool + linear</td>
<td>64.71 ± 1.05</td>
<td>71.52 ± 1.13</td>
<td>83.78 ± 0.53</td>
</tr>
<tr>
<td>Macrofeatures +</td>
<td>SP + sparse codes + max pool + kernel SVM</td>
<td>69.03 ± 1.17</td>
<td>75.72 ± 1.06</td>
<td>84.60 ± 0.38</td>
</tr>
<tr>
<td>Finer grid resolution</td>
<td>SP + sparse codes + max pool + linear</td>
<td>68.78 ± 1.09</td>
<td>75.14 ± 0.86</td>
<td>84.41 ± 0.26</td>
</tr>
</tbody>
</table>
Small NORB dataset

- 5 classes and up to 24,300 training samples per class
NORB Generic Object Recognition Dataset

- **50** toys belonging to 5 categories: **animal, human figure, airplane, truck, car**
- **10** instance per category: **5 instances used for training, 5 instances for testing**
- **Raw dataset:** **972** stereo pair of each object instance. **48,600** image pairs total.

For each instance:

- **18 azimuths**
  - 0 to 350 degrees every 20 degrees
- **9 elevations**
  - 30 to 70 degrees from horizontal every 5 degrees
- **6 illuminations**
  - On/off combinations of 4 lights
- **2 cameras (stereo)**
  - 7.5 cm apart
  - 40 cm from the object

**Training instances**  **Test instances**
Two-stage system: error rate versus number of labeled training samples
Learning Complex Cells with Invariance Properties

[Kavukcuoglu et al. CVPR 2008]
Unsupervised PSD ignores the spatial pooling step.

Could we devise a similar method that learns the pooling layer as well?

Idea [Hyvarinen & Hoyer 2001]: group sparsity on pools of features

- Minimum number of pools must be non-zero
- Number of features that are on within a pool doesn't matter
- Pools tend to regroup similar features

\[
\|Y^i - \hat{Y}\|^2 \rightarrow W_d Z \rightarrow \lambda \sum_j . \left( \sum_{k \in P_j} Z_k^2 \right)
\]

\[
g_e (W_e, Y^i) \rightarrow \|Z - \hat{Z}\|^2
\]
Learning the filters and the pools

Using an idea from Hyvarinen: topographic square pooling (subspace ICA)

1. Apply filters on a patch (with suitable non-linearity)
2. Arrange filter outputs on a 2D plane
3. Square filter outputs
4. Minimize $\sqrt{\text{sum of blocks of squared filter outputs}}$

Units in the code $Z$

Define pools and enforce sparsity across pools
Learning the filters and the pools

- The filters arrange themselves spontaneously so that similar filters enter the same pool.
- The pooling units can be seen as complex cells.
- They are invariant to local transformations of the input.
  - For some it's translations, for others rotations, or other transformations.
Pinwheels?
Invariance Properties Compared to SIFT

- Measure distance between feature vectors (128 dimensions) of 16x16 patches from natural images
  - Left: normalized distance as a function of translation
  - Right: normalized distance as a function of translation when one patch is rotated 25 degrees.

- Topographic PSD features are more invariant than SIFT
Learning Invariant Features

**Recognition Architecture**

- HPF/LCN -> filters -> tanh -> sqr -> pooling -> sqrt -> Classifier
- Block pooling plays the same role as rectification
## Recognition Accuracy on Caltech 101

- A/B Comparison with SIFT (128x34x34 descriptors)
- 32x16 topographic map with 16x16 filters
- Pooling performed over 6x6 with 2x2 subsampling
- 128 dimensional feature vector per 16x16 patch
- Feature vector computed every 4x4 pixels (128x34x34 feature maps)
- Resulting feature maps are spatially smoothed

<table>
<thead>
<tr>
<th>Method</th>
<th>Av. Accuracy/Class (%)</th>
</tr>
</thead>
</table>
| **local norm**

\[ \text{local norm}_{5 \times 5} + \text{boxcar}_{5 \times 5} + \text{PCA}_{3060} + \text{linear SVM} \]

<table>
<thead>
<tr>
<th>Method</th>
<th>Av. Accuracy/Class (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPSD (24x24)</td>
<td>50.9</td>
</tr>
<tr>
<td>SIFT (24x24) (non rot. inv.)</td>
<td>51.2</td>
</tr>
<tr>
<td>SIFT (24x24) (rot. inv.)</td>
<td>45.2</td>
</tr>
<tr>
<td>Serre et al. features [25]</td>
<td>47.1</td>
</tr>
</tbody>
</table>
| **local norm**

\[ \text{local norm}_{9 \times 9} + \text{Spatial Pyramid Match Kernel SVM} \]

<table>
<thead>
<tr>
<th>Method</th>
<th>Av. Accuracy/Class (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT [11]</td>
<td>64.6</td>
</tr>
<tr>
<td>IPSD (34x34)</td>
<td>59.6</td>
</tr>
<tr>
<td>IPSD (56x56)</td>
<td>62.6</td>
</tr>
<tr>
<td>IPSD (120x120)</td>
<td>65.5</td>
</tr>
</tbody>
</table>
A/B Comparison with SIFT (128x5x5 descriptors)  
32x16 topographic map with 16x16 filters.

### Performance on Tiny Images Dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPSD (5x5)</td>
<td>54</td>
</tr>
<tr>
<td>SIFT (5x5) (non rot. inv.)</td>
<td>53</td>
</tr>
</tbody>
</table>

### Performance on MNIST Dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPSD (5x5)</td>
<td>1.0</td>
</tr>
<tr>
<td>SIFT (5x5) (non rot. inv.)</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Learning fields of Convolutional Filters
Problem:

- With patch-level training, the learning algorithm must reconstruct the entire patch with a single feature vector.
- But when the filters are used convolutionally, neighboring feature vectors will be highly redundant.
Problem with patch-based training: high correlation between outputs of filters from overlapping receptive fields.
Filters and Basis Functions obtained with 1, 2, 4, 8, 16, 32, and 64 filters.
Learning fields of Simple Cells and Complex Cells

[Gregor and LeCun, 2010]
Training Simple Cells with Local Receptive Fields over Large Input Images

Training on 115x115 images. Kernels are 15x15
Simple Cells + Complex Cells with Sparsity Penalty: Pinwheels

Training on 115x115 images. Kernels are 15x15
119x119 Image Input
100x100 Code
20x20 Receptive field size
sigma=5
Same Method, with Training at the Image Level (vs patch)

Color indicates orientation (by fitting Gabors)
Deep Learning for Mobile Robot Vision
Getting a robot to drive autonomously in unknown terrain solely from vision (camera input).

Our team (NYU/Net-Scale Technologies Inc.) was one of 8 participants funded by DARPA.

All teams received identical robots and can only modify the software (not the hardware).

The robot is given the GPS coordinates of a goal, and must drive to the goal as fast as possible. The terrain is unknown in advance. The robot is run 3 times through the same course.

Long-Range Obstacle Detection with online, self-trained ConvNet.

Uses temporal consistency!
Obstacle Detection

Obstacles overlaid with camera image

Camera image
Detected obstacles (red)
Navigating to a goal is hard... especially in a snowstorm. stereo perspective

human perspective

especially in a snowstorm.

Yann LeCun
Self-Supervised Learning

- Stereo vision tells us what nearby obstacles look like
- Use the labels (obstacle/traversable) produced by stereo vision to train a monocular neural network
- Self-supervised “near to far” learning
Long Range Vision: Distance Normalization

Pre-processing (125 ms)

- Ground plane estimation
- Horizon leveling
- Conversion to YUV + local contrast normalization
- Scale invariant pyramid of distance-normalized image “bands”
Convolutional Net Architecture

- Operates on 12x25 YUV windows from the pyramid
- Convolutional Net Architecture
  - 20-36 pixels tall, 36-500 pixels wide
  - YUV image band
  - Convolutions with 7x6 kernels
  - Pooling/subsampling with 1x4 kernels
  - Convolutions with 6x5 kernels
  - Convolutions with 100x1x1 input windows
  - Logistic regression 100 features -> 5 classes
  - 100 features per 3x12x25 input window
Convolutional Net Architecture

YUV input

100@25x121

CONVOLUTIONS (6x5)

20@30x125

MAX SUBSAMPLING (1x4)

20@30x484

CONVOLUTIONS (7x6)

3@36x484
Long Range Vision: 5 categories

Online Learning (52 ms)

- Label windows using stereo information – 5 classes

(super-ground)  (ground)  (footline)  (obstacle)  (super-obstacle)
Trainable Feature Extraction

“Deep belief net” approach to unsupervised feature learning

Two stages are trained in sequence
- each stage has a layer of convolutional filters and a layer of horizontal feature pooling.
- Naturally shift invariant in the horizontal direction

Filters of the convolutional net are trained so that the input can be reconstructed from the features
- 20 filters at the first stage (layers 1 and 2)
- 300 filters at the second stage (layers 3 and 4)

Scale invariance comes from pyramid.
- for near-to-far generalization
Long Range Vision Results

Input image  
Stereo Labels  
Classifier Output  

Input image  
Stereo Labels  
Classifier Output
Long Range Vision Results

Input image

Stereo Labels

Classifier Output

Input image

Stereo Labels

Classifier Output
Long Range Vision Results

Input image
Stereo Labels
Classifier Output

Input image
Stereo Labels
Classifier Output
Feature Learning for traversability prediction (LAGR)

Comparing

- purely supervised
- stacked, invariant auto-encoders
- DrLIM invariant learning

Testing on hand-labeled groundtruth frames – binary labels

Comparison of Feature Extractors on Groundtruth Data

[Bar chart showing error rates for different feature extractors across various datasets such as belvoir, swri, forest trails, dry woods, coastal NJ, open lawn, man-made, and average.]
Collaborators

Current PhD students:
- Y-Lan Boureau, Koray Kavukcuoglu, Pierre Sermanet

Former PhD students:
- Raia Hadsell, Fu-Jie Huang, Marc'Aurelio Ranzato

Postdocs and Research Scientists
- Clément Farabet, Karol Gregor, Marco Scoffier

Senior Collaborators
- Rob Fergus (NYU): invariant feature learning
- Eugenio Culurciello (Yale): FPGA/ASIC design
- Yoshua Bengio (U. Montreal): deep learning
- Leon Bottou (NEC Labs): handwriting recognition
- Jean Ponce (ENS/INRIA), Francis Bach (ENS/INRIA): sparse coding.