



香港中文大學

The Chinese University of Hong Kong

ELEG 5040

Advanced Topics on Signal Processing (Introduction to Deep Learning)

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

Department of Electronic Engineering,
The Chinese University of Hong Kong

Course Information

- Course webpage

Signup link:

piazza.com/cuhk.edu.hk/spring2015/eleg5040

(If you want to take this course, please sign up with the link above)

Class link:

piazza.com/cuhk.edu.hk/spring2015/eleg5040/home

(If you are a listener, you still can access lecture notes from the link above without signup)

Course Information

- Instructor: Xiaogang Wang
 - SHB 415
 - Office hours: after Monday's class or by appointment
- Tutor: Kai Kang
 - SHB 304
 - kkang@ee.cuhk.edu.hk
 - Office hours: after the tutorial or by appointment

Course Information

- Lecture time & venue
 - Monday: 4:30pm – 5:15pm, ERB 404
 - Tuesday: 2:30pm – 4:15pm, ERB Lecture Theatre
- Options of tutorial time
 - Monday 17:30 - 18:15
 - Tuesday 17:30 - 18:15
 - Thursday 15:30 – 16:15
 - Friday 16:30 – 17:15

Course Information

- Homework (15%)
- Midterm (15%)
- Final exam (30%)
- Project (40%)
 - Applications of deep learning
 - Implementation of deep learning
 - Study deep learning algorithms
 - You should submit
 - A term paper of 4 pages (excluding figures) in maximum, double column, font size is equal or larger than 10.
 - Code and sample data
 - Project presentation
 - No survey
 - No collaboration

Course Information

- Examples of project topics
 - Implement CNN with GPU and compare its efficiency with Caffe
 - Modify Caffe to make it support multiple GPUs
 - Fast CPU implementation of CNN
 - We provide a baseline model of GoogLeNet on ImageNet, and you try improve it
 - Choose one of the deep learning related competitions (such as ImageNet), and compare your result with published ones
 - Propose a deep model to effectively learn dynamic features from videos
 - Deep learning for speech recognition
 - Deep learning for object detection
 - Will provide more later ...
 - We you can discuss your topics with me at any time

Lectures

Week	Chapter	Content
1	Introduction to deep learning	Historical review of DL. DL makes difference. Classical deep models. Why DL works?
2	Machine learning basics	Classification, regression, capacity, underfitting, overfitting, generalization error, regularization, curse of dimensionality, bias, variance, supervised learning, unsupervised learning, discriminative model, generative model
3	Multilayer neural network	Feedforward operation, backpropagation, nonlinear activation functions, expressive power of three-layer neural network, nonlinear feature mapping, target functions, gradient descent, feature/representation learning
4	Convolutional neural network (CNN)	CNN, filters, pooling, efficient convolution algorithms, BP for CNN

Lectures

Week	Chapter	Content
5	Deep belief nets	Boltzmann machine (BM), Restricted Boltzmann machine (RBM), contrast divergence, deep Boltzmann machines, deep belief nets, convolutional Boltzmann machines, unsupervised pre-training, layerwise pre-training
6	Auto-encoder	Auto-encoder, denoising auto-encoder, stacked auto-encoder, unsupervised learning, manifold learning.
7	Optimization for training deep models	Stochastic gradient descent, pre-training, dropout, data augmentation, mini-batch, monitor the training process, learning rate, error surfaces, local minimum, normalization, data augmentation, initialization, training very deep networks
8	Midterm	
9	Large scale deep learning	Fast CPU implementation, GPU implementation, asynchronous parallel implementation

Lectures

Week	Chapter	Content
10	Multi-task and transfer deep learning	Multi-task learning, transfer learning, domain adaptation
11	Recurrent neural networks	Recurrent neural networks, Long-Short-Term-Memory network
12-13	Applications of deep learning	Image classification, object detection, , automatic image caption, face recognition, speech recognition
14	Understanding deep learning	Disentangling hidden factors, sparseness, robust to data corruption, distributed representation, contextual modeling, learning effective feature representation from rich predictions, exponential gain in representational efficiency from distributed representations, exponential gain in representation efficiency from depth, visualization of deep models

Tutorials

Week	Content
1	No tutorial (decide time and venue)
2	Python
3	Theano
4-5	CUDA, GPU programming (given by engineer from NVIDIA)
6-7	Deep learning toolbox
8-10	Caffe
11-12	Sharing research experience on deep learning
13-14	Review of lectures

6-hours Tutorial on CUDA/GPU Programming for DNN

- Instructor: Bin Zhou, PhD
 - NVIDIA CUDA Fellow, USTC Adjunct Research Prof
 - Chief Scientist and Director of Marine Remote Sensing & Information Processing Lab, SDIOI
- Prerequisites
 - Computer architecture basics
 - C programming language
 - Numerical methods and analysis
 - Neural network
- Video lecture

http://www.iqiyi.com/a_19rrhbvoe9.html#vfrm=2-3-0-1

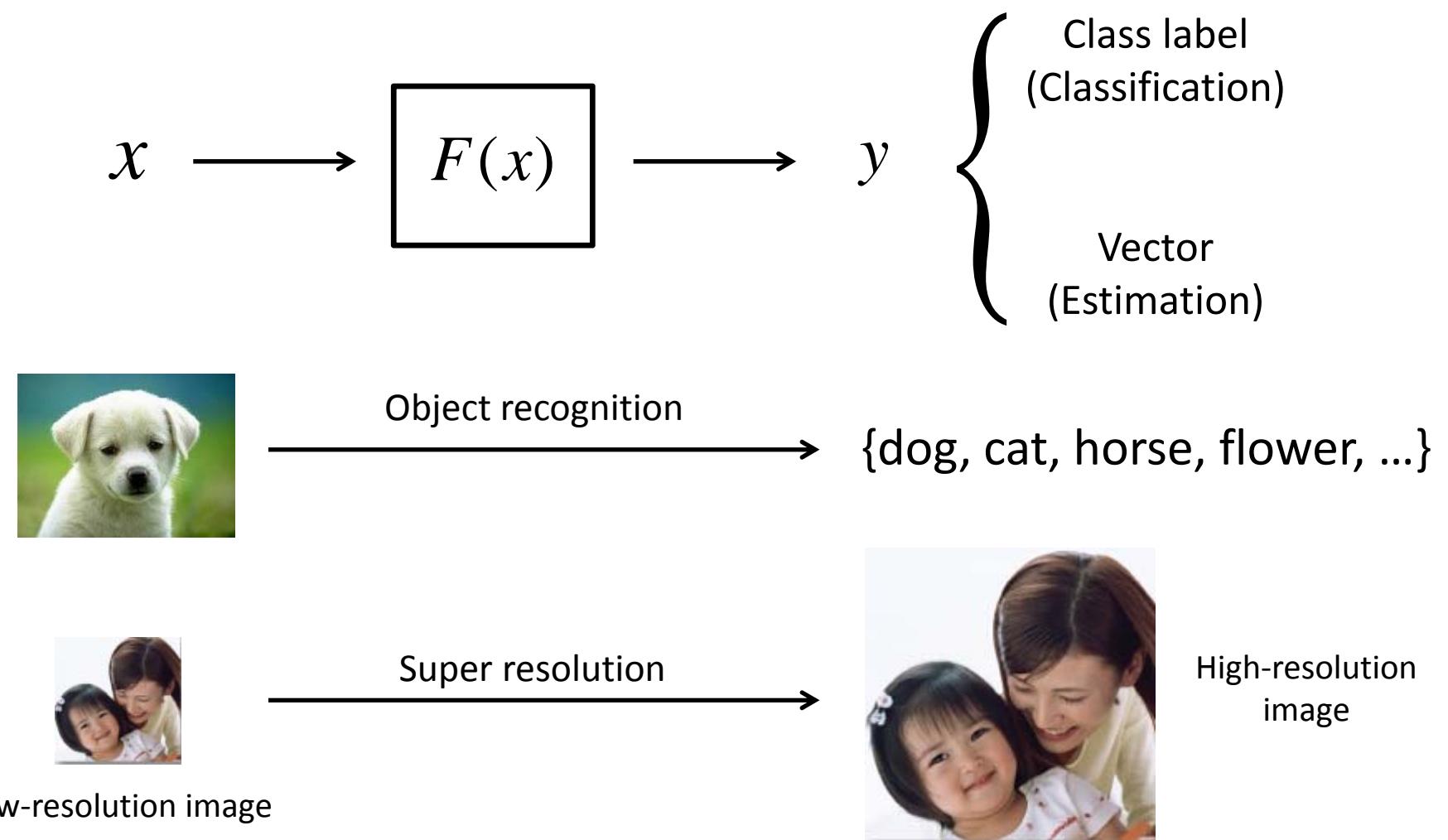
Topic	Content
Basics of CUDA (1.5h)	<ol style="list-style-type: none"> 1. CPU Architecture Review 2. Very Brief Review of Parallel Computing 3. Development Environment Configuration & Tools 4. GPU Architecture Review 5. GPU/CUDA Programming & Memory Model 6. CUDA Programming By Examples
Debugging, Profiling & Tools for CUDA/GPU (1 h, with lab works)	<ol style="list-style-type: none"> 1. Programming, Compiling 2. Debugging under windows & Linux 3. Profiling for Performance 4. Library and Tools
DNN with GPU/CUDA (1.5h, with lab works)	<ol style="list-style-type: none"> 1. Simple neural network with CUDA 2. cuDNN and cuda-convnet 3. Hands-on work for NN, cuDNN and cuda-convnet
CUDA Optimization for DNN (1h)	<ol style="list-style-type: none"> 1. General Optimization Procedure & Consideration 2. Efficient CUDA Programming Skills 3. Memory Throughput Optimization 4. DNN Analytical Optimization

Topic	Content
Advanced Topics with Multi-GPU and more (0.5h)	<ol style="list-style-type: none"> 1. Multi-GPU, Multi-Node 2. RDMA and GPUDirect 3. Hyper-Q 4. Dynamic Parallelization 5. Tegra K1
We're dealing with GPU/CUDA contents....	<ol style="list-style-type: none"> 1. Programming Model? 2. Memory Model? 3. WARP? 4. Occupancy? 5. Optimization 6. Compute Bound or memory Bound? 7. CUDA-GDB 8. Parallel Nsight?

Introduction to Deep Learning

- Historical review of deep learning
- Introduction to classical deep models
- Why does deep learning work?

Machine Learning

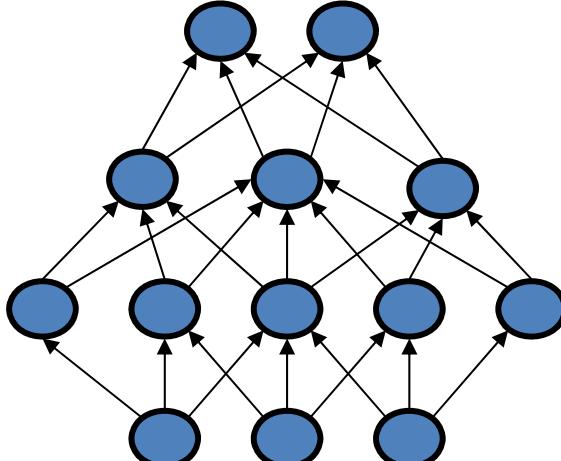


Neural network
Back propagation



Nature

1986

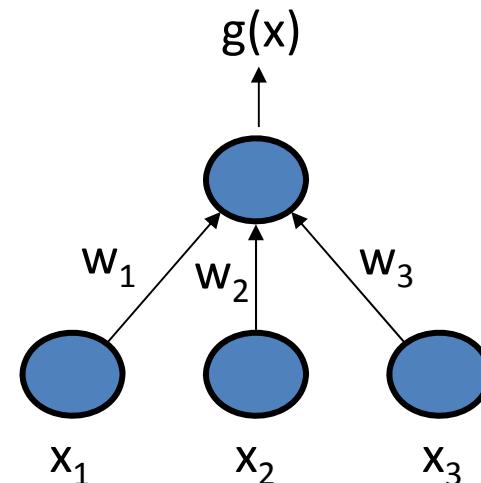
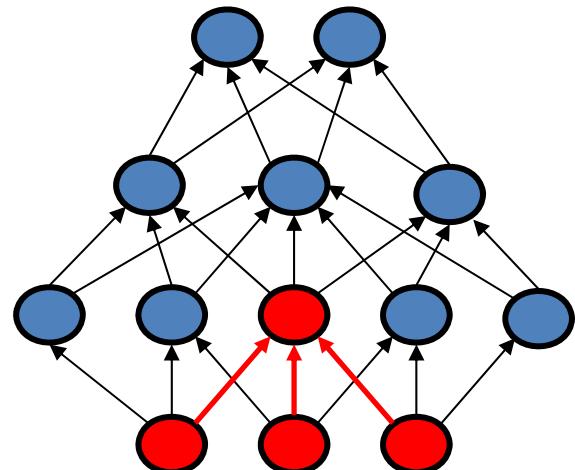
- 
- Solve general learning problems
 - Tied with biological system

Neural network
Back propagation

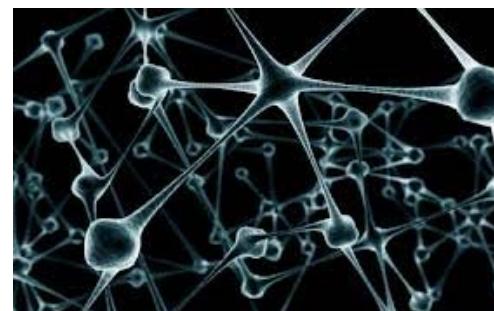
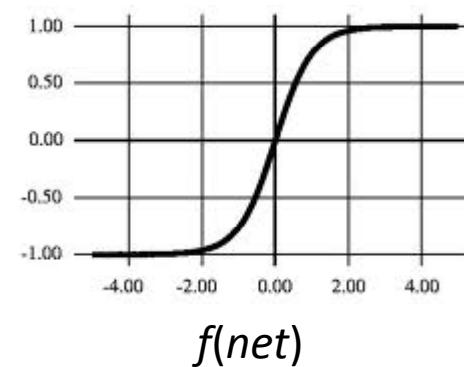


Nature

1986



$$g(\mathbf{x}) = f\left(\sum_{i=1}^d x_i w_i + w_0\right) = f(\mathbf{w}^t \mathbf{x})$$

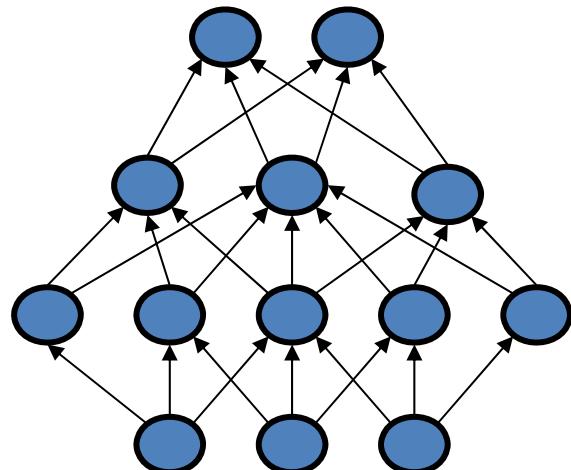


Neural network
Back propagation



Nature

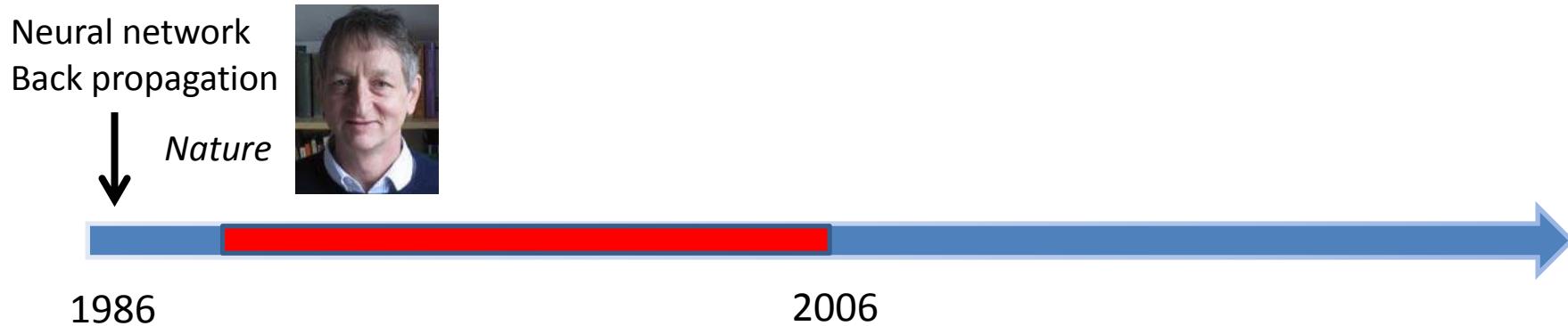
1986



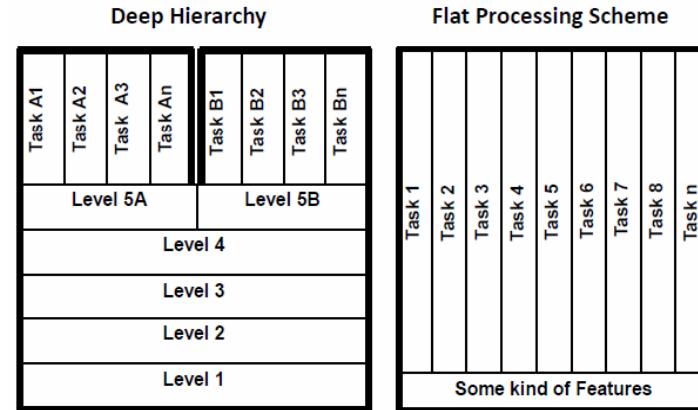
- Solve general learning problems
- Tied with biological system

But it is given up...

- Hard to train
- Insufficient computational resources
- Small training sets
- Does not work well



- SVM
 - Boosting
 - Decision tree
 - KNN
 - ...
- Flat structures
 - Loose tie with biological systems
 - Specific methods for specific tasks
 - Hand crafted features (GMM-HMM, SIFT, LBP, HOG)



Kruger et al. TPAMI'13

Neural network
Back propagation

↓
Nature

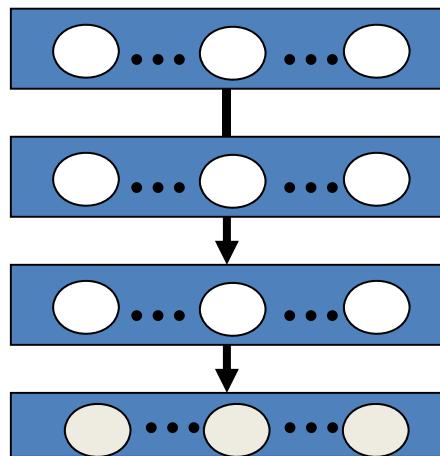
1986



Deep belief net
Science

↓

2006

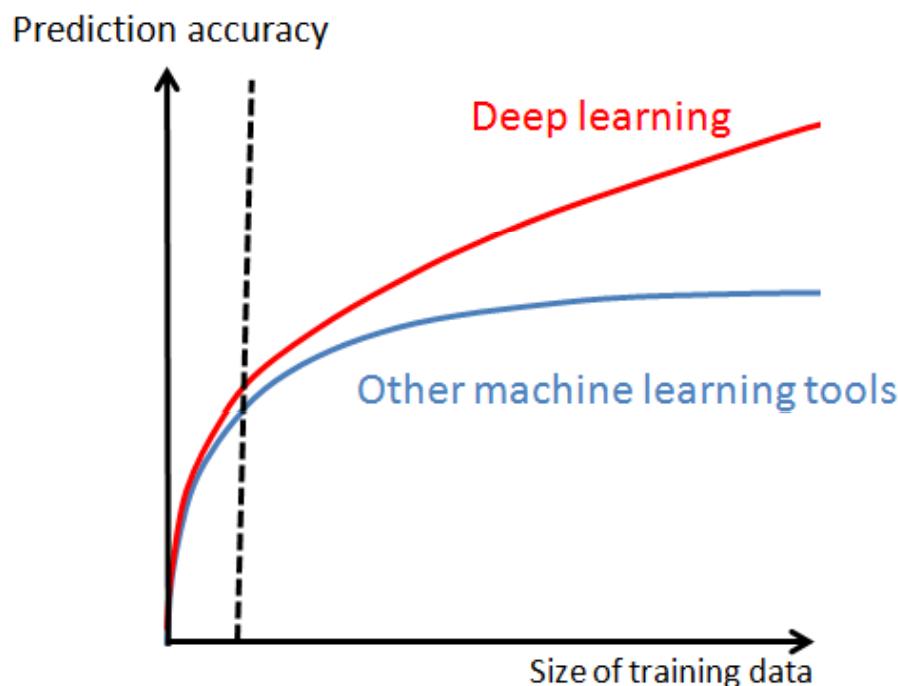


- Unsupervised & Layer-wised pre-training
- Better designs for modeling and training (normalization, nonlinearity, dropout)
- New development of computer architectures
 - GPU
 - Multi-core computer systems
- Large scale databases

Big Data !

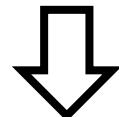
Machine Learning with Big Data

- Machine learning with small data: overfitting, reducing model complexity (capacity)
- Machine learning with big data: underfitting, increasing model complexity, optimization, computation resource

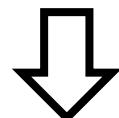


How to increase model capacity?

Curse of dimensionality

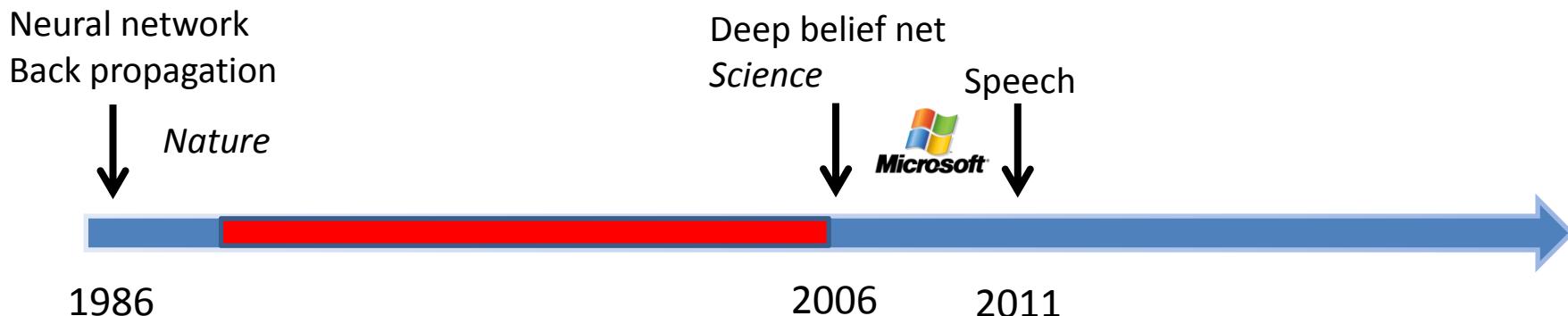


Blessing of dimensionality



**Learning hierarchical feature transforms
(Learning features with deep structures)**

D. Chen, X. Cao, F. Wen, and J. Sun. Blessing of dimensionality: Highdimensional feature and its efficient compression for face verification. In Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, 2013.



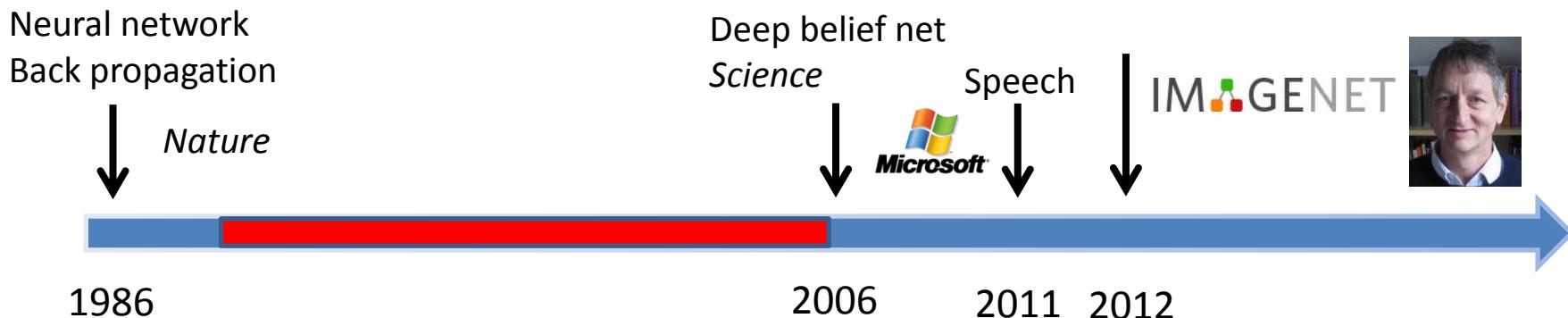
deep learning results

task	hours of training data	DNN-HMM	GMM-HMM with same data
Switchboard (test set 1)	309	18.5	27.4
Switchboard (test set 2)	309	16.1	23.6
English Broadcast News	50	17.5	18.8
Bing Voice Search (Sentence error rates)	24	30.4	36.2
Google Voice Input	5,870	12.3	
Youtube	1,400	47.6	52.3

Deep Networks Advance State of Art in Speech

Deep Learning leads to breakthrough in speech recognition at MSR.





Rank	Name	Error rate	Description
1	U. Toronto	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	Bottleneck.

Object recognition over 1,000,000 images and 1,000 categories (2 GPU)

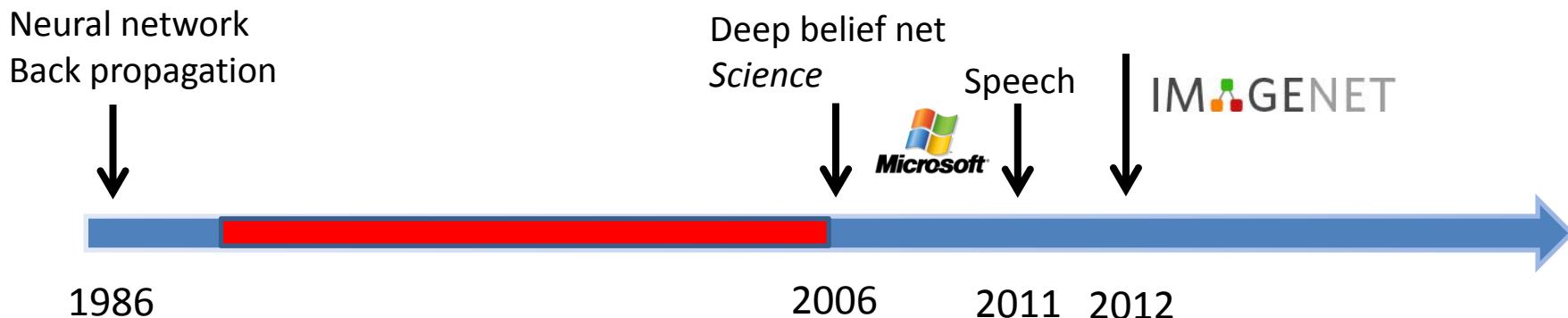
Examples from ImageNet

poster created by Fengjun Lv using VIPBase

1000 object classes that we recognize



images courtesy of ImageNet (<http://www.image-net.org/challenges/LSVRC/2010/index>)



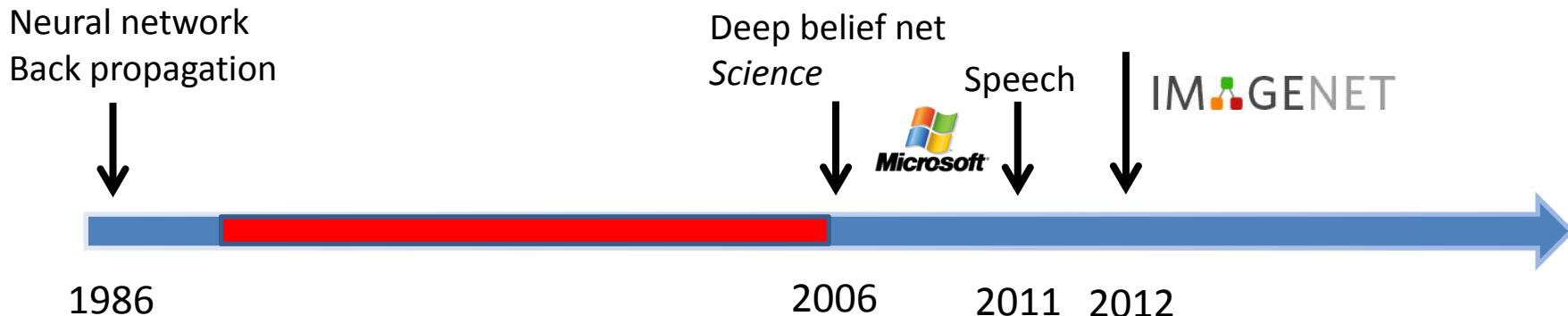
- ImageNet 2013 – image classification challenge

Rank	Name	Error rate	Description
1	NYU	0.11197	Deep learning
2	NUS	0.12535	Deep learning
3	Oxford	0.13555	Deep learning

MSRA, IBM, Adobe, NEC, Clarifai, Berkley, U. Tokyo, UCLA, UIUC, Toronto Top 20 groups all used deep learning

- ImageNet 2013 – object detection challenge

Rank	Name	Mean Average Precision	Description
1	UvA-Euvision	0.22581	Hand-crafted features
2	NEC-MU	0.20895	Hand-crafted features
3	NYU	0.19400	Deep learning

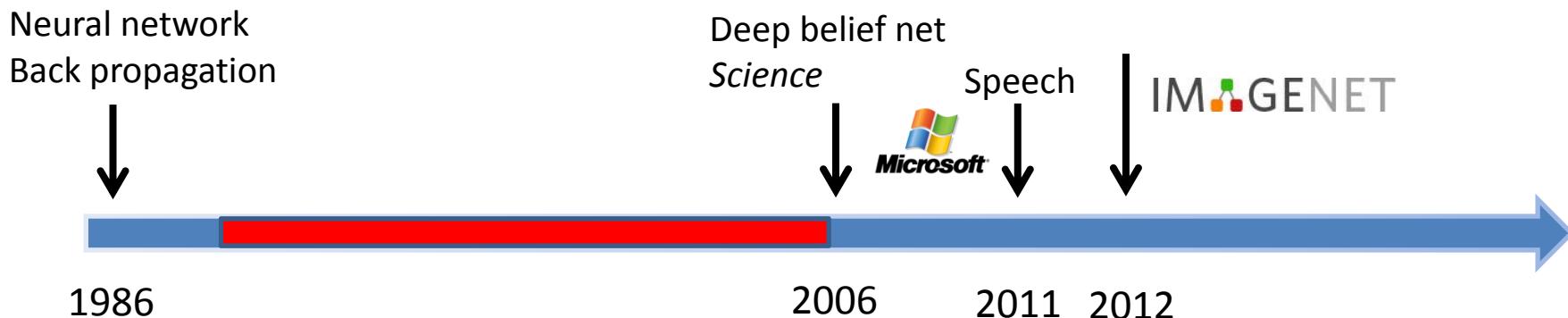


- ImageNet 2014 – Image classification challenge

Rank	Name	Error rate	Description
1	Google	0.06656	Deep learning
2	Oxford	0.07325	Deep learning
3	MSRA	0.08062	Deep learning

- ImageNet 2014 – object detection challenge

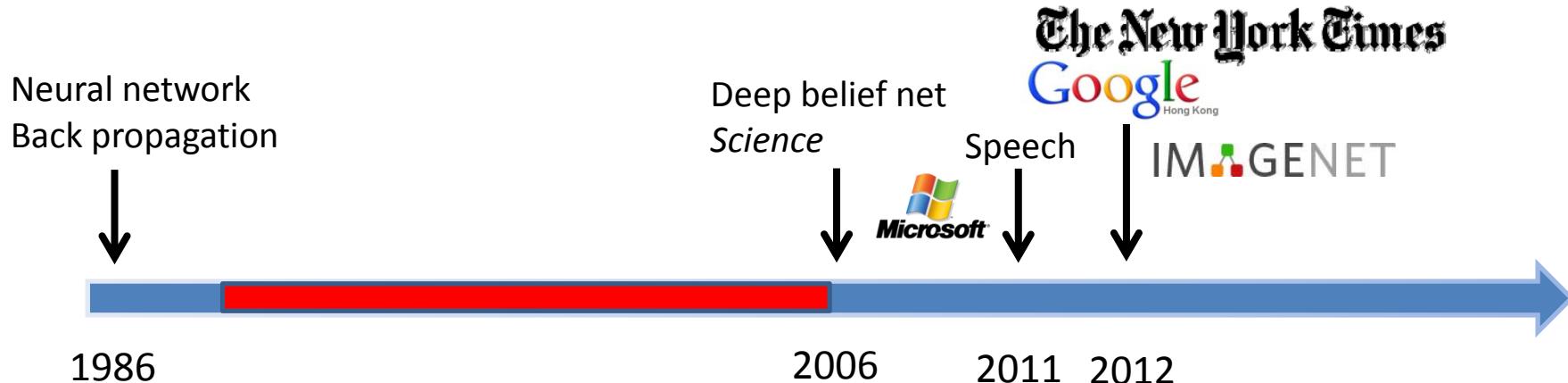
Rank	Name	Mean Average Precision	Description
1	Google	0.43933	Deep learning
2	CUHK	0.40656	Deep learning
3	DeepInsight	0.40452	Deep learning
4	UvA-Euvision	0.35421	Deep learning
5	Berkley Vision	0.34521	Deep learning



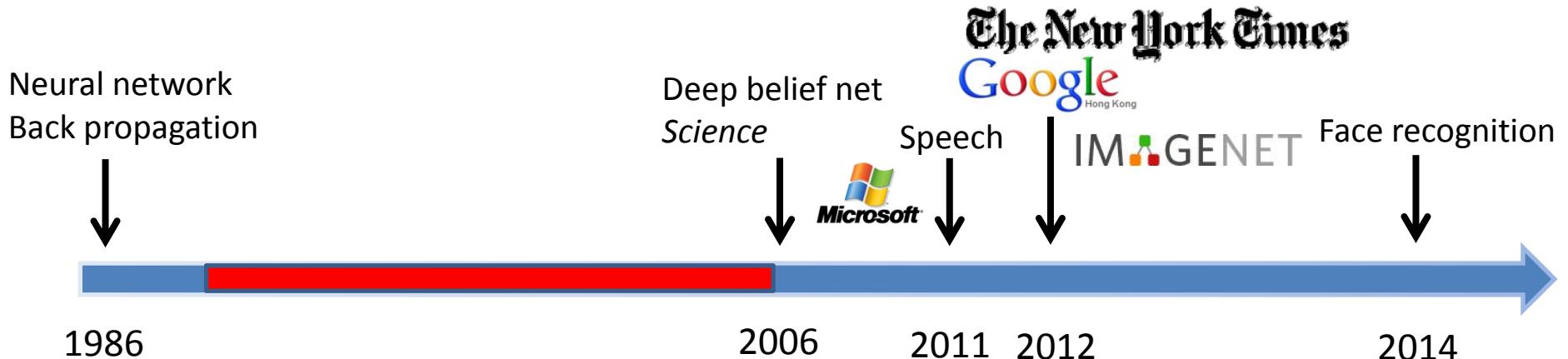
- ImageNet 2014 – object detection challenge

	GoogLeNet (Google)	DeepID-Net (CUHK)	DeepInsight	UvA-Euvision	Berkley Vision	RCNN
Model average	0.439	0.439	0.405	n/a	n/a	n/a
Single model	0.380	0.427	0.402	0.354	0.345	0.314

W. Ouyang et al. “DeepID-Net: multi-stage and deformable deep convolutional neural networks for object detection”, arXiv:1409.3505, 2014



- Google and Baidu announced their deep learning based visual search engines (2013)
 - Google
 - “on our test set we saw **double the average precision** when compared to other approaches we had tried. We acquired the rights to the technology and went full speed ahead adapting it to run at large scale on Google’s computers. We took cutting edge research straight out of an academic research lab and launched it, in just a little over six months.”
 - Baidu

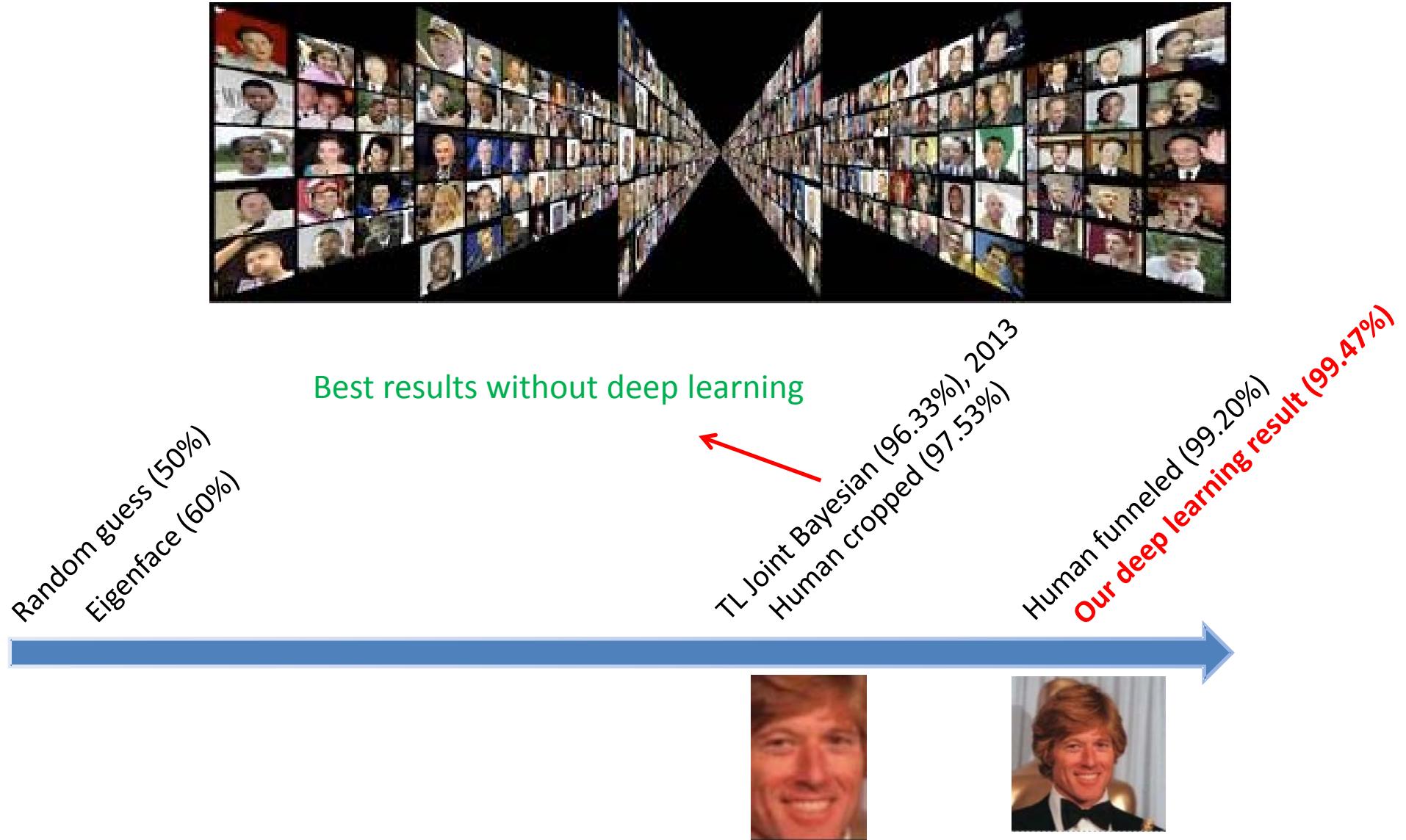


- Deep learning achieves 99.47% face verification accuracy on Labeled Faces in the Wild (LFW), higher than human performance

Y. Sun, X. Wang, and X. Tang. Deep Learning Face Representation by Joint Identification-Verification. NIPS, 2014.

Y. Sun, X. Wang, and X. Tang. Deeply learned face representations are sparse, selective, and robust. arXiv:1412.1265, 2014.

Labeled Faces in the Wild (2007)



Unrestricted, Labeled Outside Data Results

Attribute classifiers ¹¹	0.8525 ± 0.0060
Simile classifiers ¹¹	0.8414 ± 0.0041
Attribute and Simile classifiers ¹¹	0.8554 ± 0.0035
Multiple LE + comp ¹⁴	0.8445 ± 0.0046
Associate-Predict ¹⁸	0.9057 ± 0.0056
Tom-vs-Pete ²³	0.9310 ± 0.0135
Tom-vs-Pete + Attribute ²³	0.9330 ± 0.0128
combined Joint Bayesian ²⁶	0.9242 ± 0.0108
high-dim LBP ²⁷	0.9517 ± 0.0113
DFD ³³	0.8402 ± 0.0044
TL Joint Bayesian ³⁴	0.9633 ± 0.0108
face.com r2011b ¹⁹	0.9130 ± 0.0030
Face++ ⁴⁰	0.9727 ± 0.0065
DeepFace-ensemble ⁴¹	0.9735 ± 0.0025
ConvNet-RBM ⁴²	0.9252 ± 0.0038
POOF-gradhist ⁴⁴	0.9313 ± 0.0040
POOF-HOG ⁴⁴	0.9280 ± 0.0047
FR+FCN ⁴⁵	0.9645 ± 0.0025
DeepID ⁴⁶	0.9745 ± 0.0026
GaussianFace ⁴⁷	0.9852 ± 0.0066
DeepID2 ⁴⁸	0.9915 ± 0.0013

Table 6: Mean classification accuracy \hat{U} and standard error of the mean S_E .

10 BREAKTHROUGH TECHNOLOGIES 2013

[Introduction](#)[The 10 Technologies](#)[Past Years](#)

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.

Temporary Social Media

Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous.

Prenatal DNA Sequencing

Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child?

Additive Manufacturing

Skeptical about 3-D printing? GE, the world's largest manufacturer, is on the verge of using the technology to make jet parts.

Baxter: The Blue-Collar Robot

Rodney Brooks's newest creation is easy to interact with, but the complex innovations behind the robot show just how hard it is to get along with people.

Memory Implants

A maverick neuroscientist believes he has deciphered the code by which the brain forms long-term memories. Next: testing a prosthetic implant for people suffering from long-term memory loss.

Smart Watches

The designers of the Pebble watch realized that a mobile phone is more useful if you don't have to take it out of your pocket.

Ultra-Efficient Solar Power

Doubling the efficiency of a solar cell would completely change the economics of renewable energy. Nanotechnology just might make it possible.

Big Data from Cheap Phones

Collecting and analyzing information from simple cell phones can provide surprising insights into how people move about and behave – and even help us understand the spread of diseases.

Supergrids

A new high-power circuit breaker could finally make highly efficient DC power grids practical.

Is Google Cornering the Market on Deep Learning?

A cutting-edge corner of science is being wooed by Silicon Valley, to the dismay of some academics.

By Antonio Regalado on January 29, 2014

How much are a dozen deep-learning researchers worth? Apparently, more than \$400 million.

The acquisition, aimed at adding skilled experts rather than specific products, marks an acceleration in efforts by Google, Facebook, and other Internet firms to monopolize the biggest brains in artificial intelligence research.

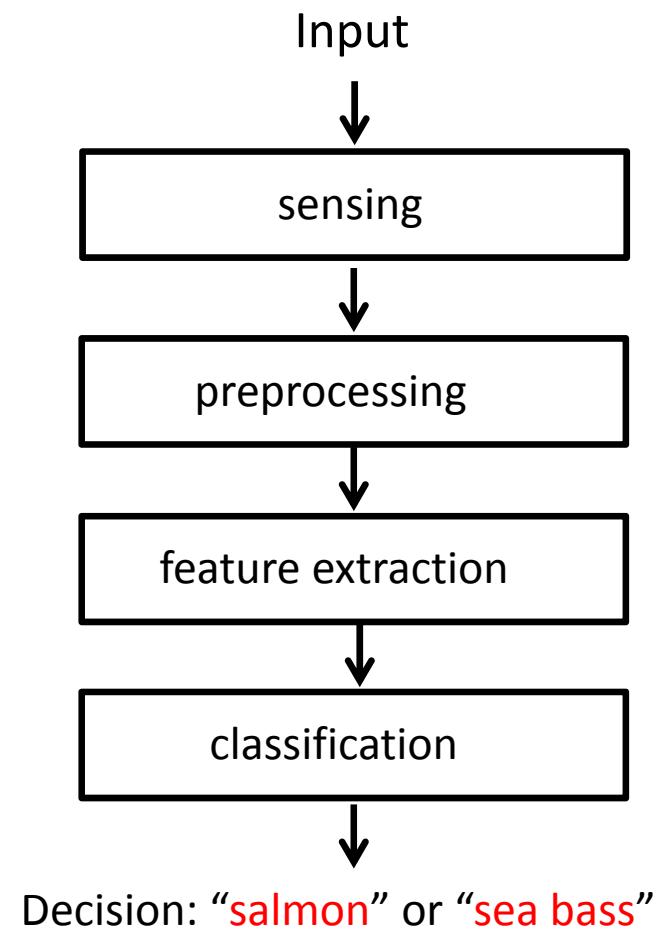
News on Deep Learning

Baidu established Institute of Deep Learning	2012
Hinton's group won ImageNet Contest	Oct. 2012
Hinton joined Google	March 2013
Google announced deep learning based visual search engine	March 2013
Baidu announced deep learning based visual search engine	June 2013
Yahoo acquired startup LookFlow working on deep learning	Oct. 2013
Facebook established a new AI lab in NewYork and recruited Yann LeCun	Dec. 2013
Google Acquires DeepMind for USD 400 Million	January 2014
Baidu established a new lab at Shenzhen, China	2014
Baidu established a new lab at silicon valley and Andrwe Ng is the director	May 2014
Deep learning reached human performance on face verification on LFW	June 2014

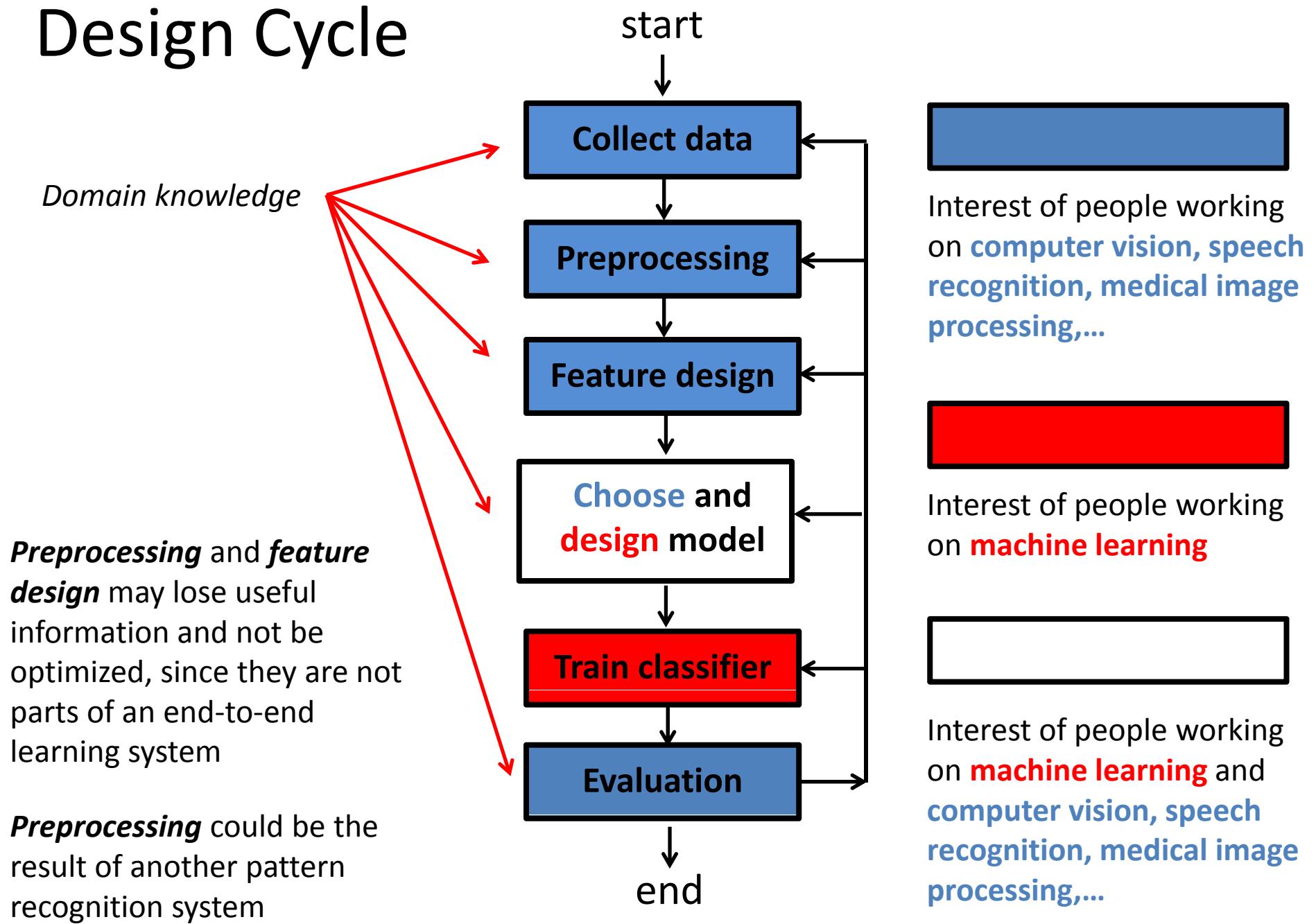
Deep Learning Changes the Design Cycle

- How do machine learning, computer vision, and speech recognition people study pattern recognition problems in different ways?
- Why did some computer vision people have concern about their research life when deep learning emerged?
- How does deep learning change the design cycle of a pattern recognition system?

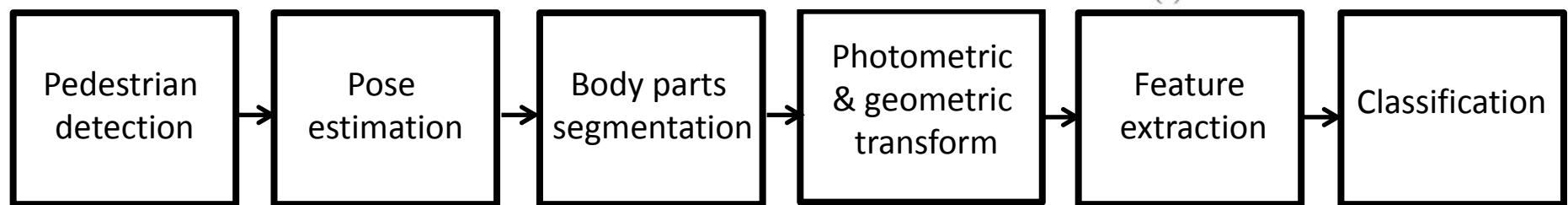
Pattern Recognition System



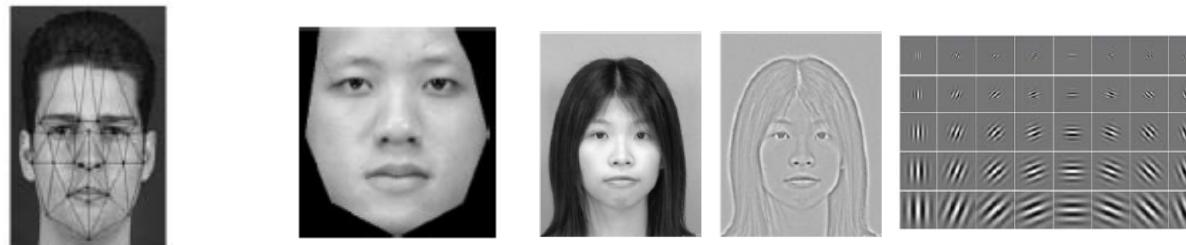
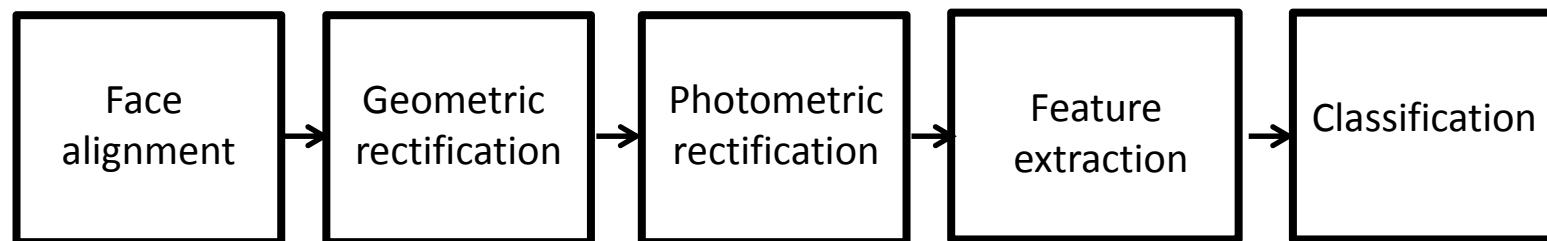
Design Cycle



Person re-identification pipeline

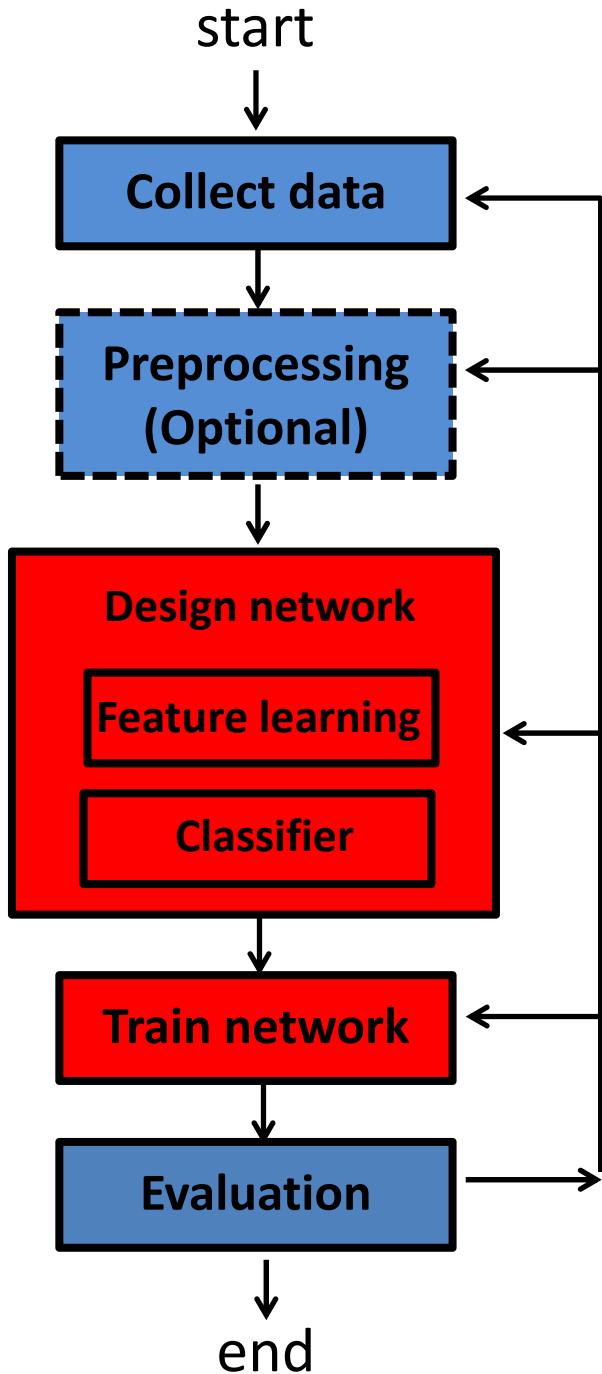


Face recognition pipeline



Design Cycle with Deep Learning

- Learning plays a bigger role in the design circle
- Feature learning becomes part of the end-to-end learning system
- Preprocessing becomes optional means that several pattern recognition steps can be merged into one end-to-end learning system
- Feature learning makes the key difference
- We underestimated the importance of data collection and evaluation



What makes deep learning successful in computer vision?

Li Fei-Fei



IM_{AGE}NET

Geoffrey Hinton



Data collection

One million images
with labels

Evaluation task

Predict 1,000 image
categories

Deep learning

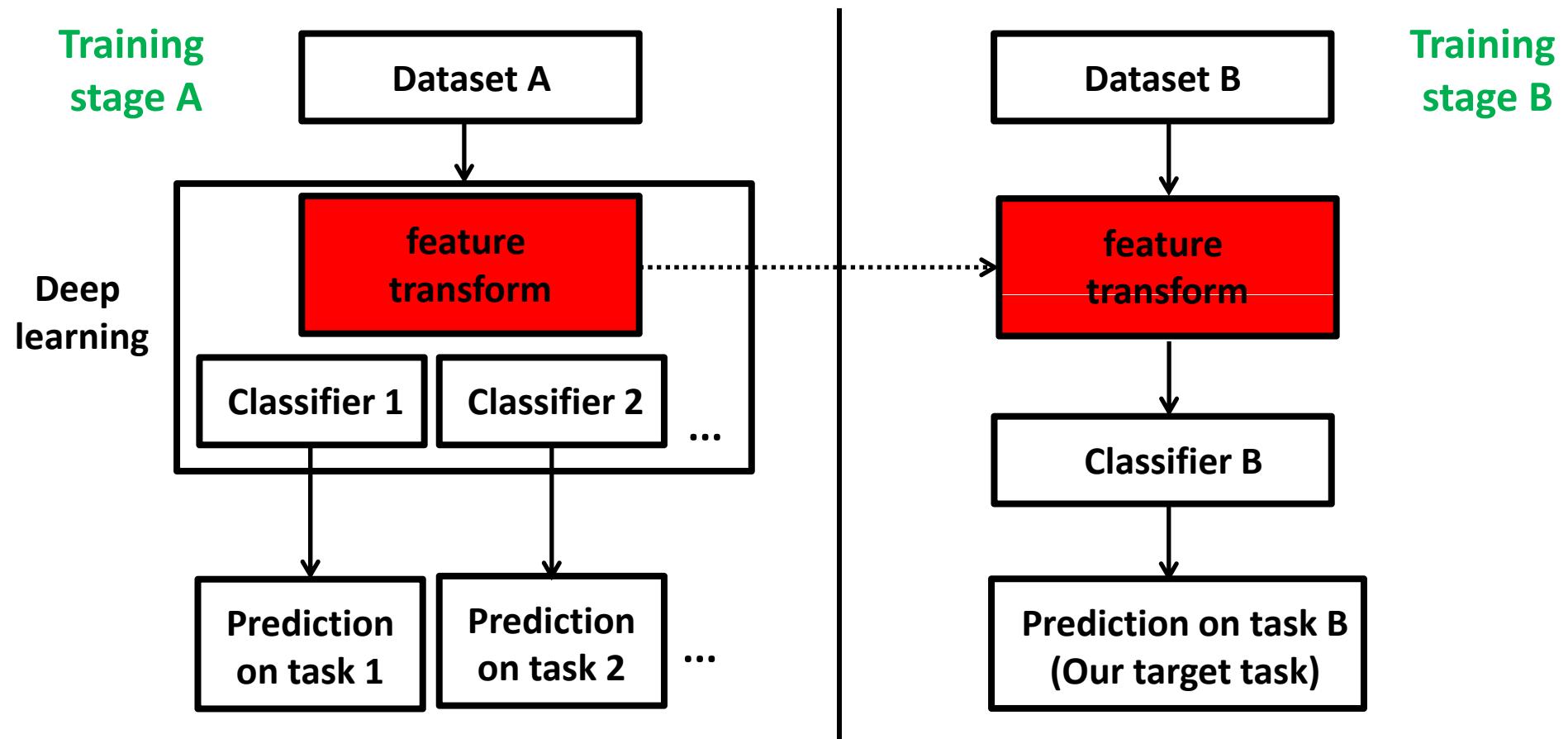
CNN is not new
Design network structure

New training strategies

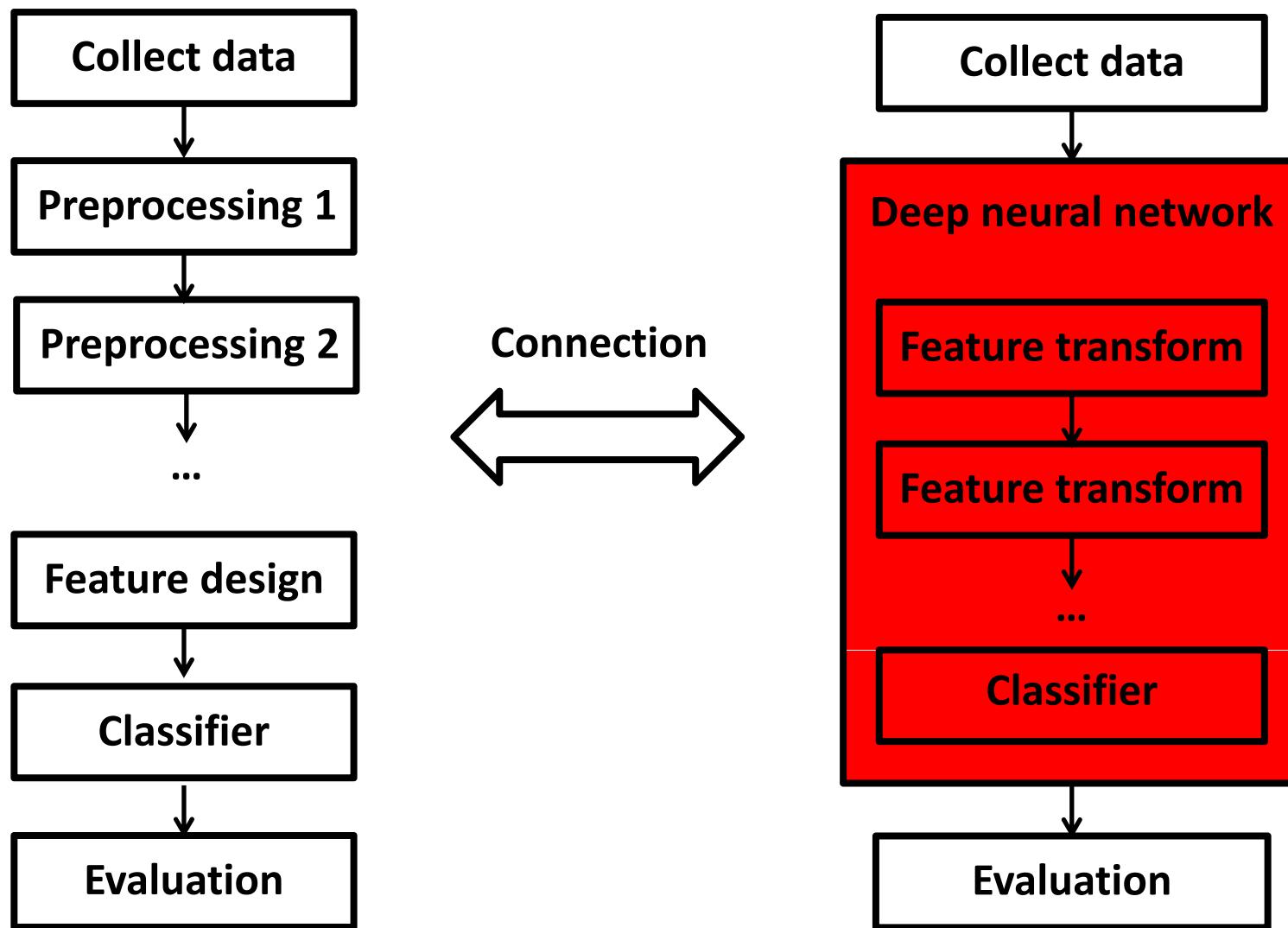
Feature learned from ImageNet can be well generalized to other tasks and datasets!

Learning features and classifiers separately

- Not all the datasets and prediction tasks are suitable for learning features with deep models



Deep learning can be treated as a language to described the world with great flexibility



Introduction to Deep Learning

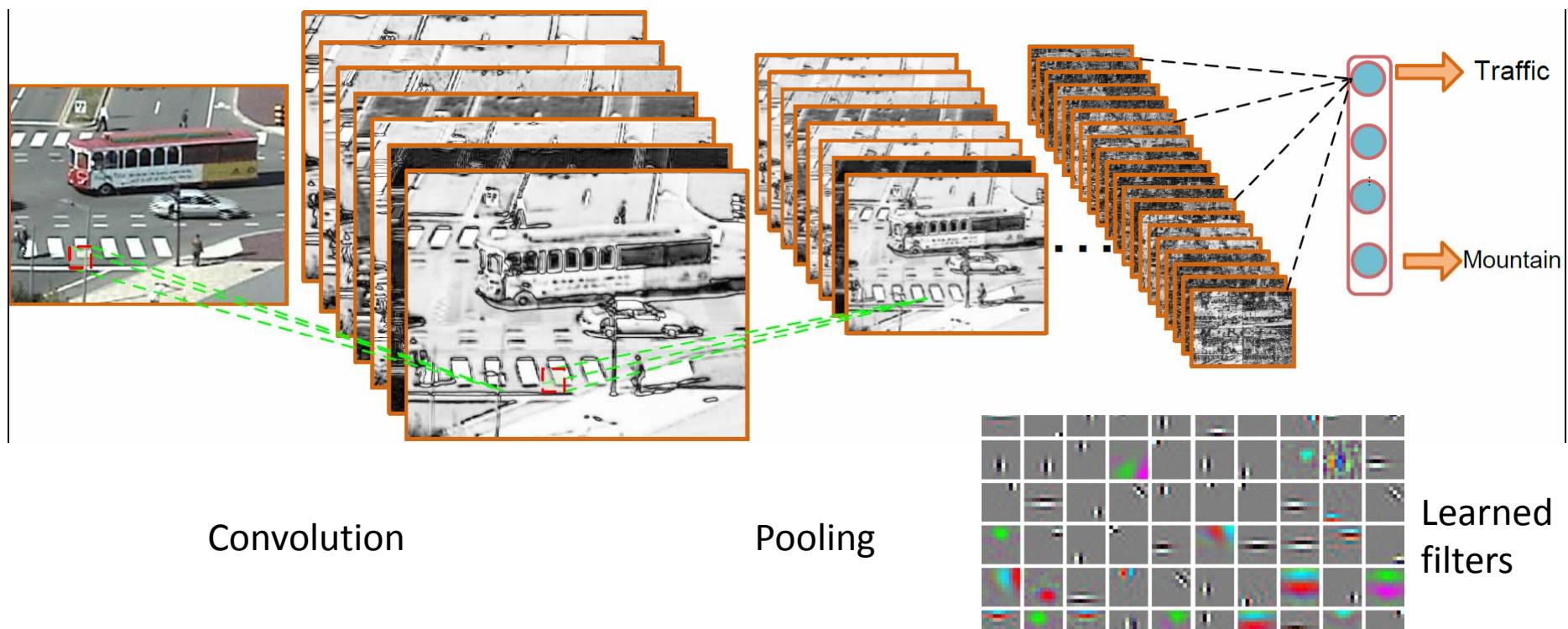
- Historical review of deep learning
- **Introduction to classical deep models**
- Why does deep learning work?

Introduction on Classical Deep Models

- Convolutional Neural Networks (CNN)
 - Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based Learning Applied to Document Recognition,” Proceedings of the IEEE, Vol. 86, pp. 2278-2324, 1998.
- Deep Belief Net (DBN)
 - G. E. Hinton, S. Osindero, and Y. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” Neural Computation, Vol. 18, pp. 1527-1544, 2006.
- Auto-encoder
 - G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” Science, Vol. 313, pp. 504-507, July 2006.

Classical Deep Models

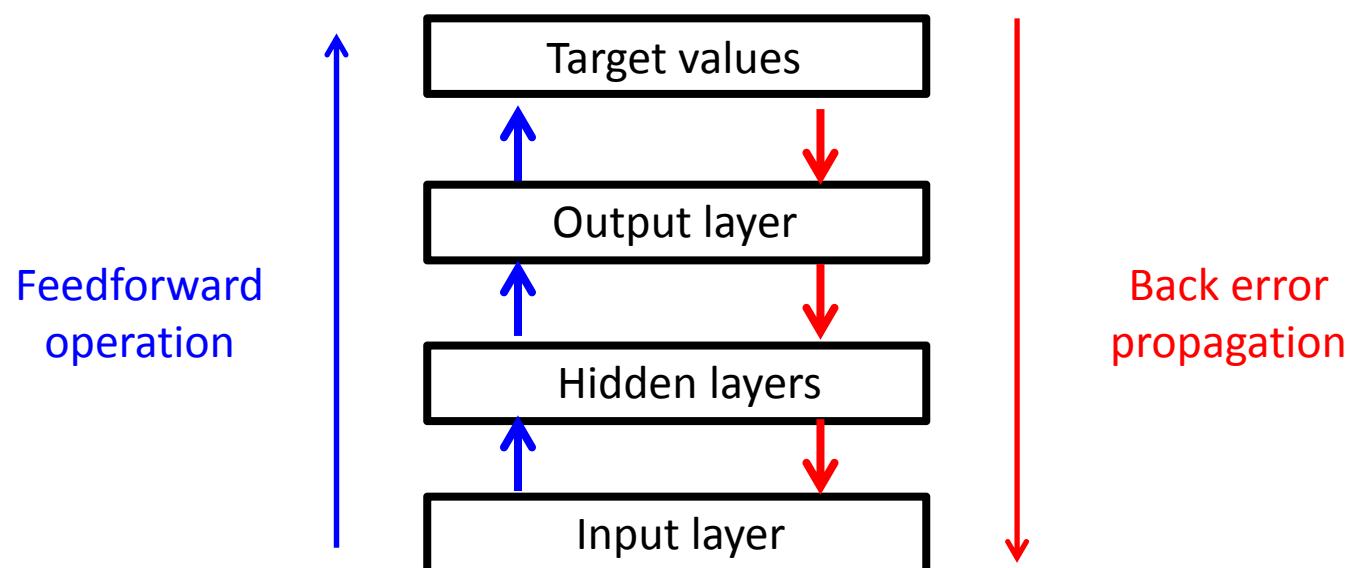
- Convolutional Neural Networks (CNN)
 - First proposed by Fukushima in 1980
 - Improved by LeCun, Bottou, Bengio and Haffner in 1998



Backpropagation

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \bigtriangledown J(\mathbf{W})$$

\mathbf{W} is the parameter of the network; J is the objective function



D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning Representations by Back-propagation Errors," Nature, Vol. 323, pp. 533-536, 1986.

Classical Deep Models

- Deep belief net
 - Hinton'06

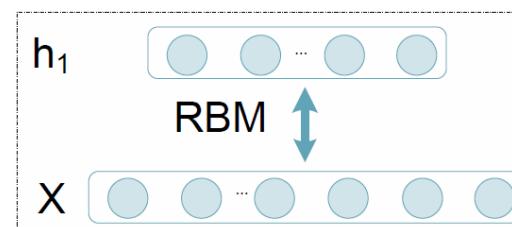
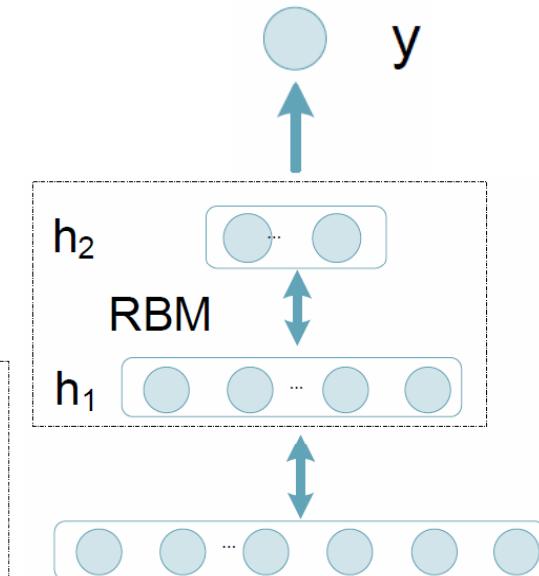
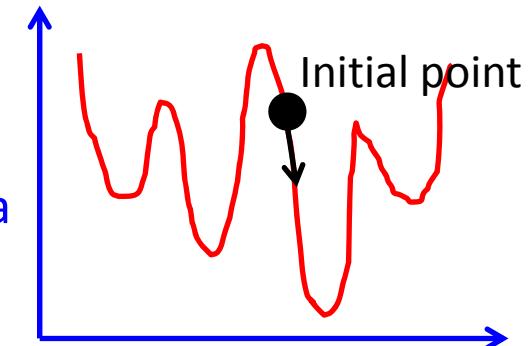
Pre-training:

- Good initialization point
- Make use of unlabeled data

$$P(\mathbf{x}, \mathbf{h}_1, \mathbf{h}_2) = p(\mathbf{x} | \mathbf{h}_1) p(\mathbf{h}_1, \mathbf{h}_2)$$

$$P(\mathbf{x}, \mathbf{h}_1) = \frac{e^{-E(\mathbf{x}, \mathbf{h}_1)}}{\sum_{\mathbf{x}, \mathbf{h}_1} e^{-E(\mathbf{x}, \mathbf{h}_1)}}$$

$$E(\mathbf{x}, \mathbf{h}_1) = \mathbf{b}' \mathbf{x} + \mathbf{c}' \mathbf{h}_1 + \mathbf{h}_1' \mathbf{W} \mathbf{x}$$



Classical Deep Models

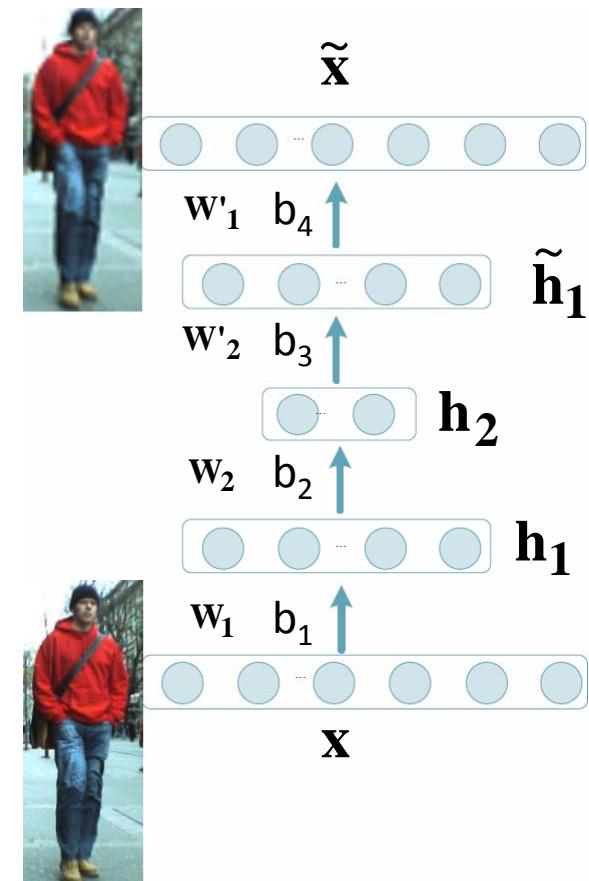
- Auto-encoder
 - Hinton and Salakhutdinov 2006

$$\text{Encoding: } \mathbf{h}_1 = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2)$$

$$\text{Decoding: } \tilde{\mathbf{h}}_1 = \sigma(\mathbf{W}'_2 \mathbf{h}_2 + \mathbf{b}_3)$$

$$\tilde{\mathbf{x}} = \sigma(\mathbf{W}'_1 \mathbf{h}_1 + \mathbf{b}_4)$$



Introduction to Deep Learning

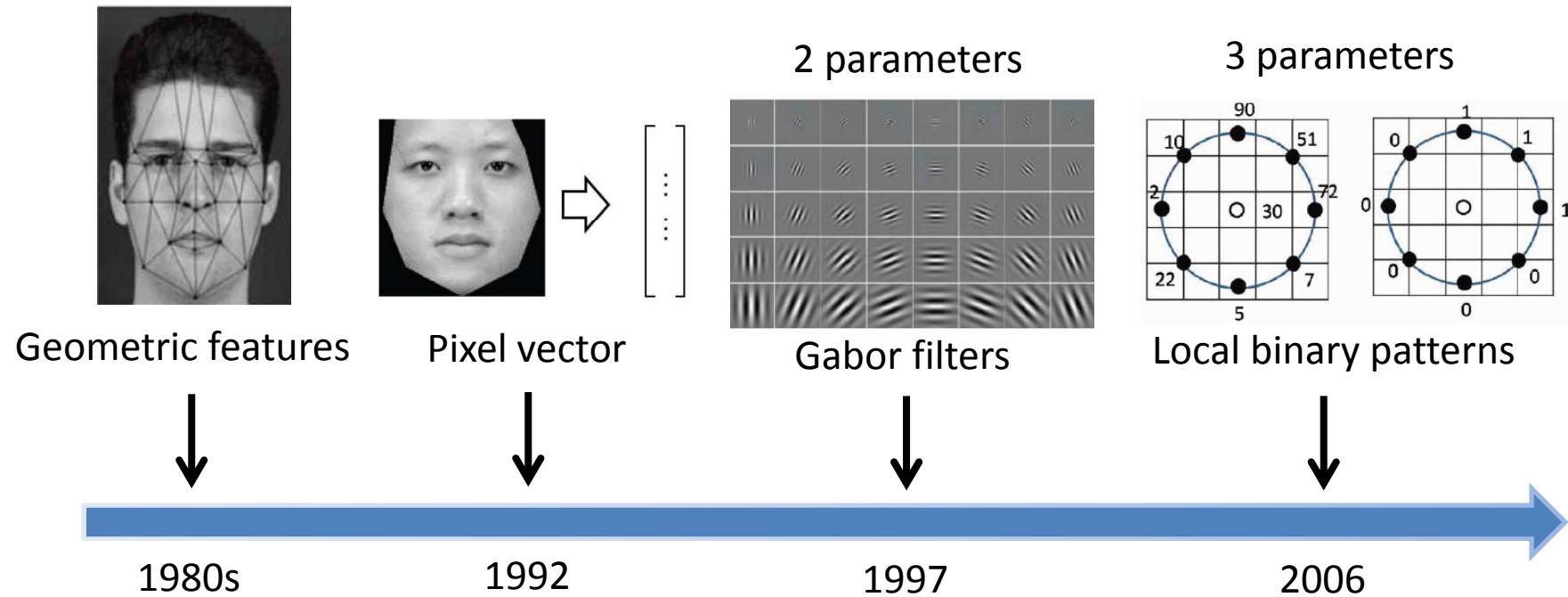
- Historical review of deep learning
- Introduction to classical deep models
- **Why does deep learning work?**

Feature Learning vs Feature Engineering

Feature Engineering

- The performance of a pattern recognition system heavily depends on feature representations
- Manually designed features dominate the applications of image and video understanding in the past
 - Reply on human domain knowledge much more than data
 - Feature design is separate from training the classifier
 - If handcrafted features have multiple parameters, it is hard to manually tune them
 - Developing effective features for new applications is slow

Handcrafted Features for Face Recognition



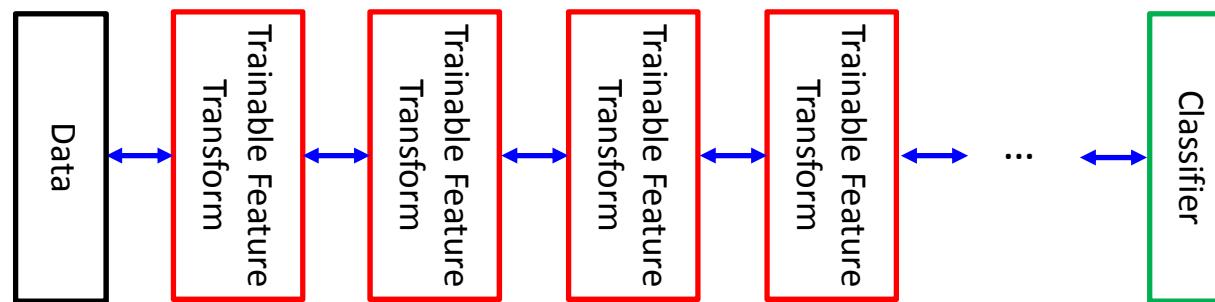
Feature Learning

- Learning transformations of the data that make it easier to extract useful information when building classifiers or predictors
 - Jointly learning feature transformations and classifiers makes their integration optimal
 - Learn the values of a huge number of parameters in feature representations, **which dramatically increase the capacity of deep models**
 - Make better use of big data
 - Faster to get feature representations for new applications

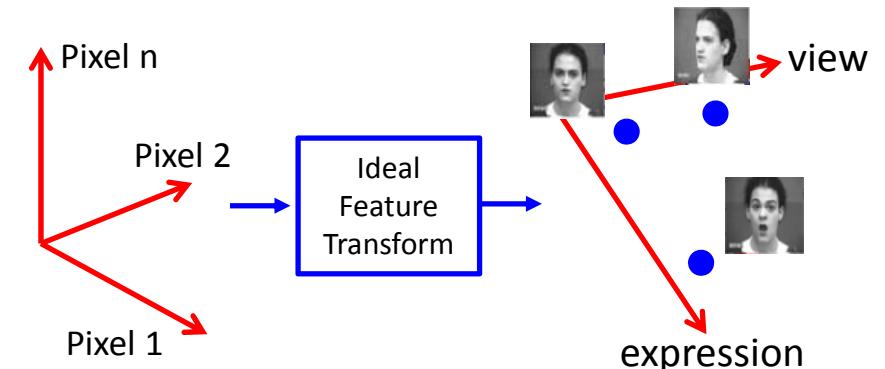
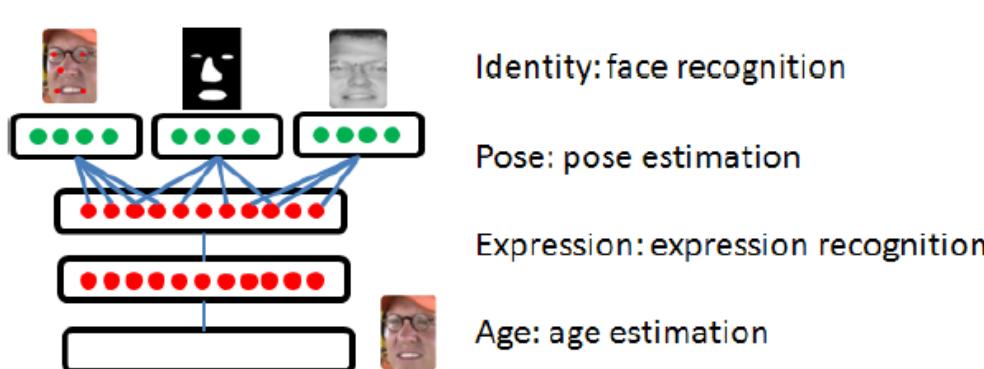
Deep Learning Means Feature Learning

- Deep learning is about learning hierarchical feature representations

$$\mathbf{y} = F(\mathbf{W}^k \cdot F(\mathbf{W}^{k-1} \cdot F(\dots F(\mathbf{W}^0 \cdot \mathbf{x})))$$

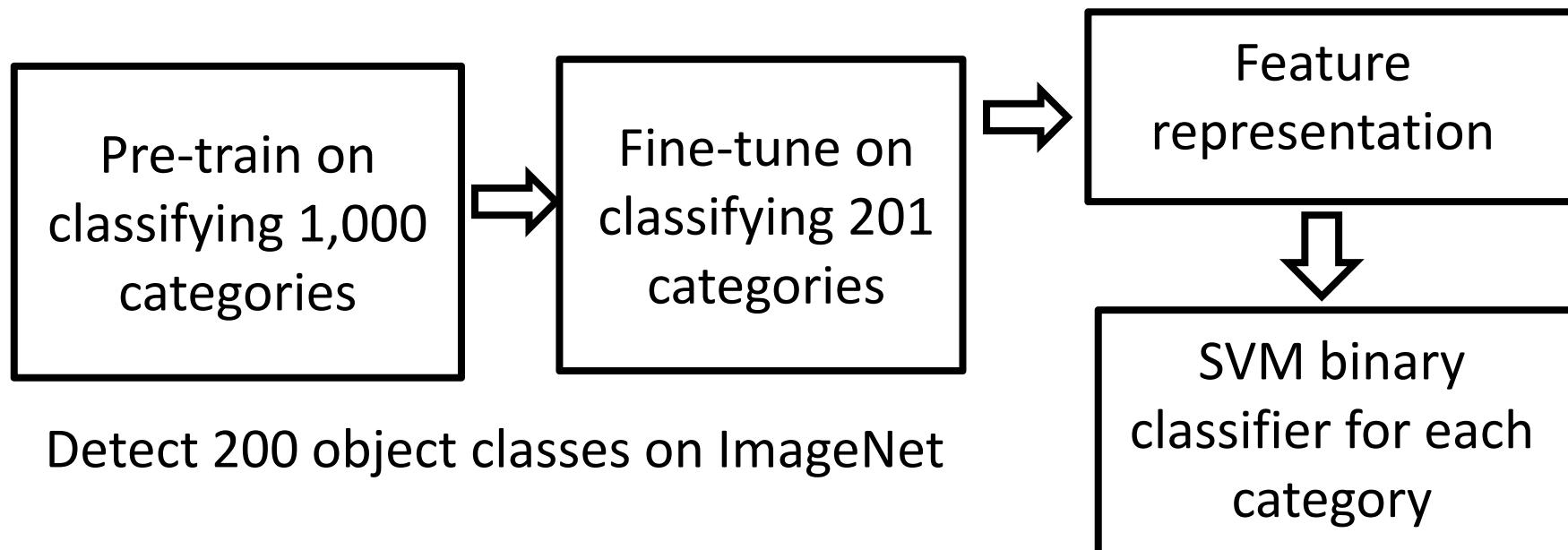


- Good feature representations should be able to disentangle multiple factors coupled in the data



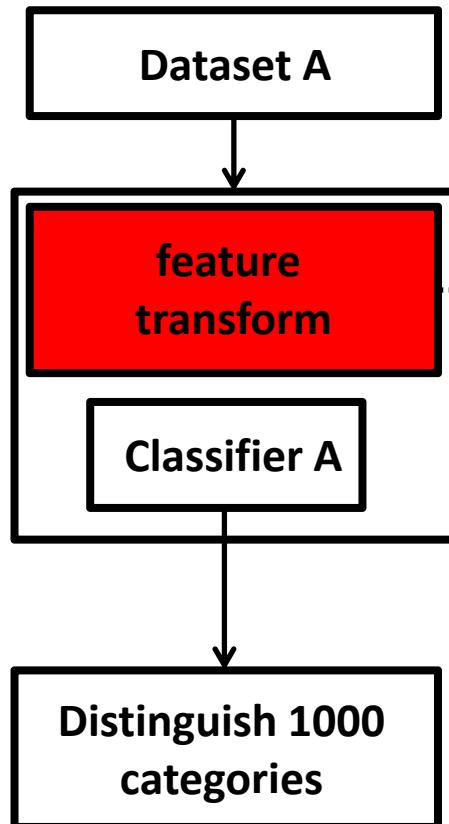
Deep Learning Means Feature Learning

- How to effectively learn features with deep models
 - With challenging tasks
 - Predict high-dimensional vectors

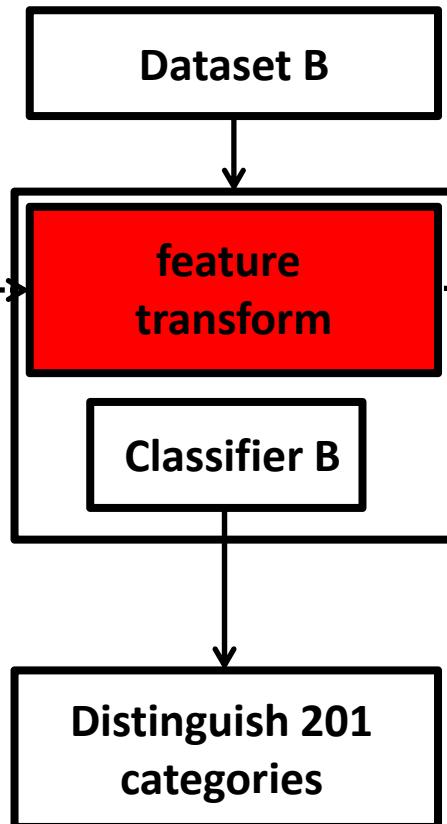


W. Ouyang et al. “DeepID-Net: multi-stage and deformable deep convolutional neural networks for object detection”, arXiv:1409.3505, 2014

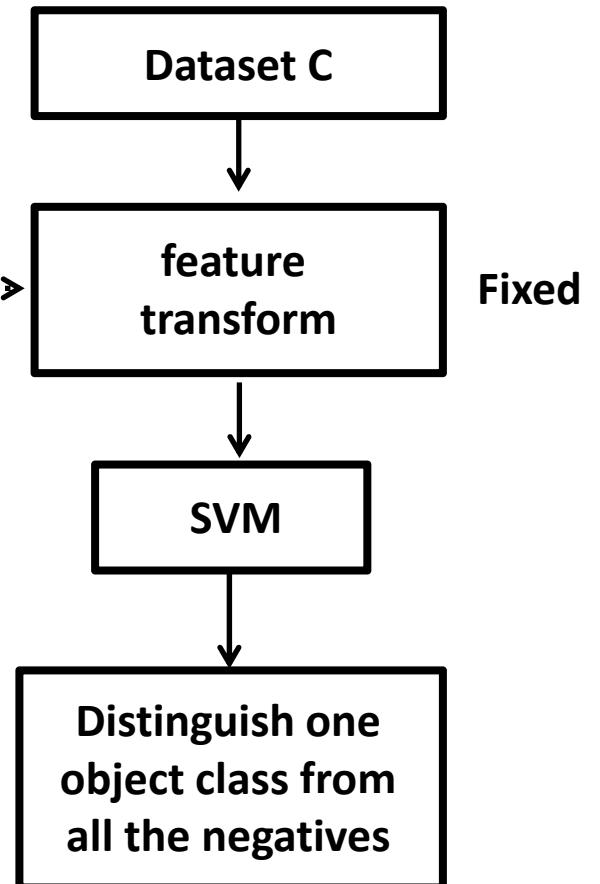
Training stage A



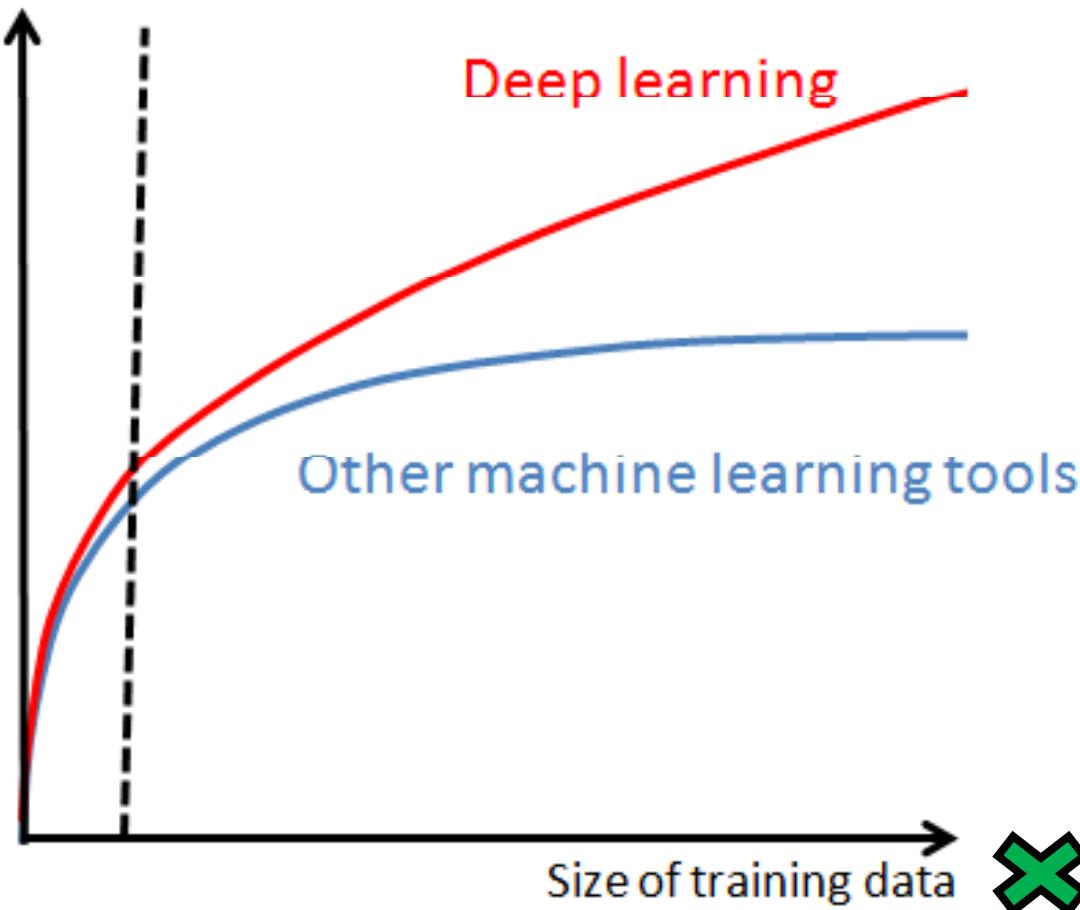
Training stage B



Training stage C



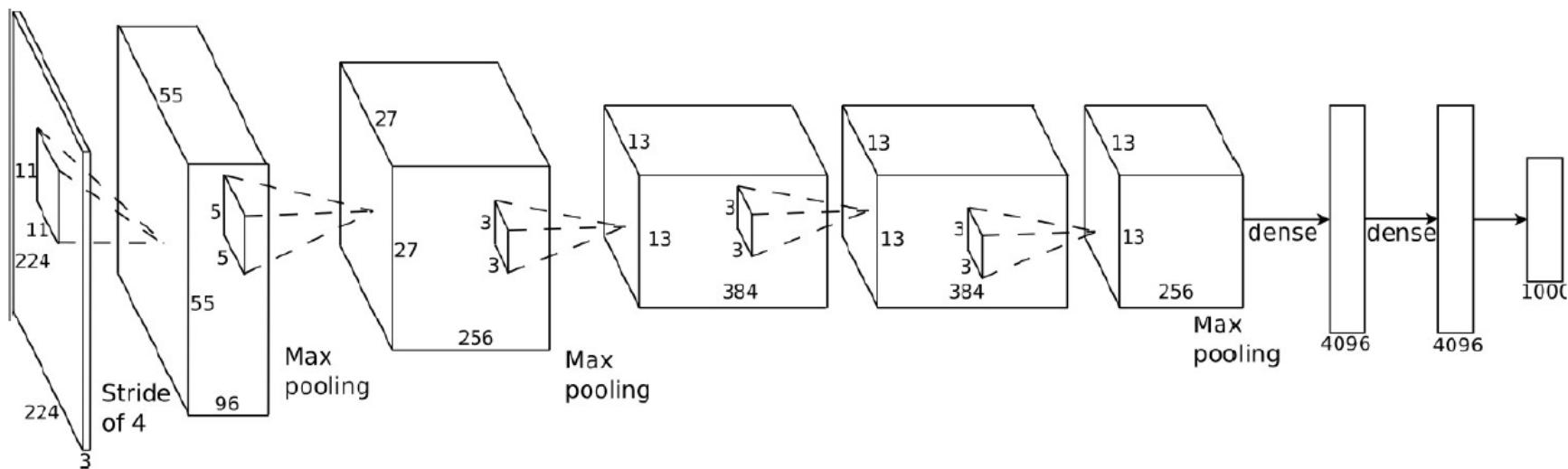
Prediction accuracy



Amount of information in the training data

Example 1: deep learning generic image features

- Hinton group's groundbreaking work on ImageNet
 - They did not have much experience on general image classification on ImageNet
 - It took one week to train the network with 60 Million parameters
 - The learned feature representations are effective on other datasets (e.g. Pascal VOC) and other tasks (object detection, segmentation, tracking, and image retrieval)



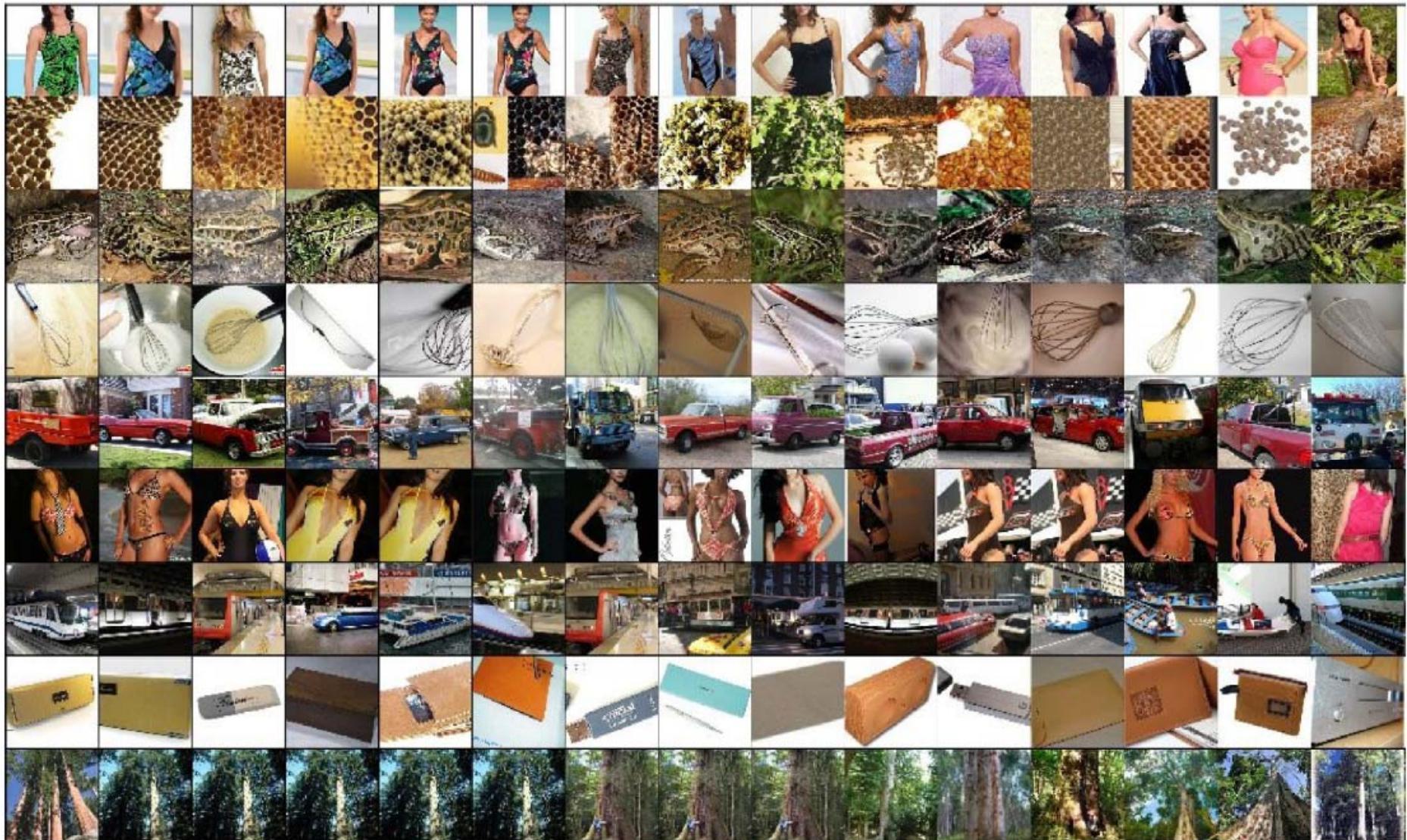
96 learned low-level filters



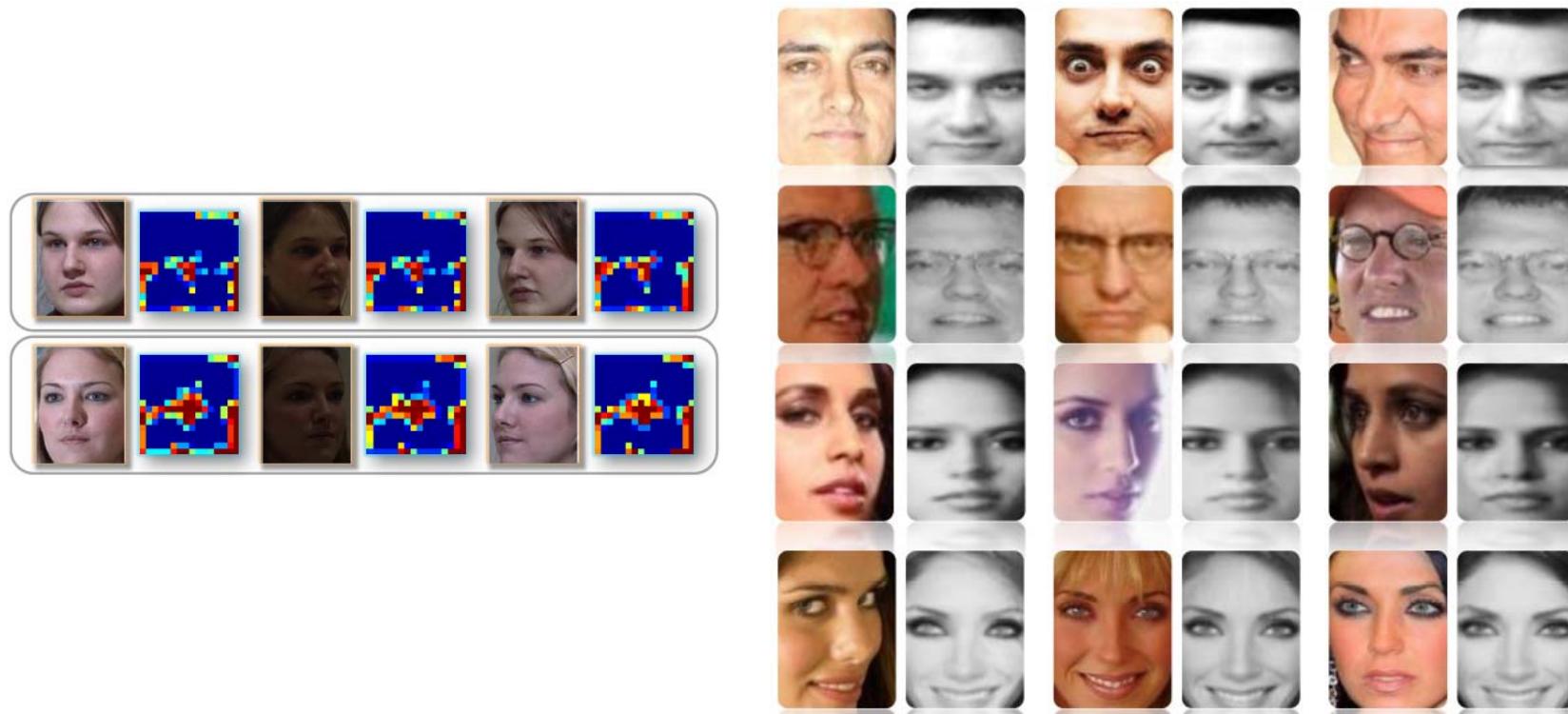
Image classification result

			
mite black widow cockroach tick starfish	container ship lifeboat amphibian fireboat drilling platform	motor scooter go-kart moped bumper car golfcart	leopard jaguar cheetah snow leopard Egyptian cat
			
grille convertible grille pickup beach wagon fire engine	mushroom agaric mushroom jelly fungus gill fungus dead-man's-fingers	cherry dalmatian grape elderberry ffordshire bullterrier currant	Madagascar cat squirrel monkey spider monkey titi indri howler monkey

Top hidden layer can be used as feature for retrieval



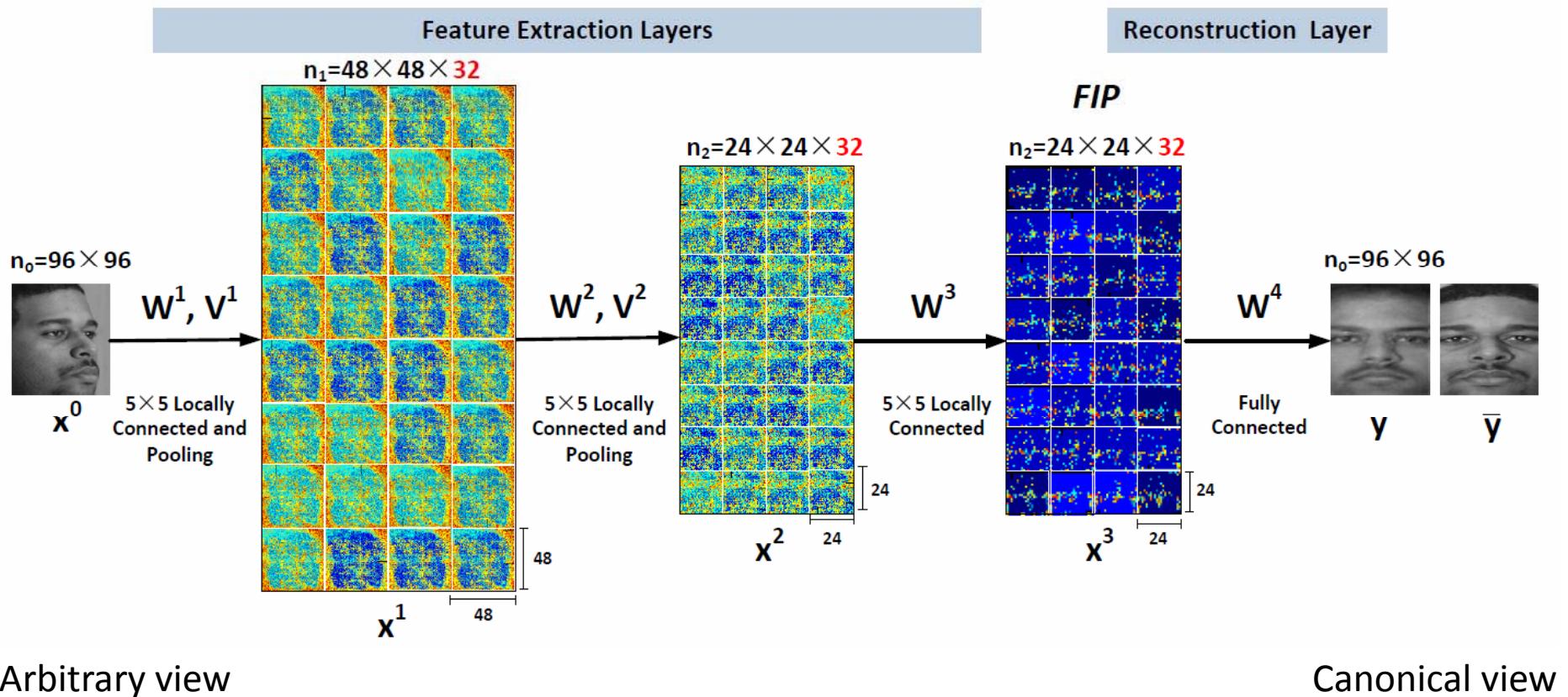
Example 2: deep learning face identity features by recovering canonical-view face images

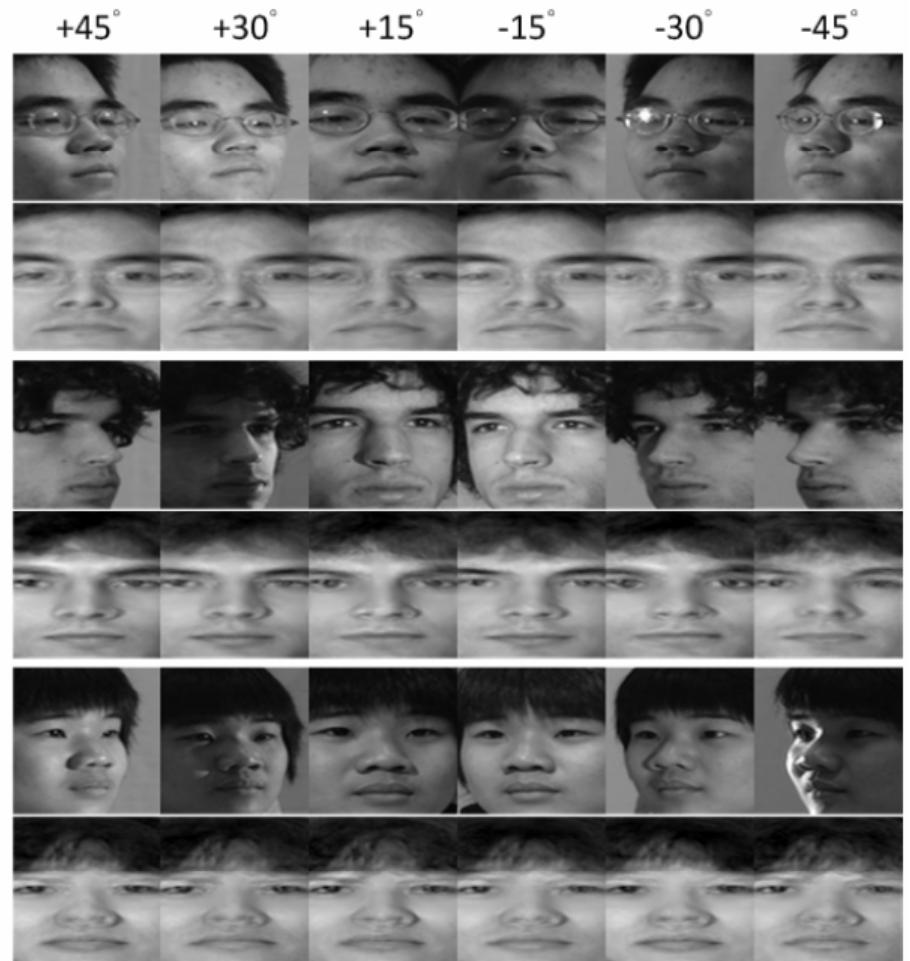


Reconstruction examples from LFW

Z. Zhu, P. Luo, X. Wang, and X. Tang, “Deep Learning Identity Preserving Face Space,” ICCV 2013.

- Deep model can disentangle hidden factors through feature extraction over multiple layers
- No 3D model; no prior information on pose and lighting condition
- Model multiple complex transforms
- Reconstructing the whole face is a much strong supervision than predicting 0/1 class label and helps to avoid overfitting





Comparison on Multi-PIE

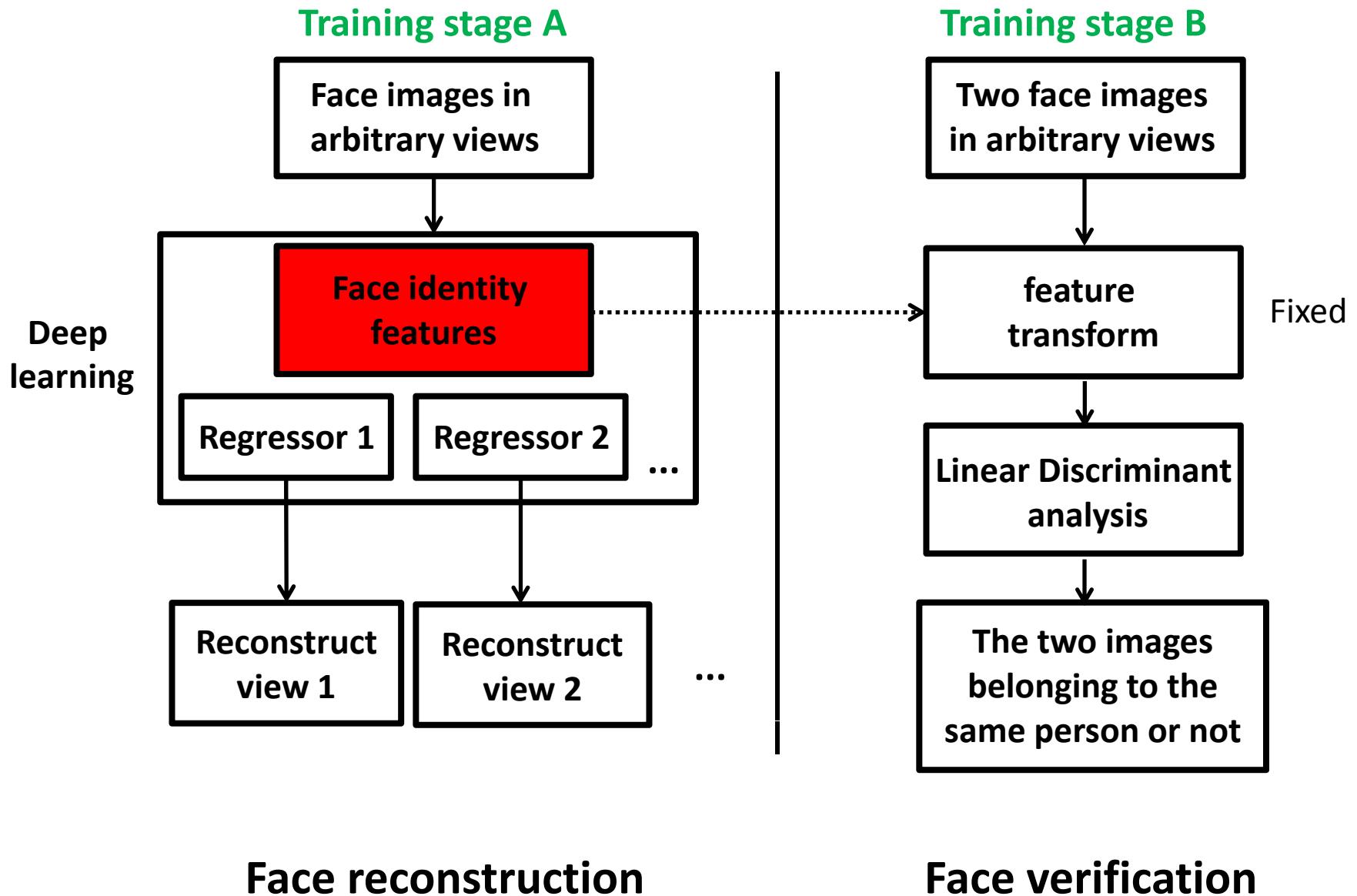
	-45°	-30°	-15°	+15°	+30°	+45°	Avg	Pose
LGBP [26]	37.7	62.5	77	83	59.2	36.1	59.3	✓
VAAM [17]	74.1	91	95.7	95.7	89.5	74.8	86.9	✓
FA-EGFC[3]	84.7	95	99.3	99	92.9	85.2	92.7	✗
SA-EGFC[3]	93	98.7	99.7	99.7	98.3	93.6	97.2	✓
LE[4] + LDA	86.9	95.5	99.9	99.7	95.5	81.8	93.2	✗
CRBM[9] + LDA	80.3	90.5	94.9	96.4	88.3	89.8	87.6	✗
Ours	95.6	98.5	100.0	99.3	98.5	97.8	98.3	✗

- [3] A. Asthana, T. K. Marks, M. J. Jones, K. H. Tieu, and M. Rohith. Fully automatic pose-invariant face recognition via 3d pose normalization. In *ICCV*, pages 937–944, 2011. [1](#), [5](#), [6](#)
- [4] Z. Cao, Q. Yin, X. Tang, and J. Sun. Face recognition with learning-based descriptor. In *CVPR*, pages 2707–2714, 2010. [2](#), [3](#), [6](#)
- [9] G. B. Huang, H. Lee, and E. Learned-Miller. Learning hierarchical representations for face verification with convolutional deep belief networks. In *CVPR*, pages 2518–2525, 2012. [3](#), [6](#)
- [17] S. Li, X. Liu, X. Chai, H. Zhang, S. Lao, and S. Shan. Morphable displacement field based image matching for face recognition across pose. In *ECCV*, pages 102–115, 2012. [1](#), [2](#), [5](#), [6](#)
- [26] W. Zhang, S. Shan, W. Gao, X. Chen, and H. Zhang. Local gabor binary pattern histogram sequence (lgbphs): A novel non-statistical model for face representation and recognition. In *ICCV*, volume 1, pages 786–791, 2005. [5](#), [6](#)

Deep learning 3D model from 2D images, mimicking human brain activities

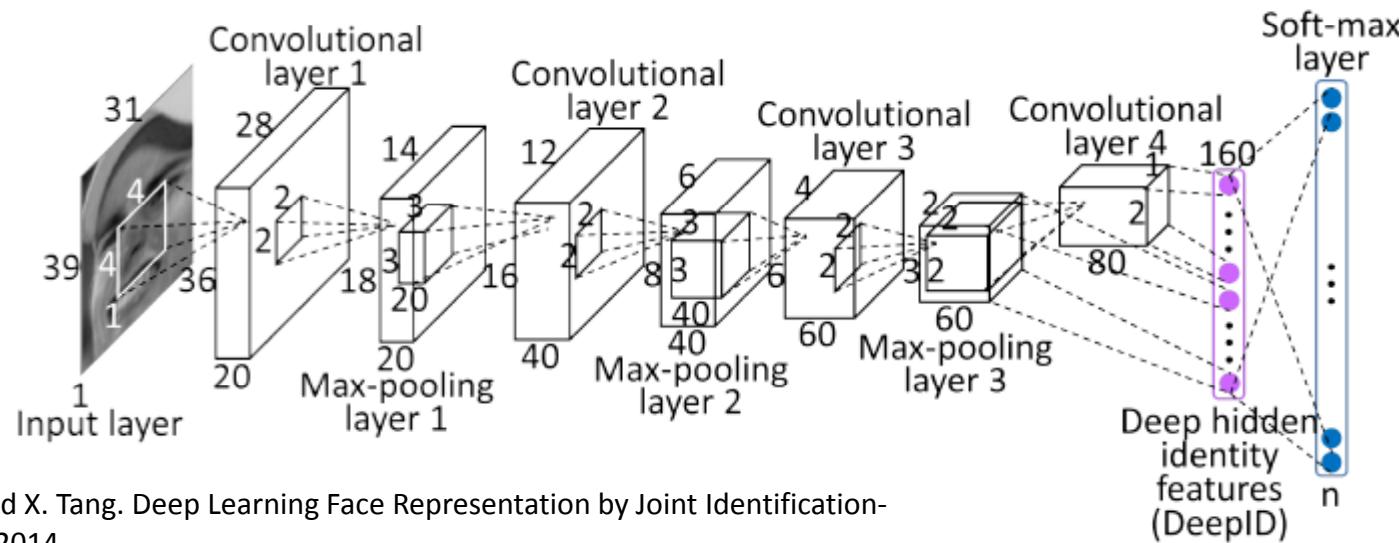


Z. Zhu, P. Luo, X. Wang, and X. Tang, "Deep Learning and Disentangling Face Representation by Multi-View Perception," NIPS 2014.

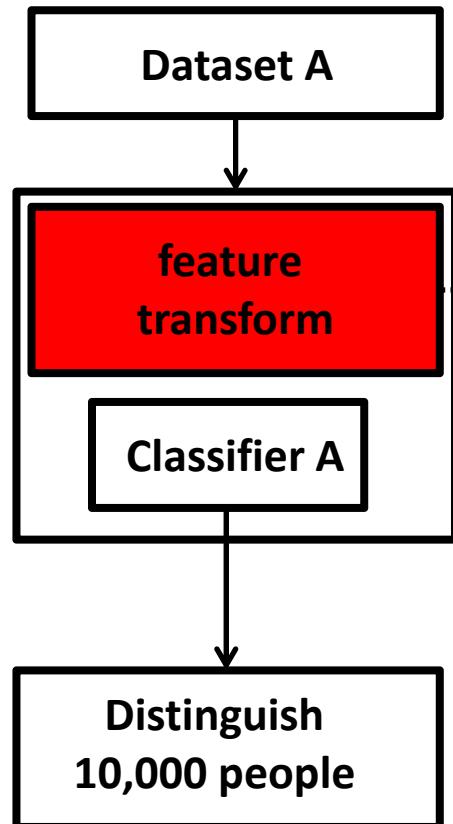


Example 3: deep learning face identity features from predicting 10,000 classes

- At training stage, each input image is classified into 10,000 identities with 160 hidden identity features in the top layer
- The hidden identity features can be well generalized to other tasks (e.g. verification) and identities outside the training set
- As adding the number of classes to be predicted, the generalization power of the learned features also improves

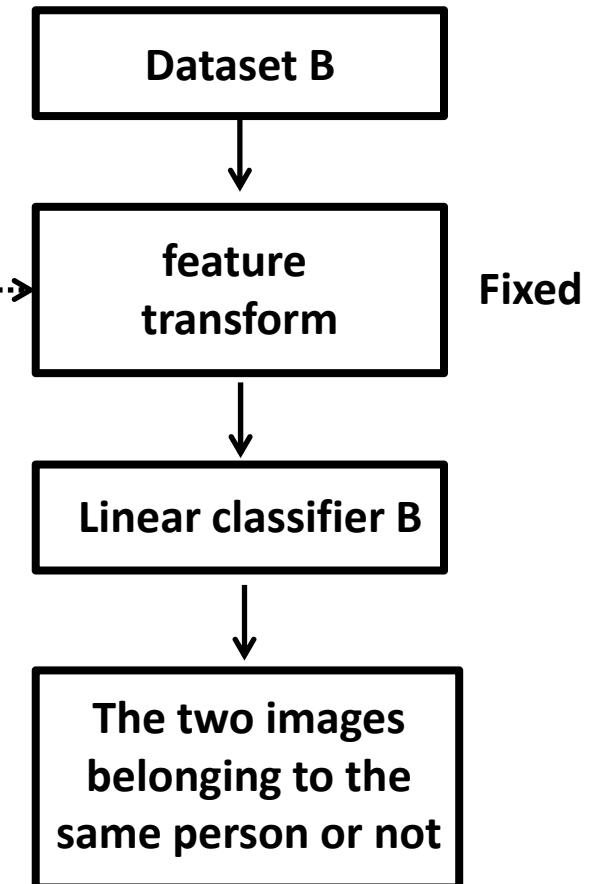


Training stage A



Face identification

Training stage B



Face verification

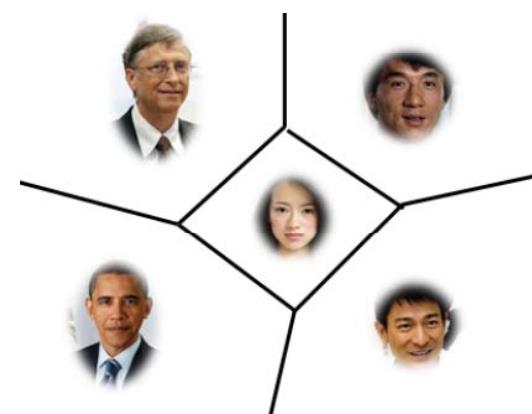
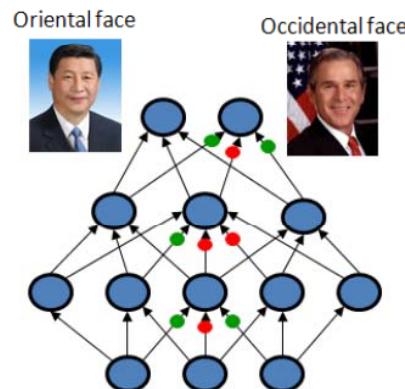
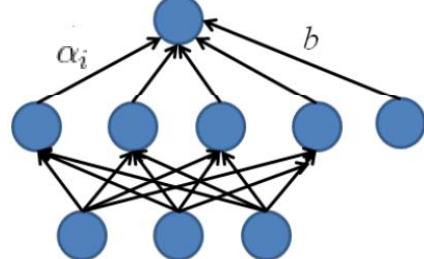
Fixed

Deep Structures vs Shallow Structures (Why deep?)

Shallow Structures

- A three-layer neural network (with one hidden layer) can approximate any classification function
- Most machine learning tools (such as SVM, boosting, and KNN) can be approximated as neural networks with one or two hidden layers
- Shallow models divide the feature space into regions and match templates in local regions. $O(N)$ parameters are needed to represent N regions

SVM $g(x) = b + \sum_i \alpha_i K(x, x_i)$



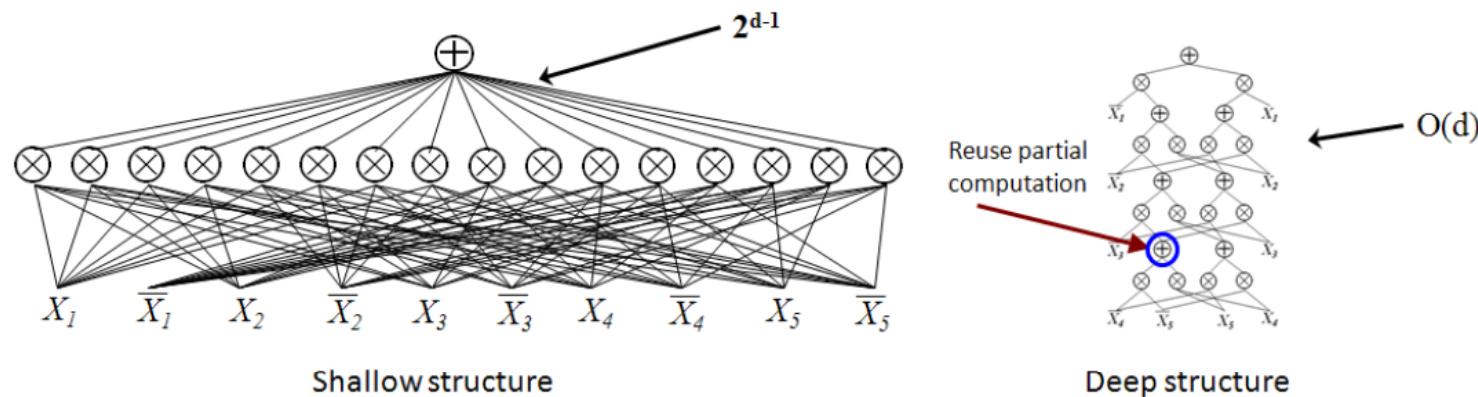
Deep Machines are More Efficient for Representing Certain Classes of Functions

- Theoretical results show that an architecture with insufficient depth can require many more computational elements, potentially exponentially more (with respect to input size), than architectures whose **depth is matched to the task** (Hastad 1986, Hastad and Goldmann 1991)

- Take the d-bit parity function as an example

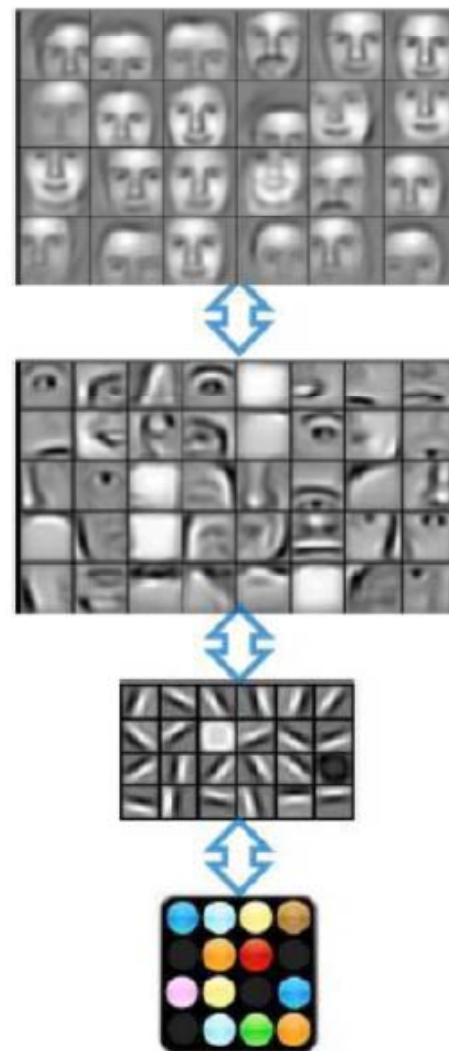
$$(x_1, \dots, x_d) \in \{0, 1\}^d \mapsto \begin{cases} 1, & \text{if } \sum_{i=1}^d x_i \text{ is even} \\ -1, & \text{otherwise} \end{cases}$$

- d-bit logical parity circuits of depth 2 have exponential size (Andrew Yao, 1985)



- There are functions computable with a polynomial-size logic gates circuits of depth k that require exponential size when restricted to depth k - 1 (Hastad, 1986)

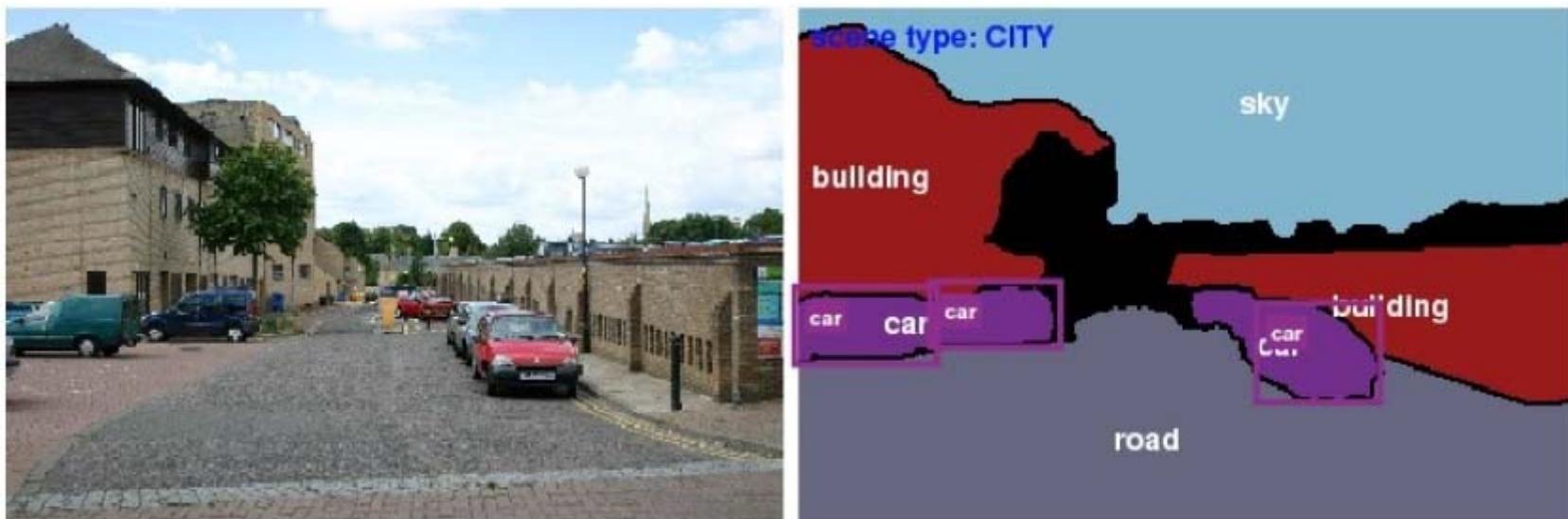
- Architectures with multiple levels naturally provide sharing and re-use of components



Honglak Lee, NIPS'10

Humans Understand the World through Multiple Levels of Abstractions

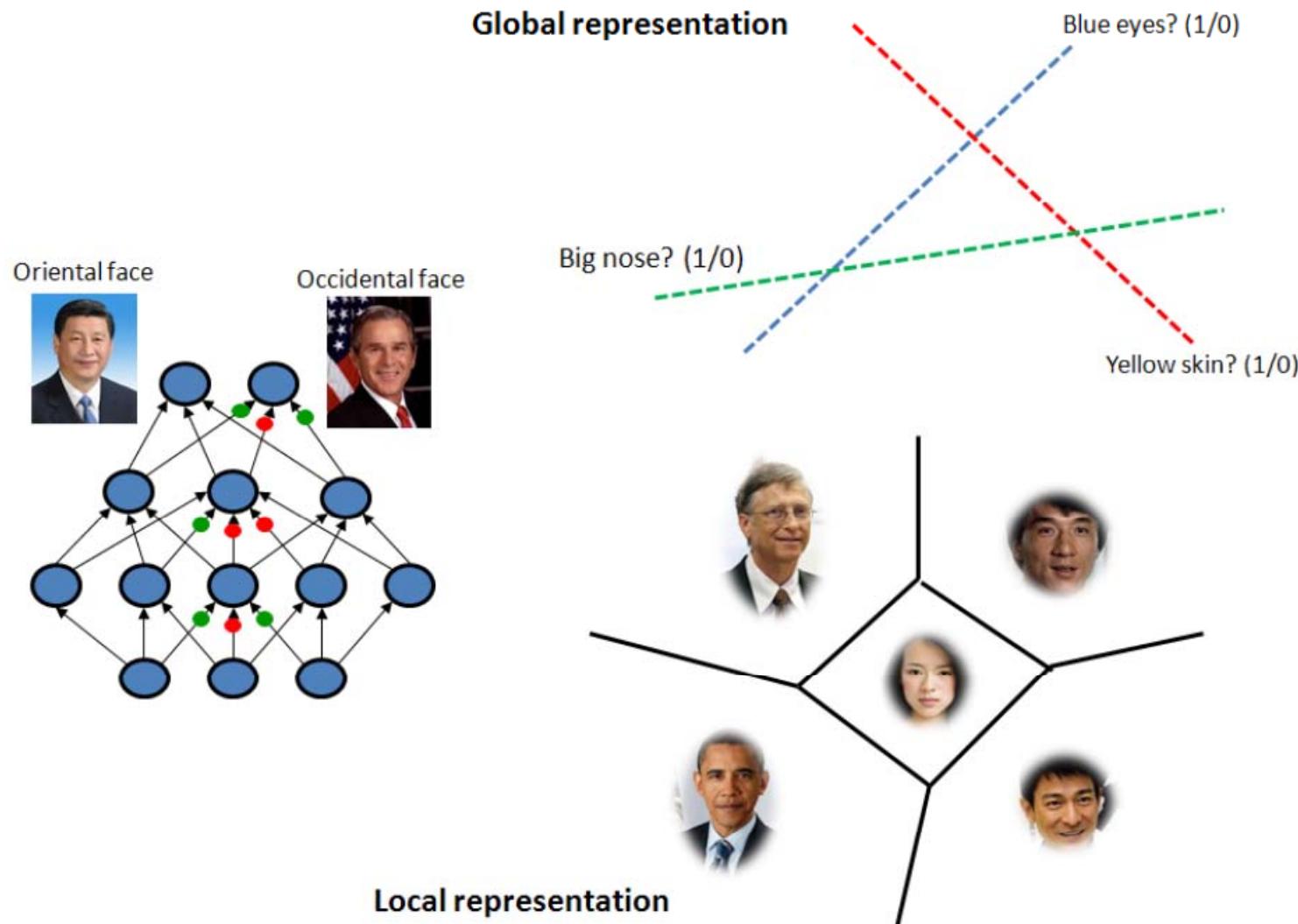
- We do not interpret a scene image with pixels
 - Objects (sky, cars, roads, buildings, pedestrians) -> parts (wheels, doors, heads) -> texture -> edges -> pixels
 - Attributes: blue sky, red car
- It is natural for humans to decompose a complex problem into sub-problems through multiple levels of representations



Humans Understand the World through Multiple Levels of Abstractions

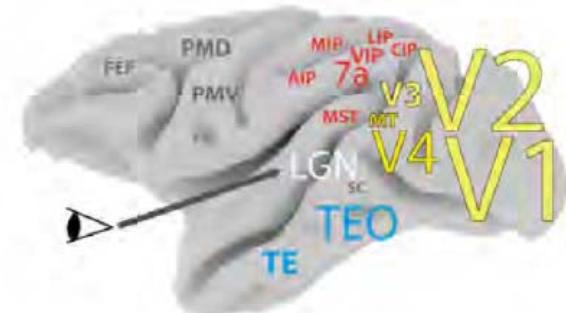
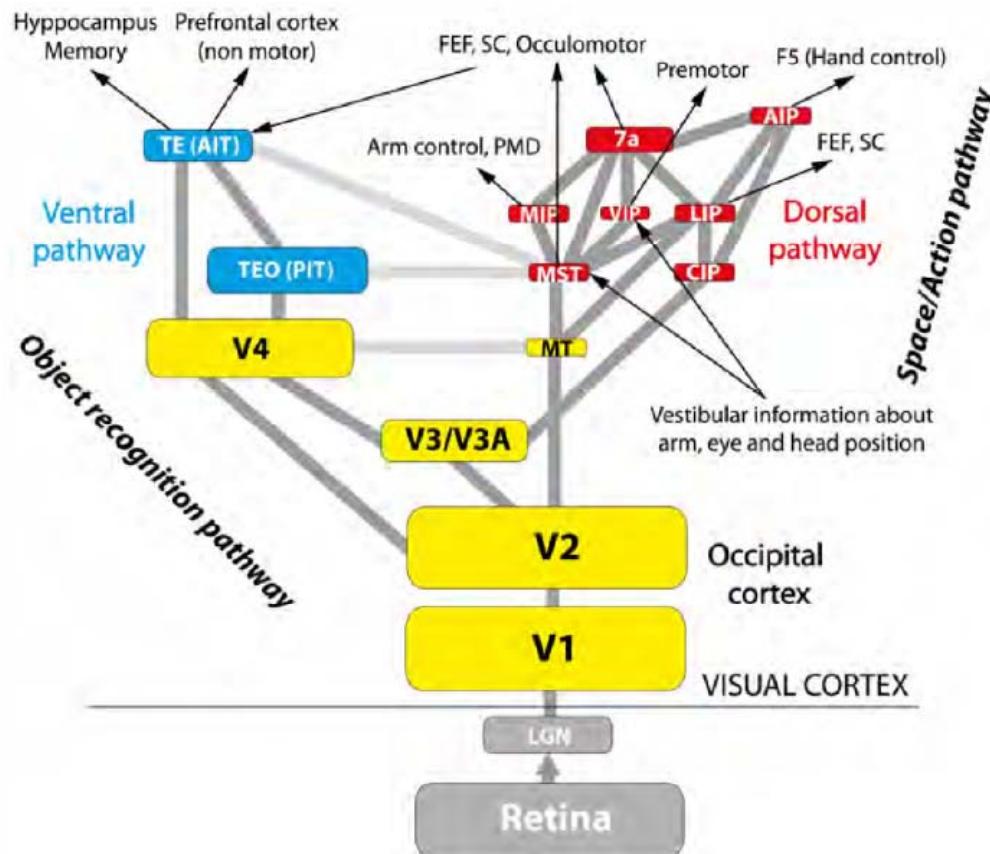
- Humans learn abstract concepts on top of less abstract ones
- Humans can imagine new pictures by re-configuring these abstractions at multiple levels. Thus our brain has good generalization can recognize things never seen before.
 - Our brain can estimate shape, lighting and pose from a face image and generate new images under various lightings and poses. That's why we have good face recognition capability.

Local and Global Representations



Human Brains Process Visual Signals through Multiple Layers

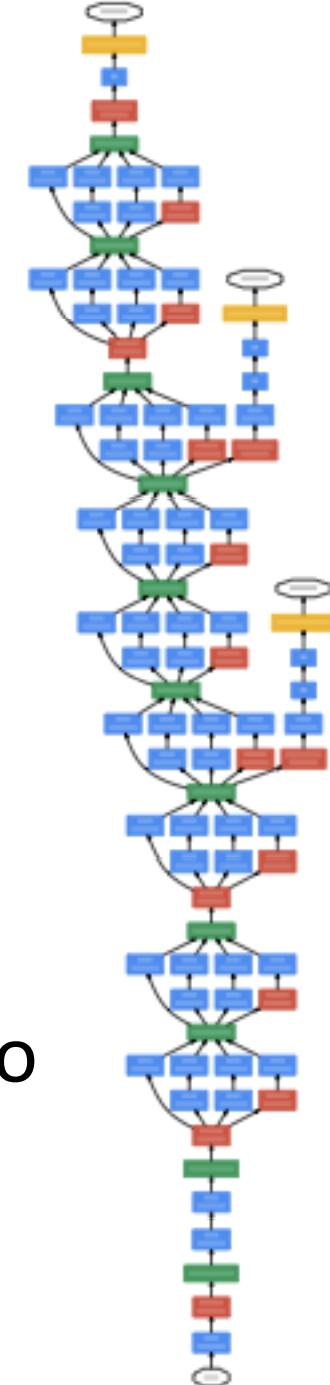
- A visual cortical area consists of six layers (Kruger et al. 2013)



GoogLeNet

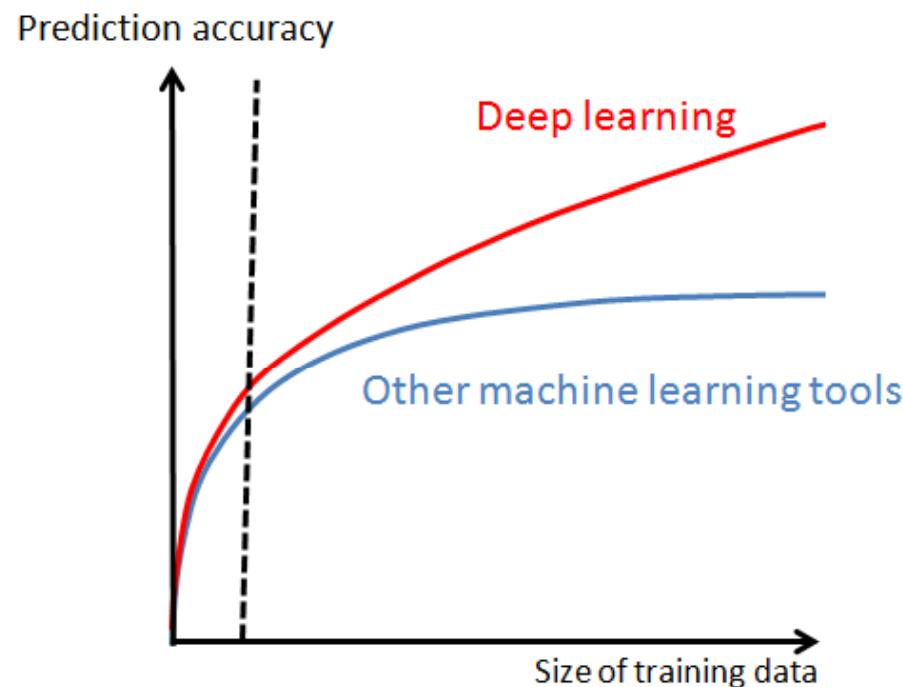
(Very Deep Neural Network)

- More than 20 layers
- The learning capacity is largely increased
- Add supervision at multiple layers
- The training error drops much more quickly
- The error rate is reduced from 15.3% to 6.6%



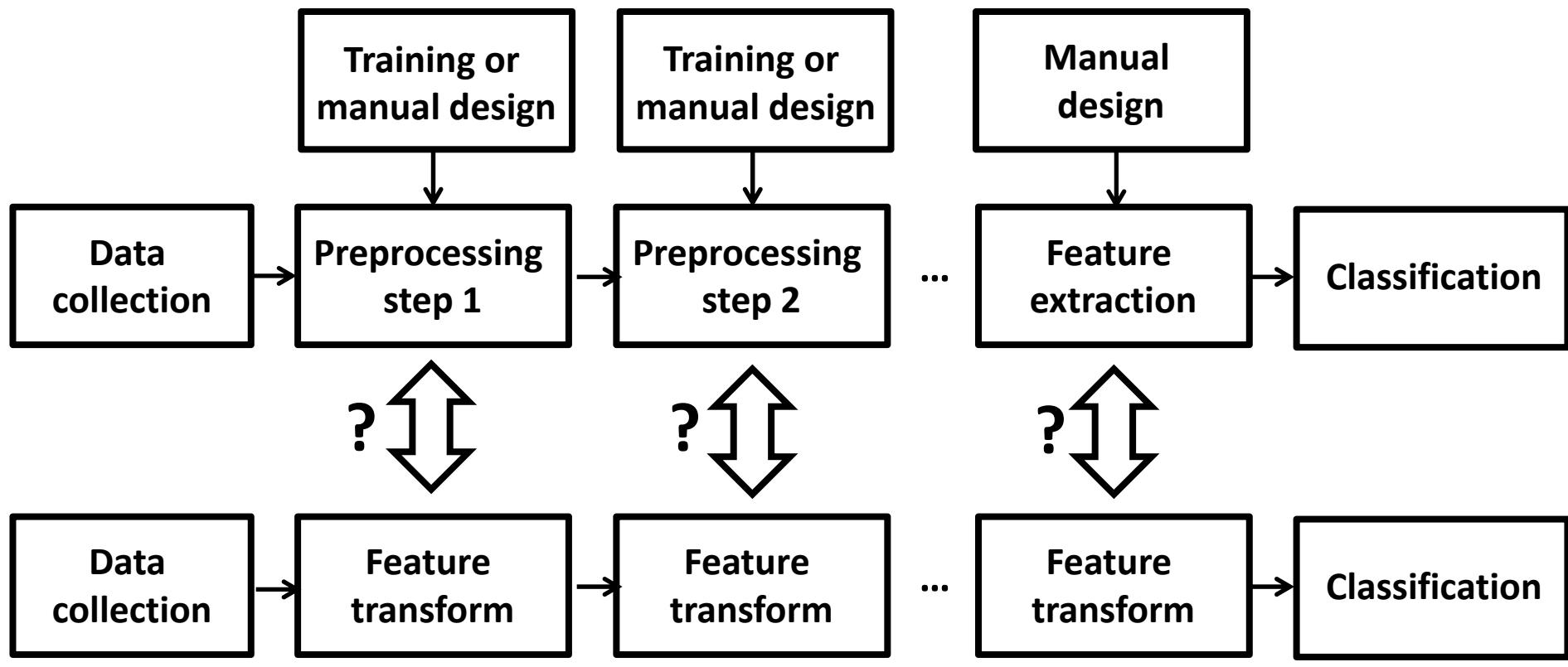
How do shallow models increase the model capacity?

- Typically increase the size of feature vectors



D. Chen, X. Cao, F. Wen, and J. Sun. Blessing of dimensionality: Highdimensional feature and its efficient compression for face verification. In Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, 2013.

Joint Learning vs Separate Learning

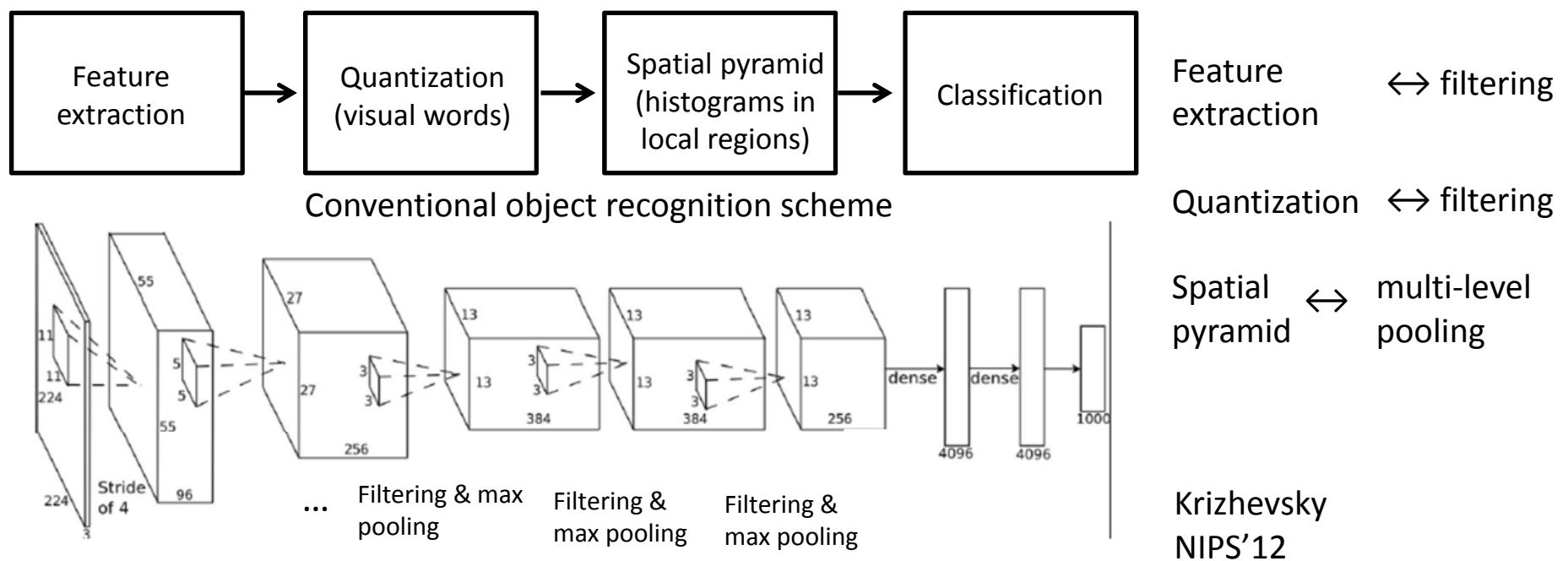


End-to-end learning

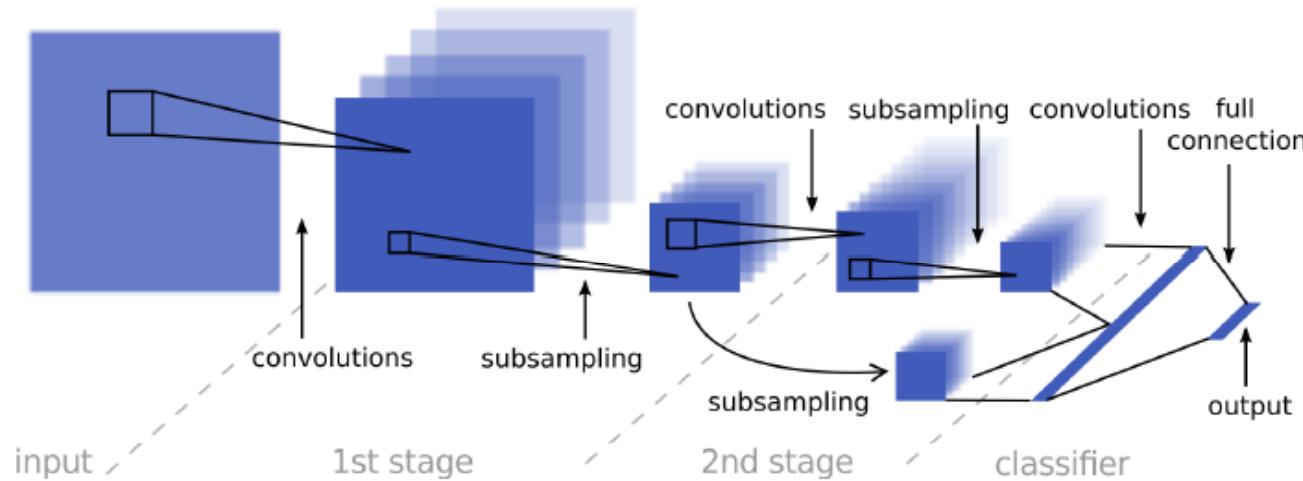
Deep learning is a framework/language but not a black-box model

Its power comes from joint optimization and
increasing the capacity of the learner

- Domain knowledge could be helpful for designing new deep models and training strategies
- How to formulate a vision problem with deep learning?
 - Make use of experience and insights obtained in CV research
 - Sequential design/learning vs **joint learning**
 - Effectively train a deep model (layerwise pre-training + fine tuning)

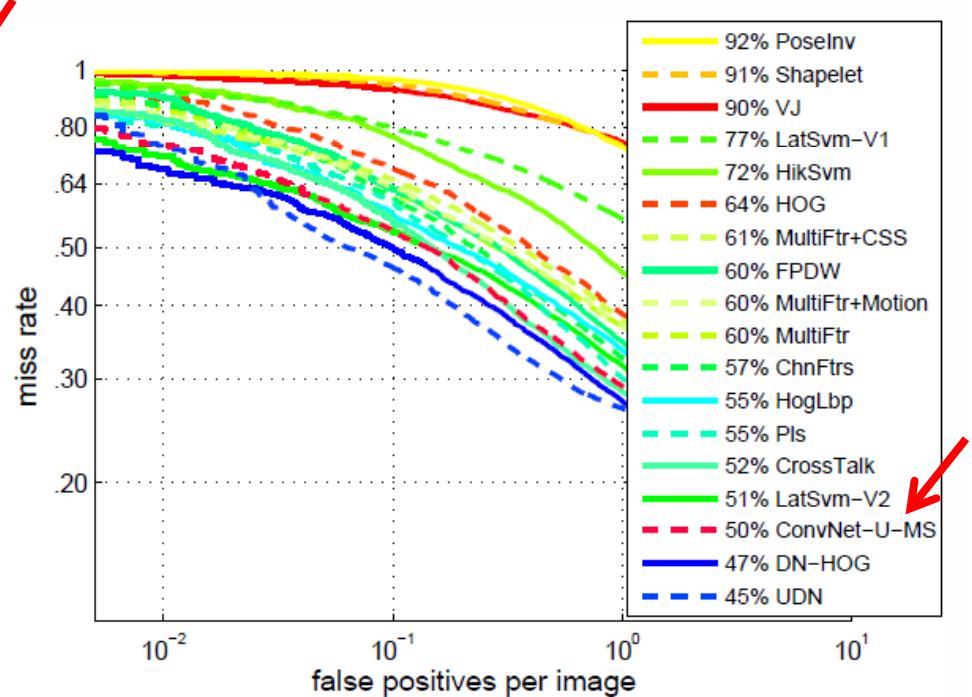
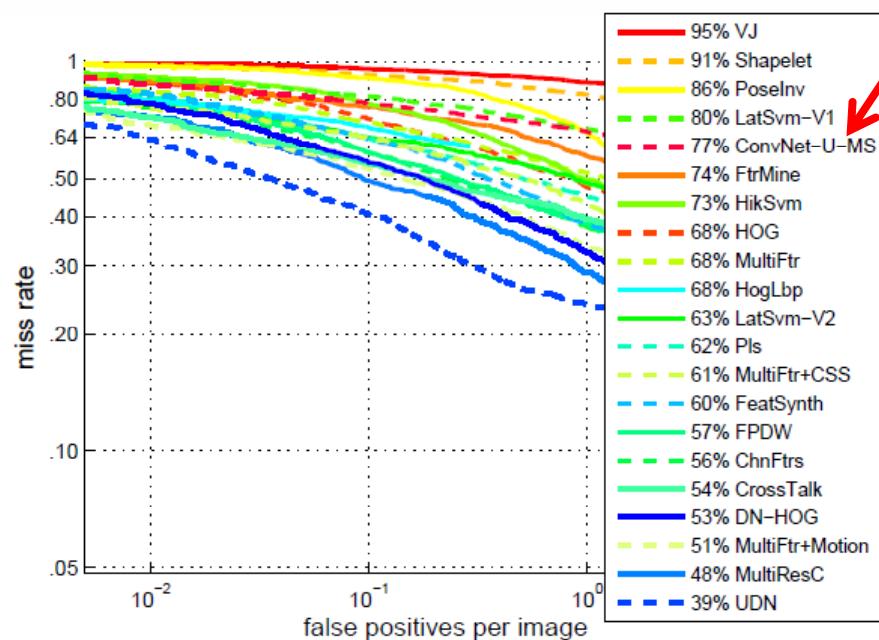


What if we treat an existing deep model as a black box in pedestrian detection?

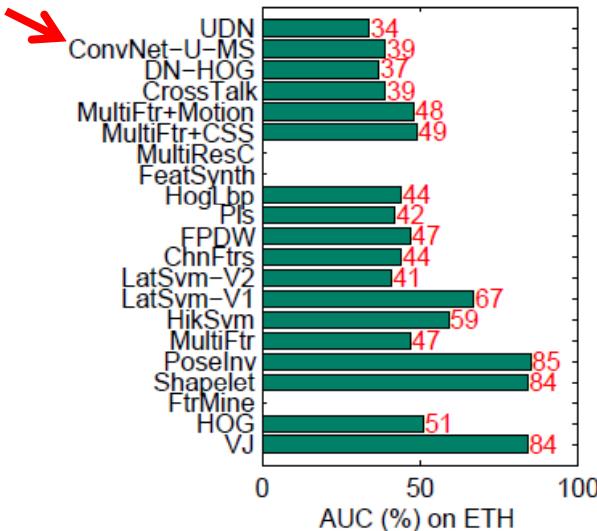


ConvNet-U-MS

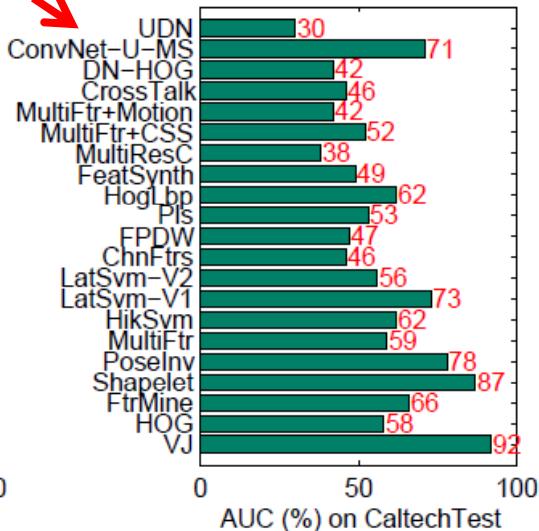
- Sermnet, K. Kavukcuoglu, S. Chintala, and LeCun, “Pedestrian Detection with Unsupervised Multi-Stage Feature Learning,” CVPR 2013.

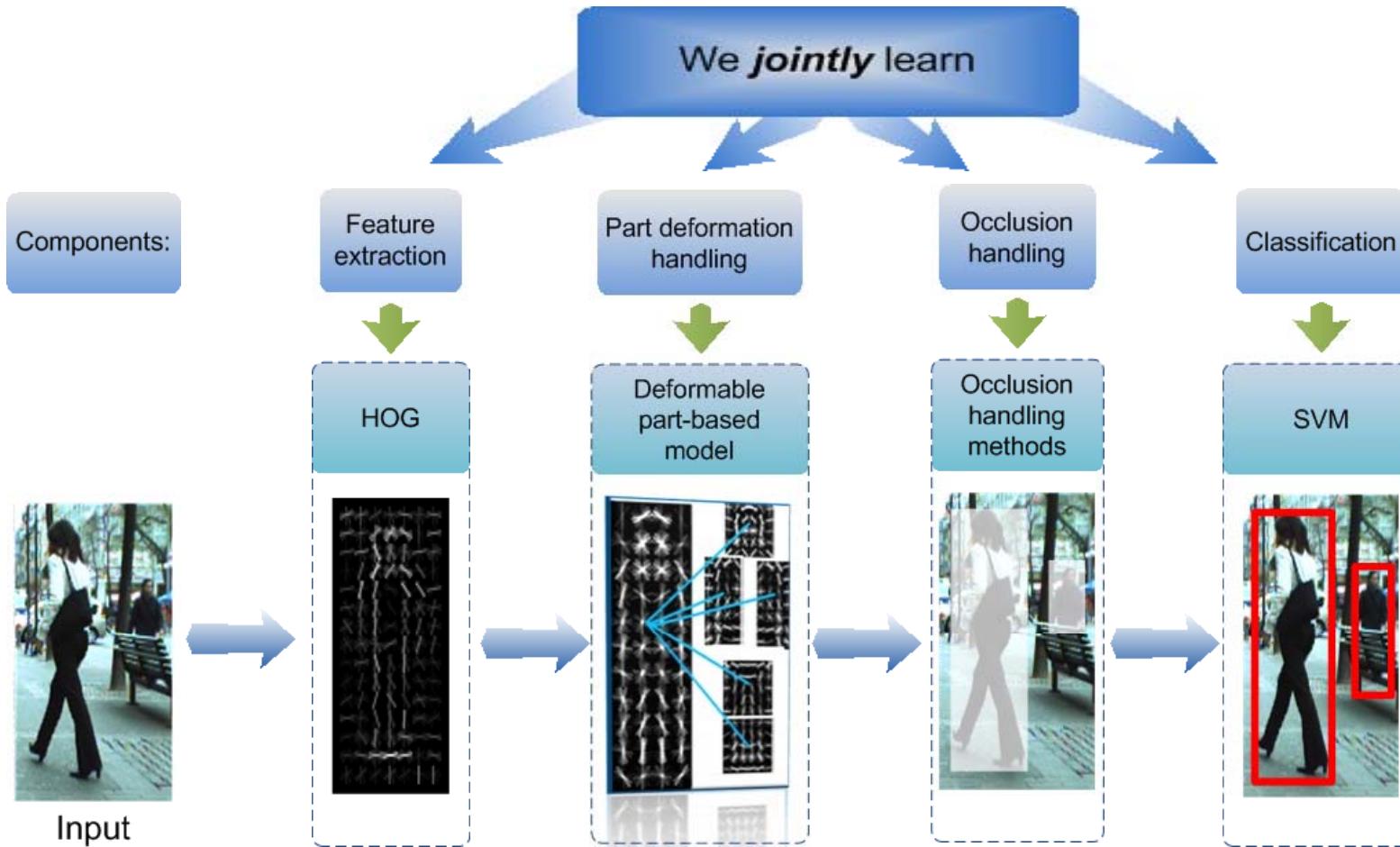


Results on Caltech Test



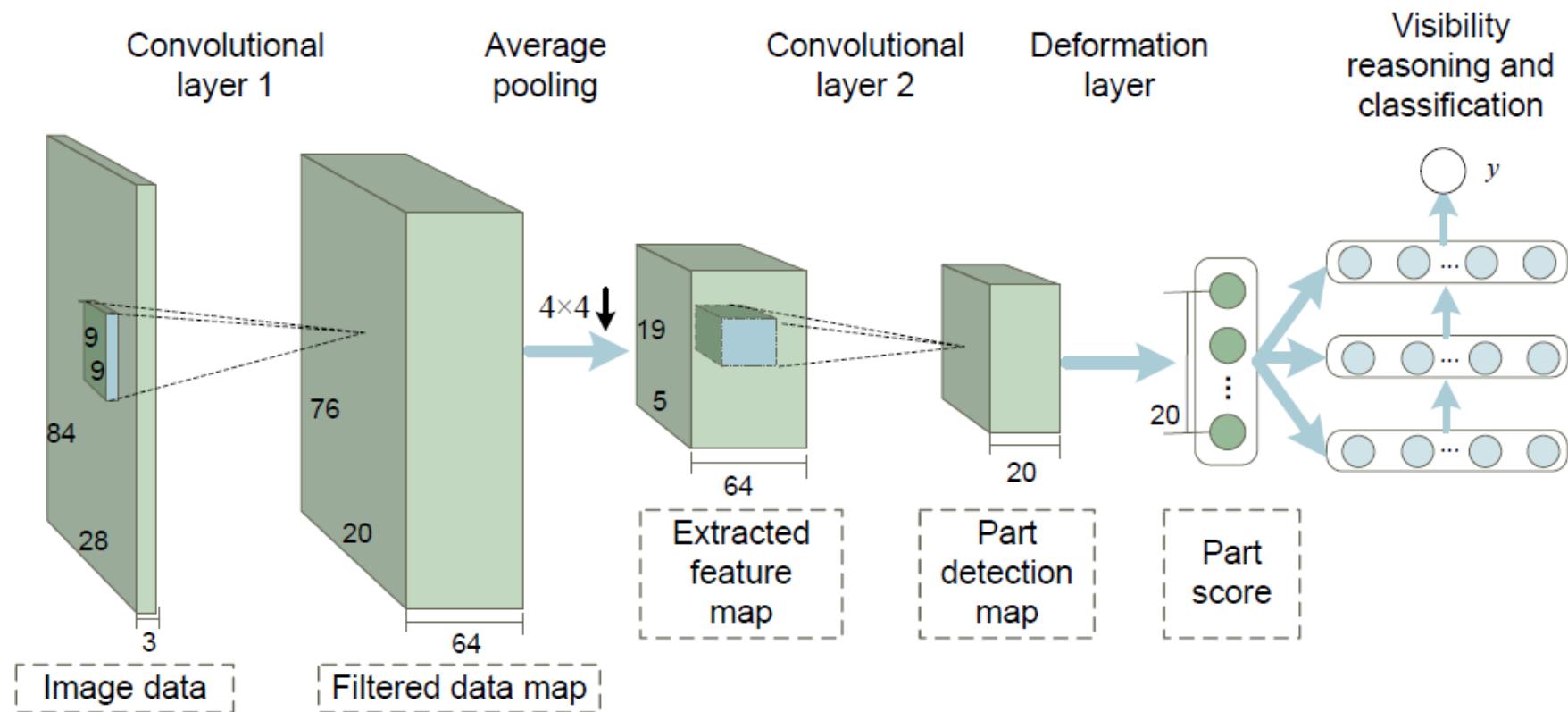
Results on ETHZ





- N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. CVPR, 2005. (6000 citations)
- P. Felzenszwalb, D. McAllester, and D. Ramanan. A Discriminatively Trained, Multiscale, Deformable Part Model. CVPR, 2008. (2000 citations)
- W. Ouyang and X. Wang. A Discriminative Deep Model for Pedestrian Detection with Occlusion Handling. CVPR, 2012.

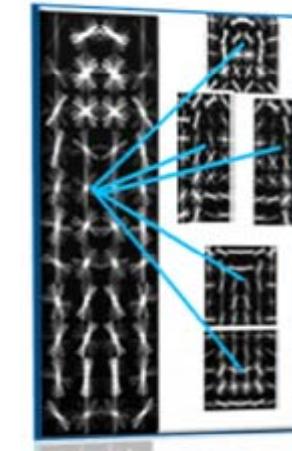
Our Joint Deep Learning Model



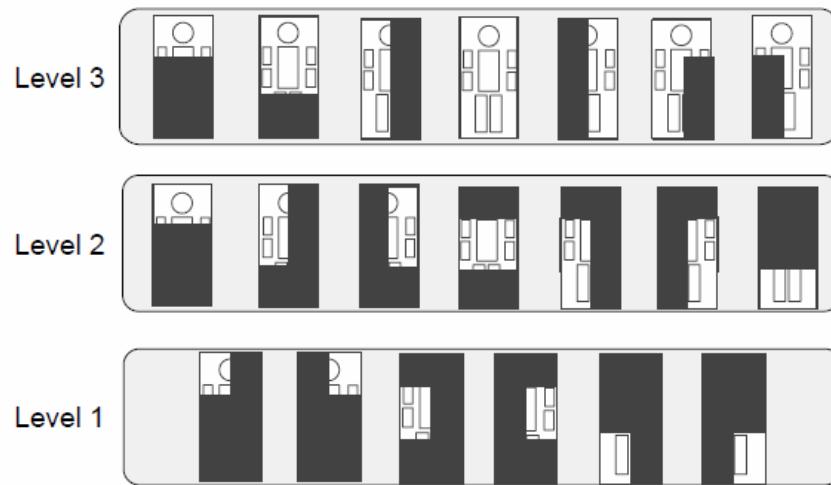
W. Ouyang and X. Wang, "Joint Deep Learning for Pedestrian Detection," Proc. ICCV, 2013.

Modeling Part Detectors

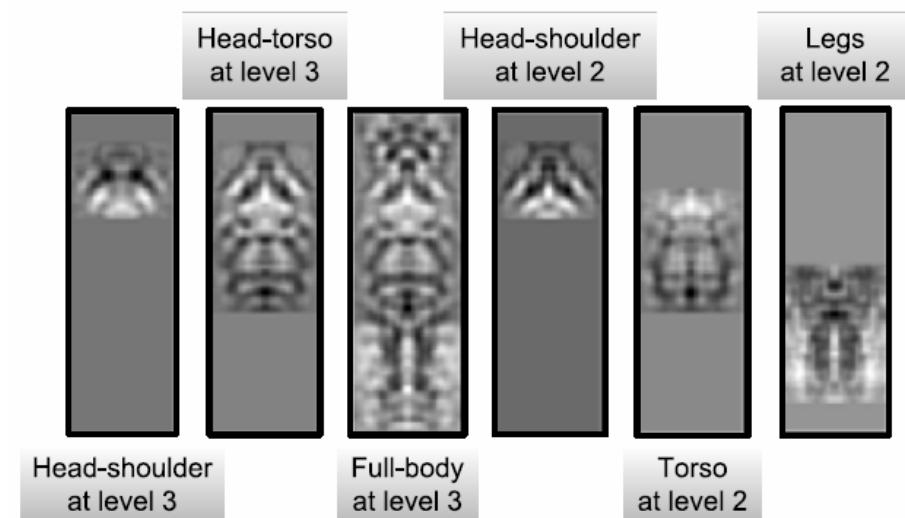
- Design the filters in the second convolutional layer with variable sizes



Part models learned
from HOG

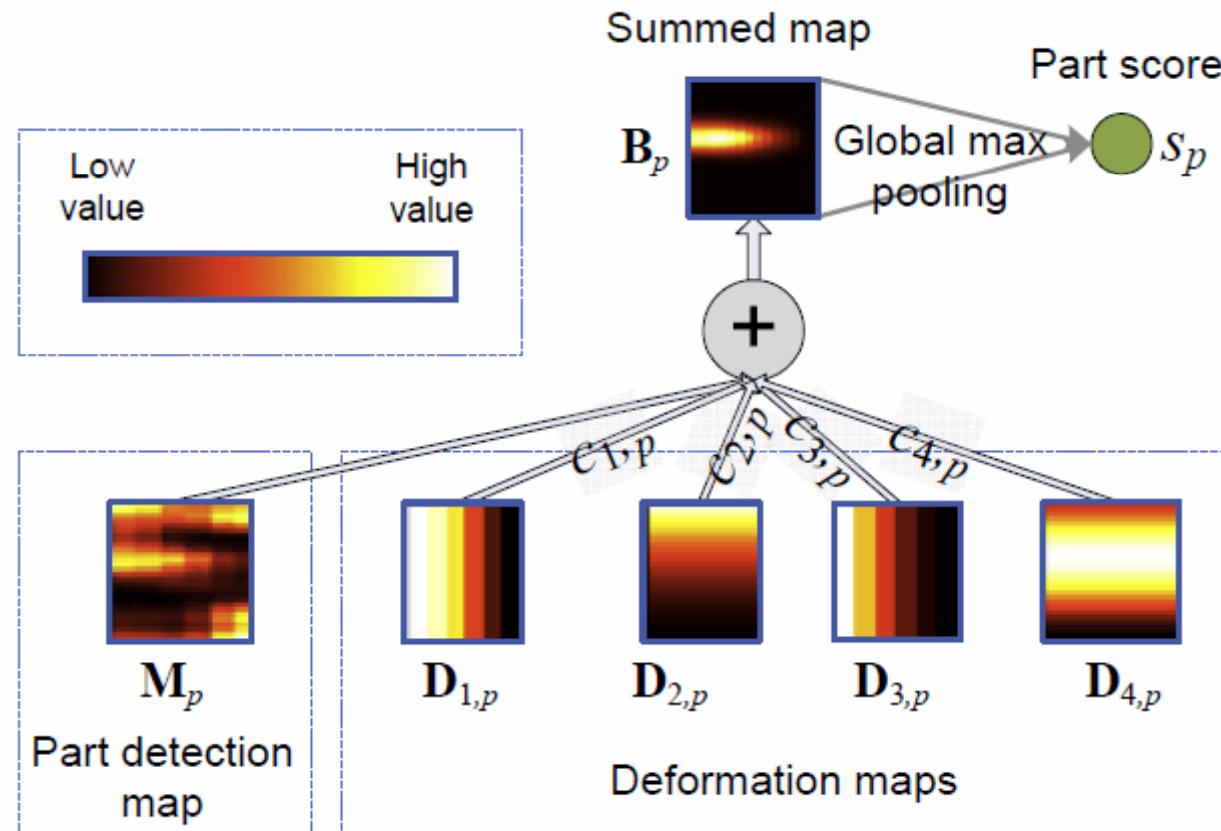


Part models

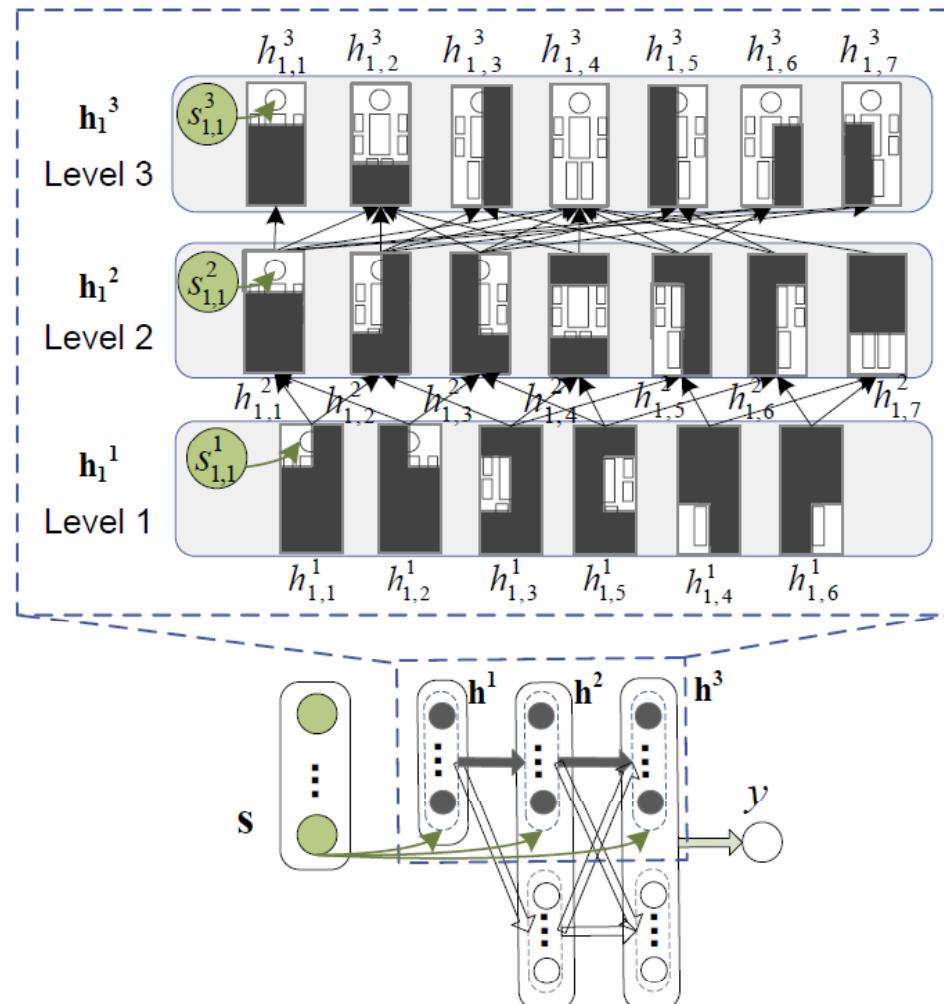


Learned filtered at the second
convolutional layer

Deformation Layer



Visibility Reasoning with Deep Belief Net

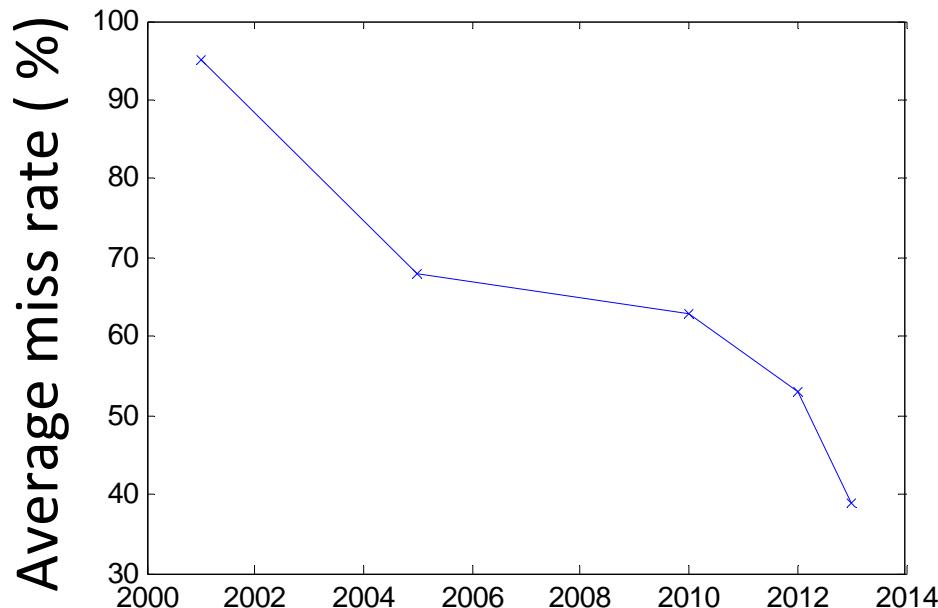


$$\tilde{h}_j^{l+1} = \sigma(\tilde{\mathbf{h}}^{l\top} \mathbf{w}_{*,j}^l + c_j^{l+1} + g_j^{l+1} s_j^{l+1})$$

— Correlates with part detection score

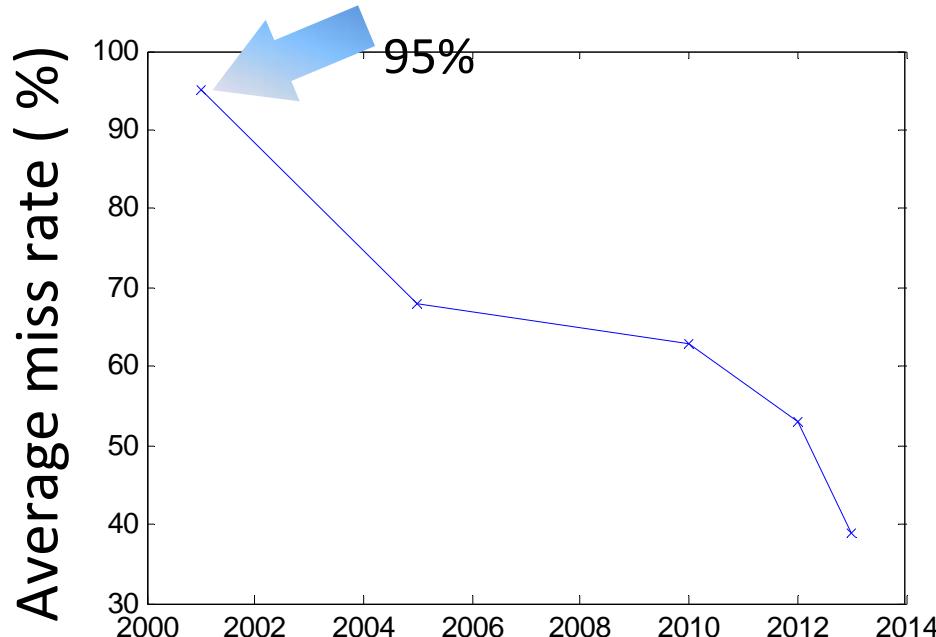
Experimental Results

- Caltech – Test dataset (largest, most widely used)



Experimental Results

- Caltech – Test dataset (largest, most widely used)



Rapid object detection using a boosted cascade of simple features

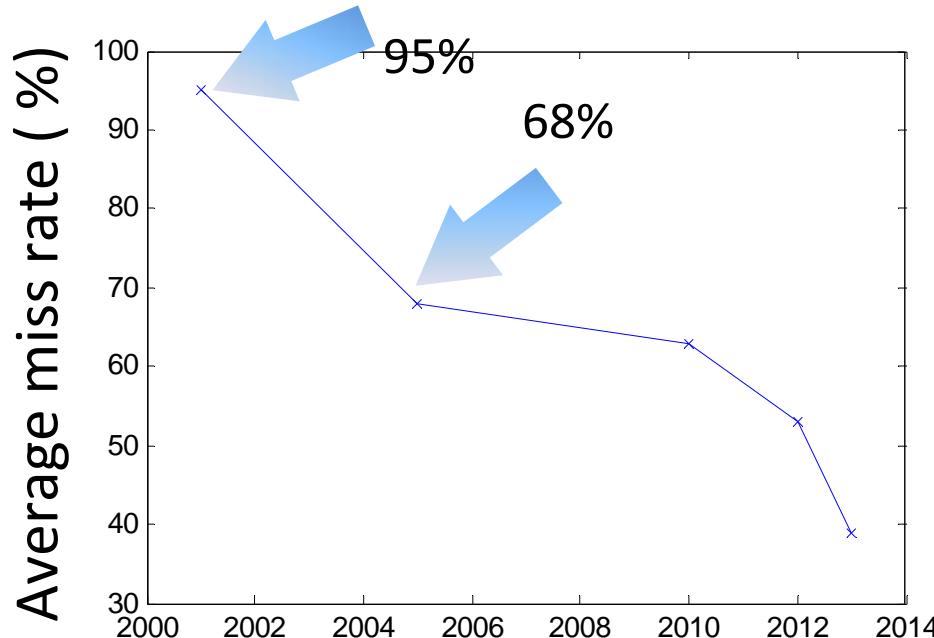
P Viola, M Jones - ... Vision and Pattern Recognition, 2001. CVPR ..., 2001 - ieeexplore.ieee.org.org

Abstract This paper describes a machine learning approach for visual **object detection** which is capable of processing images extremely rapidly and achieving high **detection** rates. This work is distinguished by three key contributions. The first is the introduction of a new ...

Cited by 7647 Related articles All 201 versions Import into BibTeX More▼

Experimental Results

- Caltech – Test dataset (largest, most widely used)



[Histograms of oriented gradients for human detection](#)

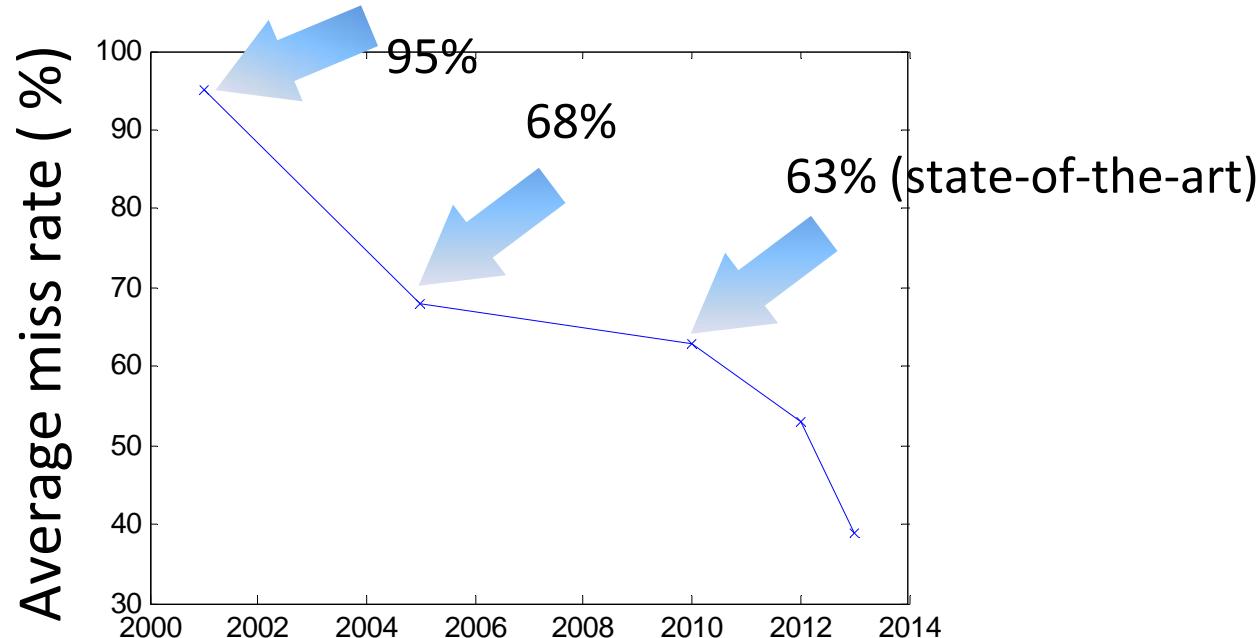
[N Dalal, B Triggs - ... and Pattern Recognition, 2005. CVPR 2005 ...](#), 2005 - ieeexplore.ieee.org

... We study the issue of feature sets for **human detection**, showing that locally normalized **Histogram of Oriented Gradient** (HOG) descriptors provide excellent performance relative to other existing feature sets including wavelets [17,22]. ...

Cited by 5438 Related articles All 106 versions Import into BibTeX More ▾

Experimental Results

- Caltech – Test dataset (largest, most widely used)



Object detection with discriminatively trained part-based models

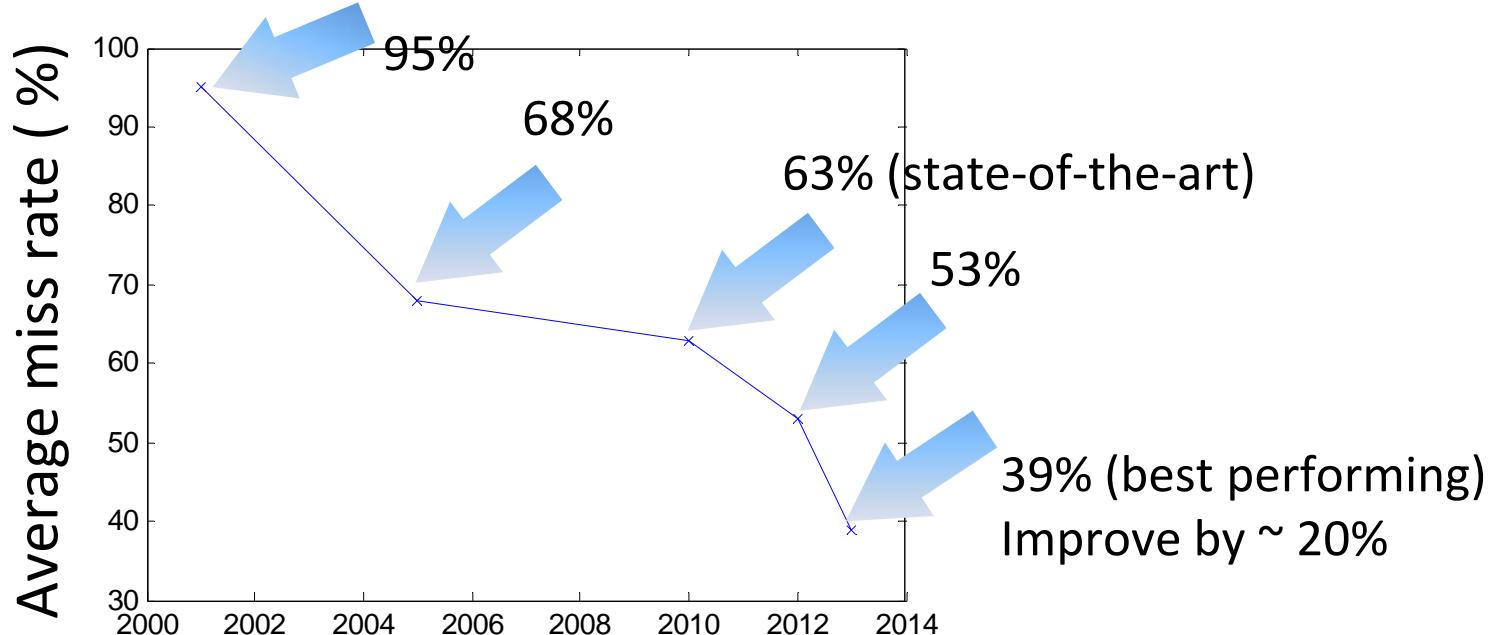
[PF Felzenszwalb, RB Girshick...](#) - Pattern Analysis and ..., 2010 - ieeexplore.ieee.org

Abstract We describe an **object detection** system **based** on mixtures of multiscale deformable **part models**. Our system is able to represent highly variable **object** classes and achieves state-of-the-art results in the PASCAL **object detection** challenges. While ...

Cited by 964 Related articles All 43 versions Import into BibTeX More ▾

Experimental Results

- Caltech – Test dataset (largest, most widely used)



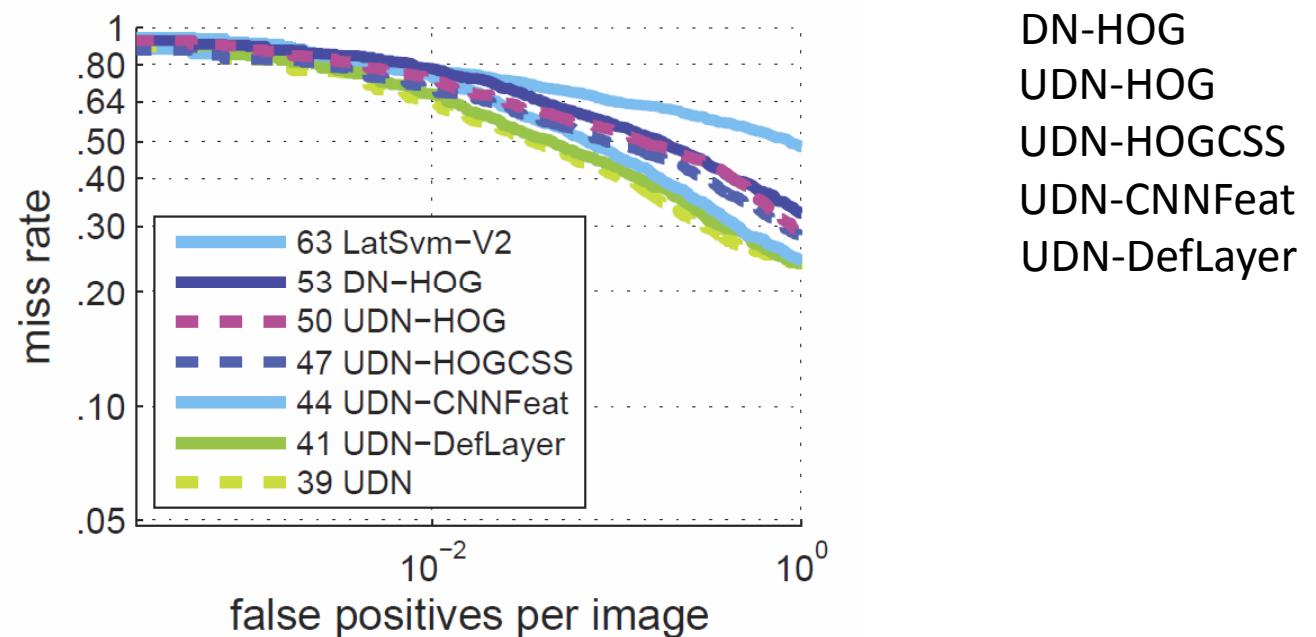
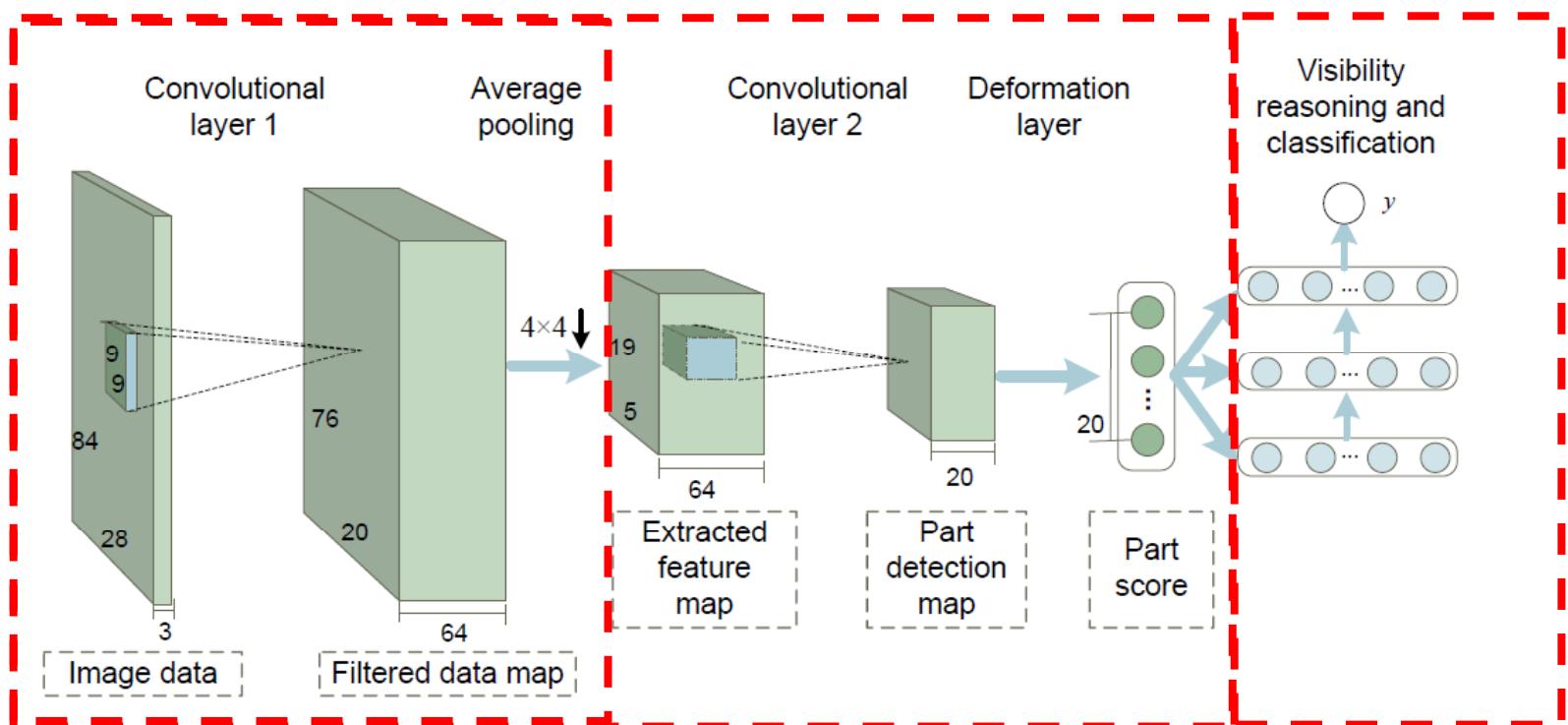
W. Ouyang and X. Wang, "A Discriminative Deep Model for Pedestrian Detection with Occlusion Handling," CVPR 2012.

W. Ouyang, X. Zeng and X. Wang, "Modeling Mutual Visibility Relationship in Pedestrian Detection ", CVPR 2013.

W. Ouyang, Xiaogang Wang, "Single-Pedestrian Detection aided by Multi-pedestrian Detection ", CVPR 2013.

X. Zeng, W. Ouyang and X. Wang, " A Cascaded Deep Learning Architecture for Pedestrian Detection," ICCV 2013.

W. Ouyang and Xiaogang Wang, "Joint Deep Learning for Pedestrian Detection," IEEE ICCV 2013.

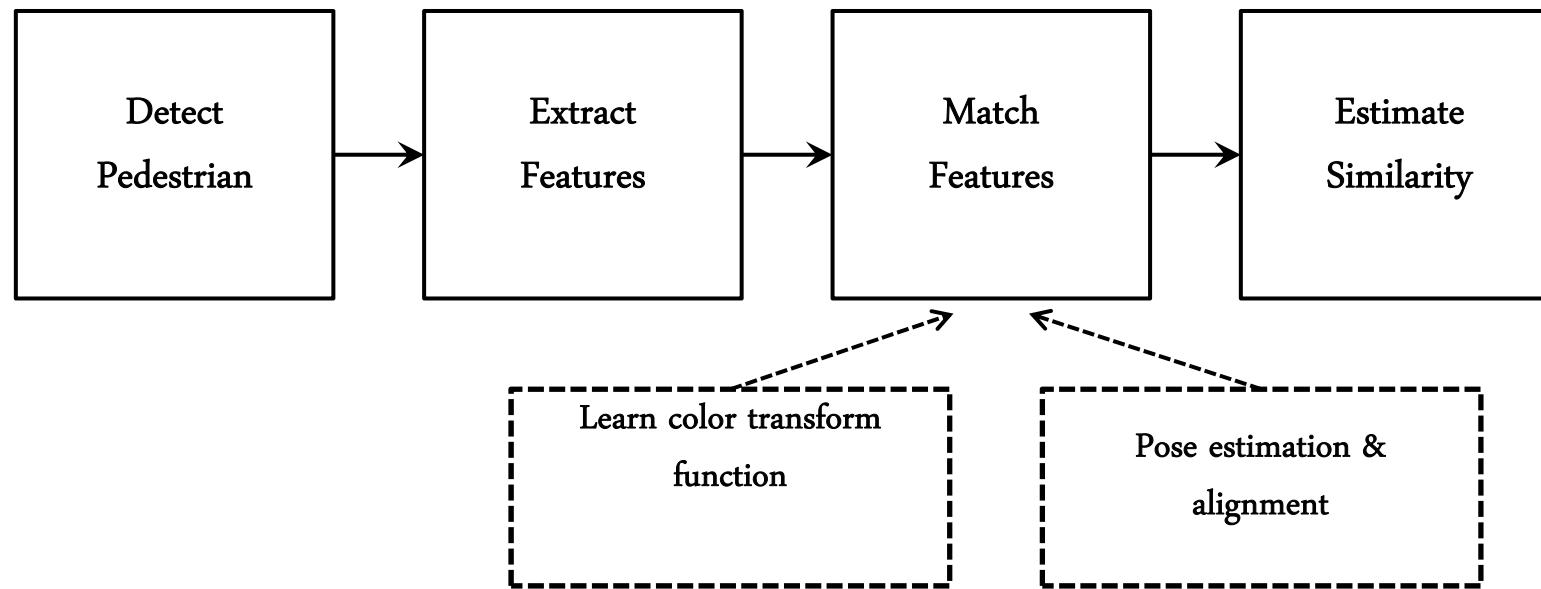


Joint Deep Learning for Person Re-Identification

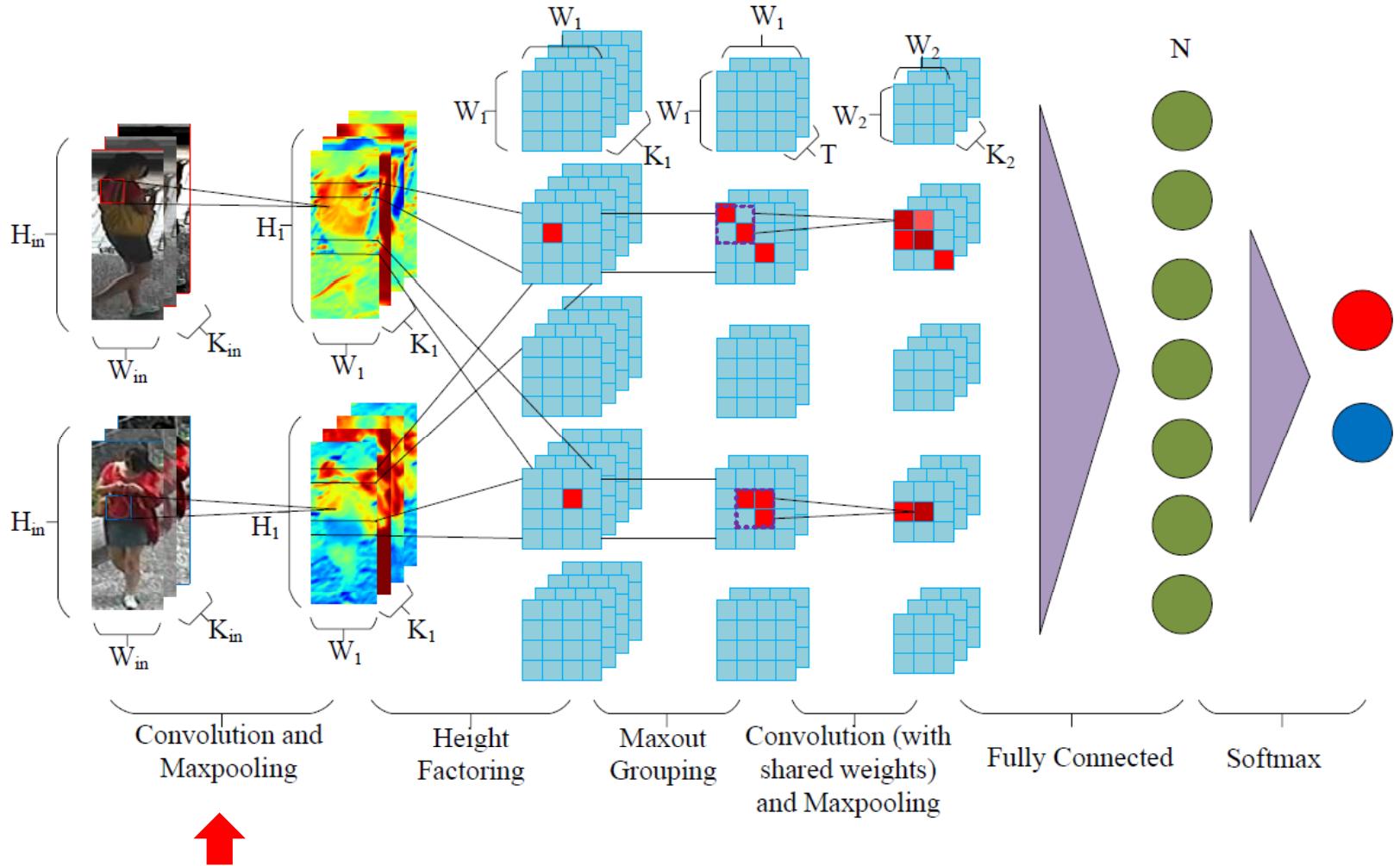
- Challenges
 - Photometric transforms (lighting, camera settings)
 - Geometric transforms (poses, views)
 - Occlusions
 - Background clutters



Pipeline of Existing Systems



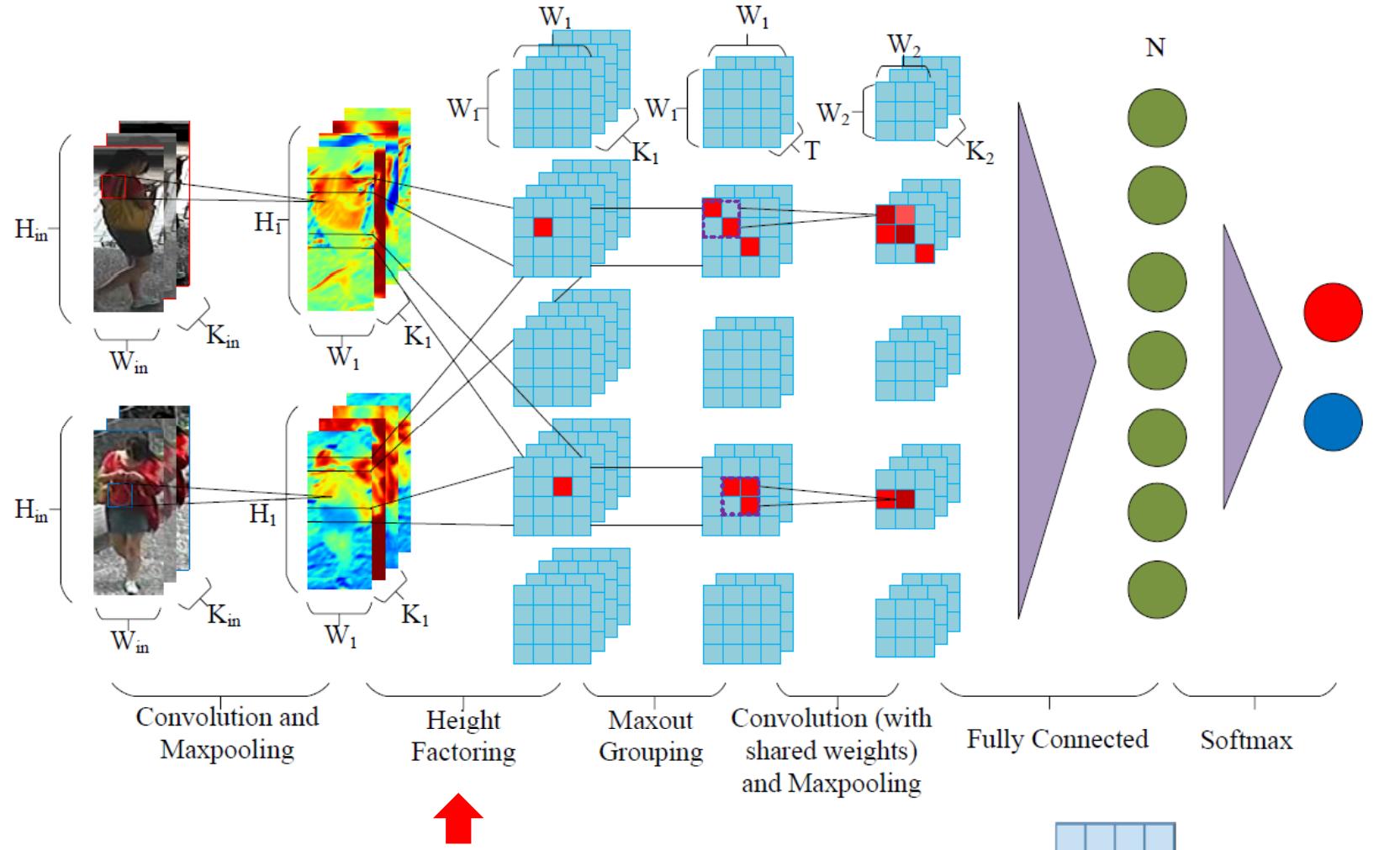
Optimize each module separately



Filter pairing: filters $(\mathbf{W}_k, \mathbf{V}_k)$ applied to different camera views are paired and their difference reflects the photometric transforms

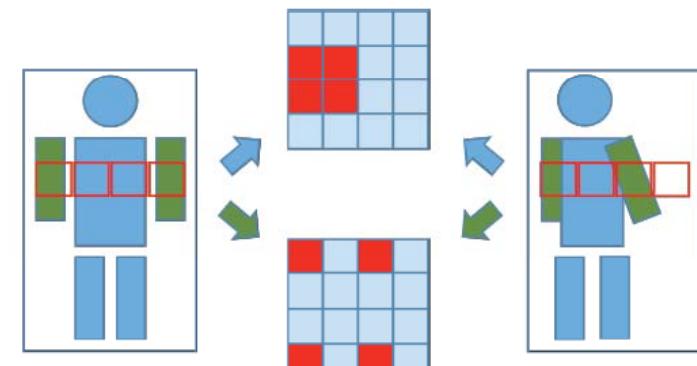
$$f_{ij}^k = \sigma((\mathbf{W}_k * \mathbf{I})_{ij} + b_k^I)$$

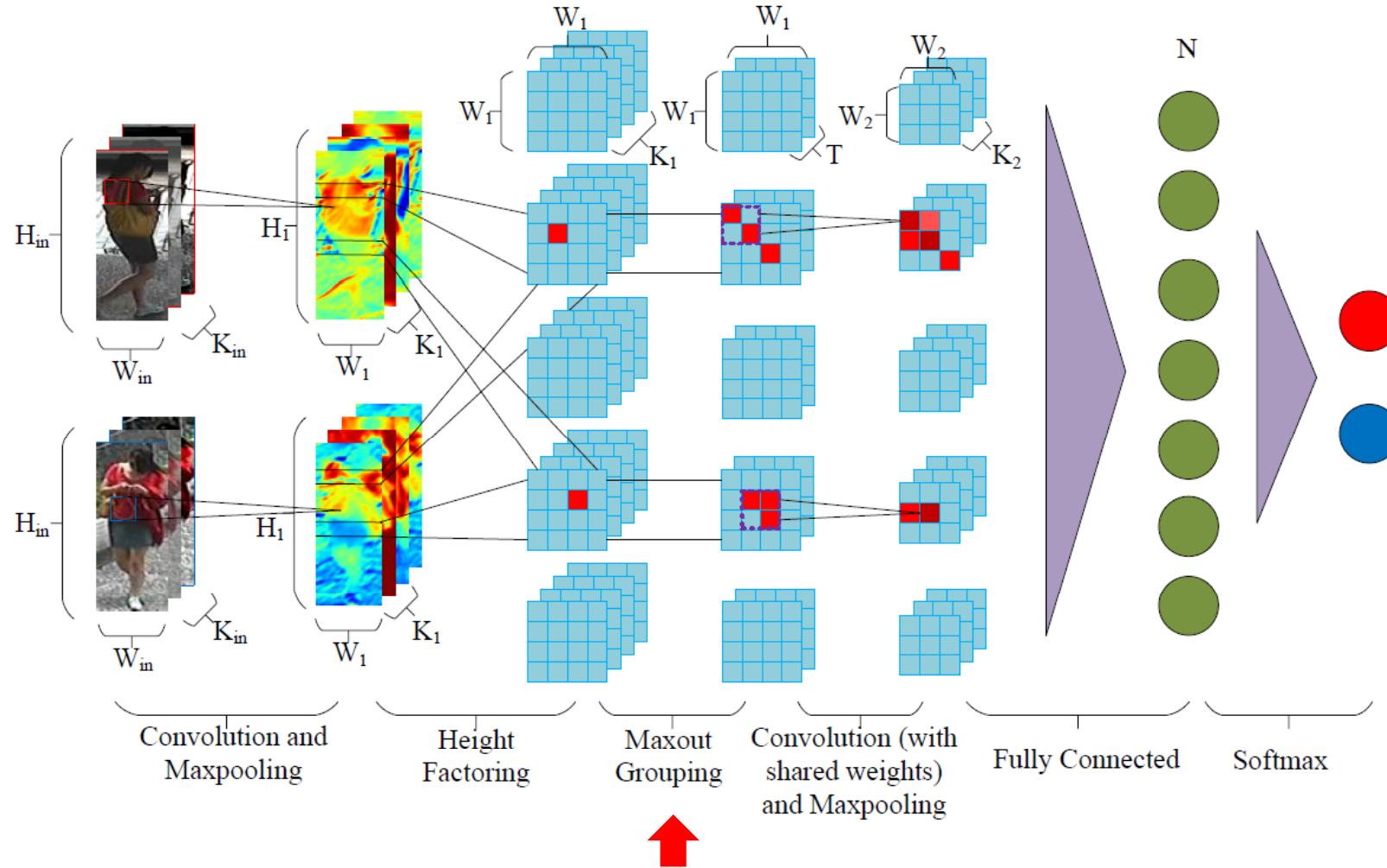
$$g_{ij}^k = \sigma((\mathbf{V}_k * \mathbf{J})_{ij} + b_k^J)$$



Patch matching layer:
 match patches in the same horizontal stripe
 K filters generate K displacement matrices for each stripe

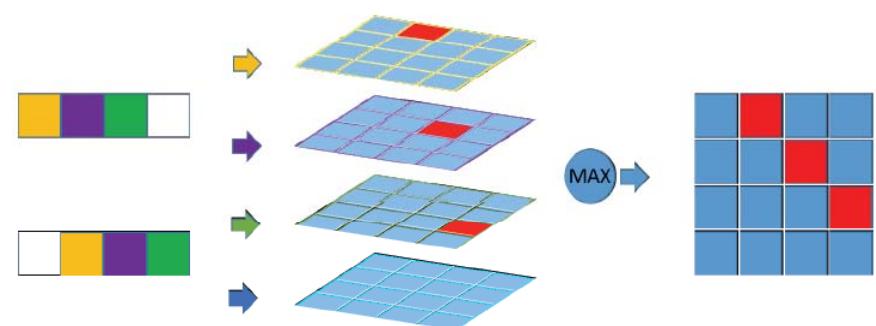
$$S_{(i,j)(i',j')}^k = f_{ij}^k g_{i'j'}^k$$

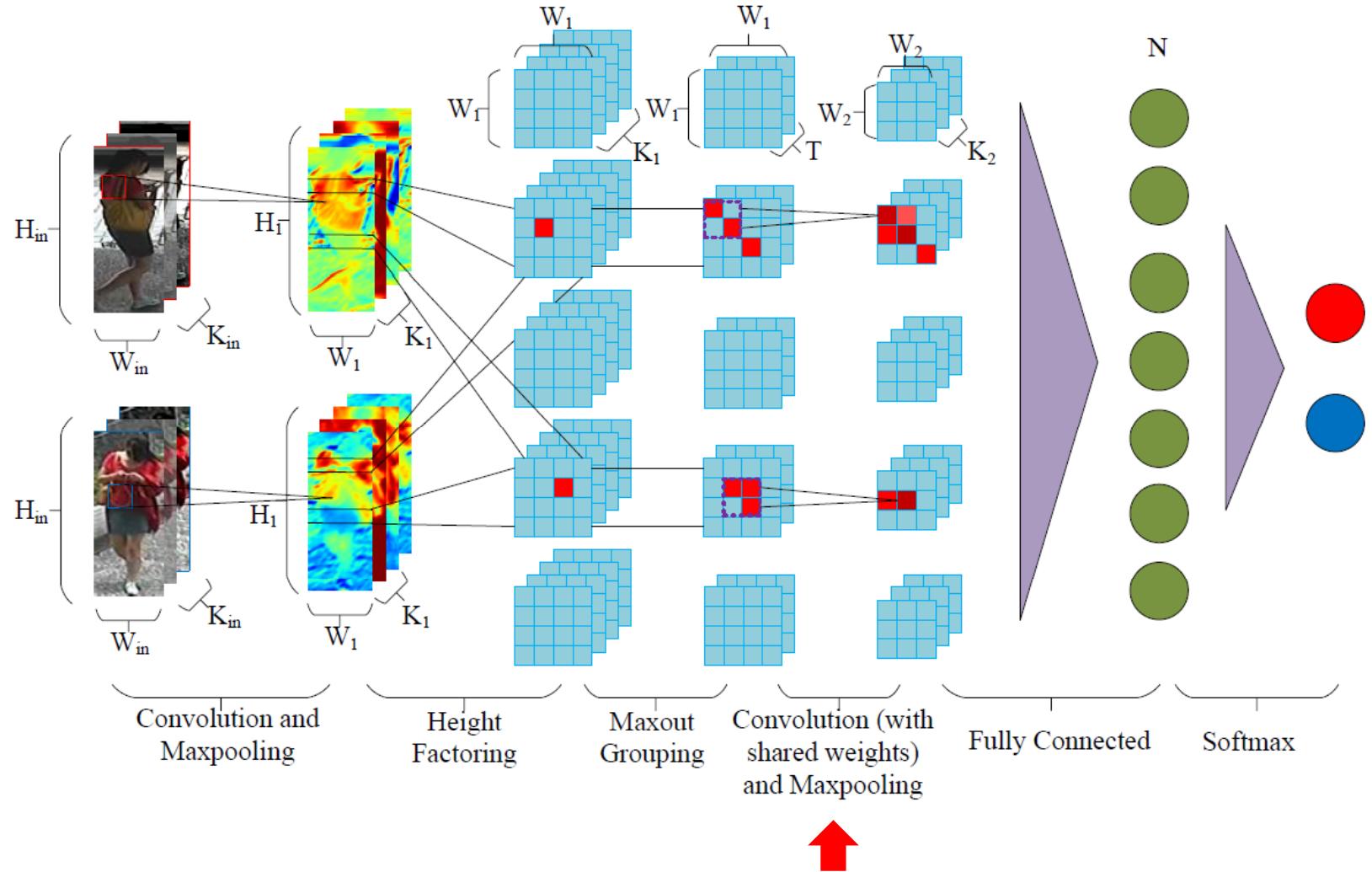




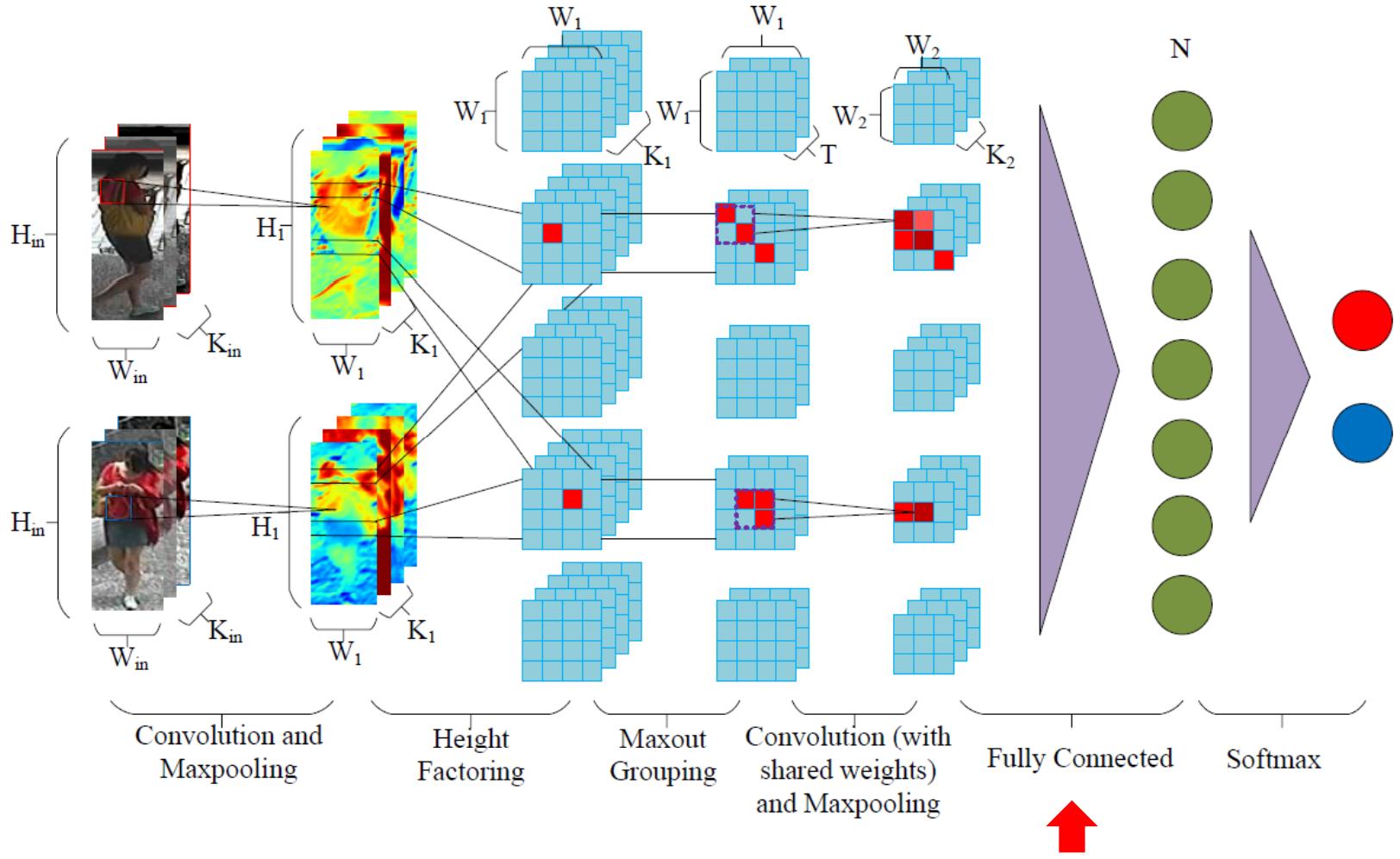
Maxout-grouping layer:

Each feature is represented by multiple channels
 Model a mixture of photometric transforms
 Robust to misdetection of patch matching





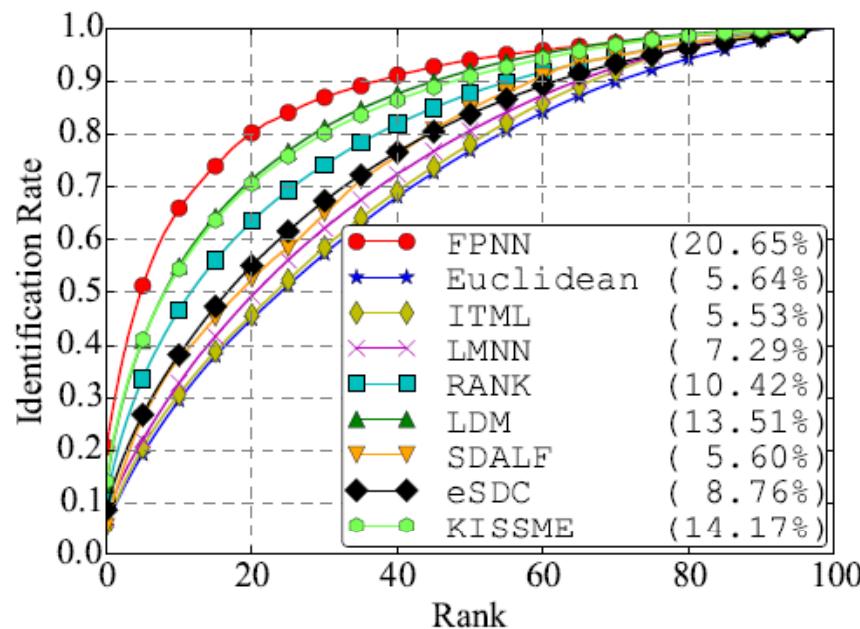
Through another **convolution and max-pooling layer**, the learned filters are applied to displacement matrices and capture local patterns of part displacements



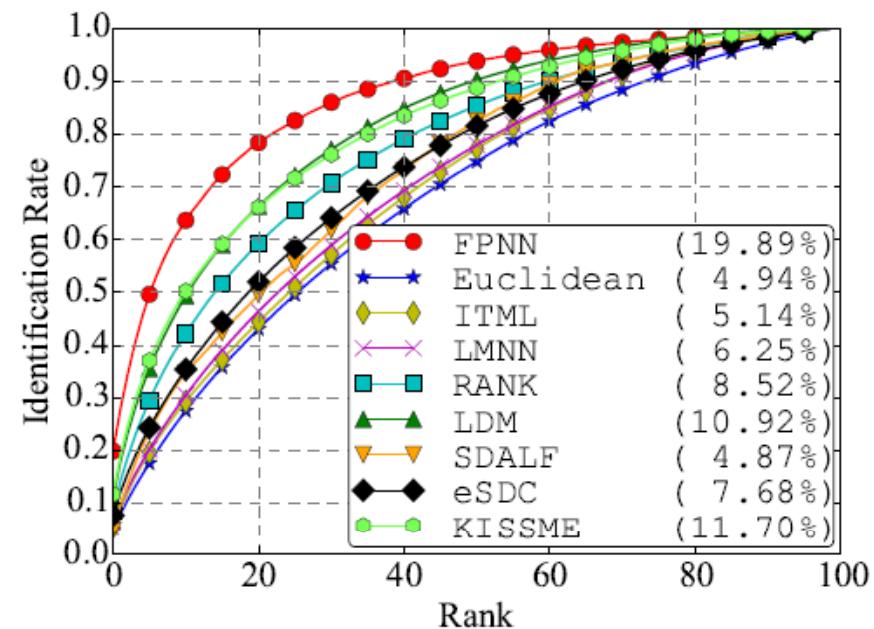
A fully connected layer captures global geometric transforms

Result

- Evaluated on 13,164 of 1,360 persons



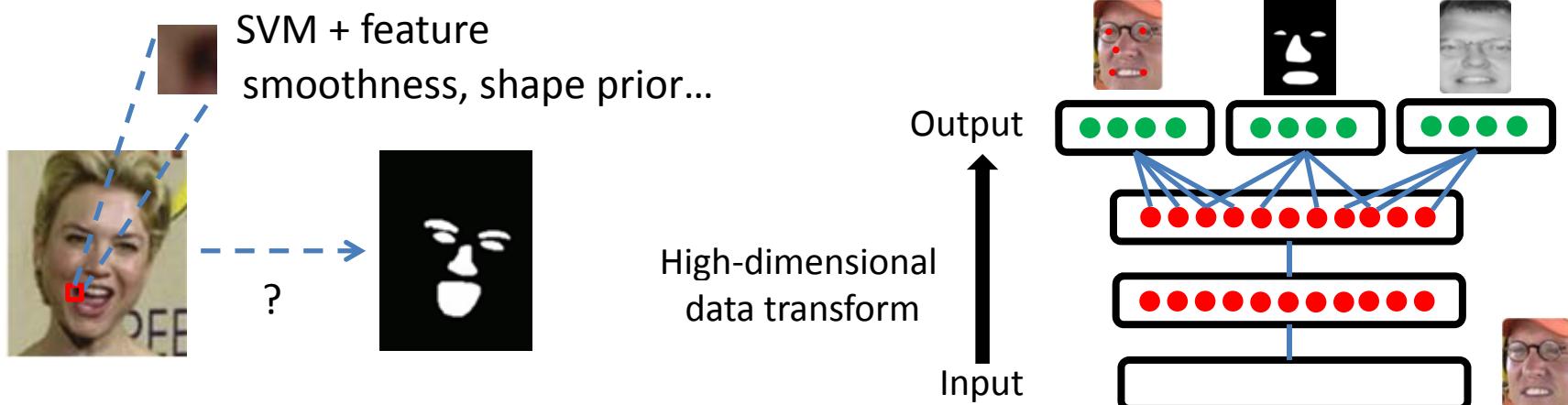
Manually labeled bounding boxes



Automatically detected bounding boxes

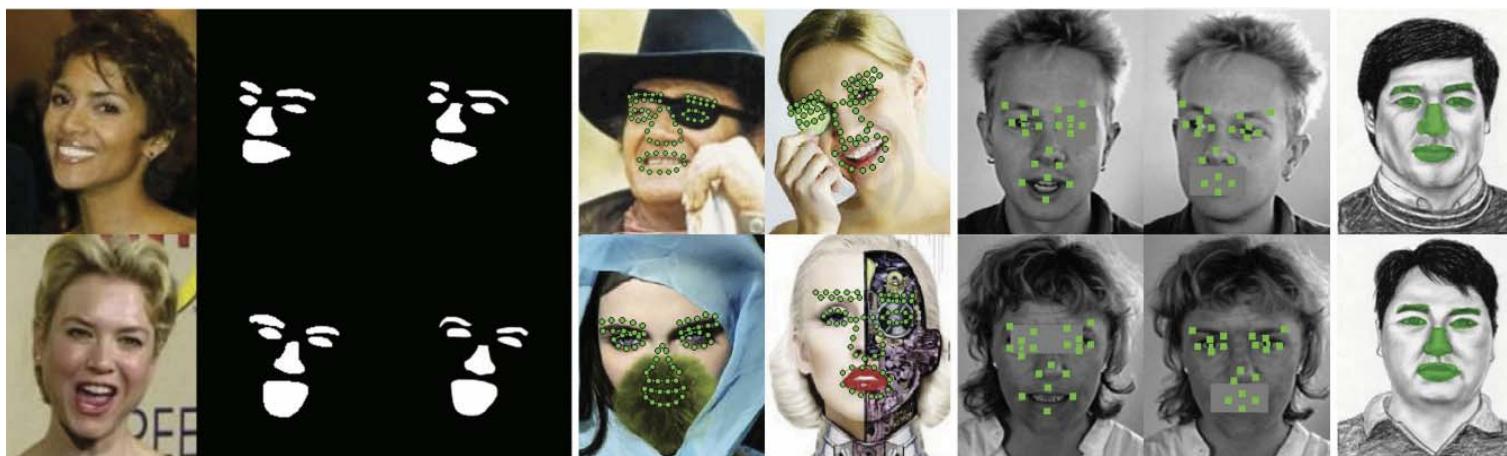
**Large learning capacity makes high dimensional
data transforms possible, and makes better use
of contextual information**

- How to make use of the large learning capacity of deep models?
 - **High dimensional data transform**
 - Hierarchical nonlinear representations



Face Parsing

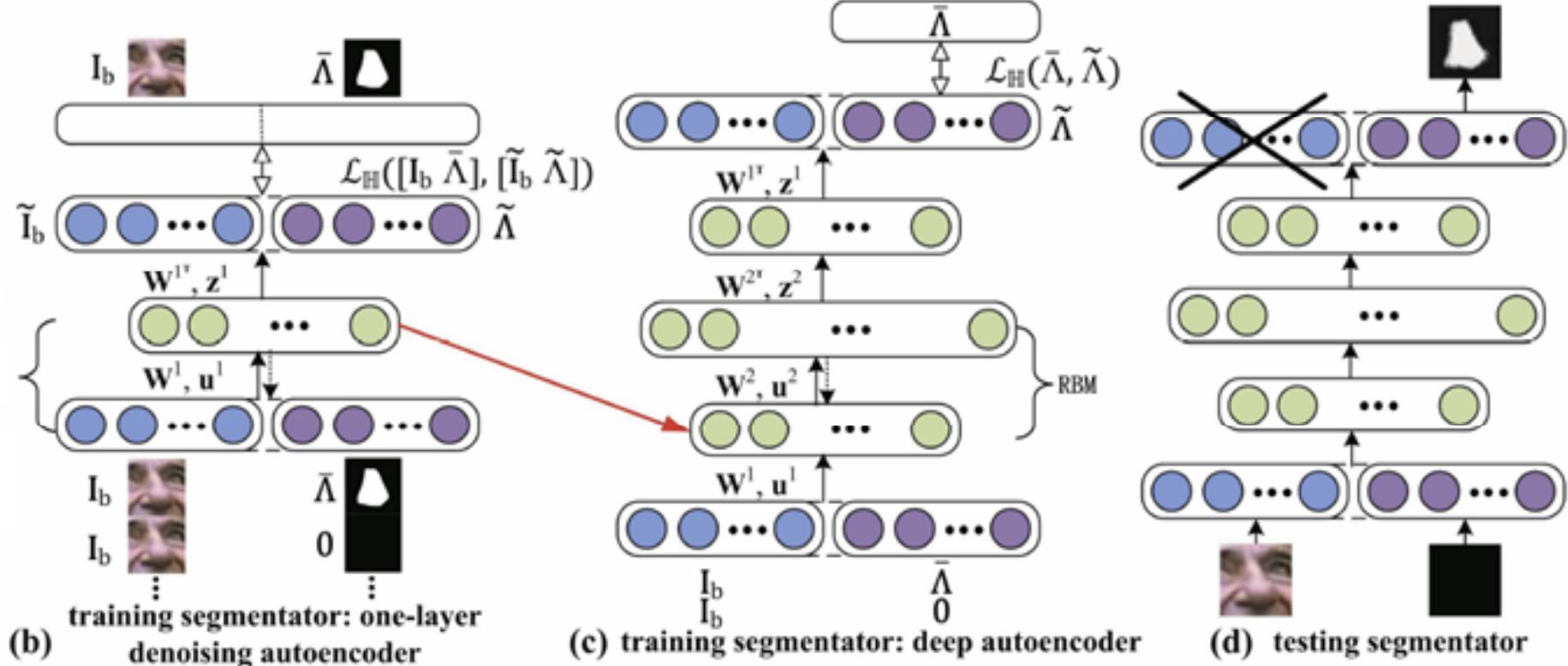
- P. Luo, X. Wang and X. Tang, “Hierarchical Face Parsing via Deep Learning,” CVPR 2012



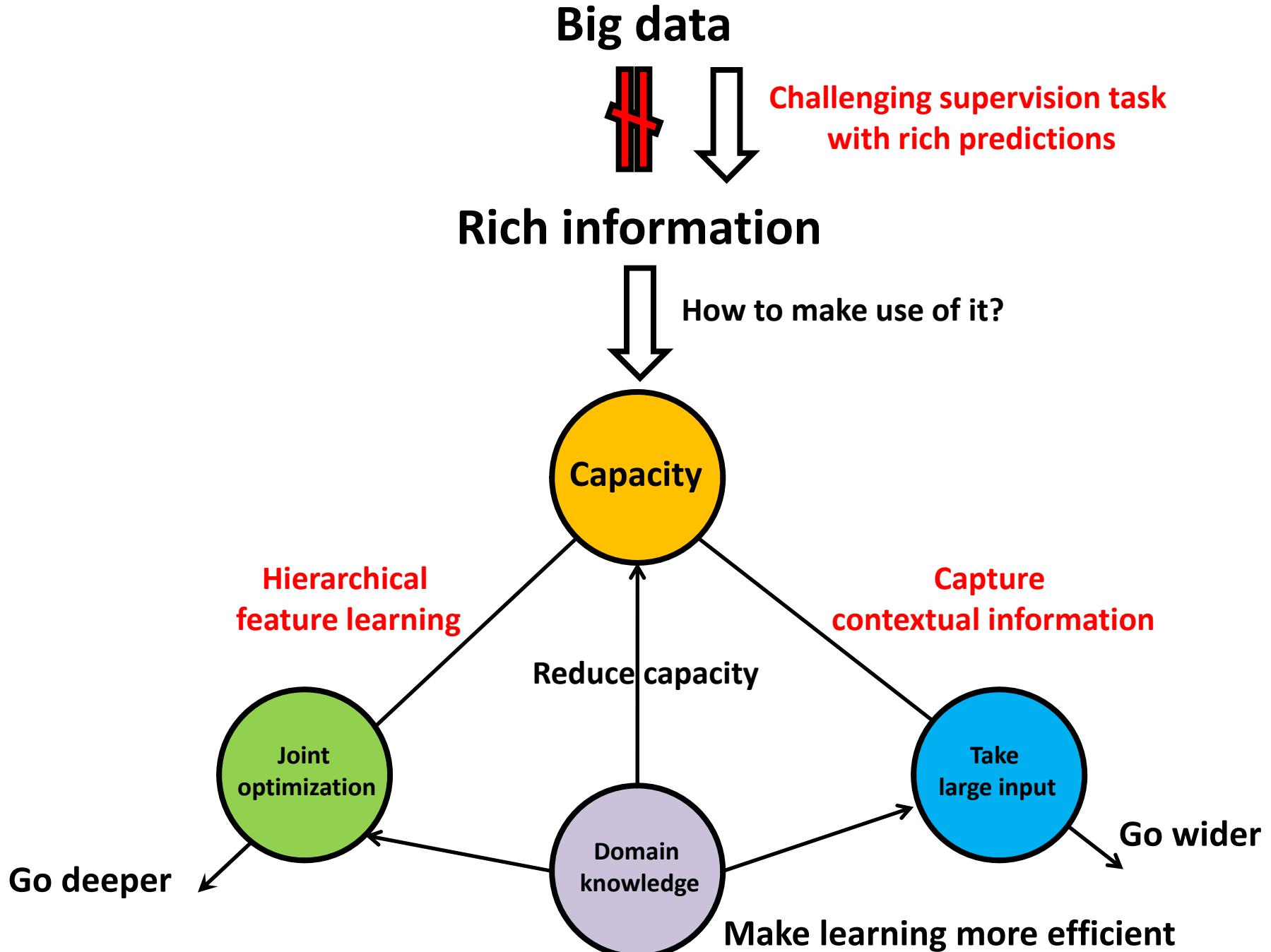
Motivations

- Recast face segmentation as a cross-modality data transformation problem
- Cross modality autoencoder
- Data of two different modalities share the same representations in the deep model
- Deep models can be used to learn shape priors for segmentation

Training Segmentators







Summary

- Automatically learns hierarchical feature representations from data and disentangles hidden factors of input data through multi-level nonlinear mappings
- For some tasks, the expressive power of deep models increases exponentially as their architectures go deep
- Jointly optimize all the components in a vision and create synergy through close interactions among them
- Benefiting the large learning capacity of deep models, we also recast some classical computer vision challenges as high-dimensional data transform problems and solve them from new perspectives
- It is more effective to train deep models with challenging tasks and rich predictions

References

- D. E. Rumelhart, G. E. Hinton, R. J. Williams, “Learning Representations by Back-propagation Errors,” *Nature*, Vol. 323, pp. 533-536, 1986.
- N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A. J. Rodriguez-Sanchez, L. Wiskott, “Deep Hierarchies in the Primate Visual Cortex: What Can We Learn For Computer Vision?” *IEEE Trans. PAMI*, Vol. 35, pp. 1847-1871, 2013.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Proc. NIPS*, 2012.
- Y. Sun, X. Wang, and X. Tang, “Deep Learning Face Representation by Joint Identification-Verification,” *NIPS*, 2014.
- K. Fukushima, “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position,” *Biological Cybernetics*, Vol. 36, pp. 193-202, 1980.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, Vol. 86, pp. 2278-2324, 1998.
- G. E. Hinton, S. Osindero, and Y. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *Neural Computation*, Vol. 18, pp. 1527-1544, 2006.

- G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” *Science*, Vol. 313, pp. 504-507, July 2006.
- Z. Zhu, P. Luo, X. Wang, and X. Tang, “Deep Learning Identity Face Space,” *Proc. ICCV*, 2013.
- Z. Zhu, P. Luo, X. Wang, and X. Tang, “Deep Learning and Disentangling Face Representation by Multi-View Perception,” *NIPS* 2014.
- Y. Sun, X. Wang, and X. Tang, “Deep Learning Face Representation from Predicting 10,000 classes,” *Proc. CVPR*, 2014.
- J. Hastad, “Almost Optimal Lower Bounds for Small Depth Circuits,” *Proc. ACM Symposium on Theory of Computing*, 1986.
- J. Hastad and M. Goldmann, “On the Power of Small-Depth Threshold Circuits,” *Computational Complexity*, Vol. 1, pp. 113-129, 1991.
- A. Yao, “Separating the Polynomial-time Hierarchy by Oracles,” *Proc. IEEE Symposium on Foundations of Computer Science*, 1985.
- Sermnet, K. Kavukcuoglu, S. Chintala, and LeCun, “Pedestrian Detection with Unsupervised Multi-Stage Feature Learning,” *CVPR* 2013.
- W. Ouyang and X. Wang, “Joint Deep Learning for Pedestrian Detection,” *Proc. ICCV*, 2013.
- P. Luo, X. Wang and X. Tang, “Hierarchical Face Parsing via Deep Learning,” *Proc. CVPR*, 2012.
- Honglak Lee, “Tutorial on Deep Learning and Applications,” *NIPS* 2010.

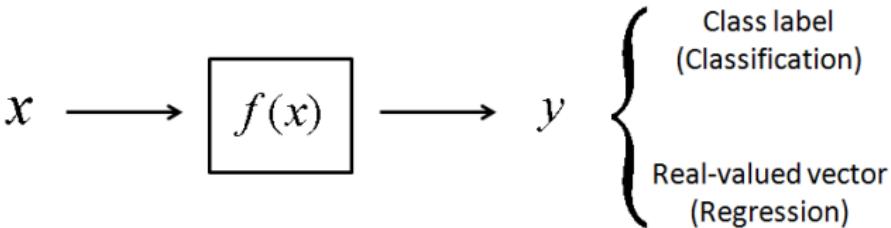
Machine Learning Basics

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

January 5, 2015

Machine Learning



Object recognition

{dog, cat, horse, flower, ...}



Low-resolution image

Super resolution



High-resolution
image

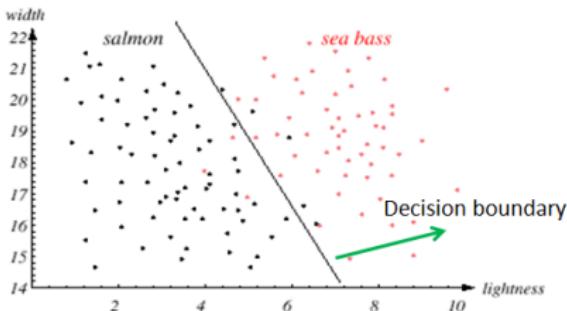
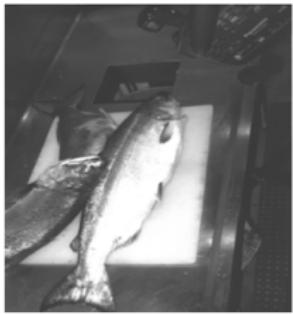
Classification

- $f(\mathbf{x})$ predicts the category that \mathbf{x} belongs to

$$f : \mathcal{R}^D \rightarrow \{1, \dots, K\}$$

- $f(\mathbf{x})$ is decided by the decision boundary
- As an variant, f can also predict the probability distribution over classes given \mathbf{x} , $f(\mathbf{x}) = P(y|\mathbf{x})$. The category is predicted as

$$y^* = \arg \max_k P(y = k | \mathbf{x})$$



(Duda et al. Pattern Classification 2000)

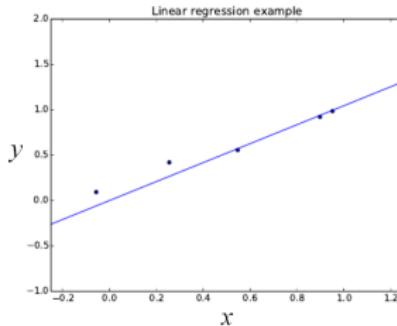
Regression

- Predict real-valued output

$$f : \mathcal{R}^D \rightarrow \mathcal{R}^M$$

- Example: linear regression

$$y = \mathbf{w}^t \mathbf{x} = \sum_{d=1}^D w_d x_d + w_0$$



Training

- Training: estimate the parameters of f from $\{(\mathbf{x}_i^{(\text{train})}, y_i^{(\text{train})})\}$
 - Decision boundary, parameters of $P(y|\mathbf{x})$, and \mathbf{w} in linear regression
- Optimize an objective function on the training set. It is a performance measure on the training set and could be different from that on the test set.
 - Mean squared error (MSE) for linear regression

$$\text{MSE}_{\text{train}} = \frac{1}{N} \sum_i \|\mathbf{w}^t \mathbf{x}_i^{(\text{train})} - y_i^{(\text{train})}\|_2^2$$

- Cross entropy (CE) for classification

$$\text{CE}_{\text{train}} = \frac{1}{N} \sum_i \log P(y = y_i^{(\text{train})} | \mathbf{x}_i^{(\text{train})})$$

Why not use classification errors $\#\{f(\mathbf{x}_i^{(\text{train})}) \neq y_i^{(\text{train})}\}$?

Optimization

- The choice of the objective function should be good for optimization
- Take linear regression as an example

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = 0$$

$$\Rightarrow \nabla_{\mathbf{w}} ||\mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})}||_2^2 = 0$$

$$\mathbf{w} = (\mathbf{X}^{(\text{train})t} \mathbf{X}^{(\text{train})})^{-1} \mathbf{X}^{(\text{train})t} \mathbf{y}^{(\text{train})}$$

where $\mathbf{X}^{(\text{train})} = [\mathbf{x}_1^{(\text{train})}, \dots, \mathbf{x}_N^{(\text{train})}]$ and $\mathbf{y}^{(\text{train})} = [y_1^{(\text{train})}, \dots, y_N^{(\text{train})}]$.

Generalization

- We care more about the performance of the model on new, previously unseen examples
- The training examples usually cannot cover all the possible input configurations, so the learner has to generalize from the training examples to new cases
- Generalization error: the expected error over **ALL** examples
- To obtain theoretical guarantees about generalization of a machine learning algorithm, we assume all the samples are drawn from a distribution $p(\mathbf{x}, y)$, and calculate generalization error (GE) of a prediction function f by taking expectation over $p(\mathbf{x}, y)$

$$GE_f = \int_{\mathbf{x},y} p(\mathbf{x}, y) \mathbf{Error}(f(\mathbf{x}), y)$$

Generalization

- However, in practice, $p(\mathbf{x}, y)$ is unknown. We assess the generalization performance with a test set $\{\mathbf{x}_i^{(\text{test})}, y_i^{(\text{test})}\}$

$$\text{Performance}_{\text{test}} = \frac{1}{M} \sum_{i=1}^M \text{Error}(f(\mathbf{x}_i^{(\text{test})}), y_i^{(\text{test})})$$

- We hope that both test examples and training examples are drawn from $p(\mathbf{x}, y)$ of interest, although it is unknown

Capacity

- The ability of the learner (or called model) to discover a function taken from a family of functions. Examples:

- Linear predictor

$$y = wx + b$$

- Quadratic predictor

$$y = w_2x^2 + w_1x + b$$

- Degree-10 polynomial predictor

$$y = b + \sum_{i=1}^{10} w_i x^i$$

- The latter family is richer, allowing to capture more complex functions
- Capacity can be measured by the number of training examples $\{\mathbf{x}_i^{(\text{train})}, y_i^{(\text{train})}\}$ that the learner **could always fit**, no matter how to change the values of $\mathbf{x}_i^{(\text{train})}$ and $y_i^{(\text{train})}$

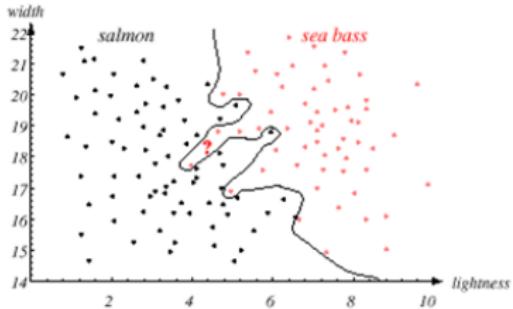
Underfitting

- The learner cannot find a solution that fits training examples well
 - For example, use linear regression to fit training examples $\{\mathbf{x}_i^{(train)}, y_i^{(train)}\}$ where $y_i^{(train)}$ is an quadratic function of $\mathbf{x}_i^{(train)}$
- Underfitting means the learner cannot capture some important aspects of the data
- Reasons for underfitting happening
 - Model is not rich enough
 - Difficult to find the global optimum of the objective function on the training set or easy to get stuck at local minimum
 - Limitation on the computation resources (not enough training iterations of an iterative optimization procedure)
- Underfitting commonly happens in deep learning with large scale training data and could be even a more serious problem than overfitting in some cases

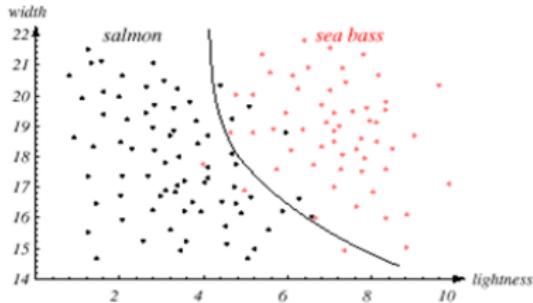
Overfitting

- The learner fits the training data well, but loses the ability to generalize well, i.e. it has small training error but larger generalization error
- A learner with large capacity tends to overfit
 - The family of functions is too large (compared with the size of the training data) and it contains many functions which all fit the training data well.
 - Without sufficient data, the learner cannot distinguish which one is most appropriate and would make an arbitrary choice among these apparently good solutions
 - A separate validation set helps to choose a more appropriate one
 - In most cases, data is contaminated by noise. The learner with large capacity tends to describe random errors or noise instead of the underlying models of data (classes)

Overfitting



Overly complex models lead to complicated decision boundaries. It leads to perfect classification on the training examples, but would lead to poor performance on new examples.



The decision boundary might represent the optimal tradeoff between performance on the training set and simplicity of classifier, therefore giving highest accuracy on new examples.

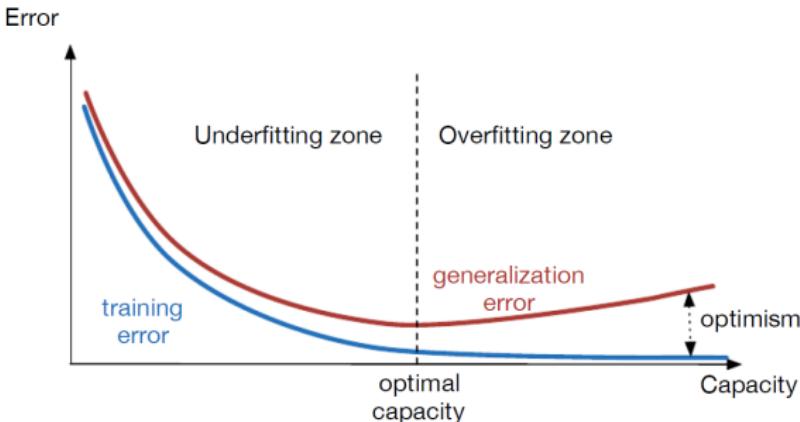
(Duda et al. Pattern Classification 2000)

- **The fundamental element of machine learning is the trade-off between capacity and generalization**
- Occam's Razor states that among competing functions that could explain the training data, one should choose the “simpler” one. Simplicity is the opposite of capacity.
- Occam's Razor suggests us pick the family of functions just enough large enough to leave only one choice that fits well the data.

Optimal capacity

- Difference between training error and generalization error increases with the capacity of the learner
- Generalization error is a U-shaped function of capacity
- Optimal capacity capacity is associated with the transition from underfitting to overfitting
 - One can use a validation set to monitor generalization error empirically
- Optimal capacity should increase with the number of training examples

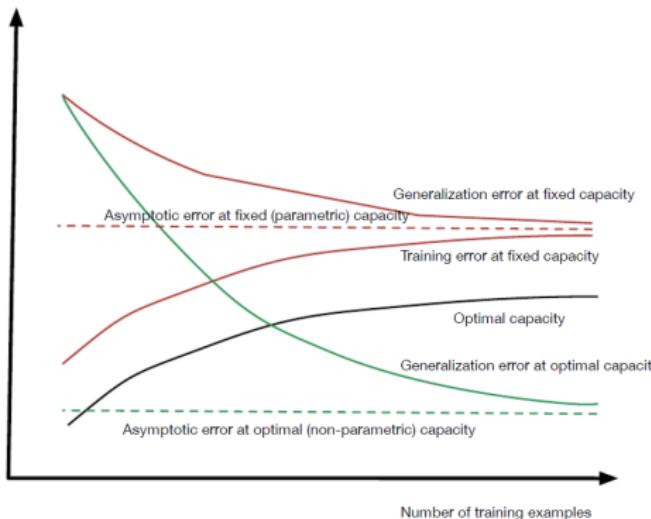
Optimal capacity



Typical relationship between capacity and both training and generalization (or test) error. As capacity increases, training error can be reduced, but the optimism (difference between training and generalization error) increases. At some point, the increase in optimism is larger than the decrease in training error (typically when the training error is low and cannot go much lower), and we enter the overfitting regime, where capacity is too large, above the optimal capacity. Before reaching optimal capacity, we are in the underfitting regime.

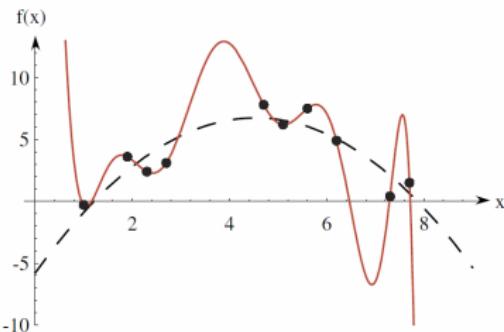
(Bengio et al. Deep Learning 2014)

Optimal capacity



As the number of training examples increases, optimal capacity (bold black) increases (we can afford a bigger and more flexible model), and the associated generalization error (green bold) would decrease, eventually reaching the (non-parametric) asymptotic error (green dashed line). If capacity was fixed (parametric setting), increasing the number of training examples would also decrease generalization error (top red curve), but not as fast, and training error would slowly increase (bottom red curve), so that both would meet at an asymptotic value (dashed red line) corresponding to the best achievable solution in some class of learned functions.

Exercise question



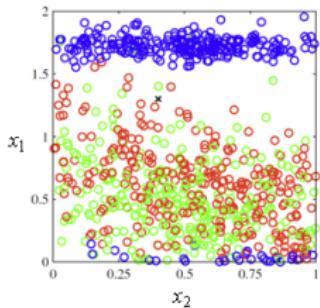
- In the figure above, the training data (10 black dots) were selected from a quadratic function plus Gaussian noise, i.e., $f(x) = w_2x^2 + w_1x + b + \epsilon$ where $p(\epsilon) = N(0, \sigma^2)$. The degree-10 polynomial fits the data perfectly. Which learner should be chosen in order to better predict new examples? The second-order function or the 10th degree function?
- If the ten training examples were generated from a 10th degree polynomial plus Gaussian noise, which learned should be chosen?
- If the one million training examples were generated from a quadratic function plus Gaussian noise, which learned should be chosen?

How to reduce capacity?

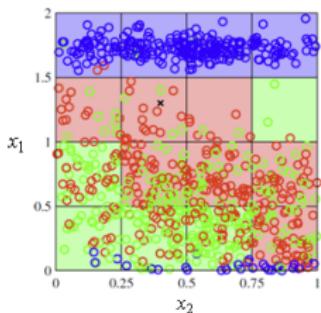
- Reduce the number of features
- Reduce the number of **independent** parameters
- Reduce the network size of deep models
- Reduce the number of training iterations
- Add regularization to the learner
- ...

Curse of dimensionality

- Why do we need to reduce the dimensionality of the feature space?



Scatter plot of the training data of three classes. Two features are used. The goal is to classify the new testing point denoted by 'x'.

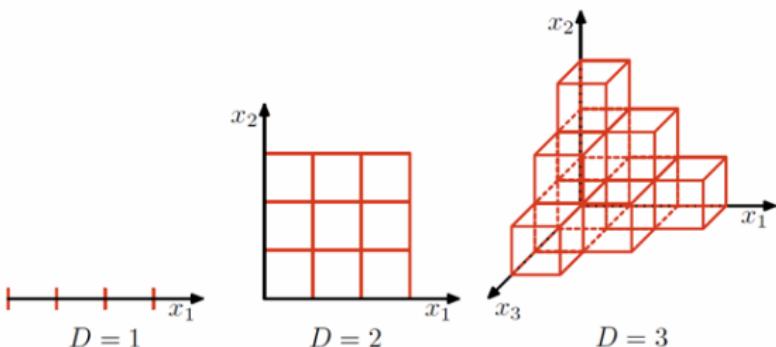


The feature space is uniformly divided into cells. A cell is labeled as a class, if the majority of training examples in that cell are from that class. The testing point is classified according to the label of the cell where it falls in.

(Duda et al. Pattern Classification 2000)

Curse of dimensionality

- The more training samples in each cell, the more robust the classifier
- The number of cells grows exponentially with the dimensionality of the feature space. If each dimension is divided into three intervals, the number of cells is $N = 3^D$
- Some cells are empty when the number of cells is very large!



(Duda et al. Pattern Classification 2000)

Regularization

- Equivalent to imposing a preference over the set of functions that a learner can obtain as a solution
- In Bayesian learning, it is reflected as a prior probability distribution over the space of functions (or equivalently their parameters) that the learner can assess
- Regularization prevents overfitting by adding penalty for complexity
- Training a classifier/regressor is to minimize
Prediction error on the training set + regularization
- Examples
 - The objective function for linear regression becomes

$$\text{MSE}_{\text{train}} + \text{regularization} = \frac{1}{N} \sum_i (\mathbf{w}^t \mathbf{x}_i^{(\text{train})} - y_i^{(\text{train})})^2 + \lambda \|\mathbf{w}\|_2^2$$

- Multi-task learning, transfer learning, dropout, sparsity, pre-training

Function estimation

- We are interested in predicting y from input \mathbf{x} and assume there exists a function that describes the relationship between y and \mathbf{x} , e.g.
 $y = f(\mathbf{x}) + \epsilon$, where ϵ is random noise following certain distribution.
- Prediction function f can be parametrized by a parameter vector θ .
- Estimating \hat{f}_n from a training set $\mathcal{D}_n = \{(\mathbf{x}_1^{(\text{train})}, y_1^{(\text{train})}), \dots, (\mathbf{x}_n^{(\text{train})}, y_n^{(\text{train})})\}$ is equivalent to estimating $\hat{\theta}_n$ from \mathcal{D}_n .
- Since \mathcal{D}_n is randomly generated from an underlying distribution, both $\hat{\theta}$ and \hat{f} are random variables (or vectors, or functions) distributed according to some probability distributions.
- The quality of estimation can be measured by bias and variance compared with the “true” parameter vector θ or function \hat{f} .
- With a better design of the parametric form of the function, the learner could achieve low generalization error even with small capacity
- This design process typically involves domain knowledge

Bias

$$\text{bias}(\hat{\theta}) = E(\hat{\theta}) - \theta$$

where expectation is over all the train sets of size n sampled from the underlying distribution

- An estimator is called unbiased if $E(\hat{\theta}) = \theta$
- Example: Gaussian distribution. $p(\mathbf{x}_i; \theta) = \mathcal{N}(\theta, \Sigma)$ and the estimator is $\hat{\theta} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^{(\text{train})}$

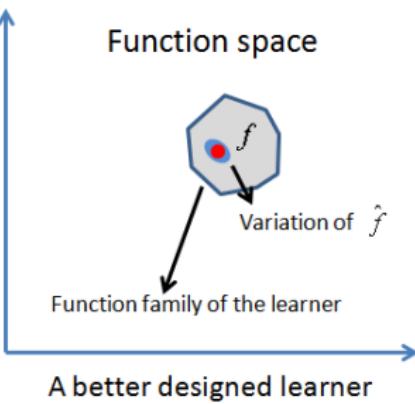
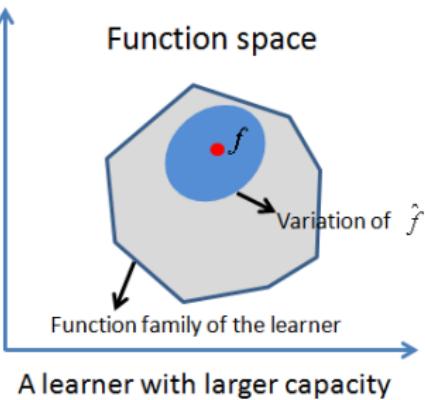
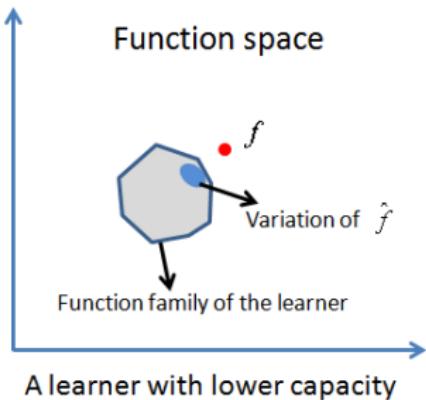
$$E(\hat{\theta}) = E \left[\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^{(\text{train})} \right] = \frac{1}{n} \sum_{i=1}^n E \left[\mathbf{x}_i^{(\text{train})} \right] = \frac{1}{n} \sum_{i=1}^n \theta = \theta$$

$$\text{Var}[\hat{\theta}] = E[(\hat{\theta} - E[\hat{\theta}])^2] = E[\hat{\theta}^2] - E[\hat{\theta}]^2$$

- Variance typically decreases as the size of the train set increases
- Both bias and variance are the sources of estimation errors

$$\text{MSE} = E[(\hat{\theta} - \theta)^2] = \text{Bias}(\hat{\theta})^2 + \text{Var}[\hat{\theta}]$$

- Increasing the capacity of a learner may also increase variance, although it has better chance to cover the true function

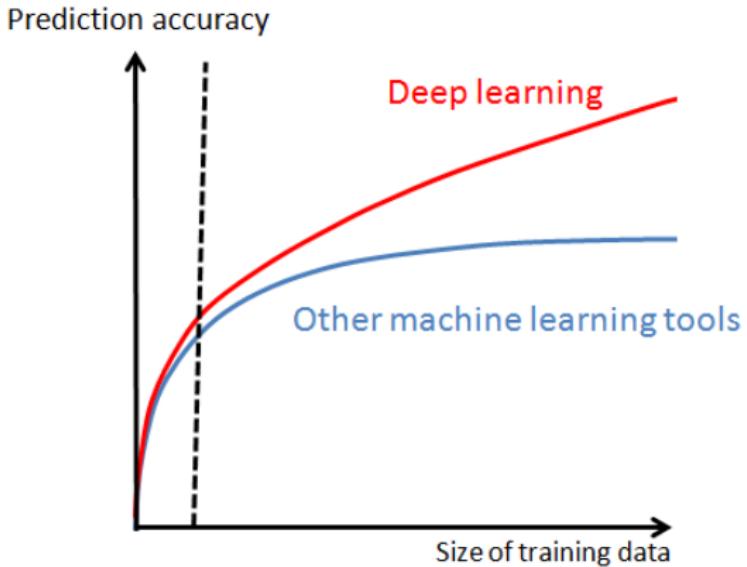


Summary: issues to be concerned in machine learning

- Effective optimization methods and models to address the underfitting problem
- How to balance the trade-off between capacity and generalization?
- How to effectively reduce capacity (which means also reducing estimation variance) without increasing the bias much?
- For machine learning with big training data, how to effectively increase capacity to cover or get closer to the true function to be estimated?

Open discussion

- Why does deep learning have different behavior than other machine learning methods for large scale training?



Discriminative model

- Directly model $P(y|\mathbf{x})$ and decision boundaries
- Learn the discriminative functions $g_k(\mathbf{x})$

$$y = \arg \max_k g_k(\mathbf{x})$$

- In the linear case, $g_k(\mathbf{x}) = \mathbf{w}_k^t \mathbf{x}$
- $P(y|\mathbf{x})$ can be estimated from the linear discriminant functions

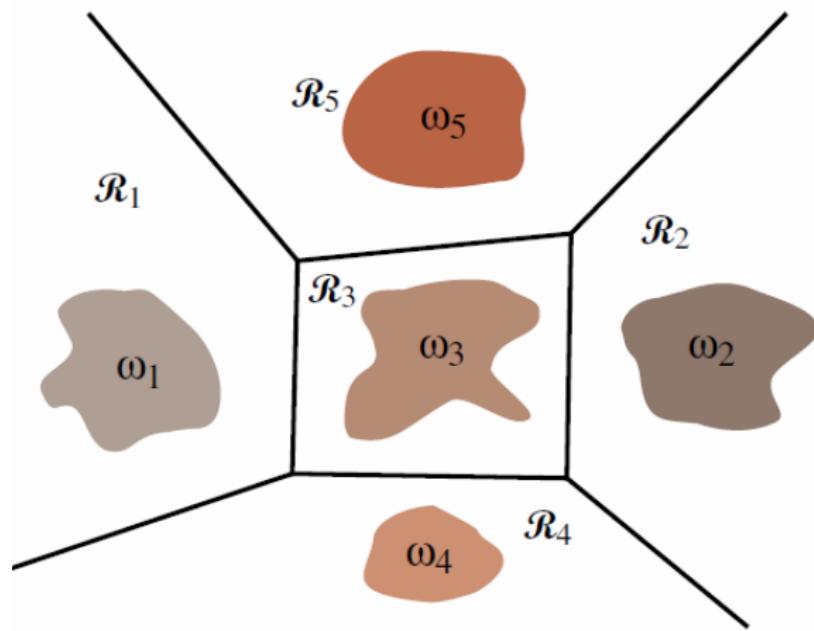
$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{w}_j^t \mathbf{x}}}{\sum_{k=1}^K e^{\mathbf{w}_k^t \mathbf{x}}}$$

It is also called softmax function

- Examples: SVM, boosting, K-nearest-neighbor

Discriminative model

- It is easier for discriminative models to fit data



Discriminative model

- Parameter $\theta = \{\mathbf{w}_k\}$ can be estimated from maximizing the data likelihood

$$\hat{\theta} = \arg \max_{\theta} P(\mathcal{D}_n | \theta) = \arg \max_{\theta} \prod_{i=1}^n P(y_n^{(\text{train})} | \mathbf{x}_n^{(\text{train})}, \theta)$$

- Maximum a posteriori* (MAP) estimation

$$\theta = \arg \max_{\theta} p(\theta | \mathcal{D}_n) = \arg \max_{\theta} \log P(\mathcal{D}_n | \theta) + \log p(\theta)$$

- According to the Bayes' rule, i.e., $p(\theta | \mathcal{D}_n) = P(\mathcal{D}_n | \theta)p(\theta) / P(\mathcal{D}_n)$,

$$\theta = \arg \max_{\theta} \log P(\mathcal{D}_n | \theta) + \log p(\theta)$$

$$\theta = \arg \max_{\theta} \sum_{i=1}^n \log P(y_n^{(\text{train})} | \mathbf{x}_n^{(\text{train})}, \theta) + \log p(\theta)$$

- prior $p(\theta)$ corresponds to a regularizer, e.g.

$$p(\theta) = e^{-\lambda ||\theta||^2}$$

Generative model

- Estimate the underlying class conditional probability densities $p(\mathbf{x}|y = k)$ and then construct the classifier using the Bayesian decision theory

$$P(y = k|\mathbf{x}) = \frac{p(\mathbf{x}|y = k)P(y = k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)P(y = k)}{\sum_{k'=1}^K p(\mathbf{x}|y = k')P(y = k')}$$

- $p(\mathbf{x}|y)$ and $P(y)$ are parameterized by θ
- Prior $P(y)$ can be used to model the dependency among predictions, such as the segmentation labels of pixels or predictions of speech sequences.
- It is more difficult to model class conditional probability densities. However, it also adds stronger regularization to model fitting, since the learned model not only needs to predict class labels but also generate the input data.
- It is easier to add domain knowledge when designing the models of $p(\mathbf{x}|y)$

Supervised and unsupervised learning

- Supervised learning: the goal is to use input-label pairs, (\mathbf{x}, y) to learn a function f that predicts a label (or a distribution over labels) given the input, $\hat{y} = f(\mathbf{x})$
- Unsupervised learning: no label or other target is provided. The data consists of a set of examples \mathbf{x} and the objective is to learn about the statistical structure of \mathbf{x} itself.
- Weakly supervised learning: the training data contains (\mathbf{x}, y) pairs as in supervised learning, but the labels y are either unreliably present (i.e. with missing values) or noisy (i.e. where the label given is not the true label)

Unsupervised learning

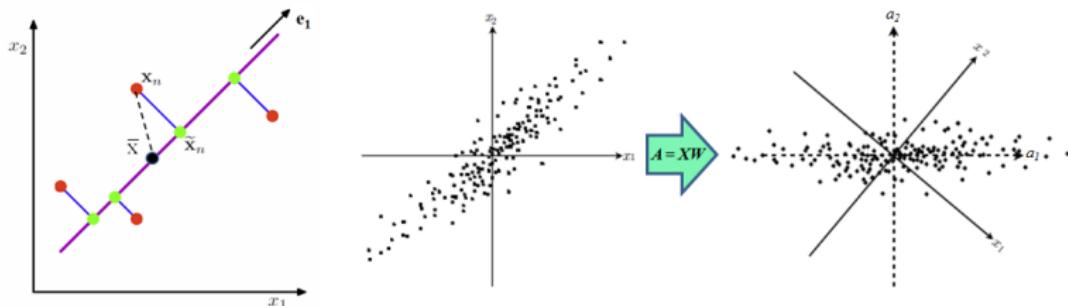
- Find the “best” representation of data that reserves as much information about \mathbf{x} as possible while being “simpler” than \mathbf{x}
Taking linear case as an example

$$\tilde{\mathbf{x}} = a_0 + \sum_{i=1}^{d'} a_i \mathbf{e}_i$$

- Lower dimensional representation: $d' < d$
- Sparse representation: the number of non-zero a_i is small
- Independent representation: disentangle the sources of variations underlying the data distributions such that the dimensions of the representation are statistically independent, i.e. a_i and a_j are statistically independent
- Deep learning is to learn data representation, but in a nonlinear and hierarchical way

Principal Component Analysis (PCA)

- There are n d -dimensional samples $\mathbf{x}_1, \dots, \mathbf{x}_n$.
- PCA seeks a **principal subspace** spanned by d' ($d' < d$) orthonormal vectors $\mathbf{e}_1, \dots, \mathbf{e}_{d'}$, such that
 - the projected samples ($\tilde{\mathbf{x}}_k = a_0 + \sum_{i=1}^{d'} a_{ki} \mathbf{e}_i$) onto this subspace has maximum variance; or equivalently
 - the mean squared distance between the samples and their projections are minimized.
 - The projections $\{a_{ki}\}$ are uncorrelated (i.e. independent if data distribution is assumed as Gaussian distribution)



Formulation of PCA

$$\arg \max_{\{\mathbf{e}_i\}} \sum_{k=1}^n \|\tilde{\mathbf{x}}_k - \bar{\mathbf{x}}\|^2$$

or

$$\arg \min_{\{\mathbf{e}_i\}} \sum_{k=1}^n \|\mathbf{x}_k - \tilde{\mathbf{x}}_k\|^2$$

$$(\|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 = \|\mathbf{x}_k - \tilde{\mathbf{x}}_k\|^2 + \|\tilde{\mathbf{x}}_k - \bar{\mathbf{x}}\|^2)$$

- zero-dimensional representation: use a single vector \mathbf{x}_0 to represent all the samples and minimize the squared-error function

$$J_0(\mathbf{x}_0) = \sum_{k=1}^n \|\mathbf{x}_0 - \mathbf{x}_k\|^2$$

The solution is $\mathbf{x}_0 = \bar{\mathbf{x}} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k$

- One-dimensional representation: project the data to a line running through the sample mean, $\tilde{\mathbf{x}}_k = \bar{\mathbf{x}} + a_{k1}\mathbf{e}_1$ and minimize the squared-error criterion function

$$J_1(a_{11}, \dots, a_{n1}, \mathbf{e}_1) = \sum_{k=1}^n \|\tilde{\mathbf{x}}_k - \mathbf{x}_k\|^2$$

Find the Principal Components a_{k1}

$$\begin{aligned} & J_1(a_{11}, \dots, a_{n1}, \mathbf{e}_1) \\ = & \sum_{k=1}^n \|(\bar{\mathbf{x}} + a_{k1}\mathbf{e}_1) - \mathbf{x}_k\|^2 = \sum_{k=1}^n \|a_{k1}\mathbf{e}_1 - (\mathbf{x}_k - \bar{\mathbf{x}})\|^2 \\ = & \sum_{k=1}^n a_{k1}^2 \|\mathbf{e}_1\|^2 - 2 \sum_{k=1}^n a_{k1} \mathbf{e}_1^t (\mathbf{x}_k - \bar{\mathbf{x}}) + \sum_{k=1}^n \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \end{aligned}$$

Since \mathbf{e}_1 is a unit vector, $\|\mathbf{e}_1\| = 1$. To minimize J_1 , set $\frac{\partial J_1}{\partial a_{k1}} = 0$ and we have

$$a_{k1} = \mathbf{e}_1^t (\mathbf{x}_k - \bar{\mathbf{x}})$$

- We obtain a least-squares solution by projecting the vector \mathbf{x}_k onto the line in the direction of \mathbf{e}_1 passing through the mean.

Find the Optimal Projection Direction \mathbf{e}_1

$$\begin{aligned} J_1(\mathbf{e}_1) &= \sum_{k=1}^n a_{k1}^2 - 2 \sum_{k=1}^n a_{k1} + \sum_{k=1}^n \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \\ &= - \sum_{k=1}^n [\mathbf{e}_1^t (\mathbf{x}_k - \bar{\mathbf{x}})]^2 + \sum_{k=1}^n \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \\ &= - \sum_{k=1}^n \mathbf{e}_1^t (\mathbf{x}_k - \bar{\mathbf{x}}) (\mathbf{x}_k - \bar{\mathbf{x}})^t \mathbf{e}_1 + \sum_{k=1}^n \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \\ &= - \mathbf{e}_1^t \mathbf{S} \mathbf{e}_1 + \sum_{k=1}^n \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 \end{aligned}$$

Find the Optimal Projection Direction \mathbf{e}_1

- $\mathbf{S} = \sum_{k=1}^n (\mathbf{x}_k - \bar{\mathbf{x}})(\mathbf{x}_k - \bar{\mathbf{x}})^t$ is the scatter matrix
- $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1 = \sum_{k=1}^n a_{k1}^2$ ($\sum_{k=1}^n a_{k1} = 0$) is the variance of the projected data
- The vector \mathbf{e}_1 that minimizes J_1 also maximizes $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1$, subject to the constraint that $\|\mathbf{e}_1\| = 1$

Lagrange Optimization

- Seek the position \mathbf{x}_0 of an extremum of a scalar-valued function $f(\mathbf{x})$ subject to the constraint that $g(\mathbf{x}) = 0$
- First form the Lagrangian function

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

where λ is a scalar called the Lagrange *undetermined multiplier*.

- Convert into an unconstrained problem by taking the derivative,

$$\frac{\partial L(\mathbf{x}, \lambda)}{\partial \mathbf{x}} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} + \lambda \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} = 0$$

- Solve \mathbf{x} and λ considering $g(\mathbf{x}) = 0$

Find the Optimal Projection Direction \mathbf{e}_1

- Use the method of Lagrange multipliers to maximize the $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1$ subject to the constraint that $\|\mathbf{e}_1\| = 1$,

$$u = \mathbf{e}_1^t \mathbf{S} \mathbf{e}_1 - \lambda(\mathbf{e}_1^t \mathbf{e}_1 - 1),$$

$$\frac{\partial u}{\partial \mathbf{e}_1} = 2\mathbf{S}\mathbf{e}_1 - 2\lambda\mathbf{e}_1 = 0.$$

- Setting the gradient vector equal to zero, we see that \mathbf{e}_1 must be an eigenvector of the scatter matrix

$$\mathbf{S}\mathbf{e}_1 = \lambda\mathbf{e}_1$$

- Since $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1 = \lambda \mathbf{e}_1^t \mathbf{e}_1 = \lambda$, to maximize $\mathbf{e}_1^t \mathbf{S} \mathbf{e}_1$, we select the eigenvector with the largest eigenvalue

d' -Dimensional Representation by PCA

- d' -dimensional representation: $\tilde{\mathbf{x}}_k = \bar{\mathbf{x}} + \sum_{i=1}^{d'} a_{ki} \mathbf{e}_i$
- Mean-squared criterion function:

$$J_{d'} = \sum_{k=1}^n \left\| \left(\bar{\mathbf{x}} + \sum_{i=1}^{d'} a_{ki} \mathbf{e}_i \right) - \mathbf{x}_k \right\|^2$$

- Define additional principal components in an incremental fashion by choosing each new direction minimizing J amongst all possible directions orthogonal to those already considered
- To minimize $J_{d'}$, $\mathbf{e}_1, \dots, \mathbf{e}_{d'}$ are the d' eigenvectors of the scatter matrix with the largest eigenvalues. a_{ki} are the **principal components** of samples.

d' -Dimensional Representation by PCA

- Since the scatter matrix is real and symmetric, its eigenvectors are orthogonal and its eigenvalues are nonnegative.
- The squared error:

$$J_{d'} = - \sum_{i=1}^{d'} \lambda_i + \sum_{k=1}^n \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2$$

$$J_d = 0 \Rightarrow \sum_{k=1}^n \|\mathbf{x}_k - \bar{\mathbf{x}}\|^2 = \sum_{i=1}^d \lambda_i$$

$$J_{d'} = \sum_{i=d'+1}^d \lambda_i$$

Variance of Data Captured by the PCA Subspace

- The variance of data projected onto the first d' eigenvectors is $\sum_{i=1}^{d'} \lambda_i$
- Measure how much variance has been captured by the first d' eigenvectors:

$$\frac{\sum_{i=1}^{d'} \lambda_i}{\sum_{j=1}^d \lambda_j}$$

Covariance Matrix of Principal Components

- The correlation between projections on \mathbf{e}_i and \mathbf{e}_j ($i \neq j$) is

$$\sum_{k=1}^n a_{ki} a_{kj} = \sum_{k=1}^n \mathbf{e}_i^t (\mathbf{x}_k - \bar{\mathbf{x}}) (\mathbf{x}_k - \bar{\mathbf{x}})^t \mathbf{e}_j = \mathbf{e}_i^t \mathbf{S} \mathbf{e}_j = \lambda_i \mathbf{e}_i^t \mathbf{e}_j = 0$$

- The covariance matrix of samples in the PCA subspace is
 $\text{diag}[\lambda_1, \dots, \lambda_{d'}]$

Summary of PCA

- The principal subspace is spanned by d' orthonormal vectors $\mathbf{e}_1, \dots, \mathbf{e}_{d'}$ which are computed as the d' eigenvectors of the scatter matrix \mathbf{S} with the largest eigenvalues $\lambda_1, \dots, \lambda_{d'}$.
- The principal components a_{ki} of the samples are computed as $a_{ki} = \mathbf{e}_i^t(\mathbf{x}_k - \bar{\mathbf{x}})$
- The variance of the projected samples onto this principal subspace is $\sum_{i=1}^{d'} = \lambda_i$.
- The mean squared distance between the samples and their projections are $\sum_{i=d'+1}^d = \lambda_i$.
- PCA disentangles the factors of variation underlying the data, assuming such variation is a Gaussian distribution
- We are interested in learning representations that disentangle more complicated forms of feature dependencies

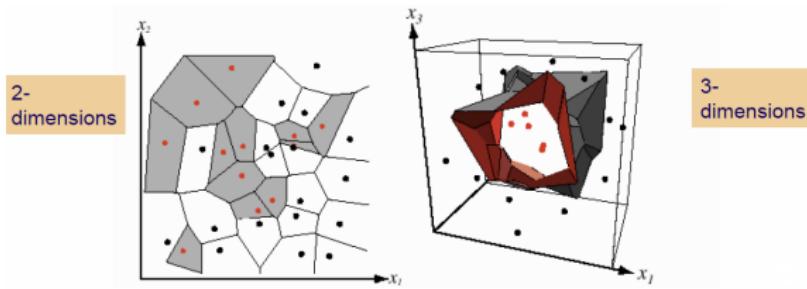
Smoothness Prior

- Shallow models assume smoothness prior on the prediction function to be learned, i.e.

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \epsilon)$$

where ϵ is a small change.

- K-nearest neighbor predictors assume piecewise constant
 - For classification and $K = 1$, $f(\mathbf{x})$ is the output class associated with the nearest neighbor of \mathbf{x} in the training set
 - For regression, $f(\mathbf{x})$ is the average of the outputs associated with the K nearest neighbors of \mathbf{x}
 - The number of distinguishable regions cannot be more than the number of training examples



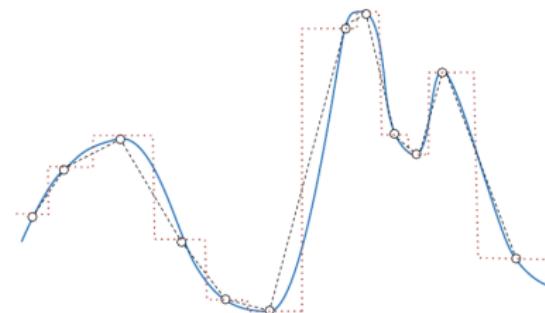
Interpolation with Kernel

$$f(\mathbf{x}) = b + \sum_{i=1}^n \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

K is a kernel function, e.g., the Gaussian kernel

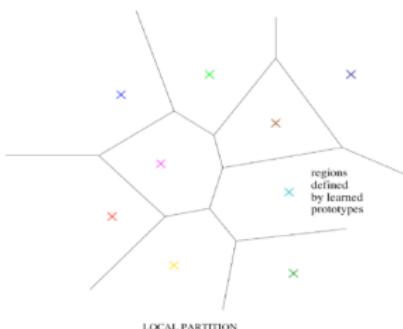
$$K(\mathbf{u}, \mathbf{v}) = N(\mathbf{u} - \mathbf{v}; 0, \sigma^2 I)$$

- b and α_i can be learned by SVM
- Treat each \mathbf{x}_i is a template and the kernel function as a similarity function that matches a template and a test example



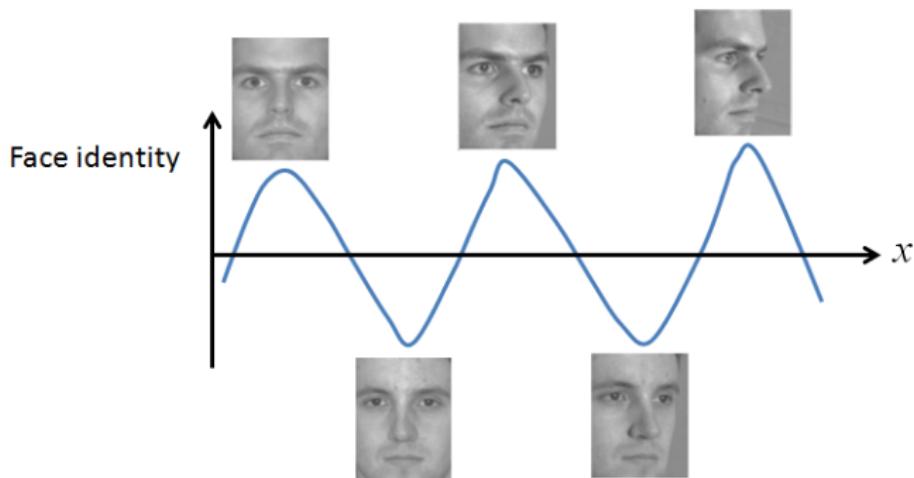
Local Representation

- One can think of the training samples as control knots which locally specify the shape of the prediction function
- The smoothness prior only allows the learner to generalize locally. If (\mathbf{x}_i, y_i) is a supervised training example and \mathbf{x}_i is a near neighbor of \mathbf{x} , we expect that $f(\mathbf{x}) \approx y_i$. Better generalization can be obtained with more neighbors.
- To distinguish $O(N)$ regions in the input space, shallow models require $O(N)$ examples (and typically there are $O(N)$ parameters associated with the $O(N)$ regions).



Local Representation

- If the function is complex, more regions and more training samples are required.
- The representation learned by deep models can be generalized non-locally



Yoshua Bengio, Ian Goodfellow and Aaron Courville, Chapter 1
“Machine Learning Basics” in “Deep Learning,” Book in
preparation for MIT Press, 2014.

<http://www.iro.umontreal.ca/~bengioy/dlbook>

MultiLayer Neural Networks

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

January 18, 2015

Outline

1 Feedforward Operation

2 Backpropagation

3 Discussions

History of neural network

- Pioneering work on the mathematical model of neural networks
 - McCulloch and Pitts 1943
 - Include recurrent and non-recurrent (with “circles”) networks
 - Use thresholding function as nonlinear activation
 - No learning
- Early works on learning neural networks
 - Starting from Rosenblatt 1958
 - Using thresholding function as nonlinear activation prevented computing derivatives with the chain rule, and so errors could not be propagated back to guide the computation of gradients
- Backpropagation was developed in several steps since 1960
 - The key idea is to use the chain rule to calculate derivatives
 - It was reflected in multiple works, earliest from the field of control

History of neural network

- Standard backpropagation for neural networks
 - Rumelhart, Hinton, and Williams, *Nature* 1986. Clearly appreciated the power of backpropagation and demonstrated it on key tasks, and applied it to pattern recognition generally
 - In 1985, Yann LeCun independently developed a learning algorithm for three-layer networks in which target values were propagated, rather than derivatives. In 1986, he proved that it was equivalent to standard backpropagation
- Prove the universal expressive power of three-layer neural networks
 - Hecht-Nielsen 1989
- Convolutional neural network
 - Introduced by Kunihiko Fukushima in 1980
 - Improved by LeCun, Bottou, Bengio, and Haffner in 1998

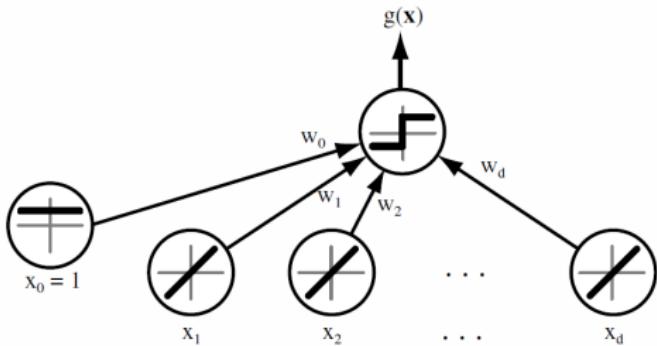
History of neural network

- Deep belief net (DBN)
 - Hinton, Osindero, and Tech 2006
- Auto encoder
 - Hinton and Salakhutdinov 2006 (*Science*)
- Deep learning
 - Hinton. Learning multiple layers of representations. *Trends in Cognitive Sciences*, 2007.
 - Unsupervised multilayer pre-training + supervised fine-tuning (BP)
- Large-scale deep learning in speech recognition
 - Geoff Hinton and Li Deng started this research at Microsoft Research Redmond in late 2009.
 - Generative DBN pre-training was not necessary
 - Success was achieved by large-scale training data + large deep neural network (DNN) with large, context-dependent output layers

History of neural network

- Unsupervised deep learning from large scale images
 - Andrew Ng et al. 2011
 - Unsupervised feature learning
 - 16000 CPUs
- Large-scale supervised deep learning in ImageNet image classification
 - Krizhevsky, Sutskever, and Hinton 2012
 - Supervised learning with convolutional neural network
 - No unsupervised pre-training

Two-layer neural networks model linear classifiers



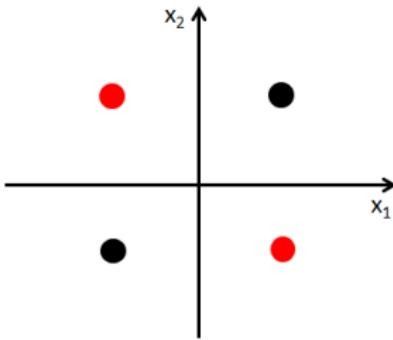
(Duda et al. Pattern Classification 2000)

$$g(\mathbf{x}) = f\left(\sum_{i=1}^d x_i w_i + w_0\right) = f(\mathbf{w}^t \mathbf{x})$$

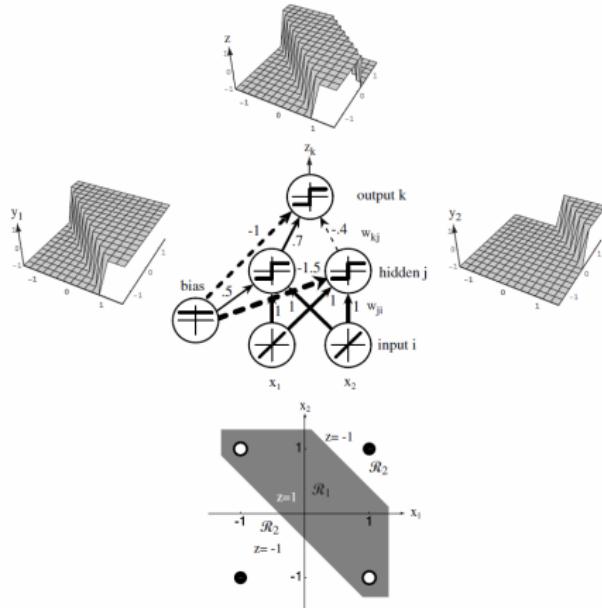
$$f(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ -1, & \text{if } s < 0 \end{cases} .$$

Two-layer neural networks model linear classifiers

- A linear classifier cannot solve the simple exclusive-OR problem

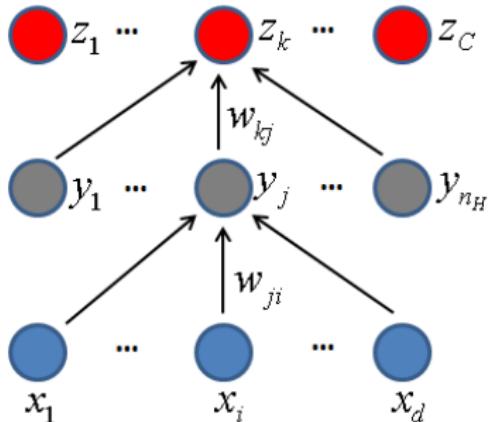


Add a hidden layer to model nonlinear classifiers



(Duda et al. Pattern Classification 2000)

Three-layer neural network



For C-class classification problem, the target vectors are represented as

$$(1 \dots 0 \dots 0)$$

⋮

$$(0 \dots 1 \dots 0)$$

⋮

$$(0 \dots 0 \dots 1)$$

Three-layer neural network

- Net activation: each hidden unit j computes the weighted sum of its inputs

$$net_j = \sum_{i=1}^d x_i w_{ji} + w_{j0} = \sum_{i=0}^d x_i w_{ji} = \mathbf{w}_j^t \mathbf{x}$$

- Activation function: each hidden unit emits an output that is a **nonlinear** function of its activation

$$y_j = f(net_j)$$

$$f(net) = Sgn(net) = \begin{cases} 1, & \text{if } net \geq 0 \\ -1, & \text{if } net < 0 \end{cases}.$$

There are multiple choices of the activation function as long as they are continuous and differentiable **almost everywhere**. Activation functions could be different for different nodes.

Three-layer neural network

- Net activation of an output unit k

$$net_k = \sum_{i=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{w}_k^t \mathbf{y}$$

- Output unit emits

$$z_k = f(net_k)$$

- The output of the neural network is equivalent to a set of discriminant functions

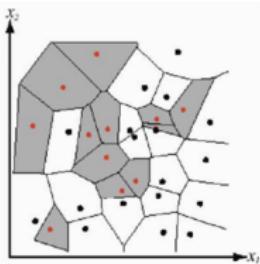
$$g_k(\mathbf{x}) = z_k = f \left(\sum_{j=1}^{n_H} w_{kj} f \left(\sum_{i=1}^d w_{ji} x_i + w_{j0} \right) + w_{k0} \right)$$

Expressive power of a three-layer neural network

- It can represent **any** discriminant function
- However, the number of hidden units required can be very large...
- Most widely used pattern recognition models (such as SVM, boosting, and KNN) can be approximated as neural networks with one or two hidden layers. They are called models with shallow architectures.
- Shallow models divide the feature space into regions and match templates in local regions. $O(N)$ parameters are needed to represent N regions.
- Deep architecture: the number of hidden nodes can be reduced exponentially with more layers for certain problems.

Expressive power of a three-layer neural network

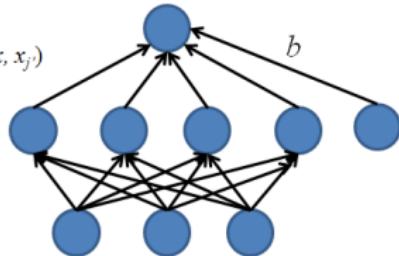
KNN



output $g(x) = \text{label}_j$
 $j = \operatorname{argmin}_{j'} d(x, x_{j'})$

hidden $d(x, x_j)$

input x

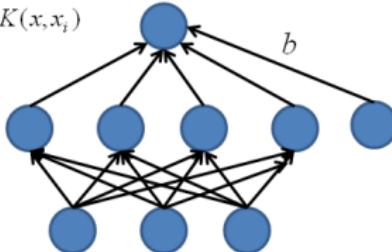


SVM

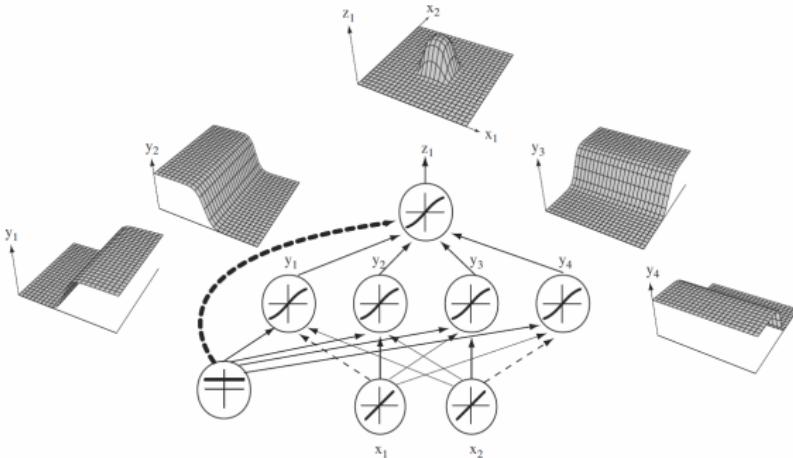
output $g(x) = b + \sum_i \alpha_i K(x, x_i)$

hidden $K(x, x_j)$

input x



Expressive power of a three-layer neural network



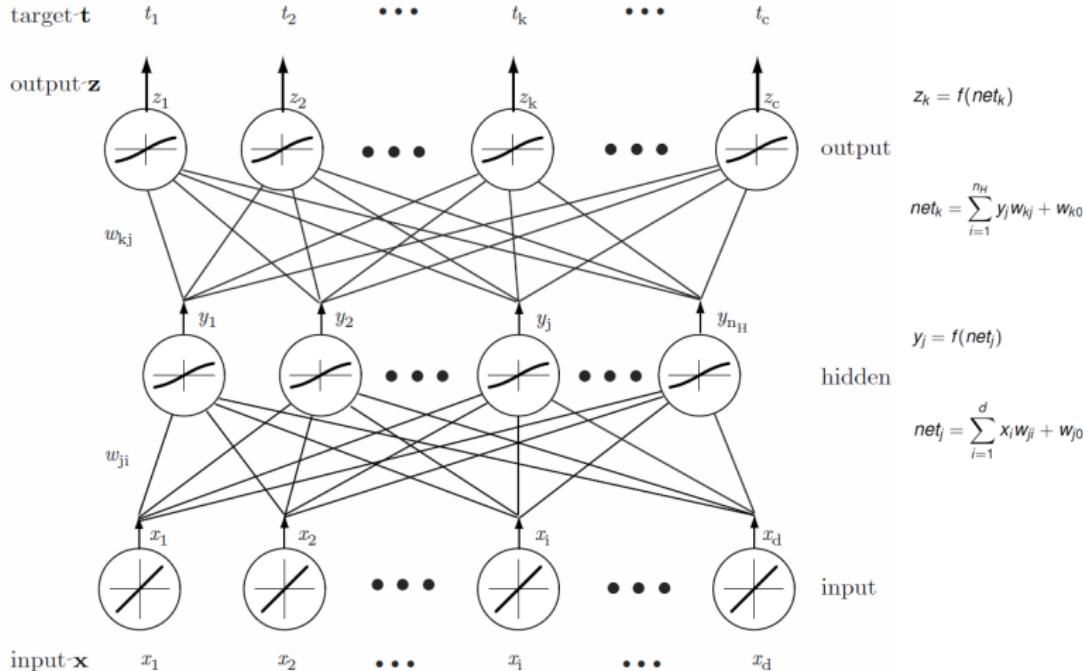
(Duda et al. Pattern Classification 2000)

With a tanh activation function $f(s) = (e^s - e^{-s})/(e^s + e^{-s})$, the hidden unit outputs are paired in opposition thereby producing a “bump” at the output unit. With four hidden units, a local mode (template) can be modeled. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network.

Backpropagation

- The most general method for supervised training of multilayer neural network
- Present an input pattern and change the network parameters to bring the actual outputs closer to the target values
- Learn the input-to-hidden and hidden-to-output weights
- However, there is no explicit teacher to state what the hidden unit's output should be. Backpropagation calculates an effective error for each hidden unit, and thus derive a learning rule for the input-to-hidden weights.

A three-layer network for illustration



Training error

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

- Differentiable
- There are other choices, such as cross entropy

$$J(\mathbf{w}) = - \sum_{k=1}^c t_k \log(z_k)$$

Both $\{z_k\}$ and $\{t_k\}$ are probability distributions.

Gradient descent

- Weights are initialized with random values, and then are changed in a direction reducing the error

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

or in component form

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

where η is the learning rate.

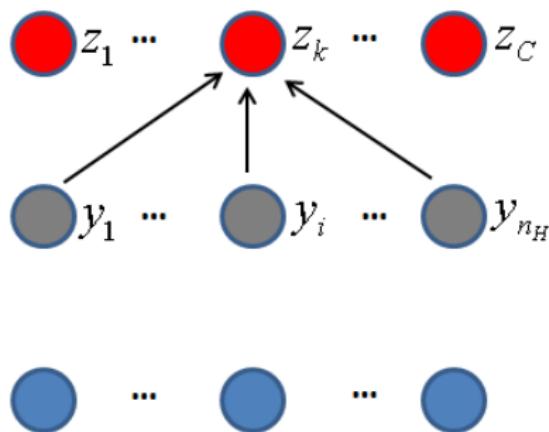
- Iterative update

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

Hidden-to-output weights w_{kj}

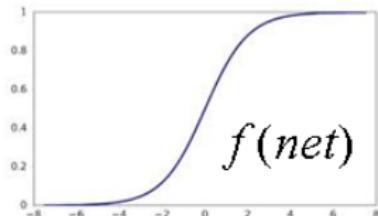
$t_1 \quad \dots \quad t_k \quad \dots \quad t_C$

Group truth



$$z_k = f(\text{net}_k)$$

$$\text{net}_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0}$$



Hidden-to-output weights w_{kj}

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

- Sensitivity of unit k

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

Describe how the overall error changes with the unit's net activation.

- Weight update rule. Since $\partial net_k / \partial w_{kj} = y_j$,

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

Activation function

- Sign function is not a good choice for $f(\cdot)$. Why?
- Popular choice of $f(\cdot)$

- Sigmoid function

$$f(s) = \frac{1}{1 + e^{-s}}$$

- Tanh function (shift the center of Sigmoid to the origin)

$$f(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

- Hard tanh

$$f(s) = \max(-1, \min(1, x))$$

- Rectified linear unit (ReLU)

$$f(s) = \max(0, x)$$

- Softplus: smooth version of ReLU

$$f(s) = \log(1 + e^s)$$

Activation function

- Popular choice of $f(\cdot)$
 - Softmax: mostly used as output non-linearity for predicting discrete probabilities

$$f(s_k) = \frac{e^{s_k}}{\sum_{k'=1}^C e^{s_{k'}}}$$

- Maxout: it generalizes the rectifier assuming there are multiple net activations

$$f(s_1, \dots, s_n) = \max_i(s_i)$$

Example 1

- Choose squared error as training error measurement and sigmoid as activation function at the output layer
- When the output probabilities approach to 0 or 1 (i.e. saturate), $f'(net)$ gets close to zero and δ_k is small even if the error $(t_k - z_k)$ is large, which is bad.

Example 2

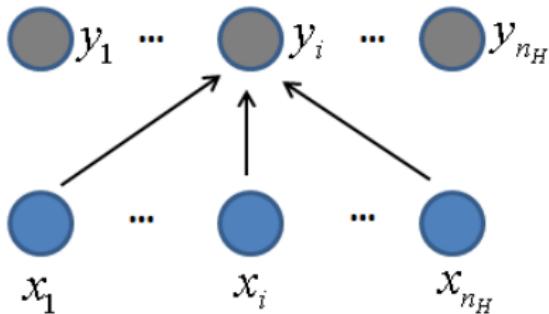
- Choose cross entropy as training error measurement and softmax as activation function at the output layer
- Sensitivity $\delta_k = -t_k(z_k - 1)$ (how to get it?)
- δ_k is large if error is large, even if z_k gets close to 0
- Softmax leads to sparser output

Input-to-hidden weights

$t_1 \quad \dots \quad t_k \quad \dots \quad t_C$

Group truth

 $z_1 \quad \dots \quad$  $z_k \quad \dots \quad$  z_C



$$y_j = f(\text{net}_j)$$

$$\text{net}_j = \sum_{j=1}^d x_j w_{ji} + w_{j0}$$

Input-to-hidden weights

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

- How the hidden unit output y_j affects the error at each output unit

$$\begin{aligned}\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j} \left[\frac{1}{2} \sum_{k=1}^c (t_k - z_k)^2 \right] \\ &= - \sum_{k=1}^c (t_k - z_k) \frac{\partial z_k}{\partial y_j} \\ &= - \sum_{k=1}^c (t_k - z_k) \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial y_j} \\ &= - \sum_{k=1}^c (t_k - z_k) f'(net_k) w_{kj} = \sum_{k=1}^c \delta_k w_{kj}\end{aligned}$$

Input-to-hidden weights

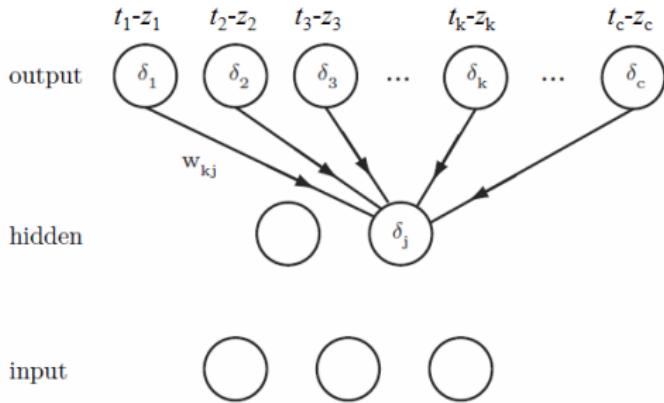
- Sensitivity for a hidden unit j

$$\delta_j = -\frac{\partial J}{\partial \text{net}_j} = -\frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial \text{net}_j} = f'(\text{net}_j) \sum_{k=1}^c w_{kj} \delta_k$$

- $\sum_{k=1}^c w_{kj} \delta_k$ is the effective error for hidden unit j
- Weight update rule. Since $\partial \text{net}_j / \partial w_{ji} = x_i$,

$$\Delta w_{ji} = \eta x_i \delta_j = \eta f'(\text{net}_j) \left[\sum_{k=1}^c w_{kj} \delta_k \right] x_i$$

Error backpropagation



(Duda et al. Pattern Classification 2000) The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units:
$$\delta_j = f'(net_j) \sum_{k=1}^c w_{kj} \delta_k$$
. The output unit sensitivities are thus propagated "back" to the hidden units.

Stochastic gradient descent

- Given n training samples, our target function can be expressed as

$$J(\mathbf{w}) = \sum_{p=1}^n J_p(\mathbf{w})$$

- Batch gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{p=1}^n \nabla J_p(\mathbf{w})$$

- In some cases, evaluating the sum-gradient may be computationally expensive. Stochastic gradient descent samples a subset of summand functions at every step. This is very effective in the case of large-scale machine learning problems. In stochastic gradient descent, the true gradient of $J(\mathbf{w})$ is approximated by a gradient at a single example (or a mini-batch of samples):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J_p(\mathbf{w})$$

Stochastic backpropagation

Algorithm 1 (Stochastic backpropagation)

```
1 begin initialize network topology (# hidden units),  $\mathbf{w}$ , criterion  $\theta$ ,  $\eta$ ,  $m \leftarrow 0$ 
2   do  $m \leftarrow m + 1$ 
3      $\mathbf{x}^m \leftarrow$  randomly chosen pattern
4      $w_{ij} \leftarrow w_{ij} + \eta \delta_j x_i$ ;  $w_{jk} \leftarrow w_{jk} + \eta \delta_k y_j$ 
5   until  $\nabla J(\mathbf{w}) < \theta$ 
6 return  $\mathbf{w}$ 
7 end
```

(Duda et al. Pattern Classification 2000)

- In stochastic training, a weight update may reduce the error on the single pattern being presented, yet increase the error on the full training set.

Mini-batch based stochastic gradient descent

- Divide the training set into mini-batches.
- In each epoch, randomly permute mini-batches and take a mini-batch sequentially to approximate the gradient
 - One epoch corresponds to a single presentations of all patterns in the training set
- The estimated gradient at each iteration is more reliable
- Start with a small batch size and increase the size as training proceeds

Batch backpropagation

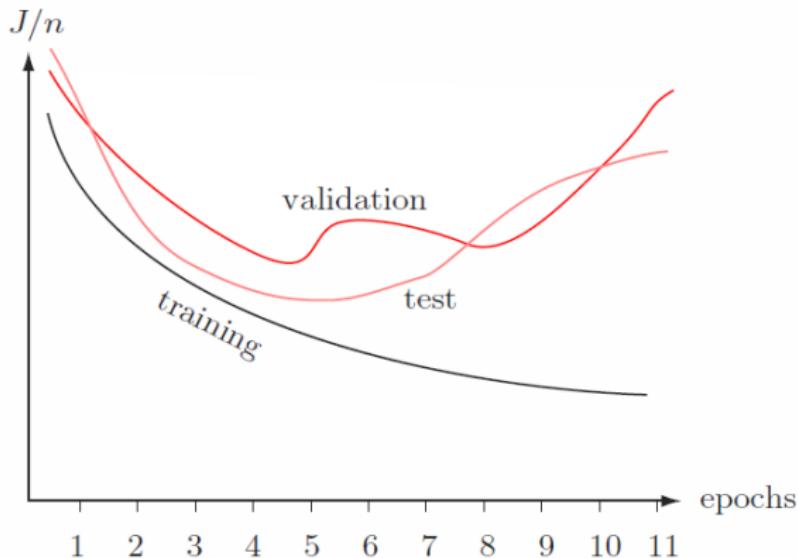
Algorithm 2 (Batch backpropagation)

```
1 begin initialize network topology (# hidden units),  $\mathbf{w}$ , criterion  $\theta, \eta, r \leftarrow 0$ 
2   do  $r \leftarrow r + 1$  (increment epoch)
3      $m \leftarrow 0; \Delta w_{ij} \leftarrow 0; \Delta w_{jk} \leftarrow 0$ 
4     do  $m \leftarrow m + 1$ 
5        $\mathbf{x}^m \leftarrow$  select pattern
6        $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i; \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$ 
7     until  $m = n$ 
8      $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}; w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$ 
9   until  $\nabla J(\mathbf{w}) < \theta$ 
10  return  $\mathbf{w}$ 
11 end
```

Summary

- Stochastic learning
 - Estimate of the gradient is noisy, and the weights may not move precisely down the gradient at each iteration
 - Faster than batch learning, especially when training data has redundancy
 - Noise often results in better solutions
 - The weights fluctuate and it may not fully converge to a local minimum
- Batch learning
 - Conditions of convergence are well understood
 - Some acceleration techniques only operate in batch learning
 - Theoretical analysis of the weight dynamics and convergence rates are simpler

Plot learning curves on the training and validation sets



(Duda et al. Pattern Classification 2000)

Plot the average error per pattern (i.e. $1/n \sum_p J_p$) versus the number of epochs.

Learning curve on the training set

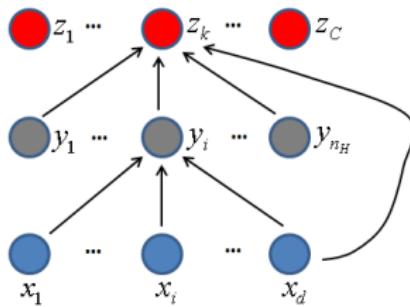
- The average training error typically decreases with the number of epochs and reaches an asymptotic value
- This asymptotic value could be high if **underfitting** happens. The reasons could be
 - The classification problem is difficult (Bayes error is high) and there are a large number of training samples
 - The expressive power of the network is not enough (the numbers of weights, layers and nodes in each layer)
 - Bad initialization and get stuck at local minimum (pre-training for better initialization)
- If the learning rate is low, the training error tends to decrease monotonically, but converges slowly. If the learning rate is high, the training error may oscillate.

Learning curve on the test and validation set

- The average error on the validation or test set is virtually always higher than on the training set. It could increase or oscillate when **overfitting** happen. The reasons could be
 - Training samples are not enough
 - The expressive power of the network is too high
 - Bad initialization and get stuck at local minimum (pre-training for better initialization)
- Stop training at a minimum of the error on the validation set

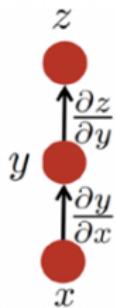
BP on general flow graphs

- BP can be applied to a general flow graph, where each node u_i is the value obtained with a computation unit and partial orders are defined on nodes. $j < i$ means u_j is computed before u_i
- The final node is the objective function depending on all the other nodes
- Directed acyclic graphs
- Example: network with skipping layers (i.e. connecting non-adjacent layers)



BP is an application of the chain rule

$$\frac{\partial C(g(\theta))}{\partial \theta} = \frac{\partial C(g(\theta))}{\partial g(\theta)} \frac{\partial g(\theta)}{\partial \theta}$$

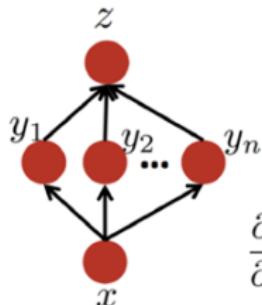


$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

$$\Delta y = \frac{\partial y}{\partial x} \Delta x$$

$$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

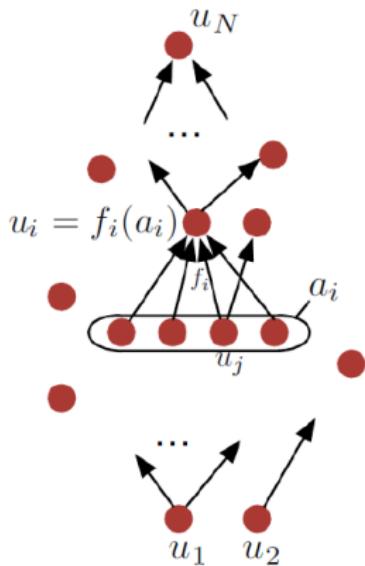


$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

(Bengio et al. Deep Learning 2014)

Flow graph forward computation

- u_1, \dots, u_N are the nodes with defined partial orders
- $a_i = (u_j)_{j \in \text{parents}(i)}$ is the set of parents of node u_i and $u_i = f_i(a_i)$



```
for i = 1 ... M do
     $u_i \leftarrow x_i$ 
end for
for i = M + 1 ... N do
     $a_i \leftarrow (u_j)_{j \in \text{parents}(i)}$ 
     $u_i \leftarrow f_i(a_i)$ 
end for
return  $u_N$ 
```

BP on a flow graph

```
 $\frac{\partial u_N}{\partial u_N} \leftarrow 1$ 
for  $j = N - 1$  down to 1 do
     $\frac{\partial u_N}{\partial u_j} \leftarrow \sum_{i:j \in \text{parents}(i)} \frac{\partial u_N}{\partial u_i} \frac{\partial u_i}{\partial u_j}$ 
end for
return  $\left( \frac{\partial u_N}{\partial u_i} \right)_{i=1}^M$ 
```

(Bengio et al. Deep Learning 2014)

$$\frac{\partial u_N}{\partial w_{ji}} = \frac{\partial u_N}{\partial u_i} \frac{\partial u_i}{\partial net_i} \frac{\partial net_i}{\partial w_{ji}} = \frac{\partial u_N}{\partial u_i} f'(net_i) u_j$$

- BP has optimal computational complexity in the sense that there is no algorithm that can compute the gradient faster (in the $O(\cdot)$ sense)
- It is an application of the principles of dynamic programming

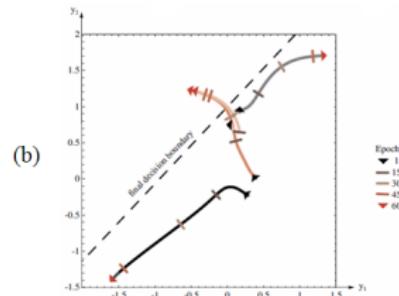
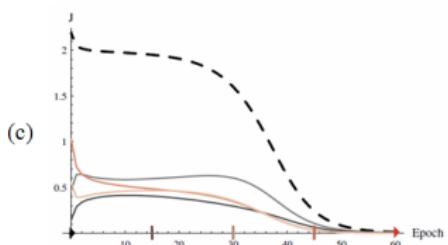
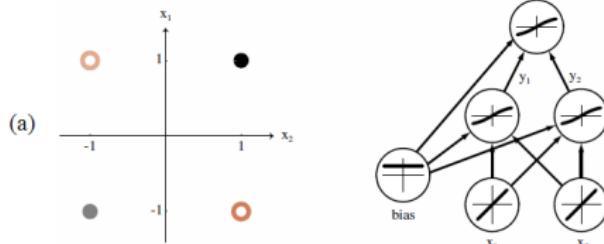
BP on a flow graph

- The derivative of the output with respect to any node can be written in the following intractable form:

$$\frac{\partial u_N}{\partial u_i} = \sum_{\text{paths } u_{k_1} \dots u_{k_n}: k_1=i, k_n=N} \prod_{j=2}^n \frac{\partial u_{k_j}}{\partial u_{k_{j-1}}}$$

where the graphs u_{k_1}, \dots, u_{k_n} go from the node $k_1 = i$ to the final node $k_n = N$ in the flow graph. Computing the sum as above would be intractable because the number of possible paths can be exponential in the depth of the graph. BP is efficient because it employs a dynamic programming strategy to re-use rather than re-compute partial sums associated with the gradients on intermediate nodes.

Nonlinear feature mapping



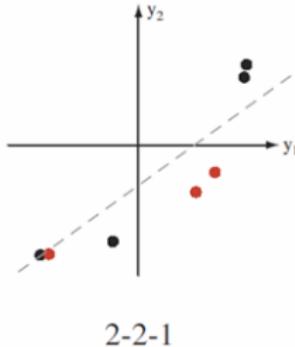
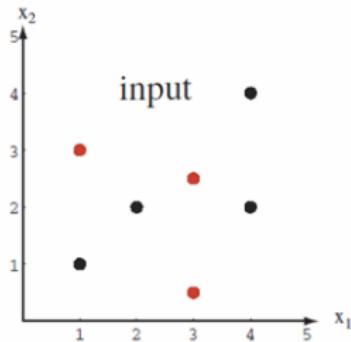
- (a) 2-2-1 backpropagation network and the four patterns of the XOR problem;
- (b) Outputs of the hidden units for each of the four patterns; these outputs move across the y_1y_2 -space as the network learns;
- (c) Learning curves of individual patterns and total error as a function of epoch

(Duda et al. Pattern Classification 2000)

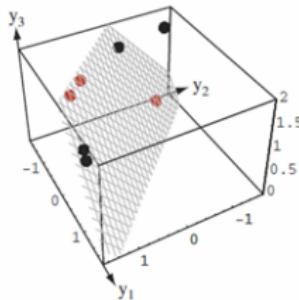
Nonlinear feature mapping

- The multilayer neural networks provide nonlinear mapping of the input to the feature representation at the hidden units
- With small initial weights, the net activation of each hidden unit is small, and thus the linear portion of their activation function is used. Such a linear transformation from \mathbf{x} to \mathbf{y} leaves the patterns linearly inseparable in the XOR problem.
- As learning progresses and the input-to-hidden weights increase in magnitude, the nonlinearities of the hidden units warp and distort the mapping from input to the hidden unit space
- The linear decision boundary at the end of learning found by the hidden-to-output weights is shown by the straight dashed line; the nonlinearly separable problem at the inputs is transformed into a linearly separable at the hidden units

Nonlinear feature mapping



2-2-1



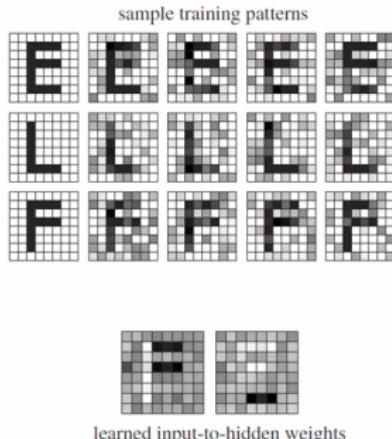
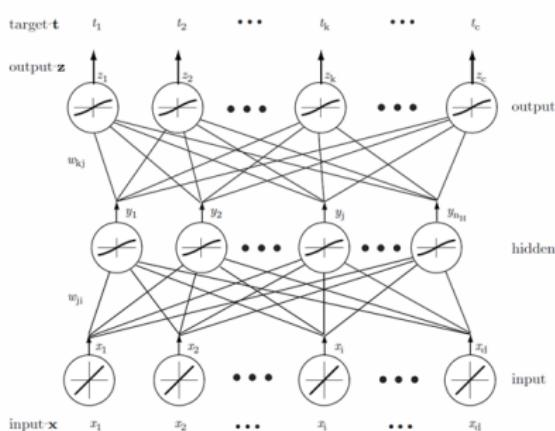
2-3-1

(Duda et al. Pattern Classification 2000)

- The expressive power of the 2-2-1 network is not high enough to separate all the seven patterns, even after the global minimum error is reached by training.
- These patterns can be separated by increasing one more hidden unit to enhance the expressive power.

Filter learning

- The input-to-hidden weights at a single hidden unit describe the input patterns that leads to maximum activation of that hidden unit, analogous to a “matched filter”
- Hidden units find feature groupings useful for the linear classifier implemented by the hidden-to-output layer weights



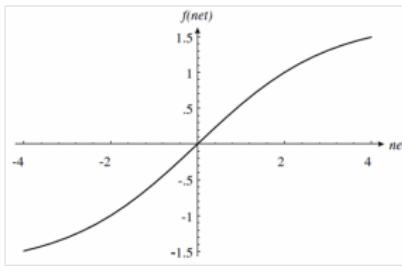
Filter learning

- The top images represent patterns from a large training set used to train a 64-2-3 neural network for classifying three characters
- The bottom figures show the input-to-hidden weights, represented as patterns, at the two hidden units after training
- One hidden unit is tuned to a pair of horizontal bars while the other is tuned to a single lower bar
- Both of these feature groups are useful building blocks for the pattern presented

A recommended sigmoid function for activation function

$$f(x) = 1.79159 \frac{e^{\frac{2}{3}x} - e^{-\frac{2}{3}x}}{e^{\frac{2}{3}x} + e^{-\frac{2}{3}x}}$$

$f(\pm 1) = \pm 1$, liner in the range of $-1 < x < 1$, $f''(x)$ has extrema near $x = \pm 1$.



Y. LeCun, Generalization and network design strategies. *Proc. Int'l Cibf, Cinectuibusn ub Oersoectuve*, 1988.

Desirable properties of activation functions

- Must be nonlinear: otherwise it is equivalent to a linear classifier
- Its output has maximum and minimum value: keep the weights and activations bounded and keep training time limited
 - Desirable property when the output is meant to represent a probability
 - Desirable property for models of biological neural networks, where the output represents a neural firing rate
 - May not be desirable in networks for regression, where a wide dynamic range may be required
- Continuous and differentiable **almost everywhere**
- Monotonicity: otherwise it introduces additional local extrema in the error surface
- Linearity for a small value of *net*, which will enable the system to implement a linear model if adequate for yielding low error

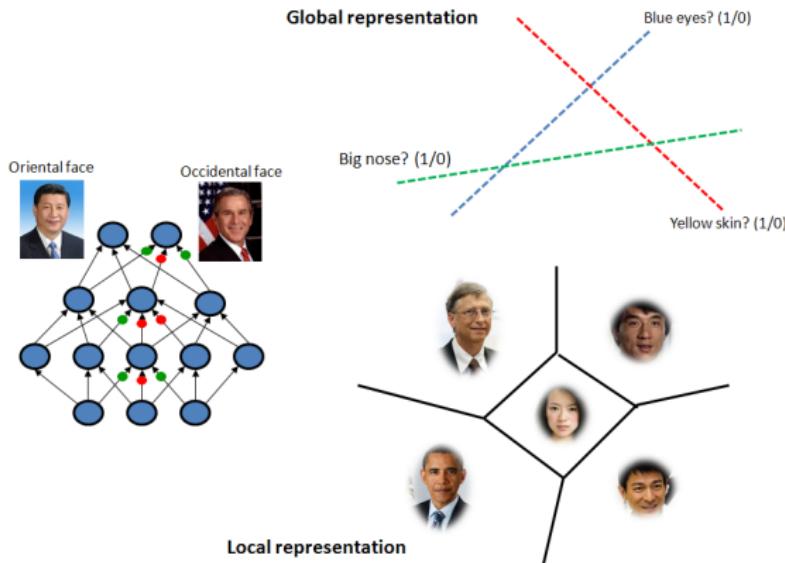
Desirable properties of activation functions

- The average of the outputs at a node is close to zero because these outputs are the inputs to the next layer
- The variance of the outputs at a node is also 1.

Global versus local representations

- Tanh function (shifting the center of *sigmoid* to 0) has all the properties above
 - It has large response for input in a large range. Any particular input \mathbf{x} is likely to yield activity through several hidden units. This affords a **distributed** or **global** representation of the input.
 - If hidden units have activation functions that have significant response only for input within a small range, then an input \mathbf{x} generally leads to fewer hidden units being active - a **local** representation.
 - Distributed representations are superior because more of the data influences the posteriors at any given input region.
 - The global representation can be better achieved with more layers

Global versus local representations



Choosing target values

- Avoid setting the target values as the sigmoid's asymptotes
 - Since the target values can only be achieved asymptotically, it drives the output and therefore the weights to be very large, which the sigmoid derivative is close to zero. So the weights may become stuck.
 - When the outputs saturate, the network gives no indication of confidence level. Large weights force all outputs to the tails of the sigmoid instead of being close to decision boundary.
- Insure that the node is not restricted to only the linear part of the sigmoid
- Choose target values at the point of the maximum second derivative on the sigmoid so as to avoid saturating the output units

Initializing weights

- Randomly initialize weights in the linear region. But they should be large enough to make learning proceed.
 - The network learns the linear part of the mapping before the more difficult nonlinear parts
 - If weights are too small, gradients are small, which makes learning slow
- To obtain a standard deviation close to 1 at the output of the first hidden layer, we just need to use the recommended sigmoid and require that the input to the sigmoid also have a standard deviation $\sigma_y = 1$. Assuming the inputs to a unit are uncorrelated with variance 1, the standard deviation of σ_{y_i} is

$$\sigma_{y_i} = \left(\sum_{j=1}^m w_{ij}^2 \right)^{1/2}$$

Initializing weights

- To ensure $\sigma_{y_i} = 1$, weights should be randomly drawn from a distribution (e.g. uniform) with mean zero and standard deviation

$$\sigma_w = m^{-1/2}$$

Reading materials

- R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern Classification," Chapter 6, 2000.
- Y. Bengio, I. J. GoodFellow, and A. Courville, "Feedforward Deep Networks," Chapter 6 in "Deep Learning", Book in preparation for MIT Press, 2014.

Reference

- W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-133, 1943.
- F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386-408, 1958.
- D. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by back-propagating errors. *Nature*, 323:533-536, 1986.
- Y. LeCun. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pages 599-604, Paris, France, 1985.
- Y. LeCun. Learning processes in an asymmetric threshold network. In Elie Bienenstock, Francoise Fogelman-Soulie, and Gerard Weisbuch, editors, *Disordered systems and biological organization*, pages 233-240, Les Houches, France, 1986. Springer-Verlag.
- R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, Vol. 1, pages 593-605, 1989.

Reference

- K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4): 93-202, 1980.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86 (11): 2278-2324, 1998.
- G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527-1554, 2006.
- G. E. Hinton, and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504-507, 2006.
- G. E. Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, Vol. 11, pp. 428-434, 2007.

Convolutional Nueral Network

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

February 2, 2015

Outline

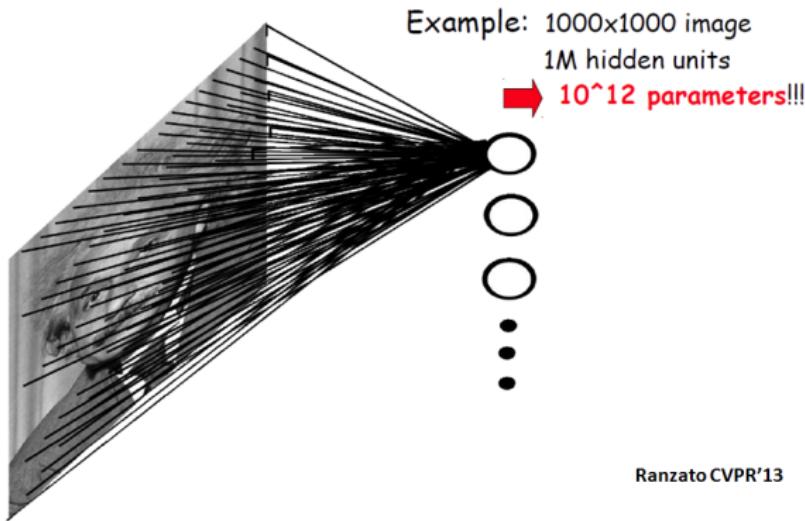
- 1 Convolutional Neural Network (CNN)
- 2 Different CNN structures for image classification
- 3 CNN for pixelwise classification

Convolutional neural network

- Specially designed for data with grid-like structures (LeCun et al. 98)
 - 1D grid: sequential data
 - 2D grid: image
 - 3D grid: video, 3D image volume
- Beat all the existing computer vision technologies on object recognition on ImageNet challenge with a large margin in 2012

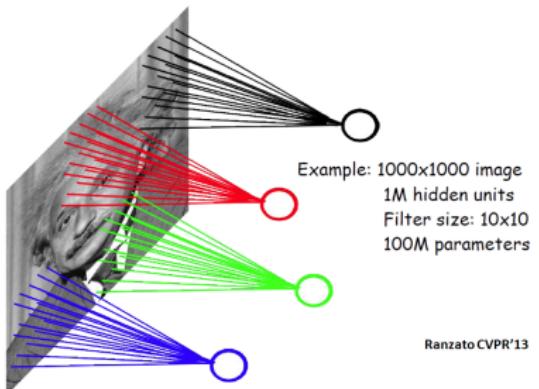
Problems of fully connected neural networks

- Every output unit interacts with every input unit
- The number of weights grows largely with the size of the input image
- Pixels in distance are less correlated



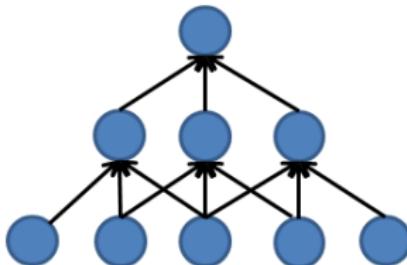
Locally connected neural networks

- Sparse connectivity: a hidden unit is only connected to a local patch (weights connected to the patch are called filter or kernel)
- It is inspired by biological systems, where a cell is sensitive to a small sub-region of the input space, called a receptive field. Many cells are tiled to cover the entire visual field.
- The design of such sparse connectivity is based on domain knowledge.
(Can we apply CNN in frequency domain?)



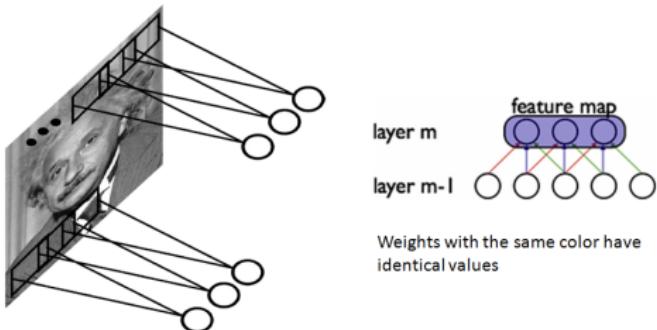
Locally connected neural networks

- The learned filter is a spatially local pattern
- A hidden node at a higher layer has a larger receptive field in the input
- Stacking many such layers leads to “filters”(not anymore linear) which become increasingly “global”



Shared weights

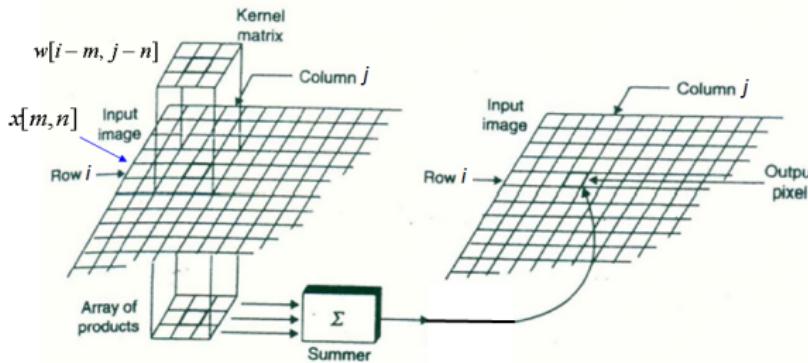
- Translation invariance: capture statistics in local patches and they are independent of locations
 - Similar edges may appear at different locations
- Hidden nodes at different locations share the same weights. It greatly reduces the number of parameters to learn
- In some applications (especially images with regular structures), we may only locally share weights or not share weights at top layers



Convolution

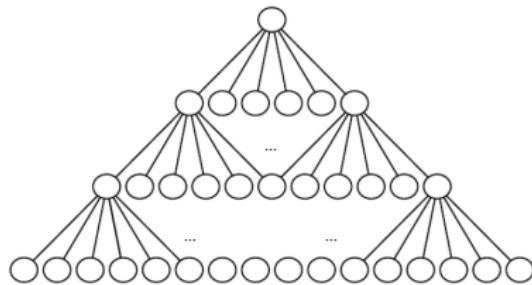
- Computing the responses at hidden nodes is equivalent to convoluting the input image \mathbf{x} with a learned filter \mathbf{w}
- After convolution, a filter map net is generated at the hidden layer
- Parameter sharing causes the layer to have *equivariance* to translation. A function $f(x)$ is equivalent to a function g if $f(g(x)) = g(f(x))$
- Is convolution equivariant to changes in the scale or rotation?

$$net[i, j] = (\mathbf{x} * \mathbf{w})[i, j] = \sum_m \sum_n x[m, n] w[i - m, j - n]$$



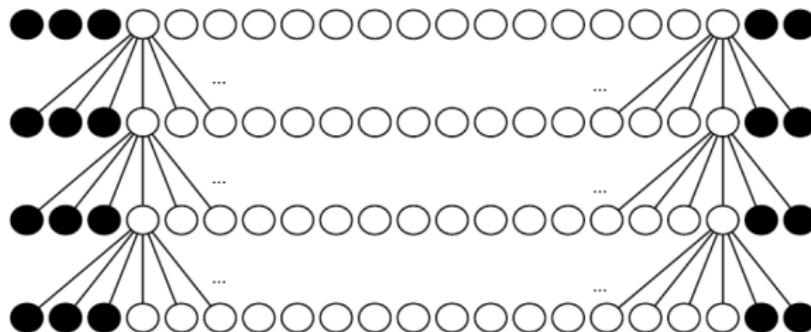
Zero-padding in convolutional neural network (optional)

- The valid feature map is smaller than the input after convolution
- Implementation of neural networks needs to zero-pad the input \mathbf{x} to make it wider
- Without zero-padding, the width of the representation shrinks by the filter width - 1 at each layer
- To avoid shrinking the spatial extent of the network rapidly, small filters have to be used



Zero-padding in convolutional neural network (optional)

- By zero-padding in each layer, we prevent the representation from shrinking with depth. It allows us to make an arbitrarily deep convolutional network



(Bengio et al. Deep Learning 2014)

Downsampled convolutional layer (optional)

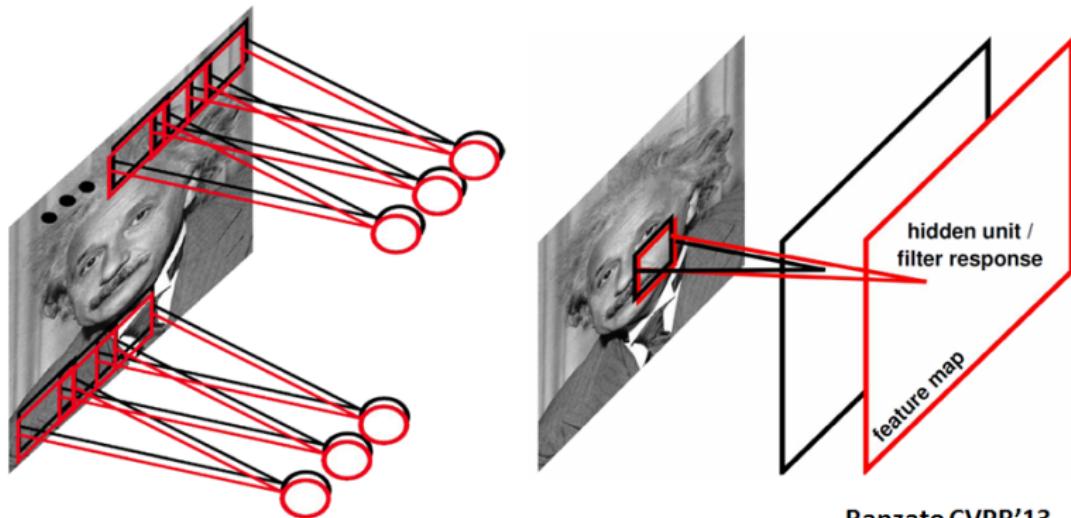
- To reduce computational cost, we may want to skip some positions of the filter and sample only every s pixels in each direction. A downsampled convolution function is defined as

$$net[i, j] = (\mathbf{x} * \mathbf{w})[i \times s, j \times s]$$

- s is referred as the *stride* of this downsampled convolution

Multiple filters

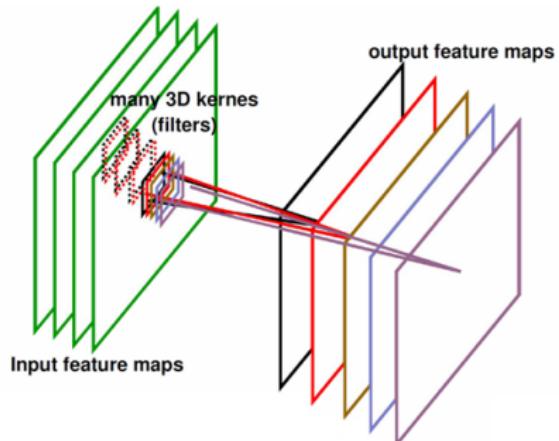
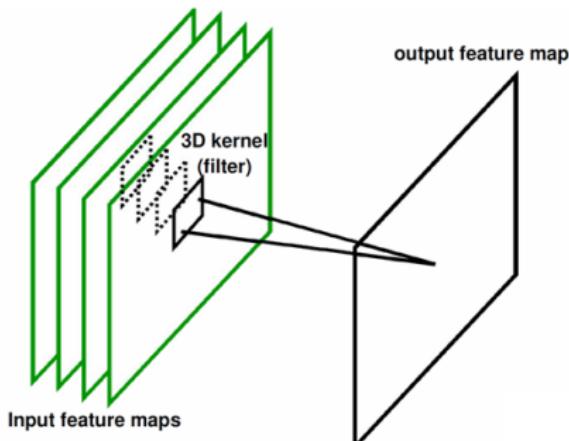
- Multiple filters generate multiple feature maps
- Detect the spatial distributions of multiple visual patterns



Ranzato CVPR'13

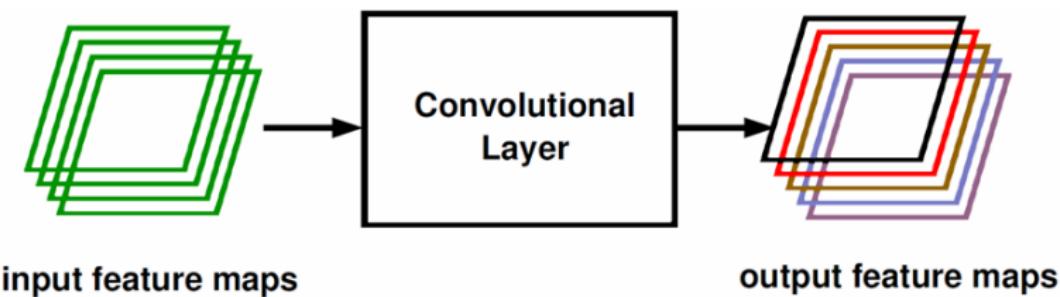
3D filtering when input has multiple feature maps

$$net = \sum_{k=1}^K \mathbf{x}^k * \mathbf{w}^k$$



Ranzato CVPR'13

Convolutional layer



Ranzato CVPR'13

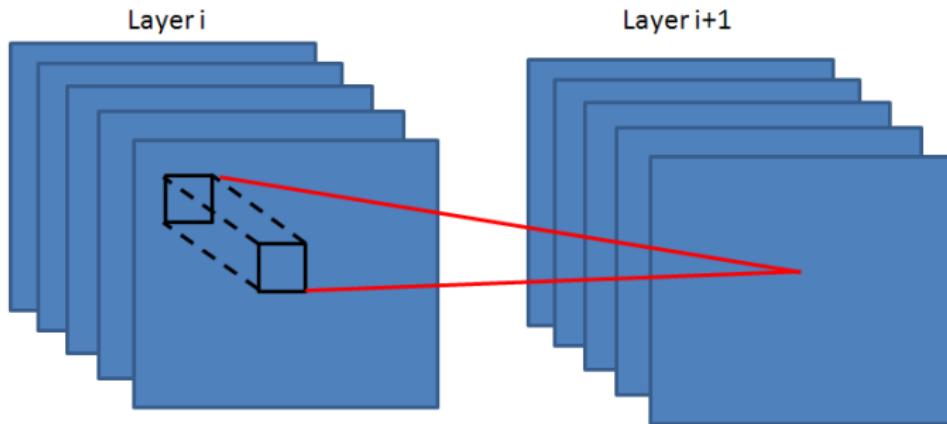
Nonlinear activation function

- $\tanh()$
- Rectified linear unit

Local contrast normalization

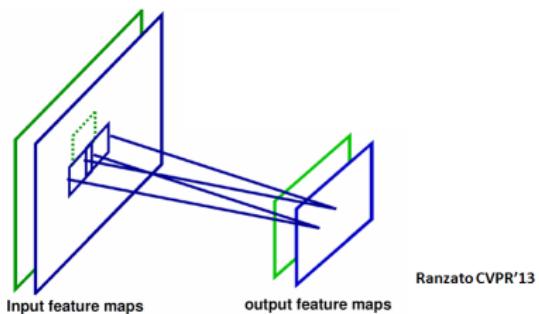
- Normalization can be done within a neighborhood along both spatial and feature dimensions

$$h_{i+1,x,y,k} = \frac{h_{i,x,y,k} - m_{i,N(x,y,k)}}{\sigma_{i,N(x,y,k)}}$$



Pooling

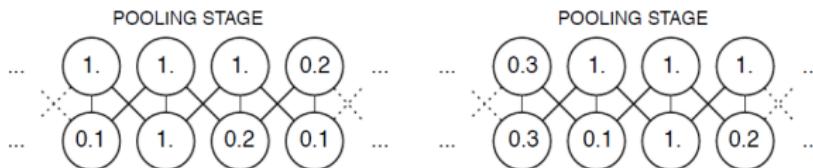
- Max-pooling partitions the input image into a set of rectangles, and for each sub-region, outputs the maximum value
- Non-linear down-sampling
- The number of output maps is the same as the number of input maps, but the resolution is reduced
- Reduce the computational complexity for upper layers and provide a form of translation invariance
- Average pooling can also be used



Ranzato CVPR'13

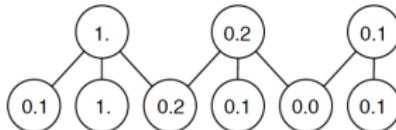
Pooling

- Pooling without downsampling (stride $s = 1$)
- Invariance vs. information loss (even if the resolution is not reduced)
- Pooling is useful if we care more about whether some feature is present than exactly where it is. It depends on applications.



(Bengio et al. Deep Learning 2014)

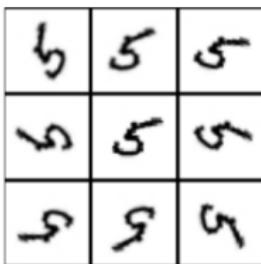
- Pooling with downsampling (commonly used)
- Improve computation efficiency



(Bengio et al. Deep Learning 2014)

Possible extension of pooling

- If we pool over the outputs of separately parameterized convolutions, the features can learn which transformations to become invariant to
- How to achieve scaling invariance?

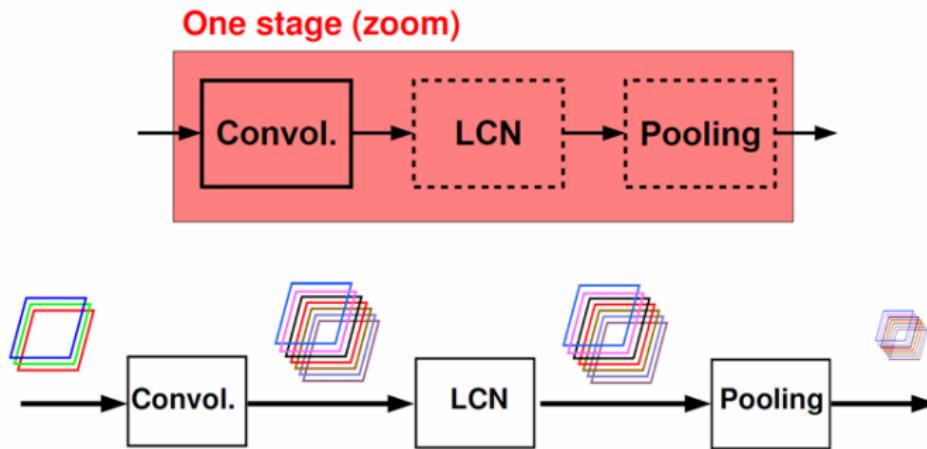


(Bengio et al. Deep Learning 2014)

Example of learned invariances: If each of these filters drive units that appear in the same max-pooling region, then the pooling unit will detect "5"s in any rotation. By learning to have each filter be a different rotation of the "5" template, this pooling unit has learned to be invariant to rotation. This is in contrast to translation invariance, which is usually achieved by hard-coding the net to pool over shifted versions of a single learned filter.

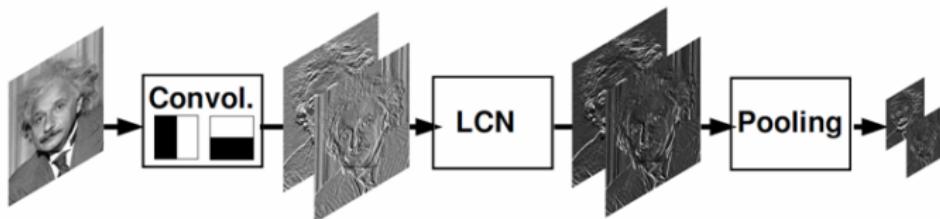
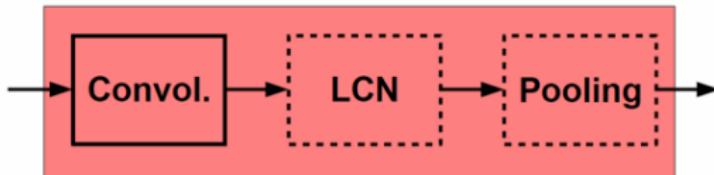
Typical architecture of CNN

- Convolutional layer increases the number of feature maps
- Pooling layer decreases spatial resolution
- LCN and pooling are optional at each stage



Ranzato CVPR'13

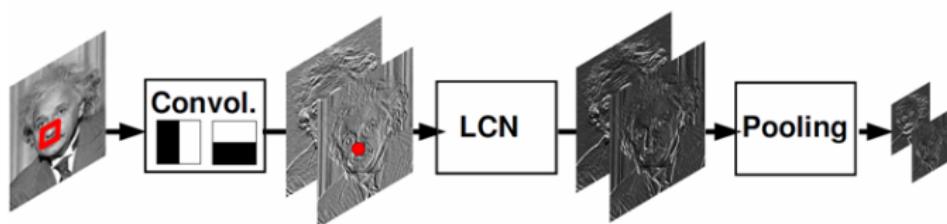
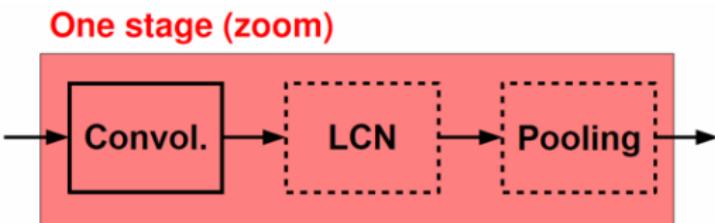
Typical architecture of CNN



Example with only two filters.

Ranzato CVPR'13

Typical architecture of CNN

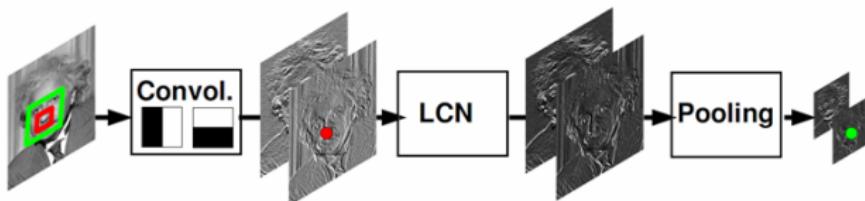
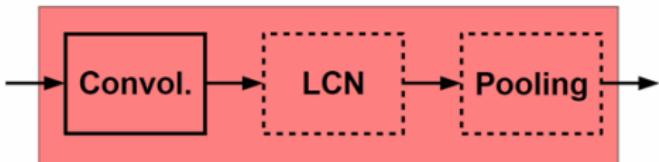


A hidden unit in the first hidden layer is influenced by a small neighborhood (equal to size of filter).

Ranzato CVPR'13

Typical architecture of CNN

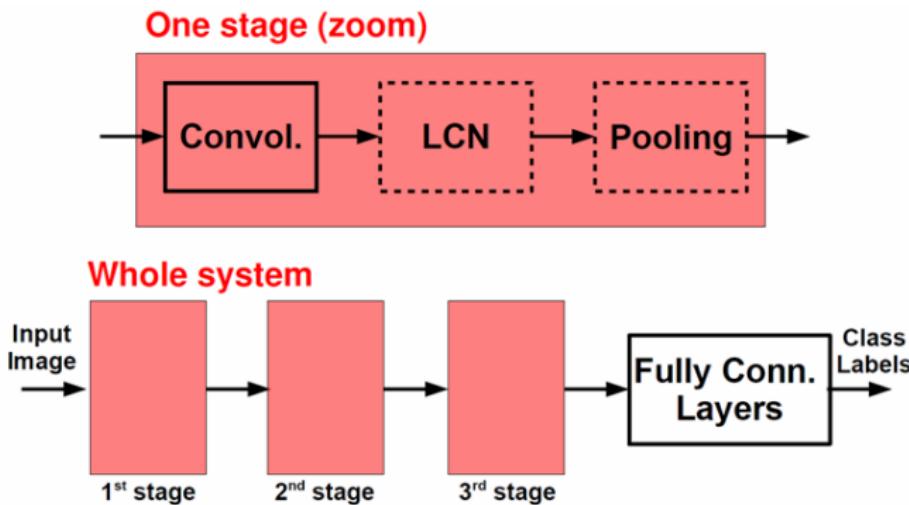
One stage (zoom)



A hidden unit after the pooling layer is influenced by a larger neighborhood
(it depends on filter sizes and the sizes of pooling regions)

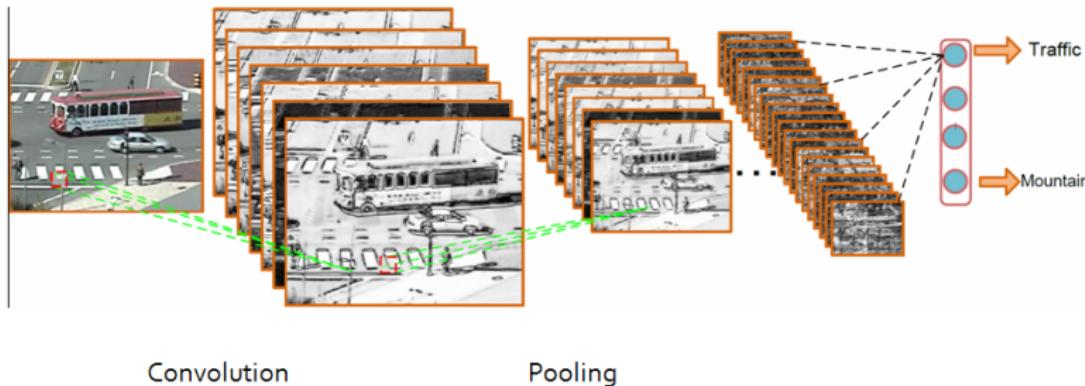
Ranzato CVPR'13

Typical architecture of CNN



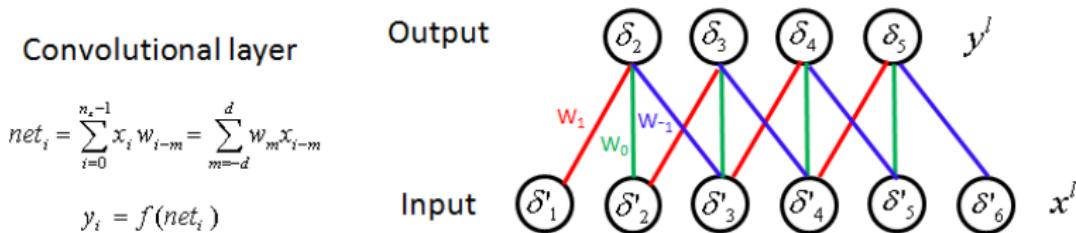
After a few stages, residual spatial resolution is very small.
We have learned a descriptor for the whole image. **Ranzato CVPR'13**

Typical architecture of CNN



BP on CNN

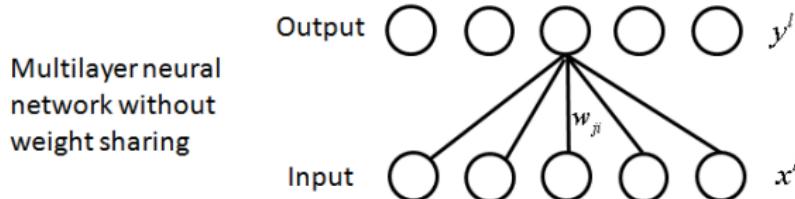
- Calculate sensitivity (back propagate errors) $\delta = -\frac{\partial J}{\partial \text{net}}$ and update weights in the convolutional layer and pooling layer
- Calculating sensitivity in the convolutional layer is the same as multilayer neural network



CNN has multiple convolutional layers. Each convolutional layer l has an input feature map (or image) \mathbf{x}^l and also an output feature map \mathbf{y}^l . The sizes (n_x^l and n_y^l) of the input and output feature maps, and the filter size d^l are different for different convolutional layers. Each convolutional layers has multiple filters, input feature maps and output feature maps. To simplify the notation, we skip the index (l) of the convolutional layer, and assume only one filter, one input feature map and one output feature map.

Calculate $\frac{\partial \text{net}}{\partial w}$ in the convolutional layer

- It is different from neural networks without weight sharing, where each weight W_{ij} is only related to one input node and one output node



$$\text{net}_j = \sum_i w_{ji} x_i \quad \frac{\partial \text{net}_j}{\partial w_{ji}} = x_i$$

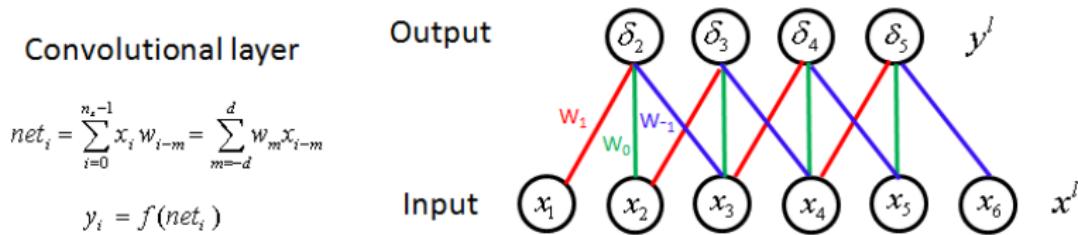
- Taking 1D data as example, in CNN, assume the input layer $\mathbf{x} = [x_0, \dots, x_{n_x-1}]$ is of size n_x and the filter $\mathbf{w} = [w_{-d}, \dots, w_d]$ is of size $2 \times d + 1$. With weight sharing, each weight is shared by multiple input and output nodes

$$\text{net}_j = \sum_{m=-d}^d w_m x_{j-m}$$

Update filters in the convolutional layer

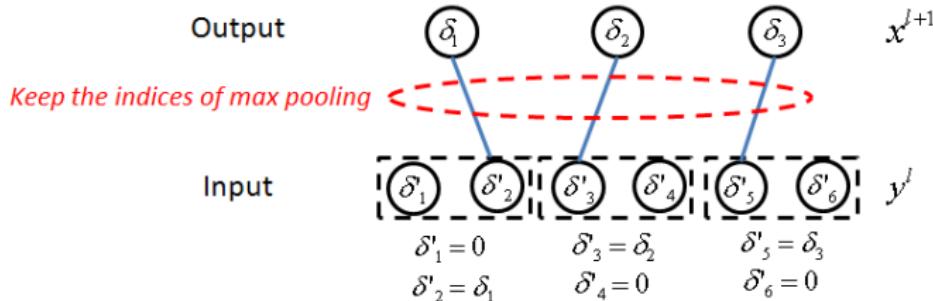
$$\frac{\partial J}{\partial w_m} = \sum_j \frac{\partial J}{\partial net_j} \frac{\partial net_j}{\partial w_m} = - \sum \delta_j x_{j-m}$$

- The gradient can be calculated from the correlation between the sensitivity map and the input feature map



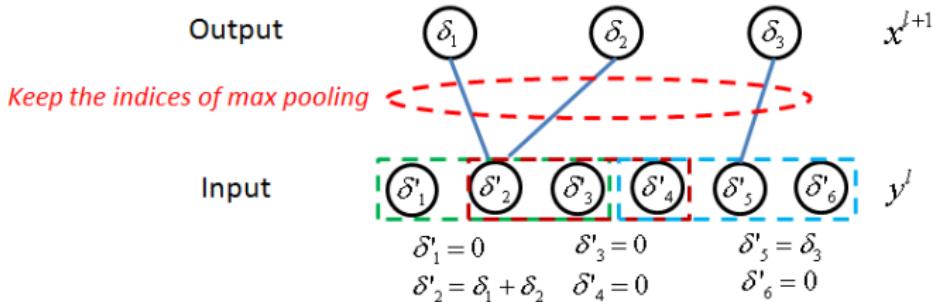
Calculate sensitivities in the pooling layer

- The input of a pooling layer l is the output feature map y^l of the previous convolutional layer. The output x^{l+1} of the pooling layer is the input of the next convolutional layer $l + 1$
- For max pooling, the sensitivity is propagated according to the corresponding indices built during max operation. If max pooling regions are nonoverlapped,



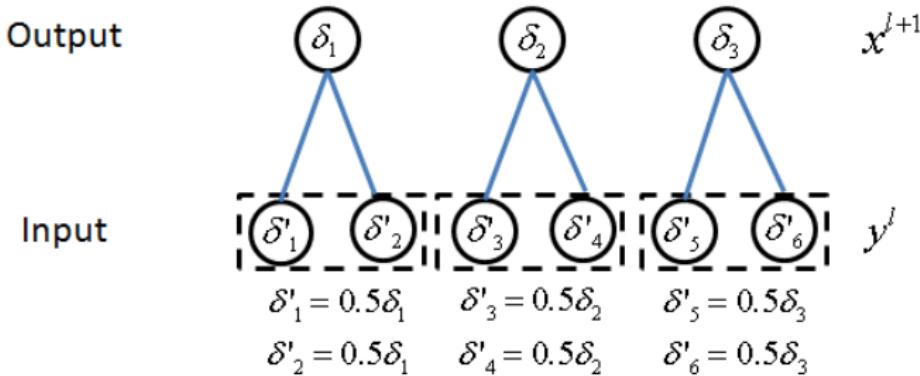
Calculate sensitivities in the pooling layer

- If pooling regions are overlapped and one node in the input layer corresponds to multiple nodes in the output layer, the sensitivities are added



Calculate sensitivities in the pooling layer

- Average pooling



- What if average pooling and pooling regions are overlapped?
- There is no weight to be updated in the pooling layer

Different CNN structures for image classification

- AlexNet
- Clarifai
- Overfeat
- VGG
- DeepImage of Baidu
- Network-in-network
- GoogLeNet

CNN for object recognition on ImageNet challenge

- Krizhevsky, Sutskever, and Hinton, NIPS 2012
- Trained on one million images of 1000 categories collected from the web with two GPU. 2GB RAM on each GPU. 5GB of system memory
- Training lasts for one week
- Google and Baidu announced their new visual search engines with the same technology six months after that
- Google observed that the accuracy of their visual search engine was doubled

Rank	Name	Error rate	Description
1	U. Toronto	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	Bottleneck.

ImageNet

1000 object classes that we recognize

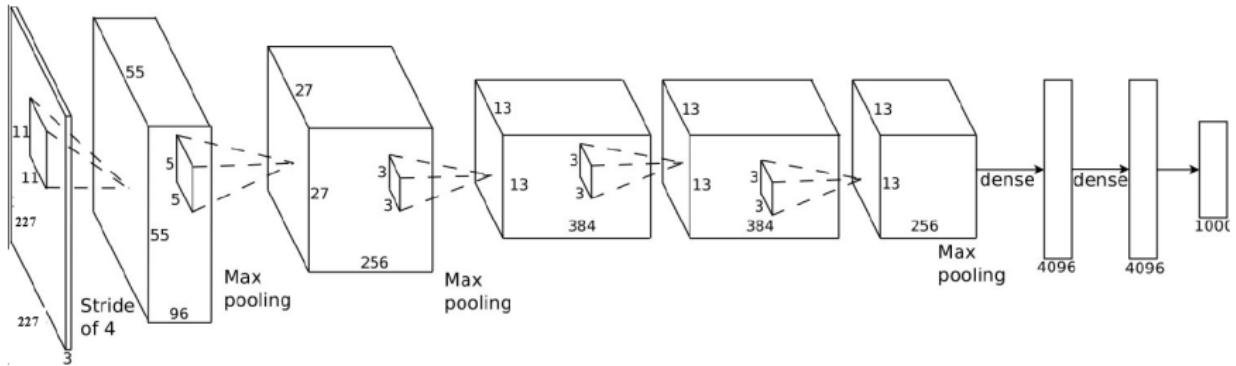


poster created by Fengjun Lv using VIPBase

Images courtesy of ImageNet (<http://www.image-net.org/challenges/LSVRC/2010/index>)

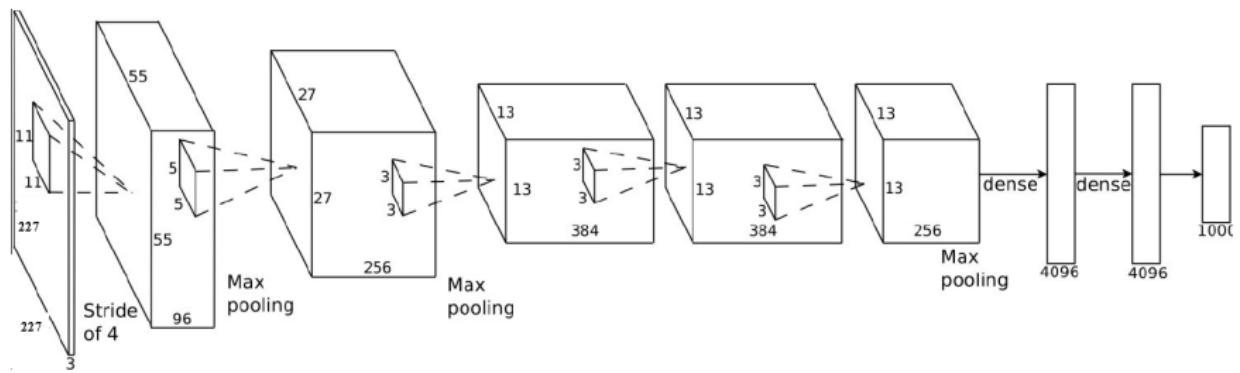
Model architecture-AlexNet Krizhevsky 2012

- 5 convolutional layers and 2 fully connected layers for learning features.
- Max-pooling layers follow first, second, and fifth convolutional layers
- The number of neurons in each layer is given by 253440, 186624, 64896, 64896, 43264, 4096, 4096, 1000
- 650000 neurons, 60000000 parameters, and 6300000000 connections



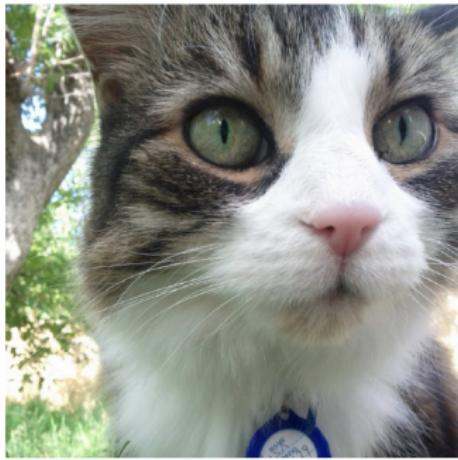
Model architecture-AlexNet Krizhevsky 2012

- The first time deep model is shown to be effective on large scale computer vision task.
- The first time a very large scale deep model is adopted.
- GPU is shown to be very effective on this large deep model.



Technical details

- Normalize the input by subtracting the mean image on the training set



An input image (256x256)



Minus sign



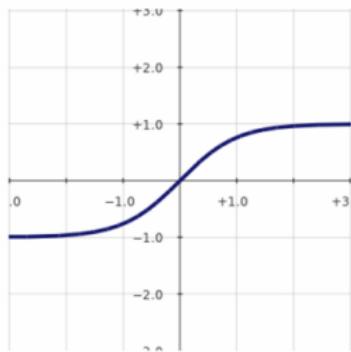
The mean input image

(Krizhevsky NIPS 2014)

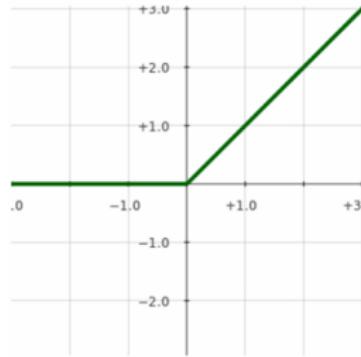
Technical details

- Choice of activation function

$$f(x) = \tanh(x)$$



$$f(x) = \max(0, x)$$

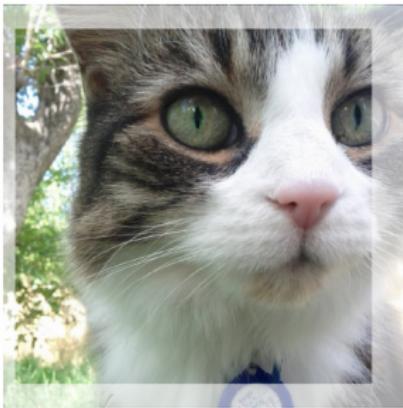


Very bad (slow to train)

Very good (quick to train)

Technical details

- Data augmentation
 - The neural net has 60M real-valued parameters and 650,000 neurons
 - It overfits a lot. 224×224 image regions are randomly extracted from 256 images, and also their horizontal reflections

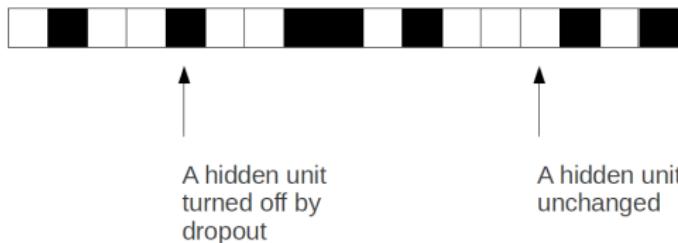


Technical details

- Dropout

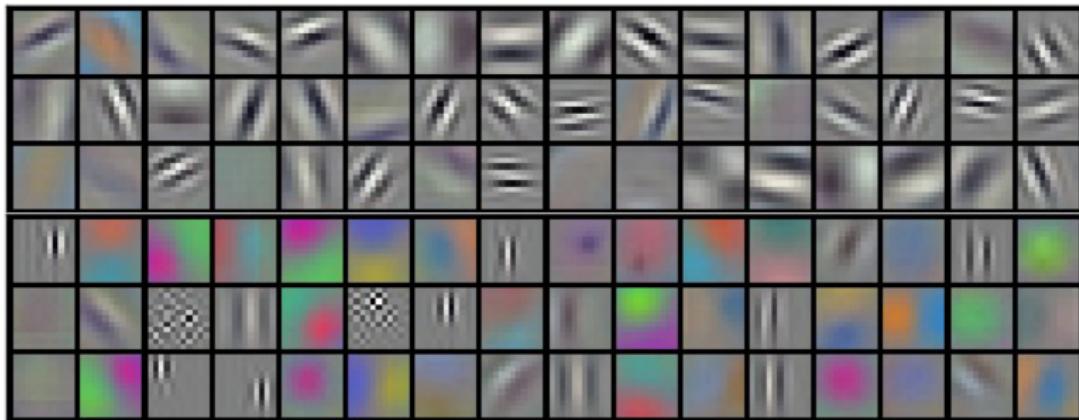
- Independently set each hidden unit activity to zero with 0.5 probability
- Do this in the two globally-connected hidden layers at the net's output

A hidden layer's activity on a given training image



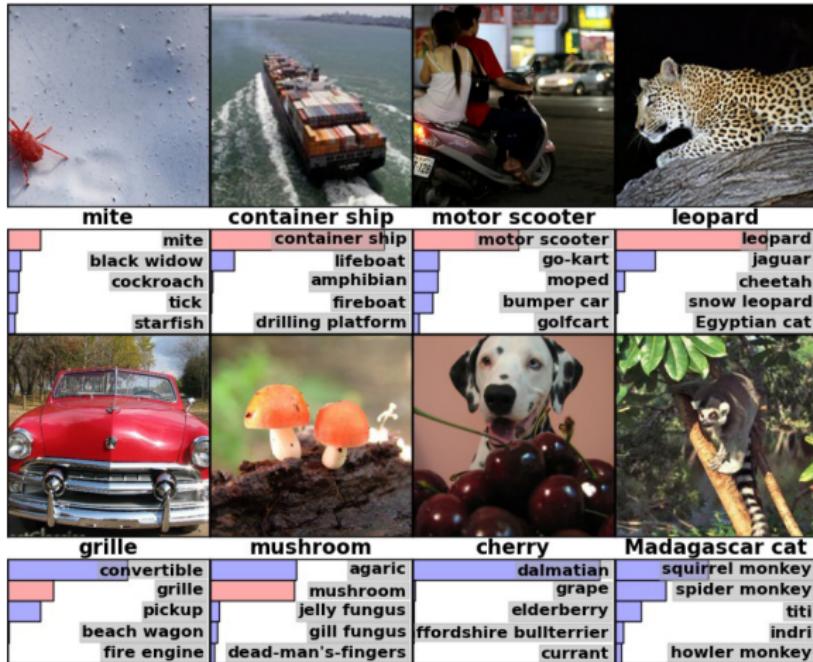
(Krizhevsky NIPS 2014)

96 learned low-level filters

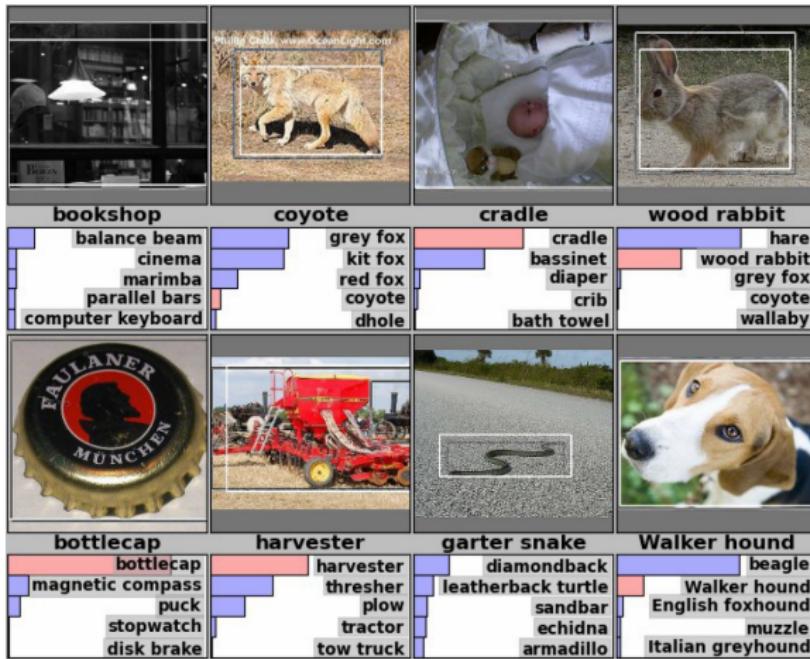


(Krizhevsky NIPS 2014)

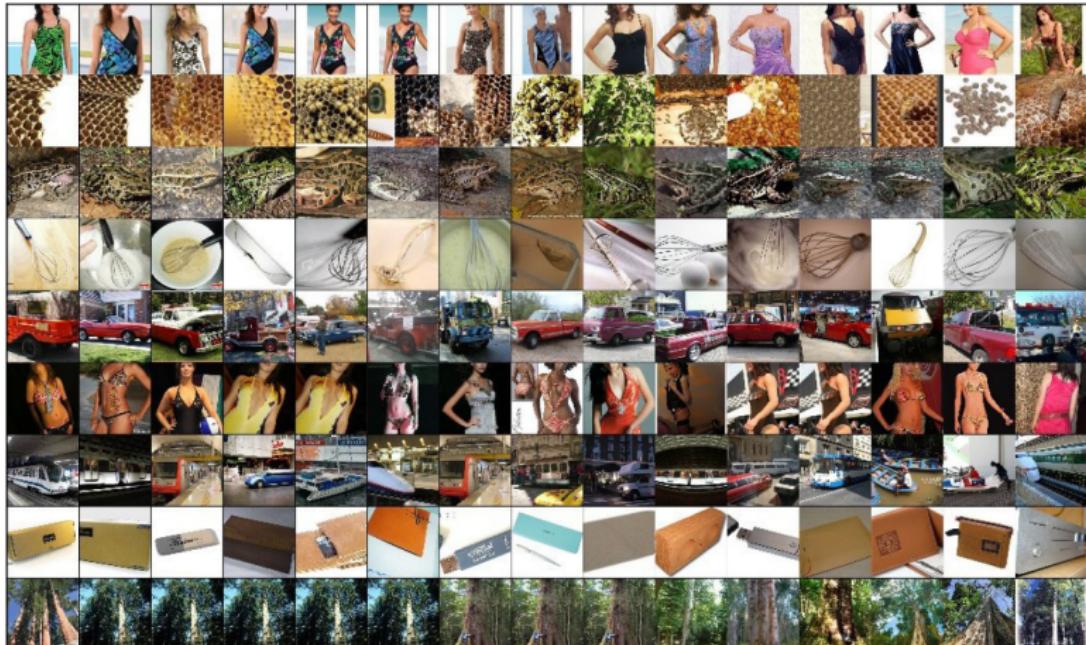
Classification result



Detection result



Top hidden layer can be used as feature for retrieval

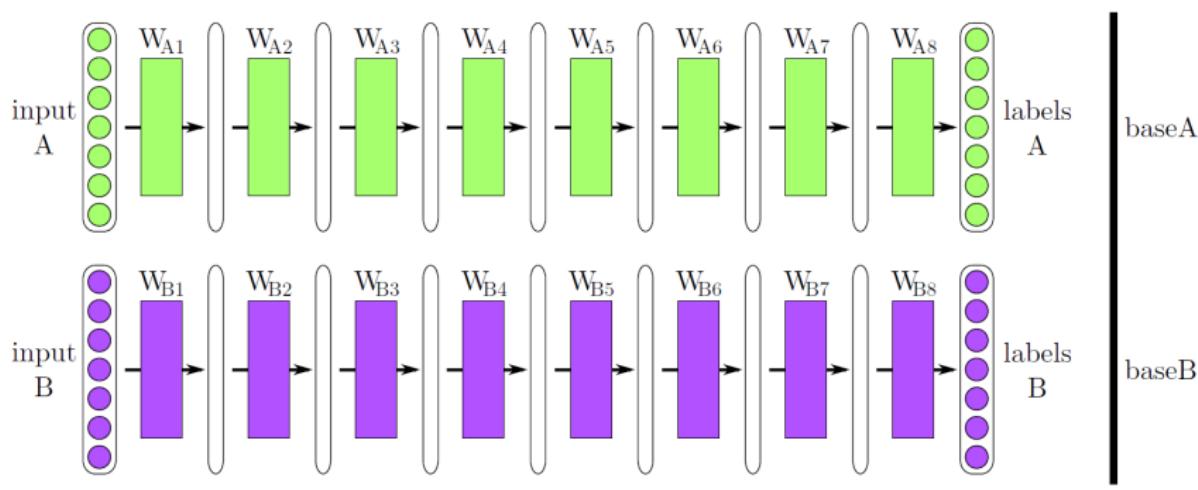


How transferable are features in CNN networks?

- (Yosinski et al. NIPS'14) investigate transferability of features by CNNs
- The transferability of features by CNN is affected by
 - Higher layer neurons are more specific to original tasks
 - Layers within a CNN network might be fragilely co-adapted
- Initializing with transferred features can improve generalization after substantial fine-tuning on a new task

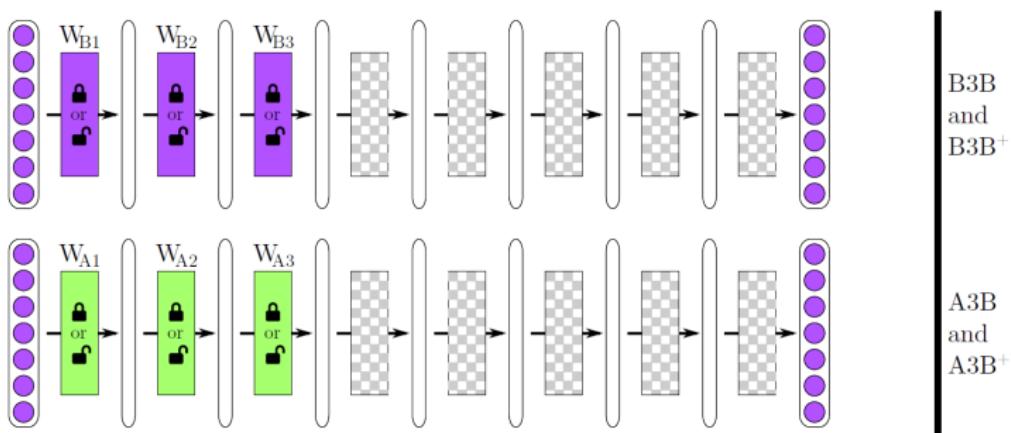
Base tasks

- ImageNet are divided into two groups of 500 classes, A and B
- Two 8-layer AlexNets, baseA and baseB, are trained on the two groups, respectively



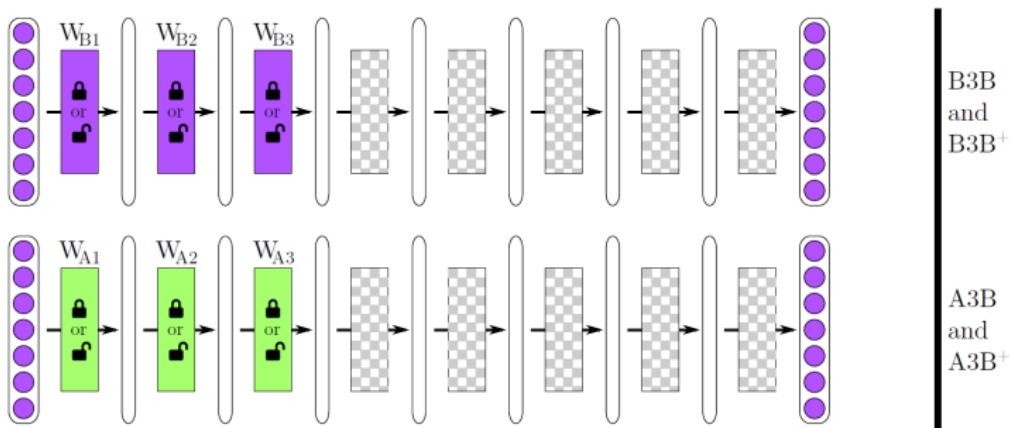
Transfer and selfer networks

- A *selfer* network BnB: the first n layers are copied from baseB and frozen. The other higher layers are initialized randomly and trained on dataset B. This is the control for transfer network
- A *transfer* network AnB: the first n layers are copied from baseA and frozen. The other higher layers are initialized randomly and trained toward dataset B

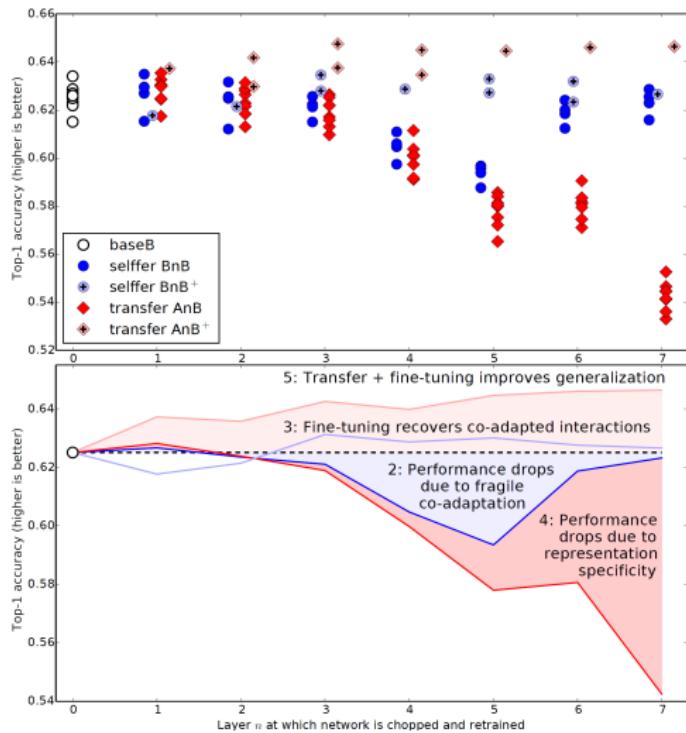


Transfer and selffer networks (cont'd)

- A *selffer* network BnB+: just like BnB, but where all layers learn
- A *transfer* network AnB+: just like AnB, but where all layers learn

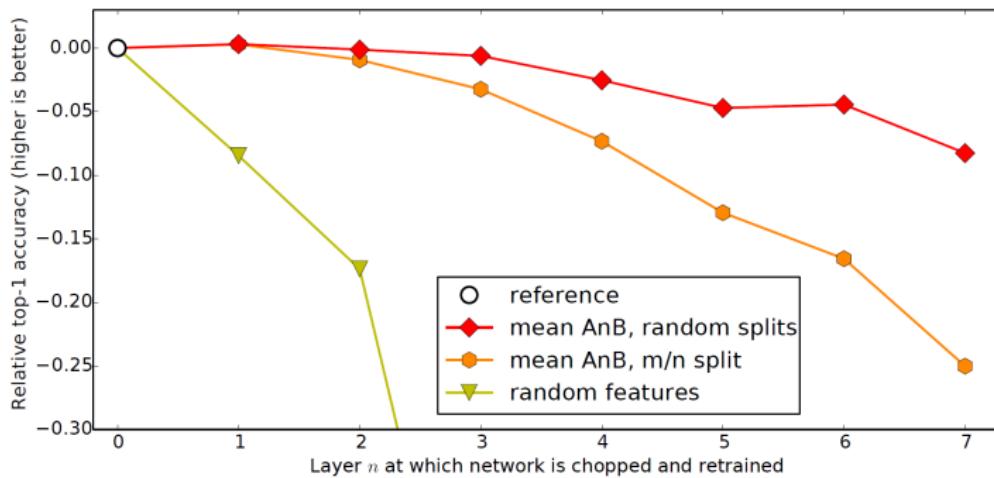


Results



Dissimilar datasets

- Divide ImageNet into man-made objects A (449 classes) and natural objects B (551 classes)
- The transferability of features decreases as the distance between the base task and target task increases



Investigate components of CNNs

- Kernel size
- Kernel (channel) number
- Stride
- Dimensionality of fully connected layers
- Data augmentation
- Model averaging

Investigate components of CNNs (cont'd)

- (Chatfield et al. BMVC'14) pre-train on ImageNet and fine-tune on PASCAL VOC 2007
- Different architectures
 - mAP: CNN-S > (marginally) CNN-M > (~%2.5) CNN-F
- Different data augmentation
 - No augmentation
 - Flipping (almost no improvement)
 - Smaller dimension downsized to 256, cropping 224×224 patches from the center and 4 corners, flipping (~ 3% improvement)

Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8
CNN-F	64x11x11 st. 4, pad 0 LRN, x2 pool	256x5x5 st. 1, pad 2 LRN, x2 pool	256x3x3 st. 1, pad 1	256x3x3 st. 1, pad 1	256x3x3 st. 1, pad 1 x2 pool	4096 drop-out	4096 drop-out	1000 soft-max
CNN-M	96x7x7 st. 2, pad 0 LRN, x2 pool	256x5x5 st. 2, pad 1 LRN, x2 pool	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1 x2 pool	4096 drop-out	4096 drop-out	1000 soft-max
CNN-S	96x7x7 st. 2, pad 0 LRN, x3 pool	256x5x5 st. 1, pad 1 x2 pool	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1 x3 pool	4096 drop-out	4096 drop-out	1000 soft-max

Fast
similar to AlexNet

Medium
similar to Clarifai model

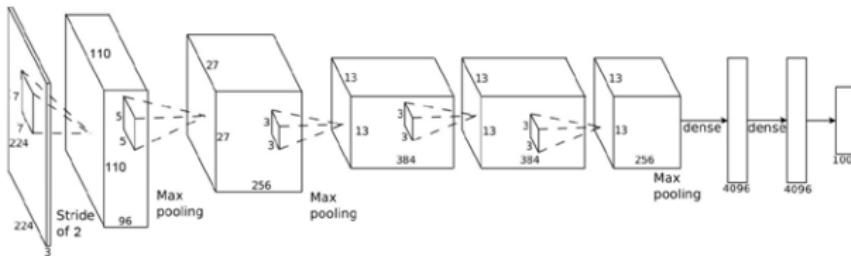
Slow
similar to OverFeat
Accurate model

Investigate components of CNNs (cont'd)

- Gray-scale vs. color ($\sim 3\%$ drop)
- Decrease the number of nodes in FC7
 - to 2048 (surprisingly, marginally better)
 - to 1024 (marginally better)
 - to 128 ($\sim 2\%$ drop but 32x smaller feature)
- Change the softmax regression loss to ranking hinge loss
 - $w_c \phi(I_{pos}) > w_c \phi(I_{neg}) + 1 - \xi$ (ξ is a slack variable)
 - $\sim 2.7\%$ improvement
 - Note, \mathcal{L}_2 normalising features account for $\sim 5\%$ of accuracy for VOC 2007
- On ILSVRC-2012, the CNN-S achieved a top-5 error rate of 13.1%
 - CNN-F: 16.7%
 - CNN-M: 13.7%
 - AlexNet: 17%

Model architecture-Clarifai

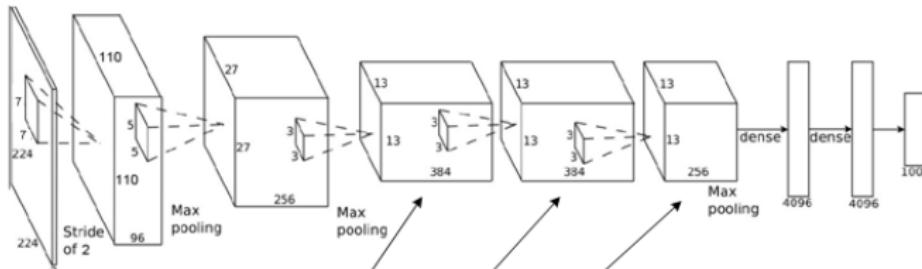
- Winner of ILSVRC 2013
- Max-pooling layers follow first, second, and fifth convolutional layers
- 11×11 to 7×7 , stride 4 to 2 in 1st layer (increasing resolution of feature maps)
- Other settings are the same as AlexNet
- reduce the error by 2%.



Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	-	-	26.2
(Krizhevsky et al., 2012), 1 convnet	40.7	18.2	---
1 convnet for Clarifai	38.4	16.5	---

Model architecture-Clarifai further investigation

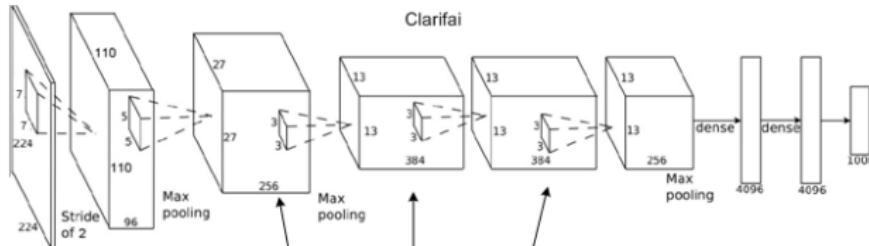
- More maps in the convolutional layers leads to small improvement.
- Model averaging leads to improvement (random initialization).



Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	-	-	26.2
(Krizhevsky et al., 2012), 1 AlexNet	40.7	18.2	---
1 convnet for Clarifai	38.4	16.5	---
5 convnets for Clarifai (a)	36.7	15.3	15.3
1 convnet for Clarifai but with layers 3,4,5: 512,1024,512 maps - (b)	37.5	16.0	16.1
6 convnets, (a) & (b) combined	36.0	14.7	14.8

Model architecture-Overfeat

- Less pooling and more filters (384 => 512 for conv3 and 384=>1024 for conv4/5).



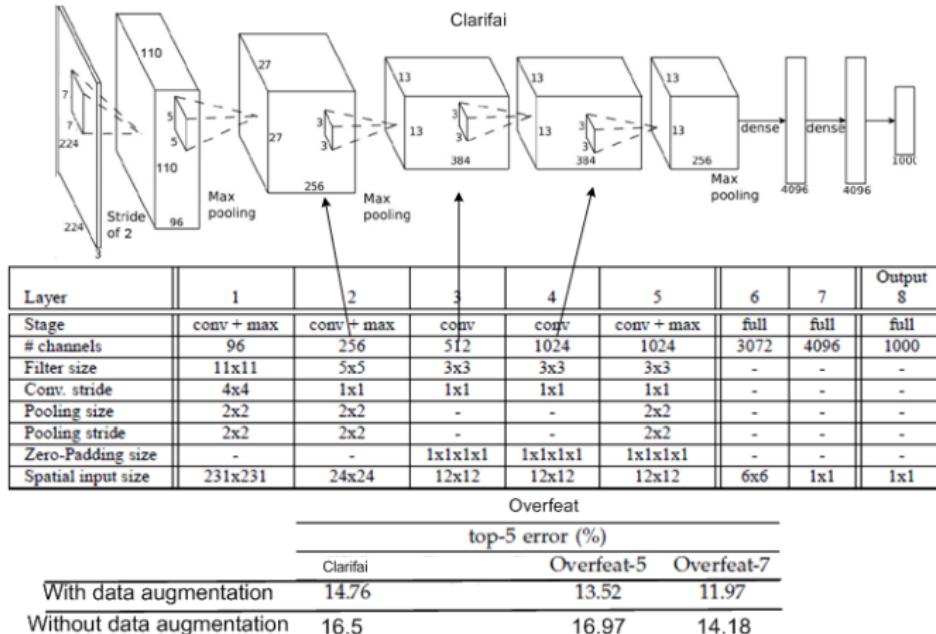
Layer	1	2	3	4	5	6	7	Output 8
Stage	conv + max	conv + max	conv	conv	conv + max	full	full	full
# channels	96	256	512	1024	1024	3072	4096	1000
Filter size	11x11	5x5	3x3	3x3	3x3	-	-	-
Conv. stride	4x4	1x1	1x1	1x1	1x1	-	-	-
Pooling size	2x2	2x2	-	-	2x2	-	-	-
Pooling stride	2x2	2x2	-	-	2x2	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	231x231	24x24	12x12	12x12	12x12	6x6	1x1	1x1

Overfeat

	top-5 error (%)		
	Clarifai	Overfeat-5	Overfeat-7
Without data augmentation	16.5	16.97	14.18

Model architecture-Overfeat

- With data augmentation, more complex model has better performance.



Model architecture-the devil of details

- CNN-F: similar to AlexNet, but less channels in conv3-5.
- CNN-S: the most complex one.
- CNN-M 2048: replace the 4096 features in fc7 by 2048 features. Makes little difference.
- Data augmentation. The input image is downsized so that the smallest dimension is equal to 256 pixels. Then 224×224 crops are extracted from the four corners and the centre of the image.

ILSVRC-2012		(top-5 error)
	(a) Clarifai 1 ConvNet	
(b) CNN F	16.7	
(c) CNN M	13.7	
(d) CNN M 2048	13.5	
(e) CNN S	13.1	

Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8
CNN-F	64x1x11 st. 4, pad 0 LRN, x2 pool	256x5x5 st. 1, pad 2 LRN, x2 pool	256x3x3 st. 1, pad 1	256x3x3 st. 1, pad 1	256x3x3 st. 1, pad 1 x2 pool	4096 drop-out	4096 drop-out	1000 soft-max
CNN-M	96x7x7 st. 2, pad 0 LRN, x2 pool	256x5x5 st. 2, pad 1 LRN, x2 pool	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1 x2 pool	4096 drop-out	4096 drop-out	1000 soft-max
CNN-S	96x7x7 st. 2, pad 0 LRN, x3 pool	256x5x5 st. 1, pad 1 x2 pool	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1 x3 pool	4096 drop-out	4096 drop-out	1000 soft-max
Clarifai	96x7x7 st. 2, LRN,x2 pool	256x5x5 st. 2, pad1 LRN,x2 pool	384x3x3 st. 1, pad1	384x3x3 st. 1, pad1	256x3x3 st. 1, pad1	4096 drop	4096 drop	4096 drop

Model architecture-very deep CNN

- The deep model VGG in 2014.
- Apply 3×3 filter for all layers.
- 11 layers (A) to 19 layers (E).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Model architecture- very deep CNN

- The deep model VGG in 2014.
- Better to have deeper layers. 11 layers (A) => 16 layers (D).
- From 16 layers (D) to 19 layers (E), accuracy does not improve.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
ConvNet config. (Table 1)		smallest image side		top-1 val. error (%)	
		train (S)	test (Q)		
A		256	256	29.6	10.4
A-LRN		256	256	29.7	10.5
B		256	256	28.7	9.9
C	256		256	28.1	9.4
	384		384	28.1	9.3
	[256;512]		384	27.3	8.8
D	256		256	27.0	8.8
	384		384	26.8	8.7
	[256;512]		384	25.6	8.1
E	256		256	27.3	9.0
	384		384	26.9	8.7
	[256;512]		384	25.5	8.0

Model architecture- very deep CNN

- Scale jittering at the training time.
- The crop size is fixed to 224×224 .
- S : the smallest side of an isotropically-rescaled training image.
- Scale jittering at the training time: [256; 512]: randomly select S to be within [256 512].
- LRN: local response normalisation. A-LRN does not improve on A.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
	256	256	28.1	9.4
C	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
	256	256	27.0	8.8
D	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
	256	256	27.3	9.0
E	384	384	26.9	8.7
	[256;512]	384	25.5	8.0

Model architecture- very deep CNN

- Multi-scale averaging at the testing time.
- The crop size is fixed to 224×224 .
- Q : the smallest side of an isotropically-rescaled testing image.
- Running a model over several rescaled versions of a test image (corresponding to different Q), followed by averaging the resulting class posteriors. Improves accuracy ($25.5 \Rightarrow 24.8$).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
	256	224,256,288	27.7	9.2
C	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	24.8	7.5
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	24.8	7.5

Model architecture- DeeplImage of Baidu

- The deep model of Baidu in 2015.
- More hidden nodes at the fully connected layer (FC1-2), upto 8192.
- 16 layers.

Table 4: Single model comparison.

Team	Top-1 val. error	Top-5 val. error
GoogLeNet [21]	-	7.89%
VGG [20]	25.9%	8.0%
Deep Image	24.88%	7.42%

Layers	Conv 1-2	Max pool	Conv 3-4	Max pool	Conv 5-6-7		Max pool
# filters	64		128		256		
Conv 8-9-10	512	Max pool	Conv 11-12-13	Max pool	FC 1-2	FC 3	Softmax
			512		6144	1000	

Model architecture- DeeplImage of Baidu

- The deep model of Baidu in 2015.
- More hidden nodes at the fully connected layer (FC1-2), upto 8192.
- 16 layers.
- Data augmentation.

Table 4: Single model comparison.

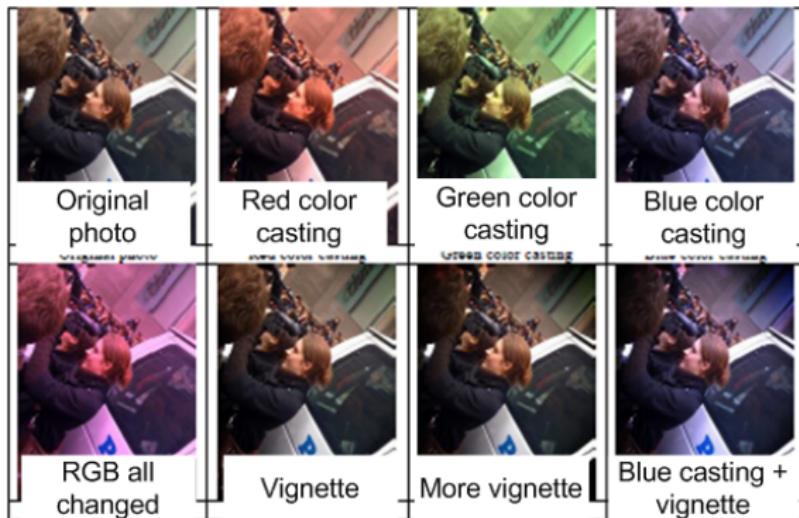
Team	Top-1 val. error	Top-5 val. error
GoogLeNet [21]	-	7.89%
VGG [20]	25.9%	8.0%
Deep Image	24.88%	7.42%

Layers	Conv 1-2	Max pool	Conv 3-4	Max pool	Conv 5-6-7		Max pool
# filters	64		128		256		
Conv 8-9-10	512	Max pool	Conv 11-12-13	Max pool	FC 1-2	FC 3	Softmax
			512		6144	1000	

Augmentation	The number of possible changes
Color casting	68920
Vignetting	1960
Lens distortion	260
Rotation	20
Flipping	2
Cropping	82944(crop size is 224x224, input image size is 512x512)

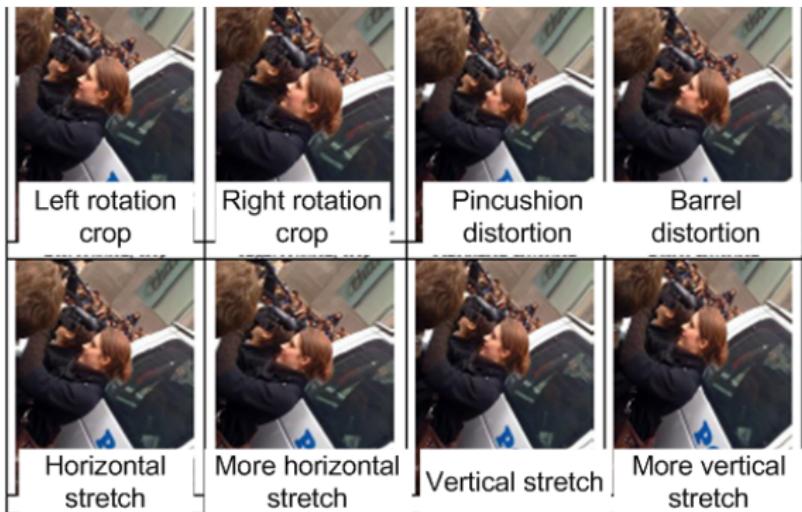
Model architecture- DeeplImage of Baidu

- Data augmentation.



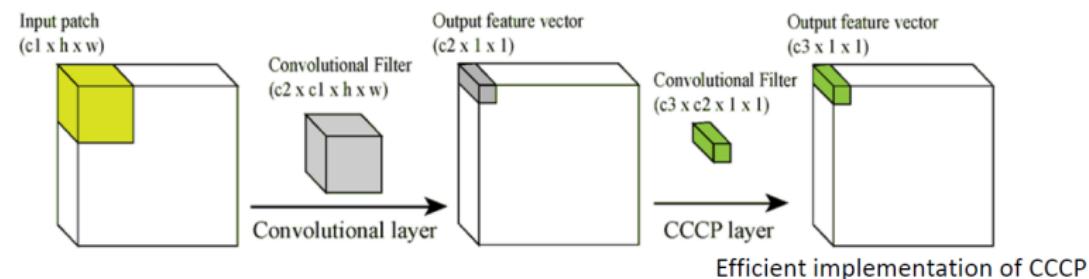
Model architecture- DeeplImage of Baidu

- Data augmentation.



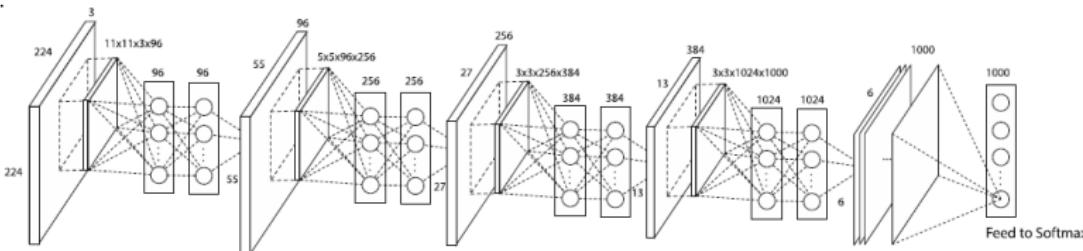
Model architecture- Network in Network

- Use 1×1 filters after each convolutional layer.



Model architecture- Network in Network

- Remove the two fully connected layers (fc6, fc7) of the AlexNet but add NIN into the AlexNet.



	Parameter Number	Performance	Time to train (GTX Titan)
AlexNet	60 Million (230 Megabytes)	40.7% (Top 1)	8 days
NIN	7.5 Million (29 Megabytes)	39.2% (Top 1)	4 days

Model architecture- GoogleNet

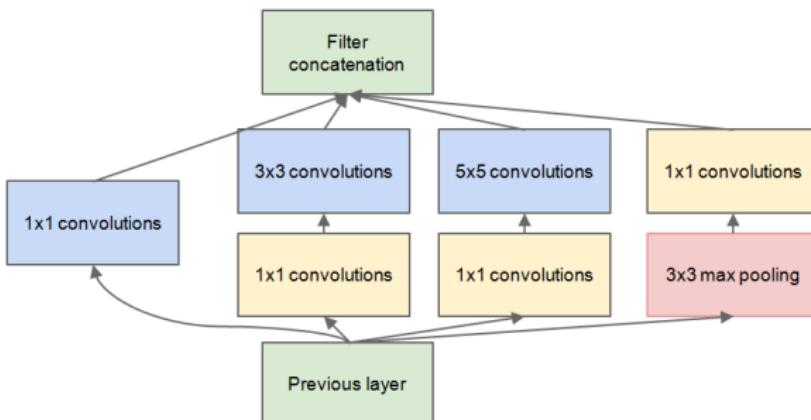
- Inspired by the good performance of NIN.



Google™

Model architecture- GoogleNet

- Inception model.
- Variable filter sizes to capture different visual patterns of different sizes. Enforce sparse connection between previous layer and output.
- The 1×1 convolutions are used for reducing the number of maps from the previous layer.



Model architecture- GoogleNet

- Based on inception model.
- Cascade of inception models.
- Widths of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.



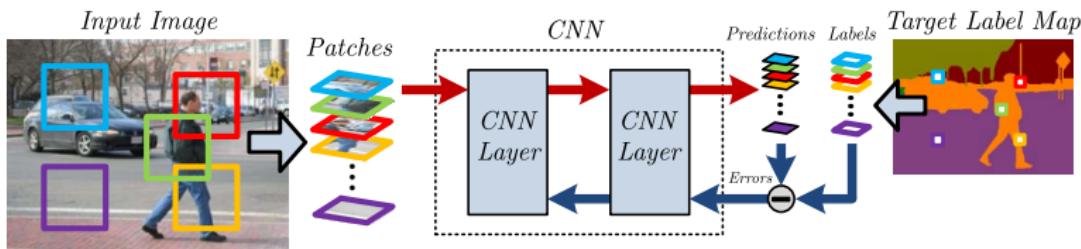
Model architecture- GoogleNet

- Parameters.

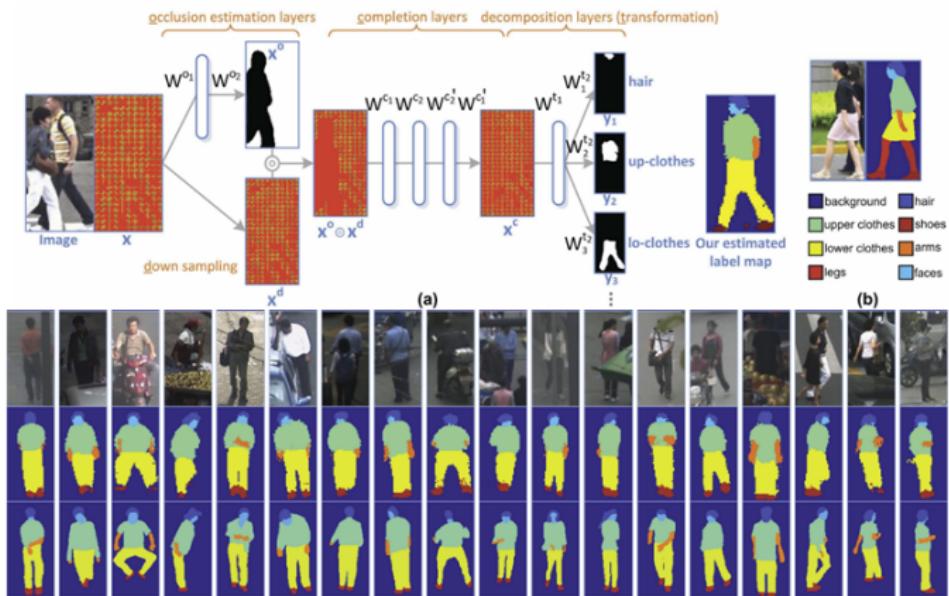
type	patch size/ stride	output size	depth	#1x1	#3x3 reduce	#3x3	#5x5 reduce	#5x5	pool proj	params	ops
convolution	7x7/2	112x112x64	1							2.7K	34M
max pool	3x3/2	56x56x64	0								
convolution	3x3/1	56x56x192	2		64	192				112K	360M
max pool	3x3/2	28x28x192	0								
inception (3a)		28x28x256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28x28x480	2	128	128	192	32	96	64	380K	304M
max pool	3x3/2	14x14x480	0								
inception (4a)		14x14x512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14x14x512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14x14x512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14x14x528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14x14x832	2	256	160	320	32	128	128	840K	170M
max pool	3x3/2	7x7x832	0								
inception (5a)		7x7x832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7x7x1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7x7/1	1x1x1024	0								
dropout (40%)		1x1x1024	0								
linear		1x1x1000	1							1000K	1M
softmax		1x1x1000	0								

CNN for pixelwise classification

- Forward and backward propagation algorithms were proposed for whole-image classification: predicting a single label for a whole image
- Pixelwise classification: predicting a label at every pixel (e.g. segmentation, detection, and tracking)
- For pixelwise classification problems, it is generally trained and tested in a patch-by-patch manner, i.e. cropping a large patch around every pixel and inputting the patch to CNN for prediction (larger patches leading to better performance)
- It involves much redundant computation and is extremely inefficient



Directly Predict Segmentation Maps



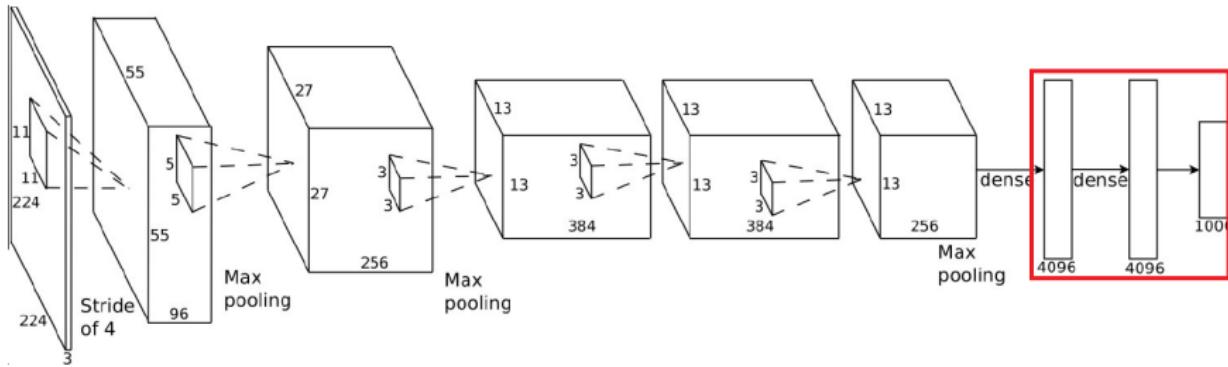
P. Luo, X. Wang, and X. Tang, "Pedestrian Parsing via Deep Decompositional Network," ICCV 2013.

Directly Predict Segmentation Maps

- Classifier is location sensitive has no translation invariance
 - Prediction not only depends on the neighborhood of the pixel, but also its location
- Only suitable for images with regular structures, such as faces and humans

Fully convolutional network

- One solution is to use fully convolutional network (Kang and Wang, arXiv:1411.4464)
- The convolution and pooling kernels can be directly applied to a full input image
- For fully connected layers, they can be converted into convolution layers



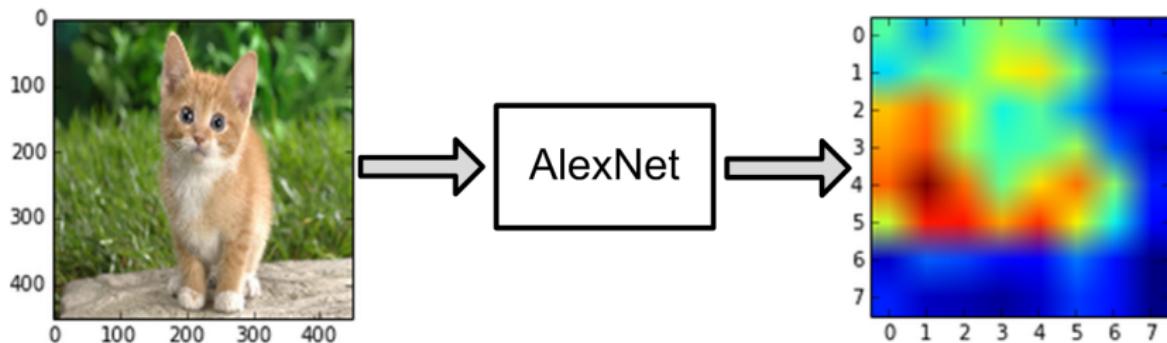
(Krizhevsky NIPS 2014)

Convert FC layers to convolution layers

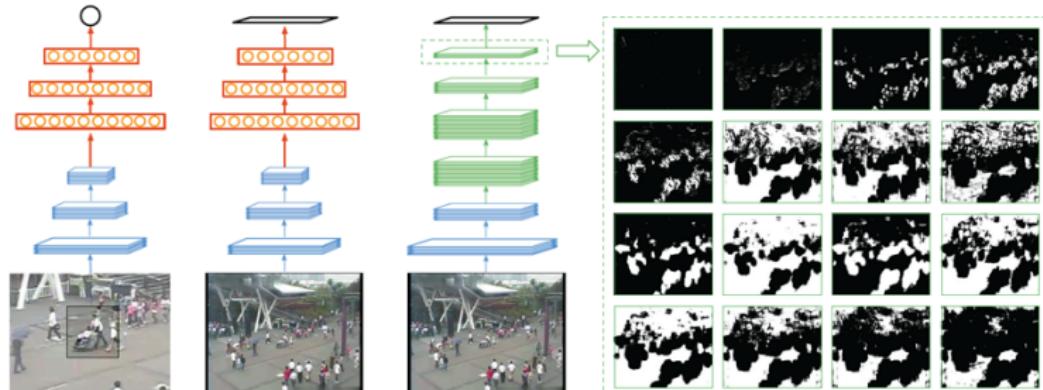
- For FC layers following a convolution or a pooling layer
 - Size of the input for the FC layer: $C \times M \times M$
 - Size of the output for the FC layer: N
 - Size of the converted convolution kernels: $C \times M \times M$
 - Number of the converted convolution kernels: N
- For FC layers following another FC layer
 - Size of the input for the FC layer: M
 - Size of the output for the FC layer: N
 - Size of the converted convolution kernels: $M \times 1 \times 1$
 - Number of the converted convolution kernels: N

Down-sampling due to greater-than-1 strides

- The output of fully convolutional network is down-sampled due to the greater-than-1 strides in convolution and pooling layers



Fully convolutional neural networks with 1×1 kernels



(a) CNN Patch-scanning

(b) CNN Regression

(c) FCNN Segmentation

(d) FCNN Feature Maps



Convolution-pooling layers



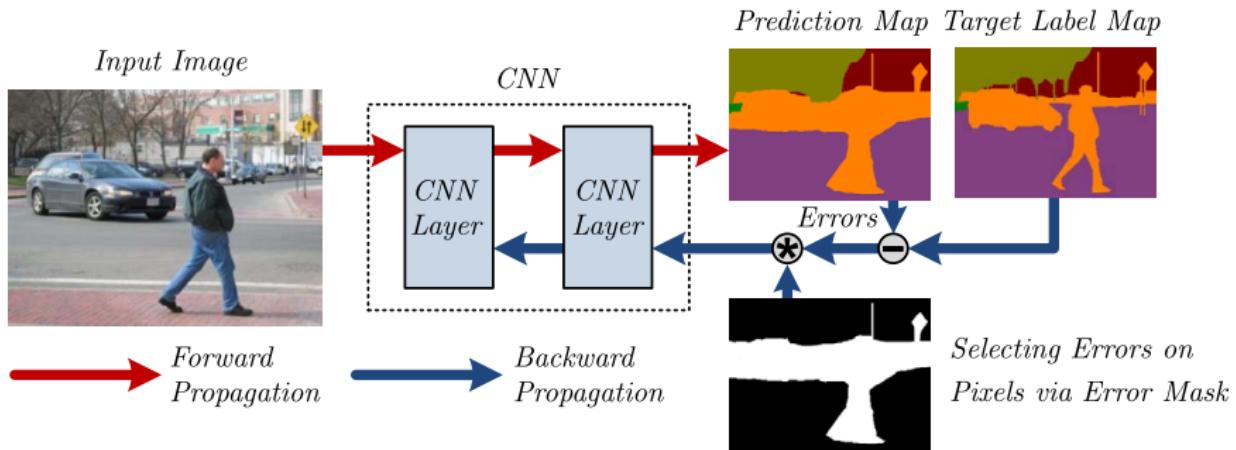
Fully connected layers



"Fusion" convolutional layers
implemented by 1×1 kernel

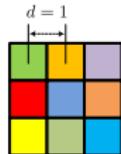
Fully convolutional networks with no down-sampling

- (Li, Zhao and Wang, arXiv:1412.4526) introduce d -regularly sparse kernels to avoid down-sampling
- The algorithm generates exactly the same results as patch-by-patch training and testing while speeds up the computation more than 1,500X

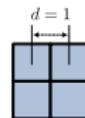


d -regularly convolution and pooling kernels

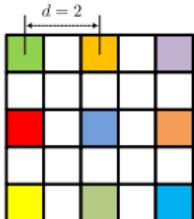
- d of original dense kernels is 1
- Insert all-zero rows and columns to create d -regularly sparse kernels



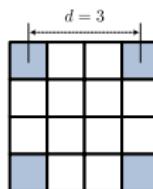
3×3 convolution kernel



2×2 pooling kernel



Converted convolution kernel



Converted pooling kernel

The algorithm

- Set stride to 1 for all convolution and pooling layers

Algorithm 1: Efficient Forward Propagation of CNN

Input: Input image I , convolution parameters W_k, b_k ,
pooling kernels P_k , strides of each layer d_k

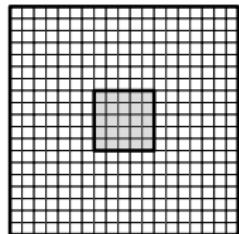
```
1 begin
2   |    $d \leftarrow 1$ 
3   |    $x_1 \leftarrow I$ 
4   |   for  $k = \{1, 2, \dots, K\}$  do
5     |     if Layer  $k ==$  convolution layer then
6       |       Convert  $W_k$  to  $W_{k,d}$ 
7       |        $y_k \leftarrow W_{k,d} *^1 x_k + b_k,$ 
8     |     else if Layer  $k ==$  pooling layer then
9       |       Convert  $P_k$  to  $P_{k,d}$ 
10      |        $y_k \leftarrow P_{k,d} \odot^1 x_k$ 
11    |     end
12    |      $d \leftarrow d \times d_k$ 
13    |      $x_{k+1} \leftarrow y_k$ 
14  end
15  return output feature map  $y_k$ 
16 end
```

A toy example

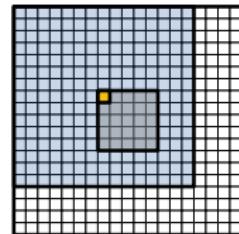
- Net architecture

Layer	input patch	conv1	pool1
size / stride	15×15	$2 \times 2 / 1$	$2 \times 2 / 2$
Layer	conv2	pool2	conv3
size / stride	$2 \times 2 / 1$	$3 \times 3 / 3$	$2 \times 2 / 1$

- A 5×5 input is properly padded to 19×19
- The 15×15 topleft patch is used for comparison

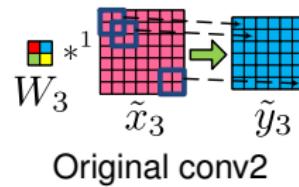
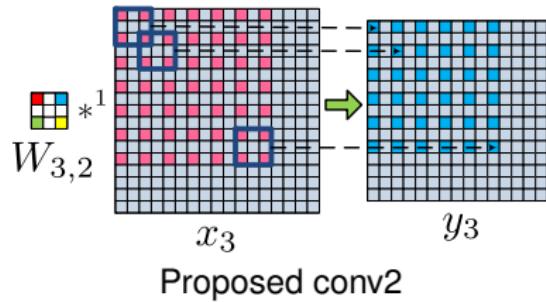
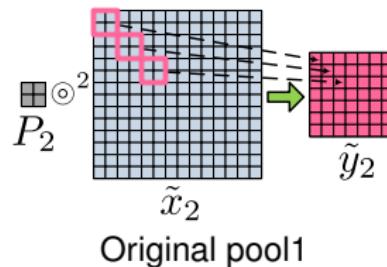
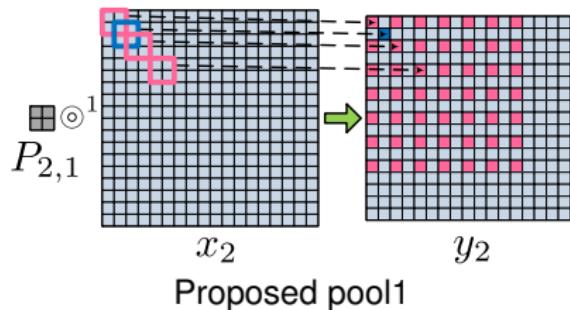


15×15 padded image

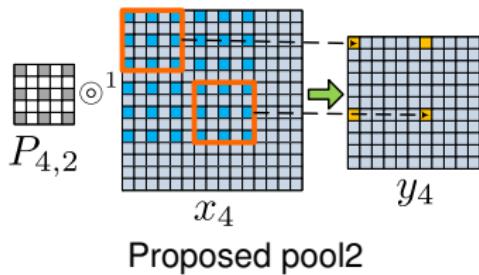


15×15 topleft patch

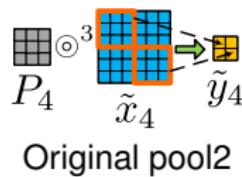
pool1 & conv2



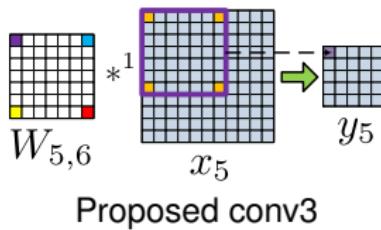
pool2 & conv3



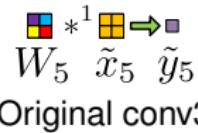
Proposed pool2



Original pool2



Proposed conv3



Original conv3

Reading materials

- Y. Bengio, I. J. GoodFellow, and A. Courville, “Convolutional Networks,” Chapter 11 in “Deep Learning”, Book in preparation for MIT Press, 2014.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. of the IEEE, 1998.
- J. Bouvrie, “Notes on Convolutional Neural Networks,” 2006.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Proc. NIPS, 2012.
- M. Ranzato, “Neural Networks,” tutorial at CVPR 2013.
- K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, “Return of the Devil in the Details: Delving Deep into Convolutional Networks,” BMVC 2014.
- P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” In Proc. Int'l Conf. Learning Representations, 2014.

Reading materials

- K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv:1409.1556, 2014.
- R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, “Deep Image: Scaling up Image Recognition,” arXiv:1501.02876, 2015.
- M. Lin, Q.. Chen, and S. Yan, “Network in network,” arXiv:1312.4400v3, 2013.
- C. Szegedy, W. Liu, Y. Jia, P. Sermanet, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” arXiv:1409.4842, 2014.
- K. Kang and X. Wang, “Fully Convolutional Neural Networks for Crowd Segmentation,” arXiv:1411.4464, 2014.
- H. Li, R. Zhao, and X. Wang, “Highly Efficient Forward and Backward Propagation of Convolutional Neural Networks for Pixelwise Classification,” arXiv:1412.4526, 2014.

Deep Belief Net

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

February 9, 2015

Outline

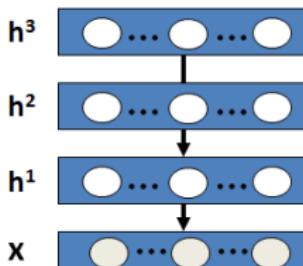
- 1 Restricted Boltzmann Machine
- 2 Deep Belief Net
- 3 Deep Boltzmann Machine

Deep belief net

- Hinton et al. 2006
- DBN is a generative model, modeling the joint distribution of observed data \mathbf{x} and hidden variables ($\{\mathbf{h}^1, \dots, \mathbf{h}^l\}$), $P(\mathbf{x}, \mathbf{h}^1, \dots, \mathbf{h}^l; \theta)$
- Hidden variables are structured into l layers and are treated as hierarchical feature representations

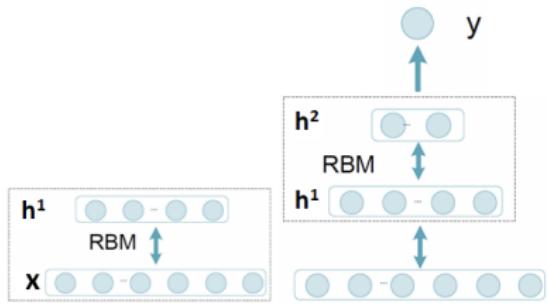
$$P(\mathbf{x}, \mathbf{h}^1, \dots, \mathbf{h}^l) = P(\mathbf{x}|\mathbf{h}^1)P(\mathbf{h}^1|\mathbf{h}^2)\dots P(\mathbf{h}^{l-1}, \mathbf{h}^l)$$
- Learn the network parameters θ by maximizing the data likelihood

$$P(\mathbf{x}; \theta) = \sum_{\mathbf{h}^1, \dots, \mathbf{h}^l} P(\mathbf{x}, \mathbf{h}^1, \dots, \mathbf{h}^l; \theta)$$
- $x_i, h_i^k \in \{0, 1\}$



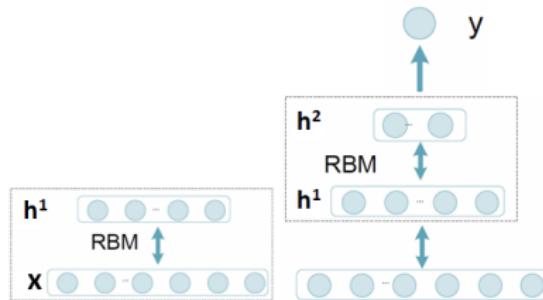
Unsupervised layerwise pre-training

- Each time only consider two layers \mathbf{h}^{k-1} and \mathbf{h}^k ($\mathbf{h}^0 = \mathbf{x}$), assuming \mathbf{h}^{k-1} is known and fixed
- The parameters between the two layers are learned by maximizing the likelihood $P(\mathbf{h}_k)$
- The joint distribution $P(\mathbf{h}^{k-1}, \mathbf{h}^k)$ of the two layers is approximated as Restricted Boltzmann Machine (RBM)
- Parameters of $P(\mathbf{h}^{k-1}, \mathbf{h}^k)$ are learned with Contrastive Divergence (CD) and fixed
- \mathbf{h}^k is sampled from $P(\mathbf{h}^k | \mathbf{h}^{k-1})$ for the training of next layer, or estimated as $\hat{\mathbf{h}}^k = \int_{\mathbf{h}^k} P(\mathbf{h}^k | \mathbf{h}^{k-1})$



Fine tuning

- The top hidden layer is used as feature to predict class label
- After pre-training, the whole network is fine-tuned with backpropagation given supervised information (e.g. class label y) provided



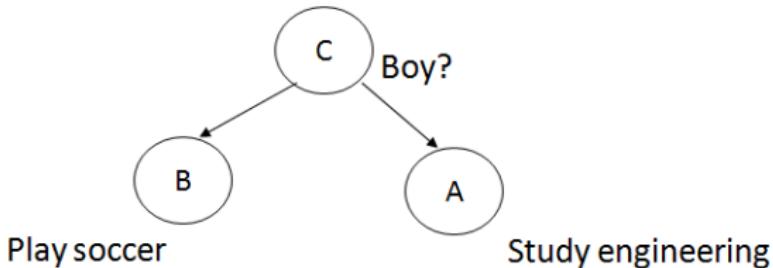
Graphical model

- Graphical models represent conditional independence between random variables
- Given C, A and B are independent:

$$P(A, B | C) = P(A|C)P(B|C)$$

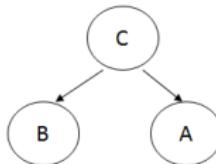
$$P(A, B, C) = P(A, B | C)P(C) = P(A|C)P(B|C)P(C)$$

- Any two nodes are conditionally independent given the values of their parents.

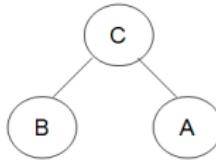


Directed and undirected graphical model

- Directed graphical model
 - $P(A, B, C) = P(A|C)P(B|C)P(C)$
 - Any two nodes are conditionally independent given the values of their parents



- Undirected graphical model
 - $P(A, B, C) = \Phi_1(B, C)\Phi_2(A, C)$
 - A and B are conditionally independent given C , if all the paths connecting A and B are blocked by C



Energy-Based Models (EBM)

- Define a probability distribution through an energy function

$$p(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z}$$

The normalization factor Z is called the partition function

$$Z = \sum_{\mathbf{x}} e^{-E(\mathbf{x})}$$

- An energy-based model can be learnt by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data

$$\ell(\theta, \mathcal{D}) = -\frac{1}{N} \sum_{\mathbf{x}^{(i)} \in \mathcal{D}} \log p(\mathbf{x}^{(i)})$$

- Use the stochastic gradient $-\frac{\partial \log p(\mathbf{x}^{(i)})}{\partial \theta}$ to update the parameters θ of the model

EBMs with hidden units

- In some cases, we do not observe the example \mathbf{x} fully, or we want to introduce some non-observed variables to increase the expressive power of the model. So we consider an observed part (\mathbf{x}) and a hidden part \mathbf{h} :

$$P(\mathbf{x}) = \sum_{\mathbf{h}} P(\mathbf{x}, \mathbf{h}) = \sum_{\mathbf{h}} \frac{e^{-E(\mathbf{x}, \mathbf{h})}}{Z}$$

- Define the free energy as

$$\mathcal{F}(\mathbf{x}) = -\log \sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})}$$

- $P(\mathbf{x})$ can be written as $P(\mathbf{x}) = \frac{e^{-\mathcal{F}(\mathbf{x})}}{Z}$ with $Z = \sum_{\mathbf{x}} e^{-\mathcal{F}(\mathbf{x})}$
- The data negative log-likelihood gradient has the form

$$-\frac{\partial \log p(\mathbf{x})}{\partial \theta} = \frac{\partial \mathcal{F}(\mathbf{x})}{\partial \theta} - \sum_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}}) \frac{\partial \mathcal{F}(\tilde{\mathbf{x}})}{\partial \theta}$$

The first term increases the probability of training data (by reducing the corresponding free energy), while the second term decreases the probability of samples generated by the model.

EBMs with hidden units

- It is usually difficult to determine this gradient analytically, as it involves the computation of $E_P[\frac{\partial \mathcal{F}(\mathbf{x})}{\partial \theta}]$, which is an expectation over all possible configurations of the input \mathbf{x} under the distribution.
- To make the computation tractable, estimate the expectation using a fixed number of samples \mathcal{N} , which are generated according to P .

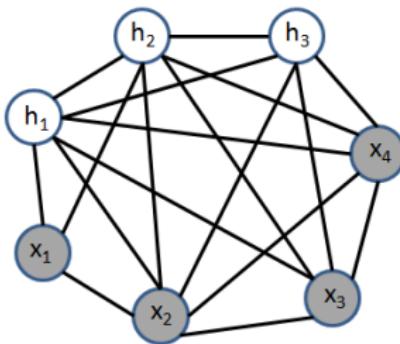
$$-\frac{\partial \log p(\mathbf{x})}{\partial \theta} = \frac{\partial \mathcal{F}(\mathbf{x})}{\partial \theta} - \frac{1}{|\mathcal{N}|} \sum_{\tilde{\mathbf{x}} \in \mathcal{N}} \frac{\partial \mathcal{F}(\tilde{\mathbf{x}})}{\partial \theta}$$

- The key question is how to generate \mathcal{N} from P .

Boltzmann Machine (BM)

$$E(\mathbf{x}, \mathbf{h}; \theta) = -(\mathbf{b}' \mathbf{x} + \mathbf{c}' \mathbf{h} + \mathbf{h}' \mathbf{Wx} + \mathbf{x}' \mathbf{Ux} + \mathbf{h}' \mathbf{Vh})$$

$$\theta = \{\mathbf{b}', \mathbf{c}', \mathbf{W}, \mathbf{U}, \mathbf{V}\}$$



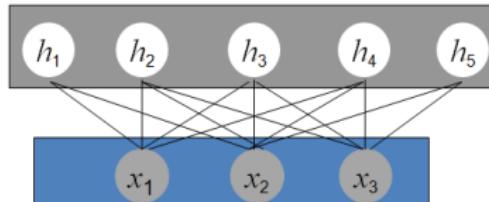
Restricted Boltzmann Machine (RBM)

- RBM does not model the interactions of variables of the same layer
- $E(\mathbf{x}, \mathbf{h})$ is the energy function. $Z = \sum_{\mathbf{x}, \mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})}$ is call partition function and serves as a normalizing factor
- \mathbf{b} , \mathbf{c} , and \mathbf{W} are the parameters to be learned

$$P(\mathbf{x}, \mathbf{h}) = \frac{e^{-E(\mathbf{x}, \mathbf{h})}}{\sum_{\mathbf{x}, \mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})}}$$

$$P(\mathbf{x}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})}}{\sum_{\mathbf{x}, \mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})}}$$

$$E(\mathbf{x}, \mathbf{h}) = -(\mathbf{b}' \mathbf{x} + \mathbf{c}' \mathbf{h} + \mathbf{h}' \mathbf{W} \mathbf{x})$$



Restricted Boltzmann Machine (RBM)

Let $\mathbf{a} = \mathbf{Wx}$

$$\begin{aligned} P(\mathbf{h}|\mathbf{x}) &\propto e^{(\mathbf{b}'\mathbf{x} + \mathbf{c}'\mathbf{h} + \mathbf{h}'\mathbf{a})} \\ &\propto e^{(\mathbf{c}'\mathbf{h} + \mathbf{h}'\mathbf{a})} \\ &= e^{\sum_i (c_i + a_i) h_i} \\ &= \prod_i e^{(c_i + a_i) h_i} \\ &= \prod_i f(h_i) \end{aligned}$$

Since \mathbf{x} is fixed,

$$P(\mathbf{h}|\mathbf{x}) \propto \prod_i f(h_i)$$

Therefore, $\{h_i\}$ are conditional independent given \mathbf{x} .

Restricted Boltzmann Machine (RBM)

- $\{h_i\}$ are conditionally independent given \mathbf{x} . $\{x_i\}$ are conditionally independent given \mathbf{h} .

$$P(\mathbf{h}|\mathbf{x}) = \prod_i P(h_i|\mathbf{x})$$

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(x_i|\mathbf{h})$$

- The conditional distributions have analytical solutions

$$P(x_j = 1|\mathbf{h}) = \sigma(b_j + \mathbf{W}'_{.j} \cdot \mathbf{h})$$

$$P(h_i = 1|\mathbf{x}) = \sigma(c_i + \mathbf{W}_{i.} \cdot \mathbf{x})$$

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Restricted Boltzmann Machine (RBM)

- Assuming there are N training samples $\mathbf{X} = [\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}]$, the parameters θ are estimated by maximizing the log-likelihood

$$L(\mathbf{X}; \theta) = \frac{1}{N} \sum_{n=1}^N \log P(\mathbf{x}^{(n)}; \theta)$$

$$\theta_{t+1} = \theta_t + \eta \frac{\partial L(\mathbf{X}; \theta)}{\partial \theta}$$

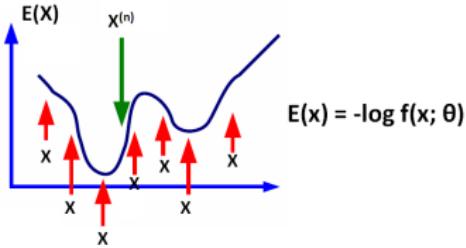
Contrastive Divergence (CD)

$$P(\mathbf{x}; \theta) = \frac{\sum_{\mathbf{h}} e^{(\mathbf{b}' \mathbf{x} + \mathbf{c}' \mathbf{h} + \mathbf{h}' \mathbf{W} \mathbf{x})}}{\sum_{\mathbf{x}, \mathbf{h}} e^{(\mathbf{b}' \mathbf{x} + \mathbf{c}' \mathbf{h} + \mathbf{h}' \mathbf{W} \mathbf{x})}} = \frac{f(\mathbf{x}; \theta)}{Z(\theta)}$$

$$\begin{aligned} \frac{\partial L(\mathbf{X}; \theta)}{\partial w_{ji}} &= \frac{1}{N} \sum_{n=1}^N \frac{\partial \log f(\mathbf{x}^{(n)}; \theta)}{\partial w_{ij}} - \frac{\partial \log Z(\theta)}{\partial w_{ij}} \\ &= \frac{1}{N} \sum_{n=1}^N \frac{\partial \log f(\mathbf{x}^{(n)}; \theta)}{\partial w_{ij}} - \frac{1}{Z(\theta)} \frac{\partial Z(\theta)}{\partial \theta} \\ &= \frac{1}{N} \sum_{n=1}^N \frac{\partial \log f(\mathbf{x}^{(n)}; \theta)}{\partial w_{ij}} - \sum_{\mathbf{x}} \frac{1}{Z(\theta)} \frac{\partial f(\mathbf{x}; \theta)}{\partial w_{ij}} \\ &= \frac{1}{N} \sum_{n=1}^N \frac{\partial \log f(\mathbf{x}^{(n)}; \theta)}{\partial w_{ij}} - \sum_{\mathbf{x}} \frac{f(\mathbf{x}; \theta)}{Z(\theta)} \frac{1}{f(\mathbf{x}; \theta)} \frac{\partial f(\mathbf{x}; \theta)}{\partial w_{ij}} \\ &= \frac{1}{N} \sum_{n=1}^N \frac{\partial \log f(\mathbf{x}^{(n)}; \theta)}{\partial w_{ij}} - \sum_{\mathbf{x}} P(\mathbf{x}; \theta) \frac{\partial \log f(\mathbf{x}; \theta)}{\partial w_{ij}} \end{aligned}$$

Contrastive Divergence (CD)

$$\begin{aligned}
 \frac{\partial L(\mathbf{X}; \theta)}{\partial w_{ij}} &= - \sum_{\mathbf{x}} P(\mathbf{x}; \theta) \frac{\partial \log f(\mathbf{x}; \theta)}{\partial w_{ij}} + \frac{1}{N} \sum_{n=1}^N \frac{\partial \log f(\mathbf{x}^{(n)}; \theta)}{\partial w_{ij}} \\
 &= - \left\langle \frac{\partial \log f(\mathbf{x}; \theta)}{\partial \theta} \right\rangle_{model} + \left\langle \frac{\partial f(\mathbf{x}; \theta)}{\partial \theta} \right\rangle_{data} \\
 &= \left\langle \frac{\partial E(\mathbf{x})}{\partial \theta} \right\rangle_{model} - \left\langle \frac{\partial E(\mathbf{x})}{\partial \theta} \right\rangle_{data} \\
 &= - \left\langle x_i h_j \right\rangle_{model} + \left\langle x_i^{(n)} h_j \right\rangle_{data} \\
 &= - \sum_{\mathbf{x}, \mathbf{h}} P(\mathbf{x}, \mathbf{h}; \theta) x_i h_j + \frac{1}{N} \sum_{n=1}^N \sum_{h_j} P(h_j | \mathbf{x}^{(n)}; \theta) x_i^{(n)} h_j
 \end{aligned}$$



Contrastive Divergence

- $\langle x_i^{(n)} h_j \rangle_{\text{data}}$ is the estimation of the average of $x_i h_j$ from the empirical distribution on the training set
- $\langle x_i h_j \rangle_{\text{model}}$ can be estimated from infinite number of samples $\{\tilde{\mathbf{x}}^{(m)}\}_{m=1}^{\infty}$ randomly drawn from $P(\mathbf{x}; \theta)$

$$\begin{aligned} \frac{\partial L(\mathbf{X}; \theta)}{\partial w_{ji}} &\approx -\frac{1}{M} \sum_{\tilde{\mathbf{x}}^{(m)} \sim P(\mathbf{x}; \theta)} \sum_{\tilde{h}_j^{(m)}} P(\tilde{h}_j^{(m)} | \tilde{\mathbf{x}}^{(m)}; \theta) \tilde{x}_i^{(m)} \tilde{h}_j^{(m)} \\ &\quad + \frac{1}{N} \sum_{n=1}^N \sum_{h_j} P(h_j | \mathbf{x}^{(n)}; \theta) x_i^{(n)} h_j \end{aligned}$$

Contrastive Divergence

- $\{\tilde{\mathbf{x}}^{(m)}\}$ can be generated from Markov Chain Monte Carlo (MCMC). Gibbs sampling is a commonly used MCMC approach and was used by Hinton et al. (1986).
- Starting from any training sample $\tilde{\mathbf{x}}^{(0)}$, a sequence of samples $\tilde{\mathbf{x}}^{(1)}, \dots, \tilde{\mathbf{x}}^{(M)}$ are generated in the following way

$$\tilde{\mathbf{h}}^{(0)} \sim P(\mathbf{h}|\mathbf{x}^{(0)})$$

$$\tilde{\mathbf{x}}^{(1)} \sim P(\mathbf{x}|\tilde{\mathbf{h}}^{(0)})$$

$$\tilde{\mathbf{h}}^{(0)} \sim P(\mathbf{h}|\tilde{\mathbf{x}}^{(1)})$$

...

$$\tilde{\mathbf{x}}^{(M)} \sim P(\mathbf{x}|\tilde{\mathbf{h}}^{(M-1)})$$

- $\tilde{\mathbf{x}}^{(1)}, \dots, \tilde{\mathbf{x}}^{(M)}$ follow the distribution of $P(\mathbf{x})$
- MCMC is slow

Contrastive Divergence

- Stochastic gradient descent: replacing the average over all the training samples with a single training sample
- Starting from the chosen training sample $\mathbf{x}^{(n)}$, only do one step MCMC sampling and use the generated sample $\tilde{x}_i^{(1)} \tilde{h}_j^{(1)}$ to approximate $\langle x_i h_j \rangle_{model}$ which is supposed to be estimated from infinite number of samples generated from MCMC

$$\frac{\partial L(\mathbf{X}; \theta)}{\partial w_{ji}} \approx -\tilde{x}_i^{(1)} P(\tilde{h}_j^{(1)} = 1 | \tilde{x}_i^{(1)}) + x_i^{(n)} P(h_j^{(n)} = 1 | \mathbf{x}^{(n)})$$

$$\frac{\partial L(\mathbf{X}; \theta)}{\partial b_i} \approx -\tilde{x}_i^{(1)} + x_i^{(n)}$$

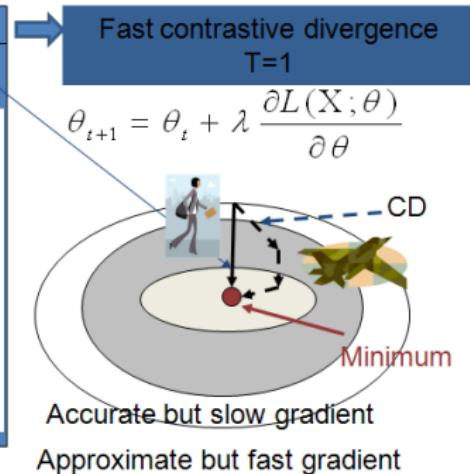
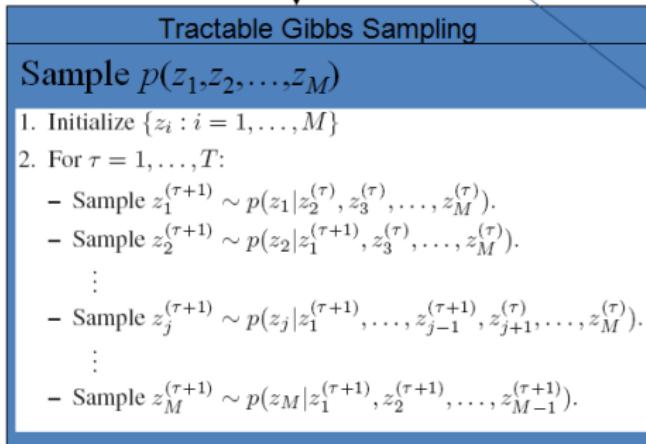
$$\frac{\partial L(\mathbf{X}; \theta)}{\partial c_j} \approx -P(\tilde{h}_j^{(1)} = 1 | \tilde{x}_i^{(1)}) + P(h_j^{(n)} = 1 | \mathbf{x}^{(n)})$$

Contrastive Divergence

$$\frac{\partial L(X; \theta)}{\partial \theta} = - \int p(x, \theta) \frac{\partial \log f(x; \theta)}{\partial \theta} dx + \frac{1}{K} \sum_{k=1}^K \frac{\partial \log f(x^{(k)}; \theta)}{\partial \theta}$$

Intractable

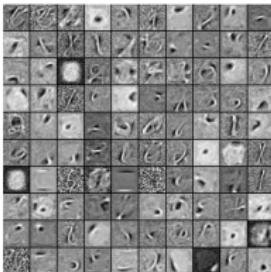
Easy to compute



Convergence of Contrastive Divergence

- The stationary points of maximum likelihood (ML) estimation is not the stationary points of CD
- CD is biased, but the bias is typically small
- CD can be used for getting close to ML solution and then ML learning can be used for fine-tuning
- It is not clear whether CD learning converges

Filters learned by RBM



Filters learned by RBM

7	9	6	3	8	8	0	8	3	8	8	9	8	8	9	8	6	9	3	3
7	6	6	3	8	8	0	8	3	8	8	9	8	8	6	8	6	9	3	3
7	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
7	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
7	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
7	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
7	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
7	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
7	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	4	3	3
9	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	4	3	3
9	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	4	3	3
9	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
9	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
9	6	6	2	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
9	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	9	3	3
9	6	6	3	8	8	0	8	3	8	8	6	8	8	6	8	6	6	3	3

Samples generated by RBM from Gibbs sampling

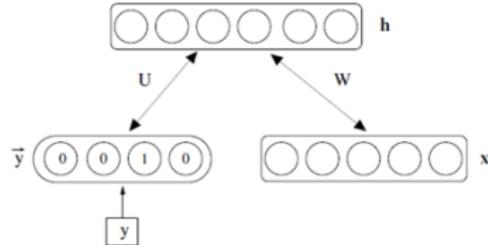
RBM for classification

- Larochelle and Bengio 2008
- \mathbf{y} : vector of class label

$$p(y, \mathbf{x}, \mathbf{h}) \propto e^{-E(y, \mathbf{x}, \mathbf{h})}$$

$$E(y, \mathbf{x}, \mathbf{h}) = -\mathbf{h}'\mathbf{W}\mathbf{x} - \mathbf{b}'\mathbf{x} - \mathbf{c}'\mathbf{h} - \mathbf{d}'\mathbf{y} - \mathbf{h}'\mathbf{U}\mathbf{y}$$

$$p(y|\mathbf{x}) = \frac{e^{d_y} \prod_j (1 + e^{c_j + U_{jy} + \sum_i W_{ji}x_i})}{\sum_{y^*} e^{d_{y^*}} \prod_j (1 + e^{c_j + U_{jy^*} + \sum_i W_{ji}x_i})}$$



RBM for classification

- Update

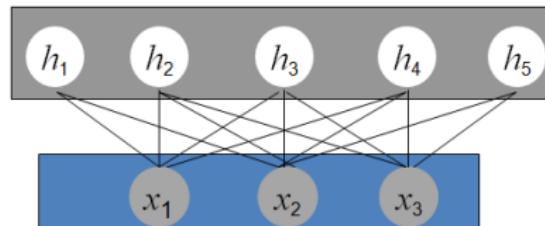
$$\begin{aligned}\frac{\partial \log p(y^{(n)} | \mathbf{x}^{(n)})}{\partial \theta} &= \sum_j \sigma(o_{y^{(n)} j}(\mathbf{x}^{(n)})) \frac{\partial o_{y^{(n)} j}(\mathbf{x}^{(n)})}{\partial \theta} \\ &\quad - \sum_{j, y^*} \sigma(o_{y^* j}(\mathbf{x}^{(n)})) p(y^* | \mathbf{x}^{(n)}) \frac{\partial o_{y^* j}(\mathbf{x}^{(n)})}{\partial \theta}\end{aligned}$$

$$o_{yj}(\mathbf{x}) = c_j + \sum_i w_{ji} x_i + U_{jy}$$

RBM leads to distributed representation

- Each hidden unit can be treated as an attribute and creates a 2-region partition of the input space. The binary setting of the N hidden units identifies one region in input space among all the 2^N regions associated with configurations of the hidden units
- If the distribution $P(x)$ represented by RMB is transformed to a mixture model, it is a sum over an exponential number of configurations

$$P(\mathbf{x}) = \sum_{\mathbf{h}} P(\mathbf{x}|\mathbf{h})P(\mathbf{h})$$



RBM leads to distributed representation

- RBM can be represented as a product of experts

$$P(\mathbf{x}) \propto e^{\mathbf{b}'\mathbf{x} + \sum_j \log \sum_{h_j} e^{h_j \mathbf{w}_j \cdot \mathbf{x}}}$$
$$\propto \prod_j e^{f_j(\mathbf{x})}$$

where $f_j = \log \sum_{h_j} e^{h_j \mathbf{w}_j \cdot \mathbf{x}}$

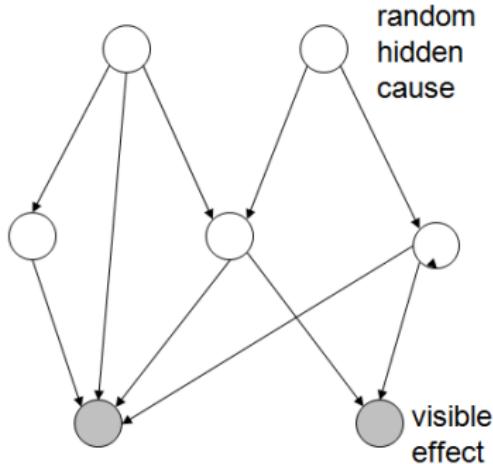
- $f_j(\mathbf{x})$ is an attribute indicator
- Hinton (1999) explains the advantages of a product of experts by opposition to a mixture of experts
 - In a mixture model, the constraint associated with an expert is an indication of belonging to a region which excludes the other regions
 - In a product of experts, the set of $f_j(\mathbf{x})$ form a distributed representation: partition the space according to all the possible configurations (where each expert can have its constraint violated or not)

Product of expert models vs mixture of expert models

- A mixture distribution can have high probability for event x when only a **single** expert assigns high probability to that event; while a product can only have high probability for an event x when **no expert** assigns an especially low probability to that event
- A single expert in a mixture has the power to pass a bill while a single expert in a product has the power to veto it
- Each component in a product represents a soft **constraint**. For an event to be likely under a product model, **all constraints** must be (approximately) satisfied
- Each expert in a mixture represents a soft **template or prototype**. An event is likely under a mixture model if it (approximately) matches with **any single template**

Belief nets

- A belief net is a directed acyclic graph composed of random variables
- Also called Bayesian network



Deep belief nets

- Belief net that is deep
- A generative model

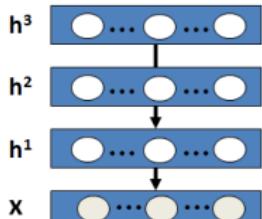
$$P(\mathbf{x}, \mathbf{h}_1, \dots, \mathbf{h}_I) = P(\mathbf{x}|\mathbf{h}_1)P(\mathbf{h}_1|\mathbf{h}_2)\dots P(\mathbf{h}_{I-2}|\mathbf{h}_{I-1})P(\mathbf{h}_{I-1}, \mathbf{h}_I)$$

- $P(\mathbf{h}^{k-1}|\mathbf{h}^k)$ ($\mathbf{x} = \mathbf{h}^0$) is conditional distribution for the visible units conditional on the hidden units of the RBM at level k

$$P(\mathbf{h}^{k-1}|\mathbf{h}^k) = \prod_i P(h_i^{k-1}|\mathbf{h}^k)$$

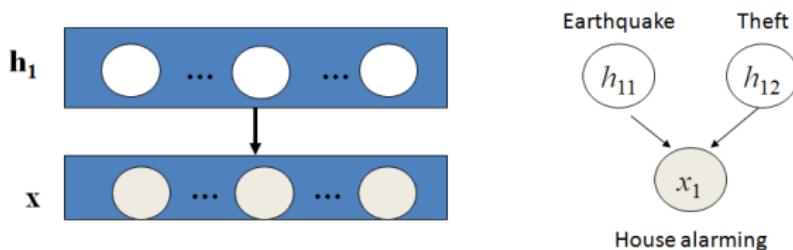
$$P(h_j^{k-1} = 1|\mathbf{h}^k) = \sigma(b_j^k + \mathbf{W}_{\cdot j}^{k'} \mathbf{h})$$

- $P(\mathbf{h}^{l-1}, \mathbf{h}^l)$ is model as RBM



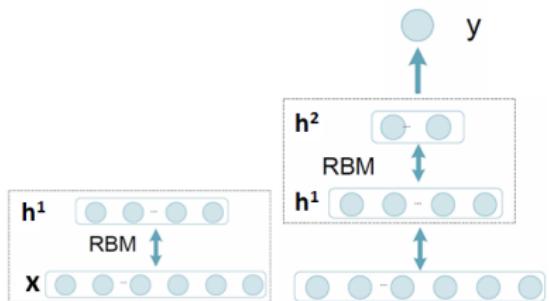
The inference of DBN is problematic

- Explaining away: $P(h_{11}, h_{12}|x_1) \neq P(h_{11}|x_1)P(h_{12}|x_1)$
- Different from RBM, DNB is a directed graphical model. Given \mathbf{x} , h_1, \dots, h_n are not independent any more. Only when $P(\mathbf{h})$ adopts certain prior (called complementary prior), DBN is equivalent to RBM, $P(\mathbf{x}, \mathbf{h}) = P(\mathbf{x}|\mathbf{h})P(\mathbf{h})$. The details of the complementary prior can be found in Hinton et al. 2006.
- Therefore, only feedforward approximate inference is used for DBN: no bottom-up and top-down



Pre-training

- Each time only consider two layers \mathbf{h}^{k-1} and \mathbf{h}^k ($\mathbf{h}^0 = \mathbf{x}$), assuming \mathbf{h}^{k-1} is known and fixed
- The parameters between the two layers are learned by maximizing the likelihood $P(\mathbf{h}_k | \mathbf{h}_{k-1})$
- The joint distribution $P(\mathbf{h}^{k-1}, \mathbf{h}^k)$ of the two layers is approximated as Restricted Boltzmann Machine (RBM)
- Parameters of $P(\mathbf{h}^{k-1}, \mathbf{h}^k)$ are learned with Contrastive Divergence (CD) and fixed
- \mathbf{h}^k is sampled from $P(\mathbf{h}^k | \mathbf{h}^{k-1})$ for the training of next layer, or estimated as $\hat{\mathbf{h}}^k = \int_{\mathbf{h}^k} P(\mathbf{h}^k | \mathbf{h}^{k-1})$
- **Because of the problem on inference, no joint optimization over all layers**



Fine tuning deep belief net

- Convert DBN to a normal multilayer neural network
- The outputs of hidden units are calculated from the inputs of the lower layer in a deterministic way

$$h_j^k = P(h_j^k = 1 | \mathbf{h}^{k-1}) = \sigma(c_j + \mathbf{W}_{j \cdot} \cdot \mathbf{h}^{k-1})$$

$$\mathbf{h}^0 = \mathbf{x}$$

- Fine tune the network with backpropagation

Deep Boltzmann Machine

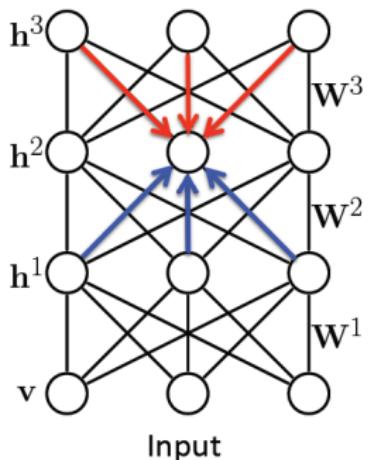
- Unsupervised feature learning
- The following slides are borrowed from Salakhutdinov's tutorial at CVPR 2012

Model

$$P_{\theta}(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3} \exp \left[\mathbf{v}^\top W^1 \mathbf{h}^1 + \mathbf{h}^1^\top W^2 \mathbf{h}^2 + \mathbf{h}^2^\top W^3 \mathbf{h}^3 \right]$$

Deep Boltzmann Machine

$\theta = \{W^1, W^2, W^3\}$ model parameters



- Dependencies between hidden variables.
- All connections are undirected.
- Bottom-up and Top-down:

$$P(h_j^2 = 1 | \mathbf{h}^1, \mathbf{h}^3) = \sigma \left(\sum_k W_{kj}^3 h_k^3 + \sum_m W_{mj}^2 h_m^1 \right)$$

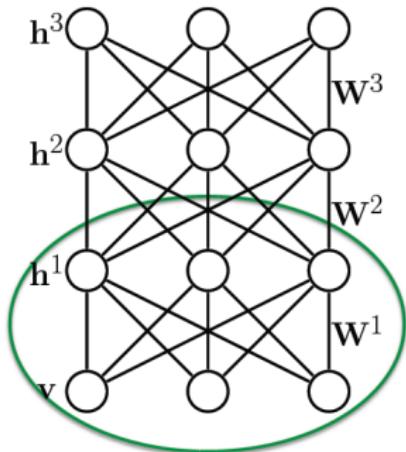
Top-down

Bottom-up

Learning

$$P_{\theta}(\mathbf{v}) = \frac{P^*(\mathbf{v})}{Z(\theta)} = \frac{1}{Z(\theta)} \sum_{\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3} \exp \left[\mathbf{v}^\top W^1 \mathbf{h}^1 + \mathbf{h}^1{}^\top W^2 \mathbf{h}^2 + \mathbf{h}^2{}^\top W^3 \mathbf{h}^3 \right]$$

Deep Boltzmann Machine $\theta = \{W^1, W^2, W^3\}$ model parameters



- Dependencies between hidden variables.

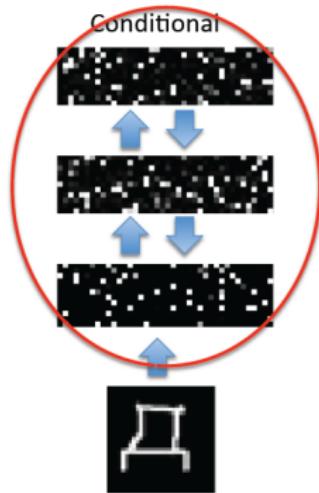
Maximum likelihood learning:

$$\frac{\partial \log P_{\theta}(\mathbf{v})}{\partial W^1} = E_{P_{data}}[\mathbf{v}\mathbf{h}^{1\top}] - E_{P_{\theta}}[\mathbf{v}\mathbf{h}^{1\top}]$$

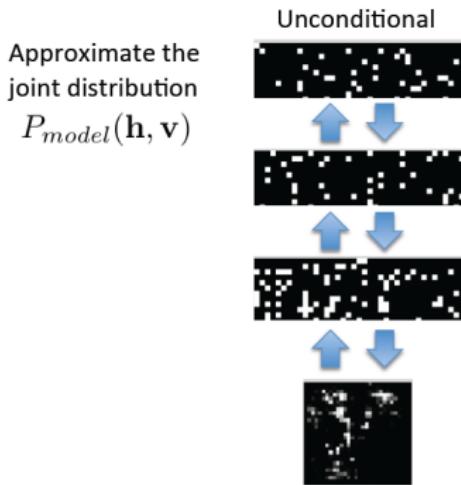
Problem: Both expectations are intractable!

Learning

Posterior Inference

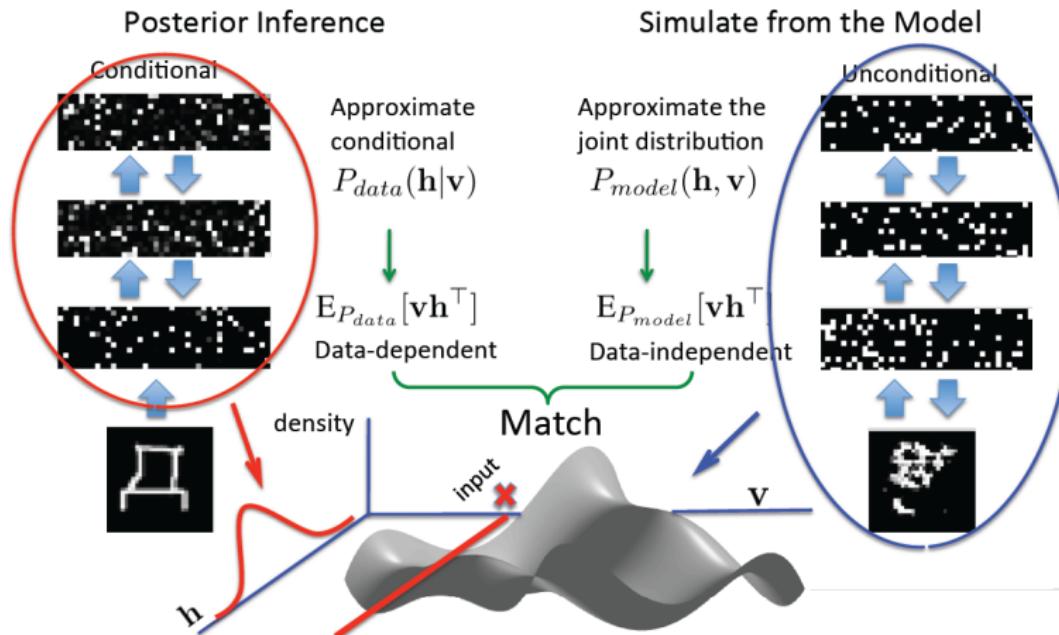


Simulate from the Model

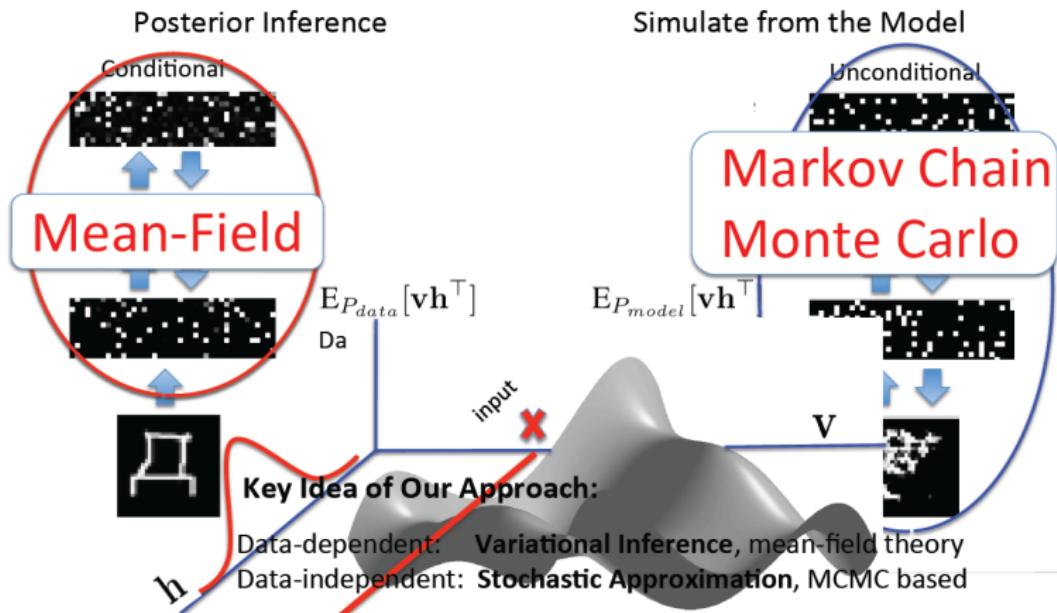


(Salakhutdinov, 2008; NIPS 2009)

Learning

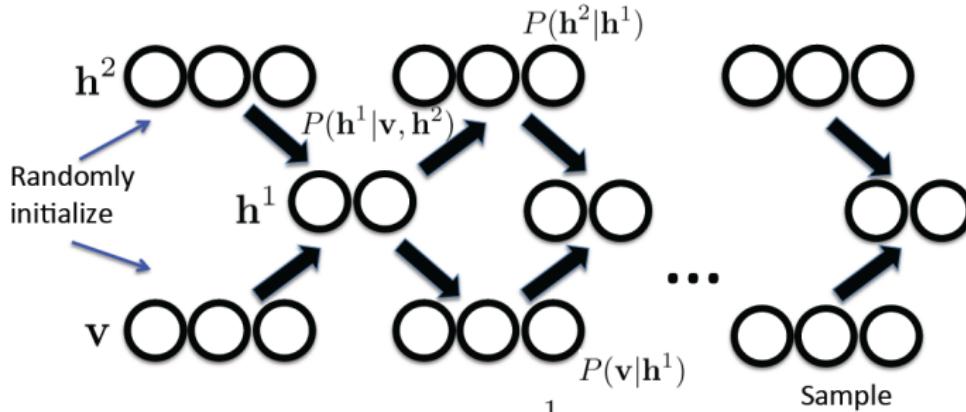


Learning



Sampling from DBMs

Sampling from two-hidden layer DBM: by running Markov chain:



$$P(h_m^1 = 1 | v, h^2) = \frac{1}{1 + \exp(-\sum_i W_{im}^1 v_i - \sum_j W_{mj}^2 h_j^2)}$$

$$P(h_j^2 = 1 | h^1) = \frac{1}{1 + \exp(-\sum_m W_{mj}^2 h_m^1)}$$

$$P(v_i = 1 | h^1) = \frac{1}{1 + \exp(-\sum_m W_{im}^1 h_m^1)}$$

Reading materials

- G. E. Hinton, S. Osindero, and Y. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *Neural Computation*, Vol. 18, pp. 1527-1544, 2006.
- H. Larochelle and Y. Bengio, “Classification using Discriminative Restricted Boltzmann Machines”, *ICML* 2008.
- G. E. Hinton, “Products of Experts,” *Proc. Int’l Conf. Artificial Neural Networks*, 1999.
- R. Salakhutdinov and G. Hinton, “Deep Boltzmann Machines,” *AISTATS* 2009.

Autoencoder

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

February 14, 2015

Outline

- 1 Autoencoder
- 2 Regularized autoencoders
- 3 Multimodal autoencoders

Autoencoder

- An autoencoder takes an input $\mathbf{x} \in [0, 1]^d$ and first maps it (with an encoder) to a hidden representation $\mathbf{y} \in [0, 1]^{d'}$ through a deterministic mapping

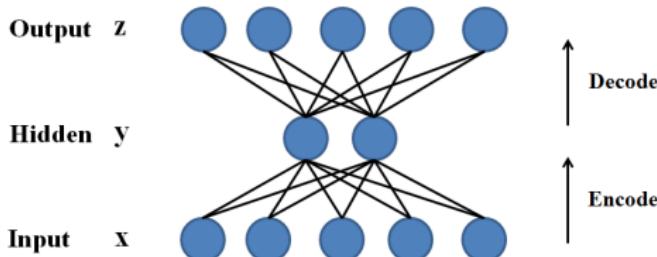
$$\mathbf{y} = s(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where s is a non-linear activation function (such as sigmoid).

- \mathbf{y} is mapped back (with a decoder) into a reconstruction \mathbf{z} of the same shape as \mathbf{x} ,

$$\mathbf{z} = s(\mathbf{W}'\mathbf{y} + \mathbf{b}')$$

\mathbf{z} is seen as a prediction of \mathbf{x} .



Autoencoder

- Encoder

$$\mathbf{y} = f_{\theta}(\mathbf{x})$$

- Decoder

$$\mathbf{z} = g_{\theta}(\mathbf{y})$$

$$\theta = \{\mathbf{W}, \mathbf{W}', \mathbf{b}, \mathbf{b}'\}$$

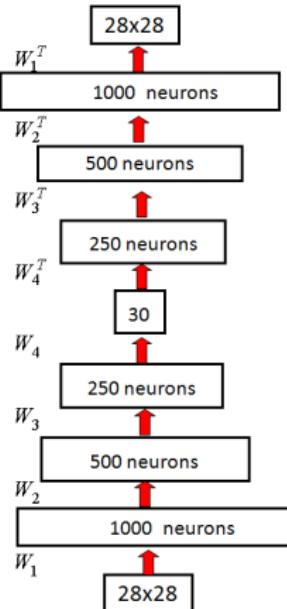
- It is important to add regularization in the training criterion or the parametrization to prevent the auto-encoder from learning the identity function, which would lead zero reconstruction error everywhere
- A particular form of regularization consists in constraining the code to have a low dimension, and this is what the classical auto-encoder or PCA do.

Autoencoder

- Optionally, the weight matrix \mathbf{W}' of the reverse mapping may be constrained to be the transpose of the forward mapping: $\mathbf{W}' = \mathbf{W}^T$, referred to as **tied weights**
- The objective function measures the reconstruction error
 - Squared error: $J(\mathbf{x}, \mathbf{z}) = ||\mathbf{x} - \mathbf{z}||^2$
 - Cross-entropy: $J(\mathbf{x}, \mathbf{z}) = -\sum_{i=1}^d [x_i \log z_i + (1 - x_i) \log(1 - z_i)]$
- \mathbf{y} is expected a distributed representation that captures the main factors of variation in data.
- If there is one linear hidden layer and the mean squared error criterion is used to train the network, the k hidden units learn to project the input in the span of the first k principal components of data.
- Autoencoder gives low reconstruction error on test examples from the same distribution as the training examples, but generally high reconstruction error on samples randomly chosen from the input space
- Autoencoder is a multi-layer neural network. The only difference is that the size of its output layer is the same as the input layer and the objective function

Deep autoencoder

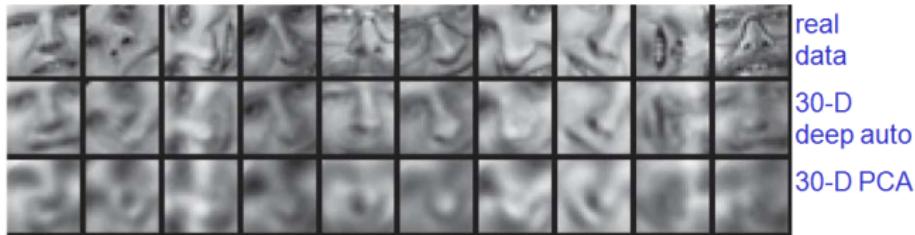
- Stack multiple encoders (and their corresponding decoders)



Deep autoencoder

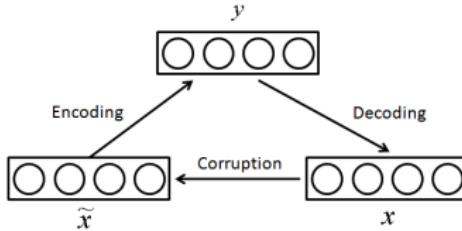
- Very difficult to optimize deep autoencoders using backpropagation
- Pre-training + fine-tuning
 - First train a stack of RBMs
 - Then “unroll” them
 - Then fine-tune with backpropagation

Comparison of methods of compressing images



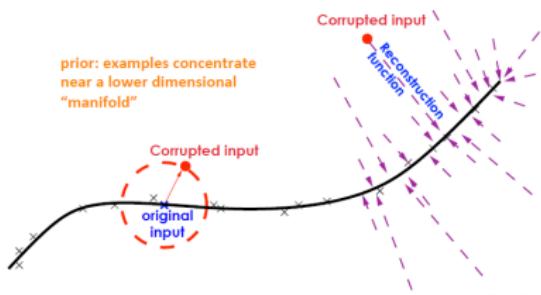
Denoising autoencoder

- In order to force the hidden layer to discover more robust features and prevent it from simply learning the identity function, train the autoencoder to reconstruct the input from a corrupted version of it
 - Encode the input (preserve the information about the input)
 - Undo the effect of a corruption process stochastically applied to the input of the auto-encoder
- To convert the autoencoder to a denoising autoencoder, all we need to do is to add a stochastic corruption step operating on the input
 - Randomly sets some of the inputs (as many as half of them) to zero. Hence the denoising auto-encoder is trying to predict the corrupted (i.e. missing) values from the uncorrupted (i.e., non-missing) values, for randomly selected subsets of missing patterns.
 - The input can be corrupted in other ways



Denoising autoencoder

- The learner must capture the structure of the input distribution in order to optimally undo the effect of the corruption process, with the reconstruction essentially being a nearby but higher density point than the corrupted input
- The denoising autoencoder is learning a reconstruction function that corresponds to a vector field pointing towards high-density regions (the manifold where examples concentrate)
- Denoising autoencoder basically learns in $r(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}$ a vector pointing in the direction $\frac{\partial \log P(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}}$



Predictive Sparse Decomposition

- Sparse coding

- Solving the encoder f_θ is non-trivial because of L_1 minimization and entails an iterative optimization

$$\mathbf{y}^* = f_\theta(\mathbf{x}) = \arg \min_{\mathbf{y}} \|\mathbf{x} - \mathbf{W}\mathbf{y}\|_2^2 + \lambda \|\mathbf{y}\|_1$$

$$J_{SC} = \sum_n \|\mathbf{x}^{(n)} - \mathbf{W}\mathbf{y}^{*(n)}\|_2^2$$

- Predictive sparse decomposition

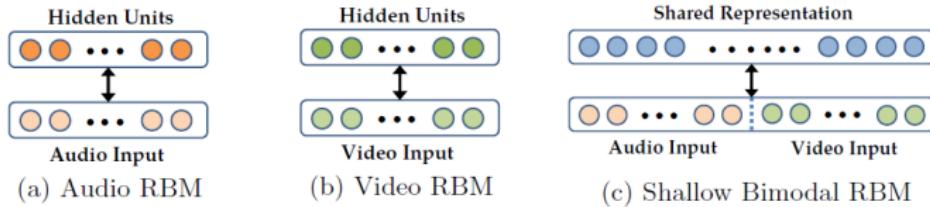
- Approximation to sparse coding
- Add sparse penalty to auto-encoder
- Replace the costly and highly non-linear encoding step by a fast non-iterative approximation
- The training criterion is simultaneously optimized with respect to the hidden codes (representation) $\mathbf{y}^{(n)}$ and with respect to the parameters θ

$$J_{PSD} = \sum_n \lambda \|\mathbf{y}^{(n)}\|_1 + \|\mathbf{x}^{(n)} - \mathbf{W}\mathbf{y}^{(n)}\|_2^2 + \|\mathbf{y}^{(n)} - f_\theta(\mathbf{x}^{(n)})\|_2^2$$

$$f_\theta(\mathbf{x}^{(n)}) = \sigma(\mathbf{b} + \mathbf{W}^T \mathbf{x}^{(n)})$$

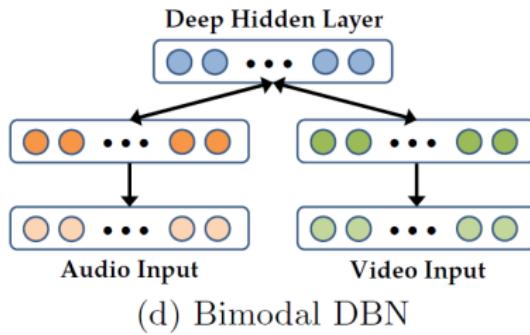
Multimodal deep learning

- Ngiam et al. ICML'11 (audio-visual speech recognition)
- Multimodal fusion: data from all modalities is available at all phases
- Cross modality learning: data from multiple modalities is available only during feature learning; during supervised training and testing, only data from a single modality is provided. The aim is to learn better single modality representations given unlabeled data from multiple modalities.
- Matching across different modalities
- A direct approach is to train a RBM over the concatenated audio and video data. Limited as a shallow model, it is hard for a RBM to learn the highly nonlinear correlations and form multimodal representations
- It was found that learning a shallow bimodal RBM results in hidden units that have strong connections to variables from individual modality but few units that connect across the modalities.



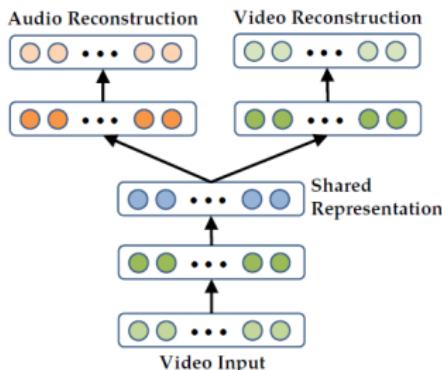
Multimodal deep learning

- Bimodal DBN: greedily layerwise training a RBM over the pre-trained layers for each modality; by representing the data through learned multilayer representations, it can be easier for the model to learn higher-order correlations across modalities. In (d), the first layer representations correspond to phonemes and visemes and the second layer models the relationships between them.
- Problems
 - It is possible for the model to find representations such that some hidden units are tuned only for audio while others are tuned only for video
 - It is not applicable in a cross modality learning setting where only one modality is present during supervised training and testing.

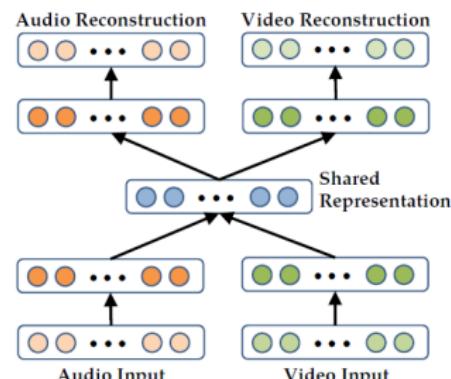


Multimodal deep auto-encoder

- In (a), the deep auto-encoder is trained to reconstruct both modalities when given only video data and thus discovers correlations across the modalities.
- Initialize the deep autoencoder with the bimodal DBN weights and discard any weights that are no longer present. The middle layer can be used as the new feature representation.
- Model (a) is used when only a single modality is present at supervised training and testing



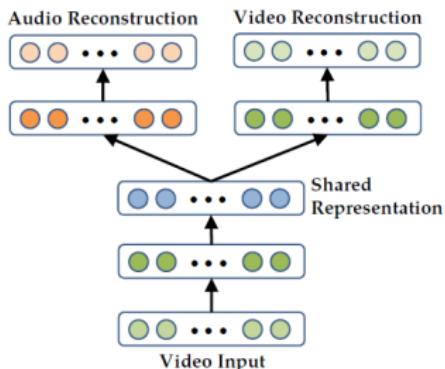
(a) Video-Only Deep Autoencoder



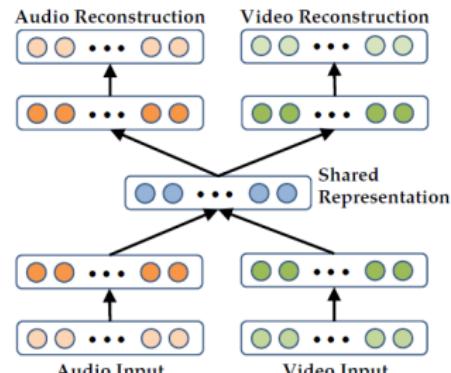
(b) Bimodal Deep Autoencoder

Multimodal deep auto-encoder

- Inspired by denoising auto-encoder, train model (b) with an augmented dataset
 - One-third of the training data has only video for input (setting zero values for the audio data)
 - Another one-third of the data has only audio
 - The last one-third of the data has both audio and video



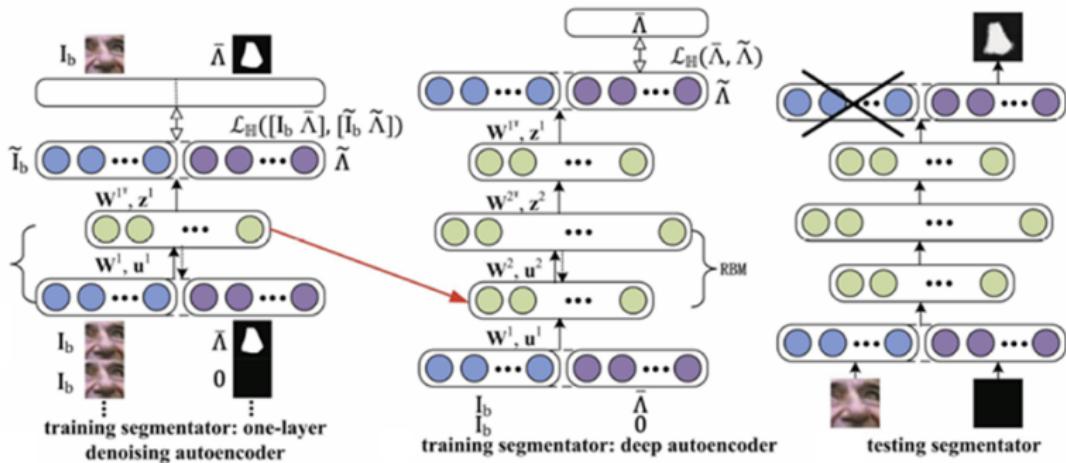
(a) Video-Only Deep Autoencoder



(b) Bimodal Deep Autoencoder

Multimodal deep auto-encoder

- For image segmentation: Luo et al. CVPR'12



Reading materials

- G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” *Science*, Vol. 313, pp. 504-507, July 2006.
- K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “Fast Inference in Sparse Coding Algorithms with Applications to Object Recognition,” CBLL-TR-2008-12-01, NYU, 2008.
- J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, “Multimodal Deep Learning,” *ICML* 2011.
- P. Luo, X. Wang, and X. Tang, “Hierarchical Face Parsing via Deep Learning,” *CVPR* 2012.

Recurrent Neural Network

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

March 7, 2015

Outline

1 HMM

- Markov Models
- Hidden Markov Model

2 Recurrent neural networks

- Recurrent neural networks
- BP on RNN
- Variants of RNN

3 Long Short-Term Memory recurrent networks

- Challenge of long-term dependency
- Combine short and long paths
- Long short-term memory net

4 Applications

Sequential data

- Sequence of words in an English sentence
- Acoustic features at successive time frames in speech recognition
- Successive frames in video classification
- Rainfall measurements on successive days in Hong Kong
- Daily values of current exchange rate
- Nucleotide base pairs in a strand of DNA
- **Instead of making independent predictions on samples, assume the dependency among samples and make a sequence of decisions for sequential samples**

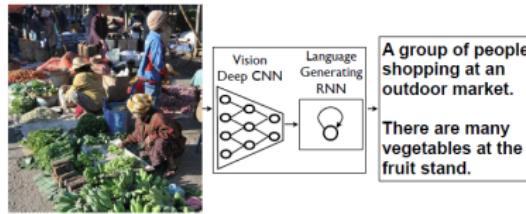
Modeling sequential data

- Sample data sequences from a certain distribution

$$P(\mathbf{x}_1, \dots, \mathbf{x}_T)$$

- Generate natural sentences to describe an image

$$P(\mathbf{y}_1, \dots, \mathbf{y}_T | I)$$



- Activity recognition from a video sequence

$$P(\mathbf{y} | \mathbf{x}_1, \dots, \mathbf{x}_T)$$

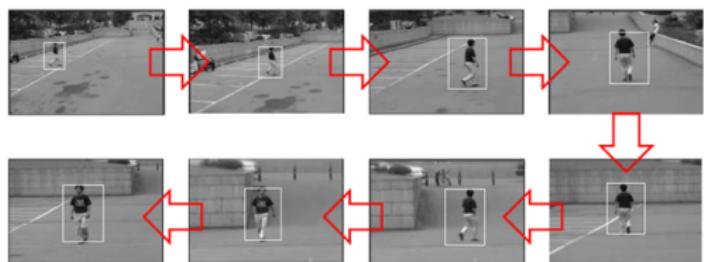
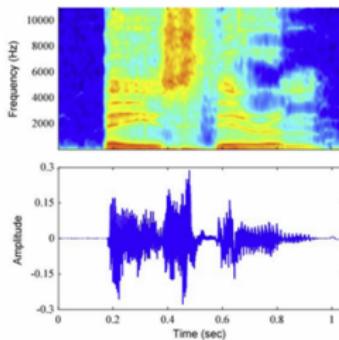
Modeling sequential data

- Speech recognition

$$P(\mathbf{y}_1, \dots, \mathbf{y}_T | \mathbf{x}_1, \dots, \mathbf{x}_T)$$

- Object tracking

$$P(\mathbf{y}_1, \dots, \mathbf{y}_T | \mathbf{x}_1, \dots, \mathbf{x}_T)$$



| b | ey | z | th | ib | er | em |
| Bayes' | Theorem |

Modeling sequential data

- Generate natural sentences to describe a video

$$P(\mathbf{y}_1, \dots, \mathbf{y}_{T'} | \mathbf{x}_1, \dots, \mathbf{x}_T)$$

- Language translation

$$P(\mathbf{y}_1, \dots, \mathbf{y}_{T'} | \mathbf{x}_1, \dots, \mathbf{x}_T)$$

Ulrich UNK , membre du conseil d' administration du constructeur automobile Audi , affirme qu' il s' agit d' une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d' administration afin qu' ils ne soient pas utilisés comme appareils d' écoute à distance .

Ulrich Hackenberg , membre du conseil d' administration du constructeur automobile Audi , déclare que la collecte des téléphones portables avant les réunions du conseil , afin qu' ils ne puissent pas être utilisés comme appareils d' écoute à distance , est une pratique courante depuis des années .

Modeling sequential data

- Use the chain rule to express the joint distribution for a sequence of observations

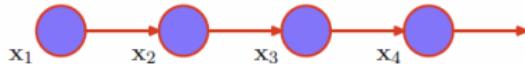
$$p(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_1, \dots, \mathbf{x}_{t-1})$$

- Impractical to consider general dependence of future dependence on all previous observations $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \dots, \mathbf{x}_0)$
 - Complexity would grow without limit as the number of observations increases
- It is expected that recent observations are more informative than more historical observations in predicting future values

Markov models

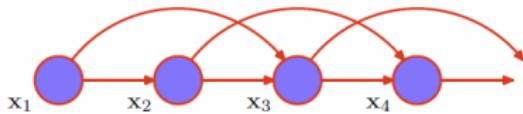
- Markov models assume dependence on most recent observations
- First-order Markov model

$$p(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1})$$



- Second-order Markov model

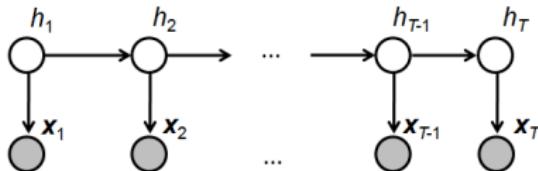
$$p(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2})$$



Hidden Markov Model (HMM)

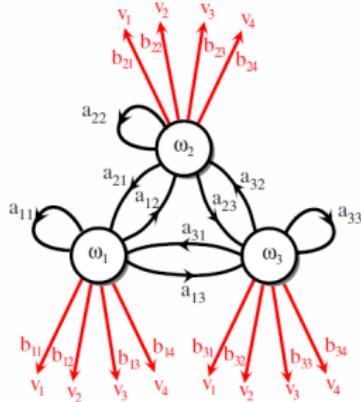
- A classical way to model sequential data
- Sequence pairs h_1, h_2, \dots, h_T (hidden variables) and $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$ (observations) are generated by the following process
 - Pick h_1 at random from the distribution $P(h_1)$. Pick \mathbf{x}_1 from the distribution $p(\mathbf{x}_1|h_1)$
 - For $t = 2$ to T
 - Choose h_t at random from the distribution $p(h_t|h_{t-1})$
 - Choose \mathbf{x}_t at random from the distribution $p(\mathbf{x}_t|h_t)$
- The joint distribution is

$$p(\mathbf{x}_1, \dots, \mathbf{x}_T, h_1, \dots, h_T, \theta) = P(h_1) \prod_{t=2}^T P(h_t|h_{t-1}) \prod_{t=1}^T p(\mathbf{x}_t|h_t)$$



Parameters of HMM

- Initial state parameters $P(h_1 = \omega_i) = \pi_i, \sum_i \pi_i = 1$
- Transition probabilities $P(h_t = \omega_j | h_{t-1} = \omega_i) = a_{ij}, \sum_j a_{ij} = 1$. $\mathbf{A} = [a_{ij}]$ is transition matrix
- Emission probabilities $p(\mathbf{x}_t | h_t)$
 - If \mathbf{x}_t is a single discrete variable, $\mathbf{x}_t = x_t \in \{v_1, \dots, v_m\}$, $P(x_t = v_k | z_t = \omega_j) = b_{jk}, \sum_k b_{jk} = 1$



Parameters of HMM

- **Evaluation problem:** determine the probability that a particular data sequence $\mathbf{x}_1, \dots, \mathbf{x}_T$ was generated by that model,

$$P(\mathbf{x}_1, \dots, \mathbf{x}_T)$$

- **Decoding problem:** given a set of observations $\mathbf{x}_1, \dots, \mathbf{x}_T$, determine the most likely sequence of hidden states h_1, \dots, h_T that lead to the observations

$$(h_1^*, \dots, h_T^*) = \operatorname{argmax}_{(h_1, \dots, h_T)} P(h_1, \dots, h_T | \mathbf{x}_1, \dots, \mathbf{x}_T)$$

- **Learning problem:** given a set of training observations of visible variables, **without knowing the hidden variables**, determine the parameters of HMM.

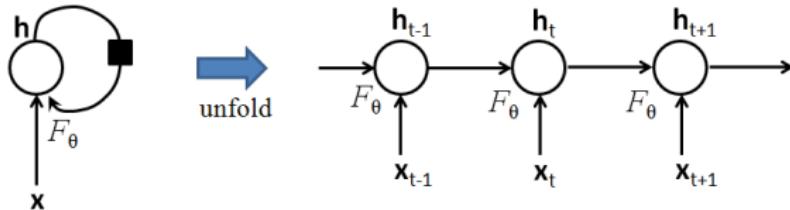
$$\theta^* = P(\mathbf{x}_1, \dots, \mathbf{x}_T; \theta)$$

Recurrent neural networks (RNN)

- While HMM is a generative model RNN is a discriminative model
- Model a dynamic system driven by an external signal \mathbf{x}_t

$$\mathbf{h}_t = F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

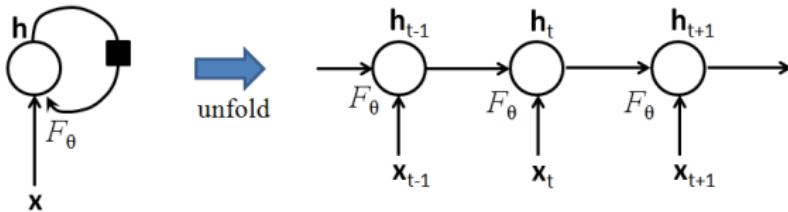
- \mathbf{h}_t contains information about the whole past sequence. The equation above implicitly defines a function which maps the whole past sequence $(\mathbf{x}_t, \dots, \mathbf{x}_1)$ to the current state $\mathbf{h}_t = G_t(\mathbf{x}_t, \dots, \mathbf{x}_1)$



Left: physical implementation of RNN, seen as a circuit. The black square indicates a delay of 1 time step. Right: the same seen as an unfolded flow graph, where each node is now associated with one particular time instance.

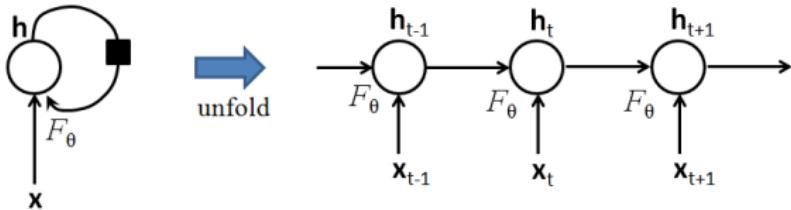
Recurrent neural networks (RNN)

- The summary is lossy, since it maps an arbitrary length sequence $(\mathbf{x}_t, \dots, \mathbf{x}_1)$ to a fixed length vector \mathbf{h}_t . Depending on the training criterion, \mathbf{h}_t keeps some important aspects of the past sequence.
- Sharing parameters: the same weights are used for different instances of the artificial neurons at different time steps
- Share a similar idea with CNN: replacing a fully connected network with local connections with parameter sharing
- It allows to apply the network to input sequences of different lengths and predict sequences of different lengths



Recurrent neural networks (RNN)

- Sharing parameters for any sequence length allows more better generalization properties. If we have to define a different function G_t for each possible sequence length, each with its own parameters, we would not get any generalization to sequences of a size not seen in the training set. One would need to see a lot more training examples, because a separate model would have to be trained for each sequence length.



A vanilla RNN to predict sequences from input

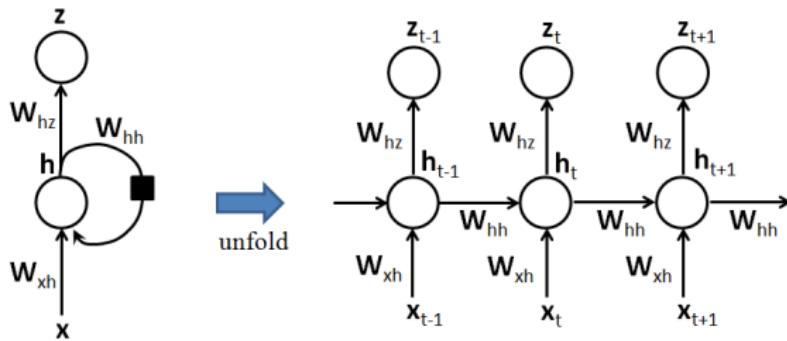
$$P(y_1, \dots, y_T | \mathbf{x}_1, \dots, \mathbf{x}_T)$$

- Forward propagation equations, assuming that hyperbolic tangent non-linearities are used in the hidden units and softmax is used in output for classification problems

$$\mathbf{h}_t = \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{z}_t = \text{softmax}(\mathbf{W}_{hz}\mathbf{h}_t + \mathbf{b}_z)$$

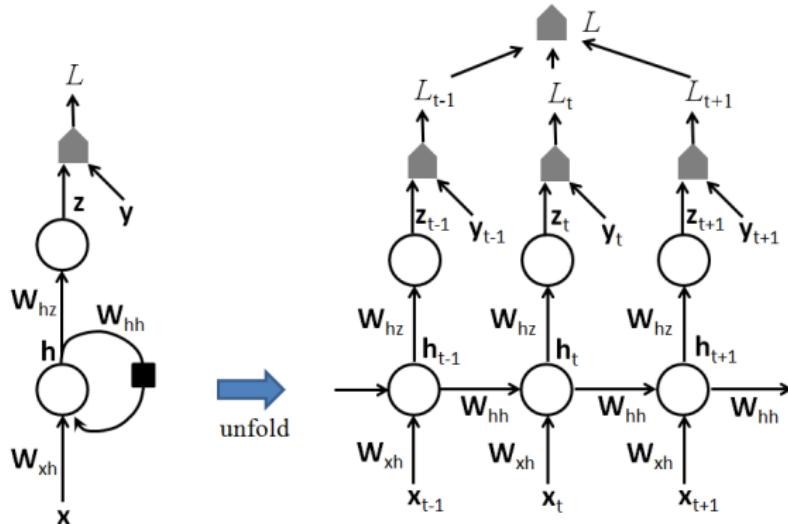
$$p(y_t = c) = z_{t,c}$$



Cost function

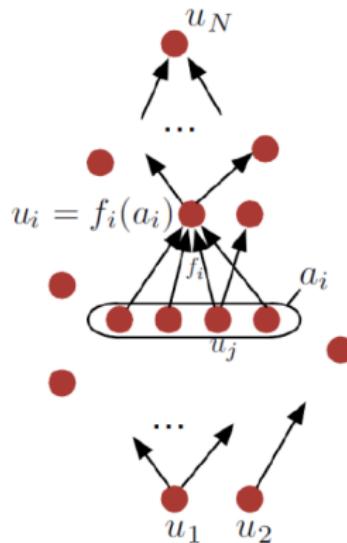
- The total loss for a given input/target sequence pair (\mathbf{x}, \mathbf{y}) , measured in cross entropy

$$L(\mathbf{x}, \mathbf{y}) = \sum_t L_t = \sum_t -\log z_{t,y_t}$$



Backpropagation on RNN

- Review BP on flow graph



```

 $\frac{\partial u_N}{\partial u_N} \leftarrow 1$ 
for  $j = N - 1$  down to 1 do
     $\frac{\partial u_N}{\partial u_j} \leftarrow \sum_{i:j \in \text{parents}(i)} \frac{\partial u_N}{\partial u_i} \frac{\partial u_i}{\partial u_j}$ 
end for
return  $\left( \frac{\partial u_N}{\partial u_i} \right)_{i=1}^M$ 

```

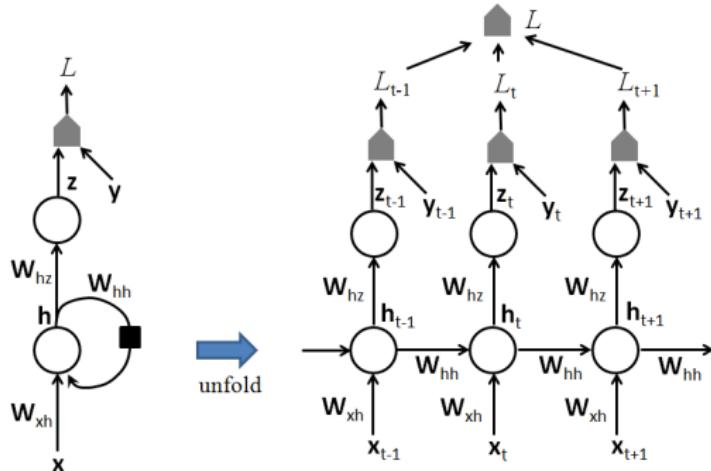
(Bengio et al. Deep Learning 2014)

$$\frac{\partial u_N}{\partial w_{ji}} = \frac{\partial u_N}{\partial u_i} \frac{\partial u_i}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial w_{ji}}$$

Gradients on \mathbf{W}_{hz} and \mathbf{b}_z

$$\frac{\partial L}{\partial L_t} = 1, \quad \frac{\partial L}{\partial \mathbf{z}_t} = \frac{\partial L}{\partial L_t} \frac{\partial L_t}{\partial \mathbf{z}_t} = \frac{\partial L_t}{\partial \mathbf{z}_t}$$

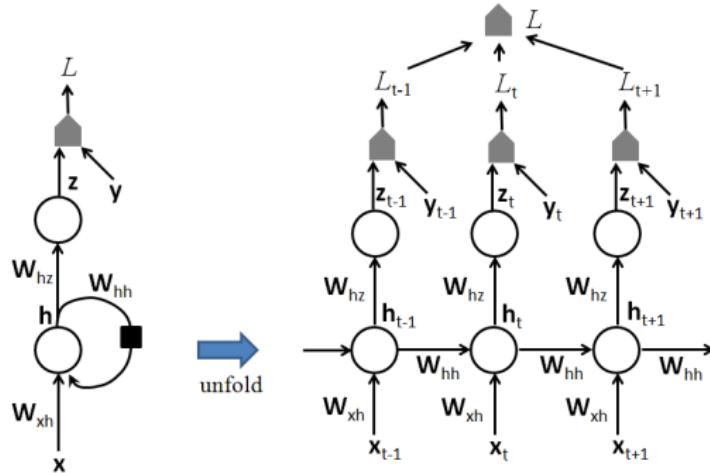
$$\frac{\partial L}{\partial \mathbf{W}_{hz}} = \sum_t \frac{\partial L_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{W}_{hz}}, \quad \frac{\partial L}{\partial \mathbf{b}_z} = \sum_t \frac{\partial L_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{b}_z}$$



Gradients on \mathbf{W}_{hh} and \mathbf{W}_{xh}

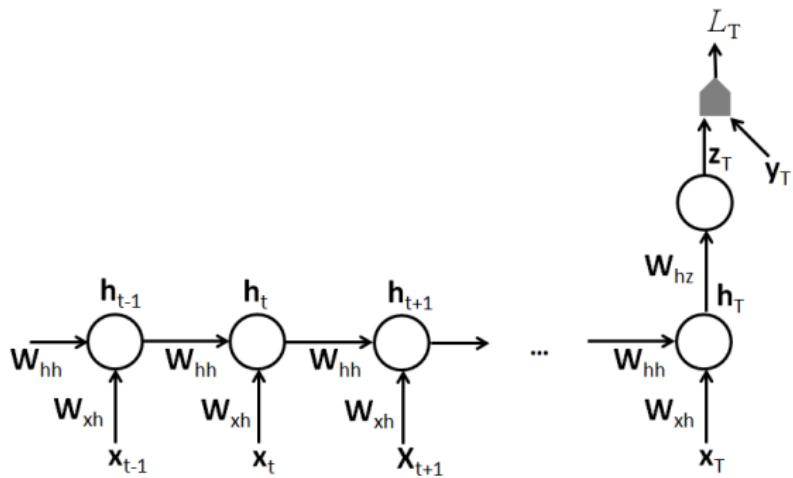
$$\frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_t \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hh}}$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_t}$$



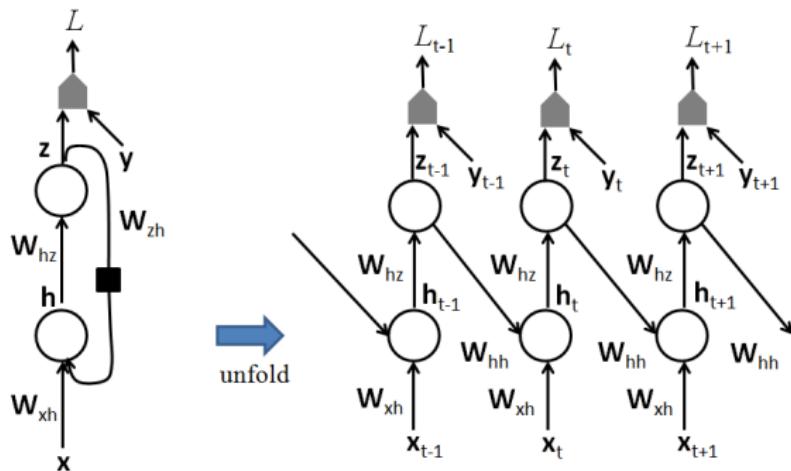
Predict a single output at the end of the sequence

- Such a network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing. There might be a target right at the end or the gradient on the output \mathbf{z}_t can be obtained by backpropagation from further downstream modules



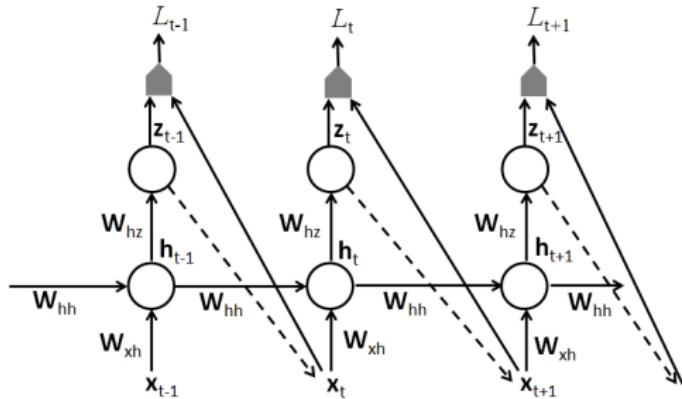
Network with output recurrence

- Memory is from the prediction of the previous target, which limits its expressive power but makes it easier to train



Generative RNN modeling $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$

- It can generate sequences from this distribution
- At the training stage, each \mathbf{x}_t of the observed sequence serves both as input (for the current time step) and as target (for the previous time step)
- The output \mathbf{z}_t encodes the parameters of a conditional distribution
 $P(\mathbf{x}_{t+1}|\mathbf{x}_1, \dots, \mathbf{x}_t) = P(\mathbf{x}_{t+1}|\mathbf{z}_t)$ for \mathbf{x}_{t+1} given the past sequence $\mathbf{x}_1, \dots, \mathbf{x}_t$



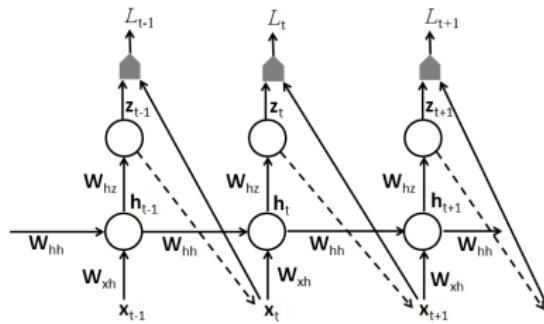
Generative RNN modeling $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$

- Cost function: negative log-likelihood of \mathbf{x} , $L = \sum_t L_t$

$$P(\mathbf{x}) = P(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T P(\mathbf{x}_t | \mathbf{x}_{t-1}, \dots, \mathbf{x}_1)$$

$$L_t = -\log P(\mathbf{x}_t | \mathbf{x}_{t-1}, \dots, \mathbf{x}_1)$$

- In generative mode, \mathbf{x}_{t+1} is sampled from the conditional distribution $P(\mathbf{x}_{t+1} | \mathbf{x}_1, \dots, \mathbf{x}_t) = P(\mathbf{x}_{t+1} | \mathbf{z}_t)$ (dashed arrows) and then that generated sample \mathbf{x}_{t+1} is fed back as input for computing the next state \mathbf{h}_{t+1}



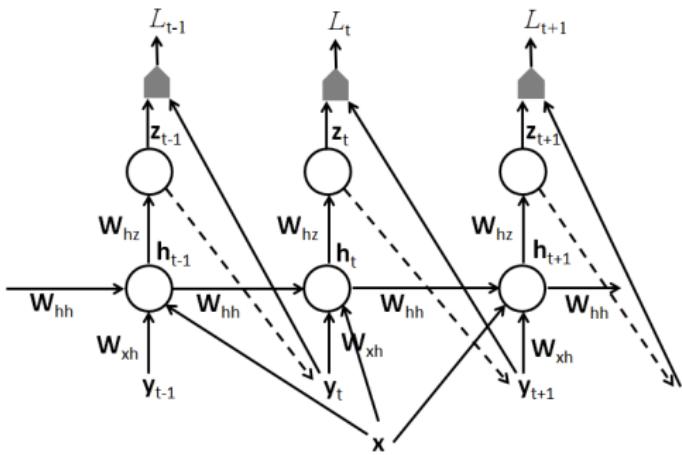
Generative RNN modeling $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$

- If RNN is used to generate sequences, one must also incorporate in the output information allowing to stochastically decide when to stop generating new output elements
- In the case when the output is a symbol taken from a vocabulary, one can add a special symbol corresponding to the end of a sequence
- One could also directly model the length T of the sequence through some parametric distribution. $P(\mathbf{x}_1, \dots, \mathbf{x}_T)$ is decomposed into

$$P(\mathbf{x}_1, \dots, \mathbf{x}_T) = P(\mathbf{x}_1, \dots, \mathbf{x}_T | T)P(T)$$

RCNNs to represent conditional distributions $P(\mathbf{y}|\mathbf{x})$

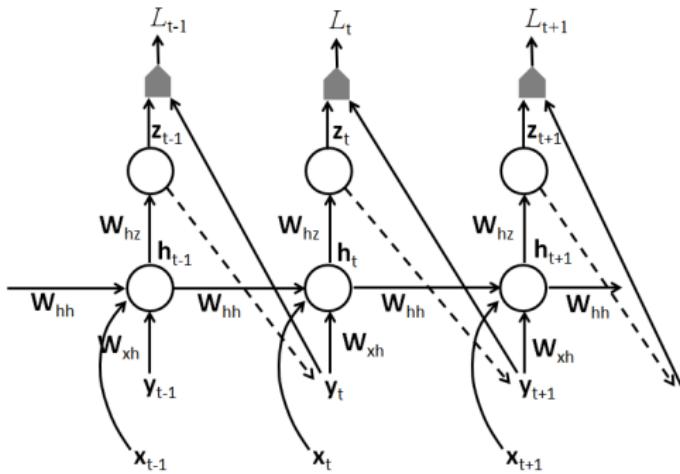
- If \mathbf{x} is a fixed-sized vector, we can simply make it an extra input of the RNN that generates the \mathbf{y} sequence. Some common ways of providing the extra input
 - as an extra input at each time step, or
 - as the initial state \mathbf{h}_0 , or
 - both
- Example: generate caption for an image



RCNNs to represent conditional distributions $P(\mathbf{y}|\mathbf{x})$

- The input \mathbf{x} is a sequence of the same length as the output sequence \mathbf{y}
- Removing the dash lines, it assumes \mathbf{y}_t 's are independent of each other when the past input sequence is given, i.e.

$$P(\mathbf{y}_t|\mathbf{y}_{t-1}, \dots, \mathbf{y}_1, \mathbf{x}_t, \dots, \mathbf{x}_1) = P(\mathbf{y}_t|\mathbf{x}_t, \dots, \mathbf{x}_1)$$
- Without the conditional independence assumption, add the dash lines and the prediction of \mathbf{y}_{t+1} is based on both the past \mathbf{x} 's and past \mathbf{y} 's

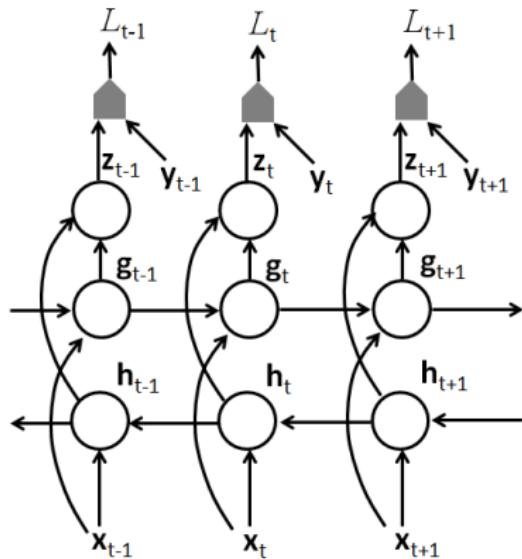


Bidirectional RNNs

- In some applications, we want to output at time t a prediction regarding an output which may depend on the whole input sequence
 - In speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because co-articulation and may depend on the next few words because of the linguistic dependencies between words
- Bidirectional recurrent neural network was proposed to address such need
- It combines a forward-going RNN and a backward-going RNN
- The idea can be extended to 2D input with four RNN going in four directions

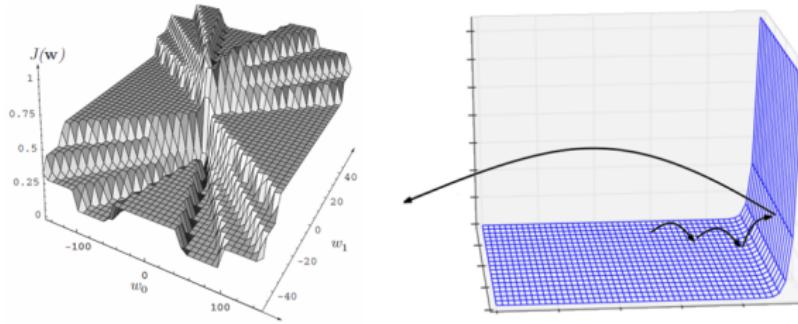
Bidirectional RNNs

- \mathbf{g}_t summarizes the information from the past sequence, and \mathbf{h}_t summarizes the information from the future sequence



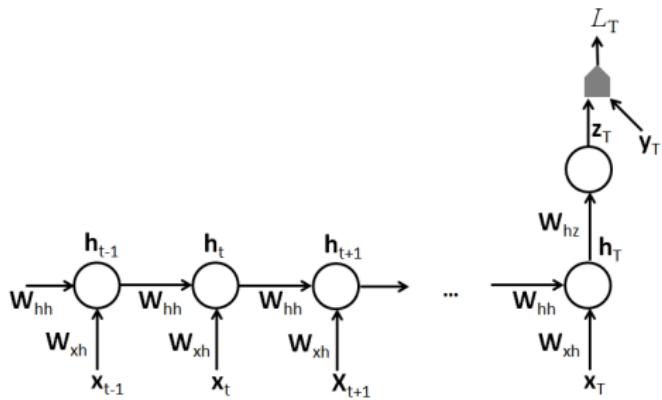
Plateaus and cliffs

- The error surfaces of training deep neural networks include local minima, plateaus (regions where error varies only slightly as a function of weights), and cliffs (regions where the gradients rise sharply)
- Plateaus and cliffs are more important barriers to training neural networks than local minima
 - It is very difficult (or slow) to effectively update the parameters in plateaus
 - When the parameters approach a cliff region, the gradient update step can move the learner towards a very bad configuration, ruining much progress made during recent training iterations.



Vanishing and exploding gradients

- Training a very deep net makes the problem even more serious, since after BP through many layers, the gradients become either very small or very large
- RNN can be treated as a deep net when modeling long term dependency



Vanishing and exploding gradients

- In very deep nets and recurrent nets, the final output is composed of a large number of non-linear transformations
- Even though each of these non-linear stages may be relatively smooth, their composition is going to be much “more non-linear”, in the sense that the derivatives through the whole composition will tend to be either very small or very large, with more ups and downs



When composing many non-linearities (like the activation non-linearity in a deep or recurrent neural network), the result is highly non-linear, typically with most of the values associated with a tiny derivative, some values with a large derivative, and many ups and downs (not shown here)

Vanishing and exploding gradients

This arises because the Jacobian (matrix of derivatives) of a composition is the product of the Jacobian of each stage, i.e. if

$$f = f_T \circ f_{T-1} \circ \dots f_2 \circ f_1$$

The Jacobian matrix of derivatives of $f(x)$ with respect to its input vector \mathbf{x} is

$$f' = f'_T f'_{T-1} \dots f'_2 f'_1$$

where

$$f' = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$$

and

$$f'_t = \frac{\partial f_t(\alpha_t)}{\partial \alpha_t}$$

where $\alpha_t = f_{t-1}(f_{t-1}(\dots f_2(f_1(\mathbf{x}))))$, i.e. composition has been replaced by matrix multiplication

Vanishing and exploding gradients

- In the scalar case, we can imagine that multiplying many numbers together tends to be either very large or very small
- In the special case where all the numbers in the product have the same value α , this is obvious, since α^T goes to 0 if $\alpha < 1$ and to ∞ if $\alpha > 1$ as T increases
- The more general case of non-identical numbers be understood by taking the logarithm of these numbers, considering them to be random, and computing the variance of the sum of these logarithms. Although some cancellation can happen, the variance grows with T . If those numbers are independent, it grows linearly with T , which means that the product grows roughly as e^T .
- This analysis can be generalized to the case of multiplying square matrices

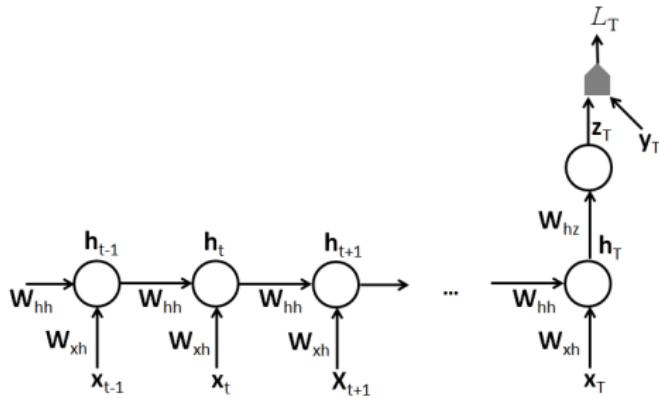
Difficulty of Learning Long-Term Dependencies

- Consider the gradient of a loss L_T at time T with respect to the parameter θ of the recurrent function F_θ

$$\mathbf{h}_t = F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\frac{\partial L_T}{\partial \theta} = \sum_{t \leq T} \frac{\partial L_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \frac{\partial F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)}{\partial \theta}$$

$\frac{\partial L_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \frac{\partial F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)}{\partial \theta}$ encodes long-term dependency when $T - t$ is large



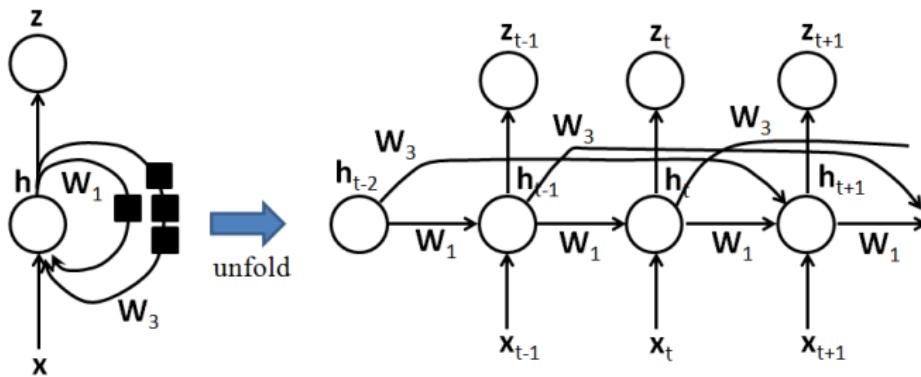
Difficulty of Learning Long-Term Dependencies

$$\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} = \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{h}_{T-2}} \cdots \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$

- Each layer-wise Jacobian $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$ is the product of two matrices: (a) the recurrent matrix \mathbf{W} and (b) the diagonal matrix whose entries are the derivatives of the non-linearities associated with the hidden units, which vary depending on the time step. This makes it likely that successive Jacobians have similar eigenvectors, making the product of these Jacobians explode or vanish even faster
- $\frac{\partial L_T}{\partial \theta}$ is a weighted sum of terms over spans $T - t$, with weights that are exponentially smaller (or larger) for long-term dependencies relating the state at t to the state at T
- The signal about long term dependencies will tend to be hidden by the smallest fluctuations arising from short-term dependencies

Combine short and long paths in unfolded flow graph

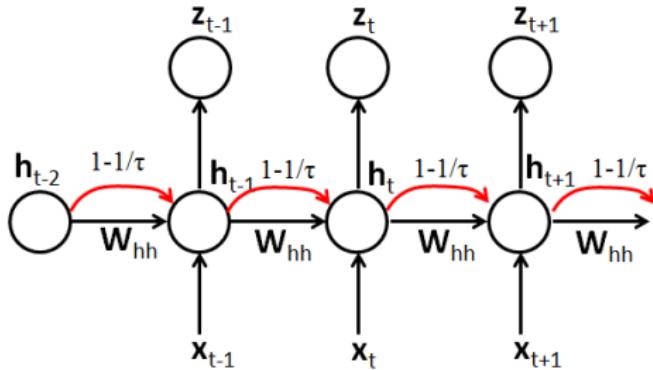
- Longer-delay connections allow to connect the past states to future states through short paths
- Gradients will vanish exponentially with respect to the number of time steps
- If we have recurrent connections with a time-delay of D , the instead of the vanishing or explosion going as $O(\lambda^T)$ over T steps (where λ is largest eigenvalue of the Jacobians $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$), the unfolded recurrent network now has paths through which gradients grow as $O(\lambda^{T/D})$ because the number of effective steps is T/D



Leaky units with self-connections

$$\mathbf{h}_{t+1} = \left(1 - \frac{1}{\tau_i}\right)\mathbf{h}_t + \frac{1}{\tau_i}\tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_t + \mathbf{b}_h)$$

- The new value of the state \mathbf{h}_{t+1} is a combination of linear and non-linear parts of \mathbf{h}_t
- The errors are easier to be back propagated through the paths of red lines, which are linear



Leaky units with self-connections

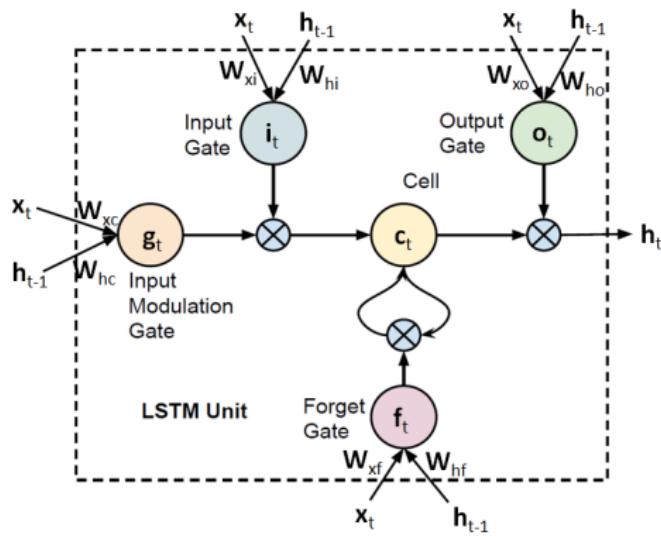
- When $\tau = 1$, there is no linear self-recurrence, only the nonlinear update which we can find in ordinary recurrent networks
- When $\tau > 1$, this linear recurrence allows gradients to propagate more easily. When τ is large, the state changes very slowly, integrating the past values associated with the input sequence
- τ controls the rate of forgetting old states. It can be viewed as a smooth variant of the idea of the previous model
- By associating different time scales τ with different units, one obtains different paths corresponding to different forgetting rates
- Those time constants can be fixed manually or can be learned as free parameters

Long Short-Term Memory (LSTM) net

- In the leaky units with self-connections, the forgetting rate is constant during the whole sequence.
- The role of leaky units is to accumulate information over a long duration. However, once that information gets used, it might be useful for the neural network to forget the old state.
 - For example, if a video sequence is composed as subsequences corresponding to different actions, we want a leaky unit to accumulate evidence inside each subsequence, and we need a mechanism to forget the old state by setting it to zero and starting to count from fresh when starting to process the next subsequence
- The forgetting rates are expected to be different at different time steps, depending on their previous hidden states and current input (conditioning the forgetting on the context)
- Parameters controlling the forgetting rates are learned from train data

Long Short-Term Memory (LSTM) net

$$\begin{aligned}
 f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f), \quad i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i), \\
 o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
 g_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \quad c_t = f_t \odot c_{t-1} + i_t \odot g_t \\
 h_t &= o_t \odot \tanh(c_t), \quad z_t = \text{softmax}(W_{hz}h_t + b_z)
 \end{aligned}$$

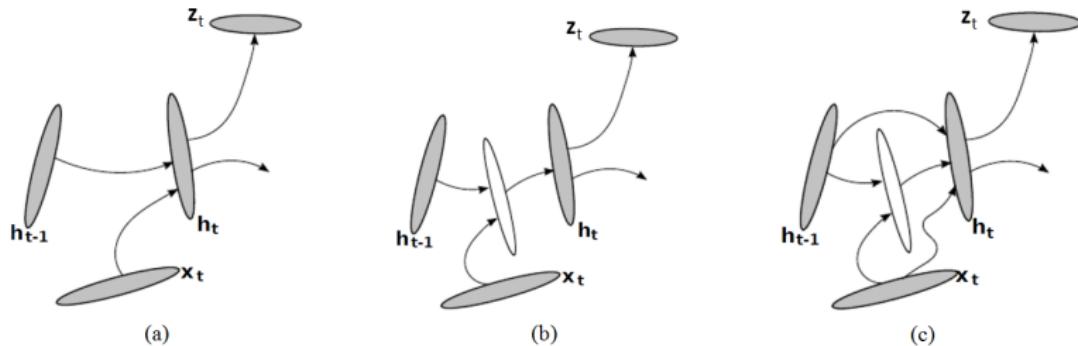


Long Short-Term Memory (LSTM) net

- The core of LSTM is a memory cell \mathbf{c}_t which encodes, at every time step, the knowledge of the inputs that have been observed up to that step.
- The memory cell \mathbf{c}_t has the same inputs (\mathbf{h}_{t-1} and \mathbf{x}_t) and outputs (\mathbf{h}_t) as a normal recurrent network, but has more parameters and a system of gating units that controls the flow of information
- \mathbf{c}_t has a linear self-connection similar to the leaky units, but the self-connection weight is controlled by a forget gate unit \mathbf{f}_t , that sets this weight to a value between 0 and 1 via a sigmoid unit
$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$$
- The input gate unit \mathbf{i}_t is computed similarly to the forget gate, but with its own parameters
- The output \mathbf{h}_t of the LSTM cell can also be shut off , via the output gate \mathbf{o}_t ($\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$), which is also a sigmoid unit for gating
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$$

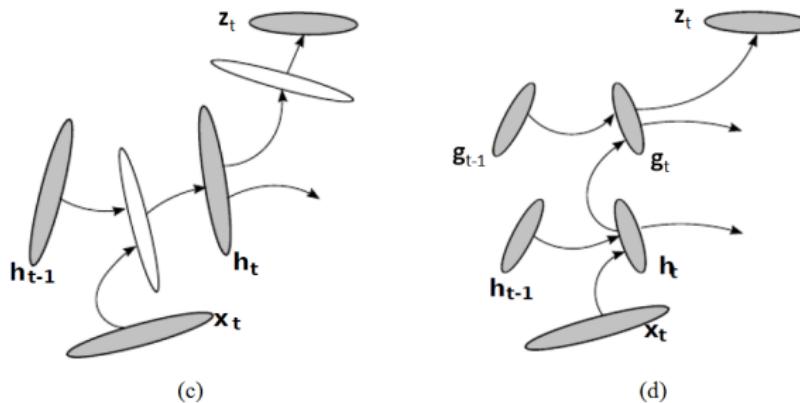
Long Short-Term Memory (LSTM) net

- (a): A vanilla RNN with input sequence and an output sequence
- (b): Add a deep hidden-to-hidden transformation
- (c): Skip connections and allow gradients to flow more easily backwards in spite of the extra non-linearity due to the intermediate hidden layer



Long Short-Term Memory (LSTM) net

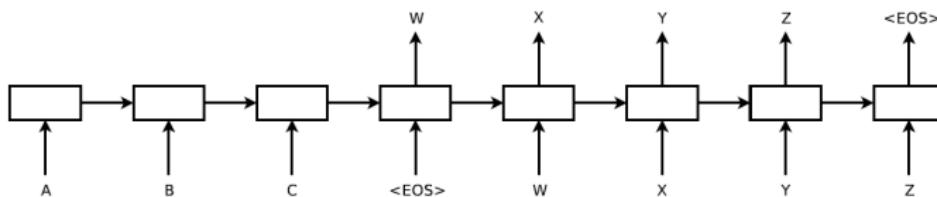
- (c): Depth can also be added in the hidden-to-output transform
- (d): A hierarchy of RNNs, which can be stacked on top of each other



Sequence-to-sequence language translation

- Sutskever, Vinyals, and Le NIPS 2014
- Model $P(\mathbf{y}_1, \dots, \mathbf{y}_{T'} | \mathbf{x}_1, \dots, \mathbf{x}_T)$. The input and output sequences have different lengths, are not aligned, and even do not have monotonic relationship
- Use one LSTM to read the input sequence $(\mathbf{x}_1, \dots, \mathbf{x}_T)$, one timestep at a time, to obtain a large fixed-dimensional vector representation \mathbf{v} , which is given by the last hidden state of the LSTM
- Then conditioned on \mathbf{v} , a second LSTM generates the output sequence $(\mathbf{y}_1, \dots, \mathbf{y}_{T'})$ and computes its probability

$$p(\mathbf{y}_1, \dots, \mathbf{y}_{T'} | \mathbf{v}) = \prod_{t=1}^{T'} p(\mathbf{y}_t | \mathbf{v}, \mathbf{y}_1, \dots, \mathbf{y}_{t-1})$$

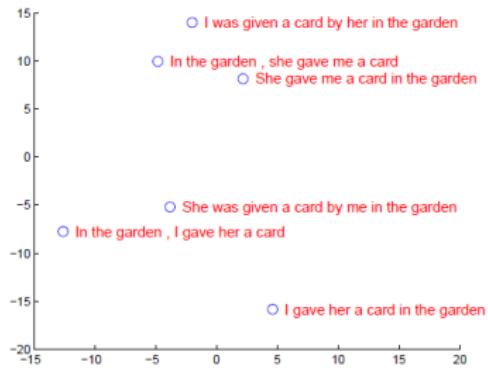
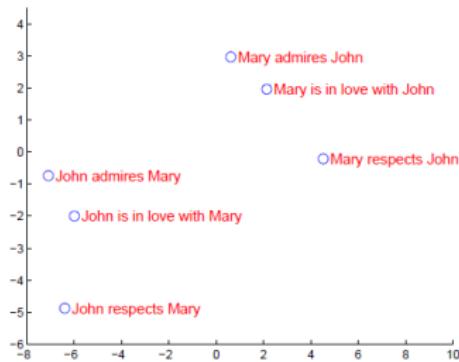


The model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token.

Sequence-to-sequence language translation

- It requires each sentence ends with a special end-of-sequence symbol “<EOS>”, which enables the model to define a distribution over sequences of all possible lengths
- It is valuable to reverse the order of the words of the input sequence. For example, instead of mapping the sentence a, b, c to the sentence α, β, γ , the LSTM is asked to map c, b, a to α, β, γ , where α, β, γ is the translation of a, b, c . This way, a is in close proximity to α , b is fairly close to β , and so on, a fact that makes it easy for stochastic gradient descent to “establish communication” between the input and the output. It introduces many short term dependencies in the data that make the optimization problem much easier.

Sequence-to-sequence language translation



The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures. The phrases are clustered by meaning, which in these examples is primarily a function of word order, which would be difficult to capture with a bag-of-words model. The figure clearly shows that the representations are sensitive to the order of words, while being fairly insensitive to the replacement of an active voice with a passive voice.

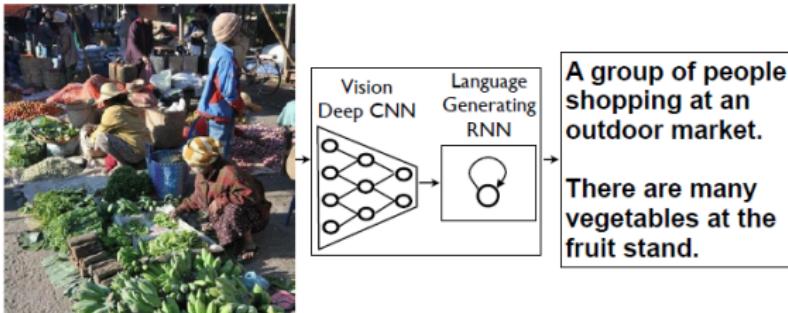
Sequence-to-sequence language translation

- LSTM can correctly translate very long sentences

Type	Sentence
Our model	Ulrich UNK , membre du conseil d' administration du constructeur automobile Audi , affirme qu' il s' agit d' une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d' administration afin qu' ils ne soient pas utilisés comme appareils d' écoute à distance .
Truth	Ulrich Hackenberg , membre du conseil d' administration du constructeur automobile Audi , déclare que la collecte des téléphones portables avant les réunions du conseil , afin qu' ils ne puissent pas être utilisés comme appareils d' écoute à distance , est une pratique courante depuis des années .
Our model	" Les téléphones cellulaires , qui sont vraiment une question , non seulement parce qu' ils pourraient potentiellement causer des interférences avec les appareils de navigation , mais nous savons , selon la FCC , qu' ils pourraient interférer avec les tours de téléphone cellulaire lorsqu' ils sont dans l' air " , dit UNK .
Truth	" Les téléphones portables sont véritablement un problème , non seulement parce qu' ils pourraient éventuellement créer des interférences avec les instruments de navigation , mais parce que nous savons , d' après la FCC , qu' ils pourraient perturber les antennes-relais de téléphonie mobile s' ils sont utilisés à bord " , a déclaré Rosenker .
Our model	Avec la crémation , il y a un " sentiment de violence contre le corps d' un être cher " , qui sera " réduit à une pile de cendres " en très peu de temps au lieu d' un processus de décomposition " qui accompagnera les étapes du deuil " .
Truth	Il y a , avec la crémation , " une violence faite au corps aimé " , qui va être " réduit à un tas de cendres " en très peu de temps , et non après un processus de décomposition , qui " accompagnerait les phases du deuil " .

Generate image caption

- Vinyals et al. arXiv 2014
- Use a CNN as an image encoder and transform it to a fixed-length vector
- It is used as the initial hidden state of a “decoder” RNN that generates the target sequence



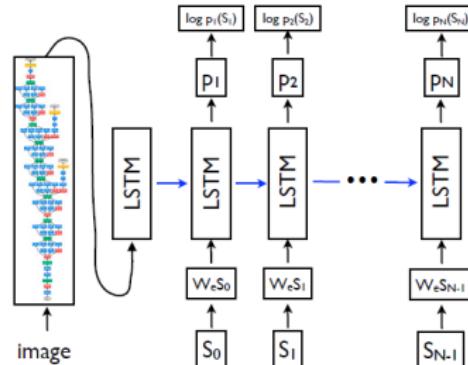
Generate image caption

- The learning process is to maximize the probability of the correct description given the image

$$\theta^* = \arg \max_{(\mathbf{I}, \mathbf{S})} \sum \log P(\mathbf{S}|\mathbf{I}; \theta)$$

$$\log P(\mathbf{S}|\mathbf{I}) = \sum_{t=0}^N \log P(\mathbf{s}_t | \mathbf{I}, \mathbf{s}_0, \dots, \mathbf{s}_{t-1})$$

\mathbf{I} is an image and \mathbf{S} is its correct description



Generate image caption

- Denote by \mathbf{S}_0 a special start work and by \mathbf{S}_N a special stop word
- Both the image and the words are mapped to the same space, the image by using CNN, the words by using word embedding \mathbf{W}_e
- The image \mathbf{I} is only input once at $t = 1$ to inform the LSTM about the image contents
- Sampling: sample the first word according to P_1 , then provide the corresponding embedding as input and sample P_2 , continuing like this until it samples the special end-of-sentence token

$$\mathbf{x}_{-1} = \text{CNN}(\mathbf{I})$$

$$\mathbf{x}_t = \mathbf{W}_e \mathbf{s}_t, t \in \{0, \dots, N-1\}$$

$$P_{t+1} = \text{LSTM}(\mathbf{x}_t), t \in \{0, \dots, N-1\}$$

$$L(\mathbf{I}, \mathbf{S}) = - \sum_{t=1}^N \log P_t(\mathbf{s}_t)$$

Translate videos to sentences

- Venugopalan et al. arXiv 2014
- The challenge is to capture the joint dependencies of a sequence of frames and a corresponding sequence of words
- Previous works simplified the problem by detecting a fixed set of semantic roles, such as subject, verb, and object, as an intermediate representation and adopted oversimplified rigid sentence templates.

Input video:

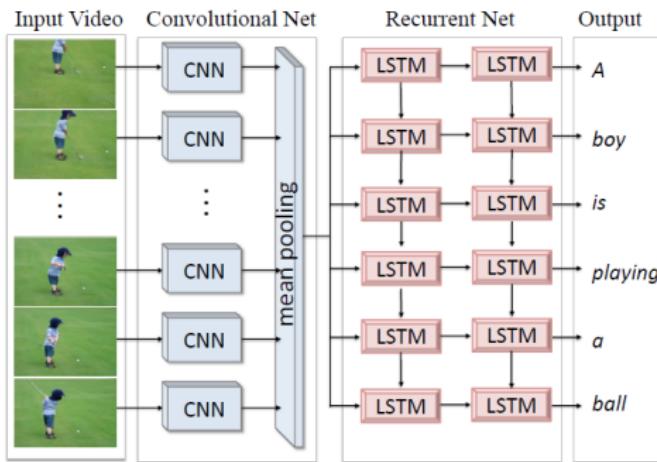


Machine output: *A cat is playing with toy.*

Humans: *A Ferret and cat fighting with each other. / A cat and a ferret are playing. / A kitten and a ferret are playfully wrestling.*

Translate videos to sentences

- Each frame is modeled as CNN pre-trained on ImageNet
- The meaning state and sequence of words is modeled by a RNN pre-trained on images with associated with sentence captions



Translate videos to sentences

- Use CNN to convert a video to a fixed length representation vector \mathbf{v}
- Use RNN to decode the vector into a sentence just like language translation

$$p(\mathbf{y}_1, \dots, \mathbf{y}_{T'} | \mathbf{v}) = \prod_{t=1}^{T'} p(\mathbf{y}_t | \mathbf{v}, \mathbf{y}_1, \dots, \mathbf{y}_{t-1})$$

- Use two layers of LSTMs (one LSTM stacked on top of another)

Translate videos to sentences



FGM: A person is playing a guitar in the house.
YT: A group of performing on stage.
YT_C: A man is doing a trick.
YT_CF: A man is jumping on a pole.
GT: Two men working on a high building.



FGM: A person is playing a guitar in the house.
YT: A boy is walking.
YT_C: A man is doing a women.
YT_CF: A man is talking on a wall.
GT: A man is doing algebraic equations on a white board.



FGM: A person is riding the horse
YT: A group of running.
YT_C: A group of elephants.
YT_CF: A group of elephants are walking on a horse.
GT: An elephant leads it's young.



FGM: A person playing the goal of the road.
YT: A player player in a goal.
YT_C: A man playing a soccer ball.
YT_CF: A soccer player is running.
GT: Two teams are playing soccer.



FGM: A person is running a race on the road.
YT: A group of running.
YT_C: A group of people are running.
YT_CF: A man is running.
GT: Eight men are running a race on a track.

FGM: factor graph model, using templates to generate sentences

YT: LSTM trained on the YouTube video dataset

YT_C: LSTM with pre-training on the Coco image dataset

YT_CF: LSTM with pre-training on the CoCo and Flickr image datasets

GT: Ground truth from human description

Reading Materials

- R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern Classification," Chapter 6, 2000.
- Y. Bengio, I. J. Goodfellow and A. Courville, "Sequence Modeling: Recurrent and Recursive Nets" in "Deep Learning", Book in preparation for MIT Press, 2014.
- I. Sutskever, O. Vinyals, and Q. Le, "Sequence to Sequence Learning with Neural Networks," NIPS 2014.
- S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, K. Saenko, "Translating Videos to Natural Language Using Deep Recurrent Neural Networks," arXiv: 1412.4729, 2014.
- J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description," arXiv:1411.4389, 2014.
- O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and Tell: A Neural Image Caption Generator," arXiv: 1411.4555, 2014.



Examples of Deep Learning Applications

Xiaogang Wang

Department of Electronic Engineering,
The Chinese University of Hong Kong

Our first deep learning project
– January 2011



Wanli Ouyang

We wish to work on pedestrian detection

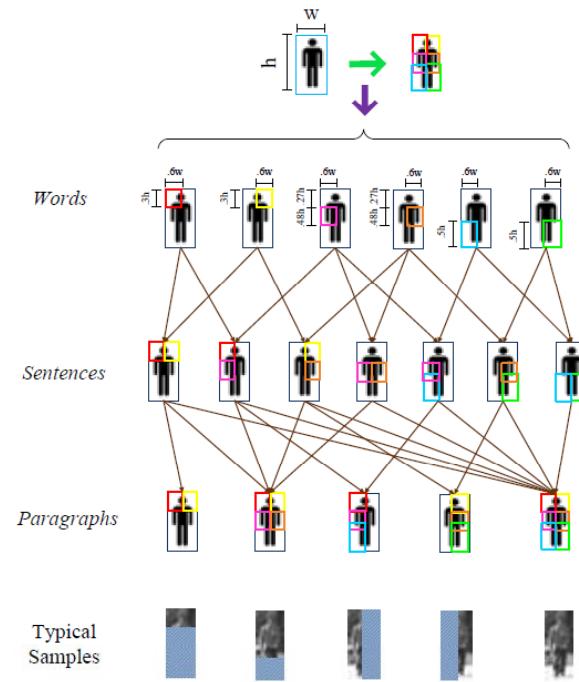
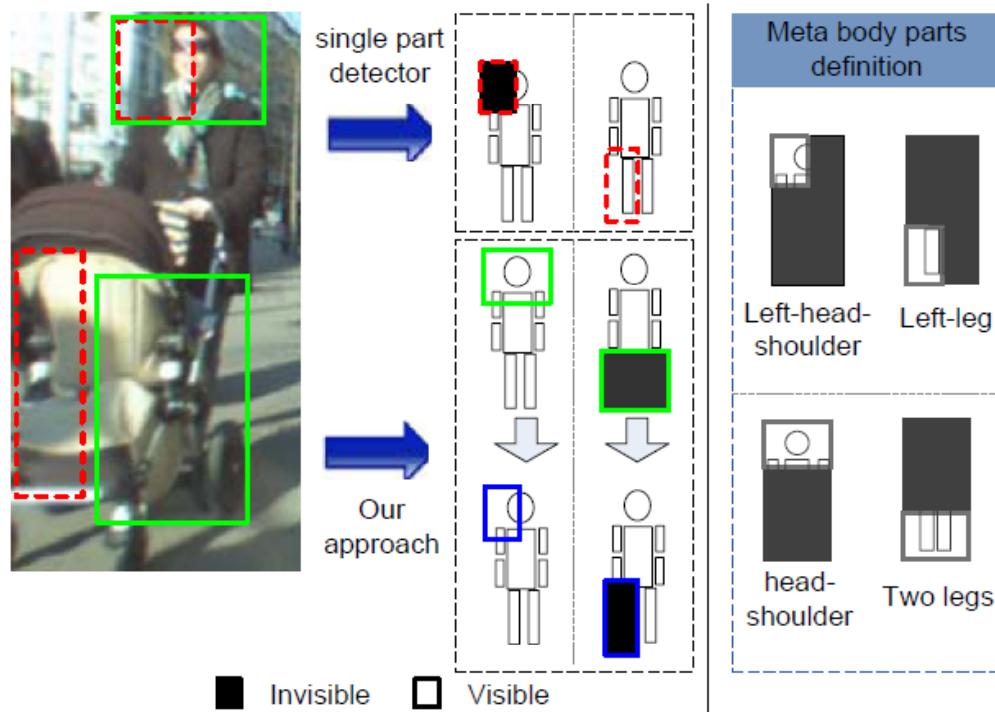
Where to start?

Our understanding of deep learning

- DBN
- Unsupervised learning
- Model complex nonlinear relationship of variables

Pedestrian detection

G. Duan, H. Ai, and S. Lao, "A structural filter approach to human detection," in ECCV, 2010.



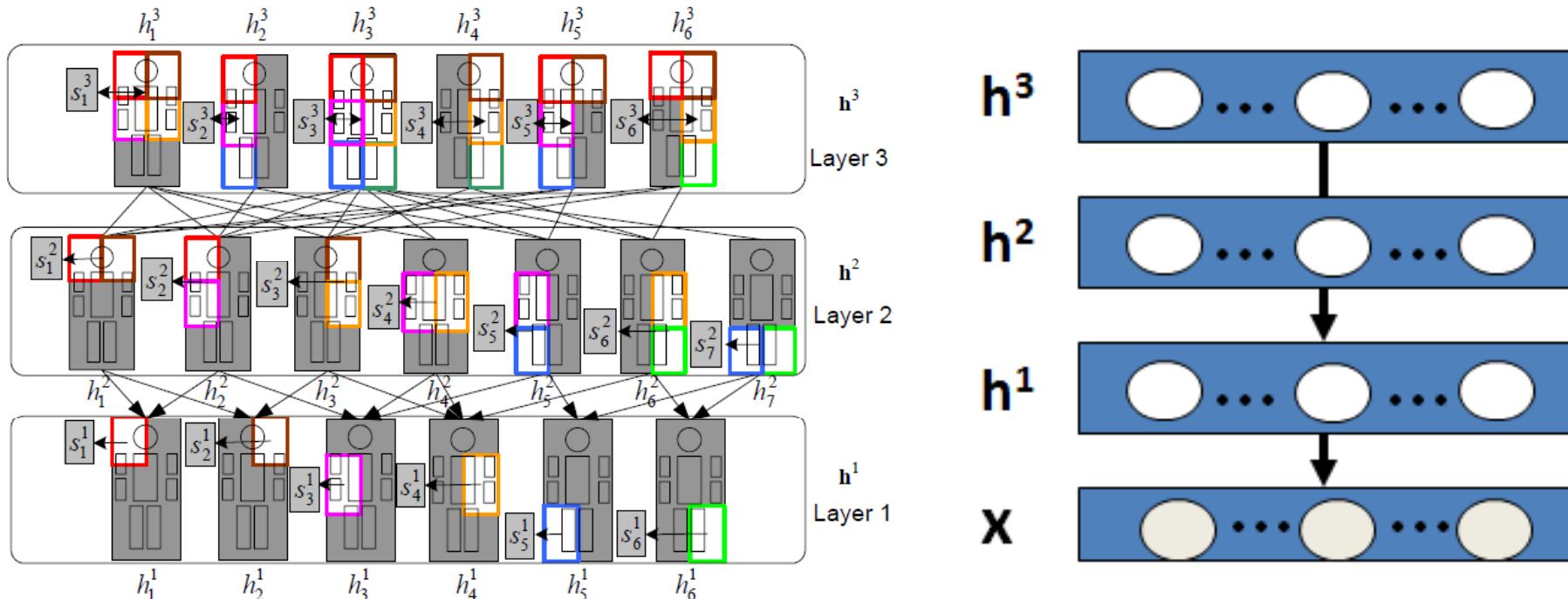
Use manually defined rules to describe the relationship between the visibility of a part and its overlapping larger parts and smaller parts, e.g. if the head or the torso was invisible, its larger part of upper-body should also be invisible.

Deep learning?

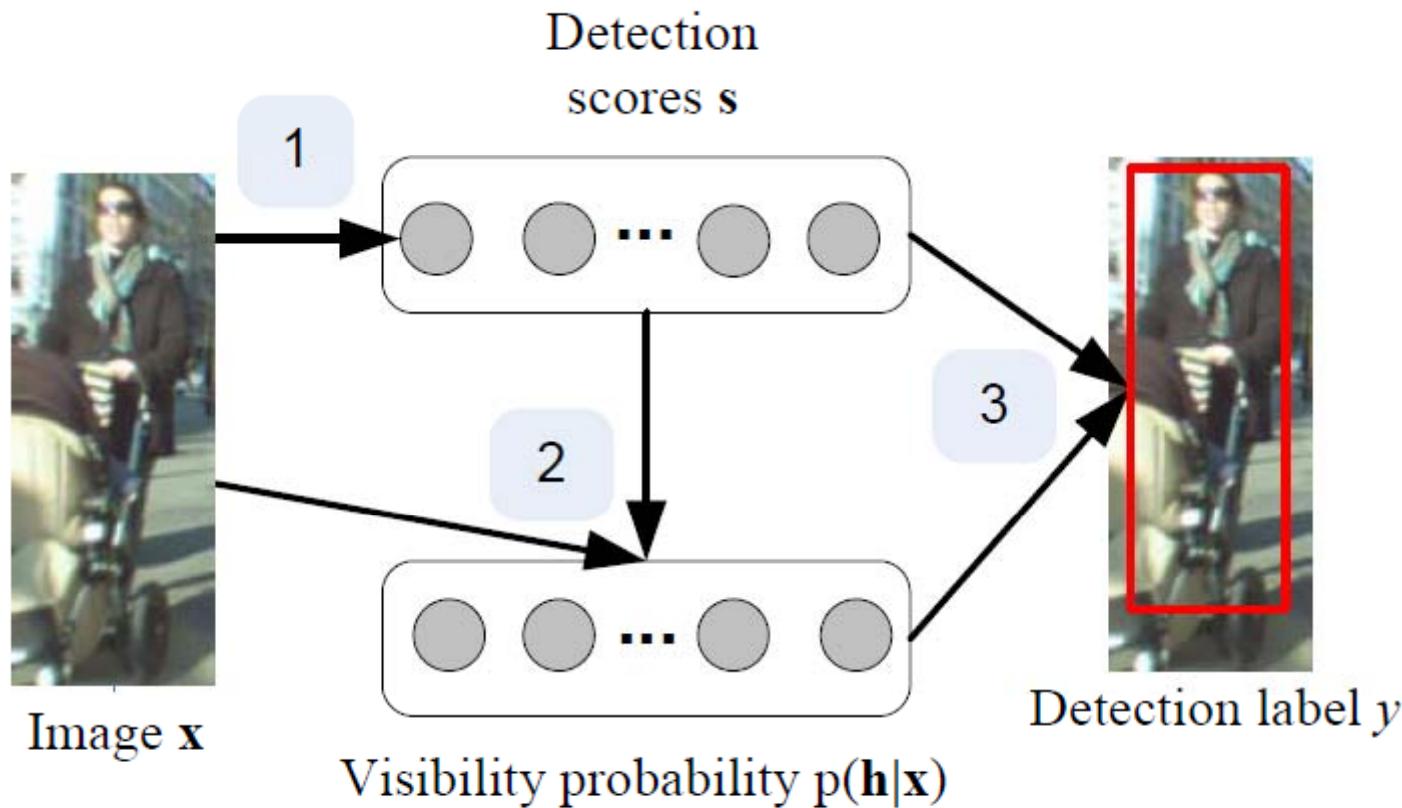
Deep belief net

$$p(y|\mathbf{x}) = \sum_{\mathbf{h}} p(y, \mathbf{h}|\mathbf{x}) = \sum_{\mathbf{h}} p(y|\mathbf{h}, \mathbf{x})p(\mathbf{h}|\mathbf{x})$$

- The hidden units in BDN have no physical meaning
- DBN is fully connected



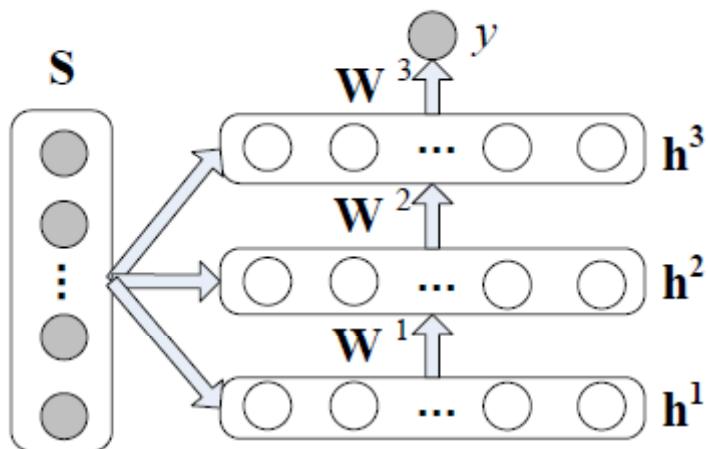
W. Ouyang and X. Wang, "A Discriminative Deep Model for Pedestrian Detection with Occlusion Handling," CVPR 2012



1. Part detection, 2. Visibility estimation,
3. Detection score integration

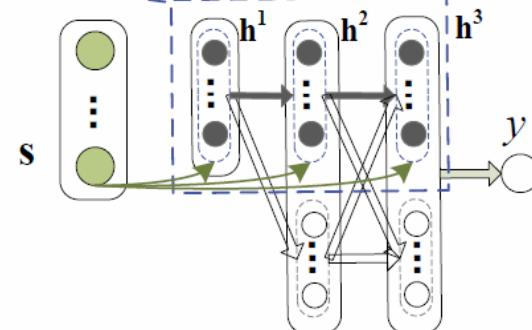
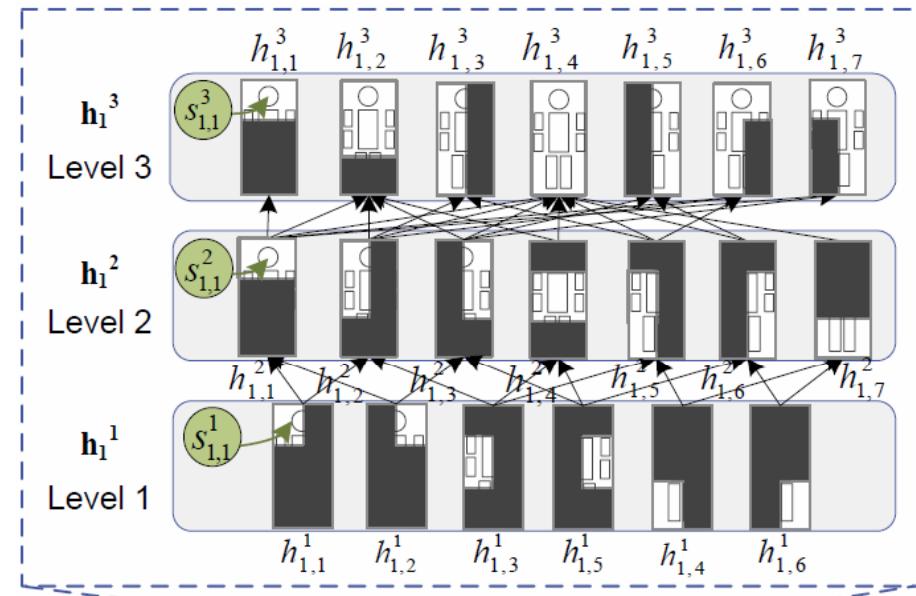
1. obtain the detection scores s by part detectors;
2. use s and x to estimate visibility probability $p(\mathbf{h}|x)$;
3. combine the detection scores s with the visibility probability $p(\mathbf{h}|x)$ to estimate the probability of an input window being pedestrian, c.f. (2) and (3).

- Each hidden unit is associated with a part detection score and it indicates the visibility of a part
- DBN is designed considering the structure of human body



$$\mathbf{W}^l = \mathbf{W}^{l,0} + \tilde{\mathbf{W}}^l \circ \tilde{\mathbf{S}}^l$$

$$\tilde{h}_j^{l+1} = \sigma(\tilde{\mathbf{h}}^{l\top} \mathbf{w}_{*,j}^l + c_j^{l+1} + g_j^{l+1} s_j^{l+1})$$



Correlates with part detection score

Structural filter

Manual design

Purely rely on
domain knowledge

Intuition is correct,
but very few
parameter setting
are explored



**Borrow the idea
from structural
filter, but allow to
explore many
more parameter
settings and learn
from data under
the formulation of
DBN**



DBN

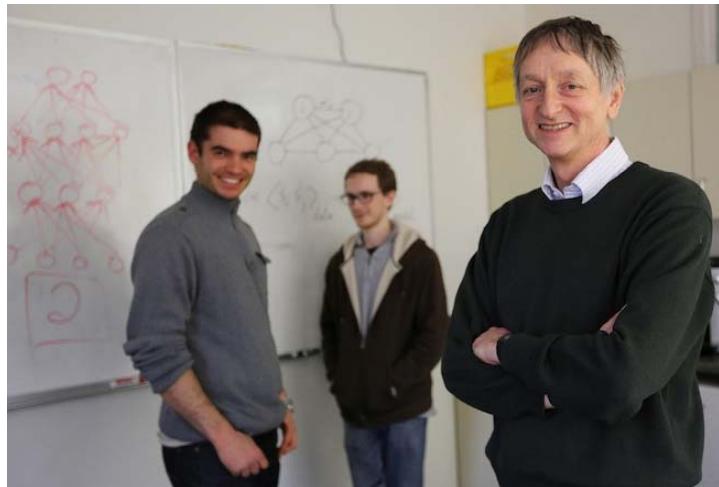
Learn from data

Black box

No domain
knowledge

No physical
meaning

Deep Learning Won ImageNet Image Classification Challenge 2012

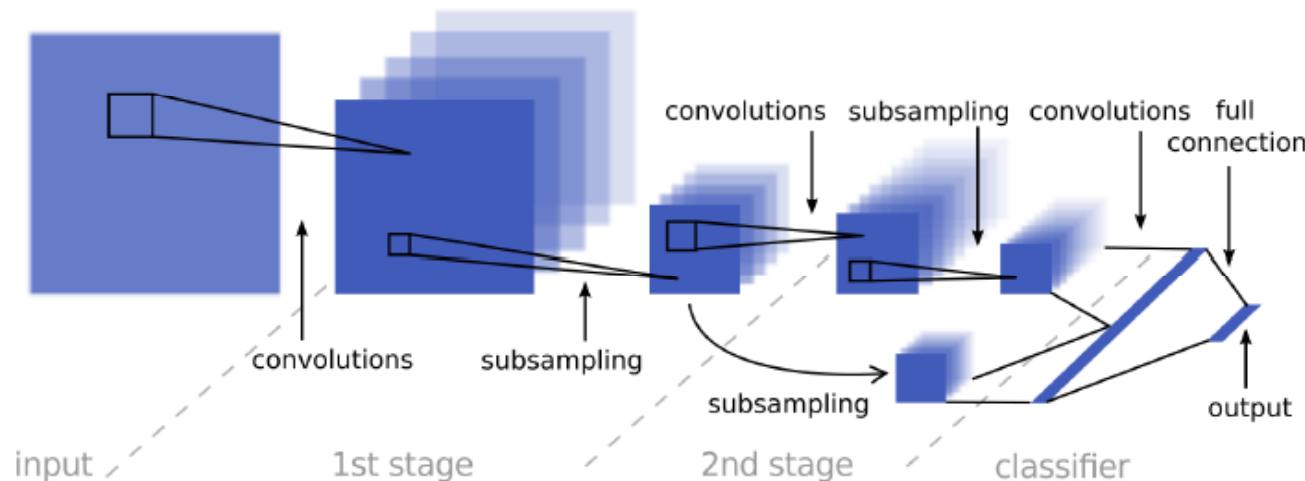


Our understanding of deep learning

- Large scale supervised learning with CNN
- The key of deep learning is to learn feature representation

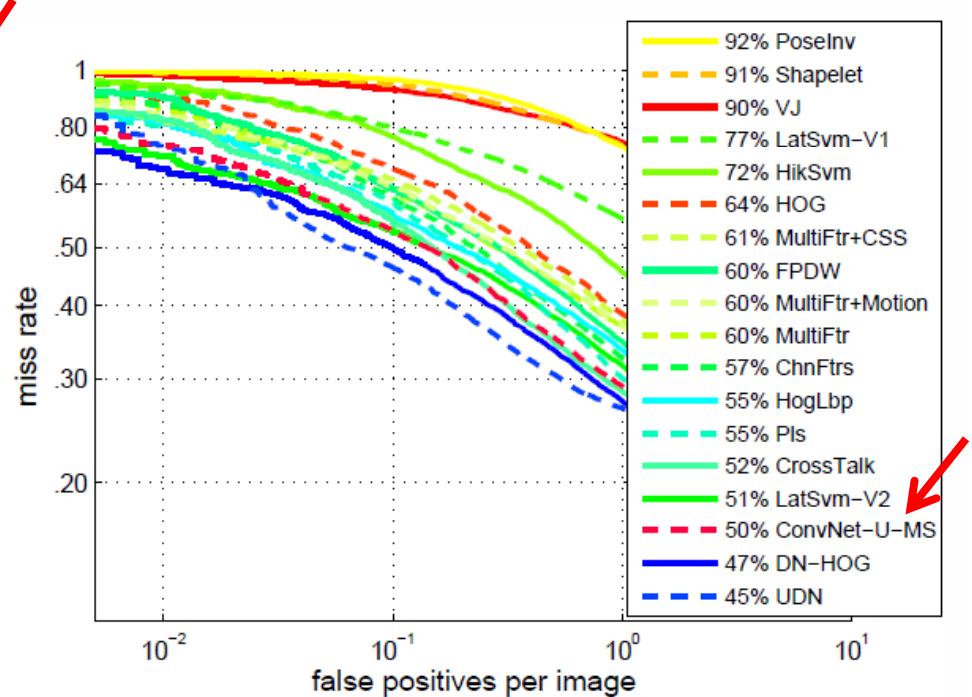
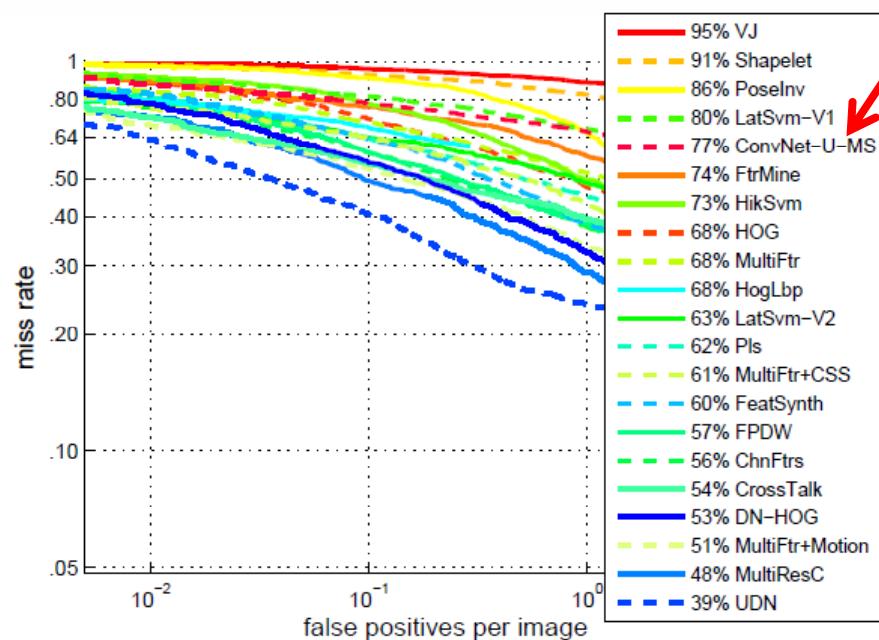
How to learn features in pedestrian detection?

It may not be a good idea to treat deep learning as a black box

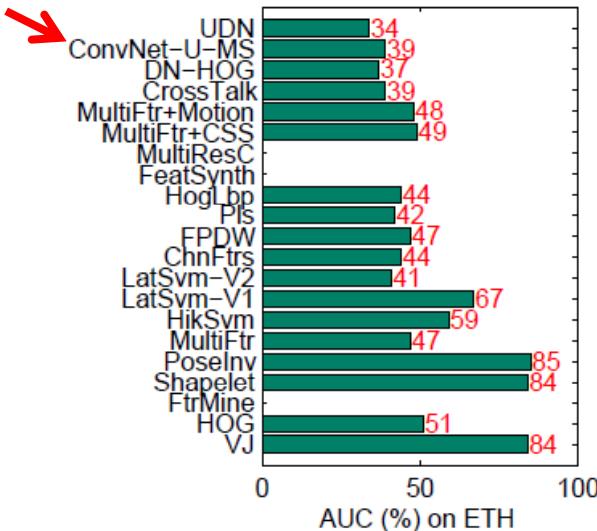


ConvNet-U-MS

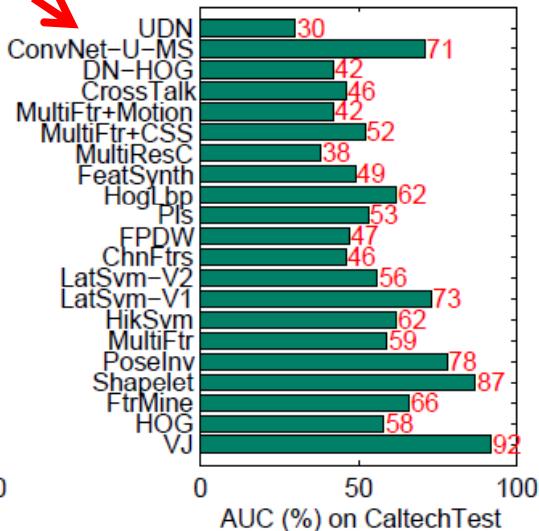
- Sermnet, K. Kavukcuoglu, S. Chintala, and LeCun, “Pedestrian Detection with Unsupervised Multi-Stage Feature Learning,” CVPR 2013.



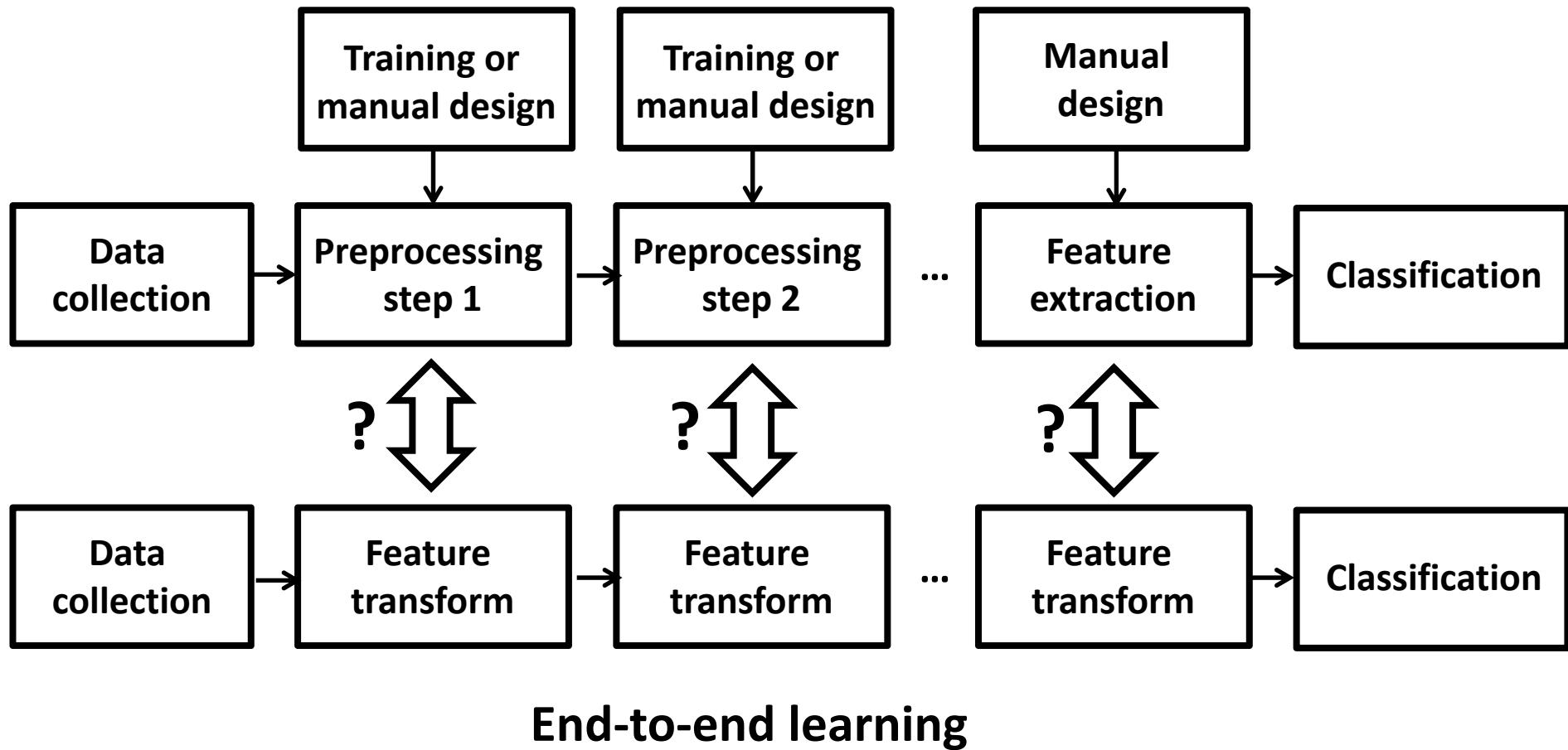
Results on Caltech Test



Results on ETHZ

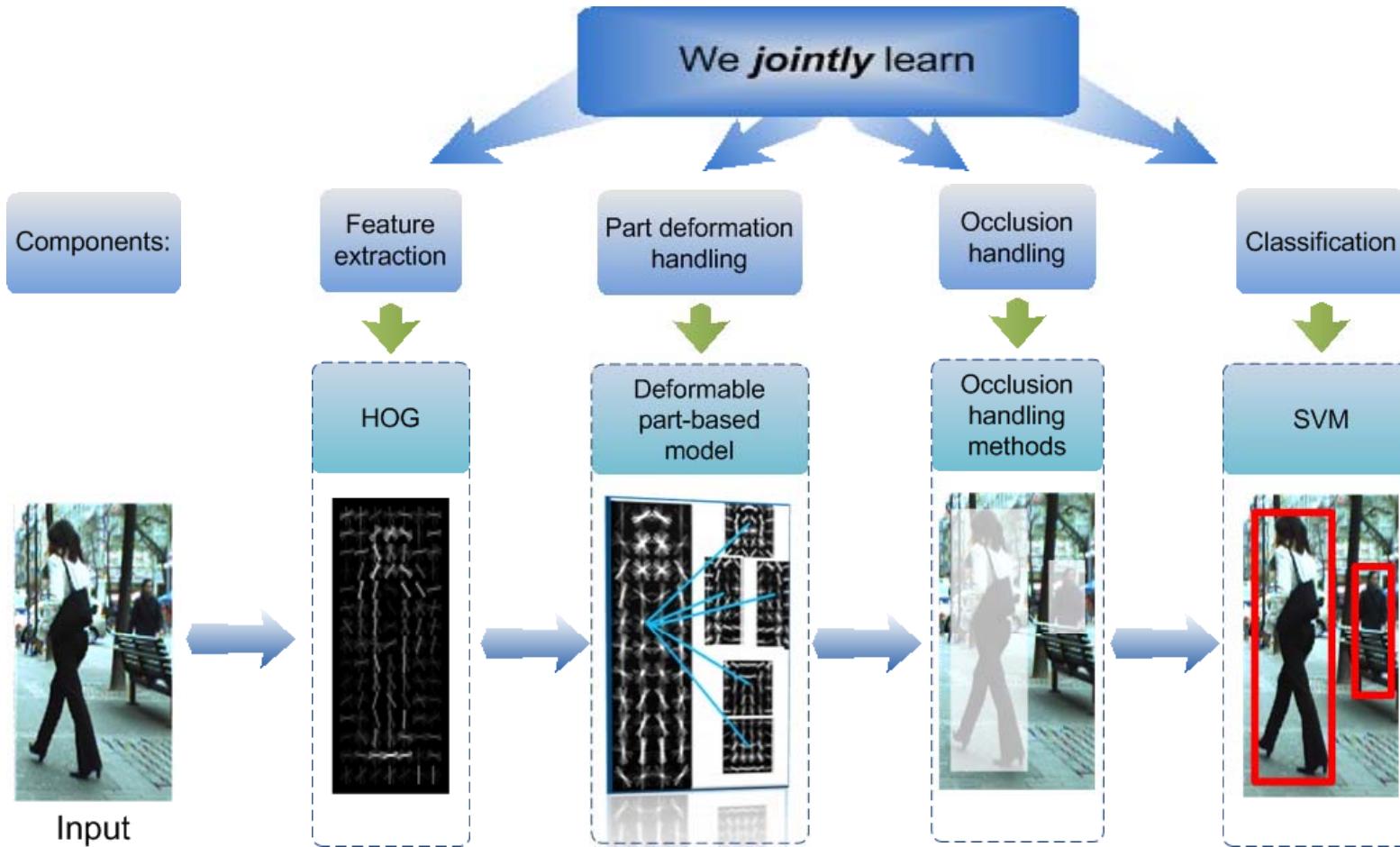


Bridge the connection between deep learning and conventional systems



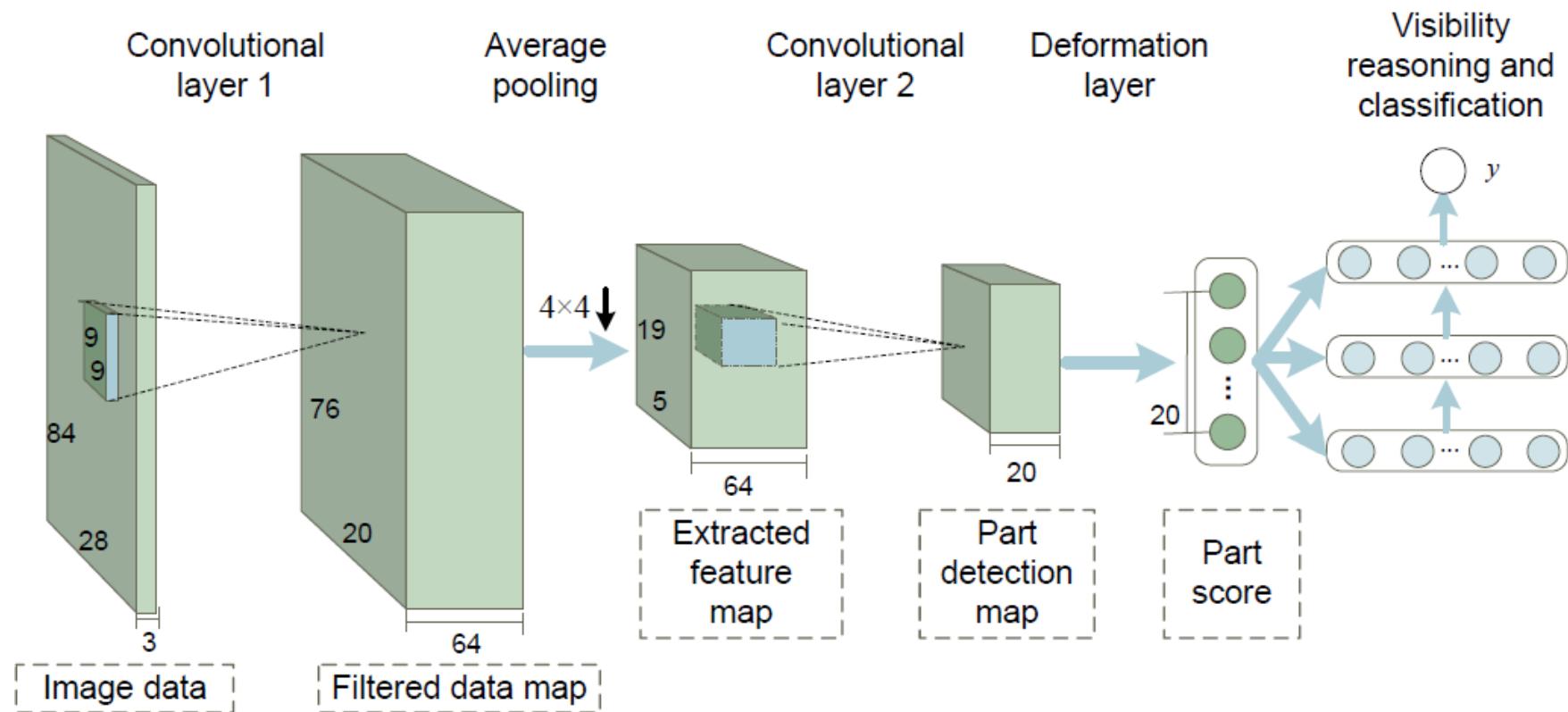
Deep learning is a framework/language but not a black-box model

Its power comes from joint optimization and
increasing the capacity of the learner



- N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. CVPR, 2005. (6000 citations)
- P. Felzenszwalb, D. McAllester, and D. Ramanan. A Discriminatively Trained, Multiscale, Deformable Part Model. CVPR, 2008. (2000 citations)
- W. Ouyang and X. Wang. A Discriminative Deep Model for Pedestrian Detection with Occlusion Handling. CVPR, 2012.

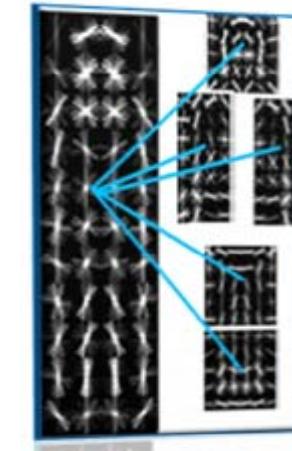
Our Joint Deep Learning Model



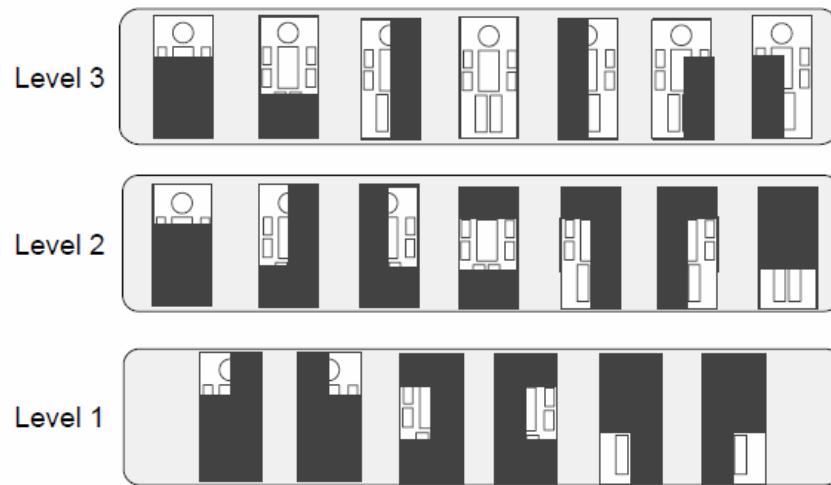
W. Ouyang and X. Wang, "Joint Deep Learning for Pedestrian Detection," Proc. ICCV, 2013.

Modeling Part Detectors

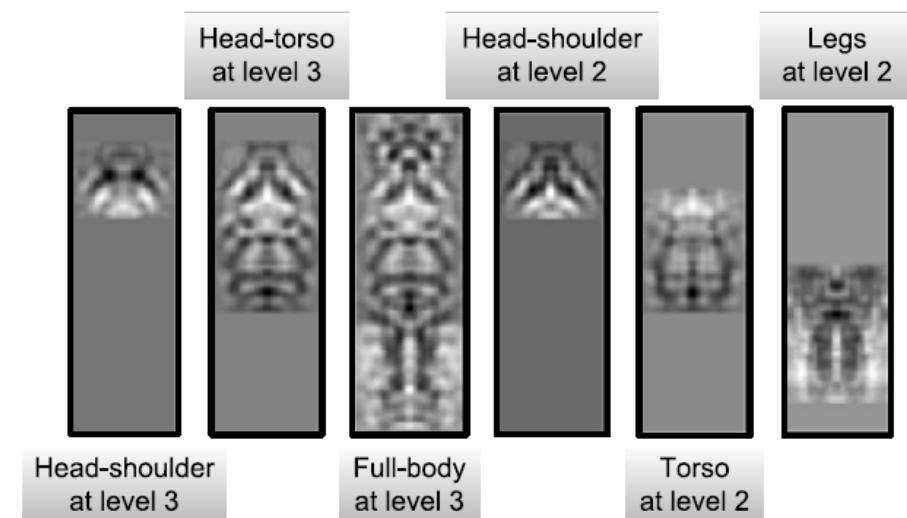
- Design the filters in the second convolutional layer with variable sizes



Part models learned
from HOG

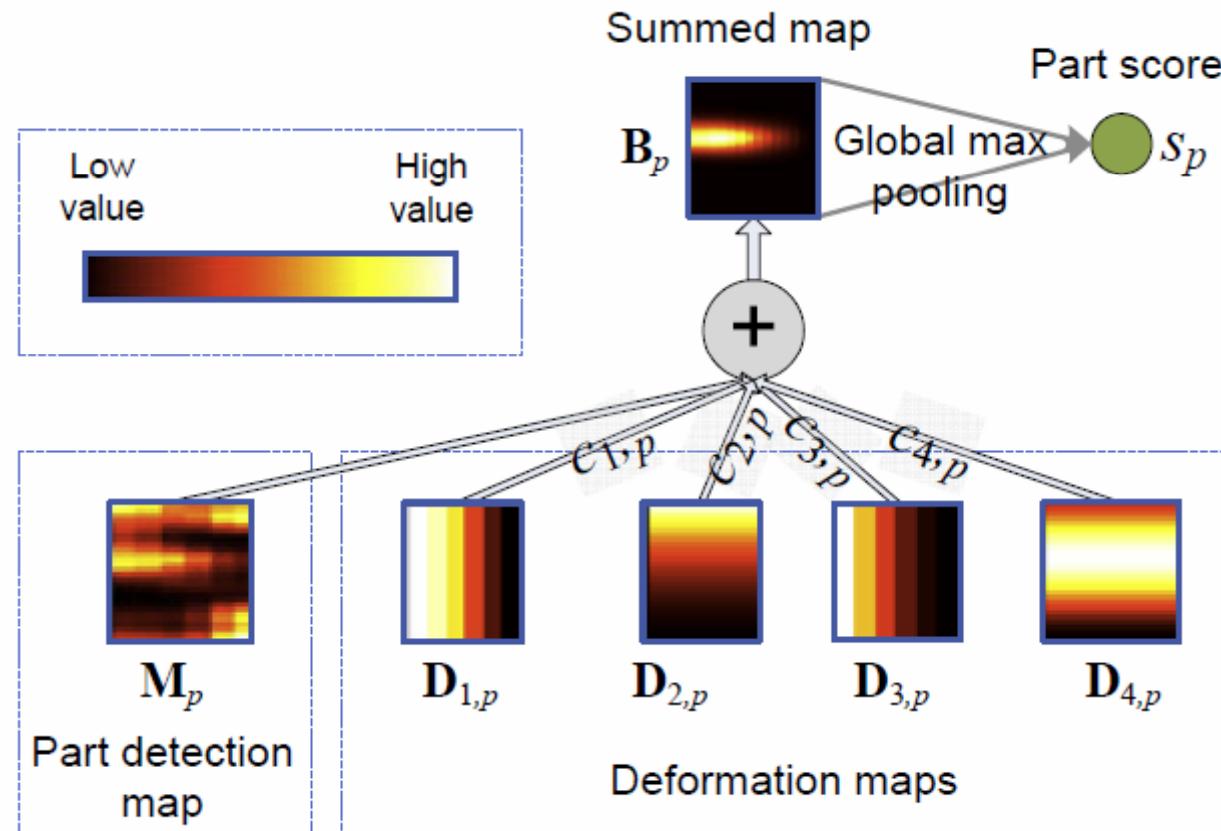


Part models

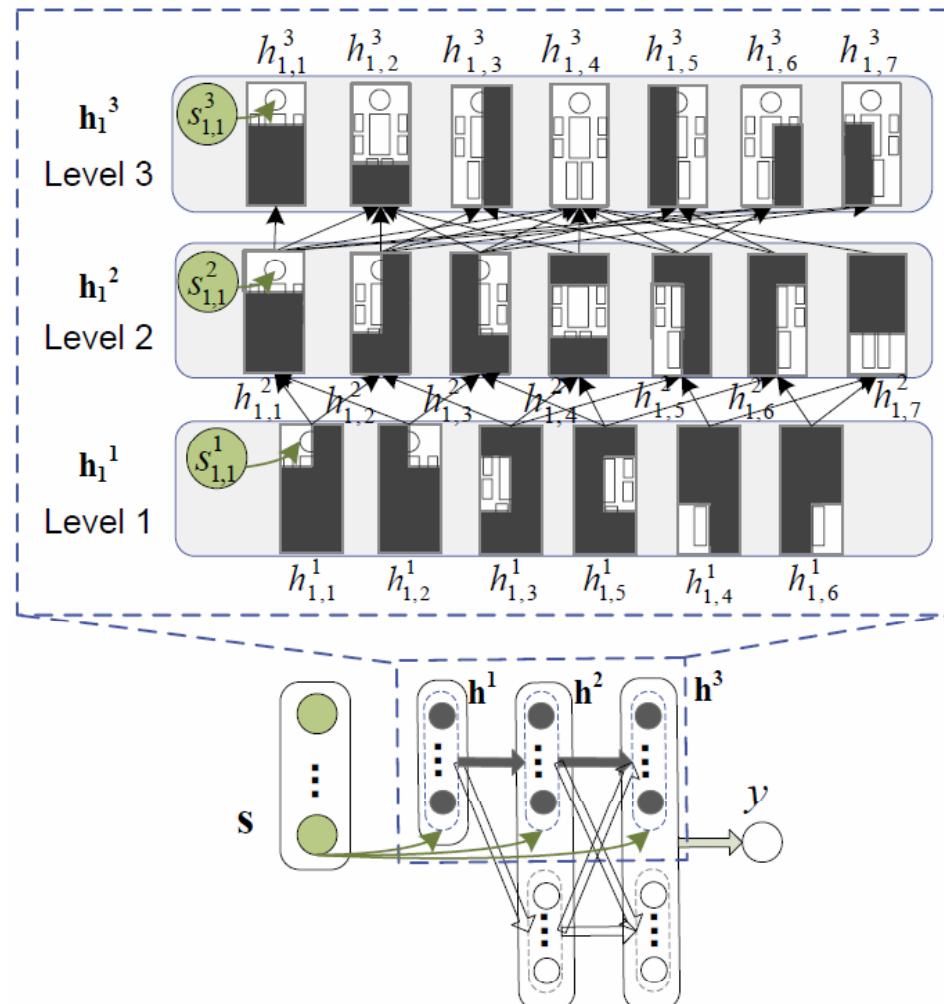


Learned filtered at the second
convolutional layer

Deformation Layer



Visibility Reasoning with Deep Belief Net

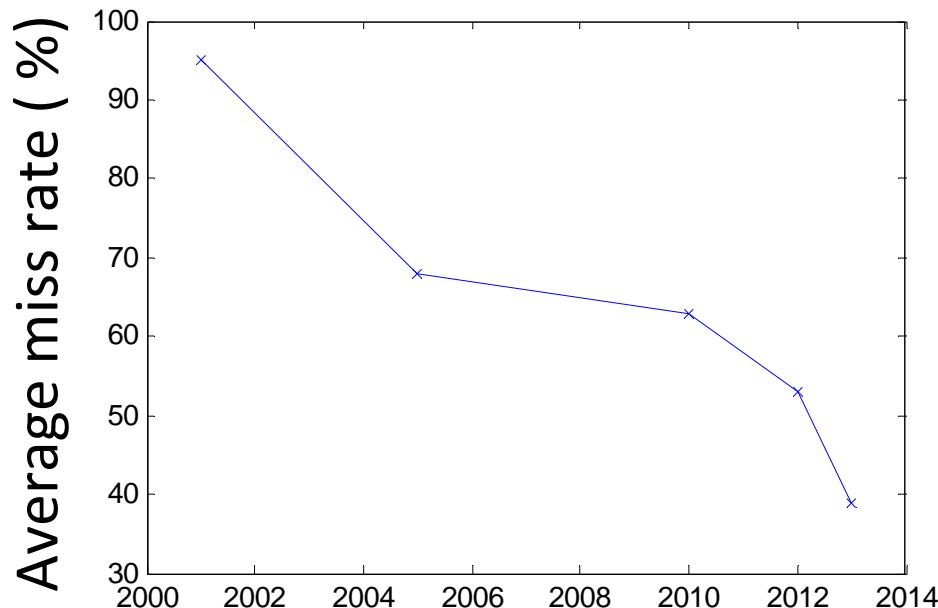


$$\tilde{h}_j^{l+1} = \sigma(\tilde{\mathbf{h}}^{l\top} \mathbf{w}_{*,j}^l + c_j^{l+1} + g_j^{l+1} s_j^{l+1})$$

— Correlates with part detection score

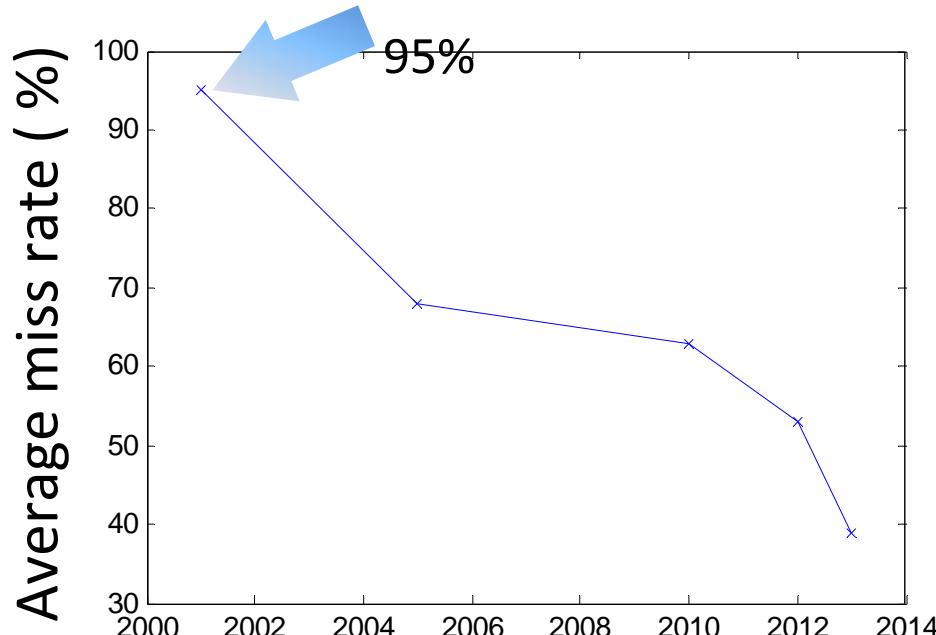
Experimental Results

- Caltech – Test dataset (largest, most widely used)



Experimental Results

- Caltech – Test dataset (largest, most widely used)



Rapid object detection using a boosted cascade of simple features

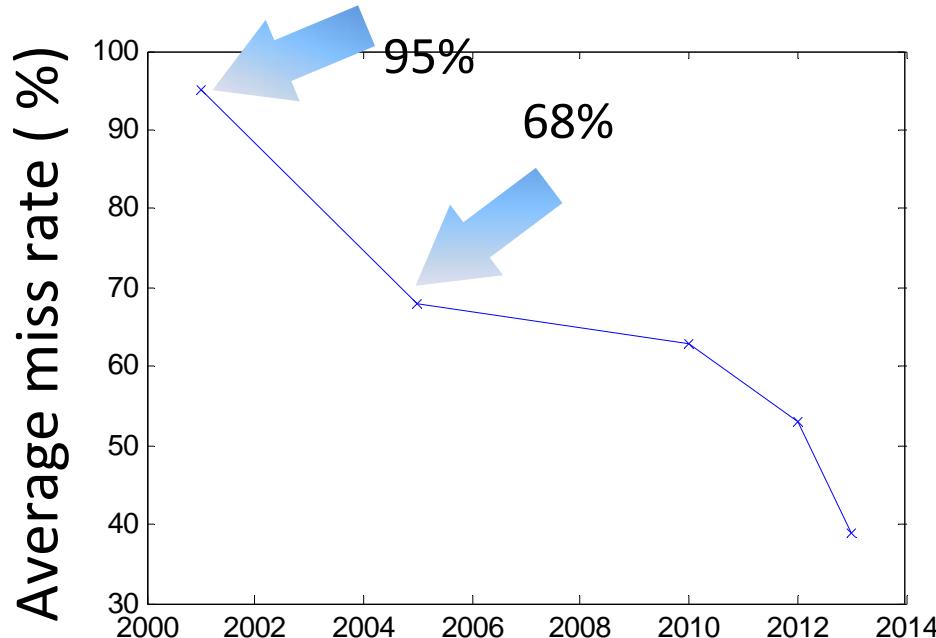
P Viola, M Jones - ... Vision and Pattern Recognition, 2001. CVPR ..., 2001 - ieeexplore.ieee.org.org

Abstract This paper describes a machine learning approach for visual **object detection** which is capable of processing images extremely rapidly and achieving high **detection** rates. This work is distinguished by three key contributions. The first is the introduction of a new ...

Cited by 7647 Related articles All 201 versions Import into BibTeX More▼

Experimental Results

- Caltech – Test dataset (largest, most widely used)



[Histograms of oriented gradients for human detection](#)

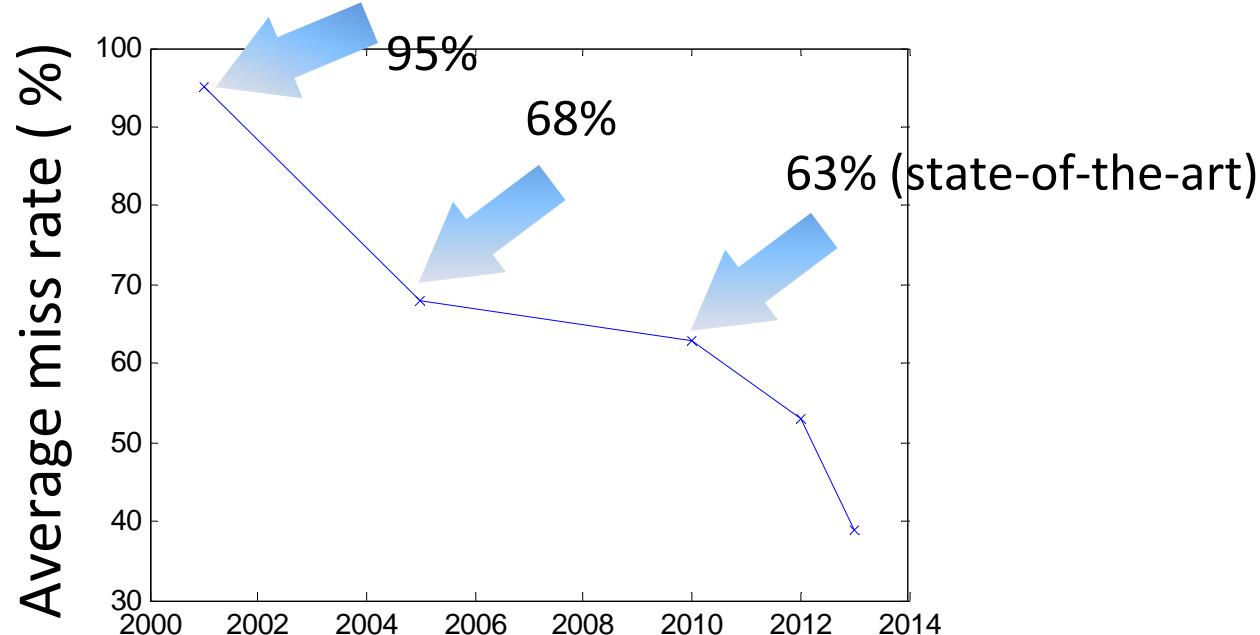
[N Dalal, B Triggs - ... and Pattern Recognition, 2005. CVPR 2005 ...](#), 2005 - ieeexplore.ieee.org

... We study the issue of feature sets for **human detection**, showing that locally normalized **Histogram of Oriented Gradient** (HOG) descriptors provide excellent performance relative to other existing feature sets including wavelets [17,22]. ...

Cited by 5438 Related articles All 106 versions Import into BibTeX More ▾

Experimental Results

- Caltech – Test dataset (largest, most widely used)



Object detection with discriminatively trained part-based models

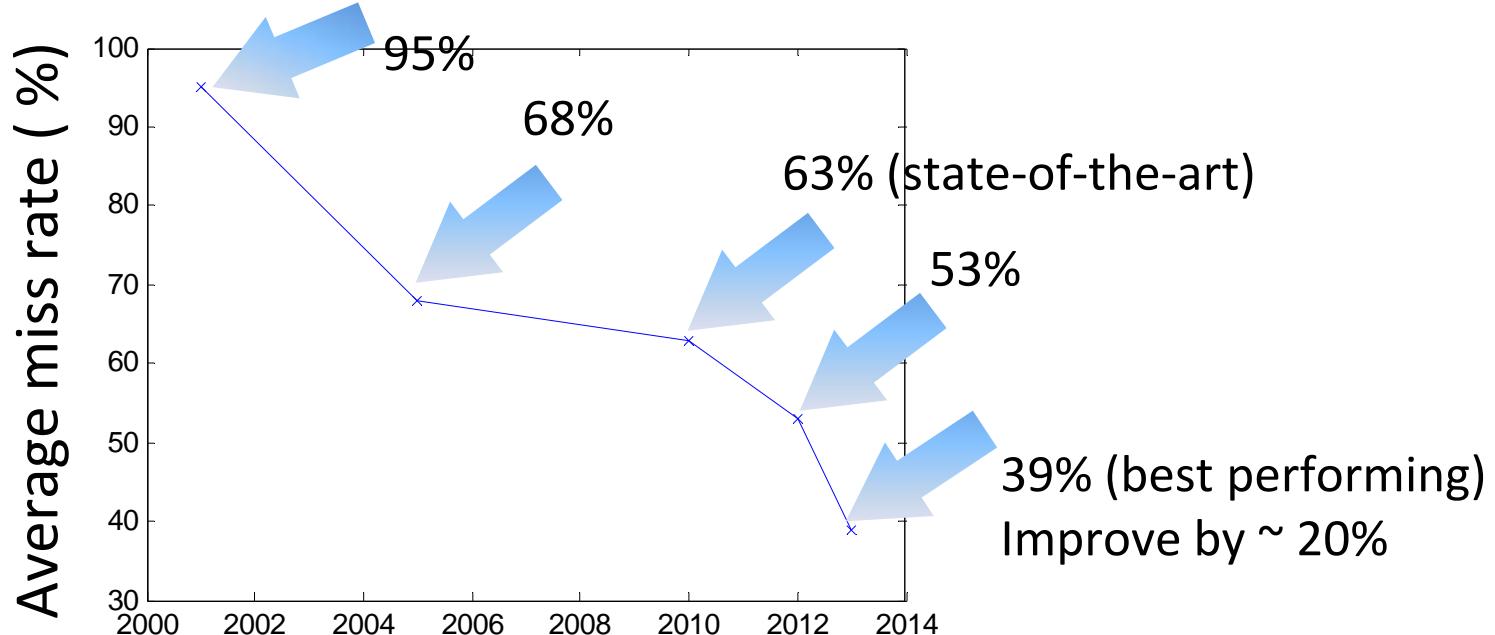
[PF Felzenszwalb, RB Girshick...](#) - Pattern Analysis and ..., 2010 - ieeexplore.ieee.org

Abstract We describe an **object detection** system **based** on mixtures of multiscale deformable **part models**. Our system is able to represent highly variable **object** classes and achieves state-of-the-art results in the PASCAL **object detection** challenges. While ...

Cited by 964 Related articles All 43 versions Import into BibTeX More ▾

Experimental Results

- Caltech – Test dataset (largest, most widely used)



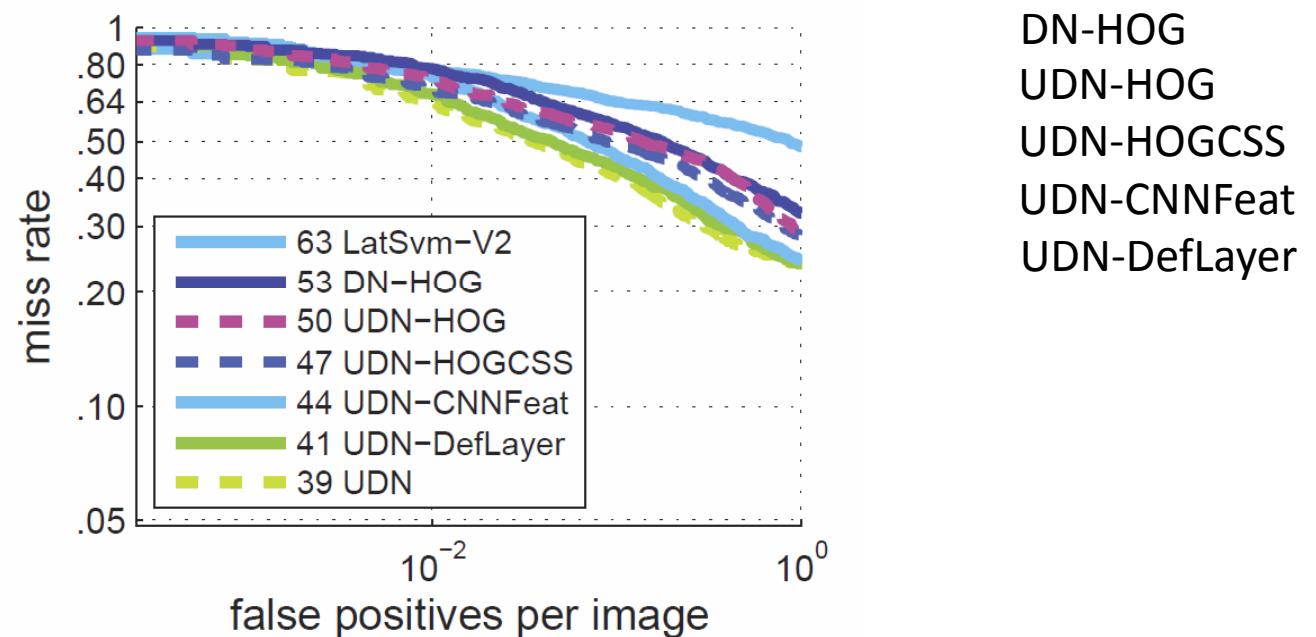
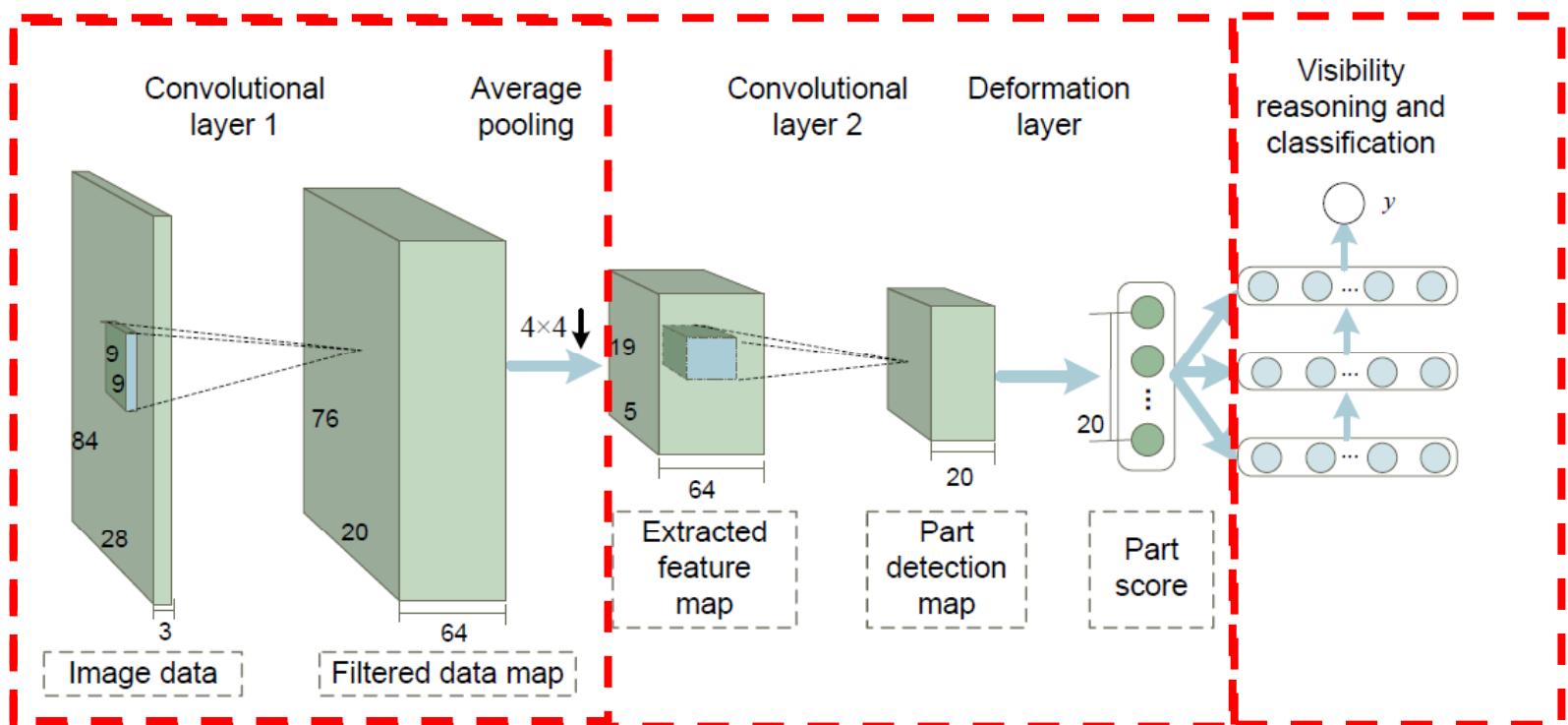
W. Ouyang and X. Wang, "A Discriminative Deep Model for Pedestrian Detection with Occlusion Handling," CVPR 2012.

W. Ouyang, X. Zeng and X. Wang, "Modeling Mutual Visibility Relationship in Pedestrian Detection ", CVPR 2013.

W. Ouyang, Xiaogang Wang, "Single-Pedestrian Detection aided by Multi-pedestrian Detection ", CVPR 2013.

X. Zeng, W. Ouyang and X. Wang, " A Cascaded Deep Learning Architecture for Pedestrian Detection," ICCV 2013.

W. Ouyang and Xiaogang Wang, "Joint Deep Learning for Pedestrian Detection," IEEE ICCV 2013.



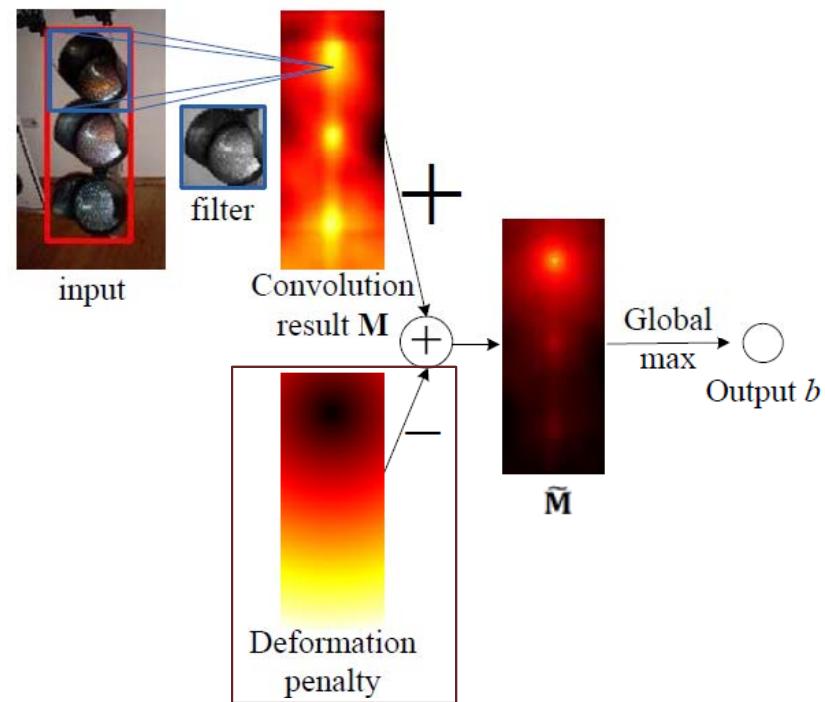
Can this idea be generalized to general object detection in ImageNet?

Deformation of parts is widely observed in general objects



Deformation Layer [b]

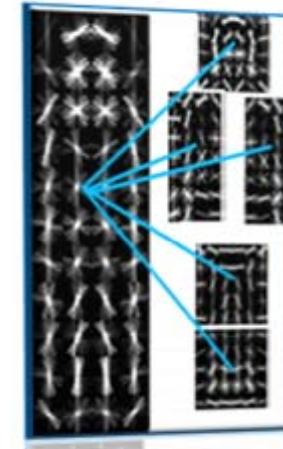
$$\mathbf{B}_p = \mathbf{M}_p + \sum_{n=1}^N c_{n,p} \mathbf{D}_{n,p}$$
$$s_p = \max_{(x,y)} b_p^{(x,y)}$$



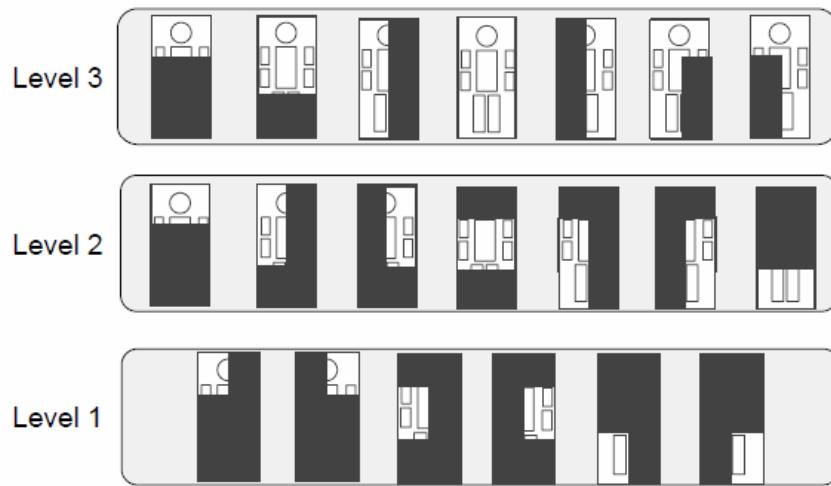
[b] Wanli Ouyang, Xiaogang Wang, "Joint Deep Learning for Pedestrian Detection ", ICCV 2013.

Modeling Part Detectors

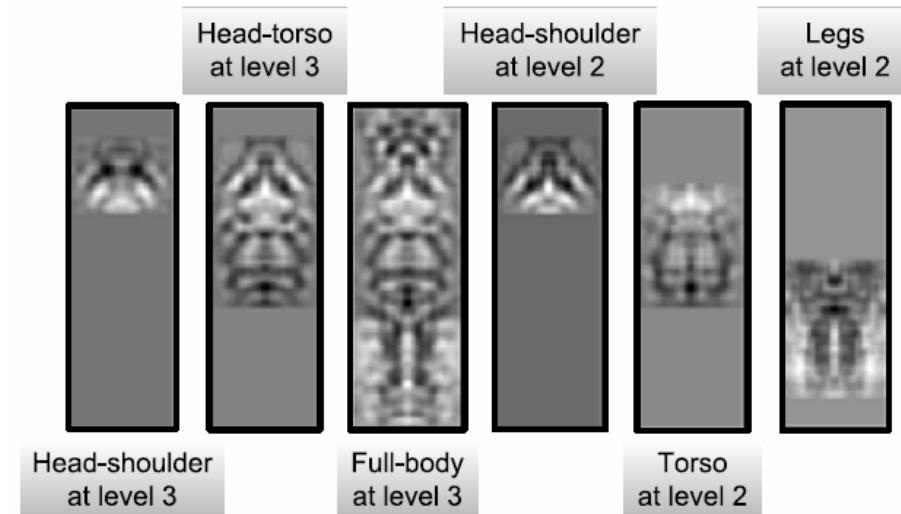
- Different parts have different sizes
- Design the filters with variable sizes



Part models learned
from HOG



Part models



Learned filtered at the second
convolutional layer

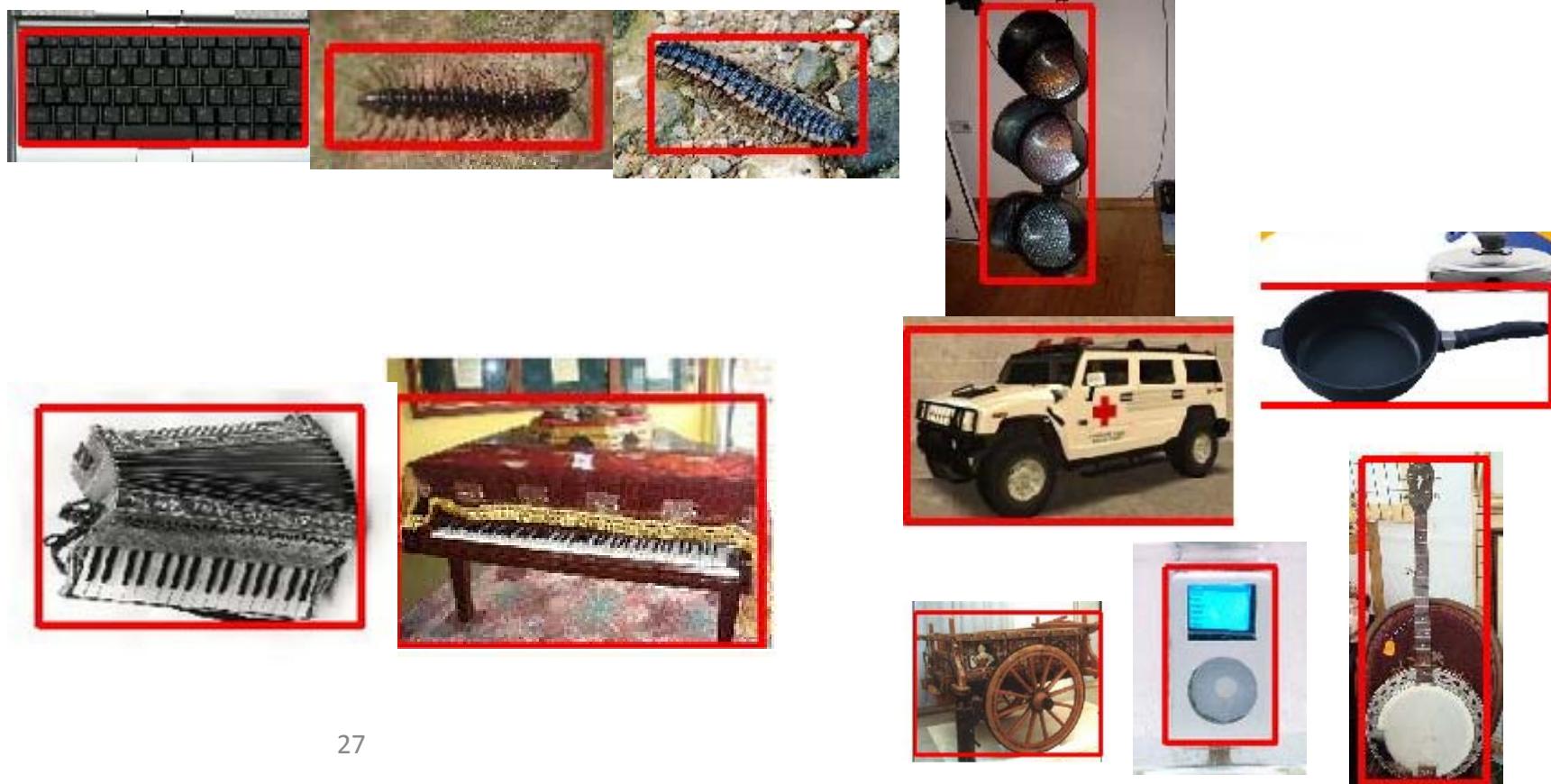
Deformation layer for repeated patterns

Pedestrian detection	General object detection
Assume no repeated pattern	Repeated patterns



Deformation layer for repeated patterns

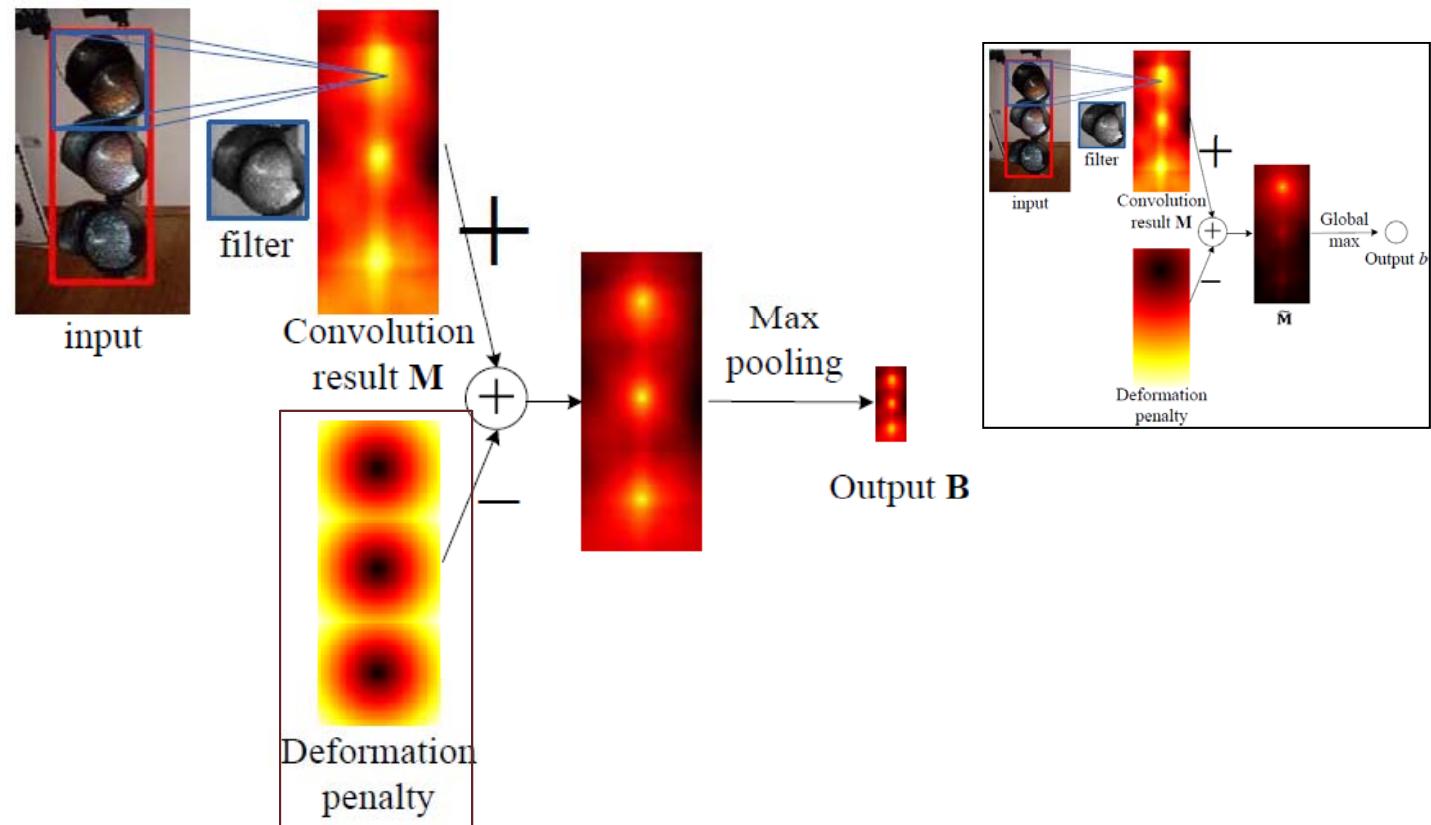
Pedestrian detection	General object detection
Assume no repeated pattern	Repeated patterns
Only consider one object class	Patterns shared across different object classes



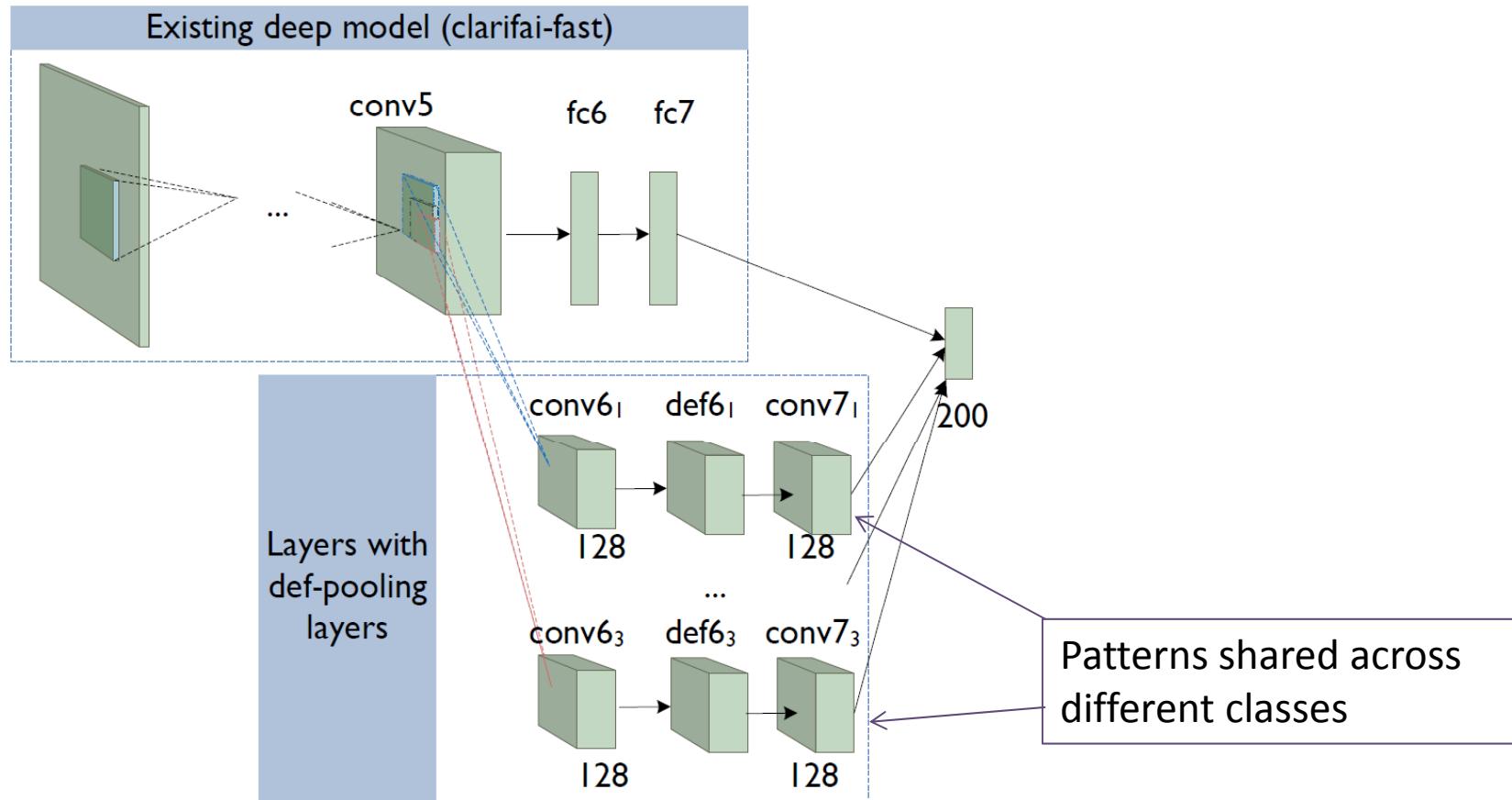
Deformation constrained pooling layer

Can capture multiple patterns simultaneously

$$b^{(x,y)} = \max_{i,j \in \{-R, \dots, R\}} \{ m^{(k_x \cdot x + i, k_y \cdot y + j)} - \sum_{n=1}^N c_n d_n^{i,j} \},$$



Our deep model with deformation layer



Training scheme	Cls+Det	Loc+Det	Loc+Det
Net structure	AlexNet	Clarifai	Clarifai+Def layer
Mean AP on val2	0.299	0.360	0.385

- ImageNet 2014 – object detection challenge

	GoogLeNet (Google)	DeepID-Net (CUHK)	DeepInsight	UvA- Euvision	Berkley Vision	RCNN
Model average	0.439	0.439	0.405	n/a	n/a	n/a
Single model	0.380	0.427	0.402	0.354	0.345	0.314

W. Ouyang et al. “DeepID-Net: deformable deep convolutional neural networks for object detection”, CVPR, 2015

Our understanding of deep learning

- Most two important operations (filtering and pooling) have been widely used in computer vision**
- Expect other domain knowledge can inspire new layers such as deformation-pooling**

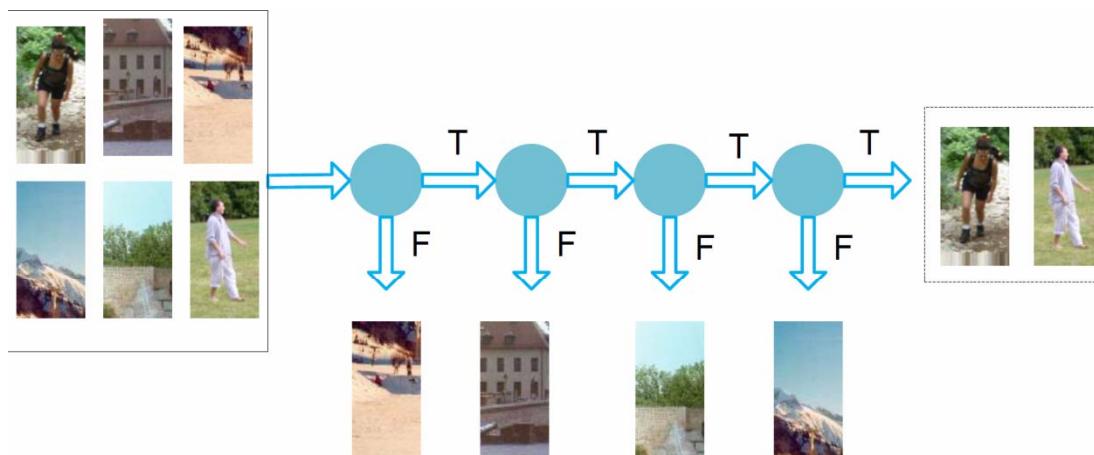
Many important ideas in object detection
can be generalized to deep learning...

Multi-Stage Contextual Deep Learning:

- ✧ Simulate cascaded detector and contextual boost
- ✧ Train different detectors for different types of samples
- ✧ Model contextual information
- ✧ Stage-by-stage pretraining strategies

Cascaded Classifiers

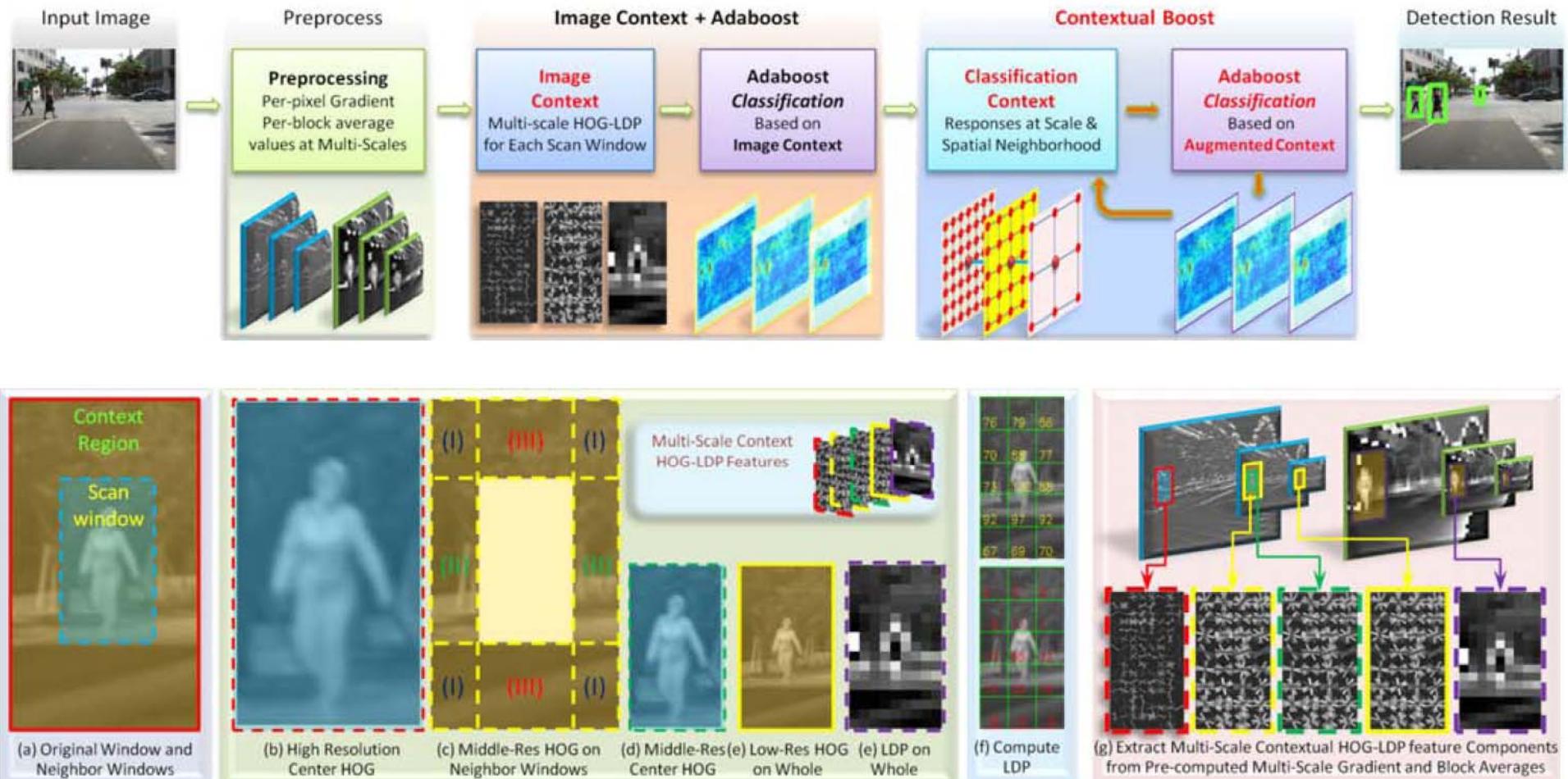
- The classifier of each stage deals with a specific set of samples
- The score map output by one classifier can serve as contextual information for the next classifier



- ❖ Only pass one detection score to the next stage
- ❖ Classifiers are trained sequentially

Conventional cascaded classifiers for detection

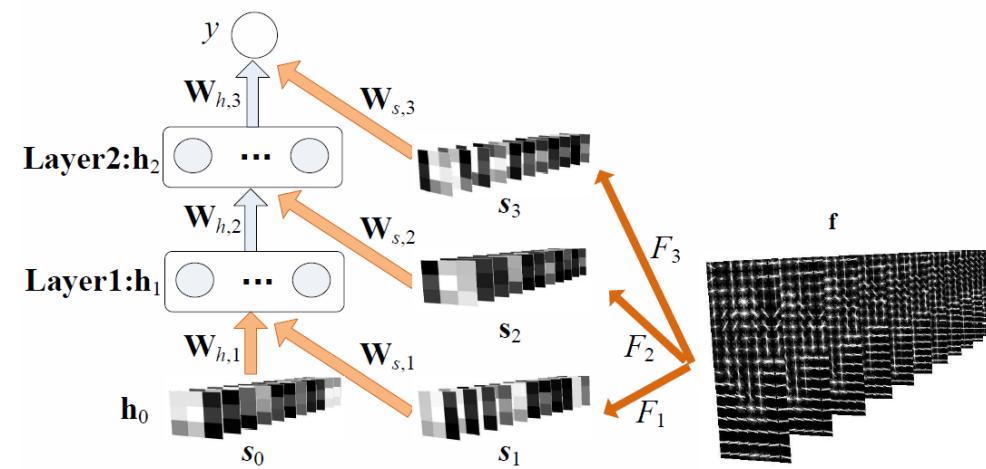
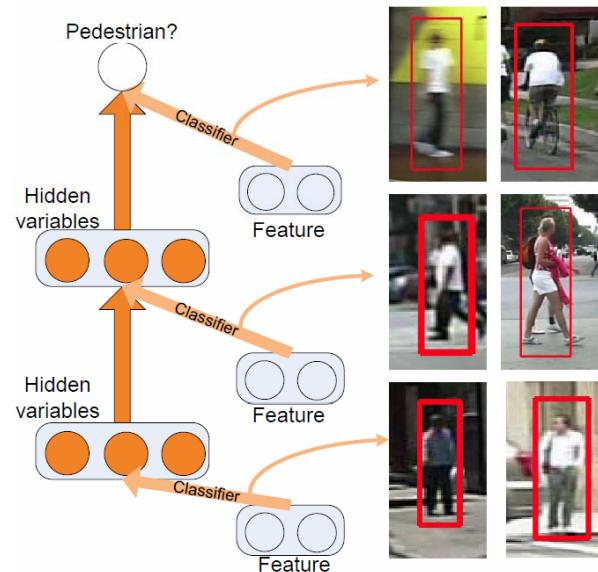
Contextual Boost



Y. Ding and J. Xiao, “Contextual Boost for Pedestrian Detection,” CVPR 2012

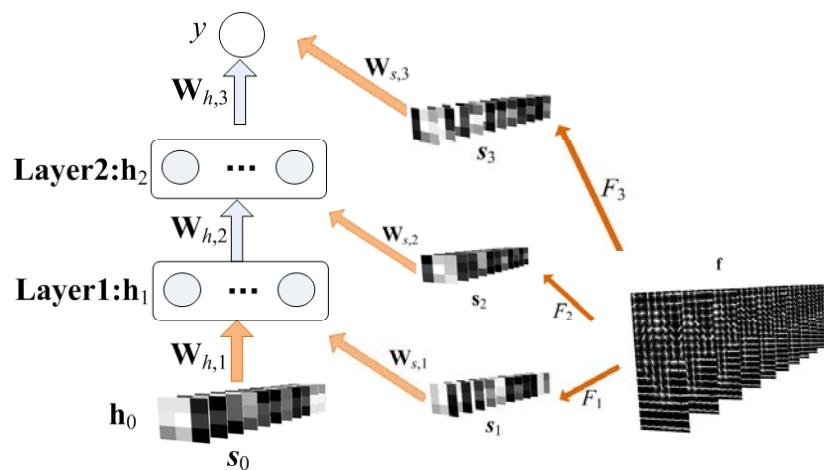
Multi-stage deep learning

- Simulate the cascaded classifiers by mining hard samples to train the network stage-by-stage
- Cascaded classifiers are jointly optimized instead of being trained sequentially
- The deep model keeps the score map output by the current classifier and it serves as contextual information to support the decision at the next stage
- To avoid overfitting, a stage-wise pre-training scheme is proposed to regularize optimization
- Multi-stage deep learning can be formulated as recurrent neural network



Training Strategies

- Unsupervised pre-train $\mathbf{W}_{h,i+1}$ layer-by-layer, setting $\mathbf{W}_{s,i+1} = 0$, $\mathbf{F}_{i+1} = 0$
- Fine-tune all the $\mathbf{W}_{h,i+1}$ with supervised BP
- Train \mathbf{F}_{i+1} and $\mathbf{W}_{s,i+1}$ with BP stage-by-stage
- A correctly classified sample at the previous stage does not influence the update of parameters
- Stage-by-stage training can be considered as adding regularization constraints to parameters, i.e. some parameters are constrained to be zeros in the early training stages



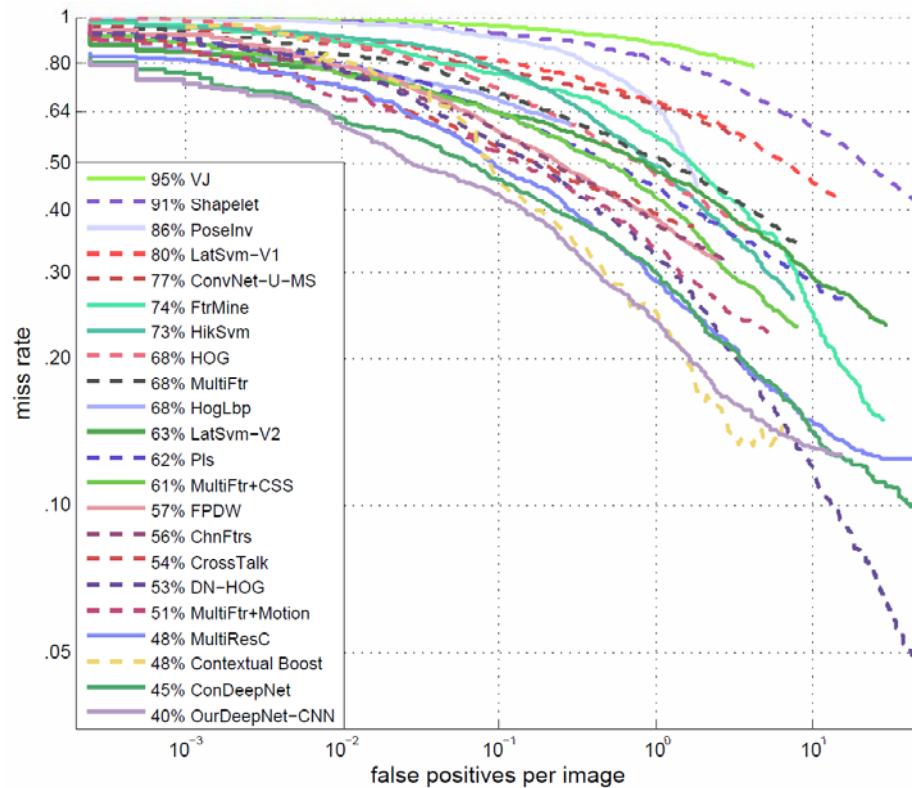
Log error function:

$$E = -l \log y - (1 - l) \log (1 - y)$$

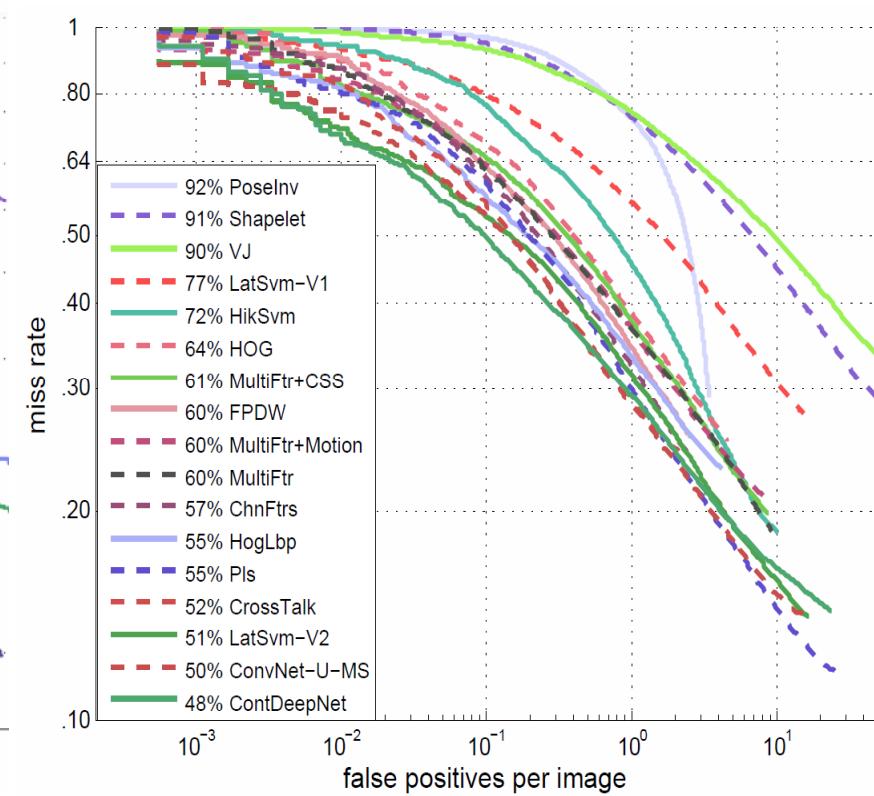
Gradients for updating parameters:

$$d\theta_{i,j} = -\frac{\partial E}{\partial \theta_{i,j}} = -\frac{\partial E}{\partial y} \frac{\partial y}{\partial \theta_{i,j}} = -(y - l) \frac{\partial y}{\partial \theta_{i,j}}$$

Experimental Results



Caltech

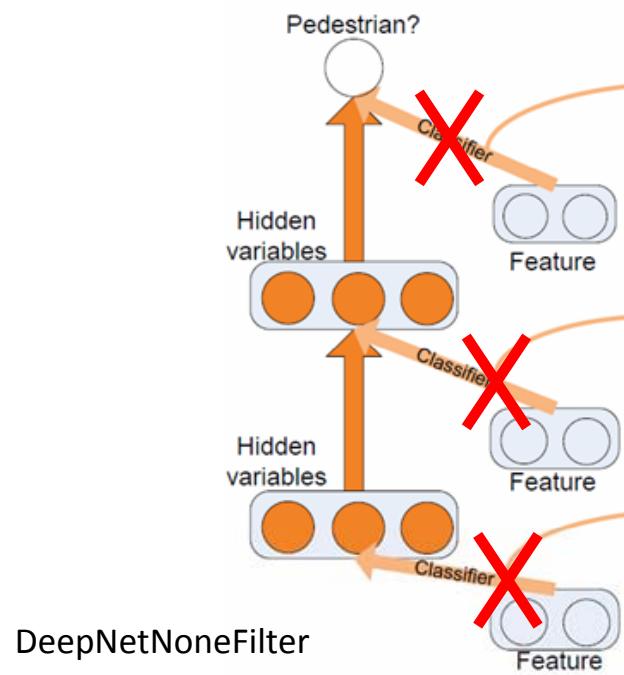
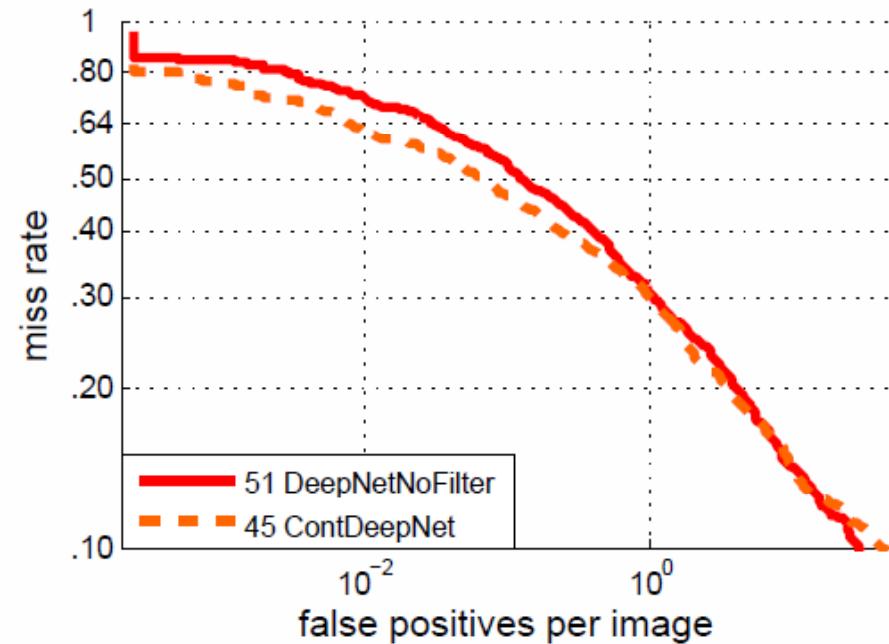


ETHZ

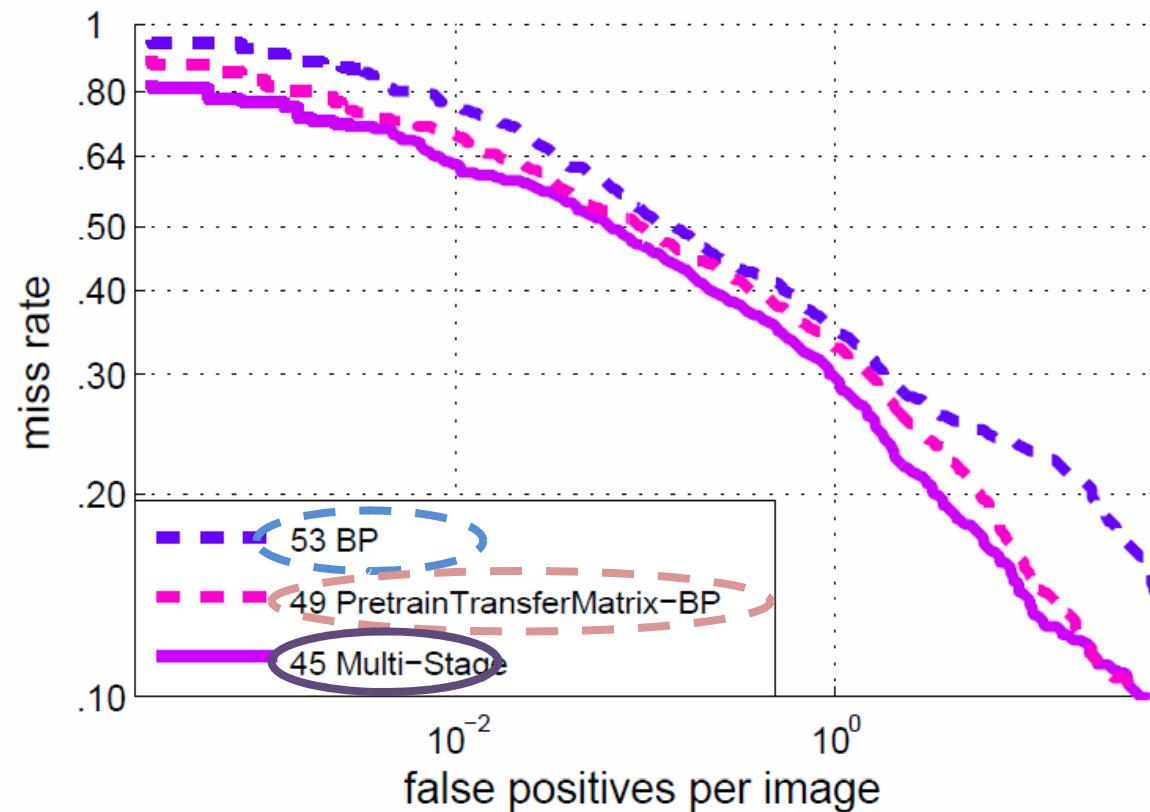
False positives of Net-NoneFilters



False negatives of Net-NoneFilters



Comparison of Different Training Strategies



Network-BP: use back propagation to update all the parameters without pre-training

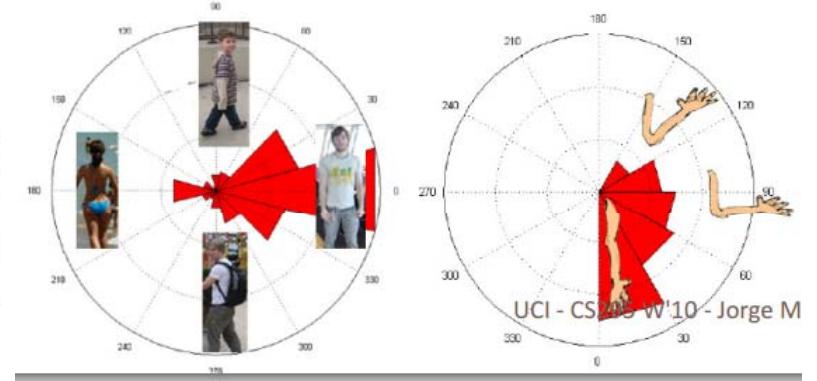
PretrainTransferMatrix-BP: the transfer matrices are unsupervised pretrained, and then all the parameters are fine-tuned

Multi-stage: our multi-stage training strategy

Switchable Deep Network

- ✧ Use mixture components to model complex variations of body parts
- ✧ Use salience maps to depress background clutters
- ✧ Help detection with segmentation information

Poselet: modeling mixture components of body parts



L. Bourdev and J. Malik, "Poselets: Body part detectors trained using 3d human pose annotations," ICCV 2009

Switchable Deep Network for Pedestrian Detection

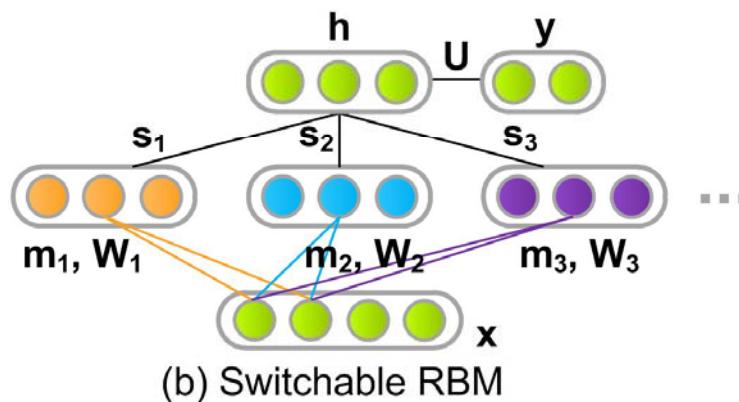
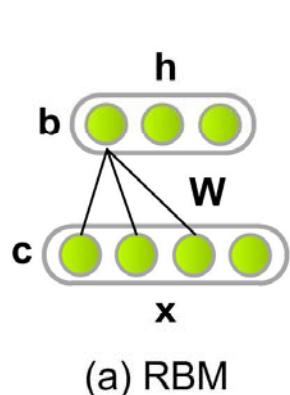


- *Background clutter* and large variations of pedestrian appearance.
- **Proposed Solution.** A Switchable Deep Network (SDN) for learning the foreground map and removing the effect background clutter.

Switchable Deep Network for Pedestrian Detection

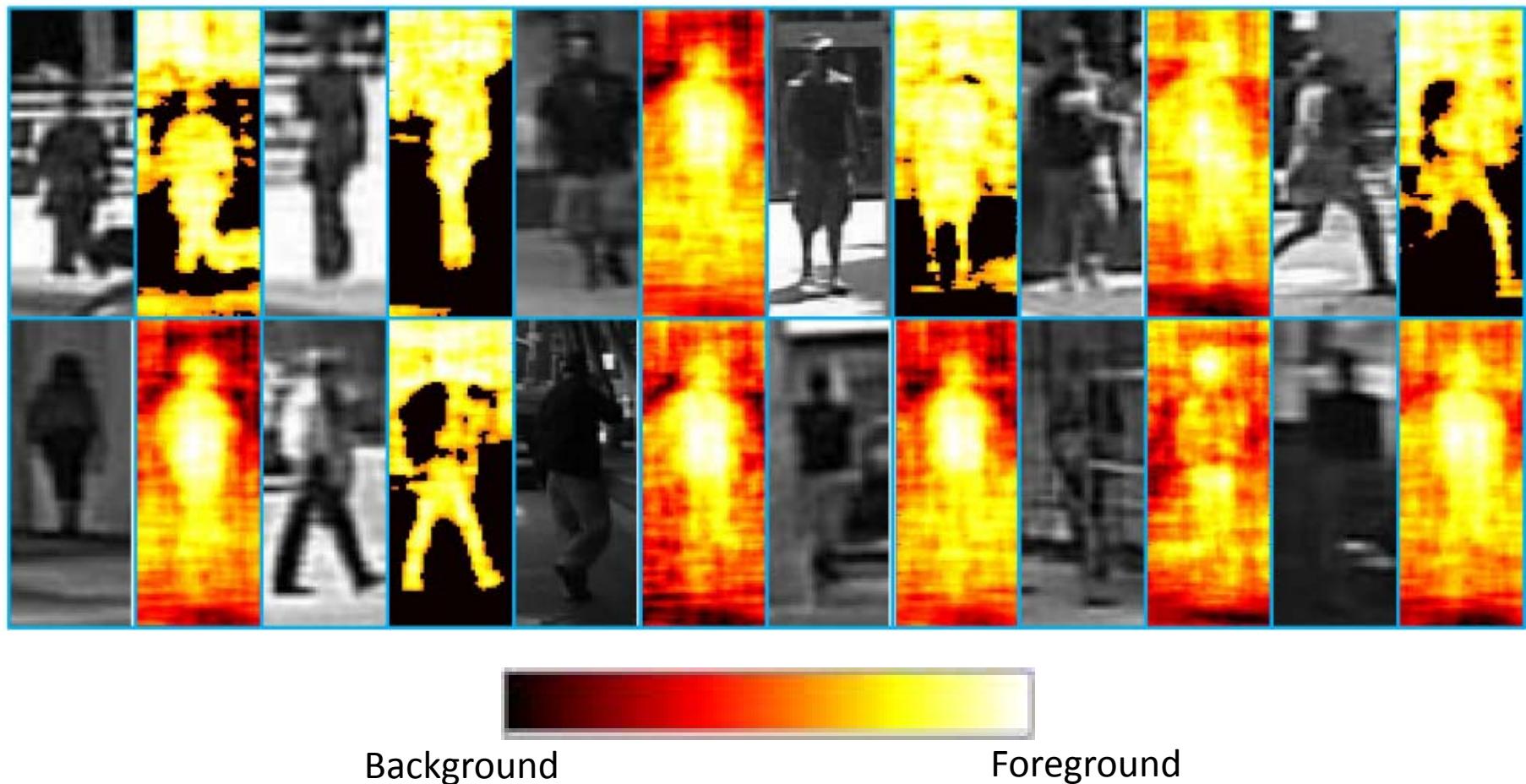
- Switchable Restricted Boltzmann Machine

$$E(\mathbf{x}, \mathbf{y}, \mathbf{h}, \mathbf{s}, \mathbf{m}; \Theta) = - \sum_{k=1}^K s_k \mathbf{h}_k^T (\mathbf{W}_k (\mathbf{x} \circ \mathbf{m}_k) + \mathbf{b}_k) - \sum_{k=1}^K s_k \mathbf{c}_k^T (\mathbf{x} \circ \mathbf{m}_k) - \mathbf{y}^T \mathbf{U} \sum_{k=1}^K s_k \mathbf{h}_k - \mathbf{d}^T \mathbf{y},$$

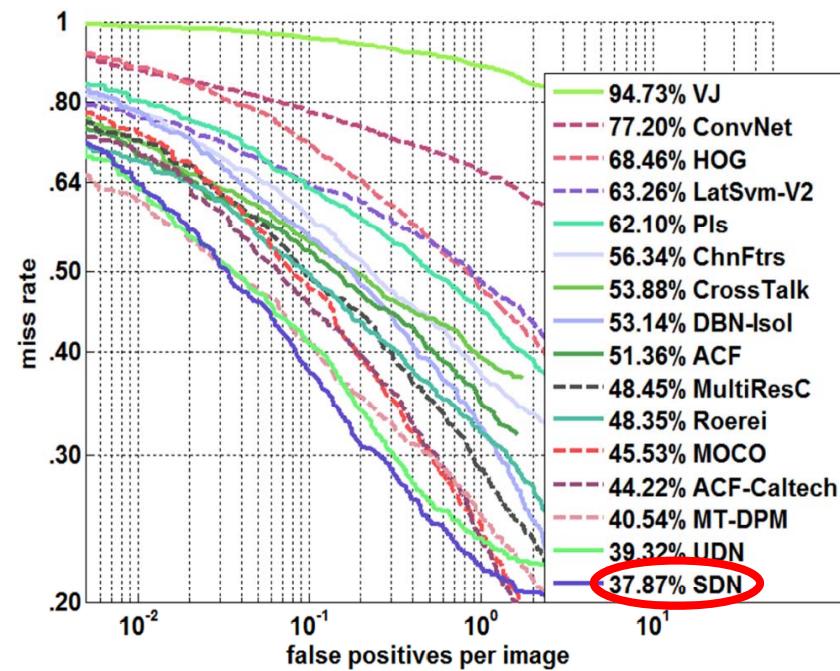


Switchable Deep Network for Pedestrian Detection

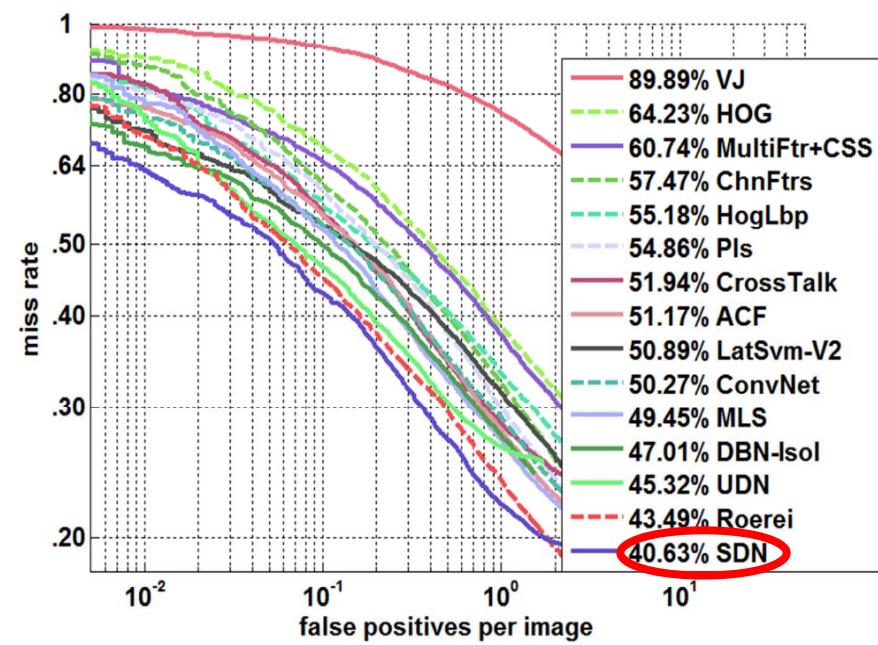
- Switchable Restricted Boltzmann Machine



Switchable Deep Network for Pedestrian Detection



(a) Performance on Caltech Test



(b) Performance on ETH

Deep Learning for Face Recognition

The projects started from December of 2012

DeepID



Yi Sun

MVP



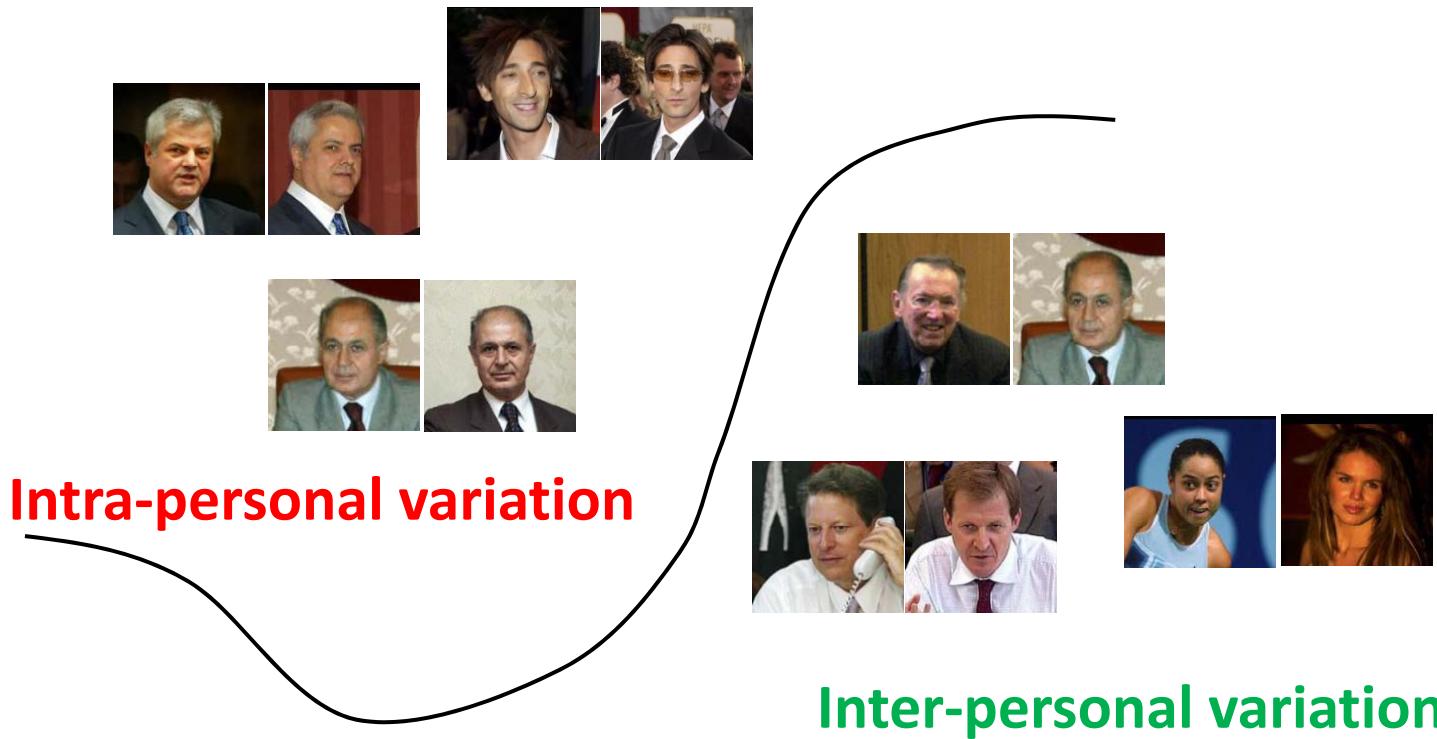
Zhenyao Zhu

Ping Luo

We started the research on face recognition since 2012

- X. Wang and X. Tang, “Unified Subspace Analysis for Face Recognition,” ICCV 2013.
- X. Wang and X. Tang, “A Unified Framework for Subspace Face Recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 26, No.9, pp. 1222-1228, 2004.

Eternal Topic on Face Recognition



How to separate the two types of variations?

Go Back to the Starting Point

- Linear discriminant analysis (LDA) (PAMI'97)
- Bayesian face recognition (PR'00)
- Unified subspace analysis (PAMI'04)

Linear Discriminate Analysis (PAMI'97)

$$\mathbf{W}^* = \arg \max_{\mathbf{W}} \frac{|\mathbf{W}' \mathbf{S}_b \mathbf{W}|}{|\mathbf{W}' \mathbf{S}_w \mathbf{W}|}$$

$$\mathbf{S}_b = \sum n_k (\bar{\mathbf{x}}_k - \bar{\mathbf{x}})(\bar{\mathbf{x}}_k - \bar{\mathbf{x}})^t \propto \sum (\bar{\mathbf{x}}_k - \bar{\mathbf{x}}_{k'})(\bar{\mathbf{x}}_k - \bar{\mathbf{x}}_{k'})^t$$

$$\mathbf{S}_w = \sum_k \sum_{i \in C_k} (\mathbf{x}_i - \bar{\mathbf{x}}_k)(\mathbf{x}_i - \bar{\mathbf{x}}_k)^t \propto \sum_{(i,j) \in \Omega} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^t$$

$$\mathbf{W}^* = \arg \max_{\mathbf{W}} |\mathbf{W}' \mathbf{S}_b \mathbf{W}| \quad s.t. \quad |\mathbf{W}' \mathbf{S}_w \mathbf{W}| = 1$$

LDA seeks for linear feature mapping which maximizes the distance between class centers under the constraint what the intrapersonal variation is constant

$$\mathbf{y}_i = f(\mathbf{x}_i) = \mathbf{W}' \mathbf{x}_i$$

$$f^* = \arg \max_f \sum_{k,k'} |f(\bar{\mathbf{x}}_k) - f(\bar{\mathbf{x}}_{k'})|^2$$

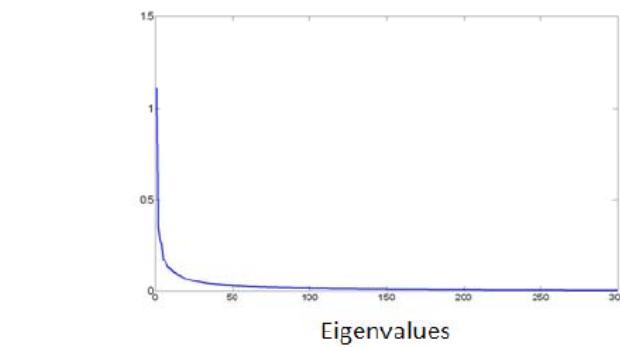
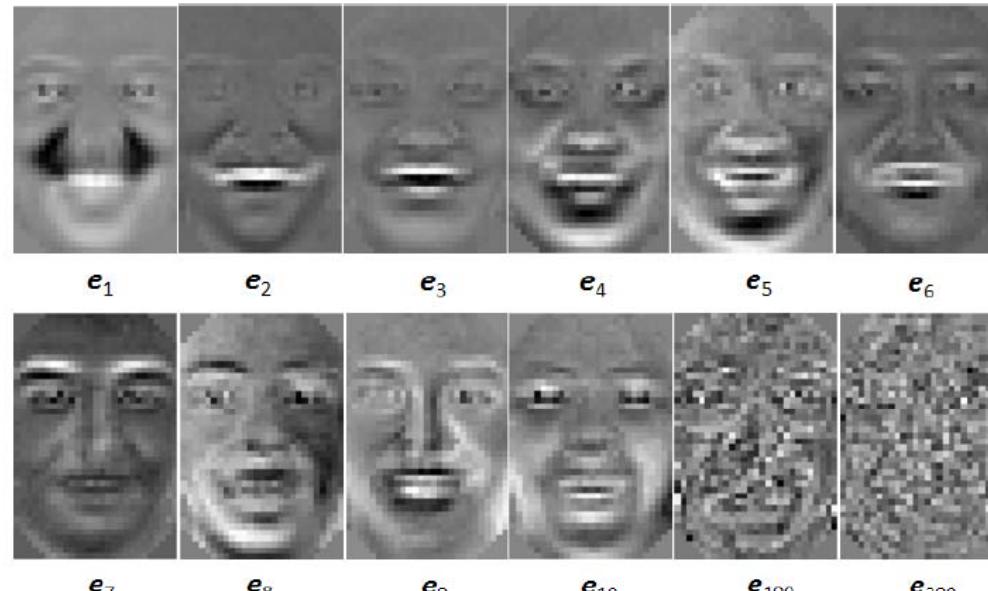
$$s.t. \sum_{(i,j) \in \Omega_f} |f(\mathbf{x}_i) - f(\mathbf{x}_j)|^2 = 1$$

Bayesian Face Recognition (PR'00)



Training images

$$\Delta = \mathbf{X}_1 - \mathbf{X}_2$$

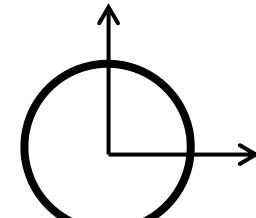
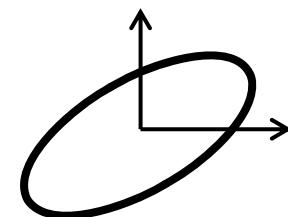


Intrapersonal subspace

$$\Delta_k = \mathbf{x}_{new} - \bar{\mathbf{x}}_k$$

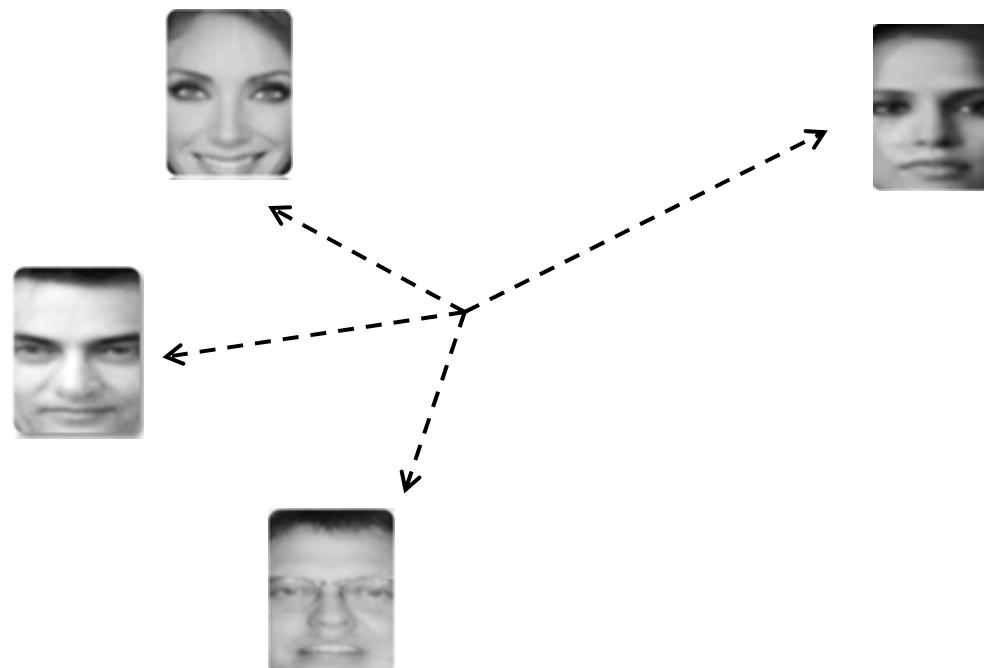
$$y_{ki} = \mathbf{e}_i^t (\mathbf{x}_{new} - \bar{\mathbf{x}}_k)$$

$$r^2(\Delta_k) = \sum_{i=1}^{d'} y_{ki}^2 / \lambda_i$$



Scatter Class Centers

- Further do PCA on class centers after reducing intrapersonal variation with whitening



Unified Subspace Analysis (PAMI'04)

- Eigenface: PCA on images to reduce dimensionality and remove noise (when later steps increase intrapersonal difference, some noise could be magnified in wrong directions)
- Bayesianface: PCA on intrapersonal difference vectors to extract the patterns of intrapersonal variations, and depress them by dividing eigenvalues
- Fisherface: PCA on class centers to make them as far as possible and extract identity information

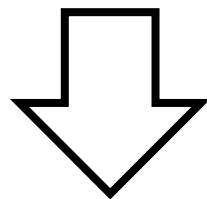
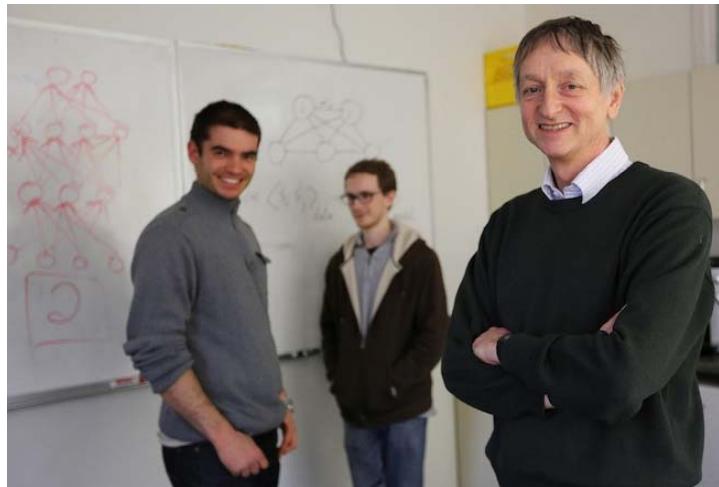
Limitations of Existing Approaches

- A lot of information has been lost when calculating the difference $\Delta = X_1 - X_2$



- Linear models with shallow structures cannot separate intra- and inter-personal variations, which are complex, nonlinear, and in high-dimensional image space

Deep Learning Won ImageNet Image Classification Challenge 2012



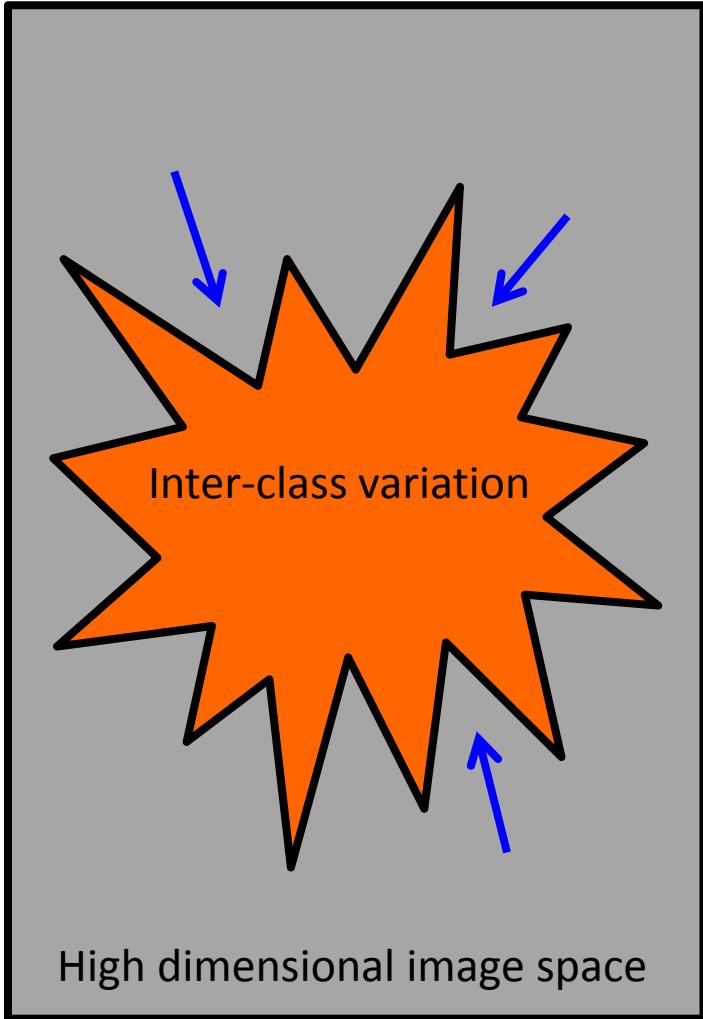
Motivated us to feed an image pair (I_1, I_2) to CNN and train a powerful nonlinear classifier

$$S(I_1, I_2) = \frac{P(\Delta|\Omega_I)P(\Omega_I)}{P(\Delta|\Omega_I)P(\Omega_I) + P(\Delta|\Omega_E)P(\Omega_E)}$$

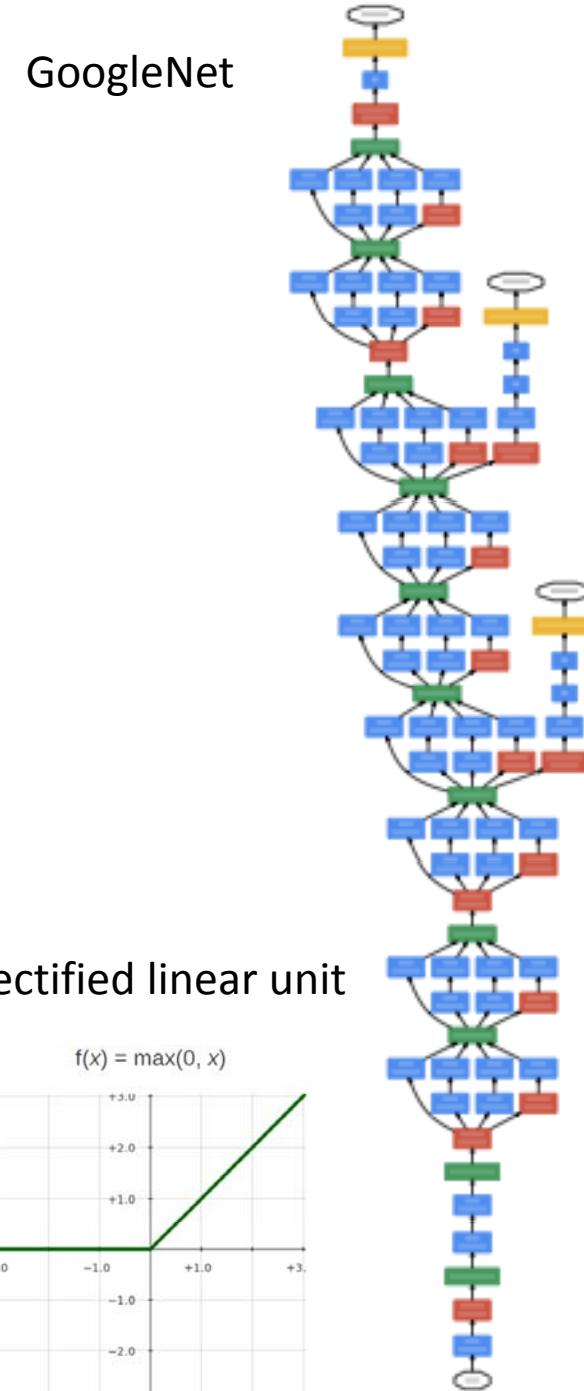
$\xrightarrow{\text{?}}$ **CNN (I_1, I_2)**

Deep Learning for Face Recognition

- Extract identity preserving features through hierarchical nonlinear mappings
- Model complex intra- and inter-personal variations with large learning capacity



- **Linear transform**
- **Pooling**
- **Nonlinear mapping**



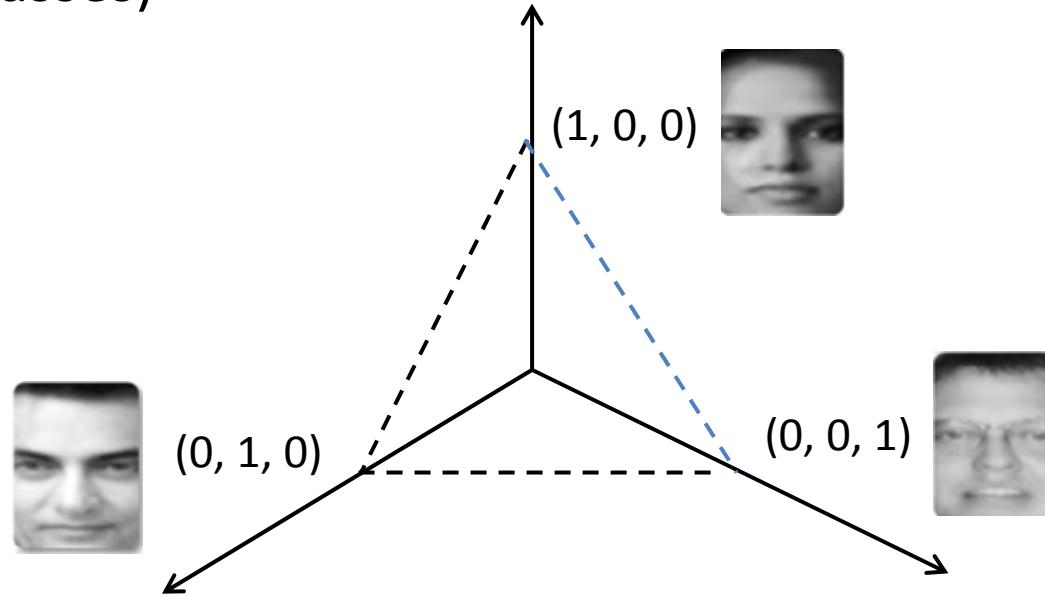
Learn Identity Features from Different Supervisory Tasks

- Face identification: classify an image into one of N identity classes
 - multi-class classification problem
- Face verification: verify whether a pair of images belong to the same identity or not
 - binary classification problem

$$S(I_1, I_2) = \frac{P(\Delta|\Omega_I)P(\Omega_I)}{P(\Delta|\Omega_I)P(\Omega_I) + P(\Delta|\Omega_E)P(\Omega_E)}$$

 **CNN (I_1, I_2)**

Minimize the intra-personal variation under the constraint that the distance between classes is constant (i.e. contracting the volume of the image space without reducing the distance between classes)



$$\mathbf{y} = f(\mathbf{x}); \quad g = \text{softmax}()$$

$$f^* = \arg \min_f \sum_{(i,j) \in \Omega_I} ||f(\mathbf{x}_i) - f(\mathbf{x}_j)||^2$$

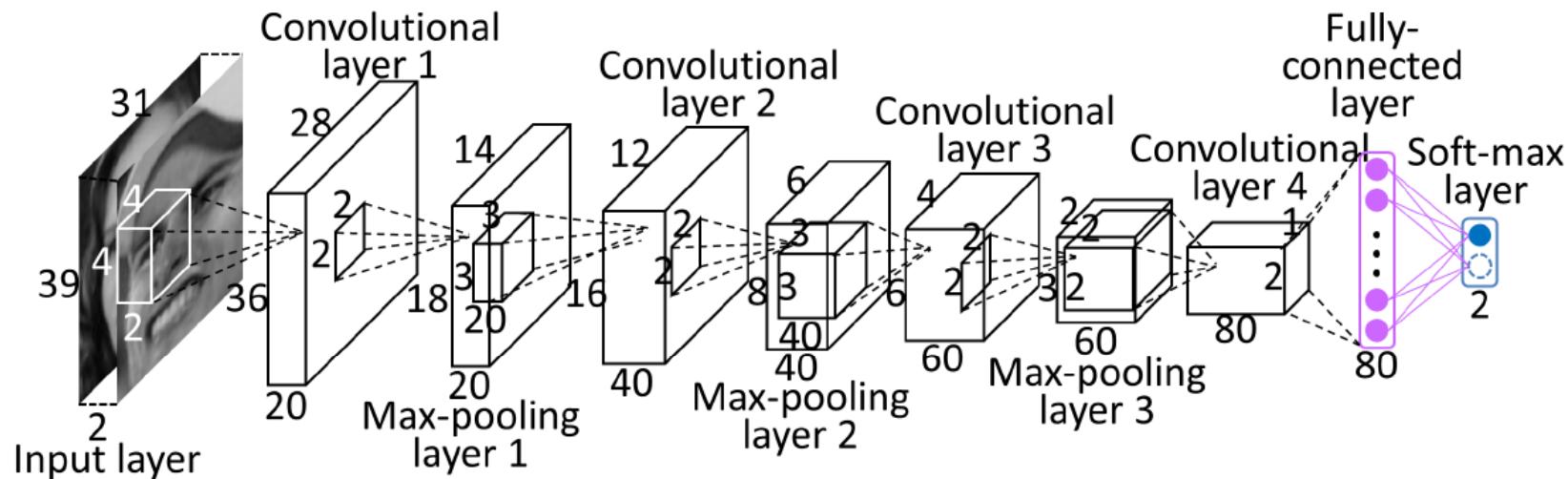
$$\text{s.t. } |g(f(\mathbf{x}_i)) - g(f(\mathbf{x}_j))| = 1, \quad \text{label}(\mathbf{x}_i) \neq \text{label}(\mathbf{x}_j)$$

Learn Identity Features with Verification Signal

- Extract relational features with learned filter pairs

$$y^j = f(b^j + k^{1j} * x^1 + k^{2j} * x^2)$$

- These relational features are further processed through multiple layers to extract global features
- The fully connected layer can be used as features to combine with multiple ConvNets



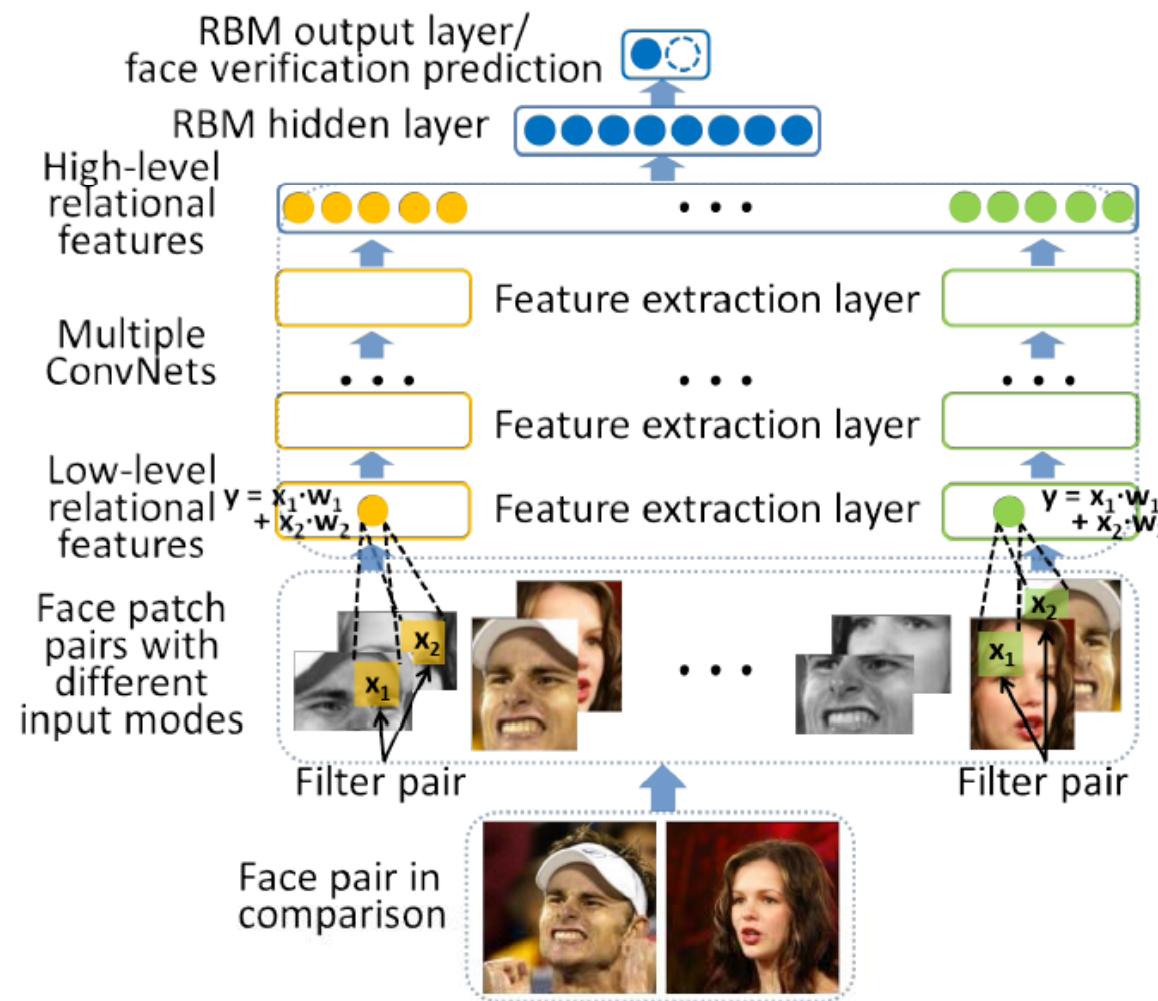
Generate Multiple CNNs

- 10 face regions, 3 scales, color/gray and 8 modes
- Base on three-point alignment



Regions and scales

RBM Combines Features Extracted by Multiple ConvNets



Results on LFW

- Outside training data: the CelebFaces dataset has 87,628 face images of 5,436 celebrities. Its identities have no overlap with LFW

	hid	hid+out	out
dimension	38,400	38,880	480
each dim (%)	60.25	60.58	86.63
PCA+LDA (%)	94.55	94.42	93.41
SVM linear (%)	95.12	95.04	93.45
SVM rbf (%)	94.95	94.89	94.00
classRBM (%)	95.56	95.32	93.79

Taking the last hidden layer (hid) as features for combination is more effective than using the output of CNNs (out)

Results on LFW

- Fine tuning RBM and ConvNets improves the performance
- Averaging 5 RBMs (each is trained with a randomly generated training set) can improves performance

	LFW (%)	CelebFaces (%)
Single ConvNet	85.05	88.46
RBM	93.45	95.56
Fine-tuning	93.58	96.60
Model averaging	93.83	97.08

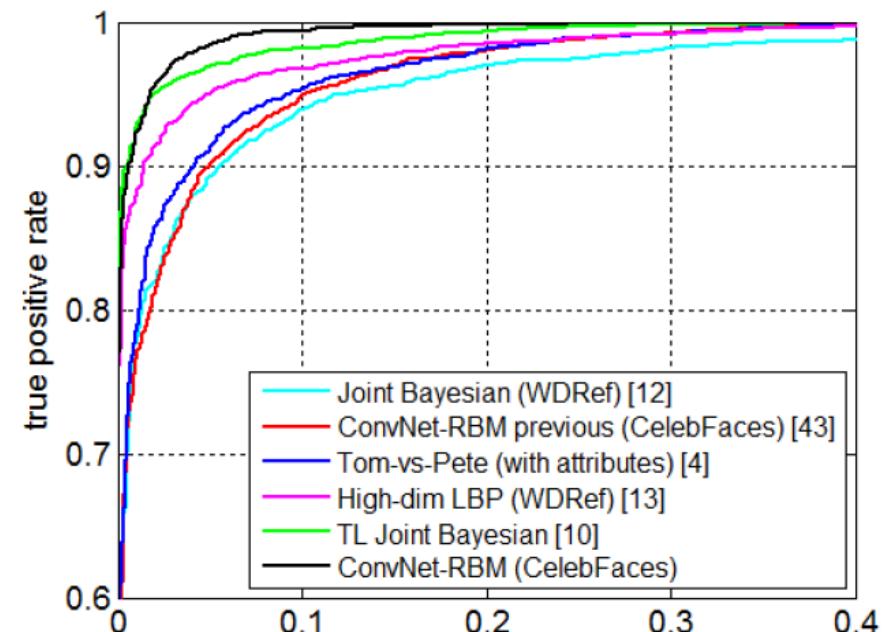
LFW: only using training images from LFW with unrestricted protocol

CelebFaces: using CelebFaces as training set without training images from LFW

Results on LFW

- Unrestricted protocol using outside training data

Method	Accuracy (%)
Joint Bayesian [12]	92.42 ± 1.08
ConvNet-RBM previous [43]	92.52 ± 0.38
Tom-vs-Pete (with attributes) [4]	93.30 ± 1.28
High-dim LBP [13]	95.17 ± 1.13
TL Joint Bayesian [10]	96.33 ± 1.08
ConvNet-RBM	97.08 ± 0.28

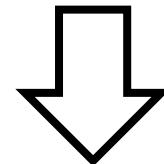


Summary of Results

- Use the last hidden layer instead of the output of CNNs as features
- Fusion of features from more face regions (CNNs) improves the performance
- Fine tuning RBM and CNNs improves performance
- Averaging the outputs of multiple RBMs improves the performance
- Drawbacks: computational cost is high and features cannot be computed offline

Features learned from a large number of classes
from ImageNet has good generalization capability

The key of deep learning is to learn feature
representations instead of classifiers

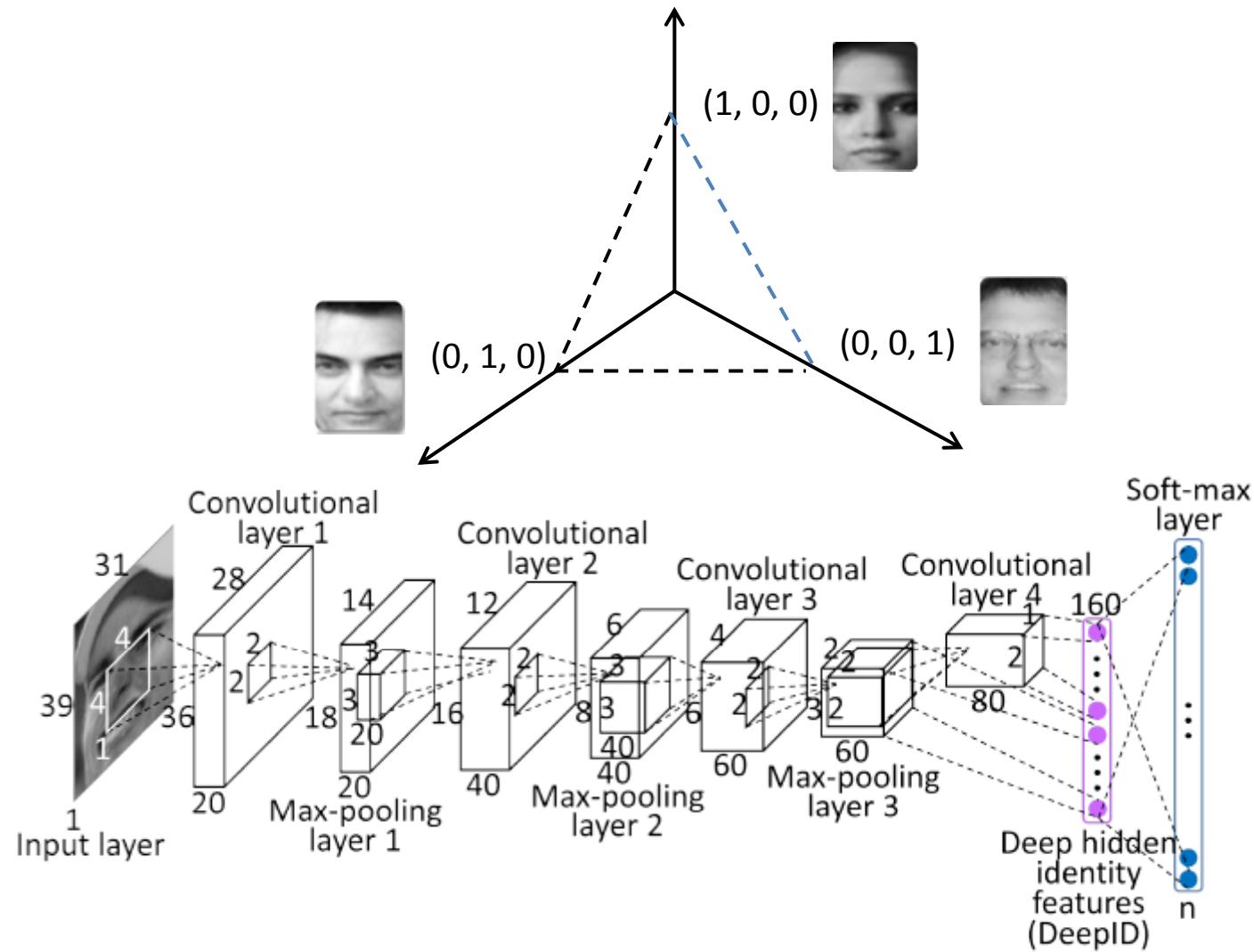


Can this idea be generalized to face recognition?

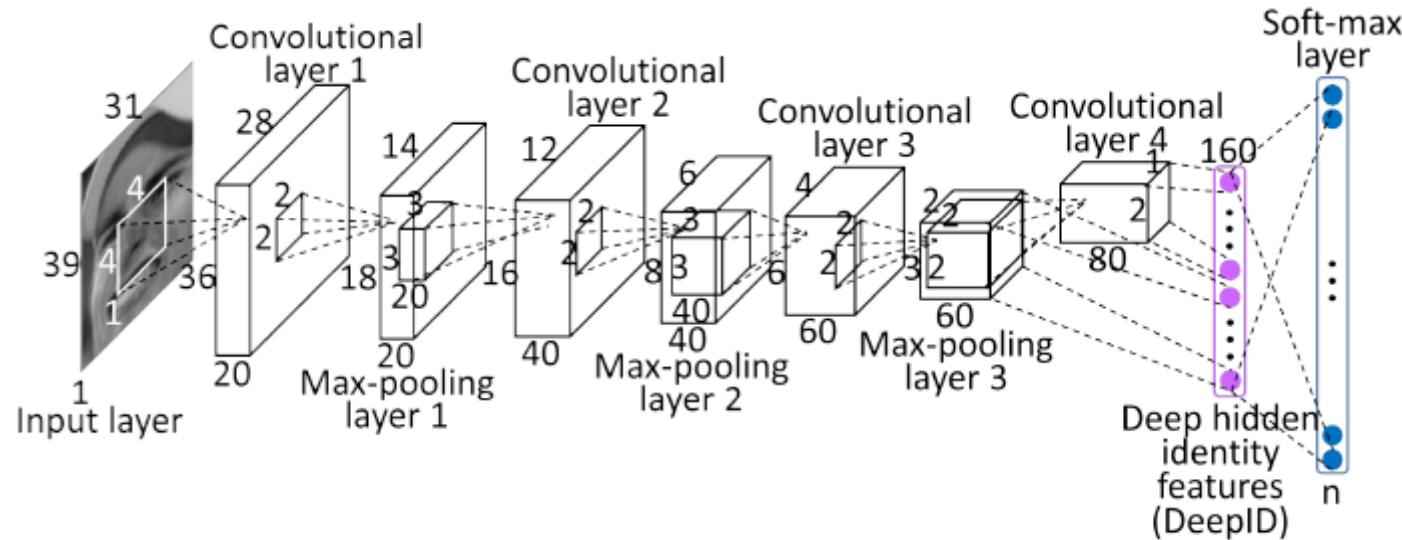
Our understanding of deep learning

- Deeply learned features can be well generalized to other datasets and recognition tasks**
- The generalization power increases when the supervision task is more challenging**

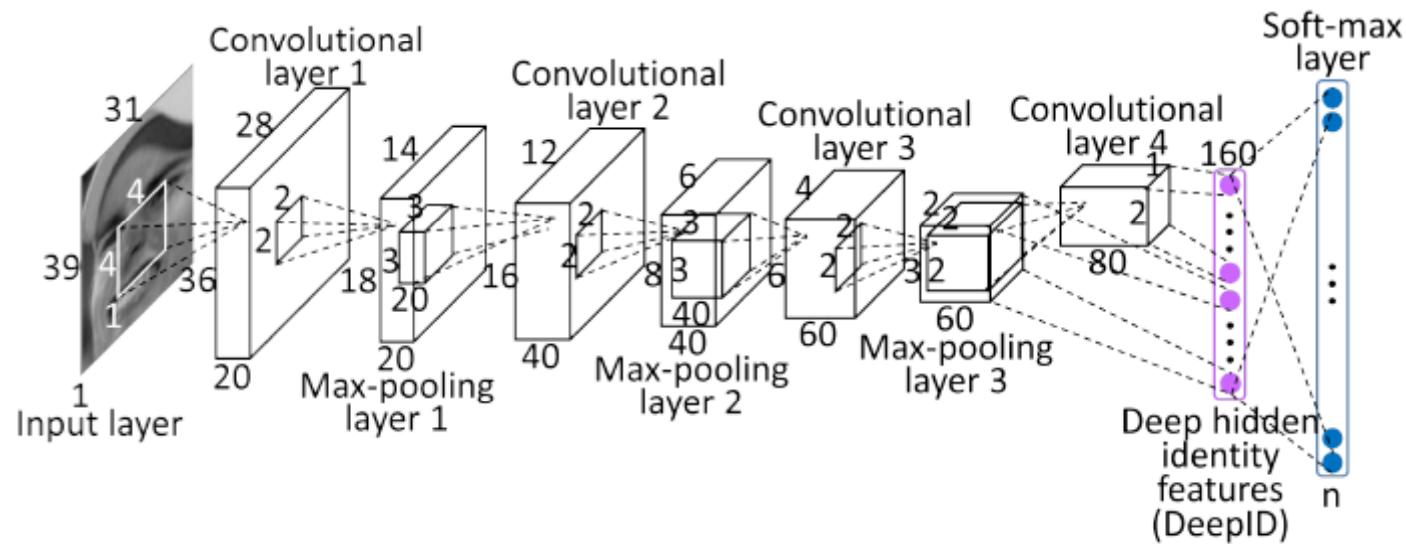
Learn Identity Features with Identification Signal



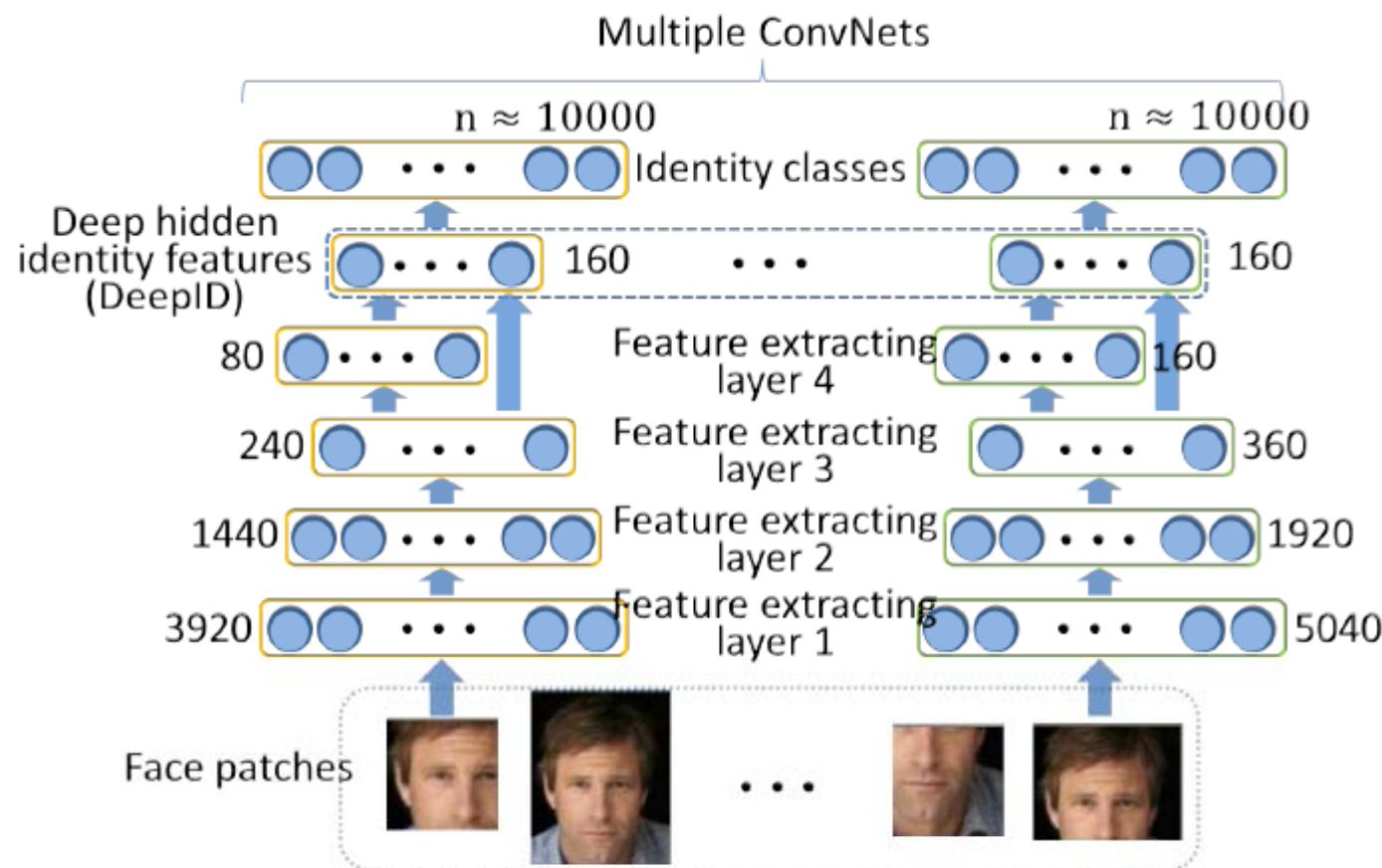
- During training, each image is classified into 10,000 identities with 160 identity features in the top layer
- These features keep rich inter-personal variations
- Features from the last two convolutional layers are effective
- The hidden identity features can be well generalized to other tasks (e.g. verification) and identities outside the training set



- High-dimensional prediction is more challenging, but also adds stronger supervision to the network
- As adding the number of classes to be predicted, the generalization power of the learned features also improves



Extract Features from Multiple ConvNets



Learn Identity Features with Identification Signal

- After combining hidden identity features from multiple CovNets and further reducing dimensionality with PCA, each face image has 150-dimensional features as signature
- These features can be further processed by other classifiers in face verification. Interestingly, we find Joint Bayesian is more effective than cascading another neural network to classify these features

Result on LFW

- We enlarge CelebFaces dataset to CelebFaces+, which include 202,599 images of 10,117 celebrities. CelebFaces+ has no overlap with LFW on identities

Method	Accuracy (%)	No. of points	No. of images	Feature dimension
Joint Bayesian [8]	92.42 (o)	5	99,773	2000×4
ConvNet-RBM [31]	92.52 (o)	3	87,628	N/A
CMD+SLBP [17]	92.58 (u)	3	N/A	2302
Fisher vector faces [29]	93.03 (u)	9	N/A	128×2
Tom-vs-Pete classifiers [2]	93.30 (o+r)	95	20,639	5000
High-dim LBP [9]	95.17 (o)	27	99,773	2000
TL Joint Bayesian [6]	96.33 (o+u)	27	99,773	2000
DeepFace [32]	97.25 (o+u)	6 + 67	$4,400,000 + 3,000,000$	4096×4
DeepID on CelebFaces	96.05 (o)	5	87,628	150
DeepID on CelebFaces+	97.05 (o)	5	202,599	150
DeepID on CelebFaces+ with transfer	97.45 (o+u)	5	202,599	150

“o” denotes using outside training data, however, without using training data from LFW

“o+u” denotes using outside training data and LFW data in the unrestricted protocol for training

Joint Identification-Verification Signals

- Every two feature vectors extracted from the same identity should be close to each other

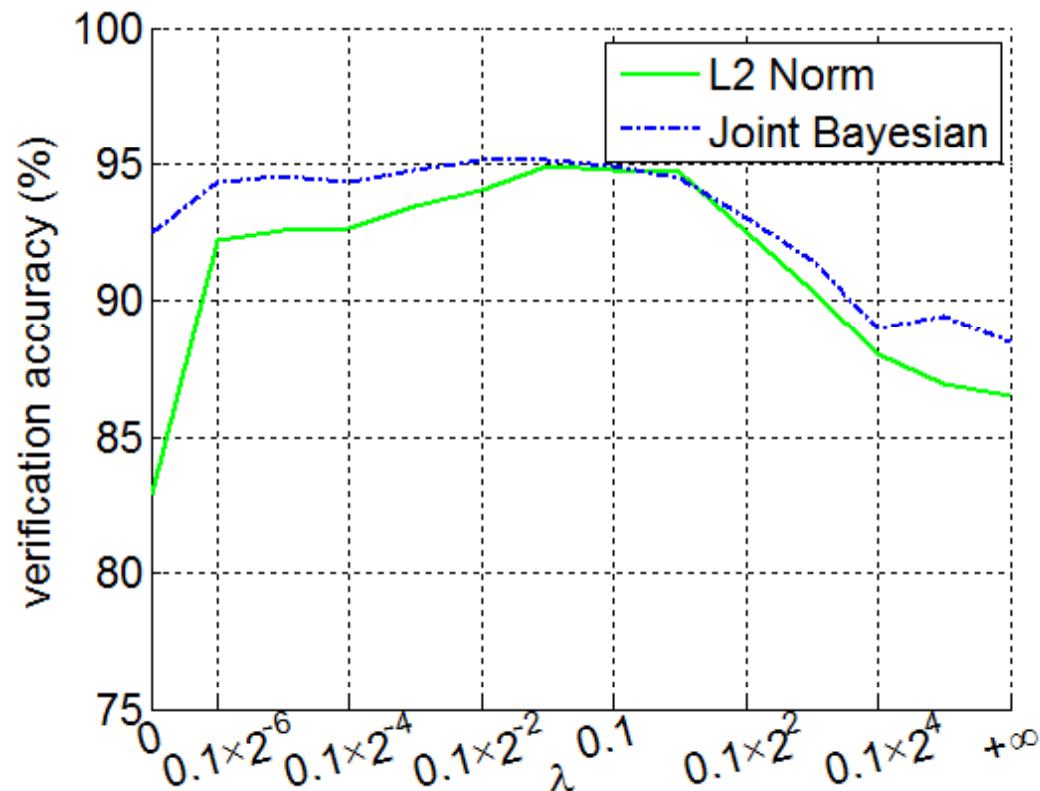
$$\text{Verif}(f_i, f_j, y_{ij}, \theta_{ve}) = \begin{cases} \frac{1}{2} \|f_i - f_j\|_2^2 & \text{if } y_{ij} = 1 \\ \frac{1}{2} \max(0, m - \|f_i - f_j\|_2)^2 & \text{if } y_{ij} = -1 \end{cases}$$

f_i and f_j are feature vectors extracted from two face images in comparison

$y_{ij} = 1$ means they are from the same identity; $y_{ij} = -1$ means different identities

m is a margin to be learned

Balancing Identification and Verification Signals with Parameter λ

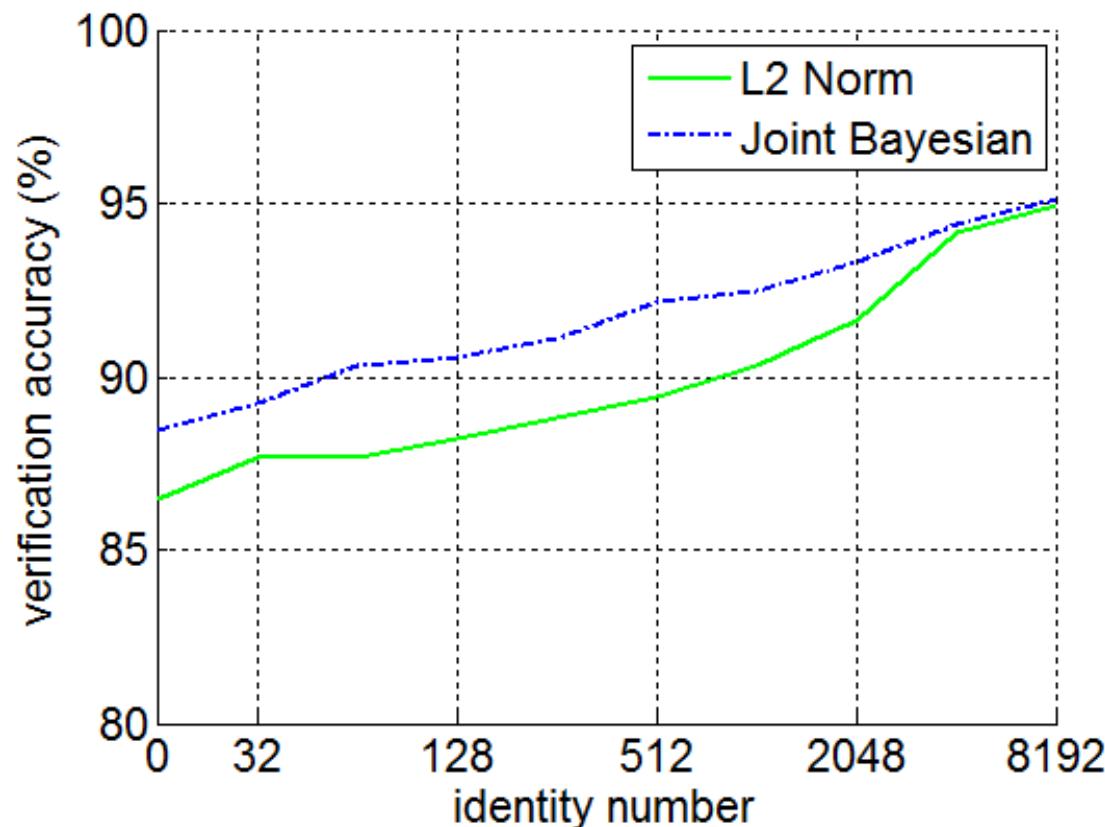


$\lambda = 0$: only identification signal

$\lambda = +\infty$: only verification signal

Rich Identity Information Improves Feature Learning

- Face verification accuracies with the number of training identities

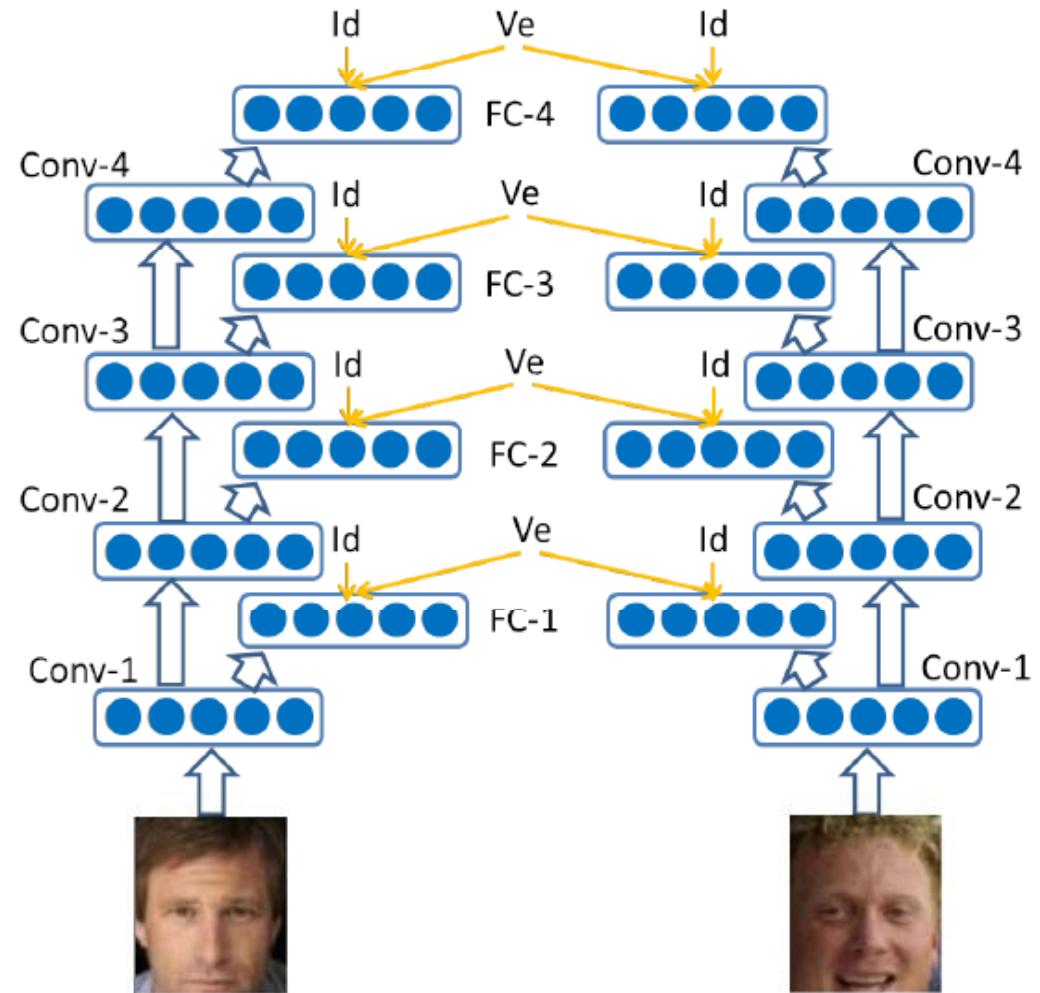


Summary of DeeplD2

- 25 face regions at different scales and locations around landmarks are selected to build 25 neural networks
- All the 160×25 hidden identity features are further compressed into a 180-dimensional feature vector with PCA as a signature for each image
- With a single Titan GPU, the feature extraction process takes 35ms per image

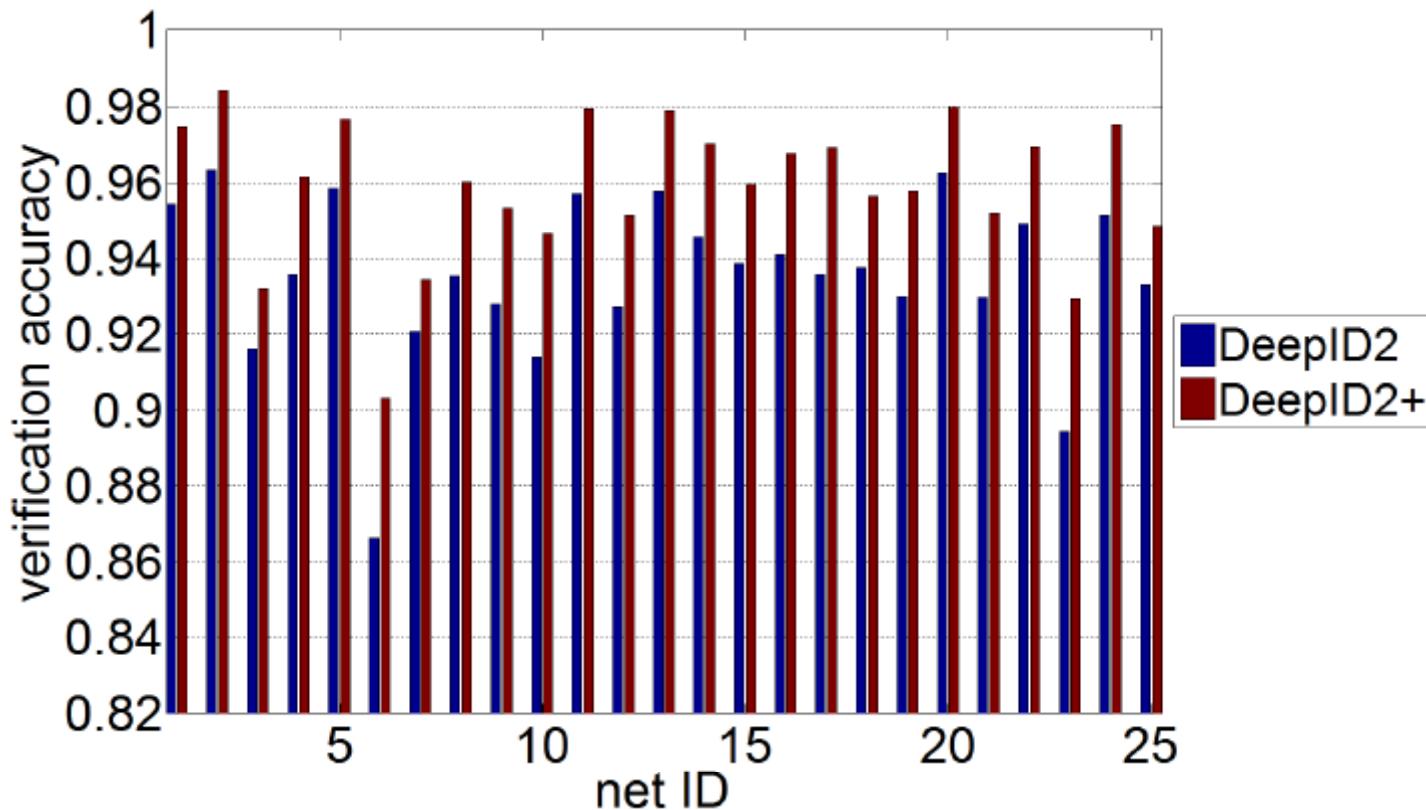
DeepID2+

- Larger net work structures
- Larger training data
- Adding supervisory signals at every layer



Y. Sun, X. Wang, and X. Tang. Deeply learned face representations are sparse, selective, and robust.
arXiv:1412.1265, 2014.

Compare DeepID2 and DeepID2+ on LFW



Comparison of face verification accuracies on LFW with ConvNets trained on 25 face regions given in DeepID2

Best single model is improved from 96.72% to 98.70%

Final Result on LFW

Methods	High-dim LBP [1]	TL Joint Bayesian [2]	DeepFace [3]	DeepID [4]	DeepID2 [5]	DeepID2+ [6]
Accuracy (%)	95.17	96.33	97.35	97.45	99.15	99.47

[1] Chen, Cao, Wen, and Sun. Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification. CVPR, 2013.

[2] Cao, Wipf, Wen, Duan, and Sun. A practical transfer learning algorithm for face verification. ICCV, 2013.

[3] Taigman, Yang, Ranzato, and Wolf. DeepFace: Closing the gap to human-level performance in face verification. CVPR, 2014.

[4] Sun, Wang, and Tang. Deep learning face representation from predicting 10,000 classes. CVPR, 2014.

[5] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep Learning Face Representation by Joint Identification-Verification. NIPS, 2014.

[6] Y. Sun, X. Wang, and X. Tang. Deeply learned face representations are sparse, selective, and robust. arXiv:1412.1265, 2014.

Closed- and open-set face identification on LFW

Method	Rank-1 (%)	DIR @ 1% FAR (%)
COST-S1 [1]	56.7	25
COST-S1+s2 [1]	66.5	35
DeepFace [2]	64.9	44.5
DeepFace+ [3]	82.5	61.9
DeepID2	91.1	61.6
DeepID2+	95.0	80.7

[1] L. Best-Rowden, H. Han, C. Otto, B. Klare, and A. K. Jain. Unconstrained face recognition: Identifying a person of interest from a media collection. *TR MSU-CSE-14-1*, 2014.

[2] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. DeepFace: Closing the gap to human-level performance in face verification. In *Proc. CVPR*, 2014.

[3] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Web-scale training for face identification. Technical report, arXiv:1406.5266, 2014.

Face Verification on YouTube Faces

Methods	Accuracy (%)
LM3L [1]	81.3 ± 1.2
DDML (LBP) [2]	81.3 ± 1.6
DDML (combined) [2]	82.3 ± 1.5
EigenPEP [3]	84.8 ± 1.4
DeepFace [4]	91.4 ± 1.1
DeepID2+	93.2 ± 0.2

[1] J. Hu, J. Lu, J. Yuan, and Y. P. Tan, “Large margin multi-metric learning for face and kinship verification in the wild,” ACCV 2014

[2] J. Hu, J. Lu, and Y. P. Tan, “Discriminative deep metric learning for face verification in the wild,” CVPR 2014

[3] H. Li, G. Hua, X. Shen, Z. Lin, and J. Brandt, “Eigen-pep for video face recognition,” ACCV 2014

[4] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the gap to human-level performance in face verification,” CVPR 2014.

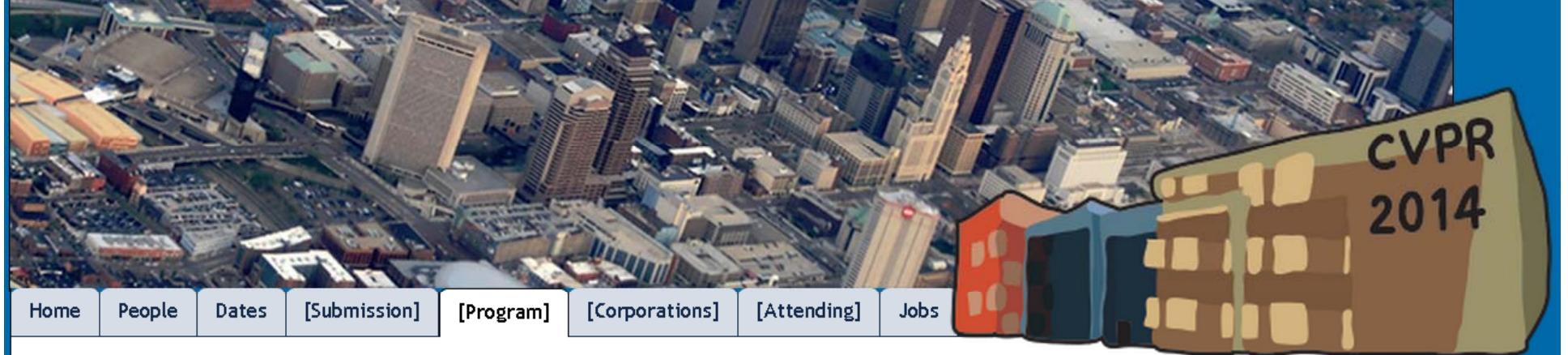
Unified subspace analysis

- Identification signal is in S_b ; verification signal is in S_w
 - Maximize distance between classes under constraint that intrapersonal variation is constant
 - Linear feature mapping
-
- Need to be careful when magnifying the inter-personal difference; Unsupervised learning many be a good choice to remove noise

Joint deep learning

- Learn features by joint identification-verification
- Minimize intra-personal variation under constraint that the distance between classes is constant
- Hierarchical nonlinear feature extraction
- Generalization power increases with more training identities

We still do not know limit of deep learning yet



Home People Dates [Submission] [Program] [Corporations] [Attending] Jobs

CVPR 2014 Plenary Speakers



Neural mechanisms for face processing

[Professor Doris Tsao](#), California Institute of Technology (Caltech)

How the brain distills a representation of meaningful objects from retinal input is one of the central challenges of systems neuroscience. Functional imaging experiments in the macaque reveal that one ecologically important class of objects, faces, is represented by a system of six discrete, strongly interconnected regions. Electrophysiological recordings show that these 'face patches' have unique functional profiles. By studying the distinct visual representations maintained in these six face patches, the sequence of information flow between them, and the role each plays in face perception, we are gaining new insights into hierarchical information processing in the brain.

What has been learned by DeepID2+?

Properties owned by neurons?

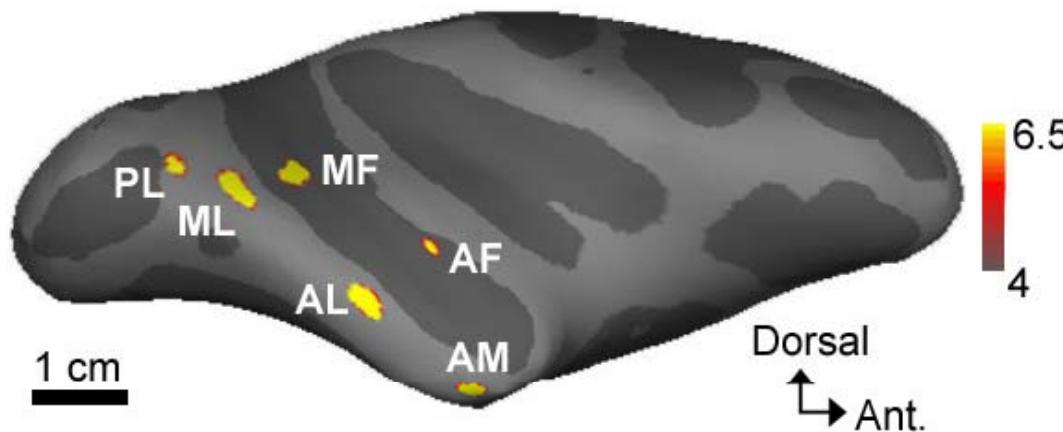
Moderate sparse

Selective to identities and attributes

Robust to data corruption

These properties are naturally owned by DeepID2+ through large-scale training, without explicitly adding regularization terms to the model

Biological Motivation

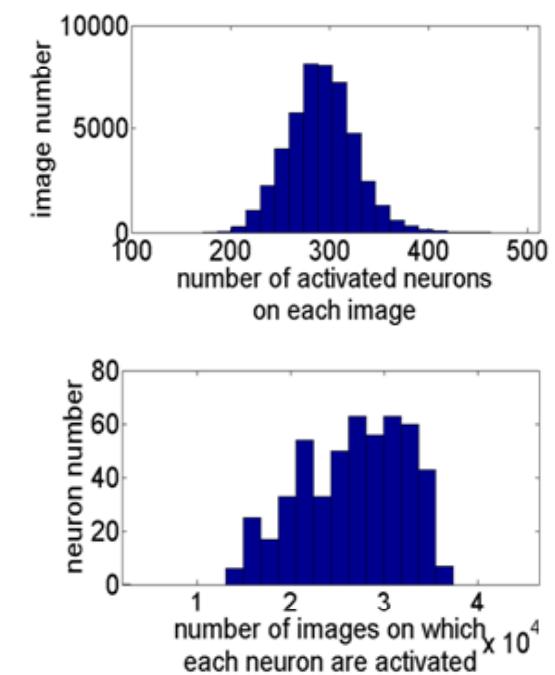
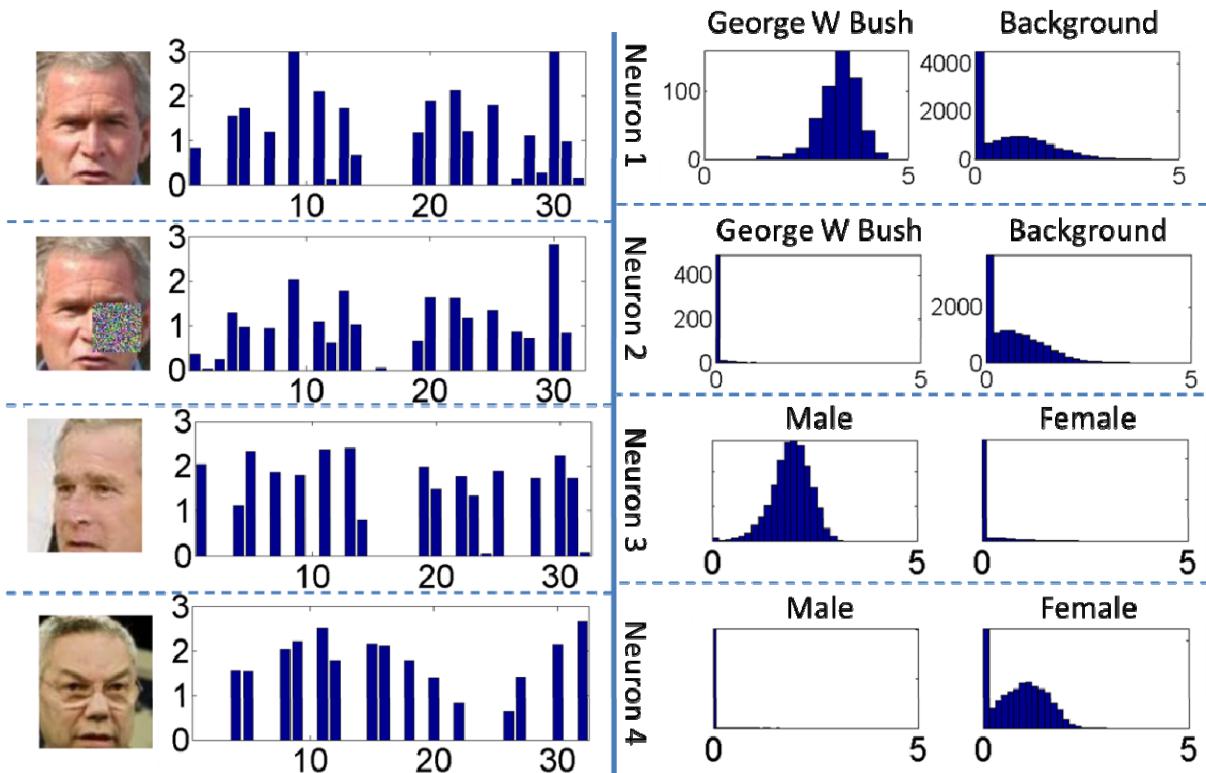


- Monkey has a face-processing network that is made of six interconnected face-selective regions
- Neurons in some of these regions were view-specific, while some others were tuned to identity across views
- View could be generalized to other factors, e.g. expressions?

Winrich A. Freiwald and Doris Y. Tsao, "Functional compartmentalization and viewpoint generalization within the macaque face-processing system," *Science*, 330(6005):845–851, 2010.

Deeply learned features are moderately sparse

- For an input image, about half of the neurons are activated
- A neuron has response on about half of the images



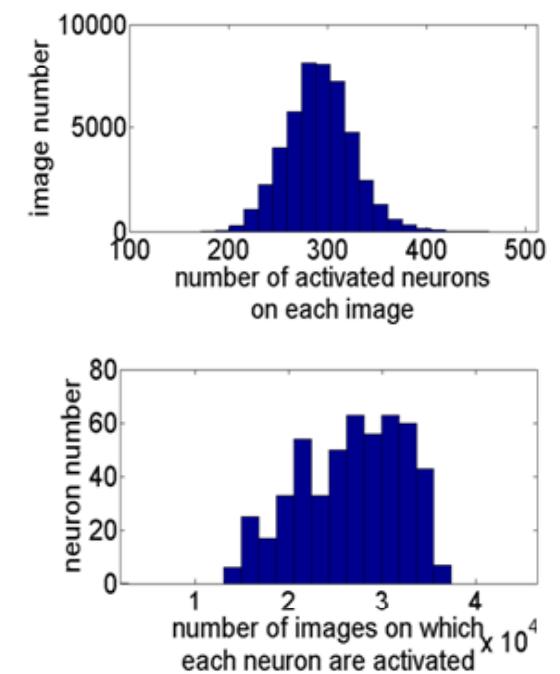
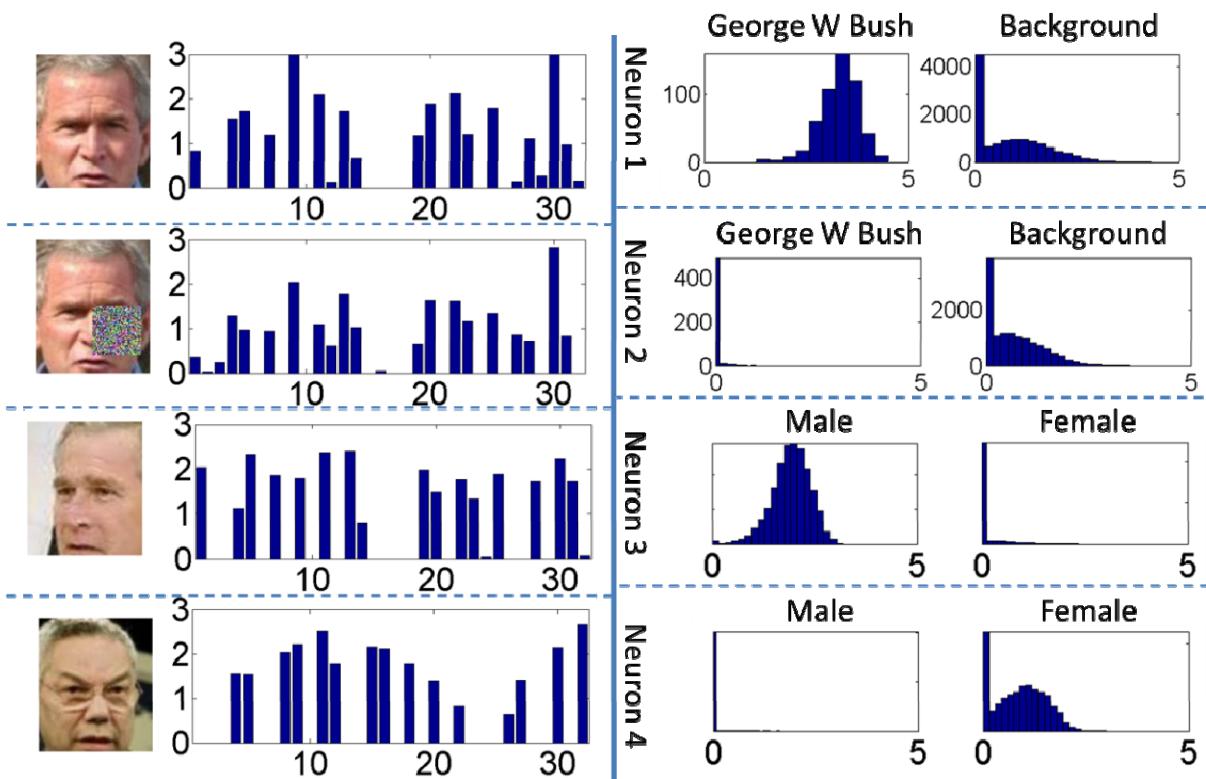
Deeply learned features are moderately space

- The binary codes on activation patterns of neurons are very effective on face recognition
- Activation patterns are more important than activation magnitudes in face recognition

	Joint Bayesian (%)	Hamming distance (%)
Single model (real values)	98.70	n/a
Single model (binary code)	97.67	96.46
Combined model (real values)	99.47	n/a
Combined model (binary code)	99.12	97.47

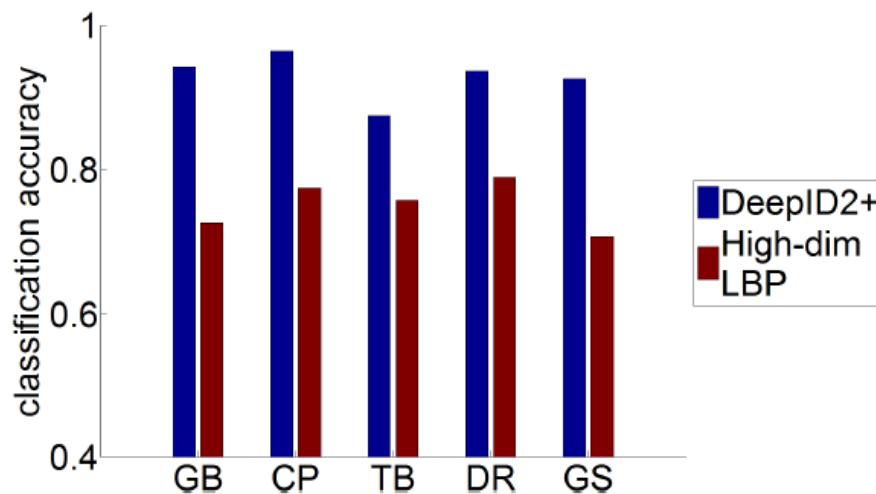
Deeply learned features are selective to identities and attributes

- With a single neuron, DeepID2 reaches 97% recognition accuracy for some identity and attribute

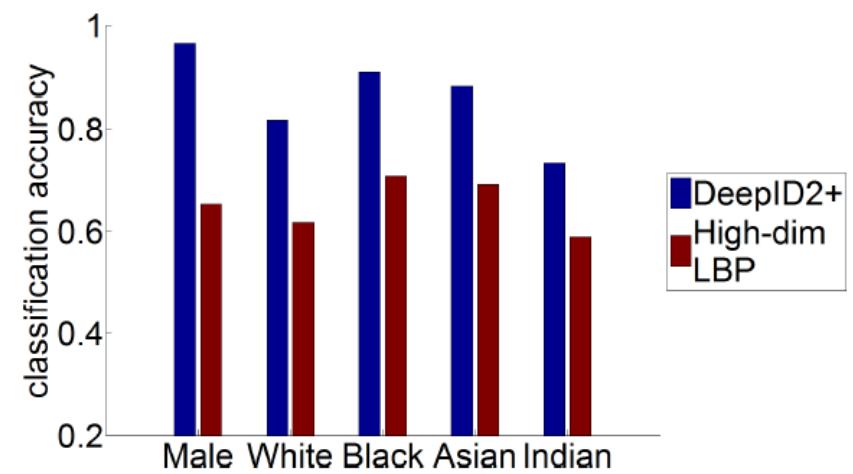


Deeply learned features are selective to identities and attributes

- With a single neuron, DeepID2 reaches 97% recognition accuracy for some identity and attribute



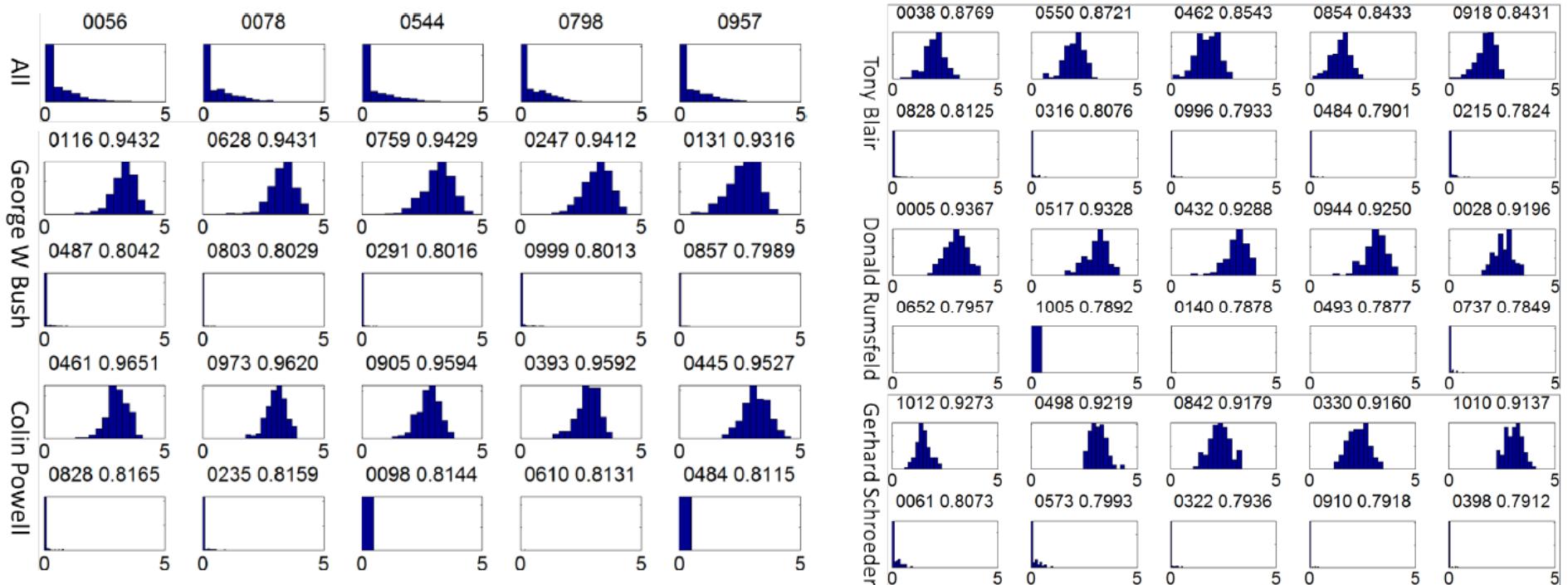
Identity classification accuracy on LFW with one single DeepID2+ or LBP feature. GB, CP, TB, DR, and GS are five celebrities with the most images in LFW.



Attribute classification accuracy on LFW with one single DeepID2+ or LBP feature.

Deeply learned features are selective to identities and attributes

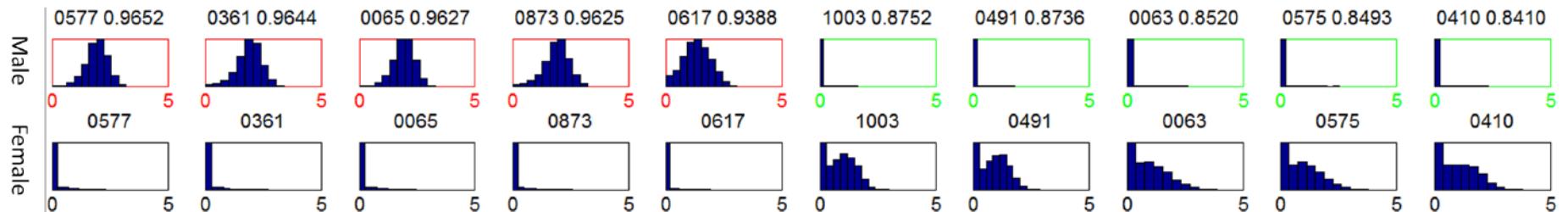
- Excitatory and inhibitory neurons



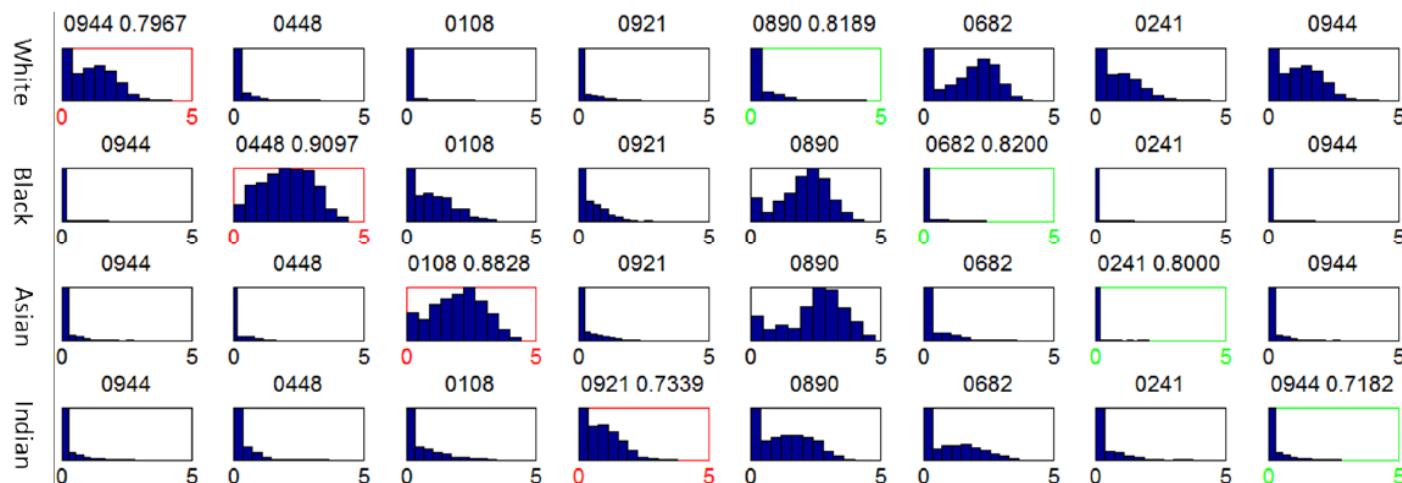
Histograms of neural activations over identities with the most images in LFW

Deeply learned features are selective to identities and attributes

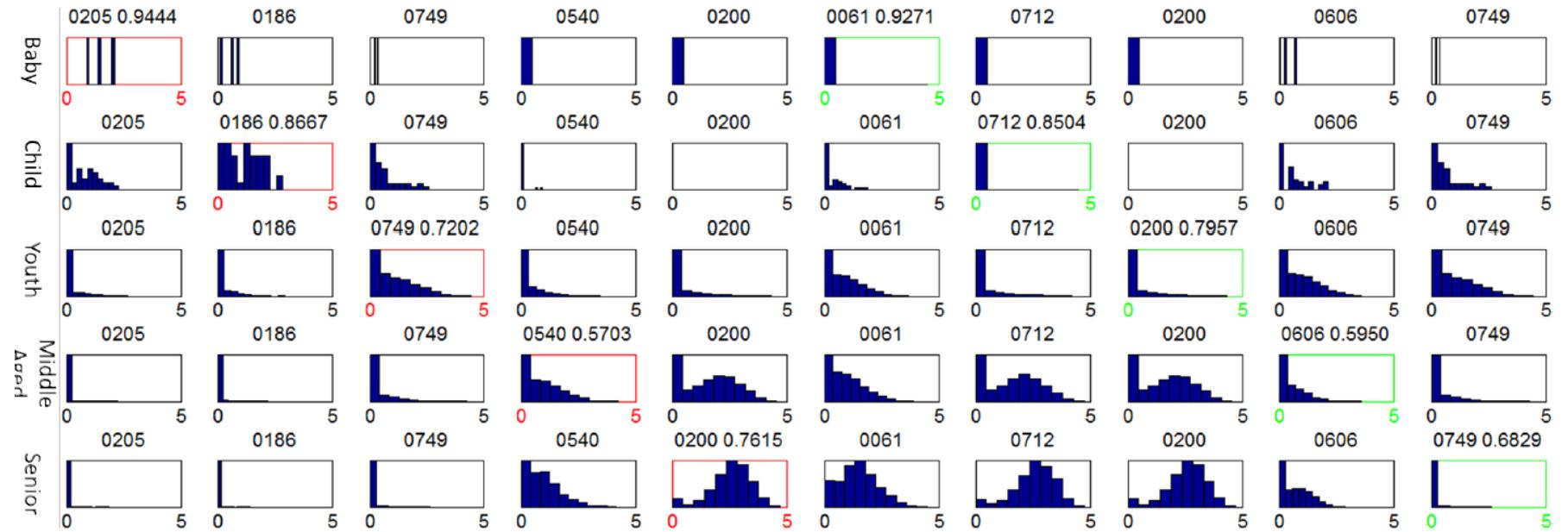
- Excitatory and inhibitory neurons



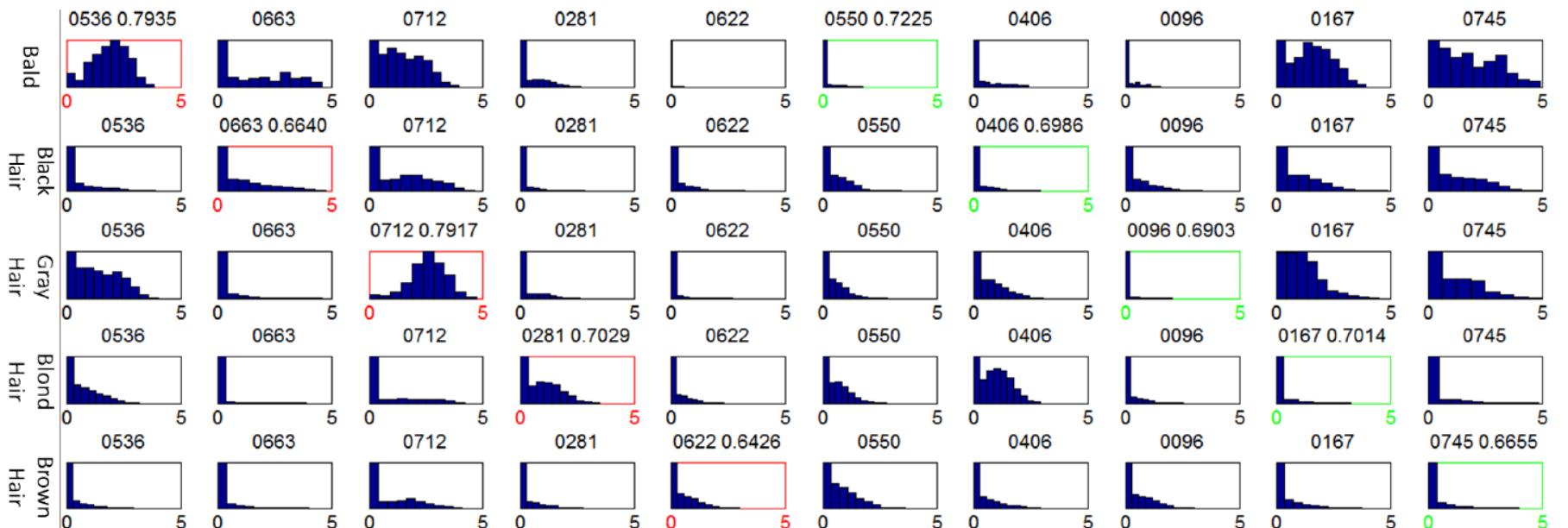
Histograms of neural activations over gender-related attributes (Male and Female)



Histograms of neural activations over race-related attributes (White, Black, Asian and India)

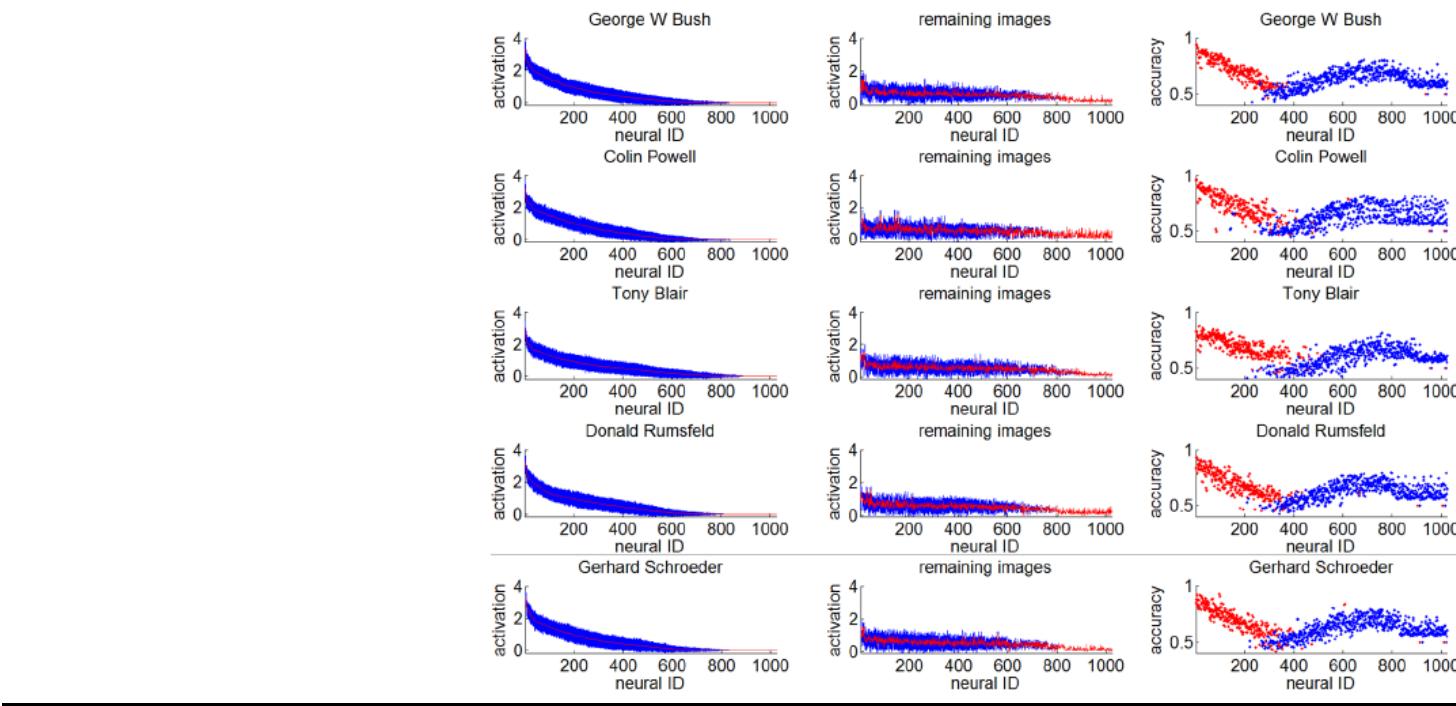


Histogram of neural activations over age-related attributes (Baby, Child, Youth, Middle Aged, and Senior)

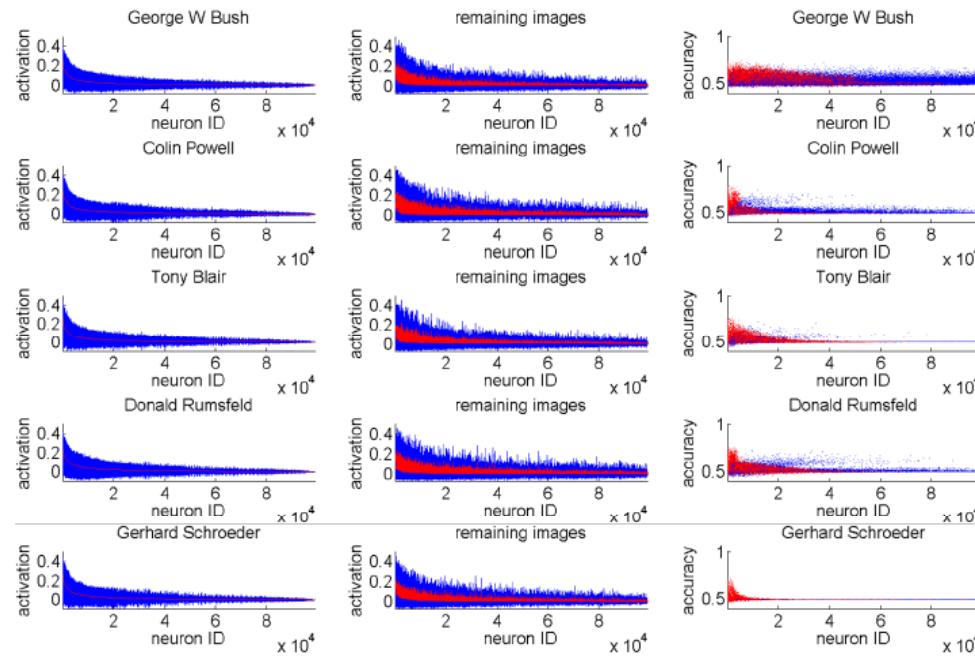


Histogram of neural activations over hair-related attributes (Bald, Black Hair, Gray Hair, Blond Hair, and Brown Hair).

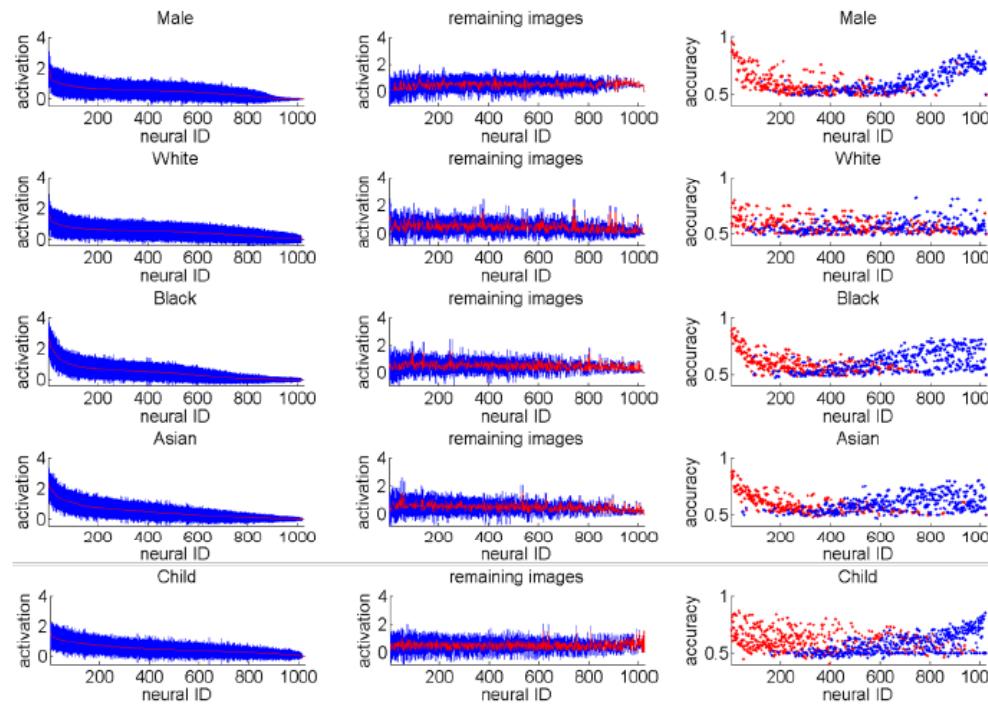
DeepID2+



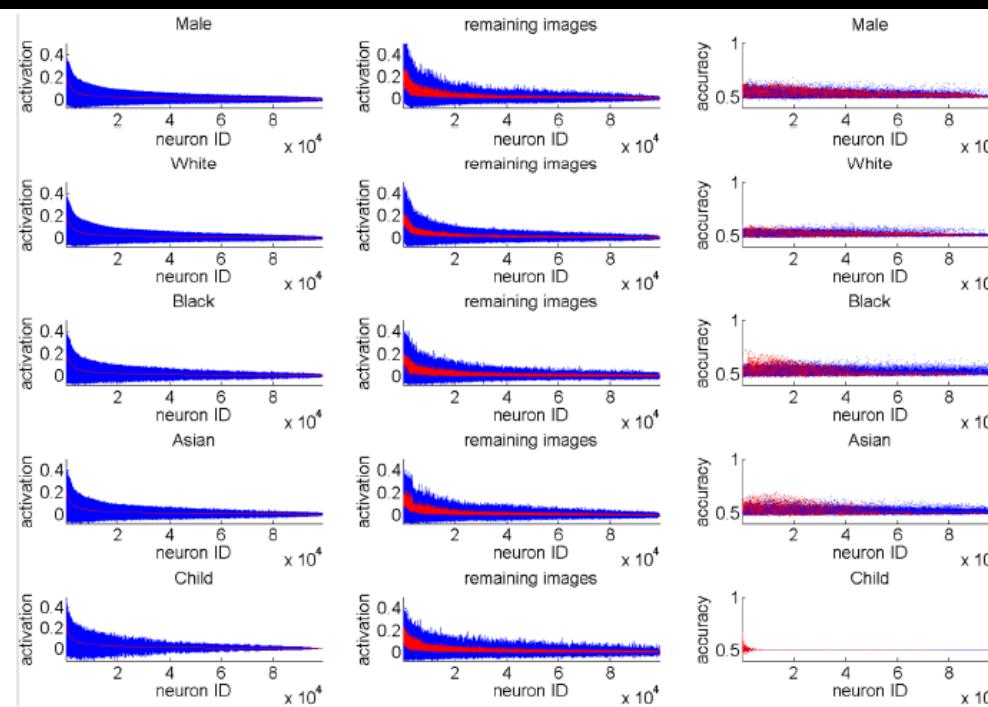
High-dim LBP



DeepID2+

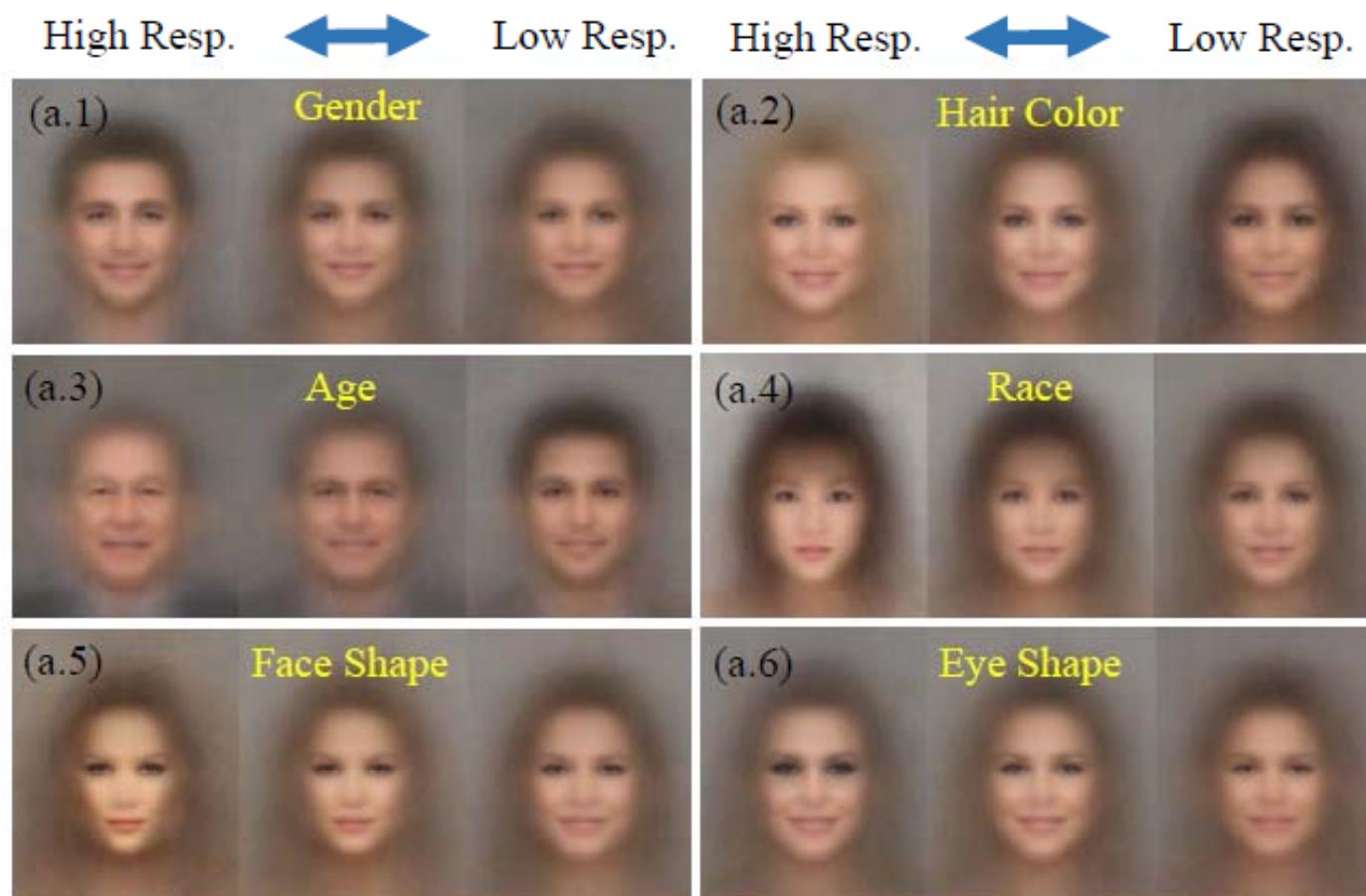


High-dim LBP



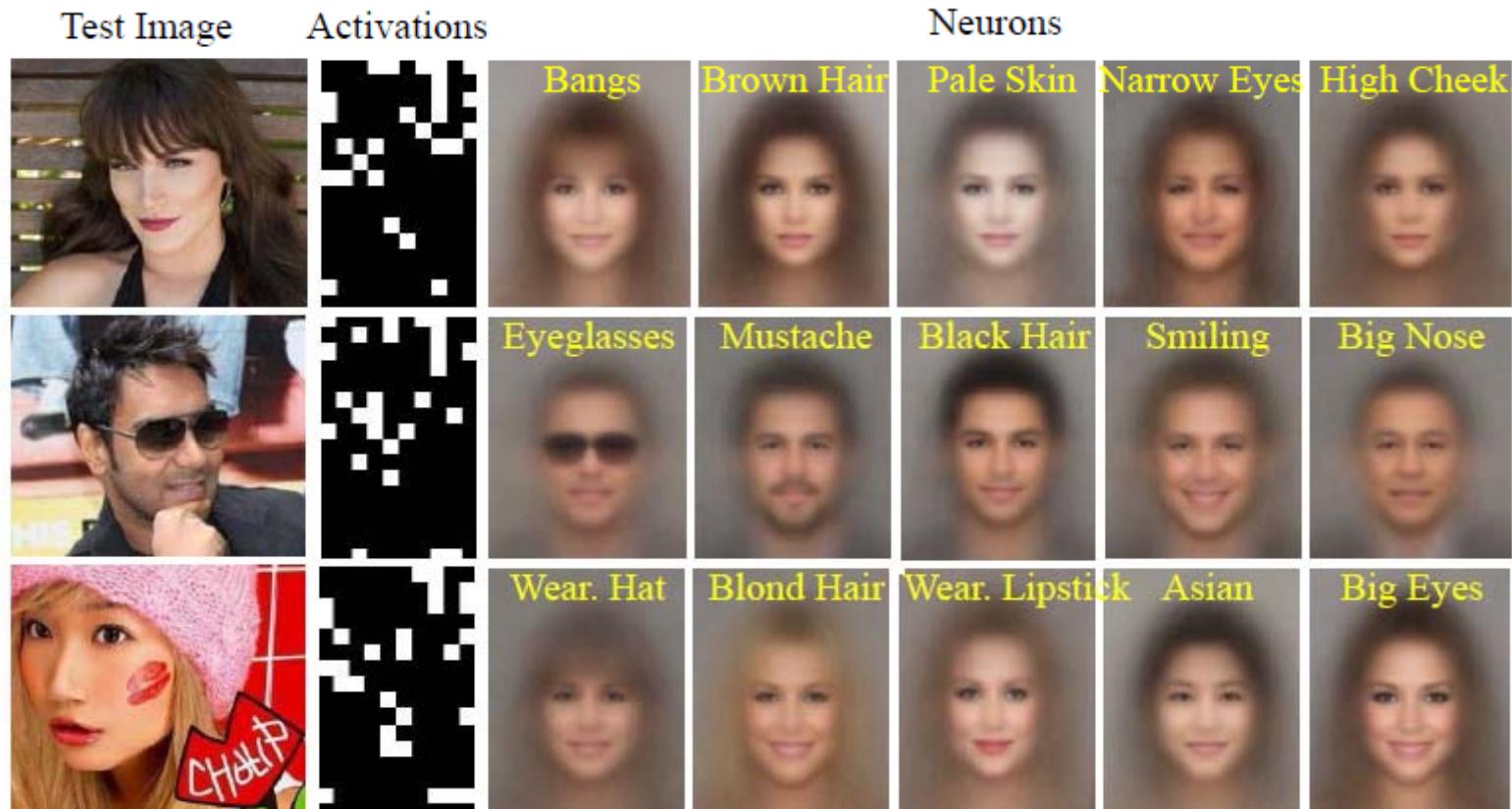
Deeply learned features are selective to identities and attributes

- Visualize the semantic meaning of each neuron



Deeply learned features are selective to identities and attributes

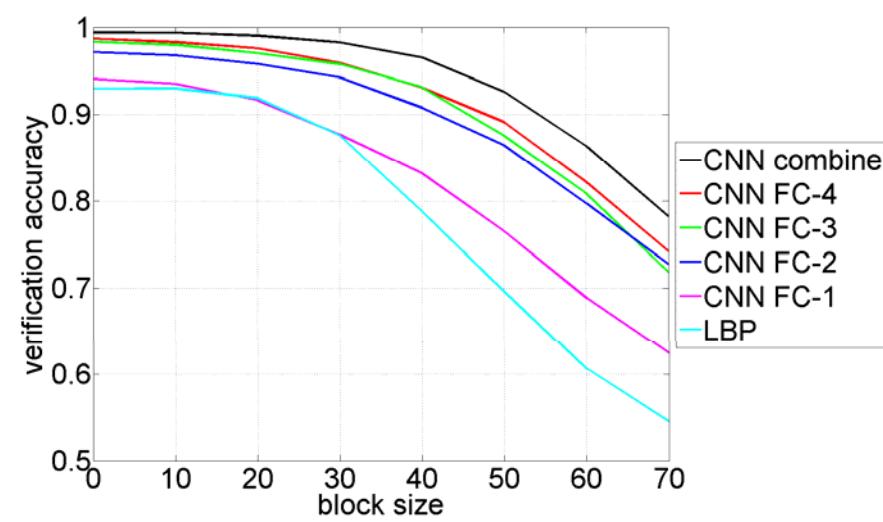
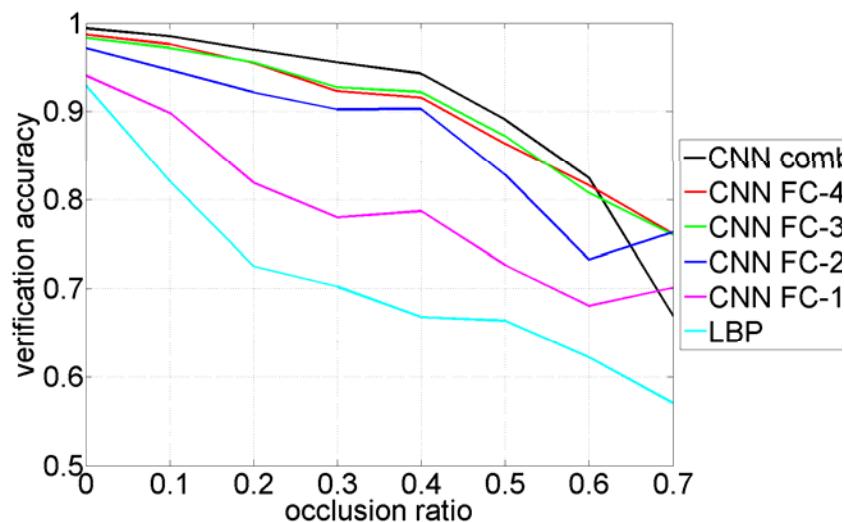
- Visualize the semantic meaning of each neuron

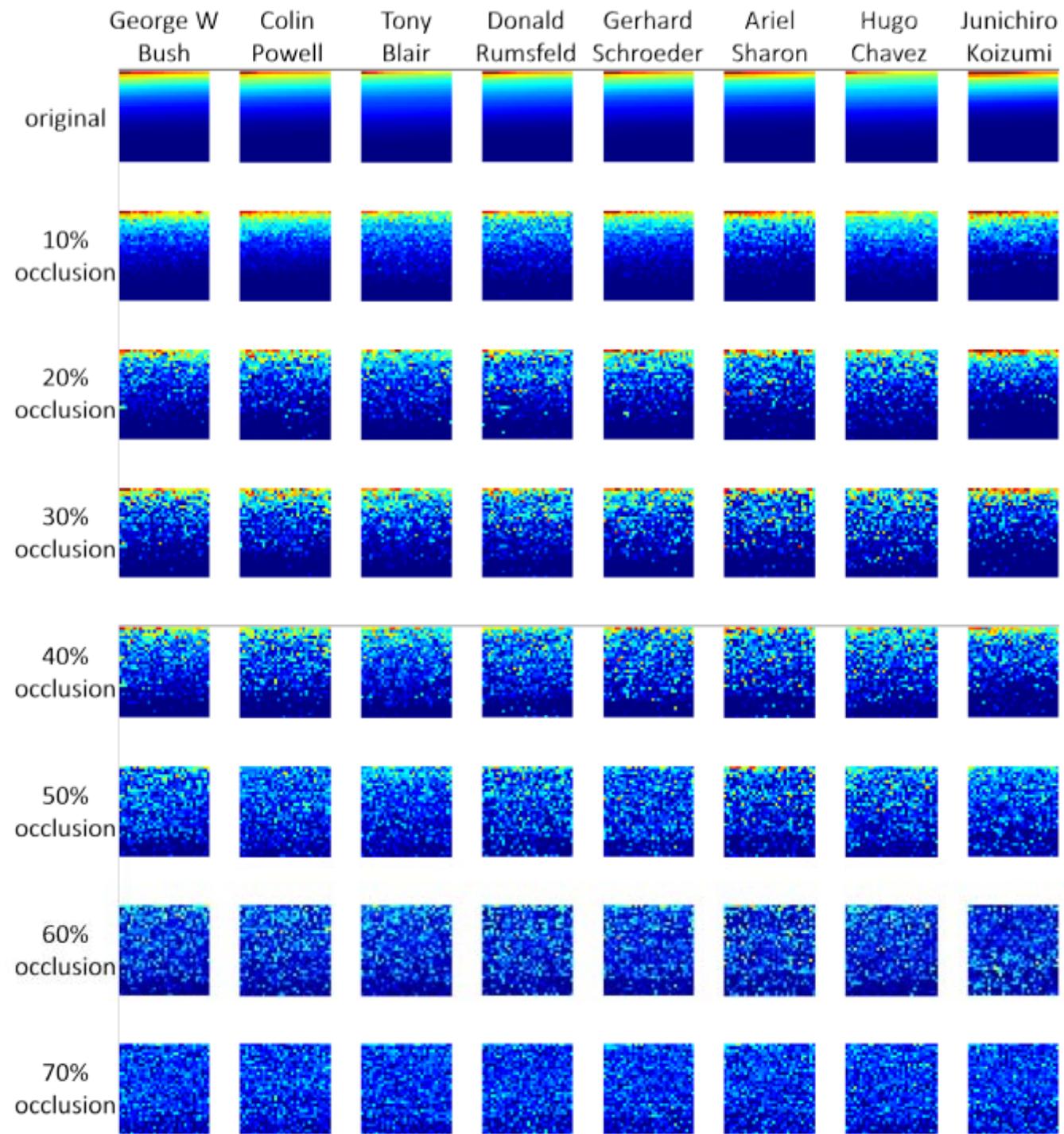


Neurons are ranked by their responses in descending order with respect to test images

Deeply learned features are robust to occlusions

- Global features are more robust to occlusions



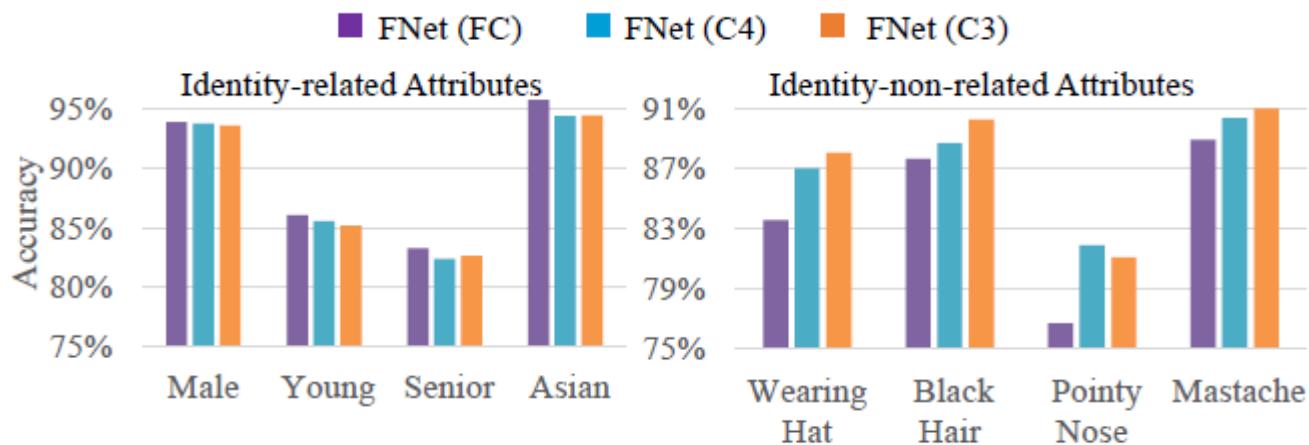


Can features learned by DeepID be effectively applied to other face related tasks, such as face localization and face attribute recognition?

Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep Learning Face Attributes in the Wild”, arXiv: 1411.7766, 2014

DeepID2 features for attribute recognition

- Features at top layers are more effective on recognizing identity related attributes
- Features at lower layers are more effective on identity-non-related attributes



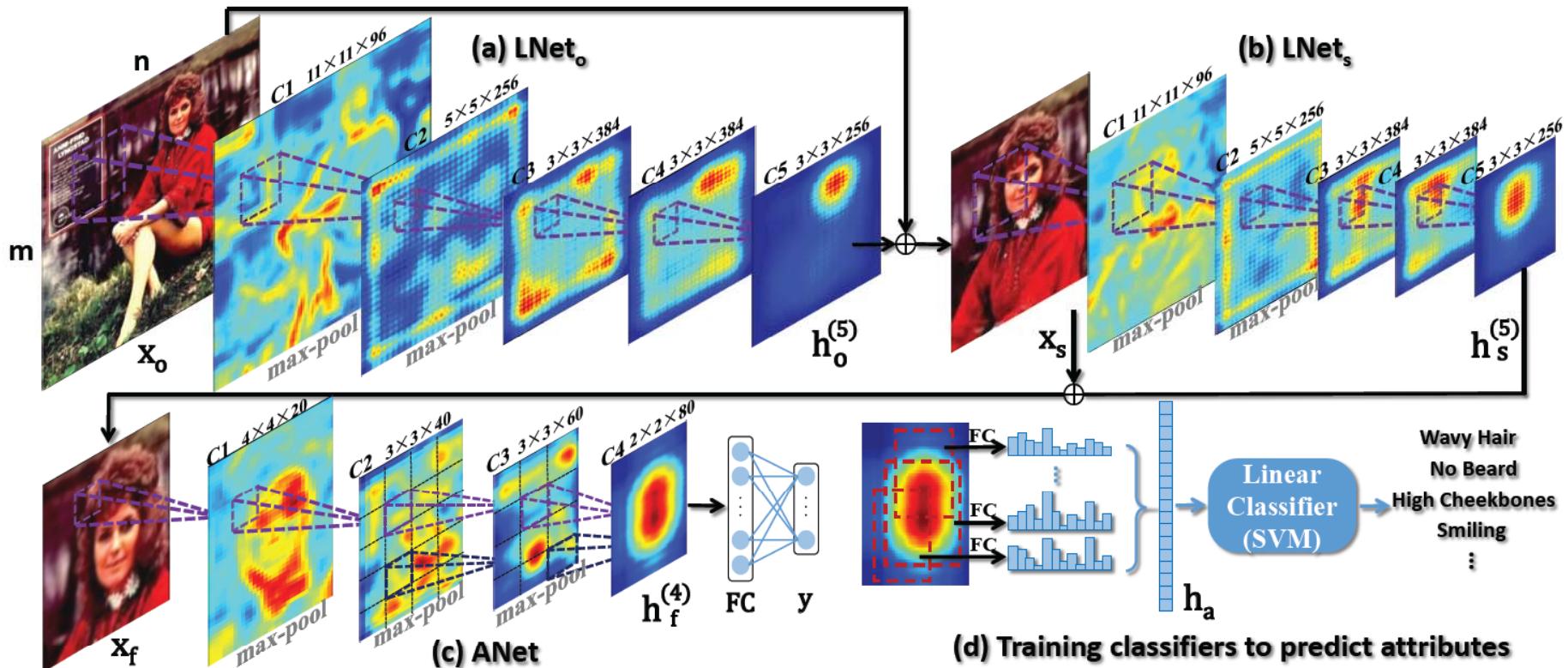
DeepID2 features for attribute recognition

- DeepID2 features can be directly used for attribute recognition
- Use DeepID2 features as initialization (pre-trained result), and then fine tune on attribute recognition
- Average accuracy on 40 attributes on CelebA and LFWA datasets

	CelebA	LFWA
FaceTracer [1] (HOG+SVM)	81	74
PANDA-W [2] (Parts are automatically detected)	79	71
PANDA-L [2] (Parts are given by ground truth)	85	81
DeepID2	84	82
Fine-tune (w/o DeepID2)	83	79
DeepID2 + fine-tune	87	84

Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep Learning Face Attributes in the Wild,” arXiv:1411.7766, 2014.

Features learned by DeepID and attribute recognition are effective on face localization



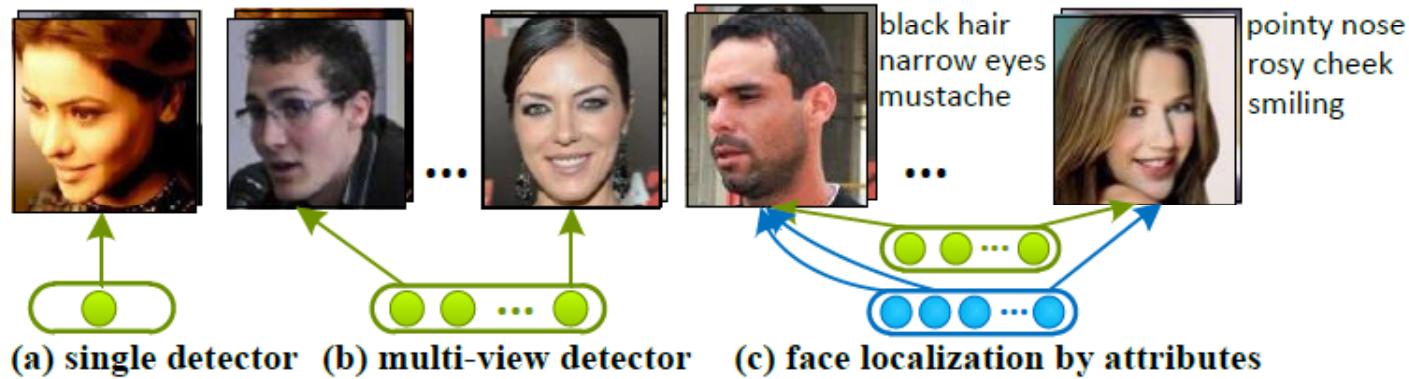
Lnets are pre-trained with ImageNet
Both are fine-trained with face attributes

Lnet_o calculates a response map which indicates the region of head-shoulder

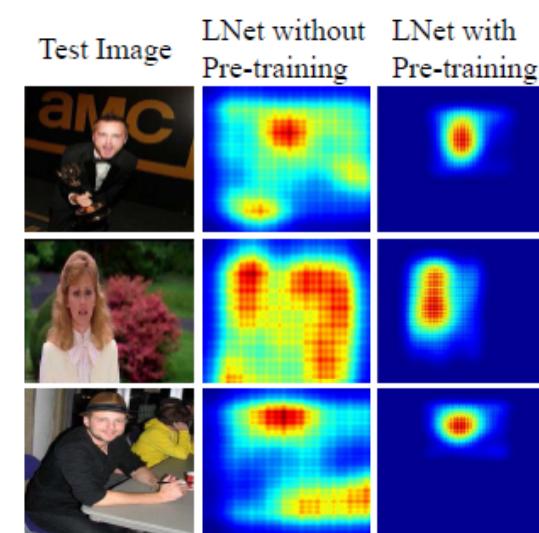
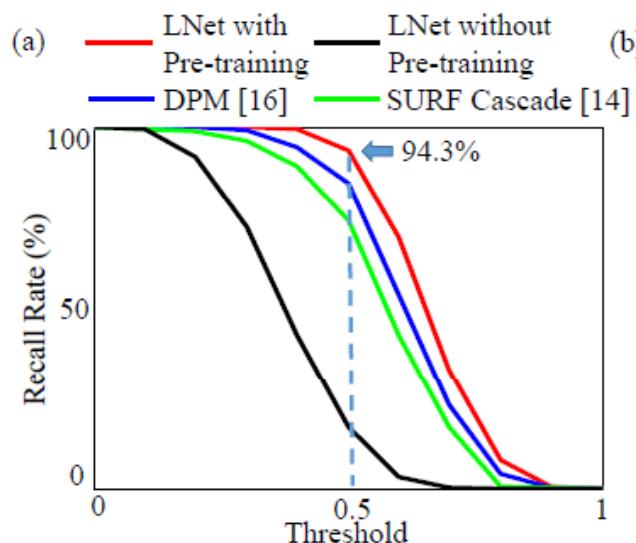
Lnet_s refines the location of face

Anet extracts features to recognize attributes

ANet is pre-trained with DeepID



Each neuron learned from face attribute recognition servers as a face detector, and it extends the idea of multi-view face detector to an extreme case



Deep Learning for Face Recognition

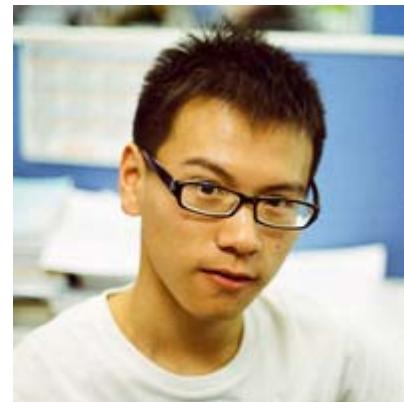
The projects started from December of 2012

DeepID



Yi Sun

MVP



Zhenyao Zhu

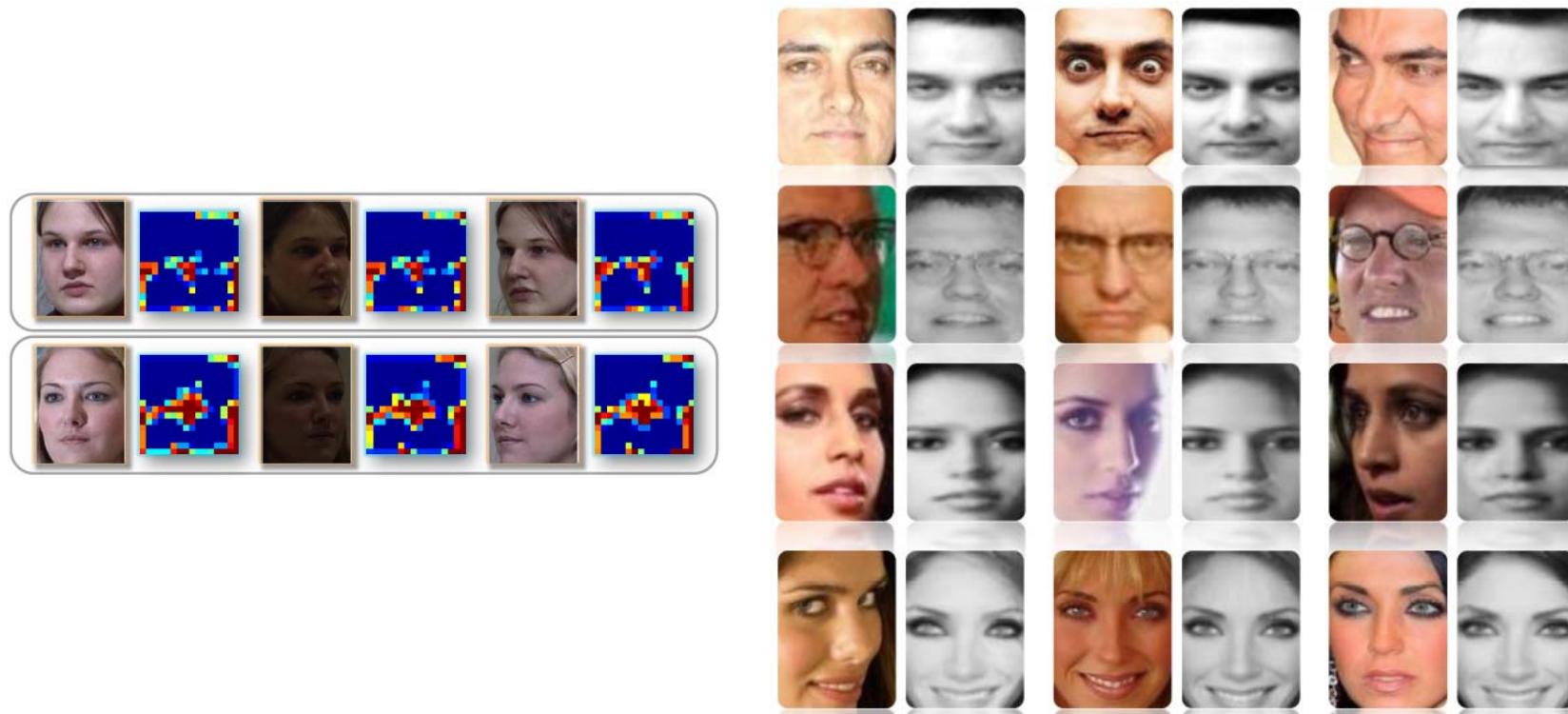


Ping Luo

Our understanding of deep learning

- Deep models can disentangle hidden factors with different neurons**
- Deep models can be a combination of random and determinant neurons**
- Image reconstruction is a stronger supervision task and can be used to learn features**

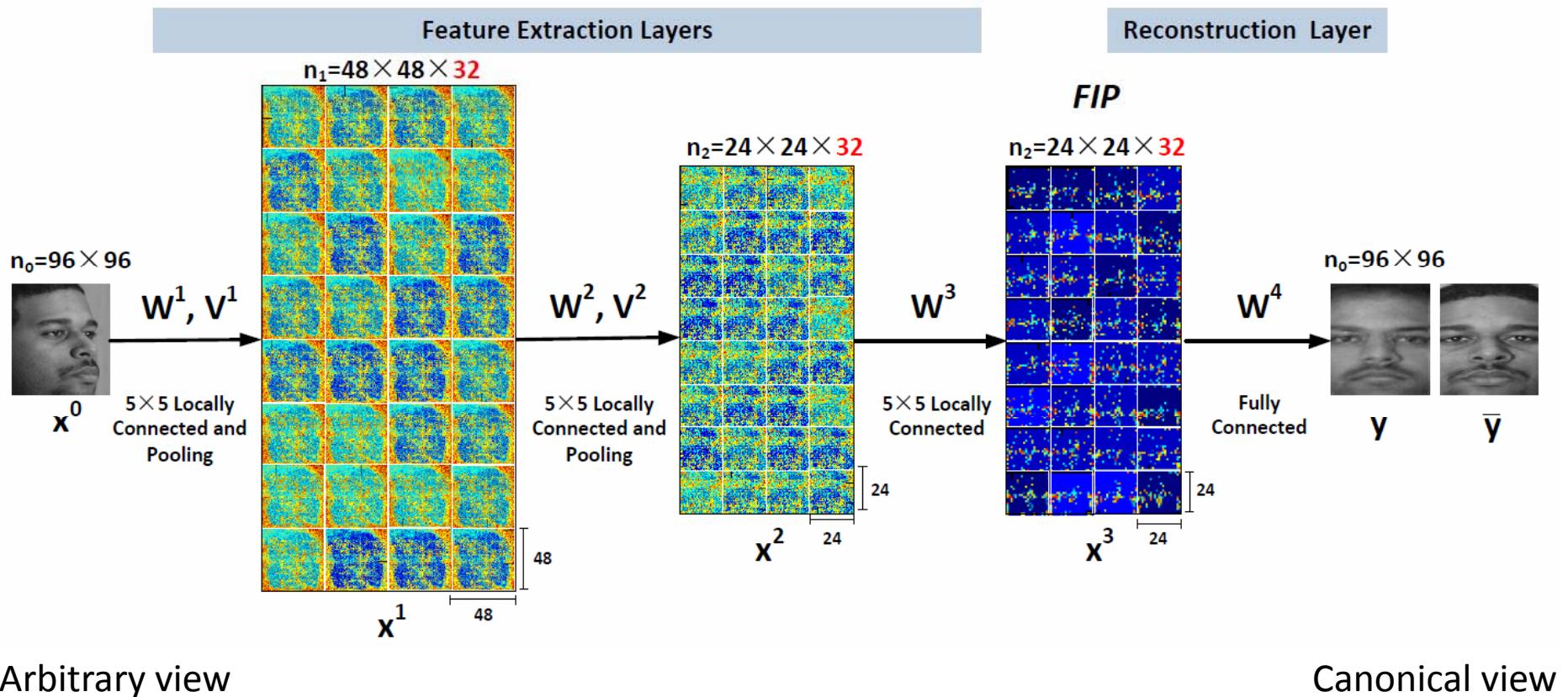
Example 2: deep learning face identity features by recovering canonical-view face images

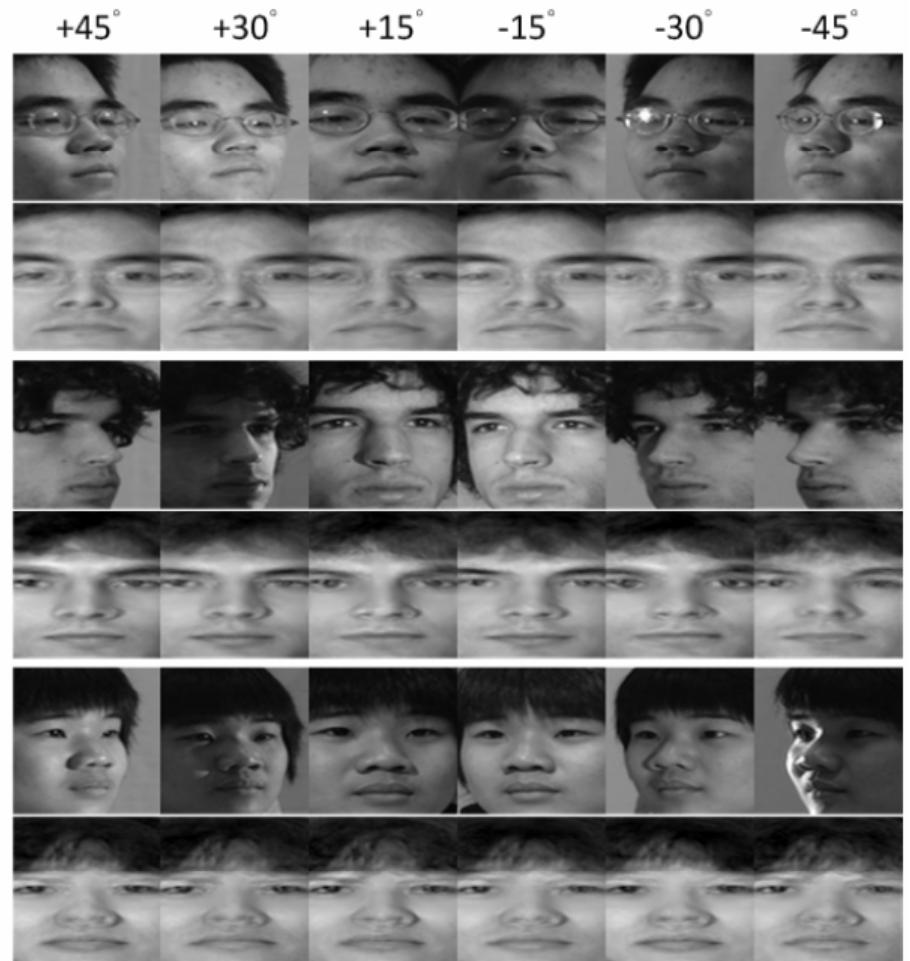


Reconstruction examples from LFW

Z. Zhu, P. Luo, X. Wang, and X. Tang, “Deep Learning Identity Preserving Face Space,” ICCV 2013.

- Deep model can disentangle hidden factors through feature extraction over multiple layers
- No 3D model; no prior information on pose and lighting condition
- Model multiple complex transforms
- Reconstructing the whole face is a much strong supervision than predicting 0/1 class label and helps to avoid overfitting





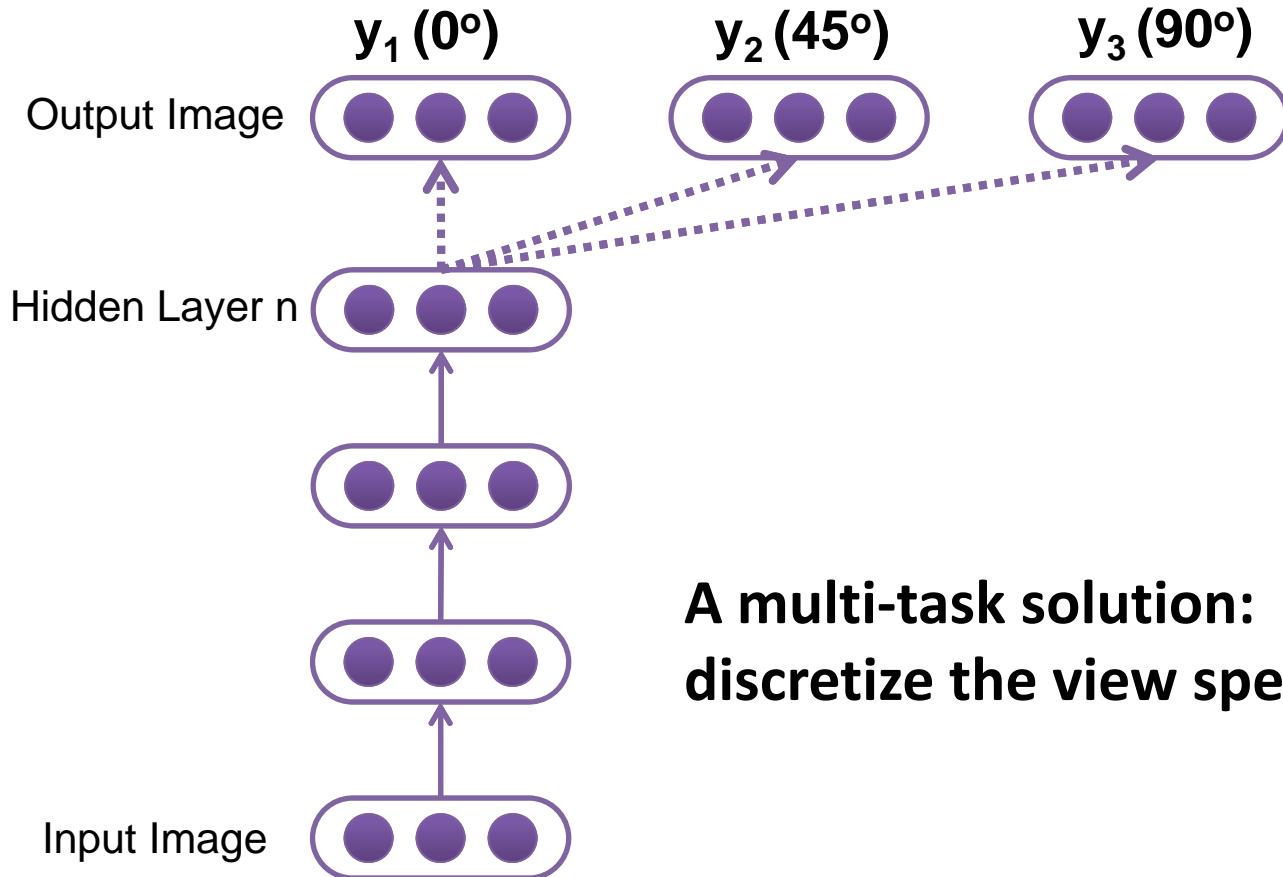
Comparison on Multi-PIE

	-45°	-30°	-15°	+15°	+30°	+45°	Avg	Pose
LGBP [26]	37.7	62.5	77	83	59.2	36.1	59.3	✓
VAAM [17]	74.1	91	95.7	95.7	89.5	74.8	86.9	✓
FA-EGFC[3]	84.7	95	99.3	99	92.9	85.2	92.7	✗
SA-EGFC[3]	93	98.7	99.7	99.7	98.3	93.6	97.2	✓
LE[4] + LDA	86.9	95.5	99.9	99.7	95.5	81.8	93.2	✗
CRBM[9] + LDA	80.3	90.5	94.9	96.4	88.3	89.8	87.6	✗
Ours	95.6	98.5	100.0	99.3	98.5	97.8	98.3	✗

- [3] A. Asthana, T. K. Marks, M. J. Jones, K. H. Tieu, and M. Rohith. Fully automatic pose-invariant face recognition via 3d pose normalization. In *ICCV*, pages 937–944, 2011. [1](#), [5](#), [6](#)
- [4] Z. Cao, Q. Yin, X. Tang, and J. Sun. Face recognition with learning-based descriptor. In *CVPR*, pages 2707–2714, 2010. [2](#), [3](#), [6](#)
- [9] G. B. Huang, H. Lee, and E. Learned-Miller. Learning hierarchical representations for face verification with convolutional deep belief networks. In *CVPR*, pages 2518–2525, 2012. [3](#), [6](#)
- [17] S. Li, X. Liu, X. Chai, H. Zhang, S. Lao, and S. Shan. Morphable displacement field based image matching for face recognition across pose. In *ECCV*, pages 102–115, 2012. [1](#), [2](#), [5](#), [6](#)
- [26] W. Zhang, S. Shan, W. Gao, X. Chen, and H. Zhang. Local gabor binary pattern histogram sequence (lgbphs): A novel non-statistical model for face representation and recognition. In *ICCV*, volume 1, pages 786–791, 2005. [5](#), [6](#)

It is still not a 3D representation yet

Can we reconstruct all the views?



**A multi-task solution:
discretize the view spectrum**

1. The number of views to be reconstructed is predefined, equivalent to the number of tasks
2. Model complexity increases as the number of views
3. Encounters problems when the training data of different views are unbalanced
4. Cannot reconstruct views not presented in the training set

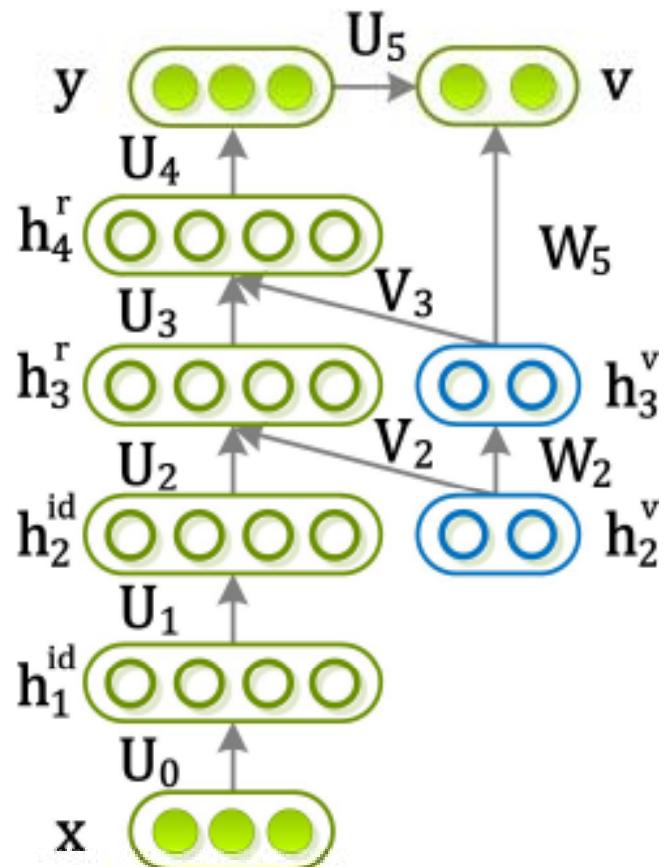
Deep Learning Multi-view Representation from 2D Images

- Identity and view represented by different sets of neurons
- Continuous view representation
- Given an image under arbitrary view, its viewpoint can be estimated and its full spectrum of views can be reconstructed



Z. Zhu, P. Luo, X. Wang, and X. Tang, "Deep Learning and Disentangling Face Representation by Multi-View Perception," NIPS 2014.

Deep Learning Multi-view Representation from 2D Images



x and y are input and output images of the same identity but in different views;

v is the view label of the output image;

h^{id} are neurons encoding identity features

h^v are neurons encoding view features

h^r are neurons encoding features to reconstruct the output images

Deep Learning by EM

- EM updates on the probabilistic model are converted to forward and backward propagation

$$\mathcal{L}(\Theta, \Theta^{old}) = \sum_{\mathbf{h}^v} p(\mathbf{h}^v | \mathbf{y}, \mathbf{v}; \Theta^{old}) \log p(\mathbf{y}, \mathbf{v}, \mathbf{h}^v | \mathbf{h}^{id}; \Theta)$$

- E-step: proposes s samples of \mathbf{h}

$$\mathbf{h}_s^v \sim \mathcal{U}(0, 1)$$

$$w_s = p(\mathbf{y}, \mathbf{v} | \mathbf{h}^v; \Theta^{old})$$

- M-step: compute gradient refer to \mathbf{h} with largest w_s

$$\frac{\partial \mathcal{L}(\Theta)}{\partial \Theta} \simeq \frac{\partial}{\partial \Theta} \left\{ w_s \left(\log p(\mathbf{v} | \mathbf{y}, \mathbf{h}_s^v) + \log p(\mathbf{y} | \mathbf{h}^{id}, \mathbf{h}_s^v) \right) \right\}$$

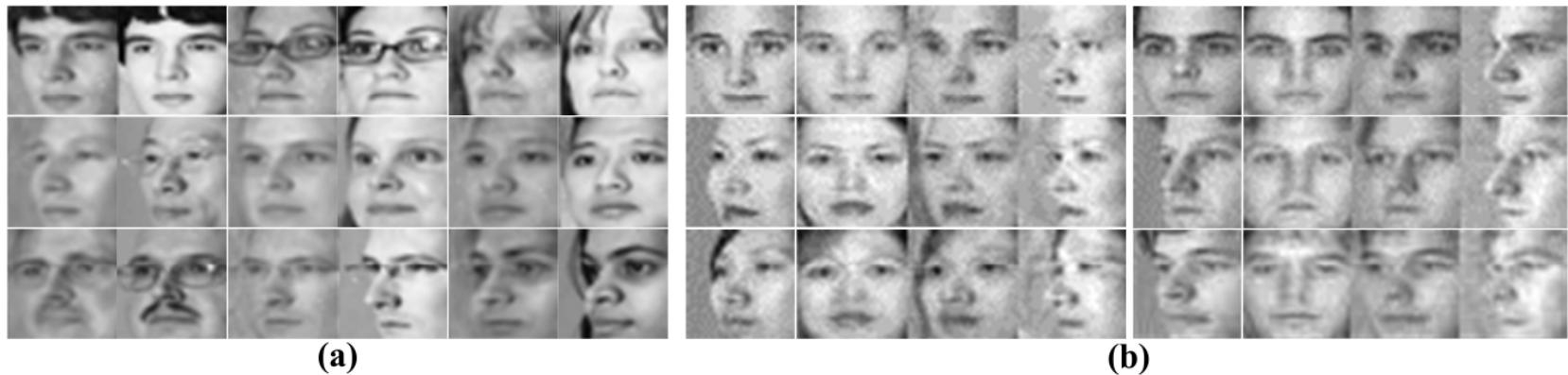
	Avg.	0°	-15°	$+15^\circ$	-30°	$+30^\circ$	-45°	$+45^\circ$	-60°	$+60^\circ$
Raw Pixels+LDA	36.7	81.3	59.2	58.3	35.5	37.3	21.0	19.7	12.8	7.63
LBP [1]+LDA	50.2	89.1	77.4	79.1	56.8	55.9	35.2	29.7	16.2	14.6
Landmark LBP [6]+LDA	63.2	94.9	83.9	82.9	71.4	68.2	52.8	48.3	35.5	32.1
CNN+LDA	58.1	64.6	66.2	62.8	60.7	63.6	56.4	57.9	46.4	44.2
FIP [28]+LDA	72.9	94.3	91.4	90.0	78.9	82.5	66.1	62.0	49.3	42.5
RL [28]+LDA	70.8	94.3	90.5	89.8	77.5	80.0	63.6	59.5	44.6	38.9
MTL+RL+LDA	74.8	93.8	91.7	89.6	80.1	83.3	70.4	63.8	51.5	50.2
$\text{MVP}_{\mathbf{h}_1^{id}}+\text{LDA}$	61.5	92.5	85.4	84.9	64.3	67.0	51.6	45.4	35.1	28.3
$\text{MVP}_{\mathbf{h}_2^{id}}+\text{LDA}$	79.3	95.7	93.3	92.2	83.4	83.9	75.2	70.6	60.2	60.0
$\text{MVP}_{\mathbf{h}_3^r}+\text{LDA}$	72.6	91.0	86.7	84.1	74.6	74.2	68.5	63.8	55.7	56.0
$\text{MVP}_{\mathbf{h}_4^r}+\text{LDA}$	62.3	83.4	77.3	73.1	62.0	63.9	57.3	53.2	44.4	46.9

Face recognition accuracies across views and illuminations on the Multi-PIE dataset. The first and the second best performances are in bold.

- [1] T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: Application to face recognition. *TPAMI*, 28:2037–2041, 2006.
- [6] Dong Chen, Xudong Cao, Fang Wen, and Jian Sun. Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification. In *CVPR*, 2013.
- [28] Z. Zhu, P. Luo, X. Wang, and X. Tang. Deep learning identity preserving face space. In *ICCV*, 2013.

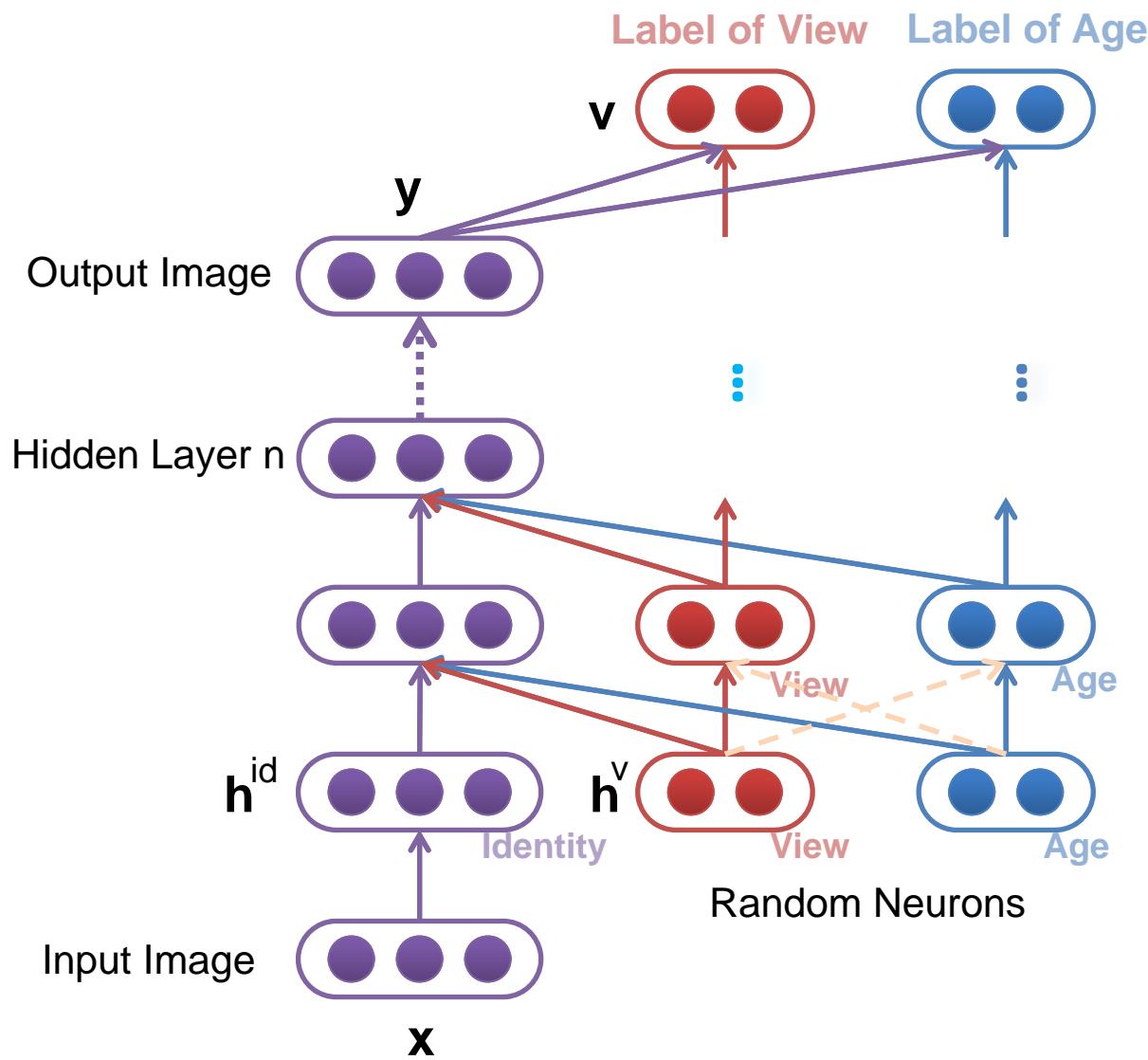
Deep Learning Multi-view Representation from 2D Images

- Interpolate and predict images under viewpoints unobserved in the training set



The training set only has viewpoints of 0° , 30° , and 60° . (a): the reconstructed images under 15° and 45° when the input is taken under 0° . (b) The input images are under 15° and 45° .

Generalize to other facial factors



Tips

- Apply deep learning to new applications
- Bridge the connection between conventional pattern recognition systems and deep models, and get ideas from domain applications to propose new deep models and training strategies
- Understand why deep learning works, get insights and generalize those insights – have your own philosophy on deep learning
- Many neural networks were proposed in 1980s and 1990s and they can be revisited

Tips

- Many machine learning models were motivated by computer vision applications. However, computer vision did not have close interaction with neural networks in the past 15 years. We expect fast development of deep learning driven by applications.
- The most successful deep model in computer vision is CNN. The two most important operations in CNN, i.e. filtering and pooling, were also widely used in vision systems. We expect other effective domain knowledge, such more advanced pooling operations which are also robust to rotation and scaling, can be incorporated into deep models.

Tips

- Study the properties of neurons, which may provide the directions of theoretical studies on deep learning. Study the difference and similarity between the mechanisms of neural networks and human brains

Optimization for Training Deep Models

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

March 22, 2015

Outline

- 1 Optimization Basics
- 2 Optimization of training deep neural networks
- 3 Multi-GPU Training

Training neural networks

- Minimize the cost function on the training set

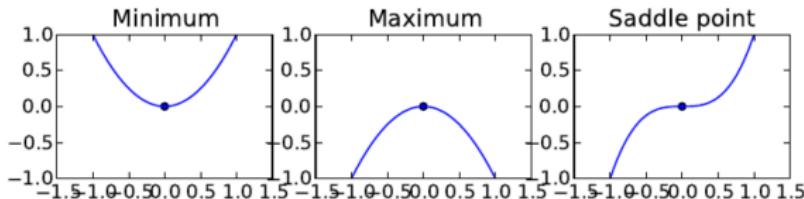
$$\theta^* = \arg \min_{\theta} J(\mathbf{X}^{(\text{train})}, \theta)$$

- Gradient descent

$$\theta = \theta - \eta \nabla J(\theta)$$

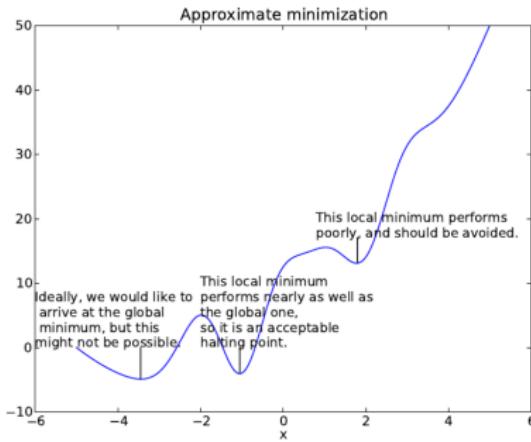
Local minimum, local maximum, and saddle points

- When $\nabla J(\theta) = 0$, the gradient provides no information about which direction to move
- Points at $\nabla J(\theta) = 0$ are known as *critical points* or *stationary points*
- A local minimum is a point where $J(\theta)$ is lower than at all neighboring points, so it is no longer possible to decrease $J(\theta)$ by making infinitesimal steps
- A local maximum is a point where $J(\theta)$ is higher than at all neighboring points, so it is no longer possible to increase $J(\theta)$ by making infinitesimal steps
- Some critical points are neither maxima nor minima. These are known as *saddle points*



Local minimum, local maximum, and saddle points

- In the context of deep learning, we optimize functions that may have many local minima that are not optimal, and many saddle points surrounded by very flat regions. All of this makes optimization very difficult, especially when the input to the function is multidimensional.
- We therefore usually settle for finding a value of J that is very low, but not necessarily minimal in any formal sense.



Jacobian matrix and Hessian matrix

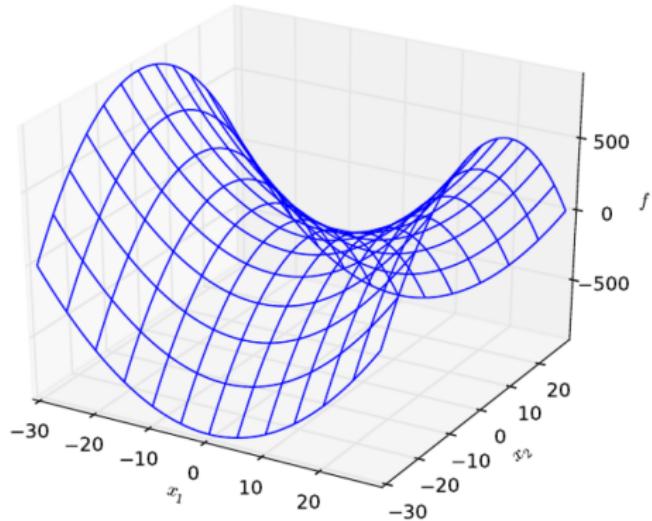
- Jacobian matrix contains all of the partial derivatives of all the elements of a vector-valued function
- Function $\mathbf{f} : \mathcal{R}^m \rightarrow \mathcal{R}^n$, then the Jacobian matrix $\mathbf{J} \in \mathcal{R}^{n \times m}$ of \mathbf{f} is defined such that $J_{i,j} = \frac{\partial}{\partial x_j} f(\mathbf{x})_i$
- The second derivative $\frac{\partial^2}{\partial x_i \partial x_j} f$ tells us how the first derivative will change as we vary the input. It is useful for determining whether a critical point is a local maximum, local minimum, or saddle point.
 - $f'(x) = 0$ and $f''(x) > 0$: local minimum
 - $f'(x) = 0$ and $f''(x) < 0$: local maximum
 - $f'(x) = 0$ and $f''(x) = 0$: saddle point or a part of a flat region
- Hessian matrix contains all of the second derivatives of the function

$$\mathbf{H}(f)(\mathbf{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x})$$

Jacobian matrix and Hessian matrix

- At a critical point, $\nabla f(\mathbf{x}) = 0$, we can examine the eigenvalues of the Hessian to determine whether the critical point is a local maximum, local minimum, or saddle point
 - When the Hessian is positive definite (all its eigenvalues are positive), the point is a local minimum: the directional second derivative in any direction must be positive
 - When the Hessian is negative definite (all its eigenvalues are negative), the point is a local maximum
 - Saddle point: at least one eigenvalue is positive and at least one eigenvalue is negative. \mathbf{x} is a local maximum on one cross section of f but a local minimum on another cross section.

Jacobian matrix and Hessian matrix



Hessian matrix

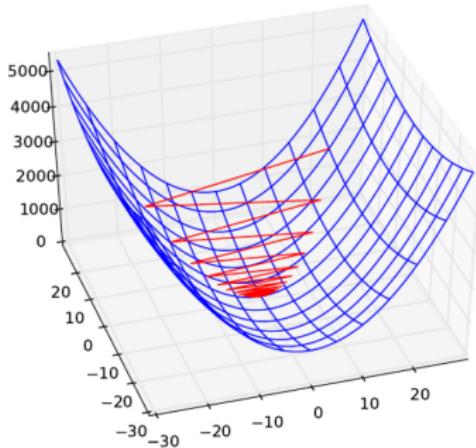
- Condition number: consider the function $f(\mathbf{x}) = \mathbf{A}^{-1}\mathbf{x}$. When $\mathbf{A} \in \mathcal{R}^{n \times n}$ has an eigenvalue decomposition, its condition number

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$$

i.e. the ratio of the magnitude of the largest and smallest eigenvalue. When this number is large, matrix inversion is particularly sensitive to error in the input

- The Hessian can also be useful for understanding the performance of gradient descent. When the Hessian has a poor condition number, gradient descent performs poorly. This is because in one direction, the derivative increases rapidly, while in another direction, it increases slowly. Gradient descent is unaware of this change in the derivative so it does not know that it needs to explore preferentially in the direction where the derivative remains negative for longer.

Hessian matrix



Gradient descent fails to exploit the curvature information contained in Hessian. Here we use gradient descent on a quadratic function whose Hessian matrix has condition number 5. The red lines indicate the path followed by gradient descent. This very elongated quadratic function resembles a long canyon. Gradient descent wastes time repeatedly descending canyon walls, because they are the steepest feature. Because the step size is somewhat too large, it has a tendency to overshoot the bottom of the function and thus needs to descend the opposite canyon wall on the next iteration. The large positive eigenvalue of the Hessian corresponding to the eigenvector pointed in this direction indicates that this directional derivative is rapidly increasing, so an optimization algorithm based on the Hessian could predict that the steepest direction is not actually a promising search direction in this context.

Second-order optimization methods

- Gradient descent uses only the gradient and is called first-order optimization. Optimization algorithms such as Newton's method that also use the Hessian matrix are called second-order optimization algorithms.
- Update with Newton's method

$$\mathbf{x}^* = \mathbf{x}_0 - H(f)(\mathbf{x}_0)^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}_0)$$

When the function can be locally approximated as quadratic, iteratively updating the approximation and jumping to the minimum of the approximation can reach the critical point much faster than gradient descent would.

- In many other fields, the dominant approach to optimization is to design optimization algorithms for a limited family of functions.
- The family of functions used in deep learning is quite complicated and complex

Stochastic gradient descent

- Given n training samples, our target function can be expressed as

$$J(\mathbf{w}) = \sum_{p=1}^n J_p(\mathbf{w})$$

- Batch gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{p=1}^n \nabla J_p(\mathbf{w})$$

- In some cases, evaluating the sum-gradient may be computationally expensive. Stochastic gradient descent samples a subset of summand functions at every step. This is very effective in the case of large-scale machine learning problems. In stochastic gradient descent, the true gradient of $J(\mathbf{w})$ is approximated by a gradient at a single example (or a mini-batch of samples):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J_p(\mathbf{w})$$

Stochastic backpropagation

Algorithm 1 (Stochastic backpropagation)

```
1 begin initialize network topology (# hidden units),  $\mathbf{w}$ , criterion  $\theta$ ,  $\eta$ ,  $m \leftarrow 0$ 
2   do  $m \leftarrow m + 1$ 
3      $\mathbf{x}^m \leftarrow$  randomly chosen pattern
4      $w_{ij} \leftarrow w_{ij} + \eta \delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta \delta_k y_j$ 
5   until  $\nabla J(\mathbf{w}) < \theta$ 
6   return  $\mathbf{w}$ 
7 end
```

- In stochastic training, a weight update may reduce the error on the single pattern being presented, yet increase the error on the full training set.

Mini-batch based stochastic gradient descent

- Divide the training set into mini-batches.
- In each epoch, randomly permute mini-batches and take a mini-batch sequentially to approximate the gradient
 - One epoch corresponds to a single presentations of all patterns in the training set
- The estimated gradient at each iteration is more reliable
- Start with a small batch size and increase the size as training proceeds

Batch backpropagation

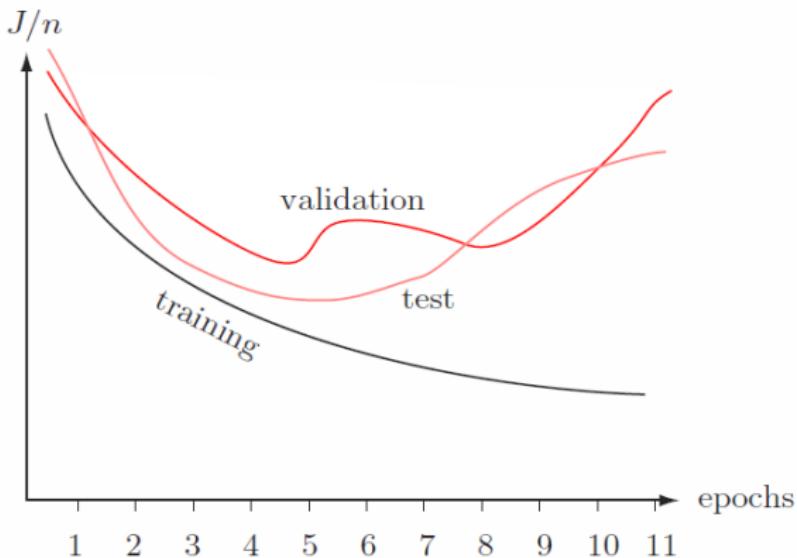
Algorithm 2 (Batch backpropagation)

```
1 begin initialize network topology (# hidden units),  $\mathbf{w}$ , criterion  $\theta, \eta, r \leftarrow 0$ 
2   do  $r \leftarrow r + 1$  (increment epoch)
3      $m \leftarrow 0; \Delta w_{ij} \leftarrow 0; \Delta w_{jk} \leftarrow 0$ 
4     do  $m \leftarrow m + 1$ 
5        $\mathbf{x}^m \leftarrow$  select pattern
6        $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i; \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$ 
7     until  $m = n$ 
8      $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}; w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$ 
9   until  $\nabla J(\mathbf{w}) < \theta$ 
10  return  $\mathbf{w}$ 
11 end
```

Summary

- Stochastic learning
 - Estimate of the gradient is noisy, and the weights may not move precisely down the gradient at each iteration
 - Faster than batch learning, especially when training data has redundancy
 - Noise often results in better solutions
 - The weights fluctuate and it may not fully converge to a local minimum
- Batch learning
 - Conditions of convergence are well understood
 - Some acceleration techniques only operate in batch learning
 - Theoretical analysis of the weight dynamics and convergence rates are simpler

Plot learning curves on the training and validation sets



Plot the average error per pattern (i.e. $1/n \sum_p J_p$) versus the number of epochs.

Learning curve on the training set

- The average training error typically decreases with the number of epochs and reaches an asymptotic value
- This asymptotic value could be high if **underfitting** happens. The reasons could be
 - The classification problem is difficult (Bayes error is high) and there are a large number of training samples
 - The expressive power of the network is not enough (the numbers of weights, layers and nodes in each layer)
 - Bad initialization and get stuck at local minimum (pre-training for better initialization)
- If the learning rate is low, the training error tends to decrease monotonically, but converges slowly. If the learning rate is high, the training error may oscillate.

Learning curve on the test and validation set

- The average error on the validation or test set is virtually always higher than on the training set. It could increase or oscillate when **overfitting** happen. The reasons could be
 - Training samples are not enough
 - The expressive power of the network is too high
 - Bad initialization and get stuck at local minimum (pre-training for better initialization)
- Stop training at a minimum of the error on the validation set

Data augmentation

- If the training set is small, one can synthesize some training samples by adding Gaussian noise to real training samples
- Domain knowledge can be used to synthesize training samples. For example, in image classification, more training images can be synthesized by translation, scaling, and rotation.

Normalizing input

- If the dynamic range of one input feature is much larger than others, during training, the network will mainly adjust weights on this feature while ignore others
- We do not want to prefer one feature over others just because they differ solely measured units
- To avoid such difficulty, the input patterns should be shifted so that the average over the training set of each feature is zero, and then be scaled to have the same variance as 1 in each feature
- Input variables should be uncorrelated if possible
 - If inputs are uncorrelated then it is possible to solve for the value of one weight without any concern for other weights
 - With correlated inputs, one must solve for multiple weights simultaneously, which is a much harder problem
 - PCA can be used to remove linear correlations in inputs

Shuffling the training samples

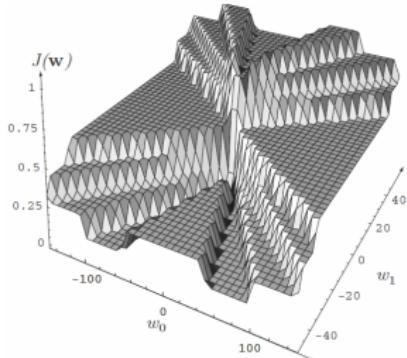
- Networks learn the fastest from the most unexpected sample
- Shuffle the training set so that successive training examples never (rarely) belong to the same class
- Present input examples that produce a large error more frequently than examples that produce a small error
 - This technique applied to data containing outliers can be disastrous because outliers can produce large errors yet should not be presented frequently

Dropout

- Randomly set some input features and the outputs of hidden units as zero during the training process
- Feature co-adaptation: a feature is only helpful when other specific features are present
 - Because of the existence of noise and data corruption, some features or the responses of hidden nodes can be misdetected
- Dropout prevents feature co-adaptation and can significantly improve the generalization of the trained network
- Can be considered as another approach to regularization
- It can be viewed as averaging over many neural networks
- Slower convergence

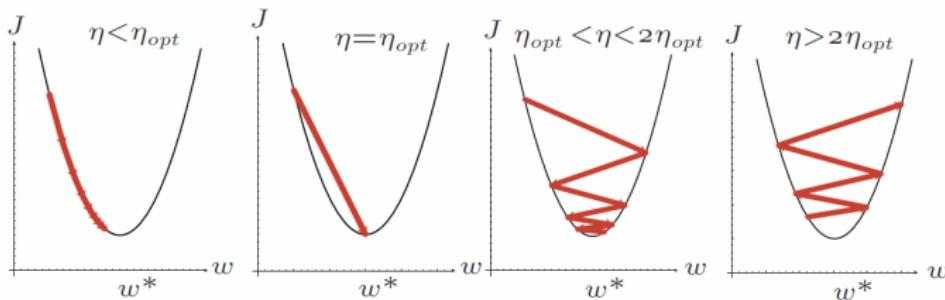
Error surfaces

- Backpropagation is based on gradient descent and tries to find the minimum point of the error surface $J(\mathbf{w})$
- Generally speaking, it is unlikely to find the global minimum since the error surface is usually very complex
- Backpropagation stops at local minimum and plateaus (regions where error varies only slightly as a function of weights)
- Therefore, it is important to find a good initialization for backpropagation (through pre-training)



Learning rate

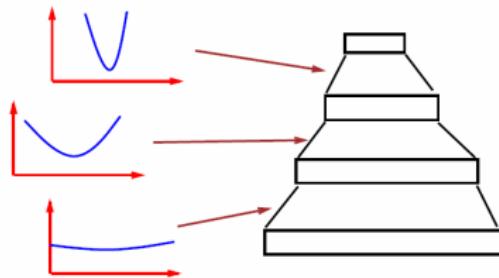
- Decrease the learning rate when the weight vector “oscillates” and increase it when the weight vector follows a steady direction
- One can choose a different learning rate for each weights, so that all the weights in the network converge roughly at the same speed



Gradient descent in a 1D quadratic criterion with different learning rates. The optimal learning rate is found by $\eta_{opt} = \left(\frac{\partial^2 J}{\partial^2 w^2} \right)^{-1}$.

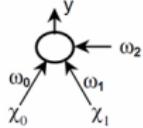
Learning rate

- Learning rates in the lower layers should generally be larger than in the higher layers, since the second derivative is often smaller in the lower layers

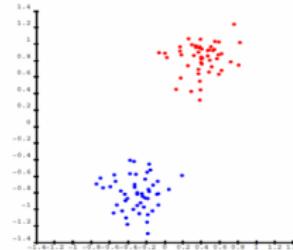


Learning rate

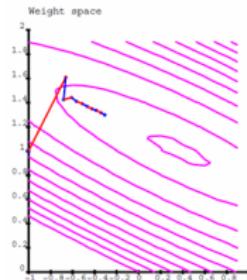
- Example of linear network trained in a batch mode.



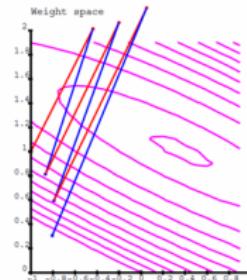
(a)



(b)



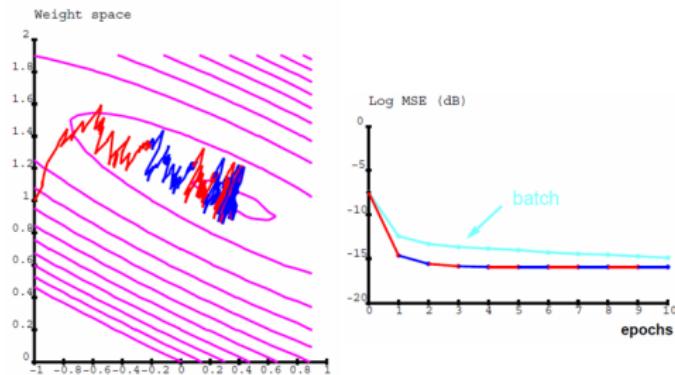
(c) $\eta = 1.5$



(d) $\eta = 2.5$

Learning rate

- Stochastic learning with $\eta = 0.2$



Incorporation of momentum

- Error surfaces often have plateaus where there are “to many” weights (especially when the number of layers is large) and thus the error depends only weakly upon any one of them.
- Include some fraction α of the previous weight update in stochastic backpropagation

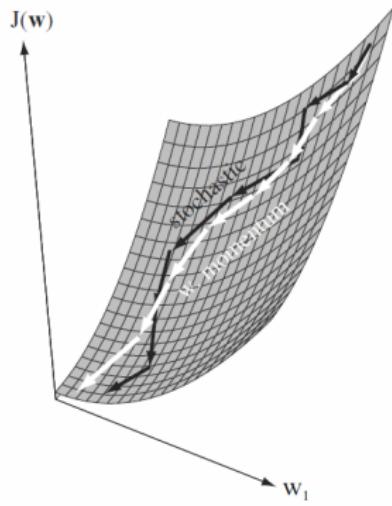
$$\mathbf{w}(m + 1) = \mathbf{w}(m) + (1 - \alpha)\Delta\mathbf{w}_{bp}(m) + \alpha\Delta\mathbf{w}(m - 1)$$

where $\Delta\mathbf{w}_{bp}(m)$ is the change in $\mathbf{w}(m)$ that would be called for by the backpropagation algorithm

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

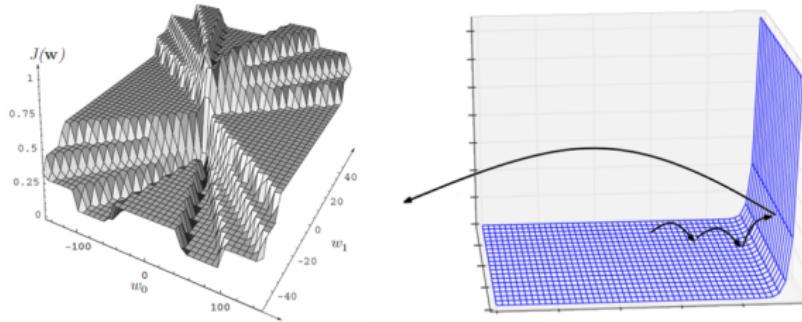
- Allow the network to learn more quickly when plateaus in the error surface exists

Incorporation of momentum



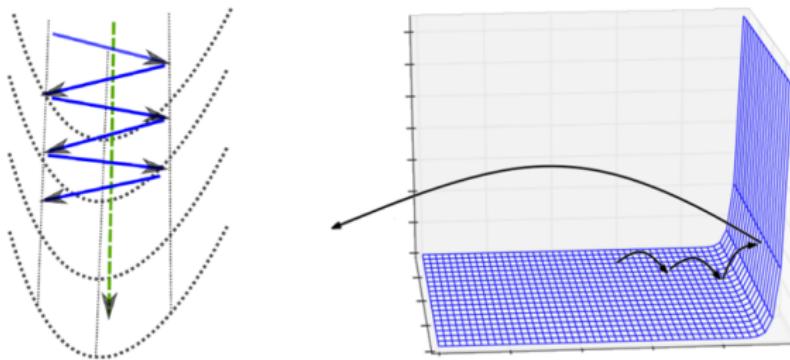
Plateaus and cliffs

- The error surfaces of training deep neural networks include local minima, plateaus (regions where error varies only slightly as a function of weights), and cliffs (regions where the gradients rise sharply)
- Plateaus and cliffs are more important barriers to training neural networks than local minima
 - It is very difficult (or slow) to effectively update the parameters in plateaus
 - When the parameters approach a cliff region, the gradient update step can move the learner towards a very bad configuration, ruining much progress made during recent training iterations.



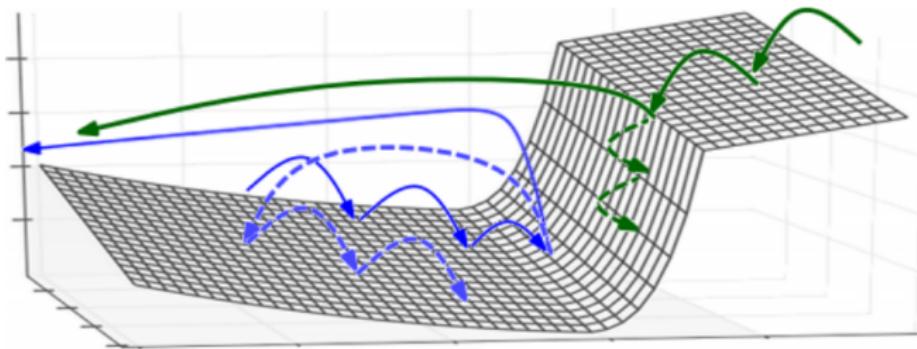
Higher-order nonlinearities

- Second-order methods or momentum assume quadratic shape around the minimum. They increase the size of steps in the low-curvature directions and decrease the sizes of steps in the high-curvature directions (the steep sides of the valley)
- When training deep models, higher order derivatives introduce a lot more non-linearity, which often does not have the nice symmetrical shapes that the second-order “valley” picture builds in our mind



Gradient clipping

- To address the presence of cliffs, a useful heuristic is to clip the magnitude of the gradient, only keeping its direction if its magnitude is below a threshold (which is a hyper-parameter). This helps to avoid the destructive big moves which would happen when approaching the cliff, either from above or below.



Vanishing and exploding gradients

- Training a very deep net makes the problem even more serious, since after BP through many layers, the gradients become either very small or very large
- In very deep nets and recurrent nets, the final output is composed of a large number of non-linear transformations
- Even though each of these non-linear stages may be relatively smooth, their composition is going to be much “more non-linear”, in the sense that the derivatives through the whole composition will tend to be either very small or very large, with more ups and downs



When composing many non-linearities (like the activation non-linearity in a deep or recurrent neural network), the result is highly non-linear, typically with most of the values associated with a tiny derivative, some values with a large derivative, and many ups and downs (not shown here)

Vanishing and exploding gradients

This arises because the Jacobian (matrix of derivatives) of a composition is the product of the Jacobian of each stage, i.e. if

$$f = f_T \circ f_{T-1} \circ \dots f_2 \circ f_1$$

The Jacobian matrix of derivatives of $f(x)$ with respect to its input vector \mathbf{x} is

$$f' = f'_T f'_{T-1} \dots f'_2 f'_1$$

where

$$f' = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$$

and

$$f'_t = \frac{\partial f_t(\alpha_t)}{\partial \alpha_t}$$

where $\alpha_t = f_{t-1}(f_{t-1}(\dots f_2(f_1(\mathbf{x}))))$, i.e. composition has been replaced by matrix multiplication

Vanishing and exploding gradients

- In the scalar case, we can imagine that multiplying many numbers together tends to be either very large or very small
- In the special case where all the numbers in the product have the same value α , this is obvious, since α^T goes to 0 if $\alpha < 1$ and to ∞ if $\alpha > 1$ as T increases
- The more general case of non-identical numbers be understood by taking the logarithm of these numbers, considering them to be random, and computing the variance of the sum of these logarithms. Although some cancellation can happen, the variance grows with T . If those numbers are independent, it grows linearly with T , which means that the product grows roughly as e^T .
- This analysis can be generalized to the case of multiplying square matrices

Why need multi-GPU?

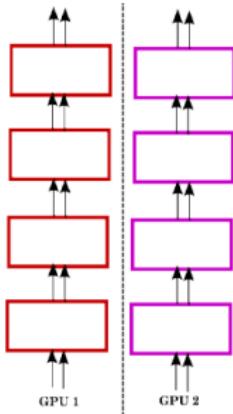
- Further speed-up
- The memory size of a single GPU is limited
 - GeForce GTX 670: 2GB
 - TITAN: 6GB
 - TITAN X: 12GB
 - Tesla K40: 12GB
 - Tesla K80: two K40
- Train bigger models
- Data parallelism
- Model parallelism

Cost of using multi-GPU

- Synchronization
- Communication overhead
 - Communication between GPUs in the same server
 - Communication between GPU servers

Data parallelism

- The mini-batch is split across several GPUs. Each GPU is responsible computing gradients with respect to all model parameters, but does so using a subset of the samples in the mini-batch
- The model (parameters) has a complete (same) copy in each GPU
- The gradients computed from multiple GPUs are averaged to update parameters in both GPUs

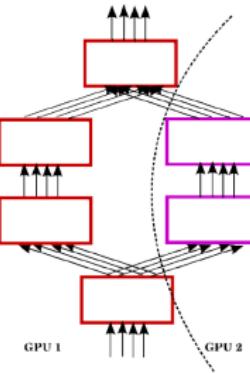


Drawbacks of data parallelism

- Require considerable communication between GPUs, since each GPU must communicate both gradients and parameter values on every update step
- Each GPU must use a large number of samples to effectively utilize the highly parallel device; thus, the mini-batch size effectively gets multiplied by the number of GPUs, hampering convergence

Model parallelism

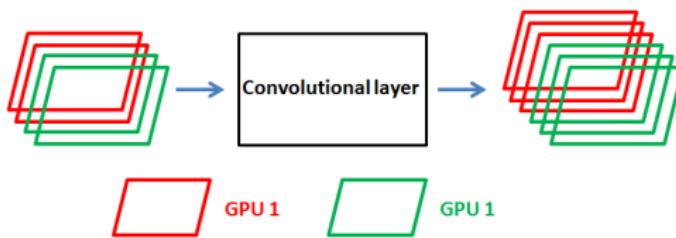
- Consist of splitting an individual network's computation across multiple GPUs
- For instance, convolutional layer with N filters can be run on two GPUs, each of which convolves its input with $N/2$ filters



The architecture is split into two columns which make easier to split computation across the two GPUs

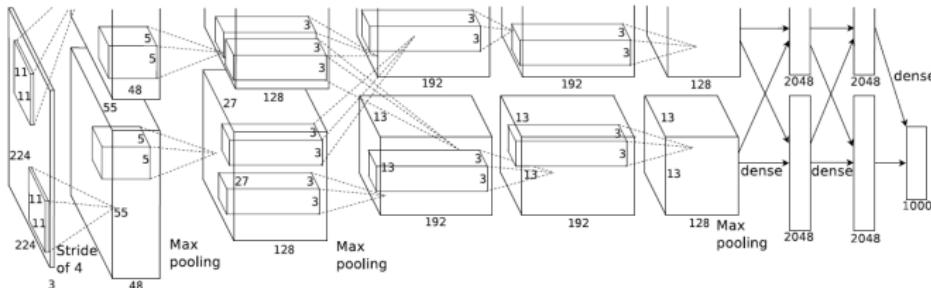
Model parallelism

- A mini batch has the same copy in each GPU
- GPUs have to be synchronized and communicate at every layer if computing gradients in a GPU requires outputs of all the feature maps at the lower layer



Model parallelism

- Krizhevsky et al. customized the architecture of the network to better leverage model parallelism: the architecture consists of two “columns” each allocated on one GPU
- Columns have cross connections only at one intermediate layer and at the very top fully connected layers



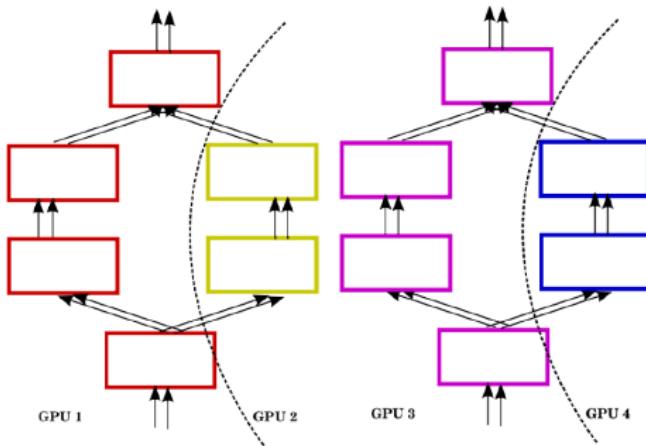
A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” in NIPS, 2012.

Model parallelism

- While model parallelism is more difficult to implement, it has two potential advantages relative to data parallelism
 - It may require less communication bandwidth when the cross connections involve small intermediate feature maps
 - It allows the instantiation of models that are too big for a single GPU's memory

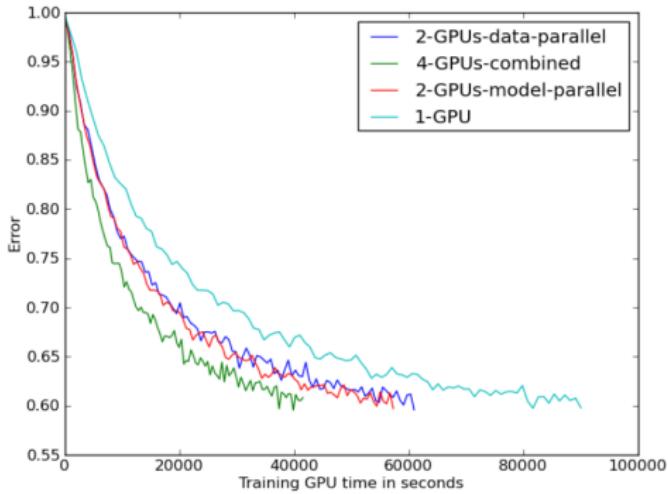
Hybrid data and model parallelism

- Data and model parallelism can be hybridized.



Examples of how model and data parallelism can be combined in order to make effective use of 4 GPUs

Hybrid data and model parallelism



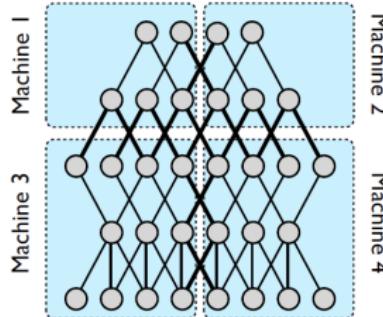
Test error on ImageNet a function of time using different forms of parallelism. All experiments used the same mini-batch size (256) and ran for 100 epochs (here showing only the first 10 for clarity of visualization) with the same architecture and the same hyper-parameter setting as in Alex net. If plotted against number of weight updates, all these curves would almost perfectly coincide.

Hybrid data and model parallelism

Configuration	Time to complete 100 epochs
1 GPU	10.5 days
2 GPUs Model parallelism	6.6 days
2 GPUs Data parallelism	7 days
4 GPUs Data parallelism	7.2 days
4 GPUs model + data parallelism	4.8 days

Distributed computation with CPU cores

- Model parallelism: Only those nodes with edges that cross partition boundaries will need to have their state transmitted between machines. Even in cases where a node has multiple edges crossing a partition boundary, its state is only sent to the machine on the other side of that boundary once.
- Within each partition, computation for individual nodes will be parallelized across all available CPU cores
- It requires data synchronization and data transfer between machines during both training and inference



Distributed computation with CPU cores

- Models with local connectivity structures tend to be more amendable to extensive distribution than fully-connected structures, given their lower communication requirements
- Models with a large number of parameters or high computational demands typically benefit from access to more CPUs and memory, up to the point where communication costs dominate
- It means that the speedup cannot keep increasing with infinite number of machines
- The typical cause of less-than-ideal speedup is variance in processing times across the different machines, leading to many machines waiting for the single slowest machine to finish a given phase of computation

Reading Materials

- R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern Classification," Chapter 6, 2000.
- Y. LeCun, L. Bottou, G. B. Orr, and K. Muller, "Efficient BackProp," Technical Report, 1998.
- Y. Bengio, I. J. GoodFellow and A. Courville, "Numerical Computation" in "Deep Learning", Book in preparation for MIT Press
- Y. Bengio, I. J. GoodFellow and A. Courville, "Numerical Optimization" in "Deep Learning", Book in preparation for MIT Press
- O. Yadan, K. Adams, Y. Taigman, and M. Ranzato, "Multi-GPU Training of ConvNets", arXiv:1312.583, 2014
- J. Dean, G. S. Corrado, R. Monga, and K. Chen, "Large Scale Distributed Deep Networks," NIPS 2012

Understand Deep Learning

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

March 29, 2015

Outline

- 1 Distributed representation and disentangling underlying factors
- 2 Better mixing via deep representations

What is a good representation?

- Makes the further learning easier
 - The joint distribution of different elements of the representation \mathbf{h} is easy to model (e.g. they are mutually independent)
 - The representation keeps the information (or at least all the relevant information in the supervised case) and makes it easy to learn functions of interest from this representation
- An ideal representation is one that disentangles the underlying causal factors of variation that generated the observed data
- Once we “understand” the underlying explanations for what we observe, it generally becomes easy to predict one thing from others

When does unsupervised learning help supervised learning?

- Whether unsupervised learning on input variables \mathbf{x} can yield representations that are useful when later trying to learn to predict some target variable \mathbf{y} , given \mathbf{x}
- Whether $P(\mathbf{y}|\mathbf{x})$ seen as a function of has anything \mathbf{x} to do with $P(\mathbf{x})$
 - If $P(\mathbf{x})$ is uniformly distributed and $E[\mathbf{y}|\mathbf{x}]$ is some function of interest, observing \mathbf{x} alone gives us no information about $P(\mathbf{y}|\mathbf{x})$
 - If \mathbf{x} arises from a mixture, with one component per value of \mathbf{y} , and the mixture components are well separated, then modeling $P(\mathbf{x})$ tells us precisely where each component is, and a single labeled example of each component will then be enough to perfectly learn $P(\mathbf{y}|\mathbf{x})$.
- If \mathbf{y} is closely associated with one of the causal factors of \mathbf{x} , $P(\mathbf{x})$ and $P(\mathbf{y}|\mathbf{x})$ will be strongly tied, and unsupervised representation learning that tries to disentangle the underlying factors of variation is likely to be useful as a semi-supervised learning strategy.

Disentangle the underlying factors of variation

- Assuming that \mathbf{y} is one of the causal factors of \mathbf{x} , and let \mathbf{h} represent all those factors, then the data has marginal probability

$$P(\mathbf{x}) = \int_{\mathbf{h}} P(\mathbf{x}|\mathbf{h})p(\mathbf{h})d\mathbf{h}$$

or, in the discrete case

$$P(\mathbf{x}) = \sum_{\mathbf{h}} P(\mathbf{x}|\mathbf{h})P(\mathbf{h})$$

- The best possible model \mathbf{x} is the one that uncovers the above “true” structures, with \mathbf{h} as a latent variable that explains the observed variations in \mathbf{x}
- The “ideal” representation learned should recover these latent factors
- If \mathbf{y} is one of them (or closely related to one of them), then it will be very easy to learn to predict \mathbf{y} from such a representation
- Not knowing which of the factors in \mathbf{h} will be the one of interest, say $\mathbf{y} = \mathbf{h}_i$, an unsupervised learning should learn a representation that disentangles all the generative factors \mathbf{h} , from each other, then making it easy to predict \mathbf{y} from \mathbf{h} .

Why is generative model more robust to discriminative model?

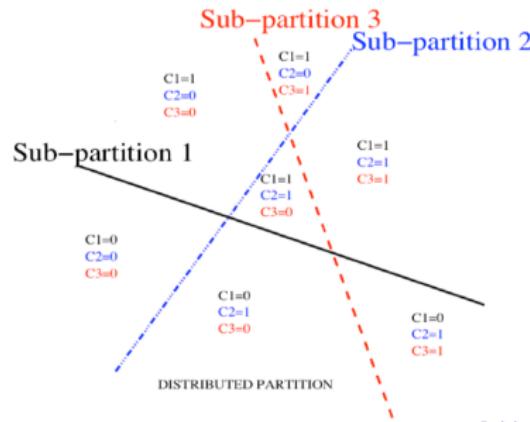
- Assuming \mathbf{x} is an effect and \mathbf{y} is a cause, the generative model learns $P(\mathbf{x}|\mathbf{y})$ and estimate $P(\mathbf{y}|\mathbf{x})$ with the Bayes rule

$$P(\mathbf{y}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{P(\mathbf{x})}$$

- The discriminative model directly learns $P(\mathbf{y}|\mathbf{x})$
- $P(\mathbf{x}|\mathbf{y})$ is more robust to changes in $P(\mathbf{y})$. To different domains, the causal mechanisms remain invariant ("the laws of the universe are constant") whereas what changes are the marginal distribution over the underlying causes (or what factors are linked to our particular task).
- If the cause-effect relationship was reversed, it would not be true, since by Bayes rule, $P(\mathbf{y}|\mathbf{x})$ would be sensitive to changes in $P(\mathbf{x})$.
- Hence, better generalization and robustness to all kinds of changes can be expected via learning a generative model that attempts to recover the causal factors \mathbf{h} and $P(\mathbf{x}|\mathbf{h})$

Distributed representation

- The core assumption behind most neural network and deep learning research relies on the notion of distributed representation
- A distributed representation can express an exponentially large number of concepts by allowing to compose the activation of many features
- An example of distributed representation is a vector of n binary features, which can take 2^n configurations, each potentially corresponding to a different region in input space



Distributed representation

- The way these regions carve the input space still depends on few parameters: this huge number of regions are not placed independently of each other
- We can thus represent a function that looks complicated but actually has structure
- The assumption is that one can learn about each feature without having to see the examples for all the configurations of all the other features, i.e., these features correspond to underlying factors explaining the data

Non-distributed representation

- Example of symbolic representation: the input is associated with a single symbol or category; only n different configurations of the representation-space are possible, carving n different regions in input space. Such a symbolic representation is also called a one-hot representation, since it can be captured by a binary vector with n bits that are mutually exclusive (only one of them can be active)
- It breaks up the input space into regions, with a separate set of parameters for each region



Examples

- Clustering methods, including the k-means algorithm: only one cluster “wins” the competition
- K-nearest neighbors algorithms: only one template or prototype example is associated with a given input
- Decision trees: only one leaf (and the nodes on the path from root to leaf) is activated when an input is given
- Gaussian mixtures and mixtures of experts: the templates (cluster centers) or experts are now associated with a degree of activation
- Kernel machines with a Gaussian kernel (or other similarly local kernel): the degree of activation of each “support vector” or template example is continuous-valued
- Disadvantages: there is no generalization to new regions, except by extending the answer for which there is data, exploiting solely a smoothness prior; it makes it difficult to learn a complicated function, with more ups and downs than the available number of examples.

Distributed representation leads to better generalization

- Generalization arises due to shared attributes between different concepts
- As pure symbols, “cat” and “dog” are as far from each other as any other two symbols. However, if one associates them with a meaningful distributed representation, then many of the things that can be said about cats can generalize to dogs and vice versa.
- Distributed representations induce a rich similarity space, in which semantically close concepts (or inputs) are close in distance, a property that is absent from purely symbolic representations

Exponential gain in representational efficiency from distributed representations

- A function that “looks complicated” can be compactly represented using a small number of parameters, if some “structure” is uncovered by the learner
- Traditional “non-distributed” learning algorithms generalize only thanks to the smoothness assumption, which states that if $u \approx v$, then the target function f to be learned has the property that $f(u) \approx f(v)$, in general.
- This assumption suffers from the curse of dimensionality: in order to learn a target function that takes many different values (e.g. many ups and downs) in a large number of regions, we may need a number of examples that is at least as large as the number of distinguishable regions.
 - e.g., exponentially many regions: in a d -dimensional space with at least 2 different values to distinguish per dimension, we might want f to differ in 2^d different regions, requiring $O(2^d)$ training examples

Exponential gain in representational efficiency from distributed representations

- One can think of each of these regions as a category or symbol: by having a separate degree of freedom for each symbol (or region), we can learn an arbitrary mapping from symbol to value. However, this does not allow us to generalize to new symbols, new regions.
- If we are lucky, there may be some regularity in the target function, besides being smooth. For example, the same pattern of variation may repeat itself many times (e.g., as in a periodic function or a checkerboard). If we only use the smoothness prior, we will need additional examples for each repetition of that pattern
- A deep architecture could represent and discover such a repetition pattern and generalize to new instances of it. Thus a small number of parameters (and therefore, a small number of examples) could suffice to represent a function that looks complicated (in the sense that it would be expensive to represent with a non-distributed architecture).

Exponential gain in representational efficiency from distributed representations

- How many regions are generated by an arrangement of n hyperplanes in \mathcal{R}^d ?
- This corresponds to the number of regions that a shallow neural network (one hidden layer) can distinguish
- Therefore, we see a growth that is exponential in the input size and polynomial in the number of hidden units

$$\sum_{j=0}^d \binom{n}{j} = O(n^d)$$

- Distributed representation has better generalization because we can learn about the location of each hyperplane with only $O(d)$ examples: we do not need to see examples corresponding to all $O(n^d)$ regions

Exponential gain in representational efficiency from distributed representations

- It is hard to detect factors such as gender, age, and the presence of glasses with a linear classifiers, i.e. a shallow neural network
- The kinds of factors that can be chosen almost independently in order to generate data are more likely to be very high-level and related in highly non-linear ways to the input
- This demands deep distributed representations, where the higher level features (seen as functions of the input) or factors (seen as generative causes) are obtained through the composition of many non-linearities

Exponential gain in representational efficiency from distributed representations

- Organizing computation through the composition of many nonlinearities and a hierarchy of re-used features can give another exponential boost to statistical efficiency
- Although 2-layer networks (e.g., with saturating non-linearities, boolean gates, sum/products, or RBF units) can generally be shown to be universal approximators , the required number of hidden units may be very large
- The main results on the expressive power of deep architectures state that there are families of functions that can be represented efficiently with a deep architecture (say depth k) but would require an exponential number of components (with respect to the input size) with insufficient depth (depth 2 or depth $k - 1$)

Example of sum-product network

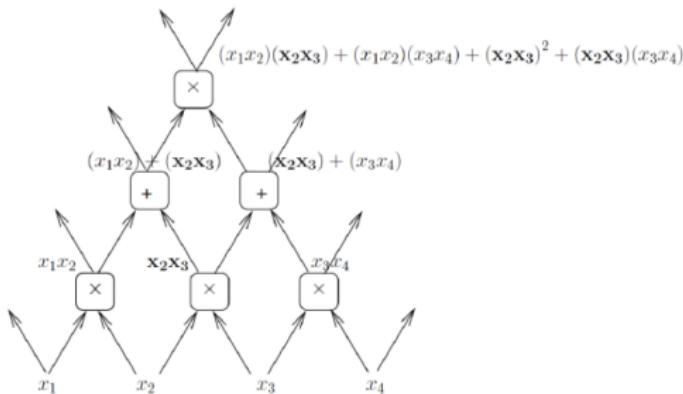


Figure 14.3: A sum-product network (Poon and Domingos, 2011) composes summing units and product units, so that each node computes a polynomial. Consider the product node computing x_2x_3 : its value is re-used in its two immediate children, and indirectly incorporated in its grand-children. In particular, in the top node shown the product x_2x_3 would arise 4 times if that node's polynomial was expanded as a sum of products. That number could double for each additional layer. In general a deep sum of product can represent polynomials with a number of min-terms that is exponential in depth, and some families of polynomials are represented efficiently with a deep sum-product network but not efficiently representable with a simple sum of products, i.e., a 2-layer network (Delalleau and Bengio, 2011).

Example of sum-product network

- More recently, Delalleau and Bengio (2011) showed that a shallow network requires exponentially many more sum-product hidden units than a deep sum-product network (Poon and Domingos, 2011) in order to compute certain families of polynomials.

Example of deep rectifier networks

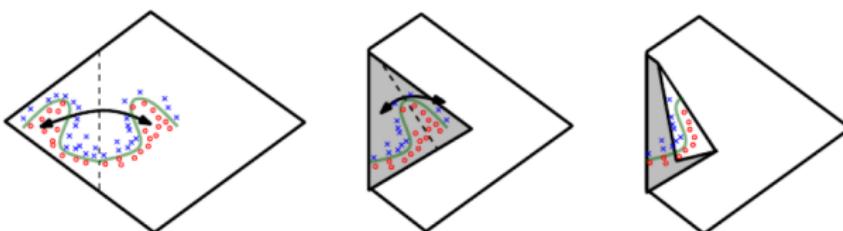


Figure 14.4: An absolute value rectification unit has the same output for every pair of mirror points in its input. The mirror axis of symmetry is given by the hyperplane defined by the weights and bias of the unit. If one considers a function computed on top of that unit (the green decision surface), it will be formed of a mirror image of a simpler pattern, across that axis of symmetry. The middle image shows how it can be obtained by folding the space around that axis of symmetry, and the right image shows how another repeating pattern can be folded on top of it (by another downstream unit) to obtain another symmetry (which is now repeated four times, with two hidden layers). This is an intuitive explanation of the exponential advantage of deeper rectifier networks formally shown in [Pascanu et al. \(2014b\)](#); [Montufar et al. \(2014\)](#).

Example of deep rectifier networks

- (Pascanu et al., 2014b; Montufar et al., 2014) showed that piecewise linear networks (e.g. obtained from rectifier non-linearities or maxout units) could represent functions with exponentially more piecewise-linear regions, as a function of depth, compared to shallow neural networks.
- Figure illustrates how a network with absolute value rectification creates mirror images of the function computed on top of some hidden unit, with respect to the input of that hidden unit. Each hidden unit specifies where to fold the input space in order to create mirror responses (on both sides of the absolute value non-linearity). By composing these folding operations, we obtain an exponentially large number of piecewise linear regions which can capture all kinds of regular (e.g. repeating) patterns.
- The main theorem in Montufar et al. (2014) states that the number of linear regions carved out by a deep rectifier network with d inputs, depth L , and n units per hidden layer, is

$$O\left(\binom{n}{d}^{d(L-1)} n^d\right)$$

Priors regarding the underlying factors

- *No-free-lunch theorem for machine learning*: with absolutely no priors, it is not possible to generalize
- In the space of all functions, which is huge, with any finite training set, there is no general purpose learning recipe that would dominate all other learning algorithms. Whereas some assumptions are required, when our goal is to build AI or understand human intelligence, it is tempting to focus our attention on the most general and broad priors, that are relevant for most of the tasks that humans are able to successfully learn.
- **Smoothness**: we want to learn functions f s.t. $\mathbf{x} \approx \mathbf{y}$ generally implies $f(\mathbf{x}) \approx f(\mathbf{y})$. This is the most basic prior and is present in most machine learning, but is insufficient to get around the curse of dimensionality, as discussed previously
- **Multiple explanatory factors**: the data generating distribution is generated by different underlying factors, and for the most part what one learns about one factor generalizes in many configurations of the other factors. This assumption is behind the idea of distributed representations

Priors regarding the underlying factors

- **Depth, or a hierarchical organization of explanatory factors:** the concepts that are useful at describing the world around us can be defined in terms of other concepts, in a hierarchy, with more abstract concepts higher in the hierarchy, being defined in terms of less abstract ones. This is the assumption exploited by having **deep representations**
- **Causal factors:** the input variables \mathbf{x} are consequences, effects, while the explanatory factors are causes, and not vice-versa. As discussed above, this enables the **semi-supervised learning** assumption, i.e., that $P(\mathbf{x})$ is tied to $P(\mathbf{y}|\mathbf{x})$, making it possible to improve the learning of $P(\mathbf{y}|\mathbf{x})$ via the learning of $P(\mathbf{x})$. More precisely, this entails that representations that are useful for $P(\mathbf{x})$ are useful when learning $P(\mathbf{y}|\mathbf{x})$, allowing sharing of statistical strength between the unsupervised and supervised learning tasks.
- **Shared factors across tasks:** in the context where we have many tasks, corresponding to different \mathbf{y}_i 's sharing the same input \mathbf{x} or where each task is associated with a subset or a function $f_i(\mathbf{x})$ of a global input \mathbf{x} , the assumption is that each \mathbf{y}_i is associated with a different subset from a common pool of relevant factors \mathbf{h} . Because these subsets overlap, learning all the $P(\mathbf{y}_i|\mathbf{x})$ via a shared intermediate representation $P(\mathbf{h}|\mathbf{x})$ allows sharing of statistical strength between the tasks.

Priors regarding the underlying factors

- **Manifolds:** probability mass concentrates, and the regions in which it concentrates are locally connected and occupy a tiny volume. In the continuous case, these regions can be approximated by low-dimensional manifolds that a much smaller dimensionality than the original space where the data lives. This manifold hypothesis is related to auto-encoders.
- **Natural clustering:** different values of categorical variables such as object classes are associated with separate manifolds. More precisely, the local variations on the manifold tend to preserve the value of a category, and a linear interpolation between examples of different classes in general involves going through a low density region, i.e., $P(\mathbf{x}|\mathbf{y} = i)$ for different i tend to be well separated and not overlap much. This hypothesis is consistent with the idea that humans have named categories and classes because of such statistical structure (discovered by their brain and propagated by their culture), and machine learning tasks often involve predicting such categorical variables.

Priors regarding the underlying factors

- **Temporal and spatial coherence:** consecutive or spatially nearby observations tend to be associated with the same value of relevant categorical concepts, or result in a small move on the surface of the high-density manifold. More generally, different factors change at different temporal and spatial scales, and many categorical concepts of interest change slowly. When attempting to capture such categorical variables, this prior can be enforced by making the associated representations slowly changing, i.e., penalizing changes in values over time or space.
- **Sparsity:** for any given observation x , only a small fraction of the possible factors are relevant. In terms of representation, this could be represented by features that are often zero, or by the fact that most of the extracted features are insensitive to small variations of x . This can be achieved with certain forms of priors on latent variables, or by using a non-linearity whose value is often flat at 0 (i.e., 0 and with a 0 derivative), or simply by penalizing the magnitude of the Jacobian matrix (of derivatives) of the function mapping input to representation.

Priors regarding the underlying factors

- Simplicity of Factor Dependencies: in good high-level representations, the factors are related to each other through simple dependencies. The simplest possible is marginal independence, $P(\mathbf{h}) = \prod_i P(\mathbf{h}_i)$, but linear dependencies are also reasonable assumptions. This can be seen in many laws of physics, and is assumed when plugging a linear predictor or a factorized prior on top of a learned representation.

Better mixing via deep representations

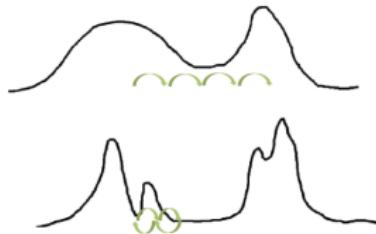
- Why and to what extent different deep learning algorithms may help disentangle underlying factors?
- Better representations can be exploited to produce Markov chains that mix faster between modes
- Mixing between modes would be more efficient at higher levels of representation
- The higher-level samples fill more uniformly the space they occupy and the high-density manifolds tend to unfold when represented at higher levels

Challenge of sampling with MCMC

- Unsupervised learned deep models can be used generate samples
- In general the associated sampling algorithms involve a Markov Chain and MCMC techniques, and these can suffer from a fundamental problem of mixing between modes: it is difficult for the Markov chain to jump from one mode of the distribution to another, when these are separated by large low-density regions, a common situation in real-world data, and under the manifold hypothesis
- This hypothesis states that natural classes present in the data are associated with low-dimensional regions in input space (manifolds) near which the distribution concentrates, and that different class manifolds are well-separated by regions of very low density
- Slow mixing between modes means that consecutive samples tend to be correlated (belong to the same mode) and that it takes many consecutive sampling steps to go from one mode to another and even more to cover all of them, i.e., to obtain a large enough representative set of samples (e.g. to compute an expected value under the target distribution).
- This happens because these jumps through the empty low-density void between modes are unlikely and rare events.

Challenge of sampling with MCMC

- When a learner has a poor model of the data, e.g., in the initial stages of learning, the model tends to correspond to a smoother and higher-entropy (closer to uniform) distribution, putting mass in larger volumes of input space, and in particular, between the modes (or manifolds).
- Keep in mind that MCMCs tend to make moves to nearby probable configurations. Mixing between modes is therefore initially easy for such poor models.
- However, as the model improves and its corresponding distribution sharpens near where the data concentrate, mixing between modes becomes considerably slower. Making one unlikely move (i.e., to a low-probability configuration) may be possible, but making N such moves becomes exponentially unlikely in N



Top: early during training, MCMC mixes easily between modes because the estimated distribution has high entropy and puts enough mass everywhere for small-steps movements (MCMC) to go from mode to mode. Bottom: later on, training relying on good mixing can stall because estimated modes are separated by vast low-density deserts.

Challenge of sampling with MCMC

- Since sampling is an integral part of many learning algorithms (e.g., to estimate the log-likelihood gradient), slower mixing between modes then means slower or poorer learning, and one may even suspect that learning stalls at some point because of the limitations of the sampling algorithm.
- Bengio et al. showed that mixing between modes is easier when sampling at the higher levels of representation
- Deeper generative models produce not only better features for classification but also better quality samples (in the sense of better corresponding to the target distribution being learned)

Hypothesis H1

- **Depth vs Better Mixing Between Modes:** A successfully trained deeper architecture has the potential to yield representation spaces in which Markov chains mix faster between modes.
- If experiments validate that hypothesis, the most important next question is: why?



Sequences of 25 samples generated with a Contractive Auto-Encoder (CAE) on TFD (rows 1 and 2, respectively for 1 or 2 hidden layers) and with an RBM on MNIST (rows 3 and 4, respectively for 1 or 2 hidden layers). On TFD, the second layer clearly allows to get quickly from woman samples (left) to man samples (right) passing by various facial expressions whereas the single hidden layer model shows poor samples. Bottom rows: On MNIST, the single-layer model gets stuck near the same mode while the second layer allows to mix among classes.

Hypothesis H2

- **Depth vs Disentangling:** Part of the explanation of H1 is that deeper representations can better disentangle the underlying factors of variation
- Imagine an abstract (high-level) representation for object image data in which one of the factors is the “reverse video bit”, which inverts black and white, e.g., flipping that bit replaces intensity $x \in [0, 1]$ by $1 - x$. With the default value of 0, the foreground object is dark and the background is light. Clearly, flipping that bit does not change most of the other semantic characteristics of the image, which could be represented in other high-level features.
- However, for every image-level mode, there would be a reverse-video counterpart mode in which that bit is flipped: these two modes would be separated by vast “empty” (low density) regions in input space, making it very unlikely for any Markov chain in input space (e.g. Gibbs sampling in an RBM) to jump from one of these two modes to another, because that would require most of the input pixels or hidden units of the RBM to simultaneously flip their value.
- Instead, if we consider the high-level representation which has a “reverse video” bit, flipping only that bit would be a very likely event under most Markov chain transition probabilities, since that flip would be a small change preserving high probability.

Hypothesis H2

- As another example, imagine that some of the bits of the high-level representation identify the category of the object in the image, independently of pose, illumination, background, etc. Then simply flipping one of these object-class bits would also drastically change the raw pixel-space image, while keeping likelihood high. Jumping from an object-class mode to another would therefore be easy with a Markov chain in representation-space, whereas it would be much less likely in raw pixel-space.
- Disentangling cannot be perfect in deep learning. Better disentangling would mean that some of the learned features have a higher mutual information with some of the known factors. One would expect at the same time that the features that are highly predictive of one factor be less so of other factors, i.e., that they specialize to one or a few of the factors, becoming invariant to others.

Hypothesis H3

- **Disentangling Unfolds and Expands.** Part of the explanation of **H2** is that more disentangled representations tend to
 - (a) unfold the manifolds near which raw data concentrates, as well as
 - (b) expand the relative volume occupied by high-probability points near these manifolds
- **H3(a)** says is that disentangling has the effect that the projection of high-density manifolds in the high-level representation space have a smoother density and are easier to model than the corresponding high-density manifolds in raw input space.
- Let us again use an object recognition analogy. If we have perfectly disentangled object identity, pose and illumination, the high-density manifold associated with the distribution of features in high-level representation-space is at: we can interpolate between some training examples (i.e. likely configurations) and yet stay in a high-probability region.
- For example, we can imagine that interpolating between two images of the same object at different poses (lighting, position, etc.) in a high-level representation-space would yield images of the object at intermediate poses (i.e., corresponding to likely natural images), whereas interpolating in pixel space would give a superposition of the two original images (i.e., unlike any natural image).

Hypothesis H3

- If interpolating between high-probability examples (i.e. within their convex set) gives high-probability examples, then it means that the distribution is more uniform (fills the space) within that convex set, which is what **H3(b)** is saying.



we interpolate between samples of different classes, at different depths (top=raw input, middle=1st layer, bottom=2nd layer). Note how in lower levels one has to go through unpalatable patterns, whereas in the deeper layers one almost jumps from a high-density region of one class to another (of the other class)

Course materials are from

- Y. Bengio, I. J. Goodfellow, and A. Courville, “Distributed Representations: Disentangling the Underlying Factors” in “Deep Learning” Book in preparation for MIT press, 2014.
- Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai, “Better Mixing via Deep Representations” ICML 2013.