Convolutional Neural Network for Computer Vision Feature Learning

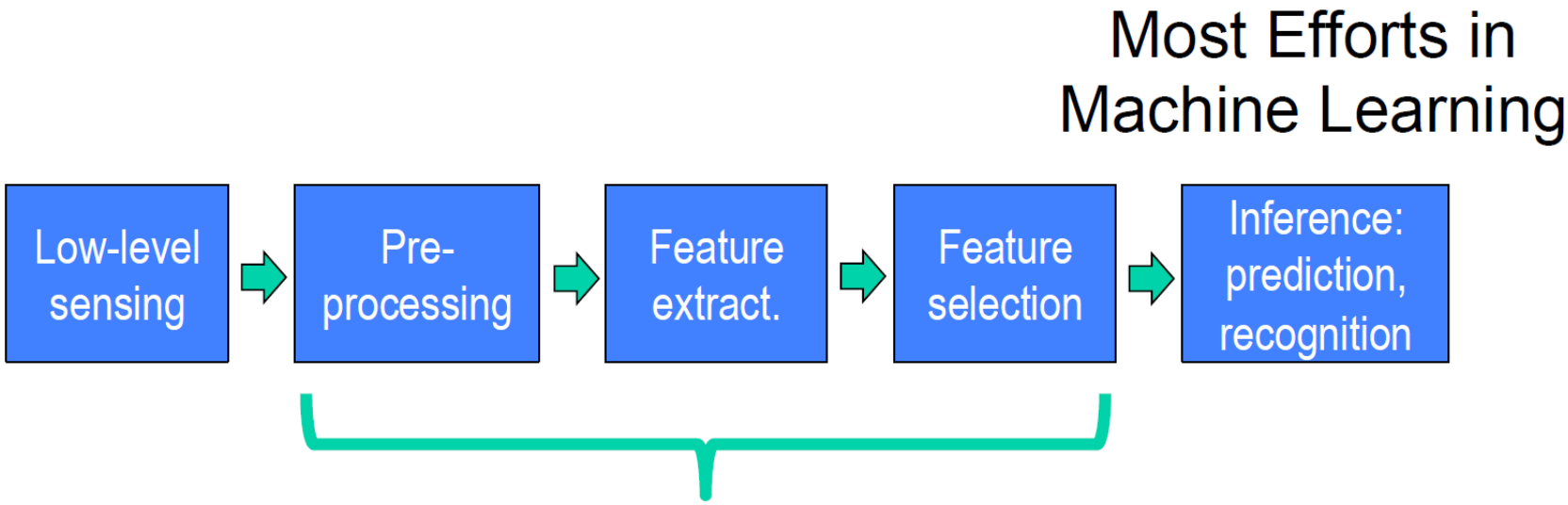
Dequan Wang

Fudan University

dqwang12@fudan.edu.cn

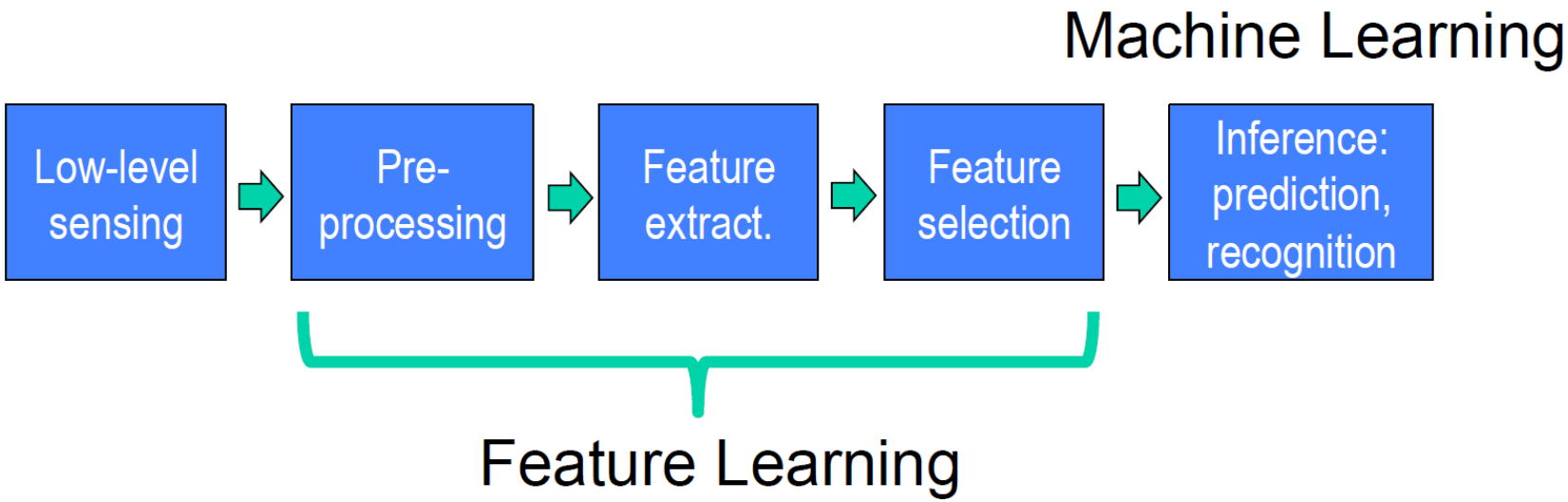
September 15, 2014

Pipeline of Machine Visual Perception



- Most critical for accuracy
- Account for most of the computation for testing
- Most time-consuming in development cycle
- Often hand-craft in practice

Learning with Structures

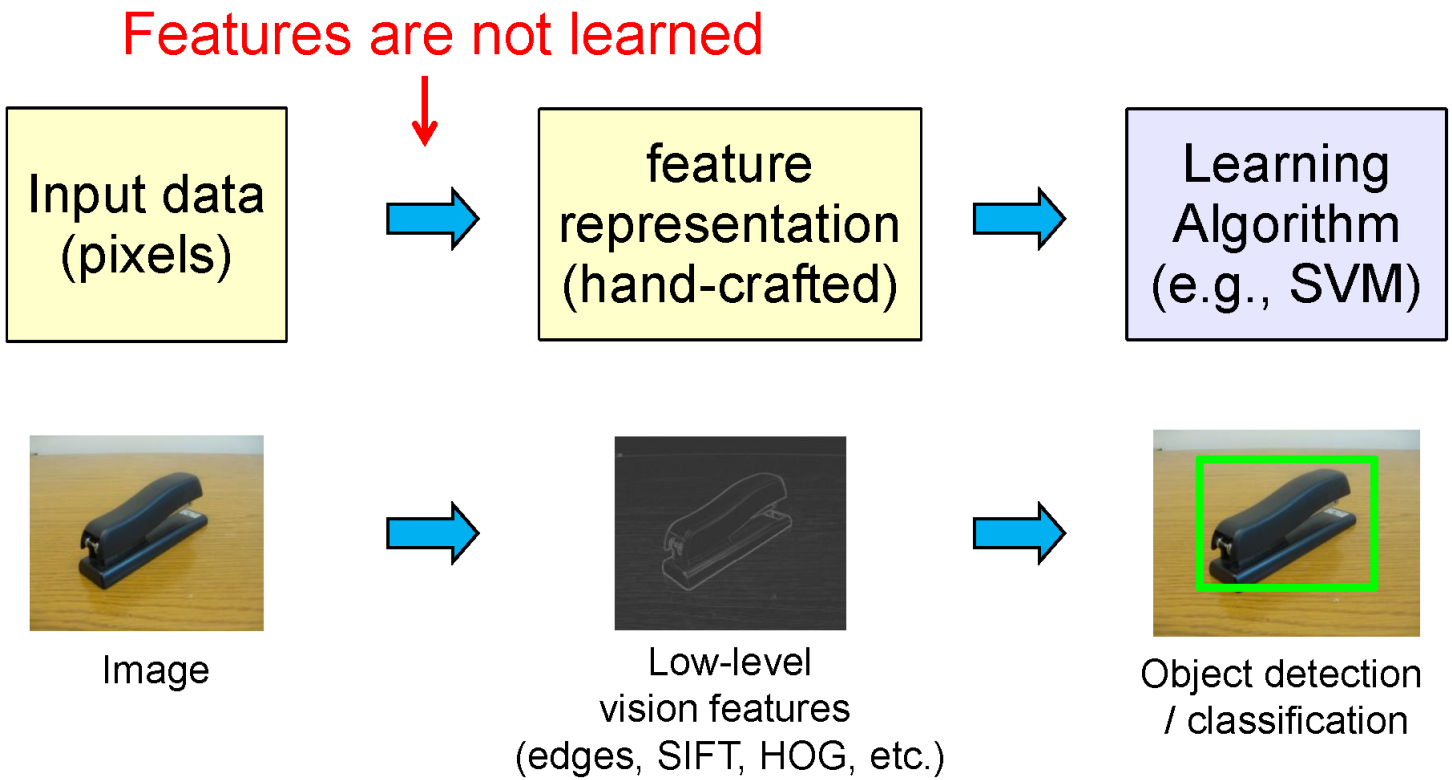


- Kernel Learning
- Transfer Learning
- Semi-Supervised Learning
- Manifold Learning
- Sparse Learning
- Structured Input-Output Prediction

Slide Courtesy: Kai Yu

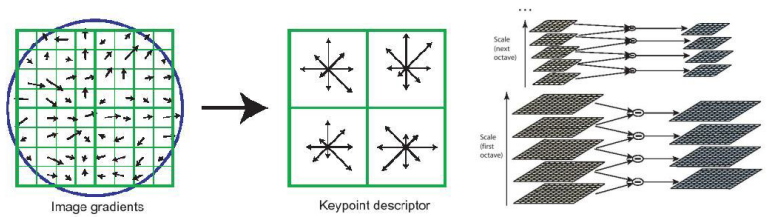
Traditional Recognition System

- Features are key to recent progress in recognition
- Multitude of hand-designed features currently in use

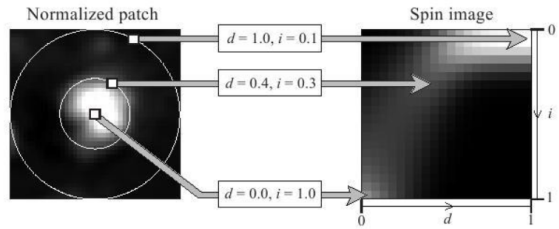


Traditional Visual Feature

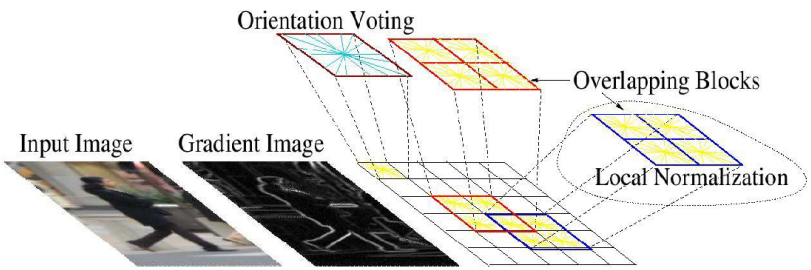
- Where next? Better classifiers or better feature?



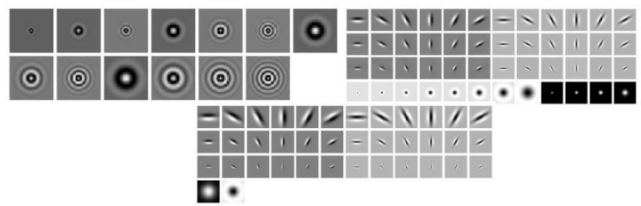
SIFT



Spin image



HoG



Textons

and many others:

SURF, MSER, LBP, Color-SIFT, Color histogram, GLOH, ...

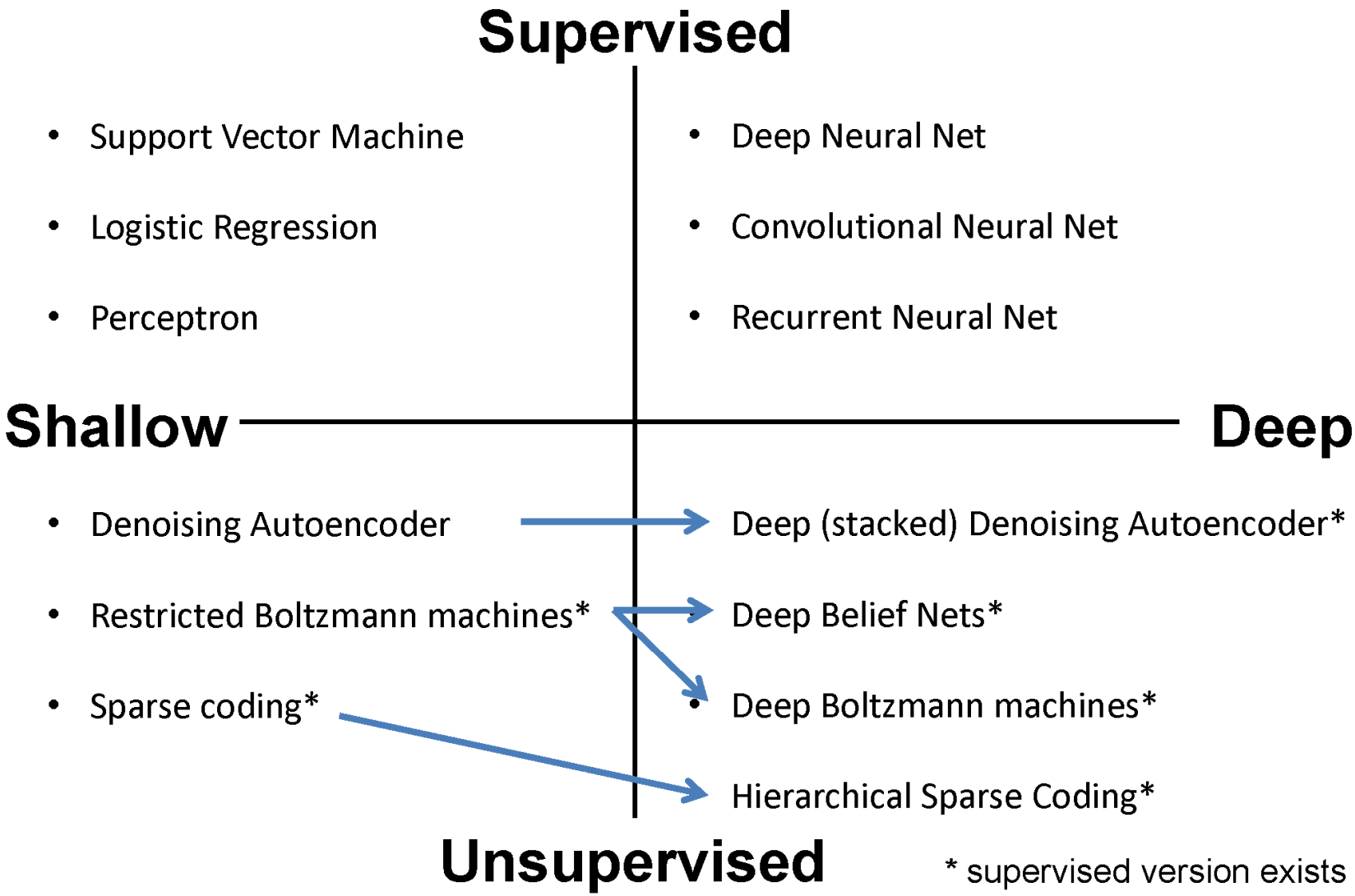
Approaches to Learning Features

- Supervised Learning
  - End-to-end learning of deep architectures (e.g., deep neural networks) with back-propagation
  - Works well when the amounts of labels is large
  - Structure of the model is important (e.g. convolutional structure)

- Unsupervised Learning
  - Learn statistical structure or dependencies of the data from unlabeled data
  - Layer-wise training
  - Useful when the amount of labels is not large

Slide Courtesy: Honglak Lee

Representation Learning

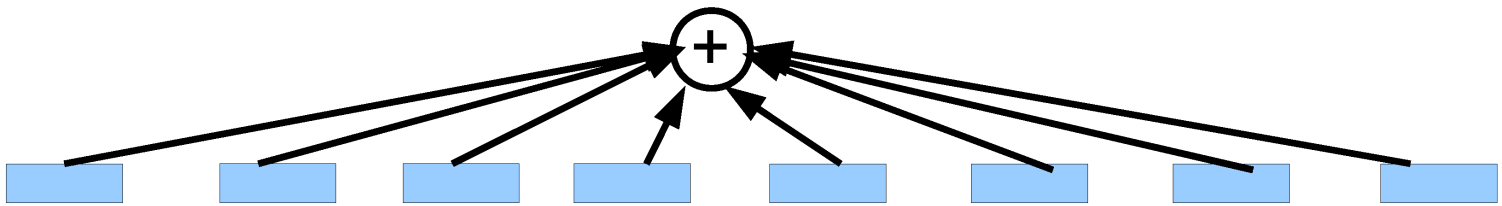


Slide Courtesy: Honglak Lee

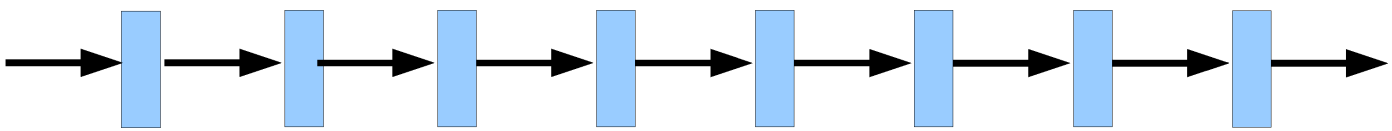
Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

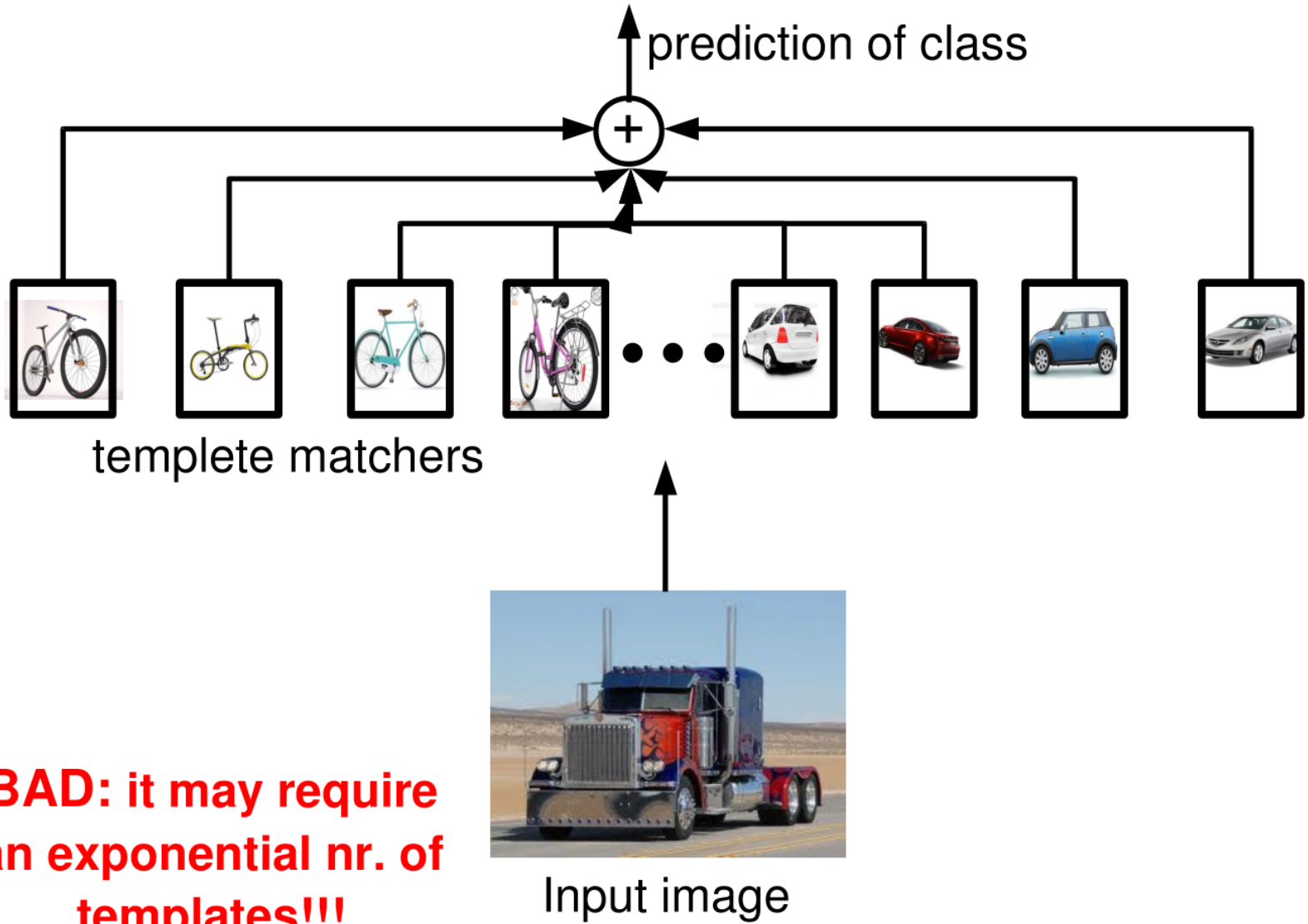
Proposal #1: linear combination $f(x) \approx \sum_j g_j$



Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



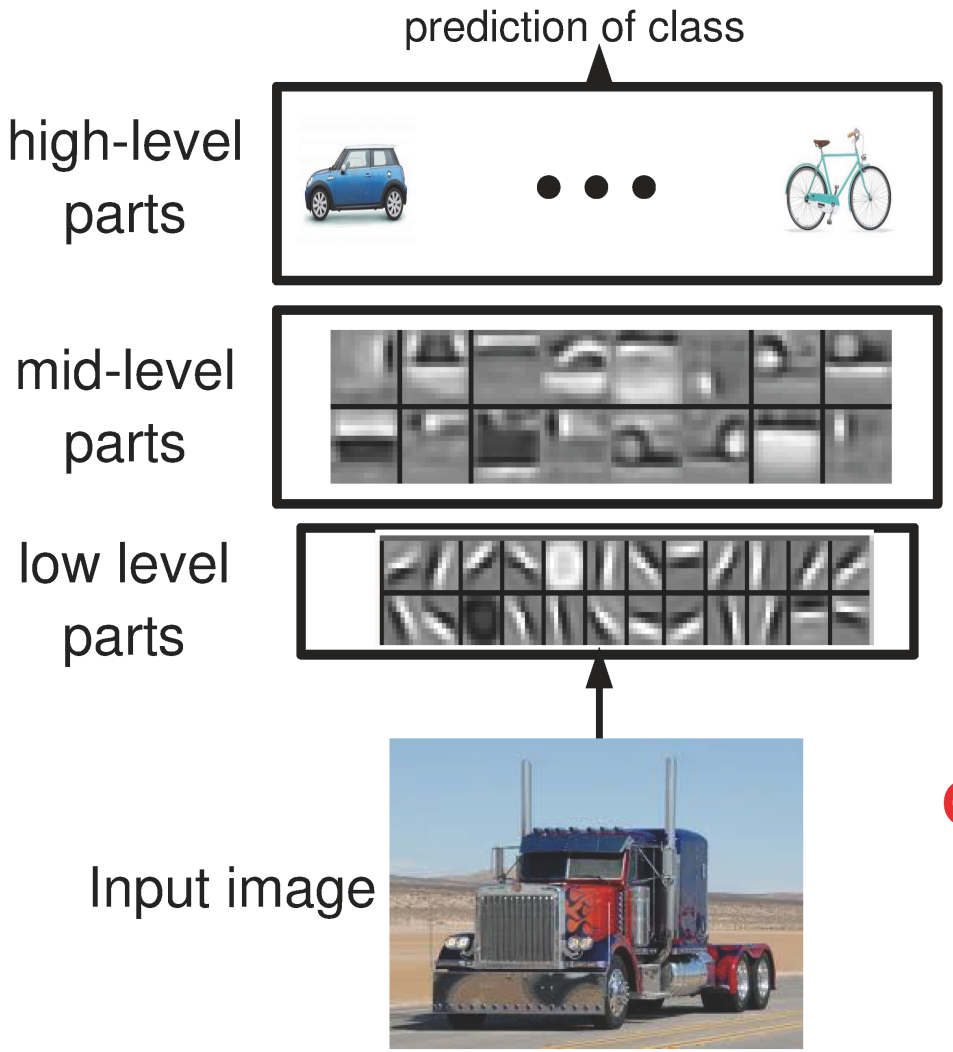
Linear Combination



BAD: it may require an exponential nr. of templates!!!

Slide Courtesy: Marc'Aurelio Ranzato

Composition



- reuse intermediate parts
- distributed representations

GOOD: (exponentially) more efficient

Slide Courtesy: Marc'Aurelio Ranzato

Ranzato's Definition

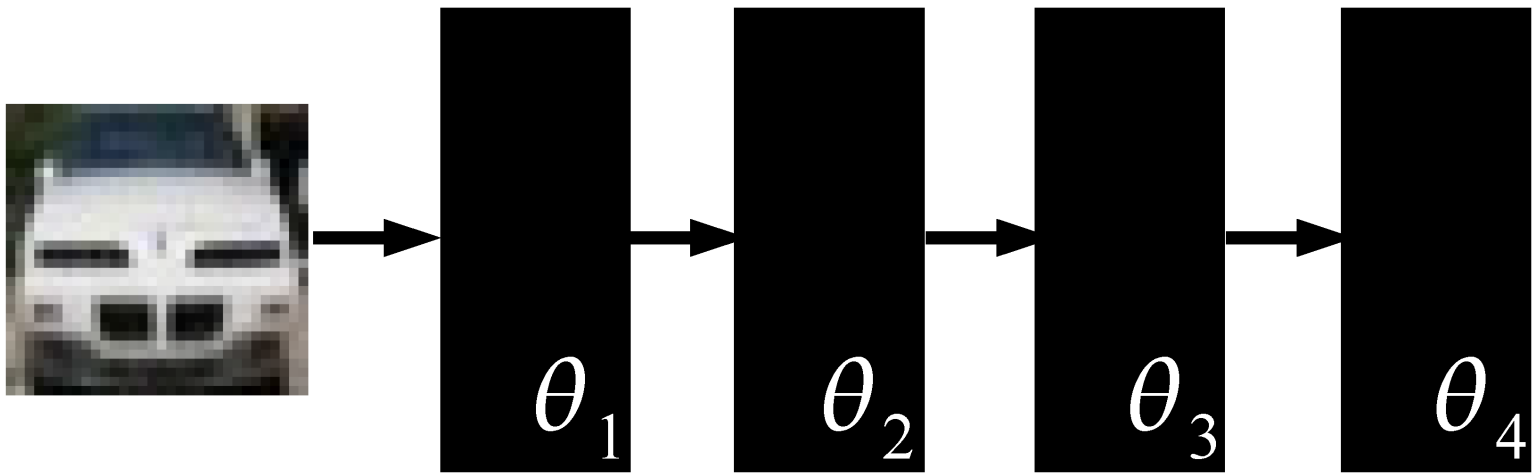
Deep Learning is a method which makes predictions by using a sequence of non-linear processing stages. The resulting intermediate representations can be interpreted as feature hierarchies and the whole system is jointly learned from data.

- Some deep learning methods are probabilistic, others are loss-based, some are supervised, others are unsupervised ...

Slide Courtesy: Marc'Aurelio Ranzato

Deep Learning in Practive

Optimization is easy, need to know a few tricks of the trade.



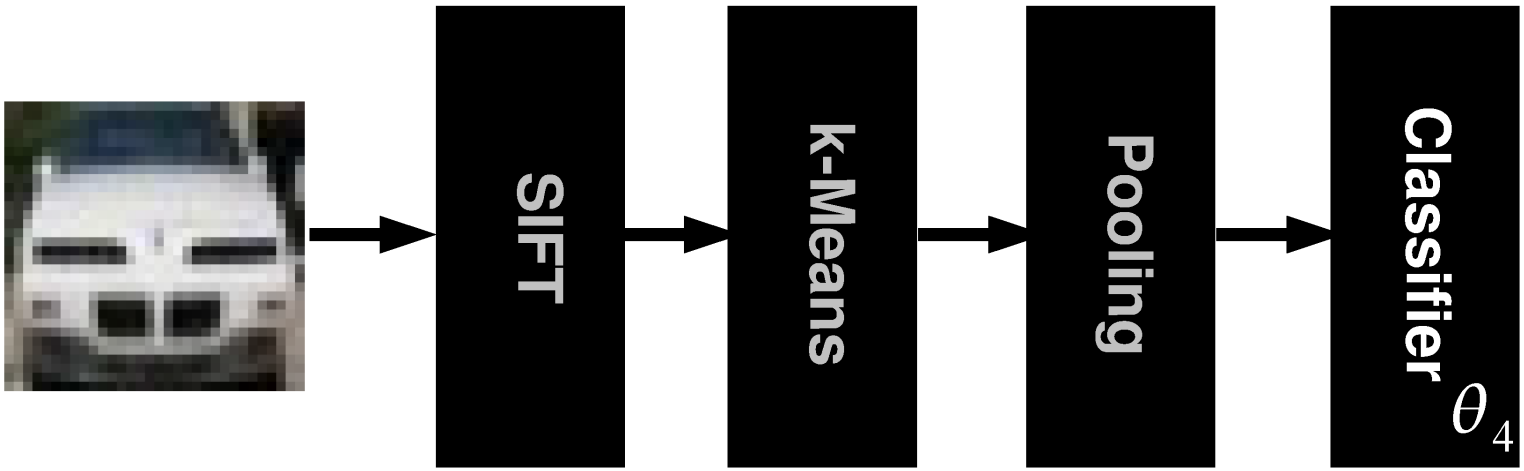
Q: What's the feature extractor? And what's the classifier?

A: No distinction, end-to-end learning!

Slide Courtesy: Marc'Aurelio Ranzato

A Potential Problem with Deep Learning

Optimization is difficult: non-convex, non-linear system



Solution #1: freeze first N-1 layer (engineer the features)

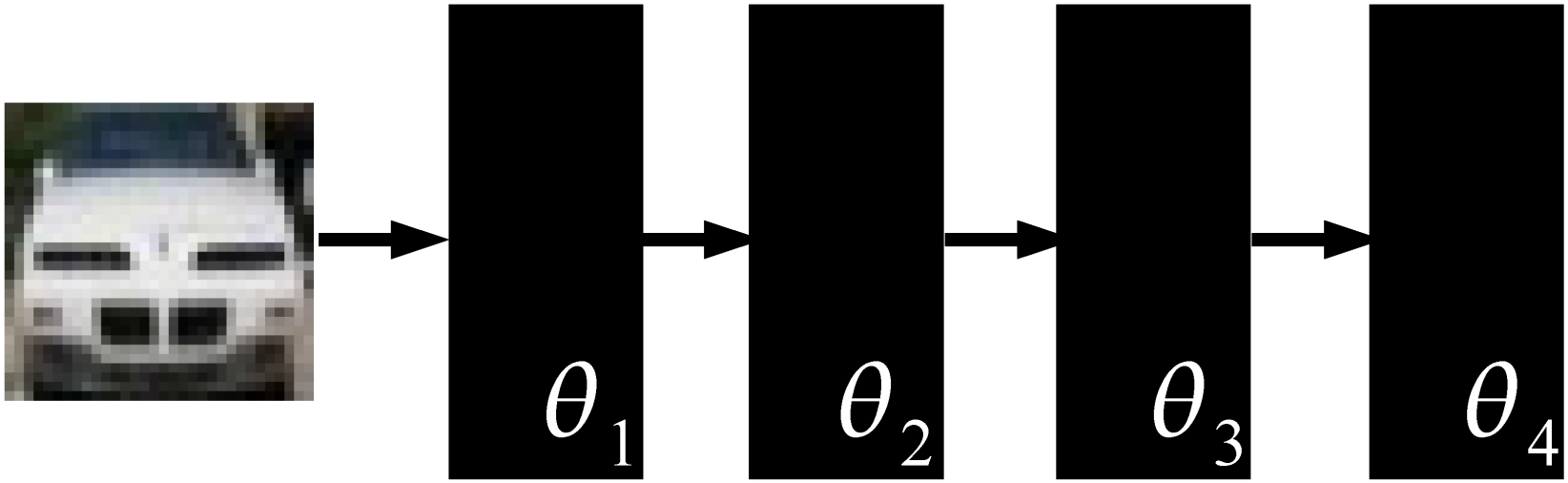
It makes it shallow!

- How to design features of features?
- How to design features for new imagery?

Slide Courtesy: Marc'Aurelio Ranzato

A Potential Problem with Deep Learning

Optimization is difficult: non-convex, non-linear system



Solution #2: live with it!

It will converge to a local minimum.

It is much more powerful!!

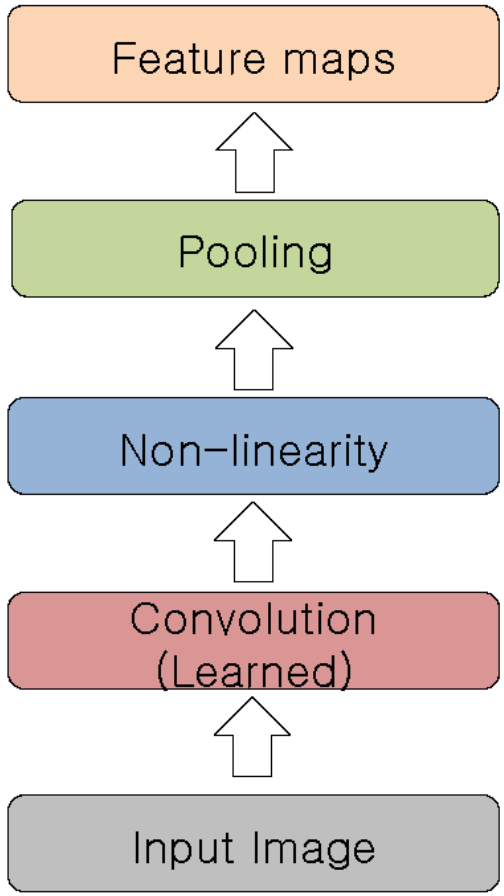
Given lots of data, engineer less and learn more!!

Just need to know a few tricks of the trade...

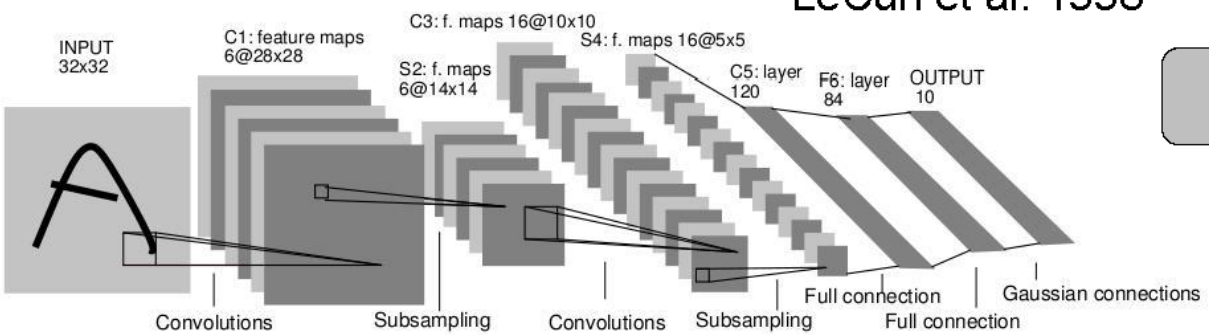
Slide Courtesy: Marc'Aurelio Ranzato

Convolutional Neural Network

- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Supervised
- Train convolutional filters by back-propagating classification error



LeCun et al. 1998



Slide Courtesy: Rob Fergus

Generalization Error Decomposition

- Given observations $(X_i, Y_i), i = 1, ..., n$
- Lear a predictive function $f(X)$
- Generalization error $E[L(f(X), Y)]$

$$E[L(f(X), Y)] = A + E + O$$

- Approximation error$\rightarrow$model class
- Estimation error$\rightarrow$data size
- Optimization error$\rightarrow$algorithm

Slide Courtesy: Kai Yu

Generalization Error Decomposition

$$E[L(f(X), Y)] = A + E + O$$

- Approximation error→use complex model
- Estimation error→collect big data
- Optimization error→design an OK algorithm

Trend of Machine Learning

- Approximation error: simple model→complex model
- Estimation error: small data→big data
- Optimization error: fine algorithm→scalable algorithm

Why Today

- Non-convex & non-linear
- Intense computation
- Sensitive to initialization
- Overfitting
- Vanishing gradient

- Big data
- GPU
- Large scale parallel computation
- Layer-wise pre-training
- RELU, drop-out, better normalization, etc.

Slide Courtesy: Kai Yu

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer "encoder" network

Benchmark for ImageNet

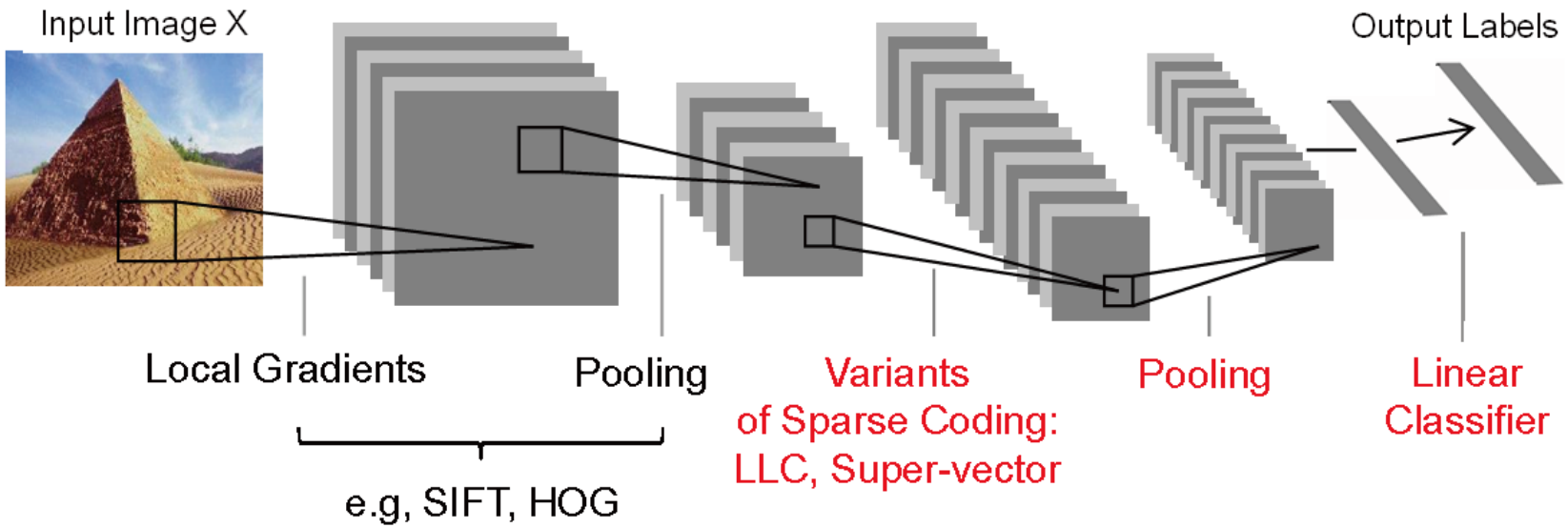
72%, 2010

74%, 2011

85%, 2012

Slide Courtesy: Kai Yu

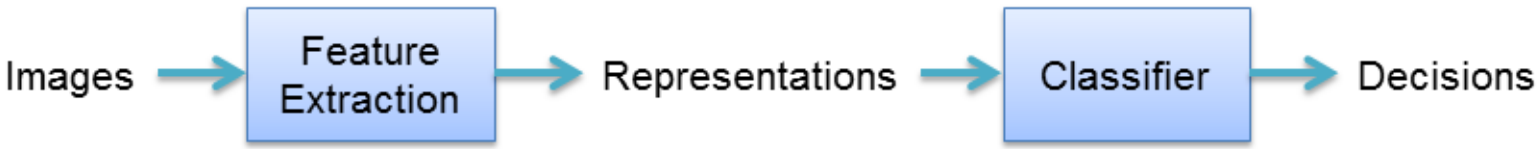
Traditional System on ImageNet



Slide Courtesy: Kai Yu

Drawback of Traditional System

- All these methods are not scalable
  - Handcrafted descriptors, time consuming, limited prior knowledge
  - Representations by unsupervised learning,not tunable
  - Cannot fully utilize large-scale training data



- We need to enable end-to-end learning.

Slide Courtesy: Kai Yu

Deep Learning for Vision@CVPR 2012 Tutorial

Buiding blocks

- RBMs, Autoencoder, Sparse Coding

Go deeper: Layerwise feature learning

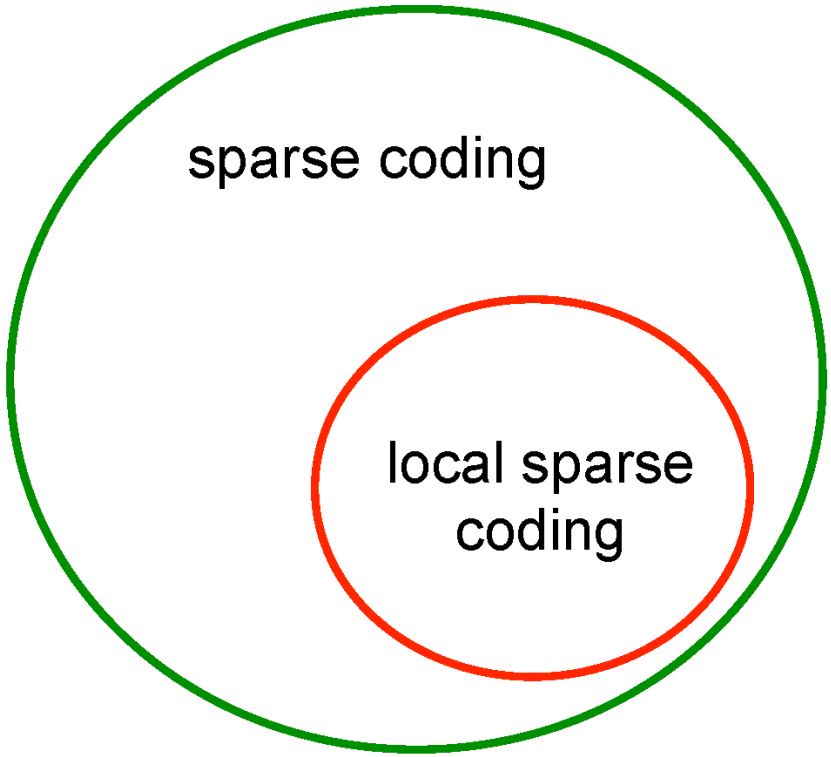
- Layer-by-layer unsupervised training
- Layer-by-layer supervised training

Fine tuning via Backpropogation

- If data are big enough, direct fine tuning is enough
- Sparsity on hidden layers are often useful

Slide Courtesy: Kai Yu

Sparsity vs. Locality



- Intuition: similar data should get similar activated features
- Local sparse coding:
  - data in the same neighborhood tend to have shared activated features;
  - data in different neighborhoods tend to have different features activated.

Slide Courtesy: Kai Yu

Challenge to Deep Learners

Key Issues

- What if no hand-craft features at all?
- What if use much deeper neural networks?

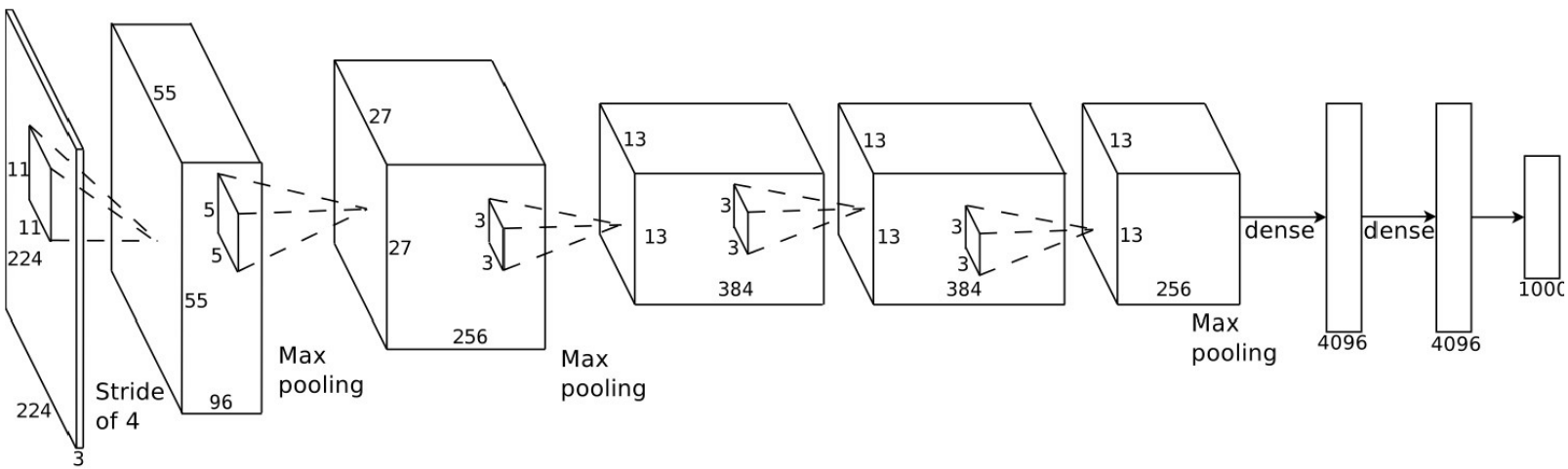
Answer from Geoff Hinton

- Our chief critic, Jitendra Malik, has said that this competition is a good test of whether deep neural networks really do work well for object recognition.

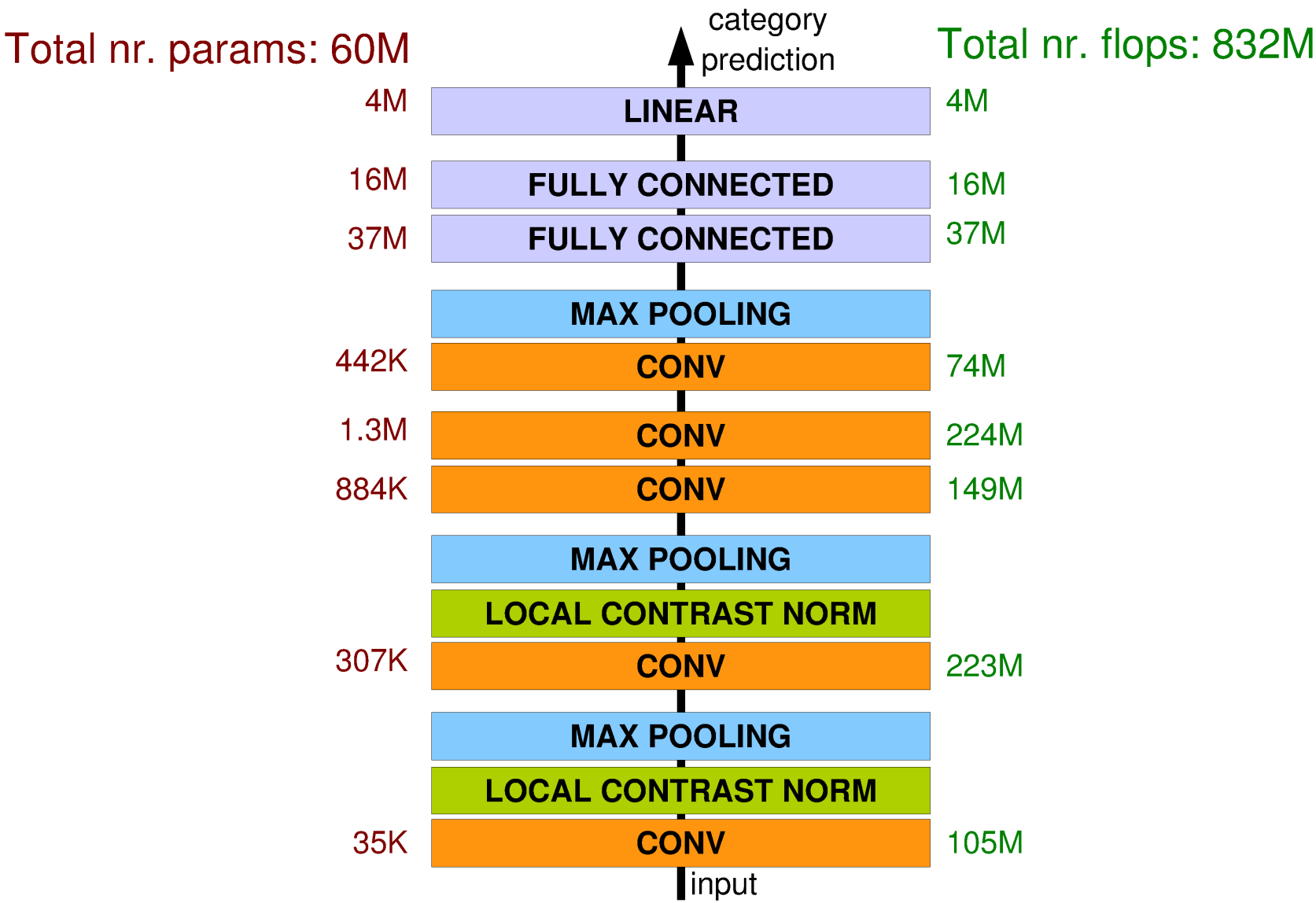
Slide Courtesy: Kai Yu

The Architecture

– 7 hidden layers not counting max pooling.
– Early layers are conv., last two layers globally connected.
– Uses rectified linear units in every layer.
– Uses competitive normalization to suppress hidden activities.



The Architecture

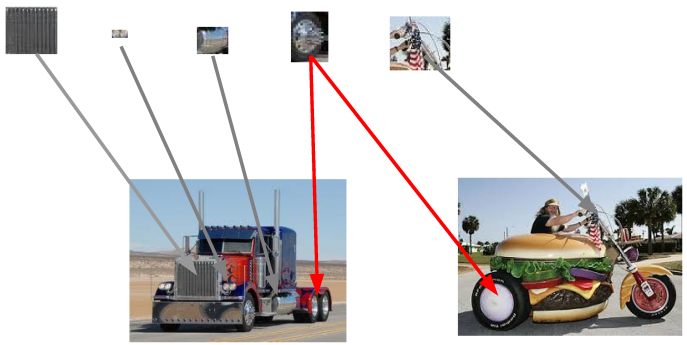


Why Deep

- When input has hierarchical structure, the use of a hierarchical architecture is potentially more efficient because intermediate computations can be re-used.
- Architectures are efficient also because they use distributed representations which are shared across classes.

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

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

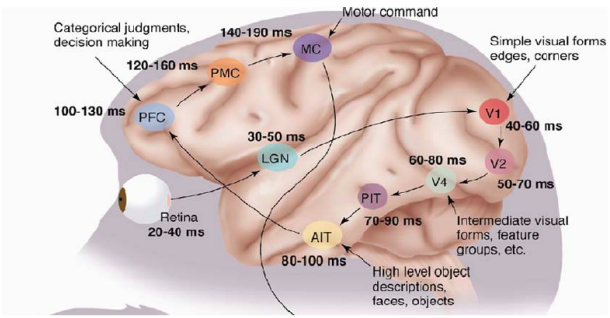


Why Hierarchy

Theoretical

- One limitation was based on the well-known depth-breadth tradeoff in circuits design Hastad [1987].
- This suggests that many functions can be much more efficiently represented with deeper architectures, often with a modest number of levels (e.g., logarithmic in the number of inputs).

Biological

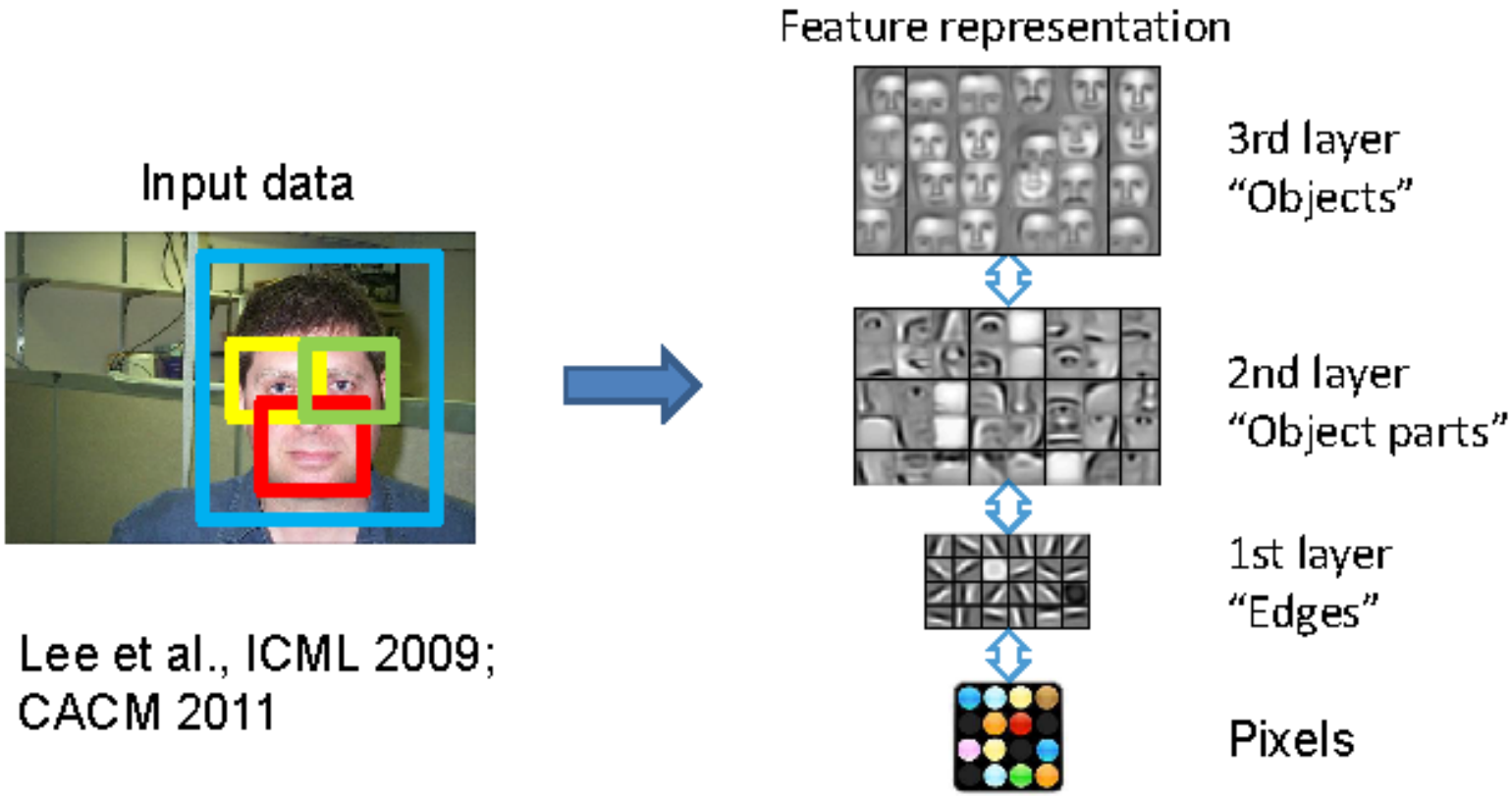


- Visual cortex is hierarchical (Hubel-Wiesel Model)

Slide Courtesy: Kai Yu

Learning Feature Hierarchy

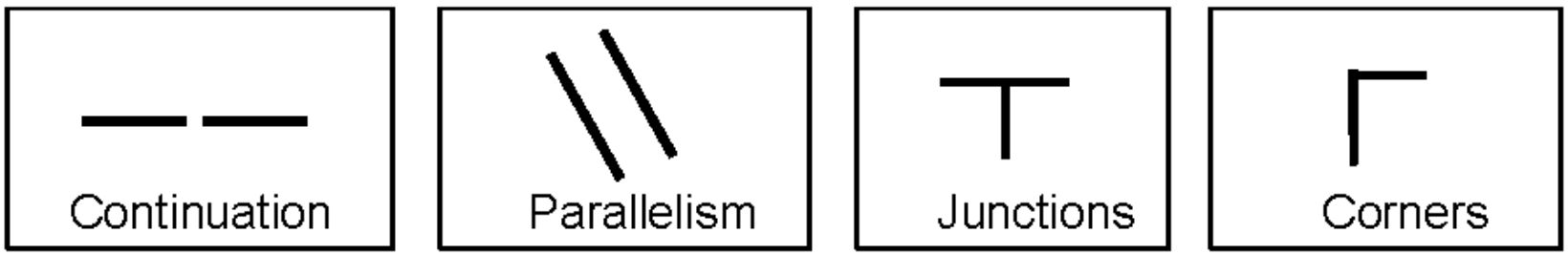
1. Learn **useful higher-level features** from images



2. **Fill in representation gap** in recognition

Mid-Level Representation

- Mid-level cues



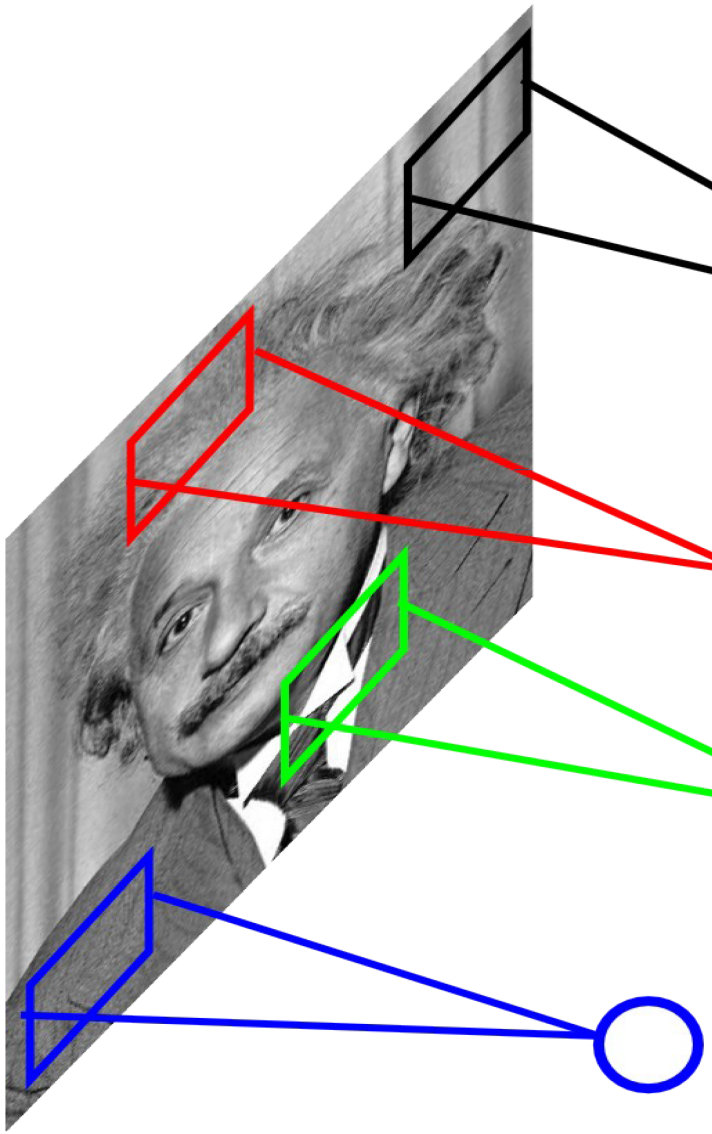
"Tokens" from Vision by D.Marr:

- Object parts:

- Difficult to hand-engineer → What about learning them?

Slide Courtesy: Rob Fergus

Locally Connected Layer



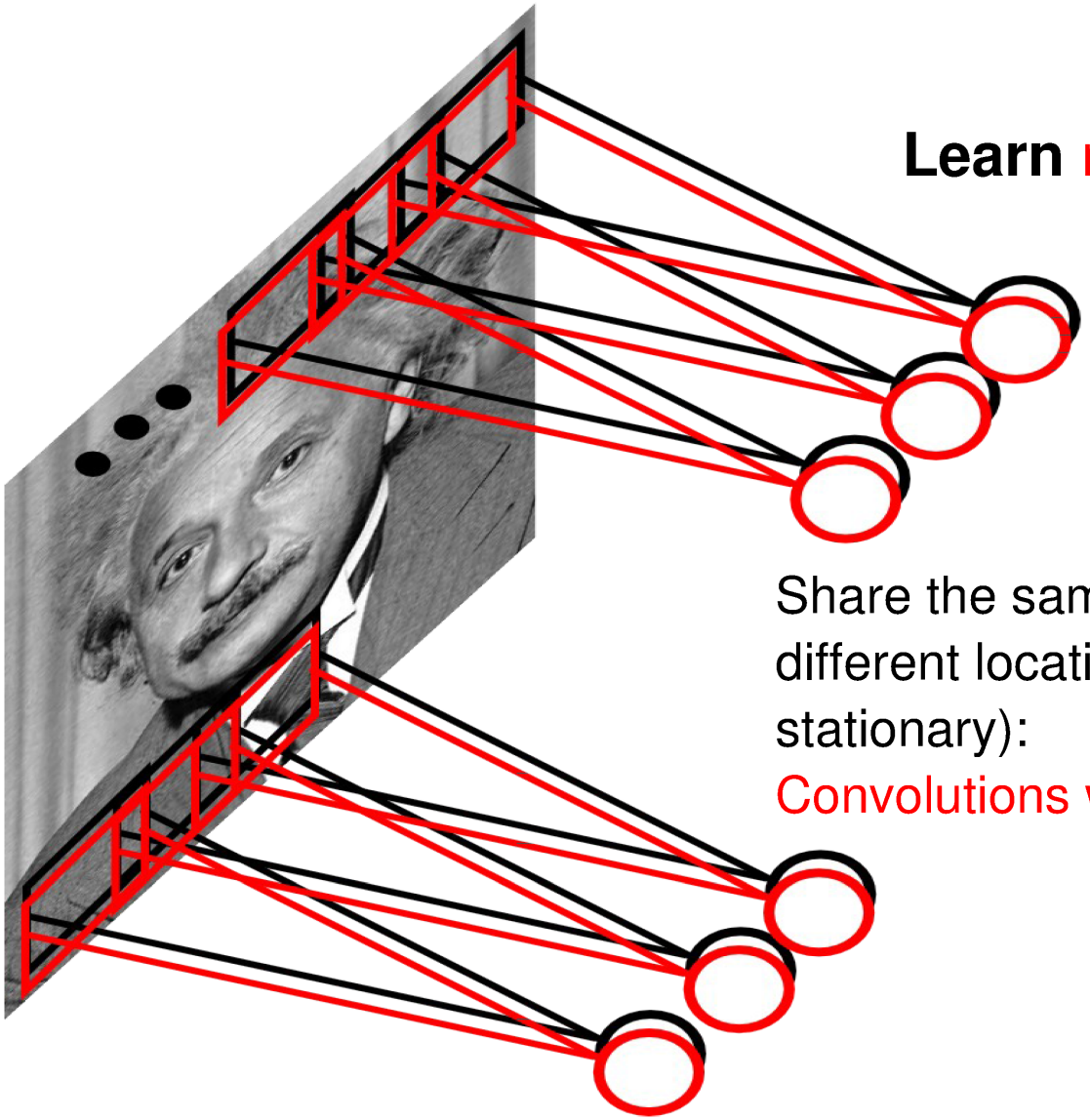
STATIONARITY? Statistics is similar at different locations

Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Slide Courtesy: Marc'Aurelio Ranzato

Convolutional Layer



Learn multiple filters.

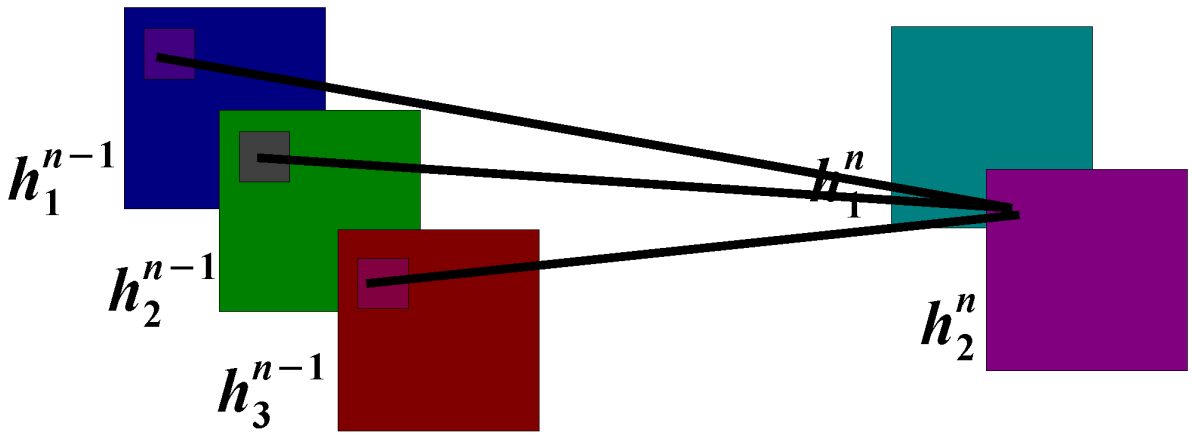
Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Slide Courtesy: Marc'Aurelio Ranzato

Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map **input feature map** **kernel**



Convolutional Layer

A standard neural net applied to images:

- scales quadratically with the size of the input
- does not leverage stationarity

Solution:

- connect each hidden unit to a small patch of the input
- share the weight across space

Convolutional Layer

What is the size of the output?

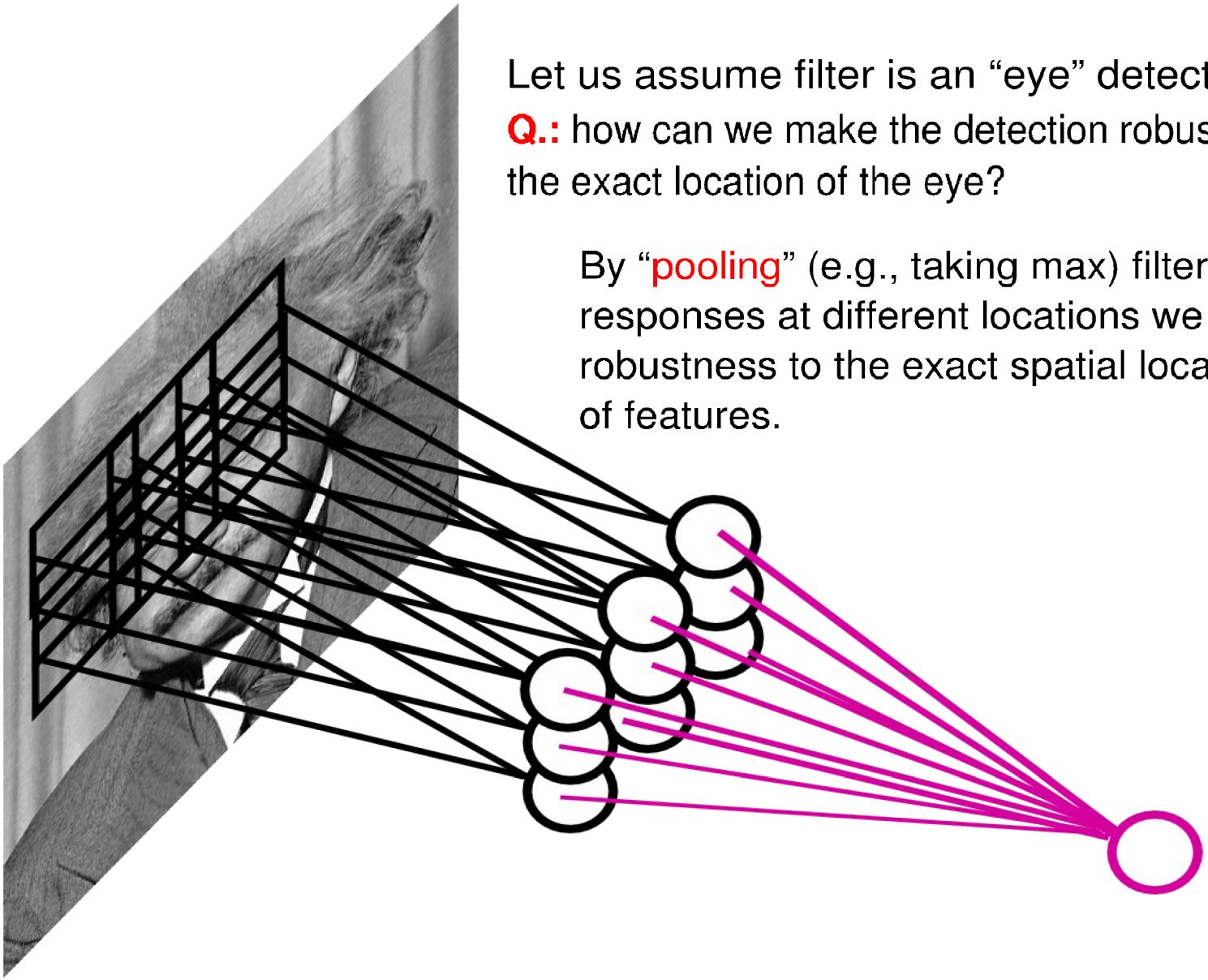
It is proportional to the number of filters and depends on the stride. If kernels have size $K \times K$, input has size $D \times D$, stride is 1, and there are M input feature maps and N output feature maps then:

- the input has size M@D×D
- the output has size N@$(D - K + 1) \times (D - K + 1)$
- the kernels have M×N×K×K coefficients

How many feature maps?

- Usually, there are more output feature maps than input feature maps. Convolutional layers can increase the number of hidden units by big factors

Pooling Layer



Let us assume filter is an “eye” detector

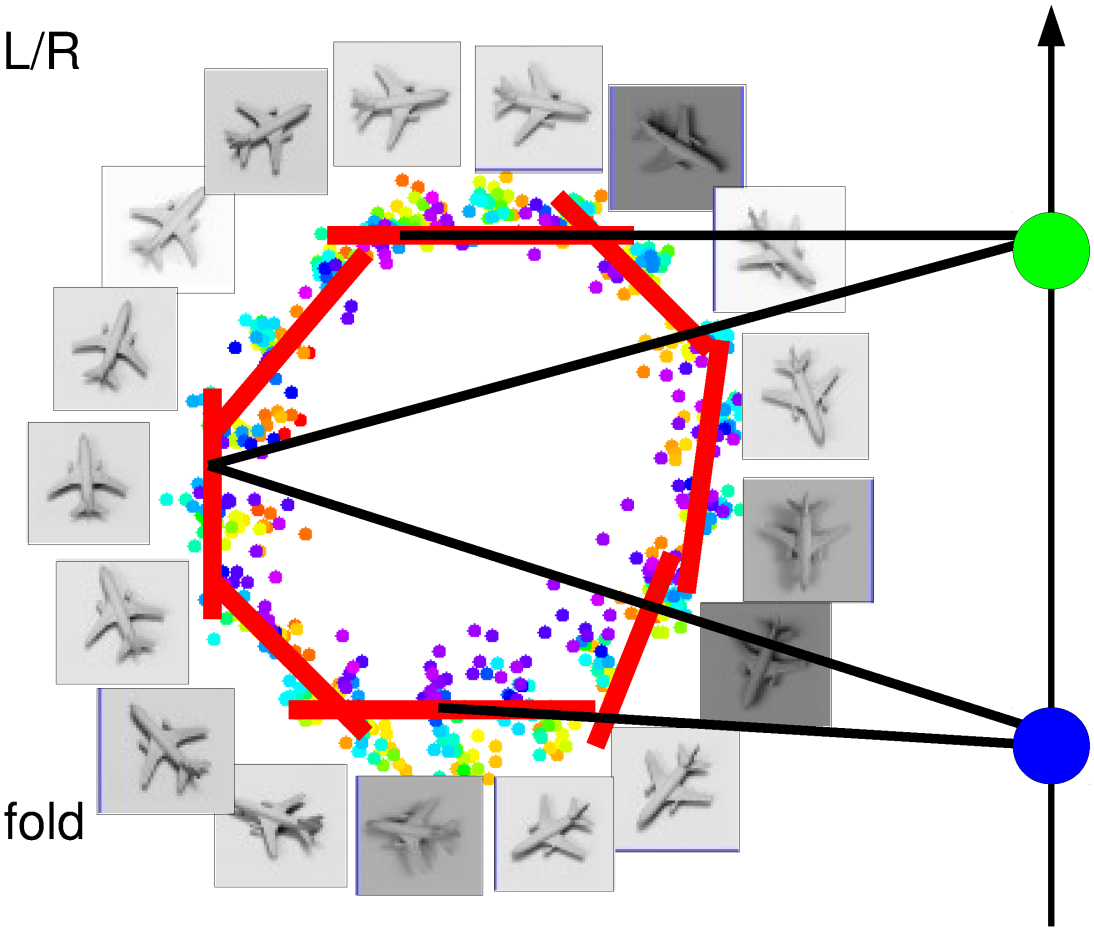
Q.: how can we make the detection robust to the exact location of the eye?

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Slide Courtesy: Marc'Aurelio Ranzato

Pooling Layer Interpretation

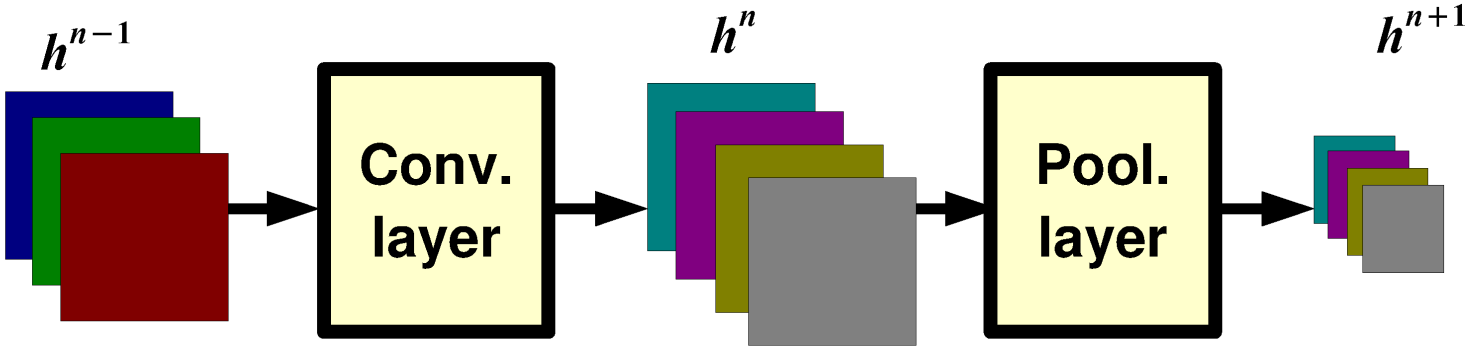
Task: detect orientation L/R



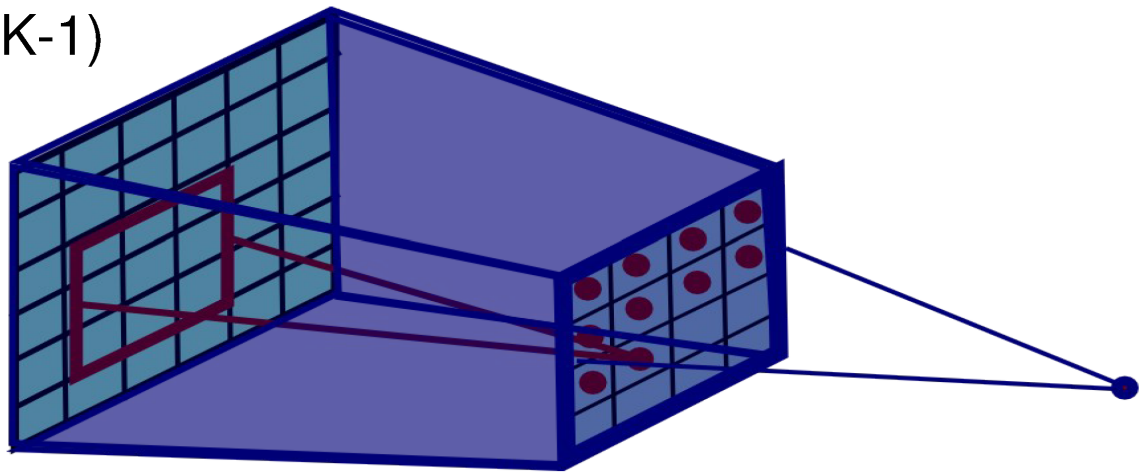
Conv layer:
linearizes manifold

Pooling layer:
collapses manifold

Pooling Layer Receptive Field Size



If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Slide Courtesy: Marc'Aurelio Ranzato

Pooling Layer Examples

Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

L2-pooling:

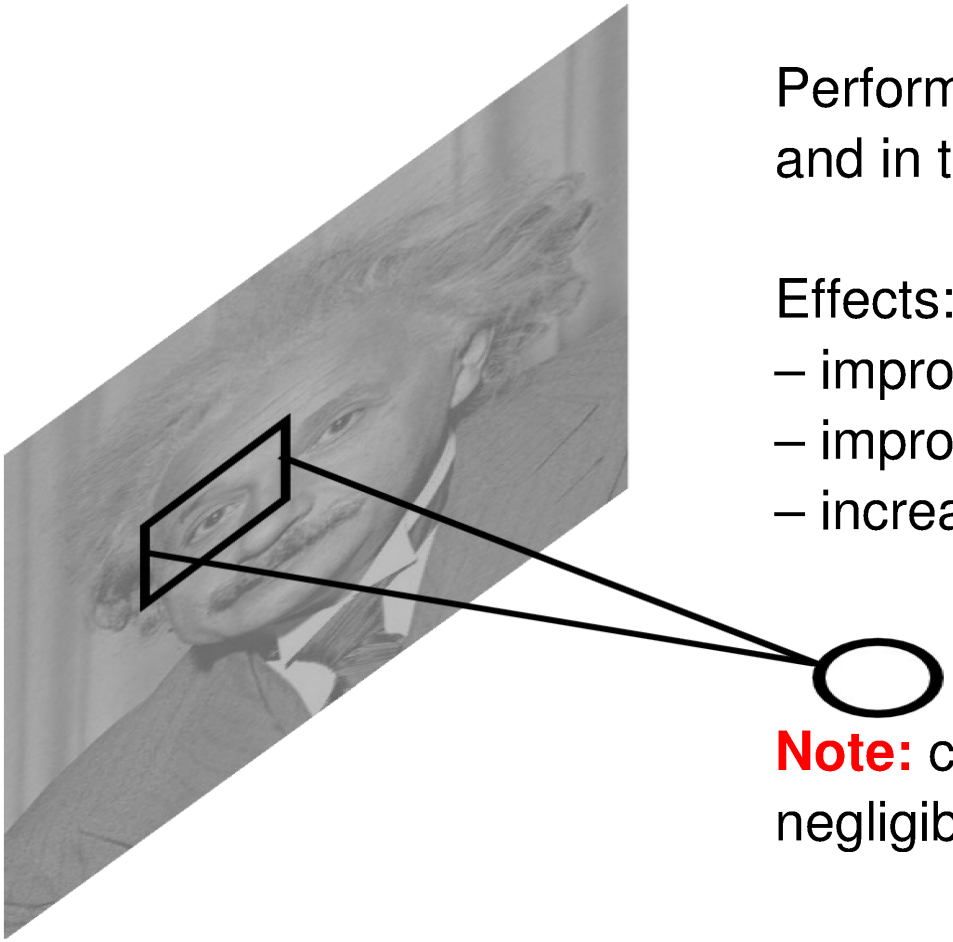
$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

L2-pooling over features:

$$h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

Local Contrast Normalization

$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{\sigma^i(N(x,y))}$$

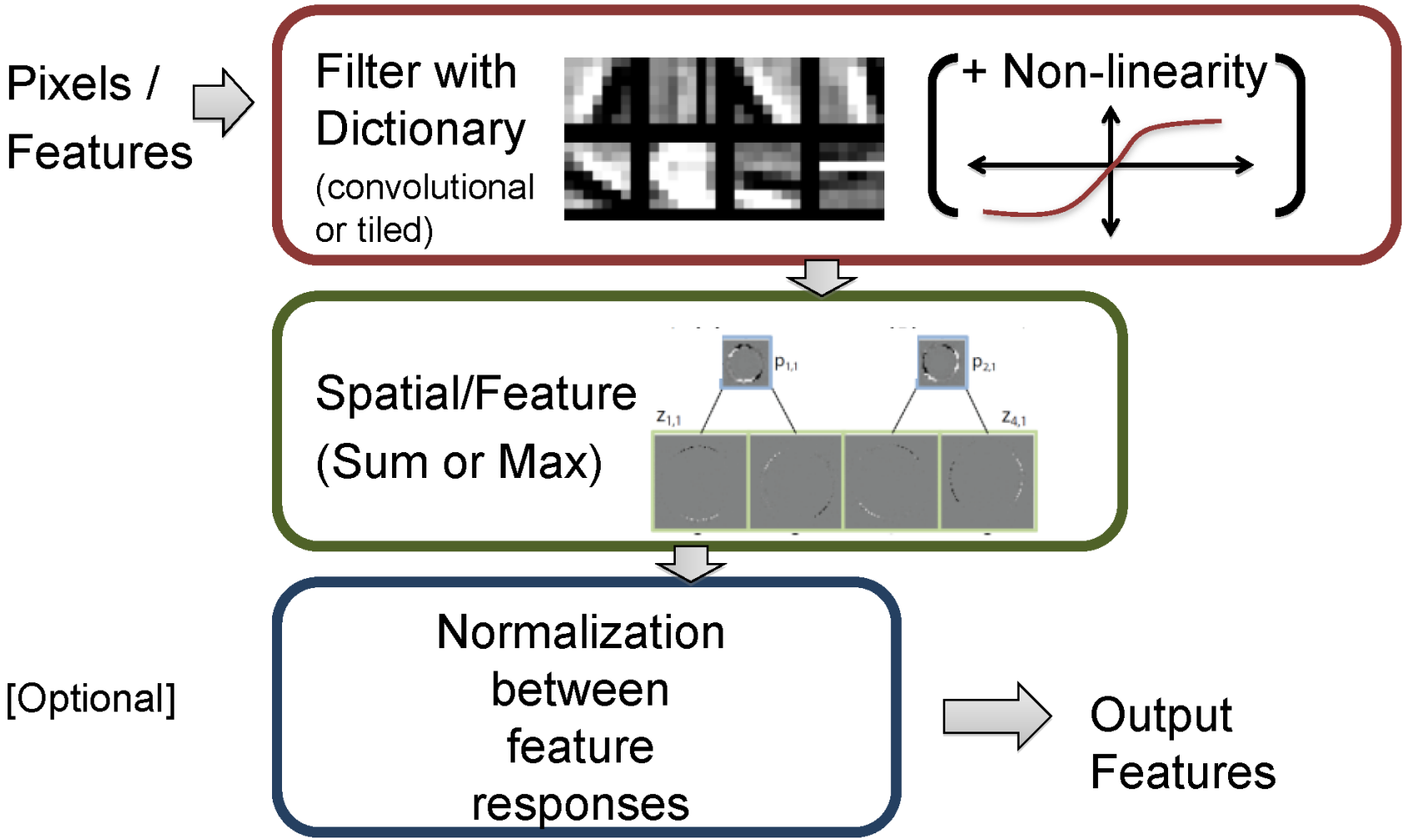


Performed also across features and in the higher layers..

Effects:
– improves invariance
– improves optimization
– increases sparsity

Note: computational cost is negligible w.r.t. conv. layer.

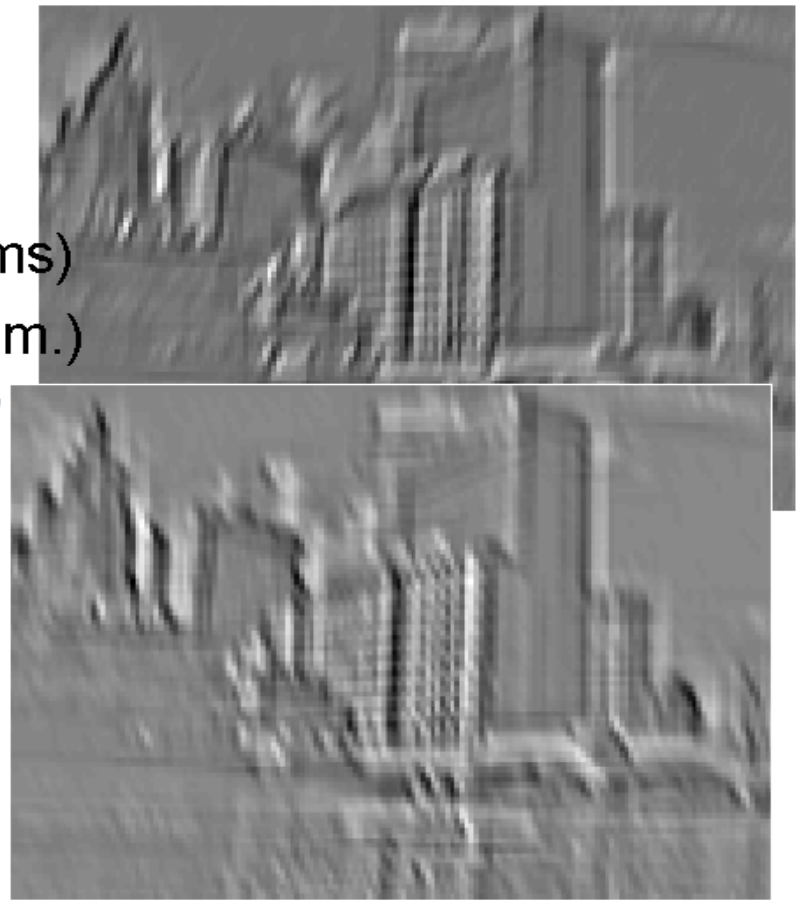
Components of Each Layer



Filtering

- Convolutional
  - Dependencies are local
  - Translation equivariance
  - Tied filter weights (few params)
  - Stride 1,2,… (faster, less mem.)

Input



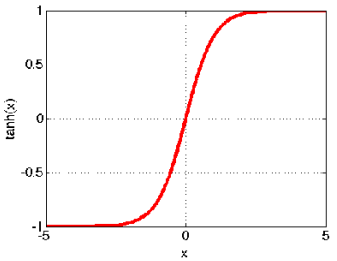
Feature Map

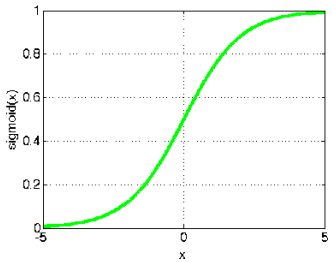
Slide Courtesy: Rob Fergus

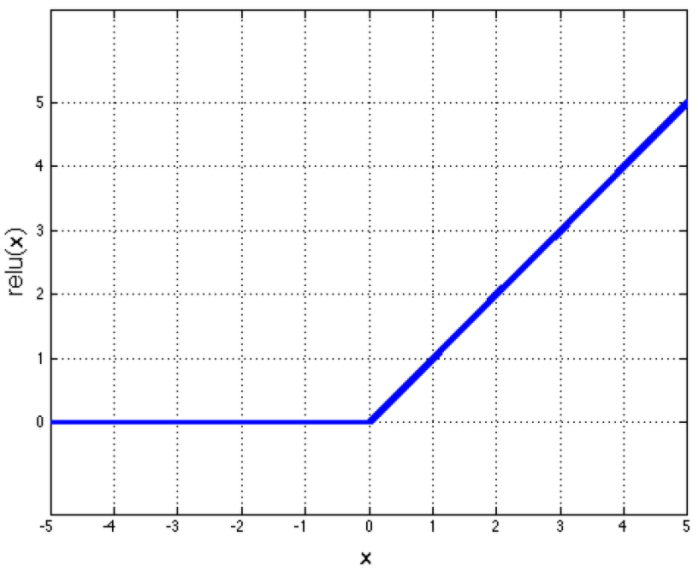
Non-Linearity

- Non-linearity
  - Per-element (independen
  - Tanh
  - Sigmoid: 1/(1+exp(-x))
  - Rectified linear
    - Simplifies backprop
    - Makes learning faster
    - Avoids saturation issues

    → Preferred option



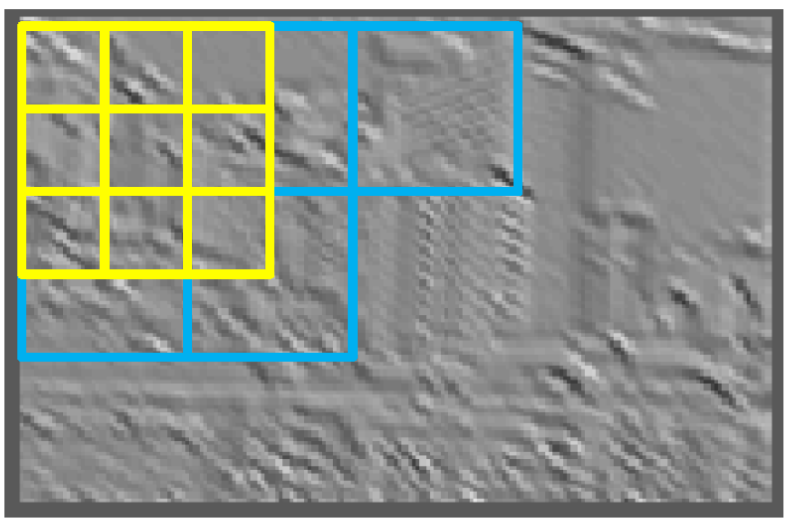


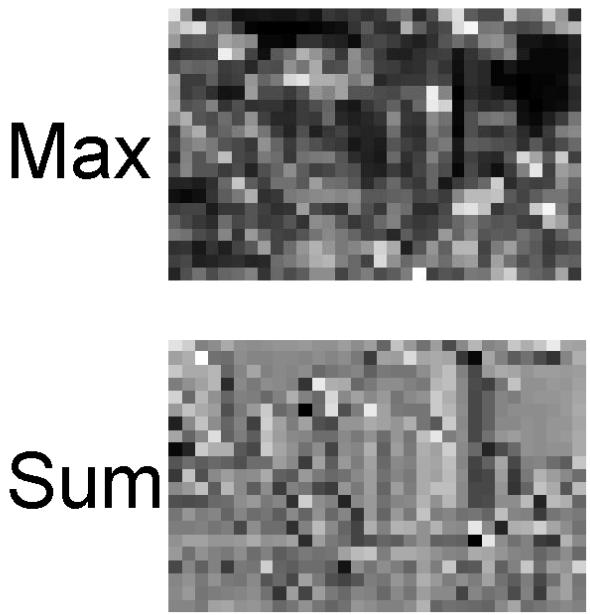


Slide Courtesy: Rob Fergus

Pooling

- Spatial Pooling
  - Non-overlapping / overlapping regions
  - Sum or max
  - Boureau et al. ICML'10 for theoretical analysis

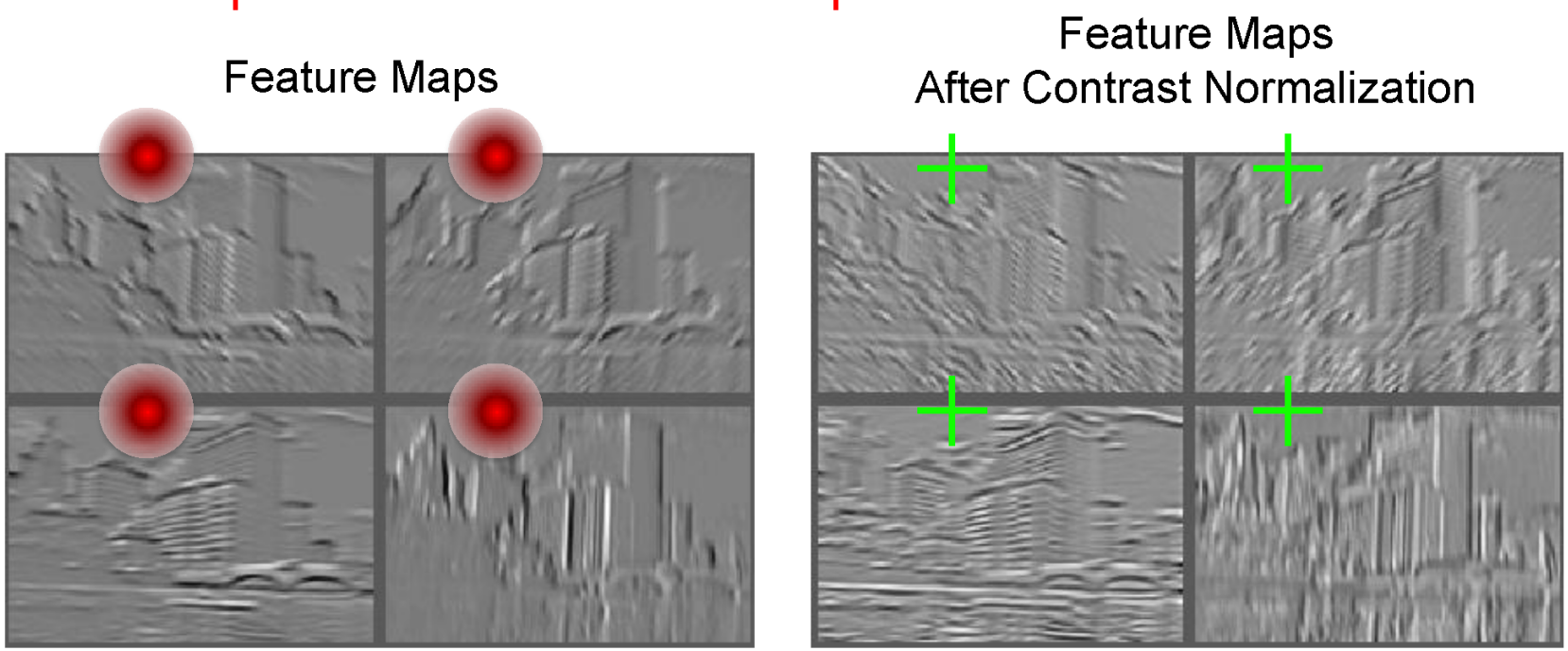




Slide Courtesy: Rob Fergus

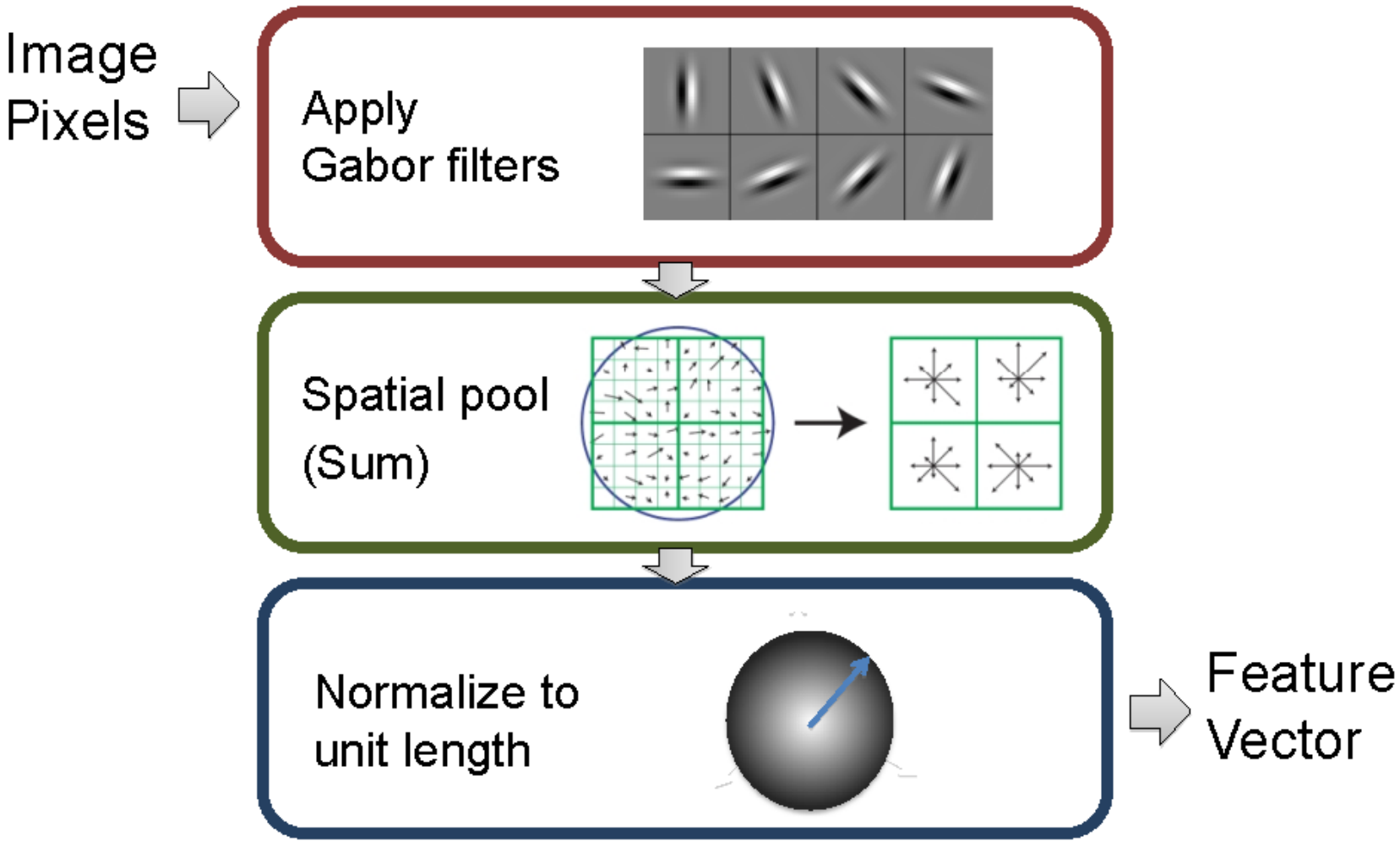
Normalization

- Contrast normalization (across feature maps)
  - Local mean = 0, local std. = 1, "Local" → 7x7 Gaussian
  - Equalizes the features maps



Slide Courtesy: Rob Fergus

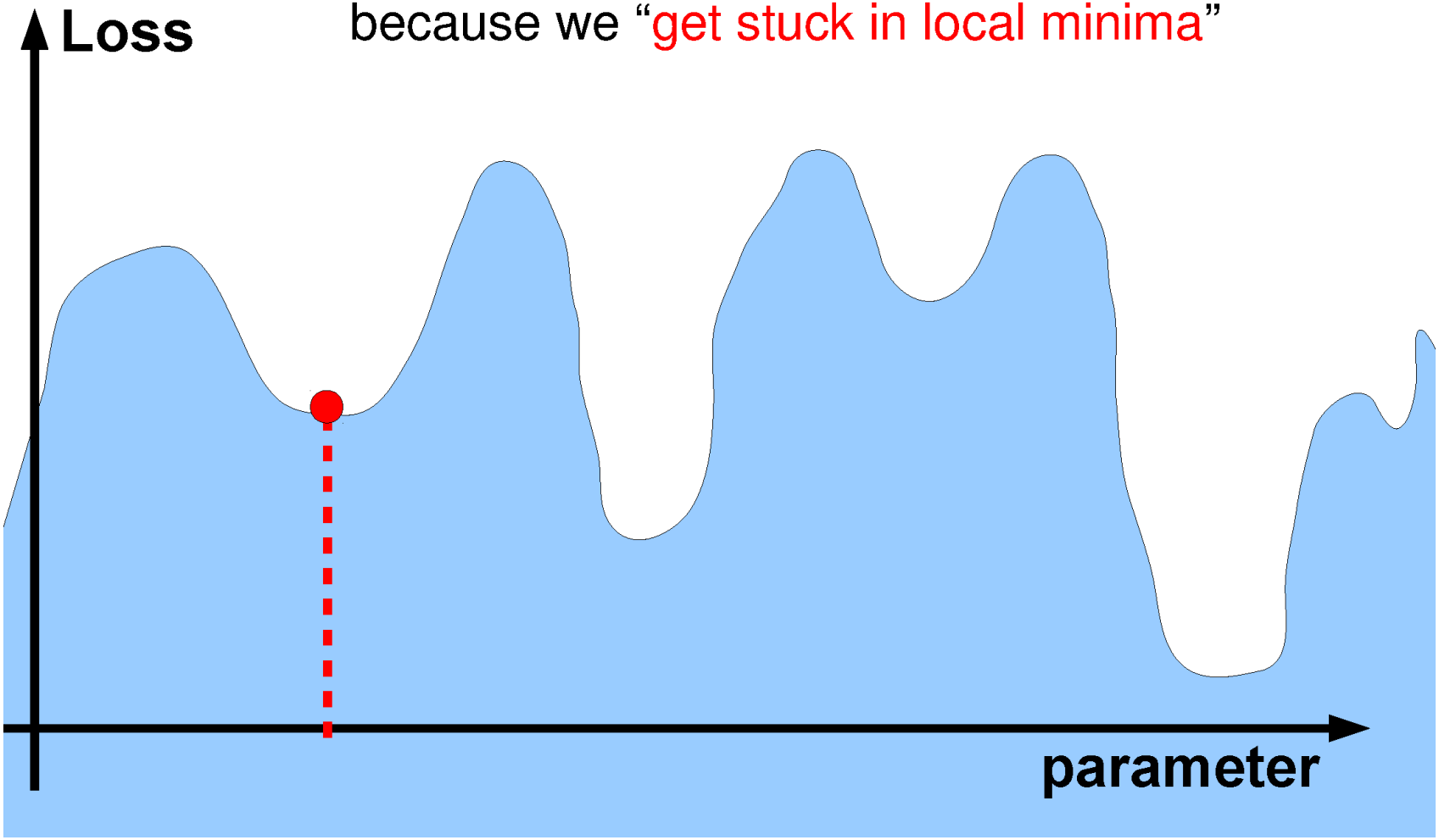
Compare with SIFT Descriptor



Slide Courtesy: Rob Fergus

ConvNets: till 2012

Common wisdom: training does not work because we "get stuck in local minima"



Slide Courtesy: Marc'Aurelio Ranzato

ConvNets: today

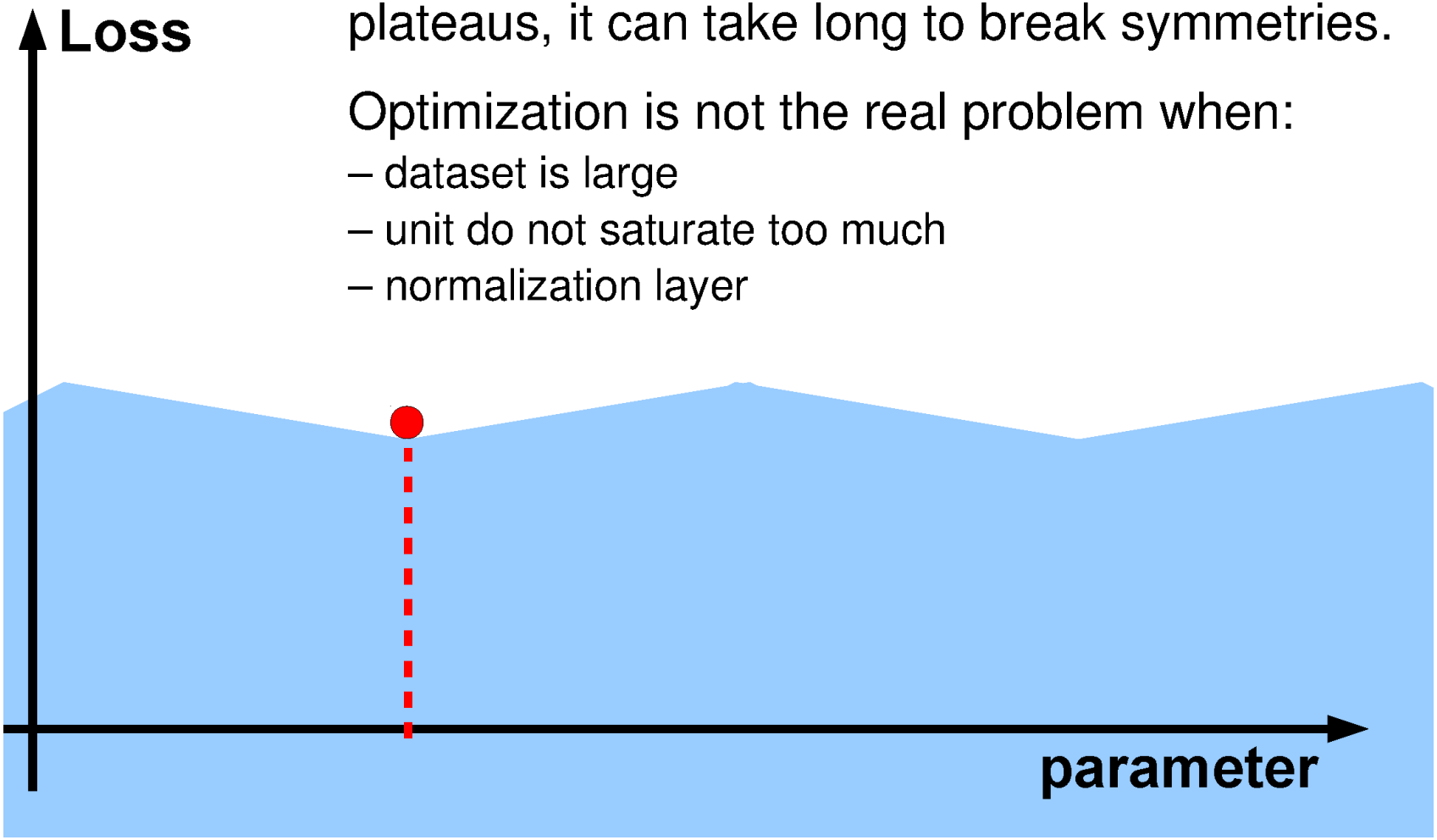
Local minima are all similar, there are long plateaus, it can take long to break symmetries.

Optimization is not the real problem when:

– dataset is large

– unit do not saturate too much

– normalization layer



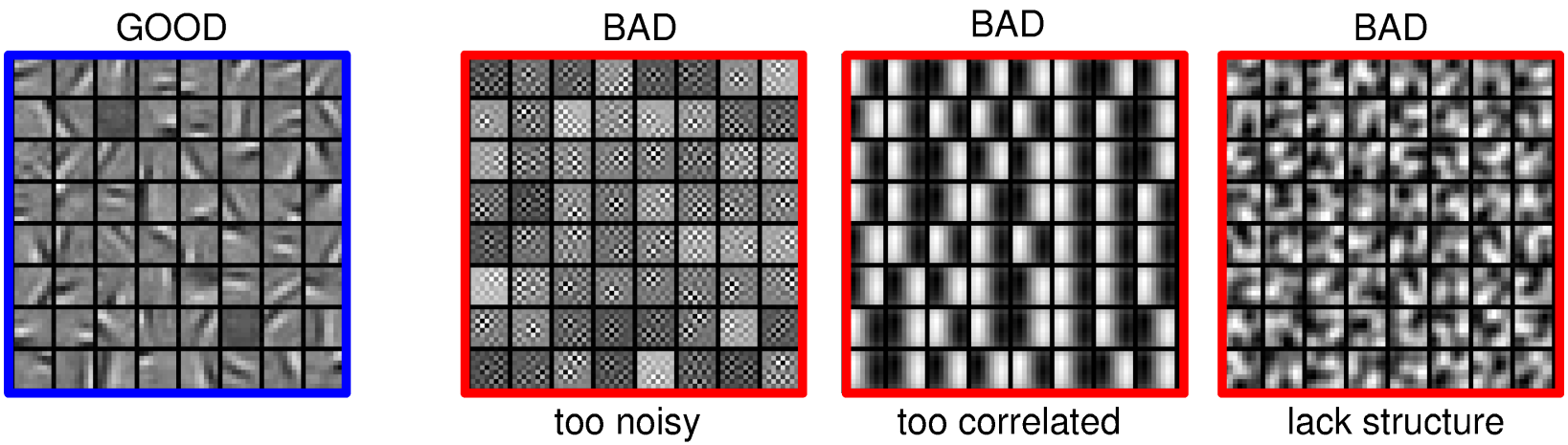
Slide Courtesy: Marc'Aurelio Ranzato

Choosing the Architecture

- Task dependent
- Cross-validation
- $[Convolution \rightarrow Pooling]* +$ Fully Connected
- The more data, the more layers and kernels
  1. Look at the number of parameters at each layer
  2. Look at the number of flops at each layer
- Computational resources

Good to Know

- Check gradients numerically by finite differences
- Visualize features
  feature maps need to be uncorrelated and have high variance
- Visualize parameters
- Measure error on both training and validation set
- Test on a small subset of the data and check the error $\rightarrow 0$



Good training: learned filters exhibit structure and are uncorrelated.

Slide Courtesy: Marc'Aurelio Ranzato

What if it does Not Work

- Training diverges:
  - Learning rate may be too large $\rightarrow$ decrease learning rate
  - Back Propagation is buggy $\rightarrow$a numerical gradient checking
- Parameters collapse $\rightarrow$ Check loss function:
  - Is it appropriate for the task you want to solve
  - Does it have degenerate solutions? Check "pull-up" term
- Network is underperforming
  - Compute flops and nr. params.$\rightarrow$ if too small, make net larger
  - Visualize hidden units/params $\rightarrow$ fix optmization
- Network is too slow
  - Compute flops and nr. params.
  - GPU, distributed framework, make net smaller

Slide Courtesy: Marc'Aurelio Ranzato

Summary

- Deep Learning = Learning Hierarchical Representations
- Supervised Learning: most successful set up today
- Optimization
  - Don't we get stuck in local minima? No, they are all the same
  - In large scale applications, local minima are even less of an issue
- Scaling
  - GPUs
  - Distributed framework
  - Better optimization techniques
- Generalization on small datasets (curse of dimensionality)
  - Data augmentation
  - Weight decay
  - Dropout
  - Unsupervised Learning
  - Multi-task Learning
  - Transfer Learning

Thank You

Q&A