

# Lecture 1: Neural Network Basics & Architecture Design

Xiangyu Zhang  
Face++ Researcher  
[zhangxiangyu@megvii.com](mailto:zhangxiangyu@megvii.com)

# Visual Recognition

A fundamental task in computer vision

- Classification
  - Object Detection
  - Semantic Segmentation
  - Instance Segmentation
  - Key point Detection
  - VQA
- ...



# Why Recognition Difficult?



Pose



Occlusion



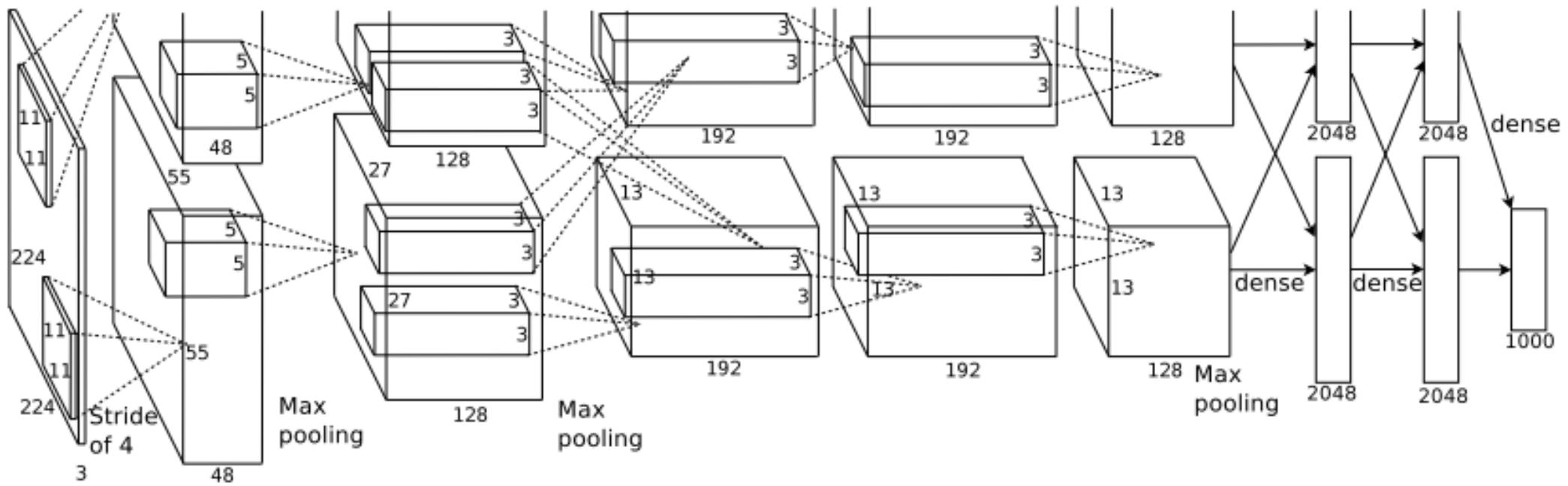
Multiple  
Objects



Inter-class  
Similarity

# Any Silver Bullet?

- Deep Neural Networks



# Outline

- Neural Network Basics
- Architecture Design

# PART 1: Neural Network Basics

- Motivation
- Deep neural networks
- Convolutional Neural Networks (CNNs)

# PART 1: Neural Network Basics

- Motivation
- Deep neural networks
- Convolutional Neural Networks (CNNs)

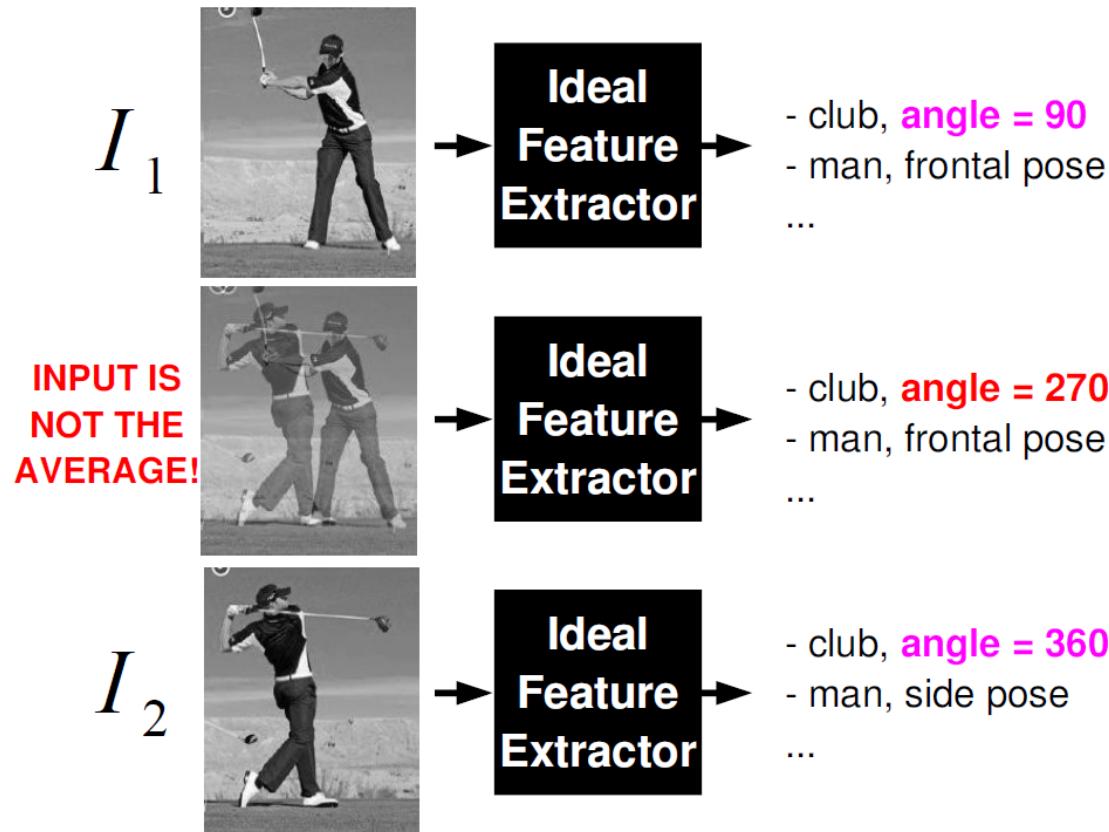
# Features for Recognition



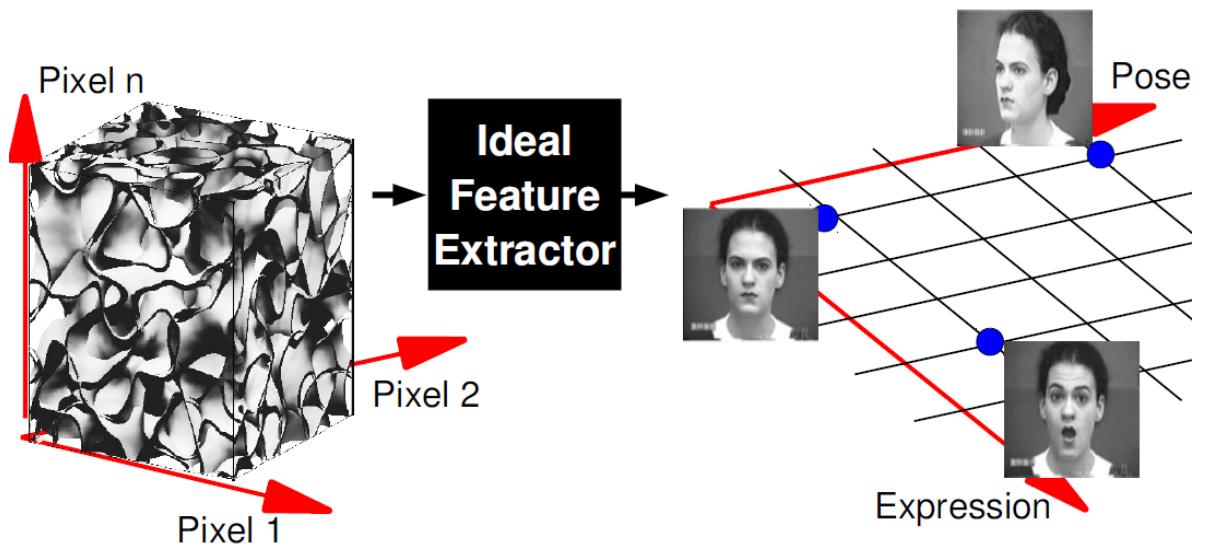
**Ideal  
Feature  
Extractor**

- window, top-left
- clock, top-middle
- shelf, left
- drawing,middle
- statue, bottom left
- ...
- hat, bottom right

# Nonlinear Features vs. Linear Classifiers

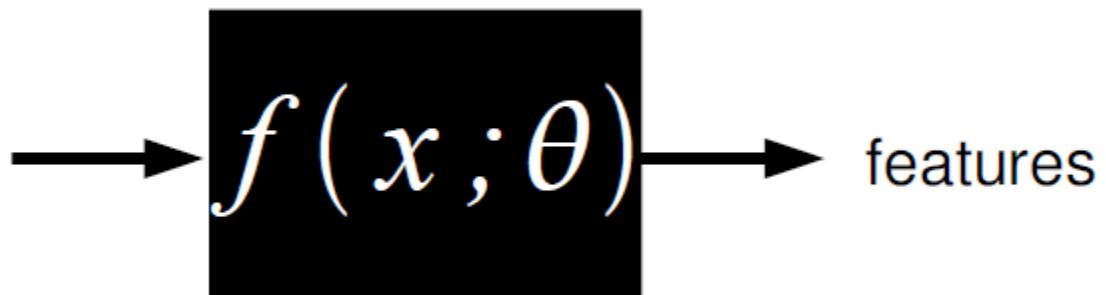


Feature extractor should be nonlinear!



# Learning Non-Linear Features

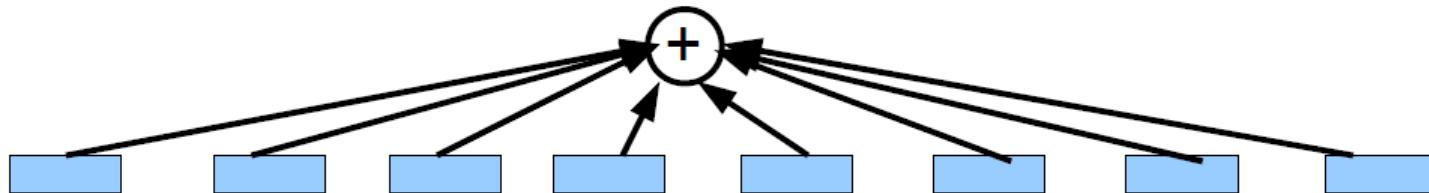
- Q: which class of non-linear functions shall we consider?



# Shallow or Deep

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

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

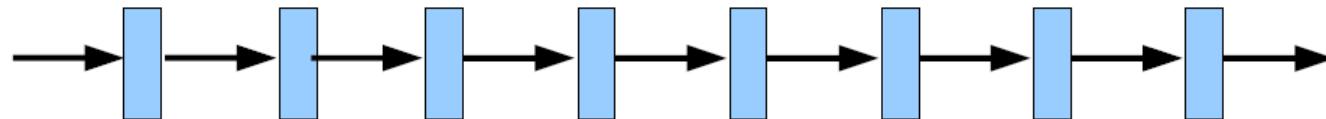


Shallow

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

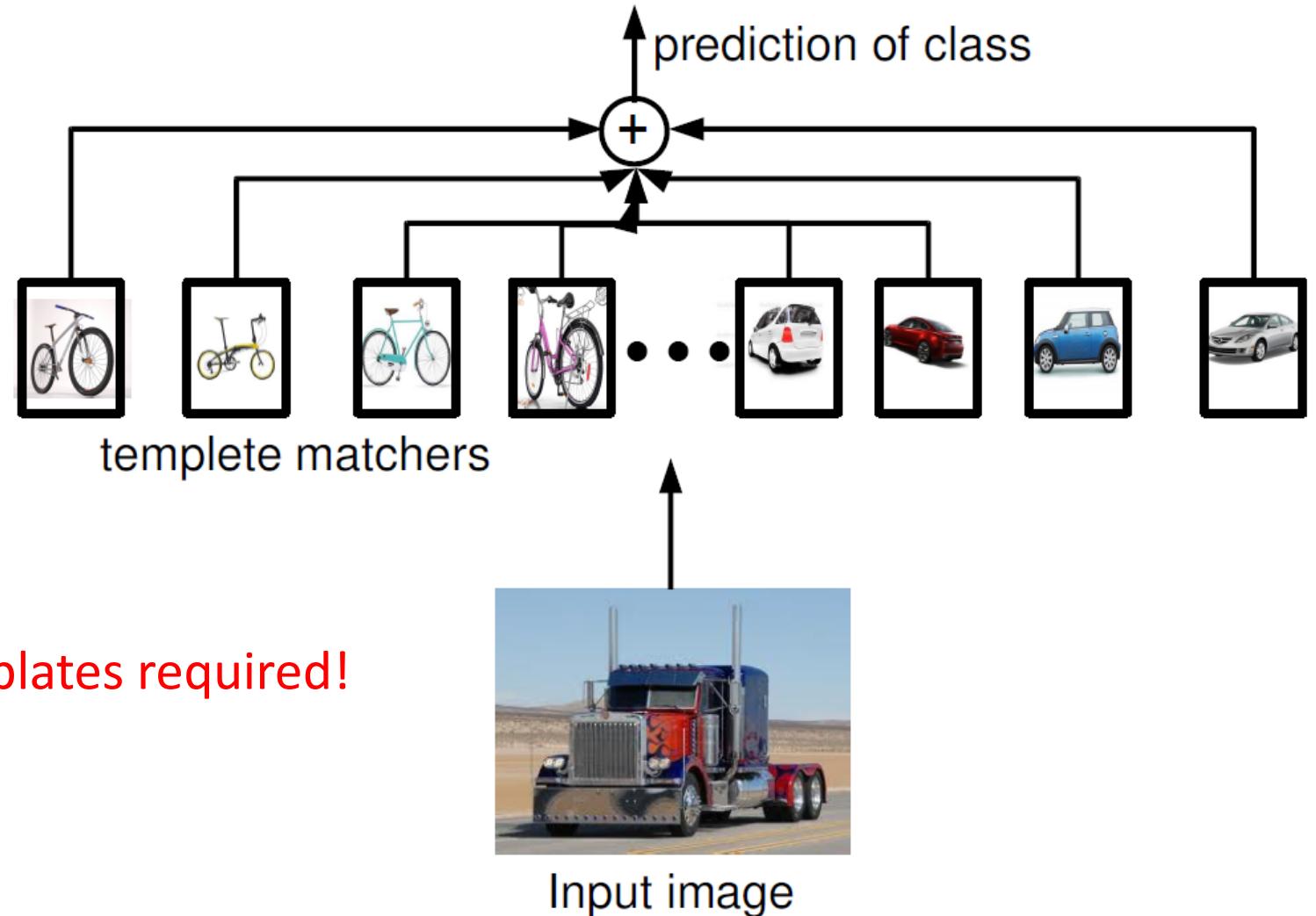
Deep

Face++ 旷视



# Linear Combination

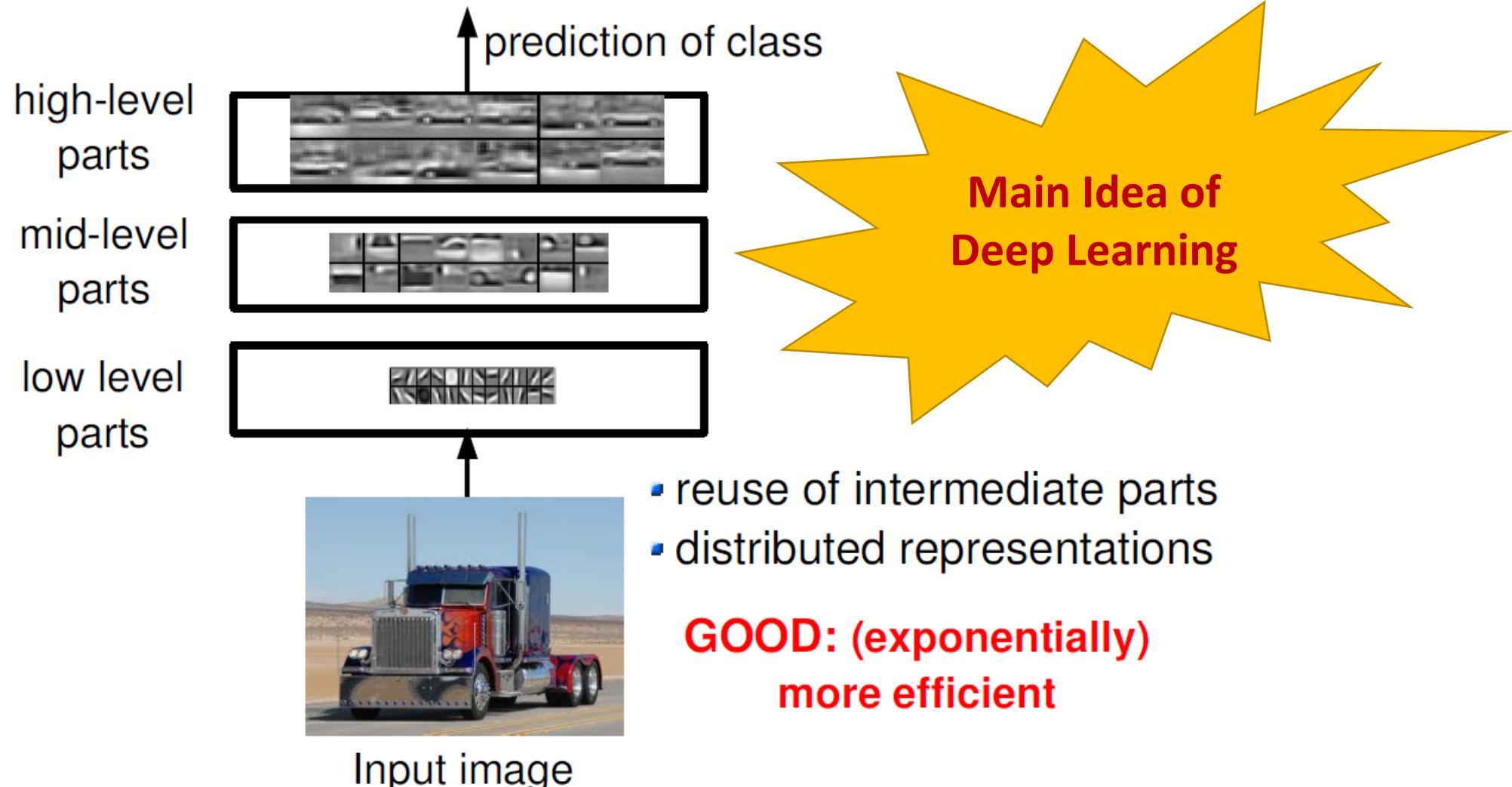
- Kernel learning
- Boosting
- ...



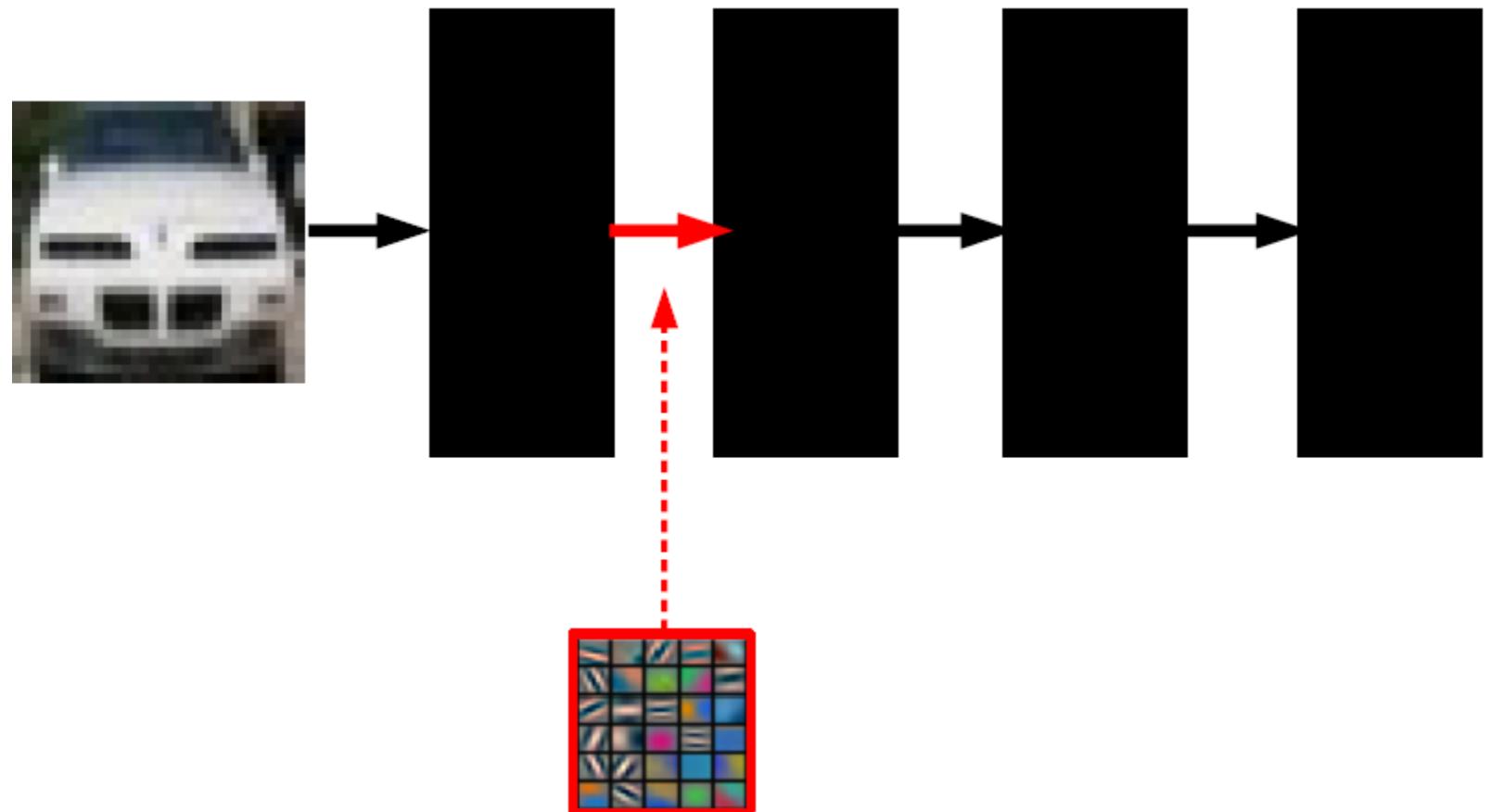
Drawbacks:

Exponential number of templates required!

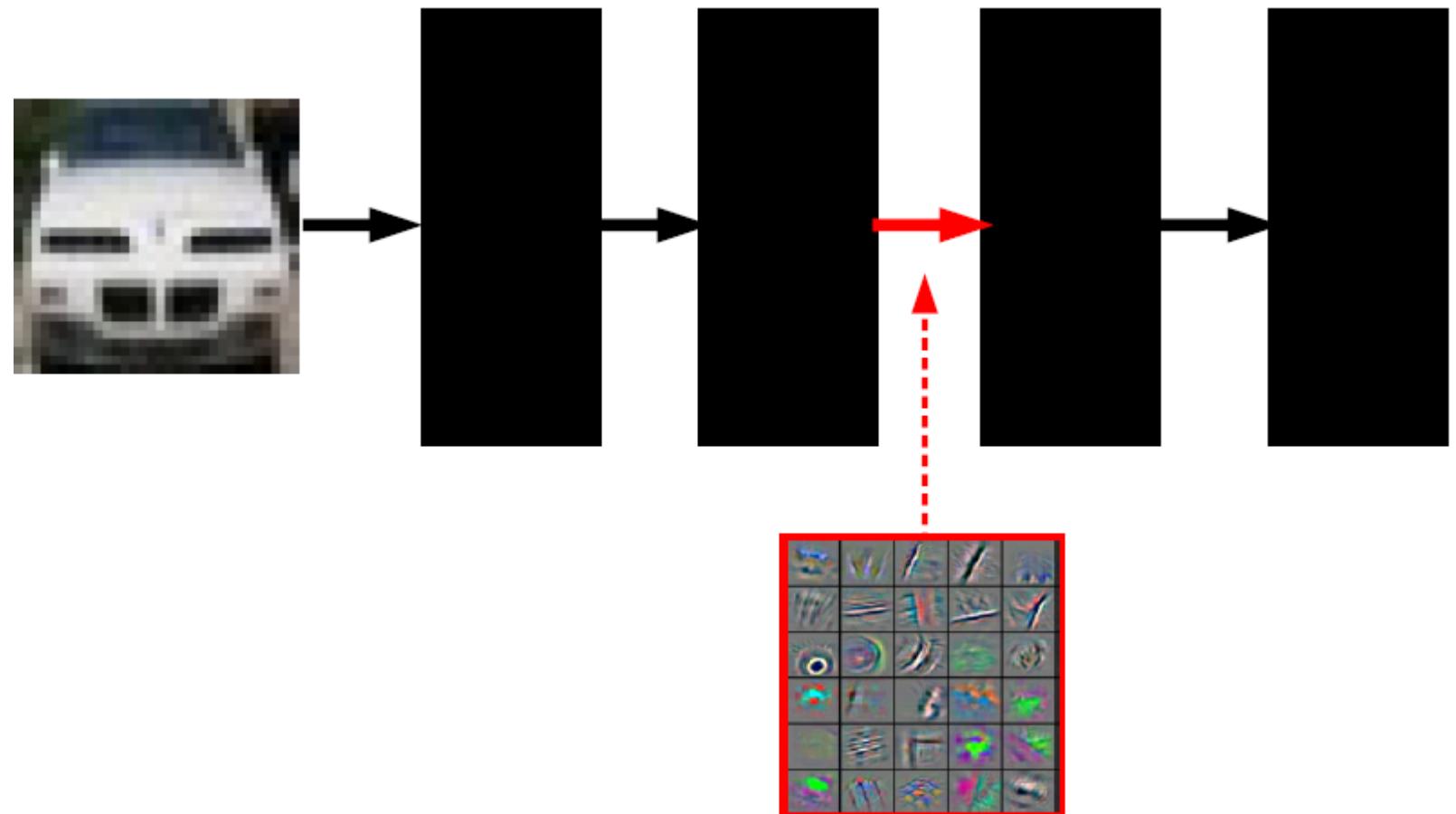
# Composition



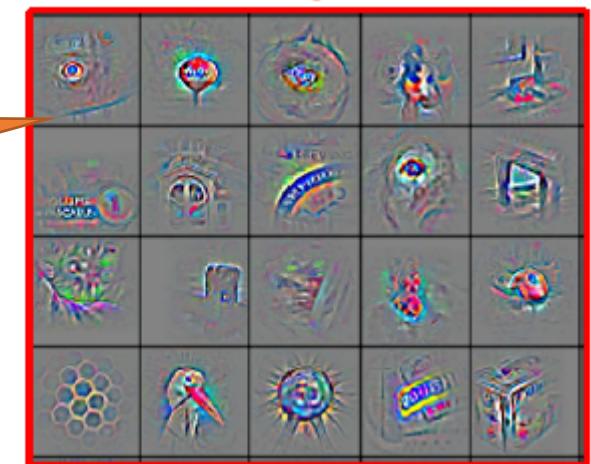
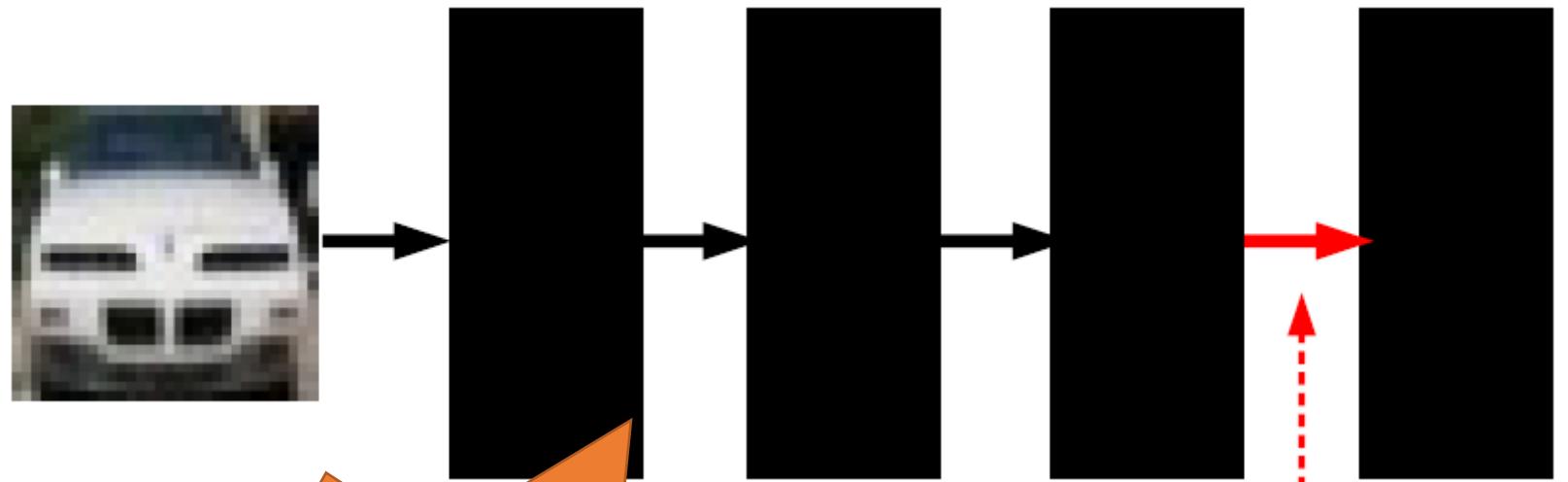
# Concepts Reuse in Deep Learning



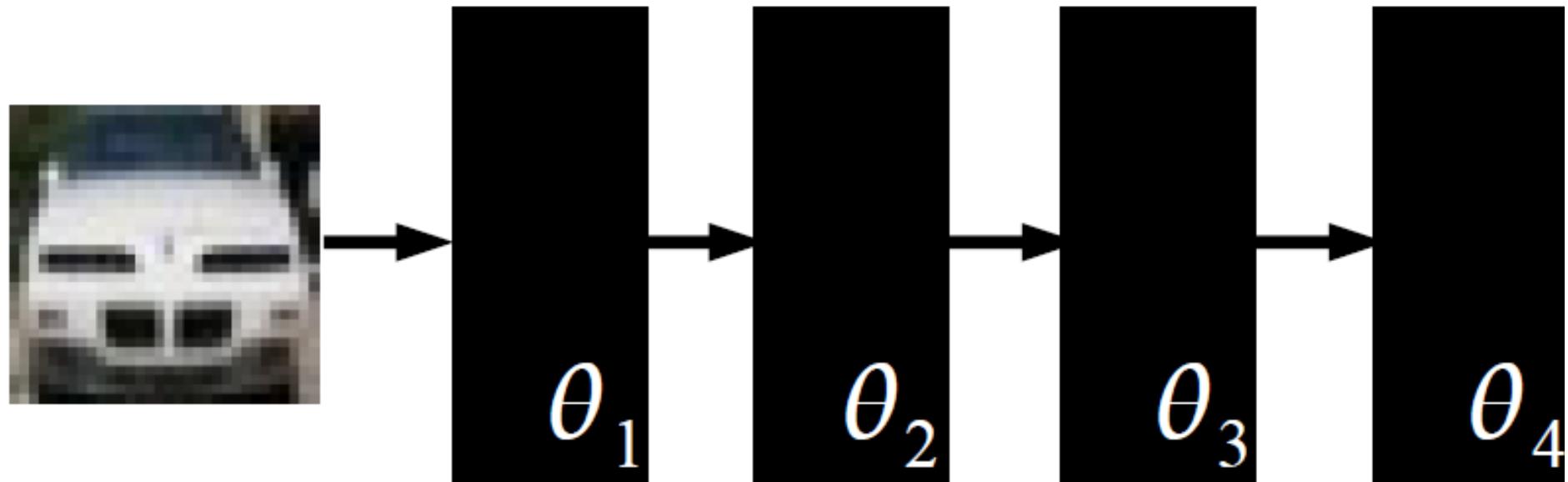
# Concepts Reuse in Deep Learning (cont'd)



# Concepts Reuse in Deep Learning (cont'd)



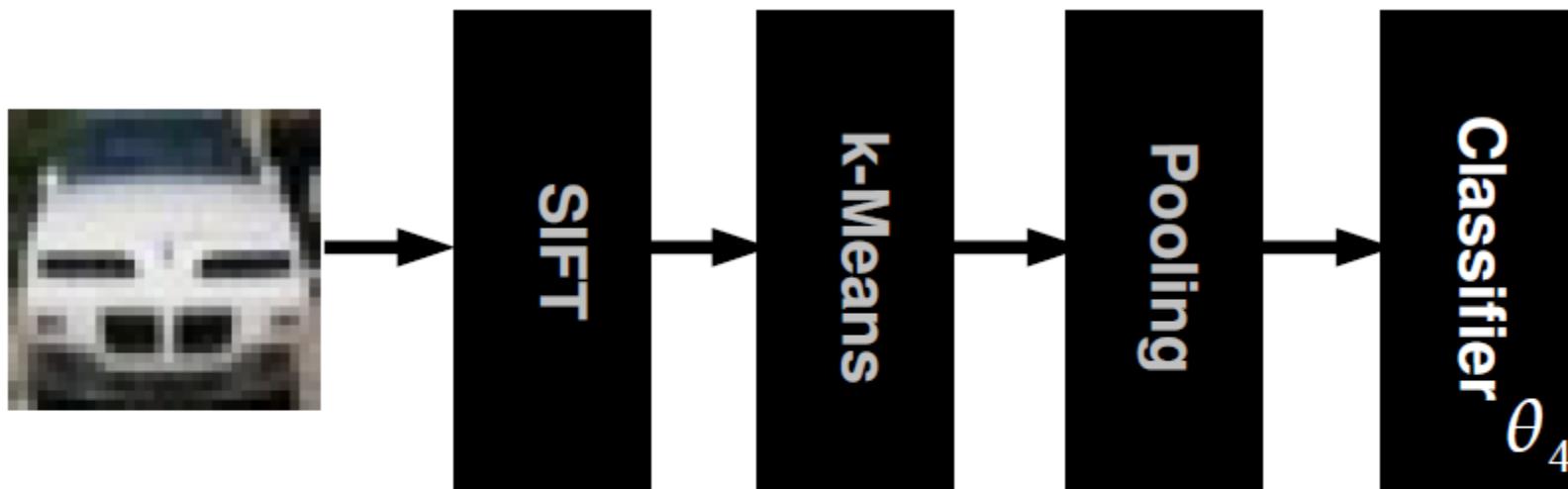
# Deep Learning Framework



A problem:

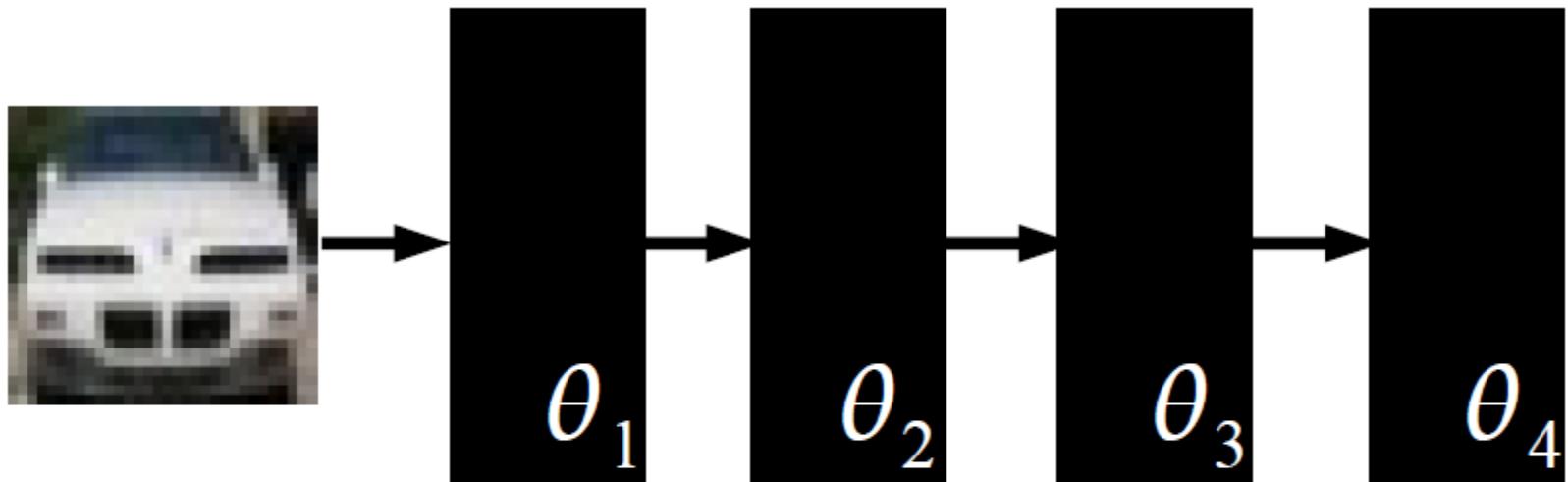
Optimization is difficult: non-convex, non-linear system

# Deep Learning Framework (cont'd)



**Solution #1:** freeze first N-1 layer (engineer the features)  
It makes it **shallow!**

# Deep Learning Framework (cont'd)



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

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

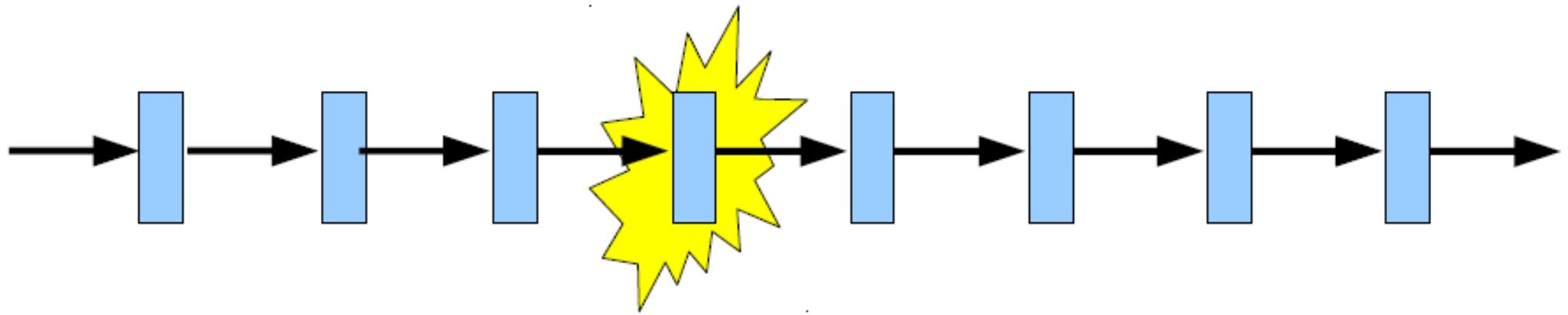
# Summary: Key Ideas of Deep Learning

- ❖ We need nonlinear system
- ❖ We need to learn it from data
- ❖ Build feature hierarchies (function composition)
- ❖ End-to-end learning

# PART 1: Neural Network Basics

- Motivation
- Deep neural networks
- Convolutional Neural Networks (CNNs)

# How to Build Deep Network?



“Neuron” or “Layer” Design

# Shallow Cases

- Linear Case:
- SVM

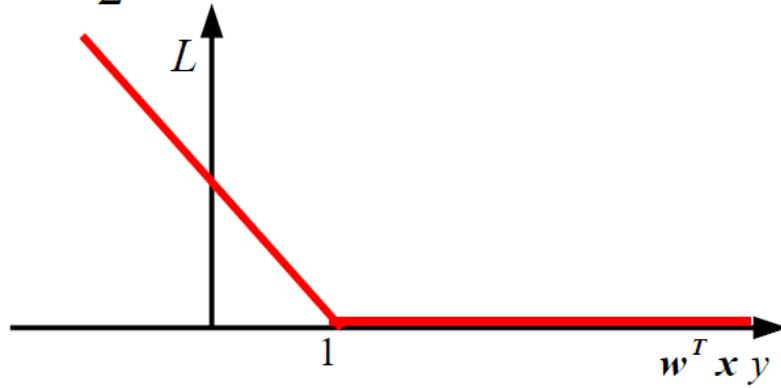
Input:  $\mathbf{x} \in R^D$

Binary label:  $y \in \{-1, +1\}$

Parameters:  $\mathbf{w} \in R^D$

Output prediction:  $\mathbf{w}^T \mathbf{x}$

Loss:  $L = \frac{1}{2} \|\mathbf{w}\|^2 + \lambda \max [0, 1 - \mathbf{w}^T \mathbf{x} y]$



# Shallow Cases (cont'd)

- Linear Case:
- Logistic Regression

Linear transformation  
+ nonlinear activation

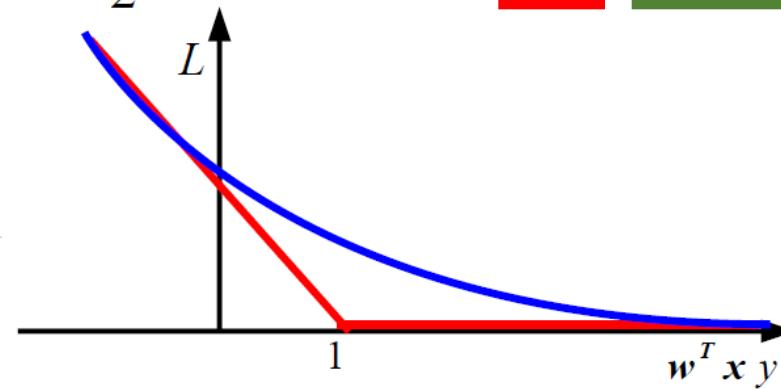
Input:  $\mathbf{x} \in R^D$

Binary label:  $y \in [-1, +1]$

Parameters:  $\mathbf{w} \in R^D$

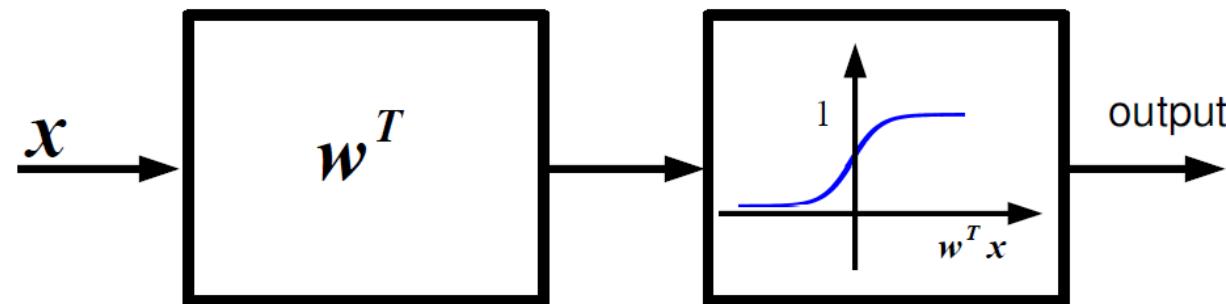
Output prediction:  $p(y=1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}}}$

Loss:  $L = \frac{1}{2} \|\mathbf{w}\|^2 + \lambda \log(1 + \exp(-\mathbf{w}^T \mathbf{x} y))$

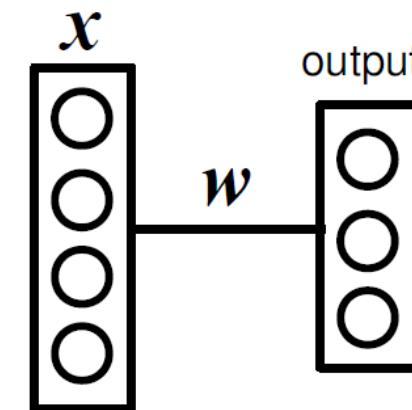
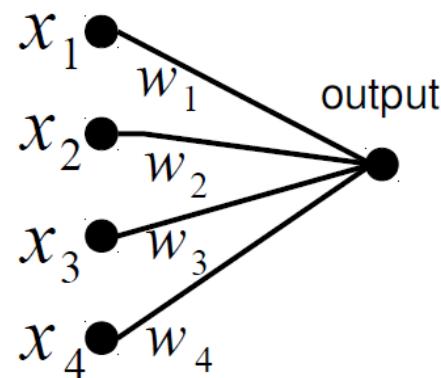


# Neuron Design

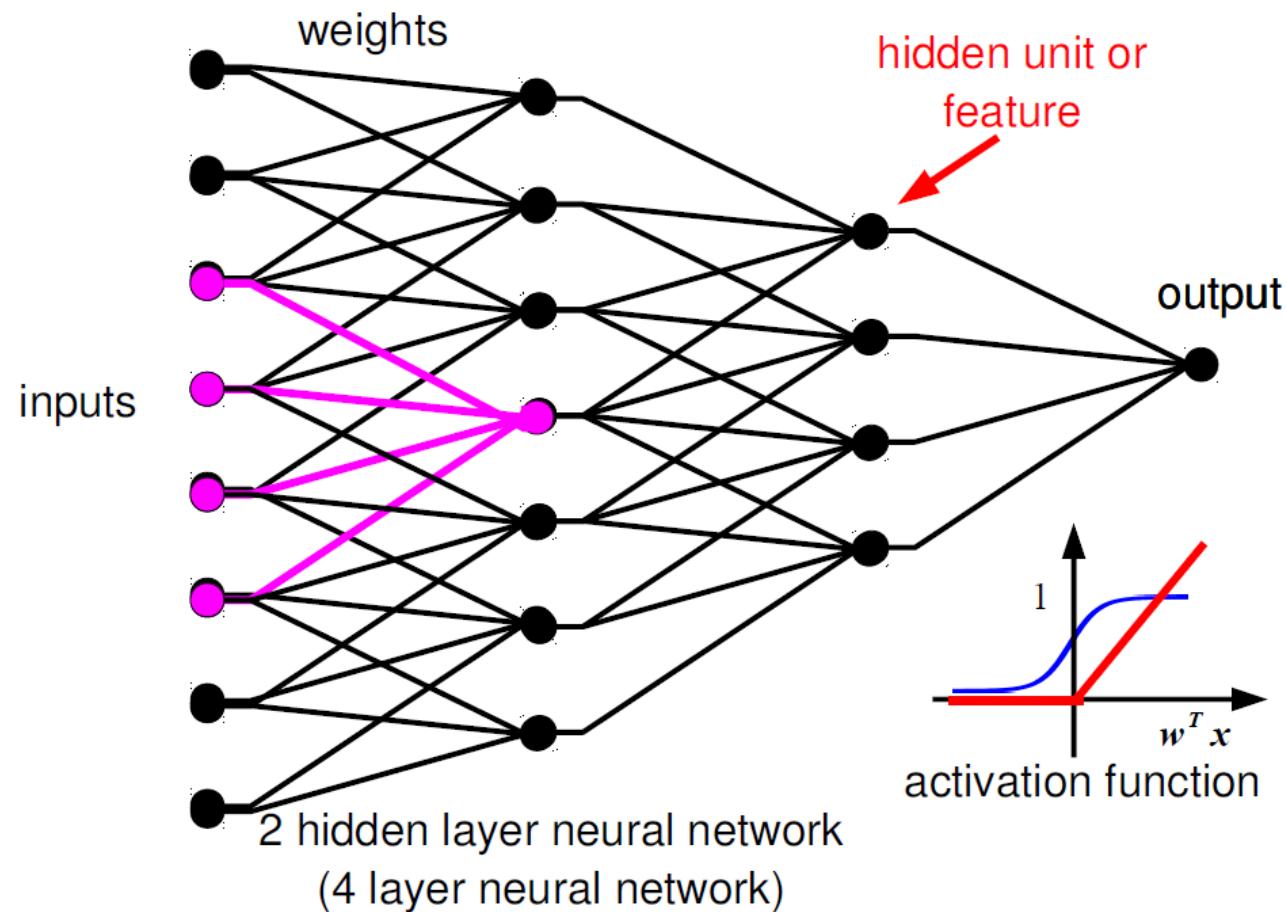
Single Neuron:



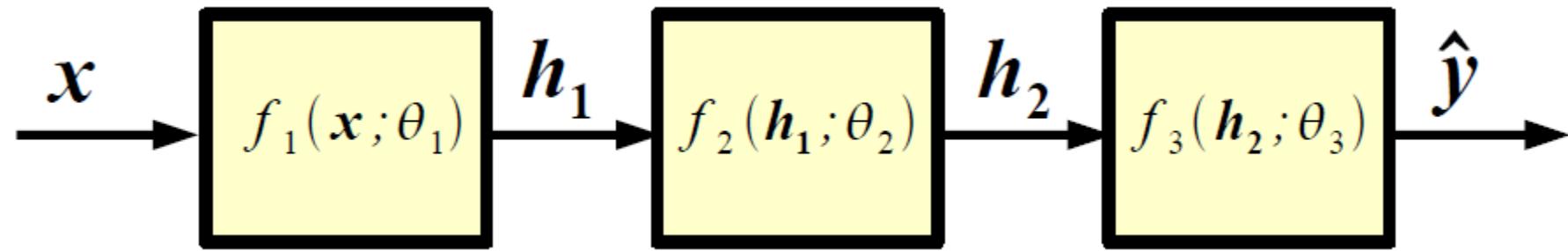
Linear Projection +  
Nonlinear Activation



# Deep Neuron Network



# Deep Neural Network (cont'd)

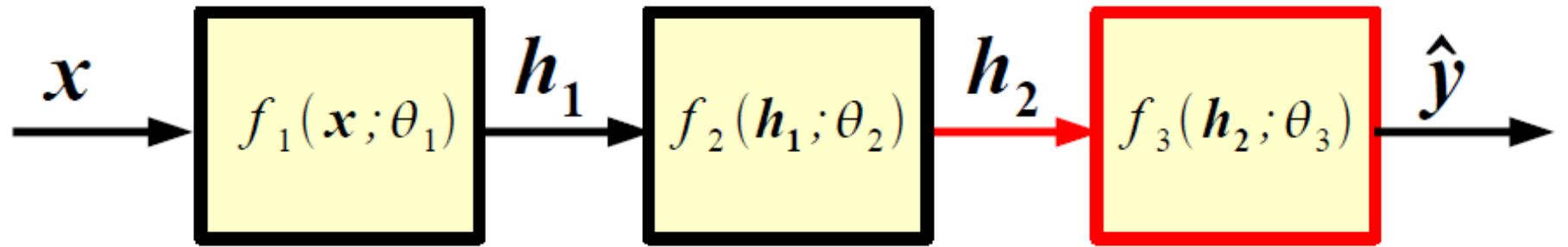


How to determine the parameters?

# Gradient-based Training

- For each iteration:
  1. Forward Propagation
  2. Backward Propagation
  3. Update Parameters (Optimization)

# Forward Propagation (FPROP)

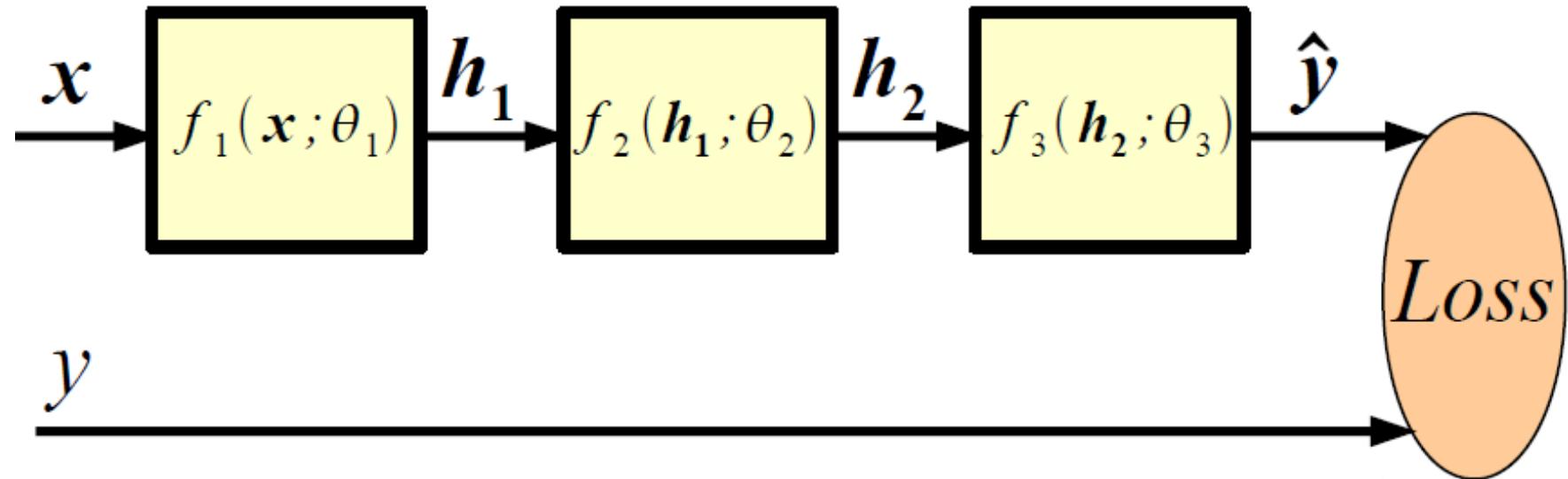


This is the typical processing at test time. At training time, we need to compute an error measure and tune the parameters to decrease the error.

For instance,

$$\hat{y}_i = p(\text{class} = i | \mathbf{x}) = \frac{e^{W_{3i}\mathbf{h}_2 + b_{3i}}}{\sum_k e^{W_{3k}\mathbf{h}_2 + b_{3k}}}$$

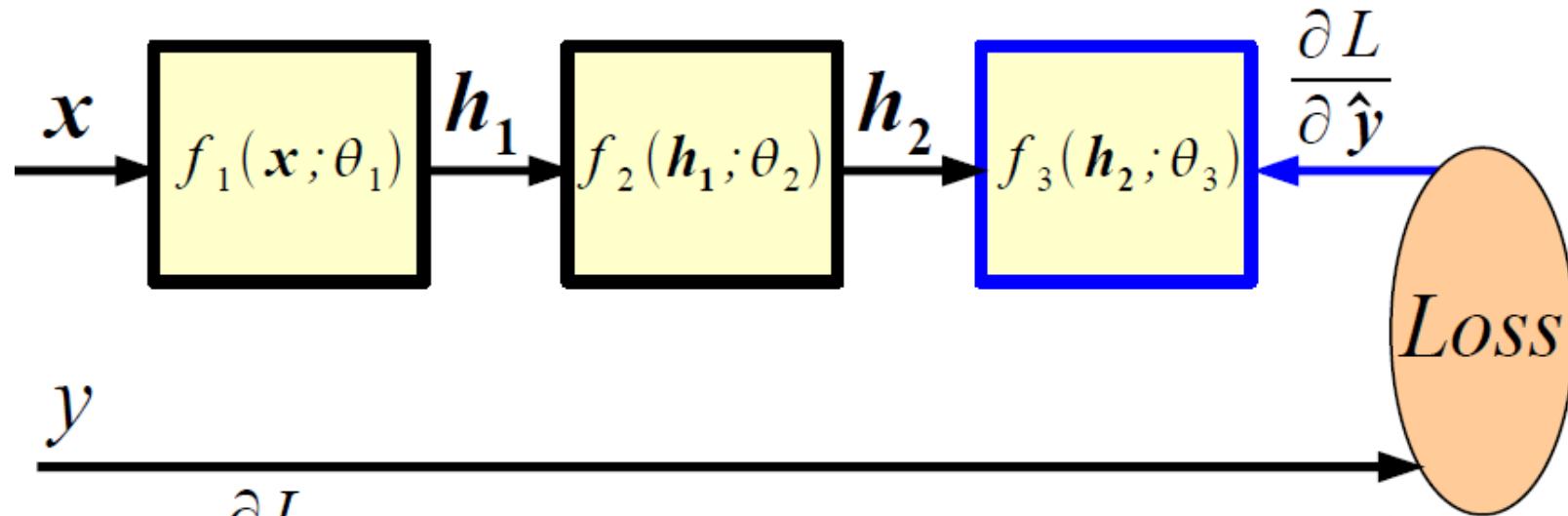
# Loss Function



**Q:** how to tune the parameters to decrease the loss?

**A:** If loss is (a.e.) differentiable we can compute gradients. We can use chain-rule, a.k.a. **back-propagation**, to compute the gradients w.r.t. parameters at the lower layers.

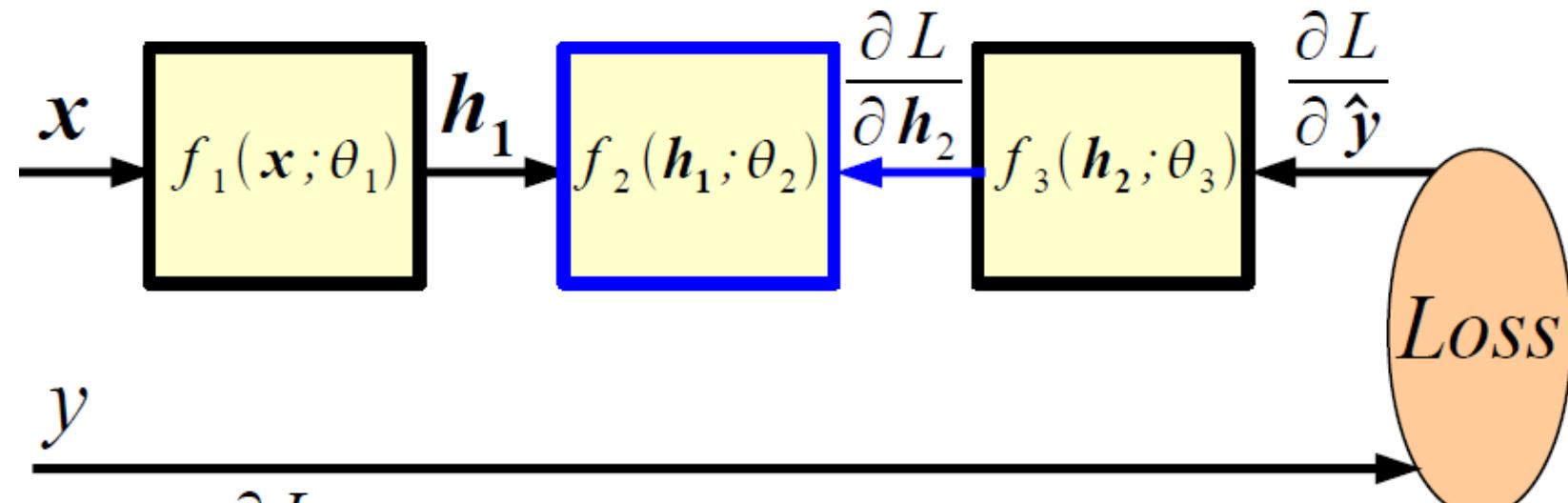
# Backward Propagation (BPROP)



Given  $\frac{\partial L}{\partial \hat{y}}$  and assuming the Jacobian of each module is easy to compute, then we have:

$$\frac{\partial L}{\partial \theta_3} = (\hat{y} - y) \ h_2' \quad \frac{\partial L}{\partial h_2} = (\hat{y} - y) \ \theta_3'$$

# Backward Propagation (BPROP) (cont'd)

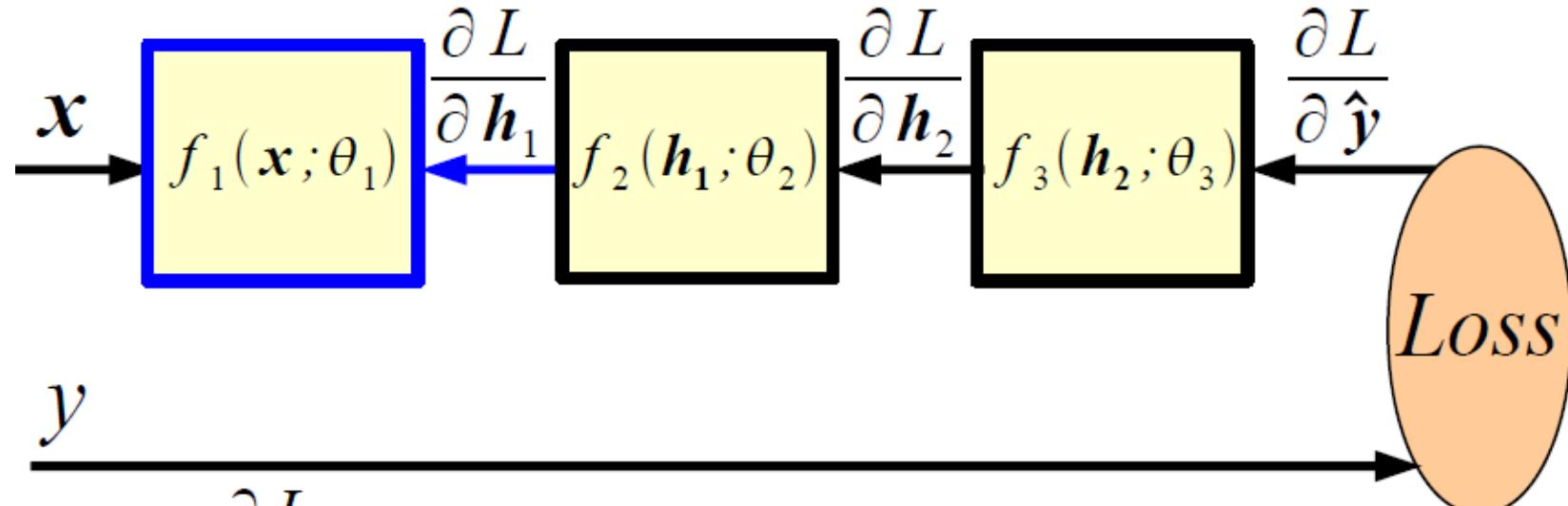


Given  $\frac{\partial L}{\partial h_2}$  we can compute now:

$$\frac{\partial L}{\partial \theta_2} = \frac{\partial L}{\partial h_2} \frac{\partial h_2}{\partial \theta_2}$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

# Backward Propagation (BPROP) (cont'd)



Given  $\frac{\partial L}{\partial \mathbf{h}_1}$  we can compute now:

$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \theta_1}$$

# Optimization

- Stochastic Gradient Descent (on mini-batches):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}, \eta \in R$$

- Stochastic Gradient Descent with Momentum:

$$\begin{aligned}\theta &\leftarrow \theta - \eta \Delta \\ \Delta &\leftarrow 0.9 \Delta + \frac{\partial L}{\partial \theta}\end{aligned}$$

# Summary: Key Ideas of Deep Neural Networks

- Neural Net = stack of feature detectors
- F-Prop / B-Prop
- Learning by SGD

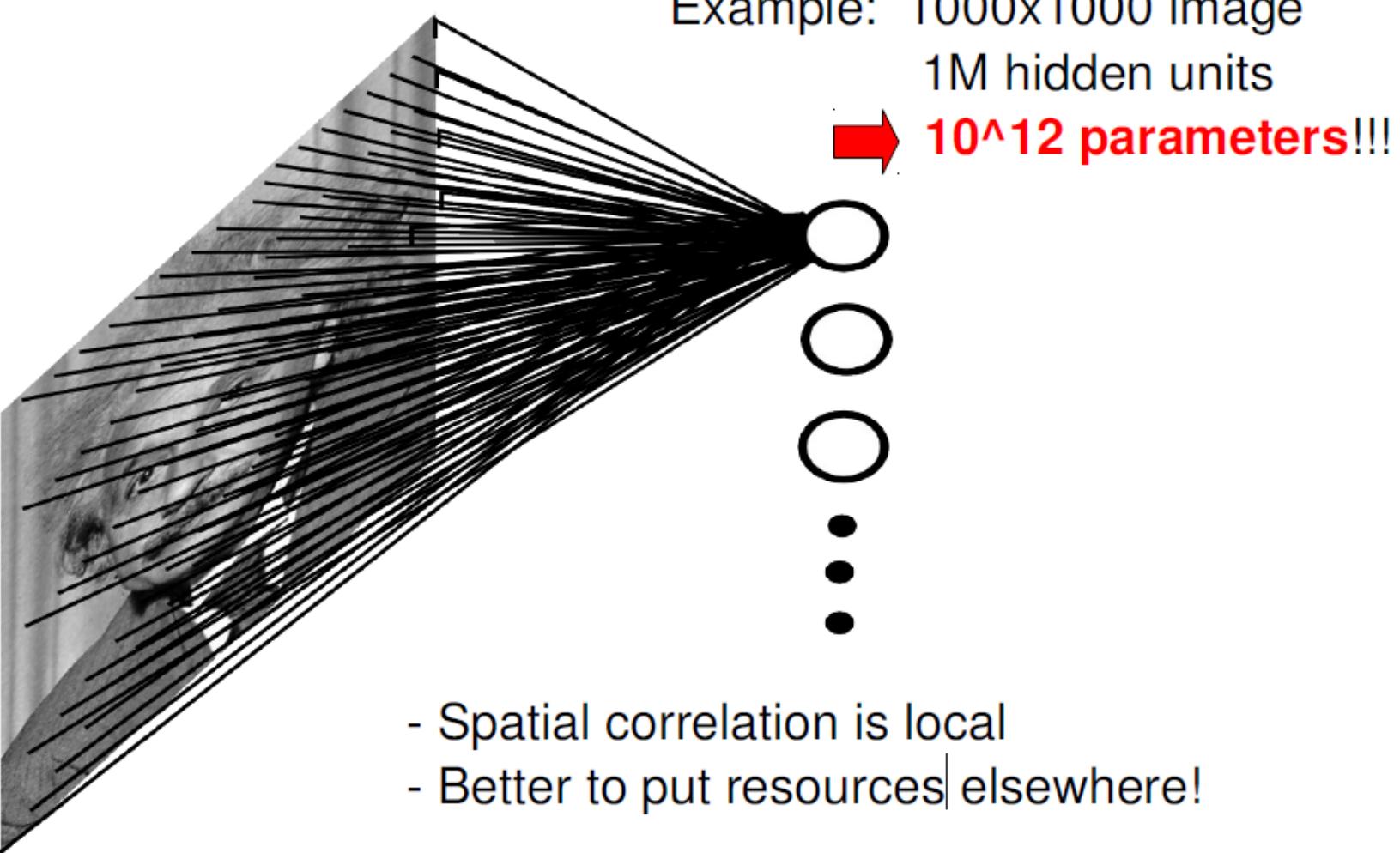
# PART 1: Neural Network Basics

- Motivation
- Deep neural networks
- **Convolutional Neural Networks (CNNs)**

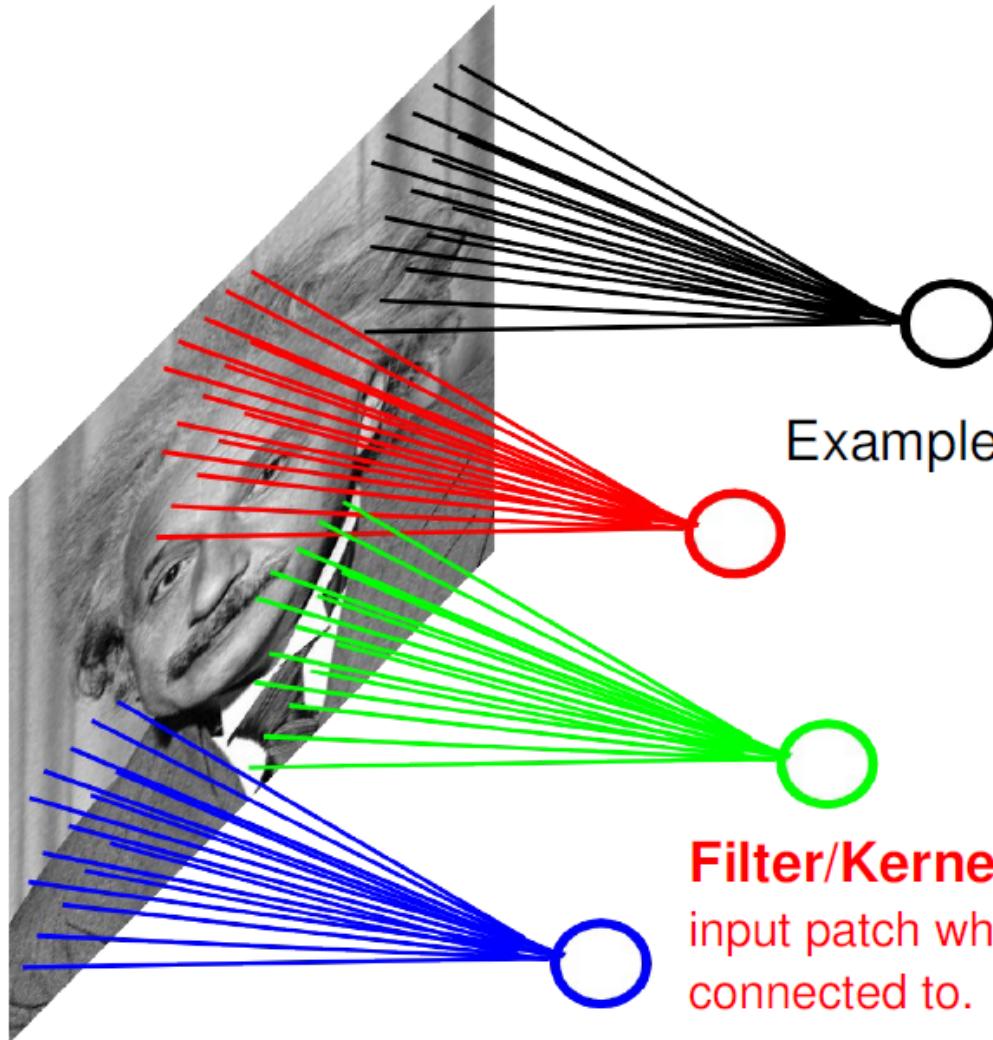
# Deep Neural Networks on Images

- How to apply a neural network on 2D or 3D inputs?

# Fully-connected Net



# Locally-connected Net



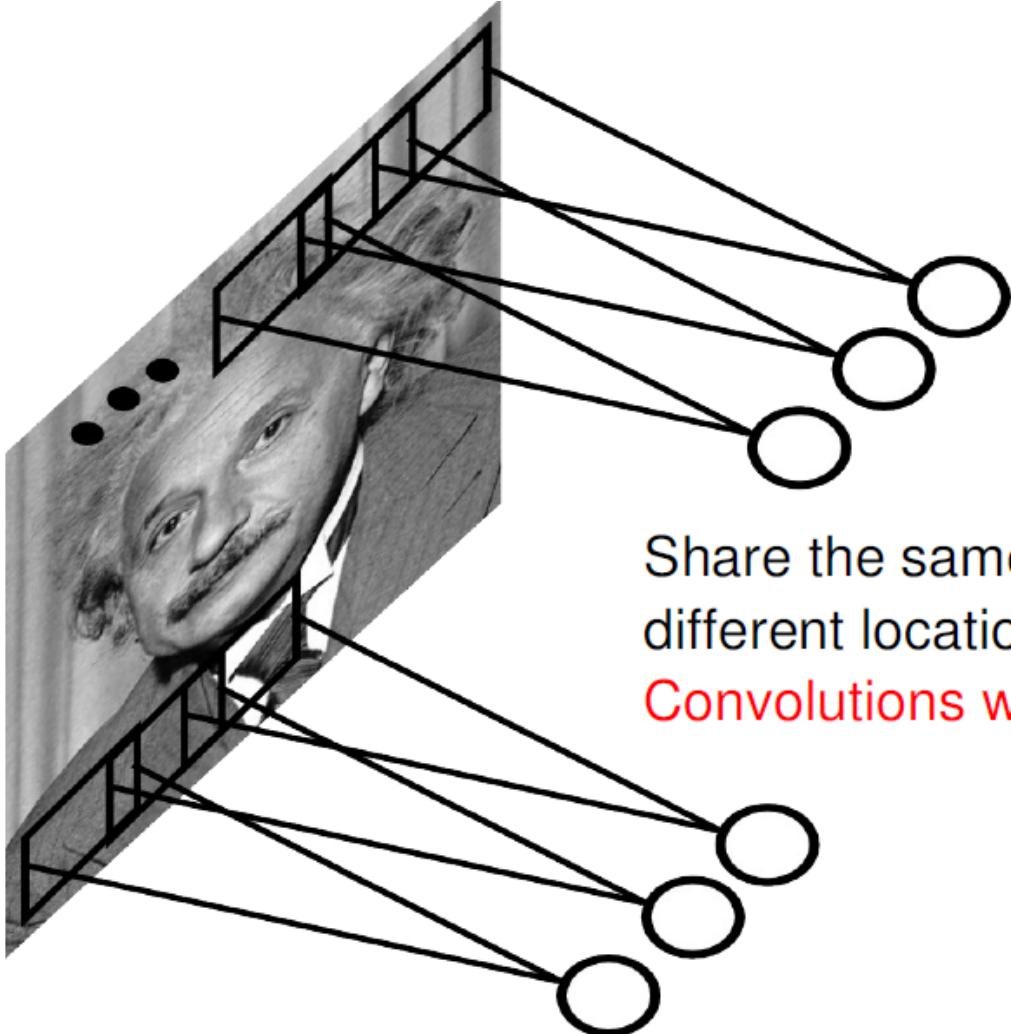
## STATIONARITY?

Statistics are similar at different locations (translation invariance)

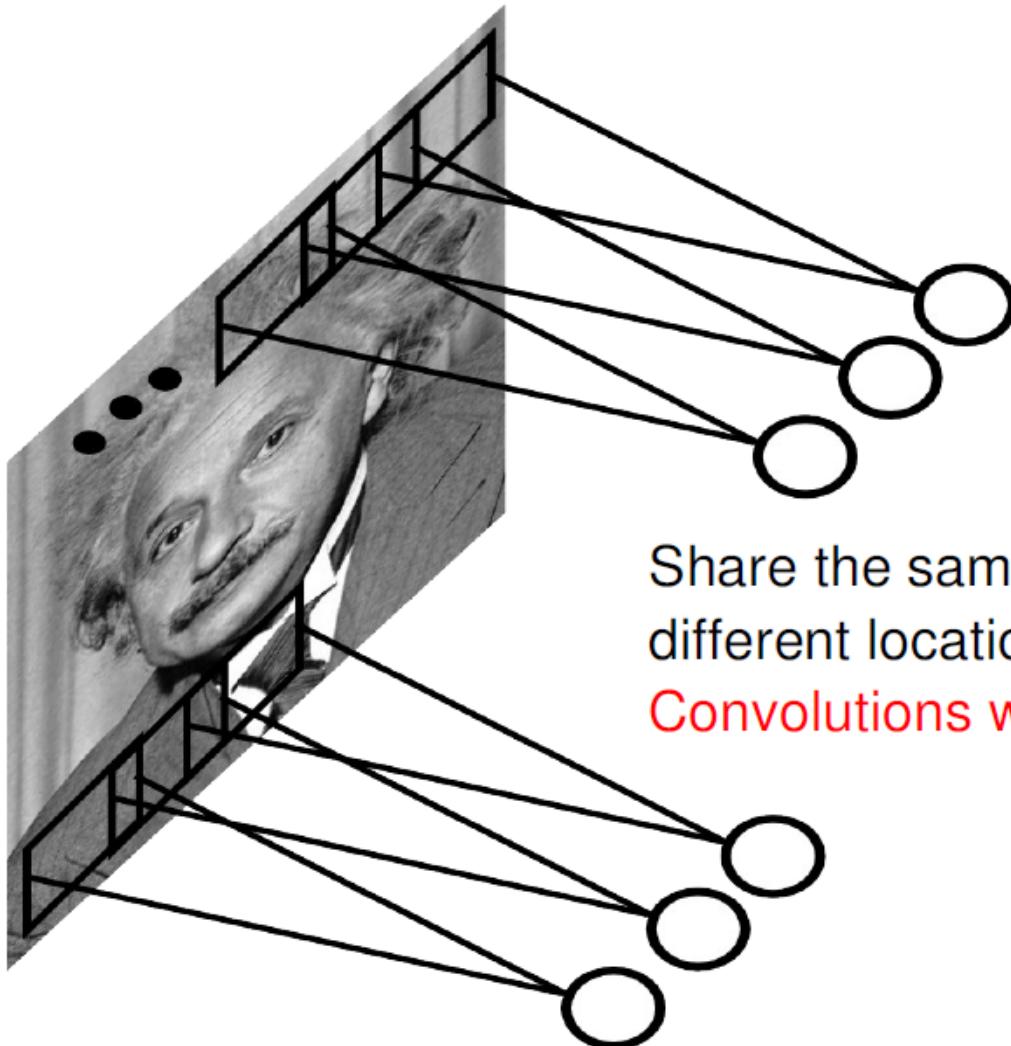
Example: 1000x1000 image  
1M hidden units  
Filter size: 10x10  
100M parameters

**Filter/Kernel/Receptive field:**  
input patch which the hidden unit is connected to.

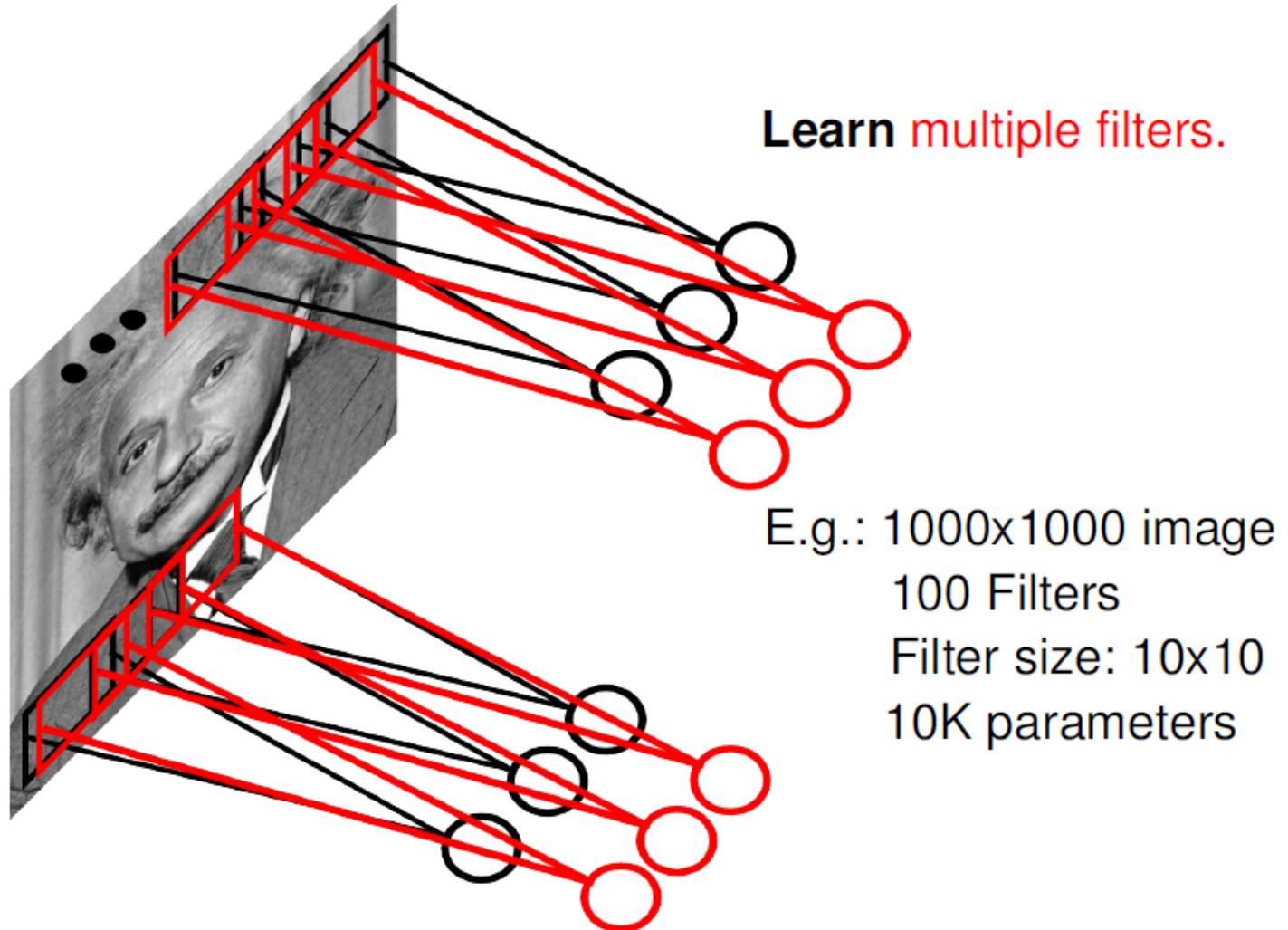
# Convolutional Net



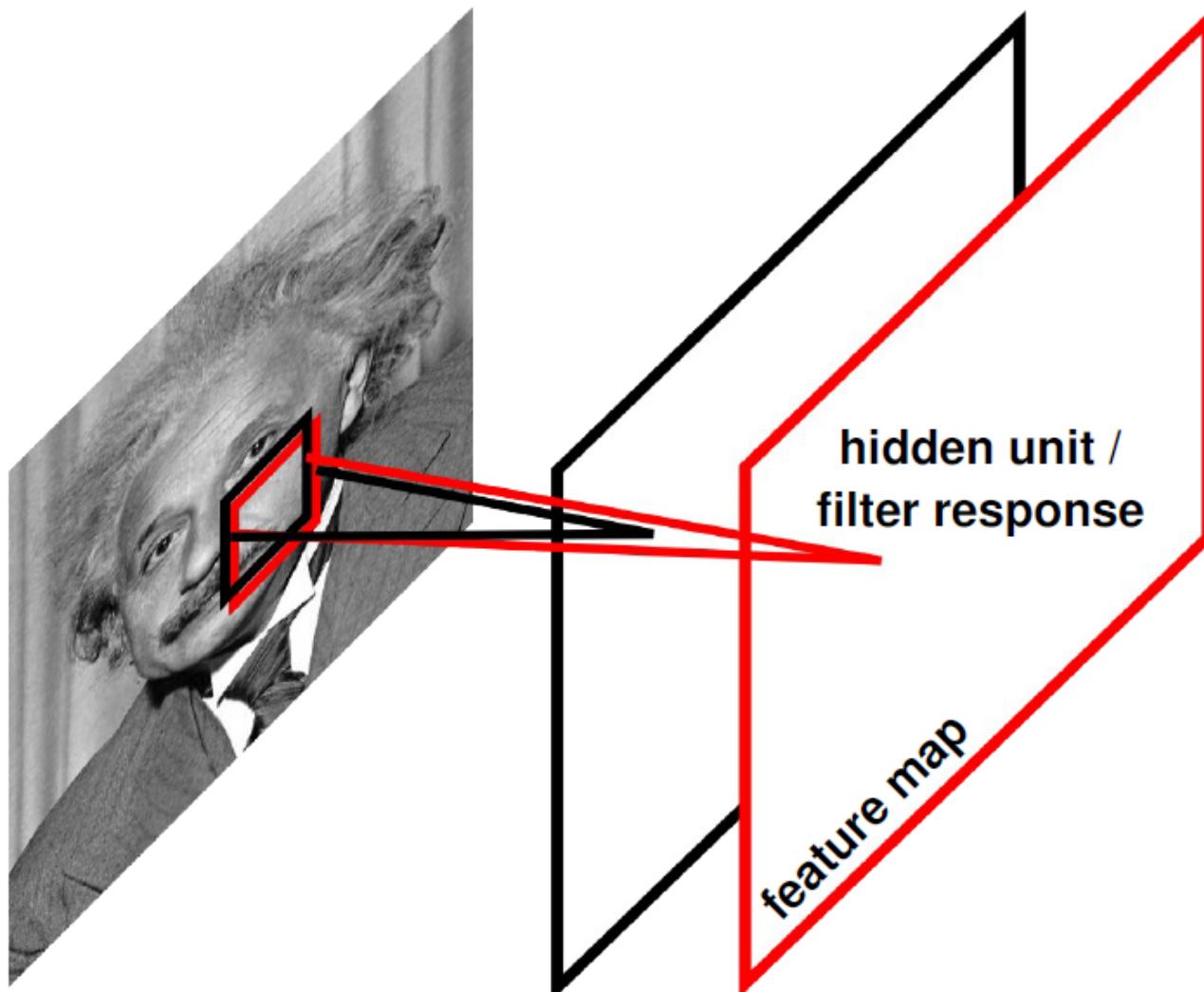
# Convolutional Net (cont'd)



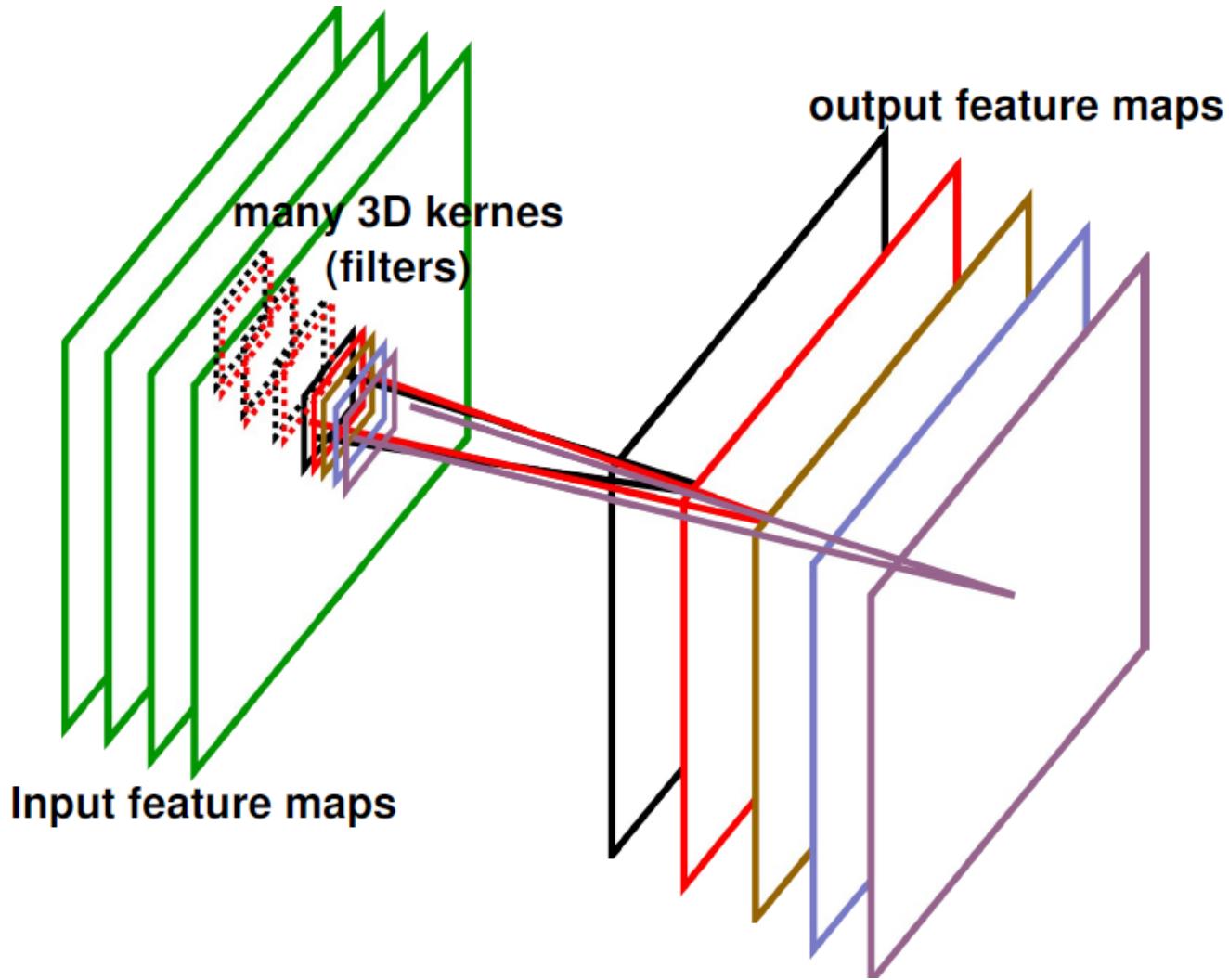
# Convolutional Net (cont'd)



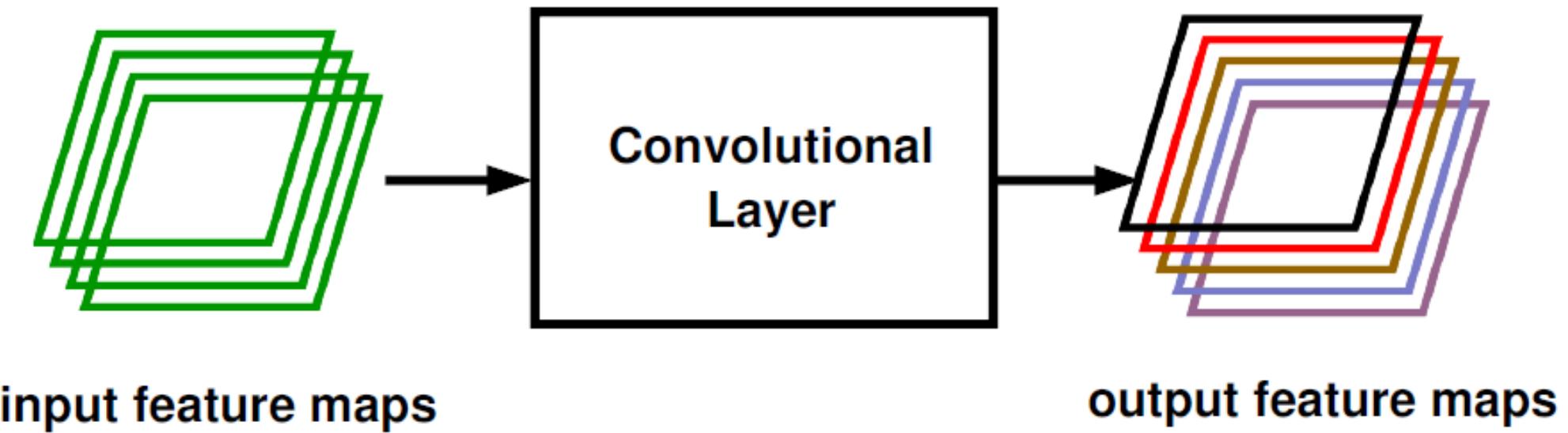
# Convolutional Net (cont'd)



# Convolutional Layer



# Convolutional Layer (cont'd)

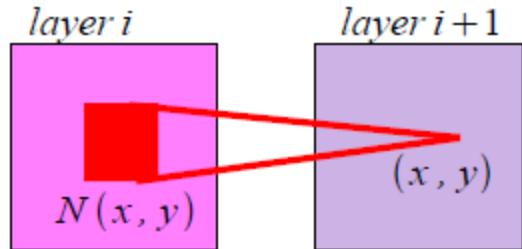


# Summary: Key Ideas of Convolutional Nets

- 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 hidden units
- This is called: **convolutional network**.

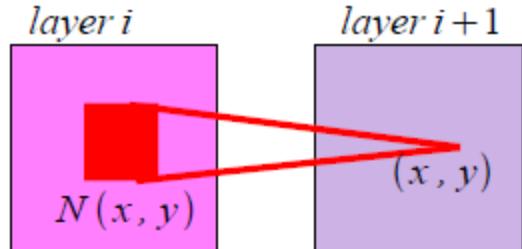
# Other Layers

- Over the years, some new modules have proven to be very effective when plugged into conv-nets:
  - **Pooling** (average, L2, max)



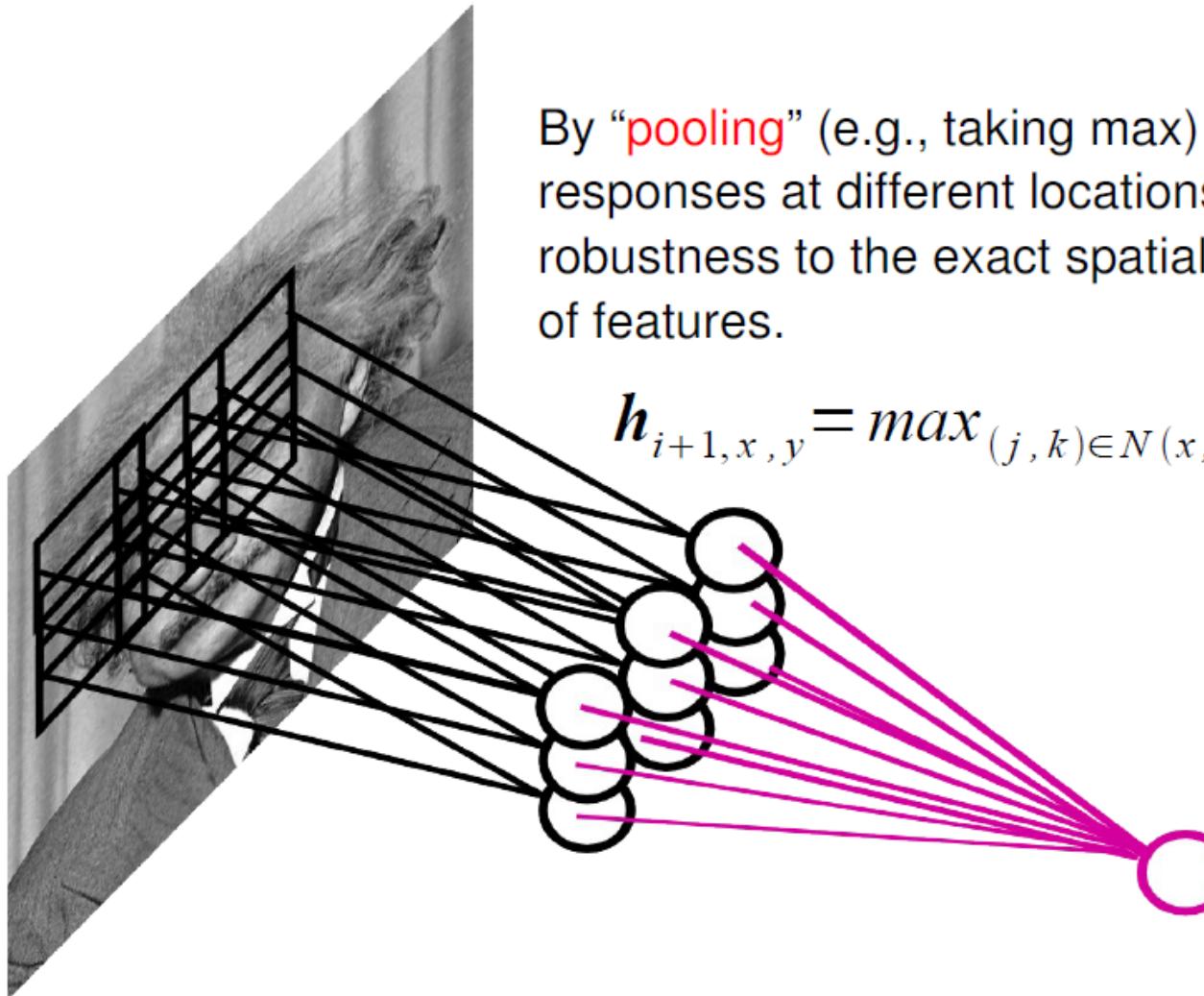
$$h_{i+1, x, y} = \max_{(j, k) \in N(x, y)} h_{i, j, k}$$

- **Local Contrast Normalization** (over space / features)



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

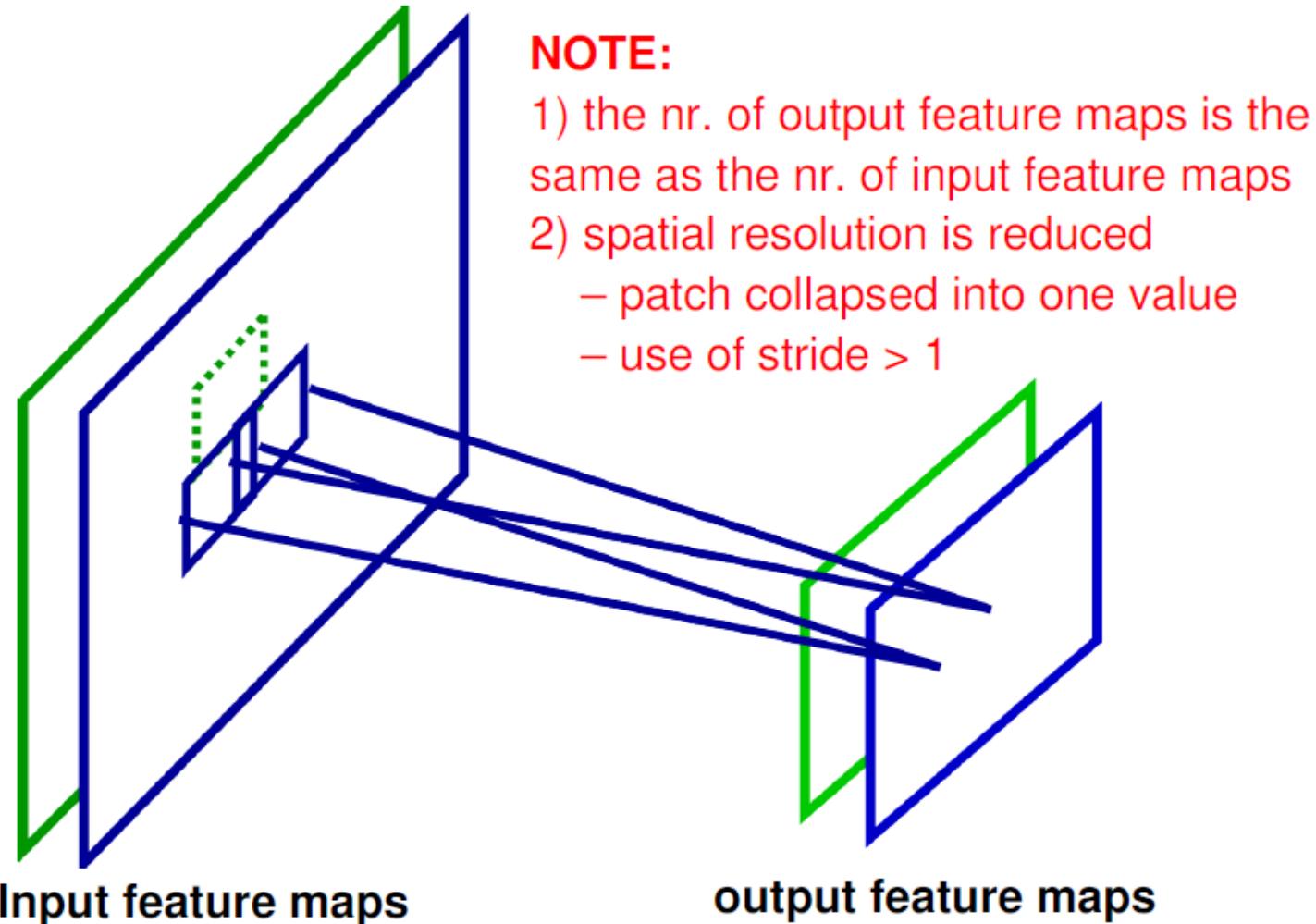
# Pooling Layer



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

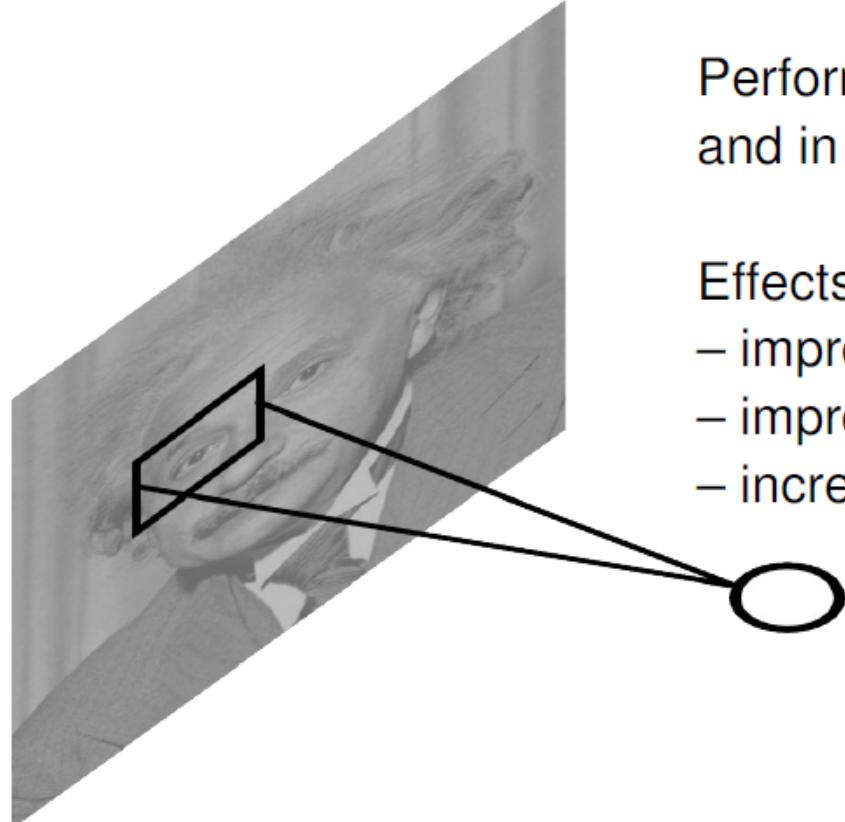
$$h_{i+1, x, y} = \max_{(j, k) \in N(x, y)} h_{i, j, k}$$

# Pooling Layer



# Local Contrast Normalization Layer

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

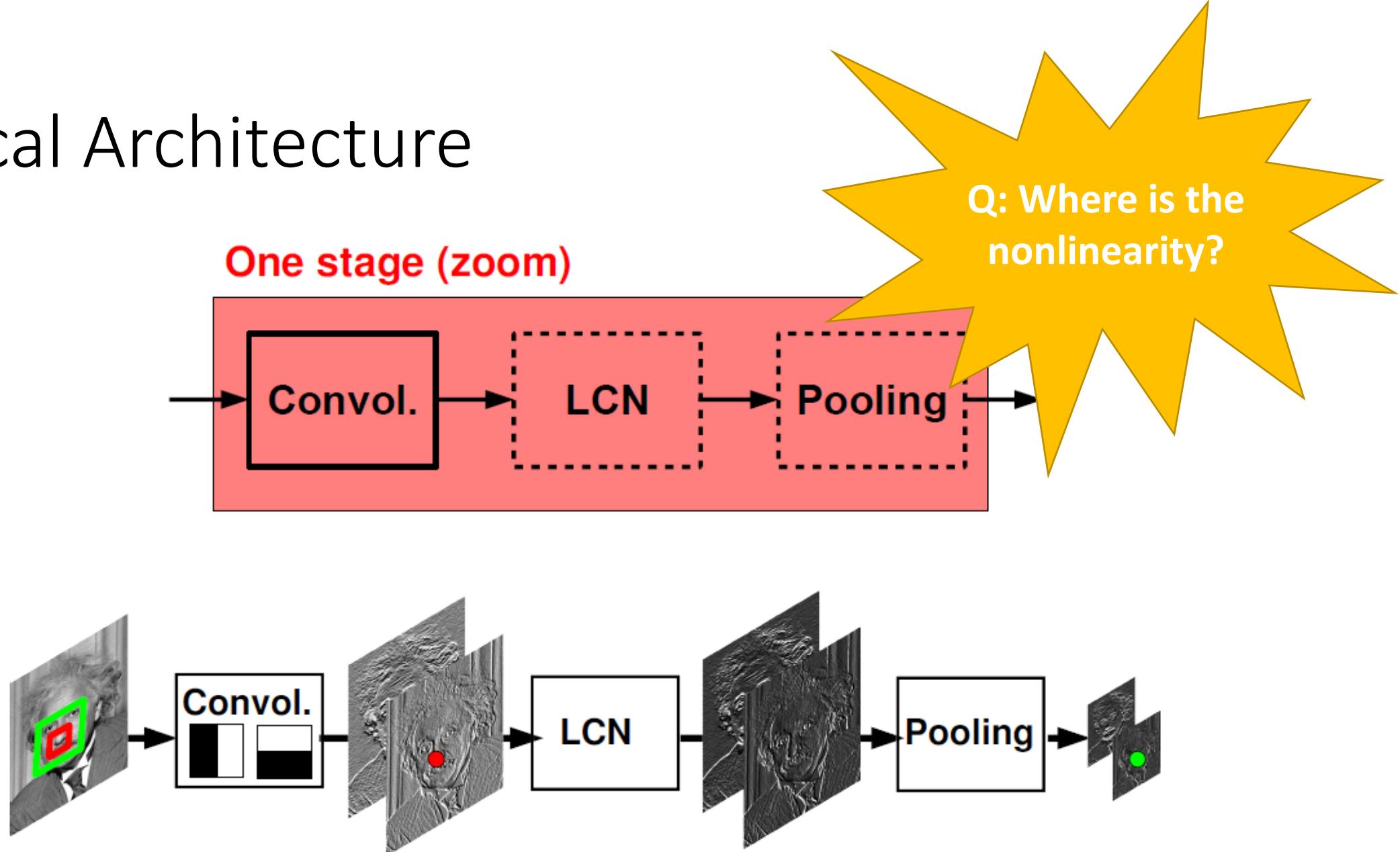


Performed also across features  
and in the higher layers.

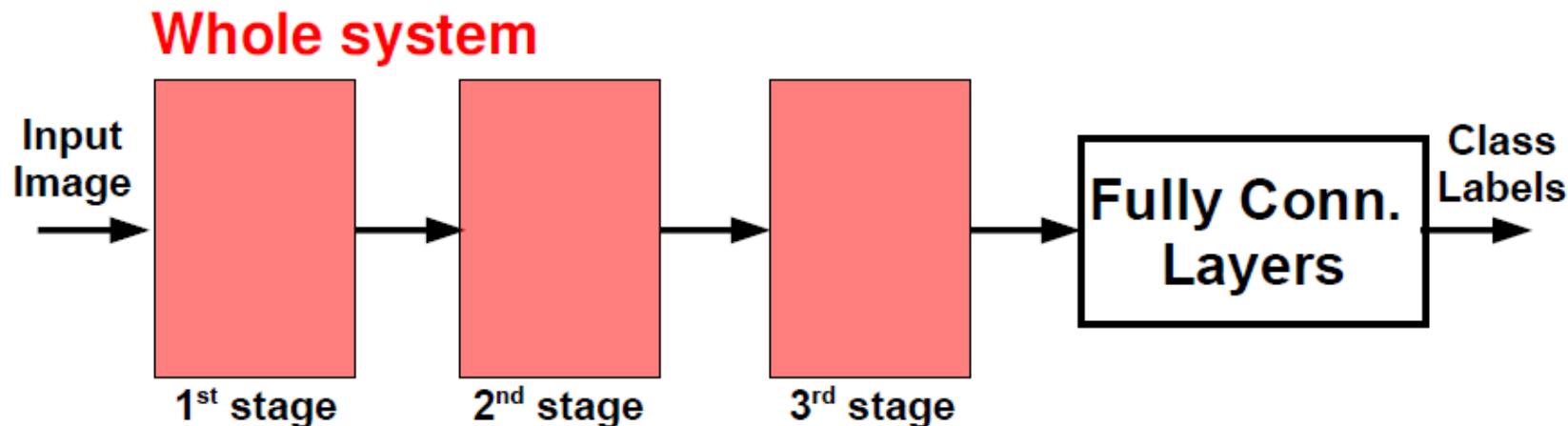
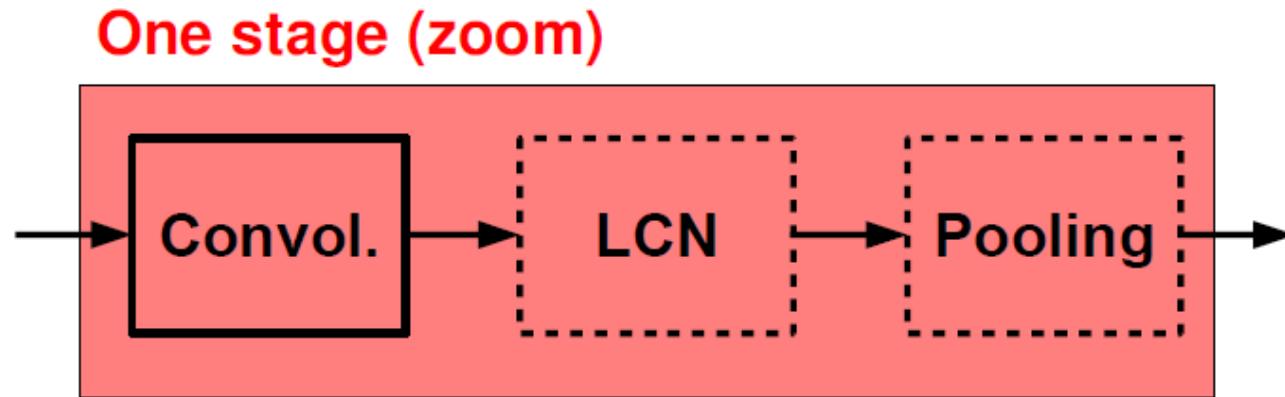
Effects:

- improves invariance
- improves optimization
- increases sparsity

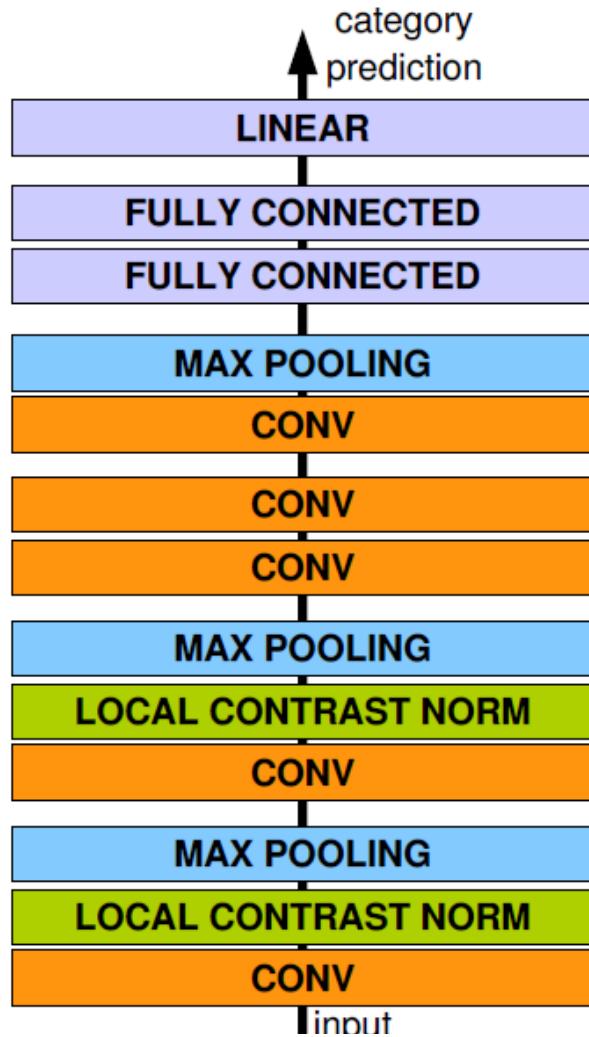
# Typical Architecture



# Typical Architecture (cont'd)



# Conv Architecture Example (AlexNet)



Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

# Convolutional Nets: Training

- All layers are differentiable (a.e.). We can use standard back-propagation.
- **Algorithm:**

Given a small mini-batch

1. F-PROP
2. B-PROP
3. PARAMETER UPDATE

# Summary: Key Ideas of Conv Nets

- Conv. Nets have special layers like:
  - – pooling, and
  - – local contrast normalization
  - Back-propagation can still be applied.
- These layers are useful to:
  - – reduce computational burden
  - – increase invariance
  - – ease the optimization

# PART 2: Architecture Design

- Overview
- Structure design
- Layer design
- Architecture for special tasks

# PART 2: Architecture Design

- Overview
- Structure design
- Layer design
- Architecture for special tasks

# Architecture Design

- What?
  - Network topology
  - Layer functions
  - Hyper-parameters
  - Optimization algorithms
  - ...
- Why?
  - Difficult to determine the optimal structures
  - Requirements of different applications, datasets or limitations

# Architecture Design (cont'd)

- How?
  - Manually
  - Automatically
- Objective
  - Representation capability
  - Robustness, anti-overfitting
  - Computation or parameter efficiency
  - Ease of optimization
  - ...



# PART 2: Architecture Design

- Overview
- **Structure design**
- Layer design
- Architecture for special tasks

# Benchmark: ImageNet Dataset

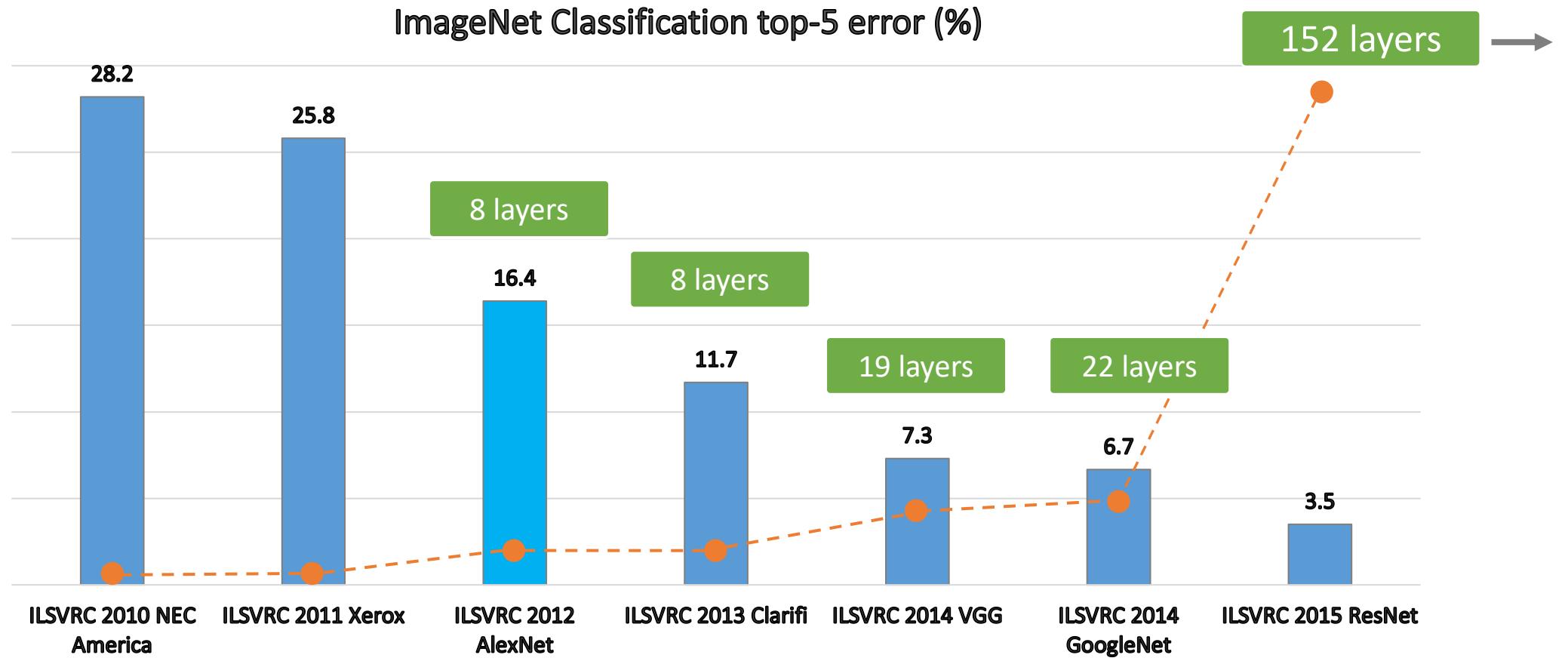
- 1K classes (for ILSVRC competition)
- 1.2M+ training images, 50K validation images, 100K test images
- ILSVRC competition

Difficulty

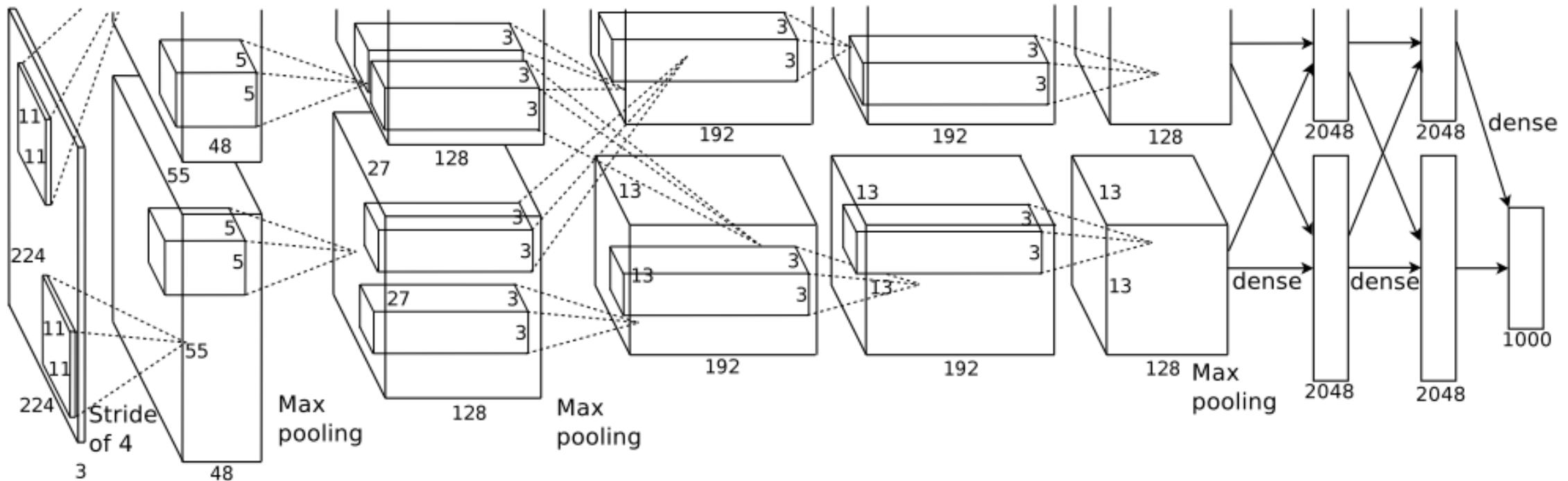
- Fine-grained classes
- Large variation
- Costly training



# Recent Nets – ImageNet Classification Scores



# AlexNet

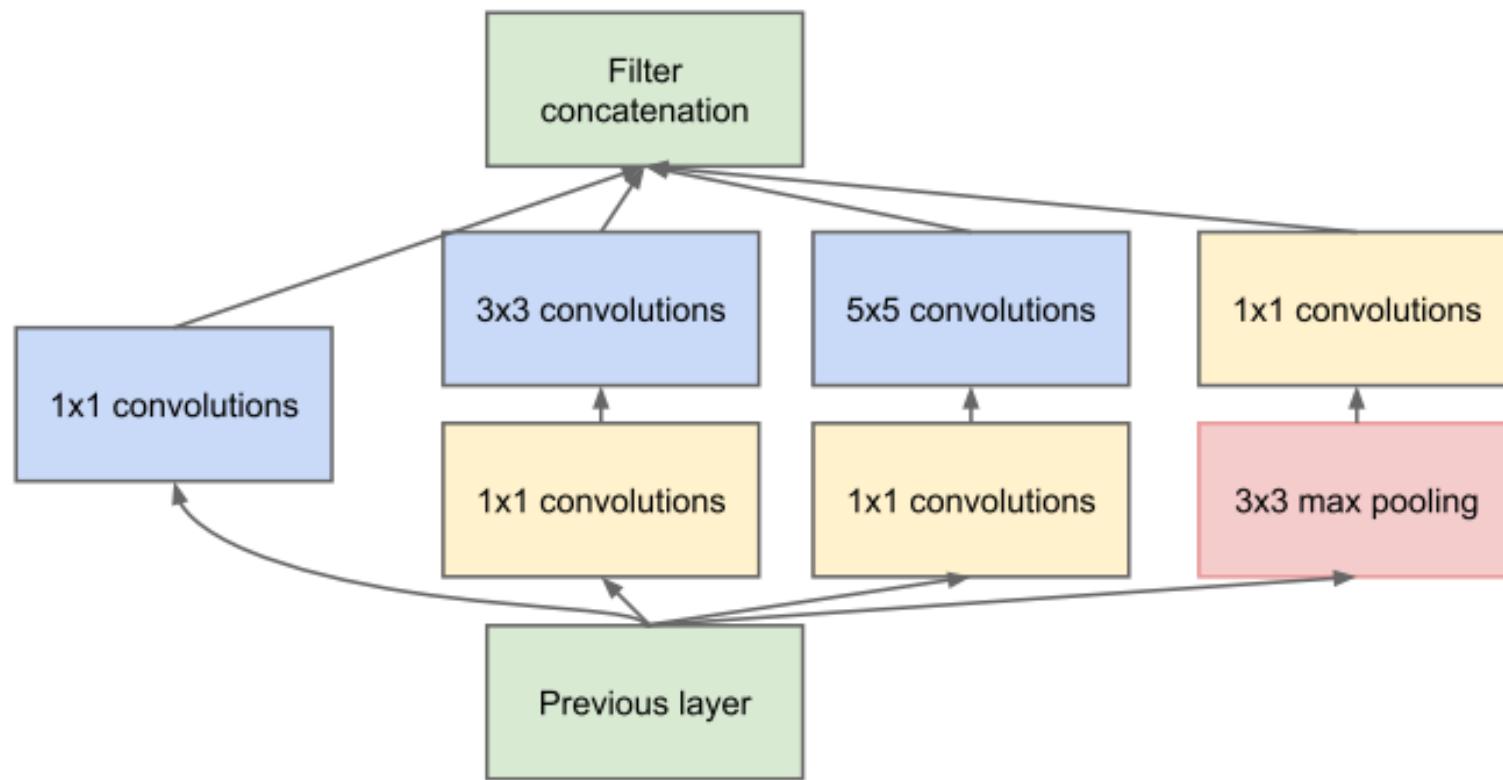


# VGGNet

Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition

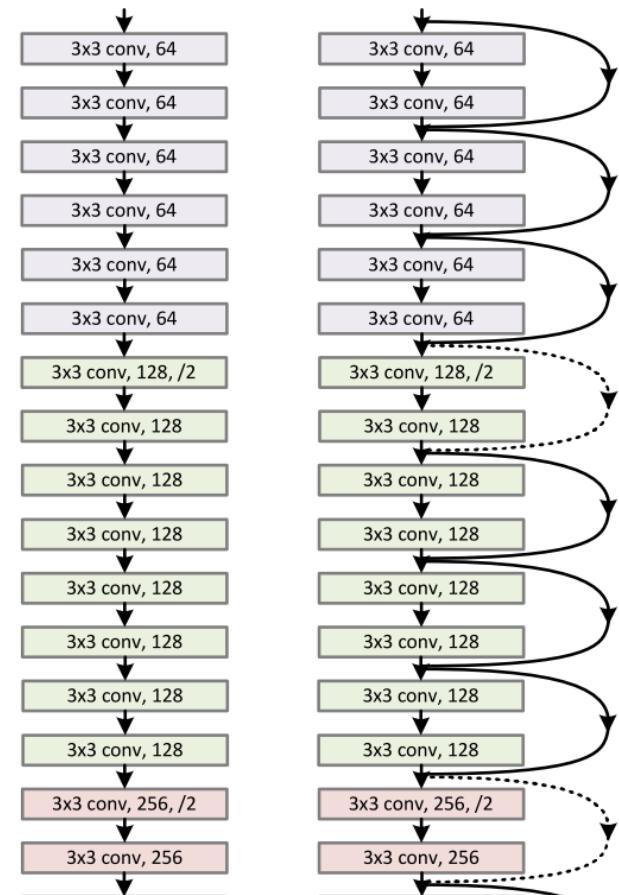
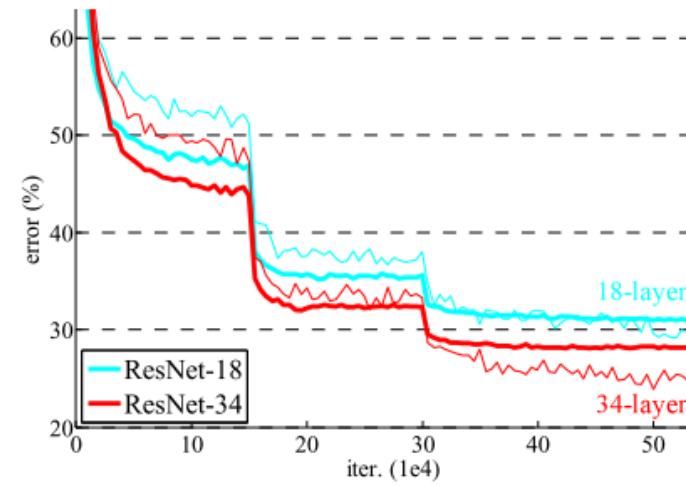
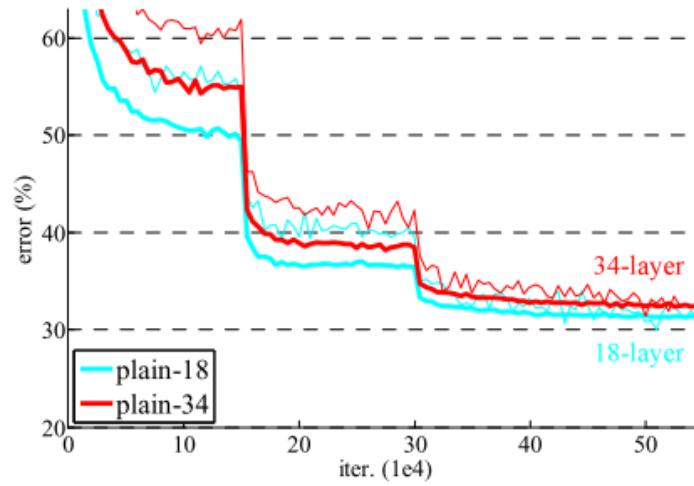
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool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

# GoogleNet



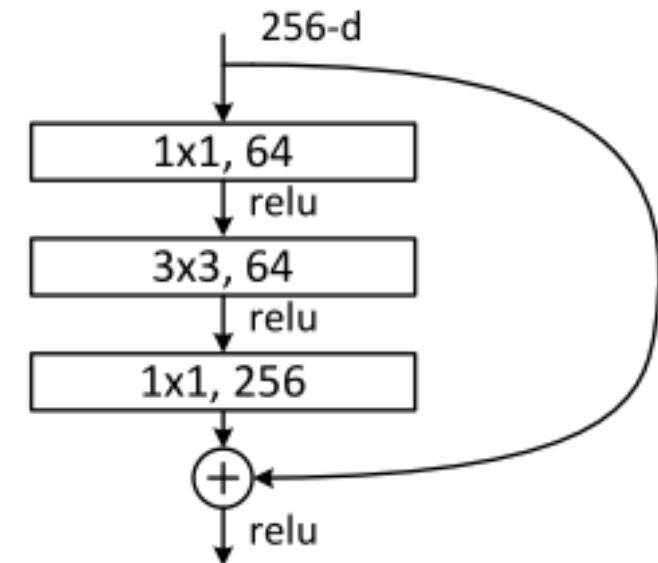
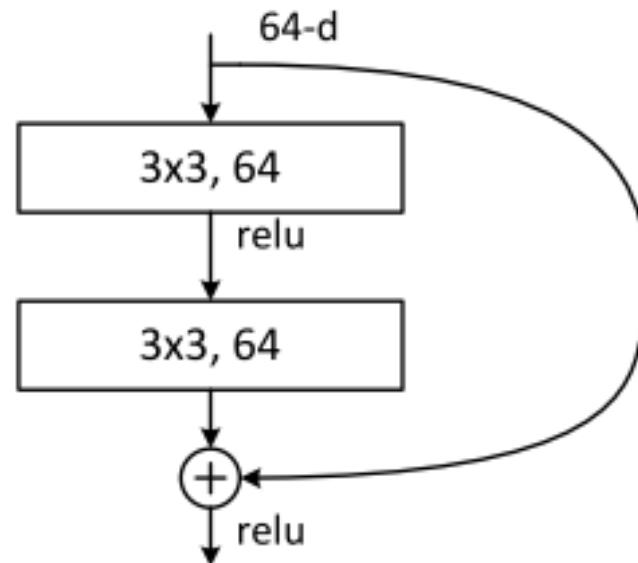
# Deep Residual Network

- Easy to optimize
- Enable very deep structures
  - Over 100 layers for ImageNet model

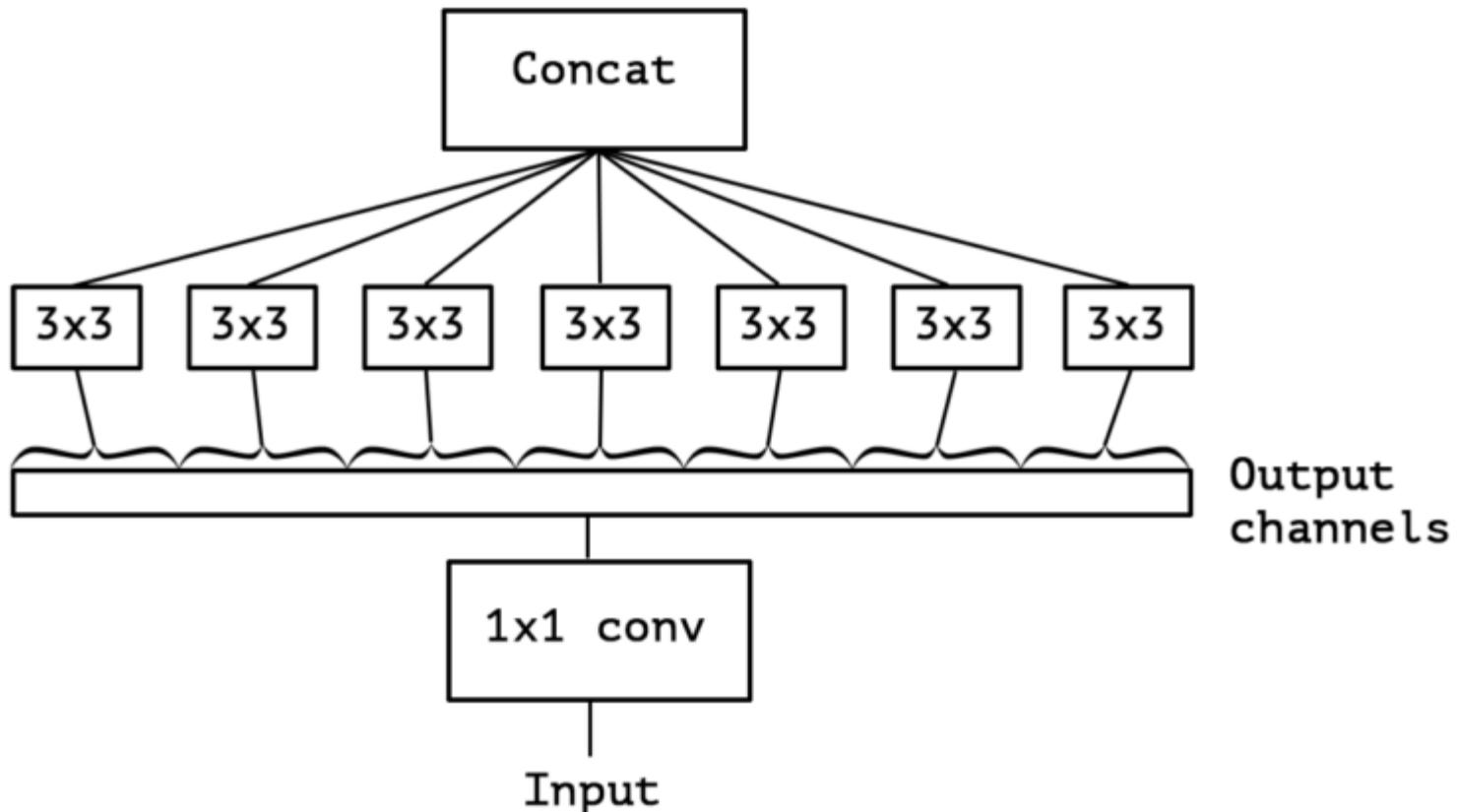


# Deep Residual Network (cont'd)

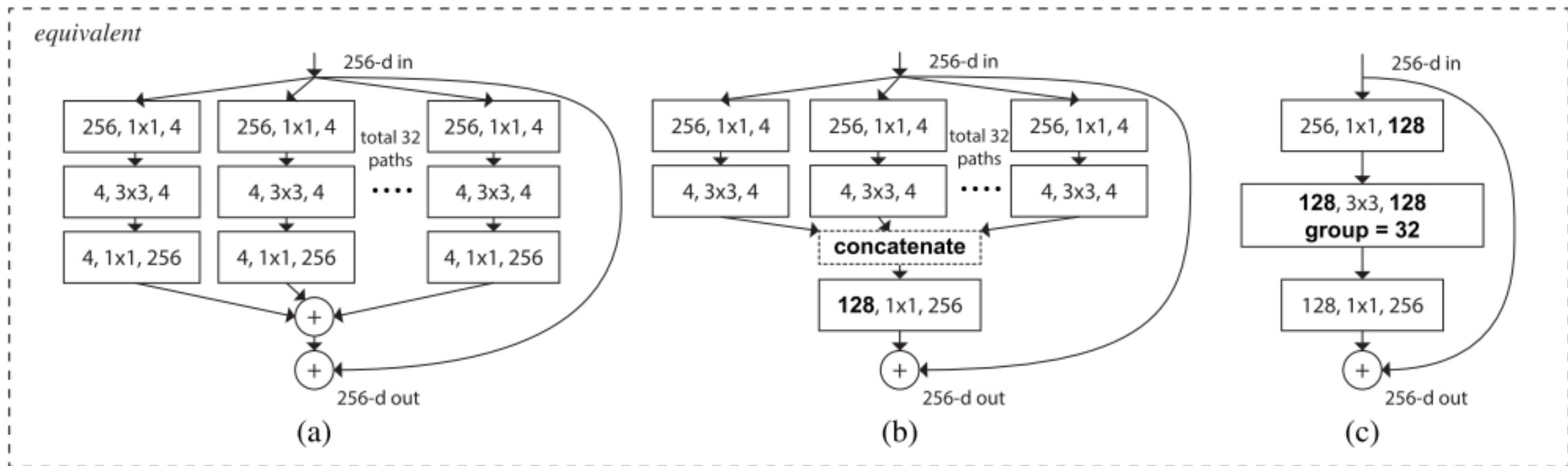
- “Bottleneck” design
- Increasing depth, less complexity



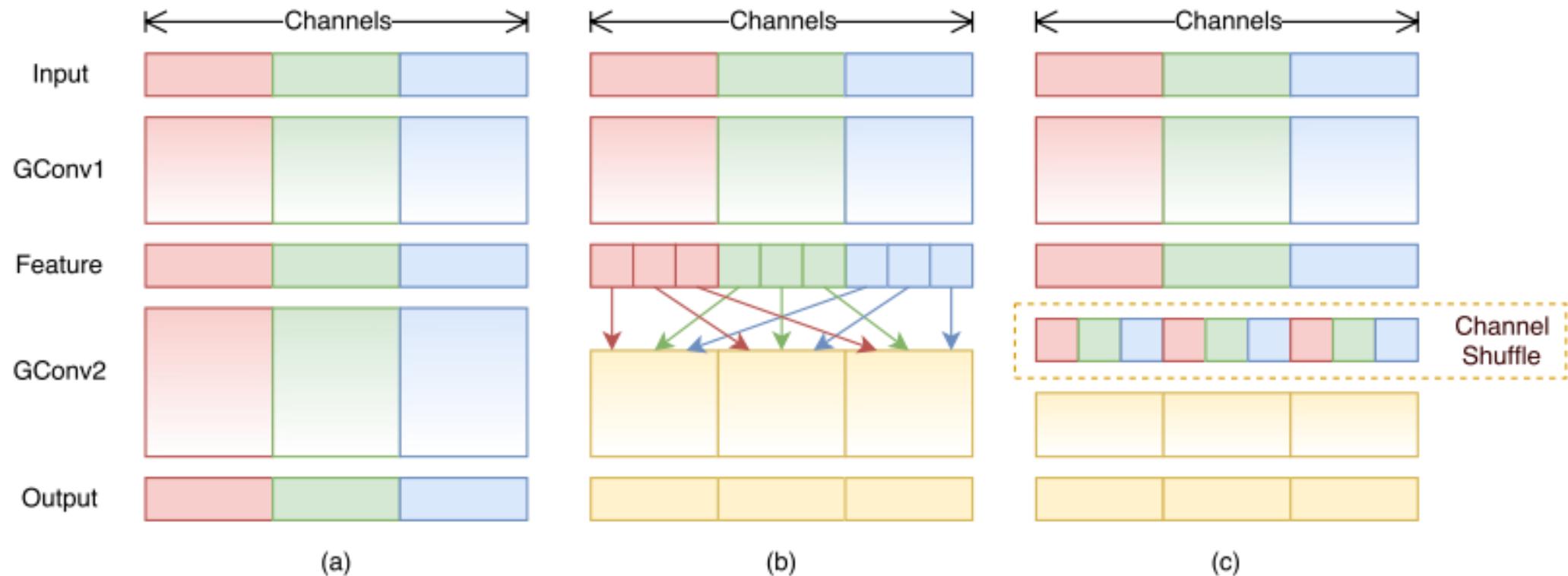
# Xception



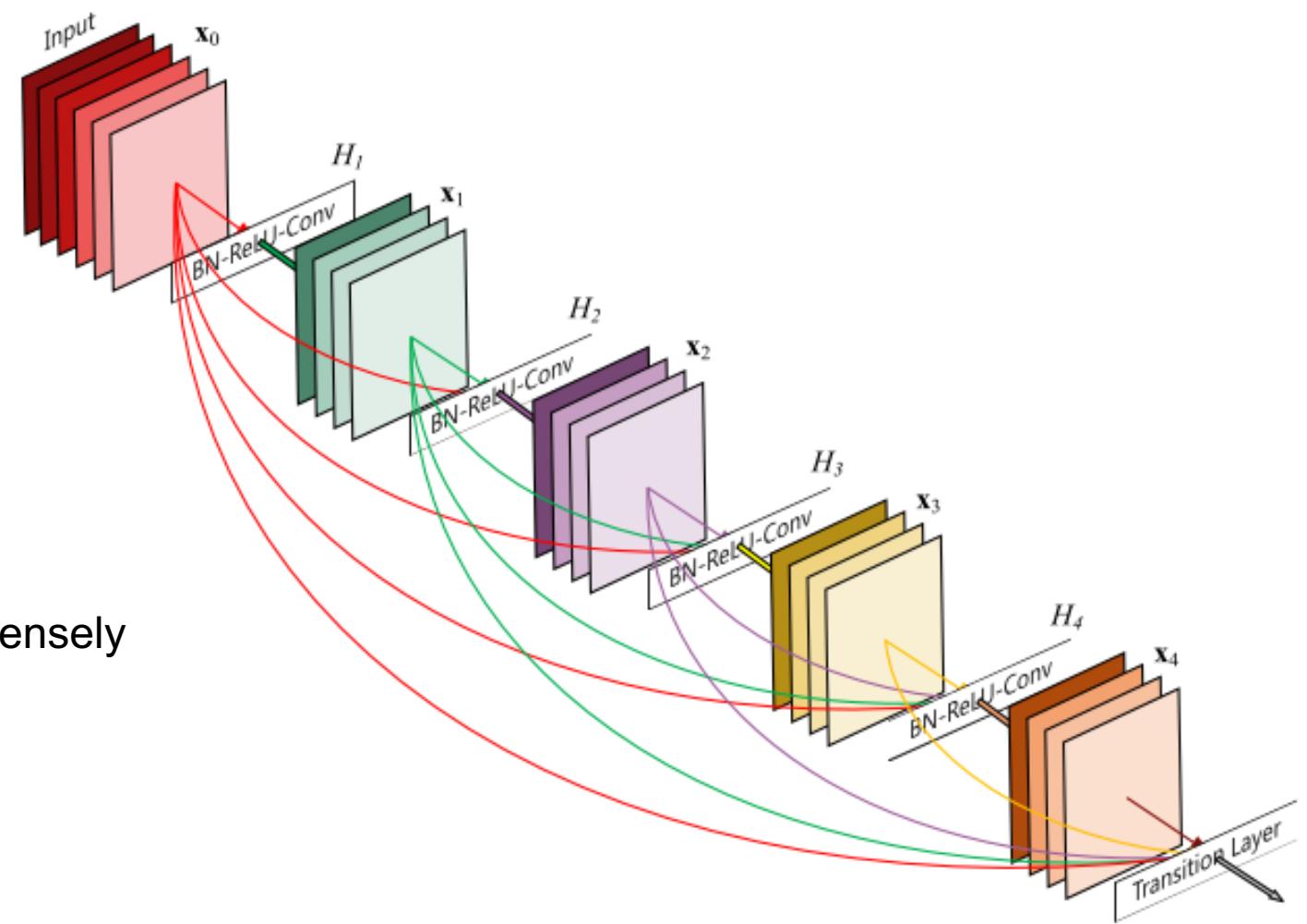
# ResNeXt



# ShuffleNet

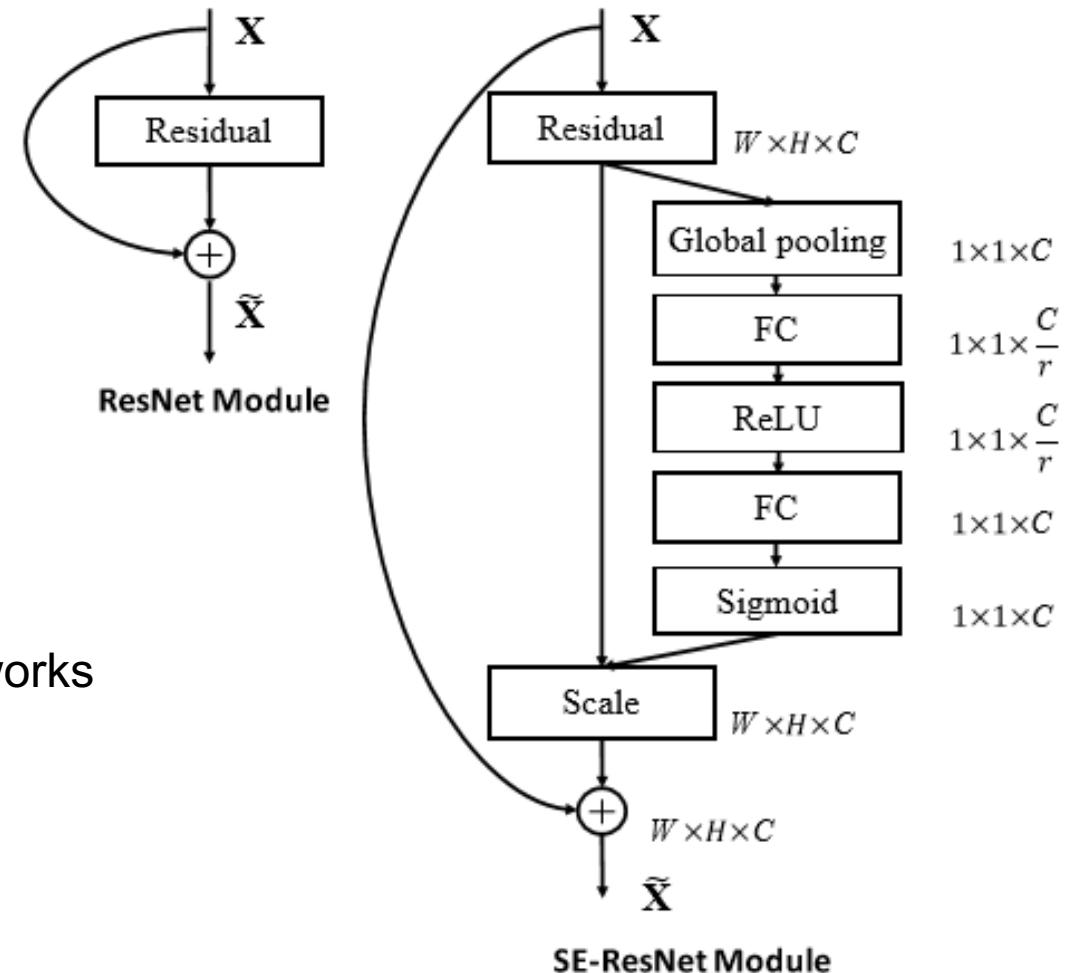


# Densely Connected Convolutional Networks



Huang G, Liu Z, Weinberger K Q, et al. Densely connected convolutional networks

# Squeeze-and-Excitation Networks



Hu J, Shen L, Sun G. Squeeze-and-Excitation Networks

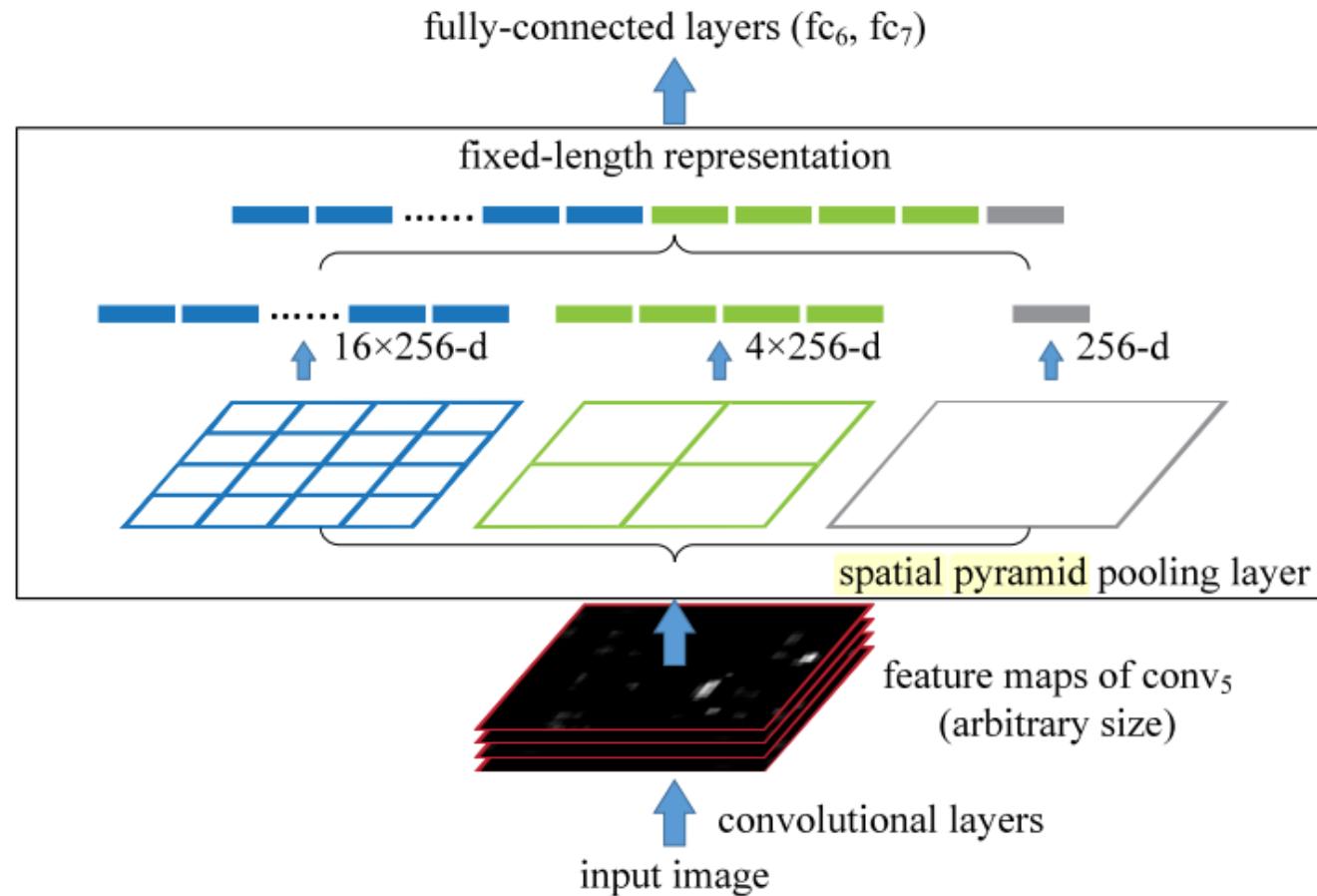
# Summary: Ideas of Structure Design

- Deeper and wider
- Ease of optimization
- Multi-path design
- Residual path
- Sparse connection

# PART 2: Architecture Design

- Overview
- Structure design
- Layer design
- Architecture for special tasks

# Spatial Pyramid Pooling

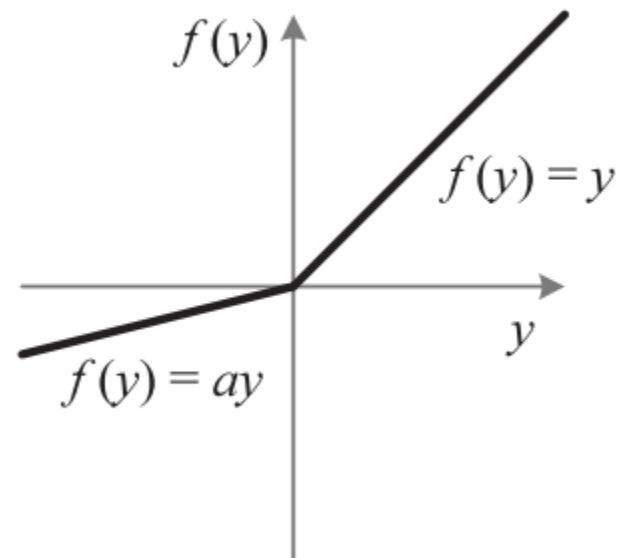
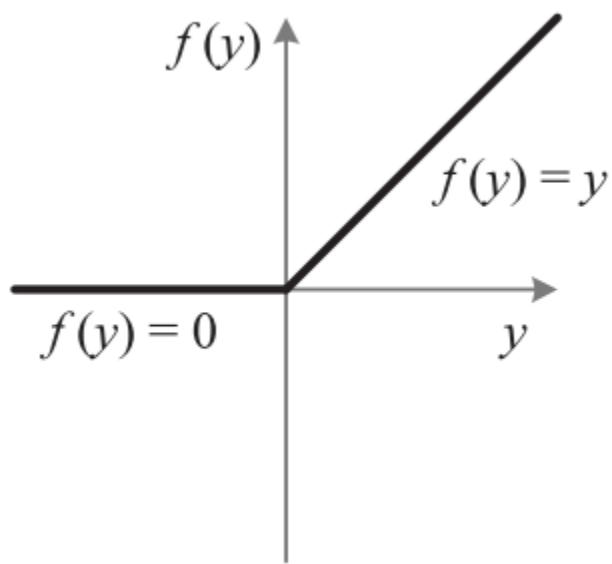


# Batch Normalization

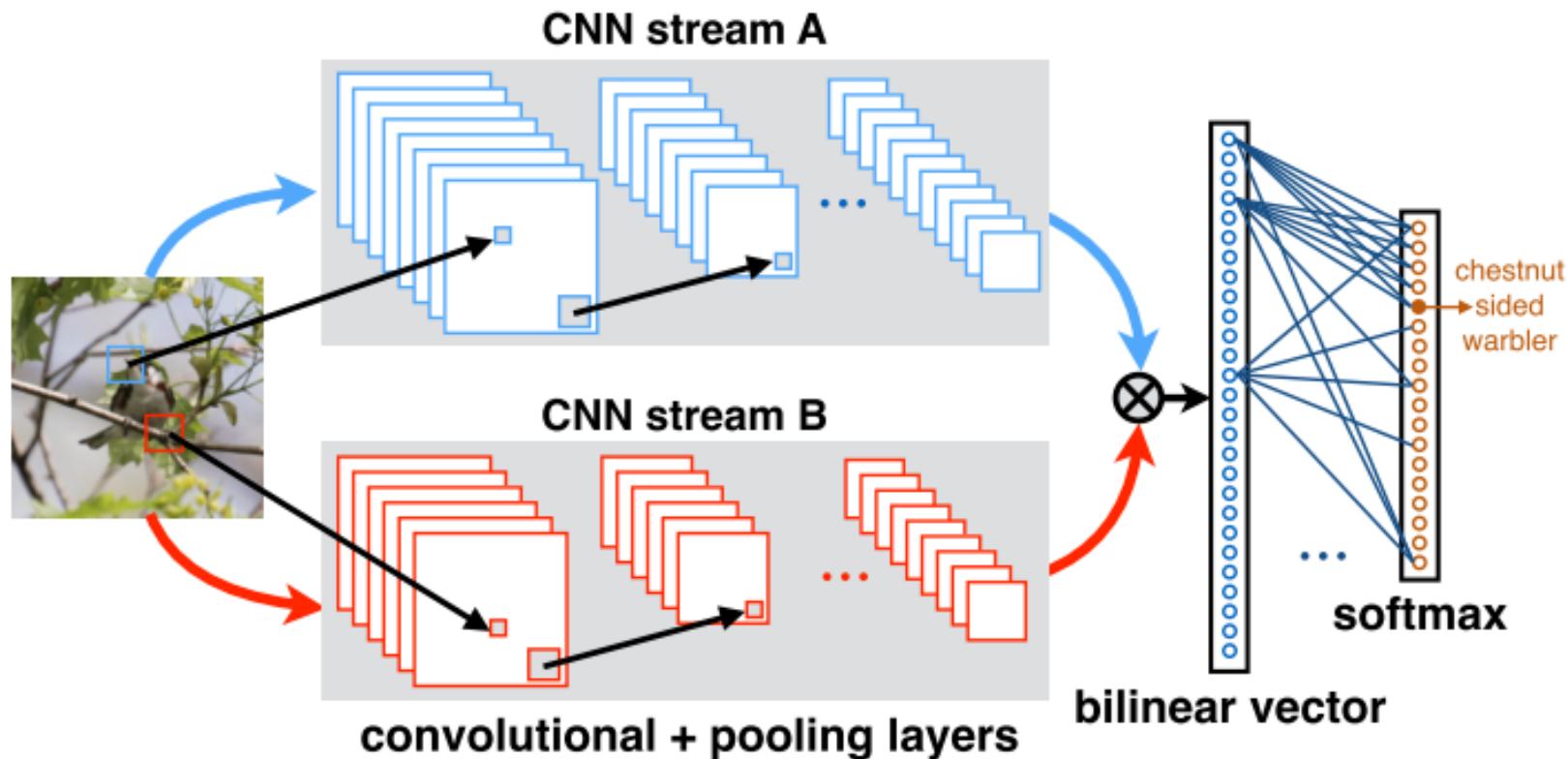
**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;  
Parameters to be learned:  $\gamma, \beta$   
**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift}\end{aligned}$$

# Parametric Rectifiers



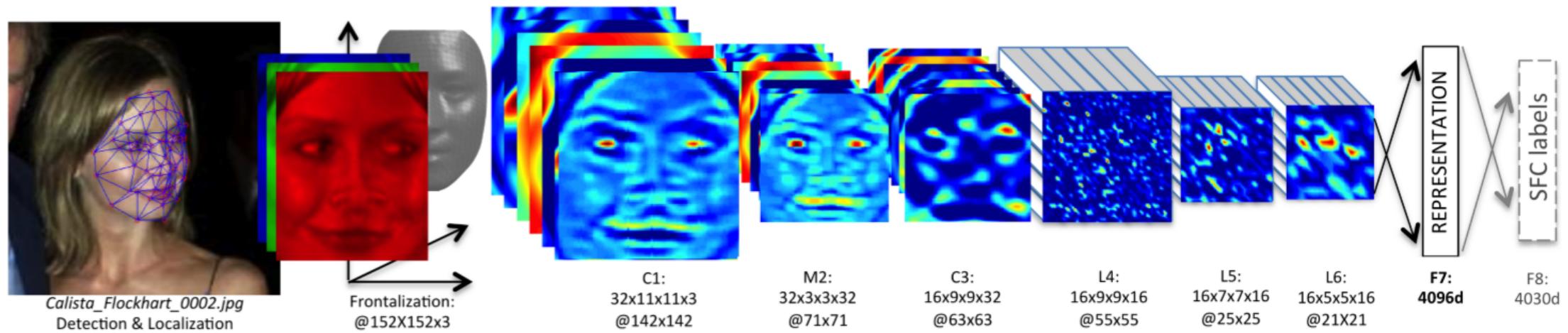
# Bilinear CNNs



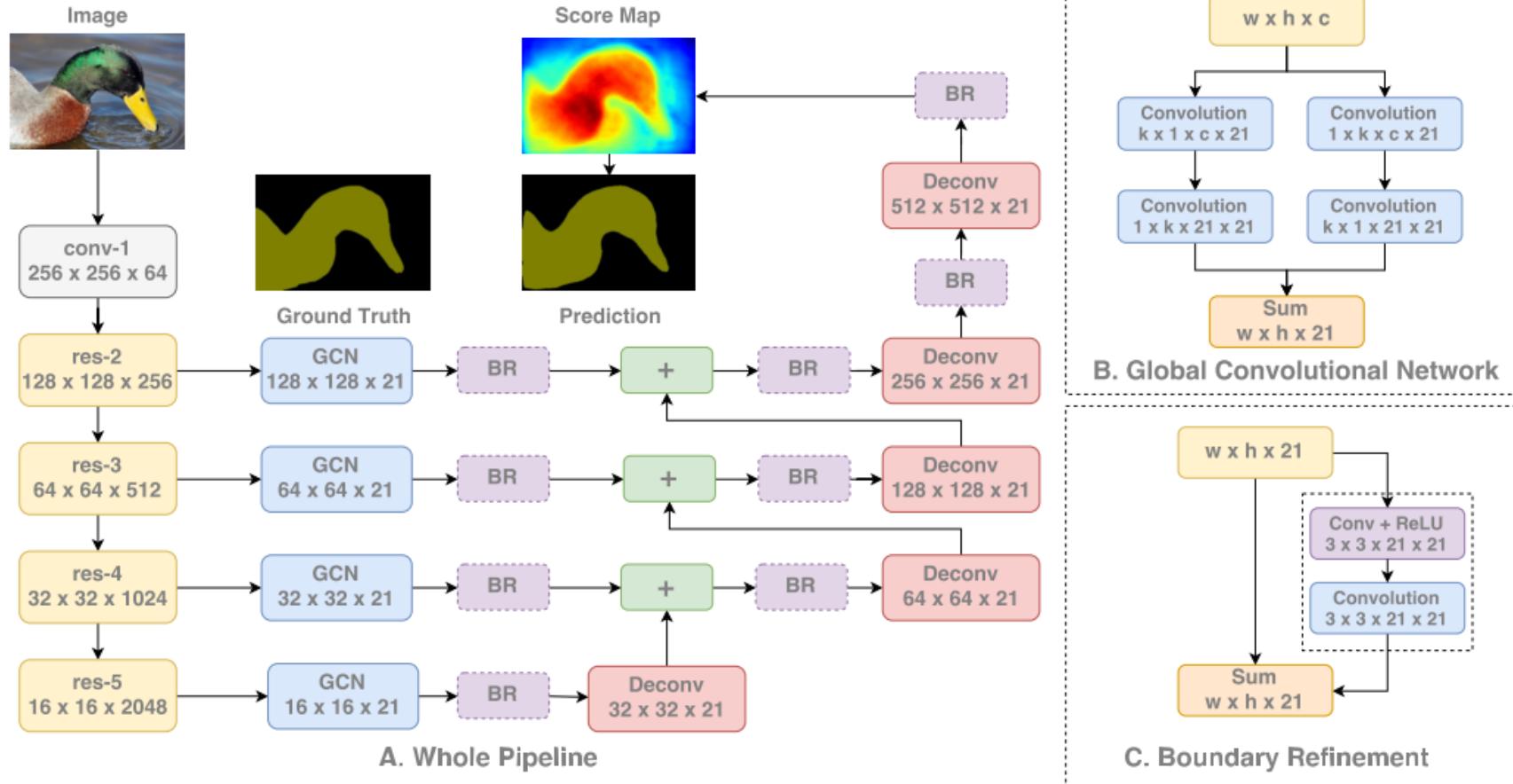
# PART 2: Architecture Design

- Overview
- Structure design
- Layer design
- **Architecture for special tasks**

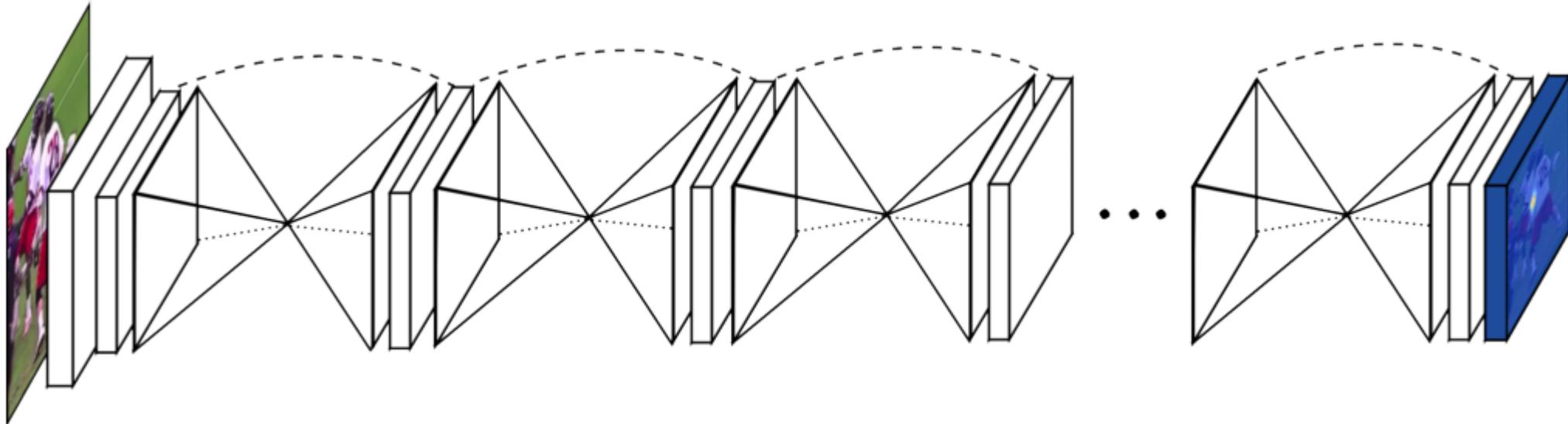
# Deepface



# Global Convolutional Networks



# Hourglass Networks



# Summary: Trends on Architecture Design

- Effectiveness and efficiency
- Task & data specific
- ML & optimization perspective
- Insight & motivation driven

Thanks