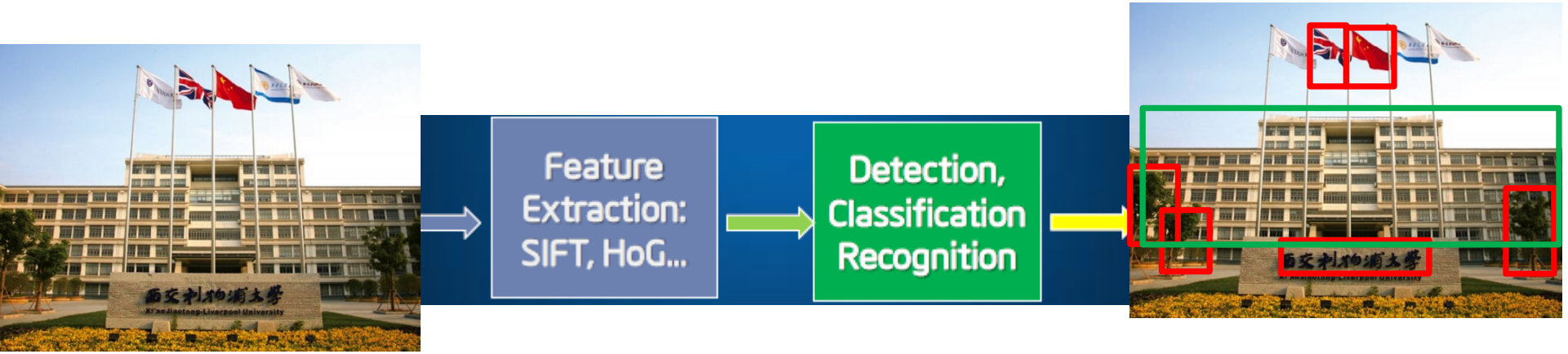
# INTRODUCTION TO CONVOLUTIONAL NEURAL NETWORK

**INT301 Bio-computation, Week 5, 2025**

# Classical Computer Vision Pipeline

1.Select / develop features ( e.g., HoG, SIFT, …)
2.Add machine learning on top of this for recognition and train classifier

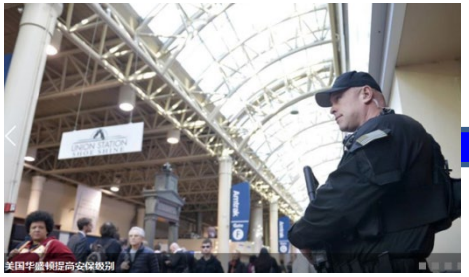



传统CV：手工设计，耗时费力

**Classical CV feature definition is domain-specific, hand engineered, and time-consuming**

# Features for machine learning

image



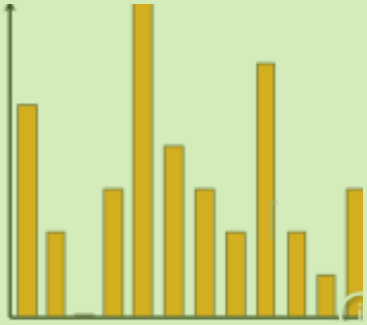
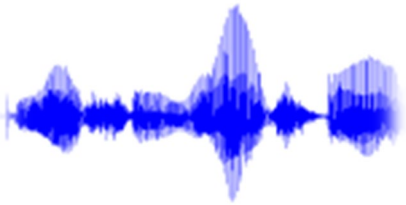
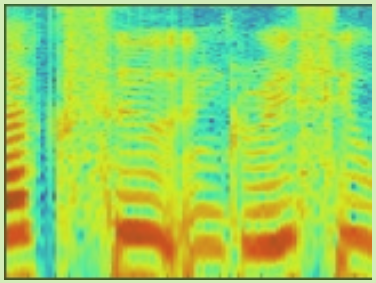


Image feature

Object detection /classification

audio

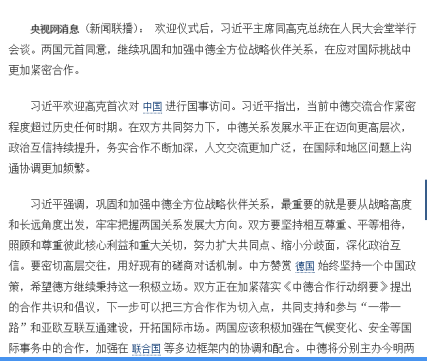


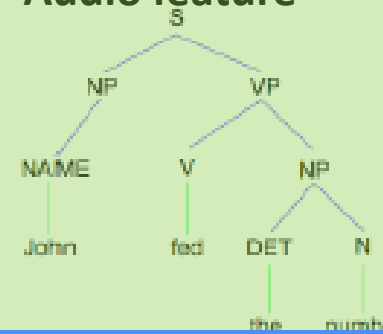


Audio feature

Speaker ID

text




Translation/ /categorization /Web search

# Key Ideas of Deep Learning

- Deal with non-linear system
- Learn feature from data (or big data)
- Build feature hierarchies (function composition)
- End-to-end learning

# Learning Feature Representations

无论图像，文本，音频，可通过统一神经网络实现，无须设计去确特征提取方法

## The idea:

- Most perception (input processing) in the brain may be due to **one learning algorithm**.
- Build learning algorithms that mimic the brain.
- Most of human intelligence may be due to **one learning algorithm**.

one learning algorithm ⟹ end-to-end system!

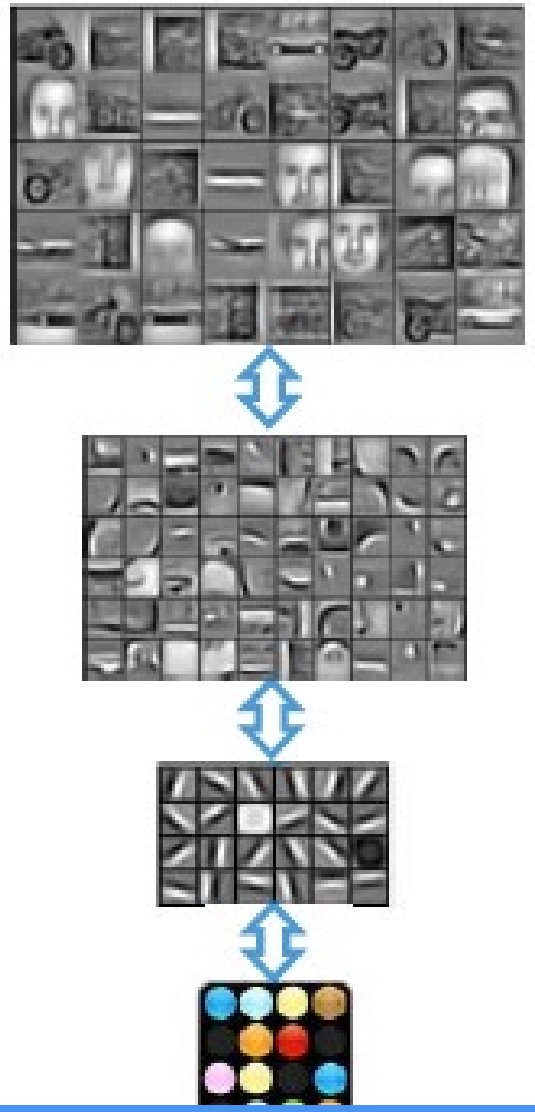
# Learning Feature Hierarchy

**Deep Learning**

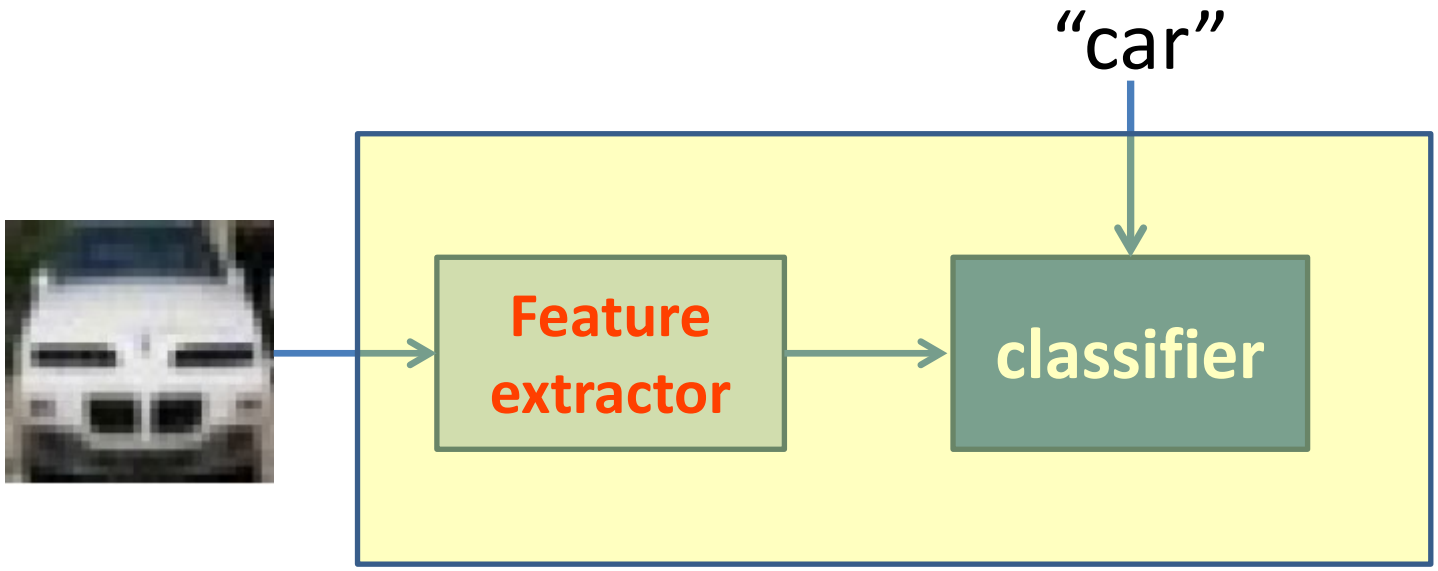
– Deep architectures can be representationally efficient.

– Natural progression from low level to high level structures.

– Can share the lower-level representations for multiple tasks.



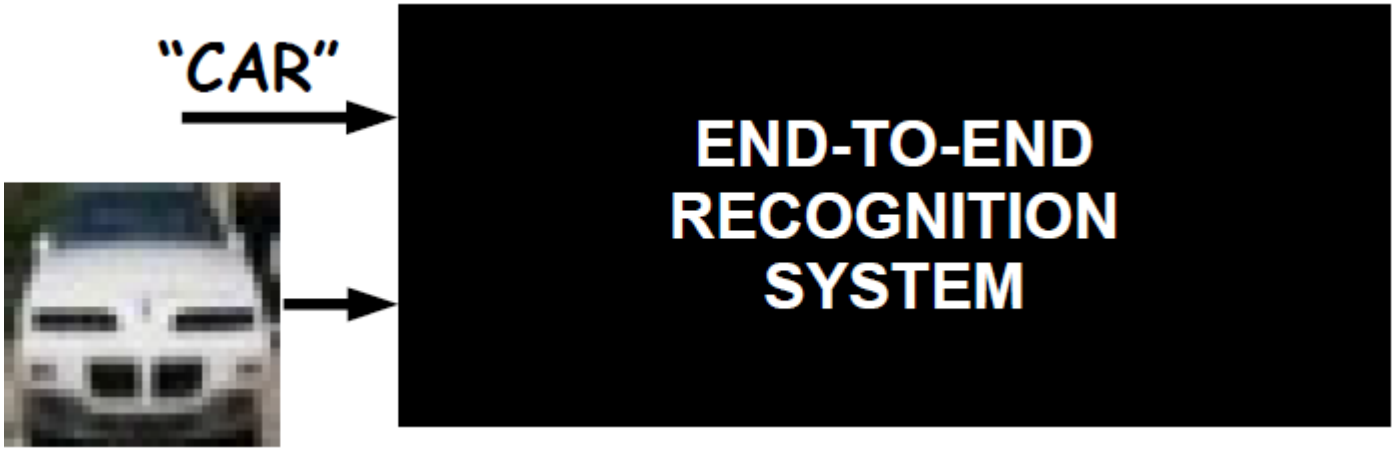
# End-to-end Object Recognition




**How to use data to optimize features for the given task?**

- Everything becomes adaptive.
- No distinction between feature extractor and classifier.
- Big non-linear system trained from raw pixels to labels

# End-to-end Object Recognition

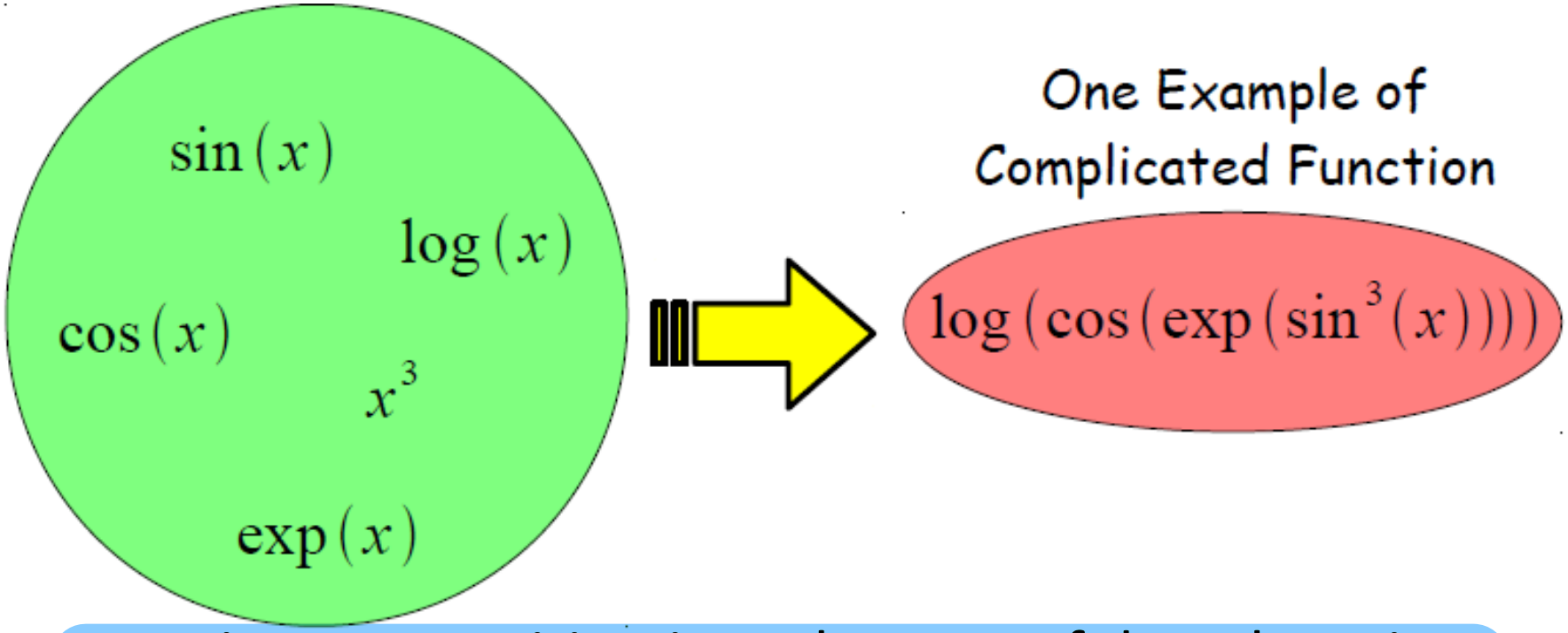



**Q:** How can we build such a highly non-linear system?

**A:** By combining simple building blocks, we can make more and more complex systems.

# Building A Complicated Function

Simple Functions

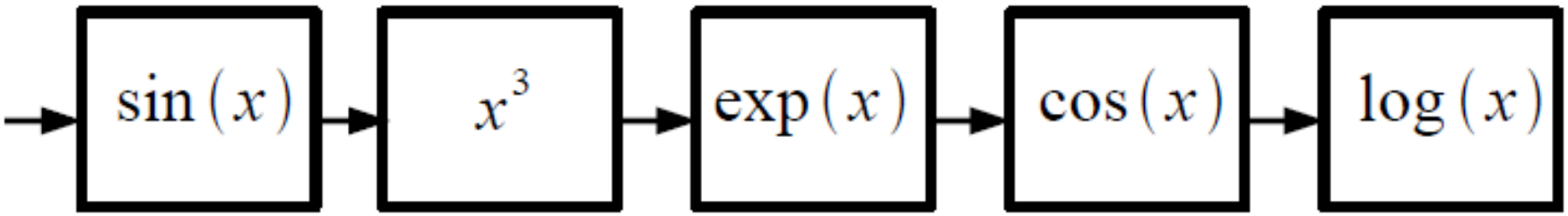



- Function composition is at the core of deep learning methods.
- Each “simple function” will have parameters subject to

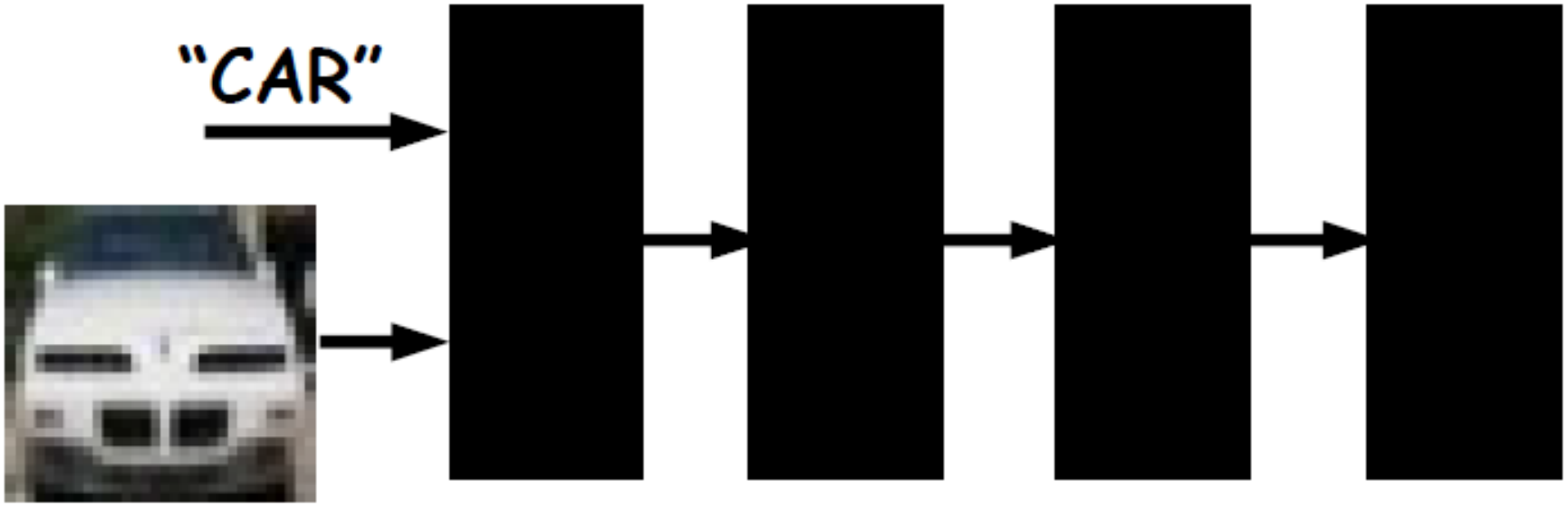
# Building A Complicated Function

Complicated Function

$$\log(\cos(\exp(\sin^3(x))))$$



# Intuition Behind Deep Neural Nets

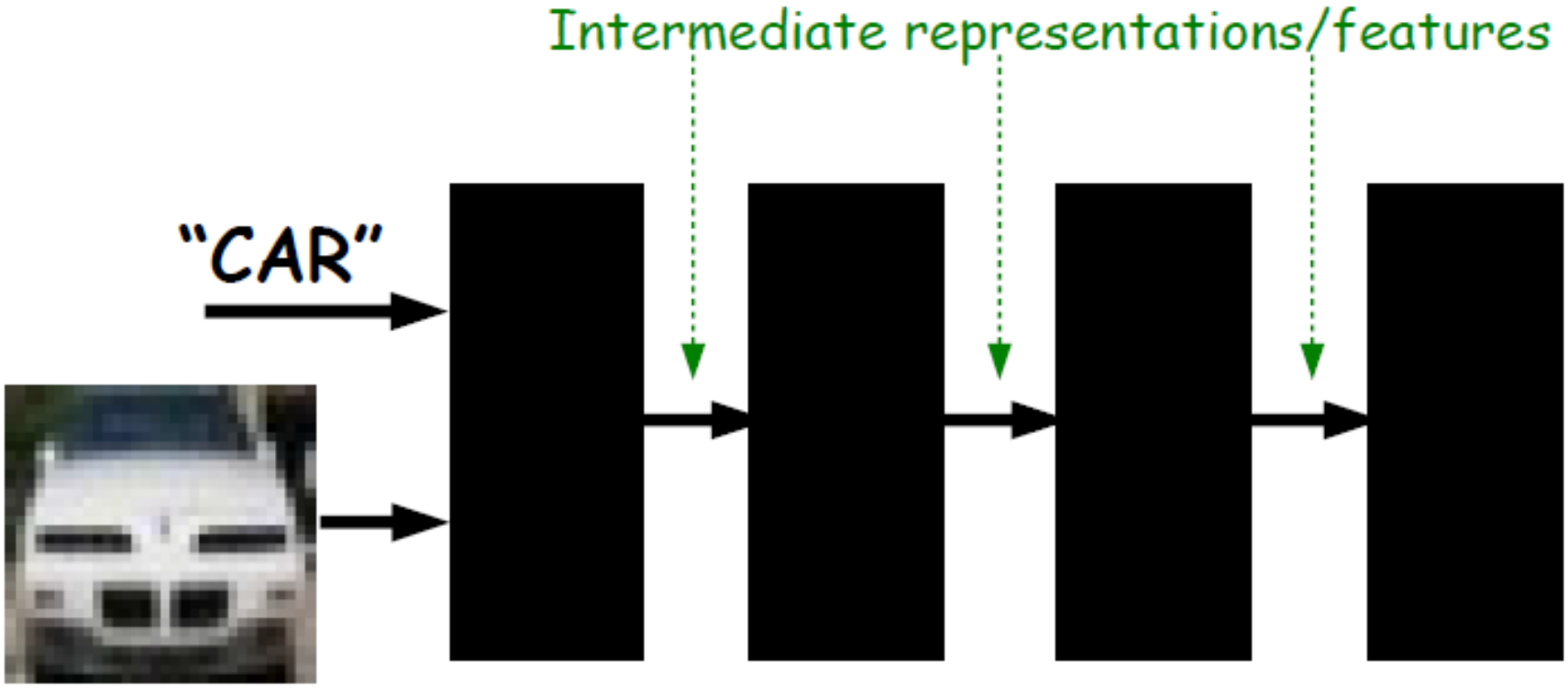



**NOTE:**

**Each black box can have trainable parameters.**

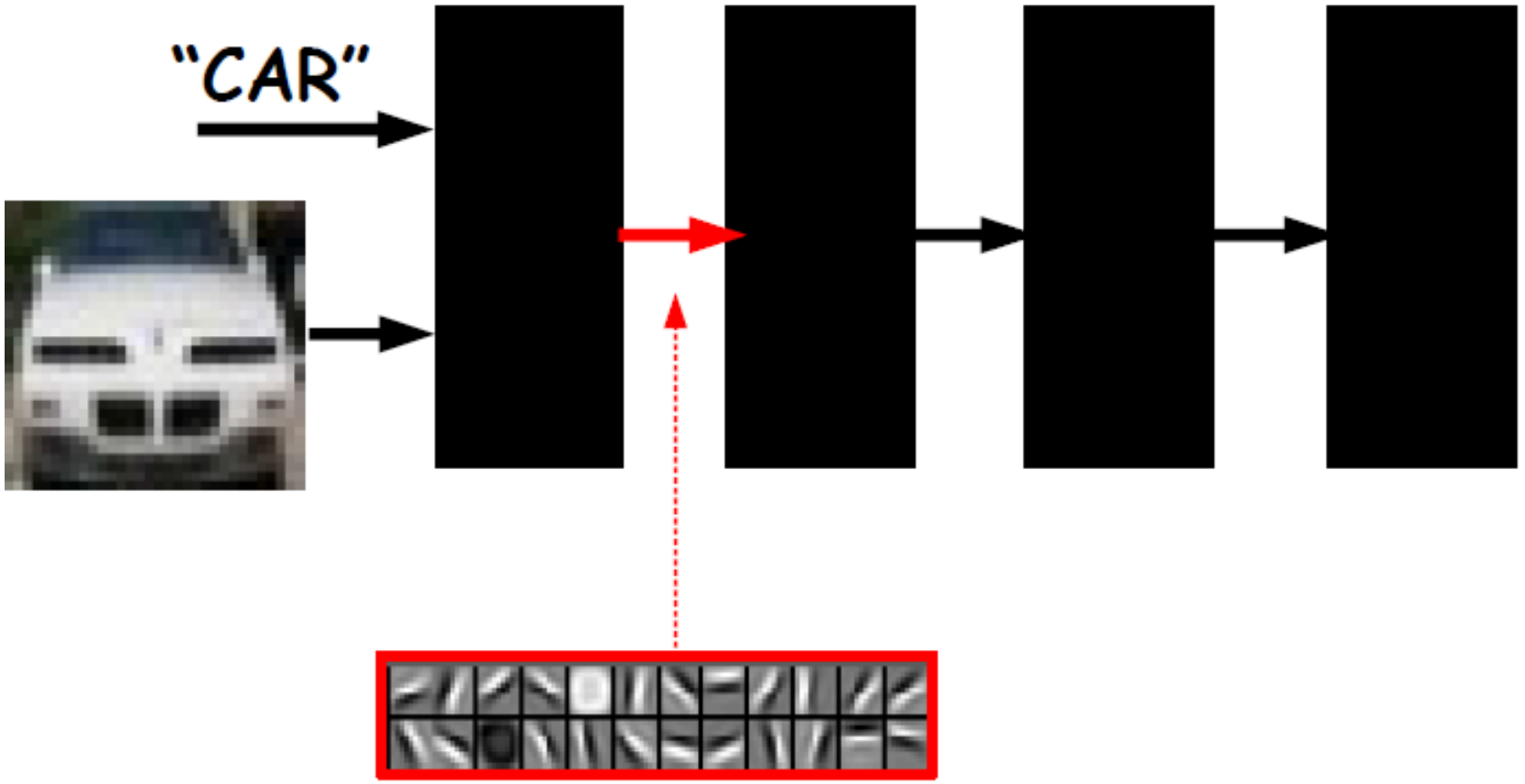
Their composition makes a highly non-linear system.

# Intuition Behind Deep Neural Nets

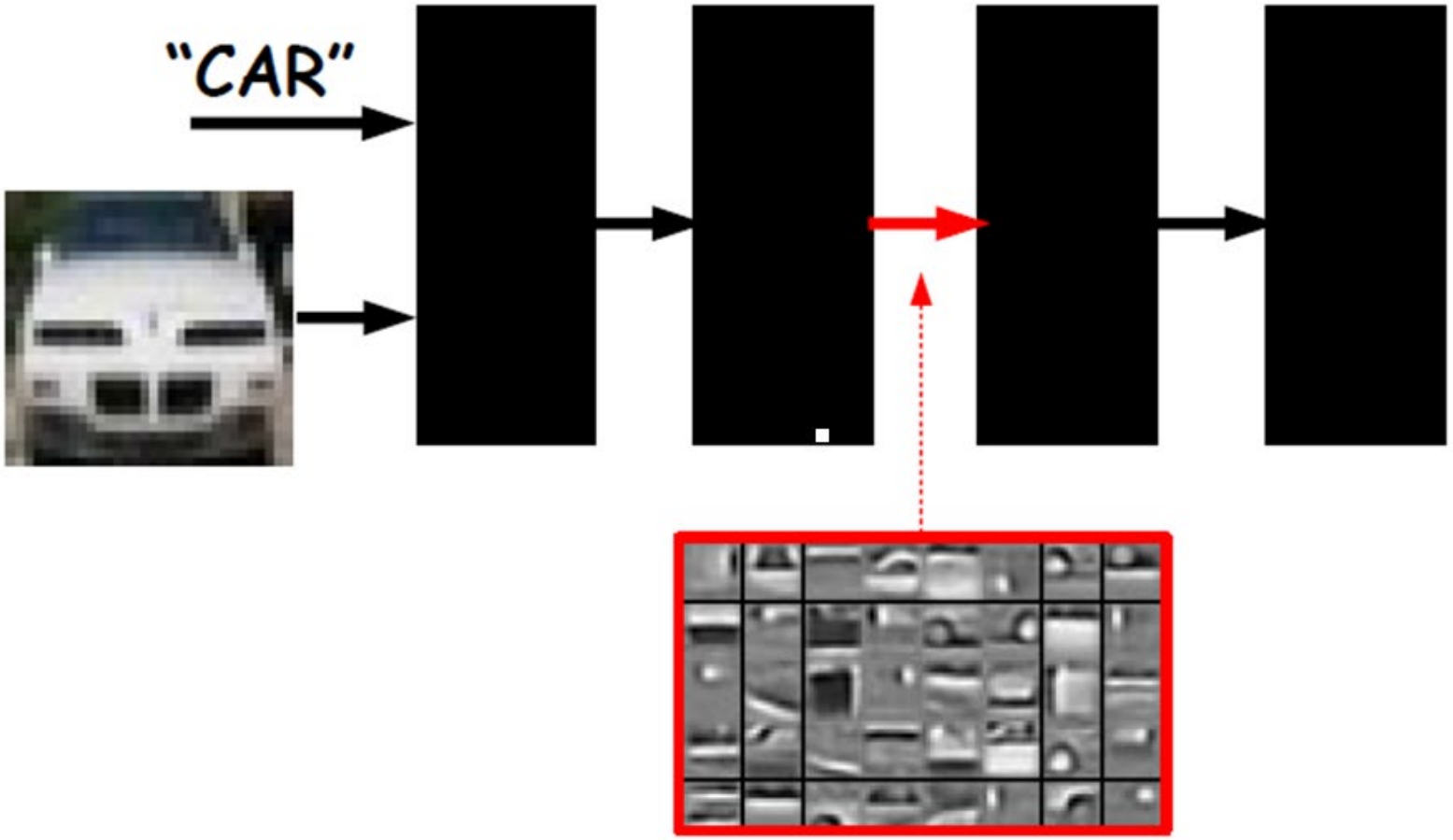



**NOTE: System produces a hierarchy of features**

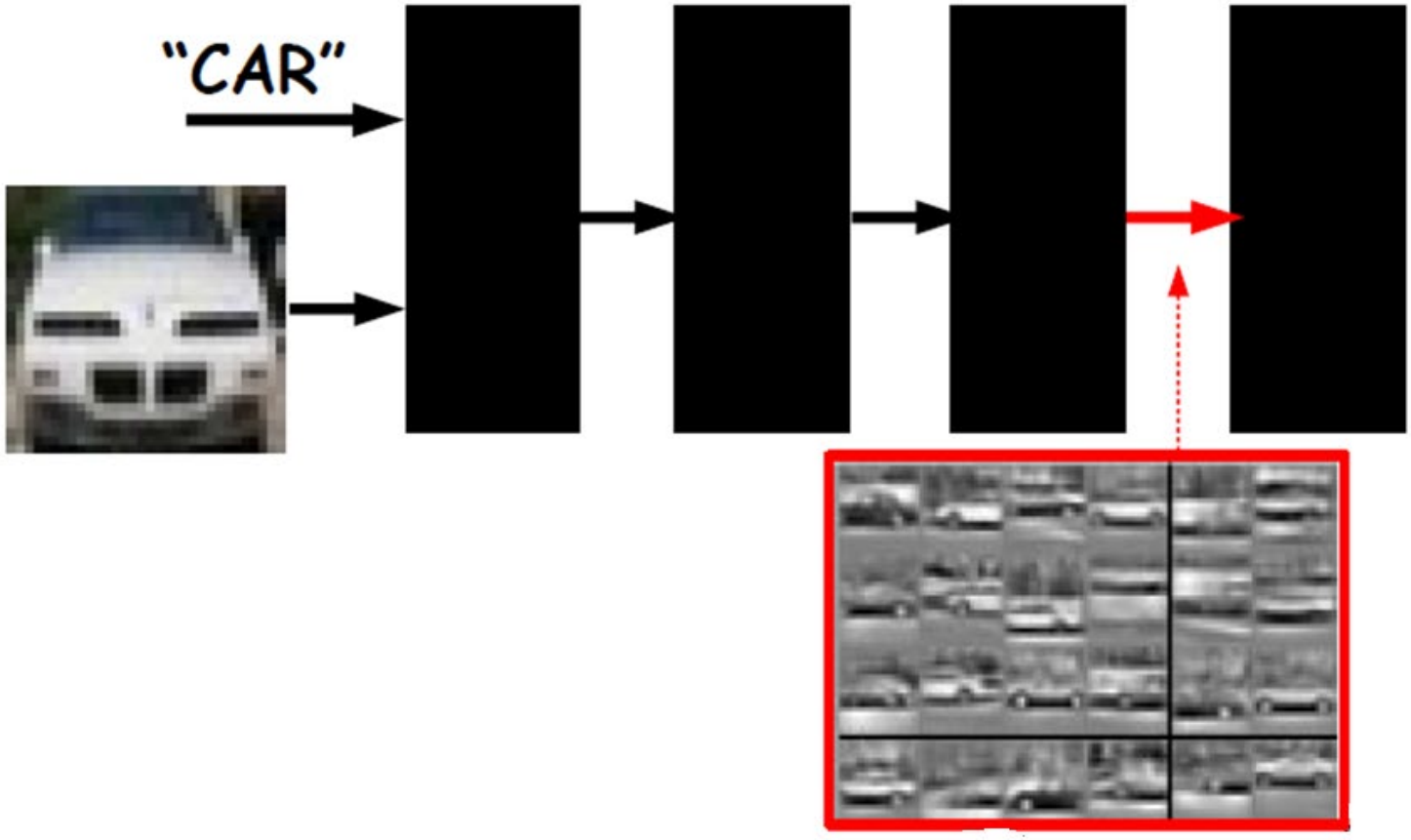
# Intuition Behind Deep Neural Nets



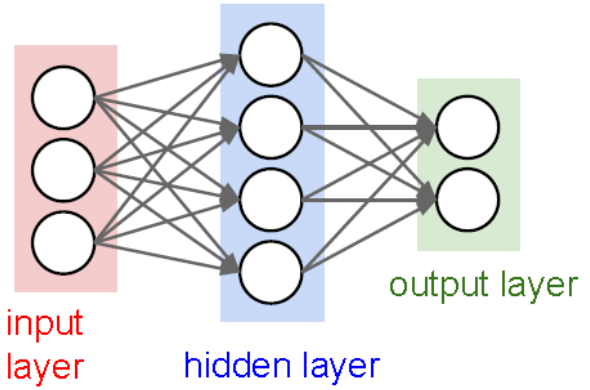
# Intuition Behind Deep Neural Nets

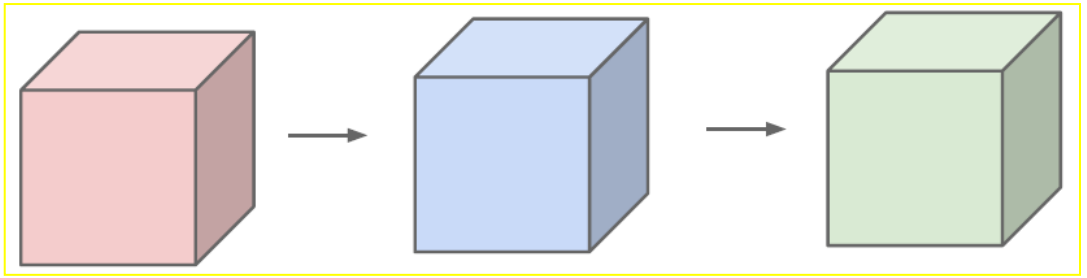


# Intuition Behind Deep Neural Nets



# Convolutional NN



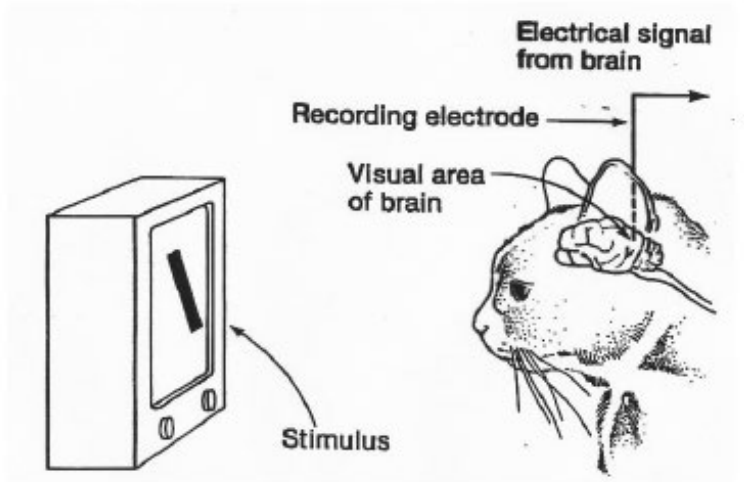
- Convolutional Neural Networks is extension of traditional Multi-layer Perceptron, based on 3 ideas:
  1. **Local receive fields** 局部感受野
  2. **Shared weights** 共享权重
  3. **Spatial / temporal sub-sampling** 空间/时间 子采样

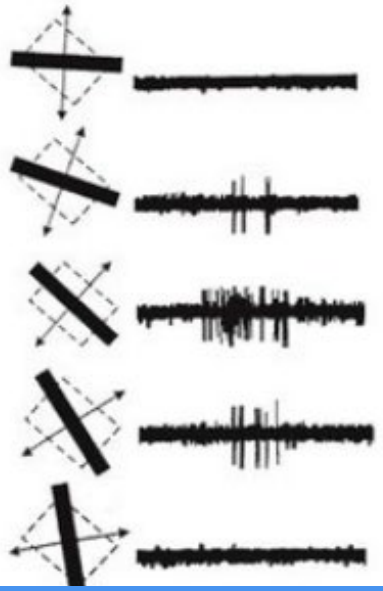
# A bit of history

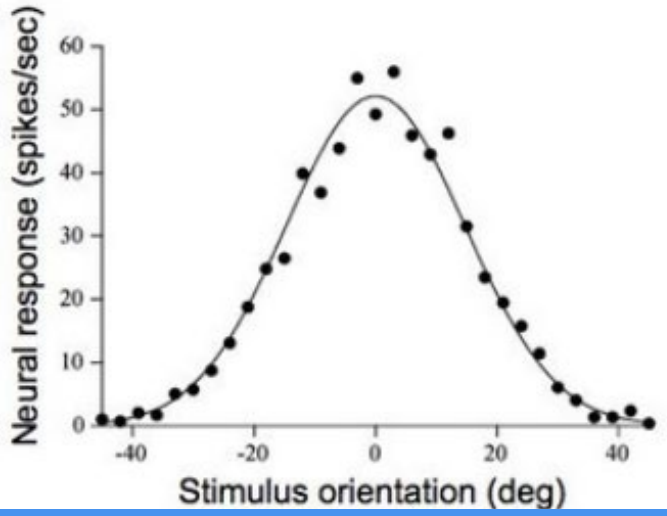
## Hubel & Wiesel

**1959 Paper**
Receptive fields of single neurones in the cat's striate cortex

**1962 Paper**
Receptive fields, binocular interaction and functional architecture in cat's visual cortex



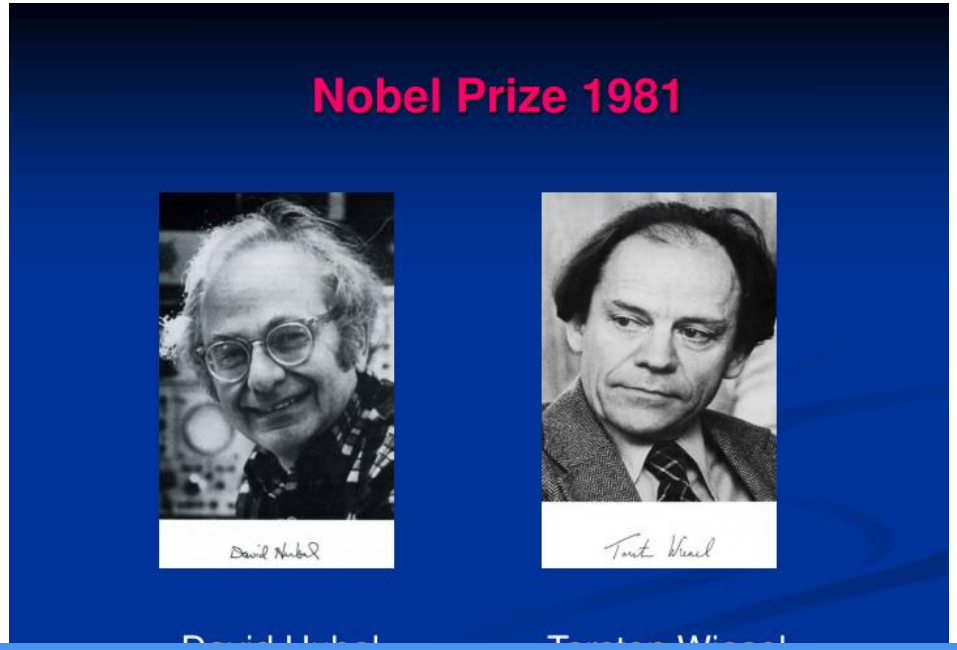



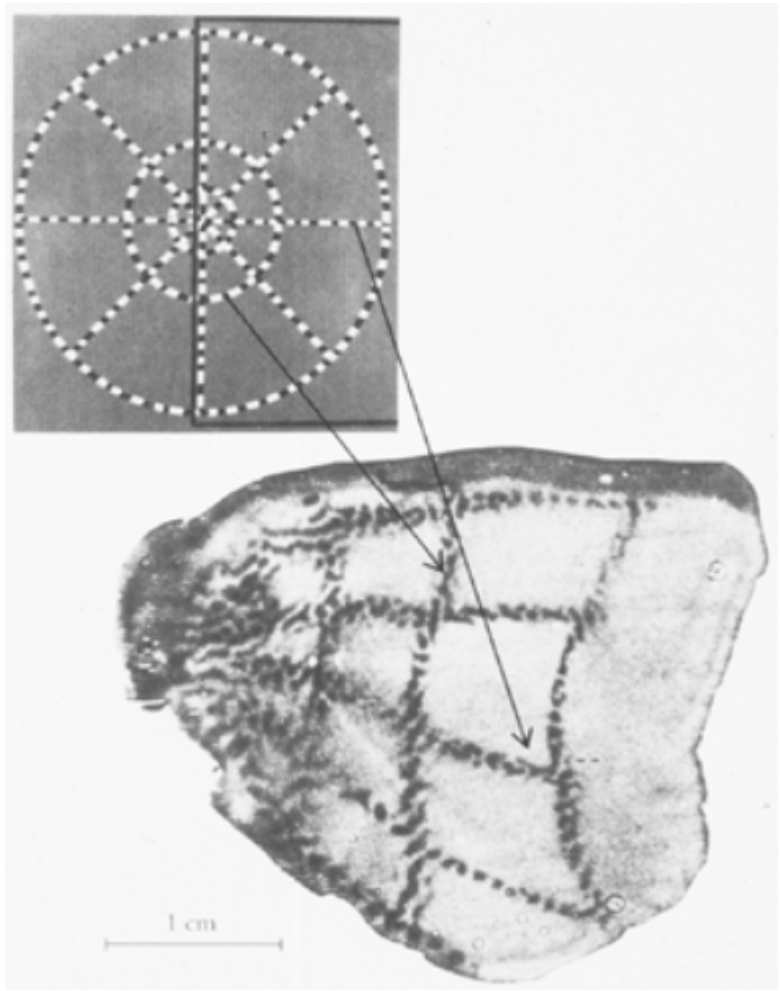


# A bit of history

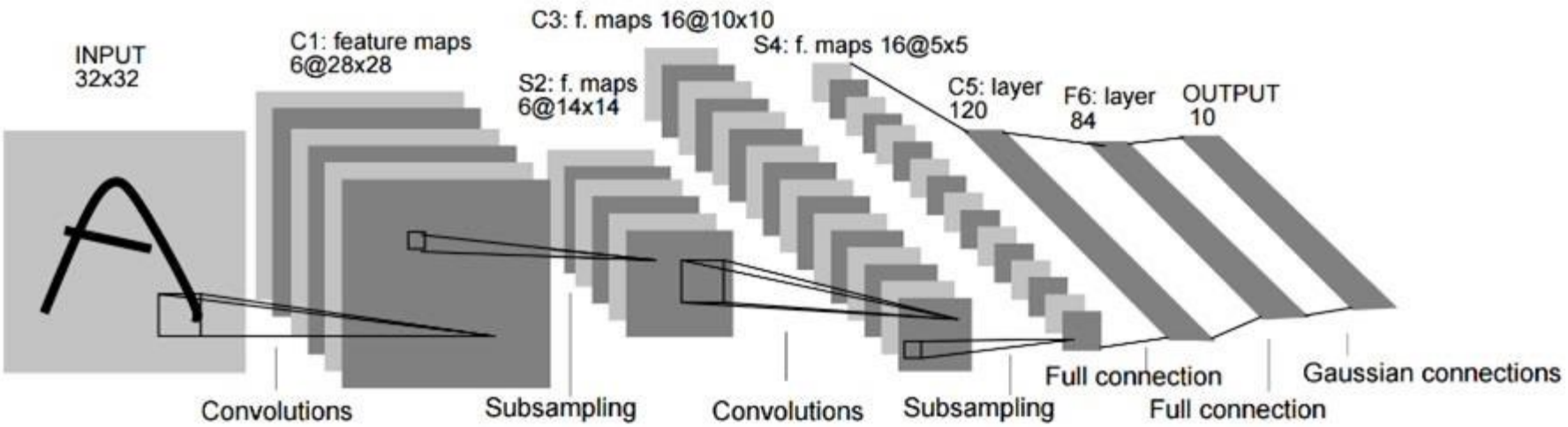
**Topographical mapping in the cortex:**

nearby cells in cortex represented nearby regions in the visual field



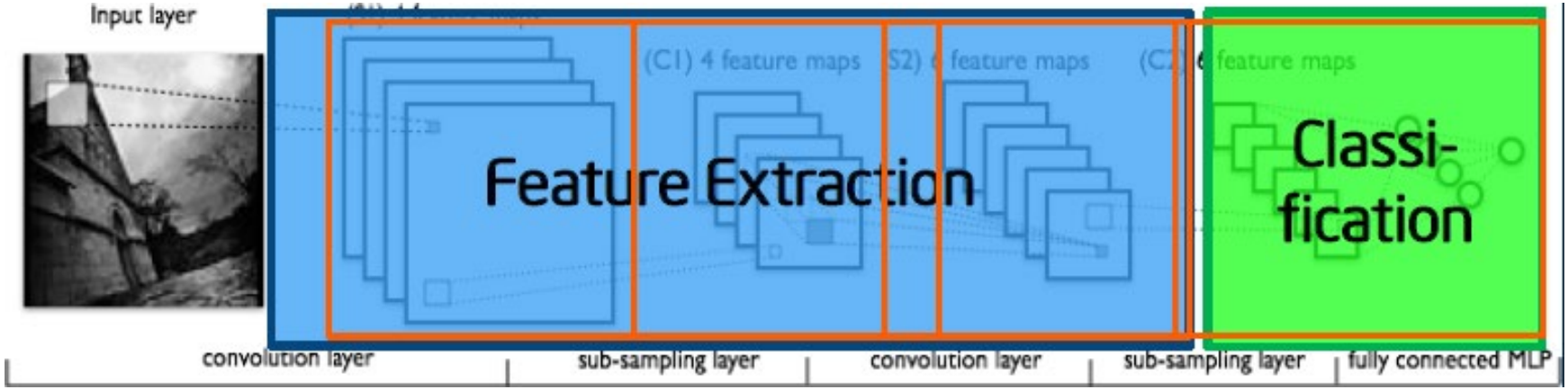
# LeNet [LeCun et al. 1998]




- See LeCun paper (1998) on text recognition:

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

# Convolutional NN (CNN)

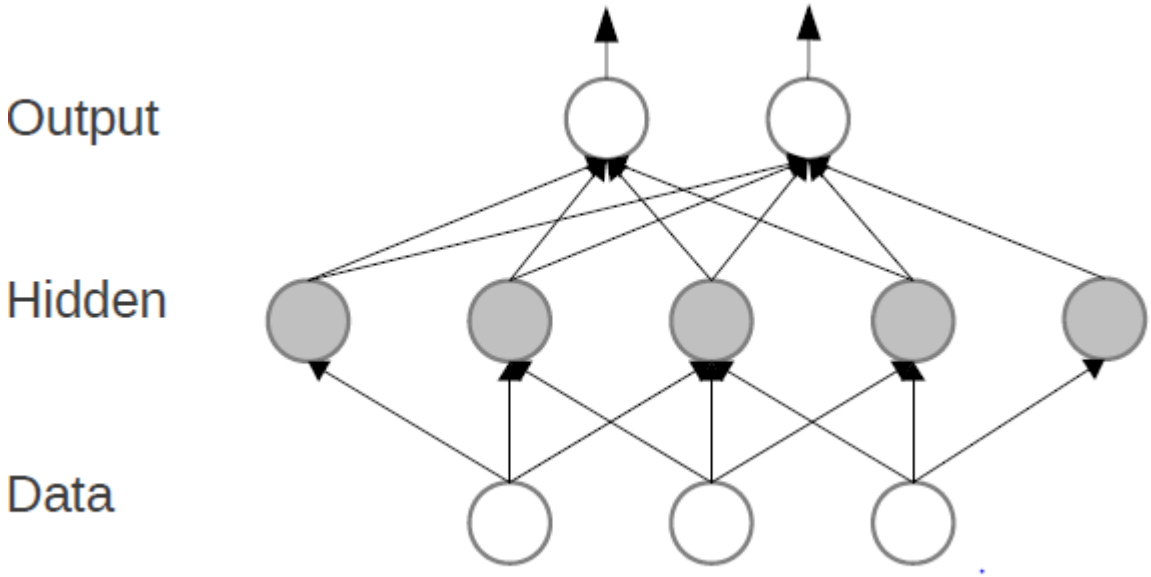



- Convolutional layer
- Sub-sampling layer
- Fully connected layers

# Basic Concept of CNN

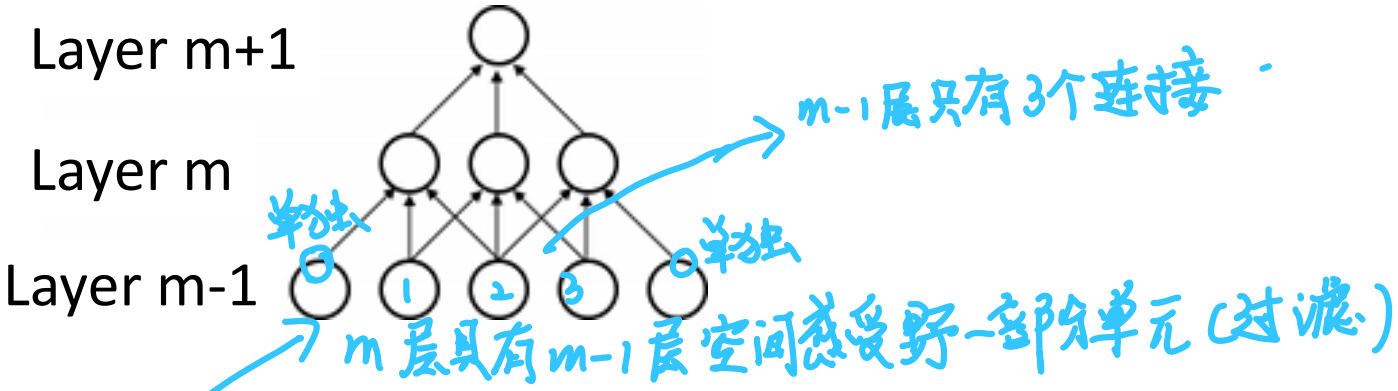
- Here's a one-dimensional convolutional neural network
- Each hidden neuron applies ***the same localized, linear filter*** to the input

濾波器



# Sparse Connectivity

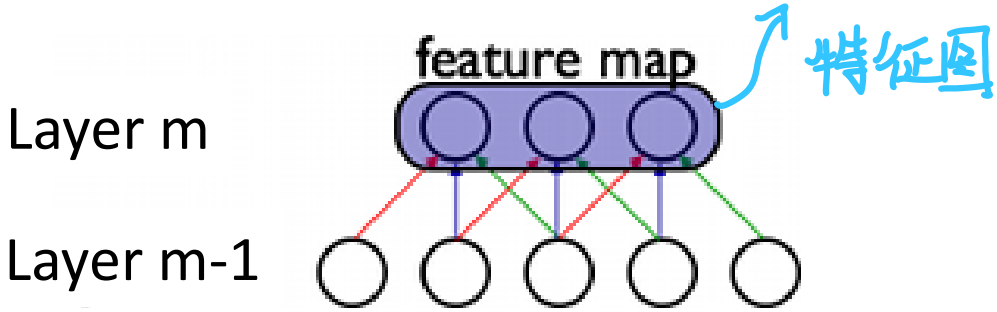
- CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers, i.e., the inputs of hidden units in layer **m** are from a subset of units in layer **m-1**, units that have spatially contiguous *receptive fields*




Imagine that layer **m-1** is the input retina. In the above figure, units in layer **m** have receptive fields of width 3 in the input retina and are thus only connected to 3 adjacent neurons in the retina layer. Units in layer **m+1** have a similar connectivity with the layer below. We say that their receptive field with respect to the layer below is also 3, but their receptive field with respect to the input is larger (5). Each unit is unresponsive to variations outside of its receptive field with respect to the retina. The architecture thus ensures that **the learnt "filters" produce the strongest response to a spatially local input pattern.**

# Shared Weights

- In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a *feature map*




In the above figure, we show 3 hidden units belonging to the same feature map. Weights of the same color are shared—constrained to be identical. Gradient descent can still be used to learn such shared parameters, with only a small change to the original algorithm.

Replicating units in this way allows for features to be detected *regardless of their position in the visual field.* Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt. The constraints on the model enable CNNs to achieve better generalization on vision problems.

# Details and Notation

- A feature map is obtained by repeated application of a function across sub-regions of the entire image, in other words, by *convolution* of the input image with a linear filter, adding a bias term and then applying a non-linear function.
- If we denote the k-th feature map at a given layer as $h^k$, whose filters are determined by the weights $W^k$ and bias $b_k$, then the feature map is obtained using convolution as follows (for *tanh* non-linearities):

$$h_{ij}^k = tanh((W^k * x)_{ij} + b_k)$$

# Convolutional Layers

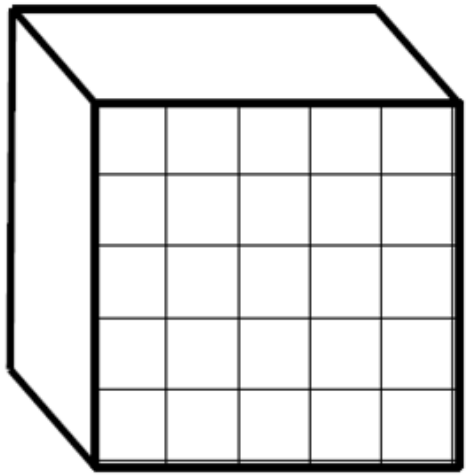
- Suppose that we have some NxN square neuron layer which is followed by our convolutional layer. If we use an mxm filter ω, our convolutional layer output will be of size (N−m+1)x(N−m+1).

- In order to compute the pre-nonlinearity input to some unit $X_{ij}^{l}$ in our layer, we need to sum up the contributions (weighted by the filter components) from the previous layer cells:

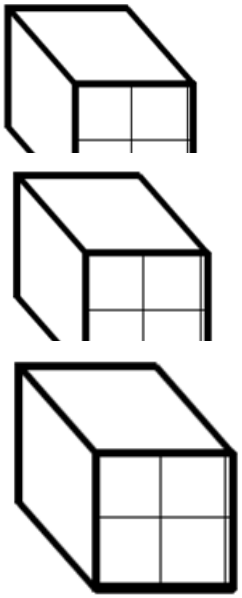
$$x_{ij}^{\ell} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{\ell-1}.$$

# Convolution in 2D

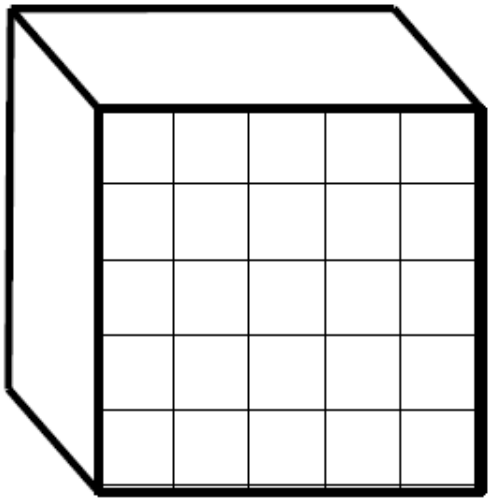
**Input "image"**



**Filter bank**



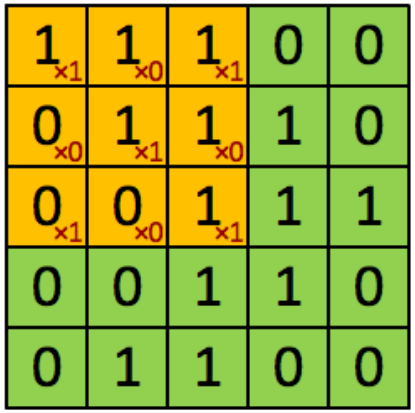
**Output map**



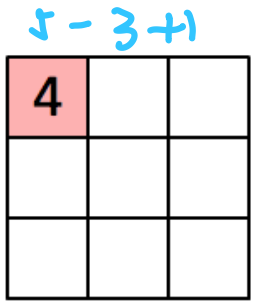
## Convolutional layers :

a rectangular grid of neurons.
The previous layer is also a rectangular grid of neurons.

Each neuron takes inputs from a rectangular section of the previous layer; the weights for this rectangular section are the same for each

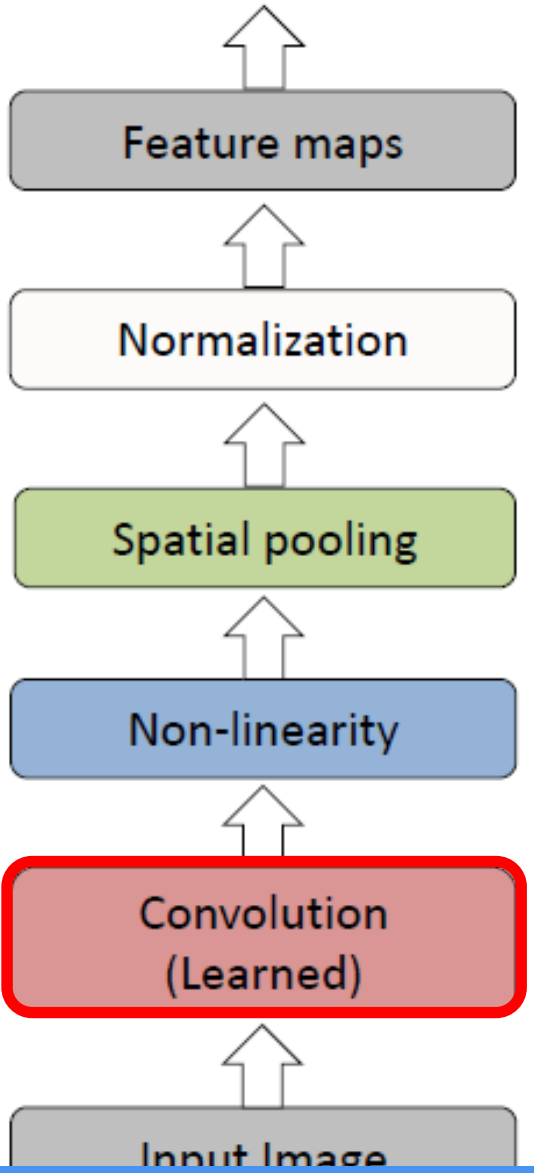


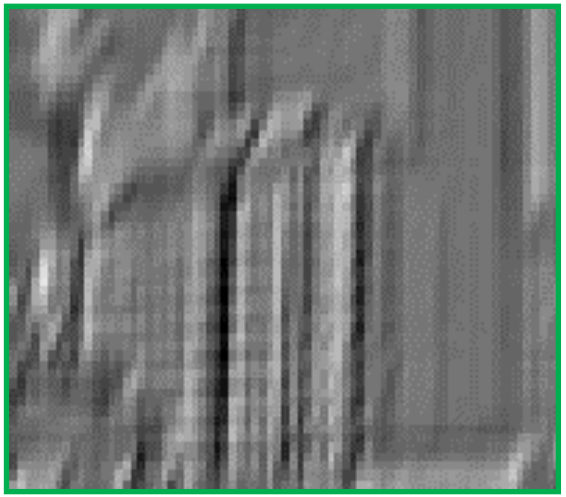

Image

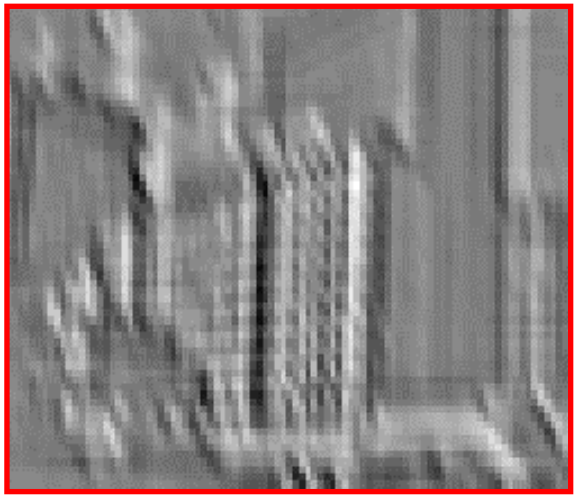


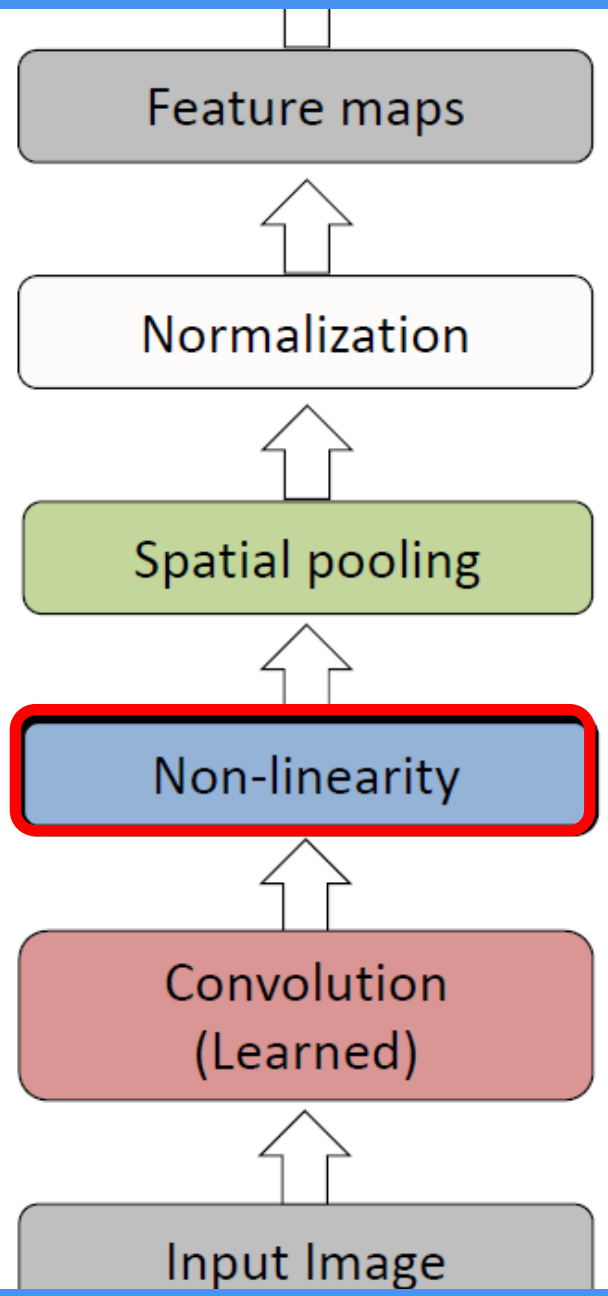

Convolved

# Convolutional Neural Networks



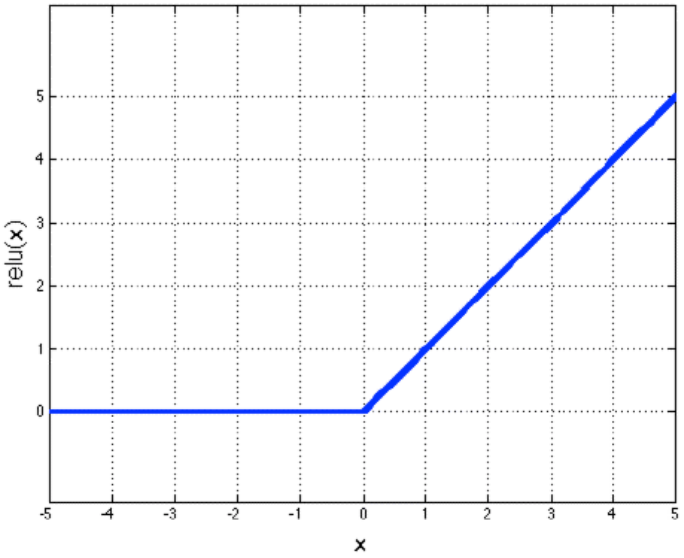





Non-saturating function

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

Rectified Linear Unit (ReLU)



# Benefits of using ReLU

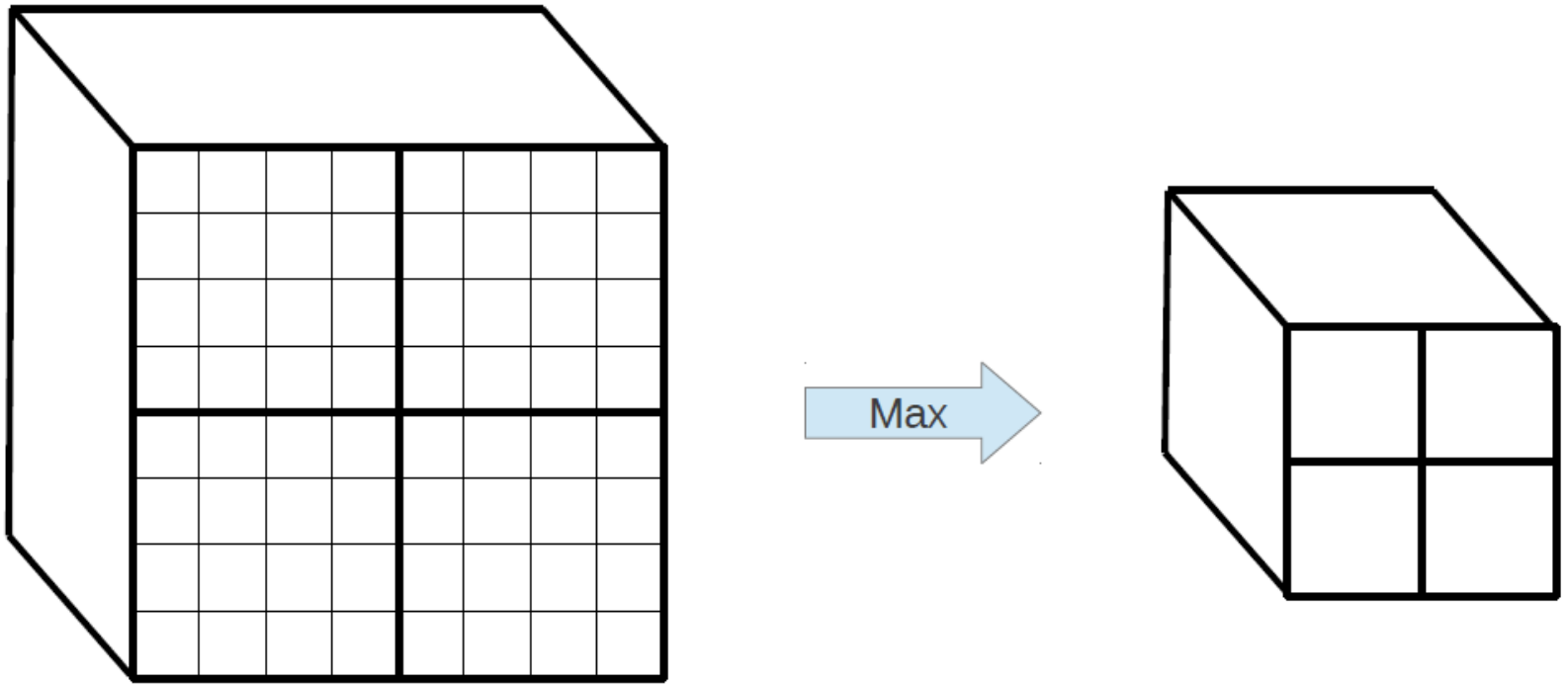
- ReLUs are much simpler computationally
  - The forward and backward passes through an ReLU are both just a simple *if* statement
  - The sigmoid activation requires computing an exponent
  - This advantage is huge when dealing with big networks with many neurons, and can significantly reduce both training and evaluation times

# Benefits of using ReLU

- Sigmoid activations are easier to saturate
  - There is a comparatively narrow interval of inputs for which the sigmoid's derivative is *sufficiently nonzero*
  - In other words, once a sigmoid reaches either the left or right plateau, it is almost meaningless to make a backward pass through it, since the derivative is very close to 0
- ReLUs only saturate when the input is less than 0
  - Even this saturation can be eliminated using leaky ReLUs
- For very deep networks, saturation hampers learning, and so ReLUs provide a nice workaround

# Local pooling operation

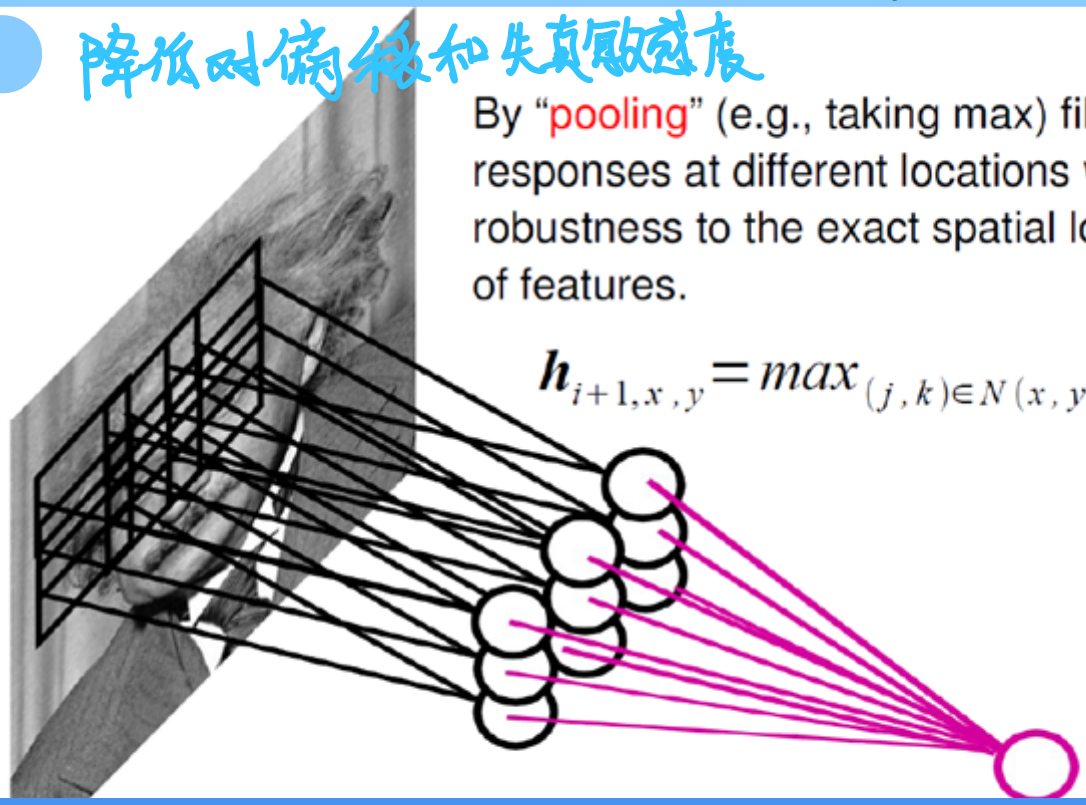
In order to reduce variance, pooling layers compute the max or average value of a particular feature over a region of the image. This will ensure that the same result will be obtained, even when image features have small translations



# More on Pooling operation

**Subsampling (pooling) Mechanism** 特征相对位置重要

- The exact positions of the extracted features are not important
- Only relative position of a feature to another feature is relevant
- Reduce spatial resolution – Reduce sensitivity to shift and distortion 降低对偏移和失真敏感度



# In another word ...

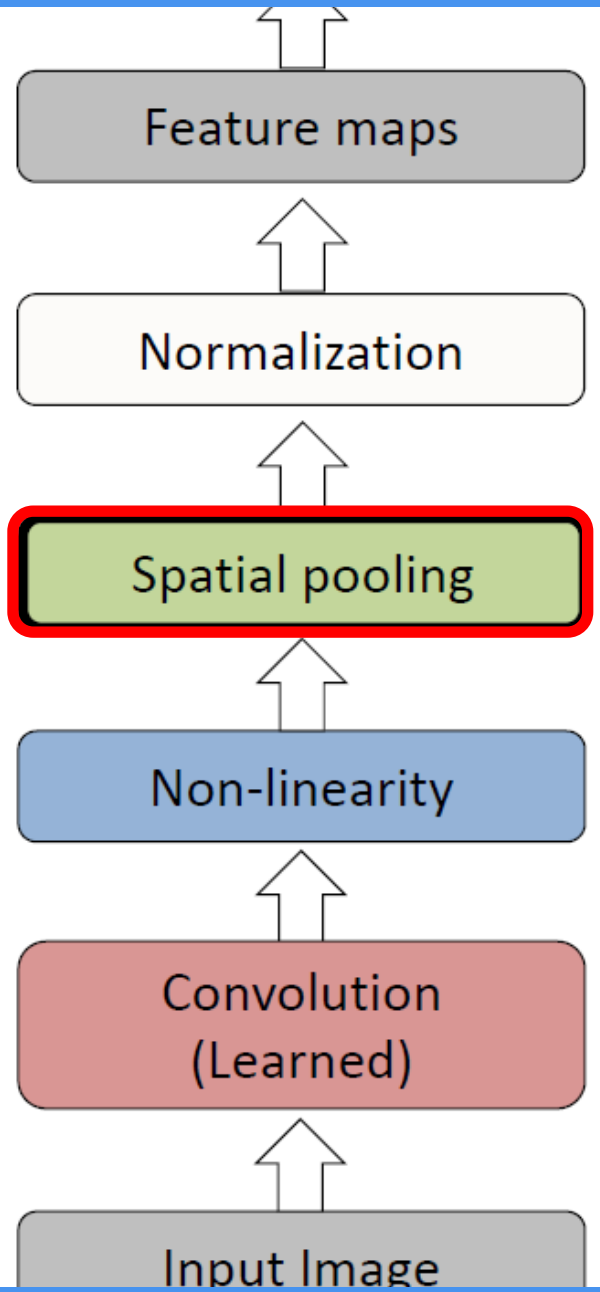
1. In general terms, the objective of pooling is to transform the joint feature representation into a new, more usable one that *preserves important information while discarding irrelevant detail*, the crux of the matter being to determine what falls in which category

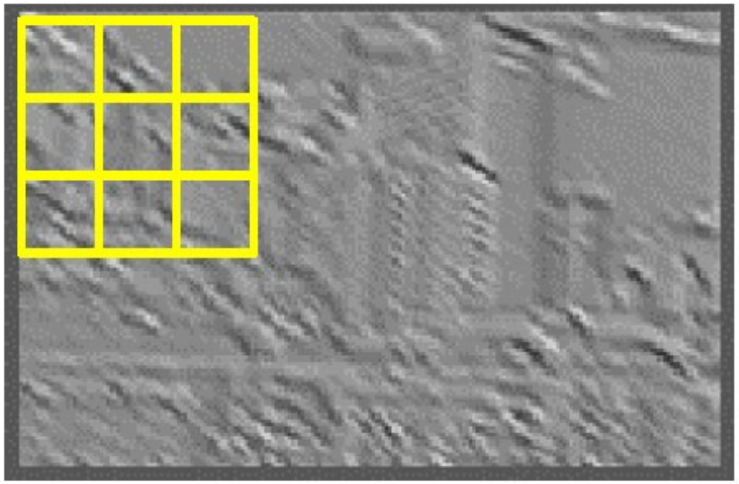
↳ 保留重要信息去无用细节

2. Achieving invariance to changes in position or lighting conditions, robustness to clutter, and compactness of representation, are all common goals of pooling

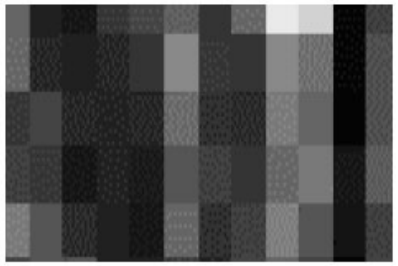
# Max Pooling

- ***Max pooling*** is a form of non-linear down-sampling, which partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value
- Max pooling is useful in vision for two reasons
  - By eliminating non-maximal values, it reduces computation for upper layers
  - It provides a form of translation invariance
- Since it provides additional robustness to position, max-pooling is a "smart" way of reducing the dimensionality of intermediate representations



Max pooling



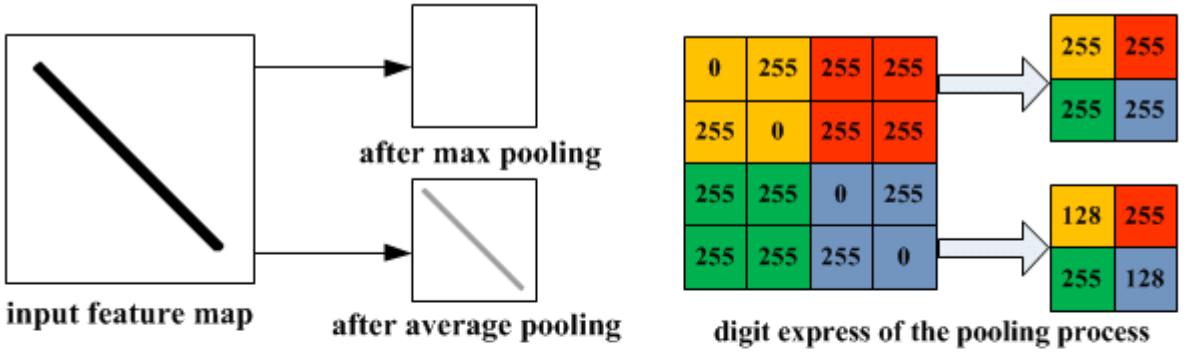
## Max pooling:

- **a non-linear down-sampling**
- **Provide *translation invariance***

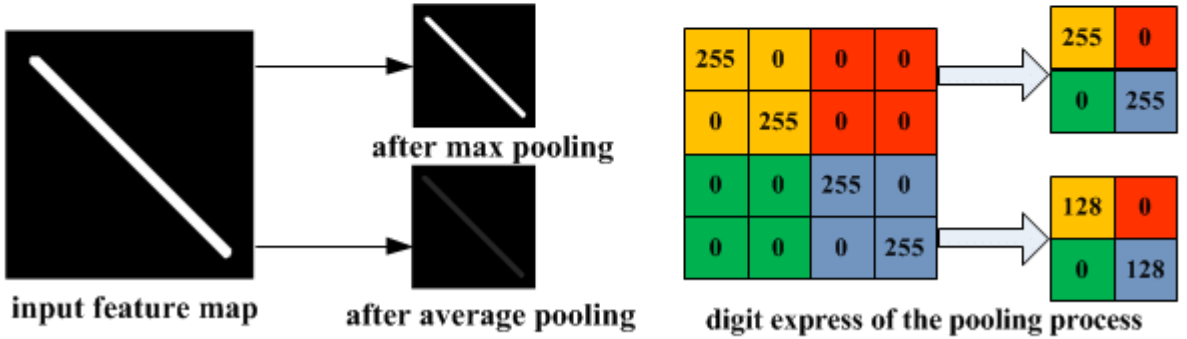
提供平移不变性

# Max Pooling & Average Pooling

都有问题




(a) Illustration of max pooling drawback




(b) Illustration of average pooling drawback

# Example

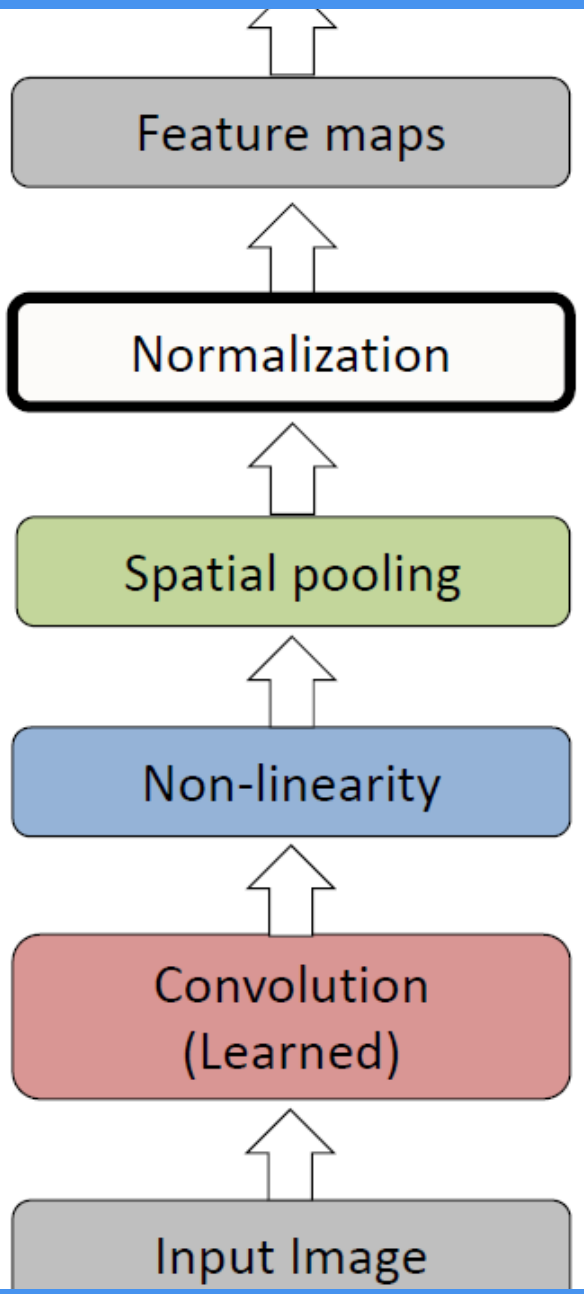
origin image

convolution 1

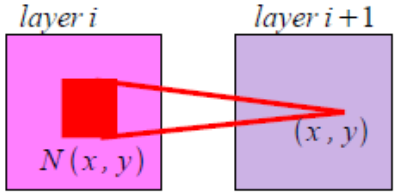
convolution 2

**Down-sampled 1**

**Down-sampled 2**

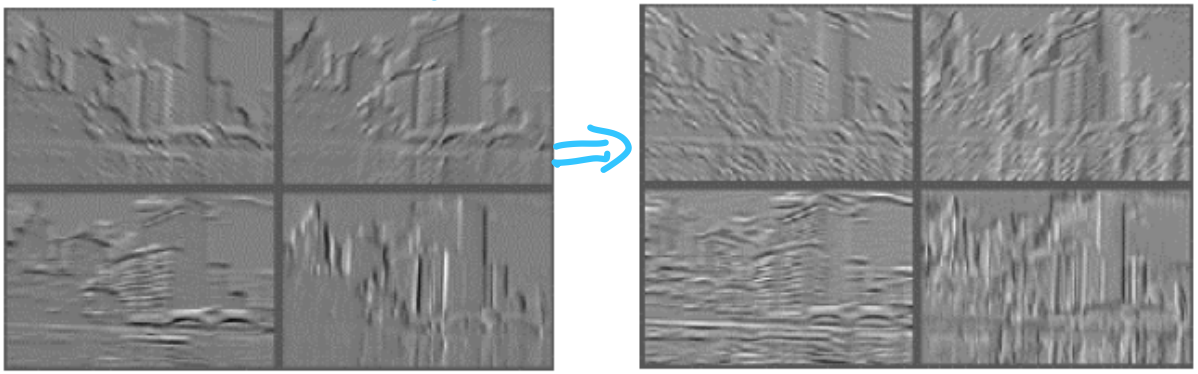



**Local Contrast Normalization** (over space / features)




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

整个图像规一化操作



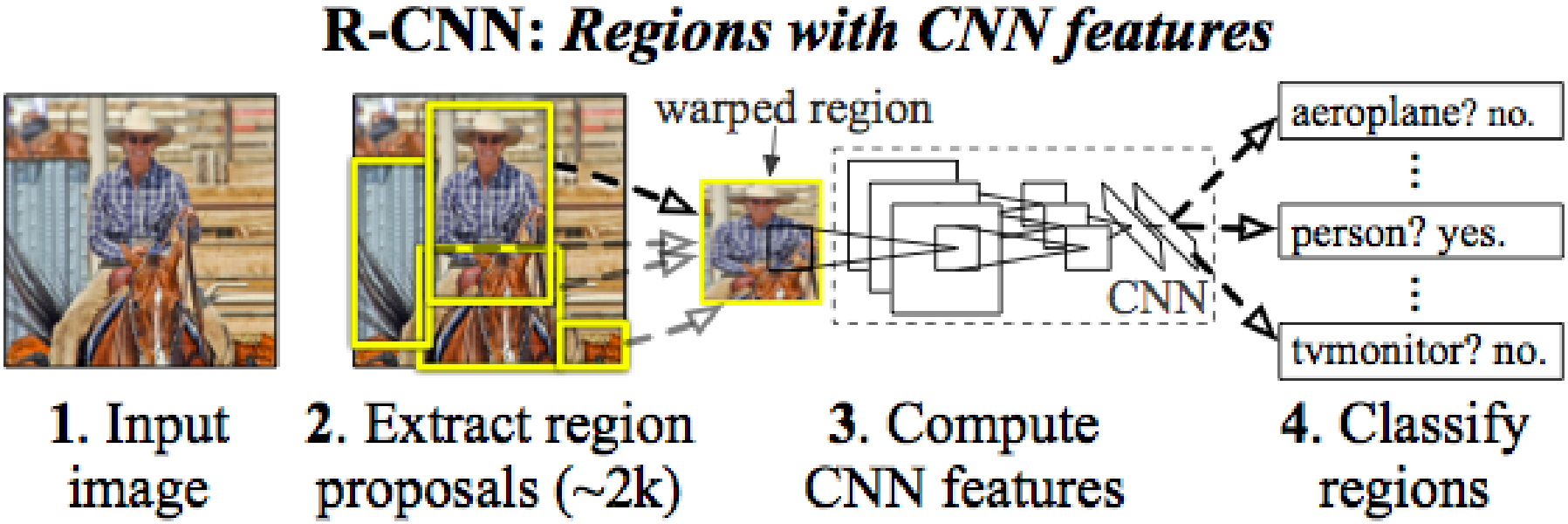
Feature Maps

Feature Maps
After Contrast
Normalization

# What is really important ?

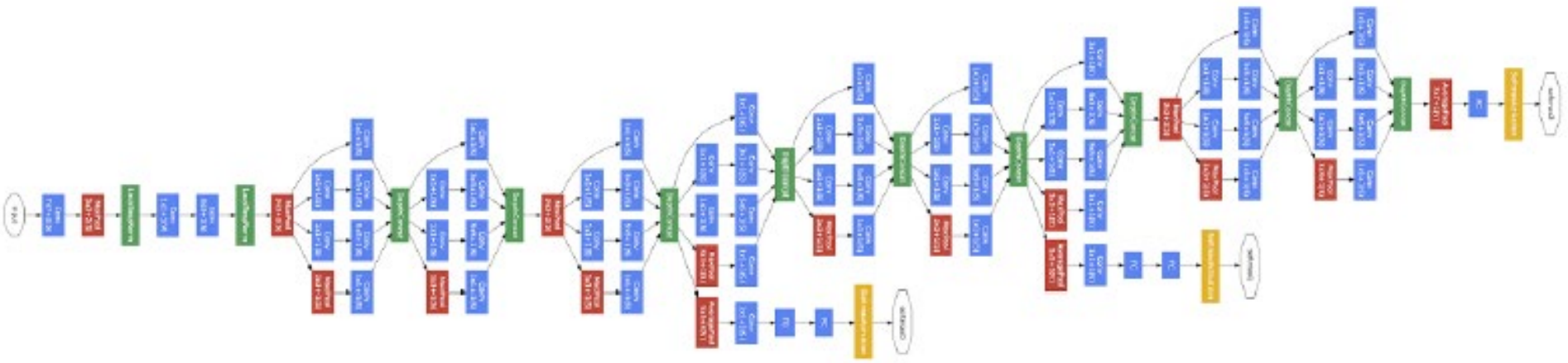
- The convolutional layers are the most important part
- A ***pre-trained*** network for image classification can be used for many different vision tasks.

**Detection:**



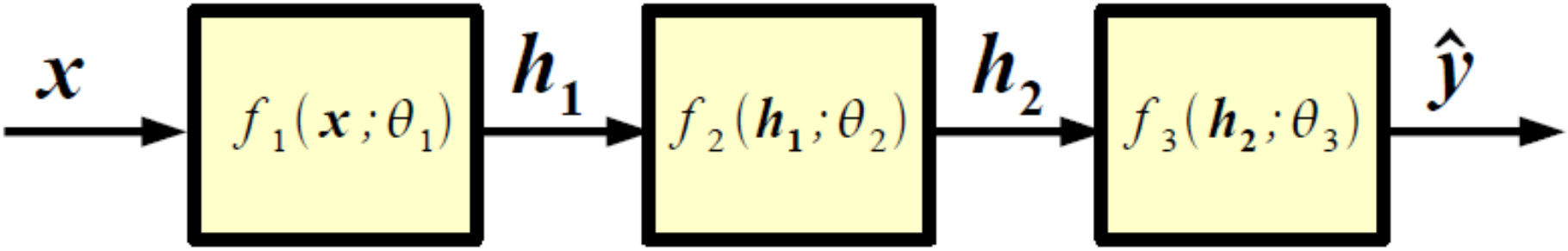
# GoogLeNet

- 22 layers' deep networks



Convolution **Max Pooling** **Softmax** **Concatenation**

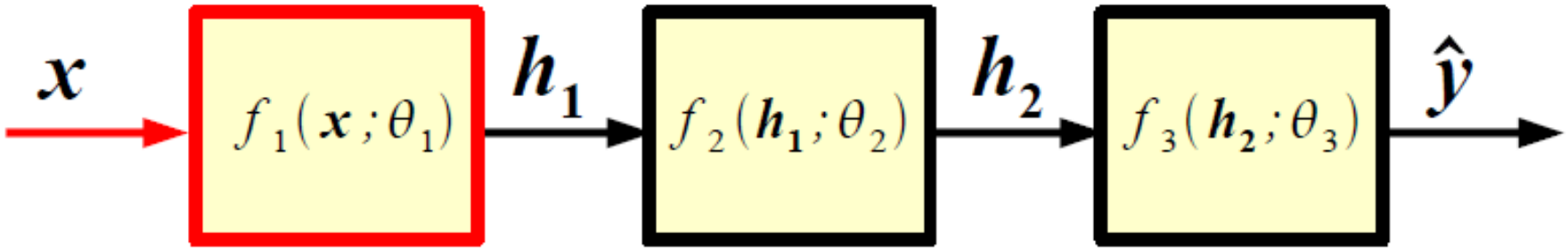
# Idea of CNN training




**NOTE:**

**In practice, any differentiable non-linear transformation is potentially good.**

# Forward Propagation (FP)



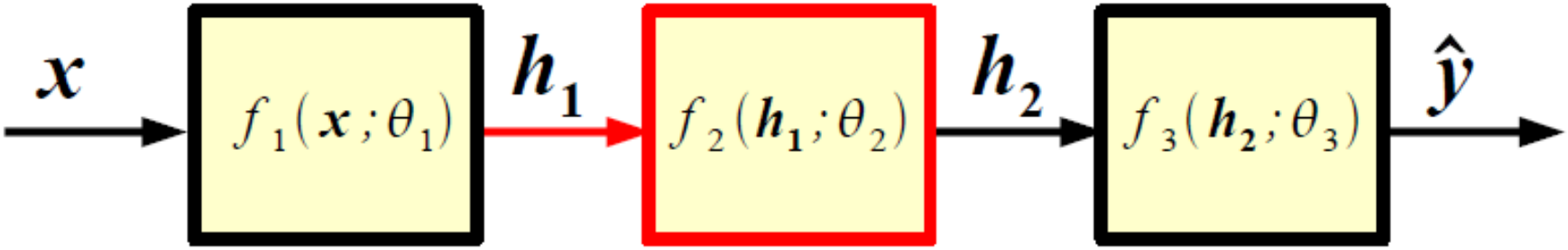

1) Given $x$ compute: $h_1 = f_1(x;\theta_1)$

For instance,

$$h_1 = max(0, W_1 x + b_1)$$

ReLU Rectified Linear Units.

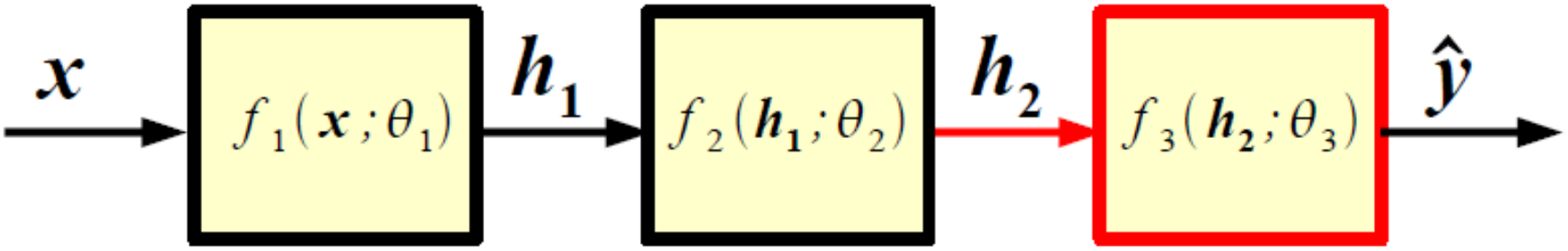
# Forward Propagation (FP)




1) Given $x$ compute: $h_1 = f_1(x;\theta_1)$

2) Given $h_1$ compute: $h_2 = f_2(h_1;\theta_2)$

# Forward Propagation (FP)




1) Given $x$ compute: $h_1 = f_1(x;\theta_1)$

2) Given $h_1$ compute: $h_2 = f_2(h_1;\theta_2)$

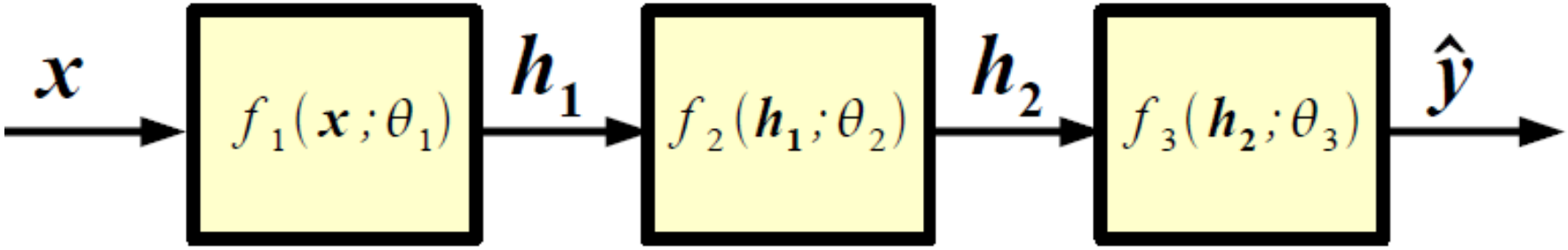
3) Given $h_2$ compute: $\hat{y} = f_3(h_2;\theta_3)$

For instance,

$$\hat{y}_i = p(class = i|x) = \frac{e^{W_{3i}h_2 + b_{3i}}}{\sum_k e^{W_{3k}h_2 + b_{3k}}}$$

Softmax output → probability that input x belongs

# Forward Propagation (FP)




1) Given $x$ compute: $h_1 = f_1(x;\theta_1)$

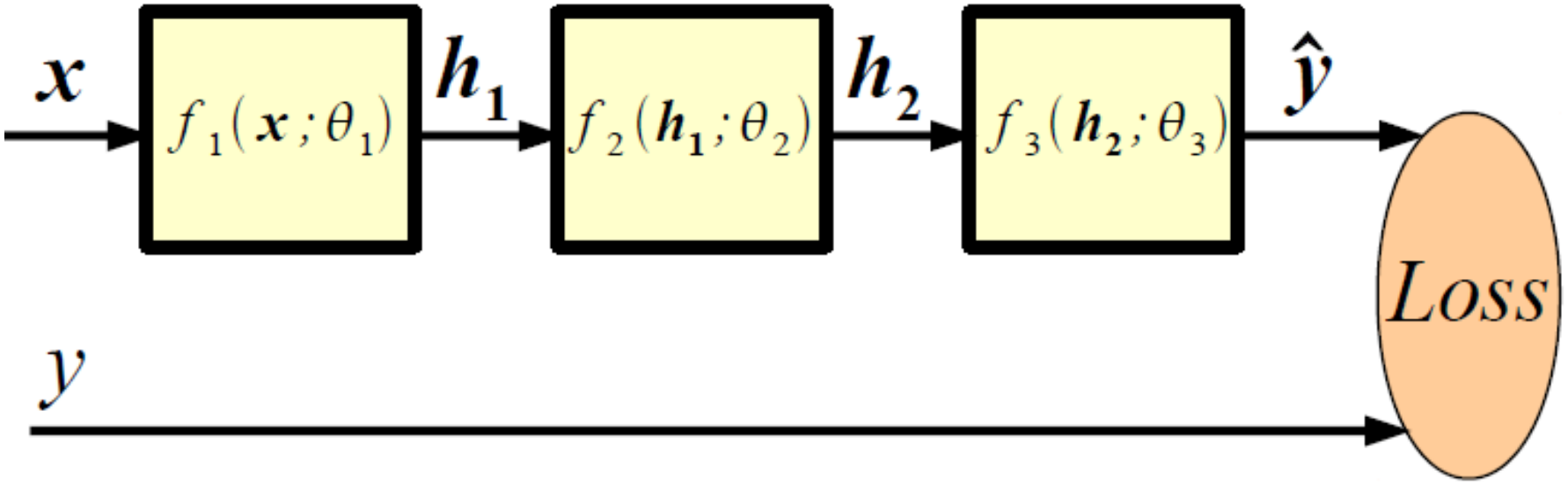
2) Given $h_1$ compute: $h_2 = f_2(h_1;\theta_2)$

3) Given $h_2$ compute: $\hat{y} = f_3(h_2;\theta_3)$

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

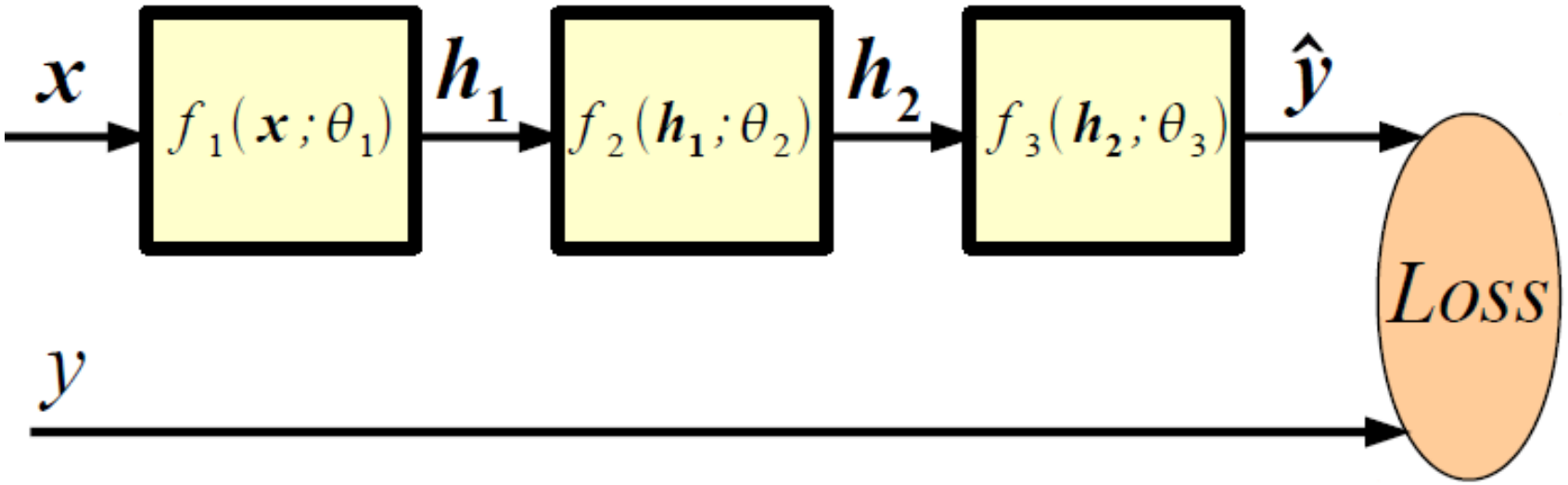
# Loss Function




The measure of how well the model fits the training set can be given by a suitable ***loss function***: $L(x, y; \theta)$

The loss depends on the input $x$ , the target label $y$ , and the parameters $\theta$

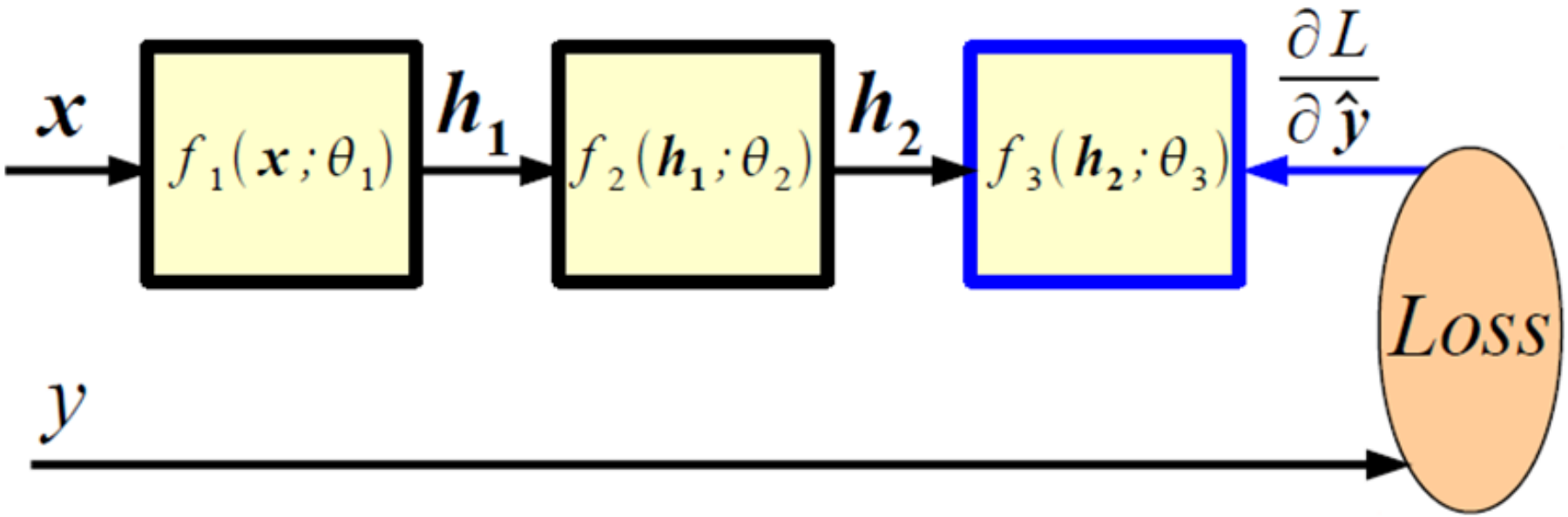
# Backward Propagation (BP)



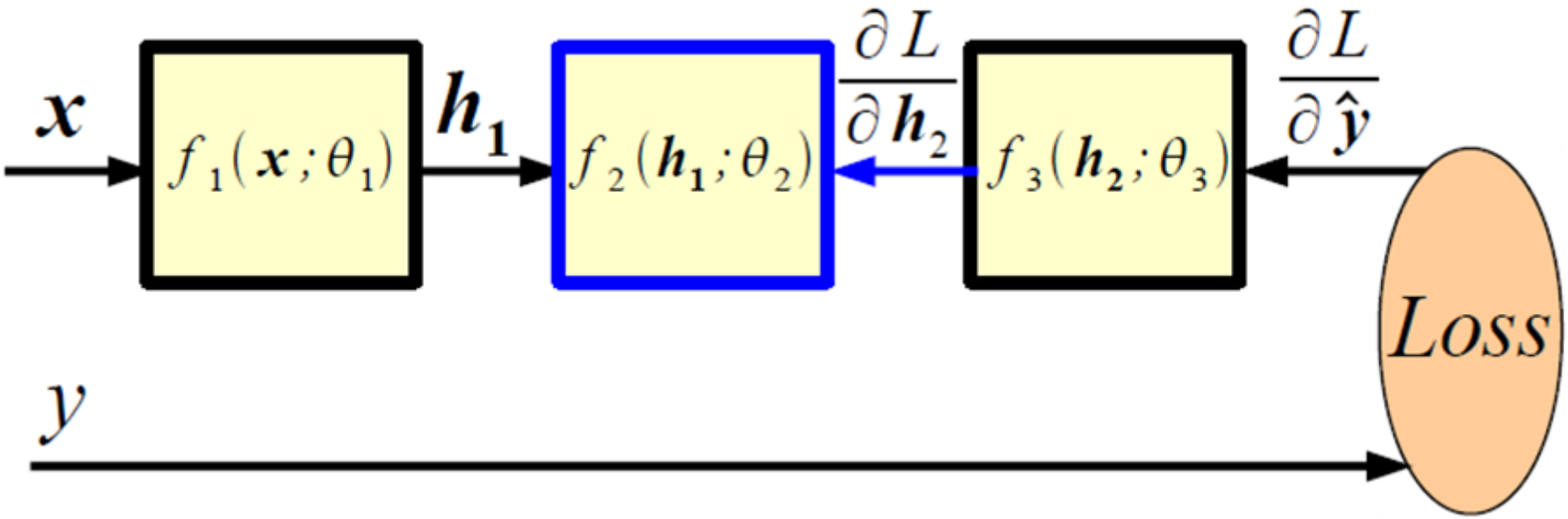

**Q.: how to tune the parameters to decrease the loss?**

If loss is differentiable we can compute gradients.
We can use **back-propagation,** to compute the gradients w.r.t. parameters at the lower layers

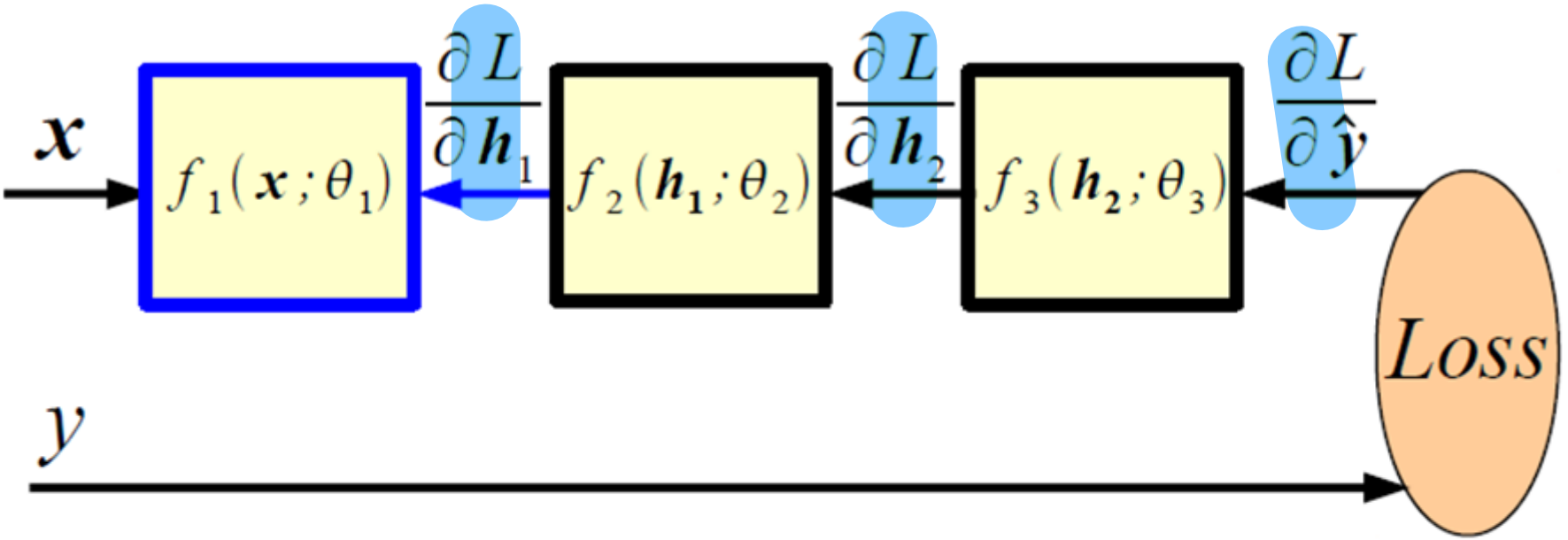
# Backward Propagation (BP)



# Backward Propagation (BP)



# Backward Propagation (BP)



# Optimization

## Stochastic Gradient Descent (on mini-batches):

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

## Stochastic Gradient Descent with Momentum:

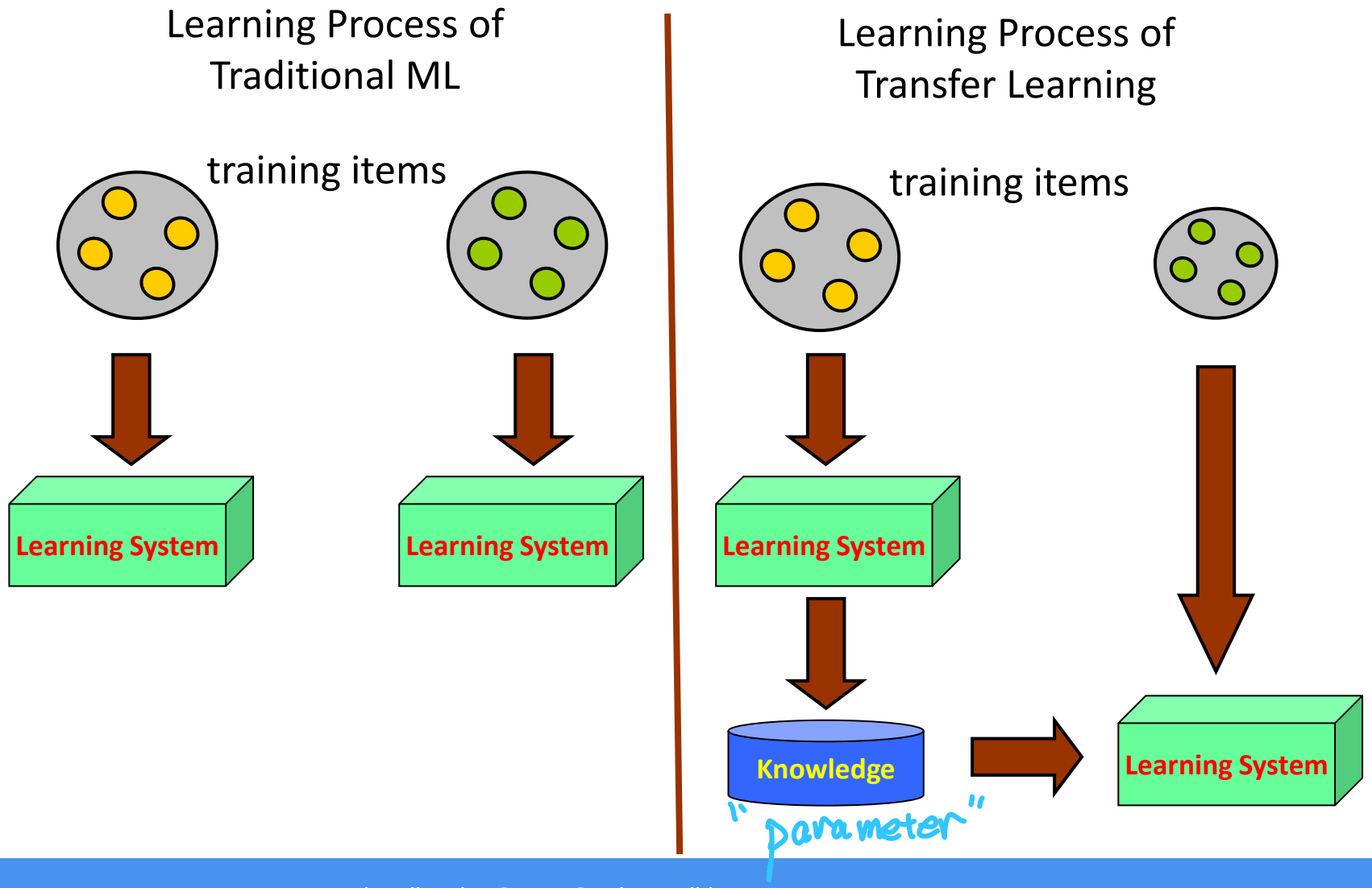
$$\theta \leftarrow \theta - \eta \Delta$$

$$\Delta \leftarrow 0.9\,\Delta + \frac{\partial L}{\partial \theta}$$

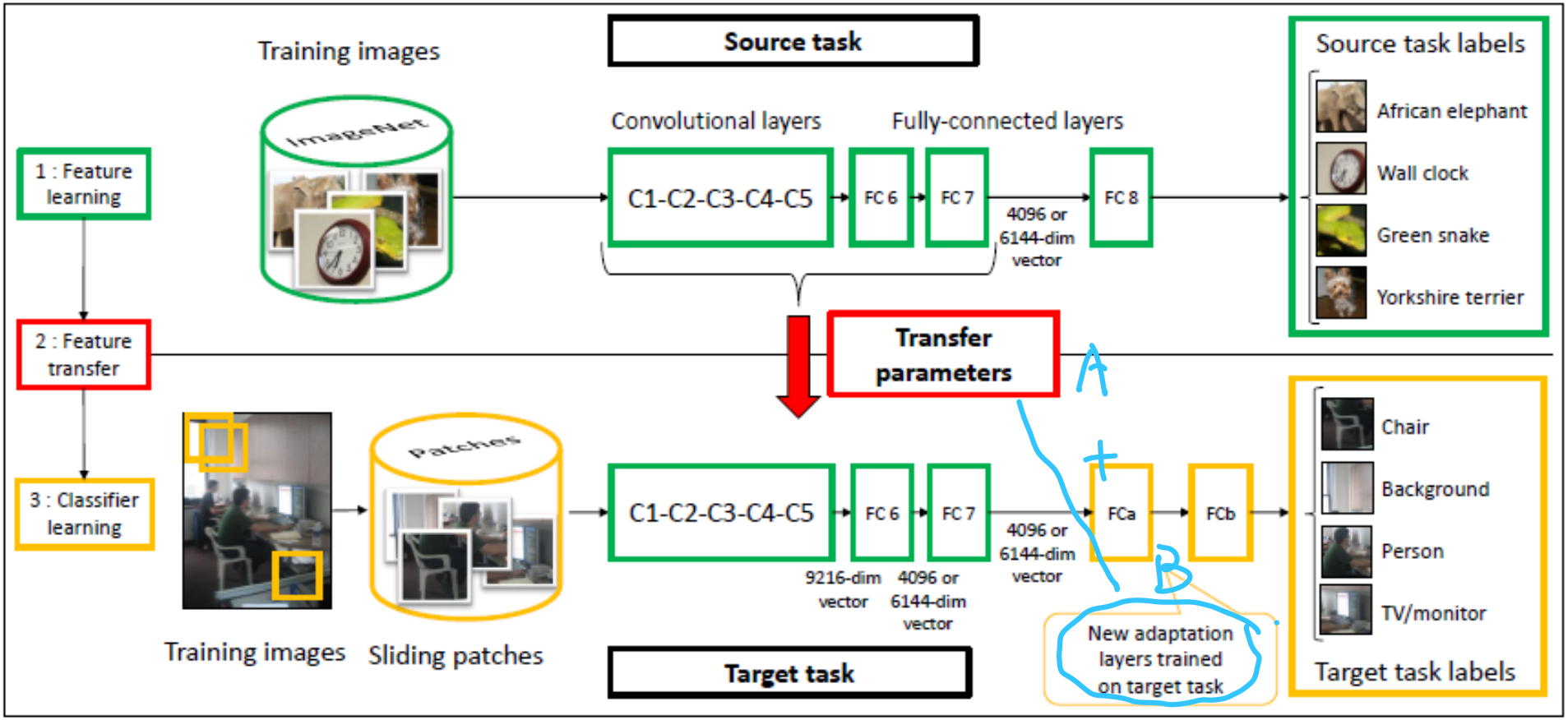
# Transfer Learning (TL) 迁移学习

- The ability of a system to recognize and apply knowledge and skills learned in previous tasks to novel tasks (in new domains)
- TL is motivated by human learning: people can often transfer knowledge learnt previously to novel situations
  - Chinese → English
  - mathematics → computer science
  - network technology for internet → social network

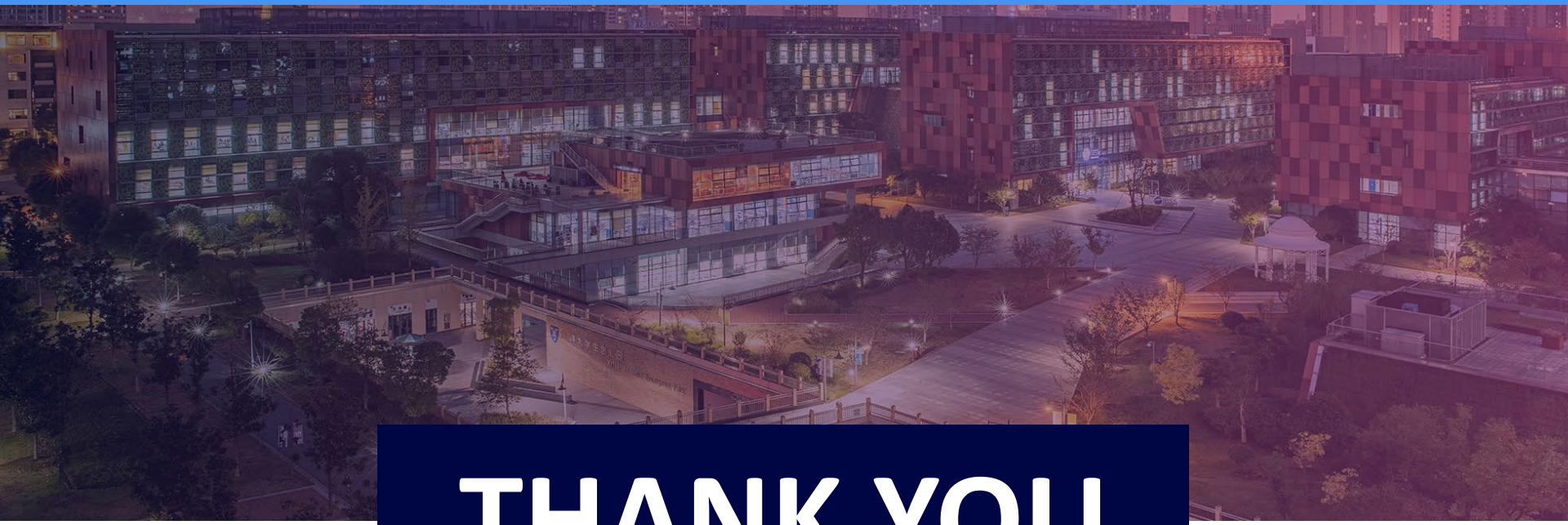
# Traditional ML vs. TL



# Deep CNN for Knowledge Transfer




The network is trained on the source task (top row) with a large amount of available labelled images. Pre-trained parameters of the internal layers of the network (C1-FC7) are then transferred to the target tasks (bottom row). To compensate for the different image statistics (type of objects, typical viewpoints, imaging conditions) of the source and target data, an adaptation layer (fully connected layers FCa and FCb) is added and trained on the labelled data of the target task.



# THANK YOU

## VISIT US

WWW.XJTLU.EDU.CN

## FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學