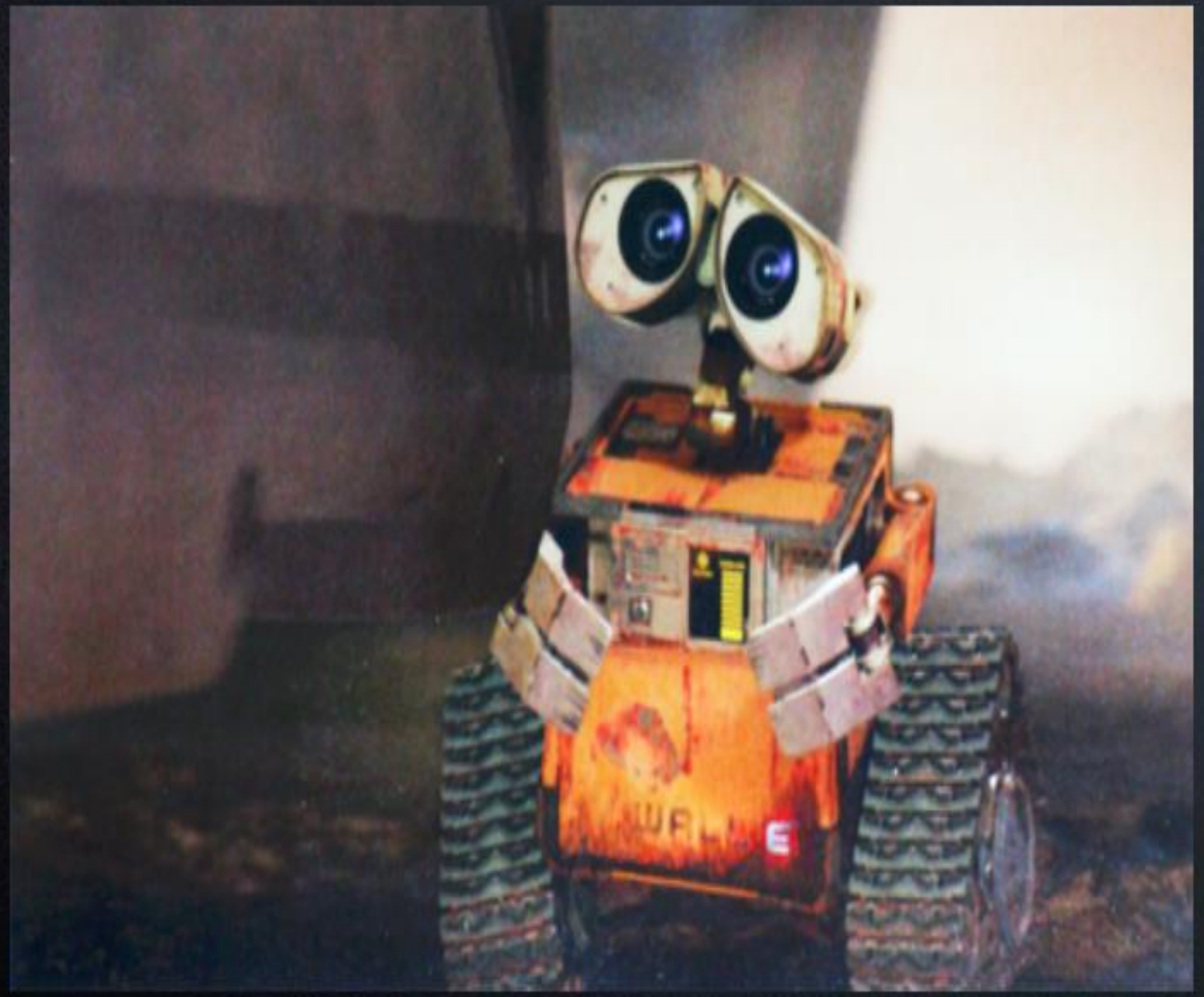
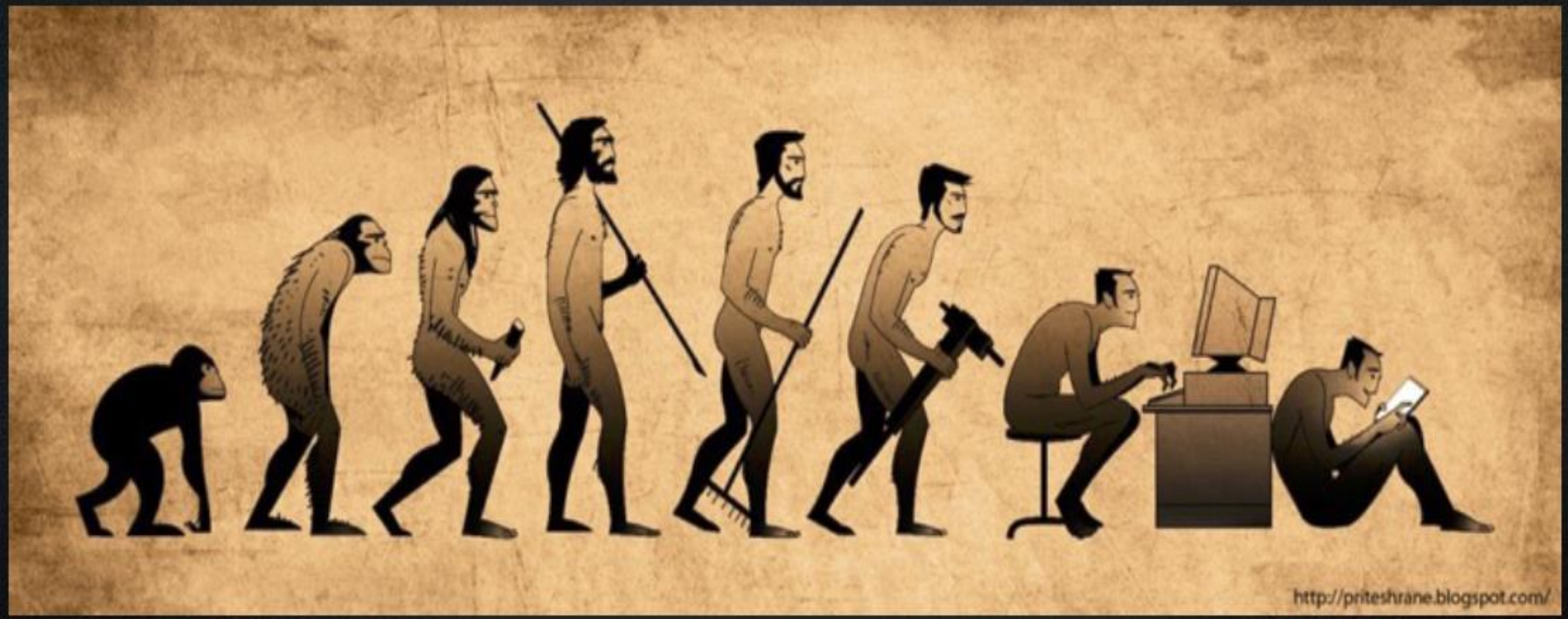


什么是
人工智能?



学习的能力，是智能的本质！

万物互联

万物智能

大数据时代

IMAGENET

www.image-net.org

22K categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formation
- Sport Activities

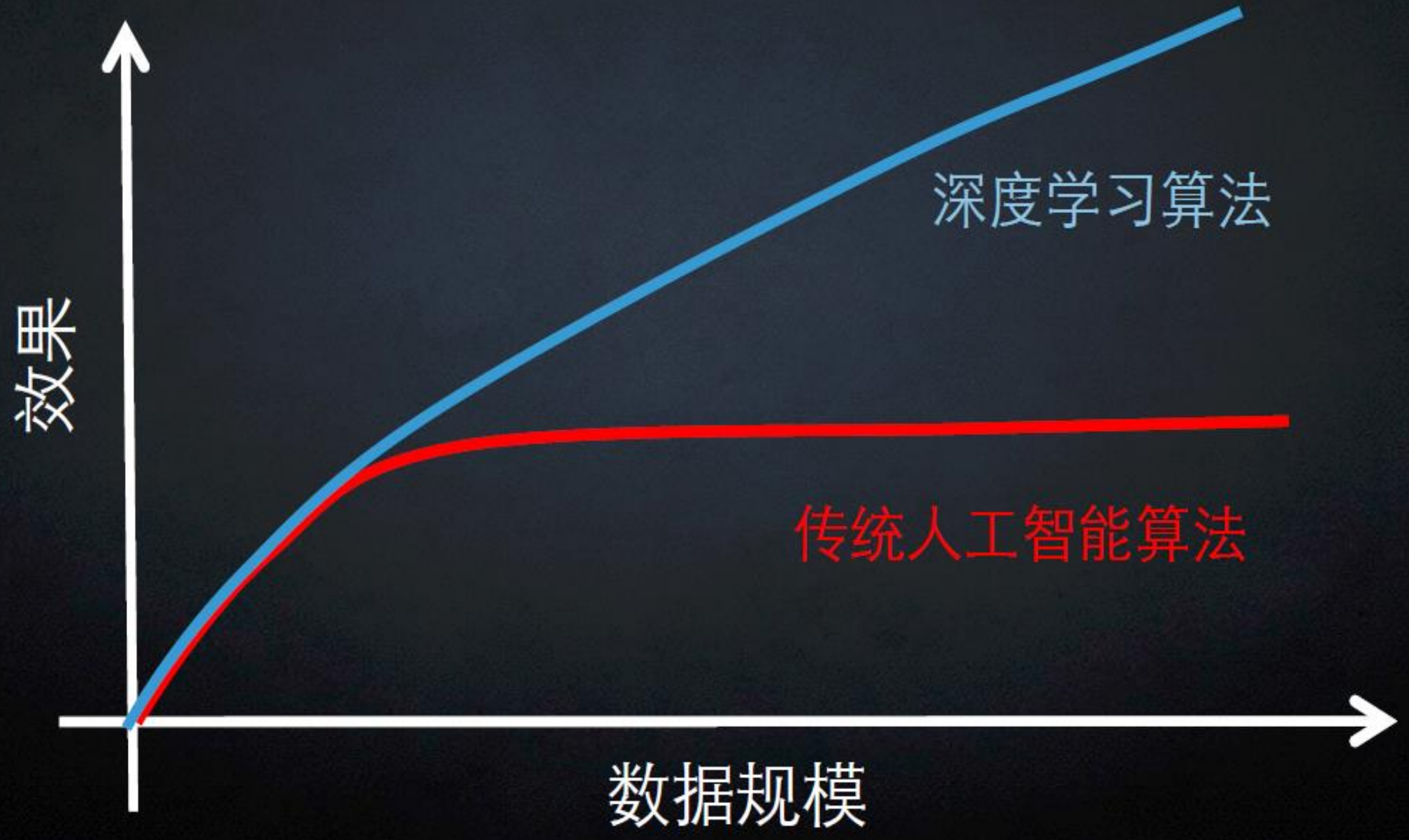
为什么围棋选手不信阿法狗呢？

结果呢？

4:1 李世石惨败

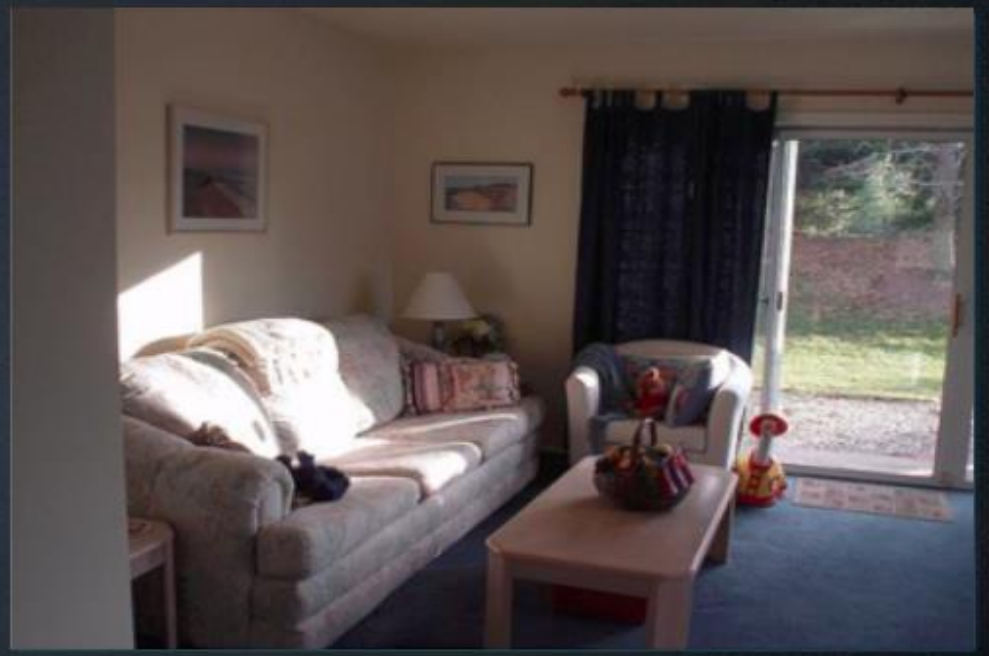
人工智能的时代已经来临

为什么人工智能技术这么厉害？


深度学习算法
传统人工智能算法
效果
数据规模

述说图片的故事

A yellow bus driving down a road with green trees and green grass in the background.



Living room with white couch and blue carpeting. The room in the apartment gets some afternoon sun.

这些字幕是深度学习程序写的

无人驾驶汽车：

1.物体检测

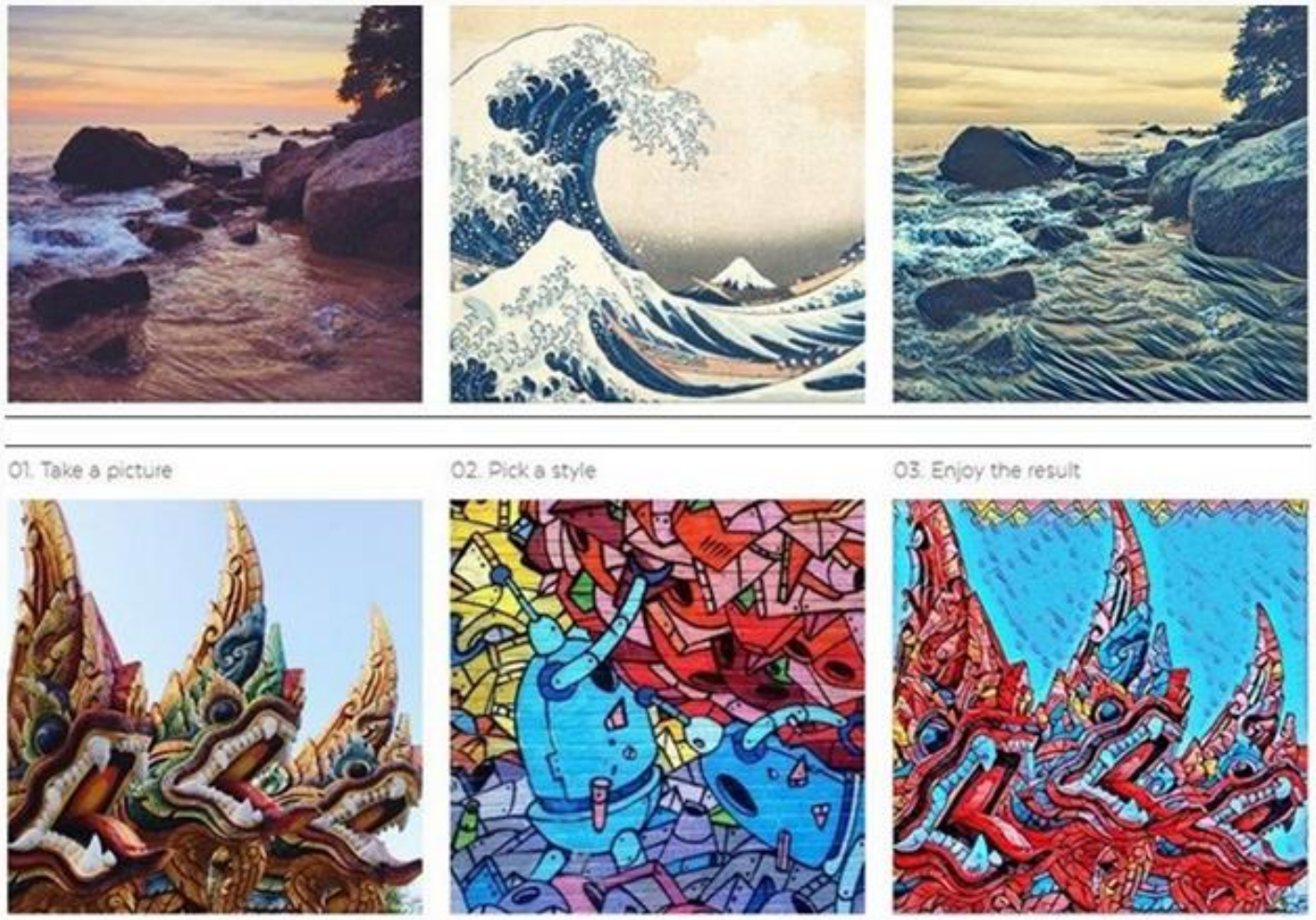
2.行人检测

3.标志识别

4.速度识别

。 。 。

深度学习



黑科技： Image Transfer

Content + Style = Interesting thing

图像分类

- 计算机视觉：

  - 图像分类任务

（假设我们有一系列的标签：狗，猫，汽车，飞机。。。）

→ 猫

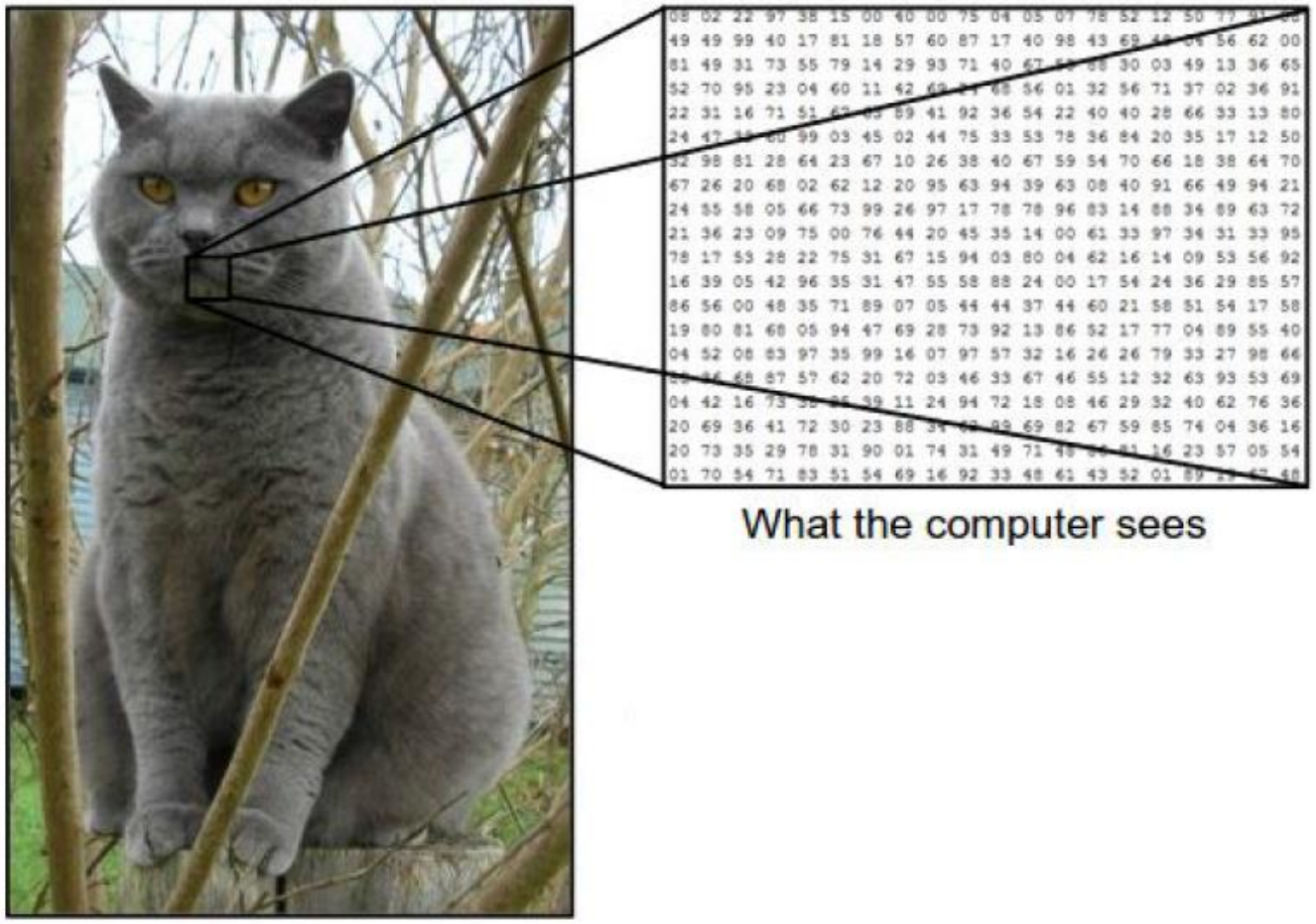
图像分类

- 计算机视觉：

  - 图像表示：计算机眼中的图像

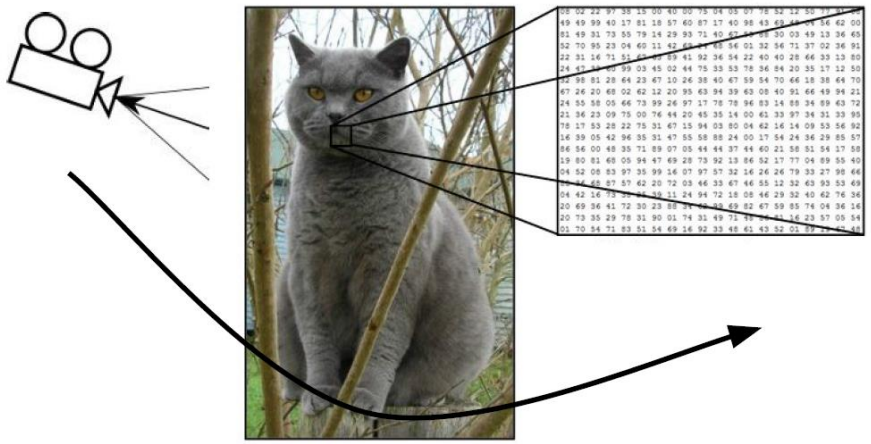
  - 一张图片被表示成三维数组的形式，每个像素的值从0到255

    例如：300*100*3



图像分类

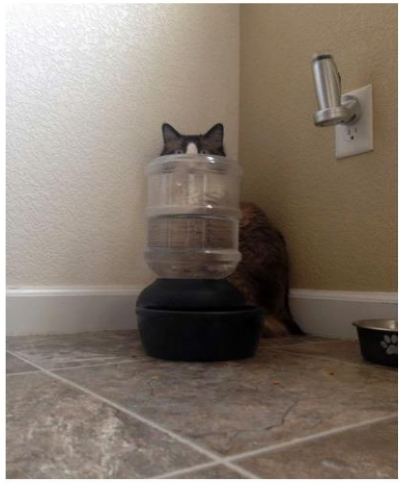
- 计算机视觉面临的挑战：
  - 照射角度：



  - 形状改变：

图像分类

- 计算机视觉面临的挑战：
  - 部分遮蔽：


  - 背景混入：

图像分类

- 机器学习常规套路：

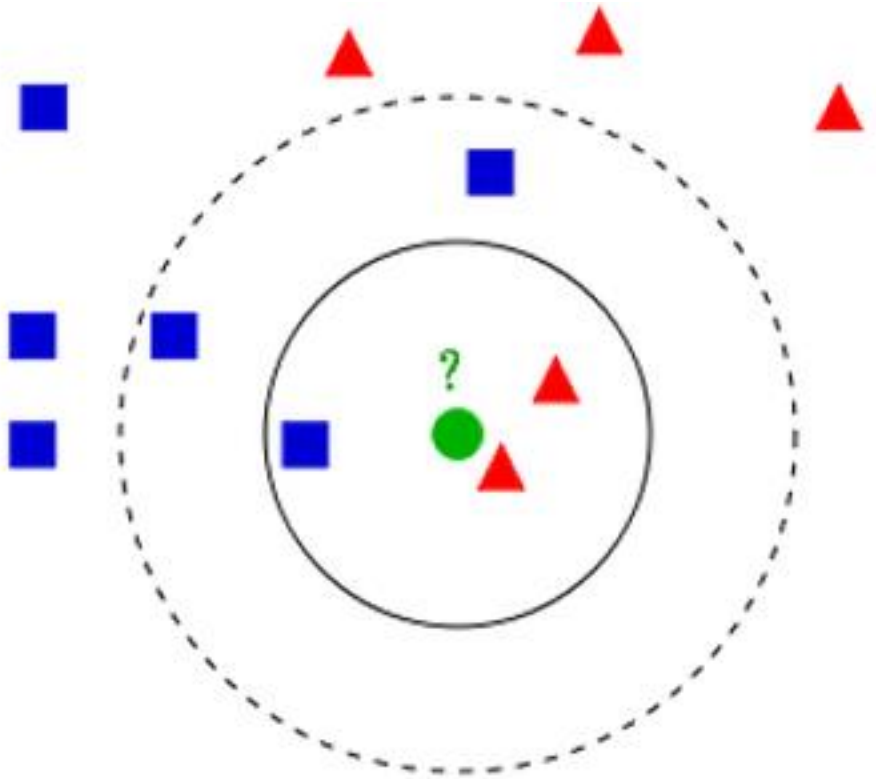
1.收集数据并给定标签
2.训练一个分类器
3.测试，评估

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

K近邻

- K近邻算法：
  - 数据：两类点方块和三角
  - 绿色的点属于方块还是三角呢？
  - K=3还是K=5？结果一样吗？



K近邻

- K近邻计算流程：

1.计算已知类别数据集中的点与当前点的距离

2.按照距离依次排序

3.选取与当前点距离最小的K个点

4.确定前K个点所在类别的出现概率

5.返回前K个点出现频率最高的类别作为当前点预测分类

K近邻

K近邻分析

- KNN 算法本身简单有效，它是一种 lazy-learning 算法。
- 分类器不需要使用训练集进行训练，训练时间复杂度为0。
- KNN 分类的计算复杂度和训练集中的文档数目成正比，
  也就是说，如果训练集中文档总数为 n，那么 KNN 的分类时间复杂度为O(n)。
- K 值的选择，距离度量和分类决策规则是该算法的三个基本要素

数据库样例：CIFAR-10

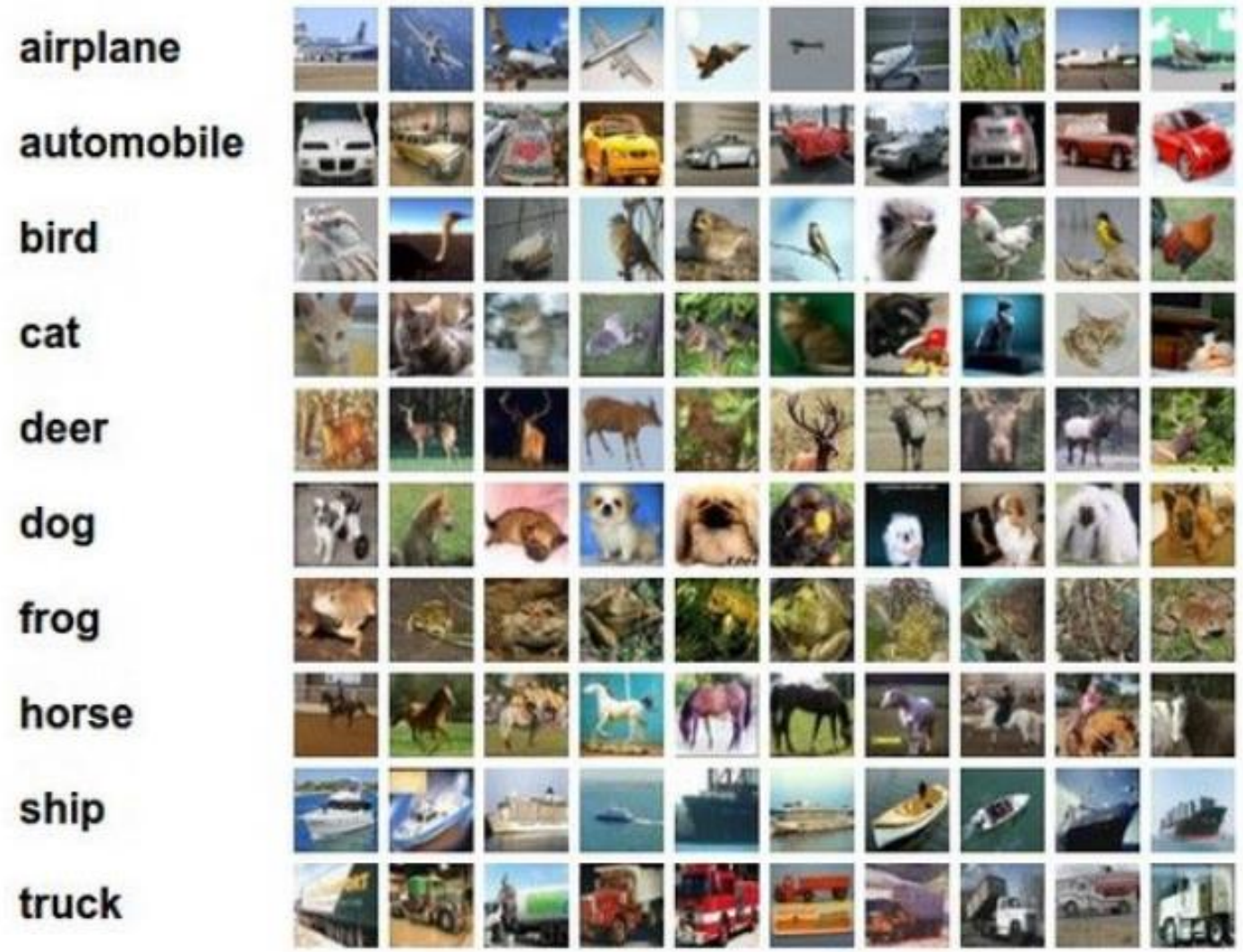
- 数据库简介：

  - 10类标签

    50000个训练数据

    10000个测试数据

    大小均为32*32



图像分类

- 距离的选择： **L1 distance:** $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

- 图像距离计算方式：

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

\-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

\=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

add → 456

图像分类：

- 测试结果：
  - 部分结果还可以的
  - 没有分类对的图像，

    问题出在哪里呢？

图像分类：

- 参数会对结果有影吗？
- 距离的定义： L1 (Manhattan) distance　　L2 (Euclidean) distance

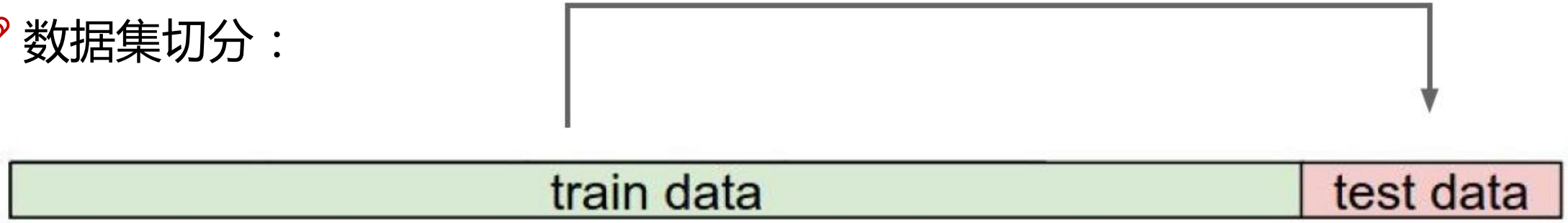
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

- 对于K近邻的K该如何选择？
- 如果有的话，其它的超参数该怎么设定呢？

图像分类：

- 选择最好的参数
  - 交叉验证：实践来检验真理
  - 数据集切分：

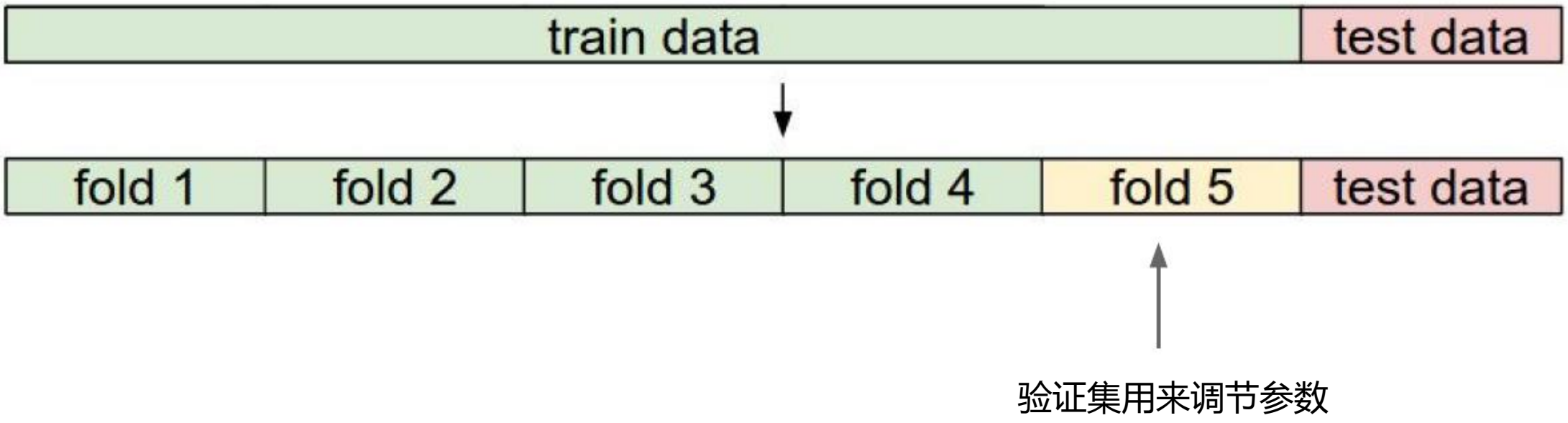


  - 用测试集来选择最好的参数吗？

图像分类：

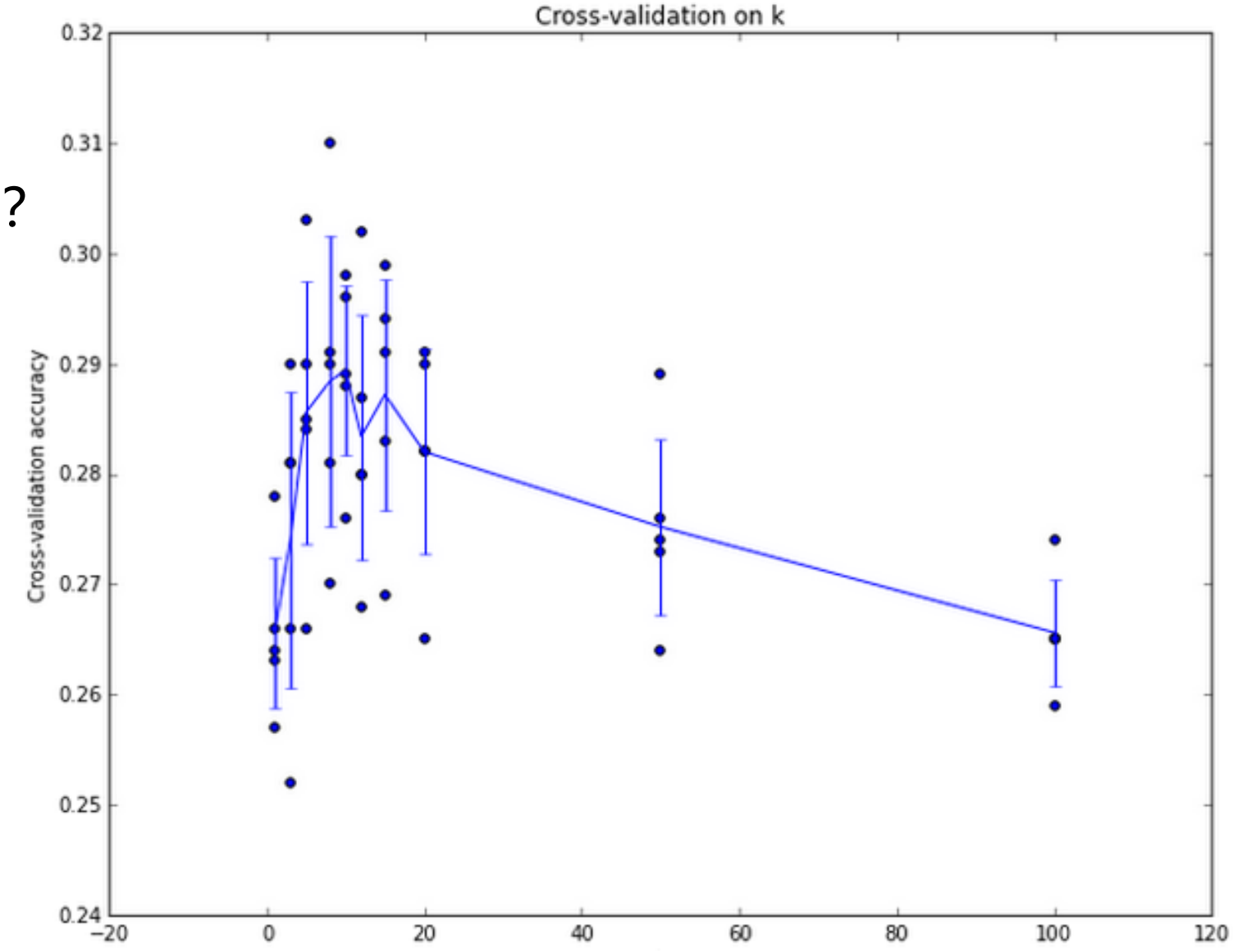
- 选择最好的参数

  - 交叉验证：得到更可靠的验证结果



图像分类：

- K近邻交叉验证结果
  - 参数对结果的影响有多大？
  - 交叉验证是最合适的参数选择方法，比经验值靠谱的多！



图像分类：

- 为什么K近邻不能用来图像分类？
  - 背景主导是一个最大的问题，我们关注的却是主体（主要成分）
  - 如何才能让机器学习到哪些是重要的成分呢？

神经网络基础

- 线性函数

  - 从输入-->输出的映射

(32x32x3)

image parameters

$$f(x, W)$$

→ 每个类别的得分

神经网络基础

- 线性函数

- 数学表示

$$f(x, W) = Wx$$

10x1　　10x3072　　3072x1

(+b)　10x1

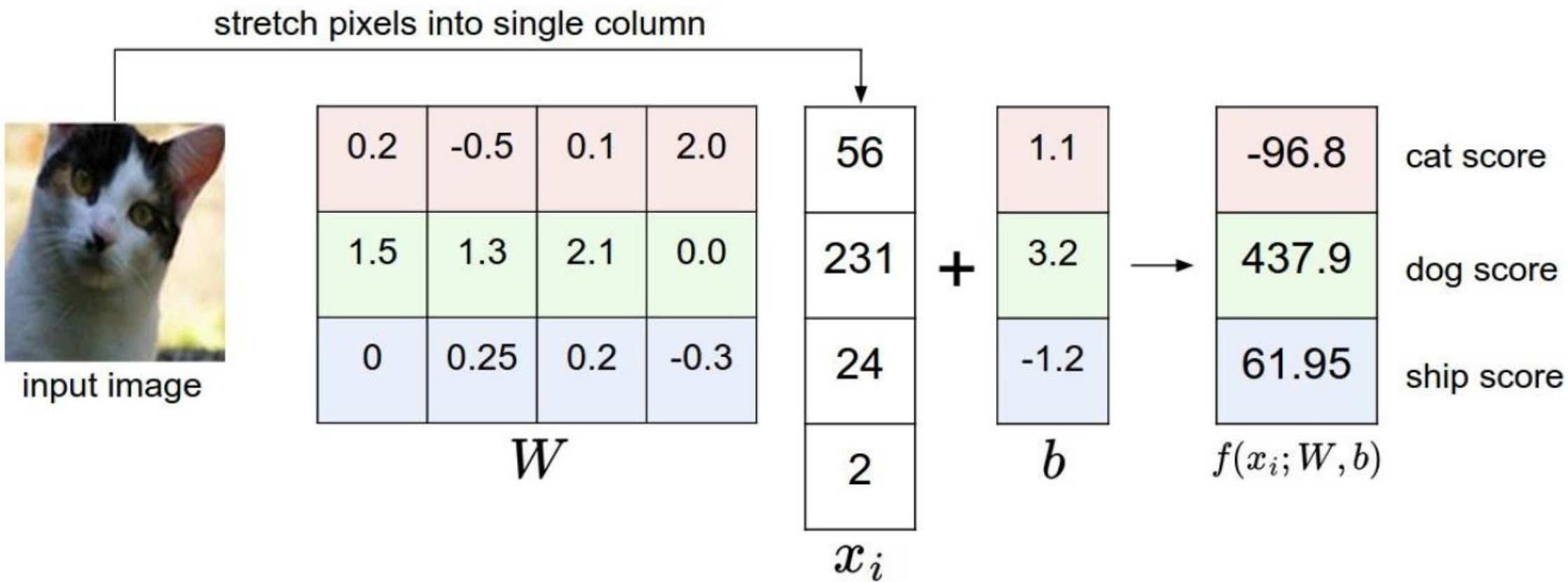
[32x32x3]

→ 每个类别的得分

神经网络基础

- 线性函数

- 计算方法

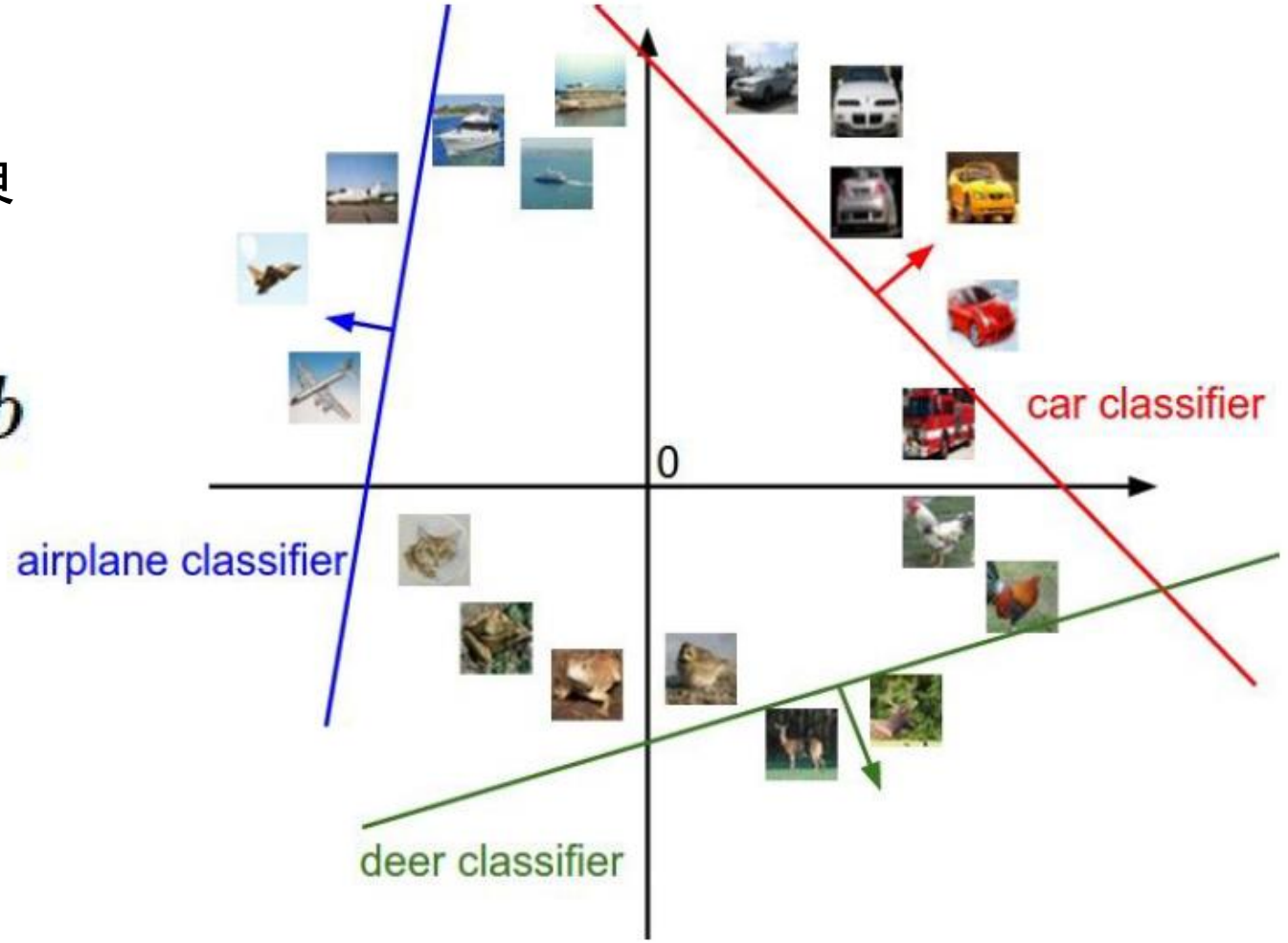


神经网络基础

- 线性函数

  - 多组权重参数构成了决策边界

$$f(x_i, W, b) = Wx_i + b$$



神经网络基础

- 损失函数

  - 如何衡量分类的结果呢？

  - 结果的得分值有着明显的差异，我们需要明确的指导模型的当前效果，有多好或是多差！



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

神经网络基础

- 损失函数 $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

- 损失函数其实有很多种，我们来实验一个



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

神经网络基础

损失函数

如何损失函数的值相同，那么意味着两个模型一样吗？

$$f(x, W) = Wx$$

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

输入数据： $x = [1, 1, 1, 1]$

模型A： $w_1 = [1, 0, 0, 0]$

模型B： $w_2 = [0.25, 0.25, 0.25, 0.25]$

$$w_1^T x = w_2^T x = 1$$

神经网络基础

损失函数

损失函数 = 数据损失 + 正则化惩罚项

正则化惩罚项

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

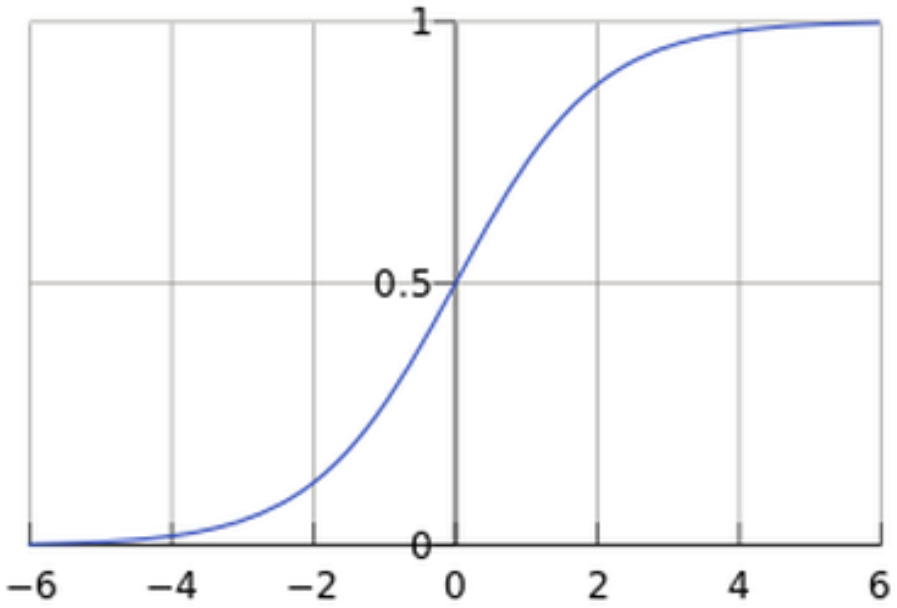
正则化惩罚项：$R(W) = \sum_k \sum_l W_{k,l}^2$

（我们总是希望模型不要太复杂，过拟合的模型是没用的）

神经网络基础

Softmax分类器

- 现在我们得到的是一个输入的得分值，但如果给我一个概率值岂不更好！
- 如何把一个得分值转换成一个概率值呢？



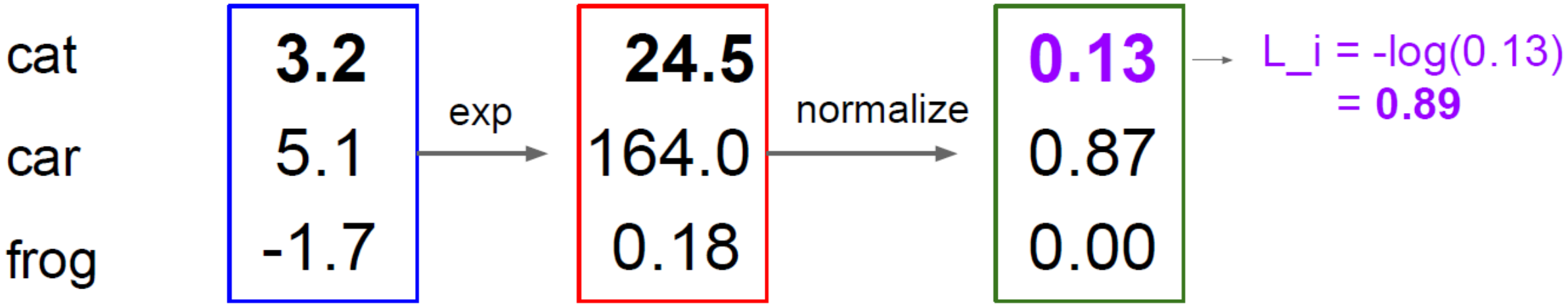
$$g(z)=\frac{1}{1+e^{-z}}$$

神经网络基础

- Softmax分类器

归一化：$P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}$ where $s=f(x_i;W)$

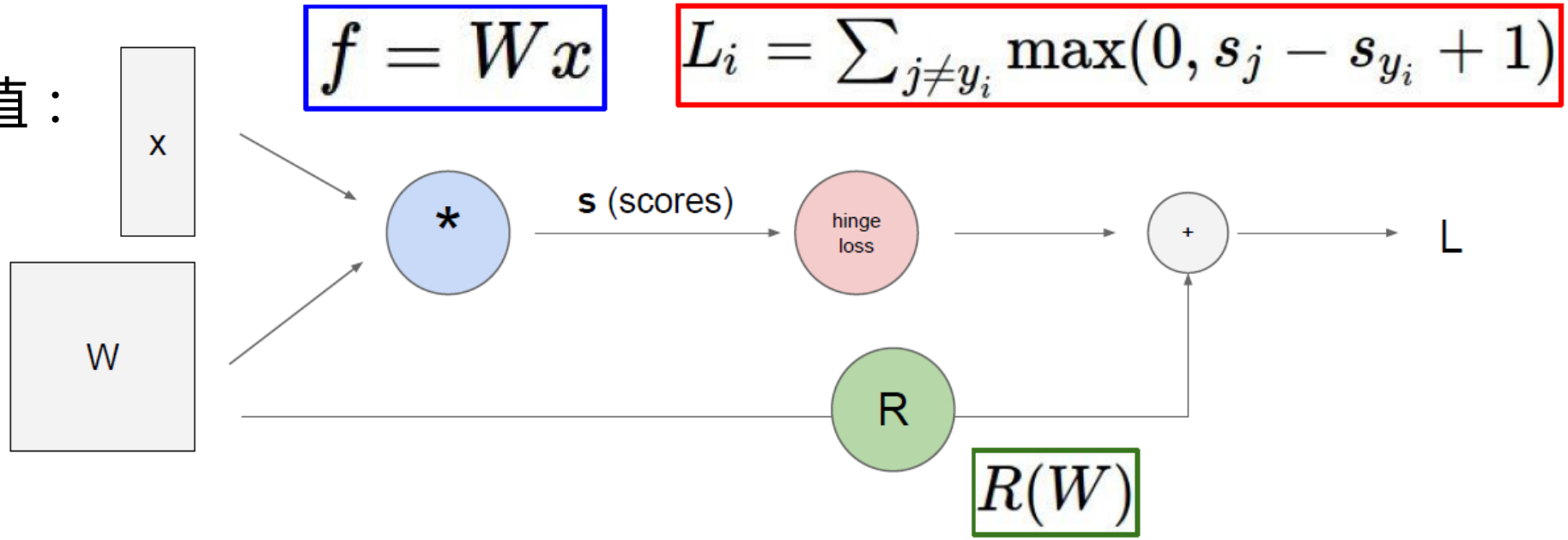
计算损失值：$L_i=-\log P(Y=y_i|X=x_i)$



神经网络基础

- 前向传播

得出损失值：



如何更新模型呢？这个就交给反向传播了（梯度下降）

线性回归

- 梯度下降

  - 引入：当我们得到了一个目标函数后，如何进行求解？
    直接求解？（并不一定可解，线性回归可以当做是一个特例）

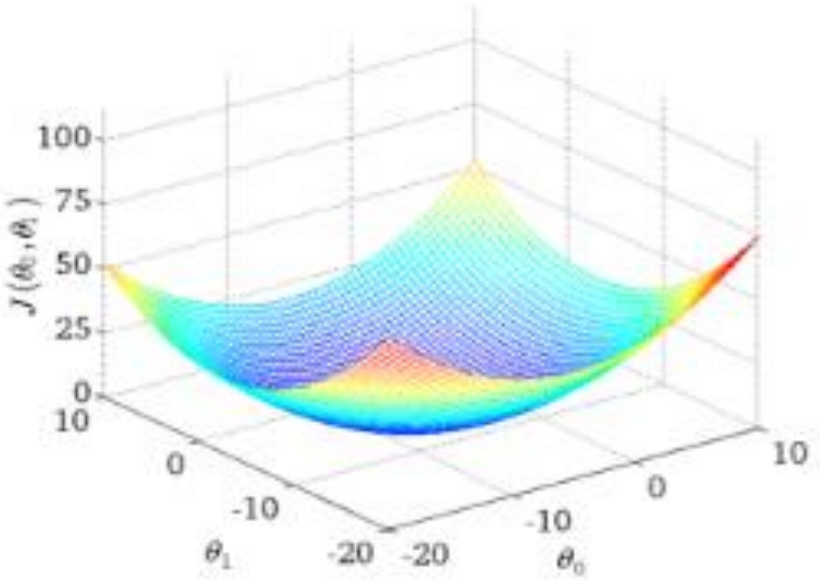
  - 常规套路：机器学习的套路就是我交给机器一堆数据，然后告诉它
    什么样的学习方式是对的（目标函数），然后让它朝着这个方向去做

  - 如何优化：一口吃不成个胖子，我们要静悄悄的一步步的完成迭代
    （每次优化一点点，累积起来就是个大成绩了）

线性回归

- 梯度下降

目标函数： $J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})-y^{(i)}\right)$



寻找山谷的最低点，也就是我们的目标函数终点
（什么样的参数能使得目标函数达到极值点）

下山分几步走呢？（更新参数）
（1）：找到当前最合适的方向
（2）：走那么一小步，走快了该”跌倒 “ 了
（3）：按照方向与步伐去更新我们的参数

线性回归

- 梯度下降，目标函数： $J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(y^i-h_\theta(x^i))^2$

  - 批量梯度下降： $\frac{\partial J(\theta)}{\partial \theta_j}=-\frac{1}{m}\sum_{i=1}^{m}(y^i-h_\theta(x^i))x_j^i \quad \theta_j^{'}=\theta_j+\frac{1}{m}\sum_{i=1}^{m}(y^i-h_\theta(x^i))x_j^i$

    （容易得到最优解，但是由于每次考虑所有样本，速度很慢）

  - 随机梯度下降： $\theta_j^{'}=\theta_j+(y^i-h_\theta(x^i))x_j^i$

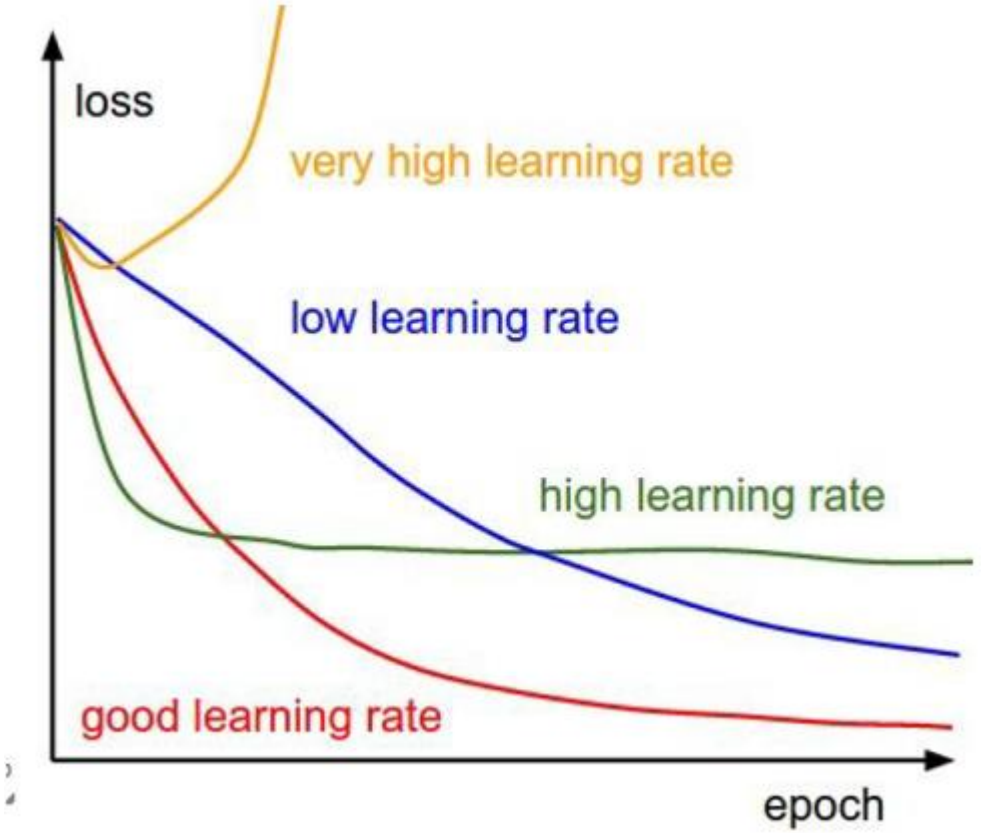
    （每次找一个样本，迭代速度快，但不一定每次都朝着收敛的方向）

  - 小批量梯度下降法： $\theta_j:=\theta_j-\alpha\frac{1}{10}\sum_{k=i}^{i+9}(h_\theta(x^{(k)})-y^{(k)})x_j^{(k)}$

    （每次更新选择一小部分数据来算，实用！）

线性回归

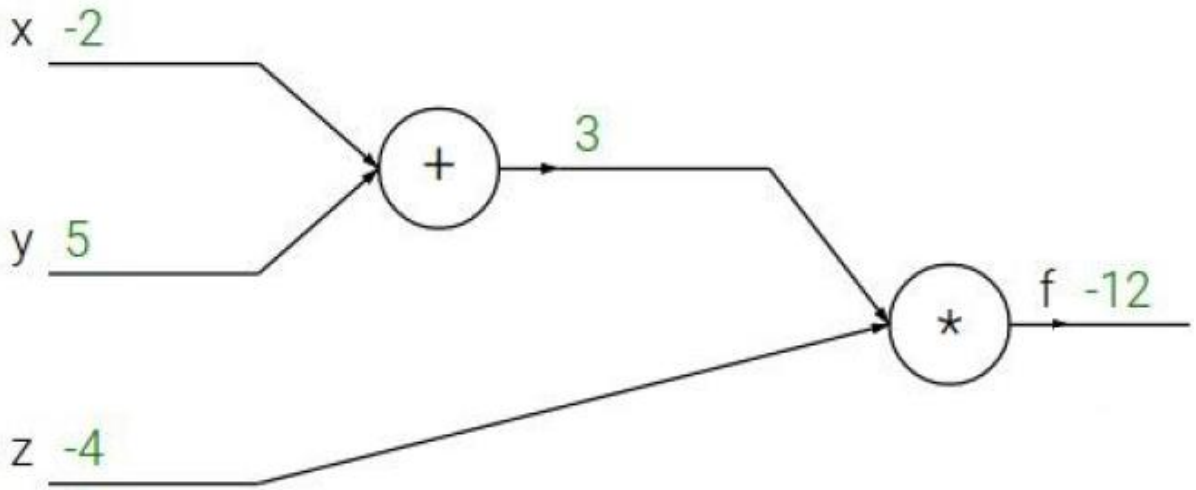
- 梯度下降
  - 学习率（步长）：对结果会产生巨大的影响，一般小一些
  - 如何选择：从小的时候，不行再小
  - 批处理数量：32，64，128都可以，很多时候还得考虑内存和效率



神经网络基础

- 反向传播

- 简单的栗子：



$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

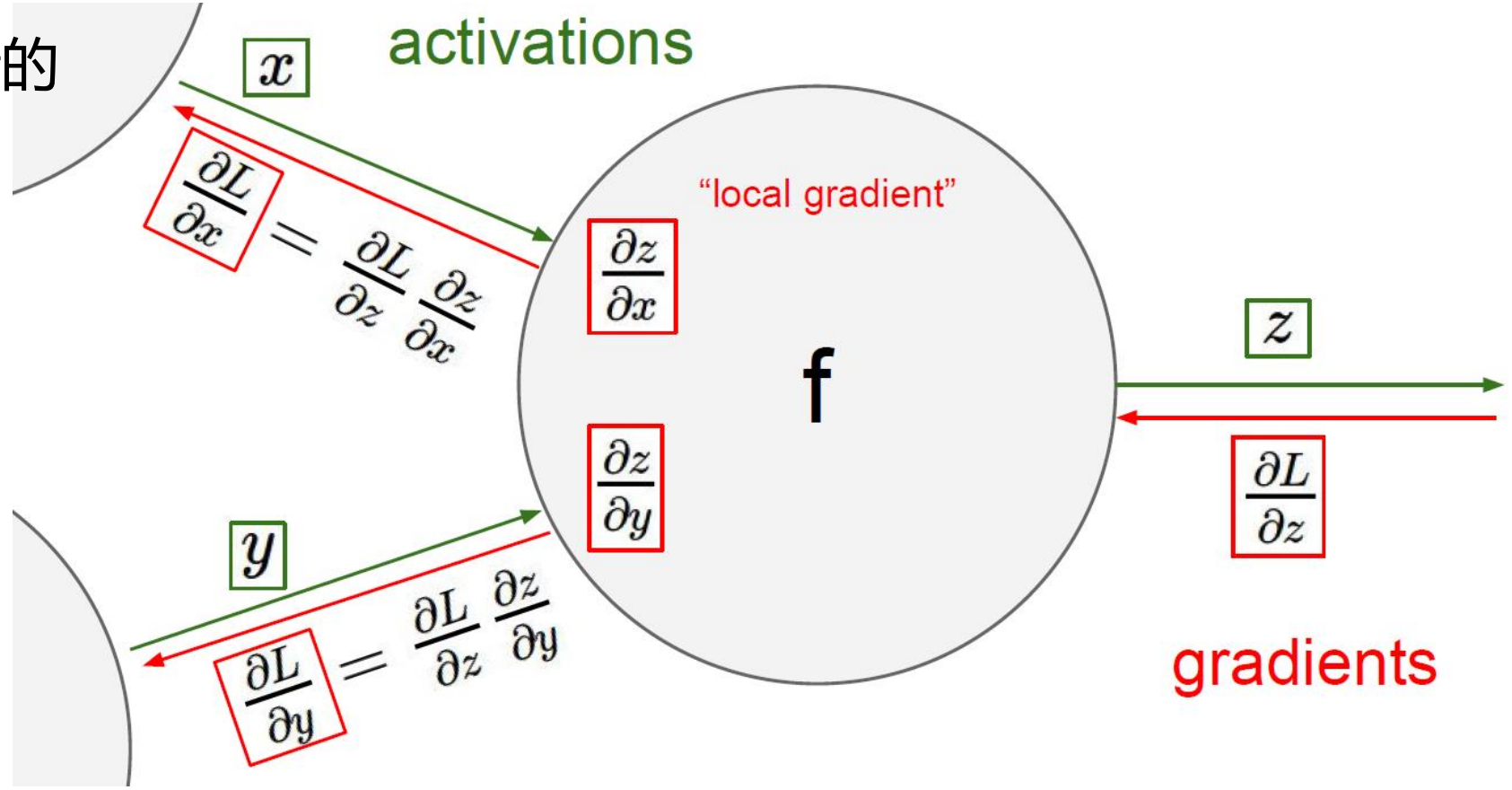
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

神经网络基础

- 链式法则

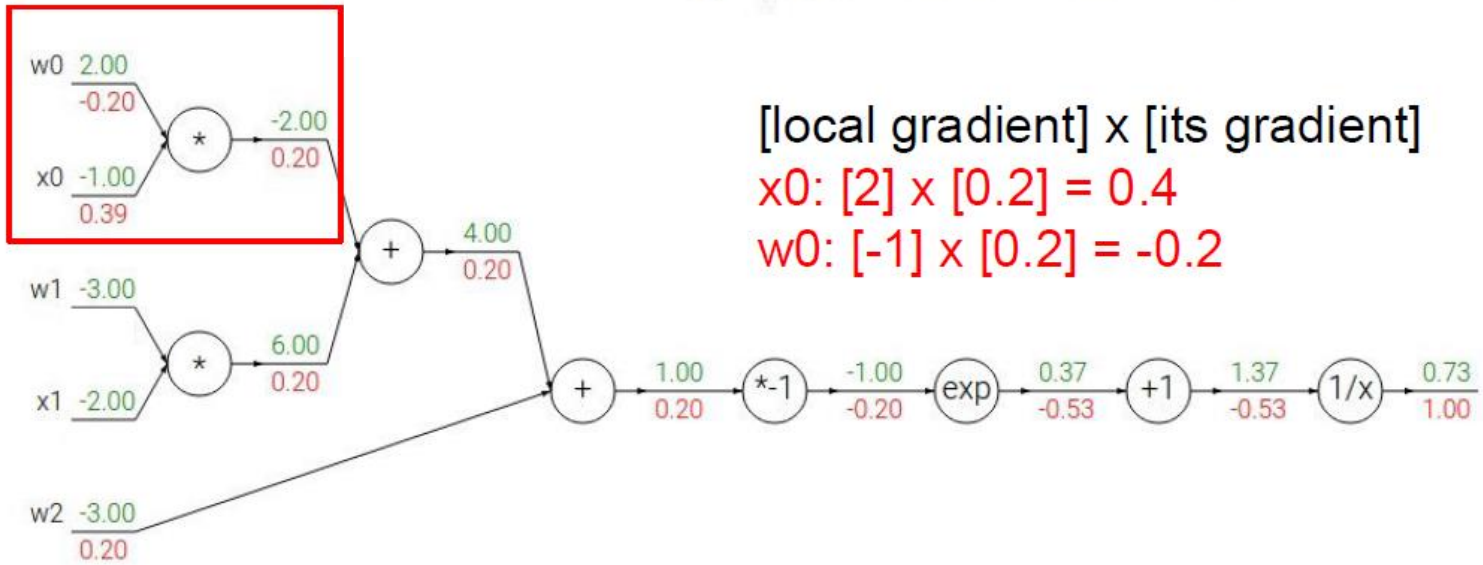
  - 梯度是一步一步传的



神经网络基础

- 反向传播

复杂的栗子： $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]
x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x \qquad f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a \qquad f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

神经网络基础

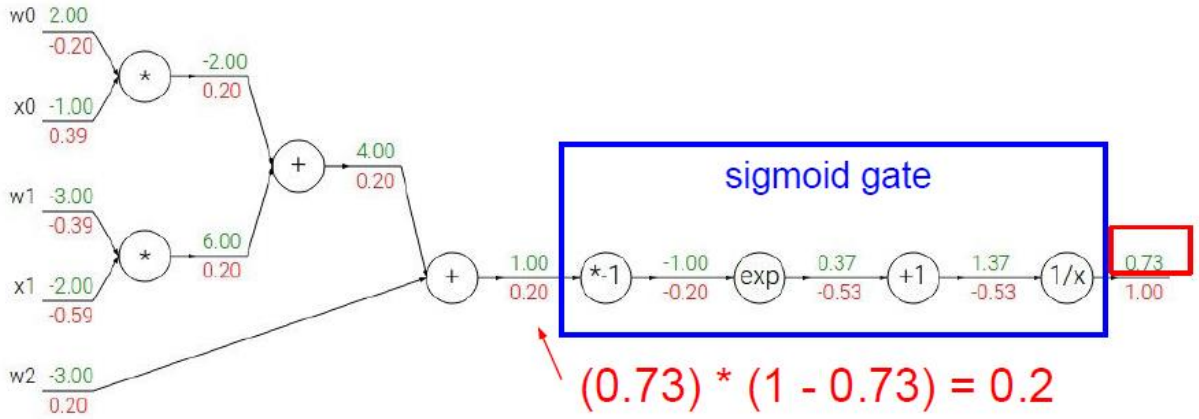
反向传播

可以一大块一大块的计算吗？

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\sigma(x)=\frac{1}{1+e^{-x}}$$ sigmoid function

$$\frac{d\sigma(x)}{dx}=\frac{e^{-x}}{(1+e^{-x})^2}=\left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)=(1-\sigma(x))\,\sigma(x)$$



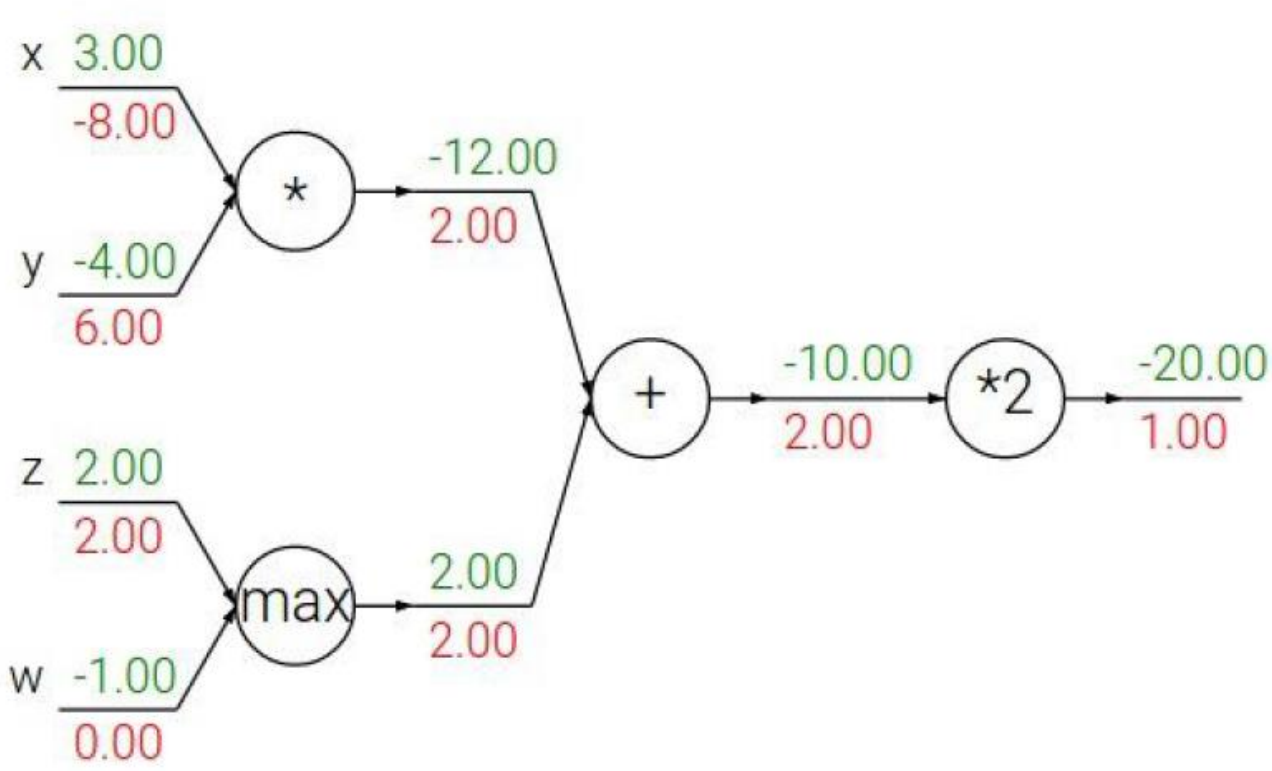
神经网络基础

- 反向传播

  - 加法门单元：均等分配

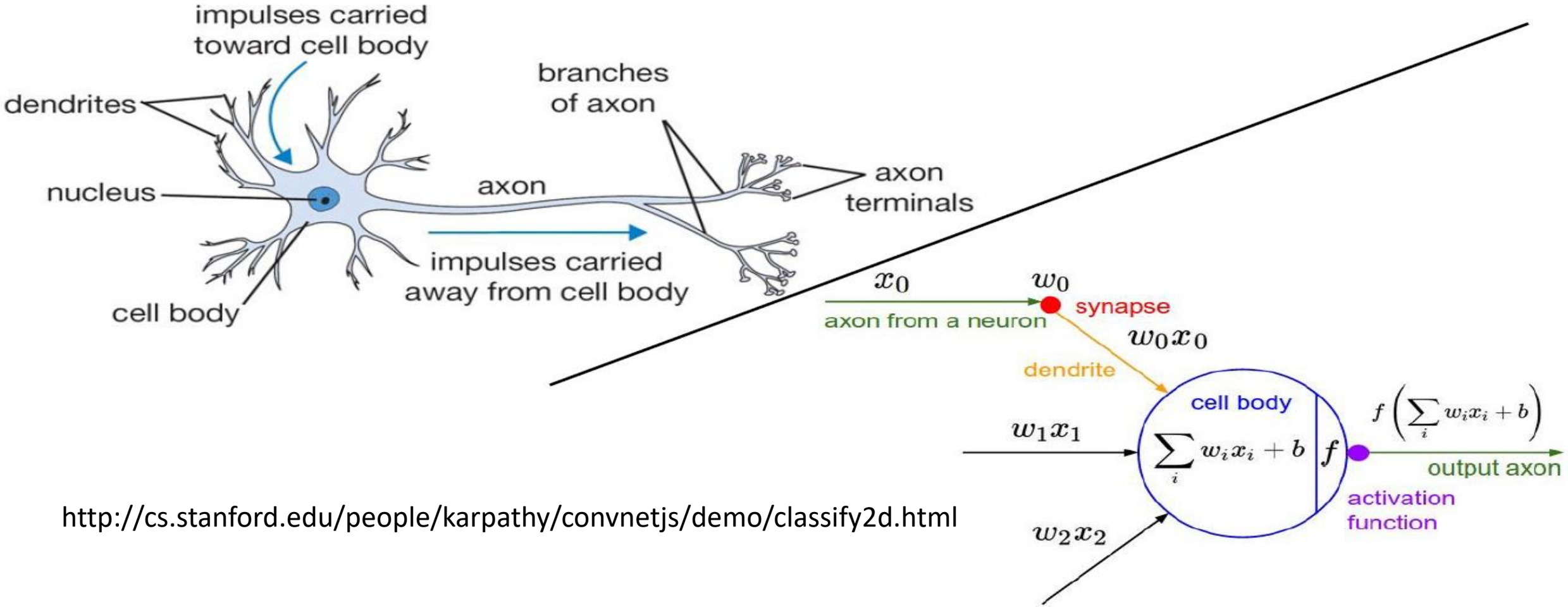
    MAX门单元：给最大的

    乘法门单元：互换的感觉



神经网络

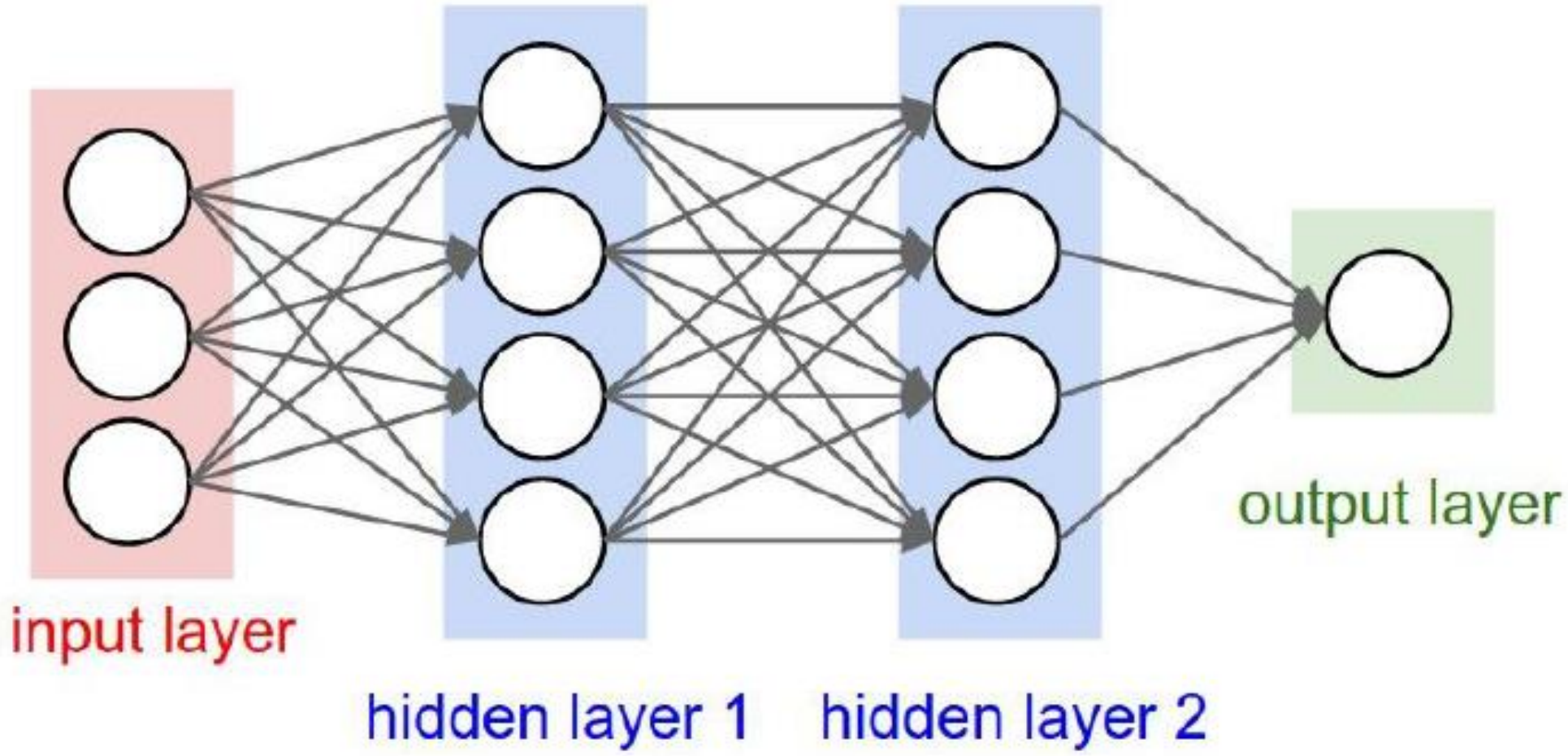
- 整体架构



http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

神经网络

- 整体架构
- 层次结构
- 神经元
- 全连接
- 非线性



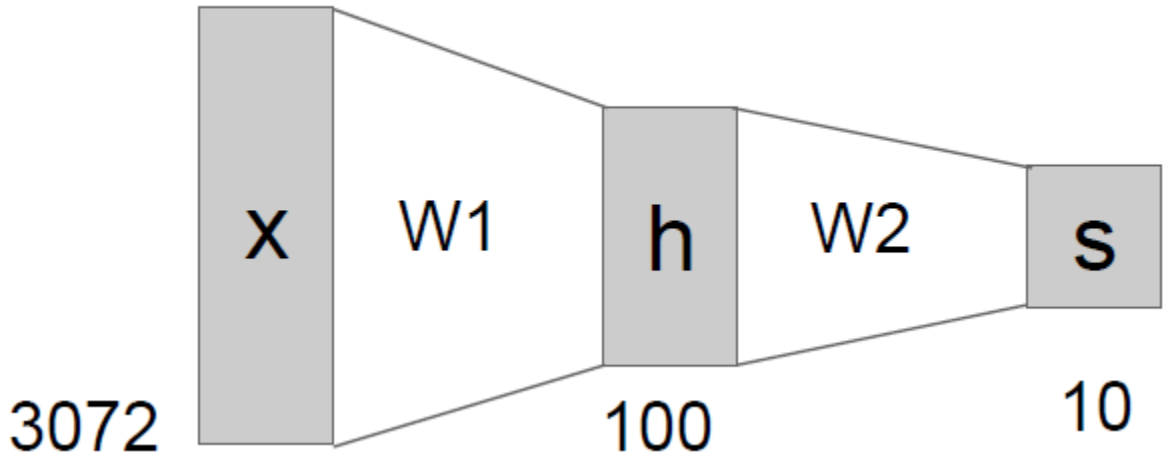
神经网络

- 整体架构

- 线性方程： $f = Wx$

- 非线性方程： $f = W_2 \max(0, W_1 x)$

- 计算结果：



神经网络

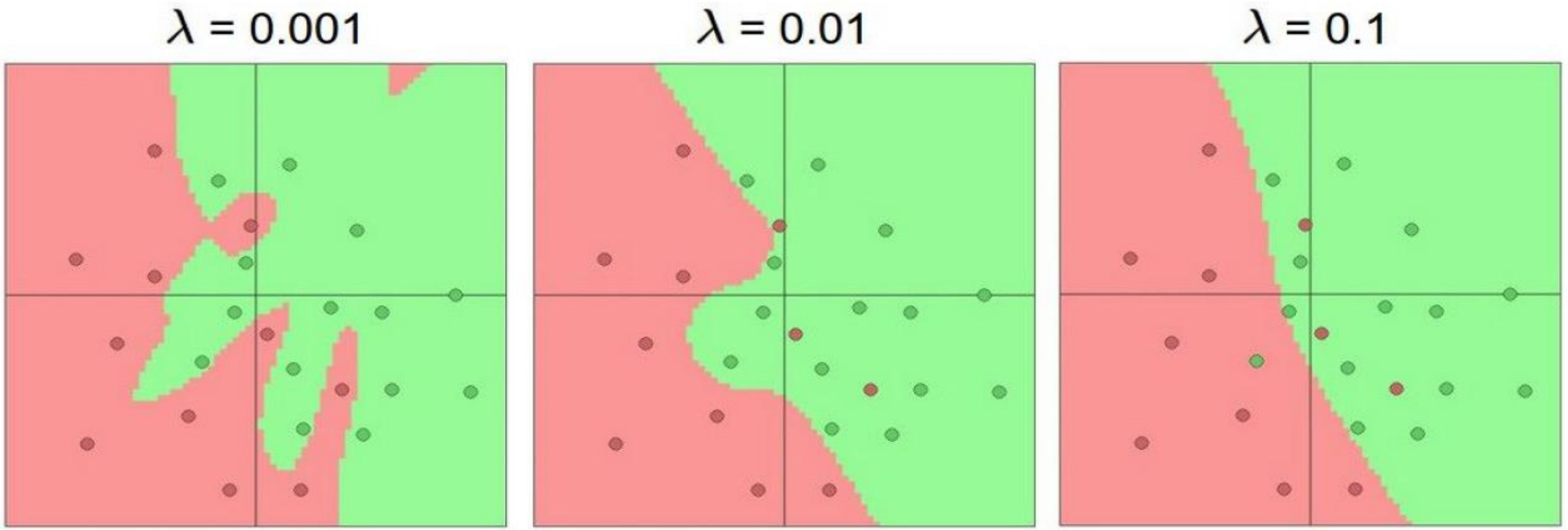
- 整体架构
  - 基本机构：$f = W_2 \max(0, W_1 x)$
  - 继续堆叠一层：$f = W_3 \max(0, W_2 \max(0, W_1 x))$
  - 神经网络的强大之处在于，用更多的参数来拟合复杂的数据

    （参数多到多少呢？百万级别都是小儿科了，但是参数越多越好吗？）

神经网络

- 正则化的作用

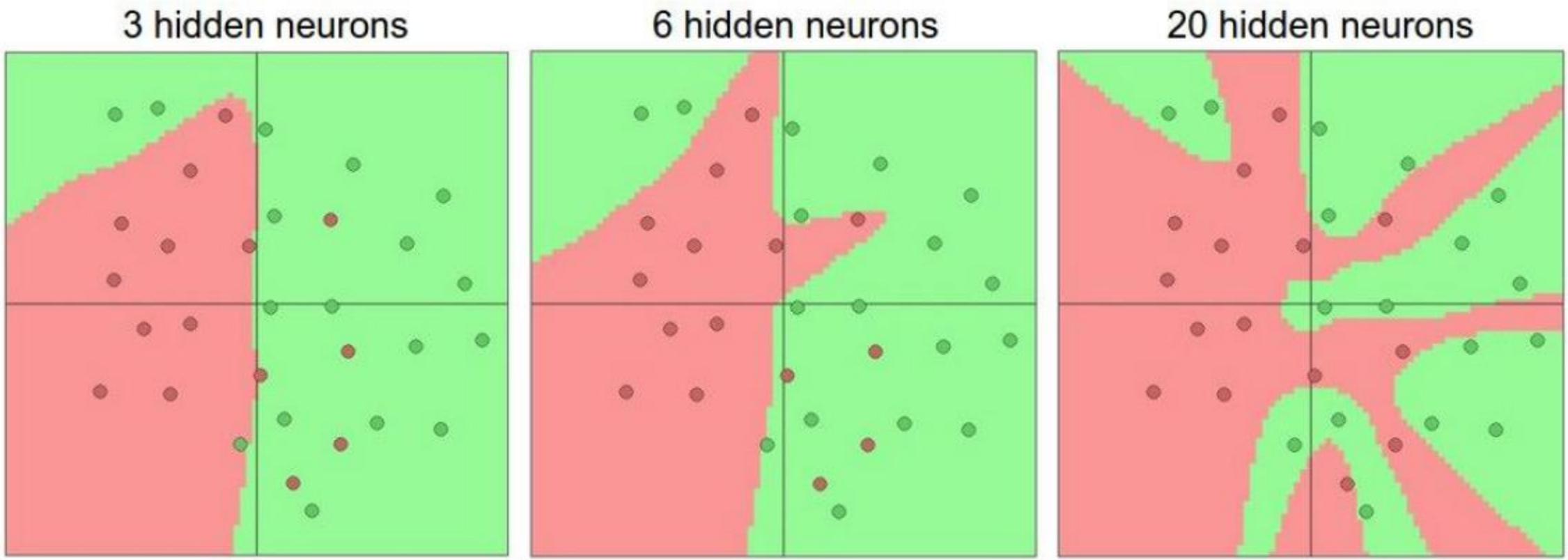
  - 惩罚力度对结果的影响：



神经网络

- 神经元

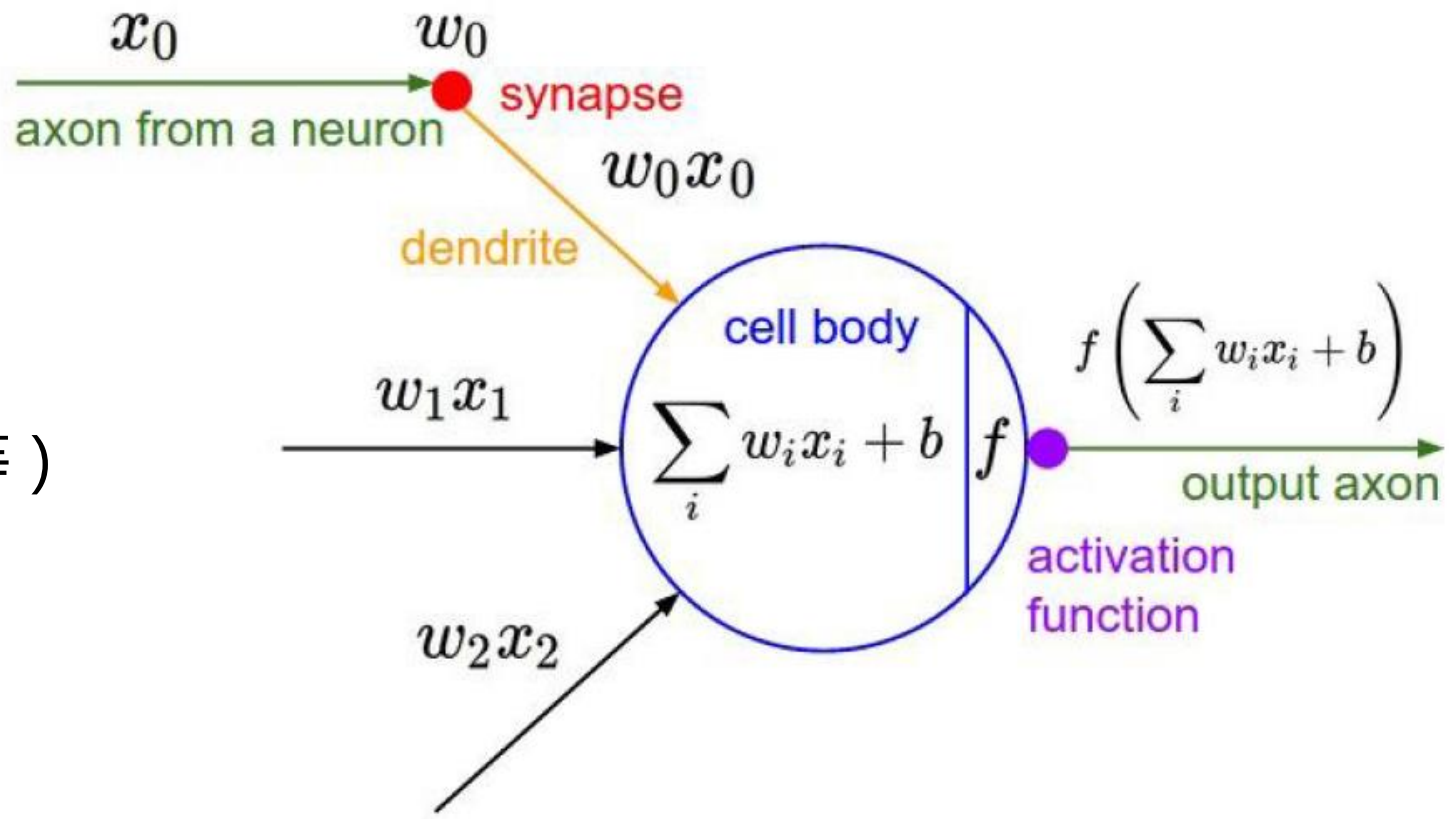
- 参数个数对结果的影响：



神经网络

- 激活函数
  - 非常重要的一部分
  - 常用的激活函数

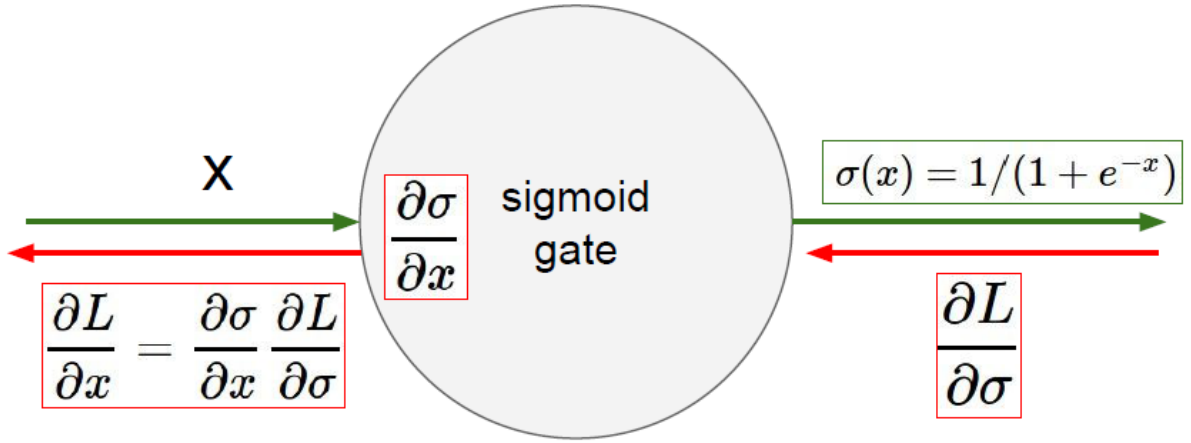
    （Sigmoid,Relu,Tanh 等）

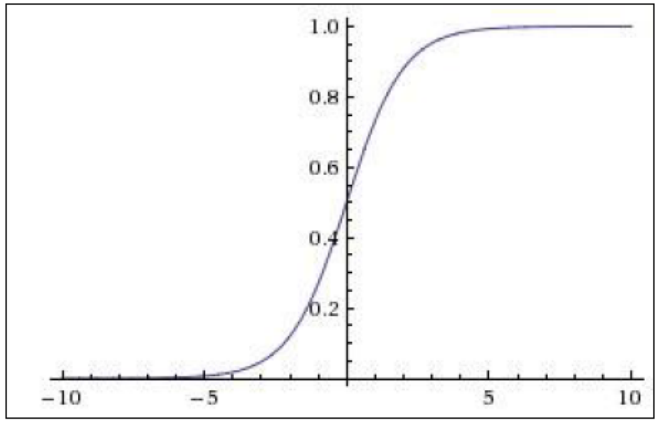


神经网络

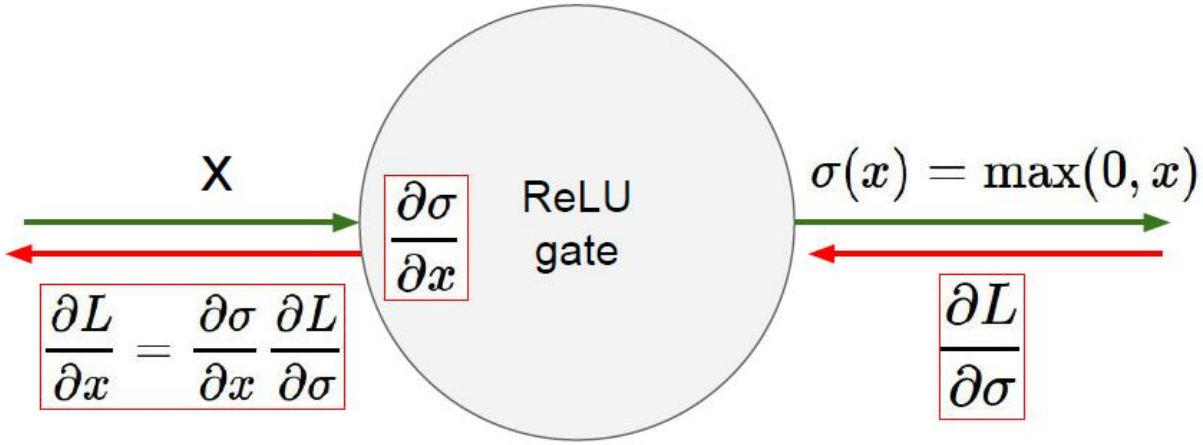
- 激活函数对比

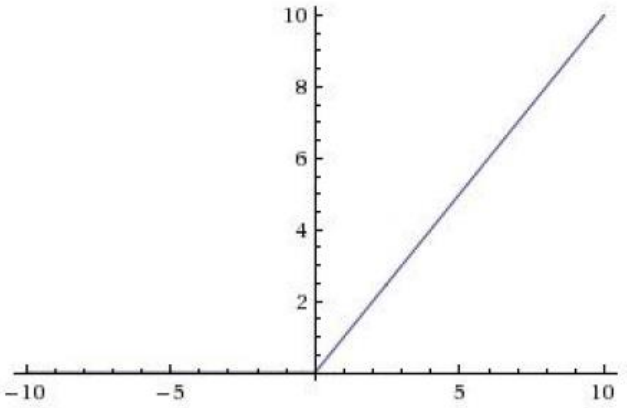
- Sigmoid:





- Relu:

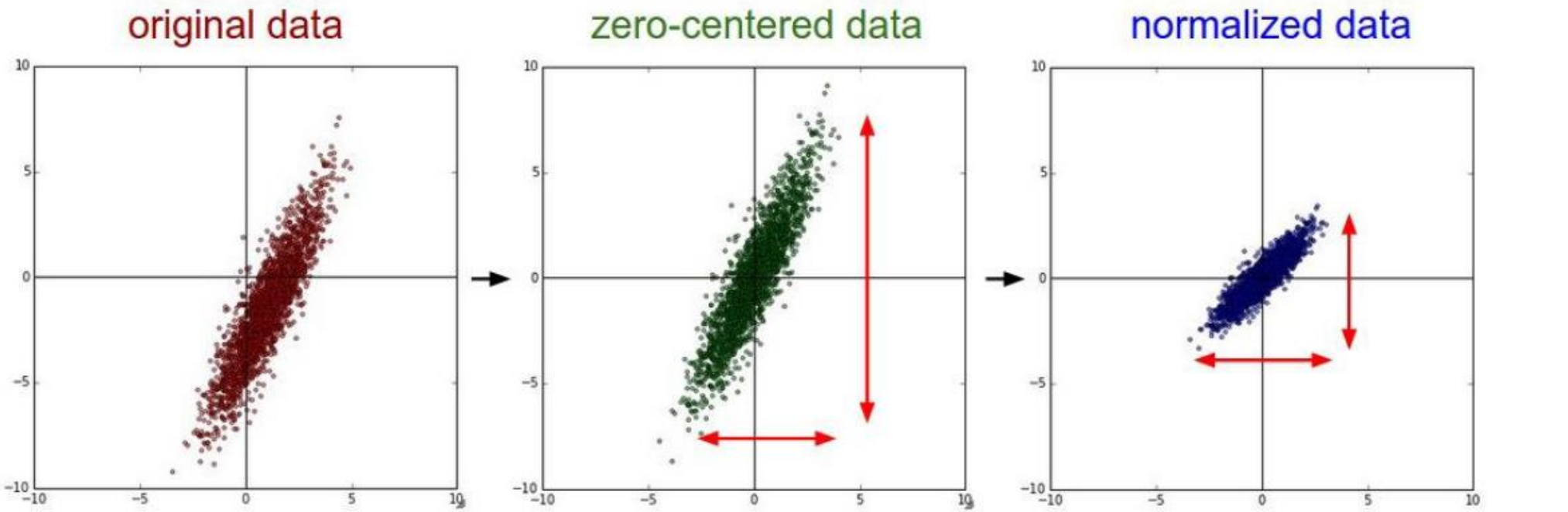




神经网络

- 数据预处理

- 不同的预处理结果会使得模型的效果发生很大的差异！



神经网络

- 参数初始化

  - 参数初始化同样非常重要！

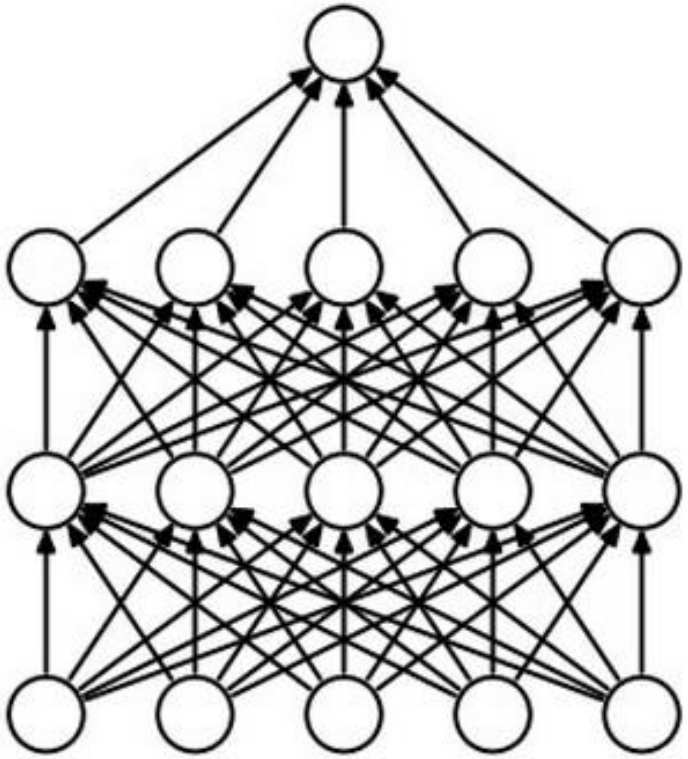
  - 通常我们都使用随机策略来进行参数初始化

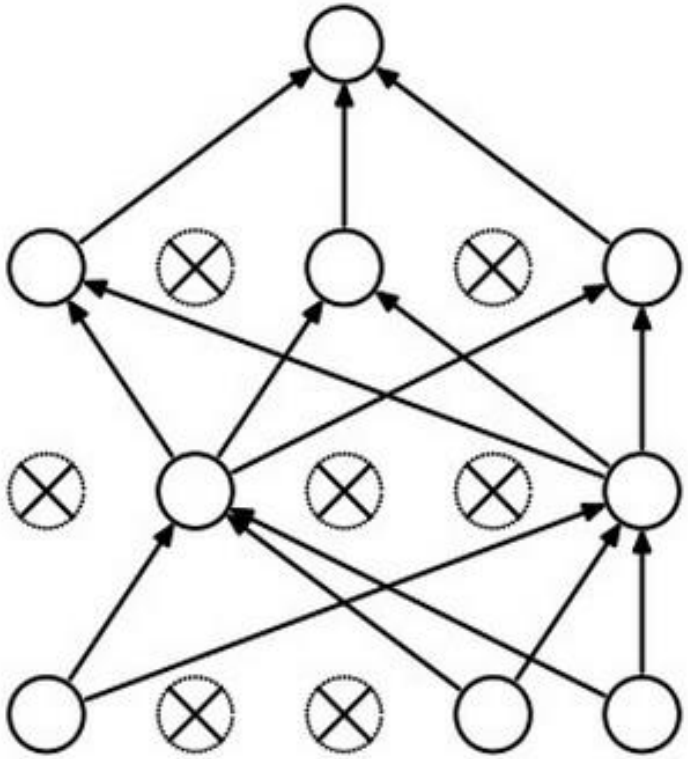
```
W = 0.01* np.random.randn(D,H)
```

神经网络

- DROP-OUT（传说中的七伤拳）

  - 过拟合是神经网络非常头疼的一个大问题！

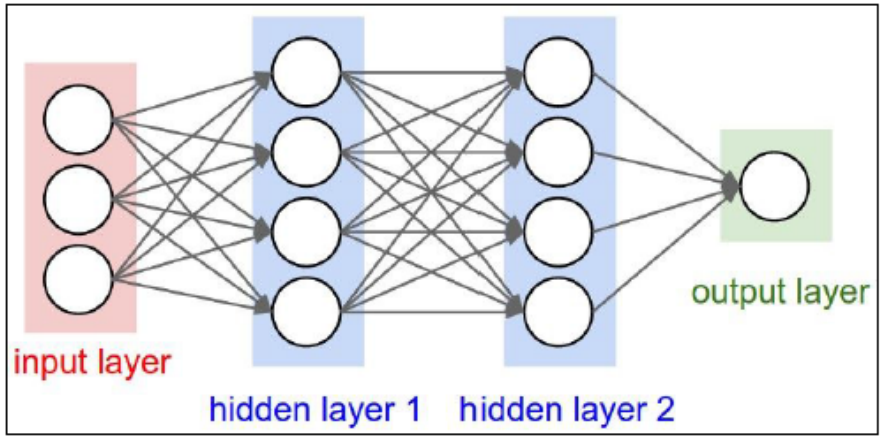




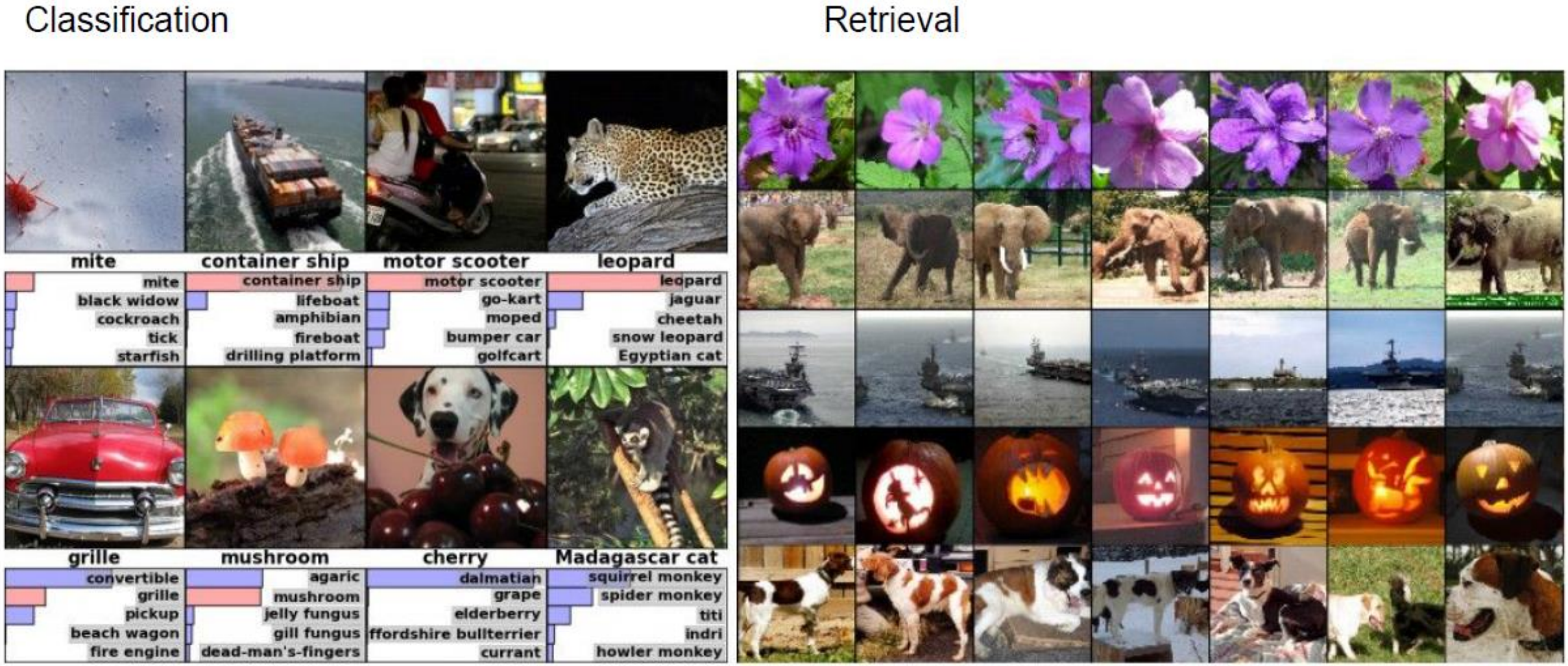
卷积神经网络：

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

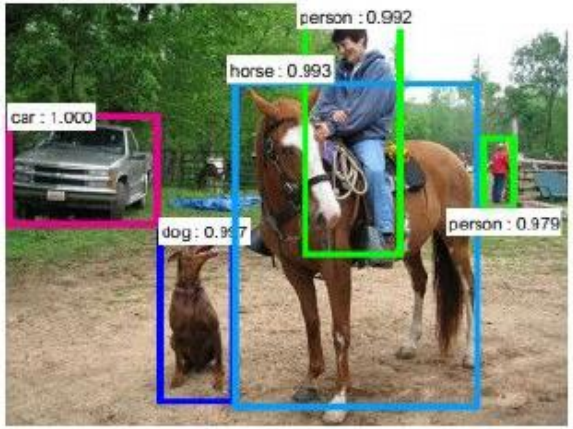


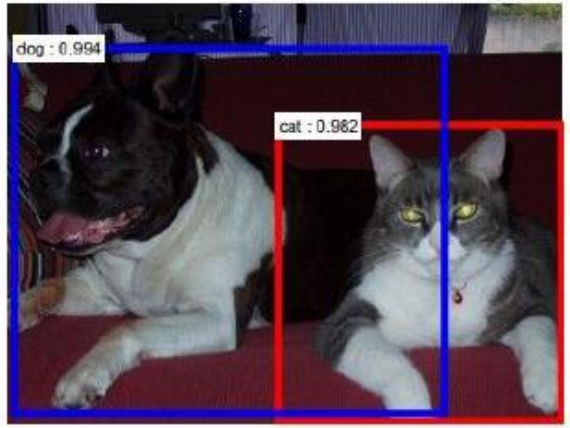
卷积神经网络：

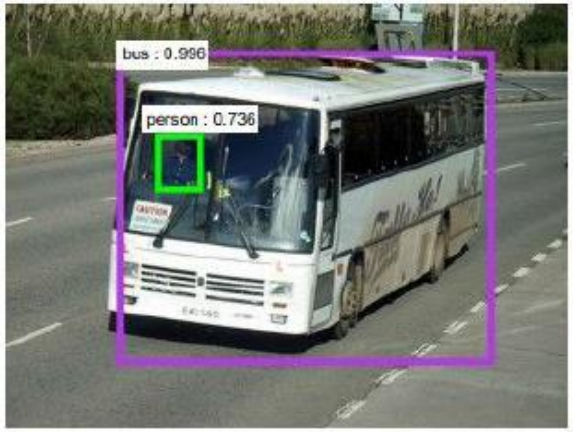

Classification
Retrieval
mite
container ship
motor scooter
leopard
mite
black widow
cockroach
tick
starfish
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
go-kart
moped
bumper car
golfcart
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
mushroom
cherry
Madagascar cat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

卷积神经网络：

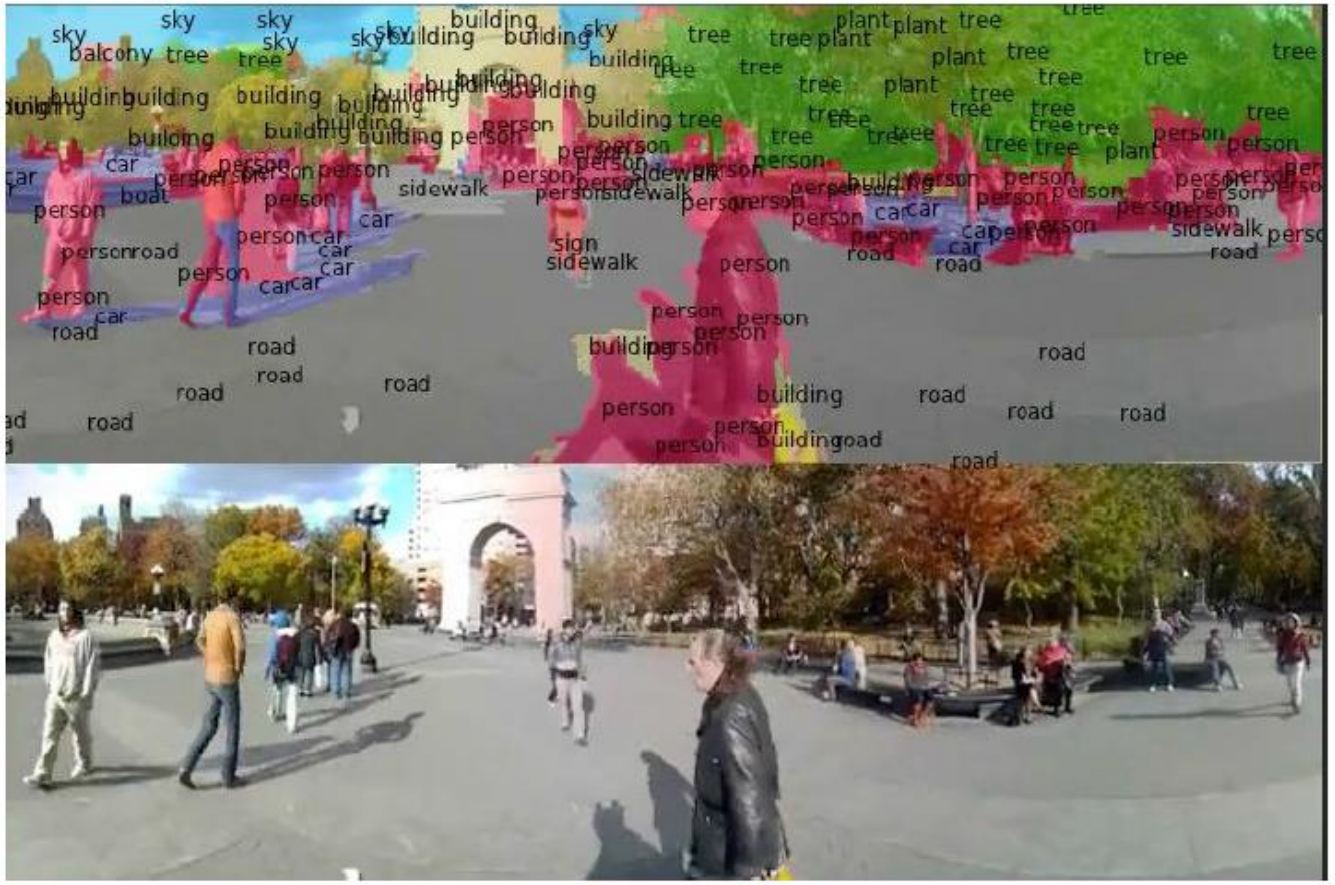
Detection







Segmentation

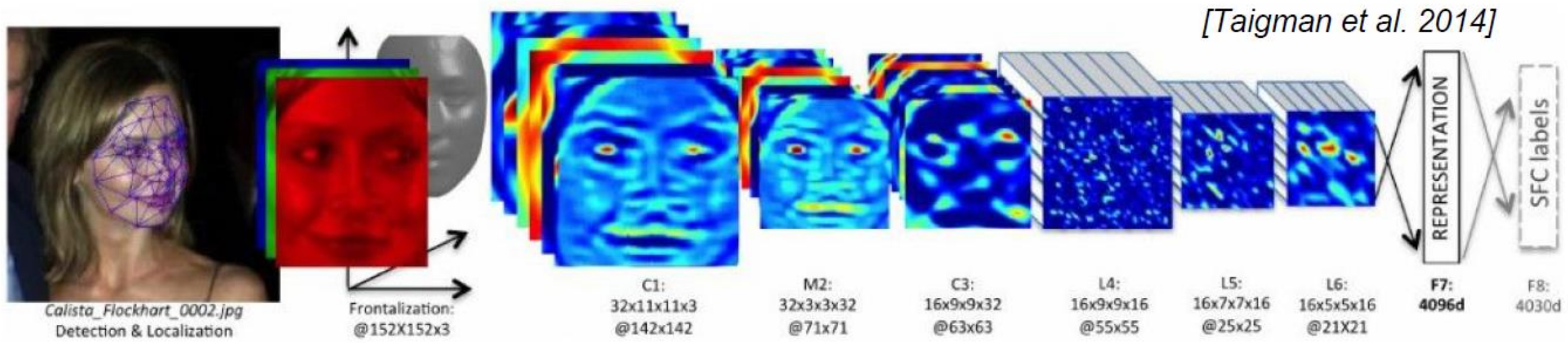


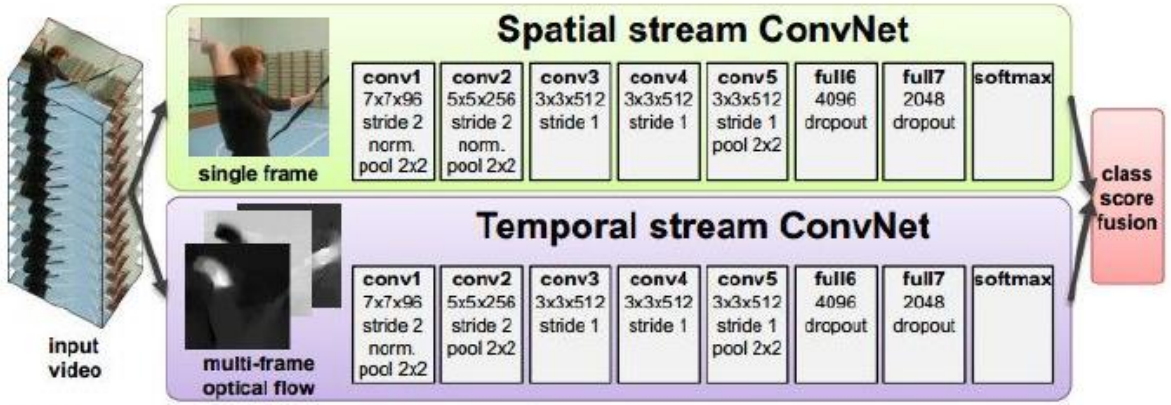
卷积神经网络：

self-driving cars

NVIDIA Tegra X1

卷积神经网络：





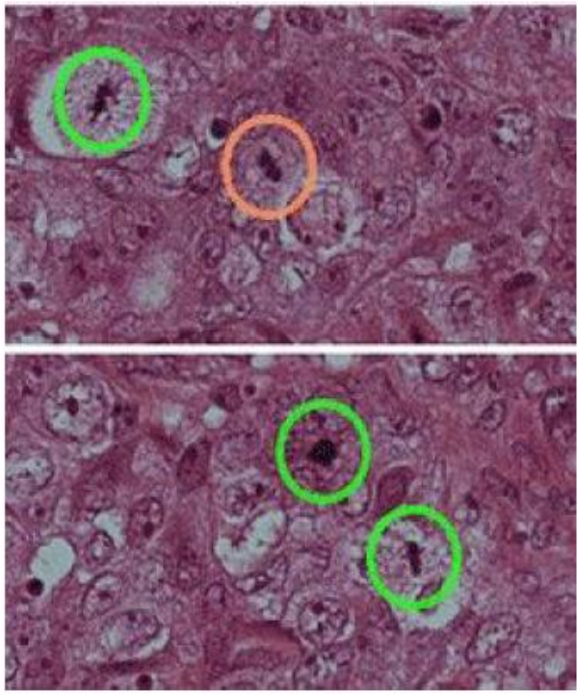
[Simonyan et al. 2014]

[Goodfellow 2014]

卷积神经网络：

[Toshev, Szegedy 2014]

卷积神经网络：

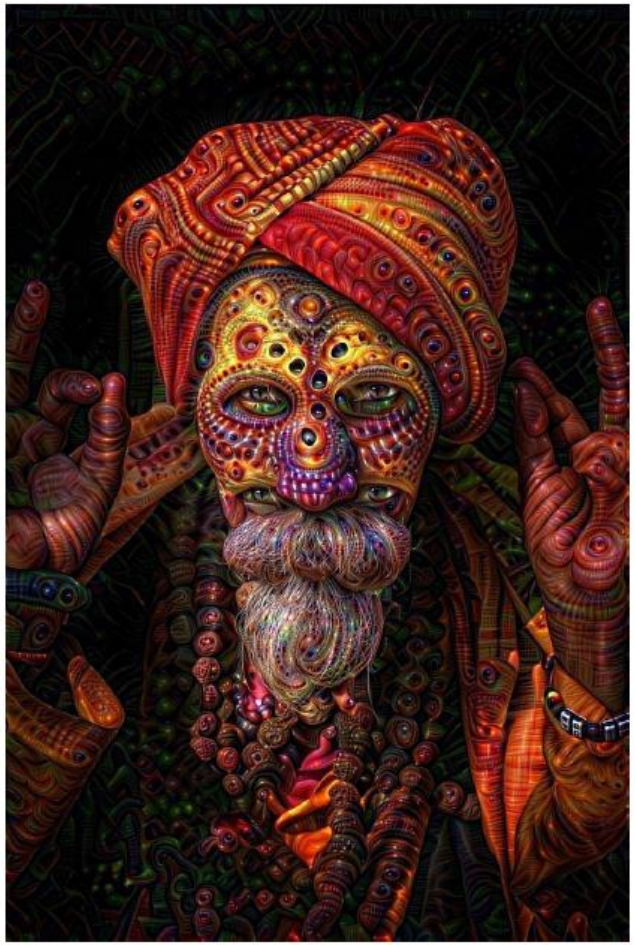


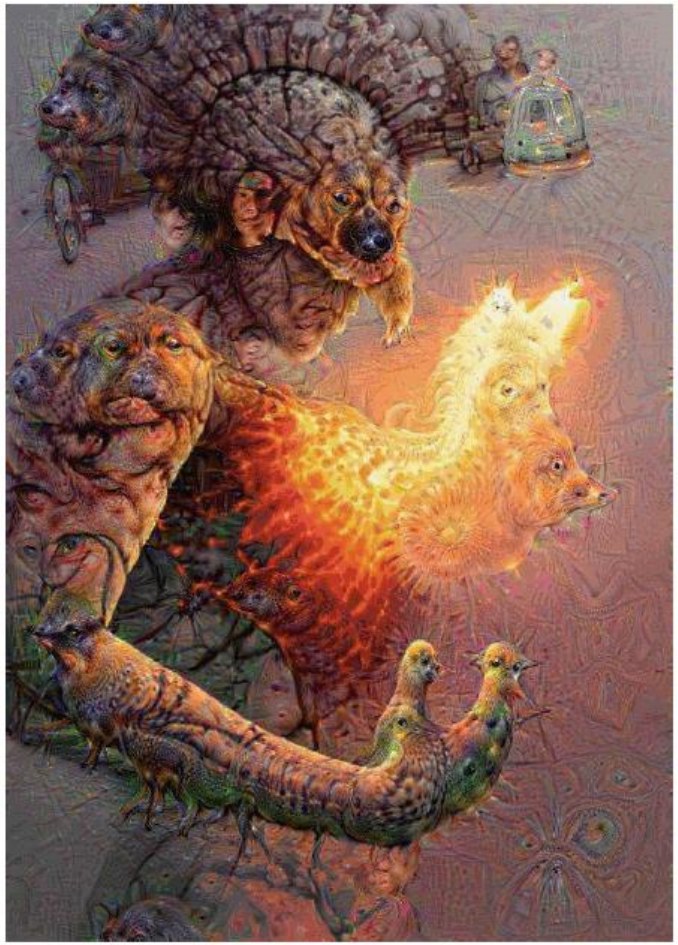
卷积神经网络：

Image Captioning

[Vinyals et al., 2015]

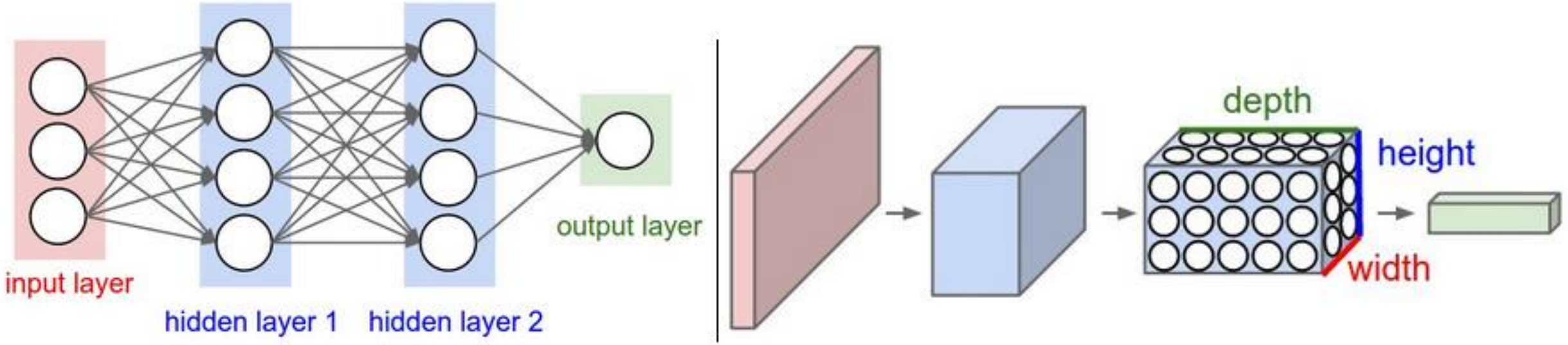
卷积神经网络：





卷积神经网络：

卷积神经网络：

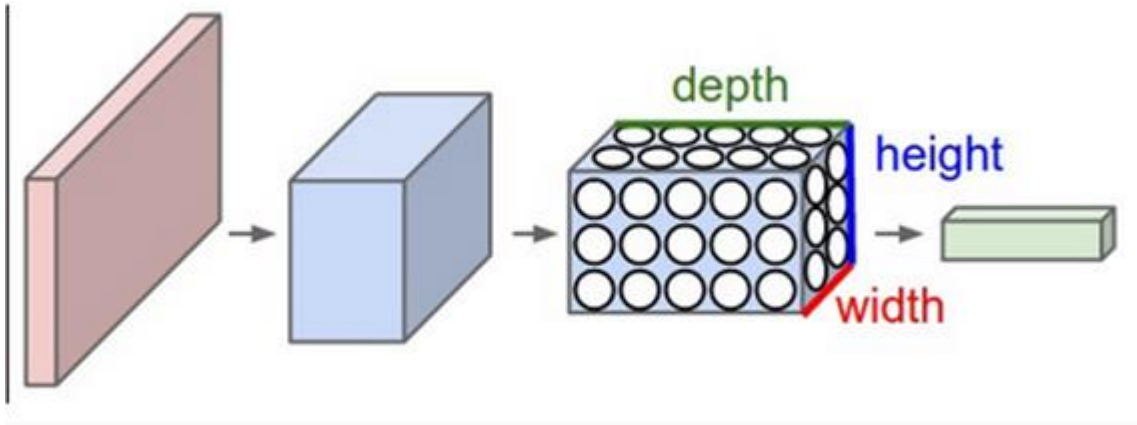

input layer
hidden layer 1
hidden layer 2
output layer
depth
height
width

卷积神经网络：

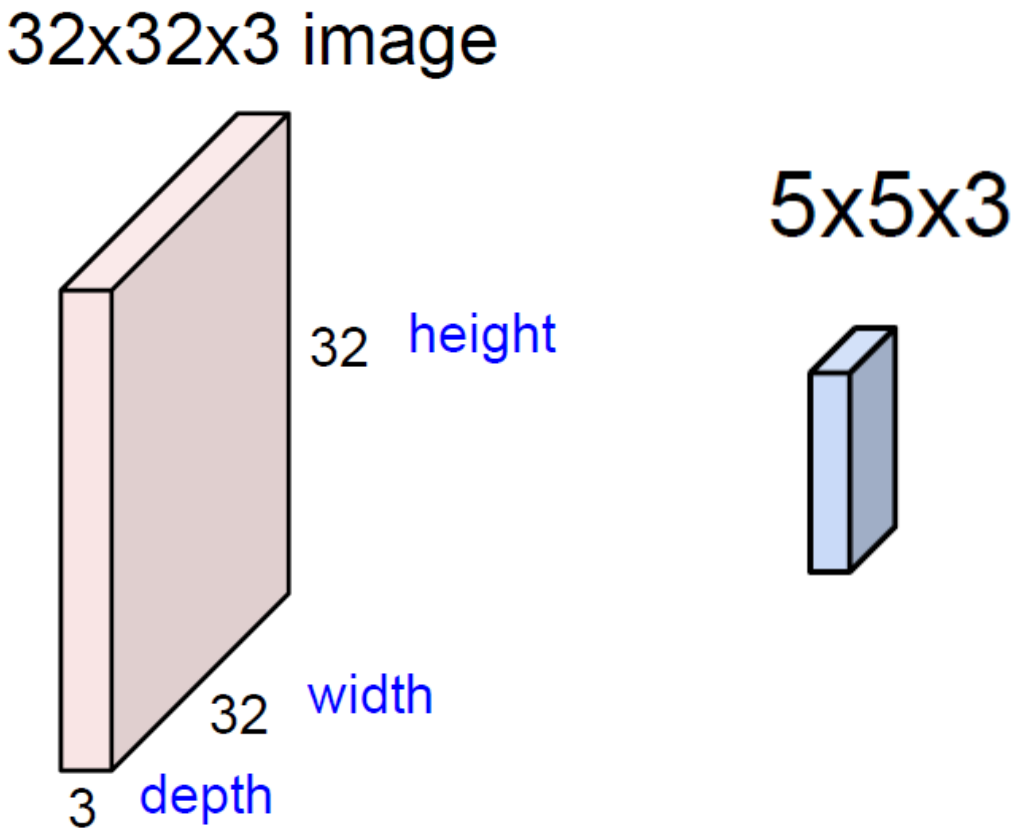
卷积神经网络组成：

[INPUT - CONV - RELU - POOL - FC]

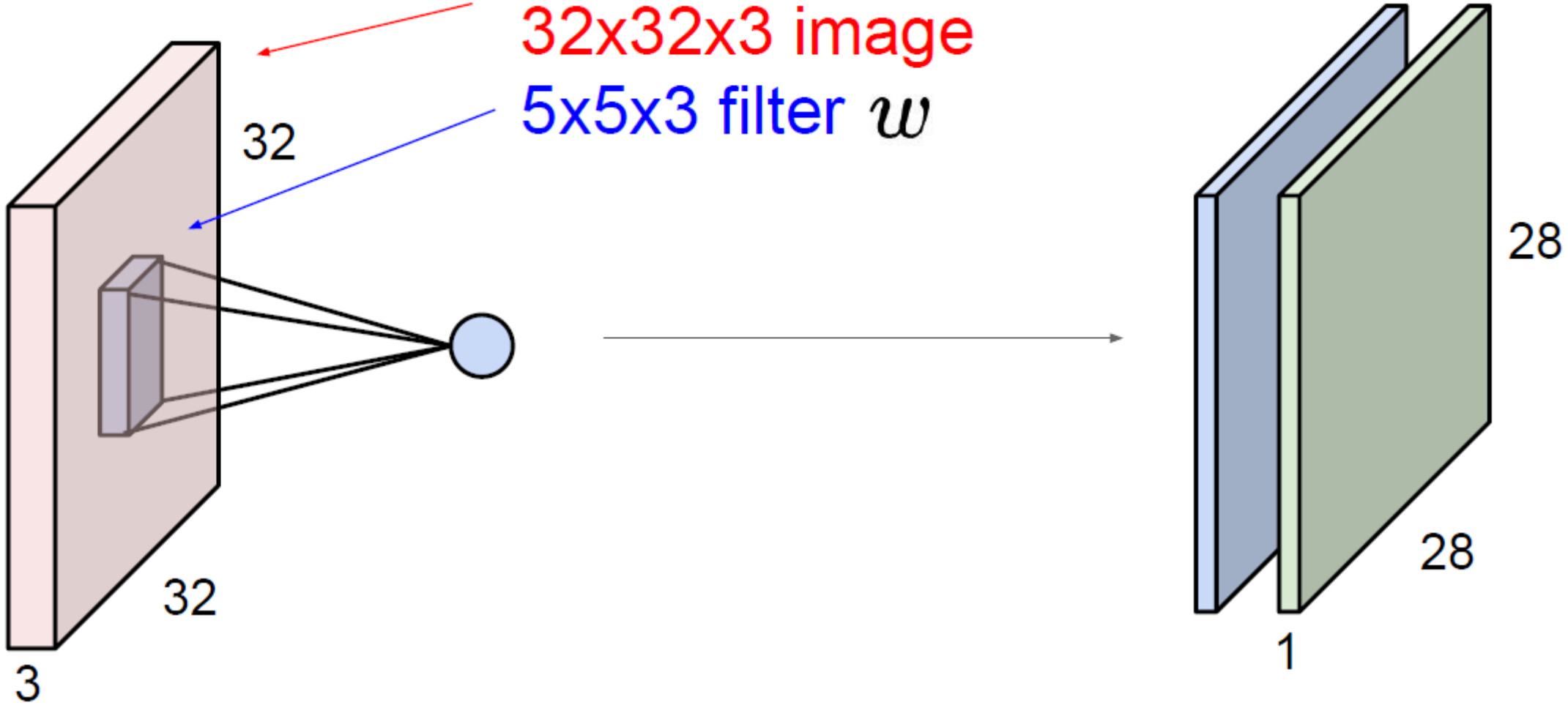
- 输入层
- 卷积层
- 激活函数
- 池化层
- 全连接层



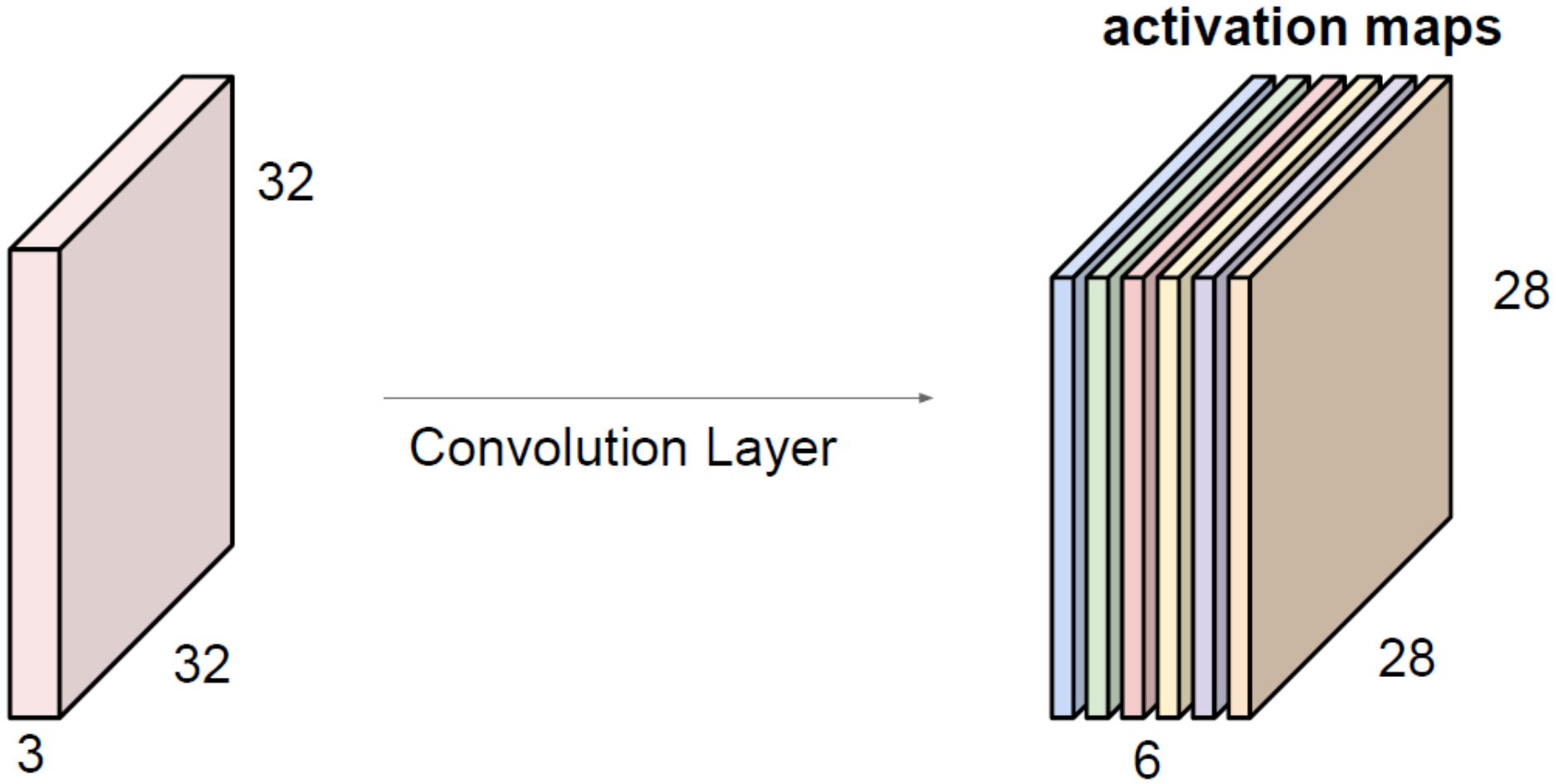
卷积神经网络：



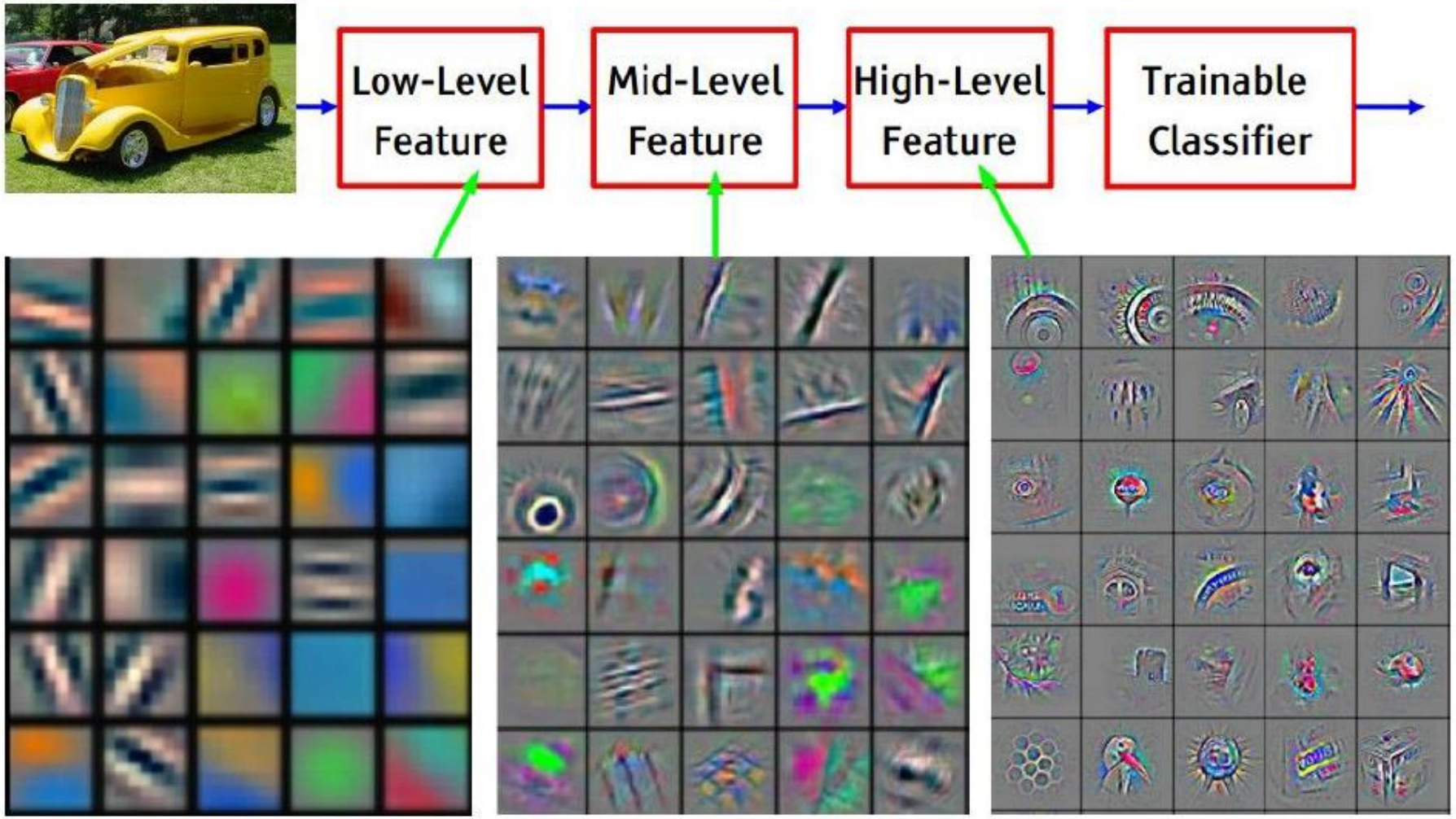
卷积神经网络：



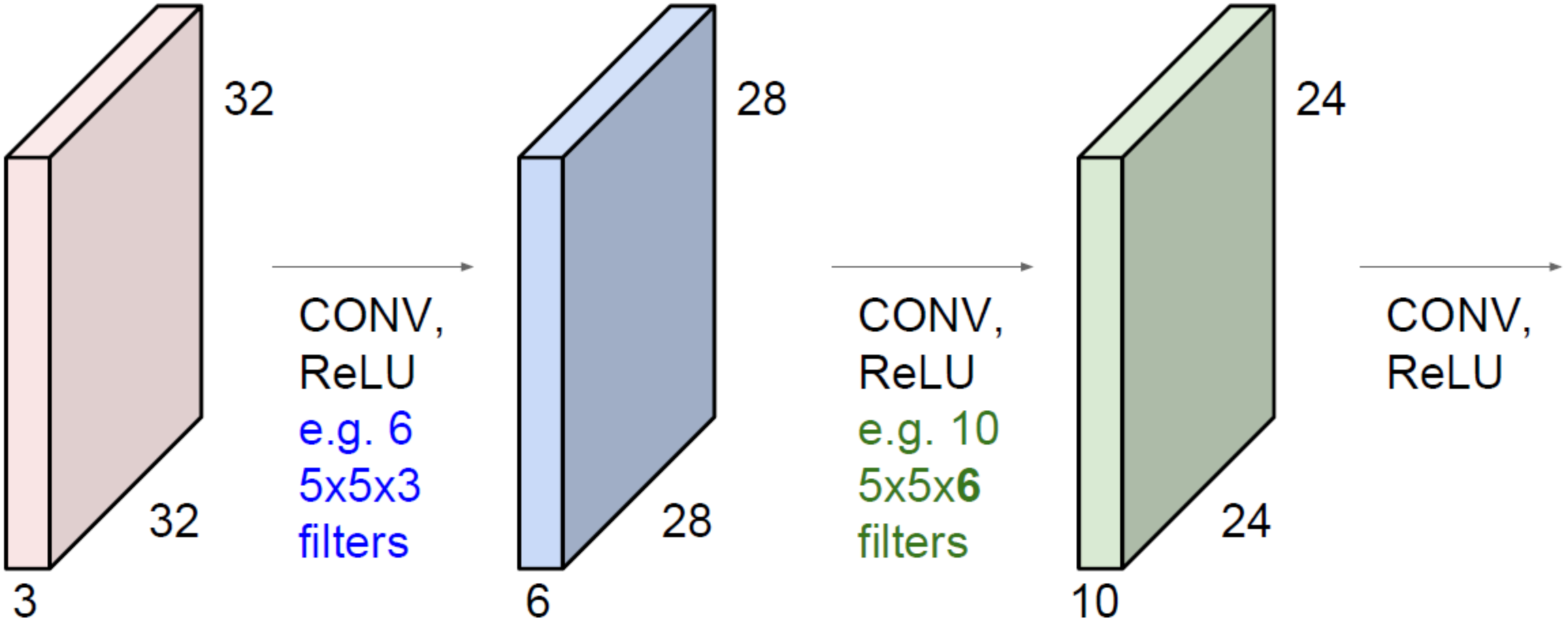
卷积神经网络：



卷积神经网络：


Low-Level Feature
Mid-Level Feature
High-Level Feature
Trainable Classifier

卷积神经网络：


32
32
3
CONV,
ReLU
e.g. 6
5x5x3
filters
28
28
6
CONV,
ReLU
e.g. 10
5x5x6
filters
24
24
10
CONV,
ReLU

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	2	2	1	0
0	1	1	1	2	1	0
0	2	1	1	0	2	0
0	2	1	0	1	2	0
0	2	1	2	2	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	1	2	0	1	0
0	2	2	1	1	0	0
0	2	1	0	0	2	0
0	1	0	0	0	2	0
0	0	1	0	1	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	0	1	2	0
0	0	0	2	1	2	0
0	2	1	0	2	1	0
0	1	1	0	0	0	0
0	0	0	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	1
-1	-1	0
-1	1	0

w0[:,:,1]

-1	-1	1
-1	1	0
-1	1	0

w0[:,:,2]

1	0	-1
0	0	0
1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	-1
-1	1	1
0	0	0

w1[:,:,1]

0	0	1
1	0	1
0	-1	-1

w1[:,:,2]

-1	1	1
0	1	1
1	-1	1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	-5	-4
4	-1	7
0	0	-1

o[:,:,1]

3	2	0
7	6	2
5	5	2

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	2	2	1	0
0	1	1	1	2	1	0
0	2	1	1	0	2	0
0	2	1	0	1	2	0
0	2	1	2	2	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	1	2	0	1	0
0	2	2	1	1	0	0
0	2	1	0	0	2	0
0	1	0	0	0	2	0
0	0	1	0	1	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	0	1	2	0
0	0	0	2	1	2	0
0	2	1	0	2	1	0
0	1	1	0	0	0	0
0	0	0	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	1
-1	-1	0
-1	1	0

w0[:,:,1]

-1	-1	1
-1	1	0
-1	1	0

w0[:,:,2]

1	0	-1
0	0	0
1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	-1
-1	1	1
0	0	0

w1[:,:,1]

0	0	1
1	0	1
0	-1	-1

w1[:,:,2]

-1	1	1
0	1	1
1	-1	1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

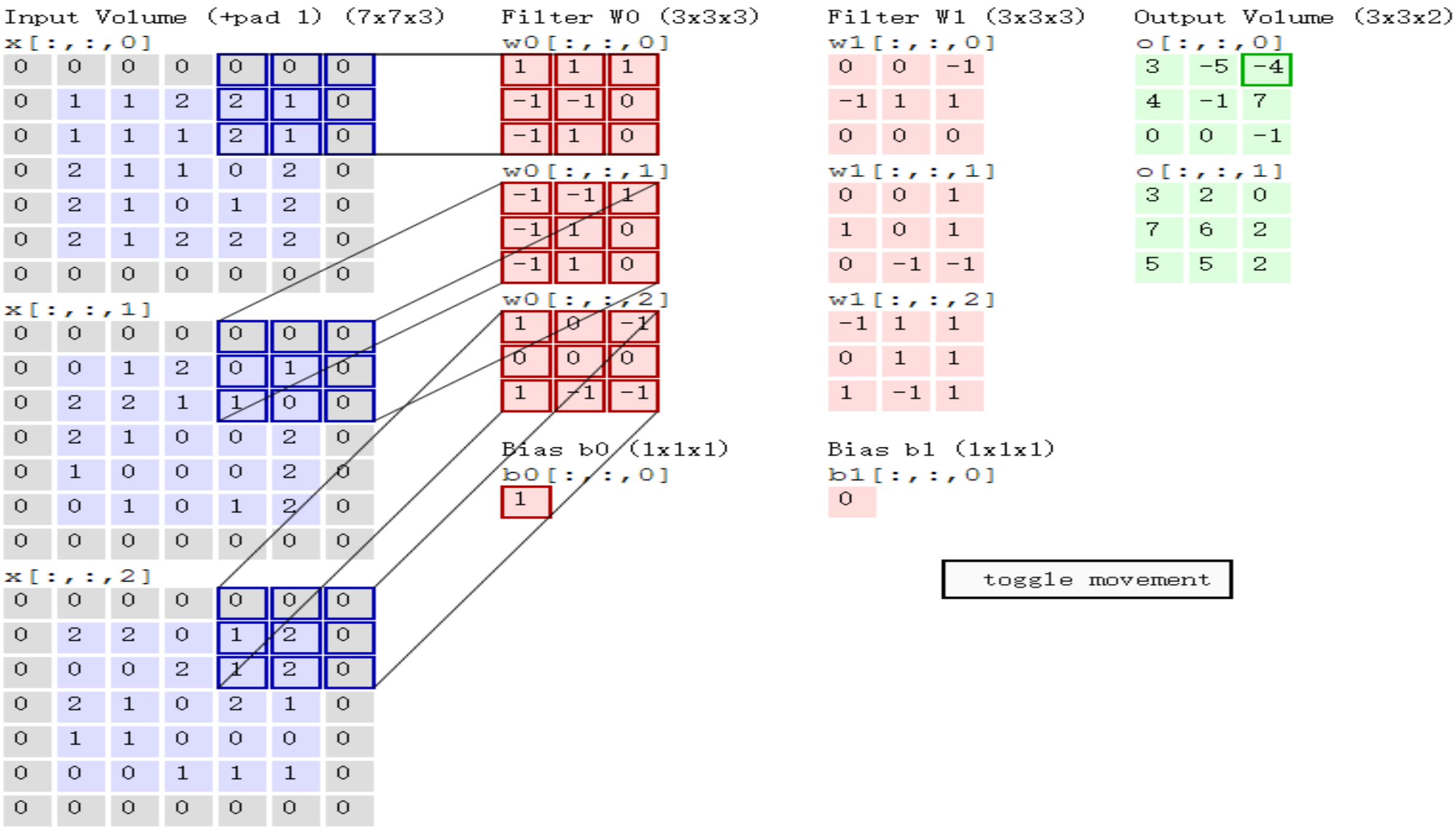
o[:,:,0]

3	-5	-4
4	-1	7
0	0	-1

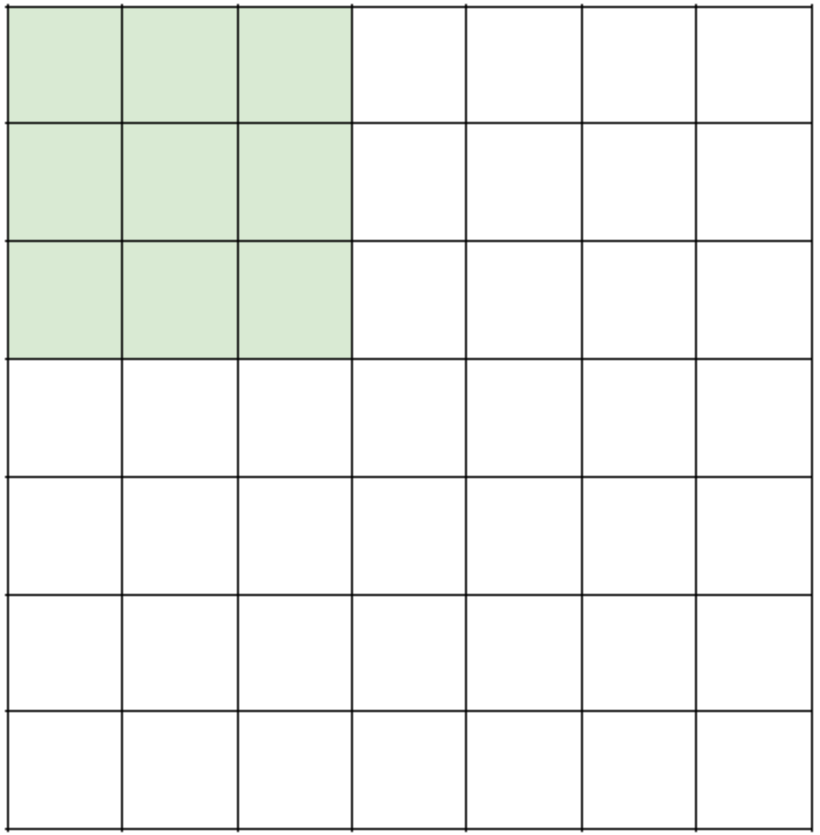
o[:,:,1]

3	2	0
7	6	2
5	5	2

toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 1 2 2 1 0
0 1 1 1 2 1 0
0 2 1 1 0 2 0
0 2 1 0 1 2 0
0 2 1 2 2 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 1 2 0 1 0
0 2 2 1 1 0 0
0 2 1 0 0 2 0
0 1 0 0 0 2 0
0 0 1 0 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 2 0 1 2 0
0 0 0 2 1 2 0
0 2 1 0 2 1 0
0 1 1 0 0 0 0
0 0 0 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 1 1
-1 -1 0
-1 1 0
w0[:,:,1]
-1 -1 1
-1 1 0
-1 1 0
w0[:,:,2]
1 0 -1
0 0 0
1 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
0 0 -1
-1 1 1
0 0 0
w1[:,:,1]
0 0 1
1 0 1
0 -1 -1
w1[:,:,2]
-1 1 1
0 1 1
1 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
3 -5 -4
4 -1 7
0 0 -1
o[:,:,1]
3 2 0
7 6 2
5 5 2
toggle movement

卷积神经网络：



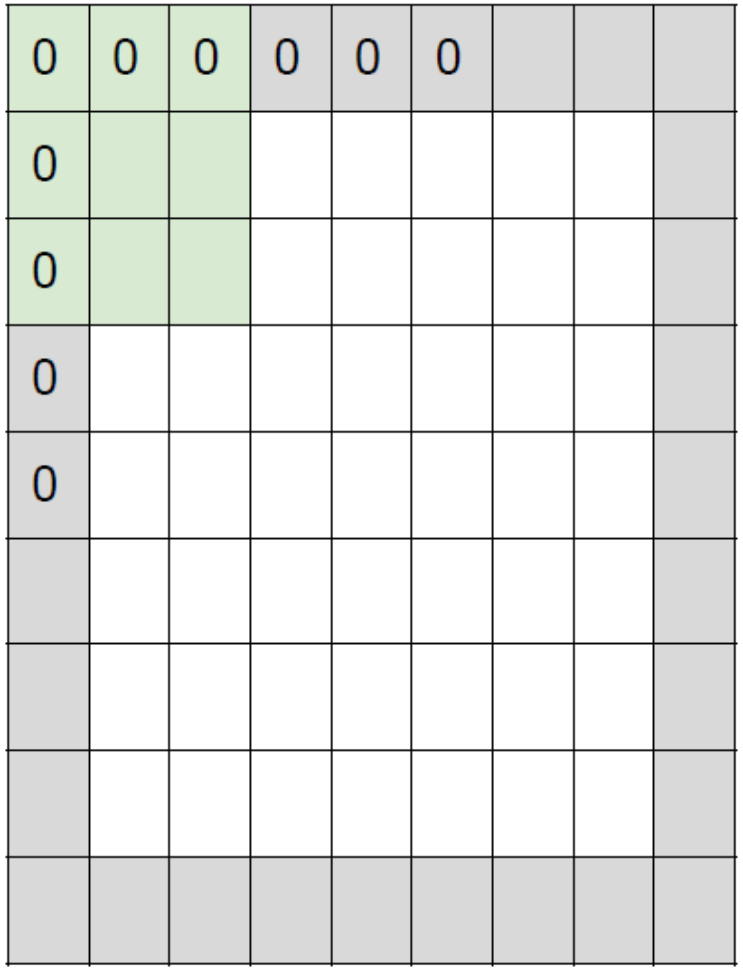
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

applied **with stride 2**
=> 3x3 output!

applied **with stride 3?**

卷积神经网络：

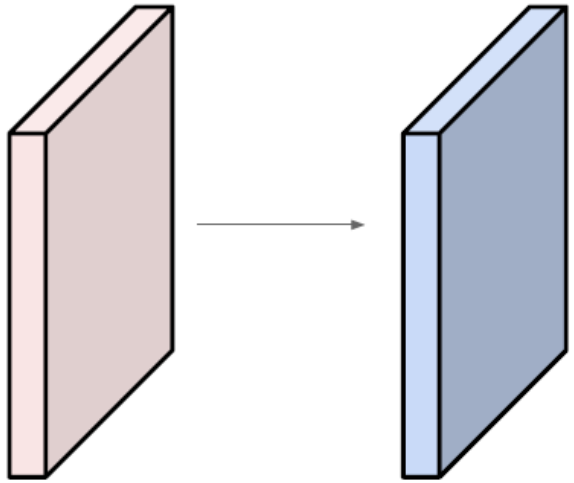


输入 = 7 x 7
Filter = 3 x 3
Pad = 1
Output = ?

F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

卷积神经网络：

Examples time:

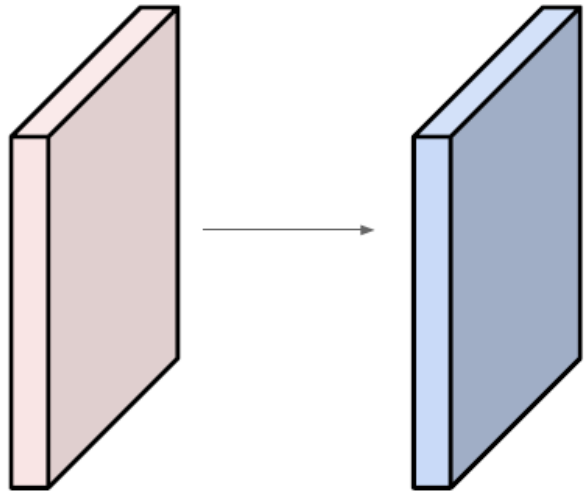


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

卷积神经网络：

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

卷积神经网络：

输入大小为：W1 x H1 x D1

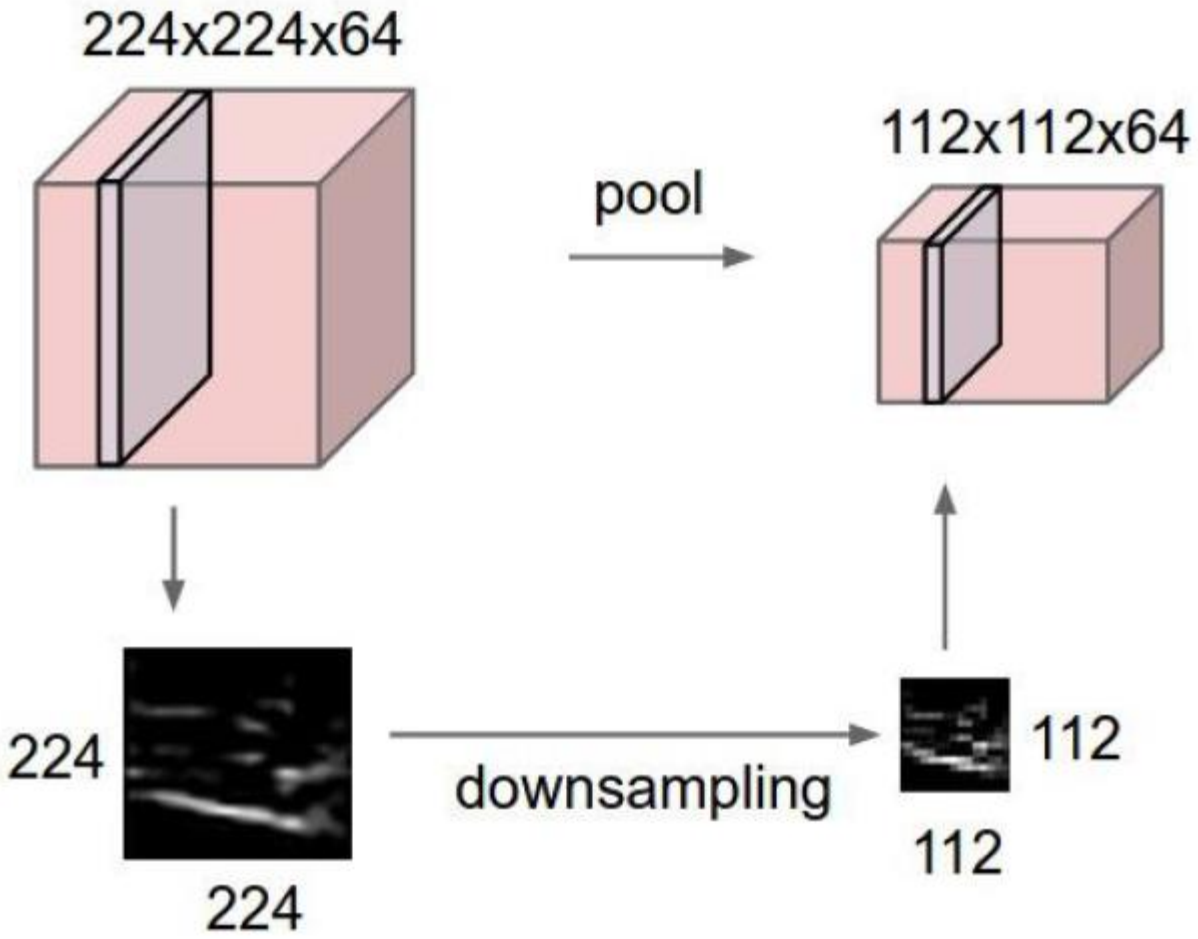
需要指定的超参数：filter个数（K），filter大小（F），步长（S），边界填充（P）

输出：

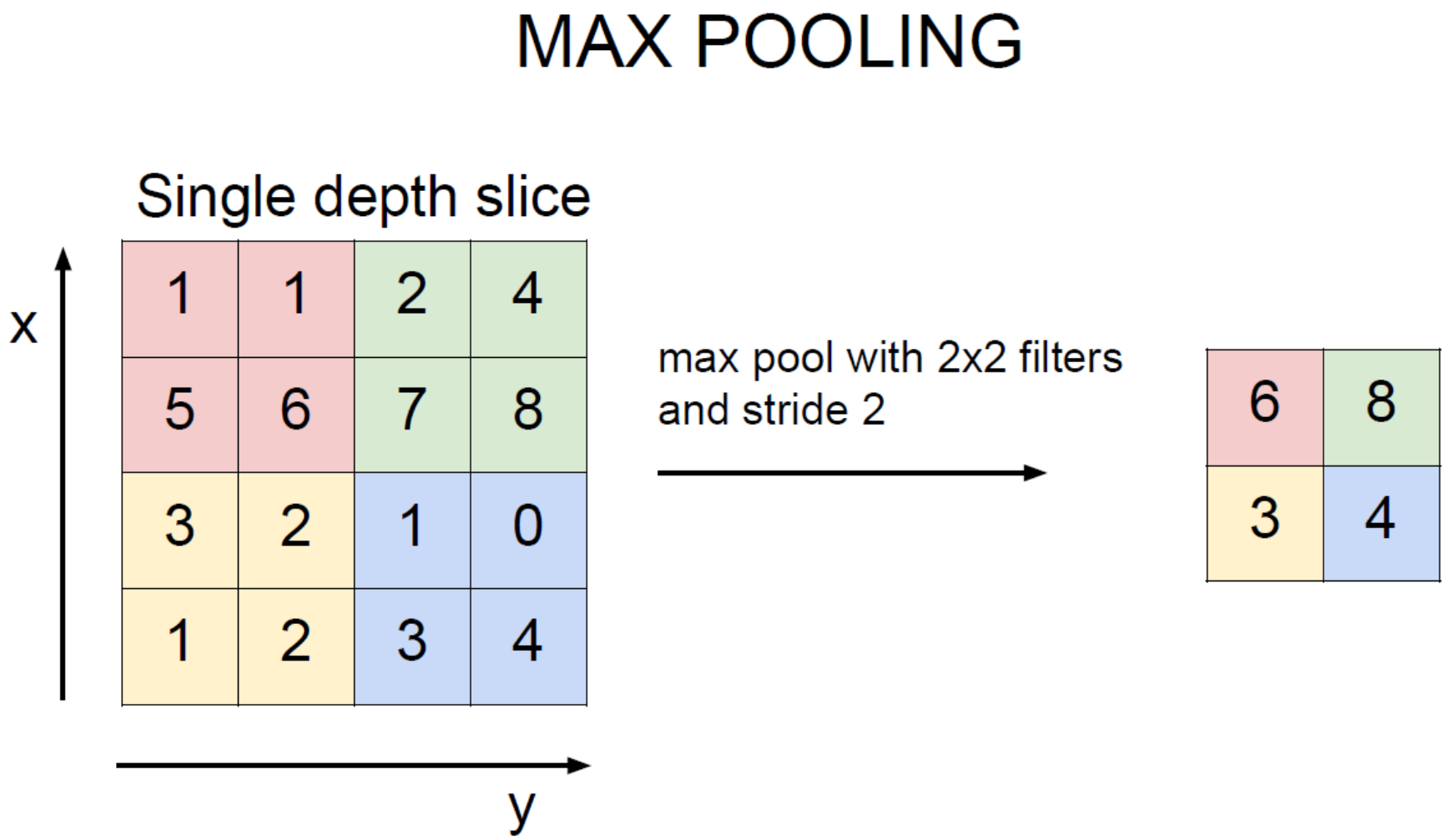
$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$
$$D_2 = K$$

卷积神经网络：

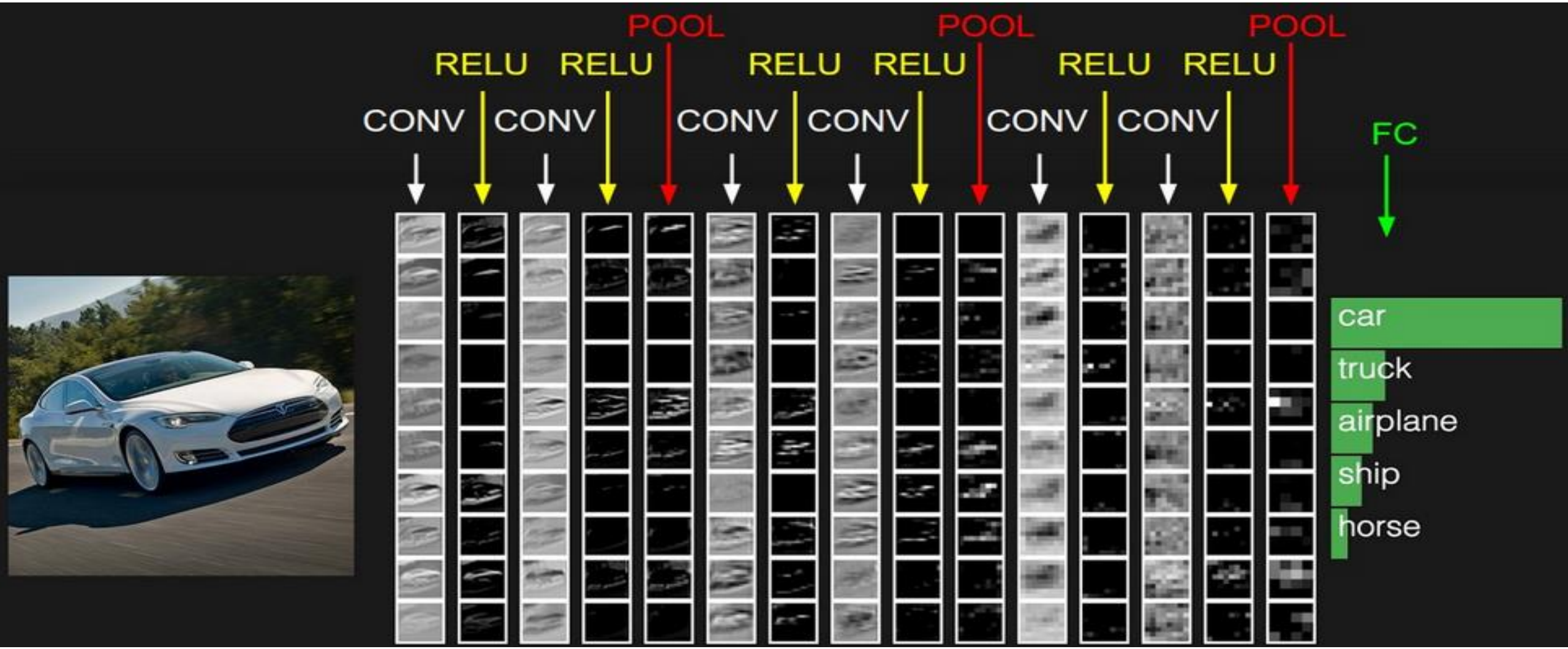
Pooling layer



卷积神经网络：

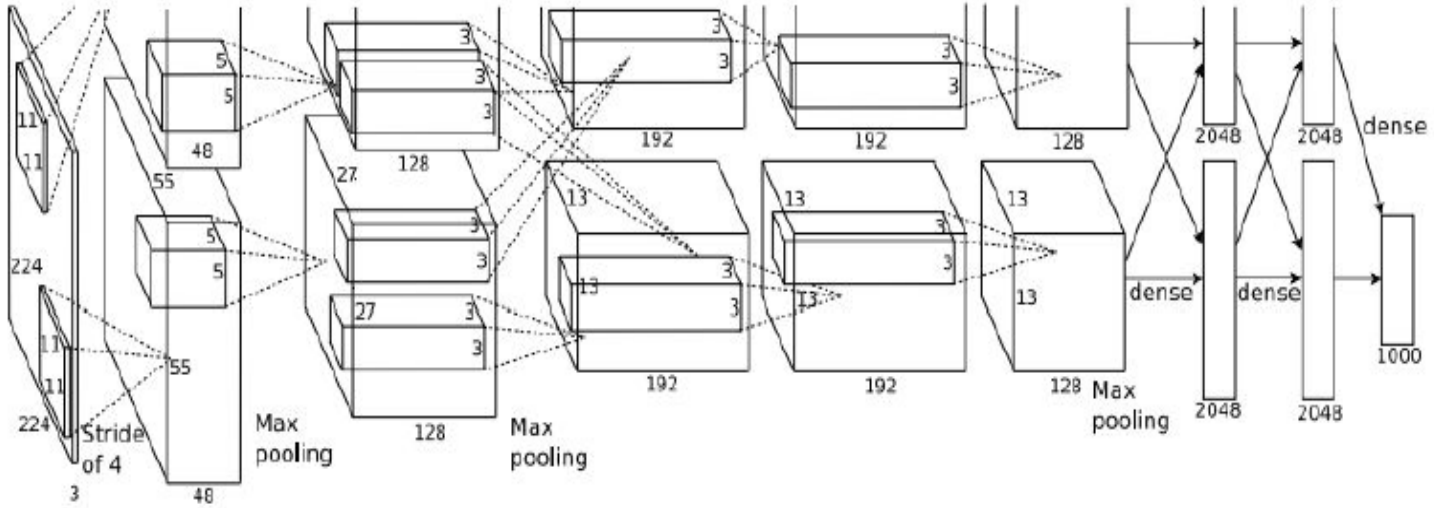

MAX POOLING
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

卷积神经网络：

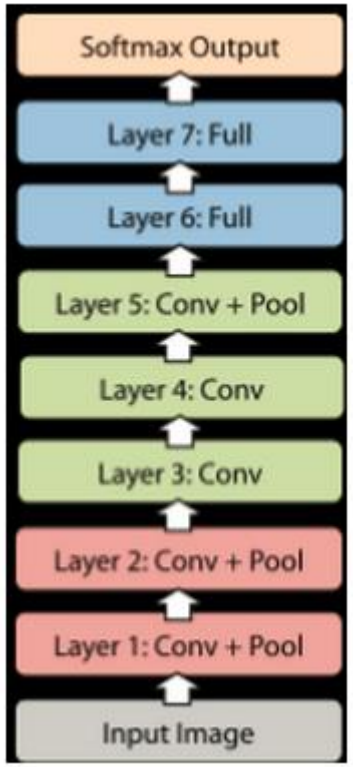

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

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 conv3-25 **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					

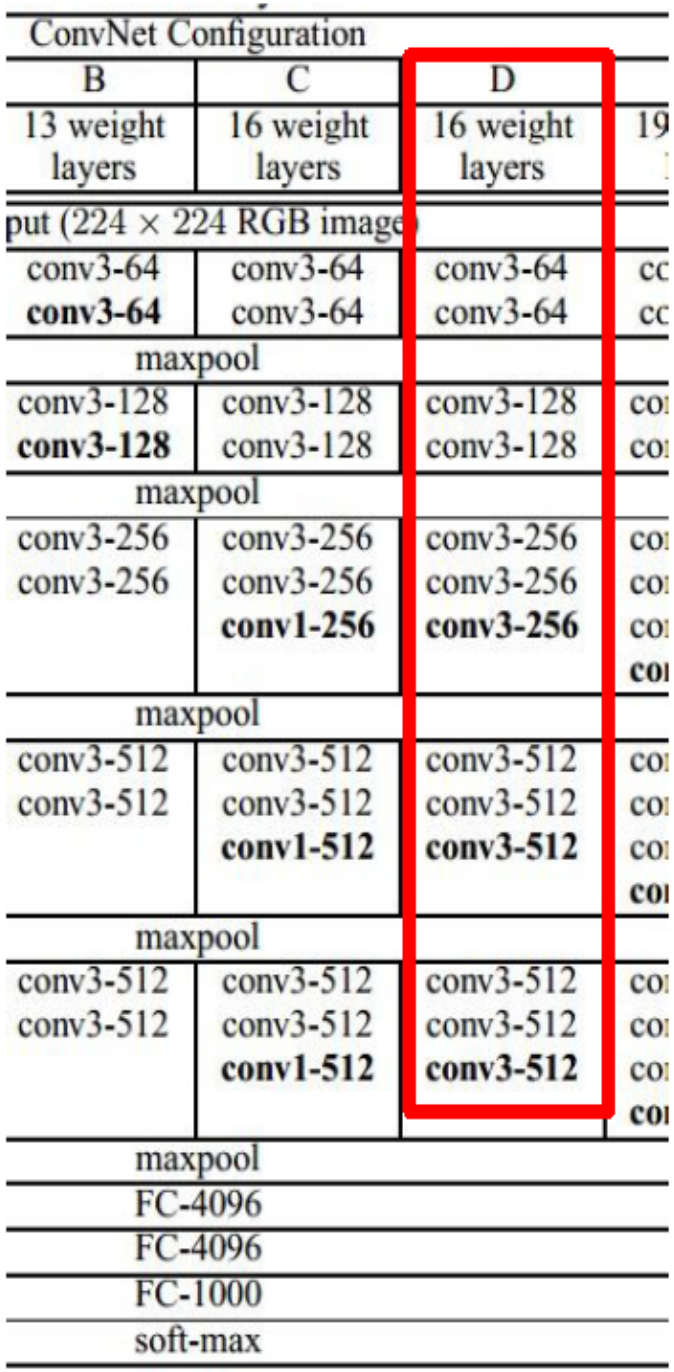
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

(not counting biases)

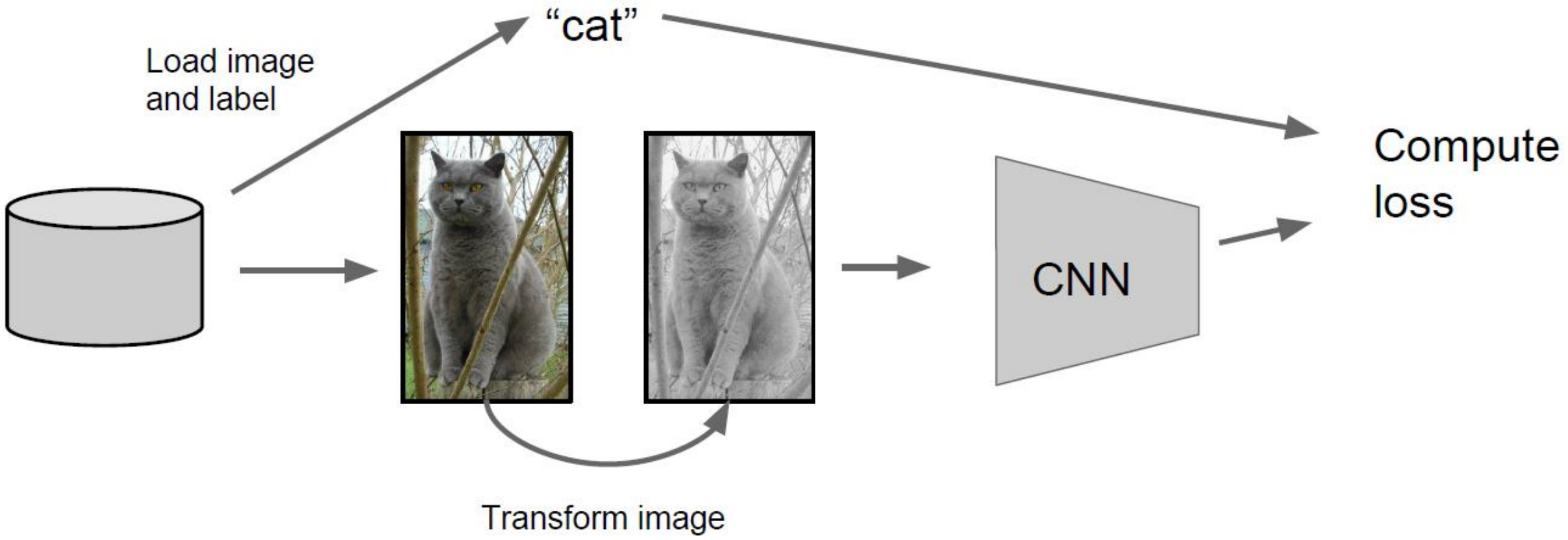
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



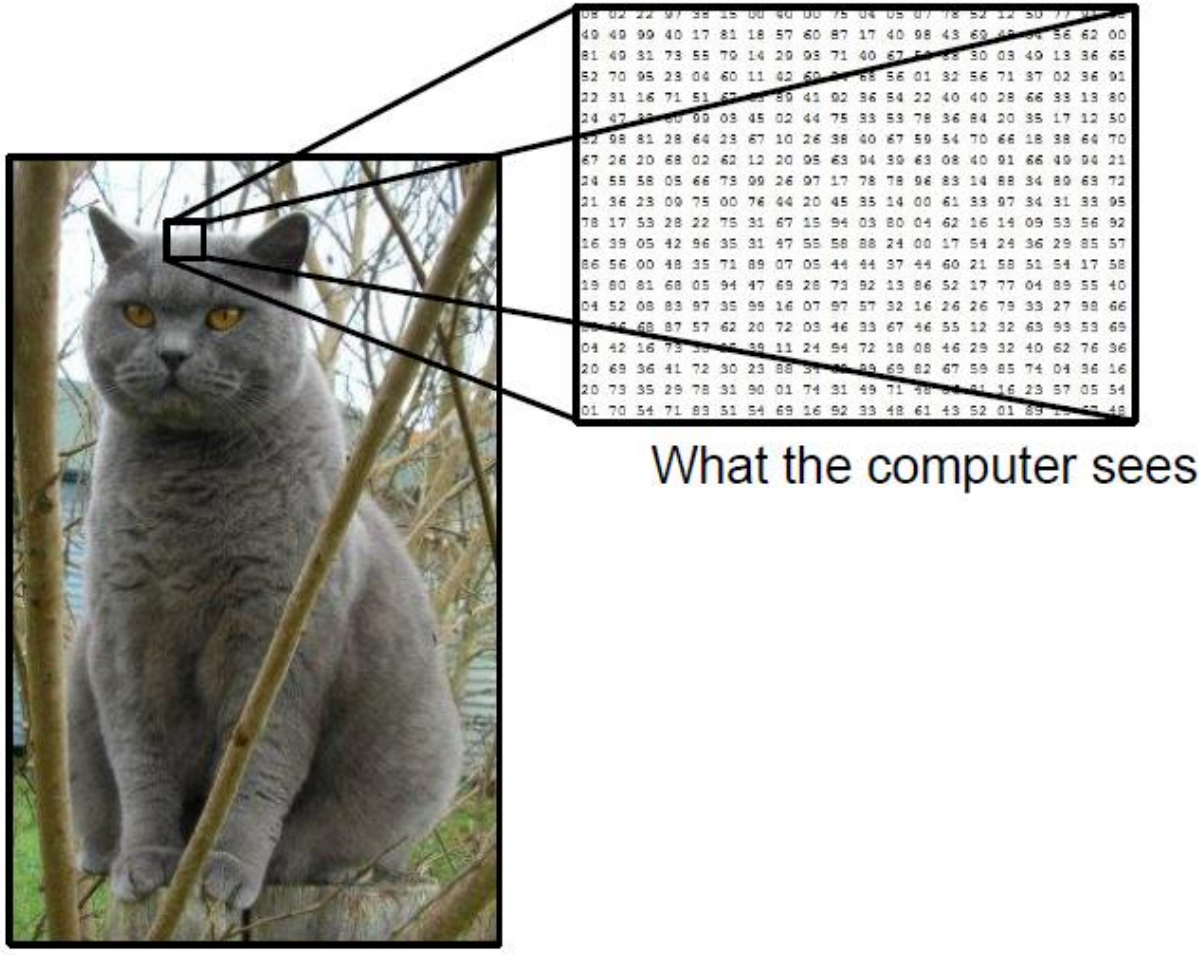
ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	cc
conv3-64	conv3-64	conv3-64	cc
maxpool			
conv3-128	conv3-128	conv3-128	co
conv3-128	conv3-128	conv3-128	co
maxpool			
conv3-256	conv3-256	conv3-256	co
conv3-256	conv3-256	conv3-256	co
	conv1-256	**conv3-256**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

Data Augmentation：



Data Augmentation:

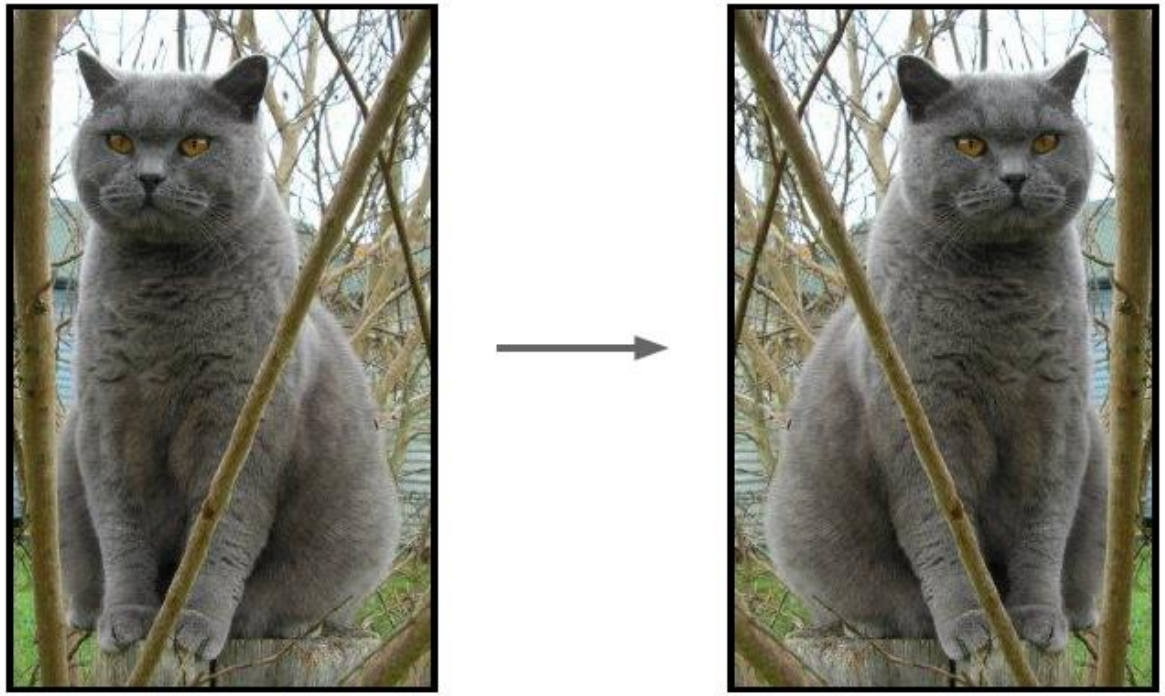
- Change the pixels without changing the label
- Train on transformed data
- VERY widely used



What the computer sees

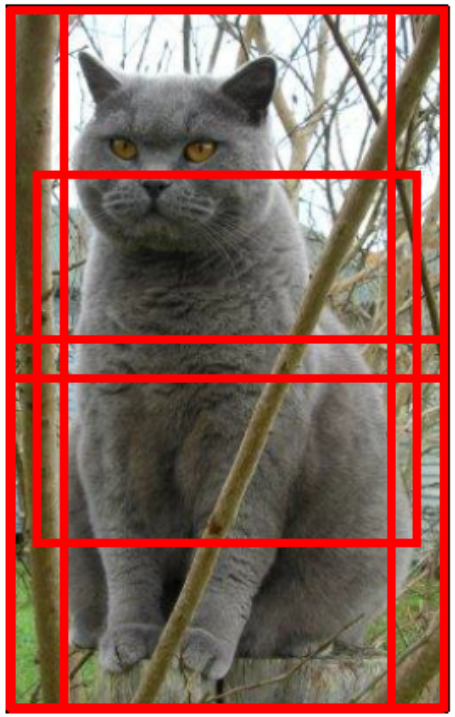
Data Augmentation:

1. Horizontal flips



Data Augmentation:

2. Random crops/scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Data Augmentation:

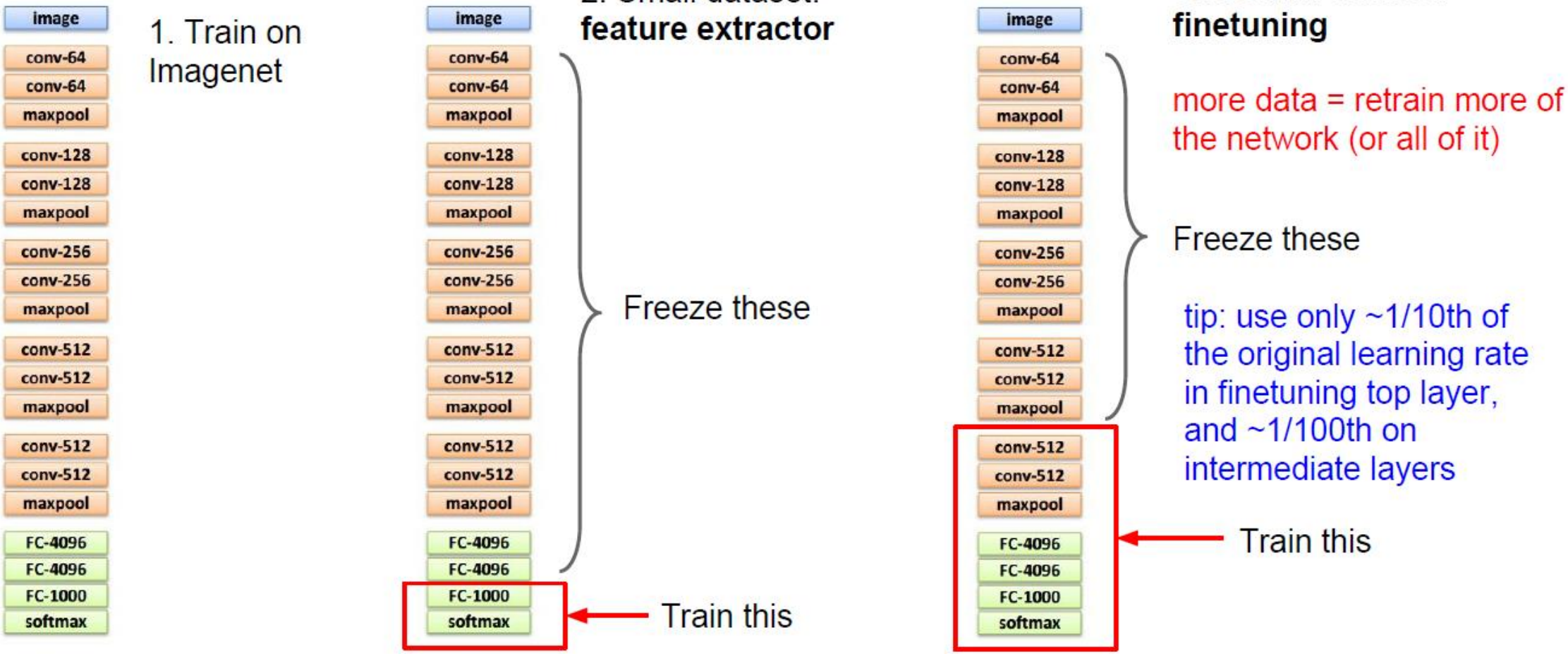
Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

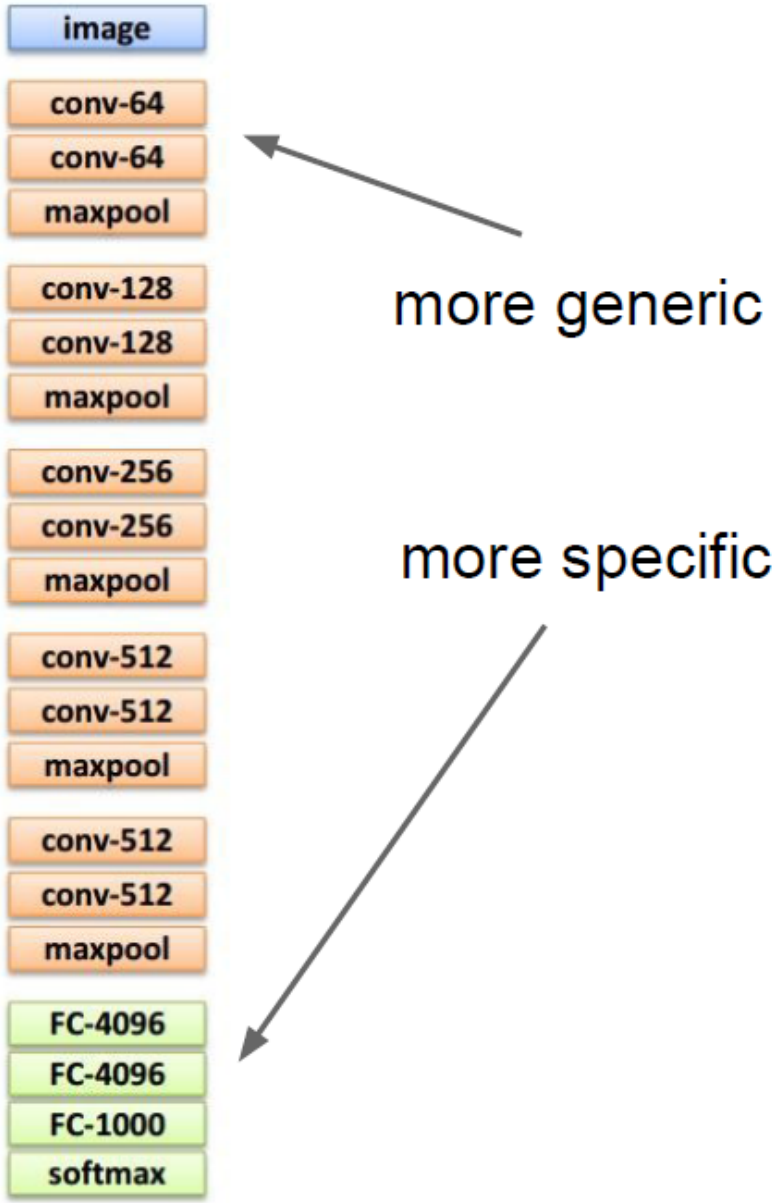
Transfer Learning:

"You need a lot of a data if you want to train/use CNNs"

Transfer Learning:



Transfer Learning:



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Transfer Learning:

Takeaway for your projects/beyond:

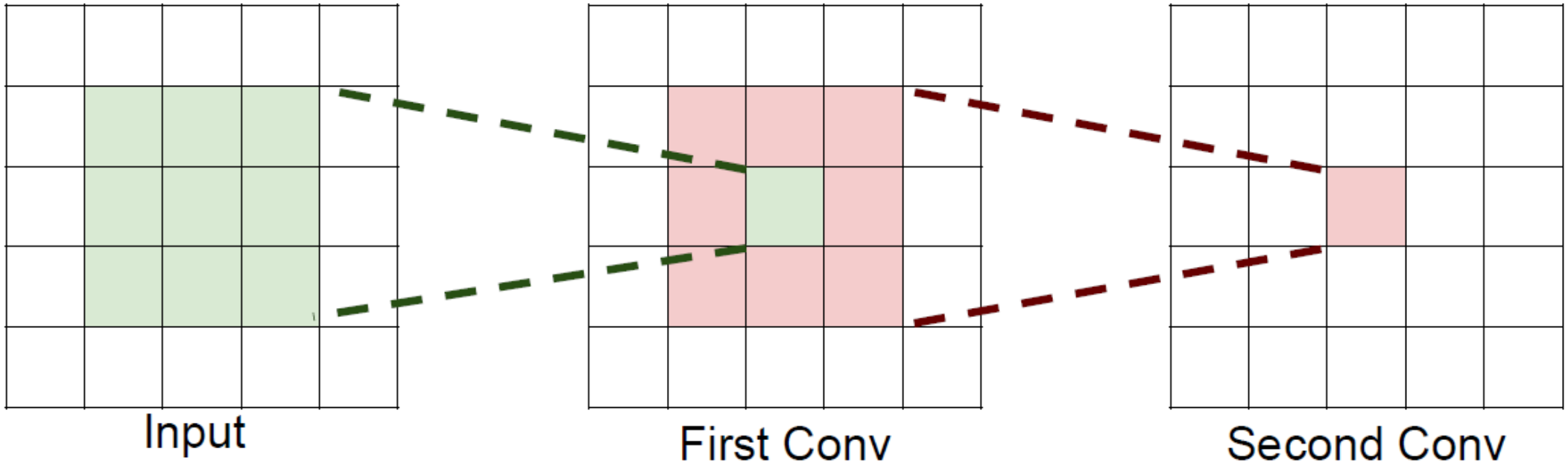
Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there.
2. Transfer learn to your dataset

Caffe ConvNet library has a **"Model Zoo"** of pretrained models:
https://github.com/BVLC/caffe/wiki/Model-Zoo

Design Conv Network:

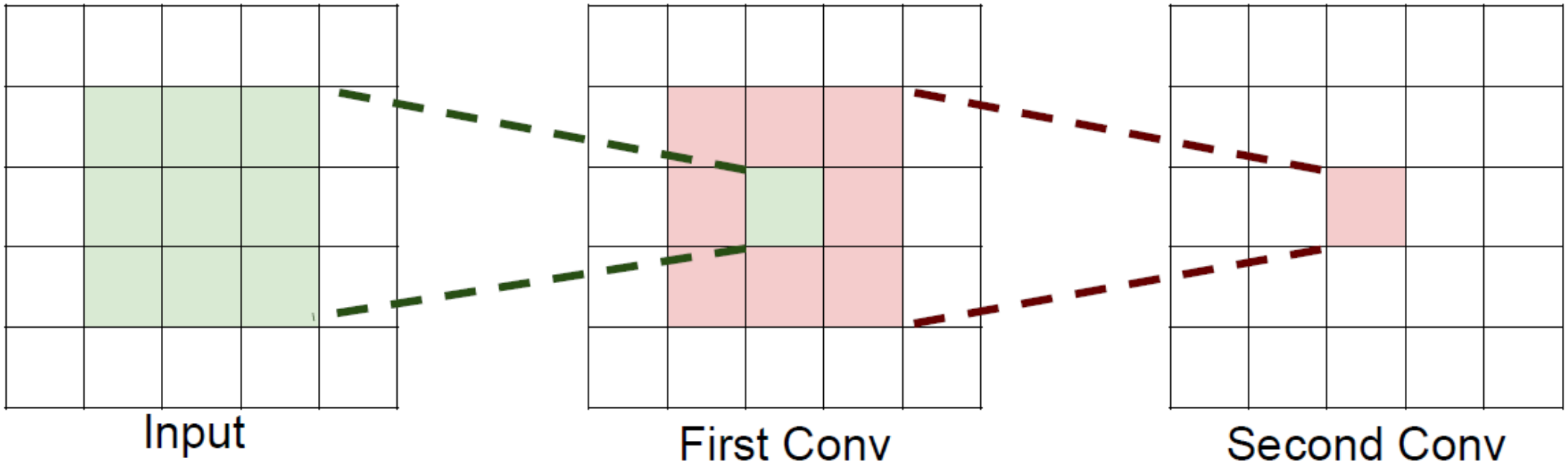
Suppose we stack two 3x3 conv layers (stride 1)
Each neuron sees 3x3 region of previous activation map



Design Conv Network:

Question: How big of a region in the input does a neuron on the second conv layer see?

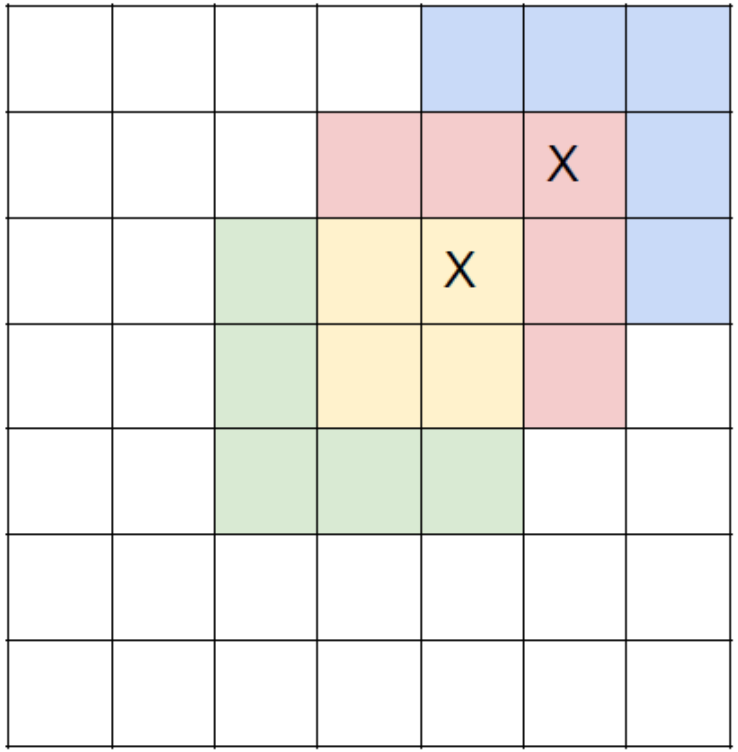
Answer: 5 x 5



Design Conv Network:

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Answer: 7 x 7



Three 3 x 3 conv gives similar representational power as a single 7 x 7 convolution

Design Conv Network:

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = **49 C^2**

three CONV with 3 x 3 filters

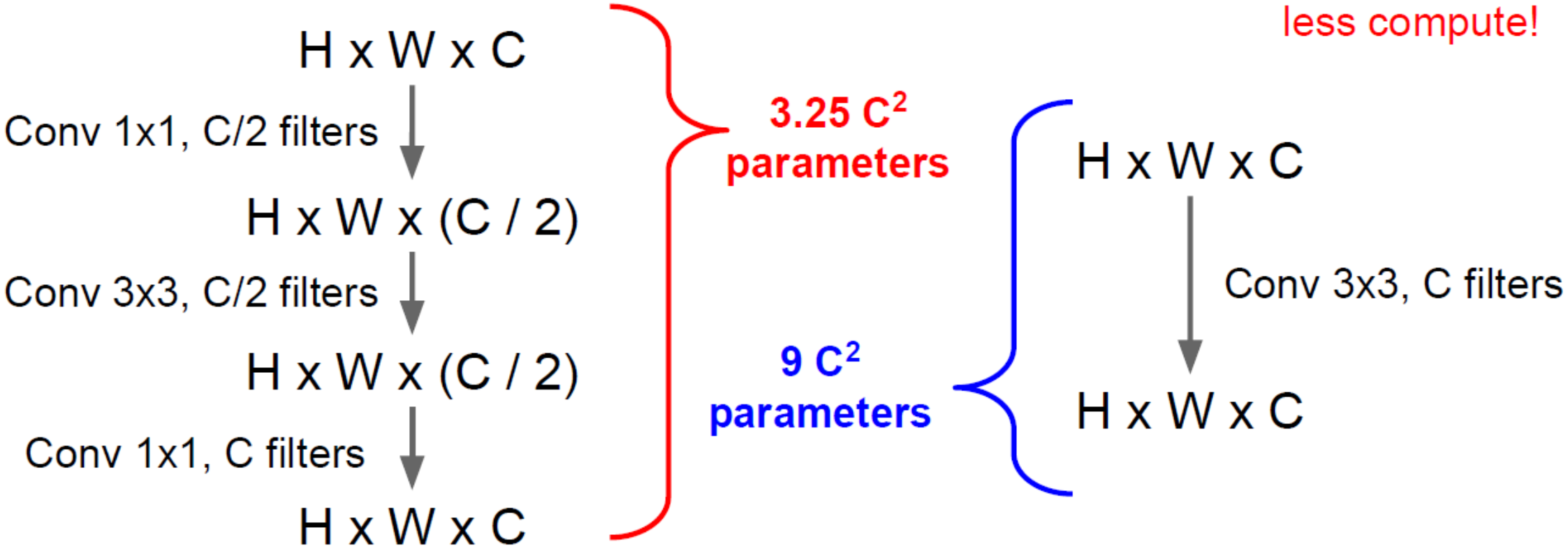
Number of weights:
= 3 x C x (3 x 3 x C) = **27 C^2**

Fewer parameters, more nonlinearity = GOOD

Design Conv Network:

Why stop at 3 x 3 filters? Why not try 1 x 1?

More nonlinearity, fewer params, less compute!



Design Conv Network：

- Replace large convolutions (5 x 5, 7 x 7) with stacks of 3 x 3 convolutions
- 1 x 1 “bottleneck” convolutions are very efficient
- Can factor N x N convolutions into 1 x N and N x 1
- All of the above give fewer parameters, less compute, more nonlinearity

GPU VS CPU:

Spot the CPU!

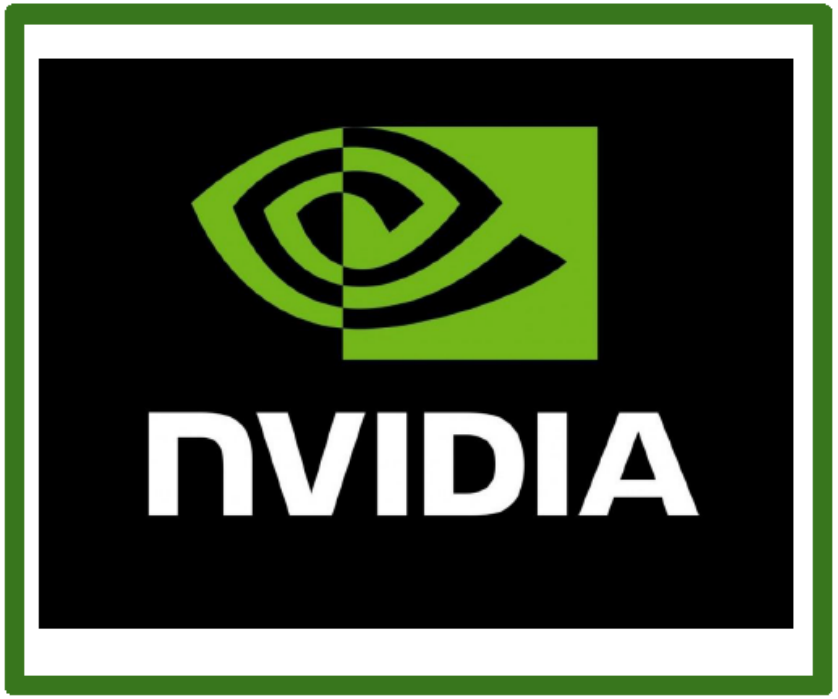
"central processing unit"

GPU VS CPU:

Spot the GPU!

"graphics processing unit"

GPU VS CPU:



VS

AMD

NVIDIA is much more common for deep learning

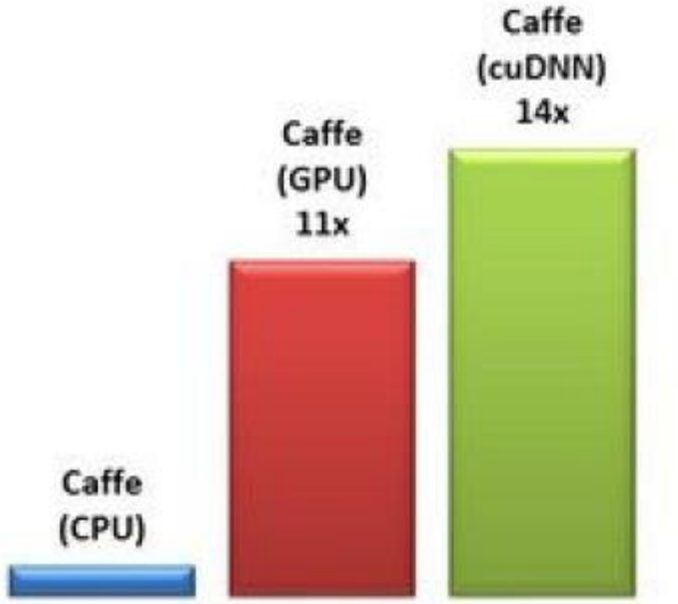
GPU VS CPU:

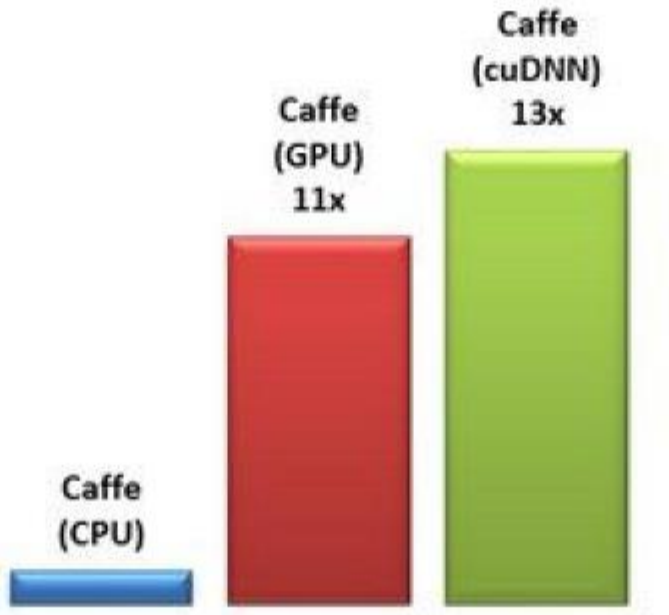
CPU
Few, fast cores (1 - 16)
Good at sequential processing

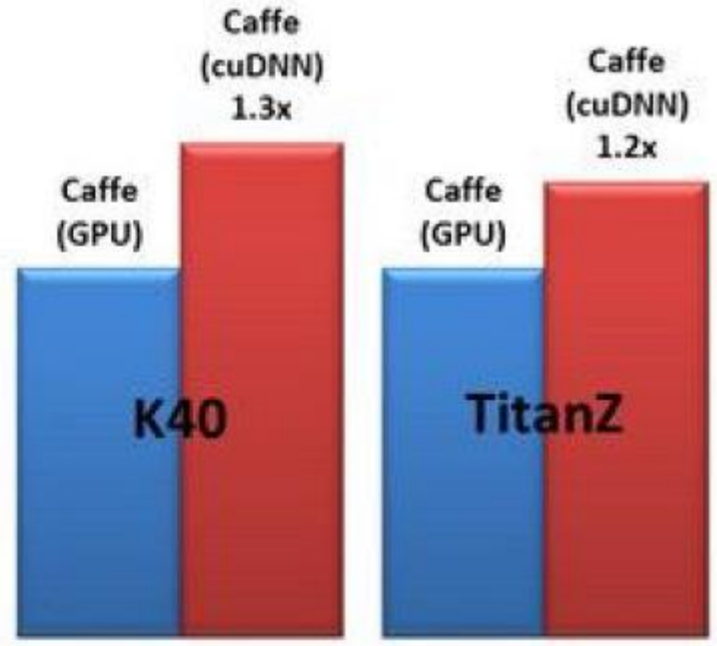
GPU
Many, slower cores (thousands)
Originally for graphics
Good at parallel computation

GPU VS CPU：

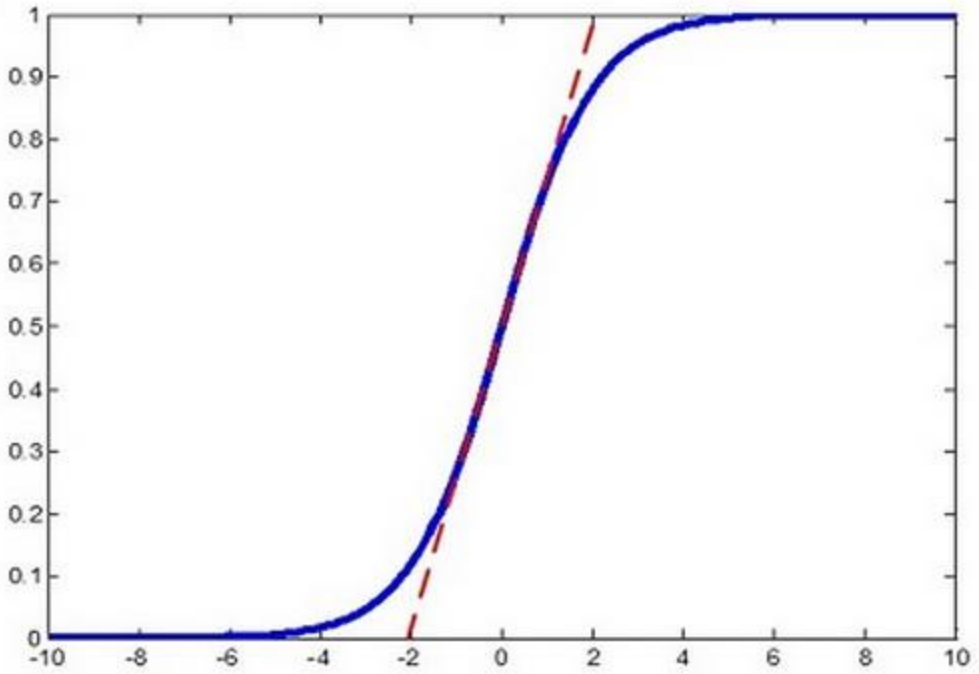
GPUs are really good at convolution (cuDNN):







Batch Normalization：



Caffe：

Caffe

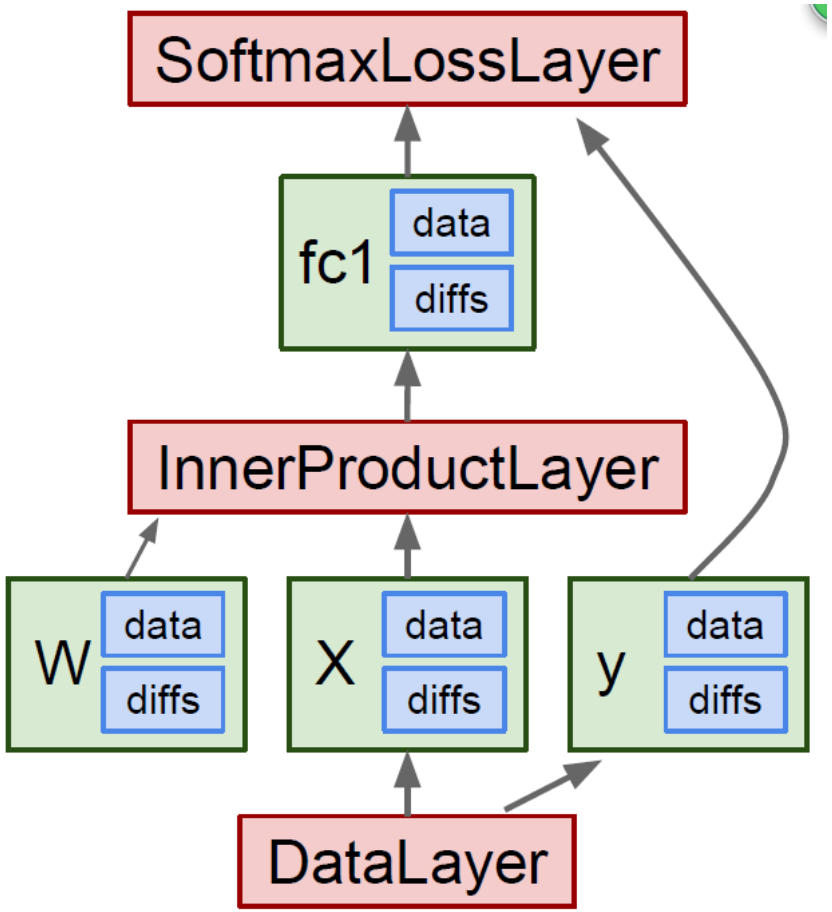
http://caffe.berkeleyvision.org

Caffe Overview:

- From U.C. Berkeley
- Written in C++
- Has Python and MATLAB bindings
- Good for training or finetuning feedforward models

Caffe Main Classes:

- **Blob**: Stores data and derivatives (header source)
- **Layer**: Transforms bottom blobs to top blobs (header + source)
- **Net**: Many layers; computes gradients via forward / backward (header source)
- **Solver**: Uses gradients to update weights (header source)



Caffe Trainging:

No need to write code!

1. Convert data (run a script)
2. Define net (edit prototxt)
3. Define solver (edit prototxt)
4. Train (with pretrained weights) (run a script)

Caffe Step 1: Convert Data

- DataLayer reading from LMDB is the easiest
- Create LMDB using convert_imageset
- Need text file where each line is
  - "[path/to/image.jpeg] [label]"
- Create HDF5 file yourself using h5py

Caffe:

Caffe Step 2: Define Net

```
name: "LogisticRegressionNet"
layers {
  top: "data"
  top: "label"
  name: "data"
  type: HDF5_DATA
  hdf5_data_param {
    source: "examples/hdf5_classification/data/train.txt"
    batch_size: 10
  }
  include {
    phase: TRAIN
  }
}
layers {
  bottom: "data"
  top: "fc1"
  name: "fc1"
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layers {
  bottom: "fc1"
  bottom: "label"
  top: "loss"
  name: "loss"
  type: SOFTMAX_LOSS
}
```

Caffe Step 3: Define Solver

- Write a prototxt file defining a SolverParameter
- If finetuning, copy existing solver. prototxt file
  - Change net to be your net
  - Change snapshot_prefix to your output
  - Reduce base learning rate (divide by 100)
  - Maybe change max_iter and snapshot

```
net: "models/bvlc_alexnet/train_val.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.01
lr_policy: "step"
gamma: 0.1
stepsize: 100000
display: 20
max_iter: 450000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/bvlc_alexnet/caffe_alexnet_train"
solver_mode: GPU
```

Caffe Step 4: Train!

```
./build/tools/caffe train \
  -gpu 0 \
  -model path/to/trainval.prototxt \
  -solver path/to/solver.prototxt \
  -weights path/to/pretrained_weights.caffemodel
```

Caffe: Model Zoo

AlexNet, VGG,
GoogLeNet, ResNet,
plus others