

# 卷积神经网络

## Introduction to Convolutional Neural Network

薛盛可  
xueshengke@zju.edu.cn

College of Information Science and Electronic Engineering  
Zhejiang University

Spring, 2016

# Outline

## 1 简介

## 2 卷积神经网络特性

## 3 前向传播

## 4 网络训练

## 5 应用

- LeNet
- MCDNN
- AlexNet
- DeepID
- Pyramid CNN
- Fast R-CNN

## 6 我的尝试

- STL-10
- CalTech101
- CIFAR-10
- 自编码算法

## 7 参考文献

# Introduction

卷积神经网络（Convolutional Neural Network, CNN）是一种特殊的人工神经网络，由多对卷积层（Convolutional layer）和池化层（Pooling layer），以及全连接层（Full-connected layer）组成。网络的前向传播主要是卷积和池化（降采样，subsample）操作，其网络参数的更新（学习）方式采用误差反向传播（Error Backpropagation）算法。

其主要有以下几个特点：

- 直接输入原始的图像数据
- 局部感知，神经元结点与前一层的部分结点连接
- 参数共享
- 多卷积核
- 误差反向传播，有效更新每一层的网络参数

# 卷积神经网络模型

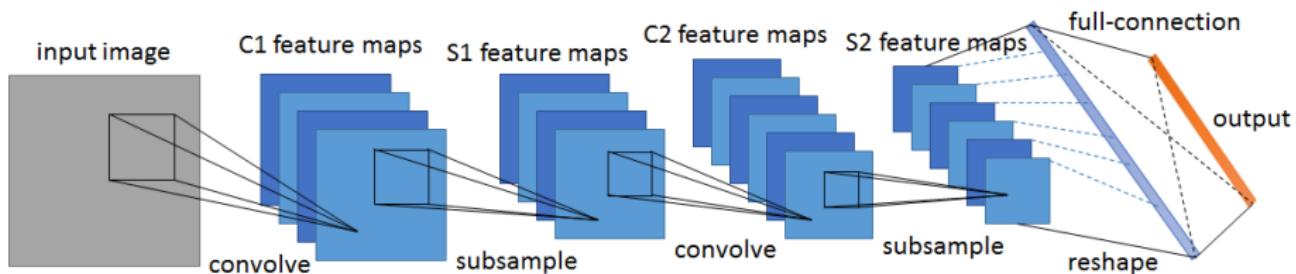


Figure 1: 一个典型的卷积神经网络结构

# Outline

1 简介

2 卷积神经网络特性

3 前向传播

4 网络训练

5 应用

- LeNet
- MCDNN
- AlexNet
- DeepID
- Pyramid CNN
- Fast R-CNN

6 我的尝试

- STL-10
- CalTech101
- CIFAR-10
- 自编码算法

7 参考文献

# 局部感知

一般认为人对图像的认知是从局部到整体的，而图像的空间联系也是邻近的像素联系较为紧密，而距离较远的像素相关性则较弱。因而，每个神经元其实没有必要对全局图像进行感知，只需要对局部进行感知，然后在更高层将局部的信息综合起来就得到了全局的信息。网络部分连通的思想，也是受启发于生物学里面的视觉系统结构：视觉皮层的神经元就是局部接受信息的（即这些神经元只响应某些特定区域的刺激）。

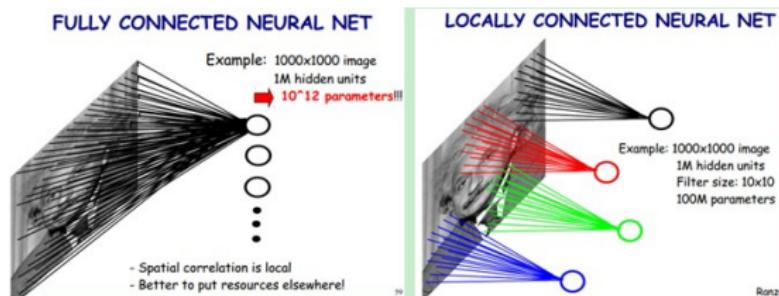


Figure 2: 全连接与局部连接

在上右图中，假如每个神经元只和  $10 \times 10$  个像素点相连，那么一共需要  $1M \times 100$  个参数，减少为全连接的万分之一。

# 参数共享

但其实这样参数仍然过多，则采用参数共享。在局部连接中，每个神经元都对应100个参数，一共1M个神经元，如果这1M个神经元的100个参数都是相同的，那么参数数量就变为100了，就称为一个  $10 \times 10$  卷积核。

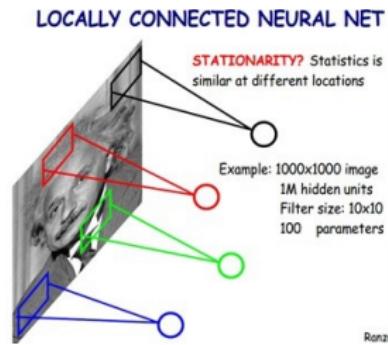


Figure 3: 参数共享

怎么理解权值共享呢？我们可以把100个参数（卷积核）看成是从图像的任意区域提取特征的方式，如边缘特征。可理解为图像的滤波操作（filtering）。

# 多卷积核

上面所述只有100个参数时，表明只有1个  $10 \times 10$  的卷积核。显然，特征提取是不充分的。但我们可以添加多个卷积核，比如100个卷积核，可以学习100种特征。此时的参数数量为  $100 \times 100$  个。

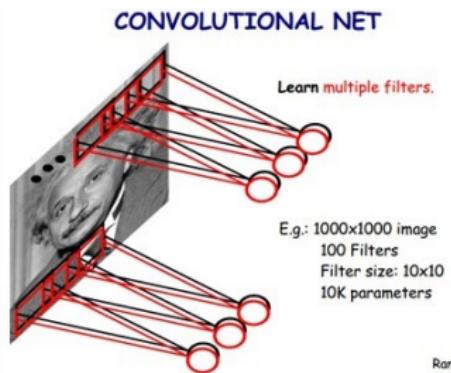


Figure 4: 多卷积核

不同颜色代表不同的卷积核。每个卷积核都会将图像生成为另一幅特征图（feature map）。比如100个卷积核就生成100幅特征图。

# Outline

1 简介

2 卷积神经网络特性

3 前向传播

4 网络训练

5 应用

- LeNet
- MCDNN
- AlexNet
- DeepID
- Pyramid CNN
- Fast R-CNN

6 我的尝试

- STL-10
- CalTech101
- CIFAR-10
- 自编码算法

7 参考文献

# 卷积

卷积：前一层的图像对应区域的像素与卷积核（翻转180度）相乘得到该层的特征图。

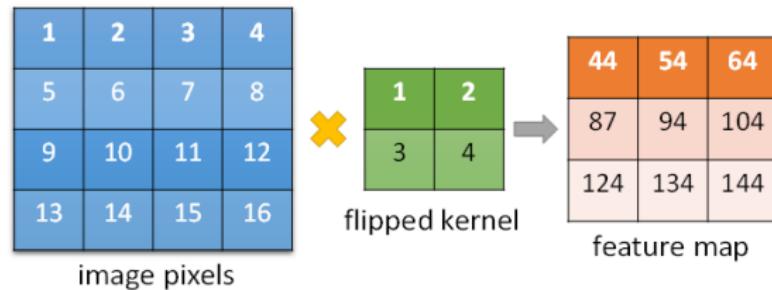


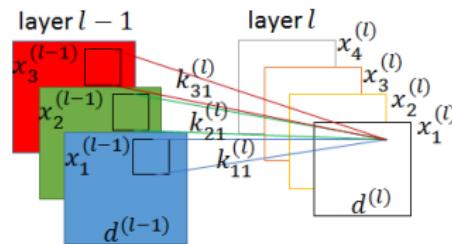
Figure 5: 图像卷积过程

这只是最简单的卷积。但是网络中前一层有多个特征图，又有多个卷积核存在，卷积过程略微复杂一些。

# 卷积

$$u_j^{(l)} = \sum_{i \in M_j} x_i^{(l-1)} \bullet k_{ij}^{(l)} + b_j^{(l)}. \quad (1)$$

其中  $\bullet$  表示卷积操作。 $i$  为  $l-1$  层的特征图数量， $j$  为  $l$  层的特征图数量。 $x_i^{(l-1)}$  为  $l-1$  层第  $i$  个特征图， $k_{ij}^{(l)}$  为对应特征图之间的卷积核， $b_j^{(l)}$  为  $l$  层第  $j$  个偏置。 $M_j$  代表输入特征图的组合，可以是全部输入，也可以选择部分作为输入。



$$k_{ij}^{(l)} : a \times a$$

图像卷积之后变小

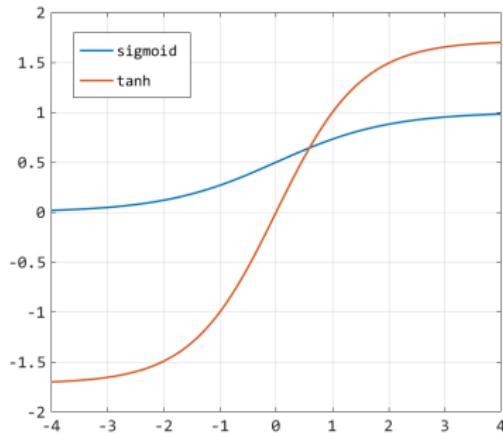
$$d^{(l)} = d^{(l-1)} - a + 1.$$

Figure 6: 卷积层操作

# 非线性

卷积之后还需经过非线性激活函数，输出才得到下一层的特征图。常见的激活函数有sigmoid和tanh两种。

$$\mathbf{x}_j^{(l)} = f(\mathbf{u}_j^{(l)}). \quad (2)$$



sigmoid

$$f(z) = 1/(1 + e^{-z})$$

derivative

$$f'(z) = f(z)(1 - f(z))$$

tanh

$$f(z) = (e^z - e^{-z})/(e^z + e^{-z})$$

derivative

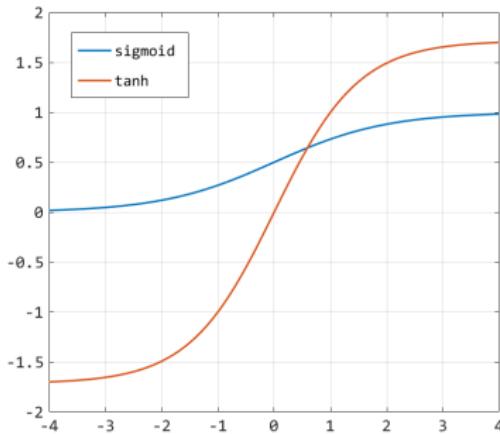
$$f'(z) = 1 - f^2(z).$$

Figure 7: 激活函数

# 非线性

卷积之后还需经过非线性激活函数，输出才得到下一层的特征图。常见的激活函数有sigmoid和tanh两种。

$$\mathbf{x}_j^{(l)} = f(\mathbf{u}_j^{(l)}). \quad (2)$$



sigmoid

$$f(z) = 1/(1 + e^{-z})$$

derivative

$$f'(z) = f(z)(1 - f(z))$$

tanh

$$f(z) = (e^z - e^{-z})/(e^z + e^{-z})$$

derivative

$$f'(z) = 1 - f^2(z).$$

Figure 7: 激活函数

# 池化

经过卷积获得了特征图(feature maps)之后，直接用特征连接到输出的分类器，会面临计算量的挑战。为了解决这个问题，采用一种对不同位置的特征聚合的操作，池化(pooling)或降采样(subsample)。池化的本质就是计算图像的(不重叠)区域上的特征的平均值(或最大值)，来降低提取到的特征的维度。池化有平均池化(mean pooling)和最大池化(max pooling)。

$$\mathbf{x}_j^{(l)} = f(\text{down}(\mathbf{x}_j^{(l-1)}) + \mathbf{b}_j^{(l)}), \text{ 一般取 } f(x) = x. \quad (3)$$

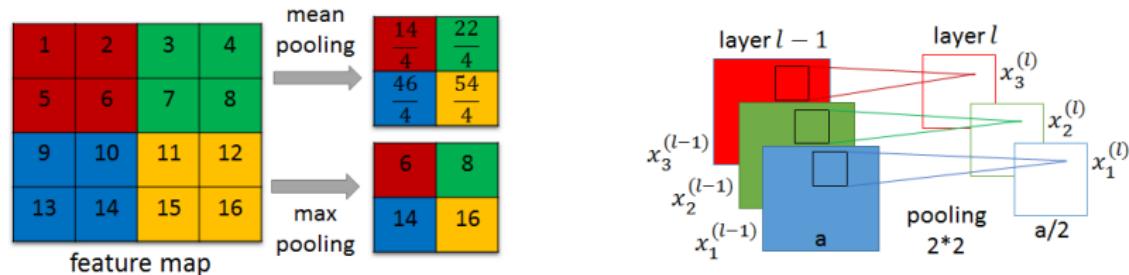
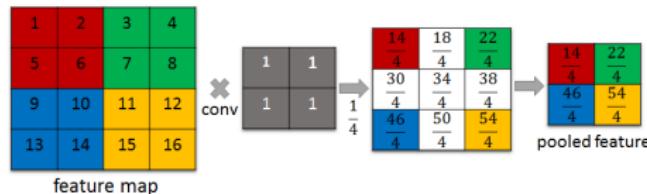


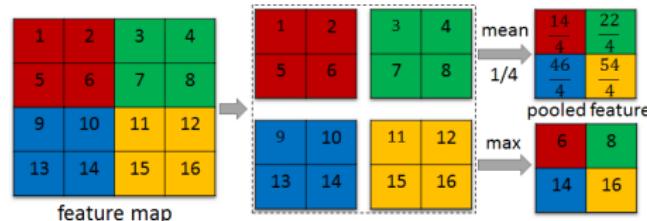
Figure 8: 池化 (降采样)

# 平均池化实现

论文[Bouvie, 2006]给出平均池化的实现方式：用一个全1的矩阵和图像卷积，接着除以核的大小，然后简单地通过等间隔的索引方法来采样得到池化的结果<sup>1</sup>。这种方式实现很简单，只需两行代码。



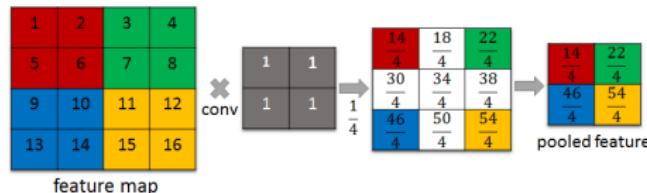
论文[Bouvie, 2006]也提及这明显浪费了一大部分的计算量。于是有一种修改方法是，直接先按间隔索引方式取出需要池化的块矩阵，再将它们取平均（或取最大）得到池化结果，这样的方式只有矩阵加法（没有乘法），而且没有浪费计算量。



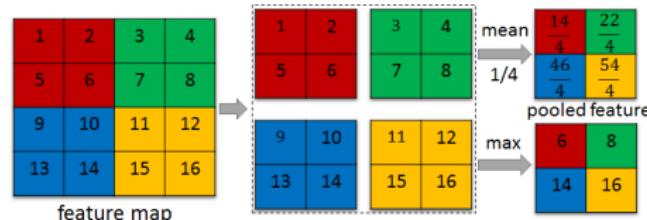
<sup>1</sup>Bouvie, Jake. "Notes on convolutional neural networks." (2006).

# 平均池化实现

论文[Bouvie, 2006]给出平均池化的实现方式：用一个全1的矩阵和图像卷积，接着除以核的大小，然后简单地通过等间隔的索引方法来采样得到池化的结果<sup>1</sup>。这种方式实现很简单，只需两行代码。



论文[Bouvie, 2006]也提及这明显浪费了一大部分的计算量。于是有一种修改方法是，直接先按间隔索引方式取出需要池化的块矩阵，再将它们取平均（或取最大）得到池化结果，这样的方式只有矩阵加法（没有乘法），而且没有浪费计算量。



<sup>1</sup>Bouvie, Jake. "Notes on convolutional neural networks." (2006).

# Outline

1 简介

2 卷积神经网络特性

3 前向传播

4 网络训练

5 应用

- LeNet
- MCDNN
- AlexNet
- DeepID
- Pyramid CNN
- Fast R-CNN

6 我的尝试

- STL-10
- CalTech101
- CIFAR-10
- 自编码算法

7 参考文献

# 训练参数

首先考虑一个简单的前馈神经网络，训练需要对每一层的权值和偏置参数进行更新。从输出层开始，首先定义代价函数（仅考虑单个输入样本）

$$J(\mathbf{y}) = \frac{1}{2} \| \mathbf{t} - \mathbf{y} \|^2 \quad (4)$$

$$\mathbf{y} = f(\mathbf{u}^{(2)}) = f(\mathbf{W}^{(2)}\mathbf{x}^{(1)} + \mathbf{b}^{(2)}) \quad (5)$$

$$\mathbf{x}^{(1)} = f(\mathbf{u}^{(1)}) = f(\mathbf{W}^{(1)}\mathbf{x}^{(0)} + \mathbf{b}^{(1)}). \quad (6)$$

其中  $\mathbf{t}$  代表标签（期望输出）， $(0), (1), (2)$  代表网络的层。

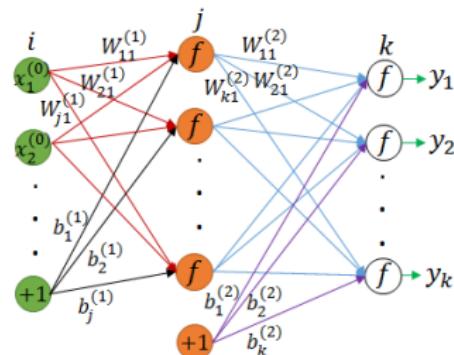


Figure 9: 一个简单的前馈神经网络

# 梯度下降法

我们需要使用梯度下降法更新权重  $\mathbf{W}$  和偏置  $\mathbf{b}$ ,

$$\mathbf{W}_{\tau+1} = \mathbf{W}_\tau - \eta \frac{\partial J}{\partial \mathbf{W}_\tau} = \mathbf{W}_\tau - \eta \nabla_{\mathbf{W}_\tau} J \quad (7)$$

$$\mathbf{b}_{\tau+1} = \mathbf{b}_\tau - \eta \frac{\partial J}{\partial \mathbf{b}_\tau} = \mathbf{b}_\tau - \eta \nabla_{\mathbf{b}_\tau} J. \quad (8)$$

其中  $\tau$  表示迭代次数。我们需求代价函数对参数的梯度  $\nabla_{\mathbf{W}} J, \nabla_{\mathbf{b}} J$ ,

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}^{(2)}} \frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{W}^{(2)}} = \underline{(\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)})} (\mathbf{x}^{(1)})^T \quad (9)$$

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}^{(2)}} \frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{x}^{(1)}} \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{u}^{(1)}} \frac{\partial \mathbf{u}^{(1)}}{\partial \mathbf{W}^{(1)}} = \underline{(\mathbf{W}^{(2)})^T (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)}) * f'(\mathbf{u}^{(1)})} (\mathbf{x}^{(0)})^T \quad (10)$$

$$\frac{\partial J}{\partial \mathbf{b}^{(2)}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}^{(2)}} \frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{b}^{(2)}} = \underline{(\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)})} \quad (11)$$

$$\frac{\partial J}{\partial \mathbf{b}^{(1)}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}^{(2)}} \frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{x}^{(1)}} \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{u}^{(1)}} \frac{\partial \mathbf{u}^{(1)}}{\partial \mathbf{b}^{(1)}} = \underline{(\mathbf{W}^{(2)})^T (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)}) * f'(\mathbf{u}^{(1)})}. \quad (12)$$

其中  $*$  表示 Hadamard 积，也就是对应元素乘积。

# 梯度下降法

我们需要使用梯度下降法更新权重  $\mathbf{W}$  和偏置  $\mathbf{b}$ ,

$$\mathbf{W}_{\tau+1} = \mathbf{W}_\tau - \eta \frac{\partial J}{\partial \mathbf{W}_\tau} = \mathbf{W}_\tau - \eta \nabla_{W_\tau} J \quad (7)$$

$$\mathbf{b}_{\tau+1} = \mathbf{b}_\tau - \eta \frac{\partial J}{\partial \mathbf{b}_\tau} = \mathbf{b}_\tau - \eta \nabla_{b_\tau} J. \quad (8)$$

其中  $\tau$  表示迭代次数。我们需求代价函数对参数的梯度  $\nabla_W J$ ,  $\nabla_b J$ ,

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}^{(2)}} \frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{W}^{(2)}} = \underline{(\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)}) (\mathbf{x}^{(1)})^T} \quad (9)$$

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}^{(2)}} \frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{x}^{(1)}} \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{u}^{(1)}} \frac{\partial \mathbf{u}^{(1)}}{\partial \mathbf{W}^{(1)}} = \underline{(\mathbf{W}^{(2)})^T (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)}) * f'(\mathbf{u}^{(1)}) (\mathbf{x}^{(0)})^T} \quad (10)$$

$$\frac{\partial J}{\partial \mathbf{b}^{(2)}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}^{(2)}} \frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{b}^{(2)}} = \underline{(\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)})} \quad (11)$$

$$\frac{\partial J}{\partial \mathbf{b}^{(1)}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}^{(2)}} \frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{x}^{(1)}} \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{u}^{(1)}} \frac{\partial \mathbf{u}^{(1)}}{\partial \mathbf{b}^{(1)}} = \underline{(\mathbf{W}^{(2)})^T (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)}) * f'(\mathbf{u}^{(1)})}. \quad (12)$$

其中  $*$  表示Hadamard积，也就是对应元素乘积。

# 误差反向传播

但是网络层数较多时，用链式求导得到前几层的梯度很繁杂，可采用误差反向传播算法（Error Backpropagation），得到非常简洁的梯度表达式。

## Definition

一个结点的误差是该节点对最终的期望值与输出激活值之差的影响度，或灵敏度（sensitivities）[Bouvie, 2006]，表达式为

$$\delta^{(l)} = \frac{\partial J}{\partial \mathbf{u}^{(l)}}. \quad (13)$$

回顾公式(9)-(12)，我们得到

$$\delta^{(2)} = \frac{\partial J}{\partial \mathbf{u}^{(2)}} = (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)}), \quad \delta^{(1)} = \frac{\partial J}{\partial \mathbf{u}^{(1)}} = (\mathbf{W}^{(2)})^T \delta^{(2)} * f'(\mathbf{u}^{(1)}) \quad (14)$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \delta^{(2)} (\mathbf{x}^{(1)})^T, \quad \frac{\partial J}{\partial \mathbf{W}^{(1)}} = \delta^{(1)} (\mathbf{x}^{(0)})^T, \quad \frac{\partial J}{\partial \mathbf{b}^{(2)}} = \delta^{(2)}, \quad \frac{\partial J}{\partial \mathbf{b}^{(1)}} = \delta^{(1)}. \quad (15)$$

$$\delta^{(L)} = (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(L)}) \quad (16)$$

$$\delta^{(l)} = (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} * f'(\mathbf{u}^{(l)}), \quad l = 1, \dots, L-1. \quad (17)$$

# 误差反向传播

但是网络层数较多时，用链式求导得到前几层的梯度很繁杂，可采用误差反向传播算法（Error Backpropagation），得到非常简洁的梯度表达式。

## Definition

一个结点的误差是该节点对最终的期望值与输出激活值之差的影响度，或灵敏度（sensitivities）[Bouvie, 2006]，表达式为

$$\boldsymbol{\delta}^{(l)} = \frac{\partial J}{\partial \mathbf{u}^{(l)}}. \quad (13)$$

回顾公式(9)-(12)，我们得到

$$\boldsymbol{\delta}^{(2)} = \frac{\partial J}{\partial \mathbf{u}^{(2)}} = (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)}), \quad \boldsymbol{\delta}^{(1)} = \frac{\partial J}{\partial \mathbf{u}^{(1)}} = (\mathbf{W}^{(2)})^T \boldsymbol{\delta}^{(2)} * f'(\mathbf{u}^{(1)}) \quad (14)$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \boldsymbol{\delta}^{(2)} (\mathbf{x}^{(1)})^T, \quad \frac{\partial J}{\partial \mathbf{W}^{(1)}} = \boldsymbol{\delta}^{(1)} (\mathbf{x}^{(0)})^T, \quad \frac{\partial J}{\partial \mathbf{b}^{(2)}} = \boldsymbol{\delta}^{(2)}, \quad \frac{\partial J}{\partial \mathbf{b}^{(1)}} = \boldsymbol{\delta}^{(1)}. \quad (15)$$

$$\boldsymbol{\delta}^{(L)} = (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(L)}) \quad (16)$$

$$\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} * f'(\mathbf{u}^{(l)}), \quad l = 1, \dots, L-1. \quad (17)$$

# 误差反向传播

但是网络层数较多时，用链式求导得到前几层的梯度很繁杂，可采用误差反向传播算法（Error Backpropagation），得到非常简洁的梯度表达式。

## Definition

一个结点的误差是该节点对最终的期望值与输出激活值之差的影响度，或灵敏度（sensitivities）[Bouvie, 2006]，表达式为

$$\boldsymbol{\delta}^{(l)} = \frac{\partial J}{\partial \mathbf{u}^{(l)}}. \quad (13)$$

回顾公式(9)-(12)，我们得到

$$\boldsymbol{\delta}^{(2)} = \frac{\partial J}{\partial \mathbf{u}^{(2)}} = (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(2)}), \quad \boldsymbol{\delta}^{(1)} = \frac{\partial J}{\partial \mathbf{u}^{(1)}} = (\mathbf{W}^{(2)})^T \boldsymbol{\delta}^{(2)} * f'(\mathbf{u}^{(1)}) \quad (14)$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \boldsymbol{\delta}^{(2)} (\mathbf{x}^{(1)})^T, \quad \frac{\partial J}{\partial \mathbf{W}^{(1)}} = \boldsymbol{\delta}^{(1)} (\mathbf{x}^{(0)})^T, \quad \frac{\partial J}{\partial \mathbf{b}^{(2)}} = \boldsymbol{\delta}^{(2)}, \quad \frac{\partial J}{\partial \mathbf{b}^{(1)}} = \boldsymbol{\delta}^{(1)}. \quad (15)$$

$$\boldsymbol{\delta}^{(L)} = (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(L)}) \quad (16)$$

$$\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} * f'(\mathbf{u}^{(l)}), \quad l = 1, \dots, L-1. \quad (17)$$

# 误差反向传播

整理一下训练网络的思路,

$$\delta^{(l)} \Rightarrow \frac{\partial J}{\partial \mathbf{W}^{(l)}}, \frac{\partial J}{\partial \mathbf{b}^{(l)}} \Rightarrow \text{梯度下降法, 更新 } \mathbf{W}^{(l)}, \mathbf{b}^{(l)}$$

$\Rightarrow$  前向传播, 计算  $J(\mathbf{y}) \Rightarrow$  是否收敛  $\Delta J(\mathbf{y}) \approx 0 \Rightarrow$  迭代 ...

因为卷积神经网络的结构中, 层间的结点的连接与前馈神经网络不同。但是思路是一致的, 所以我们的关键是计算出  $\delta^{(l)}$ 。

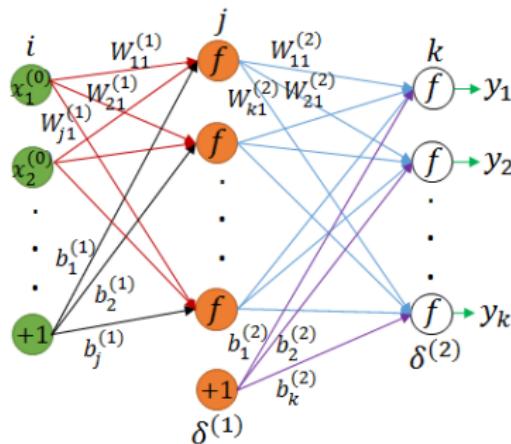


Figure 10: 前馈神经网络

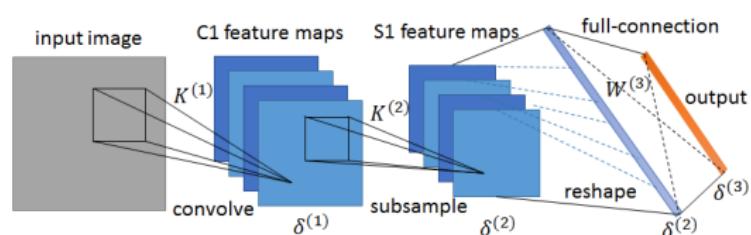


Figure 11: 卷积神经网络

# 池化层 $\leftarrow$ 全连接层

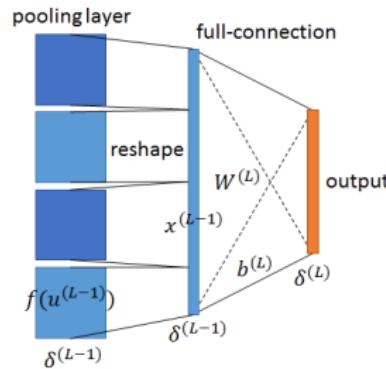


Figure 12: 全连接层的误差反向传播

该层的权值连接与之前讨论的 Figure 10 前馈神经网络一致，所以误差的反向传播公式直接得到，

$$\delta^{(L)} = (y - t) * f'(u^{(L)}) \quad (18)$$

$$\frac{\partial J}{\partial \mathbf{W}^{(L)}} = \delta^{(L)} (\mathbf{x}^{(L-1)})^T, \quad \frac{\partial J}{\partial \mathbf{b}^{(L)}} = \delta^{(L)} \quad (19)$$

$$\delta^{(L-1)} = (\mathbf{W}^{(L)})^T \delta^{(L)} * f'(u^{(L-1)}). \quad (20)$$

# 池化层 ← 全连接层

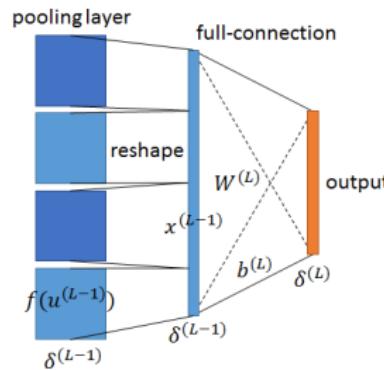


Figure 12: 全连接层的误差反向传播

该层的权值连接与之前讨论的 Figure 10 前馈神经网络一致，所以误差的反向传播公式直接得到，

$$\boldsymbol{\delta}^{(L)} = (\mathbf{y} - \mathbf{t}) * f'(\mathbf{u}^{(L)}) \quad (18)$$

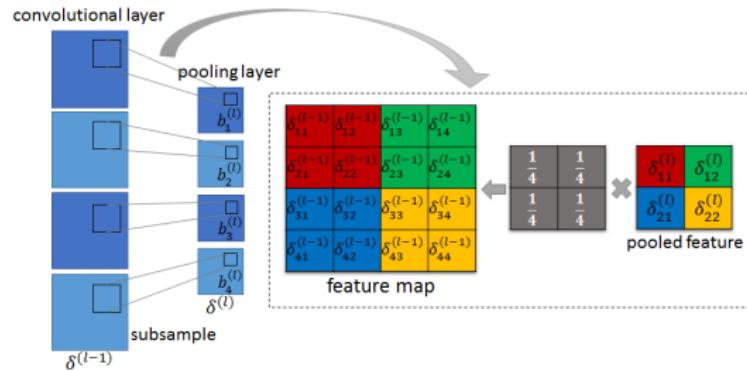
$$\frac{\partial J}{\partial \mathbf{W}^{(L)}} = \boldsymbol{\delta}^{(L)} (\mathbf{x}^{(L-1)})^T, \quad \frac{\partial J}{\partial \mathbf{b}^{(L)}} = \boldsymbol{\delta}^{(L)} \quad (19)$$

$$\boldsymbol{\delta}^{(L-1)} = (\mathbf{W}^{(L)})^T \boldsymbol{\delta}^{(L)} * f'(\mathbf{u}^{(L-1)}). \quad (20)$$

# 卷积层 ← 池化层

已经得到池化层的误差  $\delta^{(l)}$ , 但池化层的没有可调整的权值, 只有

$$\frac{\partial J}{\partial b_j^{(l)}} = \sum_{u,v} (\delta_j^{(l)})_{uv}. \quad (21)$$



平均池化操作是按块区域取平均值, 反向传播相当于逆过程, 将误差扩展成矩阵, 除以扩展的大小。

Figure 13: 池化层的误差反向传播

$$\delta^{(l-1)} = up(\delta^{(l)}) * f'(u^{(l-1)}) \quad (22)$$

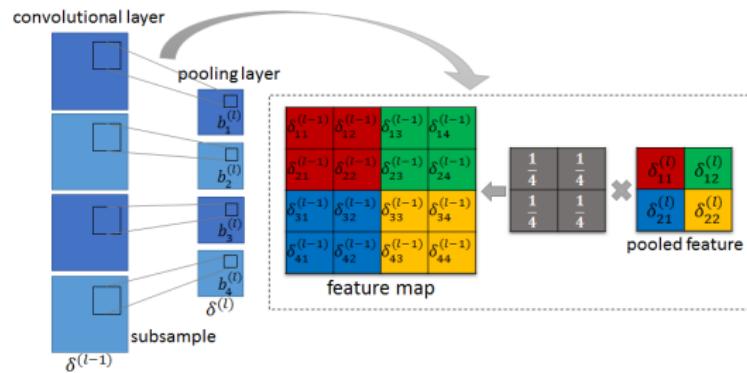
$$up(x) = x \otimes \mathbf{1}_{n \times n} / n^2. \quad (23)$$

其中  $\otimes$  表示 Kronecker 积。

# 卷积层 ← 池化层

已经得到池化层的误差  $\delta^{(l)}$ , 但池化层的没有可调整的权值, 只有

$$\frac{\partial J}{\partial b_j^{(l)}} = \sum_{u,v} (\delta_j^{(l)})_{uv}. \quad (21)$$



平均池化操作是按块区域取平均值, 反向传播相当于逆过程, 将误差扩展成矩阵, 除以扩展的大小。

Figure 13: 池化层的误差反向传播

$$\delta^{(l-1)} = up(\delta^{(l)}) * f'(u^{(l-1)}) \quad (22)$$

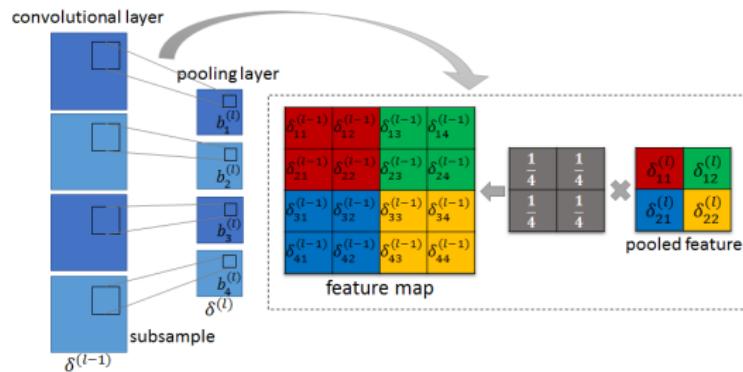
$$up(x) = x \otimes \mathbf{1}_{n \times n} / n^2. \quad (23)$$

其中  $\otimes$  表示 Kronecker 积。

# 卷积层 ← 池化层

已经得到池化层的误差  $\delta^{(l)}$ , 但池化层的没有可调整的权值, 只有

$$\frac{\partial J}{\partial b_j^{(l)}} = \sum_{u,v} (\delta_j^{(l)})_{uv}. \quad (21)$$



平均池化操作是按块区域取平均值, 反向传播相当于逆过程, 将误差扩展成矩阵, 除以扩展的大小。

Figure 13: 池化层的误差反向传播

$$\delta^{(l-1)} = up(\delta^{(l)}) * f'(\mathbf{u}^{(l-1)}) \quad (22)$$

$$up(\mathbf{x}) = \mathbf{x} \otimes \mathbf{1}_{n \times n} / n^2. \quad (23)$$

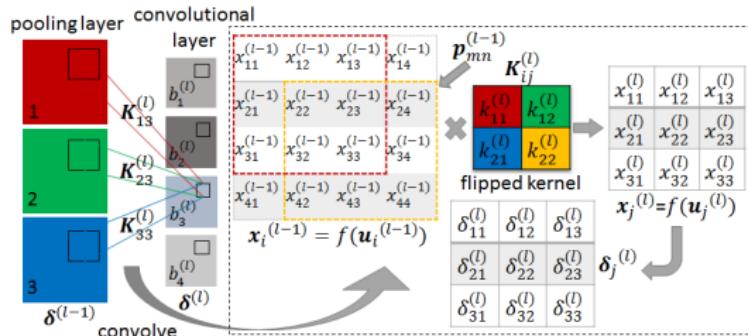
其中  $\otimes$  表示 Kronecker 积。

# 池化层 ← 卷积层

已经得到卷积层的误差  $\delta^{(l)}$ , 可调整的权值和偏置的梯度表达式如下

$$\frac{\partial J}{\partial \mathbf{K}_{ij}^{(l)}} = \text{rot180}(\mathbf{x}_i^{(l-1)} \bullet \text{rot180}(\delta_j^{(l)})), \quad \frac{\partial J}{\partial b_j^{(l)}} = \sum_{u,v} (\delta_j^{(l)})_{uv}. \quad (24)$$

用单个元素的形式表示, 更易于理解: 权值的梯度等于当前层的误差  $\delta_j^{(l)}$  与以前一层与该权值连接的部分结点的输出  $p_{mn}^{(l-1)}$  对应元素相乘之和,



$$\begin{aligned} \frac{\partial J}{\partial k_{mn}^{(l)}} &= \sum_{u,v} (\delta_j^{(l)})_{uv} (p_{mn}^{(l-1)})_{uv} \\ &= \sum_u^3 \sum_v^3 \delta_{uv}^{(l)} x_{m+u-1, n+v-1}^{(l-1)} \end{aligned}$$

$$x_{11}^{(l)} = f(k_{11}^{(l)} x_{11}^{(l-1)} + k_{12}^{(l)} x_{12}^{(l-1)} + k_{21}^{(l)} x_{21}^{(l-1)} + k_{22}^{(l)} x_{22}^{(l-1)} + b_{11}^{(l)}) \quad (25)$$

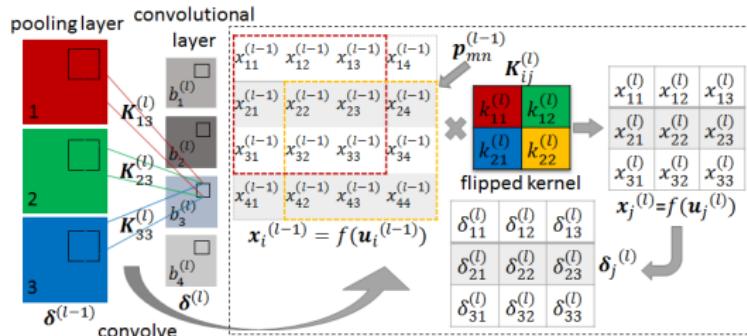
$$x_{33}^{(l)} = f(k_{11}^{(l)} x_{33}^{(l-1)} + k_{12}^{(l)} x_{34}^{(l-1)} + k_{21}^{(l)} x_{43}^{(l-1)} + k_{22}^{(l)} x_{44}^{(l-1)} + b_{33}^{(l)}). \quad (26)$$

# 池化层 ← 卷积层

已经得到卷积层的误差  $\delta^{(l)}$ , 可调整的权值和偏置的梯度表达式如下

$$\frac{\partial J}{\partial \mathbf{K}_{ij}^{(l)}} = \text{rot180}(\mathbf{x}_i^{(l-1)} \bullet \text{rot180}(\delta_j^{(l)})), \quad \frac{\partial J}{\partial b_j^{(l)}} = \sum_{u,v} (\delta_j^{(l)})_{uv}. \quad (24)$$

用单个元素的形式表示, 更易于理解: 权值的梯度等于当前层的误差  $\delta_j^{(l)}$  与以前一层与该权值连接的部分结点的输出  $\mathbf{p}_{mn}^{(l-1)}$  对应元素相乘之和,



$$\begin{aligned} \frac{\partial J}{\partial k_{mn}^{(l)}} &= \sum_{u,v} (\delta_j^{(l)})_{uv} (\mathbf{p}_{mn}^{(l-1)})_{uv} \\ &= \sum_u^3 \sum_v^3 \delta_{uv}^{(l)} x_{m+u-1, n+v-1}^{(l-1)}. \end{aligned}$$

$$x_{11}^{(l)} = f(k_{11}^{(l)} x_{11}^{(l-1)} + k_{12}^{(l)} x_{12}^{(l-1)} + k_{21}^{(l)} x_{21}^{(l-1)} + k_{22}^{(l)} x_{22}^{(l-1)} + b_{11}^{(l)}) \quad (25)$$

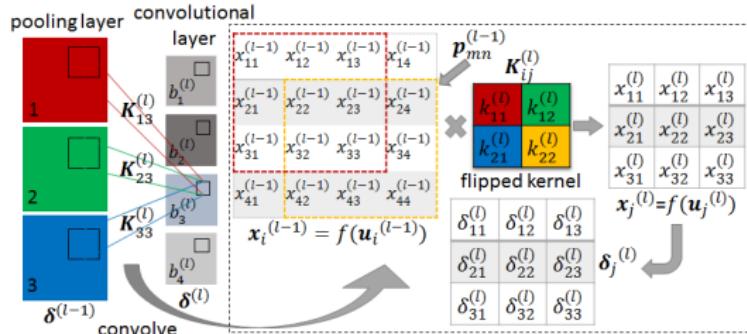
$$\vdots \qquad \vdots \\ x_{33}^{(l)} = f(k_{11}^{(l)} x_{33}^{(l-1)} + k_{12}^{(l)} x_{34}^{(l-1)} + k_{21}^{(l)} x_{43}^{(l-1)} + k_{22}^{(l)} x_{44}^{(l-1)} + b_{33}^{(l)}). \quad (26)$$

# 池化层 ← 卷积层

已经得到卷积层的误差  $\delta^{(l)}$ , 可调整的权值和偏置的梯度表达式如下

$$\frac{\partial J}{\partial \mathbf{K}_{ij}^{(l)}} = \text{rot180}(\mathbf{x}_i^{(l-1)} \bullet \text{rot180}(\delta_j^{(l)})), \quad \frac{\partial J}{\partial b_j^{(l)}} = \sum_{u,v} (\delta_j^{(l)})_{uv}. \quad (24)$$

用单个元素的形式表示, 更易于理解: 权值的梯度等于当前层的误差  $\delta_j^{(l)}$  与以前一层与该权值连接的部分结点的输出  $\mathbf{p}_{mn}^{(l-1)}$  对应元素相乘之和,



$$\begin{aligned} \frac{\partial J}{\partial k_{mn}^{(l)}} &= \sum_{u,v} (\delta_j^{(l)})_{uv} (\mathbf{p}_{mn}^{(l-1)})_{uv} \\ &= \sum_u^3 \sum_v^3 \delta_{uv}^{(l)} x_{m+u-1, n+v-1}^{(l-1)}. \end{aligned}$$

$$x_{11}^{(l)} = f(k_{11}^{(l)} x_{11}^{(l-1)} + k_{12}^{(l)} x_{12}^{(l-1)} + k_{21}^{(l)} x_{21}^{(l-1)} + k_{22}^{(l)} x_{22}^{(l-1)} + b_{11}^{(l)}) \quad (25)$$

$$\vdots \qquad \vdots$$

$$x_{33}^{(l)} = f(k_{11}^{(l)} x_{33}^{(l-1)} + k_{12}^{(l)} x_{34}^{(l-1)} + k_{21}^{(l)} x_{43}^{(l-1)} + k_{22}^{(l)} x_{44}^{(l-1)} + b_{33}^{(l)}). \quad (26)$$

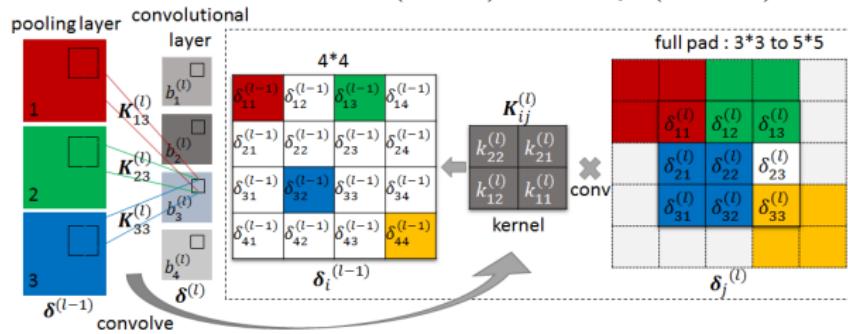
# 池化层 ← 卷积层

将当前层结点的误差沿着其连接的权值传递至前一层对应的结点，

$$\delta_i^{(l-1)} = (\text{rot180}(\mathbf{K}_{ij}^{(l)}) \bullet \delta_j^{(l)}) * f'(\mathbf{u}_i^{(l-1)}). \quad (27)$$

其中  $\bullet$  表示全卷积操作。形式类似  $\delta^{(l-1)} = (\mathbf{W}^{(l)})^T \delta^{(l)} * f'(\mathbf{u}^{(l-1)})$ ，

用单个元素形式表示，更易于理解，



$$x_{11}^{(l)} = f(k_{11}^{(l)} x_{11}^{(l-1)} + \dots) \Rightarrow \delta_{11}^{(l-1)} = k_{11}^{(l)} \delta_{11}^{(l)} f'(u_{11}^{(l-1)}).$$

$$x_{12}^{(l)} = f(k_{12}^{(l)} x_{13}^{(l-1)} + \dots),$$

$$x_{13}^{(l)} = f(k_{11}^{(l)} x_{13}^{(l-1)} + \dots) \Rightarrow \delta_{13}^{(l-1)} = (k_{11}^{(l)} \delta_{13}^{(l)} + k_{12}^{(l)} \delta_{12}^{(l)}) f'(u_{13}^{(l-1)}).$$

$$x_{21}^{(l)} = f(k_{22}^{(l)} x_{32}^{(l-1)} + \dots),$$

$$x_{22}^{(l)} = f(k_{21}^{(l)} x_{32}^{(l-1)} + \dots), \Rightarrow \delta_{32}^{(l-1)} = (k_{11}^{(l)} \delta_{32}^{(l)} + k_{12}^{(l)} \delta_{31}^{(l)}$$

$$x_{31}^{(l)} = f(k_{12}^{(l)} x_{32}^{(l-1)} + \dots), \quad + k_{21}^{(l)} \delta_{22}^{(l)} + k_{22}^{(l)} \delta_{21}^{(l)}) f'(u_{32}^{(l-1)}).$$

$$x_{32}^{(l)} = f(k_{11}^{(l)} x_{32}^{(l-1)} + \dots)$$

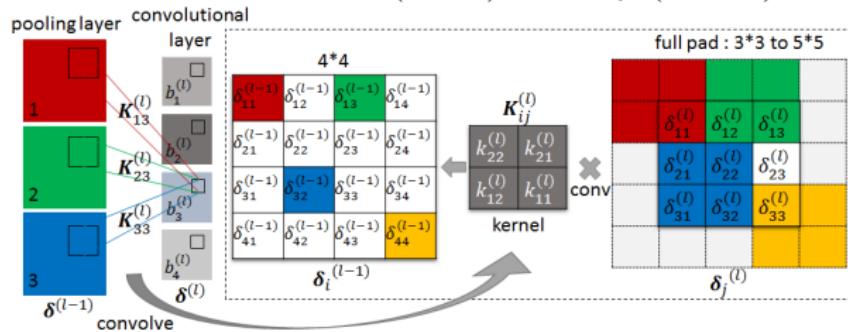
# 池化层 ← 卷积层

将当前层结点的误差沿着其连接的权值传递至前一层对应的结点，

$$\delta_i^{(l-1)} = (\text{rot180}(\mathbf{K}_{ij}^{(l)}) \bullet \delta_j^{(l)}) * f'(\mathbf{u}_i^{(l-1)}). \quad (27)$$

其中  $\bullet$  表示全卷积操作。形式类似  $\delta^{(l-1)} = (\mathbf{W}^{(l)})^T \delta^{(l)} * f'(\mathbf{u}^{(l-1)})$ ，

用单个元素形式表示，更易于理解，



$$x_{11}^{(l)} = f(k_{11}^{(l)} x_{11}^{(l-1)} + \dots) \Rightarrow \delta_{11}^{(l-1)} = k_{11}^{(l)} \delta_{11}^{(l)} f'(u_{11}^{(l-1)}).$$

$$x_{12}^{(l)} = f(k_{12}^{(l)} x_{13}^{(l-1)} + \dots),$$

$$x_{13}^{(l)} = f(k_{11}^{(l)} x_{13}^{(l-1)} + \dots) \Rightarrow \delta_{13}^{(l-1)} = (k_{11}^{(l)} \delta_{13}^{(l)} + k_{12}^{(l)} \delta_{12}^{(l)}) f'(u_{13}^{(l-1)}).$$

$$x_{21}^{(l)} = f(k_{22}^{(l)} x_{32}^{(l-1)} + \dots),$$

$$x_{22}^{(l)} = f(k_{21}^{(l)} x_{32}^{(l-1)} + \dots), \Rightarrow \delta_{32}^{(l-1)} = (k_{11}^{(l)} \delta_{32}^{(l)} + k_{12}^{(l)} \delta_{31}^{(l)}$$

$$x_{31}^{(l)} = f(k_{12}^{(l)} x_{32}^{(l-1)} + \dots), \quad + k_{21}^{(l)} \delta_{22}^{(l)} + k_{22}^{(l)} \delta_{21}^{(l)}) f'(u_{32}^{(l-1)}).$$

$$x_{32}^{(l)} = f(k_{11}^{(l)} x_{32}^{(l-1)} + \dots)$$

# 训练步骤

至此，反向传播训练需要的梯度参数都已经获得，

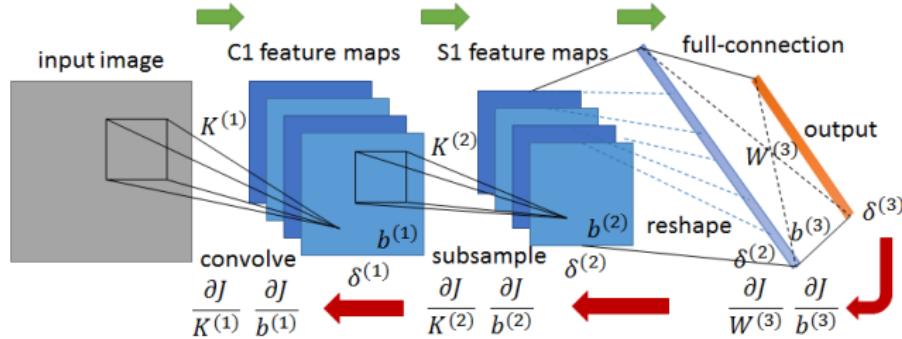


Figure 14: 卷积神经网络训练

首先给每一层的权重  $\mathbf{K}^{(l)}$ ,  $\mathbf{W}^{(l)}$  和偏置  $\mathbf{b}^{(l)}$  赋予随机的初始值。前向传播一次，反向传播一次，计算  $\delta^{(l)}$ ，使用  $\frac{\partial J}{\partial \mathbf{K}^{(l)}}$ ,  $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ ,  $\frac{\partial J}{\partial \mathbf{b}^{(l)}}$  梯度更新每一层的权重  $\mathbf{K}^{(l)}$ ,  $\mathbf{W}^{(l)}$  和偏置  $\mathbf{b}^{(l)}$ ，如此循环，直到  $J(\mathbf{y})$  收敛，或算法达到最大迭代次数，停止训练。

# 训练步骤

至此，反向传播训练需要的梯度参数都已经获得，

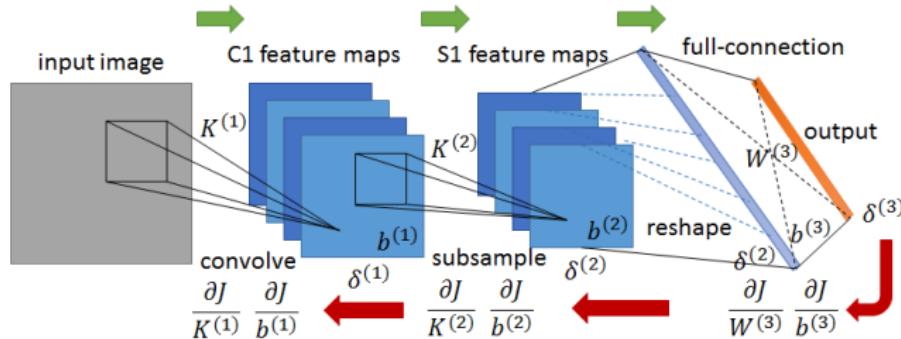


Figure 14: 卷积神经网络训练

首先给每一层的权重  $\mathbf{K}^{(l)}$ ,  $\mathbf{W}^{(l)}$  和偏置  $\mathbf{b}^{(l)}$  赋予随机的初始值。前向传播一次，反向传播一次，计算  $\delta^{(l)}$ ，使用  $\frac{\partial J}{\partial \mathbf{K}^{(l)}}$ ,  $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ ,  $\frac{\partial J}{\partial \mathbf{b}^{(l)}}$  梯度更新每一层的权重  $\mathbf{K}^{(l)}$ ,  $\mathbf{W}^{(l)}$  和偏置  $\mathbf{b}^{(l)}$ ，如此循环，直到  $J(\mathbf{y})$  收敛，或算法达到最大迭代次数，停止训练。

# Outline

1 简介

2 卷积神经网络特性

3 前向传播

4 网络训练

5 应用

- LeNet
- MCDNN
- AlexNet
- DeepID
- Pyramid CNN
- Fast R-CNN

6 我的尝试

- STL-10
- CalTech101
- CIFAR-10
- 自编码算法

7 参考文献

# LeNet-5: 分类

LeNet-5<sup>2</sup>, 是[LeCun et al., 1995][LeCun et al., 1998]设计用于识别手写数字字符的卷积神经网络模型。之前美国大多数银行使用它来识别支票上的手写数字，可见它的准确率已经相当高。

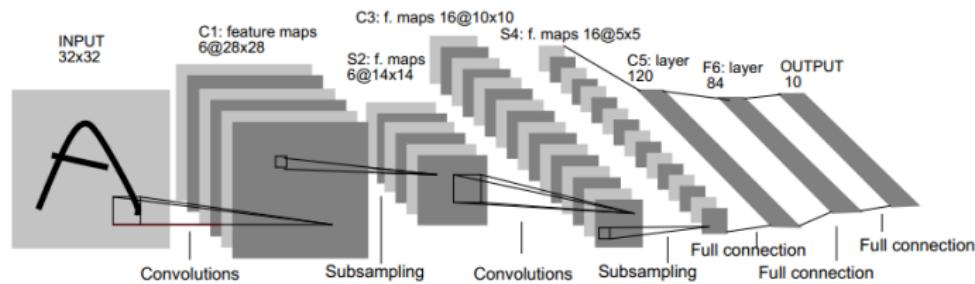


Figure 15: LeNet-5结构模型

该模型使用MNIST数据集，这是一个手写数字0-9的灰度图像集合，一共含有60000张训练样本，10000张测试样本。图片的大小都统一，且数字置于图片中心位置。

<sup>2</sup><http://yann.lecun.com/exdb/lenet/index.html>

# LeNet-5: 分类

LeNet-5<sup>2</sup>, 是[LeCun et al., 1995][LeCun et al., 1998]设计用于识别手写数字字符的卷积神经网络模型。之前美国大多数银行使用它来识别支票上的手写数字，可见它的准确率已经相当高。

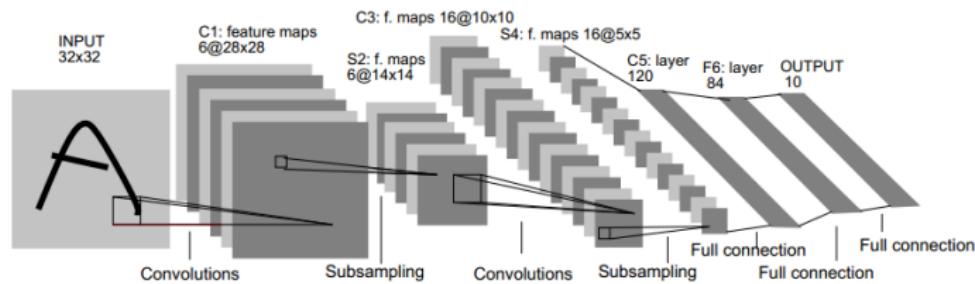


Figure 15: LeNet-5结构模型

该模型使用MNIST数据集，这是一个手写数字0-9的灰度图像集合，一共含有60000张训练样本，10000张测试样本。图片的大小都统一，且数字置于图片中心位置。

<sup>2</sup><http://yann.lecun.com/exdb/lenet/index.html>

# MNIST手写数据集

MNIST手写数据集<sup>3</sup>由Yann LeCun et al. 维护，已经被广泛用于测试图像分类算法的性能。[LeCun et al., 1998] 的准确率是99.3%，而且[Ciresan et al., 2012]已经达到最高的准确率是99.77%，已经能和人眼的识别率相匹配。

|                     |       |   |
|---------------------|-------|---|
| Linear Classifiers  | 7.6%  | LeCun et al., 1998                          |
| K-Nearest Neighbors | 0.63% | Belongie et al., IEEE PAMI 2002             |
| Boosted Stumps      | 0.87% | Kegl et al., ICML 2009                      |
| SVMs                | 0.56% | DeCoste and Scholkopf, MLJ 2002             |
| Neural Nets         | 0.35% | Ciresan et al., Neural Computation 10, 2010 |
| Convolutional nets  | 0.7%  | LeCun et al., 1998                          |
| Convolutional nets  | 0.23% | Ciresan et al., CVPR 2012                   |

Table 1: Performance of algorithms on MNIST database

<sup>3</sup><http://yann.lecun.com/exdb/mnist/>

# LeNet-5:结构

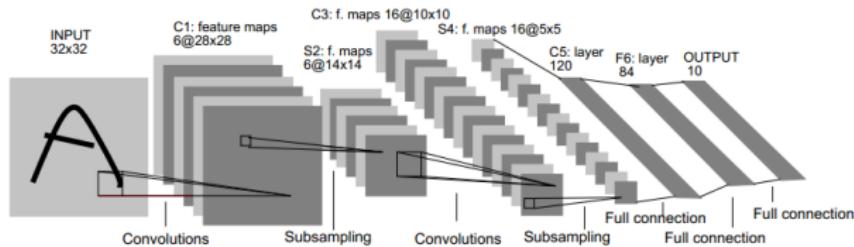


Figure 16: LeNet-5结构模型

第一个卷积层一共有  $5 \times 5 \times 6 + 6 = 156$  个参数需要调整。第二个卷积层一共有  $5 \times 5 \times 60 + 16 = 1516$  个参数需要调整[LeCun et al., 1998]。

Table 1 Each Column Indicates Which Feature Map in S2 Are Combined by the Units in a Particular Feature Map of C3

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 0 | X |   |   | X | X | X |   |   | X | X | X  | X  |    | X  | X  |    |
| 1 | X | X |   |   | X | X | X |   |   | X | X  | X  | X  |    | X  |    |
| 2 | X | X | X |   |   | X | X | X |   | X |    | X  | X  | X  |    |    |
| 3 |   | X | X | X |   |   | X | X | X | X |    | X  |    | X  | X  |    |
| 4 |   | X | X | X |   |   | X | X | X | X |    | X  | X  |    | X  |    |
| 5 |   |   | X | X | X |   |   | X | X | X | X  |    | X  | X  | X  |    |

论文[LeCun et al., 1998]说明，使用这种特征组合的原因：部分的组合打破了网络的对称性，强迫特征图学习到不同的特征，更利于分类。

# LeNet-5:结构

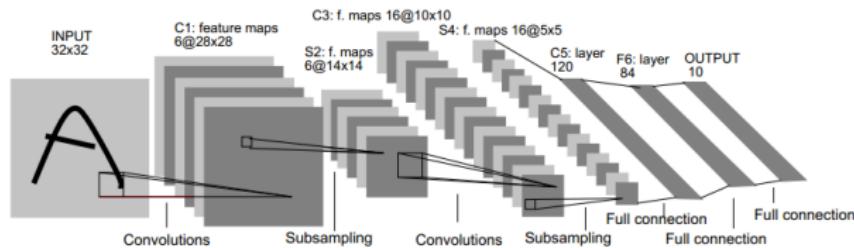


Figure 16: LeNet-5结构模型

第一个卷积层一共有  $5 \times 5 \times 6 + 6 = 156$  个参数需要调整。第二个卷积层一共有  $5 \times 5 \times 60 + 16 = 1516$  个参数需要调整[LeCun et al., 1998]。

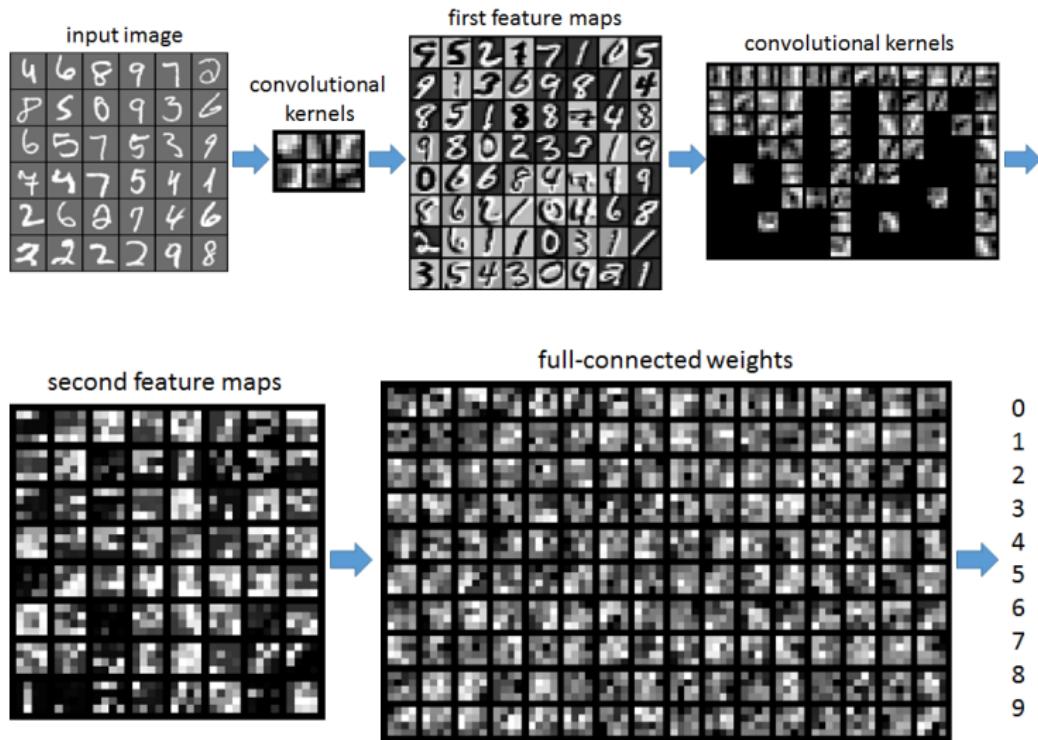
Table 1 Each Column Indicates Which Feature Map in S2 Are Combined by the Units in a Particular Feature Map of C3

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 0 | X |   |   | X | X | X |   |   | X | X | X  | X  | X  | X  | X  |    |
| 1 | X | X |   |   | X | X | X |   |   | X | X  | X  | X  | X  |    |    |
| 2 | X | X | X |   |   | X | X | X |   | X |    | X  | X  | X  |    |    |
| 3 |   | X | X | X |   |   | X | X | X | X |    | X  | X  | X  |    |    |
| 4 |   | X | X | X |   |   | X | X | X | X |    | X  | X  | X  |    |    |
| 5 |   |   | X | X | X |   |   | X | X | X | X  |    | X  | X  | X  |    |

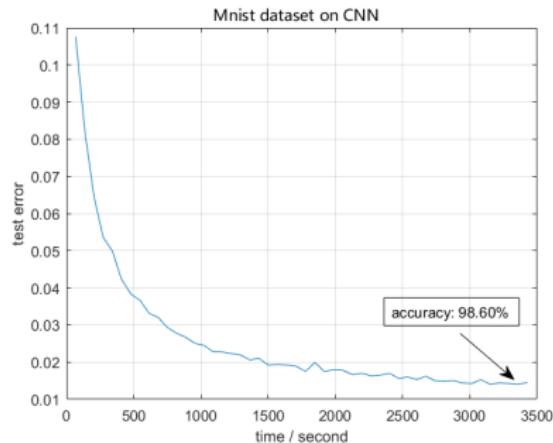
论文[LeCun et al., 1998]说明，使用这种特征组合的原因：部分的组合打破了网络的对称性，强迫特征图学习到不同的特征，更利于分类。

# LeNet-5：网络可视化

将LeNet网络的输入、卷积核、池化后的特征图、全连接权值都显示出来，



# LeNet-5：网络可视化



经过了50次迭代之后，网络模型对手写数字的分类准确率达到了98.6%。

▶ Reference

# MCDNN

2012年, [Ciresan et al., 2012]在CVPR上提出了MCDNN(Multi-column Deep Neural Network)结构, 并在各个数据集上进行测试, 给出了非常优秀的性能, 都超越了之前最佳的成果。

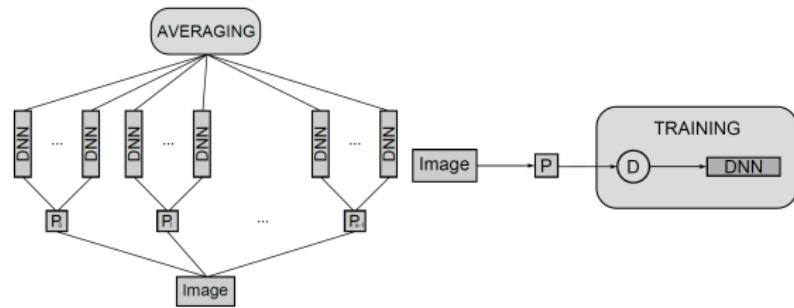
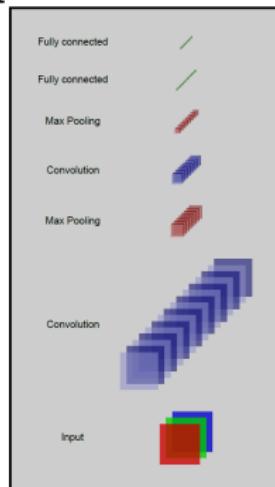
其中将MNIST数据集的最佳误差率从0.39%降低到了0.23%, CIFAR-10数据集从18.5%降低到了11.21%。

| Dataset       | Best result<br>of others [%] | MCDNN<br>[%] | Relative<br>improv. [%] |
|---------------|------------------------------|--------------|-------------------------|
| MNIST         | 0.39                         | 0.23         | 41                      |
| NIST SD 19    | see Table 4                  | see Table 4  | 30-80                   |
| HWDB1.0 on.   | 7.61                         | 5.61         | 26                      |
| HWDB1.0 off.  | 10.01                        | 6.5          | 35                      |
| CIFAR10       | 18.50                        | 11.21        | 39                      |
| traffic signs | 1.69                         | 0.54         | 72                      |
| NORB          | 5.00                         | 2.70         | 46                      |

Figure 17: 不同数据集的测试比较结果

# MCDNN

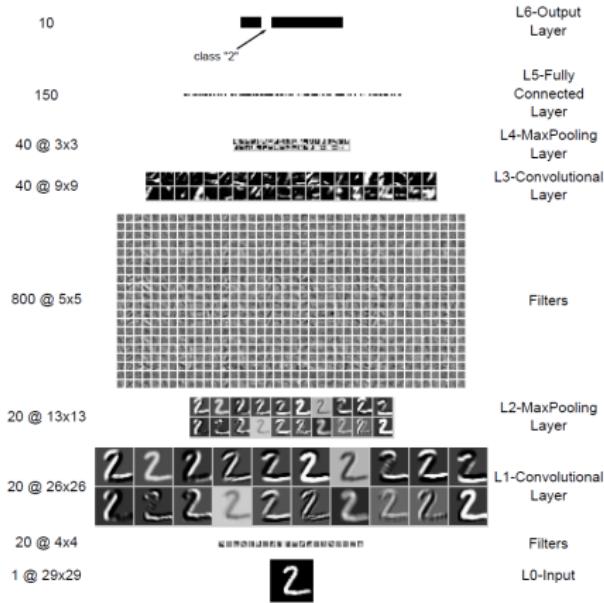
论文[Ciresan et al., 2012]中的思想如下，主要是结构的创新



构造多个结构完全一致的卷积网络模型，但是各自的输入不一致。输入图像经过预处理（preprocess）、变形（distort）作为每一个模型的训练图像。最后将输出的结果求和取平均得到预测结果。

# MCDNN

将Multi-column CNN模型用于测试MNIST数据集，论文[Ciresan et al., 2012]设计了如下的卷积网络模型， $1 \times 29 \times 29 - 20C4 - MP2 - 40C5 - MP3 - 150N - 10N$ 。



- 不经过预训练，权值随机初始化
- 卷积层激活函数tanh
- 池化层激活函数线性
- softmax作为分类器
- 递减的学习率
- 输入图像的数字被预处理成7种不同的高宽比 $20:20, 20:18, 20:16$ 等
- 输入图像经过5种不同的变形

# MCDNN

对于每一个卷积网络模型，训练800次迭代，初始学习率为0.001。经过500次迭代开始有效果体现，一共耗时14 小时。5种变形的效果如下

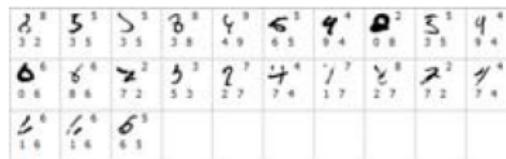


最后获得的测试结果如下

Table 2. Results on MNIST dataset.

| Method        | Paper | Error rate[%] |
|---------------|-------|---------------|
| CNN           | [32]  | 0.40          |
| CNN           | [26]  | 0.39          |
| MLP           | [5]   | 0.35          |
| CNN committee | [6]   | 0.27          |
| MCDNN         | this  | <b>0.23</b>   |

1万张测试图像中只剩下23张未能正确识别，论文[Ciresan et al., 2012]说明这几个数字包含着断的、奇怪的笔画或错误的标签。



# AlexNet

[Krizhevsky et al., 2012]和他的导师Geoffery E. Hinton用ImageNet LSVRC-2010的120万张图像用于分类1000个类别，训练深度卷积神经网络，获得了超过之前顶尖的误差率。在2012年提出了AlexNet 模型，正式参加ImageNet LSVRC-2012<sup>4</sup> 比赛，的确获得了第一名，让深度学习被广泛认可。

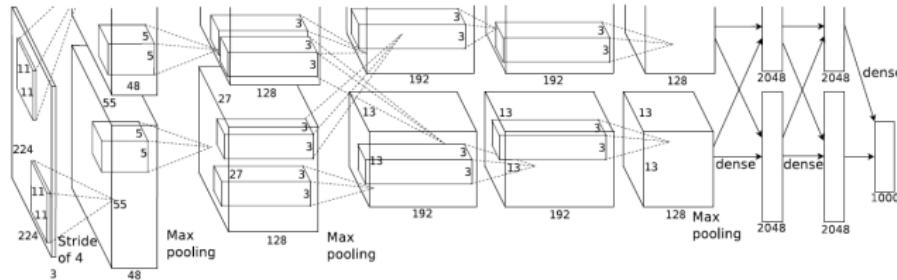


Figure 18: AlexNet 卷积神经网络模型

其大致结构是用6000万个参数和65万个神经元，由5个卷积、3个最大池化层以及3个全连接层加上1000类的softmax分类器构成。

<sup>4</sup><http://image-net.org/challenges/LSVRC/2012/index>

# AlexNet

论文[Krizhevsky et al., 2012]为了加快训练速度和提高分类准确率，设计如下网络特性：

- ① 采用非饱和激活函数ReLU
- ② 局部响应归一化
- ③ 基于双GPU进行卷积运算
- ④ 重叠池化pooling
- ⑤ dropout技术

# ReLU

神经元中采用非饱和的激活函数ReLU(Rectified Linear Unit), 网络训练速度比sigmoid或tanh 快好几倍[Krizhevsky et al., 2012]。

$$\text{ReLU } f(x) = \max(0, x), \text{ derivative } f'(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$$

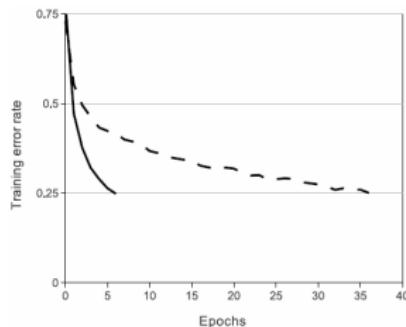


Figure 19: 实线表示ReLU, 虚线表示tanh; 在同样到达训练误差25%情况下, 时间比为6 倍

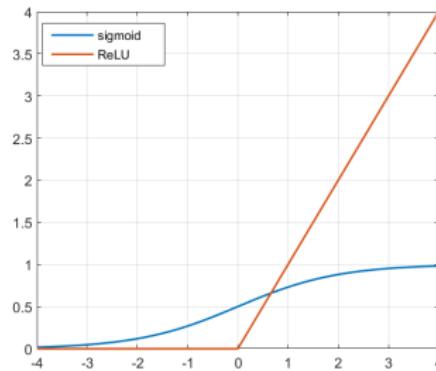


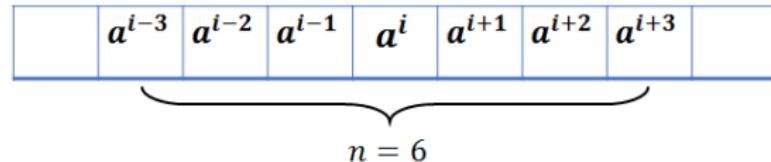
Figure 20: sigmoid与ReLU激活函数

# 局部响应归一化

ReLU没有饱和的性质，但是输出也没有限制。如果正值的输出一直累加到神经元，也会出现问题。所以AlexNet之中对神经元的输出加入了局部响应归一化。

$$b^i = a^i / \left( k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a^j)^2 \right)^\beta. \quad (28)$$

其中 $\alpha, \beta, k$ 是常系数。公式说明将 $a^i$ 周围的 $n$ 个神经元的输出进行归一化，防止其值经过多层累加值过大。



# GPU运算

为了加快训练速度，AlexNet的运算都是部署在型号为GTX 580 3GB的2个GPU上，GPU之间直接数据交互而不需要经过主机内存。GPU 内部的图像处理经过优化，卷积运算的速度比使用CPU运算快很多。

深度学习的模型大多是数据驱动型（data driven fashion）的，所以需要大量的数据，意味着需要大量的运算量。研究需要优秀的软、硬件资源支持，所以像Google、Facebook和百度之类的大公司在领头。

Andrew Ng<sup>5</sup>（1976年—）是斯坦福大学计算机科学系和电气工程系的副教授。2011年，在Google创建了Google Brain项目，通过分布式集群计算机开发超大规模的人工神经网络（使用1000台电脑训练包含10亿个连接的“神经网络”，使机器系统学会自动识别猫）。2014年5月，加入百度，负责“百度大脑”计划，并担任百度公司首席科学家。

不过，2015年6月，因在测试中作弊，百度人工智能被ImageNet LSVRC组织宣布禁止参赛一年。

---

<sup>5</sup><http://www.andrewng.org/>

# GPU运算

为了加快训练速度，AlexNet的运算都是部署在型号为GTX 580 3GB的2个GPU上，GPU之间直接数据交互而不需要经过主机内存。GPU 内部的图像处理经过优化，卷积运算的速度比使用CPU运算快很多。

深度学习的模型大多是数据驱动型（**data driven fashion**）的，所以需要大量的数据，意味着需要大量的运算量。研究需要优秀的软、硬件资源支持，所以像Google、Facebook和百度之类的大公司在领头。

Andrew Ng<sup>5</sup>（1976年—）是斯坦福大学计算机科学系和电气工程系的副教授。2011年，在Google创建了Google Brain项目，通过分布式集群计算机开发超大规模的人工神经网络（使用1000台电脑训练包含10亿个连接的“神经网络”，使机器系统学会自动识别猫）。2014年5月，加入百度，负责“百度大脑”计划，并担任百度公司首席科学家。

不过，2015年6月，因在测试中作弊，百度人工智能被ImageNet LSVRC组织宣布禁止参赛一年。

---

<sup>5</sup><http://www.andrewng.org/>

# GPU运算

为了加快训练速度，AlexNet的运算都是部署在型号为GTX 580 3GB的2个GPU上，GPU之间直接数据交互而不需要经过主机内存。GPU 内部的图像处理经过优化，卷积运算的速度比使用CPU运算快很多。

深度学习的模型大多是数据驱动型（*data driven fashion*）的，所以需要大量的数据，意味着需要大量的运算量。研究需要优秀的软、硬件资源支持，所以像Google、Facebook和百度之类的大公司在领头。

Andrew Ng<sup>5</sup>（1976年—）是斯坦福大学计算机科学系和电气工程系的副教授。2011年，在Google创建了Google Brain项目，通过分布式集群计算机开发超大规模的人工神经网络（使用1000台电脑训练包含10亿个连接的“神经网络”，使机器系统学会自动识别猫）。2014年5月，加入百度，负责“百度大脑”计划，并担任百度公司首席科学家。

不过，2015年6月，因在测试中作弊，百度人工智能被ImageNet LSVRC组织宣布禁止参赛一年。

<sup>5</sup><http://www.andrewng.org/>

# GPU运算

为了加快训练速度，AlexNet的运算都是部署在型号为GTX 580 3GB的2个GPU上，GPU之间直接数据交互而不需要经过主机内存。GPU 内部的图像处理经过优化，卷积运算的速度比使用CPU运算快很多。

深度学习的模型大多是数据驱动型（*data driven fashion*）的，所以需要大量的数据，意味着需要大量的运算量。研究需要优秀的软、硬件资源支持，所以像Google、Facebook和百度之类的大公司在领头。

Andrew Ng<sup>5</sup>（1976年—）是斯坦福大学计算机科学系和电气工程系的副教授。2011年，在Google创建了Google Brain项目，通过分布式集群计算机开发超大规模的人工神经网络（使用1000台电脑训练包含10亿个连接的“神经网络”，使机器系统学会自动识别猫）。2014年5月，加入百度，负责“百度大脑”计划，并担任百度公司首席科学家。

不过，2015年6月，因在测试中作弊，百度人工智能被ImageNet LSVRC组织宣布禁止参赛一年。

---

<sup>5</sup><http://www.andrewng.org/>

# 重叠池化

之前介绍的池化都是不重叠的池化操作，AlexNet之中采用了有重叠的操作。目的是为了更不容易在训练中出现过拟合。

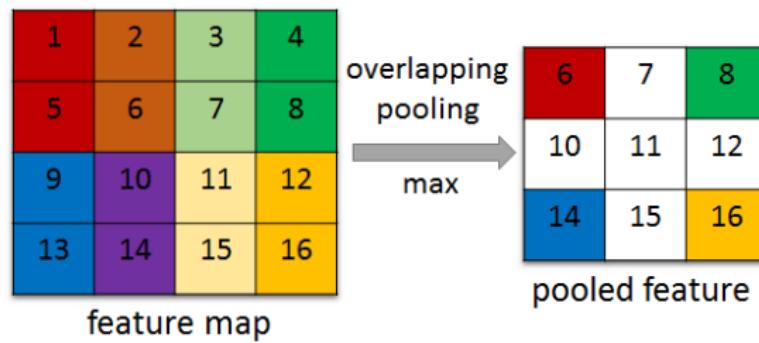
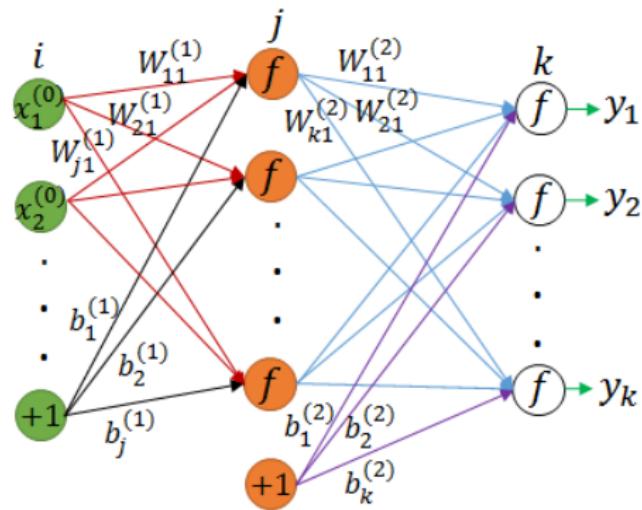


Figure 21: 重叠池化

# dropout

dropout技术的原理：以0.5的概率将每个隐含层的神经元的输出强制置0，这些神经元就不再参与前向传播和反向传播，所以对于每一次输入，训练的结构都是不同的。这种技术降低了神经元复杂的互适应关系，迫使网络学习到更为鲁棒性的特征。



# 测试结果

[Krizhevsky et al., 2012]中AlexNet与ImageNet LSVRC-2010的最佳结果进行比较

| Model                | Top-1 | Top-5 |
|----------------------|-------|-------|
| <i>Sparse coding</i> | 47.1% | 28.2% |
| <i>SIFT + FVs</i>    | 45.7% | 25.7% |
| CNN                  | 37.5% | 17.0% |

又给出了在ImageNet LSVRC-2012比赛中给出了训练的结果，

| Model           | Top-1(val) | Top-5(val) | Top-5(test) |
|-----------------|------------|------------|-------------|
| <i>SIFT+FVs</i> | -          | -          | 26.2%       |
| 1 CNN           | 40.7%      | 18.2%      | -           |
| 5 CNNs          | 38.1%      | 16.4%      | 16.4%       |
| 1 CNN*          | 39.0%      | 16.6%      | -           |
| 7 CNNs*         | 36.7%      | 15.4%      | 15.3%       |

数字表示测试多次取平均值，\*表示经过预训练的模型，val是验证，test是测试。

# 卷积核可视化

论文中给出了第一个卷积层的可视化卷积核的图，

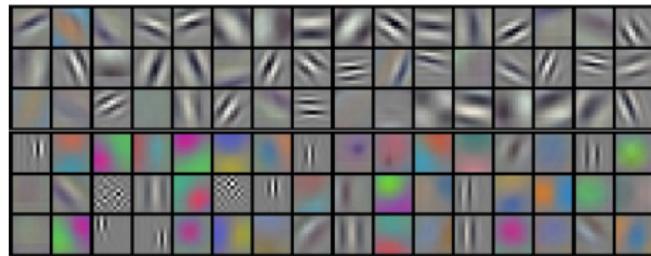


Figure 22: AlexNet第一个卷积层的卷积核

上方48个是GPU1学习到的特征，多为边缘信息；下方的48个是GPU2学习到的特征，多为颜色成分的特征。

为什么只给出第一层的卷积核，而不给出后面几层的。猜测：第一层的学习到的卷积核是图像的颜色或边缘等简单的低层次的特征，到了后面几层是将这些基本特征的一些综合，即使可视化也无法分析出其中的规律。

## ▶ Reference

# DeepID网络：人脸识别

2014年6月，香港中文大学孙伟教授[Sun et al., 2014]提出了DeepID网络结构，该结构与普通的卷积神经网络的结构相似，但是在隐含层，也就是倒数第二层，与Convolutional layer 4和Max-pooling layer 3相连，鉴于卷积神经网络层数越高视野域越大的特性，这样的连接方式可以既考虑局部的特征，又考虑全局的特征。第一代DeepID 网络在LFW 数据集测试，人脸识别率已经达到97.45%。目前发展到第三代，识别准确率已经超过了99%，使深度学习方法远远超过了人眼以及非深度学习方法在人脸识别上的准确率。

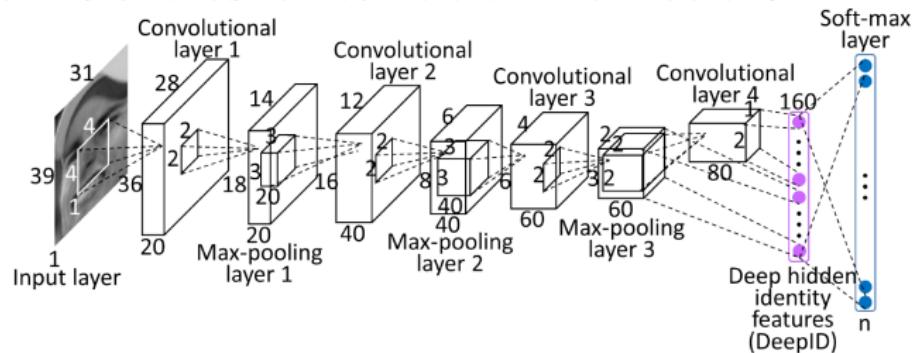


Figure 23: DeepID网络结构

## Reference

# Pyramid CNN

2014年3月，arXiv上发布了一篇论文“Learning Deep Face Representation”[Fan et al., 2014]。其中提出了一种Pyramid CNN模型，采用了逐层前向传播和逐层训练的方法来加快卷积网络的训练效率，并在LFW数据集（人脸识别）上获得了state-of-the-art的结果97.3%。

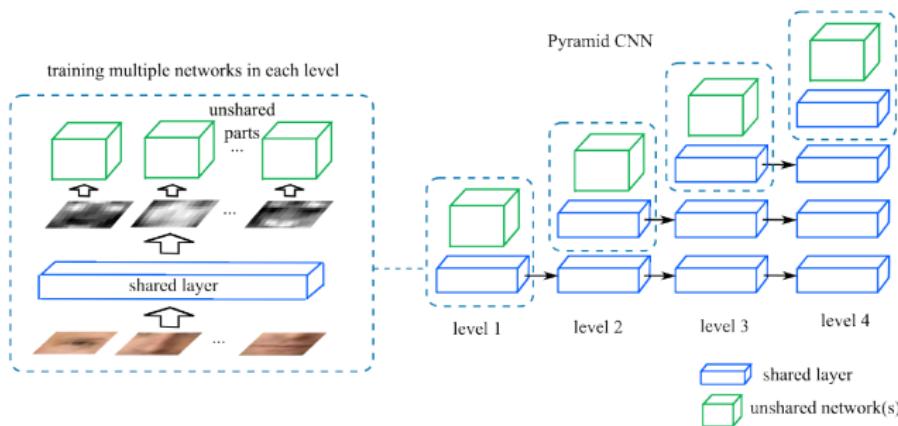


Figure 24: Pyramid CNN



ar $\chi$ iv（读音，archive）<sup>6</sup>始于1991年，是一个公开的、非盈利的收集学术论文预印本的网站。最初它只涉及物理领域，目前已经扩展数学、统计、生物、计算机科学等领域。网站由Cornell University Library和它的学术监督委员会维护和管理。目的是快速实现学术成果分享，与期刊不同。



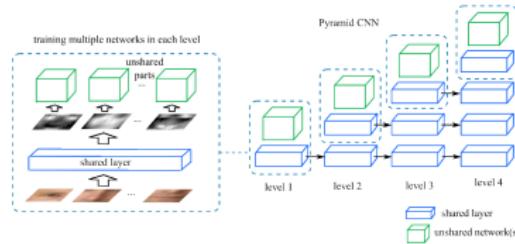
到2014年底，arXiv收藏的论文数达到1百万篇。仅2015年，网站的论文提交量为105,000，和超过139 million 的下载量。

由于其非盈利性，网站可以免费上传论文（经同行学者评审后发表）和下载论文（pdf和TEX源码），并且没有版权限制，arXiv公开的论文仍然可以投稿到学术期刊发表。

---

<sup>6</sup><http://arxiv.org/>

# Pyramid CNN



Pyramid CNN采用逐层训练卷积网络和权值共享的方式，提高了运算的速度和效率。两张人脸图像分别输入CNN，通过输出的特征判断是否属于同一个人。

$$\theta_0 = \arg \min_{\theta} L(f_{\theta}, I_{data}) \quad (29)$$

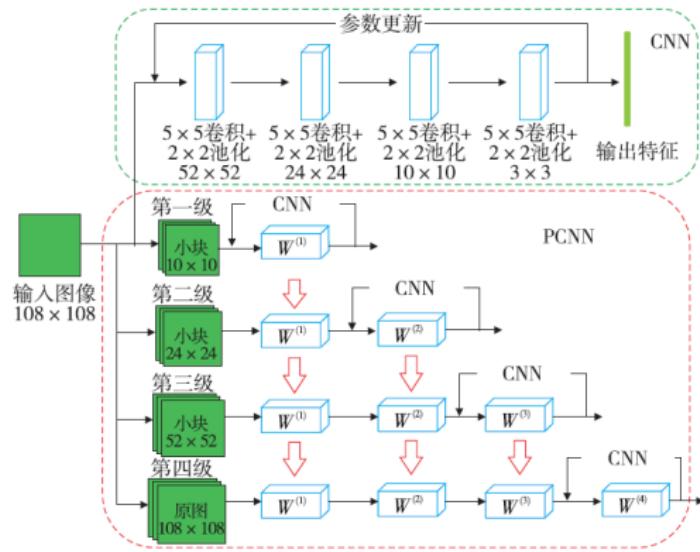
$$L = \sum_{I_1, I_2} \log(1 + \exp(\delta(I_1, I_2) D(I_1, I_2))) \quad (30)$$

$$D(I_1, I_2) = \alpha \cdot d(f_{\theta}(I_1), f_{\theta}(I_2)) - \beta \quad (31)$$

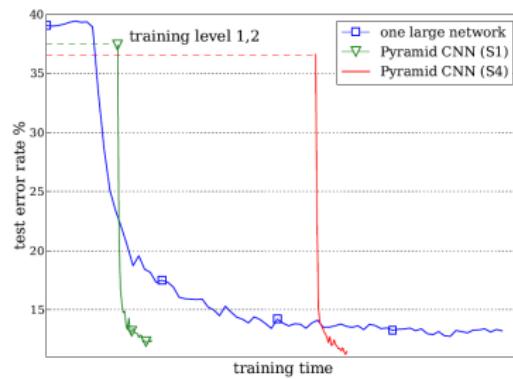
其中 $I$ 表示人脸图像， $f$ 表示网络的前向传播函数， $\delta$ 为标记 $I_1, I_2$ 是否为相同人脸， $d$ 为特征距离函数， $\theta$ 为网络权值参数。根据目标函数，训练要求尽量减小相同人脸的特征距离，而增大不同人脸的特征距离。

# Pyramid CNN

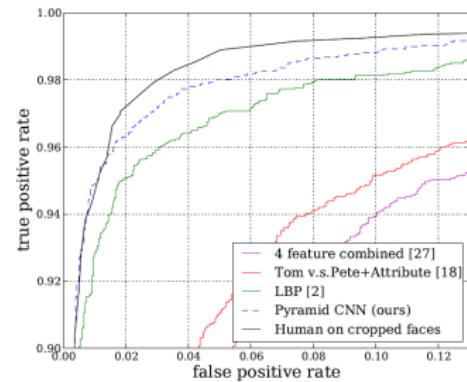
从输入中提取图像块，用于训练这层参数，反向传播只更新这一层的权值。之后固定这一层的权值（不再训练），图像通过这一层前向传播（卷积和池化），接着对下一层进行训练，如此继续，至最后一层训练完成。参数训练只在小图像块上进行，而且权值可以共享到下一次训练使用，提高了训练的效率。



# Pyramid CNN



**Figure 25:** Pyramid CNN在相同时间下，训练得到的误差比传统的CNN的误差小



**Figure 26:** Pyramid CNN在LFW数据集上的测试结果

# Pyramid CNN

关于这篇论文"Learning Deep Face Representation" [Fan et al., 2014] 有state-of-the-art的结果，却只发表在arXiv上，并没有投稿到学术期刊发表。

后来，在Google Scholar上搜索到了Yin, Qi, et al. "Learning Deep Face Representation." U.S. Patent No. 20,150,347,820. 3 Dec. 2015.

关于作者，Yin Qi, Megvii Inc.。

在2014年新闻中搜索到“印奇旷视科技Megvii，让机器看懂世界”，“旷视科技发明精确人脸识别技术获百万美元A轮融资”。

## Reference

# Pyramid CNN

关于这篇论文"Learning Deep Face Representation" [Fan et al., 2014] 有state-of-the-art的结果，却只发表在arXiv上，并没有投稿到学术期刊发表。

后来，在Google Scholar上搜索到了Yin, Qi, et al. "Learning Deep Face Representation." U.S. Patent No. 20,150,347,820. 3 Dec. 2015.

关于作者，Yin Qi, Megvii Inc.。

在2014年新闻中搜索到“印奇旷视科技Megvii，让机器看懂世界”，“旷视科技发明精确人脸识别技术获百万美元A轮融资”。

## Reference

# Pyramid CNN

关于这篇论文"Learning Deep Face Representation" [Fan et al., 2014] 有state-of-the-art的结果，却只发表在arXiv上，并没有投稿到学术期刊发表。

后来，在Google Scholar上搜索到了Yin, Qi, et al. "Learning Deep Face Representation." U.S. Patent No. 20,150,347,820. 3 Dec. 2015.

关于作者，Yin Qi, Megvii Inc.。

在2014年新闻中搜索到“印奇旷视科技Megvii，让机器看懂世界”，“旷视科技发明精确人脸识别技术获百万美元A轮融资”。

## ▶ Reference

# R-CNN

Ross [Girshick et al., 2014] 在CVPR上发表了R-CNN，又于[Girshick, 2015]，ICCV 上提出了Fast R-CNN 模型。

Region proposals with CNN (R-CNN) 是一种高性能的卷积神经网络加上区域提议（用于目标识别）的结构，该结构在VOC2012数据集上较之前最好的平均目标检测结果提升了30%以上。

而Fast R-CNN在之前的基础上又进行了改进，主要是减少了卷积网络中的计算量，显著加快了网络训练和测试的速度，所以标题突出了“Fast”。

其基于Python 和C++ (Caffe) 的代码实现都公开  
于 <http://github.com/rbgirshick/fast-rcnn>。

# R-CNN

Ross [Girshick et al., 2014] 在CVPR上发表了R-CNN，又于[Girshick, 2015], ICCV 上提出了Fast R-CNN 模型。

Region proposals with CNN (R-CNN) 是一种高性能的卷积神经网络加上区域提议（用于目标识别）的结构，该结构在VOC2012数据集上较之前最好的平均目标检测结果提升了30%以上。

而Fast R-CNN在之前的基础上又进行了改进，主要是减少了卷积网络中的计算量，显著加快了网络训练和测试的速度，所以标题突出了“Fast”。

其基于Python 和C++ (Caffe) 的代码实现都公开于 <http://github.com/rbgirshick/fast-rcnn>。

# R-CNN

Ross [Girshick et al., 2014] 在CVPR上发表了R-CNN，又于[Girshick, 2015], ICCV 上提出了Fast R-CNN 模型。

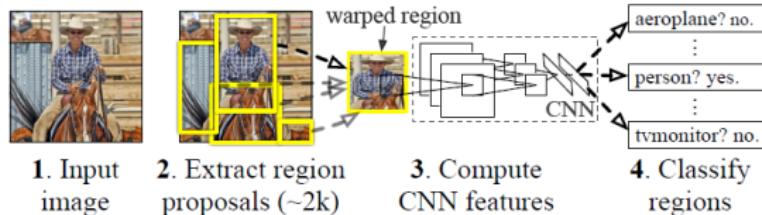
Region proposals with CNN (R-CNN) 是一种高性能的卷积神经网络加上区域提议（用于目标识别）的结构，该结构在VOC2012数据集上较之前最好的平均目标检测结果提升了30%以上。

而Fast R-CNN在之前的基础上又进行了改进，主要是减少了卷积网络中的计算量，显著加快了网络训练和测试的速度，所以标题突出了“Fast”。

其基于Python 和C++ (Caffe) 的代码实现都公开于 <http://github.com/rbgirshick/fast-rcnn>。

# R-CNN: 2014

## R-CNN: Regions with CNN features

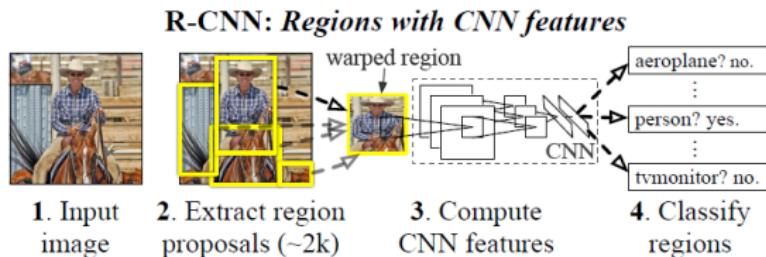


该系统的训练步骤如下

- ① 输入一整张图像
- ② 图像中（随机）提取出近2000张区域块
- ③ 每一个区域块都经过卷积网络计算出特征
- ④ 每一个区域使用线性支持向量机（SVM）分类，判断区域块属于哪一个目标

在PASCAL VOC 2010 数据集上测试得到了平均精确度53.7%，显著超过了之前最佳结果40.4%。[Girshick et al., 2014]第一次在PASCAL VOC上实现了CNN的分类转为目标检测的应用。

# R-CNN: 2014



该系统的训练步骤如下

- ① 输入一整张图像
- ② 图像中（随机）提取出近2000张区域块
- ③ 每一个区域块都经过卷积网络计算出特征
- ④ 每一个区域使用线性支持向量机（SVM）分类，判断区域块属于哪一个目标

在PASCAL VOC 2010 数据集上测试得到了平均精确度53.7%，显著超过了之前最佳结果40.4%。[Girshick et al., 2014]第一次在PASCAL VOC上实现了CNN的分类转为目标检测的应用。

# R-CNN: 2014

与分类不同，目标检测需要知道目标在图像中的具体位置。

## Recognition using regions

从一张输入图片中抽取出近2000个类别独立的目标区域块，（通过简单变形统一大小）输入到CNN中计算出特征向量，最后经过具体类别的线性支持向量机（SVM）判断区域块中的物体属于哪一个目标类别。

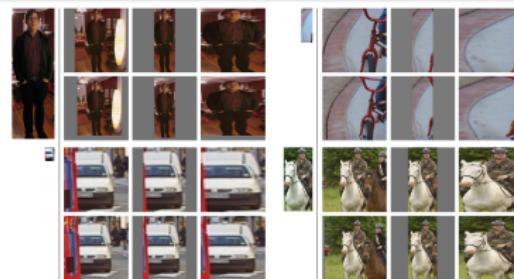
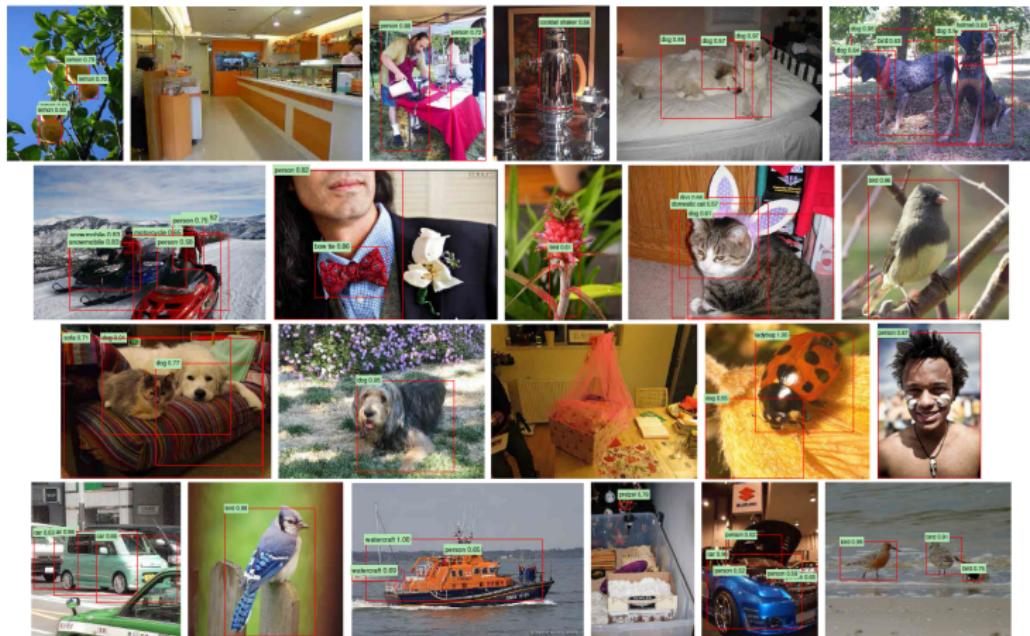


Figure 27: VOC 2007 训练集，变形后的区域块

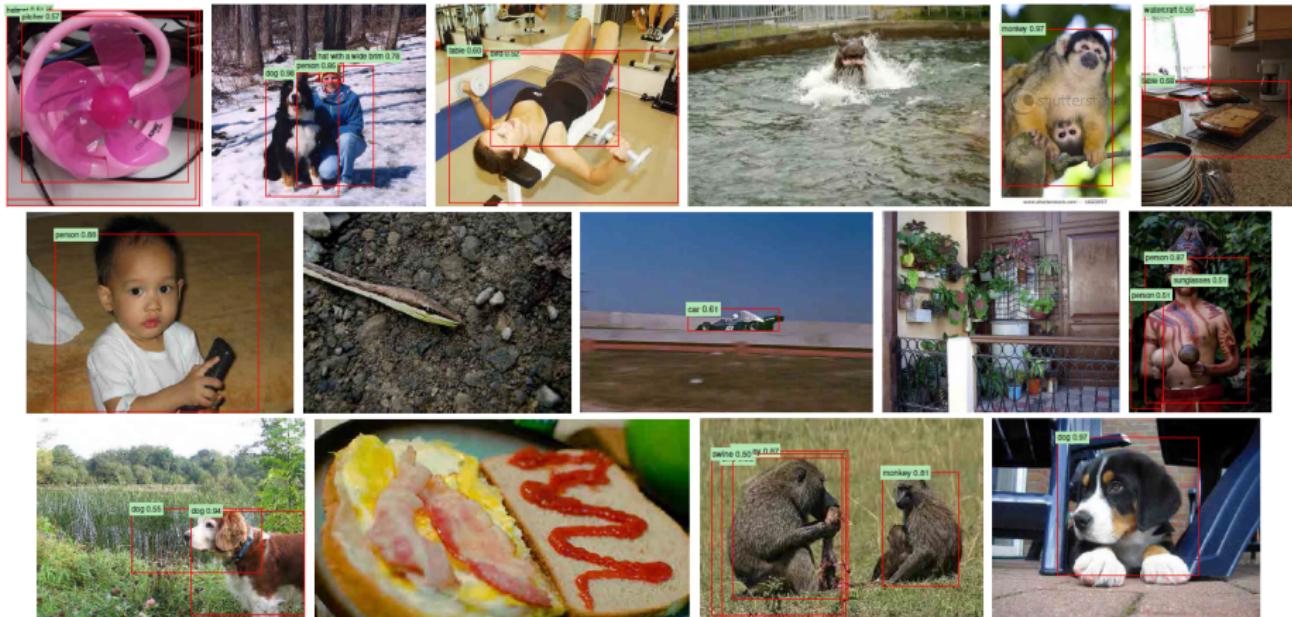
提取的目标区域块大小不统一，需要经过变形转换为统一的长宽，才能输入到CNN中。

# R-CNN: 2014

大约2000个区域块都经过CNN的前向传播，通过支持向量机SVM给每一个类别评分（可信度）。[Girshick et al., 2014]在arXiv的预印版本的附录中，给出了详细的检测结果。



# R-CNN: 2014



所有的区域块都是用同一个CNN网络的参数，计算出的特征向量是低维的。但是计算开始需要花费大量时间（13s/image on a GPU or 53s/image on a CPU）。

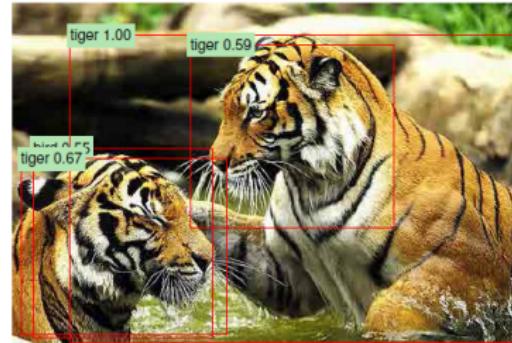
# R-CNN: 2014

## 预训练

CNN网络预先使用了ImageNet LSVRC 2012数据集上训练，基于Caffe框架，得到近似于AlexNet的性能。

## 微调

提取的大量区域块输入CNN同时也对CNN进行参数微调，使用随机梯度下降SGD。在训练过程中，我们判定提取的区域块和真实的感兴趣的区域重叠部分超过0.5，则当作正样本；反之，为负样本。



# R-CNN: 2014

## Bounding box regression

要通过大量的区域块训练来得到目标在图像中的具体位置，论文[Girshick et al., 2014]采用线性回归模型（在arXiv预印版的附录中给出），通过的最小化提取区域和真实目标位置的误差函数，来不断修正，逼近真实的目标位置。

$$\hat{G}_x = P_w d_x(P) + P_x \quad (32)$$

$$\hat{G}_y = P_h d_y(P) + P_y \quad (33)$$

$$\hat{G}_w = P_w \exp(d_w(P)) \quad (34)$$

$$\hat{G}_h = P_h \exp(d_h(P)) \quad (35)$$

$$W_{\star} = \arg \min_{\hat{W}_{\star}} \sum_i^N (t_{\star}^i - {\hat{W}_{\star}}^T \phi_5(P^i))^2 + \lambda \| \hat{W}_{\star} \|^2. \quad (36)$$

$x, y, w, h$ 分别是box的中心位置、宽度和高度， $P$ 是预测的box位置， $G$ 是真实标签位置， $d$ 是 $\phi$ 的线性函数，目标求出最优的 $W_{\star}$ 。详见[Girshick et al., 2014] arXiv预印版<http://arxiv.org/abs/1311.2524v5>。

# R-CNN: 2014

论文给出了在VOC 2010 测试数据集上的预测对比结果，

| VOC 2010 test            | aero        | bike        | bird        | boat        | bottle      | bus         | car         | cat         | chair       | cow         | table       | dog         | horse       | mbike       | person      | plant       | sheep       | sofa        | train       | tv          | mAP         |
|--------------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| DPM v5 [20] <sup>†</sup> | 49.2        | 53.8        | 13.1        | 15.3        | 35.5        | 53.4        | 49.7        | 27.0        | 17.2        | 28.8        | 14.7        | 17.8        | 46.4        | 51.2        | 47.7        | 10.8        | 34.2        | 20.7        | 43.8        | 38.3        | 33.4        |
| UVA [39]                 | 56.2        | 42.4        | 15.3        | 12.6        | 21.8        | 49.3        | 36.8        | 46.1        | 12.9        | 32.1        | 30.0        | 36.5        | 43.5        | 52.9        | 32.9        | 15.3        | 41.1        | 31.8        | 47.0        | 44.8        | 35.1        |
| Regionlets [41]          | 65.0        | 48.9        | 25.9        | 24.6        | 24.5        | 56.1        | 54.5        | 51.2        | 17.0        | 28.9        | 30.2        | 35.8        | 40.2        | 55.7        | 43.5        | 14.3        | 43.9        | 32.6        | 54.0        | 45.9        | 39.7        |
| SegDPM [18] <sup>†</sup> | 61.4        | 53.4        | 25.6        | 25.2        | 35.5        | 51.7        | 50.6        | 50.8        | 19.3        | 33.8        | 26.8        | 40.4        | 48.3        | 54.4        | 47.1        | 14.8        | 38.7        | 35.0        | 52.8        | 43.1        | 40.4        |
| R-CNN                    | 67.1        | 64.1        | 46.7        | 32.0        | 30.5        | 56.4        | 57.2        | 65.9        | 27.0        | 47.3        | 40.9        | 66.6        | 57.8        | 65.9        | 53.6        | 26.7        | 56.5        | 38.1        | 52.8        | 50.2        | 50.2        |
| R-CNN BB                 | <b>71.8</b> | <b>65.8</b> | <b>53.0</b> | <b>36.8</b> | <b>35.9</b> | <b>59.7</b> | <b>60.0</b> | <b>69.9</b> | <b>27.9</b> | <b>50.6</b> | <b>41.4</b> | <b>70.0</b> | <b>62.0</b> | <b>69.0</b> | <b>58.1</b> | <b>29.5</b> | <b>59.4</b> | <b>39.3</b> | <b>61.2</b> | <b>52.4</b> | <b>53.7</b> |

**BB**表示Bounding box regression，即修正了检测目标的区域位置。R-CNN较之前的算法性能提升了许多。

# Fast R-CNN: 2015

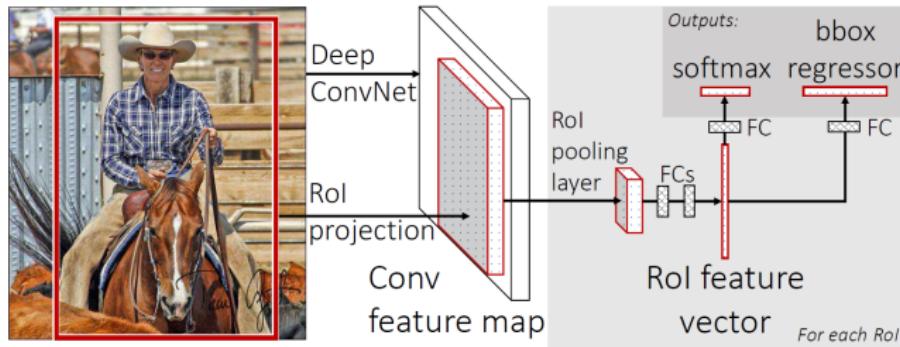
而Fast R-CNN在之前的基础上又进行了改进，主要是减少了卷积网络中的计算量，显著加快了网络训练和测试的速度，所以标题突出了“Fast”。

其基于Python 和C++ (Caffe) 的代码实现都公开于 <http://github.com/rbgirshick/fast-rcnn>。

[Girshick, 2015]给出了R-CNN的不足，

- ① CNN基于目标区域的训练，之后其特征用于SVM的训练，是分阶段的。
- ② 训练非常耗费内存空间和时间。为了SVM训练，需要将提取的大量的目标区域块存入磁盘中。
- ③ 目标检测速度过慢。

# Fast R-CNN: 2015

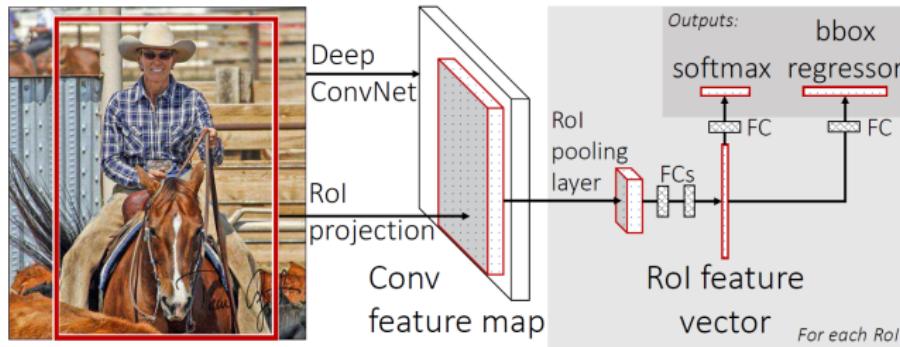


Fast R-CNN的结构如图所示，输入一整张图像和一系列的目标建议。先将整张图像通过深入卷积网络得到特征图，再用最大池化从目标建议区域的映射感兴趣区域（RoI）中提取出特征，得到固定长度的特征向量。

（R-CNN 则从原图中提取近2000张小图用于训练CNN，需要另外存储小图；而Fast R-CNN则只需记录RoI的位置）。

接着特征向量经过全连接层输出到softmax分类器（输出每个类别的概率估计）和bounding box regressor（ $r, c, h, w$ 调整bounding box的位置）。

# Fast R-CNN: 2015

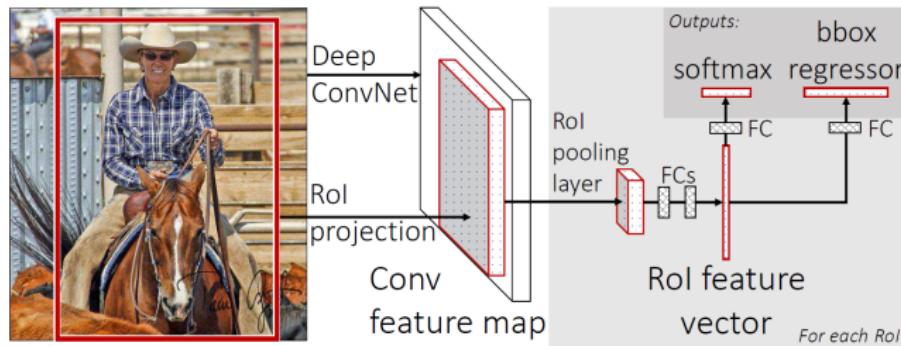


Fast R-CNN的结构如图所示，输入一整张图像和一系列的目标建议。先将整张图像通过深入卷积网络得到特征图，再用最大池化从目标建议区域的映射感兴趣区域（RoI）中提取出特征，得到固定长度的特征向量。

（R-CNN 则从原图中提取近2000张小图用于训练CNN，需要另外存储小图；而Fast R-CNN则只需记录RoI的位置）。

接着特征向量经过全连接层输出到softmax分类器（输出每个类别的概率估计）和bounding box regressor ( $r, c, h, w$ 调整bounding box的位置）。

# Fast R-CNN: 2015



Fast R-CNN的结构如图所示，输入一整张图像和一系列的目标建议。先将整张图像通过深入卷积网络得到特征图，再用最大池化从目标建议区域的映射感兴趣区域（RoI）中提取出特征，得到固定长度的特征向量。

（R-CNN 则从原图中提取近2000张小图用于训练CNN，需要另外存储小图；而Fast R-CNN则只需记录RoI的位置）。

接着特征向量经过全连接层输出到softmax分类器（输出每个类别的概率估计）和bounding box regressor（ $r, c, h, w$ 调整bounding box的位置）。

# Fast R-CNN: 2015

Fast R-CNN [Girshick, 2015] 采用的一些特性：

- ① 采用预先训练过的卷积网络
- ② Multi-task 损失函数，统一训练softmax和bbox regressor参数
- ③ Mini-batch 采样，便于池化层的反向传播
- ④ Truncated 奇异值分解

# Fast R-CNN: 2015

## 预训练

Fast R-CNN [Girshick, 2015] 采用了AlexNet [Krizhevsky et al., 2012] 训练完成的模型，在输入和输出结构作了少量修改，不需要再从初始状态耗费大量时间训练。

## 采用联合损失函数

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{Loc}(t^u, v), \quad (37)$$

$$L_{Loc}(t^u, v) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L_1}(t_i^u - v_i), \quad (38)$$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise.} \end{cases} \quad (39)$$

$p$ 代表softmax输出的类别的向量， $u$ 是类别， $t$ 为box的位置( $x, y, w, h$ )， $v$ 为真实的目标的位置， $u = 0$ 表示背景。

这样的目的是CNN的权值参数，softmax权值参数和bbox regressor的参数可以在训练时一起更新。

# Fast R-CNN: 2015

## Mini-batch 采样

从少量图中提取大量的感兴趣区域小块，如从2张图像中分别提取64张感兴趣区域图像，可提取出至少25%的兴趣区域提议与真实目标区域有至少0.5的重叠。另外，从一张图像中提取的好处是，通过池化层的反向传播可以很高效地计算。

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r, j)] \frac{\partial L}{\partial y_{rj}}. \quad (40)$$

$x_i$ 是池化层的第*i*个输入,  $y_{rj}$ 是第*r*个感兴趣区域的第*j*个输出量。偏导可以通过累加的方式反向传递回前一层。

# Fast R-CNN: 2015

## Truncated 奇异值分解

在计算感兴趣区域RoI的特征时，全连接层计算需要大量的时间，论文[Girshick, 2015]使用了truncated奇异值分解的办法，减少了权值的数量。

$$W \approx U \Sigma_t V^T = [u_1, u_2, \dots, u_t] \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_t \end{bmatrix} [v_1, v_2, \dots, v_t]^T, \quad (41)$$

$$W \in \mathbb{R}_{u \times v}, U \in \mathbb{R}_{u \times t}, \Sigma_t \in \mathbb{R}_{t \times t}, V \in \mathbb{R}_{v \times t}. \quad (42)$$

该方法将权重的参数从 $uv$ 减少到了 $t(u + v)$ ，如果 $t$ 远小于 $u, v$ 的时候效果会显著。改进的结构是将一个全连接层改为两个，权值从 $W$ 变为 $\Sigma_t V^T$ 和 $U$ ，从前向传播公式看出，原理是一样的

$$a = f(Wx + b) \Rightarrow a = f(U(\Sigma_t V^T x) + b). \quad (43)$$

# Fast R-CNN: 2015

论文[Girshick, 2015]给出了检测的比较结果,

| method        | train set | aero        | bike        | bird        | boat        | bottle      | bus         | car         | cat         | chair       | cow         | table       | dog         | horse       | mbike       | persn       | plant       | sheep       | sofa        | train       | tv          | mAP         |
|---------------|-----------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| BabyLearning  | Prop.     | 78.0        | 74.2        | 61.3        | 45.7        | 42.7        | 68.2        | 66.8        | 80.2        | 40.6        | 70.0        | 49.8        | 79.0        | 74.5        | 77.9        | 64.0        | 35.3        | 67.9        | 55.7        | 68.7        | 62.6        | 63.2        |
| NUS_NIN_c2000 | Unk.      | 80.2        | 73.8        | 61.9        | 43.7        | <b>43.0</b> | 70.3        | 67.6        | 80.7        | 41.9        | 69.7        | 51.7        | 78.2        | 75.2        | 76.9        | 65.1        | <b>38.6</b> | <b>68.3</b> | 58.0        | 68.7        | 63.3        | 63.8        |
| R-CNN BB [10] | 12        | 79.6        | 72.7        | 61.9        | 41.2        | 41.9        | 65.9        | 66.4        | 84.6        | 38.5        | 67.2        | 46.7        | 82.0        | 74.8        | 76.0        | 65.2        | 35.6        | 65.4        | 54.2        | 67.4        | 60.3        | 62.4        |
| FRCN [ours]   | 12        | 80.3        | 74.7        | 66.9        | 46.9        | 37.7        | 73.9        | 68.6        | 87.7        | 41.7        | 71.1        | 51.1        | 86.0        | 77.8        | 79.8        | 69.8        | 32.1        | 65.5        | 63.8        | 76.4        | 61.7        | 65.7        |
| FRCN [ours]   | 07++12    | <b>82.3</b> | <b>78.4</b> | <b>70.8</b> | <b>52.3</b> | 38.7        | <b>77.8</b> | <b>71.6</b> | <b>89.3</b> | <b>44.2</b> | <b>73.0</b> | <b>55.0</b> | <b>87.5</b> | <b>80.5</b> | <b>80.8</b> | <b>72.0</b> | 35.1        | <b>68.3</b> | <b>65.7</b> | <b>80.4</b> | <b>64.2</b> | <b>68.4</b> |

Table 3. **VOC 2012 test** detection average precision (%). BabyLearning and NUS\_NIN\_c2000 use networks based on [17]. All other methods use VGG16. Training set key: see Table 2, Unk.: unknown.

较之前的[Girshick et al., 2014]的R-CNN模型也有提升。

## Reference

# Outline

1 简介

2 卷积神经网络特性

3 前向传播

4 网络训练

5 应用

- LeNet
- MCDNN
- AlexNet
- DeepID
- Pyramid CNN
- Fast R-CNN

6 我的尝试

- STL-10
- CalTech101
- CIFAR-10
- 自编码算法

7 参考文献

# STL-10

STL-10<sup>7</sup>是一个可用于非监督学习的图像识别数据集。

- 10类图片：飞机、鸟、汽车、猫、鹿、狗、马、猴、船、卡车
- 彩色图像，96\*96 pixels
- 每个类别500张训练图像，800张测试图像
- 100000张没有标签的图像用于非监督学习

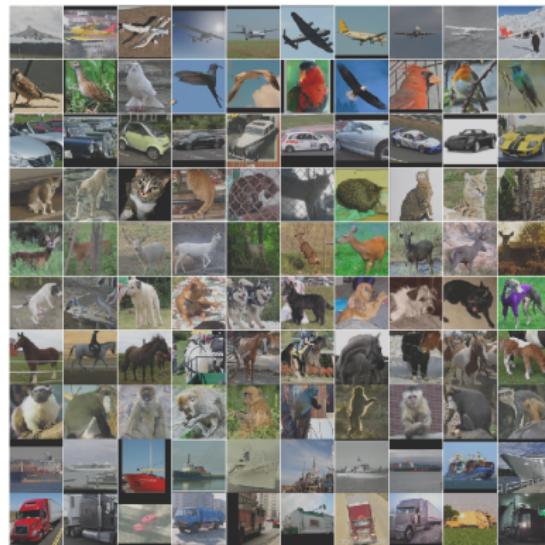


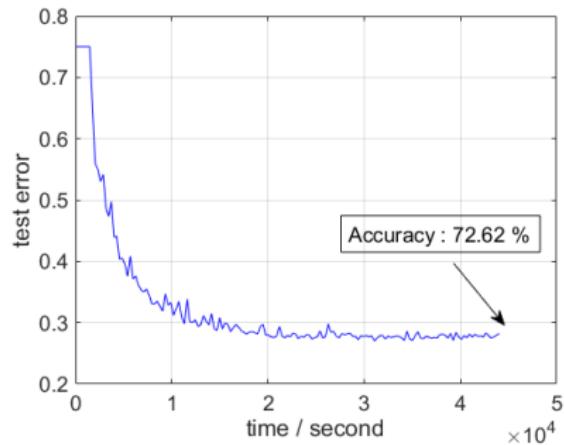
Figure 28: STL-10数据集

<sup>7</sup><http://cs.stanford.edu/~acoates/stl10>

# STL-10

选取部分图像子集用于训练和测试。4类：飞机、车、猫、狗；一共2000张训练图像，3200张测试图像；

1. 12 特征图，卷积核5\*5，池化2\*2；
2. 24 特征图，卷积核3\*3，池化2\*2；
3. 16 特征图，卷积核5\*5，池化2\*2；



| airplane | car    | cat    | dog    |
|----------|--------|--------|--------|
| 84.00%   | 81.38% | 63.12% | 58.00% |

# CalTech101

CalTech101<sup>8</sup> 是一个用于图像识别和分类的数据集，并为每张图片提供了标注。一共包含了9146张图像，100类图像和1类背景，也单独为每张图片提供了标注信息，用于目标检测。但是图片大小不一致，大部分为300\*200。每一类的图像数量也不一致，从40-800张不等。

测试时从中选取了10类（图像数量大于100）：人脸、简易人脸、豹、摩托车、飞机、盆栽、汽车、吊灯、海龟、帆船，每类随机选取50张训练图像，50张测试图像。

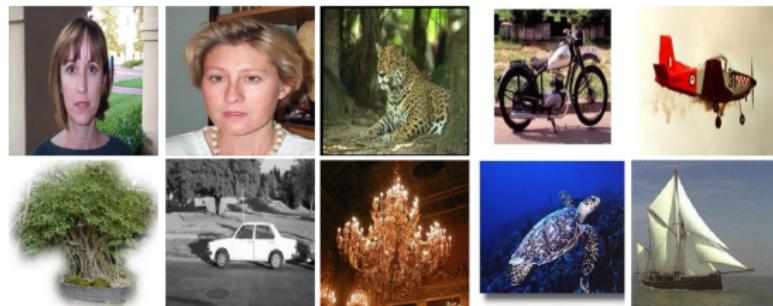


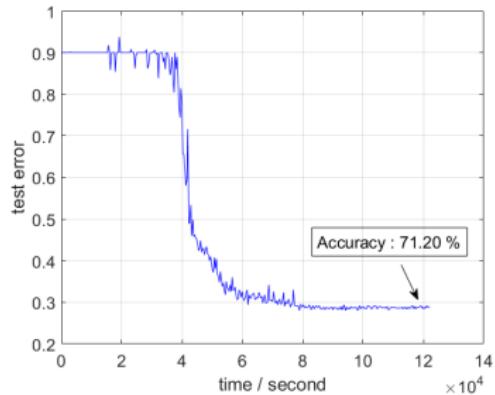
Figure 29: CalTech101数据集

<sup>8</sup>[http://www.vision.caltech.edu/Image\\_Datasets/Caltech101/](http://www.vision.caltech.edu/Image_Datasets/Caltech101/)

# CalTech101

10类图像统一为200\*200之后，一共500张训练图像，500张测试图像；

1. 12 特征图，卷积核5\*5，池化2\*2；
2. 24 特征图，卷积核3\*3，池化2\*2；
3. 20 特征图，卷积核5\*5，池化2\*2；
4. 16 特征图，卷积核5\*5，池化2\*2；



| 脸     | 简单脸   | 豹     | 摩托    | 飞机    |
|-------|-------|-------|-------|-------|
| 68.0% | 94.0% | 84.0% | 90.0% | 82.0% |
| 盆栽    | 汽车    | 吊灯    | 海龟    | 帆船    |
| 64.0% | 84.0% | 70.0% | 74.0% | 82.0% |

# CIFAR-10

CIFAR-10<sup>9</sup>是一个由Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton整理的用于图像分类的数据集。

- 10类图片：飞机，汽车，鸟，猫，鹿，狗，青蛙，马，船，卡车
- 彩色图像，32\*32 pixels
- 每类别5000张训练图像，1000张测试图像。

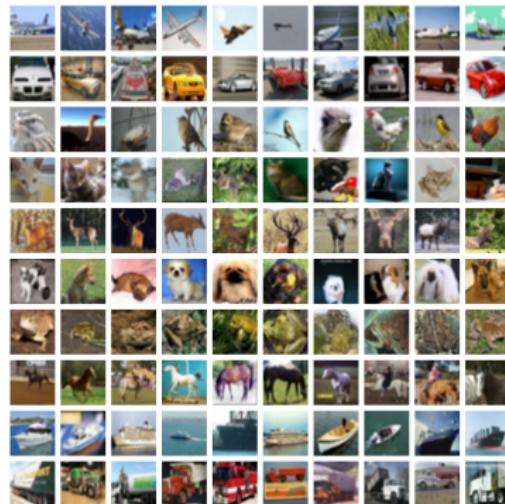


Figure 30: CIFAR-10数据集

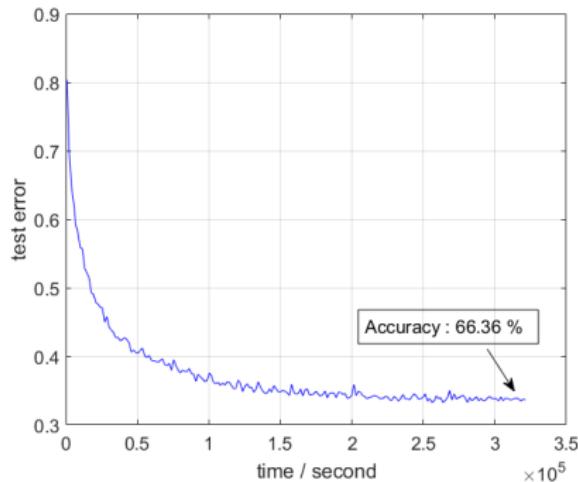
▶ Reference

<sup>9</sup><http://www.cs.toronto.edu/~kriz/cifar.html>

# CIFAR-10

10类32\*32彩色图片，一共50000张训练图像，10000张测试图像；

1. 12 特征图，卷积核5\*5，池化2\*2；
2. 24 特征图，卷积核5\*5，池化2\*2；



|             |             |            |            |             |
|-------------|-------------|------------|------------|-------------|
| 飞机<br>70.6% | 汽车<br>76.3% | 鸟<br>56.5% | 猫<br>52.3% | 鹿<br>56.8%  |
| 狗<br>49.7%  | 青蛙<br>74.0% | 马<br>67.9% | 船<br>82.5% | 卡车<br>77.0% |

# 激活函数

激活函数不仅有sigmoid和tanh函数，目前采用比较多的类型是 ReLU (Rectified Linear Unit, 修正线性单元)。为了引入稀疏性，抑制大部分神经元，减少能量消耗。采用这种激活函数，网络训练速度比sigmoid或tanh快好几倍[Krizhevsky et al., 2012]。

$$\text{sigmoid } f(x) = \frac{1}{1 + e^{-x}}$$

$$\text{derivative } f'(x) = f(x)(1 - f(x))$$

$$\text{ReLU } f(x) = \max(0, x)$$

$$\text{derivative } f'(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$$

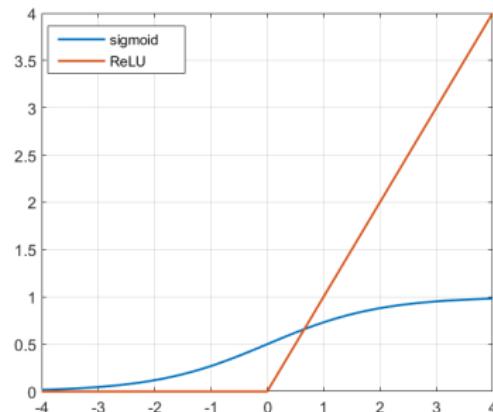
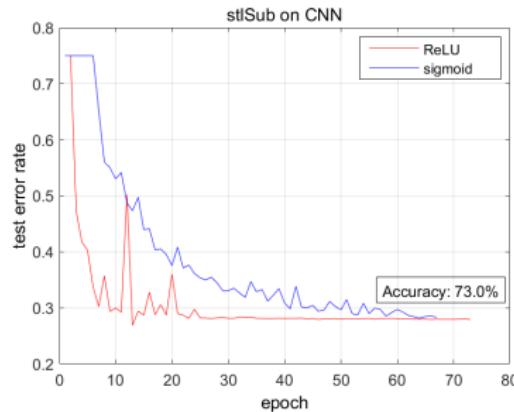
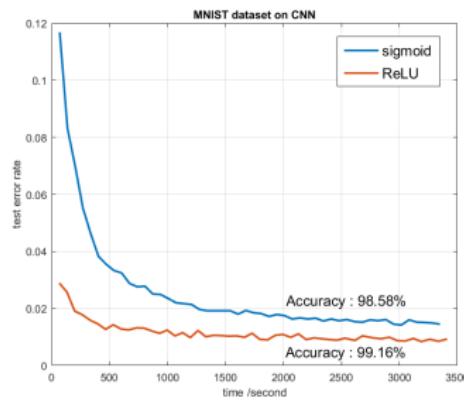


Figure 31: sigmoid与ReLU激活函数

# 激活函数

将sigmoid和ReLU激活函数分别应用到MNIST和STL数据集的测试中，对比我们的得到的识别准确率，



# 待研究

- ① 卷积操作多个输入特征图的组合方式。

$$\mathbf{u}_j^{(l)} = \sum_{i \in M_j} \mathbf{x}_i^{(l-1)} \bullet \mathbf{k}_{ij}^{(l)} + \mathbf{b}_j^{(l)}. \quad (44)$$

- ② 网络模型中feature maps 数量的设计。

**Table 1** Each Column Indicates Which Feature Map in S2 Are Combined by the Units in a Particular Feature Map of C3

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 0 | X |   |   | X | X | X |   |   | X | X | X  | X  |    |    | X  | X  |
| 1 | X | X |   |   | X | X | X |   |   | X | X  | X  | X  |    |    | X  |
| 2 | X | X | X |   |   |   | X | X | X |   | X  |    |    | X  | X  | X  |
| 3 |   | X | X | X |   |   | X | X | X | X |    | X  |    | X  | X  | X  |
| 4 |   |   | X | X | X |   |   | X | X | X | X  | X  | X  |    |    | X  |
| 5 |   |   |   | X | X | X |   |   | X | X | X  | X  | X  | X  | X  | X  |

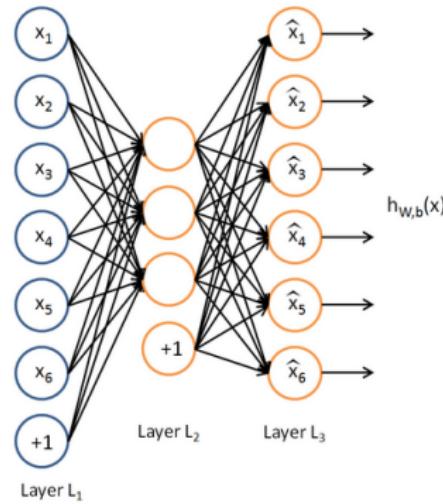
- ③ 卷积神经网络的权值的预训练，可采用一种无监督学习算法，稀疏自编码算法<sup>10</sup>。

<sup>10</sup> Andrew Ng, et al.

[http://ufldl.stanford.edu/wiki/index.php/UFLDL\\_Tutorial](http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial)

# 自编码神经网络

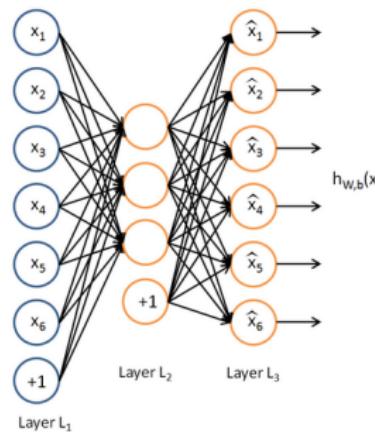
假设我们只有一个没有带类别标签的样本集合  $x^{(1)}, x^{(2)}, x^{(3)}, \dots$ ，其中  $x^{(i)} \in \mathbf{R}^n$ 。自编码神经网络采用无监督学习方式，让输出值尽量靠近输入值，即  $y^{(i)} = x^{(i)}$ 。下图是一个自编码神经网络<sup>11</sup>



自编码神经网络尝试学习一个  $h_{W,b}(x) \approx x$  的函数。换句话说，它尝试逼近一个恒等函数，从而使得输出  $\hat{x}$  接近于输入  $x$ 。

<sup>11</sup>[http://ufldl.stanford.edu/wiki/index.php/UFLDL\\_Tutorial](http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial)

# 自编码神经网络

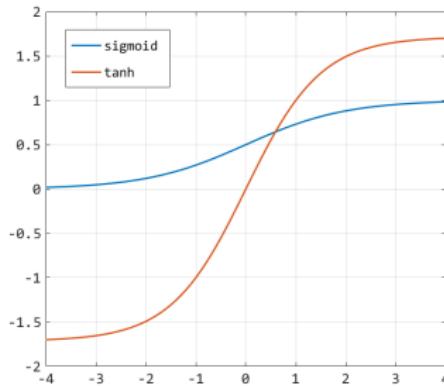


限定隐藏神经元的数量，我们就可以从输入数据中发现一些有趣的结构。举例来说，假设输入  $x$  是一张  $10 \times 10$  图像(100个像素)的像素值，于是  $n = 100$ ，其隐藏层  $L_2$  中有 50 个隐藏神经元。注意，输出也是 100 维的  $y \in \mathbf{R}^{100}$ 。由于只有 50 个隐藏神经元，迫使自编码神经网络去学习输入数据的一种压缩表示，也就是说，它必须从 50 维的隐藏神经元激活向量  $a^{(2)} \in \mathbf{R}^{50}$  中重构出 100 维的像素灰度值输入  $x$ 。

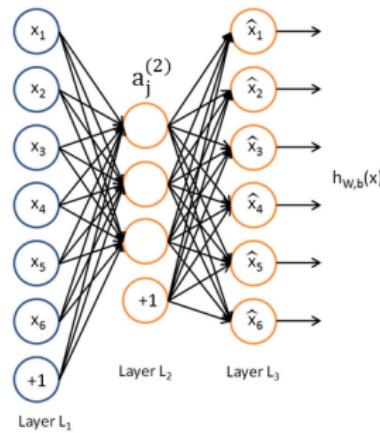
如果输入数据中隐含着一些特定的结构，比如某些输入特征是彼此相关的，那么这一算法就可以发现输入数据中的这些相关性。

# 稀疏性表示

稀疏性可以解释如下：假设激活函数是sigmoid函数（或tanh函数），当神经元的输出接近于1的时候我们认为它被激活，而输出接近于0（或-1）的时候认为它被抑制，那么使得神经元大部分的情况下都是被抑制（输出接近于0或-1）的限制被称作稀疏性限制。



# 稀疏性表示



$a_j^{(2)}$  表示隐藏神经元  $j$  的激活度，但是这一表示并未明确指出哪一个输入样本  $x$  的激活度。所以我们将使用  $a_j^{(2)}(x)$  来表示在给定输入为  $x$  情况下，自编码神经网络隐藏神经元  $j$  的激活度。而用如下

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m [a_j^{(2)}(x^{(i)})]. \quad (45)$$

表示隐藏神经元  $j$  的平均活跃度（在  $m$  个训练训练样本上取平均）。

# 稀疏性表示

我们可以再加入一条限制

$$\hat{\rho}_j = \rho, \quad (46)$$

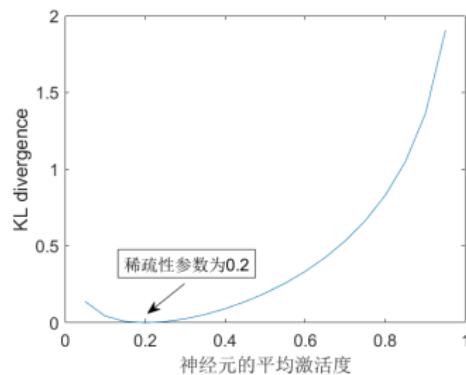
其中， $\rho$  是稀疏性参数，通常是一个接近于 0 的较小的值（比如 $\rho = 0.05$ ）。也就是说，我们想要让隐藏神经元  $j$  的平均活跃度接近 0.05，也就是大部分的隐藏神经元的活跃度必须接近于 0（sigmoid 函数）。

为了实现稀疏性限制，我们将会在优化的目标函数中加入一个额外的惩罚因子，相对熵（KL divergence）：

$$\sum_{j=1}^{s_2} \text{KL}(\rho || \hat{\rho}_j) = \sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}. \quad (47)$$

这里， $s_2$  是隐藏层中神经元的数量，而  $j$  依次代表隐藏层中的每一个神经元。

# 稀疏性表示



它将惩罚那些  $\hat{\rho}_j$  和  $\rho$  相差很大的情况从而使得隐藏神经元的平均活跃度保持在较小值附近。

$$\sum_{j=1}^{s_2} \text{KL}(\rho || \hat{\rho}_j) = \sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}. \quad (48)$$

这里,  $s_2$  是隐藏层中神经元的数量, 而  $j$  依次代表隐藏层中的每一个神经元。

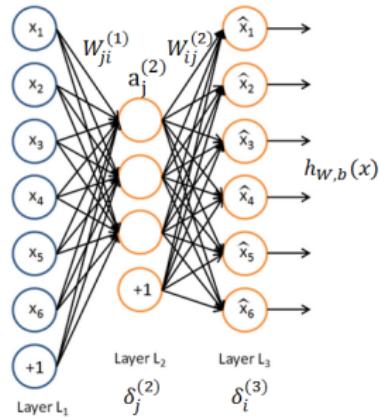
# 目标函数

整体代价函数为：

$$\begin{aligned}
 J(W, b) = & \left[ \frac{1}{m} \sum_{t=1}^m \frac{1}{2} \| h_{W,b}(x^{(t)}) - x^{(t)} \|^2 \right] \\
 & + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left( W_{ji}^{(l)} \right)^2 \\
 & + \beta \sum_{j=1}^{s_2} \text{KL}(\rho || \hat{\rho}_j).
 \end{aligned} \tag{49}$$

- 第一项  $J(W, b)$  是一个误差平方项（所有样本取平均）；
- 第二项是权重衰减项，其目的是减小权重的幅度，减缓过拟合；
- $\hat{\rho}_j$  则也（间接地）取决于  $W, b$ ，因为它是隐藏神经元  $j$  的平均激活度，由前向传播计算得到  $a = f(Wx + b)$ ；
- $\lambda, \beta$  权衡惩罚项的比例。

# 反向传播计算



$$\delta_i^{(3)} = (h_{W,b}(x^{(i)}) - x^{(i)}) f'(z_i^{(3)}), \quad (50)$$

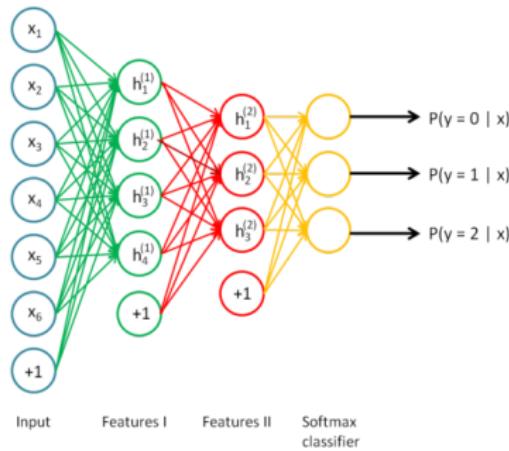
$$\delta_j^{(2)} = \left( \left( \sum_{i=1}^{s_2} W_{ij}^{(2)} \delta_i^{(3)} \right) + \underbrace{\beta \left( -\frac{\rho}{\hat{\rho}_j} + \frac{1-\rho}{1-\hat{\rho}_j} \right)}_{\text{bias term}} \right) f'(z_j^{(2)}). \quad (51)$$

$$\frac{\partial J}{\partial W_{ij}^{(2)}} = \delta_i^{(3)} a_j^{(2)} + \lambda W_{ij}^{(2)}, \quad \frac{\partial J}{\partial b_i^{(2)}} = \delta_i^{(3)}. \quad (52)$$

$$\frac{\partial J}{\partial W_{ji}^{(1)}} = \delta_j^{(2)} x_i + \lambda W_{ji}^{(1)}, \quad \frac{\partial J}{\partial b_j^{(1)}} = \delta_j^{(2)}. \quad (53)$$

# 深度神经网络

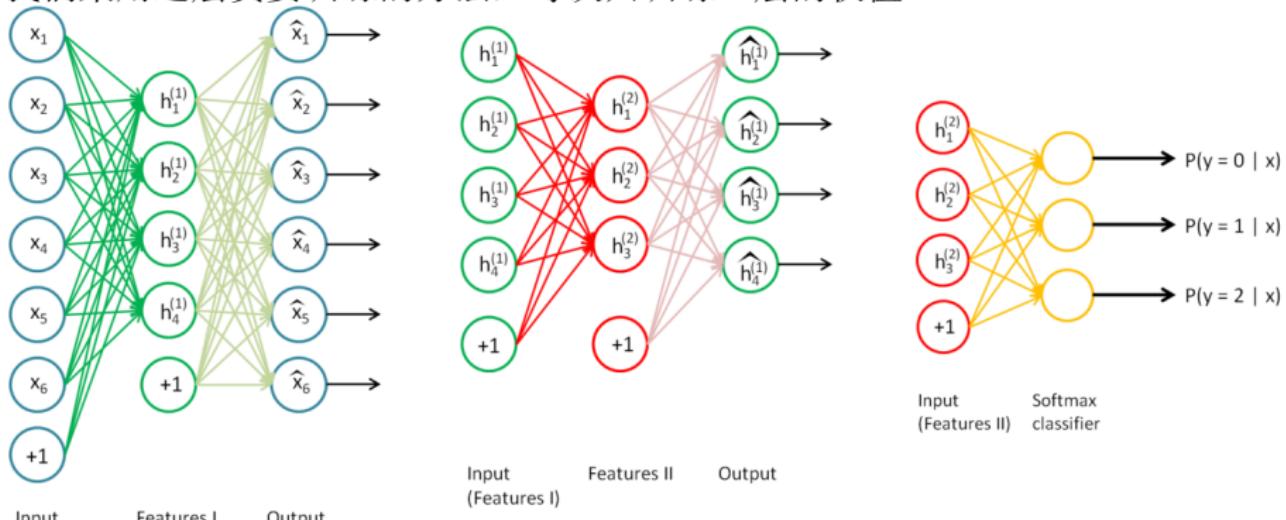
我们可以用多个自编码神经网络构建深度神经网络，用于图像分类。



举个例子，如果网络的输入数据是图像，网络的第一层会学习如何去识别边，第二层一般会学习如何去组合边，从而构成轮廓、角等。更高层会学习如何去组合更形象且有意义的特征。例如，如果输入数据集包含人脸图像，更高层会学习如何识别或组合眼睛、鼻子、嘴等人脸器官。

# 逐层贪婪训练

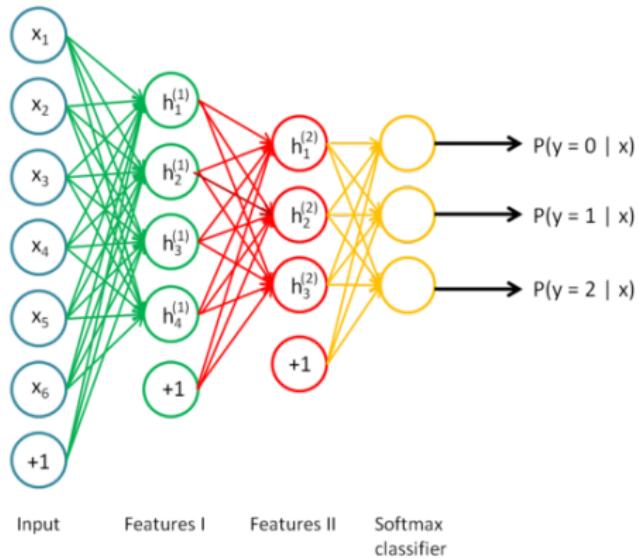
我们采用逐层贪婪训练的方法，每次只训练一层的权值



首先是（非监督）训练第一层，然后保存权值（绿色）和隐藏层的输出，舍弃之后的部分；第二层（红色）的权值也类似；最后一层是（监督）训练softmax分类器的权值(黄色)。

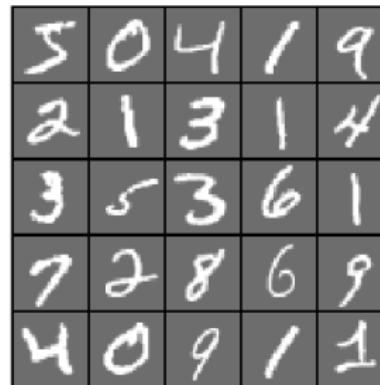
# 深度神经网络

最终，我们可以将这三层组合起来构建一个包含两个隐藏层和一个最终softmax分类器的栈式自编码网络，这个网络能够进行图像分类。



# 基于公开数据集的测试

MNIST手写数据集<sup>12</sup>由Yann LeCun et al. 维护，是一个手写数字0-9的灰度图像集合，一共含有60000张训练样本，10000张测试样本。图片的大小都统一，且数字置于图片中心位置。已经被广泛用于测试图像分类算法的性能。LeCun et al. 1998 完成准确率是99.3%，而且Ciresan et al. 2012 已经达到目前最高的准确率是99.77%，已经能和人眼的识别率相匹配。



<sup>12</sup><http://yann.lecun.com/exdb/mnist/>

# 基于MNIST数据集的分类测试

图像输入(28\*28)  $\Rightarrow$  隐藏层1(28\*28)  $\Rightarrow$  隐藏层2(28\*28)  $\Rightarrow$  softmax分类



Figure 32: 输入图像

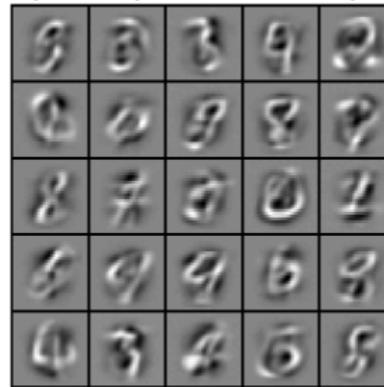


Figure 33: 第一层权值

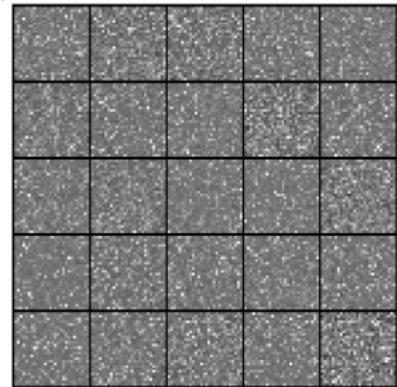


Figure 34: 第二层权值

准确率有89.34%，其实自编码网络的结构权值参数过多，所以性能并不好，不能与CNN相比拟。

# Outline

- 1 简介
- 2 卷积神经网络特性
- 3 前向传播
- 4 网络训练
- 5 应用
  - LeNet
  - MCDNN
  - AlexNet
  - DeepID
  - Pyramid CNN
  - Fast R-CNN
- 6 我的尝试
  - STL-10
  - CalTech101
  - CIFAR-10
  - 自编码算法
- 7 参考文献

# Bibliography I

-  Bouvie, J. (2006).  
Notes on convolutional neural networks.
-  Ciresan, D., Meier, U., and Schmidhuber, J. (2012).  
Multi-column deep neural networks for image classification.  
In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE.
-  Fan, H., Cao, Z., Jiang, Y., Yin, Q., and Doudou, C. (2014).  
Learning deep face representation.  
*arXiv preprint arXiv:1403.2802*.
-  Girshick, R. (2015).  
Fast r-cnn.  
In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448.

# Bibliography II

-  Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014).  
Rich feature hierarchies for accurate object detection and semantic segmentation.  
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.
-  Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012).  
Imagenet classification with deep convolutional neural networks.  
In *Advances in neural information processing systems*, pages 1097–1105.
-  LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998).  
Gradient-based learning applied to document recognition.  
*Proceedings of the IEEE*, 86(11):2278–2324.

# Bibliography III

-  LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., et al. (1995).  
Comparison of learning algorithms for handwritten digit recognition.  
In *International conference on artificial neural networks*, volume 60, pages 53–60.
-  Sun, Y., Wang, X., and Tang, X. (2014).  
Deep learning face representation from predicting 10,000 classes.  
In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

End

谢谢！

Thanks for your listening.