模式识别课程

全连接网络处理图像是，目标物占比小的时候，容易被背景淹没;
网络不宜太深；

卷积神经网络

张俊超

中南大学
航空航天学院

模式识别-神经网络分类器

- 模式识别算法
  - 统计模式识别
    - 线性分类器
      - 感知器
      - LMSE算法
      - SVM
      - Fisher线性判别
    - 贝叶斯分类器
      - 最小错误率
      - 最小风险
    - 最近邻分类器
      - 模板匹配
      - K近邻算法
    - 神经网络分类器
      - 多层感知器
      - CNN
    - 统计聚类算法
      - 层次聚类
      - 动态聚类
    - 特征选择与特征提取
  - 结构模式识别

模式识别 -神经网络分类器

卷积神经网络(CNN)

深度学习究竟是什么？

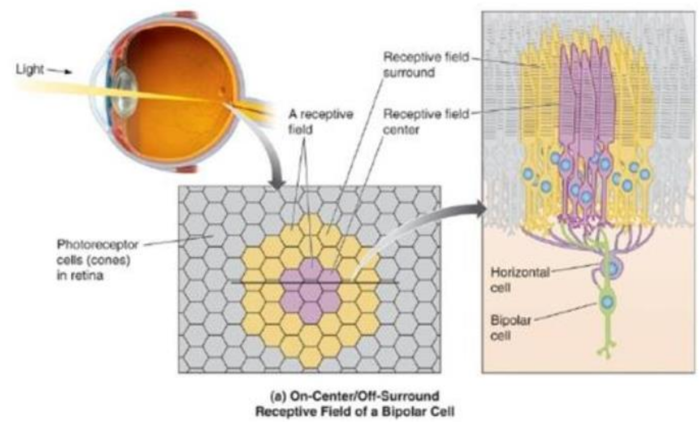
为什么要学习卷积神经网络？
和人工神经网络(或全连接网络)有什么区别？

卷积神经网络(CNN)

David H. Hubel

Torsten N. Wiesel

1981年诺贝尔生理学或医学奖获得者



视神经系统的分层结构和局部感受野

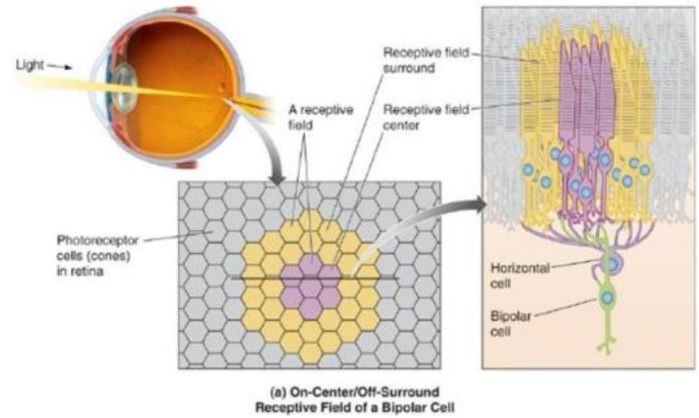
- 1959 年，哈佛医学院的神经生理学家 Hubel 和 Wiesel 在研究猫的视觉神经系统过程中，发现单个的视细胞只对一个很小区域内的光刺激有响应-**局部感受野**(Local Receptive Field)。对LRF的响应和侧抑制机制，使得视网膜能够对物体图像的边缘等信息产生强化的响应，即发现物体轮廓等关键信息。

卷积神经网络(CNN)

David H. Hubel

Torsten N. Wiesel

1981年诺贝尔生理学或医学奖获得者

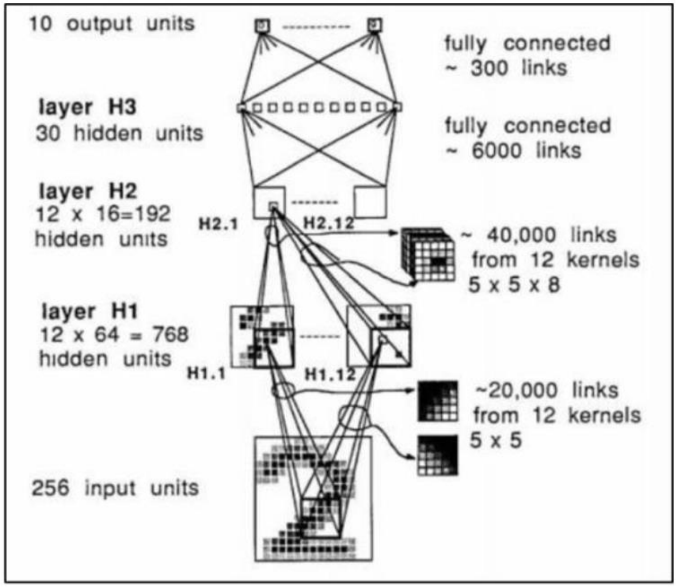


视神经系统的分层结构和局部感受野

- 他们还发现**视神经系统是分层**的，上一层视神经细胞在刺激下产生的输出，会传递到下一层视神经细胞，同样以局部感受野的形式在下一层产生刺激，使得视觉信息能够逐层进行抽象，最终形成有效的视觉认知结果。

卷积神经网络(CNN)

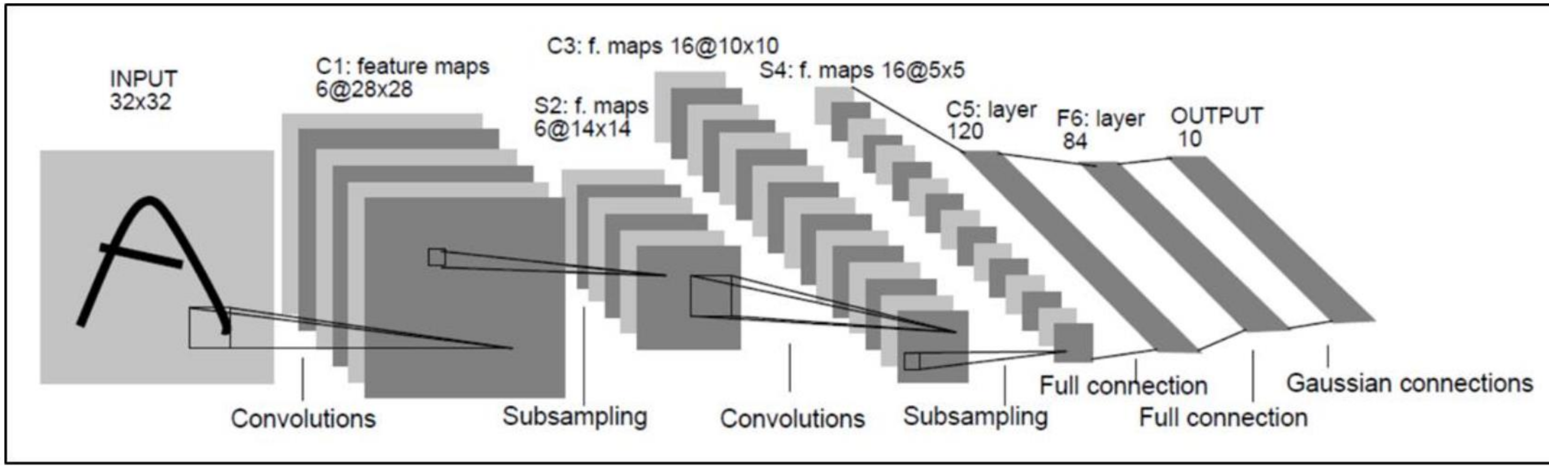
杨立昆
Yann LeCun



LeNet，1989

- 1989年，在前人工作的基础上，杨立昆结合局部感受野逐层特征提取和误差反向传播算法，提出了最初的**LeNet卷积神经网络模型**，并成功应用于信封邮政编码的手写数字识别，**达到了约90%的正确率**。

卷积神经网络(CNN)



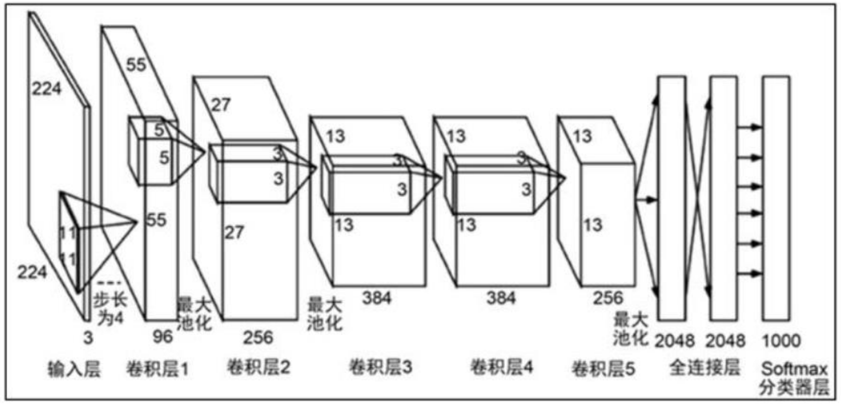
LeNet，1989 →LeNet-5，1998

- LeNet-5是Yann LeCun于1998年提出的改进的LeNet，它最重要的是在卷积层之间增加了**池化(Pooling)层**，不仅可以降低特征的维度，而且还能增强系统对平移变换的特征不变性。并且在手写数字识别训练集 MNIST 上的**正确率高达 99%以上**。

卷积神经网络(CNN)

Alex Krizhevsky

- ➢ ReLU
- ➢ Dropout
- ➢ GPU



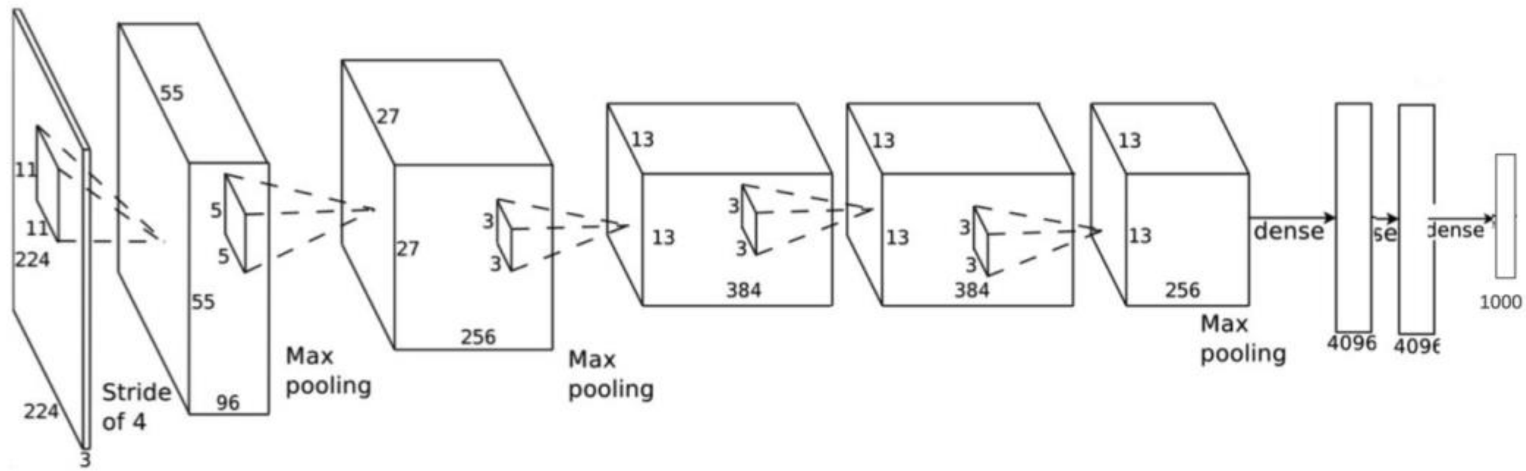
LeNet，1989　LeNet-5，1998　→AlexNet，2012

- ➢ LeNet-5 的局部成功，还没能引起学术界和社会公众的普遍关注，直到 2012 年**AlexNet被提出**。

- ➢ Alex Krizhevsky(Hinton的研究生)，他在Hinton的指导和支持下，设计了一个卷积神经网络AlexNet，在2012年的ImageNet图像识别大赛中夺得冠军，并且top-5的错误率仅为15.3%，大幅度领先第二名的 26.2% 。

卷积神经网络(CNN)

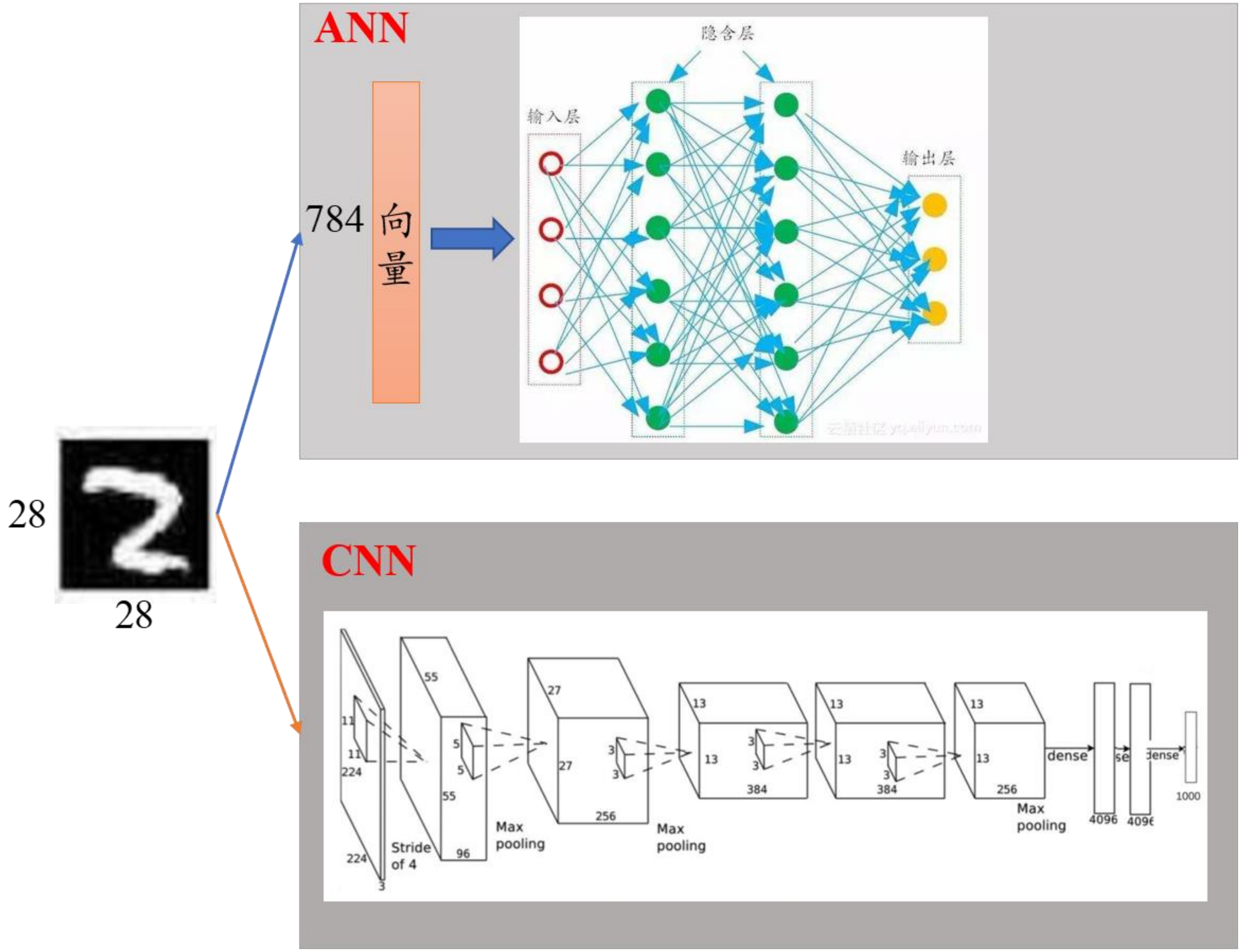
网络结构(以AlexNet为例)



- ➢层级结构
- ➢输入层、卷积层、激励层、池化层、全连接层

卷积神经网络(CNN)

与全连接网络对比



卷积神经网络(CNN)-卷积层

什么是卷积?

卷积核

$$\begin{bmatrix}1 & 0 & 1\\0 & 1 & 0\\1 & 0 & 1\end{bmatrix}$$

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

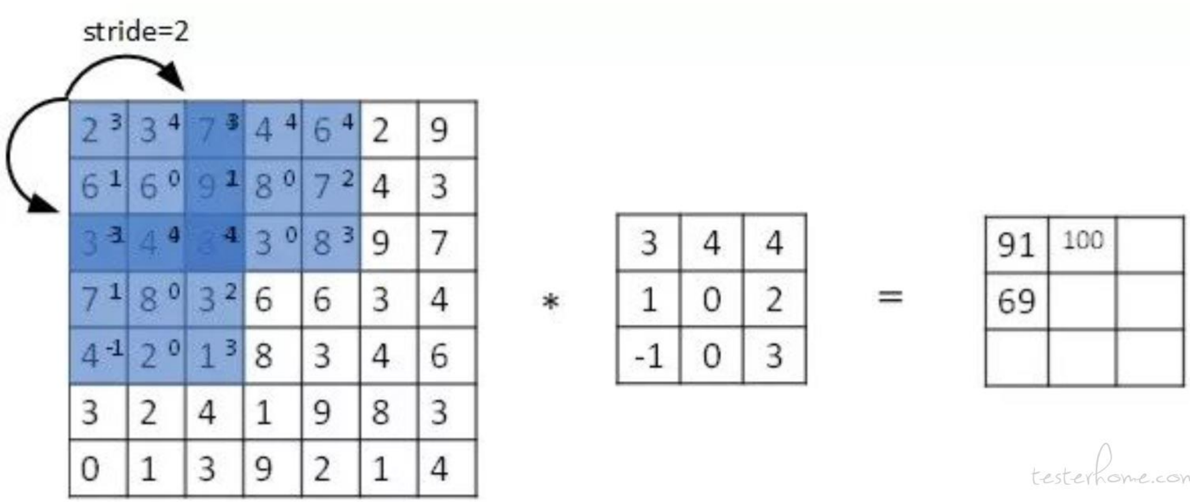
4		

Convolved Feature

$$\begin{array}{r} (1\times\boxed{1}) \\ (1\times\boxed{0}) \\ (1\times\boxed{1}) \\ (0\times\boxed{0}) \\ (1\times\boxed{1}) \\ (1\times\boxed{0}) \\ (0\times\boxed{1}) \\ (0\times\boxed{0}) \\ +\quad (1\times\boxed{1}) \\ \hline =\quad 4 \end{array}$$

卷积神经网络(CNN)-卷积层

- 明确几个概念
  - 卷积核大小/ kernel size (或filter size)
  - 深度/ depth (= 卷积核数目/ filter number)
  - 步长/ stride
  - 填充值/ padding



卷积神经网络(CNN)-卷积层

- 输入图像大小：$W*H*c$
  - 卷积核大小：$f*f*c$
  - 卷积核数目：K
  - 步长Stride: s
  - 边界填充padding: p
- 输出图像大小为：$W1*H1*K$

  其中：

$$W1 = \lfloor (W + 2p - f) / s + 1 \rfloor$$
$$H1 = \lfloor (H + 2p - f) / s + 1 \rfloor$$

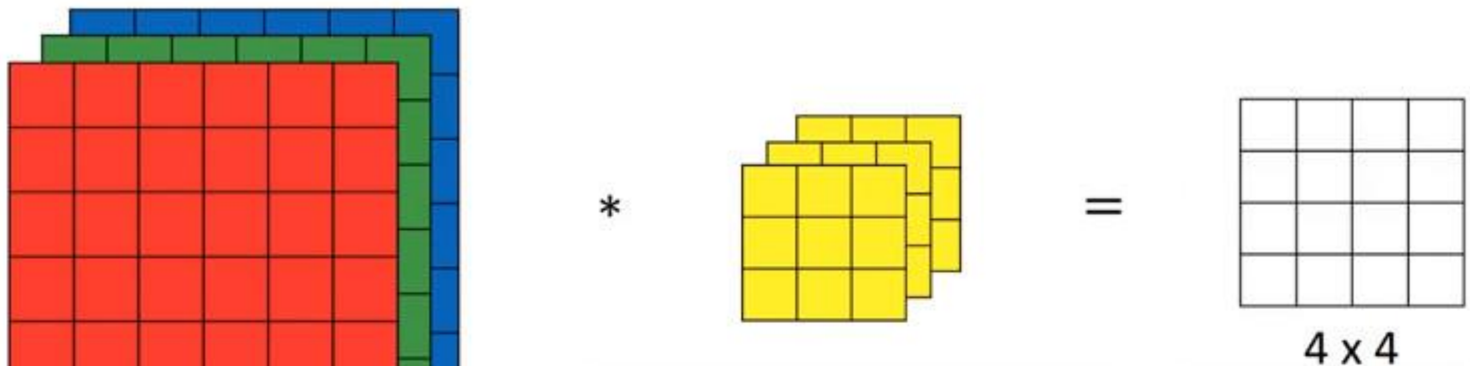
- s=1时，卷积前后保证图像大小(长和宽)不变：

$$p = \frac{f - 1}{2}$$

卷积神经网络(CNN)-卷积层

例子：$s = 1, p = 0, f = 3, c = 3, K = 1$

$$\frac{6+2*0-3}{1}+1=4$$



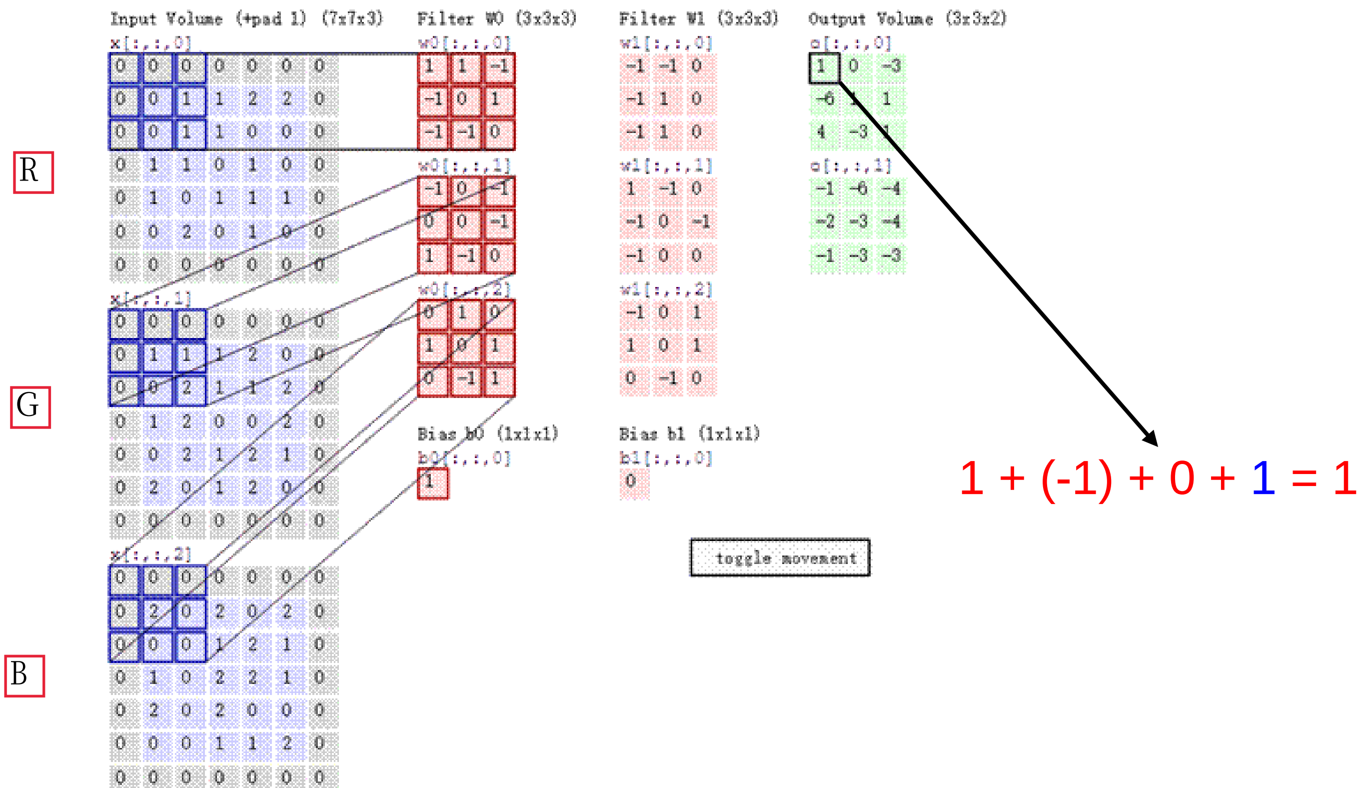
对于一张具有3通道的RGB颜色的图像其大小为6*6*3

卷积核大小也为3*3*3即具有3个颜色通道的卷积核

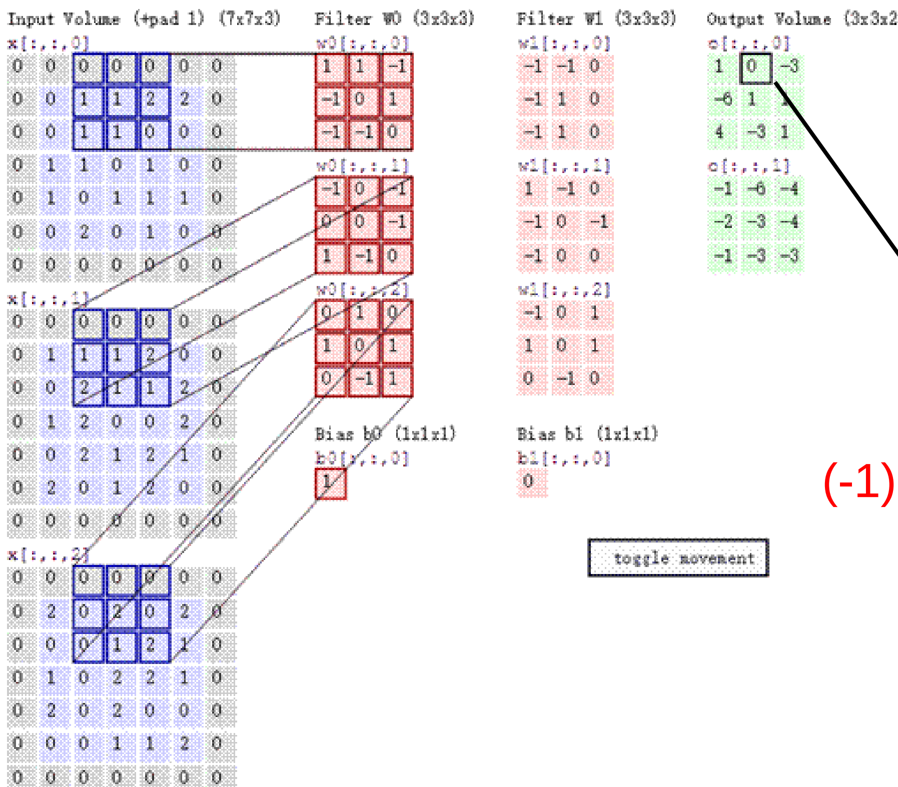
生成一个4*4大小的特征图

卷积神经网络(CNN)-卷积层

- 多通道图像如何卷积？

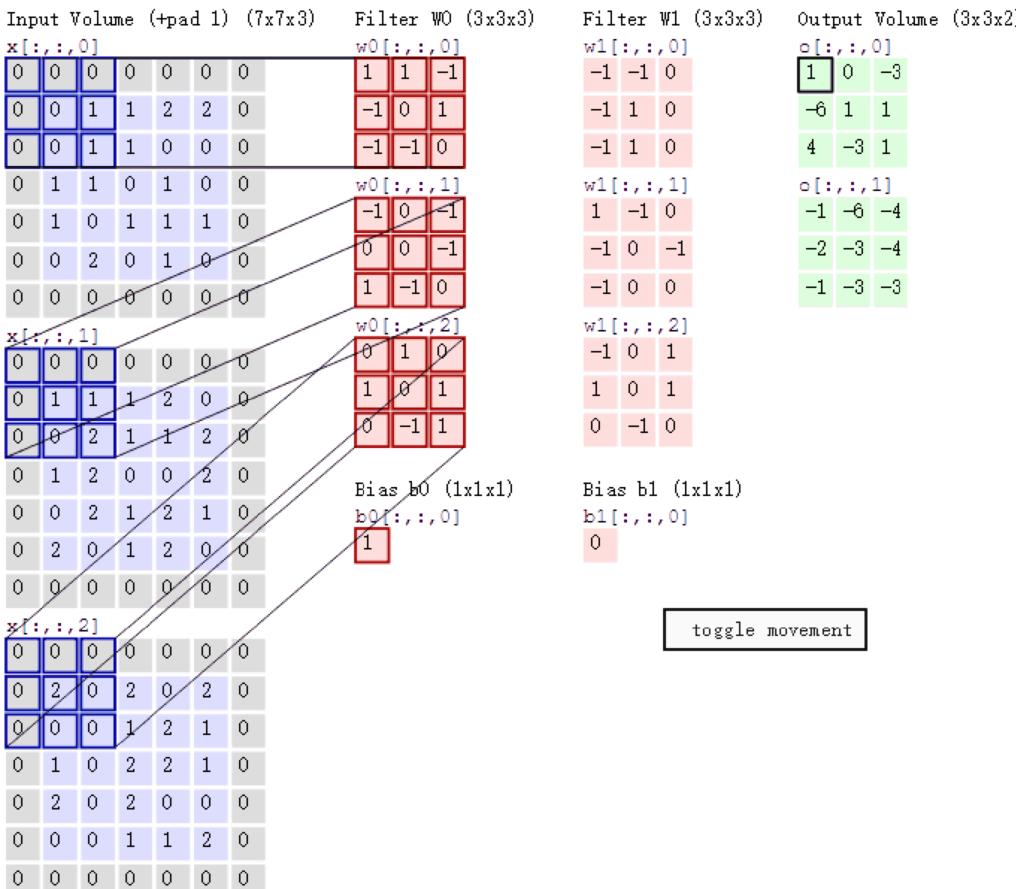


- s = 2


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

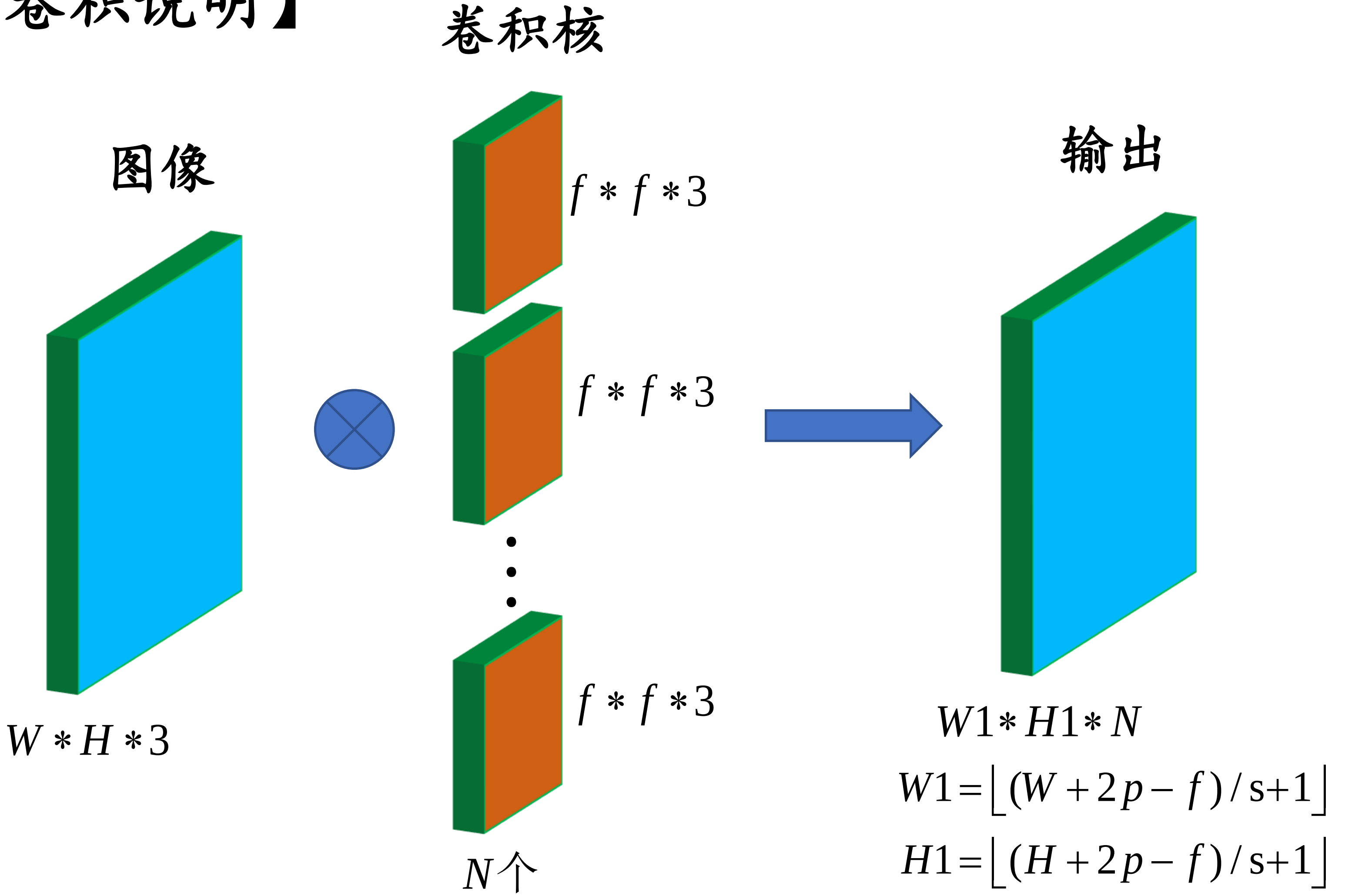
(-1) + (-1) + 1+ 1 = 0

卷积神经网络(CNN)-卷积层

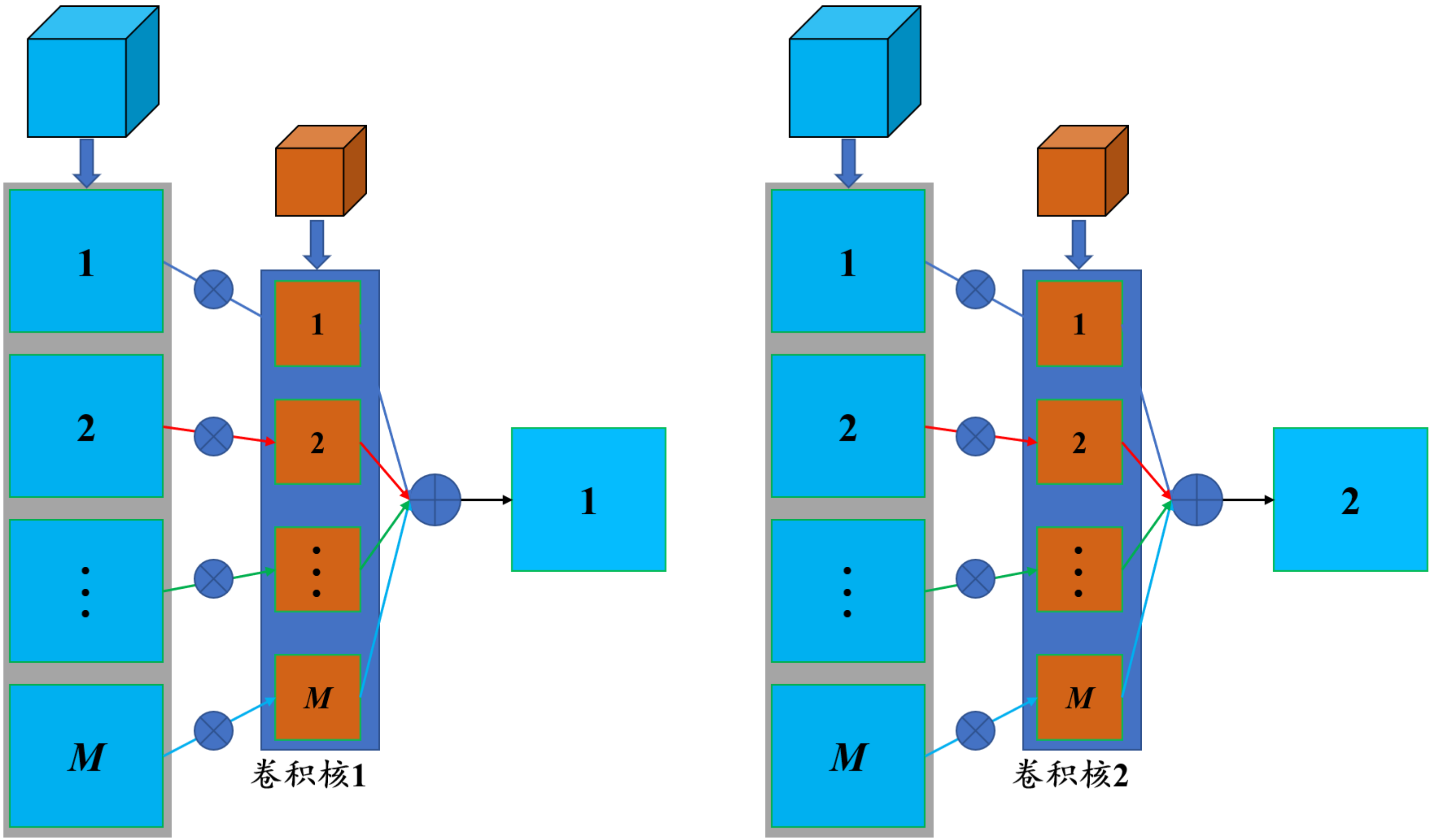


卷积神经网络(CNN)-卷积层

【卷积说明】



卷积神经网络(CNN)-卷积层

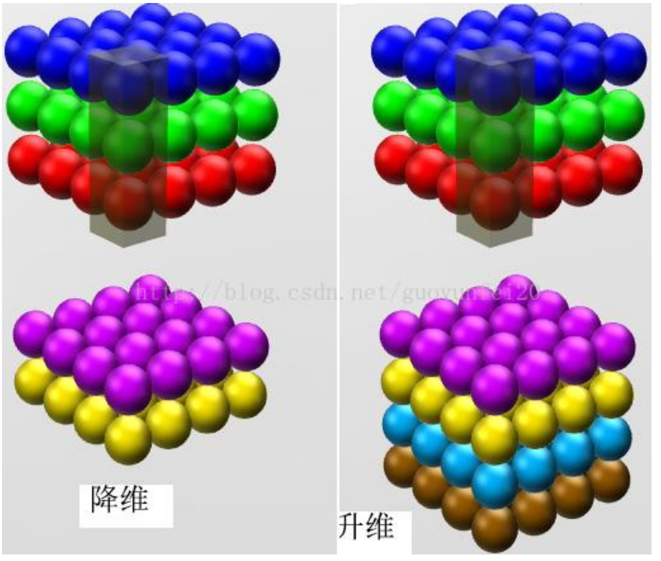


卷积神经网络(CNN)-卷积层

1*1卷积核有什么用处呢？

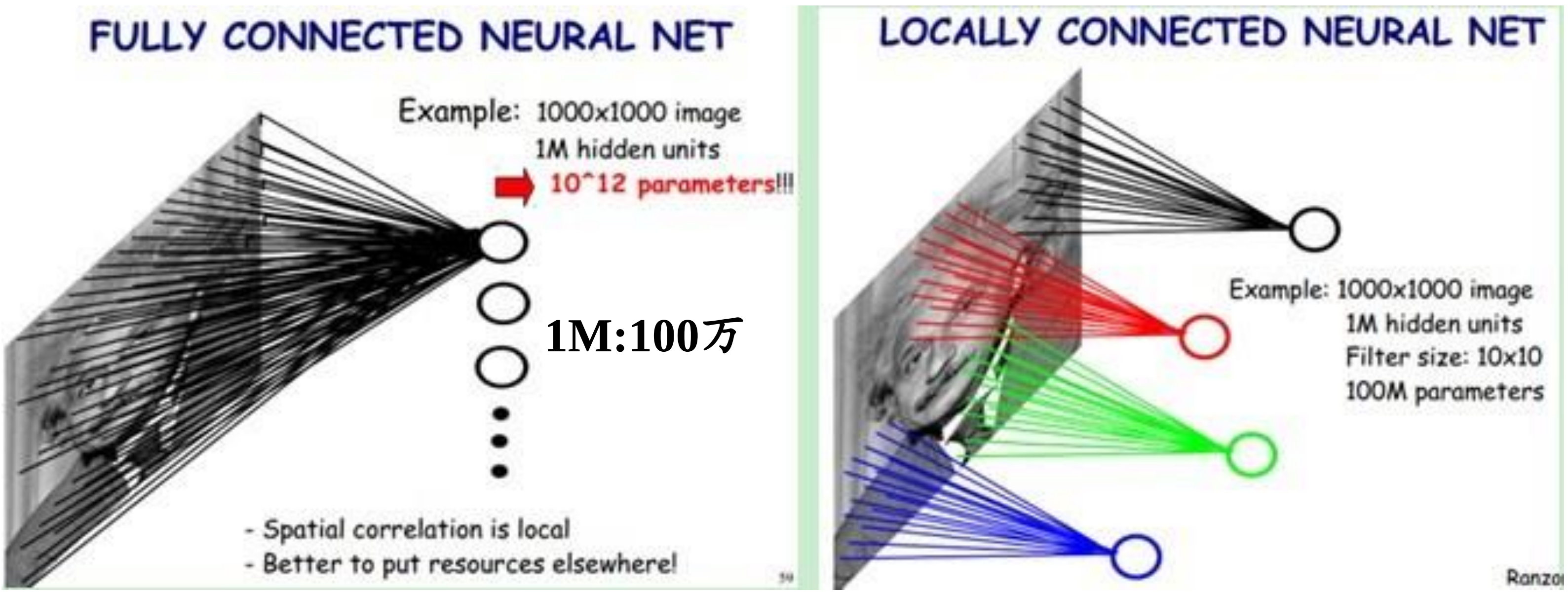
单通道图像： 没用

多通道图像：
1. 降维或升维(GoogleNet、ResNet)
2. 实现跨通道的交互和信息整合



卷积神经网络(CNN)-卷积层

与全连接相比，优势在哪？



1. 局部感知
2. 权重共享，参数数量降低至万分之一(上述例子)

卷积神经网络(CNN)–激励层

把卷积层输出结果做非线性映射

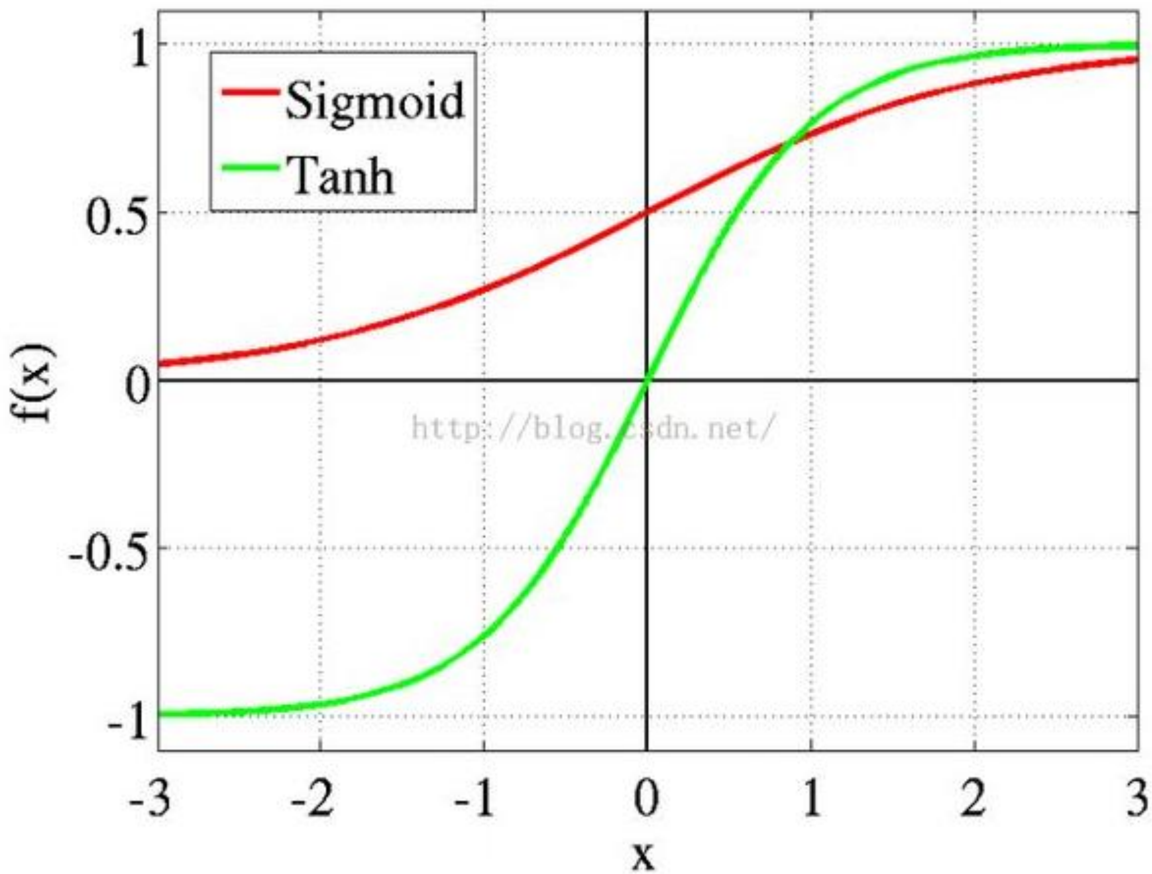
- Sigmoid
- Tanh (双曲正切)
- ReLU
- Leaky ReLU
- ELU
- …

卷积神经网络(CNN)–激励层

- Sigmoid 和 Tanh

$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

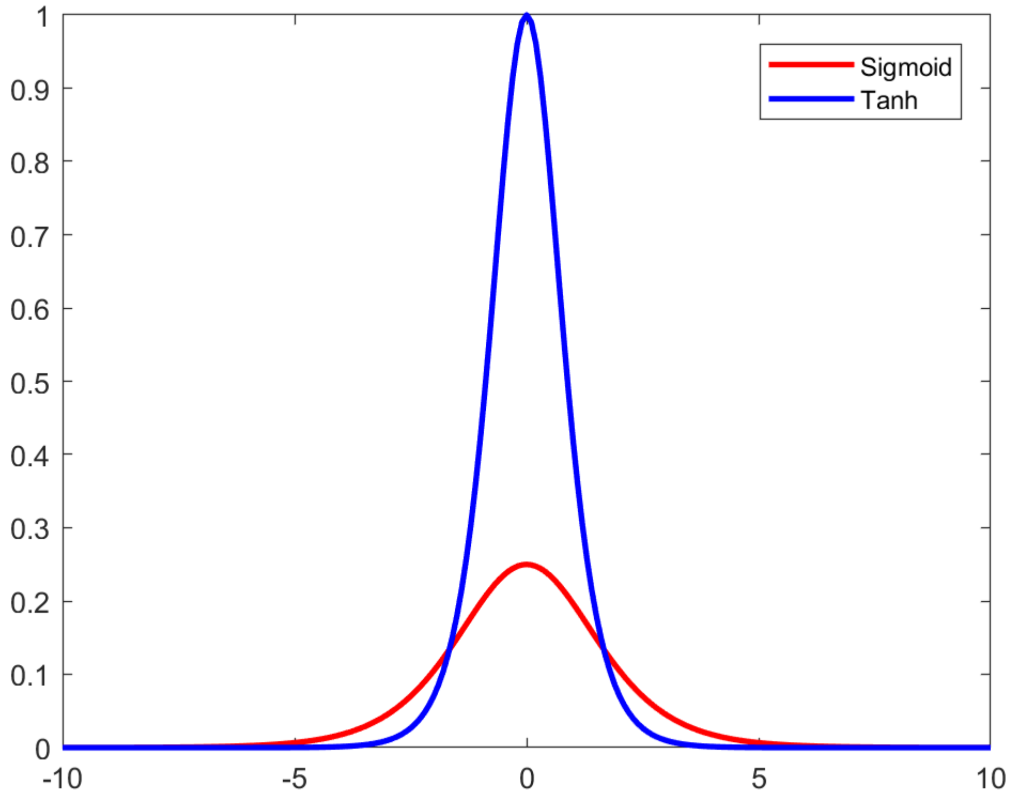
$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



卷积神经网络(CNN)-激励层

$$Sigmoid'(x) = Sigmoid(x)\left(1 - Sigmoid(x)\right)$$

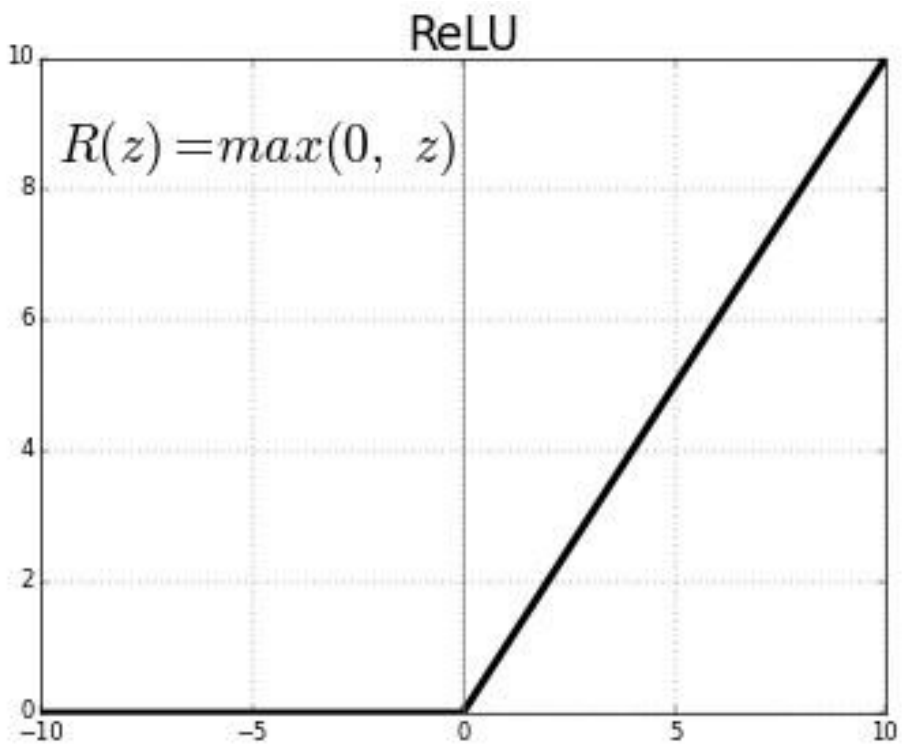
$$Tanh'(x) = 1 - \left(Tanh(x)\right)^2$$



卷积神经网络(CNN)-激励层

- ReLU (Rectified Linear Unit)

  收敛快，求梯度简单，比较脆弱



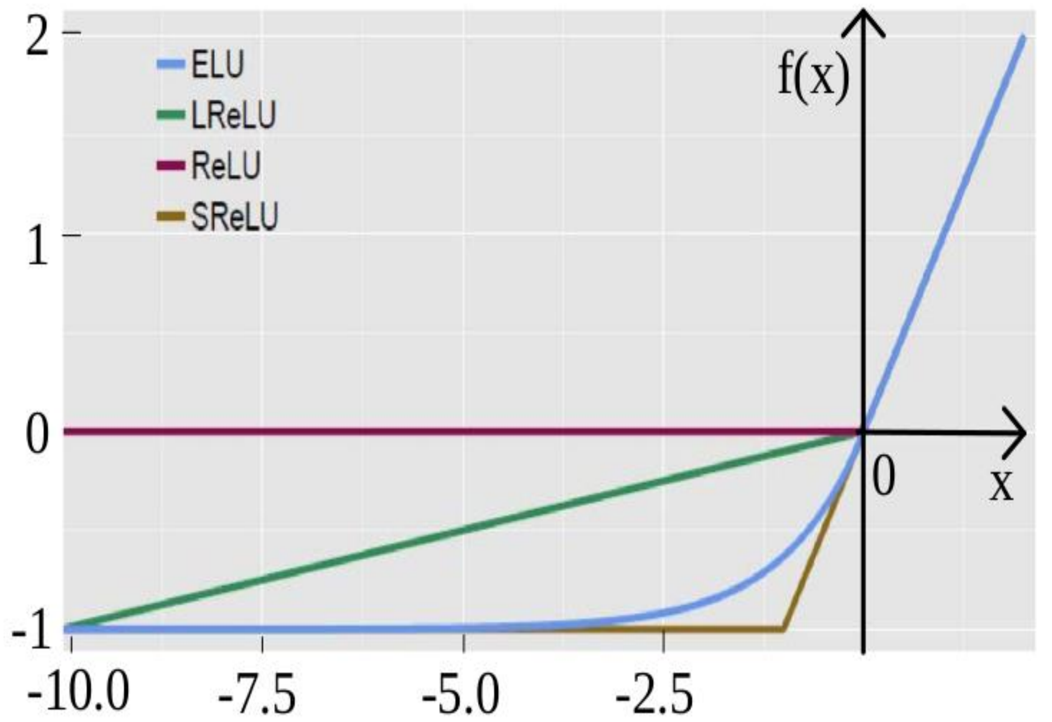
卷积神经网络(CNN)-激励层

- ReLU变形

$$LReLu(\mathrm{x})=\begin{cases} ax & ,x<0 \\ x & ,x\geq 0 \end{cases}$$

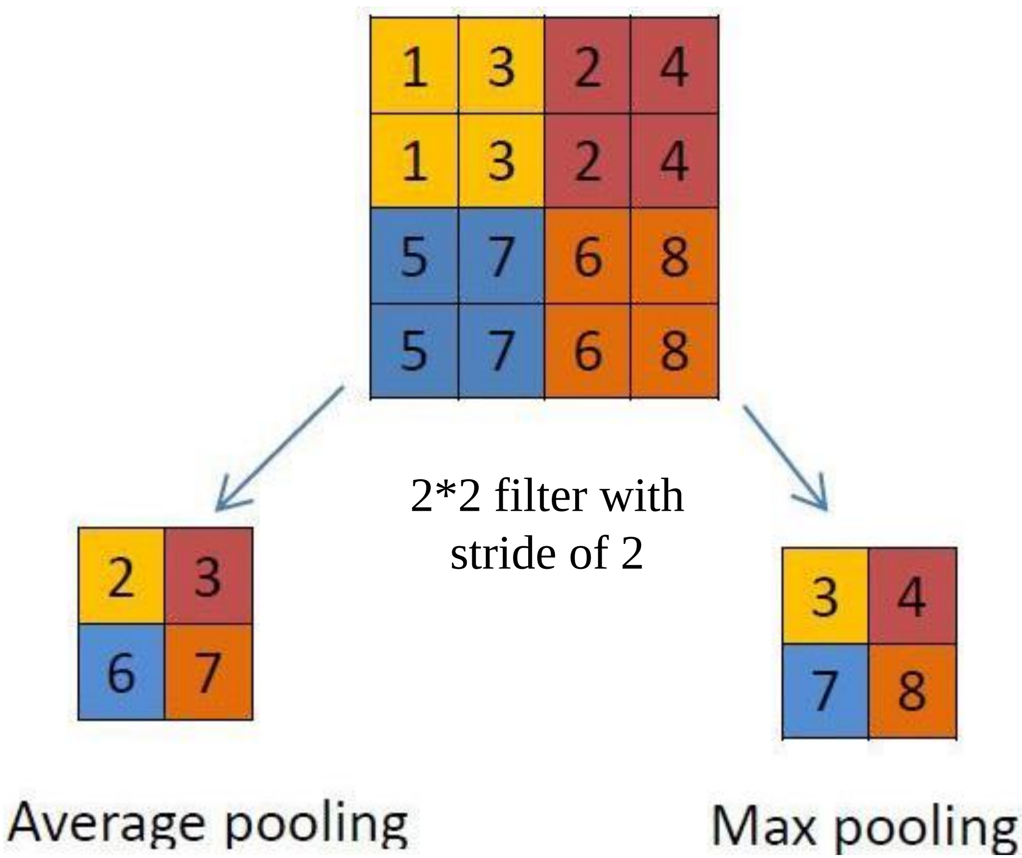
$$EReLu(\mathrm{x})=\begin{cases} a(e^{x}\text{-}1) & ,x<0 \\ x & ,x\geq 0 \end{cases}$$

$$SReLu(\mathrm{x})=\max(\text{-}a,x)$$



卷积神经网络(CNN)–池化层

- 什么是池化?
  - ➢Max-pooling
  - ➢Mean-pooling

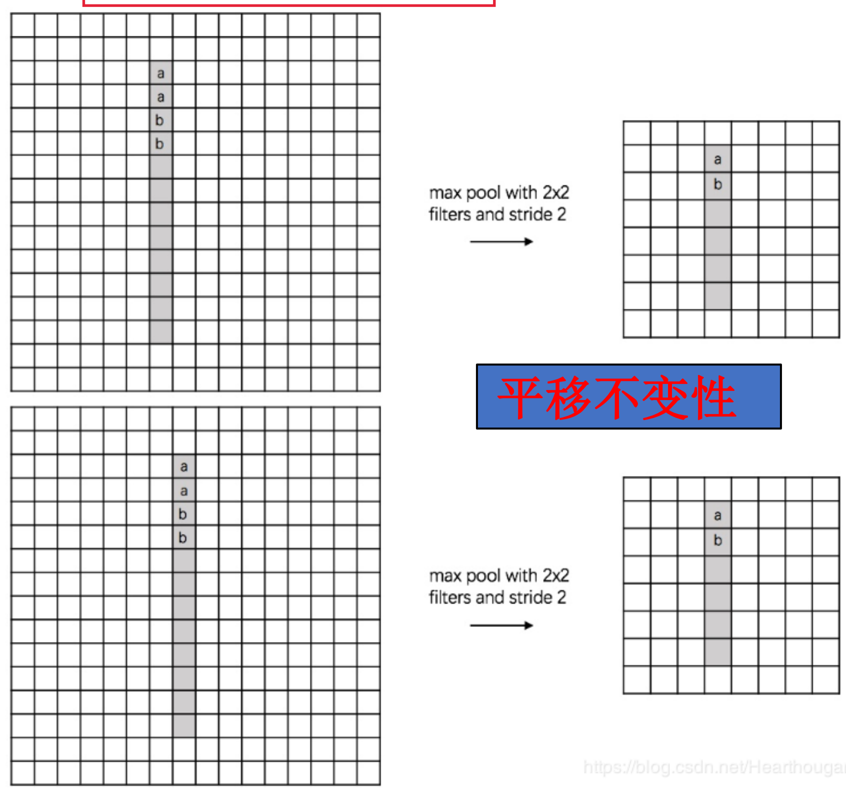


卷积神经网络(CNN)-池化层

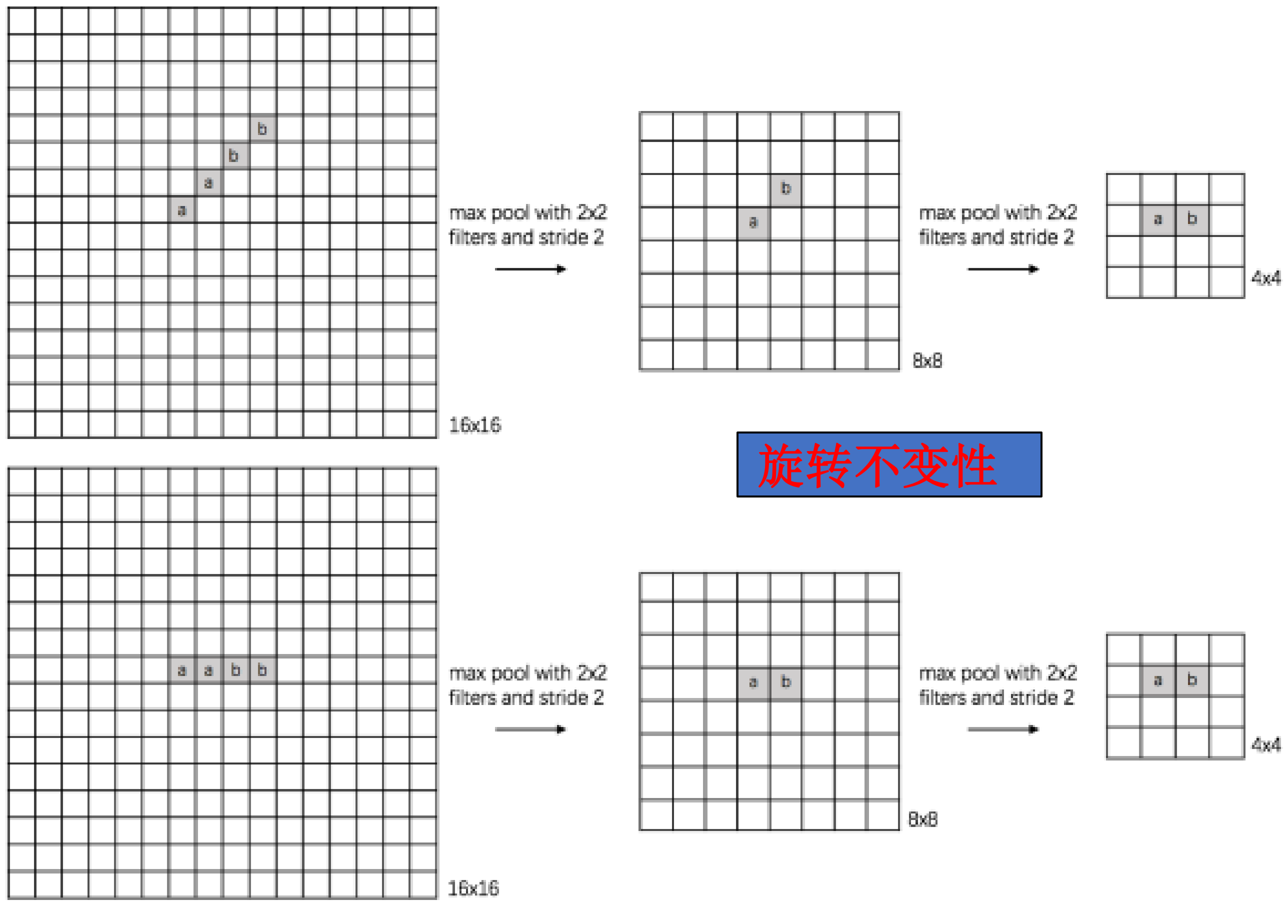
- 池化有什么作用？
  - ➢减少数据量和参数，减少过拟合
  - ➢局部不变性(小幅度变化)

分类的任务：
需要考虑增加池化层；

图像复原/去噪/超分辨：
不需要考虑增加池化层，复原本来意味着图像的每个信息都很重要，因此池化层反而丢掉了某些信息；

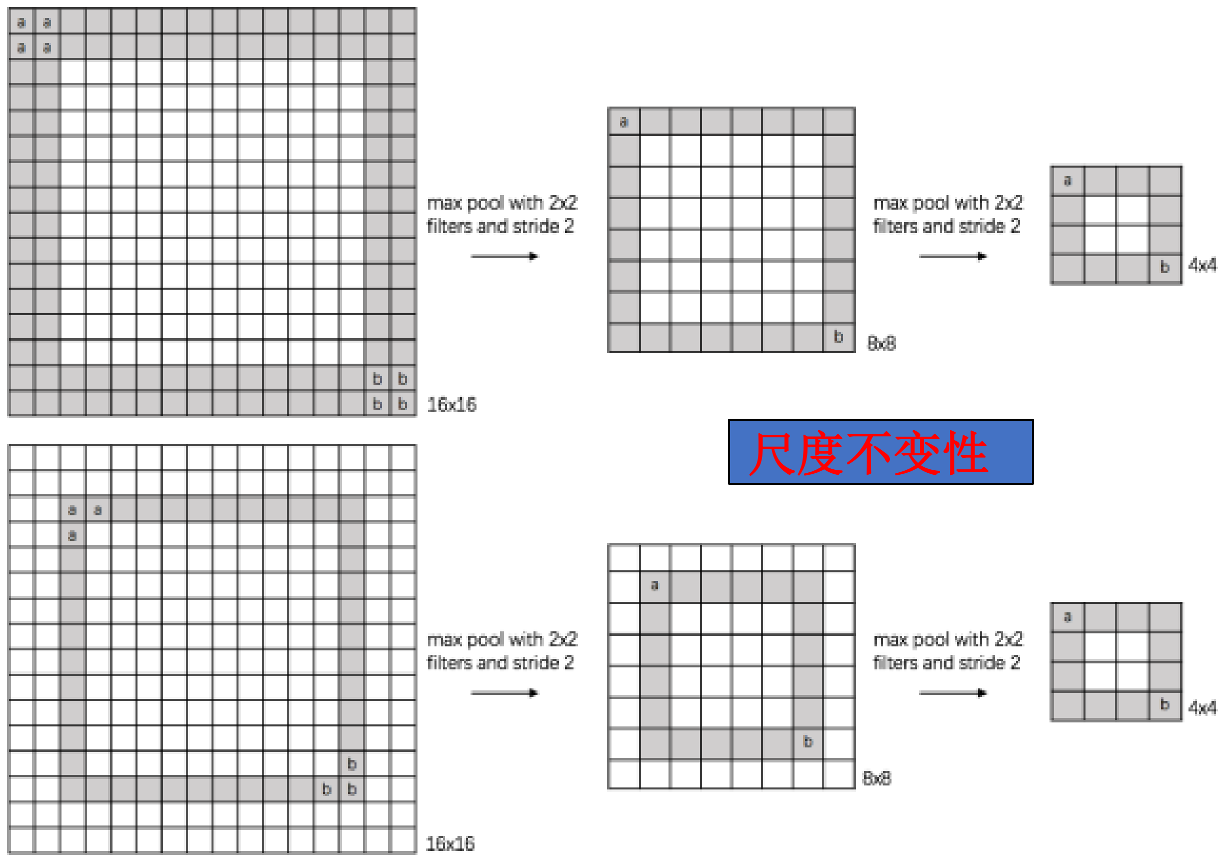


卷积神经网络(CNN)-池化层



旋转不变性

卷积神经网络(CNN)-池化层



尺度不变性

卷积神经网络(CNN)−训练

- **前向传播**

全连接网络：$\mathbf{a}^l = \sigma(\mathbf{z}^l) = \sigma(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l)$

卷积神经网络：

1.卷积层的输出：$\mathbf{A}_j^l = \sigma(\mathbf{Z}_j^l) = \sigma(\sum_{k=1}^{N} \mathbf{A}_k^{l-1} * \mathbf{W}_k^l(j) + b_j^l),\ j = 1, 2, \ldots, M$

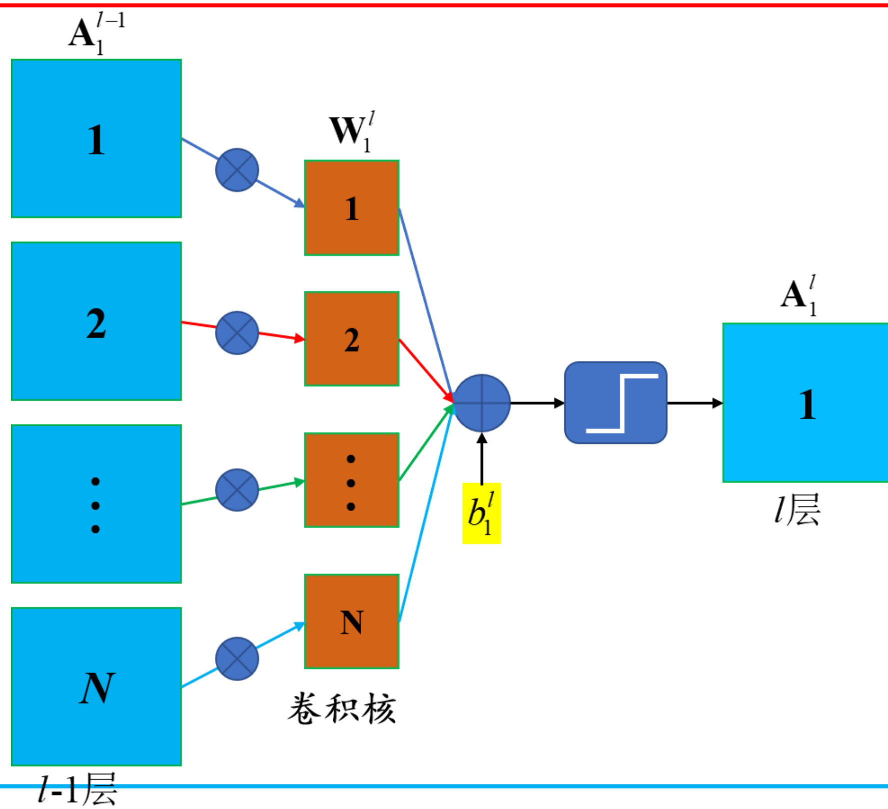
2.全连接层：$\mathbf{a}^l = \sigma(\mathbf{z}^l) = \sigma(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l)$

卷积神经网络(CNN)-训练

卷积层的输出：$\mathbf{A}_j^l=\sigma(\mathbf{Z}_j^l)=\sigma(\sum_{k=1}^{N}\mathbf{A}_k^{l-1}*\mathbf{W}_k^l(j)+\boxed{b_j^l}), j=1,2,\ldots,M$

$\mathbf{W}_k^l\in\mathbb{R}^{f\times f},\mathbf{A}_k^{l-1}\in\mathbb{R}^{W\times H},\mathbf{A}_j^l\in\mathbb{R}^{W1\times H1},\mathbf{b}^l\in\mathbb{R}^{M\times 1}$

$\mathbf{W}_k^l(j)$:第j个卷积核的$\mathbf{W}_k^l$



卷积神经网络(CNN)-训练

- **反向传播**

损失函数：$J(\mathbf{W},\mathbf{b},\mathbf{X},\mathbf{y})$

1. 全连接层： 同ANN网络的更新策略

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{W}^{l}} = \boldsymbol{\delta}^{l}(\mathbf{a}^{l-1})^{T}$$

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{b}^{l}} = \boldsymbol{\delta}^{l}$$

$$\boldsymbol{\delta}^{l} = (\mathbf{W}^{l+1})^{T}\boldsymbol{\delta}^{l+1} \odot \sigma^{'}(\mathbf{z}^{l})$$

卷积神经网络(CNN)-训练

2. 已知池化层的$\boldsymbol{\delta}_k^l$，推导上一层的$\boldsymbol{\delta}_k^{l+1}$

$$\boldsymbol{\delta}_k^l = \begin{pmatrix} 4 & 12 \\ 8 & 16 \end{pmatrix}$$

若池化区域为 2*2，stride 是 2，先还原 δ_k^l 的尺寸：

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 12 & 0 \\ 0 & 8 & 16 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

如果是 MAX，假设我们之前在前向传播时记录的最大值位置分别是左上，右下，右上，左下，则转换后的矩阵为：

$$\begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 12 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 16 & 0 \end{pmatrix}$$

卷积神经网络(CNN)-训练

如果是 Average，则进行平均：转换后的矩阵为：

$$\begin{pmatrix} 1 & 1 & 3 & 3 \\ 1 & 1 & 3 & 3 \\ 2 & 2 & 4 & 4 \\ 2 & 2 & 4 & 4 \end{pmatrix}$$

这样我们就得到了上一层 $\frac{\partial J}{\partial \mathbf{A}_k^{l-1}}$ 的值，要得到 $\boldsymbol{\delta}_k^{l-1}$：

$$\boldsymbol{\delta}_k^{l-1} = \left(\frac{\partial \mathbf{A}_k^{l-1}}{\partial \mathbf{Z}_k^{l-1}}\right)^T \cdot \frac{\partial J}{\partial \mathbf{A}_k^{l-1}} = upsample(\boldsymbol{\delta}_k^{l}) \odot \sigma'(\mathbf{Z}_k^{l-1})$$

卷积神经网络(CNN)-训练

3. 已知卷积层的 $\boldsymbol{\delta}_k^l$，推导上一层的 $\boldsymbol{\delta}_k^{l-1}$

$\mathbf{Z}_k^l$ 和 $\mathbf{Z}_k^{l-1}$ 的关系为：

$$\mathbf{Z}_k^l = \sum_{j=1}^{N} \mathbf{A}_j^{l-1} * \mathbf{W}_j^l(k) + b_k^l = \sum_{j=1,\ j\neq k}^{N} \mathbf{A}_j^{l-1} * \mathbf{W}_j^l(k) + b_k^l + \left(\mathbf{A}_k^{l-1} * \mathbf{W}_k^l(k)\right)$$

$$= \sum_{j=1,\ j\neq k}^{N} \mathbf{A}_j^{l-1} * \mathbf{W}_j^l(k) + b_k^l + \left(\sigma\left(\mathbf{Z}_k^{l-1}\right) * \mathbf{W}_k^l(k)\right)$$

因此，

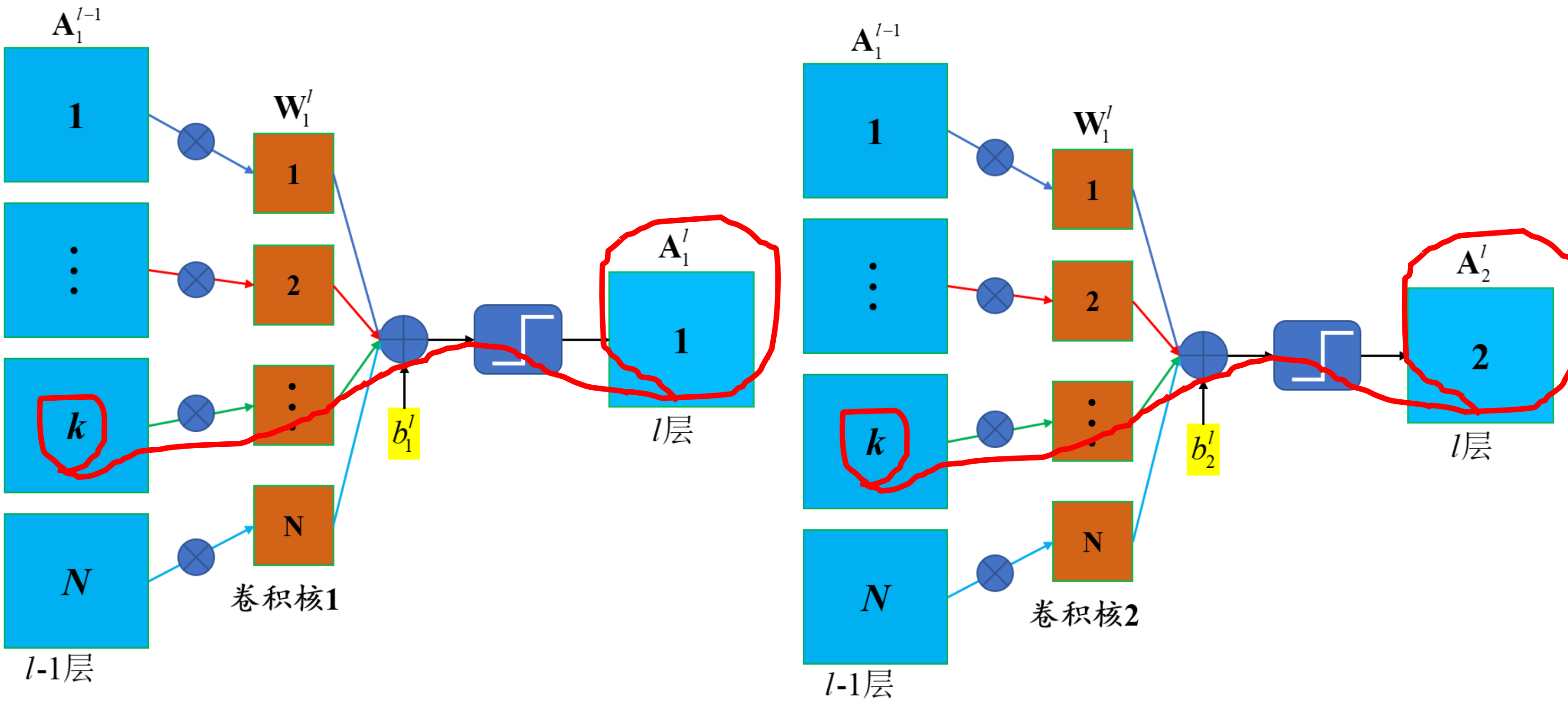
$$\boldsymbol{\delta}_k^{l-1} = \frac{\partial J}{\partial \mathbf{Z}_k^{l-1}} = \left(\frac{\partial \mathbf{Z}_k^l}{\partial \mathbf{Z}_k^{l-1}}\right)^T \frac{\partial J}{\partial \mathbf{Z}_k^l} = \left(\frac{\partial \mathbf{Z}_k^l}{\partial \mathbf{Z}_k^{l-1}}\right)^T \boldsymbol{\delta}_k^l$$

$$= \boldsymbol{\delta}_k^l * rot180(\mathbf{W}_k^l(k)) \odot \sigma'(\mathbf{Z}_k^{l-1})$$

$rot180()$ 就是上下翻转一次，接着左右翻转一次。

参见补充材料

卷积神经网络(CNN)-训练



$$\boldsymbol{\delta}_k^{l-1}=\sum_{j=1}^{M}\boldsymbol{\delta}_j^l * rot180(\mathbf{W}_k^l(j))\odot\sigma'(\mathbf{Z}_k^{l-1})$$

$\mathbf{W}_k^l(j)$:第j个卷积核的$\mathbf{W}_k^l$

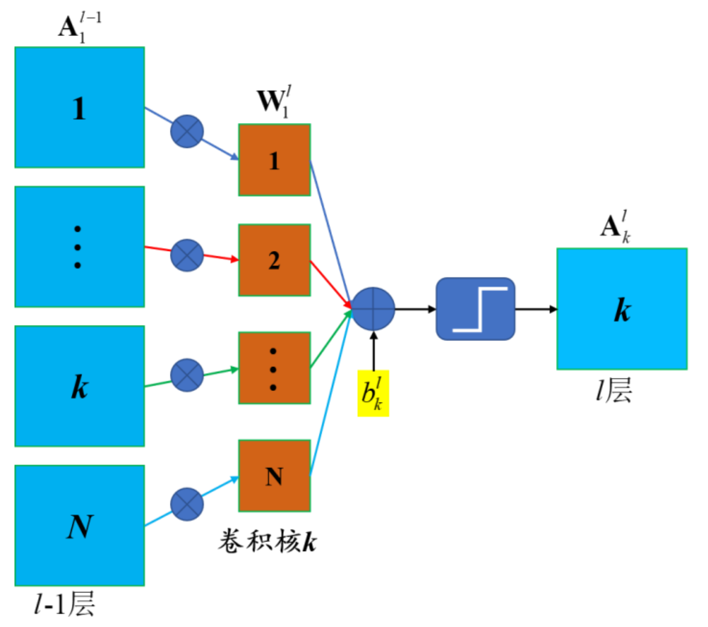
卷积神经网络(CNN)–训练

4. 已知卷积层的$\boldsymbol{\delta}_k^l$，推导$\mathbf{W}_k^l(k)$和b_k^l的梯度

$$\mathbf{Z}_k^l=\sum_{j=1}^{N}\mathbf{A}_j^{l-1}*\mathbf{W}_j^l(k)+b_k^l=\sum_{j=1,\ j\neq k}^{N}\mathbf{A}_j^{l-1}*\mathbf{W}_j^l(k)+b_k^l+\left(\mathbf{A}_k^{l-1}*\mathbf{W}_k^l(k)\right)$$

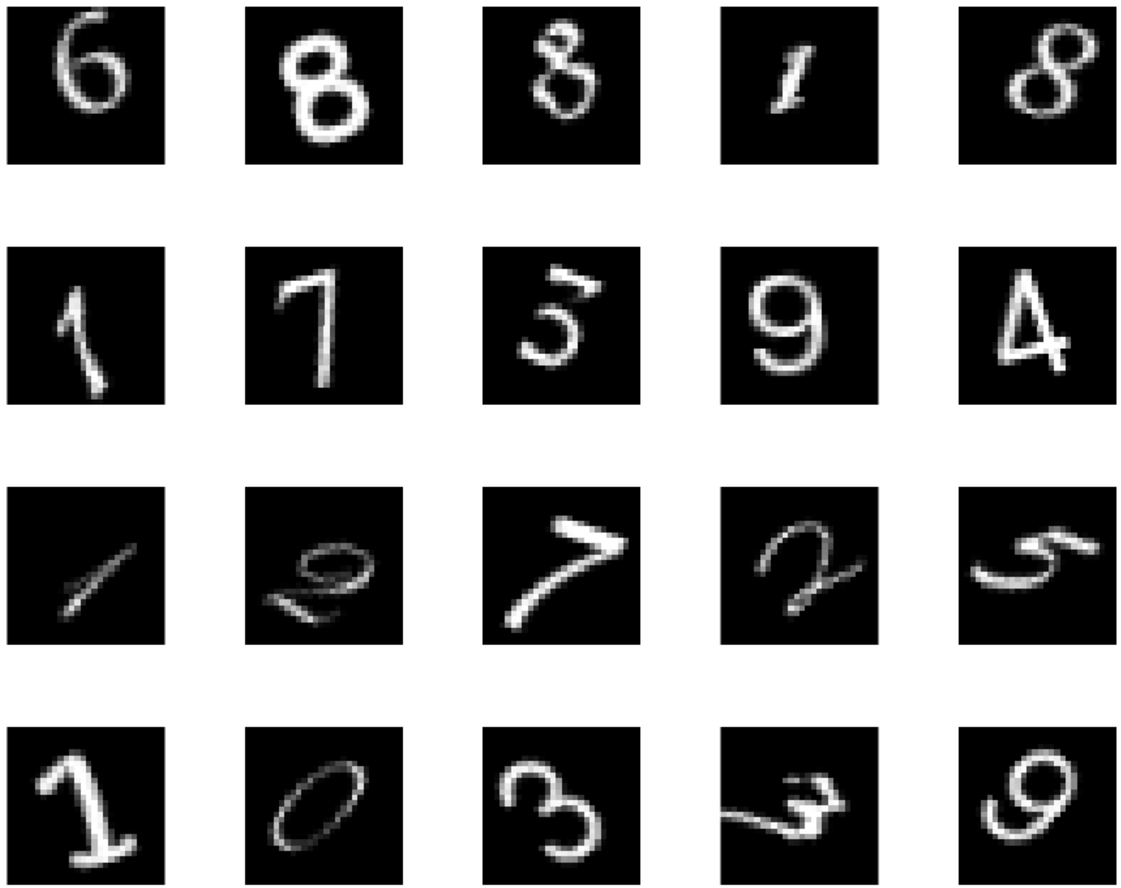
$$\frac{\partial J}{\partial \mathbf{W}_k^l(k)}=\left(\frac{\partial \mathbf{Z}_k^l}{\partial \mathbf{W}_k^l(k)}\right)^T\frac{\partial J}{\partial \mathbf{Z}_k^l}=\mathbf{A}_k^{l-1}*\boldsymbol{\delta}_k^l$$

$$\frac{\partial J}{\partial b_k^l}=\sum_{u,v}\boldsymbol{\delta}_k^l(u,v)$$



Mini-Batch SGD更新权重

卷积神经网络(CNN)-Matlab工具箱



https://www.mathworks.com/help/deeplearning/ref/trainnetwork.html

卷积神经网络(CNN)-Matlab工具箱

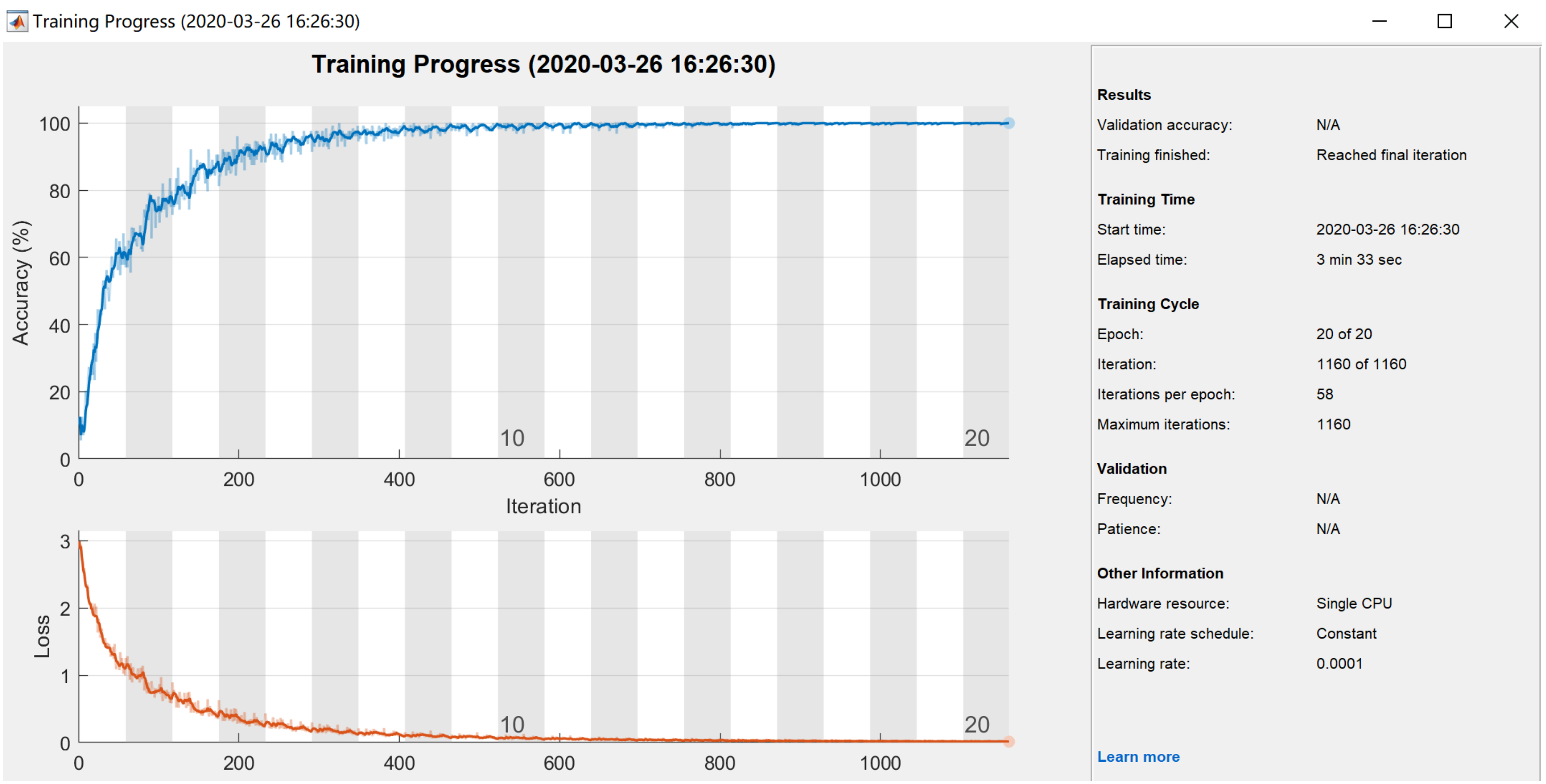
```
layers = [ ...
    imageInputLayer([28 28 1])
    convolution2dLayer(5,20)
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

损失函数与其关联

卷积神经网络(CNN)-Matlab工具箱

```
% 该部分为训练超参数设置
options = trainingOptions('sgdm', ...
    'ExecutionEnvironment','cpu', ...
    'MaxEpochs',50,...
    'InitialLearnRate',1e-3, ...
    'Verbose',false, ...
    'Plots','training-progress');
% 训练部分
net = trainNetwork(imdsTrain,layers,options);
% 预测
YPred = classify(net,imdsTest);
YTest = imdsTest.Labels;
accuracy = sum(YPred == YTest)/numel(YTest);
fprintf('精确度为: %5.2f%%\n',accuracy*100);%打印精度结果
```

卷积神经网络(CNN)-Matlab工具箱



卷积神经网络(CNN)-自实现

```
clc;
clear all;
close all;
%%
load mnist_uint8;

train_x = double(reshape(train_x',28,28,60000))/255;
test_x = double(reshape(test_x',28,28,10000))/255;
train_y = double(train_y');
test_y = double(test_y');
%% Setup
cnn.layers = {
    struct('type', 'i') %input layer
    struct('type', 'conv', 'outputmaps', 20, 'kernelsize', 5) %convolution layer
    struct('type', 'pooling', 'scale', 2) %average-pooling layer
    struct('type', 'conv', 'outputmaps', 10, 'kernelsize', 5) %convolution layer
    struct('type', 'pooling', 'scale', 2) %average-pooling layer
};
cnn = CnnSetup(cnn, train_x, train_y);
opts.MaxEpoch = 10;
opts.BatchSize = 200;
cnn.activation = 'Relu';
opts.lr = 0.01;
```

卷积神经网络(CNN)-自实现

```
%% Training
cnn = CnnTrain(cnn, train_x, train_y, opts);
L = cnn.rL;
figure,plot(1:length(L),L,'r-','LineWidth',2);
xlabel('Number of Iteration'); ylabel('CE Loss');
%% Test
Net = CnnForward(cnn,test_x);
[~, predicts] = max(Net.o);
[~, gt] = max(test_y);
accu = sum(predicts==gt)/ size(test_y, 2);
fprintf('The accuracy is %f.\n',accu);
```

卷积神经网络(CNN)-自实现

```
function net = CnnTrain(net, x, y, opts)
m = size(x, 3);
BatchSize = opts.BatchSize;
MaxEpoch = opts.MaxEpoch;
NumBatches = ceil(m / BatchSize);
N = NumBatches*BatchSize;
x = cat(3,x,x(:,:,1:N-m));
y = cat(2,y,y(:,1:N-m));
%%
net.rL = [];
for i = 1:MaxEpoch
    kk = randperm(N);
    for l = 1 : NumBatches
        batch_x = x(:,:,kk((l - 1) * BatchSize + 1 : l * BatchSize));
        batch_y = y(:,kk((l - 1) * BatchSize + 1 : l * BatchSize));

        net = CnnForward(net, batch_x);
        net = CnnBP(net, batch_y);
        net = Cnnapplygrads(net, opts);
        if isempty(net.rL)
            net.rL(1) = net.L;
        end
        net.rL(end + 1) = net.L;
    end
end
end
```

卷积神经网络(CNN)-自实现

```
function net = CnnForward(net, x)
n = numel(net.layers);
net.layers{1}.a{1} = x;
inputmaps = 1;

for l = 2 : n
    if strcmp(net.layers{l}.type, 'conv')
        for j = 1 : net.layers{l}.outputmaps
            z = zeros(size(net.layers{l - 1}.a{1}) - [net.layers{l}.kernelsize - 1 net.layers{l}.kernelsize - 1 0]);
            for i = 1 : inputmaps
                z = z + convn(net.layers{l - 1}.a{i}, net.layers{l}.k{i}{j}, 'valid');
            end
            switch net.activation
                case 'Sigmoid'
                    net.layers{l}.a{j} = sigm(z + net.layers{l}.b{j});
                case 'Relu'
                    net.layers{l}.a{j} = ReLu(z + net.layers{l}.b{j});
            end
        end
        inputmaps = net.layers{l}.outputmaps;
    elseif strcmp(net.layers{l}.type, 'pooling')
        for j = 1 : inputmaps
            z = convn(net.layers{l - 1}.a{j}, ones(net.layers{l}.scale) / (net.layers{l}.scale ^ 2), 'valid');
            net.layers{l}.a{j} = z(1 : net.layers{l}.scale : end, 1 : net.layers{l}.scale : end, :);
        end
    end
end
```

卷积神经网络(CNN)-自实现

```
net.fv = [];
for j = 1 : numel(net.layers{n}.a)
    sa = size(net.layers{n}.a{j});
    net.fv = [net.fv; reshape(net.layers{n}.a{j}, sa(1) * sa(2), sa(3))];
end

net.o = net.ffW * net.fv + repmat(net.ffb, 1, size(net.fv, 2));
net.o = softmax(net.o);
end
```

卷积神经网络(CNN)-自实现

```
function net = CnnBP(net, y)
    n = numel(net.layers);
    net.e = net.o - y;%error
    % loss function
%     net.L = 1/2* sum(net.e(:) .^ 2) / size(net.e, 2);
    net.L = -sum(sum(y .* log(net.o))) / size(net.e, 2);

    net.od = net.e;%  output delta
    net.fvd = (net.ffW' * net.od);
    if strcmp(net.layers{n}.type, 'pooling')
        switch net.activation
            case 'Sigmoid'
                net.fvd = net.fvd .* (net.fv .* (1 - net.fv));
            case 'Relu'
                net.fvd = net.fvd.*(net.fv > 0);
        end
    end

    sa = size(net.layers{n}.a{1});
    fvnum = sa(1) * sa(2);
    for j = 1 : numel(net.layers{n}.a)
        net.layers{n}.d{j} = reshape(net.fvd(((j - 1) * fvnum + 1) : j * fvnum, :), sa(1), sa(2), sa(3));
    end
```

卷积神经网络(CNN)-自实现

```
%%
for l = (n - 1) : -1 : 1
    if strcmp(net.layers{l+1}.type, 'pooling')
        for j = 1 : numel(net.layers{l}.a)
            switch net.activation
                case 'Sigmoid'
                    net.layers{l}.d{j} = net.layers{l}.a{j} .* (1 - net.layers{l}.a{j}) .* (expand(net.layers{l + 1}.d{j}, ...
                        [net.layers{l + 1}.scale net.layers{l + 1}.scale 1]) / net.layers{l + 1}.scale ^ 2);
                case 'Relu'
                    net.layers{l}.d{j} =(expand(net.layers{l + 1}.d{j}, [net.layers{l + 1}.scale net.layers{l + 1}.scale 1]) ...
                        / (net.layers{l + 1}.scale ^ 2)).*(net.layers{l}.a{j}>0);
            end
        end
    elseif strcmp(net.layers{l+1}.type, 'conv')
        for i = 1 : numel(net.layers{l}.a)
            z = zeros(size(net.layers{l}.a{1}));
            for j = 1 : numel(net.layers{l + 1}.a)
                 z = z + convn(net.layers{l + 1}.d{j}, rot180(net.layers{l + 1}.k{i}{j}), 'full');
            end
            net.layers{l}.d{i} = z;
        end
    end
end
```

卷积神经网络(CNN)-自实现

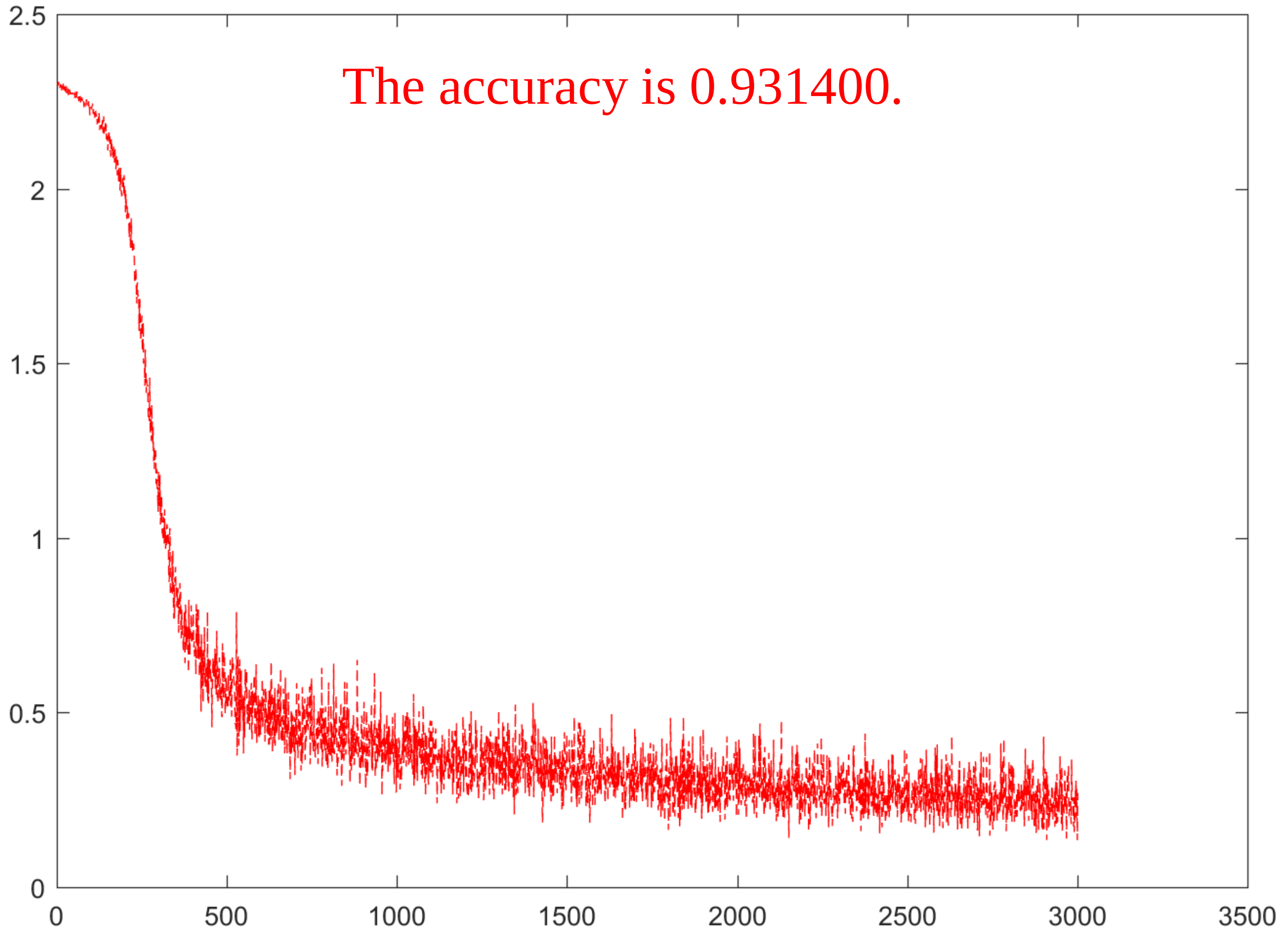
```
    %% calc gradients
    for l = 2 : n
        if strcmp(net.layers{l}.type, 'conv')
            for j = 1 : numel(net.layers{l}.a)
                for i = 1 : numel(net.layers{l - 1}.a)
                    net.layers{l}.dk{i}{j} = convn(flipall(net.layers{l - 1}.a{i}), net.layers{l}.d{j}, 'valid') / size(net.layers{l}.d{j}, 3);
                end
                net.layers{l}.db{j} = sum(net.layers{l}.d{j}(:)) / size(net.layers{l}.d{j}, 3);
            end
        end
    end
    net.dffW = net.od * (net.fv)' / size(net.od, 2);
    net.dffb = mean(net.od, 2);
end
function X = rot180(X)
        X = flipdim(flipdim(X, 1), 2);
    end
```

卷积神经网络(CNN)-自实现

```
function net = Cnnapplygrads(net, opts)
    for l = 2 : numel(net.layers)
        if strcmp(net.layers{l}.type, 'conv')
            for j = 1 : numel(net.layers{l}.a)
                for ii = 1 : numel(net.layers{l - 1}.a)
                    net.layers{l}.k{ii}{j} = net.layers{l}.k{ii}{j} - opts.lr * net.layers{l}.dk{ii}{j};
                end
                net.layers{l}.b{j} = net.layers{l}.b{j} - opts.lr * net.layers{l}.db{j};
            end
        end
    end

    net.ffW = net.ffW - opts.lr * net.dffW;
    net.ffb = net.ffb - opts.lr * net.dffb;
end
```

卷积神经网络(CNN)-自实现



卷积神经网络(CNN)-框架

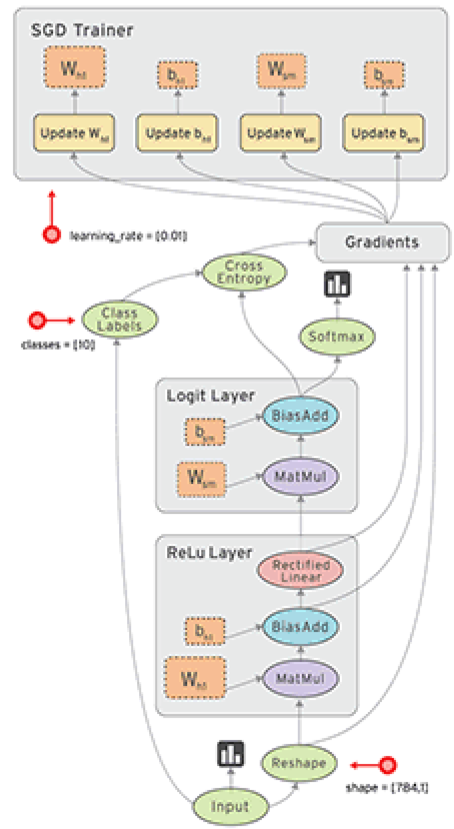
卷积神经网络(CNN)-框架

卷积神经网络(CNN)-框架

框架名稱	組織	API支援語言	Star	Fork	參考網址
Tensorflow	Google	C++, Python, GO, Java	98,120	62,206	https://github.com/tensorflow/tensorflow
Keras	François Chollet	Python	28,880	10,732	https://github.com/keras-team/keras
Caffe	BVLC	C++, Python, Matlab	23,939	14,649	https://github.com/BVLC/caffe
PyTorch	Adam Paszke	Python	14,692	3,290	https://github.com/pytorch/pytorch
CNTK	Microsoft	C++, C#, Python, Java	14,345	3,819	https://github.com/Microsoft/CNTK
MXNet	DMLC	C++, Scala, R, JS, Python, Julia, Matlab, Go	13,816	5,120	https://github.com/apache/incubator-mxnet
DL4J	DeepLearning4Java	Java, Scala	8,807	4,216	https://github.com/deeplearning4j/deeplearning4j
Theano	University of Montreal	Python	8,175	2,454	https://github.com/Theano/Theano
Torch7	Facebook	Lua	7,866	2,276	https://github.com/torch/torch7
Caffe2	Facebook	C++, Python	7,849	1,914	https://github.com/caffe2/caffe2
Paddle	Baidu(百度)	C++, Python	6,818	1,853	https://github.com/PaddlePaddle/Paddle
DSSTNE	Amazon	C++	4,098	676	https://github.com/amzn/amazon-dsstne
tiny-dnn	tiny-dnn	C++	4,044	1,075	https://github.com/tiny-dnn/tiny-dnn
Chainer	Chainer	Python	3,727	982	https://github.com/chainer/chainer
neon	Nervana Systems	Python	3,476	793	https://github.com/NervanaSystems/neon
ONNX	Microsoft	Python	3,251	392	https://github.com/onnx/onnx
BigDL	Intel	Scala	2,431	548	https://github.com/intel-analytics/BigDL
DyNet	Carnegie Mellon University	C++, Python	2,248	540	https://github.com/clab/dynet
brainstorm	IDSIA	Python	1,275	154	https://github.com/IDSIA/brainstorm
CoreML	Apple	Python	1,032	97	https://github.com/apple/coremltools

卷积神经网络(CNN)-框架

- Tensorflow：是Google开源的基于数据流图的机器学习框架，支持python和c++程序开发语言。轰动一时的AlphaGo就是使用tensorflow进行训练的



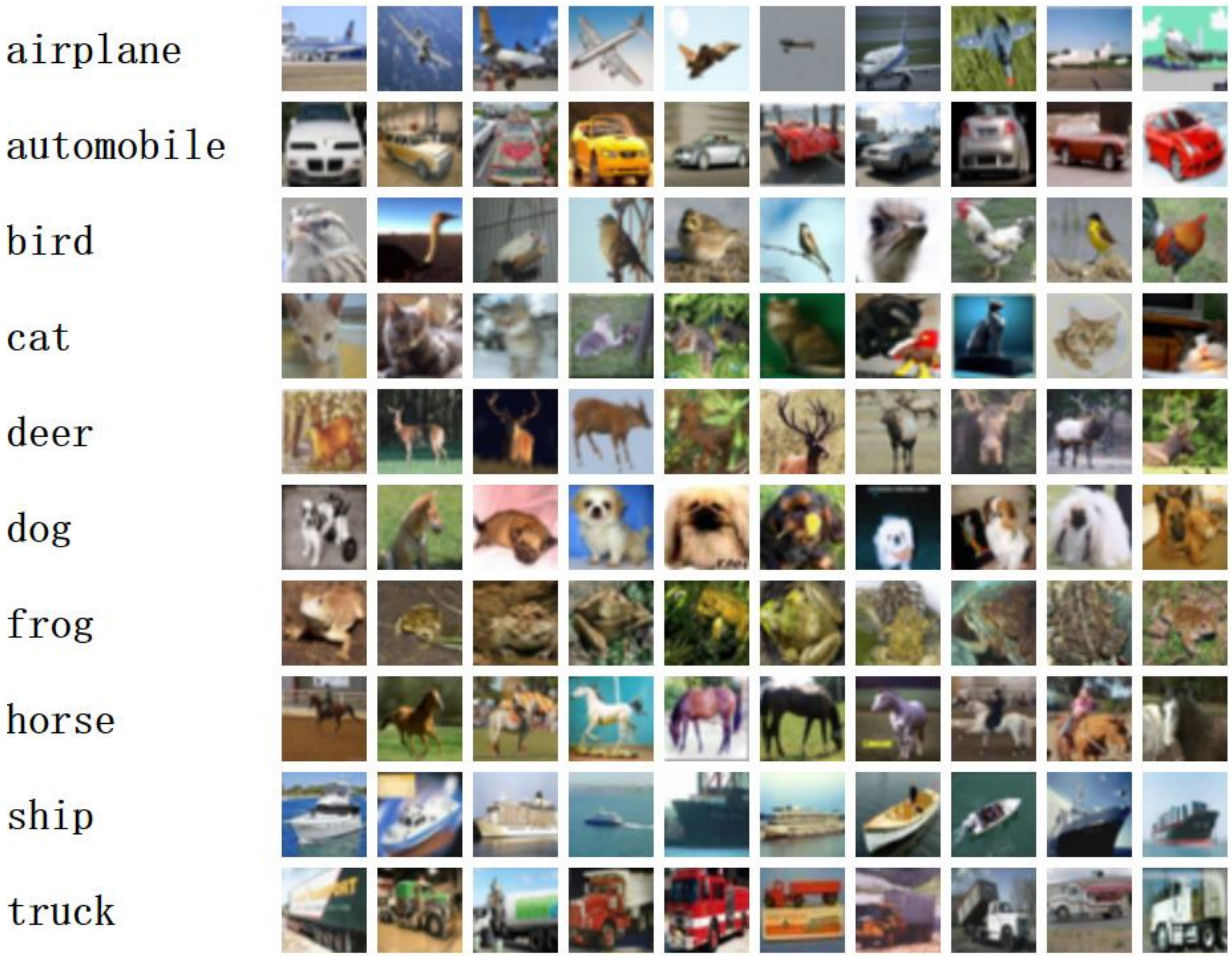
- Tensorflow自学：

中国大学MOOC

《人工智能实践：Tensorflow笔记》

模式识别 -神经网络分类器

Cifar-10数据集(http://www.cs.toronto.edu/~kriz/cifar.html)



模式识别-神经网络分类器

【作业】用神经网络编程实现Cifar-10数据集的分类

说明：

1.ANN,CNN均可(二者都实现加分)

2.编程语言：Matlab/Python

3. 提交实验报告和源代码(**命名规则：神经网络_学号_姓名**)

4. 作业迟交n天，本次作业分数乘以0.98^n。


若干素材取自网络，特此致谢！


谢谢聆听！

模式识别 -神经网络分类器

举例说明为啥要翻转：

假设 a^{l-1} 是 3*3 矩阵，W^l 是 2*2 的卷积核，stride 是 1，则 z^l 是 2*2 矩阵，我们简化 b^l 是 0，即：

$$a^{l-1} * W^l = z^l$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} * \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix}$$

模式识别-神经网络分类器

利用卷积定义，容易得出：

$$z_{11} = a_{11}w_{11} + a_{12}w_{12} + a_{21}w_{21} + a_{22}w_{22}$$
$$z_{12} = a_{12}w_{11} + a_{13}w_{12} + a_{22}w_{21} + a_{23}w_{22}$$
$$z_{21} = a_{21}w_{11} + a_{22}w_{12} + a_{31}w_{21} + a_{32}w_{22}$$
$$z_{22} = a_{22}w_{11} + a_{23}w_{12} + a_{32}w_{21} + a_{33}w_{22}$$

模拟反向求导：

$$\nabla a^{l-1} = \frac{\partial J(W,b)}{\partial a^{l-1}} = \frac{\partial J(W,b)}{\partial z^{l}} \cdot \frac{\partial z^{l}}{\partial a^{l-1}} = \delta^{l} \frac{\partial z^{l}}{\partial a^{l-1}}$$

其中 $\frac{\partial z^{l}}{\partial a^{l-1}}$ 对应上面式子中相关的项 w

比如对 a_{11} 的求导，上面 4 个式子中只有 z_{11} 和 a_{11} 有关联，从而我们有：

$$\nabla a_{11} = \delta_{11}w_{11}$$

同理可得：

$$\nabla a_{12} = \delta_{11}w_{12} + \delta_{12}w_{11}$$

矩阵卷积的形式表示：

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_{11} & \delta_{12} & 0 \\ 0 & \delta_{21} & \delta_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{pmatrix} = \begin{pmatrix} \nabla a_{11} & \nabla a_{12} & \nabla a_{13} \\ \nabla a_{21} & \nabla a_{22} & \nabla a_{23} \\ \nabla a_{31} & \nabla a_{32} & \nabla a_{33} \end{pmatrix}$$

为了符合梯度计算，我们在误差矩阵周围填充了一圈 0，此时我们将卷积核翻转后和反向传播的梯度误差进行卷积，就得到了前一次的梯度误差。这个例子直观的介绍了为什么对含有卷积的式子反向传播时，卷积核要翻转 180 度的原因。