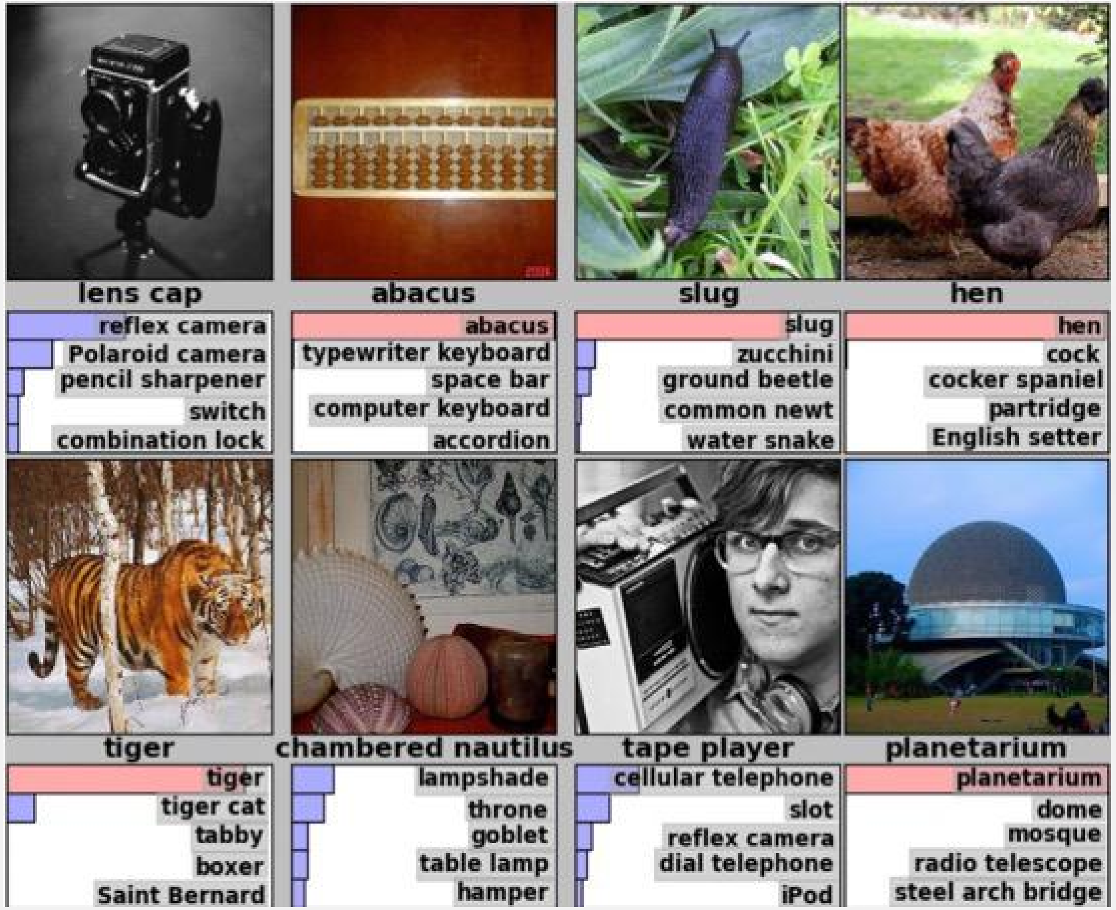
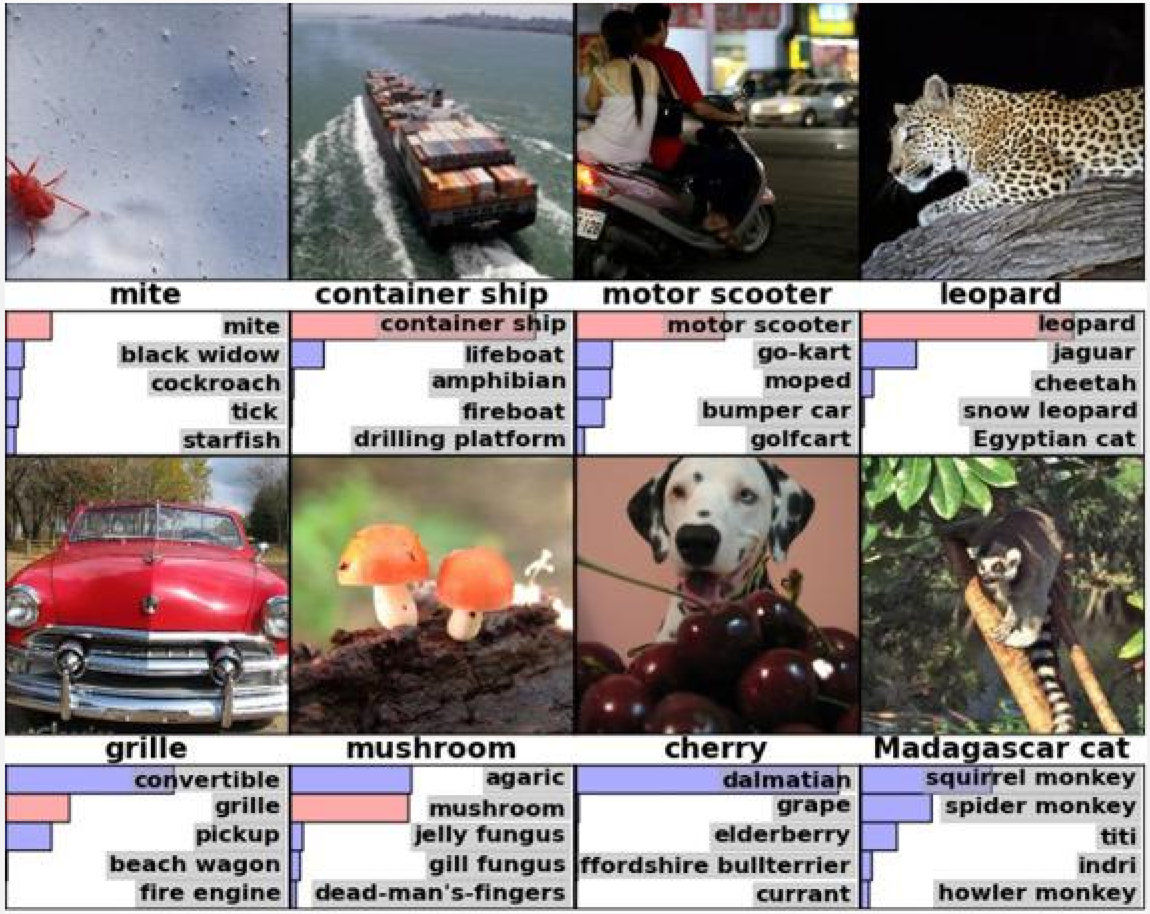
卷积神经网络

张江

北京师范大学系统科学学院
集智俱乐部，集智学园

硐堡的图像识别

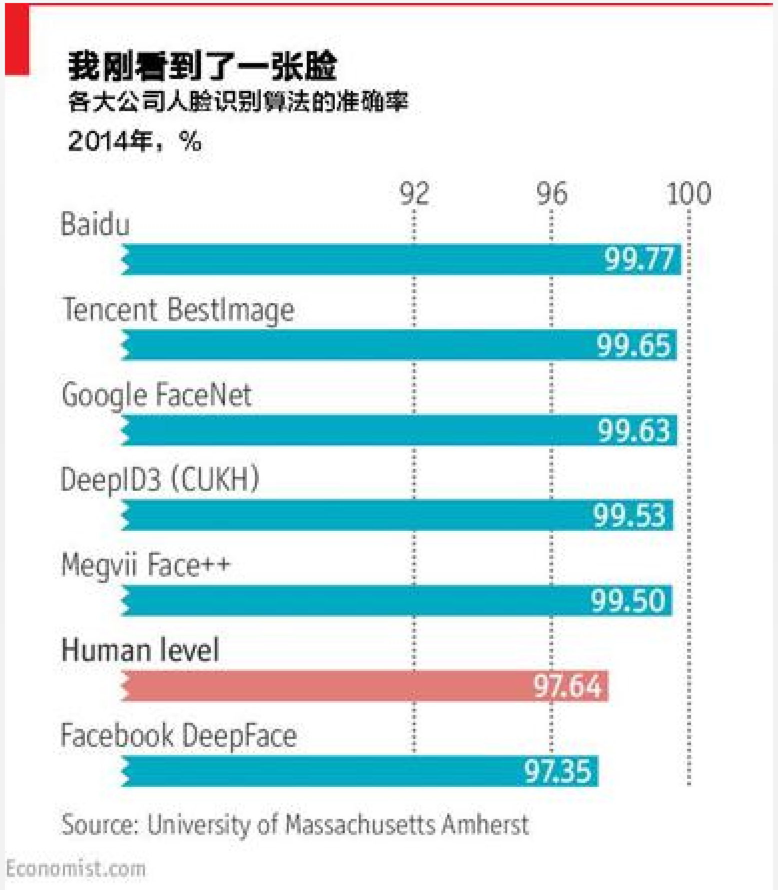


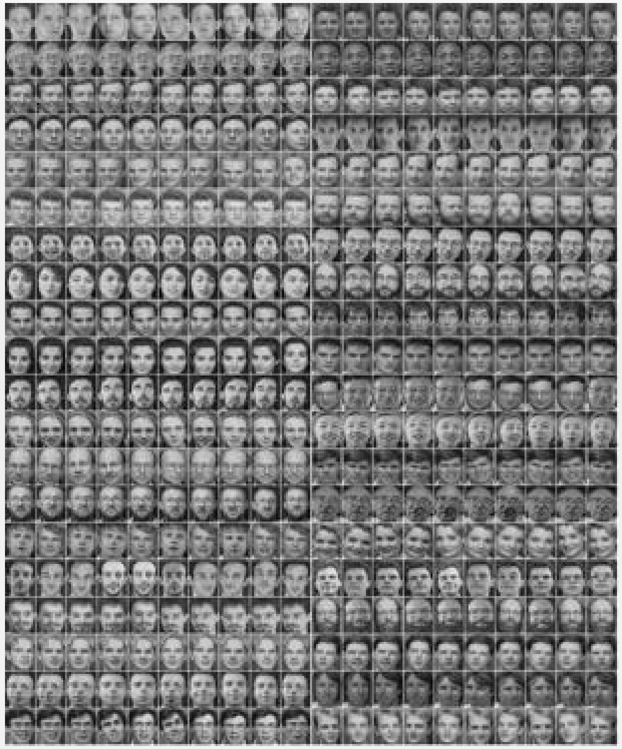


ImageNet

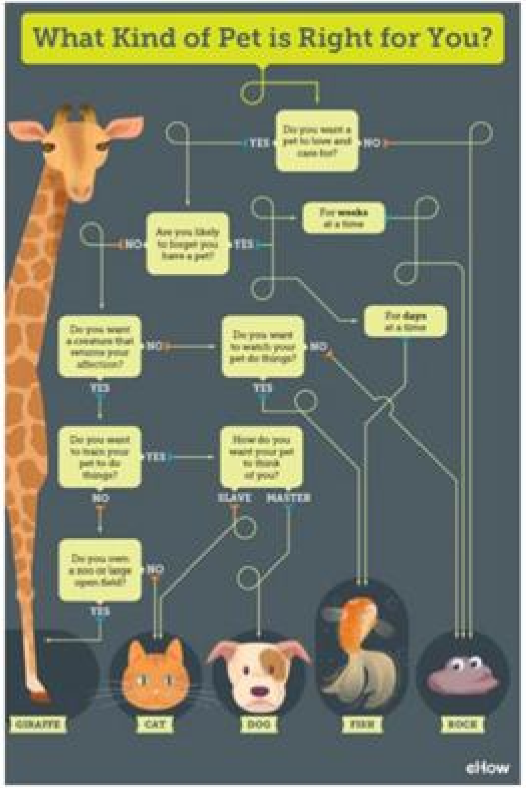
硐堡的图像识别


2017.01.06
ROBOT
VS
HUMAN


我刚看到了一张脸
各大公司人脸识别算法的准确率
2014年，%
92
96
100
Baidu
99.77
Tencent BestImage
99.65
Google FaceNet
99.63
DeepID3 (CUKH)
99.53
Megvii Face++
99.50
Human level
97.64
Facebook DeepFace
97.35
Source: University of Massachusetts Amherst
Economist.com



甚至图像理解……



从左数来第1只是哪种动物?(长颈鹿)
4/10

图中的鬣狗是在哪种场景中?(室内)
9/10

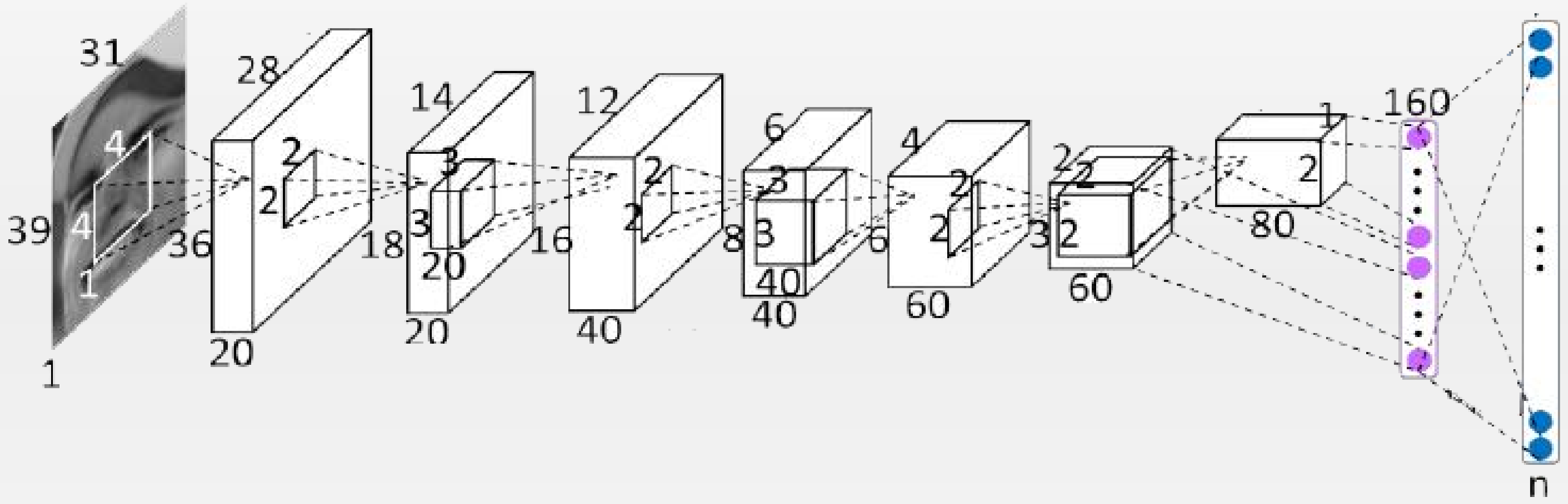
图中的长颈鹿是在哪种场景中?(室内)
9/10

图中的长颈鹿是在哪种场景中?(雪地)
7/10

图中的天竺鼠是在哪种场景中?(雪地)
5/10

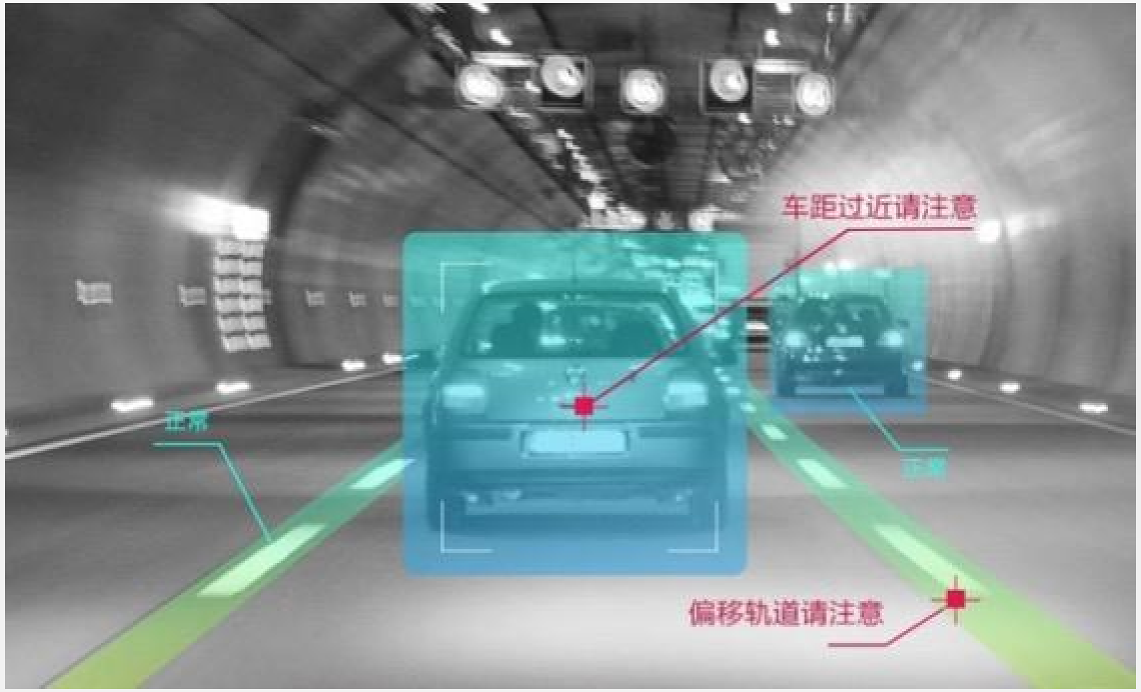
Brain of things公开比赛

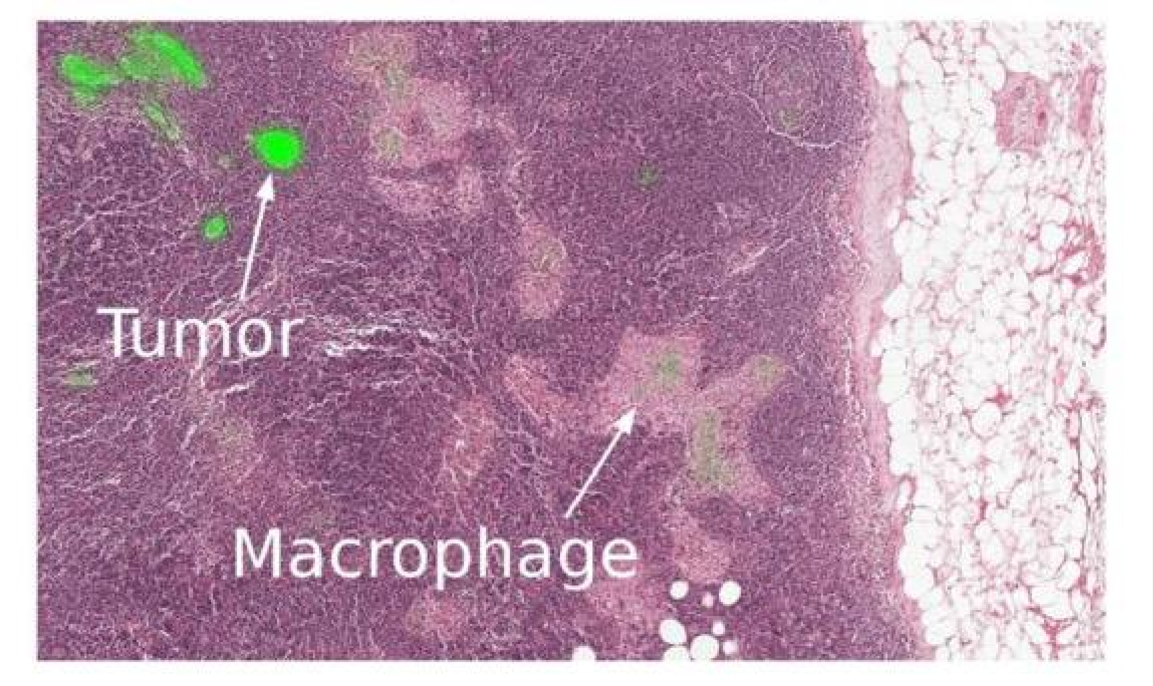
卷积神经网络(CNN)：处理图像数据



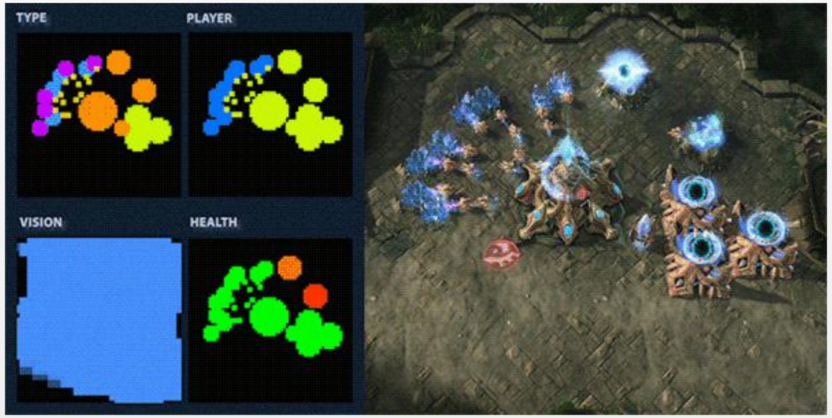
CNN可以应付图像数据中的平移、旋转等空间不变性

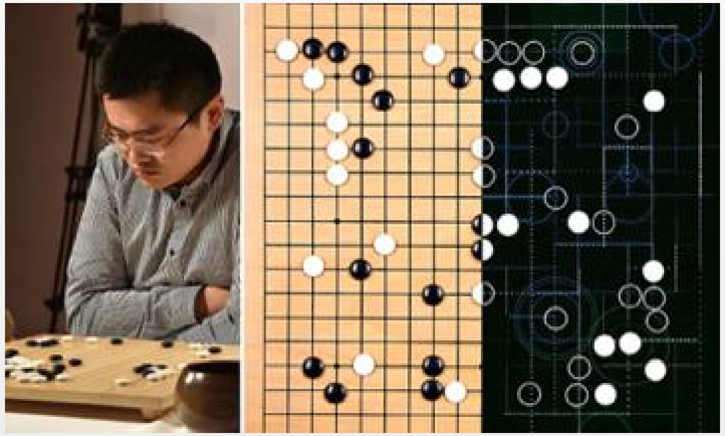
大量应用场景

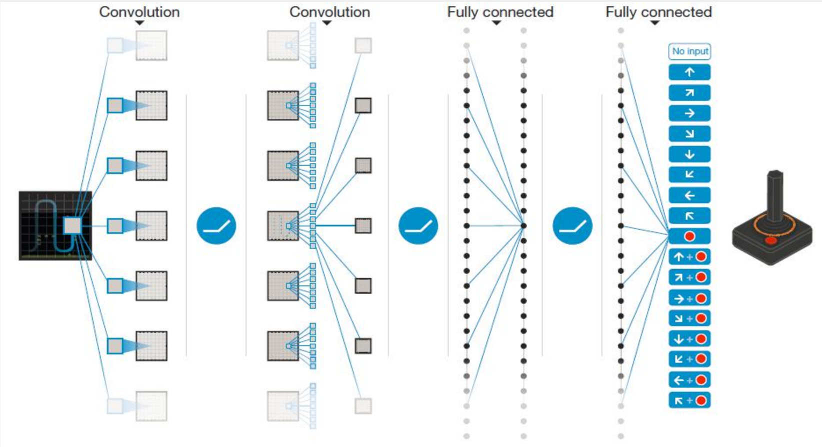


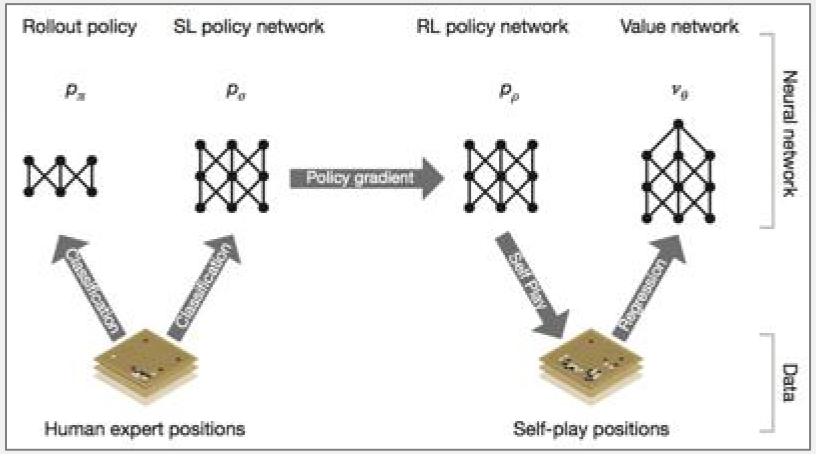


重大突破的基础

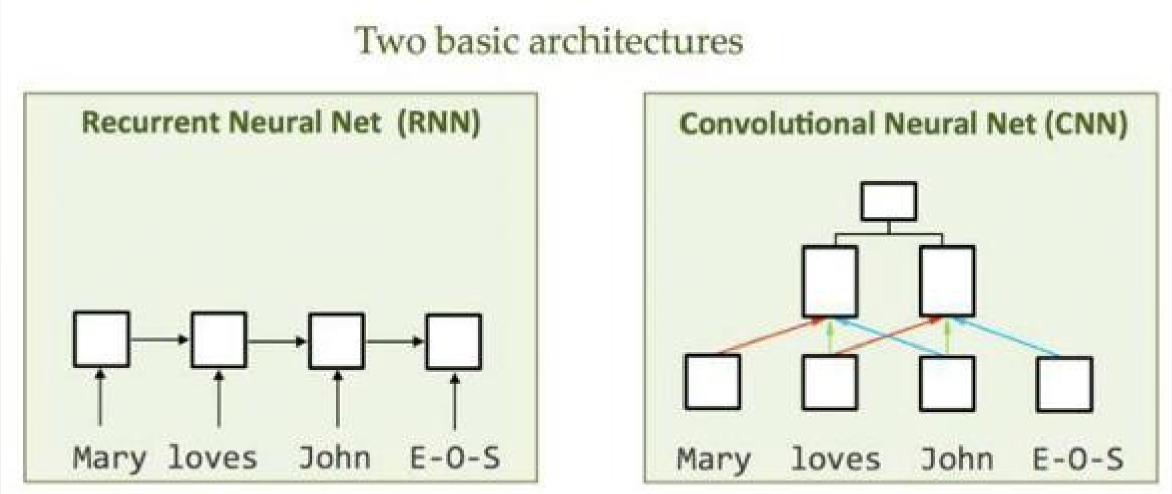

TYPE
PLAYER
VISION
HEALTH



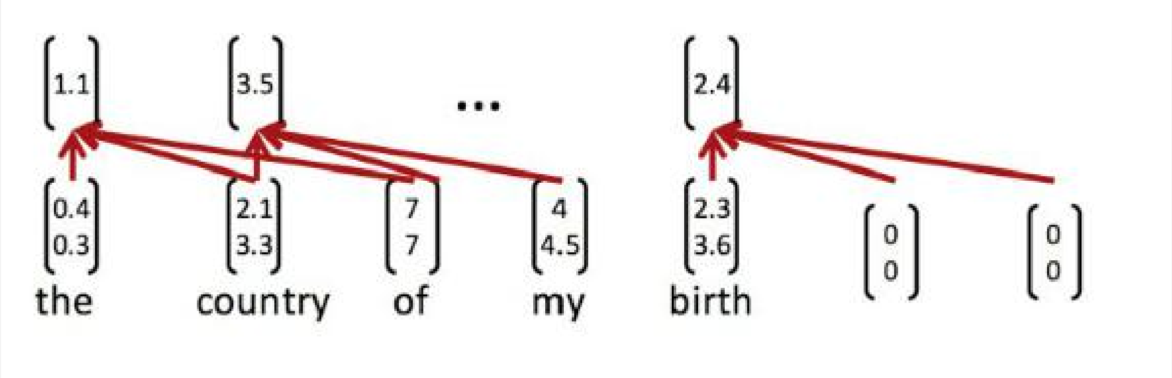

Convolution
Convolution
Fully connected
Fully connected
No input


Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Classification
Classification
Self Play
Regression
Neural network
Data
Human expert positions
Self-play positions

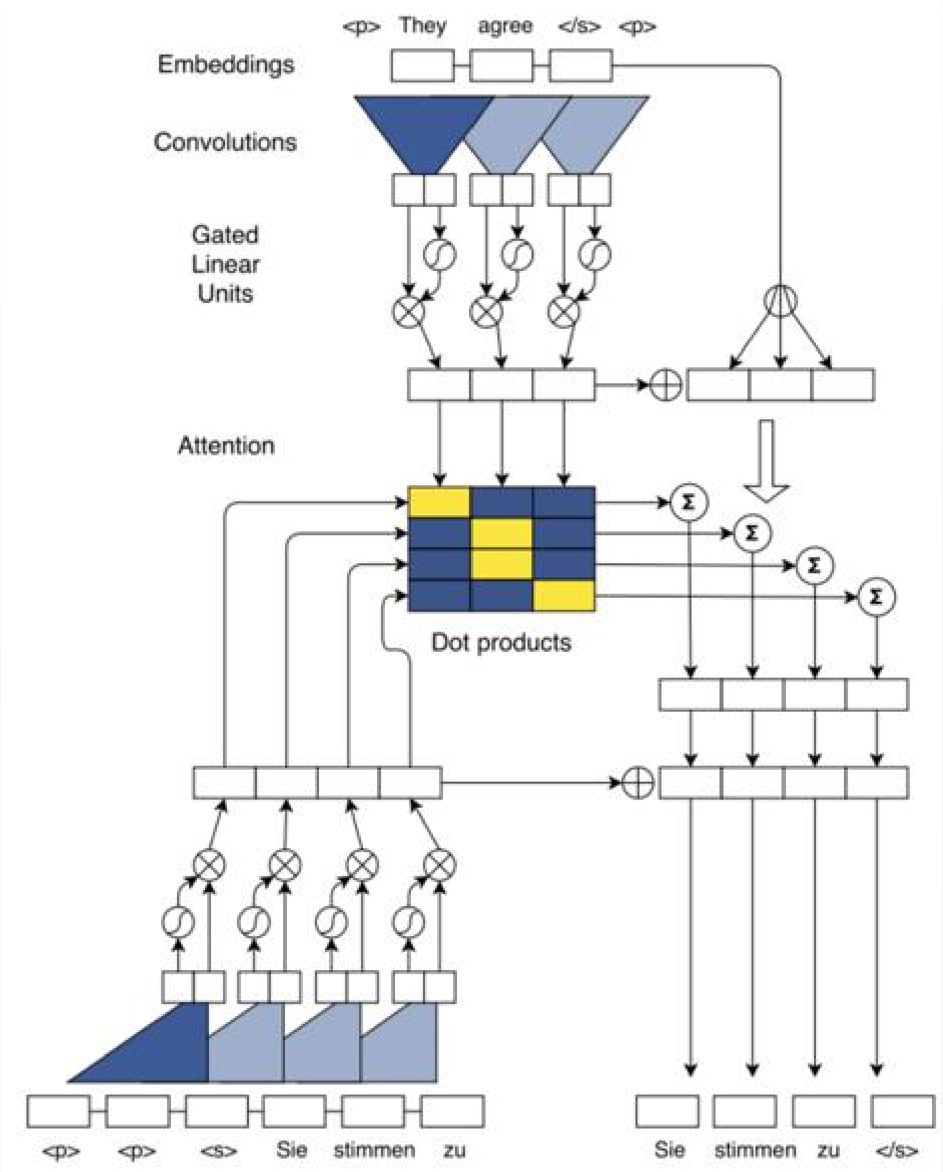
甚至抢了RNN的饭碗



Two basic architectures
Recurrent Neural Net (RNN)
Mary loves John E-O-S
Convolutional Neural Net (CNN)
Mary loves John E-O-S

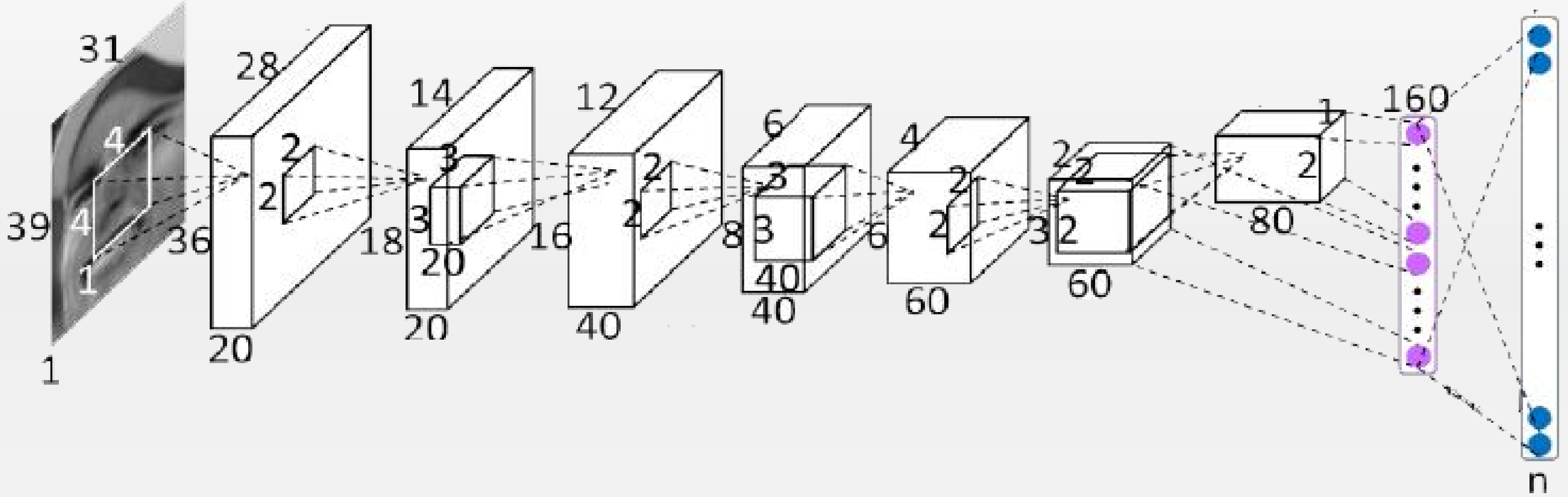


1.1
3.5
...
2.4
0.4 0.3
2.1 3.3
7 7
4 4.5
2.3 3.6
0 0
0 0
the
country
of
my
birth



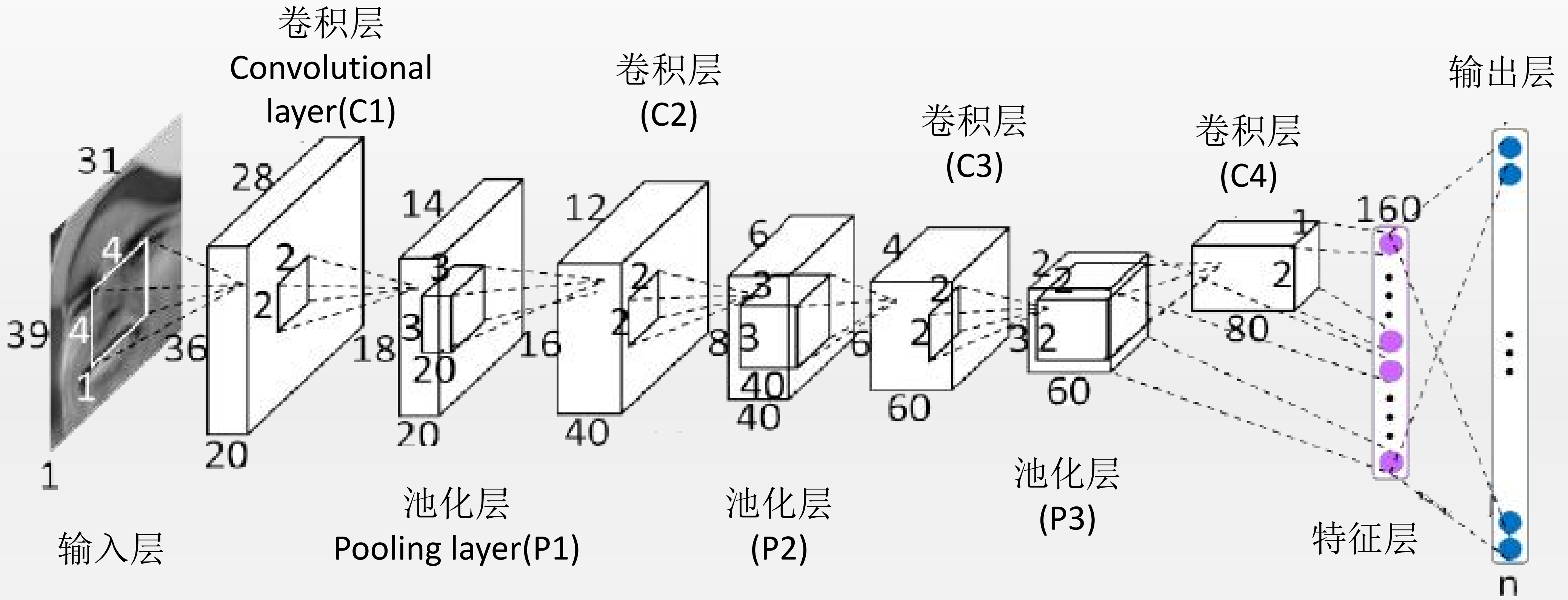
<p> They agree </s> <p>
Embeddings
Convolutions
Gated Linear Units
Attention
Dot products
<p> <p> <s> Sie stimmen zu
Sie stimmen zu </s>

卷积神经网络



- 卷积神经网络包含多个层，每层的神经元都会排布成三维的立方体

卷积神经网络

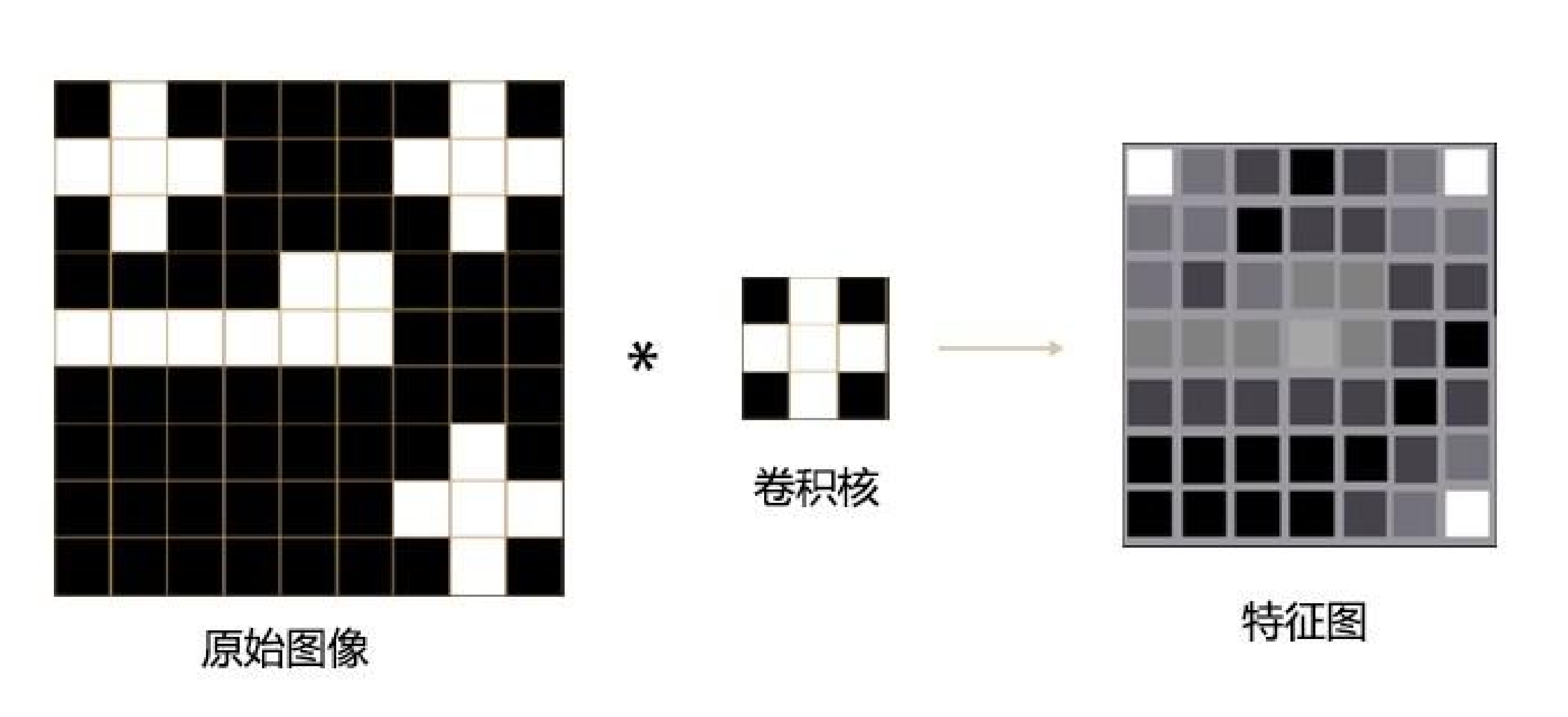


- 卷积神经网络包含多个层，每层的神经元都会排布成三维的立方体


找你妹
昵图网 www.nipic.com
By:swzhen No.20130824095410415348

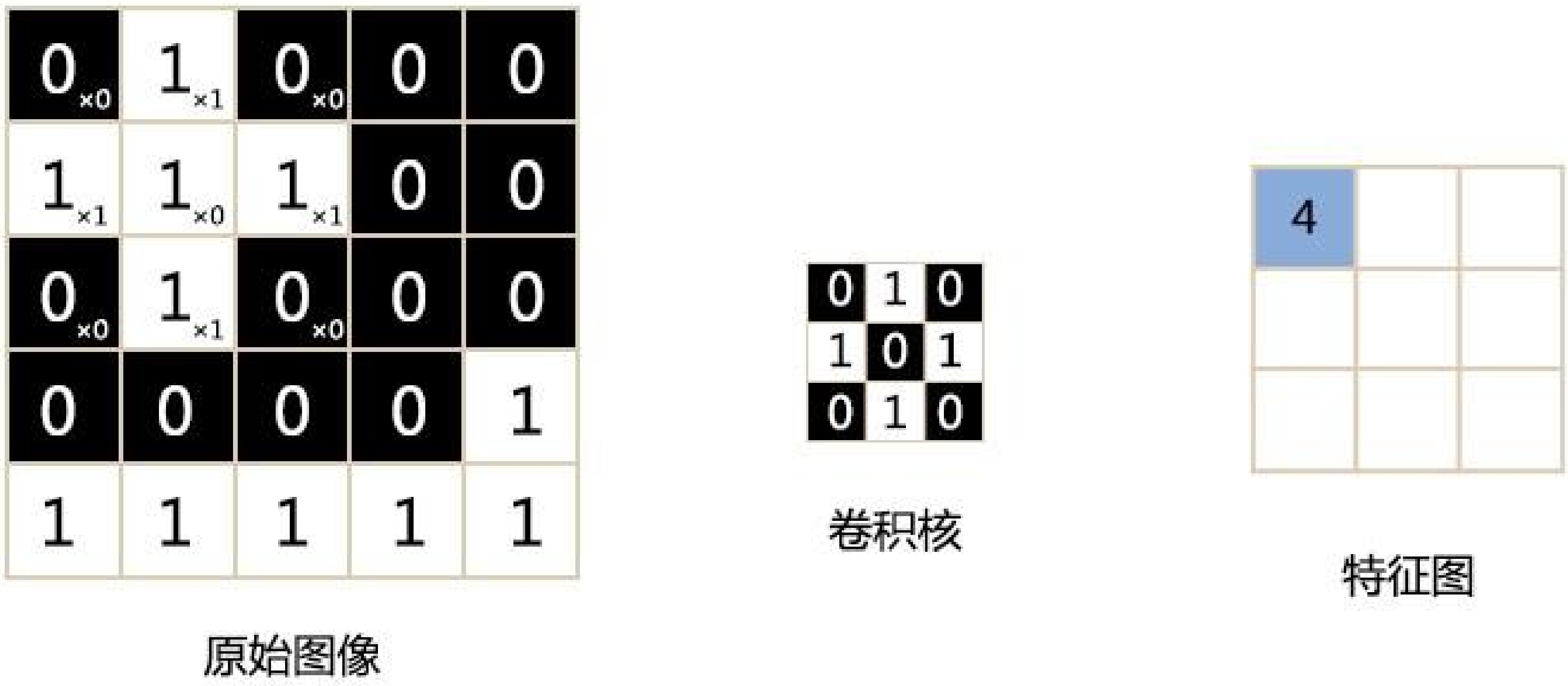

B
2
7
D
FL
P
5
P

卷积核与特征图



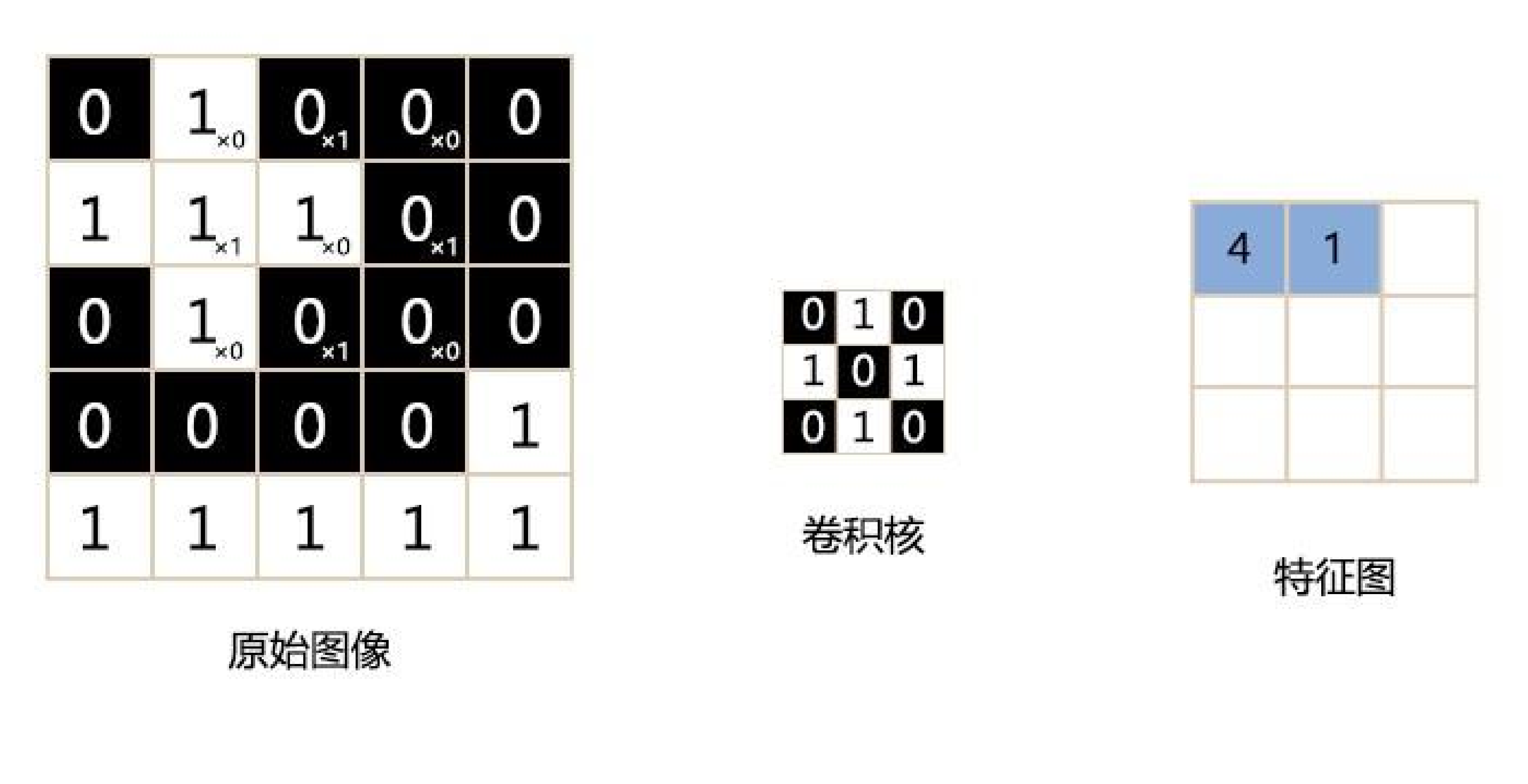
- 卷积就是在寻找与卷积核相似的区域，并将搜索结果映射到特征图上

数学上如何操作？

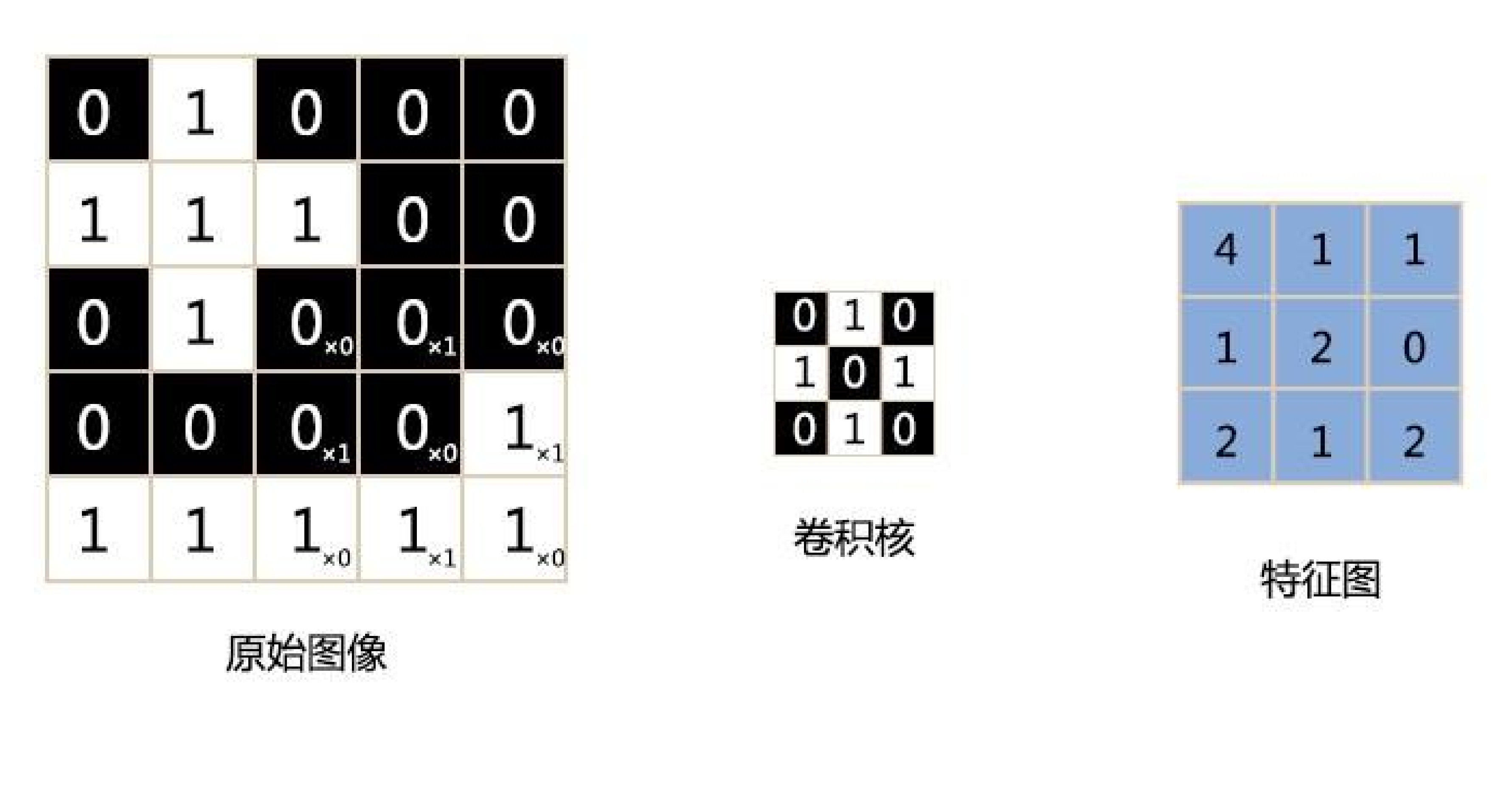


$$(f * g)(x,y) = \sum_u \sum_v f(u,v)g(x-u, y-v)$$

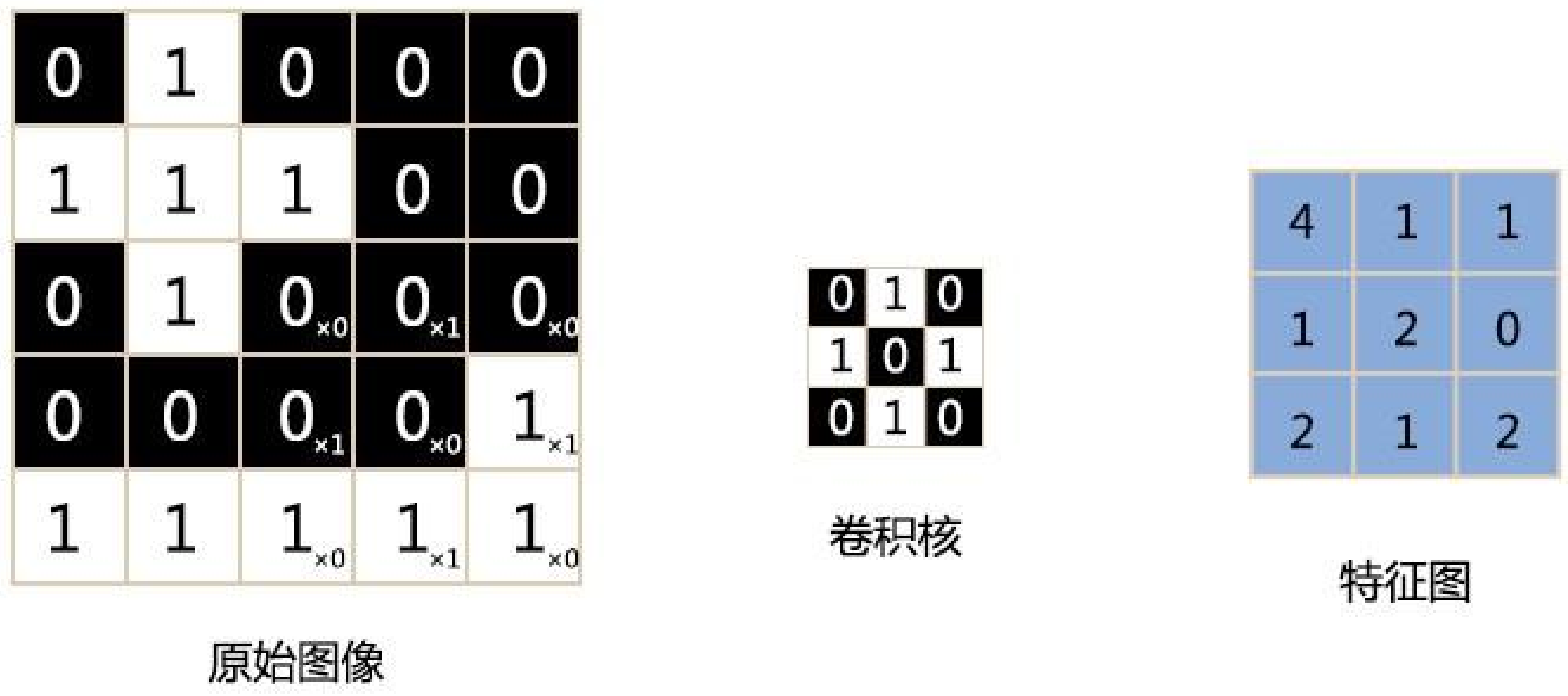
数学上如何操作？



数学上如何操作？

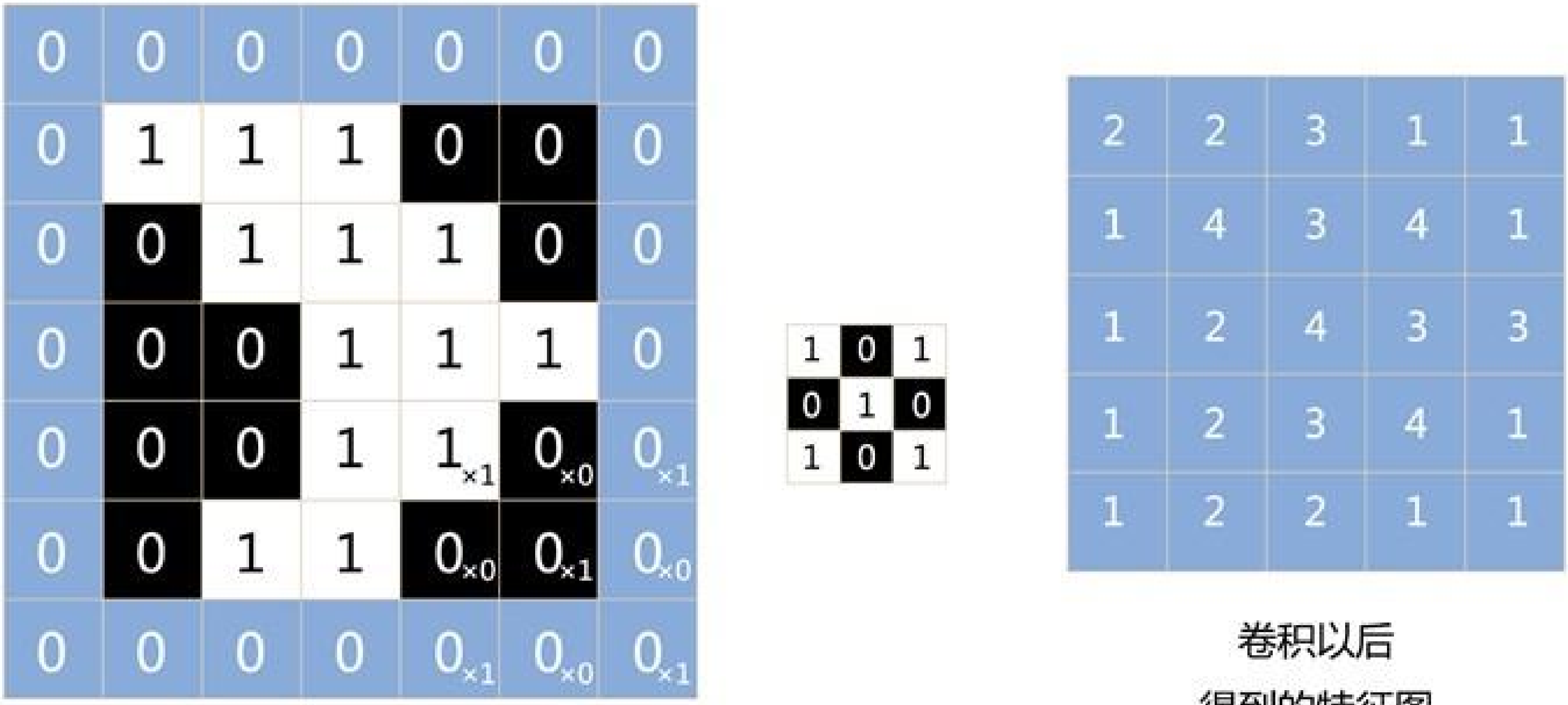


数学上如何操作？



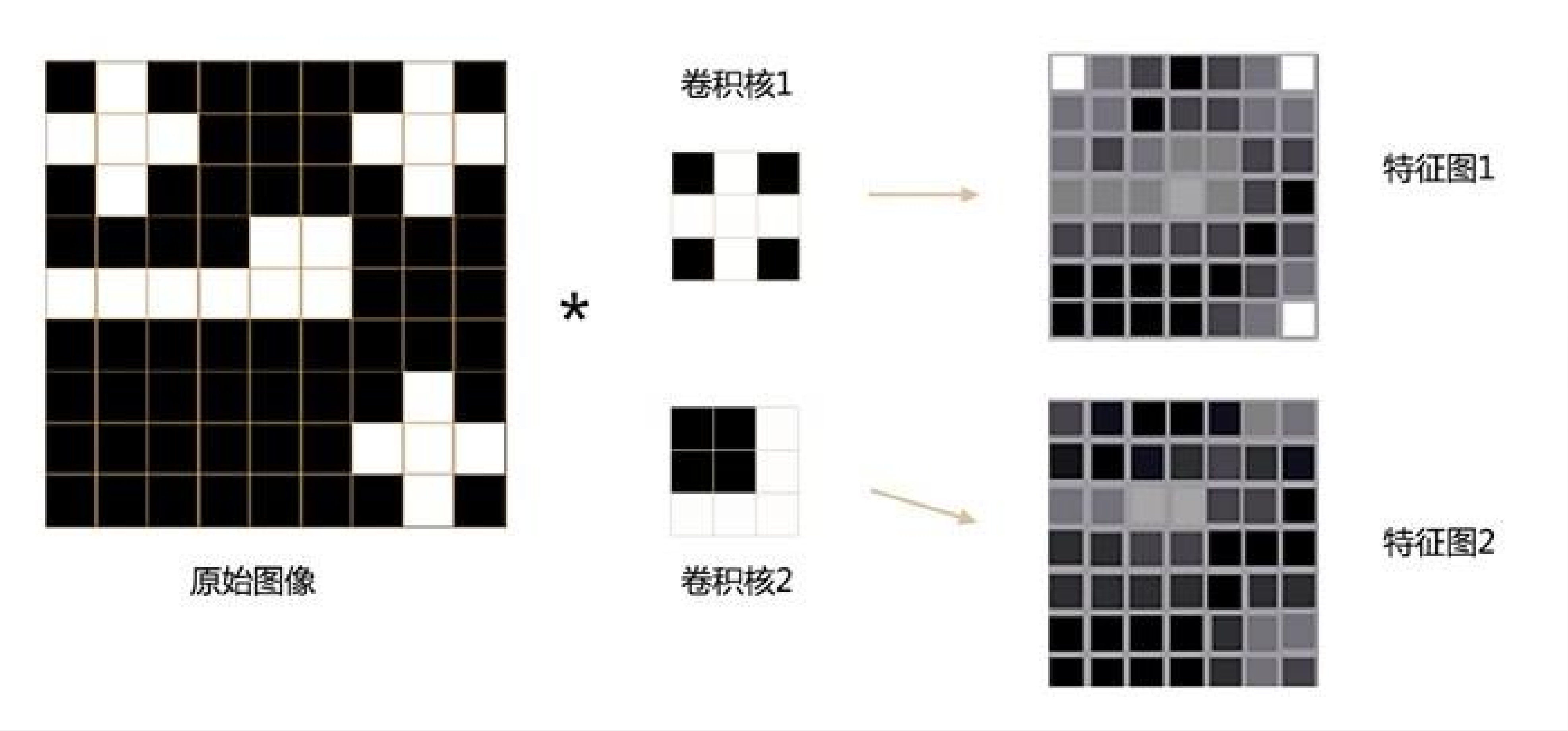
$$n \times n \rightarrow (n-w+1) \times (n-w+1)$$

补齐技术

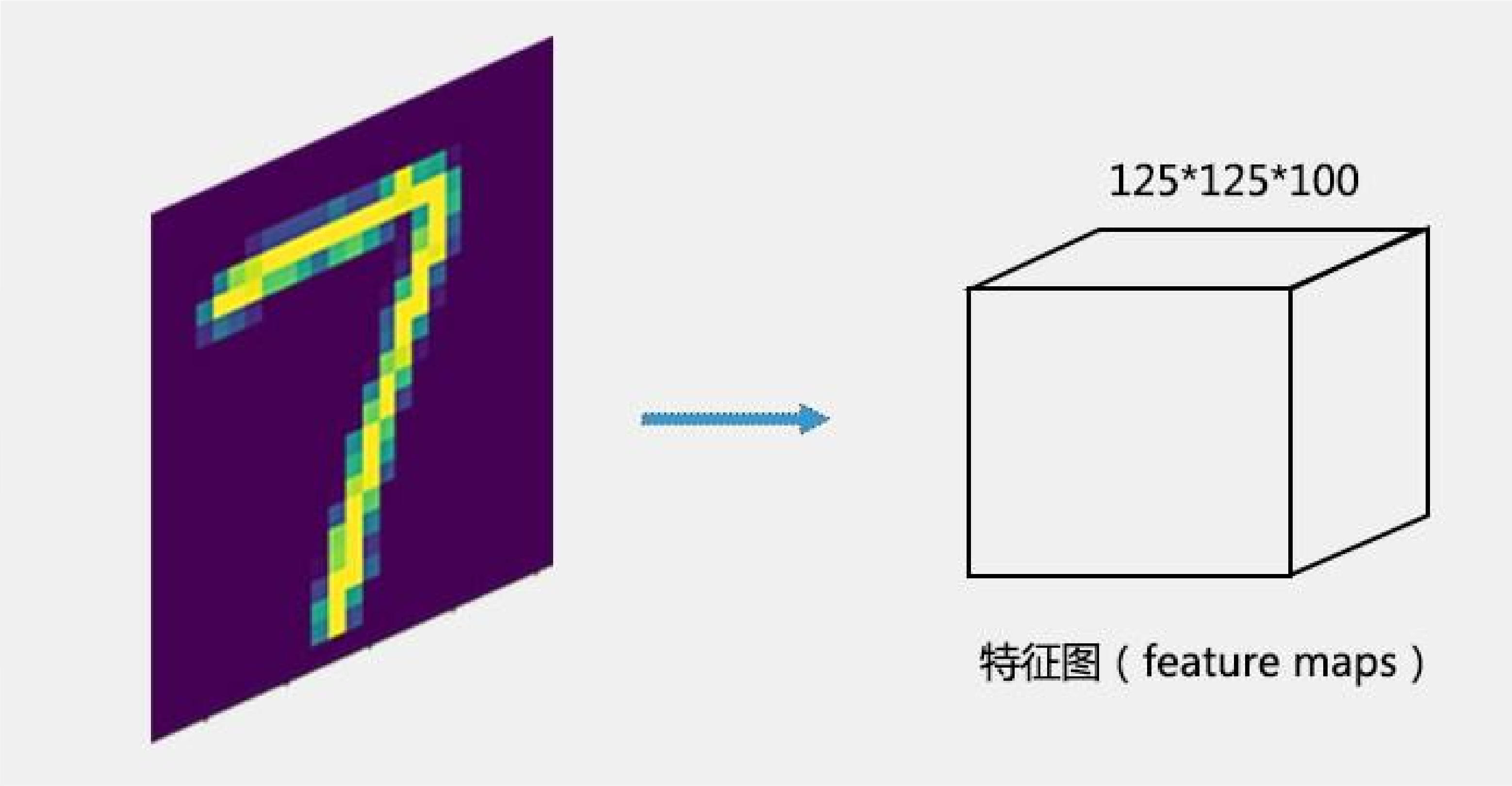


卷积以后
得到的特征图

多个卷积核

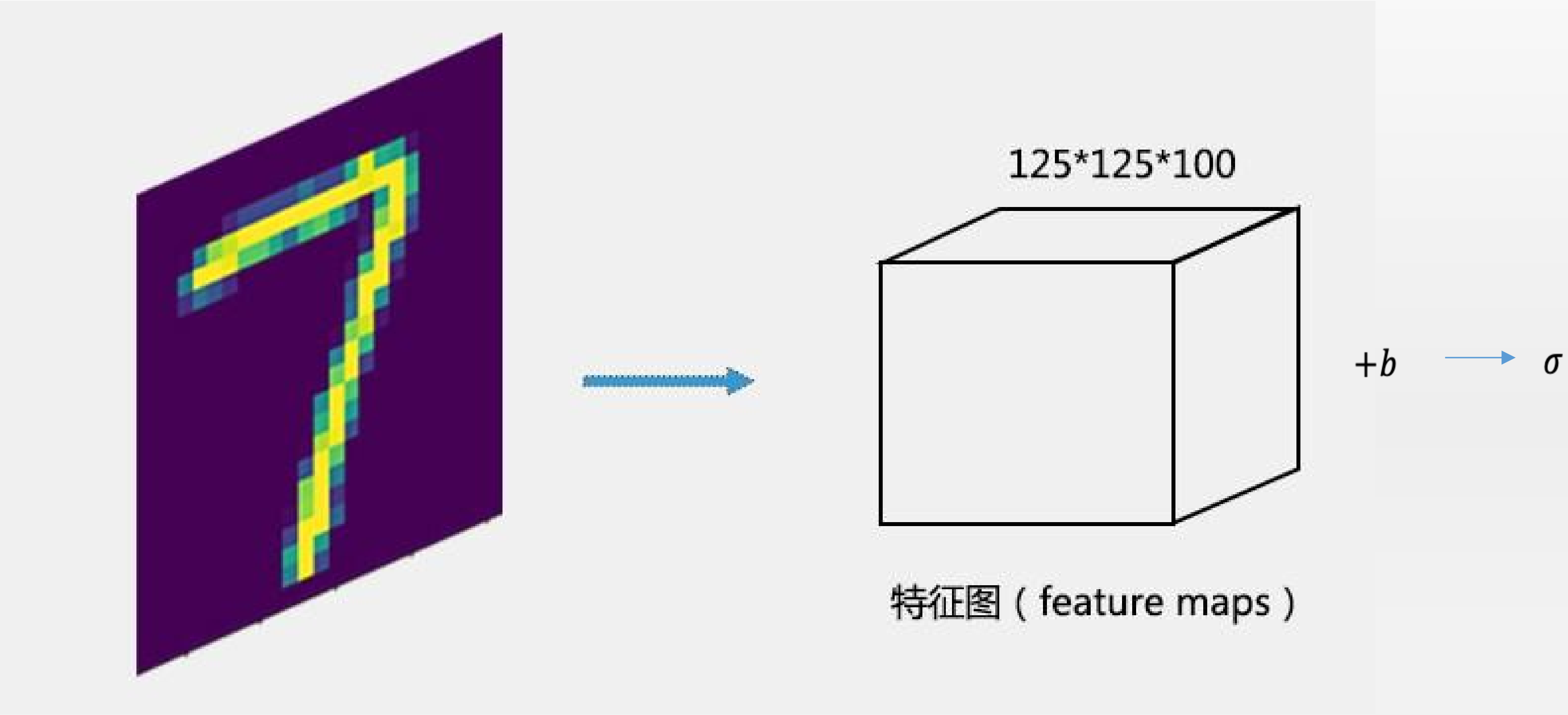


多个卷积核



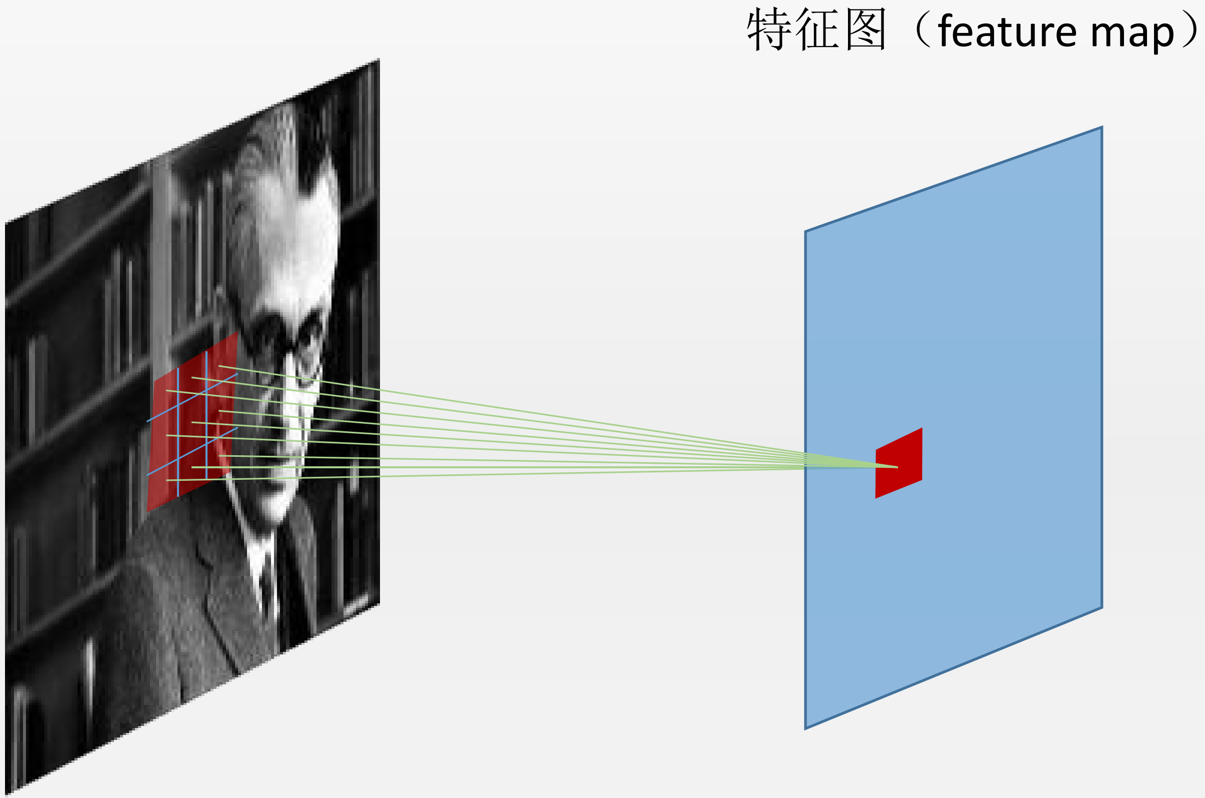
- 在表示的时候，我们可以将多个特征图拼在一起组成立方体

完整的卷积操作



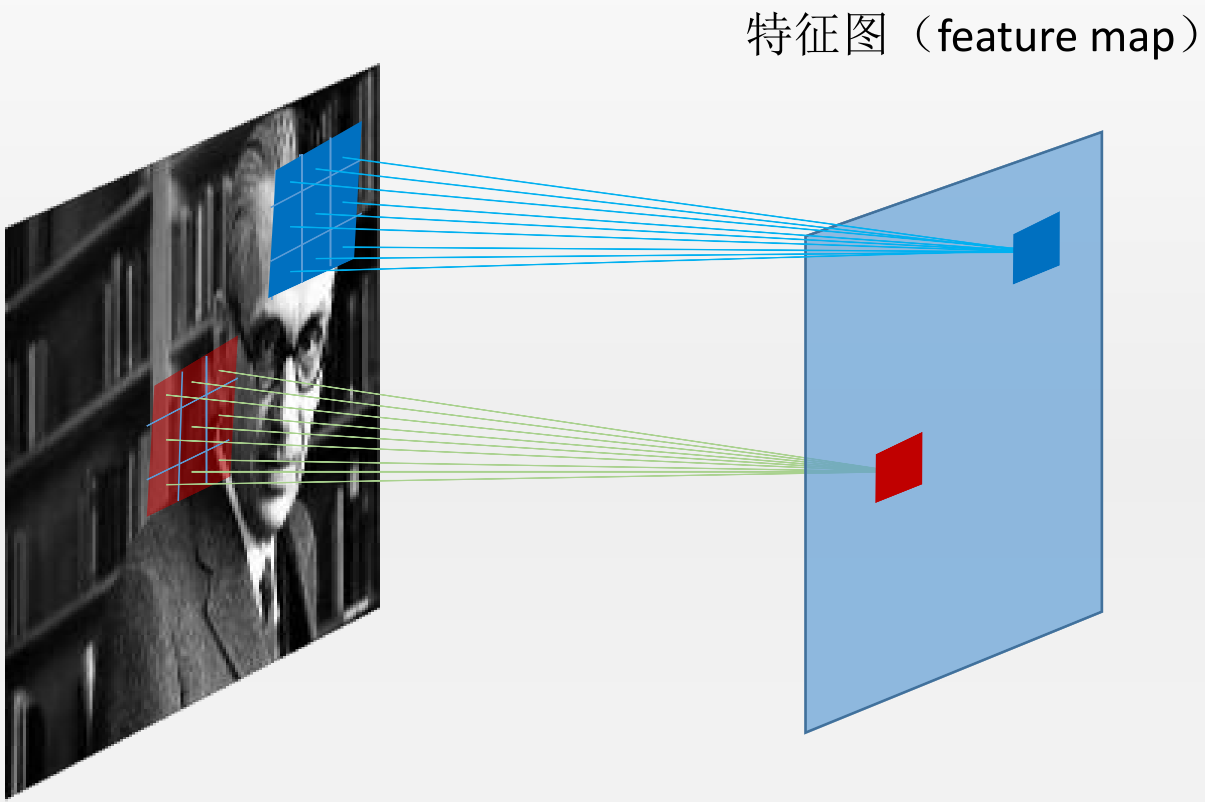
- 在表示的时候，我们可以将多个特征图拼在一起组成立方体

神经网络怎么实现？



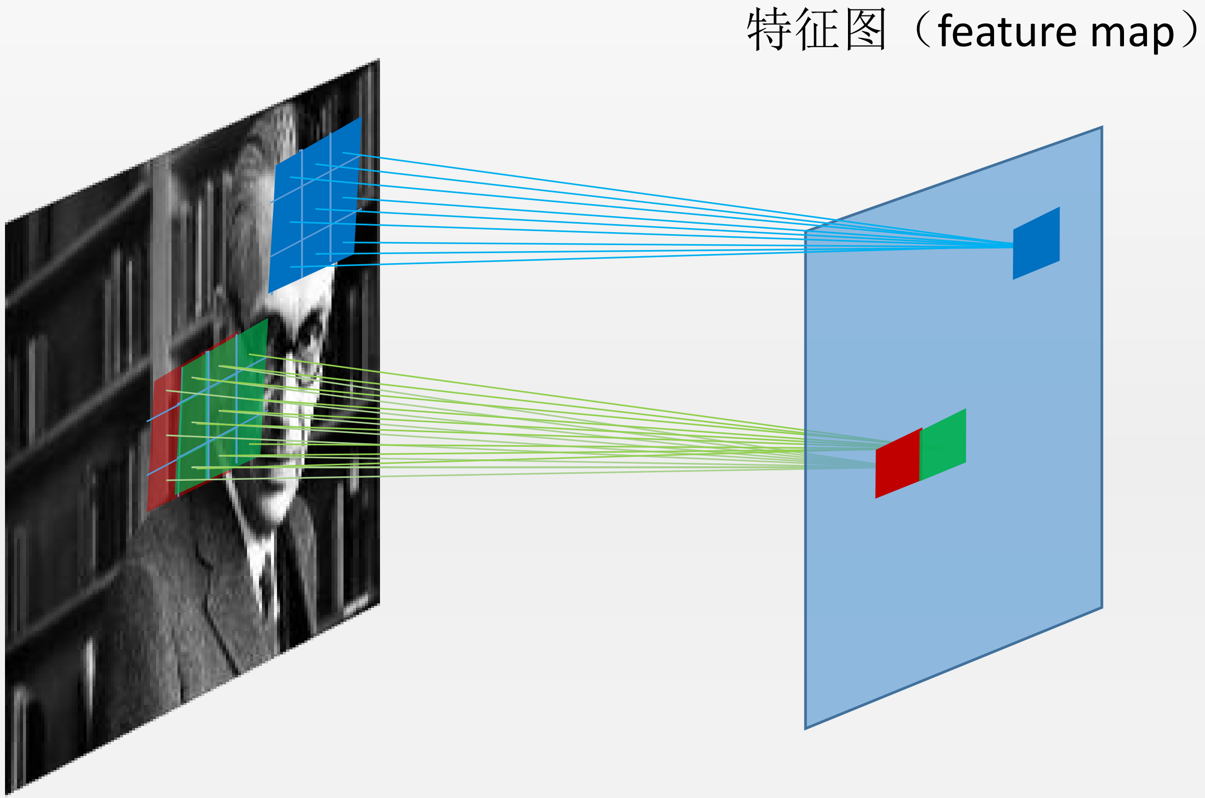
- 特征图上的每一个像素都和原图上的3*3大小的一个方形区域的像素相连（即9个链接）
- 每条连边都有一个数字，称为权重值

神经网络怎么实现？



- 蓝色像素对应了原图上的蓝色3*3大小的一个方形区域的像素，每个像素1个链接，一共有9条蓝色链接
- **权值共享：粉红色的每一条链接上面的权重值都与对应的蓝色链接权重值相等**

神经网络怎么实现？

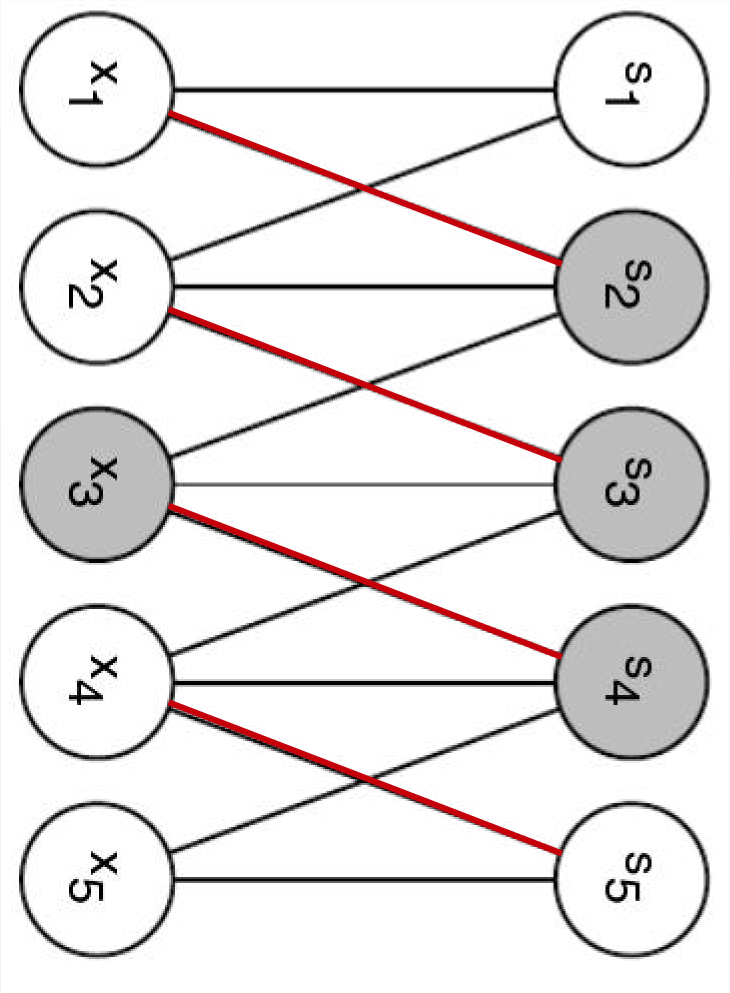


- 特征图上的绿色像素与红色相邻，它链接了原图绿色的像素，与红色区域相邻且有重叠

从一维的神经网络角度来看

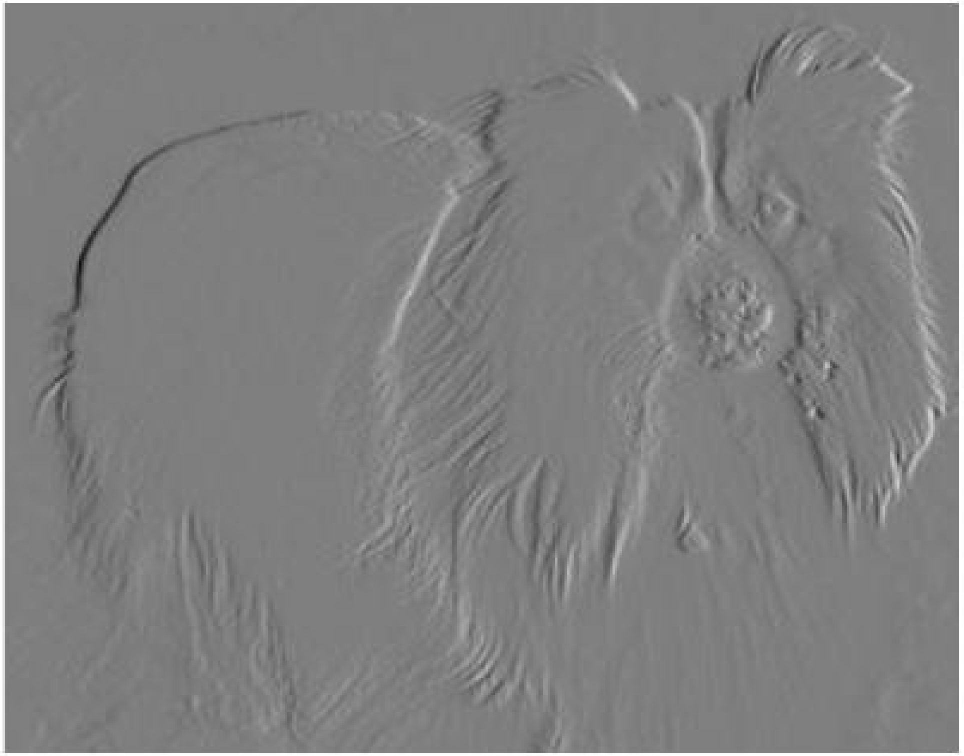
Shared weights w(x)

Input signal x



边缘检测

f(x,y)-f(x-1,y)

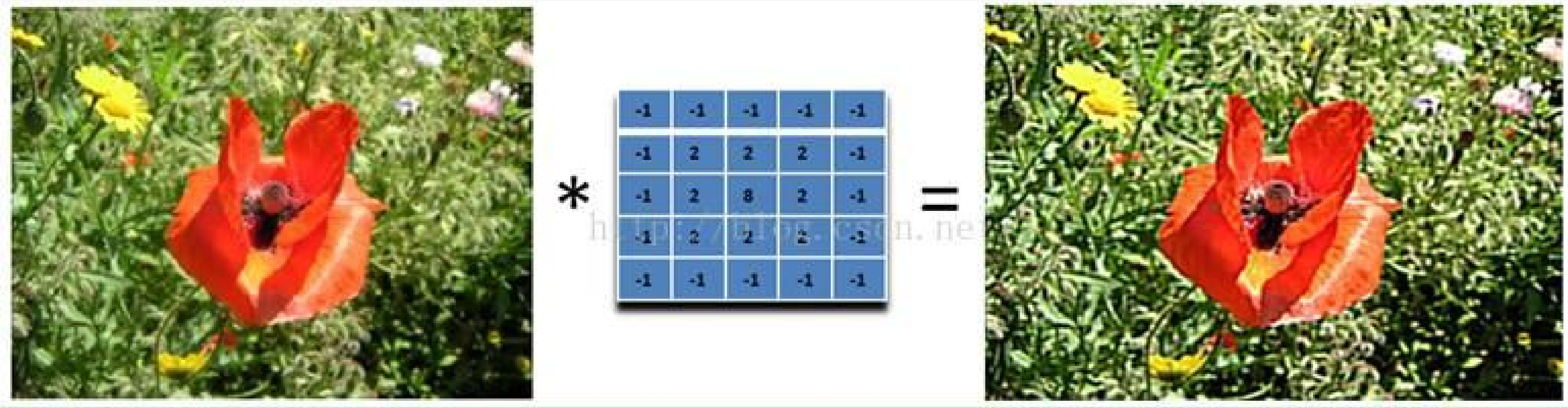


卷积核

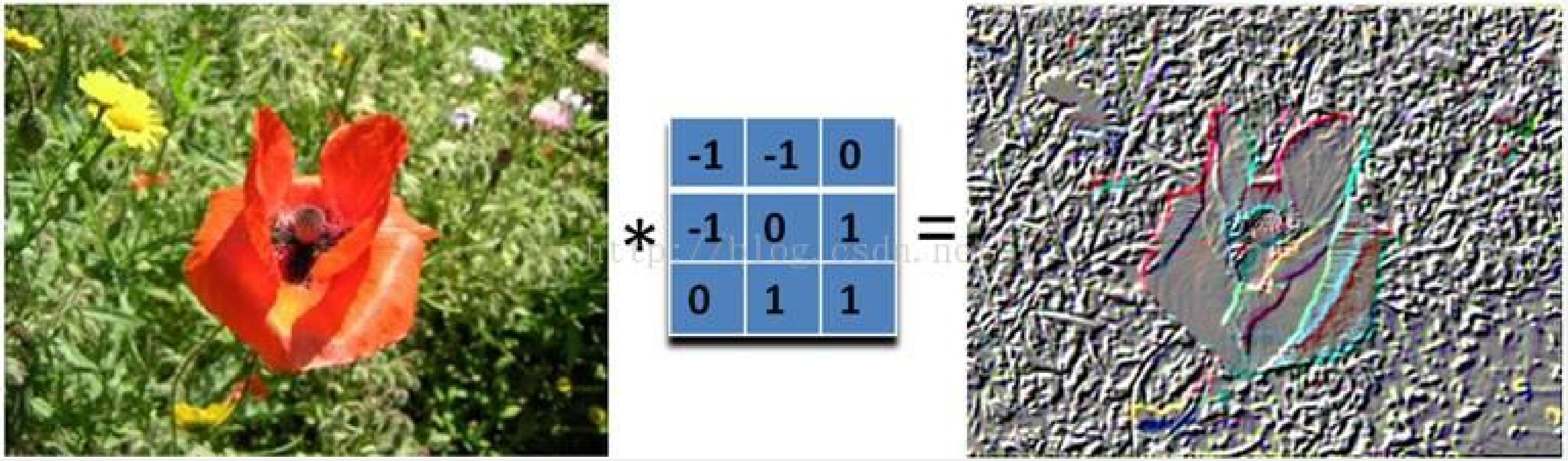
$$g(x,y) = \begin{cases} 1, & x = y \\ -1, & y = x + 1 \\ 0, & otherwise \end{cases}$$

1	-1	0
0	1	-1
0	0	1

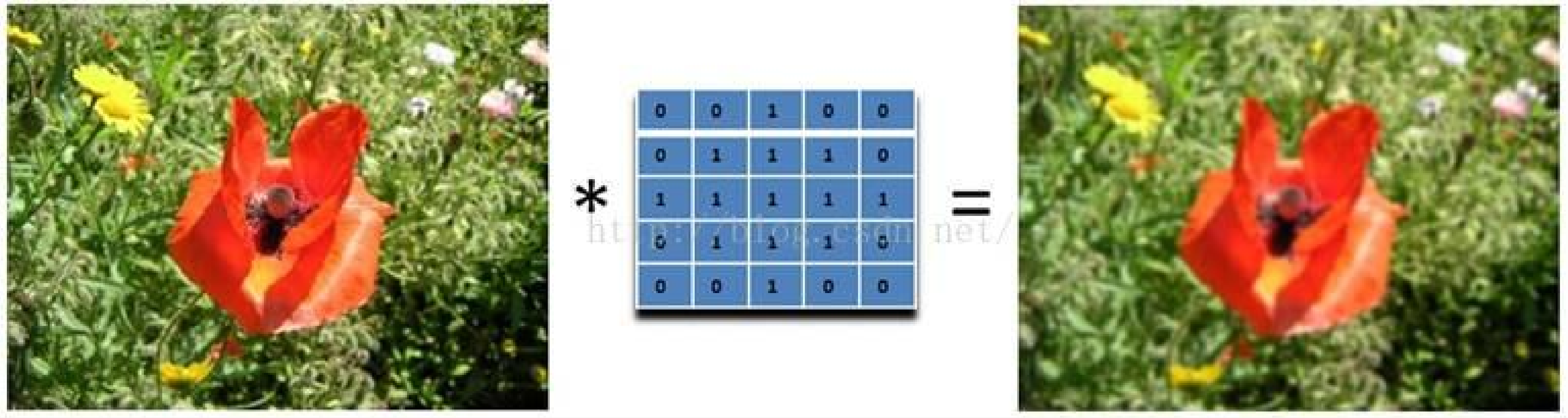
锐化



浮雕



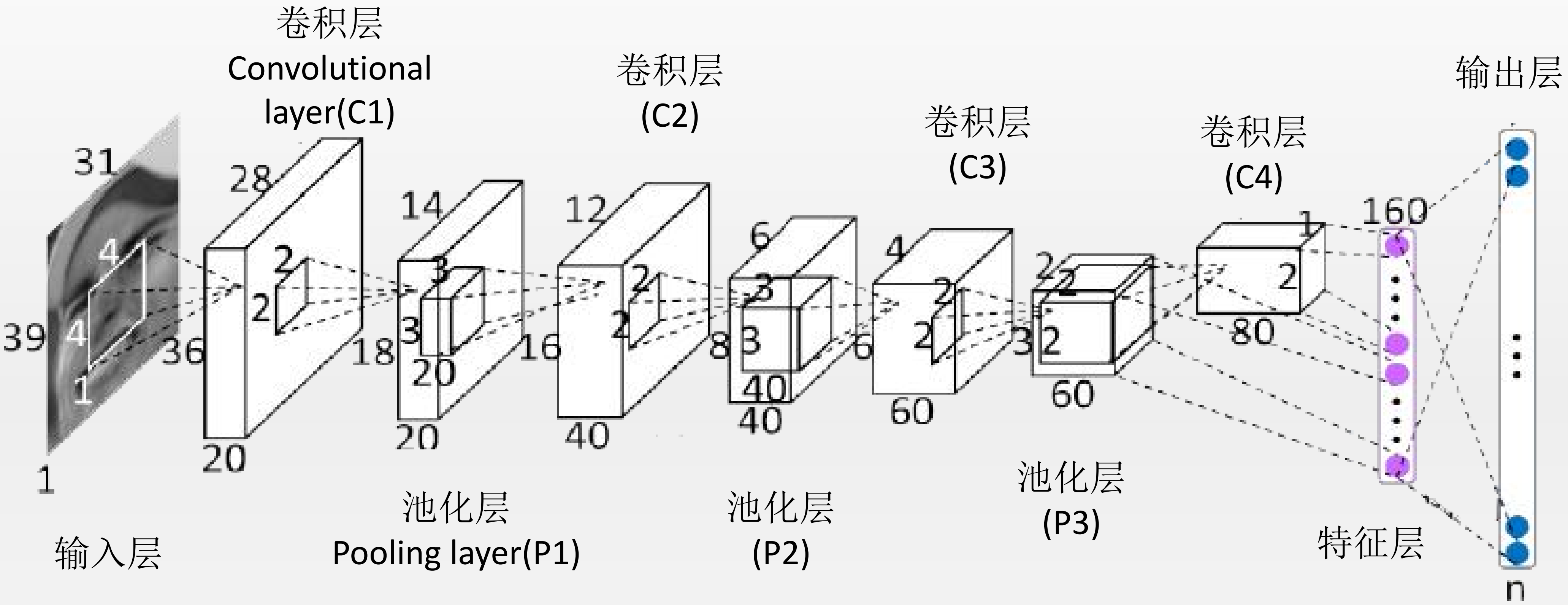
模糊



运动模糊

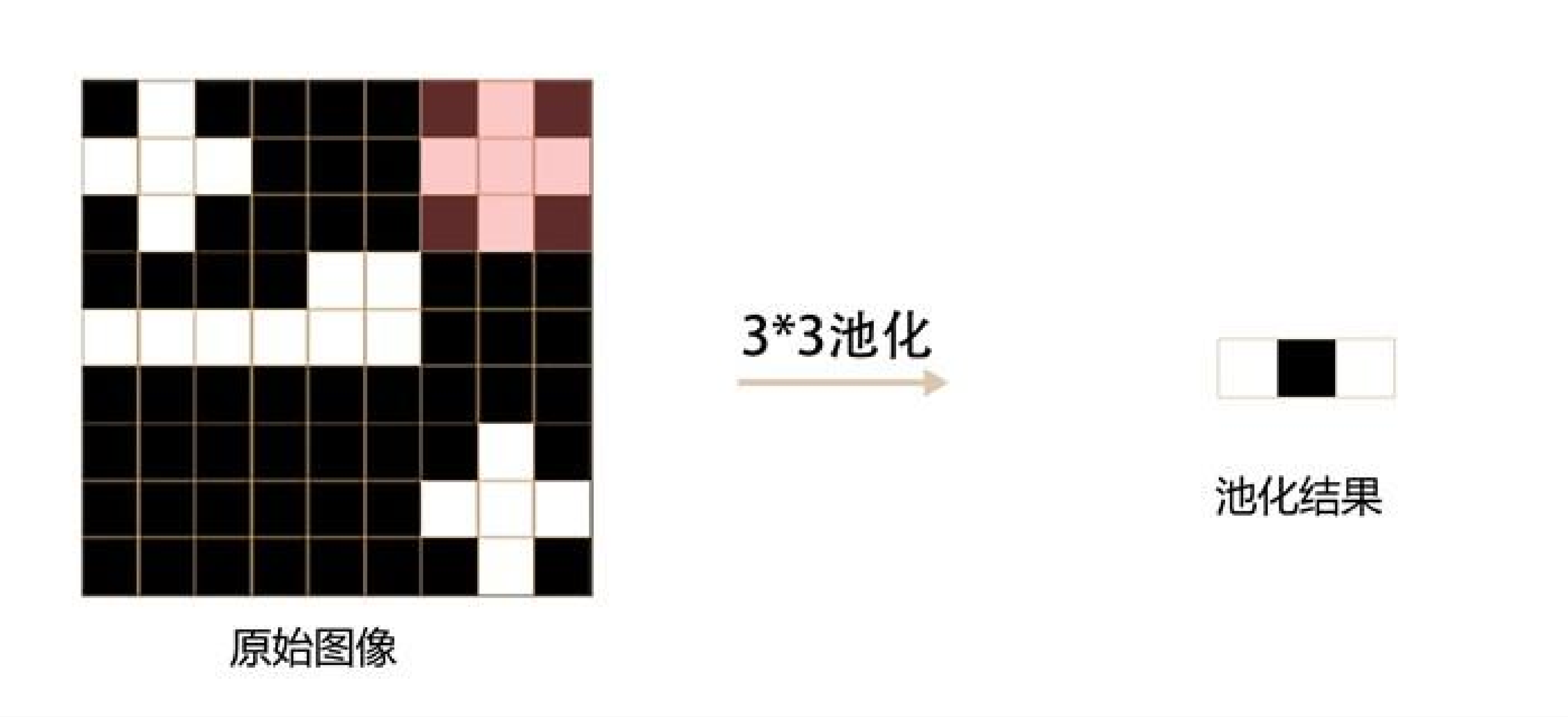
```
1, 0, 0, 0, 0, 0, 0, 0, 0
0, 1, 0, 0, 0, 0, 0, 0, 0
0, 0, 1, 0, 0, 0, 0, 0, 0
0, 0, 0, 1, 0, 0, 0, 0, 0
0, 0, 0, 0, 1, 0, 0, 0, 0
0, 0, 0, 0, 0, 1, 0, 0, 0
0, 0, 0, 0, 0, 0, 1, 0, 0
0, 0, 0, 0, 0, 0, 0, 1, 0
0, 0, 0, 0, 0, 0, 0, 0, 1
```

卷积神经网络



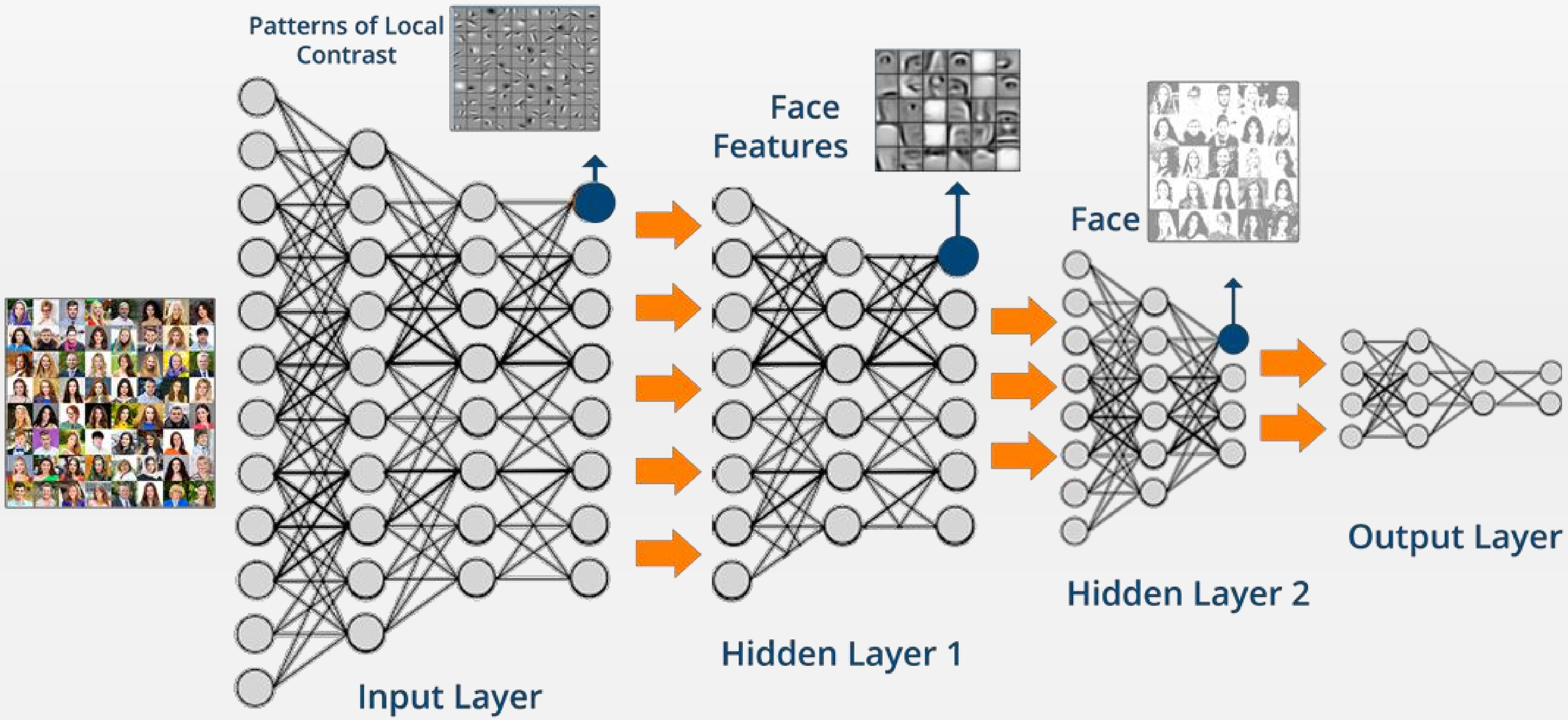
- 卷积神经网络包含多个层，每层的神经元都会排布成三维的立方体

池化操作



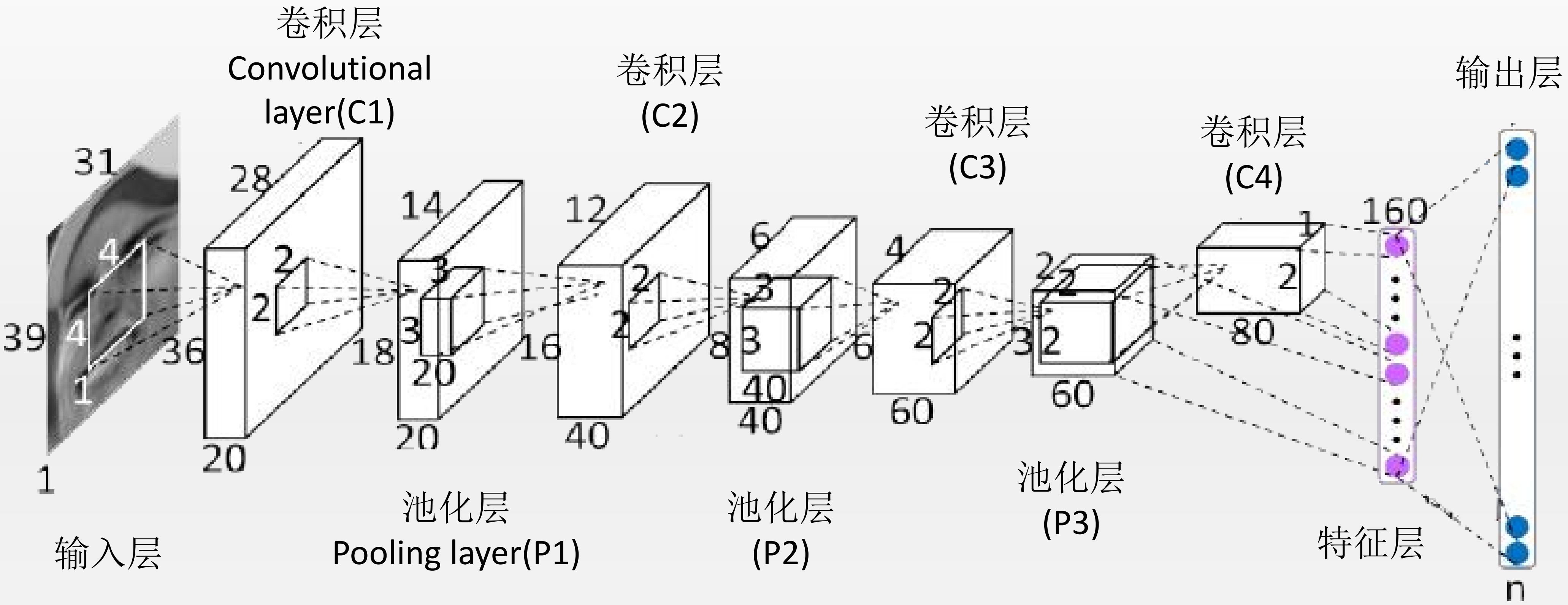
- 将原始图像划分成3*3的不重叠区域，每一个求最大值得到一个新图像

池化操作



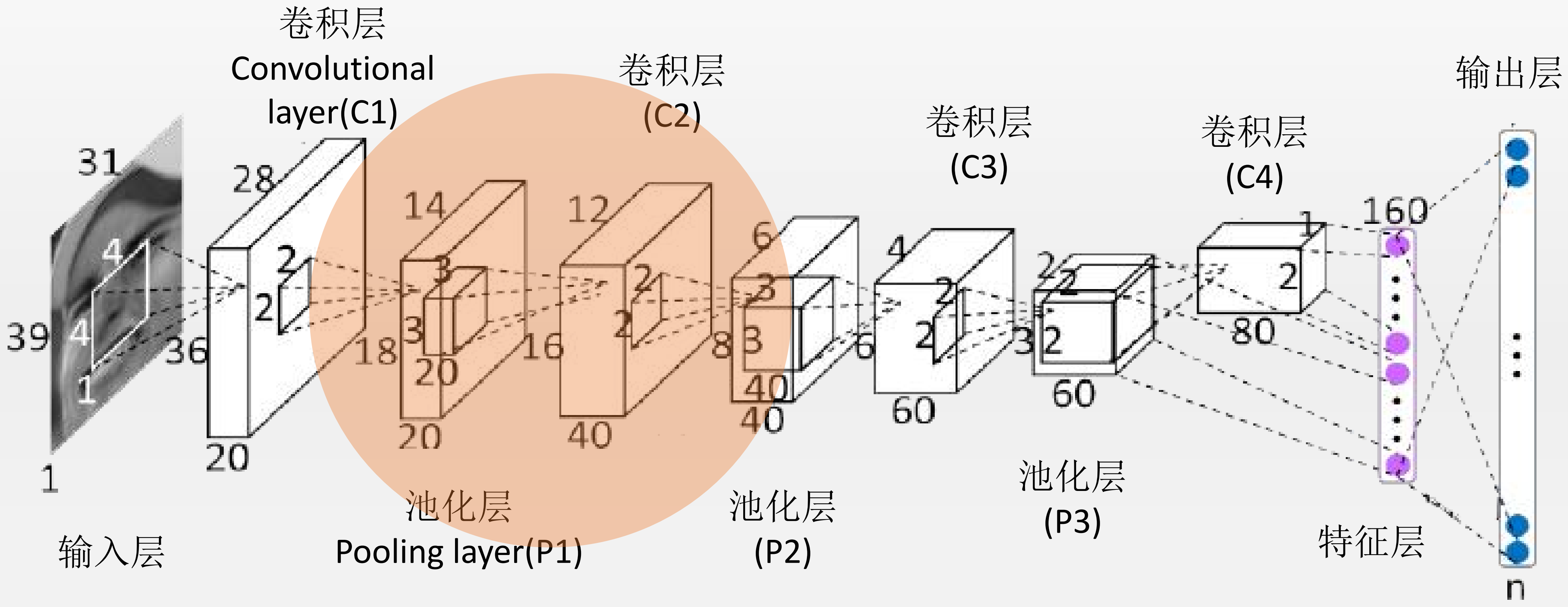
Patterns of Local Contrast
Face Features
Face
Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer

卷积神经网络

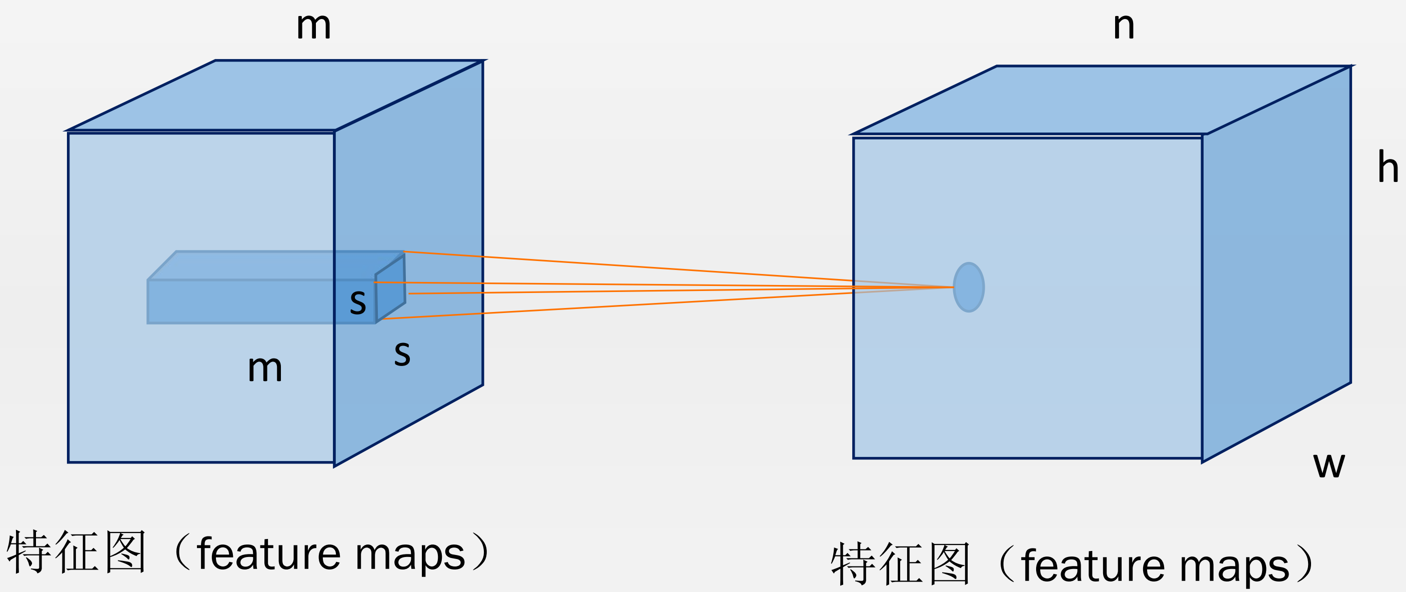


卷积层
Convolutional
layer(C1)
卷积层
(C2)
卷积层
(C3)
卷积层
(C4)
输出层
31
28
14
12
6
4
160
39
36
18
16
8
6
3
80
60
60
40
40
40
20
20
20
1
n
输入层
池化层
Pooling layer(P1)
池化层
(P2)
池化层
(P3)
特征层

整体的卷积神经网络架构



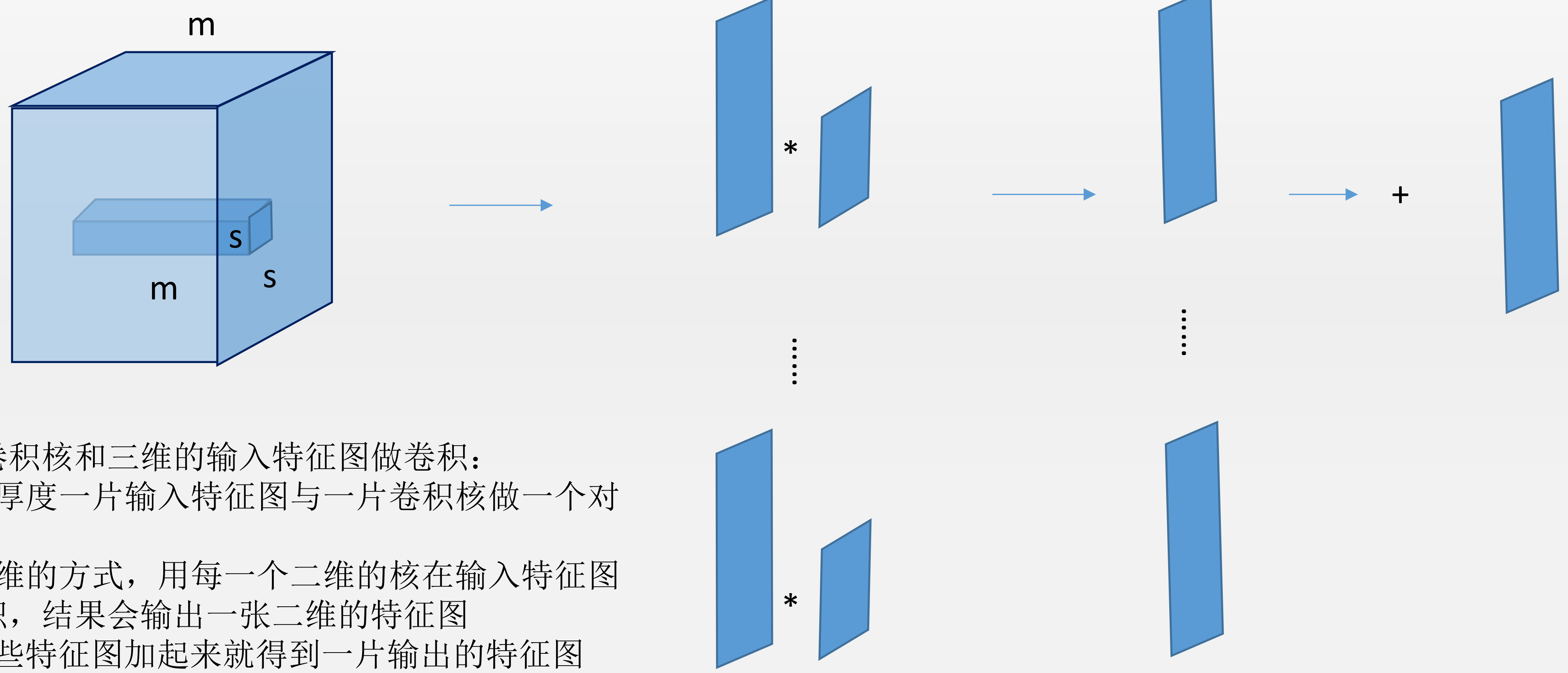
立体卷积核



对于多个特征图的卷积，要注意卷积核的维度。
共有n个卷积核，一个卷积核有m*s*s个权重

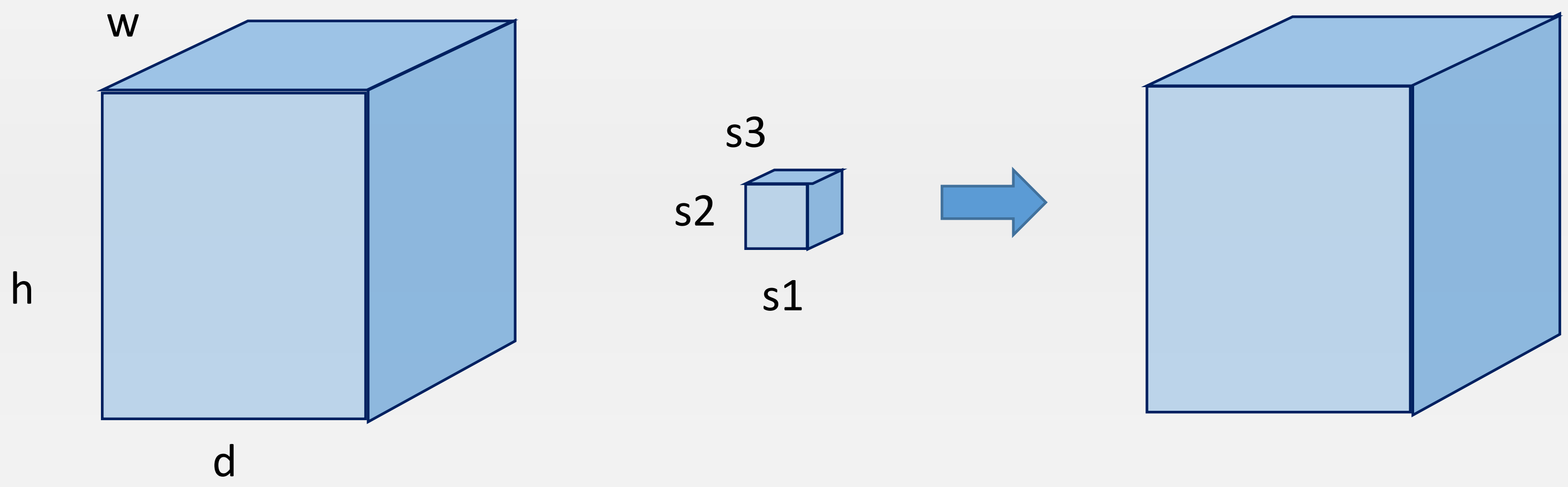
立体卷积如何运算？

输入特征图（feature maps）

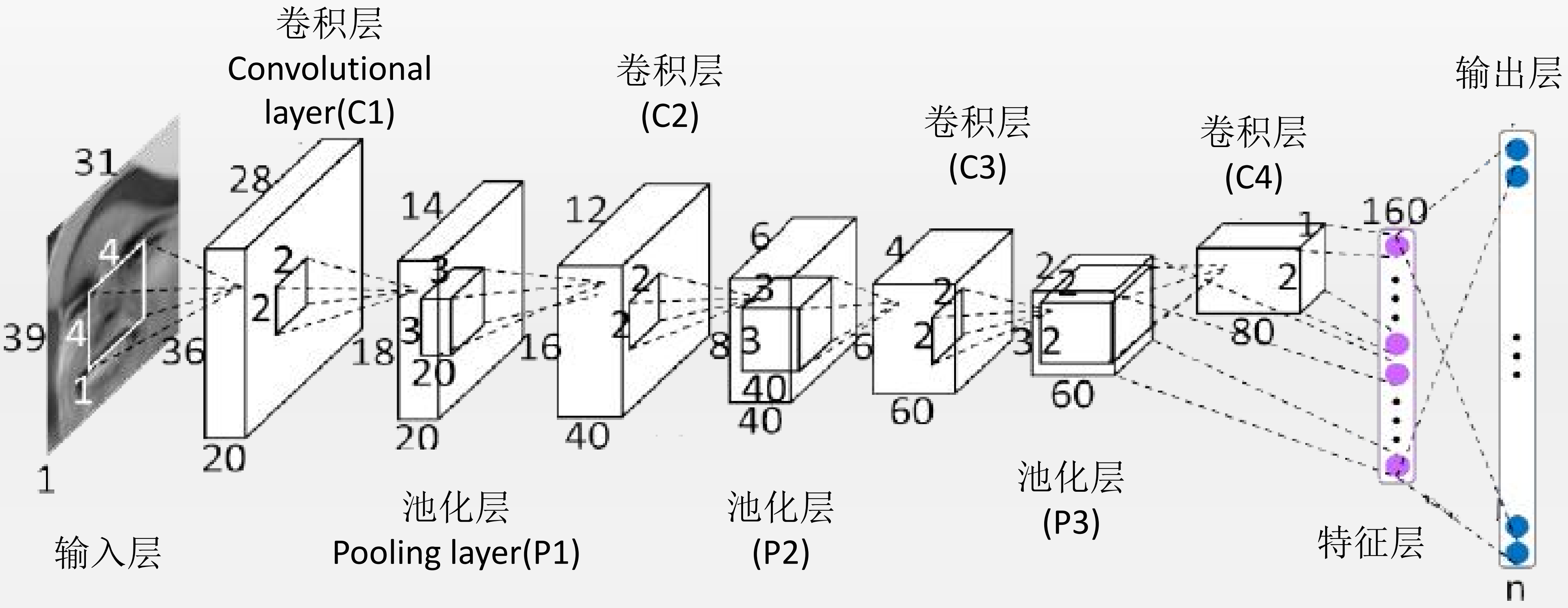


三维的卷积核和三维的输入特征图做卷积：
1、按照厚度一片输入特征图与一片卷积核做一个对应；
2、用二维的方式，用每一个二维的核在输入特征图上做卷积，结果会输出一张二维的特征图
3、把这些特征图加起来就得到一片输出的特征图

真正的3D卷积呢？

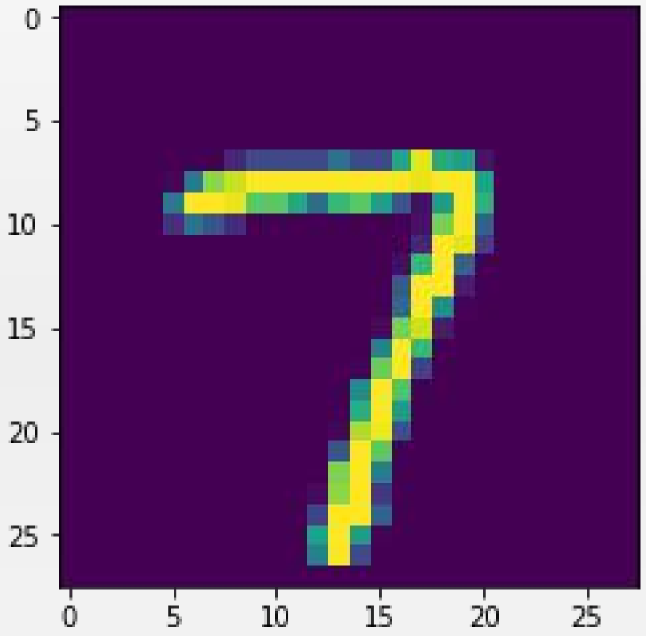


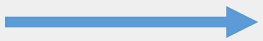
整体的卷积神经网络架构



要学习调整的就是网络中的所有卷积核

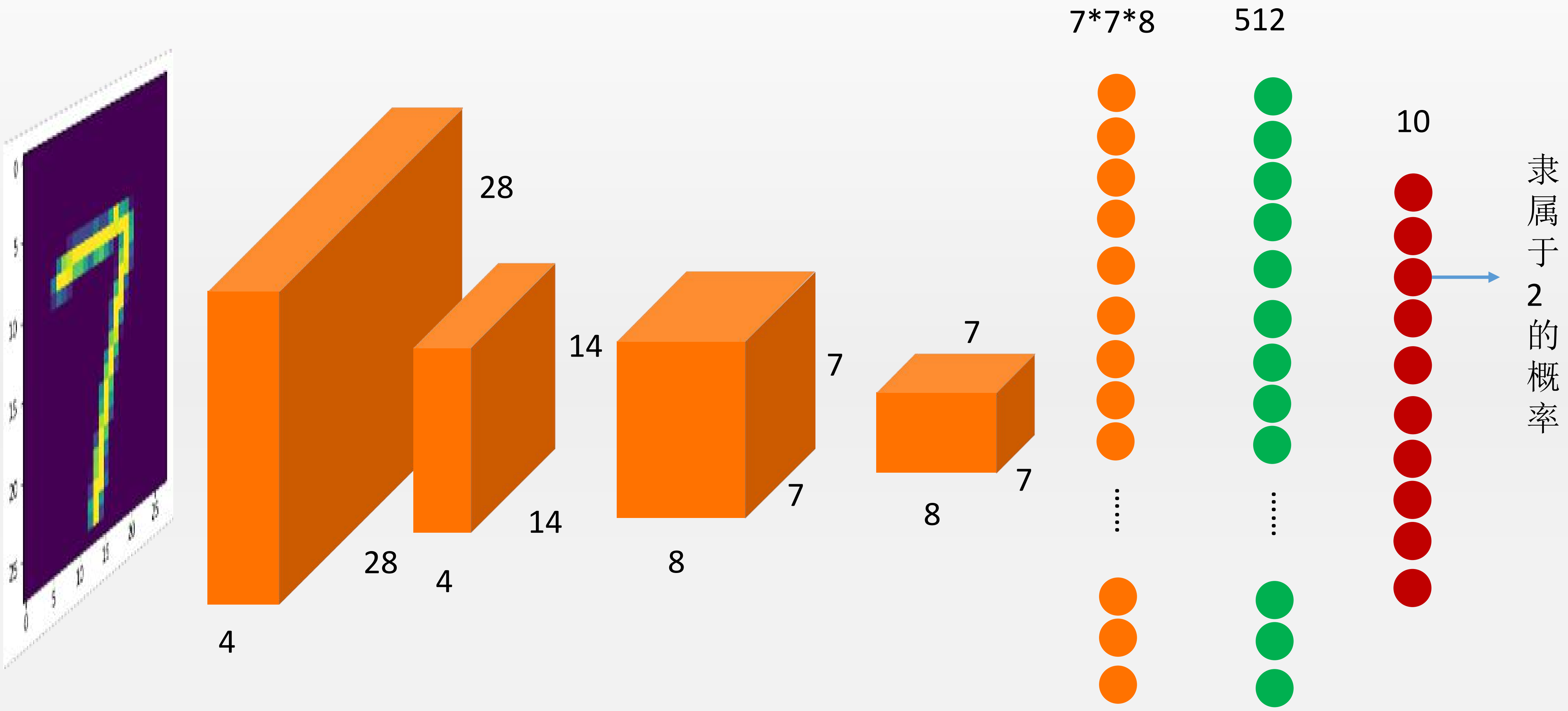
一个具体的例子：MINST



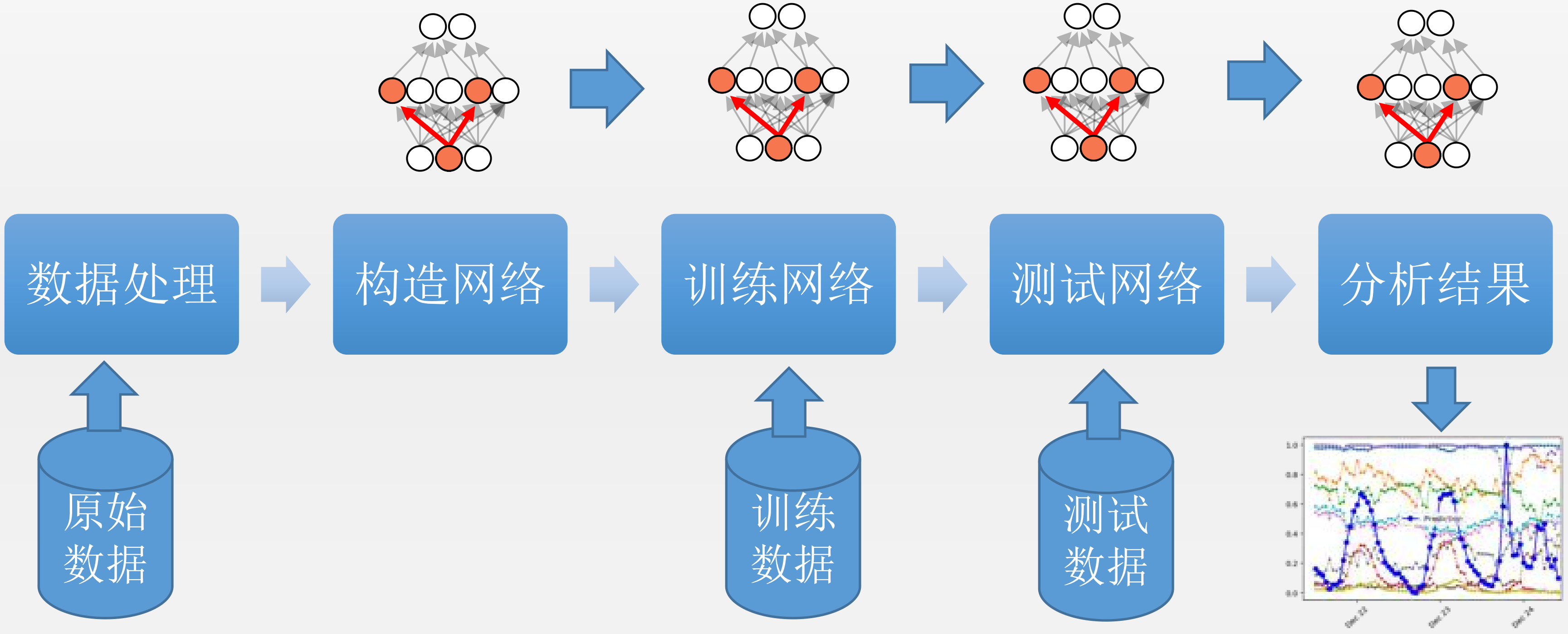


7

网络架构



数据准备



torchvision包

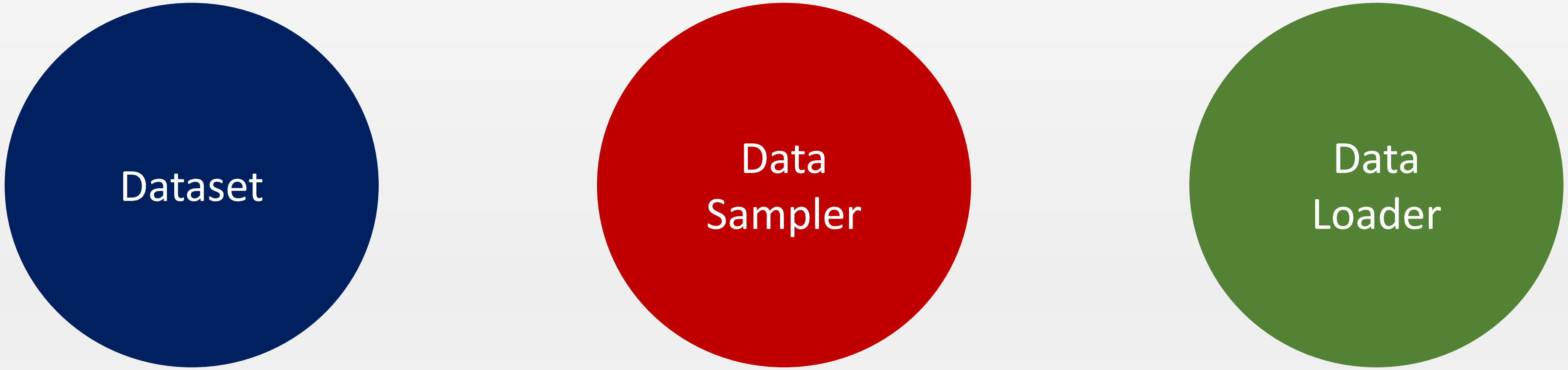
dataset

models

transforms

utils

数据集对象torch.util.data



数据加载器的建立

In [1]:

```
import torchvision.dataset as dsets
Import torchvision.transforms as transforms
```

In [2]:

```
train_dataset = dsets.MNIST(root='./data', #文件存放路径
                            train=True,  #提取训练集
                            transform=transforms.ToTensor(), #将图像转化为Tensor
                            download=True)
test_dataset = dsets.MNIST(root='./data',
                           train=False,
                           transform=transforms.ToTensor())
# 数据集的加载
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

数据集的使用

可以像访问数组一样，对数据集中的元素进行下标索引

In [1]:

```
train_dataset[0]
```

out[1]:

```
……
……
……
[torch.FloatTensor of size 1x28x28], 5)
```

返回一个二元组，前为特征，后为标签

In [2]:

```
len(train_dataset)
```

out[2]:

```
60000
```

返回train_dataset中数据的数量

对数据集的访问

可以像循环列表一样，从加载器中不断地抽取数据

In [1]:

```
for (x,y) in train_loader:
    print(x)
    print(y)
```

out[1]:

```
……
……
……
[torch.FloatTensor of size 1x28x28]

5
……
```

顺序打印train_loader中的元素

In [2]:

```
len(train_loader)
```

out[2]:

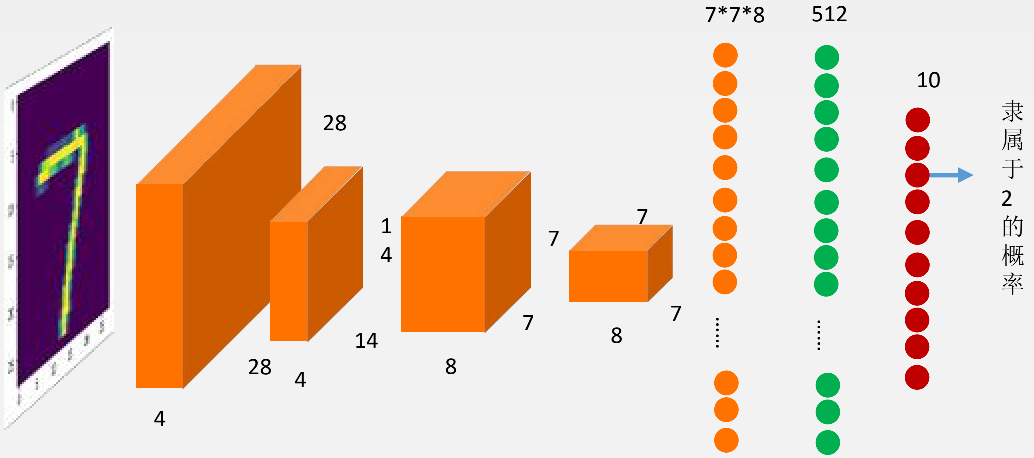
```
60000
```

返回train_dataset中数据的数量

卷积神经网络的实现

In [3]:

```
depth = [4, 8]
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 4, 5, padding = 2)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(depth[0], depth[1], 5, padding = 2)
        self.fc1 = nn.Linear(image_size // 4 * image_size // 4 * depth[1] , 512)
        self.fc2 = nn.Linear(512, num_classes)
```

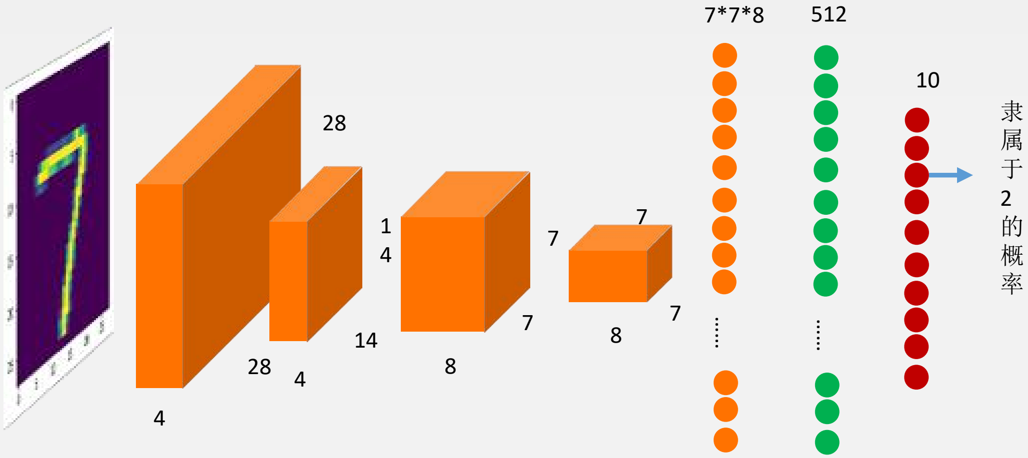


卷积神经网络的实现

In [3]:

```
depth = [4, 8]
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 =
        self.pool =
        self.conv2 = nn.Conv2d(depth[0], depth[1], 5, padding = 2)
        self.fc1 = nn.Linear(image_size // 4 * image_size // 4 * depth[1] , 512)
        self.fc2 = nn.Linear(512, num_classes)
```

nn.Conv2d(in_channel,out_channel,window_size,padding)

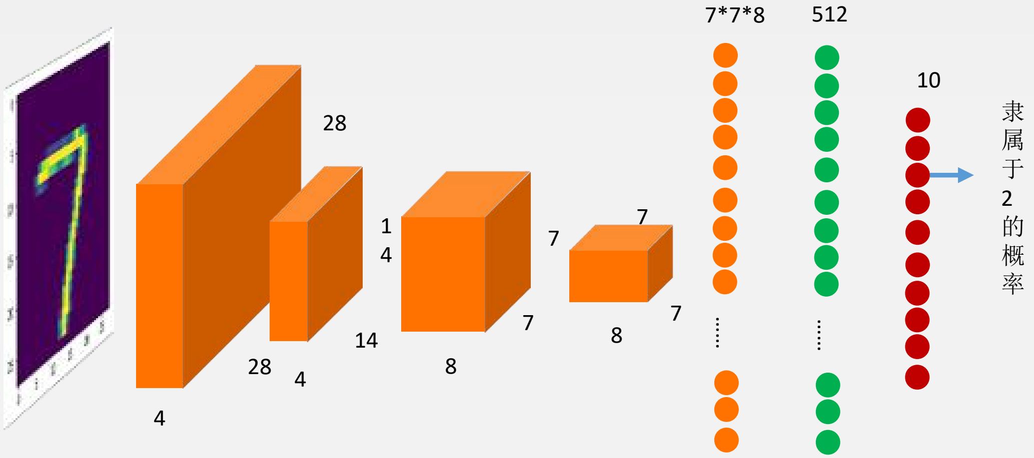


卷积神经网络的实现

In [3]:

```
depth = [4, 8]
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 4, 5, padding = 2)
        self.pool = 
        self.conv2 = nn.Conv2d(depth[0], depth[1], 5, padding = 2)
        self.fc1 = nn.Linear(image_size // 4 * image_size // 4 * depth[1] , 512)
        self.fc2 = nn.Linear(512, num_classes)
```

nn.MaxPool2d(width,height)



卷积神经网络的实现

In [3]:

```
depth = [4, 8]
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
```

```
class ConvNet(nn.Module):
```

nn.Module的一个子类

nn.Module中包含了绝大部分关于神经网络的通用计算，如初始化、前传等

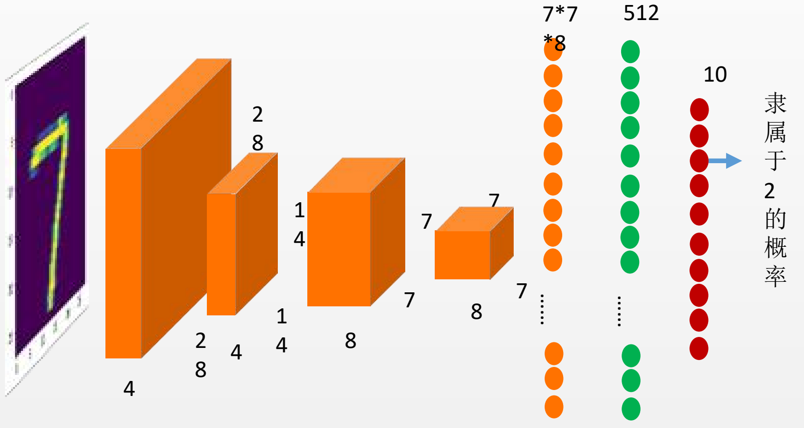
用户可以重写nn.Module中的部分函数，以实现定制化，如__init__(), 创建对象的时候调用，即：

```
net = ConvNet()
```

卷积神经网络的实现

In [3]:

```
depth = [4, 8]
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 4, 5, padding = 2)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(depth[0], depth[1], 5, padding = 2)
        self.fc1 = nn.Linear(image_size // 4 * image_size // 4 * depth[1] , 512)
        self.fc2 = nn.Linear(512, num_classes)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, image_size // 4 * image_size // 4 * depth[1])
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        x = F.log_softmax(x)
        return x
```

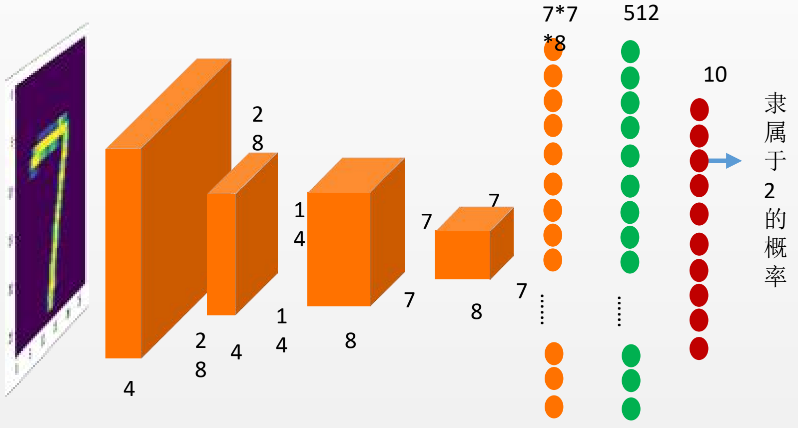


Forward在执行网络的前向传播的时候调用

卷积神经网络的实现

In [3]:

```
depth = [4, 8]
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 4, 5, padding = 2)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(depth[0], depth[1], 5, padding = 2)
        self.fc1 = nn.Linear(image_size // 4 * image_size // 4 * depth[1] , 512)
        self.fc2 = nn.Linear(512, num_classes)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, image_size // 4 * image_size // 4 * depth[1])
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        x = F.log_softmax(x)
        return x
```



在这里把张量展开一维的向量

卷积神经网络的实现

view(*args)，将张量转换为想要的形状

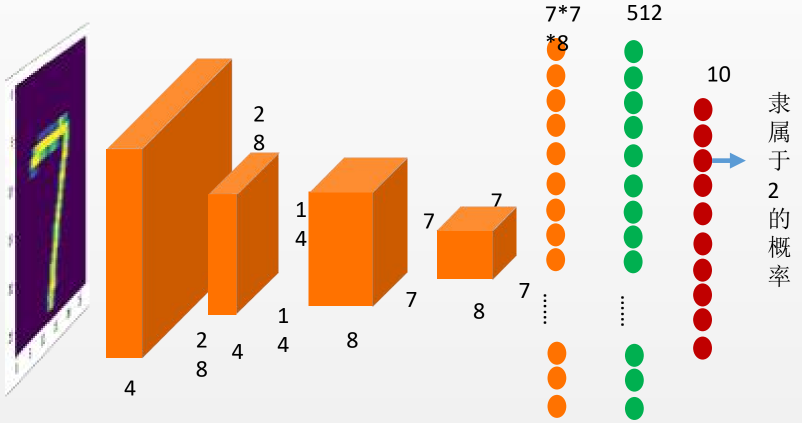
```
>>> x = torch.randn(4, 4)
 >>>x.size()
 torch.Size([4, 4])
>>> y = x.view(16)
>>> y.size()
torch.Size([16])
>>> z = x.view(-1, 8)
# 8对应的维度被锁死，其它维度尺寸会自动推断
>>> z.size()
torch.Size([2, 8])
```


在这里把张量展开一维的向量

卷积神经网络的实现

In [3]:

```
depth = [4, 8]
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 4, 5, padding = 2)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(depth[0], depth[1], 5, padding = 2)
        self.fc1 = nn.Linear(image_size // 4 * image_size // 4 * depth[1] , 512)
        self.fc2 = nn.Linear(512, num_classes)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, image_size // 4 * image_size // 4 * depth[1])
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        x = F.log_softmax(x)
        return x
```

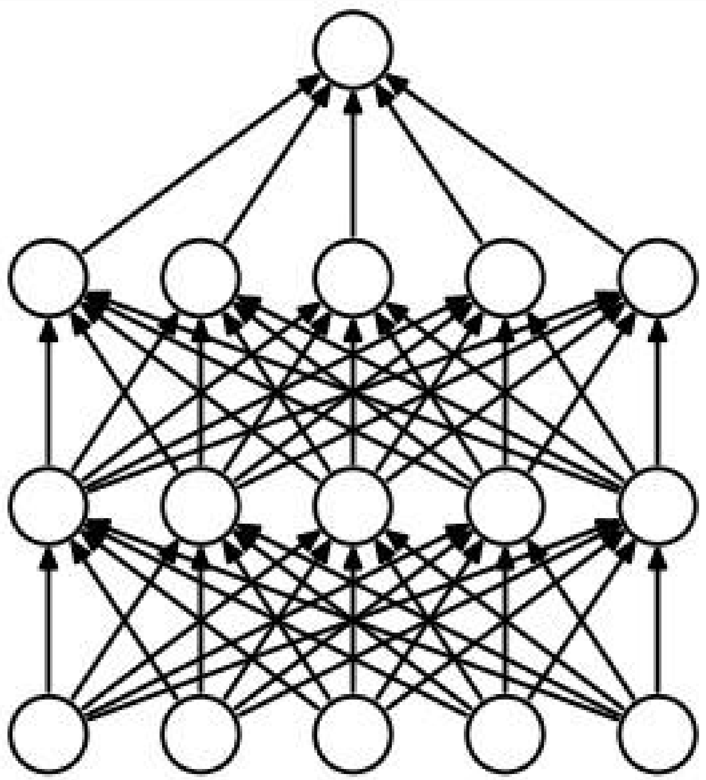


一种防止过拟合的技术

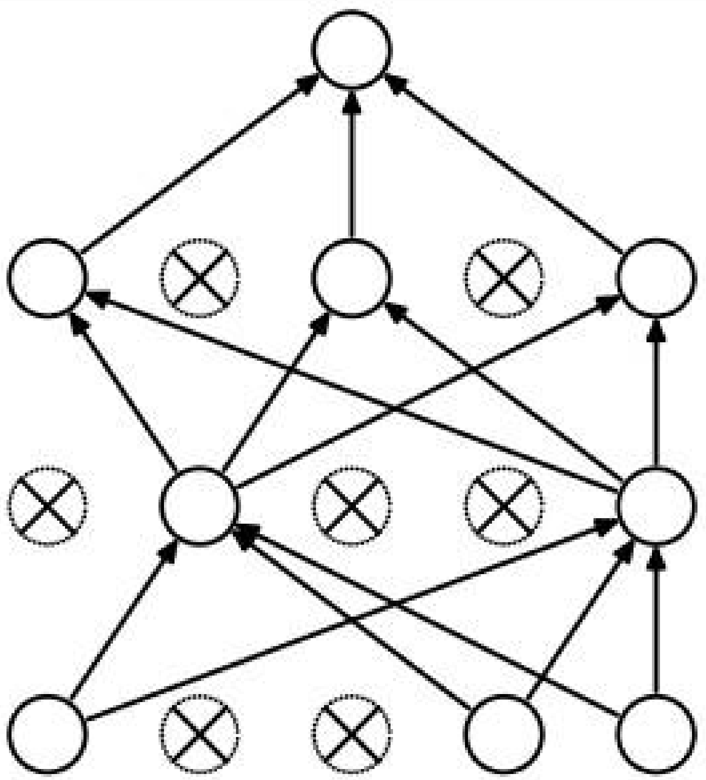
Dropout

In [3]:

```
depth = [4, 8]
class ConvN
    def __ini
        super(
        self.co
        self.po
        self.co
        self.fc1
        self.fc2
    def forwa
        x = F.re
        x = self
        x = F.re
        x = self
        x = x.vi
        x = F.re
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        x = F.log_softmax(x)
        return x
```



这是一个标准的神经网络



使用了dropout(0.5)之后的神经网络

一种防止过拟合的技术

注意，dropout只在训练的时候使用，测试的时候不使用

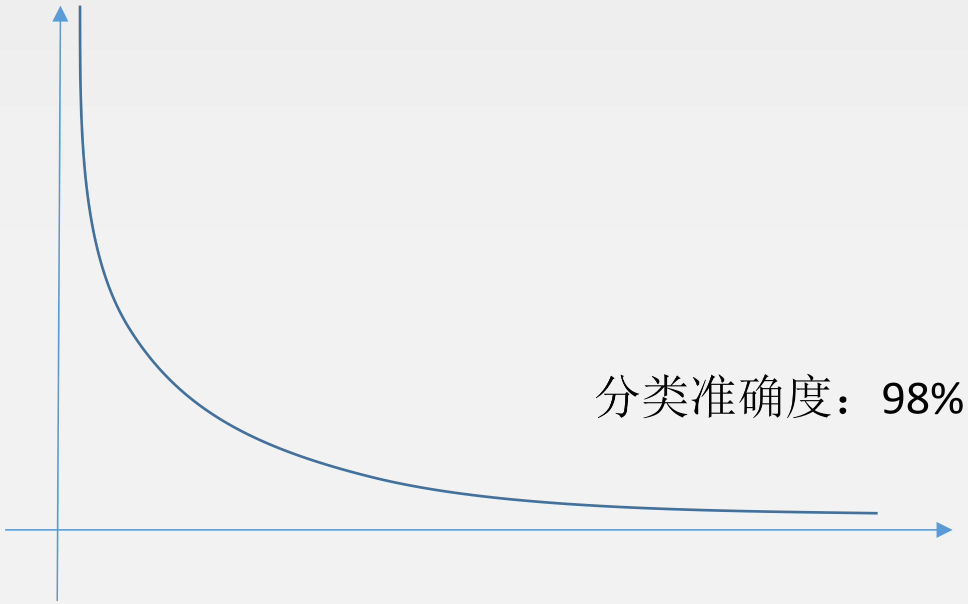
训练循环 / 测试

```
# 训练循环
for epoch in range(num_epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data), Variable(target)
        # 模型在训练集上训练
        net.train()
        output = net(data)
        loss = criterion(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            #打印和记录数据

#在测试集上运行
net.eval()
test_loss = 0
correct = 0
for data, target in test_loader:
    data, target = Variable(data), Variable(target)
    output = net(data)
    val = rightness(output, target)
```

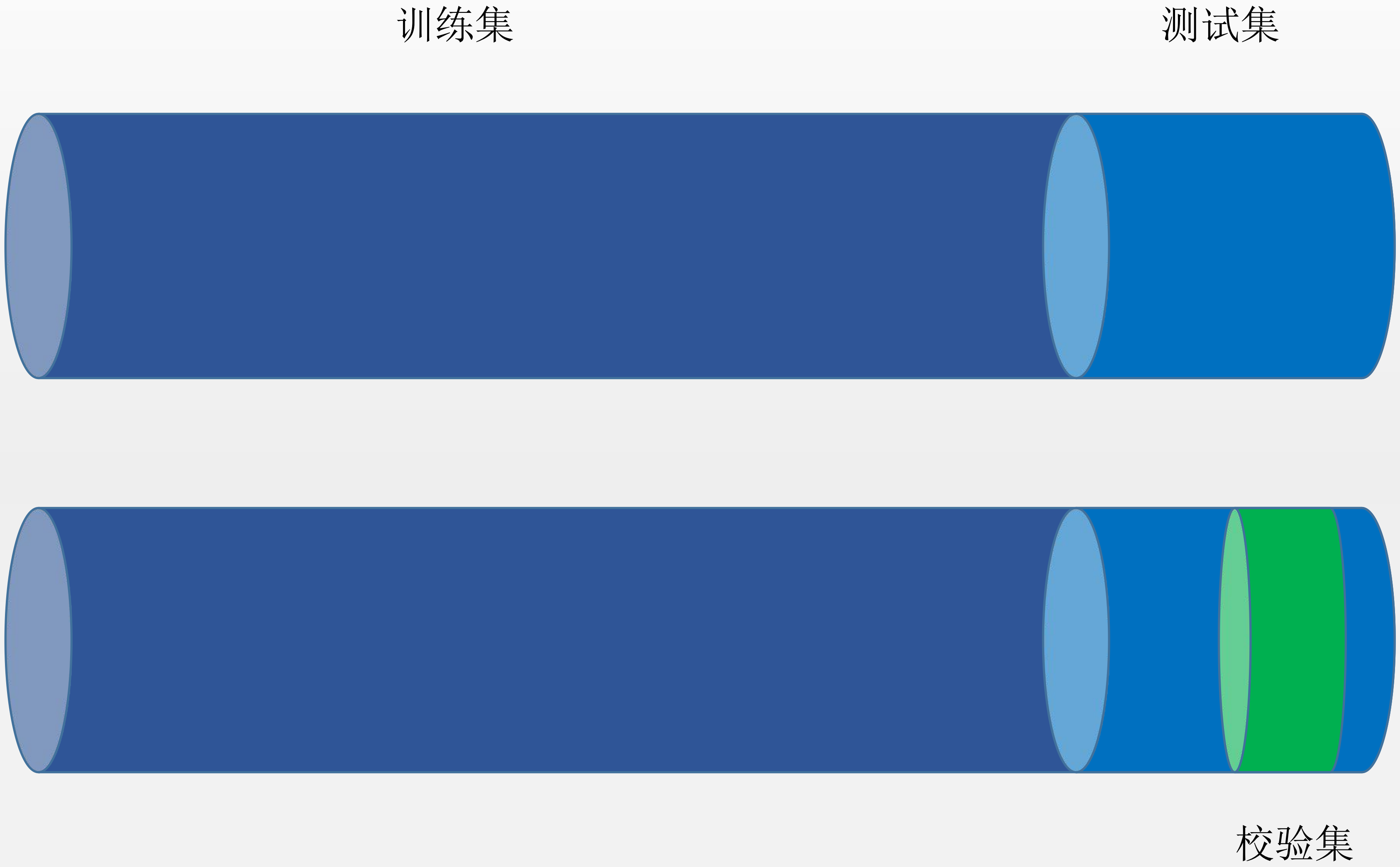
```
def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.pool(x)
    x = F.relu(self.conv2(x))
    x = self.pool(x)
    x = x.view(-1, image_size // 4 * image_size // 4 * depth[1])
    x = F.relu(self.fc1(x))
    x = F.dropout(x, training=self.training)
    x = self.fc2(x)
    x = F.log_softmax(x)
    return x
```

存在问题

- 我们是否还能进一步提高分类的准确程度？
- 如何验证当前的网络是否已经过拟合？
- 训练应该到什么时候停止？

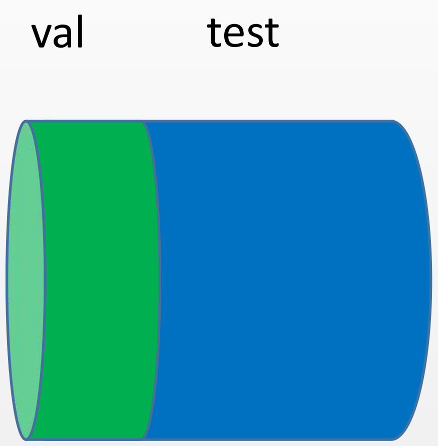


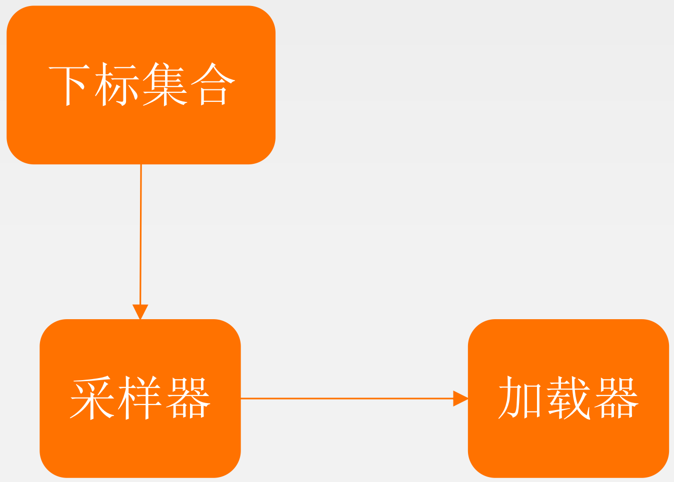
训练集 / 校验集（validation/develop） / 测试集



构建数据集

```
# 首先，我们定义下标数组indices，它相当于对所有test_dataset中数据的编码
# 然后定义下标indices_val来表示校验集数据的那些下标，indices_test表示测试集的下标
indices = range(len(test_dataset))
indices_val = indices[:5000]
indices_test = indices[5000:]

# 根据这些下标，构造两个数据集的SubsetRandomSampler采样器，它会对下标进行采样
sampler_val = torch.utils.data.sampler.SubsetRandomSampler(indices_val)
sampler_test = torch.utils.data.sampler.SubsetRandomSampler(indices_test)

# 根据两个采样器来定义加载器，注意将sampler_val和sampler_test分别赋值给了validation_loader和test_loader
validation_loader = torch.utils.data.DataLoader(dataset =test_dataset,
                                                batch_size = batch_size,
                                                shuffle = True,
                                                sampler = sampler_val
                                               )
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          sampler = sampler_test
                                         )
```



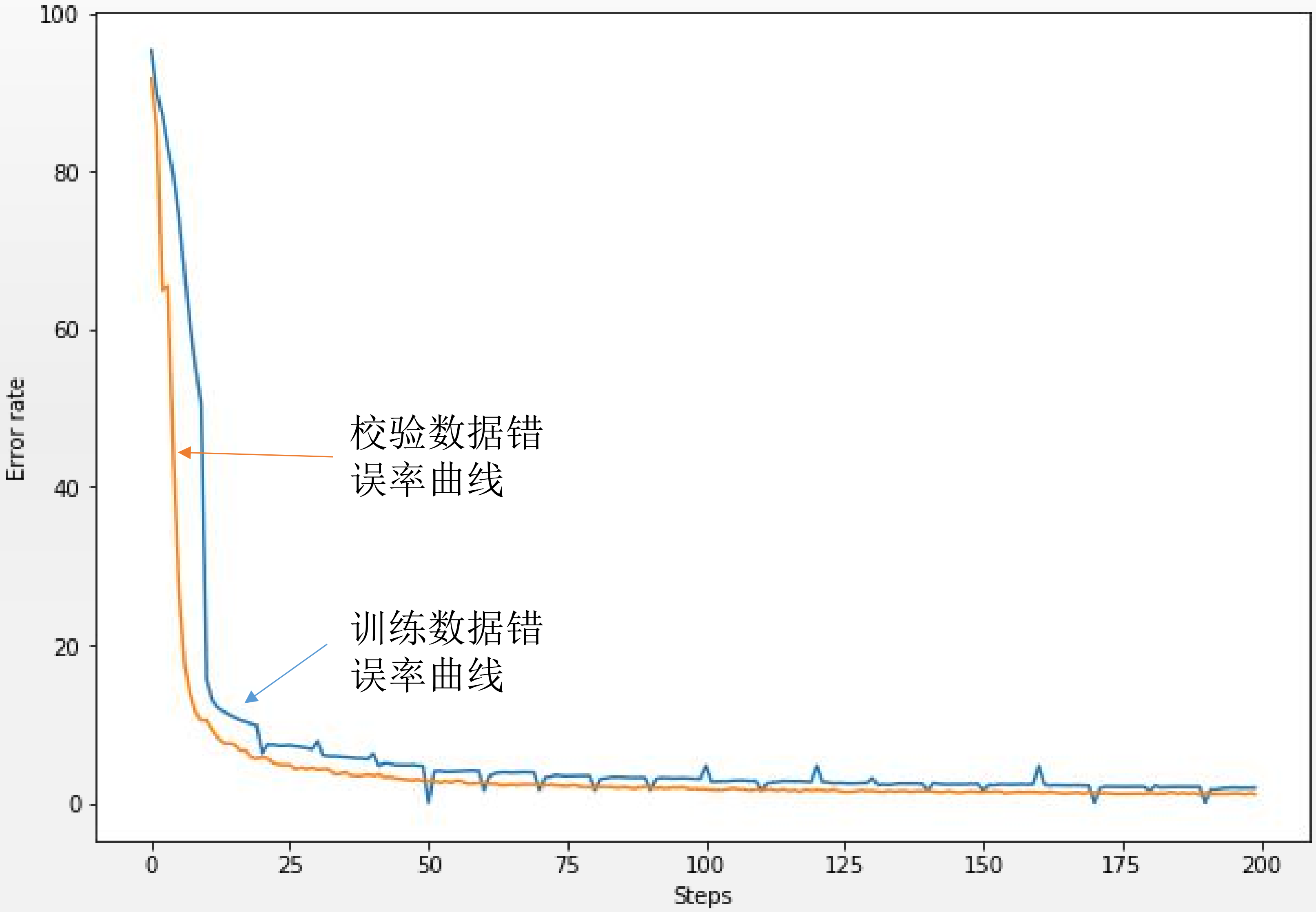


训练 / 校验循环

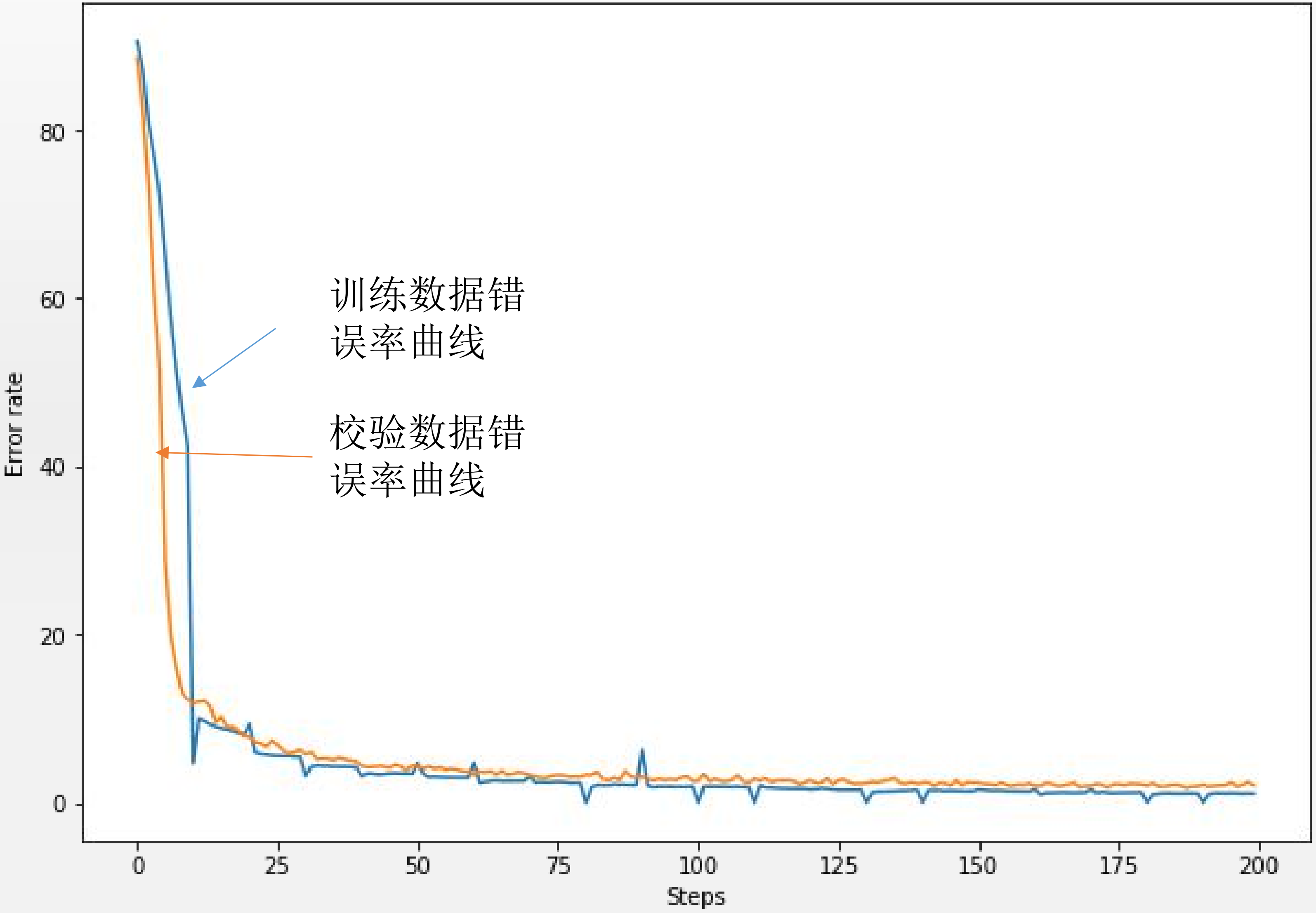
```
for epoch in range(num_epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data), Variable(target)
        # 模型在训练集上训练
        net.train()
        output = net(data)
        loss = criterion(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            #校验集上测试
            net.eval()
            val_rights = []
            for (data, target) in validation_loader:
                data, target = Variable(data), Variable(target)
                output = net(data) #完成一次预测
                right = rightness(output, target)
```

```
sampler_val = torch.utils.data.sampler.SubsetRandomSampler(indices_val)
validation_loader = torch.utils.data.DataLoader(dataset =train_dataset,
                                                batch_size = batch_size,
                                                shuffle = False,
                                                sampler = sampler_val
                                               )
```

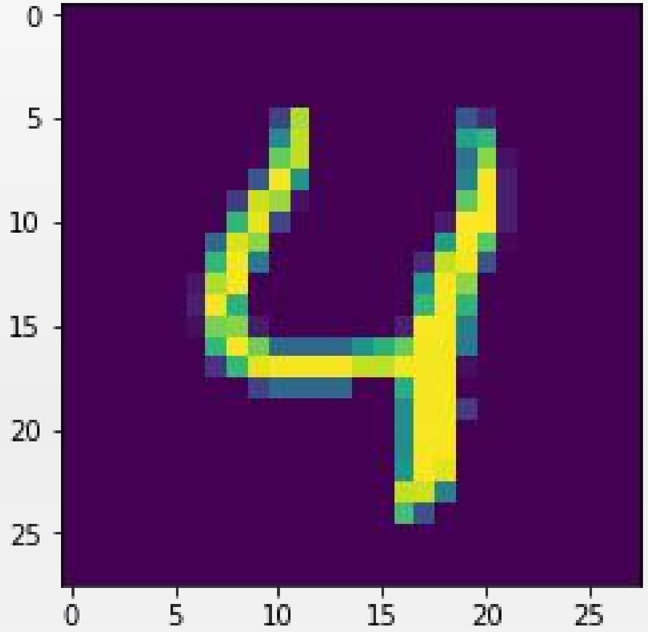
训练结果


100
80
60
40
20
0
Error rate
0
25
50
75
100
125
150
175
200
Steps
校验数据错
误率曲线
训练数据错
误率曲线

训练结果—关闭dropout



测试结果



输入

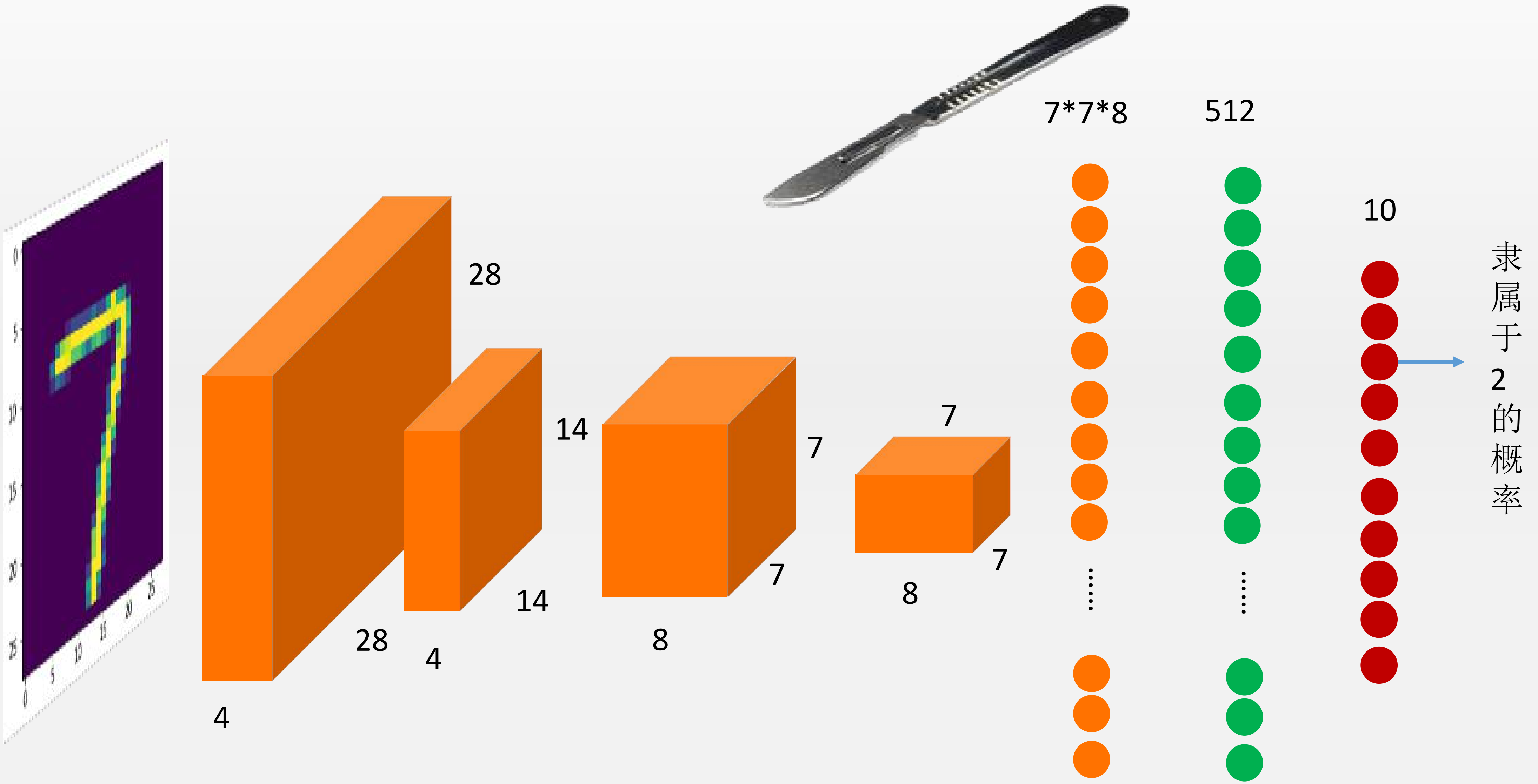
4

- 测试准确度： 0.986

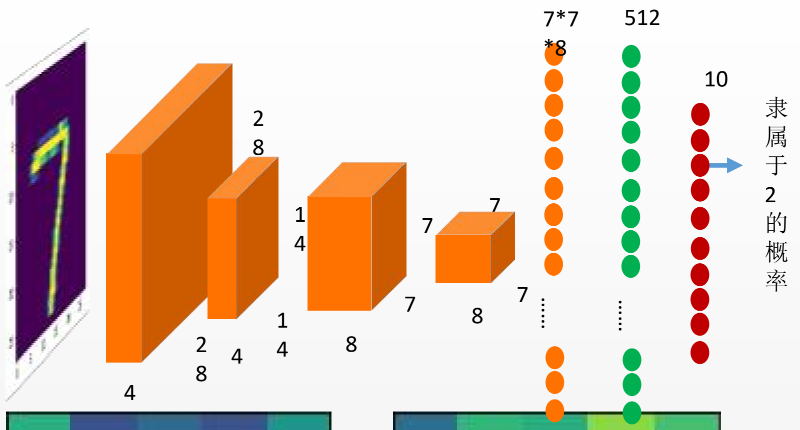
存在问题

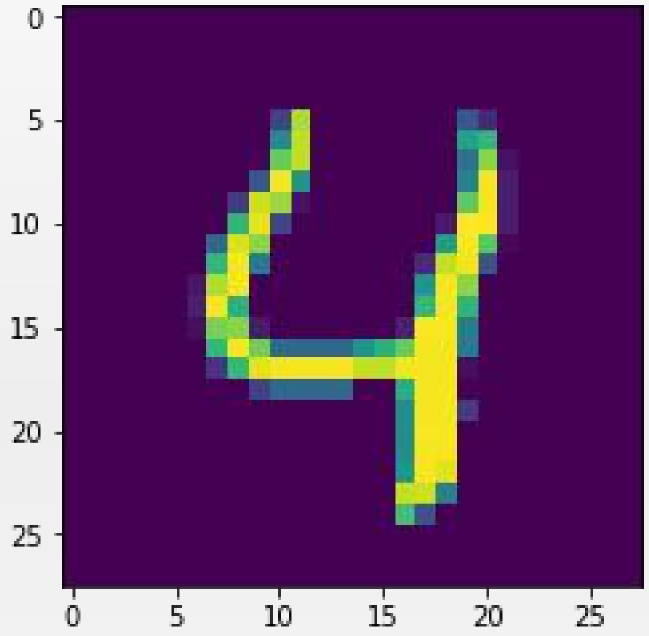
- 我们是否还能进一步提高分类的准确程度？ 可以，增加层数！
- 如何验证当前的网络是否已经过拟合？没有，因为校验曲线的误差率仍大于训练数据的误差率。
- 训练应该到什么时候停止？当校验和训练曲线相交的时候

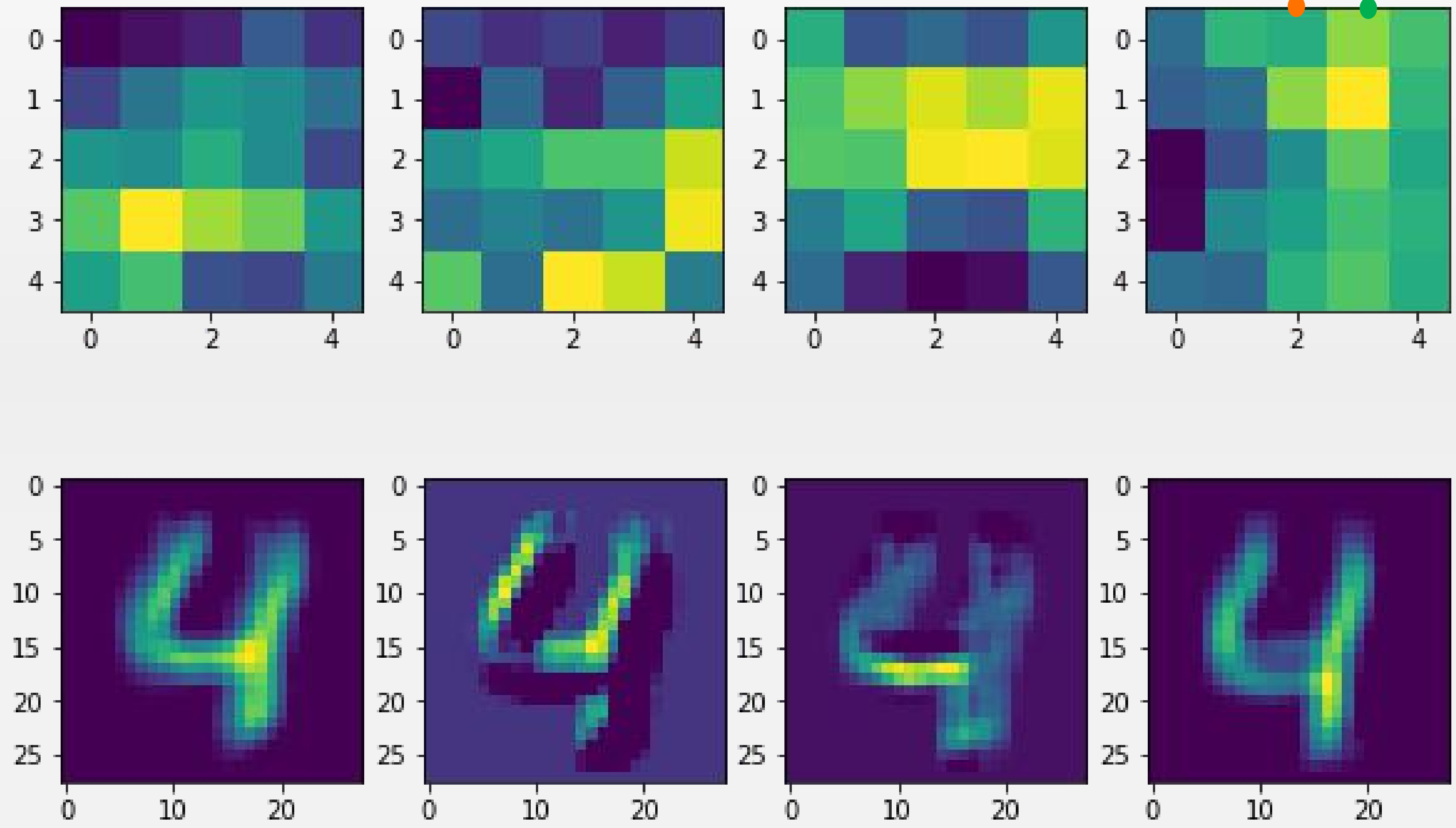
解剖卷积神经网



她学到了什么？卷积核与特征图







提取特征图的代码

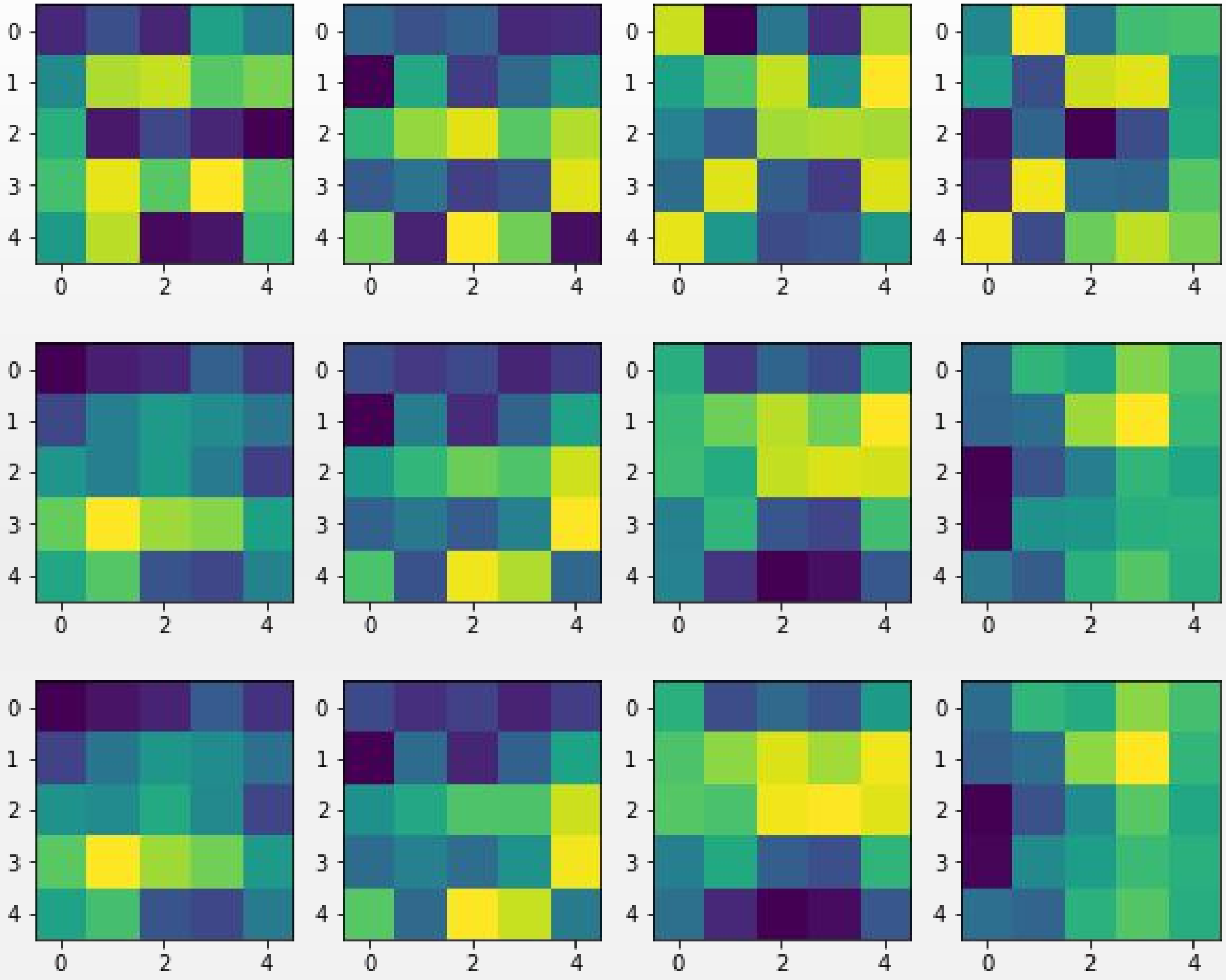
```
def retrieve_features(self, x):
        feature_map1 = F.relu(self.conv1(x))
        x = self.pool(feature_map1)
        feature_map2 = F.relu(self.conv2(x))
        return (feature_map1, feature_map2)

feature_maps = net.retrieve_features(Variable(test_dataset[idx][0].unsqueeze(0)))
```

学习过程

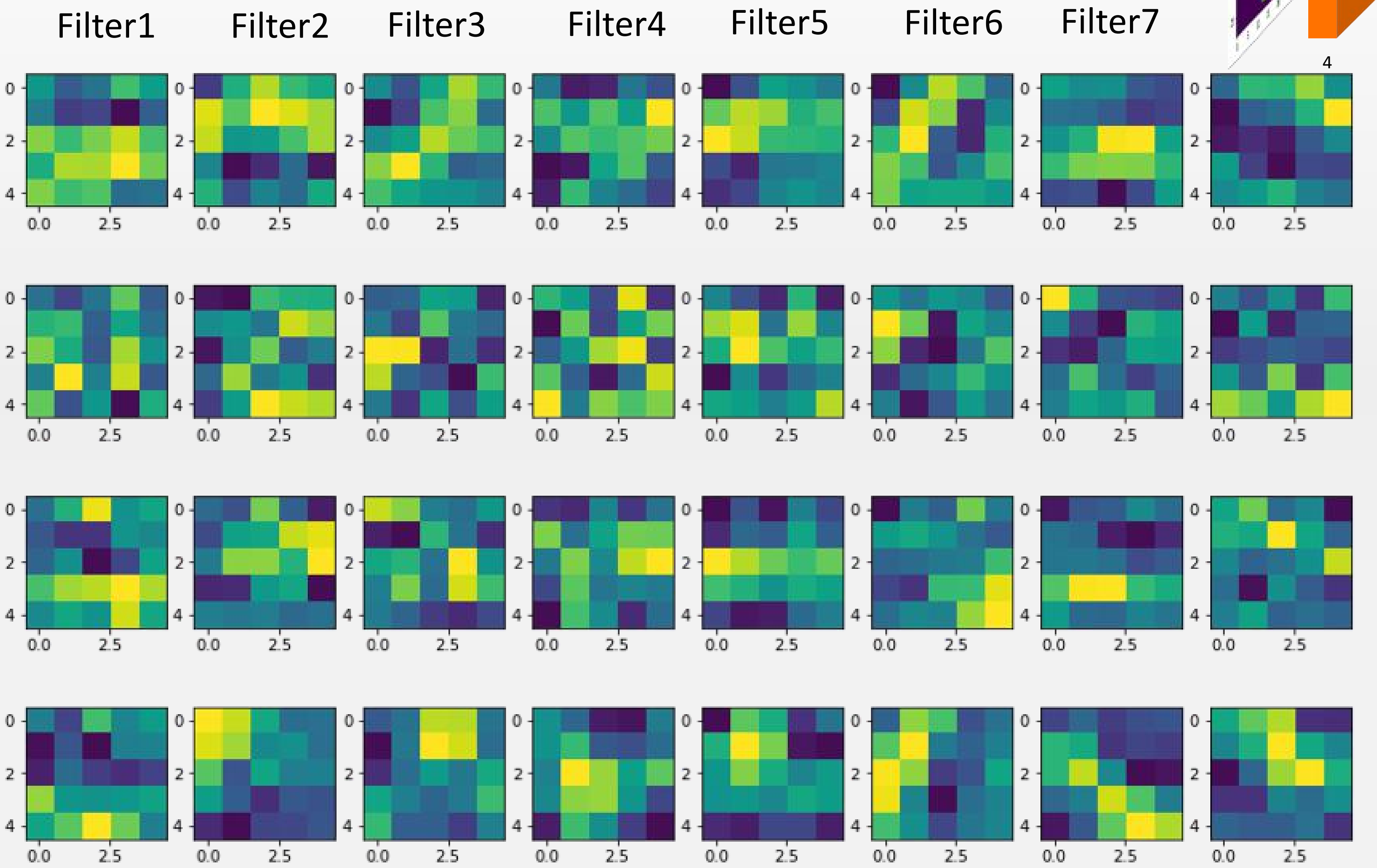
0

6000

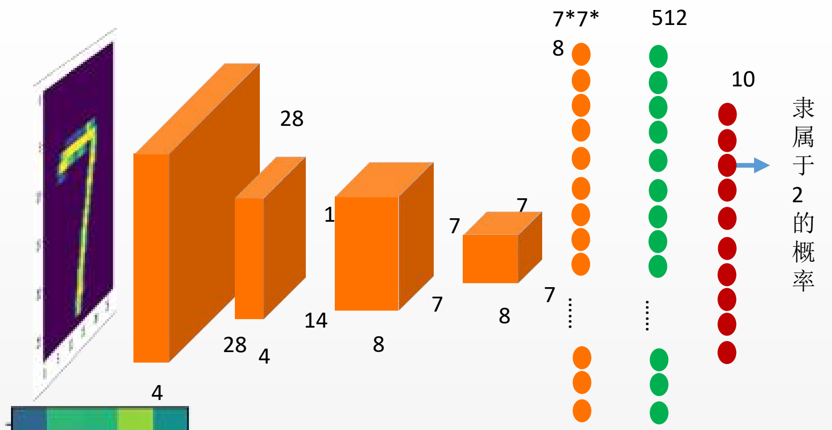
20000



她学到了什么？

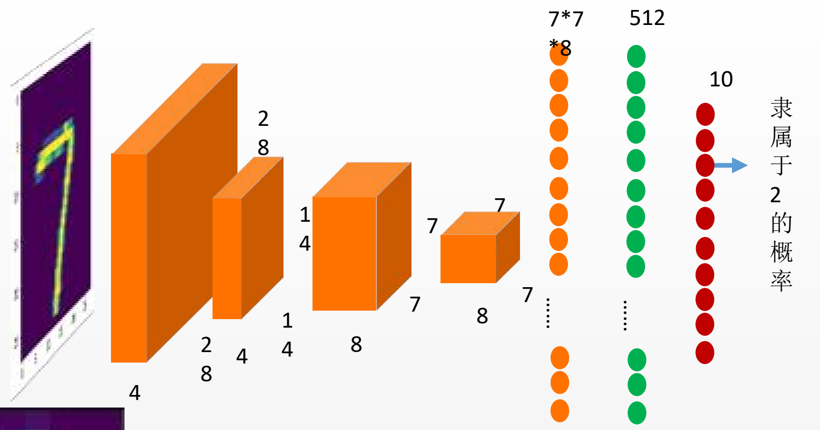


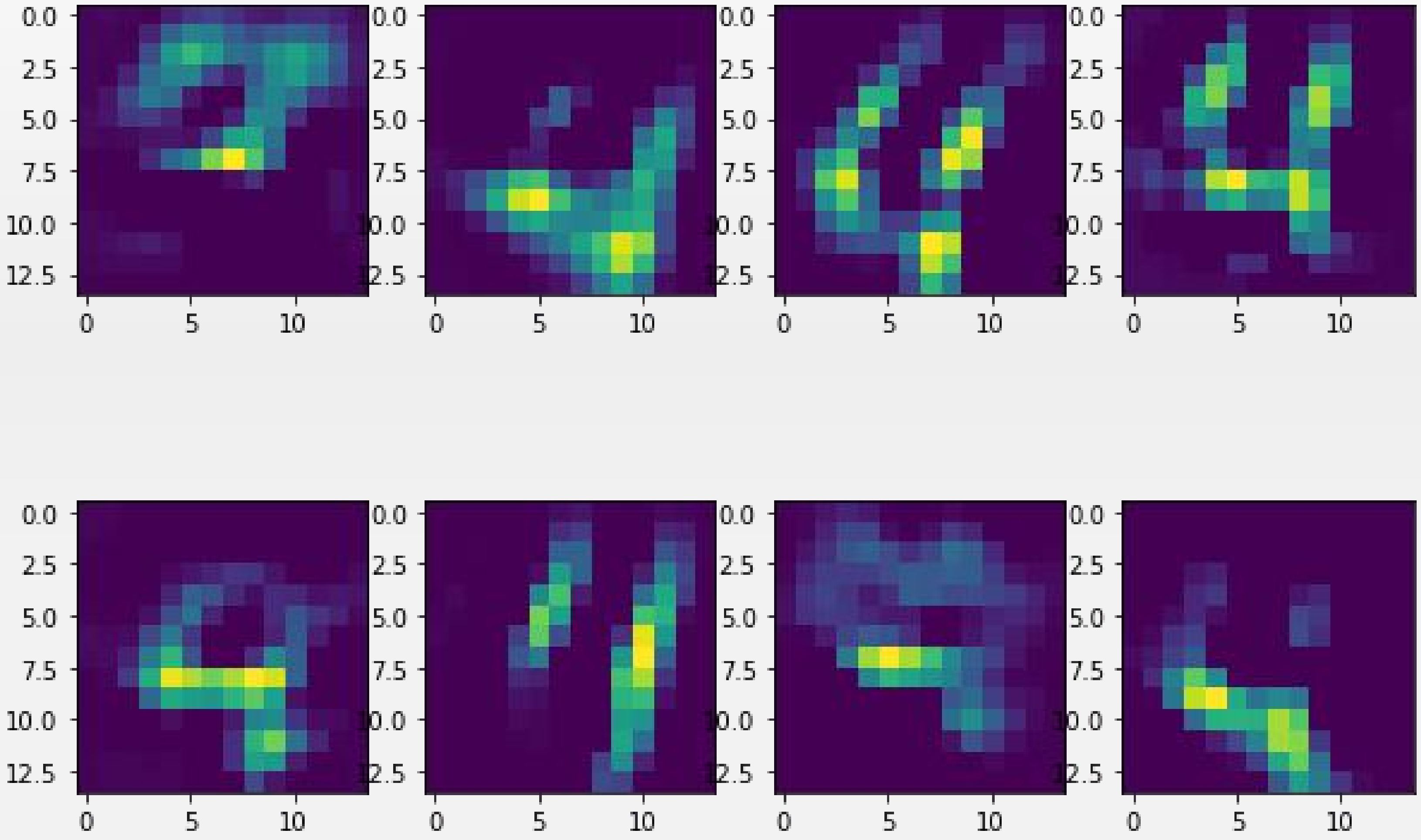
Filter1
Filter2
Filter3
Filter4
Filter5
Filter6
Filter7



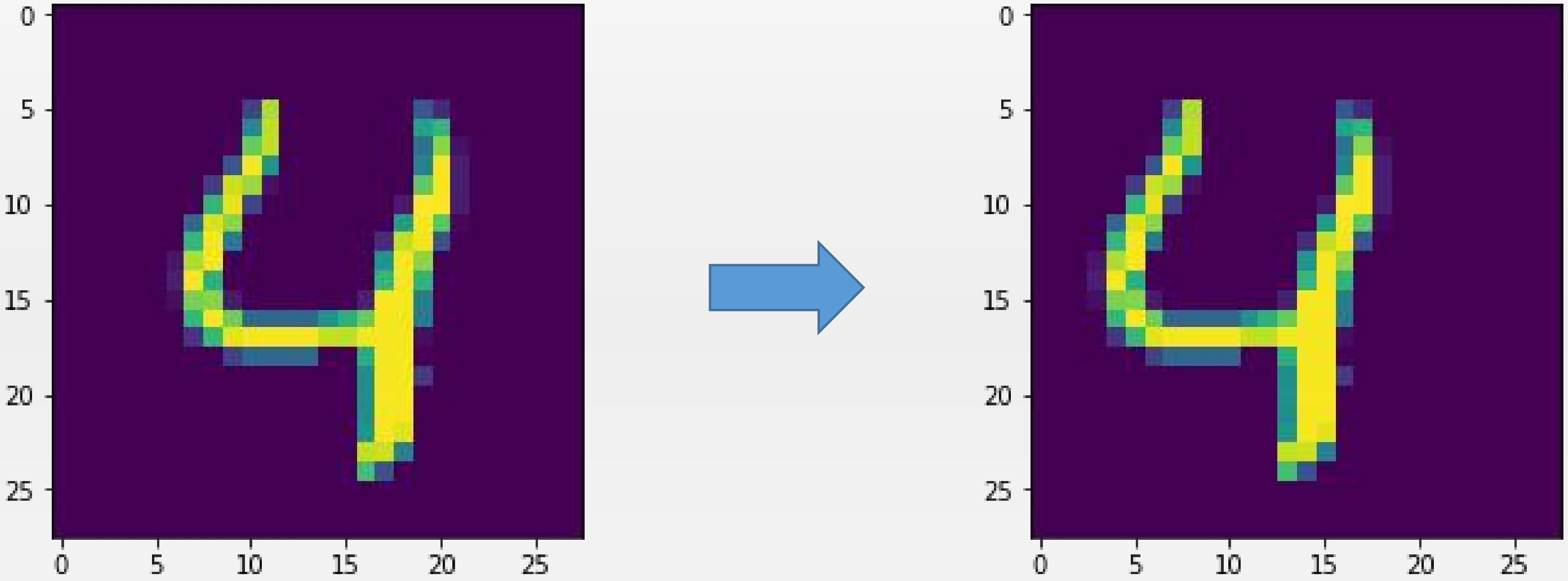
7*7*8
512
10
28
28
4
4
14
8
7
7
8
7
隶属于2的概率

第二层特征图

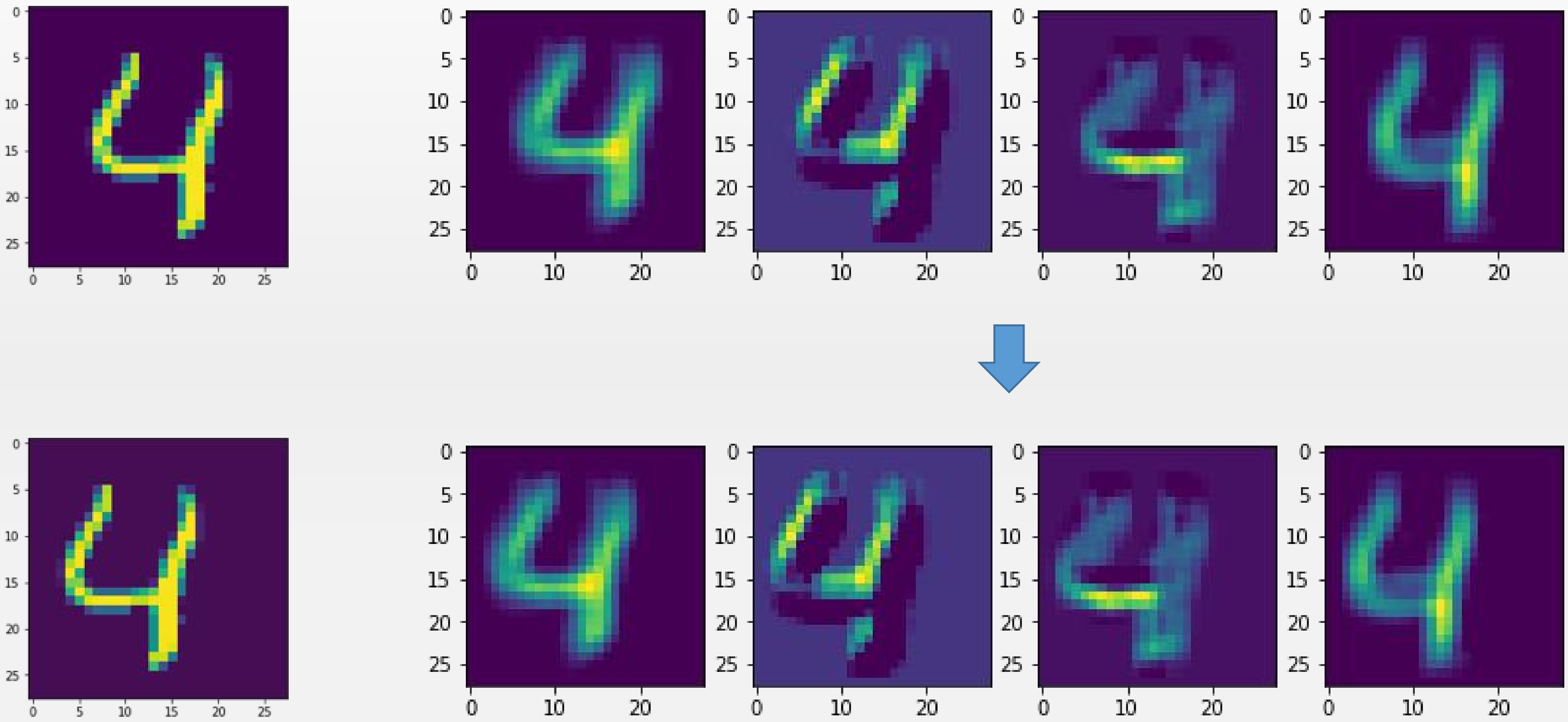




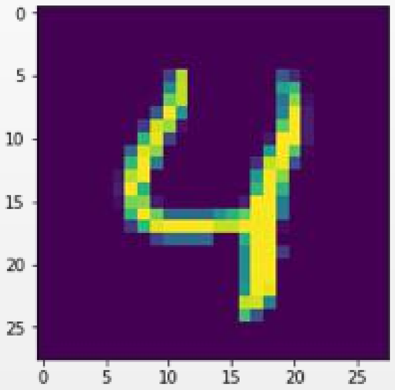
平移图像的鲁棒性

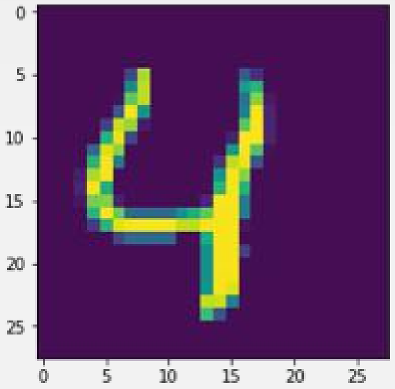


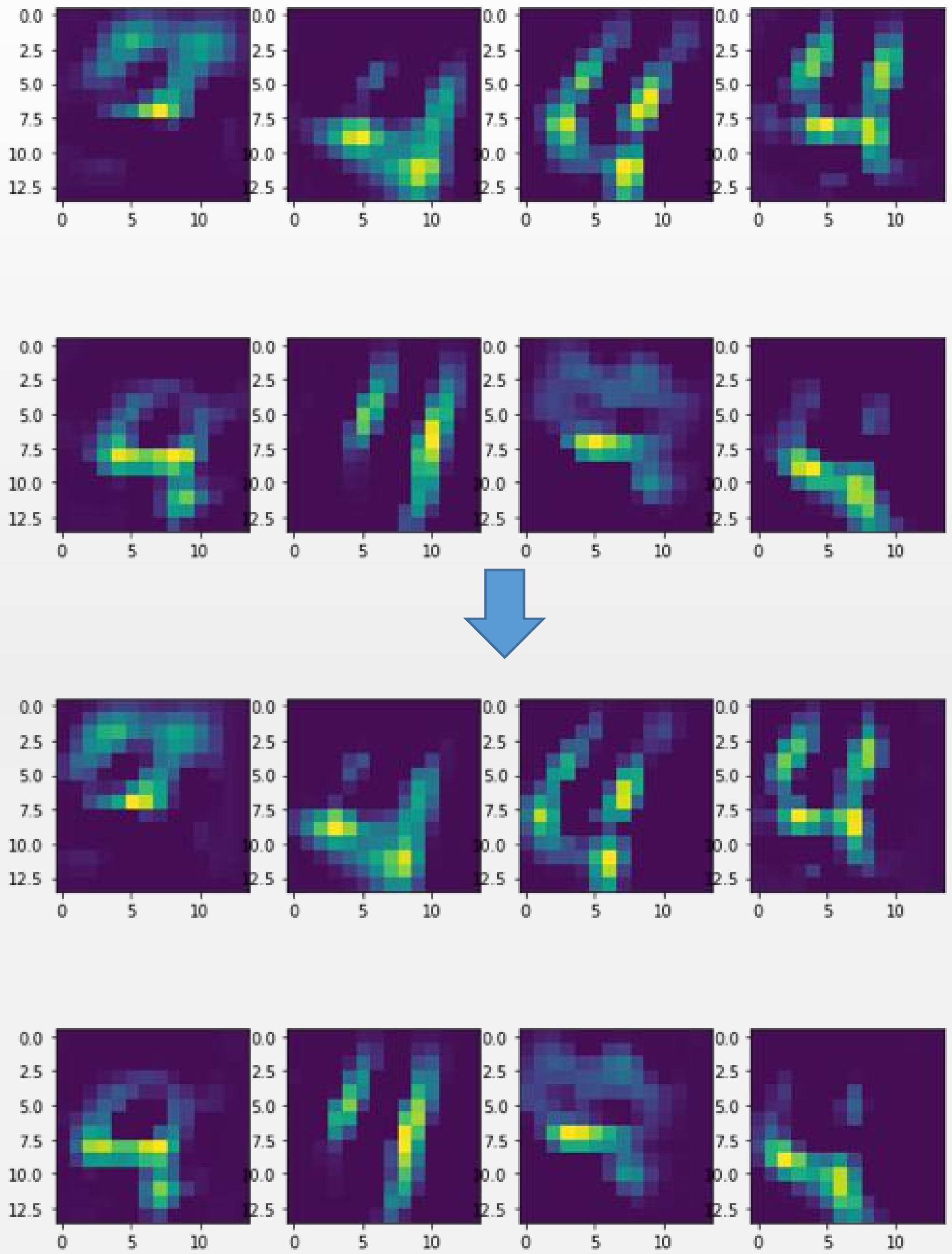
相应特征图的变化



相应特征图的变化





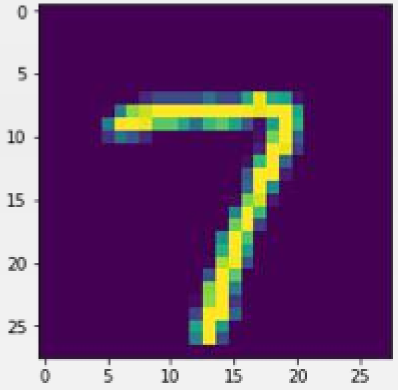


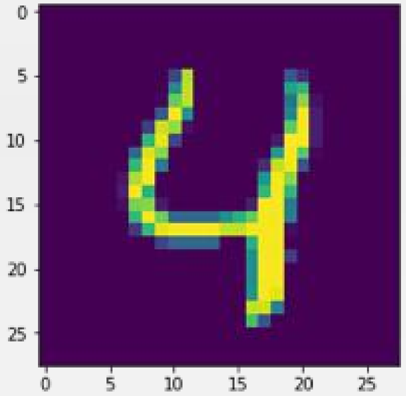
要点重述

- 可以将卷积过程理解为模板匹配
- 卷积核与特征图的对应
- 池化操作是一个粗糙的过程
- Dropout：防止过拟合的一种方法
- 数据集：训练、校验、测试
- 卷积神经网络的代码实现

作业：手写数字加法机

- 构造一个神经网络，和相应的数据集，实现：输入任意给定的两张手写数字图像对，输出一个数字为这两个数字的和





11