九曲阑干

AI深度学习

LeNet卷积神经网络详解

LeNet

九曲阑干

本节学习目录

- LeNet网络介绍
- LeNet网络应用
- LeNet各层网络结构分析
- LeNet网络总结

LeNet网络介绍

九曲阑干

LeNet

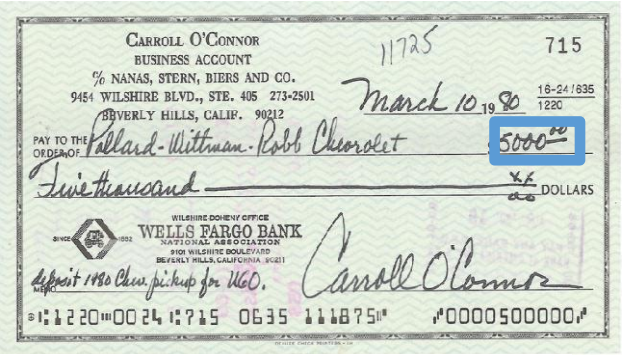
诞生于 1994 年，是最早的卷积神经网络之一，并且推动了深度学习领域的发展。

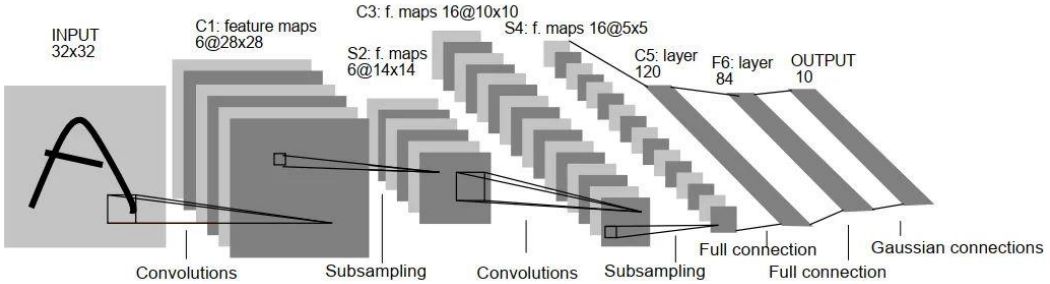
自从 1998 年开始，在许多次成功的迭代后，这项由 Yann LeCun 完成的开拓性成果被命名为 LeNet5。

LeNet网络应用

应用1：识别美国邮政服务提供的手写邮编

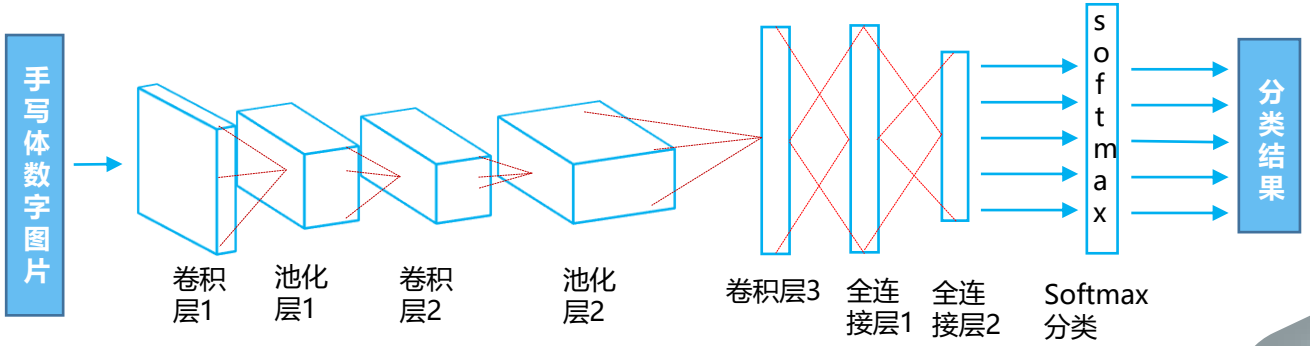
应用2：识别美国银行服务提供的手写支票





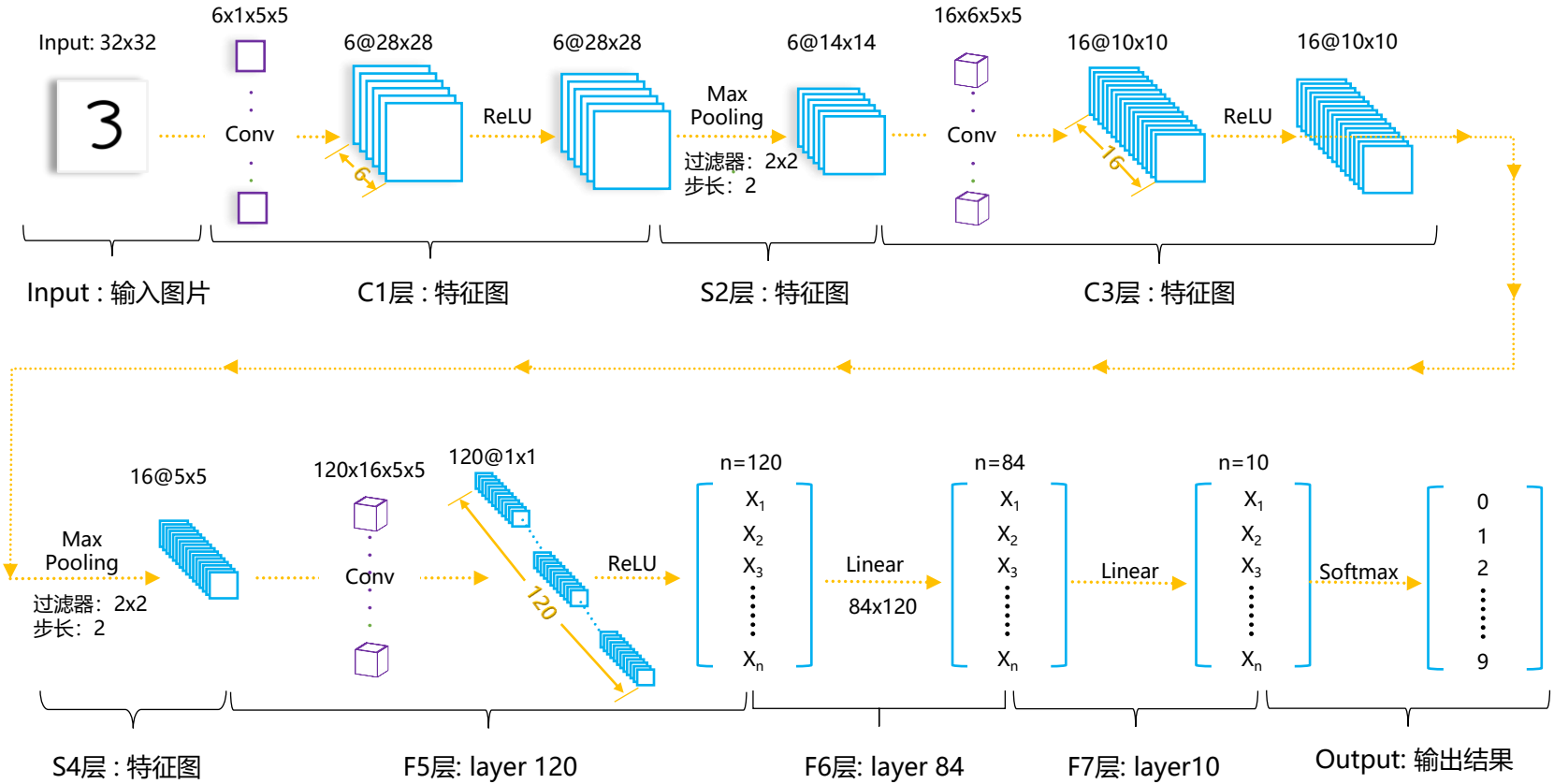
LeNet网络结构介绍

LeNet-5出自论文Gradient-Based Learning Applied to Document Recognition，是一种用于手写体字符识别的非常高效的卷积神经网络。

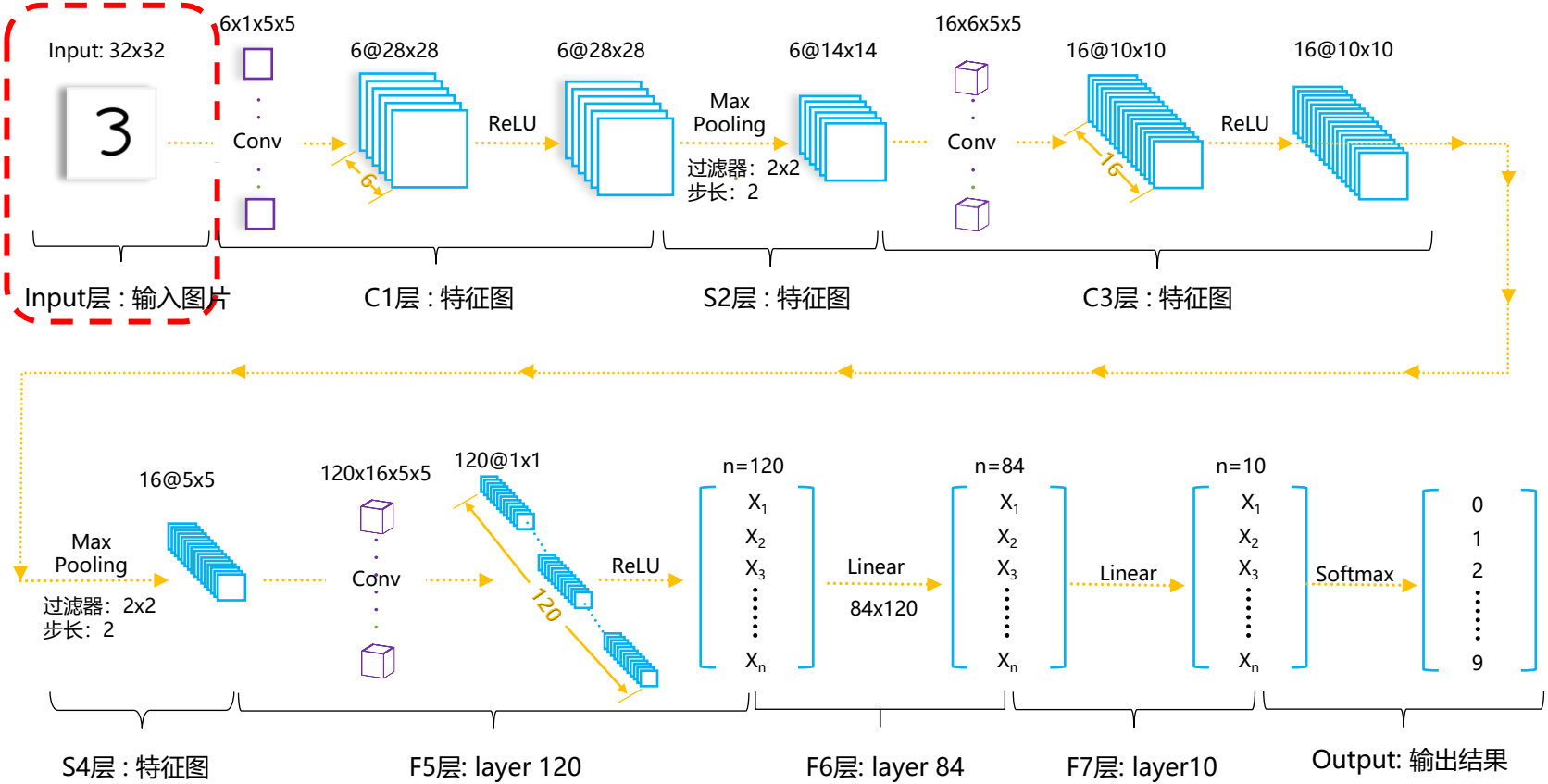


LeNet-5 卷积神经网络

LeNet网络结构分析

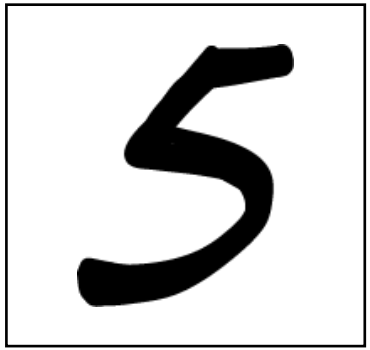


LeNet网络结构分析



LeNet网络结构分析

灰度图



32x32

32x32

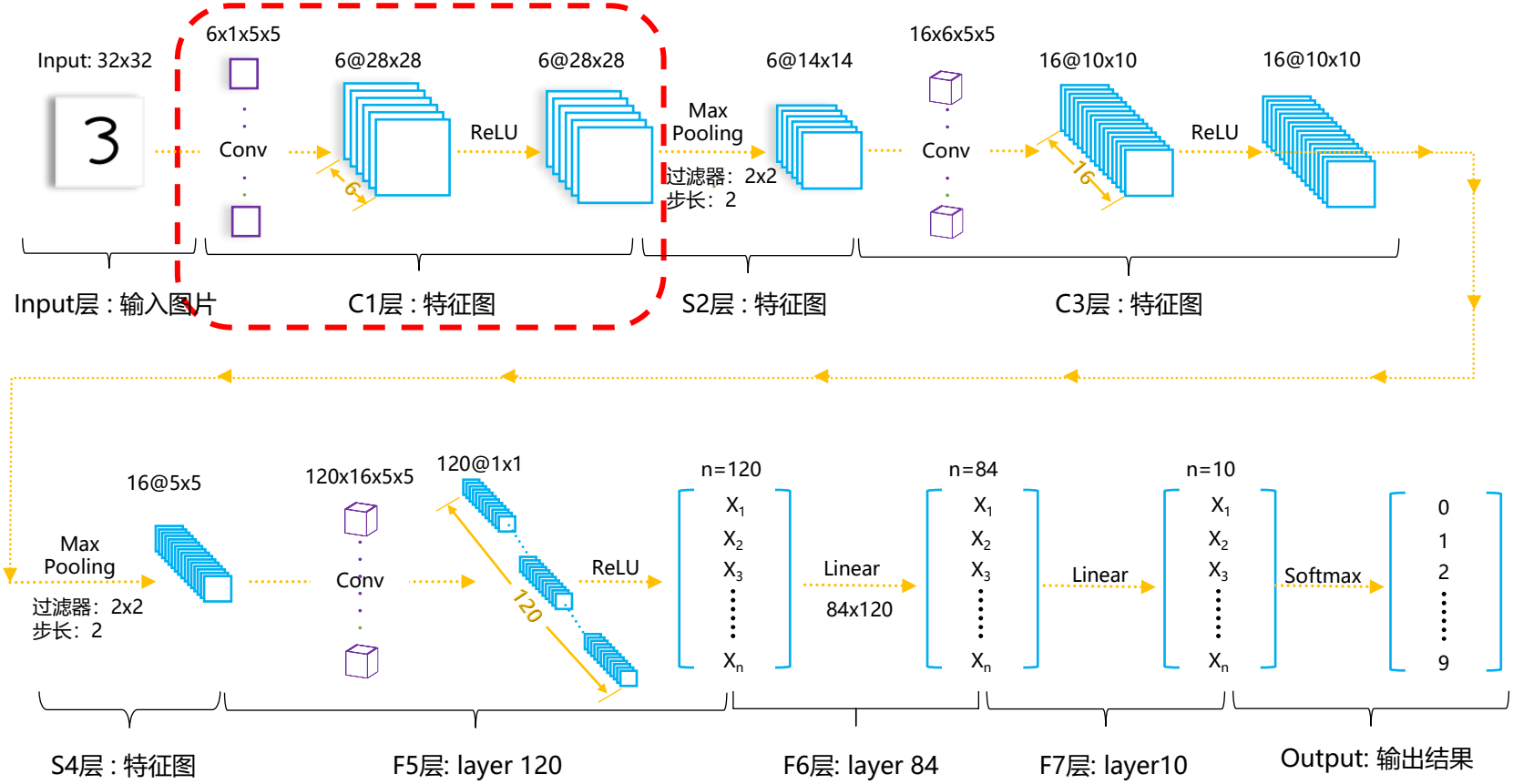
1表示纯白色，
0表示纯黑色

32x32

0	0	0	0	0	0	0	0	0	.....	0
0	0	0	0	0	0	0	0	0	.....	0
0	0	0	0	1	1	1	0	0	.....	0
0	0	0	0	1	0	0	0	0	.....	0
0	0	0	0	0	1	1	0	0	.....	0
0	0	0	0	0	0	1	0	0	.....	0
0	0	0	0	0	1	0	0	0	.....	0
0	0	0	1	1	0	0	0	0	.....	0
0	0	0	0	0	0	0	0	0	.....	0

示意图：32x32

LeNet网络结构分析



LeNet网络结构分析

C1层-卷积层

输入图片：32*32

卷积核大小：5*5

卷积核个数：6

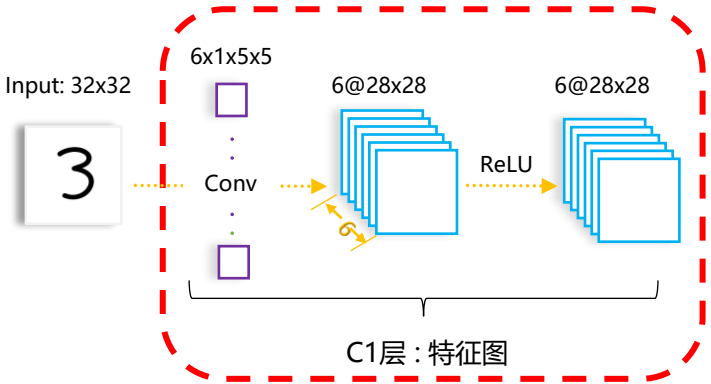
输出特征图大小：28*28 （32-5+1）=28

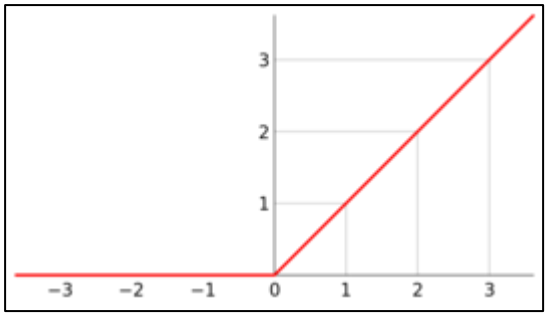
神经元数量：28*28*6

可训练参数： （5*5+1）* 6

（每个卷积核5*5=25个unit参数和一个bias参数，一共6个卷积核）

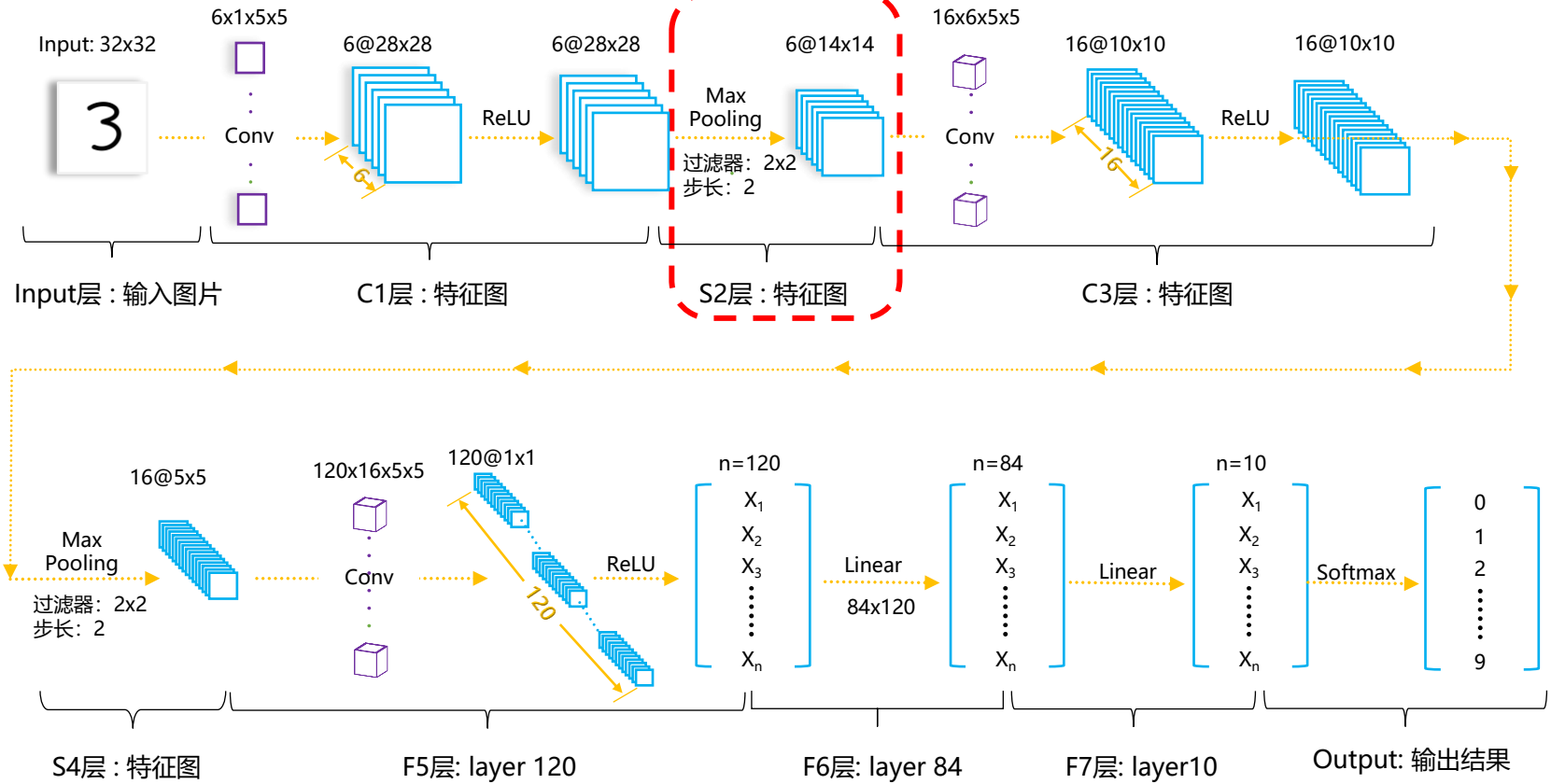
原始LeNet论文里使用了Sigmoid函数，我们代码中使用ReLU。
相比Sigmoid，ReLU更加不容易出现梯度消失的问题。





ReLU函数

LeNet网络结构分析



LeNet网络结构分析

S2层-池化层（下采样层）

输入：28*28

池化核（采样区域）：2*2

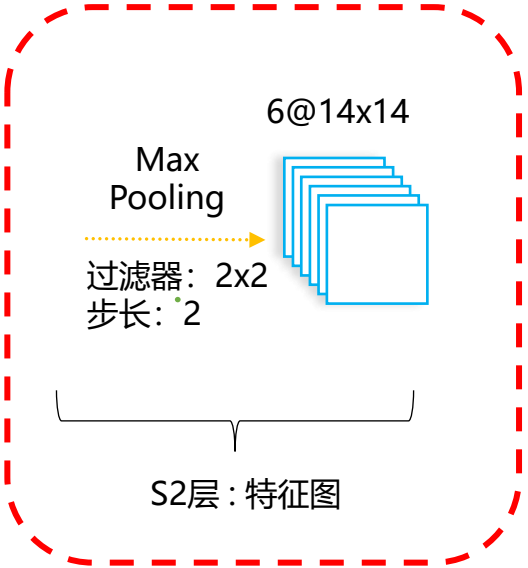
池化方式：最大池化

池化核（采样）个数：6

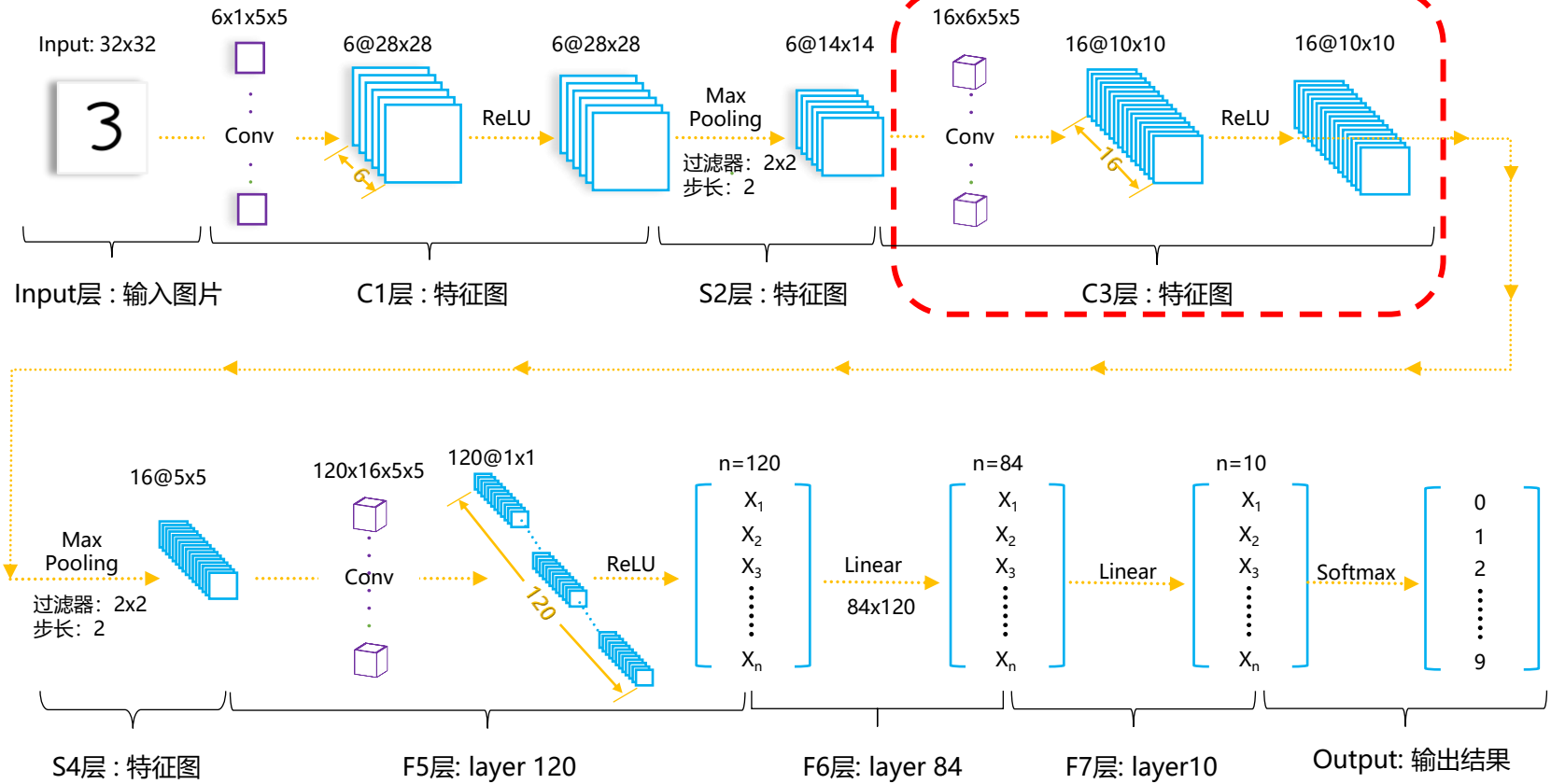
输出featureMap大小：14*14（28/2）

神经元数量：14*14*6

连接数：（2*2+1）*6*14*14



LeNet网络结构分析

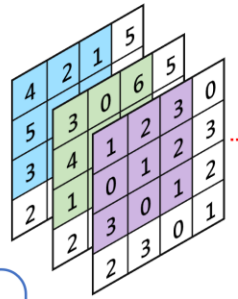


03. 卷积层 – 多通道卷积运算

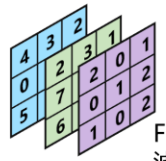
输入特征图维度： (N, Cin=3, Heigh=4,Width=4)

- 步长：1
- 卷积核尺寸(长宽)：3x3
- 卷积核通道数：3
- 卷积核个数：Cout

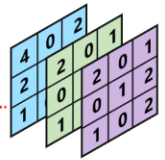
输入特征图维度： (N, Cout, Heigh=2,Width=2)



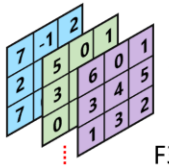
Input data
输入数据



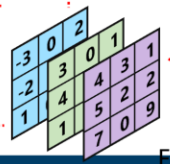
Filter 1
滤波器



F2

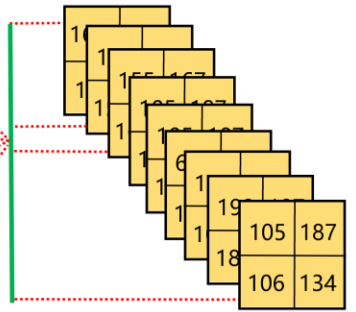


F3



F_Cout

一个卷积核提取一个特征，
多个卷积核产生多个特征



Output 1
Output 2
Output 3

Output
Cout

每个特征对应于一个通道，
卷积：多通道输入->多通道输出

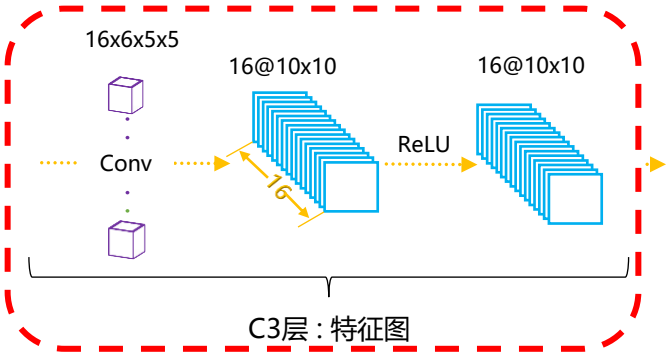
面试问题：多通道卷积（多个卷积核）的作用是什么？

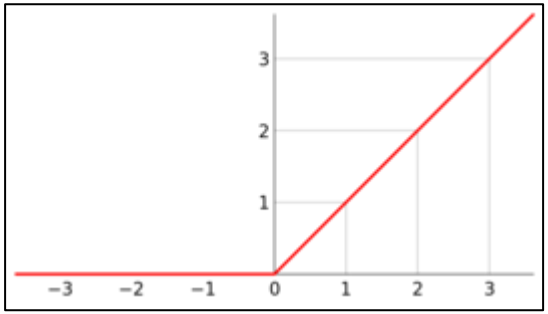
LeNet网络结构分析

C3层：卷积层
输入：S2的6个特征图
卷积核大小：5*5
卷积核个数：16
输出特征图大小：10*10 (14-5+1)=10

原始论文：16 个卷积核并不是都与 S2 的 6 个通道层进行卷积操作，如下图所示，C3 的前六个特征图（0,1,2,3,4,5）由 S2 的相邻三个特征图作为输入，对应的卷积核尺寸为：5x5x3；接下来的 6 个特征图（6,7,8,9,10,11）由 S2 的相邻四个特征图作为输入对应的卷积核尺寸为：5x5x4；接下来的 3 个特征图（12,13,14）号特征图由 S2 间断的四个特征图作为输入对应的卷积核尺寸为：5x5x4；最后的 15 号特征图由 S2 全部(6 个)特征图作为输入，对应的卷积核尺寸为：5x5x6。

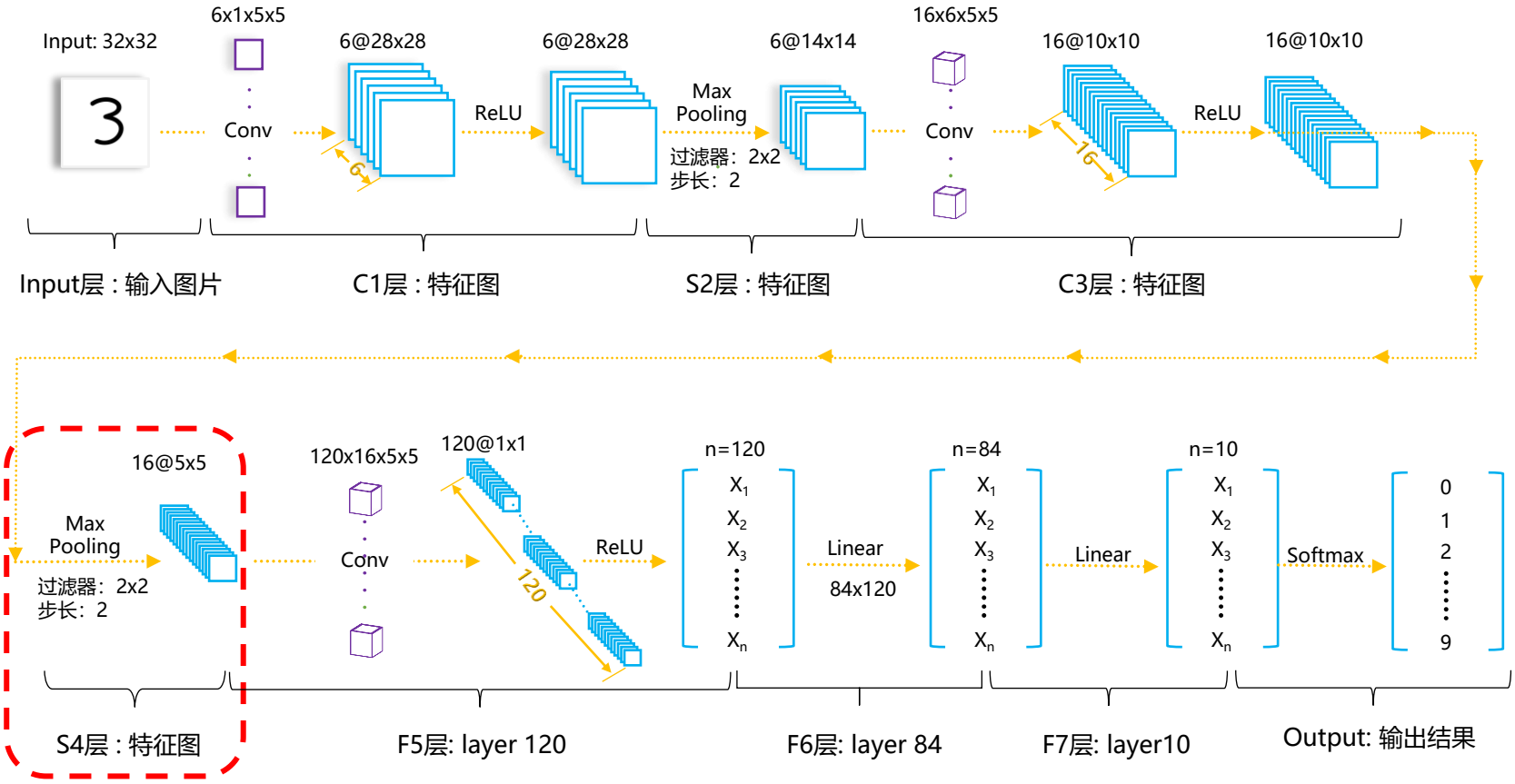
代码示例：6个输入特征图都参与计算16个输出特征图
可训练参数(6*5*5+1)*16=2416（每个卷积核6*5*5=150个weight参数和一个bias参数，一共6个卷积核）





ReLU函数

LeNet网络结构分析



LeNet网络结构分析

S4层-池化层（下采样层）

输入特征图矩阵：10*10

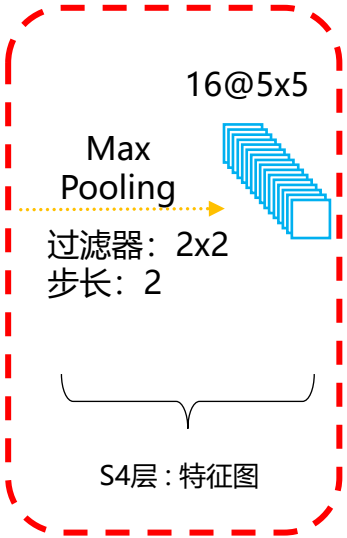
池化核大小：2*2

池化方式：最大池化

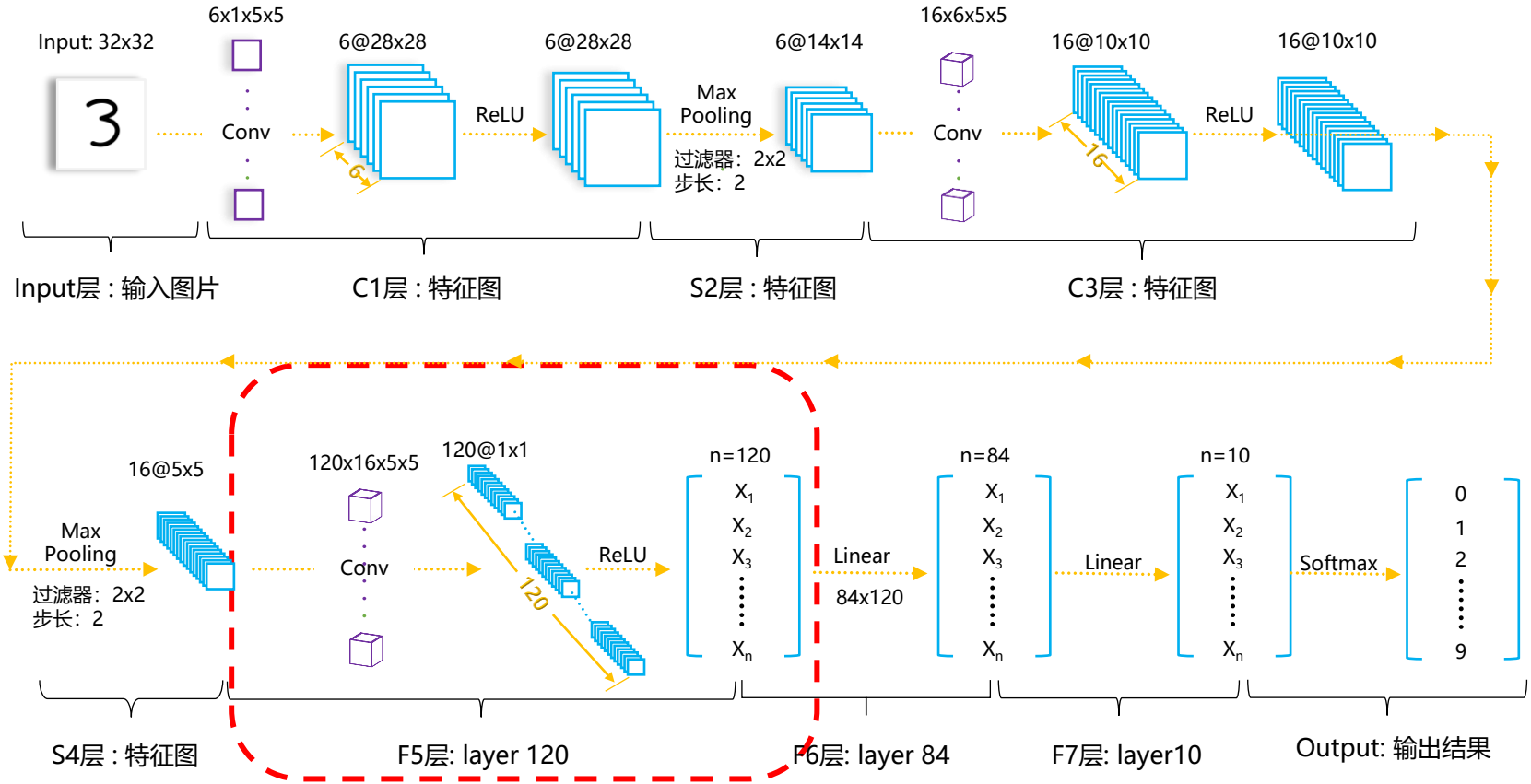
池化核个数：16

输出特征图矩阵大小：5*5（10/2）

神经元数量：5*5*16=400



LeNet网络结构分析



LeNet网络结构分析

F5层-用卷积实现的全连接层

输入：S4层的全部16个单元特征图（与S4全相连）

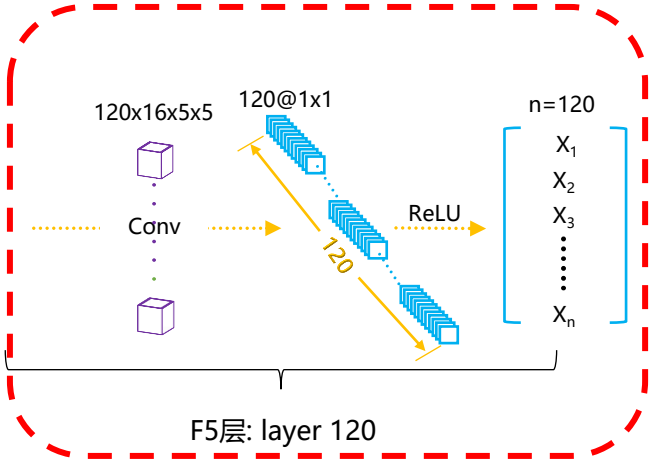
卷积核大小：5*5

卷积核个数：120

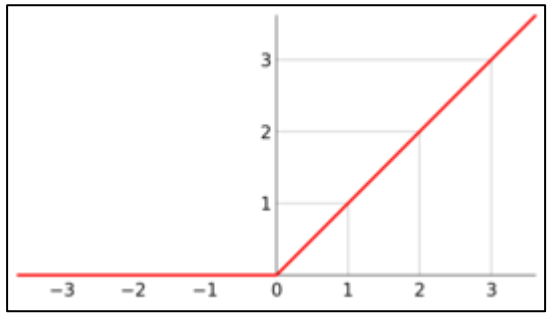
输出特征图大小：1*1（5-5+1）=1

可训练参数：(16*5*5+1)*120=48120（每个卷积核16*5*5个weight参数和一个bias参数，一共120个卷积核）

由于卷积核大小=输入特征图大小，本质上是全连接

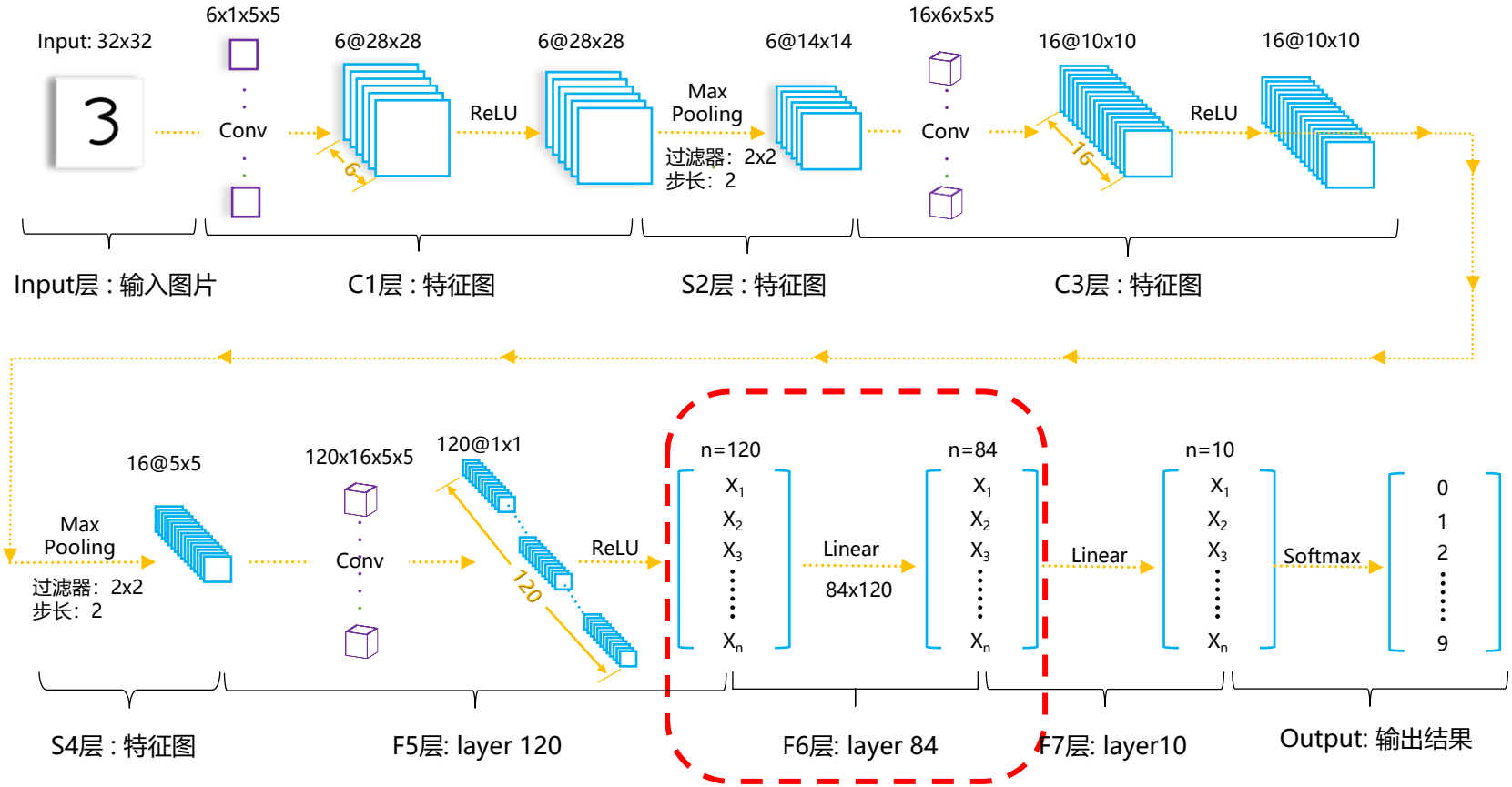


F5层: layer 120



ReLU函数

LeNet网络结构分析



LeNet网络结构分析

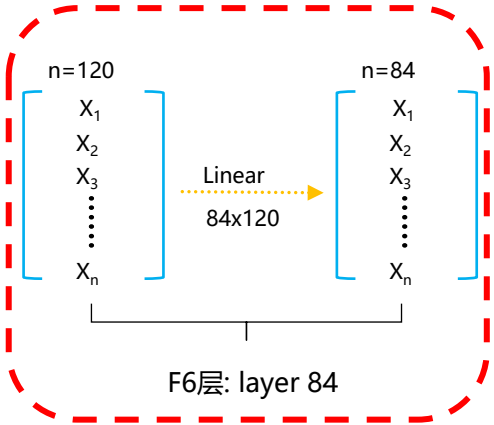
F6层-全连接层

输入：120维向量

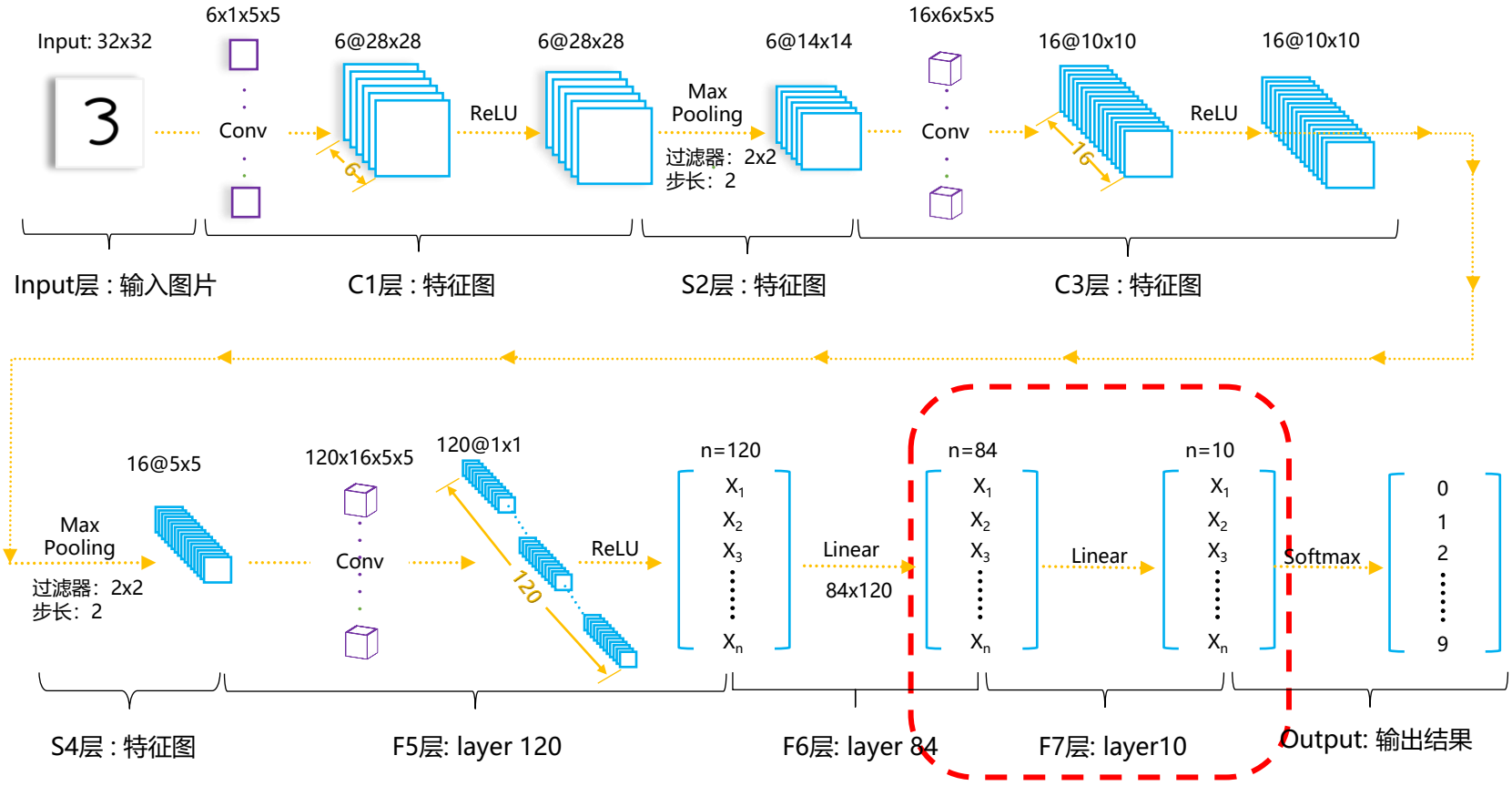
计算方式：计算输入向量和权重向量之间的点积，再加上一个偏置，结果通过sigmoid函数输出。

输出：84维向量

可训练参数:84*(120+1)=10164



LeNet网络结构分析



LeNet网络结构分析

F7层-全连接层

输入：84维向量

计算方式：计算输入向量和权重向量之间的点积，再加上一个偏置，结果通过Softmax函数输出。

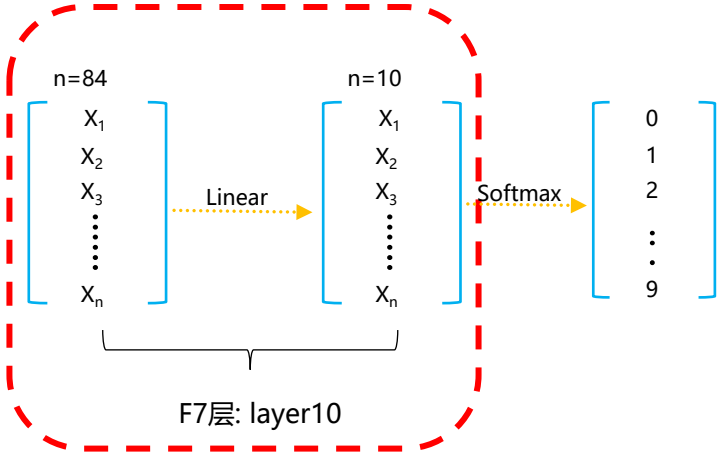
输出：10维向量

可训练参数: 84 × 10+10 = 850

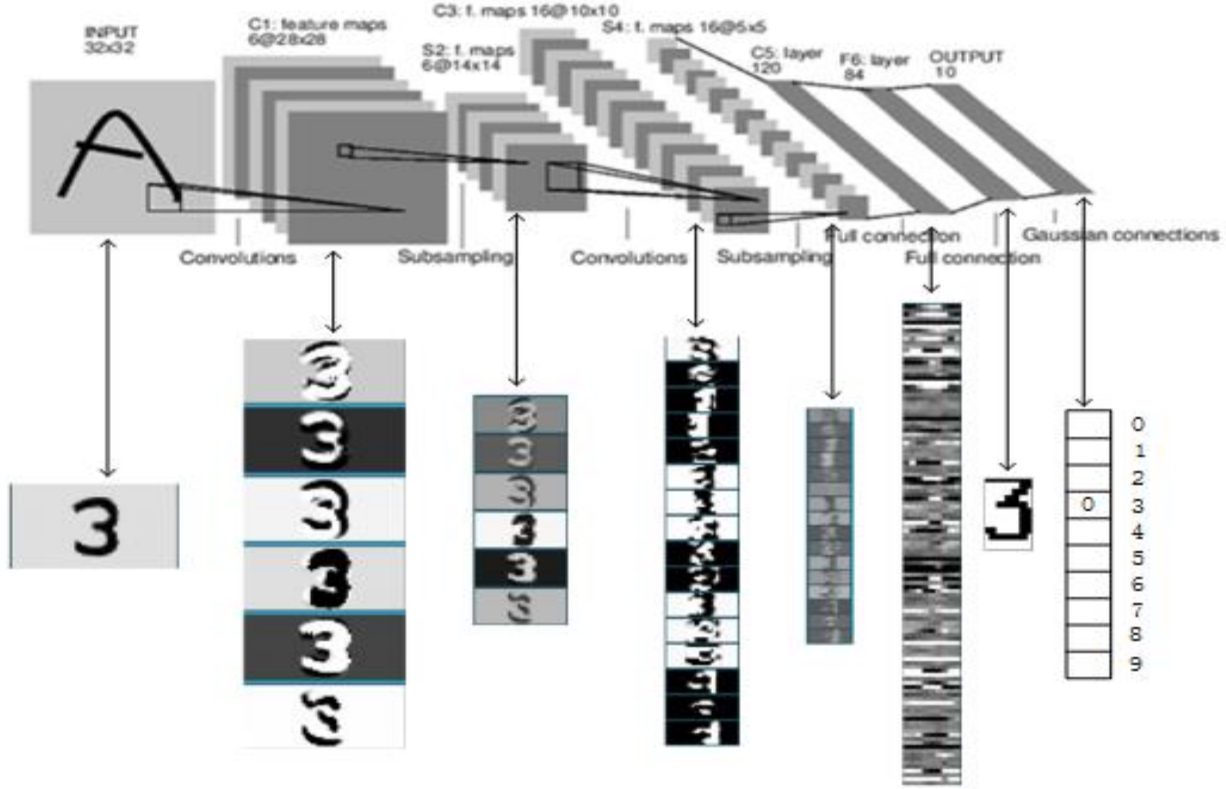
备注：原始论文中使用Gaussian Connections，采用了 RBF 函数（即径向欧式距离函数），计算输入向量和参数向量之间的欧式距离（目前已经被Softmax 取代）

$$y_i = \sum_j (x_j - w_{ij})^2$$

RBF函数



LeNet网络总结


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections
0
1
2
3
4
5
6
7
8
9

```
    F6 - 84
    tanh
    F7 - 10 (Output)
    """
    def __init__(self):
        super(LeNet5, self).__init__()

        self.convnet = nn.Sequential(OrderedDict([
            ('c1', nn.Conv2d(1, 6, kernel_size=(5, 5))),
            ('relu1', nn.ReLU()),
            ('s2', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c3', nn.Conv2d(6, 16, kernel_size=(5, 5))),
            ('relu3', nn.ReLU()),
            ('s4', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c5', nn.Conv2d(16, 120, kernel_size=(5, 5))),
            ('relu5', nn.ReLU())
        ]))

        self.fc = nn.Sequential(OrderedDict([
            ('f6', nn.Linear(120, 84)),
            ('relu6', nn.ReLU()),
            ('f7', nn.Linear(84, 10)),
        ]))

    def forward(self, img):
        output = self.convnet(img)
        output = output.view(img.size(0), -1)
        output = self.fc(output)
        return output
                                                    43,1          Bot
```

```python
def recognition():
    print("begin to recogition")
    #test_img = Image.open('./test.png')
    test_img = Image.open('./3.png')
    resize_img = test_img.resize((28,28))
    gray_img = resize_img.convert('L')

    #squeeze()
    #unsqueeze()
    trans_img = transform(gray_img).unsqueeze(0)
    input_img = trans_img.to(device)

    result = predict(input_img)

    #tensor : result
    print(result)

    #Return : result is a tensor
    #item() convert a tensor to number
    print("result is :" + str(result.item()))

if __name__ == "__main__":
    recognition()
```

```python
def predict(image):

    net = LeNet5()
    net.load_state_dict(torch.load('../train/params.pkl', map_location='cpu'))
    net.to(device)
    #torch.no_grad()

    #input_img
    output = net(image)

    print("output:" +str(output))
    print("output shape:"+str(output.shape))
    print("output size:"+str(output.size()))

    # argument: Output, 0/1
    # 0/1: column-0; row-1
    # Return:a namedtuple(values, indices)
    ret_val, predicted = torch.max(output, 1)
    _, predicted = torch.max(output, 1)


    print("ret_val:"+str(ret_val))
    print("predict:"+str(predicted))

    return predicted;
```