

**深 度 学 习 实 验**

**实 验 报 告**

**实验名称：卷积神经网络**

**姓 名：李岚琦**

**学 号：2111078**

**专 业：智能科学与技术**

**人工智能学院**

**2023年11月**

目录

[一、 问题简述 2](#_Toc151975546)

[二、 实验目的 2](#_Toc151975547)

[三、 编译环境 2](#_Toc151975548)

[四、 基础代码 2](#_Toc151975549)

[1. 导入所需的包和数据集： 2](#_Toc151975550)

[2. 定义网络结构，实现LeNet： 2](#_Toc151975551)

[3. 定义使用gpu训练的函数： 3](#_Toc151975552)

[4. 定义训练函数： 3](#_Toc151975553)

[5. 训练并可视化： 4](#_Toc151975554)

[五、 基础要求与附加题 4](#_Toc151975555)

[1. 基础要求 4](#_Toc151975556)

[要求1 5](#_Toc151975557)

[要求2 5](#_Toc151975558)

[要求3 6](#_Toc151975559)

[要求4 8](#_Toc151975560)

[要求5 10](#_Toc151975561)

[要求6 11](#_Toc151975562)

[要求7 11](#_Toc151975563)

[要求8 11](#_Toc151975564)

[2. 附加题 11](#_Toc151975565)

[要求1 11](#_Toc151975566)

[要求2 11](#_Toc151975567)

[要求3 11](#_Toc151975568)

[要求4 11](#_Toc151975569)

[要求5 11](#_Toc151975570)

# 问**题简**述

利用以LeNet为代表的卷积神经网络，对MINIST数据集进行分类，并且探究更换不同的网络结构和采用不同的学习策略对训练和测试结果的影响。

# 实验目的

1. 探究LeNet结构下，选用不同激活函数对训练结果的影响

2. 探究LeNet结构下，选用不同卷积方式对训练结果的影响

3. 探究LeNet结构下，选用不同参数与优化器对训练结果的影响

4. 探究将LeNet的部分结构替换，对训练结果的影响

# 编译环境

语言：python 版本： 3.7.16

编译器： vscode 版本：2023

深度学习框架 ： pytorch 版本： 1.10.1

工具包 ： d2l

# 基础代码

## 导入所需**的包和数据集**：

**import** torch

**from** matplotlib **import** pyplot **as** plt

**from** torch **import** nn

**from** d2l **import** torch **as** d2l

**import** dataset **as** ppd

**import** math

**import** numpy

d2l**.**use\_svg\_display()

device **=** torch**.**device('cuda')

batch\_size **=** 256

train\_iter**,** test\_iter **=** d2l**.**load\_data\_fashion\_mnist(**batch\_size=**batch\_size)

## 定义网络结构，实现LeNet：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**5**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**5)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 5 **\*** 5**,** 120)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(84**,** 10))

## 定义使用gpu训练的函数：

**def** evaluate\_accuracy\_gpu(**net,** **data\_iter,** **device=**None)**:** *#@save*

    """使用GPU计算模型在数据集上的精度"""

**if** isinstance(net**,** nn**.**Module)**:**

        net**.**eval()  *# 设置为评估模式*

**if** **not** device**:**

            device **=** next(iter(net**.**parameters()))**.**device

*# 正确预测的数量，总预测的数量*

    metric **=** d2l**.**Accumulator(2)

**with** torch**.**no\_grad()**:**

**for** X**,** y **in** data\_iter**:**

**if** isinstance(X**,** list)**:**

*# BERT微调所需的（之后将介绍）*

                X **=** [x**.**to(device) **for** x **in** X]

**else:**

                X **=** X**.**to(device)

            y **=** y**.**to(device)

            metric**.**add(d2l**.**accuracy(net(X)**,** y)**,** y**.**numel())

**return** metric[0] **/** metric[1]

## 定义训练函数：

*#@save*

**def** train\_ch6(**net,** **train\_iter,** **test\_iter,** **num\_epochs,** **lr,** **device**)**:**

    """用GPU训练模型(在第六章定义)"""

**def** init\_weights(**m**)**:**

**if** type(m) **==** nn**.**Linear **or** type(m) **==** nn**.**Conv2d**:**

            nn**.**init**.**xavier\_uniform\_(m**.**weight)

    net**.**apply(init\_weights)

    print('training on'**,** device)

    net**.**to(device)

    optimizer **=** torch**.**optim**.**SGD(net**.**parameters()**,** **lr=**lr)

    loss **=** nn**.**CrossEntropyLoss()

    animator **=** d2l**.**Animator(**xlabel=**'epoch'**,** **xlim=**[1**,** num\_epochs]**,**

**legend=**['train loss'**,** 'train acc'**,** 'test acc'])

    timer**,** num\_batches **=** d2l**.**Timer()**,** len(train\_iter)

**for** epoch **in** range(num\_epochs)**:**

*# 训练损失之和，训练准确率之和，样本数*

        metric **=** d2l**.**Accumulator(3)

        net**.**train()

**for** i**,** (X**,** y) **in** enumerate(train\_iter)**:**

            timer**.**start()

            optimizer**.**zero\_grad()

            X**,** y **=** X**.**to(device)**,** y**.**to(device)

            y\_hat **=** net(X)

            l **=** loss(y\_hat**,** y)

            l**.**backward()

            optimizer**.**step()

**with** torch**.**no\_grad()**:**

                metric**.**add(l **\*** X**.**shape[0]**,** d2l**.**accuracy(y\_hat**,** y)**,** X**.**shape[0])

            timer**.**stop()

            train\_l **=** metric[0] **/** metric[2]

            train\_acc **=** metric[1] **/** metric[2]

**if** (i **+** 1) **%** (num\_batches **//** 5) **==** 0 **or** i **==** num\_batches **-** 1**:**

                animator**.**add(epoch **+** (i **+** 1) **/** num\_batches**,**

                             (train\_l**,** train\_acc**,** None))

        test\_acc **=** evaluate\_accuracy\_gpu(net**,** test\_iter)

        animator**.**add(epoch **+** 1**,** (None**,** None**,** test\_acc))

    print(f'loss {train\_l:.3f}, train acc {train\_acc:.3f}, '

          f'test acc {test\_acc:.3f}')

    print(f'{metric[2] **\*** num\_epochs **/** timer**.**sum():.1f} examples/sec '

          f'on {str(device)}')

## 训练并可视化：

lr**,** num\_epochs **=** 0.9**,** 10

train\_ch6(net**,** train\_iter**,** test\_iter**,** num\_epochs**,** lr**,** d2l**.**try\_gpu())

plt**.**show()

# 基础要求与附加题

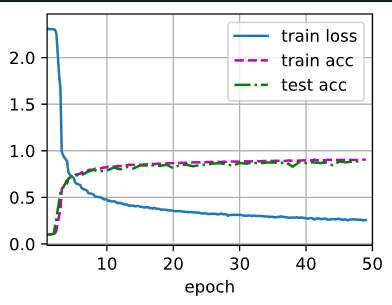
## 1. 基础要求

考虑到训练误差和测试误差，均是1减去相应的准确率，因此下文中损失函数使用train loss，而训练和测试效果均使用acc来表示。

### 要求1

使用LeNet，分别画出损失函数、训练误差、测试误差随迭代epoch的变化曲线。

将num\_epoch设置为50，运行上述代码得到：



并且查看最终效果：



可见，在原生的LeNet上面，最终train acc能够达到0.9左右。然而test acc比train acc略低，说明出现了过拟合现象。

### 要求2

将sigmond激活函数替换成ReLu。将网络结构更改为：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**5**,** **padding=**2)**,** nn**.**ReLU()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**5)**,** nn**.**ReLU()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

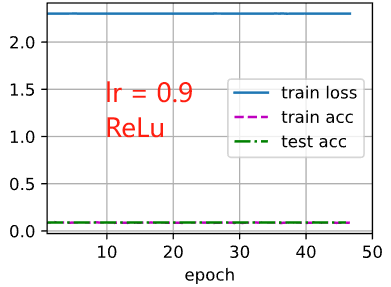
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 5 **\*** 5**,** 120)**,** nn**.**ReLU()**,**

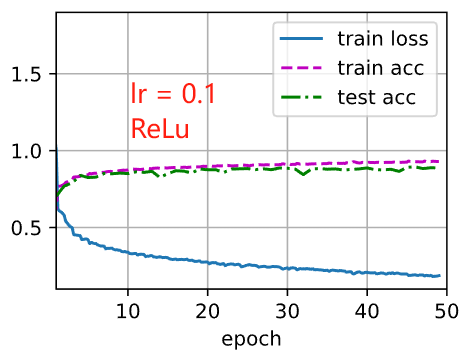
    nn**.**Linear(120**,** 84)**,** nn**.**ReLU()**,**

    nn**.**Linear(84**,** 10))

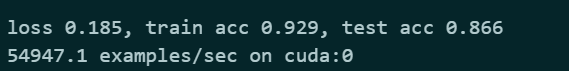
运行结果：



可见，由于ReLu将小于零的值直接归零，导致出现了明显的梯度消失现象。因此，考虑将lr减小，再次进行实验。将lr设置为0.1，运行结果为：



并且最终效果为：



可以看到，和使用sigmond函数相比，使用ReLu在较大的学习率下，有可能发生梯度消失现象。而在较小学习率下，虽然ReLu可能最终效果比sigmond好，但是相应的，过拟合风险也更高。

### 要求3

在每一层sigmond函数后面加上batch normalization，也即将网络结构更改为：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**5**,** **padding=**2)**,**

    nn**.**BatchNorm2d(6)**,**

    nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**5)**,**

    nn**.**BatchNorm2d(16)**,**

    nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 5 **\*** 5**,** 120)**,**

    nn**.**BatchNorm1d(120)**,**

    nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,**

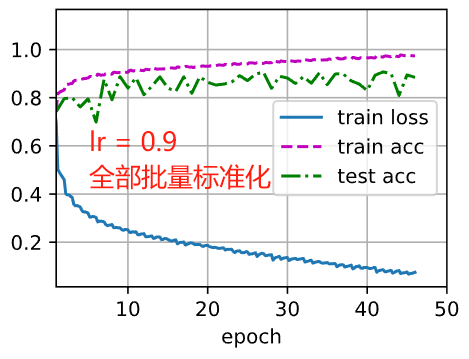
    nn**.**BatchNorm1d(84)**,**

    nn**.**Sigmoid()**,**

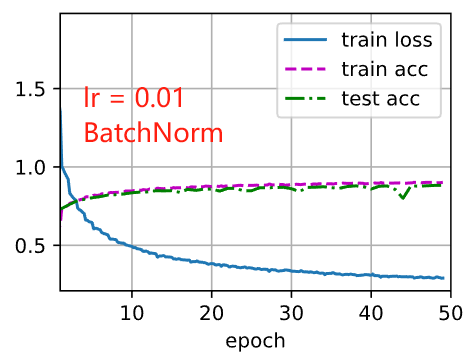
    nn**.**Linear(84**,** 10)

)

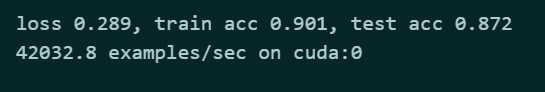
运行结果：



可见，虽然训练acc保持稳定上升，但是测试准确率却一直在波动，并且出现了过拟合现象。推测是因为lr设置过大，同时BatchNorm层设置过多导致的。因此，首先考虑将lr减小到0.01，运行结果为：



最终训练效果为：



可见，虽然较小的学习率能够有效缓和训练震荡问题，但是最终的准确率有所下降，说明陷入了局部最优解。同时也出现了一定的过拟合现象。

### 要求4

调整卷积大小，观察效果。

首先考虑将卷积层的卷积核减小，减小为3 x 3。为了使得各个结构的输入输出大小不变，因此对两个卷积层的padding核stride也有所更改：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**3**,** **padding=**1)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**3**,padding=**8**,stride=**3)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

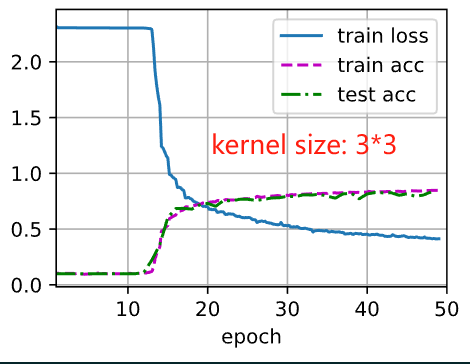
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 5 **\*** 5**,** 120)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(84**,** 10))

运行结果为：



可以看到，在换用小卷积核之后，训练效果在10epoch之后才开始明显。推测是因为为了保持各个层的输入输出不变，而在第二个卷积层上面使用了较大的填充（padding = 8），因此使得真正的特征占比变小而导致的。为此，我们再重新设置各个层，使得该卷积网络更加适配3\*3卷积核。将网络结构修改为：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**3**,** **padding=**1)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**3**,padding=**2**,stride=**3)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

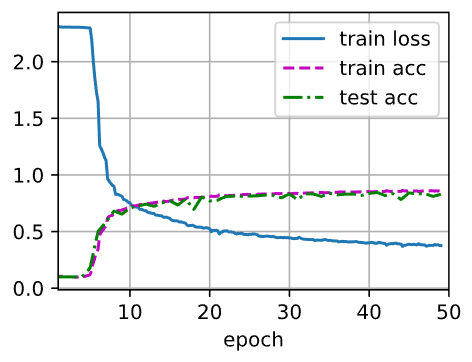
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 3 **\*** 3**,** 64)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(64**,** 32)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(32**,** 10))

运行结果为：



可见确实使得开始训练的时间往前推移，但是最终效果与5\*5卷积的准确率相差不大，并且也出现了过拟合现象。

### 要求5

改变输出通道数量。

首先考虑减少通道数量，将原有的1, 6, 16通道减少为1，2，4通道：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 2**,** **kernel\_size=**5**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(2**,** 4**,** **kernel\_size=**5)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

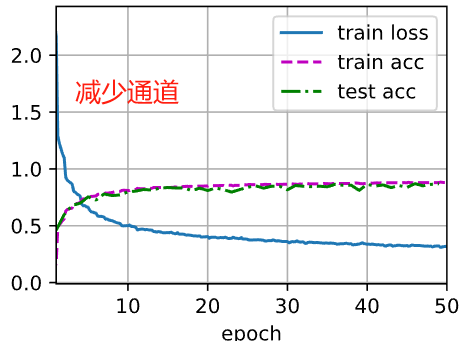
    nn**.**Flatten()**,**

    nn**.**Linear(4 **\*** 5 **\*** 5**,** 64)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(64**,** 32)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(32**,** 10))

运行结果：



再考虑增加通道数，将网络修改为：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 16**,** **kernel\_size=**5**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(16**,** 32**,** **kernel\_size=**5)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

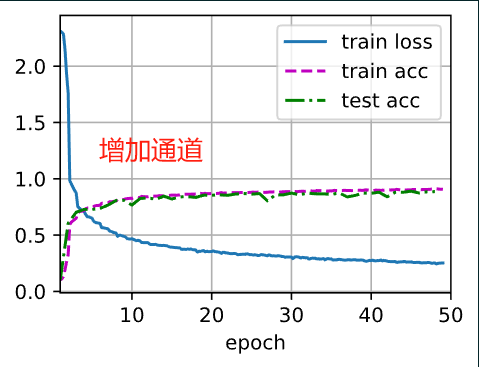
    nn**.**Flatten()**,**

    nn**.**Linear(32 **\*** 5 **\*** 5**,** 240)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(240**,** 64)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(64**,** 10))

运行结果为：



可以看到改变通道数在本实验中对于最终准确率和过拟合现象没有明显影响，但是多通道数的网络在最初的损失函数下降速度更快。

### 要求6

调整卷积层数，观察训练效果。

首先考虑减少卷积层数。将第二个卷积层（及其配套的池化层和激活层）删除，将网络结构变为：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**5**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

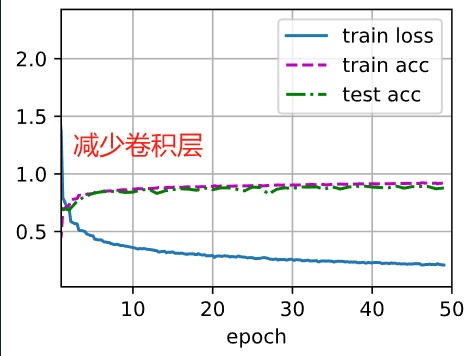
    nn**.**Flatten()**,**

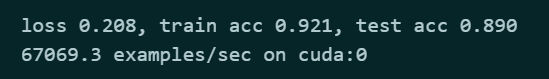
    nn**.**Linear(6 **\*** 14 **\*** 14**,** 120)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(84**,** 10))

运行结果为：





相较原本网络有略微提升。再考虑增加一个卷积层，为了适配各个层的输入输出大小，新加入的卷积层为3\*3卷积核，填充为1，步长为1的卷积层，而相应的池化层：卷积核为2\*2，步长为1，不填充。将网络结构修改为：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**5**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**5)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(16**,**16**,kernel\_size=**3**,padding=**1)**,**nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,stride=**1)**,**

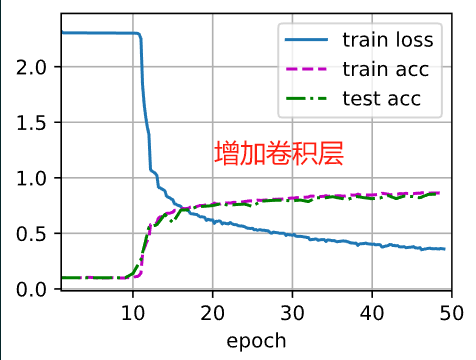
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 4 **\*** 4**,** 120)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(84**,** 10))

运行结果为：



可见，增加卷积层使得神经网络结构更加复杂，相应地，训练也更加缓慢。在本次任务中，增加卷积层使得最终准确率有略微下降。

### 要求7

将average polling替换为max polling，观察训练效果。

将网络修改为：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**5**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**MaxPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**5)**,** nn**.**Sigmoid()**,**

    nn**.**MaxPool2d(**kernel\_size=**2**,** **stride=**2)**,**

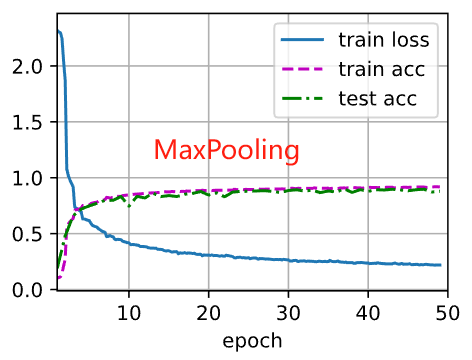
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 5 **\*** 5**,** 120)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(84**,** 10))

运行结果为：



相比于AvgPooling，准确率略有下降，但是训练速度有提升。

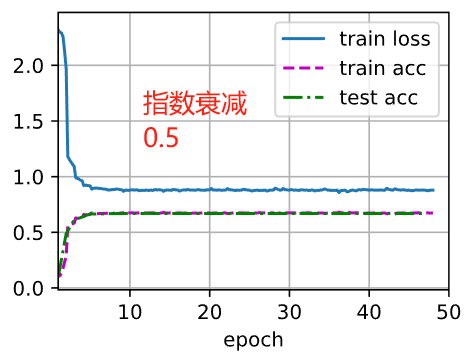
### 要求8

使用实验二中的不同优化器及学习率，效果如何？

使用指数衰减，并设置参数为0.5：

scheduler **=** lr\_scheduler**.**ExponentialLR(optimizer**,**0.5)

运行结果：



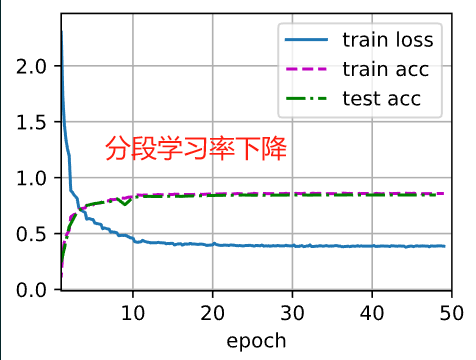
可见，学习率过早地衰减，导致训练尚未达到最优解，便已经停滞了。

使用分段学习率：

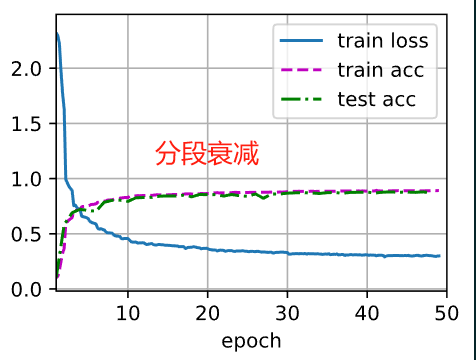
 scheduler **=** lr\_scheduler**.**StepLR(**optimizer=**optimizer**,step\_size=**10**,gamma=**0.2**,verbose=**True)

将衰减系数设置为0.2，并且每10个epoch衰减一次。

运行结果：



效果比刚刚的指数衰减要好一些，但是最终准确率仍然略有下降，应当还是学习率过早降低的结果。将衰减系数设置为0.6，再次尝试：

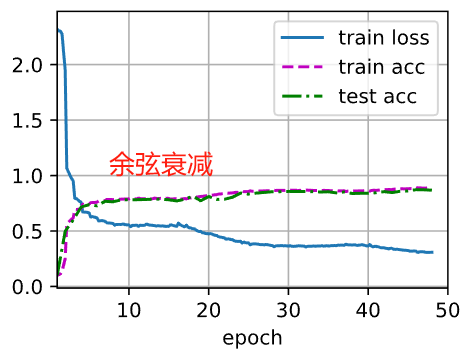


最终准确率有所提高，并且过拟合现象得到了有效抑制。

使用余弦衰减：

scheduler **=** lr\_scheduler**.**CosineAnnealingLR(**optimizer=**optimizer**,T\_max=**10**,eta\_min=**0.01)

运行结果：



可见，使用余弦衰减可以使得损失函数逐渐下降，并且训练和测试准确率也逐步上升，在本次任务中效果最佳。

## 2. 附加题

### 要求1

将LeNet的一个或多个卷积层替换成一个或多个VGG块，效果如何？

首先定义VGG块：

**def** vgg\_block(**num\_convs,** **in\_channels,** **out\_channels**)**:**

    layers **=** []

**for** \_ **in** range(num\_convs)**:**

        layers**.**append(nn**.**Conv2d(in\_channels**,** out\_channels**,**

**kernel\_size=**3**,** **padding=**1))

        layers**.**append(nn**.**ReLU())

        in\_channels **=** out\_channels

    layers**.**append(nn**.**MaxPool2d(**kernel\_size=**2**,stride=**2))

**return** nn**.**Sequential(**\***layers)

将原有网络的第一个卷积层替换为VGG块：

net **=** nn**.**Sequential(

    vgg\_block(**num\_convs=**2**,in\_channels=**1**,out\_channels=**6)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**5**,padding=**2**,stride=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

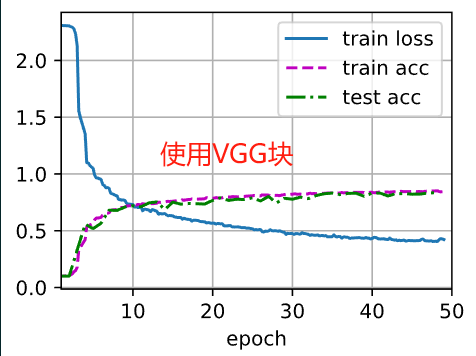
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 2 **\*** 2**,** 32)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(32**,** 16)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(16**,** 10))

运行结果为：



出现了较为明显的过拟合现象，并且最终准确率相差不大。

### 要求2

将LeNet的一个或多个卷积层替换成一个或多个NIN块，效果如何？

定义NIN块：

**def** nin\_block(**in\_channels,** **out\_channels,** **kernel\_size,** **strides,** **padding**)**:**

**return** nn**.**Sequential(

        nn**.**Conv2d(in\_channels**,** out\_channels**,** kernel\_size**,** strides**,** padding)**,**

        nn**.**ReLU()**,**

        nn**.**Conv2d(out\_channels**,** out\_channels**,** **kernel\_size=**1)**,** nn**.**ReLU()**,**

        nn**.**Conv2d(out\_channels**,** out\_channels**,** **kernel\_size=**1)**,** nn**.**ReLU())

将第一个卷积层替换为NIN块：

net **=** nn**.**Sequential(

    nin\_block(1**,** 6**,** **kernel\_size=**5**,strides=**1**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**5)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

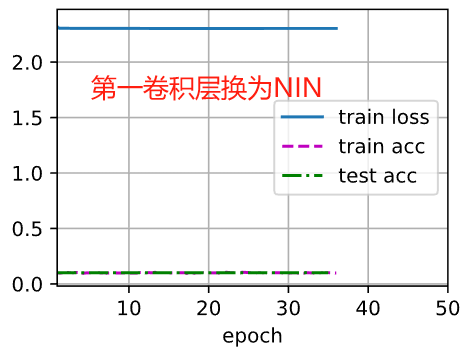
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 5 **\*** 5**,** 120)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(84**,** 10))

运行结果为：



可见，出现了梯度消失现象，导致网络无法正确学习。接下来考虑只替换第二卷积层：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**5**,stride=**1**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nin\_block(6**,** 16**,** **kernel\_size=**5**,strides=**1**,padding=**0)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

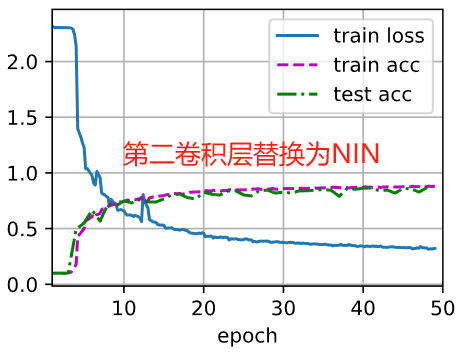
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 5 **\*** 5**,** 120)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(84**,** 10))

运行结果为：



由图可见，过拟合现象要比原来网络更明显一些。

### 要求3

将LeNet的一个或多个卷积层替换成一个或多个GoogleNet中的Inception块，效果如何？

定义inception块：

class **Inception**(*nn***.***Module*)**:**

*# c1--c4是每条路径的输出通道数*

**def** \_\_init\_\_(*self***,** **in\_channels,** **c1,** **c2,** **c3,** **c4,** **\*\*kwargs**)**:**

        super(Inception**,** *self*)**.**\_\_init\_\_(**\*\***kwargs)

*# 线路1，单1x1卷积层*

*self***.**p1\_1 **=** nn**.**Conv2d(in\_channels**,** c1**,** **kernel\_size=**1)

*# 线路2，1x1卷积层后接3x3卷积层*

*self***.**p2\_1 **=** nn**.**Conv2d(in\_channels**,** c2[0]**,** **kernel\_size=**1)

*self***.**p2\_2 **=** nn**.**Conv2d(c2[0]**,** c2[1]**,** **kernel\_size=**3**,** **padding=**1)

*# 线路3，1x1卷积层后接5x5卷积层*

*self***.**p3\_1 **=** nn**.**Conv2d(in\_channels**,** c3[0]**,** **kernel\_size=**1)

*self***.**p3\_2 **=** nn**.**Conv2d(c3[0]**,** c3[1]**,** **kernel\_size=**5**,** **padding=**2)

*# 线路4，3x3最大汇聚层后接1x1卷积层*

*self***.**p4\_1 **=** nn**.**MaxPool2d(**kernel\_size=**3**,** **stride=**1**,** **padding=**1)

*self***.**p4\_2 **=** nn**.**Conv2d(in\_channels**,** c4**,** **kernel\_size=**1)

**def** forward(*self***,** **x**)**:**

        p1 **=** F**.**relu(*self***.**p1\_1(x))

        p2 **=** F**.**relu(*self***.**p2\_2(F**.**relu(*self***.**p2\_1(x))))

        p3 **=** F**.**relu(*self***.**p3\_2(F**.**relu(*self***.**p3\_1(x))))

        p4 **=** F**.**relu(*self***.**p4\_2(*self***.**p4\_1(x)))

*# 在通道维度上连结输出*

**return** torch**.**cat((p1**,** p2**,** p3**,** p4)**,** **dim=**1)

并且把第二个卷积层替换为inception块：

net **=** nn**.**Sequential(

    nn**.**Conv2d(1**,** 6**,** **kernel\_size=**5**,** **padding=**2)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

    nn**.**Conv2d(6**,** 16**,** **kernel\_size=**5)**,** nn**.**Sigmoid()**,**

    nn**.**AvgPool2d(**kernel\_size=**2**,** **stride=**2)**,**

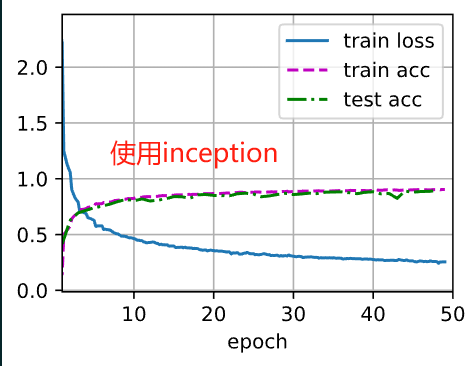
    nn**.**Flatten()**,**

    nn**.**Linear(16 **\*** 5 **\*** 5**,** 120)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(120**,** 84)**,** nn**.**Sigmoid()**,**

    nn**.**Linear(84**,** 10))

运行结果为：



可见，训练效果较为平稳，并且过拟合现象也不太严重。

### 要求4

实现一个小号的ResNet

定义卷积块和卷积层之后（见代码文件），搭建ResNet网络：

b1 **=** nn**.**Sequential(nn**.**Conv2d(1**,** 64**,** **kernel\_size=**7**,** **stride=**2**,** **padding=**3)**,**

                   nn**.**BatchNorm2d(64)**,** nn**.**ReLU()**,**

                   nn**.**MaxPool2d(**kernel\_size=**3**,** **stride=**2**,** **padding=**1))

b2 **=** nn**.**Sequential(**\***resnet\_block(64**,** 64**,** 2**,** **first\_block=**True))

b3 **=** nn**.**Sequential(**\***resnet\_block(64**,** 128**,** 2))

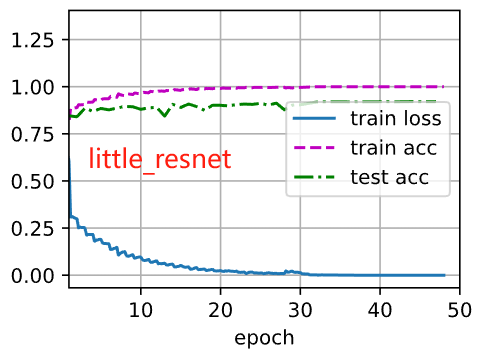
b5 **=** nn**.**Sequential(**\***resnet\_block(128**,** 512**,** 2))

little\_resnet **=** nn**.**Sequential(b1**,** b2**,** b3**,** b5**,**

                    nn**.**AdaptiveAvgPool2d((1**,**1))**,**

                    nn**.**Flatten()**,** nn**.**Linear(512**,** 10))

运行结果：



可见，ResNet对于训练集的准确度有非常明显的提升，但是在本次的ResNet网络中，仍然出现了较明显的过拟合现象。

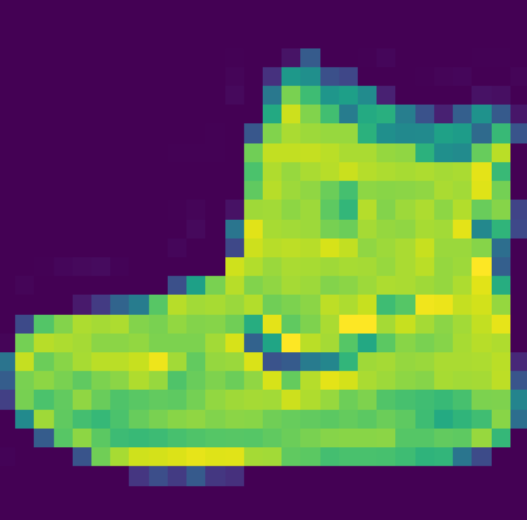
### 要求5

将LeNet隐藏层提取的特征转换成像素形式显示，尝试解释深度学习的工作原理

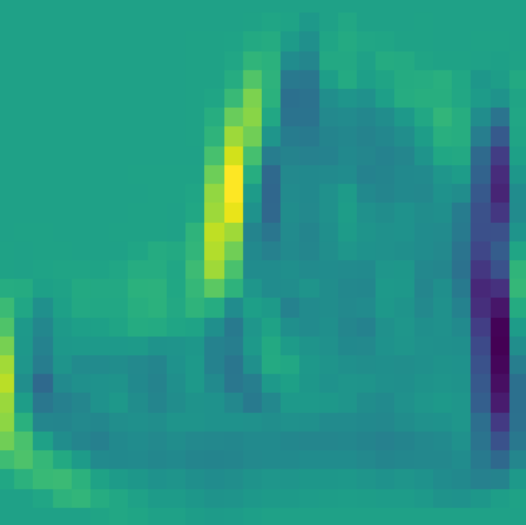
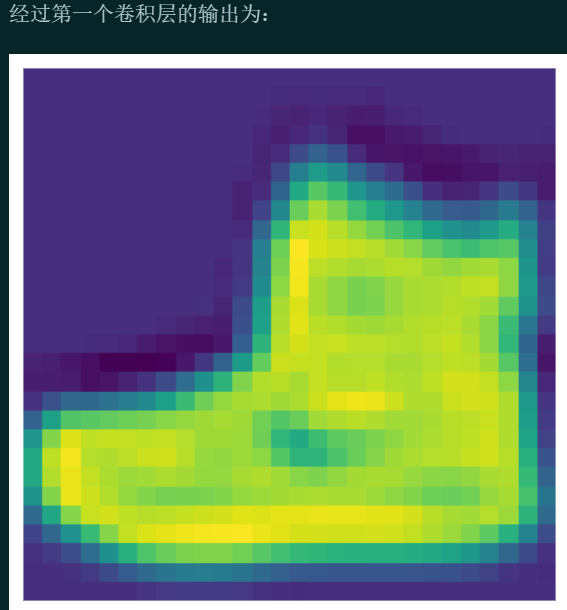
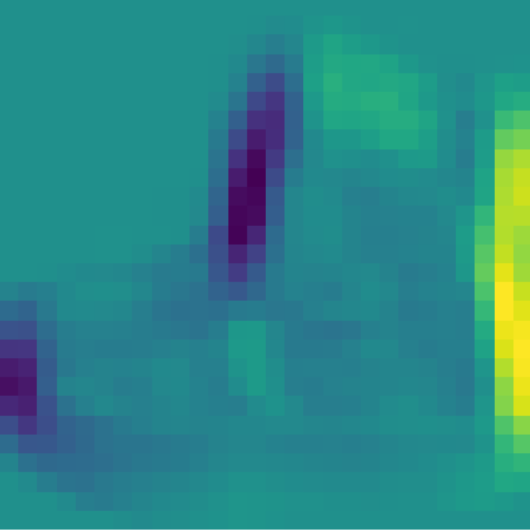
考虑查看前六层的输出。（因为之后是将图像展平，特征不太明显）

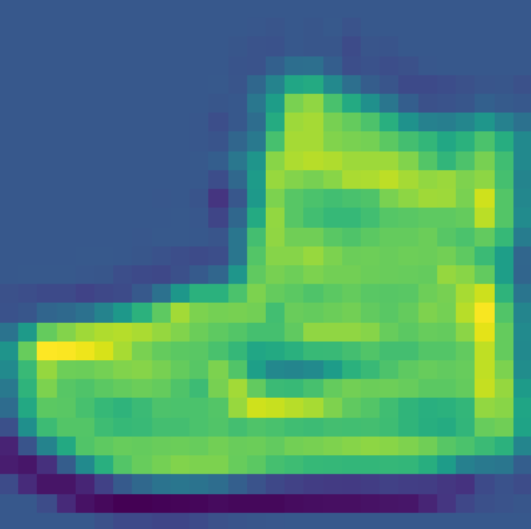
编写查看中间输出的函数并进行特征查看（代码请见代码文件）：

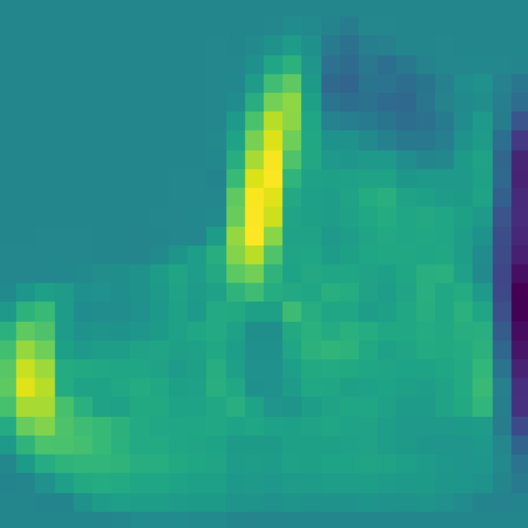
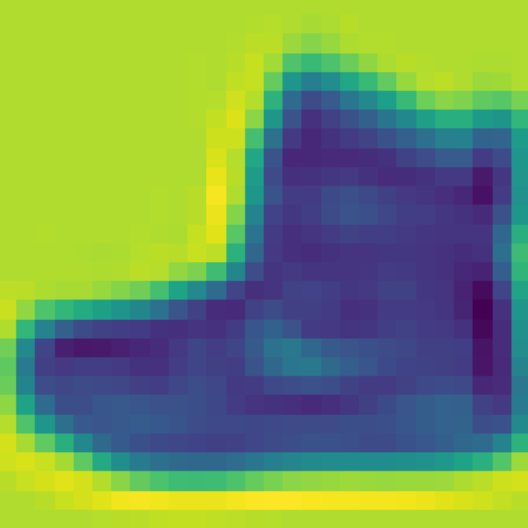
以一张图片为例，选取dataset中第一张图片（索引为0），其原图为：



查看经过第一个卷积层的输出（6通道）：

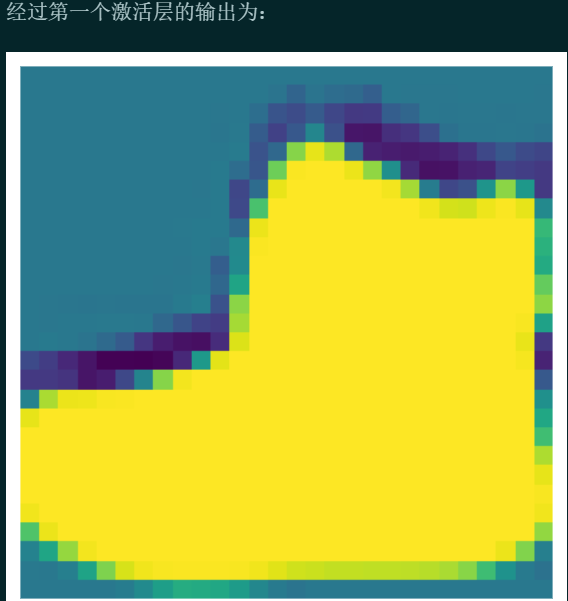
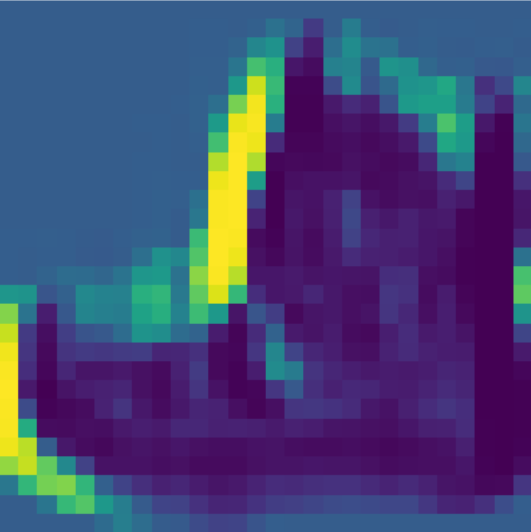
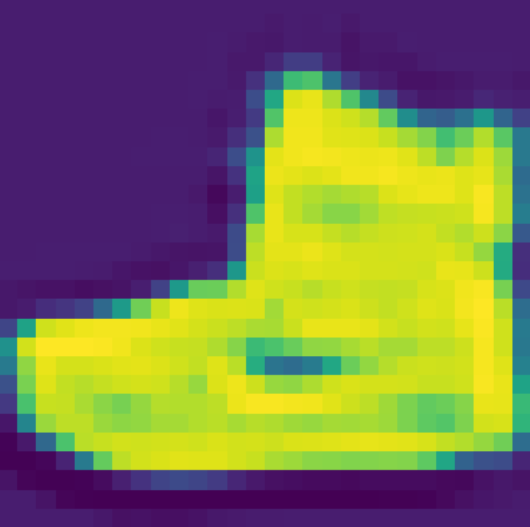
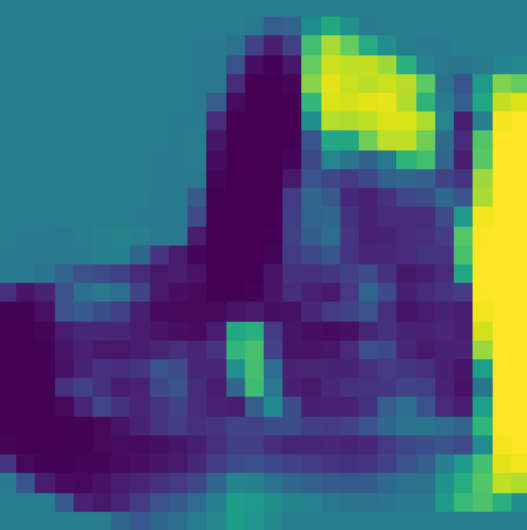


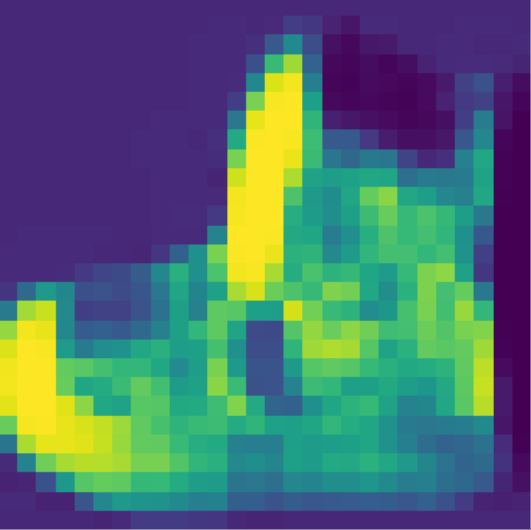
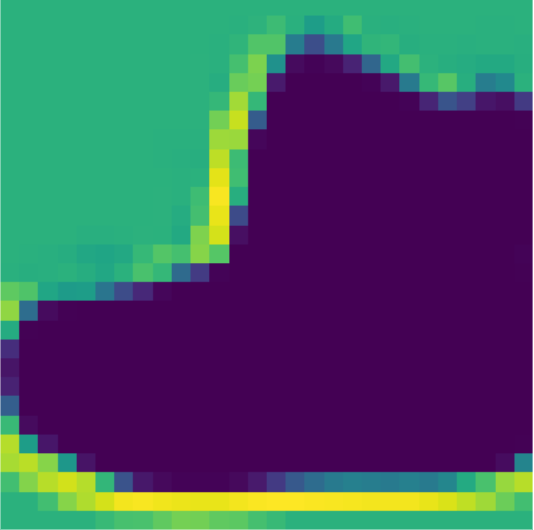




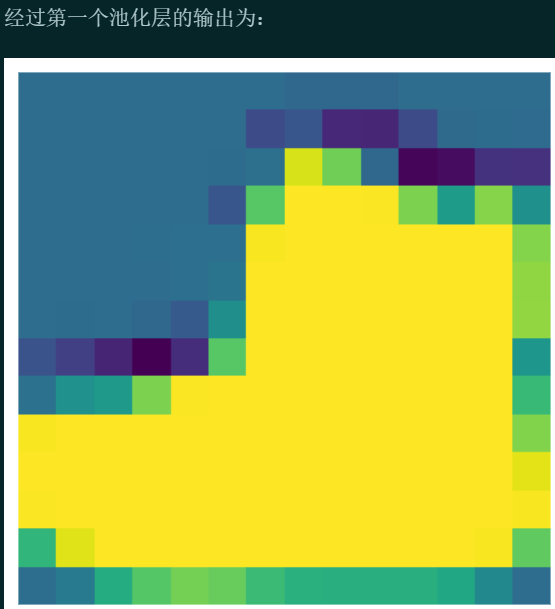
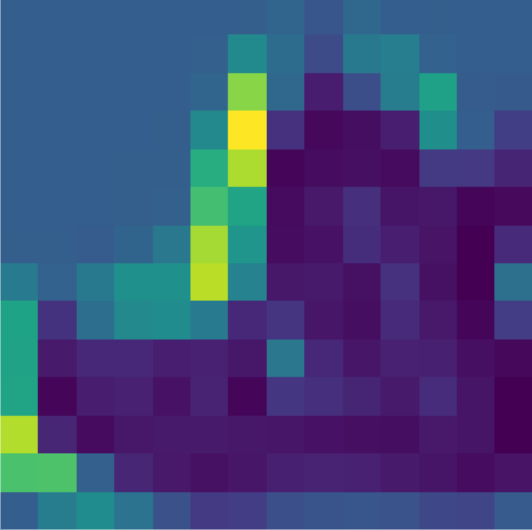
可见，经过一个卷积层之后，可以从不同维度让原图像的特征变得更加明显。

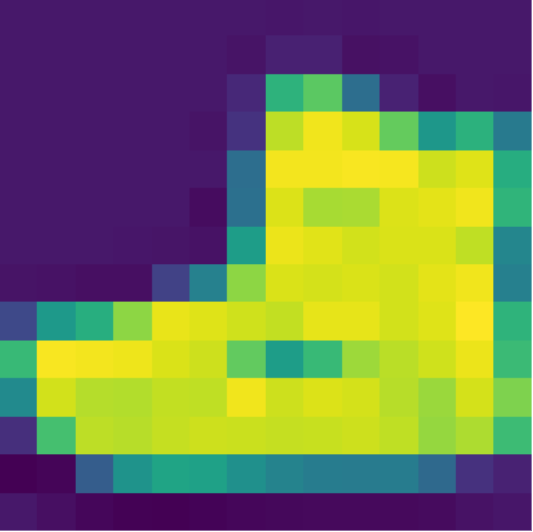
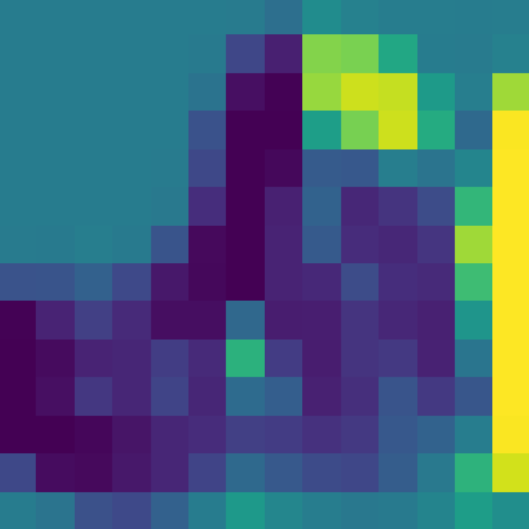
查看经过第一个激活层的输出（6通道）：

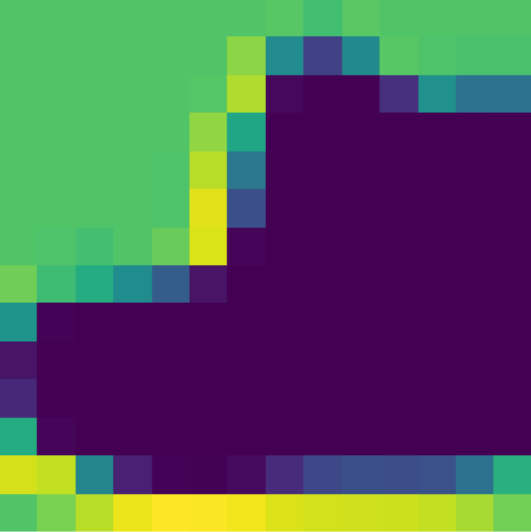


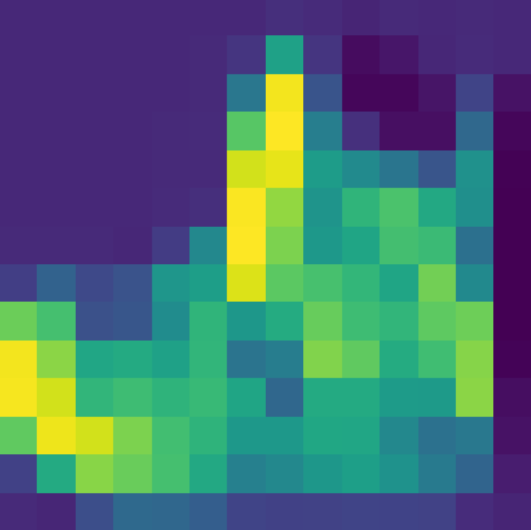


可见，激活层使得卷积层提取出来的特征变得更加明显，从直观来看，即是图像的对比度更加明显。

查看第一个池化层的输出：

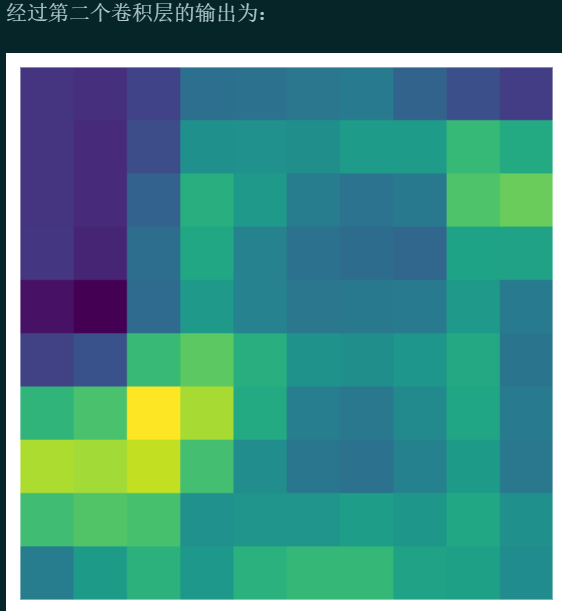
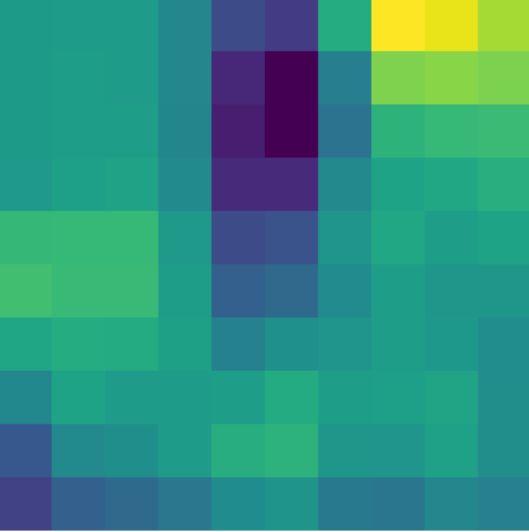
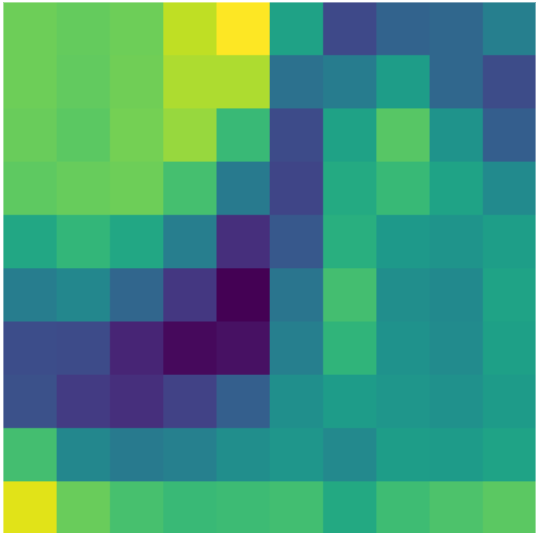
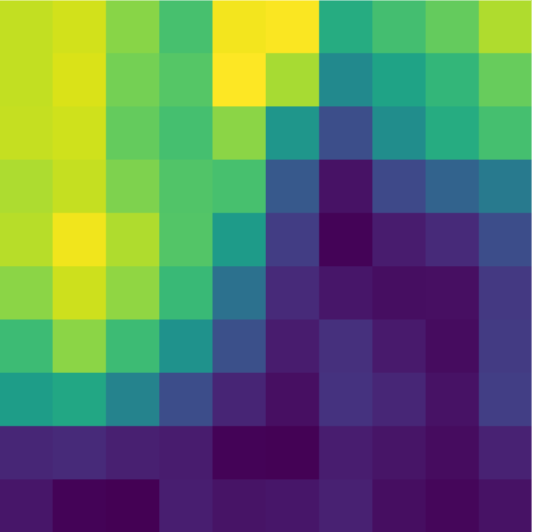


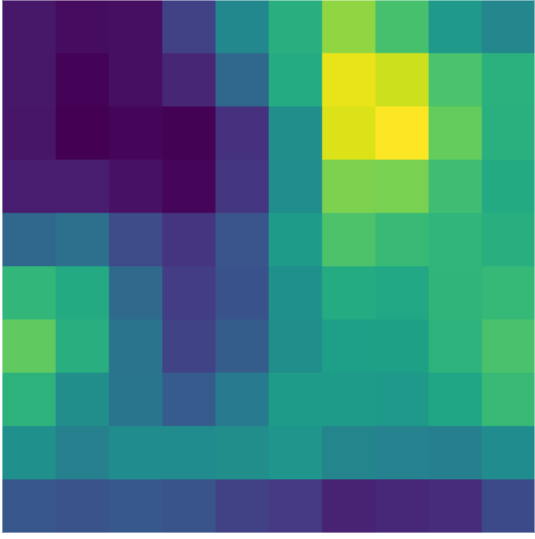
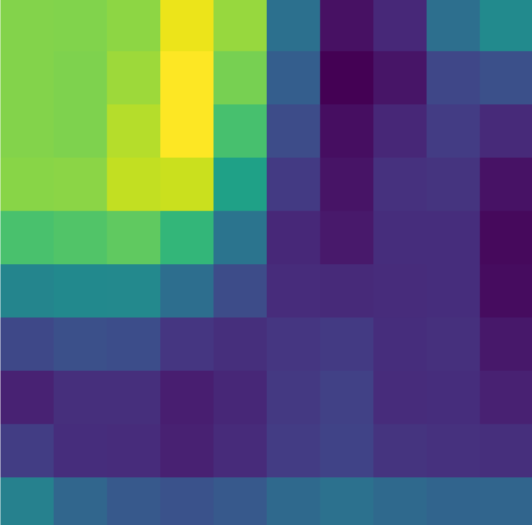


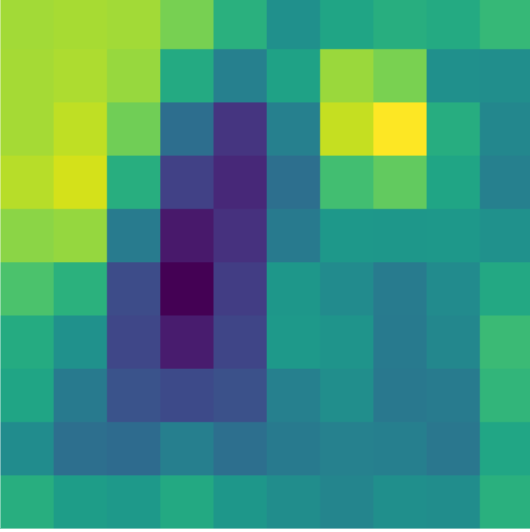
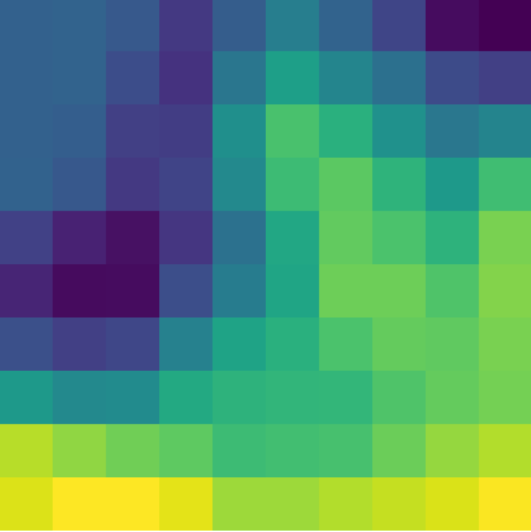
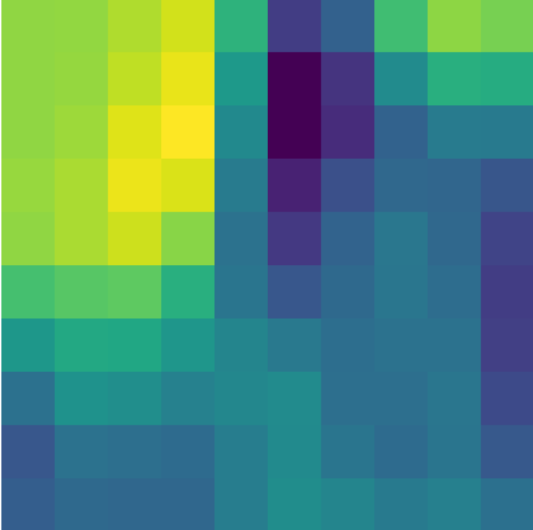
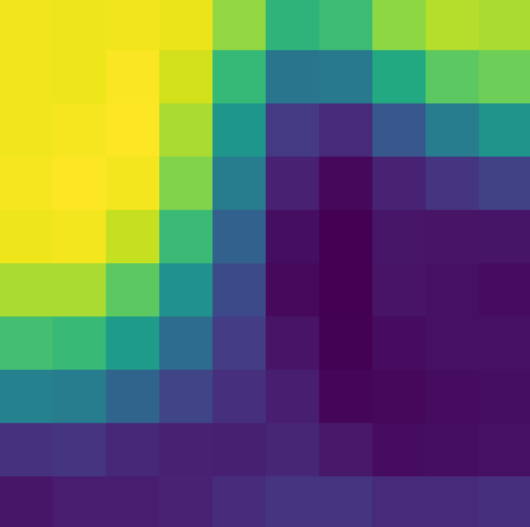


可见，池化层将图像的大小减半，因而使得信息量进一步减少，使得主要特征信息的占比上升。

查看经过第二个卷积层的输出（以前10个通道为例）：

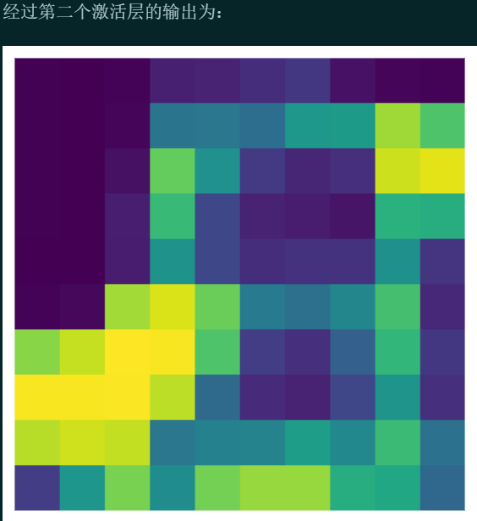
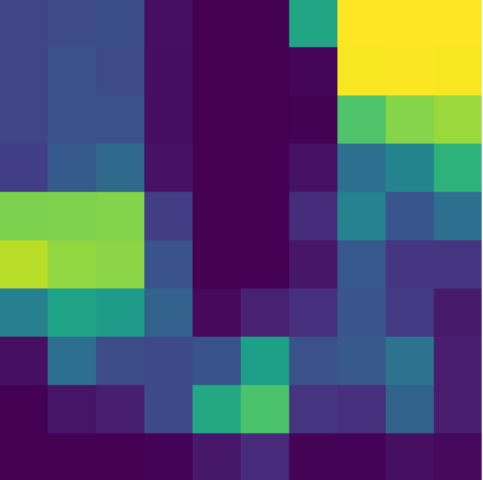


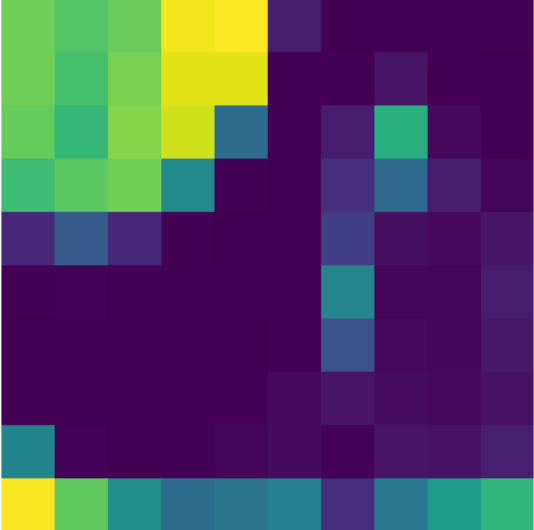
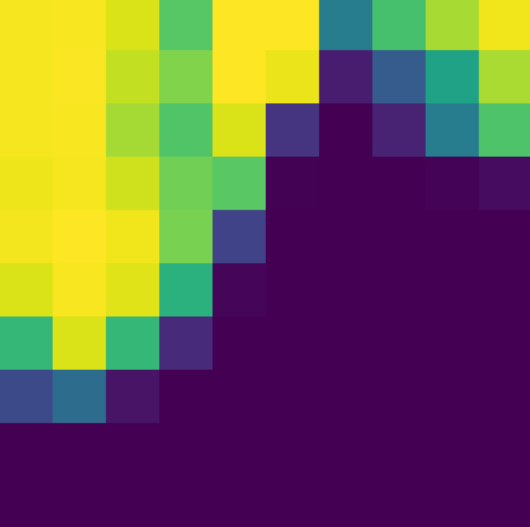




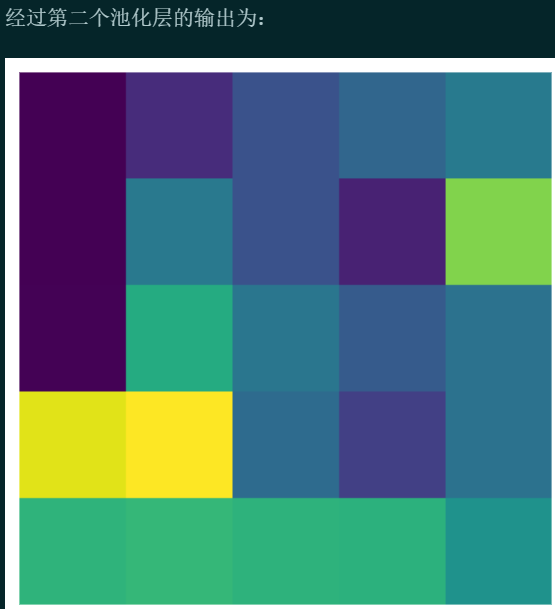
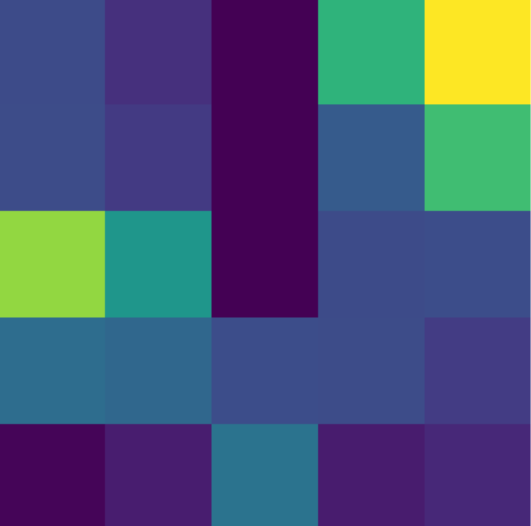
可见，经过第二个卷积层之后，在第一次提取的特征基础上，再次提取到新的特征，并将其扩充到了16通道。

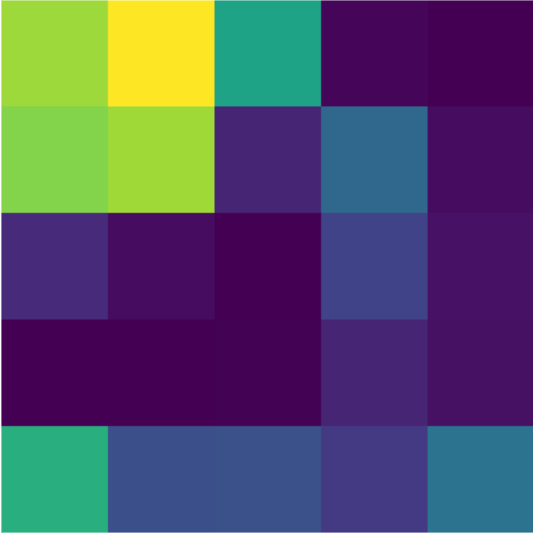
查看第二个激活层的输出（以前四个通道为例）：





可见，通过第二个激活层之后，提取出的特征更加明显，输出图像的对比度也进一步增加。。

查看第二个池化层的输出（以前四个通道为例）：



可见，经过第二个池化层，各个特征特征变为5\*5大小，也更加泛化。

综上可以看出，经过两次的卷积-激活-池化，可以将原本图片的特征进行扩充、增强、压缩，从而提取出更具有泛化能力的特征。