实验 4、实验 5：使用卷积神经网络识别手写数字

一、　实验内容

实验内容：用PyTorch 来实现一个卷积神经网络，从而实现手写数字识别任务。除此之外，还对卷积神经网络的卷积核、特征图等进行了分析，引出了过滤器的概念，并简单示了卷积神经网络的工作原理。

实验环境：Python-3.11;PyTorch-cu121;CUDA-12.1;VS code/JupyterNotebook

二、　实验过程和结果

（一）实验数据准备

1. 引入相关包

```
# 使用卷积神经网络识别手写数字
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.optim as optim
import torch.nn.functional as F
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
```

2. 定义超参数

```
# 定义超参数
image_size=28
num_classes=10
num_epochs=20
batch_size=64
```

✓ 0.0s

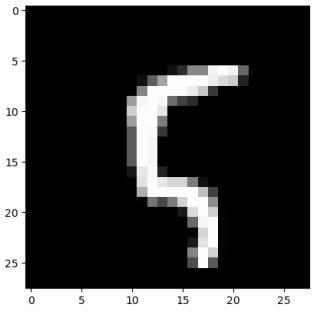
数据为 28×28 的灰度图像，输出类别 0~9，训练轮次 20，每撮大小 64

3. 使用 PyTorch 数据加载三套件

```
# 1数据集处理
# !wget http://labfile.oss.aliyuncs.com/course/1073/MNIST/data.zip
# !unzip data.zip
train_dataset=dsets.MNIST(root='./data',train=True,transform=transforms.ToTensor(),download=True)
test_dataset=dsets.MNIST(root='./data',train=False,transform=transforms.ToTensor())
train_loader=torch.utils.data.DataLoader(
    dataset=train_dataset,
    batch_size=batch_size,
    shuffle=True
    )
```

✓ 0.0s

首先下载手写数字识别数据集，使用 Pytorch 数据加载器加载数据集，将数据集分为训练集和测试集，使用 DataLoader 对训练集进行分割。将测试集分割成前 5000 条验证集和后面的测试集，使用 Sampler 对集合进行随机采样方法的构建，再按照和训练集相同的方法将校验集和验证数据集分割成多个撮。输出训练集中的 100 号数据和它的标签。



```
indices=range(len(test_dataset))
indices_val=indices[:5000]
indices_test=indices[5000:]
sampler_val=torch.utils.data.SubsetRandomSampler(indices_val)
sampler_test=torch.utils.data.SubsetRandomSampler(indices_test)

# 校验数据集的加载器
validation_loader=torch.utils.data.DataLoader(
    dataset=test_dataset,
    batch_size=batch_size,
    sampler=sampler_val
)
# 验证数据集的加载器
test_loader=torch.utils.data.DataLoader(
    dataset=test_dataset,
    batch_size=batch_size,
    sampler=sampler_test
)

idx=100
muteimg=train_dataset[idx][0].numpy()
plt.imshow(muteimg[0,...],cmap='gray')
print('标签是',train_dataset[idx][1])
```

✓ 0.3s

（二）基本的卷积神经网络

1. 构建网络

```
# 2构建基本的卷积神经网络
# 定义类
depth=[4,8]
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet,self).__init__()
        # 卷积层1，1输入，4输出，5*5卷积核，padding大小为2保持输出大小
        self.conv1=nn.Conv2d(1,4,5,padding=2)
        # 池化层，2*2池化窗口
        self.pool=nn.MaxPool2d(2,2)
        # 卷积层2，4输入，8输出，5*5卷积核，padding大小为2保持输出大小
        self.conv2=nn.Conv2d(depth[0],depth[1],5,padding=2)
        # 线性层，输入为最后一层卷积后大小，输出512
        self.fc1=nn.Linear(image_size//4*image_size//4*depth[1],512)
        # 线性层，输入512，输出10个符号的概率
        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)

        #将张量铺成一维
        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)#全连接
        x=F.log_softmax(x,dim=1)
        return x

    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
```

```
#实例化
net=ConvNet()
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(net.parameters(),lr=0.001,momentum=0.9)
```
✓ 0.0s

构建卷积神经网络需要使用定义类的方法，父类为 nn.Module。depth 的两个参数定义了两层卷积层卷积核的个数，重写__init__方法，在实例化类的时候会被自动调用，其中包括 RNN 类的各个层的结构。卷积层 1 输入通道为 1，输出通道为 4，对应四个卷积核，卷积核的大小为 5×5，padding 的设置与卷积核的大小有关，使图像经过二维卷积处理后大小不变。池化层不需要可学习的参数，且两个池化层的处理方法相同，只需定义一个池化窗口为 2×2 的池化层后面调用两次。卷积层 2 的输入通道数为卷积层 1 的输入通道数，输出通道数为 8，因此需要 8 个 5×5 的卷积核，同样 padding 大小为 2，随后连接两个前馈全连接神经网络，第一层前馈网络输入是将第二层池化后的结果压缩成一维向量后的输出，单层图像经过卷积层大小不变，经过 2×2 的池化窗长宽减半，所以最终池化的结果为 28/4×28/4 共八层的三阶张量，输出为 512 个神经元。第二层前馈网络输入为 512，输出为 10 个数字类别。

重写 forward 方法，forward 会在神经网络正向计算时被自动调用。按神经网络顺序连接各层，view 方法和 reshape 方法一样，将张量重排成一列，经过全连接层和 relu 函数激活，再经过全连接层和 softmax 函数对输出归一化为概率后输出。其中的 dropout 函数在训练过程中随机丢弃一些神经元不进行训练，可以有效防止过拟合。

retriev_features 方法用于提取神经网络产生的特征图，其中只完成了第一层卷积、一层池化和第二层卷积，用于输出训练后卷积输出的结果。

然后定义网络将卷积神经网络实例化，定义损失函数为交叉熵损失函数，优化算法为随机梯度下降算法。

2. 训练卷积神经网络

定义函数求解正确个数，定义函数训练网络，定义函数使用校验数据集验证网络有效性，其中 train 模式开启 dropout 防止训练出现过拟合，eval 模式关闭 dropout 对网络进行使用和验证。

然后正式对网络开始训练，共 20 个轮次在每一个训练轮次中对训练集使用迭代器，按撮将数据输入网路训练得到每一撮的正确率和损失值，追加到本轮正确率的数组中，每隔 100 撮统计一次正确率并输出，将准确率保存到全局变量数组中用于曲线图绘制，并将网络权重参数使用 clone 方法保存。

```python
def rightness(predictions,labels):
    pred=torch.max(predictions.data,1)[1]
    rights=pred.eq(labels.data.view_as(pred)).sum()
    return rights,len(labels)

def train_model(data,target):
    net.train()
    output=net(data)
    loss=criterion(output,target)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    right=rightness(output,target)
    return right,loss

def evaluation_model():
    net.eval()
    val_rights=[]
    for (data,target)in validation_loader:
        data,target=Variable(data),Variable(target)
        output=net(data)
        right=rightness(output,target)
        val_rights.append(right)

    return val_rights
```

✓ 0.0s

```python
record=[]
weights=[]

for epoch in range(num_epochs):
    train_rights=[]
    for batch_idx,(data,target) in enumerate(train_loader):
        data,target=Variable(data),Variable(target)
        right,loss=train_model(data,target)
        train_rights.append(right)
        if batch_idx%100==0:
            val_rights=evaluation_model()
            val_r=(sum([tup[0] for tup in val_rights]),sum([tup[1] for tup in val_rights]))
            train_r=(sum([tup[0] for tup in train_rights]),sum([tup[1] for tup in train_rights]))

            print('训练周期: {}[{}/{}({:.0f}%)]\tLoss:{:.6f}\t训练正确率:{:.2f}%\t校验正确率:{:.2f}%'.format(
                epoch,batch_idx*batch_size,len(train_loader.dataset),
                100.*batch_idx/len(train_loader.dataset),loss.data,
                100.*train_r[0].numpy()/train_r[1],
                100.*val_r[0].numpy()/val_r[1]))

            record.append((100-100.*train_r[0]/train_r[1],100-100.*val_r[0]/val_r[1]))

            weights.append(
                [net.conv1.weight.data.clone(),net.conv1.bias.data.clone,
                 net.conv2.weight.data.clone(),net.conv2.bias.data.clone])
```

✓ 4m 30.6s

```
训练周期: 0[51200/60000(1%)]     Loss:1.155061    训练正确率:26.58%        校验正确率:71.58%
训练周期: 0[57600/60000(2%)]     Loss:0.714355    训练正确率:31.84%        校验正确率:79.98%
训练周期: 1[0/60000(0%)] Loss:0.514713    训练正确率:84.38%        校验正确率:82.26%
训练周期: 1[6400/60000(0%)]      Loss:0.665957    训练正确率:82.33%        校验正确率:85.06%
训练周期: 1[12800/60000(0%)]     Loss:0.404375    训练正确率:83.83%        校验正确率:86.20%
训练周期: 1[19200/60000(0%)]     Loss:0.472677    训练正确率:84.72%        校验正确率:88.28%
训练周期: 1[25600/60000(1%)]     Loss:0.279011    训练正确率:85.70%        校验正确率:89.40%
训练周期: 1[32000/60000(1%)]     Loss:0.290153    训练正确率:86.33%        校验正确率:89.24%
训练周期: 1[38400/60000(1%)]     Loss:0.287751    训练正确率:86.96%        校验正确率:90.66%
训练周期: 1[44800/60000(1%)]     Loss:0.380980    训练正确率:87.46%        校验正确率:91.46%
训练周期: 1[51200/60000(1%)]     Loss:0.143987    训练正确率:87.89%        校验正确率:91.76%
训练周期: 1[57600/60000(2%)]     Loss:0.229992    训练正确率:88.29%        校验正确率:91.58%
训练周期: 2[0/60000(0%)] Loss:0.141314    训练正确率:95.31%        校验正确率:90.78%
训练周期: 2[6400/60000(0%)]      Loss:0.225468    训练正确率:92.73%        校验正确率:91.90%
训练周期: 2[12800/60000(0%)]     Loss:0.186487    训练正确率:92.43%        校验正确率:92.94%
训练周期: 2[19200/60000(0%)]     Loss:0.139839    训练正确率:92.60%        校验正确率:93.02%
训练周期: 2[25600/60000(1%)]     Loss:0.148689    训练正确率:92.69%        校验正确率:93.22%
...
训练周期: 19[38400/60000(1%)]    Loss:0.070758    训练正确率:98.06%        校验正确率:97.56%
训练周期: 19[44800/60000(1%)]    Loss:0.015198    训练正确率:98.06%        校验正确率:97.50%
训练周期: 19[51200/60000(1%)]    Loss:0.166006    训练正确率:98.07%        校验正确率:97.72%
训练周期: 19[57600/60000(2%)]    Loss:0.067254    训练正确率:98.08%        校验正确率:97.78%
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

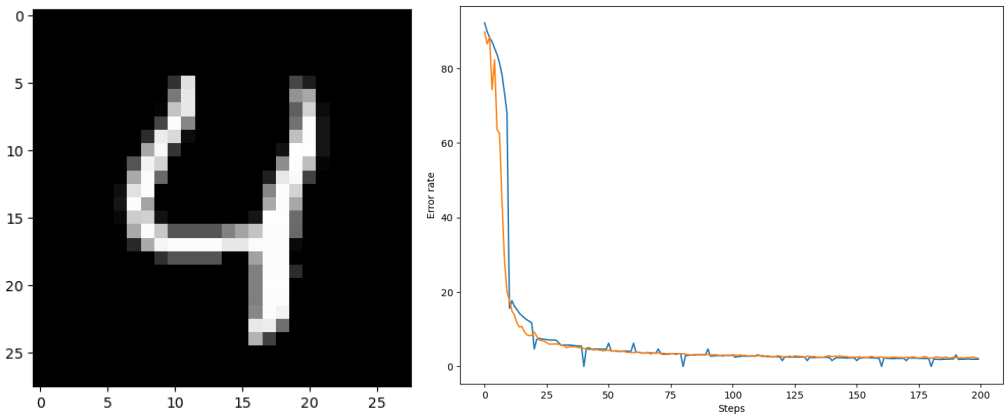
3. 观察并验证模型训练效果

```
plt.figure(figsize=(10,7))
plt.plot(record)
plt.xlabel('Steps')
plt.ylabel('Error rate')
```
✓ 0.0s

```
net.eval()
vals=[]
for data,target in test_loader:
    data,target=Variable(data,requires_grad=True),Variable(target)
    output=net(data)
    val=rightness(output,target)
    vals.append(val)

rights=(sum([tup[0] for tup in vals]),sum([tup[1] for tup in vals]))
rights_rate=1.0*rights[0].data.numpy()/rights[1]
rights_rate
```
✓ 0.3s

```
0.9906
```

首先根据训练中保存的错误率数据绘制训练的错误率变化曲线。然后使用测试集数据，输入神经网络获得识别结果与数据标签进行比对得到测试正确率。输出测试集中 index 为 4 的数据图像，输入到网络得到的预测结果，并输出其标签正确结果，两者相同。



```
idx=4
muteimg=test_dataset[idx][0].numpy()
plt.imshow(muteimg[0,...],cmap='gray')
print('正确的标签是：',test_dataset[idx][1])

test_input=torch.Tensor(muteimg).view(1,1,28,28)
out =net(Variable(test_input))
print('模型预测结果是：',torch.max(out,1)[1].data.numpy())
```

✓ 0.0s

```
正确的标签是：  4
模型预测结果是：  [4]
```

经过校验集验证正确率达到 99.06%

（三）解剖卷积神经网络

1. 第一层卷积核训练得到了什么

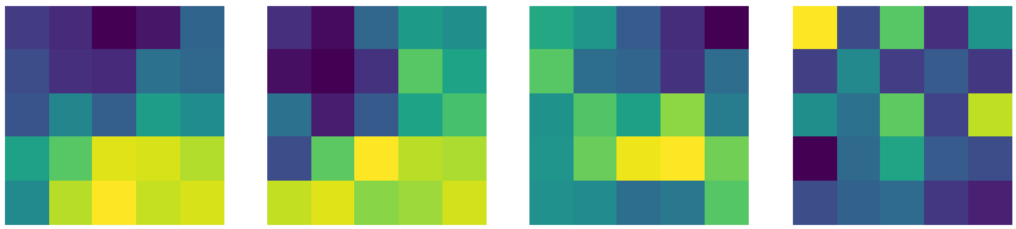
```
net.parameters
```

✓ 0.0s

```
<bound method Module.parameters of ConvNet(
  (conv1): Conv2d(1, 4, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(4, 8, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (fc1): Linear(in_features=392, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=10, bias=True)
)>
```

```
plt.figure(figsize=(10,7))
for i in range(4):
    plt.subplot(1,4,i+1)
    plt.axis('off')
    plt.imshow(net.conv1.weight.data.numpy()[i,0,...])
```

✓ 0.0s



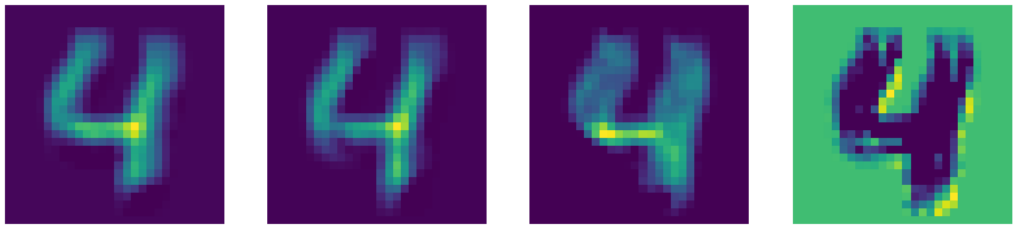
通过 net.parameters 输出网络各层的大小和可训练参数的大小。将第一层卷积层的四个卷积核以图像的方式输出，每个卷积核为 5×5 大小的二维张量。

2. 观察第一层卷积核所对应的 4 个特征图

```
idx=4
input_x=test_dataset[idx][0].unsqueeze(0)
feature_maps=net.retrieve_features(Variable(input_x))
plt.figure(figsize=(10,7))
for i in range(4):
    plt.subplot(1,4,i+1)
    plt.axis('off')
    plt.imshow(feature_maps[0][0,i,...].data.numpy())
```

✓ 0.0s

将测试集 index 为 4 的数据图像作为输入，用先前定义的 feature_maps 函数返回图像经过第一层卷积层 4 个卷积核处理后的的结果。可以看出来经过卷积核处理图像的边缘信息被提取出来，边缘加强。

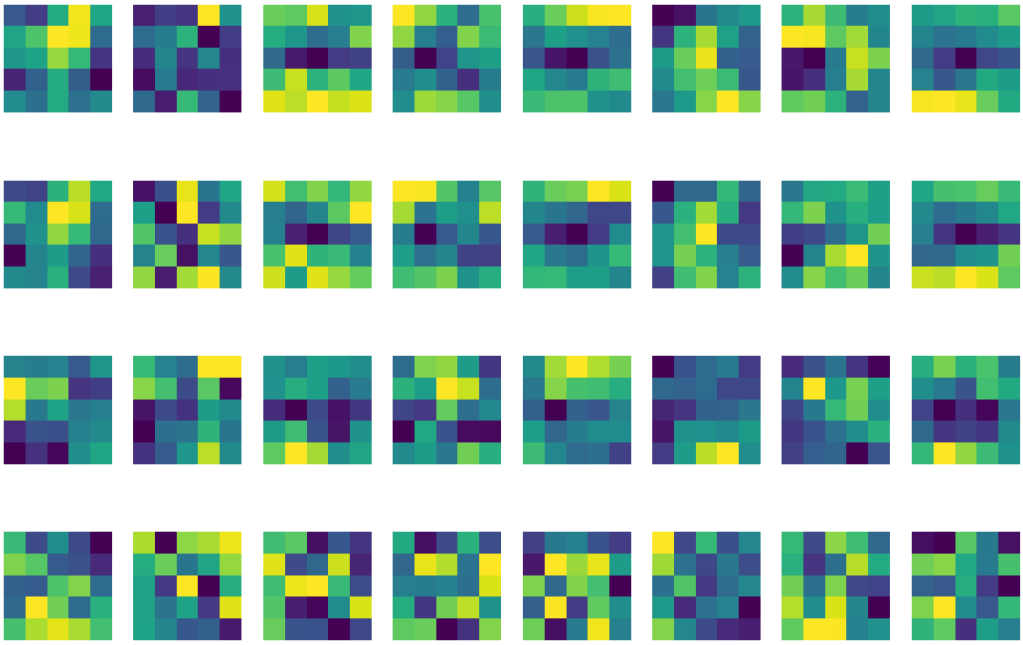


3. 观察第二层卷积的卷积核

```
plt.figure(figsize=(15,10))
for i in range(4):
    for j in range(8):
        plt.subplot(4,8,i*8+j+1)
        plt.axis('off')
        plt.imshow(net.conv2.weight.data.numpy()[j,i,...])
```

✓ 0.3s

与第一层卷积核不同，第二层输入为 14×14×4 的三阶张量，因此第二层的每一个卷积核都有四层，共 8 个，将其按层展开得到下图。

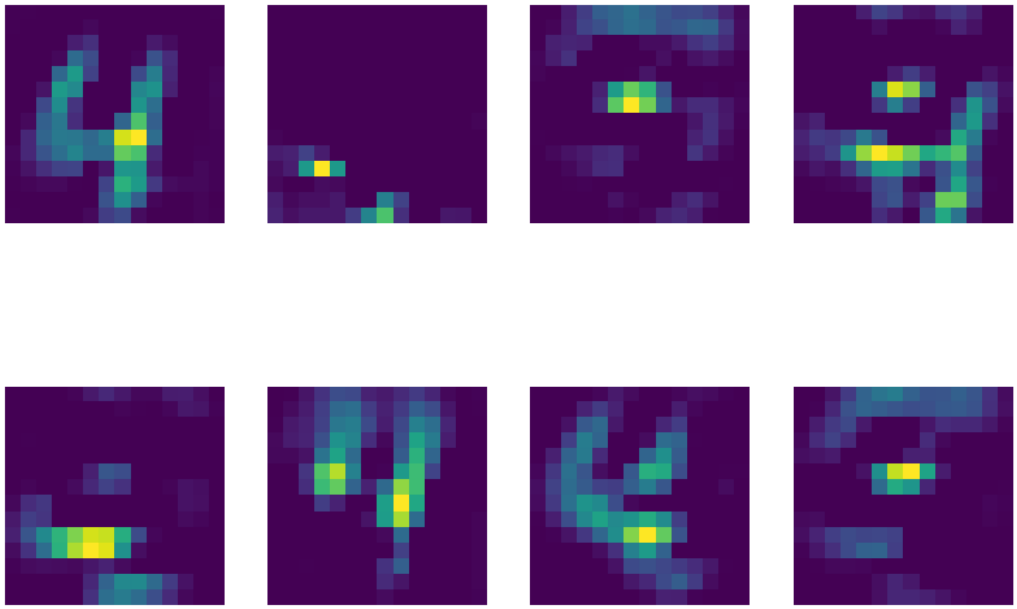


4. 第二层卷积核都是什么东西

```
plt.figure(figsize=(10,7))
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.axis('off')
    plt.imshow(feature_maps[1][0,i,...].data.numpy())
```

✓ 0.0s

经过第二层卷积层处理后大多数图像已经几乎看不出原图的大体特征，说明经过池化和二层卷积后，图像更加粗粒度的信息和抽象的特征被网络提取出来了



5. 卷积神经网络的鲁棒性试验

```
a = test_dataset[idx][0][0]
b = torch.zeros(a.size())
w=3
for i in range(a.size()[0]):
    for j in range(0, a.size()[1]-w):
        b[i,j] = a[i,j+w]

muteimg = b.numpy()
plt.axis('off')
plt.imshow(muteimg)
prediction = net(Variable(b.unsqueeze(0).unsqueeze(0)))
pred = torch.max(prediction.data, 1)[1]
print('预测结果: ', pred)
feature_maps=net.retrieve_features(Variable(b.unsqueeze(0).unsqueeze(0)))
```

✓ 0.0s

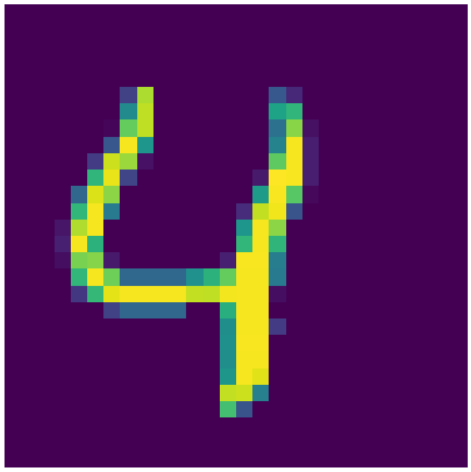
```
预测结果:  tensor([4])
```

```
plt.figure(figsize =(10,7))
for i in range(4):
    plt.subplot(1,4,i+1)
    plt.axis('off')
    plt.imshow(feature_maps[0][0,i,...].data.numpy())
```

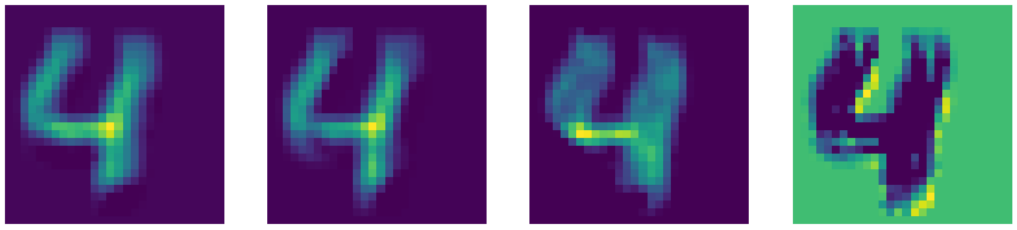
✓ 0.0s

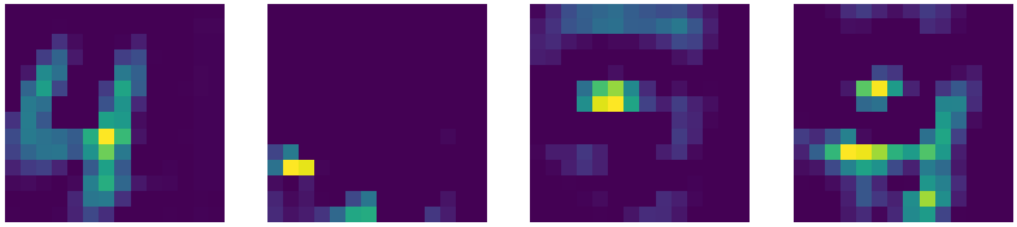
```
plt.figure(figsize =(10,7))
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.axis('off')

    plt.imshow(feature_maps[1][0,i,...].data.numpy())
```

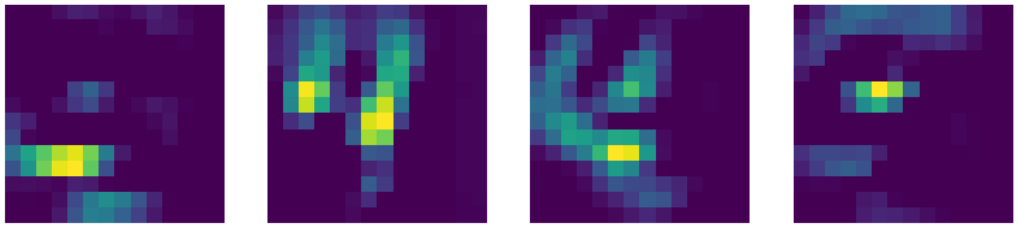
✓ 0.0s



鲁棒性测试就是使系统受到干扰测试其结果的稳定性，实验中施加的干扰为将输入图像平移，对图像数组进行平移操作，然后再输入到网络中，观察其预测结果，依然正确。参照之前的方法提取网络计算过程中的特征图。对比没有平移前的特征图，差别不大。







三、　心得体会

本次实验练习了如何搭建卷积神经网络，首次采用了定义类的方法，我认为改方法的优点是像搭积木一样采用了 Pytorch 中已定义好的各类网络层，将其作为类的属性初始化，再在 forwar 函数中将其按顺序连接。这种方法易读，参数可调，可以快速布置。由于继承了父类 nn.Moudle，定义的类可以实现可实现参数的自动提取，训练等操作。此外在数据集导入时还使用了 Pytorch 的数据加载三件套，可以方便地分割数据集并对数据集进行采样，实现按一定概率或随机采样。

池化层是不包含可学习参数的，因此可以定义一个池化层两个使用。在进行全连接线性网络时，使用了 dropout()函数，用于在训练过程中随机丢弃一些神经元不进行训练，可以有效防止过拟合，而在测试时将 dropout 关闭可以使网络的正确率得到提高。

卷积神经网络具有很好的抗干扰性。图像经过尺度变换，平移旋转变换后特征图与之前相几乎没有变化。池化操作进行了更大尺度的特征提取，局部的小变不会引起高层特征图大变化。这就使得卷积神经网络能够体现出很强的鲁棒。