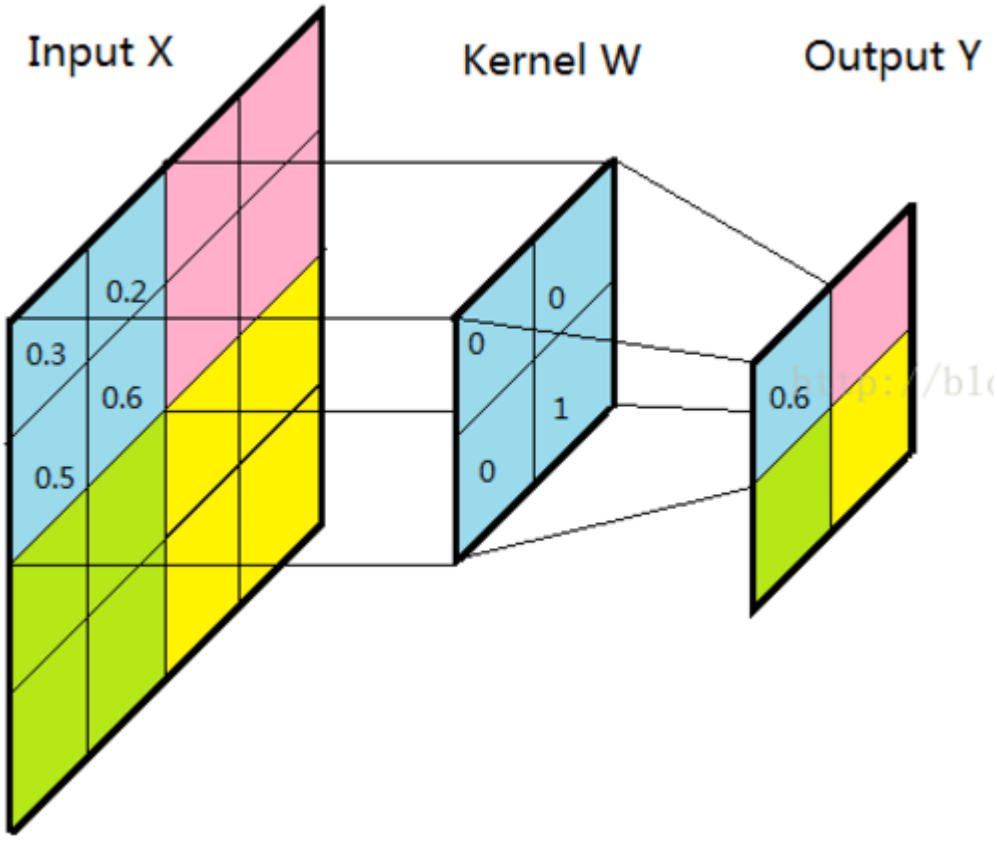
实验二：使用卷积神经网络进行图像识别

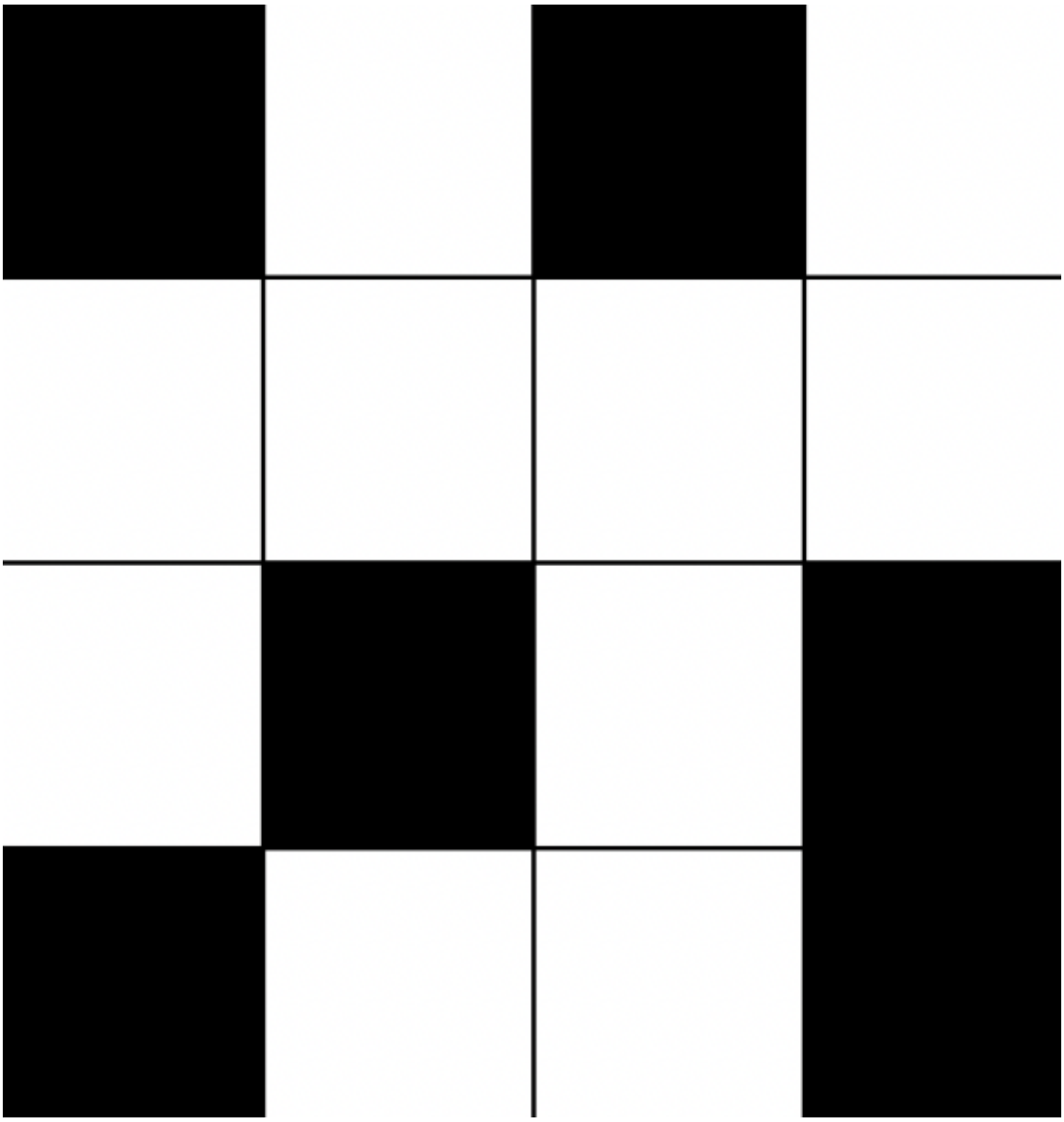
实验内容

卷积神经网络（英语：Convolutional Neural Network，缩写：CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。

卷积神经网络由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。与其他深度学习结构相比，卷积神经网络在图像和语音识别方面能够给出更好的结果。



本次实验的内容是使用Pytorch实现一个卷积神经网络，完成一个图像分类任务。在4*4的图片中，比较外围黑色像素点和内圈黑色像素点个数的大小将图片分类。



如上图图片外围黑色像素点5个大于内圈黑色像素点1个分为0类，反之1类。数据集将由主教直接提供

实验要求

1. 使用Pytorch实现一个2d卷积神经网络，并完成对MNIST数据集的训练和测试。卷积核大小，卷积层数量，batch size和epoch均按需自行设置。
2. 在测试集上的准确率>90%

实验考察能力

1. 掌握卷积神经网络的实现

实验指导

准备工作

除torch, numpy等库外，本次实验需要安装pillow库，可以使用pip install pillow来完成安装

数据集处理

使用下列代码，读取助教提供的训练集和测试集文件即可。

```
import torch.utils.data as Data

class MyDataset(torch.utils.data.Dataset):

    def __init__(self, root, datacsv, transform=None):
        super(MyDataset, self).__init__()
        with open(f'{root}/{datacsv}', 'r') as f:
            imgs = []
            # 读取csv信息到imgs列表
            for path,label in map(lambda line:line.rstrip().split(','),f):
                imgs.append((path, int(label)))
        self.imgs = imgs
        self.transform = transform if transform is not None else lambda x:x

    def __getitem__(self, index):
        path, label = self.imgs[index]
        img = self.transform(Image.open(path).convert('1'))
        return img, label

    def __len__(self):
        return len(self.imgs)

trainData=MyDataset(root = root,datacsv='trainDataInfo.csv',
transform=transforms.ToTensor())
testData=MyDataset(root = root,datacsv='testDataInfo.csv',
transform=transforms.ToTensor())

trainIter = Data.DataLoader(dataset=trainData, batch_size=batchSize, shuffle=True)
testIter = Data.DataLoader(dataset=testData, batch_size=batchSize)
```

为了平衡训练集中0与1类的数量关系，我们对1类进行上采样，代码如下

```
def chooseData(dataset,scale):
    # 将类别为1的排序到前面
    dataset.imgs.sort(key=lambda x:x[1],reverse=True)
    # 获取类别1的数目 ，取scale倍的数组，得数据不那么偏斜
    trueNum =collections.Counter(itertools.chain.from_iterable(dataset.imgs))[1]
    end = min(trueNum*scale,len(dataset))
    dataset.imgs=dataset.imgs[:end]
    random.shuffle(dataset.imgs)
```

卷积神经网络构建

可构建网络如下：

```
from torch import nn
from torch.autograd import Variable
from torch.nn import Module,Linear,Sequential,Conv2d,ReLU,ConstantPad2d
import torch.nn.functional as F

class Net(Module):
    def __init__(self):
        super(Net, self).__init__()

        self.cnnLayers = Sequential(
            Conv2d(1, 1, kernel_size=1, stride=1,bias=True)
        )
        self.linearLayers = Sequential(
            ReLU(),
            Linear(16, 2)
        )

    def forward(self, x):
        x = self.cnnLayers(x)
        x = x.view(x.shape[0], -1)
        x = self.linearLayers(x)
        return x
```

构建损失函数，优化器

```
loss = nn.CrossEntropyLoss()

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

训练并测试模型

```
# 计算准确率
def evaluateAccuracy(dataIter, net):
    accSum, n = 0.0, 0
    with torch.no_grad():
        for X, y in dataIter:
            accSum += (net(X).argmax(dim=1) == y).float().sum().item()
            n += y.shape[0]
    return accSum / n

def trainAndTest(net, trainIter, testIter, loss, numEpochs, batchSize,
              optimizer):
    for epoch in range(numEpochs):
        trainLossSum, trainAccSum, n = 0.0, 0.0, 0
        for X,y in trainIter:
```

```
            yHat = net(X)
            l = loss(yHat,y).sum()
            optimizer.zero_grad()
            l.backward()
            optimizer.step()
            # 计算训练准确度和loss
            trainLossSum += l.item()
            trainAccSum += (yHat.argmax(dim=1) == y).sum().item()
            n += y.shape[0]
        # 评估测试准确度
        testAcc = evaluateAccuracy(testIter, net)
        print('epoch {:d}, loss {:.4f}, train acc {:.3f}, test acc
{:.3f}'.format(epoch + 1, trainLossSum / n, trainAccSum / n, ))
```

完整代码

```
import torch
import torchvision
import torchvision.transforms as transforms
from torch import nn
from torch.autograd import Variable
from torch.nn import Module,Linear,Sequential,Conv2d,ReLU,ConstantPad2d
import torch.utils.data as Data
import torch.nn.functional as F
import numpy as np
from PIL import Image
import csv
import collections
import os
import shutil
import itertools
import random

class MyDataset(torch.utils.data.Dataset):

    def __init__(self, root, datacsv, transform=None):
        super(MyDataset, self).__init__()
        with open(f'{root}/{datacsv}', 'r') as f:
            imgs = []
            # 读取csv信息到imgs列表
            for path,label in map(lambda line:line.rstrip().split(','),f):
                imgs.append((path, int(label)))
        self.imgs = imgs
        self.transform = transform if transform is not None else lambda x:x

    def __getitem__(self, index):
        path, label = self.imgs[index]
        img = self.transform(Image.open(path).convert('1'))
        return img, label
```

```
    def __len__(self):
        return len(self.imgs)

class Net(Module):
    def __init__(self):
        super(Net, self).__init__()

        self.cnnLayers = Sequential(
            Conv2d(1, 1, kernel_size=1, stride=1,bias=True)
        )
        self.linearLayers = Sequential(
            ReLU(),
            Linear(16, 2)
        )

    def forward(self, x):
        x = self.cnnLayers(x)
        x = x.view(x.shape[0], -1)
        x = self.linearLayers(x)
        return x

def trainAndTest(net, trainIter, testIter, loss, numEpochs, batchSize,
             optimizer):
    for epoch in range(numEpochs):
        trainLossSum, trainAccSum, n = 0.0, 0.0, 0
        for X,y in trainIter:
            yHat = net(X)
            l = loss(yHat,y).sum()
            optimizer.zero_grad()
            l.backward()
            optimizer.step()
            # 计算训练准确度和loss
            trainLossSum += l.item()
            trainAccSum += (yHat.argmax(dim=1) == y).sum().item()
            n += y.shape[0]
        # 评估测试准确度
        testAcc = evaluateAccuracy(testIter, net)
        print('epoch {:d}, loss {:.4f}, train acc {:.3f}, test acc
{:.3f}'.format(epoch + 1, trainLossSum / n, trainAccSum / n, testAcc))


def chooseData(dataset,scale):
    # 将类别为1的排序到前面
    dataset.imgs.sort(key=lambda x:x[1],reverse=True)
    # 获取类别1的数目 ，取scale倍的数组，得数据不那么偏斜
    trueNum =collections.Counter(itertools.chain.from_iterable(dataset.imgs))[1]
    end = min(trueNum*scale,len(dataset))
    dataset.imgs=dataset.imgs[:end]
    random.shuffle(dataset.imgs)

# 计算准确率
def evaluateAccuracy(dataIter, net):
    accSum, n = 0.0, 0
    with torch.no_grad():
```

```
        for X, y in dataIter:
            accSum += (net(X).argmax(dim=1) == y).float().sum().item()
            n += y.shape[0]
    return accSum / n



root='data'
trainData=MyDataset(root = root,datacsv='trainDataInfo.csv',
transform=transforms.ToTensor())
testData=MyDataset(root = root,datacsv='testDataInfo.csv',
transform=transforms.ToTensor())

scale = 10
chooseData(trainData,scale)
print(len(trainData))
print(len(testData))

# 超参数
batchSize = 64
lr = 0.1
numEpochs = 10

trainIter = Data.DataLoader(dataset=trainData, batch_size=batchSize, shuffle=True)
testIter = Data.DataLoader(dataset=testData, batch_size=batchSize)


# 交叉熵损失函数
loss = nn.CrossEntropyLoss()

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

trainAndTest(net, trainIter, testIter, loss, numEpochs, batchSize,optimizer)
```