# 全部代码(及部分注释)：

# import torch import numpy as np import torchvision from torch.utils.data import DataLoader from torchvision.datasets import mnist from torch import nn from torch.autograd import Variable from torch import optim from torchvision import transforms

*# 定义神经网络***class** CNN(nn.Module):  
 **def** \_\_init\_\_(self):  
 super(CNN, self).\_\_init\_\_()

#调用父类初始化函数  
  
 self.model = nn.Sequential(

#将网络结构放在一个序列当中，可以方便forward函数的书写  
  
 nn.Conv2d(3, 48, kernel\_size=11, stride=4, padding=2),

#卷积层 *input[3, 120, 120] output[48, 55, 55]* nn.ReLU(inplace=**True**),

#激活层，非线性处理  
 nn.MaxPool2d(kernel\_size=3, stride=2),

#最大池化  *output[48, 27, 27]*

nn.Conv2d(48, 128, kernel\_size=5, padding=2),

*# output[128, 27, 27]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2),

*# output[128, 13, 13]* nn.Conv2d(128, 192, kernel\_size=3, padding=1),

*# output[192, 13, 13]* nn.ReLU(inplace=**True**),  
  
 nn.Conv2d(192, 192, kernel\_size=3, padding=1),

*# output[192, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.Conv2d(192, 128, kernel\_size=3, padding=1),

*# output[128, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2),

*# output[128, 6, 6]* nn.Flatten(),

#降维展平  
 nn.Dropout(p=0.5),

#防止模型过拟合  
 nn.Linear(512, 2048),

#线性层  
 nn.ReLU(inplace=**True**),  
 nn.Dropout(p=0.5),  
 nn.Linear(2048, 1024),  
 nn.ReLU(inplace=**True**),  
 nn.Linear(1024, 5),  
 )  
 **def** forward(self, x):

#前向传播  
 x = self.model(x)  
 **return** x  
  
*# 预处理*

data\_transform = {  
 **"train"**: transforms.Compose([transforms.CenterCrop(120),  
 transforms.RandomHorizontalFlip(),  
 transforms.ToTensor(),  
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]),  
 **"val"**: transforms.Compose([transforms.CenterCrop(120),  
 transforms.RandomHorizontalFlip(),  
 transforms.ToTensor(),  
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])}  
  
train\_set = torchvision.datasets.ImageFolder(root=**"date\\train"**, transform=data\_transform[**"train"**])

#准备数据集  
train\_data = DataLoader(dataset=train\_set, batch\_size=128, shuffle=**True**, num\_workers=0)  
#加载数据集  
test\_set = torchvision.datasets.ImageFolder(root=**"date\\val"**, transform=data\_transform[**"val"**])  
test\_data = DataLoader(dataset=test\_set, batch\_size=128, shuffle=**True**, num\_workers=0)  
net = CNN()  
criterion = nn.CrossEntropyLoss() *# 使用交叉熵损失*optimizer = optim.SGD(net.parameters(), 1e-2) *# 随机梯度下降，学习率为0.1*nums\_epoch = 20

#训练轮数  
device = torch.device(**"cuda:0" if** torch.cuda.is\_available() **else "cpu"**)

#优先使用GPU(如果有)

*# 开始训练*  
**for** epoch **in** range(nums\_epoch):  
 print(epoch + 1)  
 train\_loss = 0  
 train\_acc = 0  
 net = net.train()  
 **for** img, label **in** train\_data:img.to(device)  
 label.to(device)  
 img = Variable(img)label = Variable(label)  
out = net(img)  
 loss = criterion(out, label)optimizer.zero\_grad() *# 每次将梯度重置为0* loss.backward() *# 反向调整参数* optimizer.step()  
  
 *# 记录误差* train\_loss += loss.item()  
 *# 计算分类的准确率* pred = out.max(1) *# 取评分最高的结果作为所分的类别* num\_correct = (pred == label).sum().item()  
 acc = num\_correct / img.shape[0]  
  
 train\_acc += acc  
  
 eval\_loss = 0  
 eval\_acc = 0  
 *# 测试集不训练* **for** img, label **in** test\_data:  
 img.to(device)  
 label.to(device)  
  
 net.eval()img = Variable(img)  
 label = Variable(label)  
  
 out = net(img)  
  
 loss = criterion(out, label)  
  
 *# 记录误差* eval\_loss += loss.item()  
  
 \_, pred = out.max(1)  
 num\_correct = (pred == label).sum().item()  
 acc = num\_correct / img.shape[0]  
  
 eval\_acc += acc  
 print(**'Epoch {} Train Loss {} Train Accuracy {} Teat Loss {} Test Accuracy {}'**.format(  
 epoch + 1, train\_loss / len(train\_data), train\_acc / len(train\_data), eval\_loss / len(test\_data),  
 eval\_acc / len(test\_data)))  
 torch.save(net.state\_dict(), **"net.pth"**)

#保存模型至“net.pth”  
 print(**"模型已保存"**)