基于CNN的手写数字识别

2025年12月16日

柯维
202421324924

训练数据

先定义划分函数

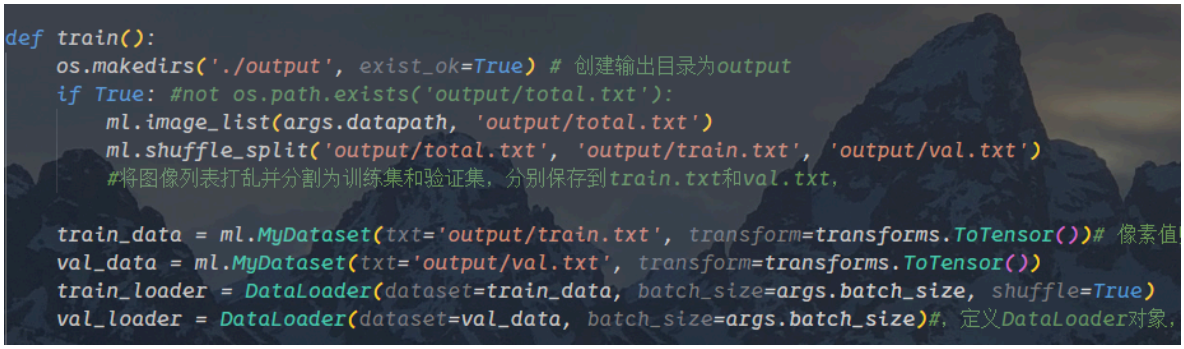
In [11]:
```
from IPython.display import Image, display

display(Image(filename='pictures/loader.png'))
```

```
def shuffle_split(listFile, trainFile, valFile):
    with open(listFile, 'r') as f:
        records = f.readlines()
    random.shuffle(records)
    num = len(records)
    trainNum = int(num * 0.8)#先打乱顺序，再划分 80% 训练集，20% 测试集
    #trainNum = 100
    with open(trainFile, 'w') as f:
        f.writelines(records[0:trainNum])
    with open(valFile, 'w') as f1:
        f1.writelines(records[trainNum:])
```

划分80%训练集，20%测试集，随机采样。

- 在主训练函数中，首先创建输出目录，使用自定义shuffle函数将图像打乱，并将图像分为训练集和测试集。再使用totensor预处理将传入图片的像素值归一化到0-1之间。
- 设置自定义dataloader加载数据集，并设置batchsize大小。

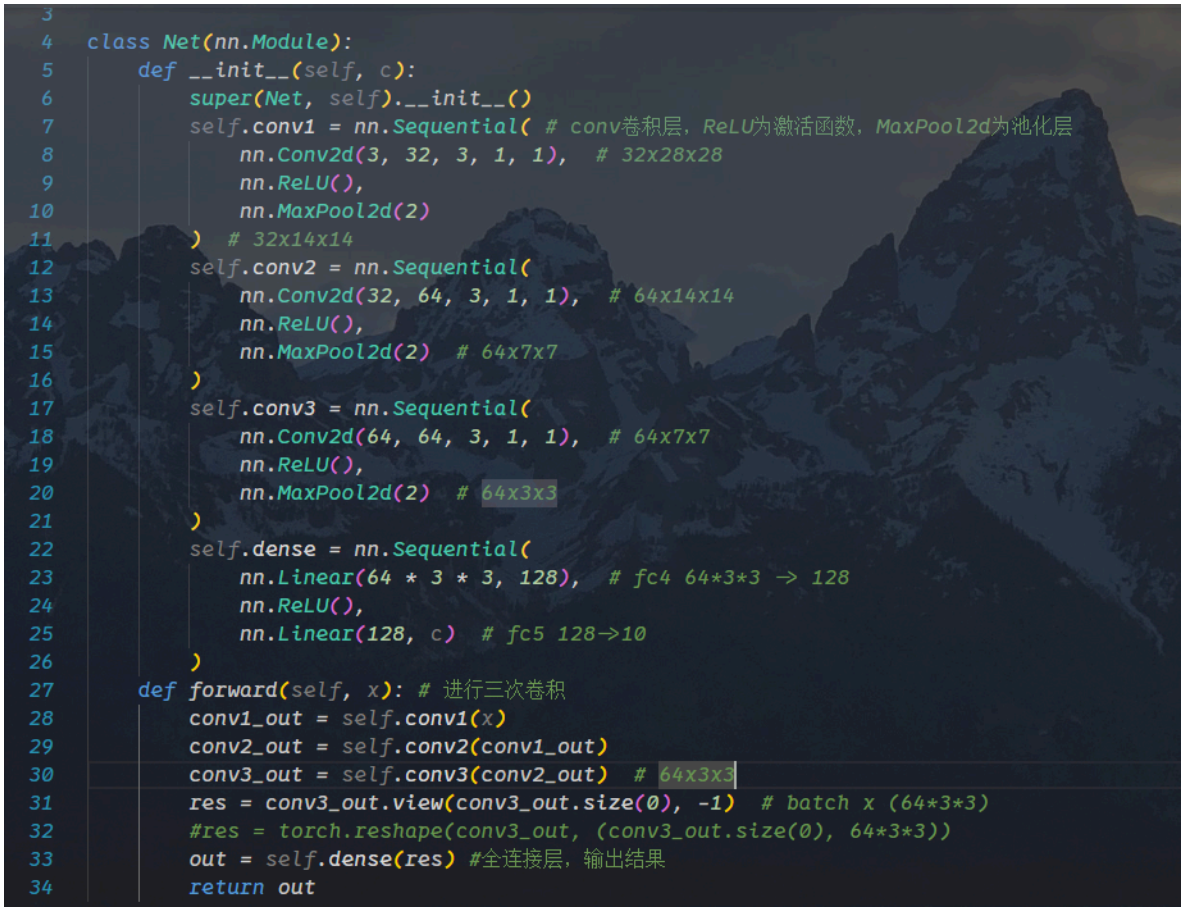
In [10]:
```
from IPython.display import Image, display

display(Image(filename='pictures/train_dataset.png'))
```



```
def train():
    os.makedirs('./output', exist_ok=True) # 创建输出目录为output
    if True: #not os.path.exists('output/total.txt'):
        ml.image_list(args.datapath, 'output/total.txt')
        ml.shuffle_split('output/total.txt', 'output/train.txt', 'output/val.txt')
        #将图像列表打乱并分割为训练集和验证集，分别保存到train.txt和val.txt,

    train_data = ml.MyDataset(txt='output/train.txt', transform=transforms.ToTensor())# 像素值
    val_data = ml.MyDataset(txt='output/val.txt', transform=transforms.ToTensor())
    train_loader = DataLoader(dataset=train_data, batch_size=args.batch_size, shuffle=True)
    val_loader = DataLoader(dataset=val_data, batch_size=args.batch_size)#, 定义DataLoader对象,
```

卷积神经网络如下

In [8]:
```
from IPython.display import Image, display

display(Image(filename='pictures/cnn.png'))
```



```
class Net(nn.Module):
    def __init__(self, c):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential( # conv卷积层，ReLU为激活函数，MaxPool2d为池化层
            nn.Conv2d(3, 32, 3, 1, 1),  # 32x28x28
            nn.ReLU(),
            nn.MaxPool2d(2)
        )  # 32x14x14
        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, 3, 1, 1),  # 64x14x14
            nn.ReLU(),
            nn.MaxPool2d(2)  # 64x7x7
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 64, 3, 1, 1),  # 64x7x7
            nn.ReLU(),
            nn.MaxPool2d(2)  # 64x3x3
        )
        self.dense = nn.Sequential(
            nn.Linear(64 * 3 * 3, 128),  # fc4 64*3*3 -> 128
            nn.ReLU(),
            nn.Linear(128, c)  # fc5 128->10
        )
    def forward(self, x): # 进行三次卷积
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        conv3_out = self.conv3(conv2_out)  # 64x3x3
        res = conv3_out.view(conv3_out.size(0), -1)  # batch x (64*3*3)
        #res = torch.reshape(conv3_out, (conv3_out.size(0), 64*3*3))
        out = self.dense(res) #全连接层，输出结果
        return out
```

- conv1，2，3都是由一个卷积层，一个ReLU层，一个最大池化层组成。经过三次卷积后进行全连接。

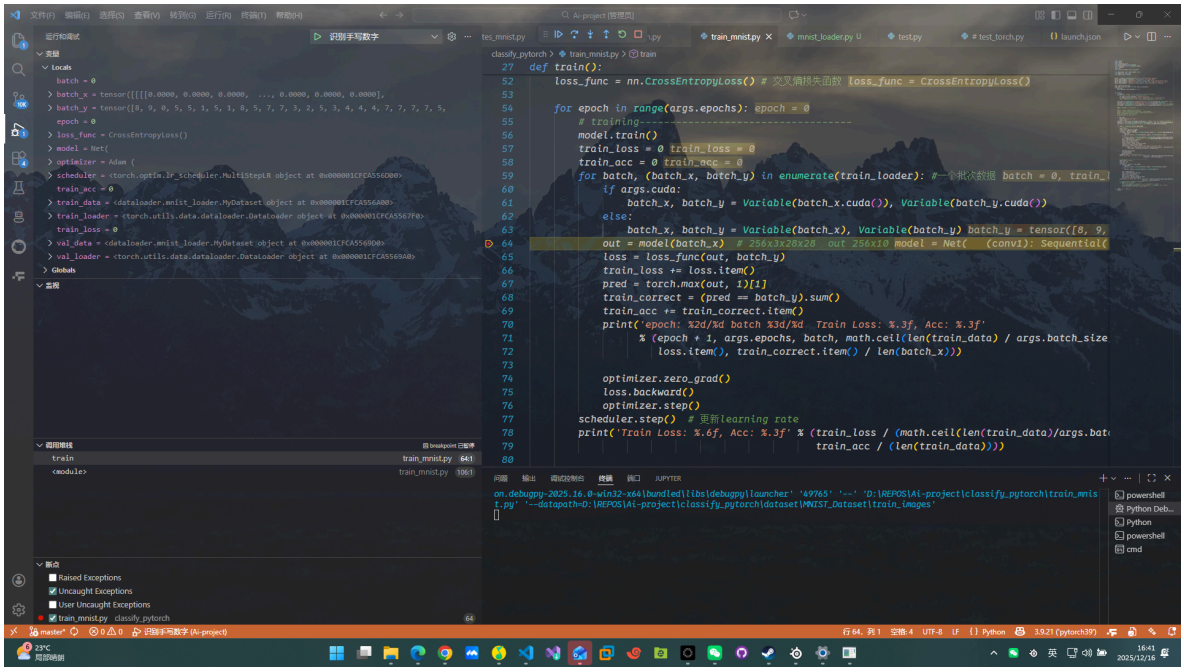
训练时阶梯调整学习速率，初始时为0.01，在第20,30个epoch降低为原来的0.1倍。

In [12]:
```
from IPython.display import Image, display

display(Image(filename='pictures/learning_rate.png'))
```

```
if args.cuda:
    print('training with cuda')
    model.cuda()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=1e-3)# 阶梯调整学习率
scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, [20, 30], 0.1)
loss_func = nn.CrossEntropyLoss() # 交叉熵损失函数
```

训练中添加断点，batch_x为 $256*3*28*28$张量 其中3表示rgb3通道，$28*28$表示图片大小

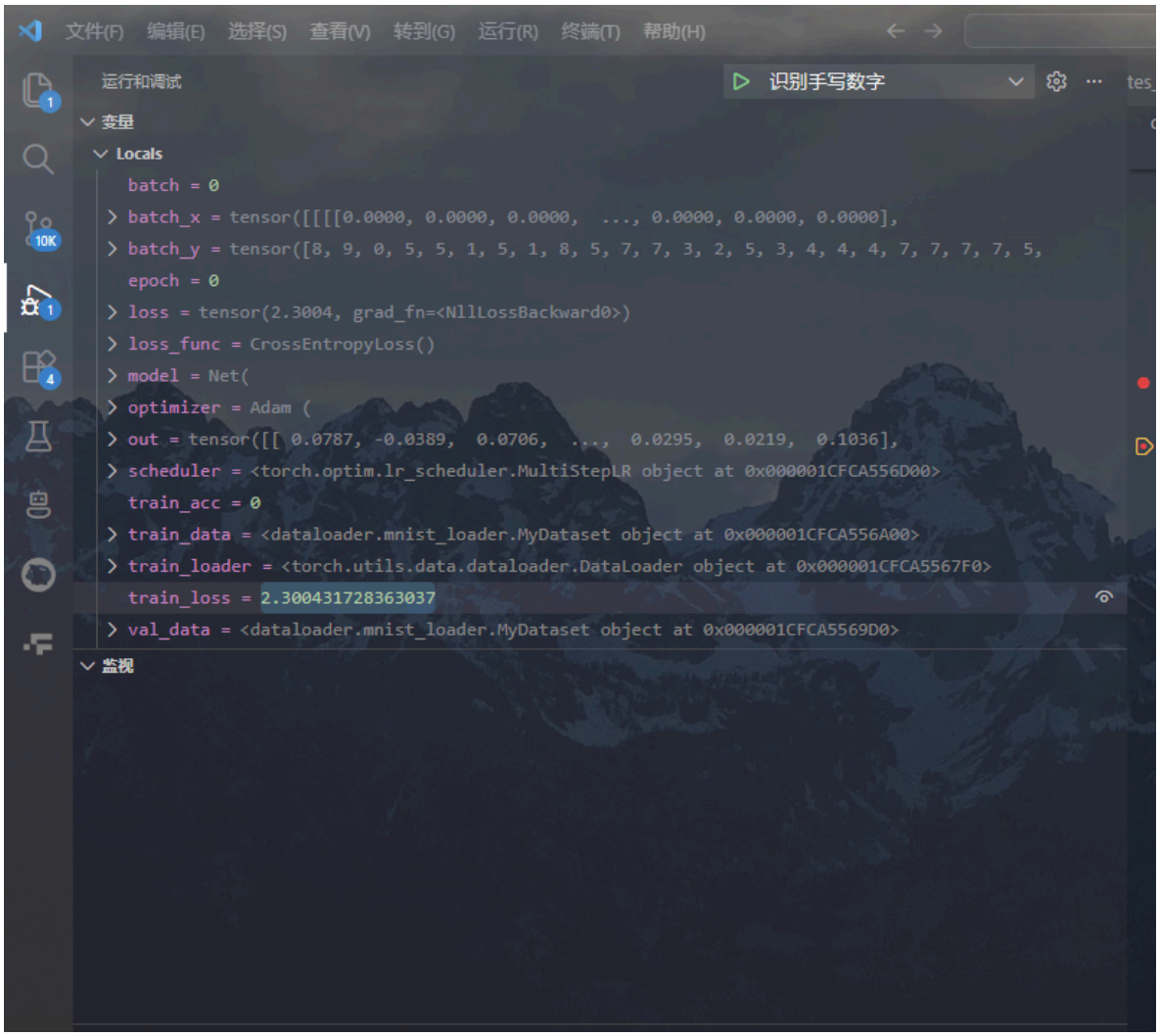
In [ ]:
```
from IPython.display import Image, display

display(Image(filename='pictures/breakpoint.png'))
```



输出为256*10，表示一批256个样本，每个图片有10个值，分别表示每个类别的概率。

概率经过交叉熵函数得到

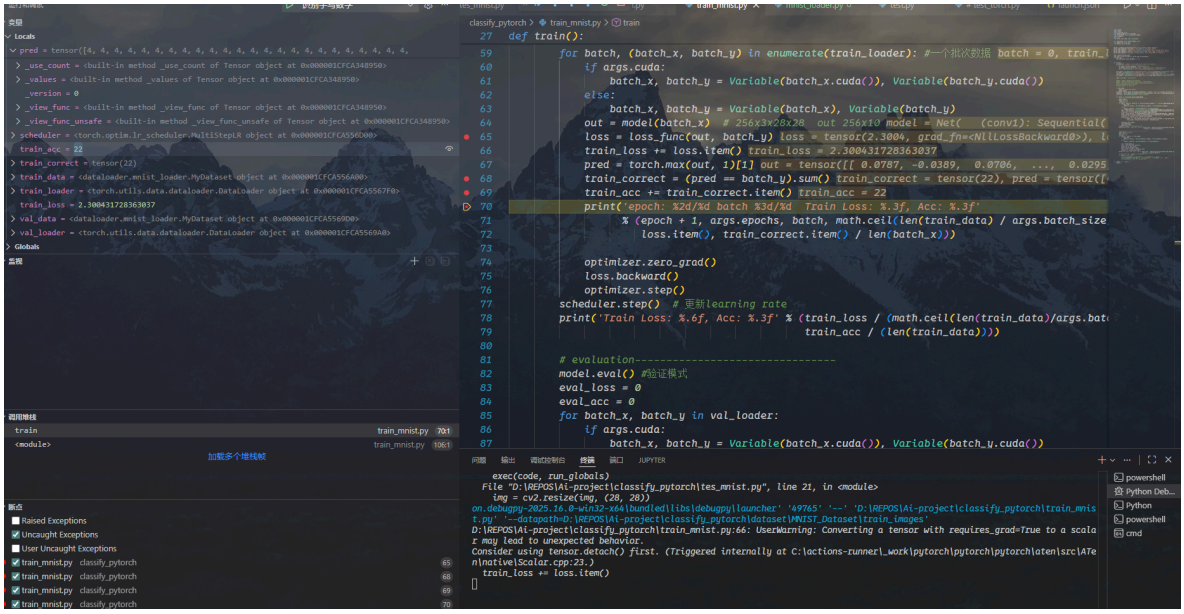
In [6]:
```
from IPython.display import Image, display

display(Image(filename='pictures/train_loss.png'))
```



得到损失函数值约为2.3

与标签作比较得到一个批次训练的准确率

In [7]:
```
from IPython.display import Image, display

display(Image(filename='pictures/accuracy.png'))
```



- acc=22，即256个样本中有22个分类正确，准确率为22/256=8.59%，经过多批次训练准确率会提高。

训练结果可视化

导入库

In [ ]:
```
import torch
import cv2
import numpy as np
import matplotlib.pyplot as plt
from torch.autograd import Variable
from torchvision import transforms
from models.cnn import Net
```

先调用模型计算出概率，获取预测类别。

In [ ]:
```
probabilities = torch.nn.functional.softmax(prediction, dim=1)
pred = torch.max(prediction, 1)[1]
```

创建可视化画布，左侧子图显示输入图像，右侧子图显示预测结果。

可视化结果：

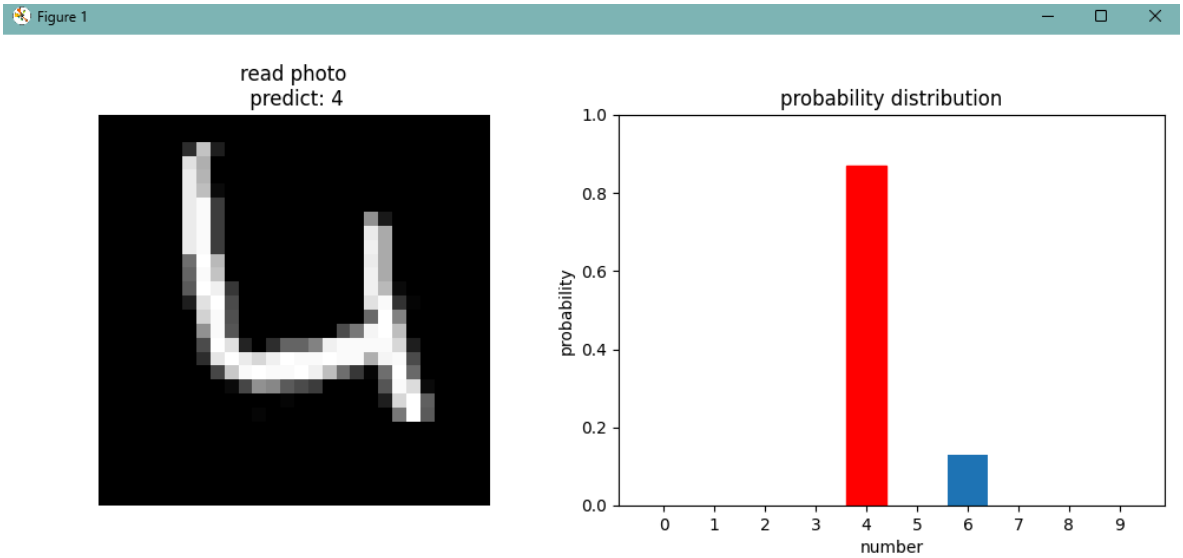
In [ ]:
```
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  # BGR转RGB显示
```

```
plt.title(f"read photo\n predict: {pred.item()}")
plt.axis('off')
```

结果分析

训练50个epoch，分析简单手写数字的识别准确率几乎为100%，为展示效果，使用第三次训练的结果。

In [13]:
```
from IPython.display import Image, display

display(Image(filename='pictures/probability.png'))
```



- 输入图片为数字4，识别为数字4的概率为0.8691

In [ ]:
```

```