

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'): #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,  
        ml.image_list(args.datapath, 'output/total.txt')  
        ml.shuffle_split('output/total.txt', 'output/train.txt', 'output/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'))

3
4  class Net(nn.Module):
5      def __init__(self, c):
6          super(Net, self).__init__()
7          self.conv1 = nn.Sequential( # conv卷积层, ReLU为激活函数, MaxPool2d为池化层
8              nn.Conv2d(3, 32, 3, 1, 1), # 32x28x28
9              nn.ReLU(),
10             nn.MaxPool2d(2)
11         ) # 32x14x14
12         self.conv2 = nn.Sequential(
13             nn.Conv2d(32, 64, 3, 1, 1), # 64x14x14
14             nn.ReLU(),
15             nn.MaxPool2d(2) # 64x7x7
16         )
17         self.conv3 = nn.Sequential(
18             nn.Conv2d(64, 64, 3, 1, 1), # 64x7x7
19             nn.ReLU(),
20             nn.MaxPool2d(2) # 64x3x3
21         )
22         self.dense = nn.Sequential(
23             nn.Linear(64 * 3 * 3, 128), # fc4 64*3*3 -> 128
24             nn.ReLU(),
25             nn.Linear(128, c) # fc5 128->10
26         )
27     def forward(self, x): # 进行三次卷积
28         conv1_out = self.conv1(x)
29         conv2_out = self.conv2(conv1_out)
30         conv3_out = self.conv3(conv2_out) # 64x3x3
31         res = conv3_out.view(conv3_out.size(0), -1) # batch x (64*3*3)
32         #res = torch.reshape(conv3_out, (conv3_out.size(0), 64*3*3))
33         out = self.dense(res) #全连接层, 输出结果
34         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'))
```

文件(F) 菜单(M) 选项(O) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H) < - > Q A-project (标签页)

本地文件夹

资源

本地

batch_x = tensor([[0.0000, 0.0000, 0.0000],
... , 0.0000, 0.0000, 0.0000],
epoch = 0
loss_func = CrossEntropyLoss()
model = Net()
optimizer = Adam(
scheduler = torch.optim.lr_scheduler.MultiStepLR object at 0x000001CFC5A56000)
train_acc = 0
train_loader = torch.utils.data.DataLoader object at 0x000001CFC5A56700
train_loss = 0
val_data = DataLoader.mnist_loader.MyDataset object at 0x000001CFC5A56800
val_loader = torch.utils.data.DataLoader object at 0x000001CFC5A56900

Globals

监视

全局变量

train

train.module

train.train_minst.py

train_minst.py (81)

train_minst.py

classify_pytorch > train_minst.py > train_minst.py

```
def train():
    loss_func = nn.CrossEntropyLoss() # 叉积损失函数 loss_func = CrossEntropyLoss()
    for epoch in range(args.epochs): epoch = 0
        # training
        model.train()
        train_loss = 0
        train_acc = 0
        for batch, (batch_x, batch_y) in enumerate(train_loader): # 一个批次数据 batch = 0, train_
            if args.cuda:
                batch_x, batch_y = Variable(batch_x.cuda()), Variable(batch_y.cuda())
            else:
                batch_x, batch_y = Variable(batch_x), Variable(batch_y)
            out = model(batch_x) # 256x3x28x28 out 256x10 model = Net() (conv1): Sequential(
            loss = loss_func(out, batch_y)
            train.loss += loss.item()
            pred = torch.max(out, 1)[1]
            train.correct = (pred == batch.y).sum()
            train.acc += train.correct.item()
            print('epoch %d/%d batch %d/%d Train Loss: %.3f, Acc: %.3f'
                  % (epoch + 1, args.epochs, batch, math.ceil(len(train_data)) / args.batch_size
                     .item(), train.correct.item() / len(batch_x)))
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        scheduler.step() # 更新learning rate
        print('Train Loss: %.6f, Acc: %.3f' % (train.loss / (math.ceil(len(train_data)) / args.batch_size
                                                       .item()), train.acc / (len(train_data))))
```

问题 跳出 新建会话 代码 调试 窗口 Jupyter

on_debugpy-2025.16.0-win32-x64|bundle|libs|debugpy|launcher|49965' '--': 'D:\REPOS\AI-project\classify_pytorch\train_minst.py' '--datapath=D:\REPOS\AI-project\classify_pytorch\dataset\MNIST_Dataset\train_images'

23°C 局部快照

行 64, 列 1 空格:4 UTF-8 LF [] Python 3.9.21 (pytorch39) 2025/12/16 16:41

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'))
```

The screenshot shows a PyTorch debugger interface with the following details:

- File Menu:** 文件(F), 编辑(E), 选择(S), 查看(V), 转到(G), 运行(R), 终端(T), 帮助(H)
- Toolbar:** 左侧有撤销、重做、剪切、复制、粘贴、粘贴为新行、粘贴为新窗口、粘贴为新线程、粘贴为新线程(所有)、粘贴为新线程(所有)等图标。
- 运行和调试:** 显示“识别手写数字”正在运行。
- 变量:** 展开的“Locals”部分显示了以下变量：

 - batch = 0
 - batch_x = tensor([[[[0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000], ...]
 - batch_y = tensor([8, 9, 0, 5, 5, 1, 5, 1, 8, 5, 7, 7, 3, 2, 5, 3, 4, 4, 4, 4, 7, 7, 7, 7, 5,
 - epoch = 0
 - loss = tensor(2.3004, grad_fn=<NllLossBackward0>)
 - loss_func = CrossEntropyLoss()
 - model = Net()
 - optimizer = Adam()
 - out = tensor([[0.0787, -0.0389, 0.0706, ..., 0.0295, 0.0219, 0.1036], ...]
 - scheduler = <torch.optim.lr_scheduler.MultiStepLR object at 0x000001CFCA556D00>
 - train_acc = 0
 - train_data = <dataloader.mnist_loader.MyDataset object at 0x000001CFCA556A00>
 - train_loader = <torch.utils.data.dataloader.DataLoader object at 0x000001CFCA5567F0>
 - train_loss = 2.300431728363037
 - val_data = <dataloader.mnist_loader.MyDataset object at 0x000001CFCA5569D0>

- 监视:** 展开，显示了正在监视的变量。

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'))
```

```
def train():
    for batch, (batch_x, batch_y) in enumerate(train_loader): #一个批次数据 batch = 0, train_
        if args.cuda:
            batch_x, batch_y = Variable(batch_x.cuda()), Variable(batch_y.cuda())
        else:
            batch_x, batch_y = Variable(batch_x), Variable(batch_y)
        out = model(batch_x) # 256x3x28x28 out 256x10 model = Net_ (conv1): Sequential(
        loss = loss_func(out, batch_y) loss = tensor(2.3004, grad_fn=<NilLossBackward>),
        train_loss += loss.item() train.loss = 2.3004317283636037
        pred = torch.max(out, 1)[1] out = tensor([ 0.0787, -0.0369, 0.0706, ..., 0.0295
        train_correct = (pred == batch_y).sum() train.correct = tensor(22), pred = tensor([
        train_acc += train_correct.item() train.acc = 22
        print('epoch: %d/%d batch %d/%d Train Loss: %.3f, Acc: %.3f'
              % (epoch + 1, args.epochs, batch, math.ceil(len(train_data) / args.batch_size
              loss.item()), train_correct.item() / len(batch_x)))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        scheduler.step() # 更新learning rate
        print('Train Loss: %.6f, Acc: %.3f' % (train_loss / (math.ceil(len(train_data)) / args.batch_size
                                                       train_acc / (len(train_data)))))
    # evaluation-----
    model.eval() #验证模式
    eval_loss = 0
    eval_acc = 0
    for batch_x, batch_y in val_loader:
        if args.cuda:
            batch_x, batch_y = Variable(batch_x.cuda()), Variable(batch_y.cuda())
        out = model(batch_x)
        eval_loss += loss_func(out, batch_y).item()
        pred = torch.max(out, 1)[1]
        eval_correct = (pred == batch_y).sum()
        eval_acc += eval_correct.item()
    print('Val Loss: %.6f, Acc: %.3f' % (eval_loss / len(val_loader), eval_acc / len(val_loader)))
```

- 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(probabilities, 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:

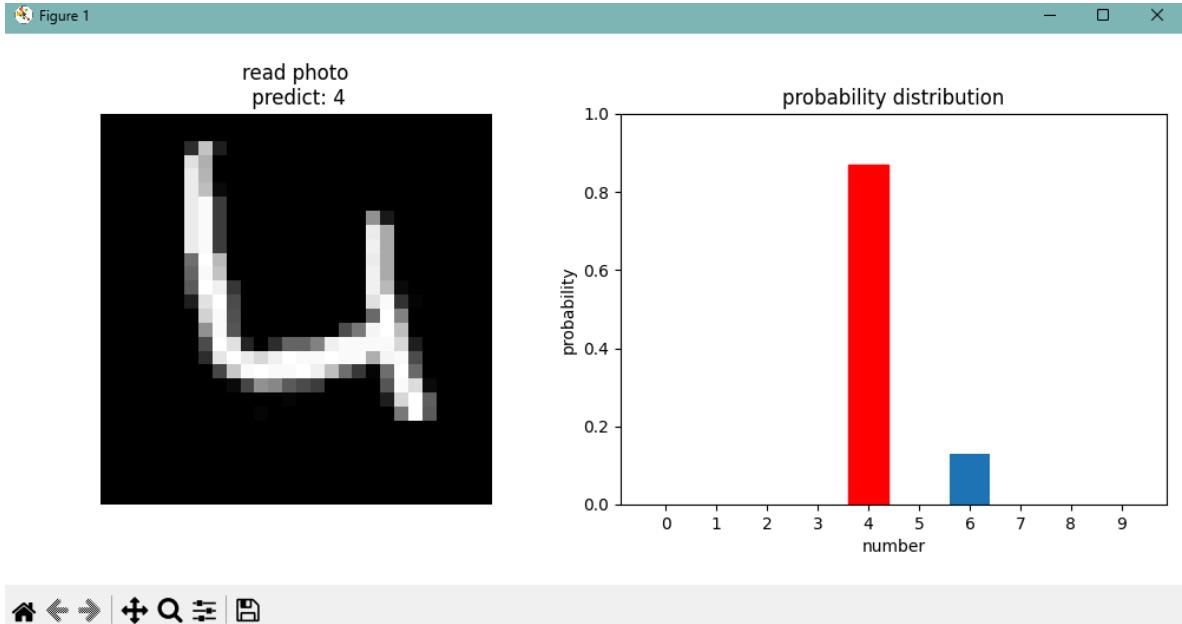
```
In [ ]: plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)) # BGR转RGB显示
```

```
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 [ ]:
```