**手写数字识别的卷积神经网络实现**

手写数字识别的卷积神经网络实现-pytorch

[**数据集**](https://so.csdn.net/so/search?q=数据集&spm=1001.2101.3001.7020)**下载**

卷积神经网络经常被应用于手写数字的数据集[mnist](https://so.csdn.net/so/search?q=mnist&spm=1001.2101.3001.7020)的识别，若数据集下载出现异常，可以到【[MNIST数据集](http://yann.lecun.com/exdb/mnist/)】进行数据集下载。

|  |
| --- |
| Python  train\_data = torchvision.datasets.MNIST(  root='./mnist', #保存或者提取位置  train=True, #如果为True则为训练集，如果为False则为测试集  transform=torchvision.transforms.ToTensor(), #将图片转化成取值[0,1]的Tensor用于网络处理  download=False #是否下载数据集  ) |

**训练数据获取**

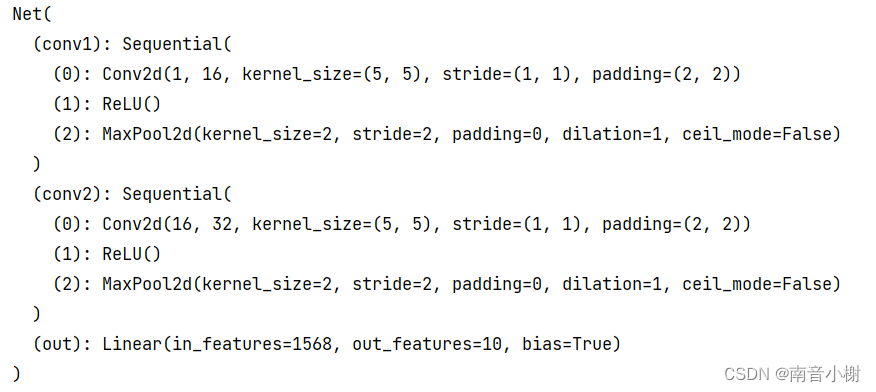
|  |
| --- |
| Python  def dataLoader():  # 获取Mnist手写数字数据集  train\_data = torchvision.datasets.MNIST(  root='./mnist', #保存或者提取位置  train=True, #如果为True则为训练集，如果为False则为测试集  transform=torchvision.transforms.ToTensor(), #将图片转化成取值[0,1]的Tensor用于网络处理  download=False #是否下载数据集  )  # plt.imshow(train\_data.train\_data[0].numpy(),cmap='gray')  # plt.title('%i'%train\_data.train\_labels[0])  # plt.show()  loader = Data.DataLoader(  dataset=train\_data,  batch\_size=50, #最小训练批量  shuffle=True, #是否对数据进行随机打乱  num\_workers=2, #多线程来读数据  )    return loader |

**测试数据获取**

|  |
| --- |
| Python  def dataTest():  #获取测试数据  test\_data = torchvision.datasets.MNIST(root='./mnist/', train=False)  #测试前2000个数据  test\_x = torch.unsqueeze(test\_data.test\_data, dim=1).type(torch.FloatTensor)[:2000]/255. to (2000, 1, 28, 28), value in range(0,1)  test\_y = test\_data.test\_labels[:2000]  return test\_x,test\_y |

**网络搭建**

|  |
| --- |
| Python  #搭建网络  class Net(nn.Module):  def \_\_init\_\_(self):  super(Net, self).\_\_init\_\_()  self.conv1 = nn.Sequential( # input shape (1, 28, 28)  nn.Conv2d(  in\_channels=1, # input height  out\_channels=16, # n\_filters  kernel\_size=5, # filter size  stride=1, # filter movement/step  padding=2, # if want same width and length of this image after Conv2d, padding=(kernel\_size-1)/2 if stride=1  ), # output shape (16, 28, 28)  nn.ReLU(), # activation  nn.MaxPool2d(kernel\_size=2), # choose max value in 2x2 area, output shape (16, 14, 14)  )  self.conv2 = nn.Sequential( # input shape (16, 14, 14)  nn.Conv2d(16, 32, 5, 1, 2), # output shape (32, 14, 14)  nn.ReLU(), # activation  nn.MaxPool2d(2), # output shape (32, 7, 7)  )  self.out = nn.Linear(32 \* 7 \* 7, 10) # fully connected layer, output 10 classes    def forward(self, x):  x = self.conv1(x)  x = self.conv2(x)  x = x.view(x.size(0), -1) # flatten the output of conv2 to (batch\_size, 32 \* 7 \* 7)  output = self.out(x)  return output, x # return x for visualization |



**训练过程**

|  |
| --- |
| Python  if \_\_name\_\_=="\_\_main\_\_":  #模拟数据  loader=dataLoader()  test\_x,test\_y=dataTest()  net=Net()  print(net)  #定义优化器  optimizer = torch.optim.Adam(net.parameters(), lr=0.001)  #定义误差函数  loss\_fun=nn.CrossEntropyLoss()    plt.ion()  #迭代训练  for epoch in range(1):  for step, (batch\_x, batch\_y) in enumerate(loader):  #预测  prediction=net(batch\_x)[0]  #计算误差  loss=loss\_fun(prediction,batch\_y)  #梯度降为0  optimizer.zero\_grad()  #反向传递  loss.backward()  #优化梯度  optimizer.step()    if step % 50 == 0:  test\_output, last\_layer = net(test\_x)  pred\_y = torch.max(test\_output, 1)[1].data.numpy()  accuracy = float((pred\_y == test\_y.data.numpy()).astype(int).sum()) / float(test\_y.size(0))  print('Epoch: ', epoch, '| train loss: %.4f' % loss.data.numpy(), '| test accuracy: %.2f' % accuracy) |

**训练结果**

