手写数字识别：

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

from torchvision import datasets, transforms

import torchvision

from torch.autograd import Variable

from torch.utils.data import DataLoader

import cv2

# 下载训练集

train\_dataset = datasets.MNIST(root='./num/',

train=True,

transform=transforms.ToTensor(),

download=True)

# 下载测试集

test\_dataset = datasets.MNIST(root='./num/',

train=False,

transform=transforms.ToTensor(),

download=True)

# dataset 参数用于指定我们载入的数据集名称

# batch\_size参数设置了每个包中的图片数据个数

# 在装载的过程会将数据随机打乱顺序并进打包

# 装载训练集

train\_loader = torch.utils.data.DataLoader(dataset=train\_dataset,

batch\_size=batch\_size,

shuffle=True)

# 装载测试集

test\_loader = torch.utils.data.DataLoader(dataset=test\_dataset,

batch\_size=batch\_size,

shuffle=True)

images, labels = next(iter(data\_loader\_train))

img = torchvision.utils.make\_grid(images)

img = img.numpy().transpose(1, 2, 0)

std = [0.5, 0.5, 0.5]

mean = [0.5, 0.5, 0.5]

img = img \* std + mean

print(labels)

cv2.imshow('win', img)

key\_pressed = cv2.waitKey(0)

# 卷积层使用 torch.nn.Conv2d

# 激活层使用 torch.nn.ReLU

# 池化层使用 torch.nn.MaxPool2d

# 全连接层使用 torch.nn.Linear

class LeNet(nn.Module):

def \_\_init\_\_(self):

super(LeNet, self).\_\_init\_\_()

self.conv1 = nn.Sequential(nn.Conv2d(1, 6, 3, 1, 2), nn.ReLU(),

nn.MaxPool2d(2, 2))

self.conv2 = nn.Sequential(nn.Conv2d(6, 16, 5), nn.ReLU(),

nn.MaxPool2d(2, 2))

self.fc1 = nn.Sequential(nn.Linear(16 \* 5 \* 5, 120),

nn.BatchNorm1d(120), nn.ReLU())

self.fc2 = nn.Sequential(

nn.Linear(120, 84),

nn.BatchNorm1d(84),

nn.ReLU(),

nn.Linear(84, 10))

# 最后的结果一定要变为 10，因为数字的选项是 0 ~ 9

def forward(self, x):

x = self.conv1(x)

x = self.conv2(x)

x = x.view(x.size()[0], -1)

x = self.fc1(x)

x = self.fc2(x)

x = self.fc3(x)

return x

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

batch\_size = 64

LR = 0.001

net = LeNet().to(device)

# 损失函数使用交叉熵

criterion = nn.CrossEntropyLoss()

# 优化函数使用 Adam 自适应优化算法

optimizer = optim.Adam(

net.parameters(),

lr=LR,

)

epoch = 1

if \_\_name\_\_ == '\_\_main\_\_':

for epoch in range(epoch):

sum\_loss = 0.0

for i, data in enumerate(train\_loader):

inputs, labels = data

inputs, labels = Variable(inputs).cuda(), Variable(labels).cuda()

optimizer.zero\_grad() #将梯度归零

outputs = net(inputs) #将数据传入网络进行前向运算

loss = criterion(outputs, labels) #得到损失函数

loss.backward() #反向传播

optimizer.step() #通过梯度做一步参数更新

# print(loss)

sum\_loss += loss.item()

if i % 100 == 99:

print('[%d,%d] loss:%.03f' %

(epoch + 1, i + 1, sum\_loss / 100))

sum\_loss = 0.0

net.eval() #将模型变换为测试模式

correct = 0

total = 0

for data\_test in test\_loader:

images, labels = data\_test

images, labels = Variable(images).cuda(), Variable(labels).cuda()

output\_test = net(images)

\_, predicted = torch.max(output\_test, 1)

total += labels.size(0)

correct += (predicted == labels).sum()

print("correct1: ", correct)

print("Test acc: {0}".format(correct.item() /

len(test\_dataset)))