# 手写字符案例分析

# 1、模型结构

# 1、模型代码

```
class Net(nn.Module):
2
        def __init__(self):
 3
            super(Net, self).__init__()
4
            self.conv1 = nn.Sequential(
 5
                nn.Conv2d(3, 32, 3, 1, 1), # 32x28x28
                nn.ReLU(),
6
7
                nn.MaxPool2d(2)
8
            ) # 32x14x14
9
            self.conv2 = nn.Sequential(
10
                nn.Conv2d(32, 64, 3, 1, 1), # 64x14x14
11
                nn.ReLU(),
12
                nn.MaxPool2d(2) # 64x7x7
13
            self.conv3 = nn.Sequential(
14
                nn.Conv2d(64, 64, 3, 1, 1), # 64x7x7
15
16
                nn.ReLU(),
17
                nn.MaxPool2d(2) # 64x3x3
18
19
20
            self.dense = nn.Sequential(
21
                nn.Linear(64 * 3 * 3, 128), # fc4 64*3*3 -> 128
22
                nn.ReLU(),
                nn.Linear(128, 10) # fc5 128->10
23
24
            )
25
26
        def forward(self, x):
27
            conv1\_out = self.conv1(x)
28
            conv2_out = self.conv2(conv1_out)
29
            conv3\_out = self.conv3(conv2\_out) # 64x3x3
            res = conv3\_out.view(conv3\_out.size(0), -1) # batch x (64*3*3)
30
31
            out = self.dense(res)
32
            return out
33
```

# 2、主要函数

```
Conv2d函数简介:
torch.nn.Conv2d(
in_channels,
out_channels,
kernel_size,
stride=1,
padding=0,
dilation=1,
groups=1,
```

```
bias=True,
padding_mode='zeros',
device=None,
dtype=None)
in_channels (int) - 输入图像中的通道数
out_channels (int) - 卷积产生的通道数即输出图片的通道数
kernel_size (int or tuple) - 卷积核的大小(可以是个数,也可以是元组)
stride (int or tuple, optional) -- 卷积的步幅。默认值:1
padding (int, tuple or str, optional) -- 填充添加到输入的所有四个边。默认值:0
padding_mode (string, optional) -- 填充的几个选择 'zeros', 'reflect', 'replicate' 或 'circular'。默认值:"零"
dilation (int or tuple, optional) -- 内核元素之间的间距。默认值:1
groups (int, optional) -- 从输入通道到输出通道的阻塞连接数。默认值:1
bias (bool, optional) -- 如果为真,则为输出添加可学习的偏差。默认值:真
```

MaxPool2d函数简介

#### 先来看源码:

```
1 class _MaxPoolNd(Module):
     __constants__ = ['kernel_size', 'stride', 'padding', 'dilation',
 2
 3
                      'return_indices', 'ceil_mode']
4
    return_indices: bool
 5
     ceil_mode: bool
 6
        # 构造函数,这里只需要了解这个初始化函数即可。
     def __init__(self, kernel_size: _size_any_t, stride:
    Optional[_size_any_t] = None,
8
                  padding: _size_any_t = 0, dilation: _size_any_t = 1,
                  return_indices: bool = False, ceil_mode: bool = False) ->
9
    None:
10
        super(_MaxPoolNd, self).__init__()
         self.kernel_size = kernel_size
11
12
        self.stride = stride if (stride is not None) else kernel_size
13
        self.padding = padding
14
        self.dilation = dilation
15
         self.return_indices = return_indices
16
         self.ceil_mode = ceil_mode
17
18
     def extra_repr(self) -> str:
19
        return 'kernel_size={kernel_size}, stride={stride}, padding=
    {padding}' \
             ', dilation={dilation}, ceil_mode=
20
    {ceil_mode}'.format(**self.__dict__)
21
22
    class MaxPool2d(_MaxPoolNd):
     kernel_size: _size_2_t
23
24
     stride: _size_2_t
25
     padding: _size_2_t
26
     dilation: _size_2_t
27
     def forward(self, input: Tensor) -> Tensor:
28
```

return F.max\_pool2d(input, self.kernel\_size, self.stride,
self.padding, self.dilation, self.ceil\_mode,
self.return\_indices)

MaxPool2d 这个类的实现十分简单。

#### 我们先来看一下基本参数,一共六个:

kernel\_size: 表示做最大池化的窗口大小,可以是单个值,也可以是tuple元组

stride: 步长,可以是单个值,也可以是tuple元组 padding:填充,可以是单个值,也可以是tuple元组

dilation: 控制窗口中元素步幅

return\_indices: 布尔类型,返回最大值位置索引

ceil\_mode: 布尔类型, 为True, 用向上取整的方法, 计算输出形状; 默认是向下取整。

关于 kernel\_size 的详解:

注意这里的 kernel\_size 跟卷积核不是一个东西。 kernel\_size 可以看做是一个滑动窗口,这个窗口的大小由自己指定,如果输入是单个值,例如 3 33 ,那么窗口的大小就是 3 × 3 ,还可以输入元组,例如 (3, 2) ,那么窗口大小就是 3 × 2 。

最大池化的方法就是取这个窗口覆盖元素中的最大值。

#### 关于 stride 的详解:

上一个参数我们确定了滑动窗口的大小,现在我们来确定这个窗口如何进行滑动。如果不指定这个参数,那么默认步长跟最大池化窗口大小一致。如果指定了参数,那么将按照我们指定的参数进行滑动。例如 stride=(2,3) , 那么窗口将每次向右滑动三个元素位置,或者向下滑动两个元素位置。

#### 关于 padding 的详解:

这参数控制如何进行填充,填充值默认为0。如果是单个值,例如 1,那么将在周围填充一圈0。还可以用元组指定如何填充,例如 padding=(2,1) ,表示在上下两个方向个填充两行0,在左右两个方向各填充一列0。

关于 dilation 的详解:

#### 不会

关于 return\_indices 的详解:

这是个布尔类型值,表示返回值中是否包含最大值位置的索引。注意这个最大值指的是在所有窗口中产生的最大值,如果窗口产生的最大值总共有5个,就会有5个返回值。

关于 ceil mode 的详解:

这个也是布尔类型值,它决定的是在计算输出结果形状的时候,是使用向上取整还是向下取整。怎么计算输出形状,下面会讲到。一看就知道了。

nn.linear()是用来设置网络中的全连接层的,而在全连接层中的输入与输出都是二维张量,一般形状为[batch\_size, size],与卷积层要求输入输出是4维张量不同。 用法与形参见说明如下:

#### CLASS torch.nn.Linear(in\_features, out\_features, bias=True)

Applies a linear transformation to the incoming data:  $y = xA^T + b$ 

#### **Parameters**

- in\_features size of each input sample
- · out\_features size of each output sample
- bias If set to False, the layer will not learn an additive bias. Default: True

nn.Linear

in\_features指的是输入的二维张量的大小,即输入的[batch\_size, size]中的size。batch\_size指的是每次训练(batch)的时候样本的大小。比如CNN train的样张图片是60张,设置batch\_size=15,那么iteration=4。如果想多训练几次(因为可以每次的batch不是相同的数据),那么就是epoch。

所以nn.Linear()中的输入包括有输入的图片数量,同时还有每张图片的维度。 out\_features指的是输出的二维张量的大小,即输出[batch\_size, size]中的size是输出的张量维度,而batch\_size与输入中的一致。

## 3、结构分析

- Conv1层
- 1. 卷积层:输入3通道,输出32通道,卷积核3×3,步长为1,四边填入1
- 2. 激活函数: ReLu ()
- 3. 池化层:最大池化,窗口2×2
- Conv2层
- 1. 卷积层:输入32通道,输出64通道,卷积核3×3,步长为1,四边填入1
- 2. 激活函数: ReLu ()
- 3. 池化层:最大池化,窗口2×2
- Conv3层
- 1. 卷积层:输入64通道,输出64通道,卷积核3×3,步长为1,四边填入1
- 2. 激活函数: ReLu ()
- 3. 池化层:最大池化,窗口2×2
- dense层
- 1. 全连接层1: 输入为64×3×3, 输出为128
- 2. 激活函数: ReLu ()
- 3. 全连接层2: 输入为128, 输出为10 (十个分类)

流程: Conv1->Conv2->Conv3->dense

# 2、数据加载代码

## 1、代码总览

```
def image_list(imageRoot, txt='list.txt'):
 2
        f = open(txt, 'wt')
 3
        for (label, filename) in enumerate(sorted(os.listdir(imageRoot),
    reverse=False)):
 4
            if os.path.isdir(os.path.join(imageRoot, filename)):
 5
                 for imagename in os.listdir(os.path.join(imageRoot, filename)):
 6
                     name, ext = os.path.splitext(imagename)
 7
                     ext = ext[1:]
 8
                     if ext == 'jpg' or ext == 'png' or ext == 'bmp':
 9
                         f.write('%s %d\n' % (os.path.join(imageRoot, filename,
    imagename), label))
        f.close()
10
11
12
13
    def shuffle_split(listfile, trainfile, valfile):
14
        with open(listfile, 'r') as f:
             records = f.readlines()
15
16
        random.shuffle(records)
        num = len(records)
17
        trainNum = int(num * 0.8)
18
19
        with open(trainfile, 'w') as f:
            f.writelines(records[0:trainNum])
20
21
        with open(valfile, 'w') as f1:
            f1.writelines(records[trainNum:])
22
23
24
25
    class MyDataset(Dataset):
26
        def __init__(self, txt, transform=None, target_transform=None):
27
            fh = open(txt, 'r')
28
            imgs = []
29
            for line in fh:
30
                line = line.strip('\n')
31
                line = line.rstrip()
32
                words = line.split()
                imgs.append((words[0], int(words[1])))
33
            self.imgs = imgs
34
            self.transform = transform
35
36
            self.target_transform = target_transform
37
        def __getitem__(self, index):
38
39
            fn, label = self.imgs[index]
40
            img = cv2.imread(fn, cv2.IMREAD_COLOR)
41
            if self.transform is not None:
                img = self.transform(img)
42
43
             return img, label
44
        def __len__(self):
45
46
            return len(self.imgs)
```

## 2、代码分析

## 1, image\_list

```
if os.path.isdir(os.path.join(imageRoot, filename)):
```

若在路径中该文件存在, 执行下面内容

遍历文件,os.path.splitext用于分离文件名和拓展名,使用name和ext来储存图片名和后缀。在本案例中,文件均为jpg格式,故ext==".jpg",所以使用 ext = ext[1:]去除"."

再进行判断,若拓展名为"jpg""png"或者"bmp",将文件路径、文件名、图片名和标签写入txt文件中

## 2、shuffle\_split

```
with open(listfile, 'r') as f:
records = f.readlines()
random.shuffle(records)
```

先读取txt文件中每行的内容,将其打乱

```
num = len(records)
trainNum = int(num * 0.8)
with open(trainfile, 'w') as f:
f.writelines(records[0:trainNum])
with open(valfile, 'w') as f1:
f1.writelines(records[trainNum:])
```

计算列表中的行数(即文件数量),前80%写入训练集的txt文件,后20%写入验证集的txt文件 因为文件在前面已经打乱,故在这一步也相当于随机划分

## 3. MyDataset

```
for line in fh:
line = line.strip('\n')
line = line.rstrip()
words = line.split()
imgs.append((words[0], int(words[1])))
```

使用strip () 函数去除首尾指定字符

使用rsetrip()函数去除末尾指定字符,缺省为去除空格

使用split ()函数以输入实参为分割线进行分割,缺省为所有空字符,包括空格、换行和制表符 所以最后word[0]为图片路径,word[1]为标签(即0-9)

```
fn, label = self.imgs[index]
img = cv2.imread(fn, cv2.IMREAD_COLOR)
if self.transform is not None:
img = self.transform(img)
return img, label
```

使用fn和label分别储存图片路径和标签,使用imread读取出来(cv2.IMREAD\_COLOR为默认参数,读入一副彩色图片,忽略alpha通道)

返回值为图像和标签

# 3、训练代码

## 1、代码总览

```
def train():
1
2
        os.makedirs('./output', exist_ok=True)
3
        if True: #not os.path.exists('output/total.txt'):
            ml.image_list(args.datapath, 'output/total.txt')
4
5
            ml.shuffle_split('output/total.txt', 'output/train.txt',
    'output/val.txt')
6
7
        train_data = ml.MyDataset(txt='output/train.txt',
                                         transform=transforms.ToTensor())
8
        val_data = ml.MyDataset(txt='output/val.txt',
    transform=transforms.ToTensor())
9
        train_loader = DataLoader(dataset=train_data,
    batch_size=args.batch_size,
     shuffle=True)
10
        val_loader = DataLoader(dataset=val_data, batch_size=args.batch_size)
11
12
        model = Net()
        model.to(torch.device("cuda:0"))
13
14
        #model = models.resnet18(num_classes=10) # 调用内置模型
        #model.load_state_dict(torch.load('./output/params_10.pth'))
15
16
        #from torchsummary import summary
17
        #summary(model, (3, 28, 28))
18
        if args.cuda:
19
20
            print('training with cuda')
21
            model.cuda()
22
        optimizer = torch.optim.Adam(model.parameters(), 1r=0.01,
    weight_decay=1e-3)
        scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, [10, 20],
23
    0.1)
24
        loss_func = nn.CrossEntropyLoss()
25
26
        for epoch in range(args.epochs):
            # training------
27
28
           model.train()
29
            train_loss = 0
30
           train_acc = 0
            # for batch, (batch_x, batch_y) in enumerate(train_loader):
31
            for (batch_x, batch_y) in tqdm(train_loader,
32
```

```
desc= 'Epoch:' + str(epoch+1)
33
    + '/' +
    str(args.epochs),
34
                                                   colour='Green'):
35
                if args.cuda:
                    batch_x, batch_y = Variable(batch_x.cuda()),
36
    Variable(batch_y.cuda())
                else:
37
                    batch_x, batch_y = Variable(batch_x), Variable(batch_y)
38
39
                out = model(batch_x) # 256x3x28x28 out 256x10
                loss = loss_func(out, batch_y)
40
41
                train_loss += loss.item()
                pred = torch.max(out, 1)[1]
42
43
                train_correct = (pred == batch_y).sum()
44
                train_acc += train_correct.item()
45
46
                # print('epoch: %2d/%d batch %3d/%d Train Loss: %.3f, Acc:
    %.3f'
                        % (epoch + 1, args.epochs, batch,
47
    math.ceil(len(train_data) / args.batch_size),
48
                           loss.item(), train_correct.item() / len(batch_x)))
49
50
                optimizer.zero_grad()
51
                loss.backward()
                optimizer.step()
53
                scheduler.step() # 更新learning rate
            print('Train Loss: %.6f, Acc: %.3f' % (train_loss /
54
    (math.ceil(len(train_data)/args.batch_size)),
55
                                                   train_acc /
    (len(train_data))))
56
57
            # evaluation-----
            model.eval()
58
59
            eval_loss = 0
60
            eval_acc = 0
61
            # for batch_x, batch_y in val_loader:
            for batch_x, batch_y in tqdm(val_loader,
62
63
                                         desc='Epoch:' + str(epoch + 1) + '/' +
    str(args.epochs),
64
                                         colour='Green'):
                if args.cuda:
65
                    batch_x, batch_y = Variable(batch_x.cuda()),
66
    Variable(batch_y.cuda())
67
                else:
68
                    batch_x, batch_y = Variable(batch_x), Variable(batch_y)
69
                out = model(batch_x)
70
71
                loss = loss_func(out, batch_y)
72
                eval_loss += loss.item()
73
                pred = torch.max(out, 1)[1]
74
                num_correct = (pred == batch_y).sum()
75
                eval_acc += num_correct.item()
76
            print('Val Loss: %.6f, Acc: %.3f' % (eval_loss /
    (math.ceil(len(val_data)/args.batch_size)),
77
                                                 eval_acc / (len(val_data))))
78
            # save model -----
79
            if (epoch + 1) \% 1 == 0:
                # torch.save(model, 'output/model_' + str(epoch+1) + '.pth')
80
```

## 2、代码分析

```
1  ml.image_list(args.datapath, 'output/total.txt')
2  ml.shuffle_split('output/total.txt', 'output/train.txt', 'output/val.txt')
```

image\_list () 函数用来读取路径下的文件

shuffle\_split () 函数用来将图片按 4: 1 的比例划分为训练集和验证集

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=1e-3)
scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, [10, 20], 0.1)
loss_func = nn.CrossEntropyLoss()
```

第一条设置了优化器函数和学习率等参数

第二条是学习率更新,含义为在第10和第20个epochs时,执行lr = 0.1\*lr

第三条是设置损失函数, 即交叉熵函数

```
for (batch_x, batch_y) in tqdm(train_loader,
desc= 'Epoch:' + str(epoch+1) + '/' + str(args.epochs),
colour='Green'):
```

深度学习中常用的进度可视化函数,用以取代

```
1 | for batch, (batch_x, batch_y) in enumerate(train_loader):
```

同时依照上面介绍的MyDataset,这段代码中的batch\_x为图像,batch\_y为标签

```
out = model(batch_x) # 256x3x28x28 out 256x10
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()
```

将图片输入进行预测,然后通过损失函数计算出损失进行累加

得到的这个out是一个256×10的张量,第二个维度中10个数的和为1,可以理解为0-9各个数字的概率 max函数会返回out[1]中的最大值和最大值索引,我们只需要索引(即0-9),所以在后面加一个[1] 下面两条是累计正确率,若预测结果与标签一致,记为正确

```
# evaluation-----
model.eval()
```

这一段是在验证集上进行验证,故先将模型切换到验证模式,因为不需要进行反向传播,所以可以考虑在前面添加一行with torch.no\_grad():

```
1  # save model ------
2     if (epoch + 1) % 1 == 0:
3          # torch.save(model, 'output/model_' + str(epoch+1) + '.pth')
4          torch.save(model.state_dict(), 'output/params_' + str(epoch + 1)
          + '.pth')
```

这一段是每执行一个epoch保存一次,过于消耗空间,可以进行修改

# 4、代码修改

## 1、代码总览

```
1 import torch
   import math
   import torch.nn as nn
   from torch.autograd import Variable
   from torchvision import transforms, models
   import argparse
    import os
8
   from torch.utils.data import DataLoader
9
   from tgdm import tgdm
10
11 from dataloader import mnist_loader as ml
12
   from models.cnn import Net
13
   # from toonnx import to_onnx
14
15
16
   parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--datapath', required=True, help='data path')
18
    parser.add_argument('--batch_size', type=int, default=1024, help='training
    batch size')
19
    parser.add_argument('--epochs', type=int, default=200, help='number of
    epochs to train')
    parser.add_argument('--use_cuda', default=False, help='using CUDA for
20
    training')
21
22
   args = parser.parse_args()
23
   # args.cuda = args.use_cuda and torch.cuda.is_available()
24
    args.cuda = True
25
   if args.cuda:
        torch.backends.cudnn.benchmark = True
26
27
28
   def train():
29
30
        os.makedirs('./output', exist_ok=True)
31
        if True: #not os.path.exists('output/total.txt'):
32
            ml.image_list(args.datapath, 'output/total.txt')
            ml.shuffle_split('output/total.txt', 'output/train.txt',
33
    'output/val.txt')
34
35
        train_data = ml.MyDataset(txt='output/train.txt',
    transform=transforms.ToTensor())
```

```
36
        val_data = ml.MyDataset(txt='output/val.txt',
    transform=transforms.ToTensor())
        train_loader = DataLoader(dataset=train_data,
37
    batch_size=args.batch_size, shuffle=True, num_workers=4,)
38
        val_loader = DataLoader(dataset=val_data, batch_size=args.batch_size,
    num_workers=4,)
39
40
        model = Net()
        # model.load_state_dict(torch.load('output/params_mnist.pth'))
41
42
         model.to(torch.device("cuda:0"))
        #model = models.resnet18(num_classes=10) # 调用内置模型
43
44
        #model.load_state_dict(torch.load('./output/params_10.pth'))
45
        #from torchsummary import summary
46
        #summary(model, (3, 28, 28))
47
        if args.cuda:
48
49
            print('training with cuda')
50
            model.cuda()
51
        optimizer = torch.optim.Adam(model.parameters(), lr=0.01,
    weight_decay=1e-3)
        # scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, [10, 20],
52
    0.1)
53
        loss_func = nn.CrossEntropyLoss()
54
        best_loss = 1000000
55
        no\_optim = 0
56
57
        for epoch in range(args.epochs):
            # training-----
58
59
            model.train()
60
            train_loss = 0
61
            train_acc = 0
            # for batch, (batch_x, batch_y) in enumerate(train_loader):
62
63
            for (batch_x, batch_y) in tqdm(train_loader,
64
                                                   desc= 'Epoch:' +
    str(epoch+1) + '/' + str(args.epochs),
65
                                                   colour='Green'):
66
                if args.cuda:
67
                    batch_x, batch_y = Variable(batch_x.cuda()),
    Variable(batch_y.cuda())
68
                else:
                    batch_x, batch_y = Variable(batch_x), Variable(batch_y)
69
                out = model(batch_x) # 256x3x28x28 out 256x10
70
71
                loss = loss_func(out, batch_y)
72
                train_loss += loss.item()
73
                pred = torch.max(out, 1)[1]
74
                train_correct = (pred == batch_y).sum()
75
                train_acc += train_correct.item()
76
77
                # print('epoch: %2d/%d batch %3d/%d Train Loss: %.3f, Acc:
    %.3f'
78
                        % (epoch + 1, args.epochs, batch,
    math.ceil(len(train_data) / args.batch_size),
79
                           loss.item(), train_correct.item() / len(batch_x)))
80
81
                optimizer.zero_grad()
82
                loss.backward()
83
                optimizer.step()
84
            # scheduler.step() # 更新learning rate
```

```
print('Train Loss: %.6f, Acc: %.3f' % (train_loss /
 85
     (math.ceil(len(train_data)/args.batch_size)),
 86
                                                    train_acc /
     (len(train_data))))
 87
             if loss >= best_loss:
 88
                 no\_optim += 1
 89
             else:
 90
                 no\_optim = 0
                 best_loss = loss
 91
 92
                 torch.save(model.state_dict(), 'output/params_mnist' + '.pth')
 93
             if no_optim >= 3:
 94
                 model.load_state_dict(torch.load('output/params_mnist.pth'))
 95
                 torch.optim.lr_scheduler.MultiStepLR(optimizer, [(epoch + 1)],
     0.1)
 96
                 print(("Learn rate changed!!!Best Loss is
     {}!").format(best_loss))
 97
             if no_optim > 6:
 98
                 print('early stop at %d epoch' % (epoch + 1))
99
100
             # evaluation-----
101
             with torch.no_grad():
102
103
                 model.eval()
104
                 eval_loss = 0
105
                 eval_acc = 0
106
                 # for batch_x, batch_y in val_loader:
107
108
                 for batch_x, batch_y in tqdm(val_loader,
109
                                              desc='Epoch:' + str(epoch + 1) +
     '/' + str(args.epochs),
110
                                              colour='Green'):
111
                     if args.cuda:
112
                         batch_x, batch_y = Variable(batch_x.cuda()),
     Variable(batch_y.cuda())
113
                     else:
114
                         batch_x, batch_y = Variable(batch_x), Variable(batch_y)
115
116
                     out = model(batch_x)
117
                     loss = loss_func(out, batch_y)
118
                     eval_loss += loss.item()
119
                     pred = torch.max(out, 1)[1]
120
                     num_correct = (pred == batch_y).sum()
121
                     eval_acc += num_correct.item()
122
                 print('Val Loss: %.6f, Acc: %.3f' % (eval_loss /
     (math.ceil(len(val_data)/args.batch_size)),
123
                                                      eval_acc /
     (len(val_data))))
124
             # save model -----
             # if (epoch + 1) % 1 == 0:
125
126
             #
                   # torch.save(model, 'output/model_' + str(epoch+1) + '.pth')
127
                   torch.save(model.state_dict(), 'output/params_' + str(epoch +
     1) + '.pth')
128
                 #to_onnx(model, 3, 28, 28, 'params.onnx')
129
130
     if __name__ == '__main__':
131
         train()
132
```

## 2、主要修改

### 1、数据加载

```
train_loader = DataLoader(dataset=train_data, batch_size=args.batch_size,
shuffle=True)
val_loader = DataLoader(dataset=val_data, batch_size=args.batch_size)
```

改为

```
train_loader = DataLoader(dataset=train_data, batch_size=args.batch_size,
shuffle=True, num_workers=4)
val_loader = DataLoader(dataset=val_data, batch_size=args.batch_size,
num_workers=4)
```

加快了数据读取速度

### 2、模型保存

```
1  # save model ------
2     if (epoch + 1) % 1 == 0:
3          # torch.save(model, 'output/model_' + str(epoch+1) + '.pth')
4          torch.save(model.state_dict(), 'output/params_' + str(epoch + 1) + '.pth')
```

改为

```
if loss >= best_loss:
    no_optim += 1

else:
    no_optim = 0
    best_loss = loss
    torch.save(model.state_dict(), 'output/params_mnist' + '.pth')
```

首先定义一个最小损失,当这个epoch的损失小于最小损失时,保存模型同时更新最小损失这样只需要保存一个模型,同时能够保证保存的模型为最优模型

### 3、学习率更新

```
1 | scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, [10, 20], 0.1)
```

改为

当连续四次模型损失值未下降时,认为在当前学习率下模型已收敛,此时加载最优模型,同时更新学习率为原来的0.1

### 4、停止训练

添加

当连续8次损失未下降时,认为此时模型已经收敛,直接退出训练

# 5、测试准确度

## 1、代码总览

### 1、单张图片测试(源代码)(不做分析)

```
1 import torch
 2 from PIL.Image import Image
 3 from torch.utils.data import DataLoader
   import torch.utils.data as Data
5
   import cv2
 6 from torch.autograd import Variable
7
   from torchvision import transforms
8 from models.cnn import Net
9
   from toonnx import to_onnx
10 | from tqdm import tqdm
11
   from PIL import Image
12
   import os
13
   from dataloader import mnist_loader as ml
14
    import argparse
15
16 | use_cuda = True
17
    model = Net()
    model.load_state_dict(torch.load('output/params_30.pth'))
18
19
    model = torch.load('output/model.pth')
20
    model.eval()
    if use_cuda and torch.cuda.is_available():
21
       model.cuda()
22
23
24
    to_onnx(model, 3, 28, 28, 'output/params.onnx')
25
26
    img = cv2.imread('4_00440.jpg')
27
    img_tensor = transforms.ToTensor()(img)
28
    img_tensor = img_tensor.unsqueeze(0)
29
    if use_cuda and torch.cuda.is_available():
        prediction = model(Variable(img_tensor.cuda()))
30
31
    else:
       prediction = model(Variable(img_tensor))
32
33 | pred = torch.max(prediction, 1)[1]
34
   print(pred)
35 cv2.imshow("image", img)
36 cv2.waitKey(0)
```

### 2、多张图片测试(自行编写)

```
import torch
 2
    from PIL.Image import Image
    from torch.utils.data import DataLoader
 4
    import torch.utils.data as Data
    import cv2
 5
    from torch.autograd import Variable
 7
    from torchvision import transforms
 8
    from models.cnn import Net
9
    from toonnx import to_onnx
10
    from tqdm import tqdm
11
    from PIL import Image
12
    import os
13
    from dataloader import mnist_loader as ml
    import argparse
14
15
16
    def My_loader (Id, root):
        img = cv2.imread(os.path.join(root, 'test_images/{}').format(Id[0]),
17
18
                          cv2.IMREAD_COLOR
19
                          )
20
        label = Id[1]
21
        # img = transforms(id)
22
        transform = transforms.Compose([
23
            transforms.ToTensor()
24
        ])
25
        img = transform(img)
26
27
        return img, label
28
29
30
    class ImageFolder(Data.Dataset):
31
        def __init__(self, testlist, root):
32
33
            self.ids1 = testlist[0]
34
            self.ids2 = testlist[1]
35
            self.loader = My_loader
36
            self.root = root
37
38
        def __getitem__(self,index):
39
            Id = [self.ids1[index], self.ids2[index]]
            img, label = self.loader(Id, self.root)
40
41
42
            return img, label
43
        def __len__(self):
44
            return len(list(self.ids1))
45
46
47
48
    if __name__ == '__main__':
49
50
        Root = 'E:/python/DeepLearning/MNIST_Dataset/'
51
        img_list = os.listdir(os.path.join(Root, 'test_images'))
52
        label_list = []
53
        for i in range(len(img_list)):
54
            label_list.append(int(img_list[i][0]))
55
```

```
56
        Id = [img_list, label_list]
57
58
        dataset = ImageFolder(Id, Root)
59
        test_loader = torch.utils.data.DataLoader(
60
            dataset,
            batch_size=1024,
61
62
            shuffle=True,
63
            num_workers=4,
            drop_last=False)
64
65
        model = Net()
66
67
        model.load_state_dict(torch.load('output/params_mnist.pth'))
68
        model.eval()
        model.cuda()
69
70
        to_onnx(model, 3, 28, 28, 'output/params_mnist.onnx')
71
72
73
        with torch.no_grad():
74
            model.eval()
75
            eval_acc = 0
76
77
            # for batch_x, batch_y in val_loader:
78
            for batch_x, batch_y in tqdm(test_loader,
79
                                          desc='process',
80
                                           colour='Green'):
81
                batch_x, batch_y = Variable(batch_x.cuda()),
    Variable(batch_y.cuda())
82
                out = model(batch_x)
83
                pred = torch.max(out, 1)[1]
85
                num_correct = (pred == batch_y).sum()
86
                eval_acc += num_correct.item()
87
88
        print("Acc: %.3f" % (eval_acc / (len(dataset))))
```

## 2、代码分析

```
1
    def My_loader (Id, root):
 2
        img = cv2.imread(os.path.join(root, 'test_images/{}').format(Id[0]),
                          cv2.IMREAD_COLOR
 3
 4
                          )
 5
        label = Id[1]
        # img = transforms(id)
 6
 7
        transform = transforms.Compose([
8
            transforms.ToTensor()
9
        ])
10
11
        img = transform(img)
12
        return img, label
```

首先在这里的Id由两部分组成,Id[0]是文件名+拓展名,Id[1]是对应标签

所以直接利用Id[0]读出img,然后把Id[1]赋给label

将img转为张量,返回img和label

```
1 class ImageFolder(Data.Dataset):
```

```
3
        def __init__(self, testlist, root):
 4
            self.ids1 = testlist[0]
            self.ids2 = testlist[1]
 5
 6
            self.loader = My_loader
 7
            self.root = root
 8
        def __getitem__(self,index):
9
10
            Id = [self.ids1[index], self.ids2[index]]
11
            img, label = self.loader(Id, self.root)
12
13
            return img, label
14
15
        def __len__(self):
            return len(list(self.ids1))
```

这里的testlist我们传入之前的Id, 所以这个ids1就是图片名, ids2是标签

在第二个函数里面我们将图片和标签一张张取出来

```
1
     Root = 'E:/python/DeepLearning/MNIST_Dataset/'
 2
        img_list = os.listdir(os.path.join(Root, 'test_images'))
 3
        label_list = []
        for i in range(len(img_list)):
 4
 5
            label_list.append(int(img_list[i][0]))
 6
 7
        Id = [img_list, label_list]
 8
        dataset = ImageFolder(Id, Root)
 9
10
        test_loader = torch.utils.data.DataLoader(
11
            dataset,
12
            batch_size=1024,
13
            shuffle=True,
14
            num_workers=4,
15
            drop_last=False)
```

img\_list用来获取文件夹中所有文件名,也就是所有测试图片名

然后把所有测试图片名的第一个数字(标签)存入label\_list

把img\_list和label\_list组成ld

后面不多赘述