同济大学
计算机科学与技术系

数据挖掘课程实验报告

学　　号　1752919

姓　　名　祁好雨

专　　业　计算机科学与技术

授课老师　向阳

二〇二〇年十一月

一、问题描述

以 MNIST 手写数据集划分训练集和验证集，将自己手写拍照上传的数字图片作为测试样本，训练卷积神经网络分类器识别 0-9 的手写数字。

二、数据集说明

MNIST 手写数字数据集包含 60000 张用于训练的手写数字图片和 10000 张用于测试的手写数字图片。所有的图片具有相同的尺寸（28x28 pixels），数字均位于图片中央。

数据集链接地址如下：http://yann.lecun.com/exdb/mnist/。

原训练集各类别分布和示例样本如下图所示：

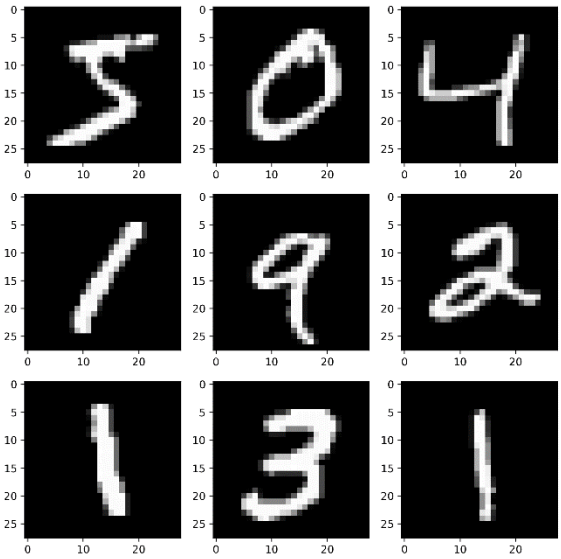


图 1 训练集图片示例

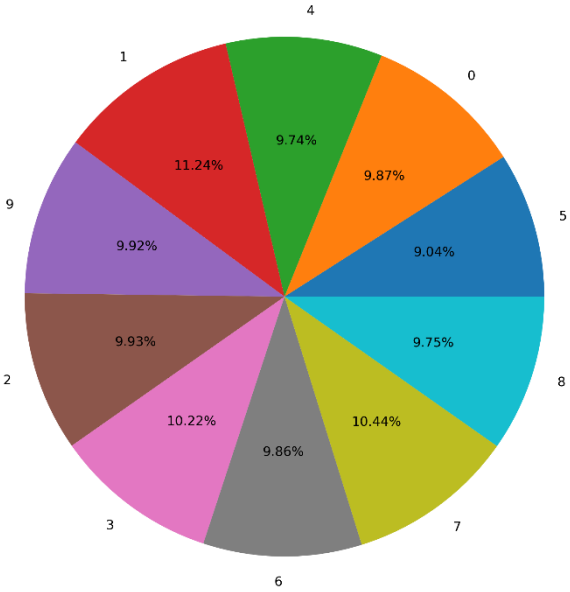


图 2 训练集样本分布

在本实验中，原数据集中训练集被划分为训练集和验证集，其中验证集占比 0.2，训练集占比 0.8。原数据集的测试集作为测试集使用，用于评估模型性能。

在上述的分布图中不难发现，原数据集中的训练集基本上是类别均衡的；为了不破坏原数据集的类别均衡性，在划分训练集和验证集时，采用分类别抽样(stratified sampling)划分的方法。

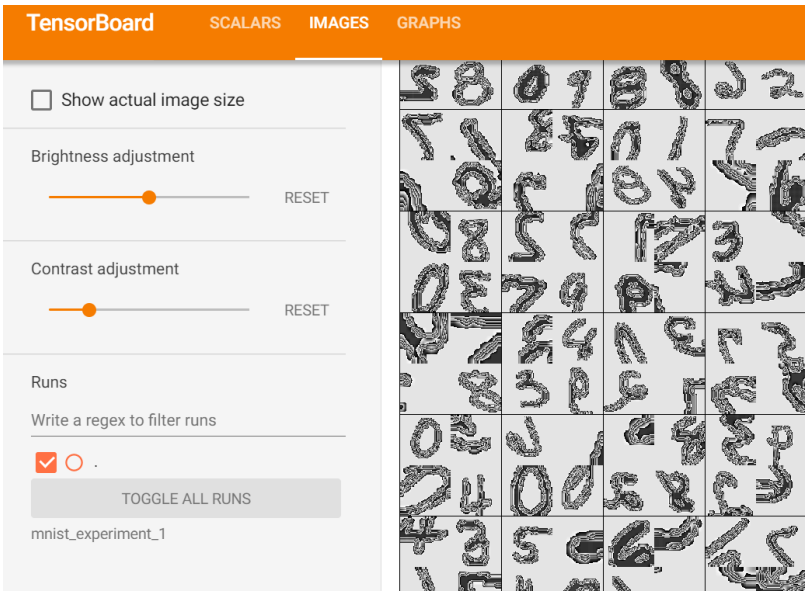


图 3 TensorBoard makegrid 看到的输入图片

三、算法代码

算法概述：

模型架构：resnet18 修改了第一个卷积层使其接受 BW 格式图片；修改了 fc 层使其输出 10 个类别的概率分布。未使用预先训练的权重。

数据处理：图片转换为 Tensor；按照 imagenet 图片第一个通道的均值和方差标准化处理。

权重更新：采用带动量的随机梯度下降法，初始学习率为 1e-3,动量参数为 0.9;学习率每 7 个 epoch 缩小为 0.1 倍

损失计算：softmax + crossentropy loss

其他设置：batch_size=128, epoch_num = 24

使用说明：

训练模块设置了五个命令行参数，用户可自行决定数据文件夹位置，日志存放位置,模型保存位置,模型接受图片大小,训练集验证集的划分比例。

第三方包安装：

Pytorch
Tensorboard
Sk-learn
Numpy
Matplotlib
Seaborn(代码中没用上)

1）导入库

```
# import necessary packages
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import Dataset, DataLoader, TensorDataset, Subset
import torchvision
from torchvision import transforms,utils,models
import argparse
import seaborn as sns
from collections import Counter
import time
import copy
from torch.utils.tensorboard import SummaryWriter
from sklearn.metrics import confusion_matrix, accuracy_score
```

2）可视化数据模块

```
def dataVisualization(x_train,y_train,x_test,y_test):
    '''return null

    visualing first nine images
    save it as a image
    then visualize distribution of training set
    save it as a image
    '''
    plt.figure(figsize=(9, 9))
    for i in range(9):
        plt.subplot(331 + i)
        plt.imshow(x_train[i], cmap=plt.get_cmap('gray'))
    plt.savefig('sample.png', bbox_inches='tight', dpi=300)
    print(f'训练集的样本数：{x_train.shape[0]}，测试集的样本数：
{x_test.shape[0]}')
    print(f'输入图像的大小：{x_train.shape[1]}*{x_train.shape[2]}')
    y_train = np.array(y_train)
    label_cnt = Counter(y_train)  # 统计每个类别的样本数
    print('训练集的图像类别分布：', label_cnt)
    # 可视化图像类别分布
    plt.figure(figsize=(9,9))
    plt.pie(x=label_cnt.values(), labels=label_cnt.keys(), autopct='%.2
f%%')
    plt.savefig('label_distribution.png', bbox_inches='tight', dpi=300)
```

3）Stratified sampling

将原训练集分为训练集和验证集，先用自定义函数选择索引，再通过torch.utils.Subset函数划分训练集和验证集。

```
def stratified_sampling(ds,k):
    '''return trainset,validset

    sampling from ds while maintaining data balance
    acoss different classes.
    k is the number of samples in each class in trainset.
    '''
    class_counts = {}
    train_indices = []
    test_indices = []
    for i, (data, label) in enumerate(ds):
        c = label
        class_counts[c] = class_counts.get(c, 0) + 1
        if class_counts[c] <= k:
            train_indices.append(i)
        else:
            test_indices.append(i)

    print('stratified sampling done')
    return (train_indices,test_indices)
```

```
    # transform data and generate data loader to create batches
    raw_trainset = torchvision.datasets.MNIST(args.datasetPath, train=T
rue, download=False,
                                              transform=data_transforms['
train'])
    train_indices, valid_indices = stratified_sampling(trainset, 0.8*le
n(trainset)/10)
    trainset = Subset(raw_trainset,train_indices)
    validset = Subset(raw_trainset,valid_indices)
    train_data_loader = DataLoader(trainset,batch_size=128,shuffle=True
,drop_last=False)
    valid_data_loader = DataLoader(validset,batch_size=128,shuffle=Fals
e,drop_last=False)
```

4）构建网络模型并检查

```
def my_model():
    model_body = models.resnet18(pretrained=False)

    num_ftrs = model_body.fc.in_features
    # Here the size of each output sample is set to 2.
```

```
    # Alternatively, it can be generalized to nn.Linear(num_ftrs, len(c
lass_names)).
    model_body.fc = nn.Linear(num_ftrs, 10)

    # modify first conv to accept bw img
    # weight = copy.deepcopy(model_body.conv1.weight)
    # new_weight = (torch.sum(weight,dim = 1)/3).unsqueeze(1)

    model_body.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2,
2), padding=(3, 3), bias=False)
    # model_body.conv1.weight.data = new_weight

    return model_body
```

```
def check_model(trainloader,net):
    # get some random training images
    dataiter = iter(trainloader)
    images, labels = dataiter.next()
    images = images.to(device)
    labels = labels.to(device)

    # create grid of images
    img_grid = torchvision.utils.make_grid(images)

    # write to tensorboard
    writer.add_image('four_mnist_images', img_grid)

    writer.add_graph(net, images)
    writer.close()
```

5）训练模型

```
def train_model(model, dataloaders, writer, criterion, optimizer, sched
uler, device, args, num_epochs=25, save=True):
    '''return model

    load input model as last best model, begin trainning for num_epochs
    '''
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    best_epoch = 0

    for epoch in range(num_epochs):
```

```
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            count = 0
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
                # print(inputs.shape)
                # print(labels.shape)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    # print(f"outputs done:{outputs.shape}")
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
                count += inputs.size(0)

            epoch_loss = running_loss / count
            epoch_acc = running_corrects.double() / count
```

```
            if phase == 'train':
                scheduler.step()
                # log the running loss
                writer.add_scalar('training loss',
                                  epoch_loss,
                                  epoch)
                writer.add_scalar('training acc',
                                  epoch_acc,
                                  epoch)

            if phase == 'val':
                # log the running loss
                writer.add_scalar('validation loss',
                                  epoch_loss,
                                  epoch)
                writer.add_scalar('validation acc',
                                  epoch_acc,
                                  epoch)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_epoch = epoch
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)

    if save:
        torch.save({
            'epoch': best_epoch,
            'model_state_dict': best_model_wts,
            'optimizer_state_dict': optimizer.state_dict(),
```

```
            'acc': best_acc,
            }, args.SaveModel)
    return model
```

```
# instantiate network(body:resnet + head:fc)
    model = my_model()
    model = model.to(device)
    check_model(train_data_loader,model)

    criterion = nn.CrossEntropyLoss()

    # Observe that all parameters are being optimized
    optimizer_ft = optim.SGD(model.parameters(), lr=0.001, momentum=0.9
)

    # Decay LR by a factor of 0.1 every 7 epochs
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, g
amma=0.1)

    model = train_model(model, dataloaders, writer, criterion, optimize
r_ft, exp_lr_scheduler,
                        device, args, num_epochs=25,save=True)
```

6）测试模型

```
def test(model,test_loader,device):
    # set model to test mode
    model.eval()
    test_acc = 0.0
    count = 0

    # Initialize the prediction and label lists(tensors)
    predlist=torch.zeros(0,dtype=torch.long, device='cpu')
    lbllist=torch.zeros(0,dtype=torch.long, device='cpu')

    for i, (images, labels) in enumerate(test_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Predict classes using images from the test set
        outputs = model(images)
        _, prediction = torch.max(outputs.data, 1)

        test_acc += torch.sum(prediction == labels.data).float()
        count += images.shape[0]
```

```
        # Append batch prediction results
        predlist=torch.cat([predlist, prediction.view(-1).cpu()])
        lbllist=torch.cat([lbllist, labels.view(-1).cpu()])

    test_acc = test_acc / count
    print(f"total test accuracy:{test_acc} over {count} test samples")

    # Confusion matrix
    conf_mat=confusion_matrix(lbllist.numpy(), predlist.numpy())
    writer.add_image('confusion_matrix',conf_mat)

    # Per-class accuracy
    accuracy = accuracy_score(lbllist, predlist)
    return test_acc
```

7）整体流程

```
if __name__ == '__main__':
    # argparse for using this code base
    parser = argparse.ArgumentParser(description="Train a network for M
NIST dataset,  \
                                            please make sure your compu
ter has a GPU")

    parser.add_argument("datasetPath", help="input path where your data
sets are",
                        type=str, nargs='?', default='/home/432/qihaoyu/dat
a/MNIST')

    parser.add_argument("logDir", help="input dir where your logs will
be recorded", \
                        type=str, nargs='?', default='/home/432/qihaoyu/vsc
ode_workspace/homework/mnist_experiment_1')

    parser.add_argument('SaveModel', help="input path where your model
will be saved", \
                        type=str,nargs='?',default='/home/432/qihaoyu/vscod
e_workspace/homework/models')
    parser.add_argument("img_size", help="standard image size to feed i
nto the network",
                        type=int, nargs='?', default=224)

    parser.add_argument("splits", help="ratio for trainning samples in
MNIST trainset, \
```

```
                            remaining samples will be treated as valids
et", type=float,
                            nargs='?', default=0.8)

    args = parser.parse_args()

    # transformation
    data_transforms = {
    'train': transforms.Compose([
        # transforms.RandomResizedCrop(224),
        # transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485], [0.229])
    ]),
    'test': transforms.Compose([
        # transforms.Resize(256),
        # transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485], [0.229])
    ])
    }

    # create tensorboard writer
    writer = SummaryWriter(args.logDir)

    # get pytorch datasets
    # if there's no existing datasets in the root directory, you should
 download it first
    trainset = torchvision.datasets.MNIST(args.datasetPath, train=True,
 download=False)
    testset = torchvision.datasets.MNIST(args.datasetPath, train=False,
 download=False)

    x_train = trainset.train_data
    y_train = trainset.train_labels
    x_test = testset.test_data
    y_test = testset.test_labels

    # visualize data and distribution according to instructions in ppt
    dataVisualization(x_train, y_train,x_test,y_test)

    # transform data and generate data loader to create batches
    raw_trainset = torchvision.datasets.MNIST(args.datasetPath, train=T
rue, download=False,
```

```
                                            transform=data_transforms['
train'])
    train_indices, valid_indices = stratified_sampling(trainset, 0.8*le
n(trainset)/10)
    trainset = Subset(raw_trainset,train_indices)
    validset = Subset(raw_trainset,valid_indices)
    train_data_loader = DataLoader(trainset,batch_size=128,shuffle=True
,drop_last=False)
    valid_data_loader = DataLoader(validset,batch_size=128,shuffle=Fals
e,drop_last=False)

    testset = torchvision.datasets.MNIST(args.datasetPath, train=False,
download=False,
                                            transform=data_transforms['
test'])
    test_data_loader = DataLoader(testset,batch_size=128,shuffle=False,
drop_last=False)

    # generate dataloaders dict
    dataloaders = {'train':train_data_loader, 'val':valid_data_loader}
    dataset_sizes = {'train':len(trainset), 'val':len(validset)}

    # if running on GPU and we want to use cuda move model there
    device = torch.device("cuda:0" if torch.cuda.is_available() else "c
pu")


    # instantiate network(body:resnet + head:fc)
    model = my_model()
    model = model.to(device)
    check_model(train_data_loader,model)

    criterion = nn.CrossEntropyLoss()

    # Observe that all parameters are being optimized
    optimizer_ft = optim.SGD(model.parameters(), lr=0.001, momentum=0.9
)

    # Decay LR by a factor of 0.1 every 7 epochs
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, g
amma=0.1)

    model = train_model(model, dataloaders, writer, criterion, optimize
r_ft, exp_lr_scheduler,
                        device, args, num_epochs=25,save=True)
```

```
# test model
test(model,test_data_loader,device)
```

四、实验分析

实验结果：

在测试集上的准确率为：0.9863999485969543

Confusion matrix 如下所示：

可以看出模型最易混淆的为数字 4 和数字 9

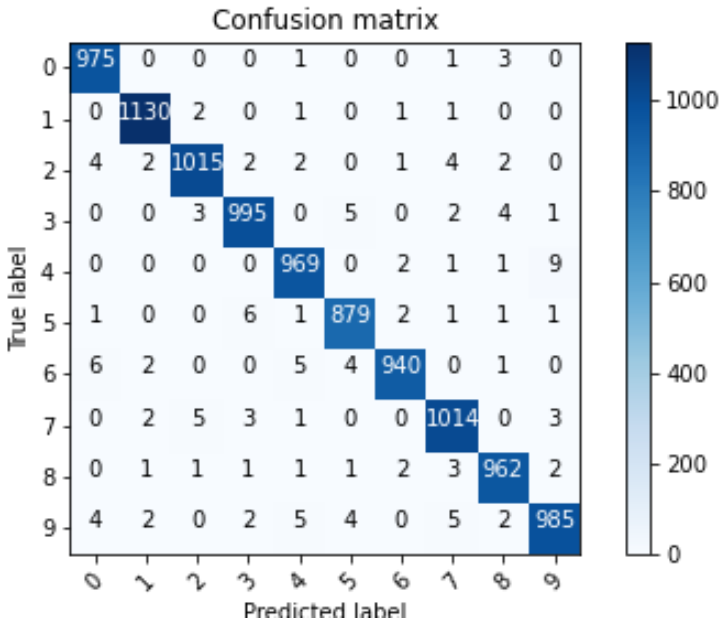


图 4 测试集 confusion matrix

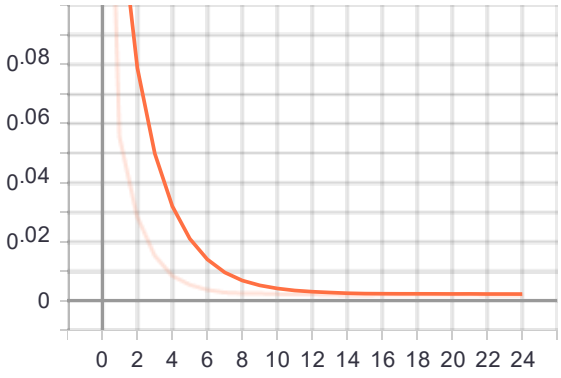


图 5 训练 loss

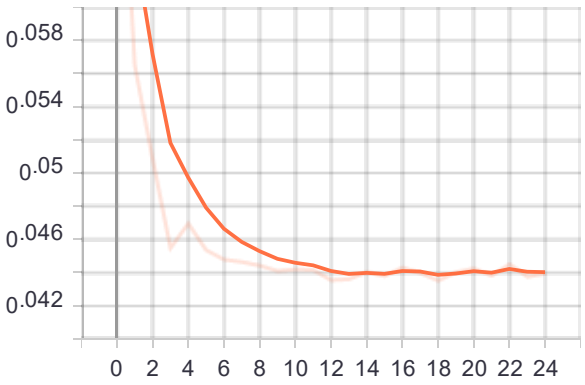


图 6 验证 loss

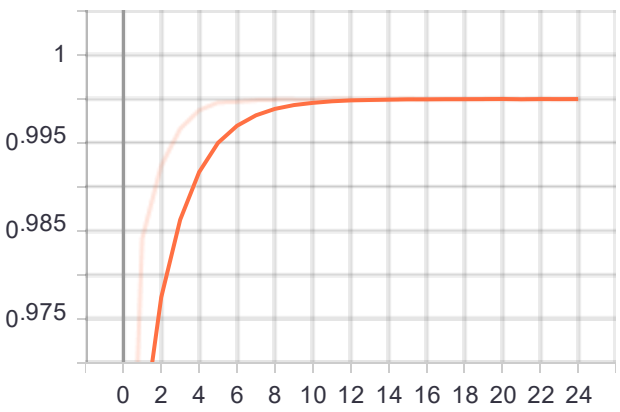


图 7 训练准确率

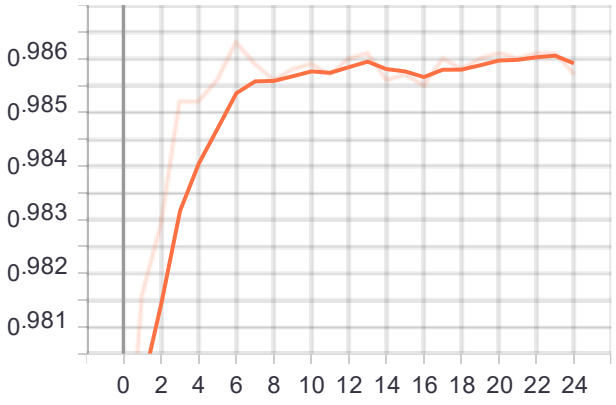


图 8 验证准确率

从上述训练集的损失和准确率变化可以看出模型在训练集上存在一定的过拟合现象，后续实验可以通过加 dropout 或者其他正则化手段来避免太大程度的过拟合。

训练过程如下，在下述 epoch 中选择 validset 上准确率最高的 epoch 对应的模型权重作为最终模型。

```
Epoch 0/24
----------
train Loss: 0.2946 Acc: 0.9147
val Loss: 0.0904 Acc: 0.9732

Epoch 1/24
----------
train Loss: 0.0623 Acc: 0.9824
val Loss: 0.0672 Acc: 0.9799

Epoch 2/24
----------
train Loss: 0.0322 Acc: 0.9916
val Loss: 0.0611 Acc: 0.9828

Epoch 3/24
----------
train Loss: 0.0165 Acc: 0.9967
val Loss: 0.0589 Acc: 0.9812

Epoch 4/24
----------
train Loss: 0.0092 Acc: 0.9990
val Loss: 0.0566 Acc: 0.9830

Epoch 5/24
```

```
----------
train Loss: 0.0063 Acc: 0.9995
val Loss: 0.0561 Acc: 0.9830

Epoch 6/24
----------
train Loss: 0.0043 Acc: 0.9997
val Loss: 0.0563 Acc: 0.9838

Epoch 7/24
----------
train Loss: 0.0033 Acc: 0.9999
val Loss: 0.0565 Acc: 0.9829

Epoch 8/24
----------
train Loss: 0.0032 Acc: 0.9999
val Loss: 0.0558 Acc: 0.9835

Epoch 9/24
----------
train Loss: 0.0029 Acc: 0.9999
val Loss: 0.0554 Acc: 0.9836

Epoch 10/24
----------
train Loss: 0.0028 Acc: 1.0000
val Loss: 0.0556 Acc: 0.9836

Epoch 11/24
----------
train Loss: 0.0028 Acc: 1.0000
val Loss: 0.0554 Acc: 0.9838

Epoch 12/24
----------
train Loss: 0.0026 Acc: 1.0000
val Loss: 0.0555 Acc: 0.9833

Epoch 13/24
----------
train Loss: 0.0026 Acc: 1.0000
val Loss: 0.0556 Acc: 0.9834
```

```
Epoch 14/24
----------
train Loss: 0.0027 Acc: 1.0000
val Loss: 0.0554 Acc: 0.9835

Epoch 15/24
----------
train Loss: 0.0025 Acc: 1.0000
val Loss: 0.0553 Acc: 0.9838

Epoch 16/24
----------
train Loss: 0.0026 Acc: 1.0000
val Loss: 0.0554 Acc: 0.9840

Epoch 17/24
----------
train Loss: 0.0027 Acc: 1.0000
val Loss: 0.0560 Acc: 0.9835

Epoch 18/24
----------
train Loss: 0.0025 Acc: 1.0000
val Loss: 0.0550 Acc: 0.9842

Epoch 19/24
----------
train Loss: 0.0026 Acc: 0.9999
val Loss: 0.0556 Acc: 0.9831

Epoch 20/24
----------
train Loss: 0.0025 Acc: 1.0000
val Loss: 0.0556 Acc: 0.9841

Epoch 21/24
----------
train Loss: 0.0025 Acc: 1.0000
val Loss: 0.0559 Acc: 0.9837

Epoch 22/24
----------
train Loss: 0.0026 Acc: 1.0000
val Loss: 0.0554 Acc: 0.9834
```

```
Epoch 23/24
----------
train Loss: 0.0025 Acc: 1.0000
val Loss: 0.0564 Acc: 0.9835

Epoch 24/24
----------
train Loss: 0.0024 Acc: 1.0000
val Loss: 0.0562 Acc: 0.9833

Training complete in 7m 36s
Best val Acc: 0.984250
total test accuracy:0.9863999485969543 over 10000 test samples
```

上传 10 张自己拍摄的手写数字照片，转换为灰度格式，resize 到 28x28 的像素，检测模型输出，代码如下：

```python
def predict_for_customized_data(model,dataPath,device):
    model.eval()
    for i in range(10):
        img1 = Image.open(dataPath/(str(i)+'.JPG')).convert('L')
        sized_img1 = transforms.Resize((28,28))(img1)
        tensor1 = transforms.ToTensor()(sized_img1)
        tensor1 = torch.unsqueeze(tensor1,0)
        assert tensor1.shape == (1,1,28,28)
        inputs = tensor1.to(device)
        outputs = model(inputs)
        print(outputs)
        _, prediction = torch.max(outputs.data, 1)
        print(prediction)
```

最终输出如下：

```
tensor([[ 0.0627,  3.3247, -1.5043, -2.4198, -0.0387,  0.9379, -0.0958,  1.3305,
         -0.1327, -2.4302]], device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
tensor([[-0.0849,  3.3900, -1.5465, -2.3464,  0.1393,  0.9960, -0.1491,  1.2198,
         -0.0301, -2.4716]], device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
tensor([[ 0.0470,  3.1663, -1.4129, -2.3464,  0.0404,  0.8320, -0.0912,  1.2537,
          0.0387, -2.3751]], device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
tensor([[ 0.0604,  3.3914, -1.4023, -2.2675,  0.0237,  0.8554, -0.2231,  1.2724,
         -0.1125, -2.4638]], device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
```

```
tensor([[ 1.4152e-01,  3.2076e+00, -1.4329e+00, -2.4117e+00, -1.3021e-01,
          8.3629e-01,  1.1261e-04,  1.3240e+00, -4.8039e-02, -2.4092e+00]],
       device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
tensor([[ 0.0408,  3.3281, -1.4669, -2.3937,  0.0214,  0.9613, -0.1727,  1.3236,
         -0.1440, -2.4152]], device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
tensor([[ 0.0352,  3.3936, -1.5284, -2.4327, -0.0555,  0.9108, -0.0615,  1.2695,
         -0.1061, -2.4949]], device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
tensor([[ 1.6809e-02,  3.3713e+00, -1.5663e+00, -2.3387e+00, -1.7898e-03,
          9.6868e-01, -1.6990e-01,  1.3084e+00, -1.1342e-01, -2.4470e+00]],
       device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
tensor([[-0.1321,  3.6067, -1.6129, -2.4619, -0.0661,  1.0501, -0.1779,  1.4918,
         -0.2713, -2.4176]], device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
tensor([[-0.1155,  3.4807, -1.5790, -2.3580,  0.0713,  1.0390, -0.1681,  1.3022,
         -0.1777, -2.4345]], device='cuda:0', grad_fn=<AddmmBackward>)
tensor([1], device='cuda:0')
```

模型将手写数字全部预测为 1，检查输入数据后发现这是因为手写的数字为笔画的地方暗而背景亮。训练集 MNIST 中的数据却是笔画的地方亮而背景暗，且背景为一致的黑色。

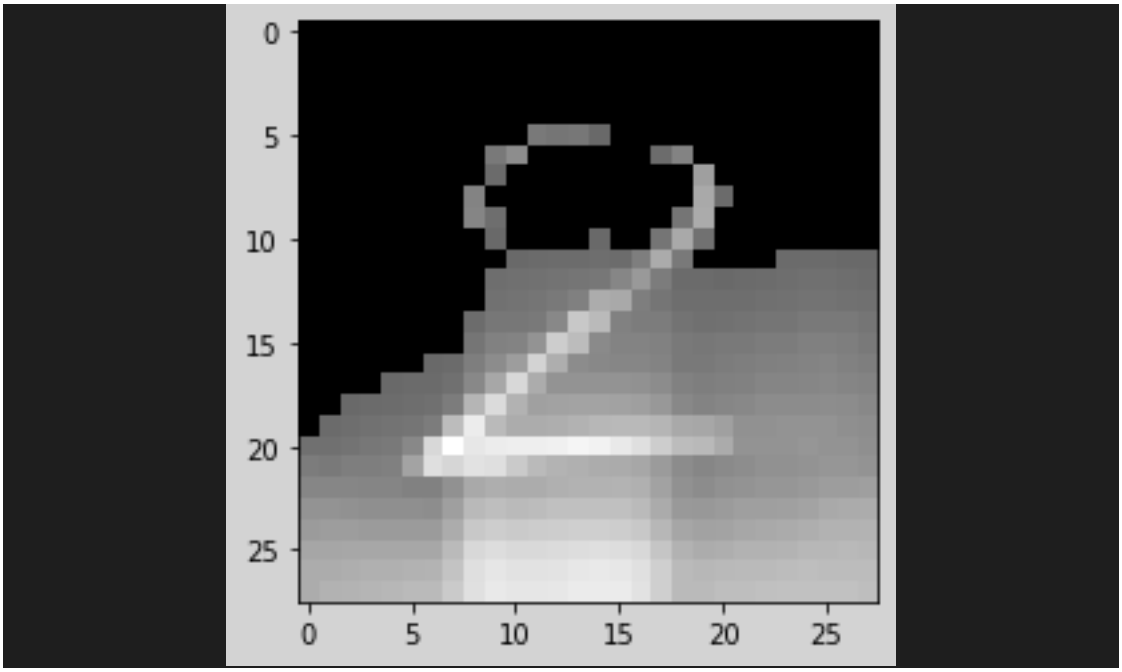
图 9 转为灰度图放缩后的图片

因此如果希望模型能够预测手写数字图片，我们需要对手写数字图片做一些处理。

下图是用像素值最大值 255 减去整张图片翻转明度后，对于小于 60 的值置 0 的结果，可以看到，由于拍摄照片时光线的因素，数字 2 的上半部分和下半部分对应的像素亮度不一致。

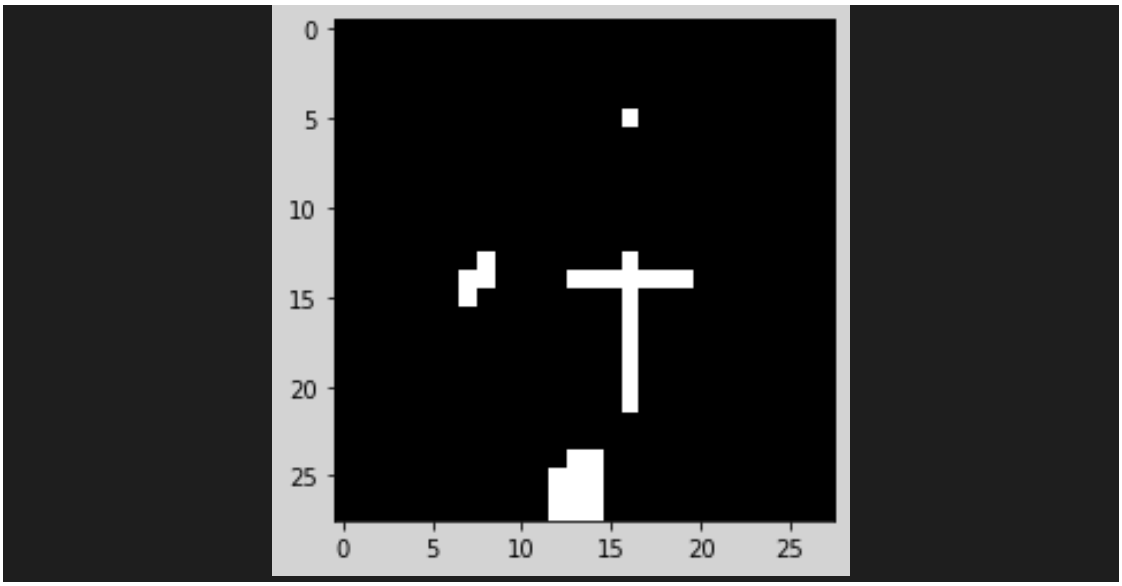
这样的处理方法得到模型的预测值为 tensor([1], device='cuda:0')

tensor([1], device='cuda:0')
tensor([6], device='cuda:0')
tensor([0], device='cuda:0')
tensor([6], device='cuda:0')
tensor([6], device='cuda:0')
tensor([6], device='cuda:0')
tensor([6], device='cuda:0')
tensor([1], device='cuda:0')
tensor([6], device='cuda:0')



下图是将输入图片二值化，对于低于 128 的值置 0，其余值置 255 的结果，可以看到，手写数字 4 的许多笔画都被遗漏掉了。这样处理之后模型的预测结果为 tensor([5], device='cuda:0')

tensor([5], device='cuda:0')
tensor([1], device='cuda:0')
tensor([2], device='cuda:0')
tensor([5], device='cuda:0')
tensor([5], device='cuda:0')
tensor([1], device='cuda:0')
tensor([1], device='cuda:0')
tensor([1], device='cuda:0')
tensor([1], device='cuda:0')



解决的方法还可以调用 opencv 的库函数来提高原图对比度，锐化图片之后再进行二值化等，但这样就并不是专注在神经网络模型上了。

对于修改神经网络模型来获得更好的泛化性能使之能够适应自己拍照上传的手写数字这个问题，也许可以在数据集生成时加入一定的数据增强的手段，比如让原数据像素值上下浮动一定范围等。

本模型在最初训练的时候尝试了随机裁剪和水平翻转等数据增强手段，但手写数字数据是一个较为依赖像素顺序，图片方向的数据（比如 6 和 9 的上下翻转或者旋转就会破坏原数据携带的信息）。这样做的结果反而不好（模型测试集准确率在 83%左右）。

综合来看，模型在同分布的 MNIST 测试集上表现良好。

附录

完整代码：

```
"""train

This module is homework1 for data mining class.
Main process in this module is visualizing MNIST
dataset then add some customized data and train
it, finally plot results.

the following comment has nothing to do with the code
but to record the process of experiment as to write into
homework report.
实验过程：
1. 去掉了 crop 和 flip 等 data augmentation，因为部分数据会被裁掉
2. 修改了 resnet 第一层以适应 BW 格式图片
3. 使用没有预训练的 resnet 模型
"""

from __future__ import barry_as_FLUFL

__version__ = '1.0'
__author__ = 'Haoyu Qi'

# import necessary packages
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import Dataset, DataLoader, TensorDataset, Subset
import torchvision
from torchvision import transforms,utils,models
import argparse
import seaborn as sns
from collections import Counter
import time
import copy
from torch.utils.tensorboard import SummaryWriter
from sklearn.metrics import confusion_matrix, accuracy_score
from PIL import Image
from pathlib import Path
```

```
# set flags / seeds
torch.backends.cudnn.benchmark = True
np.random.seed(1)
torch.manual_seed(1)
torch.cuda.manual_seed(1)

def dataVisualization(x_train,y_train,x_test,y_test):
    '''return null

    visualing first nine images
    save it as a image
    then visualize distribution of training set
    save it as a image
    '''
    plt.figure(figsize=(9, 9))
    for i in range(9):
        plt.subplot(331 + i)
        plt.imshow(x_train[i], cmap=plt.get_cmap('gray'))
    plt.savefig('sample.png', bbox_inches='tight', dpi=300)
    print(f'训练集的样本数：{x_train.shape[0]}，测试集的样本数：
{x_test.shape[0]}')
    print(f'输入图像的大小：{x_train.shape[1]}*{x_train.shape[2]}')
    y_train = np.array(y_train)
    label_cnt = Counter(y_train)  # 统计每个类别的样本数
    print('训练集的图像类别分布：', label_cnt)
    # 可视化图像类别分布
    plt.figure(figsize=(9,9))
    plt.pie(x=label_cnt.values(), labels=label_cnt.keys(), autopct='%.2
f%%')
    plt.savefig('label_distribution.png', bbox_inches='tight', dpi=300)

def train_model(model, dataloaders, writer, criterion, optimizer, sched
uler, device, args, num_epochs=25, save=True):
    '''return model

    load input model as last best model, begin trainning for num_epochs
    '''
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    best_epoch = 0

    for epoch in range(num_epochs):
```

```
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            count = 0
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
                # print(inputs.shape)
                # print(labels.shape)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    # print(f"outputs done:{outputs.shape}")
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
                count += inputs.size(0)

            epoch_loss = running_loss / count
            epoch_acc = running_corrects.double() / count
```

```
            if phase == 'train':
                scheduler.step()
                # log the running loss
                writer.add_scalar('training loss',
                                  epoch_loss,
                                  epoch)
                writer.add_scalar('training acc',
                                  epoch_acc,
                                  epoch)

            if phase == 'val':
                # log the running loss
                writer.add_scalar('validation loss',
                                  epoch_loss,
                                  epoch)
                writer.add_scalar('validation acc',
                                  epoch_acc,
                                  epoch)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_epoch = epoch
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)

    if save:
        torch.save({
            'epoch': best_epoch,
            'model_state_dict': best_model_wts,
            'optimizer_state_dict': optimizer.state_dict(),
```

```
            'acc': best_acc,
            }, args.SaveModel)
    return model

def test(model,test_loader,device):
    # set model to test mode
    model.eval()
    test_acc = 0.0
    count = 0

    # Initialize the prediction and label lists(tensors)
    predlist=torch.zeros(0,dtype=torch.long, device='cpu')
    lbllist=torch.zeros(0,dtype=torch.long, device='cpu')

    for i, (images, labels) in enumerate(test_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Predict classes using images from the test set
        outputs = model(images)
        _, prediction = torch.max(outputs.data, 1)

        test_acc += torch.sum(prediction == labels.data).float()
        count += images.shape[0]

        # Append batch prediction results
        predlist=torch.cat([predlist, prediction.view(-1).cpu()])
        lbllist=torch.cat([lbllist, labels.view(-1).cpu()])

    test_acc = test_acc / count
    print(f"total test accuracy:{test_acc} over {count} test samples")

    # Confusion matrix
    conf_mat=confusion_matrix(lbllist.numpy(), predlist.numpy())
    print(conf_mat)

    # Per-class accuracy
    accuracy = accuracy_score(lbllist, predlist)
    return test_acc

def my_model():
    model_body = models.resnet18(pretrained=False)

    num_ftrs = model_body.fc.in_features
```

```
    # Here the size of each output sample is set to 2.
    # Alternatively, it can be generalized to nn.Linear(num_ftrs, len(c
lass_names)).
    model_body.fc = nn.Linear(num_ftrs, 10)

    # modify first conv to accept bw img
    # weight = copy.deepcopy(model_body.conv1.weight)
    # new_weight = (torch.sum(weight,dim = 1)/3).unsqueeze(1)

    model_body.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2,
2), padding=(3, 3), bias=False)
    # model_body.conv1.weight.data = new_weight

    return model_body

def check_model(trainloader,net):
    # get some random training images
    dataiter = iter(trainloader)
    images, labels = dataiter.next()
    images = images.to(device)
    labels = labels.to(device)

    # create grid of images
    img_grid = torchvision.utils.make_grid(images)

    # write to tensorboard
    writer.add_image('four_mnist_images', img_grid)

    writer.add_graph(net, images)
    writer.close()

def stratified_sampling(ds,k):
    '''return trainset,validset

    sampling from ds while maintaining data balance
    acoss different classes.
    k is the number of samples in each class in trainset.
    '''
    class_counts = {}
    train_indices = []
    test_indices = []
    for i, (data, label) in enumerate(ds):
        c = label
        class_counts[c] = class_counts.get(c, 0) + 1
```

```
        if class_counts[c] <= k:
            train_indices.append(i)
        else:
            test_indices.append(i)

    print('stratified sampling done')
    return (train_indices,test_indices)

def predict_for_customized_data(model,dataPath,device):
    model.eval()
    for i in range(10):
        img1 = Image.open(dataPath/(str(i)+'.JPG')).convert('L')
        sized_img1 = transforms.Resize((28,28))(img1)
        sized_img1 = 255 - np.array(sized_img1)
        sized_img1[sized_img1<128]=0
        sized_img1[sized_img1>=128]=255
        tensor1 = transforms.ToTensor()(sized_img1)
        tensor1 = transforms.Normalize([0.485], [0.229])(tensor1)
        tensor1 = torch.unsqueeze(tensor1,0)
        assert tensor1.shape == (1,1,28,28)
        inputs = tensor1.to(device)
        outputs = model(inputs)
        # print(outputs)
        _, prediction = torch.max(outputs.data, 1)
        print(prediction)


if __name__ == '__main__':
    # argparse for using this code base
    parser = argparse.ArgumentParser(description="Train a network for M
NIST dataset,  \
                                          please make sure your compu
ter has a GPU")

    parser.add_argument("datasetPath", help="input path where your data
sets are",
                        type=str, nargs='?', default='/home/432/qihaoyu/dat
a/MNIST')

    parser.add_argument("logDir", help="input dir where your logs will
be recorded", \
                        type=str, nargs='?', default='/home/432/qihaoyu/vsc
ode_workspace/homework/mnist_experiment_1')
```

```
    parser.add_argument('SaveModel', help="input path where your model
will be saved", \
                        type=str,nargs='?',default='/home/432/qihaoyu/vscod
e_workspace/homework/models')
    parser.add_argument("img_size", help="standard image size to feed i
nto the network",
                        type=int, nargs='?', default=224)

    parser.add_argument("splits", help="ratio for trainning samples in
MNIST trainset, \
                               remaining samples will be treated as valids
et", type=float,
                               nargs='?', default=0.8)

    args = parser.parse_args()

    # transformation
    data_transforms = {
    'train': transforms.Compose([
        # transforms.RandomResizedCrop(224),
        # transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485], [0.229])
    ]),
    'test': transforms.Compose([
        # transforms.Resize(256),
        # transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485], [0.229])
    ])
    }

    # create tensorboard writer
    writer = SummaryWriter(args.logDir)

    # get pytorch datasets
    # if there's no existing datasets in the root directory, you should
 download it first
    trainset = torchvision.datasets.MNIST(args.datasetPath, train=True,
 download=False)
    testset = torchvision.datasets.MNIST(args.datasetPath, train=False,
 download=False)

    x_train = trainset.train_data
```

```
    y_train = trainset.train_labels
    x_test = testset.test_data
    y_test = testset.test_labels

    # visualize data and distribution according to instructions in ppt
    dataVisualization(x_train, y_train,x_test,y_test)

    # transform data and generate data loader to create batches
    raw_trainset = torchvision.datasets.MNIST(args.datasetPath, train=T
rue, download=False,
                                              transform=data_transforms['
train'])
    train_indices, valid_indices = stratified_sampling(trainset, 0.8*le
n(trainset)/10)
    trainset = Subset(raw_trainset,train_indices)
    validset = Subset(raw_trainset,valid_indices)



    train_data_loader = DataLoader(trainset,batch_size=128,shuffle=True
,drop_last=False)
    valid_data_loader = DataLoader(validset,batch_size=128,shuffle=Fals
e,drop_last=False)

    testset = torchvision.datasets.MNIST(args.datasetPath, train=False,
download=False,
                                              transform=data_transforms['
test'])
    test_data_loader = DataLoader(testset,batch_size=128,shuffle=False,
drop_last=False)

    # generate dataloaders dict
    dataloaders = {'train':train_data_loader, 'val':valid_data_loader}
    dataset_sizes = {'train':len(trainset), 'val':len(validset)}

    # if running on GPU and we want to use cuda move model there
    device = torch.device("cuda:0" if torch.cuda.is_available() else "c
pu")


    # instantiate network(body:resnet + head:fc)
    model = my_model()
    model = model.to(device)
    check_model(train_data_loader,model)
```

```
    criterion = nn.CrossEntropyLoss()

    # Observe that all parameters are being optimized
    optimizer_ft = optim.SGD(model.parameters(), lr=0.001, momentum=0.9
)

    # Decay LR by a factor of 0.1 every 7 epochs
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, g
amma=0.1)

    model = train_model(model, dataloaders, writer, criterion, optimize
r_ft, exp_lr_scheduler,
                        device, args, num_epochs=25,save=True)

    # test model
    test(model,test_data_loader,device)

    dataPath = Path('/home/432/qihaoyu/vscode_workspace/homework/imgs')
    predict_for_customized_data(model,dataPath,device)
```