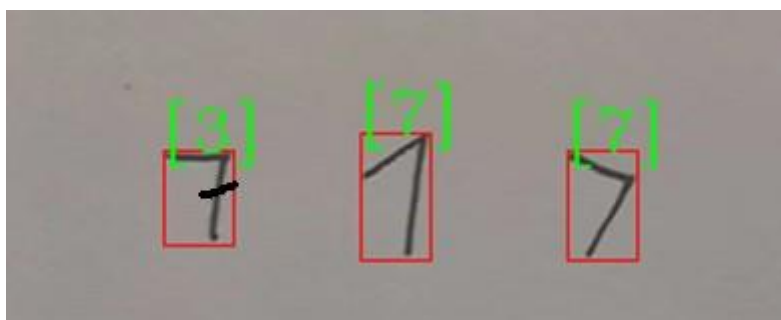


大作业一：自定义数据集实现手写数字识别

一、作业背景

MNIST 数据集是手写数字识别问题中最经典的数据集之一，许多模型在该数据集上取得了良好效果。但由于其来自美国国家标准与技术研究所，数字写法不够丰富，不符合中国人的书写习惯，在某些情况下难以识别出正确效果，如下图。



故丰富 **MNIST** 数据集使之可以识别更多种写法的手写数字，从而提升识别效果成为了当务之急。

二、作业介绍

本次作业的主要任务是使用自己手写的数字拍照扩充 **mnist** 的手写数字数据集训练一个全连接神经网络模型（**BP**），使之适应中国人的书写习惯，提升识别效果。

作业应有三部分组成，数据采集与处理，模型训练，模型交叉验证，结果可视化验证。

数据采集与处理部分的主要工作是收集自己用手机拍照 **0** 到 **9** 的手写数字图片，将其处理为 **mnist** 格式（**28x28**），添加进训练数据集；模型训练部分的主要工作是使用上述数据集进行模型训练，得到手写数字识别的全连接神经网络，同时使用混淆矩阵，提升交叉验证的效果；结果可视化验证部分，使用新的手写数字图片检验模型效果，并进行可视化展示。

三、代码详解

3.1. 图片预处理（`preprocess.py`）

（1）检测数字边框

本项目基于 **opencv** 的一个寻找轮廓的方法，利用 **OpenCV** 里面的 **findContours** 函数将边缘找出来，然后通过 **boundingRect** 将边缘拟合成一个矩形输出边框的左上角和右下角。

```
# 寻找边缘，返回边框的左上角和右下角（利用cv2.findContours）
def findBorderContours(path, maxArea=100):
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    img = accessBinary(img)
    #这里关于返回值是几个网上不太确定，但是本工程是2个
    contours, _ = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    borders = []
    for contour in contours:
        # 将边缘拟合成一个边框
        x, y, w, h = cv2.boundingRect(contour)
        if w * h > maxArea:
            if h > 20:
                border = [(x, y), (x + w, y + h)]
                borders.append(border)
    return borders
```

（2）转化为 MNIST 格式

首先将边框转为 28*28 的正方形，因为背景是黑色的，我们可以通过边界填充的形式，将边界扩充成黑色即可，其中值得注意的是 MNIST 那种数据集的格式是字符相对于居中的，我们得出的又是比较准的边框，所以为了和数据集相对一致，我们要上下填充一点像素。

```
def transMNIST(path, borders, size=(28, 28)):
    imgData = np.zeros((len(borders), size[0], size[0], 1), dtype='uint8')
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    img = accessBinary(img)
    for i, border in enumerate(borders):
        borderImg = img[border[0][1]:border[1][1], border[0][0]:border[1][0]]
        h = abs(border[0][1] - border[1][1]) #高度
        # 根据最大边长拓展像素
        extendPixel = (max(borderImg.shape) - min(borderImg.shape)) // 2
        h_extend = h // 5
        # print(h_extend)
        targetImg = cv2.copyMakeBorder(borderImg, h_extend, h_extend, int(extendPixel*1.1), int(extendPixel*1.1))
        targetImg = cv2.resize(targetImg, size)
        targetImg = np.expand_dims(targetImg, axis=-1)
        imgData[i] = targetImg
    return imgData
```

3.2. 数据集定义（add_data.py）

自定义 dataset 类，实现添加数据、交叉验证等功能。

加载 mnist 数据集。

```
def load_mnist():
    """Load MNIST data from 'path'"""
    labels_path = 'data\\MNIST\\raw\\train-labels-idx1-ubyte'
    images_path = 'data\\MNIST\\raw\\train-images-idx3-ubyte'

    with open(labels_path, 'rb') as lbpath:
        # magic, n = struct.unpack('>II', lbpath.read(8))
        labels = np.fromfile(lbpath, dtype=np.uint8)[8:]

    with open(images_path, 'rb') as imgpath:
        # magic, num, rows, cols = struct.unpack('>IIII', imgpath.read(16))
        images = np.fromfile(imgpath, dtype=np.uint8)[16:].reshape((-1, 28, 28))

    labels2_path = 'data\\MNIST\\raw\\t10k-labels-idx1-ubyte'
    images2_path = 'data\\MNIST\\raw\\t10k-images-idx3-ubyte'

    with open(labels2_path, 'rb') as lb2path:
        labels = np.concatenate((labels, np.fromfile(lb2path, dtype=np.uint8)[8:]), axis=0)

    with open(images2_path, 'rb') as img2path:
        images = np.concatenate((images, np.fromfile(img2path, dtype=np.uint8)[16:].reshape((-1, 28, 28))), axis=0)

    return images, labels
```

自定义 dataset，将两部分数据统一组织，并实现交叉验证。

```

add_data.py > Mydataset > __init__ > path
34 class Mydataset(Dataset):
35     def __init__(self, minst_img, label, transform, k, ki, typ) -> None:
36         super().__init__()
37         self.imgs = minst_img
38         self.label = label
39         for i in range(1):
40             path = 'data{}'.format(i)
41             # path = 'test1.png'
42             borders = findBorderContours(path)
43             imgdata = transMNIST(path, borders).reshape((-1, 28, 28))
44
45             self.imgs = np.concatenate((self.imgs, imgdata), axis=0)
46             self.label = np.concatenate((self.label, np.array([i] * imgdata.shape[0])))
47         leng = self.imgs.shape[0]
48         every_z_len = leng // k
49         if typ == 'test':
50             self.trimage = self.imgs[every_z_len * ki : every_z_len * (ki+1)]
51             self.trlabel = self.label[every_z_len * ki : every_z_len * (ki+1)]
52         elif typ == 'train':
53             self.trimage = np.concatenate((self.imgs[: every_z_len * ki], self.imgs[every_z_len * k : ]))
54             self.trlabel = np.concatenate((self.label[: every_z_len * ki], self.label[every_z_len * k : ]))
55
56         self.transform = transform
57
58     def __getitem__(self, idx):
59         return self.transform(self.imgs[idx]), self.label[idx]
60
61     def __len__(self):
62         return self.imgs.shape[0]

```

3.3. 模型定义（mymodel.py）

使用三层全连接网络实现手写数字分类。

```

mymodel.py > ...
1 import torch.nn as nn
2
3 class MLP_3Layer(nn.Module):
4     def __init__(self, w1, w2, w3, w4):
5         super(MLP_3Layer, self).__init__()
6         self.model = nn.Sequential(
7             nn.Linear(w1, w2),
8             nn.LeakyReLU(inplace=True),
9             nn.Linear(w2, w3),
10            nn.LeakyReLU(inplace=True),
11            nn.Linear(w3, w4),
12            nn.LeakyReLU(inplace=True)
13        )
14    def forward(self, x):
15        x = self.model(x)
16        return x

```

3.4. 模型训练与验证（main.py）

模型训练函数，同时使用 log 进行记录。

```

def train(kfold):
    # 定义logging输出的格式和内容
    logging.basicConfig(filename='log.txt',
                        level=logging.INFO,
                        format="%(asctime)s %(levelname)s %(message)s",
                        datefmt="%a %d %Y %H:%M:%S")
    ]
    imgs, labels = load_mnist()
    for j in range(kfold):
        logging.info('Kfold={}'.format(j))

        train_dataset = Mydataset(imgs, labels, transform=data_tf, k=kfold, ki=j, typ='train')
        test_dataset = Mydataset(imgs, labels, transform=data_tf, k=kfold, ki=j, typ='test')

```

加载训练好的模型进行验证。

```
# 预测手写数字
def predict(imgData):
    model = mymodel.MLP_3Layer(28*28, 300, 100, 10)
    model.load_state_dict(torch.load('org_model_mnist.pth'))
    result_number = []
    model.eval()
    img = data_tf(imgData)
    img = img.reshape(-1,784)
    print(img.shape)
    test_xmw = DataLoader(img)
    for a in test_xmw:
        img = Variable(a)
        out = model(img)
```

可视化展示预测结果。

```
def show_result():
    path = 'test3.png' #获取图像地址
    # print(path)
    borders = findBorderContours(path) #获取数字边界并截取成单个数字图像
    imgData = transMNIST(path, borders) #转变成mnist格式图像
    results = predict(imgData) #进行预测
    showResults(path, borders, results) #图像展示
```

四、 模型效果

训练过程的 loss 和 acc 见 log.txt。可视化结果展示如下图：

