基于Pytorch的单字识别

主要是使用卷积神经网络实现单字识别。网络是2层卷积，3层全连接的网络，训练了20次，测试正确率大约为98%（因为选用的训练集和测试集本身差别并不大）。图片处理成64X64的单通道灰度图片，卷积核大小为12X12，步长为4，池化层窗口大小和步长都为2，用最大池化方法。
任务的完成主要是在网上找的各种代码，然后添加和修改了一些代码用于这个项目。做的过程中也遇到了一些困难，比如安装Pytorch一系列的包，总是安装不成功；长时间不用，对Python的语法生疏；此外对于卷积神经网络的理解还不到位，一些代码并不知道为什么要这么写，仍需要后续的学习。当然也有一些收获：通过对这个任务的尝试，对于使用Pytorch实现卷积神经网络有了一定的了解，也更加了解到自己的不足还有接下来需要去学习的地方。下面是具体实现步骤：

1. 数据选取和文件夹处理

由于电脑处理速度原因，从老师所给的数据中选取100个彝文进行训练，并将这100个彝文的图片分别分为两部分，一部分用做训练集，一部分用作测试集。由于每个字体的文件夹是用Unicode编码，不便于处理，所以将每个文件夹名字重新用数字命名，作为每个彝文的label。

```
path = 'C:\\Users\\81568\\Desktop\\单字识别\\test\\' # 如果是训练集 改成train
dirs = os.listdir('C:\\Users\\81568\\Desktop\\单字识别\\test')
print(dirs)
f = sorted(dirs)
print(f)
n = 0
n1 = 0
for i in f:
    oldname = path + f[n]  # 设置旧文件名就是路径+文件名
    newname = path + str(n1)  # 设置新文件名
    os.rename(oldname, newname)  # 用os模块中的rename方法对文件改名
    print(oldname, '======>', newname)
    n += 1
    n1 += 1
```

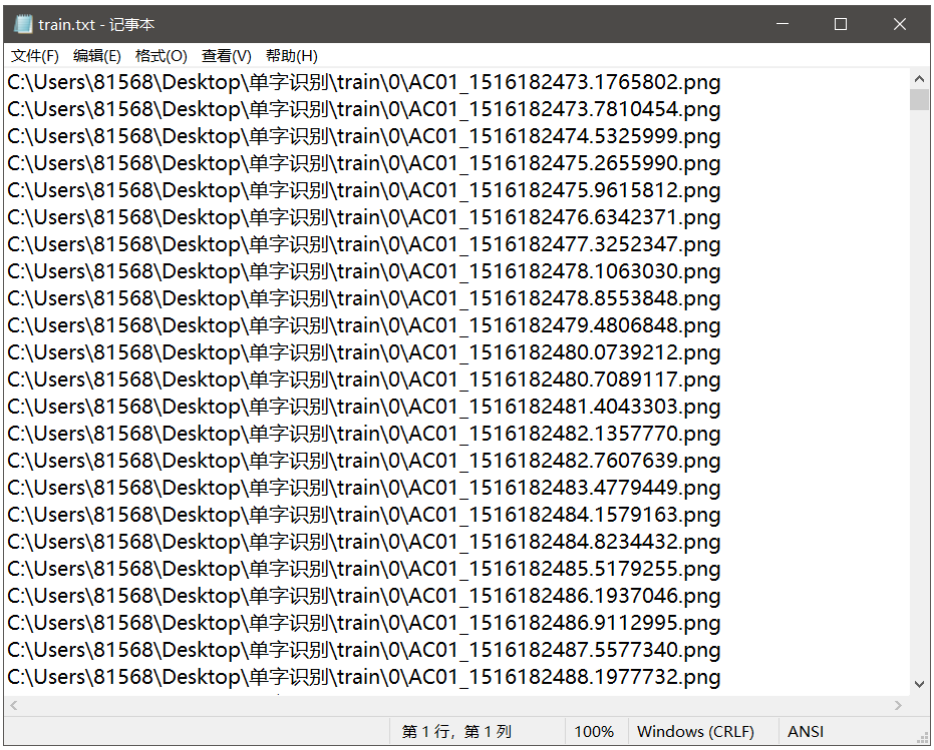
2. 提取图片路径

将训练集和测试集中每个彝文字体文件夹中的图片提取出来，生成txt文件，方便将每个图片输入到网络中

```
def classes_txt(root, out_path, num_class=None):
    dirs = os.listdir(root)  # 列出根目录下所有类别所在文件夹名
    if not num_class:  # 不指定类别数量就读取所有.不指定的时候num_class为空
```

```
            num_class = len(dirs)
        if not os.path.exists(out_path):  # 不存在就新建
            f = open(out_path, 'w')
            f.close()
        with open(out_path, 'r+') as f:  # 打开输出txt文件
            try:
                end = int(f.readlines()[-1].split('\\')[-2]) + 1
                # 读取txt文件所有行,取最后一行,以/为标志切割并取倒数第二个字符串,
取整加一因为文件是从0开始的
            except:
                end = 0
            if end < num_class - 1:
                dirs.sort()  # 对列表的对象进行排序
                for dir in dirs:  #
                    files = os.listdir(os.path.join(root, dir))  # 生成一
个彝文对应多张图片名称的列表
                    # 路径拼接成
                    for file in files:  # 取单张图片的名称——对于图片文件会
有.png后缀，文件夹则无后缀。
                        f.write(os.path.join(root, dir, file) + '\n')  #
将单张图片的路径信息写入txt文件，并换行
```

处理好的txt文件如下：



3. 数据定义和预处理

定义和记录彝文的标签信息，也就是文件夹名。labels利用transform图像预处理包，对图像进行预处理：大小统一设置为64*64、数据类型转换为Pytorch可处理的tensor形式、单通道灰度图像模式。读取图像数据，将可视化的图像处理为数字信息用于计算。

```
'''由于数据集图片尺寸不一，因此要进行resize（重设大小）,先设置transform的参数ToTensor():将PIL.Image读的图片（或者numpy的ndarray）转换成(C,H, W)的Tensor格式，并且归一化至[0~1]。（归一化至[0-1]是直接除以255）通道的具体顺序与cv2读的还是PIL.Image读的图片有关系。PIL.Image:(R, G, B)。cv2:(B,G,R)Grayscale：将图像转换为灰度图像'''
transform = transforms.Compose([transforms.Resize((64, 64)),  # 将图片大小重设为 64 * 64
                                transforms.Grayscale(),
                                transforms.ToTensor()])
```

4. 读取彝文图片

```
class MyDataset(Dataset):
    def __init__(self, txt_path, num_class, transforms=None):
        super(MyDataset, self).__init__()
        images = []  # 存储图片路径
        labels = []  # 彝文类别名，也就是文件夹名
        with open(txt_path, 'r') as f: # 打开存放图片路径的txt文件
            # 遍历f的每一行line,生成新的list,line for line in ...是为了对遍历的每一行做处理的
            for line in f:  # 本身就是一行一行读取。
                if int(line.split("\\")[-2]) >= num_class:  # 只读取前num_class 个类
                    break
                # 由于此处本就是一行一行读取，不去掉换行符的话，会多一个换行----即多一行空格
                line = line.strip('\n')  # 移除字符串头尾指定的字符（默认为空格）--此处\n转义为换行
                images.append(line)  # 使用append()给images添加元素----储存单张图片路径
                labels.append(int(line.split('\\')[-2]))
                # 两者蕴含的信息是前num_class个类别字的所有图片 这里不实际加载图片，只是指定图片的路径和标签 调用__getitem__时才会真正读取图片）
        self.images = images  # 图片的路径
        self.labels = labels  # 哪个文字
        self.transforms = transforms  # 图片需要进行的变换，ToTensor()
    # 真正去读取图片
    def __getitem__(self, index):
        image = Image.open(self.images[index])  # 用PIL.Image读取图像，打开为RGB模式（彩色图像模式，输入通道数为3）。
        label = self.labels[index]  # 取某张图片的对应彝文的标签
        if self.transforms is not None:
            image = self.transforms(image)  # 进行变换 变成灰度图像了，输入通道就是1
        return image, label
    # 用于getitem函数中取图像路径--对应彝文标签
    def __len__(self):
        # print('labels is' + len(self.labels))
```

```
        return len(self.labels)  # 获得lables的长度–也就是所有图片的数量。
```

5. 构建卷积神经网络

一层卷积层的几个参数:in_channels=3:表示的是输入的通道数，由于是RGB型的，所以通道数是3. 此处实例输入通道设置为1，应该是RGB型后经过了某种处理.out_channels=96:表示的是输出的通道数，设定输出通道数的96（这个是可以根据自己的需要来设置的）。kernel_size=12:表示卷积核的大小是12x12的，也就是上面的“F”，F=12。stride=4:表示的是步长为4，也就是上面的S, S=4。padding=2:表示的是填充值的大小为2，也就是上面的P, P=2。假如你的图像的输入size是256x256的，由计算公式知N=(256-12+2x2)/4+1=63,也就是输出size为63x63的

```
class NetSmall(nn.Module):
    # 卷积→池化→卷积→全连接→全连接→输出100个汉字的概率(略去了softmax层)
    def __init__(self):
        super(NetSmall, self).__init__()  # 父类继承
        self.conv1 = nn.Conv2d(1, 6, 3)  # 3个参数分别是in_channels，out_channels，kernel_size，还可以加padding
        #  in_channels – 输入信号的通道；out_channels(int) – 卷积产生的通道；kerner_size - 卷积核|滤波器/卷积层窗口的尺寸
        self.pool = nn.MaxPool2d(2, 2)  # 池化层窗口大小，窗口移动的步长。（二者通常设置为相同，默认也是步长=窗口大小）
        self.conv2 = nn.Conv2d(6, 16, 5)  # 第二层的输入通道=第一层的输出通道
        self.fc1 = nn.Linear(2704, 512)  # linear是全连接层，2个参数分别是输入神经元数、输出神经元数。【与连接的两层神经元数量保持一致】
        self.fc2 = nn.Linear(512, 84)  # 使用的是Xavier权重初值
        self.fc3 = nn.Linear(84, 100)  # 100代表一次处理100个文字
    # （64-3）/2取31，（31-5）/2取13，到第一层全连接层，13*13*16=2704
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # 第一层：卷积+激活+池化
        x = self.pool(F.relu(self.conv2(x)))  # 第二层：卷积+激活+池化
        x = x.view(-1, 2704)  # view函数相当于numpy的reshape
        x = F.relu(self.fc1(x))  # 全连接+激活
        x = F.relu(self.fc2(x))  # 全连接+激活
        x = self.fc3(x)  # 全连接输出。不加softmax函数不影响分类结果
        return x
```

6. 模型训练方法

定义参数

```
# 定义超参数
EPOCH = 20  # 训练次数
# 一个epoch代表  所有训练数据/batch_size  进行学习的数据大小
BATCH_SIZE = 50  # 数据集划分。size of mini-batch
LR = 0.001  # 学习率
```

选择使用Adam参数更新方法

```
model = NetSmall()
# 网络的可学习参数可以通过model.parameters()返回----variable对象
optimizer = torch.optim.Adam(model.parameters(), lr=LR)  # 参数优化方法选择
loss_func = nn.CrossEntropyLoss()  # 分类误差计算函数----交叉熵误差损失函数
device = torch.device('cpu')
model.to(device)
```

7. 放进迭代器中进行训练

将读出来的图片信息放入迭代器中，使得数据可以被batch操作，一次batch操作50张图片，测试集batch设置成10000主要是为了一次将所有图片处理完，不用分开处理。shuffle 设置为True时会在每个epoch重新打乱数据(默认: False).

```
# 真正开始读图片
train_set = MyDataset(root + '/train.txt', num_class=100, transforms=transform)  # num_class选取100种彝文
test_set = MyDataset(root + '/test.txt', num_class=100, transforms=transform)
# 将读出来的图片信息放入迭代器中
train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_set, batch_size=10000, shuffle=True)

for epoch in range(EPOCH):  # 100种彝文 ，epoch=20
    epoch = epoch + 1
    for step, (x, y) in enumerate(train_loader): # enumerate打乱顺序
        train_x, labels_train = x.to(device), y.to(device)
        output = model(train_x)
        tr_loss = loss_func(output, labels_train)  # 训练数据损失函数
        optimizer.zero_grad()  # 把梯度置零。等价为model.zero_grad()
        tr_loss.backward()  # 反向传播，计算当前梯度
        optimizer.step()  # 根据梯度更新网络参数

        if step % 50 == 0:  # 验证训练出来的模型（参数）对测试数据的识别度
            test_output = model(test_x)
            pred_y_tx = torch.max(test_output, 1)[1].data.squeeze()
            tx_accuracy = (pred_y_tx == labels_test).sum().item() / labels_test.size(0)
            # 训练数据的识别准确度
            train_output = model(train_x)
            pred_y_tr = torch.max(train_output, 1)[1].data.squeeze()
            tr_accuracy = (pred_y_tr == labels_train).sum().item() / labels_train.size(0)
            self_output = model(self_x)
```

```
            pred_y_tse = torch.max(self_output, 1)[1].data.squeeze()
            se_accuracy = (pred_y_tse == labels_self).sum().item() /
labels_self.size(0)
            print('Epoch:', epoch, '| train loss:%.4f' %
tr_loss.data, '| test accuracy:', tx_accuracy)  # 输出训练次数、误差、测
试准确率
```

输出训练次数、误差、测试准确率：

```
Epoch: 16 | train loss:0.0096 | test accuracy: 0.9874500855675984
Epoch: 17 | train loss:0.0080 | test accuracy: 0.9910629397223807
Epoch: 17 | train loss:0.0081 | test accuracy: 0.9897318881916715
Epoch: 17 | train loss:0.0061 | test accuracy: 0.9904924890663624
Epoch: 18 | train loss:0.0039 | test accuracy: 0.9887811370983076
Epoch: 18 | train loss:0.0809 | test accuracy: 0.9849781327248527
Epoch: 18 | train loss:0.0036 | test accuracy: 0.9897318881916715
Epoch: 19 | train loss:0.0739 | test accuracy: 0.9855485833808709
Epoch: 19 | train loss:0.0034 | test accuracy: 0.993915193002472
Epoch: 19 | train loss:0.0085 | test accuracy: 0.9897318881916715
Epoch: 20 | train loss:0.0074 | test accuracy: 0.9884008366609621
Epoch: 20 | train loss:0.0372 | test accuracy: 0.9834569309754706
Epoch: 20 | train loss:0.0318 | test accuracy: 0.9887811370983076
Finish training
```

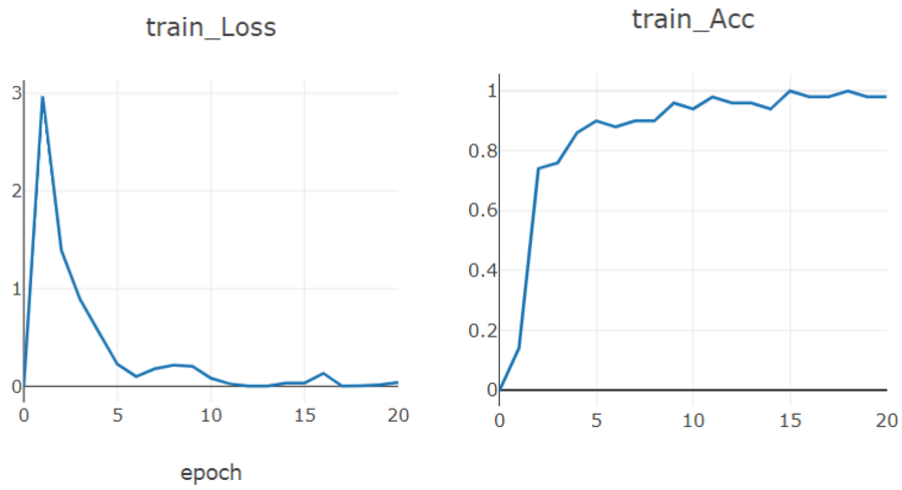
8. 可视化结果分析

需要先执行命令`python -m visdom.server`

```
viz = visdom.Visdom(env='dev')
# 初始化
train_loss_x, train_loss_y = 0, 0
win1 = viz.line(X=np.array([train_loss_x]),
Y=np.array([train_loss_y]), opts=dict(title='train_Loss',
xlabel='epoch'))
test_acc_x, test_acc_y = 0, 0
win2 = viz.line(X=np.array([test_acc_x]), Y=np.array([test_acc_y]),
opts=dict(title='test_Acc', xlabel='epoch'))

# 每个epoch更新一次
viz.line(X=np.array([epoch]), Y=np.array([tr_loss.data]), win=win1,
update='append')
viz.line(X=np.array([epoch]), Y=np.array([tx_accuracy]), win=win2,
update='append')
```

训练数据损失值，以及测试数据识别准确度：

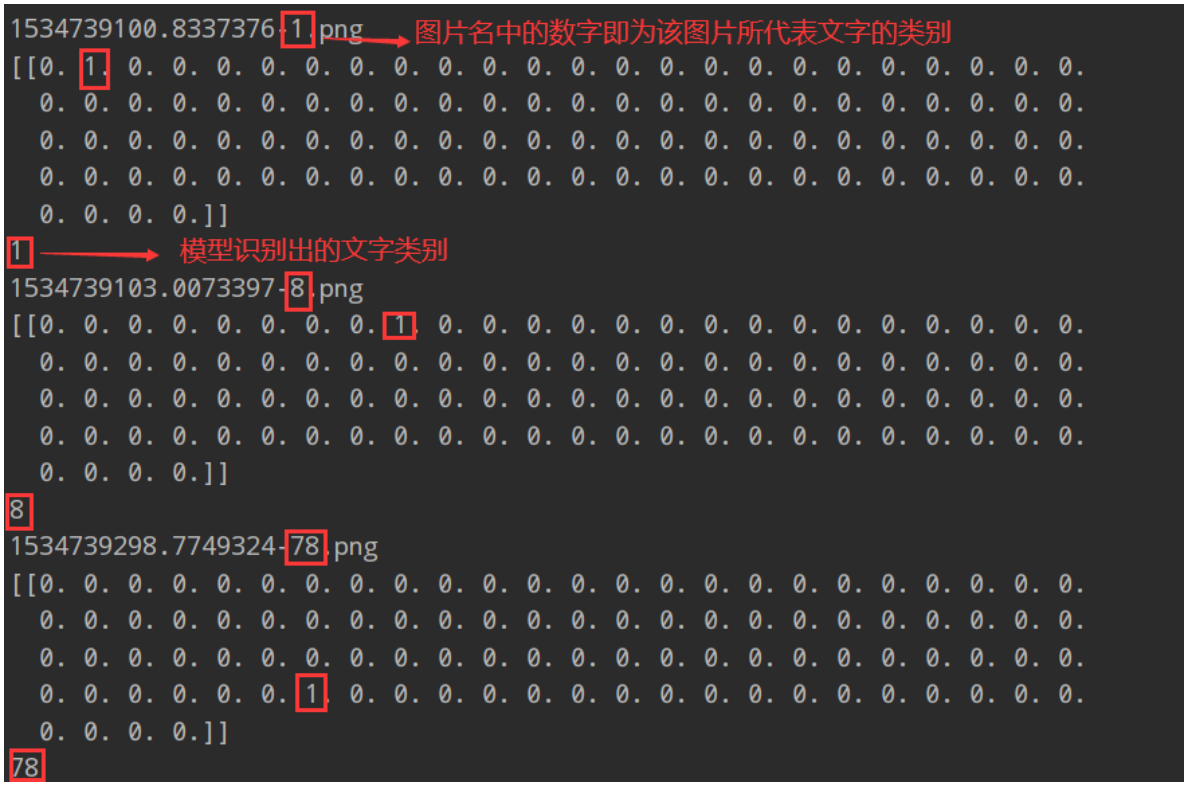


9. 验证模型能否识别

从数据中选择几张彝文图片，然后输入到模型，然后输入模型识别每张图片的类别的概率

```
def valid():
    for pngfile in glob.glob(r'*.png'): # 选择文件夹中用于验证的图片
        print(pngfile)  # 打印图片名称，以与结果进行对照
        img = cv2.imread(pngfile)  # 读取要预测的图片，读入的格式为BGR
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # 图片转为灰度图，因为训练数据集都是灰度图
        img = cv2.resize(img, (64, 64)) # 设置大小
        img = np.array(img).astype(np.float32)
        img = np.expand_dims(img, 0)
        img = np.expand_dims(img, 0)  # 扩展后，为[1，1，64，64]
        img = torch.from_numpy(img)
        output_1 = model(Variable(img))
        prob = F.softmax(output_1, dim=1)
        prob = Variable(prob)
        prob = prob.cpu().numpy()  # 用GPU的数据训练的模型保存的参数都是gpu形式的，要显示则先要转回cpu，再转回numpy模式
        print(prob)  # prob是100个分类的概率
        pred = np.argmax(prob)  # 选出概率最大的一个
        print(pred.item()) # 输出
```

输出结果如下：



10. 参考资料

[1] TensorFlow与中文手写汉字识别

[2] MNIST 训练完的模型加载及输入本地图片识别(pytorch

[3] PytorchCNN实现手写汉字识别（数据集制作，网络搭建，训练验证测试全部代码)

[4] 基于Pytorch的手写汉字识别

[5] PyTorch：使用Visdom进行可视化