Python 小学期

作业报告

作业主题：基于人脸图像自动识别性别

班级：无 04 班

学号：2020010733

姓名：邵晨扬

日期：2021 年 8 月 5 日

目录

一、数据集概念学习及准备过程

首先我花了一定的功夫厘清了训练集、验证集和测试集三者的概念。

在编写人脸识别程序之前，我研究了一下非常经典的“手写数字识别”的项目。这个项目是有一个现成的数据集“MNIST”数据库。在著名的神经网络研究员 Yann LeCun 的网站中可以下载两个 CSV 文件，一个是训练集，一个是测试集。其中训练集是用来训练神经网络的 60000 个标记样本集，而测试集中只有 10000 个样本来测试最终的模型的好坏程度。

我也思考过为什么要将训练集和测试集分开，我觉得是为了确保可以使用神经网络之前没有见过的数据进行再次测试。否则实际上可以使用一种欺骗性的手段，让神经网络简单地记忆训练数据，从而得到一个完美的但是有欺骗性的结果。这个结果显然是和我们的初衷背道而驰的。我注意到，在助教发布的手写数字识别程序中，是训练一轮就用测试集测试一次。最后会输出测试集的准确率最高的一次。

然后，在作业的要求中，还提到了验证集，这个概念是在 MNIST 数据中没有体现的。经过我的调查，我对于验证集的作用大体有了一些了解，主要可以归结为三个方面：

一：验证集可以帮助我们及时发现模型或者参数的问题，当结果异常的时候可以及时终止训练，重新调参或者调整模型，而不需要等到训练结束。

二：验证集可以及时地验证模型的泛化能力，如果在验证集上的效果就已经比训练集上差很多，就该考虑模型是否过拟合。而不必等到用测试集来进行测试。

三：验证集参与了**超参数**的训练

我了解到在训练模型的时候参数可以分为两种，一种是普通的模型参数，一种是需要人工调参的超参数。而普通参数的训练使用的是训练集的数据不停地训练调整。但是有的参数是无法通过一轮轮的训练来主动调整的。比如迭代次数（历元），层数，每层神经元的个数
、学习率等等等等。这些超参数的设置都依赖于验证集的表现。迭代次数（历元），层数，每层神经元的个数

所以这三个数据集的使用方法是这样的：在训练集上训练模型，在验证集上评估模型，一旦找到的最佳的参数，就在测试集上最后测试一次。测试集上的误差就作为泛化误差的近似。（泛化误差：是用来衡量一个学习机器推广未知数据的能力的指标）

为了方便起见，我打算利用 pytorch 中的 ImageFolder 来加载我的数据集。那么首先我的工作就是把 “lfw_funneled”文件夹里的所有的图片按照那两个记载了男女名字的文档进行分类，我不仅仅要按照男女分类，还要按照 train、val、test 三个文件夹进行分类。也就是说我最终分类出的的 data 文件夹下有三个子文件夹，分别是 train、val、test，而这三个子文件下又有两个文件夹，一个是 male，一个是 female，相当于是图片的标签。在 male 和 female 中是按比例随机分配的男女人脸图像。比例就是要求中的 4:1:5。

我编写了一个程序来实现这样的分配模式。源代码如下：

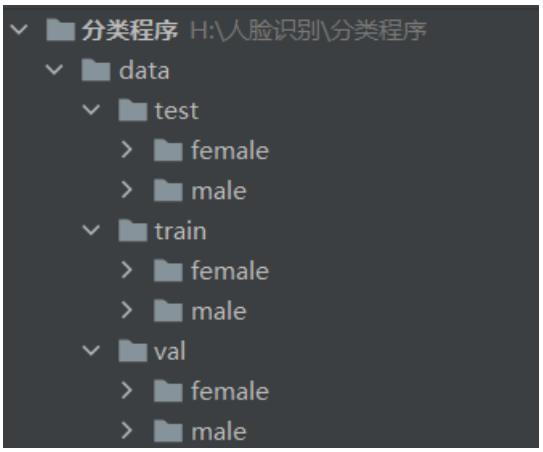
```
# 导入需要的包
import os
import random
import shutil
```

```
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
os.environ["CUDA_VISIBLE_DEVICES"] = '1'

path = 'H:\\Pytorch\\face_read-try1\\lfw_funneled'

file = open('H:\\人脸识别\\分类程序\\female_names.txt')
data_female = []
for line in file.readlines():
    line = line.strip('\n')
    data_female.append(line)

file = open('H:\\人脸识别\\分类程序\\male_names.txt')
data_male = []
for line in file.readlines():
    line = line.strip('\n')
    data_male.append(line)

trainmale = 0  # 4105
testmale = 0  # 1026
valmale = 0  # 5132

trainfemale = 0  # 1186
testfemale = 0  # 297
valfemale = 0  # 1483

for filepath, dirnames, filenames in os.walk(r'H:\\人脸识别\\分类程序
\\lfw_funneled'):
    for filename in filenames:  # filename 就是文件名
        filename2 = os.path.join(filepath, filename)  # filename2 是绝对路径
        print(filename2)
        if filename in data_male:
            n = random.randint(0, 2)
            if (n == 0) and (trainmale >= 4105):
                n += 1
            if (n == 1) and (testmale >= 1026):
                n += 1
            if (n == 2) and (valmale >= 5132):
                n -= 1
                if testmale >= 1026:
                    n -= 1
            if n == 0:
                shutil.move(filename2, 'H:\\人脸识别\\分类程序\\data\\train\\male')
                trainmale += 1
            elif n == 1:
                shutil.move(filename2, 'H:\\人脸识别\\分类程序\\data\\test\\male')
                testmale += 1
            else:
                shutil.move(filename2, 'H:\\人脸识别\\分类程序\\data\\val\\male')
                valmale += 1
        if filename in data_female:
            n = random.randint(0, 2)
            if (n == 0) and (trainfemale >= 1186):
                n += 1
            if (n == 1) and (testfemale >= 297):
                n += 1
            if (n == 2) and (valfemale >= 1483):
                n -= 1
                if testmale >= 297:
                    n -= 1
            if n == 0:
                shutil.move(filename2, 'H:\\人脸识别\\分类程序\\data\\train\\female')
                trainfemale += 1
            elif n == 1:
                shutil.move(filename2, 'H:\\人脸识别\\分类程序\\data\\test\\female')
```

```
            testfemale += 1
        else:
            shutil.move(filename2, 'H:\\人脸识别\\分类程序\\data\\val\\female')
            valfemale += 1
```

有趣的是，在原来的数据集中还有一些错误。比如有的图片名在两个文档中都没有出现，而有的图片名在两个文档中都出现了。经过一些手动的调整和纠正，最终成功实现了所有图片的划分，完成了数据集的准备。

最终的数据集是这样的：



二、使用模型及方法

我最终选择采用卷积神经网络来完成这次作业。我主要参考的模型是 AlexNet 网络结构，也就是文章《ImageNet Classification with Deep Convolutional Neural Networks》所介绍的网络结构。这篇论文可以说是多层 CNN 用在图像领域的首次尝试，此前还有一个 LeNet 结构，用在了手写数字识别上，但是没有用到连续多层 CNN。这篇文章中设计并验证了这样一个多层卷积层网络的有效性，对我的编程的启示很大。

首先我去详细了解了一下什么是卷积神经网络。卷积神经网络与传统的多层卷积网络的不同之处正是在于组成 CNN 的各个网络层不是常见的神经网络层，而是由卷积层、池化层、ReLU 层和全连接层四种特殊的网络构成的。

四层各有作用：卷积层可以概括成对图像特征的抽取；池化层是用来减少参数数量和计算量；ReLU 层用一个激活函数来提升整个网络的非线性判断能力，而全连接层用于最后的输出。

搞清楚了基本的架构，我开始学习 AlexNet 的结构。AlexNet 一共有 8 层。由 5 层卷积层和 3 层全连接层构成。而且在每层卷积层和全连.接层之后都有 ReLU 激活函数，在 1、2、5 层卷积层之后还添加了最大池化的操作。输入采用 RGB 的 3 通道输入。

在这篇论文中还有很多细节的信息，比如为何采用 ReLU 而不用 sigmoid 或 tanh 等激活函数，比如多 GPU 训练等等。

我的卷积神经网络就是借鉴的 AlexNet 的模型，采用 5 层卷积层和 3 层全连接层的结构。网络的结构基本没有变化。

三、测试结果

在我的程序中有这样几个超参数，是需要手动设置的，一个是学习率 LR，一个是训练的轮数，一个是随机梯度下降法中有一个 momentum 参数可能需要调整。其中训练的轮数其实不太需要主要的关注，只要在程序中不断保存在验证集上准确率最高的模型就可以了。而学习率和 momentum 参数是需要重点调整的。

我本来是打算使用随机梯度下降法作为优化算法。但是，我偶然阅读到了一片介绍 Adam 优化算法的论文，了解到这是一种对随机梯度下降法的扩展，可以代替经典的随机梯度下降法来更有效地更新网络权重。与传统的学习率保持单一不变的随机梯度下降法相比，Adam 使用动量和自适应学习率来加快收敛速度。所以我决定先使用 Adam 优化算法来进行尝试，结果是这样：

```
Epoch 9/9
----------
100%|██████████| 42/42 [00:26<00:00,  1.60it/s]
train Loss: 0.2305 Acc: 0.8993
100%|██████████| 11/11 [00:13<00:00,  1.26s/it]
val Loss: 0.2517 Acc: 0.8934
  0%|          | 0/52 [00:00<?, ?it/s]
 Training complete in 6m 53s
Best Val Acc: 0.893424



******test begin: ******
100%|██████████| 52/52 [00:23<00:00,  2.21it/s]
***  test Loss: 0.2723 Acc: 0.8830
Test Acc: 0.882993

Process finished with exit code 0
```

【由于多线程计算以及 pytorch-1.9.0 版本的某些问题，进度条出现了多行打印的问题】

这次实验一共是 10 轮，最终的效果是：在验证集上最好的一次准确率是 89.34%，在测试集上的准确率是 88.30%。效果还是不错的，毕竟只有十轮。

接下来，我利用这次训练保存下来的模型进行下一阶段的训练，这一次依旧是训练 10 轮。效果是这样的：

```
 Training complete in 13m 47s
Best Val Acc: 0.918367
```

```
******test begin: ******
100%|██████████| 52/52 [00:22<00:00,  2.30it/s]
***  test Loss: 0.2266 Acc: 0.9208
Test Acc: 0.920786

Process finished with exit code 0
```

在验证集上的准确率最高是 91.84%，而在测试集上的准确率是 92.08%。效果还是不错的。

但是我看了一下训练过程的一些数据，就注意到一个**问题**：在这一轮训练训练到第 5 次的时候，在验证集上的效果就已经达到了最好，后面的几轮一直在那个数值附近波动，没有太大的变化。有两种可能：一种可能是因为训练过程本就不是平滑上升的过程，波动是正常的；另一种可能就是这可能已经到了模型的极限了，准确率不会再有明显的增加。为了进一步探究到底是那种情况，我在这一轮的基础上又训练了十轮，也就是进行第 21-30 次训练。

训练的结果是在验证集上的最好表现是 92.8%，而在测试集上的表现是 93.05%。和上一轮相比，确实没有大的改观。所以我觉得是模型本身出了问题，有点过拟合了。

然后我尝试过修改了一些参数，可是始终没有较大的改观，始终没有稳定突破过 95%的准确率。（除去偶然几次能上 95%）

一时找不到原因，我又拿经典的随机梯度下降法进行了试验，我设置了学习率在每 5 轮之后会降低为原来的 1/10，出乎意料，训练的效果还是不错的。

这是一共训练 30 轮的情况：（先 10 轮后 20 轮，分了两段）：

```
Epoch 19/19
----------
100%|██████████| 42/42 [00:27<00:00,  1.54it/s]
  0%|          | 0/11 [00:00<?, ?it/s]train Loss: 0.0201 Acc: 0.9932
100%|██████████| 11/11 [00:17<00:00,  1.59s/it]
val Loss: 0.2115 Acc: 0.9501

Training complete in 15m 33s
Best Val Acc: 0.956160



******test begin: ******
100%|██████████| 52/52 [00:23<00:00,  2.20it/s]
***  test Loss: 0.1854 Acc: 0.9543
Test Acc: 0.954346

Process finished with exit code 0
```

训练到后期，在训练集上的准确率已经相当高了，在验证集上的最高准确率也达到了 95.62%，（其实后来一直稳定在 95.5%附近）在测试集上的准确率达到了 95.43%。相当不错。

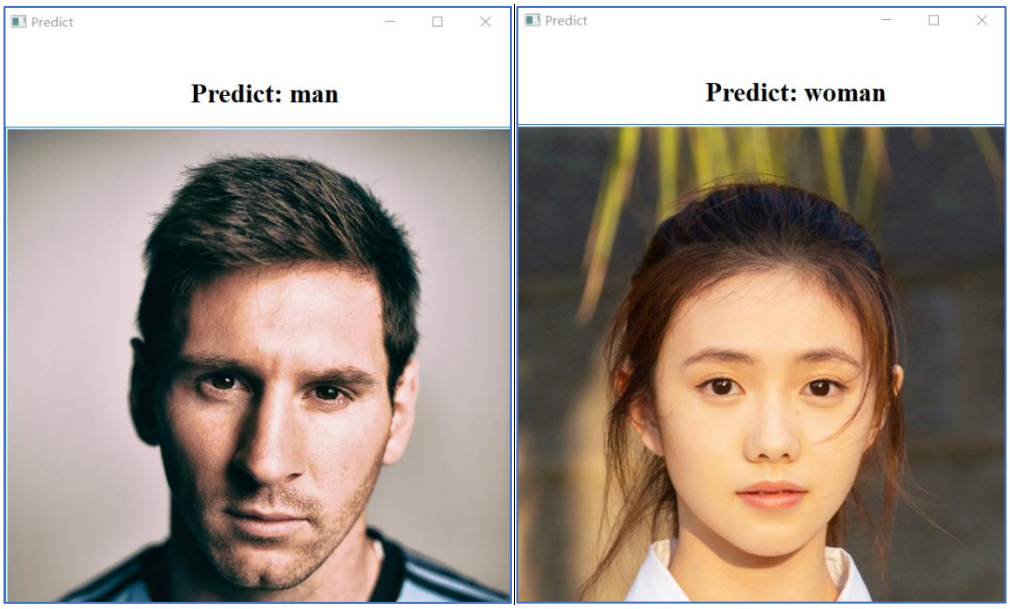
我甚至怀疑这是偶然情况，于是从头开始又训练了 30 轮：结果是这样的：

```
Training complete in 8m 57s
Best Val Acc: 0.966742
******test begin: ******
100%|██████████| 207/207 [00:22<00:00,  9.15it/s]
***  test Loss: 0.1157 Acc: 0.9601
Test Acc: 0.960091

Process finished with exit code 0
```

表现依然十分不错，又进行了几次实验，结果也都差不多。所以也许随机梯度下降法比 Adam 优化算法更适合我的程序。

四、附加任务

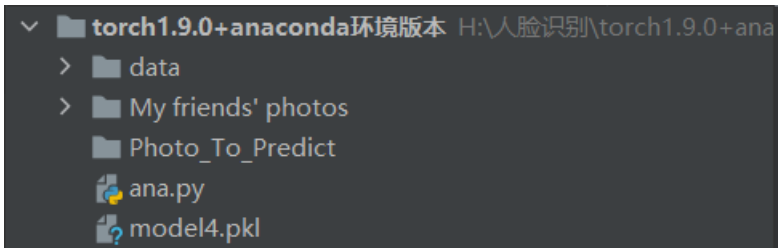
这一次的附加任务的话，我是制作了一个简单的单张照片的预测小程序。界面外观就像这样。



其实这只是一个简单的显示的界面，并不能完成程序和用户的交互作用。主体的操作并不能通过这个显示界面来完成。还是得在程序中来进行。

这个预测程序是这样的，为了方便起见，我直接在原来的训练、检测的程序中，加了一段用来可视化的操作。想要具体地实现单张照片的预测，需要两步：

第一步是把程序中的 EPOCH，也就是模型的训练轮数设置成 0，这样程序就会自动判断是要进行单张照片的预测。然后在项目下有一个叫做“Photo_To_Predict”的文件夹，只需要把你想要预测的照片放在这个文件下就可以了。



运行程序，就可以得到以上两幅图的效果。

五、参考文献

1、Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. “Imagenet classification with deep convolutional neural networks.” Advances in neural information processing systems. 2012.
2、Diederik P. Kingma, Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In Proceedings of ICLR 2015. (Citation: 37,480)
3、Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In Proceedings of ACL 2002. (Citation: 10,700)
4、Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In Proceedings of NIPS 2014. (Citation: 9,432)

六、源程序

图片分类的源代码已经在上面给过了，这里仅仅是卷积神经网络的代码（单张照片的预测界面的代码也在其中）：（所有的代码文件在作业中也给出）

```
# 导入需要的包
import os
import shutil

os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # 不显示等级 2 以下的提示信息
import torch
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import datasets, transforms
import time
```

```
from tqdm import tqdm
import matplotlib.pyplot as plt
import sys
import wx
from PIL import Image

# 设置超参数
BATCH_SIZE = 128  # 如果是笔记本电脑跑或者显卡显存较小，可以减小此值
LR = 0.001  # 学习率
MM = 0.95  # 随机梯度下降法中 momentum 参数  通常，在初始学习稳定之前，取 0.5，之后取 0.9 或更大。
EPOCH = 0  # 训练轮数
Num_Workers = 4

# device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device = torch.device('cuda')

# 图像变换
data_transforms = {
    "train":
        transforms.Compose([
            transforms.Resize((227, 227)),  # 调整图片大小为给定大小
            transforms.RandomHorizontalFlip(),  # 依概率 p 水平翻转：transforms.RandomHorizontalFlip(p=0.5)，p 默认值为 0.5
            transforms.ToTensor(),  # 将图片转换为 tensor，并且归一化至[0-1] 归一化至[0-1]是直接除以 255
            transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
        ]),
    "test":
        transforms.Compose([
            transforms.Resize((227, 227)),  # 调整图片大小为给定大小
            transforms.RandomHorizontalFlip(),  # 依概率 p 水平翻转：transforms.RandomHorizontalFlip(p=0.5)，p 默认值为 0.5
            transforms.ToTensor(),  # 将图片转换为 tensor，并且归一化至[0-1] 归一化至[0-1]是直接除以 255
            transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
        ]),
}
# 通用的数据加载器
train_dataset = datasets.ImageFolder(root='./data/train', transform=data_transforms["train"])  # 加载训练集
train_dataloader = DataLoader(dataset=train_dataset,
                              batch_size=BATCH_SIZE,  # 一次训练所抓取的数据样本数量
                              shuffle=True,  # 每次对数据进行重新排序
```

```
                                num_workers=Num_Workers)  # 有几个进程来处理
data loading

val_dataset = datasets.ImageFolder(root='./data/val', transform=data_transforms["te
st"])  # 加载验证集
val_dataloader = DataLoader(dataset=val_dataset,
                            batch_size=BATCH_SIZE,
                            shuffle=True,  # 在验证集中不需要每次对数据进行重新排序
                            num_workers=Num_Workers)  # 有几个进程来处理 data loading

test_dataset = datasets.ImageFolder(root='./data/test', transform=data_transforms["
test"])  # 加载测试集
test_dataloader = DataLoader(dataset=test_dataset,
                             batch_size=BATCH_SIZE,  # 一次训练所抓取的数据样本数量;
                             shuffle=True,  # 在测试集中不需要每次对数据进行重新排序
                             num_workers=Num_Workers)  # 有几个进程来处理
data loading

"""
返回的 dataset 都有以下三种属性:
self.classes: 用一个 list 保存类别名称
self.class_to_idx: 类别对应的索引,与不做任何转换返回的 target 对应
self.imgs: 保存(img-path, class) tuple 的 list

pytorch 的数据加载到模型的操作顺序是这样的:
① 创建一个 Dataset 对象
② 创建一个 DataLoader 对象
③ 循环这个 DataLoader 对象,将 img, label 加载到模型中进行训练
"""

# 定义网络结构,简单的网络结构可以通过 nn.Sequential 来实现,复杂的
# 网络结构需要通过继承 nn.Module 来自定义网络类来实现,在此使用自定义
# 类的方法给出一个简单的卷积神经网络,包括两个卷积层和两个全连接层
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 96, 11, 4)
        self.conv2 = nn.Conv2d(96, 256, 5, 1, 2)
        self.conv3 = nn.Conv2d(256, 384, 3, 1, 1)
        self.conv4 = nn.Conv2d(384, 384, 3, 1, 1)
        self.conv5 = nn.Conv2d(384, 256, 3, 1, 1)
        self.dropout1 = nn.Dropout(0.5)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 4096)
```

```
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, 2)
        self.relu = nn.ReLU()
        self.max_pool = nn.MaxPool2d(3, 2)

    def forward(self, x):  # in:277*277*3
        x = self.conv1(x)  # in:227*227*3 out:96*55*55
        x = self.relu(x)
        x = self.max_pool(x)  # in:96*55*55  out:27*27*96
        x = self.conv2(x)  # in:27*27*96 out:13x13x256
        x = self.relu(x)
        x = self.max_pool(x)
        x = self.conv3(x)  # in:27*27*96 out:13x13x384
        x = self.relu(x)
        x = self.conv4(x)  # in:13x13x384 out:13x13x384
        x = self.relu(x)
        x = self.conv5(x)  # in:13x13x384 out:6x6x256
        x = self.relu(x)
        x = self.max_pool(x)

        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x

model = Net().to(device)

# 定义损失函数，分类问题采用交叉熵损失函数
loss_func = nn.CrossEntropyLoss()

# 定义优化方法，此处使用随机梯度下降法
optimizer_ft = optim.SGD(model.parameters(), lr=LR, momentum=MM)
# 定义优化方法，此处采取 adam 优化算法
# optimizer_ft = optim.Adam(model.parameters(), lr=LR, betas=(0.9, 0.999), eps=1e-
08, weight_decay=0)

# 定义每 5 个 epoch，学习率变为之前的 0.1
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=5, gamma=0.1)
```

```python
# 训练神经网络
def train_model(model, criterion, optimizer, scheduler):
    model.train()

    running_loss = 0.0
    running_corrects = 0

    for inputs, labels in tqdm(train_dataloader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)
    scheduler.step()

    epoch_loss = running_loss / len(train_dataset)
    epoch_acc = running_corrects.double() / len(train_dataset)

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

def val_model(model, criterion):  # 这是利用验证集对每一轮训练完成的模型进行检验
    model.eval()  # 不启用 Batch Normalization 和 Dropout。
    running_loss = 0.0
    running_corrects = 0.0
    with torch.no_grad():
        for inputs, labels in tqdm(val_dataloader):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            loss = criterion(outputs, labels)

            running_loss += loss.item() * inputs.size(0)
```

```
            running_corrects += torch.sum(preds == labels.data)

    epoch_loss = running_loss / len(val_dataset)
    epoch_acc = running_corrects.double() / len(val_dataset)

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

def test_model(model, criterion):  # 这是利用测试集进行最终的对模型效果的测试
    model.eval()  # 不启用 Batch Normalization 和 Dropout。
    running_loss = 0.0
    running_corrects = 0
    with torch.no_grad():
        for inputs, labels in tqdm(test_dataloader):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            loss = criterion(outputs, labels)

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

    epoch_loss = running_loss / len(test_dataset)
    epoch_acc = running_corrects.double() / len(test_dataset)

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

def pre_model(model, criterion):  # 这是利用训练好的模型对单张图片进行预测
    model.eval()  # 不启用 Batch Normalization 和 Dropout。
    fig = plt.figure()
    with torch.no_grad():
        for inputs, labels in pre_dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            PreLable = labels.data.item()
            if preds == labels.data:
```

```
                if PreLable == 1:
                    print("man\n")
                    return "man"
                    sys.exit()
                else:
                    print("woman\n")
                    return "woman"
                    sys.exit()
            else:
                if PreLable == 0:
                    print("man\n")
                    return "man"
                    sys.exit()
                else:
                    print("woman\n")
                    return "woman"
                    sys.exit()

class Myframe(wx.Frame):
    def __init__(self, filename, stt):
        wx.Frame.__init__(self, None, -1, u'Predict', size=(640, 740))
        self.filename = filename
        self.Bind(wx.EVT_SIZE, self.change)
        self.p = wx.Panel(self, -1)
        self.SetBackgroundColour('white')

        text = wx.StaticText(self, -1, stt, (230, 50))
        font = wx.Font(20, wx.ROMAN, wx.NORMAL, wx.BOLD)
        text.SetFont(font)

    def start(self):
        self.p.DestroyChildren()  # 抹掉原先显示的图片
        self.width, self.height = self.GetSize()
        self.height += 100
        image = Image.open(self.filename)
        self.x, self.y = image.size
        self.x = self.width / 2 - self.x / 2
        self.y = self.height / 2 - self.y / 2
        self.pic = wx.Image(self.filename, wx.BITMAP_TYPE_ANY).ConvertToBitmap()
        # 通过计算获得图片的存放位置
        self.button = wx.BitmapButton(self.p, -
1, self.pic, pos=(int(self.x), int(self.y)))
        self.p.Fit()
```

```
    def change(self, size):  # 如果检测到框架大小的变化，及时改变图片的位置
        if self.filename != "":
            self.start()
        else:
            pass

# 训练和测试
if __name__ == "__main__":
    since = time.time()
    best_acc = 0.0

    model.load_state_dict(torch.load('model-0810-7.pkl'))

    for epoch in range(EPOCH):
        print('\nEpoch {}/{}'.format(epoch, EPOCH - 1))
        print('-' * 10)

        model = train_model(model, loss_func, optimizer_ft, exp_lr_scheduler)
        epoch_acc = val_model(model, loss_func)
        if epoch_acc > best_acc:
            torch.save(model.state_dict(), 'model-0810-9.pkl.pkl')
        best_acc = epoch_acc if epoch_acc > best_acc else best_acc
    if EPOCH != 0:
        time_elapsed = time.time() - since
        print('\nTraining complete in {:.0f}m {:.0f}s'.format(
            time_elapsed // 60, time_elapsed % 60))
        print('Best Val Acc: {:4f}'.format(best_acc))

        print("\n\n\n******test begin: ******")
        test_acc = test_model(model, loss_func)
        print('Test Acc: {:4f}'.format(test_acc))

# 以下均为单张图片的预测程序
    if EPOCH == 0:
        mkdirName4 = 'H:\\人脸识别\\torch1.9.0+anaconda 环境版本\\data\\pre\\female'
        shutil.rmtree(mkdirName4)
        os.mkdir(mkdirName4)
        file_dir = "./Photo_To_Predict"
        for root, dirs, files in os.walk(file_dir, topdown=False):
            #print(root)  # 当前目录路径
            # print(dirs)  # 当前目录下所有子目录
            # print(files[0])  # 当前路径下所有非目录子文件
            photoName = files[0]
            filename2 = root + '/' + files[0]
```

```
            shutil.move(filename2, 'H:\\人脸识别\\torch1.9.0+anaconda 环境版本
\\data\\pre\\female')

        pre_dataset = datasets.ImageFolder(root='./data/pre', transform=data_transf
orms["test"])  # 加载预测集
        pre_dataloader = DataLoader(dataset=pre_dataset,
                                    batch_size=1,
                                    shuffle=False,  # 在预测集中不需要每次对数据进行重
新排序
                                    num_workers=1)  # 有几个进程来处理 data loading

        stt = pre_model(model, loss_func)

        app = wx.App()
        filename3 = './data/pre/female/'+photoName
        frame = Myframe(filename3, "Predict: " + stt)
        frame.start()
        frame.Center()
        frame.Show()
        app.MainLoop()

# torch.save(model.state_dict(),'model.pkl')
```