**程序报告**

学号：2212998 姓名：胡博浩

1. **问题重述**

今年一场席卷全球的新型冠状病毒给人们带来了沉重的生命财产的损失。

有效防御这种传染病毒的方法就是积极佩戴口罩。

我国对此也采取了严肃的措施，在公共场合要求人们必须佩戴口罩。

在本次实验中，我们要建立一个目标检测的模型，可以识别图中的人是否佩戴了口罩。

**实验要求：**

1）建立深度学习模型，检测出图中的人是否佩戴了口罩，并将其尽可能调整到最佳状态。

2）学习经典的模型 MTCNN 和 MobileNet 的结构。

3）学习训练时的方法。

实验环境：

可以使用基于 Python 的 OpenCV 、PIL 库进行图像相关处理，使用 Numpy 库进行相关数值运算，使用 Pytorch 等深度学习框架训练模型等。

1. **设计思想**

首先介绍一下代码的思路过程，主要分为数据预处理、加载模型、训练模型以及预测四个部分：

1.数据预处理：

对数据进行优化，分别比对了添加了图片一定角度旋转和归一化时候更改新参数的测试结果。具体如下：

1）先定义好对图片进行预处理的操作transform ，原始图片作为输入，返回一个转换后的图片。

2）调用torchvision.datasets.ImageFolder函数，传入图片的根目录和transform预处理函数。

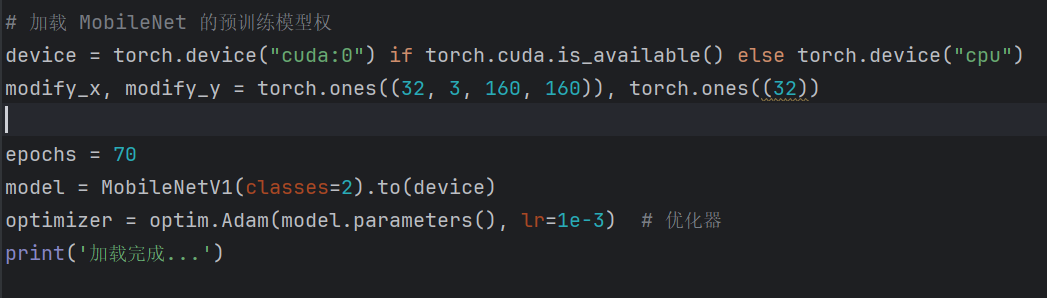
3）然后划分数据集，根据参数test\_split的比例划分训练和测试集

4）最后创建DataLoader对象，设置每一个batch加载样本的组数为传入的batch\_size参数，并且每一个 epoch 之后对样本进行随机打乱。



2.加载预训练模型：

主要为三步：调用预处理函数，将数据集划分；调用torch.py 里写的MobileNetV1网络，传入classes为2，并且调大epoch；设置优化器 optimizer



3.训练模型：

首先调整学习率，然后训练模型：

1）设置外层大循环，共进行 epochs 次训练；

2）设置内层循环，进行分批迭代；

3）在每次训练中，更新记录截至目前为止最好的模型状态



4.预测：

即先加载模型，然后将 cv2.imread 图像转化为 PIL.Image 图像，用来兼容测试输入的 cv2 读取的图像，之后利用训练好的模型进行口罩佩戴识别的预测。



然后说明一下实验的调优过程：

1.选择模型：

我对应用的模型进行比对测试，寻找针对口罩检测表现更加良好的模型，对比了resnet18和resnet50和MobilenetV1模型，经过测试发现还是MobileNetV1模型在这个应用上表现良好。

2.调整epoch大小：

从10开始以10为跨度进行调整，直到epochs达到80，具体测试结果这里不进行展示，但是最后发现epochs等于70的时候效果比较好。

3.调整优化器：

这里我选择用optim.Adam，对于里面的参数lr，我分别测试了1e-3、2e-3、3e-3、9e-4、1e-4等，最后选择了比较好的le-4。

4.调整学习率：

其中optimizer是要调整学习率的优化器对象；max指定验证集上监测指标的模式。在这种情况下，调度器将监测验证集上的指标，并在其不再增加时降低学习率；factor是学习率降低的因子。当验证指标停止改善时，学习率将乘以factor；patience是检测验证指标停止改善的持续轮数，达到patience时候将降低学习率。

最终经过不断测试，选择了初始的factor=0.5,patience=2。

5.调整batch\_size大小：

经过不断测试，最终选择了128作为最终参数

1. **代码内容**

**1. train.py**

import warnings  
  
# 忽视警告  
warnings.filterwarnings('ignore')  
  
import cv2  
from PIL import Image  
import numpy as np  
import copy  
import matplotlib.pyplot as plt  
from tqdm.auto import tqdm  
import torch  
import torch.nn as nn  
import torch.optim as optim  
from torchvision.datasets import ImageFolder  
import torchvision.transforms as T  
from torch.utils.data import DataLoader  
  
from torch\_py.Utils import plot\_image  
from torch\_py.MTCNN.detector import FaceDetector  
from torch\_py.MobileNetV1 import MobileNetV1  
from torch\_py.FaceRec import Recognition  
  
torch.set\_num\_threads(6)  
  
  
def processing\_data(data\_path, height=224, width=224, batch\_size=32,  
 test\_split=0.1):  
 *"""  
 数据处理部分  
 :param data\_path: 数据路径  
 :param height:高度  
 :param width: 宽度  
 :param batch\_size: 每次读取图片的数量  
 :param test\_split: 测试集划分比例  
 :return:  
 """* transforms = T.Compose([  
 T.Resize((height, width)),  
 T.RandomHorizontalFlip(0.1), # 进行随机水平翻转  
 T.RandomVerticalFlip(0.1), # 进行随机竖直翻转  
 T.ToTensor(), # 转化为张量  
 T.Normalize([0], [1]), # 归一化  
 ])  
  
 dataset = ImageFolder(data\_path, transform=transforms)  
 # 划分数据集  
 train\_size = int((1 - test\_split) \* len(dataset))  
 test\_size = len(dataset) - train\_size  
 train\_dataset, test\_dataset = torch.utils.data.random\_split(dataset, [train\_size, test\_size])  
 # 创建一个 DataLoader 对象  
 train\_data\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  
 valid\_data\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=True)  
  
 return train\_data\_loader, valid\_data\_loader  
  
  
data\_path = './datasets/5f680a696ec9b83bb0037081-momodel/data/image'  
train\_data\_loader, valid\_data\_loader = processing\_data(data\_path=data\_path, height=160, width=160, batch\_size=128,  
 test\_split=0.1)  
  
# 加载 MobileNet 的预训练模型权  
device = torch.device("cuda:0") if torch.cuda.is\_available() else torch.device("cpu")  
modify\_x, modify\_y = torch.ones((32, 3, 160, 160)), torch.ones((32))  
  
epochs = 70  
model = MobileNetV1(classes=2).to(device)  
optimizer = optim.Adam(model.parameters(), lr=1e-4，weight\_decay=1e-8) # 优化器  
print('加载完成...')  
  
# 学习率下降的方式，acc三次不下降就下降学习率继续训练，衰减学习率  
scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer,  
 'max',  
 factor=0.5,  
 patience=2)  
# 损失函数  
criterion = nn.CrossEntropyLoss()  
  
best\_loss = 1e9  
best\_acc = 0  
best\_model\_weights = copy.deepcopy(model.state\_dict())  
loss\_list = [] # 存储损失函数值  
for epoch in range(epochs):  
 model.train()  
  
 for batch\_idx, (x, y) in tqdm(enumerate(train\_data\_loader, 1)):  
 x = x.to(device)  
 y = y.to(device)  
 pred\_y = model(x)  
  
 loss = criterion(pred\_y, y)  
  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
 # if loss < best\_loss:  
 # best\_model\_weights = copy.deepcopy(model.state\_dict())  
 # best\_loss = loss  
  
 total = 0  
 right\_cnt = 0  
 for b\_x, b\_y in valid\_data\_loader:  
 b\_x = b\_x.to(device)  
 b\_y = b\_y.to(device)  
 output = model(b\_x)  
 pred\_y = torch.max(output, 1)[1]  
 right\_cnt += (pred\_y == b\_y).sum()  
 total += b\_y.size(0)  
  
 accuracy = right\_cnt.float() / total  
 print('val acc:', accuracy)  
  
 if accuracy > best\_acc:  
 best\_model\_weights = copy.deepcopy(model.state\_dict())  
 best\_acc = accuracy  
 # loss\_list.append(loss)  
  
# print('step:' + str(epoch + 1) + '/' + str(epochs) + ' || Total Loss: %.4f' % (loss, ))  
# print('best\_loss: %.4f' % (best\_loss, ))  
print('best\_accuracy: %.4f' % (best\_acc,))  
torch.save(best\_model\_weights, './results/temp.pth')  
print('Finish Training.')

**2. main.py**

import warnings  
  
# 忽视警告  
warnings.filterwarnings('ignore')  
  
import cv2  
from PIL import Image  
import numpy as np  
import copy  
import matplotlib.pyplot as plt  
from tqdm.auto import tqdm  
import torch  
import torch.nn as nn  
import torch.optim as optim  
from torchvision.datasets import ImageFolder  
import torchvision.transforms as T  
from torch.utils.data import DataLoader  
  
from torch\_py.Utils import plot\_image  
from torch\_py.MTCNN.detector import FaceDetector  
from torch\_py.MobileNetV1 import MobileNetV1  
from torch\_py.FaceRec import Recognition  
  
from torch\_py.Utils import plot\_image  
from torch\_py.MTCNN.detector import FaceDetector  
from torch\_py.MobileNetV1 import MobileNetV1  
from torch\_py.FaceRec import Recognition  
from torch\_py.FaceRec import Recognition  
from PIL import Image  
import cv2  
  
# -------------------------- 请加载您最满意的模型 ---------------------------  
# 加载模型(请加载你认为的最佳模型)  
# 加载模型,加载请注意 model\_path 是相对路径, 与当前文件同级。  
# 如果你的模型是在 results 文件夹下的 dnn.h5 模型，则 model\_path = 'results/temp.pth'  
model\_path = 'results/temp.pth'  
  
  
# ---------------------------------------------------------------------------  
  
def predict(img):  
 *"""  
 加载模型和模型预测  
 :param img: cv2.imread 图像  
 :return: 预测的图片中的总人数、其中佩戴口罩的人数  
 """* # -------------------------- 实现模型预测部分的代码 ---------------------------  
 # 将 cv2.imread 图像转化为 PIL.Image 图像，用来兼容测试输入的 cv2 读取的图像（勿删！！！）  
 # cv2.imread 读取图像的类型是 numpy.ndarray  
 # PIL.Image.open 读取图像的类型是 PIL.JpegImagePlugin.JpegImageFile  
 if isinstance(img, np.ndarray):  
 # 转化为 PIL.JpegImagePlugin.JpegImageFile 类型  
 img = Image.fromarray(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))  
  
 recognize = Recognition(model\_path)  
 img, all\_num, mask\_num = recognize.mask\_recognize(img)  
 # -------------------------------------------------------------------------  
 return all\_num, mask\_num

1. **实验结果**

在Mo环境下进行代码运行测试，效果不错。



1. **总结**

在口罩识别的实验中，我取得了较高的正确率，达到了预期目标。

为了进一步提升模型性能，可以从多个方面着手。首先，在数据处理方面，计划尝试更多的数据增强技术，以增强模型的泛化能力。其次，考虑尝试使用其他更先进的模型架构，如 EfficientNet 等，以获得更好的性能表现。同时，也可以在训练过程中采用更复杂的学习率调整策略，如余弦退火或分段线性学习率等，以进一步提高模型的收敛速度和性能。

在实验过程中，我遇到了一些挑战，其中包括模型训练所需的大量计算资源，以及调参过程中的耗时和耗力。为了克服这些困难，我利用了molab平台的GPU时间进行模型训练，充分利用了2小时GPU时间。在调参过程中，我提交了数十次来进行参数调整，努力寻找最佳超参数组合。

此外，我考虑了一些额外的优化方向，如数据集增强和扩充、模型集成、端到端的检测与识别模型、引入更大规模的数据集、模型优化和推理加速，以及针对口罩样式和遮挡的处理。这些优化方向可以进一步提高模型的性能和鲁棒性。

总的来说，通过这次口罩识别实验，我获得了宝贵的经验和知识。我意识到了数据集的重要性以及计算资源的关键作用，并且对模型的选择、超参数调整和优化方向有了更清晰的认识。虽然实验过程中遇到了一些挑战，但通过不懈的努力和实践，我取得了令人满意的结果，并且为进一步学习人工智能算法奠定了坚实的基础。