程序报告

学号： 2113997　　姓名：齐明杰

一、问题重述

（简单描述对问题的理解，从问题中抓住主干，必填）

==========

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。

二、设计思想

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

==========

所采用的方法：本实验采用了 ResNet50 作为基本模型进行口罩识别任务，而不是 MobileNetV1。ResNet50 是一个具有 50 层的深度卷积神经网络，由于其具有残差连接结构，可以有效避免梯度消失问题，使模型在保持较高性能的同时具有较好的泛化能力。

改进：在本实验中，我们对 ResNet50 的全连接层进行了修改，以适应二分类任务（戴口罩和不戴口罩）。同时，我们采用了针对性的数据增强技术，如 ColorJitter、RandomAffine 和 RandomPerspective 等，以提高模型在不同场景和口罩样式上的泛化能力。

优化方向：

参数调整：可以通过调整学习率、权重衰减等超参数来优化模型性能。

框架调整：可以尝试使用更深或更浅的网络结构，或者尝试其他类型的模型，如 MobileNet、EfficientNet 等。

方法局限性和常见问题：ResNet50 作为一个较深的网络，在计算资源和时间有限的情况下，可能会导致训练速度较慢。此外，当数据集规模较小或存在类别不平衡时，可能会出现过拟合现象。

伪代码:

1. 加载数据集并进行预处理
2. 创建 ResNet50 模型并修改全连接层
3. 定义优化器、学习率调整策略和损失函数
4. 训练模型：
   a. 遍历所有训练批次
   b. 执行前向传播
   c. 计算损失
   d. 反向传播并更新权重
5. 验证模型
6. 保存最佳模型权重

三、代码内容

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

main.py

```
================================================================================
import warnings
warnings.filterwarnings('ignore')
from torch_py.FaceRec import Recognition
from PIL import Image
import numpy as np
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
```

train.py

```
================================================================================
import warnings
import copy
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 torchvision import models

# 忽略警告
warnings.filterwarnings('ignore')

# 设置线程数量
torch.set_num_threads(6)

# 数据处理部分
def processing_data(data_path, height=224, width=224, batch_size=32, test_split=0.1):
    transforms = T.Compose([
        T.Resize((height, width)),
        T.RandomHorizontalFlip(0.1),
        T.RandomVerticalFlip(0.1),
        T.RandomRotation(15),
        T.RandomResizedCrop(height, scale=(0.8, 1.0)),
        T.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
        T.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.9, 1.1), shear=10),
        T.RandomPerspective(distortion_scale=0.3, p=0.5),
        T.ToTensor(),
        T.Normalize([0], [1]),
    ])


    dataset = ImageFolder(data_path, transform=transforms)
    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])
    train_data_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    valid_data_loader = DataLoader(test_dataset, batch_size=1000, shuffle=True)

    return train_data_loader, valid_data_loader

# 载入数据
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=32, test_split=0.2)

device = torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
```

```
# 定义模型、优化器和损失函数
model = models.resnet50(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)


model = model.to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'max', factor=0.5, patience=2)
criterion = nn.CrossEntropyLoss()

# 训练和验证
epochs = 10
best_acc = 0
best_model_weights = copy.deepcopy(model.state_dict())

for epoch in range(epochs):
    model.train()
    running_loss = 0.0

    for x, y in tqdm(train_data_loader):
        x = x.to(device)
        y = y.to(device)
        pred_y = model(x)
        loss = criterion(pred_y, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    train_loss = running_loss / len(train_data_loader)

    model.eval()
    total = 0
    right_cnt = 0
    valid_loss = 0.0

    with torch.no_grad():
        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)
                loss = criterion(output, b_y)
                valid_loss += loss.item()
                pred_y = torch.max(output, 1)[1]
                right_cnt += (pred_y == b_y).sum()
                total += b_y.size(0)

        valid_loss = valid_loss / len(valid_data_loader)
        accuracy = right_cnt.float() / total
        print(f'Epoch: {epoch+1}/{epochs} || Train Loss: {train_loss:.4f} || Val Loss: {valid_loss:.4f}
|| Val Acc: {accuracy:.4f}')
        # 更新学习率
        scheduler.step(valid_loss)
        # 保存最佳模型权重
        if accuracy > best_acc:
            best_model_weights = copy.deepcopy(model.state_dict())
            best_acc = accuracy
            torch.save(best_model_weights, './results/temp.pth')

print(f'Best Accuracy: {best_acc:.4f}')
print('Finish Training.')
```

test.py

```
================================================================================
import warnings
warnings.filterwarnings('ignore')
from torch_py.FaceRec import Recognition, plot_image
from PIL import Image
import numpy as np
import cv2

model_path = 'results/temp.pth'

def predict(img):
        if isinstance(img, np.ndarray):
            img = Image.fromarray(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
        recognize = Recognition(model_path)
        img, all_num, mask_num = recognize.mask_recognize(img)
        return img, all_num, mask_num

img = cv2.imread("./test1.jpg")
img, all_num, mask_nums = predict(img)
plot_image(img)
print("图中的人数有：" + str(all_num) + "个")
print("戴口罩的人数有：" + str(mask_nums) + "个")
```

四、实验结果

（实验结果，必填）

==========

在本实验中，经过训练和验证，我们的 ResNet50 模型在口罩识别任务上取得了非常高的分数(98 分)。这表明所采用的方法在解决口罩识别问题上具有较好的效果，如下图：

同时，利用 test.py 在测试图片上的结果如下:

五、总结

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

==========

是否达到目标预期：根据实验结果，我在口罩识别任务上取得了较高的正确率。因此，可以认为实验达到了目标预期。

可能改进的方向：

在数据处理方面，我可以尝试更多的数据增强技术，以进一步提高模型的泛化能力。

在模型结构方面，我可以尝试其他更先进的模型架构（如 EfficientNet 系列），以获得更好的性能。

在训练过程中，我可以采用更加复杂的学习率调整策略，如余弦退火或者分段线性学习率等。
实现过程中遇到的困难：（请根据您的实际经历添加具体内容，例如，模型训练速度较慢、数据集不平衡导致的过拟合问题等）

从哪些方面我可以提升性能：
优化数据处理和增强技术，以更好地适应口罩识别任务的特点。
调整超参数，例如学习率、权重衰减等，以提高模型性能。
尝试使用更适合口罩识别任务的模型结构。

模型的超参数和框架搜索是否合理：在实验中，我使用了合适的数据处理方法和预训练的 ResNet50 模型。同时，采用了 Adam 优化器和学习率衰减策略。这些选择在实验中取得了较好的结果。尽管如此，为了进一步提高模型性能，仍可以尝试在超参数调整和模型框架搜索方面进行优化。