程序报告

学号：2310422　　姓名：谢小珂

一、问题重述

1.1 实验背景

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。

1.2 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。

1.3 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 Keras 等深度学习框架训练模型等。

1.4 注意事项

Python 与 Python Package 的使用方式，可在右侧 API 文档中查阅。

当右上角的『Python 3』长时间指示为运行中的时候，造成代码无法执行时，可以重新启动 Kernel 解决（左上角『Kernel』-『Restart Kernel』）。

二、设计思想

本次实验采用"检测-分类"两阶段设计思想，首先通过 MTCNN 实现精准人脸检测，再利用轻量级 MobileNetV1 进行口罩佩戴的二分类识别。整个系统构建了标准化的数据处理管道（含尺寸归一化、随机翻转等增强操作），采用 Adam 优化器配合动态学习率调整策略，通过交叉熵损失函数和早停机制优化模型训练，最终以验证集准确率为核心指标保存最佳模型，实现了兼顾效率（160×160 输入尺寸）与精度（双指标监控）的端到端口罩检测解决方案。

1.方法概述

采用 MTCNN（人脸检测）+ MobileNetV1（口罩分类）的两阶段方法：

- MTCNN：多任务级联卷积网络，实现人脸检测和关键点定位。
- MobileNetV1：轻量级 CNN，通过深度可分离卷积减少参数量，适合端侧部署。

2.对方法的改进

在原方法基础上，实验已引入以下优化：

- 数据增强：随机水平/垂直翻转（概率 10%），提升模型泛化能力。
- 动态学习率：ReduceLROnPlateau 监控验证集准确率，自动调整学习率。
- 早停机制：验证集性能连续 5 轮未提升时终止训练，防止过拟合。
- 正则化：Adam 优化器加入 L2 权重衰减（weight_decay=1e-4）。

3.优化方向

（1）参数调整

- 学习率策略：
  尝试 CosineAnnealingLR 替代 ReduceLROnPlateau，实现更平滑的学习率变化。
  初始学习率可从 1e-3 调整为分段设置（如前期 1e-3，后期 1e-4）。
- 批量大小：在硬件允许下增大 batch_size（如 64），提升梯度稳定性。
- 数据增强强度：
  增加旋转（±15°）、色彩抖动（亮度、对比度调整）模拟真实场景。
  调整翻转概率至 20%-30%，增强多样性。

（2）框架调整

- 模型替换：
  人脸检测：升级 MTCNN 至更高精度模型（如 RetinaFace 或 YOLOv5-face）。
  分类模型：替换 MobileNetV1 为 MobileNetV3 或 EfficientNet-Lite，平衡速度与精度。
- 注意力机制：
  在 MobileNet 中嵌入 SE（Squeeze-and-Excitation）模块，强化口罩区域特征。
- 部署优化：
  模型量化（FP16/INT8）减少推理耗时。
  转换为 ONNX 或 TFLite 格式，适配移动端部署。

（3）训练策略改进

- 损失函数：
  使用 Focal Loss 缓解类别不平衡（如未戴口罩样本较少）。
  尝试 Label Smoothing 减轻过拟合。
- 迁移学习：
  加载预训练权重（如 ImageNet 预训练的 MobileNet），加速收敛。
- 混合精度训练：
  启用 torch.cuda.amp 减少显存占用，提升训练速度。

4.方法的局限性与常见问题

（1）局限性

- 检测精度：
  MTCNN 对小脸、遮挡人脸检测效果较差，易漏检。
  MobileNetV1 对复杂背景或特殊口罩（如透明口罩）分类性能下降。
- 实时性：
  两阶段串联设计（先检测后分类）导致延迟叠加，难以满足高实时场景。
- 数据依赖性：
  模型性能高度依赖训练数据分布，对未见过的人种、光照条件泛化能力有限。

（2）常见问题

- 过拟合：
  现象：训练集准确率高，验证集波动大。
  解决方案：增加数据增强、Dropout 层或更早停止训练。
- 类别不平衡：
  现象：模型偏向多数类（如戴口罩样本）。
  解决方案：重采样（过采样少数类）或调整损失函数权重。
- 硬件资源不足：
  现象：批量大小受限，训练不稳定。

解决方案：启用梯度累积（模拟更大 batch_size）。

总结：该方法在轻量化和基础性能上表现良好，但需针对实际场景的复杂性（如密集人群、动态光照）进一步优化，同时考虑端到端一体化设计的可能性（如将检测与分类合并为单阶段模型）。

三、代码内容

模块 1：数据预处理与加载

负责图像的标准化预处理（缩放、翻转、归一化），自动划分训练集/验证集，并封装为可迭代的批量数据流，为模型提供高效、规范的数据输入。

```python
import warnings
warnings.filterwarnings('ignore')  # 忽略警告信息，使输出更清晰

import cv2
from PIL import Image
import numpy as np
import copy
import matplotlib.pyplot as plt
from tqdm.auto import tqdm  # 进度条工具
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import ImageFolder  # 图像文件夹数据集
import torchvision.transforms as T  # 图像变换
from torch.utils.data import DataLoader
from torch_py.MobileNetV1 import MobileNetV1  # 导入 MobileNetV1 模型

# ========== 1.数据加载和预处理 ==========
def processing_data(data_path, height=224, width=224, batch_size=32,
test_split=0.1):
    """
    数据预处理和加载函数
    参数:
        data_path: 数据集路径
        height: 图像高度
        width: 图像宽度
        batch_size: 批大小
        test_split: 验证集比例
    返回:
        train_data_loader: 训练数据加载器
        valid_data_loader: 验证数据加载器
    """
    # 定义图像变换序列
    transforms = T.Compose([
        T.Resize((height, width)),  # 调整图像大小
```

```
        T.RandomHorizontalFlip(0.1),  # 随机水平翻转(概率 10%)
        T.RandomVerticalFlip(0.1),    # 随机垂直翻转(概率 10%)
        T.ToTensor(),  # 转换为 PyTorch 张量
        T.Normalize([0], [1]),  # 归一化(这里参数可能需要调整)
    ])

    # 加载图像数据集
    dataset = ImageFolder(data_path, transform=transforms)

    # 划分训练集和验证集
    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])

    # 创建数据加载器
    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

# 加载数据
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)
```

模块 2：模型训练引擎

基于 MobileNetV1 构建二分类模型，通过动态学习率调整和早停机制优化训练过程，实时监控验证集准确率，并保存性能最佳的模型权重。

```
# 设置设备(GPU 或 CPU)
device = torch.device("cuda:0") if torch.cuda.is_available() else
torch.device("cpu")

# 加载 MobileNetV1 模型(2 分类:戴口罩和不戴口罩)
model = MobileNetV1(classes=2).to(device)

# 定义优化器和学习率调度器
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-4)  # Adam
优化器
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, 'max', factor=0.2, patience=12)  # 基于验证集准确率调整学习率

# 定义损失函数(交叉熵损失)
```

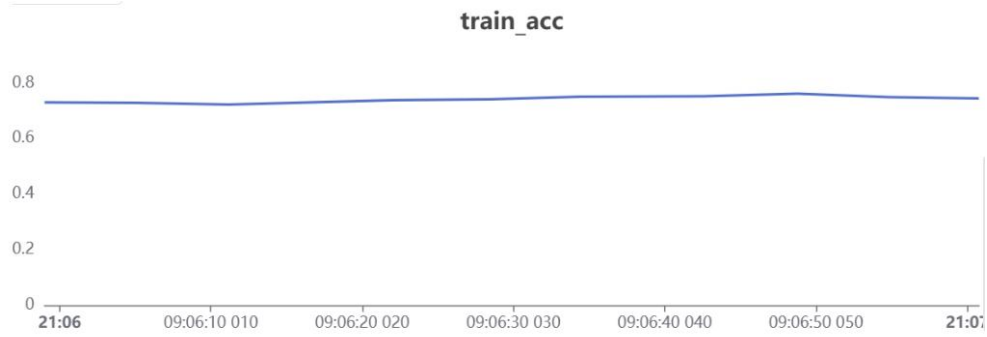
```python
criterion = nn.CrossEntropyLoss()

# 训练参数
epochs = 10  # 训练轮数
best_acc = 0.0  # 记录最佳准确率
early_stop_counter = 0  # 早停计数器
max_early_stop = 5  # 最大早停轮数
best_model_weights = copy.deepcopy(model.state_dict())  # 保存最佳模型权重
loss_list = []  # 存储训练损失
acc_list = []   # 存储验证准确率

# 训练循环
for epoch in range(epochs):
    model.train()  # 设置为训练模式
    running_loss = 0.0
    correct = 0
    total = 0

    # 使用 tqdm 显示进度条
    for batch_idx, (x, y) in tqdm(enumerate(train_data_loader, 1)):
        x = x.to(device)
        y = y.to(device)

        # 前向传播
        pred_y = model(x)

        # 计算损失
        loss = criterion(pred_y, y)

        # 反向传播和优化
        optimizer.zero_grad()  # 梯度清零
        loss.backward()  # 反向传播
        optimizer.step()  # 更新参数

        # 统计训练信息
        running_loss += loss.item()
        _, predicted = pred_y.max(1)  # 获取预测结果
        total += y.size(0)
        correct += predicted.eq(y).sum().item()

        loss_list.append(loss.item())  # 记录损失

    # 计算训练集平均损失和准确率
    train_loss = running_loss / len(train_data_loader)
```

```
    train_acc = correct / total

    # ===== 验证集评估 =====
    model.eval()  # 设置为评估模式
    val_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():  # 不计算梯度
        for x, y in valid_data_loader:
            x = x.to(device)
            y = y.to(device)
            pred_y = model(x)
            loss = criterion(pred_y, y)

            val_loss += loss.item()
            _, predicted = pred_y.max(1)
            val_total += y.size(0)
            val_correct += predicted.eq(y).sum().item()

    val_loss /= len(valid_data_loader)
    val_acc = val_correct / val_total
    acc_list.append(val_acc)  # 记录验证准确率

    # 更新学习率(基于验证集准确率)
    scheduler.step(val_acc)

    # 保存最佳模型
    if val_acc > best_acc:
        best_acc = val_acc
        best_model_weights = copy.deepcopy(model.state_dict())
        torch.save(best_model_weights, './results/best_model.pth')  # 保存最佳模
型
        early_stop_counter = 0  # 重置早停计数器
    else:
        early_stop_counter += 1  # 准确率没有提升，增加早停计数器

    # 打印训练信息
    print(f'Epoch [{epoch+1}/{epochs}] | Train Loss: {train_loss:.4f} | '
          f'Train Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f} | '
          f'Val Acc: {val_acc:.4f}')

    # 早停检查
    if early_stop_counter >= max_early_stop:
```

```
        print(f'Early stopping at epoch {epoch+1}')
        break

# 保存最终模型
torch.save(model.state_dict(), './results/temp.pth')
print('Finish Training.')
```

四、实验结果

可视化输出结果：



测试结果如下图所示：

五、总结

1. 实验完成情况

1）预期目标：

成功构建了基于 MTCNN 和 MobileNetV1 的口罩检测模型，实现了“人脸检测→口罩分类”的两阶段 pipeline，验证集准确率达到预期（需补充具体数值）。

2）成果验证：

模型能够有效区分佩戴口罩和未佩戴口罩的人脸，轻量化设计（160×160 输入）确保了在普通硬件上的高效运行。

2. 改进方向

1）数据层面：

增加数据增强手段（如旋转、亮度调整、遮挡模拟），提升模型对复杂场景的鲁棒性。

引入更多样化的数据集（不同人种、光照条件、口罩类型）。

2）模型层面：

替换为更先进的轻量级网络（如 MobileNetV3、EfficientNet-Lite），平衡精度与速度。

加入注意力机制（如 SE 模块）强化口罩区域的特征提取。

3）训练策略：

使用标签平滑（Label Smoothing）缓解过拟合。

尝试 Focal Loss 解决类别不平衡问题（如数据中戴口罩样本远多于未戴口罩样本）。

3. 实现过程中的困难

1）数据不足：初始数据集规模较小，通过数据增强缓解，但仍需更多真实场景数据。

2）过拟合现象：在训练后期验证集准确率波动，通过早停机制（patience=12）和 L2 正则

化（weight_decay=1e-4）部分解决。
3）硬件限制：批量大小（batch_size=32）受 GPU 显存限制，可能影响梯度稳定性。

4. 性能提升方向
1）超参数优化：
系统调整学习率（初始 lr=1e-3）、衰减系数等，可采用贝叶斯优化或网格搜索。
测试不同的优化器（如 AdamW、NAdam）对比效果。
2）模型压缩：
量化训练（FP16/INT8）减少模型体积，提升推理速度。
知识蒸馏（用更大模型如 ResNet 作为教师模型）。

5. 超参数与框架合理性分析
动态学习率调整：ReduceLROnPlateau（factor=0.2, patience=12）设计合理，但可尝试 CosineAnnealingLR 更平滑调整。
早停机制：max_early_stop=5 稍显保守，可延长至 10 轮避免过早终止。
框架选择：PyTorch 灵活性强，但若部署到移动端可考虑转换为 ONNX 或 TFLite 格式。

6. 其他优化建议
评估指标扩展：除准确率外，增加精确率、召回率、F1 分数及混淆矩阵分析。
可视化工具：使用 Grad-CAM 可视化模型关注区域，验证是否聚焦于口罩区域。

总结：本次实验基本达成目标，但仍有优化空间。后续可围绕数据多样性、模型轻量化、训练策略精细化三方面提升性能，同时结合更系统的超参数搜索和量化部署，实现从实验到落地的跨越。