程序报告

学号：2212046　　姓名：王昱

一、问题重述

1.1 垃圾短信检测问题：

口罩佩戴检测问题是一个计算机视觉和深度学习应用领域的典型任务，其目标是自动检测和识别人脸上是否佩戴了口罩。这一任务在公共健康安全，尤其是在疫情防控中具有重要意义。

==========

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 对问题的理解

口罩佩戴检测旨在通过图像或视频流判断人脸是否正确佩戴口罩。这项技术在疫情防控和公共卫生中具有重要意义。我计划通过收集多样化的标注数据，使用数据增强技术和预训练模型（如 MobileNet），结合优化算法（如 Adam），并进行模型调优和交叉验证，以实现高效、准确的检测系统，确保在公共场所的实时监测。

二、设计思想

此代码的设计思想是创建一个深度学习训练流水线，使用预训练的模型对图像分类任务进行微调。我们通过数据增强技术来增加训练数据的多样性，选择合适的优化器和学习率调度器来优化模型参数，从而提高模型的训练准确率和泛化能力。

2.1 训练准备

首先导入相关的库，导入深度学习和数据处理所需的库，如 PyTorch 和 torchvision。

```
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
```

设置环境: 忽略警告信息，并设置多线程数以提高数据加载的效率

```
# 设置线程数量
warnings.filterwarnings('ignore')
torch.set_num_threads(6)
```

2.2 加载数据与数据处理

- **使用到的方法**

（1）数据增强: 使用 torchvision.transforms 对图像数据进行多种增强操作（如翻转、旋转、颜色抖动等），以增加训练数据的多样性。

（2）数据集划分: 使用 torch.utils.data.random_split 将数据集划分为训练集和测试集。

（3）数据加载: 使用 torch.utils.data.DataLoader 创建数据加载器，以便在训练过程中高效地批量加载数据。

- **创建一个 Dataset 对象：使用通用的数据加载器——torchvision.datasets.ImageFolder**

（1）先定义好对图片进行预处理的操作（函数） 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。

- **创建一个 DataLoader 对象：使用 torch.utils.data.DataLoader 包装数据**

（1）先划分数据集，根据参数 test_split 的比例划分训练和测试集。

（2）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。

```
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(0.5),
        T.RandomVerticalFlip(0.5),
        T.RandomRotation(30),
        T.RandomResizedCrop(height, scale=(0.8, 1.0)),
        T.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.2),
        T.RandomAffine(degrees=30, translate=(0.2, 0.2), scale=(0.8, 1.2), shear=20),
        T.RandomPerspective(distortion_scale=0.5, p=0.5),
        T.ToTensor(),
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ])

    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=batch_size, shuffle=True)
    return train_data_loader, valid_data_loader
```

2.3 模型选择与调参，训练

- 加载数据：

```
device = torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
train_data_loader, valid_data_loader = processing_data(data_path=data_path,
height=160, width=160, batch_size=32)
```

- 模型选择：一开始我使用了实验已经搭建好的 MCTNN 网络，使用了预训练模型 MobleNet,并且经过多次训练与调参之后，选择了 Adam 优化器，1e-3 的学习率，并且多次调整批量大小和训练轮数后，从 70 分到 90 分，得出了较为优秀的模型。

```
modify_x, modify_y = torch.ones((32, 3, 160, 160)), torch.ones((32))
epochs = 40
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3) # 优化器
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
'max', factor=0.2,patience=12)
```

但后续经过调参也没有提高，于是我改用了 Resnet50 模型，ResNet50 在 ImageNet 上的预训练模型广泛可用，可以通过迁移学习快速适应新的任务，减少训练时间和计算资源。修改后的模型如下。由于 resnet50 较为复杂，可能会我还加入了 dropout 层防止过拟合。并且使用了综合更优秀的 AdamW 优化器以及学习率，调整训练论数为 15 轮。

```
model = models.resnet50(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(num_ftrs, 2)
)
model = model.to(device)
optimizer = optim.AdamW(model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'max', factor=0.5, patience=1)
criterion = nn.CrossEntropyLoss()
```

- **模型训练：**

1. 训练过程: 在每个 epoch 中，将模型设为训练模式，迭代训练数据集，计算损失，反向传播并更新模型参数。
2. 验证过程: 在每个 epoch 结束时，将模型设为评估模式，迭代验证数据集，计算验证损失和准确率。
3. 保存最佳模型: 在每个 epoch 结束时，如果当前验证准确率超过历史最佳准确率，则保存当前模型权重。
4. 输出结果: 打印每个 epoch 的训练损失、验证损失和验证准确率，最后打印最佳准确率和训练完成信息。

三、代码内容

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(0.5),
        T.RandomVerticalFlip(0.5),
        T.RandomRotation(30),
        T.RandomResizedCrop(height, scale=(0.8, 1.0)),
        T.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.2),
        T.RandomAffine(degrees=30, translate=(0.2, 0.2), scale=(0.8, 1.2), shear=20),
        T.RandomPerspective(distortion_scale=0.5, p=0.5),
        T.ToTensor(),
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ])

    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=batch_size, shuffle=True)

    return train_data_loader, valid_data_loader

# 载入数据
data_path = r'D:\2212046wy\datasets\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.Sequential(
    nn.Dropout(0.5),
```

```
        nn.Linear(num_ftrs, 2)
)

model = model.to(device)
optimizer = optim.AdamW(model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'max', factor=0.5, patience=1)
criterion = nn.CrossEntropyLoss()

# 训练和验证
epochs = 20  # 增加训练轮数

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().item()
                total += b_y.size(0)

        valid_loss = valid_loss / len(valid_data_loader)
        accuracy = right_cnt / 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, r'D:\2212046wy\result\best_model.pth')

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

==========

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
```

四、实验结果

训练过程截图：

如图，训练了 15 轮之后，模型的最好准确率达到了 0.9917

```
Epoch: 10/15 || Train Loss: 0.0237 || Val Loss: 0.0278 || Val Acc: 0.9917
100%|██████████| 120/120 [05:33<00:00,  2.78s/it]
Epoch: 11/15 || Train Loss: 0.0208 || Val Loss: 0.0308 || Val Acc: 0.9834
100%|██████████| 120/120 [05:27<00:00,  2.73s/it]
Epoch: 12/15 || Train Loss: 0.0170 || Val Loss: 0.0560 || Val Acc: 0.9844
100%|██████████| 120/120 [04:01<00:00,  2.01s/it]
Epoch: 13/15 || Train Loss: 0.0206 || Val Loss: 0.0504 || Val Acc: 0.9865
100%|██████████| 120/120 [04:02<00:00,  2.02s/it]
  0%|          | 0/120 [00:00<?, ?it/s]Epoch: 14/15 || Train Loss: 0.0170 || Val Loss: 0.0448 || Val Acc: 0.9896
100%|██████████| 120/120 [03:54<00:00,  1.95s/it]
Epoch: 15/15 || Train Loss: 0.0108 || Val Loss: 0.0339 || Val Acc: 0.9917
Best Accuracy: 0.9917
Finish Training.
```

将模型导入平台进行测试：（注意修改文件路径）

如图，该模型训练后达到了 100 的准确率，说明训练出来的模型性能十分优秀（开心^ ^）

五、总结

1.更改模型后，发现用 resnet 训练的模型测试会报错，经过思考后发现需要修改 torch_py 中的 FaceRec.py 函数，有如下原因：

- 模型架构不一致

ResNet50 和 MobileNet 是两种不同的模型架构，它们的层数、层的类型、参数等都不相同。如果你在 ResNet50 上训练模型，然后试图用 MobileNet 的代码来加载和运行这个模型，模型架构的差异会导致加载错误。

- 模型输入大小不一致

不同的模型架构可能对输入图像的大小有不同的要求。例如，ResNet50 的预处理可能要求输入图像大小为 224x224，而 MobileNet 可能有不同的预处理要求。如果输入图像大小不符合预期，也会导致错误。

- 权重文件格式不匹配

即使模型架构相同，不同的训练过程和保存方法也可能导致权重文件格式不匹配。例如，如果你在保存模型时使用了一种特定的方法，但在加载模型时使用了另一种方法，可能会导致错误。

- 输出层不匹配

不同的模型可能有不同的输出层配置。例如，ResNet50 的输出层可能配置为适应特定数量的类，而 MobileNet 的输出层可能配置为不同数量的类。如果模型的输出层配置不一致，也会导致错误。

经过这些方面的修改后预测部分成功运行，模型得出了很好的准确率。

==========

2.模型参数的选择：

通过实验，我了解到不同的模型架构在处理复杂度和性能上的差异。ResNet50 在准确率和泛化能力方面表现优异，适合高精度要求的任务。通过引入 Dropout 层，我们进一步改善了模型的泛化能力，避免了过拟合现象。

在实验过程中，我们尝试了不同的超参数设置（如学习率、批量大小、优化器类型等）。学习率对模型的收敛速度和稳定性有显著影响，适当调整学习率可以显著提高模型的训练效率和性能。此外，不同的优化器（如 AdamW）和学习率调度器（如 CosineAnnealingLR）的使用，使得我们能够更有效地训练模型。

- 最终选择：

```
model = model.to(device)
optimizer = optim.AdamW(model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode: 'max', factor=0.5, patience=1)
criterion = nn.CrossEntropyLoss()
```

==========

3.本次实验验证了 ResNet50 在口罩佩戴检测任务中的优势，数据增强和超参数调优在提升模型性能方面具有重要作用。通过系统的实验和调优，实现了高效、准确的口罩检测系统，为未来的研究和实际应用提供了坚实基础。