程序报告

学号：2213924　　姓名：申宗尚

一、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。

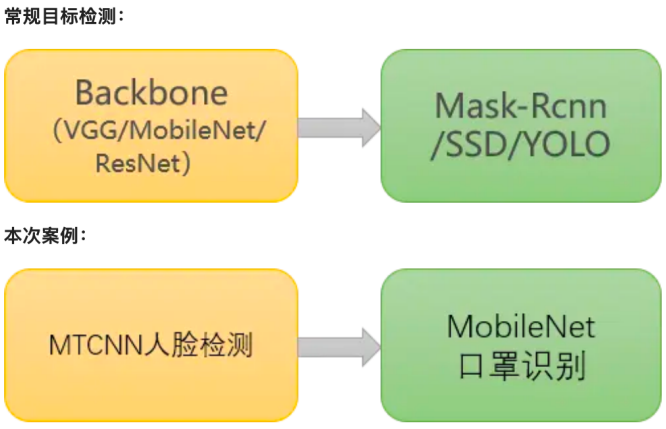
二、设计思想

在本次实验中，由于要建立一个目标检测的模型，识别人是否配戴了口罩，因此可以将本次任务分为两个部分：目标识别和位置检测。

通常情况下，特征提取需要由特有的特征提取神经网络来完成，如 VGG，MobileNet、ResNet 等，这些特征提取网络往往被称为 Backbone。而在 Backbone 后面连接全连接层（FC）就可以执行分类任务。但 FC 对目标的位置识别乏力。经过算法的发展，当前主要以特定的功能网络来代替 FC 的作用，如 Mask-Rcnn、SSD、YOLO 等。

我们选择充分使用已有的人脸检测模型 MTCNN，再训练一个识别口罩的模型，从而增加模型的准确率。

即，MTCNN 人脸检测->MobileNet 口罩识别。



由于本次实验的模型大体结构和代码已经给出，因此只需要进行参数的调整，就可以获得适合解决本问题本数据集的模型，根据对代码的分析，发现有以下参数可以进行调整：

1、Batch_size：批次大小。由于每次仅使用一个批次的训练样本来计算梯度、更新参数，所以可以调整批次大小来得到随机梯度和批量梯度折中的较优效果。

2、Loss：损失函数，通过对损失函数的梯度下降反向传播更新参数，由于本次问题为二分类问题，采用二分类问题适用的 Loss：交叉熵损失。

3、Optimizer：类型有 Adam，SGD，RMSprop 等等，在本实验中使用 Adam

4、Epoch：模型训练轮数，由于模型的训练轮数过多可能会产生过拟合效果，但轮次过少又会欠拟合，因此要调整参数使得模型最适应解决当前问题。
5、Learning Rate：学习率。是模型通过梯度下降一次下降的多少。大了难以收敛并完成训练，小了容易使训练整体流程过慢，因此需要折中考虑。在本次实验中，先给一个大的 Learning Rate，然后不断减小，知道发现最好的学习率。

三、代码介绍

Letterbox_image 函数：

```python
def letterbox_image(image, size):
    """
    调整图片尺寸
    :param image: 用于训练的图片
    :param size: 需要调整到网络输入的图片尺寸
    :return: 返回经过调整的图片
    """
    new_image = cv2.resize(image, size, interpolation=cv2.INTER_AREA)
    return new_image
```

参数：

image：用于训练的图像

size：调整后的图像尺寸(宽、高)

返回值：

New_image 调整后的图像

首先，调整输入图像的尺寸，以适应网格输入的尺寸要求，这里使用 OpenCV 的 rusize 函数对图像进行插值调整。

Processing_date 函数：

```python
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
```

参数：

Data_path：数据集路径

Height、width：调整后的图像高度、宽度

Batch_size：每次读取的图像数量

Test_split：测试集划分比例

返回值：

Train_data_loader:训练数据加载器

Valid_data_loader:验证数据加载器

在这个函数中，使用 torchvision 的 T.Compose 函数进行数据增强和预处理（调整尺寸、随机翻转、归一化等）。然后使用 ImageFolder 加载数据集，按照指定比例分割数据集为训练集和测试集，最后创建 DataLoader 对象以便后续批量读取数据。

Show_tensor_img 函数：

```
def show_tensor_img(img_tensor):
    img = img_tensor[0].data.numpy()
    img = np.swapaxes(img, 0, 2)
    img = np.swapaxes(img, 0, 1)
    img = np.array(img)
    plot_image(img)
```

参数：

Img_tensor：要显示的图像张量。

这个函数通过将张量转换为 numpy 数组，并进行轴交换以适应图像格式，然后使用 plot_image 函数显示图像。

训练代码段：

```
# 加载 MobileNet 的预训练模型权
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=3
modify_x, modify_y = torch.ones((32, 3, 160, 160)), torch.ones((32))

epochs = 20
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-2)  # 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_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)

        # print(pred_y.shape)
        # print(y.shape)

        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

        loss_list.append(loss)

    print('step:' + str(epoch + 1) + '/' + str(epochs) + ' || Total Loss: %.4f' % (loss))
torch.save(model.state_dict(), './results/temp.pth')
print('Finish Training.')
```

最后，使用 MobileNetV1 模型进行训练。

加载 MobileNetV1 模型后，并将其移动到合适的设备（CPU 或 GPU）。定义优化器、学习率调度器和损失函数。训练循环中，模型进入训练模式，读取训练数据，计算预测结果和损失，进行梯度回传和优化，最后保存最佳模型的权重和损失值。

超参数部分：

在本次实验中，主要进行了超参数的调试，下面给出代码中超参数部分的修改。

```
width=160, batch_size=32)
```

1、batch_size ：批次大小经过多次学习和测量，定为 32 最合适。

```
80    epochs = 20
81    model = MobileNetV1(class
```

2、epochs：最终定为 20 轮（初始为 2 轮）

```
optimizer = optim.Adam(model.parameters(), lr=1e-2)  # 优化器
```

3、optimizer，learning rate：优化器使用 Adam，learning rate 由大到小进行了测试，最终确定为 1e-2 最为合适。

```
criterion = nn.CrossEntropyLoss()
```

4、损失函数：如前所说，对于二分类问题，使用交叉熵损失。

四、代码内容

下面给出本次的训练代码：

```
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
# 数据集路径
data_path = "./datasets/5f680a696ec9b83bb0037081-momodel/data/"
def letterbox_image(image, size):
```

```
    """
    调整图片尺寸
    :param image: 用于训练的图片
    :param size: 需要调整到网络输入的图片尺寸
    :return: 返回经过调整的图片
    """
    new_image = cv2.resize(image, size, interpolation=cv2.INTER_AREA)
    return new_image
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=32)
```

```
def show_tensor_img(img_tensor):
    img = img_tensor[0].data.numpy()
    img = np.swapaxes(img, 0, 2)
    img = np.swapaxes(img, 0, 1)
    img = np.array(img)
    plot_image(img)

pnet_path = "./torch_py/MTCNN/weights/pnet.npy"
rnet_path = "./torch_py/MTCNN/weights/rnet.npy"
onet_path = "./torch_py/MTCNN/weights/onet.npy"

# 加载 MobileNet 的预训练模型权
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)
modify_x, modify_y = torch.ones((32, 3, 160, 160)), torch.ones((32))

epochs = 20
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-2)  # 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_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)

        # print(pred_y.shape)
        # print(y.shape)
```

```
            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

            loss_list.append(loss)

        print('step:' + str(epoch + 1) + '/' + str(epochs) + ' || Total
Loss: %.4f' % (loss))
torch.save(model.state_dict(), './results/temp.pth')
print('Finish Training.')
```

五、实验结果

准确率 100%，通过测试。

六、总结

在本次实验中，成功地构建了一个基于深度学习的目标检测模型，用于识别图像中的人是否佩戴口罩。实验的主要流程包括问题重述、设计思想、代码实现、参数调试以及结果验证。

本次实验区别于前面四次实验最大的不同就是不用自己编写代码，但是非常考验对于超参数的理解和调试。通过不断的网上学习和多次尝试，（也可能是运气），最终训练到了一个非常不错的模型。

其实本次实验的问题并不算一个简单问题，一开始看到的时候非常恐惧，感觉难度不小，但是好在这次实验老师都已经提供了一个写好的 MCTNN，可以直接针对图片划分所有识别的人脸，减少了非常非常多的可能遇见的问题。通过本次实验的学习，我也对深度学习、超参数调试有了更深的理解。