程序报告

学号：2313721　　姓名：许洋

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Pytorch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』）。

1.5 参考资料

论文

Joint Face Detection and Alignment using Multi – task Cascaded Convolutional Networks：https://kpzhang93.github.io/MTCNN_face_detection_alignment/

OpenCV：https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

PIL：https://pillow.readthedocs.io/en/stable/

Numpy：https://www.numpy.org/

Scikit – learn： https://scikit-learn.org/

PyTorch：https://pytorch.org/

二、设计思想

- **1. 数据处理部分**

(1) 数据可视化

```
mask_num = 4
fig = plt.figure(figsize=(15, 15))
for i in range(mask_num):
    sub_img = cv2.imread(data_path + "/image/mask/mask_"
+ str(i + 101) + ".jpg")
    sub_img = cv2.cvtColor(sub_img, cv2.COLOR_RGB2BGR)
    ax = fig.add_subplot(4, 4, (i + 1))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title("mask_" + str(i + 1))
    ax.imshow(sub_img)
```

- 读取带有口罩的图片（`mask_101.jpg` 到 `mask_104.jpg`），并显示在 `4x4` 的子图中。
- `cv2.imread` 用于读取图片，`cv2.cvtColor` 将图片从 BGR 格式转换为 RGB 格式（因为 OpenCV 默认读取为 BGR 格式）。
- 使用 `matplotlib` 的 `subplot` 方法将图片显示在子图中。

(2) 图片尺寸调整

```
def letterbox_image(image, size):
    new_image = cv2.resize(image, size,
interpolation=cv2.INTER_AREA)
    return new_image
```

- 定义了一个函数 `letterbox_image`，用于调整图片尺寸。
- 使用 `cv2.resize` 方法将图片调整为目标尺寸 `size`。

(3) 数据加载与预处理

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

- 定义了一个函数 `processing_data`，用于加载和预处理数据集。
- 使用 `torchvision.transforms` 对图片进行预处理，包括调整尺寸、随机水平翻转、随机垂直翻转、转换为张量并归一化。

- 使用 `ImageFolder` 加载数据集，并通过 `random_split` 划分为训练集和测试集。
- 创建 `DataLoader`，用于批量加载数据。

2. 模型训练部分

(1) 模型初始化

```
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)
scheduler =
optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'max',
factor=0.55, patience=12, min_lr=1e-6, threshold=1e-3,
cooldown=1)
criterion = nn.CrossEntropyLoss()
```

- 使用 `MobileNetV1` 作为模型，输出类别为 2（戴口罩和不戴口罩）。
- 使用 Adam 优化器，学习率为 `1e-3`。
- 使用学习率调度器 `ReduceLROnPlateau`，当验证集准确率在连续 12 个 epoch 内没有提升时，学习率减半。添加最小学习率等参数。
- 使用交叉熵损失函数。

(2) 训练过程

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

- 进行 30 个 epoch 的训练。
- 每个 epoch 中，将数据批量加载到模型中，计算损失并进行反向传播。
- 如果当前损失小于最佳损失，则保存当前模型的权重。
- 训练完成后，将模型权重保存到 `results/temp.pth` 文件中。

3. 模型评估与预测部分

(1) 加载模型

```
model_path = './results/temp.pth'
recognize = Recognition(model_path)
```

- 加载训练好的模型权重。

(2) 预测函数

```
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 all_num, mask_num
```

- 将输入的图片（`cv2.imread` 读取的格式）转换为 PIL 图像格式。
- 调用 `Recognition` 类的 `mask_recognize` 方法，对图片进行口罩检测。
- 返回图片中总人数和戴口罩的人数。

(3) 测试图片

```
img = cv2.imread("test1.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
all_num, mask_num = predict(img)
print(all_num, mask_num)
```

- 读取测试图片并进行预测。
- 输出图片中总人数和戴口罩的人数。

4. 其他功能

(1) 人脸检测

```
detector = FaceDetector()
recognize = Recognition(model_path='results/temp.pth')
draw, all_num, mask_nums = recognize.mask_recognize(img)
plt.imshow(draw)
plt.show()
```

- 使用 `FaceDetector` 检测人脸。
- 在检测到的人脸上绘制方框，并显示结果。

三、代码内容

```
import warnings
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
data_path = "./datasets/5f680a696ec9b83bb0037081-momodel/data/"
mask_num = 4
fig = plt.figure(figsize=(15, 15))
for i in range(mask_num):
    sub_img = cv2.imread(data_path + "/image/mask/mask_" + str(i +
101) + ".jpg")
    sub_img = cv2.cvtColor(sub_img, cv2.COLOR_RGB2BGR)
    ax = fig.add_subplot(4, 4, (i + 1))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title("mask_" + str(i + 1))
    ax.imshow(sub_img)
nomask_num = 4
fig1 = plt.figure(figsize=(15, 15))
for i in range(nomask_num):
    sub_img = cv2.imread(data_path + "/image/nomask/nomask_" +
str(i + 130) + ".jpg")
    sub_img = cv2.cvtColor(sub_img, cv2.COLOR_RGB2BGR)
    ax = fig1.add_subplot(4, 4, (i + 1))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title("nomask_" + str(i + 1))
    ax.imshow(sub_img)
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
read_img = Image.open("test1.jpg")
read_img = np.array(read_img)
print("调整前图片的尺寸:", read_img.shape)
read_img = letterbox_image(image=read_img, size=(50, 50))
read_img = np.array(read_img)
print("调整前图片的尺寸:", read_img.shape)
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)
    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
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)
for index, (x, labels) in enumerate(train_data_loader):
    print(index, "\nfeature:",x[0], "\nlabels:",labels)
    show_tensor_img(x)
    break
pnet_path = "./torch_py/MTCNN/weights/pnet.npy"
rnet_path = "./torch_py/MTCNN/weights/rnet.npy"
onet_path = "./torch_py/MTCNN/weights/onet.npy"
torch.set_num_threads(1)
img = Image.open("test.jpg")
recognize = Recognition()
draw = recognize.face_recognize(img)
plot_image(draw)
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 = 30
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)  # 优化器
print('加载完成...')
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                 'max',
                                                 factor=0.55,
 # 学习率减半
                                                 patience=12,
 # 连续 12 次准确率不提升则降低学习率
                                                 min_lr=1e-6,
# 最小学习率
                                                 threshold=1e-3,
 # 准确率提升小于 0.001 不算改善
```

```
                                                cooldown=1 )     #
降低学习率后等待 1 个 epoch)
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)
        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.')
plt.plot(loss_list,label = "loss")
plt.legend()
plt.show()
img = Image.open("test.jpg")
detector = FaceDetector()
recognize = Recognition(model_path='results/temp.pth')
draw, all_num, mask_nums = recognize.mask_recognize(img)
plt.imshow(draw)
plt.show()
print("all_num:", all_num, "mask_num", mask_nums)
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
```

```
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
    """
    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 all_num,mask_num
img = cv2.imread("test1.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
all_num, mask_num = predict(img)
print(all_num, mask_num)
```

四、实验结果

- 平台检测结果：

测试点	状态	时长	结果
在 5 张图片上测试模型	✓	5s	得分:100.0

五、总结

- 通过本次实验，我成功实现了基于深度学习的口罩检测模型，能够准确识别图中人物是否佩戴口罩。在实验过程中，模型训练效果良好，损失值逐渐下降，并最终保存了最佳权重。同时，结合人脸检测与口罩识别，展示了清晰的检测结果。然而，实验中也遇到了一些挑战，模型训练时间较长，尤其是在资源受限的情况下，同时，在调整参数的过程中进行复杂的探索，通过查阅各种资料，添加了其他的限制参数。针对这些问题，未来可以增加数据增强方法以提升模型泛化能力，尝试更先进的轻量级模型以优化性能，并探索多任务学习框架以进一步提升检测效果。

- 通过本次实验，我掌握了数据处理、模型训练、评估和预测的完整流程，熟悉了 `PyTorch` 框架的使用方法，并学习了 `MTCNN` 和 `MobileNet` 的结构与应用。这些收获为后续的深度学习研究和项目开发奠定了坚实的基础。展望未来，我计划将该模型应用于视频流检测，实现实时口罩检测功能，并进一步优化模型性能，提高检测速度和准确率，以满足实际应用的需求。