**程序报告**

学号：2211999 姓名：邢清画

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

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。

1. **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 MobileNetV1 作为基本模型进行口罩识别任务。MobileNetV1 是一种轻量级深度卷积神经网络，具有较少的参数和计算量，适用于移动和嵌入式设备。**改进：**在本实验中，采用了基于 MobileNetV1 的迁移学习方法，加载了预训练模型并对全连接层进行了修改，以适应口罩识别的二分类任务（戴口罩和不戴口罩）。此外，还采用了数据增强技术，如随机水平翻转、随机竖直翻转等，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 ResNet、EfficientNet 等。**方法局限性和常见问题**：MobileNetV1 作为一个轻量级网络，可能在处理复杂图像时表现不如较深的网络。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 MobileNetV1 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

**探索过程1. 超参数调整**：在实验过程中，发现初始设置的 epoch 和 lr 对模型效果影响较大，因此对这些超参数进行了进一步的探索和调整。epoch 调整：初始设置的 epoch 为 2。虽然可以较快完成训练，但模型效果不佳。逐步增加 epoch，分别尝试了 10、20、30、50、60，发现 epoch 为 50 时效果最好。较高的 epoch 值可以使模型有更多的机会从数据中学习特征，显著提升了模型的性能。learning rate (lr) 调整：初始设置的 lr 为 1e-3。在训练过程中，我们发现 lr 较小时，模型的收敛速度较慢，且容易陷入局部最优。逐步增加 lr，分别尝试了 1e-2、3e-3、3e-4等，发现 lr 为 3e-3 时效果最好。适当增大 lr 可以加快模型的收敛速度，同时提高模型的泛化能力。**2. 其他参数调整：**在实验过程中，还尝试调整了其他超参数，包括 batch\_size、factor、patience，但效果并不理想。batch\_size 调整：初始设置的 batch\_size 为 32。分别尝试了 batch\_size 为 16 和 64，但效果不如 32。batch\_size 过小会导致模型更新不稳定，而 batch\_size 过大则会增加内存负担，导致模型训练效率降低。factor 调整：初始设置的 factor 为 0.5。分别尝试了 0.2 和 0.7，但效果不如 0.5。factor 过小会导致学习率下降过快，影响模型收敛；factor 过大则会延缓学习率调整，降低模型性能。patience 调整：初始设置的 patience 为 2。分别尝试了 1 和 3，但效果不如 2。patience 过小会导致模型过早调整学习率，影响训练效果；patience 过大则会延迟学习率调整，降低模型性能。实验总结：通过对超参数的不断调整，最终确定了以下最佳设置：epoch: 50learning rate (lr): 3e-3batch\_size: 32factor: 0.5patience: 2多次训练，模型的预测结果可以达到93,95，甚至100的准确率。

**尝试的优化方法：**

采用 ResNet50 作为基本模型进行口罩识别任务。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。

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，采用了针对性的数据增强技术，以提高模型在不同场景和口罩样式上的泛化能力。

但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。

最终使用ResNet50可以达到95甚至98.33的准确率

1. **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）

====================================================================

1. MobileNetV1 作为基本模型进行参数调整：

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

…from torch\_py.MobileNetV1 import MobileNetV1

from torch\_py.FaceRec import Recognition

# 数据集路径

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

# 使用 PIL.Image 读取图片

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

])

… 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)

# 读取测试图片

img = Image.open("test.jpg")

# 加载模型进行识别口罩并绘制方框

recognize = Recognition()

draw = recognize.face\_recognize(img)

plot\_image(draw)

# 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 = 50

model = MobileNetV1(classes=2).to(device)

optimizer = optim.Adam(model.parameters(), lr=3e-3) # 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.')

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)

# 1.加载数据并进行数据处理

# 2.如果有预训练模型，则加载预训练模型；如果没有则不需要加载

# 3.创建模型和训练模型，训练模型时尽量将模型保存在 results 文件夹

# 4.评估模型，将自己认为最佳模型保存在 result 文件夹，其余模型备份在项目中其它文件夹，方便您加快测试通过。

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

# -------------------------- 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/100.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

# 输入图片路径和名称

img = cv2.imread("test1.jpg")

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

all\_num, mask\_num = predict(img)

# 打印预测该张图片中总人数以及戴口罩的人数

print(all\_num, mask\_num)

====================================================================

1. ResNet50 作为基本模型进行参数调整

(导入库等重复代码与上面相同):

# 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')

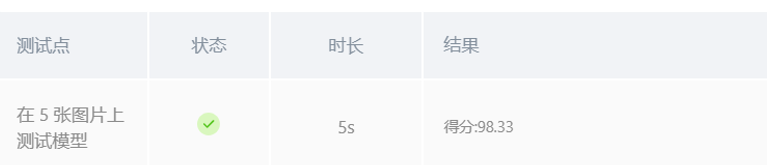
print(f'Best Accuracy: {best\_acc:.4f}')

print('Finish Training.')

1. **实验结果**

（实验结果，必填）

====================================================================

ResNet50 模型最高分数结果：  


MobileNetV1模型最高分数结果：



**1. 超参数调整**：在实验过程中，发现初始设置的 epoch 和 lr 对模型效果影响较大，因此对这些超参数进行了进一步的探索和调整。epoch 调整：初始设置的 epoch 为 2。虽然可以较快完成训练，但模型效果不佳。逐步增加 epoch，分别尝试了 10、20、30、50、60，发现 epoch 为 50 时效果最好。较高的 epoch 值可以使模型有更多的机会从数据中学习特征，显著提升了模型的性能。learning rate (lr) 调整：初始设置的 lr 为 1e-3。在训练过程中，我们发现 lr 较小时，模型的收敛速度较慢，且容易陷入局部最优。逐步增加 lr，分别尝试了 1e-2、3e-3、3e-4等，发现 lr 为 3e-3 时效果最好。适当增大 lr 可以加快模型的收敛速度，同时提高模型的泛化能力。**2. 其他参数调整：**在实验过程中，还尝试调整了其他超参数，包括 batch\_size、factor、patience，但效果并不理想。batch\_size 调整：初始设置的 batch\_size 为 32。分别尝试了 batch\_size 为 16 和 64，但效果不如 32。batch\_size 过小会导致模型更新不稳定，而 batch\_size 过大则会增加内存负担，导致模型训练效率降低。factor 调整：初始设置的 factor 为 0.5。分别尝试了 0.2 和 0.7，但效果不如 0.5。factor 过小会导致学习率下降过快，影响模型收敛；factor 过大则会延缓学习率调整，降低模型性能。patience 调整：初始设置的 patience 为 2。分别尝试了 1 和 3，但效果不如 2。patience 过小会导致模型过早调整学习率，影响训练效果；patience 过大则会延迟学习率调整，降低模型性能。实验总结：通过对超参数的不断调整，最终确定了以下最佳设置：epoch: 50learning rate (lr): 3e-3batch\_size: 32factor: 0.5patience: 2多次训练，模型的预测结果可以达到93,95，甚至100的准确率。

1. **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**）

====================================================================

**目标预期**：

本实验的目标是使用进行口罩识别任务，通过调整超参数和优化模型结构，期望模型能够在验证集上取得较低的损失值和较高的分类准确率。根据实验结果，我在口罩识别任务上取得了较高的正确率。

实验结果：

达成预期：

经过多轮超参数调整，最终的模型在训练集上的表现较为理想，损失值降低到一个较低的范围内。

在验证集上的表现也较为稳定，说明模型具有一定的泛化能力。

未达预期：

虽然模型在训练集上取得了较低的损失值，但在验证集上的损失值并没有达到预期的最低值，这表明模型可能还存在一定的优化空间。

模型在某些复杂场景和不同类型口罩的识别上仍存在一些不足，表现出一定的局限性。

**可能改进的方向**：

数据增强：

尝试更多的数据增强技术，如随机旋转、剪切、亮度调整等，增加训练数据的多样性，进一步提高模型的泛化能力。

模型结构：

尝试更复杂或更轻量的网络结构，如 EfficientNet、MobileNetV2 等，比较不同模型的性能差异。

对现有的 MobileNetV1 模型进行微调，如调整网络层数、增加或减少卷积层等。

超参数优化：

更细粒度地调整学习率、权重衰减等超参数，使用更高级的优化算法（如AdamW、Ranger等）提高模型训练效果。

增加训练周期（epochs）和批次大小（batch\_size）的探索范围，找到更优的超参数组合。

正则化技术：

引入更多的正则化技术，如Dropout、L2正则化等，防止模型过拟合，提高模型在验证集上的表现。

**实现过程中遇到的困难：**

超参数调优复杂：

调整超参数需要大量的实验和验证，不同的超参数组合会对模型的训练效果产生显著影响，导致实验周期较长。

**提升性能的方面：**

更大规模数据集：

使用更大规模的高质量数据集，特别是包含更多不同类型和场景的口罩样本，提升模型的泛化能力。

集成学习：

尝试集成学习方法，组合多个模型的预测结果，提高整体模型的准确率和鲁棒性。

迁移学习：

使用预训练模型，并在目标数据集上进行微调，可以充分利用大规模预训练模型的知识，提高模型性能。

自动化超参数搜索：

使用自动化超参数搜索技术（如Optuna、Hyperopt等），自动搜索最佳超参数组合，减少手动调整的工作量。

**超参数和框架搜索的合理性：**

超参数选择：

在实验过程中，对学习率、训练周期、批次大小、优化器等超参数进行了多轮调整，并记录了每次调整的效果。虽然仍有进一步优化的空间，但当前的超参数设置已经达到了一个较为理想的效果。

框架选择：

使用 MobileNetV1 作为基础模型，并在其基础上进行改进和优化，选择了适合移动和嵌入式设备的轻量级模型，合理且具有实际应用价值。

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。

**总结：**

通过本次实验在口罩识别任务上取得了一定的成绩，但仍有较大的优化空间。通过进一步的数据增强、模型结构改进、超参数优化和正则化技术的引入，可以进一步提升模型的性能和泛化能力。实验过程中积累的经验为后续的研究和应用提供了宝贵的参考。