**程序报告**

学号：2213041 姓名：李雅帆

1. **问题重述**

1.实验背景

有效防御新冠病毒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来检测图像中的人的口罩佩戴情况。

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

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。

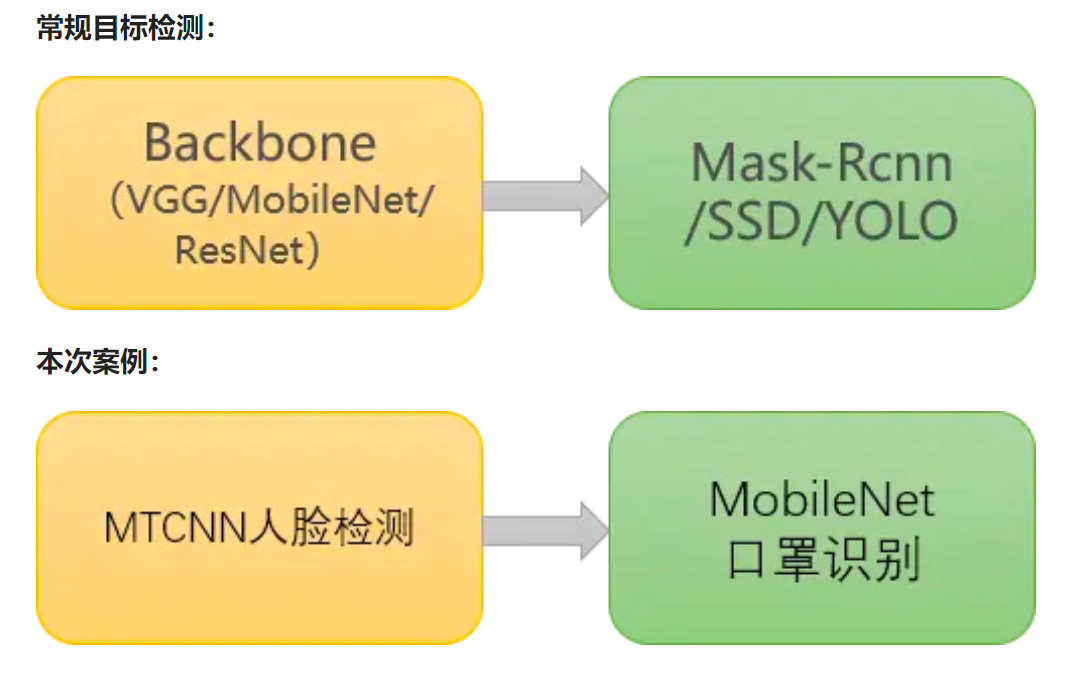
1. **设计思想**

**1.针对目标检测的任务，可以分为两个部分：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的模型，再训练一个识别口罩的模型，从而提高训练的开支、增强模型的准确率。



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

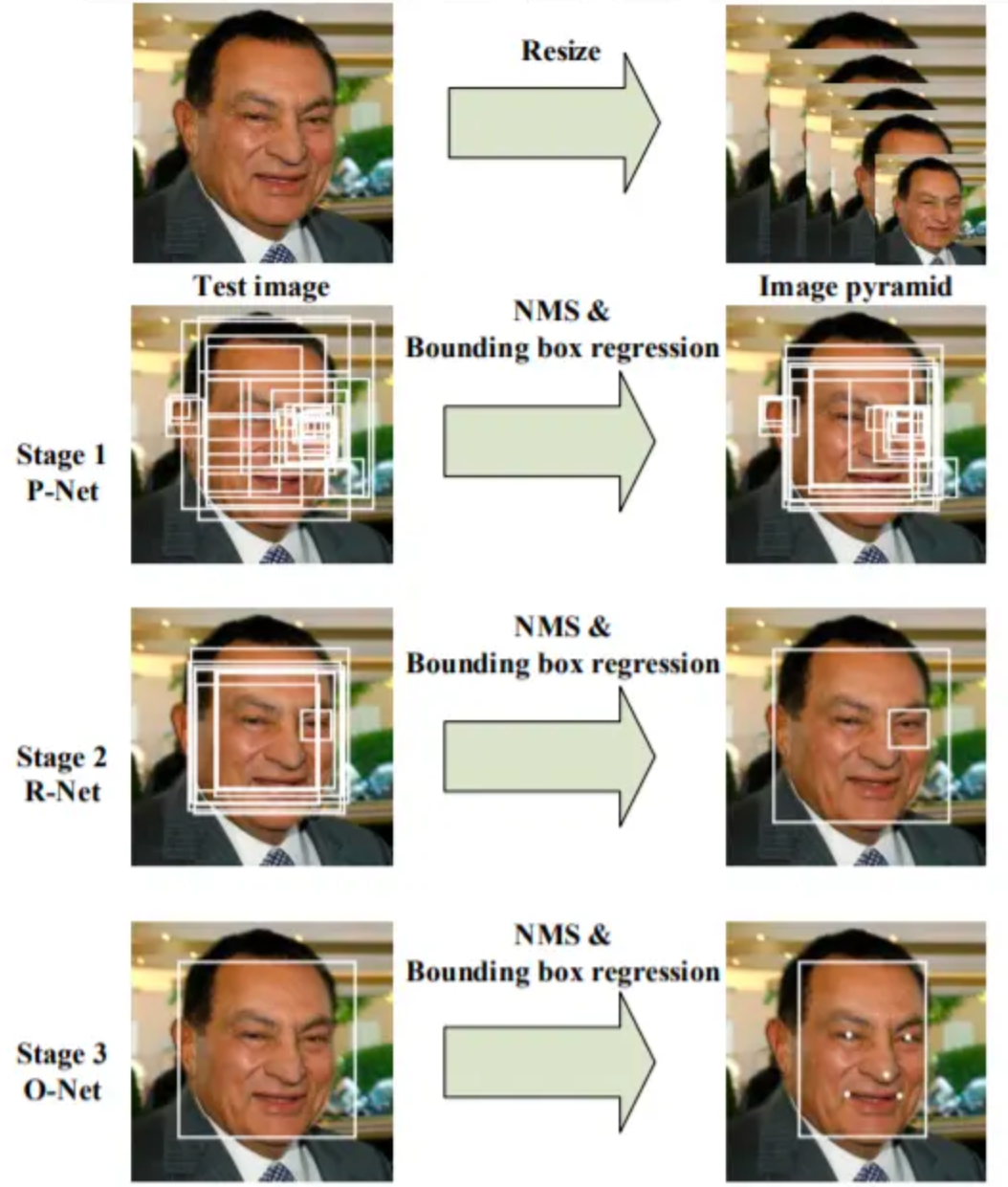
**2.代码设计思想流程。**

（1）数据准备和预处理： 首先加载口罩和非口罩的图像数据，并进行预处理。预处理包括调整图像大小、随机水平和垂直翻转、转换为张量、归一化等操作。这些步骤可以为模型提供标准化的输入数据，并增加数据的多样性，提高模型的泛化能力。

（2）人脸检测： 使用MTCNN（Multi-task Cascaded Convolutional Networks）进行人脸检测，识别出图像中的人脸，并绘制方框。

1. 三阶段的级联（cascaded）架构
2. coarse-to-fine 的方式
3. new online hard sample mining 策略
4. 同时进行人脸检测和人脸对齐

⑤ state-of-the-art 性能



（3）口罩识别：使用MobileNetV1模型进行口罩识别的训练和评估，在本步骤可以调整学习率等参数。

（4）模型训练和评估： 使用训练数据集进行模型的训练，采用交叉熵损失函数和Adam优化器进行模型参数的优化。训练过程中监控损失函数的变化，并保存在验证集上表现最好的模型。在训练完成后，对模型在测试集上的性能进行评估，包括准确率、精确率、召回率等指标。

（5）模型部署和预测： 将训练好的模型保存到文件中，在需要进行口罩识别的时候加载模型进行预测。预测过程中，输入待预测的图像，经过模型处理后输出预测结果，包括图像中总人数和戴口罩的人数。

（6）优化和改进： 在整个流程中，可以对数据处理、模型结构、超参数等进行优化和改进，以提高口罩识别系统的性能和稳定性。这些优化和改进可以通过实验和评估来验证其效果，不断地优化口罩识别系统的整体性能。

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

**3.在本次实验中，我们主要对以下参数进行了修改。**

（1）调整 epoch 时期数：

在深度学习中，一个 epoch （时期）指的是将整个训练数据集（dataset）在神经网络中正向传播和反向传播一次的过程。在训练神经网络时，通常会将训练数据集划分为若干个 batch（批次），每次从数据集中抽取一个 batch 的数据用于训练，一个 epoch 则是指将整个数据集中的所有样本都用于训练一次。

调整 epoch 数是一种优化神经网络模型的常用方法，有以下作用：

①提高模型性能： 增加 epoch 数可以让模型更多次地观察和学习数据集中的样本，从而更充分地学习数据的特征和规律，提高模型的性能。

②优化参数： 通过增加 epoch 数，模型可以更多次地进行参数更新，从而优化模型的参数，使其更好地拟合训练数据，提高模型的泛化能力。

③克服欠拟合： 如果模型在训练集上的表现较差，可能由模型没有充分学习数据的特征造成。增加 epoch 数可以让模型有更多机会学习数据的特征，从而克服欠拟合问题。

④监控模型性能： 通过观察模型在训练集和验证集上的表现随 epoch 数的变化，可以更好地监控模型的训练过程，及时发现模型是否出现过拟合或欠拟合的情况。

模型初始设置的epoch为2，这个epoch值有一点小，模型可能尚未拟合收敛。因此，我们在调整参数的过程中，逐渐增大epoch，发现其值为40的时候，模型达到较好的效果，因此我们最将epoch的值设为40。



（2）调整学习率：

factor 和 patience 是用于调整学习率的两个重要参数。

Factor： 在学习率调度器中，factor 是一个衰减因子，用于控制学习率的下降速度。当模型在验证集上的性能停止提升时，学习率调度器会根据设定的规则减小学习率。factor 就是在每次调整学习率时将当前学习率乘以的系数。例如，如果 factor 设置为 0.1，则每次调整学习率时，学习率都会减小到原来的 0.1 倍。

Patience： patience 是一个用于指定触发学习率调整的条件，通常是在验证集上监控模型性能时使用。它表示在验证集上连续 patience 次 epoch 内模型性能没有改善时，触发学习率的调整。也就是说如果在 patience 次 epoch 内模型的性能没有提升，则认为模型已经停止改善，可以通过调整学习率来尝试找到更优的模型参数。

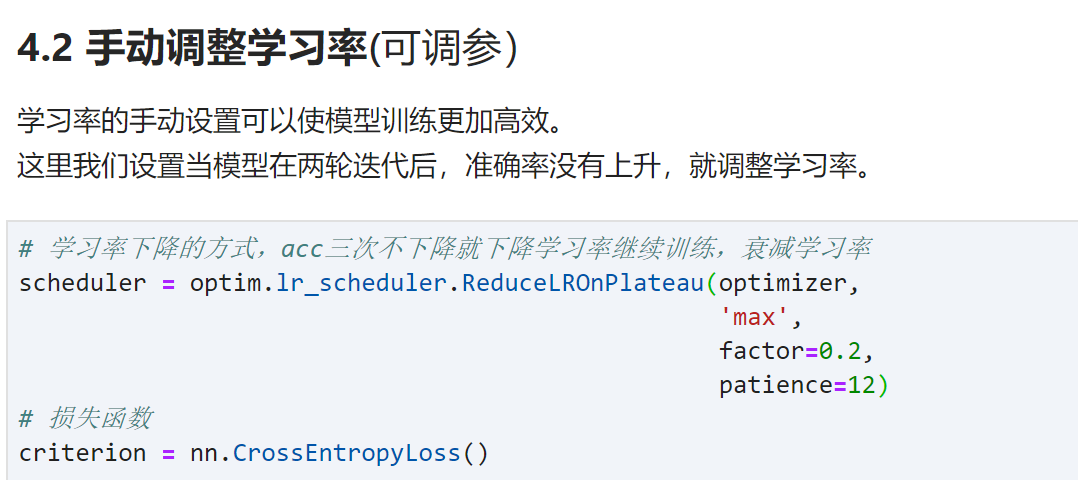
通过调整 factor 和 patience 可以进行参数优化的原因是：

①控制学习率下降速度： 通过调整 factor 可以控制学习率的下降速度。如果 factor 设置得太大，可能会导致学习率下降得太快，影响模型的性能；如果 factor 设置得太小，可能会导致学习率下降得太慢，影响模型的收敛速度。调整 factor 可以帮助找到一个合适的学习率下降速度，从而提高模型的性能。

我们调整这个factor为0.2，是一个比较缓慢的学习率下降速度。当学习率过大会导致网络不能收敛，徘徊在最优值附近

②提高模型鲁棒性： 通过调整 patience 可以提高模型的鲁棒性。较大的 patience 值可以使模型更加稳定地训练，避免过早地调整学习率。而较小的 patience 值可以使模型更加灵活地调整学习率，及时发现模型的性能变化。

我们调整 patience = 12 ，表示我们将忽略前 12个没有改善的 epoch，并且仅在第 12 个 epoch 之后损失仍然没有改善的情况下降低 LR。



1. **代码内容**

**1.预处理及数据增强**

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/"

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)

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

**2.创建数据集**

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)

for index, (x, labels) in enumerate(train\_data\_loader):

print(index, "\nfeature:",x[0], "\nlabels:",labels)

show\_tensor\_img(x)

break

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

**3.利用MTCNN进行人脸识别**

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)

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

**4.利用MobileNet进行口罩识别**

# 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 = 40

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

optimizer = optim.Adam(model.parameters(), lr=1e-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=0.2,

patience=12)

# 损失函数

criterion = nn.CrossEntropyLoss()

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

**5.训练得到最终模型**

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)

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

**6.提交结果**

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/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

# 输入图片路径和名称

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

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

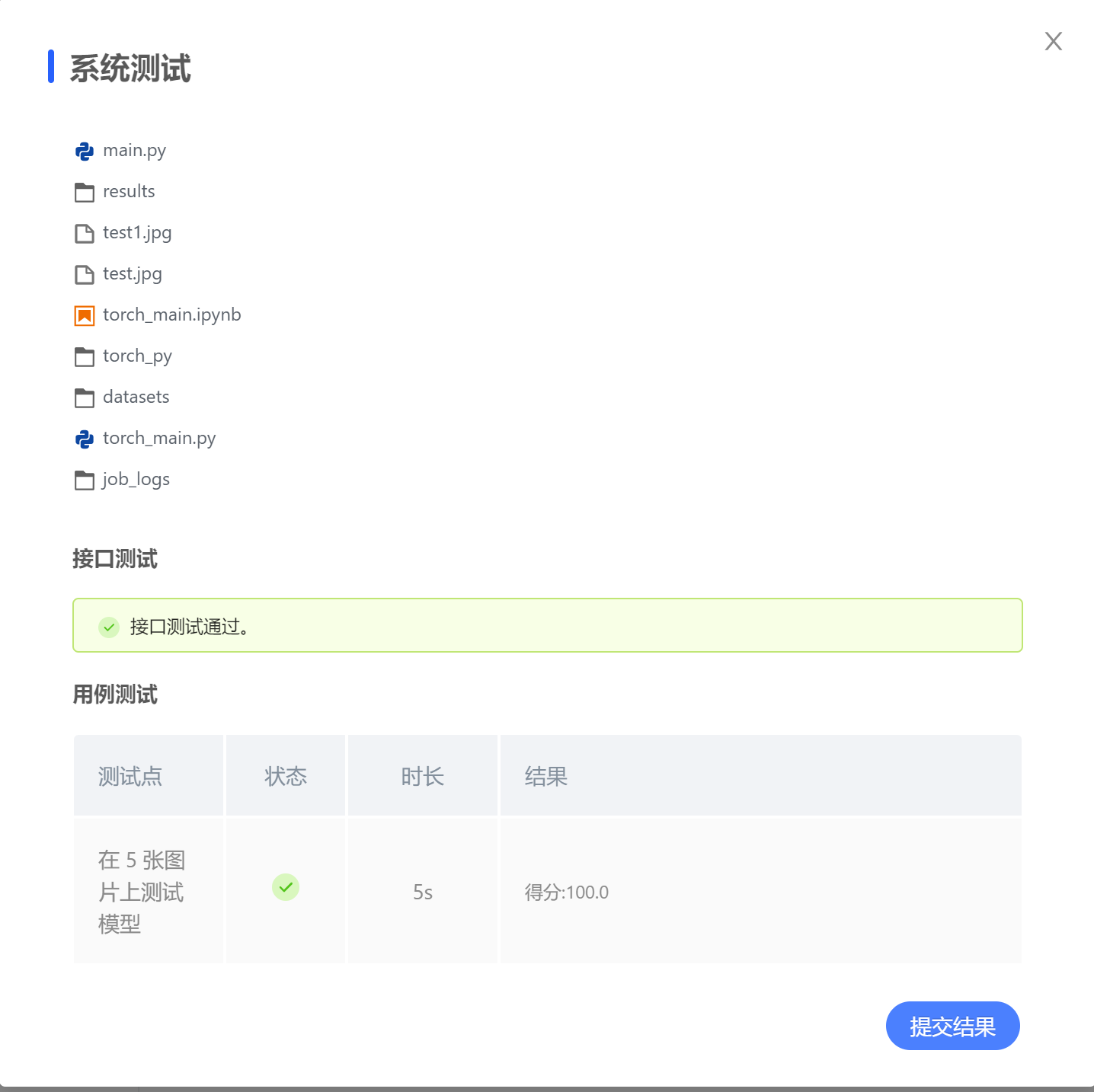
all\_num, mask\_num = predict(img)

# 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. **实验结果**

对参数进行多次调整，并对模型进行多次训练，最终调整到了合适的参数并训练出得分100.0的模型。



1. **总结**

在本次实验中我已经建立深度学习模型，检测出图中的人是否佩戴了口罩，并通过调整参数将其尽可能调整到最佳状态，实现了预期目标。

本实验中最大的难点就是通过对参数进行调整来优化口罩识别系统的整体性能。

我采用的基于torch的版本，还可以尝试基于keras的版本和mindspore的版本，并进行相关优化。