**口罩佩戴检测问题程序报告**

学号2112213姓名：冯思程

1. **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：

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. **设计思想**

**方法：**

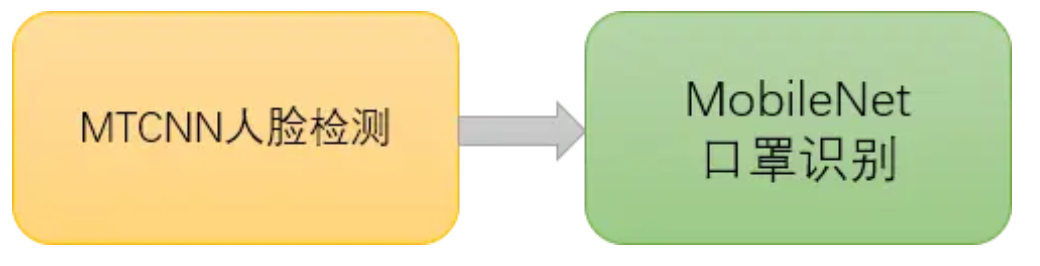
针对目标检测的任务，可以分为两个部分：目标识别和位置检测。

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。

这里我们具体利用的两个模型分别是：MTCNN人脸检测、MoblieNet口罩识别。



MTCNN人脸检测模型：

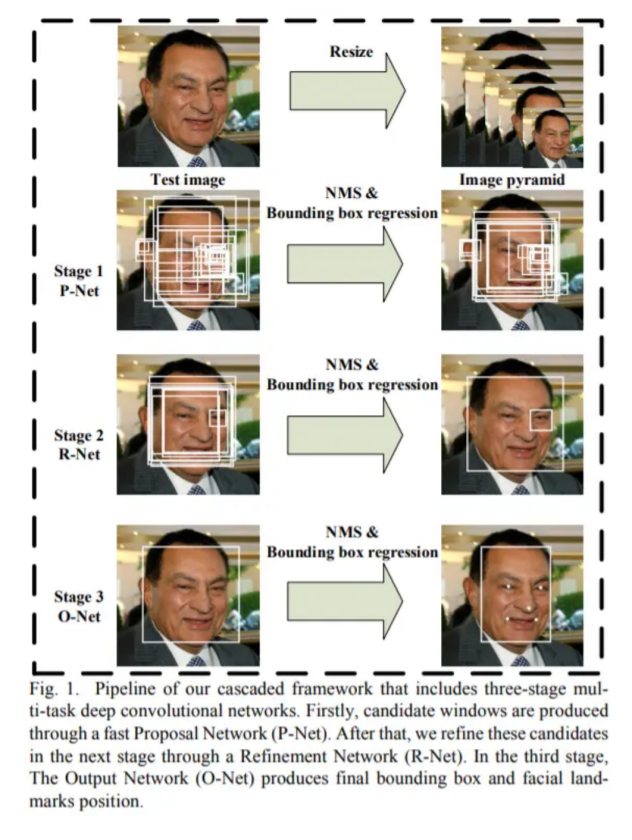
1)三阶段的级联（cascaded）架构

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

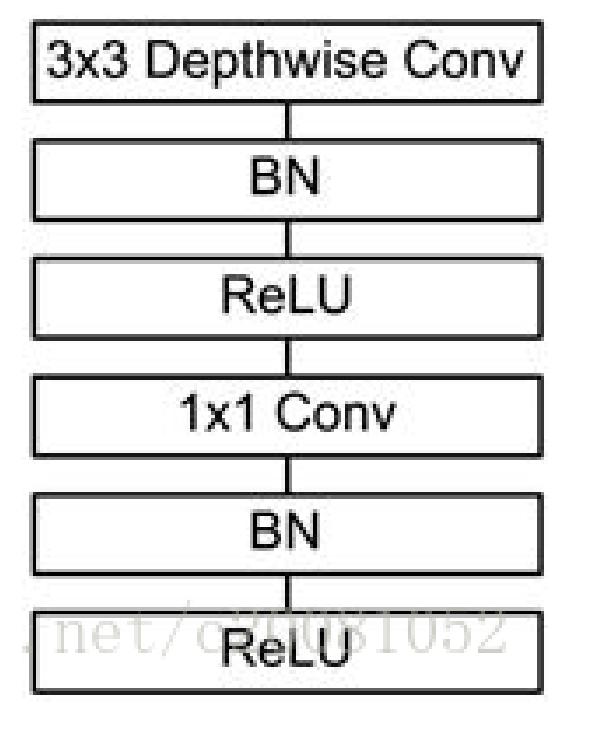
5)state-of-the-art 性能



MobileNet模型结构：

MobileNet的网络结构如下图所示。首先是一个3x3的标准卷积，然后后面就是堆积

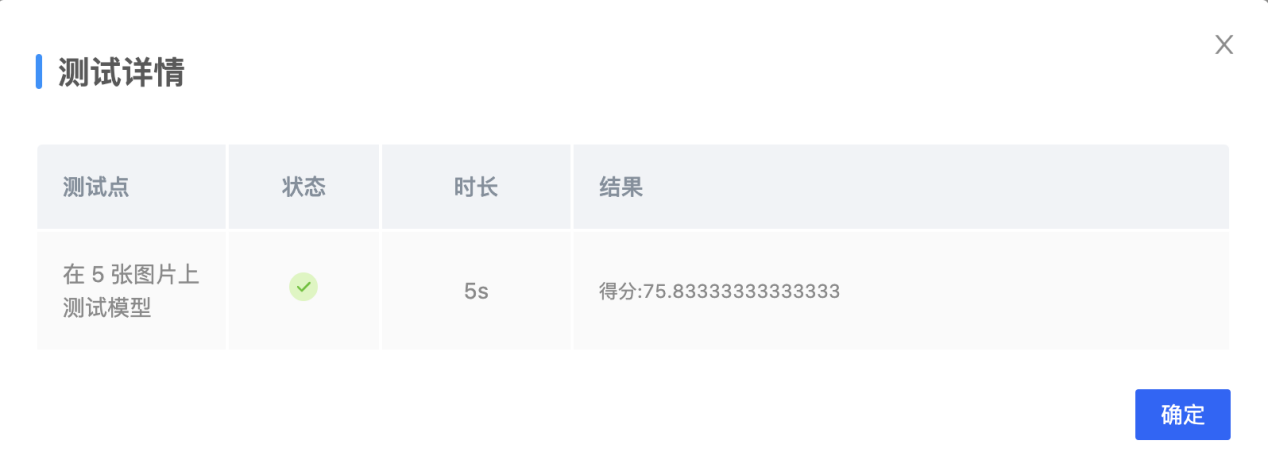
depthwise separable convolution，并且可以看到其中的部分depthwise convolution会通过strides=2进行down sampling。然后采用average pooling将feature变成1x1，根据预测类别大小加上全连接层，最后是一个softmax层。

****

**优化方向：**

这里我考虑到的优化方向有如下：

1. 首先对应用的模型进行比对测试，寻找针对口罩检测表现更加良好的模型，对比了resnet18和resnet50和MobilenetV1模型，经过测试发现还是MobileNetV1模型在这个应用上表现良好。
2. 考虑对数据库预处理进行优化，分别比对了添加了图片一定角度旋转和归一化时候更改新参数的测试结果，如下：



最后选择的代码如下：

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

])

1. 然后对学习率调度器scheduler进行尝试调整，其中optimizer是要调整学习率的优化器对象；max指定验证集上监测指标的模式。在这种情况下，调度器将监测验证集上的指标，并在其不再增加时降低学习率；factor是学习率降低的因子。当验证指标停止改善时，学习率将乘以factor；patience是检测验证指标停止改善的持续轮数，达到patience时候将降低学习率。然后通过对比调整测试发现最终代码如下：

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer,

'max',

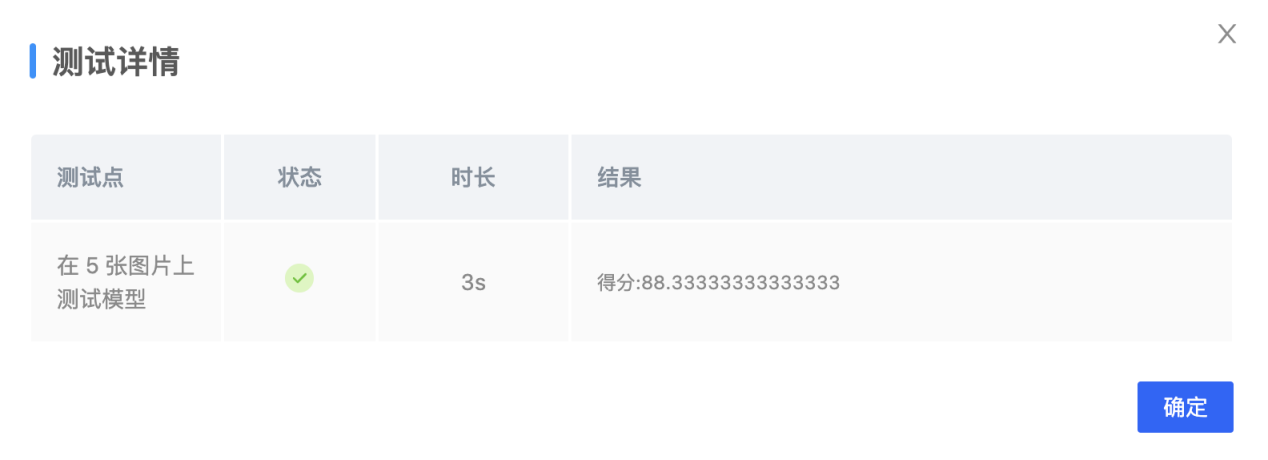
factor=0.6,

patience=2)

1. 对学习率lr进行调整，分别尝试了1e-3、2e-3、3e-3、9e-4等等，进行测试，最后

发现还是lr等于1e-3的时候是最高分，对比测试结果如下：





1. 对epochs进行调整，从10开始以10为跨度进行调整，直到epochs达到80，具体测试结果这里不进行展示，但是最后发现epochs等于20的时候是最高分。

**局限性：**

1）准确性与泛化性能： MTCNN和MobileNet都是经过训练和验证的模型，但它们的性能可能受限于训练数据集的多样性和数量。如果训练数据集中的人脸和口罩样本不够充分或不具有代表性，模型的准确性和泛化性能可能会受到影响。因此，确保训练数据集具有足够的多样性，并考虑使用更大规模的数据集进行训练，以提高模型的性能。

2）复杂场景下的表现： MTCNN和MobileNet在理想条件下可能表现良好，但在复杂场景下（如光照变化、遮挡、不同角度等）可能会遇到挑战。例如，人脸被遮挡部分或口罩样式多样性可能导致错误的检测或识别结果。针对这个问题，可以考虑引入更复杂的目标检测模型，如Mask R-CNN或更高级的模型，以提高在复杂场景下的性能。

3）计算资源要求： 使用MTCNN和MobileNet这样的模型进行口罩检测可能需要较大的计算资源，特别是在实时或较大规模应用中。这些模型的计算复杂度可能较高，可能需要较长的推理时间。为了克服这个局限性，可以考虑使用轻量级的模型或优化技术，如模型压缩、量化或模型剪枝，以减少计算资源的需求并提高推理速度。

1. **代码内容**

**Train.py**的代码是**：**

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.MobileNetV1 import MobileNetV1

torch.set\_num\_threads(6)

# 1.加载数据并进行数据处理

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=1000, 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, test\_split=0.1)

# 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

# 加载 MobileNet 的预训练模型权

device = torch.device("cuda:0") if torch.cuda.is\_available() else torch.device("cpu")

modify\_x, modify\_y = torch.ones((32, 3, 160, 160)), torch.ones((32, ))

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

optimizer = optim.Adam(model.parameters(), lr=1e-3) # 优化器

print('加载完成...')

# 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

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer,

'max',

factor=0.6,

patience=2)

# 损失函数

criterion = nn.CrossEntropyLoss()

epochs = 20

best\_loss = 1e9

best\_acc = 0

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

total = 0

right\_cnt = 0

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)

pred\_y = torch.max(output, 1)[1]

right\_cnt += (pred\_y == b\_y).sum()

total += b\_y.size(0)

accuracy = right\_cnt.float() / total

print('val acc:', accuracy)

if accuracy > best\_acc:

best\_model\_weights = copy.deepcopy(model.state\_dict())

best\_acc = accuracy

# loss\_list.append(loss)

# print('step:' + str(epoch + 1) + '/' + str(epochs) + ' || Total Loss: %.4f' % (loss, ))

# print('best\_loss: %.4f' % (best\_loss, ))

print('best\_accuracy: %.4f' % (best\_acc, ))

torch.save(best\_model\_weights, './results/temp.pth')

print('Finish Training.')

# 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。

**main.py**的代码是：

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

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

import numpy as np

# -------------------------- 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

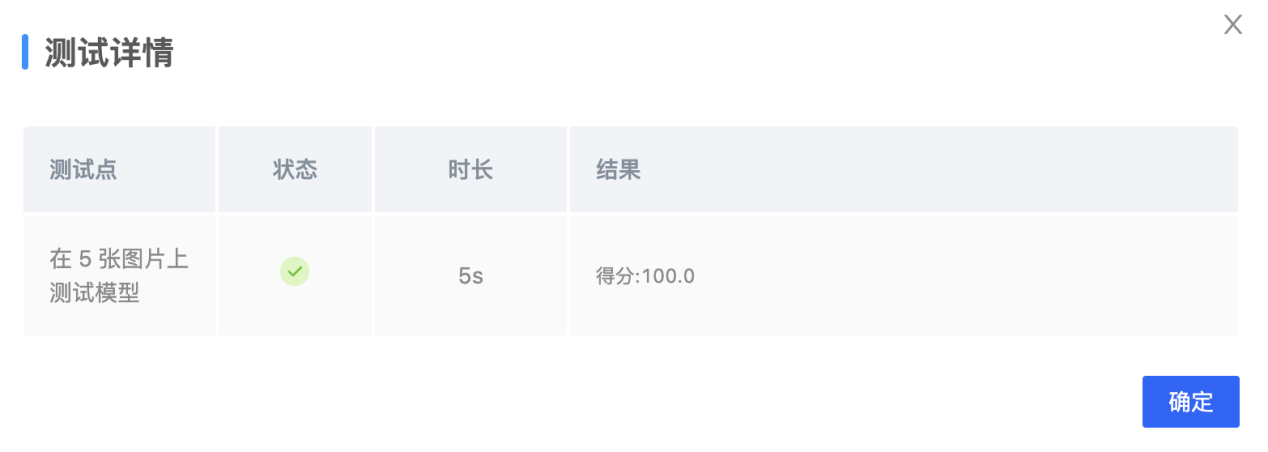
print("over")#标记程序正确执行完毕

1. **实验结果**

train模型的准确已经达到接近90%，而且在GPU上运行训练速度是3-4its/s，速度足够快，每次epochs等于20的训练只需要1-2分钟。速度有保障不会耽误时间。



mo平台测试结果：



1. **总结**

这里由于训练需要的算力较大，我这里采用了利用molab平台的GPU时间进行模型训练，由于之前注册过，我一共有4小时的GPU的时间，在这次不断调参的过程一共用了大量的GPU时间，提交了几十次来进行不断的参数调整，如下图：

截屏2023-05-20 下午8.55.59



额外考虑到的一些**更加深层次的优化方向**：

1.数据集增强和扩充： 确保训练数据集具有多样性，包括不同人脸姿势、光照条件、遮挡情况等。可以考虑使用数据增强技术来生成更多的样本，例如旋转、裁剪、缩放、遮挡等，以增加模型的泛化能力。

2.模型集成： 考虑采用模型集成的方法，将多个口罩检测模型或目标检测模型进行组合，以提高整体性能和鲁棒性。可以尝试使用投票或融合不同模型的输出来得出最终的口罩检测结果。

3.端到端的检测与识别模型：考虑采用端到端的检测与识别模型，如基于注意力机制的方法或一体化的目标检测与识别模型。这些模型可以同时处理人脸检测和口罩识别任务，通过联合训练来优化两个任务之间的关系，从而提高整体性能。

4.引入更大规模的数据集： 考虑使用更大规模的数据集进行训练，特别是口罩识别方面的数据集。这将有助于模型更好地学习和泛化口罩的特征，提高口罩识别的准确性。可以寻找公开的大规模口罩数据集或考虑数据采集和标注来增加训练数据的多样性和数量。

5.模型优化和推理加速： 探索模型优化和推理加速的方法，以提高口罩检测的效率和实时性。可以尝试使用模型压缩技术、量化、剪枝等方法来减少模型的参数量和计算复杂度，从而提高推理速度和节约计算资源。

6.针对口罩样式和遮挡的处理： 针对不同口罩样式和遮挡情况，可以进一步优化模型的鲁棒性。例如，收集包含各种口罩样式和遮挡情况的样本，针对性地进行训练和数据增强，以提高模型对口罩变化的适应能力。

**心得与感想总结**：

在进行口罩检测的实验过程中，我遇到了一些问题，但也获得了宝贵的经验和知识。首先，我注意到数据集的重要性。发现在系统给出的数据集中出现了各种各样的口罩和人员和佩戴方式。

其次，我意识到计算资源的重要性。在实验中，我使用了MobileNet作为口罩识别模型，但由于其计算复杂度较高，导致推理速度较慢。这对于实时应用或大规模应用来说是不可接受的。通过研究和学习，我了解到可以使用模型压缩、量化或剪枝等技术来减少模型的参数量和计算复杂度，从而提高推理速度和节约计算资源。这将是我进一步优化口罩检测方法的重要方向之一。

而且在实验过程中，不断的枯燥而复杂的调参过程是十分折磨的，磨练了我的耐心和心态，让我在之后的学习过程中有更好的韧性和耐力。

综合来说，通过这次口罩检测实验，我获得了宝贵的经验和知识。我学会了如何选择和组合不同的模型来解决特定的任务，以及如何处理数据集的多样性和规模问题。我也认识到了口罩检测任务面临的挑战，并对进一步优化方法有了清晰的思路。