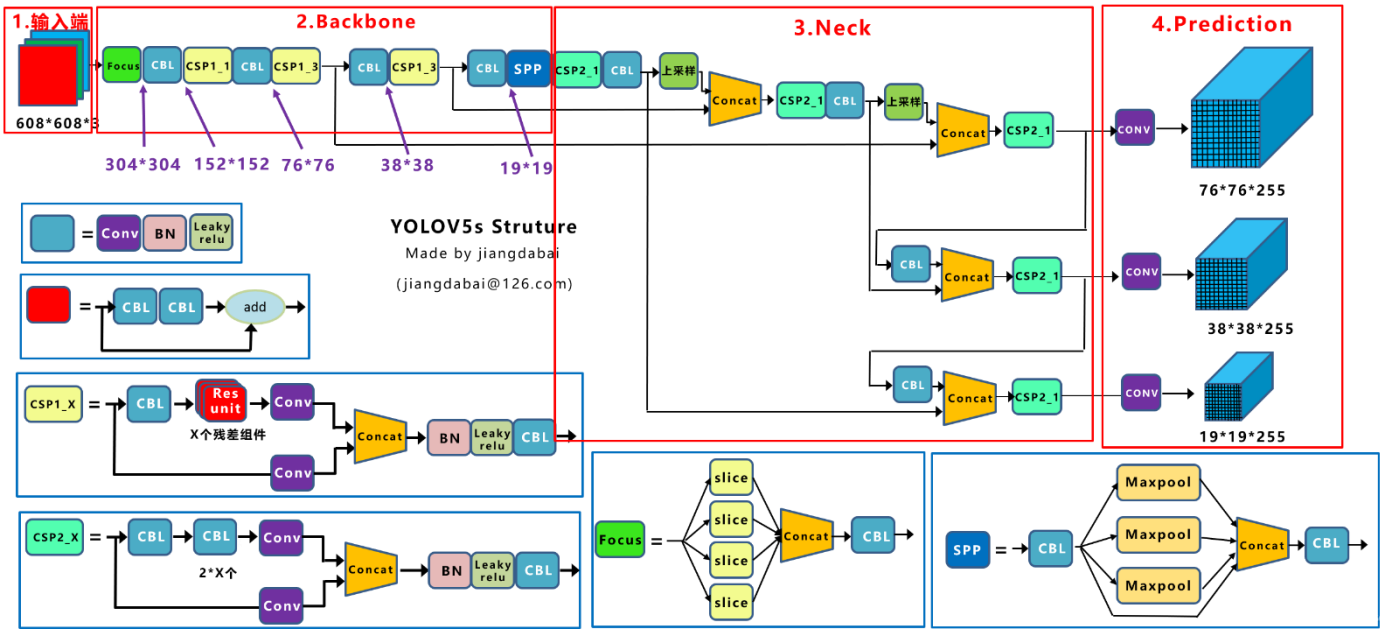
（一）实现方法

1.

Yolo v5 模型结构：



YOLOv5 的网络结构分为 输入端、Backbone、Neck、输出端 四个部分。

输入端 主要包括 Mosaic 数据增强、图片尺寸处理以及自适应锚框计算三部分。Mosaic 数据增强将四张图片进行组合，达到丰富图片背景的效果；图片尺寸处理对不同长宽的原始图像自适应的添加最少的黑边，统一缩放为标准尺寸；自适应锚框计算在初始锚框的基础上，将输出预测框与真实框进行比对，计算差距后再反向更新，不断迭代参数来获取最合适的锚框值。

Backbone 主要包含 CSP 和 Focus 模块。CSP 模块前面的卷积核的大小都是 3*3，stride=2，因此可以起到下采样的作用，采用 CSP 模块先将基础层的特征映射划分为两部分，然后通过跨阶段层次结构将它们合并，在减少了计算量的同时可以保证准确率。Focus 模块对图片进行切片操作，将输入通道扩充为原来的 4 倍，并经过一次卷积得到下采样特征图，在实现下采样的同时减少了计算量并提升了速度。

Neck 采用了 FPN 与 PAN 结合的结构，将常规的 FPN 层与自底向上的特征金字塔进行结合，将所提取的语义特征与位置特征进行融合，同时将主干层与检测层进行特征融合，使模型获取更加丰富的特征信息。

输出端 输出一个向量，该向量具有目标对象的类别概率、对象得分和该对象边界框的位置。

检测网络由三层检测层组成，不同尺寸的特征图用于检测不同尺寸的目标对象。每个检测层输出相应的向量，最后生成原图像中目标的预测边界框和类别并进行标记。

2.

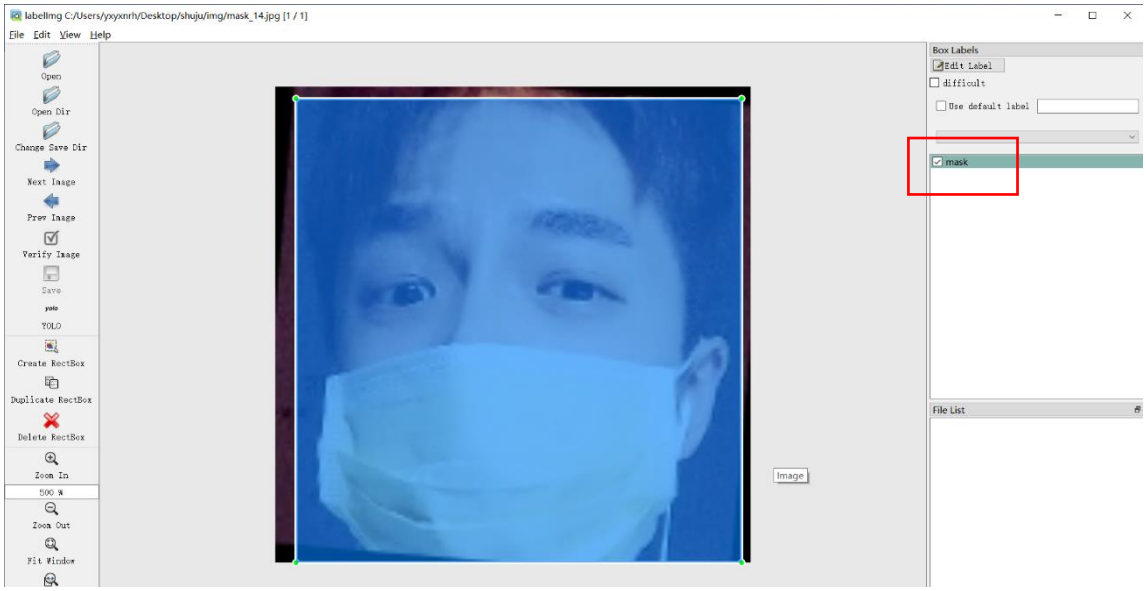
数据集介绍：

验证集（300 张，自己人工标注）

Yolo v5 模型读取 yolo 格式的图片数据（带标签），使用 labelimg 来为图片人工加上标签：在项目 cmd 里 pip install labelimg，成功安装 labelimg。然后启动 labelimg

```
C:\Users\yxyxnrh\Desktop\ex2\yolov5-mask-42-master>conda activate

(base) C:\Users\yxyxnrh\Desktop\ex2\yolov5-mask-42-master>conda activate py21

(py21) C:\Users\yxyxnrh\Desktop\ex2\yolov5-mask-42-master>labelimg
```

然后从网上下载有口罩的人脸或无口罩的人脸，自己人工标注，有口罩为 mask，没有则为 face，共标注 300 张图



训练集（网上下载，选取 1000 张）

因为训练集数量较大，所以从网上下载现有的标注好的 volo 格式的图片（别人已经用 labelimg 标注好的），开源口罩佩戴检测数据集介绍:

https://zhuanlan.zhihu.com/p/103922982?utm_source=wechat_session

数据集百度云盘地址：https://pan.baidu.com/share/init?surl=J8SwsZ9F5XFbUlHpEVQK3A

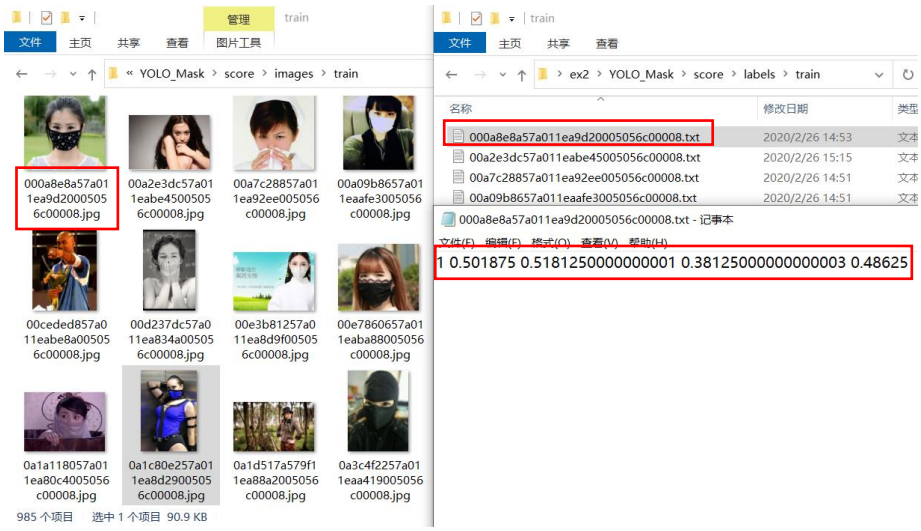
提取码：h9sl

此数据集总数超过 8000 张，开源项目的作者以爬取百度与谷歌的图片-口罩、隔离病房、医院监控等关键字图片为主，后经过 labelimg 处理。

测试集

从训练集中随机选出 300 张

以上三类数据集比例大致为 2：6：2，符合 yolo 模型学习的规律。通过 labelimg 标注处理过的图片，会生成与图片对应的 label（标签，txt 文件），如图



txt 中的 1 与 0 表示戴或不戴口罩，后面 4 个数值表示标签的 x，y 坐标与长，宽

标记完成的数据按照下面的格式进行放置，方便程序进行索引。

```
YOLO_Mask #数据集文件夹
└── score
        ├── images #图片数据文件夹
        |     ├── test # 下面放测试集图片
        |     ├── train # 下面放训练集图片
        |     └── val # 下面放验证集图片
        └── labels #标签数据文件夹
              ├── test # 下面放测试集标签
              ├── train # 下面放训练集标签
              ├── val # 下面放验证集标签
```

损失函数：

作为目标检测任务，使用 CIoU Loss 来作为 Bounding Box Regeression Loss（回归损失函数）：

IoU Loss：

1 减去 预测框与目标框的交集 / 预测框与目标框并集

$$L_{IoU} = 1 - IoU = 1 - \frac{I(X)}{U(X)}$$

GIoU Loss:

解决 IoU 的缺点：当预测框和目标框不相交时，IoU(A,B)=0 时，不能反映 A,B 距离的远近，此时损失函数不可导，IoU Loss 无法优化两个框不相交的情况。为了解决边界框不重合时的问题。

GIoU 的实现方式如下，其中 C 为预测框 A 和目标框 B 的外接矩形。用 C 减去 A 和 B 的并集除以 C 得到一个数值，然后再用框 A 和 B 的 IoU 减去这个数值即可得到 GIoU 的值。

$$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$$

L(GIoU) = 1 - GIoU

DIoU Loss：

解决 GIoU 缺点，当目标框完全包裹预测框的时候，IoU 和 GIoU 的值都一样，此时 GIoU 退化为 IoU，无法区分其相对位置关系；所以 DIoU 加入了中心点归一化距离，考虑边界框中心点距离的信息

$$DIoU = \frac{\rho^2(A,B)}{c^2}$$

A,B 分别表示预测框和目标框的中心点， ρ (A,B)表示欧氏距离，c 表示两个框的最小外接矩形的对角线距离，DIoU Loss 全式为:

$$L_{DIoU} = 1 - IoU + \frac{\rho^2(A,B)}{c^2}$$

CIoU Loss：

在 DIoU Loss 基础上增加了一个影响因子，将预测框长宽比拟合目标框的长宽比考虑进去，考虑边界框宽高比的尺度信息

$$CIoU = \frac{\rho^2(A,B)}{c^2} + \alpha v$$

α 是用于做 trade-off 的参数

$$\alpha = \frac{v}{(1 - IoU) + v}$$

v 是用来衡量长宽比一致性的参数

$$v = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2$$

$$CIoU = 1 - IoU + \frac{\rho^2(A,B)}{c^2} + \alpha v$$

Yolov5 中采用以上的 CIOU Loss 做 Bounding box 的损失函数，代码如下。

```
def bbox_iou(box1, box2, x1y1x2y2=True, GIoU=False, DIoU=False, CIoU=False, eps=1e-7):
    """在 ComputeLoss 的__call__函数中调用计算回归损失
    :params box1: 预测框
    :params box2: 预测框
    :return box1 和 box2 的 IoU/GIoU/DIoU/CIoU
    """
    box2 = box2.T

    # Get the coordinates of bounding boxes
    if x1y1x2y2:  # x1, y1, x2, y2 = box1
        b1_x1, b1_y1, b1_x2, b1_y2 = box1[0], box1[1], box1[2], box1[3]
        b2_x1, b2_y1, b2_x2, b2_y2 = box2[0], box2[1], box2[2], box2[3]
    else:  # transform from xywh to xyxy
        b1_x1, b1_x2 = box1[0] - box1[2] / 2, box1[0] + box1[2] / 2
        b1_y1, b1_y2 = box1[1] - box1[3] / 2, box1[1] + box1[3] / 2
        b2_x1, b2_x2 = box2[0] - box2[2] / 2, box2[0] + box2[2] / 2
        b2_y1, b2_y2 = box2[1] - box2[3] / 2, box2[1] + box2[3] / 2

    # Intersection area   tensor.clamp(0): 将矩阵中小于0的元数变成0
    inter = (torch.min(b1_x2, b2_x2) - torch.max(b1_x1, b2_x1)).clamp(0) * \
            (torch.min(b1_y2, b2_y2) - torch.max(b1_y1, b2_y1)).clamp(0)

    # Union Area
    w1, h1 = b1_x2 - b1_x1, b1_y2 - b1_y1 + eps
    w2, h2 = b2_x2 - b2_x1, b2_y2 - b2_y1 + eps
    union = w1 * h1 + w2 * h2 - inter + eps

    iou = inter / union
    if GIoU or DIoU or CIoU:
        cw = torch.max(b1_x2, b2_x2) - torch.min(b1_x1, b2_x1)  # 两个框的最小闭包区域的width
        ch = torch.max(b1_y2, b2_y2) - torch.min(b1_y1, b2_y1)  # 两个框的最小闭包区域的height
        if CIoU or DIoU:  # Distance or Complete IoU https://arxiv.org/abs/1911.08287v1
            c2 = cw ** 2 + ch ** 2 + eps  # convex diagonal squared
            rho2 = ((b2_x1 + b2_x2 - b1_x1 - b1_x2) ** 2 +
                    (b2_y1 + b2_y2 - b1_y1 - b1_y2) ** 2) / 4  # center distance squared
            if DIoU:
                return iou - rho2 / c2  # DIoU
```

```
            elif CIoU:  # https://github.com/Zzh-tju/DIoU-SSD-pytorch/blob/master/utils/box/box_utils.py#L47
                v = (4 / math.pi ** 2) * torch.pow(torch.atan(w2 / h2) - torch.atan(w1 / h1), 2)
                with torch.no_grad():
                    alpha = v / (v - iou + (1 + eps))
                return iou - (rho2 / c2 + v * alpha)  # CIoU
        else:  # GIoU https://arxiv.org/pdf/1902.09630.pdf
            c_area = cw * ch + eps  # convex area
            return iou - (c_area - union) / c_area  # GIoU
    else:
        return iou  # IoU
```

3.

训练过程：

在 models 文件夹下建立一个 mask_yolov5s.yaml 的模型配置文件，设定 nc =2 表示这次训练的分类有两类：face 和 mask：

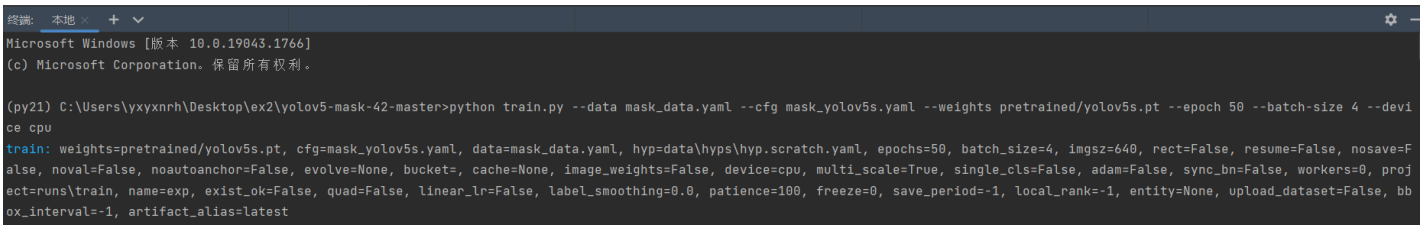
```yaml
# Parameters
nc: 2  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
```

在 mask_data.yaml 设定训练集和验证集数据的地址路径（images 和 labels 文件夹一一对应），设定好训练的类数（两类），类名（mask,face）

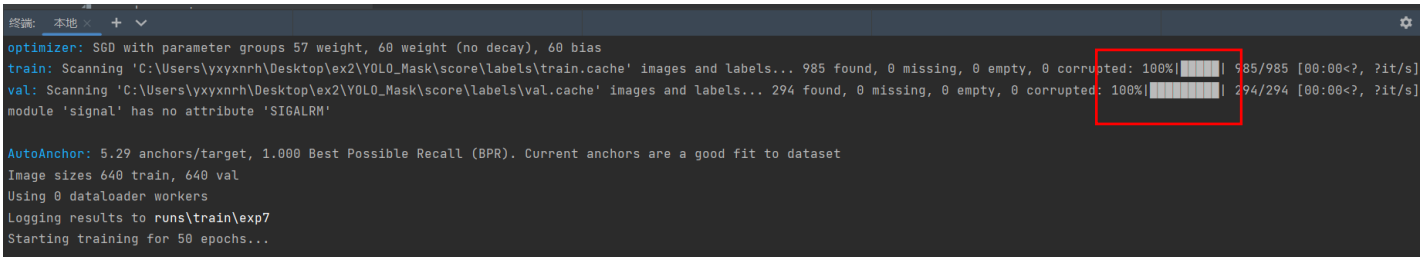
mask_data.yaml | yolov5s.yaml | train.py | loss.py | lr_scheduler.py | mask_yolov5s.yam

```yaml
# train and val data as 1) directory: path/images/, 2) file: path/images.txt, or 3)
train: C:/Users/yxyxnrh/Desktop/ex2/YOLO_Mask/score/images/train     #训练集图片路径
val: C:/Users/yxyxnrh/Desktop/ex2/YOLO_Mask/score/images/val         #验证集图片路径

# number of classes
nc: 2

# class names
names: ['mask', 'face']
```

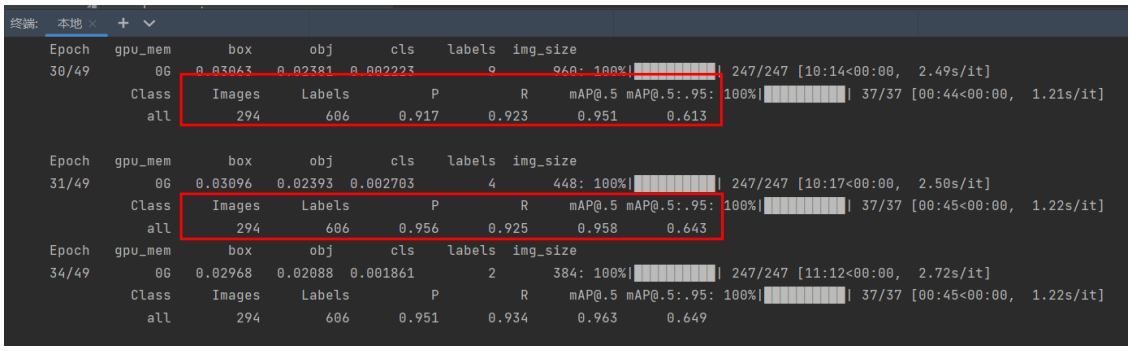
在终端输入以下指令：python train.py --data mask_data.yaml --cfg mask_yolov5s.yaml --weights pretrained/yolov5s.pt --epoch 50 --batch-size 4，表示读取扫描 mask_data.yaml 的路径的数据集，借助 yolo v5 的预训练文件 yolov5s.pt（小图片处理）训练，设定训练参数（batch size）大小为 4，且训练 50 轮（大概 8-10 个小时）



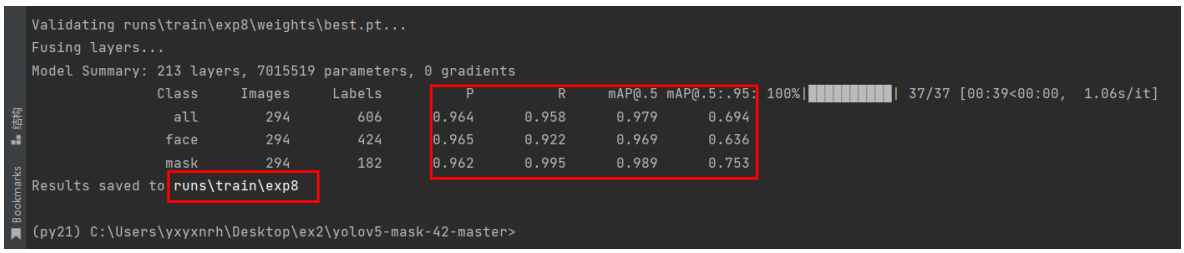
开始训练，开始扫描 images 和 labels 文件夹里的图片与标签数据，扫描成功：



我们能看到模型训练过程中有训练次数(Epoch)，标签(Labels)，P(precision)，R(Recall)，mAP 这些参数的显示，而且 mAP 的值越来越大接近于 1，说明准确率和召回度在提升

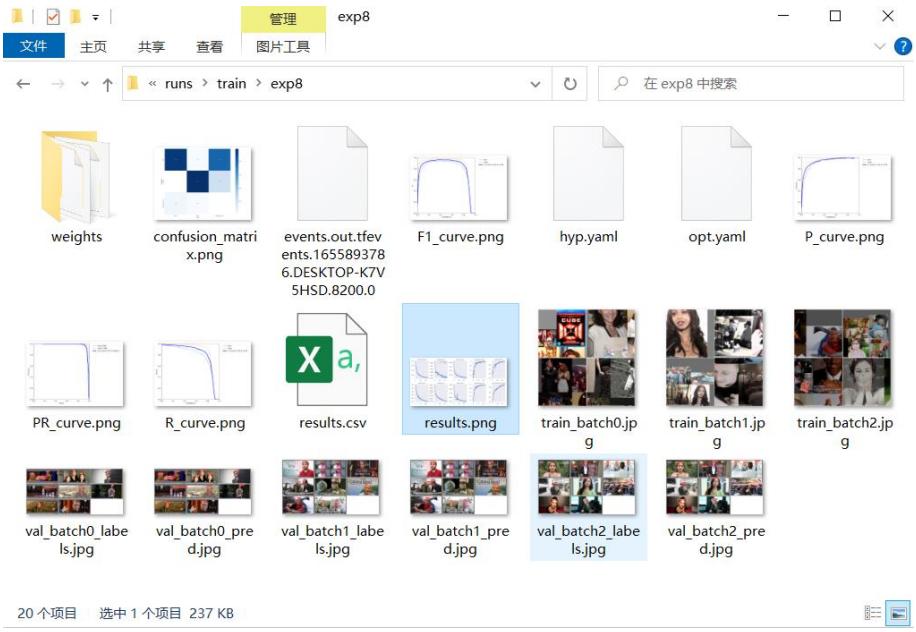


训练成功，权重文件和验证准确率的结果被存储在 exp8 文件夹，且输出所有类别（总共，face-人脸，mask-口罩）训练过程的各个指标（Precision，Recall，mAP0.5，mAP0.5:0.95）：

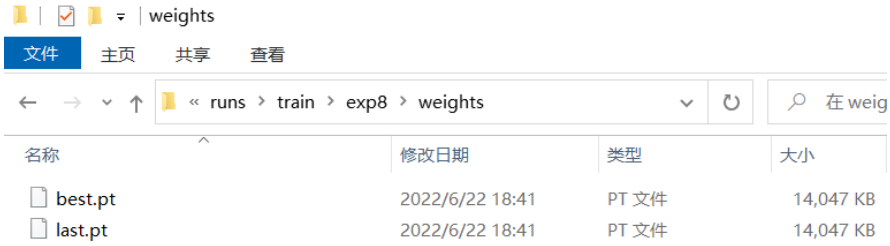


yolov5 每次 train 完成（如果没有中途退出）都会在 run 目录下生成 expX 目录（X 代表生成结果次数 第一次训练完成生成 exp0 第二次生成 exp1.....以此类推）。expX 目录下会保存训练生

成 weights 以及指标图。下图是 ex8 中模型训练过程中生成的一些指标图

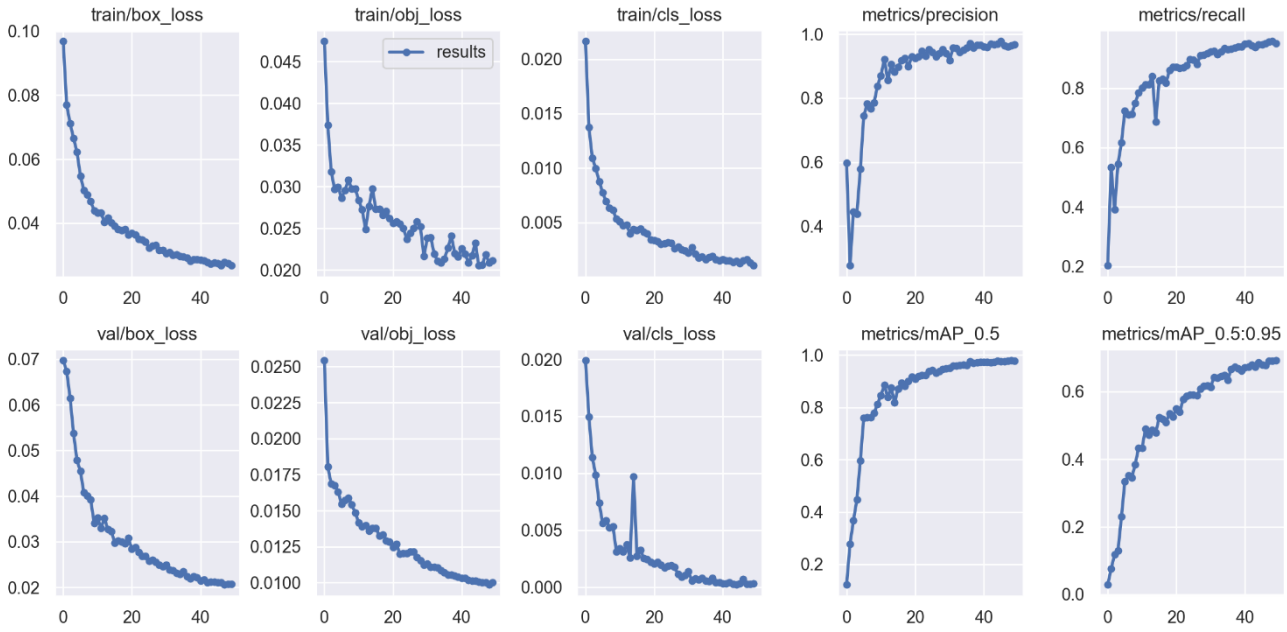


weights 文件夹里是生成的.pt 权重文件，best.pt 是实现模型识别效果最好的，last.pt 是最新一次生成的



Loss 变化:

在上面所说的文件夹里找到 yolo 模型自动生成的训练(train)和验证(val)过程中 loss 变化示意图：



从上到下，从左到右依次是：

box_loss：YOLO V5 使用 CIOU Loss 作为 bounding box 的损失，Box 为 CIoU 损失函数均值，方框越小识别越准

val box: 验证集 bounding box 损失

object_loss：目标检测 loss 均值，越小目标检测越准

val object_loss：验证集目标检测 loss 均值

cls_loss：推测为分类 loss 均值，越小分类越准

val cls_loss：验证集分类 loss 均值

precision：精度（找对的正类 / 所有找到的正类）

mAP_0.5：表示阈值大于 0.5 的平均 mAP

recall：真实为 positive 的准确率，即正样本有多少被找出来

mAP_0.5:0.95：表示在不同 IoU 阈值（从 0.5 到 0.95，步长 0.05）上的平均 mAP

综上所述，模型在训练和验证过程中，loss 不断变小趋近于 0，精度不断变大趋近于 1

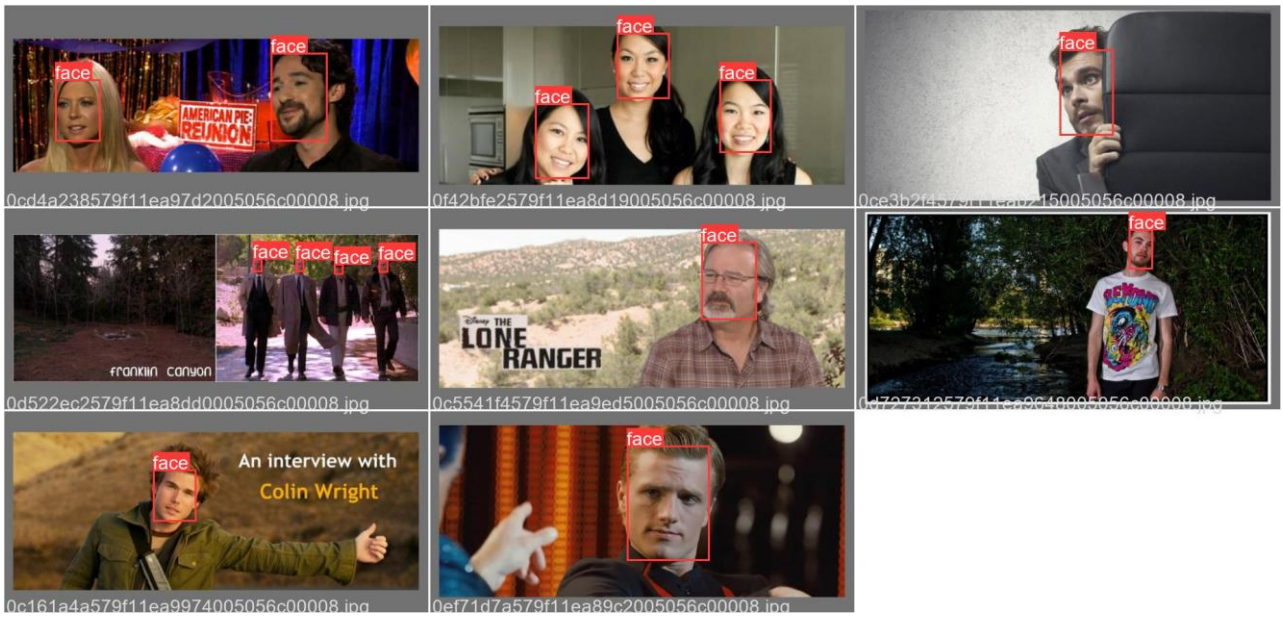
（二）实现结果

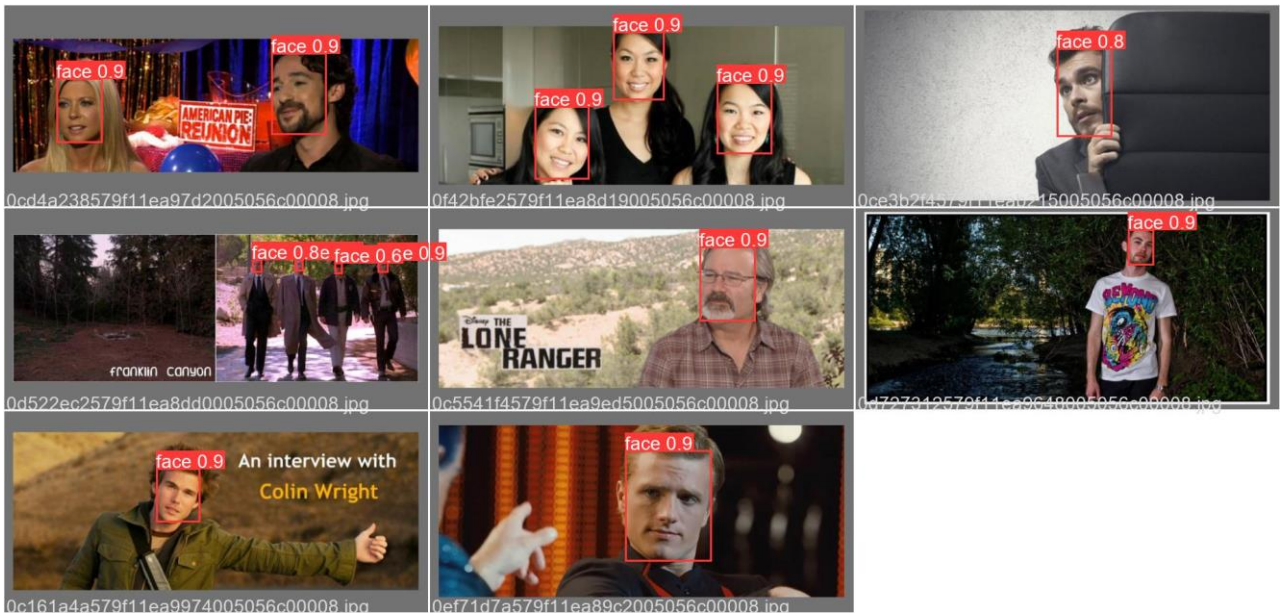
1.

测试结果：

以下为训练过程中产生的示例，8 张合成一新图（上为 labels 表示打上标签，下为 pred 表示预测概率）：

示例 1：





示例 2：


face
STRACTA
HÔTEL
TCG +00:27:11;13
mask
Catalina Island
Chamber of Commerce & Visitor Bureau
Wer oder was sind Gelatkins!


face 0.9
face 0.8
STRACTA
HÔTEL
TCG +00:27:11;13
mask 0.9
Catalina Island
Chamber of Commerce & Visitor Bureau
Wer oder was sind Gelatkins!

示例 3：

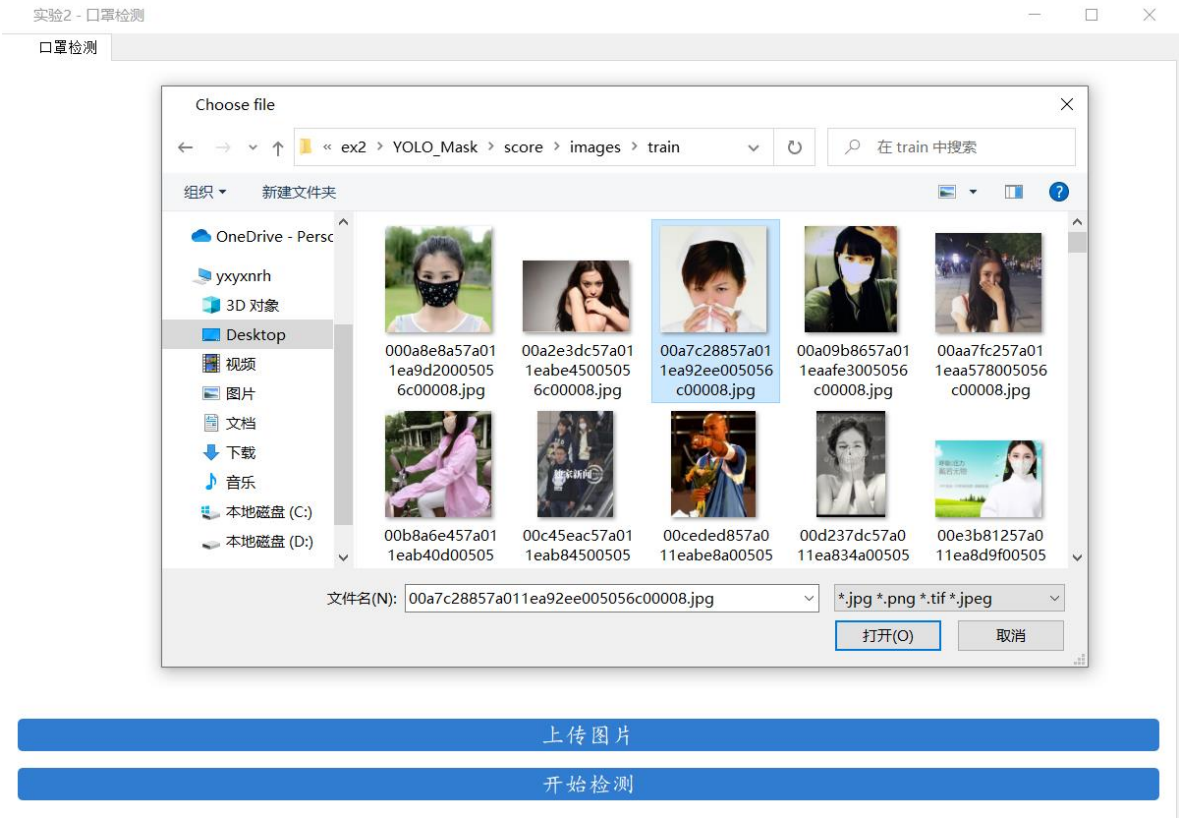
尝试加入 qt 界面：

下载 pyqt 包，在 window.py 里设定一个窗口主类，定义图片检测界面与 UI:

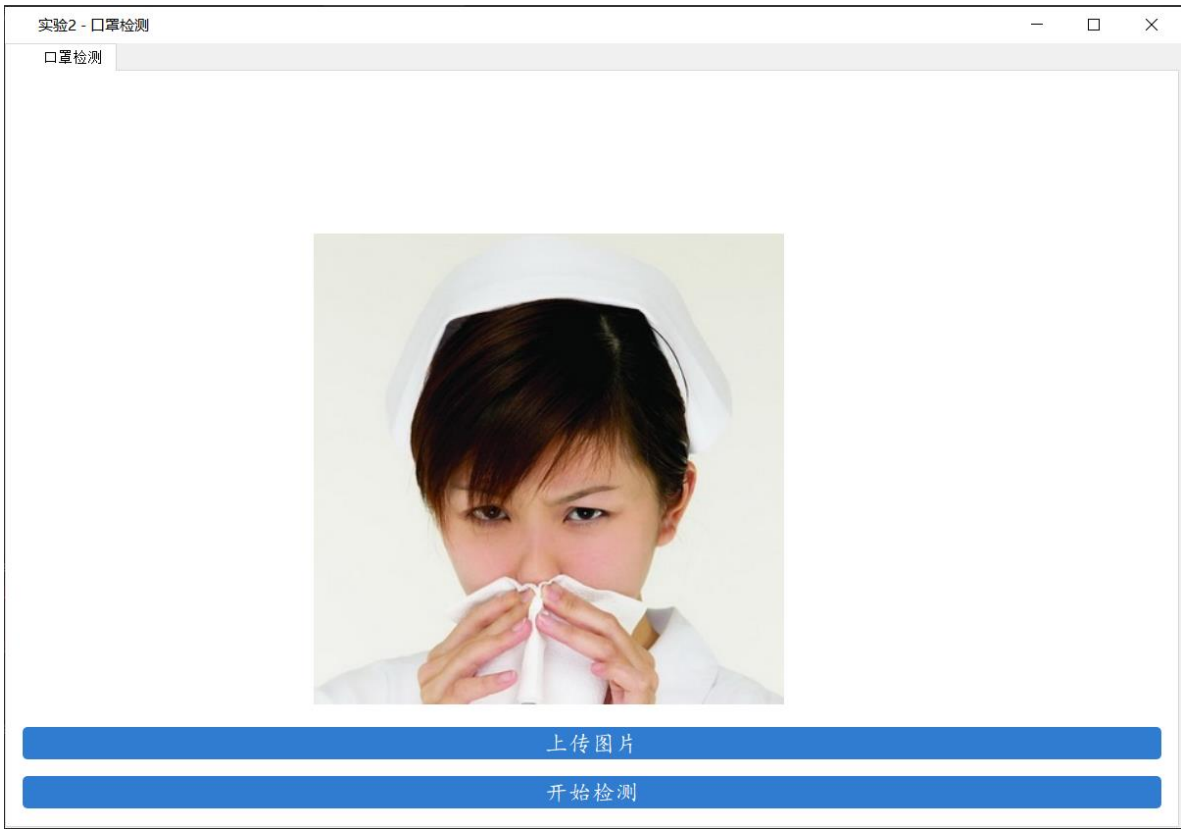
```
def initUI(self):
        font_title = QFont('楷体', 16)
        font_main = QFont('楷体', 14)
        img_detection_widget = QWidget()
        img_detection_layout = QVBoxLayout()
        img_detection_title = QLabel("")
        img_detection_title.setFont(font_title)
        mid_img_widget = QWidget()
        mid_img_layout = QHBoxLayout()
        self.left_img = QLabel()
        self.right_img = QLabel()
        self.left_img.setPixmap(QPixmap("images/UI/1.png"))
        self.right_img.setPixmap(QPixmap("images/UI/1.png"))
        self.left_img.setAlignment(Qt.AlignCenter)
        self.right_img.setAlignment(Qt.AlignCenter)
        mid_img_layout.addWidget(self.left_img)
        mid_img_layout.addStretch(0)
        mid_img_layout.addWidget(self.right_img)
        mid_img_widget.setLayout(mid_img_layout)
        up_img_button = QPushButton("上传图片")
        det_img_button = QPushButton("开始检测")
        up_img_button.clicked.connect(self.upload_img)
        det_img_button.clicked.connect(self.detect_img)
        up_img_button.setFont(font_main)
        det_img_button.setFont(font_main)
```

同理，定义按钮等 qt 事件，使用训练好的权重文件，最后右键运行 window.py，跳出如下界面：

点击“上传图片”：


实验2 - 口罩检测
口罩检测
Choose file
ex2 › YOLO_Mask › score › images › train
在 train 中搜索
组织
新建文件夹
OneDrive - Perso
yxyxnrh
3D 对象
Desktop
视频
图片
文档
下载
音乐
本地磁盘 (C:)
本地磁盘 (D:)
00a8e8a57a011ea9d20005056c00008.jpg
00a2e3dc57a011eabe45005056c00008.jpg
00a7c28857a011ea92ee005056c00008.jpg
00a09b8657a011eaafe3005056c00008.jpg
00aa7fc257a011eaa578005056c00008.jpg
00b8a6e457a011eab40d00505
00c45eac57a011eab84500505
00ceded857a011eabe8a00505
00d237dc57a011ea834a00505
00e3b81257a011ea8d9f00505
文件名(N): 00a7c28857a011ea92ee005056c00008.jpg
*.jpg *.png *.tif *.jpeg
打开(O)
取消
上传图片
开始检测

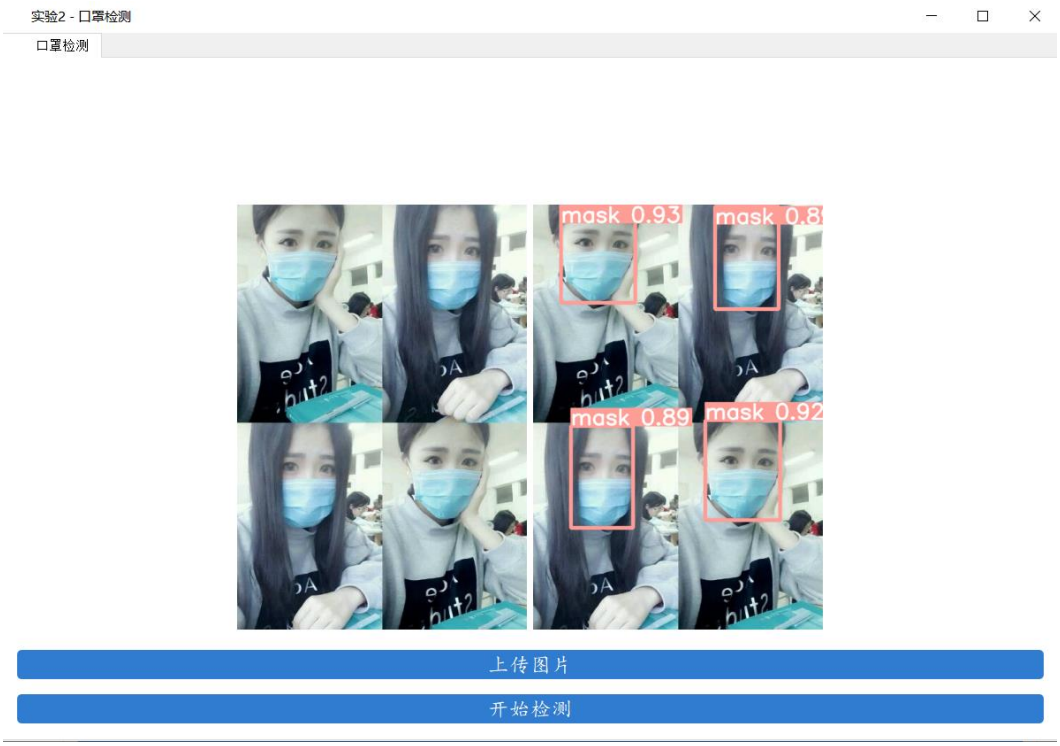
上传成功：


实验2 - 口罩检测
口罩检测
上传图片
开始检测

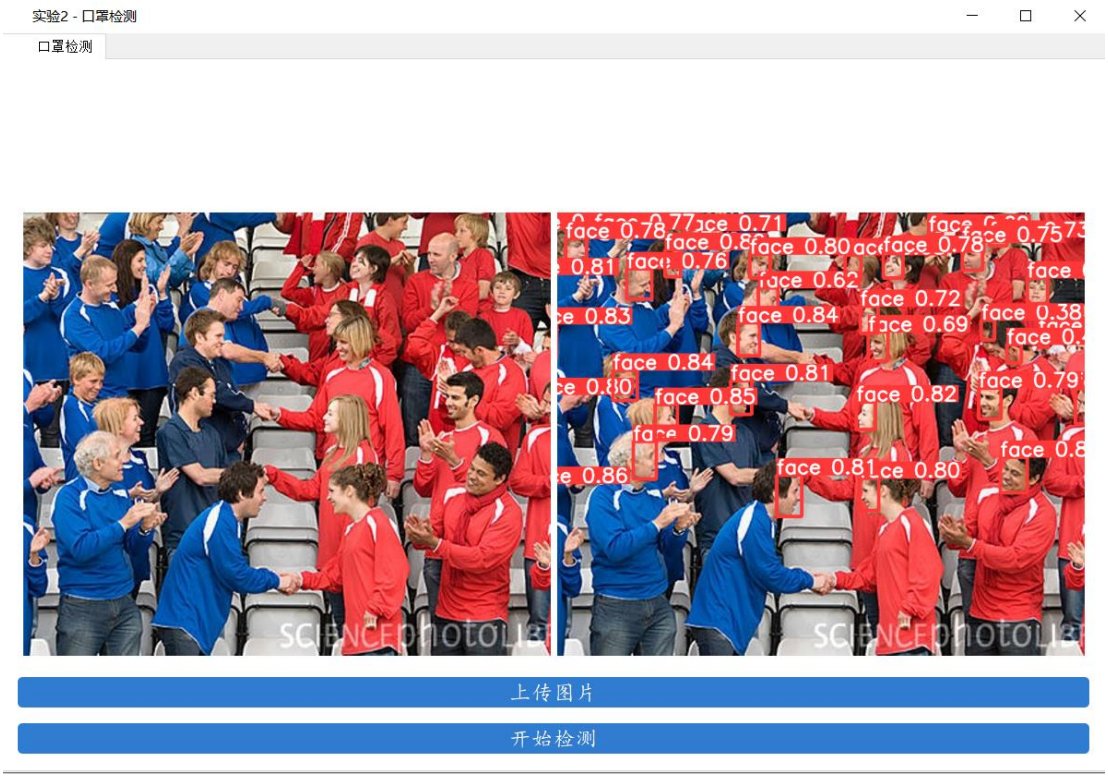
点击开始检测：

同理测试以下图片：

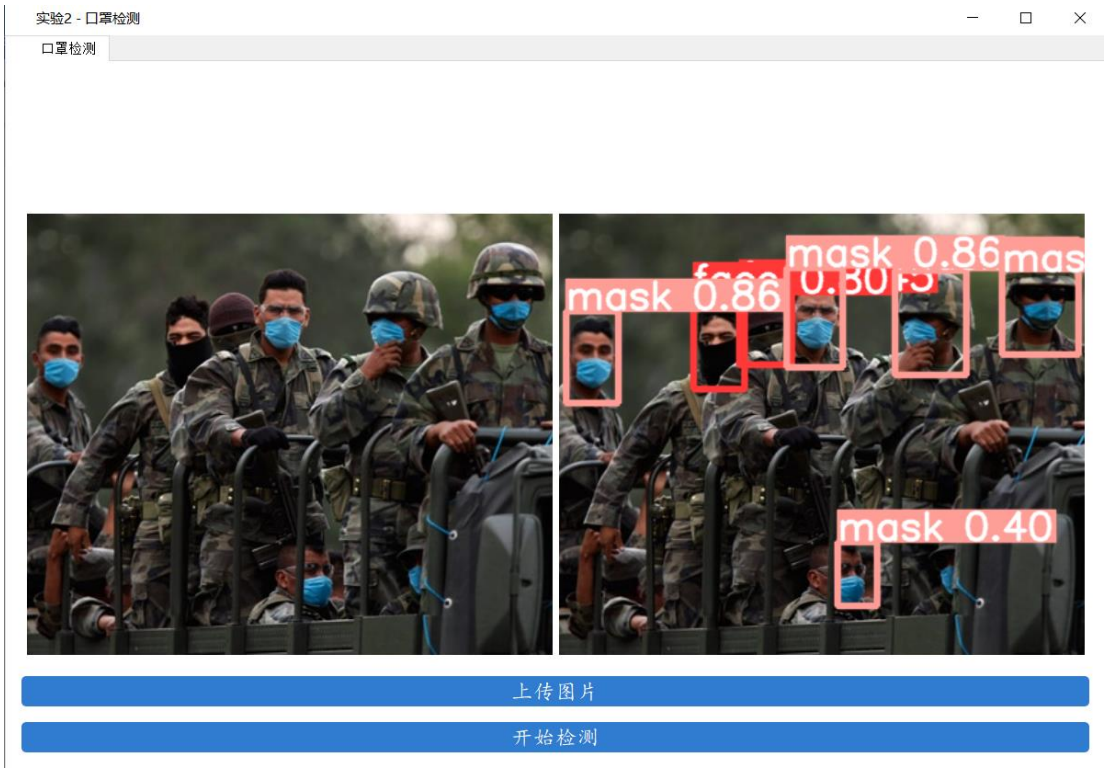
（1）



（2）

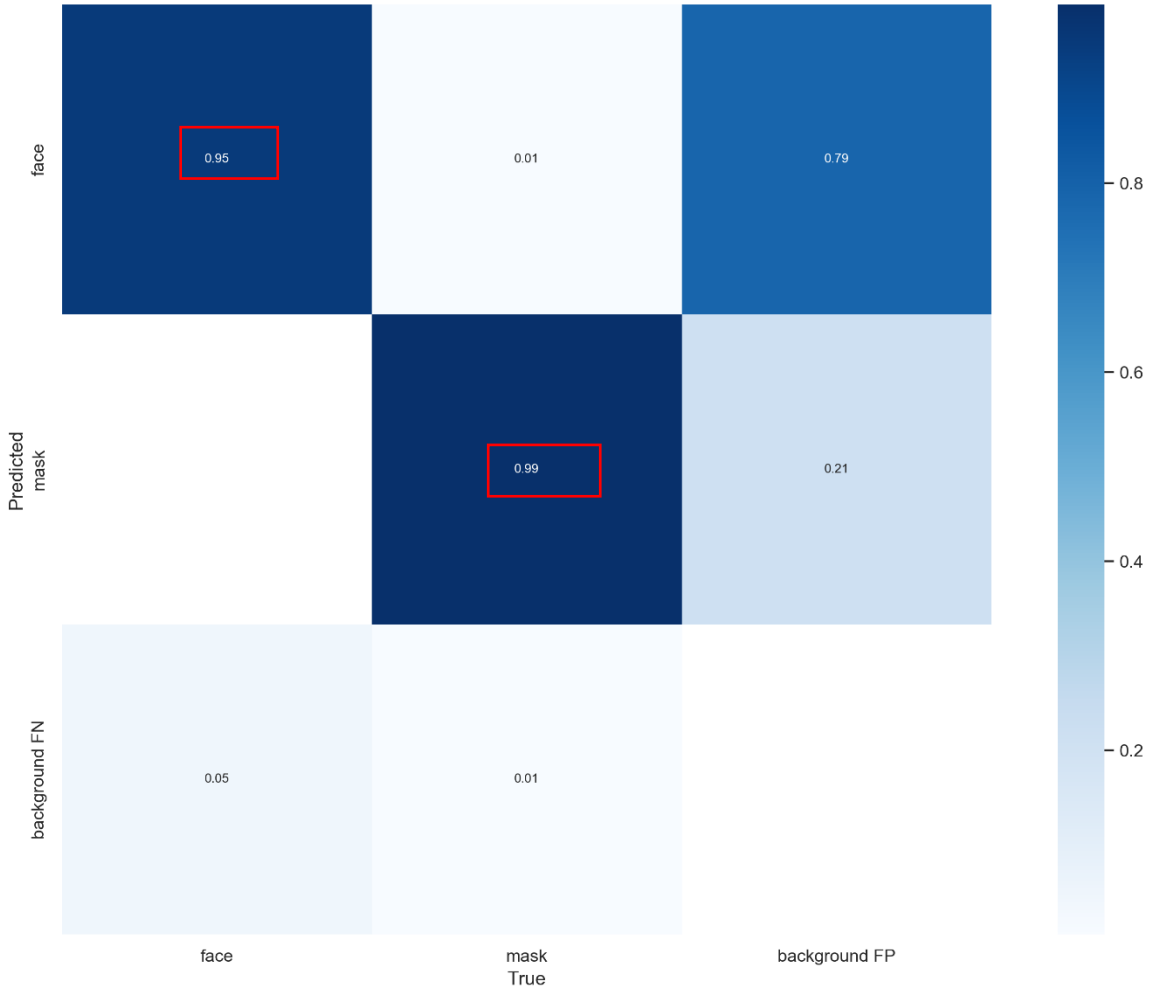

实验2 - 口罩检测
口罩检测
face 0.78
face 0.80
face 0.76
face 0.62
face 0.72
face 0.84
face 0.69
face 0.84
face 0.81
face 0.85
face 0.82
face 0.79
face 0.81
face 0.80
上传图片
开始检测

（3）


实验2 - 口罩检测
口罩检测
mask 0.86
mask 0.40
上传图片
开始检测

2. 准确率评价：

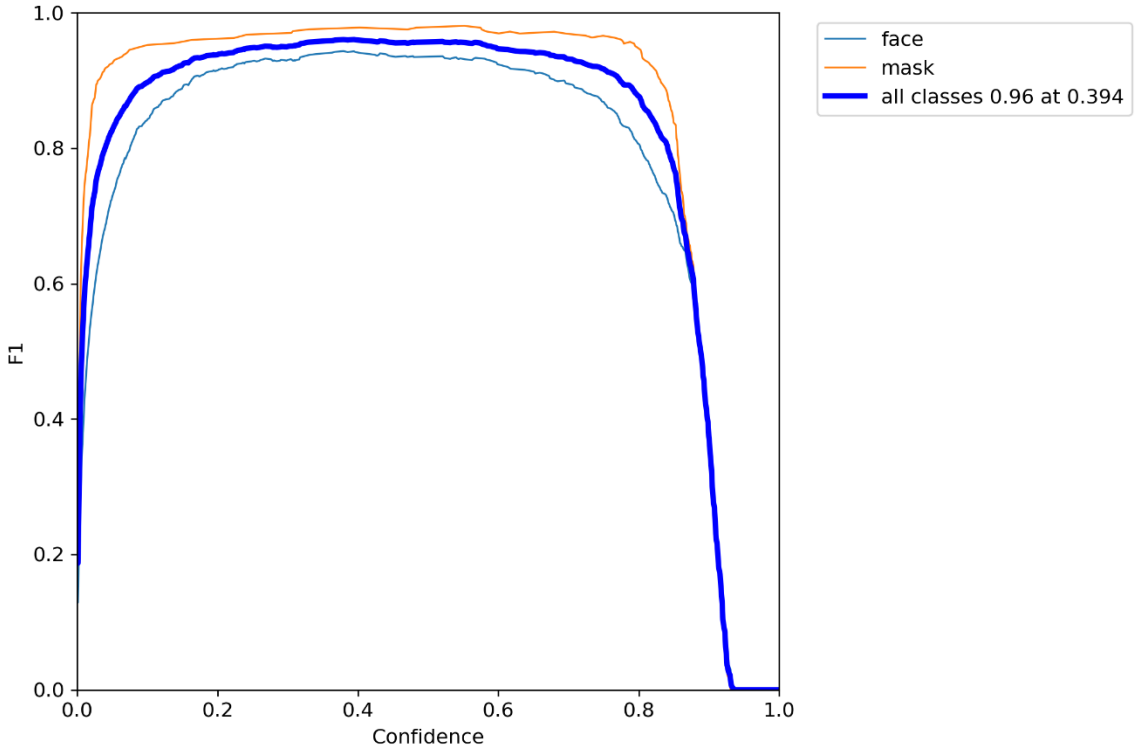
Ex8 文件夹下 yolo 训练生成的混淆矩阵（confusion_matrix.png）



上图表示该模型在类别判断上的精度，左上的“face”为 0.95，表示人脸检测精度在 0.95 左右，中间的 mask 为“0.99”，表示口罩检测精度在 0.99 左右

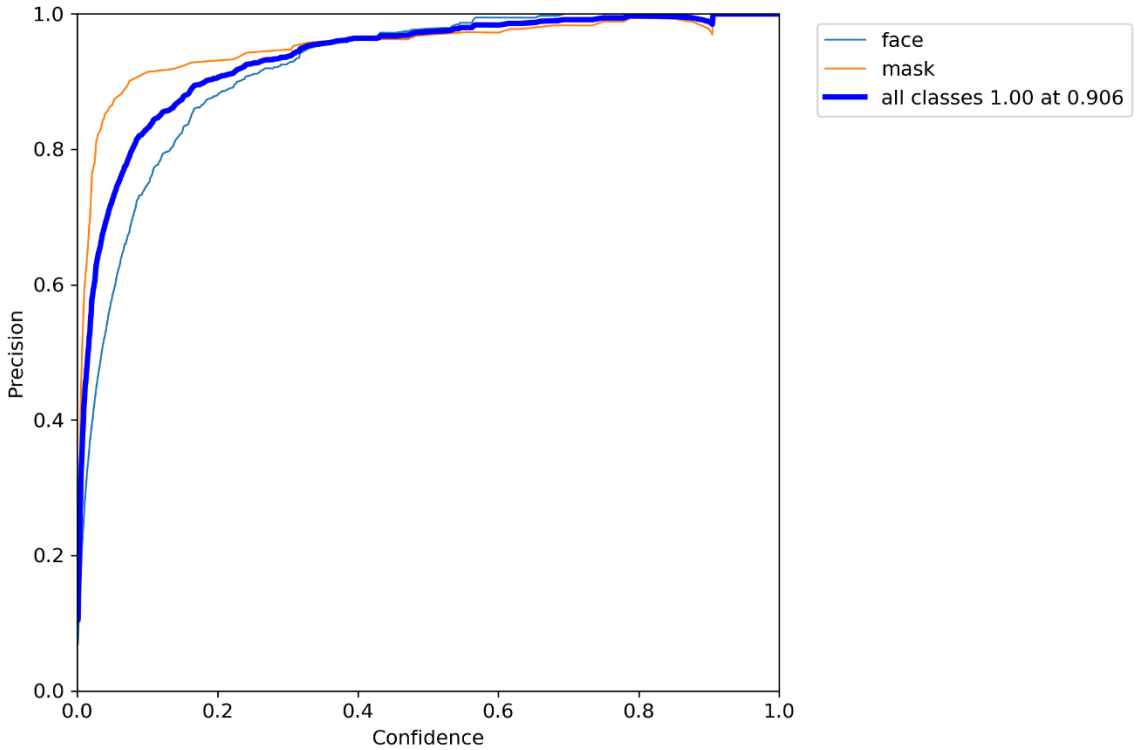
F1_curve.png：

F1 分数（F1-score）是分类问题的一个衡量指标，是精确率 precision 和召回率 recall 的调和平均数，最大为 1，最小为 0, 1 是最好，0 是最差。如下图可知 F1-score 平均在 0.96

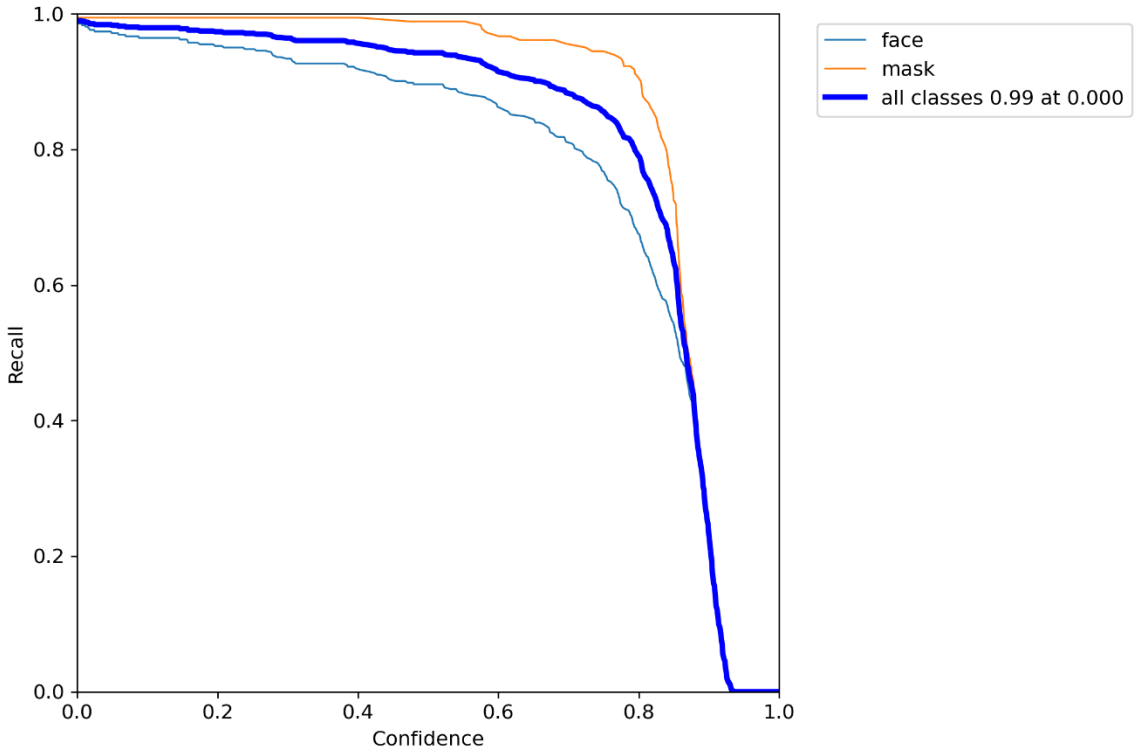


P_curve.png：

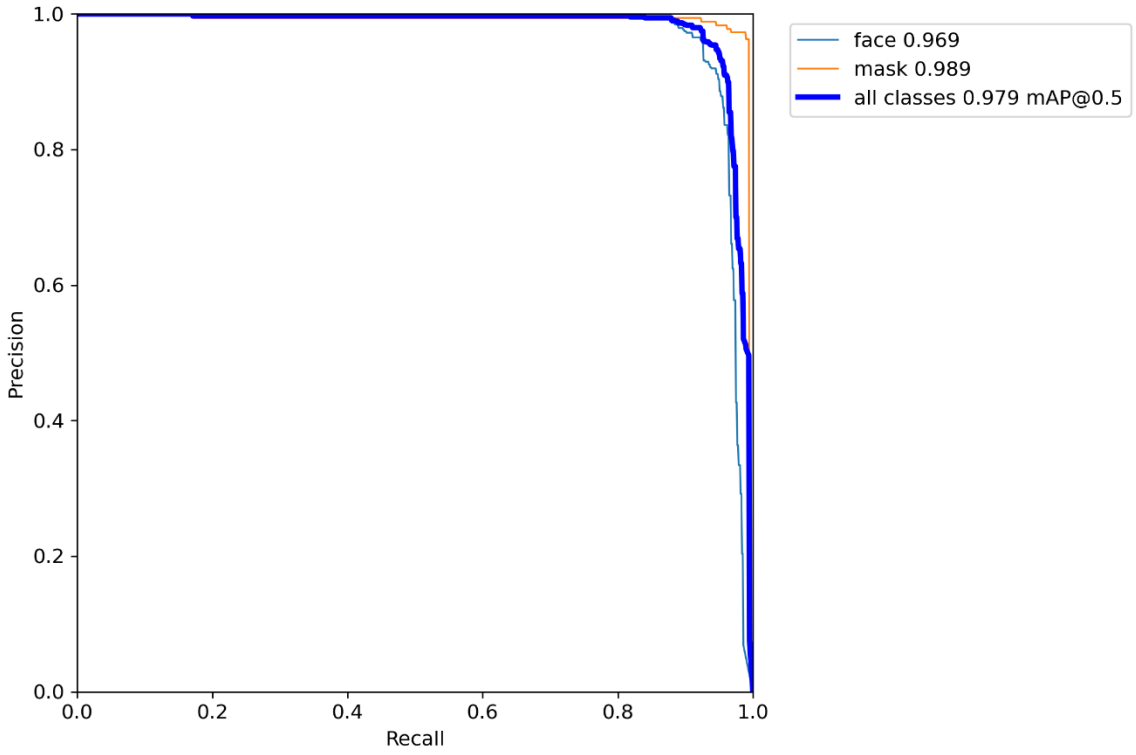
准确率 precision 和置信度 confidence 的关系图，准确率趋近于 1



R_curve.png：召回率在 0.99 左右



PR_curve.png：模型在验证集上的均值平均密度（mAP）为 0.979



综上所述，我们模型的精确度还是十分理想的，能够达到较为精确识别口罩与人脸的要求

（三）结论分析

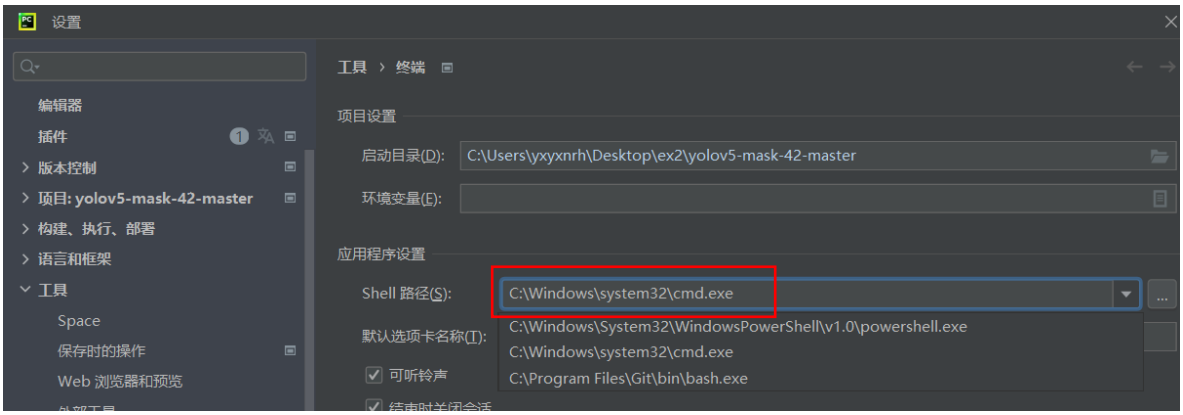
1. 出现的问题与解决方法。

1）问题：

因为是在桌面上新建的文件夹 cmd 里进行 conda，安装虚拟环境的，发现在 cmd 中可以直接 python detect.py 运行成功，但是在 pycharm 的项目，通过终端，进行 python detect.py 发现会提示缺少 cv2 的库，但是我的虚拟环境明明已经提示已安装 opencv-python （Requirement already satisfied: opencv-python in）。

解决方案：

最后还是在 pycharm 设置中更改 Shell 路径，将 powershell.exe 改成 cmd.exe，将终端的 PS 换成 cmd，测试成功:



发现这样运行成功：

```
(py21) C:\Users\yxyxnrh\Desktop\ex2\yolov5-mask-42-master>python detect.py --source data/images/bus.jpg --weights pretrained/yolov5s.pt
detect: weights=['pretrained/yolov5s.pt'], source=data/images/bus.jpg, imgsz=[640, 640], conf_thres=0.5, iou_thres=0.45, max_det=1000, d
image 1/1 C:\Users\yxyxnrh\Desktop\ex2\yolov5-mask-42-master\data\images\bus.jpg: 640x480 3 persons, 1 bus, Done. (0.022s)
Speed: 0.0ms pre-process, 22.0ms inference, 4.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp5

(py21) C:\Users\yxyxnrh\Desktop\ex2\yolov5-mask-42-master>
```

2）问题：

使用 labelimg 软件对图片进行标注（face 或 mask）时，发现 images 文件夹对应的 labels 文件夹里的数据都是.xml 格式，因此导致 mask_data.yaml 中无法读取数据集

解决方案：最后发现 labelimg 的“保存格式”务必要设置为“YOLO”格式，才能使得 labels 文件夹里生成的标注数据都是.txt 格式

2. 尚存在的问题。

1）目前对数据集的训练要求需要有标签的数据图片，单纯使用 labelimg 对图片进行手工标签来区分 face 和 mask 确实有些麻烦，为了方便目前只能从网上下载特定的有标签图片，而且生成的标签必须符合 yolo 格式。

2）Yolo v5 确实性能更快，但其对于小目标的检测还是有一定的瓶颈，特别是大分辨率图像的小目标检测，如果采用直接对输入端输入高分辨率原图的方式，很多小目标都无法检测出。