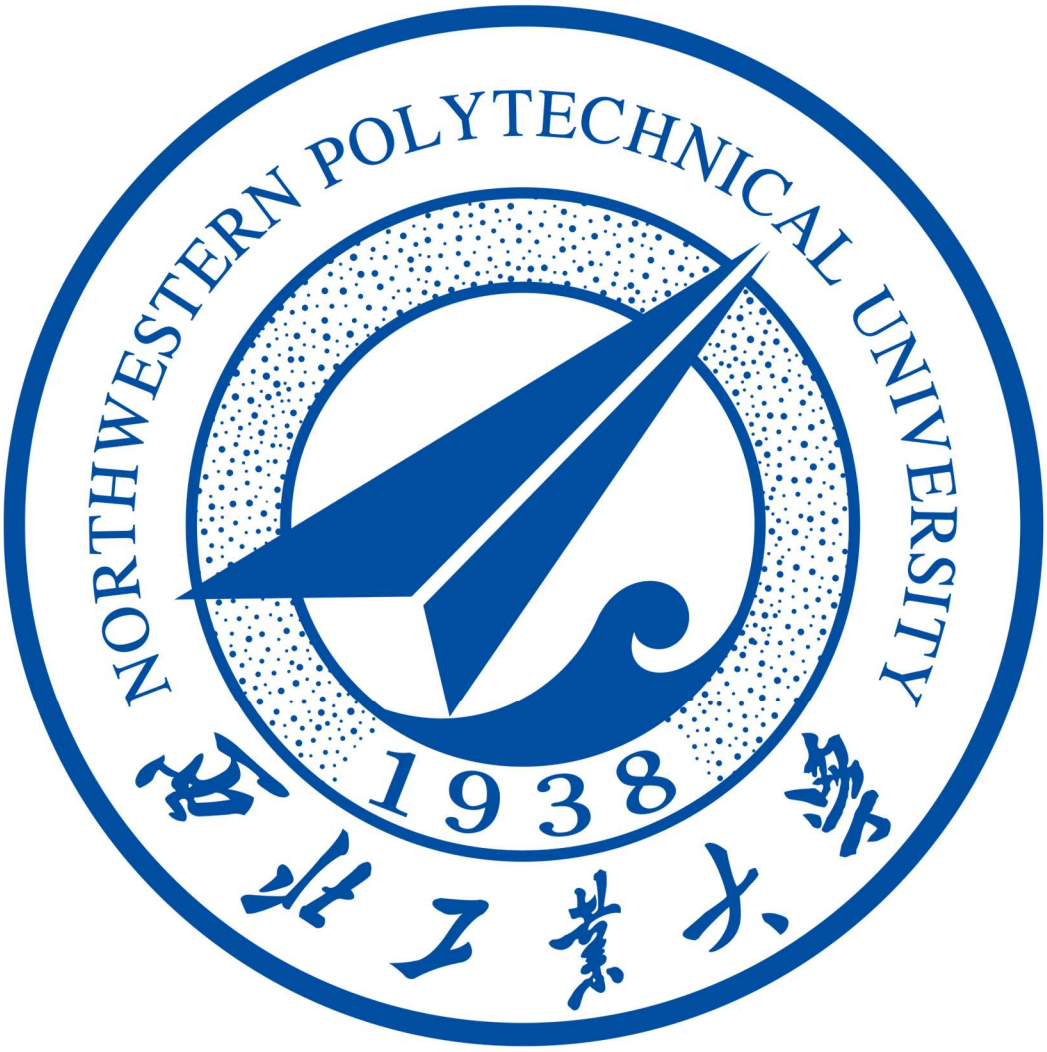


机器视觉：理论和基于 Pytorch 的实践

基于 YOLOV5 的口罩检测模型

学号：2021302330

姓名：张晶玮

目录

摘要

在疫情防控趋于常态化的当下，口罩已经成为我们外出必不可少的一件装备，带着口罩的人脸识别呈现出一定难度和迫切的现实需求，所以我希望基于 YOLOv5 算法复现一个口罩检测系统，使用 YOLOv5 训练检测人脸和口罩的模型，并使用 pyqt5 做一个图形化的界面。本文介绍了 YOLOv5 模型的框架和具体的训练过程。

1 问题分析

基本功能：用户提交一张照片/一段视频。训练好的模型能够自动框出人脸在画面中的位置，并判断是否佩戴口罩，标注 face/mask。

问题拆分：需要能够实现多个目标检测的模型；需要足够多的人脸及戴口罩的人脸的数据集以供训练和检测。

拓展需求：尝试不同的训练参数（置信度、NMS、训练轮数 epoch）进行训练，分析它们对最终结果产生的影响。

2 方法介绍

2.1 模型选择

目标识别是一种基于目标几何和统计特征的图像分割。主要评价指标有：交并比(IoU)、准确率(Precision)、召回率(Recall)、单类平均准确率(Average-Precision)等。

人脸和口罩检测的模型属于目标检测模型，是对物体分类之后的更进一步，涵盖类别和位置信息。除了传统的 CV 算法，基于深度学习的经典检测方法包括 one-stage 和 two-stage，后者如 RCNN 等多了一步提取出候选框的操作，纵然精度较高但麻烦、处理时间长，前者如 YOLO 是从 CNN 特征回归直接得到检测目标，损失了一定的精度但便捷快速，适合用于对实时性要求高的检测，如视频检测、实时监测等。

YOLOv5(You Only Look Once)是一种有监督的深度学习算法，可以实现 end to end，输入一张图片不需要经过任何处理直接通过模型就可以得到结果。它的基本思想如下图所示：经过卷积神经网络得到一定大小的特征图，将特征图划分为 S*S 的栅格(grid cells)，每个栅格负责对落入其中的目标进行检测，一次性预测所有栅格所含目标的边界框(bounding boxes)、定位置信度(confidence)、以及所有类别概率向量(class probability map)，其中：边界框表征目标的大小以及精确位置，置信度表征预测框的可信程度，取值范围 0~1，值越大说明该矩形框中越可能存

在目标，类别概率向量表征目标所属类别的可能性。

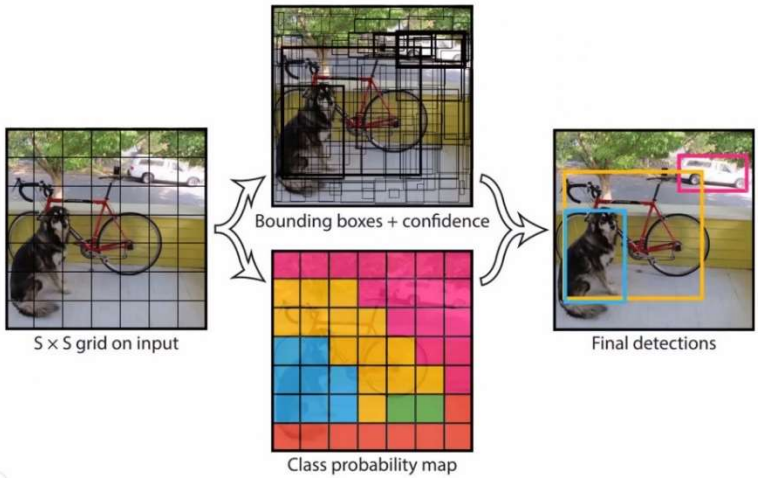


图 1 YOLO 算法基本思想

在实际检测的步骤如下：首先判断每个预测框的预测置信度是否超过设定阈值，若超过则认为该预测框内存在目标，从而得到目标的大致位置。接着根据非极大值抑制算法(NMS)对存在目标的预测框进行筛选，剔除对应同一目标的重复矩形框。最后根据筛选后预测框的分类概率，取最大概率对应的索引，即为目标的分类索引号，从而得到目标的类别。

YOLOv5 提供了 YOLO5s、YOLOv5m、YOLOv5l、YOLOv5x 的四种模型，这几种模型的网络结构相似，YOLOv5s 是四种网络中最小最基础的模型，其他的三种模型都是在其基础上加深深度和加宽宽度。YOLOv5s 模型基于残差网络(ResNet)架构，采用多尺度预测的方式，用于目标检测和识别。它的优点在于尺寸小、速度快，适合在移动设备或嵌入式系统中使用。但由于模型尺寸较小，准确率可能会有所下降。但由于考虑到我拥有设备的算力大小，和需要的模型规模大小，选择使用 YOLOv5s 模型，最符合设计需求。

2.2 网络结构

YOLOv5 的网络结构主要由 Backbone、Neck、Head 组成，其中 Backbone 主要使用 CSPdarknet+SPP 结构，Neck 使用 PANet 结，Head 使用 yolov3 head。

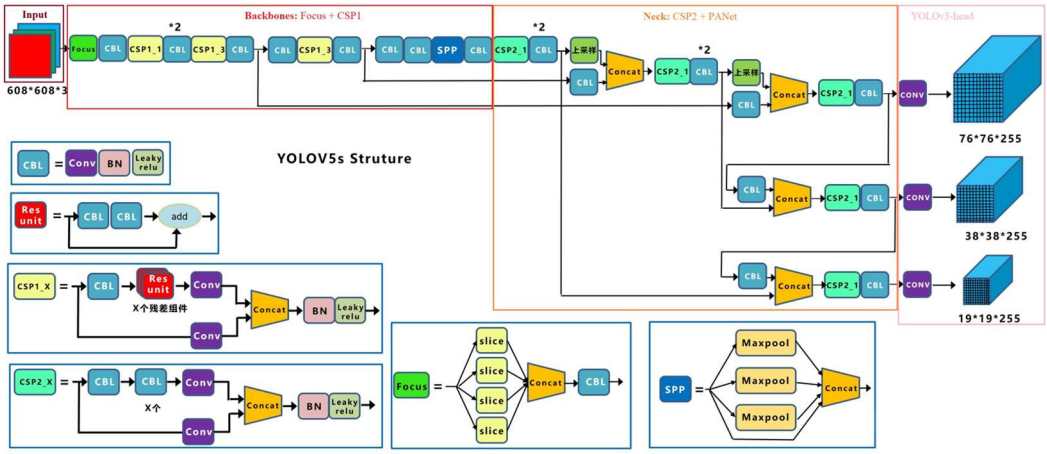


图 2 YOLOv5 总体网络结构

2.2.1 Anchor 设定

YOLOv5 采用 anchor-based 的方法进行目标检测，使用不同尺度的 anchor 直接回归目标框并一次性输出目标框的位置和类别置信度。最初的 YOLOv1 的初始训练过程很不稳定，在 YOLOv2 的设计过程中，作者观察了大量图片的 ground truth(gt)，发现相同类别的目标实例具有相似的 gt 长宽比：比如车的 gt 都是矮胖的长方形，行人的 gt 都是瘦高的长方形。所以作者受此启发，从数据集中预先准备几个概率比较大的 bounding box，再以它们为基准进行预测，如下：

```
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
```

其中，浅层的特征图（P3），包含较多的低层级信息，适合用于检测小目标，所以这一特征图所用的 anchor 尺度较小；同理，深层的特征图（P5），包含更多高层级的信息，如轮廓、结构等信息，适合用于大目标的检测，所以这一特征图所用的 anchor 尺度较大。P4 特征图上就用介于这两个尺度之间的 anchor 用来检测中等大小的目标。YOLOv5 之所以能高效快速地检测跨尺度目标，这种对不同特征图使用不同尺度的 anchor 的思想功不可没。

对于大部分图片而言，由于其尺寸与我们预设的输入尺寸不符，所以在输入阶段就做了 resize，导致预先标注的 bounding box 大小也发生变化。而 anchor 是根据我们输入网络中的 bounding box 大小计算得到的，所以在这个 resize 过程中就存在 anchor 重新聚类的过程。在 yolov5/utils/autoanchor.py 文件下，有一个函数 kmeans_anchor，通过 kmeans 的方法计算得到 anchor。

2.2.2 Backbone 特征提取骨干网络模块

（1）Focus

Focus 是一种对 feature map 的切片操作把宽度(width)和高度(height)的信息整合到通道(channel)维度，具体来说就是将相距为 2 的四个位置进行堆叠到一个通道上去，因此长和宽都缩小两倍，通道数增加了 4 倍，Focus 模块设计用于降低 FLOPS 和提高速度，而不是提高 mAP。

```
class Focus(nn.Module):
    # Focus wh information into c-space
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel, stride, padding, groups
        super(Focus, self).__init__()
        self.conv = Conv(c1 * 4, c2, k, s, p, g, act)

```

```
    def forward(self, x):  # x(b,c,w,h) -> y(b,4c,w/2,h/2)
        #::2 从0开始每两个取一个数，1::2 从1开始每两个取一个数。第一维是c ，第二维是 w，第三维是 h
        #x[..., ::2, ::2] 长宽从0开始相距2的位置也就是上图中的黄色0，x[..., 1::2, ::2] 宽从1 长从0 开始相距2 的位置也就是上图中的绿色1
        #因此x[..., ::2, 1::2]为红色 2，x[..., 1::2, 1::2]为蓝色3，
        #最后cat 将这四个部分拼接到第一维 c，所以通道数c 扩大四倍，w 和h 各缩小两倍
        return self.conv(torch.cat([x[..., ::2, ::2], x[..., 1::2, ::2], x[..., ::2, 1::2], x[..., 1::2, 1::2]], 1))
```

（2）CSP 结构（Cross Stage Partial）

CSP 结构是从网络结构设计的角度，来解决以往工作在推理过程中，需要很大计算量的问题。CSP 结构认为网络优化中的梯度信息重复导致推理计算繁复。CSP 结构通过将基础层的特征图划分为两个部分，然后通过 CSP 结构将它们合并，可以在能够实现更丰富的梯度组合的同时减少计算量。

YOLOv5 使用 CSPDarknet 作为 Backbone，从输入图像中提取丰富的信息特征。CSPNet 解决了其他大型卷积神经网络框架 Backbone 中网络优化的梯度信息重复问题，将梯度的变化从头到尾地集成到特征图中，因此减少了模型的参数量和 FLOPS 数值，既保证了推理速度和准确率，又减小了模型尺寸。

YOLOv5 采用了两种的 CSP 结构，第一种主要在 Backbone 中使用 CSP_1（结构图中明黄色）其中的 Bottleneck 就是采用 Res 结构，第二种就是在 Neck 中使用 CSP_2（结构图中蓝绿色）其中的 Bottleneck 没有采用 Res 结构。

CSP 结构代码：common.pyc - BottleneckCSP

```
class BottleneckCSP(nn.Module):
    # CSP Bottleneck https://github.com/WongKinYiu/CrossStagePartialNetworks
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super(BottleneckCSP, self).__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)    #Conv = Conv2d + BatchNorm2d + LeakyReLU
        self.cv2 = nn.Conv2d(c1, c_, 1, 1, bias=False)
        self.cv3 = nn.Conv2d(c_, c_, 1, 1, bias=False)
        self.cv4 = Conv(2 * c_, c2, 1, 1)
```

```
        self.bn = nn.BatchNorm2d(2 * c_)  # applied to cat(cv2, cv3)
        self.act = nn.LeakyReLU(0.1, inplace=True)
        self.m = nn.Sequential(*[Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)]) #n 个 Bottleneck

    def forward(self, x):
        #特征图分为两个部分
         #第一部分：先后经过 Conv（C+B+L） 和 n 个 Bottleneck 以及 Conv2d 得到 y 1
        y1 = self.cv3(self.m(self.cv1(x)))
        y2 = self.cv2(x)   #第二部分：经过卷积 conv 得到 y2
        #合并两个部分：cat y1 和 y2，然后进行 BatchNorm2d 标准化 和 LeakyReLU 激活 再经过 Conv
        return self.cv4(self.act(self.bn(torch.cat((y1, y2), dim=1))))
```

（3）LeakyReLU

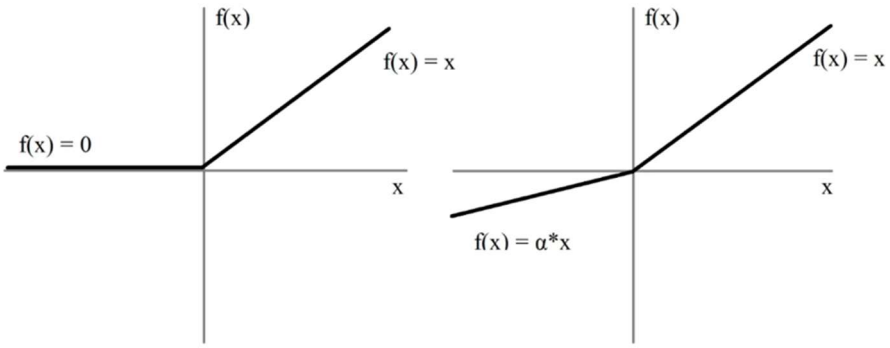


图 3 ReLU 和 LeakyReLU 函数图像

ReLU 是最常用的激活函数，**ReLU(x) = max(0, x)**，但 ReLU 会面临一个问题，在训练过程中部分神经元不会被激活，导致相应的参数永远不能被更新，为了解决这种问题，改进版本 LeakyReLU 提出了将 ReLU 的前半段设为 ax 而非 0。**LeakyReLUa(x)=max(ax, x)**，激活函数有一个参数 a，控制着 leaks 的数量，斜率非常的小，但是能够保证神经元能够激活。

pytorch 里相关的函数为：torch.nn.LeakyReLU(0.1,inplace=True)

（4）SPP 层：Spatial Pyramid Pooling（空间金字塔池化）

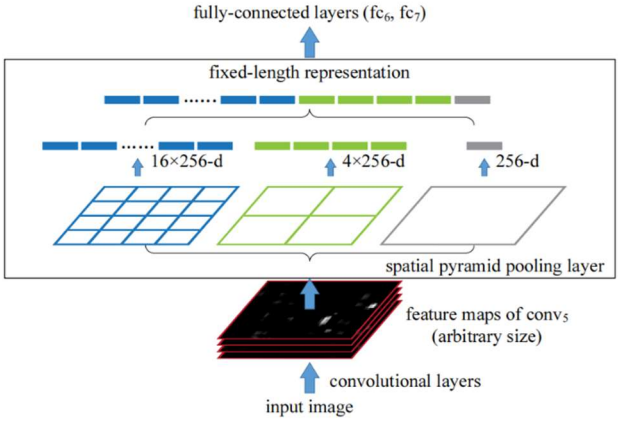


图 4 Spatial Pyramid Pooling 原理图

如上图，feature maps 是经过三个 pooling 窗口进行池化，将分别得到的结果在 channel 维度进行 concat。SPP 可以增大感受野，有助于解决 anchor 和 feature map 的对齐问题。引入 SPP 层将原有多个不同尺寸的图片可以统一输入到网络，规范下一层的输入。

SPP 的代码如下： common.py - class SPP

```
1.   class SPP(nn.Module):
2.       # Spatial pyramid pooling layer used in YOLOv3-SPP
3.       def __init__(self, c1, c2, k=(5, 9, 13)):
4.           super(SPP, self).__init__()
5.           c_ = c1 // 2  # hidden channels
6.           self.cv1 = Conv(c1, c_, 1, 1)
7.           self.cv2 = Conv(c_ * (len(k) + 1), c2, 1, 1)
8.           self.m = nn.ModuleList([nn.MaxPool2d(kernel_size=
   x, stride=1, padding=x // 2) for x in k])
9.
10.      def forward(self, x):
11.          x = self.cv1(x)
12.          return self.cv2(torch.cat([x] + [m(x) for m in se
   lf.m], 1))
```

2.2.3 Neck 特征融合网络模块

YOLOv5 的 Neck 部分采用了 PANet 结构，Neck 主要用于生成特征金字塔。特征金字塔会增强模型对于不同缩放尺度对象的检测，从而能够识别不同大小和

尺度的同一个物体。

PANet 结构是在 FPN 的基础上引入了 Bottom-up path augmentation 结构。FPN 主要是通过融合高低层特征提升目标检测的效果，尤其可以提高小尺寸目标的检测效果。Bottom-up path augmentation 结构可以充分利用网络浅特征进行分割，网络浅层特征信息对于目标检测非常重要，因为目标检测是像素级别的分类浅层特征多是边缘形状等特征。PANet 在 FPN 的基础上加了一个自底向上方向的增强，使得顶层 feature map 也可以享受到底层带来的丰富的位置信息，从而提升了大物体的检测效果。如下图所示，(a)FPN backbone, (b)Bottom-up path augmentation。

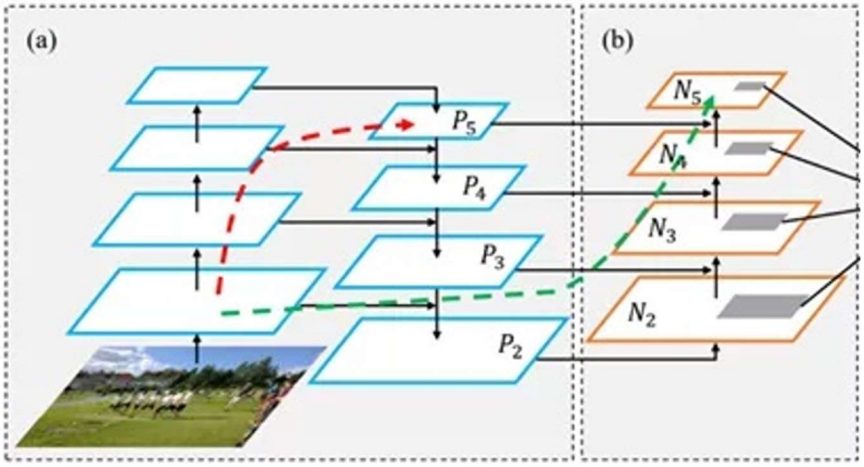


图 5 Bottom-up path augmentation 结构

2.2.4 Head 推理网络模块

Head 进行最终检测部分，在 yolov5.yaml 的相关配置如下：

```
[-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1]],  # 26 (P5/32-large)
```

YOLOv5 采用了与 YOLOv3 相同的 head 网络，都是 1*1 的卷积结构，并有三组 output，输出的特征图大小分辨为：

- bs * 255 * 80 * 80；
- bs * 255 * 40 * 40；
- bs * 255 * 20 * 20；

其中，bs 是 batch size，255 的计算方式为 [na * (nc + 1 + 4)]，具体参数含义如下：

- na(number of anchor) 为每组 anchor 的尺度数量（YOLOv5 中一共有 3 组 anchor，每组有 3 个尺度）；
- nc 为 number of class；
- 1 为前景背景的置信度 score；
- 4 为中心点坐标和宽高；

最后，输出的特征图上会应用锚定框，并生成带有类别概率、置信度得分和包围框的最终输出向量。与 YOLOv3 不同的是，在 anchor 上 YOLOv5 跨网格匹配规

则的方式来区分 anchor 的正负样本。

2.3 损失函数的定义

损失函数的作用为度量神经网络预测信息与期望信息（标签）的距离，预测信息越接近期望信息，损失函数值越小。训练时主要包含三个方面的损失：分类损失(classification loss)、置信度损失(confidence loss)、定位损失(location loss)。

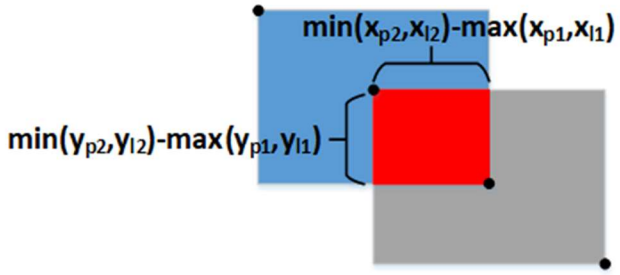
表 1 损失函数的组成

损失类型	计算方法	计算对象
confidence loss 置信度损失（是否存在目标）	BCE loss（二值交叉熵损失）	所有样本的损失（预测边界框与 GT Box 的误差）
classification loss 分类损失（分类是否准确）	BCE loss（二值交叉熵损失）	只计算正样本的分类损失
location loss 定位损失（是否定位准确）	CIoU loss	只计算正样本的定位损失（预测边界框与 GT Box 的误差）

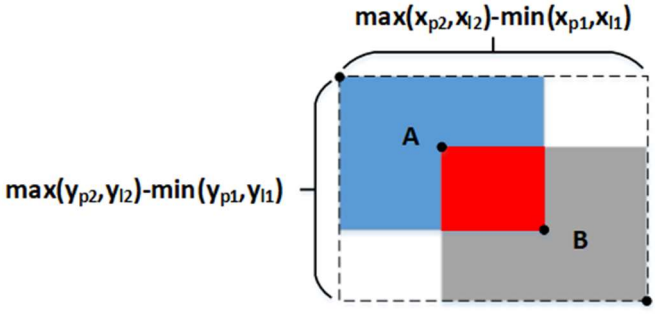
2.3.1 定位损失计算

比较各个 IoU 系列损失函数的不同点：

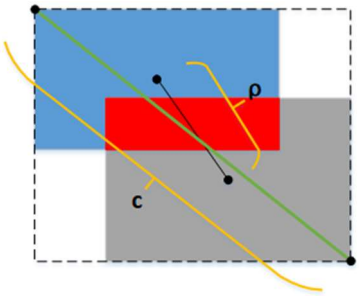
1.IOU_Loss：主要考虑检测框和目标框重叠面积。



2.GIOU_Loss：在 IOU 的基础上，加上一个可以框住 A、B 的大矩形框，解决边界框不重合时的问题。



3.DIOU_Loss：在 IOU 和 GIOU 的基础上，考虑边界框中心点距离的信息。



定义为：

$$DIOU = IOU - \frac{\rho^2}{c^2}$$
$$\rho^2 = (x_p - x_l)^2 + (y_p - y_l)^2 \quad (1)$$
$$c^2 = \left(\max(x_{p2}, x_{l2}) - \min(x_{p1}, x_{l1})\right)^2 + \left(\max(y_{p2}, y_{l2}) - \min(y_{p1}, y_{l1})\right)^2$$

4.CIOU_Loss：在 DIOU 的基础上，考虑边界框宽高比的尺度信息。
定义为：

$$CIOU = IOU - \frac{\rho^2}{c^2} - \alpha v = DIOU - \alpha v$$
$$v = \frac{4}{\pi^2}\left(\arctan\frac{w_l}{h_l} - \arctan\frac{w_p}{h_p}\right)^2 = \frac{4}{\pi^2}\left(\arctan\frac{x_{l2} - x_{l1}}{y_{l2} - y_{l1}} - \arctan\frac{x_{p2} - x_{p1}}{y_{p2} - y_{p1}}\right)^2 \quad (2)$$
$$\alpha = \frac{v}{1 - IOU + v}$$

这样 CIOU_Loss 就将目标框回归函数应该考虑三个重要几何因素：重叠面积、中心点距离，长宽比全都考虑进去了。

$$loss_{CIOU} = 1 - CIOU \quad (3)$$

2.3.2 置信度损失计算

对于一张图像分割成的 80*80 的网格，神经网络对其中每个格子都预测三个位于该格子附近的矩形框（简称预测框），每个预测框的预测信息包括中心坐标、宽、高、置信度、分类概率，因此神经网络总共输出 3*80*80 个 0~1 的预测置信度，与 3*80*80 个预测框一一对应。预测框的置信度越大表示该预测框越可信，也即越接近目标的真实最小包围框。

标签的维度应该与神经网络的输出维度保持一致，因此置信度的标签也是维度为 3*80*80 的矩阵。利用 mask 掩码矩阵：以维度同样为 3*80*80 的 mask 矩阵为标记，对置信度标签矩阵进行赋值。对 mask 为 true 的位置不直接赋 1，而

是计算对应预测框与目标框的 CIOU，使用 CIOU 作为该预测框的置信度标签，对 mask 为 false 的位置直接赋 0。所以，标签值的大小与预测框、目标框的重合度有关，两框重合度越高则标签值越大。不过，CIOU 的取值范围是-1.5~1，而置信度标签的取值范围是 0~1，所以需要对 CIOU 做一个截断处理：当 CIOU 小于 0 时直接取 0 值作为标签。

BCE loss 损失函数

假设置信度标签为矩阵 L，预测置信度为矩阵 P，那么矩阵中每个数值的 BCE loss 的计算公式如下：

$$\begin{aligned} &loss_{BCE}(z,x,y) = -L(z,x,y)*\log P(z,x,y) - (1-L(z,x,y))*\log(1-P(z,x,y)) \\ &0 \le z < 3 \\ &0 \le x < 80 \\ &0 \le y < 80 \end{aligned} \tag{4}$$

BCE loss 要求输入数据的取值范围必须在 0~1 之间。

从而得到 80*80 网格的置信度损失值：

$$\begin{cases} l_{obj} = \dfrac{1}{num(mask = true)} \sum\limits_{\substack{mask=true \\ 0\le z<3 \\ 0\le x<80 \\ 0\le y<80}} loss_{BCE}(z,x,y) \\ l_{noobj} = \dfrac{1}{num(mask = false)} \sum\limits_{\substack{mask=false \\ 0\le z<3 \\ 0\le x<80 \\ 0\le y<80}} loss_{BCE}(z,x,y) \\ loss_{obj80} = a*l_{obj} + (1-a)*l_{noobj} \end{cases} \tag{5}$$

2.3.3 分类损失计算

神经网络对 80*80 网格的每个格子预测三个预测框，每个预测框的预测信息都包含了 N 个分类概率。其中 N 为总类别数，比如 COCO 数据集有 80 个类别，那么 N 取 80，本项目 N=2。所以对于数据集，每个预测框有 N 个 0~1 的分类概率，那么网络总共预测 3*80*80*N 个分类概率，组成预测概率矩阵。

80*80 网格的标签概率矩阵与预测概率矩阵的维度一样，也是 3*80*80*N。每个预测框的标签，由解析 json 标签文件得到，是一个 0~N-1 的数值，需要将

0~N-1 的数值转换成 N 个数的独热码。为了减少过拟合，且增加训练的稳定性，通常对独热码标签做一个平滑操作。同样假设置标签概率为矩阵 Lsmooth，预测概率为矩阵 P，那么矩阵中每个数值的 BCE loss 的计算公式类似。

3 方法实现

3.1 训练过程

3.1.1 数据集

本文数据集来自互联网、COCO 数据集和 WIDER FACE，从中提取有关人脸和口罩的图片。按照 4:1 的比例划分数据集和验证集，经过筛选出 1600 个训练集，400 个验证集。

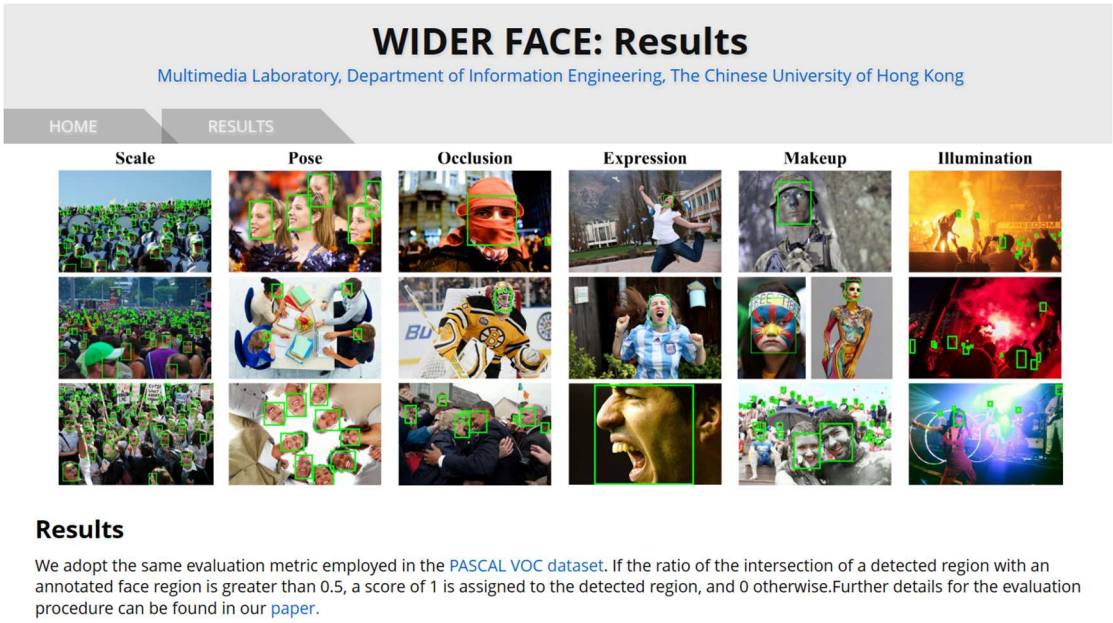
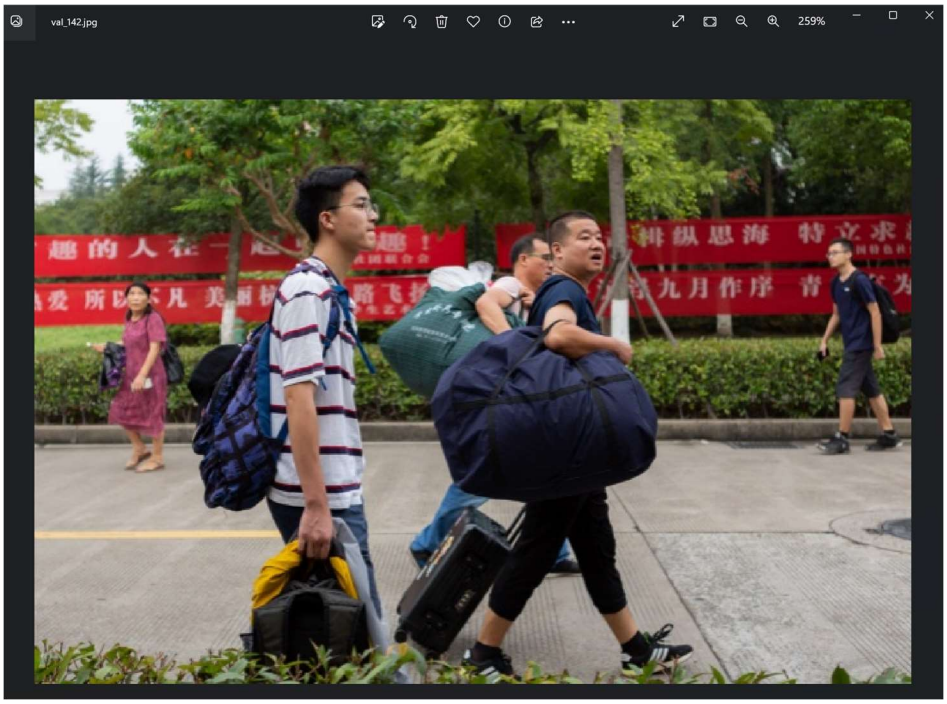


图 5 WIDER FACE 人脸数据集网站

使用 labelimg 软件进行数据标注：在收集好的图片文件夹的同级目录下，新建一个 labels 文件夹用于存放打标的记录，每一张图片与一个记录了目标框信息的文本文件一一对应。

图 6 labelimg 标注图片

文本文件中每一行表示一个目标，以空格进行区分，分别表示目标的类别信息（0 代表 mask，1 代表 face），和归一化处理之后的中心点 x 坐标、y 坐标、目标框的 width 和 height。



val_142 - 记事本

文件 编辑 查看

1 0.6320312594179995 0.2640845108544454 0.04218750062864274 0.09154929709620774
1 0.3718750055413693 0.21009389974642545 0.0437500006519258 0.11032864009030163
1 0.1164062517345883 0.3462441364536062 0.029687500442378223 0.049295775359496474
1 0.9148437636322342 0.2723004734143615 0.023437500349245965 0.042253521736711264
1 0.5781250086147338 0.2816901449114084 0.02500000037252903 0.06572770047932863

图 7 图片与对应的标注信息

典型的数据增强算法有如下几种：

表 2 数据增强

类型名称	作用效果
Mosaic	多张图片合成一张，增加数据的多样性
Copy paste	实例分割，将每个目标复制粘贴多次
Random affine	随机缩放平移
MixUp	透明度重叠
Augment HSV	色度 饱和度 明度
Random horizontal flip	随机水平翻转

为了对模型具有良好的泛化性和有效性，本算法使用 Mosaic 数据增强：

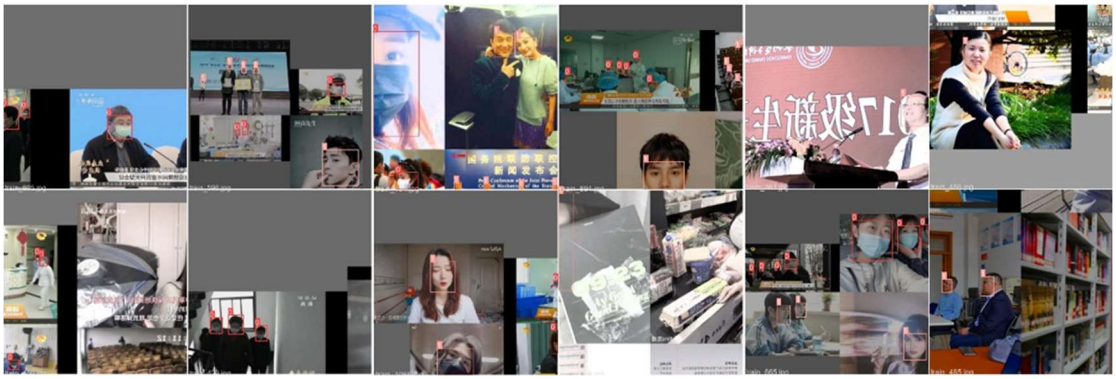


图 8 mosaic 数据增强

3.1.2 模型训练

本算法采用的平台为 PyTorch1.13(1.13.1+cu117)深度学习框架，Cuda（及 cudnn）版本为 cuda11.7，Python 版本为 3.8。使用笔记本电脑操作系统为 Windows11，Intel(R) Core(TM) i9-12900H CPU@ 2.50 GHz，Ge-Force GTX 3060 进行模型训练和验证集的测试。

```
(yolo5) C:\Users\19328\Desktop\ZJW\machine vision\yolov5-mask-42-master>python
Python 3.8.15 | packaged by conda-forge | (default, Nov 22 2022, 08:42:03) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.13.1+cu117'
>>> torch.cuda.is_available()
True
>>>
```

图 9 cuda、pytorch 版本

需要修改：1、数据集配置文件中的数据集地址（mask_data.yaml）2、模型配置文件中类别数（mask_yolov5s.yaml）

执行命令用到的其他参数为：3、预训练模型的权重（pretrained/yolov5s.pt）4、训练轮数（epoch 200）5、一次训练所选取的样本数（batch-size 8）

执行 python train.py --data mask_data.yaml --cfg models/mask_yolov5s.yaml

--weights pretrained/yolov5s.pt --epoch 200 --batch-size 8 --device 0

3.2 测试数据及结果展示

3.2.1 评价指标

表 3 混淆矩阵名词解释

	True 正确　正类	False 错误　负类
Positive 被检测到	TP（正确的检测了）	FP（错误的检测了）
Negative 未被检测到	TN（正确的没检测）	FN（错误的未检测，本应该检测到的）

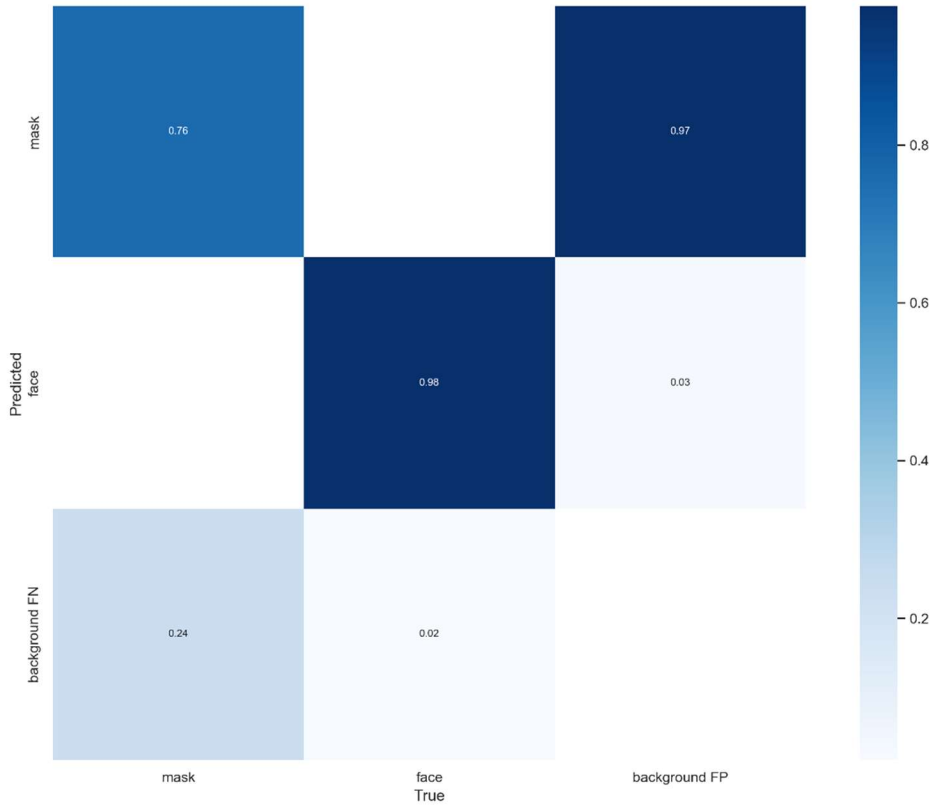


图 11 混淆矩阵

表 4 评价指标

P	Precision，精确率=正确数 TP / 预测总数(TP+FP)
R	Recall，召回率=预测正确数 TP /某类真实总数(TP+FN)
mAP	mean Average Precision=每个类的 AP 值的平均数 AP=$\int_0^1 p(r)dr$

mAP@.5	当 IoU 为 0.5 时的 mAP
mAP@.5 : .95	当 IoU 为 range(0.5 : 0.95)时的 mAP 的平均数

PR 图：横坐标是 R 值，纵坐标是 P 值，曲线表示当召回率为 R 时，精确率 P 的大小。P 值是随着 R 的升高而降低， PR 图左下方的面积越大，则表示模型对该数据集的效果越好。

3.2.2 结果分析

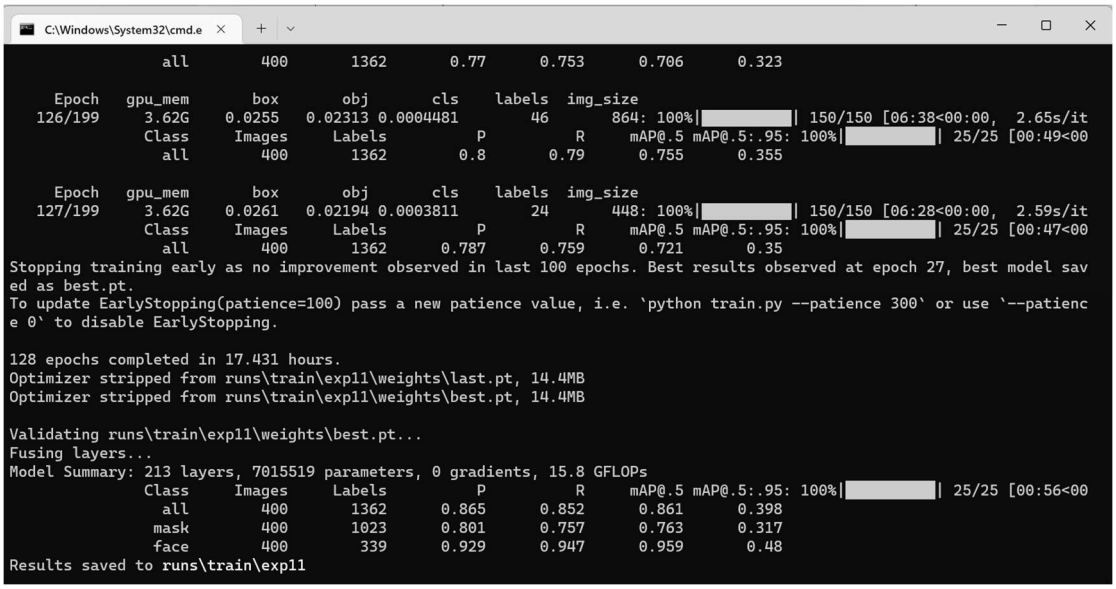


图 12 训练结果

由图可以看出，验证集一共有 400 张图片，1362 个标签（口罩 1023 个，人脸 339 个）。总体的精度为 0.865，召回率为 0.852，mAP 值达到了 0.861，但人脸和口罩的 mAP 值分别是 0.959 和 0.763，可见模型总体的训练效果较好，对人脸识别的训练效果优于口罩。

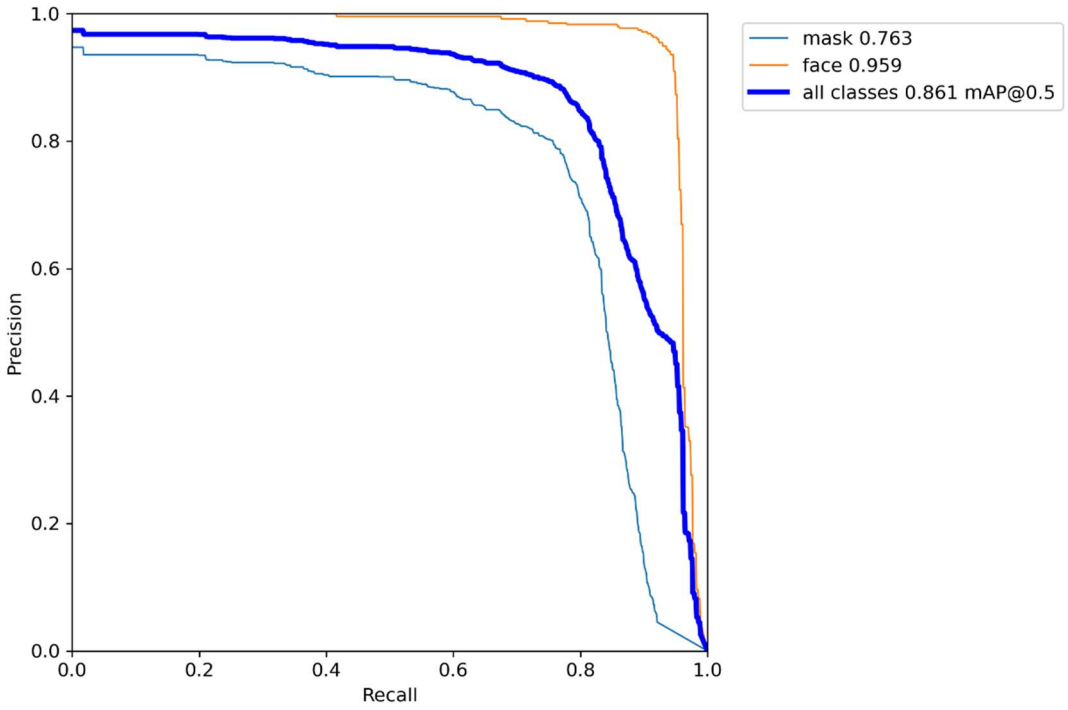


图 13 PR 图

由 PR 图可以看出，对比 mask 和 face 的 PR 曲线所包含的面积，也可以得出相似结论。

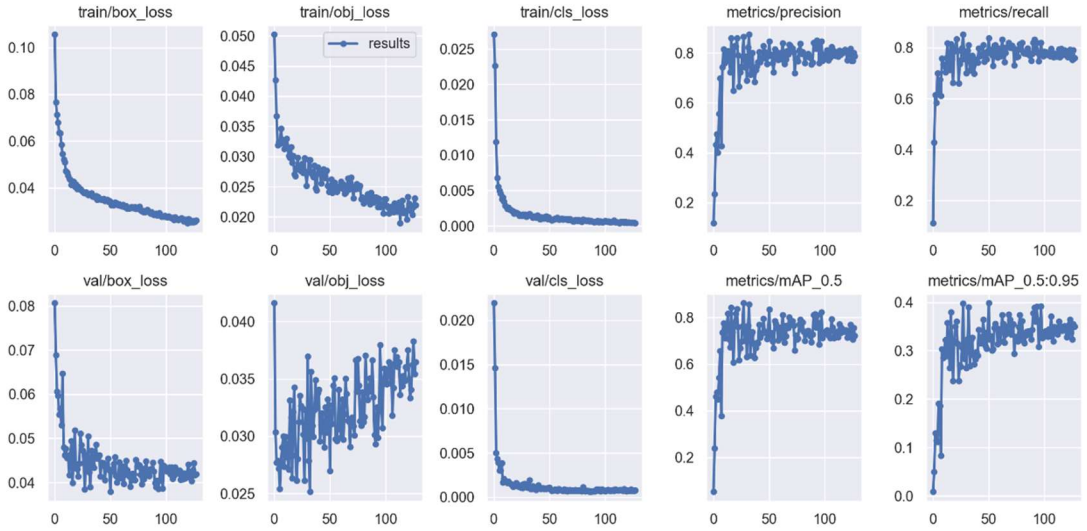


图 14 结果统计

由关于训练轮数的函数图可以看出，训练在 30 轮左右就已经达到较为理想的值，训练速度较快，证明参数设置较为合理。

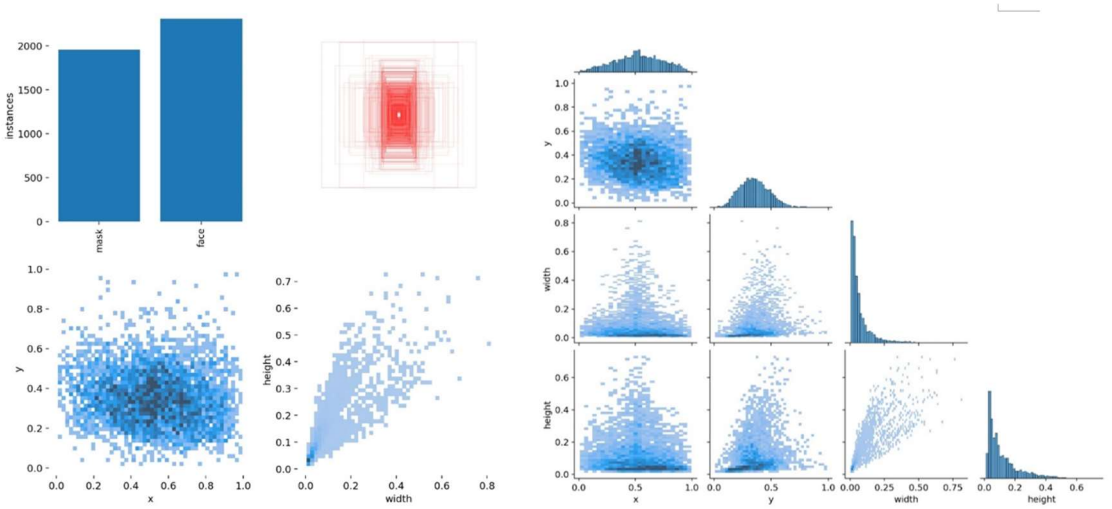


图 15 数据的分类、位置、大小数据分布统计

图 16 测试集结果

图 17 图形化界面测试结果

用自己拍摄的照片测试结果较好。

3.2.3 参数调试

YOLOv5 的参数皆可在指定的 YAML 文件下配置，也可以在运行时附加参数。分为以下几种

1.网络配置文件

（1）模型的深度和宽度

（2）Anchor（预设了三种模式）

（3）Backbone
（4）Head
2.初始化超参数
（1）hpy 超参数（包括 lr、weight_decay、momentum 和图像处理的参数等）
（2）训练超参数（包括预训练，batch-size，epoch 等）
由于时间与精力限制，主要进行了以下三个参数的调试：

- *conf-thres*：置信度，default=0.25
- *NMS*：非极大值抑制，default=false
- *epochs*：训练的轮数，default=200

根据控制变量的原则，并严格使用测试集而非训练集进行测试。
调试发现：①置信度越小，则不符合目标特征的对象越可能会被框选。置信度过大，符合目标特征的对象可能不会框选，经测试 0.25 为较优值。②NMS 的值并不会对检测产生明显的变化，推测这是由于 Anchor 参数已经被设置为小目标检测模式导致的。③训练轮数在 127 时已经和最后的最优结果（epoch=27）相差无几，继续增加训练轮数没有太大意义。

项目已上传 GitHub，地址：https://github.com/reasimei/yolov5-mask-detection

参考文献

[1] REDMON J, DIVVALA S,GIRSHICK R, et al. You only look once: unified, real-time object detection [EB/OL].[2020- 03-10].https:/arxiv.org/pdf/1506.02640v5.
[2] REDMON J, FARHADI A. YOLO9000: better, faster, stronger[C]//IEEE Conference on Computer Vision & Pattern Recognition, 2017:6517-6525
[3]刘翀豪,潘理虎,杨帆,张睿.改进 YOLOv5 的轻量化口罩检测算法[J/OL].计算机工程与应用:1-11[2023-01-19].http://kns.cnki.net/kcms/detail/11.2127.TP.20221230.1300.011.html
[4]杨锦辉,李鸿,杜芸彦,毛耀,刘琼.基于改进 YOLOv5s 的轻量化目标检测算法[J/OL].电光与控制:1-11[2023-01-19].http://kns.cnki.net/kcms/detail/41.1227.tn.20221229.1556.002.html