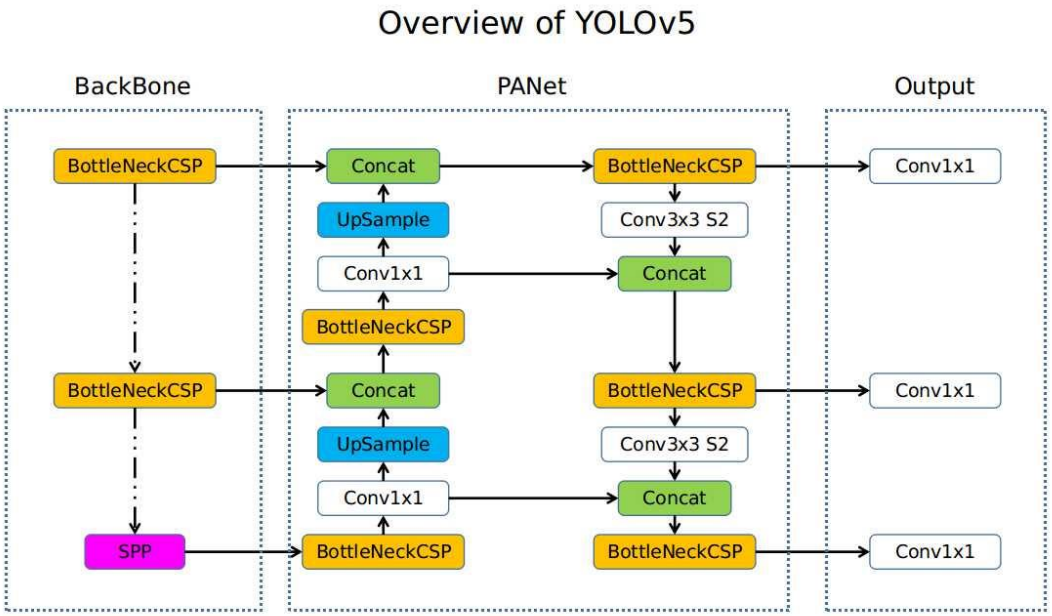
1 绪论

目标检测是深度学习的一个主题，也是计算机视觉领域重要和具有挑战性的分支之一。它在人们生活中得到了广泛的应用，如监控安全、自动驾驶等。目标检测的任务是定位某一类语义对象的实例。为了完成本次比赛识别桌面物体的任务，我们采用了基于 pytorch 包的开源框架 yolov5，它具有简单灵活，可扩展能力高的优势。首先, 我们进行数据集的准备，将训练的数据集分成训练集和测试集，并将标注转换为和 darknet format 相同的标注形式。之后，我们修改 yolov5 的配置文件以注册数据集，并选用不同的权重文件进行对比训练。然后，我们根据不同的模型进行准确度的评估，并且对网络参数进行一些优化，分析了一些代表性的参数。最后，我们利用这一模型生成了指定格式的 txt 文件。

2　具体算法

Yolov5 的整体结构图如下所示



相较于 Yolov3、Yolov4, Yolov5 具有如下改进和优势：

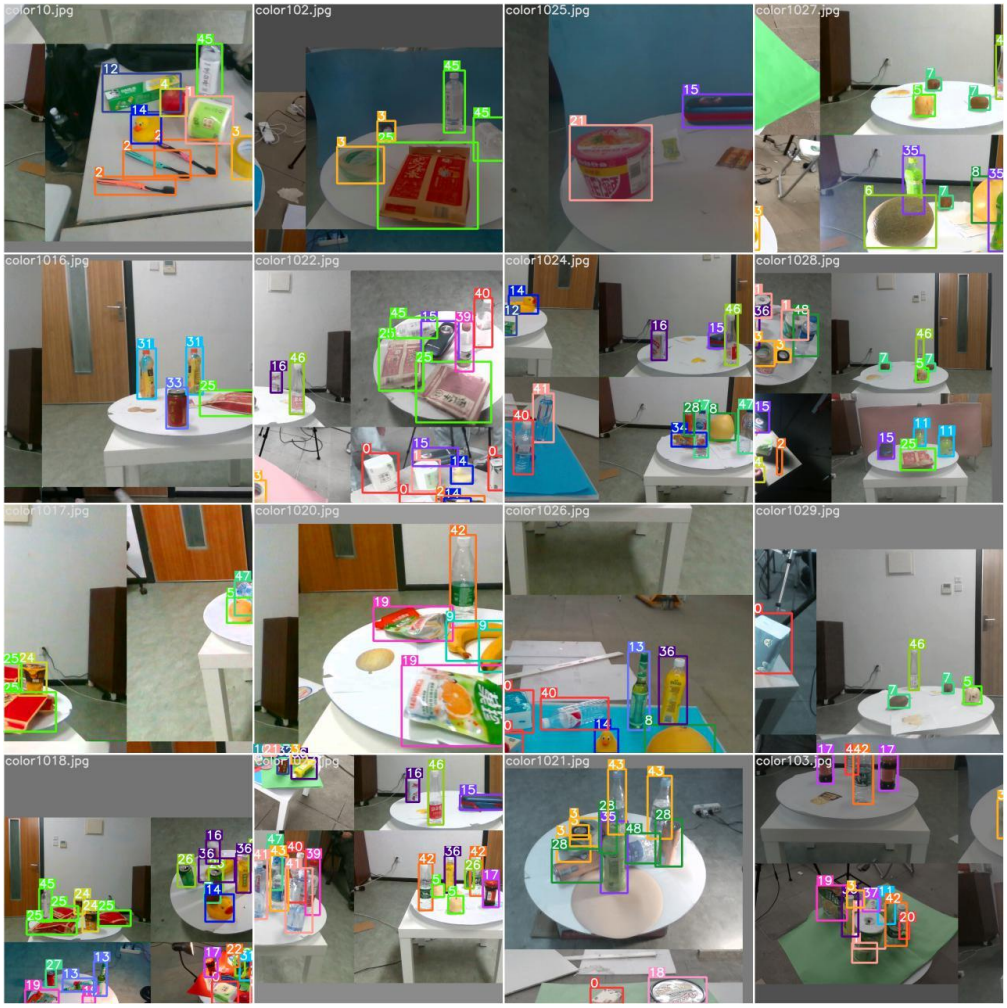
1 自适应锚框计算

在 Yolov3、Yolov4 中，训练不同的数据集时，计算初始锚框的值是通过单独的程序运行的。但 Yolov5 中将此功能嵌入到代码中， 通过 yaml 配置文件进行组织，更加简洁高效。

```
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
```

2 数据增强

Yolov5 通过 Mosaic 数据增强的方式，将 4 张图片作为一个集合进行旋转、裁剪等操作，增强了网络的鲁棒性和适应性。同时采取了自适应图片缩放的算法，将原始图片统一缩放到一个标准尺寸，并将其它部分填充为灰色。使得模型可以接受不同尺寸的图片。



3 实验与分析

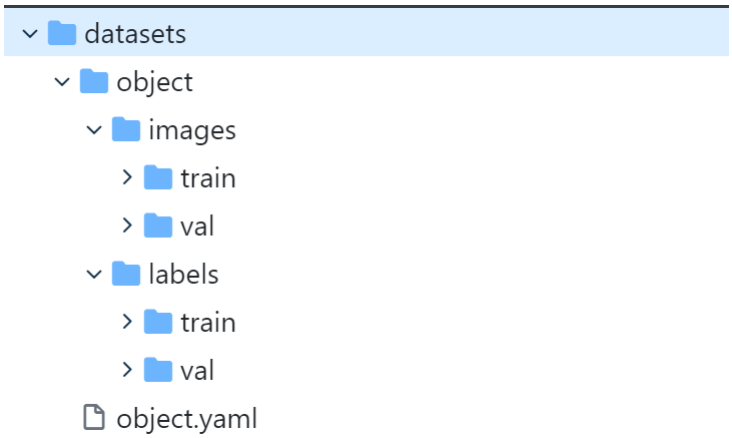
3.1 数据集的划分和加载

将原始数据集按照 9 : 1 的比例分为训练集和测试集

```python
def moveFile(img_Dir, label_Dir):
        img_path = os.listdir(img_Dir)
        # print(len(img_path))
        filenumber = len(img_path)
        rate = 0.1
        picknumber = int(filenumber * rate)
        sample = random.sample(img_path, picknumber)
        # print(sample)
        for val_name in sample:
            shutil.move(img_Dir + val_name, val_img_tarDir + val_name)
            shutil.move(label_Dir + val_name[:-4] + '.txt', val_label_tarDir + val_name[:-4] + '.txt')
        for train_name in img_path:
            if train_name not in sample:
                shutil.move(img_Dir + train_name, train_img_tarDir + train_name)
                shutil.move(label_Dir + train_name[:-4] + '.txt', train_label_tarDir + train_name[:-4] + '.txt')
        # print(img_path)
if __name__ == '__main__':
    img_Dir = "images/train/"    #源图片文件夹路径
    label_Dir = "labels/train/"

    train_img_tarDir = 'datasets/object/images/train/'    # 移动到新的文件夹路径
    val_img_tarDir = 'datasets/object/images/val/'
    train_label_tarDir = 'datasets/object/labels/train/'
    val_label_tarDir = 'datasets/object/labels/val/'
    moveFile(img_Dir, label_Dir)
```

数据集的组织格式如下：



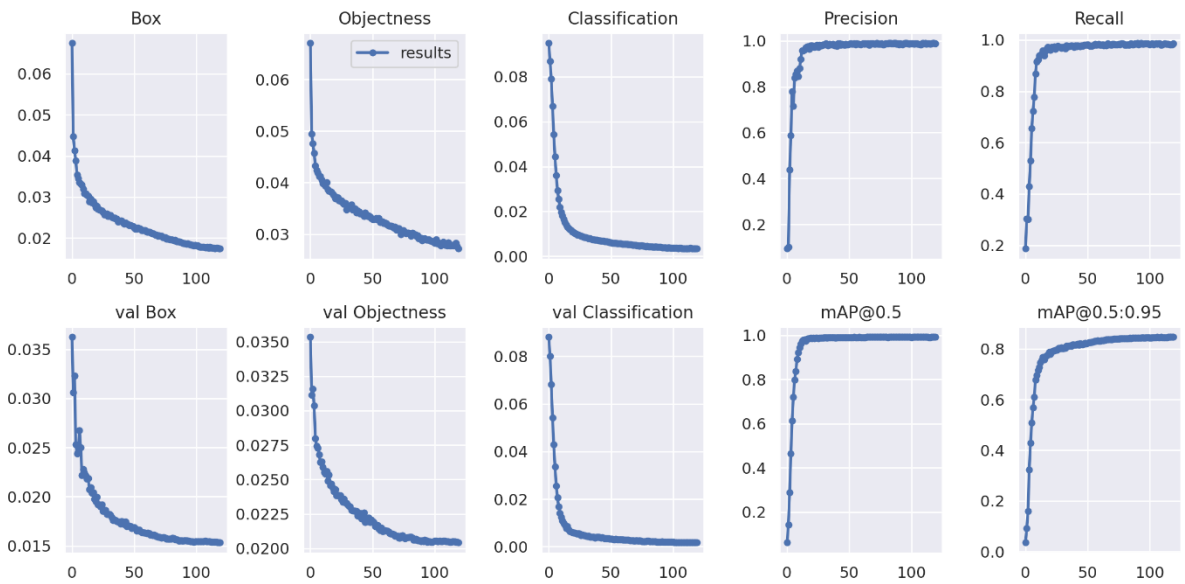
其中 object.yaml 为数据集的配置文件，包含物体类别数、backbone、head 等基础参数

3.2 训练模型

在 Ubuntu 18.04 OS 上训练我们的模型

```
$ nohup python train.py --img-size 640 --batch-size 16 --epochs 100 --data ../DLUT-
detection/datasets/object.yaml --cfg ./models/score/yolov5x.yaml --
weights ../yolov5x.pt --device 0,1,2,3,4,5,6,7 &
```

训练结果如下



可见，在训练 100 epochs 后模型很快收敛，且具有接近与 1 的 Precision 和 Recall 率，在误差允许的范围内，能够对物体做出比较精确的识别和定位。

3.3 评估模型

获得初步训练结果之后，我们又尝试了不同的权重文件 yolov5s.yaml、yolov5m.yaml、yolov5l.yaml、并且编写了文件 compare.py 用于比较不同的模型对于相同图像的准确度。经过分析，我们发现由于 yolov5x 网络整体结构最大、深度最广，对于本例的图像识别具有最好的效果。但是当训练 epoch > 150 后，训练效果较差，且网络会出现过拟合的现象。故综上，我们选用了 yolov5x 权重文件迭代 100 次的模型。

在测试数据集的时候，我们发现一些没有出现在桌子上的物体也被模型框选。且这些物体上显示了较小的概率。因此我们修改了 loss.py 文件，将置信区间设为 0.8，得到了较好的效果。

```
            # tobj[b, a, gj, gi] = (1.0 - self.gr) + self.gr * score_iou  # iou ratio
            tobj[b, a, gj, gi] = 0.8
```

3.4 测试模型

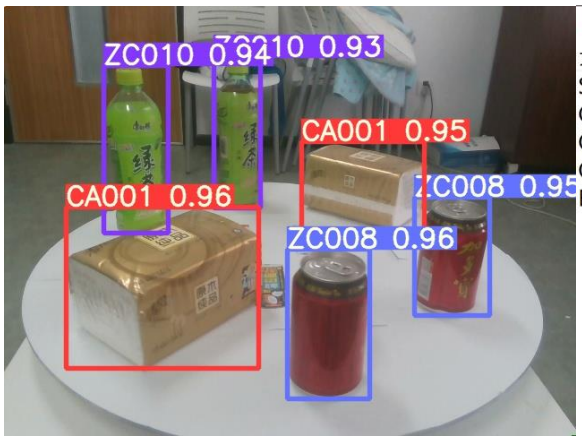
为了输出比赛要求格式的 txt 文件，我们修改了 dectect.py 文件，并将其另存为 my_dectect.py

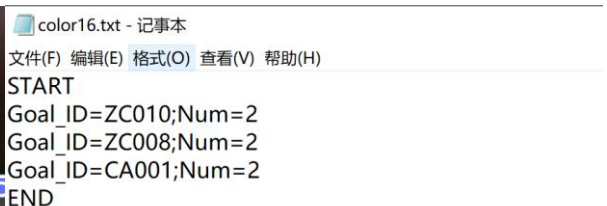
```
138                 category = {}
139                 with open(txt_path + '.txt', 'a') as f:
140                 # Write results
141                     f.write('START\n')
142                     for *xyxy, conf, cls in reversed(det):
143                         if save_txt:  # Write to file
144                             cls = cls.item()
145                             if cls in category.keys():
146                                 category[cls] += 1
147                             else:
148                                 category[cls] = 1
149                     for key, value in category.items():
150                         f.write(f'Goal_ID={names[int(key)]};Num={str(value)}\n')
151                     f.write('END\n')
```

在 Windows Powershell 上运行命令

```
$ python my_detect.py --source ../DLUT-detection/val --weights x.pt --save-txt
```

我们得到了模型输出的 txt 文件





由此可见，模型具有较高的准确度。