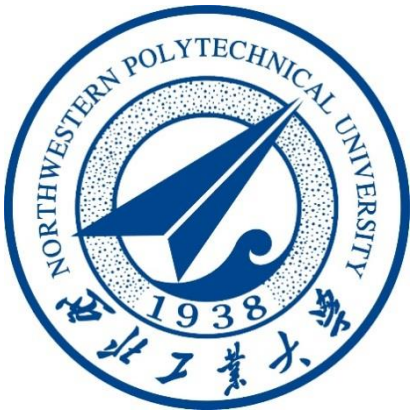


# 机器视觉：理论和基于 PyTorch 的实践课程报告

## YOLOv5 在 Robocup 机器人足球赛中的应用

姓　　名：陈鹏宇

班　　级：09011805

学　　号：2018301898

日　　期：2020.11.12

**已在课堂上报告**

# 目　录

# 一、 动机和贡献

## 1、项目介绍

RoboCup 机器人足球世界杯赛及学术大会（ The Robot World Cup Soccer Games and Conferences ）是国际上级别最高、规模最大、影响最广泛的机器人足球赛事和学术会议，每年举办一次。 机器人足球赛是由硬件或仿真机器人进行的足球赛，比赛规则与人类正规的足球赛类似。硬件机器人足球队的研制涉及计算机、自动控制、传感与感知融合、无线通讯、精密机械和仿生材料等众多学科的前沿研究与综合集成。

在足球赛中，很容易出现的一种情况就是点球。根据规则，对在自己的禁区内和比赛中犯下直接犯规的六次（代替：十次）犯规的球队判罚一次罚球。参赛队可以直接从点球得分。很多情况下点球直接左右了比赛的胜负。因此，如何更好的完成点球项目是一个比较重要的问题。

点球项目最简单的策略——看见球，然后机器人出脚踢。所以最基本的一点：如何看到球并识别出球？

**图 1：往届点球比赛图片**

在以往的比赛中，我们使用的是 Robotis OP2 机器人。其上位机性能相对较低，因此考虑到其性能，我们一向使用的是传统的目标识别方法，如阈值提取，图像分割的方法，进行目标识别。

我们的另一个移动踢球（Moving Kick）比赛，选择的是帧差法。即比较一个视频流相邻的两帧的像素变化，可以认为像素变化大的区域检测到了球。因为球移动较快，

视频帧差有比较好的识别效果。

但是今年我们换成了性能更加强大的 op3 机器人：

**图 2：op3 机器人**

ROBOTIS-OP3 规格:

| | |
|---|---|
| 高度 | 约 510mm |
| 重量 | 约 3.5kg（无皮肤覆盖） |
| 执行器 | XM430-W350-R |
| 主控制器（上位机） | 英特尔 NUC i3　Intel Core i3 双核处理器<br>8GB RAM 2133MHz 128GB M.2 SSD |
| 副控制器（下位机） | OpenCR |
| 相机 | 罗技 C920 HD Pro 网络摄像头 |
| IMU 传感器 | 3 轴陀螺仪，3 轴加速度计，3 轴磁力计 |
| 开发环境 | 操作系统：Linux（64-bit）<br>C ++，ROS，DYNAMIXEL SDK |

OP3 的上位机配备了 Intel Core i3 双核处理器，处理器性能更加强大。对于点球项目，因为球处于静止状态，同时由于 OP3 机器人性能上的支持。我们可以在 OP3 上部署深度神经网络，实现足球的跟踪识别，提高准确度。其中比较理想的是部署 yolo 框架。相较 op2，op3 机器人上位机更强大，为部署深度神经网络提供了可能。

一开始是部署了 yolov3 模型，效果还可以。之后我在机器人上搭建了 python 环境，部署了 YOLOv5 的框架，进行了进一步的测试。

### 2、主要贡献

因为我主要负责的是识别球的环节，所以主要贡献包括 OP3 机器人上位机 python 环境的搭建，YOLOv5 框架的部署。进行图片数据集的采集，在自己的笔记本上进行 YOLOv5 模型的训练，在 op3 机器人上运行目标跟踪代码，并返回目标球的 box 在 2D 图像上的坐标。


**图 3：期望目标检测效果图**

## 二、 所用的方法

### 1、框架选择

我采用的是当今主流的 YOLOv5 框架。由于 YOLOv5 是在 PyTorch 中实现的，它受益于成熟的 PyTorch 生态系统，支持更简单，部署更容易。根据 YOLOv5 官方给出的信息，相对于 YOLOv4，YOLOv5 具有以下优点：

1）速度更快。在 YOLOv5 上，运行 TeslaP100，我们看到每张图像的推理时间仅需 0.007 秒，这意味着每秒 140 帧（FPS），速度是 YOLOv4 的 2 倍还多。

2）精度更高。在 Roboflow 对血细胞计数和检测（BCCD）数据集的测试中，只训练了 100 个 epochs 就达到了大约 0.895 的平均精度（mAP）。在准确率没有任何损失的情况下，看到如此全面的性能提升是非常罕见的。

3）体积更小。YOLOv5 的权重文件是 27 兆字节。YOLOv4（采用 Darknet 架构）的权重文件是 244 兆。YOLOv5 比 YOLOv4 小了近 90%。这意味着 YOLOv5 可以更容易地部署到嵌入式设备上。这也符合我们机器人的嵌入式设备的需求。

## 2、网络结构

YOLOv5 一共有四种网络模型。

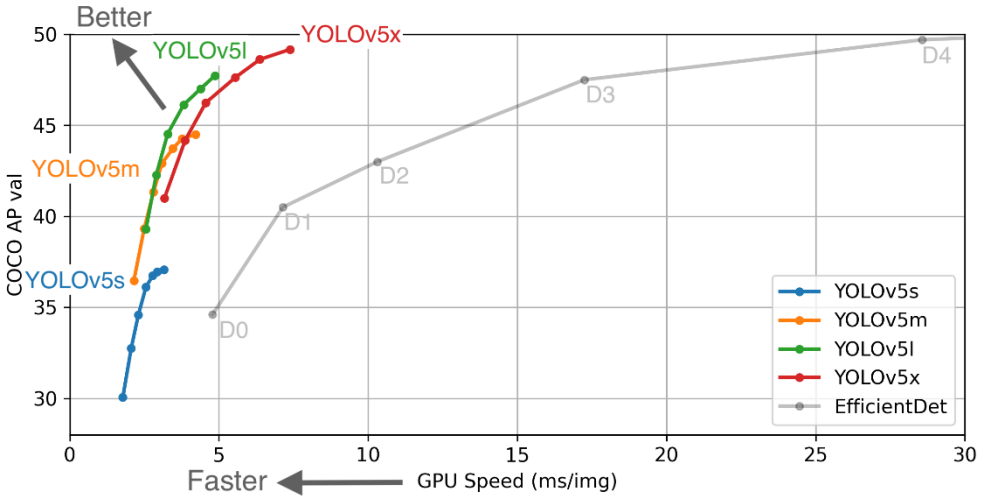


**图 4：YOLOv5 四种网络性能比较图**

我选择的是 YOLOv5m 网络进行训练，比起 5s 训练精度高，同时比 5l 和 5x 速度要快。考虑到识别的环境并不复杂，用 YOLOv5m 应该足够获得较好的精度。

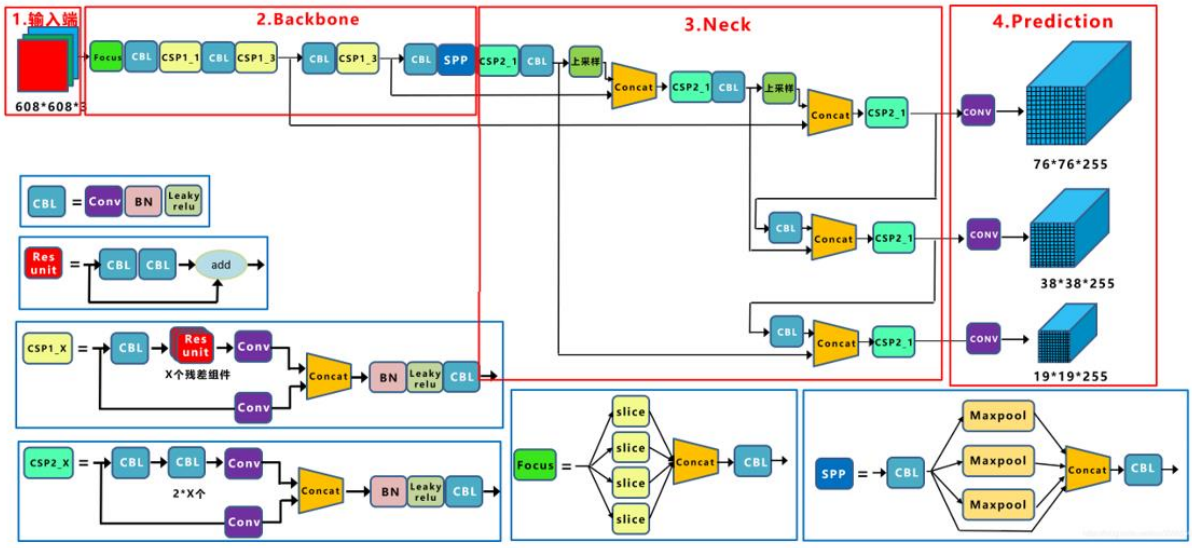


**图 5：YOLOv5m 网络结构图**

下面对于 YOLOV5m 的各部分结构进行解释说明。

**1）输入端：**

示例中给出的是输入端图片的常见尺寸，为 608*608*3。所以需要进行自适应图片缩放。输入端采用了 Mosaic 数据增强，具体为采用了 4 张图片，用随机缩放、随机裁剪、随机排布的方式进行拼接。

小目标的 AP 一般比中目标和大目标低很多，但在整体的数据集中，小、中、大目标的占比并不均衡。

Mosaic 数据增强可以为我们带来以下好处：

1、丰富数据集：随机使用 4 张图片，随机缩放，再随机分布进行拼接，大大丰富了检测数据集，特别是随机缩放增加了很多小目标，让网络的鲁棒性更好。

2、减少 GPU：可能会有人说，随机缩放，普通的数据增强也可以做，但作者考虑到很多人可能只有一个 GPU，因此 Mosaic 增强训练时，可以直接计算 4 张图片的数据，使得 Mini-batch 大小并不需要很大，一个 GPU 就可以达到比较好的效果。

同时，输入端还有自适应锚框计算。在 Yolo 算法中，针对不同的数据集，都会有初始设定长宽的锚框。在网络训练中，网络在初始锚框的基础上输出预测框，进而和真实框 ground truth 进行比对，计算两者差距，再反向更新，迭代网络参数。

**图 6：Mosaic 数据增强**

**2）Backbone：**在不同图像细粒度上聚合并形成图像特征的卷积神经网络

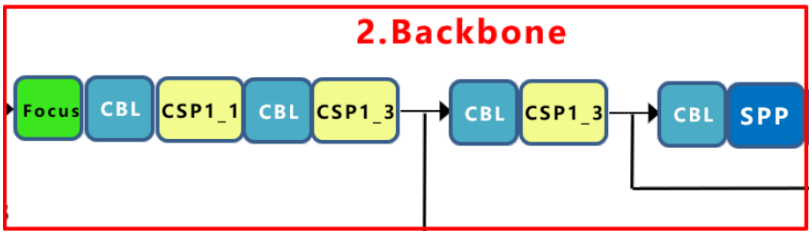


**图 7：Backbone 网络结构**

CBL：Yolov3 网络结构中的最小组件，由 Conv（卷积层）+ BN + Leaky_relu 激活函数三者组成。其中 BN 层本质上是一个归一化网络层，可以加快训练速度，提高网络的泛化能力。


**图 8：CBL 网络结构**

Res unit：借鉴 Resnet 网络中的残差结构，让网络可以构建的更深。


**图 9：Res unit 网络结构**

CSP：跨阶段局部网络，以缓解以前需要大量推理计算的问题。过将梯度的变化从头到尾地集成到特征图中。增强了 CNN 的学习能力，能够在轻量化的同时保持准确性。Yolov5 中设计了两种 CSP 结构， CSP1_X 结构应用于 Backbone 主干网络，另一种 CSP2_X 结构则应用于 Neck 中。

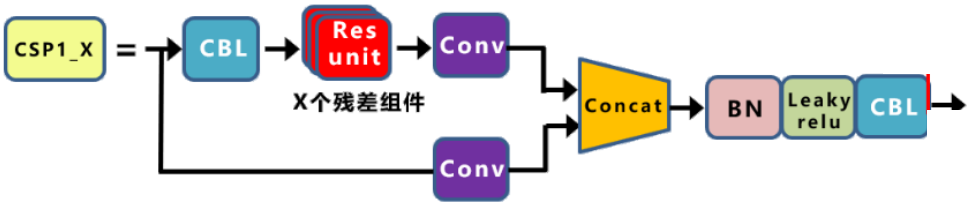


**图 10：CSP1_X 网络结构**

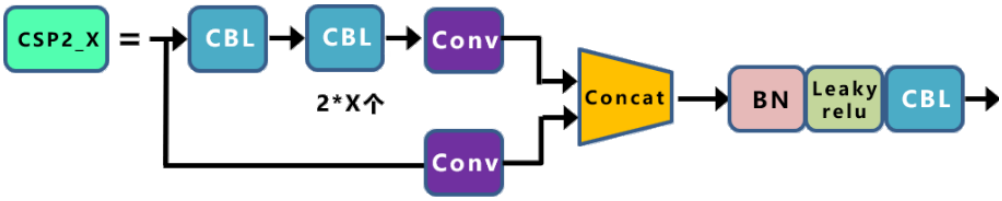


**图 11：CSP2_X 网络结构**

Focus：主要是进行切片操作再拼接的操作，再经过卷积层进行卷积操作，实现对输入图像的预处理。

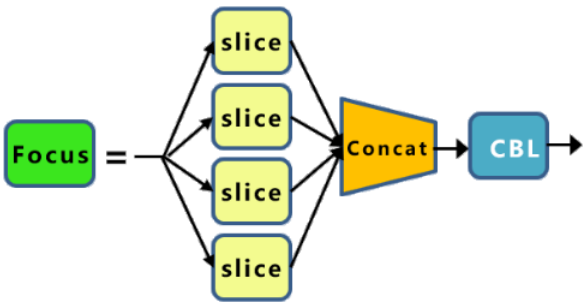


**图 12：Focus 网络结构**

**3）Neck：** 一系列混合和组合图像特征的网络层，并将图像特征传递到预测层

采用的是 FPN+PAN 结构。FPN 层的后面还添加了一个自底向上的特征金字塔。其中包含两个 PAN 结构。FPN 是自顶向下的，将高层的特征信息通过上采样的方式进行传递融合，得到进行预测的特征图。而特征金字塔则自底向上传达强定位特征，从不同的主干层对不同的检测层进行参数聚合。

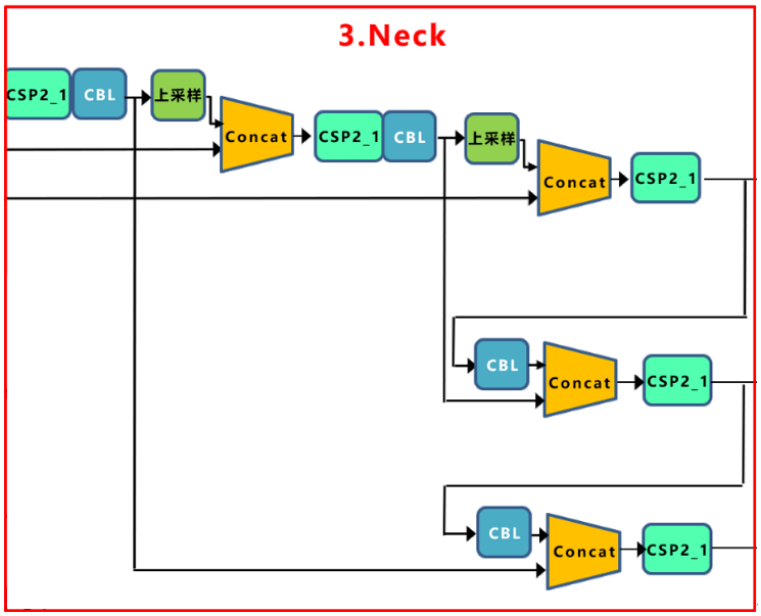


图 13：Neck 网络结构

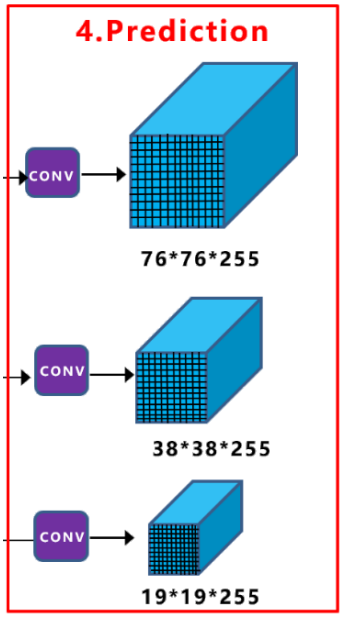


图 14：Prediction 网络结构

**4）Prediction：**对图像特征进行预测，生成边界框和并预测类别

输出层最需要考虑的有两部分。损失函数的选择及 NMS 非极大值抑制。

损失函数：Yolov5 中采用其中的 GIOU_Loss 做 Bounding box 的损失函数：

$$GIoU = IoU - \frac{|A_c - U|}{|A_c|}$$

先计算两个框的最小闭包区域面积（即同时包含了预测框和真实框的最小框的面积），再计算出 IoU，再计算闭包区域中不属于两个框的区域占闭包区域的比重，最后用 IoU 减去这个比重得到 GIoU。

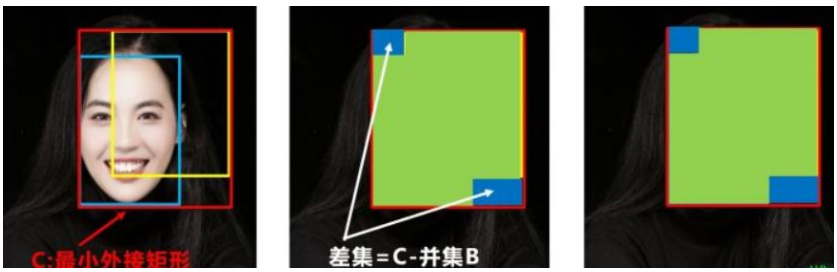


图 15：GIoU 损失函数

NMS 非极大值抑制：顾名思义就是抑制不是极大值的元素，搜索局部的极大值。在目标检测的后处理过程中，针对很多目标框的筛选，通常需要 nms 操作，筛除不合适的

目标框。Yolov5 中采用加权 nms 的方式。滑动窗口经提取特征，经分类器分类识别后，每个窗口都会得到一个分数。但是滑动窗口会导致很多窗口与其他窗口存在包含或者大部分交叉的情况。这时就需要用到 NMS 来选取那些邻域里分数最高，并且抑制那些分数低的窗口。

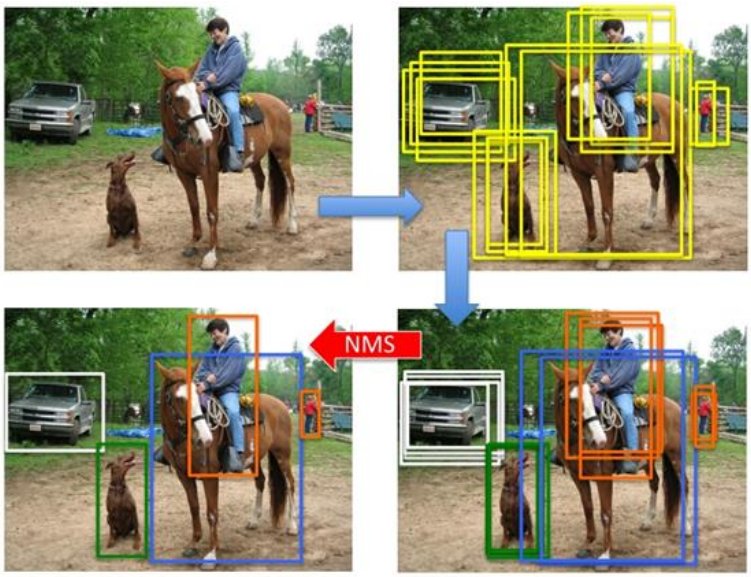


**图 16：NMS 非极大值抑制操作**

至此，完成了 YOLOv5 网络结构的介绍。

# 三、实验方法

## 1、训练平台

运行系统：elementary os（基于 ubuntu18.04）

CPU：Core i7

GPU：NVIDIA GeForce GTX 1050 Ti

Cuda 版本：11.10

## 2、数据收集

首先整体介绍一下数据收集的方式。因为我们的跟踪目标是在绿色草坪上的足球，因此需要采集大量绿色草坪上足球的图片，并框选标注出足球在图片中的位置。大多数的方法是先获取一个视频流，从视频流中截取图片的方式获得数据集。在采集视频流时注意应拍摄不同视角，不同清晰度的图片，从根本上丰富数据集中图片的种类。结合输入层的 mosaic 数据增强的方法可以获得一个比较丰富的数据集。

数据采集是我工作的一个最重要的环节，我先后采用了两种方式进行数据采集，分别是自己手动标注数据集及通过 Opencv 的自动跟踪模块进行自动标注。下面对着两种

方式分别进行介绍。

**1）手动标注**

首先获得一段视频序列，将球放在草坪上，绕着球拍摄一圈，其中切换不同角度和清晰度（如抖动相机等方式）。另存为文件名为 football.mp4


**图 17：视频流截图**

在获得图片之后，需要用某种手段将视频流分割成一张张图片，并按序号命名。因为是在 linux 系统上跑，我选择的是在使用 ffmpeg 命令，可以方便地从视频中生成图片序列，用于制作 gif 动图等用途。

```
指令：ffmpeg -i football.mp4 -r 1 -ss %03d.png
```

执行这条指令，可以实现将视频流转换成图片，名字按 001.jpg 这样的格式保存。

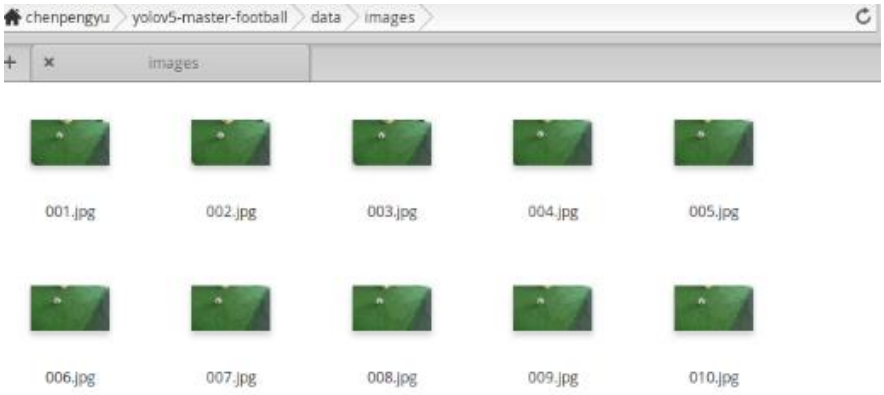


**图 18：生成图片序列**

在获得图片之后，我们需要对它进行手动标定，选择的数据标注软件是 labelImg。该软件可以框选出图片数据集中的目标物体，并打上对应分类的标签，我设置的标签是 football。输出格式选择 PascalVoc，里面包含的文件是 xml 格式。xml 文件中主要包含了被标注图片被保存的绝对路径，尺寸大小，被标注物体的锚框。锚框保存的是这个矩形框所包围的范围对应 x,y 像素位置的最大值和最小值。

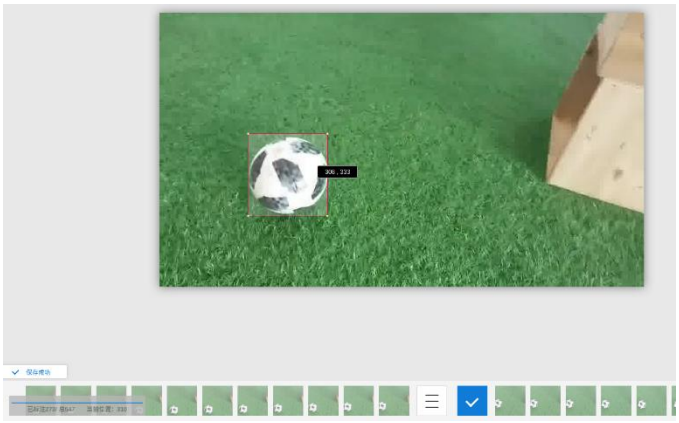

图 19：labelImg 标注图片

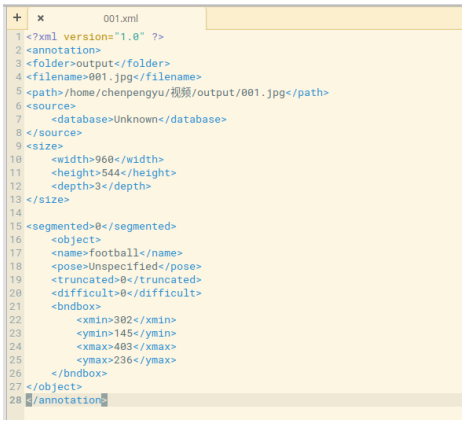


图 20：xml 数据格式

因为 github 有开源的 YOLOv5 框架，我们在这个基础上就行修改即可。在 data 目录下新建 Annotations，images，ImageSets，JPEGImages，labels 五个文件夹。其中 images 和 JPEGImages 存放的是原始的图片数据集，Annotations 存放的是标记后生成的 xml 文件，labels 存放的是保存标记内容的 txt 文件，ImageSets 存放的是训练数据集和测试数据集的分类情况。

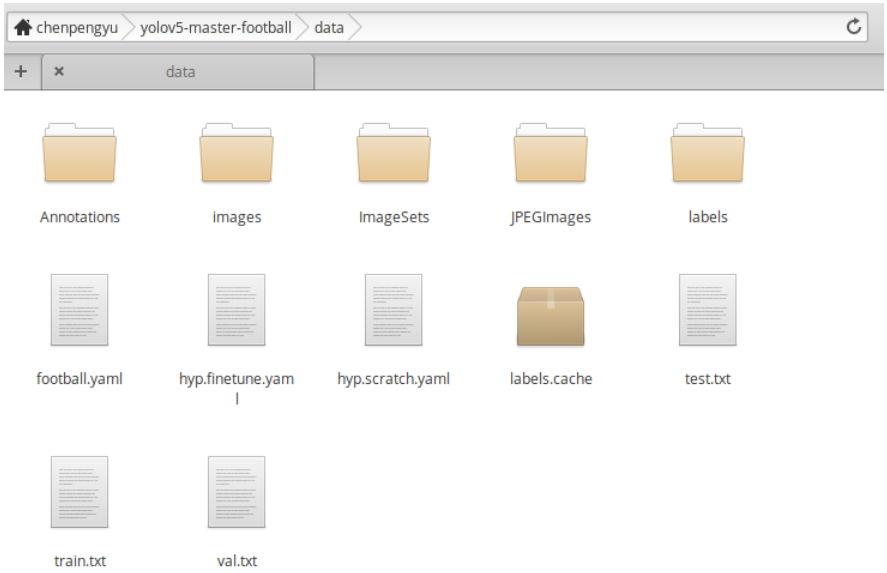


图 21：data 文件夹存放文件

其中，ImageSets 中的分类结果是通过运行 maketxt.py 获得。实现思路是读取 Annotations 中的所有 xml 文件，根据文件名进行随机划分，保存在 txt 文件中的是被分到某一数据集中的图片的序号。需要分出来的有训练集 train，验证集 val 和测试集 test。

在 labels 中 txt 存放的是每张图经过标注后的图片对应的数据。数据由 5 部分组成：[class x y w h]。其中，class 指的是分类的类别。因为我们这个项目相当于是一个单

目标跟踪识别，仅有一种类别，所以全部标 0。YOLOv5 的网络中要求输入的标签[x y w h]指的是[锚框中心点的 x 坐标　锚框中心点的 y 坐标　锚框的宽度　锚框的高度]，并且存储的数据是归一化后的。例如，设图片的整个长度为 length，锚框中心对应的 x 坐标为$x_l$，则：$x = \frac{x_l}{length}$，其余三个量同理。

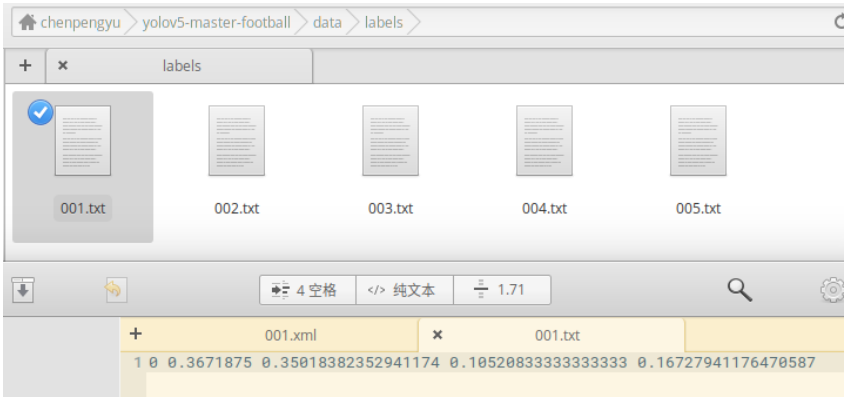


**图 22：数据集对应 label 文件**

至此，完成了手动标注的数据集。手动标注的优点是因为有人工的参与，可以高精度的提取数据集图片中的目标图像，可以提高训练的精度。但缺点就是随着数据集数据量的增大，手动标注是一个费时费力的工作，对于做一个限定场景下的单目标跟踪问题，性价比不是很高。

### 2）自动标注

利用 opencv 中的自动跟踪模块，可以用其中集成好的目标跟踪算法跟踪图片中的目标，再对一个视频序列进行处理，获得大量已经标好锚框的图片数据集。

Opencv 库中提供了我们许多现成的追踪器 Tracker，一共有 8 种，各有各的优缺点和应用场景。考虑到我们的目标是获得数据集，应保证数据集中的目标被标定的尽可能精确，而不用考虑视频流读取的 FPS 的实时性，因此选择精确度最高的 CSRT Tracker。

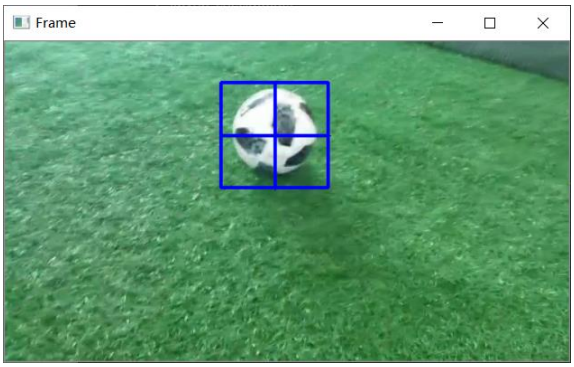


**图 23：框选目标球**


**图 24：CSRT Tracker 自动跟踪目标球**

利用 opencv 的跟踪器可以获得自动跟踪并返回目标球的锚框的参数。但 opencv 中

返回的参数值是以图片左上角为坐标原点的，因此转化到 YOLOv5 的标签还需要进行标签值的转换和归一化处理。（这里省略一部分之后补充）

通过自动标注的方法，可以在短时间内获得大量的经过标注的数据集，但由于 opencv 追踪器的锚框的尺度变化效果不好，追踪精度也和手动标注的相差很多，因此对于一些复杂环境下的目标跟踪问题，该方法制作出的数据集质量不是很高，还需要进行数据清洗，将识别效果比较差的数据剔除。但是我们的环境比较简单且单一，因此这样的方式制作出的数据集也足够我们使用。

## 3、训练过程

在制作完数据集之后，就可以开始训练过程。首先进行预训练，官方提供了四种预训练模型，我选择的是 yolov5m。预训练模型具有一定的好处，我们可以在自己的数据集上使用预训练模型，它可以为我们节省了大量的时间。通常来讲，使用预训练模型可以使每个损失下降更快和计算资源节省,可以加快梯度下降的收敛速度，更有可能获得一个低模型误差，或者低泛化误差的模型。可以降低因未初始化或初始化不当导致的梯度消失或者梯度爆炸问题。此情况会导致模型训练速度变慢，崩溃，直至失败。

修改 yaml 文件，里面包含了训练集、测试集存放位置、待识别物体标签等相关信息。选择对应网络结构的 yaml 文件，如 yolov5m.yaml。修改 train.py 中的部分参数。

接下来执行 train.py 开始训练，对于两种不同方式采集的数据集，我分别设置了 150 个 epoch，各训练了大概 3 个小时。一开始在另一台没有 GPU 的笔记本，在 ubuntu16 的环境下用 CPU 训练，结果内存不足直接报错，更换电脑之后问题得到解决。

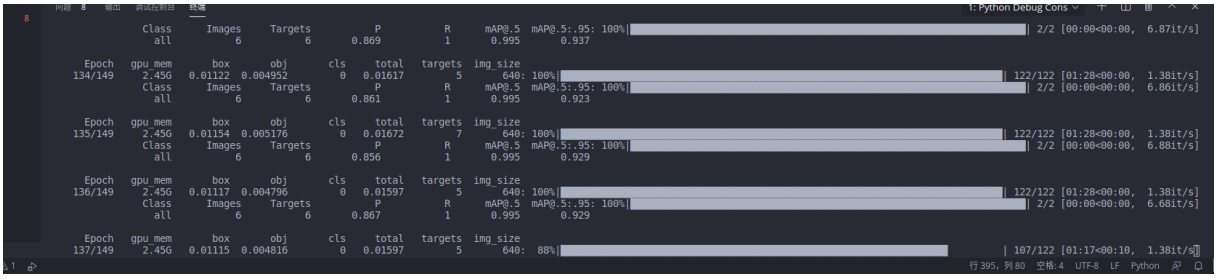


**图 25：训练过程**

在训练完成之后，可以可视化训练过程中各个参数的变化情况，各参数意义如下：

| | |
|---|---|
| Box | 对应的是损失函数 GIoU 均值，该值越小方框越准 |
| Objectness | 推测为目标检测 loss 均值，越小目标检测越准 |
| Classification | 越小分类越准，因为只有一类所以无 |
| Precision | 准确率 |

| | |
|---|---|
| Recall | 召回率 |
| mAP@0.5& mAP@0.5:0.95 | AP 是用 Precision 和 Recall 作为两轴作图后围成的面积，m 表示平均，@后面的数表示判定 iou 为正负样本的阈值，@0.5:0.95 表示阈值取 0.5:0.05:0.95 后取均值。 |

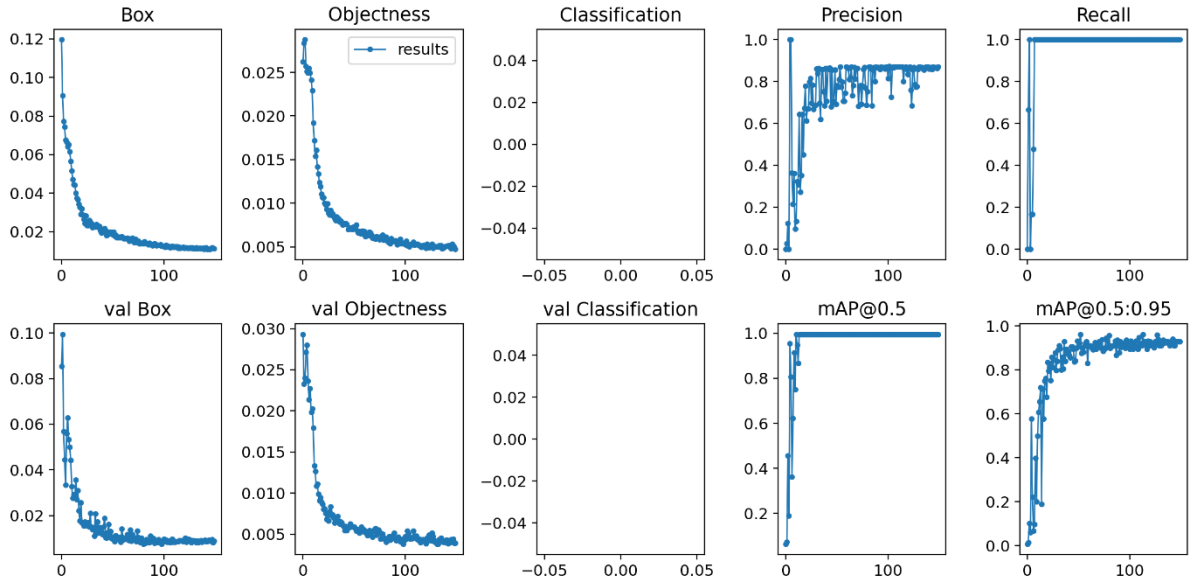


**图 26：数据集 1）训练过程中各参数变化**

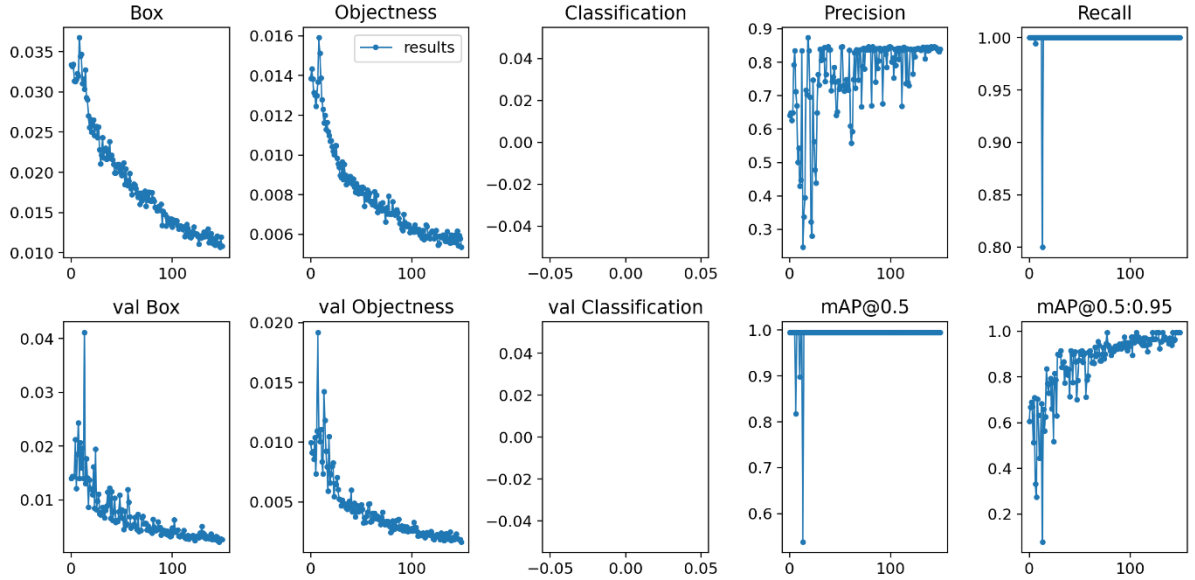


**图 27：数据集 2）训练过程中各参数变化**

训练结束以后，最后生成两个权重文件，一个是 best.pt，一个是 last.pt。其中 best.pt 是所有训练中得到的最好的一个权重存储的文件，last.pt 是最后一次训练所得的权重。这里我选择 best.pt 作为训练的结果，将其放到根目录，供之后进行测试使用。

## 4、测试过程

测试过程分成了两个部分，首先在自己的笔记本上测试，修改 detect.py 中的数据后直接运行：

指令：`python detect.py --source 0` ，调用电脑的前置摄像头开始测试。

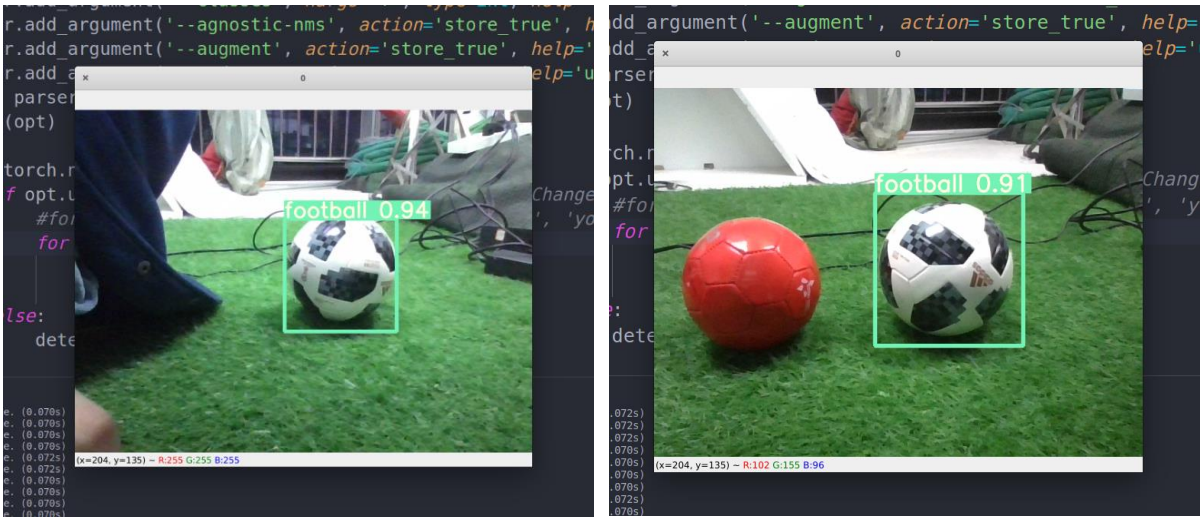


图 28：笔记本识别球测试效果

可以看出，球可以被准确识别。将网络搭载到机器人的上位机上进行测试，效果如下：

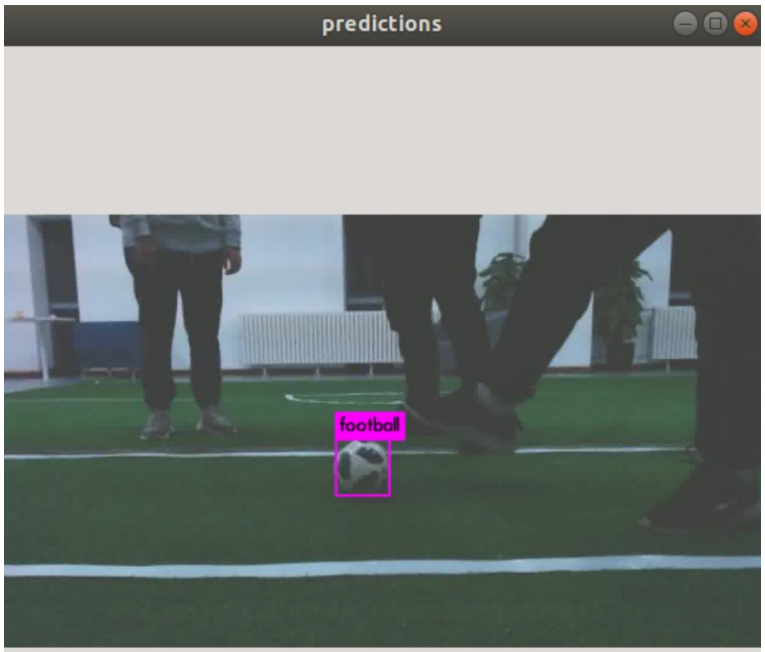


图 29：机器人识别球测试效果

可以看出，机器人在搭载 yolov5 的网络之后，同样可以准确识别球。但是同样也有很大的问题，受 CPU 的性能限制，在机器人上得到的识别视频流大约仅为 5 帧/s，只能勉强满足识别的需求。但也比从前部署的网络，只能达到三秒一帧要好得多。

# 四、结论

通过以上研究我们可以得到，yolov5 在目标跟踪任务上确实具有很强的性能，同时适合部署在一些低性能的嵌入式设备上，在一定程度上可以满足实时跟踪识别的要求。

考虑到不能满足实时性的实际情况，我查阅资料，思考了几种可能的改进方式：

1、神经网络在运行时仍然需要占用很大的内存，计算量很大，很大一部分原因是数据格式为 FP32，浮点运算需要的计算量非常大。针对浮点运算计算量很大的问题，一个神经网络的研究方向是模型量化（低精度推理）。将 FP32 的模型保存成 INT8 的格式。深度学习模型主要是记录每个 layer（比如卷积层/全连接层） 的 weights 和 bias，FP32 模型中，每个 weight 数值原本需要 32-bit 的存储空间，量化之后只需要 8-bit 即可。因此，模型的大小将直接降为将近 1/4。不仅模型大小明显降低，采用 8-bit 之后也将明显减少对内存的使用，这也意味着低精度推理过程将明显减少内存的访问带宽需求，提高高速缓存命中率。

2、将神经网络通过 C 语言来实现。Python 的运行环境会限制神经网络的性能，而 C 语言相较 python 更接近计算机的底层，具有更快的运算和处理速度。

3、选择更高性能的上位机。如 Nvida 公司推出的 TX2 等嵌入式设备，具有足够数量的 CUDA core，在神经网络的运算上具有比 cpu 更高的运行速度。

# 五、成果展示

我所在的团队是足球机器人基地人型实物组，我们在 2020 年 11 月 13 日-15 日于南京举办的 2020Robocup 机器人世界杯中国赛中，最终取得了正赛全国亚军的好成绩，实现了基地在这一项目上的突破。虽然这个项目不是我主要负责，但是在 pytorch 课堂上学到的知识，确实让我帮助他们更好的完成了这个比赛，最后才能取得优异的成绩。

**图 30：2020Robocup 机器人世界杯中国赛 op3 比赛中**


图 31：2020Robocup 机器人世界杯中国赛获奖


图 32：2020Robocup 机器人世界杯中国赛颁奖典礼

# 附录

针对数据集 2）构建的部分代码实现：

opencv_object_tracking.py （获取锚框的对应坐标）

```
from imutils.video import VideoStream
from imutils.video import FPS
import argparse
import imutils
import time
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", type=str,
                help="path to input video file")
ap.add_argument("-t", "--tracker", type=str, default="kcf",
                help="OpenCV object tracker type")
args = vars(ap.parse_args())

(major, minor) = cv2.__version__.split(".")[:2]

if int(major) == 3 and int(minor) < 3:
    tracker = cv2.Tracker_create(args["tracker"].upper())
else:
    OPENCV_OBJECT_TRACKERS = {
        "csrt": cv2.TrackerCSRT_create,
        "kcf": cv2.TrackerKCF_create,
        "boosting": cv2.TrackerBoosting_create,
        "mil": cv2.TrackerMIL_create,
        "tld": cv2.TrackerTLD_create,
        "medianflow": cv2.TrackerMedianFlow_create,
        "mosse": cv2.TrackerMOSSE_create
    }
    tracker = OPENCV_OBJECT_TRACKERS[args["tracker"]]()

initBB = None

if not args.get("video", False):
    print("[INFO] starting video stream...")
    vs = VideoStream(src=0).start()
    time.sleep(1)
else:
    vs = cv2.VideoCapture(args["video"])

fps = None

num = 1
# loop over frames from the video stream
while True:
    # grab the current frame, then handle if we are using a
    # VideoStream or VideoCapture object
    frame = vs.read()
    frame = frame[1] if args.get("video", False) else frame
    # check to see if we have reached the end of the stream
```

```
    if frame is None:
        break
    # resize the frame (so we can process it faster) and grab the
    # frame dimensions
    frame = imutils.resize(frame, width=500)
    (H, W) = frame.shape[:2]
    # check to see if we are currently tracking an object
    if initBB is not None:
        # grab the new bounding box coordinates of the object
        (success, box) = tracker.update(frame)
        # check to see if the tracking was a success
        if success:
            cv2.imwrite("./data/images/{}.jpg".format(str(num).rjust(3,'0')),
frame)
            (x, y, w, h) = [int(v) for v in box]
            cv2.imwrite("./data/JPEGImages/{}.jpg".format(str(num).rjust(3,'0'
)), frame)
            cv2.rectangle(frame, (x, y), (x + w, y + h),
                (0, 255, 0), 2)
            dw = 1./frame.shape[1]
            dh = 1./frame.shape[0]
            x = round(int(x+w/2)*dw,3)
            y = round(int(y+h/2)*dh,3)
            w = round(w*dw,3)
            h = round(h*dh,3)
            print("{}.jpg : ".format(str(num).rjust(3,'0')),x,y,w,h)
            label = open("./data/labels/{}.txt".format(str(num).rjust(3,'0')),
'w') #写入模式
            label.write("0 {} {} {} {}".format(x,y,w,h))
            label.close()
            num+=1
        # update the FPS counter
        fps.update()
        fps.stop()
        # initialize the set of information we'll be displaying on
        # the frame
        info = [
            ("Tracker", args["tracker"]),
            ("Success", "Yes" if success else "No"),
            ("FPS", "{:.2f}".format(fps.fps())),
        ]
        # loop over the info tuples and draw them on our frame
        for (i, (k, v)) in enumerate(info):
            text = "{}: {}".format(k, v)
            cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    # if the 's' key is selected, we are going to "select" a bounding
    # box to track
    if key == ord("s"):
        # select the bounding box of the object we want to track (make
        # sure you press ENTER or SPACE after selecting the ROI)
        initBB = cv2.selectROI("Frame", frame, fromCenter=False,
            showCrosshair=True)
```

```
        # start OpenCV object tracker using the supplied bounding box
        # coordinates, then start the FPS throughput estimator as well
        tracker.init(frame, initBB)
        fps = FPS().start()
    # if the `q` key was pressed, break from the loop
    elif key == ord("q"):
        break
# if we are using a webcam, release the pointer
if not args.get("video", False):
    vs.stop()
# otherwise, release the file pointer
else:
    vs.release()
# close all windows
cv2.destroyAllWindows()
```

maketxt.py（划分训练集、验证集、测试集）

```
import os
import random
trainval_percent = 0.1
train_percent = 0.9
srcfilepath = 'data/labels'
txtsavepath = 'data/ImageSets'
total_src = os.listdir(srcfilepath)
num = len(total_src)
list = range(num)
tv = int(num * trainval_percent)
tr = int(tv * train_percent)
trainval = random.sample(list, tv)
train = random.sample(trainval, tr)
ftrainval = open('data/ImageSets/trainval.txt', 'w')
ftest = open('data/ImageSets/test.txt', 'w')
ftrain = open('data/ImageSets/train.txt', 'w')
fval = open('data/ImageSets/val.txt', 'w')

for i in list:
    name = total_src[i][:-4] + '\n'
    if i in trainval:
        ftrainval.write(name)
        if i in train:
            ftest.write(name)
        else:
            fval.write(name)
    else:
        ftrain.write(name)
ftrainval.close()
ftrain.close()
fval.close()
ftest.close()
```

voc_label.py （构建标签）

```
# xml 解析包
import xml.etree.ElementTree as ET
import pickle
import os
```

```
# os.listdir() 方法用于返回指定的文件夹包含的文件或文件夹的名字的列表
from os import listdir, getcwd
from os.path import join


sets = ['train', 'test', 'val']
classes = ['football']

# 返回当前工作目录
wd = getcwd()
print(wd)


for image_set in sets:  #['train', 'test', 'val']
    '''
    对所有的文件数据集进行遍历
    做了两个工作：
        1．讲所有图片文件都遍历一遍，并且将其所有的全路径都写在对应的 txt 文件中去，方便定位
        2．同时对所有的图片文件进行解析和转化，将其对应的 bundingbox 以及类别的信息全部解析写到 label 文件中去
                最后再通过直接读取文件，就能找到对应的 label 信息
    '''
    # 先找 labels 文件夹如果不存在则创建
    if not os.path.exists('data/labels/'):
        os.makedirs('data/labels/')
    # 读取在 ImageSets/Main 中的 train、test..等文件的内容
    # 包含对应的文件名称
    image_ids = open('data/ImageSets/%s.txt' % (image_set)).read().strip().split()
    # 打开对应的 2012_train.txt 文件对其进行写入准备
    list_file = open('data/%s.txt' % (image_set), 'w')
    # 将对应的文件_id 以及全路径写进去并换行
    for image_id in image_ids:
        list_file.write('data/images/%s.jpg\n' % (image_id))
    # 关闭文件
    list_file.close()
```