一文带你学会使用 YOLO 及 Opencv 完成图像目标检测 | 附源码

作者：Adrian Rosebrock

来源：我是程序员

编译：阿里云云栖社区

原文：

https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

代码：

https://app.monstercampaigns.com/c/ubdhbuoif9qpbqkebe2t/

摘要： 本文介绍使用opencv和yolo完成图像目标检测，代码解释详细，附源码，上手快。

计算机视觉领域中，目标检测一直是工业应用上比较热门且成熟的应用领域，比如人脸识别、行人检测等，相对于图像分类任务而言，目标检测会更加复杂一些，不仅需要知道这是哪一类图像，而且要知道图像中所包含的内容有什么及其在图像中的位置，因此，其工业应用比较广泛。那么，今天将向读者介绍该领域中表现优异的一种算算法——“你只需要看一次”（you only look once，yolo），提出该算法的作者风趣幽默可爱，其个人主页及论文风格显示了其性情，目前该算法已是第三个版本，简称YoLo V3。闲话少叙，下面进入教程的主要内容。

在本教程中，将学习如何使用YOLO、OpenCV和Python检测图像的对象。主要内容有：

- 简要讨论YOLO算法；
- 使用YOLO、OpenCV、Python进行图像检测；

什么是YOLO?

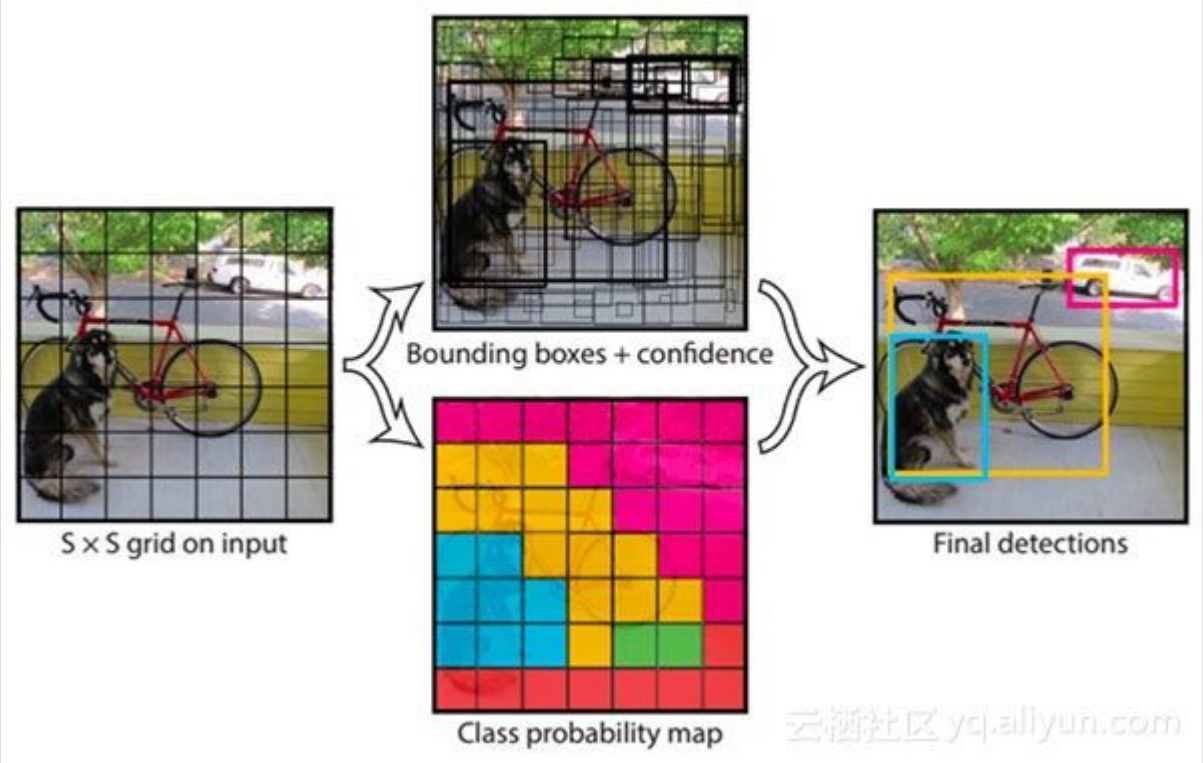

图1： YOLO目标检测器简化示意图

当涉及基于深度学习的对象检测时，常用的三类算法有：

- R-CNN家族系列算法：R-CNN、fast R-CNN以及faster R-CNN；
- 单发检测器（SSD）；
- YOLO算法；

R-CNN算法是最早的基于深度学习的目标检测器之一，其结构是两级网络：

- 首先需要诸如选择性搜索之类的算法来提出可能包含对象的候选边界框；
- 然后将这些区域传递到CNN算法进行分类；

R-CNN算法存在的问题在于其仿真很慢，并且不是完整的端到端的目标检测器。

Fast R-CNN算法对原始R-CNN进行了相当大的改进，即提高准确度并减少执行正向传递所花费的

时间，但是，该模型仍然依赖于外部区域搜索算法。

直到2015年，faster R-CNN才成为真正的端到端深度学习目标检测器，删除了选择性搜索的要求，而是依赖于（1）完全卷积的区域提议网络（RPN）和（2）可以预测对象边界框和“对象”分数（量化它是一个区域的可能性的分数）。然后将RPN的输出传递到R-CNN组件以进行最终分类和标记。

R-CNN系列算法的检测结果一般都非常准确，但R-CNN系列算法最大的问题在仿真速度——非常慢，即使是在GPU上也仅获得5 FPS。

为了提高基于深度学习的目标检测器的速度，单次检测器（SSD）和YOLO都使用单级检测器策略（one stage）。这类算法将对象检测视为回归问题，获取给定的输入图像并同时学习边界框坐标和相应的类标签概率。通常，单级检测器往往不如两级检测器准确，但其速度明显更快。YOLO是单级检测器中一个很好的算法。

YOLO算法于2015年提出，在GPU上获得了 45 FPS性能，此外，同时也提出了一个较小的变体称为“Fast YOLO”，在GPU上达到155 FPS的性能。

YOLO经历了许多次的迭代，包括YOLOv2，能够检测超过9,000个目标。直到最近提出的YOLOv3算法，YOLOv3模型比之前的版本要复杂得多，但它是YOLO系列目标检测器中最好的一款。

本文使用YOLOv3，并在COCO数据集上进行训练。

COCO数据集由80个标签组成，可以使用此链接找到YOLO在COCO数据集上训练的内容的完整列表。

项目结构

在终端中使用tree命令，可以很方便快捷地生成目标树：

```
$ tree
.
├── images
│   ├── baggage_claim.jpg
│   ├── dining_table.jpg
│   ├── living_room.jpg
│   └── soccer.jpg
├── output
│   ├── airport_output.avi
│   ├── car_chase_01_output.avi
│   ├── car_chase_02_output.avi
│   └── overpass_output.avi
├── videos
│   ├── airport.mp4
│   ├── car_chase_01.mp4
│   ├── car_chase_02.mp4
│   └── overpass.mp4
├── yolo-coco
│   ├── coco.names
│   ├── yolov3.cfg
│   └── yolov3.weights
├── yolo.py
└── yolo_video.pyhttp
```

从上面可以看出，项目包括4个文件夹和2个Python脚本。

目录（按重要性顺序）是：

- `yolo - coco /`：YOLOv3对象检测器预先（在COCO数据集上）训练得到最终的权重文件，可以在Darknet团队主页找到对应的文件；
- `images /`：此文件夹包含四个静态图像，之后将执行对象检测以进行测试和评估；
- `videos/`：使用YOLO对图像进行目标检测器后，将实时处理视频。该文件夹中包含四个示例视频可供测试；
- `输出/`：输出已由YOLO处理并带有边界框和类名称注释的视频可以放在此文件夹中；

此外还有两个Python脚本—— `yolo .py` 和 `yolo_video.py` ，第一个脚本用于图像处理，第二个脚本用于视频处理。下面进入实战内容，你准备好了吗？

.

将YOLO应用于图像对象检测

首先将YOLO目标检测器应用于图像中，首先打开项目中的 `yolo .py` 并插入以下代码：

```
# import the necessary packages
import numpy as np
import argparse
import time
import cv2
import os

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-y", "--yolo", required=True,
    help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
    help="threshold when applying non-maxima suppression")
args = vars(ap.parse_args())
```

在使用之前，需要为此脚本安装 3.4.2+版本以上的OpenCV，可以直接使用 `pip install opencv-python==3.4.2` 安装，你也可以在这里找到OpenCV安装教程，这里注意一点，OpenCV 4目前处于测试阶段，这里建议去安装OpenCV 3.4.2+。

首先，导入所需的数据包——OpenCV和NumPy。现在解析四个命令行参数，命令行参数在运行时处理，允许我们从终端更改脚本的输入。如果你对其不熟悉，建议阅读相关的内容。命令行参数包括：

- `-- image` ：输入图像的路径；

- `-- yolo` ：YOLO文件路径，脚本将加载所需的YOLO文件，以便在图像上执行对象检测；
- `-- confidence` ：过滤弱检测的最小概率，默认值设置为0.5，但该值也可以随意设置；
- `-- threshold` ：非最大值抑制阈值，默认值设置为 0.3，可以在此处阅读有关非最大值抑制的更多信息。

  解析之后，`args` 变量是一个包含命令行参数的键值对的字典。下面为每个标签设置随机颜色：

```
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
    dtype="uint8")
```

上述加载所有类 LABELS，其类型是列表，保存的是类别名称，然后将随机颜色分配给每个标签 。

下面设置YOLO权重和配置文件的路径，然后从磁盘加载YOLO文件：

```
# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])

# load our YOLO object detector trained on COCO dataset (80 classes)
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
```

从磁盘加载YOLO文件后，并利用OpenCV中的 `cv2.dnn.readNetFromDarknet` 函数从中读取网络文件及权重参数，此函数需要两个参数 `configPath` 和 `weightsPath` ，这里再次强调，：OpenCV 的版本至少是3.4.2及以上才能运行此代码，因为它需要加载YOLO所需的更新的 `dnn` 模块。

下面加载图像并处理：

```
# load our input image and grab its spatial dimensions
image = cv2.imread(args["image"])
(H, W) = image.shape[:2]

# determine only the *output* layer names that we need from YOLO
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# construct a blob from the input image and then perform a forward
# pass of the YOLO object detector, giving us our bounding boxes and
# associated probabilities
blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
    swapRB=True, crop=False)
net.setInput(blob)
start = time.time()
layerOutputs = net.forward(ln)
end = time.time()

# show timing information on YOLO
print("[INFO] YOLO took {:.6f} seconds".format(end - start))
```

在该代码中：

- 加载输入 图像并获得其尺寸；
- 确定YOLO模型中的输出图层名称；
- 从图像构造一个 blob结构；

当blob准备好了后，我们就会

- 通过YOLO网络进行前向传递；
- 显示YOLO的推理时间；

  现在采取措施来过滤和可视化最终的结果。首先，让我们初步化一些处理过程中需要的列表：

```
# initialize our lists of detected bounding boxes, confidences, and
# class IDs, respectively
boxes = []
confidences = []
classIDs = []
```

这些列表包括：

- `boxes`：对象的边界框。
- `confidences`：YOLO分配给对象的置信度值，较低的置信度值表示该对象可能不是网络认为的对象。上面的命令行参数中将过滤掉不大于 0.5阈值的对象。
- `classIDs`：检测到的对象的类标签。

下面用YOLOlayerOutputs中的数据填充这些列表：

```
# loop over each of the layer outputs
for output in layerOutputs:
    # loop over each of the detections
    for detection in output:
        # extract the class ID and confidence (i.e., probability) of
        # the current object detection
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]

        # filter out weak predictions by ensuring the detected
        # probability is greater than the minimum probability
        if confidence > args["confidence"]:
            # scale the bounding box coordinates back relative to the
            # size of the image, keeping in mind that YOLO actually
            # returns the center (x, y)-coordinates of the bounding
            # box followed by the boxes' width and height
            box = detection[0:4] * np.array([W, H, W, H])
            (centerX, centerY, width, height) = box.astype("int")

            # use the center (x, y)-coordinates to derive the top and
            # and left corner of the bounding box
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))

            # update our list of bounding box coordinates, confidences,
            # and class IDs
            boxes.append([x, y, int(width), int(height)])
            confidences.append(float(confidence))
            classIDs.append(classID)
```

在这个块中：

- 循环遍历每个 `layerOutputs` ；
- 循环每个 `detection` 中 `output` ；
- 提取 `classID` 和 `confidence` ；
- 使用 `confidence` 滤除弱检测；

过滤掉了不需要的检测结果后，我们将：

- 缩放边界框坐标，以便我们可以在原始图像上正确显示它们；
- 提取边界框的坐标和尺寸，YOLO返回边界框坐标形式： （centerX，centerY，width，height）；
- 使用此信息导出边界框的左上角（x，y）坐标；
- 更新 `boxes`， `confidences` ， `classIDs` 列表。

有了这些数据后，将应用“非最大值抑制”（non-maxima suppression，nms）：

```
# apply non-maxima suppression to suppress weak, overlapping bounding
# boxes
idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
    args["threshold"])
```

YOLO算法并没有应用非最大值抑制，这里需要说明一下。应用非最大值抑制可以抑制明显重叠的边界框，只保留最自信的边界框，NMS还确保我们没有任何冗余或无关的边界框。

利用OpenCV内置的NMS DNN模块实现即可实现非最大值抑制，所需要的参数是边界 框、 置信度、以及置信度阈值和NMS阈值。

最后在图像上绘制检测框和类文本：

```
# ensure at least one detection exists
if len(idxs) > 0:
    # loop over the indexes we are keeping
    for i in idxs.flatten():
        # extract the bounding box coordinates
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])

        # draw a bounding box rectangle and label on the image
        color = [int(c) for c in COLORS[classIDs[i]]]
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])
        cv2.putText(image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
            0.5, color, 2)

# show the output image
cv2.imshow("Image", image)
cv2.waitKey(0)
```

假设存在至少一个检测结果，就循环用非最大值抑制确定idx 。然后，我们使用随机类颜色在图像上绘制边界框和文本 。最后，显示结果图像，直到用户按下键盘上的任意键。

下面进入测试环节，打开一个终端并执行以下命令：

```
$ python yolo.py --image images/baggage_claim.jpg --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] YOLO took 0.347815 seconds
```

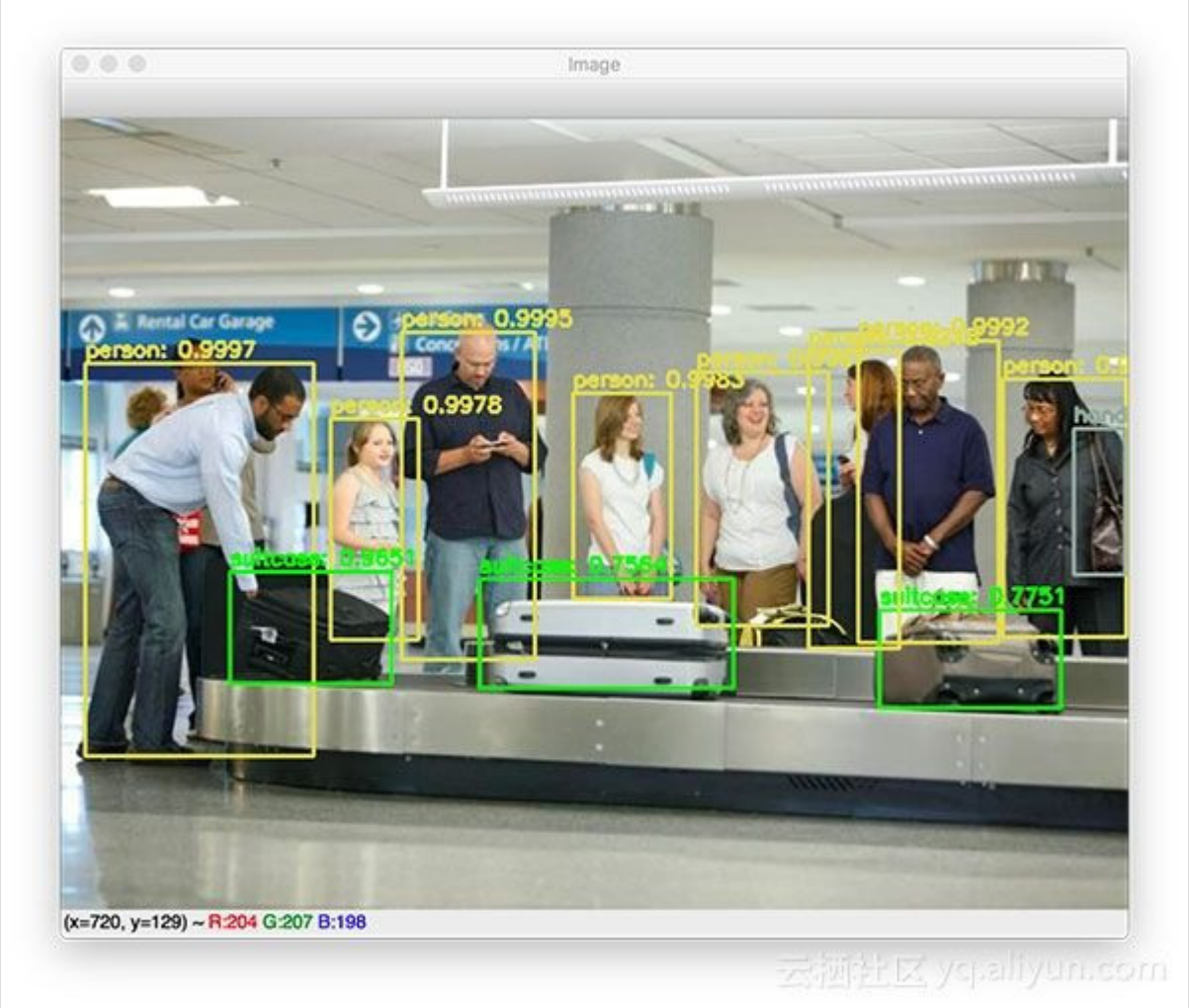

图2：YOLO用于检测机场中的人员和行李

从上图可以看到，YOLO不仅检测了输入图像中的每个人，还检测了手提箱。此外，可以从图像的右上角看到，YOLO还检测到女士肩上的手提包。

我们试试另一个例子：

```
$ python yolo.py --image images/living_room.jpg --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] YOLO took 0.340221 seconds
```

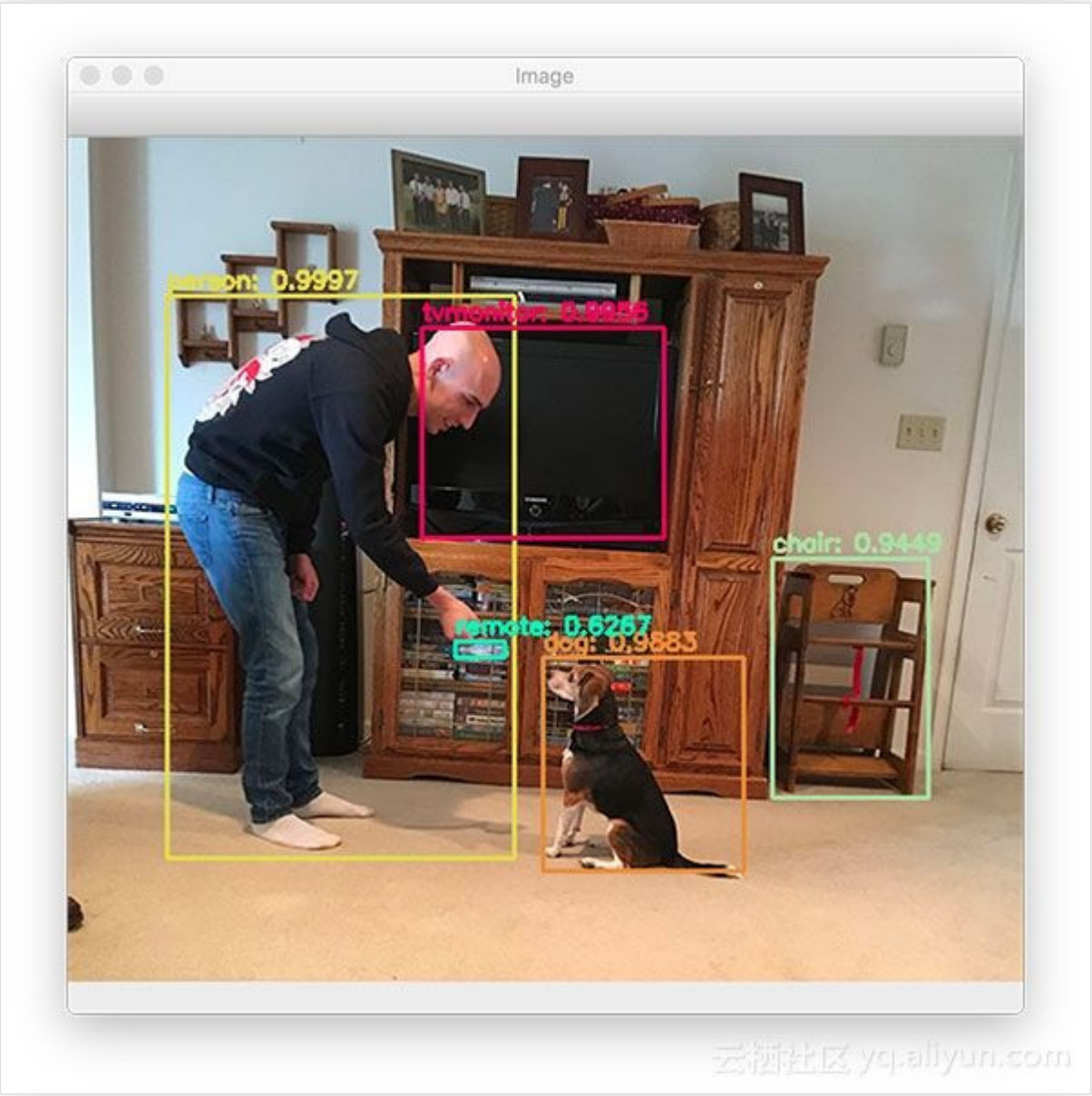

图3： YOLO用于检测人、狗、电视和椅子

YOLO还可以检测电视显示器和椅子，令我惊讶的是YOLO能够检测到椅子，因为它是手工制作的老式“婴儿高脚椅”。

有趣的是，YOLO认为我手中有一个遥控器，它实际上不是遥控器——玻璃反射的VHS录，仔细盯着这个地方看，它实际上看起来非常像遥控器。

以下示例图像演示了YOLO对象检测器的局限性和弱点：

```
$ python yolo.py --image images/dining_table.jpg --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] YOLO took 0.362369 seconds
```

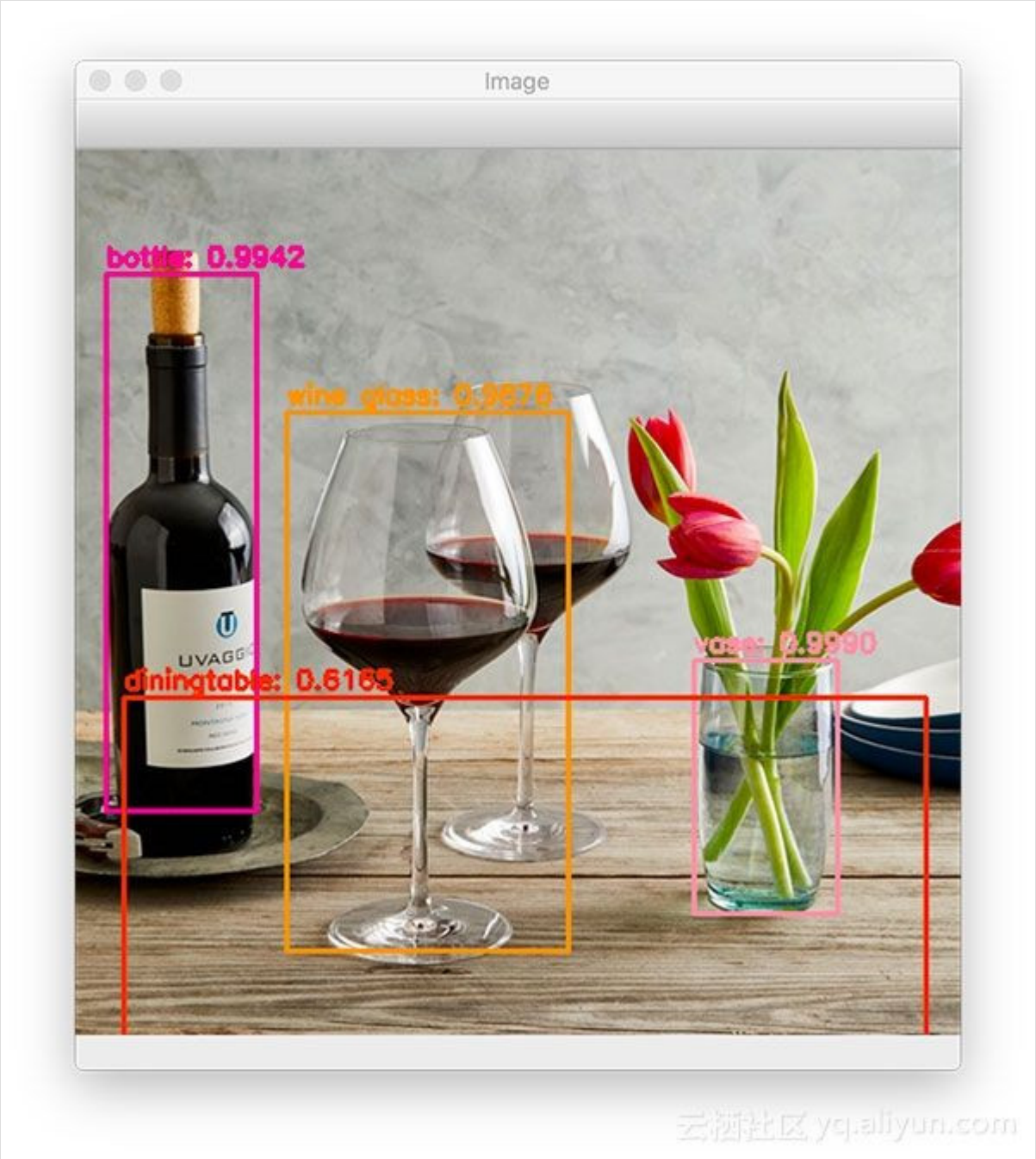

图4： YOLO用于检测餐桌

虽然YOLO正确检测到葡萄酒瓶、餐桌和花瓶，但只有两个酒杯中的一个被正确检测到。

下面尝试最后一幅图像：

```
$ python yolo.py --image images/soccer.jpg --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] YOLO took 0.345656 seconds
```

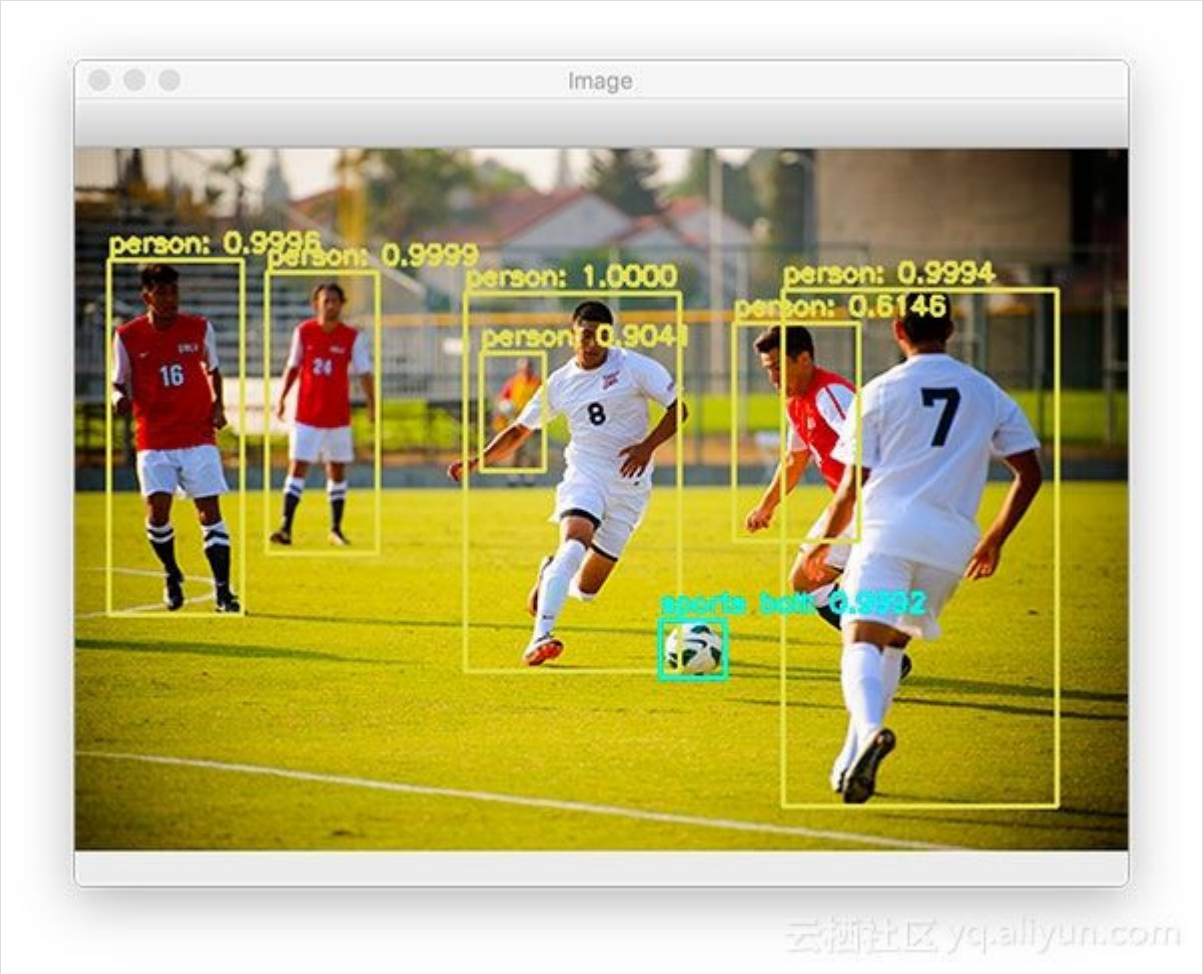

图5：使用YOLO检测足球运动员和足球

YOLO能够正确地检测球场上的每个球员，包括足球本身。请注意，尽管区域高度模糊且部分遮挡，但仍会检测到背景中的人。

推荐阅读

- 盘点 NeurIPS 两届神经网络对抗赛 (NIPS2017/2018)
- 深度学习图像识别的未来：机遇与挑战并存
- 多目标追踪器：用 OpenCV 实现多目标追踪（C++/Python）

微信公众号: 极市平台（ID: extrememart ）