**基于YOLO\_V3的火焰检测系统**

目录

**[1. 研究背景与意义](#_Toc9990_WPSOffice_Level1)** **[2](#_Toc9990_WPSOffice_Level1)**

**[2. 功能与指标](#_Toc4333_WPSOffice_Level1)** **[2](#_Toc4333_WPSOffice_Level1)**

**[3. 算法原理](#_Toc31445_WPSOffice_Level1)** **[3](#_Toc31445_WPSOffice_Level1)**

[3.1 YOLO\_v3网络结构](#_Toc4333_WPSOffice_Level2) [3](#_Toc4333_WPSOffice_Level2)

[3.2 Darknet-53特征提取网络](#_Toc31445_WPSOffice_Level2) [4](#_Toc31445_WPSOffice_Level2)

[3.3 利用多尺度特征进行对象检测](#_Toc8146_WPSOffice_Level2) [4](#_Toc8146_WPSOffice_Level2)

[3.4 多种尺度的先验框](#_Toc2669_WPSOffice_Level2) [5](#_Toc2669_WPSOffice_Level2)

[3.5 输入映射到输出](#_Toc30933_WPSOffice_Level2) [5](#_Toc30933_WPSOffice_Level2)

[3.6 损失函数](#_Toc12831_WPSOffice_Level2) [6](#_Toc12831_WPSOffice_Level2)

[3.7 迁移学习](#_Toc2861_WPSOffice_Level2) [6](#_Toc2861_WPSOffice_Level2)

**[4. 实现过程](#_Toc8146_WPSOffice_Level1)** **[6](#_Toc8146_WPSOffice_Level1)**

[4.1 网络爬取数据和标注数据](#_Toc31220_WPSOffice_Level2) [6](#_Toc31220_WPSOffice_Level2)

[4.2 模型的构建](#_Toc26045_WPSOffice_Level2) [8](#_Toc26045_WPSOffice_Level2)

[4.3 模型训练](#_Toc18489_WPSOffice_Level2) [12](#_Toc18489_WPSOffice_Level2)

[4.4 火焰检测](#_Toc1201_WPSOffice_Level2) [12](#_Toc1201_WPSOffice_Level2)

[4.5 构建基于Flask的实时火焰检测网站](#_Toc25395_WPSOffice_Level2) [13](#_Toc25395_WPSOffice_Level2)

**[5. 结果分析](#_Toc2669_WPSOffice_Level1)** **[15](#_Toc2669_WPSOffice_Level1)**

[5.1 图片检测效果](#_Toc3292_WPSOffice_Level2) [16](#_Toc3292_WPSOffice_Level2)

[5.2 Flask构建的网站及实时检测](#_Toc32136_WPSOffice_Level2) [16](#_Toc32136_WPSOffice_Level2)

[5.3 视频文件](#_Toc27390_WPSOffice_Level2) [17](#_Toc27390_WPSOffice_Level2)

**[6. 参考文献](#_Toc30933_WPSOffice_Level1)** **[18](#_Toc30933_WPSOffice_Level1)**

**[7. 分工合作](#_Toc12831_WPSOffice_Level1)** **[18](#_Toc12831_WPSOffice_Level1)**

# 研究背景与意义

物体检测一直是计算机视觉领域经久不衰的研究方向。物体检测同样是一个主观的过程，对于人类来说相当简单。就连一个没受过任何训练的孩子通过观察图片中不同的颜色、区域等特征就能轻易定位出目标物体。但计算机收到这些RGB像素矩阵，不会直接得到目标（如行人、车辆等）的抽象概念，更不必说定位其位置了。再加上目标形态千差万别，目标和背景重合等问题，使得目标检测难上加难。

传统的目标检测算法包括三个阶段，首先生成目标建议框，接着提取每个建议框中的特征，最后根据特征进行分类。以下是这三个阶段的具体过程：

1，生成目标建议框。当输入一张原始图片时，计算机只认识每一个像素点，想要用方框框出目标的位置以及大小，最先想到的方法就是穷举建议框，具体的做法就是用滑动窗口扫描整个图像，还要通过缩放来进行多尺度滑窗。很显然这种方法计算量很大，很多都是重复的计算，并且效率极低。

2，提取每个建议框中的特征。在传统的检测中，常见的HOG[20]算法对物体边缘使用直方图统计来进行编码，有较好的表达能力。然而传统特征设计需要人工指定，达不到可靠性的要求。

3，分类器的设计。传统的分类器在机器学习领域非常多。具有代表性的SVM将分类间隔最大化来获得分类平面的支持向量，在指定特征的数据集上表现良好。

 后来人们提出一种新的物体检测方法--YOLO。YOLO之前的物体检测方法主要是通过区域监测产生大量的可能包含待检测物体的潜在预测框，再用分类器去判断每个边界框里是否包含有物体，以及物体所属类别的 可能性或者 置信度，如R-CNN,Fast-R-CNN,Faster-R-CNN等。YOLO不同于这些物体检测方法，它将物体检测任务当做一个回归问题来处理，使用一个神经网络，直接从一整张图像来预测出边界框的坐标、框中包含物体的置信度和物体的可能性。

我们的课题做了关于图片火焰检测，或者视频火焰检测，或者摄像头实时检测火焰。本课题有十分重要的现实意义，在日常生活中火灾给人们带来的灾难和损失都是难以想象的，我们的工作从搜集不同大小的火焰，例如蜡烛的火焰，打火机的火焰到火灾的火焰，不同尺度的火焰，然后人工进行标记火焰的位置，这个工作量是很大的，但也是必要的，因为只要高质量的输入数据才能让模型有较好的泛华性。最后我们完成的模型检测一张图片仅需要0.001s~0.004s,就是说在100fps以下的1080P的视频或者摄像头的画面数据都能准确且实时地识别，能够及时发现火焰，很大程度地减少火灾的发生，能够挽救无数生命和财产损失。有极强的现实意义。

# 功能与指标

1. 能够准检测出别给定图片中的火焰及其数量。
2. 能够准确检测出火焰所在的位置并能够准确框出物体并输出。
3. Flask制作网页前端，能够有更好的交互性，界面友好。
4. 能够在视频中准确地检测火焰及其数量。
5. 能够在视频中能够准确检测出火焰所在的位置并能够准确框出物体并输出。
6. 要求算法和系统能够足够快，能够实时处理分析视频或者摄像头传来的数据。

# 算法原理

## YOLO\_v3网络结构

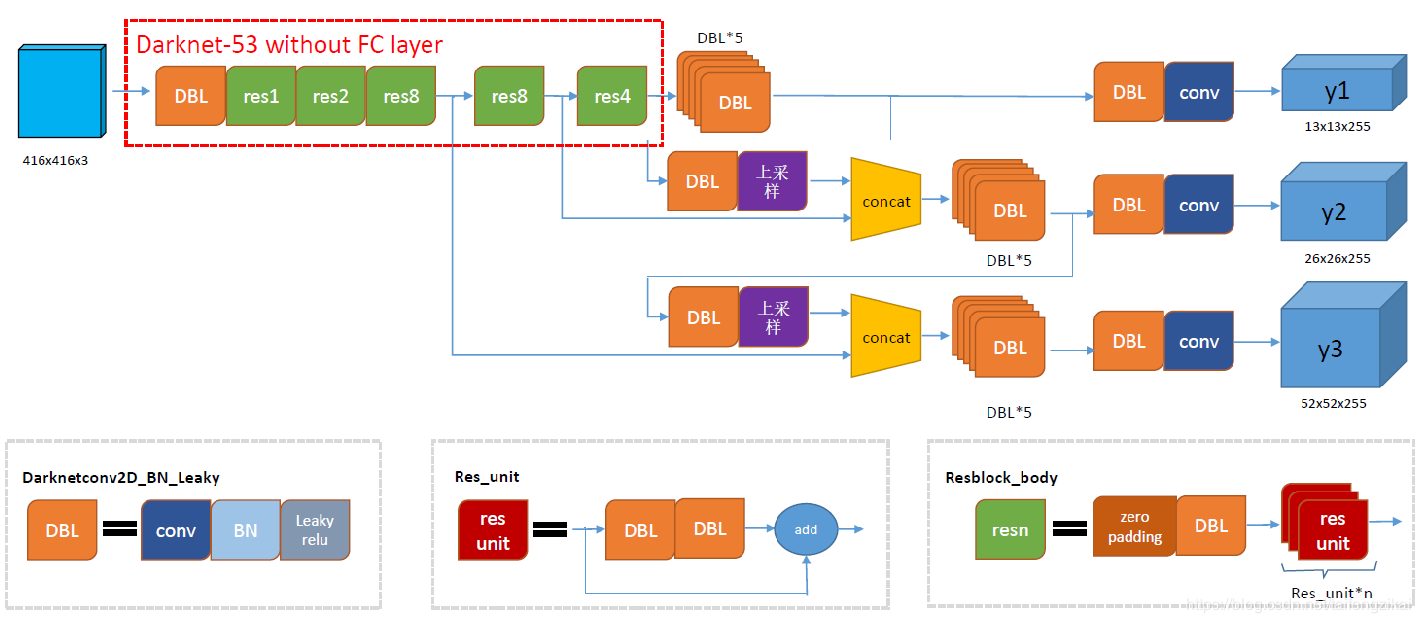


图1 Yolo v3结构图

图1中的DBL是Yolo v3的基本组件。yolo3.model中的DarknetConv2D\_BN\_Leaky函数所定义的那样，Darknet的卷积层后接BatchNormalization（BN）和LeakyReLU。除最后一层卷积层外，在yolo v3中BN和LeakyReLU已经是卷积层不可分离的部分了，共同构成了最小组件。

主干网络中使用了5个resn结构。n代表数字，有res1，res2, … ,res8等等，表示这个res\_block里含有n个res\_unit，这是Yolo v3的大组件。从Yolo v2的darknet-19上升到Yolo v3的darknet-53，前者没有残差结构。Yolo v3开始借鉴了ResNet的残差结构，使用这种结构可以让网络结构更深。对于res\_block的解释，可以在图1.1的右下角直观看到，其基本组件也是DBL。

在预测支路上有张量拼接（concat）操作。其实现方法是将darknet中间层和中间层后某一层的上采样进行拼接。值得注意的是，张量拼接和Res\_unit结构的add的操作是不一样的，张量拼接会扩充张量的维度，而add只是直接相加不会导致张量维度的改变。

## Darknet-53特征提取网络

Yolo v3中使用了一个53层的卷积网络，这个网络由残差单元叠加而成。Joseph Redmon的实验表明，在分类准确度上与效率的平衡上，Darknet-53模型比ResNet-101、 ResNet-152和Darknet-19表现得更好。Yolo v3并没有那么追求速度，而是在保证实时性(fps>60)的基础上追求更好的表现。

一方面，Darknet-53网络采用全卷积结构，Yolo v3前向传播过程中，张量的尺寸变换是通过改变卷积核的步长来实现的。卷积的步长为2，每次经过卷积之后，图像边长缩小一半。如图2.1中所示，Darknet-53中有5次卷积的步长为2。经过5次缩小，特征图缩小为原输入尺寸的1/32。所以网络输入图片的尺寸为32的倍数，取为416×416。Yolo v2中对于前向过程中张量尺寸变换，都是通过最大池化来进行，一共有5次。而v3是通过卷积核增大步长来进行，也是5次。

另一方面，Darknet-53网络引入了residual结构。Yolo v2中还是类似VGG那样直筒型的网络结构，层数太多训起来会有梯度问题，所以Darknet-19也就19层。得益于ResNet的residual结构，训练深层网络的难度大大减小。因此Darknet-53网络做到53层，精度提升比较明显。

## 3.3 利用多尺度特征进行对象检测

在YOLO3更进一步采用了3个不同尺度的特征图来进行对象检测。

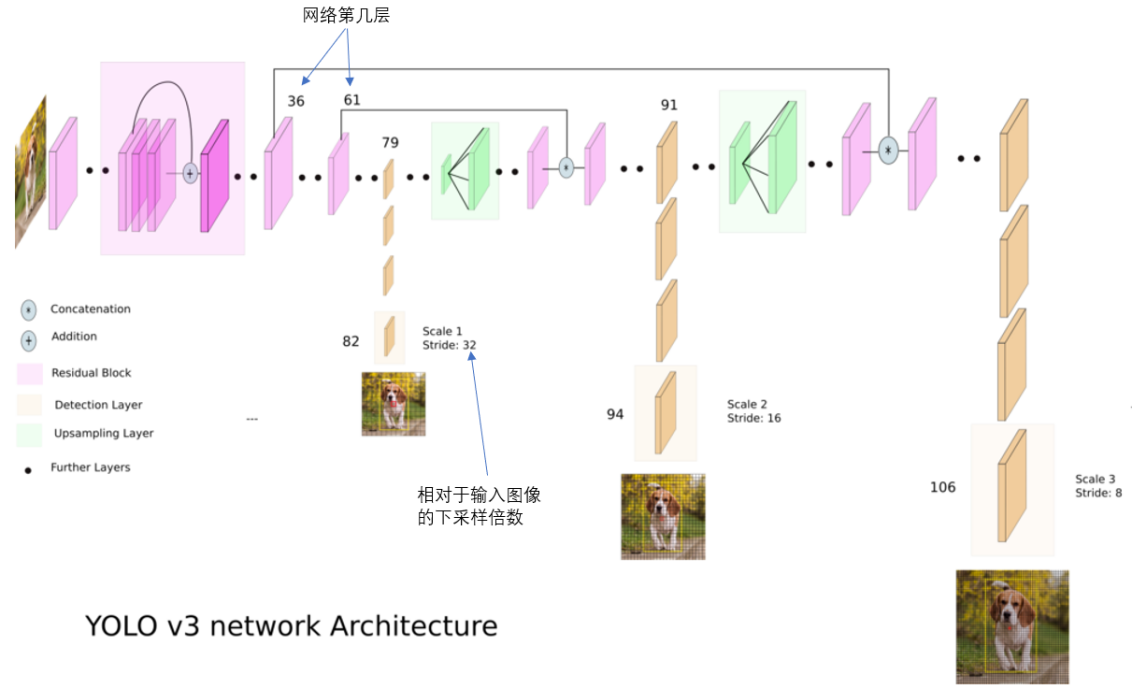
结合下图看，卷积网络在79层后，经过下方几个黄色的卷积层得到一种尺度的检测结果。相比输入图像，这里用于检测的特征图有32倍的下采样。比如输入是416\*416的话，这里的特征图就是13\*13了。由于下采样倍数高，这里特征图的感受野比较大，因此适合检测图像

图2. 多尺度预测结构

为了实现细粒度的检测，第79层的特征图又开始作上采样（从79层往右开始上采样卷积），然后与第61层特征图融合（Concatenation），这样得到第91层较细粒度的特征图，同样经过几个卷积层后得到相对输入图像16倍下采样的特征图。它具有中等尺度的感受野，适合检测中等尺度的对象。

最后，第91层特征图再次上采样，并与第36层特征图融合（Concatenation），最后得到相对输入图像8倍下采样的特征图。它的感受野最小，适合检测小尺寸的对象。

## 3.4 多种尺度的先验框

随着输出的特征图的数量和尺度的变化，先验框的尺寸也需要相应的调整。YOLO2已经开始采用K-means聚类得到先验框的尺寸，YOLO3延续了这种方法，为每种下采样尺度设定3种先验框，总共聚类出9种尺寸的先验框。在COCO数据集这9个先验框是：(10x13)，(16x30)，(33x23)，(30x61)，(62x45)，(59x119)，(116x90)，(156x198)，(373x326)。

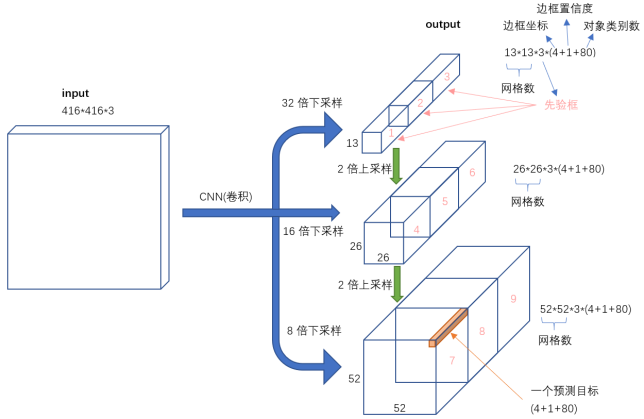
分配上，在最小的13\*13特征图上（有最大的感受野）应用较大的先验框(116x90)，(156x198)，(373x326)，适合检测较大的对象。中等的26\*26特征图上（中等感受野）应用中等的先验框(30x61)，(62x45)，(59x119)，适合检测中等大小的对象。较大的52\*52特征图上（较小的感受野）应用较小的先验框(10x13)，(16x30)，(33x23)，适合检测较小的对象。

## 3.5 输入映射到输出

不考虑神经网络结构细节的话，总的来说，对于一个输入图像，YOLO3将其映射到3个尺度的输出张量，代表图像各个位置存在各种对象的概率。

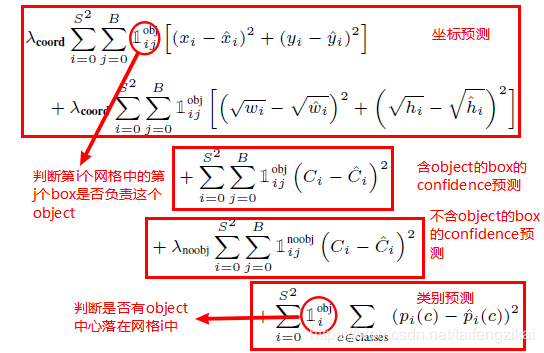
我们看一下YOLO3共进行了多少个预测。对于一个416\*416的输入图像，在每个尺度的特征图的每个网格设置3个先验框，总共有 13\*13\*3 + 26\*26\*3 + 52\*52\*3 = 10647 个预测。每一个预测是一个(4+1+80)=85维向量，这个85维向量包含边框坐标（4个数值），边框置信度（1个数值），对象类别的概率（对于COCO数据集，有80种对象（我们的进行迁移学习，所以最后只有一种火焰类别））。

YOLO3的尝试预测边框数量增加了10多倍，而且是在不同分辨率上进行，所以mAP以及对小物体的检测效果有一定的提升。



## 3.6 损失函数

在目标检测任务里，有几个关键信息是需要确定的:(x,y),(w,h),class,confidence 。根据关键信息的特点可以分为上述四类，损失函数应该由各自特点确定。最后加到一起就可以组成最终的loss function了，也就是一个loss function搞定端到端的训练。



Yolo损失函数

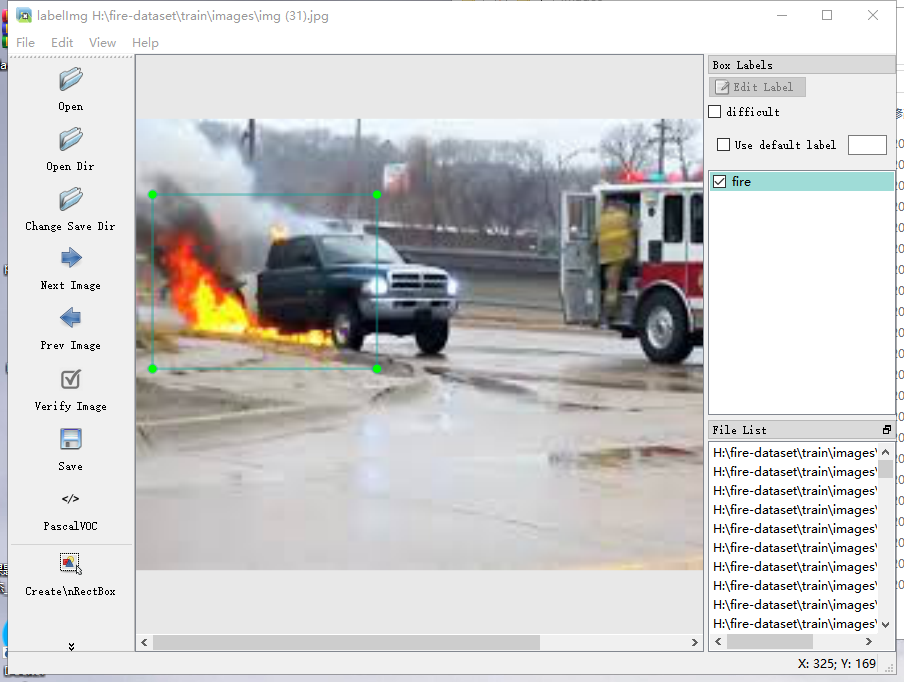
## 3.7 迁移学习

目标识别从头开始训练模型的权重代价是非常昂贵的。所以我们使用了迁移学习，用yolo3预训练好的权重，在构建我们自己的网络的时候冻结前面的网络，改变最后几层全连接层使其符合我们自己的火焰类别的识别。

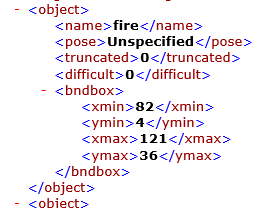
# 实现过程

## 网络爬取数据和标注数据

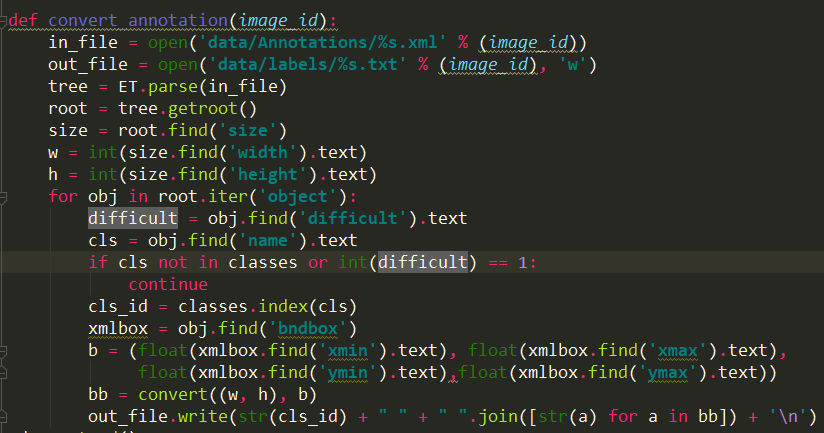
我们共搜集和爬取数据502张，其中412张用于训练，剩余90张用于交叉验证模型的好坏。使用的Scrapy框架爬取，局限于篇幅不再赘述爬虫过程（也是相当繁琐）。接下来要标记数据，使用的Labelimg工具进行标注（Github上可以下载）

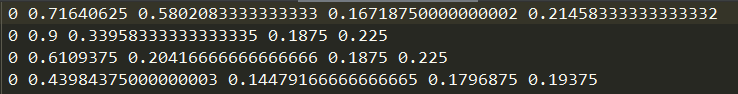


如图所示对每一张图片进行标注，格式为PascalVOC格式。标记完所有数据后，每个图片的标记会以xml文件保存，其中包括了物体类别（fire）,检测框的左上角和右下角的像素坐标。



经过voc\_label.py后面处理xml文件里的信息能够被提取出来，并存储为模型可以直接使用的label文件。





Label文件

Label文件中分别为（类别 物体中心x坐标 物体中心y坐标 边界框宽比例 边界框高比例）这正是模型需要的几个数据。至此所有数据标签和数据准备就完成了。

## 4.2 模型的构建

代码主要分为以下几个部分

4.2.1 Parse\_config.py

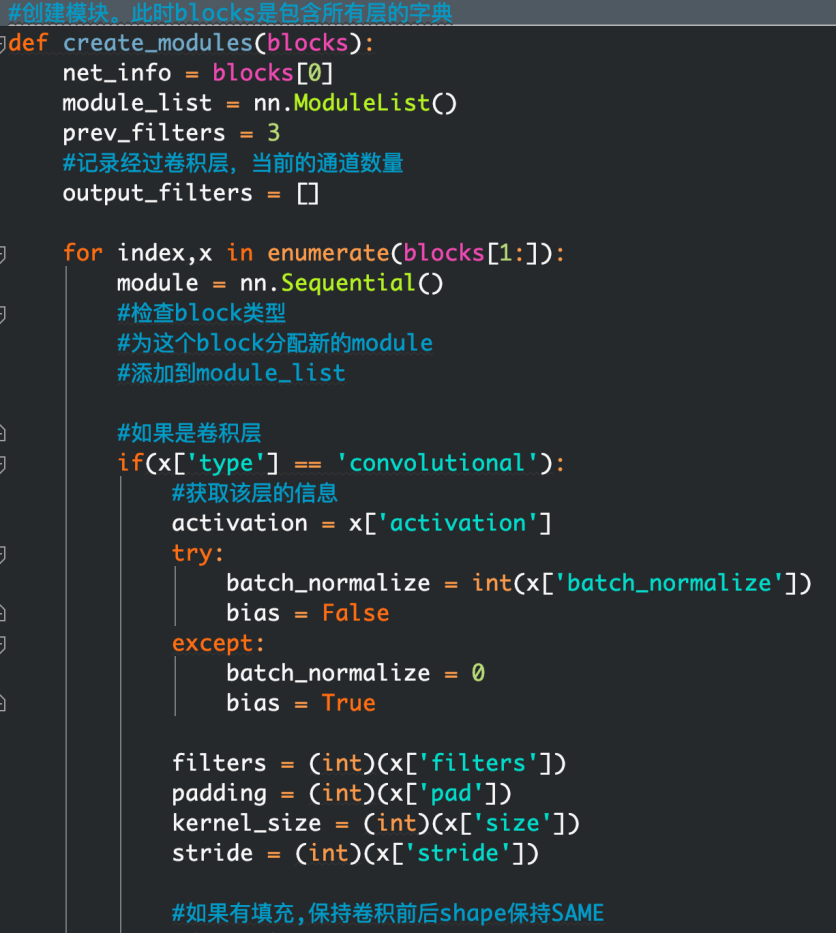
-parse\_config(config):

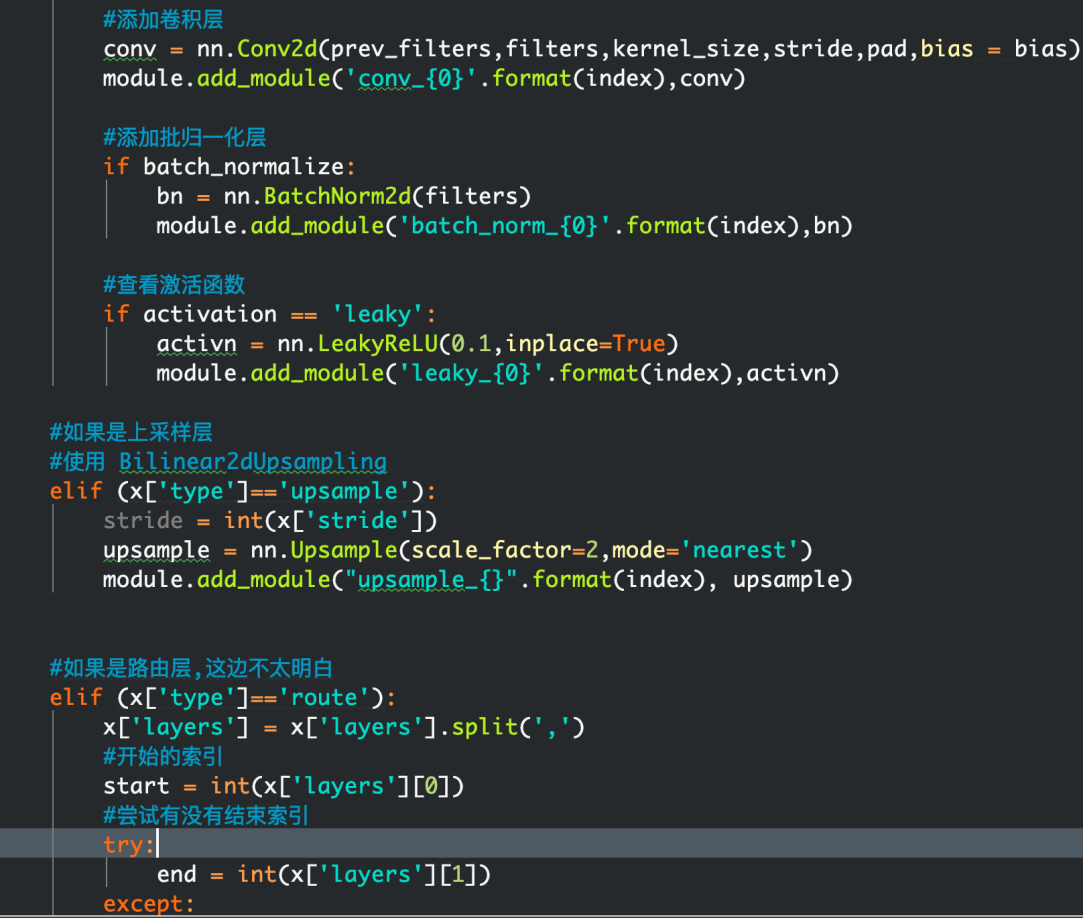
解析模型配置文件

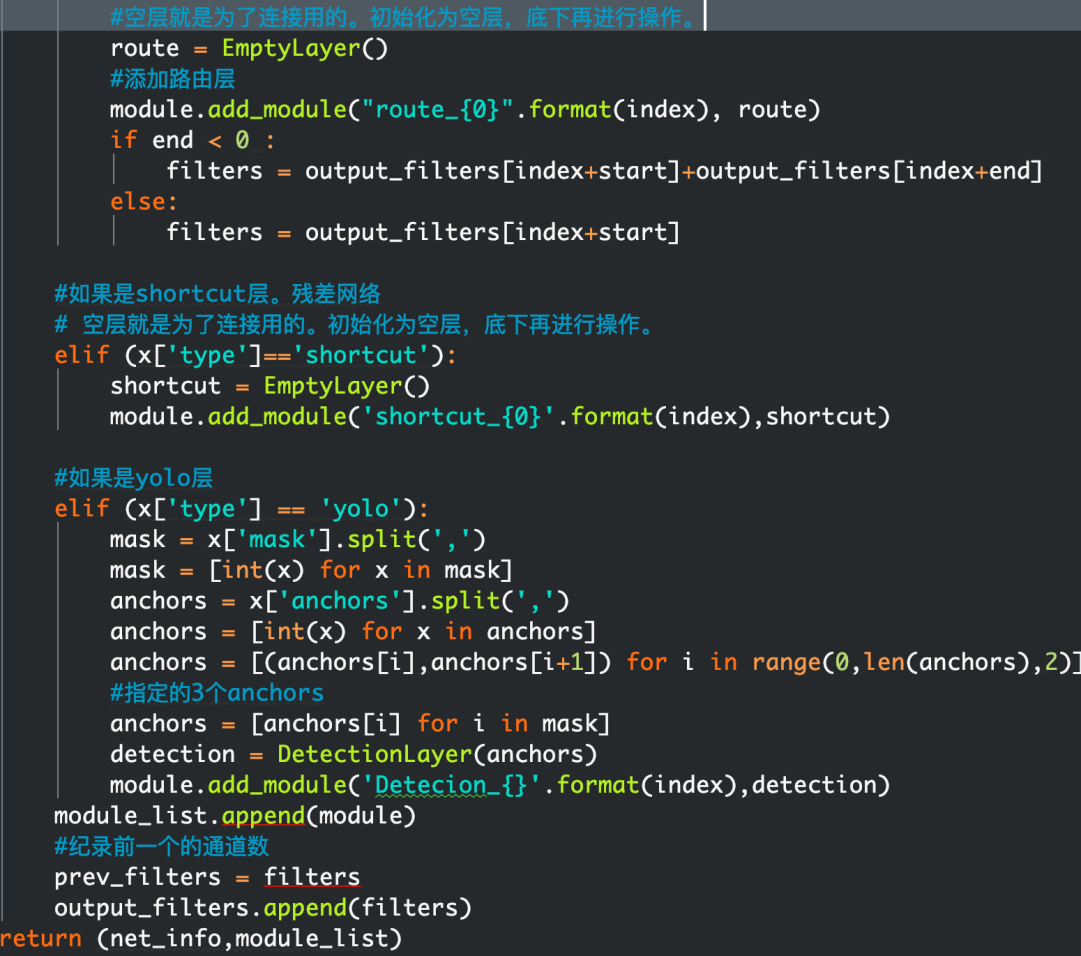


-create\_modules(blocks)

解析配置文件。并创建Module\_list字典，字典中包含了每个模块的类别，例如是卷积层还是路由层还是残差网络或者yolo层。

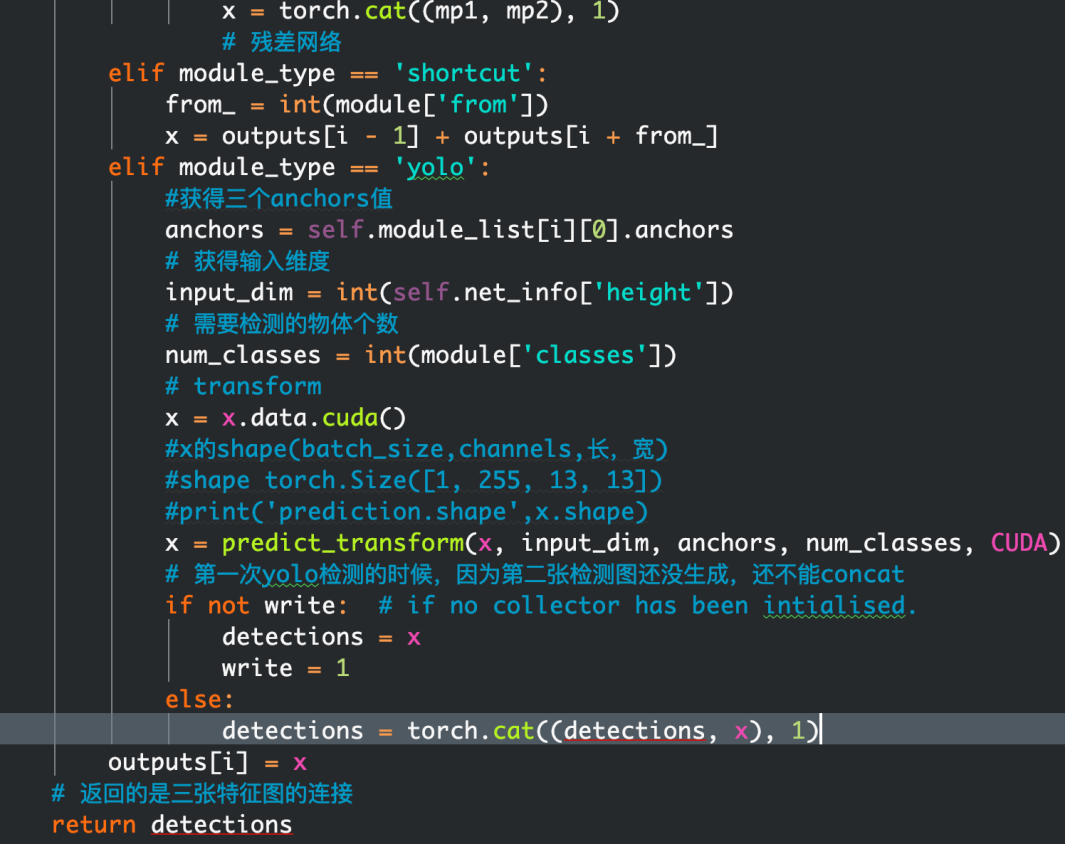




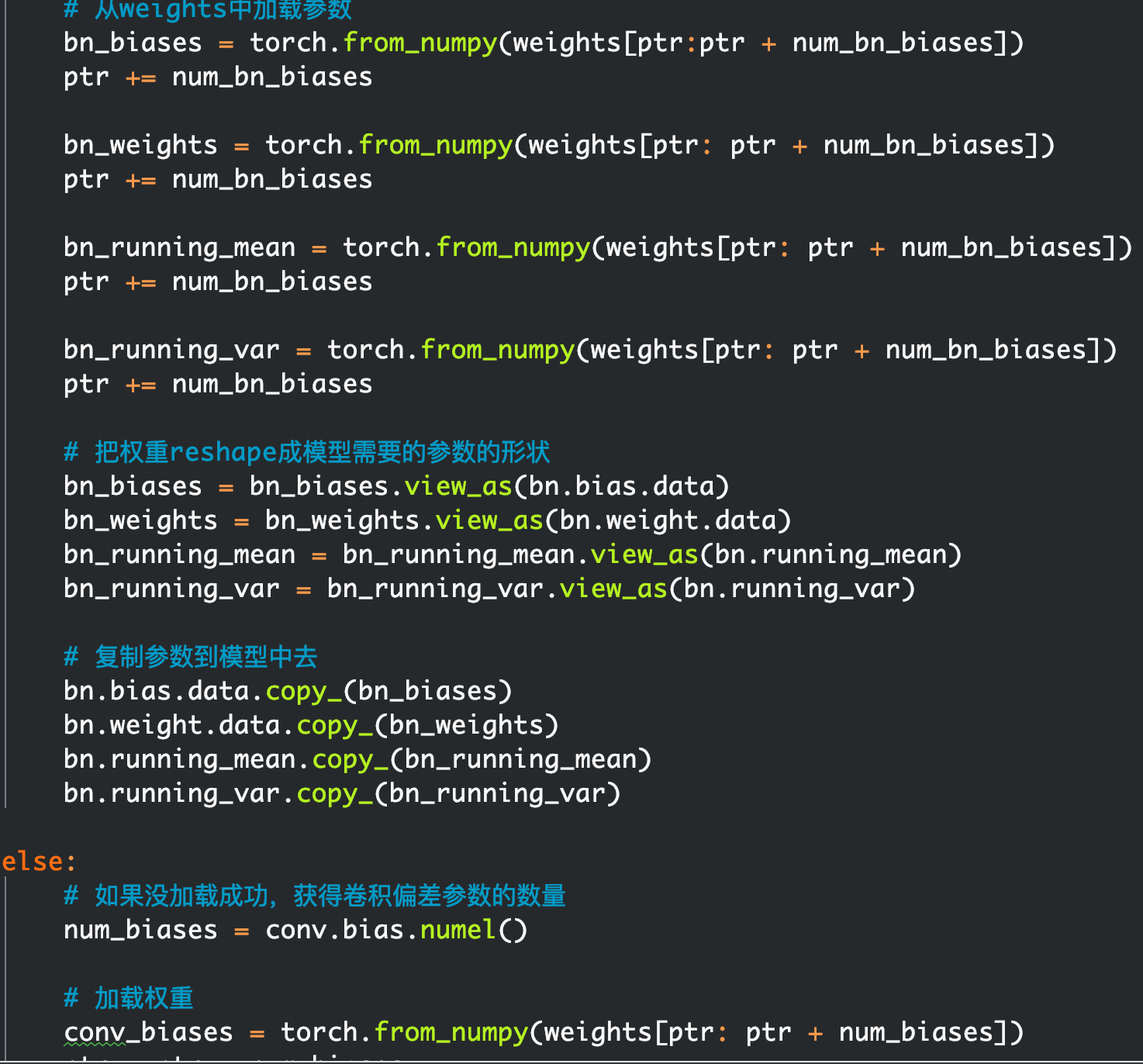


4.2.2 DarkNet.py 创建网络，并定义前向传播。返回的是三张不同尺度特征图按照通道连接。返回模型检测的结果向量。

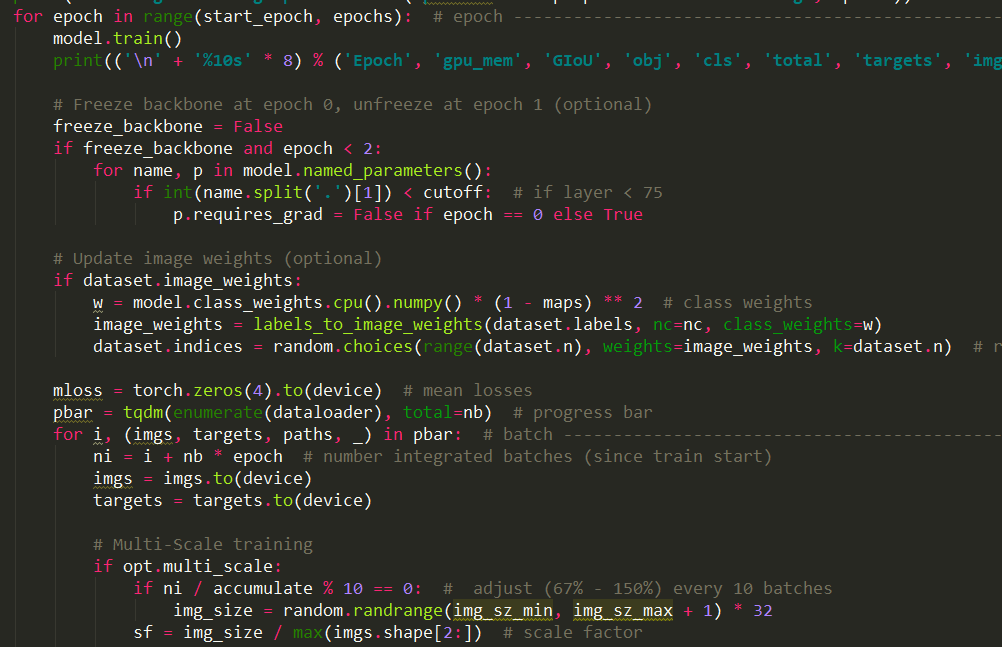


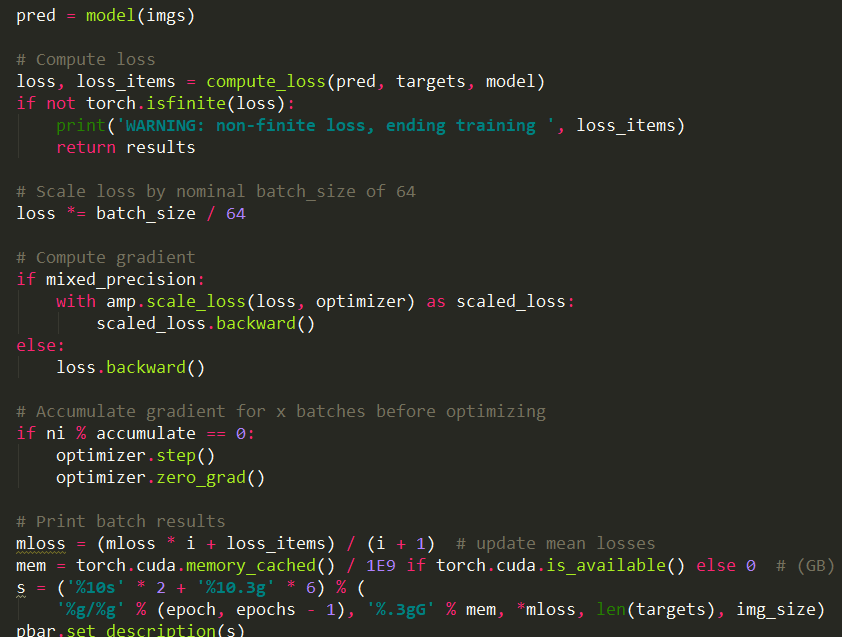


按照存储顺序加载权重。



## 4.3 模型训练





模型训练我们使用了torch的混合精度训练，相当于实验5中tensorflow的量化模型，把float32转化为float16，显著减小了gpu显存的开销和模型权重文件的大小。我们的指标是map（均值平均精度，即MAP）。经过300轮迭代，最终map能够到达70%左右，已经是相当不错了。（不同于accuracy，acuuracy可能要到95%及以上就非常好了。）

## 4.4 火焰检测

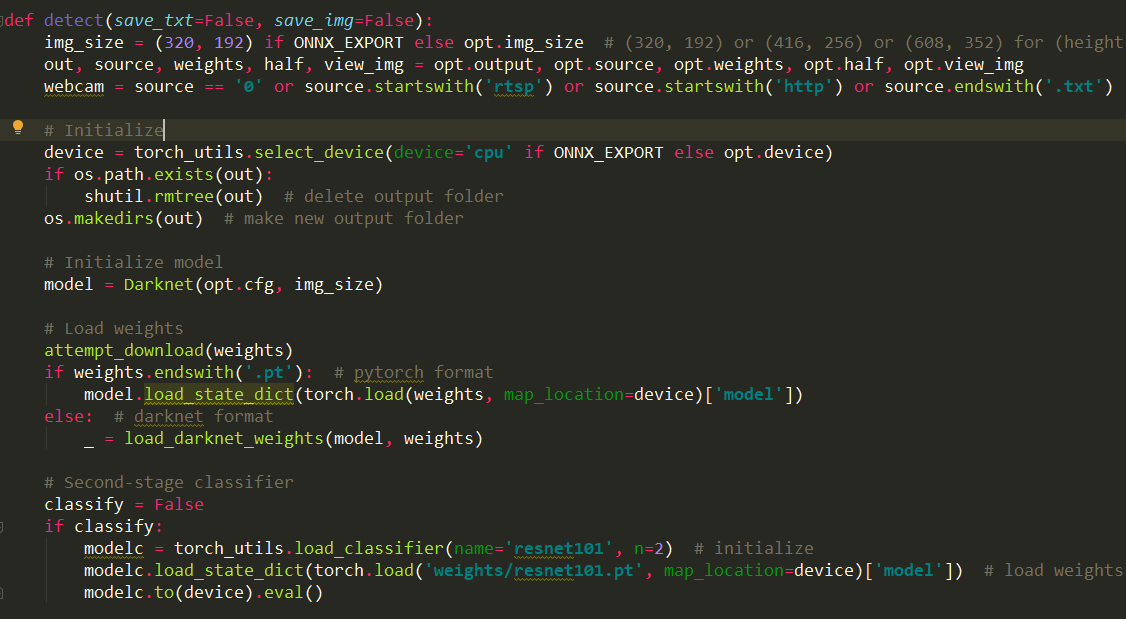
-arg\_parse进行参数设置。设置需要的文件路径，批量大小，目标的置信度阈值，非极大值抑制的阈值，或者需要检测的类别图片视频或者摄像头。

用cv2.VideoCapture(arg)能识别不同的类别，如果需要检测视频，arg为视频位置。如果要实时检测摄像头中的数据，那么arg=0为检测摄像头。



Detect\_images 对设置的文件目录下所有的图片进行检测和画框处理，并把处理的图片存储到目标文件夹下。

其中def detect（）函数为核心的检测函数。



## 4.5 构建基于Flask的实时火焰检测网站

### 4.5.1 概述

我们需要通过Http协议以较低延迟呈现给浏览器实时监控视频。通过Http协议传输视频常见有HTTP-FLV方式和MJPEG方式。HTTP-FLV协议将音视频数据封装成FLV, 然后通过 HTTP 协议的Chunked-Encoding传输给客户端，理论上（除去网络延迟）可以做到一个音视频 Tag 的延迟。而该Project采用实现起来更为简单且延迟较低，但以增加带宽占用为代价的MJPEG流方式传输影像；即将摄像头每帧输入转为jpeg编码图像，通过Http Multipart字段传输每一帧的jpeg图像，浏览器收到后通过<image>标签显示，不断重复获取每一帧并刷新。

### 4.5.2 Http Multipart字段

MJPEG流主要依靠Http Multipart字段传输，该字段被绝大部分浏览器支持。Multipart字段由RFC1341定义，具有多种不同的类型，针对流媒体，使用multipart/x-mixed-replace。Multipart将一系列数据块组合在一个Http Body里，并将HTTP头部的Content-Type字段指定为multipart；每个Http Body包含至少一块数据，块与块间用boundary分隔，最后一块末尾为一个closing boundary，boundary由程序指定，通常为一定长度的随机字符串，只要不容易与数据内容重复即可。浏览器收到这种Multipart类型时，会使用当前块的数据替换先前块的数据，由此画面内容将被逐帧显示出来。只要服务器不主动关闭连接，浏览器会源源不断地获取并显示每一帧图像，由此形成一个连续的视频流。

服务器通过HTTP Multipart字段向浏览器传输MJPEG流的一个典型HTTP响应例子如下（仅列出关键信息）:

HTTP/1/1 200 OK

Content-Type:multipart/x-mixed-replace; boundary=KSoRhZx7t8UKsx8po39G6lz1oLd

--KSoRhZx7t8UKsx8po39G6lz1oLd

Content-Type:image/jpeg

Content-Length:436312

[image1 enconded jpeg data]

--KSoRhZx7t8UKsx8po39G6lz1oLd

Content-Type:image/jpeg

Content-Length:473253

[image2 enconded jpeg data]

...

其中”[image x encoded jpeg data]”表示经过编码后的jpeg图片数据，图片间分隔符为程序随机生成的字符串，例如“KSoRhZx7t8UKsx8po39G6lz1oLd”。

### 4.5.3 代码实现

指定http协议所需要的“multipart/x-mixed-replace”MIME类型，当浏览器uri为/video\_streamer/<cam\_index>时产生该Response。

@app.route('/video\_streamer/<cam\_index>')

def video\_streamer(*cam\_index*):

*# A stream where each part replaces the previous part the multipart/x-mixed-replace content type must be used.*

*return* Response(jpeg\_gen(),*mimetype*='multipart/x-mixed-replace; boundary=frame\_boundary')

利用Python提供的Generator语法每次生成一个数据帧，包括Header和Payload。

def jpeg\_gen():

*while* True:

jpeg=fire\_detector.get\_jpeg()

*# Builds 'jpeg' data with header and payload*

*yield* (b'--frame\_boundary\r\n'

b'Content-Type:image/jpeg\r\n\r\n'+jpeg+b'\r\n\r\n')

### 4.5.4 性能与优劣

由于MJPEG只采用帧内压缩，编解码算法简单，故节约硬件资源[21]。压缩后除了JPEG自身的损失，没有帧间编码带来的画质损失。经测试，局域网条件下传输MJPEG流用户几乎感觉不到延迟（延迟≤0.2s），如果在公网传输，同一个城域网下延迟一般在1秒左右，实时性较好。

缺点主要为MJPEG编码方式带来的对视频压缩效率较低，网络传输占用带宽较大等负面影响，720p中低质量MJPEG流带宽占用约为6.4 Mbps。

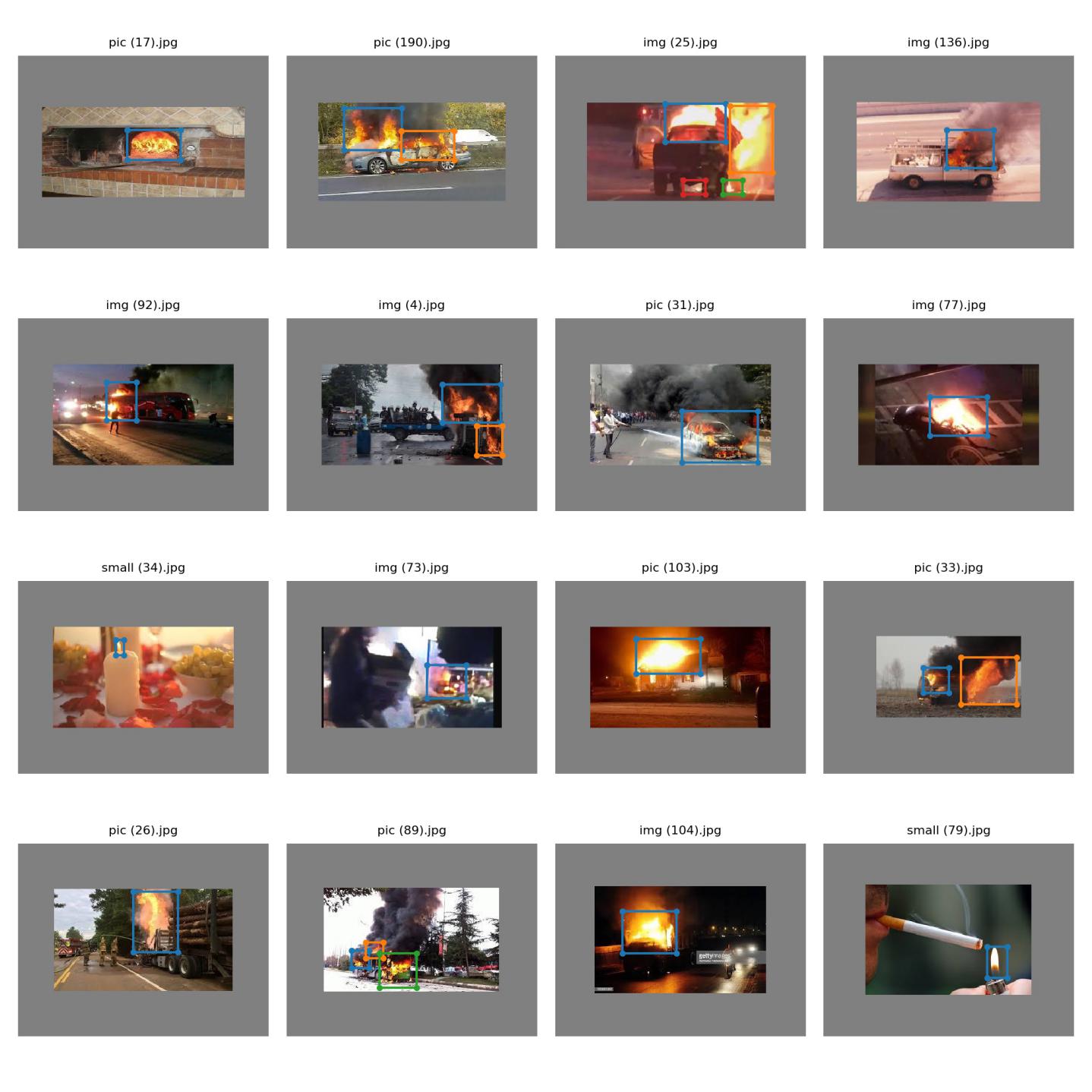
### 4.5.5 Flask线程与YOLO线程交互

YOLO线程读摄像头帧进行火灾检测并将处理后的帧画面写入一个帧全局变量，Flask线程处理网络连接并读该全局变量，将识别后的画面实时传输出去；帧全局变量使用Python提供的Condition和RLock进行同步。Flask在收到Stream请求后开启一个新线程来推流，它会阻塞在Condition上，当YOLO处理完一帧后唤醒它，让其将这一帧画面包装成HTTP响应发送给浏览器。

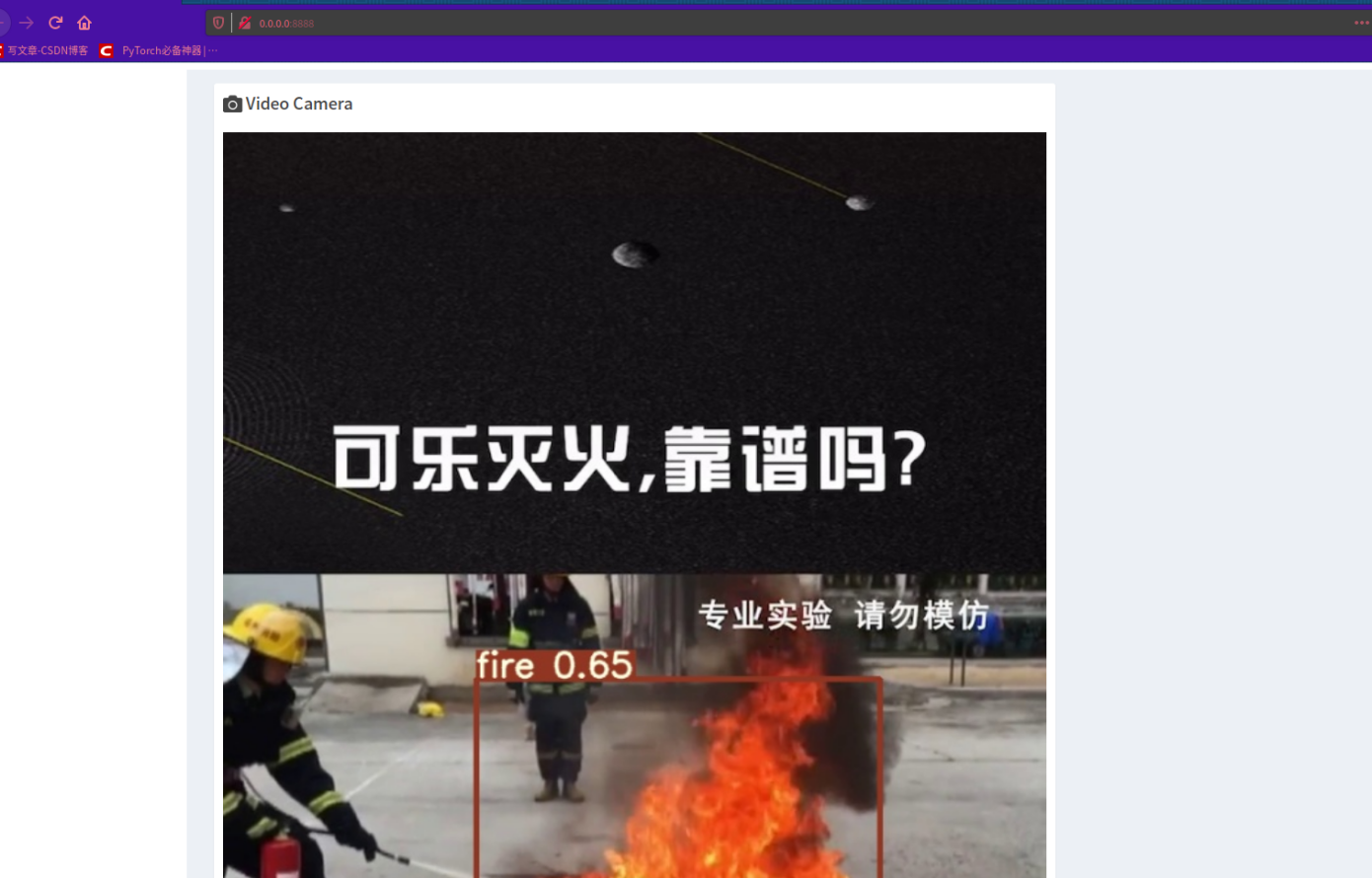
# 结果分析

经过不断优化，模型能够比较完美的检查到目标并能够确定目标的位置。

## 图片检测效果



## 5.2 Flask构建的网站及实时检测



## 5.3 视频文件

请见文件夹中demo.mp4文件。能够实时并且准确地识别火焰，能够有效预防火灾。

# 参考文献

You Only Look Once: Unified, Real-Time Object Detection; Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi;The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788

YOLOv3: An Incremental Improvement; [Joseph Redmon](https://arxiv.xilesou.top/search/cs?searchtype=author&query=Redmon,+J), [Ali Farhadi](https://arxiv.xilesou.top/search/cs?searchtype=author&query=Farhadi,+A); arXiv:1804.02767v1 [cs.CV] 8 Apr 2018

# 分工合作

* YOLO\_v3的网络构建） 李亘杰 周渝茗
* 训练集的搜寻和数据标注 周渝茗
* 算法的细节实现（预测向量的转化，反向传播的调节） 李亘杰 周渝茗
* 模型的冻结，训练，调参 杨雨丰 李亘杰
* 图片目标检测的测试 杨雨丰 周渝茗
* 视频的实时检测 李亘杰 杨雨丰
* Flask构建实时火焰检测网站 杨雨丰