模式识别创新研修课课程总结报告

1170400423尉前进2020.6.16

YOLOV3目标检测算法解读以及应用

一.解读代码流程部分

从train.py看起，比较容易些

首先是获得网络需要的y\_true的过程，因为ground\_truth不是直接被使用的，它需要找到Anchor,变成y\_true的格式才能被真正用于网络训练；

1.首先定义训练数据生成器函数：(train.py\_中)



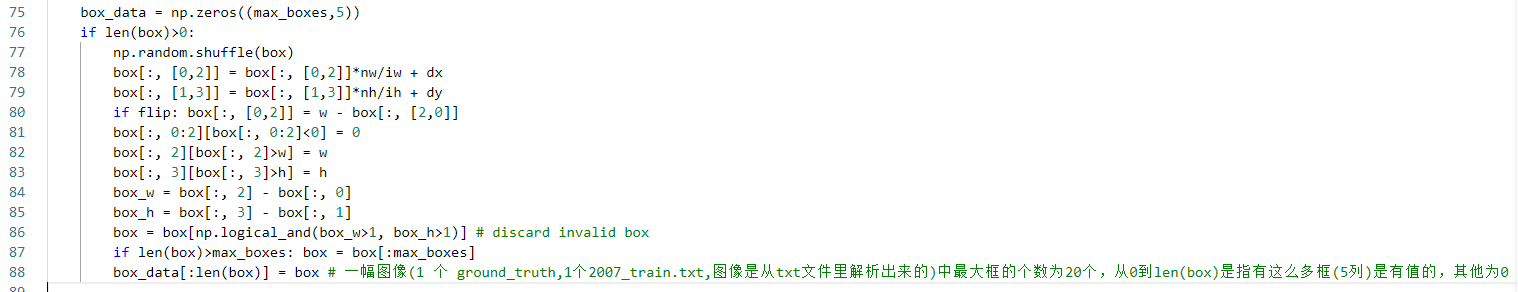
在训练数据生成器中

**第一个重点**是进行数据增强处理，**做数据增强的意义在于每一个epoch训练的时候，输入的信息都是有差别的。**相当于扩充了数据集。(在utils.py中)

在数据增强处理中，先把图像resize成标准的416\*416形式，**再做图像随机翻转，图像扭曲**；最后做图像像素的归一化处理。

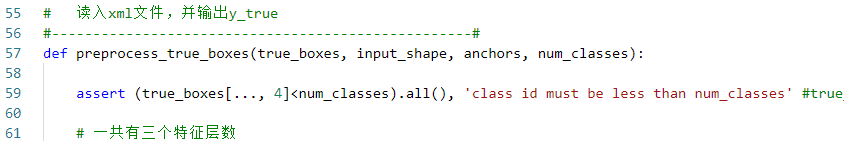


由于图片被resize成标准形式，**它对应的框也要变成标准形式**；代码如下：

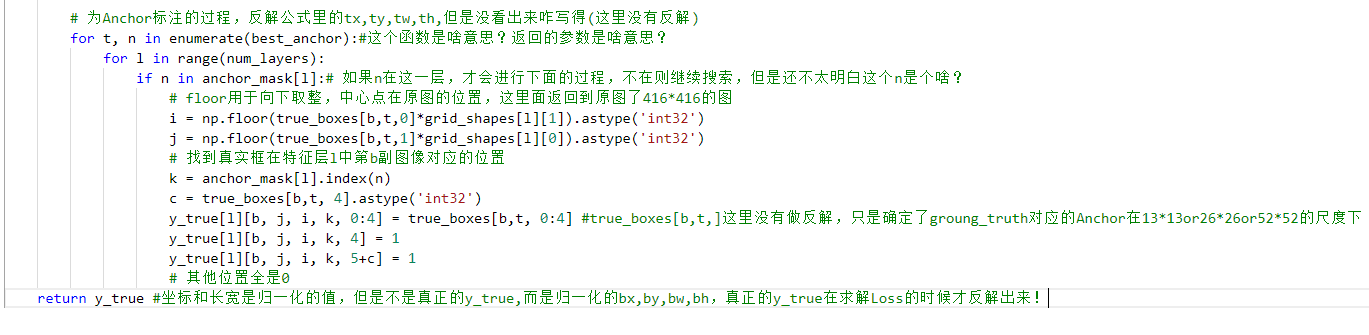


**第二个重点是编码过程，即得到y\_true的过程**

对应的函数为：(在train.py中)



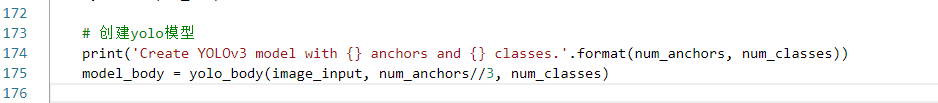
* xml标注的格式是矩形主对角线上两点的坐标值，首先将其转换成矩形的中心点坐标和宽高信息；
* 然后再除以416变成归一化的坐标格式
* 对Anchor和修改过的ground\_truth(即未经过归一化的宽和高)进行处理后，计算真实框和哪个Anchor的IOU最大，就取这个Anchor,这里计算的时候不分层，等确定了Anchor后便通过enumerate函数得到这个Anchor位于哪一层
* 最后，得到归一化的bx,by,bw,bh,但是反解的tx,ty,th,tw在Loss中才求解得到



到此，编码过程结束，得到了真正的标签 y\_true.（没有反解的，不算真的。2020.6.15修改）

1. 接下来便是训练过程中的前向传播过程

首先在train.py中看到加载Yolo模型的地方，这便是Model的搭建过程



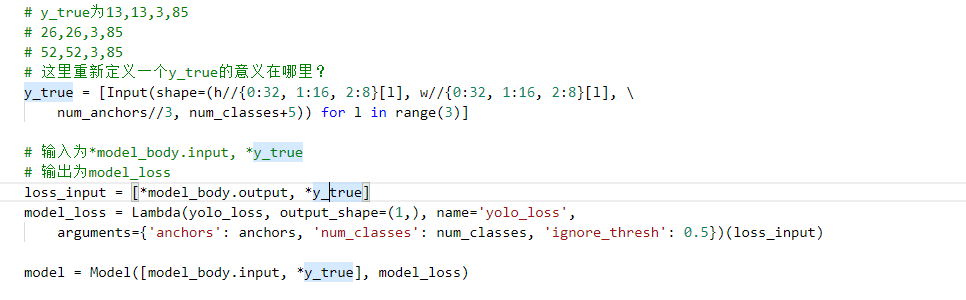
然后，一切关于训练中前向传播的过程必然在yolo\_body中（yolo3.py)中

分析过程如下：



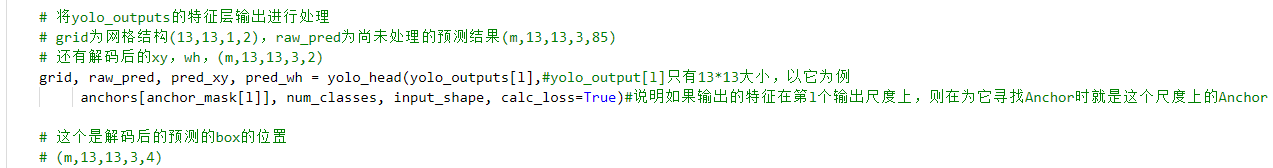
输入的值是416\*416，返回的结果是一个模型Model(Model(input,[y1,y2,y3]),其中y1,y2,y3为我们最后需要的3个尺度下的输出，它的格式与y\_true是一致的。因此便可以计算Loss了

然后按着代码读下去，就到了



然后我们去寻找yolo\_loss这个函数，看如何计算Loss,在(loss.py)中；

得到y\_pre的过程：



在yolo\_head()函数中，返回解码的值

yolo\_head()中，有两个作用，在训练时和在预测时的返回值不同

在训练时返回的是：

1.grid

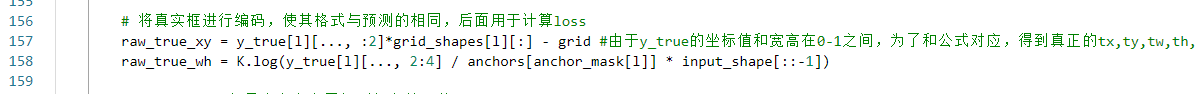
2.网络输出的feats经过了一次reshape,变成与y\_true一致的格式，便于后面求解Loss，但内容没有发生改变，还是tx,ty,tw,th,这几个参数

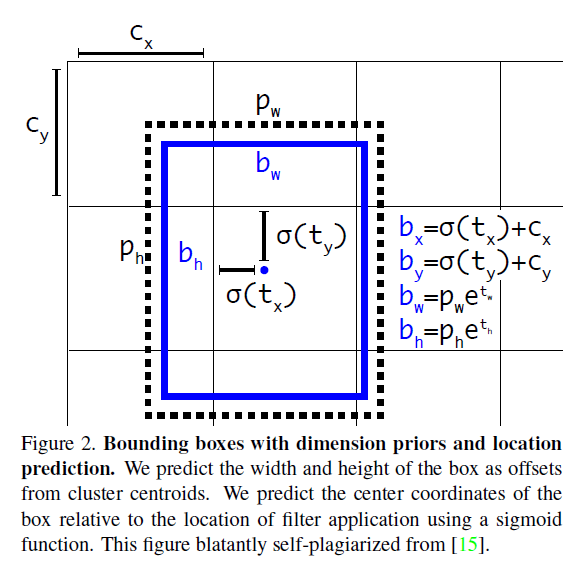
3.还要返回bx,by,bw,bh这几个经过论文公式转换的值，在后面会用到



接下来便是**用公式反解**真正的y\_true:

代码如下：





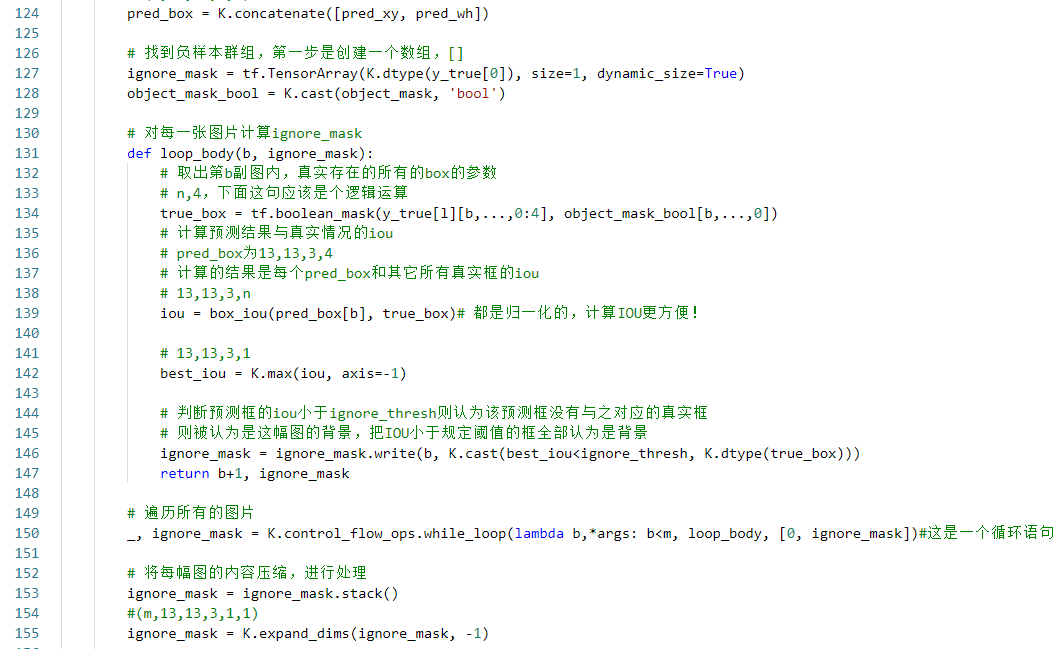
这里面虽然没有的反函数，但是它**求解出来的的值就是在0-1之间**的（经过实际计算发现就是这样的）而grid就是论文中的，grid是一个具有两个值的列表；

而对于宽和高，即,它们的值应该是原416\*416图像中的值反解出来的，而不是归一化的值反解出来的，但是我们之前得到的y\_true所有值都被归一化了，因此先乘以416，再利用论文里的公式反解得到，至此，得到了真正的，然后网络的输出要逐渐逼近这几个值，得到最优的，再由公式得到最优的，但是需要注意的是不光在0-1之间，**而且也被归一化到了0-1之间，代码是这么写的，可能是为了与y\_true保持一致，便于后面在训练的过程中计算IOU**,这几个最优的值是在测试过程中返回的，即yolo\_head的另一个作用；它在测试的时候又被调用(yolo3.py中),

这个在测试的时候再具体分析；

再解码后（调用yolo\_head()以后），便可以得到一个真正的bounding\_box(bx,by,bw,bh),虽然在坐标预测值的回归过程中用不到，但是在类别Loss的计算中要用到，下面具体分析：

如图：

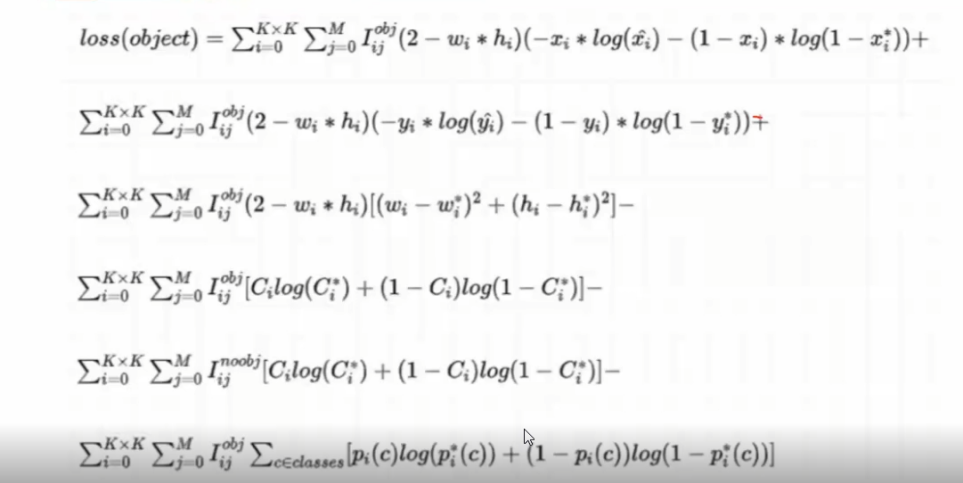


它返回了一个ignore\_mask,大致意思就是说把那些有中心点落在区域范围内，但是Anchor与bounding\_box的IOU太小的，认为它也是没有物体的，是背景框，这一步在计算是否有物体的Loss中得以体现，具体作用还不清楚。

接下来便是Loss\_Function的构建过程。代码如下：



而Loss\_function的计算公式为：



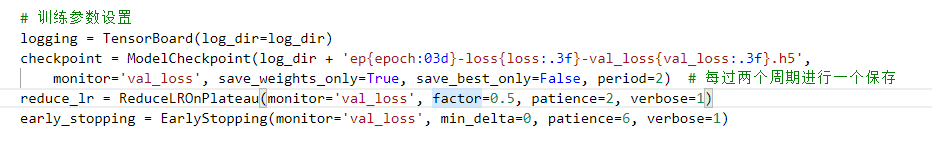
可见，代码构建的这个有些许不同，具体标准的是哪一个，还需要证实

1. 虽然在坐标预测的过程中这是一个回归问题，但是却用的是交叉熵损失函数，因为它的值是在0-1之间的，所以可以用，但是**为什么要用交叉熵损失函数代替均方误差损失函数还不清楚**
2. 在类别的Loss\_Function中也用交叉熵损失函数而代替Softmax损失函数，是因为一个框可以预测多个类别，属于多标签分类？不理解这块儿。

至此，Loss\_Function也构建完毕；

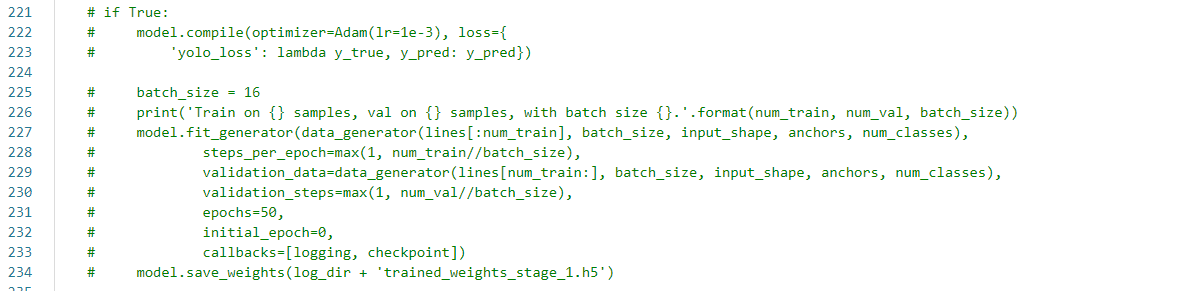
然后利用迁移学习的思想，加载CoCo数据集的预训练权重，先对前249层（为什么）冻结，只训练后几层，等到Loss不再下降了，就解冻全部训练

关于训练参数的设置：

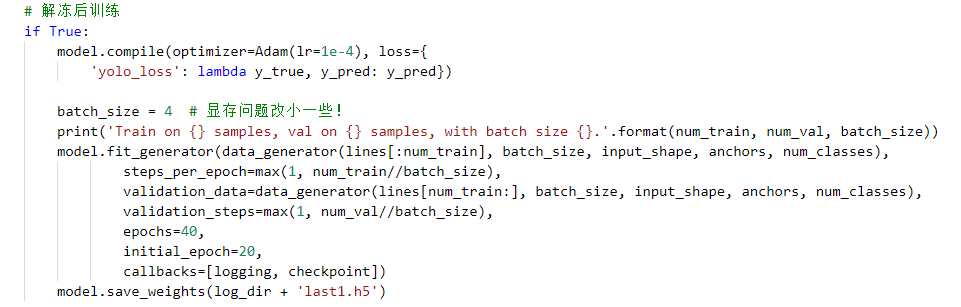


Loss不下降，就减小学习率；一直不下降，训练就提前停止

关于学习率、epoch、Batch\_size的设置：

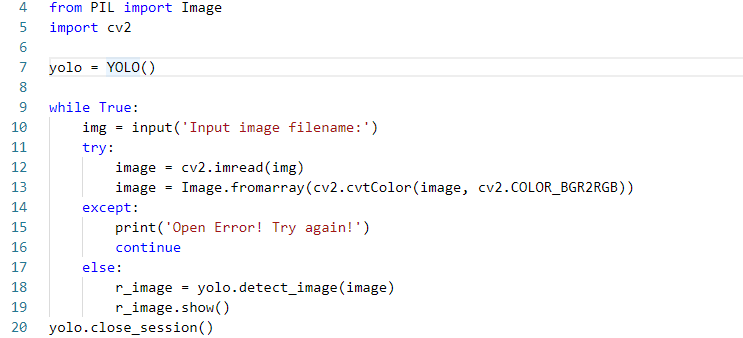


解冻之后的设置：



然后，就是预测部分：

首先，要看的当然是predict.py,在里面发现关键是调用了yolo这个类，所以我们去寻找它，（在yolo.py中）



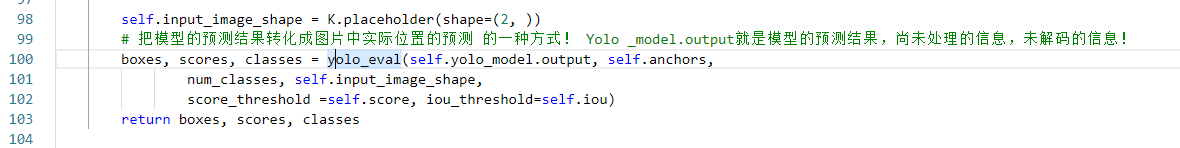
首先看一下它的配置默认值：



然后，下面就是一堆初始化和数据处理过程；不是很重要

但是网络的输出仍然需要经过解码，才能最终转换为我们想要的框和结果：

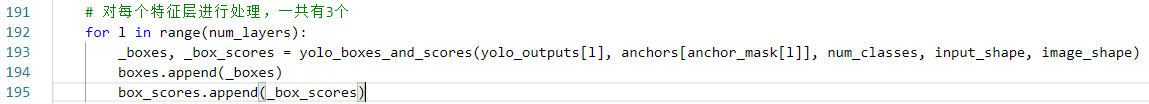
在里面有一个很关键的函数：yolo\_eval()(在yolo3.py中）

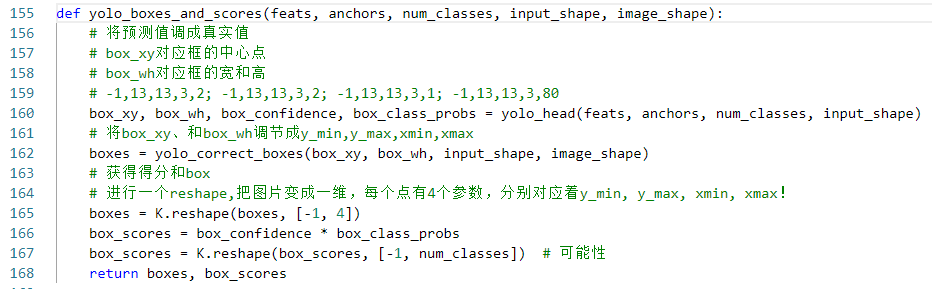


它返回的是得分、框（bbox）、类别

分别看一下它们怎么得到：

对于boxes、scores 通过 yolo\_boxes\_and\_scores()函数得到





可以看到，得分相当于是一个条件概率；

而boxes则来自于另外一个函数：



这里还存在一些问题未解决

最后，得到boxes、score、classes再进行一个NMS，便得到了最终的结果。

1. 应用

口罩检测实战

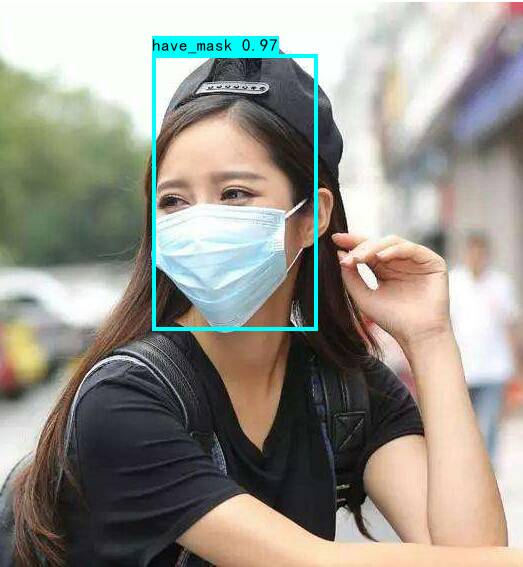
数据集：

共513张正样本，651张负样本，相当于是一个2分类的目标检测的应用

策略:由于这些口罩都在人脸上，所以，如果单纯引入正样本，很可能会导致模型学习到一部分人脸的特征，所以需要将不带口罩的人脸作为负样本，增强模型的识别能力

最后的Loss在10左右

下面给出检测结果：

下一步计划：

1. 用轻量化的模型提高检测帧率
2. 进一步理解YoloV3算法原理
3. 尝试用单类物体的YoloV3目标检测