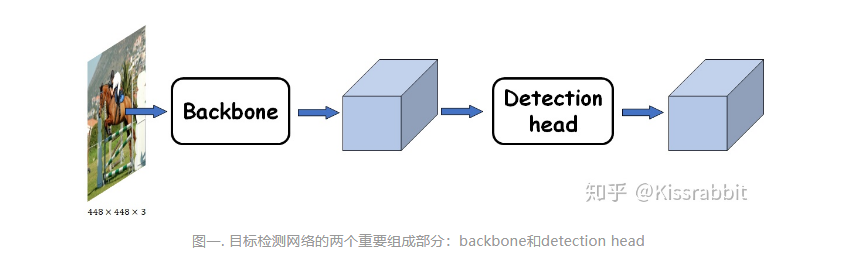
# 相关基础知识：

## 目标检测网络的基本结构

目前对于目标检测结构大家对它的结构划分就类似于人的骨骼结构那样划分的，脊柱backbone-脖子neck-头head

### Backbone

Backbone是骨干的意思，在目标检测中是指用于提取特征的前面部分的神经网络。



深度学习技术之所以能够这么出色地完成图像分类任务，基本上体现出了其在图像特征提取这一块的出色表现。另外，考虑到目标检测任务中，也需要对图像中的物体进行类别的识别，因此，一个很直接的想法就是将图像分类的网络直接拿来做目标检测的任务。

随着迁移学习概念的兴起和普及，通过在检测任务中的数据集上对分类网络进行微调似乎是一个很不错的想法，所谓的“ImageNet pretrained model”概念就诞生了。简单地来说，就是目标检测和图像分类这两个任务具有一定的相似性，因为可以将分类的网络，比如VGG、ResNet等，用来做特征提取器。

不过目标检测除了需要分类，更重要的一点是在于定位，而这一点恰恰是做分类任务的网络所不具备的。

那么该怎么办？

### Head

解决的办法相当简单，既然仅仅靠分类网络是不行的，那么我们就在后面加一些网络层，让这些额外加进来的网络层去弥补分类网络无法定位的先天缺陷。于是，这条脉络就非常清晰了：分类网络迁移过来，用作特征提取器（通过在OD数据集上进行微调，并且与后续的网络的共同训练，使得它提取出来的特征更适合OD任务），后续的网络负责从这些特征中，检测目标的位置和类别。那么，我们就将分类网络所在的环节称之为“Backbone”，后续连接的网络层称之为“Detection head”。Backbone也译作脊梁，然后在上面接一个头head，这样理解起来也是比较合理的。

### Neck

随着技术的发展，除了backbone和head这两部分，又增加了一个模块，用于特征融合的neck，详细内容见下面的部分

## 特征融合

### 基本概念

特征融合是指将CNN不同深度学习到的特征融合起来进行预测。因为CNN低层特征具有更好的位置，细节信息，分辨率更高，但其特征抽象度不够高，即语义性更弱。而CNN高层特征语义性更强，但是丢失了细节信息，对细节感知能力差，分辨率更低。

这个很容易想到，假如我们简单描述一张图片，一只黄色的站着的狗，但是这里面丢失了更细节的信息，狗的四条腿在图片中的位置，姿势等等。

这种情况改怎么办，想办法将高低层的特征结合起来考虑，我们称之为**特征融合**。怎么结合呢，一种简单的方法，最后的全连接层将底层特征图和高层特征图合起来作为输入。这一类方法就是特征的早融合，就是先将特征融合起来，然后进行预测。还有融合的方式吗，那就是用最后的预测器分别处理底层特征和高层特征，然后综合考虑两者的输出。这称之为**晚融合。**

早融合两种方法：

（1）concat：系列特征融合，直接将两个特征进行连接。两个输入特征x和y的维数若为p和q，输出特征z的维数为p+q；

（2）add：并行策略，将这两个特征向量组合成复向量，对于输入特征x和y，z = x + iy，其中i是虚数单位。

晚融合的两种代表思路：

（1）feature不融合，多尺度的feture分别进行预测，然后对预测结果进行综合，如Single Shot MultiBox Detector (SSD) , Multi-scale CNN(MS-CNN)

（2）feature进行**金字塔融合**，融合后进行预测，如Feature Pyramid Network(FPN)等。

**（**我**）**和resnet短连接的区别：有点疑问的是-短连接似乎也把低层的特征x输入上去了呀？不是的，应该是x应该已经变化了，并且短连接只是服务于高层特征，让高层特征更好，不能让最后的预测器获取到准确的低层特征。

自然地融合的时候，不是只考虑某一个低层，而是所有低层都按照一定的重要性占比进行考虑。通过一定的方式组织起来，就会很像金字塔结构。下面是经典的FPN结构。

Ps

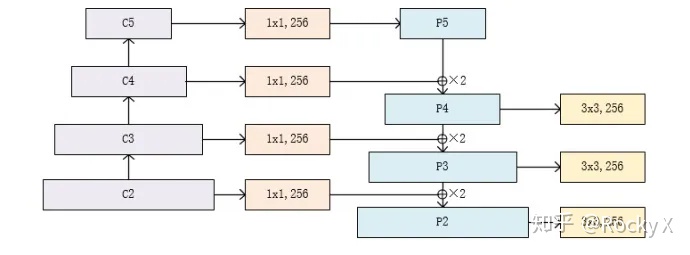
目前的话，neck这一部分的研究点还是相对来说比较多的，只需要提出一个好的结构或模块，加到现有的sota的网络中，看涨不涨点，涨点了，那就可以发文章了~。

还有一个地方，特征金字塔使用不同分辨率的特征图会便于检测不同大小的目标，尤其小目标，要使用前面的高分辨率的大的特征图才行，因为高层特征感受野可能很大，对应的框会直接越过目标。

### FPN

FPN（Feature Pyramid（金字塔） Network）算法同时利用低层特征高分辨率和高层特征的高语义信息，通过融合这些不同层的特征达到预测的效果。并且预测是在每个融合后的特征层上单独进行的，这和常规的特征融合方式不同。

FPN将深层信息上采样，与浅层信息逐元素地相加，从而构建了尺寸不同的特征金字塔结构，性能优越，**现已成为目标检测算法的一个标准组件**。FPN的结构如下所示。



自下而上：最左侧为普通的卷积网络，默认使用ResNet结构，用作提取语义信息。C1代表了ResNet的前几个卷积与池化层，而C2至C5分别为不同的ResNet卷积组，这些卷积组包含了多个Bottleneck结构，组内的特征图大小相同，组间大小递减。

自上而下：首先对C5进行1×1卷积降低通道数得到P5，然后依次进行上采样得到P4、P3和P2，目的是得到与C4、C3与C2长宽相同的特征，以方便下一步进行逐元素相加。这里采用2倍最邻近上采样，即直接对临近元素进行复制，而非线性插值。

横向连接（Lateral Connection）：目的是为了将上采样后的高语义特征与浅层的定位细节特征进行融合。高语义特征经过上采样后，其长宽与对应的浅层特征相同，而通道数固定为256，因此需要对底层特征C2至C4进行11卷积使得其通道数变为256，然后两者进行逐元素相加得到P4、P3与P2。由于C1的特征图尺寸较大且语义信息不足，因此没有把C1放到横向连接中。

卷积融合：在得到相加后的特征后，利用3×3卷积对生成的P2至P4再进行融合，目的是消除上采样过程带来的重叠效应，以生成最终的特征图。

### ASFF

上面的FPN是硬融合的时候是每一层固定的比例1融合，进一步的，参数化的配置融合比例会不会更好？这样是可以的，在YOLO3中提出了ASFF，自适应特征融合，通过可学习的参数来控制融合。

## Anchor

anchor字面意思是锚，anchor在计算机视觉中可以为锚框或锚点，目标检测中常出现的anchor box是锚框，表示固定的参考框。

目标检测是"在哪里有什么"的任务，在这个任务中，目标的类别不确定、数量不确定、位置不确定、尺度不确定，传统**非深度学习方法**如VJ和DPM，和**早期深度学习方法**，**都要金字塔多尺度+遍历滑窗的方式**，逐尺度逐位置判断"这个尺度的这个位置处有没有认识的目标"，**非常笨重耗时。**

由于CNN特征提取能力的加强，目前顶尖(SOTA)的目标检测方法几乎都用了**anchor技术**。首先预设一组不同尺度、不同比例、不同位置的固定参考框，注意关键词，**预设**，**不同尺寸**、不同比例、**不同位置、固定参考框**，覆盖几乎所有位置和尺度，每个参考框负责检测与其交并比大于阈值 (训练预设值，常用0.5或0.7) 的目标，anchor技术将问题转换为"这个固定参考框中有没有认识的目标，目标框偏离参考框多远"，不再需要多尺度遍历滑窗，真正实现了又好又快，如在Faster R-CNN和SSD两大主流目标检测框架及扩展算法中anchor都是重要部分。

Anchor算法，由Faster R-CNN提出，这个东西没有统一标准，但是它们基本是从Faster这里发展起来的，遵循它的基本结构。所以比较好的是从Faster R-CNN入手，不然学起来比较乱。

RetinaNet从 {P3, P4, P5, P6, P7} 拉出5个feature map进行检测，每个位置预测三尺度三比例，每个尺度预测9个形状anchor。

**注意目前选取位置的方法，就是将检测图的所有点都当成一个锚点位置，一般并没有特别的采用什么特别的选取方式，因为这种方式够了也是合理的。然后设置不同比例，不同尺寸。**

**感受野和检测尺寸相关**

**论文中提到的anchor设置没有对齐感受野，通常几个像素的中心位置偏移，对大目标来说IOU变化不会很大，但对小目标IOU变化剧烈，尤其感受野不够大的时候，anchor很可能偏移出感受野区域，影响性能。**

FPN中的bottom-up pathway将深度层特征上采样后与当前层相加，相加时感受野取最大，即相加后感受野等于深度层的感受野，FPN仅用微小计算量，实现了浅层feature map的感受野大幅提升，高效解决了“深度悖论”，目前是目标检测的标配。

**2. 还是论文中提到的，anchor尺度和比例都是人工给定的，尺度和比例必须要覆盖相应任务的可能出现的所有目标，如PASCAL VOC评价每图目标数较少、目标较大，所以SSD中最小60x60的anchor或许够用，但MS COCO中目标密集且小、数量多，对anchor的密度和最小尺度都有严格要求，60x60的最小框完全不够看，只能放大输入图像。**

一个重要的点，貌似是，目前anchor模块的除了总体思路，具体设计没有特别统一规范的方法，自行操作的地方比较多，不同网络可能都采取了很不相同的anchor模块，可以总结如下

总结以上几篇论文，设计目标检测算法时，anchor设置应该考虑一下几个方面 (以RetinaNet为例)：

anchor的密度：由检测所用feature map的stride决定，这个值与前景阈值密切相关，如同样检测{32}尺度，FPN阈值0.7用P2，RetinaNet前景阈值0.5用P3。

anchor的范围： RetinaNet中是anchor范围是32~512，这里应根据任务检测目标的范围确定，按需调整anchor范围，或目标变化范围太大如MS COCO，这时候应采用多尺度测试，当然，实际应用中需考虑多尺度测试的复杂度问题。一般来说最后特征提取层感受野一定要大于anchor大小，而且越大越好，这里感受野明显不足以支撑最大尺度的anchor，导致大尺度目标检测性能不够好；

anchor的形状数量：RetinaNet每个位置预测三尺度三比例共9个形状的anchor，这样可以增加anchor的密度，但stride决定这些形状都是同样的滑窗步进，需考虑步进会不会太大，如RetinaNet框架前景阈值是0.5时，一般anchor大小是stride的4倍左右。

检测层Pn的数量：RetinaNet中是P3~P7共5个检测层，如需检测更小的目标，可以考虑加入P2检测层。

检测层Pn的感受野：前面计算了RetinaNet中是ResNet+FPN的感受野在检测中、大目标时都是够用的，检测小目标时略显疲乏，kaiming最新论文《Rethinking ImageNet Pre-training》既然告诉我们预训练模型没那么重要时，那检测任务就可以着重考虑按照感受野的需求设计ConvNet。

总之，要做到anchor以足够密度实现全覆盖。

## CPSNET结构

Cross Stage Partial Network(CSPNet)部分阶段交叉网络-就是从网络结构设计的角度来解决以往工作在推理过程中需要很大计算量的问题。

实际上这个基本结构就相当于一个更大的残差块≈“在残差块堆叠层简称残差层的的上面再增加一个短连接”，然后只是细节上有一些不同。

作者认为推理计算过高的问题是由于网络优化中的梯度信息重复导致的。CSPNet通过将梯度的变化从头到尾地集成到特征图中，在减少了计算量的同时可以保证准确率。CSPNet是一种处理的思想，可以和ResNet、ResNeXt和DenseNet结合。

下图是cspnet用于分类中，可以看出计算量大幅下降，准确率保持不变或者略有提升（ps: 分类的准确率提升确实不多）

在目标识别中可以cpsnet可以大幅提示准确率，计算量自然地。

CSPNet提出主要是解决三个方面问题：

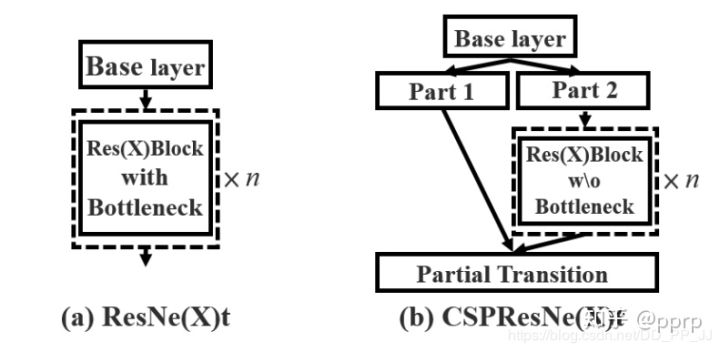
增强CNN的学习能力，能够在轻量化的同时保持准确性。

降低计算瓶颈

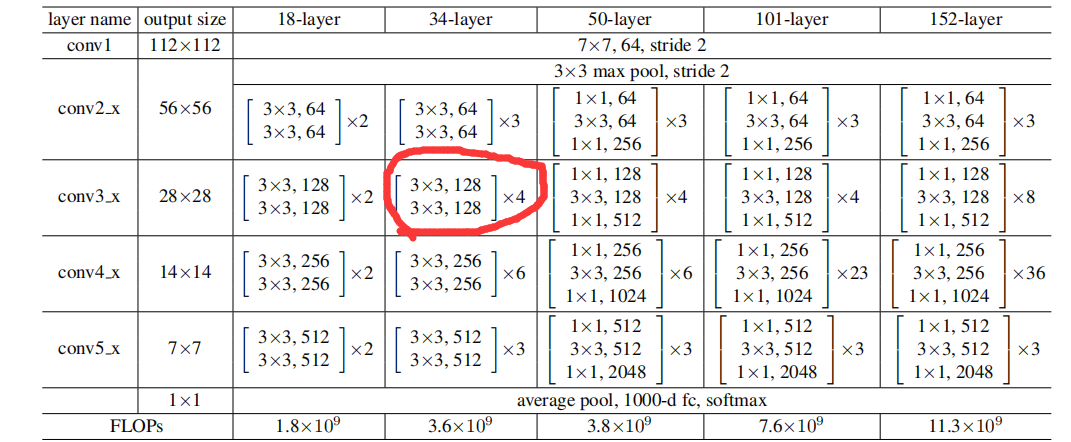
降低内存成本

CSPNet具体结构

### Backbone中的结构

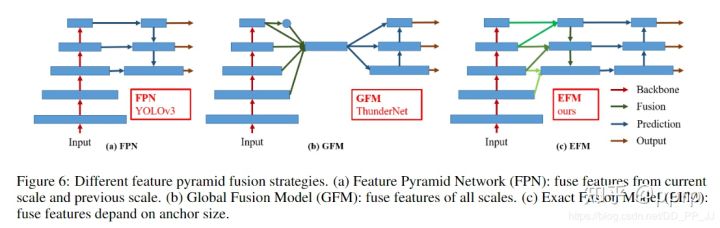


跟CSPDenseNet一样，将上一层分为两部分，Part1不进行操作直接concate，Part2进行卷积操作。\*n就对应resnet中的一个layer



### 融合中的结构

Neck特征融合中的结构



论文中列举了三种FPN：

第一个如（a）图所示，是最常见的FPN，在YOLOv3中使用。（ps: YOLOv3中的FPN跟原始FPN不同，其融合的方式是concate）

第二个如（b）图所示，是ThunderNet中提出的GFM, 之前的文章中有详解，直接将多个不同分辨率的特征进行融合，具体融合方式是相加。

第三个如（c）图所示，是EFM，也就是本文提出的融合方式，没两层之间特征进行融合，并且在原有FPN基础上再进行一次bottom-up的融合方式。

## SPP-NET

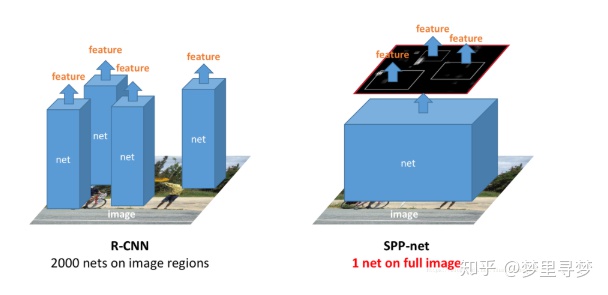
### 基本概念：

也是经典的方法，何凯明提出，被引量5k+（2020.11）

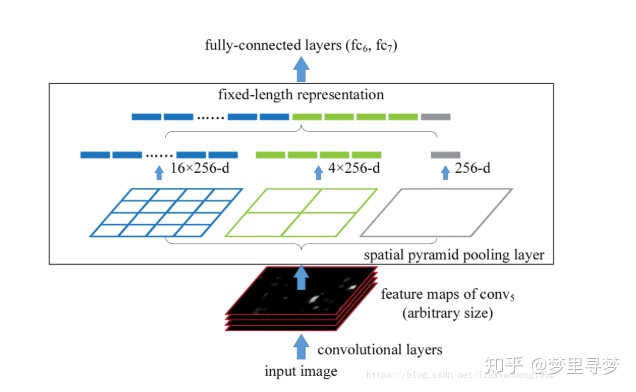
简单介绍如下

R-CNN对于一张图片，先使用segment seletive方法提取出约2000个候选区域，然后将这两千个候选区域分别送入网络中，即一张图片要经历2000次前向传播，这样会造成大量冗余。

SPP-net则提出了一种从候选区域到全图的特征(feature map)之间的对应映射关系，通过此种映射关系可以直接获取到候选区域的特征向量，不需要重复使用CNN提取特征，从而大幅度缩短训练时间。每张图片只需进行一次前向传播即可。



### SPP层(spatial pyramid pooling) 首先要明确的是这一层的位置，这一层加在最后一个卷积层与全连接层之间，目的就是为了输出固定长度的特征传给要求固定输入的全连接层 SPP层的结构如下图所示

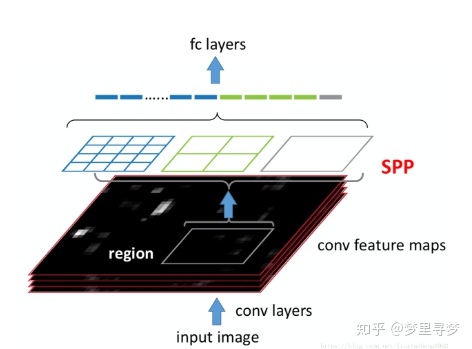


### SPP层的输入：

如下图灰色框所示

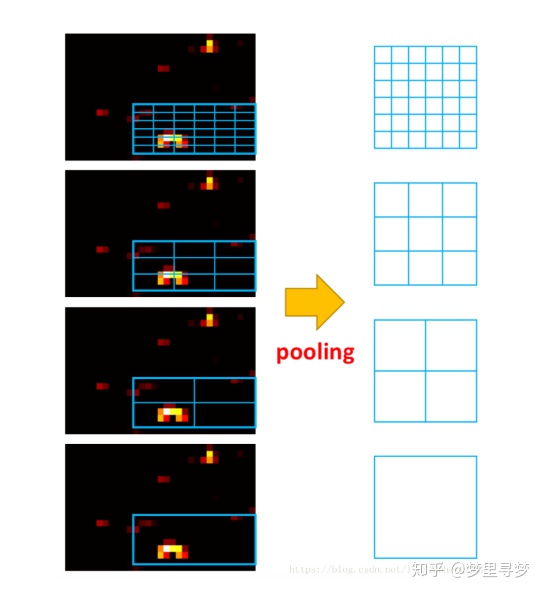
最后一层卷积输出的特征(我们称为feature map)，feature map为下图的黑色部分表示，SPP层的输入为与候选区域对应的在feature map上的一块区域

上面这句话可能有点绕，我们可以理解为一张图有约2000个候选区域，而对一张图做完卷积后得到feature map，在这个feature map上也有约2000个与候选区域对应的区域(这里的对应关系下面会详细讲解)



### SPP层的输出：

SPP layer分成1x1，2x2，4x4三个pooling结构(这部分结构如下图所示)，对每个输入(这里每个输入大小是不一样的)都作max pooling(论文使用的)，出来的特征再连接到一起，就是(16+4+1)x256的特征向量。



无论输入图像大小如何，出来的特征固定是(16+4+1)x256维度。这样就实现了不管图像中候选区域尺寸如何，SPP层的输出永远是(16+4+1)x256特征向量。

2.候选区域在原图与feature map之间的映射关系

这部分的计算其实就是感受野大小的计算。

优点主要有：

解决输入图片大小不一造成的缺陷。

由于把一个feature map从不同的尺寸进行pooling特征抽取，再聚合，提高了算法的robust和精度。

图像分类、目标检测都可以用，而且效果很棒。

一定程度上缓解了R-CNN耗时过多等问题。

# 正题，YOLOV5结构

## 基本概念：

### 基本组件命名和解释：

YOLOv3中的

CBL：Yolov3网络结构中的最小组件，由Conv+Bn+Leaky\_relu激活函数三者组成。

Res unit：借鉴Resnet网络中的残差结构，让网络可以构建的更深。

ResX：由一个CBL和X个残差组件构成，是Yolov3中的大组件。每个Res模块前面的CBL都起到下采样的作用，因此经过5次Res模块后，得到的特征图是608->304->152->76->38->19大小。

Yolov4中的：

CBM：Yolov4网络结构中的最小组件，由Conv+Bn+Mish激活函数三者组成。

CBL：由Conv+Bn+Leaky\_relu激活函数三者组成。

Res unit：借鉴Resnet网络中的残差结构，让网络可以构建的更深。

CSPX：借鉴CSPNet网络结构，由卷积层和X个Res unint模块Concate组成。

SPP：采用1×1，5×5，9×9，13×13的最大池化的方式，进行多尺度融合。

## 预处理：

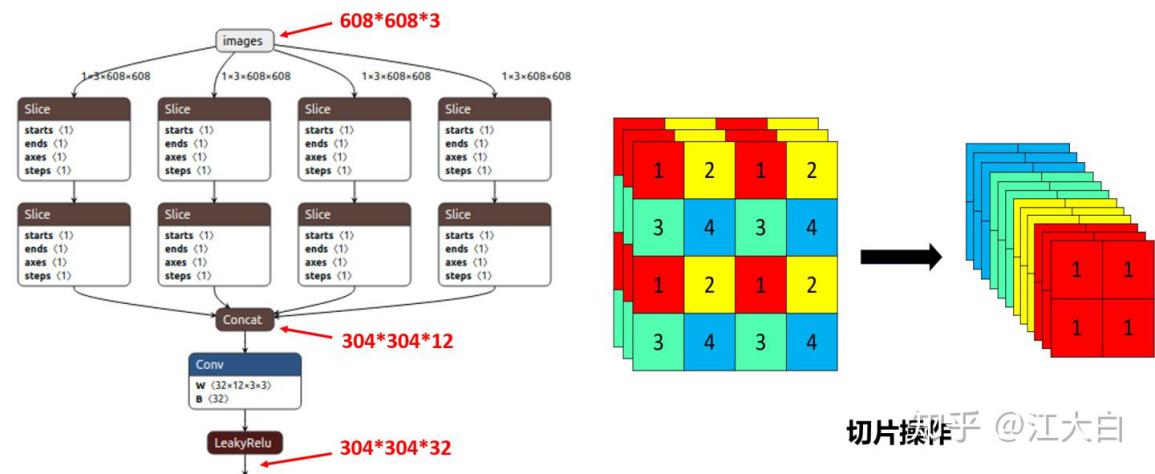
### 图片缩放问题的处理

目前检测很多时候会输入长宽不同的图片，这种情况下一般的处理是将图片缩放到正方形的框里面，并填充某种颜色，但是这种方案存在一定的缺点。填充过多，对检测有干扰，且影响运算速度。现在的处理方式就是不变成正方形，而是根据模型最后要缩放的倍数，比如这个的32倍，将图片一边对齐需要的长或宽，比如640，然后另一边填充到最接近的32的倍数。

“这种方式在之前github上Yolov3中也进行了讨论：[https://github.com/ultralytics/yolov3/issues/232](https://link.zhihu.com/?target=https://wx.qq.com/cgi-bin/mmwebwx-bin/webwxcheckurl?requrl=https%3A%2F%2Fgithub.com%2Fultralytics%2Fyolov3%2Fissues%2F232&skey=%40crypt_96d23a7c_7a713cdc64109256773c39e67ce4a665&deviceid=e850832231813449&pass_ticket=TgSQoHNgevOIg9%252B8R3aPNK%252F5sw6ZIUuR2A96p1sbiAGBktXTseCh8r9U9jZAQojj&opcode=2&scene=1&username=@8bbd87b4deb686cd79c1471b85752510" \t "https://zhuanlan.zhihu.com/p/_blank)

在讨论中，通过这种简单的改进，推理速度得到了37%的提升，可以说效果很明显”

### Focus结构



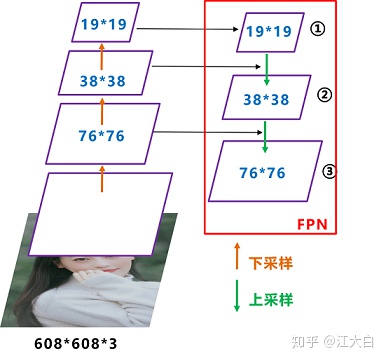
Focus结构，在Yolov3&Yolov4中并没有这个结构，其中比较关键是切片操作。

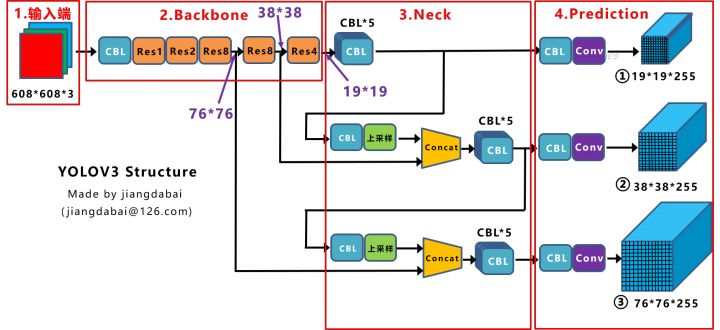
比如右图的切片示意图，4\*4\*3的图像切片后变成2\*2\*12的特征图。

以Yolov5s的结构为例，原始608\*608\*3的图像输入Focus结构，采用切片操作，先变成304\*304\*12的特征图

## Neck

从YOLOv3开始理解起来比较好





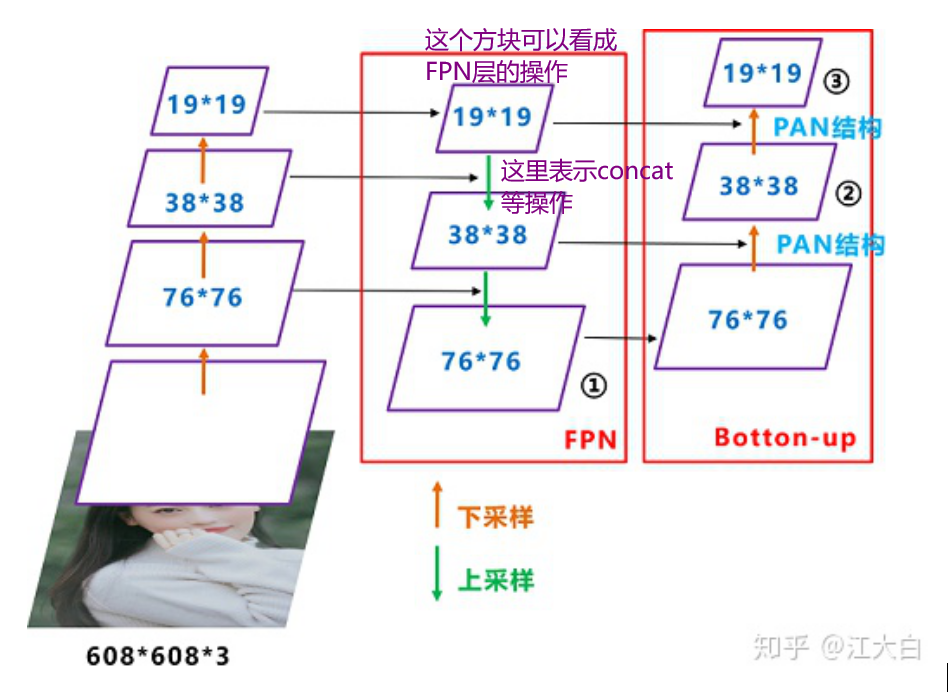
根据前面的特征融合金字塔，这个还是比较好理解的，

FPN第一层 顶层19\*19->多个CBL块处理->输出（进行预测了）

FPN第二层 FPN第一层得到的结果多个CBL块处理后的输入，用CBL+上采样变成38\*38，然后与前面的38\*38特征图concat—>然后就是同样的多个CBL块处理->输出

FPN第三层以此类推

YOLOv4中使用了FPN+PAN两级结构，原理示意图如下



先理解清楚这个示意图，不然后面的网络图看不懂

这个还是分了层的，可以看成分成了FPN层和PAN层，但是网络结构里面看上去是混到一起的。

图上的的向上向下箭头表示通过某种方式增加或降低特征图尺寸，然后两个箭头和到一起，表示concat等特征结合的操作。

这里FPN层和yolov3是一样的，然后后面就是

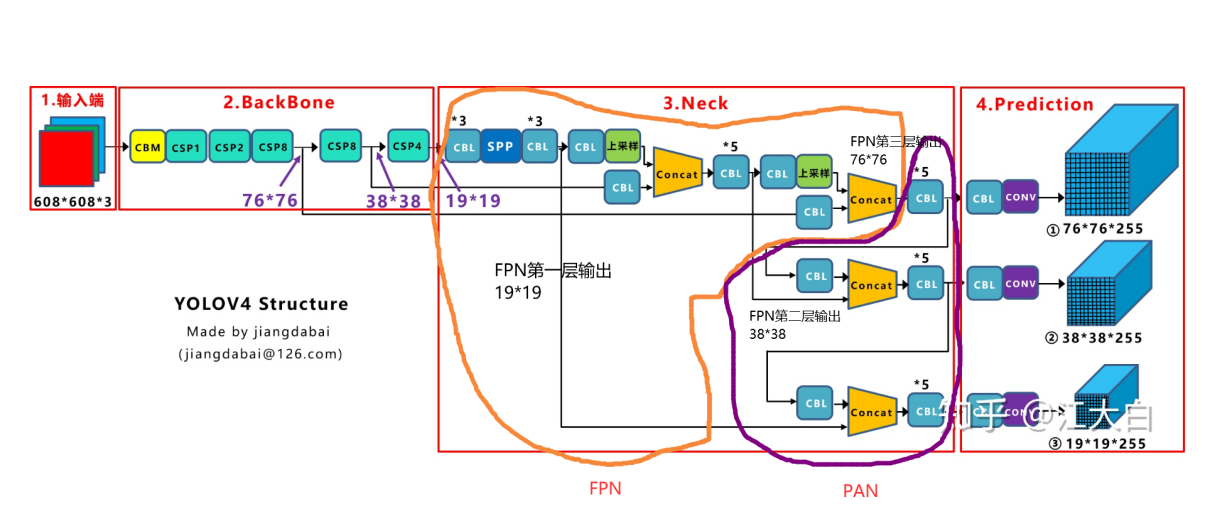
PAN最下面一层 把FPN最下面的输出传入PAN->PAN处理->输出（进入预测）

PAN往上第二层 76\*76的上采样和PAN输出的38\*38的层连起来->PAN处理->输出

PAN往上第三层 以此类推

可以看出，PAN和FPN基本上就是一个反向操作，FPN从把上面处理过的和下面连起来，然后处理，PAN反之，把下面处理过的和上面连起来，然后处理

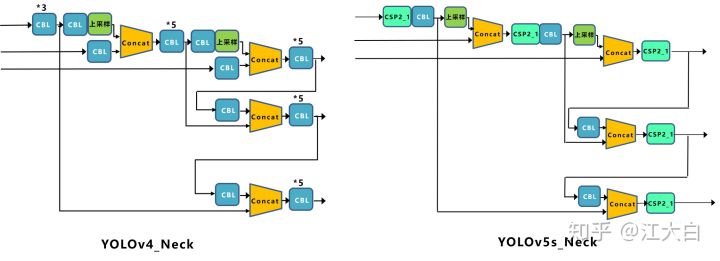
结构图如下：



然后看看这个结构图，这个结构图直接看上去是比较绕的，好多条连接连过去连过来的。但是是完全遵从上面的原理的，要理解一下。原图可能PAN这块反了，从上到下更好理解。

YOLOv5这一块就没多变了，只换了一个地方，将部分CBL块换成了CSP2块

Yolov4的Neck结构中，采用的都是普通的卷积操作。而Yolov5的Neck结构中，采用借鉴CSPnet设计的CSP2结构，加强网络特征融合的能力。



## Head预测部分

看结构图大输出特征图检测小物体，小输出特征图检测大物体

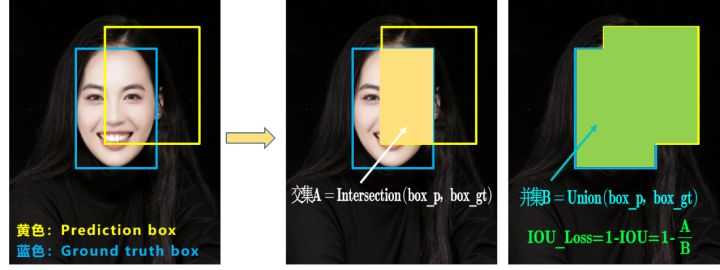
目标检测任务的损失函数一般由Classificition Loss（分类损失函数）和Bounding Box Regeression Loss（回归损失函数）两部分构成。

### 框损失的计算

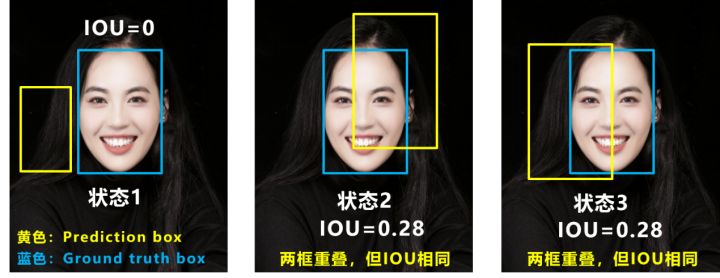
Bounding Box Regeression的Loss近些年的发展过程是：Smooth L1 Loss-> IoU Loss（2016）-> GIoU Loss（2019）-> DIoU Loss（2020）->CIoU Loss（2020）

1. CIOU\_loss

就是求面积的交并比



可以看到IOU的loss其实很简单，主要是交集/并集，但其实也存在两个问题。

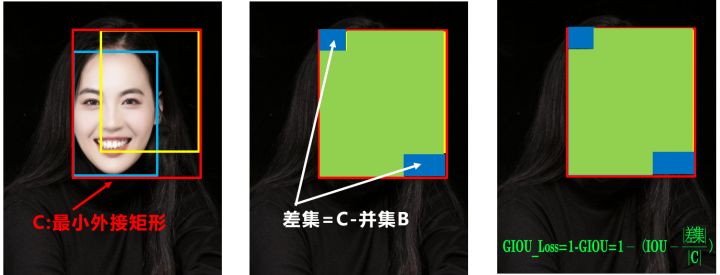


问题1：即状态1的情况，当预测框和目标框不相交时，IOU长期0，即loss变平了，梯度为0，梯度下降无法进行优化。

问题2：即状态2和状态3的情况，当两个预测框大小相同，两个IOU也相同，IOU\_Loss无法区分两者相交情况的不同。

还有不少情况，下面会进一步暴露

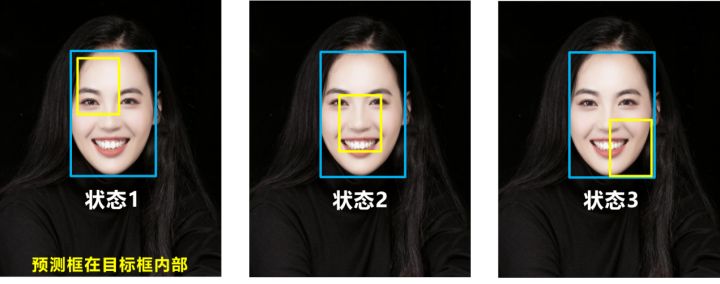
b.GIOU\_Loss



可以看到右图GIOU\_Loss中，增加了相交尺度的衡量方式，缓解了单纯IOU\_Loss时的尴尬。

但为什么仅仅说缓解呢？

因为还存在一种不足：



问题：状态1、2、3都是预测框在目标框内部且预测框大小一致的情况，这时预测框和目标框的差集都是相同的，因此这三种状态的GIOU值也都是相同的，这时GIOU退化成了IOU，无法区分相对位置关系。

c.DIOU\_Loss

可以归纳一下，好的目标框回归函数是怎样的呢，只要从我们的人的评价来看，效果好坏不同预测框应该得到高低不同的分数。

比如预测的框大小相同都在真实框内部，但是中心位置一个在真实框中间，一个在角上，显然前者更好。

所以应该这样考虑：重叠面积占比、重叠面积占比相同了比较中心点距离，中心距离可能相同，长宽比会不同，还有？

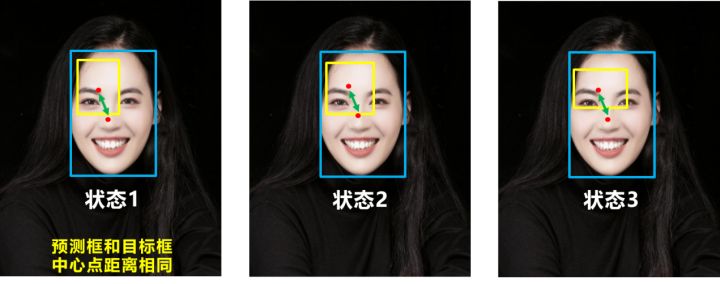
所以上面的方案都是有不足的，框的回归还可以进一步优化，于是有

DIOU\_Loss（Distance\_IOU\_Loss）



DIOU\_Loss考虑了重叠面积和中心点距离，当目标框包裹预测框的时候，直接度量2个框的距离，因此DIOU\_Loss收敛的更快。

但就像前面好的目标框回归函数所说的，没有考虑到长宽比。



针对这个问题，又提出了CIOU\_Loss

d.CIOU\_Loss

CIOU\_Loss和DIOU\_Loss前面的公式都是一样的，不过在此基础上还增加了一个影响因子，将预测框和目标框的长宽比都考虑了进去。

IMG_262

其中v是衡量长宽比一致性的参数，我们也可以定义为：

IMG_263

再来综合的看下各个Loss函数的不同点：

IOU\_Loss：主要考虑检测框和目标框重叠面积。

GIOU\_Loss：在IOU的基础上，解决边界框不重合时的问题。

DIOU\_Loss：在IOU和GIOU的基础上，考虑边界框中心点距离的信息。

CIOU\_Loss：在DIOU的基础上，考虑边界框宽高比的尺度信息。

Yolov4中采用了CIOU\_Loss的回归方式，使得预测框回归的速度和精度更高一些。

### 非极大值抑制方法：

#### 基本概念

在现有的基于anchor的目标检测算法中，都会产生数量巨大的候选矩形框，这些矩形框有很多是指向同一目标，因此就存在大量冗余的候选矩形框。**非极大值抑制算法的目的正在于此，它可以消除多余的框，找到最佳的物体检测位置。**目前，常用的目标检测算法，无论是One-stage的SSD系列算法、YOLO系列算法还是Two-stage的基于RCNN系列的算法，非极大值抑制都是其中必不可少的一个组件

非极大值抑制（Non-Maximum Suppression，以下简称NMS算法）的思想是**搜索局部极大值，抑制非极大值元素。**针对不同的应用场景和检测算法，由于矩形框的表征方式不同，NMS算法具有各种变体，本文针对NMS及其各种变体（主要是本文检测相关）进行介绍。

#### 经典NMS

经典NMS最初第一次应用到目标检测中是在RCNN算法中，其实现严格按照搜索局部极大值，抑制非极大值元素的思想来实现的，具体的实现步骤如下：

1. 设定目标框的置信度阈值，常用的阈值是0.5左右
2. 根据置信度降序排列候选框列表
3. 选取置信度最高的框A添加到输出列表，并将其从候选框列表中删除
4. 计算A与候选框列表中的所有框的IoU值，删除大于阈值的候选框
5. 重复上述过程，直到候选框列表为空，返回输出列表

#### 改进一：sofer-nms

经典NMS算法存在着一些问题：对于重叠物体无法很好的检测。当图像中存在两个重叠度很高的物体时，经典NMS会过滤掉其中置信度较低的一个。如下图所示，图中存在两个王闹海，经典NMS过滤后的结果如下图4所示：

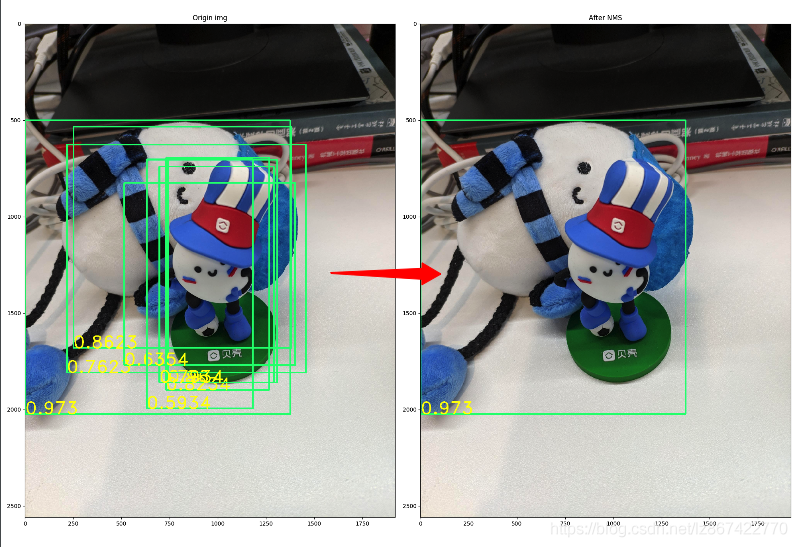


                                                                                              图4

而我们期望的结果是两个腰子都被算法成功检测出来。

为了解决这类问题，[Jan Hosang](https://arxiv.org/search/cs?searchtype=author&query=Hosang,+J),等人提出了Soft-NMS算法。Soft-NMS的算法伪代码如图5所示。其中红色框为经典NMS的步骤，而绿色框中的内容为Soft-NMS改进的步骤。可以看出，相对于经典NMS算法，Soft-NMS仅仅修改了一行代码。当选取了最大置信度的Bounding box之后，计算其余每个Bounding box与Bounding box的I ou值，经典NMS算法的做法是直接删除Iou大于阈值的Bounding box；而Soft-NMS则是使用一个基于Iou的衰减函数，降低Iou大于阈值Nt的Bounding box的置信度，IoU越大，衰减程度越大。

#### 更多：

Softer-nms考研置信度最高的不一定是最佳的候选框，具体方法待解

DIOU-nms相比经典NMS，不删除中心点距离较远的框，能够简单而较为有效的处理对象重复的问题。

### 输出层：

输出可以分成3个tensor，size 特征图大小 \* num\_anchors\*(num\_classes+5)，也就是

76 \* 76 \* 255， 38\*38\* 255， 19\*19 \*255

num\_anchors 表示一个锚点对应的几个框

+5 就是用5个数值表示框的具体位置

num\_classes 就是输出一个框可能是每种分类的概率

所以合起来就是，对每个层次的特征图比如38\*38，它的所有点都是建议点共38\*38减一点，一个建议点对应若干个预测框，也就是38\*38\*number\_anchors个框，每个框需要输出每个类的概率+框的位置=num\_classes+5（5个数值，还有一个数值是什么？），所以总共输出特38 \* 38 \* num\_anchors\*(num\_classes+5) = 38\*38\*255个数值。

# YOLO5 技巧总结

Yolov5技术总结

## Backbone：

Detection (YOLO) layer：  
损失函数：  
Yolov5-1.0、2.0、3.0：  
Backbone：  
Focus： 网络第一层进行 pixel un-shuffling而不是stride=2的卷积；该模块的设计主要是减少计算量加快速度；  
作者原话：https://github.com/ultralytics/yolov5/issues/847  
Focus() module is designed for FLOPS reduction and speed increase, not mAP increase.  
Also designed for layer count reduction. 1 Focus module replaces 3 yolov3/4 layers.  
  
SPP： 分别采用5/9/13的最大池化，再进行concat融合，提高感受野；  
BottleNeckCSP： Cross Stage Partial Networks，减少计算量，提高卷积神经网络学习能力；  
  
Detection (YOLO) layer：  
Anchor： 根据超参数中的hyp[‘anchor\_t’]来检查默认anchor与数据集标签的契合度，如果＜0.98，则根据数据集标签进行聚类重新获得anchor；  
默认anchor如下：  
  
[10,13, 16,30, 33,23] # P3/8  
[30,61, 62,45, 59,119] # P4/16  
[116,90, 156,198, 373,326] # P5/3  
参考：https://github.com/ultralytics/yolov5/issues/471  
边框回归： 在进行边框回归 筛选样本对应anchor的时候，就是通过hyp[‘anchor\_t’]来筛选，而不是iou；  
新的边框回归方式：  
  
在这里插入图片描述

## 损失函数：

边框回归：CIOU loss  
Objectness：GIOU  
分类：BCE  
损失平衡：ciou=0.05，objectness=1， cls=0.5;  
三个输出层损失平衡：4.0, 1.0, 0.4分别对应下采样8,16,32的输出层

## 优化策略：

Warmup热身训练；  
Cosine余弦退火；  
梯度累积；  
EMA；  
数据增强：  
Mosaic；  
仿射变换，随机的旋转，平移，缩放，裁剪，上下左右翻转；  
随机hsv；  
进行数据增强操作还有一个bbox筛选的过程：  
去除被裁剪过小的框(面积小于裁剪前的20%) ，并且还有长和宽必须大于2个像素，且长宽比范围在(1/20, 20)之间的限制；

## Yolov5-1.0、2.0、3.0：

1.0->2.0:  
yolov5x mAP有提升，但yolov5s mAP却下降了,  
训练策略的改变，包括余弦退火的公式更新了，以及类别损失cls\_loss的系数gain，对数据进行仿射变换(dataset.py数据增强部分)的超参数进行调整，三个output的损失比重balance的调整。  
2.0->3.0:  
V3.0据作者所说，大约10％的推理速度为代价提高了所有模型的mAP。尽管CUDA内存需求增加了约10％，但训练速度并未受到明显影响，具体未测试；  
最小的模型从Hardswish（）激活中受益最大，YOLOv5s / m / l / x的增加幅度为+0.9/+0.8/+0.7/+0.2mAP@0.5：0.95。  
  
主要做出的变化是，采用了hardswish激活函数替换CONV模块的LeakyReLu，但是注意：BottleneckCSP模块中的LeakyReLu未被替换，采用了CIOU作为损失函数(但这个更新好像是还在v2.0版本过度的时候已经更新)，还更改了一个默认超参数:translate=0.5 → 0.1(数据增强的仿射系数)。  
  
由于没有发paper，只有自己看代码学习，有不正确欢迎指正，谢谢！

Yolov5技术总结  
  
Laughing-q 2020-09-25 16:38:47 657 收藏 11  
分类专栏： YOLOV5 文章标签： 深度学习 人工智能 计算机视觉  
版权  
Yolov5技术总结  
Backbone：  
Detection (YOLO) layer：  
损失函数：  
Yolov5-1.0、2.0、3.0：  
Backbone：  
Focus： 网络第一层进行 pixel un-shuffling而不是stride=2的卷积；该模块的设计主要是减少计算量加快速度；  
作者原话：https://github.com/ultralytics/yolov5/issues/847  
Focus() module is designed for FLOPS reduction and speed increase, not mAP increase.  
Also designed for layer count reduction. 1 Focus module replaces 3 yolov3/4 layers.  
  
SPP： 分别采用5/9/13的最大池化，再进行concat融合，提高感受野；  
BottleNeckCSP： Cross Stage Partial Networks，减少计算量，提高卷积神经网络学习能力；  
  
Detection (YOLO) layer：  
Anchor： 根据超参数中的hyp[‘anchor\_t’]来检查默认anchor与数据集标签的契合度，如果＜0.98，则根据数据集标签进行聚类重新获得anchor；  
默认anchor如下：  
  
[10,13, 16,30, 33,23] # P3/8  
[30,61, 62,45, 59,119] # P4/16  
[116,90, 156,198, 373,326] # P5/3  
参考：https://github.com/ultralytics/yolov5/issues/471  
边框回归： 在进行边框回归 筛选样本对应anchor的时候，就是通过hyp[‘anchor\_t’]来筛选，而不是iou；  
新的边框回归方式：  
  
在这里插入图片描述  
  
损失函数：  
边框回归：CIOU loss  
Objectness：GIOU  
分类：BCE  
损失平衡：ciou=0.05，objectness=1， cls=0.5;  
三个输出层损失平衡：4.0, 1.0, 0.4分别对应下采样8,16,32的输出层  
  
优化策略：  
Warmup热身训练；  
Cosine余弦退火；  
梯度累积；  
EMA；  
数据增强：  
Mosaic；  
仿射变换，随机的旋转，平移，缩放，裁剪，上下左右翻转；  
随机hsv；  
进行数据增强操作还有一个bbox筛选的过程：  
去除被裁剪过小的框(面积小于裁剪前的20%) ，并且还有长和宽必须大于2个像素，且长宽比范围在(1/20, 20)之间的限制；  
  
Yolov5-1.0、2.0、3.0：  
1.0->2.0:  
yolov5x mAP有提升，但yolov5s mAP却下降了,  
训练策略的改变，包括余弦退火的公式更新了，以及类别损失cls\_loss的系数gain，对数据进行仿射变换(dataset.py数据增强部分)的超参数进行调整，三个output的损失比重balance的调整。  
2.0->3.0:  
V3.0据作者所说，大约10％的推理速度为代价提高了所有模型的mAP。尽管CUDA内存需求增加了约10％，但训练速度并未受到明显影响，具体未测试；  
最小的模型从Hardswish（）激活中受益最大，YOLOv5s / m / l / x的增加幅度为+0.9/+0.8/+0.7/+0.2mAP@0.5：0.95。  
  
主要做出的变化是，采用了hardswish激活函数替换CONV模块的LeakyReLu，但是注意：BottleneckCSP模块中的LeakyReLu未被替换，采用了CIOU作为损失函数(但这个更新好像是还在v2.0版本过度的时候已经更新)，还更改了一个默认超参数:translate=0.5 → 0.1(数据增强的仿射系数)。  
  
由于没有发paper，只有自己看代码学习，有不正确欢迎指正，谢谢！