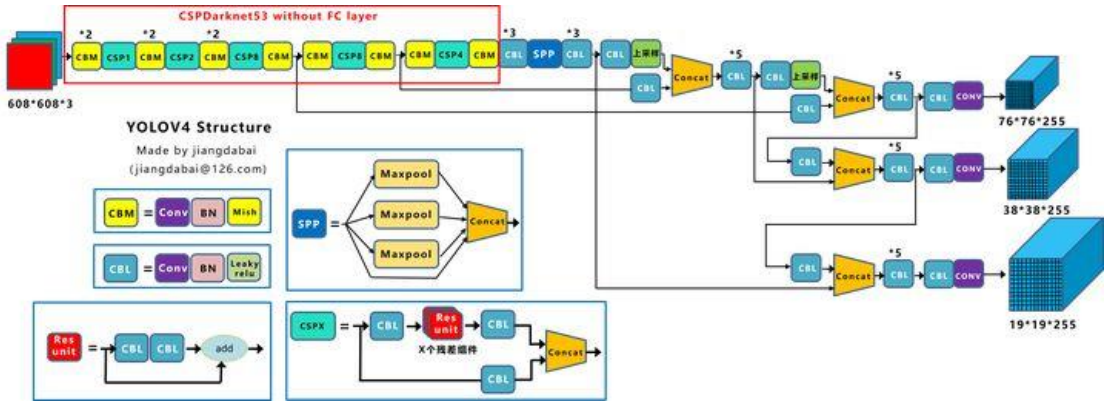
YOLOV4 原理分析文档

团队编号:2002066　二仙桥老大爷

目录

1.YoloV4 核心基础内容

1.1 网络结构图



Yolov4 的结构图和 Yolov3 相比，因为多了 **CSP 结构，PAN 结构**，如果单纯看可视化流程图，会觉得很绕，不过在绘制出上面的图形后，会觉得豁然开朗，其实整体架构和 Yolov3 是相同的，不过使用各种新的算法思想对各个子结构都进行了改进。

1.2 Yolov4 的五个基本组件

1. **CBM：** Yolov4 网络结构中的最小组件，由 Conv+Bn+Mish 激活函数三者组成。
2. **CBL：** 由 Conv+Bn+Leaky_relu 激活函数三者组成。
3. **Res unit：** 借鉴 Resnet 网络中的残差结构，让网络可以构建的更深。
4. **CSPX：**借鉴 CSPNet 网络结构，由三个卷积层和 X 个 Res unint 模块 Concate 组成。
5. **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。

其他基础操作：

1. **Concat：** 张量拼接，维度会扩充，和 Yolov3 中的解释一样，对应于 cfg 文件中的 route 操作。
2. **add：** 张量相加，不会扩充维度，对应于 cfg 文件中的 shortcut 操作。

Backbone 中卷积层的数量：

和 Yolov3 一样，再来数一下 Backbone 里面的卷积层数量。

每个 CSPX 中包含 3+2*X 个卷积层，因此整个主干网络 Backbone 中一共包含 2+（3+2*1）+2+（3+2*2）+2+（3+2*8）+2+（3+2*8）+2+（3+2*4）+1=72。

2.核心基础内容

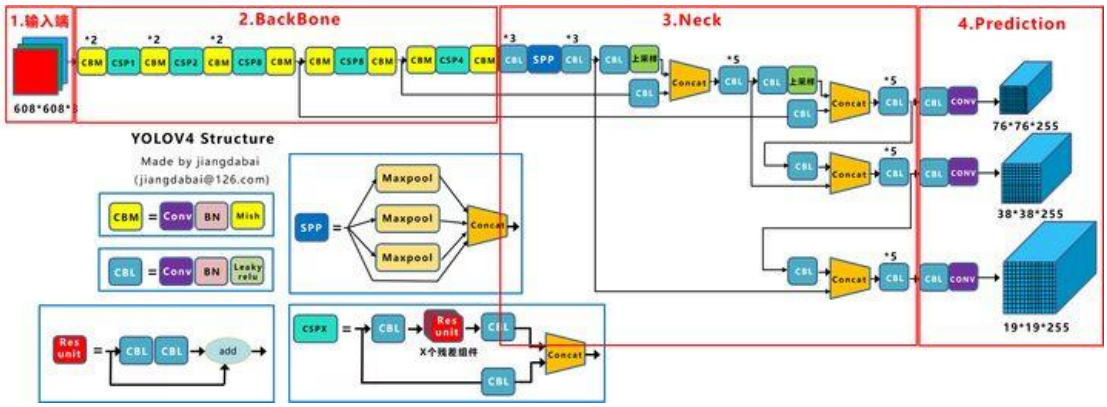
Yolov4 算法创新分为三种方式：

1. **第一种：面目一新的创新**，比如 Yolov1、Faster-RCNN、Centernet 等，开创出新的算法领域，不过这种也是最难的
2. **第二种：守正出奇的创新**，比如将图像金字塔改进为特征金字塔
3. **第三种：各种先进算法集成的创新**，比如不同领域发表的最新论文的 tricks，集成到自己的算法中，却发现有出乎意料的改进 **Yolov4** 既有第

二种也有第三种创新，组合尝试了大量深度学习领域最新论文的 20 多项研究成果。

目前 Yolov4 代码的 star 数量已经 **1 万多**，目前超过这个数量的，目标检测领域只有 **Facebook 的 Detectron(v1-v2)**、和 **Yolo(v1-v3)官方代码库(已停止更新)**。

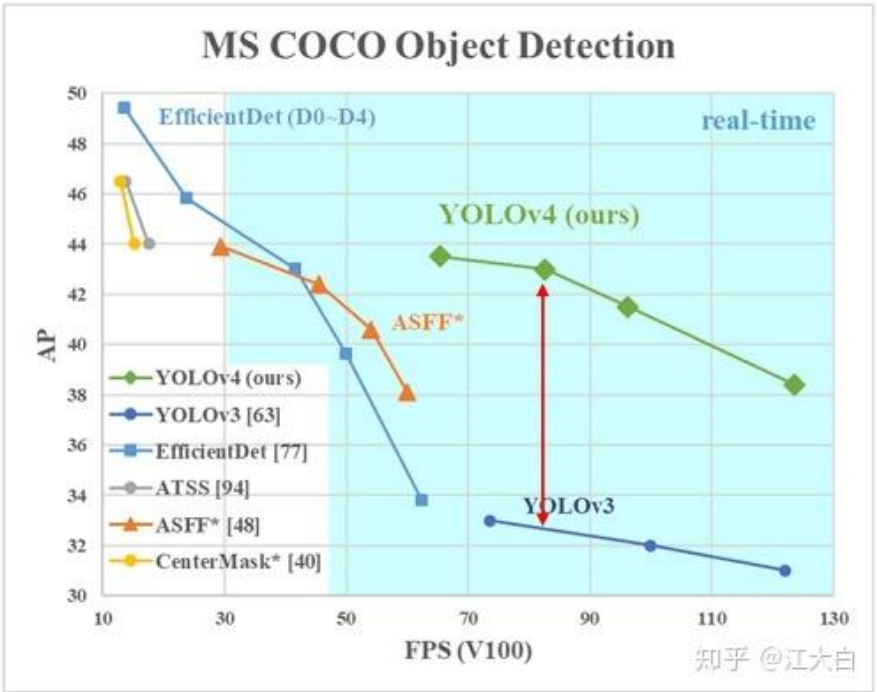
为了便于分析，将 Yolov4 的整体结构拆分成四大板块：



我们主要从以上 4 个部分对 YoloV4 的创新之处进行讲解。

1. **输入端：**这里指的创新主要是训练时对输入端的改进，主要包括 **Mosaic 数据增强、cmBN、SAT 自对抗训练**
2. **BackBone 主干网络：**将各种新的方式结合起来，包括：**CSPDarknet53、Mish 激活函数、Dropblock**
3. **Neck：**目标检测网络在 BackBone 和最后的输出层之间往往会插入一些层，比如 Yolov4 中的 **SPP 模块、FPN+PAN 结构**
4. **Prediction：**输出层的锚框机制和 Yolov3 相同，主要改进的是训练时的损失函数 **CIOU_Loss**，以及预测框筛选的 nms 变为 **DIOU_nms**

总体来说，**Yolov4** 对 **Yolov3** 的各个部分都进行了改进优化，下面丢上作者的算法对比图。



仅对比 **Yolov3 和 Yolov4**，在 COCO 数据集上，同样的 FPS 等于 83 左右时，

Yolov4 的 AP 是 43，而 Yolov3 是 33，直接上涨了 **10 个百分点**。
不得不服，当然可能针对具体不同的数据集效果也不一样，但总体来说，改进效果是很优秀的，下面我们对 **Yolov4 的各个创新点**继续进行深挖。

3.输入端创新

由于我们服务器 GPU 显卡数量并不是很多，**Yolov4** 对训练时的输入端进行改进，使得训练在单张 GPU 上也能有不错的成绩。比如**数据增强 Mosaic、cmBN、SAT 自对抗训练。**

但感觉 cmBN 和 SAT 影响并不是很大，所以这里主要讲解 Mosaic 数据增强。

（1）Mosaic 数据增强

Yolov4 中使用的 **Mosaic** 是参考 2019 年底提出的 **CutMix 数据增强**的方式，但 **CutMix** 只使用了两张图片进行拼接，而 **Mosaic 数据增强**则采用了 4 张图片，**随机缩放、随机裁剪、随机排布**的方式进行拼接。

在平时项目训练时，**小目标的 AP** 一般比中目标和大目标低很多。而 Coco 数据集中也包含大量的小目标，但比 增强中 1是小目标的分布**并不均匀**。
首先看下小、中、大目标的定义：
2019 年发布的论文《Augmentation for small object detection》对此进行了区分：

	Min rectangle area	Max rectangle area
Small object	**0*0**	**32*32**
Medium object	32*32	96*96
Large object	96*96	∞*∞

可以看到小目标的定义是目标框的长宽 0×0\~32×32 之间的物体。

	Small	Mid	Large
Ratio of total boxes(%)	41.4	34.3	24.3
Ratio of images included(%)	52.3	70.7	83.0

但在整体的数据集中，小、中、大目标的占比并不均衡。
如上表所示，Coco 数据集中小目标占比达到 **41.4\%**，数量比中目标和大目标都要多。

但在所有的训练集图片中，只有 **52.3\%**的图片有小目标，而中目标和大目标的分布相对来说更加均匀一些。

针对这种状况，Yolov4 的作者采用了 **Mosaic 数据增强**的方式。
主要有几个优点：

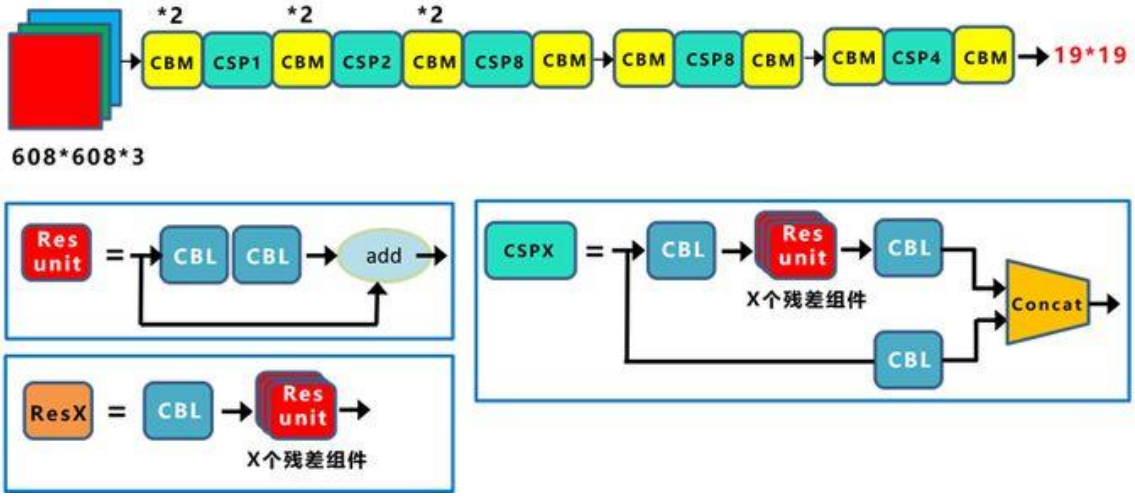
1. **丰富数据集：**随机使用 **4 张图片**，随机缩放，再随机分布进行拼接，大大丰富了检测数据集，特别是随机缩放增加了很多小目标，让网络的鲁棒性更好。
2. **减少 GPU：**可能会有人说，随机缩放，普通的数据增强也可以做，但作者考虑到很多人可能只有一个 GPU，因此 Mosaic 增强训练时，可以直接计算 4 张图片的数据，使得 Mini-batch 大小并不需要很大，一个 GPU 就可

以达到比较好的效果。

4. BackBone 创新

4.1 CSPDarknet53

CSPDarknet53 是在 Yolov3 主干网络 **Darknet53** 的基础上，借鉴 **2019 年 CSPNet** 的经验，产生的 **Backbone** 结构，其中包含了 **5 个 CSP** 模块。



每个 CSP 模块前面的卷积核的大小都是 3*3，因此可以起到下采样的作用。因为 Backbone 有 5 个 **CSP 模块**，输入图像是 **608*608**，所以特征图变化的规律是：**608->304->152->76->38->19**

经过 5 次 CSP 模块后得到 19*19 大小的特征图。

而且作者只在 Backbone 中采用了 **Mish 激活函数**，网络后面仍然采用 **Leaky_relu 激活函数。**

CSPNet 全称是 Cross Stage Paritial Network，主要从网络结构设计的角度解决推理中从计算量很大的问题。

CSPNet 的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。

因此采用 CSP 模块先将基础层的特征映射划分为两部分，然后通过跨阶段层次结构将它们合并，在减少了计算量的同时可以保证准确率。

因此 Yolov4 在主干网络 Backbone 采用 CSPDarknet53 网络结构，主要有三个方面的优点：

优点一： 增强 CNN 的学习能力，使得在轻量化的同时保持准确性。

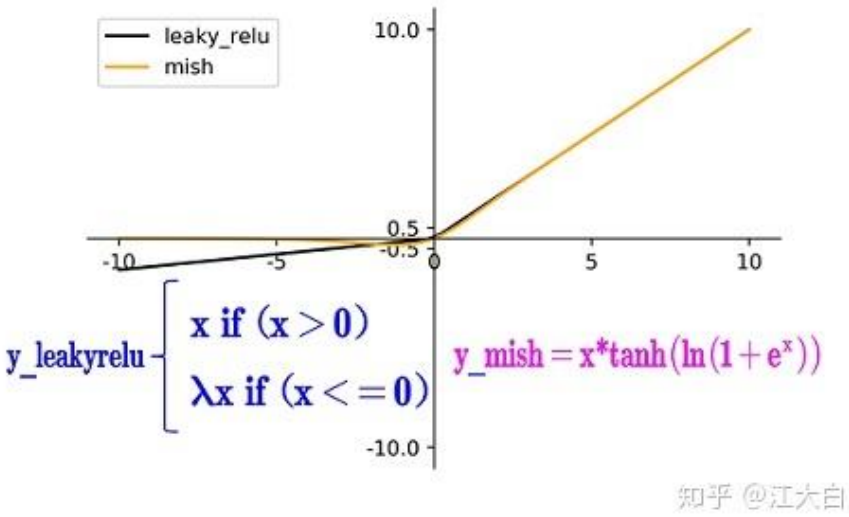
优点二： 降低计算瓶颈

优点三： 降低内存成本

4.2 Mish 激活函数

Mish 激活函数是 **2019 年下半年**提出的激活函数

和 **Leaky_relu 激活函数**的图形对比如下：



Yolov4 的 **Backbone** 中都使用了 **Mish 激活函数**，而后面的网络则还是使用 leaky_relu 函数。

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.2%	93.6%
	✓	✓		✓			**77.8%**	**94.4%**
	✓	✓		✓		✓	**78.7%**	**94.8%**

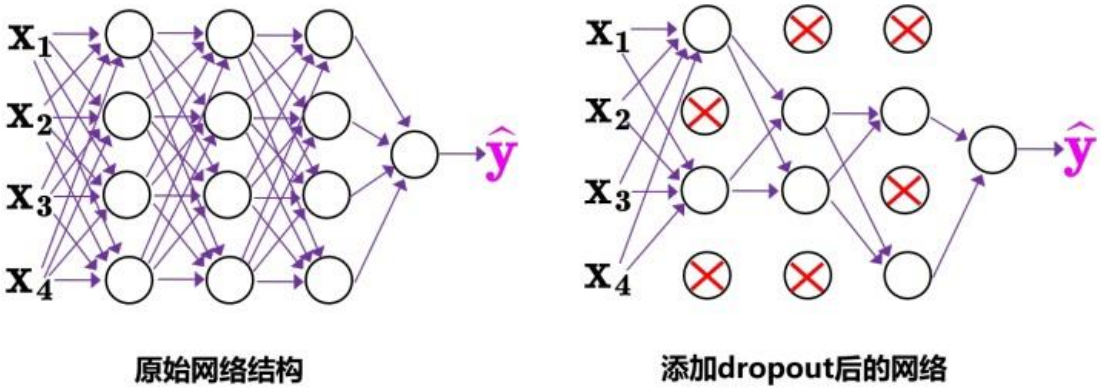
Yolov4 作者实验测试时，使用 **CSPDarknet53** 网络在 **ImageNet 数据集**上做图像分类任务，发现使用了 Mish 激活函数的 **TOP-1** 和 **TOP-5** 的精度比没有使用时都略高一些。

因此在设计 Yolov4 目标检测任务时，主干网络 Backbone 还是使用 **Mish 激活函数。**

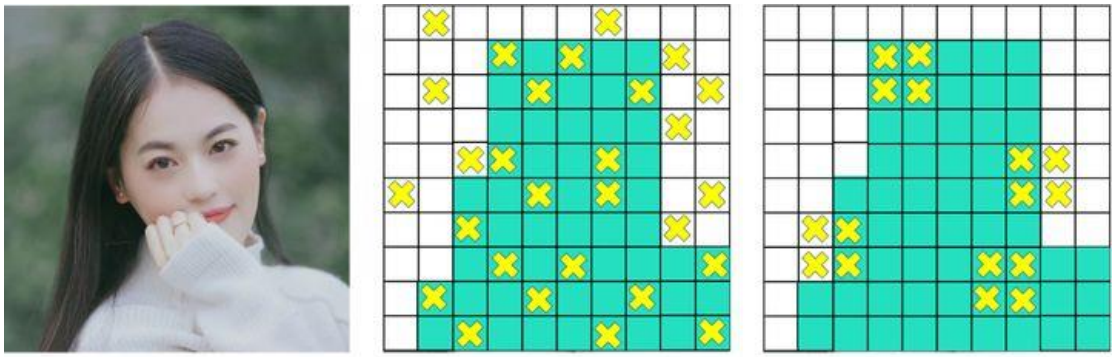
4.3 Dropblock

Yolov4 中使用的 **Dropblock**，其实和常见网络中的 Dropout 功能类似，也是缓解过拟合的一种正则化方式。

传统的 Dropout 很简单，一句话就可以说的清：**随机删除减少神经元的数量，使网络变得更简单。**



而 Dropblock 和 Dropout 相似，比如下图：



原始图片　　Dropout　　Dropblock

中间 Dropout 的方式会随机的删减丢弃一些信息，但 **Dropblock 的研究者**认为，卷积层对于这种随机丢弃并不敏感，因为卷积层通常是三层连用：**卷积+激活+池化层**，池化层本身就是对相邻单元起作用。而且即使随机丢弃，卷积层仍然可以从相邻的激活单元学习到**相同的信息**。

因此，在全连接层上效果很好的 Dropout 在卷积层上**效果并不好**。

所以**右图 Dropblock 的研究者**则干脆整个局部区域进行删减丢弃。

这种方式其实是借鉴 **2017 年的 cutout 数据增强**的方式，cutout 是将输入图像的部分区域清零，而 Dropblock 则是将 Cutout 应用到每一个特征图。而且并不是用固定的归零比率，而是在训练时以一个小的比率开始，随着训练过程**线性的增加这个比率**。

CIFAR-10 上Cutout数据增强

Dropblock 的研究者与 **Cutout** 进行对比验证时，发现有几个特点：

优点一：Dropblock 的效果优于 Cutout

优点二：Cutout 只能作用于输入层，而 Dropblock 则是将 Cutout 应用到网络中的每一个特征图上

优点三：Dropblock 可以定制各种组合，在训练的不同阶段可以修改删减的概率，从空间层面和时间层面，和 Cutout 相比都有更精细的改进。

Yolov4 中直接采用了更优的 **Dropblock**，对网络的正则化过程进行了全面的升级改进。

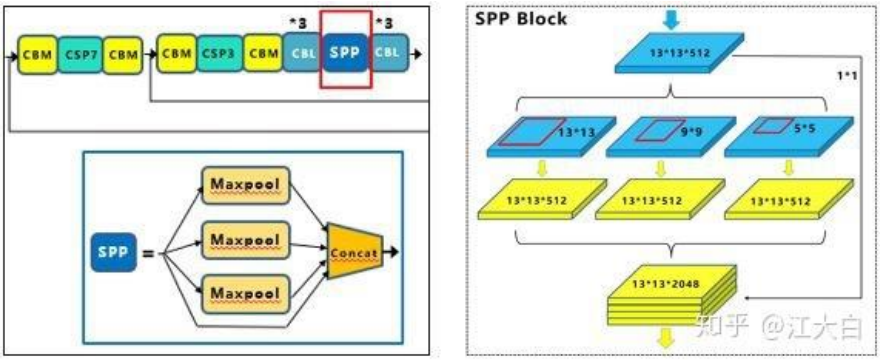
5.Neck 创新

在目标检测领域，为了更好的提取融合特征，通常在 **Backbone** 和**输出层**，会插入一些层，这个部分称为 Neck。相当于目标检测网络的颈部，也是非常关键的。

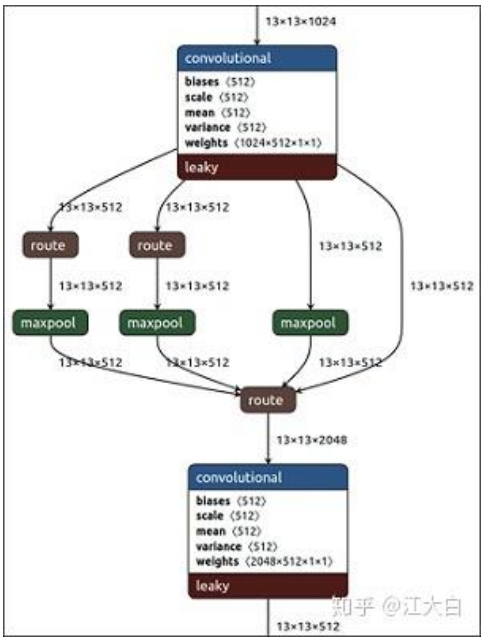
Yolov4 的 Neck 结构主要采用了 **SPP 模块**、**FPN+PAN** 的方式。

5.1 SPP 模块

SPP 模块，其实在 Yolov3 中已经存在了，在 **Yolov4** 的 C++代码文件夹**中**有一个 **Yolov3_spp 版本**，但有的同学估计从来没有使用过，在 Yolov4 中，SPP 模块仍然是在 Backbone 主干网络之后：



作者在 SPP 模块中，使用 k={1*1,5*5,9*9,13*13}的最大池化的方式，再将不同尺度的特征图进行 Concat 操作。



和 Yolov4 作者的研究相同，采用 **SPP 模块**的方式，比单纯的使用 **k*k 最大池化**的方式，更有效的增加主干特征的接收范围，显著的分离了最重要的上下文特征。

Yolov4 的作者在使用 **608*608** 大小的图像进行测试时发现，在 COCO 目标检测任务中，以 0.5\%的额外计算代价将 AP50 增加了 2.7\%，因此 Yolov4 中也采用了 **SPP 模块**。

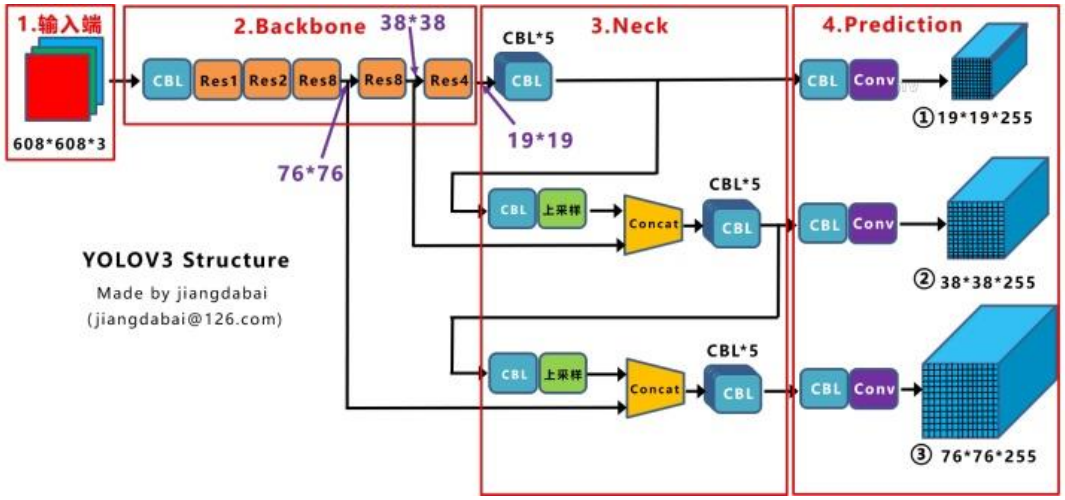
5.2 FPN+PAN

PAN 结构比较有意思，看了网上 Yolov4 关于这个部分的讲解，大多都是讲的比较笼统的，而 PAN 是借鉴图像分割领域 PANet 的创新点，有些同学可能不是很清楚。

下面大白将这个部分拆解开来，看下 Yolov4 中是如何设计的。

Yolov3 结构：

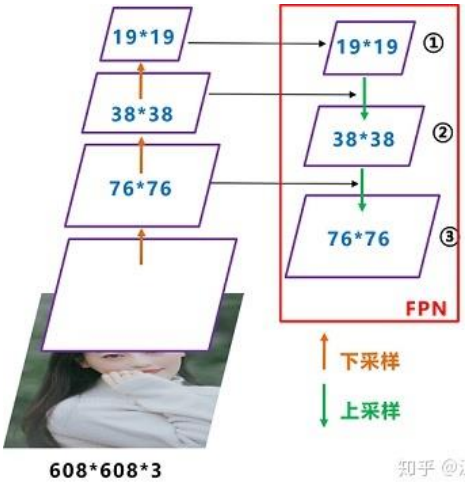
我们先来看下 Yolov3 中 Neck 的 FPN 结构



可以看到经过几次下采样，三个紫色箭头指向的地方，输出分别是 **76*76、38*38、19*19。**

以及最后的 **Prediction** 中用于预测的三个特征图 **①19*19*255、②38*38*255、③76*76*255。[注：255 表示 80 类别(1+4+80)×3=255]**

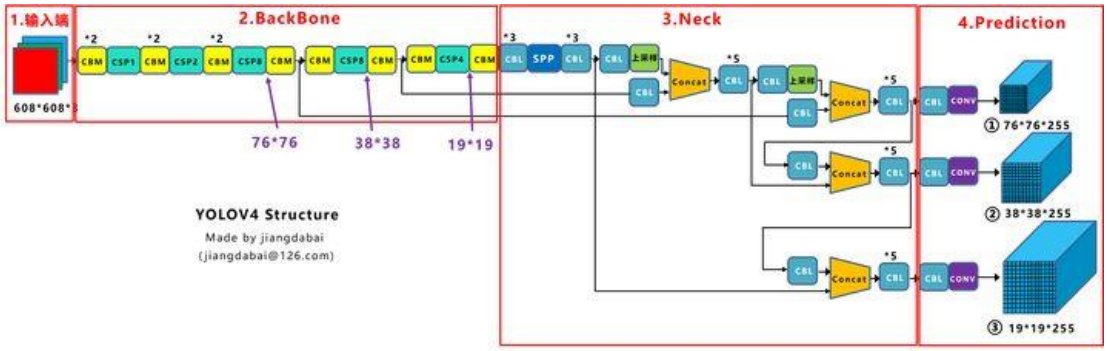
我们将 Neck 部分用立体图画出来，更直观的看下两部分之间是如何通过 **FPN 结构**融合的。



如图所示，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。

Yolov4 结构：

而 Yolov4 中 Neck 这部分除了使用 FPN 外，还在此基础上使用了 PAN 结构：

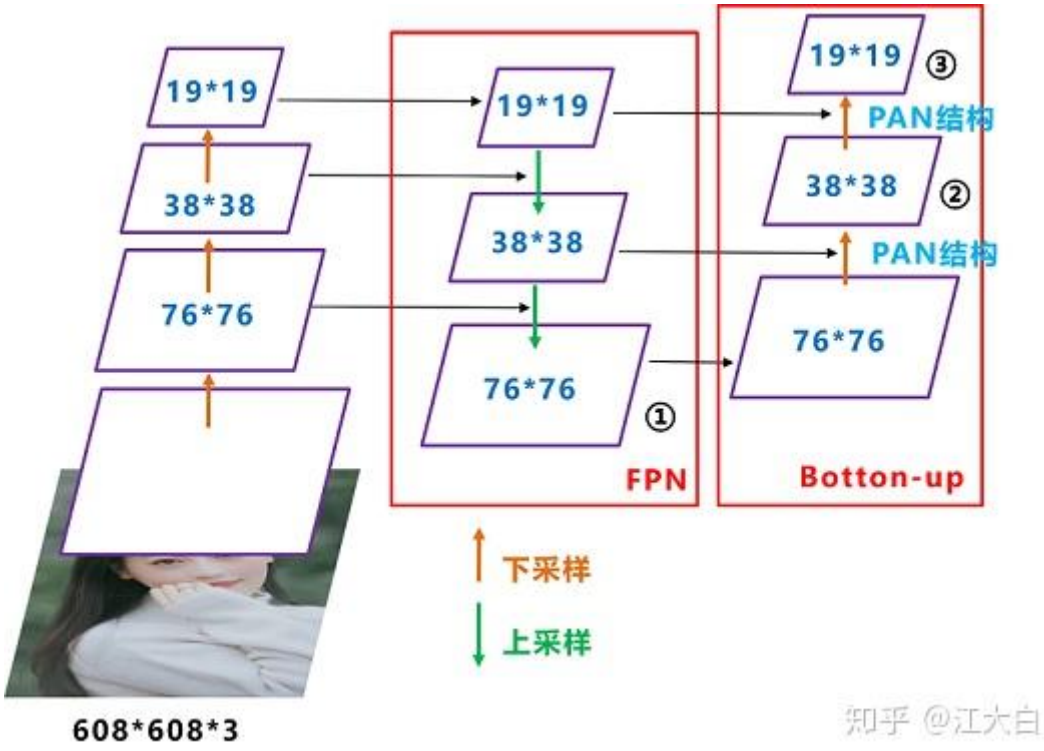


前面 CSPDarknet53 中讲到，每个 CSP 模块前面的卷积核都是 **3*3 大小**，相当于下采样操作。

因此可以看到三个紫色箭头处的特征图是 **76*76、38*38、19*19。**

以及最后 Prediction 中用于预测的三个特征图：**①76*76*255，②38*38*255，③19*19*255。**

我们也看下 **Neck** 部分的立体图像，看下两部分是如何通过 **FPN+PAN 结构**进行融合的。



和 Yolov3 的 FPN 层不同，Yolov4 在 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。**

这样结合操作，FPN 层自顶向下传达**强语义特征**，而特征金字塔则自底向上传达**强定位特征**，两两联手，从不同的主干层对不同的检测层进行参数聚合，这

样的操作确实很皮。

FPN+PAN 借鉴的是 18 年 CVPR 的 **PANet**，当时主要应用于**图像分割领域**，但 Alexey 将其拆分应用到 Yolov4 中，进一步提高特征提取的能力。

不过这里需要注意几点：

注意一：

Yolov3 的 FPN 层输出的三个大小不一的特征图①②③直接进行预测

但 Yolov4 的 FPN 层，只使用最后的一个 76*76 特征图①，而经过两次 PAN 结构，输出预测的特征图②和③。

这里的不同也体现在 cfg 文件中，这一点有很多同学之前不太明白，

比如 Yolov3.cfg 最后的三个 Yolo 层，

第一个 Yolo 层是最小的特征图 **19*19**，mask=**6,7,8**，对应**最大的 anchor box。**

第二个 Yolo 层是中等的特征图 **38*38**，mask=**3,4,5**，对应**中等的 anchor box。**

第三个 Yolo 层是最大的特征图 **76*76**，mask=**0,1,2**，对应**最小的 anchor box。**

而 Yolov4.cfg 则**恰恰相反**

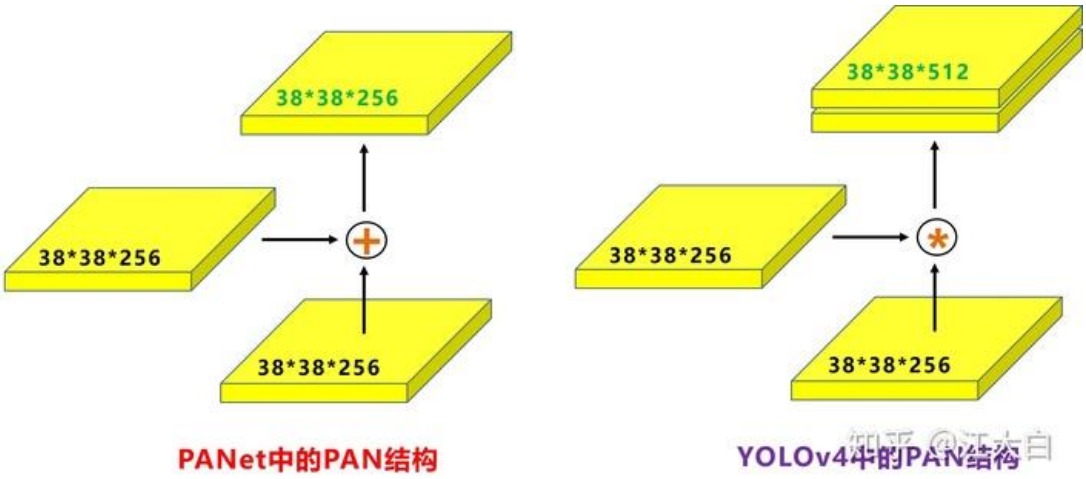
第一个 Yolo 层是最大的特征图 **76*76**，mask=**0,1,2**，对应**最小的 anchor box。**

第二个 Yolo 层是中等的特征图 **38*38**，mask=**3,4,5**，对应**中等的 anchor box。**

第三个 Yolo 层是最小的特征图 **19*19**，mask=**6,7,8**，对应**最大的 anchor box。**

注意点二：

原本的 PANet 网络的 **PAN 结构**中，两个特征图结合是采用 **shortcut** 操作，而 Yolov4 中则采用 **concat（route）**操作，特征图融合后的尺寸发生了变化。



6 Prediction 创新

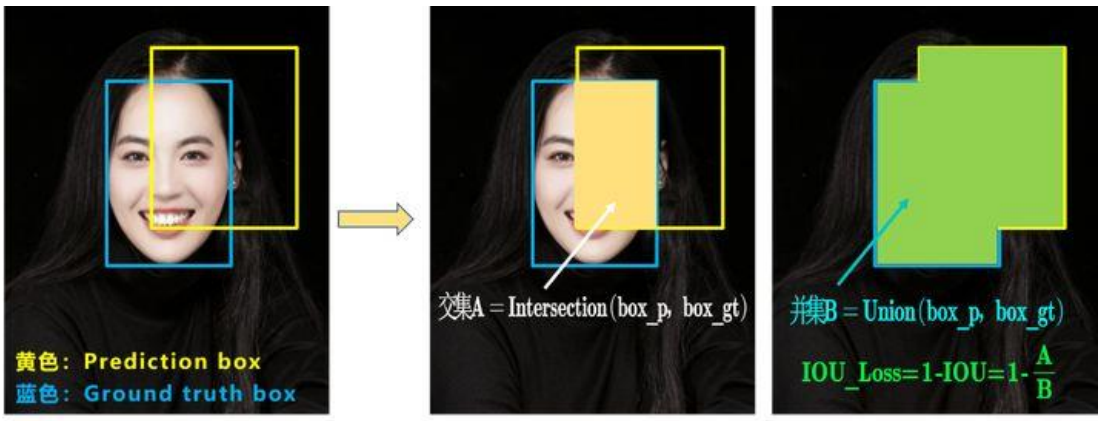
6.1 CIOU_loss

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。

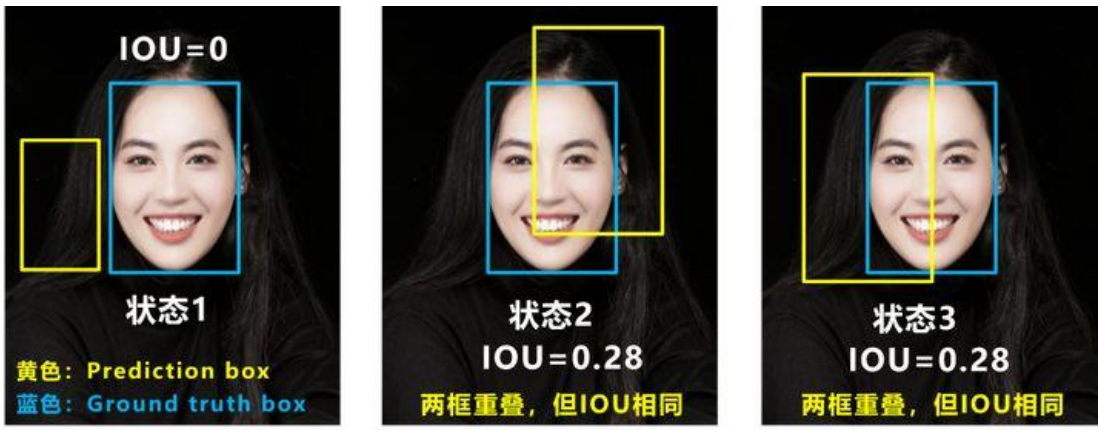
Bounding Box Regeression 的 Loss 近些年的发展过程是：**Smooth L1 Loss-> IoU Loss（2016）-> GIoU Loss（2019）-> DIoU Loss（2020）->CIoU Loss（2020）**

我们从最常用的 **IOU_Loss** 开始，进行对比拆解分析，看下 Yolov4 为啥要选择 **CIOU_Loss。**

a. IOU_Loss



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。

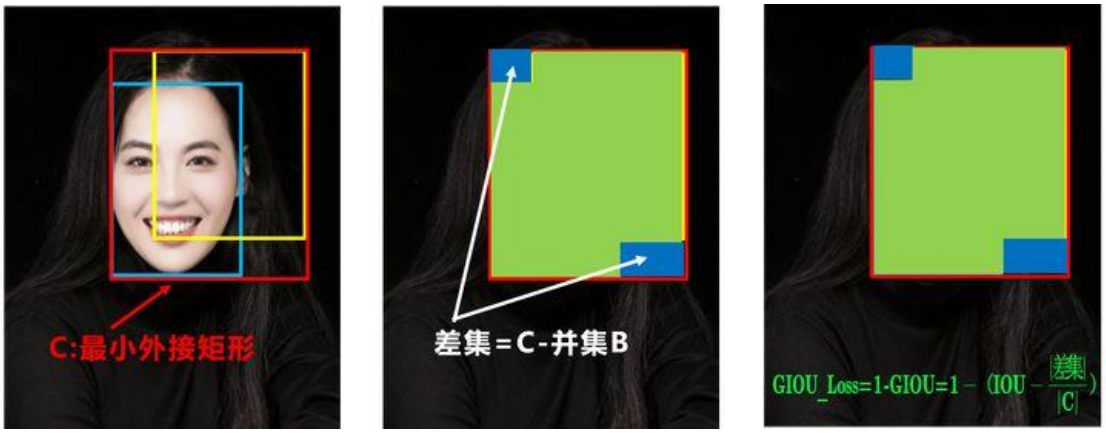


问题 1：即状态 1 的情况，当预测框和目标框不相交时，IOU=0，无法反应两个框距离的远近，此时损失函数不可导，IOU_Loss 无法优化两个框不相交的情况。

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。

因此 2019 年出现了 GIOU_Loss 来进行改进。

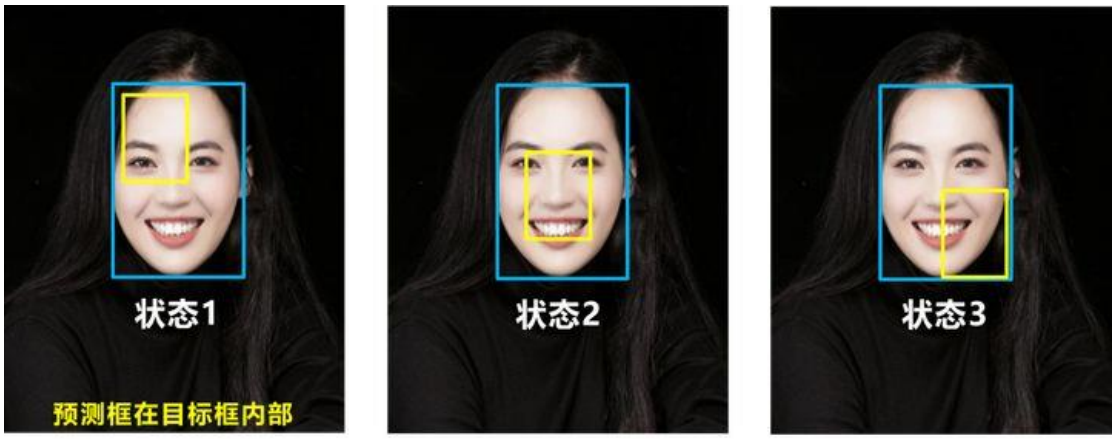
b. GIOU_Loss



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。

基于这个问题，**2020 年**的 AAAI 又提出了 **DIOU_Loss**。

c. DIOU_Loss

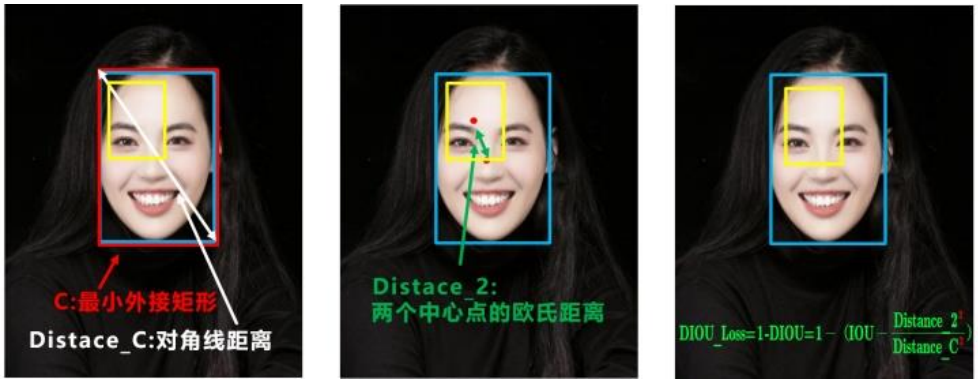
好的目标框回归函数应该考虑三个重要几何因素：**重叠面积、中心点距离，长宽比。**

针对 IOU 和 GIOU 存在的问题，作者从两个方面进行考虑

一：如何最小化预测框和目标框之间的归一化距离？

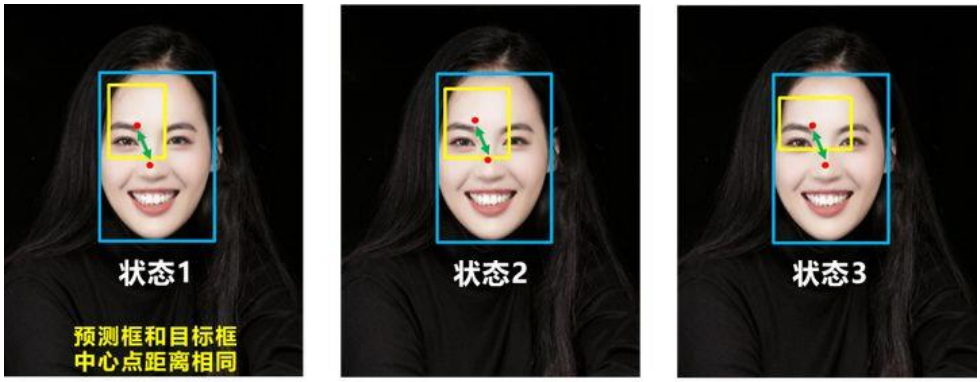
二：如何在预测框和目标框重叠时，回归的更准确？

针对第一个问题，提出了 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。



比如上面三种情况，目标框包裹预测框，本来 DIOU_Loss 可以起作用。但预测框的中心点的位置都是一样的，因此按照 DIOU_Loss 的计算公式，三者的值都是相同的。

针对这个问题，又提出了 CIOU_Loss，不对不说，科学总是在解决问题中，不断进步!!

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。

$$CIOU\_Loss=1-CIOU=1-\left(IOU-\frac{Distance\_2^2}{Distance\_C^2}-\frac{\nu^2}{(1-IOU)+\nu}\right)$$

其中 v 是衡量长宽比一致性的参数，我们也可以定义为：

$$\nu=\frac{4}{\pi^2}\left(\arctan\frac{w^{gt}}{h^{gt}}-\arctan\frac{w^{p}}{h^{p}}\right)^2$$

这样 CIOU_Loss 就将目标框回归函数应该考虑三个重要几何因素：重叠面积、中心点距离，长宽比全都考虑进去了。

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。

将其中计算 IOU 的部分替换成 DIOU 的方式。

总体来说，YOLOv4 的论文称的上良心之作，将近几年关于深度学习领域最新研究的 tricks 移植到 Yolov4 中做验证测试，将 Yolov3 的精度提高了不少。虽然没有全新的创新，但很多改进之处都值得借鉴，借用 Yolov4 作者的总结。

Yolov4 主要带来了 3 点新贡献：

（1）提出了一种高效而强大的目标检测模型，使用 1080Ti 或 2080Ti 就能训练出超快、准确的目标检测器。

（2）在检测器训练过程中，验证了最先进的一些研究成果对目标检测器的影响。

（3）改进了 SOTA 方法，使其更有效、更适合单 GPU 训练。