

# 2D Object Detections of YOLOvx 2D目标检测之YOLO系列

2023/04/24 22:50 基于一系列文章综合整理用于学习使用，不用商业使用！

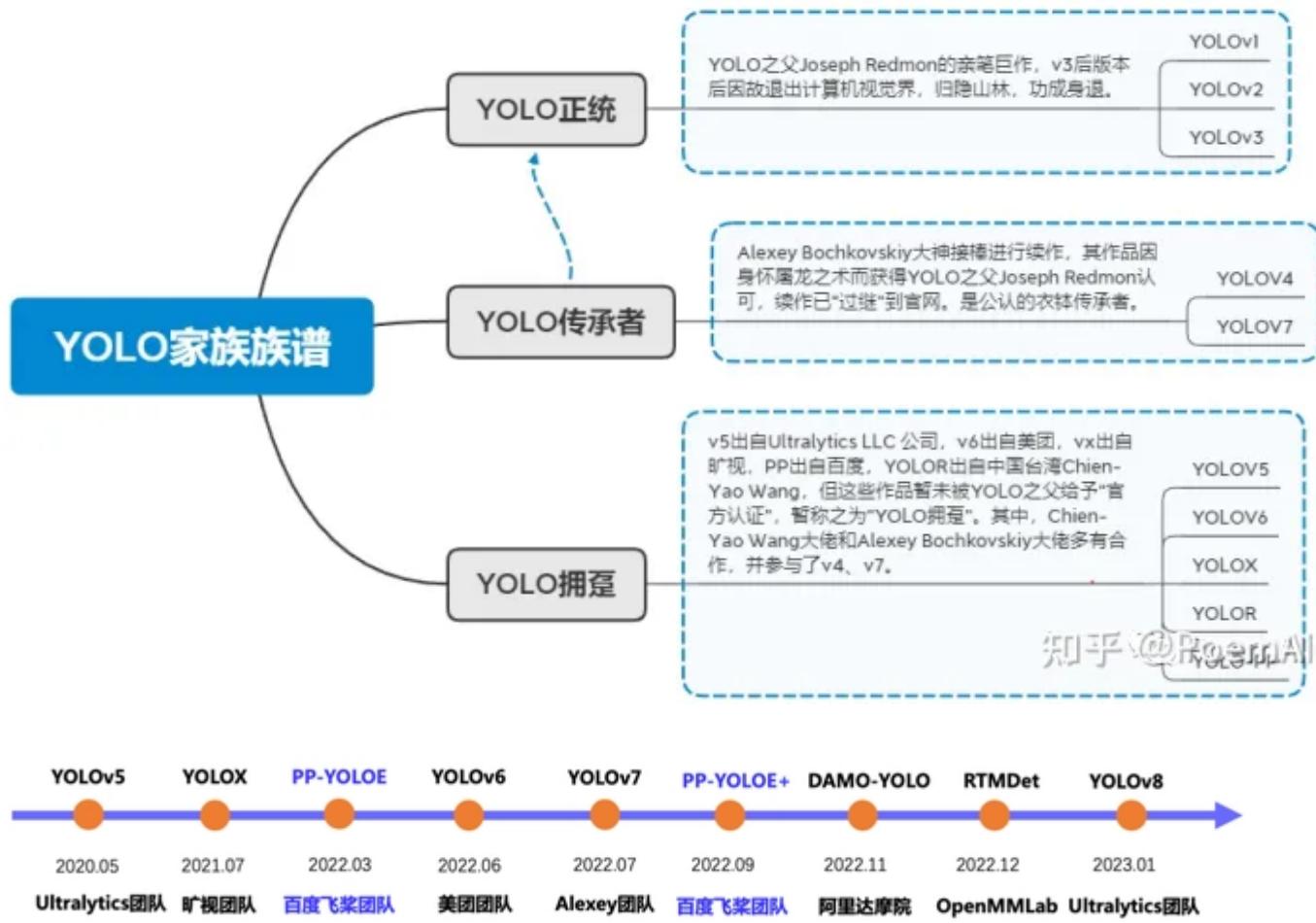
- 2D Object Detections of YOLOvx 2D目标检测之YOLO系列
  - 前言
    - 家族演化
    - 评价指标
      - IOU
      - Precision (精度) 、 Recall (召回率)
      - mAP
    - 双阶段 (two-stage) 的R-CNN系列算法
    - YOLOv1 vs Faster R-CNN
  - YOLOv1: You Only Look Once: Unified, Real-Time Object Detection
    - 核心思想
    - 实现方法
    - 网络模型
    - 损失函数
    - 损失函数 (四部分组成)
    - NMS非极大值抑制
    - 优缺特点
  - YOLOv2: YOLO9000: Better, Faster, Stronger
    - 优缺特点
    - 核心思想
    - 网络模型
    - 网络改进
      - 1. Batch Normalization
      - 2. High resolution classifier
      - 3. Convolution with anchor boxes
      - 4. Dimension clusters
      - 5. Direct location prediction
      - 6. Fine-Grained Features
      - 7. Multi-ScaleTraining
      - 8. Darknet-19

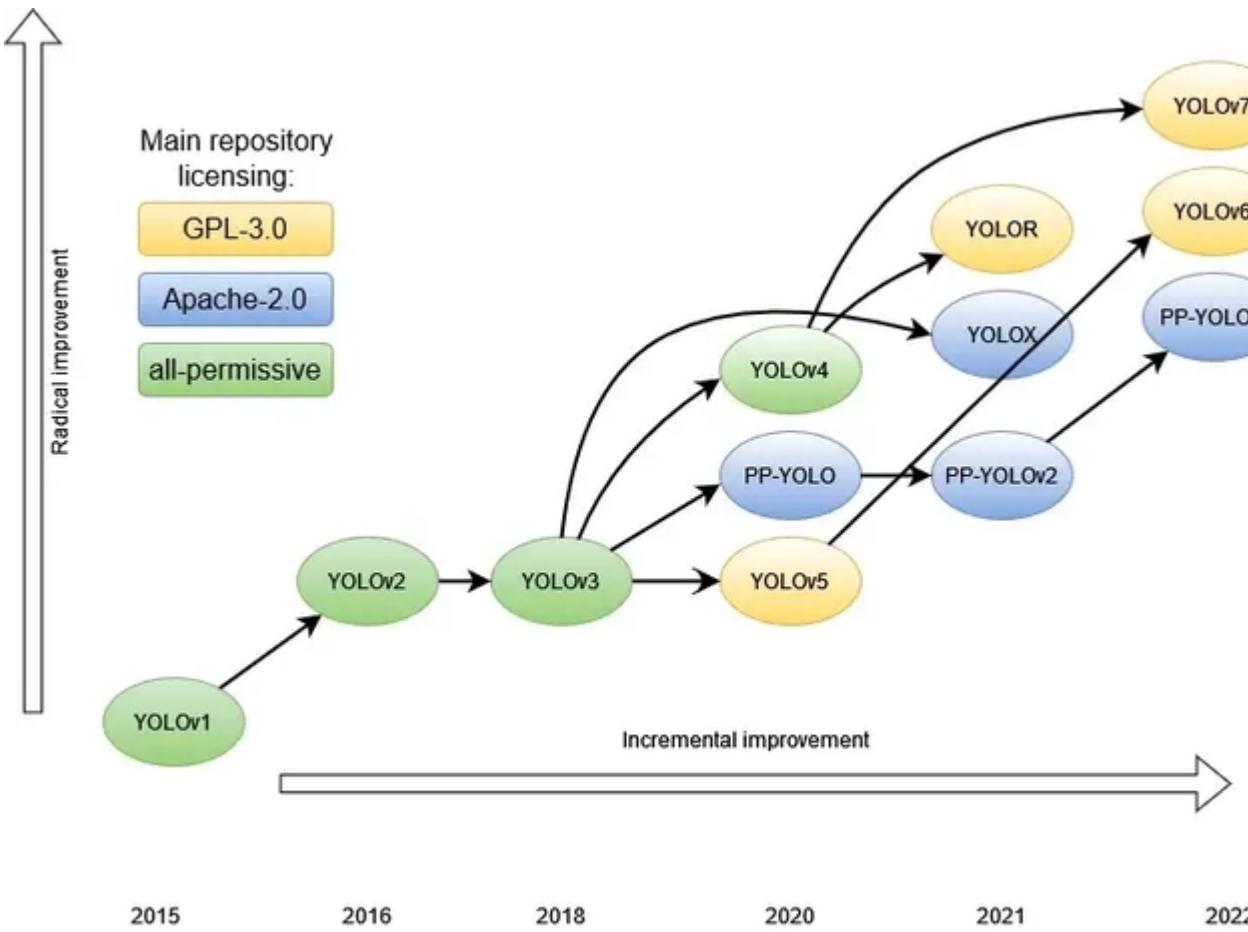
- 9. Hierarchical classification
- YOLOv3: An Incremental Improvement
  - 核心思想
  - 网络模型
  - 网络改进
    - 1. Darknet-53
    - 2. 边框回归
    - 3. 多尺度预测：更好地对应不同大小的目标物体
    - 4. ResNet残差结构：更好地获取物体特征
    - 5. 替换softmax层：对应多重label分类
  - 常见问题
    - YOLOv3为什么不使用Softmax对每个框进行分类？
    - 分类损失采用 binary cross-entropy loss
- YOLOv4: Optimal Speed and Accuracy of Object Detection
  - 核心思想
  - 网络模型
  - 网络改进
    - 1. BackBone:CSPDarknet53
    - 2. Neck:SPP+PAN
    - 3. CutMix数据增强和马赛克（Mosaic）数据增强
    - 4. DropBlock正则化
    - 5. BackBone推理策略
      - Mish激活函数
      - MiWRC策略
    - 6. 检测头训练策略
      - CIoU-loss
      - CmBN策略
      - 自对抗训练(SAT)
      - 消除网格敏感度
      - 余弦模拟退火
    - 7. 检测头推理策略
      - SAM模块
      - DIoU-NMS
    - 8. 标签平滑（Label Smoothing）
- YOLOv5
  - 算法简介
  - 优缺特点
  - 网络模型

- YOLOv5基础组件
- Backbone (特征提取模块)
- 基准网络细节详解
- Neck网络细节详解
- 改进之处
- YOLOX: Exceeding YOLO Series in 2021
  - 算法简介
  - 网络改进
- YOLOv6
  - 算法简介
  - 算法改进
    - Hardware-friendly 的骨干网络设计
    - 更简洁高效的 Decoupled Head
    - 更有效的训练策略
      - Anchor-free 无锚范式
      - SimOTA 标签分配策略
      - SIoU 边界框回归损失
- YOLOv7
  - 性能表现
  - 网络模型
    - Backbone
    - Head
    - Loss Function
  - 算法改进
    - RepVGG (最大改进)
    - 正样本分配策略
    - 相对偏移量计算 (yolov5/v7版)
    - 辅助头 (auxiliary head) +主头 (lead head)
- YOLOv8
  - 算法改进
  - 网络模型
    - 1.C3和C2F
    - 2.PAN-FPN
    - 3.Head
    - 4.损失函数
    - 5.样本的匹配

# 前言

## 家族演化





YOLO系列是one-stage且是基于深度学习的回归方法，而R-CNN、Fast-RCNN、Faster-RCNN等是two-stage且是基于深度学习的分类方法。

## 评价指标

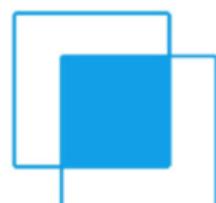
### IOU

💡 IOU:

- Ground truth
- Prediction



$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



CSDN @胖墩会武术

Precision (精度) 、 Recall (召回率)

**Precision** =  $\frac{TP}{TP + FP}$

**Recall** =  $\frac{TP}{TP + FN}$

T (true) 表示分类目标  
P (positives) 表示识别正确  
TP表示分类目标，成功识别为分类目标  
TN表示分类目标，错误识别为非分类目标

F (false) 表示非分类目标  
N (negatives) 表示识别失败  
FP表示非分类目标，错误识别为分类目标  
FN表示非分类目标，错误识别为非分类目标

**置信度**（即只有IOU大于阈值时，检测框才有效）

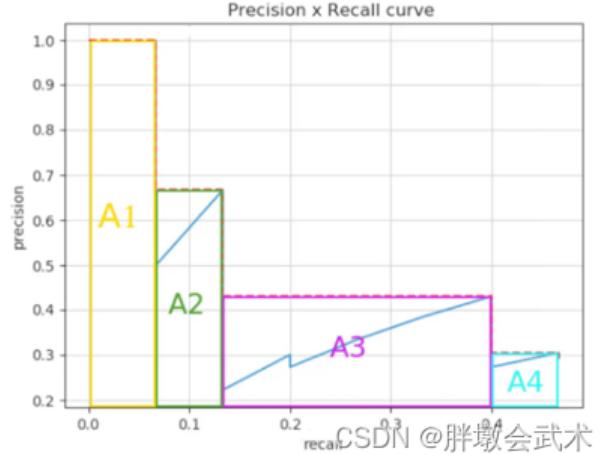
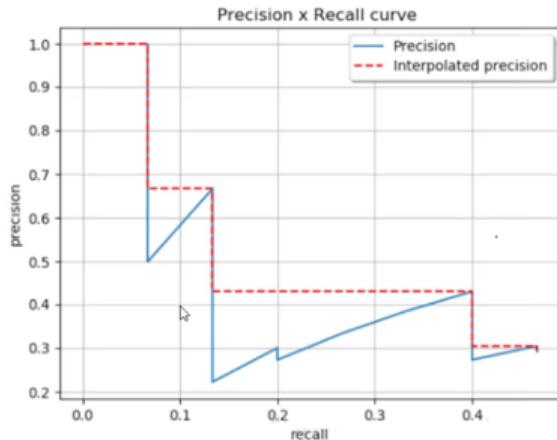
基于置信度阈值计算精度（Precision）和召回率（Recall）。下图蓝色表示真实框，绿色表示预测框。当阈值=0.9，只有第一张图有效。TP=1、FP=0、FN=2。Precision=1/1；Recall=1/3；



CSDN @胖墩会武术

**mAP**

## map指标：综合衡量检测效果



## 双阶段（two-stage）的R-CNN系列算法

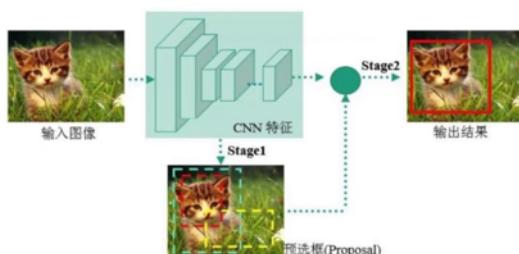
- DPM, R-CNN, Fast-RCNN, Faster-RCNN, SSD

yolov之前，双阶段（two-stage）的R-CNN系列算法在目标检测领域独占鳌头。先利用RPN网络进行感兴趣区域的生成，再对该区域进行分类与位置的回归。

优缺点：提升了精度，但限制了检测速度。2016年，单阶段（one-stage）的YOLO（You Only Look Once）初出茅庐。利用CNN卷积神经网络进行特征提取，并识别种类和位置。

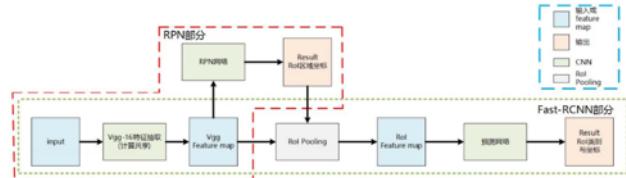
## ✓ 深度学习经典检测方法

### 📝 two-stage (两阶段) : Faster-rcnn Mask-Rcnn系列

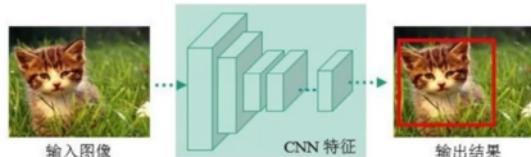


📝 速度通常较慢 (5FPS)，但是效果通常还是不错的！

📝 非常实用的通用框架MaskRcnn，建议熟悉下！



### 📝 one-stage (单阶段) : YOLO系列



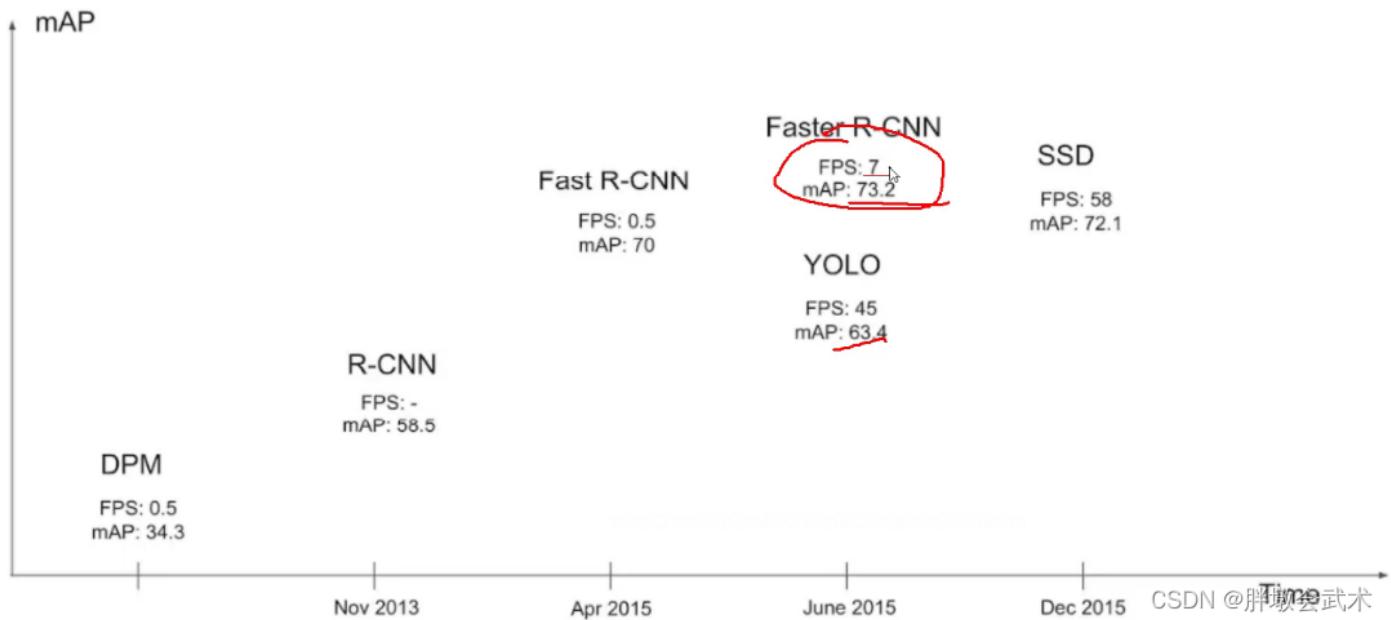
📝 最核心的优势：速度非常快，适合做实时检测任务！

📝 但是缺点也是有的，效果通常情况下不会太好！

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46	link	
SSD500	COCO trainval	test-dev	46.5	-	19	link	
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	-	-	7.67 Bn	200	cfg	weights

CSDN @胖墩会武术

**备注** : FPS是指视频每秒传输的帧数。例如: FPS=45 表示为45帧/秒。帧数愈多，所显示的动作就会越流畅。



### YOLOv1 vs Faster R-CNN

1、统一网络：YOLO没有显示求取region proposal的过程。Faster R-CNN中尽管RPN与fast rcnn共享卷积层，但是在模型训练过程中，需要反复训练RPN网络和fast rcnn网络。相对于R-CNN系列的“看两眼”(候选框提取与分类)，YOLO只需要Look Once.

2、YOLO统一为一个回归问题，而Faster R-CNN将检测结果分为两部分求解：物体类别（分类问题）、物体位置即bounding box（回归问题）。

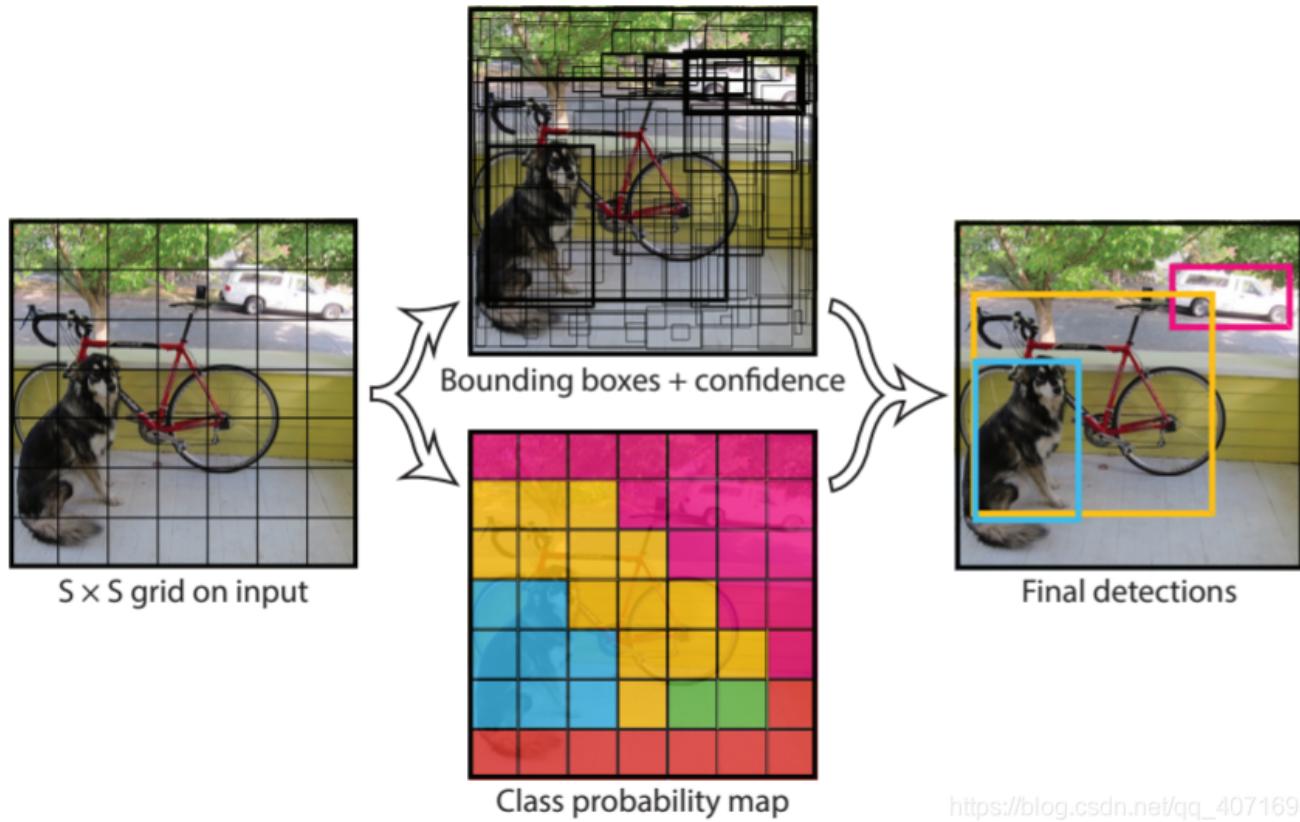
# YOLOv1: You Only Look Once: Unified, Real-Time Object Detection

[论文下载](#), [开源代码](#)

## 核心思想

- **yolo核心思想：**把目标检测转变成一个回归问题。将整个图像作为网络的输入，仅  
仅经过一个神经网络，得到边界框的位置及其所属的类别。
- YOLOv1的核心思想就是利用整张图作为网络的输入，直接在输出层回归bounding  
box的位置和bounding box所属的类别。
- Faster RCNN中也直接用整张图作为输入，但是Faster-RCNN整体还是采用了RCNN  
那种 proposal+classifier的思想，只不过是将提取proposal的步骤放在CNN中实现  
了，而YOLOv1则采用直接回归的思路。

## 实现方法



[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

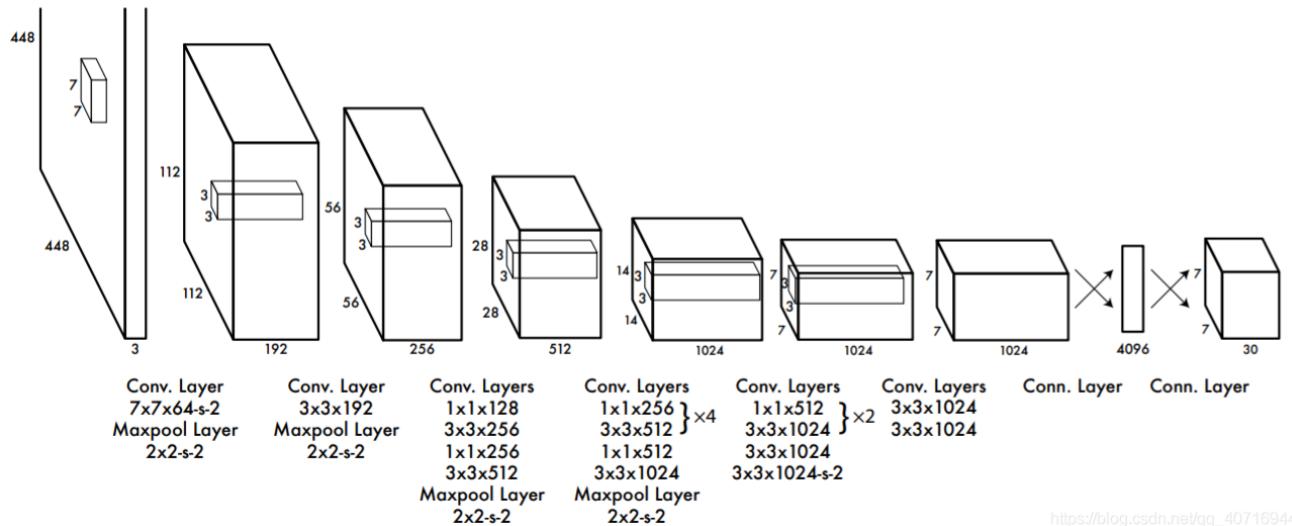
- 将一幅图像分成 $S \times S$ 个网格(grid cell)，如果某个object的中心落在这个网格中，则这  
个网格就负责预测这个object。
  - 给一个输入图像，首先将图像划分成 $7 \times 7$  的网格

- 对于每个网格，我们都预测2个边框（包括每个边框是目标的置信度以及每个边框区域在多个类别上的概率）
- 根据上一步可以预测出 $7 \times 7 \times 2$ 个目标窗口，然后根据阈值去除可能性比较低的目标窗口，最后NMS去除冗余窗口即可
- 每个网格要预测B个bounding box，每个bounding box除了要回归自身的位置之外，还要附带预测一个confidence值。这个confidence代表了所预测的box中含有object的置信度和这个box预测的有多准两重信息，其值是这样计算的：

$$\text{Pr}(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

该表达式含义：如果有object落在一个grid cell里，则第一项取1，否则取0。第二项是预测的bounding box和实际的groundtruth之间的IoU值。

- 每个bounding box要预测(x, y, w, h)和confidence共5个值，每个网格还要预测一个类别信息，记为C类。则 $S \times S$ 个网格，每个网格要预测B个bounding box还要预测C个categories。输出就是 $S \times S \times (5 \times B + C)$ 的一个tensor。
- 注意：class信息是针对每个网格的，confidence信息是针对每个bounding box的。**



[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

- 在test的时候，每个网格预测的class信息和bounding box预测的confidence信息相乘，就得到每个bounding box的class-specific confidence score，得到每个box的class-specific confidence score以后，设置阈值，滤掉得分低的boxes，对保留的boxes进行NMS处理，就得到最终的检测结果。

$$\Pr(\text{Class}_i | \text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

- 该表达式含义：等式左边第一项就是每个网格预测的类别信息，第二三项就是每个 bounding box预测的confidence。这个乘积即encode了预测的box属于某一类的概率，也有该box准确度的信息。
- 注意：

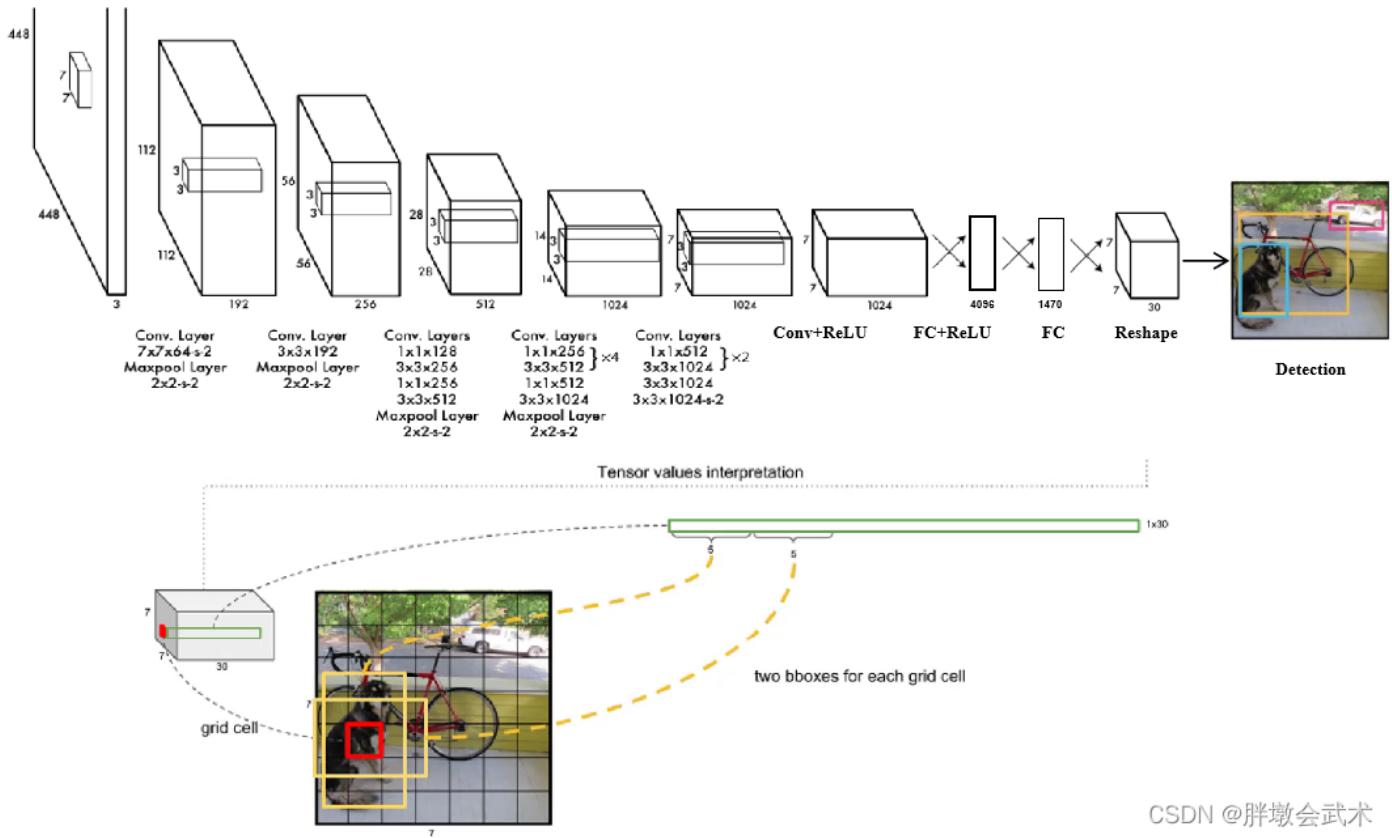
- 由于输出层为全连接层，因此在检测时，YOLOv1模型的输入只支持与训练图像相同的输入分辨率。
- 虽然每个格子可以预测B个bounding box，但是最终只选择IOU最高的 bounding box作为物体检测输出，即每个格子最多只预测出一个物体。当物体占画面比例较小，如图像中包含畜群或鸟群时，每个格子包含多个物体，但却只能检测出其中一个。

- 简单的概括就是：**

- 给一个输入图像，首先将图像划分成7\*7的网格
- 对于每个网格，我们都预测2个边框（包括每个边框是目标的置信度以及每个边框区域在多个类别上的概率）
- 根据上一步可以预测出7 \* 7 \* 2个目标窗口，然后根据阈值去除可能性比较低的目标窗口，最后NMS去除冗余窗口即可

## 网络模型

- 输入图像的尺寸固定为448×448（与全连接层的输出大小有关），经过24个卷积与2个全连接层后，最后输出的特征图为7×7×30。
  - 在3×3的卷积后接1×1卷积，既降低了计算量，也提升了模型的非线性能力。
  - 除最后一层使用线性激活函数外，其余层都使用ReLU激活函数。
  - 在训练中使用Dropout与数据增强的方法来防止过拟合。
  - 训练时的图像尺寸：224×224；测试时的图像尺寸：448×448。原因：224×224×3相比448×448×3相差四倍，其像素点大幅度降低，减少对计算机的性能要求。



### • 最大创新：7x7x30特征图

在整张图中，共预测 $7 \times 7 \times 2 = 98$ 个边框，每个边框的大小与位置都不相同。

- 类别概率：由于PASCAL VOC数据集共有20个物体类别，因此预测的每个边框都会得到20个类别的概率值。
- 置信度（confidence）：表示该网格内是否包含物体的概率（前景、背景）。两个边框得到两个置信度预测值。
- 边框位置：每一个边框需要预测四个值：中心坐标（x, y）、宽w、高h。两个边框得到8个预测值。
- 将输入图像划分为7x7的网格，每个网格预测2个边框（bounding box），每个边框得到五个值（x, y, w, h, confidence）。论文采用20分类，故最后得到通道数为 $5 \times 2 + 20 = 30$ ，代表每个网格预测了30个特征。

CSDN @胖墩会武术

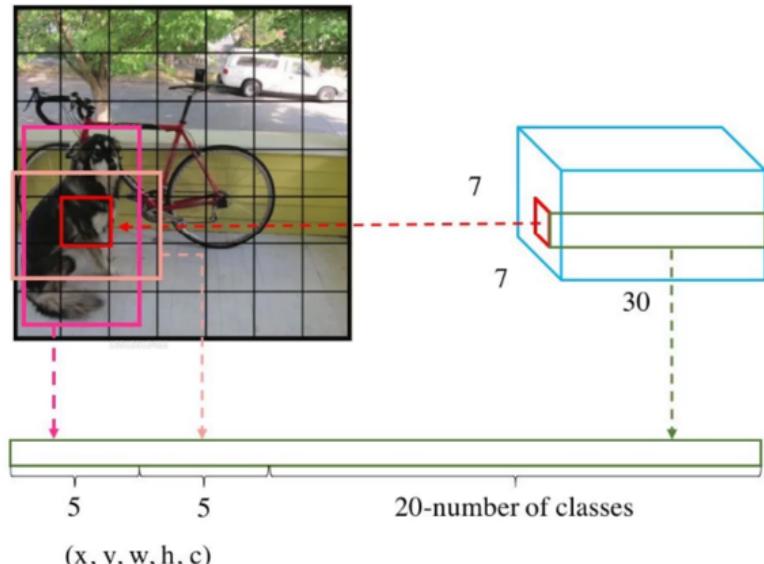
✓ 每个数字的含义：

10=(x,y,w,h,c)\*B, B=2

当前数据集类别C=20

最终网格大小为7\*7

7\*7\*30=(S\*S) \* (B\*(x,y,w,h,c)+C)

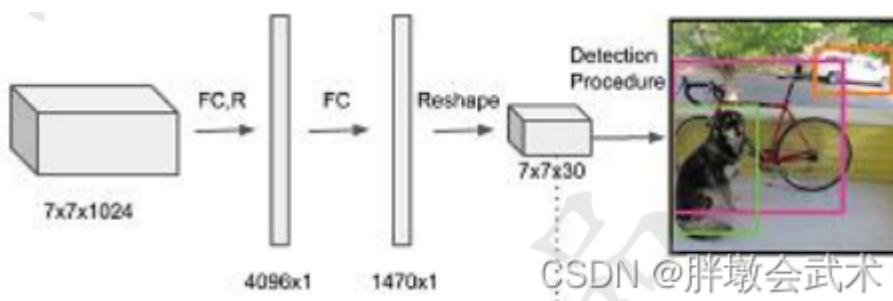


yolov1将图像划分为 $7 \times 7 = 49$ 个网格，每个网格有**2个bounding box**。每个bounding box预测(x,y,w,h)和confidence共5个值。

CSDN @胖墩会武术

- 连续使用两个全连接层的作用

- 第一个全连接层作用：将卷积得到的分布式特征映射到样本标记空间。即把该输入图像的所有卷积特征整合到一起。
- 第二个全连接层作用：将所有神经元得到的卷积特征进行维度转换，最后得到与目标检测网络输出维度相同的维度。



- 【小问题思考】两个全连接层连用 1x1 卷积作用

## 损失函数

每个grid有30维，这30维中，8维是回归box的坐标，2维是box的confidence，还有20维是类别。其中坐标的x,y用对应网格的offset归一化到0-1之间，w,h用图像的width和height归一化到0-1之间。在实现中，最主要的就是怎么设计损失函数，让这个三个方面得到很好的平衡。作者简单粗暴的全部采用了sum-squared error loss来做这件事。

- 这种做法存在以下几个问题：

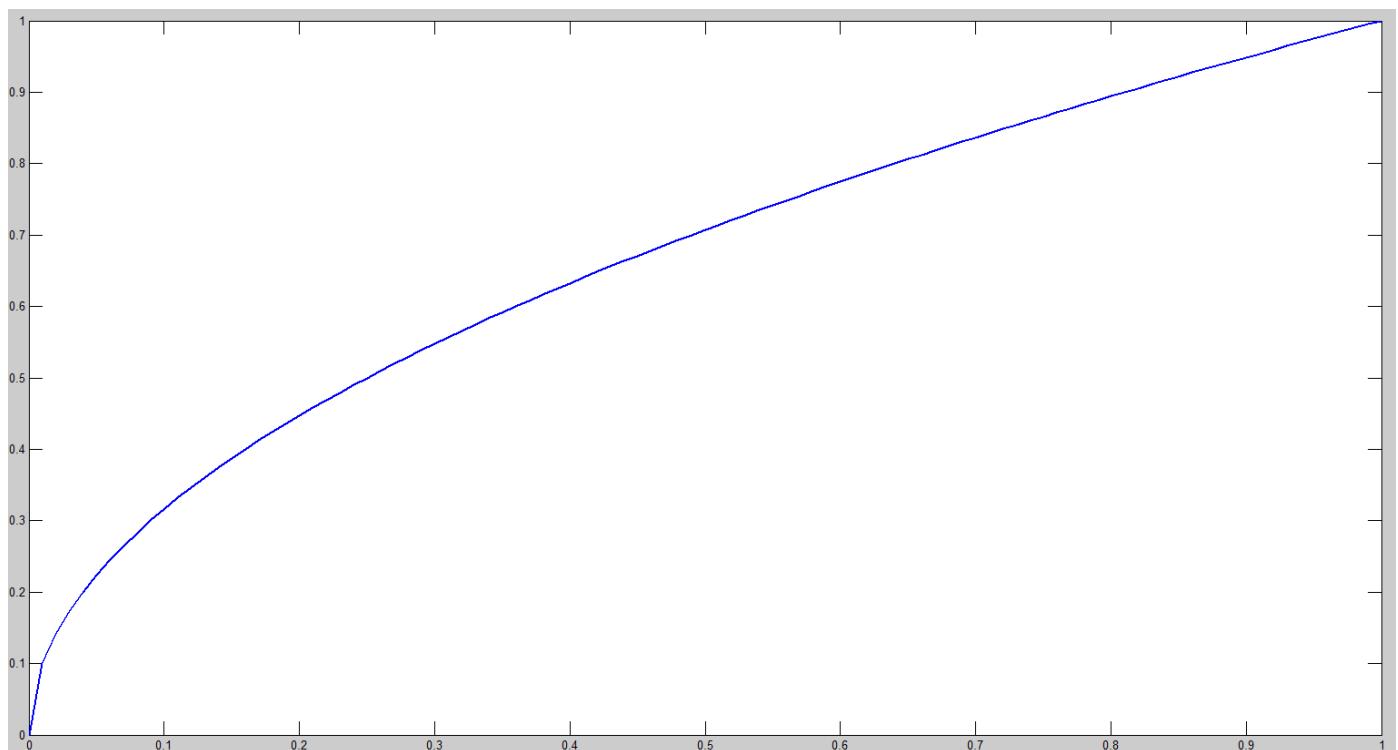
- 第一，8维的localization error和20维的classification error同等重要显然是不合理的
- 第二，如果一个网格中没有object（一幅图中这种网格很多），那么就会将这些网格中的box的confidence push到0，相比于较少的有object的网格，这种做

法是overpowering的，这会导致网络不稳定甚至发散

- **解决方法：**

- 更重视8维的坐标预测，给这些损失前面赋予更大的loss weight；
- 对没有object的box的confidence loss，赋予小的loss weight；
- 有object的box的confidence loss和类别的loss的loss weight正常取1。

YOLOv1在对不同大小的box预测中，相比于大box预测偏一点，小box预测偏一点肯定更不能被忍受的。而sum-square error loss中对同样的偏移loss是一样。为了缓和这个问题，作者用了一个比较取巧的办法，就是将box的width和height取平方根代替原本的height和width。这个参考下面的图很容易理解，小box的横轴值较小，发生偏移时，反应到y轴上相比大box要大。(也是个近似逼近方式)



一个网格预测多个box，希望每个box predictor专门负责预测某个object。具体做法就是看当前预测的box与ground truth box中哪个IoU大，就负责哪个。这种做法称作box predictor的specialization。

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2$$

坐标预测

判断第*i*个网格中的第*j*个  
box是否负责这个object

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2$$

含object的box的  
confidence预测

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2$$

不含object的box的  
confidence预测

判断是否有object中  
心落在网格*i*中

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

类别预测

在YOLOv1的损失函数中：

只有当某个网格中有object的时候才对classification error进行惩罚。只有当某个box predictor对某个ground truth box负责的时候，才会对box的coordinate error进行惩罚，而对哪个ground truth box负责就看其预测值和ground truth box的IoU是不是在那个cell的所有box中最大。

- 注意：

- YOLOv1方法模型训练依赖于物体识别标注数据，因此，对于非常规的物体形状或比例，YOLOv1的检测效果并不理想。
- YOLOv1采用了多个下采样层，网络学到的物体特征并不精细，因此也会影响检测效果。
- YOLOv1的loss函数中，大物体IoU误差和小物体IoU误差对网络训练中loss贡献值接近（虽然采用求平方根方式，但没有根本解决问题）。因此，对于小物体，小的IoU误差也会对网络优化过程造成很大的影响，从而降低了物体检测的定位准确性。

## 损失函数（四部分组成）

通过卷积神经网络得到每个边框的预测值后，需要确定每个边框对应的是前景框（真实物体）还是背景框（无关物体），即区分正样本、负样本

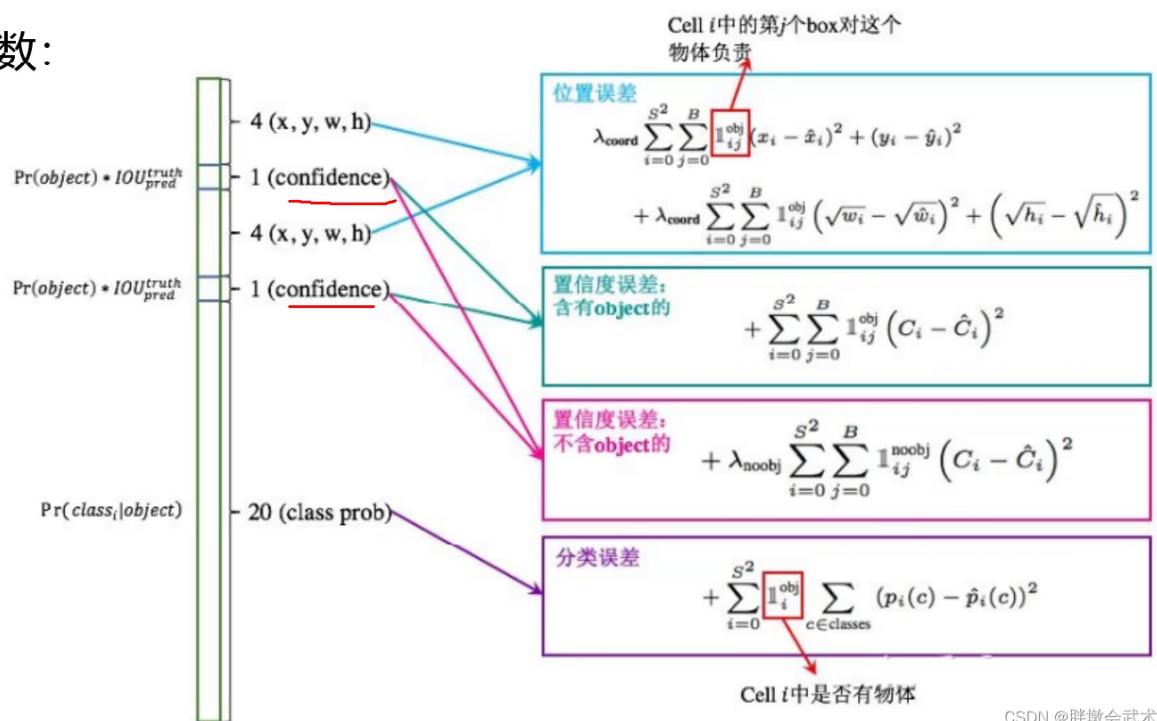
- 正样本：将与真实物体有最大IoU的边框设为正样本，每个边框的置信度都为1。

- 负样本：其余边框为负样本，置信度全为0。负样本没有类别损失与边框位置损失，只有置信度损失。

损失函数由 4 个部分组成，均使用均方差损失。共有  $S^2$  个区域 ( $7 \times 7 = 49$ )， $B$  表示每个网格有 2 个边框， $obj$  表示对应真实物体， $noobj$  表示没有对应真实物体。

- (1) 位置误差：提取每个网格的两个边框中，IoU最大的一个边框，并计算该边框的预测值与真实值的位置误差。前一个计算正样中心点坐标的损失，后一个计算正样本宽和高的损失。其中： $\lambda_{coord}$ 用于调整位置误差的权重。由于宽高差值受物体尺寸的影响，因此对w和h进行平方根处理，降低对物体尺的敏感度，强化小物体的损失权重。
  - (2) \*\*置信度误差 (obj)：前景误差。\*\*计算边框与正样本的误差。若边框的IoU大于置信度阈值，则该边界框属于前景。若存在多个满足要求的边框，则进行非极大值抑制。我们希望前景框的误差趋近于1。
  - (3) \*\*置信度误差 (noobj)：背景误差。\*\*计算边框与负样本的误差。若边框的IoU小于置信度阈值或IoU=0，则该边界框属于背景。我们希望背景框的误差趋近于0。其中： $\lambda_{noobj}$ 用于调整负样本置信度损失的权重（默认为0.5）。由于背景框的数量远远大于前景框，故对背景框误差设置阈值（如：0.1），降低背景框误差对损失函数的影响。
  - (4) 分类误差：计算每个边框得到的20个分类概率值与正样本的误差。

## ✓ 损失函数：

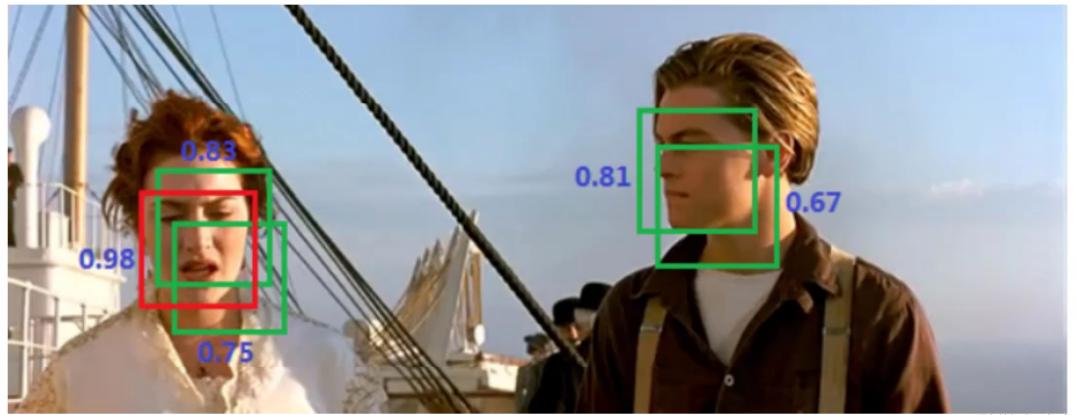


## NMS非极大值抑制

**非极大值抑制**可以用来修正多重检测目标，能增加2~3%的mAP。即在检测结果中，若存在多个检测框的IOU大于置信度阈值，通过非极大值抑制最后只取一个框。如下图：

五个框中只取最大值（置信度=0.98）的预测框。

## ✓ NMS(非极大值抑制)



CSDN @群墩会武术

## 优缺点

### • 优点

- 快速， pipeline简单
- YOLO检测速度非常快。标准版本的YOLO可以每秒处理 45 帧图像，极速版本可以每秒处理 150 帧图像，完全满足视频的实时检测要求；而对于欠实时系统，在保证准确率的情况下，速度也快于其他方法。
- 背景误检率低
- YOLO 实时检测的平均精度是其他实时监测系统的两倍。
- 迁移能力强，能运用到其他的新的领域（比如：艺术品目标检测）。
- 通用性强，**YOLO v1**对于艺术类作品中的物体检测同样适用。它对非自然图像物体的检测率远远高于 **DPM**和 **RCNN**系列检测方法

### • 缺点

- YOLO对相互靠的很近的物体和很小的群体检测效果不好，这是因为一个网格中只预测了两个框，并且只属于一类。
- 由于每个网格只有两个边框做预测，并且只有一个类别。导致对于小物体以及靠的特别近的物体，检测效果不好。
- 虽然每个格子可以预测B BB个bounding box，但是最终只选择只选择IOU最高的bounding box作为物体检测输出，即每个格子最多只预测出一个物体。当物体占画面比例较小，如图像中包含畜群或鸟群时，每个格子包含多个物体，但却只能检测出其中一个。这是YOLOv1方法的一个缺陷。
- 同一类物体出现的新的不常见的长宽比和其他情况时，泛化能力偏弱

- 由于没有类似于Anchor的先验框，对于新物体或宽高比例不常见的物体，检测效果不好。
- YOLO v1 loss函数中，大物体IOU误差和小物体IOU误差对网络训练中loss贡献值接近（虽然采用求平方根方式，但没有根本解决问题）。因此，对于小物体，小的IOU误差也会对网络优化过程造成很大的影响，从而降低了物体检测的定位准确性。
- 在损失函数中，大物体与小物体的物质损失权重是一样的，导致同等比例的位置误差，大物体的损失会比小物体大，导致物体定位的不准确。
- 由于损失函数的问题，定位误差是影响检测效果的主要原因。尤其是大小物体的处理上，还有待加强
- 由于输出层为全连接层，因此在检测时，**YOLOv1**训练模型只支持与训练图像相同的输入分辨率

## YOLOv2: YOLO9000: Better, Faster, Stronger

论文下载：<https://arxiv.org/abs/1612.08242>

代码下载：[https://github.com/yjh0410/yolov2-yolov3\\_PyTorch](https://github.com/yjh0410/yolov2-yolov3_PyTorch)

官方代码：[YOLO: Real-Time Object Detection](https://pjreddie.com/darknet/yolo/)

官方代码：<http://pjreddie.com/darknet/yolo/>

### 优缺点

- 2017年，提出了yolov2和yolo9000，yolo9000能够实时检测超过9000种物体，主要检测网络还是yolov2。yolov2的整体网络架构和基本思想没有变化，重点解决yolov1召回率和定位精度方面的不足。相比其它的检测器，速度更快、精度更高、可以适应多种尺寸的图像输入。
- yolov1是利用全连接层直接预测Bounding Box的坐标。而yolov2借鉴了Faster R-CNN的思想，引入Anchor机制；利用K-means聚类的方法在训练集中聚类计算出更好的Anchor模板，大大提高了算法的召回率；同时结合图像细粒度特征，将浅层特征与深层特征相连，有助于对小尺寸目标的检测

### 核心思想

- 该论文使用一种新颖的多尺度训练方法，YOLOv2可以在不同的尺寸下运行，在速度和精度之间提供了一个简单的权衡。在67FPS的情况下，YOLOv2在VOC 2007上获得76.8mAP。在40FPS的情况下，YOLOv2获得78.6 mAP。

- 最后提出了一种目标检测与分类的联合训练方法，利用该方法在COCO检测数据集和ImageNet分类数据集上同时训练YOLO9000，它可以预测9000多种不同的物体类别的检测结果。

## 网络模型

YOLOv2相对v1版本，在继续保持处理速度的基础上，从预测更准确（Better），速度更快（Faster），**识别对象更多（Stronger）**这三个方面进行了改进。其中识别更多对象也就是扩展到能够检测9000种不同对象，称之为YOLO9000。

文章提出了一种新的训练方法—联合训练算法，这种算法可以把这两种的数据集混合到一起。使用一种分层的观点对物体进行分类，用巨量的分类数据集数据来扩充检测数据集，从而把两种不同的数据集混合起来。联合训练算法的基本思路就是：同时在检测数据集和分类数据集上训练物体检测器（Object Detectors），用检测数据集的数据学习物体的准确位置，用分类数据集的数据来增加分类的类别量、提升健壮性。

YOLO9000就是使用联合训练算法训练出来的，他拥有9000类的分类信息，这些分类信息学习自ImageNet分类数据集，而物体位置检测则学习自COCO检测数据集。

Darknet-19采用了19个卷积层，5个池化层。

- (1) 取消yolov1的两个全连接层。yolov1的依据全连接层直接预测 Bounding Boxes 的坐标值。而yolov2采用 Faster R-CNN 的方法，只用卷积层与 Region Proposal Network 来预测 Anchor Box 偏移量与置信度，而不是直接预测坐标值。
- (2) 添加了五个最大池化层（2的5次方）。最终的输出大小：输入图像（h, w）转换为（h / 32, w / 32）。
- (3) yolov2的实际输入图像大小为416×416，而不是448×448（416/32=13、448/32=14）。因为我们希望最后得到的是奇数值，有实际的中心点。最终得到13×13的输出。与yolov1的7×7相比，可以预测更多的先验框。
- (4) 基于VGG的思想，大部分的卷积核都是3×3，一方面权重参数少，一方面感受野比较大；且采用降维的思想，将1×1的卷积核置于3×3之间，在保持整体网络结构的同时减少权重参数。并且每一次池化后，下一层卷积核的通道数 = 池化输出的通道 × 2。
- (5) 在网络模型的最后，而增加了一个全局平均池化层。

Type	Filters	Size/Stride	Output
Convolutional	32	$3 \times 3$	$224 \times 224$
Maxpool		$2 \times 2/2$	$112 \times 112$
Convolutional	64	$3 \times 3$	$112 \times 112$
Maxpool		$2 \times 2/2$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Convolutional	64	$1 \times 1$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Maxpool		$2 \times 2/2$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Convolutional	128	$1 \times 1$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Maxpool		$2 \times 2/2$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Maxpool		$2 \times 2/2$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$

Convolutional	1000	$1 \times 1$	$7 \times 7$
Avgpool		Global	1000
Softmax		CSDN @胖墩会武术	

## Darknet-19 与 yolov1、VGG16网络的性能对比

- (1) VGG-16：大多数检测网络框架都是以VGG-16作为基础特征提取器，它功能强大，准确率高，但是计算复杂度较大，所以速度会相对较慢。因此YOLOv2的网络结构基于该方面进行改进。
- (2) yolov1：基于GoogLeNet的自定义网络，比VGG-16的速度快，但是精度稍不如VGG-16。
- (3) Darknet-19：速度方面，处理一张图片仅需要55.8亿次运算，相比于VGG的306.9亿次，速度快了近6倍。精度方面，在ImageNet上的测试精度为：top1准确率为72.9%，top5准确率为91.2%。

## 网络改进

YOLOv2相比于 YOLOv1的改进如下：

	YOLO	YOLOv2							
batch norm?	✓	✓	✓	✓	✓	✓	✓	✓	
hi-res classifier?		✓	✓	✓	✓	✓	✓	✓	
convolutional?		✓	✓	✓	✓	✓	✓	✓	
anchor boxes?		✓	✓						
new network?			✓	✓	✓	✓		✓	
dimension priors?				✓	✓	✓		✓	
location prediction?					✓	✓	✓	✓	
passthrough?						✓	✓	✓	
multi-scale?							✓	✓	
hi-res detector?								✓	
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	
								78.6	

## 1. Batch Normalization

mAP提升2.4%

关于BN的详细信息，感兴趣的可以看本人专栏[深度学习基础](#)下的[深度学习七 —— BN & LN & IN & GN](#)这里不多做讲述

批量归一化有助于解决反向传播过程中的梯度消失和梯度爆炸问题，降低对一些超参数（比如学习率、网络参数的大小范围、激活函数的选择）的敏感性，并且每个batch分别进行归一化的时候，起到了一定的正则化效果（YOLOv2不再使用dropout），从而能够获得更好的收敛速度和收敛效果。

使用Batch Normalization对网络进行优化，让网络提高了收敛性，同时还消除了对其他形式的正则化（regularization）的依赖。通过对YOLOv2的每一个卷积层增加Batch Normalization，最终使得mAP提高了2%，同时还使model正则化。使用Batch Normalization可以从model中去掉Dropout，而不会产生过拟合。

关于批规一化的更多信息可以参考：[Batch Normalization原理与实战](#)

**具体操作**：在每一个卷积层后面都加入BN，对数据进行预处理操作（如：统一格式、均衡化、去噪等）。**优点**：解决梯度消失和爆炸问题，起到一定的正则化效果（yolov2不再使用dropout），获得更好的收敛速度。

## 2. High resolution classifier

高分辨率图像分类器，mAP提升了3.7%。

图像分类的训练样本很多，而标注了边框的用于训练对象检测的样本相比而言就比较少了，因为标注边框的人工成本比较高。所以对象检测模型通常都先用图像分类样本训练卷积层，提取图像特征。但这引出的另一个问题是，图像分类样本的分辨率不是很高。所以YOLOv1使用ImageNet的图像分类样本采用  $224 \times 224$  作为输入，来训练CNN卷积

层。然后在训练对象检测时，检测用的图像样本采用更高分辨率的 $448 \times 448$  的图像作为输入。但这样切换对模型性能有一定影响。

所以YOLOv2在采用 $224 \times 224$  图像进行分类模型预训练后，再采用 $448 \times 448$  的高分辨率样本对分类模型进行微调（10个epoch），使网络特征逐渐适应 $448 \times 448$  的分辨率。然后再使用 $448 \times 448$  的检测样本进行训练，缓解了分辨率突然切换造成的影响。最终通过使用高分辨率，mAP提升了4%。

背景：yolov1训练时的分辨率： $224 \times 224$ ；测试时： $448 \times 448$ 。具体操作：yolov2保持yolov1的操作不变，但在原训练的基础上又加上了（10个epoch）的 $448 \times 448$ 高分辨率样本进行微调，使网络特征逐渐适应 $448 \times 448$  的分辨率；然后再使用 $448 \times 448$  的样本进行测试，缓解了分辨率突然切换造成的影响。

### 3. Convolution with anchor boxes

使用先验框，召回率大幅提升到88%，同时mAP轻微下降了0.2左右。

YOLO v1包含有全连接层，从而能直接预测Bounding Boxes的坐标值。Faster R-CNN的方法只用卷积层与Region Proposal Network来预测Anchor Box的偏移值与置信度，而不是直接预测坐标值。作者发现通过预测偏移量而不是坐标值能够简化问题，让神经网络学习起来更容易。

借鉴Faster RCNN的做法，YOLOv2也尝试采用先验框（anchor）。在每个grid预先设定一组不同大小和宽高比的边框，来覆盖整个图像的不同位置和多种尺度，这些先验框作为预定义的候选区在神经网络中将检测其中是否存在对象，以及微调边框的位置。

之前YOLOv1并没有采用先验框，并且每个grid只预测2个bounding box，整个图像98个。YOLOv2如果每个grid采用9 99个先验框，总共有 $13 \times 13 \times 9 = 1521$ 个先验框。所以最终YOLO v2去掉了全连接层，使用Anchor Boxes来预测 Bounding Boxes。作者去掉了网络中一个Pooling层，这让卷积层的输出能有更高的分辨率。收缩网络让其运行在 $416 \times 416$  而不是 $448 \times 448$ 。

由于图片中的物体都倾向于出现在图片的中心位置，特别是那种比较大的物体，所以有一个单独位于物体中心的位置用于预测这些物体。YOLOv2的卷积层采用32这个值来下采样图片，所以通过选择 $416 \times 416$ 用作输入尺寸最终能输出一个 $13 \times 13$ 的Feature Map。使用Anchor Box会让精度稍微下降，但用了它能让YOLO v2能预测出大于一千个框，同时recall从81%达到88%，mAP达到69.2%。

召回率升高，mAP轻微下降的原因是：因为YOLOv2不使用anchor boxes时，每个图像仅预测98个边界框。但是使用anchor boxes，YOLO v2模型预测了一千多个框，由于存在很多无用的框，这就导致了mAP值的下降。但是由于预测的框多了，所以能够预测出

来的属于ground truth的框就多了，所以召回率就增加了。目标检测不是只以mAP为指标的，有些应用场景下要求召回率高。

---

❤️ yolov1边界框都是手工设定的，通过直接对边界框的 (x, y, w, h) 位置进行预测，方法简单但训练困难，很难收敛。 ❤️ Faster R-CNN共有9种先验框：分三个不同的 scale (大中小)，每个scale的 (h, w) 比例分为：1:1、1:2、2:1。

❤️ yolov2引入先验框机制。但由于Faster R-CNN中先验框的大小和比例是按经验设定的，不具有很好的代表性。故yolov2对训练集中所有标注的边界框先进行聚类分析（比如：5类），然后获取每一类的中心值即实际的 (w, h) 比值作为先验框，该值与真实值更接近，使得网络在训练时更容易收敛。备注1：yolov1将图像拆分为 $7 \times 7$ 个网格，每个网格grid只预测2个边界框，共 $7 \times 7 \times 2 = 98$ 个。备注2：yolov2将图像拆分为 $13 \times 13$ 个网格，在Faster R-CNN的9种先验框基础上，将所有的边界框 $13 \times 13 \times 9 = 1521$ 进行K-means聚类，最终选择最优参数k=5。即yolov2的每个网格grid只预测5个边界框，共 $13 \times 13 \times 5 = 845$ 个。

#### 4. Dimension clusters

聚类提取先验框的尺度信息

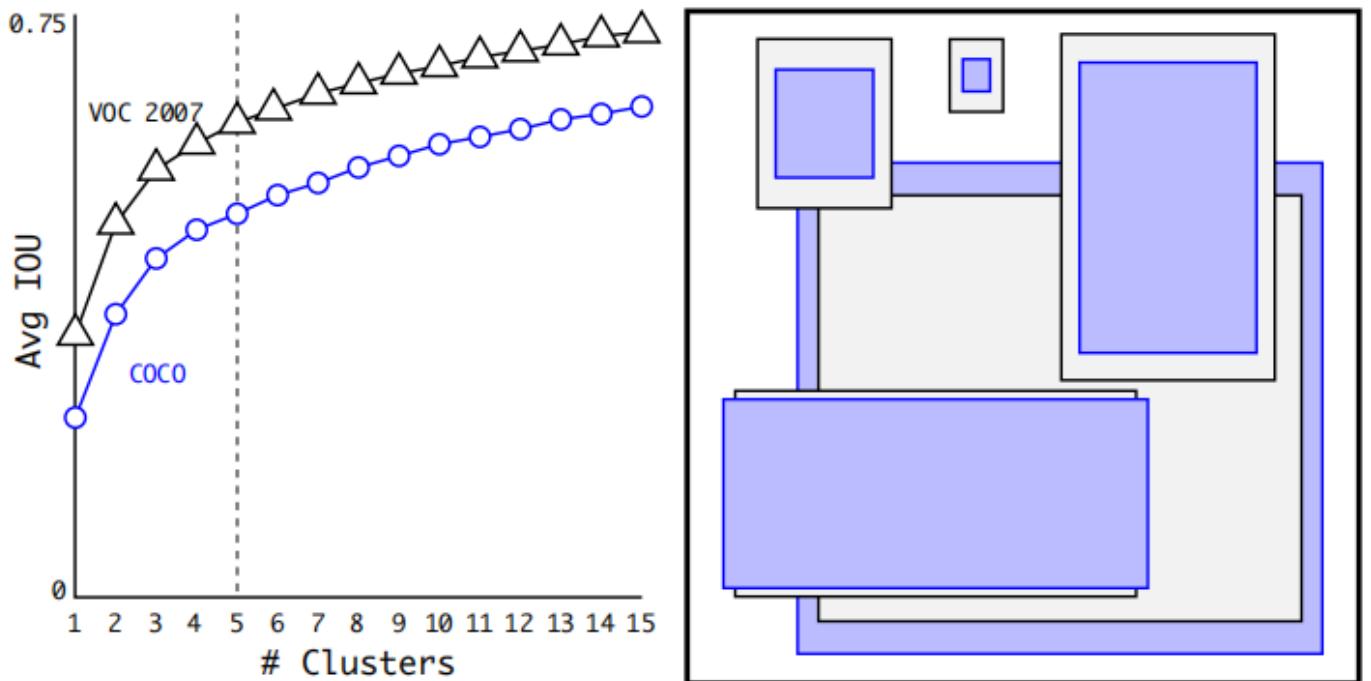
之前Anchor Box的尺寸是手动选择的，所以尺寸还有优化的余地。YOLO v2尝试统计出更符合样本中对象尺寸的先验框，这样就可以减少网络微调先验框到实际位置的难度。YOLO v2的做法是对训练集中标注的边框进行K-mean聚类分析，以寻找尽可能匹配样本的边框尺寸

关于K-Means聚类的详细内容，有需要可以看专栏 图像算法 中的[图像算法六 —— K-Means和KNN](#)

如果我们用标准的欧式距离的k-means，尺寸大的框比小框产生更多的错误。因为我们的目的是提高IOU分数，这依赖于Box的大小，所以距离度量的使用：

$$d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$$

其中，centroid是聚类时被选作中心的边框，box就是其它边框，d就是两者间的“距离”，IOU越大，“距离”越近。YOLOv2给出的聚类分析结果如下图所示，通过分析实验结果（Figure 2），在model复杂性与high recall之间权衡之后，选择聚类分类数K=5。



**Figure 2: Clustering box dimensions on VOC and COCO.** We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for  $k$ . We find that  $k = 5$  gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

Table1是说明用K-means选择Anchor Boxes时，当Cluster IOU选择值为5时，AVG IOU的值是61，这个值要比不用聚类的方法的60.9要高。选择值为9的时候，AVG IOU更有显著提高。总之就是说明用聚类的方法是有效果的。

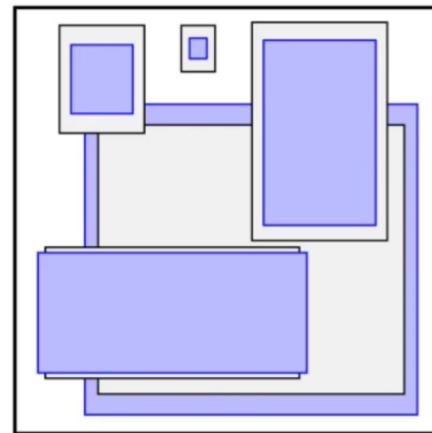
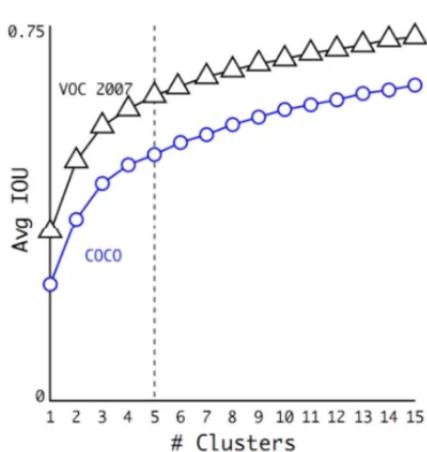
Box Generation	#	Avg IOU
Cluster SSE	5	58.7
Cluster IOU	5	61.0
Anchor Boxes [15]	9	60.9
Cluster IOU	9	67.2

[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

传统K-means聚类方法使用标准的欧氏距离，将导致大的box会比小的box产生更多的误差。而yolo的目的是使得先验框与真实框有更大的IOU值，故自定义距离公式。

**距离公式**：计算每一类的中心值对应的先验框centroids与真实框box的距离。即计算  $IOU = \frac{\text{（先验框与真实框的交集）}}{\text{（先验框与真实框的并集）}}$ 。**IOU越大，越相关，则距离越小，反之亦然。** 备注：数据均已采用批标准化处理。

**K-means聚类中的距离：**  $d(box, centroids) = 1 - IOU(box, centroids)$



CSDN @胖墩会武术

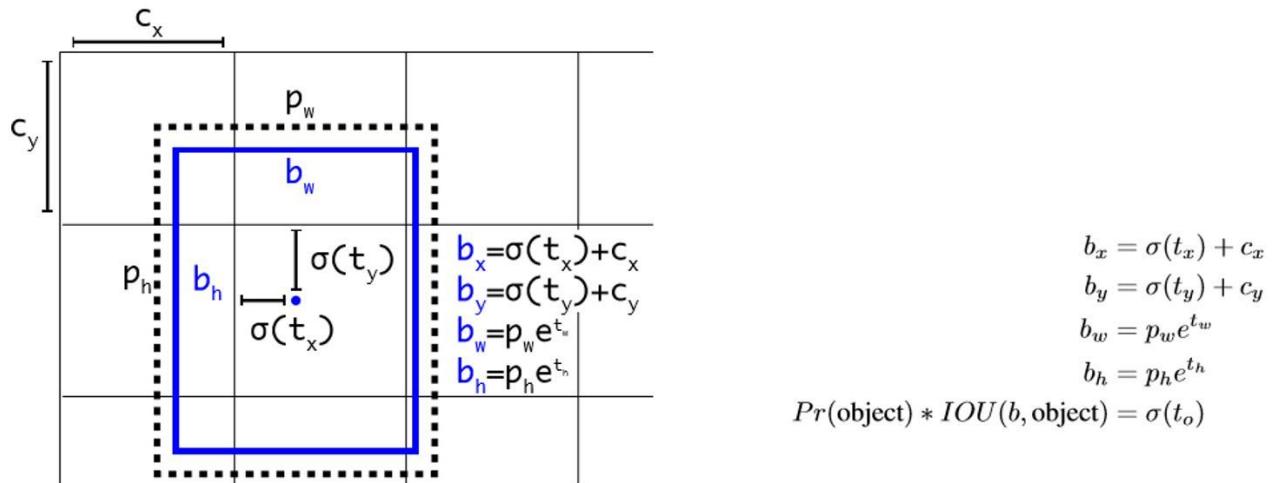
**左图**：x轴表示k的个数，y轴表示平均IOU值。紫色与黑色分别表示两个不同的数据集（形状相似）。综合考虑精确度和运算速度后，**yolov2最终取k=5个先验框**。**右图**：k=5个先验框的图形化显示。

## 5. Direct location prediction

约束预测边框的位置

用Anchor Box的方法，会让model变得不稳定，尤其是在最开始几次迭代的时候。大多数不稳定因素产生自预测Box的(x,y)位置的时候。按照之前YOLOv1的方法，网络不会预测偏移量，而是根据YOLOv1中的网格单元的位置来直接预测坐标，这就让Ground Truth的值介于0到1之间。而为了让网络的结果能落在这一范围内，网络使用一个 Logistic Activation来对于网络预测结果进行限制，让结果介于0到1之间。网络在每一个网格单元中预测出5个Bounding Boxes，每个Bounding Boxes有五个坐标值tx, ty, tw, th, to，他们的关系见下图。假设一个网格单元对于图片左上角的偏移量是cx, cy，

Bounding Boxes Prior的宽度和高度是pw, ph, 那么预测的结果见下图右面的公式:



**Figure 3: Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

<http://blog.csdn.net/hysteric314>

借鉴 Faster RCNN 的先验框的方法，在训练的早期阶段，其位置预测容易不稳定。位置预测公式通常为：

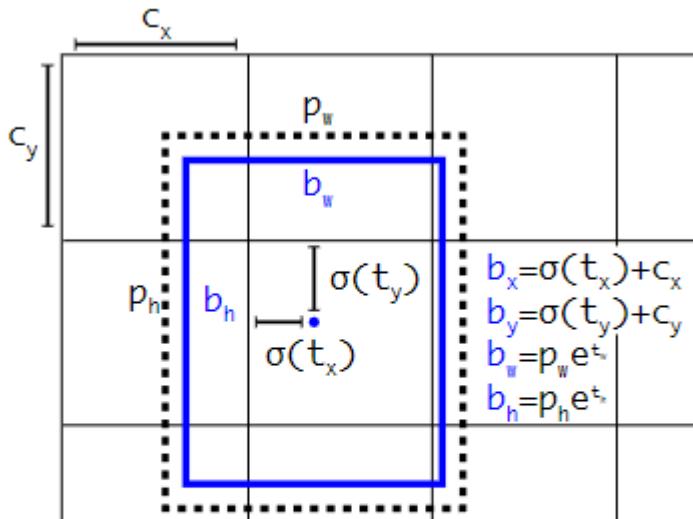
$$x = (t_x * w_a) + x_a$$

$$y = (t_y * h_a) + y_a$$

其中，

- x,y 预测边框的中心，
- x\_a, y\_a anchor的中心点坐标，
- w\_a, h\_a anchor的宽和高，
- t\_x, t\_y 学习的参数。

注意：YOLO中用的是 $x = (t_x * w_a) - x_a$ , Faster RCNN中用的是 $+$ 。由于 $t_x, t_y$ 的取值没有任何约束，因此预测边框的中心可能出现在任何位置，训练早期阶段不容易稳定。YOLOv2调整了预测公式，将预测边框的中心约束在特定grid网格内，如下图所示：



**Figure 3: Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

CSDN @L☆☆

其中,

- $b_x, b_y, b_w, b_h$  预测框的中心点和宽高,
- $c_x, c_y$  当前网格左上角到图像左上角的距离,
- $\sigma$  sigmoid函数,
- $t_x, t_y, p_w, p_h$  学习参数, 用于预测边框的中心和宽高。

因为使用了限制让数值变得参数化, 也让网络更容易学习、更稳定。

背景: 已知先验框的位置为  $(x, y, w, h)$ , 现在得到的预测边界框为  $(tx, ty, tw, th)$ , 即系统判定需要在先验框位置的基础上进行一定的偏移, 进而可以得到更真实的位置。故需要将预测的偏移量加到先验框中  $(x+tx, y+ty, w+tw, h+th)$ 。问题: 由于模型刚开始训练时, 网络参数都是随机初始化, 虽然进行了批标准化但是参数的基数比较大, 将导致预测的边界框加上偏移量之后到处乱飘。

yolov2的本质: 在当前网格中进行相对位置的微调。下图参数说明:

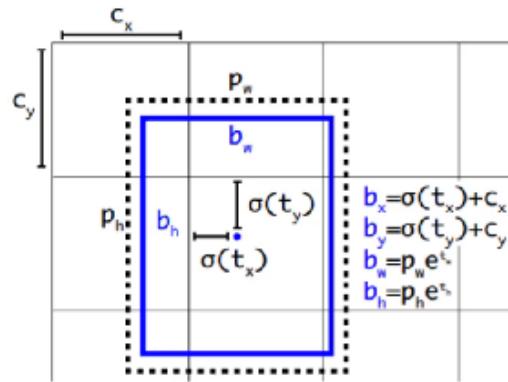
$(Cx, Cy)$  : 表示当前网格的左上角位置坐标。  $(tx, ty, tw, th)$  : 表示预测的结果在当前网格相对位置的偏移量。  $\sigma(tx)$ : 表示对漂移量  $tx$  取sigmoid函数, 得到  $(0 \sim 1)$  之间的值。即预测边框的蓝色中心点被约束在蓝色背景的网格内。约束边框位置使得模型更容易学习, 且预测更为稳定。  $e^{tw}$ : 是由于预测时取的log()对数值, 故计算位置时进行还原。  $(bx, by, bw, bh)$  : 表示当前预测结果在特征图位置 (即预处理

后得到的 $13 \times 13$ 网格)。

## ✓ Relative position prediction

计算公式为:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned}$$



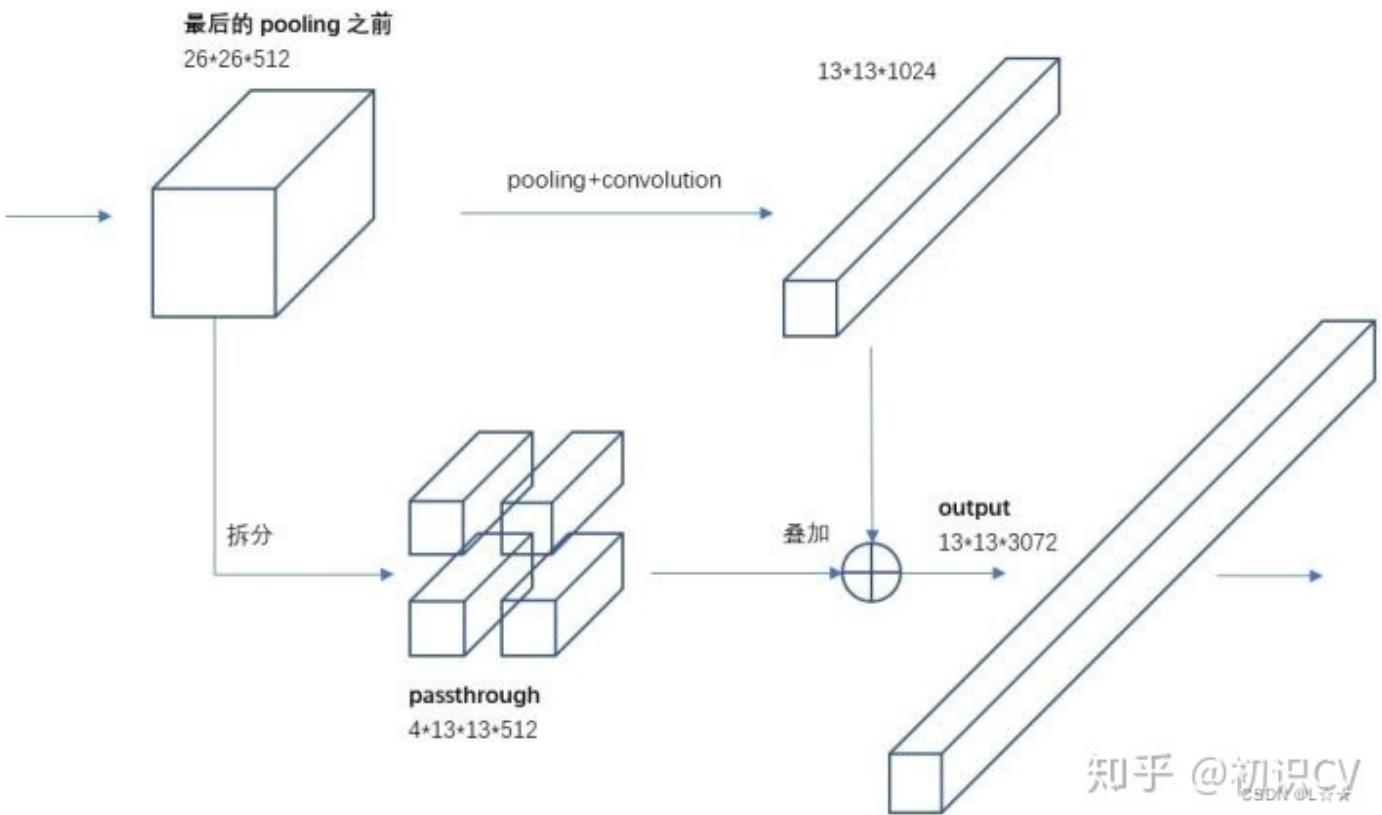
例如: 已知预测值( $\sigma t_x, \sigma t_y, t_w, t_h$ )=(0.2,0.1,0.2,0.32), 先验框pw=3.19275, ph=4.00944。

在特征图中位置 ( $13 \times 13$ 网格)	$b_x = 0.2 + 1 = 1.2$ $b_y = 0.1 + 1 = 1.1$ $b_w = 3.19275 * e^{0.2} = 3.89963$ $b_h = 4.00944 * e^{0.32} = 5.52151$	在输入图像中位置 ( $32=2^5$ )	$b_x = 1.2 * 32 = 38.4$ $b_y = 1.1 * 32 = 35.2$ $b_w = 3.89963 * 32 = 124.78$ $b_h = 5.52151 * 32 = 176.68$
---------------------------------	---	--------------------------	--

## 6. Fine-Grained Features

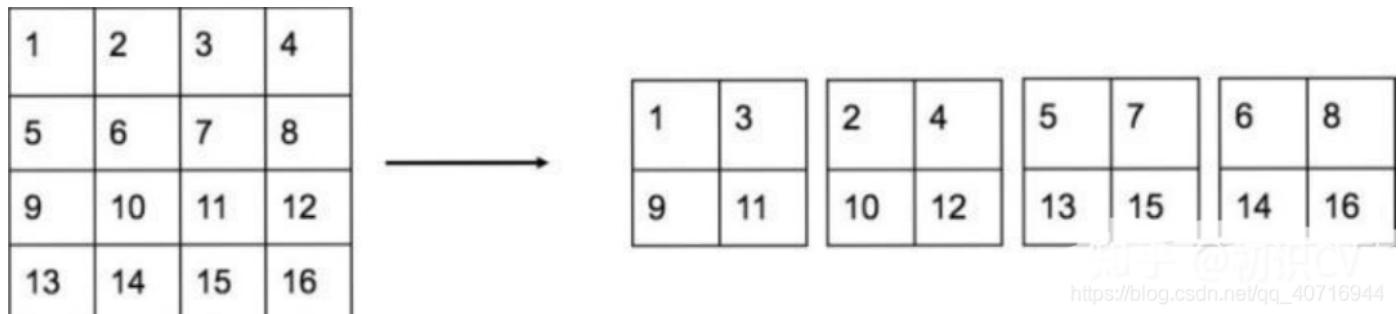
passthrough层检测细粒度特征, mAP提升1%。

对象检测面临的一个问题是图像中对象会有大有小, 输入图像经过多层网络提取特征, 最后输出的特征图中(比如YOLO v2中输入 $416 \times 416$ 经过卷积网络下采样最后输出是 $13 \times 13$ ), 较小的对象可能特征已经不明显甚至被忽略掉了。为了更好的检测出一些比较小的对象, 最后输出的特征图需要保留一些更细节的信息。YOLO v2引入一种称为passthrough层的方法在特征图中保留一些细节信息。具体来说, 就是在最后一个pooling之前, 特征图的大小是 $26 \times 26 \times 512$ , 将其 $111$ 拆 $444$ , 直接传递(passthrough)到pooling后(并且又经过一组卷积)的特征图, 两者叠加到一起作为输出的特征图。



另外，根据YOLO v2的代码，特征图先用 $1 \times 1$ 卷积从 $26 \times 26 \times 512$ 降维到 $26 \times 26 \times 64$ ，再做1拆4并passthrough。

具体怎样将1个特征图拆成4个特征图，见下图，图中示例的是1个 $4 \times 4$ \*拆成4个 $2 \times 2$ ，因为深度不变，所以没有画出来。

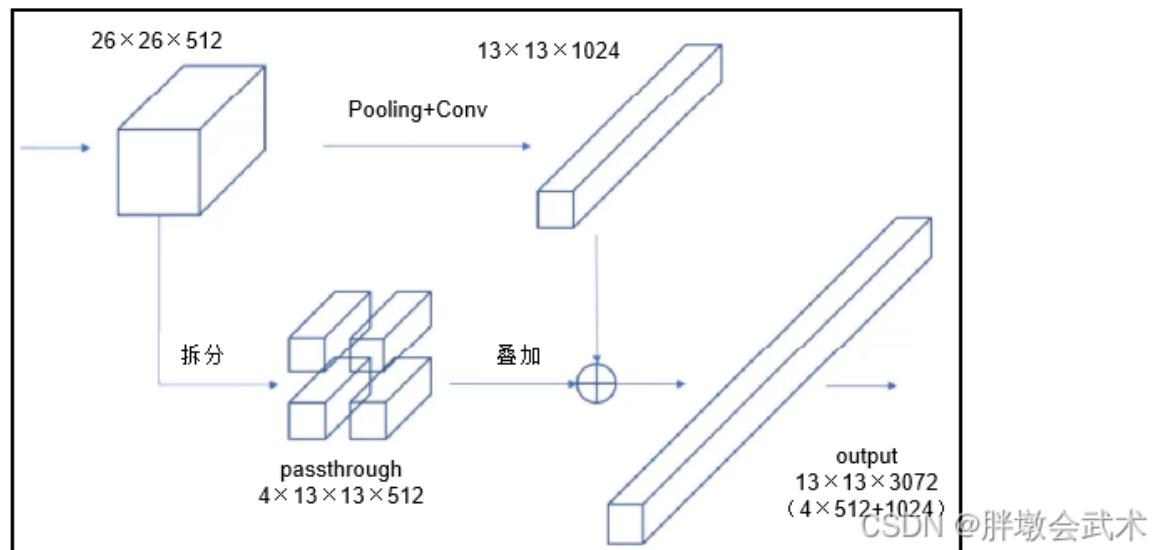


背景：

由于Faster R-CNN有大中小三种尺度scale的经验框，最终将对应得到小中大三种感受野。感受野越大，其在原图像中对应的尺度越大，导致其对尺度较小的目标不敏感，故无法兼顾考虑小尺度目标。备注：高分辨率（尺度大） - 感受野小；低分辨率（尺度小） - 感受野大。yolov2需要同时考虑三种不同的感受野，通过不同层的特征融合实现。具体操作：通过添加一个passthrough Layer，把高分辨率的浅层特征 $(26 \times 26 \times 512)$ 进行拆分，叠加到低分辨率的深层特征 $(13 \times 13 \times 1024)$ 中，然后进行特征融合 $(13 \times 13 \times 3072)$ ，最后再检测。（在yolov1中，FC起到全局特征融合的作用）

用）。目的：提高对小目标的检测能力

📌 最后一层时感受野太大了，小目标可能丢失了，需融合之前的特征



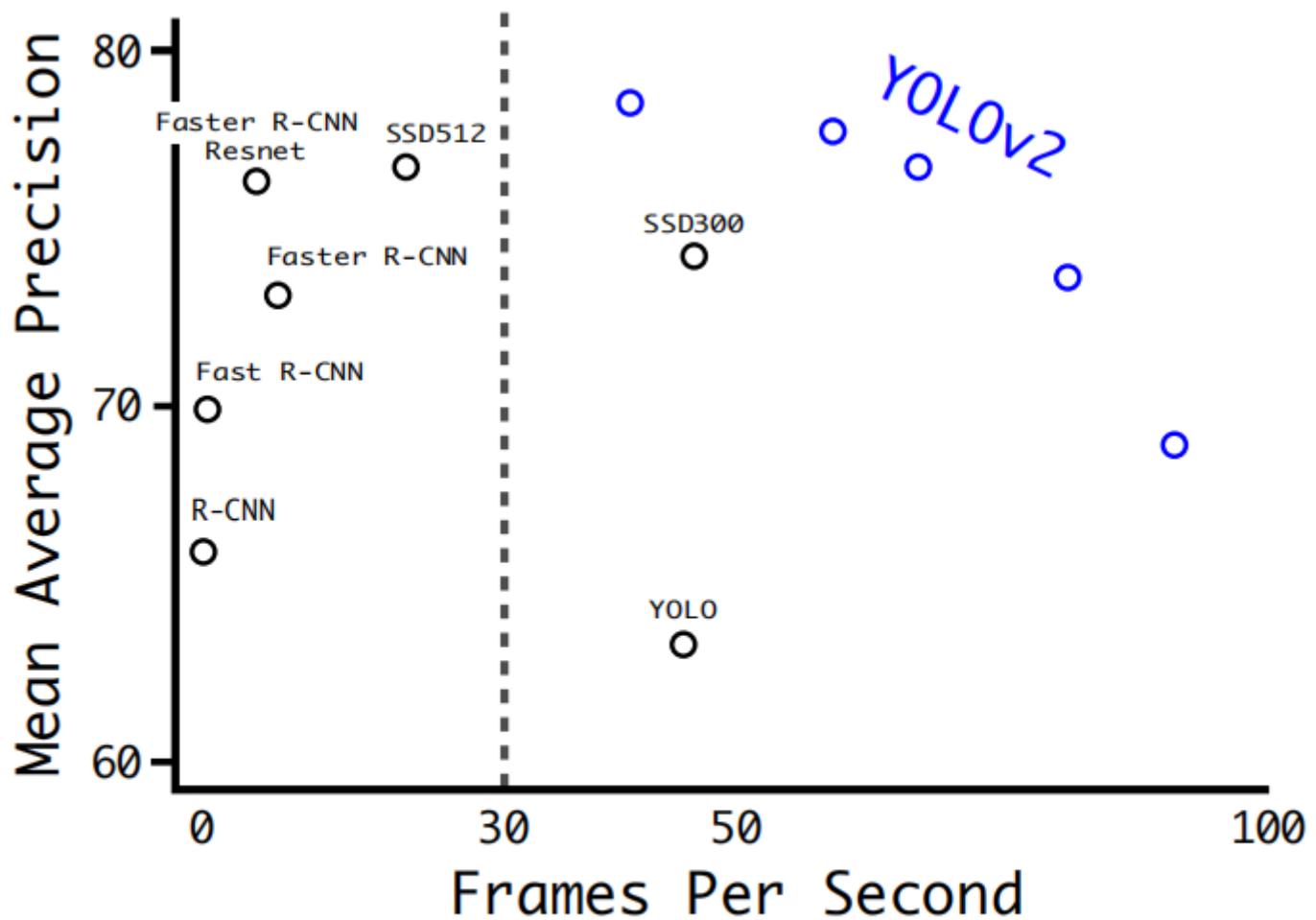
## 7. Multi-ScaleTraining

多尺度图像训练

作者希望YOLOv2能健壮的运行于不同尺寸的图片之上，所以把这一想法用于训练model中。

区别于之前的补全图片的尺寸的方法，YOLOv2每迭代几次都会改变网络参数。每10个Batches，网络会随机地选择一个新的图片尺寸，由于使用了下采样参数是32，所以不同的尺寸大小也选择为32的倍数320，352……608 最小 $320 \times 320$ ，最大 $608 \times 608$ ，网络会自动改变尺寸，并继续训练的过程。

这一政策让网络在不同的输入尺寸上都能达到一个很好的预测效果，同一网络能在不同分辨率上进行检测。当输入图片尺寸比较小的时候跑的比较快，输入图片尺寸比较大的时候精度高，所以你可以在YOLOv2的速度和精度上进行权衡。



**Figure 4: Accuracy and speed on VOC 2007.**

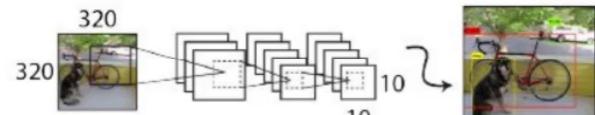
[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	<b>78.6</b>	40

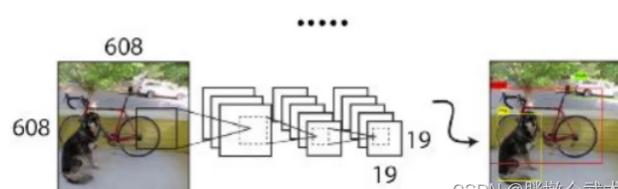
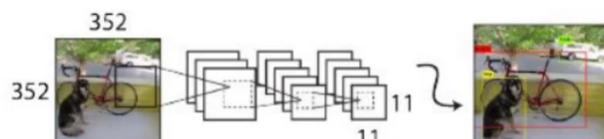
[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

背景：由于实际检测数据的输入图像大小不一，若都裁剪为相同大小，最后检测结果将有所下降。限制：由于yolov2只有卷积层，故对输入图像大小没有限制；而yolov1由于有全连接层，故输入图像大小固定。具体操作：训练模型每经过一定迭代之后，可以进行输入图像尺度变换。如：每迭代100次，输入图像尺寸大小增加10%。（备注：输入图像大小必须可以被32整除）

**最小的图像尺寸为320 x 320**



**最大的图像尺寸为608 x 608**



CSDN @机器学习技术

## 8. Darknet-19

backbone网络YOLOv1的backbone使用的是GoogleLeNet，速度比VGG-16快，YOLOv1完成一次前向过程只用8.52 billion 运算，而VGG-16要30.69billion，但是YOLOv1精度稍低于VGG-16。

YOLOv2基于一个新的分类model，有点类似与VGG。YOLOv2使用 $3 \times 3$ filter，每次Pooling之后都增加一倍Channels的数量。YOLOv2使用Global Average Pooling，使用Batch Normalization来让训练更稳定，加速收敛，使model规范化。最终的model—Darknet19，有19个卷积层和5个maxpooling层，处理一张图片只需要5.58 billion次运算，在ImageNet

上达到72.9%top-1精确度，91.2%top-5精确度。

Type	Filters	Size/Stride	Output
Convolutional	32	$3 \times 3$	$224 \times 224$
Maxpool		$2 \times 2/2$	$112 \times 112$
Convolutional	64	$3 \times 3$	$112 \times 112$
Maxpool		$2 \times 2/2$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Convolutional	64	$1 \times 1$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Maxpool		$2 \times 2/2$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Convolutional	128	$1 \times 1$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Maxpool		$2 \times 2/2$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Maxpool		$2 \times 2/2$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	1000	$1 \times 1$	$7 \times 7$
Avgpool		Global	1000
Softmax			<a href="https://blog.csdn.net/qq_40716944">https://blog.csdn.net/qq_40716944</a>

Darknet-19有19个卷积层和5个maxpooling层，采用全局平均池化的方法进行预测，并采用 $1 \times 1$ 卷积来压缩 $3 \times 3$ 卷积之间的特征表示。使用批处理归一化来稳定训练，加快收敛速度，并对模型进行规范化。

## Training for classification

网络训练在ImageNet 1000类分类数据集上训练了160epochs，使用随机梯度下降，初始学习率为0.1，polynomial rate decay with a power of 4, weight decay of 0.0005 and momentum of 0.9。训练期间使用标准的数据扩大方法：随机裁剪、旋转、变换颜色(hue)、变换饱和度(saturation)，变换曝光度(exposure shifts)。在训练时，把整个网络在更大的448\*448分辨率上Fine Turnning 10个epochs，初始学习率设置为0.001，这种网络达到76.5%top-1精确度，93.3%top-5精确度。

## Training for detection

网络去掉了最后一个卷积层，而加上了三个33卷积层，每个卷积层有1024个Filters，每个卷积层紧接着一个11卷积层。对于VOC数据，网络预测出每个网格单元预测五个Bounding Boxes，每个Bounding Boxes预测5个坐标和20类，所以一共125个Filters，增加了Passthrough层来获取前面层的细粒度信息，网络训练了160epoches，初始学习率0.001，数据扩大方法相同，对COCO与VOC数据集的训练对策相同。

## 9. Hierarchical classification

### 分层分类

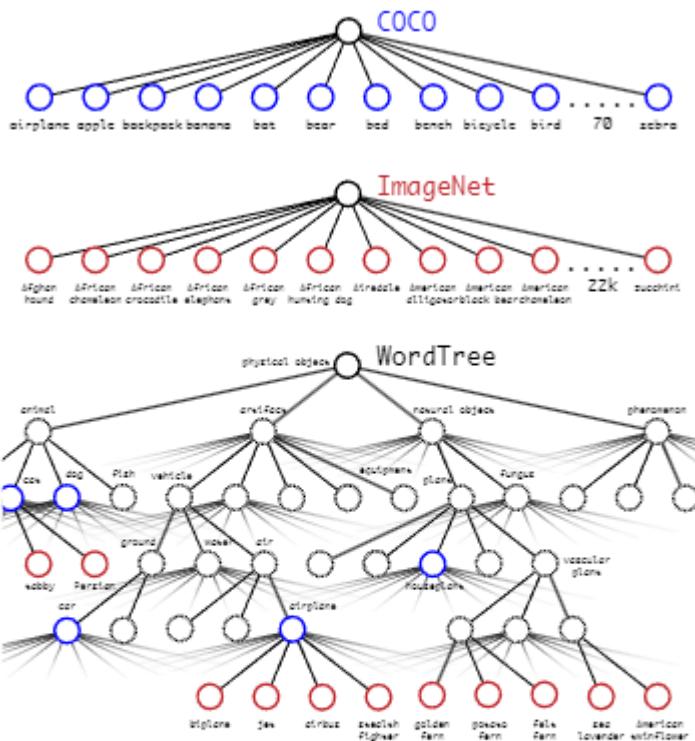
作者提出了一种在分类数据集和检测数据集上联合训练的机制。使用检测数据集的图片去学习检测相关的信息，例如bounding boxes 坐标预测，是否包含物体以及属于各个物体的概率。使用仅有类别标签的分类数据集图片去扩展可以检测的种类。

作者通过ImageNet训练分类、COCO和VOC数据集来训练检测，这是一个很有价值的思路，可以让我们达到比较优的效果。通过将两个数据集混合训练，如果遇到来自分类集的图片则只计算分类的Loss，遇到来自检测集的图片则计算完整的Loss。

但是ImageNet对应分类有9000种，而COCO则只提供80 8080种目标检测，作者使用multi-label模型，即假定一张图片可以有多个label，并且不要求label间独立。

通过作者Paper里的图来说明，由于ImageNet的类别是从WordNet选取的，作者采用以下策略重建了一个树形结构（称为分层树）

- 遍历Imagenet的label，然后在WordNet中寻找该label到根节点(指向一个物理对象)的路径
- 如果路径直有一条，那么就将该路径直接加入到分层树结构中
- 否则，从剩余的路径中选择一条最短路径，加入到分层树



**Figure 6: Combining datasets using WordTree hierarchy.** Using the WordNet concept graph we build a hierarchical tree of visual concepts. Then we can merge datasets together by mapping the classes in the dataset to synsets in the tree. This is a simplified view of WordTree for illustration purposes.

CSDN @L☆☆

这个分层树我们称之为 WordTree，作用就在于 **将两种数据集按照层级进行结合**。

## YOLOv3: An Incremental Improvement

论文下载：<https://arxiv.org/abs/1804.02767>

论文地址：<https://pjreddie.com/media/files/papers/YOLOv3.pdf>

代码下载：[https://github.com/yjh0410/yolov2-yolov3\\_PyTorch](https://github.com/yjh0410/yolov2-yolov3_PyTorch)

### 核心思想

YOLOv3的核心思想在于多尺度的预测，也就是使用三种不同的网格来划分原始图像。其中 $13 \times 13$ 的网格划分的每一块最大，用于预测大目标。 $26 \times 26$ 的网格划分的每一块中等大小，用于预测中等目标。 $52 \times 52$ 的网格划分的每一块最小，用于预测小目标。

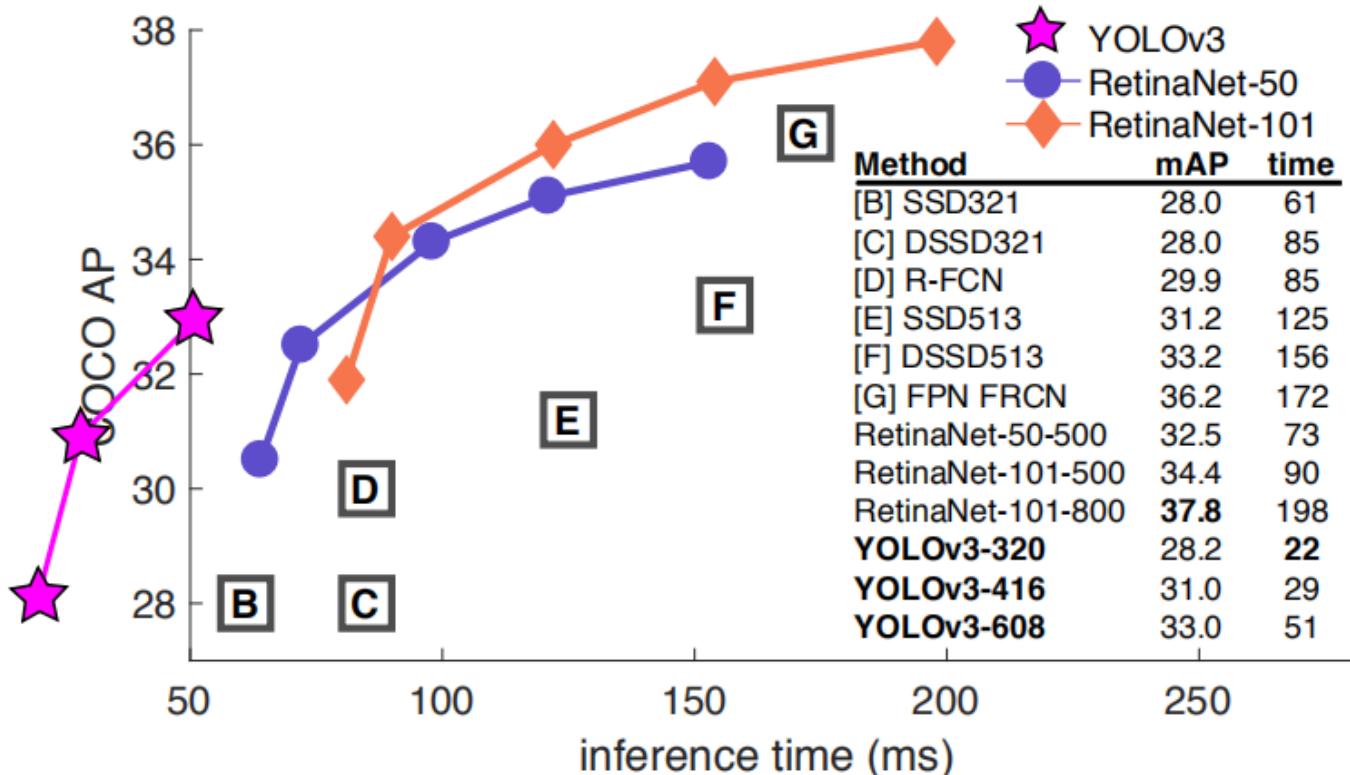
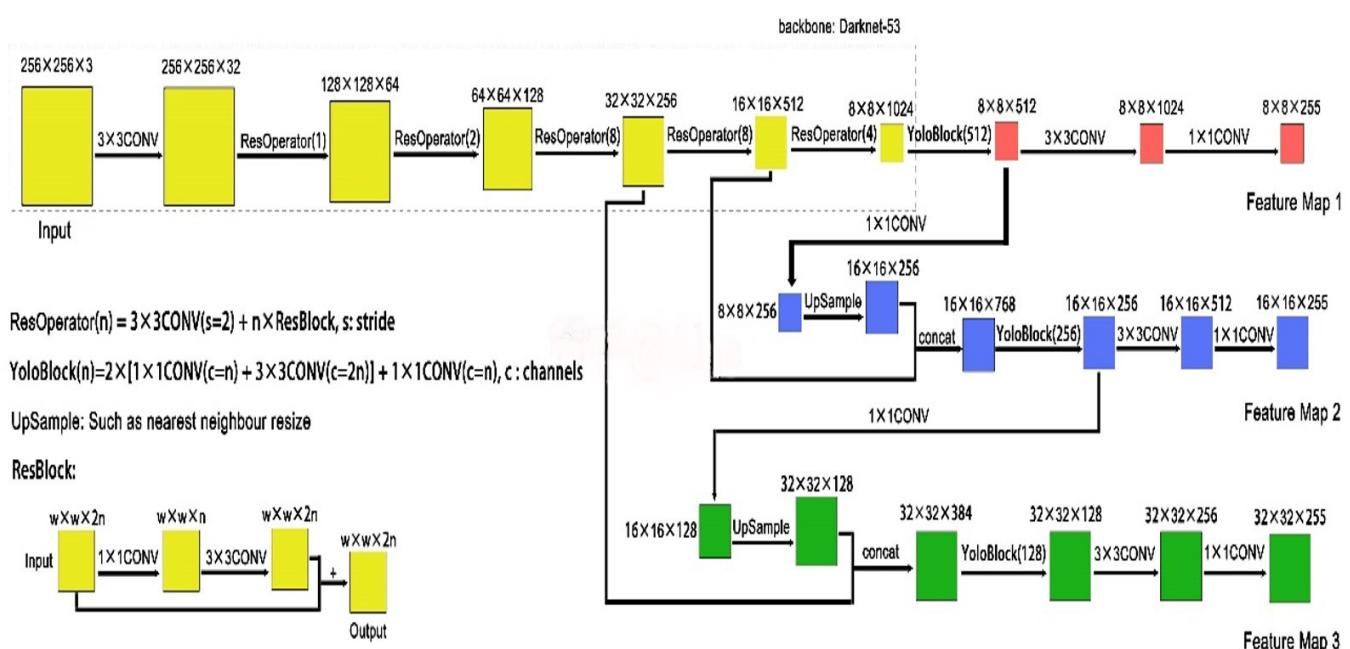


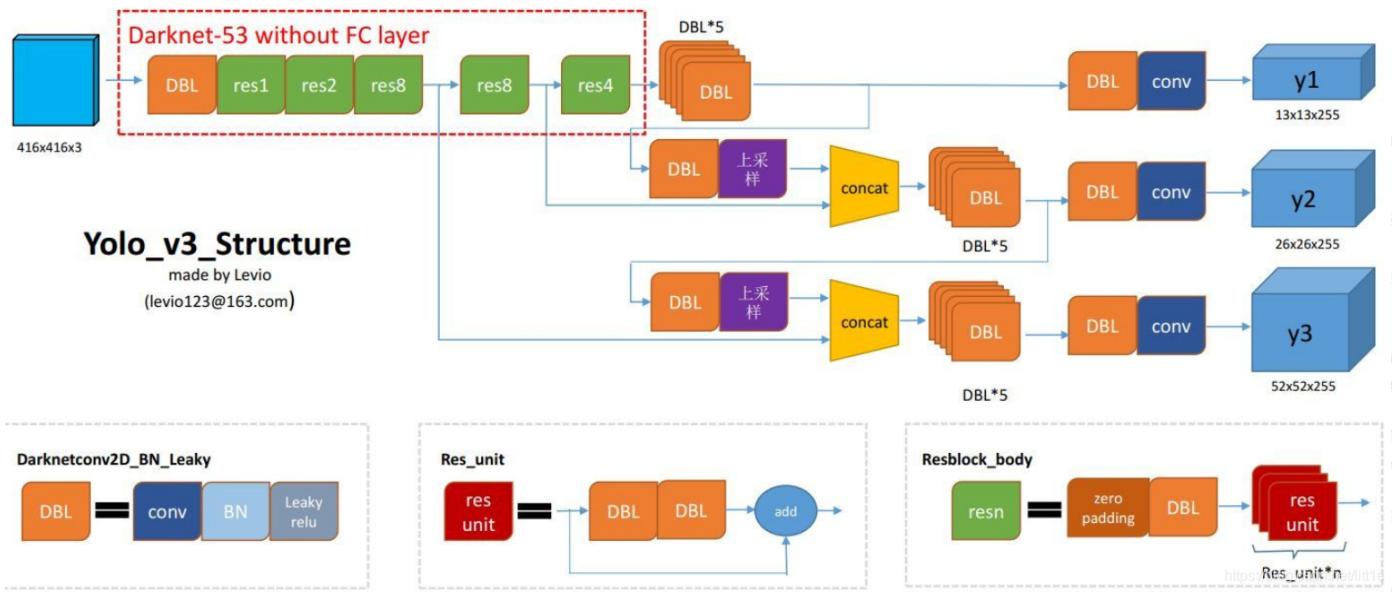
Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

## 网络模型



从[这里](#)盗了张图，这张图很好的总结了YOLOv3的结构，让我们对YOLO有更加直观的理解。



DBL: 代码中的Darknetconv2d\_BN\_Leaky，是YOLOv3的基本组件，就是卷积+BN+Leaky relu。resn: n代表数字，有res1, res2, ..., res8等等，表示这个res\_block里含有多少个res\_unit。不懂resnet请戳这儿 concat: 张量拼接；将darknet中间层和后面的某一层的上采样进行拼接。拼接的操作和残差层add的操作是不一样的，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。

## 网络改进

### 简述

- 多尺度预测（引入FPN）。
- 更好的基础分类网络（darknet-53, 类似于ResNet引入残差结构）。
- 分类器不在使用Softmax, 分类损失采用binary cross-entropy loss（二分类交叉损失熵）

### 1. Darknet-53

下图是Darknet-53的网络结构图

- 此结构主要由53个卷积层构成，卷积层对于分析物体特征最为有效。由于没有使用全连接层，该网络可以对应任意大小的输入图像。
- 此外，池化层也没有出现在其中，取而代之的是将卷积层设置stride=2来达到下采样的效果，同时将尺度不变特征传送到下一层。
- 除此之外，Darknet-53中还使用了类似ResNet的结构

Type	Filters	Size	Output
Convolutional	32	$3 \times 3$	$256 \times 256$
Convolutional	64	$3 \times 3 / 2$	$128 \times 128$
1x	32	$1 \times 1$	
Convolutional	64	$3 \times 3$	
			$128 \times 128$
Convolutional	128	$3 \times 3 / 2$	$64 \times 64$
2x	64	$1 \times 1$	
Convolutional	128	$3 \times 3$	
			$64 \times 64$
Convolutional	256	$3 \times 3 / 2$	$32 \times 32$
8x	128	$1 \times 1$	
Convolutional	256	$3 \times 3$	
			$32 \times 32$
Convolutional	512	$3 \times 3 / 2$	$16 \times 16$
Convolutional	256	$1 \times 1$	
8x	512	$3 \times 3$	
			$16 \times 16$
Convolutional	1024	$3 \times 3 / 2$	$8 \times 8$
4x	512	$1 \times 1$	
Convolutional	1024	$3 \times 3$	
			$8 \times 8$
Avgpool		Global	
Connected		1000	
Softmax			

Table 1. Darknet-53.

CSDN @L☆★

为了达到更好的分类效果，作者自己设计训练了darknet-53，在ImageNet数据集上实验发现这个darknet-53，的确很强，相对于ResNet-152和ResNet-101，darknet-53不仅在分类精度上差不多，计算速度还比ResNet-152和ResNet-101强多了，网络层数也比他们少，测试结果如图所示。

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	<b>171</b>
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	<b>77.6</b>	<b>93.8</b>	29.4	1090	37
Darknet-53	77.2	<b>93.8</b>	18.7	<b>1457</b>	78

darknet-53的网络结构如下图所示。YOLOv3使用了darknet-53的前面的52层（没有全连接层），YOLOv3这个网络是一个全卷积网络，大量使用残差的跳层连接，并且为了降低池化带来的梯度负面影响，作者直接摒弃了POOLing，用conv的stride来实现降采样。在这个网络结构中，使用的是步长为2的卷积来进行降采样。

为了加强算法对小目标检测的精确度，YOLOv3中采用类似FPN的upsample和融合做法（最后融合了3个scale，其他两个scale的大小分别是 $26 \times 26$ 和 $52 \times 52$ ），在多个scale的feature map上做检测。

作者在3条预测支路采用的也是全卷积的结构，其中最后一个卷积层的卷积核个数是255，是针对COCO数据集的80类： $3 \times (80+4+1) = 255$ ，3表示一个grid cell包含3个bounding box，4表示框的4个坐标信息，1表示objectness score。

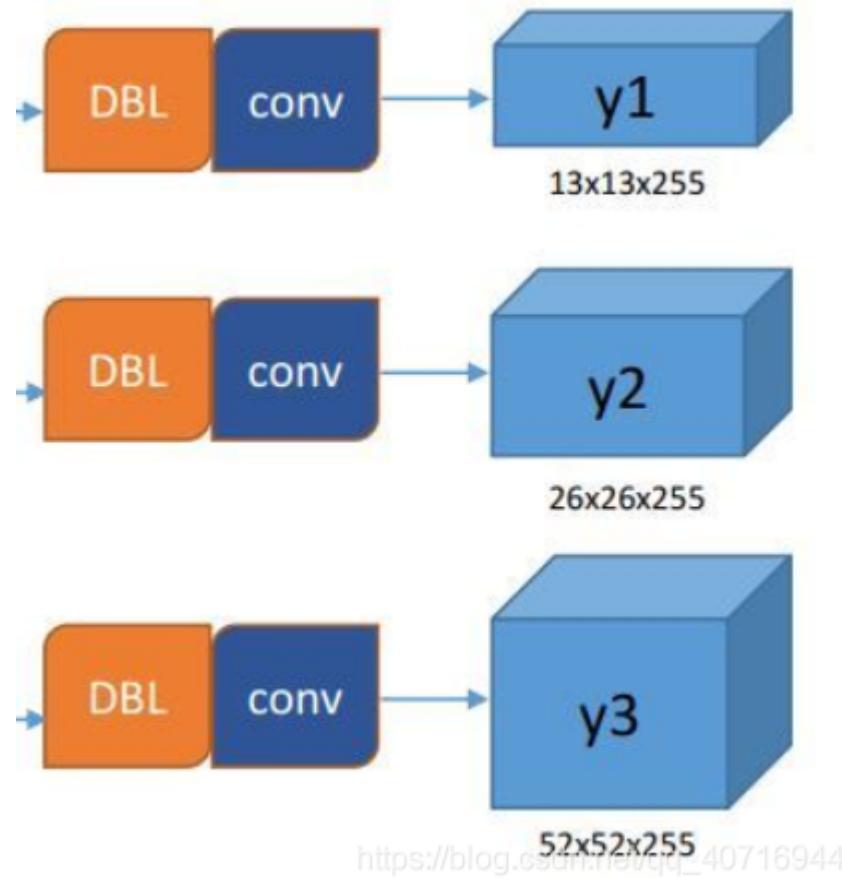
Type	Filters	Size	Output
Convolutional	32	$3 \times 3$	$256 \times 256$
Convolutional	64	$3 \times 3 / 2$	$128 \times 128$
Convolutional	32	$1 \times 1$	
Convolutional	64	$3 \times 3$	
Residual			$128 \times 128$
Convolutional	128	$3 \times 3 / 2$	$64 \times 64$
Convolutional	64	$1 \times 1$	
Convolutional	128	$3 \times 3$	
Residual			$64 \times 64$
Convolutional	256	$3 \times 3 / 2$	$32 \times 32$
Convolutional	128	$1 \times 1$	
Convolutional	256	$3 \times 3$	
Residual			$32 \times 32$
Convolutional	512	$3 \times 3 / 2$	$16 \times 16$
Convolutional	256	$1 \times 1$	
Convolutional	512	$3 \times 3$	
Residual			$16 \times 16$
Convolutional	1024	$3 \times 3 / 2$	$8 \times 8$
Convolutional	512	$1 \times 1$	
Convolutional	1024	$3 \times 3$	
Residual			$8 \times 8$
Avgpool		Global	
Connected		1000	
Softmax			

全连接层是为了分类

[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

所谓的多尺度就是来自这3条预测之路，y1,y2和y3的深度都是255，边长的规律是13:26:52。YOLOv3设定的是每个网格单元预测3个box，所以每个box需要有(x, y, w, h,

confidence)五个基本参数，然后还要有80个类别的概率。所以 $3 \times (5 + 80) = 255$ ，这个



255就是这么来的。

下面我们具体看看 $y1, y2, y3$ 是如何而来的。网络中作者进行了三次检测，分别是在32倍降采样，16倍降采样，8倍降采样时进行检测，这样在多尺度的feature map上检测跟SSD有点像。在网络中使用up-sample（上采样）的原因：网络越深的特征表达效果越好，比如在进行16倍降采样检测，如果直接使用第四次下采样的特征来检测，这样就使用了浅层特征，这样效果一般并不好。如果想使用32倍降采样后的特征，但深层特征的大小太小，因此YOLOv3使用了步长为2的up-sample（上采样），把32倍降采样得到的feature map的大小提升一倍，也就成了16倍降采样后的维度。同理8倍采样也是对16倍降采样的特征进行步长为2的上采样，这样就可以使用深层特征进行detection。

作者通过上采样将深层特征提取，其维度是与将要融合的特征层维度相同的（channel不同）。如下图所示，85层将 $13 \times 13 \times 256$ 的特征上采样得到 $26 \times 26 \times 256$ ，再将其与61层的特征拼接起来得到 $26 \times 26 \times 768$ 。为了得到channel255，还需要进行一系列的 $3 \times 3$ ,  $1 \times 1$ 卷积操作，这样既可以提高非线性程度增加泛化性能提高网络精度，又能减少参数提高实时性。 $52 \times 52 \times 255$ 的特征也是类似的过程。

layer	filters	size	input		output	
0 conv	32	3 x 3 / 1	416 x 416 x 3	->	416 x 416 x 32	一次采样 1.595 BFLOPs
1 conv	64	3 x 3 / 2	416 x 416 x 32	->	208 x 208 x 64	1.595 BFLOPs
2 conv	32	1 x 1 / 1	208 x 208 x 64	->	208 x 208 x 32	0.177 BFLOPs
3 conv	64	3 x 3 / 1	208 x 208 x 32	->	208 x 208 x 64	1.595 BFLOPs
4 res	1		208 x 208 x 64	->	208 x 208 x 64	1.595 BFLOPs
5 conv	128	3 x 3 / 2	208 x 208 x 64	->	104 x 104 x 128	1.595 BFLOPs
6 conv	64	1 x 1 / 1	104 x 104 x 128	->	104 x 104 x 64	0.177 BFLOPs
7 conv	128	3 x 3 / 1	104 x 104 x 64	->	104 x 104 x 128	1.595 BFLOPs
8 res	5		104 x 104 x 128	->	104 x 104 x 128	1.595 BFLOPs
9 conv	64	1 x 1 / 1	104 x 104 x 128	->	104 x 104 x 64	0.177 BFLOPs
10 conv	128	3 x 3 / 1	104 x 104 x 64	->	104 x 104 x 128	1.595 BFLOPs
11 res	8		104 x 104 x 128	->	104 x 104 x 128	1.595 BFLOPs
12 conv	256	3 x 3 / 2	104 x 104 x 128	->	52 x 52 x 256	二次下采样 1.595 BFLOPs
13 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128	0.177 BFLOPs
14 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256	1.595 BFLOPs
15 res	12		52 x 52 x 256	->	52 x 52 x 256	1.595 BFLOPs
16 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128	0.177 BFLOPs
17 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256	1.595 BFLOPs
18 res	15		52 x 52 x 256	->	52 x 52 x 256	1.595 BFLOPs
19 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128	0.177 BFLOPs
20 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256	1.595 BFLOPs
21 res	18		52 x 52 x 256	->	52 x 52 x 256	1.595 BFLOPs
22 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128	0.177 BFLOPs
23 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256	1.595 BFLOPs
24 res	21		52 x 52 x 256	->	52 x 52 x 256	1.595 BFLOPs
25 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128	0.177 BFLOPs
26 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256	1.595 BFLOPs
27 res	24		52 x 52 x 256	->	52 x 52 x 256	1.595 BFLOPs
28 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128	0.177 BFLOPs
29 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256	1.595 BFLOPs
30 res	27		52 x 52 x 256	->	52 x 52 x 256	1.595 BFLOPs
31 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128	0.177 BFLOPs
32 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256	1.595 BFLOPs
33 res	30		52 x 52 x 256	->	52 x 52 x 256	1.595 BFLOPs
34 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128	0.177 BFLOPs
35 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256	1.595 BFLOPs
36 res	33		52 x 52 x 256	->	52 x 52 x 256	1.595 BFLOPs
37 conv	512	3 x 3 / 2	52 x 52 x 256	->	26 x 26 x 512	四次下采样 1.595 BFLOPs
38 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256	0.177 BFLOPs

http://blog.csyun.net/https://blog.csyun.net/

39 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512	1.595 BFLOPs
40 res	37		26 x 26 x 512	->	26 x 26 x 512	
41 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256	0.177 BFLOPs
42 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512	1.595 BFLOPs
43 res	40		26 x 26 x 512	->	26 x 26 x 512	
44 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256	0.177 BFLOPs
45 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512	1.595 BFLOPs
46 res	43		26 x 26 x 512	->	26 x 26 x 512	
47 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256	0.177 BFLOPs
48 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512	1.595 BFLOPs
49 res	46		26 x 26 x 512	->	26 x 26 x 512	
50 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256	0.177 BFLOPs
51 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512	1.595 BFLOPs
52 res	49		26 x 26 x 512	->	26 x 26 x 512	
53 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256	0.177 BFLOPs
54 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512	1.595 BFLOPs
55 res	52		26 x 26 x 512	->	26 x 26 x 512	
56 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256	0.177 BFLOPs
57 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512	1.595 BFLOPs
58 res	55		26 x 26 x 512	->	26 x 26 x 512	
59 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256	0.177 BFLOPs
60 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512	1.595 BFLOPs
61 res	58		26 x 26 x 512	->	26 x 26 x 512	五次下采样
62 conv	1024	3 x 3 / 2	26 x 26 x 512	->	13 x 13 x 1024	1.595 BFLOPs
63 conv	512	1 x 1 / 1	13 x 13 x 1024	->	13 x 13 x 512	0.177 BFLOPs
64 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x 1024	1.595 BFLOPs
65 res	62		13 x 13 x 1024	->	13 x 13 x 1024	
66 conv	512	1 x 1 / 1	13 x 13 x 1024	->	13 x 13 x 512	0.177 BFLOPs
67 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x 1024	1.595 BFLOPs
68 res	65		13 x 13 x 1024	->	13 x 13 x 1024	
69 conv	512	1 x 1 / 1	13 x 13 x 1024	->	13 x 13 x 512	0.177 BFLOPs
70 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x 1024	1.595 BFLOPs
71 res	68		13 x 13 x 1024	->	13 x 13 x 1024	
72 conv	512	1 x 1 / 1	13 x 13 x 1024	->	13 x 13 x 512	0.177 BFLOPs
73 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x 1024	1.595 BFLOPs
74 res	71		13 x 13 x 1024	->	13 x 13 x 1024	
75 conv	512	1 x 1 / 1	13 x 13 x 1024	->	13 x 13 x 512	0.177 BFLOPs
76 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x 1024	1.595 BFLOPs
77 conv	512	1 x 1 / 1	13 x 13 x 1024	->	13 x 13 x 512	0.177 BFLOPs
78 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x 1024	1.595 BFLOPs
79 conv	512	1 x 1 / 1	13 x 13 x 1024	->	13 x 13 x 512	0.177 BFLOPs
80 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x 1024	1.595 BFLOPs
81 conv	255	1 x 1 / 1	13 x 13 x 1024	->	13 x 13 x 255	0.088 BFLOPs
82 detection						32倍降采样 <a href="https://blog.csdn.net/little888">https://blog.csdn.net/little888</a>
83 route	79					此处route层直接把79处特征区过来

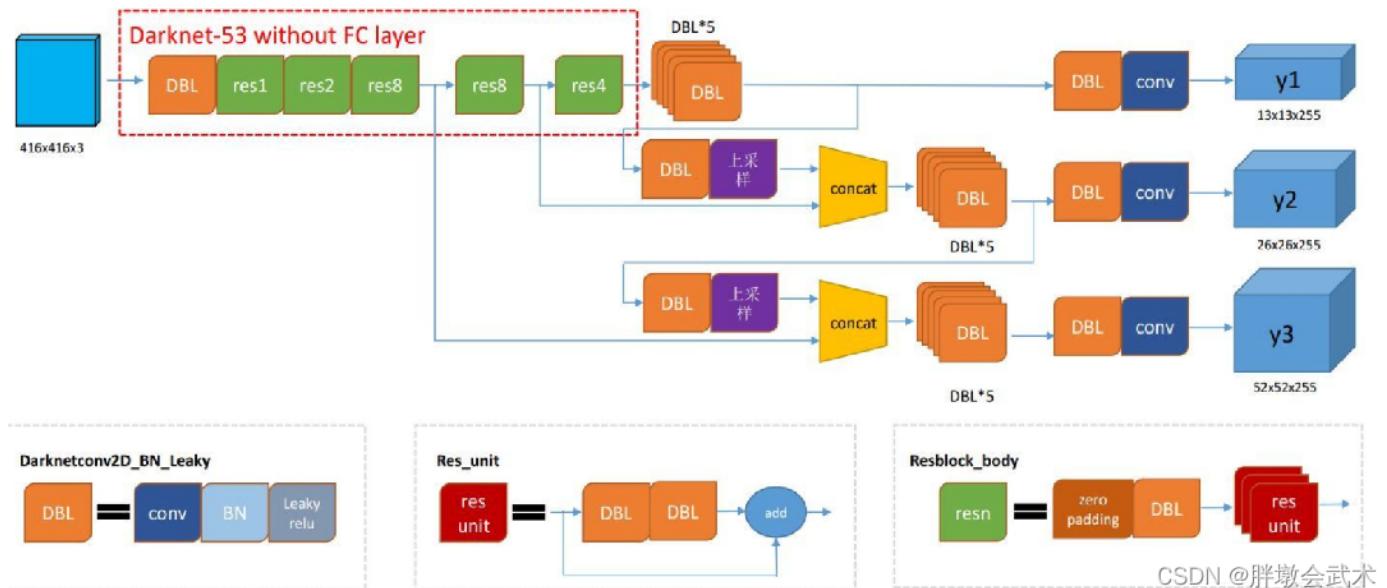
Darknet-53网络架构： (1) 由53个卷积层构成，包括1x1和3x3的卷积层，卷积省时省力速度快效果好，对于分析物体特征最为有效。每个卷积层之后包含一个批量归一化层和一个Leaky ReLU，加入这两个部分的目的是为了防止过拟合。 (2) 没有全连接层，可以对应任意大小的输入图像。 (3) 没有池化层，通过控制卷积层conv的步长stride达到下采样的效果，需要下采样时stride=2；否则stride=1； (4) 除此之外，Darknet-53中还使用了类似ResNet结构。

Type	Filters	Size	Output
Convolutional	32	$3 \times 3$	$256 \times 256$
Convolutional	64	$3 \times 3 / 2$	$128 \times 128$
Convolutional	32	$1 \times 1$	
1x Convolutional	64	$3 \times 3$	
Residual			$128 \times 128$
Convolutional	128	$3 \times 3 / 2$	$64 \times 64$
Convolutional	64	$1 \times 1$	
2x Convolutional	128	$3 \times 3$	
Residual			$64 \times 64$
Convolutional	256	$3 \times 3 / 2$	$32 \times 32$
Convolutional	128	$1 \times 1$	
8x Convolutional	256	$3 \times 3$	
Residual			$32 \times 32$
Convolutional	512	$3 \times 3 / 2$	$16 \times 16$
Convolutional	256	$1 \times 1$	
8x Convolutional	512	$3 \times 3$	
Residual			$16 \times 16$
Convolutional	1024	$3 \times 3 / 2$	$8 \times 8$
Convolutional	512	$1 \times 1$	
4x Convolutional	1024	$3 \times 3$	
Residual			$8 \times 8$
Avgpool		Global	
Connected	1000		
Softmax			

全连接层是为了分类

CSDN @胖墩会武术

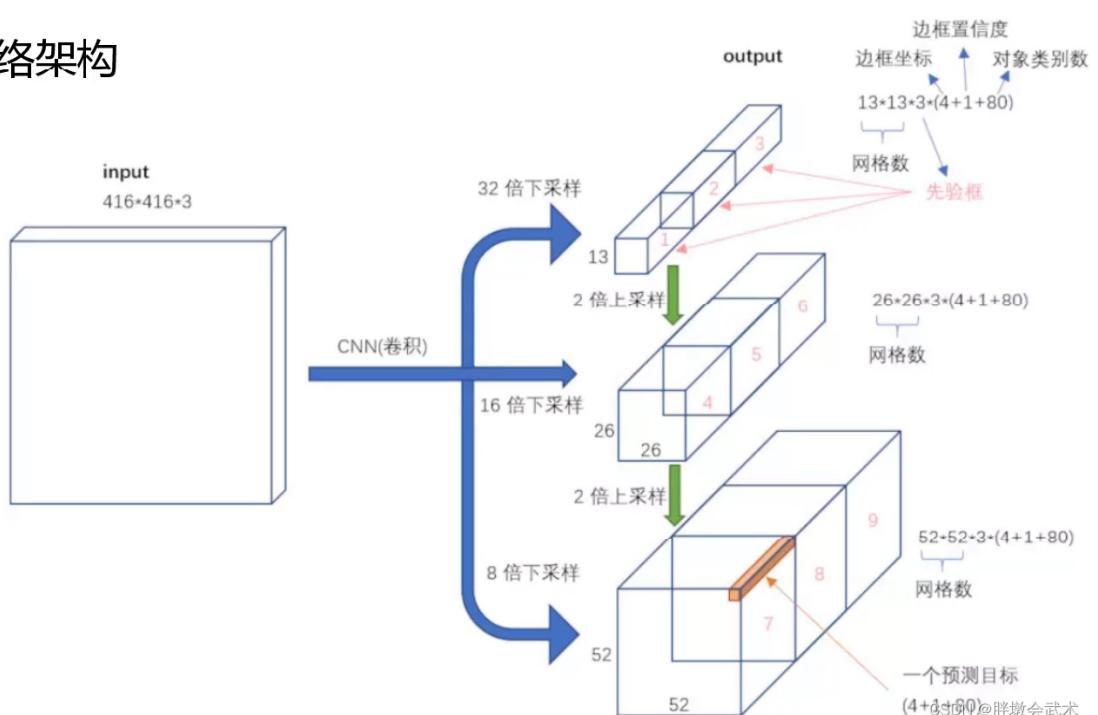
Darknet-53网络及在yolov3中的实际应用。可以看下面这张图：



- DBL：由一个卷积层、一个批量归一化层和一个Leaky ReLU组成的基本卷积单元。在Darknet-53中，共有53个这样的DBL，所以称其为Darknet-53。

- res unit：输入通过两个DBL后，再与原输入进行特征add，得到与原图像大小维度相同的图像；这是一种常规的残差单元。残差单元的目的是为了让网络可以提取到更深层的特征，同时避免出现梯度消失或爆炸。残差网络的特点：至少不比原来差。
- res(n)：表示n个res unit。  $\text{resn} = \text{Zero Padding} + \text{DBL} + n \times \text{res unit}$ 。
- y1、y2、y3：分别表示yolov3的三种不同尺度输出（分别对应：大中小感受野）。
- concat1：（大中小感受野）将大感受野的特征图像进行上采样，得到与中感受野的特征图像相同大小，然后进行维度拼接，达到多尺度特征融合的目的。为了加强算法对小目标检测的精确度
- concat2：（大中小感受野）将中感受野的特征图像进行上采样，得到与小感受野的特征图像相同大小，然后进行维度拼接，达到多尺度特征融合的目的。为了加强算法对小目标检测的精确度

## ✓ 核心网络架构



### • bounding box 与 anchor box的输出区别

- (1) Bounding box输出：框的位置（中心坐标与宽高），confidence以及N个类别。
- (2) anchor box输出：一个尺度即只有宽高。

## 2. 边框回归

一个回归框是由4 44个参数决定 $x, y, w, h$ 。YOLOv3是在训练的数据集上聚类产生prior boxes的一系列宽高(是在图像416x416的坐标系里)，默认9种。YOLO v3思想理论是将输入图像分成 $S \times S$ 个格子（有三处进行检测，分别是在 $52 \times 52, 26 \times 26, 13 \times 13$ 的feature map上）。

map上，即S SS会分别为52,26,13），若某个物体Ground truth的中心位置的坐标落入到某个格子，那么这个格子就负责检测中心落在该栅格中的物体。

三次检测，每次对应的感受野不同：

- 32倍降采样的感受野最大( $13 \times 13$ )，适合检测大的目标，每个cell的3个anchor boxes为 $(116,90), (156,198), (373,326)$ 。
- 16 倍 ( $26 \times 26$ )适合一般大小的物体，anchor boxes为 $(30, 61), (62, 45), (59, 119)$ 。
- 8倍的感受野最小( $52 \times 52$ )，适合检测小目标，因此anchor boxes为 $(10, 13), (16, 30), (33, 23)$ 。
- 所以当输入为 $416 \times 416$ 时，实际总共有 $(52 \times 52 + 26 \times 26 + 13 \times 13) \times 3 = 10647$ 个proposal boxes。

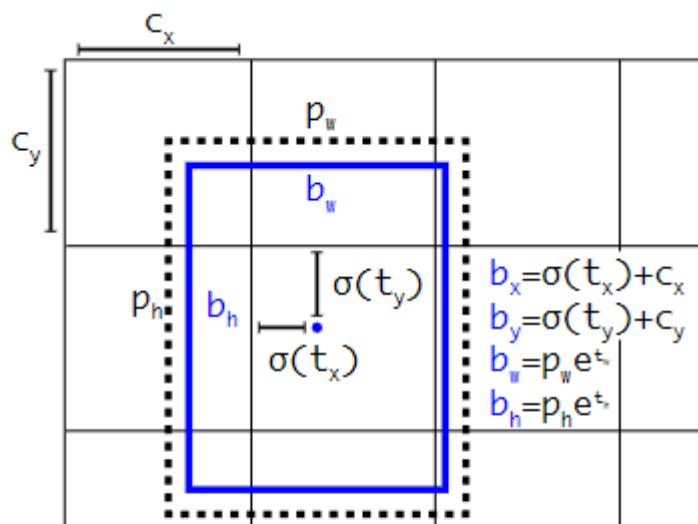


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

CSDN @L☆☆

YOLOv3对每个bounding box预测偏移量和尺度缩放四个值  $t_x, t_y, t_w, t_h$ （网络学习的参数），对于预测的cell（一幅图划分成 $S \times S$ 个网格cell）根据图像左上角的偏移  $(c_x, c_y)$ ，每个grid cell在feature map中的宽和高均为1，以及预先设的anchor box的宽和高  $p_w, p_h$ （预设聚类的宽高需要除以stride映射到feature map上）。最终得到的边框坐标值是  $b$ ，而网络学习目标是  $t^*$ ，用sigmoid函数、指数转换。可以对bounding box按如上图的方式进行预测。

## Bounding Box

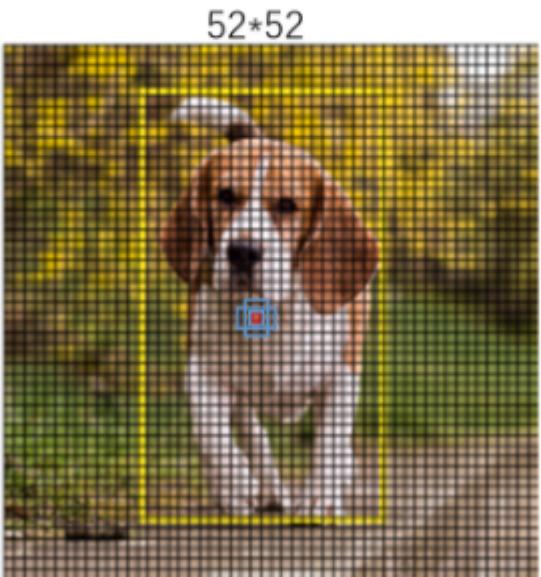
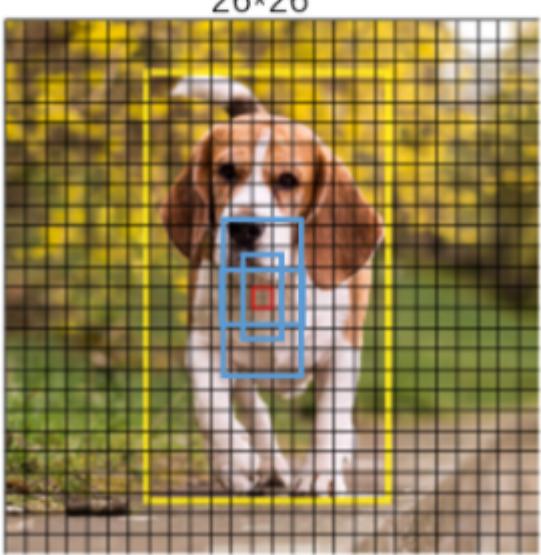
YOLOv3的Bounding Box由YOLOv2又做出了更好的改进。在YOLOv2和YOLOv3中，都采用了对图像中的object采用k-means聚类。feature map中的每一个cell都会预测3个边

界框 (bounding box) , 每个bounding box都会预测三个东西： (1) 每个框的位置 (4个值, 中心坐标tx和ty, 框的高度bh和宽度bw) , (2) 一个objectness prediction , (3) N个类别, coco数据集80类, voc20类。

三次检测, 每次对应的感受野不同, 32倍降采样的感受野最大, 适合检测大的目标, 所以在输入为416×416时, 每个cell的三个anchor box为(116 ,90); (156 ,198); (373 ,326)。16倍适合一般大小的物体, anchor box为(30,61); (62,45); (59,119)。8倍的感受野最小, 适合检测小目标, 因此anchor box为(10,13); (16,30); (33,23)。所以当输入为416×416时, 实际总共有  $(52 \times 52 + 26 \times 26 + 13 \times 13) \times 3 = 10647$  个proposal box。

特征图	13*13			26*26			52*52		
感受野	大			中			小		
先验框	(116x90)	(156x198)	(373x326)	(30x61)	(62x45)	(59x119)	(10x13)	(16x30)	(33x23)

感受一下9种先验框的尺寸, 下图中蓝色框为聚类得到的先验框。黄色框式ground truth, 红框是对象中心点所在的网格。

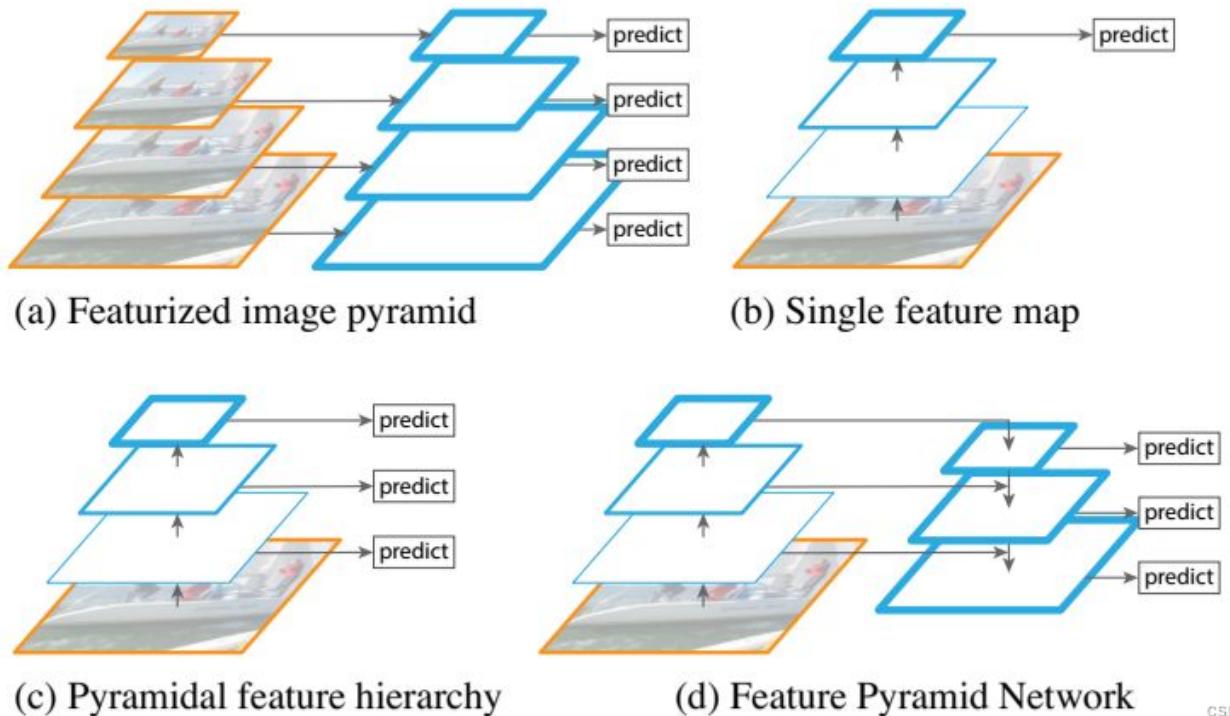


这里注意**bounding box**与**anchor box**的区别： Bounding box输出的是框的位置（中心坐标与宽高）， confidence以及N个类别。anchor box只是一个尺度即只有宽高。

### 3. 多尺度预测：更好地对应不同大小的目标物体

每种尺度预测3个box, anchor的设计方式仍然使用聚类, 得到9个聚类中心, 将其按照大小均分给3个尺度.

尺度1: 在基础网络之后添加一些卷积层再输出box信息. 尺度2: 从尺度1中的倒数第二层的卷积层上采样( $\times 2$ )再与最后一个 $16 \times 16$ 大小的特征图相加, 再次通过多个卷积后输出box信息。相比尺度1变大2倍。 尺度3: 与尺度2类似, 使用了 $32 \times 32$ 大小的特征图。

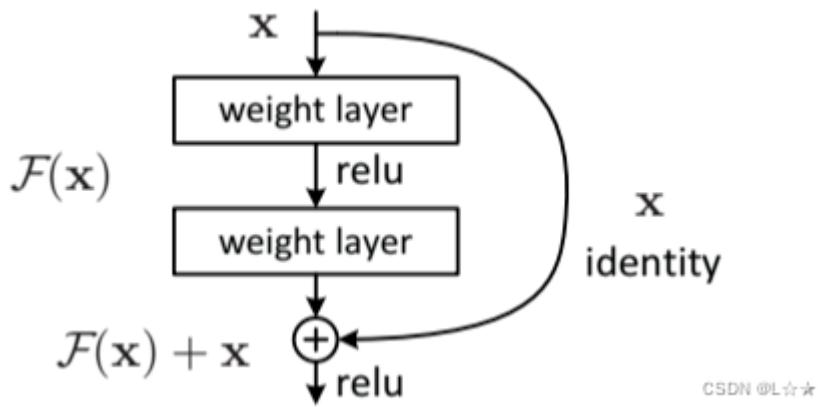


CSDN @L☆\*

前提：分辨率信息直接反映目标的像素数量。分辨率越高，像素数量越多，对细节表现越丰富。在目标检测中，语义信息主要用于区分前景（目标）和背景（非目标）。其不需要很多细节信息，分辨率大反而会降低语义信息。yolov3主要针对小目标检测的不足之处做出改进。具体形式：在网络预测的最后某些层进行上采样+拼接操作。

#### 4. ResNet残差结构：更好地获取物体特征

YOL0v3中使用了 ResNet 结构（对应着在上面的 YOL0v3 结构图中的 ResBlock）。ResBlock 是有一系列卷基层和一条 shortcut path 组成。shortcut 如下图所示：



CSDN @L☆★

图中曲线箭头代表的便是shortcut path。除此之外，此结构与普通的CNN结构并无区别。

但是，随着网络越来越深，学习特征的难度也就越来越大。但是如果我们在加一条shortcut path的话，学习过程就从直接学习特征，变成在之前学习的特征的基础上添加某些特征，来获得更好的特征。这样一来，一个复杂的特征H(x)，之前是独立一层一层学习的，现在就变成了这样一个模型。 $H(x)=F(x)+x$ ，其中x是shortcut开始时的特征，而F(x)就是对x进行的填补与增加，成为残差。因此学习的目标就从学习完整的信息，变成学习残差了。这样学习优质特征的难度就大大减小了。

## 5. 替换softmax层：对应多重label分类

Softmax层被替换为一个 $1 \times 1$ 的卷积层+logistic激活函数的结构。使用softmax层的时候其实已经假设每个输出仅对应某一个单个的class，但是在某些class存在重叠情况（例如woman和person）的数据集中，使用softmax就不能使网络对数据进行很好的拟合。

## LOSS Function

YOLOv3重要改变之一：**No more softmaxing the classes**。YOLOv3现在对图像中检测到的对象执行多标签分类。

logistic回归用于对anchor包围的部分进行一个目标性评分(objectness score)，即这块位置是目标的可能性有多大。这一步是在predict之前进行的，可以去掉不必要的anchor，可以减少计算量。

如果模板框不是最佳的即使它超过我们设定的阈值，我们还是不会对它进行predict。不同于Faster R-CNN的是，YOLOv3只会对1个prior进行操作，也就是那个最佳prior。而logistic回归就是用来从9个anchor priors中找到objectness score(目标存在可能性得分)最高的那一个。logistic回归就是用曲线对prior相对于 objectness score映射关系的线性建模。

```

lxy, lwh, lcls, lconf = ft([0]), ft([0]), ft([0]), ft([0])
txy, twh, tcls, indices = build_targets(model, targets) #在13 26 52维度中
找到大于iou阈值最适合的anchor box 作为targets
#txy[维度(0:2),(x,y)] twh[维度(0:2),(w,h)] indices=[0,anchor索引, gi, gj]

# Define criteria
MSE = nn.MSELoss()
CE = nn.CrossEntropyLoss()
BCE = nn.BCEWithLogitsLoss()

# Compute losses
h = model.hyp # hyperparameters
bs = p[0].shape[0] # batch size
k = h['k'] * bs # loss gain
for i, pi0 in enumerate(p): # layer i predictions, i
    b, a, gj, gi = indices[i] # image, anchor, gridx, gridy
    tconf = torch.zeros_like(pi0[..., 0]) # conf

    # Compute losses
    if len(b): # number of targets
        pi = pi0[b, a, gj, gi] # predictions closest to anchors 找到p中
与targets对应的数据lxy
        tconf[b, a, gj, gi] = 1 # conf
        # pi[..., 2:4] = torch.sigmoid(pi[..., 2:4]) # wh power loss
(uncomment)

        lxy += (k * h['xy']) * MSE(torch.sigmoid(pi[..., 0:2]), txy[i])
# xy loss
        lwh += (k * h['wh']) * MSE(pi[..., 2:4], twh[i]) # wh yolo loss
        lcls += (k * h['cls']) * CE(pi[..., 5:], tcls[i]) # class_conf
loss

        # pos_weight = ft([gp[i] / min(gp) * 4.])
        # BCE = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
        lconf += (k * h['conf']) * BCE(pi0[..., 4], tconf) # obj_conf loss
    loss = lxy + lwh + lconf + lcls

```

以上是一段pytorch框架描述的YOLOv3的loss\_function代码。忽略恒定系数不看，以下我想着重说几点：

- 首先，YOLOv3要先build target，因为我们知道正样本是label与anchor box iou大于0.5的组成，所以我们根据label找到对应的anchor box。如何找出label中存放着[image,class,x(归一化),y,w(归一化),h],我们可以用这些坐标在对应13×13 Or 26×26 or 52×52的map中分别于9个anchor算出iou，找到符合要求的，把索引与位置记录好。用记录好的索引位置找到predict的anchor box。
- x y w h是由均方差来计算loss的，其中预测的xy进行sigmoid来与label xy求差，label xy是grid cell中心点坐标，其值在0-1之间，所以predict出的xy要sigmoid。

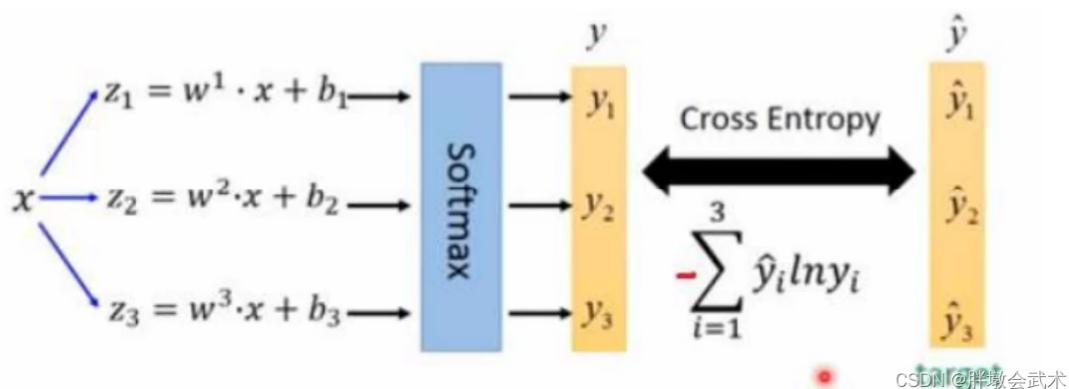
- 分类用的多类别交叉熵，置信度用的二分类交叉熵。只有正样本才参与class, xywh的loss计算，负样本只参与置信度loss。

将yolov2的单标签分类改进为yolov3的多标签分类。即softmax()分类函数更改为logistic()分类器。具体形式：逻辑分类器通过对每个类别都进行二分类，以实现多标签分类。使用sigmoid函数将特征图的结果约束在[0~1]之间，如果有一个或多个值大于设定阈值，就认定该目标框所对应的目标属于该类。多个值称为多标签对象。（如：一个人有woman、person、地球人等多个标签）

## ✓ softmax层替代

📝 物体检测任务中可能一个物体有多个标签

📝 logistic激活函数来完成，这样就能预测每一个类别是/不是



## 常见问题

YOLOv3为什么不使用Softmax对每个框进行分类？

主要考虑因素有两个：

- Softmax使得每个框分配一个类别（score最大的一个），而对于Open Images这种数据集，目标可能有重叠的类别标签，因此Softmax不适用于多标签分类。
- Softmax可被独立的多个logistic分类器替代，且准确率不会下降。

## 分类损失采用 binary cross-entropy loss

Bounding box的预测公式中，为什么使用sigmoid函数呢？YOLOv3不预测边界框中心的绝对坐标，它预测的是偏移量，预测的结果通过一个sigmoid函数，迫使输出的值在0~1之间。

例如，若对中心的预测是(0.4,0.7)，左上角坐标是(6,6)，那么中心位于 $13 \times 13$ 特征地图上的(6.4,6.7)。若预测的x, y坐标大于1，比如(1.2,0.7)，则中心位于(7.2,6.7)。注意现在中心位于图像的第7排第8列单元格，这打破了YOLOv3背后的理论，因为如果假设原区域负责预测某个目标，目标的中心必须位于这个区域中，而不是位于此区域旁边的其他网格里。

为解决这个问题，输出是通过一个sigmoid函数传递的，该函数在0~1的范围内缩放输出，有效地将中心保持在预测的网格中。

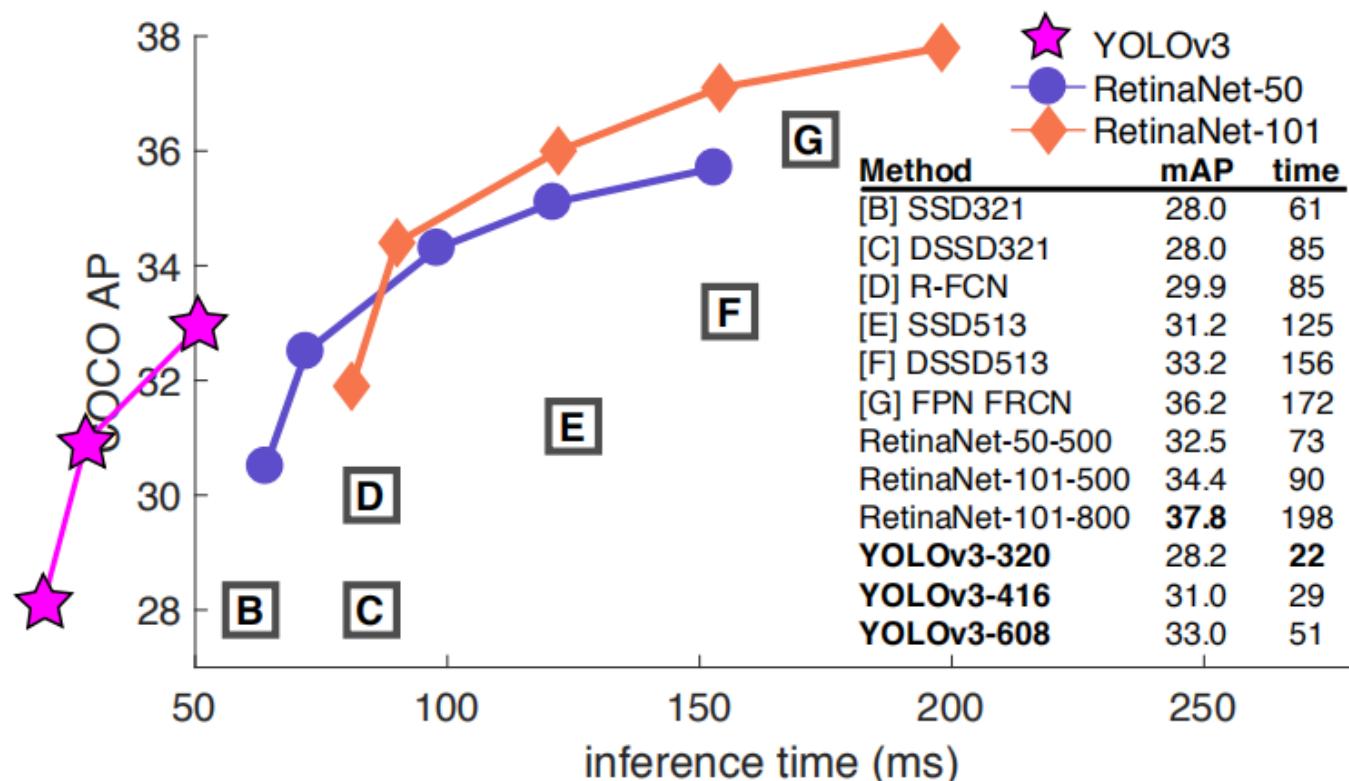


Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

[https://blog.csdn.net/qq\\_40716944](https://blog.csdn.net/qq_40716944)

## YOLOv4: Optimal Speed and Accuracy of Object Detection

论文下载：<https://arxiv.org/abs/2004.10934>

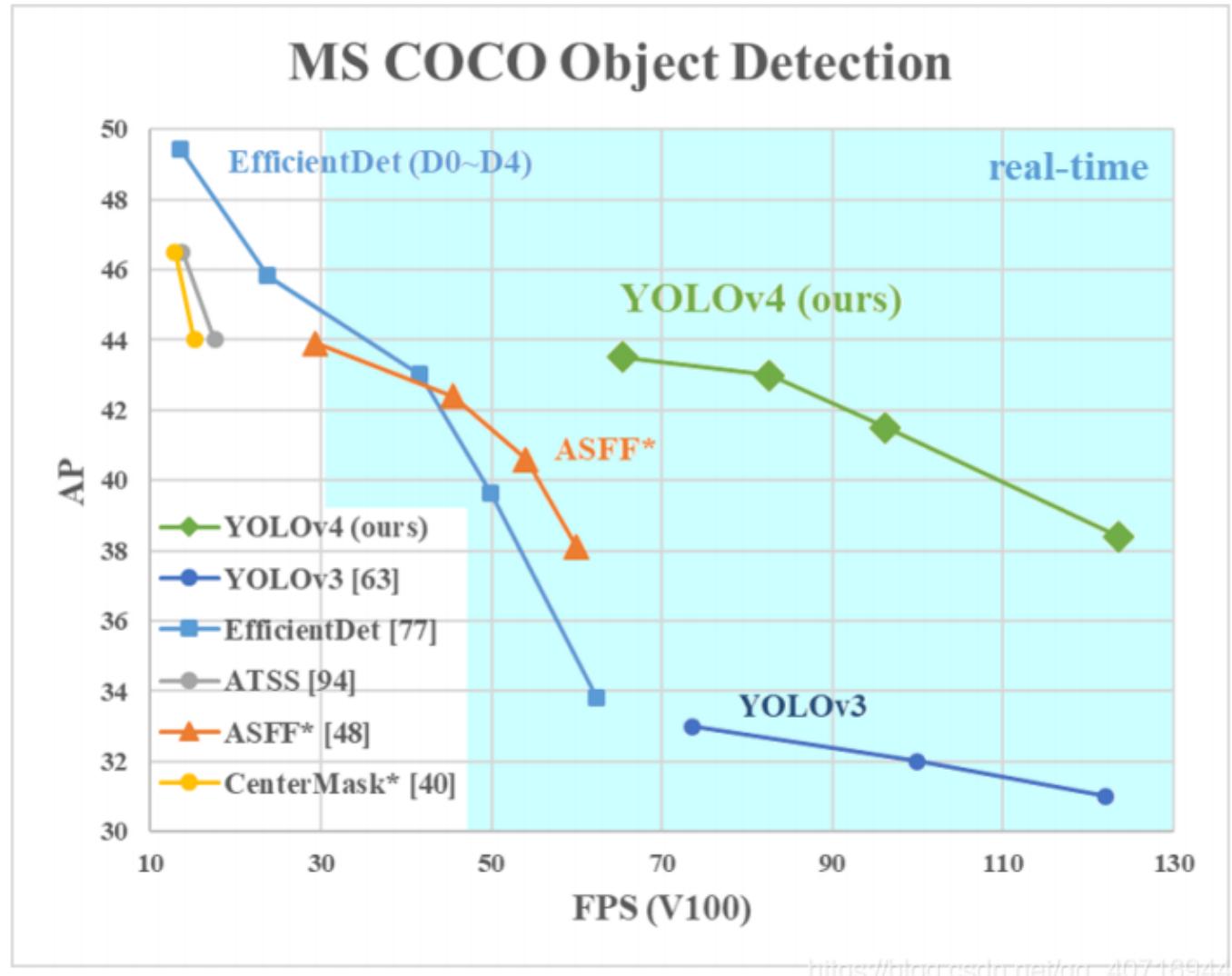
论文：<https://arxiv.org/abs/2004.10934>

代码下载：<https://github.com/AlexeyAB/darknet>

## 核心思想

从本质上, **YOLOv4**就是**筛选**了一些从 **YOLOv3**发布至今, 被用在各式各样检测器上, 能够**提高检测精度的 tricks**, 并以 **YOLOv3**为基础进行改进的目标检测模型。**YOLOv4**在保证速度的同时, 大幅提高模型的检测精度(相较于 **YOLOv3**)。

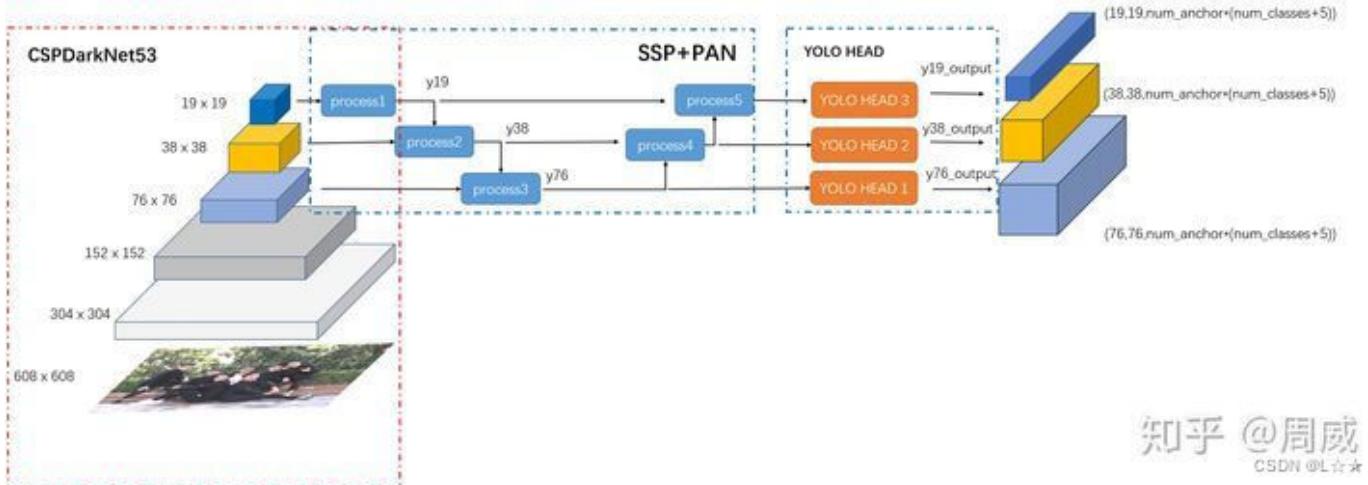
YOLOv4其实是一个结合了大量前人研究技术, 加以组合并进行适当创新的算法, 实现了速度和精度的完美平衡。可以说有许多技巧可以提高卷积神经网络(CNN)的准确性, 但是某些技巧仅适合在某些模型上运行, 或者仅在某些问题上运行, 或者仅在小型数据集上运行; 我们来码一码这篇文章里作者都用了哪些调优手段: 加权残差连接(WRC),跨阶段部分连接(CSP),跨小批量标准化(CmBN),自对抗训练(SAT),Mish激活,马赛克数据增强,CmBN,DropBlock正则化,CIoU Loss等等。经过一系列的堆料, 终于实现了目前最优的实验结果: 43.5%的AP(在Tesla V100上, MS COCO数据集的实时速度约为65FPS)。



- 核心思想：yolov4筛选了一些从yolov3发布至今，被用在各式各样检测器上，能够提高检测精度的tricks，并加以组合及适当创新的算法，实现了速度和精度的完美平衡。虽然有许多技巧可以提高卷积神经网络CNN的准确性，但是某些技巧仅适合在某些模型上运行，或者仅在某些问题上运行，或者仅在小型数据集上运行。
- 主要调优手段：加权残差连接(WRC)、跨阶段部分连接(CSP)、跨小批量标准化(CmBN)、自对抗训练(SAT)、Mish激活、马赛克数据增强、CmBN、DropBlock正则化、CIoU Loss等等。经过一系列的堆料，终于实现了目前最优的实验结果：43.5%的AP(在Tesla V100上，MS COCO数据集的实时速度约为65FPS)。

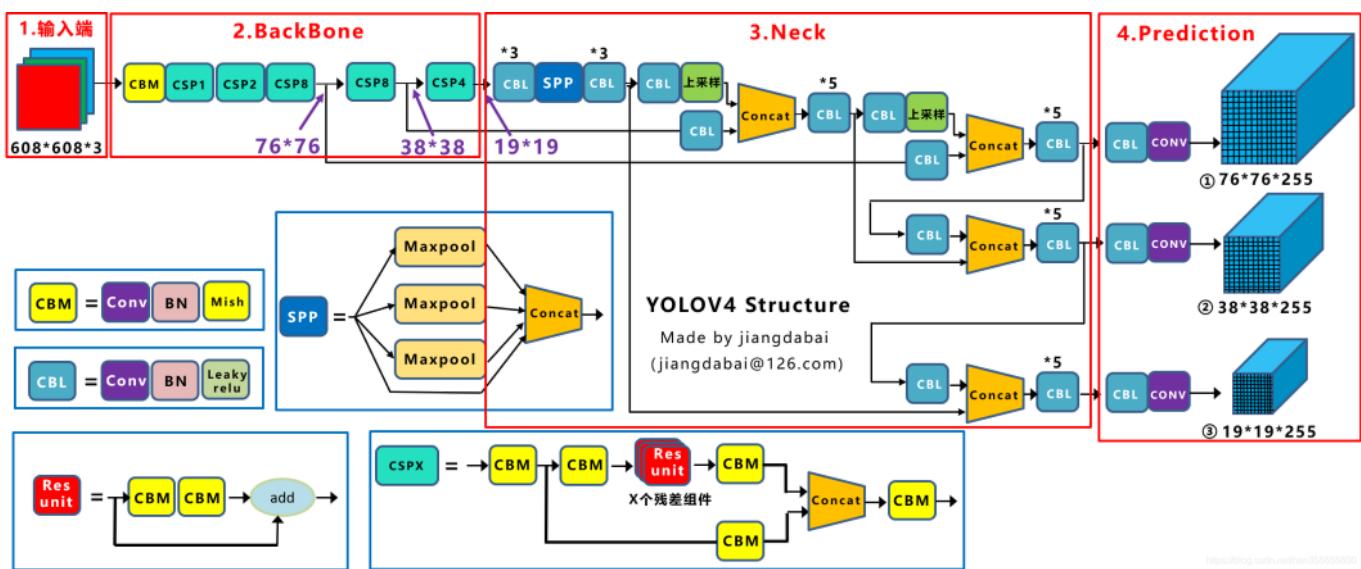
## 网络模型

由于 YOLOV4实在是综合了太多的技巧，就从知乎一个大佬那借图一展，我主要从下一节的网络改进来详细讲



知乎 @周威  
CSDN @L☆☆

先直接上 YOLOv4的整体原理图(来源网络)如下：



yolov4的CSPDarknet53与yolov3的Darknet-53相比，主要区别：

(1) 将原来的Darknet53与CSPNet进行结合，形成Backbone网络。 (2) 采用SPPNet适应不同尺寸的输入图像大小，且可以增大感受野； (3) 采用SAM引入空间注意力机制； (4) 采用PANet充分利用了特征融合； (5) 激活函数由Mish替换Leaky ReLU；在yolov3中，每个卷积层之后包含一个批归一化层和一个Leaky ReLU。而在yolov4的主干网络CSPDarknet53中，使用Mish替换原来的Leaky ReLU。

CSPDarknet53网络架构：

Type	Filters	Size	Output
Convolutional	32	3x3	256x256
Convolutional	64	3x3/2	128x128
CrossStagePartial			
1x	32	1x1	
	64	3x3	
	Residual		128x128
Convolutional	128	3x3/2	64x64
CrossStagePartial			
2x	64	1x1	
	64	3x3	
	Residual		64x64
Convolutional	256	3x3/2	32x32
CrossStagePartial			
8x	128	1x1	
	128	3x3	
	Residual		32x32
Convolutional	512	3x3/2	16x16
CrossStagePartial			
8x	256	1x1	
	256	3x3	
	Residual		16x16
Convolutional	1024	3x3/2	8x8
CrossStagePartial			
4x	512	1x1	
	512	3x3	
	Residual		8x8
Avgpool		Global	
Connected			1000
Softmax			

CSDN @胖墩会武术

跨阶段部分网络 (Cross Stage Partial Networks, CSPNet)

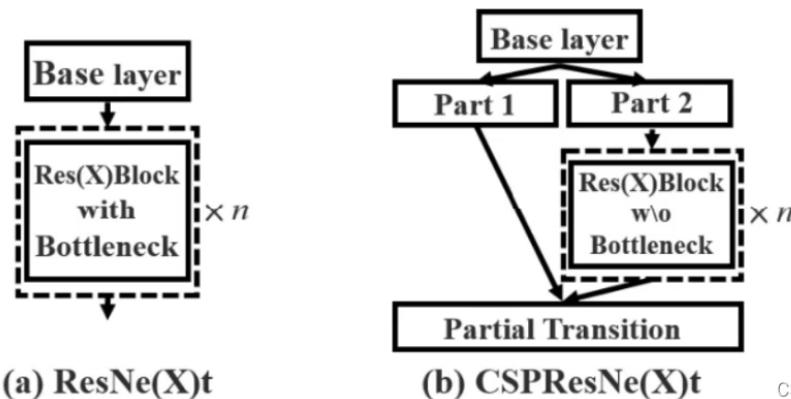
背景： 2019年Chien-Yao Wang等人提出，用来解决网络优化中的重复梯度信息问题，在ImageNet dataset和MS COCO数据集上有很好的测试效果。且易于实现，在ResNet、ResNeXt和DenseNet网络结构上都能通用。目的：实现更丰富的梯度组合，同时减少计算量。具体方式：将基本层的特征图分成两部分：11、主干部分继续堆叠原来的残

差块；22、支路部分则相当于一个残差边，经过少量处理直接连接到最后。

## ✓ CSPNet (Cross Stage Partial Network)

📝 每一个block按照特征图的channel维度拆分成两部分

📝 一份正常走网络，另一份直接concat到这个block的输出



CSDN @胖墩会武术

## 网络改进

YOLO v4由于结合了太多的技巧，所以就大概列一部分主要的，如下：

- 相较于YOLO v3的DarkNet53，YOLO v4用了CSPDarkNet53
- 相较于YOLO v3的FPN，YOLO v4用了SPP+PAN
- CutMix数据增强和马赛克（Mosaic）数据增强
- DropBlock正则化

**BackBone训练策略：** 数据增强、自对抗训练、DropBlock正则化、类标签平滑、CIoU 损失函数、DIoU-NMS等。

### 1. BackBone:CSPDarknet53

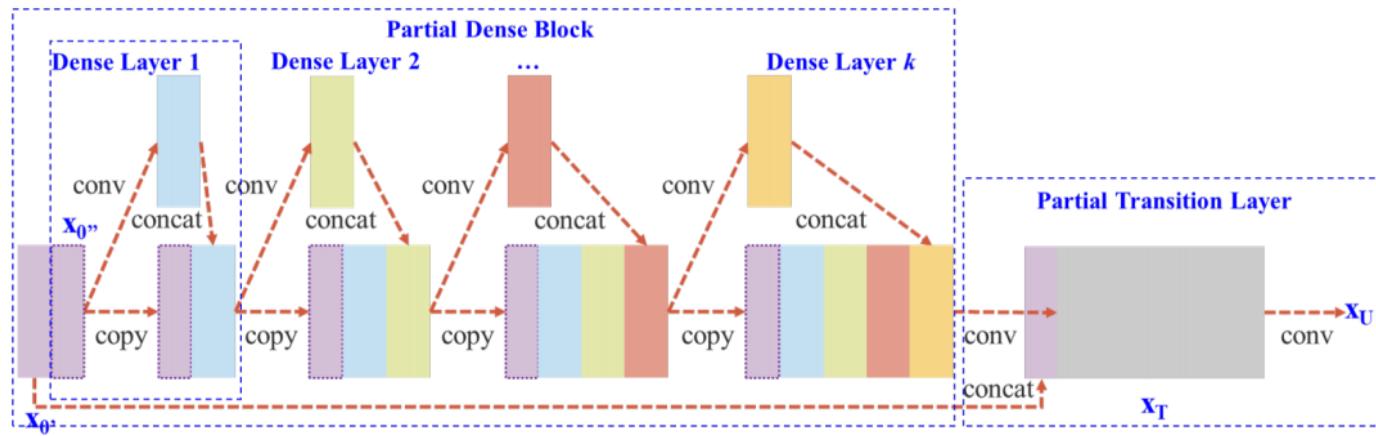
作者在选在BackBone的时候选择了CSPDarknet53，原因也很简单，在论文中也有提到，当时在目标检测领域，该网络精度最好。

有兴趣的可以去看CSPDarknet53的原文，不多讲述：[CSPNet: A New Backbone that can Enhance Learning Capability of CNN](#)

---

我们前面知道在YOLOv3中，特征提取网络使用的是Darknet53，而在YOLOv4中，对Darknet53做了一点改进，借鉴了CSPNet，CSPNet全称是Cross Stage Partial Networks，也就是跨阶段局部网络。CSPNet解决了其他大型卷积神经网络框架

Backbone中网络优化的梯度信息重复问题，将梯度的变化从头到尾地集成到特征图中，因此减少了模型的参数量和FLOPS数值，既保证了推理速度和准确率，又减小了模型尺寸。如下图：



CSPNet实际上是基于Densnet的思想，复制基础层的特征映射图，通过dense block发送副本到下一个阶段，从而将基础层的特征映射图分离出来。这样可以有效缓解梯度消失问题(通过非常深的网络很难去反推丢失信号)，支持特征传播，鼓励网络重用特征，从而减少网络参数数量。CSPNet思想可以和ResNet、ResNeXt和DenseNet结合，目前主要有CSPResNext50 和CSPDarknet53两种改造Backbone网络

考虑到几方面的平衡：输入网络分辨率/卷积层数量/参数数量/输出维度。一个模型的分类效果好不见得其检测效果就好，想要检测效果好需要以下几点：

- 更大的网络输入分辨率——用于检测小目标
- 更深的网络层——能够覆盖更大面积的感受野
- 更多的参数——更好的检测同一图像内不同size的目标

这样最终的CSPDarknet53结构就如下图：

Type	Filters	Size	Output
Convolutional	32	3x3	256x256
Convolutional	64	3x3/2	128x128
CrossStagePartial			
1x	Convolutional	32	1x1
	Convolutional	64	3x3
	Residual		128x128
CrossStagePartial			
2x	Convolutional	128	3x3/2
	CrossStagePartial		
	Convolutional	64	1x1
8x	Convolutional	64	3x3
	Residual		64x64
	Convolutional	256	3x3/2
CrossStagePartial			
8x	Convolutional	128	1x1
	Convolutional	128	3x3
	Residual		32x32
CrossStagePartial			
4x	Convolutional	512	3x3/2
	CrossStagePartial		
	Convolutional	256	1x1
4x	Convolutional	256	3x3
	Residual		16x16
	Convolutional	1024	3x3/2
CrossStagePartial			
4x	Convolutional	512	1x1
	Convolutional	512	3x3
	Residual		8x8

Avgpool

Global

为了增大感受野，作者还使用了 **SPP-block**，使用 **PANet**代替 **FPN**进行参数聚合以适用于不同 **level** 的目标检测。

Connected Softmax

1000

## 2. Neck:SPP+PAN

**SSP**不多讲，有兴趣看何凯明大神的论文

**SSP** : Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition

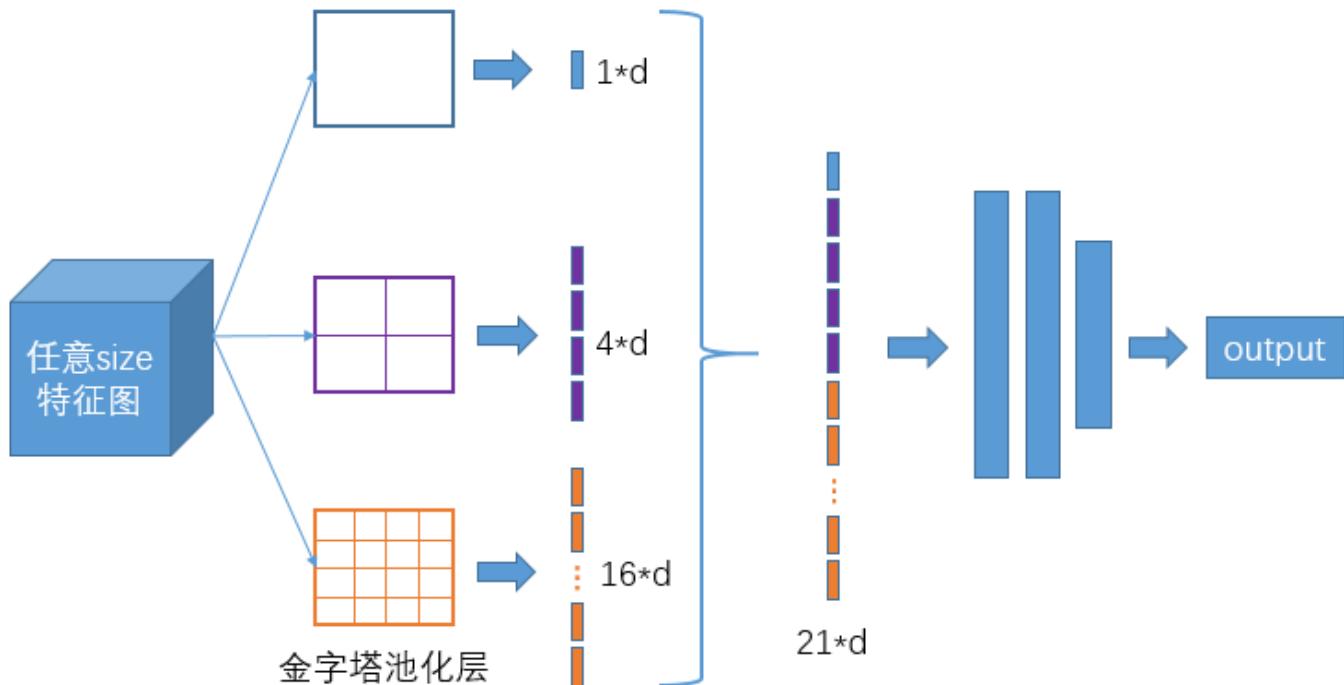
而SSP+PAN借鉴的是18年CVPR的PANet，当时主要应用于图像分割领域，作者将其拆分应用到YOLOv4中，进一步提高特征提取的能力。

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

**SPP-Net结构**我们之前也有学过，**SPP-Net**全称 **Spatial Pyramid Pooling**

**Networks**，当时主要是用来解决不同尺寸的特征图如何进入全连接层的，直接看下图，下图中对任意尺寸的特征图直接进行固定尺寸的池化，来得到固定数量的特征。



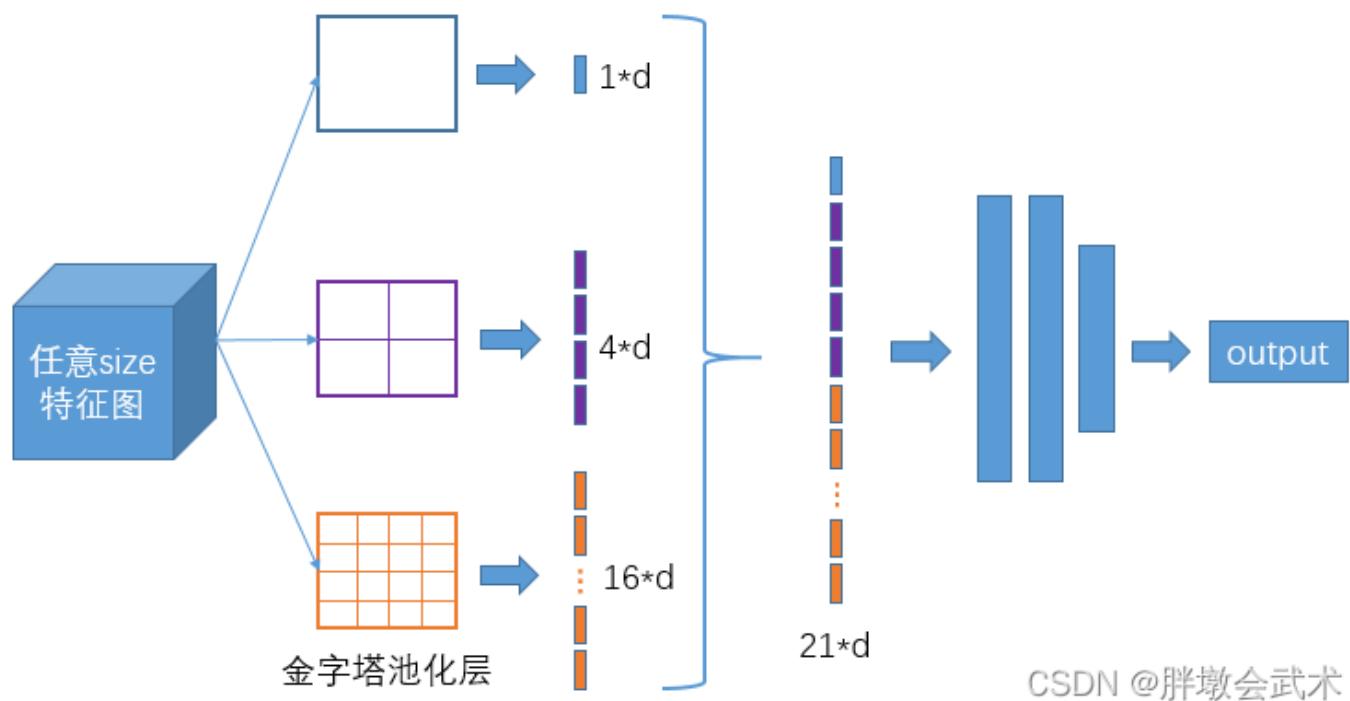
如上图，以3个尺寸的池化为例，对特征图进行一个最大值池化，即一张特征图得取其最大值，得到 $d$ ( $d$ 是特征图的维度)个特征；对特征图进行网格划分为 $2 \times 2$ 的网格，然后对每个网格进行最大值池化，那么得到 $4d$ 个特征；同样，对特征图进行网格划分为 $4 \times 4$ 个网格，对每个网格进行最大值池化，得到 $16 \times d$ 个特征。接着将每个池化得到的特征合

起来即得到固定长度的特征个数（特征图的维度是固定的），接着就可以输入到全连接层中进行训练网络了。用到这里是为了增加感受野。

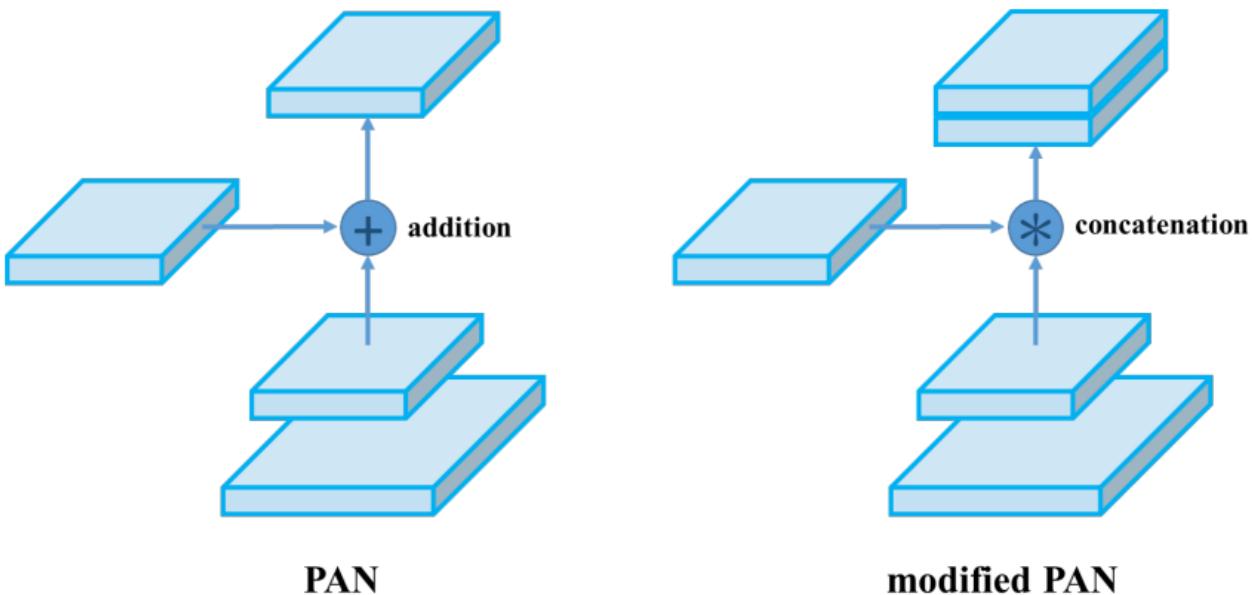
空间金字塔池化网络（Spatial Pyramid Pooling Network, SPPNet）

论文地址：[Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition](#)

yolov1背景：yolov1训练时的分辨率： $224 \times 224$ ；测试时： $448 \times 448$ 。yolov2背景：yolov2保持yolov1的操作不变，但在原训练的基础上又加上了（10个epoch）的 $448 \times 448$ 高分辨率样本进行微调，使网络特征逐渐适应 $448 \times 448$ 的分辨率；然后再使用 $448 \times 448$ 的样本进行测试，缓解了分辨率突然切换造成的影响。**目的**：使得网络模型的输入图像不再有固定尺寸的大小限制。**=通过最大池化将不同尺寸的输入图像变得尺寸一致=**。**优点**：增大感受野。如图是SPP中经典的空间金字塔池化层。



PAN结构, YOLOv4 使用 PANet(Path Aggregation Network)代替 FPN 进行参数聚合以适用于不同 level 的目标检测, PANet 论文中融合的时候使用的方法是 Addition, YOLOv4 算法将融合的方法由加法改为 Concatenation。如下图：

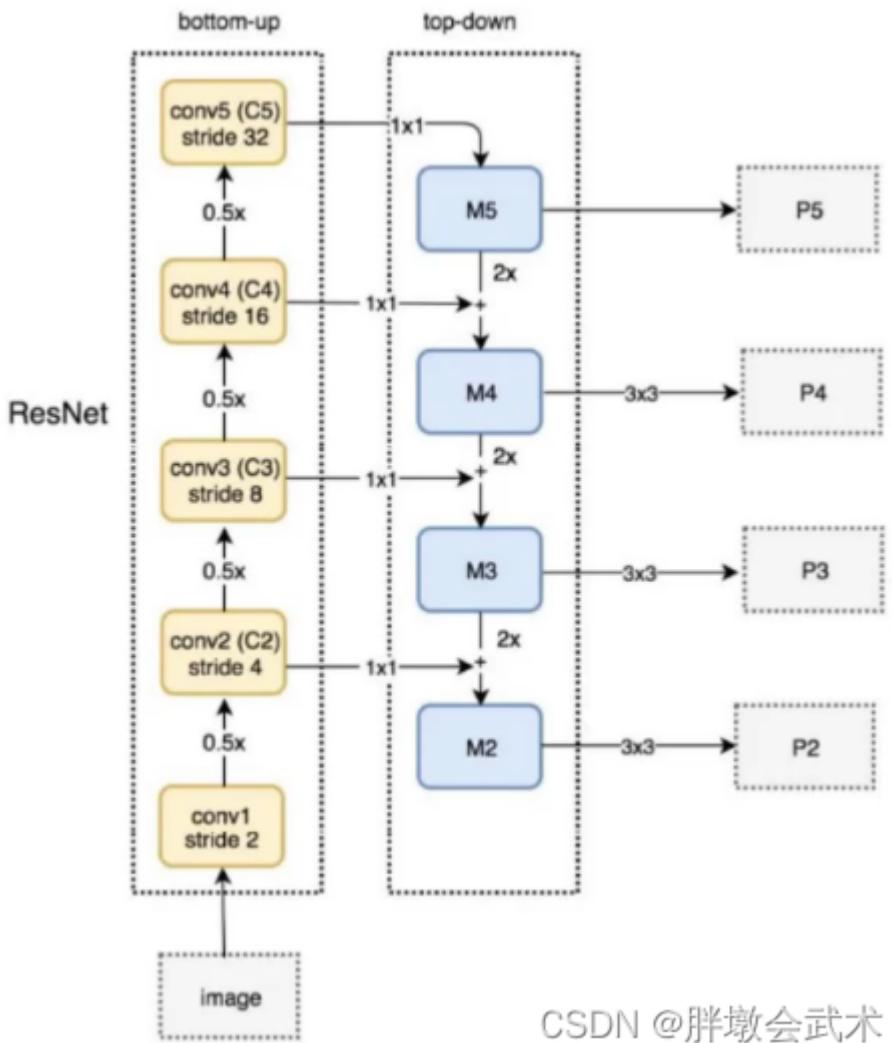


## 路径聚合网络 (Path Aggregation Network, PANet)

论文地址 (FPNet) : [Feature Pyramid Networks for Object Detection](#) 论文地址 (PANet) : [Path Aggregation Network for Instance Segmentation](#)

背景： PANet发表于CVPR2018，其是COCO2017实例分割比赛的冠军，也是目标检测比赛的第二名。具体方式：yolov4采用改进的PANet方法 优化历程： FPNet (Feature Pyramid Networks) -> PANet (Path Aggregation Network) -> 改进的PAN 优化原因：

(1) FPNet网络采取自上而下的方式，将高层特征逐层与中高层、中层、中底层、低层特征进行融合。缺点是无法自下而上融合，而PANet的优化了该部分不足，详见示意图的 (b) 部分。 (2) FANet采用特征相加的融合方式，而yolo采用特征拼接的融合方式。加法可以得到一个加强版的特征图，但特征权重不大于1，而拼接可能得到大于1的特征图。FPNet示意图



PANet示意图

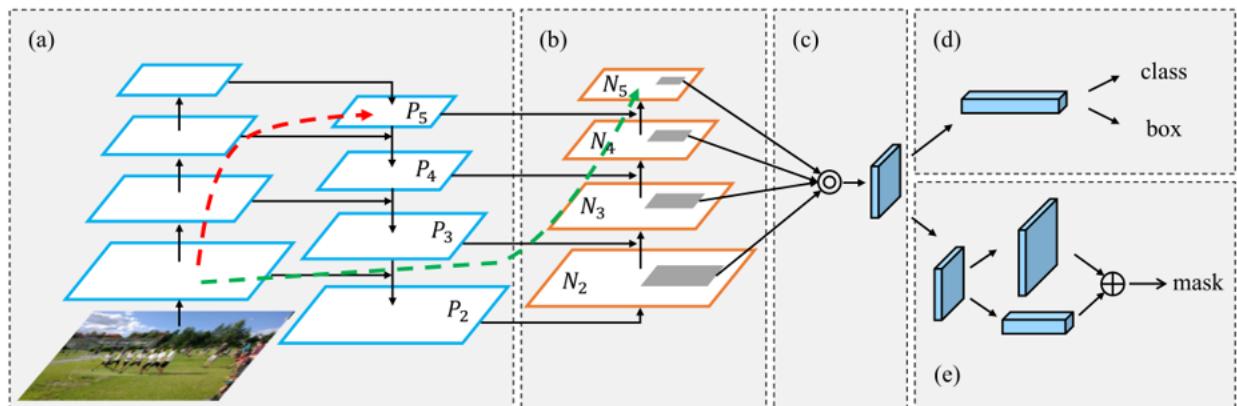
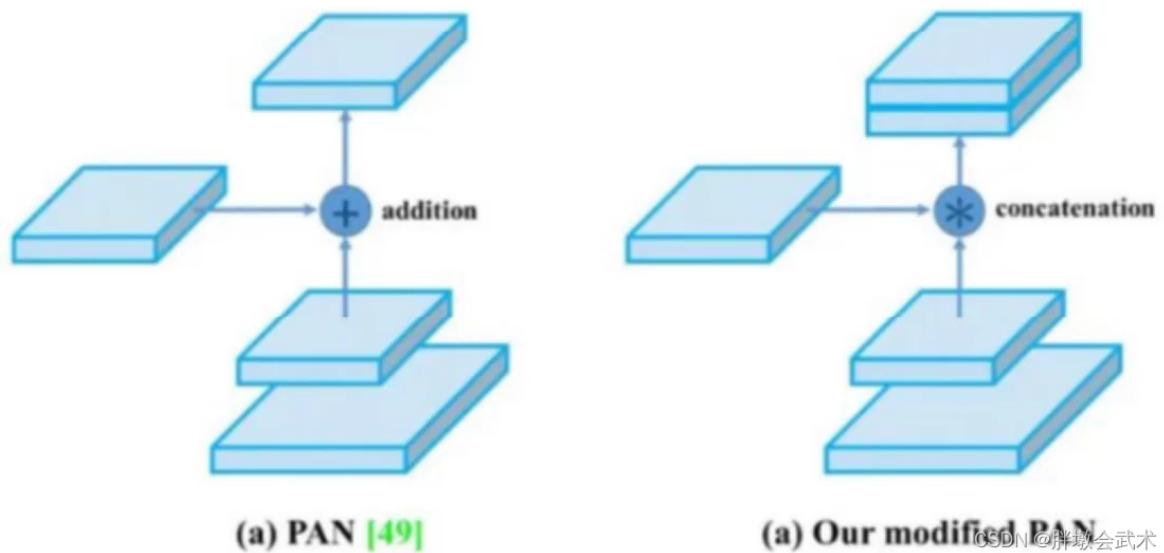


Figure 1. Illustration of our framework. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) Box branch. (e) Fully-connected fusion. Note that we omit channel dimension of feature maps in (a) and (b) for brevity. CSDN @胖墩会武术

(a) FPNet: 通过融合高层特征来提升目标检测的效果。 (b) Bottom-up Path Augmentation: 通过融合低层特征（边缘形状等）来提升目标检测的效果。 (c) Adaptive Feature Pooling: 采用拼接特征融合。详见下图。拼接相比加法，特征更明

显，可以提高检测效果。 (d) Fully-connected Fusion

## 💡 YOLOV4中并不是加法，而是拼接



### 3. CutMix数据增强和马赛克（Mosaic）数据增强

- CutMix数据增强：就是将一部分区域cut掉但不填充0像素而是随机填充训练集中的其他数据的区域像素值，分类结果按一定的比例分配
- Mosaic数据增强：核心思想是随机4张图拼合成一张送入网络

---

这里我们主要从数据增强，DropBlock正则化，类标签平滑方面来学习下BackBone训练策略。

数据增强,CutMix

YOLOv4选择用CutMix的增强方式，CutMix的处理方式也比较简单，同样也是对一对图片做操作，简单讲就是随机生成一个裁剪框Box,裁剪掉A图的相应位置，然后用B图片相应位置的ROI放到A图中被裁剪的区域形成新的样本，ground truth标签会根据patch的面积按比例进行调整，比如0.6像狗，0.4像猫，计算损失时同样采用加权求和的方式进行求解。这里借CutMix的地方顺带说下几种类似的增强方式：

	ResNet-50	Mixup	Cutout	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	<b>78.6</b> <b>(+2.3)</b>
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	<b>47.3</b> <b>(+1.0)</b>
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	<b>76.7</b> <b>(+1.1)</b>

上图是CutMix论文中作者对几种增强方式做的对比，结果显而易见，CutMix的增强方式在三个数据集上的表现都是最优的。其中Mixup是直接求和两张图，如同附身，鬼影一样，模型很难学到准确的特征图响应分布。Cutout是直接去除图像的一个区域，这迫使模型在进行分类时不能对特定的特征过于自信。然而，图像的一部分充满了无用的信息，这是一种浪费。在CutMix中，将图像的一部分剪切并粘贴到另一个图像上，使得模型更容易区分异类。

CutMix论文：<https://arxiv.org/pdf/1905.04899v2.pdf>

---

马赛克（Mosaic）数据增强 + CutMix数据增强

CutMix论文：<https://arxiv.org/pdf/1905.04899v2.pdf>

最大特点：使得yolov4只通过单CPU就能完成训练，不用再担心设备问题。具体方式：

11、采用常用的数据增强方法（如：亮度、饱和度、对比度；随机缩放、旋转、翻转等）对所有的图像进行数据增强； 22、采用CutMix数据增强方法。详细见下。 33、采

取马赛克 (Mosaic) 数据增强方法，即随机取四张图像拼接为一张图像。

## ✓ Mosaic data augmentation

↓

📎 方法很简单，参考CutMix然后四张图像拼接成一张进行训练

	ResNet-50	Mixup [48]	Cutout [3]	CutMix			
Image							
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4			
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	<b>78.6</b> <b>(+2.3)</b>			
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	<b>47.3</b> <b>(+1.0)</b>			
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	<b>76.7</b> <b>(+1.1)</b>			

CSDN @胖墩会武术

由图可得（左）：CutMix表现最优。（1）ResNet-50：采用常规的数据增强方法。如：调整亮度、饱和度、对比度；随机缩放、旋转、翻转等。（2）Mixup：将猫狗两张图像进行图像融合，其中狗和猫的权重参数都为0.5，故标签概率值都为0.5。（3）Cutout：随机删除/遮挡一个区域。（4）CutMix：随机删除/遮挡一个区域，并用A图像的一部分粘贴到B图像上。如：将狗头替换为猫头，其中狗和猫的权重参数分别为0.6、0.4，故标签softmax的概率值分别为0.6、0.4。

备注1：softmax能够得到当前输入属于各个类别的概率。备注2：标签（分类结果）会根据patch的面积按比例分配，计算损失时同样采用加权求和的方式进行求解。

```
# 图像融合: cv2.addWeighted(src1, alpha, src2, beta, gamma)
# 功能: 将两张相同shape的图像按权重进行融合
# 输入参数      src1/src2          图像1与图像2
#                  alpha/beta        图像1与图像2对应的权重 (融合后的图像偏向于权重高的一边)
#                  gamma            相当于 (y=a*x+b) 中的截距。用于调节亮度
# 权重融合公式: dst = src1 * alpha + src2 * beta + gamma
```

CSDN @胖墩会武术

数据增强的其余方法扩展：

## ✓ 数据增强

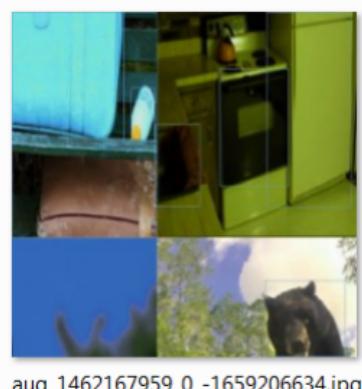
✏️ Random Erase: 用随机值或训练集的平均像素值替换图像的区域

✏️ Hide and Seek: 根据概率设置随机隐藏一些补丁



## Mosaic

Yolov4的Mosaic数据增强是参考CutMix数据增强，理论上类似。区别在于Mosaic是一种将4张训练图像合并成一张进行训练的数据增强方法(而不是CutMix中的2张)。这增强了对正常背景(context)之外的对象的检测，丰富检测物体的背景。此外，每个小批包含一个大的变化图像(4倍)，因此，减少了估计均值和方差的时需要大mini-batch的要求，降低了训练成本。如下图：



## 4. DropBlock正则化

Dropout 被广泛地用作全连接层的正则化技术，但是对于卷积层，通常不太有效。

Dropout 在卷积层不 work 的原因可能是由于卷积层的特征图中相邻位置元素在空间上共享语义信息，所以尽管某个单元被 dropout 掉，但与其相邻的元素依然可以保有该位置的语义信息，信息仍然可以在卷积网络中流通。

因此，针对卷积神经网络就需要一种结构形式的dropout 来正则化，即按块来丢弃。

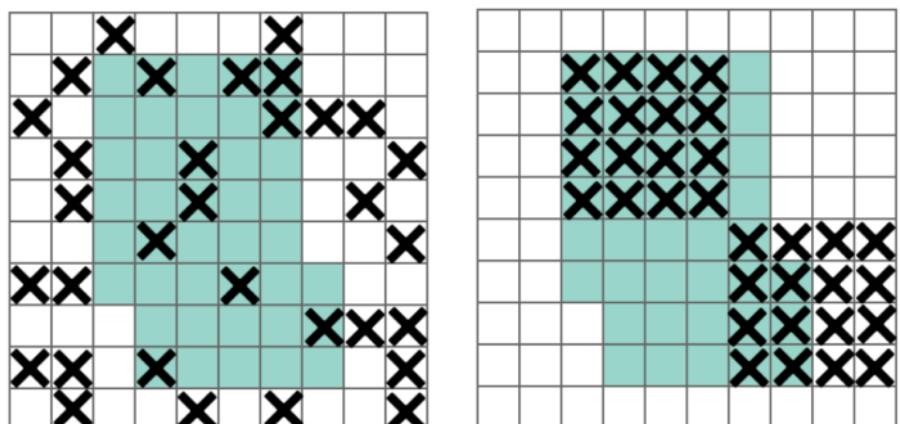
DropBlock是一种结构化的 dropout 形式，它将 feature map 相邻区域中的单元放在一起 drop 掉。除了卷积层外，在跳跃连接中应用 DropBlock 可以提高精度。

此外，在训练过程中，逐渐增加 dropped unit 的数量会有更好的准确性和对超参数选择的鲁棒性。

---

正则化技术有助于避免数据科学专业人员面临的最常见的问题，即过拟合。对于正则化，已经提出了几种方法，如L1和L2正则化、Dropout、Early Stopping和数据增强。这里YOLOv4用了DropBlock正则化的方法。

DropBlock方法的引入是为了克服Dropout随机丢弃特征的主要缺点，Dropout被证明是全连接网络的有效策略，但在特征空间相关的卷积层中效果不佳。DropBlock技术在称为块的相邻相关区域中丢弃特征。这样既可以实现生成更简单模型的目的，又可以在每次训练迭代中引入学习部分网络权值的概念，对权值矩阵进行补偿，从而减少过拟合。如下图：



DropBlock论文中作者最终在 ImageNet 分类任务上，使用 Resnet-50 结构，将精度提升 1.6% 个点，在 COCO 检测任务上，精度提升 1.6% 个点。

DropBlock论文：<https://arxiv.org/pdf/1810.12890.pdf>

对于分类问题，特别是多分类问题，常常把向量转换成one-hot-vector，而one-hot带来的问题：对于损失函数，我们需要用预测概率去拟合真实概率，而拟合one-hot的真实概率函数会带来两个问题：

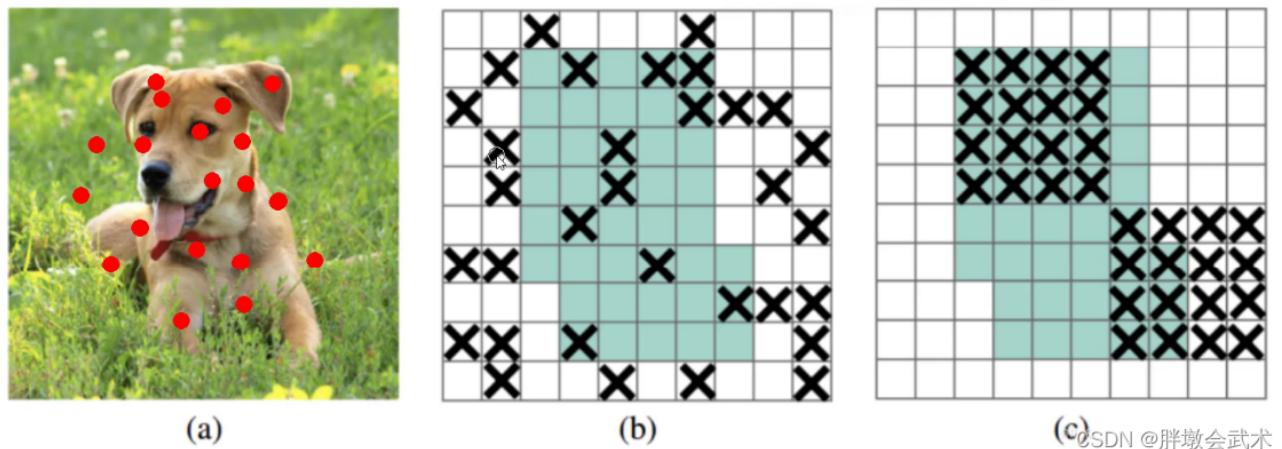
无法保证模型的泛化能力，容易造成过拟合；全概率和0概率鼓励所属类别和其他类别之间的差距尽可能加大，而由梯度有界可知，这种情况很难适应。会造成模型过于相信预测的类别。对预测有100%的信心可能表明模型是在记忆数据，而不是在学习。标签平滑调整预测的目标上限为一个较低的值，比如0.9。它将使用这个值而不是1.0来计算损失。这个概念缓解了过度拟合。说白了，这个平滑就是一定程度缩小label中min和max的差距，label平滑可以减小过拟合。所以，适当调整label，让两端的极值往中间凑，可以增加泛化性能。

## 改进的Dropout (DropBlock)

- **b图**：=Dropout是随机删除一些神经元=（如：**a图**的红点），但对于整张图来说，效果并不明显。比如：眼睛被删除，我们仍然可以通过眼睛的周边特征（眼角、眼圈等）去近似识别。
- **c图**：=DropBlock是随机删除一大块神经元。=如：将狗头的左耳全部删除。

### ✓ DropBlock

之前的dropout是随机选择点(b)，现在吃掉一个区域



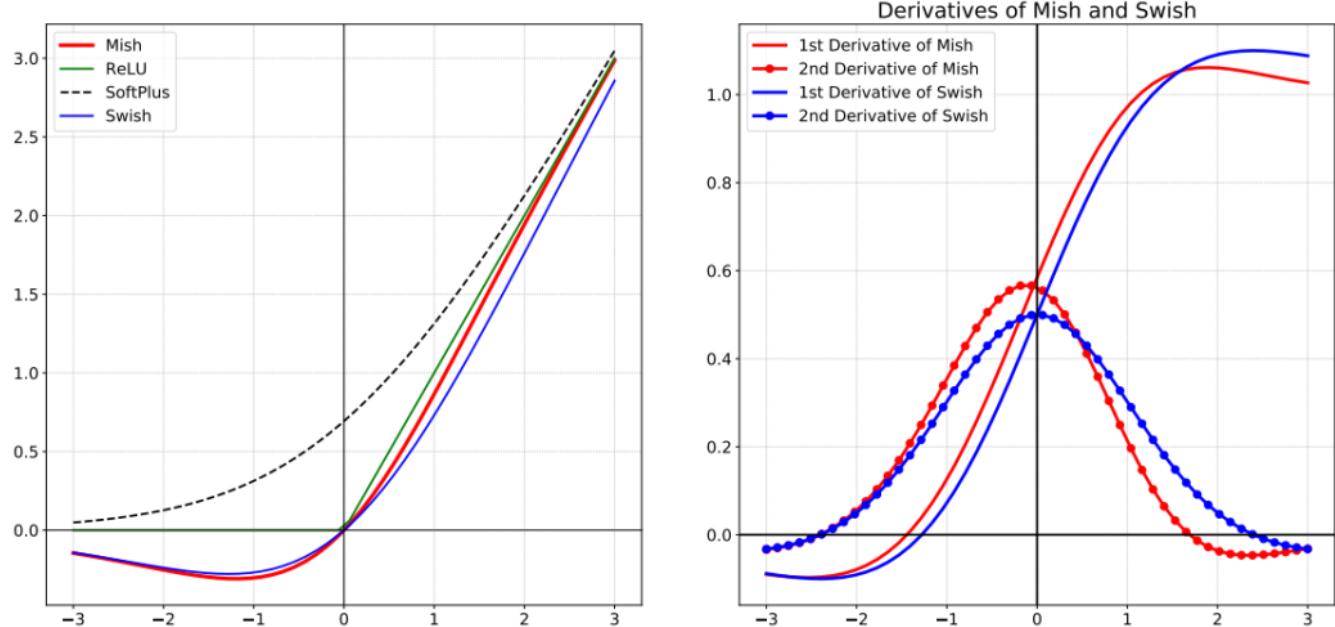
## 5. BackBone推理策略

### Mish激活函数

对激活函数的研究一直没有停止过，ReLU还是统治着深度学习的激活函数，不过，这种情况有可能会被 Mish改变。Mish是另一个与 ReLU和 Swish非常相似的激活函数。正如论文所宣称的那样，Mish可以在不同数据集的许多深度网络中胜过它们。公式如下：

$$y = x * \tanh(\ln(1 + e^x))$$

Mish是一个平滑的曲线，平滑的激活函数允许更好的信息深入神经网络，从而得到更好的准确性和泛化；在负值的时候并不是完全截断，允许比较小的负梯度流入。实验中，随着层深的增加，ReLU激活函数精度迅速下降，而Mish激活函数在训练稳定性、平均准确率(1%-2.8%)、峰值准确率(1.2% - 3.6%)等方面都有全面的提高。如下图：



**Mish**论文：<https://arxiv.org/pdf/1908.08681.pdf>

论文地址：[Mish: A Self Regularized Non-Monotonic Activation Function](#)

Mish在负值的时候并不是完全截断，允许比较小的负梯度流入。实验中，随着层深的增加，ReLU激活函数精度迅速下降，而Mish激活函数在训练稳定性、平均准确率(1%-2.8%)、峰值准确率(1.2% - 3.6%)等方面都有全面的提高。[22个激活函数](#)

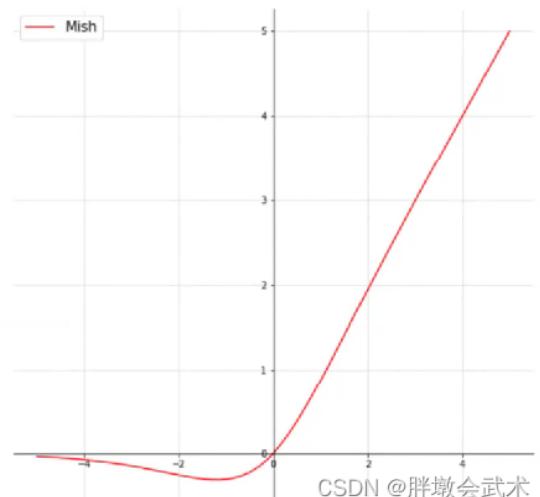
### ✓ Mish：（也许就是明日之星）

📝 别一棒子全给打死，给个改过自新的机会

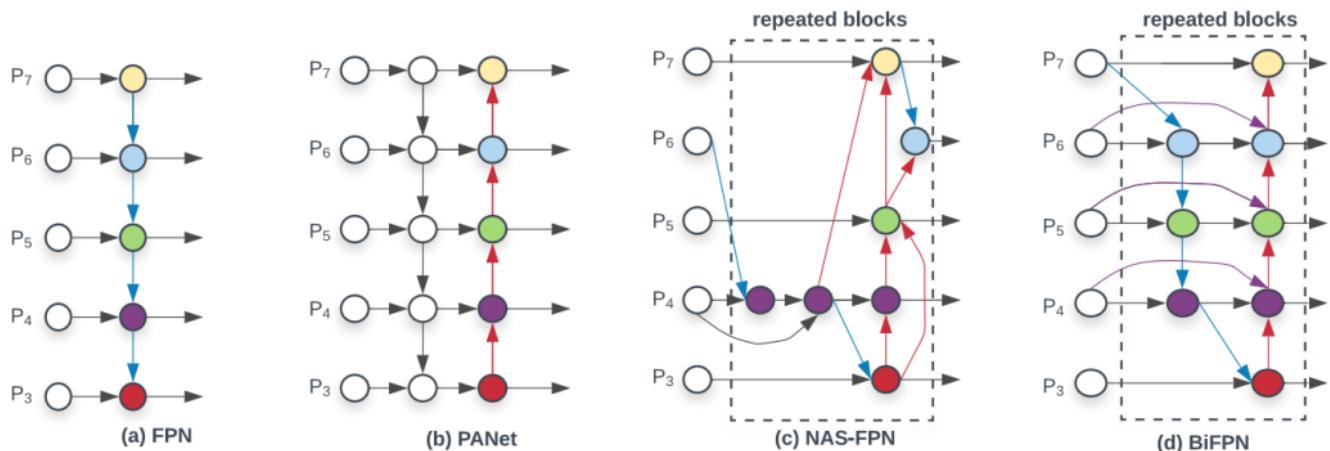
📝 Relu有点太绝对了，Mish更符合实际

📝 公式： $f(x) = x \cdot \tanh(\ln(1 + e^x))$

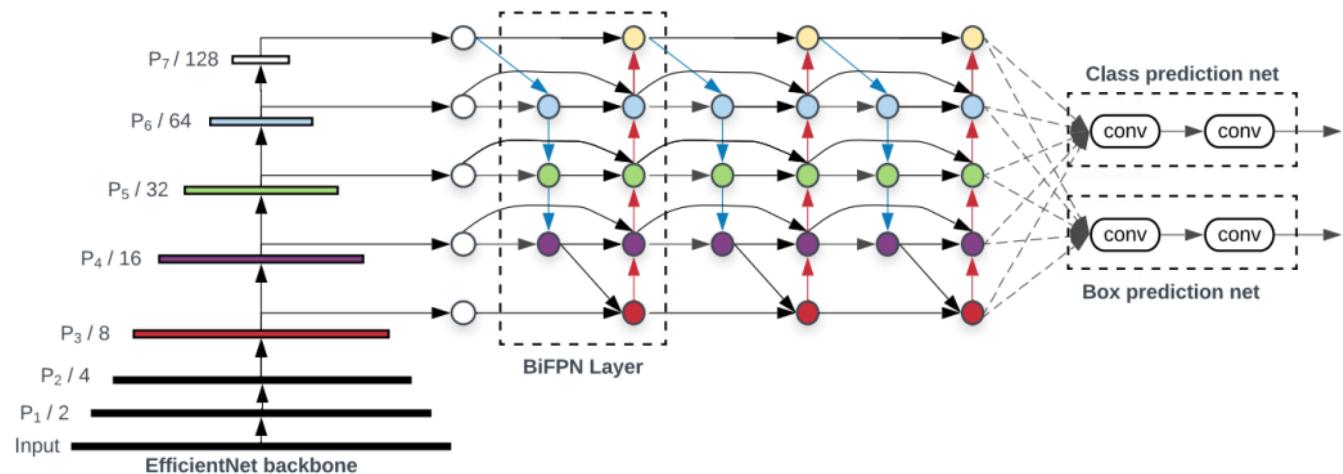
📝 但是计算量确实增加了，效果会提升一点



MiWRC是Multi-input weighted residual connections的简称，在BiFPN中，提出了用MiWRC来执行标尺度级重加权，添加不同尺度的特征映射。我们已经讨论了FPN和PAN作为例子。下面的图(d)显示了另一种被称为BiFPN的neck设计，根据BiFPN的论文，该设计具有更好的准确性和效率权衡。



上图中 (a)FPN引入自顶向下的路径，将多尺度特征从3级融合到7级(P3-P7)；(b)PANET在FPN之上增加一个额外的自下而上的路径；(c)NAS-FPN使用神经网络搜索找到一个不规则的特征拓扑网络，然后重复应用同一块拓扑结构；(d)是这里的BiFPN，具有更好的准确性和效率权衡。将该neck放到整个整个网络的连接中如下图：



上图采用 **EfficientNet**作为骨干网络，**BiFPN**作为特征网络，共享 **class/box**预测网络。基于不同的资源约束，**BiFPN**层和类/盒网层都被重复多次。

**BiFPN**论文：<https://arxiv.org/pdf/1911.09070.pdf>

## 6. 检测头训练策略

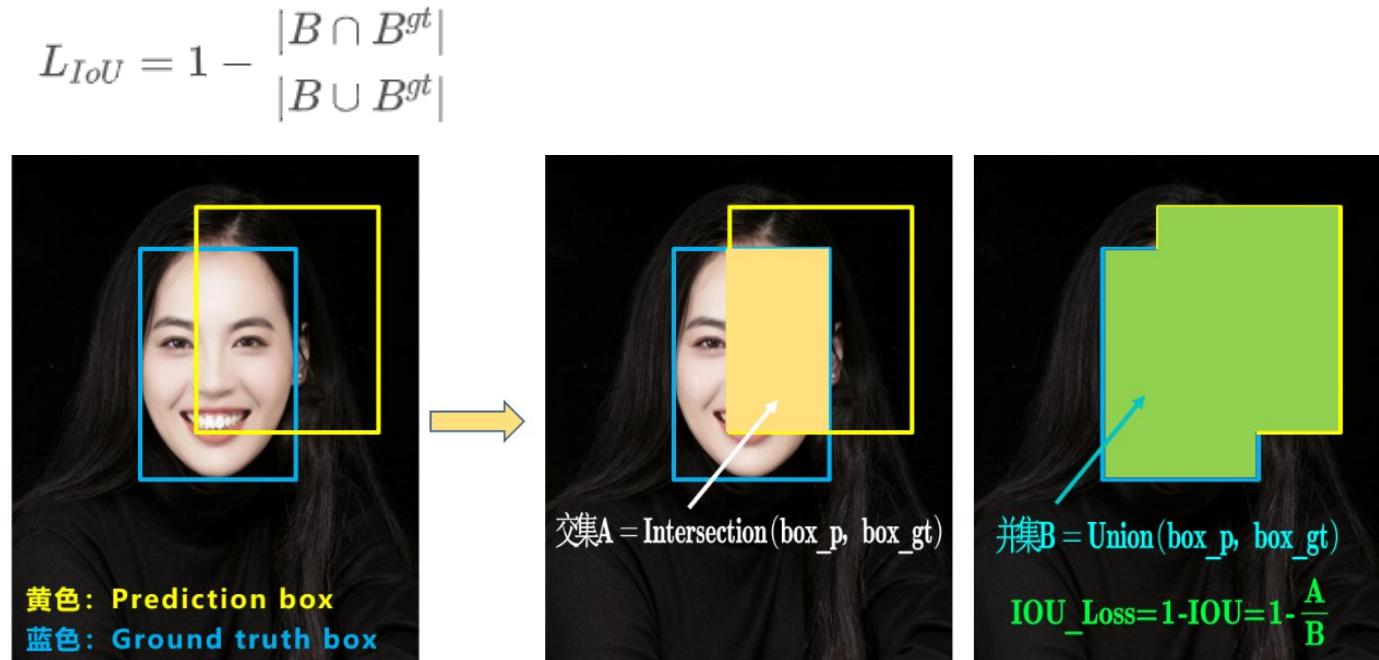
### CIoU-loss

损失函数给出了如何调整权重以降低loss。所以在做出错误预测的情况下，我们期望它能给我们指明前进的方向。但如果使用IoU，考虑两个预测都不与ground truth重

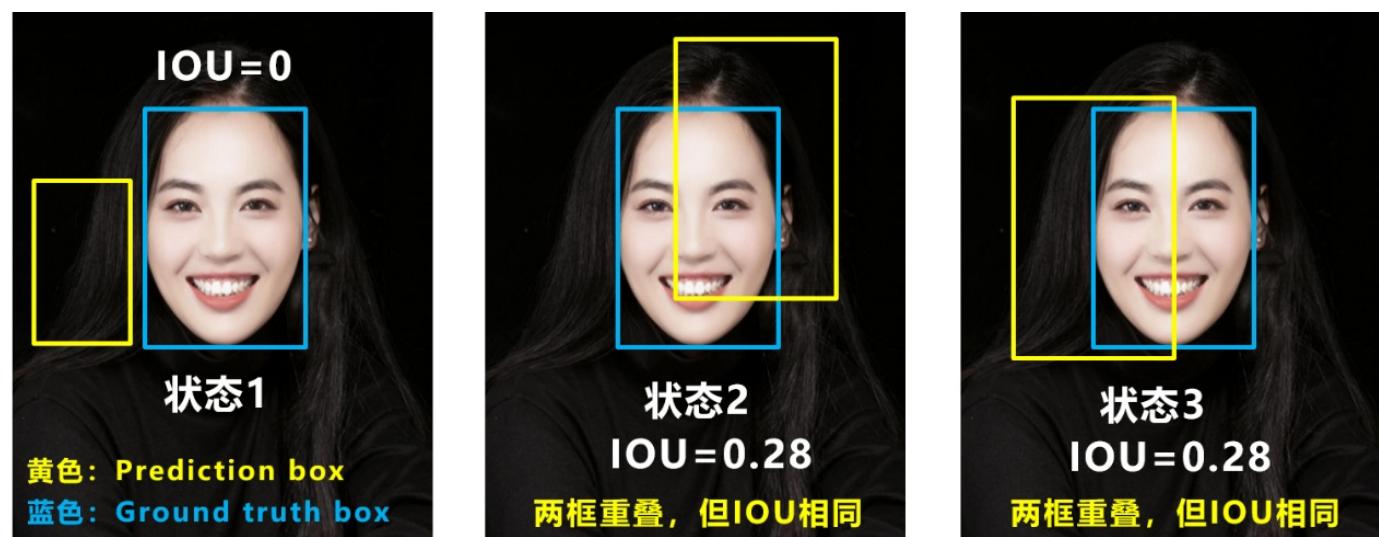
叠，那么IoU损失函数不能告诉哪一个是更好的，或者哪个更接近ground truth。这里顺带看下常用的几种loss的形式，如下：

## 1、经典IoU loss：

IoU算法是使用最广泛的算法，大部分的检测算法都是使用的这个算法。



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。



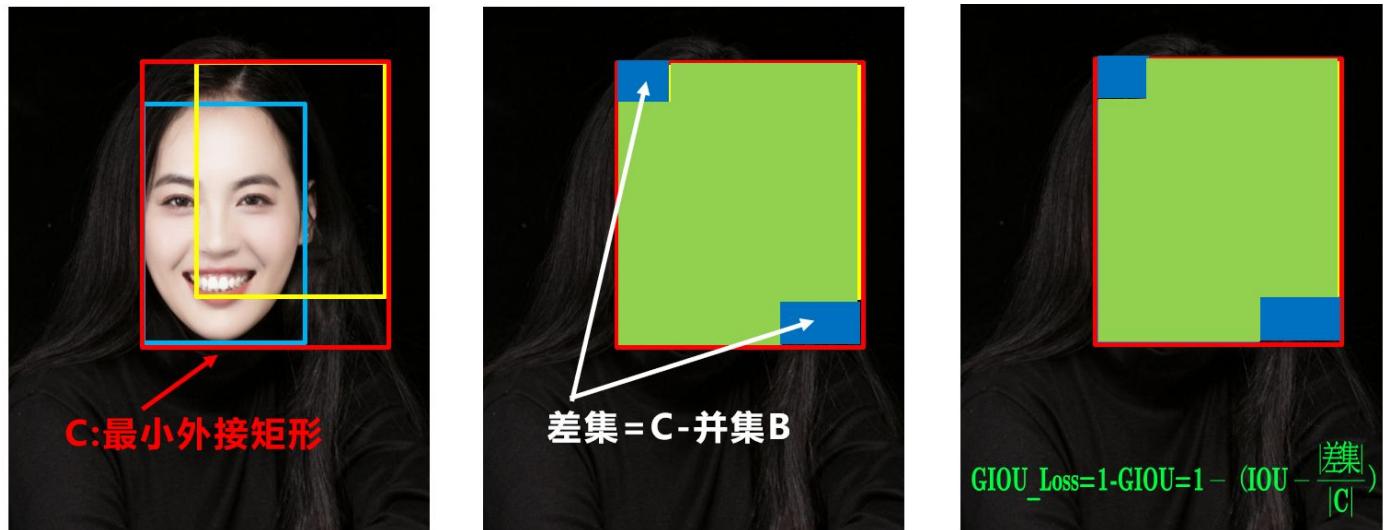
问题1：即状态1的情况，当预测框和目标框不相交时， $\text{IOU}=0$ ，无法反应两个框距离的远近，此时损失函数不可导， $\text{IOU\_Loss}$ 无法优化两个框不相交的情况。

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， $\text{IOU\_Loss}$ 无法区分两者相交情况的不同。

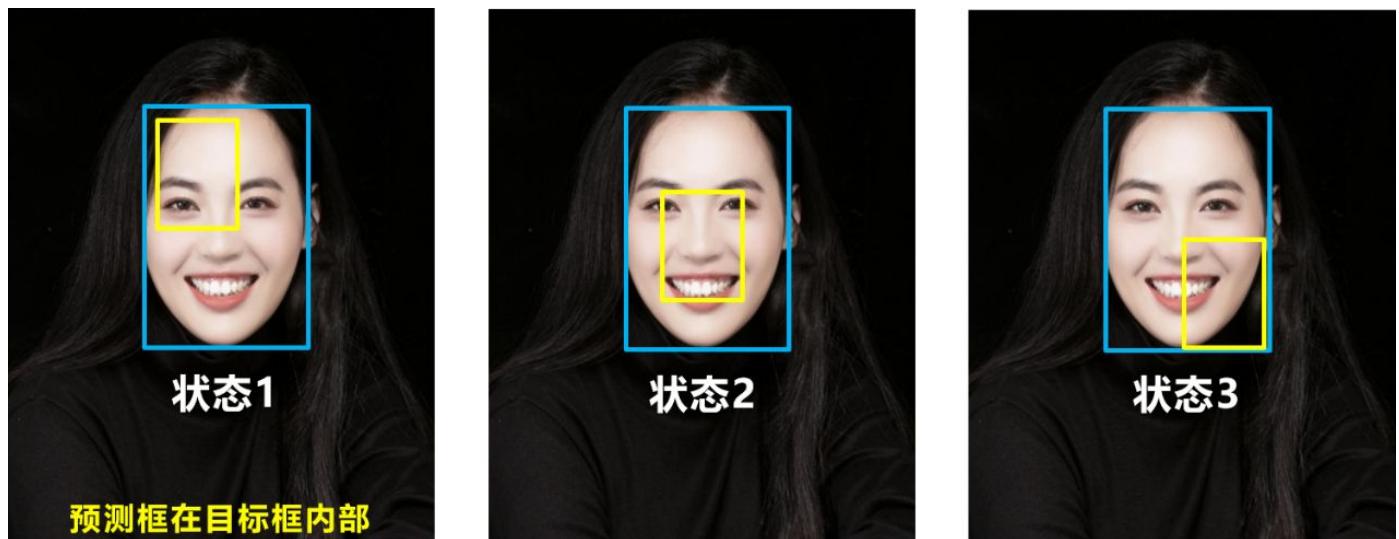
因此2019年出现了GIOU\_Loss来进行改进。

## 2、GloU: Generalized IoU

GloU考虑到，当检测框和真实框没有出现重叠的时候IoU的loss都是一样的，因此GloU就加入了C检测框（C检测框是包含了检测框和真实框的最小矩形框），这样就可以解决检测框和真实框没有重叠的问题。其中，C是指能包含predict box和Ground Truth box的最小box。



可以看到上图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**。

## 3、DIoU: Distance IoU

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

(1): 如何最小化预测框和目标框之间的归一化距离?

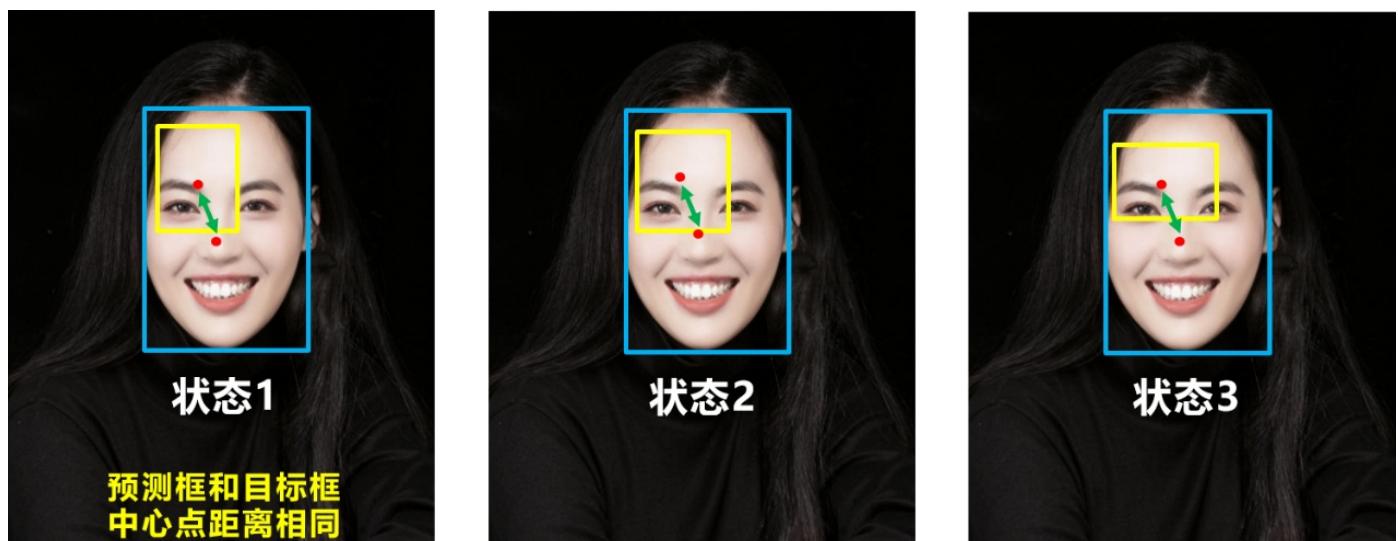
(2): 如何在预测框和目标框重叠时, 回归的更准确?

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! !

#### 4、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。

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

其中 $\nu$ 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。

---

**效果：**采用CIOU Loss损失函数，使得预测框回归的速度和精度更高一些。

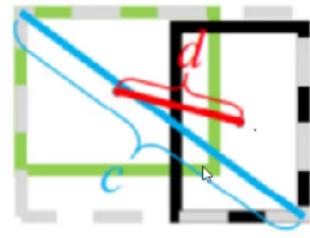
**loss优化历程：**经典IOU损失 -> GIOU损失 (Generalized IoU) -> DIOU损失 (Distance IoU) -> CIOU损失

**优缺点：**

- 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。

## ✓ DIOU损失

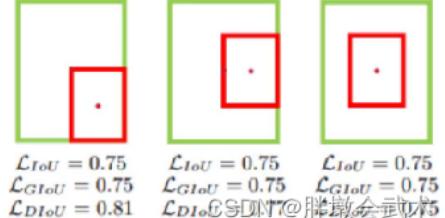
💡 公式:  $\mathcal{L}_{DIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}$



💡 其中分子计算预测框与真实框的中心点欧式距离d

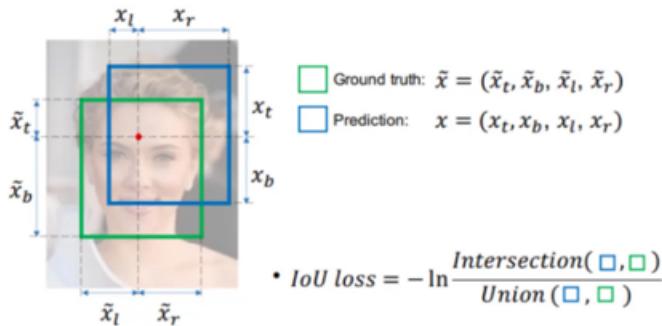
💡 分母是能覆盖预测框与真实框的最小BOX的对角线长度c

💡 直接优化距离，速度更快，并解决GIOU问题



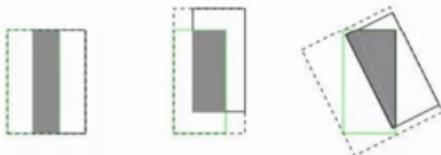
## ✓ IOU损失

💡 公式:  $IoU \text{ loss} = 1 - IoU$



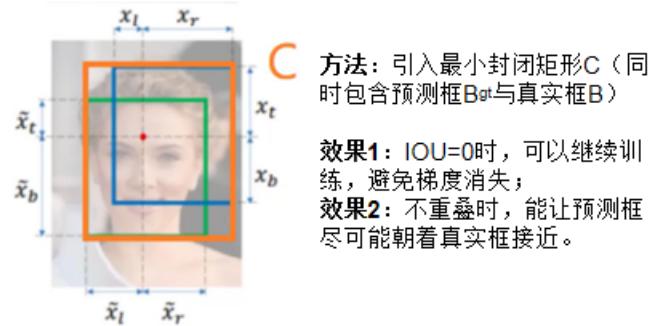
问题1: 没有相交则IoU=0无法计算；

问题2: 相同的IoU无法反应实际情况，如下图。

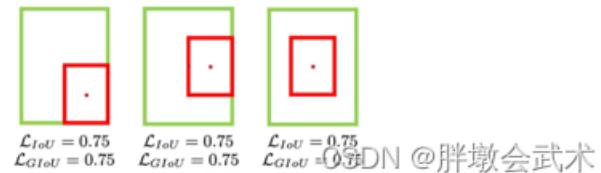


## ✓ GIOU损失

💡 公式:  $\mathcal{L}_{GIoU} = 1 - IoU + \frac{|C - B \cup B^{gt}|}{|C|}$



问题: 重叠时，无法计算。如下图。



## CmBN策略

BN就是仅仅利用当前迭代时刻信息进行norm，而CBN在计算当前时刻统计量时候会考虑前k个时刻统计量，从而实现扩大batch size操作。同时作者指出CBN操作不会引入比较大的内存开销，训练速度不会影响很多，但是训练时候会慢一些，比GN还慢。

CmBN是CBN的改进版本，其把大batch内部的4个mini batch当做一个整体，对外隔离。CBN在第t时刻，也会考虑前3个时刻的统计量进行汇合，而CmBN操作不会，不再滑动cross，其仅仅在mini batch内部进行汇合操作，保持BN一个batch更新一次可训练参数。

**BN****– assume a batch contains four mini-batches**

accumulate  $W^{(t-3)}$   
calculate  $BN^{(t-3)}$   
normalize BN

accumulate  $W^{(t-3 \sim t-2)}$   
calculate  $BN^{(t-2)}$   
normalize BN

accumulate  $W^{(t-3 \sim t-1)}$   
calculate  $BN^{(t-1)}$   
normalize BN

accumulate  $W^{(t-3 \sim t)}$   
calculate  $BN^{(t)}$   
normalize BN  
update  $W$ , ScaleShift

**CBN****– assume cross four iterations**

update  $W^{(t-3)}$   
accumulate  $BN^{(t-3 \sim t-6)}$   
normalize BN  
update ScaleShift

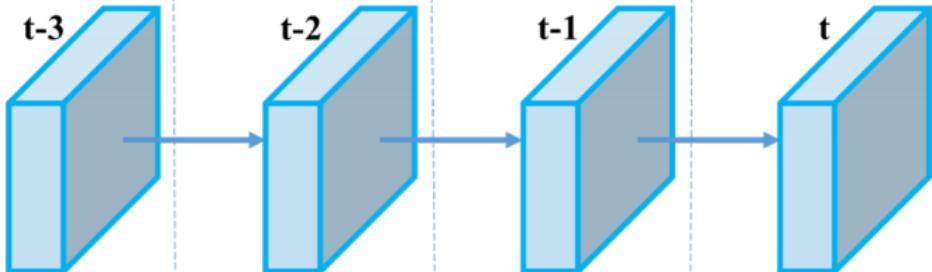
update  $W^{(t-2)}$   
accumulate  $BN^{(t-2 \sim t-5)}$   
normalize BN  
update ScaleShift

update  $W^{(t-1)}$   
accumulate  $BN^{(t-1 \sim t-4)}$   
normalize BN  
update ScaleShift

update  $W^{(t)}$   
accumulate  $BN^{(t \sim t-3)}$   
normalize BN  
update ScaleShift

Lets:

Bias, scale – ScaleShift  
Mean, variance – BN  
Weights – W

**CmBN – assume a batch contains four mini-batches**

accumulate  $W^{(t-3)}$   
accumulate  $BN^{(t-3)}$   
normalize BN

accumulate  $W^{(t-3 \sim t-2)}$   
accumulate  $BN^{(t-3 \sim t-2)}$   
normalize BN

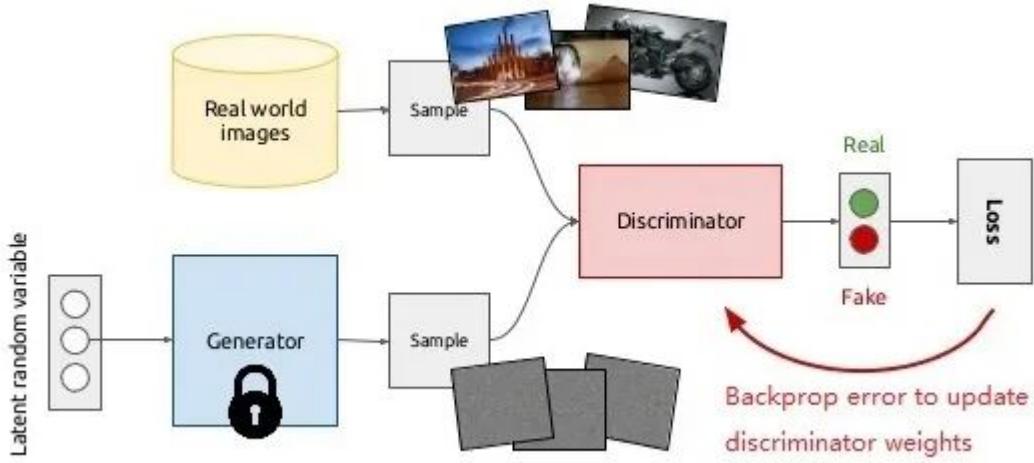
accumulate  $W^{(t-3 \sim t-1)}$   
accumulate  $BN^{(t-3 \sim t-1)}$   
normalize BN

accumulate  $W^{(t-3 \sim t)}$   
accumulate  $BN^{(t-3 \sim t)}$   
normalize BN  
update  $W$ , ScaleShift

BN: 无论每个batch被分割为多少个mini batch, 其算法就是在每个mini batch前向传播后统计当前的BN数据 (即每个神经元的期望和方差) 并进行Nomalization, BN数据与其他mini batch的数据无关。 CBN: 每次iteration中的BN数据是其之前n次数据和当前数据的和 (对非当前batch统计的数据进行了补偿再参与计算), 用该累加值对当前的batch进行Nomalization。好处在于每个batch可以设置较小的size。 CmBN: 只在每个Batch内部使用CBN的方法, 个人理解如果每个Batch被分割为一个mini batch, 则其效果与BN一致; 若分割为多个mini batch, 则与CBN类似, 只是把mini batch当作batch进行计算, 其区别在于权重更新时间点不同, 同一个batch内权重参数一样, 因此计算不需要进行补偿。

### 自对抗训练(SAT)

SAT为一种新型数据增强方式。在第一阶段, 神经网络改变原始图像而不是网络权值。通过这种方式, 神经网络对其自身进行一种对抗式的攻击, 改变原始图像, 制造图像上没有目标的假象。在第二阶段, 训练神经网络对修改后的图像进行正常的目标检测。



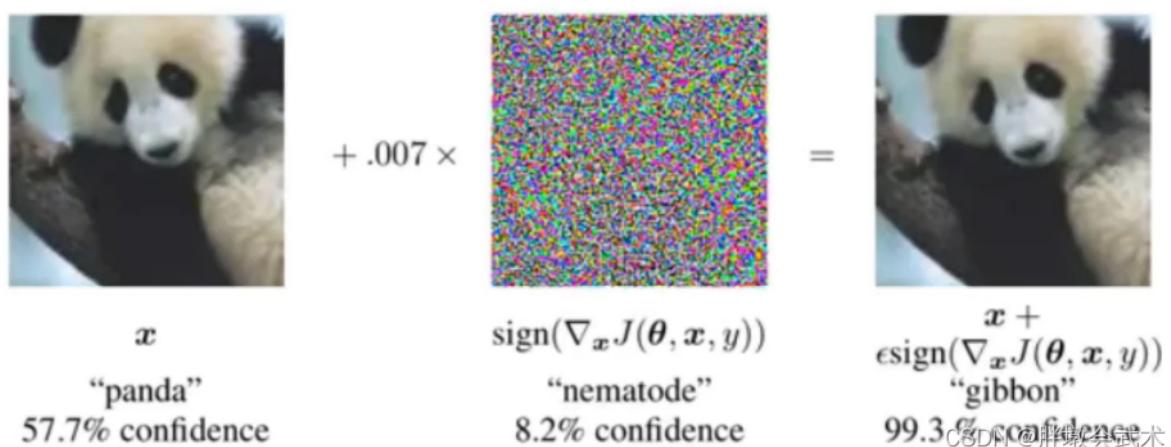
Self-Adversarial Training是在一定程度上抵抗对抗攻击的数据增强技术。CNN计算出Loss, 然后通过反向传播改变图片信息, 形成图片上没有目标的假象, 然后对修改后的图像进行正常的目标检测。需要注意的是在SAT的反向传播的过程中, 是不需要改变网络权值的。使用对抗生成可以改善学习的决策边界中的薄弱环节, 提高模型的鲁棒性。因此这种数据增强方式被越来越多的对象检测框架运用。

## 自对抗训练 (Self-Adversarial Training, SAT)

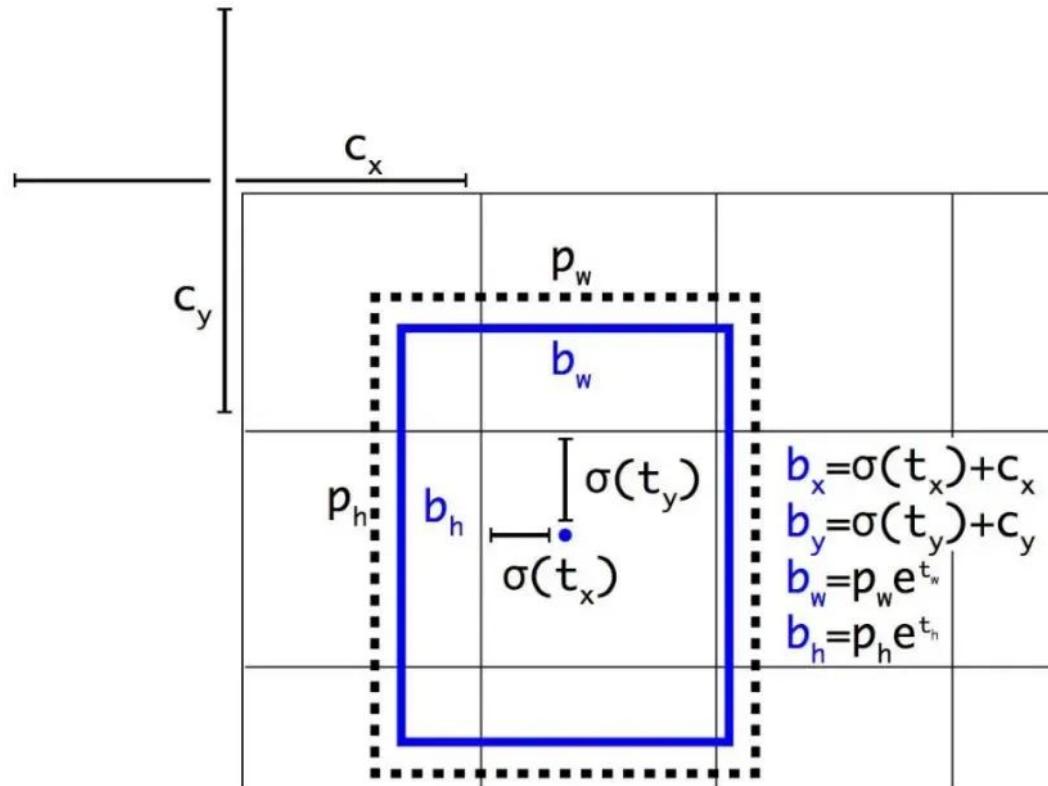
**在第一阶段**：在原始图像的基础上，添加噪音并设置权重阈值，让神经网络对自身进行对抗性攻击训练。**在第二阶段**：用正常的方法训练神经网络去检测目标。备注：详细可参考对抗攻击的快速梯度符号法（FGSM）。

### ✓ Self-adversarial-training(SAT)

通过引入噪音点来增加游戏难度



对于 $b_x = c_x$ 和 $b_x = c_x + 1$ 的情况，我们需要 $t_x$ 分别具有很大的负值和正值。但我们可以将 $\sigma$ 与一个比例因子(>1.0)相乘，从而更轻松地实现这一目标



### 余弦模拟退火

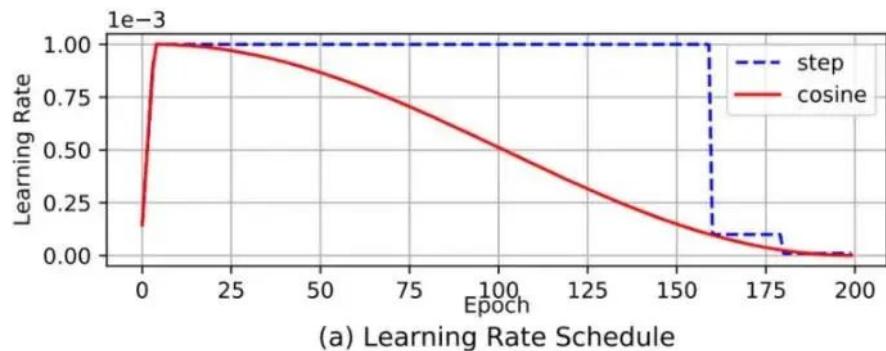
余弦调度会根据一个余弦函数来调整学习率。首先，较大的学习率会以较慢的速度减小。然后在中途时，学习的减小速度会变快，最后学习率的减小速度又会变得很慢。

$$\eta_t = \eta_{min}^i + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i}\pi)),$$

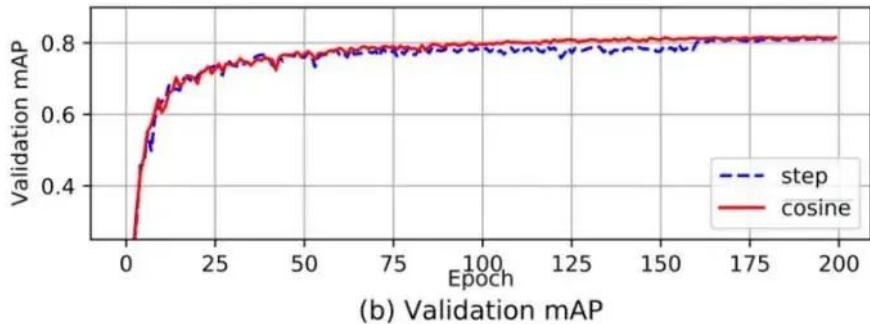
Annotations for the equation:

- $i^{th}$  run
- current learning rate
- target learning range
- # epochs have been performed since the last restart (can be in fraction, like 0.3)
- # epochs performed when SGD is restarted

这张图展示了学习率衰减的方式（下图中还应用了学习率预热）及其对 mAP的影响。可能看起来并不明显，这种新的调度方法的进展更为稳定，而不是在停滞一段时间后又取得进展。



(a) Learning Rate Schedule



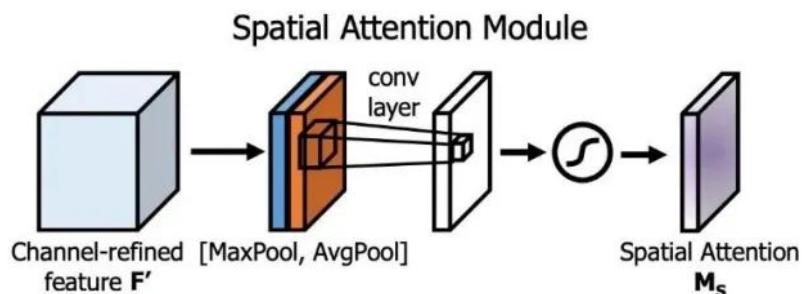
(b) Validation mAP

余弦模拟退火论文：<https://arxiv.org/pdf/1608.03983.pdf>

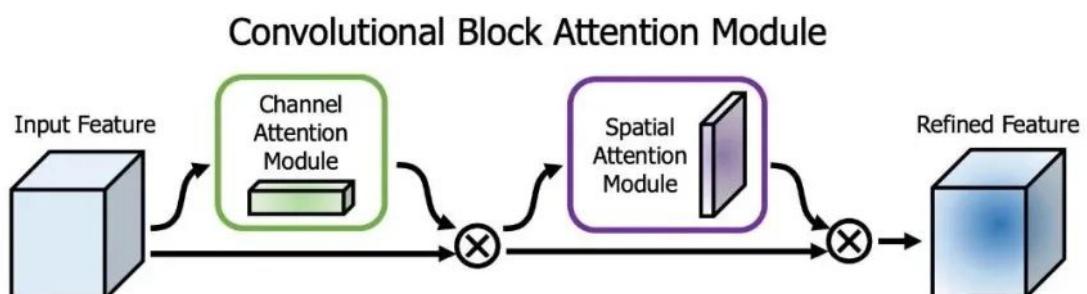
## 7. 检测头推理策略

### SAM模块

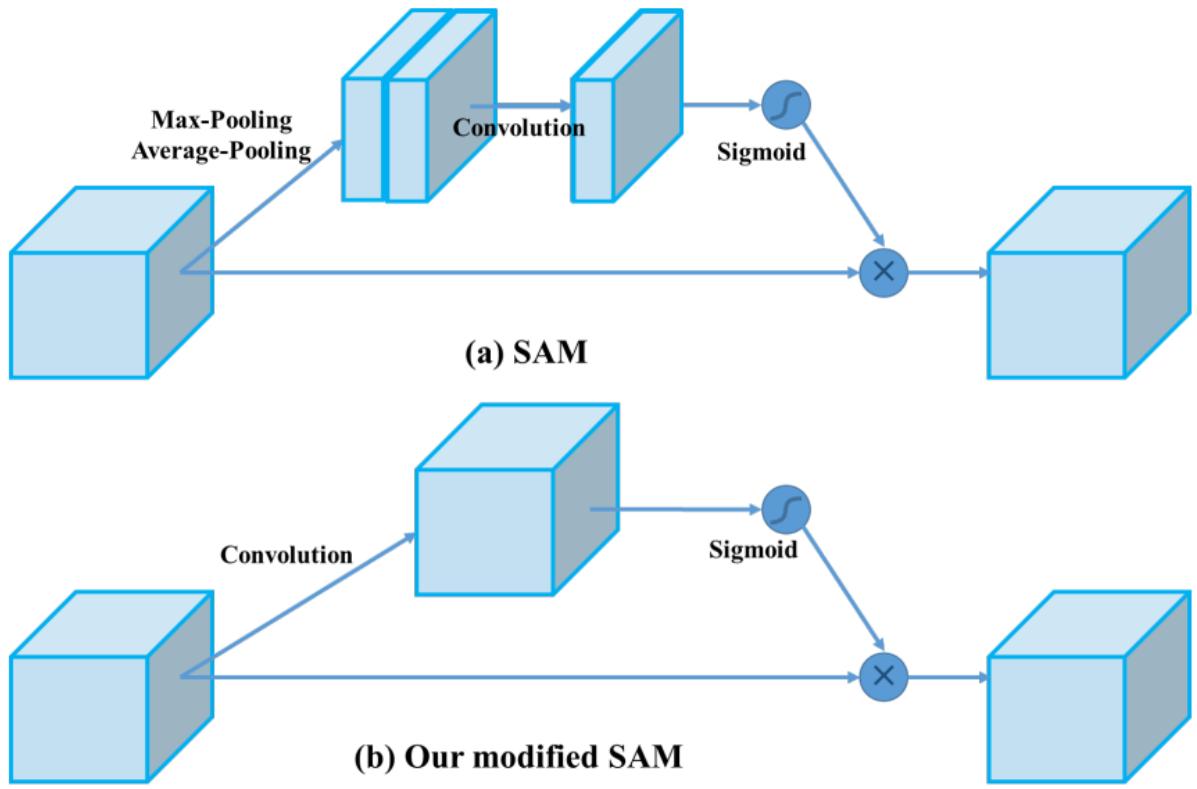
注意力机制在 DL 设计中被广泛采用。在 SAM 中，最大值池化和平均池化分别用于输入 feature map，创建两组 feature map。结果被输入到一个卷积层，接着是一个 Sigmoid 函数来创建空间注意力。



将空间注意掩模应用于输入特征，输出精细的特征图。



在 YOLOv4 中，使用修改后的 SAM 而不应用最大值池化和平均池化。



在 YOLOv4 中，FPN 概念逐渐被实现/替换为经过修改的 SPP、PAN 和 PAN。

---

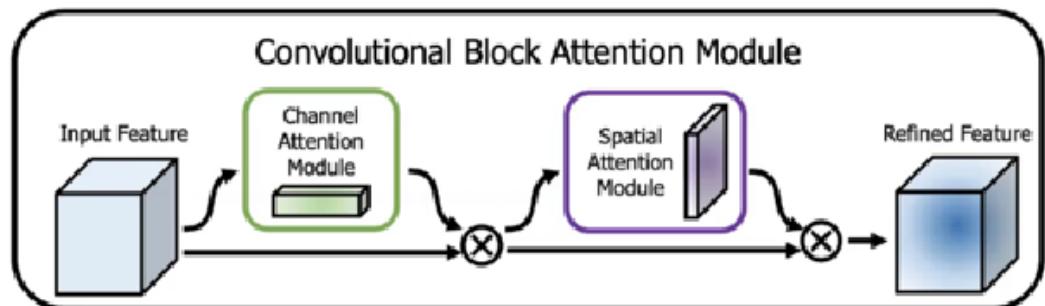
### 空间注意力机制 (Spatial Attention Module, SAM)

具体方式：yolov4采用改进的SAM方法 优化历程：CBAM (Convolutional Block AM) -> SAM (Spatial Attention Module) -> 改进的SAM 优化原因：（1）由于CBAM计算比较复杂且耗时，而yolo的出发点是速度，故只计算空间位置的注意力机制。（2）常规的SAM最大值池化层和平均池化层分别作用于输入的feature map，得到两组shape相同的feature map，再将结果输入到一个卷积层。过程过于复杂，yolo采取直接卷积进行简化。

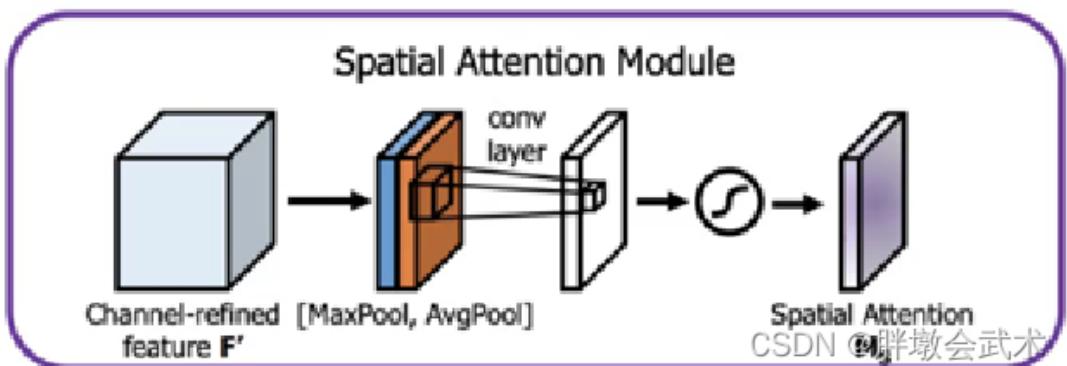
CBAM与SAM的区别：特征图注意力机制 (Channel Attention Module)：在 Channel 维度上，对每一个特征图 (channel) 加一个权重，然后通过 sigmoid 得到对应的概率值，最后乘上输入图像，相当于对输入图像的特征图进行加权，即注意力。如： $32 \times 32 \times 256$ ，对 256 个通道进行加权。空间注意力机制 (Spatial Attention Module)：在 Spatial 维度上，对每一个空间位置 (Spatial) 加一个权重，然后通过 sigmoid 得到对应的概率值，最后乘上输入图像，相当于对输入图像的所有位置特征进行加权，即注意

力。如： $32 \times 32 \times 256$ ，对任意空间位置进行加权。

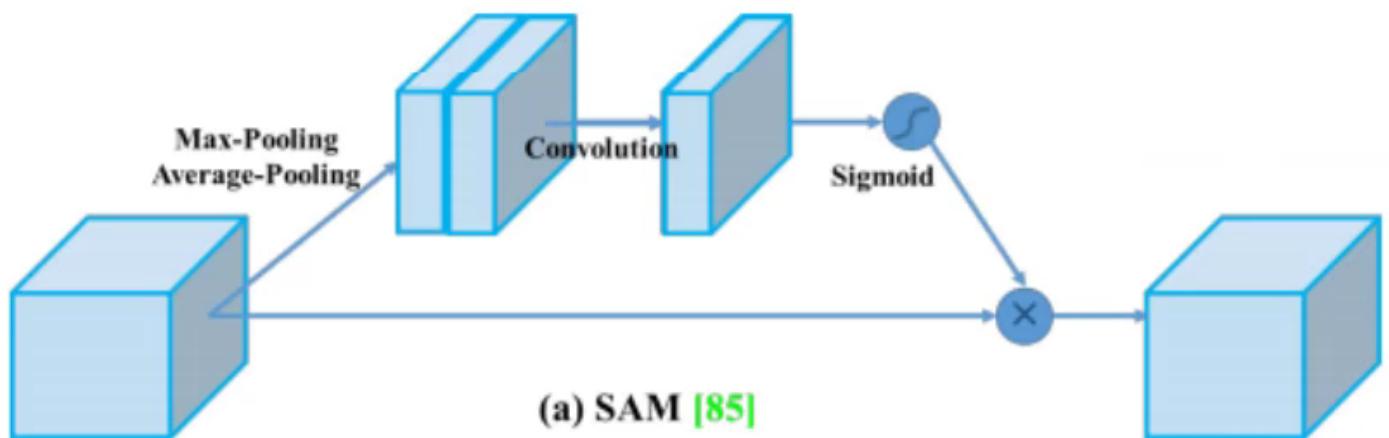
CBAM



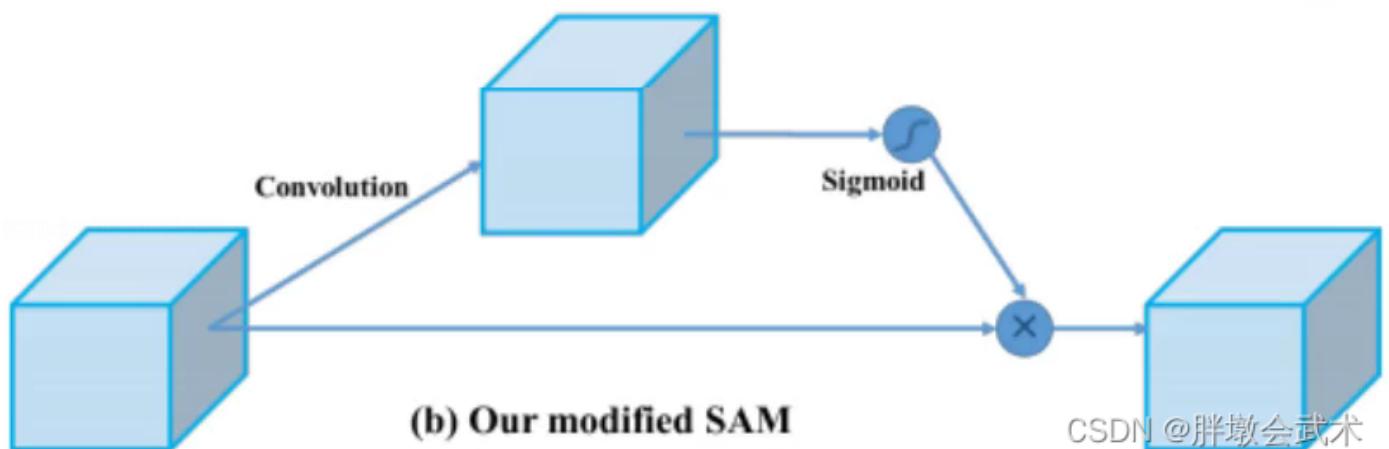
SAM



SAM与改进的SAM的区别：



(a) SAM [85]

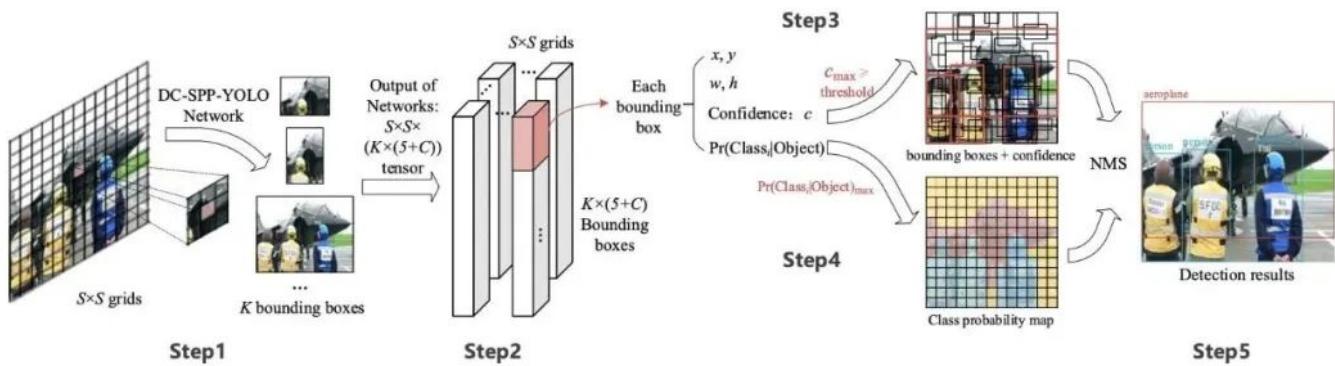


(b) Our modified SAM

CSDN @胖墩会武术

DIoU-NMS

NMS过滤掉预测相同对象的其他边界框，并保留具有最高可信度的边界框。



**DIoU**(前面讨论过的) 被用作非最大值抑制(**NMS**)的一个因素。该方法在抑制冗余框的同时，采用 **IoU**和两个边界盒中心点之间的距离。这使得它在有遮挡的情况下更加健壮。

在检测结果中，若存在多个检测框的IoU大于置信度阈值 (1) NMS非极大值抑制：只取IoU最大值对应的框。 (2) DIoU-NMS：只取公式计算得到的最大值对应的框。取最高置信度的IoU，并计算最高置信度候选框 ( $M$ ) 与其余所有框 ( $B_i$ ) 的中心点距离。优点：在有遮挡的情况下识别效果更好。

## ✓ DIOU-NMS

📝 之前使用NMS来决定是否删除一个框，现在改用DIOU-NMS

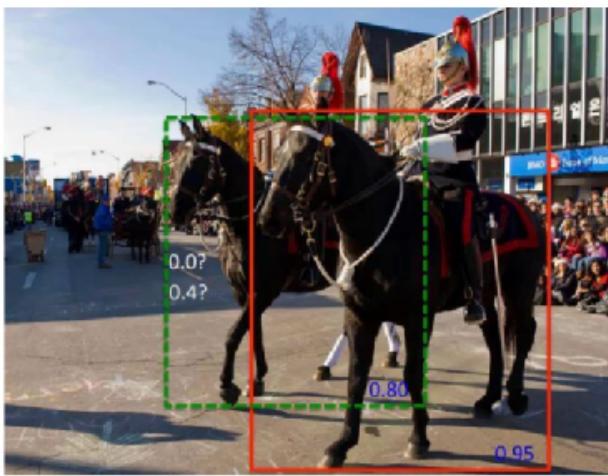
$$\text{公式: } s_i = \begin{cases} s_i, & IoU - \mathcal{R}_{DIoU}(M, B_i) < \varepsilon, \\ 0, & IoU - \mathcal{R}_{DIoU}(M, B_i) \geq \varepsilon, \end{cases} \quad \mathcal{R}_{DIoU} = \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}$$

📝 其中M表示高置信度候选框，Bi就是遍历各个框跟置信度高的重合情况

SOFT-NMS：对于不满足要求，且与最大置信度对应的检测框高度重叠的检测框，不直接删除，而采取降低置信度的方式。优点：召回率更高

## ✓ SOFT-NMS

📝 做人留一面日好相见，柔和一点的NMS，更改分数而且直接剔除



```
begin
    D ← {}
    while B ≠ empty do
        m ← argmax S
        M ← bm
        D ← D ∪ M; B ← B - M
        for bi in B do
            if iou(M, bi) ≥ Nt then
                | B ← B - bi; S ← S - si
            end
            si ← softmax(iou(M, bi))
        end
    end
    return D, S
end
```

CSDN @胖墩会武术

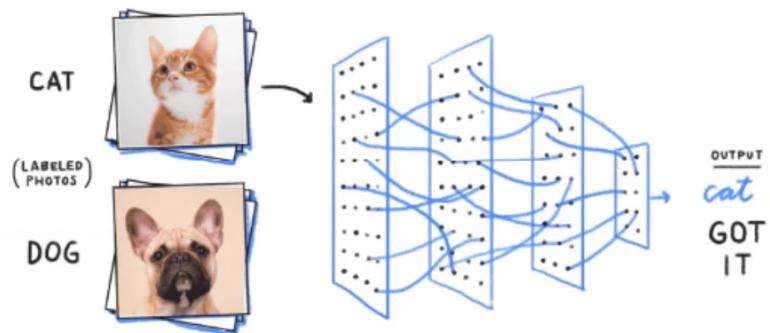
## 8. 标签平滑 (Label Smoothing)

**问题**：标签绝对化：要么0要么1。该现象将导致神经网络在训练过程中，自我良好，从而过拟合。**具体方式**：将绝对化标签进行平滑（如：[0, 0] ~ [0.05, 0.95]），即分类结果具有一定的模糊化，使得网络的抗过拟合能力增强。

## ✓ Label Smoothing

📝 神经网络最大的缺点：自觉不错（过拟合），让它别太自信

📝 例如原来标签为 (0,1) :  $[0, 1] \times (1 - 0.1) + 0.1/2 = [0.05, 0.95]$

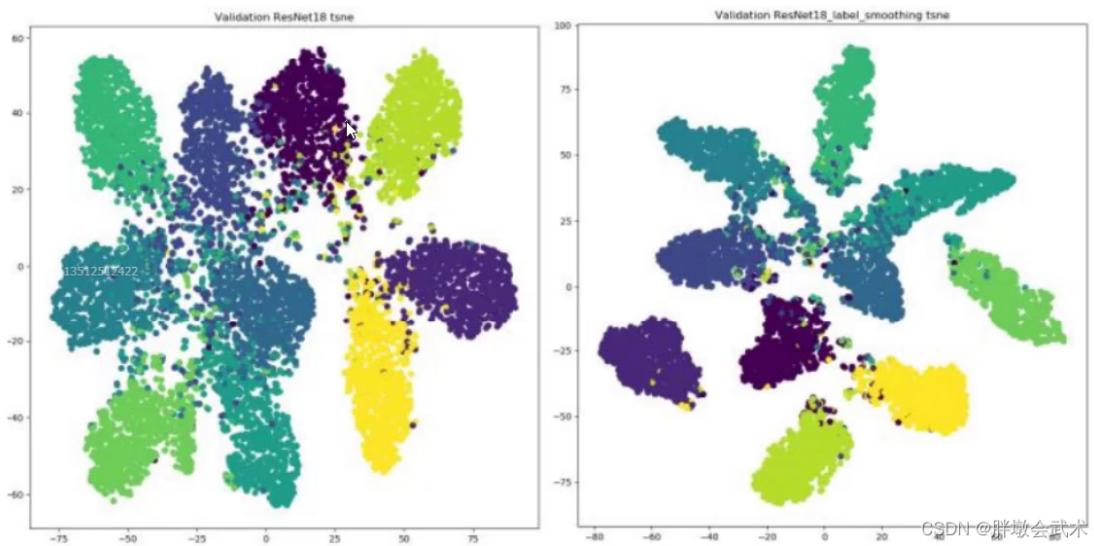


CSDN @胖墩会武术

左图（使用前）：分类结果相对不错，但各类别之间存在一定的误差；右图（使用后）：分类结果比较好，簇内距离变小，簇间距离变大。

## ✓ Label Smoothing

📎 使用之后效果分析（右图）：簇内更紧密，簇间更分离



## YOLOv5

论文下载：YOLO v5目前暂无论文

代码下载：<https://github.com/ultralytics/yoloV5>

2020年2月 YOLO之父Joseph Redmon宣布退出计算机视觉研究领域，2020年4月23日 YOLOv4发布，2020年6月10日 YOLOv5发布，但是两个版本的改进都属于多种技术堆积版本，且两个版本差异不大。

YOLOv5性能与 YOLOv4不相伯仲，同样也是现今最先进的对象检测技术，并在推理速度上是目前最强。

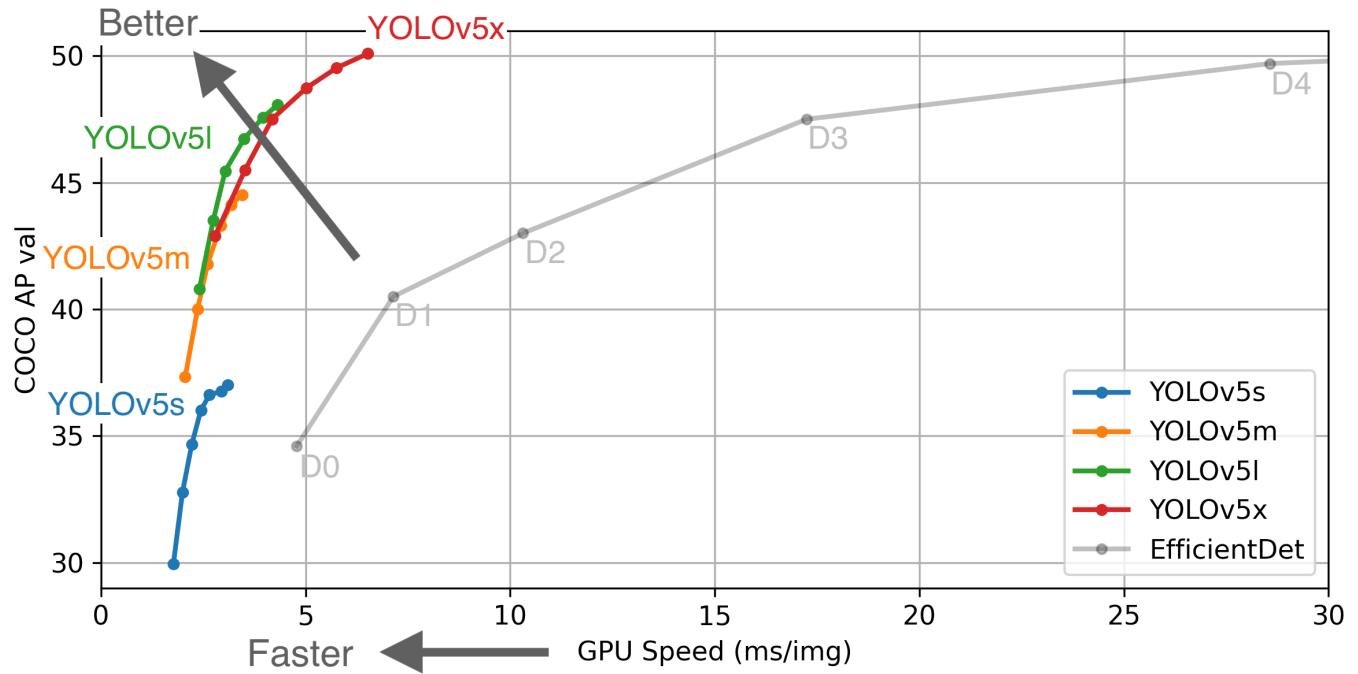
2020年2月YOLO之父Joseph Redmon宣布退出计算机视觉研究领域🚀。2020年4月23日YOLOv4发布🍀。2020年6月10日YOLOv5发布🔥。⭐ (1) 该两个版本的改进都属于多种技术堆积版本，且两个版本差异不大。⭐ (2) 一直在更新中，且更新较快（平均2~3个月一次）。⭐ (3) yolov5对应的GitHub上有详细的项目说明。但由于v5项目的训练数据集过于庞大，故可以选择自己的数据集 or 小样本数据集学习。

Roboflow：开源自动驾驶数据集。该数据集已经画好边界框；下载格式：YOLO v5 PyTorch。

---

YOLOv4出现之后不久，YOLOv5横空出世。YOLOv5在YOLOv4算法的基础上做了进一步的改进，检测性能得到进一步的提升。虽然YOLOv5算法并没有与YOLOv4算法进行性

能比较与分析，但是YOLOv5在COCO数据集上面的测试效果还是挺不错的。大家对YOLOv5算法的创新性半信半疑，有的人对其持肯定态度，有的人对其持否定态度。在我看来，YOLOv5检测算法中还是存在很多可以学习的地方，虽然这些改进思路看来比较简单或者创新点不足，但是它们确定可以提升检测算法的性能。其实工业界往往更喜欢使用这些方法，而不是利用一个超级复杂的算法来获得较高的检测精度。



yolov5是在COCO数据集上预训练的系列模型，包含5个模型：YOLOv5n、YOLOv5s、YOLOv5m、YOLOv5l、YOLOv5x。不同的变体模型使得YOLOv5能很好的在精度和速度中权衡，方便用户选择。

## Pretrained Checkpoints

Model	size (pixels)	mAP <sup>val</sup> 50-95	mAP <sup>val</sup> 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 55.8	72.7 72.7	3136 -	26.2 -	19.4 -	140.7 CSDN @胖墩会武术	209.8

## 算法简介

YOLOv5是一种单阶段目标检测算法，该算法在YOLOv4的基础上添加了一些新的改进思路，使其速度与精度都得到了极大的性能提升。主要的改进思路如下所示：

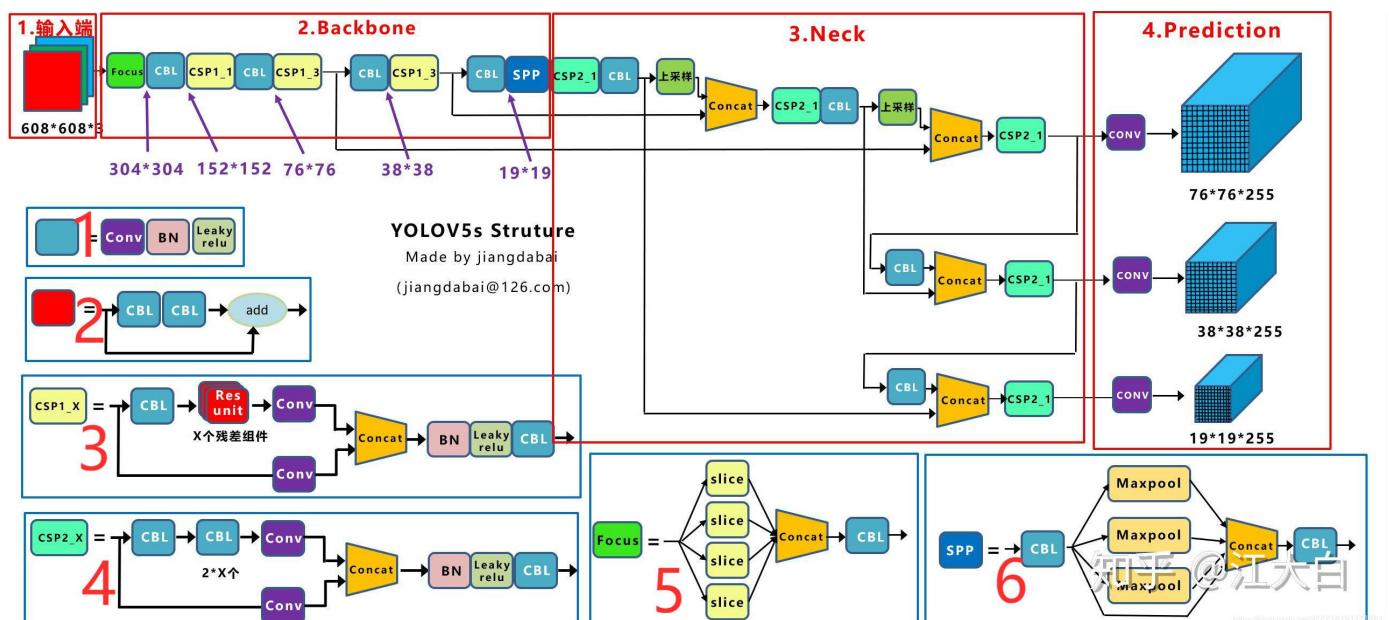
- 输入端：在模型训练阶段，提出了一些改进思路，主要包括Mosaic数据增强、自适应锚框计算、自适应图片缩放；
- 基准网络：融合其它检测算法中的一些新思路，主要包括：Focus结构与CSP结构；
- Neck网络：目标检测网络在BackBone与最后的Head输出层之间往往会插入一些层，Yolov5中添加了FPN+PAN结构；
- Head输出层：输出层的锚框机制与YOLOv4相同，主要改进的是训练时的损失函数 GIOU\_Loss，以及预测框筛选的DIOU\_nms。

## 优缺特点

- 使用Pytorch框架，对用户非常友好，能够方便地训练自己的数据集，相对于YOLO v4采用的Darknet框架，Pytorch框架更容易投入生产。
- 代码易读，整合了大量的计算机视觉技术，非常有利于学习和借鉴。

- 不仅易于配置环境，模型训练也非常快速，并且批处理推理产生实时结果。
- 能够直接对单个图像，批处理图像，视频甚至网络摄像头端口输入进行有效推理。
- 能够轻松的将Pytorch权重文件转化为安卓使用的ONNX格式，然后可以转换为OpenCV的使用格式，或者通过CoreML转化为IOS格式，直接部署到手机应用端。
- 有非常轻量级的模型大小，YOLO v5 的大小仅有 27 MB 27MB27MB， 使用Darknet 架构的 YOLO v4 有 244 MB 244MB244MB
- 最后YOLO v5高达140 F P S 140FPS140FPS的对象识别速度令人印象非常深刻，使用体验非常棒。

## 网络模型



上图展示了YOLOv5目标检测算法的整体框图。对于一个目标检测算法而言，我们通常可以将其划分为4个通用的模块，具体包括：输入端、基准网络、Neck网络与Head输出端，对应于上图中的4个红色模块。YOLOv5算法具有4个版本，具体包括：YOLOv5s、YOLOv5m、YOLOv5l、YOLOv5x四种，本文重点讲解YOLOv5s，其它的版本都在该版本的基础上对网络进行加深与加宽。

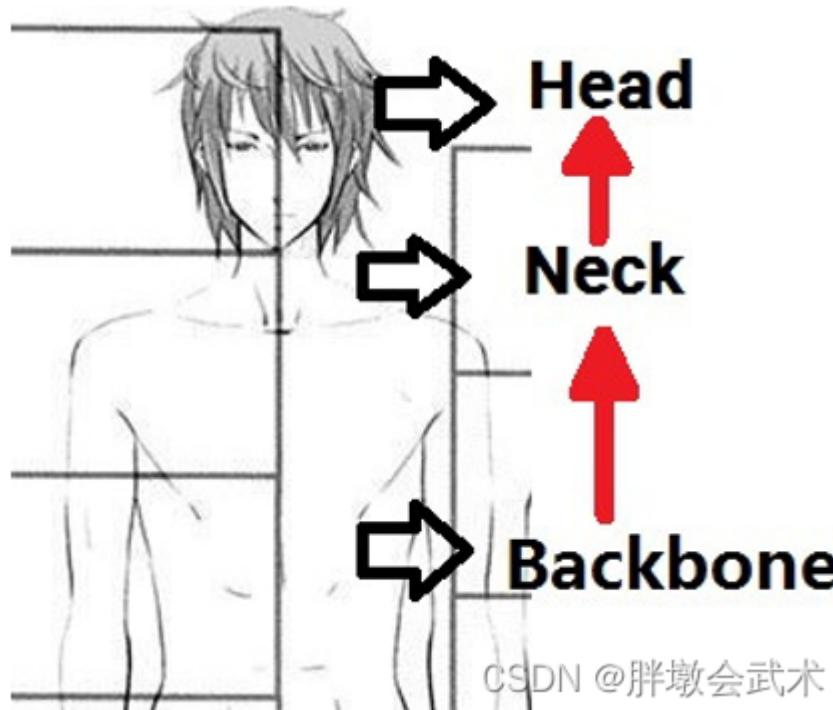
- **输入端**-输入端表示输入的图片。该网络的输入图像大小为608\*608，该阶段通常包含一个图像预处理阶段，即将输入图像缩放到网络的输入大小，并进行归一化等操作。在网络训练阶段，YOLOv5使用Mosaic数据增强操作提升模型的训练速度和网络的精度；并提出了一种自适应锚框计算与自适应图片缩放方法。
- **基准网络**-基准网络通常是一些性能优异的分类器种的网络，该模块用来提取一些通用的特征表示。YOLOv5中不仅使用了CSPDarknet53结构，而且使用了Focus结构作为基准网络。
- **Neck网络**-Neck网络通常位于基准网络和头网络的中间位置，利用它可以进一步提升特征的多样性及鲁棒性。虽然YOLOv5同样用到了SPP模块、FPN+PAN模块，但是实现的细节有些不同。

- Head输出端-Head用来完成目标检测结果的输出。针对不同的检测算法，输出端的分支个数不尽相同，通常包含一个分类分支和一个回归分支。YOLOv4利用GIOU\_Loss来代替Smooth L1 Loss函数，从而进一步提升算法的检测精度。

#### YOLOv5基础组件

- CBL-CBL模块由Conv+BN+Leaky\_relu激活函数组成，如上图中的模块1所示。
- Res unit-借鉴ResNet网络中的残差结构，用来构建深层网络，CBM是残差模块中的子模块，如上图中的模块2所示。
- CSP1\_X-借鉴CSPNet网络结构，该模块由CBL模块、Res unit模块以及卷积层、Concat组成而成，如上图中的模块3所示。
- CSP2\_X-借鉴CSPNet网络结构，该模块由卷积层和X个Res unit模块Concat组成而成，如上图中的模块4所示。
- Focus-如上图中的模块5所示，Focus结构首先将多个slice结果Concat起来，然后将其送入CBL模块中。
- SPP-采用 $1 \times 1$ 、 $5 \times 5$ 、 $9 \times 9$ 和 $13 \times 13$ 的最大池化方式，进行多尺度特征融合，如上图中的模块6所示。

#### Backbone（特征提取模块）



- (1) Backbone（骨干网络）：用于提取图像特征的网络。\*常用不是我们自己设计的网络，而是通用网络模型resnet、VGG等。使用方式：将通用模型作为backbone时，直接加载预训练模型的参数，再拼接上我们自己的网络。网络训练方法参考迁移学习的微调算法，即对预训练模型进行微调，使其更适合我们自己的任务。

- (2) Neck (脖子) : 在backbone和head之间, 是为了更好的利用backbone提取的特征。
- (3) Bottleneck (瓶颈) : 指输出维度比输入维度小很多, 就像由身体到脖子, 变细了。经常设置的参数 bottle\_num=256, 指的是网络输出的数据的维度是256 , 可是输入进来的可能是1024维度的。
- (4) Head (头部) : head是获取网络输出内容的网络, 利用之前提取的特征, head利用这些特征, 做出预测。

Backbone结构主要分成三类: CNNs结构 (非轻量级、轻量级) 、Transformer结构、CNNs+Transformer结构。

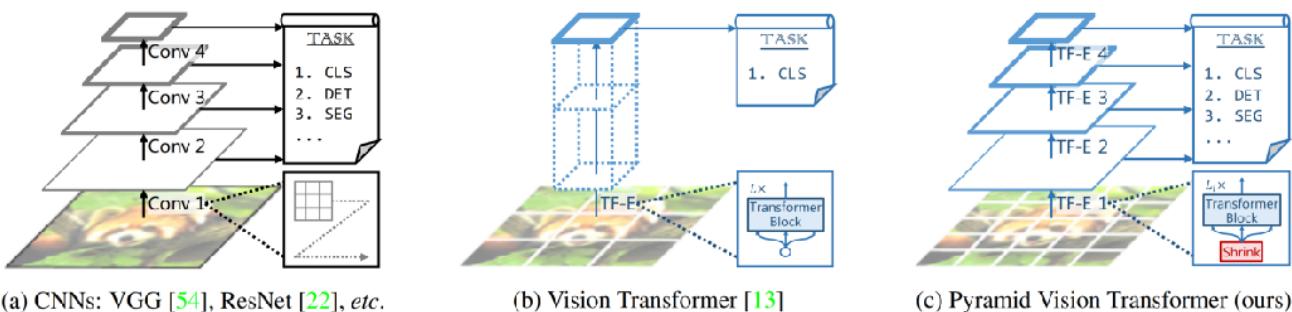


Figure 1: **Comparisons of different architectures**, where “Conv” and “TF-E” stand for “convolution” and “Transformer encoder”, respectively. (a) Many CNN backbones use a pyramid structure for dense prediction tasks such as object detection (DET), instance and semantic segmentation (SEG). (b) The recently proposed Vision Transformer (ViT) [13] is a “columnar” structure specifically designed for image classification (CLS). (c) By incorporating the pyramid structure from CNNs, we present the Pyramid Vision Transformer (PVT), which can be used as a versatile backbone for many computer vision tasks, broadening the scope and impact of ViT. Moreover, our experiments also show that PVT can easily be combined with DETR [6] to build an end-to-end object detection system without convolutions.

CSDN @胖墩会武术

## 深度学习框架-Backbone汇总 (超详细讲解)

### 一、普通 (非轻量化) CNNs结构Backbone

- LeNet5: (1998)
- AlexNet: (2012)
- VGG: (2014)
- GoogLeNet (InceptionNet) 系列: Inception-v1 (GoogleNet) : (2015)、Inception-v2 (2015, BN-inception) 、Inception-v3 (2015)、Inception-v4: (2017)、Inception-resnet-v2: (2017)
- Resnet: (2016)
- ResNet变种: ResNeXt (2016) 、ResNeSt (2020) 、Res2Net (2019) 、DenseNet (2017)
- DPNet: (2017)
- NasNet: (2018)
- SENet及其变体SKNet: SENet (2017) 、SKNet (2019)
- EfficientNet 系列: EfficientNet-V1(2019)、EfficientNet-V2(2021)

- Darknet系列: Darknet-19 (2016, YOLO v2 的 backbone) 、Darknet-53 (2018, YOLOv3的 backbone)
- DLA (2018, Deep Layer Aggregation)

## 二、轻量化CNNs结构Backbone

- SqueezeNet: (2016)
- MobileNet-v1: (2017)
- Xception: (2017, 极致的 Inception)
- MobileNet V2: (2018)
- ShuffleNet-v1: (2018)
- ShuffleNet-v2: (2018)
- MnasNet: (2019)
- MobileNet V3 (2019)
- CondenseNet (2017)
- ESPNet系列: ESPNet (2018) 、ESPNetv2 (2018)
- ChannelNets
- PeleeNet
- IGC系列: IGCV1、IGCV2、IGCV3
- FBNet系列: FBNet、FBNetV2、FBNetV3
- GhostNet
- WeightNet
- MicroNet

## 三、ViT (Vision Transformer ) 结构Backbone

- ViT-H/14 和 ViT-L/16 (2020) (Vision Transformer, ViT)
- Swin Transformer (2021)
- PVT (2021, Pyramid Vision Transformer)
- MPViT (CVPR 2022, Multi-path Vision Transformer, 多路径 Vision Transformer)
- EdgeViTs (CVPR 2022, 轻量级视觉Transformer)

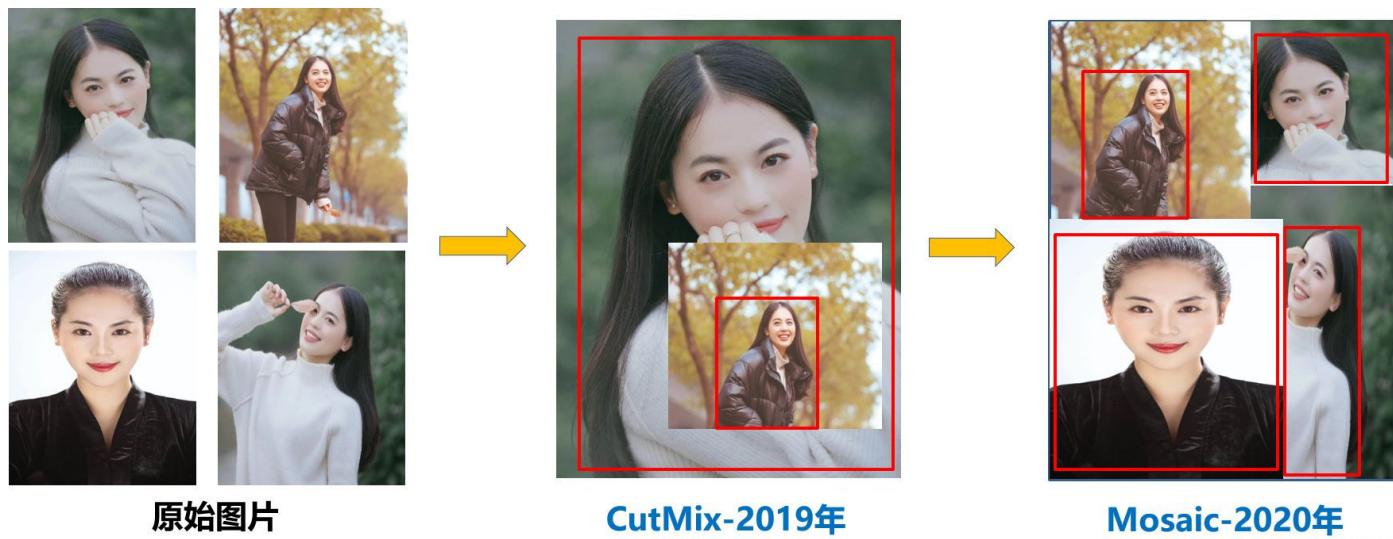
## 四、CNNs+Transformer/Attention结构Backbone

- CoAtNet (#2 2021)
- BoTNet (#1 2021)

EfficientNet

[EfficientNet网络详解](#)

Mosaic数据增强-YOLOv5中在训练模型阶段仍然使用了Mosaic数据增强方法，该算法是在CutMix数据增强方法的基础上改进而来的。CutMix仅仅利用了两张图片进行拼接，而Mosaic数据增强方法则采用了4张图片，并且按照随机缩放、随机裁剪和随机排布的方式进行拼接而成，具体的效果如下图所示。这种增强方法可以将几张图片组合成一张，这样不仅可以丰富数据集的同时极大的提升网络的训练速度，而且可以降低模型的内存需求。

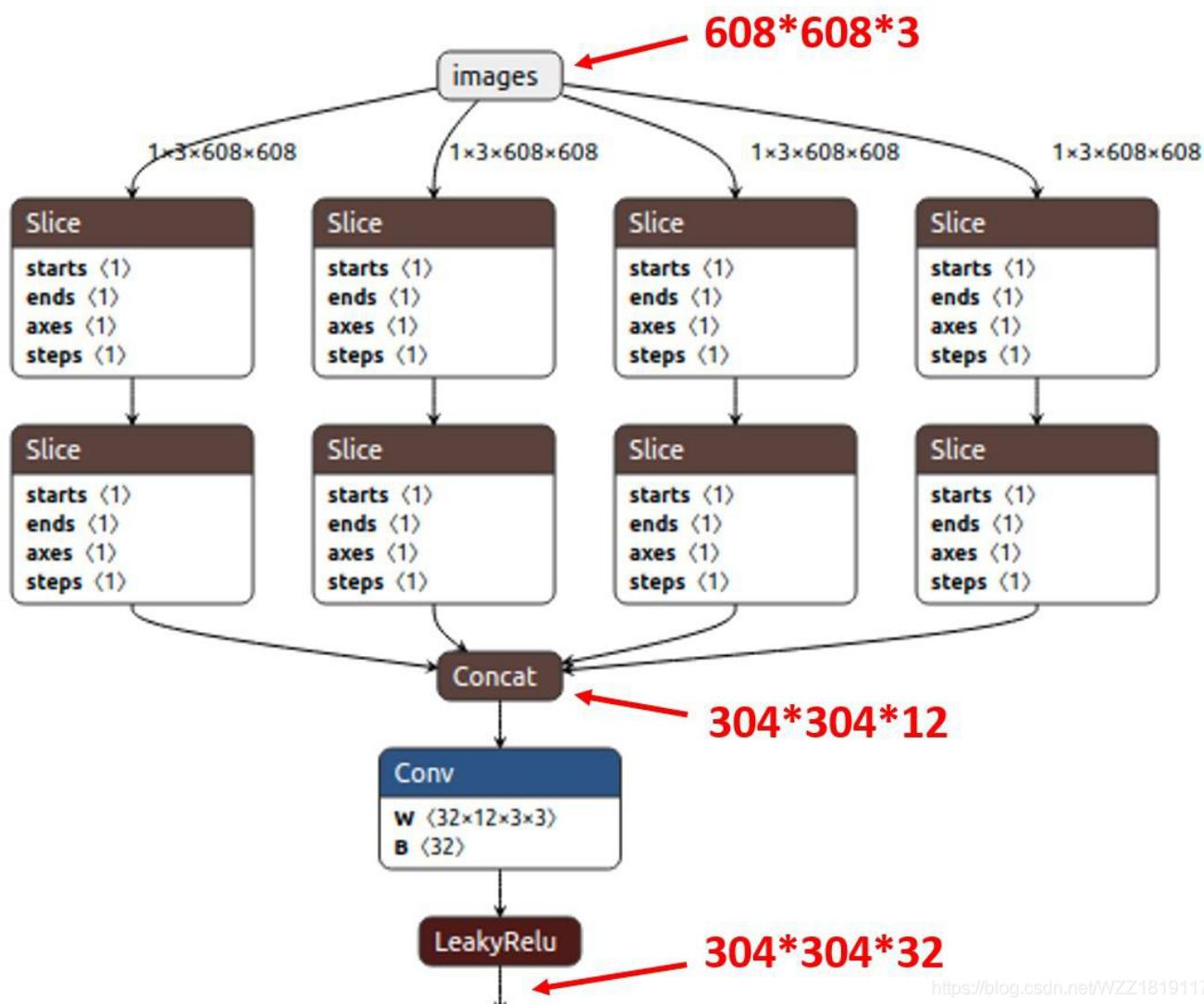


- **自适应锚框计算**-在YOLOv5系列算法中，针对不同的数据集，都需要设定特定长宽的锚点框。在网络训练阶段，模型在初始锚点框的基础上输出对应的预测框，计算其与GT框之间的差距，并执行反向更新操作，从而更新整个网络的参数，因此设定初始锚点框也是比较关键的一环。在YOLOv3和YOLOv4检测算法中，训练不同的数据集时，都是通过单独的程序运行来获得初始锚点框。YOLOv5中将此功能嵌入到代码中，每次训练时，根据数据集的名称自适应的计算出最佳的锚点框，用户可以根据自己的需求将功能关闭或者打开，具体的指令为`parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')`，如果需要打开，只需要在训练代码时增加`--noautoanch` or 选项即可。
- **自适应图片缩放**-针对不同的目标检测算法而言，我们通常需要执行图片缩放操作，即将原始的输入图片缩放到一个固定的尺寸，再将其送入检测网络中。YOLO系列算法中常用的尺寸包括416\*416, 608 \*608等尺寸。原始的缩放方法存在着一些问题，由于在实际的使用中的很多图片的长宽比不同，因此缩放填充之后，两端的黑边大小都不相同，然而如果填充的过多，则会存在大量的信息冗余，从而影响整个算法的推理速度。为了进一步提升YOLOv5算法的推理速度，该算法提出一种方法能够自适应的添加最少的黑边到缩放之后的图片中。

## 基准网络细节详解

**Focus结构**-该结构的主要思想是通过slice操作来对输入图片进行裁剪。如下图所示，原始输入图片大小为608\*608，经过Slice与Concat操作之后输出一个304\*304\*12的特征映射。

射；接着经过一个通道个数为32的Conv层（该通道个数仅仅针对的是YOLOv5s结构，其它结构会有相应的变化），输出一个 $304 \times 304 \times 32$ 大小的特征映射。

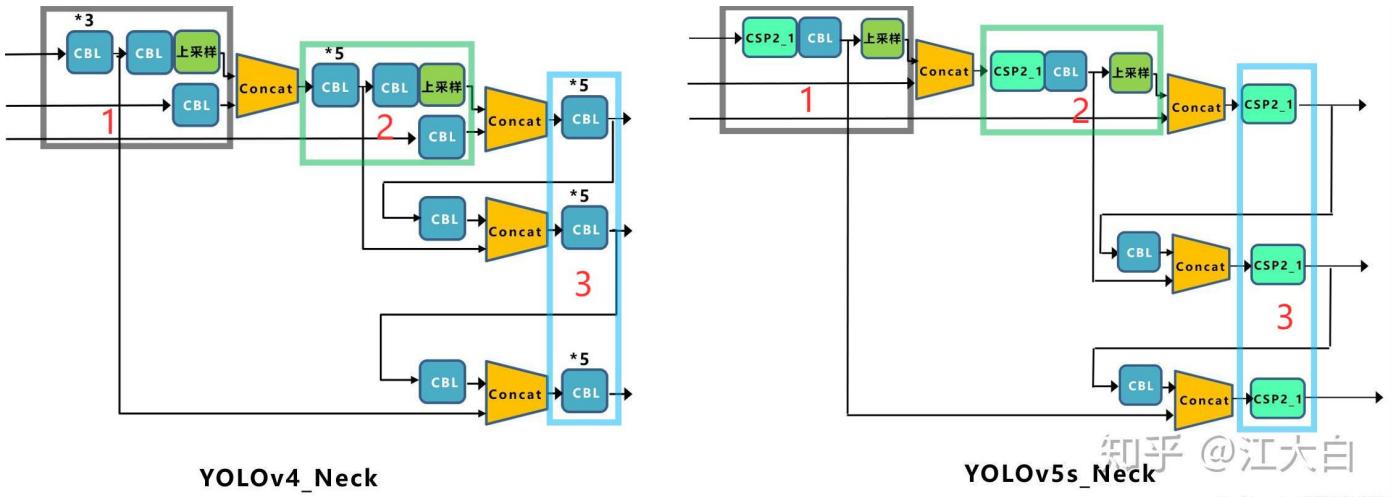


<https://blog.csdn.net/WZZ18191171661>

CSP结构-YOLOv4网络结构中，借鉴了CSPNet的设计思路，仅仅在主干网络中设计了CSP结构。而YOLOv5中设计了两种CSP结构，以YOLOv5s网络为例，CSP1\_X结构应用于Backbone主干网络中，另一种CSP2\_X结构则应用于Neck网络中。CSP1\_X与CSP2\_X模块的实现细节如3.1所示。

### Neck网络细节详解

FPN+PAN-YOLOv5的Neck网络仍然使用了FPN+PAN结构，但是在它的基础上做了一些改进操作，YOLOv4的Neck结构中，采用的都是普通的卷积操作。而YOLOv5的Neck网络中，采用借鉴CSPnet设计的CSP2结构，从而加强网络特征融合能力。下图展示了YOLOv4与YOLOv5的Neck网络的具体细节，通过比较我们可以发现：（1）灰色区域表示第1个不同点，YOLOv5不仅利用CSP2\_1结构代替部分CBL模块，而且去掉了下方的CBL模块；（2）绿色区域表示第2个不同点，YOLOv5不仅将Concat操作之后的CBL模块更换为CSP2\_1模块，而且更换了另外一个CBL模块的位置；（3）蓝色区域表示第3个不同点，YOLOv5中将原始的CBL模块更换为CSP2\_1模块。



## 改进之处

深入浅出Yolo系列之Yolov5核心基础知识完整讲解

# YOLOX: Exceeding YOLO Series in 2021

论文下载：<https://arxiv.org/abs/2107.08430>

代码下载：<https://github.com/Megvii-BaseDetection/YOLOX>

## 算法简介

YOLO X是旷视科技在目标检测方面的最新技术总结，同时也是CVPR2021自动驾驶竞赛冠军方案的技术总结。其将近两年来目标检测领域的各个角度的优秀进展与YOLO进行了巧妙地集成组合(比如解耦头、数据增广、标签分配、Anchor-free机制等)得到了YOLO X，性能取得了大幅地提升，同时仍保持了YOLO系列一贯地高效推理。

## 网络改进

作者将YOLO检测器调整为了Anchor-Free形式并集成了其他先进检测技术(比如decoupled head、label assignment SimOTA)取得了SOTA性能，比如：

- 对于YOLO-Nano，所提方法仅需0.91M参数+1.08G FLOPs取得了25.3%AP指标，以1.8%超越了NanoDet；
- 对于YOLO v3，所提方法将指标提升到了47.3%，以3%超越了当前最佳；
- 具有与YOLOv4-CSP、YOLOv5-L相当的参数量，YOLOX-L取得了50.0%AP指标同事具有68.9fps推理速度(Tesla V100)，指标超过YOLOv5-L1.8%；YOLOX-L凭借单模型取得了Streaming Perception(Workshop on Autonomous Driving at CVPR 2021)竞

赛冠军。值得一提的是，作者使用的baseline是 YOLO v3 + DarkNet53（所谓的 YOLO v3-SSP）

- Decoupled Head 解耦头 作者研究发现，检测头耦合会影响模型性能。采用解耦头替换YOLO的检测头可以显著改善模型收敛速度。因此，作者将YOLO的检测头替换为轻量解耦头：它包含一个 $1 \times 1$ 卷积进行通道降维，后接两个并行分支(均为 $3 \times 3$ 卷积)。注：轻量头可以带来1.1ms的推理耗时。
- Mosaic + Mixup 数据增强
  - Mosaic数据增强
  - Mixup数据增强：对图像进行混类增强的算法，它将不同类之间的图像进行混合，从而扩充训练数据集
- Anchor-free
  - YOLOv4、YOLOv5均采用了 YOLOv3原始的anchor设置。然而anchor机制存在诸多问题：
    - (1) 为获得最优检测性能，需要在训练之前进行聚类分析以确定最佳 anchor集合，这些anchor集合存在数据相关性，泛化性能较差；
    - (2) anchor机制提升了检测头的复杂度。
  - Anchor-free检测器在过去两年得到了长足发展并取得了与anchor检测器相当的性能。
  - 将YOLO转换为anchor-free形式非常简单，我们 **将每个位置的预测从3下降为1并直接预测四个值**：即两个offset以及高宽。这种改进可以降低检测器的参数量于GFLOPs进而取得更快更优的性能： **42.9%AP** 。
- Multi positives
  - 为确保与YOLOv3的一致性，前述anchor-free版本仅仅对每个目标赋予一个正样本，而忽视了其他高质量预测。参考FCOS，我们简单的赋予中心 $3 \times 3$ 区域为正样本。此时 **模型性能提升到45.0%，超过了当前最佳U版YOLOv3的44.3%** 。
- SimOTA
  - OTA从全局角度分析了标签分配并将其转化为最优运输问题取得了SOTA性能。
  - 然而，作者发现：最优运输问题优化会带来25 % 25%25%的额外训练耗时。因此，我们将其简化为动态top-k top-k策略以得到一个近似解(SimOTA)。SimOTA不仅可以降低训练时间，同时可以避免额外的超参问题。

SimOTA的引入可以将模型的性能从45.0%提升到47.3%，大幅超越U版YOLOv6的44.3%。

- End-to-End YOLO

- YOLO X参考PSS添加了两个额外的卷积层，one-to-one标签分配以及stop gradient。这些改进使得目标检测器进化成了端到端形式，但会轻微降低性能与推理速度。
- 因此，我们将该改进作为可选模块，并未包含在最终模型中。

## YOLOv6

### 算法简介

#### [手把手教你运行YOLOv6（超详细）](#)

yolov6与v7相差不到十天，区别不大。

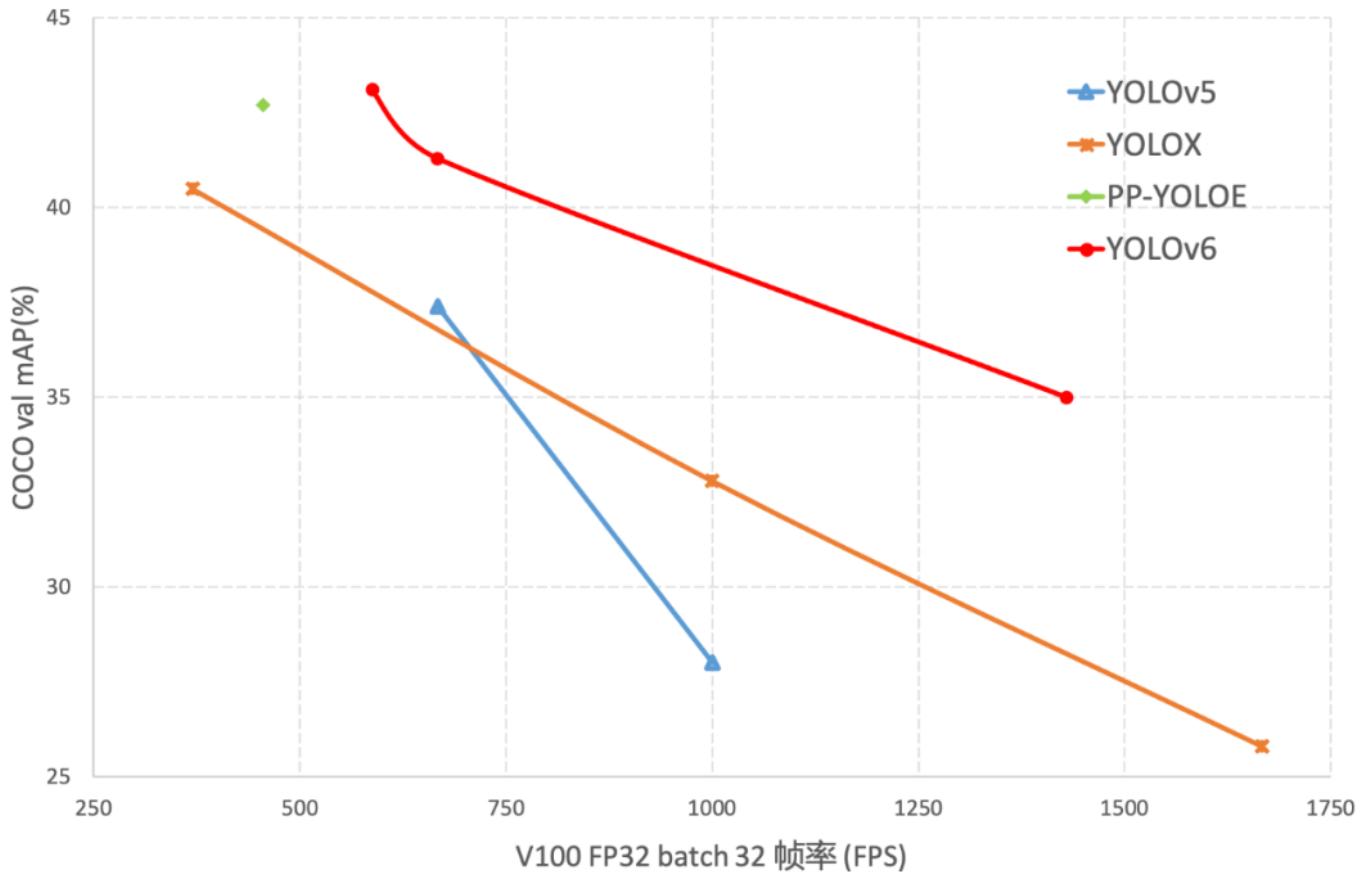
---

YOLOv6 是美团视觉智能部研发的一款目标检测框架，致力于工业应用。本框架同时专注于检测的精度和推理效率，在工业界常用的尺寸模型中：YOLOv6-nano 在 COCO 上精度可达 35.0% AP，在 T4 上推理速度可达 1242 FPS；YOLOv6-s 在 COCO 上精度可达 43.1% AP，在 T4 上推理速度可达 520 FPS。在部署方面，YOLOv6 支持 GPU (TensorRT)、CPU (OPENVINO)、ARM (MNN、TNN、NCNN) 等不同平台的部署，极大地简化工程部署时的适配工作。

### 精度与速度远超 YOLOv5 和 YOLOX 的新框架

目标检测作为计算机视觉领域的一项基础性技术，在工业界得到了广泛的应用，其中 YOLO 系列算法因其较好的综合性能，逐渐成为大多数工业应用时的首选框架。至今，业界已衍生出许多 YOLO 检测框架，其中以 YOLOv5、YOLOX 和 PP-YOLOE 最具代表性，但在实际使用中，我们发现上述框架在速度和精度方面仍有很大的提升的空间。基于此，我们通过研究并借鉴了业界已有的先进技术，开发了一套新的目标检测框架——YOLOv6。该框架支持模型训练、推理及多平台部署等全链条的工业应用需求，并在网络结构、训练策略等算法层面进行了多项改进和优化，在 COCO 数据集上，YOLOv6 在精度和速度方面均超越其他同体量算法，相关结果如下图所示：

## 模型性能对比 (不同模型系列)



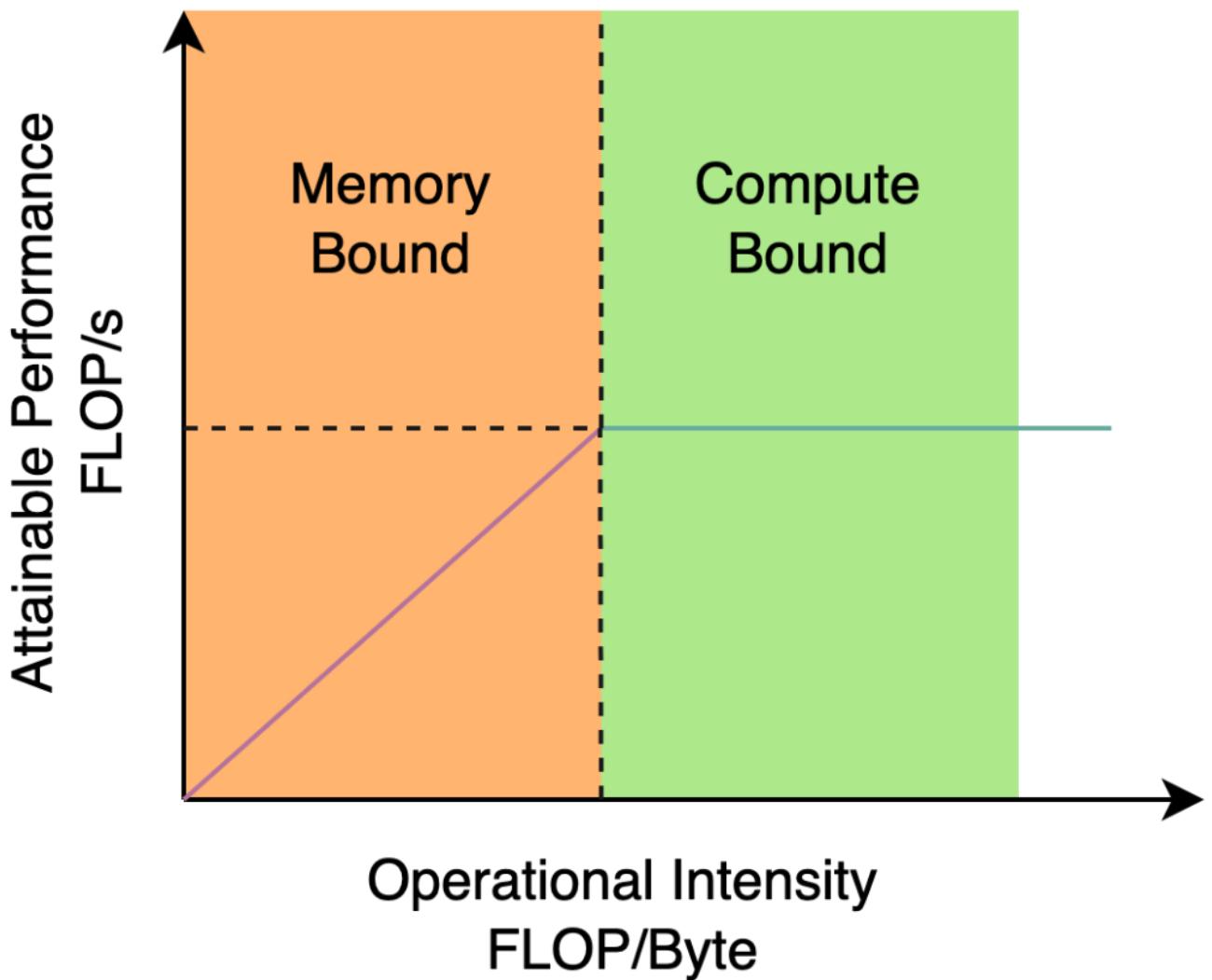
## 算法改进

YOLOv6 主要在 Backbone、Neck、Head 以及训练策略等方面进行了诸多的改进：

- 统一设计了更高效的 Backbone 和 Neck：受到硬件感知神经网络设计思想的启发，基于 RepVGG style设计了可重参数化、更高效的骨干网络 EfficientRep Backbone 和 Rep-PAN Neck。
- 优化设计了更简洁有效的 Efficient Decoupled Head，在维持精度的同时，进一步降低了一般解耦头带来的额外延时开销。
- 在训练策略上，我们采用Anchor-free 无锚范式，同时辅以 SimOTA标签分配策略以及 SIoU边界框回归损失来进一步提高检测精度。

## Hardware-friendly 的骨干网络设计

- YOLOv5/YOLOX 使用的 Backbone 和 Neck 都基于 CSPNet[5] 搭建，采用了多分支的方式和残差结构。对于 GPU 等硬件来说，这种结构会一定程度上增加延时，同时减小内存带宽利用率。下图为计算机体系结构领域中的 Roofline Model介绍图，显示了硬件中计算能力和内存带宽之间的关联关系。



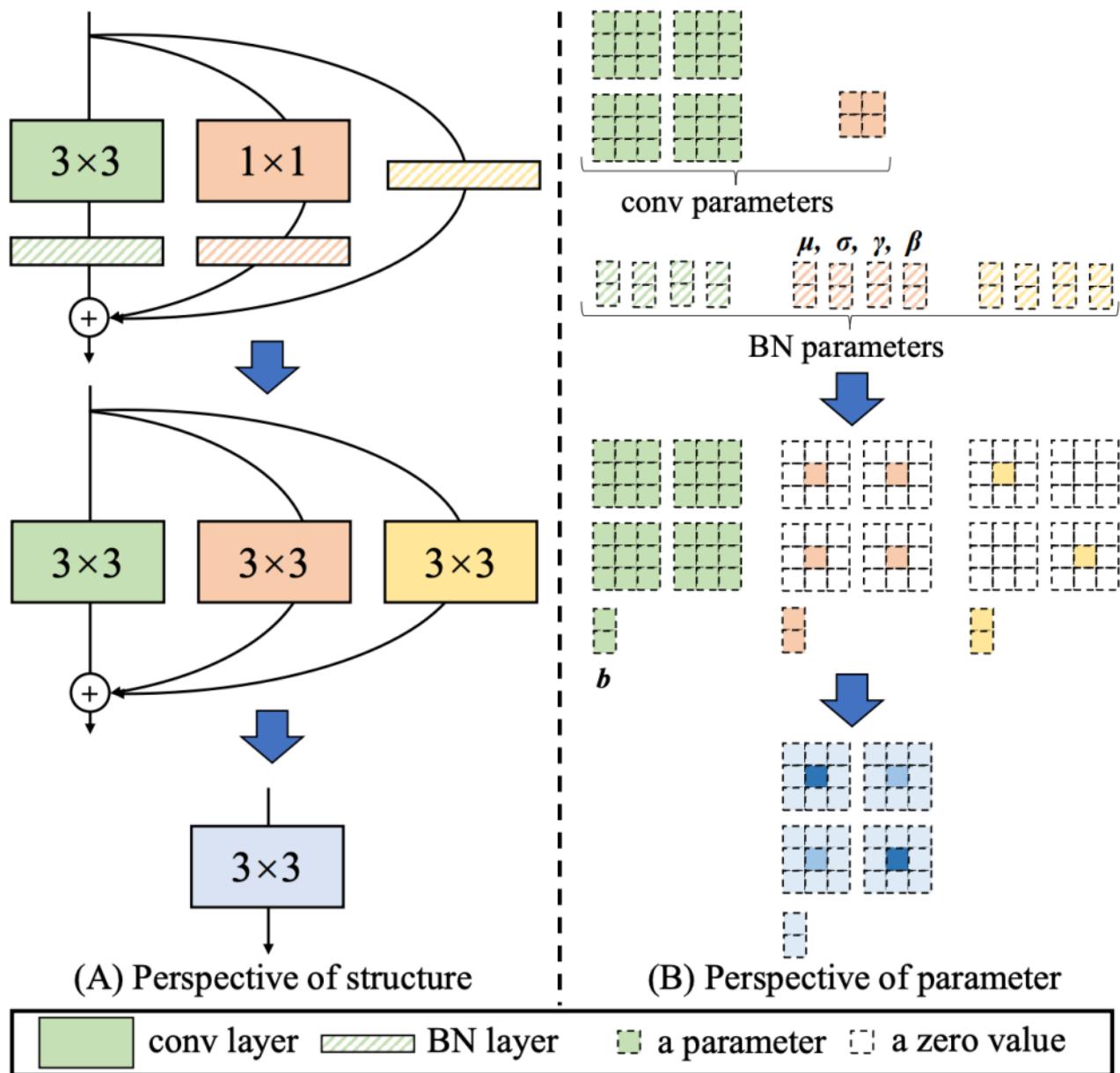
于是，我们基于硬件感知神经网络设计的思想，对 Backbone 和 Neck 进行了重新设计和优化。该思想基于硬件的特性、推理框架/编译框架的特点，以硬件和编译友好的结构作为设计原则，在网络构建时，综合考虑硬件计算能力、内存带宽、编译优化特性、网络表征能力等，进而获得又快又好的网络结构。对上述重新设计的两个检测部件，我们在 YOLOv6 中分别称为 EfficientRep Backbone 和 Rep-PAN Neck，其主要贡献点在于：

- 引入了 RepVGG[4] style 结构。
- 基于硬件感知思想重新设计了 Backbone 和 Neck。

RepVGG Style 结构是一种在训练时具有多分支拓扑，而在实际部署时可以等效融合为单个  $3 \times 3$  卷积的一种可重参数化的结构（融合过程如下图所示）。通过融合成的  $3 \times 3$  卷积结构，可以有效利用计算密集型硬件计算能力（比如 GPU），同时也可获得 GPU/CPU 上已经高度优化的 NVIDIA cuDNN 和 Intel MKL 编译框架的帮助。

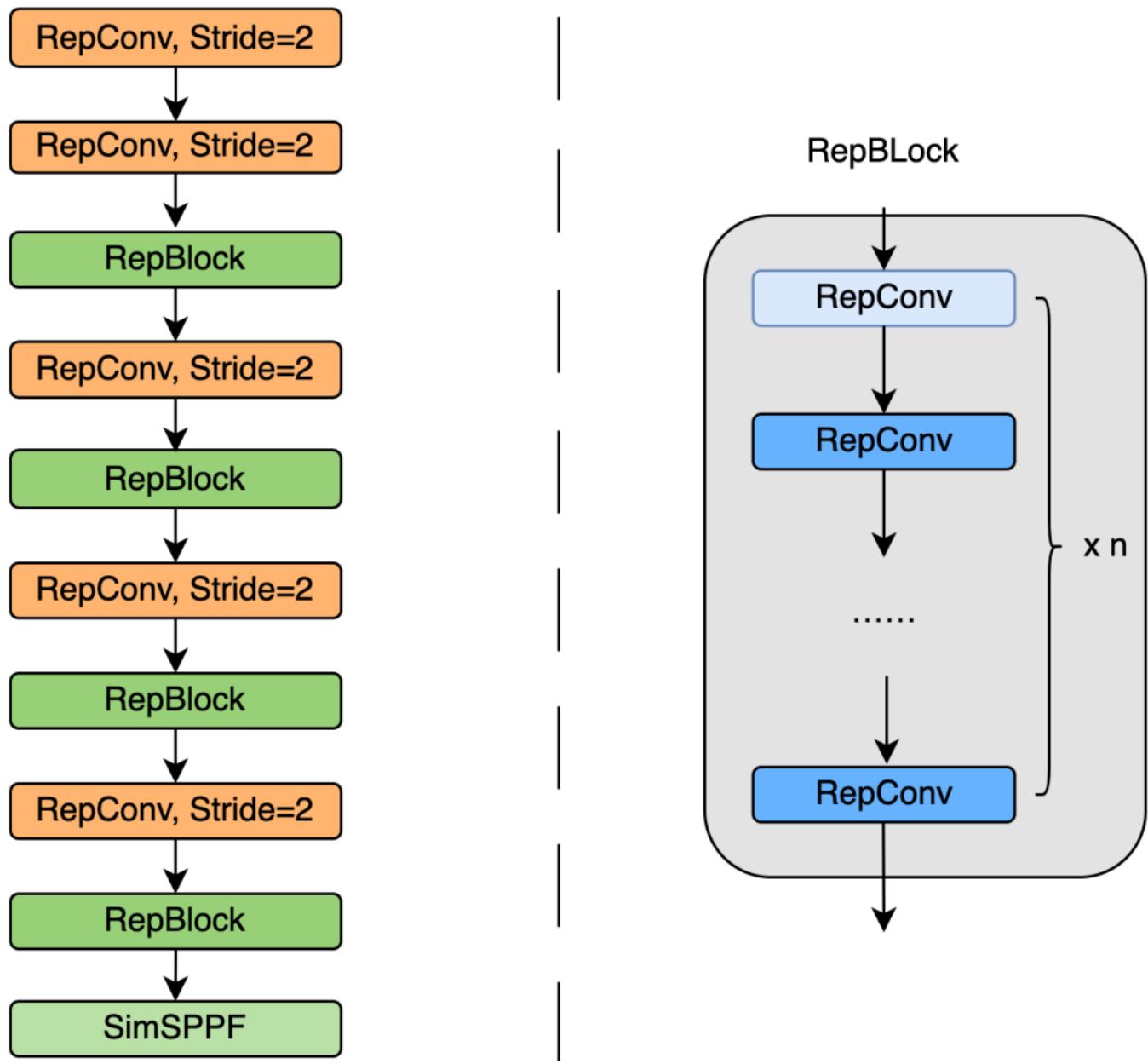
实验表明，通过上述策略，YOLOv6 减少了在硬件上的延时，并显著提升了算法的精度，让检测网络更快更强。以 nano 尺寸模型为例，对比 YOLOv5-nano 采用的网络结

构，本方法在速度上提升了21%，同时精度提升3.6% AP。



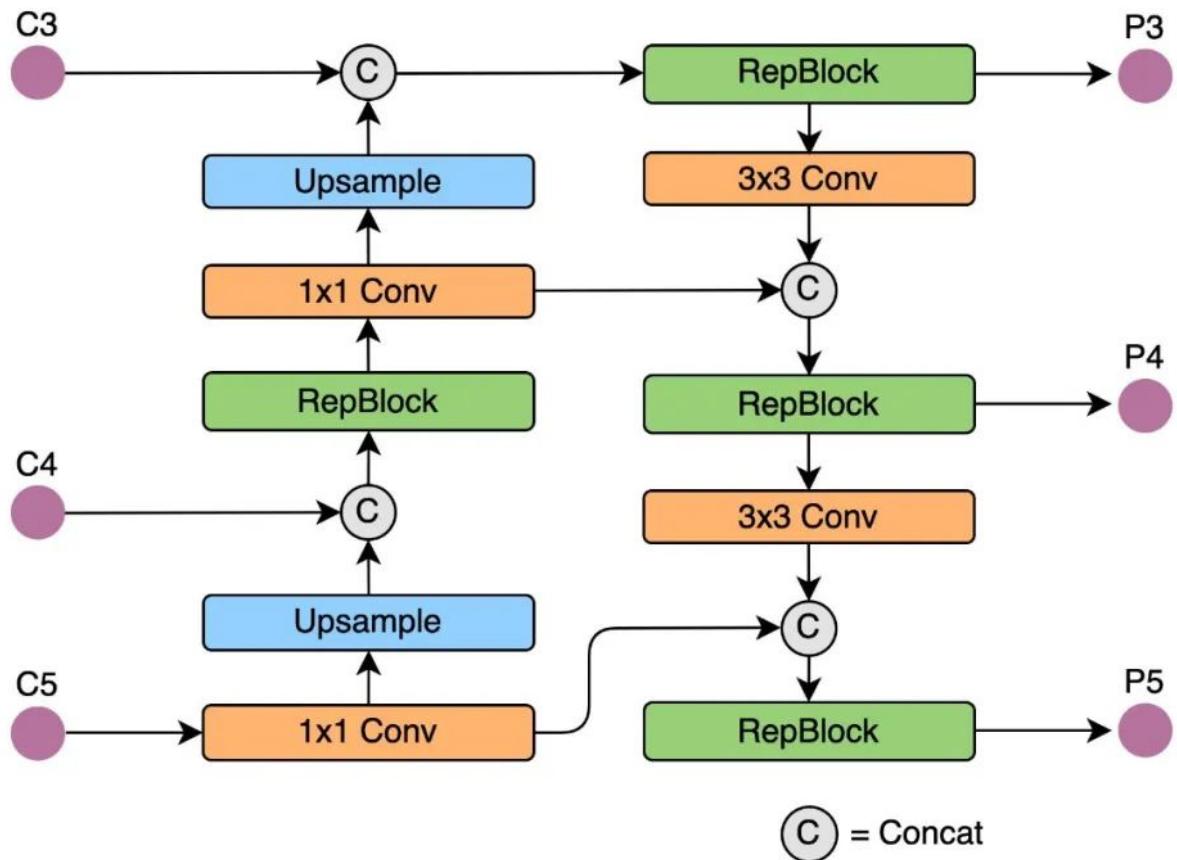
**EfficientRep Backbone:** 在 Backbone 设计方面，我们基于以上 Rep 算子设计了一个高效的Backbone。相比于 YOLOv5 采用的 CSP-Backbone，该 Backbone 能够高效利用硬件（如 GPU）算力的同时，还具有较强的表征能力。

下图为 EfficientRep Backbone 具体设计结构图，我们将 Backbone 中 stride=2 的普通 Conv 层替换成了 stride=2 的 RepConv 层。同时，将原始的 CSP-Block 都重新设计为 RepBlock，其中 RepBlock 的第一个 RepConv 会做 channel 维度的变换和对齐。另外，我们还将原始的 SPPF 优化设计为更加高效的 SimSPPF。



**Rep-PAN:** 在 Neck 设计方面，为了让其在硬件上推理更加高效，以达到更好的精度与速度的平衡，我们基于硬件感知神经网络设计思想，为 YOLOv6 设计了一个更有效的特征融合网络结构。

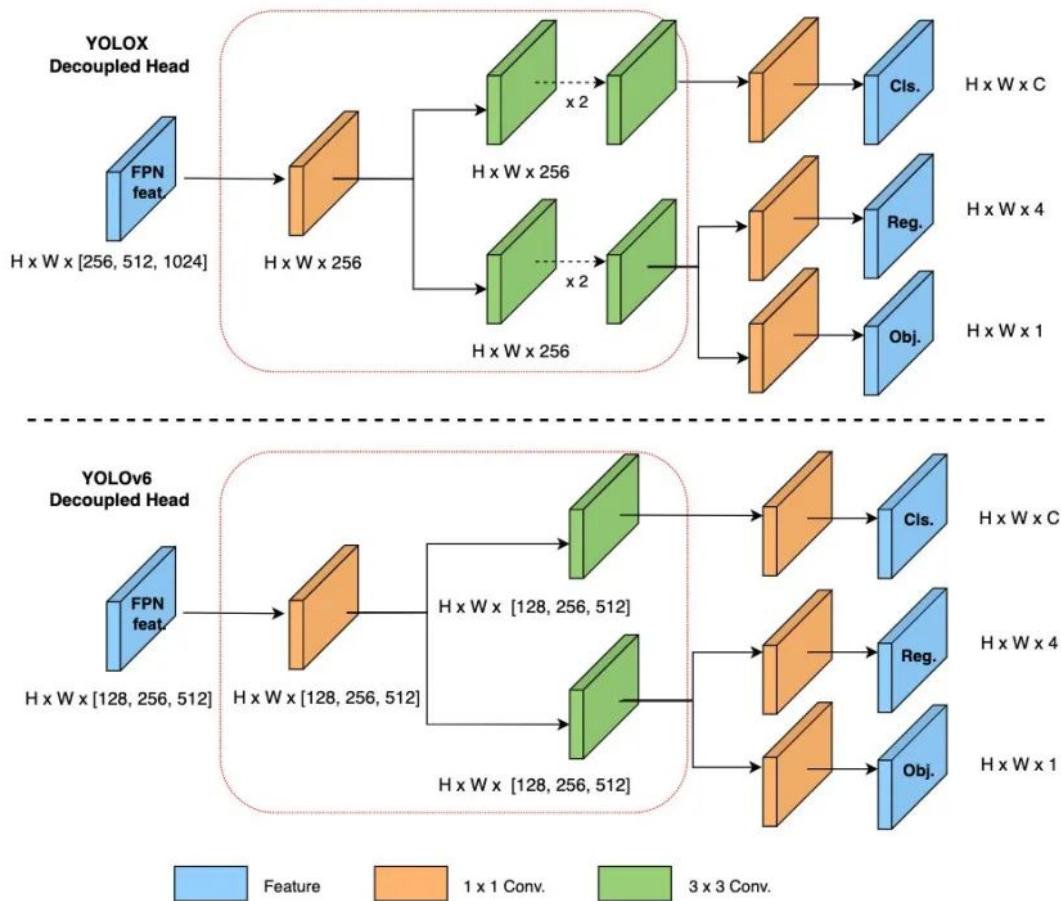
Rep-PAN 基于 PAN 拓扑方式，用 RepBlock 替换了 YOLOv5 中使用的 CSP-Block，同时对整体 Neck 中的算子进行了调整，目的是在硬件上达到高效推理的同时，保持较好的多尺度特征融合能力（Rep-PAN 结构图如下图所示）。



### 更简洁高效的 Decoupled Head

在 YOLOv6 中，我们采用了解耦检测头（Decoupled Head）结构，并对其进行了精简设计。原始 YOLOv5 的检测头是通过分类和回归分支融合共享的方式来实现的，而 YOLOX 的检测头则是将分类和回归分支进行解耦，同时新增了两个额外的 3x3 的卷积层，虽然提升了检测精度，但一定程度上增加了网络延时。

因此，我们对解耦头进行了精简设计，同时综合考虑到相关算子表征能力和硬件上计算开销这两者的平衡，采用 Hybrid Channels 策略重新设计了一个更高效的解耦头结构，在维持精度的同时降低了延时，缓解了解耦头中 3x3 卷积带来的额外延时开销。通过在 nano 尺寸模型上进行消融实验，对比相同通道数的解耦头结构，精度提升 0.2% AP 的同时，速度提升 6.8%。



## 更有效的训练策略

为了进一步提升检测精度，我们吸收借鉴了学术界和业界其他检测框架的先进研究进展：Anchor-free 无锚范式、SimOTA 标签分配策略以及 SIoU 边界框回归损失。

### Anchor-free 无锚范式

YOLOv6 采用了更简洁的 Anchor-free 检测方法。由于 Anchor-based 检测器需要在训练之前进行聚类分析以确定最佳 Anchor 集合，这会一定程度提高检测器的复杂度；同时，在一些边缘端的应用中，需要在硬件之间搬运大量检测结果的步骤，也会带来额外的延时。而 Anchor-free 无锚范式因其泛化能力强，解码逻辑更简单，在近几年中应用比较广泛。经过对 Anchor-free 的实验调研，我们发现，相较于 Anchor-based 检测器的复杂度而带来的额外延时，Anchor-free 检测器在速度上有 51% 的提升。

### SimOTA 标签分配策略

为了获得更多高质量的正样本，YOLOv6 引入了 SimOTA [4] 算法动态分配正样本，进一步提高检测精度。YOLOv5 的标签分配策略是基于 Shape 匹配，并通过跨网格匹配策略增加正样本数量，从而使得网络快速收敛，但是该方法属于静态分配方法，并不会随着网络训练的过程而调整。

近年来，也出现不少基于动态标签分配的方法，此类方法会根据训练过程中的网络输出来分配正样本，从而可以产生更多高质量的正样本，继而又促进网络的正向优化。例如，OTA通过将样本匹配建模成最佳传输问题，求得全局信息下的最佳样本匹配策略以提升精度，但 OTA 由于使用了Sinkhorn-Knopp 算法导致训练时间加长，而 SimOTA[4] 算法使用 Top-K 近似策略来得到样本最佳匹配，大大加快了训练速度。故 YOLOv6 采用了SimOTA 动态分配策略，并结合无锚范式，在 nano 尺寸模型上平均检测精度提升 1.3% AP。

#### **SIoU** 边界框回归损失

为了进一步提升回归精度，YOLOv6 采用了 SIoU边界框回归损失函数来监督网络的学习。目标检测网络的训练一般需要至少定义两个损失函数：分类损失和边界框回归损失，而损失函数的定义往往对检测精度以及训练速度产生较大的影响。

近年来，常用的边界框回归损失包括IoU、GIoU、CIoU、DIoU loss等等，这些损失函数通过考虑预测框与目标框之前的重叠程度、中心点距离、纵横比等因素来衡量两者之间的差距，从而指导网络最小化损失以提升回归精度，但是这些方法都没有考虑到预测框与目标框之间方向的匹配性。SIoU 损失函数通过引入了所需回归之间的向量角度，重新定义了距离损失，有效降低了回归的自由度，加快网络收敛，进一步提升了回归精度。通过在 YOLOv6s 上采用 SIoU loss 进行实验，对比 CIoU loss，平均检测精度提升 0.3% AP。

## **YOLOv7**

论文下载：[YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time objectdetectors](#) 代码地址：<https://gitcode.net/mirrors/WongKinYiu/yolov7>

在项目实战中，只研究yolov5或yolov7对应的项目即可，yolov3不要再研究了。因为现在的torch版本是高版本，而v3当时是低版本。

## **性能表现**

## MS COCO Object Detection

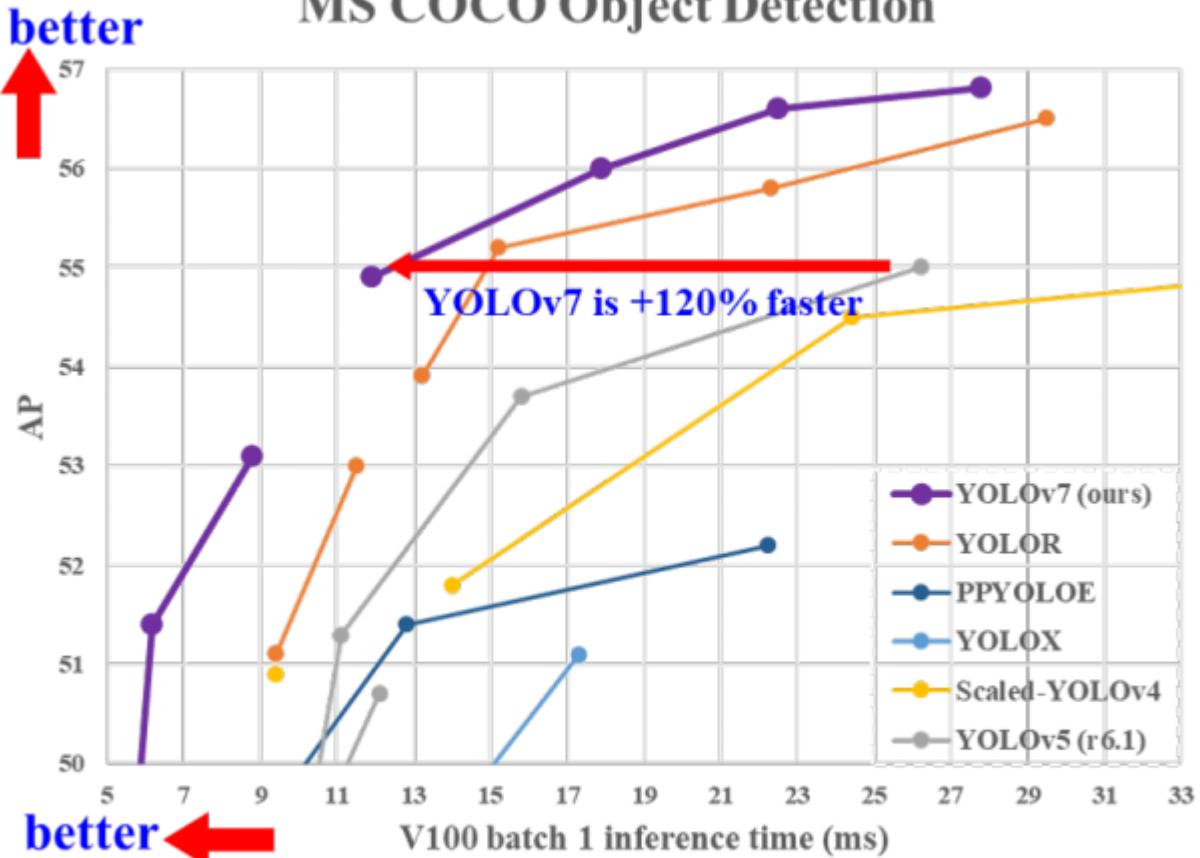
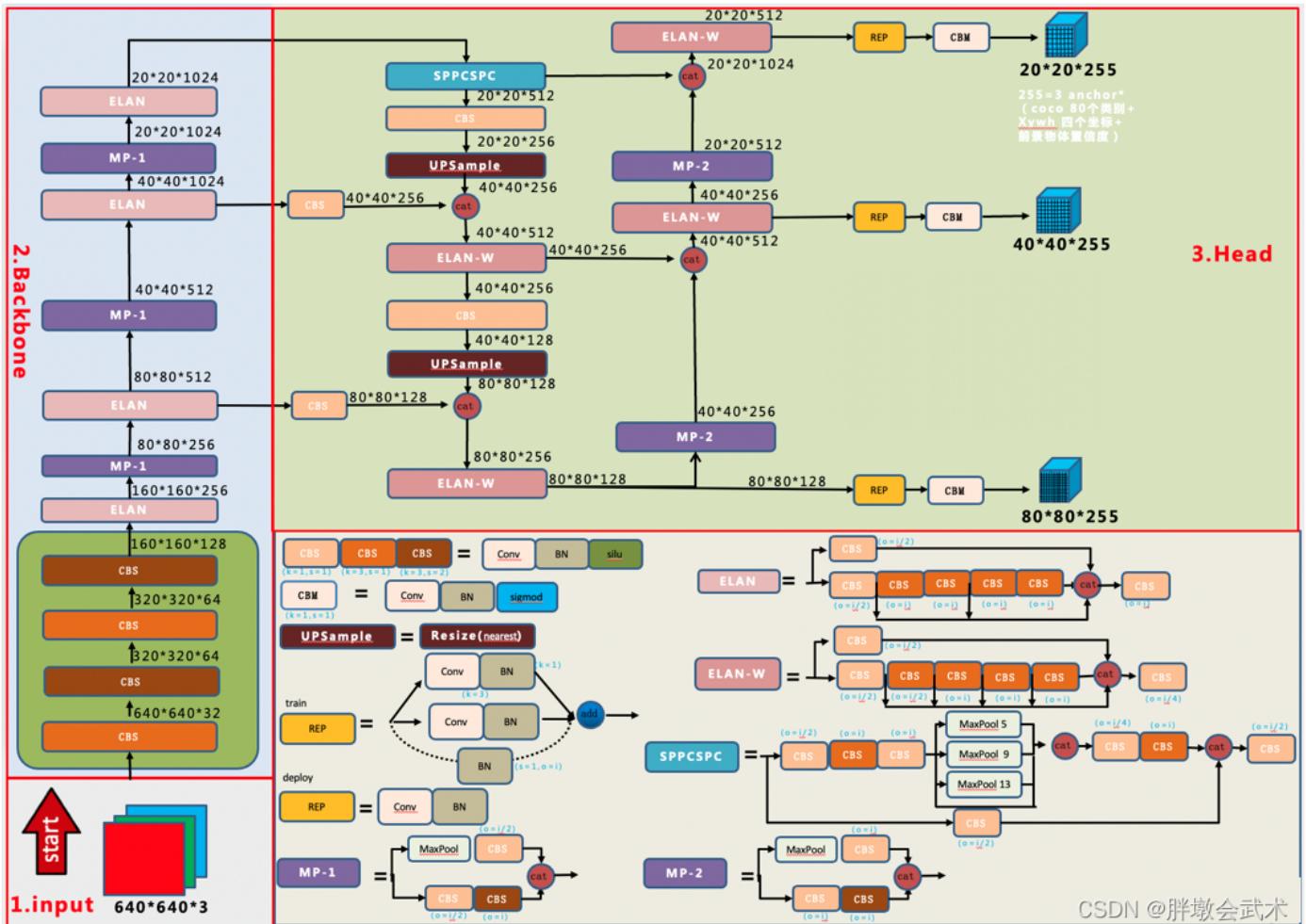


Figure 1: Comparison with other real-time object detectors, our proposed methods achieve state-of-the-arts performance.

Table 2: Comparison of state-of-the-art real-time object detectors.

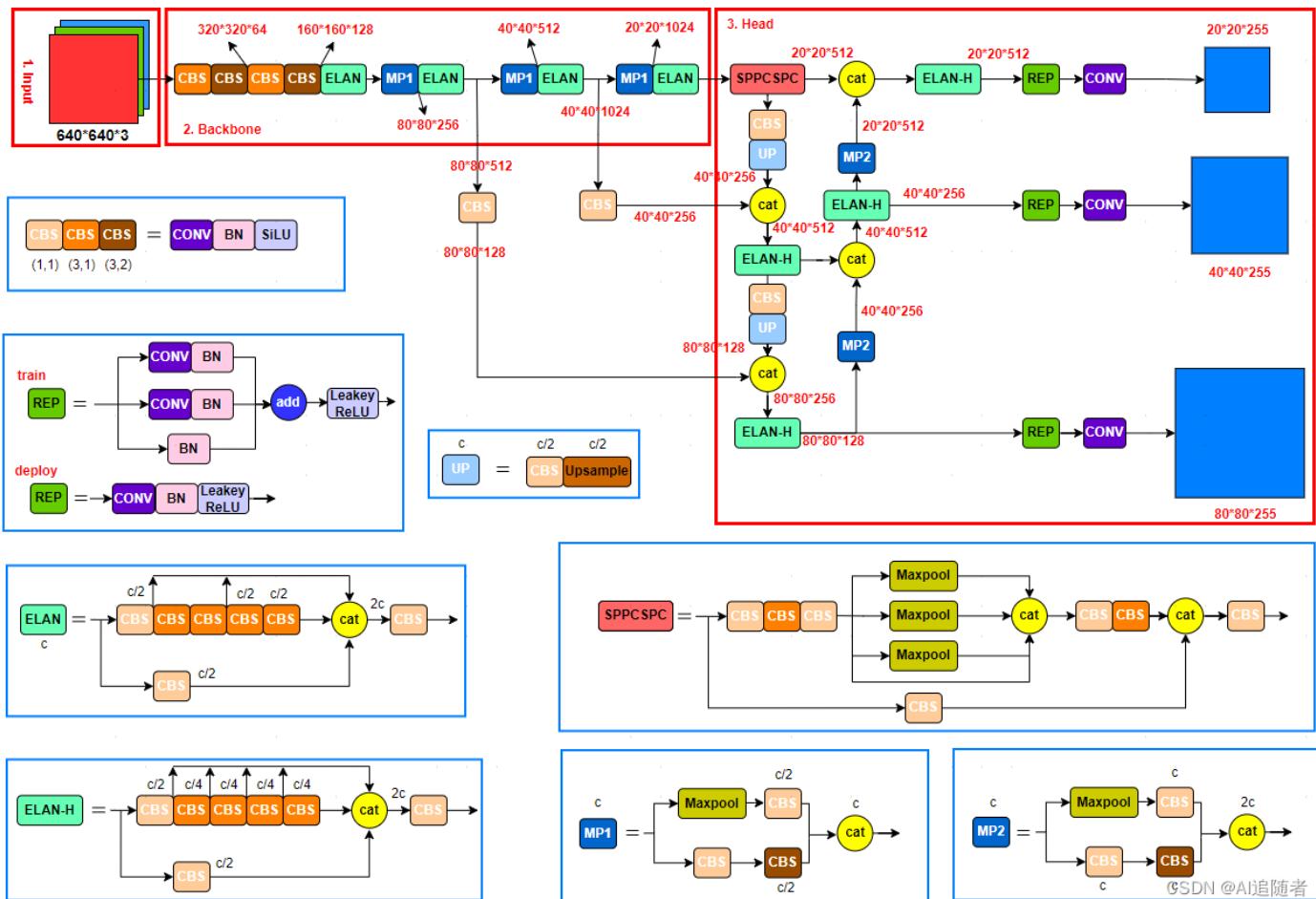
Model	#Param.	FLOPs	Size	FPS	AP <sup>test</sup> / AP <sup>val</sup>	AP <sup>test</sup> <sub>50</sub>	AP <sup>test</sup> <sub>75</sub>	AP <sup>test</sup> <sub>S</sub>	AP <sup>test</sup> <sub>M</sub>	AP <sup>test</sup> <sub>L</sub>
YOLOX-S [21]	9.0M	26.8G	640	102	40.5% / 40.5%	-	-	-	-	-
YOLOX-M [21]	25.3M	73.8G	640	81	47.2% / 46.9%	-	-	-	-	-
YOLOX-L [21]	54.2M	155.6G	640	69	50.1% / 49.7%	-	-	-	-	-
YOLOX-X [21]	99.1M	281.9G	640	58	51.5% / 51.1%	-	-	-	-	-
PPYOLOE-S [85]	7.9M	17.4G	640	208	43.1% / 42.7%	60.5%	46.6%	23.2%	46.4%	56.9%
PPYOLOE-M [85]	23.4M	49.9G	640	123	48.9% / 48.6%	66.5%	53.0%	28.6%	52.9%	63.8%
PPYOLOE-L [85]	52.2M	110.1G	640	78	51.4% / 50.9%	68.9%	55.6%	31.4%	55.3%	66.1%
PPYOLOE-X [85]	98.4M	206.6G	640	45	52.2% / 51.9%	69.9%	56.5%	33.3%	56.3%	66.4%
YOLOv5-N (r6.1) [23]	1.9M	4.5G	640	159	- / 28.0%	-	-	-	-	-
YOLOv5-S (r6.1) [23]	7.2M	16.5G	640	156	- / 37.4%	-	-	-	-	-
YOLOv5-M (r6.1) [23]	21.2M	49.0G	640	122	- / 45.4%	-	-	-	-	-
YOLOv5-L (r6.1) [23]	46.5M	109.1G	640	99	- / 49.0%	-	-	-	-	-
YOLOv5-X (r6.1) [23]	86.7M	205.7G	640	83	- / 50.7%	-	-	-	-	-
YOLOR-CSP [81]	52.9M	120.4G	640	106	51.1% / 50.8%	69.6%	55.7%	31.7%	55.3%	64.7%
YOLOR-CSP-X [81]	96.9M	226.8G	640	87	53.0% / 52.7%	71.4%	57.9%	33.7%	57.1%	66.8%
YOLOv7-tiny-SiLU	6.2M	13.8G	640	286	38.7% / 38.7%	56.7%	41.7%	18.8%	42.4%	51.9%
YOLOv7	36.9M	104.7G	640	161	51.4% / 51.2%	69.7%	55.9%	31.8%	55.5%	65.0%
YOLOv7-X	71.3M	189.9G	640	114	53.1% / 52.9%	71.2%	57.8%	33.8%	57.1%	67.4%
YOLOv5-N6 (r6.1) [23]	3.2M	18.4G	1280	123	- / 36.0%	-	-	-	-	-
YOLOv5-S6 (r6.1) [23]	12.6M	67.2G	1280	122	- / 44.8%	-	-	-	-	-
YOLOv5-M6 (r6.1) [23]	35.7M	200.0G	1280	90	- / 51.3%	-	-	-	-	-
YOLOv5-L6 (r6.1) [23]	76.8M	445.6G	1280	63	- / 53.7%	-	-	-	-	-
YOLOv5-X6 (r6.1) [23]	140.7M	839.2G	1280	38	- / 55.0%	-	-	-	-	-
YOLOR-P6 [81]	37.2M	325.6G	1280	76	53.9% / 53.5%	71.4%	58.9%	36.1%	57.7%	65.6%
YOLOR-W6 [81]	79.8G	453.2G	1280	66	55.2% / 54.8%	72.7%	60.5%	37.7%	59.1%	67.1%
YOLOR-E6 [81]	115.8M	683.2G	1280	45	55.8% / 55.7%	73.4%	61.1%	38.4%	59.7%	67.7%
YOLOR-D6 [81]	151.7M	935.6G	1280	34	56.5% / 56.1%	74.1%	61.9%	38.9%	60.4%	68.7%
YOLOv7-W6	70.4M	360.0G	1280	84	54.9% / 54.6%	72.6%	60.1%	37.3%	58.7%	67.1%
YOLOv7-E6	97.2M	515.2G	1280	56	56.0% / 55.9%	73.5%	61.2%	38.0%	59.9%	68.4%
YOLOv7-D6	154.7M	806.8G	1280	44	56.6% / 56.3%	74.0%	61.8%	38.8%	60.1%	69.5%
YOLOv7-E6E	151.7M	843.2G	1280	36	56.8% / 56.8%	74.4%	62.1%	39.3%	60.5%	69.0%

<sup>1</sup> Our FLOPs is calculated by rectangle input resolution like 640 × 640 or 1280 × 1280.<sup>2</sup> Our inference time is estimated by using letterbox resize input image to make its long side equals to 640 or 1280.



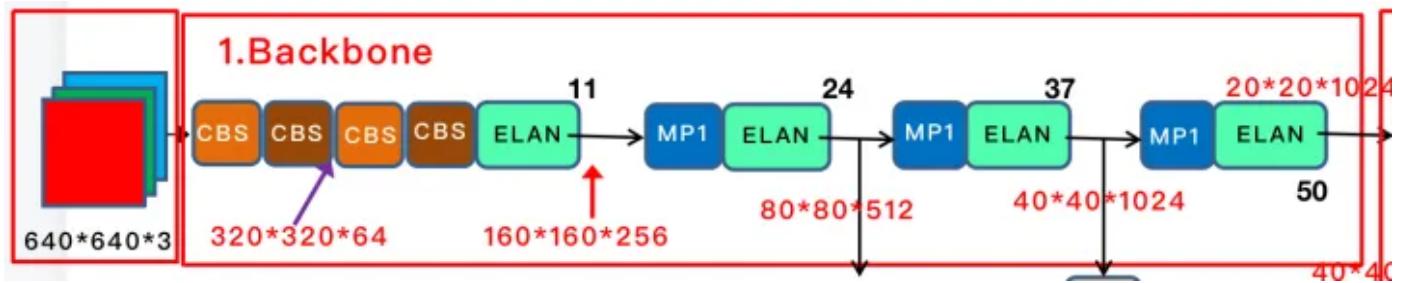
CSDN @胖墩会武术

我们先整体来看下 YOLOV7，首先对输入的图片 resize 为  $640 \times 640$  大小，输入到 backbone 网络中，然后经 head 层网络输出三层不同 size 大小的 **feature map**，经过 Rep 和 conv 输出预测结果，这里以 coco 为例子，输出为 80 个类别，然后每个输出( $x, y, w, h, o$ ) 即坐标位置和前后背景，3 是指的 anchor 数量，因此每一层的输出为  $(80+5) \times 3 = 255$  再乘上 feature map 的大小就是最终的输出了。

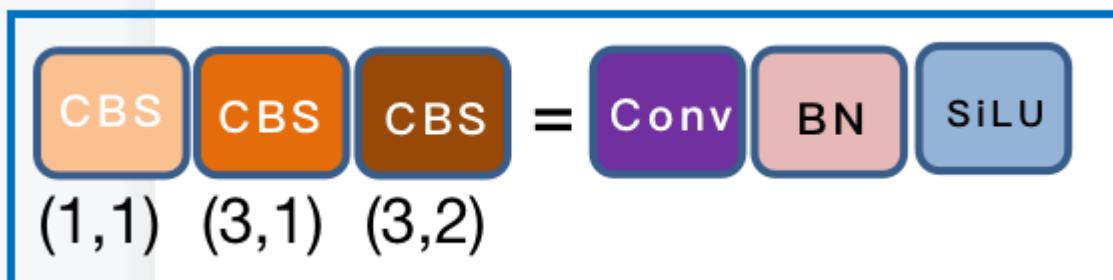


## Backbone

OLOV7 的 backbone 如下图所示



总共有 50 层, 我在上图用黑色数字把关键层数标示出来了。首先是经过 4 层卷积层, 如下图, CBS 主要是 Conv + BN + SiLU 构成, 我在图中用不同的颜色表示不同的 size 和 stride, 如 (3, 2) 表示卷积核大小为 3, 步长为 2。在 config 中的配置如图。



```

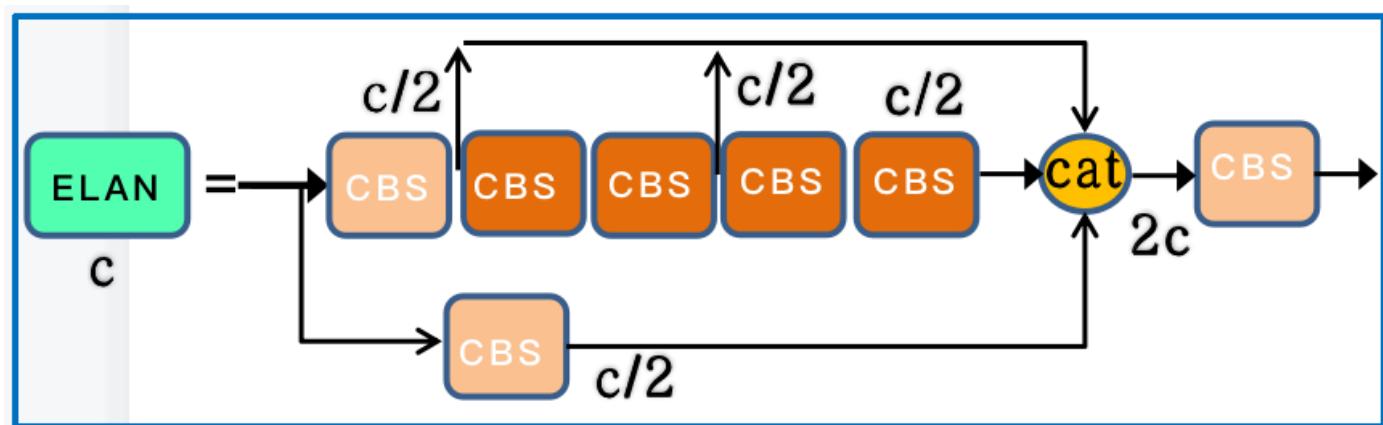
# [from, number, module, args]
[[-1, 1, Conv, [32, 3, 1]], # 0

[-1, 1, Conv, [64, 3, 2]], # 1-P1/2
[-1, 1, Conv, [64, 3, 1]],

[-1, 1, Conv, [128, 3, 2]], # 3-P2/4

```

经过 4 个 CBS 后，特征图变为  $160 * 160 * 128$  大小。随后会经过论文中提出的 ELAN 模块，ELAN 由多个 CBS 构成，其输入输出特征大小保持不变，通道数在开始的两个 CBS 会有变化，后面的几个输入通道都是和输出通道保持一致的，经过最后一个 CBS 输出为需要的通道。



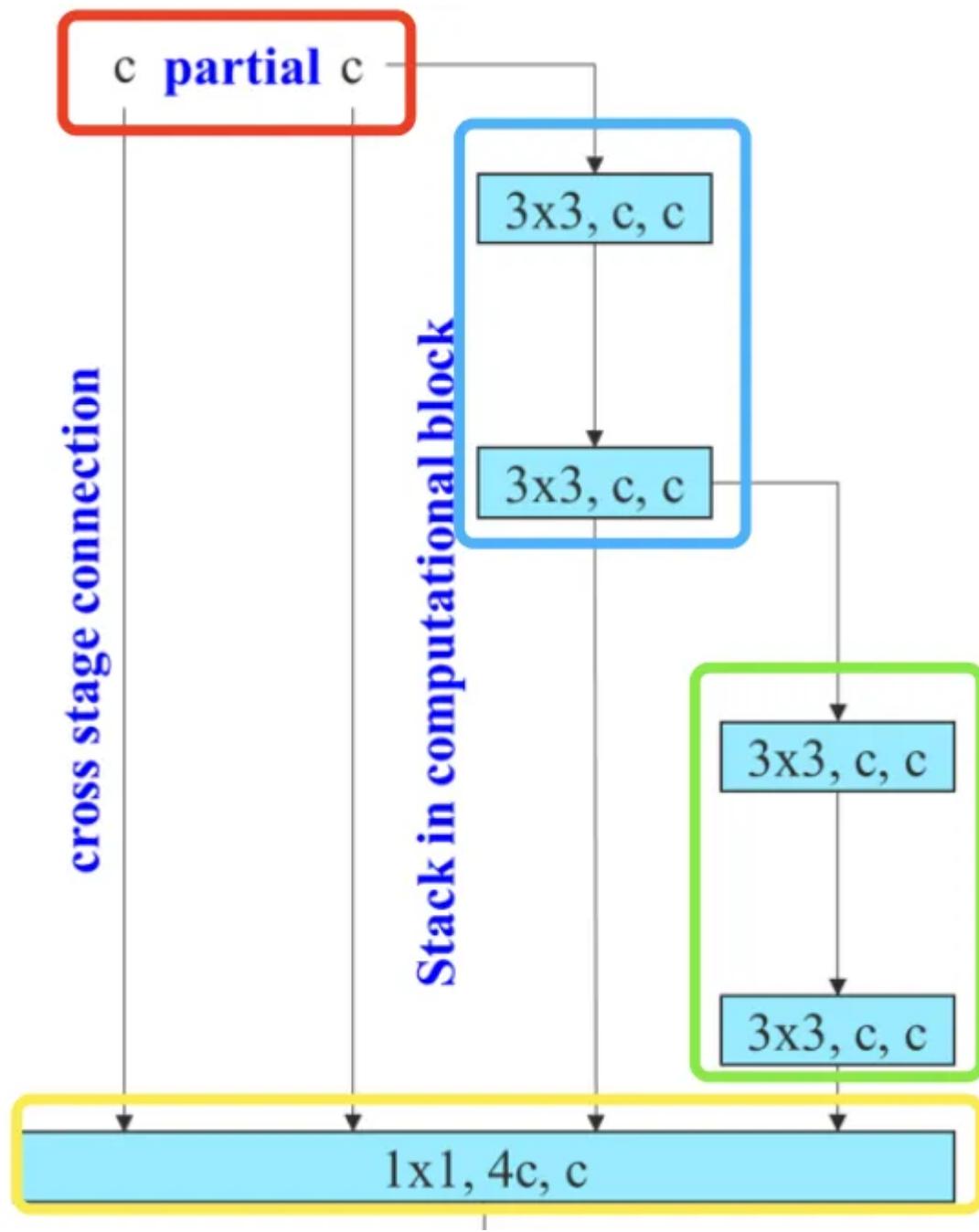
```

[-1, 1, Conv, [128, 3, 2]], # 3-P2/4
[-1, 1, Conv, [64, 1, 1]],
[-2, 1, Conv, [64, 1, 1]],

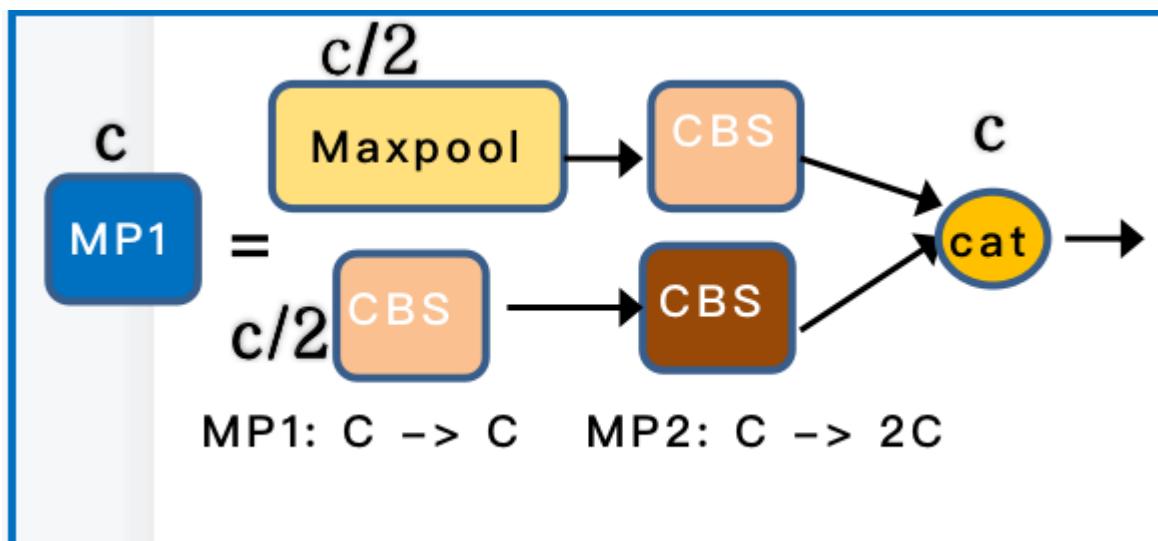
[-1, 1, Conv, [64, 3, 1]],
[-1, 1, Conv, [64, 3, 1]],
[-1, 1, Conv, [64, 3, 1]],
[-1, 1, Conv, [64, 3, 1]],

[[-1, -3, -5, -6], 1, Concat, [1]],
[-1, 1, Conv, [256, 1, 1]], # 11

```

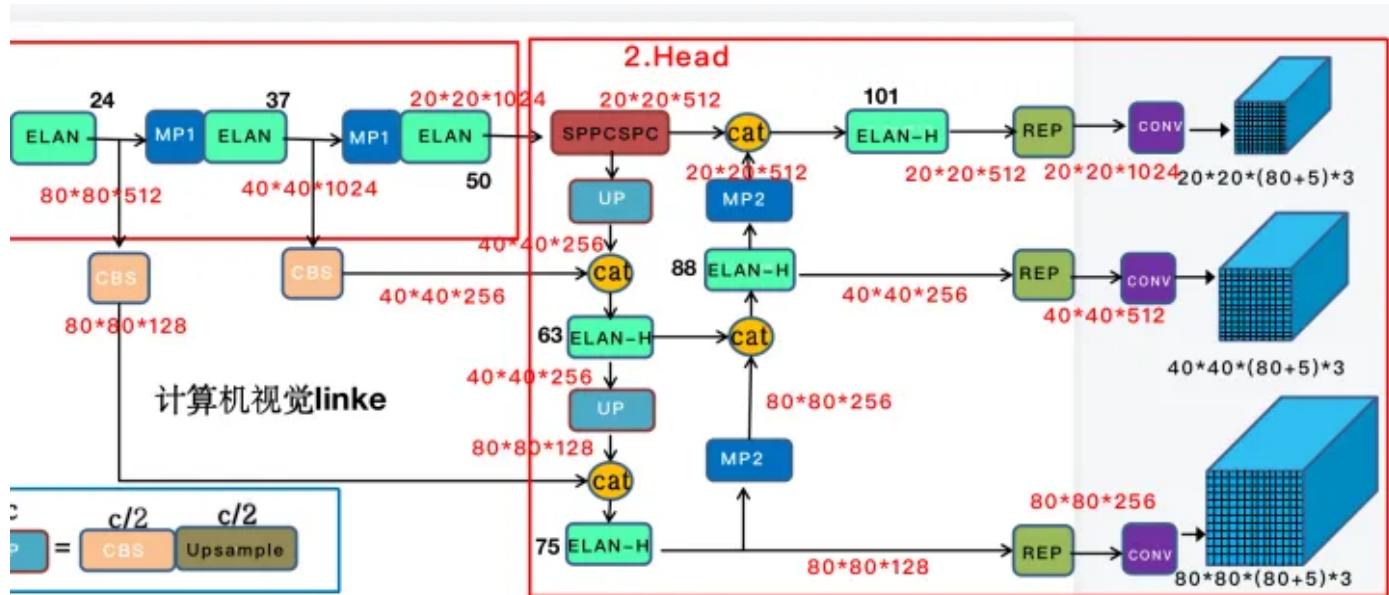


MP 层主要是分为 Maxpool 和 CBS，其中 MP1 和 MP2 主要是通道数的比变化。



backbone的基本组件就介绍完了，我们整体来看下 backbone，经过 4 个 CBS 后，接入例如一个 ELAN，然后后面就是三个 MP + ELAN 的输出，对应的就是 C3/C4/C5 的输出，大小分别为  $80 * 80 * 512$ ， $40 * 40 * 1024$ ， $20 * 20 * 1024$ 。每一个 MP 由 5 层，ELAN 有 8 层，所以整个 backbone 的层数为  $4 + 8 + 13 * 3 = 51$  层，从 0 开始的话，最后一层就是第 50 层。

## Head



```
# yolov7 head  
head:  
    [[-1, 1, SPPCSPC, [512]], # 51  
  
     [-1, 1, Conv, [256, 1, 1]],  
     [-1, 1, nn.Upsample, [None, 2, 'nearest']],  
     [37, 1, Conv, [256, 1, 1]], # route backbone P4  
     [[-1, -2], 1, Concat, [1]],  
  
     [-1, 1, Conv, [256, 1, 1]],  
     [-2, 1, Conv, [256, 1, 1]],  
     [-1, 1, Conv, [128, 3, 1]],  
     [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],  
     [-1, 1, Conv, [256, 1, 1]], # 63  
  
     [-1, 1, Conv, [128, 1, 1]],  
     [-1, 1, nn.Upsample, [None, 2, 'nearest']],  
     [24, 1, Conv, [128, 1, 1]], # route backbone P3  
     [[-1, -2], 1, Concat, [1]],  
  
     [-1, 1, Conv, [128, 1, 1]],  
     [-2, 1, Conv, [128, 1, 1]],  
     [-1, 1, Conv, [64, 3, 1]],  
     [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],  
     [-1, 1, Conv, [128, 1, 1]], # 75
```

```
[-1, 1, MP, []],  
[-1, 1, Conv, [128, 1, 1]],  
[-3, 1, Conv, [128, 1, 1]],  
[-1, 1, Conv, [128, 3, 21],  
[[[-1, -3, 63], 1, Concat, [1]],
```

```
[-1, 1, Conv, [256, 1, 1]],  
[-2, 1, Conv, [256, 1, 1]],  
[-1, 1, Conv, [128, 3, 1]],  
[[[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],  
[-1, 1, Conv, [256, 1, 1]], # 88
```

```
[-1, 1, MP, []],  
[-1, 1, Conv, [256, 1, 1]],  
[-3, 1, Conv, [256, 1, 1]],  
[-1, 1, Conv, [256, 3, 21],  
[[[-1, -3, 51], 1, Concat, [1]],
```

```
[-1, 1, Conv, [512, 1, 1]],  
[-2, 1, Conv, [512, 1, 1]],  
[-1, 1, Conv, [256, 3, 1]],  
[[[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],  
[-1, 1, Conv, [512, 1, 1]], # 101
```

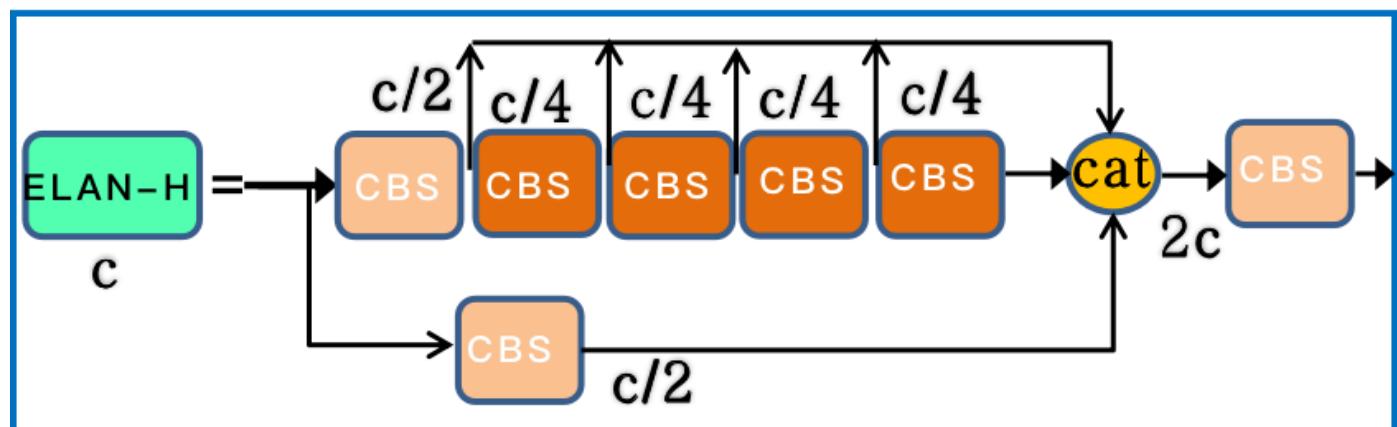
```
[75, 1, RepConv, [256, 3, 1]],  
[88, 1, RepConv, [512, 3, 1]],  
[101, 1, RepConv, [1024, 3, 1]],
```

```
[[102, 103, 104], 1, IDetect, [nc, anchors]], # Detect(P3, P4, P5)
```

YOLOV7 head 其实就是一个 **pafpn** 的结构，和之前的YOLOV4, YOLOV5一样。首先，对于 backbone 最后输出的 32 倍降采样特征图 C5，然后经过 SPPCSP，通道数从 1024 变为 512。先按照 top down 和 C4、C3 融合，得到 P3、P4 和 P5；再按 bottom-

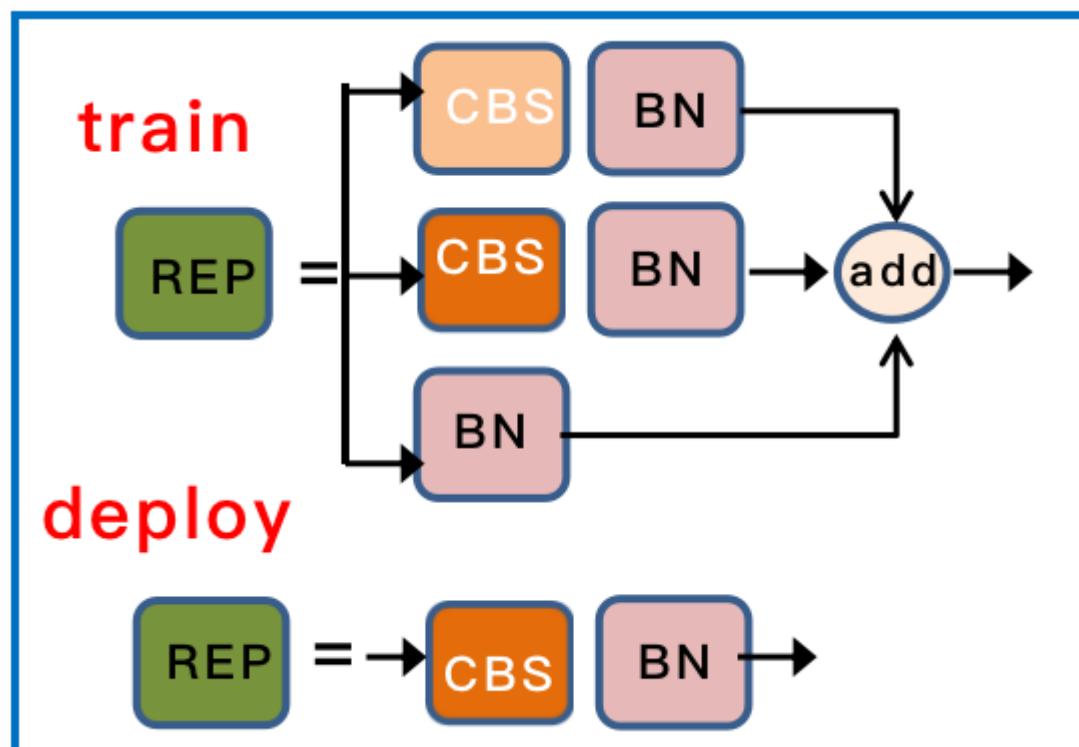
up 去和 P4、P5 做融合。这里基本和 YOLOV5 是一样的，区别在于将 YOLOV5 中的 CSP 模块换成了 ELAN-H 模块，同时下采样变为了 MP2 层。

ELAN-H 模块是我自己命名的，它和 backbone 中的 ELAN 稍微有点区别就是 cat 的数量不同。



对于 **pafpn** 输出的 P3、P4 和 P5，经过 RepConv 调整通道数，最后使用  $1 \times 1$  卷积去预测 objectness、class 和 bbox 三部分。

RepConv 在训练和推理是有一定的区别。训练时有三个分支的相加输出，部署时会将分支的参数重参数化到主分支上。



## Loss Function

主要分带和不带辅助训练头两种，对应的训练脚本是 train.py 和 train\_aux.py。

- 不带辅助训练头（分损失函数和匹配策略两部分讨论）。

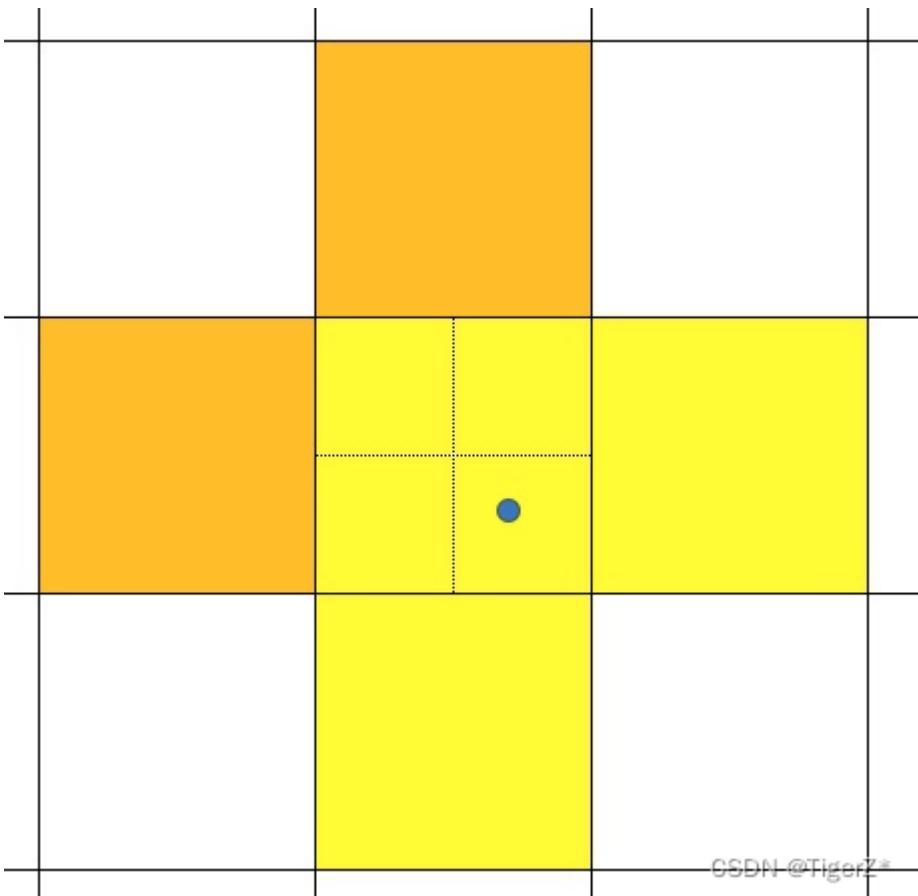
- 损失函数

整体和YOLOV5 保持一致，分为坐标损失、目标置信度损失（GT就是训练阶段的普通iou）和分类损失三部分。其中目标置信度损失和分类损失采用 BCEWithLogitsLoss（带log的二值交叉熵损失），坐标损失采用CIoU损失。详细参见utils/loss.py 里面的 ComputeLossOTA 函数 配合 配置文件里的各部分的权重设置。

- 匹配策略

主要是参考了YOLOV5 和YOLOV6使用的当下比较火的simOTA.

- S1.训练前，会基于训练集中gt框，通过k-means聚类算法，先验获得9个从小到大排列的anchor框。(可选)
- S2.将每个gt与9个anchor匹配：Yolov5为分别计算它与9种anchor的宽与宽的比值（较大的宽除以较小的宽，比值大于1，下面的高同样操作）、高与高的比值，在宽比值、高比值这2个比值中，取最大的一个比值，若这个比值小于设定的比值阈值，这个anchor的预测框就被称为正样本。一个gt可能与几个anchor均能匹配上（此时最大9个）。所以一个gt可能在不同的网络层上做预测训练，大大增加了正样本的数量，当然也会出现gt与所有anchor都匹配不上的情况，这样gt就会被当成背景，不参与训练，说明anchor框尺寸设计的不好。
- S3.扩充正样本。根据gt框的中心位置，将最近的2个邻域网格也作为预测网格，也即一个groundtruth框可以由3个网格来预测；可以发现粗略估计正样本数相比前yolo系列，增加了三倍（此时最大27个匹配）。图下图浅黄色区域，其中实线是YOLO的真实网格，虚线是将一个网格四等分，如这个例子中，GT的中心在右下虚线网格，则扩充右和下真实网格也作为正样本。
- S4.获取与当前gt有top10最大iou的prediction结果。将这top10（5-15之间均可，并不敏感）iou进行sum，就为当前gt的k。k最小取1。
- S5.根据损失函数计算每个GT和候选anchor损失（前期会加大分类损失权重，后面减低分类损失权重，如1:5->1:3），并保留损失最小的前K个。
- S6.去掉同一个anchor被分配到多个GT的情况。



CSDN @TigerZ\*

带辅助训练头（分损失函数和匹配策略两部分讨论）。

论文中，将负责最终输出的Head为lead Head，将用于辅助训练的Head称为auxiliary Head。本博客不重点讨论，原因是论文中后面的结构实验实现提升比较有限（0.3个点），具体可以看原文。

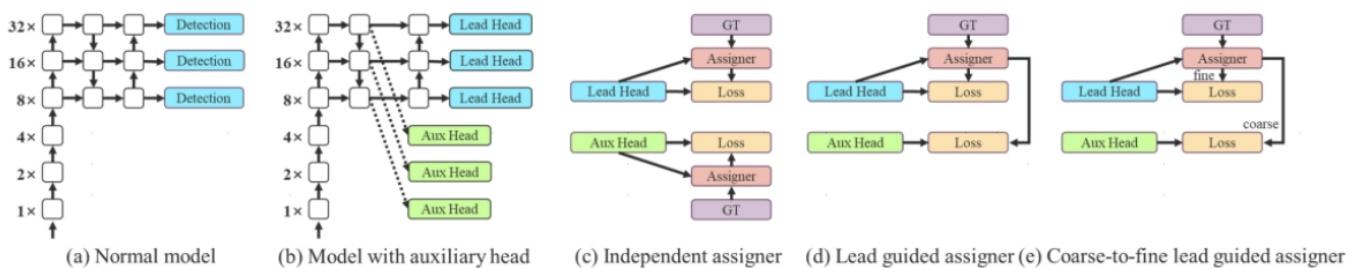


Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be

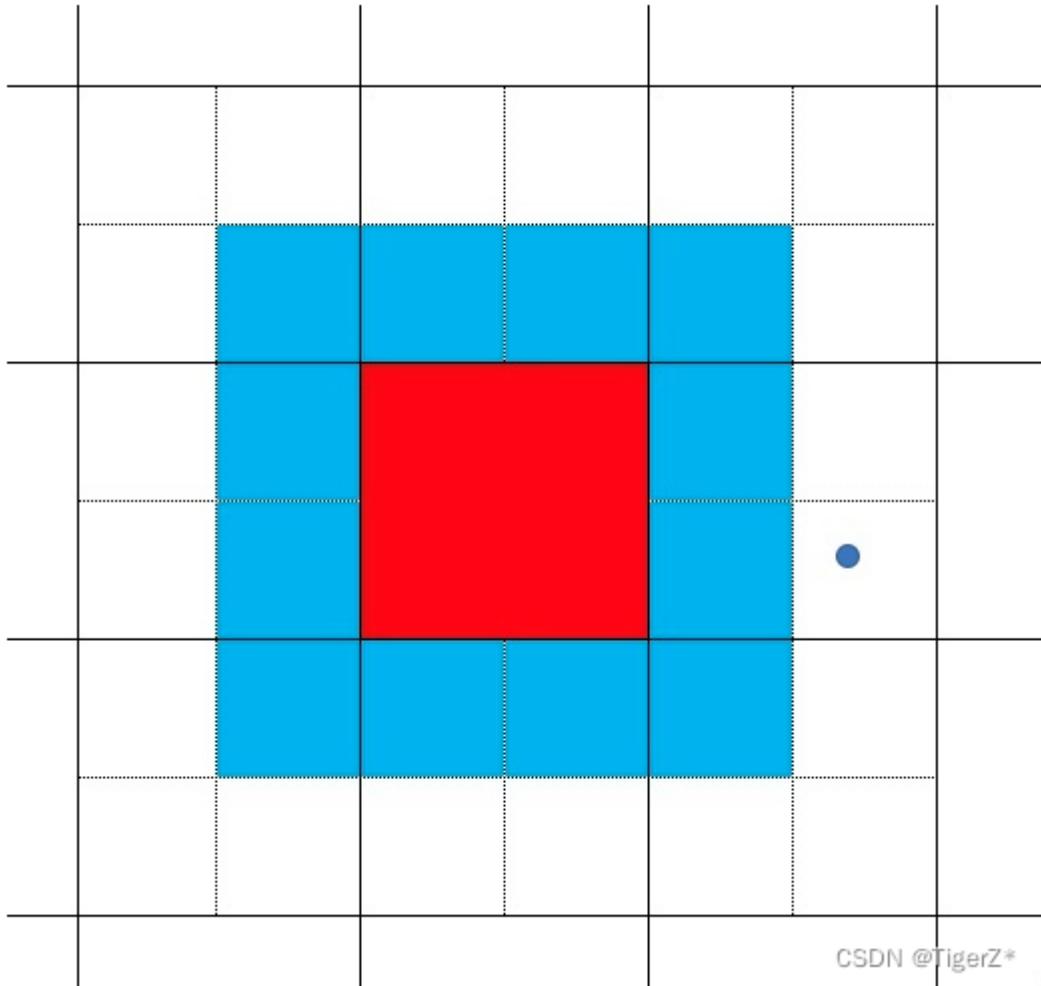
一些细节：其loss函数和不带辅助头相同，加权系数不能过大（aux head loss 和lead head loss 按照0.25:1的比例），否则会导致lead head出来的结果精度变低。匹配策略和上面的不带辅助头（只有lead head）只有很少不同，其中辅助头：

\*lead head中每个网格与gt如果匹配上，附加周边两个网格，而aux head附加4个网格（如上面导数第二幅图，匹配到浅黄+橘黄共5个网格）。

\*lead head中将top10个样本iou求和取整，而aux head中取top20。

aux head更关注于recall，而lead head从aux head中精准筛选出样本。

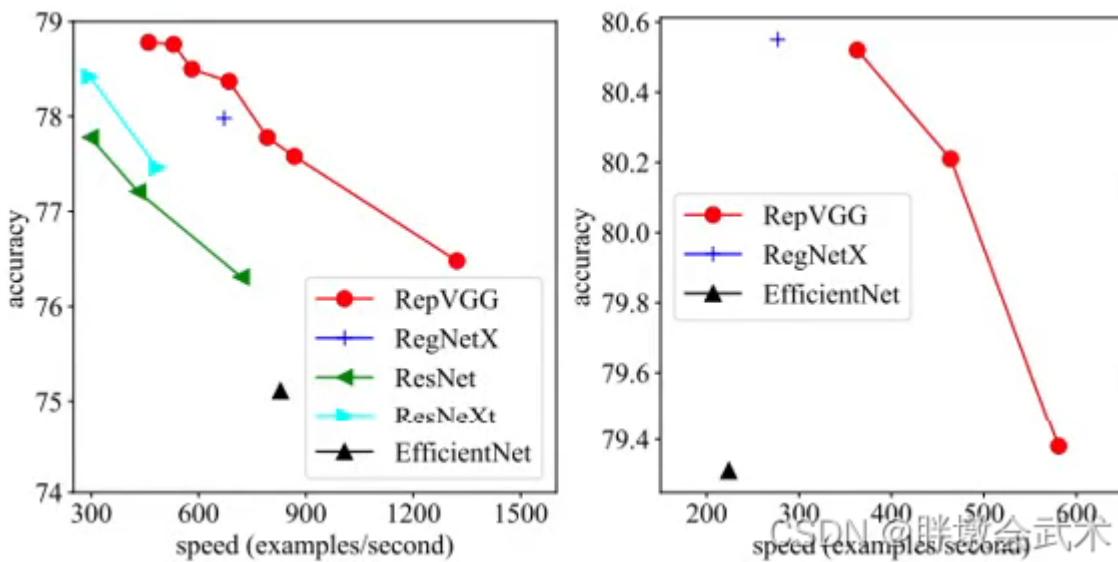
需要注意依照yolov5中的中心点回归方式，仅能将图中红色特征grid，预测在图中红色+蓝色区域（实线组成的网格代表着特征图grid，虚线代表着一个grid分成了4个象限），是根本无法将中心点预测到gt处(蓝色点)！而该红色特征grid在训练时是会作为正样本的。所以在aux head中，即使被分配为正样本的区域，经过不断的学习，可能仍然无法完全拟合至效果特别好。



## 算法改进

### RepVGG（最大改进）

- 2014年：牛津大学著名研究组VGG (Visual Geometry Group)，提出VGGNet。
- 2021年：清华大学、旷视科技以及香港科技大学等机构，基于VGG网络提出了RepVGG网络。



由图可得：RepVGG无论是在精度还是速度上都已经超过了ResNet、EffcientNet以及ResNeXt等网络。

## 结构重参数化

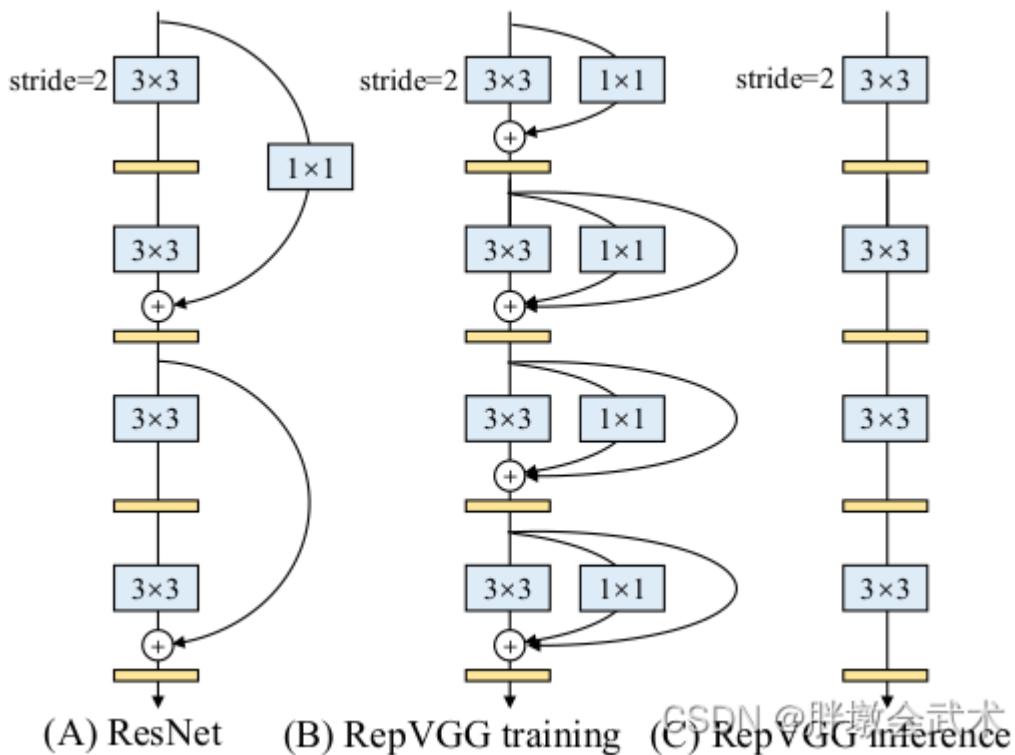
RepVGG采用结构重参数化方法（structural re-parameterization technique）。详细过程：

- (1) 在训练时，使用ResNet-style的多分支模型（特点：增加模型的表征能力）；
- (2) 在测试时，转化成VGG-style的单线路模型（特点：速度更快、更省内存并且更加的灵活。）。过程特点：训练的网络结构和测试的网络结构可以不一样。

**核心操作**：在测试时，将训练时的多分支模型进行合并得到一条单线路模型，即将 $1 \times 1$ 卷积 + BN（批标准化）与 $3 \times 3$ 卷积进行合并。详见下图。[RepVGG网络：结构重参数化 - 详细过程](#)

- (1) 将 $1 \times 1$ 卷积转换成 $3 \times 3$ 卷积=
- (2) 将BN和 $3 \times 3$ 卷积进行融合，转换成 $3 \times 3$ 卷积=
- (3) 多分支融合

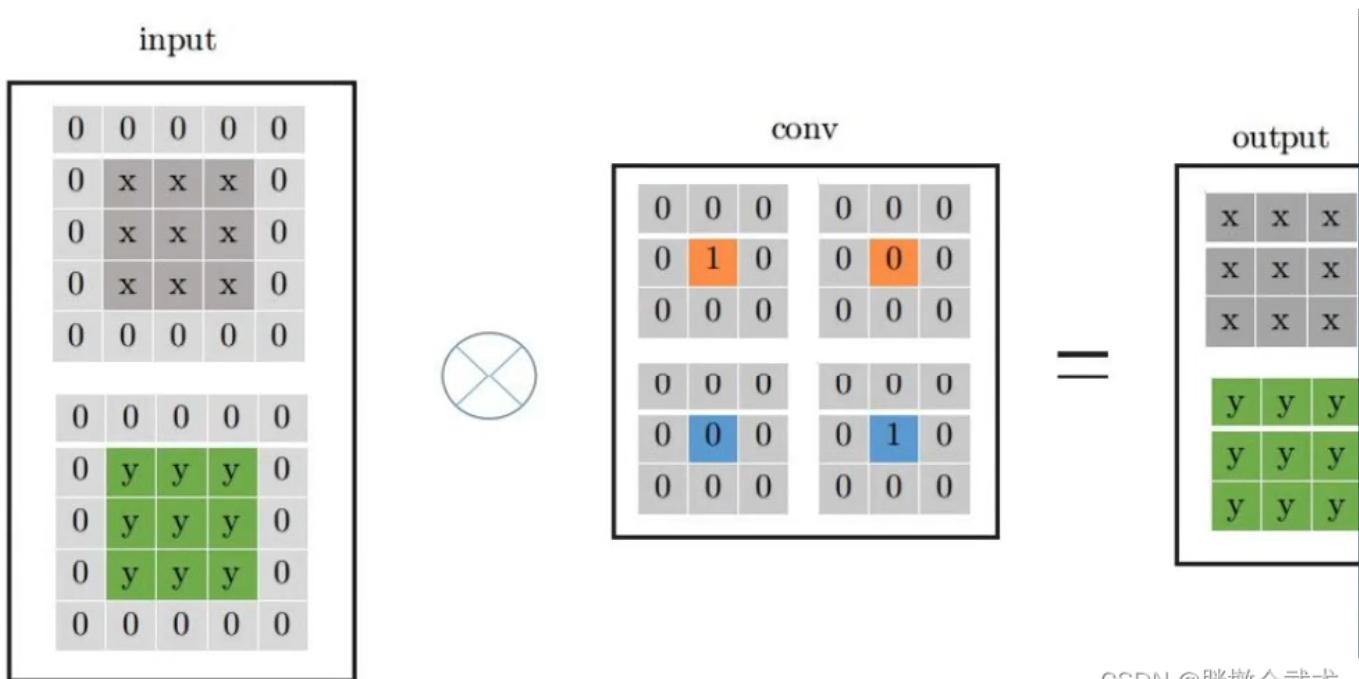
**备注1** yolo的核心是检测速度快，而不是检测精度高。 **备注2** 在前六个版本的优化后，网络层只留下了 $3 \times 3$ 卷积、 $1 \times 1$ 卷积和BN（每一个网络层之后都进行批标准化）。 **备注3** VGG在2014年告诉我们， $3 \times 3$ 卷积是计算速度最快的，优化最好的



**备注4：**黄色模块是激活函数ReLU，蓝色模块是卷积层。**备注5：**单支路模型可以节约内存。

### 将 $1 \times 1$ 卷积转换成 $3 \times 3$ 卷积

**具体过程：** (1) 取 $1 \times 1$ 卷积（卷积核：1个参数），设置padding=1（即在其周围填补一圈全为零的元素） (2) 设置原始输入的padding=1 (3) 输入与卷积核进行卷积操作，得到 $3 \times 3$ 的卷积层。注意：原始输入和 $1 \times 1$ 卷积都需要设置padding



### 将BN和 $3 \times 3$ 卷积进行融合，转换成 $3 \times 3$ 卷积

CSDN @胖墩会武术

## ✓ RepConv-BN计算拆解

一个batch数据,  $x_1, x_2, \dots, x_n$  对他们来进行归一化操作:

$$\hat{x}_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad \hat{x}_i = \frac{\gamma x_i}{\sqrt{\sigma^2 + \epsilon}} + \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \epsilon}}$$

其中:  $\gamma$  (平移缩放系数) 和  $\beta$  (缩放系数) 是可学习参数;  
 $\epsilon$  是防止除0异常;  
均值  $\mu$  和标准差  $\sigma$  每个channel单独计算。



$y = a * x + b$

$$\begin{pmatrix} \hat{F}_{1,i,j} \\ \hat{F}_{2,i,j} \\ \vdots \\ \hat{F}_{C-1,i,j} \\ \hat{F}_{C,i,j} \end{pmatrix} = \begin{pmatrix} \frac{\gamma_1}{\sqrt{\hat{\sigma}_1^2 + \epsilon}} & 0 & \cdots & 0 \\ 0 & \frac{\gamma_2}{\sqrt{\hat{\sigma}_2^2 + \epsilon}} & & \\ \vdots & \ddots & & \vdots \\ & & \frac{\gamma_{C-1}}{\sqrt{\hat{\sigma}_{C-1}^2 + \epsilon}} & 0 \\ 0 & \cdots & 0 & \frac{\gamma_C}{\sqrt{\hat{\sigma}_C^2 + \epsilon}} \end{pmatrix} \cdot \begin{pmatrix} F_{1,i,j} \\ F_{2,i,j} \\ \vdots \\ F_{C-1,i,j} \\ F_{C,i,j} \end{pmatrix} + \begin{pmatrix} \beta_1 - \gamma_1 \frac{\hat{\mu}_1}{\sqrt{\hat{\sigma}_1^2 + \epsilon}} \\ \beta_2 - \gamma_2 \frac{\hat{\mu}_2}{\sqrt{\hat{\sigma}_2^2 + \epsilon}} \\ \vdots \\ \beta_{C-1} - \gamma_{C-1} \frac{\hat{\mu}_{C-1}}{\sqrt{\hat{\sigma}_{C-1}^2 + \epsilon}} \\ \beta_C - \gamma_C \frac{\hat{\mu}_C}{\sqrt{\hat{\sigma}_C^2 + \epsilon}} \end{pmatrix}$$

CSDN @胖墩会武术

## ✓ RepConv-BN计算拆解, 更新后的卷积核:

- filter weights:  $\mathbf{W} = \mathbf{W}_{BN} \cdot \mathbf{W}_{conv}$
- bias:  $\mathbf{b} = \mathbf{W}_{BN} \cdot \mathbf{b}_{conv} + \mathbf{b}_{BN}$

合并方法 (用一个卷积代替原来的卷积+BN)

Let,  $\mathbf{W}_{BN} \in \mathbb{R}^{C \times C}$  and  $\mathbf{b}_{BN} \in \mathbb{R}^C$  - are parameters of the BN

$\mathbf{W}_{conv} \in \mathbb{R}^{C \times (C_{prev}k^2)}$  and  $\mathbf{b}_{conv} \in \mathbb{R}^C$  - are parameters of the Convolutional layer that precede BN

$F_{prev}$  - input to the convolutional

$C_{prev}$  - the number of channels of the input layer

$k$  - is the filter size.

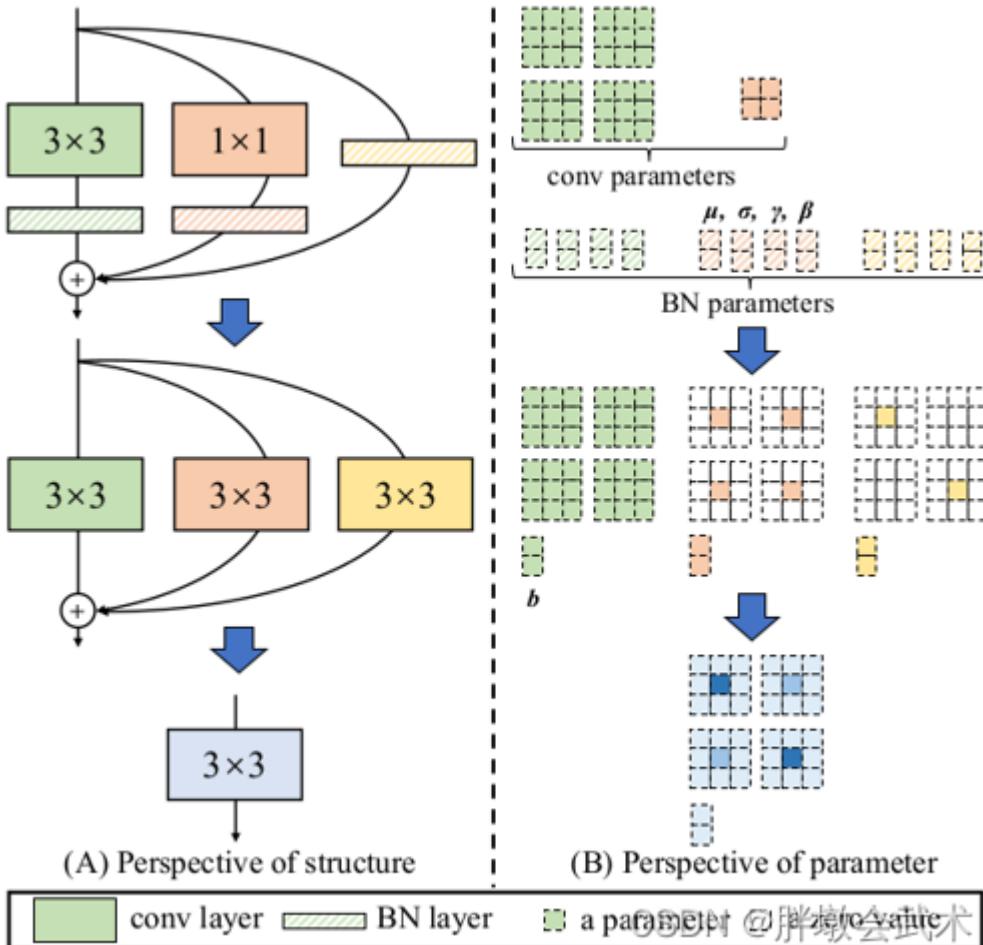
$k \times k$  part of  $F_{prev}$  reshaped into a  $k^2 \cdot C_{prev}$  vector  $\mathbf{f}_{i,j}$ , so the resulting formula will be:

$$\hat{\mathbf{f}}_{i,j} = \mathbf{W}_{BN} \cdot (\mathbf{W}_{conv} \cdot \mathbf{f}_{i,j} + \mathbf{b}_{conv}) + \mathbf{b}_{BN}$$

CSDN @胖墩会武术

## 多分支融合

具体过程 : = (1) 将1x1卷积 + BN全部转换为3x3卷积, 然后与3x3卷积进行合并, 得到一个3x3卷积。=



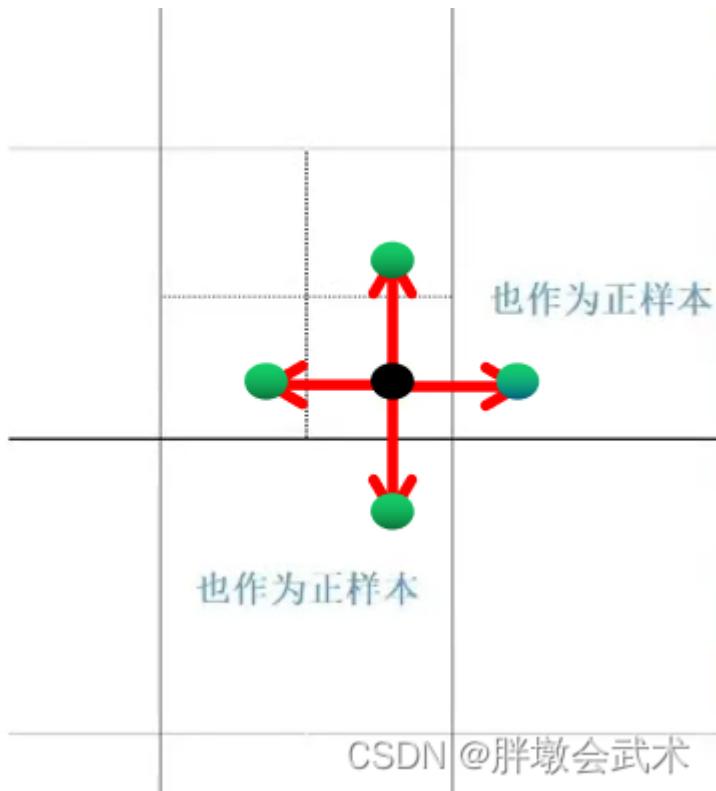
### 正样本分配策略

主要目的：为了得到更多的正样本。=正样本即先验框（anchor），负样本即背景。=

具体计算过程分两个步骤：（1）提取anchor；（2）筛选anchor。

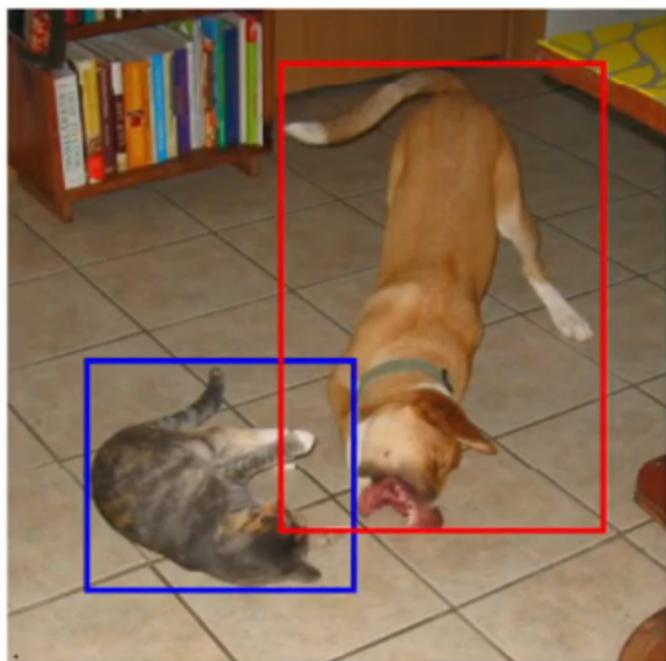
### 具体过程（提取anchor）：

- = (1) 计算先验框的中心点位置=
- = (2) 在当前网格中进行上、下、左、右四个方向的位置偏移，偏移大小为 0.5。=
- = (3) 最后取当前网格 + 四个方向的中心点所对应的除当前网格的二个网格。共三个网格作为正样本=

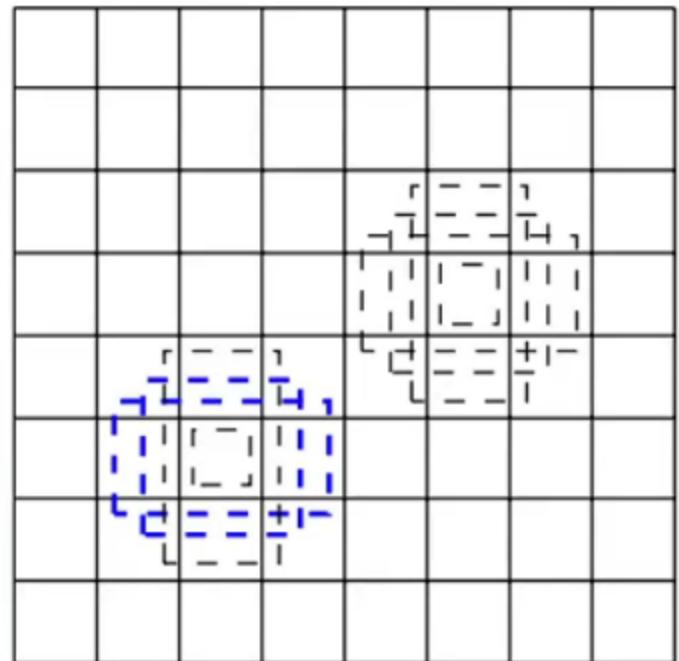


**具体过程（筛选anchor）**：提取满足要求的anchor，去掉匹配度低的anchor（该类anchor无意义）

条件一：候选框和先验框（anchor）的长宽比范围：[0.25, 4]。条件二：候选框和先验框（anchor）的IOU要大于自定义阈值。条件三：候选框和先验框（anchor）的类别预测损失要大于自定义阈值。条件四：将以上三个条件进行权重相加，并进行损失排名。 $loss = (\text{权重系数1} * \text{条件一}) + (\text{权重系数2} * \text{条件二}) + (\text{权重系数3} * \text{条件三})$



(a) Image with GT boxes



(b)  $8 \times 8$  feature map

举例：以下是具体过程（筛选anchor）中，条件二的损失计算。

## ✓ 正样本分配之IOU损失计算：

📝 例如当前输入正样本3个，候选框13个则，可以得到[3,13]矩阵

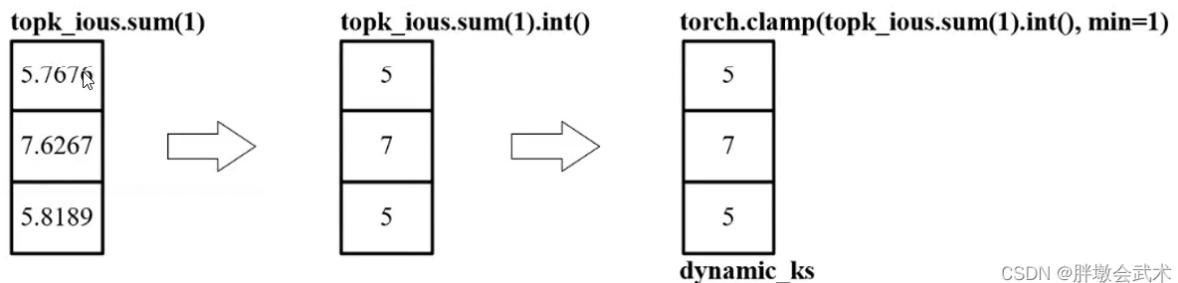
0.9324	0.5090	0.9462	0.2005	0.5925	0.2649	0.6287	0.2600	0.8671	0.1942	0.0697	0.2821	0.4847
0.0783	0.5240	0.8098	0.8982	0.7448	0.3034	0.8672	0.9714	0.6882	0.2981	0.7148	0.4133	0.9950
0.4791	0.5816	0.2205	0.8987	0.4111	0.8098	0.3369	0.8633	0.2661	0.6773	0.3269	0.3906	0.3705

📝 接下来TOPK，多了是10个，要是不够就会更少，得到[3,10]矩阵

0.9462	0.9324	0.8671	0.6287	0.5925	0.5090	0.4847	0.2821	0.2649	0.2600
0.9950	0.9714	0.8982	0.8672	0.8098	0.7448	0.7148	0.6882	0.5240	0.4133
0.8987	0.8633	0.8098	0.6773	0.5816	0.4791	0.4111	0.3906	0.3705	0.3369

CSDN @胖墩会武术

📝 得到每一个GT所对应候选框数量，最少也得一个



CSDN @胖墩会武术

- 备注1：计算=真实框 (Ground Truth, GT) = 对应的候选框数量 (损失计算得到的结果)：向下取整。
- 备注2：若一个候选框同时和多个anchor高度匹配，则按照损失计算原则，只能匹配损失最小对应的一个anchor。

相对偏移量计算 (yolov5/v7版)

## ✓ 相对偏移量计算

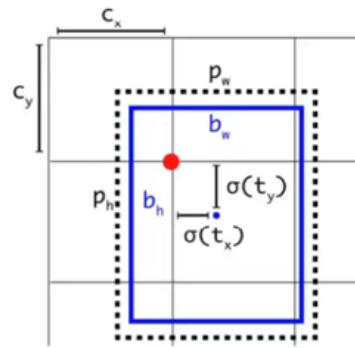
$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

💡 V3V4要预测的结果:  $b_w = p_w e^{t_w}$   
 $b_h = p_h e^{t_h}$

💡 (黑色:anchor, 蓝色:gt)

$b_x = 2\sigma(t_x) - 0.5 + c_x$
$b_y = 2\sigma(t_y) - 0.5 + c_y$
$b_w = p_w(2\sigma(t_w))^2$
$b_h = p_h(2\sigma(t_h))^2$



(Cx, Cy): 表示当前网格的左上角位置坐标。

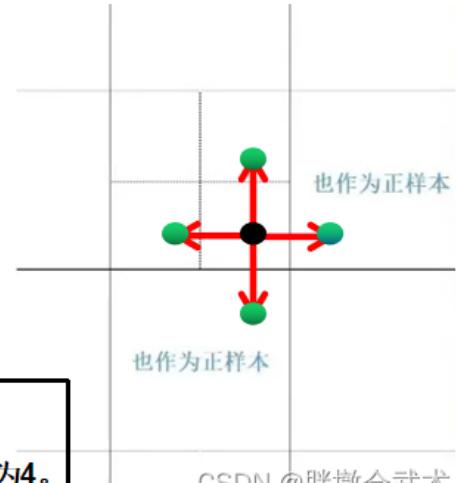
(tx, ty, tw, th): 表示预测的结果在当前网格相对位置的偏移量。

$\sigma(tx)$ : 表示对漂移量 tx 取sigmoid函数, 得到 (0~1) 之间的值。

参数	值范围
$\sigma$	[0,1]
$2\sigma-0.5$	[0.5,1.5]
$(2\sigma)^2$	[0,4]

[0.5,1.5]表示上下/左右的最大移动范围。

[0,4]表示候选框和先验框 (anchor) 的长宽比最大为4。



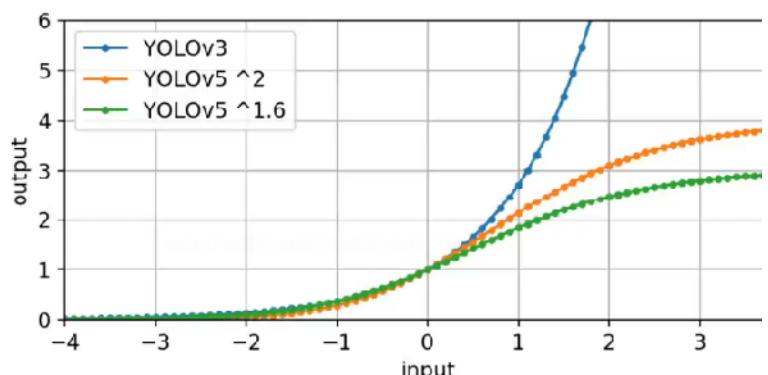
CSDN @胖墩会武术

## ✓ 预测值套sigmoid的原因

💡 V3梯度直接炸了

💡 V5平方后取值范围0-4

💡 也就是倍率不能差异太多



💡 选正样本的时候也是要选符合一定范围倍率的 (0.25-4)

CSDN @胖墩会武术

辅助头 (auxiliary head) + 主头 (lead head)

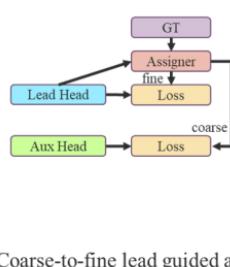
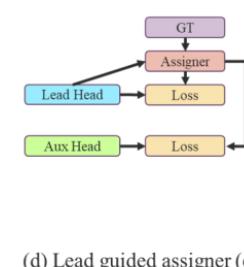
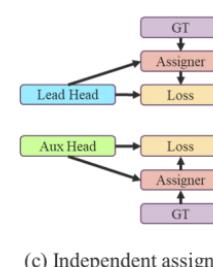
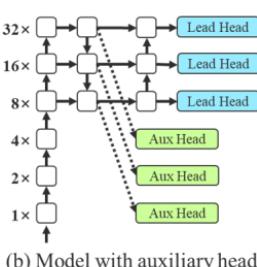
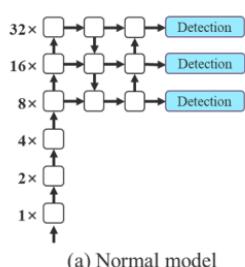


Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be elaborated in Appendix.

CSDN @胖墩会武术

图5:辅助用粗, 头部用细。与常规模型(a)相比, (b)模式具有辅助头。与通常的独立标签分配器©不同, 我们提出(d)铅头引导标签分配器和(e)粗至细铅头引导标签分配器。该标签分配器通过前导头预测和地面真实值进行优化, 同时得到训练前导头和辅助头的标签。详细的从粗到细的实现方法和约束设计细节将在附录中详细阐述。

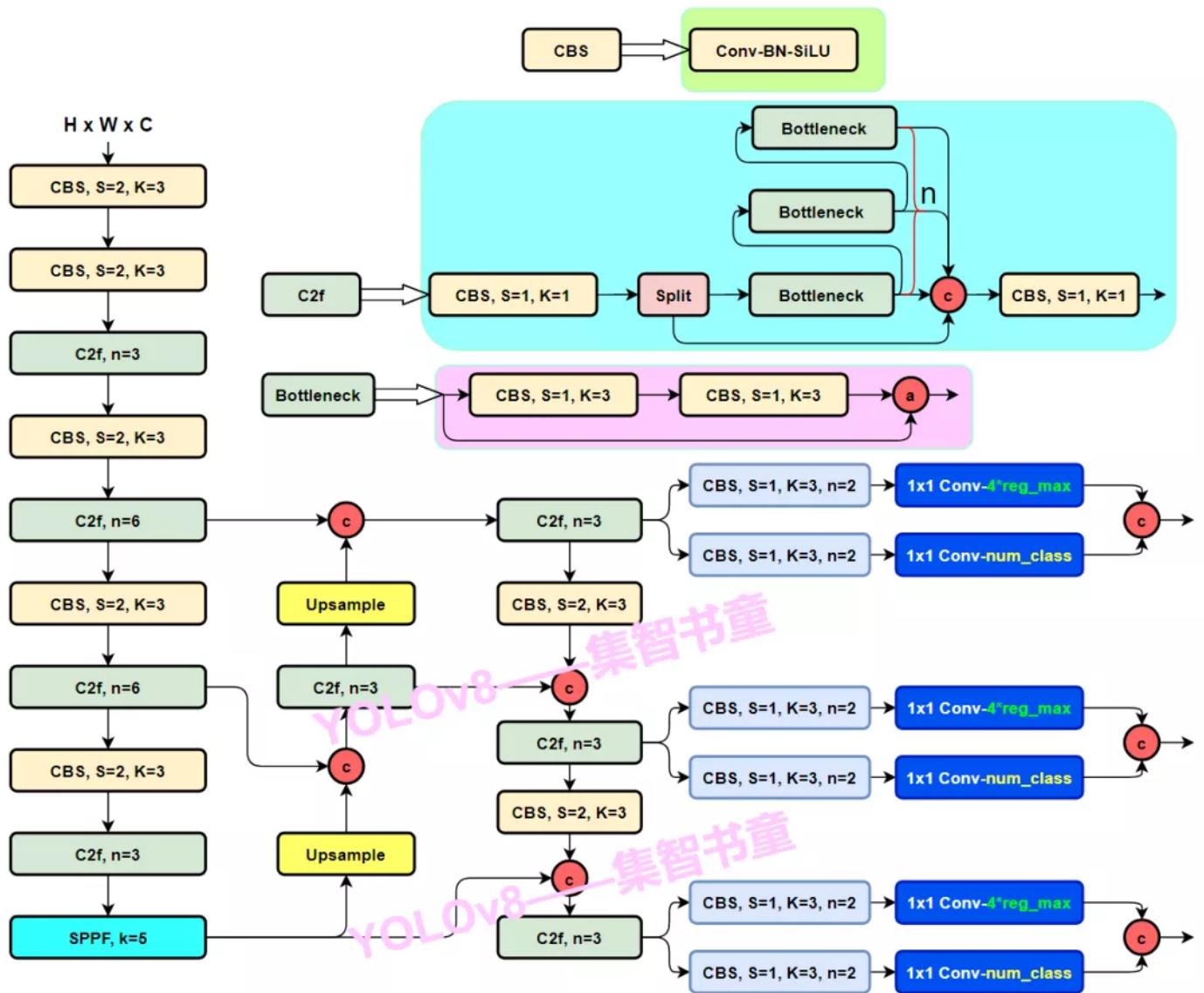
# YOLOv8

## 算法改进

具体改进如下:

- Backbone: 使用的依旧是CSP的思想, 不过YOLOv5中的C3模块被替换成C2f模块, 实现了进一步的轻量化, 同时YOLOv8依旧使用了YOLOv5等架构中使用的SPPF模块
- PAN-FPN: YOLOv8依旧使用了PAN的思想, 不过通过对YOLOv5与YOLOv8的结构图可以看到, YOLOv8将YOLOv5中PAN-FPN上采样阶段中的CBS 1\*1的卷积结构删除了, 同时也将C3模块替换为了C2f模块;
- Decoupled-Head: YOLOv8使用了Decoupled-Head;
- Anchor-Free: YOLOv8抛弃了以往的Anchor-Base, 使用了Anchor-Free的思想;
- 损失函数: YOLOv8使用VFL Loss作为分类损失, 使用DFL Loss+CIOU Loss作为分类损失
- 样本匹配: YOLOv8抛弃了以往的IOU匹配或者单边比例的分配方式, 而是使用了Task-Aligned Assigner匹配方式。

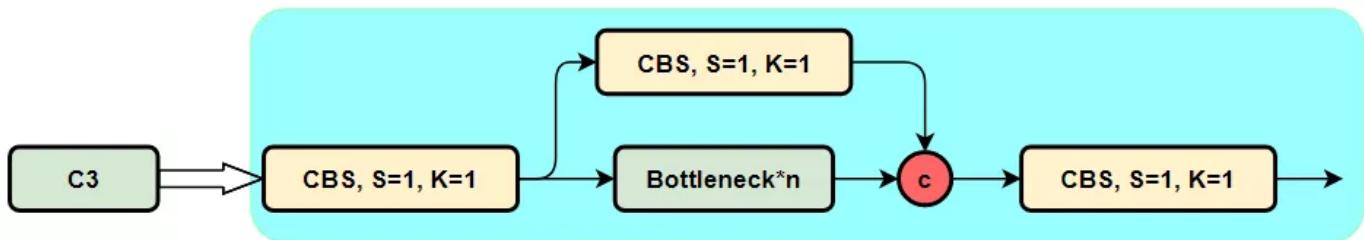
## 网络模型



## 1.C3和C2F

C3模块，其主要是借助CSPNet提取分流的思想，同时结合残差结构的思想，设计了所谓的C3 Block，这里的CSP主分支梯度模块为BottleNeck模块，也就是所谓的残差模块。同时堆叠的个数由参数n来进行控制，也就是说不同规模的模型，n的值是有变化的。

其实这里的梯度流主分支，可以是任何之前你学习过的模块，比如，美团提出的YOLOv6中就是用来重参模块RepVGGBlock来替换BottleNeck Block来作为主要的梯度流分支，而百度提出的PP-YOLOE则是使用了RepResNet-Block来替换BottleNeck Block来作为主要的梯度流分支。而YOLOv7则是使用了ELAN Block来替换BottleNeck Block来作为主要的梯度流分支



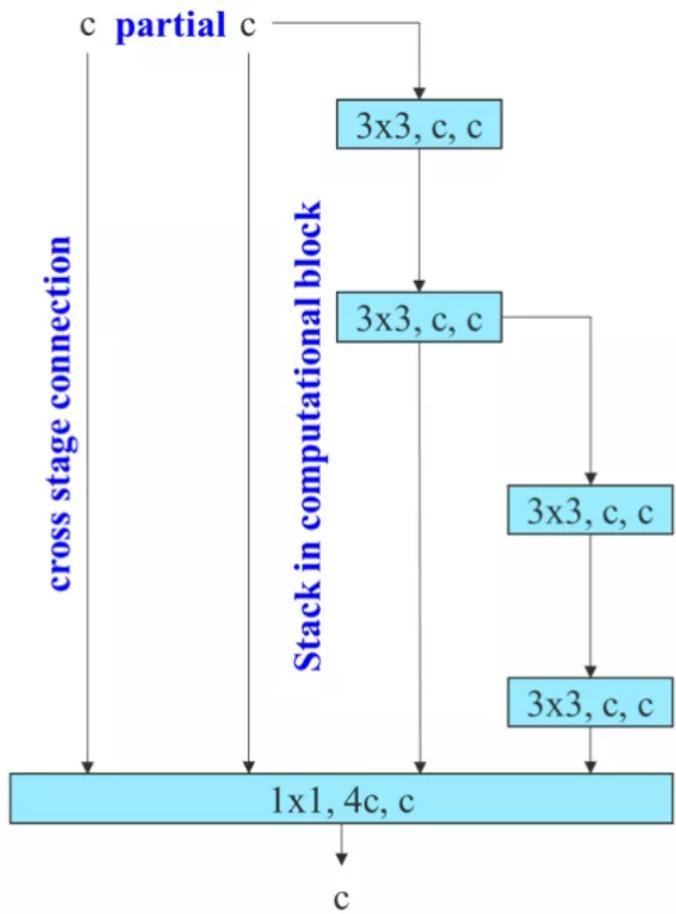
```

class C3(nn.Module):
    # CSP Bottleneck with 3 convolutions
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in,
    ch_out, number, shortcut, groups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c1, c_, 1, 1)
        self.cv3 = Conv(2 * c_, c2, 1)  # optional act=FReLU(c2)
        self.m = nn.Sequential(*[Bottleneck(c_, c_, shortcut, g, e=1.0) for
        _ in range(n)])
    def forward(self, x):
        return self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x)), 1))

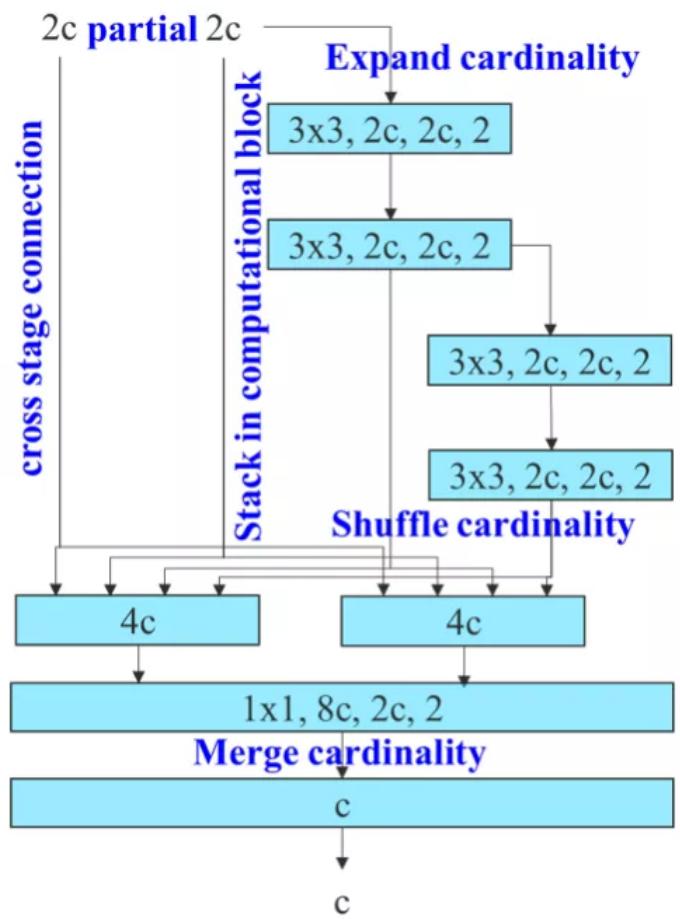
```

通过C3模块的代码以及结构图可以看到，C3模块和名字思路一致，在模块中使用了3个卷积模块（Conv+BN+SiLU），以及n个BottleNeck。通过C3代码可以看出，对于cv1卷积和cv2卷积的通道数是一致的，而cv3的输入通道数是前者的2倍，因为cv3的输入是由主梯度流分支（BottleNeck分支）依旧次梯度流分支（CBS，cv2分支）cat得到的，因此是2倍的通道数，而输出则是一样的。

不妨我们再看一下YOLOv7中的模块：



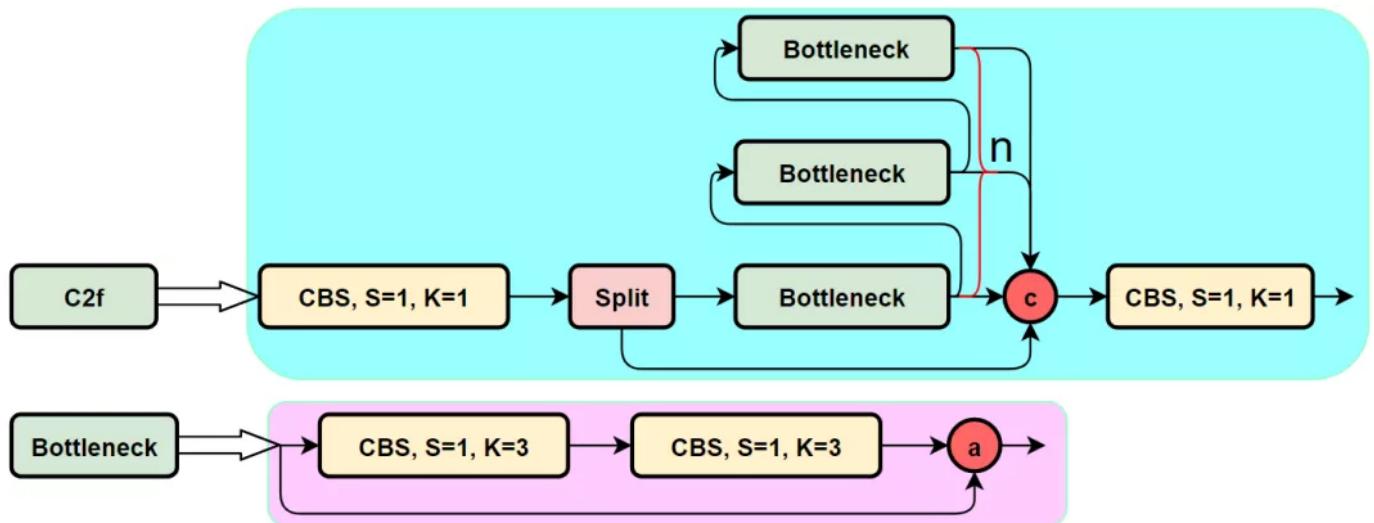
(c) ELAN [1]



(d) E-ELAN

YOLOv7通过并行更多的梯度流分支，放ELAN模块可以获得更丰富的梯度信息，进而或者更高的精度和更合理的延迟。

C2f模块的结构图如下：



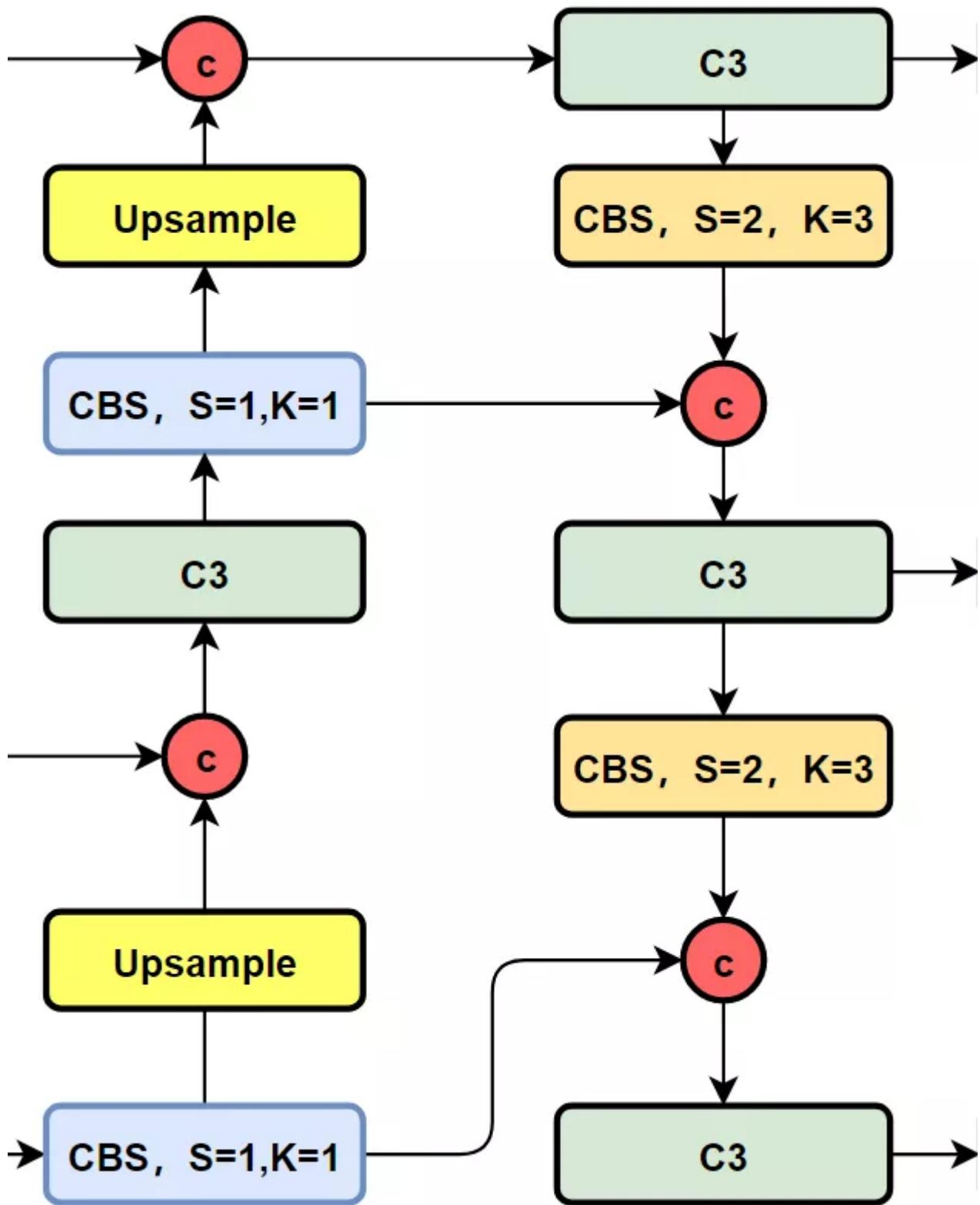
我们可以很容易的看出，C2f模块就是参考了C3模块以及ELAN的思想进行的设计，让YOLOv8可以在保证轻量化的同时获得更加丰富的梯度流信息

```
class C2f(nn.Module):
    # CSP Bottleneck with 2 convolutions
    def __init__(self, c1, c2, n=1, shortcut=False, g=1, e=0.5):  # ch_in,
        ch_out, number, shortcut, groups, expansion
        super().__init__()
        self.c = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, 2 * self.c, 1, 1)
        self.cv2 = Conv((2 + n) * self.c, c2, 1)  # optional act=FReLU(c2)
        self.m = nn.ModuleList(Bottleneck(self.c, self.c, shortcut, g, k=
        ((3, 3), (3, 3)), e=1.0) for _ in range(n))

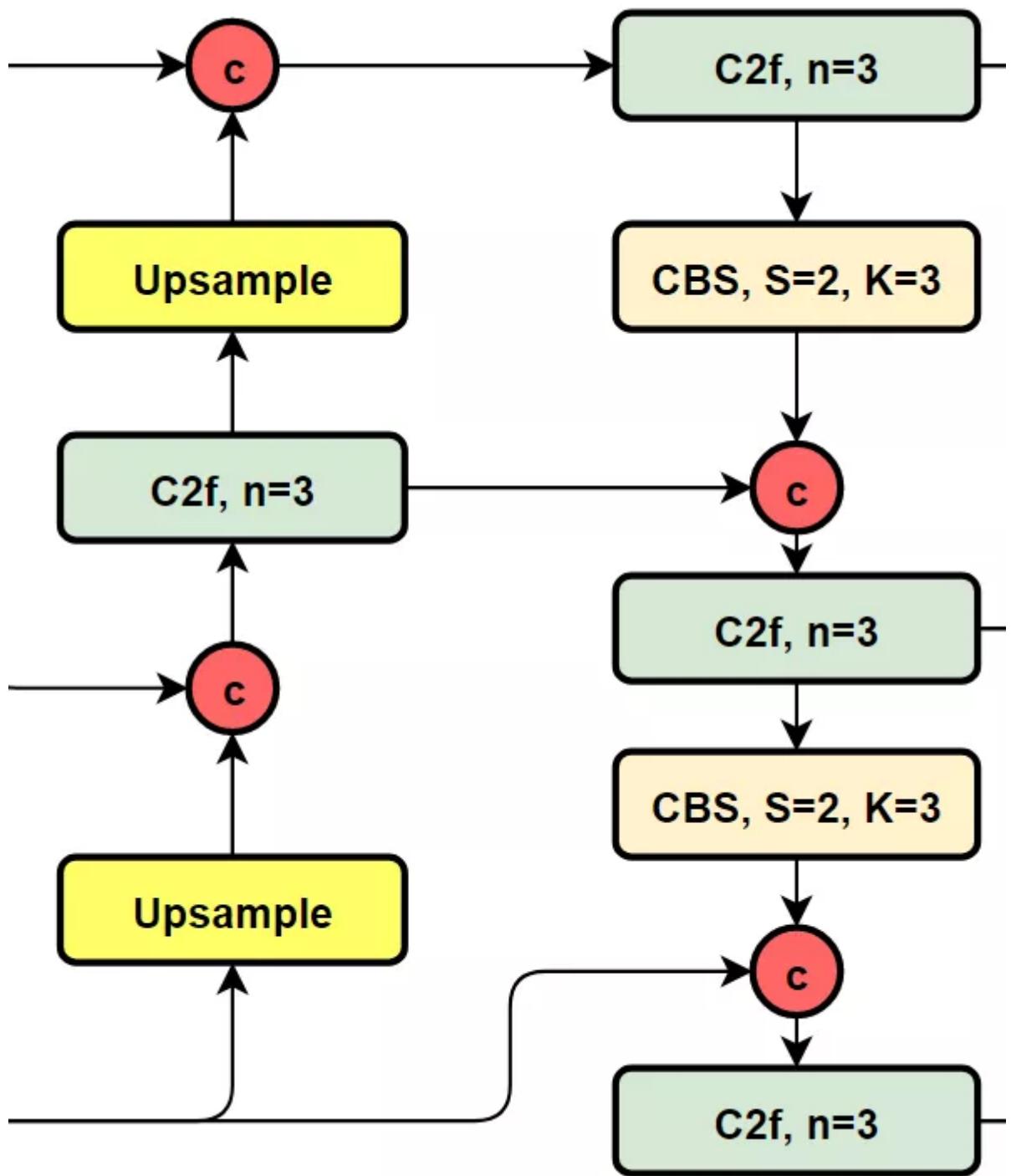
    def forward(self, x):
        y = list(self.cv1(x).split((self.c, self.c), 1))
        y.extend(m(y[-1]) for m in self.m)
        return self.cv2(torch.cat(y, 1))
```

## 2.PAN-FPN

YOLOv5的Neck部分的结构图如下：



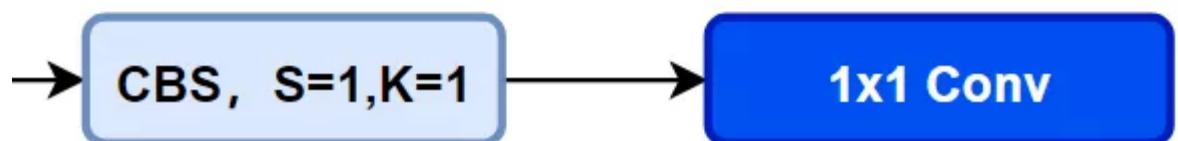
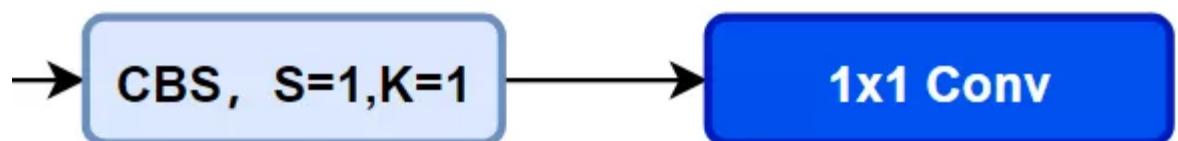
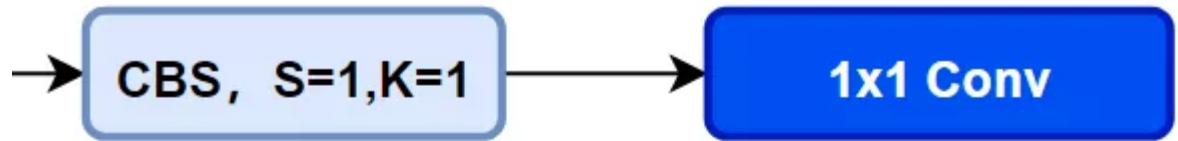
YOLOv8的Neck部分的结构图如下：



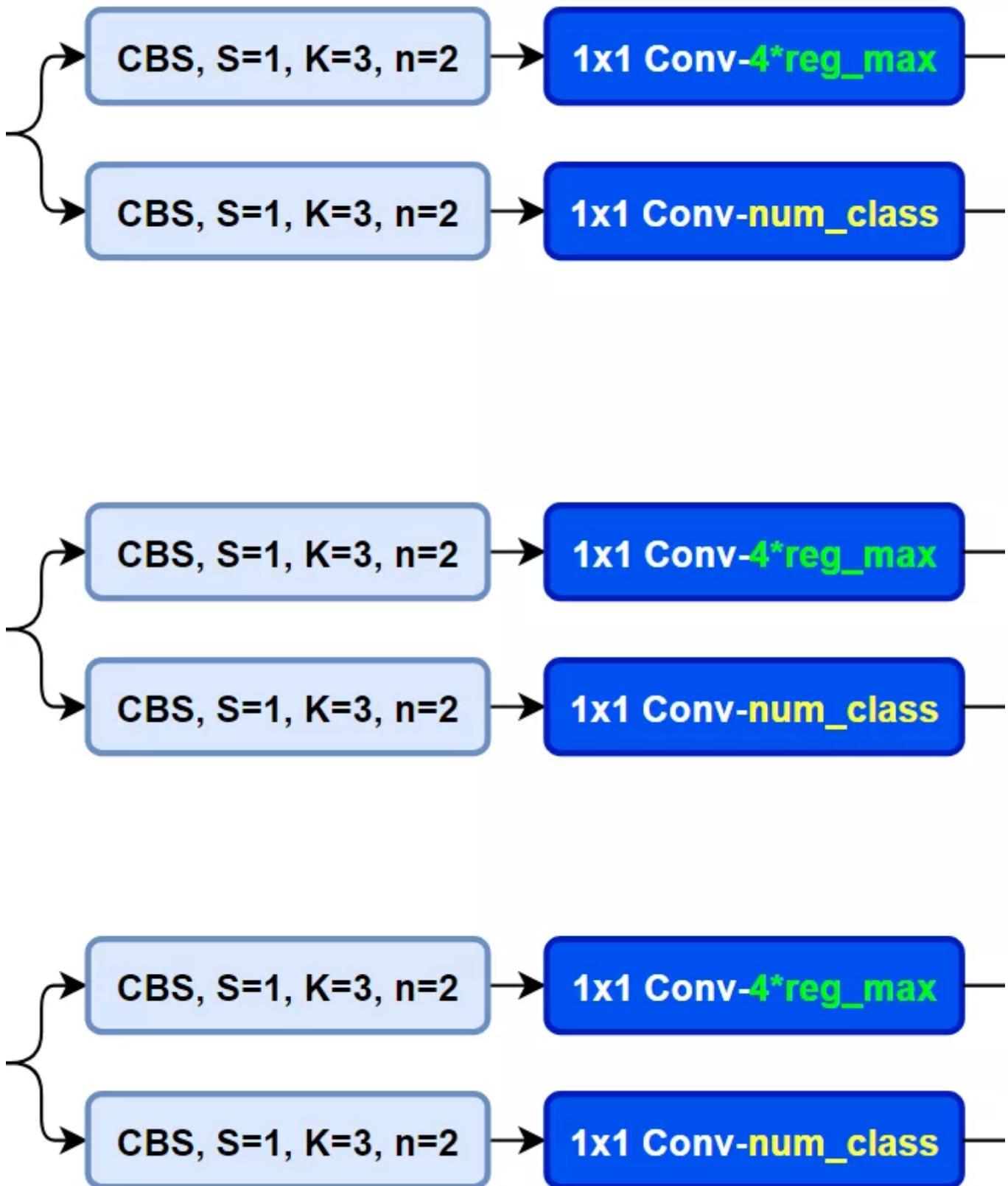
可以看到，相对于YOLOv5，YOLOv8将C3模块以及RepBlock替换为了C2f，同时细心可以发现，相对于YOLOv5，YOLOv8选择将上采样之前的 $1 \times 1$ 卷积去除了，将Backbone不同阶段输出的特征直接送入了上采样操作。

### 3.Head

YOLO v5的head的部分使用一个卷积同时做分类和回归 (Coupled-Head)



而YOLOv8则是参考了YOLOX和YOLOV6，使用了Decoupled-Head，即使用两个卷积分  
别做分类和回归，同时由于使用了DFL 的思想，因此回归头的通道数也变成了  
 $4 * \text{reg\_max}$ 的形式：



从配置文件上看，YOLO v8相当于对代码做了优化，在下采样32倍时，通道数不加倍。与16倍的通道数相同，上采样做拼接时，不使用 $1 \times 1$ 的卷积调整通道数，此外，将通道数的调整放入下采样和c2f模块。

yolo v5配置文件（右）和yolo v8配置文件（左）

```
- [-1, 3, C2f, [128, True]]
- [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
- [-1, 6, C2f, [256, True]]
- [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
- [-1, 6, C2f, [512, True]]
- [-1, 1, Conv, [512, 3, 2]] # 7-P5/32
- [-1, 3, C2f, [512, True]]
- [-1, 1, SPPF, [512, 5]] # 9

# YOLOv8.0x head
head:
- [-1, 1, nn.Upsample, [None, 2, 'nearest']]
- [[-1, 6], 1, Concat, [1]] # cat backbone P4
- [-1, 3, C2f, [512]] # 13

- [-1, 1, nn.Upsample, [None, 2, 'nearest']]
- [[-1, 4], 1, Concat, [1]] # cat backbone P3
- [-1, 3, C2f, [256]] # 17 (P3/8-small)

- [-1, 1, Conv, [256, 3, 2]]
- [[-1, 12], 1, Concat, [1]] # cat head P4
- [-1, 3, C2f, [512]] # 20 (P4/16-medium)

- [-1, 1, Conv, [512, 3, 2]]
- [[-1, 9], 1, Concat, [1]] # cat head P5
- [-1, 3, C2f, [512]] # 23 (P5/32-large)

- [[15, 18, 21], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

```

[-1, 6, C3, [256]],
[-1, 1, Conv, [512, 3, 2]], # 5-P4/16
[-1, 9, C3, [512]],
[-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
[-1, 3, C3, [1024]],
[-1, 1, SPPF, [1024, 5]], # 9
]

# YOLOv5 v6.0 head
head:
[[[-1, 1, Conv, [512, 1, 1]],
[-1, 1, nn.Upsample, [None, 2, 'nearest']],
[[-1, 6], 1, Concat, [1]], # cat backbone P4
[-1, 3, C3, [512, False]], # 13

[-1, 1, Conv, [256, 1, 1]],
[-1, 1, nn.Upsample, [None, 2, 'nearest']],
[[-1, 4], 1, Concat, [1]], # cat backbone P3
[-1, 3, C3, [256, False]], # 17 (P3/8-small)

[-1, 1, Conv, [256, 3, 2]],
[[-1, 14], 1, Concat, [1]], # cat head P4
[-1, 3, C3, [512, False]], # 20 (P4/16-medium)

[-1, 1, Conv, [512, 3, 2]],
[[-1, 10], 1, Concat, [1]], # cat head P5
[-1, 3, C3, [1024, False]], # 23 (P5/32-large)

[[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)

```

#### 4. 损失函数

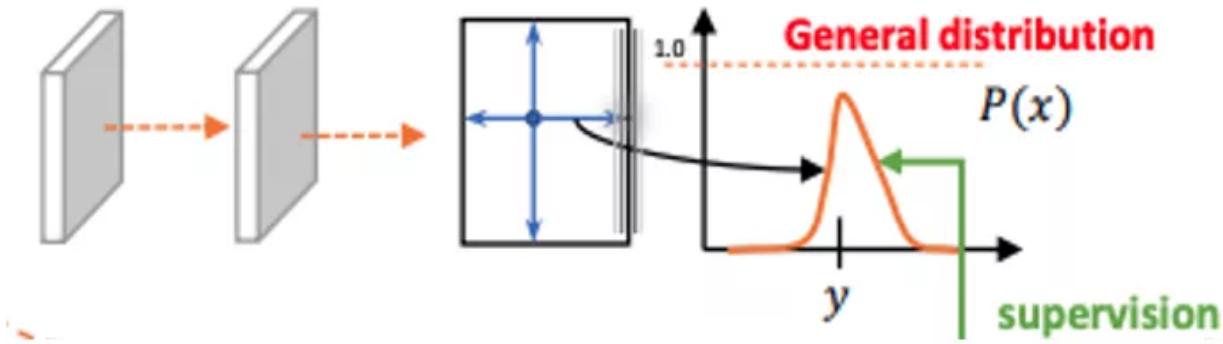
对于YOLOv8，其分类损失为VFL Loss，其回归损失为CIOU Loss+DFL的形式，这里Reg\_max默认为16。

VFL主要改进是提出了非对称的加权操作，FL和QFL都是对称的。而非对称加权的思想来源于论文PISA，该论文指出首先正负样本有不平衡问题，即使在正样本中也存在不等权问题，因为mAP的计算是主正样本。

$$\text{VFL}(p, q) = \begin{cases} -q(q\log(p) + (1-q)\log(1-p)) & q > 0 \\ -\alpha p^\gamma \log(1-p) & q = 0, \end{cases}$$

$q$ 是label，正样本时候 $q$ 为bbox和gt的IoU，负样本时候 $q=0$ ，当为正样本时候其实没有采用FL，而是普通的BCE，只不过多了一个自适应IoU加权，用于突出主样本。而为负样本时候就是标准的FL了。可以明显发现VFL比QFL更加简单，主要特点是正负样本非对称加权、突出正样本为主样本。

针对这里的DFL (Distribution Focal Loss)，其主要是将框的位置建模成一个 general distribution，让网络快速的聚焦于和目标位置距离近的位置的分布。



$$\mathbf{DFL}(\mathcal{S}_i, \mathcal{S}_{i+1}) = -((y_{i+1} - y) \log(\mathcal{S}_i) + (y - y_i) \log(\mathcal{S}_{i+1})).$$

DFL 能够让网络更快地聚焦于目标  $y$  附近的值，增大它们的概率；DFL的含义是以交叉熵的形式去优化与标签 $y$ 最接近的一左一右2个位置的概率，从而让网络更快的聚焦到目标位置的邻近区域的分布；也就是说学出来的分布理论上是在真实浮点坐标的附近，并且以线性插值的模式得到距离左右整数坐标的权重。

## 5. 样本的匹配

标签分配是目标检测非常重要的一环，在YOLOv5的早期版本中使用了MaxIOU作为标签分配方法。然而，在实践中发现直接使用边长比也可以达到一阿姨你的效果。而YOLOv8则是抛弃了Anchor-Base方法使用Anchor-Free方法，找到了一个替代边长比例的匹配方法，TaskAligned。

为与NMS搭配，训练样例的Anchor分配需要满足以下两个规则：正常对齐的Anchor应当可以预测高分类得分，同时具有精确定位；不对齐的Anchor应当具有低分类得分，并在NMS阶段被抑制。基于上述两个目标TaskAligned设计了一个新的Anchor alignment metric 来在Anchor level 衡量Task-Alignment的水平。并且，Alignment metric 被集成在了 sample 分配和 loss function 里来动态的优化每个 Anchor 的预测。

**Anchor alignment metric:** 分类得分和 IoU 表示了这两个任务的预测效果，所以，TaskAligned 使用分类得分和 IoU 的高阶组合来衡量 Task-Alignment 的程度。使用下列的方式来对每个实例计算 Anchor-level 的对齐程度：

$$t = s^\alpha \times u^\beta$$

s 和 u 分别为分类得分和 IoU 值， $\alpha$  和  $\beta$  为权重超参。从上边的公式可以看出来，t 可以同时控制分类得分和 IoU 的优化来实现 Task-Alignment，可以引导网络动态的关注于高质量的 Anchor。

**Training sample Assignment:** 为提升两个任务的对齐性，TOOD 聚焦于 Task-Alignment Anchor，采用一种简单的分配规则选择训练样本：对每个实例，选择 m 个具有最大 t 值的 Anchor 作为正样本，选择其余的 Anchor 作为负样本。然后，通过损失函数(针对分类与定位的对齐而设计的损失函数)进行训练。

## 参考资源

- YOLO 系列总结：[YOLOv1](#), [YOLOv2](#), [YOLOv3](#), [YOLOv4](#), [YOLOv5](#), [YOLOX](#)
- YOLO 系列详解：[YOLOv1](#)、[YOLOv2](#)、[YOLOv3](#)、[YOLOv4](#)、[YOLOv5](#)、[YOLOv6](#)、[YOLOv7](#)
- 三万字硬核详解：[yolov1](#)、[yolov2](#)、[yolov3](#)、[yolov4](#)、[yolov5](#)、[yolov7](#)
- YOLO v8 详解