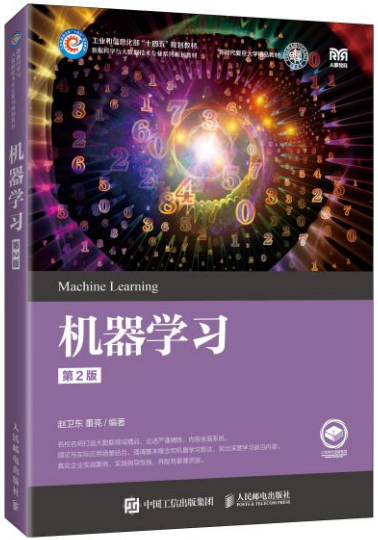


目标检测YOLOv4-v5

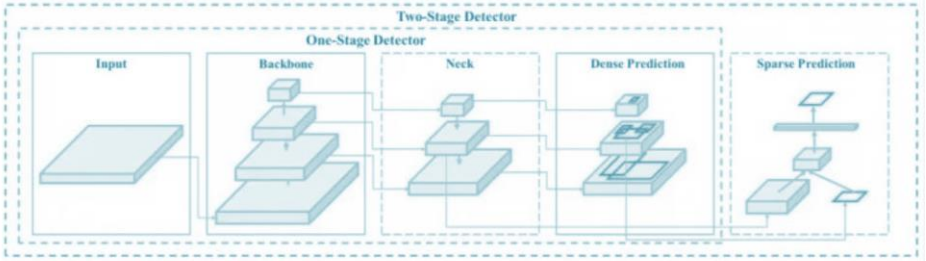
复旦大学 赵卫东

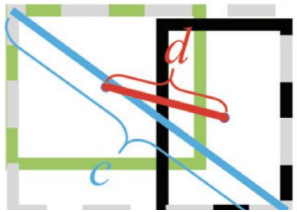
wdzhao@fudan.edu.cn

YOLOv4算法

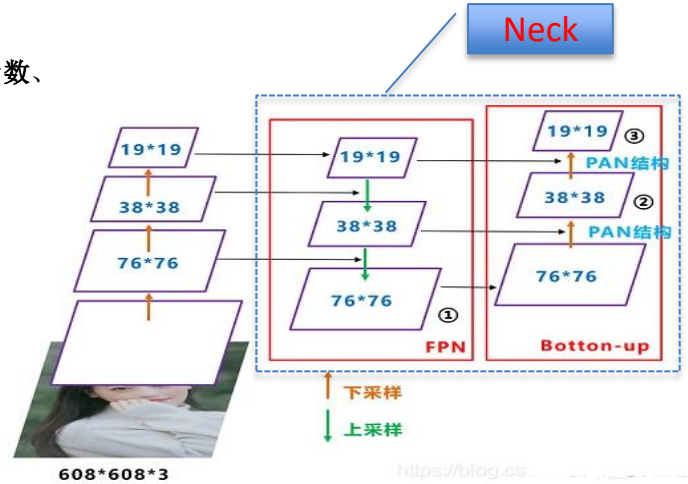
- YOLOv4算法输入端采用**mosaic数据增强**，Backbone上采用了**CSPDarknet53**、**Mish激活函数**、**Dropblock正则化**等方式，Neck中采用了**SPP**、**FPN+PAN**的结构
- YOLO v4算法输出端则采用**CIOU_Loss**、**DIOU_nms**操作。
- 使用 Darknet 架构的 YOLOv4 只有 244 MB。

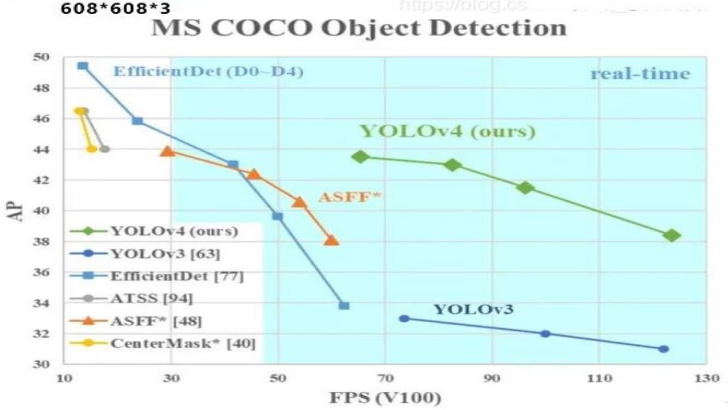
堆料





- DIoU Loss使用两个框的欧氏距离作为惩罚项
- CIoU Loss进一步在惩罚项中加入了矩形框的相对比例

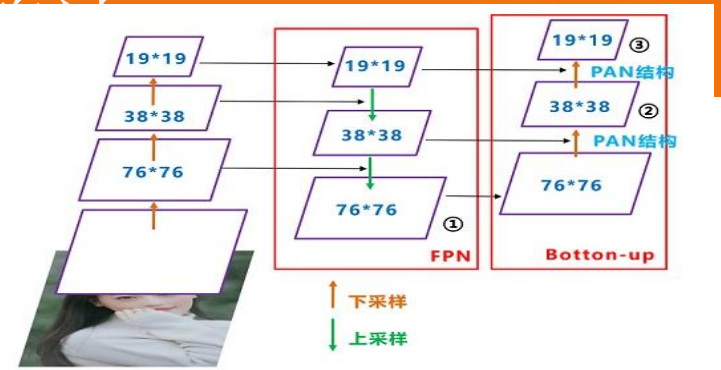


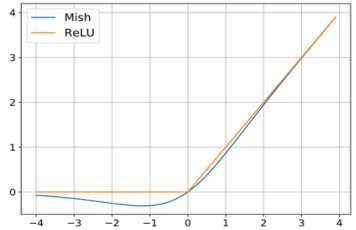


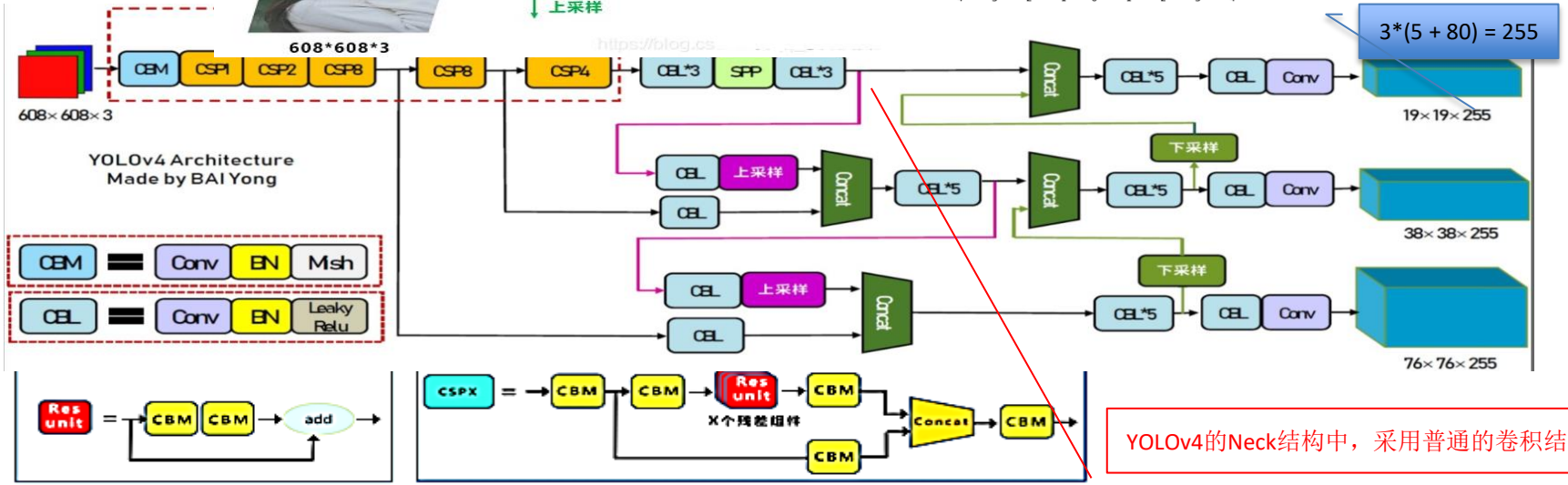
YOLOv4

- Backbone : CSPDarkn
- Neck : SPP , PAN
- Head : YOLOv3

YOLOv4 = CSPDarknet







YOLOv4的Neck结构中，采用普通的卷积结构

CIOU_Loss

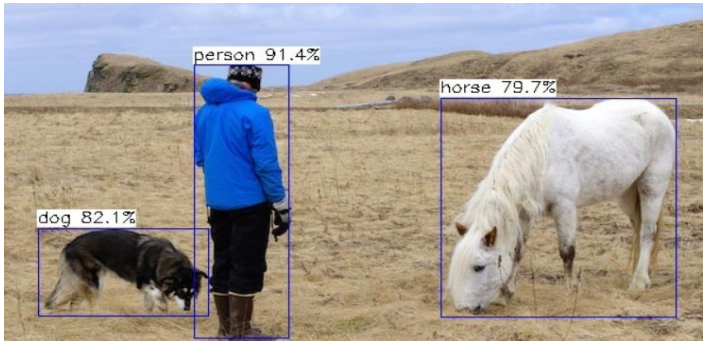
- 目标检测损失函数的演变过程如下：IOU --> GIOU --> DIOU -->CIOU损失函数，依次有所提升。
- 考虑三种几何参数：重叠面积、中心点距离、长宽比

GIOU损失函数：

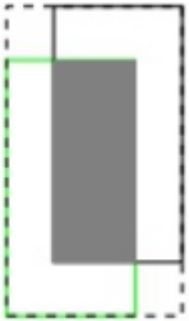
$$GIoU = IoU - \frac{A^c - u}{A^c}$$

$$-1 \le GIoU \le 1$$

A^c表示A和B的区域用矩形框框起来的面积，u表示A和B的并集。

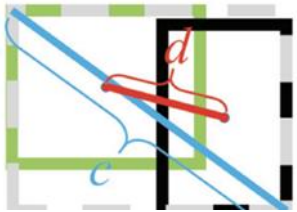


*CIoU*考虑三个参数：重叠面积、中心距离、长宽比



$$DIoU = IoU - \frac{\rho^2(b, b^{gt})}{c^2}$$

- ✓ *-1≤DIoU ≤1*
- ✓ *b*、b^{gt}表示预测框和GT的中心点
- ✓ ρ^2计算两个中心点的欧氏距离
- ✓ *c*包含真实框和预测框的包闭区域的对角线距离。



$$L_{DIoU} = 1 - DIoU$$

比GIoU loss收敛快得多

$$CIoU = IoU - \left(\frac{\rho^2(b, b^{gt})}{c^2}\right) - \alpha v$$

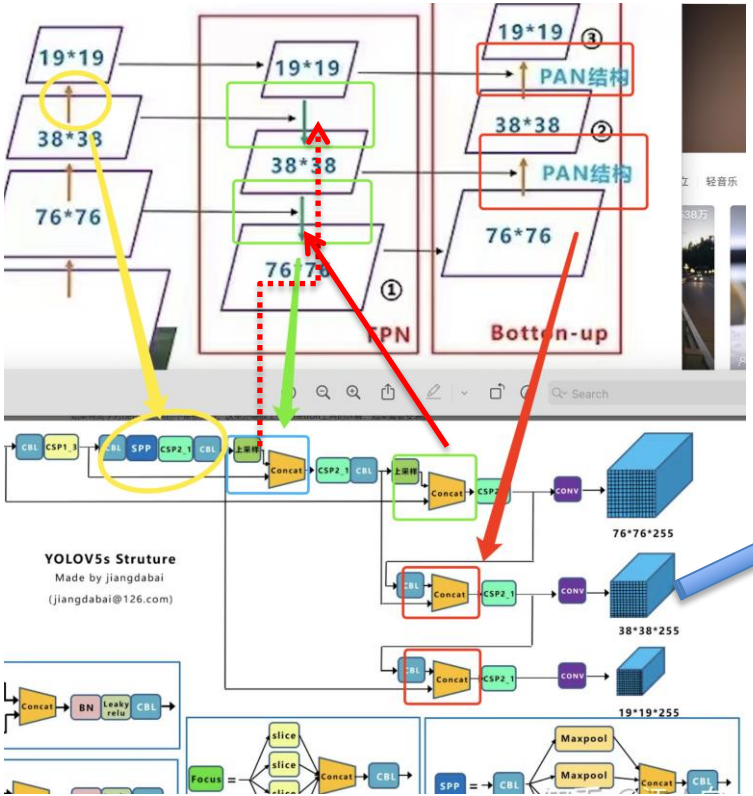
$$v = \frac{4}{\pi^2}\left(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h}\right)^2$$

$$\alpha = \frac{v}{(1 - IoU) + v}$$

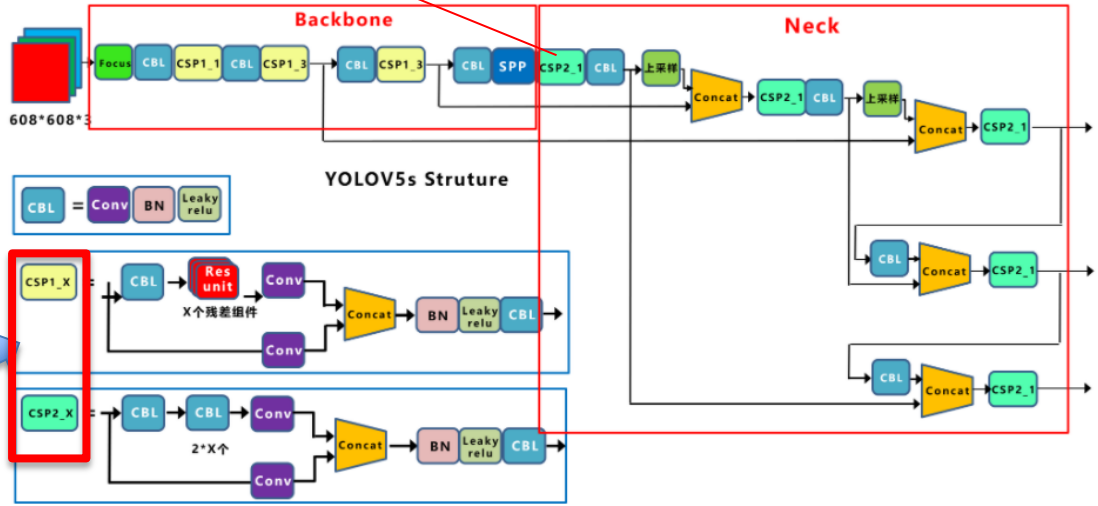
其中α是权重参数，*v*度量长宽比的相似性。基于DIoU提升回归精确度。

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v$$

PAN结构

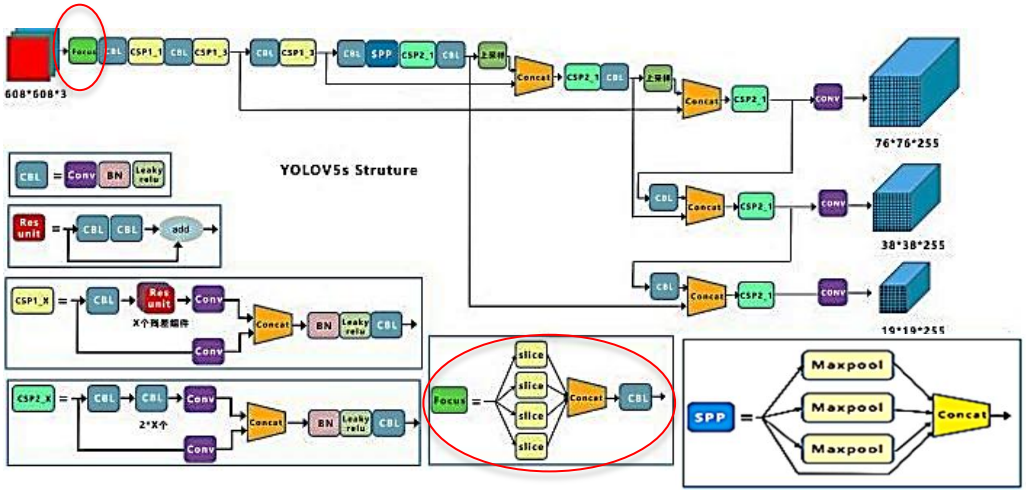


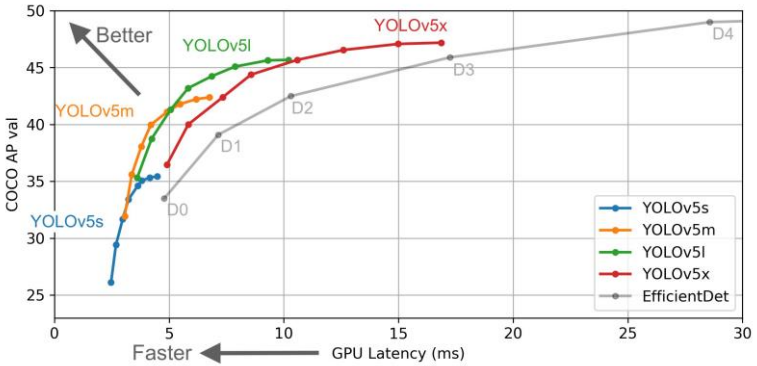
YOLOv5的Neck结构中，采用CSPnet设计的CSP2结构



YOLOv5算法

- YOLOv5算法由输入端、Backbone、Neck、Prediction等部分组成。
- 2020 年 6 月发布



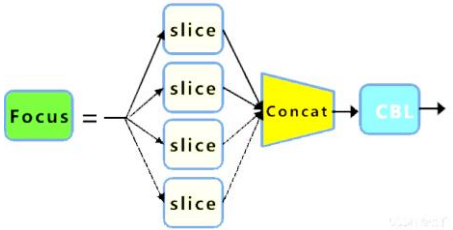


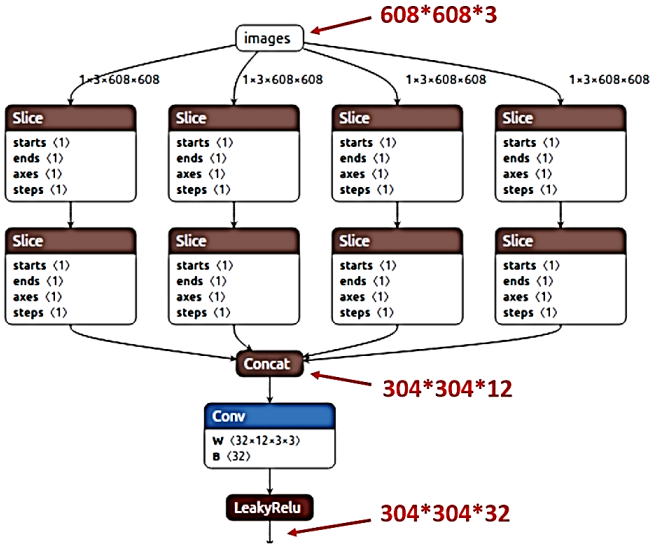
https://github.com/ultralytics/yolov5

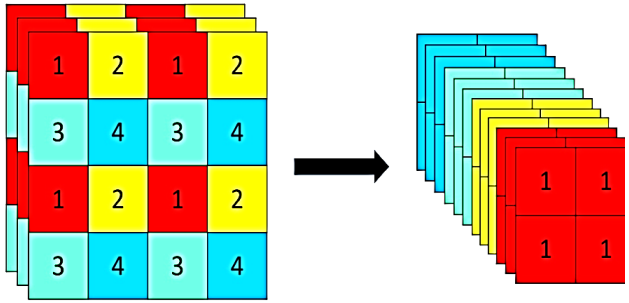
https://www.kaggle.com/c/global-wheat-detection

Focus结构

- 原始608×608×3的图像输入Focus结构，采用切片操作，先变成304×304×12的特征图，再经过一次32个卷积核的卷积操作，最终变成304×304×32的特征图。
- 启发于**YOLOv2中的PassThrough**，减少下采样带来的信息损失。



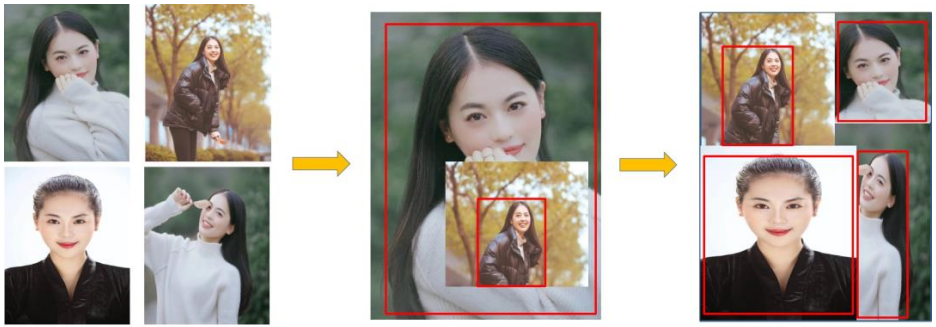




切片操作

Mosaic数据增强

- Mosaic数据增强是参考CutMix数据增强的方式，采用了4张图片进行随机缩放、随机裁剪、随机排布的方式进行拼接，提升小物体检测性能。
- 随机缩放增加了很多小目标，提升了网络的鲁棒性。



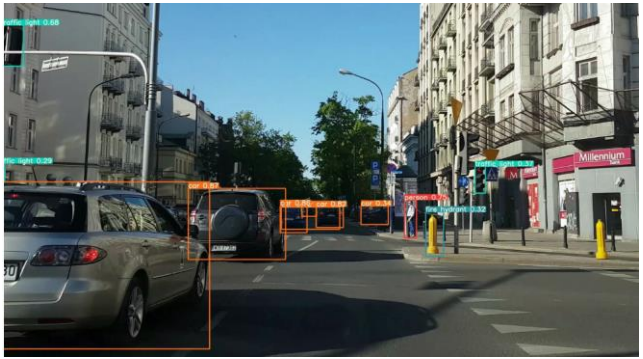
自适应锚框计算

- 在YOLOv3、YOLOv4中，训练不同的数据集时，初始候选框（锚）是通过单独的程序运行的。
- YOLOv5中将此功能嵌入到代码中，每次训练自适应地计算不同训练集中的最佳候选框。

```
anchors:
  - [116,90, 156,198, 373,326]  # P5/32
  - [30,61, 62,45, 59,119]  # P4/16
  - [10,13, 16,30, 33,23]  # P3/8
```

```
parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
```

在代码中类似设置，每次训练时，不会自动计算



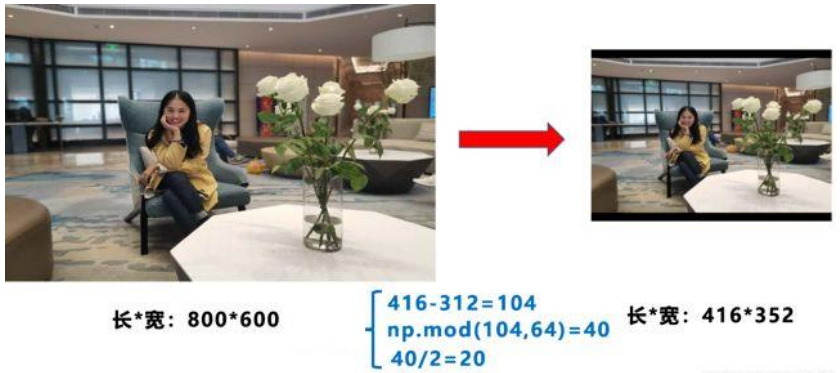
自适应图片缩放

- YOLOv5采用以下方式将原始图片统一缩放到一个标准尺寸，再送入检测网络中。
- YOLOv5的输入图像可以是方的，例如608*608,416*416等，可以使用mosic数据增强；也可以是非对称的，但不支持mosic数据增强。
- 实际项目可能会指定输入图像的长和高，需要对YOLOv5代码做一些修改。

缩放填充

传统填充

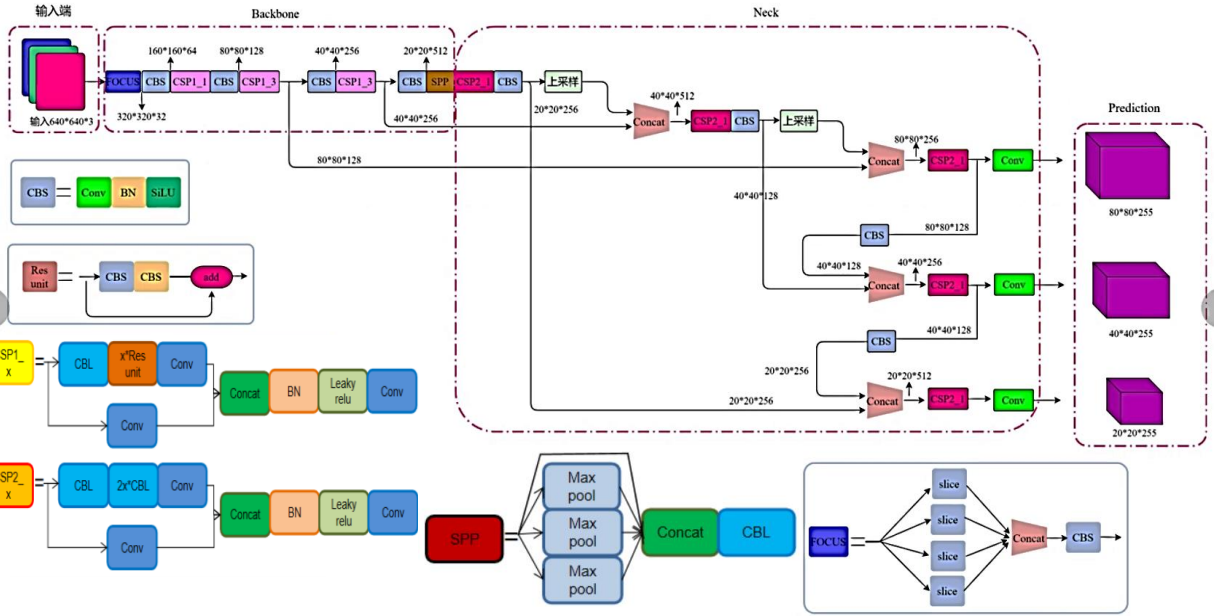
自适应的添加最少的黑边（推理时用）

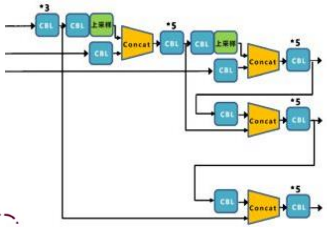


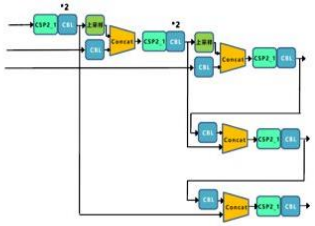
- 原始图片的长宽都乘以最小的缩放系数0.52，宽变成了416，高变成了312（选择小的缩放系数）。
- 416-312=104，得到原本需要填充的高度，采用取余数的方式，得到40个像素，再除以2，得到图片高度两端需要填充的数值。

CSP结构

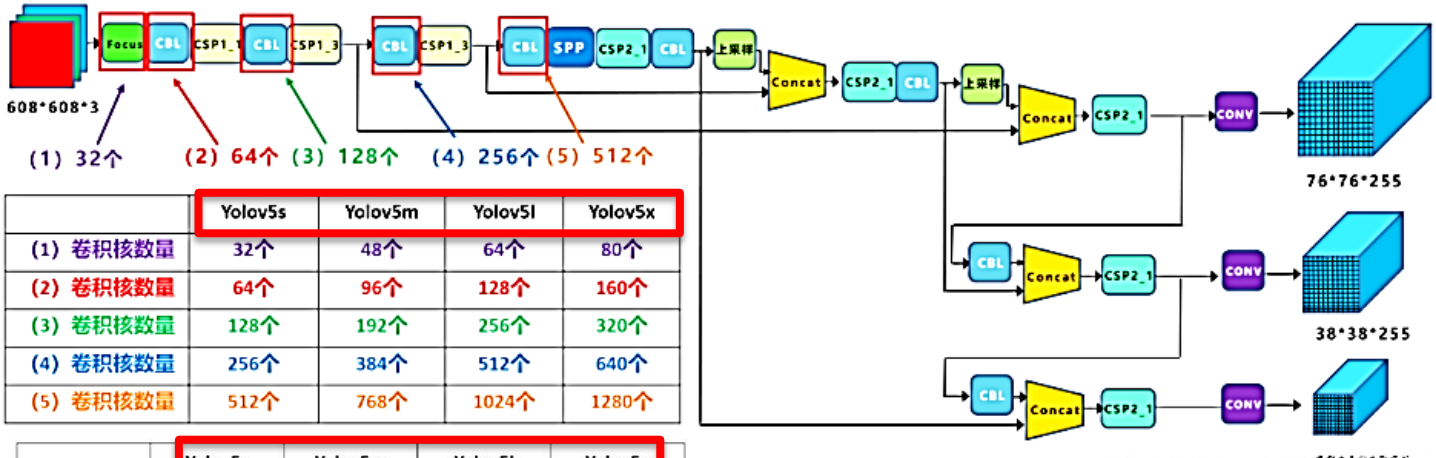
- YOLOv5中设计了两种CSP结构：CSP1_X结构应用于Backbone主干网络，CSP2_X结构则应用于Neck中。
- YOLOv5的Neck结构中，采用CSP2结构，加强网络特征融合的能力。





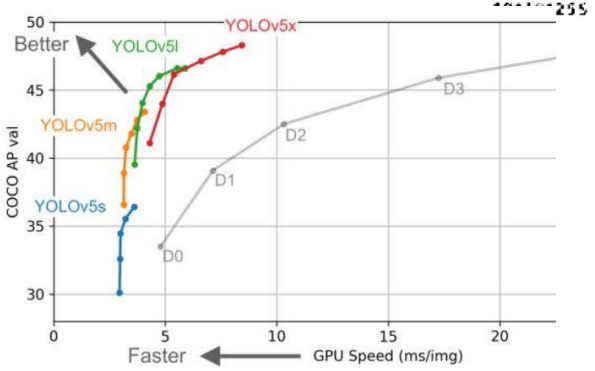


YOLOv5的几种版本



	Yolov5s	Yolov5m	Yolov5l	Yolov5x
(1) 卷积核数量	32个	48个	64个	80个
(2) 卷积核数量	64个	96个	128个	160个
(3) 卷积核数量	128个	192个	256个	320个
(4) 卷积核数量	256个	384个	512个	640个
(5) 卷积核数量	512个	768个	1024个	1280个

	Yolov5s	Yolov5m	Yolov5l	Yolov5x
第一个CSP1	CSP1_1	CSP1_2	CSP1_3	CSP1_4
第二个CSP1	CSP1_3	CSP1_6	CSP1_9	CSP1_12
第三个CSP1	CSP1_3	CSP1_6	CSP1_9	CSP1_12
第一个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4
第二个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4
第三个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4
第四个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4
第五个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4



YOLOv5的损失函数

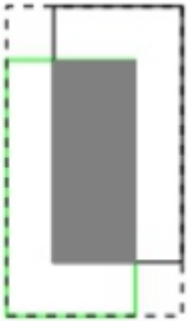
- YOLOv4中采用CIOU_Loss作为Bounding box的损失函数，而YOLOv5中采用GIOU_Loss作为Bounding box的损失函数。

GIOU损失函数：

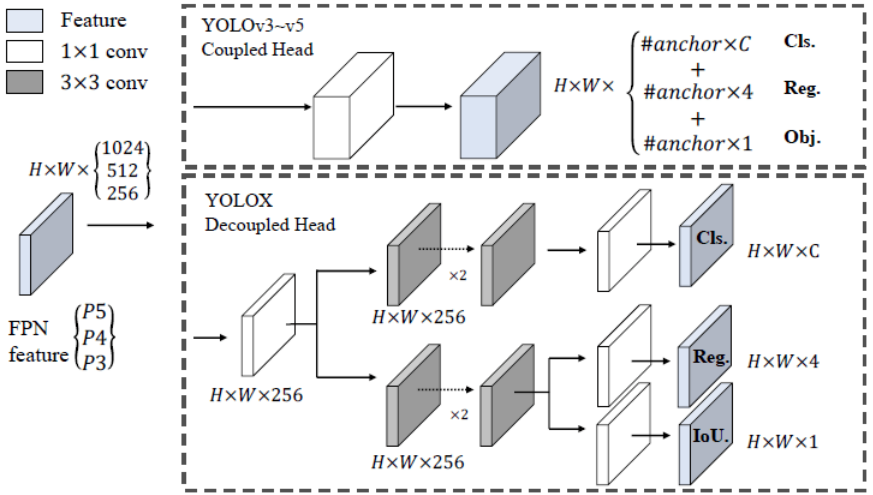
$$GIoU = IoU - \frac{A^c - u}{A^c}$$

$$-1 \le GIoU \le 1$$

$$L_{GIoU} = 1 - GIoU$$

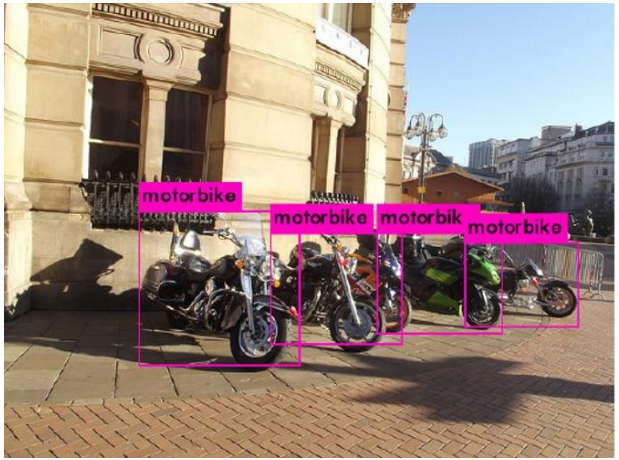


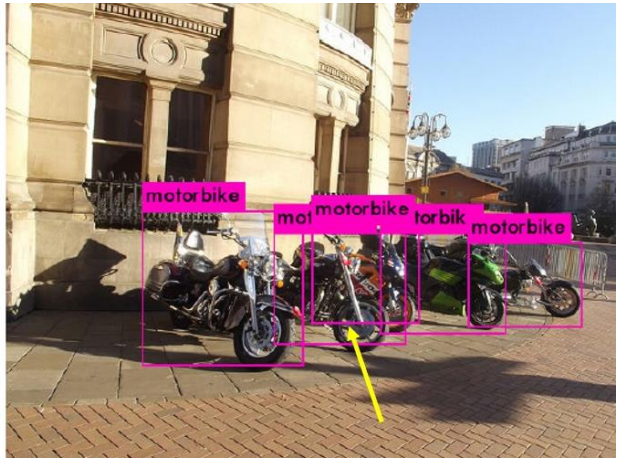
A^c表示A和B的区域用矩形框框起来的面积，u表示A和B的并集。



非极大值抑制

- 当两个目标非常近或相互有部分遮挡时，置信度低的目标会被置信度高的框抑制。
- YOLOv4采用DIOU_NMS，而YOLOv5中采用加权NMS。对于一些**遮挡重叠**的目标，可能会有一些改进。





softNMS

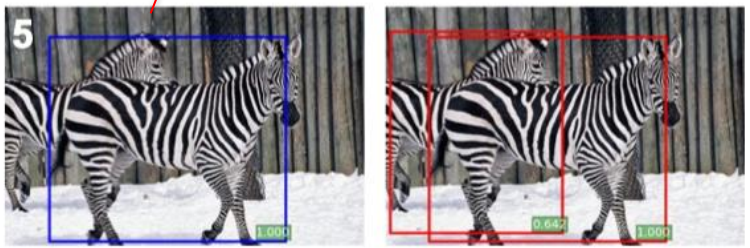
- 使用 softNMS，设置稍低一点的置信度分数，而不是像NMS直接置零。

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ s_i(1 - \text{iou}(\mathcal{M}, b_i)), & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases},$$

当iou条件满足时，si乘上一个1-iou，线性变小 $s_i = s_i e^{-\frac{iou(\mathcal{M}, b_i)^2}{\sigma}}, \forall b_i \notin \mathcal{D}$

高斯函数惩罚，越接近高斯分布中心，惩罚力度越大

Soft-NMS的效果也比较明显：重叠的物体被更大程度的保留下来



Input : $\mathcal{B} = \{b_1, .., b_N\}, \mathcal{S} = \{s_1, .., s_N\}, N_t$
$\mathcal{B}$ is the list of initial detection boxes
$\mathcal{S}$ contains corresponding detection scores
N_t is the NMS threshold

begin
  $\mathcal{D} \leftarrow \{\}$
  while $\mathcal{B} \neq$ *empty* **do**
    $m \leftarrow$ argmax $\mathcal{S}$
    $\mathcal{M} \leftarrow b_m$
    $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$
    for b_i *in* $\mathcal{B}$ **do**
      if $iou(\mathcal{M}, b_i) \geq N_t$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} - b_i; \mathcal{S} \leftarrow \mathcal{S} - s_i$
      end NMS
      $s_i \leftarrow s_i f(iou(\mathcal{M}, b_i))$ Soft-NMS
    end
  end
  return $\mathcal{D}, \mathcal{S}$
end

原始NMS算法

Soft-NMS算法