YOLOv5

Newest and Fastest One-Stage Objection Detection Model

分享人：王立杰

2020.09.11

目录

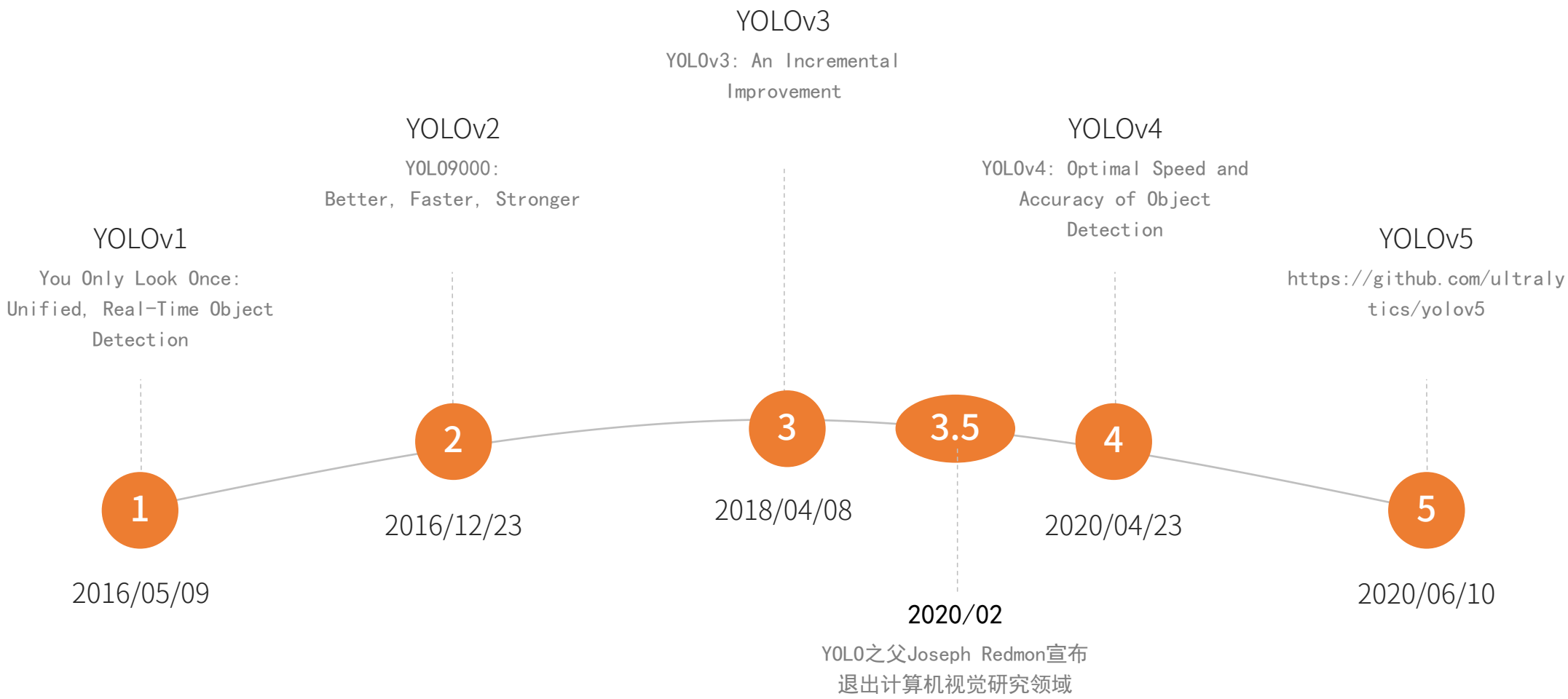
CONTENT

01

v5之引入

The Introduction of YOLOv5

YOLO系列时间轴


YOLOv3
YOLOv3: An Incremental Improvement
YOLOv2
YOLO9000: Better, Faster, Stronger
YOLOv4
YOLOv4: Optimal Speed and Accuracy of Object Detection
YOLOv1
You Only Look Once: Unified, Real-Time Object Detection
YOLOv5
https://github.com/ultralytics/yolov5
1
2
3
3.5
4
5
2016/05/09
2016/12/23
2018/04/08
2020/02
2020/04/23
2020/06/10
YOLO之父Joseph Redmon宣布退出计算机视觉研究领域

YOLOv5介绍

项目地址：https://github.com/ultralytics/yolov5
暂无Paper

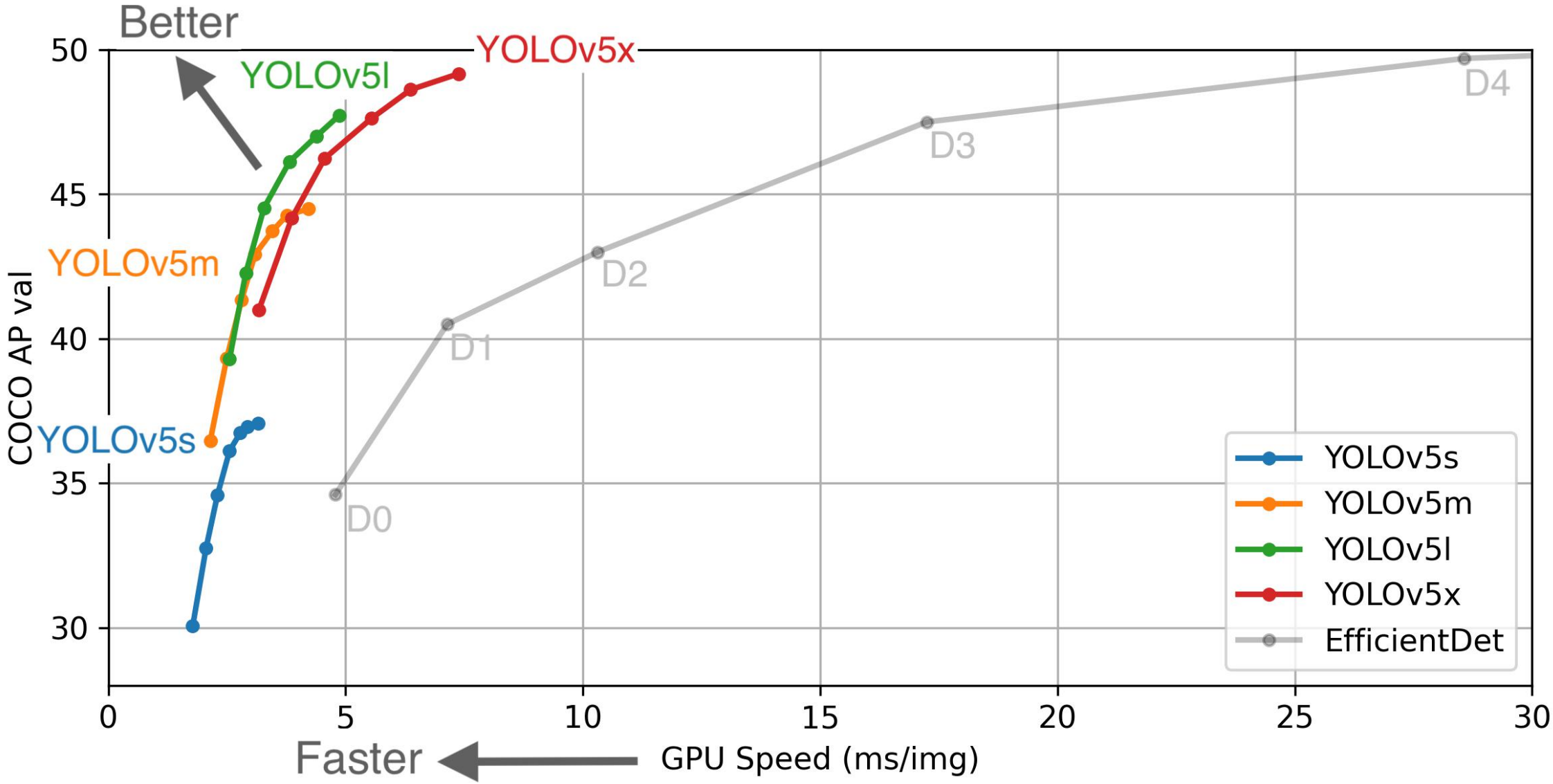
命名与争议：Is YOLOv5 the Correct Name?

反对者：
(1) he is not an original YOLO author,
(2) he did not publish a paper, and/or
(3) the implementation is not sufficiently novel.

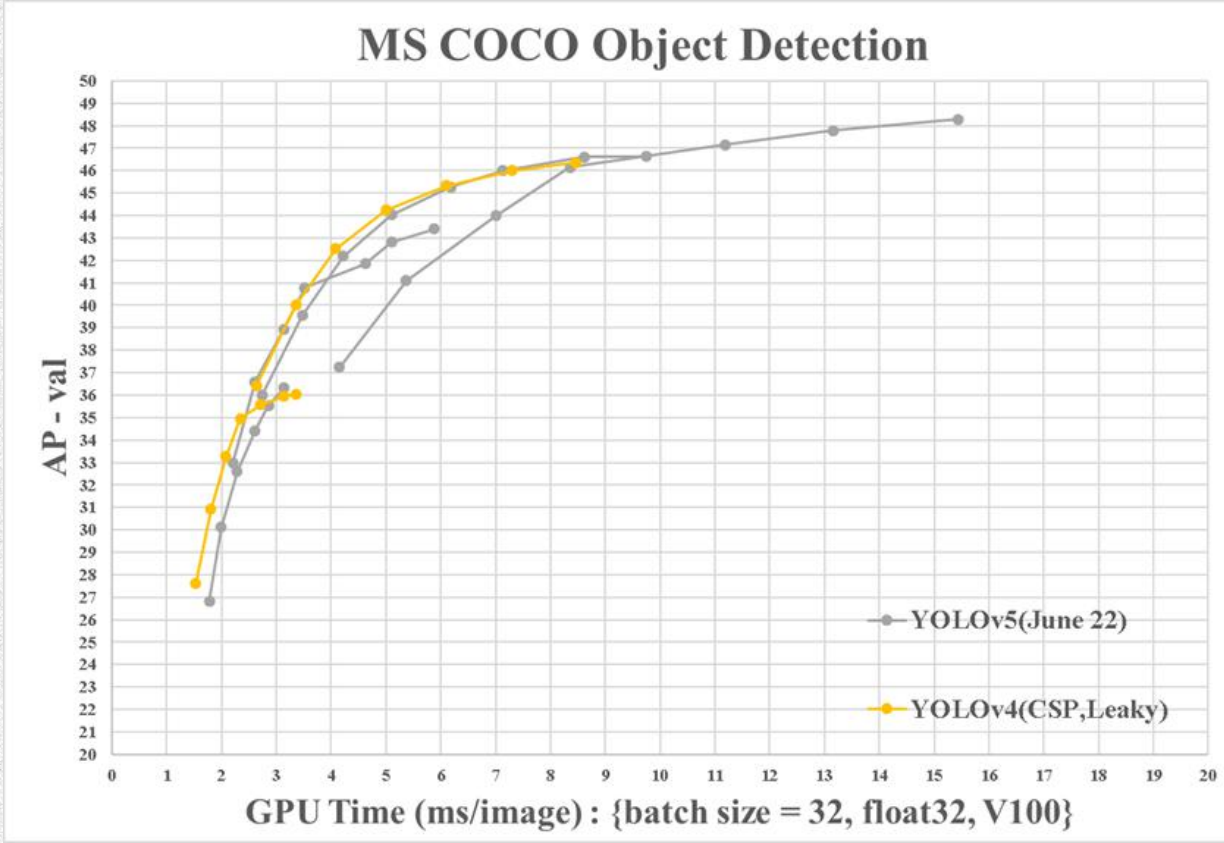
认可者/中立者：
(1) PyTorch版最强模型；
(2) 动机认可：v4性能提升最多的Mosaic的发明者；
(3) 命名无所谓，有贡献就行。

02
v5之性能
The Performance of YOLOv5

YOLOv5性能（官方）

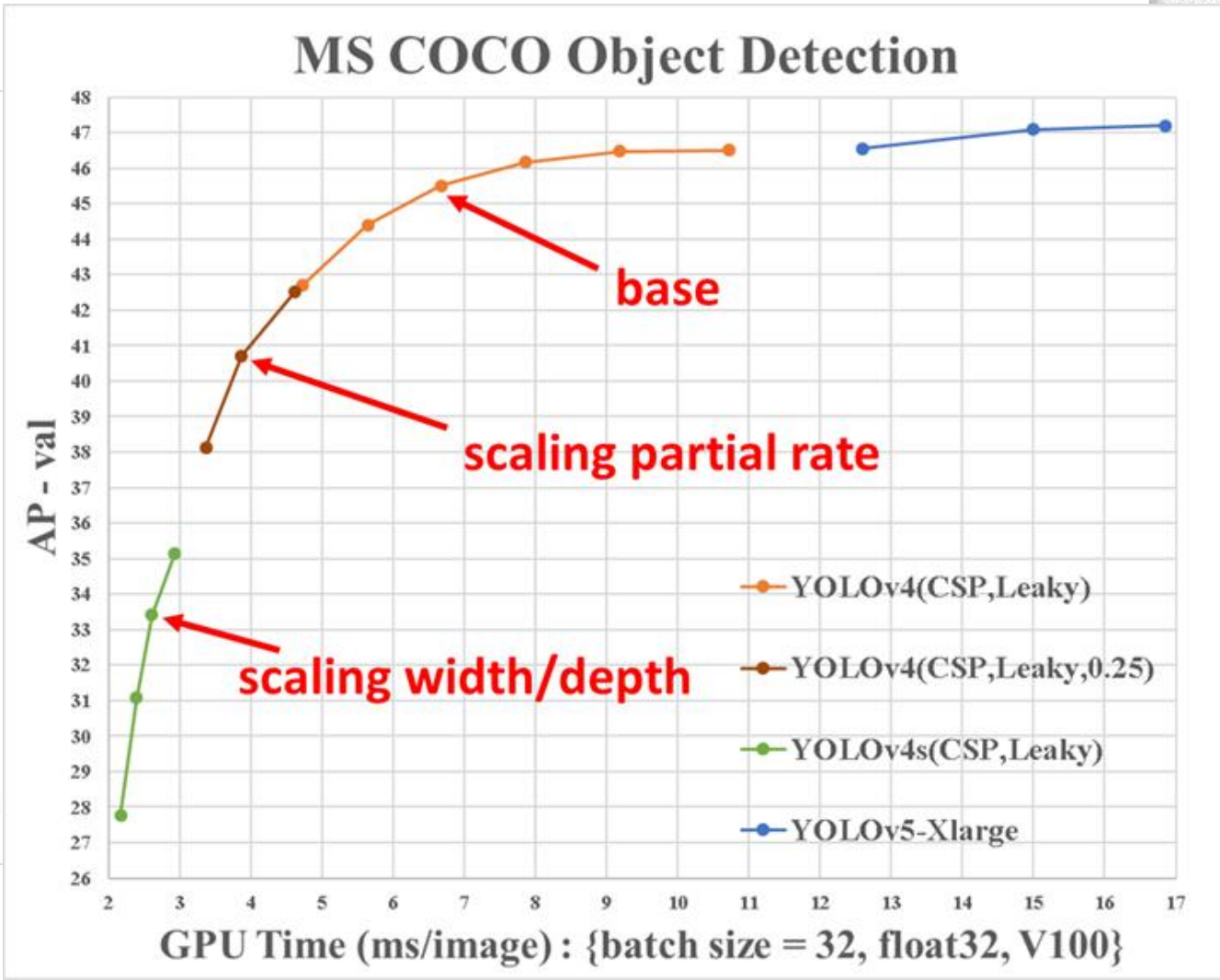

Better
YOLOv5x
YOLOv5l
YOLOv5m
YOLOv5s
D0
D1
D2
D3
D4
COCO AP val
50
45
40
35
30
0
5
10
15
20
25
30
Faster
GPU Speed (ms/img)
YOLOv5s
YOLOv5m
YOLOv5l
YOLOv5x
EfficientDet

YOLOv5性能（非官方）



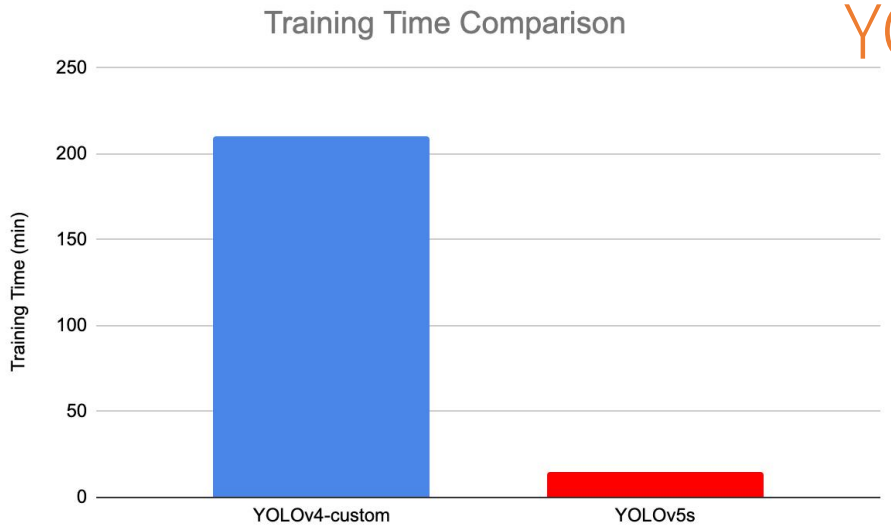
两者性能其实很接近，但是从数据上看YOLO V4仍然是最佳对象检测框架。

https://github.com/ultralytics/yoloV5/issues/6

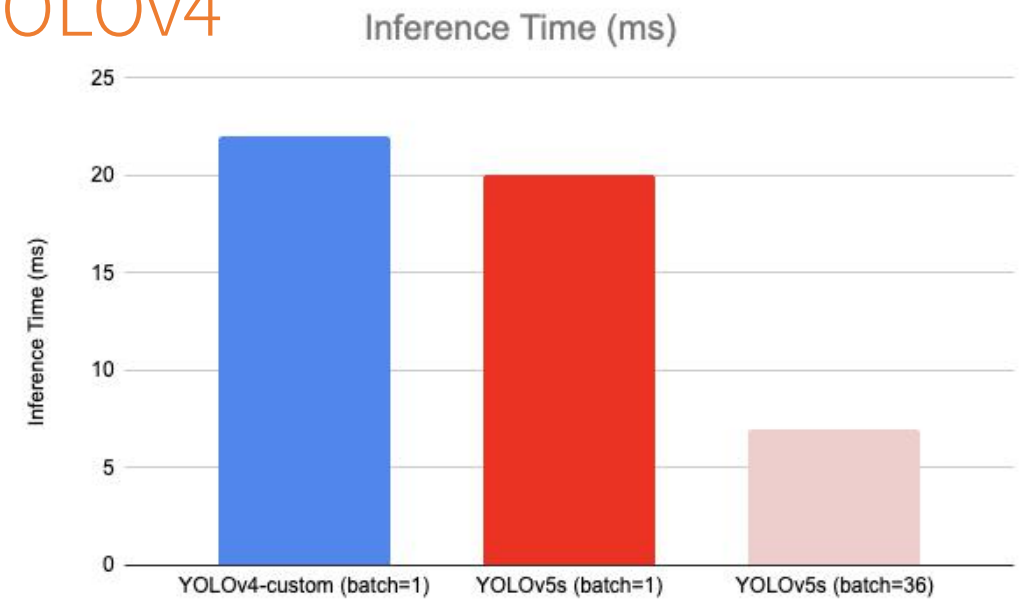


only keep Pareto optimality curve

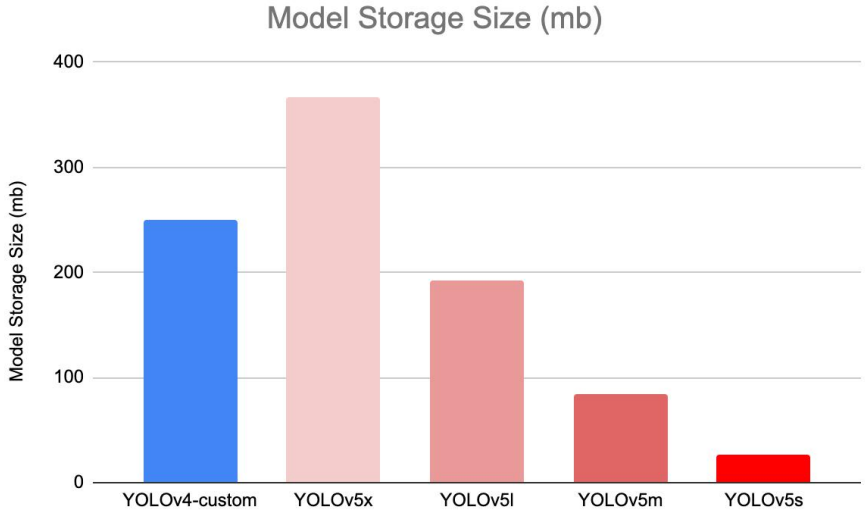
YOLOv5vsYOLOv4



max validation evaluation: 3.5 hrs(1300) vs 14.46 minutes(200)



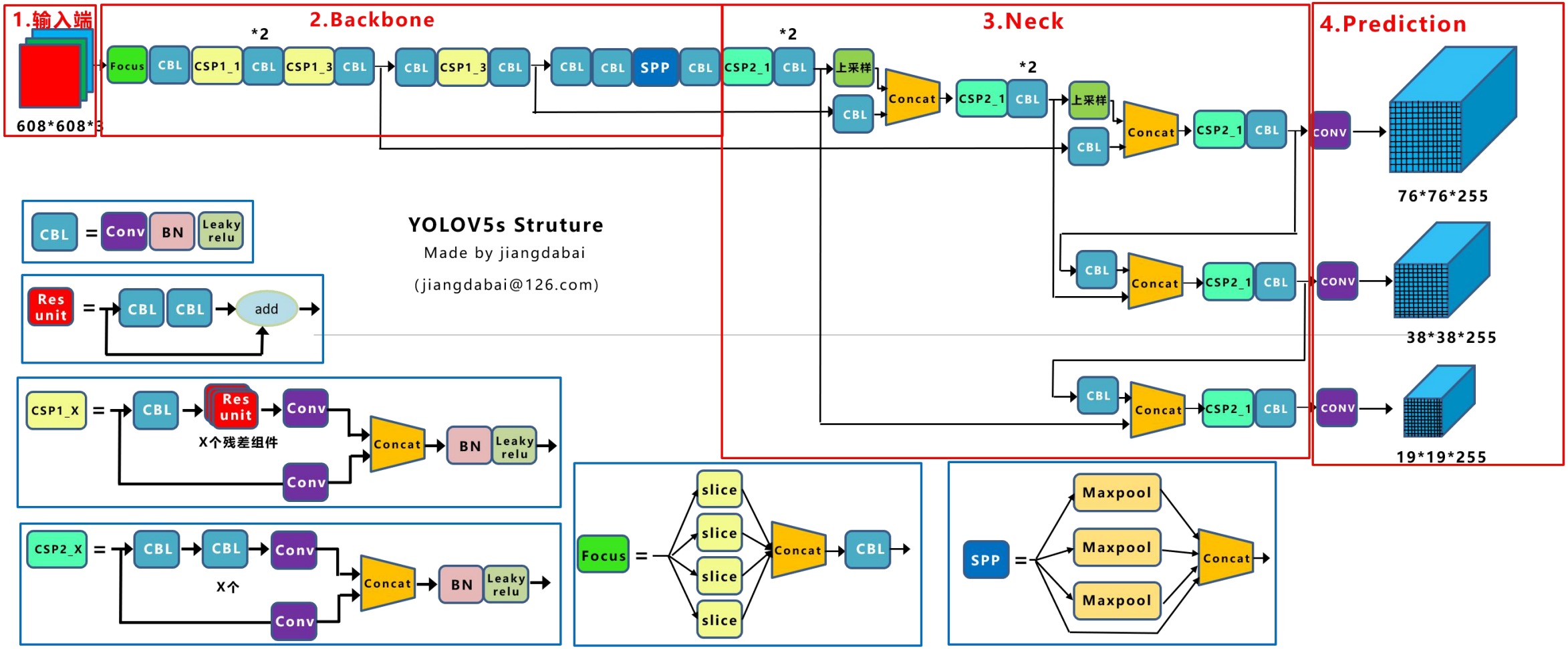
mAP was similar, both models achieved their max.



https://blog.roboflow.com/yolov4-versus-yolov5/

03
v5之框图
The Architecture of YOLOv5

YOLOv5框架


1.输入端
608*608*3
2.Backbone
*2
Focus
CBL
CSP1_1
CBL
CSP1_3
CBL
CBL
CSP1_3
CBL
CBL
CBL
SPP
CBL
3.Neck
*2
CSP2_1
CBL
上采样
CBL
Concat
*2
CSP2_1
CBL
上采样
CBL
Concat
CSP2_1
CBL
CBL
Concat
CSP2_1
CBL
CBL
Concat
CSP2_1
CBL
4.Prediction
CONV
76*76*255
CONV
38*38*255
CONV
19*19*255
CBL = Conv BN Leaky relu
Res unit = CBL CBL add
YOLOV5s Struture
Made by jiangdabai
(jiangdabai@126.com)
CSP1_X = CBL Res unit Conv Concat BN Leaky relu
X个残差组件
Conv
CSP2_X = CBL CBL Conv Concat BN Leaky relu
X个
Conv
Focus = slice slice slice slice Concat CBL
SPP = Maxpool Maxpool Maxpool Concat

Yolov5的四种网络结构

Yolov5代码中的四种网络，和之前的Yolov3，Yolov4中的cfg文件不同，都是以yaml的形式来呈现。

而且四个文件的内容基本上都是一样的，只有最上方的depth_multiple和width_multiple两个参数不同。

yolov5s.yaml

```yaml
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
```

yolov5m.yaml

```yaml
depth_multiple: 0.67  # model depth multiple
width_multiple: 0.75  # layer channel multiple
```

yolov5l.yaml

```yaml
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple
```

yolov5x.yaml

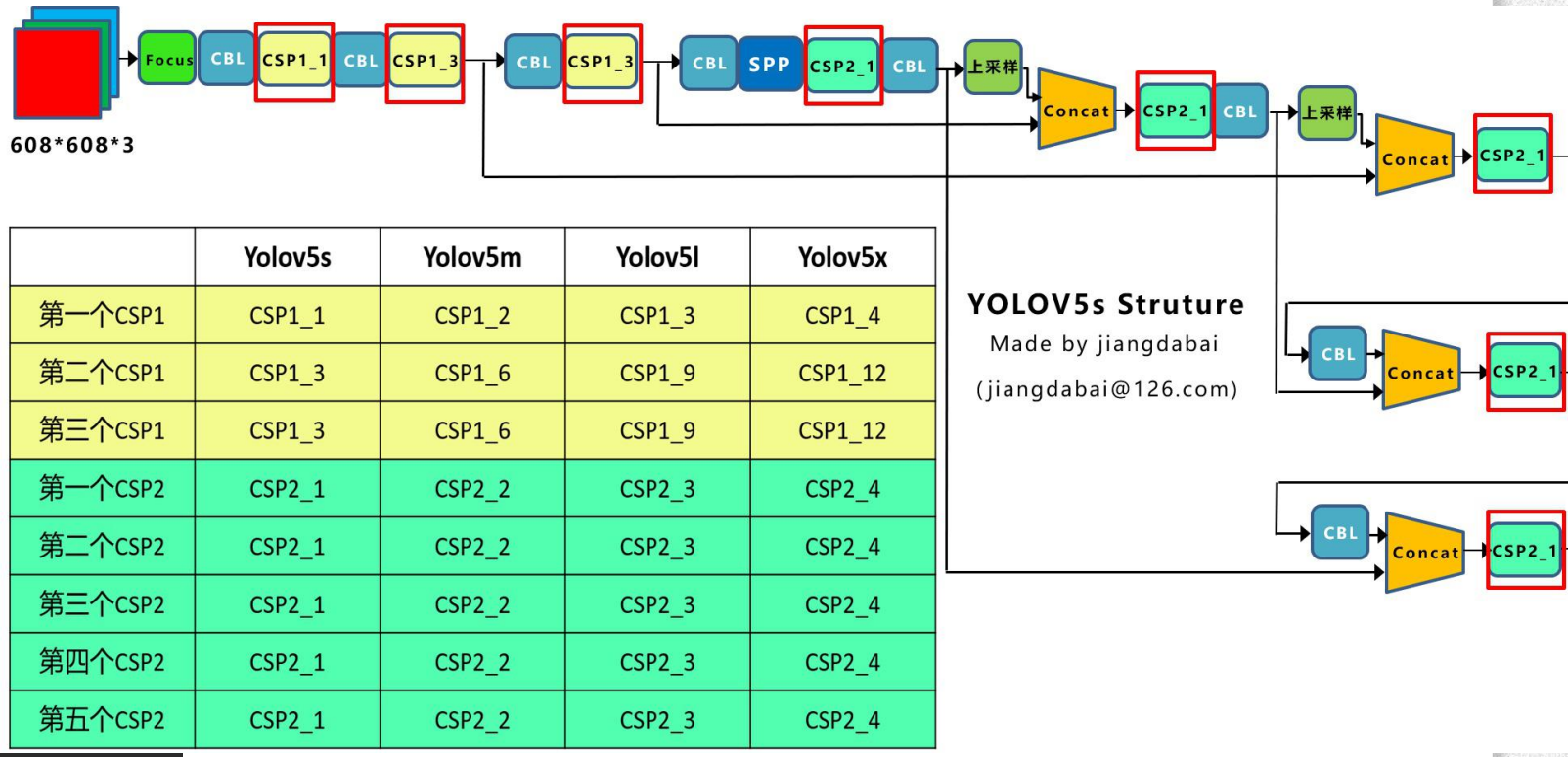
```yaml
depth_multiple: 1.33  # model depth multiple
width_multiple: 1.25  # layer channel multiple
```

四种结构就是通过上面的两个参数，来进行控制网络的深度和宽度。其中**depth_multiple控制网络的深度，width_multiple控制网络的宽度。**

Yolov5的四种网络结构

depth_multiple控制网络的深度

两种CSP结构，CSP1和CSP2，其中CSP1结构主要应用于Backbone中，CSP2结构主要应用于Neck中。



	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

```yaml
# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],    第一个CSP1
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],    第二个CSP1
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],    第三个CSP1
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  # 9
  ]
```

Yolov5中，网络的不断加深，也在不断增加网络特征提取和特征融合的能力。

yolo.py

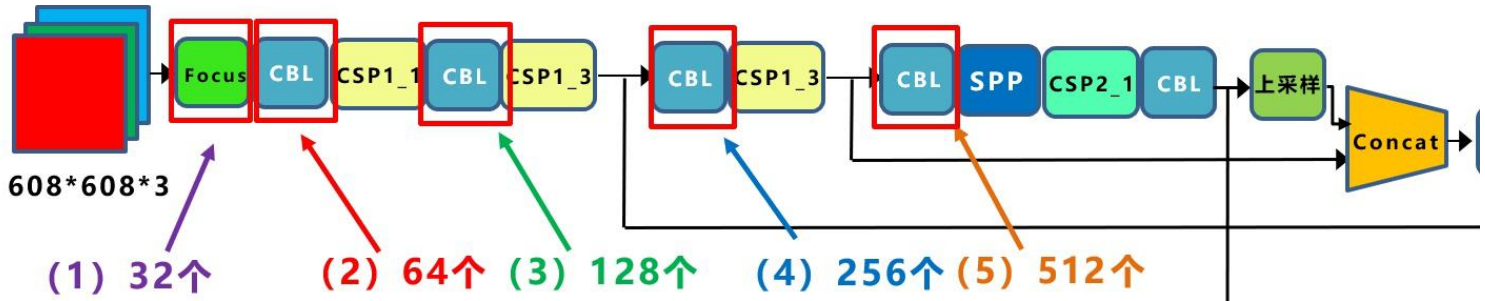
```
177         anchors, nc, gd, gw = d['anchors'], d['nc'], d['depth_multiple'], d['width_multiple']
182     for i, (f, n, m, args) in enumerate(d['backbone'] + d['head']):  # from, number, module, args
190             n = max(round(n * gd), 1) if n > 1 else n  # depth gain
```

Yolov5的四种网络结构

width_multiple控制网络的宽度

四种yolov5结构在不同阶段的卷积核的数量都是不一样的，即网络的宽度。

当然卷积核的数量越多，特征图的宽度越宽，网络提取特征的学习能力也越强。



	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

yolo.py

```
177        anchors, nc, gd, gw = d['anchors'], d['nc'], d['depth_multiple'], d['width_multiple']
182        for i, (f, n, m, args) in enumerate(d['backbone'] + d['head']):  # from, number, module, args
191            if m in [nn.Conv2d, Conv, Bottleneck, SPP, DWConv, MixConv2d, Focus, CrossConv, BottleneckCSP, C3]:
192                c1, c2 = ch[f], args[0]
193                c2 = make_divisible(c2 * gw, 8) if c2 != no else c2
```

04
v5之改进

Improvement in YOLOv5

YOLOv5改进

01

输入端

（1）Mosaic数据增强
（2）自适应锚框计算
（3）自适应图片缩放

02

Backbone

（1）Focus结构
（2）CSP结构

03

Neck

（1）FPN+PAN结构

04

Prediction

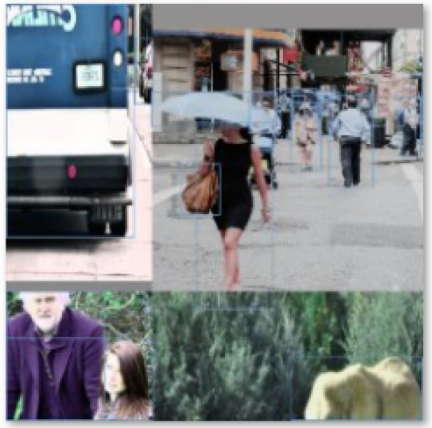
（1）GIOU_Loss
（2）加权nms

输入端之一：Mosaic数据增强


aug_-319215602_0_-238783579.jpg


aug_-1271888501_0_-749611674.jpg


aug_1462167959_0_-1659206634.jpg


aug_1474493600_0_-45389312.jpg


aug_1715045541_0_603913529.jpg


aug_1779424844_0_-589696888.jpg

优点：

（1）**丰富数据集：**

随机使用4张图片，随机缩放，再随机分布进行拼接，大大丰富了检测数据集，特别是随机缩放增加了很多小目标，让网络的鲁棒性更好。

（2）**减少GPU：**

Mosaic增强训练时，可以直接计算4张图片的数据，使得Mini-batch大小并不需要很大，一个GPU就可以达到比较好的效果。

输入端之二：自适应锚框计算

yolo5l.yaml

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

train.py

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

```python
# Check anchors
if not opt.noautoanchor:
    check_anchors(dataset, model=model, thr=hyp['anchor_t'], imgsz=imgsz)
```

在Yolov3、Yolov4中，训练不同的数据集时，计算初始锚框的值是通过单独的程序运行的。

但Yolov5中将此功能嵌入到代码中，每次训练时，自适应的计算不同训练集中的最佳锚框值。

控制的代码即train.py中上面一行代码，设置成**True**，每次训练时，不会自动计算。

输入端之二：自适应锚框计算

generate.py/check_anchors

```
83  def check_anchors(dataset, model, thr=4.0, imgsz=640):
84      # Check anchor fit to data, recompute if necessary
85      print('\nAnalyzing anchors... ', end='')
```

```
91      def metric(k):  # compute metric
92          r = wh[:, None] / k[None]
93          x = torch.min(r, 1. / r).min(2)[0]  # ratio metric
94          best = x.max(1)[0]  # best_x
95          aat = (x > 1. / thr).float().sum(1).mean()  # anchors above threshold
96          bpr = (best > 1. / thr).float().mean()  # best possible recall
97          return bpr, aat
98
99      bpr, aat = metric(m.anchor_grid.clone().cpu().view(-1, 2))
100     print('anchors/target = %.2f, Best Possible Recall (BPR) = %.4f' % (aat, bpr), end='')
101     if bpr < 0.98:  # threshold to recompute
102         print('. Attempting to generate improved anchors, please wait...' % bpr)
103         na = m.anchor_grid.numel() // 2  # number of anchors
104         new_anchors = kmean_anchors(dataset, n=na, img_size=imgsz, thr=thr, gen=1000, verbose=False)
105         new_bpr = metric(new_anchors.reshape(-1, 2))[0]
106         if new_bpr > bpr:  # replace anchors
107             new_anchors = torch.tensor(new_anchors, device=m.anchors.device).type_as(m.anchors)
108             m.anchor_grid[:] = new_anchors.clone().view_as(m.anchor_grid)  # for inference
109             m.anchors[:] = new_anchors.clone().view_as(m.anchors) / m.stride.to(m.anchors.device).view(-1, 1, 1)  # loss
110             check_anchor_order(m)
111             print('New anchors saved to model. Update model *.yaml to use these anchors in the future.')
112         else:
113             print('Original anchors better than new anchors. Proceeding with original anchors.')
114     print('')  # newline
```

输入端之三：自适应图片缩放

train.py

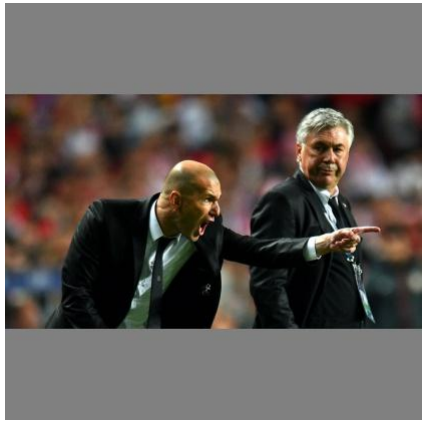
```
396         parser.add_argument('--rect', action='store_true', help='rectangular training')
```

datasets.py

```
294 class LoadImagesAndLabels(Dataset):  # for training/testing
```

```
325         self.image_weights = image_weights
326         self.rect = False if image_weights else rect
327         self.mosaic = self.augment and not self.rect  # load 4 images at a time into a mosaic (only during training)
```

可以看到，Mosaic和Rectangular Training是互相对立的，不能同时实现。



效果：About 1/3 faster on mixed aspect ratio datasets like COCO

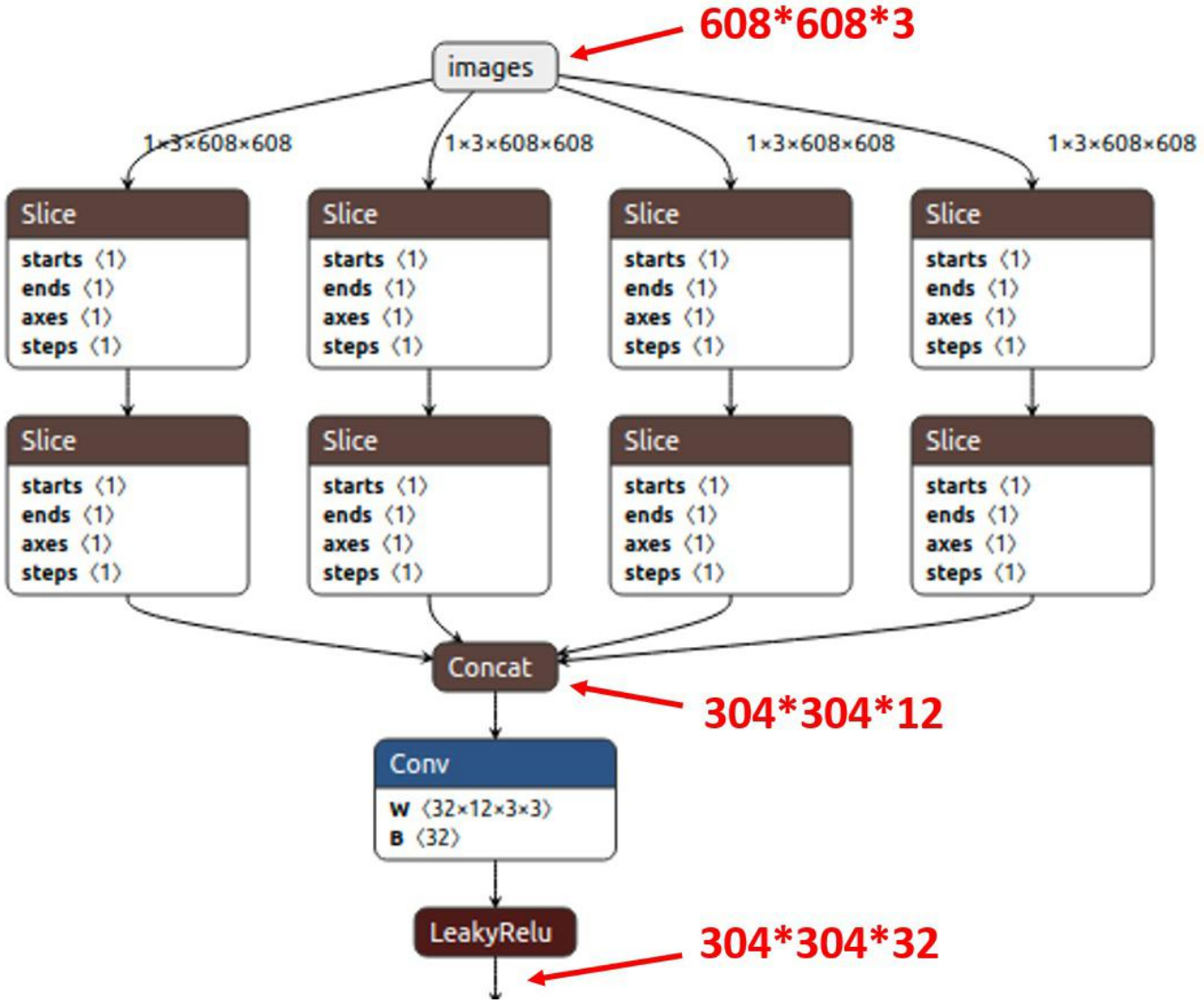
Rectangular Training: https://github.com/ultralytics/yolov3/issues/232

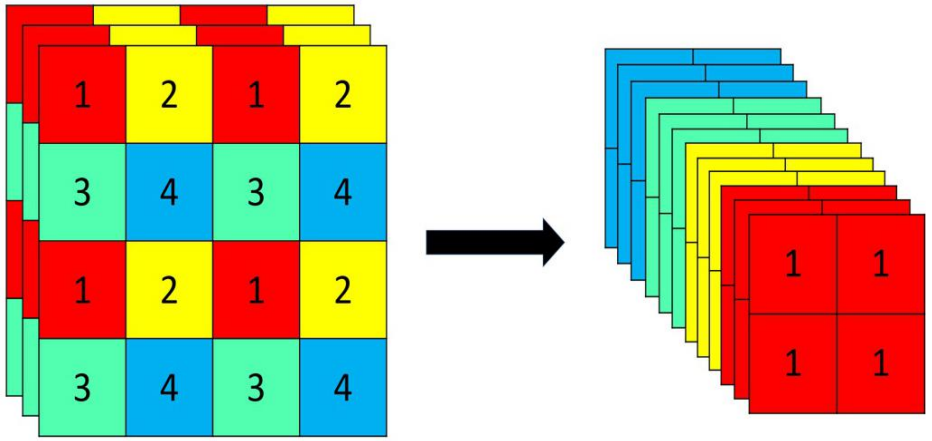
输入端之三：自适应图片缩放

datasets.py

```
294  class LoadImagesAndLabels(Dataset):  # for training/testing
480      def __getitem__(self, index):
485          if self.mosaic:
486              # Load mosaic
497          else:
498              # Load image
499              img, (h0, w0), (h, w) = load_image(self, index)
500
501              # Letterbox
502              shape = self.batch_shapes[self.batch[index]] if self.rect else self.img_size  # final letterboxed shape
503              img, ratio, pad = letterbox(img, shape, auto=False, scaleup=self.augment)
504              shapes = (h0, w0), ((h / h0, w / w0), pad)  # for COCO mAP rescaling
505
506              # Load labels
507              labels = []
508              x = self.labels[index]
509              if x.size > 0:
510                  # Normalized xywh to pixel xyxy format
511                  labels = x.copy()
512                  labels[:, 1] = ratio[0] * w * (x[:, 1] - x[:, 3] / 2) + pad[0]  # pad width
513                  labels[:, 2] = ratio[1] * h * (x[:, 2] - x[:, 4] / 2) + pad[1]  # pad height
514                  labels[:, 3] = ratio[0] * w * (x[:, 1] + x[:, 3] / 2) + pad[0]
515                  labels[:, 4] = ratio[1] * h * (x[:, 2] + x[:, 4] / 2) + pad[1]
```

Backbone之一：Focus结构





切片示意图（4*4*3的图像切片后变成2*2*12的特征图）

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

其最大好处是可以**最大程度的减少信息损失而进行下采样操作**。

Backbone之一：Focus结构

common.py

```python
class Focus(nn.Module):
    # Focus wh information into c-space
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel, stride, padding, groups
        super(Focus, self).__init__()
        self.conv = Conv(c1 * 4, c2, k, s, p, g, act)

    def forward(self, x):  # x(b,c,w,h) -> y(b,4c,w/2,h/2)
        return self.conv(torch.cat([x[..., ::2, ::2], x[..., 1::2, ::2], x[..., ::2, 1::2], x[..., 1::2, 1::2]], 1))
```

yolo5s.yaml

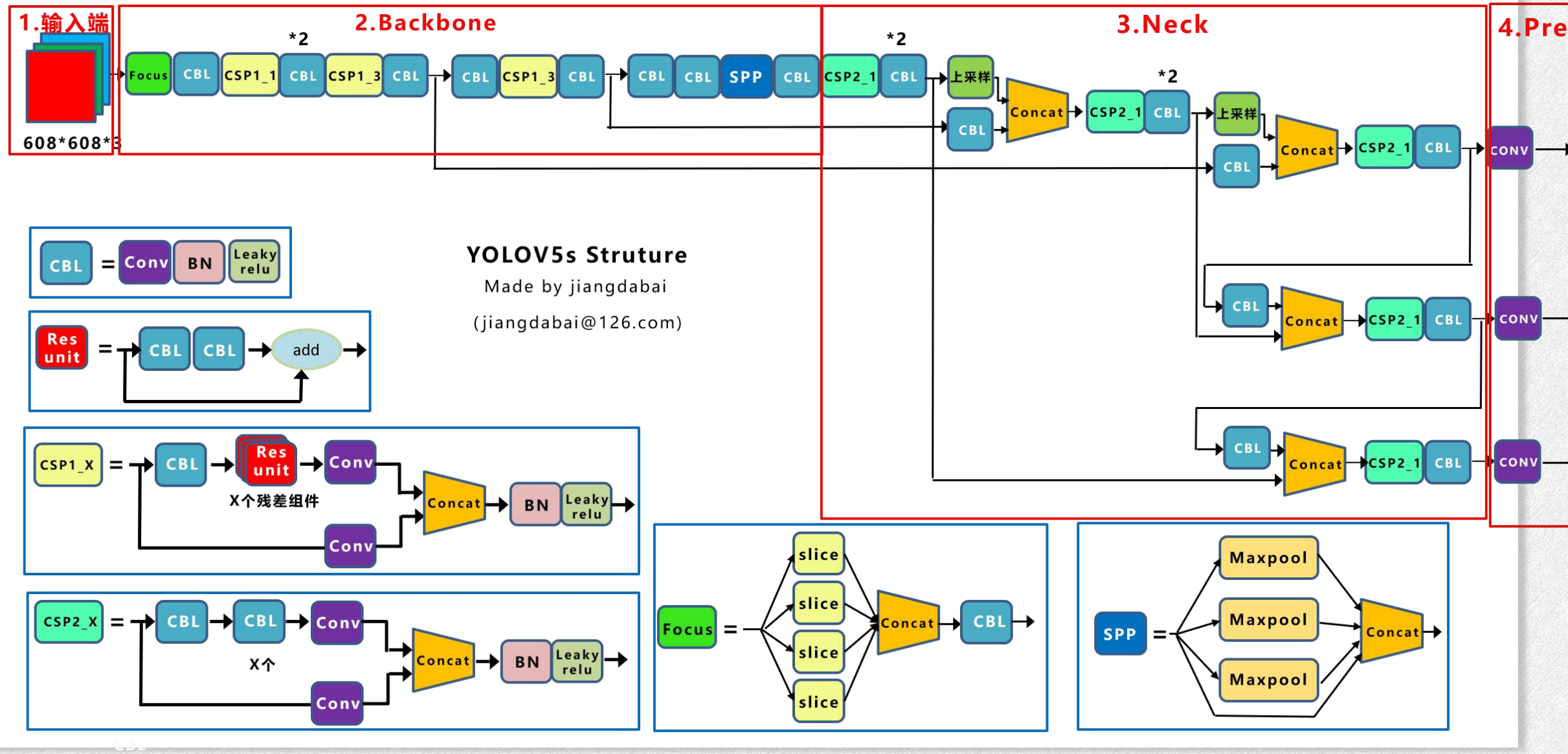
```yaml
# parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
```

```yaml
# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
```

train.py输出：

```
                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Focus                     [3, 32, 3]
```

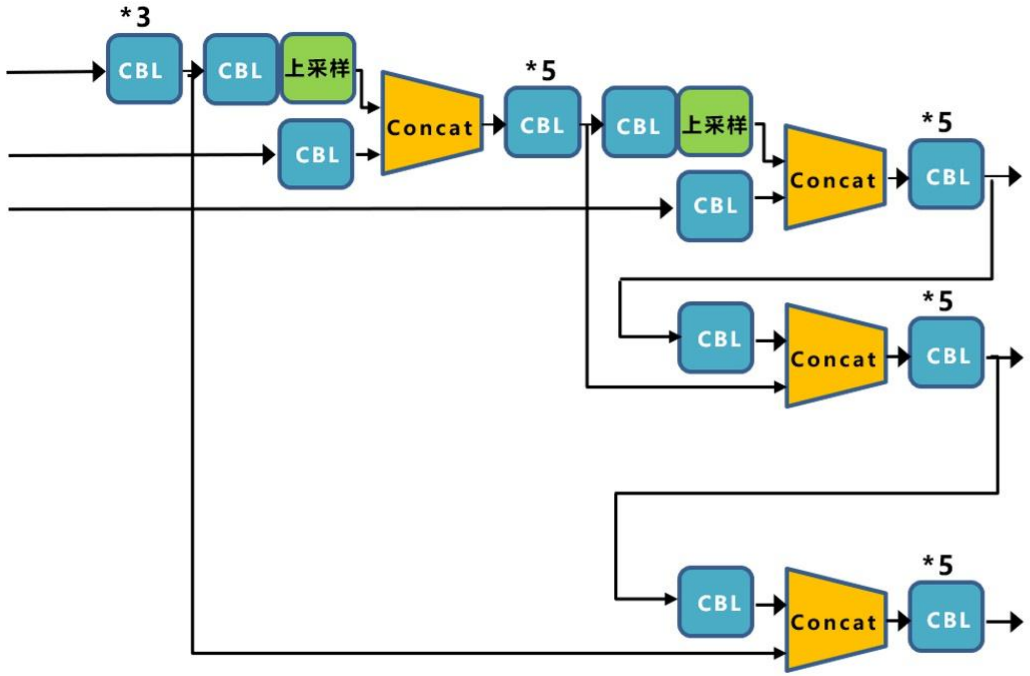
Backbone之二：CSP结构



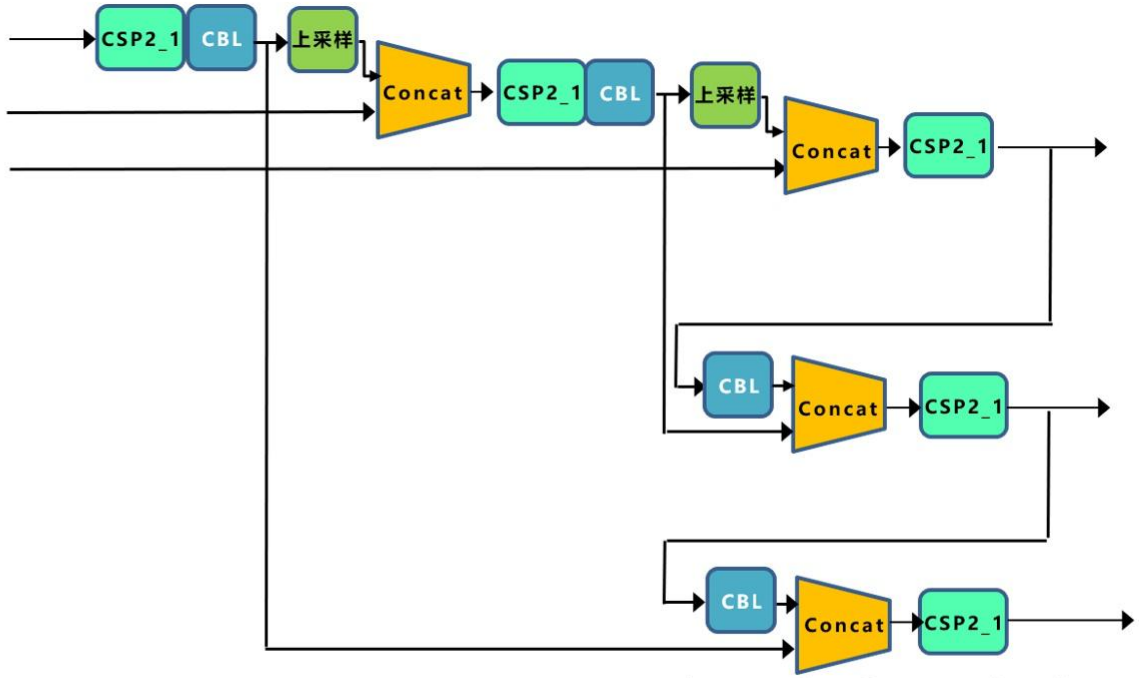
Neck端：FPN+PAN

参考yolov4

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



YOLOv4_Neck



YOLOv5s_Neck

输出端之一：Bounding box损失函数

整体loss：分类loss(BCE) + conf loss(BCE) + bbox loss(GIoU)

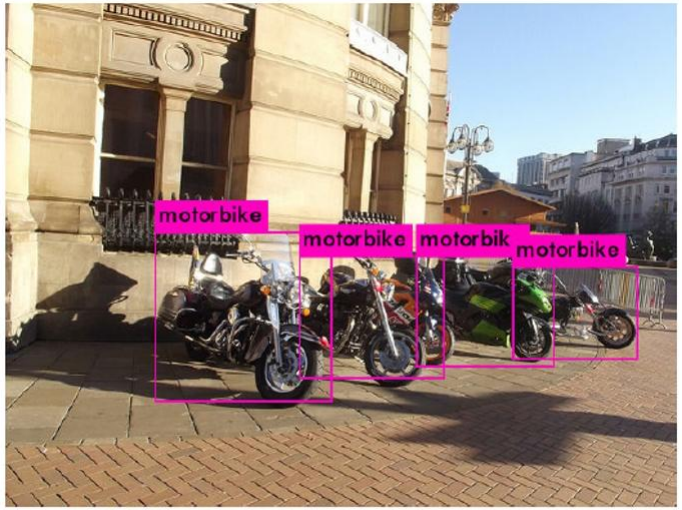
Yolov4：CIOU_Loss

Yolov5：GIOU_Loss

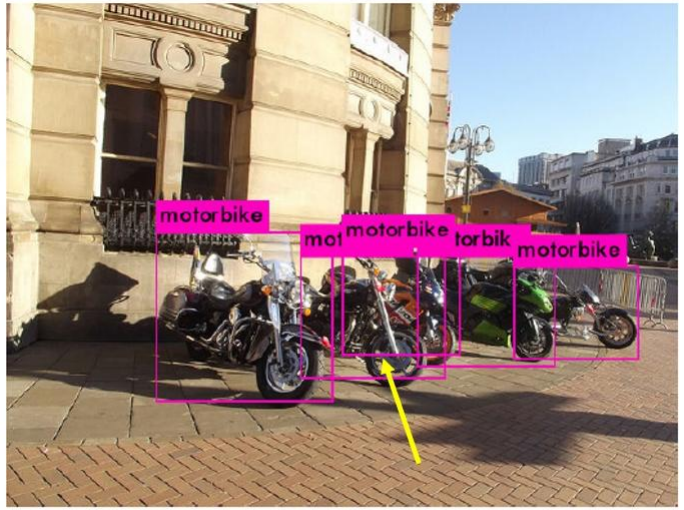
在v4中都有介绍，此处略。

输出端之二：nms非极大值抑制

在目标检测的后处理过程中，针对很多目标框的筛选，通常需要nms操作。



CIOU_Loss+NMS



CIOU_Loss+DIOU_nms

Yolov4：**CIOU_Loss + DIOU_nms**；在同样的参数情况下，将nms中IOU修改成DIOU_nms，改进了目标遮挡重叠的问题。

在参数和普通的IOU_nms一致的情况下，修改成DIOU_nms，可以将两个目标检出。（左图黄色箭头）

为什么不用CIOU_nms?
在测试推理时，没有groundtruth，则不考虑影响因子v，CIOU_nms退化为DIOU_nms。

Yolov5中采用**加权nms**的方式。

虽然大多数状态下效果差不多，但在不增加计算成本的情况下，有稍微的改进也是好的。

参考资料

1. 深入浅出Yolo系列之Yolov5核心基础知识完整讲解：https://zhuanlan.zhihu.com/p/172121380
2. 如何评价YOLOv5？：https://www.zhihu.com/question/399884529
3. Responding to the Controversy about YOLOv5：https://blog.roboflow.com/yolov4-versus-yolov5/
4. 一文读懂YOLO V5 与 YOLO V4：https://zhuanlan.zhihu.com/p/161083602
5. About reproduced results #6：https://github.com/ultralytics/yoloV5/issues/6
6. How to Train YOLOv5 On a Custom Dataset：https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/
7. Train Custom Data：https://github.com/ultralytics/yolov3/wiki/Train-Custom-Data
8. 进击的后浪yolov5深度可视化解析：https://zhuanlan.zhihu.com/p/183838757
9. PPYOLO：2020不容错过的目标检测调参Tricks：https://mp.weixin.qq.com/s/pHOFqFihkkRVTYbkSTlG4w
10. YOLO系列：YOLOv1,YOLOv2,YOLOv3,YOLOv4,YOLOv5简介：https://zhuanlan.zhihu.com/p/136382095
11. 深入浅出Yolo系列之Yolov3&Yolov4&Yolov5核心基础知识完整讲解：https://zhuanlan.zhihu.com/p/143747206

提问时间

QUESTION TIME