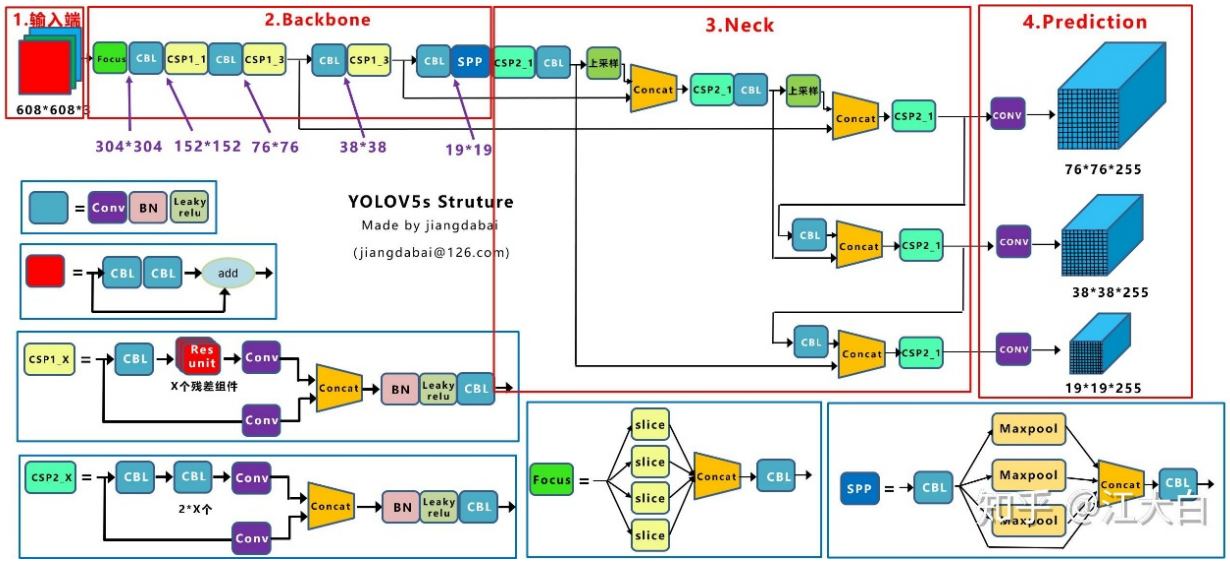
YOLO v5

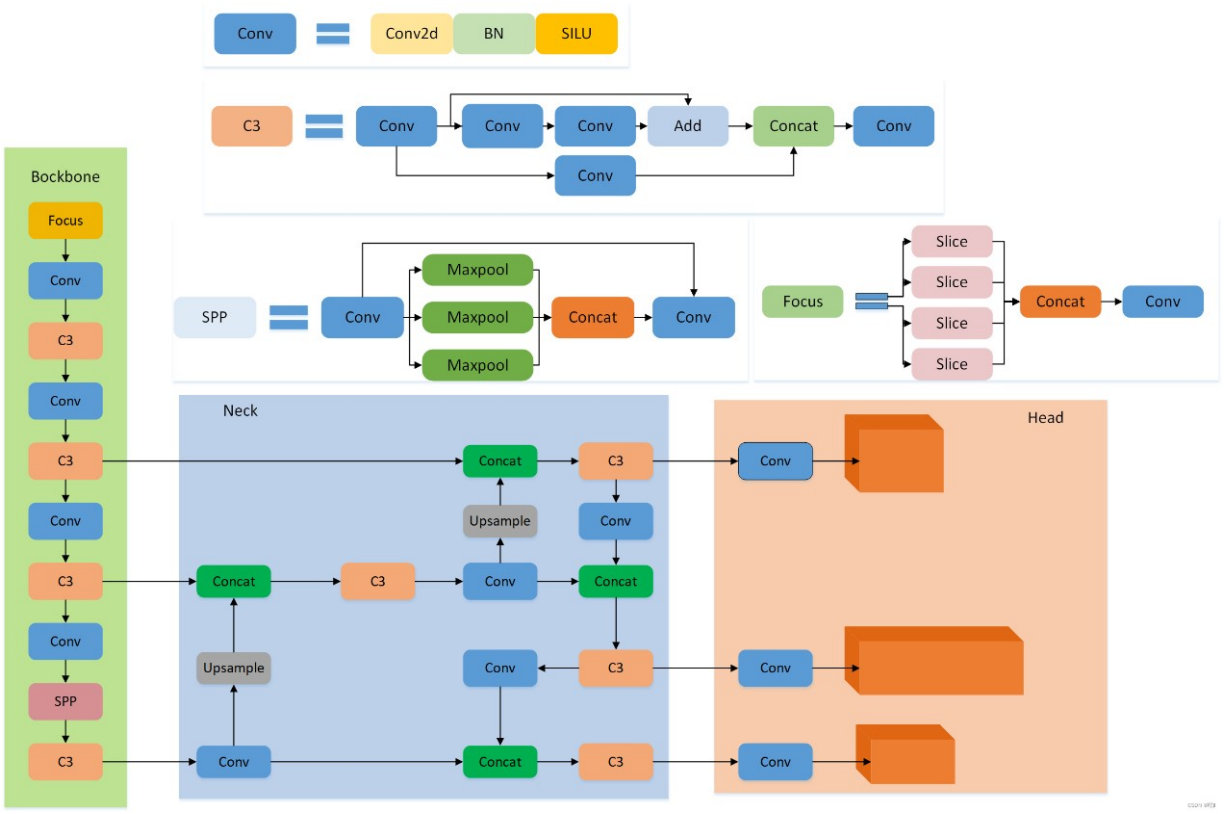
1. Network Design

1.1 Overall Architecture

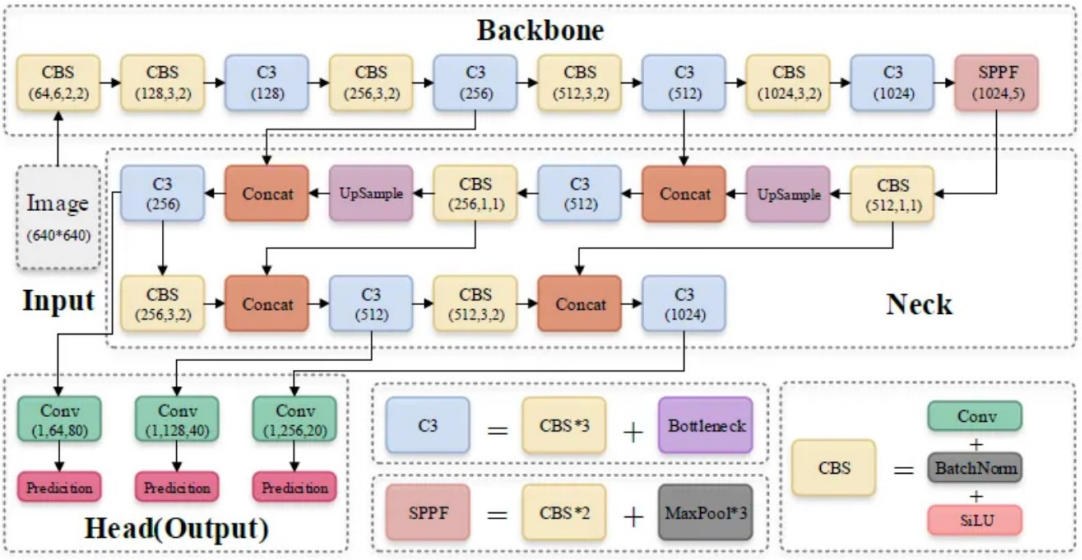
v1.0



v4.0



v6.0



YOLOv5: Model Architecture

1.2 Backbone

The backbone is **CSPDarknet53**.

The main structure is the stacking of one **Focus** module, multiple **CBL** modules and **CSP** modules, and finally one **SPP** module is connected.

1.2.1 CBL

CBL = Conv + BatchNorm + Leaky relu

【Update】Change activation function

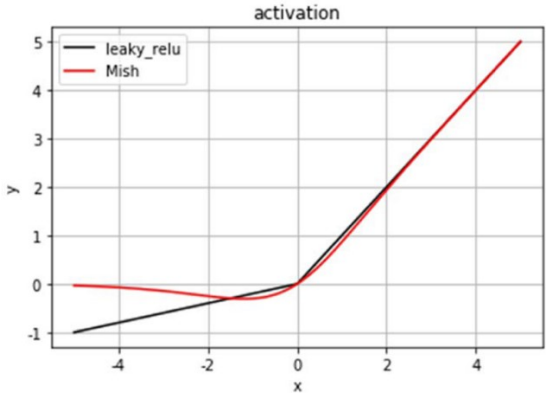
in v3.0:

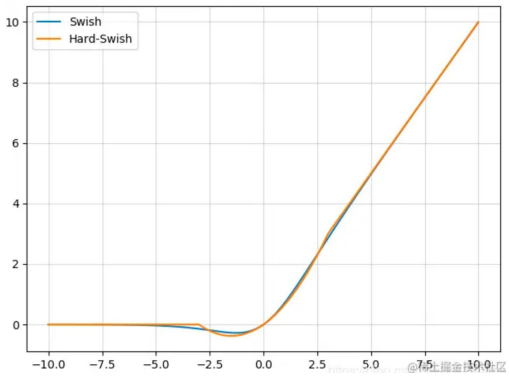
Hardswish replaces Leaky relu in base convolution module

in v4.0:

SiLU (Swish) replaces Leaky relu and Hardswish throughout the model, simplifying the architecture as we now only have one single activation function used everywhere rather than the two types before.

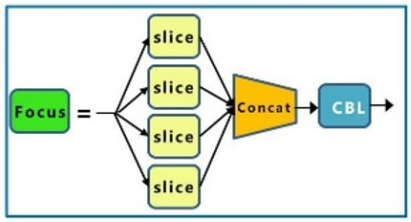
And CBL module becomes **CBS**.

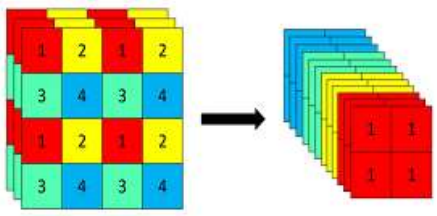




1.2.2 Focus

slice + concat + CBL

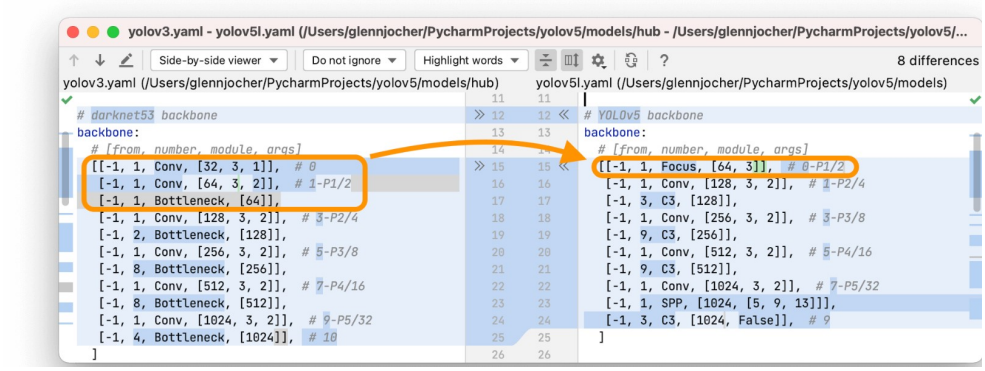




slice and concat is like **Passthrough** in YOLO v2. Slice the image into 4 parts, and concat them. Image size from **608*608*3** to **304*304*12**.

Q 为什么引入 Focus Layer?

The YOLOv5 Focus layer replaces the **first 3 YOLOv3 layers** with a **single layer**:



To reduce layers, reduce parameters, reduce FLOPS, reduce CUDA memory, increase forward and backward speed.
在减少特征信息损失的情况下，减少特征图尺寸的大小。

How to reduce parameters and FLOPS?
可参考 yolov5 中的 Focus 模块的理解_focus 结构-CSDN 博客

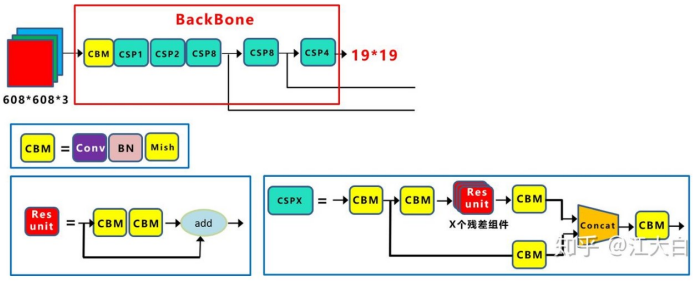
【Update】Remove Focus in v6.0
Focus 的缺点：
1. 某些设备不支持 Focus，且不友好，开销很大，另外切片对不齐模型就崩了。
2. 在比较新的 GPU 设备上，focus 比 conv 慢，尤其是反向传播

So, Replacement of Focus() with an equivalent **Conv (k=6, s=2, p=2) layer** for **improved exportability. 优先考虑主流设备、新一代设备。**

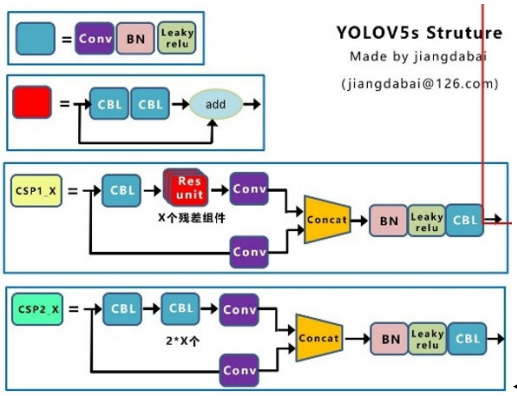
为什么 Conv(k=6, s=2, p=2) layer 可以等价 Focus?
可参考 yolov5 源码解析(0)--focus 层哪去了？_yolov5 为什么不要 focus 层了-CSDN 博客

1.2.3 CSP

YOLO v4 中只有 Backbone 使用了 CSP 结构，而 Yolov5 中设计了两种 CSP 结构。



← CSPX in YOLO v4



← CSP1_X, CSP2_X in YOLO v5

CSP 模块的作用?

1. 缓解梯度信息重复问题
2. 降低计算量与显存占用
3. 增强特征表达能力

CSP1_X 结构应用于 Backbone。
CSP2_X 则应用于 Neck。与 CSP1_X 的区别在于，将 Res unit 换成了 2 * X 个 CBL。

具体实现：yolov5/models/common.py at v1.0 · ultralytics/yolov5

```
class BottleneckCSP(nn.Module):
    # CSP Bottleneck https://github.com/WongKinYiu/CrossStagePartialNetworks
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super(BottleneckCSP, self).__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = nn.Conv2d(c1, c_, 1, 1, bias=False)
        self.cv3 = nn.Conv2d(c_, c_, 1, 1, bias=False)
        self.cv4 = Conv(c2, c2, 1, 1)
        self.bn = nn.BatchNorm2d(2 * c_)  # applied to cat(cv2, cv3)
        self.act = nn.LeakyReLU(0.1, inplace=True)
        self.m = nn.Sequential(*[Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)])

    def forward(self, x):
        y1 = self.cv3(self.m(self.cv1(x)))
        y2 = self.cv2(x)
        return self.cv4(self.act(self.bn(torch.cat((y1, y2), dim=1))))
```

Q: nn.Conv2d 里面，bias 的意思？作用?

bias 是针对每个输出通道加上的一组可学习的值（通常是一维的，形状为 [out_channels]）。
假设 input 尺寸是 3x3，in_channels=3

conv = nn.Conv2d(in_channels=3, out_channels=4, kernel_size=3, bias=True)
bias 是一组值（每个 out_channel 对应 1 个值，一共 4 个值）。 做完卷积后，每个 out_channel 的特征图里的 9 个像素值，逐一加上该通道对应的 bias 值。
作用：调整数值偏移，为后续激活创造更好的分布，提供更多的灵活性以拟合数据

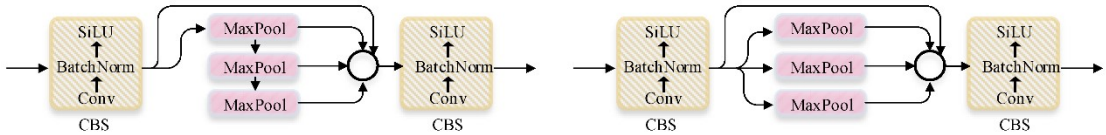
如果后面要加 batch norm，bias 应该设置为 false。因为 bn 里已经有偏移项参数。

1.2.4 SPP（Spatial Pyramid Pooling）

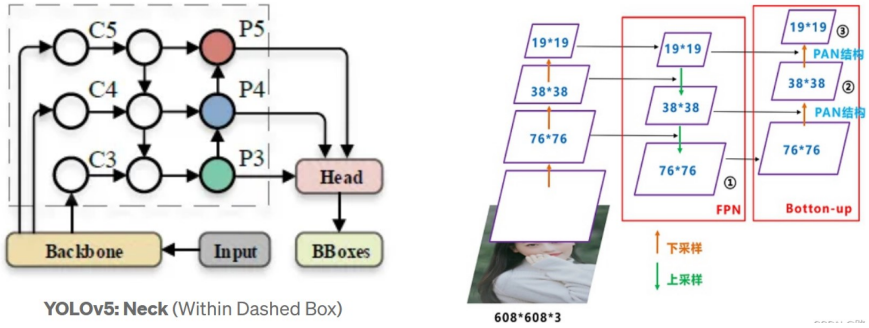
The purpose of SPP in YOLOv5 is to aggregate context at different scales and enhance the receptive field, which is beneficial for detecting objects of various sizes.
同 YOLO v4 一样!

【Update】SPPF replace SPP in v6.0

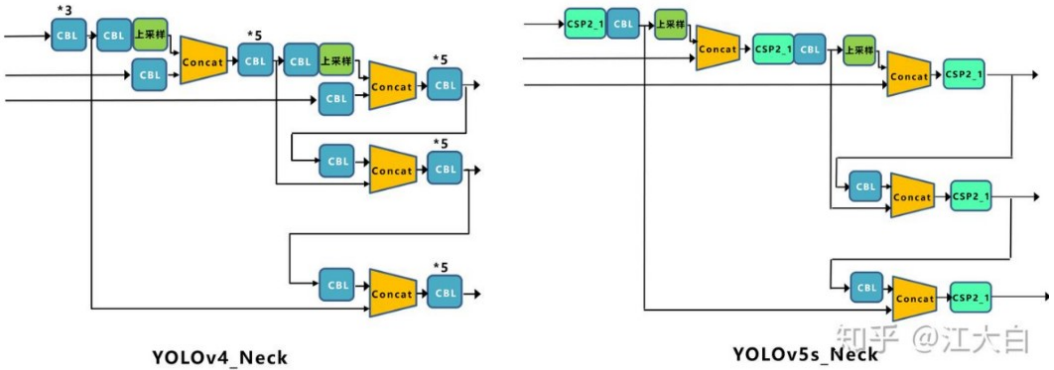
Optimized version of SPP: The **SPPF (Spatial Pyramid Pooling - Fast)** layer uses fewer resources while maintaining the benefits of multi-scale feature aggregation. is mathematically identical with less FLOPs.
用 maxpool (5, 1, 2)进行三次串行的 maxpooling 操作
效果没有变化，模型的计算量变小了很多，模型速度提升。



1.3 Neck



YOLO v5 uses the methods of FPN (top-down) and PAN (bottom-up), same as YOLO v4.



v4 的 Neck 中采用的都是普通的卷积操作; YOLO v5 的 Neck 结构中为什么要加入 CSP 结构?

为了加强网络特征融合的能力。

Densenet

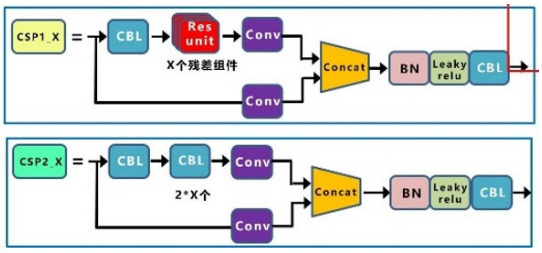
Concat add 区别 好处?

Neck 中为什么要用 CSP2_X，而不是 CSP1_X?

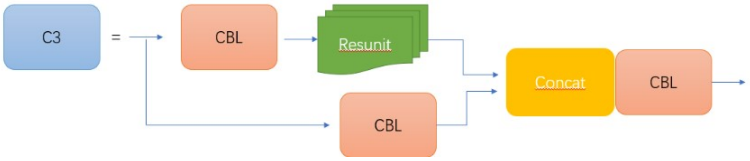
两者的区别在于：CSP2_X 将 Res unit 换成了 2 * X 个 CBL。因为 Neck 网络没那么深，通常 3-4 层，所以不需要残差模块。用**普通卷积堆叠**代替残差模块，进行轻量级的特征变换，强调**快速融合和多尺度信息传递**，避免过度抽象。

【Update】C3 modules replace CSP in v4.0

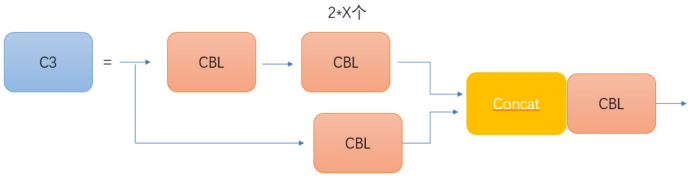
CSP1_X、CSP2_X →



C3 in backbone →



C3 in neck →



Why C3 modules?

Latest models are all slightly **smaller** due to removal of one convolution within each bottleneck, which have been renamed as C3() modules now in light of the 3 I/O convolutions each one does vs the 4 in the standard CSP bottleneck. The previous manual concatenation and LeakyReLU(0.1) activations have both removed, **simplifying the architecture**, **reducing parameter count**, and **better exploiting the fuse operation** at inference time.

In general, the changes result in **smaller models** (89.0M params -> 87.7M YOLOv5x), **faster inference times** (6.9ms -> 6.0ms), and **improved mAP** (49.2 -> 50.1) for all models except YOLOv5s, which reduced mAP slightly (37.0 -> 36.8). In general, the largest models benefit the most from this update.

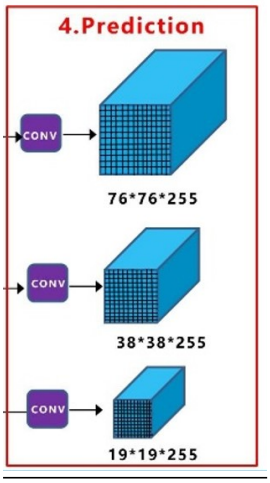
C3 modules 如何切换?

in models/yolov5s.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, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, 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, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, C3, [1024, False]],  # 9
  ]
```

If **no False**, enable res unit, is **CSP1_X**. If **False**, disable res unit, is **CSP2_X**.

1.4 Head



Output channel still 255, same as YOLO v3 and v4.

Each grid has **3** bounding boxes. Each box has **4** box coordinates (t_x, t_y, t_w, t_h) (the definition of box coordinates follows from YOLOv2), **1** objectness score, and **80** classification scores. Total = **3 x (4 + 1 + 80) = 255.**

$$
\begin{aligned}
b_x &= \sigma(t_x) \times 2 - 0.5 + c_x \\
b_y &= \sigma(t_y) \times 2 - 0.5 + c_y \\
b_w &= p_w(\sigma(t_w) \times 2)^2 \\
b_h &= p_h(\sigma(t_h) \times 2)^2
\end{aligned}
$$

约束条件变？

1.5 轻量级调整因子

在**.yaml** 文件中，引入两个参数 width_multiple，depth_multiple，来生成不同尺度的模型（YOLO v5 s/m/l/x）。

depth_multiple 调整模型的深度（层数）

```
c2 = make_divisible(c2 * gw, 8) if c2 != no else c2
```

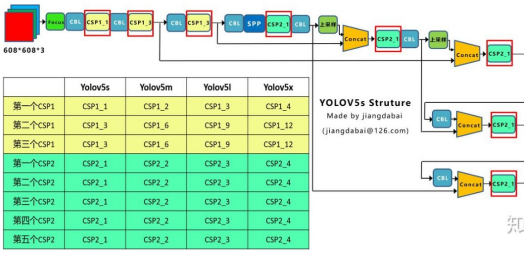
width_multiple 调整模型每一层的宽度（通道数）

```
n = max(round(n * gd), 1) if n > 1 else n  # depth gain
```

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

# anchors
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

# 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]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
  ]
```

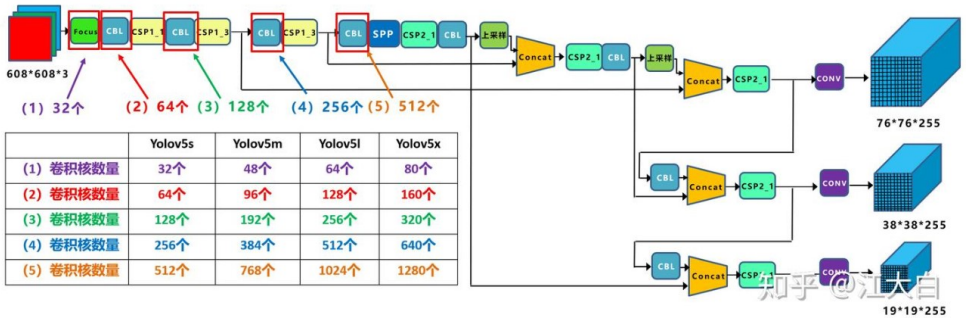
```yaml
# parameters
nc: 80  # number of classes
depth_multiple: 0.67  # model depth
width_multiple: 0.75  # layer channe

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

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/
   [-1, 1, Conv, [128, 3, 2]],  # 1-
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]], # 7-
   [-1, 1, SPP, [1024, [5, 9, 13]]],
  ]
```

2.3.3 Yolov5四种网络的深度



	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

2.3.4 Yolov5四种网络的宽度



	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

2. Training

2.1 自适应图片缩放

以前的做法是？

长*宽：800*600　　长*宽：416*416

自适应图片缩放的做法是：通过智能填充，在保持图像不变形的同时，将无效的计算区域最小化。在 utils/datasets.py 的 letterbox()中实现

无用的黑边变小了。目的是减少计算冗余，提升推理速度。

为什么 np.mod 函数用 32?
因为 Yolov5 的网络经过 5 次下采样，2^5=32。这样使得短边也能被 32 整除。

注意：**只在推理时采用**，训练时还是采用传统填充的方式（**Why?** ）
1. 保持训练数据的一致性，以便批量处理，高效地并行训练；确保特征图大小一致，便于 anchor 分配和损失计算。
2. 增强泛化能力。自适应填充反而可能让模型习惯于“干净图像”，降低对复杂场景的适应性。

2.2 Data Augmentation
Mosaic data augmentation
使用四张训练图像按一定比例组合成一张图像，使模型学会在更小的范围内识别对象。其次还有助于显著减少对 batch size 的需求。(Same as YOLO v4.0)

2.3 Auto Learning Bounding Box Anchors 自适应锚定框
YOLO v3 v4 都是先采用 k-means 聚类算法在数据集中预先训练得到 anchor 值
缺点：
1. 这些 anchor 值是针对**特定数据集**（COCO）优化的，不一定适合自己的数据集
2. 用户需要自己运行聚类脚本来计算自己数据集上的锚框尺寸，然后手动更新配置文件，过程繁琐且容易出错。

因此，YOLO v5 中将此功能嵌入到代码中，自动化得出适应数据集的 anchor 值

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

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

在 train.py 中，“noautoanchor”参数用来控制是否启用自适应 anchor 机制。
noautoanchor = False，则采用给定的**--anchors** 参数或配置文件里的参数
noautoanchor = Ture 则启用自适应 anchor 机制。在训练开始前，利用 **k-means** 聚类和遗传算法，基于当前训练数据集自动计算出最优的锚定框尺寸。

具体做法：
1. 加载数据集： 读取训练数据集中的所有真实目标框标注。
2. 运行聚类算法： 对数据集中所有 GT 的**宽和高**（注意：不是绝对坐标，是归一化后的宽高，相对于图像尺寸）运行 **k-means 聚类**。默认的 **k** 值是 9

3. 度量标准为遗传算法 + Best Possible Recall (BPR)： YOLOv5 的聚类目标函数不是欧式距离，而是基于 1 - IoU(anchor, gt)，目标是最大化平均 IoU / BPR。并且使用遗传算法（GA）来进化初始聚类中心。BPR 定义为数据集中能被某个锚框（以一定 IoU 阈值，默认 0.98）覆盖的真实框的比例，高 BPR 意味着预设的锚框能够很好地覆盖数据集中的各种目标形状。

4. 评估与决定（自适应）：

计算当前锚框配置在当前数据集上的 BPR 或 mean IoU。

如果当前锚框的 BPR 低于某个阈值（默认为 0.98），或者 mean IoU 低于某个阈值（默认为 0.60），YOLOv5 就认为当前锚框对于这个数据集不是最优的。

如果判定当前锚框不够好，YOLOv5 会自动丢弃预设的锚框，并用步骤 2 中聚类得到的新锚框替换它们。

5. 更新模型： 使用新计算出的、针对当前数据集优化的锚框尺寸开始正式训练。

2.4 正样本划分

(1) 跨 anchor 预测

假设一个 GT 框落在了某个预测分支的某个网格内，该网格具有 3 种不同大小 anchor，将 GT 框和这 3 种 anchor 采用 **shape 规则**匹配，匹配到的 anchor 都可以来预测该 GT 框，即一个 GT 框可以使用多种 anchor 来预测。

shape 规则：该 GT 框和当前层的 anchor 框计算宽高比。若宽高比例小于设定阈值，则认为该 anchor 与 GT 匹配成功，被视为正样本。如果大于设定阈值，则说明该 GT 框和该 anchor 框匹配度不够，在该层预测中认为是背景，为负样本。

(2) 跨 grid 预测

每个 GT 框中心点所在的网格被选为预测位置。为了增加正样本数量，还会选取该网格周围最近的两个邻近网格；所以每个 GT 框最多可由 **3 个网格**负责预测。

所以，对于每个 GT，在每一预测层上保留的 anchor 总数在不考虑越界情况下是 **3 或者 6 或者 9。**

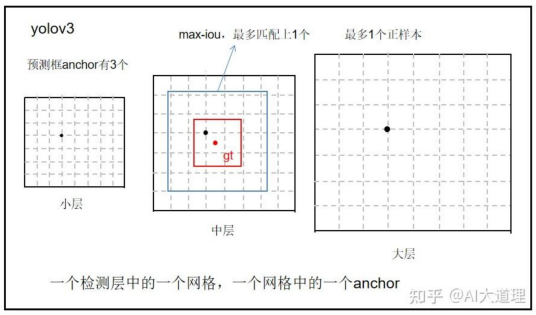
(3) 跨预测层

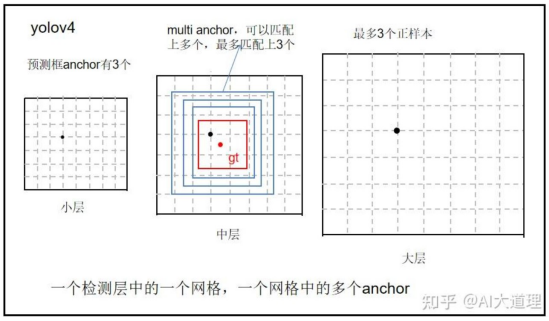
YOLOv5 有三个不同尺度的预测分支（对应不同的特征图层），对每个预测分支，都进行（1）（2）操作。

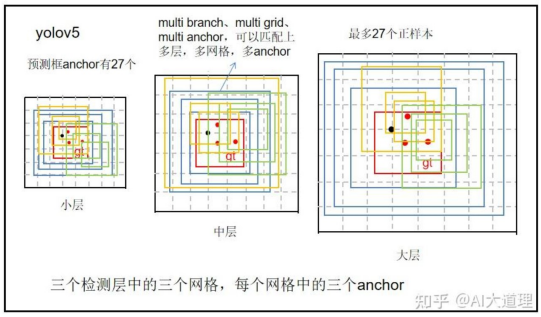
如果一个 GT 框能与多个分支上的 anchor 匹配成功，则这些分支都可以预测该 GT

因此：一个 GT 框最多可以匹配到 **3(分支)× 3(网格) × 3 (anchor) = 27 个正样本**

Q 和之前的 YOLO 系列有什么不同？







在一个预测层里，一个 GT 匹配的 anchor 数范围最多是 9 个；还允许跨层预测。一个 GT 最多可以匹配 27 个正样本。大大增加了正样本的数量。（**好处**）

不过，这种特别暴力增加正样本做法还是存在很大弊端：虽然可以加速收敛，但是由于引入了很多低质量 anchor，对最终结果还是有影响的。

2.5 Loss function

确定正样本之后会生成一个 **mask 掩码矩阵，用于损失函数计算**。**mask 掩码矩阵**是一个布尔矩阵，形状为(batch_size, num_anchors)，用于标记哪些 anchor 是正样本（True），哪些是负样本（False）。

Total Loss = λ_box * **Bbox_Loss** + λ_obj * **Obj_Loss** + λ_cls * **Cls_Loss**
λ_box, λ_obj, λ_cls 是超参数，用于平衡不同任务的权重。默认值分别为 0.05, 1.0, 0.58

回归损失和**分类损失**只对 mask 为 True 的 anchor 计算；**置信度损失**则对所有 anchor 计算，但正负样本标签不同。

Bbox_Loss: CIOU

- CIOU
  "a good loss for bounding box regression should consider three important geometric factors, i.e., overlap area, central point distance and aspect ratio"

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

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

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

$$L_{CIoU} = 1 - IoU + \rho 2(b, bgt)c2 + \alpha v$$

综合考虑了重叠面积、中心点距离、宽高比
ρ为框 A 和框 B 的中心点距离
c 为框 A 和框 B 的最小包围矩形的对角线长度
v 为框 A、框 B 的宽高比相似度，取值范围为 0~1，当宽高比相等时 v 取 0，当宽高比相差无限大时 v 取 1
α为 v 的**影响因子，** IOU 越大也即 A、B 的重叠区域越大，则α越大，从而 v 的影响越大；反之 IOU 越小也即 A、B 的重叠区域越小，则α越小，从而 v 的影响越小。
当框 A、框 B 的距离无限远，且宽高比差别无限大时 DIOU 取-1，v 取 1，alpha 取 0.5，此时 CIOU 取-1-0.5=-1.5；当框 A、框 B 完全重叠时，DIOU 取 1，v 取 0，α取 0，则 CIOU 取 1。因此 **CIOU 取值范围是-1.5~1**。

Obj_Loss:
使用 Binary Cross Entropy（BCE）Loss 来计算置信度损失。
Obj_Loss = BCE(pred_confidence, target_confidence)

$$L_{obj} = -\left[t \cdot \log(\sigma(p)) + (1-t) \cdot \log(1-\sigma(p))\right]$$

所有正负样本的 anchor 都参与计算，但标签不同。

yolo 之前版本直接对 mask 矩阵为 true 的 anchor 赋值 1， false 赋值 0，认为只要 mask 为 true 就表示对应预测框完美包围了目标。这样太绝对了。

YOLO v5 的做法：
mask = True 的 anchor：不直接赋值为 1，而是将 CIOU 值（范围 0~1）作为置信度标签；CIOU 越高 → 框越准确 → 标签越接近 1。
mask = False 的 anchor：置信度标签直接赋值为 0。

这样做，标签值的大小与预测框、目标框的重合度有关，重合度越高则标签值越大。
注意：CIOU 的取值范围是-1.5~1，而置信度标签的取值范围是 0~1，所以需要对 CIOU 做一个截断处理：当 CIOU 小于 0 时直接取 0 值作为标签。

Classification Loss:
默认使用 binary cross-entropy (Same as YOLO v3)

$$\mathrm{y_i} = \mathrm{Sigmoid}(\mathrm{x_i}) = \frac{1}{1+\mathrm{e}^{-\mathrm{x_i}}}$$

$$\mathrm{L_{class}} = -\sum_{\mathrm{n=1}}^{\mathrm{N}} \mathrm{y_i^*}\,\log(\mathrm{y_i}) + (1-\mathrm{y_i^*})\log(1-\mathrm{y_i})$$

2.6

六、训练策略

（1）多尺度训练（Multi-scale training）。 如果网络的输入是416 x 416。那么训练的时候就会从0.5 x 416 到 1.5 x 416 中任意取值，但所取的值都是32的整数倍。

（2）训练开始前使用 warmup 进行训练。 在模型预训练阶段，先使用较小的学习率训练一些epochs或者steps (如4个 epoch 或10000个 step)，再修改为预先设置的学习率进行训练。

（3）使用了 cosine 学习率下降策略（Cosine LR scheduler）。

（4）采用了 EMA 更新权重(Exponential Moving Average)。 相当于训练时给参数赋予一个动量，这样更新起来就会更加平滑。

（5）使用了 amp 进行混合精度训练（Mixed precision）。 能够减少显存的占用并且加快训练速度，但是需要 GPU 支持。

（5）

Reference

Releases · ultralytics/yolov5

Brief Review: YOLOv5 for Object Detection | by Sik-Ho Tsang | Medium

(39 封私信 / 15 条消息) Yolov5 总结文档(理论、代码、实验结果) - 知乎

(39 封私信 / 17 条消息) 深入浅出 Yolo 系列之 Yolov5 核心基础知识完整讲解 - 知乎

(39 封私信 / 17 条消息) 一文读懂 YOLO V5 与 YOLO V4 - 知乎

(39 封私信 / 17 条消息) 进击的后浪 yolov5 深度可视化解析 - 知乎

yolov5 中的 Focus 模块的理解_focus 结构-CSDN 博客

YOLOv5 Focus() Layer · ultralytics/yolov5 · Discussion #3181

yolov5 源码解析(0)--focus 层哪去了？ yolov5 为什么不要 focus 层了-CSDN 博客

原始 SPP 及在 YOLO 中的 SPP/SPPF 对比详解_sppf 和 spp-CSDN 博客

Understanding SPP and SPPF implementation · Issue #8785 · ultralytics/yolov5

YOLOv5 中的 CSP 结构-CSDN 博客

AI 大视觉（十八） | Yolo v5 的改进思想 - AI 大道理 - 博客园

(39 封私信 / 21 条消息) yolov5 目标检测神经网络——损失函数计算原理 - 知乎