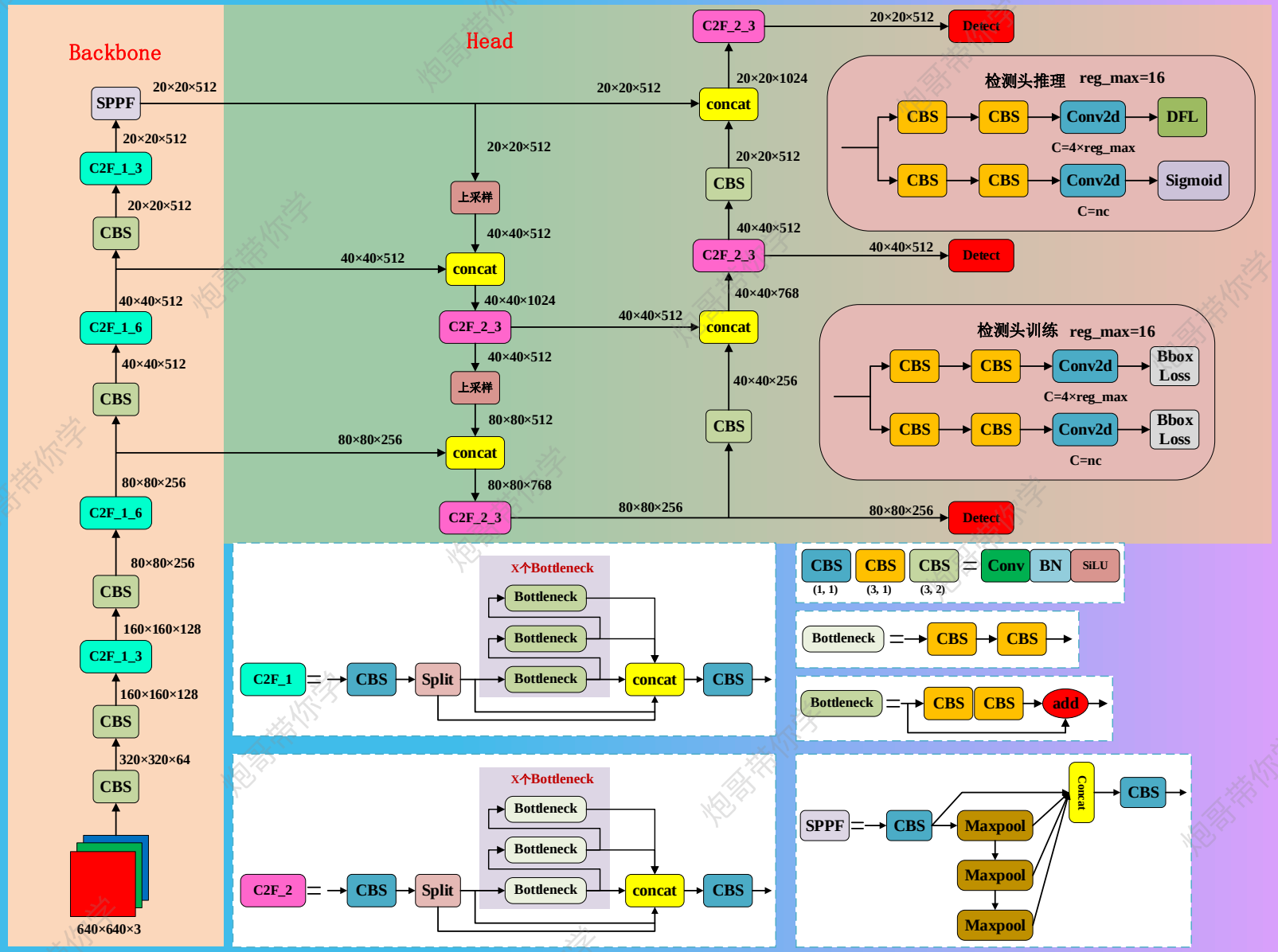
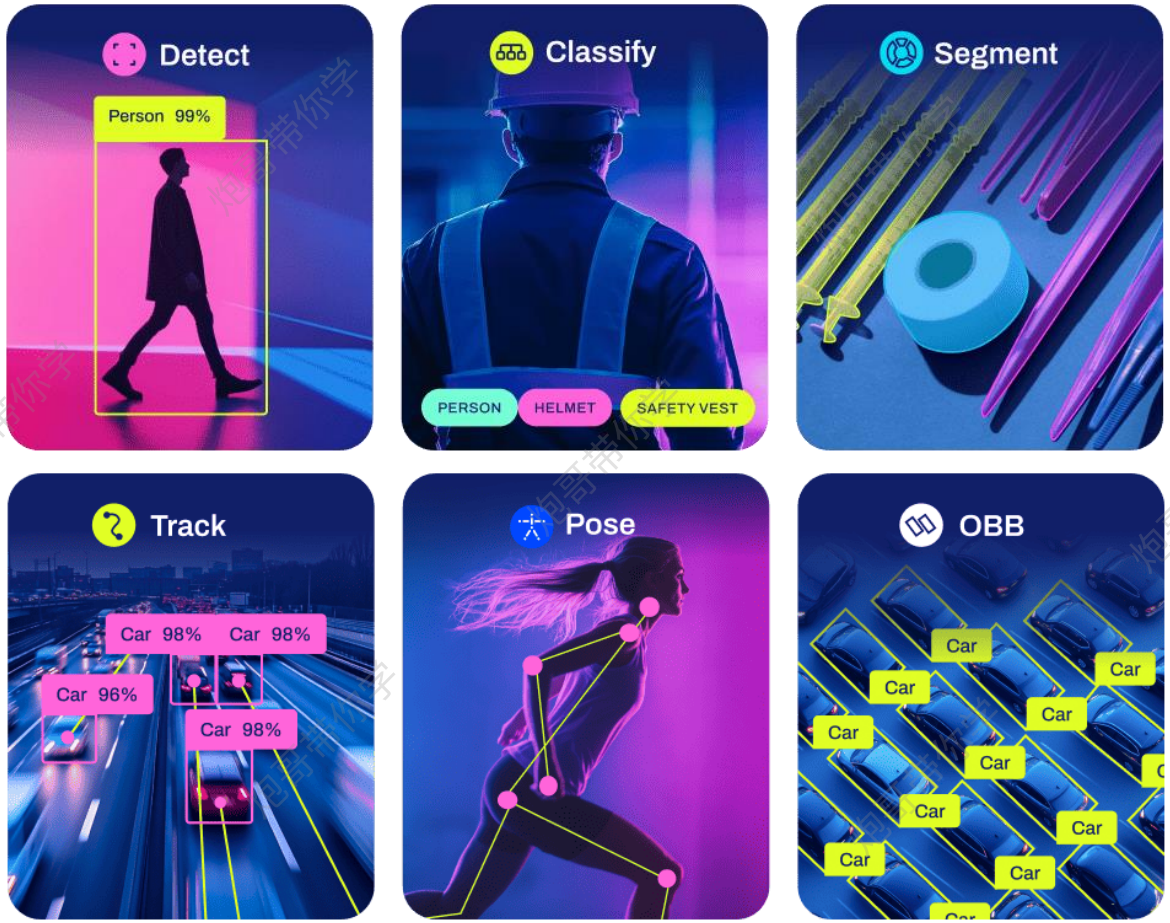


第2章 Yolov8 目标检测 算法原理

Yolov8算法的背景



YOLOv8是Ultralytics公司（发布yolov5的公司）在YOLO系列基础上进行优化的结果，发布于2023年 1月。它是当前 YOLO系列模型中最新且性能最强的一款，具备更高的速度、准确性，并支持多任务。

Backbone:

骨干网络和 Neck 部分可能参考了 YOLOv7 ELAN 设计思想，将 YOLOv5 的 C3 结构换成了梯度流更丰富的 C2f 结构，并对不同尺度模型调整了不同的通道数。

Head：

Head部分较yolov5而言有两大改进：

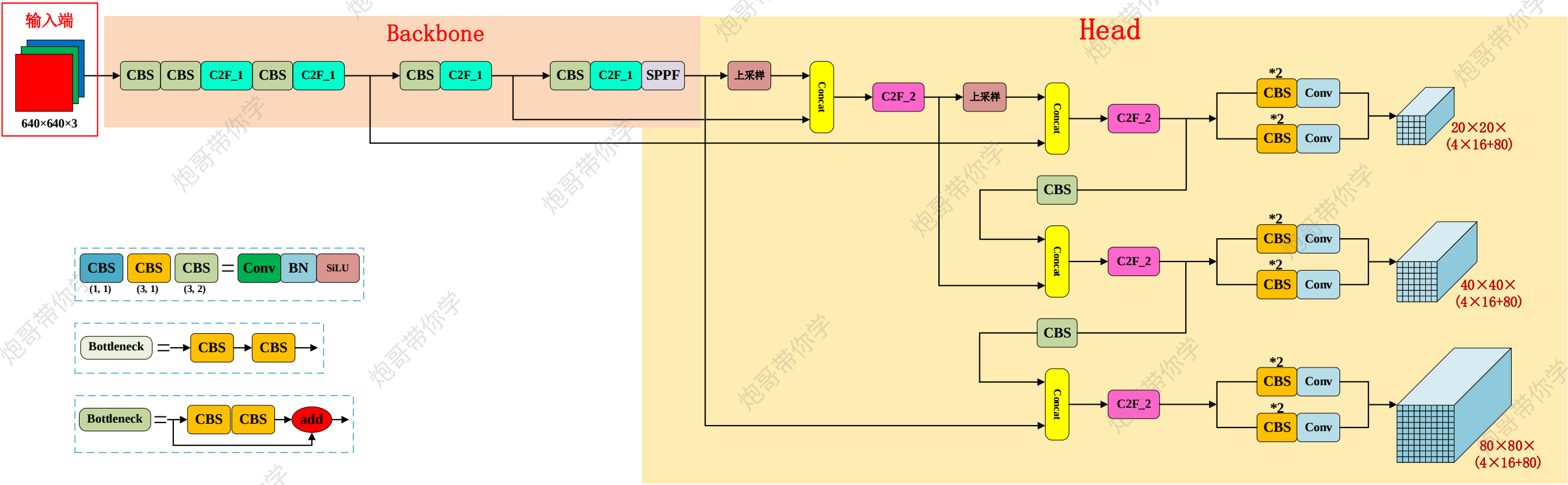
1）换成了目前主流的解耦头结构(Decoupled-Head)，将分类和检测头分离

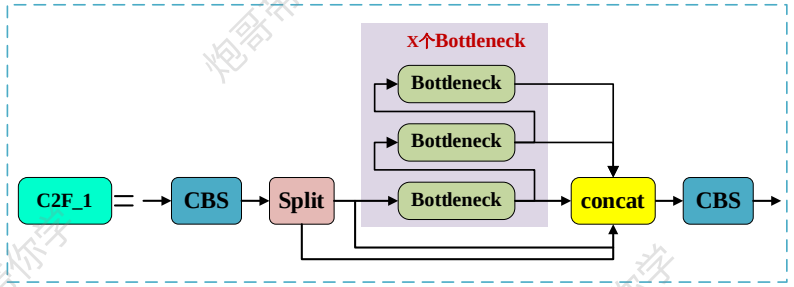
2）同时也从 Anchor-Based 换成了 Anchor-Free

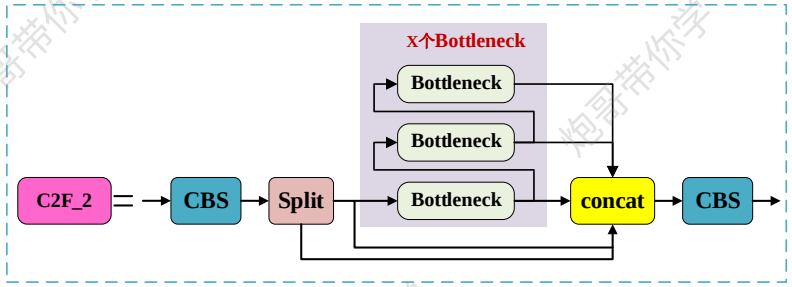
Loss：

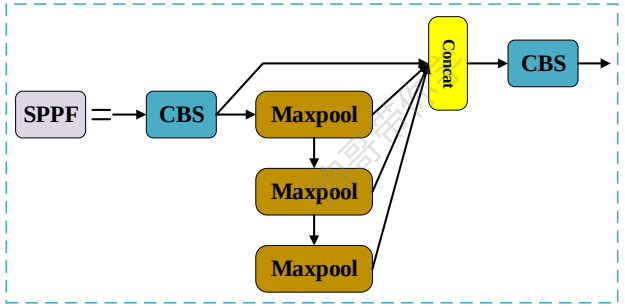
YOLOv8抛弃了以往的IOU匹配的分配方式，而是使用了Task-Aligned Assigner (TAA)正负样本匹配方式。并引入了 Distribution Focal Loss(DFL)

Yolov8整体网络结构

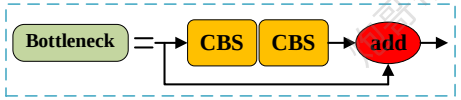


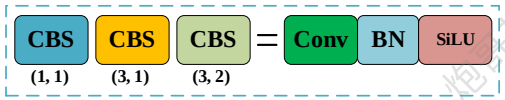


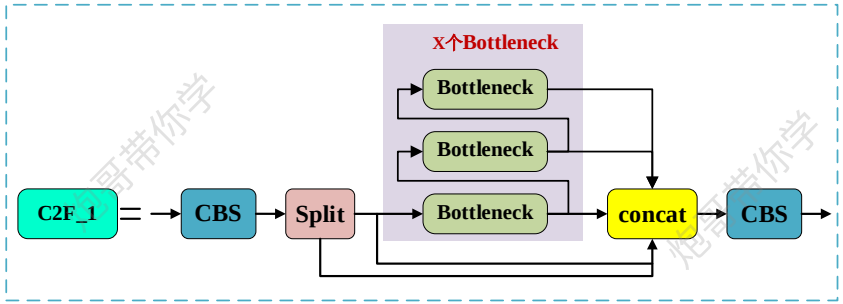


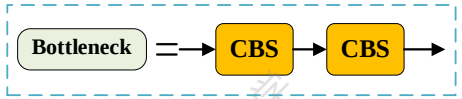


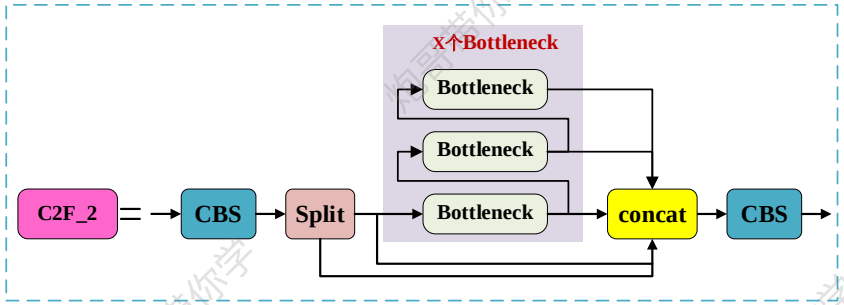
Yolov8网络结构代码解读

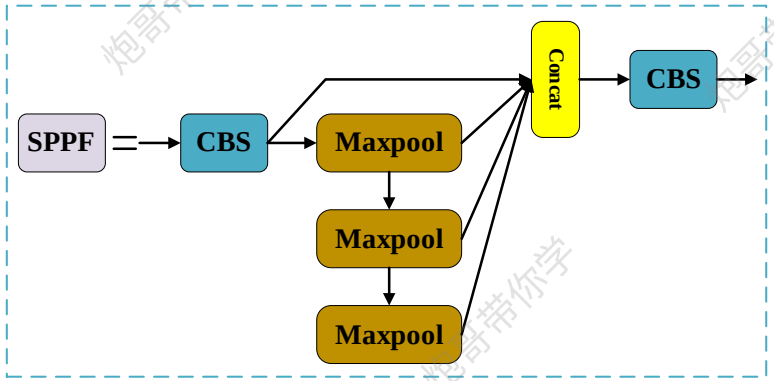













```yaml
nc: 80 # number of classes
scales: # model compound scaling constants, i.e. 'model=
  # [depth, width, max_channels]
  n: [0.33, 0.25, 1024] # YOLOv8n summary: 225 layers
  s: [0.33, 0.50, 1024] # YOLOv8s summary: 225 layers, 1
  m: [0.67, 0.75, 768] # YOLOv8m summary: 295 layers, 25
  l: [1.00, 1.00, 512] # YOLOv8l summary: 365 layers, 43
  x: [1.00, 1.25, 512] # YOLOv8x summary: 365 layers, 68

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P
  - [-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, [1024, 3, 2]] # 7-P5/32
  - [-1, 3, C2f, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9
```

几个不同大小的yolov8对应的型号，其中不同的型号对应的深度和宽度缩放系数都不一样，最大通道数都不一样。

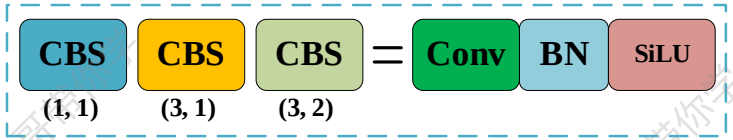
该层的输入特征来自哪层

该层模块重复次数

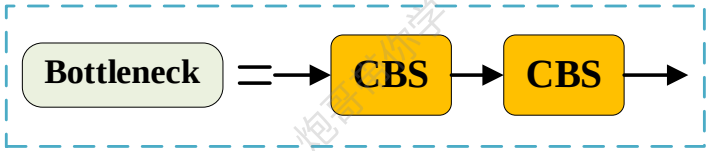
模块对应的参数

模块名称

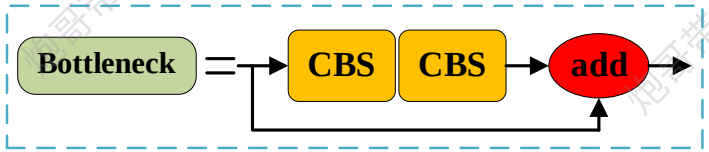
Yolov8中的Bottleneck模块



3种参数的CBS



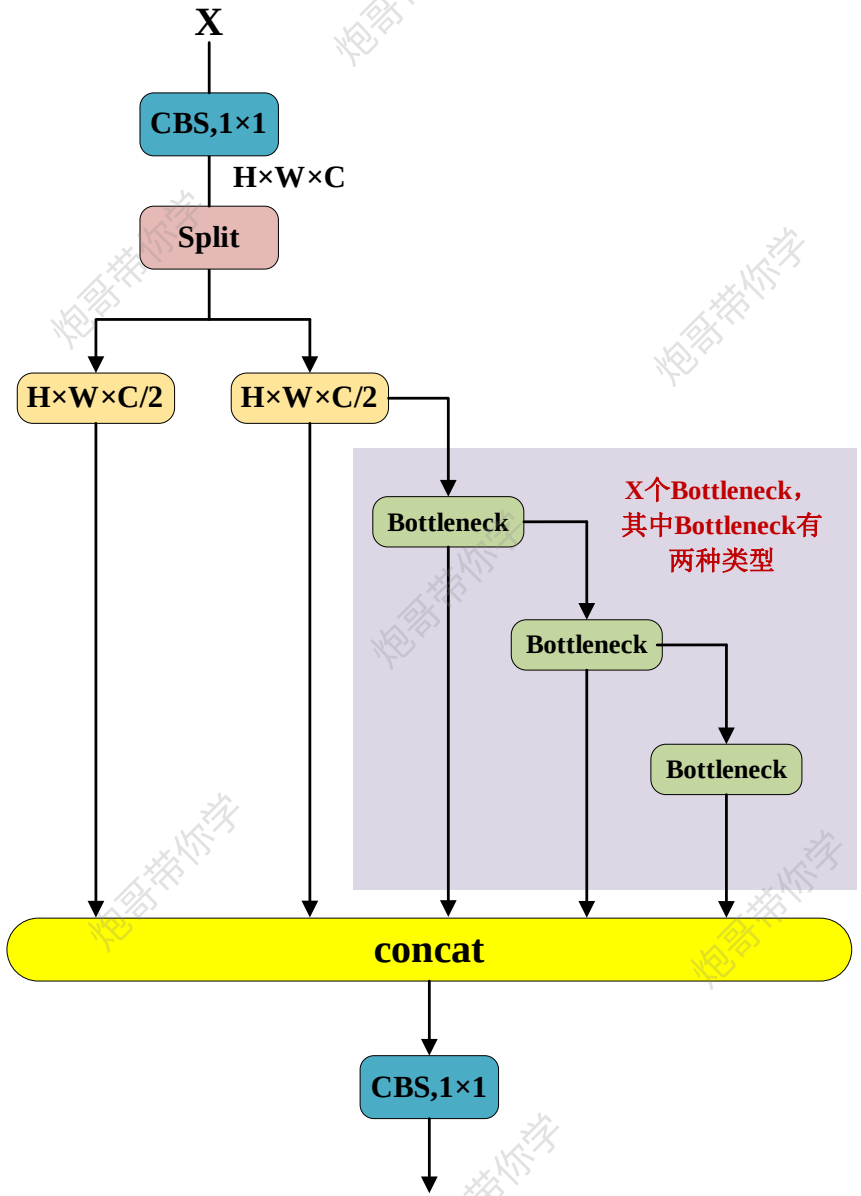
不带残差的Bottleneck

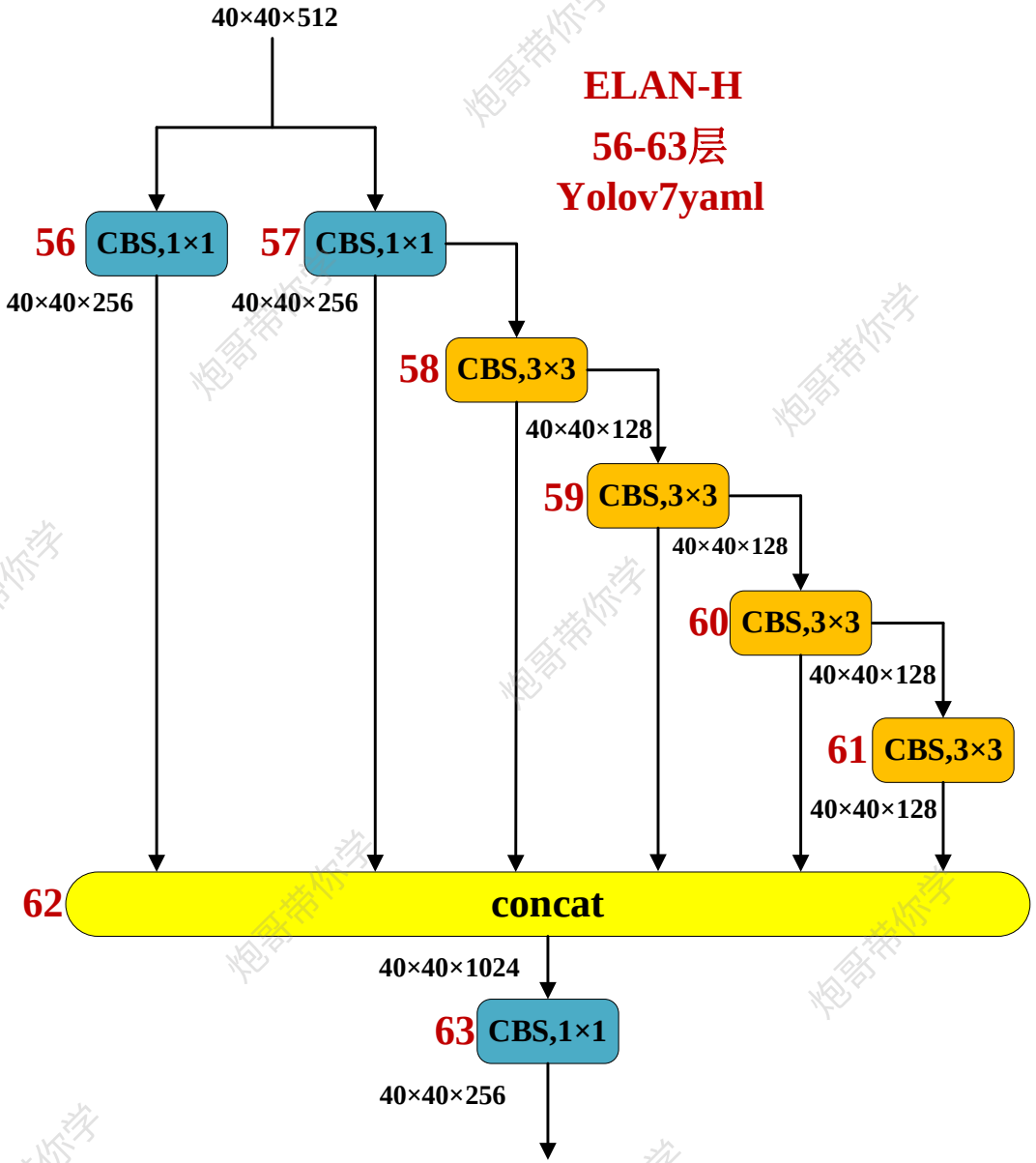


带残差的Bottleneck

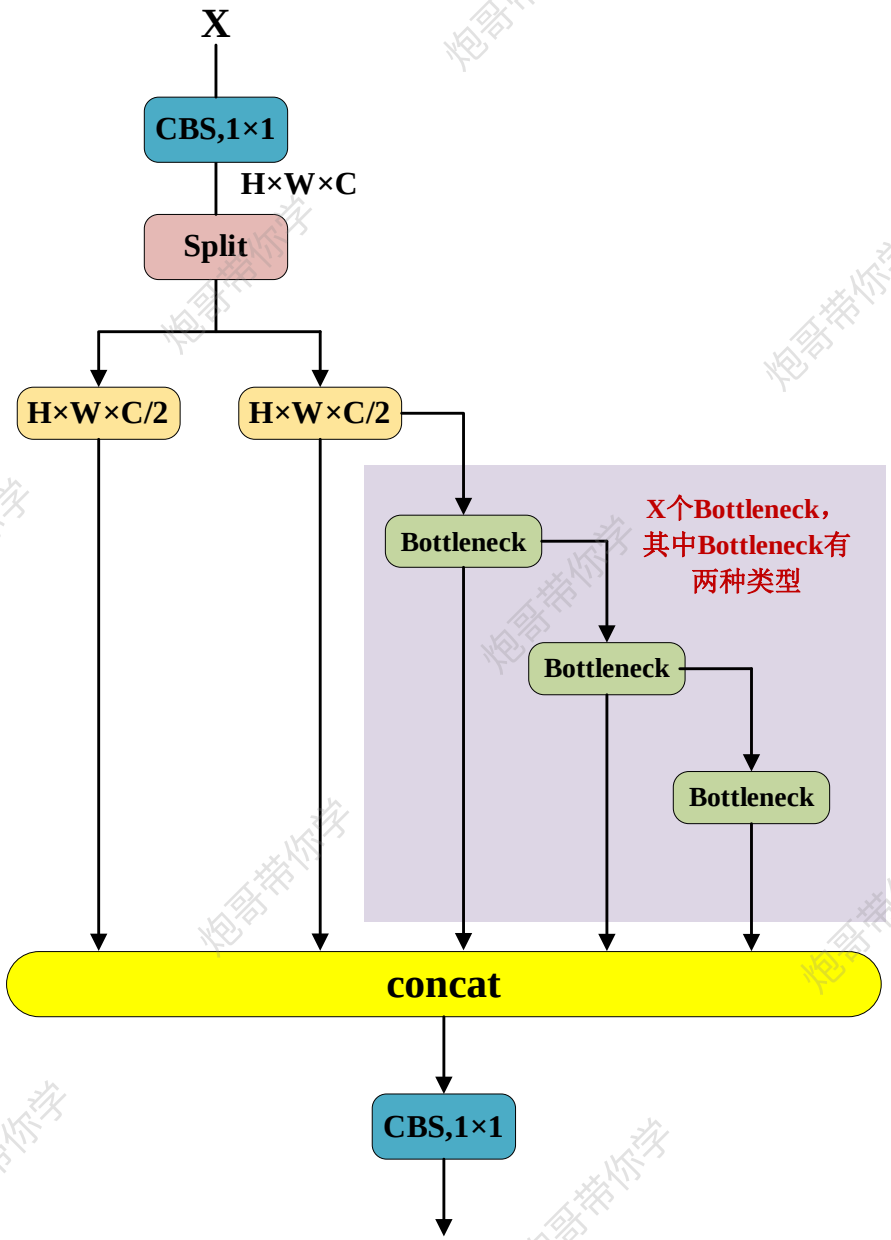
```python
class Bottleneck(nn.Module):
    """Standard bottleneck."""

    def __init__(self, c1, c2, shortcut=True,
                 g=1, k=(3, 3), e=0.5):
        """Initializes a standard bottleneck module with
        optional shortcut connection
        and configurable parameters."""
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, k[0], s: 1)
        self.cv2 = Conv(c_, c2, k[1], s: 1, g=g)
        self.add = shortcut and c1 == c2

    def forward(self, x):
        """Applies the YOLO FPN to input data."""
        return x + self.cv2(self.cv1(x)) if self.add \
            else self.cv2(self.cv1(x))
```

Yolov8中的C2F模块和v7中ELAN-H模块对比



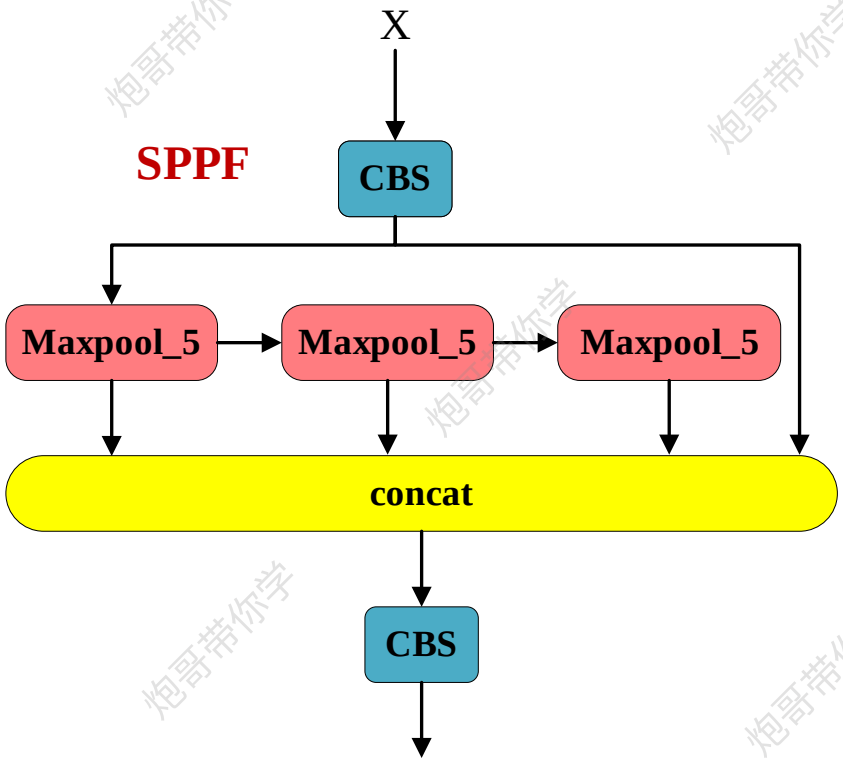


Yolov8中的C2F模块

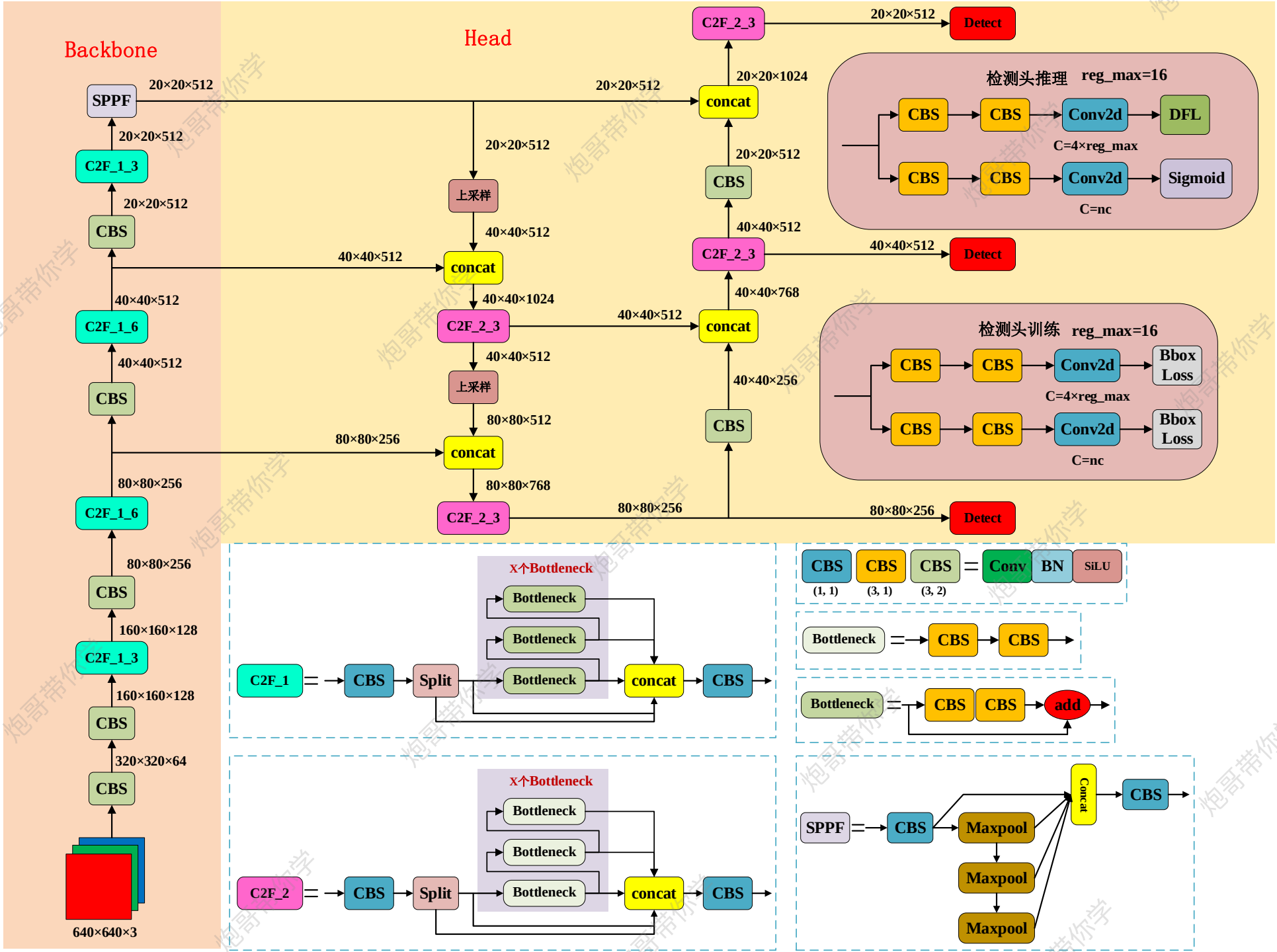



```python
class C2f(nn.Module):
    """Faster Implementation of CSP Bottleneck with 2 convolutions."""
    def __init__(self, c1, c2, n=1, shortcut=False, g=1, e=0.5):
        """Initializes a CSP bottleneck with 2 convolutions and n Bottleneck
        super().__init__()
        self.c = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, 2 * self.c, k: 1, s: 1)
        self.cv2 = Conv((2 + n) * self.c, c2, k: 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):
        """Forward pass through C2f layer."""
        y = list(self.cv1(x).chunk(2, 1))
        y.extend(m(y[-1]) for m in self.m)
        return self.cv2(torch.cat(y, dim: 1))
    1 usage
    def forward_split(self, x):
        """Forward pass using split() instead of chunk()."""
        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, dim: 1))
```

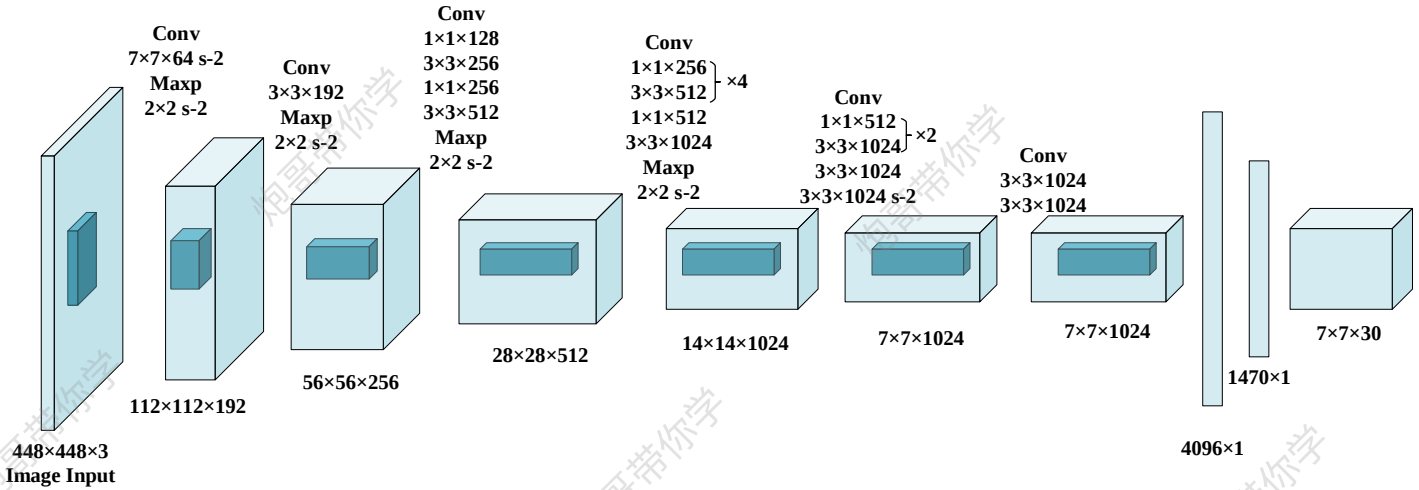
Yolov8算法中的SPPF

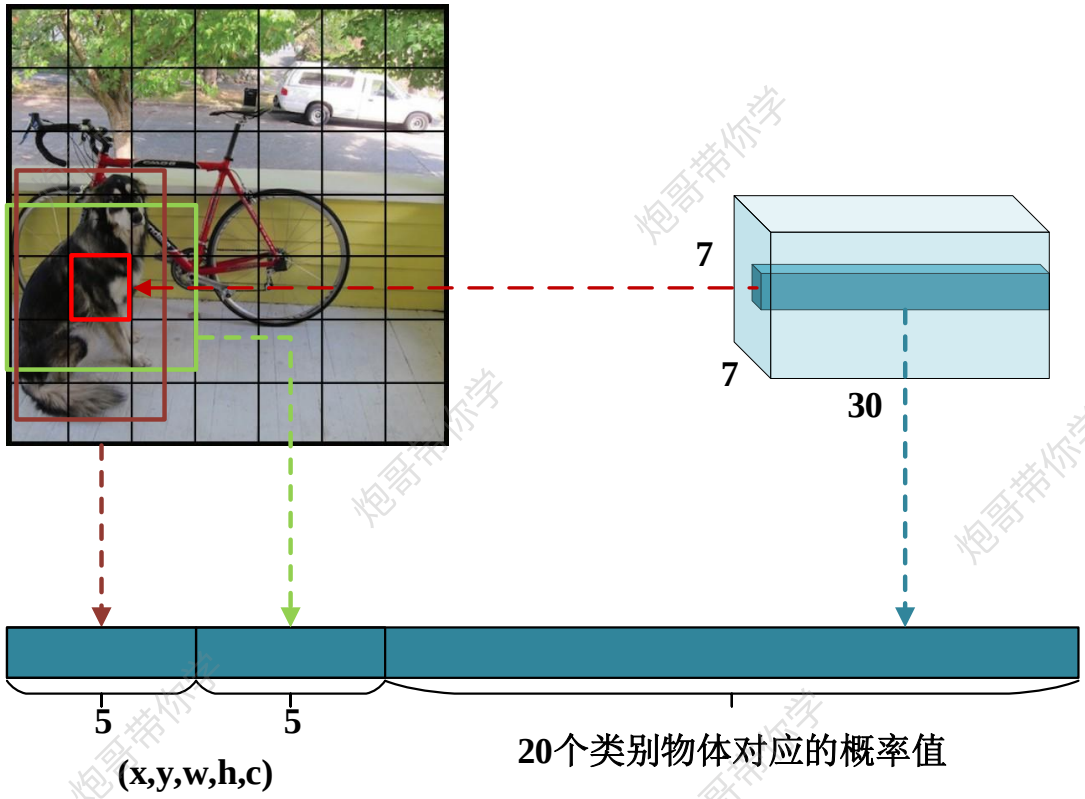



```python
class SPPF(nn.Module):
    """Spatial Pyramid Pooling - Fast (SPPF) layer for YOLOv5 by Glenn
    def __init__(self, c1, c2, k=5):
        """
        Initializes the SPPF layer with given input/output channels an
        This module is equivalent to SPP(k=(5, 9, 13)).
        """
        super().__init__()
        c_ = c1 // 2  # hidden channels
        self.cv1 = Conv(c1, c_, k: 1, s: 1)
        self.cv2 = Conv(c_ * 4, c2, k: 1, s: 1)
        self.m = nn.MaxPool2d(kernel_size=k, stride=1, padding=k // 2)

    def forward(self, x):
        """Forward pass through Ghost Convolution block."""
        y = [self.cv1(x)]
        y.extend(self.m(y[-1]) for _ in range(3))
        return self.cv2(torch.cat(y, dim: 1))
```


Backbone
Head
640×640×3
320×320×64
160×160×128
80×80×256
40×40×512
20×20×512
SPPF
C2F_1_3
C2F_1_6
C2F_2_3
CBS
上采样
concat
Detect
20×20×1024
40×40×768
80×80×512
80×80×768
检测头推理 reg_max=16
Conv2d
DFL
Sigmoid
C=4×reg_max
C=nc
检测头训练 reg_max=16
Bbox Loss
X个Bottleneck
Bottleneck
C2F_1
C2F_2
Split
BN
SiLU
Conv
(1, 1)
(3, 1)
(3, 2)
add
Maxpool
Concat

yolov1的Anchor-Free





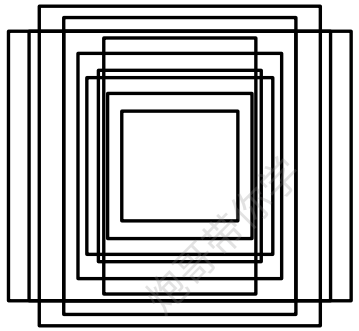
对于每个grid cell：

（1）预测B个边界框（在yolov1中选择2个边界框作为预测结果）。

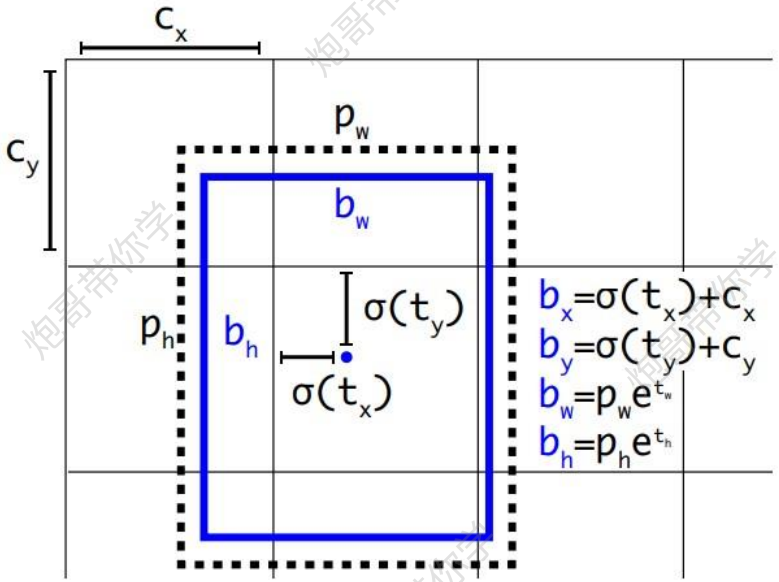
（2）每个边界框包含5个元素，分别为，中心坐标：x，y；检测框的宽和高：w，h；对应检测框的置信度c

Anchor-Based

Anchor-based 方法 是目标检测算法中常用的一种框架，尤其在 YOLO、SSD 和 Faster R-CNN 等经典方法中得到了广泛应用。其基本思想是通过预定义的一组 **锚框（Anchor Boxes）**，将物体检测问题转化为一个回归问题，使得模型可以直接预测这些锚框的 **偏移量** 和 **类别**，从而得到最终的边界框（Bounding Box）及其分类结果。



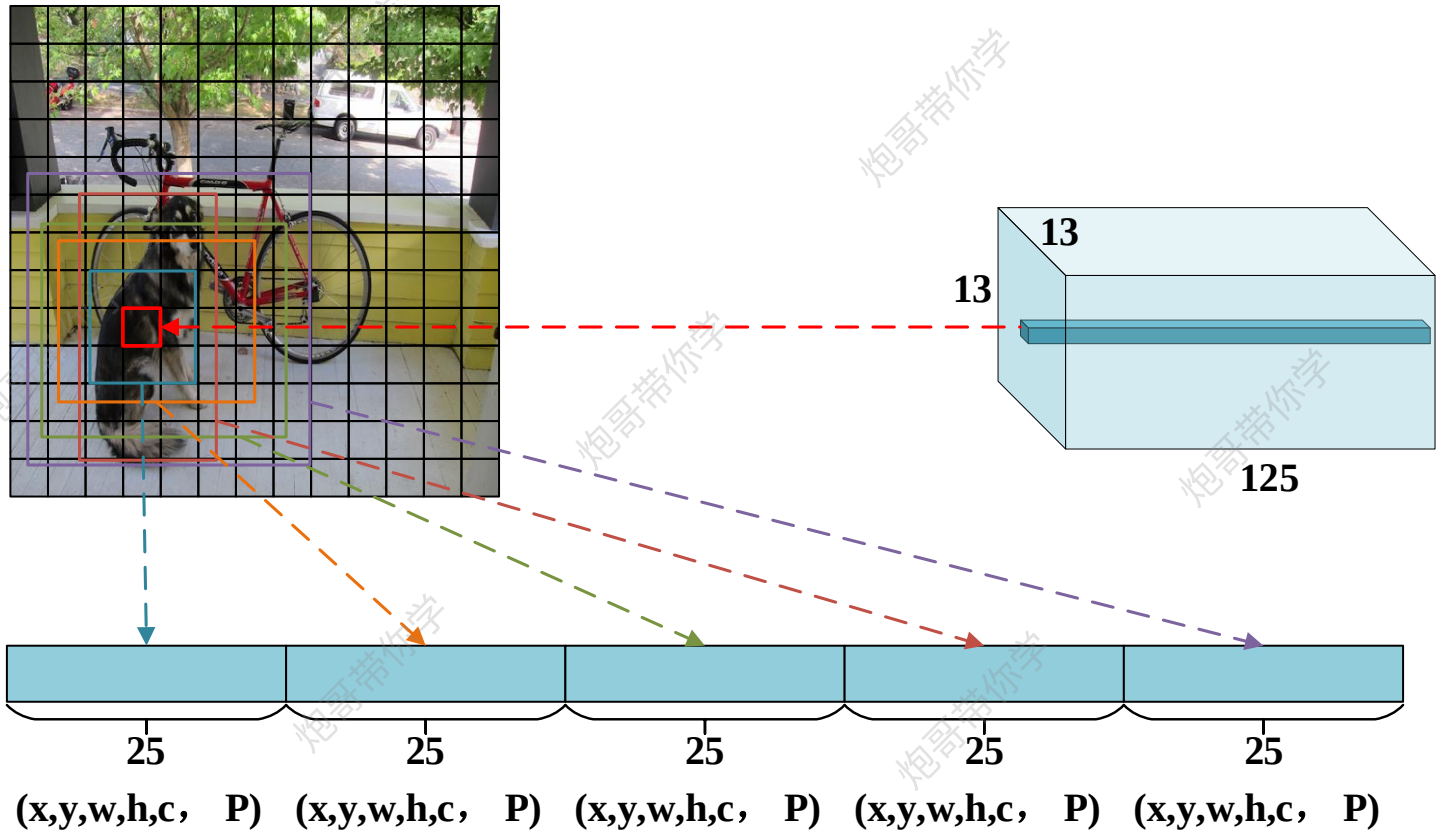
锚框是一些事先设定的矩形框，它们的形状和大小可以根据训练数据集中的物体尺寸来选择（通常通过 K-means 聚类等方法获得）。锚框并不直接对应真实物体，而是作为候选框存在，算法通过学习如何调整锚框来逼近物体的真实边界框。



Anchor-based 预测框的工作原理：

在 Anchor-based方法中，目标检测网络会在每个网格单元中为每个锚框预测以下几个参数：位置偏移（Bounding Box Regression）：通过回归来预测锚框与真实物体框之间的偏移量，通常包括中心坐标（tx, ty）和框的宽度（tw）与高度（th）的调整。分类：预测每个锚框是否包含物体，并为每个锚框预测物体的类别。

Anchor-Based



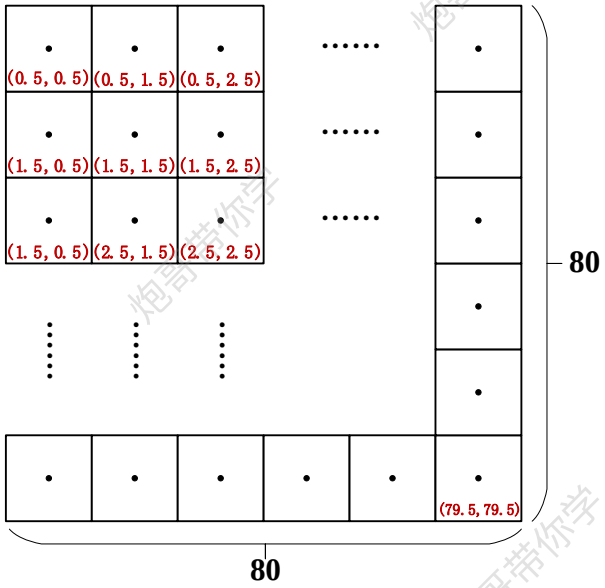
①Anchor的设置需要手动去设计(长宽比，尺度大小，anchor的数量），对不同数据集也需要不同的设计，相当麻烦。

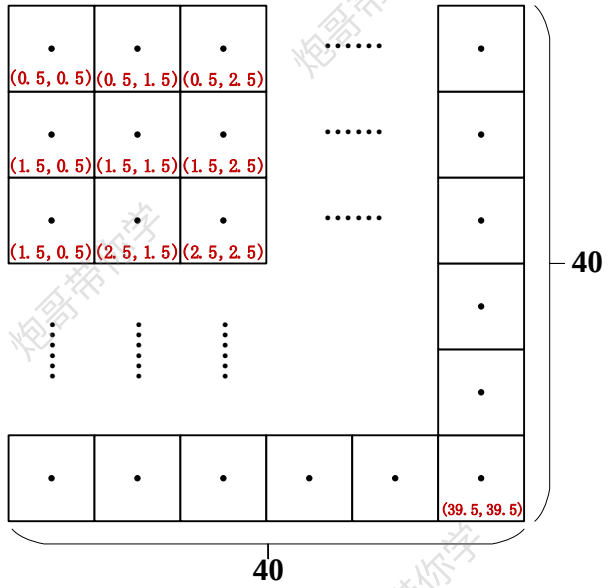
②Anchor的匹配机制使得极端尺度(特别大/小的object)被匹配到的频率，相对于大小适中的object被匹配到的频率更低，网络在学习时不容易学习这些极端样本。

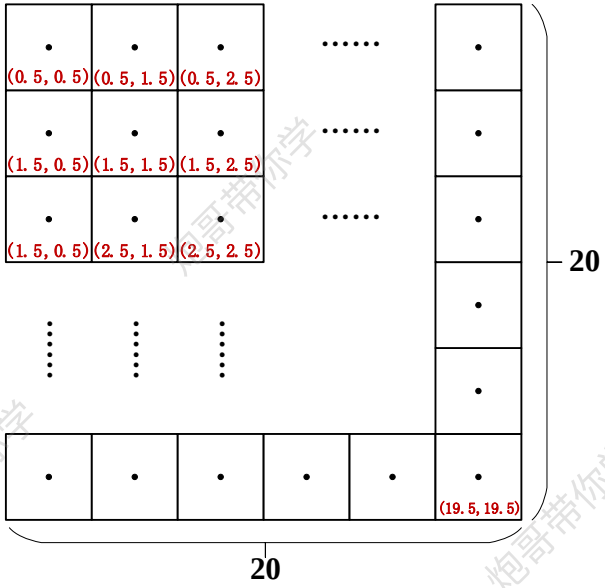
③Anchor的庞大数量使得存在严重的不平衡问题，涉及到采样、聚类的过程。但聚类的表达能力在复杂情况下是有限的。

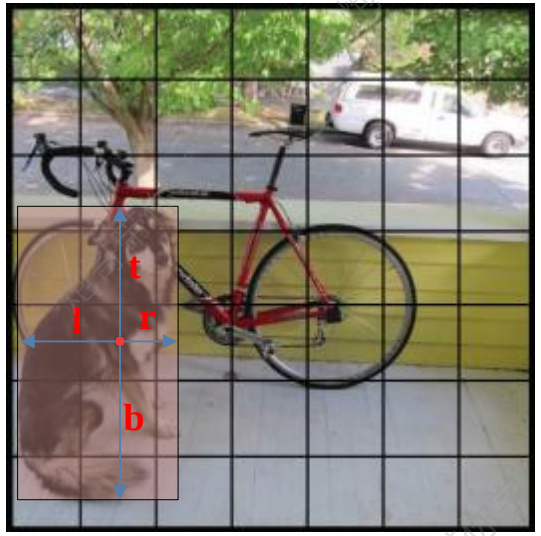
④Anchor-Based为了兼顾多尺度下的预测能力，推理得到的预测框也相对较多，在输出处理时的nms计算也会更加耗时。

yolov8的Anchor-Free









因为在输入数据大小为640的情况下最后3个检测头输出特征图大小为80×80，40×40，20×20；预测框数量为80×80+ 40×40+ 20×20=8400。

其中：

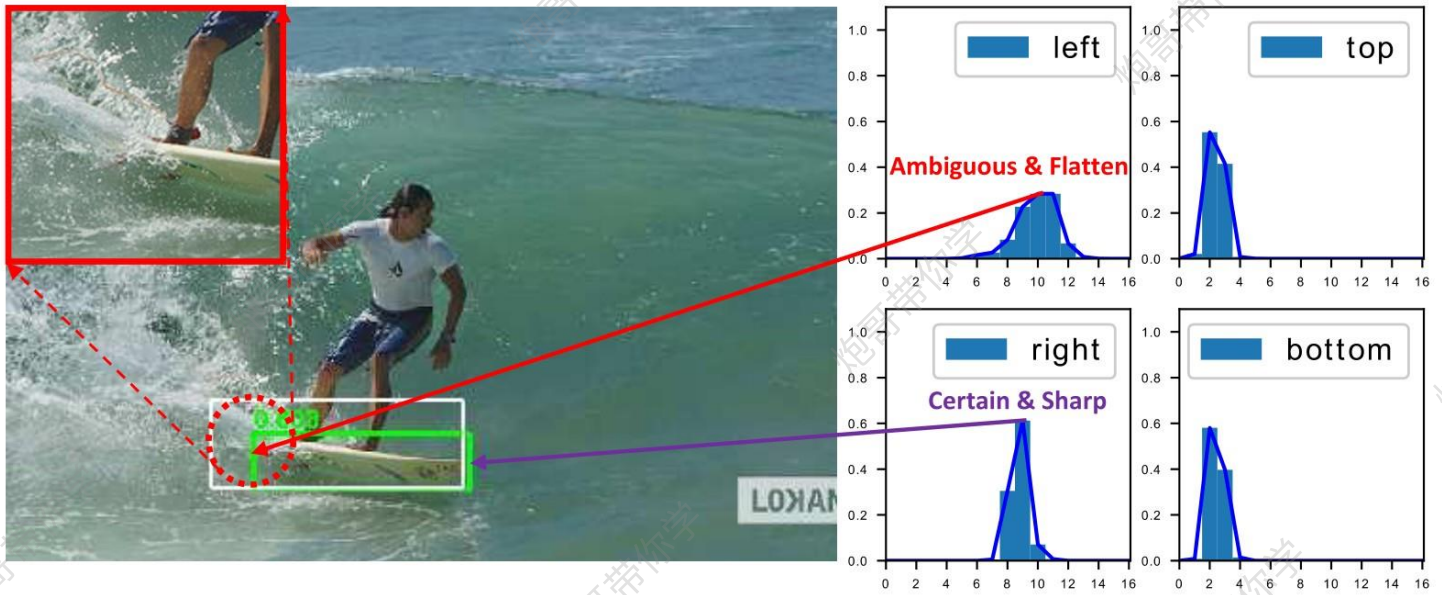
l：anchor point预测到左边框的距离

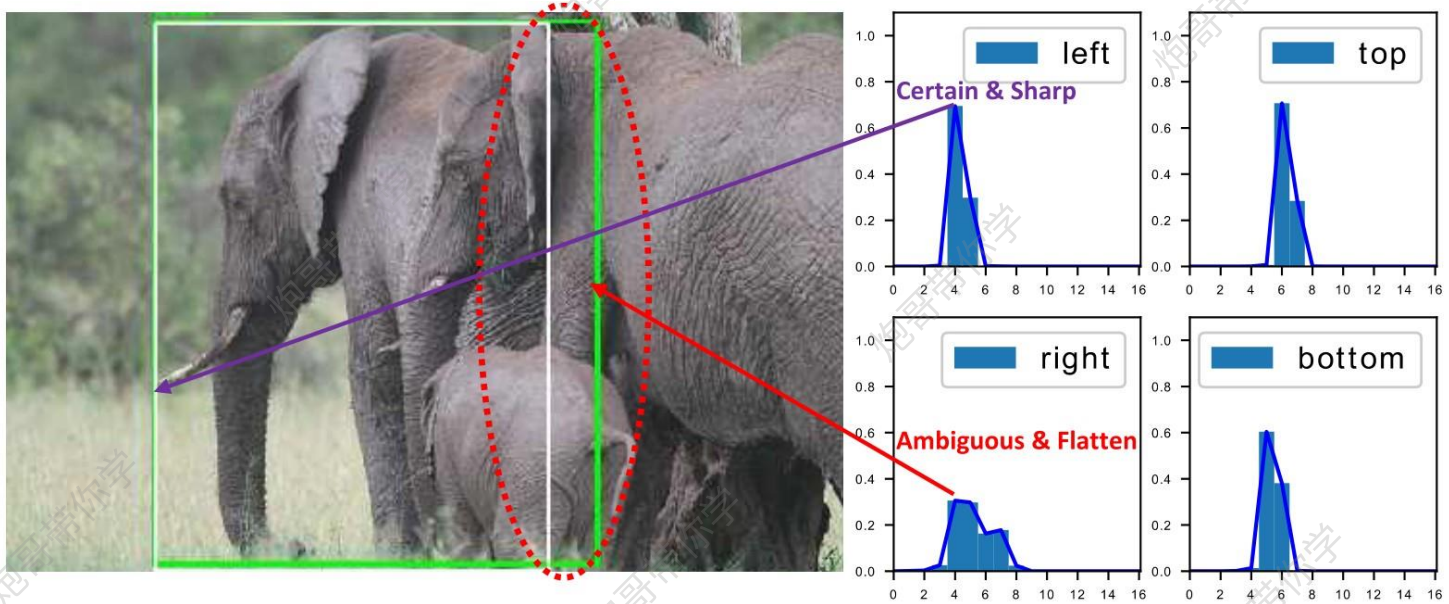
r：anchor point预测到右边框的距离

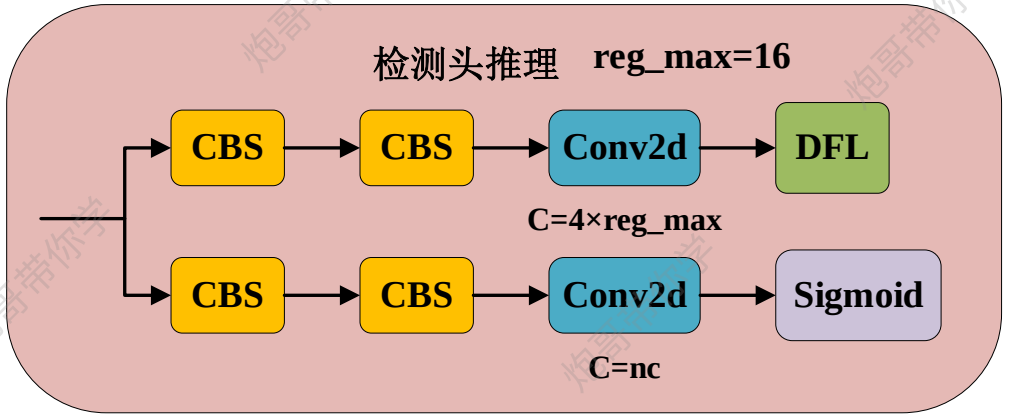
t：anchor point预测到上边框的距离

b：anchor point预测到下边框的距离

yolov8的Anchor-Free预测结果解析







步骤 1：获取坐标偏移量

步骤 2：通过 DFL 模块对偏移量进行加权，计算每个候选框的匹配度，并为每个候选框分配一个归一化的权重（通过 Softmax 操作）。

步骤3：利用权重和对应的位置进行加权求和得到对应的中心点的距离值。

步骤4：将加权计算后的坐标解码，还原到原图上。

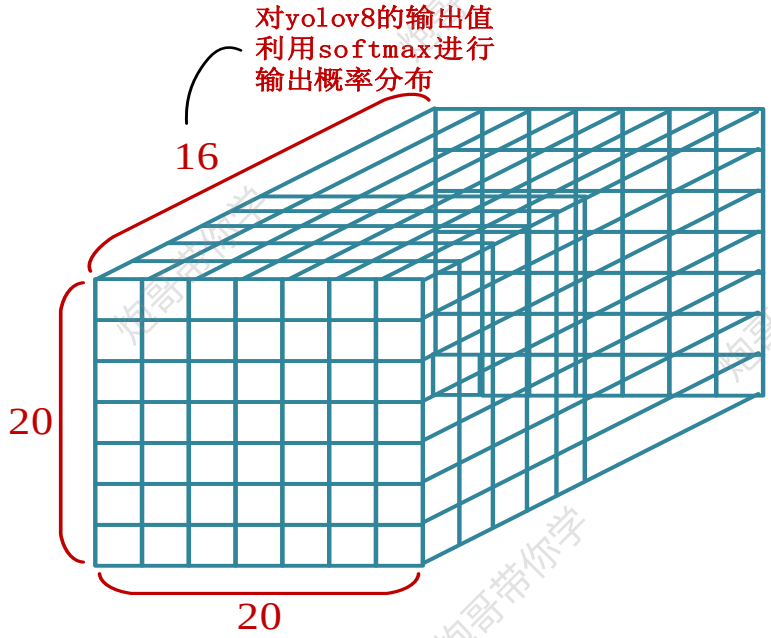
yolov8的Anchor-Free预测结果解析

步骤 1：获取坐标偏移量

步骤 2：通过 DFL 模块对偏移量进行加权，计算每个候选框的匹配度，并为每个候选框分配一个归一化的权重（通过 Softmax 操作）。

步骤3：利用权重和对应的位置进行加权求和得到对应的中心点的距离值。

步骤4：将加权计算后的坐标解码，还原到原图上。



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.05	0.05	0.6	0.3	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.03	0.2	0.7	0.07	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.05	0.0	0.0	0.0	0.0	0.05	0.1	0.8	0.08	0.02	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.06	0.04	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

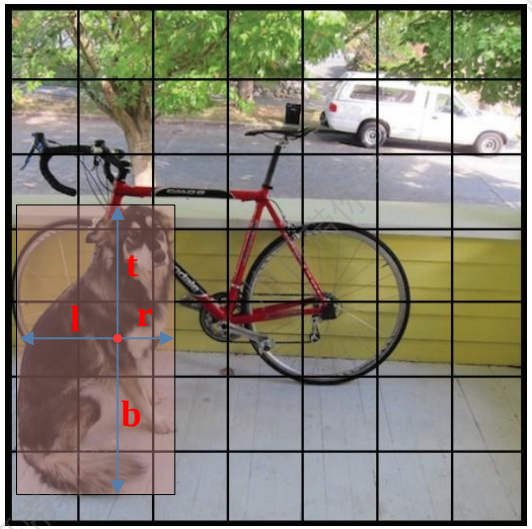
左=0.05×8+0.05×9+0.6×10+0.3×11=10.15

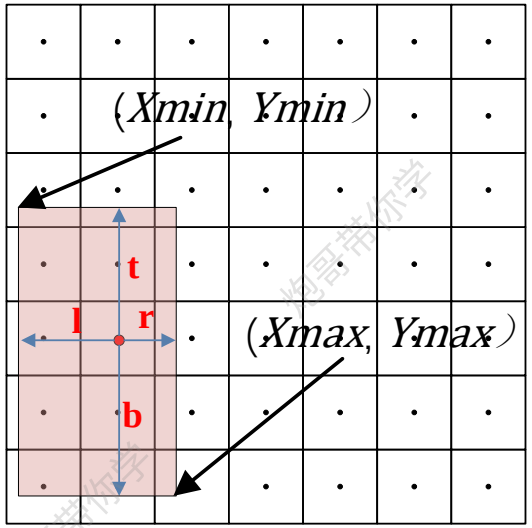
上=0.03×3+0.2×4+0.7×5+0.07×6=4.81

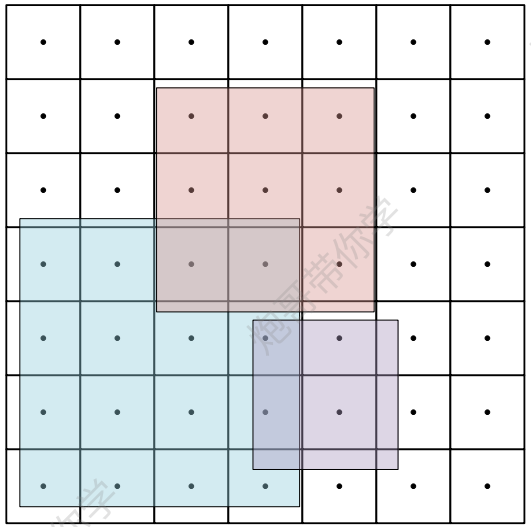
右=0.1×10+0.8×11+0.08×12+0.02×13=11.02

下=0.4×4+0.5×5+0.06×6+0.04×7=3.58

Yolov8正负样本匹配







网络输出值预处理

1、获取网络输出值：预测框、对应预测框的概率值。

2、将三个检测头的输出结果都映射到原图上，并将网络的输出结果还原到原图尺寸（20->640， 40->640 ，80->640 ）。同时将坐标转化为左上坐标，右下坐标的形式（早期的yolo版本代码中就已经是这样的形式了）。

3、初步正样本筛选：将三个检测头对应的anchor point和标注框做计算，所有的anchor point在标注框中的预测框，为初筛正样本。

Yolov8正负样本匹配

Anchor point质量计算式：

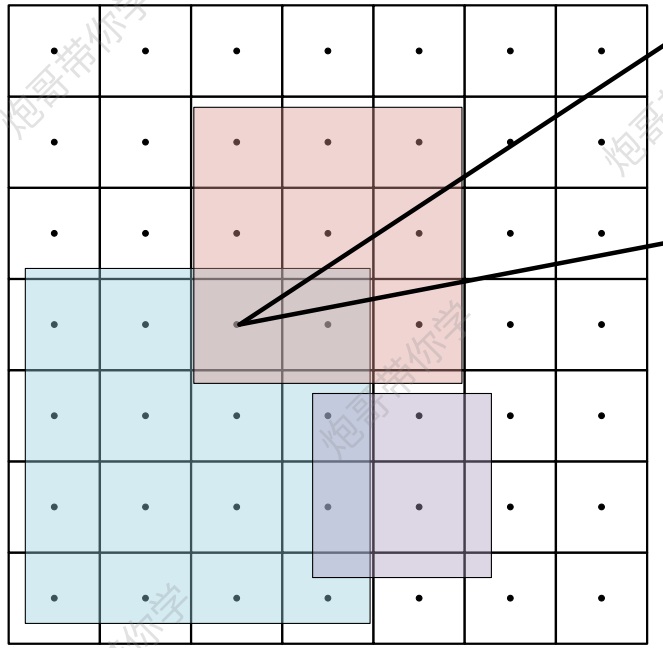
$$t = s^{\alpha} \times u^{\beta}$$

s 为粗筛正样本预测框与真实框的CIOU

u 为对应的预测框的物体概率

α 其中在v8中默认值为0.5

β 其中在v8中默认值为6



精筛正样本：

1、利用质量计算公式计算所有初筛正样本和真实框得分

2、选择得分最高的前k个作为候选框，默认选择前10个为正样本

3、去重（找出分配给多个真实框的预测框，只保留CIOU最大值的预测框为正样本）

yolov8的损失函数

预测框类别损失：BCE（二元交叉熵损失）

预测框定位损失：CIOU、DFL(分布焦点损失)

损失函数公式如下

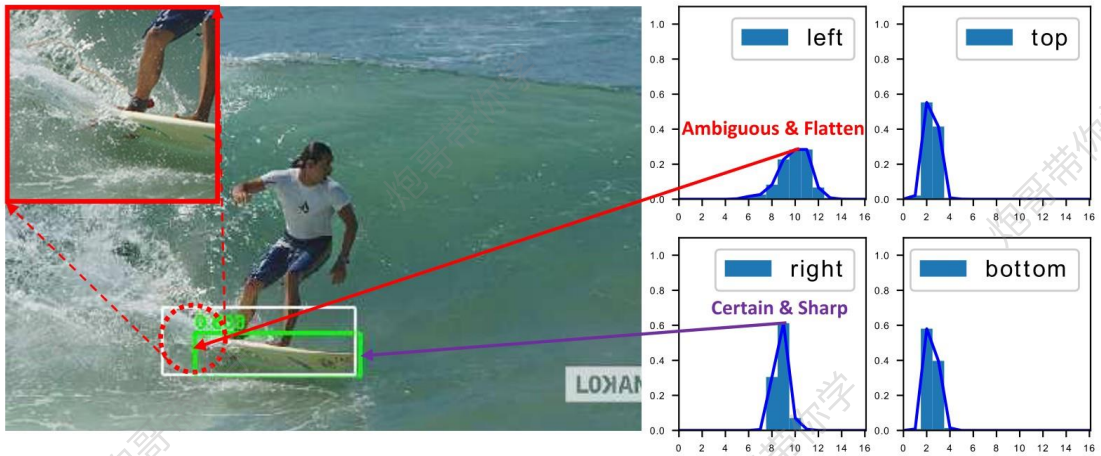
$$\text{Loss} = \lambda_1 \cdot L_{cls} + \lambda_2 \cdot L_{CIoU} + \lambda_3 \cdot L_{DFL}$$

```
warmup_bias_lr: 0.1 # (float) warmup initial bias lr
box: 7.5 # (float) box loss gain
cls: 0.5 # (float) cls loss gain (scale with pixels)
dfl: 1.5 # (float) dfl loss gain
pose: 12.0 # (float) pose loss gain
kobj: 1.0 # (float) keypoint obj loss gain
label_smoothing: 0.0 # (float) label smoothing (fract
nbs: 64 # (int) nominal batch size
```

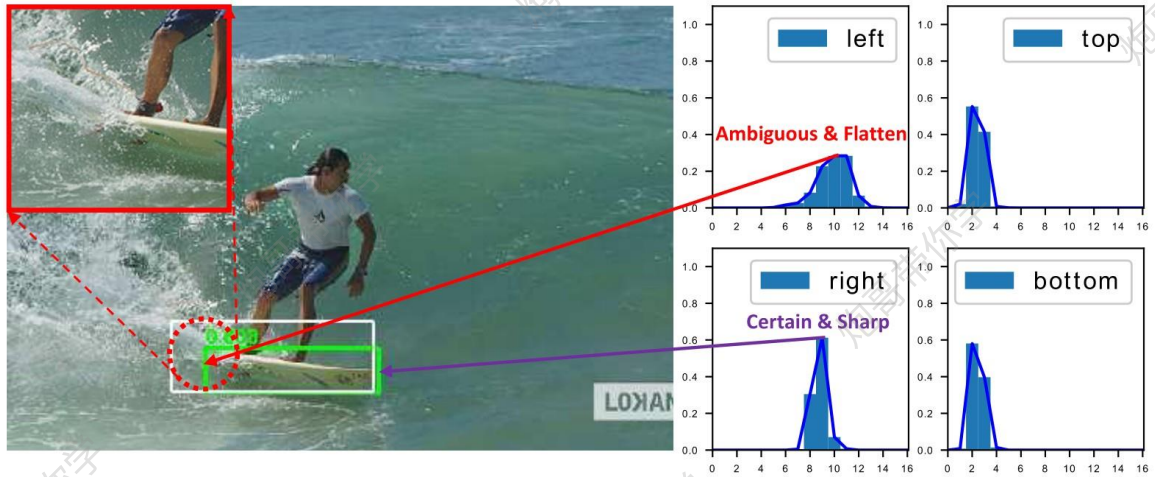
$$\text{Loss} = 0.5 \cdot L_{cls} + 7.5 \cdot L_{CIoU} + 1.5 \cdot L_{DFL}$$

坐标框的预测结果是通过概率和对应下标相乘再累加获得的。

$$\hat{y} = \sum_{i=0}^{n} P(y_i) y_i$$



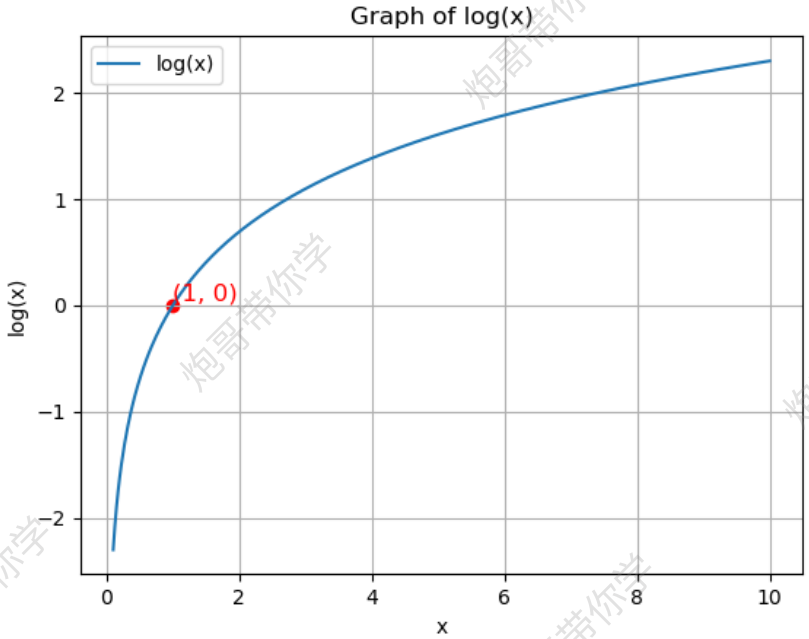
yolov8的损失函数



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

其中，$\mathcal{S}_i$ 为输出的概率值，用softmax函数映射出。

$$softmax(x)_i=\frac{e^{x_i}}{\sum_j e^{x_j}}$$



DFL损失函数计算步骤：

1、获取标签坐标大小（缩放后的），例如坐标为3.2。

2、获取 y_i，坐标的下采样取整3，y_{i+1}上采样取整4。

3、获取对应的3的sofmax的预测概率 $\mathcal{S}_i$，同理获取4的概率 $\mathcal{S}_{i+1}$

4、计算3的概率损失的权重$(y_{i+1}-y)$，同理获取4的概率损失权重$(y-y_i)$

5、利用log函数的特性来更新loss函数。

yolov8的实验总结