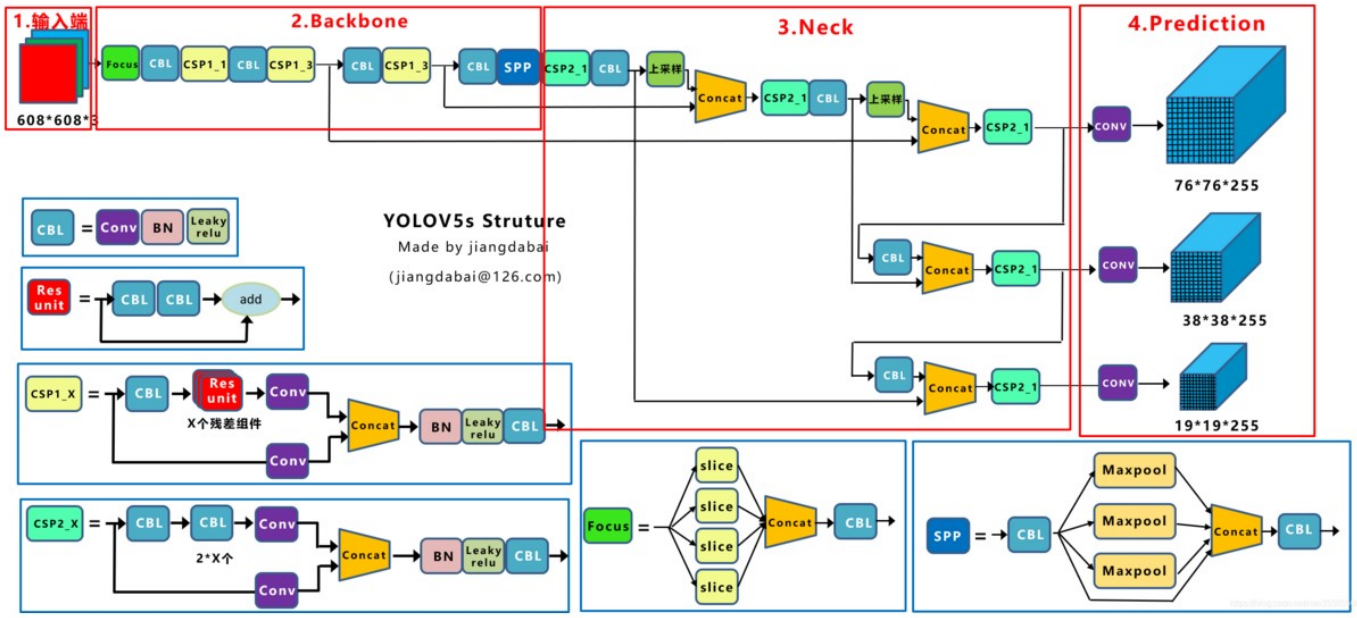
一、整体网络

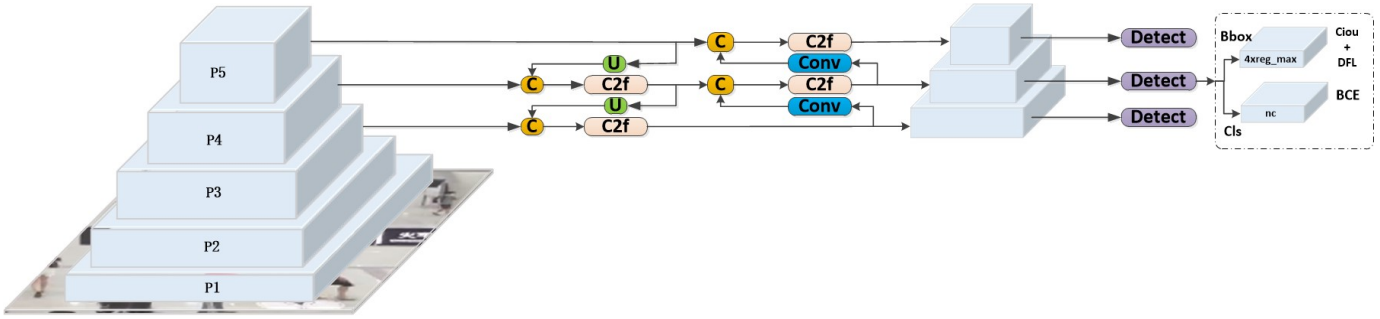
本文结合YOLOv5网络进行讲解，通过与YOLOv5网络进行比较，进一步理解YOLOv8，尽快上手。

(1) YOLOv5

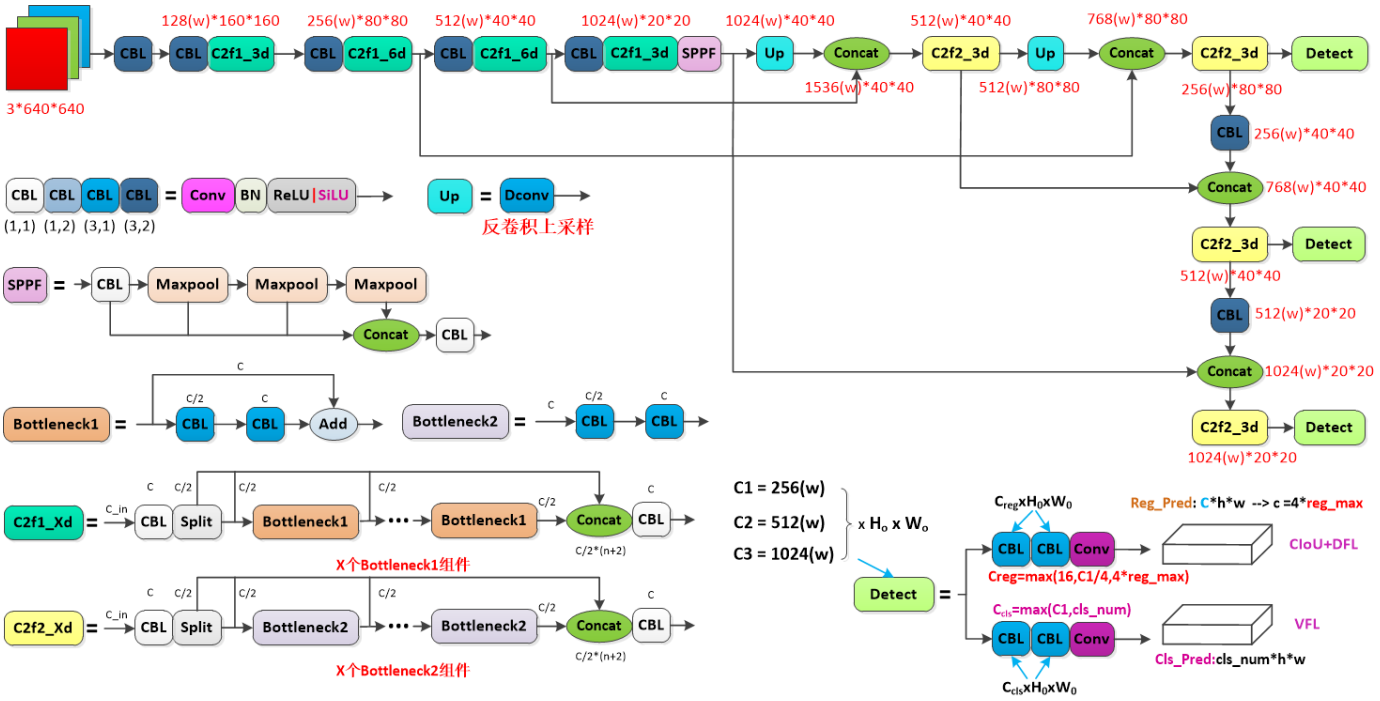


(2)YOLOv8

<1> 大体结构



<2> 详细结构

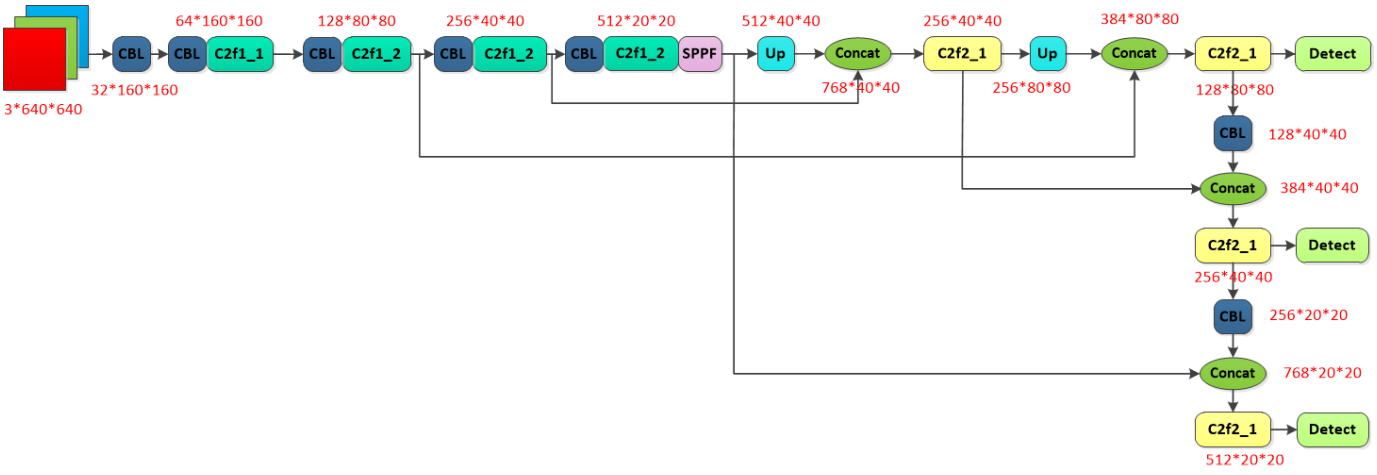


<3> 实例展示

上图中的d和w分别代表不同类型YOLOv8网络(yolov8n、yolov8s、yolov8m、yolov8l)的深度和网络的宽度的比率，如下所示：

model	d (depth_multiple)	w (width_multiple)
n	0.33	0.25
s	0.33	0.50
m	0.67	0.75
l	1.00	1.00
x	1.00	1.25

下面以yolov8s为例，其中d = 0.33, w=0.5,计算过程中向上取整。



二、比较

（1）YOLOv8与YOLOv5比较

相同点：

从整体上来看，YOLOv8和YOLOv5基本一致，都是backbone + PANet + Head的结构，且PANet部分都是先上采样融合再下采样融合；

不同点：

<1> Head部分不同，YOLOv5是整体上输出的，以80类为例，因为每个像素点为3个anchor，故每个像素点的size为：3*（4 + 1 + 80 ）= 255；而YOLOv8Head部分，Cls和Box是分开预测的,并且从Anchor-Based换成了Anchor-Free。

yolov8中提到的anchors与yolov5有本质的区别，在yolov8中只是一个锚点,即预测的中心点或者可以理解为每个像素的中心点，并且**yolov8中，每个像素只有一个锚点，例如，yolov8 输出共有 80*80 + 40*40 20*20 = 8400个锚点。**

<2> YOLOv8的**Backbone**和**Neck**中采用的**C2f结构**，其参考了**YOLOv7**的**ELAN**的设计思想，用于替换**YOLOv5**中的**CSP结构**，由于**C2f结构**有着更多的残差连接，所以其有着**更丰富的梯度流。**(不过这个C2f模块中存在Split操作，对特定硬件部署并不友好)

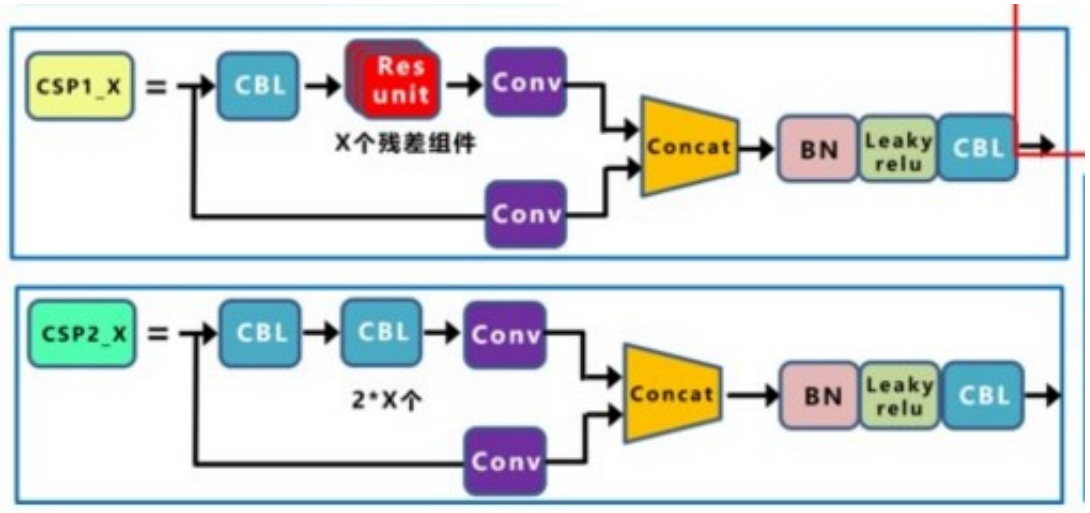
<3> **Loss** 计算方面采用了**TaskAlignedAssigner正样本匹配策略**，并引入了**Distribution Focal Loss**.

<4> **训练部分**，采用了**YOLOX**的训练方式，在最后的10个Epoch关闭了Mosiac增强操作，可以有效地提升精度。

(2) 庖丁解牛

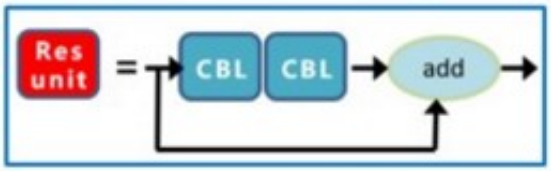
<1> CSP 到 C2f

a. **YOLOv5**中的CSP结构如下所示：



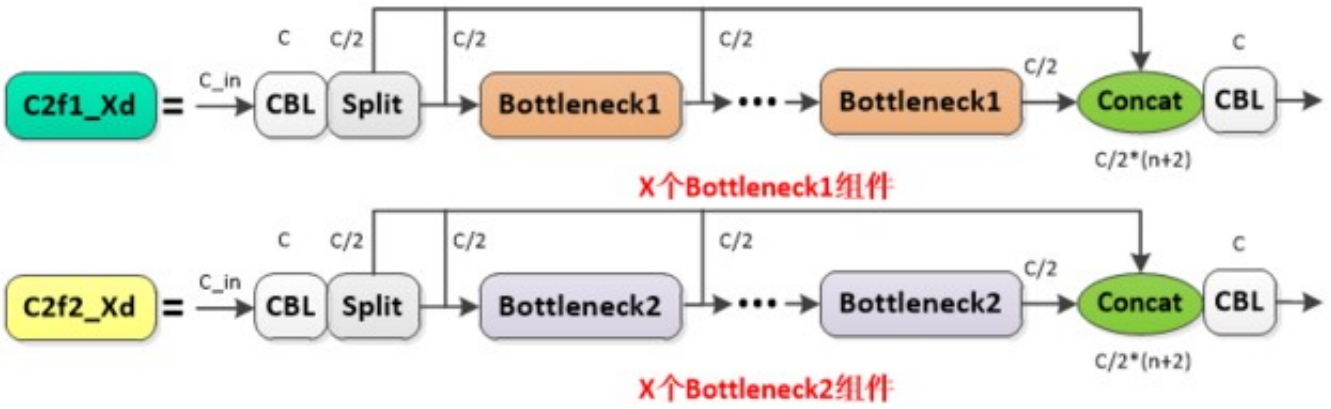
CSP在Backbone与Neck中是不同的，

在Backbone中为**CSP1_X**，其X个组件为残差结构，如下图所示，其中**两个CBL输入与输出的Shape是相同**的。



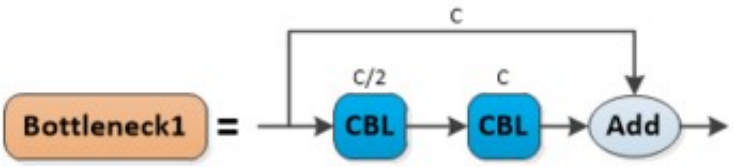
在Neck中为**CSP2_X**，其X组件为2*X个CBL模块，**每对CBL的输入输出Shape都是相同的。**

b. **YOLOv8**中的C2f结构如下所示，可以发现与CSP相比，其**增加了更多的跳层连接和额外的Split操作。**

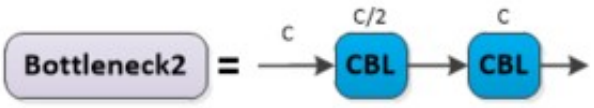


C2f在Backbone和Neck中也是不同的，

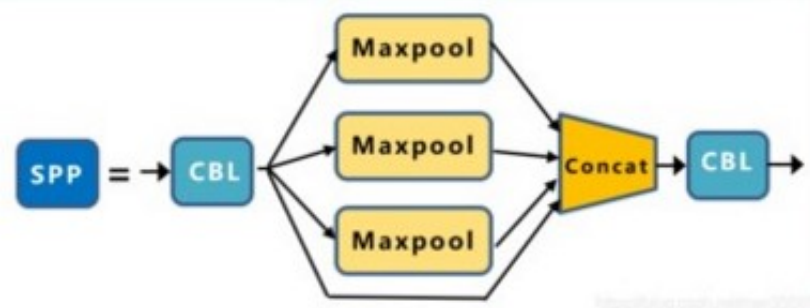
在Backbone中为**C2f1**，其X个组件为是带有残差边的残差模块，如下图所示，其中第一个CBL通道数减少一半，第二个又恢复到C。

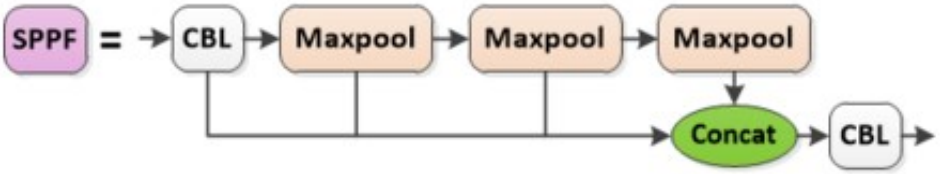


在Neck中为**C2f2**,其X个组件没有残差边，如下图所示，



<2> 从SPP变到SPPF



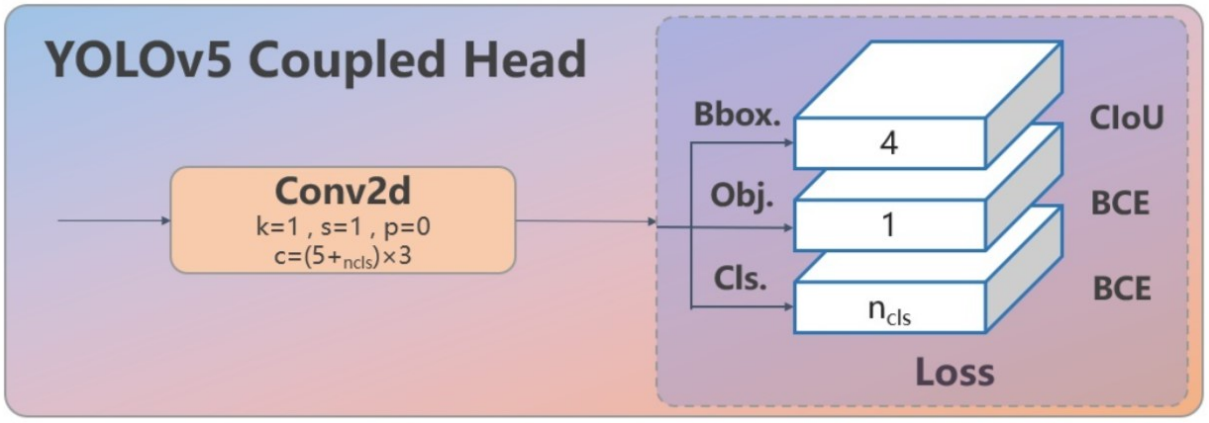


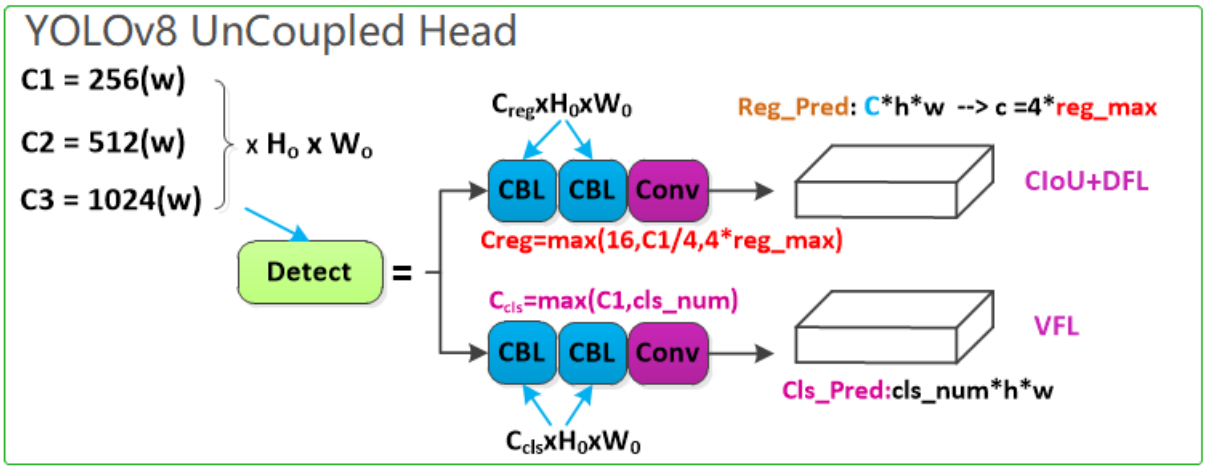
SPPF较SPP，可以在一定程度上降低计算量，并且增大感受野。

用两个2个5x5代替9x9，用3个5x5代替13x13.

<3> Head从耦合头变为解耦头

从YOLOv5的耦合Head，变为的解耦Head，并且由Anchor-Based 变成了Anchor-Free，如下所示：





可以看出，YOLOv8的Head中，不再有之前的Obj 分支，只有解耦的分类和回归分支，并且回归分支使用了**Distribution Focal Loss**中提到的**积分形式表示法**。

同时，从上图YOLOv8 Head可以看出，解耦头中的Cls类别分支和box 回归分支的通道数是不相等的，因为各自表征了两种不同的特征，因此:

a. 对于Cls类别分支,其通道数**Ccls** = max(C1,cls_num)

b. box回归分支，其通道数**Creg** = max(16,C1/4,4*reg_max)

以COCO数据集为例，则解耦头的通道数配置为：

Ccls = max(256,80) = 256

Creg = max(16,256/4,4*16) = 64 ; 其中reg_max = 16

同时，YOLOv8 抛弃了Anchor Box, 因为聚类anchor box是依赖于数据集的，数据集如果不够充分，就无法较为准确地反映数据本身地特征分布，那聚类出来地anchor box就只能是次优的，甚至更差。

三、正负样本筛选

既然YOLOv8没有了anchor box，那么正负样本的匹配就需要依赖于多尺度分配。

(1) 匹配策略

YOLOv8的正负样本匹配策略采用的是**Task-Aligned Assigner**，（顾名思义就是**对齐分配器**），那是怎样的对齐方式呢?

根据分类与回归的分数，作为加权分数，选择正样本。公式如下：

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

其中，S是GT的预测分值，U是预测框和GT Box的ciou，α和β为权重超参数，两者**相乘**就可以**衡量对齐程度**，当Cls的分值越高且IOU越高时，t的值就越接近于1.

通过训练t可以引导网络动态的关注于高质量的正样本。

a. 对于每个GT，对**所有预测框**基于该GT的**类别cls score** 结合 与该GT box的 IOU,加权得到一个关联Cls及Box Reg的对齐分数**alignment_metrics**。

b. 对于每个GT，直接基于**alignment_metrics**对齐分数，通过排序后，选取topK个预测框作为正样本。

(2) 损失函数

Loss包括2个分支：Cls 与 Box Reg，**并没有之前的Obj 前景/背景 的分支**。

<1> 分类损失 Lcls

使用sigmoid函数来计算每个类别的概率，并计算全局的类别损失。

采用 **VFL Loss 或 BCE Loss**

a. **VFL**

其训练(学习)标签： 正样本的类别标签就是IOU值(0 ~1)因为采用的sigmoid；负样本的学习标签全为0.

与**Focal Loss**相比，提出了**非对称的加权**操作，Focal Loss中的加权是对称的(正负样本都进行**同等规则的加权**)。公式如下：

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

其中，q为label，正样本的时候q为预测bbox和GT的IoU，负样本时q=0。

如上面公式，当为正样本的时候没有采用Focal Loss,而是普通的BCE Loss，只不过多了一个自适应的IOU加权，用于突出主样本(在正样本范围内，主样本的IOU值最大)。

负样本采用的是Focal Loss。

可见VFL是对正负样本进行**非对称加权**，突出主样本。

b. BCE

从 工程代码中，发现YOLOv8实际采用的是BCE损失，如下图所示：

```
# cls loss
# loss[1] = self.varifocal_loss(pred_scores, target_scores, target_labels) / target_scores_sum  # VFL way
loss[1] = self.bce(pred_scores, target_scores.to(dtype)).sum() / target_scores_sum  # BCE
```

说明，YOLOV8团队应该是对VFL和BCE都尝试过，但最终发现使用VFL和使用普通的BCE效果相当，优势不明显，故**采用了简单的BCE Loss。**

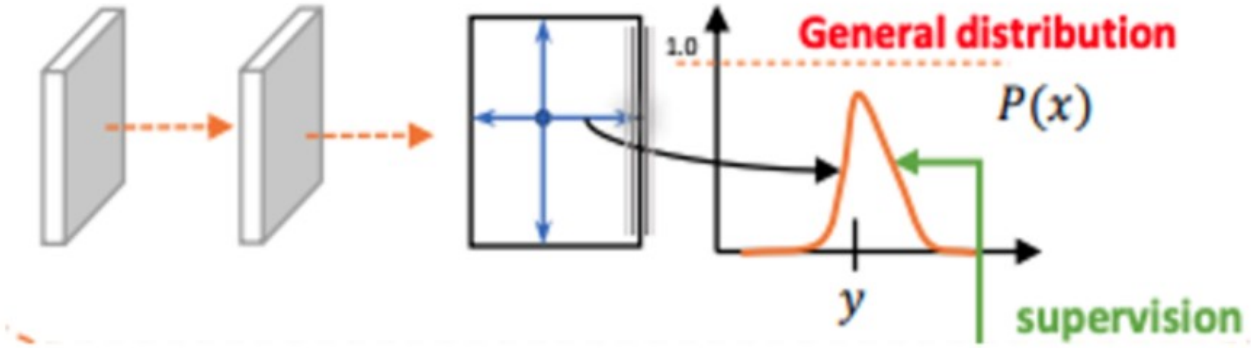
<2> 回归损失 Lbox

对于位置损失，YOLOv8分为两个部分**Clou_Loss + Distribution Focal Loss**（DFL Reg_max默认为16）默认为16。

第一部分就是计算预测框与目标框之间的IOU，一如既往地采用了CIOU Loss，而第二部分采用地就是DFL.

DFL 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。

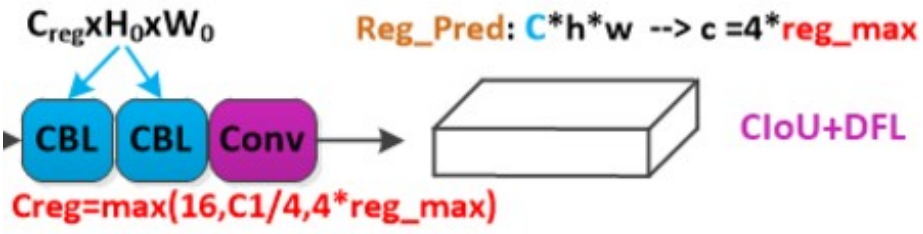


$$\mathbf{DFL}(\mathcal{S}_i, \mathcal{S}_{i+1}) = -\left((y_{i+1} - y)\log(\mathcal{S}_i) + (y - y_i)\log(\mathcal{S}_{i+1})\right)$$

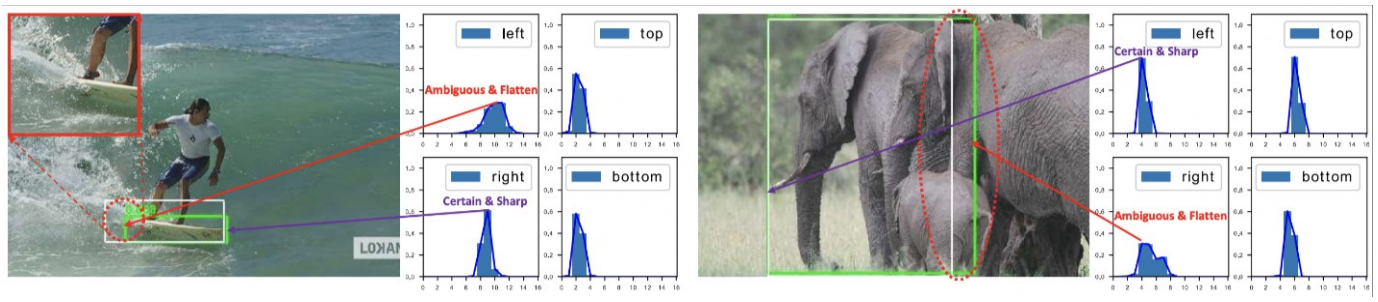
DFL 能够让网络更快地聚焦于目标y附近的值，增大它们的概率。

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，从而让网络更快地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**。

问题：上面提到需要将bbox的预测，建模成一个任意分布，通过对该任意分布进行回归，来得到边界框。那么这个任意分布是如何建立的?



由上图可见，bboxes预测输出通道为C，如reg_max=16, 则通过这16个值建模一个类似于高斯的分布(只能说是类似，其参数值在训练过程中是不断更新的)

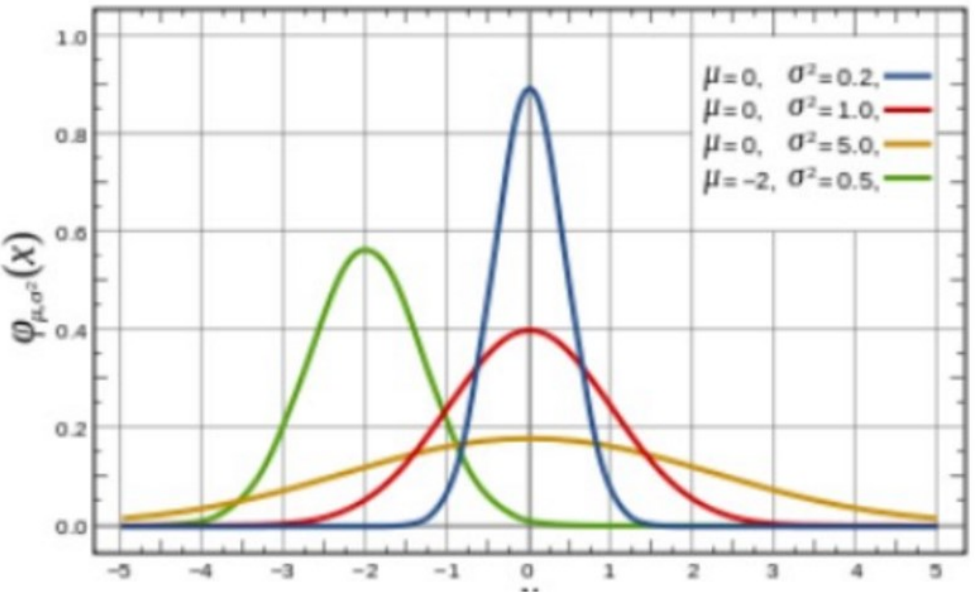


如上图左侧所示，对于滑板左侧被水花模糊，故左侧边界的预测分布是任意而扁平的，对右侧边界的预测分布是明确而尖锐的。右侧的大象预测同理。

所以对于边界的预测，**围绕GT位置结合周围位置**，采类似高斯用任意分布进行处理，我们可以回忆一下高斯函数：

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

a表示得到曲线的高度，b表示曲线在x轴的中心，c表示width(与半峰全宽有关)，图像如下所示：



通过**16个值**对分布进行离散化处理，可见GT位置的值是最大的，两边的值是逐渐变小的，结合高斯函数**我们如何对离散的分布进行处理呢?**

可以用**softmax**，实现离散回归，从结构上来讲，softmax的公式计算与高斯函数是相似的，而且在代码中方便应用。

最后通过对**分布进行积分处理**，就得到了bbox坐标四个预测值的结果。

tip：讲到这，突然想到，yolov8对坐标预测的采用任意分布的方式，与yolor 预测采用隐性知识向量，有着异曲同工之妙。