前言

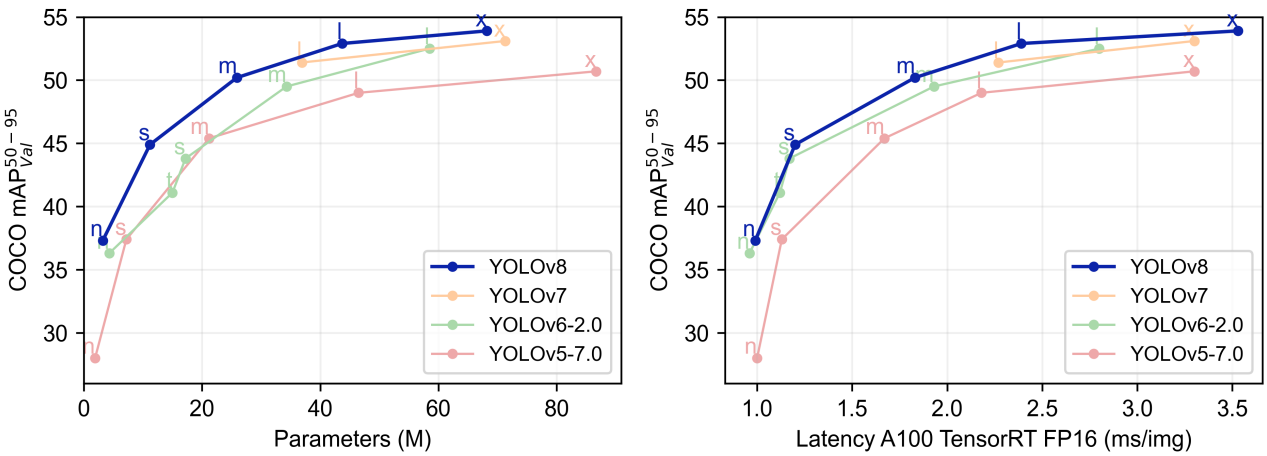
Yolov8 为 ultralytics 公司在23年1月10号对开源的 v5 版本进行一个重大更新版本。支持图像分类、物体检测和实例分割任务。

其创新点包含：**一个新的骨干网络、一个新的 Anchor-Free 检测头和一个新的 loss 损失函数。**

ultralytics 公司团队打算将 ultralytics 打造成一个算法框架库，具备可扩展性。不仅仅只服务于 YOLO 检测系列模型，而是能够支持非 YOLO 模型以及 分类、分割、姿态估计 等各类任务。

ultralytics 开源库的两大优点：

- 融合众多当前 SOTA 技术于一体；
- 未来将支持除 YOLO 系列之外的更多算法。



下标为官方在 COCO Val 2017 数据集上测试的 mAP、参数量 和 FLOPs 结果。精度上 v8 都比 v5 要提升很多，而 n、s、m 型号的model 参数量，v8 要比 v5 高些。结合上图也可以看出，推理速度 v8 相较于 v5 都要慢上一些。

模型	YOLOv5	params(M)	FLOPs@640 (B)	YOLOv8	params(M)	FLOPs@640 (B)
n	28.0(300e)	1.9	4.5	37.3 (500e)	3.2	8.7
s	37.4 (300e)	7.2	16.5	44.9 (500e)	11.2	28.6
m	45.4 (300e)	21.2	49.0	50.2 (500e)	25.9	78.9
l	49.0 (300e)	46.5	109.1	52.9 (500e)	43.7	165.2
x	50.7 (300e)	86.7	205.7	53.9 (500e)	68.2	257.8

> 现在已经出到 v8 版本的 YOLO，但是这都只是在 COCO 数据集上都有显著提升，但不意味着对其他数据集也有较好的效果，泛化性不确定，所以不必一味的去追随新模型，可以尝试使用。
>
> 毕竟仍有传闻说 v5 的泛化性更好。

YOLOv8 概述

其提供一个全新的 SOTA 模型，包括 P5 640 和 P6 1280 分辨率的**目标检测网络** 和 基于 YOLACT （19年提出的一种分割模型）的实例分割模型。也实现了基于缩放基数提供 N S M L X 不同 size 的 model。

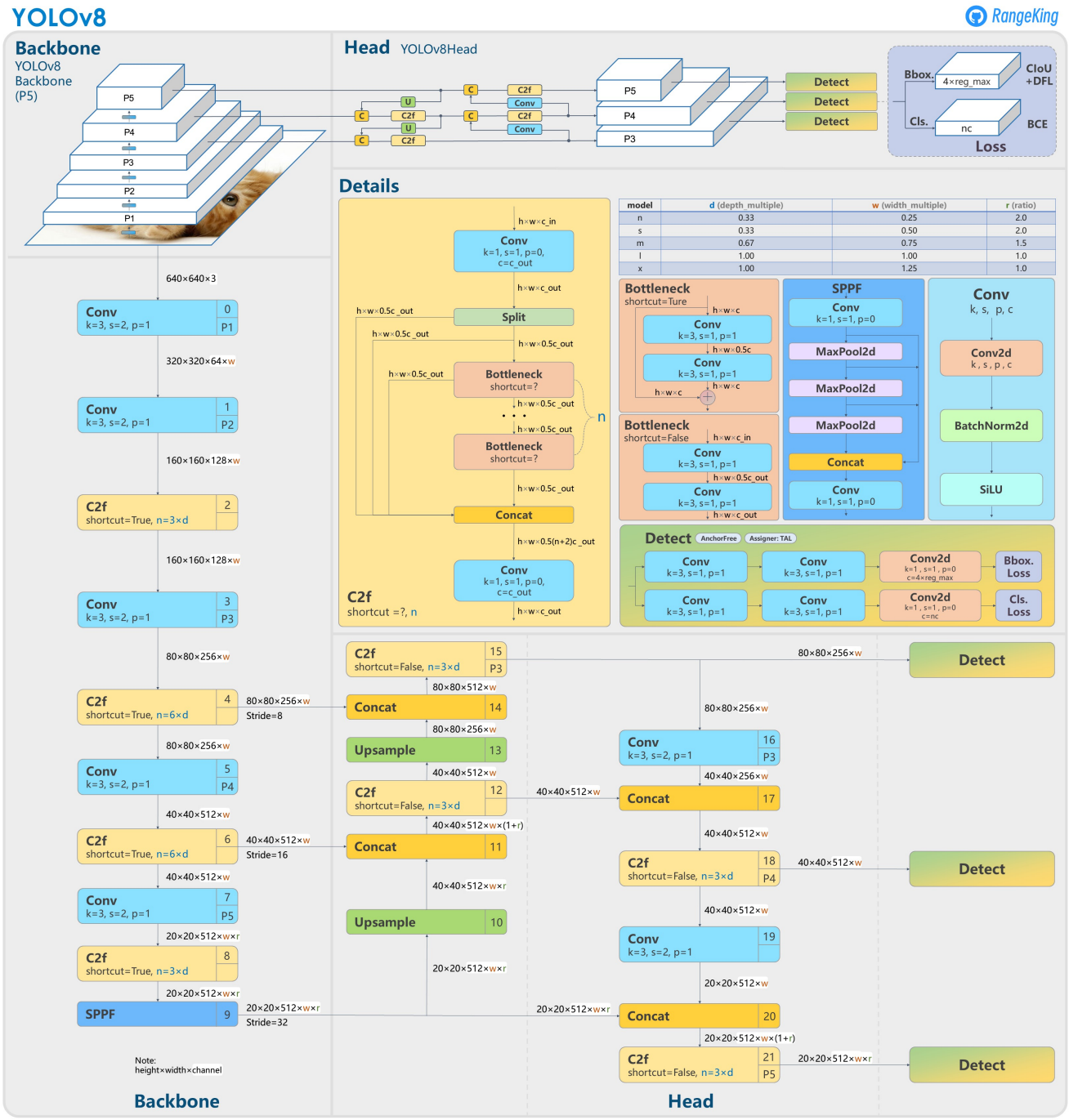
1. **骨干网络 和 Neck** 部分可能参考了 YOLOv7 ELAN 设计思想，将 YOLOv5 的 C3 结构换成了梯度流更丰富的 C2f 结构，并对不同尺度模型调整了不同的通道数，属于对模型结构精心微调，不再是无脑一套参数应用所有模型，大幅提升了模型性能。不过这个 C2f 模块中存在 Split 等操作对特定硬件部署没有之前那么友好了（应该是会影响到后面的推理速度）。
    - Backbone：使用的依旧是 **CSP （为了检测不同大小的目标，分别有大中小的特征提取）**思想，以及依旧使用 v5 中的 **SPPF 模块（SPP/SPPF作用：实现局部特征和全局特征的 feature Map 级别的融合）**，将 v5 中的 **C3 模块换成了 C2f 模块** ==> 实现进一步轻量化。

- PAN-FPN：v8 依旧使用了 PAN 思想，不同的是 v8 将 v5 中 PAN-FPN 上采样阶段中的卷积结构删除了，同时 C3 换成了 C2f。

  > FPN（Feature pyramid network，特征金字塔）：自顶向下进行上采样，使得底层特征图包含更强的图像强语义信息；
  >
  > PAN（Path Aggregation network，路径聚合网络）：自底向上进行下采样，使得顶层特征图包含图像位置信息；
  >
  > 上采样：feature map --> img size，类似反向卷机过程；
  >
  > 下采样：又叫降采样，可理解为传统卷积过程；

以上的内容可以看出其实 v8 没有过多的创新，主要还是参考了 YOLOX、v6、v7 和 PPYOLOE 等算法的相关设计，v8 偏向工程实践。

模型结构



从 yaml 配置文件可以看出，v8 与 v5 在 Backbone 中的改动较小。

```yaml
nc: 80  # number of classes
scales: # model compound scaling constants, i.e. 'model=yolov8n.yaml' will call yolov8.yaml with scale 'n'
  # [depth, width, max_channels]
  n: [0.33, 0.25, 1024]  # YOLOv8n summary: 225 layers,  3157200 parameters,  3157184 gradients,   8.9 GFLOPs
  s: [0.33, 0.50, 1024]  # YOLOv8s summary: 225 layers, 11166560 parameters, 11166544 gradients,  28.8 GFLOPs
  m: [0.67, 0.75, 768]   # YOLOv8m summary: 295 layers, 25902640 parameters, 25902624 gradients,  79.3 GFLOPs
  l: [1.00, 1.00, 512]   # YOLOv8l summary: 365 layers, 43691520 parameters, 43691504 gradients, 165.7 GFLOPs
  x: [1.00, 1.25, 512]   # YOLOv8x summary: 365 layers, 68229648 parameters, 68229632 gradients, 258.5 GFLOPs

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]]  # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]]  # 1-P2/4
  - [-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
```

yolov8.yaml - Backbone

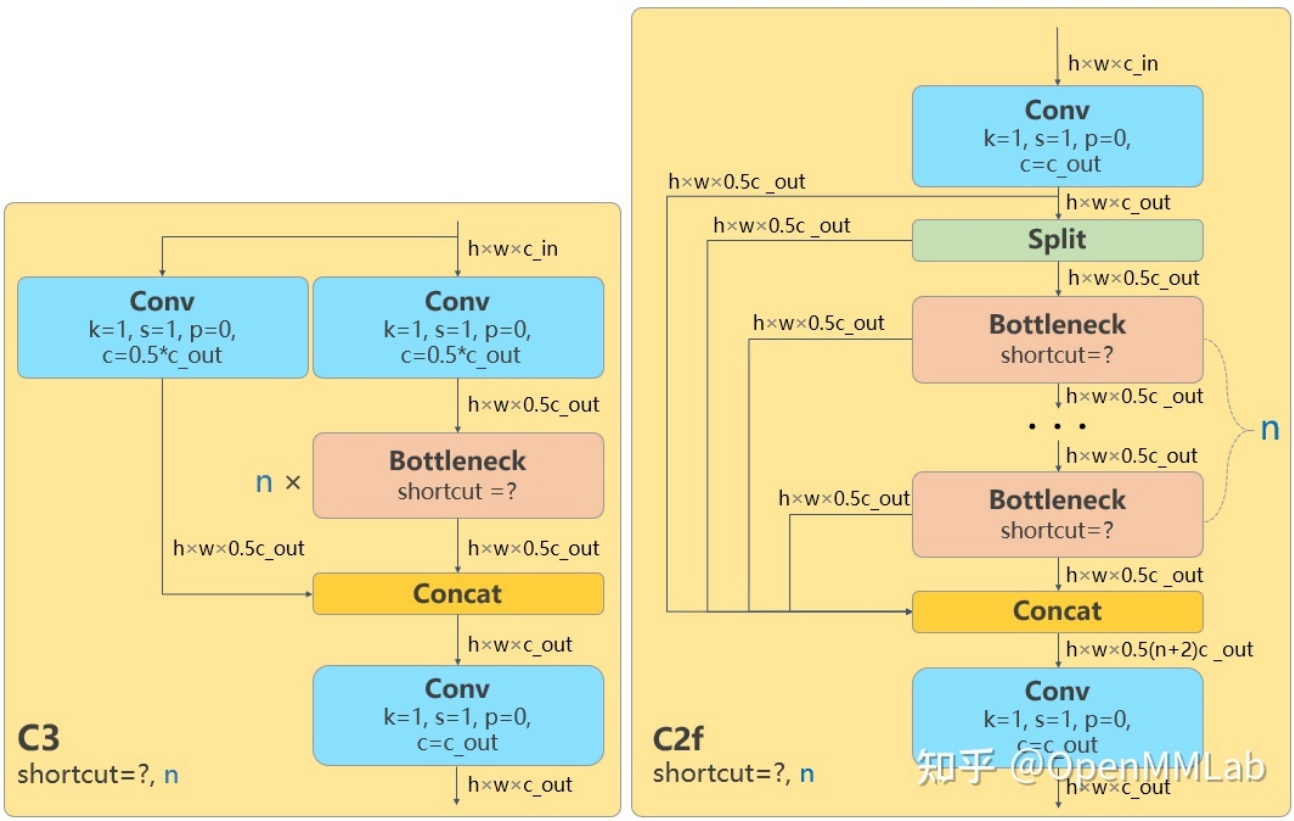
```yaml
nc: 80  # number of classes
scales: # model compound scaling constants, i.
  # [depth, width, max_channels]
  n: [0.33, 0.25, 1024]
  s: [0.33, 0.50, 1024]
  m: [0.67, 0.75, 1024]
  l: [1.00, 1.00, 1024]
  x: [1.33, 1.25, 1024]

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 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, 6, 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, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]
```

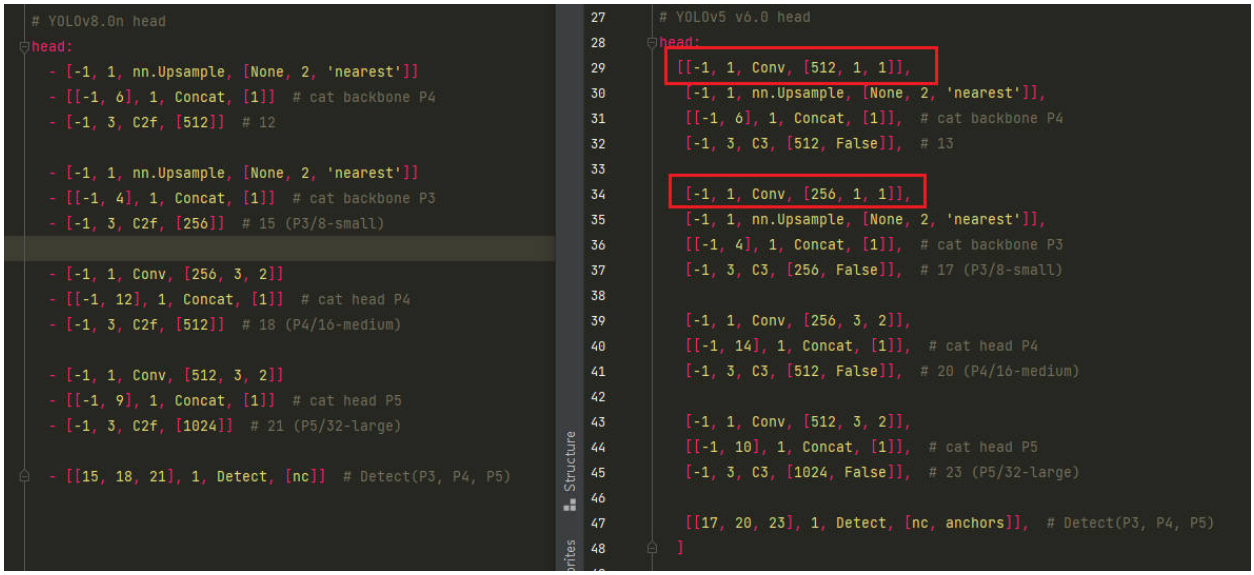
yolov5.yaml - Backbone

1）骨干网络 和 Neck 部分的具体变化：

- 第一个卷积层的 kernel 从 6x6 变成了 3x3；
- 所有的 C3 模块换成了 C2f 模块（其中增加了更多的跳层连接和额外的 Split 操作）；



- 去掉了 head 模块中的 2 个卷积连接层；



- Backbone 中 C2f 的block 模块数量从 3-6-9-3 改成了 3-6-6-3；
- 查看 N/S/M/L/X 等不同大小模型，可以发现 N/S 和 L/X 两组模型只是改了缩放系数，但是 S/M/L 等骨干网络的通道数设置不一样，**没有遵循同一套缩放系数**。如此设计的原因应该是同**一套缩放系数下的通道设置不是最优设计**，YOLOv7 网络设计时也没有遵循一套缩放系数作用于所有模型；

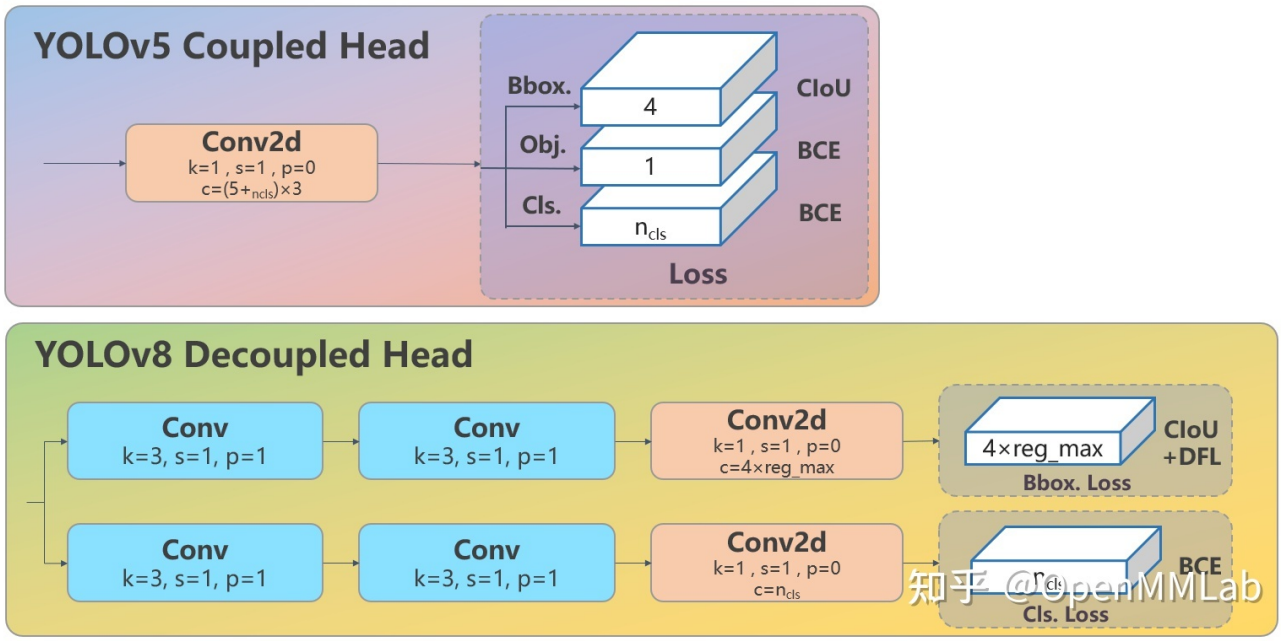
将 YOLOv5 的 C3 结构换成了梯度流更丰富的 C2f 结构，并对不同尺度模型调整了不同的通道数，属于对模型结构精心微调，不再是无脑一套参数应用所有模型，大幅提升了模型性能。不过这个 C2f 模块中存在 Split 等操作对特定硬件部署没有之前那么友好了。

- Backbone：使用的依旧是 **CSP （为了检测不同大小的目标，分别有大中小的特征提取）**思想，以及依旧使用 v5 中的 **SPPF 模块（SPP/SPPF作用：实现局部特征和全局特征的 feature Map 级别的融合）**，将 v5 中的 **C3 模块换成了 C2f 模块** ==> 保证轻量化的同时丰富梯度流信息。
- PAN-FPN：v8 依旧使用了 PAN 思想，不同的是 v8 将 v5 中 PAN-FPN 上采样阶段中的卷积结构删除了，同时 C3 换成了 C2f。

> FPN（Feature pyramid network，特征金字塔）：自顶向下进行上采样，使得底层特征图包含更强的图像强语义信息；
>
> PAN（Path Aggregation network，路径聚合网络）：自底向上进行下采样，使得顶层特征图包含图像位置信息；
>
> 上采样：feature map --> img size，类似反向卷机过程；
>
> 下采样：又叫降采样，可理解为传统卷积过程；

2）Head 部分：

由原先的耦合头变成了解耦头，由 v5 的 Anchor-Based 变成了 Anchor-Free。



主要特点：就是不再像 v5 那样去预先设置一些尺寸的anchor，而是通过 TaskAlignedAssigner 类的 assigner 去对齐 gt 和 pre_box。

核心思路：在特征图上每个网格作为一个 anchor，选取所有符合条件的 grid 作为正样本。（选取条件：1、预测分数；2、iou值；3、top-k 思想；）

> 若 anchor 对应多个 gt，则选取 iou 最高的。

3）Loss 计算：

两大块：正样本分配策略 和 Loss function；

现代目标检测器大部分都会在正负样本分配策略上面做文章，典型的如 YOLOX 的 simOTA、TOOD 的 TaskAlignedAssigner 和 RTMDet 的 DynamicSoftLabelAssigner，这类 Assigner 大都是动态分配策略，而 YOLOv5 采用的依然是静态分配策略。考虑到动态分配策略的优异性，YOLOv8 算法中则直接引用了 TOOD 的 TaskAlignedAssigner。

TaskAlignedAssigner：根据分类和回归的分数加权的分数选择正样本。

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

s 是标注类别对应的预测分数，u 是预测框和GT的iou，两者相乘就是衡量对齐程度。

1. 对于每一个GT，对所有的预测框都会计算得到一个对齐分数 t，即 alignment_metrics；
2. 对于每一个GT，直接基于 alignment_metrics 对齐分数，选取 Top-K 作为正样本。

> focal loss:
>
> $$FL(p,y)=\begin{cases}-\alpha(1-p)^{\gamma}\log(p) & if \quad y=1\\ -(1-\alpha)p^{\gamma}\log(1-p) & otherwise\end{cases}$$
>
> 想法就是：让困难样本有更高的权重；
>
> VFL（varifocal loss）：
>
> $$VFL(p,q)=\begin{cases}-q(q\log(p)+(1-q)\log(1-p)) & if \quad q>0\\ -\alpha p^{\gamma}\log(1-p) & if \quad q=0\end{cases}$$
>
> VFL 从 focal loss 中借鉴样本加权思想，解决类不平衡问题；其中 p 是分类得分，q 是目标 IOU 得分。在训练过程中，负样本的 q 为0；
>
> - 正样本时：就是普通的 BCE；
> - 负样本时：就是标准的 FL；
>
> DFL：
>
> $$DFL(S_i,S_{i+1})=-((y_{i+1}-y)\log(S_i)+(y-y_i)\log(S_{i+1}))$$
>
> DFL 可以让网络更快的聚焦于目标 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，从而让网络更快的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。（还得在打磨打磨）

三个 Loss function 加权相加。

4）训练的数据增强部分引入了 YOLOX 中的最后 10 epoch 关闭 Mosiac 增强的操作，可以有效提升精度。

使用yaml文件加载模型，通过文件名"yolov8{}.yaml"的形式来控制加载模型的型号。

```
# yolov8.yaml

# Ultralytics YOLO 🚀, AGPL-3.0 license
```

```yaml
# YOLOv8 object detection model with P3-P5 outputs. For Usage examples see https://docs.ultralytics.com/tasks/detect

# Parameters
nc: 80  # number of classes
scales: # model compound scaling constants, i.e. 'model=yolov8n.yaml' will call yolov8.yaml with scale 'n'
  # [depth, width, max_channels]
  n: [0.33, 0.25, 1024]  # YOLOv8n summary: 225 layers,  3157200 parameters,  3157184 gradients,   8.9 GFLOPs
  s: [0.33, 0.50, 1024]  # YOLOv8s summary: 225 layers, 11166560 parameters, 11166544 gradients,  28.8 GFLOPs
  m: [0.67, 0.75, 768]   # YOLOv8m summary: 295 layers, 25902640 parameters, 25902624 gradients,  79.3 GFLOPs
  l: [1.00, 1.00, 512]   # YOLOv8l summary: 365 layers, 43691520 parameters, 43691504 gradients, 165.7 GFLOPs
  x: [1.00, 1.25, 512]   # YOLOv8x summary: 365 layers, 68229648 parameters, 68229632 gradients, 258.5 GFLOPs

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]]  # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]]  # 1-P2/4
  - [-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

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, 'nearest']]
  - [[-1, 6], 1, Concat, [1]]  # cat backbone P4
  - [-1, 3, C2f, [512]]  # 12

  - [-1, 1, nn.Upsample, [None, 2, 'nearest']]
  - [[-1, 4], 1, Concat, [1]]  # cat backbone P3
  - [-1, 3, C2f, [256]]  # 15 (P3/8-small)
```

```
  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 12], 1, Concat, [1]]  # cat head P4
  - [-1, 3, C2f, [512]]  # 18 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 9], 1, Concat, [1]]  # cat head P5
  - [-1, 3, C2f, [1024]]  # 21 (P5/32-large)

  - [[15, 18, 21], 1, Detect, [nc]]  # Detect(P3, P4, P5)
```

利用default.yaml文件配置 **模型执行任务、配置训练参数、验证测试参数、检测参数、超参数的设置** 等，而这些的配置信息会统一加载到每次的训练文件夹中进行记录，对工程化训练的跟踪有了很大的帮助。

```
# default.yaml ---> setup yolo args
# Ultralytics YOLO 🚀, AGPL-3.0 license
# Default training settings and hyperparameters for medium-augmentation COCO training

task: detect  # YOLO task, i.e. detect, segment, classify, pose
mode: train  # YOLO mode, i.e. train, val, predict, export, track, benchmark

# Train settings -------------------------------------------------------------------------------------------------
model: # path to model file, i.e. yolov8n.pt, yolov8n.yaml
data:  # path to data file, i.e. coco128.yaml
epochs: 200  # number of epochs to train for
patience: 50  # epochs to wait for no observable improvement for early stopping of training
batch: 16  # number of images per batch (-1 for AutoBatch)
imgsz: 640  # size of input images as integer or w,h
save: True  # save train checkpoints and predict results
save_period: -1 # Save checkpoint every x epochs (disabled if < 1)
cache: False  # True/ram, disk or False. Use cache for data loading
device:  # device to run on, i.e. cuda device=0 or device=0,1,2,3 or device=cpu
workers: 8  # number of worker threads for data loading (per RANK if DDP)
project:  # project name
name:  # experiment name, results saved to 'project/name' directory
exist_ok: False  # whether to overwrite existing experiment
pretrained: False  # whether to use a pretrained model
optimizer: SGD  # optimizer to use, choices=['SGD', 'Adam', 'AdamW', 'RMSProp']

verbose: True  # whether to print verbose output
```

```
seed: 0  # random seed for reproducibility
deterministic: True  # whether to enable deterministic mode
single_cls: False  # train multi-class data as single-class
image_weights: False  # use weighted image selection for training
rect: False  # rectangular training if mode='train' or rectangular validation if mode='val'
cos_lr: False  # use cosine learning rate scheduler
close_mosaic: 0  # (int) disable mosaic augmentation for final epochs
resume: False  # resume training from last checkpoint
amp: True  # Automatic Mixed Precision (AMP) training, choices=[True, False], True runs AMP
check
# Segmentation
overlap_mask: True  # masks should overlap during training (segment train only)
mask_ratio: 4  # mask downsample ratio (segment train only)
# Classification
dropout: 0.0  # use dropout regularization (classify train only)

# Val/Test settings ----------------------------------------------------------------------------------------
val: True  # validate/test during training
split: val  # dataset split to use for validation, i.e. 'val', 'test' or 'train'
save_json: False  # save results to JSON file
save_hybrid: False  # save hybrid version of labels (labels + additional predictions)
conf:  # object confidence threshold for detection (default 0.25 predict, 0.001 val)
iou: 0.7  # intersection over union (IoU) threshold for NMS
max_det: 300  # maximum number of detections per image
half: False  # use half precision (FP16)
dnn: False  # use OpenCV DNN for ONNX inference
plots: True  # save plots during train/val

# Prediction settings --------------------------------------------------------------------------------------
source: # source directory for images or videos
show: False  # show results if possible
save_txt: False  # save results as .txt file
save_conf: False  # save results with confidence scores
save_crop: False  # save cropped images with results
show_labels: True  # show object labels in plots
show_conf: True  # show object confidence scores in plots
vid_stride: 1  # video frame-rate stride
line_thickness: 3  # bounding box thickness (pixels)
visualize: False  # visualize model features
augment: False  # apply image augmentation to prediction sources
agnostic_nms: False  # class-agnostic NMS
classes:  # filter results by class, i.e. class=0, or class=[0,2,3]
```

```
retina_masks: False  # use high-resolution segmentation masks
boxes: True  # Show boxes in segmentation predictions

# Export settings ------------------------------------------------------------------------------------------
format: torchscript  # format to export to
keras: False  # use Keras
optimize: False  # TorchScript: optimize for mobile
int8: False  # CoreML/TF INT8 quantization
dynamic: False  # ONNX/TF/TensorRT: dynamic axes
simplify: False  # ONNX: simplify model
opset:  # ONNX: opset version (optional)
workspace: 4  # TensorRT: workspace size (GB)
nms: False  # CoreML: add NMS

# Hyperparameters ------------------------------------------------------------------------------------------
lr0: 0.01  # initial learning rate (i.e. SGD=1E-2, Adam=1E-3)
lrf: 0.01  # final learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 7.5  # box loss gain
cls: 0.5  # cls loss gain (scale with pixels)
dfl: 1.5  # dfl loss gain
pose: 12.0  # pose loss gain
kobj: 1.0  # keypoint obj loss gain
label_smoothing: 0.0  # label smoothing (fraction)
nbs: 64  # nominal batch size
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
copy_paste: 0.0  # segment copy-paste (probability)
```

```yaml
# Custom config.yaml ---------------------------------------------------------------------------------------------
cfg:  # for overriding defaults.yaml

# Debug, do not modify -------------------------------------------------------------------------------------------
v5loader: False  # use legacy YOLOv5 dataloader

# Tracker settings -----------------------------------------------------------------------------------------------
tracker: botsort.yaml  # tracker type, ['botsort.yaml', 'bytetrack.yaml']
```

```yaml
# coco128.yaml
# Ultralytics YOLO 🚀, AGPL-3.0 license
# COCO128 dataset https://www.kaggle.com/ultralytics/coco128 (first 128 images from COCO train2017) by Ultralytics
# Example usage: yolo train data=coco128.yaml
# parent
# ├── ultralytics
# └── datasets
#     └── coco128  ← downloads here (7 MB)


# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: ../datasets/coco128  # dataset root dir
train: images/train2017  # train images (relative to 'path') 128 images
val: images/train2017  # val images (relative to 'path') 128 images
test:  # test images (optional)

# Classes
names:
  0: person
  1: bicycle
  2: car
  3: motorcycle
  4: airplane
  5: bus
```

```yaml
# Download script/URL (optional)
download: https://ultralytics.com/assets/coco128.zip
```