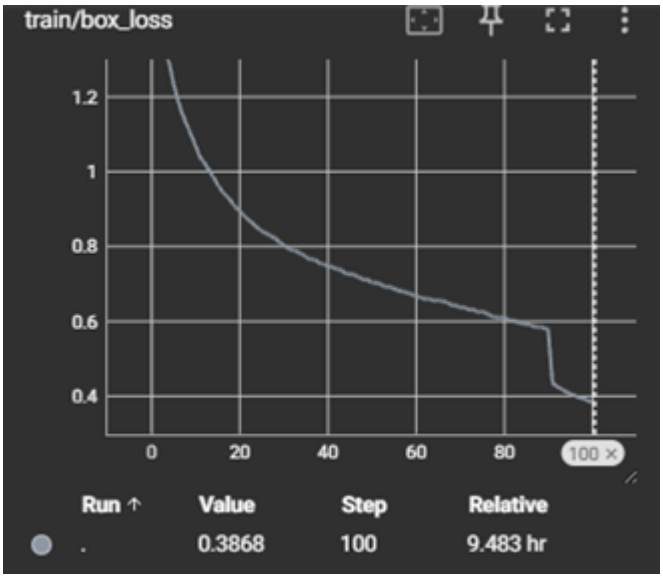
1、yolo11训练后10代损失快速下降

模型训练默认在最后10代默认开启close_mosaic，数据变得简单，因此更易学习。



2、yolo11的多尺度输入

多尺度输入是由multi_scale超参控制，默认是False，即只使用原图输入；

如果设置为True，则会在原图输入的基础上，增加不同尺度的输入。

每一批次的输入尺寸是一样的，不同批次的输入尺寸之间会有变化。

media > dataStore > ReID > .local > lib > python3.10 > site-packages > ultralytics > models > yolo > detect > train.py > DetectionTrainer

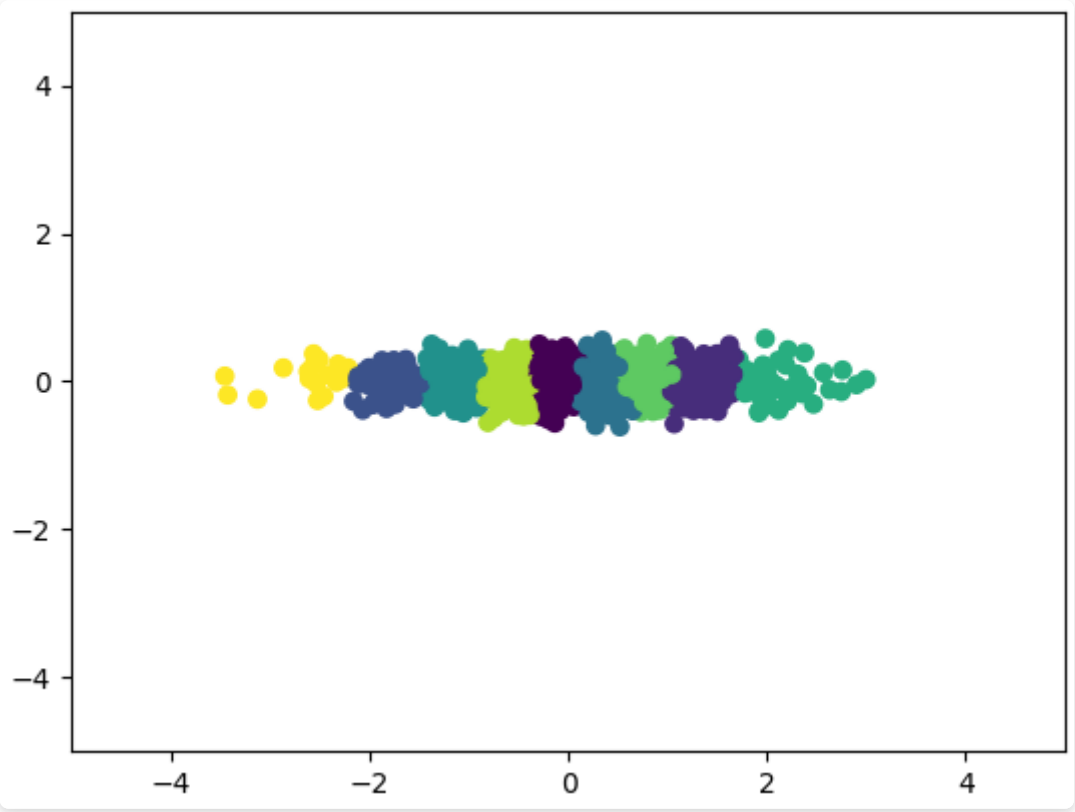
```
19  class DetectionTrainer(BaseTrainer):
45      def get_dataloader(self, dataset_path, batch_size=16, rank=0, mode="train"):
47          assert mode in {"train", "val"}, f"Mode must be 'train' or 'val', not {mode}."
48          with torch_distributed_zero_first(rank):  # init dataset *.cache only once if DDP
49              dataset = self.build_dataset(dataset_path, mode, batch_size)
50          shuffle = mode == "train"
51          if getattr(dataset, "rect", False) and shuffle:
52              LOGGER.warning("WARNING ⚠ 'rect=True' is incompatible with DataLoader shuffle, setting shuffle=False")
53              shuffle = False
54          workers = self.args.workers if mode == "train" else self.args.workers * 2
55          return build_dataloader(dataset, batch_size, workers, shuffle, rank)  # return dataloader
56
57      def preprocess_batch(self, batch):
58          """Preprocesses a batch of images by scaling and converting to float."""
59          batch["img"] = batch["img"].to(self.device, non_blocking=True).float() / 255
60          if self.args.multi_scale:
61              imgs = batch["img"]
62              sz = (
63                  random.randrange(int(self.args.imgsz * 0.5), int(self.args.imgsz * 1.5 + self.stride))
64                  // self.stride
65                  * self.stride
66              )  # size
67              sf = sz / max(imgs.shape[2:])  # scale factor
68              if sf != 1:
69                  ns = [
70                      math.ceil(x * sf / self.stride) * self.stride for x in imgs.shape[2:]
71                  ]  # new shape (stretched to gs-multiple)
72                  imgs = nn.functional.interpolate(imgs, size=ns, mode="bilinear", align_corners=False)
73              batch["img"] = imgs
74          return batch
```

3、yolo anchor-based

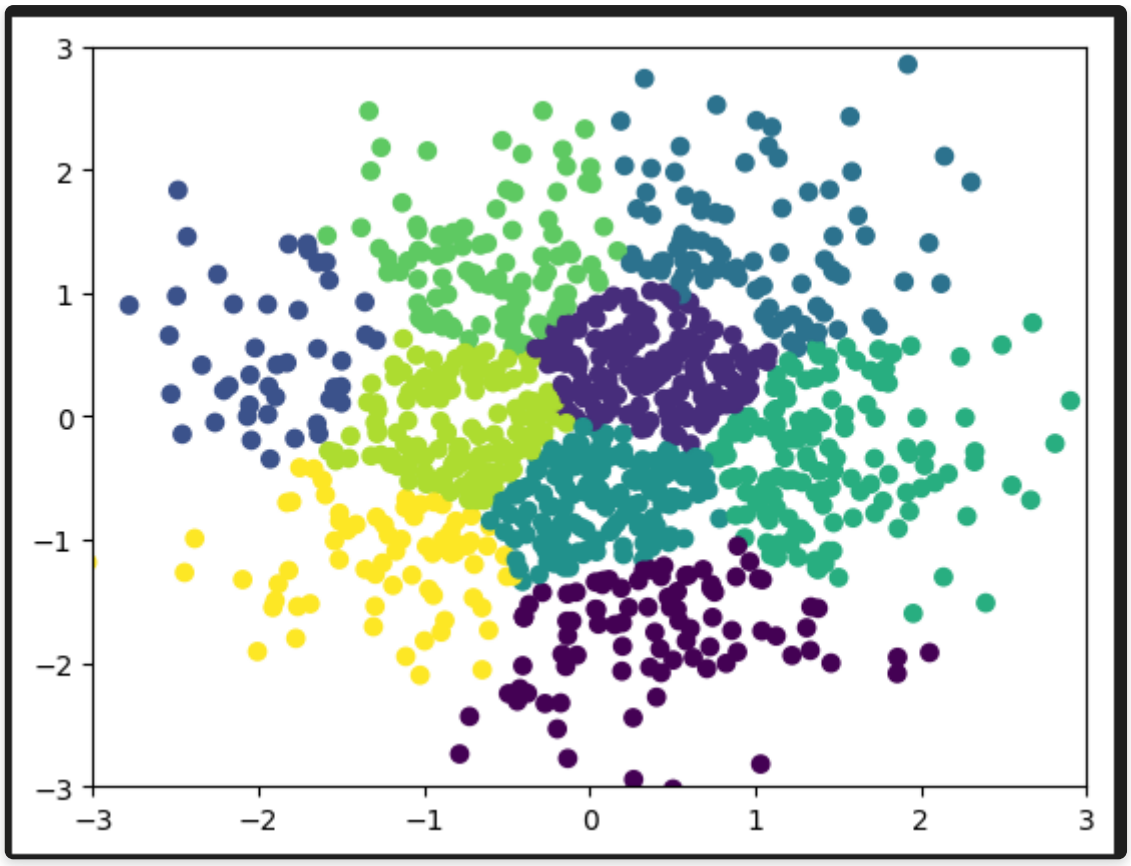
yolov5是使用的anchor-base方法，使用kmeans聚类。
yolov8使用的是anchor-free方法。
yolov5的方法在autoanchor.py文件中实现
yolov8的方法在utils/tal.py文件中实现TaskAlignedAssigner类

yolov5的autoanchor按照宽高来进行聚类，anchor数量是指定的9个。
因为聚类时使用的是欧氏距离,为了防止一个方向上数值占据主导,计算时要对宽高两个方向的数据除以标准差。
得到n个anchor后，对训练数据进行anchor分配，数据和anchor的长宽方向上，任意一个方向的覆盖率达到25%以上，则为该数据分配该锚框。如果有多个锚框满足条件，则选择覆盖率最高的锚框分配。计算出一个fitness值，fitness值越大，则该锚框的效果越好。
再使用遗传算法对锚框进行优化，选取fitness最高的锚框组合。

yolov5会输出3种尺度的特征图，锚框平均分配给这三个尺度的特征图。模型会为特征图上每个点会为每个锚框预测一个置信度和偏移量。



未除以标准差数据聚类结果



除以标准差数据聚类结果

4、yolo anchor-free

Anchor-Based的局限性
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。

anchor-free的方法不需要从训练数据中聚类anchor

yolov11 没有使用anchor-base方法，而是特征图上每个特征点预测1个目标框相四个方向的偏移值和置信度。
偏移值范围是[0,16]，通过特征图上的stride来转换为真实框大小。
同样是3种尺度的特征图，在三种尺度特征图上创建mesh矩阵，然后计算每个特征点对应的框，通过特征图上对应的stride来转换为真实框大小。训练时与gt进行IoU计算，进行框的筛选。