

本章重点以 PyTorch 目标检测项目为例介绍,同时也会给出使用 Paddle 以及 MindSpore 的目标检测代码。

5.1 目标检测任务概述

在图像分类任务中,我们假设图像中只有一个主要物体对象,我们只关注如何识别其类别。然而,很多时候图像里有多个我们感兴趣的目标,我们不仅想知道它们的类别,还想得到它们在图像中的具体位置。如图 5-1 所示,目标(街上的行人、汽车等)的位置通过预测的边界框进行标记,框内物体的类别则以文本标签形式标注。在计算机视觉中,我们将这类任务称为目标检测或目标识别。

目标检测的任务目标:对图像中的物体进行分类,同时检测出该物体(在图像中)的位置。通过目标检测,能够实现自动驾驶中的行人与车辆识别、安防监控中的异常行为检测,甚至是手机中的人脸解锁功能。这一技术已经深入人们日常生活的方方面面,成为人工智能应用的重要基石。本章将带领读者探索目标检测的关键算法和实践技巧,以 YOLOv8 模型为例,体验如何让计算机“看见”并“理解”复杂场景中的多个对象,开启计算机视觉更广阔的应用世界。图 5-2 展示了基于 YOLOv8 实现的无人机追踪系统应用实例。



图 5-1 YOLO 模型检测路上行人、汽车等^①



图 5-2 YOLOv8 无人机追踪器

^① https://blog.csdn.net/qq_41895003/article/details/129456332

1. 数据增强

大型数据集是成功应用深度神经网络的先决条件。图像增广在对训练图像进行一系列的随机变化之后,生成相似但不同的训练样本,从而扩大了训练集的规模。此外,应用图像增广的原因是,随机改变训练样本可以减少模型对某些属性的依赖,从而提高模型的泛化能力。例如,可以以不同的方式裁剪图像,使感兴趣的对象出现在不同的位置,减少模型对于对象出现位置的依赖。还可以调整亮度、颜色等因素来降低模型对颜色的敏感度。

2. 边界框

在目标检测中,通常使用边界框来描述对象的空间位置。边界框是矩形的,由矩形左上角的坐标 (x_1, y_1) 以及右下角的坐标 (x_2, y_2) 决定。另一种常用的边界框表示方法是边界框中心的轴坐标 (x, y) 以及框的宽度 width 和高度 height。

```

1. 0,0 -----> X (width)
2. |
3. | (Left,Top) ,(x1,y1) ,(xmin,ymin)
4. | * -----
5. | |         |
6. | |         |
7. | |         |
8. | * -----
9. (Right,Bottom) ,(x2,y2) ,(xmax,ymax)
10. Y
11. (height)

```

3. 非极大值抑制

同一物体可能被多个重叠框检测到,如同一目标被多个框同时标出,而我们期望的是对于每一个目标有一个预测框,这时就需要进行非极大值抑制处理。

第一步,按照置信度从高到低排序所有预测框。

第二步,保留最高分框,删除与其重叠度(IoU,交并比,交集/并集)超过阈值(如 0.5)的其他框。

第三步,重复第二步直到处理完所有框。

4. 损失函数

目标检测任务中可能会用到多种损失函数,在本例中,YOLOv8 目标检测任务中应用了如下损失函数。

- (1) 分类损失,交叉熵损失,确保正确分类物体类别。
- (2) 定位损失,CIoU Loss,精准预测边界框位置。
- (3) 置信度损失,二元交叉熵,判断框内是否有物体。

5.2 目标检测 PyTorch 实战

5.2.1 PyTorch YOLOv8

PyTorch YOLOv8 目标检测工程使用 YOLOv8 模型检测三种目标:头(head)、头盔(helmet)、安全带(safetybelt)。本项目的文件系统结构如下。

```

1.  /pytorch_yolov8/           # 工程文件夹、主路径
2.    - /weights                # 预训练权重
3.    - /VOCdevkit/VOC2007/     # 数据集
4.      - Annotations 标注.xml 文件、ImageSets 划分数据集.txt 文件、JPEGImages 数据集图像.jpg 文件
5.    - /model_data              # 模型训练权重、字体文件、数据集类别信息等
6.      - simhei.ttf 字体文件、classes.txt 数据集类别信息
7.    - /logs                    # 记录训练过程中的一些信息,如模型断点 checkpoint、损失 loss 等
8.    - /img                     # 测试图像
9.
10.   - /nets                    # package 包
11.     - __init__.py             # 建议给 package 保留一个__init__.py 文件,空文件也行
12.     - backbone.py             # 定义 YOLOv8 的模型主干 backbone
13.     - yolo_net.py             # 定义完整的 YOLOv8 模型
14.     - yolo_for_training.py    # 定义了正负样本选取策略、iou、Bboxloss 等
15.
16.   - /utils                    # package 包
17.     - __init__.py             # 建议给 package 保留一个__init__.py 文件,空文件也行
18.     - callbacks.py            # 定义了 loss 记录函数等
19.     - dataloader.py           # 数据加载器,如何读取数据集、数据增强方法
20.     - utils.py                # 设置随机数种子 seed、分布式训练、图像归一化等
21.     - utils_bbox.py           # 定义了坐标处理的函数,如从模型输出中得到预测的 bbox 坐标等
22.     - utils_fit.py            # 定义模型训练一个 epoch 的函数 fit_one_epoch
23.     - utils_map.py            # 调用官方工具计算模型的 mAP
24.
25.   - cfg.yaml                  # 工程配置
26.   - _model_info.py            # 查看模型的信息
27.   - train.py                  # 训练脚本
28.   - yolo_for_infer.py         # YOLO 推理的封装脚本
29.   - voc_annotation.py         # 数据集构建脚本
30.   - get_map.py                # 获取模型的目标检测指标 mAP

```

5.2.2 工程配置

工程配置 yaml 文件(工程路径下的 cfg.yaml 文件)的好处就是,我们仅需要在这一个配置文件中更新参数或配置,而不需要到对应的.py 文件中更新。本工程配置文件大致如下。

```

1.  # 配置文件
2.  logger_cfg: 'train.log'
3.
4.  data_cfg:
5.    VOCdevkit_path: 'VOCdevkit'
6.    VOCdevkit_sets:
7.      - ['2007', 'train']
8.      - ['2007', 'val']
9.    classes_file: 'model_data/my_classes.txt' #
10.  trainval_percent: 0.9                      # split: trainval, test
11.  train_percent: 0.9                         # split trainval: train, val

```

yaml 可以用 pip 指令安装: pip install pyyaml。在工程中使用配置文件也非常简单。

```

1.  import yaml
2.  def load_cfg():

```

```

3.     with open("path/to/cfg.yaml", "r", encoding = 'utf-8') as file:
4.         return yaml.safe_load(file)
5.     cfg = load_cfg()
6.
7.     model_name = cfg['model_cfg']['name']
8.     # YOLOv8

```

注意事项：科学计数法会被解析成一个字符串,5e-3 写成 0.005 即可,当然也可以将这个解析的字符串转换一下。True、TRUE、true 会被解析成 Python 中的 True,"false"、"true"、false 类似。

5.2.3 数据标注

这里会用到头盔安全带检测数据集,需要我们对图片中的 head、helmet、safetybelt 进行标注。目标检测数据的标注工具用 labeling, labeling 可以通过 pip 安装: pip install labeling。

打开 labeling, 示例中为标注工具 labeling 新建了一个虚拟环境(conda create -n labeling python=3.9), 如图 5-3 所示。

```

(base) PS C:\Program Files\PowerShell\7> conda activate labeling
(labeling) PS C:\Program Files\PowerShell\7> labeling

```

图 5-3 激活虚拟环境

选择 PascalVOC, 保存为 .xml 标注文件格式, 如图 5-4 所示。

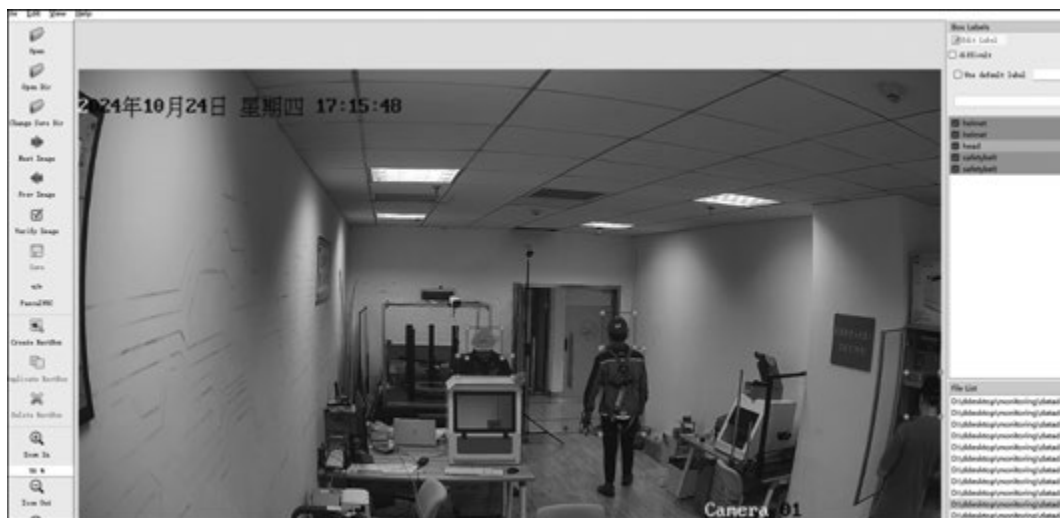


图 5-4 labeling 标注界面

选择图像文件夹, 再选择 .xml 标注文件的保存路径、画框、保存(.xml 标注文件)。

然后可以用 VSCode 打开这个保存的 .xml 文件, 简单查看一下信息。 .xml 文件将图像信息、标注信息按照一定的结构保存起来了。

```

1.     <annotation>
2.         <folder> frames_2 </folder>
3.         <filename> D03_20241024164349_960.jpg </filename>

```

```

4.     <path> D:\ddesktop\monitoring\datadata\frames_2\D03_20241024164349_960.jpg </path>
5.     <source>
6.         <database> Unknown </database>
7.     </source>
8.     <size>
9.         <width> 2560 </width>
10.        <height> 1440 </height>
11.        <depth> 3 </depth>
12.    </size>
13.    <segmented> 0 </segmented>
14.    <object>
15.        <name> helmet </name>
16.        <pose> Unspecified </pose>
17.        <truncated> 0 </truncated>
18.        <difficult> 0 </difficult>
19.        <bndbox>
20.            <xmin> 1153 </xmin>
21.            <ymin> 767 </ymin>
22.            <xmax> 1257 </xmax>
23.            <ymax> 857 </ymax>
24.        </bndbox>
25.    </object>
26.    <object>

```

数据集标注好之后,需要将图片和标注移动到数据集路径下,可以参考之前的工程文件系统结构。当然也可以自定义一个路径,之后会写代码来读取这些数据。

至此,本项目的数据集制作完成。这里选取工程中一些关键的代码进行分析,学习整个工程需要查看完整的工程代码以及注释。

5.2.4 关键代码分析

1. 数据集划分和标签

voc_annotation.py 文件主要做的事情就是处理标签并划分数据集。

annotation_mode=1: 划分数据集,将文件名逐行写到一个.txt文件中,并保存到/VOCdevkit/VOC2007/ImageSets,有 trainval.txt、test.txt、train.txt、val.txt。

annotation_mode=2: 构建完整的数据路径,处理标签到.txt文件 2007_train.txt、2007_val.txt 中。

函数 convert_annotation 用于解析 VOC 格式的标注文件(XML 格式)。

```

1. def convert_annotation(year, image_id, list_file):
2.     in_file = open(os.path.join(VOCdevkit_path, 'VOC%s/Annotations/%s.xml' % (year,
3.     image_id)), encoding='utf-8')
4.     tree = ET.parse(in_file)
5.     root = tree.getroot()
6.     (for) obj in root.iter('object'):
7.         difficult = 0
8.         if obj.find('difficult') != None:
9.             difficult = obj.find('difficult').text

```

```

10.     cls = obj.find('name').text
11.     if cls not in classes or int(difficult) == 1:
12.         continue
13.     cls_id = classes.index(cls)
14.     xmlbox = obj.find('bndbox')
15.     b = (int(float(xmlbox.find('xmin').text)),
16.          int(float(xmlbox.find('ymin').text)),
17.          int(float(xmlbox.find('xmax').text)),
18.          int(float(xmlbox.find('ymax').text)))
19.     list_file.write(" " + ",".join([str(a) for a in b]) + ',' + str(cls_id))
20.
21.     nums[classes.index(cls)] = nums[classes.index(cls)] + 1

```

它首先打开指定年份和图像 ID 对应的 XML 文件,并使用 ElementTree 库(import xml.etree.ElementTree as ET)解析该文件(我们之前查看过,XML 文件信息具有清晰的结构)。然后,遍历所有标注的物体(object 标签),获取每个物体的名称(name)和是否为困难样本(difficult)。如果物体属于指定的类别并且不是困难样本,则提取该物体的边界框坐标(xmin,ymin,xmax,ymax)并将这些信息(坐标和类别 ID)写入指定的列表文件中。最后,函数会更新类别出现的次数。运行 voc_annotation.py,可能会看到如图 5-5 所示信息。

```

Generate txt in ImageSets.
train and val size 108
train size 97
Generate txt in ImageSets done.
Generate 2007_train.txt and 2007_val.txt for train.
Generate 2007_train.txt and 2007_val.txt for train done.
|      head | 11 |
|    helmet | 373 |
| safetybelt | 136 |

```

图 5-5 数据集划分

在 pytorch_yolov8/VOCdevkit/VOC2007/ImageSets/Main 下可以看到有多个.txt 文件生成,这是划分好的数据集。如图 5-6 所示,.txt 文件中的每一行都是文件名(没有.jpg、.xml 后缀)。

<div> <div>VOCdevkit_VOC2007</div> <div> <div>> Annotations</div> <div> <div>imageSets: Main</div> <div> <div>1 README.txt</div> <div>2 test.txt</div> <div>3 train.txt</div> <div>4 trainval.txt</div> <div>5 val.txt</div> </div> </div> </div> </div>	<pre> 10 D02_20241024172241_290 11 D02_20241104143825_26 12 e708e9ca-f5cc-463b-8924-365b93fb9fcd_a_5 13 </pre>
---	--

图 5-6 生成的.txt 文件

关于训练集、测试集、验证集,这里做一个简短的说明。通常来说,人们习惯性地会用训练集来表示训练过程中使用的数据集,用测试集来表示测试过程中使用的数据集,也建议读者习惯这样的方式。但需要知道的是,训练过程中通常会使用两个数据集:训练集和验证集。训练集会使用复杂的数据增强方法,测试集仅进行基础预处理,在训练集上模型是训练模式(model.train()),前向传播、反向传播等都会进行,在验证集上模型是评估模式(model.eval()),只会进行前向传播。测试集用于最终评估模型的性能,模型在测试集上只会进行前向传播(model.eval()),测试集上也仅进行基础预处理。在本例中,训练集 train、

验证集 val、测试集 test,如图 5-7 所示。

训练		测试
训练(train)	验证集(val)	测试集(test)

图 5-7 数据集划分

此外,也生成了 2007_train.txt 和 2007_val.txt,这两个.txt 文件方便后续训练过程中读取样本信息,文件中的每一行可能是这样的:

```
d:\path\to\object_detection\pytorch_yolov8\VOCdevkit\VOC2007\JPEGImages\
08d7893d-d16e-4b12-a426-c333565a3b0c_a_0.jpg 1624,865,1669,918,1 1664,869,1732,
956,2 1926,863,1963,904,1 1885,881,1943,957,2 556,912,594,956,1 437,996,483,
1036,1 351,982,396,1029,1
```

<图片路径> <空格> <标注信息> <空格> <标注信息>...,空格字符的作用是作为分隔符。

从 2007_train/val.txt 读取得到图像和标注,代码示例如下。

```
1. train_lines_path_ = "2007_train.txt"
2. with open(train_lines_path_, encoding='utf-8') as f:
3.     train_lines = f.readlines()
4.     for line in train_lines:
5.         line_content = line.split()
6.         image = Image.open(line_content[0])
7.         image = cvtColor(image)
8.         iw, ih = image.size
9.         box = np.array([np.array(list(map(int, box.split(',')))) for box in line_content[1:]])
```

函数 cvtColor 用于将输入的 image 转换成 RGB 三通道彩色图像。

2. 数据集

通过继承自 torch.utils.data.dataset.Dataset 数据集基类,构建本项目的数据集 YoloDataset,定义数据读取方式、数据增强方法等。然后定义 DataLoader 数据加载器,需要传入 YoloDataset、批量大小、批处理方法等。自定义构建本项目的数据集 YoloDataset。

```
1. from torch.utils.data.dataset import Dataset
2. class YoloDataset(Dataset):
3.     def __init__(self, annotation_lines, input_shape, num_classes, epoch_length,
4.                  mosaic, mixup, mosaic_prob, mixup_prob, train, special_aug_ratio = 0.7):
5.         super(YoloDataset, self).__init__()
6.         self.annotation_lines = annotation_lines # 2007_train/val.txt 中的一行
7.         self.input_shape = input_shape # [640,640]
8.         self.num_classes = num_classes # 类别数
9.         self.epoch_length = epoch_length # 训练总轮数
10.        self.mosaic = mosaic # 是否使用马赛克增强 bool
11.        self.mosaic_prob = mosaic_prob # 应用马赛克增强的概率
12.        self.mixup = mixup # 是否使用 mixup 增强 bool
13.        self.mixup_prob = mixup_prob # 是否使用
14.        self.train = train # bool,用于区分是否使用复杂的数据增强
15.        self.special_aug_ratio = special_aug_ratio # 前百分之多少个 epoch 使用复杂的数据增强
```

```

16.
17.         self.epoch_now    = -1          # 当前的训练的 epoch 数,预设一个初始值
18.         self.length       = len(self.annotation_lines)    # 数据集的长度

```

图像预处理函数 `cvtColor` 和 `preprocess_input` 的源码位于 `utils/utils.py` 文件中。

```

1.  def cvtColor(image):
2.      if len(np.shape(image)) == 3 and np.shape(image)[2] == 3:
3.          return image
4.      else:
5.          # 保证是 RGB 通道顺序的三通道彩色图像
6.          image = image.convert('RGB')
7.          return image
8.
9.  def preprocess_input(image):
10.     # 归一化
11.     image /= 255.0
12.     return image

```

`__getitem__` 函数定义了如何读取数据集中的—个样本,通常返回可以直接用于训练的图像数据张量、标签张量。下面是前处理部分,读取图像数据和标签,对图像数据进行数据增强,图像数据变化了,标签也需要对齐变换之后的图像。

```

1.  def __getitem__(self, index):
2.      index    = index % self.length
3.      if (self.mosaic and self.rand() < self.mosaic_prob
4.          and self.epoch_now <
5.              self.epoch_length * self.special_aug_ratio):
6.          # 从数据集中抽样三个图像数据
7.          lines = sample(self.annotation_lines, 3)
8.          # append 当前索引的图像数据,一共 4 个
9.          lines.append(self.annotation_lines[index])
10.         # 打乱顺序
11.         shuffle(lines)
12.         # 马赛克增强
13.         image, box = self.get_random_data_with_Mosaic(lines,
14.                                                         self.input_shape)
15.         if self.mixup and self.rand() < self.mixup_prob:
16.             # 再抽样一个图像数据,覆盖了之前 lines 变量
17.             lines    = sample(self.annotation_lines, 1)
18.             # self.train 是一个布尔值,是否使用随机增强
19.             image_2, box_2 = self.get_random_data(lines[0],
20.                                                    self.input_shape, random_ = self.train)
21.             # mixup 数据增强
22.             image, box    = self.get_random_data_with_MixUp(
23.                 image, box, image_2, box_2)
24.         else:
25.             # 当前索引的图像数据,数据增强
26.             image, box    = self.get_random_data(
27.                 self.annotation_lines[index], self.input_shape, random_ = self.train)

```

数据格式处理、标注框处理的后半部分代码,最后返回图像张量和标签张量。代码较

长,这里分段进行展示,需要注意一下代码的缩进。

```

1.  # 变换数据的 shape -> nchw
2.  image = np.transpose(preprocess_input(np.array(image, dtype = np.float32)), (2, 0, 1))
3.  box    = np.array(box, dtype = np.float32)
4.
5.  # 对真实框进行预处理
6.  nL      = len(box)
7.  labels_out = np.zeros((nL, 6))
8.  if nL:
9.      # 归一化, YOLO 官方标注的格式是 xywh
10.     box[:, [0, 2]] = box[:, [0, 2]] / self.input_shape[1]
11.     box[:, [1, 3]] = box[:, [1, 3]] / self.input_shape[0]
12.     # 序号为 0,1 的部分,为真实框的中心
13.     # 序号为 2,3 的部分,为真实框的宽高
14.     # 序号为 4 的部分,为真实框的种类
15.     box[:, 2:4] = box[:, 2:4] - box[:, 0:2]
16.     box[:, 0:2] = box[:, 0:2] + box[:, 2:4] / 2
17.     # 调整顺序,符合训练的格式
18.     # 有可能有 4 张图 box 的第一个数 box[0]: 0123 表示位置,box 的位置需要对齐
19.     labels_out[:, 1] = box[:, -1]
20.     labels_out[:, 2:] = box[:, :4]
21.
22.  return image, labels_out

```

现在简单可视化一个 batch 的数据,如在工程主路径下新建一个 scripts.ipynb 文件(也可以是.py 文件)。在这个过程中会实例化刚才自定义的 YoloDataset,并且定义 DataLoader,然后从 DataLoader 取出一个 batch 的数据进行可视化。

导入依赖项:

```

1.  from utils.dataloader import YoloDataset, yolo_dataset_collate
2.  import random, torch
3.  import numpy as np
4.  from torch.utils.data import DataLoader
5.  from functools import partial
6.  import cv2
7.
8.  import matplotlib
9.  # 内嵌显示静态图
10. %matplotlib inline
11. from matplotlib import pyplot as plt

```

一些会用到的函数如下。

```

1.  # utils/utils.py
2.  def get_classes(classes_path):
3.      # 获取数据集类别信息 model_data/my_classes.txt head helmet safetybelt
4.      with open(classes_path, encoding = 'utf-8') as f:
5.          class_names = f.readlines()
6.          class_names = [c.strip() for c in class_names]
7.          return class_names, len(class_names)
8.  def seed_everything(seed = 11):

```

```

9.     # 设置随机种子
10.    random.seed(seed)
11.    np.random.seed(seed)
12.    torch.manual_seed(seed)
13.    torch.cuda.manual_seed(seed)
14.    torch.cuda.manual_seed_all(seed)
15.    torch.backends.cudnn.deterministic = True
16.    torch.backends.cudnn.benchmark = False
17.    def worker_init_fn(worker_id, rank, seed):
18.        # 给每个线程设置随机种子
19.        worker_seed = rank + seed
20.        random.seed(worker_seed)
21.        np.random.seed(worker_seed)
22.        torch.manual_seed(worker_seed)

```

读取训练集信息 2007_val.txt 和 2007_train.txt:

```

1.     # 训练集信息
2.     val_lines_path_ = "2007_val.txt"
3.     train_lines_path_ = "2007_train.txt"
4.     with open(train_lines_path_, encoding='utf-8') as f:
5.         train_lines = f.readlines()
6.     with open(val_lines_path_, encoding='utf-8') as f:
7.         val_lines = f.readlines()

```

设置配置参数:

```

1.     # 配置参数
2.     input_shape = [640,640]
3.     classes_path = "model_data/my_classes.txt"
4.     epoch = 5         # 训练一个 epoch 就是将数据集训练一遍
5.     batch_size = 4    # 一次取出 4 个样本(图像和标注)
6.     num_workers = 0   # 0 不启用多线程. 2 启用两个线程, 加载数据更快, 需根据具体硬件设置.
7.     class_names, num_classes = get_classes(classes_path)

```

实例化数据集:

```

1.     # 实例化数据集,包括数据增强方法等
2.     train_dataset_ = YoloDataset(train_lines, input_shape, num_classes,
3.                                  epoch_length=epoch,
4.                                  mosaic=True, mixup=True, mosaic_prob=0.6,
5.                                  mixup_prob=0.6, train=True,
6.                                  special_aug_ratio=0.8)
7.     # 训练过程中使用的测试集通常并不会使用复杂的数据增强方法
8.     val_dataset_ = YoloDataset(val_lines, input_shape, num_classes,
9.                                 epoch_length=epoch,
10.                                mosaic=False, mixup=False, mosaic_prob=0,
11.                                mixup_prob=0,
12.                                train=False, special_aug_ratio=0)

```

定义数据加载器 DataLoader:

```

1.     # 实例化数据加载器 Dataloader
2.     gen_ = DataLoader(train_dataset_, shuffle = True,

```

```

3.         batch_size = batch_size,
4.         num_workers = num_workers, pin_memory = True,
5.         drop_last = True, collate_fn = yolo_dataset_collate,
6.         sampler = None,
7.         worker_init_fn = partial(worker_init_fn, rank = 0,
8.                                 seed = 11))
9.     gen_val_ = DataLoader(val_dataset_, shuffle = False,
10.                          batch_size = batch_size,
11.                          num_workers = num_workers, pin_memory = True,
12.                          drop_last = True, collate_fn = yolo_dataset_collate,
13.                          sampler = None,
14.                          worker_init_fn = partial(worker_init_fn, rank = 0,
15.                                                  seed = 11))

```

定义可视化一个 batch 的函数 show_batch_annotations()。

```

1.     # 将标注信息绘制到对应的图像上
2.     def show_batch_annotations(my_dataloader_ = gen_):
3.         for iteration, batch in enumerate(my_dataloader_):
4.             images, bboxes = batch
5.             # print(images.shape)           # [batch_size, 3, H, W]
6.             # print(bboxes[0])             # position_id cls_id x y w h
7.             # print(bboxes.shape)          # torch.Size([_, 6])
8.             for i in range(images.size(0)): # 遍历每一张图片
9.                 img = images[i].cpu().numpy().transpose(1, 2, 0)
10.                # 归一化图像到[0, 255]区间
11.                img = np.clip(img * 255, 0, 255).astype(np.uint8)
12.                # print(f"img type:{type(img)}, img shape: {img.shape}")
13.                # img type:<class 'numpy.ndarray'>, img shape: (640, 640, 3)
14.                img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
15.
16.            # 对图像进行标注
17.            for bbox in bboxes:
18.                if i == bbox[0]:
19.                    cls_id = int(bbox[1])
20.                    x_center, y_center, width, height = bbox[2:6]
21.                    # 将中心坐标和宽高转回为左上角和右下角
22.                    x_min = int((x_center - width / 2) * input_shape[0])
23.                    y_min = int((y_center - height / 2) * input_shape[1])
24.                    x_max = int((x_center + width / 2) * input_shape[0])
25.                    y_max = int((y_center + height / 2) * input_shape[1])
26.                    # print(x_min, x_max, y_min, y_max)
27.
28.                    color = (0, 255, 0)      # 设置框的颜色为绿色
29.                    th = 2                    # 设置框的宽度
30.                    # 绘制边框
31.                    cv2.rectangle(img, (x_min, y_min), (x_max, y_max),
32.                                  color, th)
33.                    # 在框上方绘制类别名称
34.                    cv2.putText(img, class_names[cls_id], (x_min, y_min - 10),
35.                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
36.
37.            plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

```

```
38.     plt.axis('off')           # 不显示坐标轴
39.     plt.show()
40.     break
```

调用 `show_batch_annotations(gen_)`, 然后可能会看到 4 张图 (`batch_size=4`), 如图 5-8 所示。

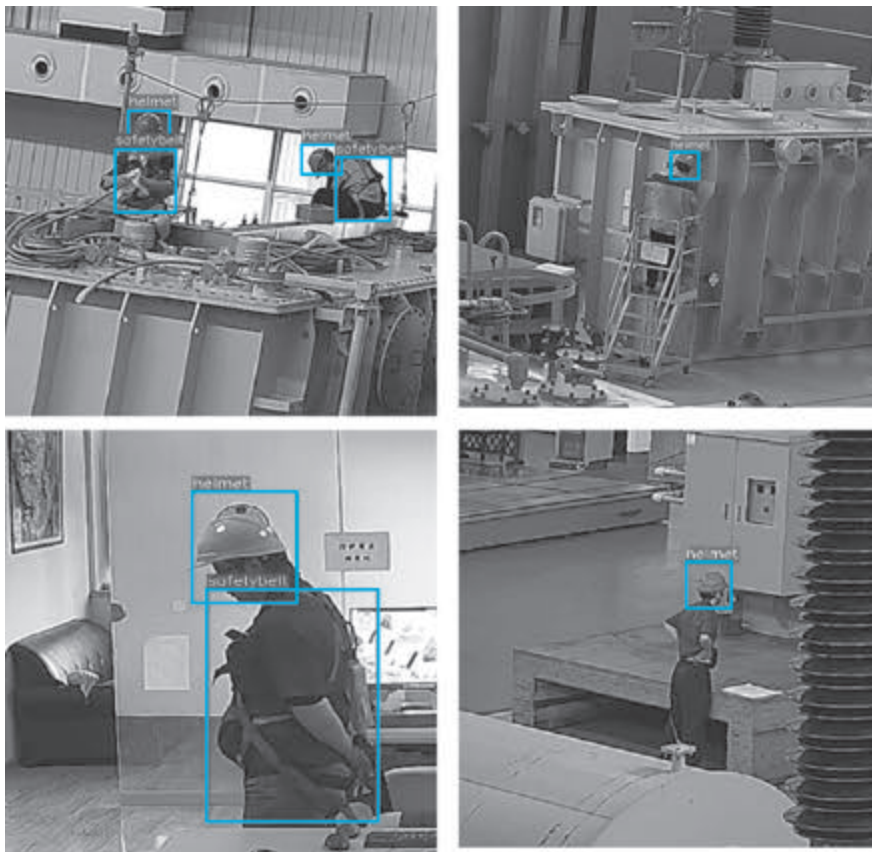


图 5-8 数据增强示例

3. YOLOv8 模型

现在简单了解一下模型部分。YOLOv8 模型的结构如图 5-9 所示。代码文件包括 `nets/backbone.py`, `nets/yolo_net.py`, `nets/yolo_for_raining.py`, 这里不详细介绍代码。

YOLOv8 结构分为三个部分: Backbone、FPN 以及 YOLO Head。

Backbone 是 YOLOv8 的主干特征提取网络, 输入的图片首先会在主干网络里面进行特征提取, 提取到的特征可以被称作特征层, 是输入图片的特征集合。在主干部分, 获取了三个特征层进行下一步网络的构建, 这三个特征层称为有效特征层。

FPN 是 YOLOv8 的加强特征提取网络, 在主干部分获得的三个有效特征层会在这一部分进行特征融合, 特征融合的目的是结合不同尺度的特征信息。在 FPN 部分, 已经获得的有效特征层被用于继续提取特征。在 YOLOv8 里依然使用到了 Panet 的结构, 我们不仅会对特征进行上采样实现特征融合, 还会对特征再次进行下采样实现特征融合。

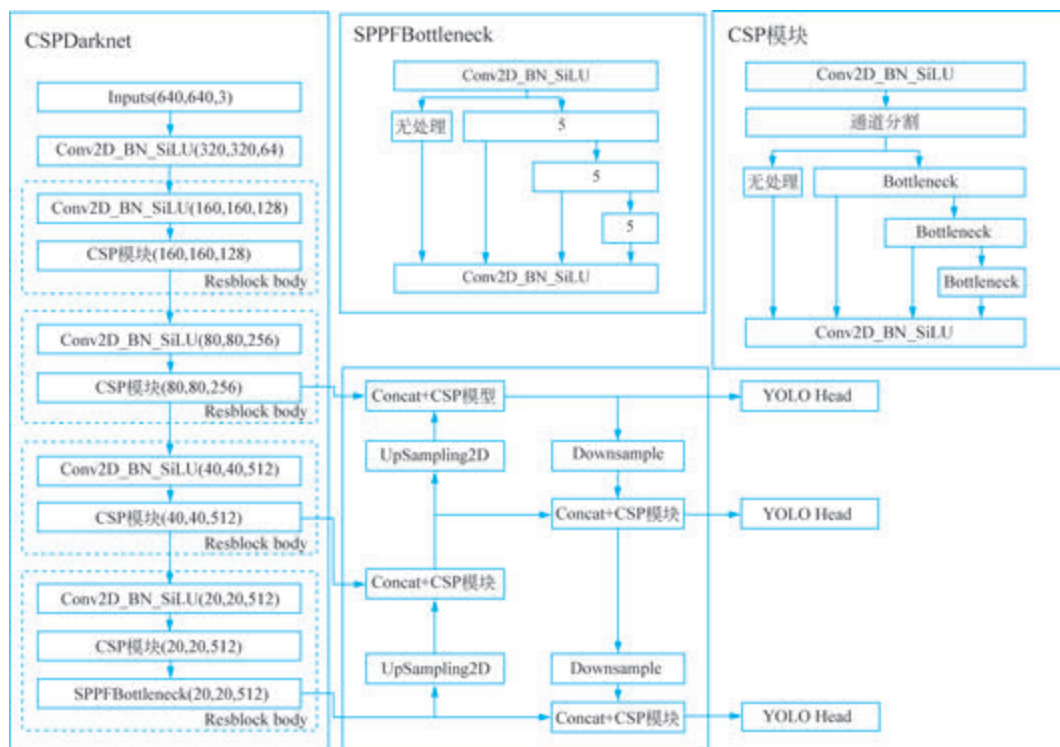


图 5-9 YOLOv8 模型结构

YOLO Head 是 YOLOv8 的分类器与回归器,通过 Backbone 和 FPN,已经可以获得三个加强过的有效特征层。每一个特征层都有宽、高和通道数,此时可以将特征图看作一个又一个特征点的集合,每个特征点作为先验点,而不再存在先验框,每一个先验点都有通道数个特征。YOLO Head 实际上所做的工作就是对特征点进行判断,判断特征点上的先验框是否有物体与其对应。YOLOv8 所用的解耦头是分开的,也就是分类和回归不在一个 1×1 卷积里实现。

因此,整个 YOLOv8 网络所做的工作依然是特征提取-特征加强-预测先验框对应的物体情况。

模型的前向传播过程如下。

首先,从 Backbone 得到三个有效特征层,分别对应深层特征、中层特征和浅层特征:

```
1. def forward(self, x):
2.
3.     # yolov8 s : self.deep_mul = 1.0
4.     # : self.reg_max = 16
5.
6.     # backbone -> 三个有效特征层
7.     feat1, feat2, feat3 = self.backbone.forward(x)
8.     # feat1 shape: n, 1024, 20, 20 n: batch_size
9.     # feat2 shape: n, 512, 40, 40
10.    # feat3 shape: n, 256, 80, 80
```

随后经过加强特征提取网络 FPN。

```

1.  # 加强特征提取网络 FPN                                nchw
2.  P5_upsample = self.upsample(feats3)                    # n, 1024, 20, 20 => n, 1024, 40, 40
3.  P4 = torch.cat([P5_upsample, feat2], 1)                # n, 1024 + 512, 40, 40
4.  P4 = self.conv3_for_upsample1(P4)                      # => n, 512, 40, 40
5.
6.  P4_upsample = self.upsample(P4)                        # n, 512, 40, 40 => n, 512, 80, 80
7.  P3 = torch.cat([P4_upsample, feat1], 1)                # => n, 512 + 256, 80, 80
8.  P3 = self.conv3_for_upsample2(P3)                      # => n, 256, 80, 80
9.
10. P3_downsample = self.down_sample1(P3)                  # => n, 256, 40, 40
11. P4 = torch.cat([P3_downsample, P4], 1)                 # => n, 512 + 256, 40, 40
12. P4 = self.conv3_for_downsample1(P4)                   # => n, 512, 40, 40
13.
14. P4_downsample = self.down_sample2(P4)                  # => n, 512, 20, 20
15. P5 = torch.cat([P4_downsample, feat3], 1)              # => n, 1024 + 512, 20, 20
16. P5 = self.conv3_for_downsample2(P5)                   # => n, 1024, 20, 20
17.
18. # 加强特征提取网络 FPN 的输出:
19. # P3 shape: n, 256, 80, 80
20. # P4 shape: n, 512, 40, 40
21. # P5 shape: n, 1024, 20, 20

```

最后,将特征送入检测头 Head,该检测头将分类任务和回归任务分离处理,分别预测目标的类别和边界框位置。

```

1.  # Head
2.  shape = P3.shape # n, 256, 80, 80
3.  x = [P3, P4, P5]
4.  for i in range(self.nl): # self.nl = 3
5.      # P3 => n, 3 + 16 * 4, 80, 80 num_classes + self.reg_max * 4, 80, 80
6.      # P4 => n, 3 + 16 * 4, 40, 40 num_classes + self.reg_max * 4, 40, 40
7.      # P5 => n, 3 + 16 * 4, 20, 20 num_classes + self.reg_max * 4, 20, 20
8.      x[i] = torch.cat((self.cv2[i](x[i]), self.cv3[i](x[i])), 1)
9.      # 仅改变通道数的卷积操作,然后 cat
10.     # self.cv2 -> 输出通道数是 64, 负责边界框回归
11.     # self.cv3 -> 输出通道数为 3, 负责类别预测
12.
13. if self.shape != shape: # self.shape 初始值是 None
14.     self.anchors, self.strides = (x.transpose(0, 1) for x in make_anchors(x, self.stride, 0.5))
15.     # makeanchor 就是将宽度 W、高度 H 的特征图均分成块,例如,将 n,c,20,20 的输入特征图分
16.     # 成 20×20 块,然后取块的中心点(off_set = 0.5)作为锚点, 20×20 = 400 个锚点
17.     # 每个锚点 anchors 有步长 strides,对应到输入特征图的图像坐标系,用于位置对齐
18.     self.shape = shape
19.
20. # 80×80 + 60×60 + 20×20 = 8400
21. # n, 3 + 64, 8400 | num_classes + self.reg_max * 4, 8400
22. # => cls num_classes, 8400; box self.reg_max * 4, 8400
23. box, cls = torch.cat([xi.view(shape[0], self.no, -1) for xi in x], 2).split(
24.     (self.reg_max * 4, self.num_classes), 1)

```

```

23.
24. # 将边界框预测通过 DFL 模块处理,将[batch_size, 64, 8400]的输出转换为[batch_size, 4, 8400]
25. # DFL 通过将每个坐标(x, y, w, h)建模为分布并取加权和,提高边界框回归精度
26. dbox = self.dfl(box)
27.
28. return dbox, cls, x, self.anchors.to(dbox.device), self.strides.to(dbox.device)

```

Head 之后得到了三个特征层的预测结果,数据集 $\text{num_classes}=3$ 。

80×80 : bbox 回归输出($n, 64, 80, 80$); 类别预测输出($n, 3, 80, 80$)。

40×40 : bbox 回归输出($n, 64, 40, 40$); 类别预测输出($n, 3, 40, 40$)。

20×20 : bbox 回归输出($n, 64, 20, 20$); 类别预测输出($n, 3, 20, 20$)。

将所有特征层在高和宽上平铺后堆叠,可以得到总的回归输出为($n, 4, 8400$),种类输出为($n, 3, 8400$)。

回归输出 4 个参数用于判断每一个特征点的回归参数,回归参数调整后可以获得预测框,前两个序号的内容代表预测框左上角的距离,后两个序号的内容代表预测框右下角的距离。种类输出三个参数用于判断每一个特征点所包含的物体种类。

以 20×20 这个特征图为例,makeanchor 会根据这个特征图制作 400 个特征点(拟定中心点)。

如图 5-10 所示,空心圆点表示 20×20 个特征点,模拟特征图上所有可能的目标中心位置。其中,实心黑点代表当前正在处理的目标的所在位置,即模型预测存在目标的中心点,以下简称“目标点”。针对该目标点,结合回归输出的四个偏移量($\Delta x, \Delta y, \Delta w, \Delta h$),可以解码得到目标的边界框:预测框左上角点坐标,由目标点坐标减去回归预测的前两个偏移量($\Delta x, \Delta y$)得到;预测框右下角点坐标,由目标点坐标加上回归预测的后两个偏移量($\Delta w, \Delta h$)得到。由上述计算得到的左上角和右下角构成一个矩形框,即为目标的预测边界框(如右图中粗黑框所示)。图中两条虚线箭头分别从中心点指向预测框的左上角点和右下角点,表示“减偏移量”和“加偏移量”的解码逻辑。

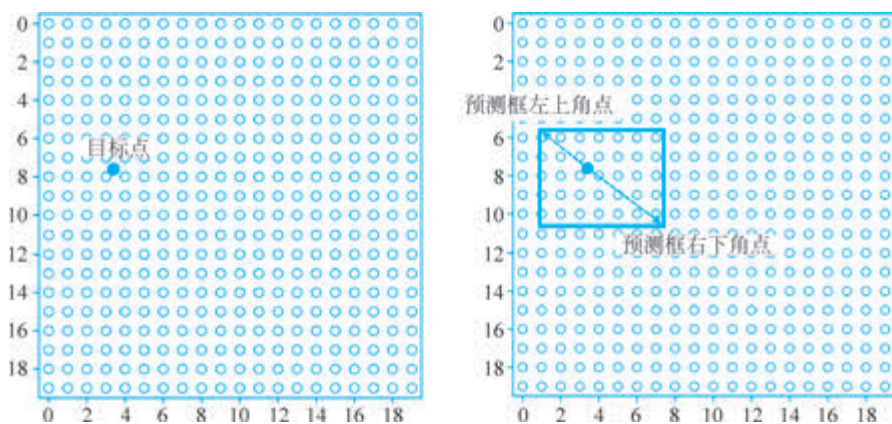


图 5-10 特征点

除去这样的解码操作,还有非极大值抑制,为的是避免同一种类的框堆积在同一个位置。得到最终的预测结果后还要进行得分排序与非极大抑制筛选。得分筛选就是筛选出

得分满足 confidence 置信度的预测框。非极大抑制就是筛选出一定区域内属于同一种类得分最大的框。

得分筛选与非极大抑制的过程可以概括如下。找出该图片中得分大于门限函数的框,在进行重合框筛选前就进行得分的筛选可以大幅度减少框的数量;对种类进行循环,非极大抑制的作用是筛选出一定区域内属于同一种类得分最大的框,对种类进行循环可以帮助我们每一个类分别进行非极大抑制;根据得分对该种类进行从大到小排序;每次取出得分最大的框,计算其与其他所有预测框的重合程度,重合程度过大的则剔除。得分筛选与非极大抑制后的结果就可以用于绘制预测框了,如图 5-11 所示。



图 5-11 NMS 处理前后

_model_info.py 脚本用于查看模型的基本信息结构、参数量、浮点计算次数 FLOPs。使用到了第三方库 thop,可以通过 pip install thop 安装。此外还需要指定以下模型的基本信息,如 input_shape(如[640,640])、num_classes(本项目中是 3,head-helmet-safetybelt)、phi 为's'表示 YOLOv8s。

```
1. import torch
2. from thop import clever_format, profile
3. from nets.yolo_net import YoloBody
4.
5. if __name__ == "__main__":
6.     input_shape = [640, 640]
7.     anchors_mask = [[6, 7, 8], [3, 4, 5], [0, 1, 2]] # 一般不用更改
8.     num_classes = 80
9.     phi = 's'
```

如 YOLOv8s 的 GFLOPS 是 28.651G,参数量为 11.17M。

4. 运行

现在设置一下训练配置,就可以开始训练了。

```
1. if __name__ == "__main__":
2.     os.environ["KMP_DUPLICATE_LIB_OK"] = "True" # 避免多线程下系统层面的报错
3.     # os.environ['CUDA_VISIBLE_DEVICES'] = "0, 1" # 设置 GPU 设备可见
```

```

4.      Cuda                = False      # 是否使用 CUDA 训练
5.      seed                = 11         # 随机数种子
6.
7.      # ----- 可直接使用默认值 -----
8.      distributed          = False      # 分布式训练
9.      sync_bn              = False      # 是否使用 sync_bn, DDP 模式多卡可用
10.     fpl6                 = False      # 是否使用混合精度训练
11.     # -----
12.     classes_path          = "model_data/my_classes.txt"
13.     model_path            = "weights/yolov8_s.pth"
14.     input_shape           = [640,640]
15.     phi                   = "s"
16.     pretrained            = False      # 一般不用更改
17.     mosaic                = True
18.     mosaic_prob            = 0.5
19.     mixup                  = True
20.     mixup_prob             = 0.6
21.     special_aug_ratio      = 0.9
22.     label_smoothing        = 0.005    # 标签平滑
23.     Init_Epoch             = 0         # 从第 0 个 epoch 开始训练
24.     Freeze_Epoch           = 20        # 冻结训练的 epoch 数, 冻结 backbone 的权重,
                                           # 更新其余的权重, 需要的显存小
25.     Freeze_batch_size      = 4         # 冻结训练的批大小, 可以设置大一些, 但通常都是
                                           # 2 的倍数 2 4 8 16
26.     UnFreeze_Epoch         = 100       # 训练的总轮次
27.     Unfreeze_batch_size    = 4         # 批大小
28.     Freeze_Train           = True       # 是否冻结训练, 默认先冻结主干训练后解冻训练
29.     Init_lr                = 1e-2
30.     Min_lr                 = Init_lr * 0.001
31.     optimizer_type         = "sgd"
32.     momentum               = 0.937
33.     weight_decay           = 5e-4      # 权重衰减, 避免过拟合
34.     lr_decay_type          = "cos"
35.     save_period            = 20        # 训练多少个 epoch 保存一次权重
36.     save_dir               = "log"     #
37.     eval_flag              = True      # 是否使用 test 数据集进行测试
38.     eval_period            = 20        # 每隔多少个 epoch 测试一次
39.     num_workers            = 0         # 是否使用多线程
40.     train_annotation_path   = "2007_train.txt"
41.     val_annotation_path    = "2007_val.txt"

```

加载预训练权重的情况, 如图 5-12 所示。

```

initialize network with normal type
Load weights weights/yolov8_s.pth.

Successful Load Key: ['backbone.stem.conv.weight', 'backbone.stem.bn.weight', 'backbone.stem.bn.bias', 'backbone.stem.bn
.running_mean', 'backbone.stem.bn.running_var', 'backbone.stem.bn.num_batches_tracked', 'backbone.dark2.0.conv.weight',
'backbone.dark2.0.bn.weight', 'backbone.dark2.0.bn.bias', 'backbone.dark2.0.bn.running_mean', 'backbone.dark2.0.bn
.running_var', 'backbone.dark2.0.bn.num_batches_tracked', 'backbone.dark2.1.cv1.conv.weight', 'backbone.dark2.1.cv1.bn
.weight', 'backbone.dark2.1.cv1.bn.bias', 'backbo ...
Successful Load Key Num: 349

Fail To Load Key: ['cv3.0.2.weight', 'cv3.0.2.bias', 'cv3.1.2.weight', 'cv3.1.2.bias', 'cv3.2.2.weight', 'cv3.2.2.bias'] ...
Fail To Load Key num: 6

```

图 5-12 模型加载预训练权重

训练的一些配置信息可视化,如图 5-13 所示。

Configurations:		
keys	values	
classes_path	model_data/my_classes.txt	
model_path	weights/yolov8.s.pth	
input_shape	[640, 640]	
Init_Epoch	0	
Freeze_Epoch	2	
UnFreeze_Epoch	10	
Freeze_batch_size	4	
Unfreeze_batch_size	4	
Freeze_Train	True	
Init_lr	0.01	
Min_lr	0.0001	
optimizer_type	sgd	
momentum	0.937	
lr_decay_type	cos	
save_period	1	
save_dir	logs	
num_workers	0	
num_train	97	
num_val	11	

图 5-13 模型配置信息

训练过程可视化,如图 5-14 所示,刚开始 loss 很大,然后开始变小。

```

Start Train
Epoch 1/10: 100%|██████████| 24/24 [00:33<00:00, 1.41s/it, loss=1.38e+3, lr=6.25e-5]
Finish Train
Start Validation
Epoch 1/10: 100%|██████████| 2/2 [00:01<00:00, 1.44it/s, val_loss=686]
Finish Validation
Epoch:1/10
Total Loss: 1377.577 || Val Loss: 686.103
Save best model to best_epoch_weights.pth
Epoch 2/10: 0%|██████████| 0/24 [00:00<?, ?it/s<class 'dict'>]Start Train
Epoch 2/10: 100%|██████████| 24/24 [00:35<00:00, 1.47s/it, loss=122, lr=0.000625]
Epoch 2/10: 0%|██████████| 0/2 [00:00<?, ?it/s<class 'dict'>]Finish Train
Start Validation
Epoch 2/10: 100%|██████████| 2/2 [00:01<00:00, 1.52it/s, val_loss=6.28]
Finish Validation
Epoch:2/10
Total Loss: 122.336 || Val Loss: 6.279
Save best model to best_epoch_weights.pth
Start Train
Epoch 3/10: 12%|██████████| 3/24 [00:06<00:42, 2.03s/it, loss=6.94, lr=0.000601]

```

图 5-14 训练过程可视化

训练过程记录位于 logs 文件夹下,loss 变化位于“loss_时间”文件夹下,这个时间就是开始运行 train.py 的时间,如图 5-15 所示。

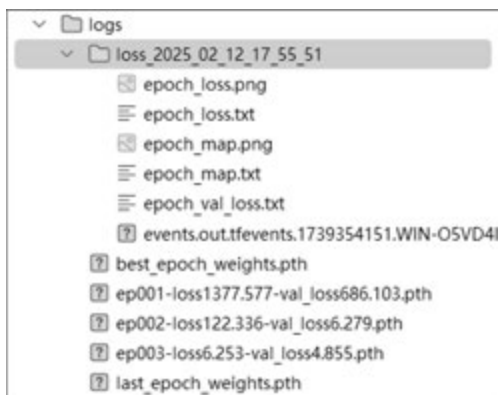


图 5-15 训练过程记录

权重文件保存在 logs 文件夹下,有上一个 epoch 的权重 last_epoch_weights. pth,有 mAP 指标最好的 best_model_weights. pth,以及设置的中间模型权重(隔 save_period 个 epoch 保存一次模型权重)。

现在简单介绍一下训练过程的代码,这部分代码可能并不需要更改,但是希望读者能理解它们。

示例代码并没有使用 CUDA 加速运算,在使用 CUDA 加速运算的情况下需要将数据移动到 GPU 上进行并行运算,并行运算计算效率高、速度快。CUDA 加速运算中的 DP (Data Parallelism,数据并行)模式和 DDP(Distributed Data Parallel,分布式数据并行)模式是两种常见的并行计算方式,通常用于深度学习训练过程中的加速。

DP 模式: 在 DP 模式下,数据被划分成多个小批次,并且每个 GPU(或设备)会处理数据的一部分。每个设备计算梯度并更新自己的权重。通常,多个设备的梯度会被汇总并平均,以便同步更新模型的参数。这种模式的优点是简单易实现,但可能存在较大的通信开销,尤其是在多设备之间需要同步时。

DDP 模式: DDP 模式是分布式训练的升级版,它在多个 GPU 上运行时,每个设备维护自己的模型副本,并计算自己的梯度。在每个训练步骤后,通过高效的通信机制(如 NCCL)将每个设备上的梯度进行同步更新,从而确保每个设备的模型始终一致。DDP 的优势在于通信优化和并行化,通常可以更好地利用分布式资源,特别适用于大规模训练。它在多节点分布式训练中表现尤为突出。

DP 模式适合单节点多 GPU 的训练,且实现简单;而 DDP 模式则适用于多节点训练,能有效利用分布式环境中的计算资源,并减少同步开销。Windows 系统支持 DP 模式。

代码中有一些变量是为了配置 DDP 模式的训练而设计的,下面先了解 DP 模式、CPU 训练的代码逻辑。

指定训练设备的变量 device:

```
1.  if distributed: ...
2.  else:
3.      # 多 GPU DP 模式、单 GPU (默认 DP 模式)、CPU 训练的情况
4.      # 指定训练设备 device
5.      device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```

6.     local_rank      = 0      # 非 DDP 模式训练
7.     rank             = 0      # 非 DDP 模式训练

```

实例化模型,如果 model_path 非空就加载 model_path 指向的预训练权重。

```

1.     # 从 classes_path 获取数据集的类别信息
2.     class_names, num_classes = get_classes(classes_path)
3.     # 定义 YOLO 模型, pretrained 为 False 默认初始化模型权重
4.     model = YoloBody(input_shape, num_classes, phi='s')
5.     if model_path != '': # load weights
6.         ...
7.         # 加载模型权重, 移动到 device 上
8.         pretrained_dict = torch.load(model_path, map_location = device)

```

实例化损失 Loss 和损失记录 LossHistory:

```

1.     # 实例化损失 Loss nets/yolo_for_training.py
2.     yolo_loss = Loss(model)
3.     if local_rank == 0:
4.         # 指定一个 logs_dir 下记录训练过程的文件夹
5.         time_str = datetime.datetime.strftime(datetime.datetime.now(), '%Y_%m_%d_%H_%M_%S')
6.         log_dir = os.path.join(save_dir, "loss_" + str(time_str))
7.         # 实例化记录训练过程损失等的 LossHistory utils/callbacks.py
8.         loss_history = LossHistory(log_dir, model, input_shape = input_shape)

```

设置混合精度训练,示例中 fp16 为 False。混合精度训练(Mixed Precision Training)是一种在深度学习训练中提高计算效率和减少内存消耗的技术。它通过使用不同的数据精度(例如 32 位浮点数和 16 位浮点数)来加速训练过程,同时尽量减少精度损失。设置混合精度训练的代码很简单,这都归功于深度学习框架 PyTorch。

```

1.     # 混合精度训练
2.     if fp16:
3.         from torch.cuda.amp import GradScaler as GradScaler
4.         scaler = GradScaler()
5.     else:
6.         scaler = None

```

训练模式下配置 DP 模式训练,实例化 ema 指数移动平均模型,指数移动平均(Exponential Moving Average)也叫作权重移动平均(Weighted Moving Average),是一种给予近期数据更高权重的平均方法。模型权重在最后的 n 步(step)内,会在实际的最优点处抖动,所以取最后 n 步的平均,能使得模型更加鲁棒。一个 step 就是训练一个 batch_size,训练一个 epoch 就是先后训练 dataset_len//batch_size 个 step,越靠后的 step,模型性能越强,但是也会在实际的最优点处抖动。

```

1.     # 模型切换到训练模式
2.     model_train = model.train()
3.
4.     # DDP 模式下设置 sync_bn
5.
6.     # 是否使用 GPU 训练

```

```

7.  if Cuda:
8.      if distributed: ...                # DDP
9.      else:                             # DP 模式
10.         # torch.nn.DataParallel 包装 model
11.         model_train = torch.nn.DataParallel(model)
12.         cudnn.benchmark = True         # DP 模式下的设置
13.         # 将 model_train 移动到 GPU 上
14.         model_train = model_train.cuda()
15.
16.     # 取最后几个 step 的平均,得到平均模型
17.     ema = ModelEMA(model_train)
18.
19.     # 读取训练集信息
20.     # train、val_annotation_path

```

在可视化训练配置信息和检查数据集能否支持训练代码之后,进入训练代码的逻辑部分。

```

1.  if True:
2.      UnFreeze_flag = False              # 开始冻结训练
3.      if Freeze_Train: ...               # backbone 参数 requires_grad = False
4.
5.      # code: 设置 batch_size
6.      # code: 设置学习率变化范围
7.      # code: 设置优化器的参数更新策略 optimizer.add_param_group(...)
8.      # code: 学习率优化函数
9.      # code: 定义训练 step 数 epoch_step、验证 step 数 epoch_step_val
10.     # code: ema Model
11.     train_dataset = YoloDataset(...)    # 定义数据集
12.     val_dataset = YoloDataset(...)
13.
14.     if distributed: ...                  # DDP
15.     else: # DP 模式、CPU 训练
16.         train_sampler = None            # DDP 模式需设置
17.         val_sampler = None               # DP 模式设为 None 即可
18.         shuffle = True                   # DataLoader()的参数
19.
20.     gen = DataLoader(...)                # 定义数据加载器
21.     gen_val = DataLoader(...)
22.
23.     if local_rank == 0:
24.         eval_callback = EvalCallback(...) # 实例化在验证集上的测试过程
25.
26.     for epoch in range(Init_Epoch, UnFreeze_Epoch):
27.         if epoch >= Freeze_Epoch and not UnFreeze_flag \
28.         and Freeze_Train:                # 解冻训练的第一个 epoch
29.             # code: 学习率范围、学习率策略
30.             # code: backbone 的权重开始更新 requires_grad = True
31.             # code: 检查数据集数量能否支持解冻训练
32.             # code: ema Model
33.             gen = DataLoader(...)         # 定义数据加载器
34.             gen_val = DataLoader(...)
35.             UnFreeze_flag = True         # 值更新为 True,此 if 语句仅执行这一次

```

```

36.
37.     fit_one_epoch(...)                # 传入参数开始训练一个 epoch
38.
39.     # code: loss_history.writer.close() 关闭 loss 记录

```

fit_one_epoch 函数在 utils/uitls.fit.py 文件中,输入参数说明如下。

```

1.     def fit_one_epoch(model_train, model,
2.                        ema, yolo_loss, loss_history,
3.                        eval_callback, optimizer, epoch,
4.                        epoch_step, epoch_step_val, gen,
5.                        gen_val, Epoch, cuda, fp16, scaler,
6.                        save_period, save_dir, local_rank = 0):
7.     """
8.     Args:
9.         model_train    : 训练模型 model.train()实例
10.        model           : model 实例, 会用到 model 实例的属性
11.        ema             : ema model 实例 ema = ModelEMA(model_train)
12.        yolo_loss       : loss 实例 yolo_loss = Loss(model)
13.        loss_history    : LossHistory 的实例
14.        eval_callback   : 验证过程类 EvalCallback 的实例
15.        optimizer       : 优化器实例
16.        epoch           : 当前的训练 epoch 数,索引从 0 开始
17.        epoch_step      : 训练 step 数
18.        epoch_step_val  : 验证 step 数
19.        gen             : 训练集加载器的实例
20.        gen_val         : 验证集加载器的实例
21.        Epoch           : 训练总 epoch 数
22.        cuda            : bool, 是否使用 CUDA
23.        fp16            : bool, 是否要混合精度训练
24.        scaler          : 混合精度训练自动处理工具的实例
25.        save_period     : int, 权重保存的 epoch 间隔
26.        save_dir        : str, (训练过程、训练权重)保存路径
27.        local_rank      : 非 DDP 模式默认为 0
28.    """

```

fit_one_epoch 函数解析,先给 loss 设置一个初始值:

```

1.     # 定义 loss 的初始值
2.     loss      = 0
3.     val_loss  = 0

```

在训练集上进行训练:

```

1.     # code: 定义控制台可视化展示训练过程的信息
2.     for iteration, batch in enumerate(gen): # 训练集上训练
3.         if iteration >= epoch_step:        # 如果训练数据已用完,跳出 for 循环进入验证阶段
4.             break
5.
6.         images, bboxes = batch
7.         # ----- with 语句 -----
8.         with torch.no_grad():
9.             if cuda:

```

```

10.     images = images.cuda(local_rank)      # 将数据从 CPU 搬到 GPU 上
11.     bboxes = bboxes.cuda(local_rank)
12.     # -----
13.     optimizer.zero_grad()                  # 训练开始之前先清空梯度
14.     if not fp16:                            # 非混合精度训练
15.         outputs = model_train(images)
16.         loss_value = yolo_loss(outputs, bboxes)
17.         loss_value.backward()
18.         torch.nn.utils.clip_grad_norm_(model_train.parameters(), max_norm = 10.0) #
            clip gradients
19.         optimizer.step()
20.     else: ...
21.     if ema: ema.update(model_train)          # ema model update
22.     loss += loss_value.item()
23.     # code: 控制台打印 loss、lr 等信息

```

在验证集上测试：

```

1.     # code: 定义控制台可视化展示验证过程的信息
2.     if ema: model_train_eval = ema.ema      # ema model
3.     for iteration, batch in enumerate(gen_val): # 验证集上测试
4.         if iteration >= epoch_step_val:      # 验证数据用完了,跳出循环
5.             break
6.         images, bboxes = batch[0], batch[1]
7.         with torch.no_grad():
8.             if cuda:
9.                 images = images.cuda(local_rank)      # 将数据从 CPU 搬到 GPU 上
10.                bboxes = bboxes.cuda(local_rank)
11.                optimizer.zero_grad()
12.                outputs = model_train_eval(images)
13.                loss_value = yolo_loss(outputs, bboxes)
14.
15.            val_loss += loss_value.item()            # 计算 val_loss
16.            # code: 控制台打印信息

```

统计整个 epoch_val 验证过程的损失：

```

1.     # code: 统计整个 epoch val 测试的情况,并打印信息
2.     eval_callback.on_epoch_end(epoch + 1, model_train_eval)
    # 如果 epoch + 1 等于 eval_period,就会在验证集上粗略计算 mAP 等指标

```

保存模型的逻辑：

```

1.     if ema: save_state_dict = ema.ema.state_dict() # 定义 save_state_dict()
2.     # code: 是否到了指定保存模型权重的 save_period epoch 数,如果是,则保存这个模型
3.     # code: 是否满足条件,保存 best model,训练阶段的第一个模型或者验证集损失 val_loss 最
    # 小的模型
4.     # code: 保存当前 epoch 的训练权重

```

然后将推理的逻辑封装成函数,集成到一个 YOLO_INFER 类中,yolo_for_infer.py 脚本。依赖项导入：

```

1. import colorsys, os, PIL, cv2, av, time, torch, random
2. import numpy as np
3. import torch.nn as nn
4. from PIL import ImageDraw, ImageFont, ImageOps
5. from nets.yolo_net import YoloBody
6. from utils.utils import (cvtColor, get_classes, preprocess_input,
7.                           resize_myimage, show_config, load_cfg)
8. from utils.utils_bbox import DecodeBox, check_version
9. from collections import OrderedDict, defaultdict

```

推理部分。定义 YOLO_INFER 的初始化参数：

```

1. class YOLO_INFER(object):
2.     _defaults = {
3.         "model_path"      : "logs/best_epoch_weights.pth",
4.         "classes_path"    : "model_data/my_classes",
5.         "input_shape"     : [640, 640],
6.         "phi"             : 's',
7.         "confidence"      : 0.5,
8.         "nms_iou"         : 0.3,
9.         "letterbox_image": True,
10.        "cuda"            : False}
11.
12.    @classmethod
13.    def get_defaults(cls, n):
14.        if n in cls._defaults:
15.            return cls._defaults[n]
16.        else:
17.            return "Unrecognized attribute name '" + n + "'"

```

初始化函数，一个类别对应一个颜色，以及模型输出得到预测框的解析方法：

```

1. def __init__(self, **kwargs):
2.     self.__dict__.update(self._defaults)
3.     for name, value in kwargs.items():
4.         setattr(self, name, value)
5.         self._defaults[name] = value
6.
7.     self.class_names, self.num_classes = get_classes(self.classes_path)
8.     self.bbox_util = DecodeBox(self.num_classes, (self.input_shape[0], self.input_shape[1]))
9.
10.    hsv_tuples = [(x / self.num_classes, 1., 1.) for x in range(self.num_classes)]
11.    self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
12.    self.colors = list(map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] *
13.    255)), self.colors))
14.    self.generate()
15.    # show_config(**self._defaults)

```

定义模型加载权重的函数：

```

1. def generate(self, onnx = False):
2.

```

```

3.     self.net = YoloBody(self.input_shape, self.num_classes, self.phi)
4.
5.     # device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
6.     checkpoint = torch.load(self.model_path, map_location='cpu')
7.     new_state_dict = OrderedDict()
8.     for k, v in checkpoint.items():
9.         if k.startswith('module.'):
10.            new_state_dict[k[7:]] = v
11.        else:
12.            new_state_dict[k] = v
13.     self.net.load_state_dict(new_state_dict)
14.
15.     self.net = self.net.fuse().eval()
16.     print('{} model, and classes loaded.'.format(self.model_path))
17.     if not onnx:
18.         if self.cuda:
19.             self.net = nn.DataParallel(self.net)
20.             self.net = self.net.cuda()

```

封装图像检测的方法：

```

1.     def detect_pilimage(self, image, show_ = False):
2.         image_shape = (image.size[1], image.size[0])
3.         image_data = resize_myimage(image, (self.input_shape[1], self.input_shape[0]),
4. self.letterbox_image)
5.         image_data = np.expand_dims(np.transpose(preprocess_input(np.array(image_data,
6. dtype='float32')), (2, 0, 1)), 0)
7.         with torch.no_grad():
8.             images = torch.from_numpy(image_data)
9.             if self.cuda:
10.                images = images.cuda()
11.            outputs = self.net(images)
12.            outputs = self.bbox_util.decode_box(outputs)
13.            results = self.bbox_util.non_max_suppression(
14.                outputs, self.num_classes, self.input_shape,
15.                image_shape, self.letterbox_image,
16.                conf_thres = self.confidence, nms_thres = self.nms_iou
17.            )
18.            if results[0] is None:
19.                return image, None, None, None
20.
21.            top_label = np.array(results[0][:, 5], dtype='int32')
22.            top_conf = results[0][:, 4]
23.            top_boxes = results[0][:, :4]
24.            top_label, top_conf, top_boxes = self.nms_cross_class(top_label, top_conf, top_
25.                boxes, iou_threshold=0.8)
26.
27.            font = ImageFont.truetype(font='model_data/simhei.ttf', size=np.floor(3e-2 *
28.                image.size[1] + 0.5).astype('int32'))
29.            thickness = int(max((image.size[0] + image.size[1]) // np.mean(self.input_shape), 1))
30.
31.            for i, c in list(enumerate(top_label)):
32.                predicted_class = self.class_names[int(c)]

```

```

29.     box = top_boxes[i]
30.     score = top_conf[i]
31.     top, left, bottom, right = box
32.     top = max(0, np.floor(top).astype('int32'))
33.     left = max(0, np.floor(left).astype('int32'))
34.     bottom = min(image.size[1], np.floor(bottom).astype('int32'))
35.     right = min(image.size[0], np.floor(right).astype('int32'))
36.     label_text = '{} {:.2f}'.format(predicted_class, score)

```

画框的逻辑:

```

1.     draw = ImageDraw.Draw(image)
2.     label_size = (
3.         draw.textbbox((0, 0), label_text, font = font)[2] - draw.textbbox((0, 0), label_
4.         text, font = font)[0], \
5.         draw.textbbox((0, 0), label_text, font = font)[3] - draw.textbbox((0, 0), label_
6.         text, font = font)[1]) \
7.         if check_version(PIL.__version__, '9.2.0') else draw.textsize(label_text, font)
8.
9.     if top - label_size[1] >= 0:
10.        text_origin = np.array([left, top - label_size[1]])
11.    else:
12.        text_origin = np.array([left, top + 1])
13.
14.    for i in range(thickness):
15.        draw.rectangle([left + i, top + i, right - i, bottom - i], outline=self.colors[c])
16.    draw.rectangle([tuple(text_origin), tuple(text_origin + label_size)], fill = self.
17.    colors[c])
18.    draw.text(text_origin, label_text, fill = (0, 0, 0), font = font)
19.    del draw
20.
21.    if show_:
22.        image.show()
23.
24.    return image, top_boxes, top_conf, top_label

```

在 inference.py 中调用 YOLO_INFER 来构建不同情况的推理代码:

```

1.     import time
2.     import os
3.     import av
4.     import cv2
5.     import numpy as np
6.     from PIL import Image
7.     from yolo_for_infer import YOLO_INFER

```

```

1.     if __name__ == "__main__":
2.         yolo = YOLO_INFER()
3.         mode = "CLI_input"
4.
5.         # dir_predict
6.         dir_origin_path = r"D:\belthelmet\1107_test\img_test"
7.         dir_save_path = r"D:\belthelmet\1107_test\img_test"

```

```

8.     # 控制台输入图像路径
9.     if mode == "CLI_input":
10.         while True:
11.             img = input('Input image filename:')
12.             try:
13.                 image = Image.open(img)
14.             except:
15.                 print('Open Error! Try again!')
16.                 continue
17.             else:
18.                 rr_image = yolo.dectect_pilimage(image, show_ = True)[0]
19.
20.     # 推理单张图片
21.     elif mode == "predict":
22.         image = Image.open(r"path\to\a.png")
23.         assert image is not None, "图片路径错误或文件不存在!"
24.         r_image = yolo.dectect_pilimage(image)[0]
25.         r_image.show()
26.
27.     # 推理一个文件夹下的图片
28.     elif mode == "dir_predict":
29.         import os
30.         from tqdm import tqdm
31.         img_names = os.listdir(dir_origin_path)
32.         for img_name in tqdm(img_names):
33.             if img_name.lower().endswith(('.bmp', '.dib', '.png', '.jpg', '.jpeg', '.pbm', '.pgm',
34.             '.ppm', '.tif', '.tiff')):
35.                 image_path = os.path.join(dir_origin_path, img_name)
36.                 image = Image.open(image_path).convert('RGB')
37.                 r_image, *_ = yolo.dectect_pilimage(image)
38.                 if not os.path.exists(dir_save_path):
39.                     os.makedirs(dir_save_path)
40.                 r_image.save(os.path.join(dir_save_path, img_name.replace(".jpg", ".png")),
41.                             quality=95, subsampling=0)

```

检测结果示例如图 5-16 所示。



图 5-16 检测结果示例

5.3 目标检测 Paddle 实战

同样还是基于 5.1 节中 PyTorch YOLOv8 目标检测项目的头盔、头、安全带目标检测数据集,现在用 Paddle 来构建 YOLOv8 目标检测项目。这里使用百度 Paddle 官方的仓库 PaddleYOLO。PaddleYOLO 是基于 PaddleDetection^① 的 YOLO 系列模型库,只包含 YOLO 系列模型的相关代码,支持 YOLOv3、PP-YOLO、PP-YOLOv2、PP-YOLOE、PP-YOLOE+、RT-DETR、YOLOX、YOLOv5、YOLOv6、YOLOv7、YOLOv8、YOLOv5u、YOLOv7u、YOLOv6Lite、RTMDet 等模型。

5.3.1 GitHub 仓库、Gitee 仓库下载

1. GitHub 与 Gitee 简介

GitHub^② 是全球最大的开源代码托管平台,支持版本控制和协作开发,汇聚了全球开发者的开源项目。

Gitee 是国内的代码托管平台,提供与 GitHub 类似的功能,且在国内访问速度更快,适合国内开发者协作和下载项目。

2. GitHub 仓库、Gitee 仓库下载

下面以 PaddleYOLO 仓库为例介绍 GitHub 仓库、Gitee 仓库的下载。

PaddleYOLO GitHub 仓库下载:进入 PaddleYOLO GitHub 仓库地址^③,PaddleYOLO 是百度飞桨官方 Paddle^④ 的一个仓库。单击 Code 按钮,在下拉选项中选择 Download ZIP 将仓库的压缩包下载到本地(带下画线的链接需要配置 Git^⑤ 才能使用,在 Git bash 终端或者 PowerShell 中输入“git clone git@github.com:PaddlePaddle/PaddleYOLO.git”,即可将代码仓库克隆到本地,输入 git pull 指令可以拉取该仓库的最近更新到本地),如图 5-17 所示。

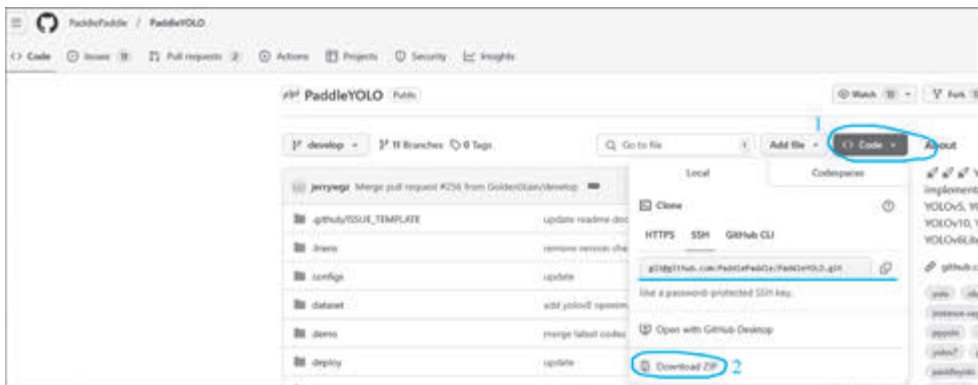


图 5-17 GitHub 仓库下载

① <https://github.com/PaddlePaddle/PaddleDetection>

② <https://github.com/>

③ <https://github.com/PaddlePaddle/PaddleYOLO>

④ <https://github.com/PaddlePaddle>

⑤ <https://git-scm.cn/docs/user-manual>

PaddleYOLO Gitee 仓库下载：进入 PaddleYOLO Gitee 仓库地址^①，类似地，单击“克隆/下载”按钮，如图 5-18 所示。



图 5-18 Gitee 仓库下载

5.3.2 PaddleYOLO 工程

现在调用 PaddleYOLO 仓库来快速构建我们的 Paddle YOLOv8 目标检测项目。首先下载 PaddleYOLO 仓库到本地，大致得到下面的工程结构。

```
1.  ~ /paddle_yolov8
2.    - /daatset
3.      - /VOCDevkit          # 这是我们在 pytorch-yolov8 中用的数据集
4.    - /model_data
5.      - my_classes.txt      # 数据集类别
6.    - /PaddleYOLO          # 下载的 PaddleYOLO 仓库
7.    - /preweights           # 存放预训练权重
8.    - projs_align.ipynb     # 处理数据集和配置脚本
```

在调用 PaddleYOLO 项目之前，需要整理好数据集的格式、工程配置文件。新建一个 projs_align.ipynb 来整理和记录这些事情（需要在 pdet 环境下：pip install jupyter）。

首先，需要配置 PaddleYOLO 仓库的依赖，进到 PaddleYOLO 仓库主目录下：

```
1.  (pdet) PS path\to\paddle_yolov8: cd PaddleYOLO
2.  (pdet) PS path\to\paddle_yolov8\PaddleYOLO: pip install -r requirements.txt
3.  (pdet) PS path\to\paddle_yolov8\PaddleYOLO: python setup.py install
```

PaddleYOLO 仓库的结构：

```
1.  # PaddleYOLO
2.  ~ /PaddleYOLO
3.    - /configs              # 各 YOLO 的配置入口
4.    - /dataset              # 数据集入口，有提供默认数据集的下载、处理脚本
5.    - /demo                 # 几张用于测试的图片
6.    - /deploy               # 部署入口
7.    - /docs                 # 工程文档入口，教程都在这里
8.    - /ppdet # import ppdet # ppdet package 源码（setup.py 中有 package 的构建信息）
9.    - /scripts
10.   - /test_tipc            # Paddle 包装的训练测试开发工具入口
11.   - /tools                # 集成的脚本 如 train.py、infer.py ...
12.   - requirements.txt , setup.py # 用于配置/构建 PaddleYOLO 工程环境
13.   - README.md             # 工程 README.md，优先看
```

^① <https://gitee.com/monkeycc/PaddleYOLO>

5.3.3 PaddleYOLO 数据集

接下来开始整理数据集和数据集、模型、训练等的配置文件(这里建议读者完整地看完 README.md,以及 docs 下的教程文档)。这里直接将 pytorch-yolov8 中整理好的数据集 VOCDevkit(VOC 数据集格式)复制到 paddle_yolov8/dataset 路径下。

现在整理出目标检测数据集 VOCDevkit 的结构和 PaddleYOLO 要求的 VOC 数据集格式。

我们的数据集结构:

```
1. # 我们的数据集:
2. ~ /paddle_yolov8/dataset/VOCDevkit/VOC2007
3.   - /Annotations      # *.xml
4.   - /ImageSets
5.     - /Main           # *.txt(划分的数据集.txt) txt 内每一行都是无后缀的数据文件名
6.   - /JPEGImages       # *.jpg
```

PaddleYOLO 要求的 VOC 数据集格式:

```
1. # 目标数据集:
2. ~ /PaddleYOLO/dataset/paddledet_voc
3.   - label_list.txt
4.   - train.txt :path/to/( *.jpg *.xml) VOCDevkit/VOC2007/ *.jpg VOCDevkit/VOC2007/ *.xml
5.   - val.txt   :path/to/( *.jpg *.xml) VOCDevkit/VOC2007/ *.jpg VOCDevkit/VOC2007/ *.xml
6.   - /VOCDevkit/VOC2007      # 我们的数据集
7.     - /Annotations          # *.xml
8.     - /ImageSets
9.       - /Main               # *.txt
10.      - /JPEGImages         # *.jpg
```

需要在 PaddleYOLO/dataset 下新建 paddledet_voc 文件夹,将我们的数据集 VOCDevkit 以及 model_data 下的 my_classes.txt 文件移动过来,然后需要生成 train.txt、val.txt。将 my_classes.txt 重命名为 label_list.txt。

生成 train.txt、val.txt 的函数:

```
1. def get_paddetVocTxt(dataset_root = "PaddleYOLO/dataset/paddledet_voc", txt_name =
   "trainval.txt"):
2.     txtpth = os.path.join(os.path.join(dataset_root, "VOCDevkit/VOC2007/ImageSets/
   Main"), txt_name)
3.     with open(txtpth, 'r', encoding = 'utf-8') as f:
4.         lines = f.readlines()
5.         lists_path = os.path.join(dataset_root, txt_name)
6.         lists = open(lists_path, 'w', encoding = 'utf-8')
7.
8.         for line in lines:
9.             image_line = os.path.normpath(os.path.join("VOCDevkit/VOC2007/JPEGImages", line.strip()))
10.            anno_line = os.path.normpath(os.path.join("VOCDevkit/VOC2007/Annotations", line.strip()))
11.            lists_line = f"{image_line}.jpg {anno_line}.xml\n"
12.            lists.write(lists_line)
13.        lists.close()
```

train.txt 中的每一行是数据的相对路径,相对于所用数据集路径,也就是说,在 PaddleYOLO 工程下,数据集路径(“dataset/paddledet_voc”)和 train.txt 中的一行(“VOCDevkit\VOC2007\JPEGImages\5d05775c-ec46-4736-9330-720d4ac88c58_a_3.jpg VOCDevkit\VOC2007\Annotations\5d05775c-ec46-4736-9330-720d4ac88c58_a_3.xml”)“拼接”起来就能够得到一个 PaddleYOLO 工程下的相对路径,能够读取到数据。

5.3.4 PaddleYOLO 配置文件

现在为工程整理配置文件,需要关注的文件夹有 PaddleYOLO/configs/datasets(数据集配置)和 PaddleYOLO/configs/yolov8(工程主配置文件、模型、训练等配置),配置文件后缀名.yaml 和 .yml 没有什么区别。在介绍配置文件的一些项目时会用到中文字符,希望读者在工程中使用这些配置文件的时候去掉中文字符。

新建数据集配置文件 PaddleYOLO/configs/datasets/paddledet_voc.yml:

```
1.  metric: VOC           # 目前支持 COCO、VOC、WiderFace 等评估标准
2.  map_type: l1point     # 计算 mAP
3.  num_classes: 3        # 数据集的类别数,不包含背景类,roadsign 数据集为 4 类,其他数据
                           # 需要修改为自己的数据类别
4.
5.  TrainDataset:
6.    !VOCDataset
7.      dataset_dir: dataset/paddledet_voc # 训练集的图片所在文件相对于 dataset_dir 的路径
8.      anno_path: train.txt               # 训练集的标注文件相对于 dataset_dir 的路径
9.      label_list: label_list.txt         # 数据集所在路径,相对于 PaddleDetection 路径
10.     data_fields: ['image', 'gt_bbox', 'gt_class', 'difficult']
                           # 控制 dataset 输出的 sample 所包含的字段,注意此为
                           # 训练集 Reader 独有的且必须配置的字段
11.
12. EvalDataset:
13.   !VOCDataset
14.     dataset_dir: dataset/paddledet_voc # 数据集所在路径,相对于 PaddleDetection 的路径
15.     anno_path: val.txt                 # 验证集的标注文件相对于 dataset_dir 的路径
16.     label_list: label_list.txt         # 标签文件,相对于 dataset_dir 的路径
17.     data_fields: ['image', 'gt_bbox', 'gt_class', 'difficult']
18.
19. TestDataset:
20.   !ImageFolder
21.     anno_path: label_list.txt           # 标注文件所在路径,仅用于读取数据集的类别信
                                           # 息,支持 json 和 txt 格式
22.     dataset_dir: dataset/paddledet_voc
```

新建工程的主配置文件 PaddleYOLO/configs/yolov8/yolov8_s_main.yml:

```
1.  _BASE_: [
2.    '../datasets/paddledet_voc.yml', # 数据集配置
3.    '../runtime.yml',                 # 训练设备、模型导出设置
4.    '_base_/optimizer_500e.yml',      # 优化器配置
5.    '_base_/yolov8_cspdarknet.yml',   # 模型配置
6.    '_base_/yolov8_reader.yml',       # 数据加载配置
7.  ]
8.  depth_mult: 0.33
```

```

9.   width_mult: 0.50
10.
11.  log_iter: 50
12.  snapshot_epoch: 10
13.  # pretrain_weights: configs/yolov8/preweights/yolov8_s_500e_coco.pdparams
14.  weights: output/yolov8_s_paddledetVoc/model_final
15.
16.  TrainReader:
17.    batch_size: 2

```

主配置文件会覆盖其他配置文件(__BASE__变量对应的配置)。

详细的配置文件参数优先级由高到低如下,出现同名参数时,低优先级参数会被高优先级参数覆盖。

- 用户命令行传入参数。
- Python 执行.py 文件中 parser 的默认参数。
- 命令行传入 config 参数对应的.yml、.yaml 文件参数。
- 命令行传入 config 参数对应的.yml、.yaml 文件中__BASE__参数中包含的.yml、.yaml 文件参数。

DDP 模式下才需要设置 sync_bn,到 configs/yolov8/_base_/yolov8_cspdarknet.yml 中更新 stnc_bn 设置为 bn。

```

1.  architecture: YOLOv8
2.  norm_type: bn # ddp sync_bn

```

yolov8_s 模型的预训练权重在 paddle_yolov8/preweights 文件夹下,可能需要将 preweights 文件夹移动到 PaddleYOLO/configs/yolov8 下。CPU 使用预训练权重可能会 loss nan。

5.3.5 运行

打开终端,激活 padet,进入 PaddleYOLO 路径下,设置 CUDA 设备可见。

```
1. set CUDA_VISIBLE_DEVICES = 0
```

使用 GPU 训练,执行指令:

```
1. python tools/train.py -c configs/yolov8/yolov8_s_main.yml --eval
```

开始训练,如图 5-19 所示。

```

[02/17 11:48:38] pddet.utils.checkpoint INFO: ['yolo_head.conv.weight'] in pretrained weight is not used in the model, and it will not be loaded
[02/17 11:48:38] pddet.utils.checkpoint INFO: The shape [80] in pretrained weight yolo_head.conv_cls.0.2.bias is unmatched with the shape [3] in model yolo_head.conv_cls.0.2.bias.
And the weight yolo_head.conv_cls.0.2.bias will not be loaded
[02/17 11:48:38] pddet.utils.checkpoint INFO: The shape [80, 128, 1, 1] in pretrained weight yolo_head.conv_cls.0.2.weight is unmatched with the shape [3, 128, 1, 1] in model yolo_head.conv_cls.0.2.weight. And the weight yolo_head.conv_cls.0.2.weight will not be loaded
[02/17 11:48:38] pddet.utils.checkpoint INFO: The shape [80] in pretrained weight yolo_head.conv_cls.1.2.bias is unmatched with the shape [3] in model yolo_head.conv_cls.1.2.bias.
And the weight yolo_head.conv_cls.1.2.bias will not be loaded
[02/17 11:48:38] pddet.utils.checkpoint INFO: The shape [80, 128, 1, 1] in pretrained weight yolo_head.conv_cls.1.2.weight is unmatched with the shape [3, 128, 1, 1] in model yolo_head.conv_cls.1.2.weight. And the weight yolo_head.conv_cls.1.2.weight will not be loaded
[02/17 11:48:38] pddet.utils.checkpoint INFO: The shape [80] in pretrained weight yolo_head.conv_cls.2.2.bias is unmatched with the shape [3] in model yolo_head.conv_cls.2.2.bias.
And the weight yolo_head.conv_cls.2.2.bias will not be loaded
[02/17 11:48:38] pddet.utils.checkpoint INFO: The shape [80, 128, 1, 1] in pretrained weight yolo_head.conv_cls.2.2.weight is unmatched with the shape [3, 128, 1, 1] in model yolo_head.conv_cls.2.2.weight. And the weight yolo_head.conv_cls.2.2.weight will not be loaded
[02/17 11:48:38] pddet.utils.checkpoint INFO: Finish loading model weights: configs/yolov8/preweights/yolov8_s_500e_coco.pdparams
[02/17 11:48:43] pddet.engine INFO: Epoch: [0] [ 0/49] eta: 23:01:59 lr: 0.000000 loss: 3969.156738 loss_cls: 3955.365234 loss_box: 9.487659 loss_dfl: 4.381764 loss_l1: 2.250870 b
atch_cost: 3.3845 data_cost: 0.0000 ips: 0.9909 images/s

```

图 5-19 PaddleYOLO 开始训练

如果是使用 CPU 训练,请将 runtime.yml 中的 use_gpu 更新为 False。

推理单张图片时,如果使用 CPU 推理,在指令中去掉 CUDA_VISIBLE_DEVICES=0。

```
CUDA_VISIBLE_DEVICES = 0 python tools/infer.py -c configs/yolov8/yolov8_s_main.yml -o
weights = output/yolov8_s_main/best_model.pdparams --infer_img = path/to/demo.jpg --draw_
threshold = 0.5
```

检测结果示例如图 5-20 所示。

```
(ppdet) ~$ python tools/infer.py -c configs/yolov8/yolov8_s_main.yml -o
weights=output/yolov8_s_main/best_model.pdparams --infer_img=demo/000096.jpg --draw_threshold=0.5
[02/19 09:50:14] ppdet.utils.checkpoint INFO: Finish loading model weights: output/yolov8_s_main/best_model.pdparams
[02/19 09:50:14] ppdet.engine INFO: Test loader length is 1, test batch_size is 1.
[02/19 09:50:14] ppdet.engine INFO: Starting predicting .....
100% | 1/1 [00:00:00:00, 1.27it/s]
[02/19 09:50:15] ppdet.engine INFO: Detection bbox results save in output\000096.jpg
```

图 5-20 PaddleYOLO 推理

测试时,如果使用 CPU 测试,在指令中去掉 CUDA_VISIBLE_DEVICES=0。

```
1. CUDA_VISIBLE_DEVICES = 0 python tools/eval.py -c configs/yolov8/yolov8_s_main.yml -o
weights = output/yolov8_s_main/best_model.pdparams --classwise
```

测试结果示例如图 5-21 所示。

```
[02/19 09:56:43] ppdet.utils.checkpoint INFO: Finish loading model weights: output/yolov8_s_main/best_model.pdparams
[02/19 09:56:43] ppdet.engine INFO: Eval loader length is 2, eval batch_size is 8.
[02/19 09:56:43] ppdet.engine INFO: Starting evaluation .....

[02/19 09:56:48] ppdet.engine INFO: Eval iter: 0
[02/19 09:56:50] ppdet.metrics.metrics INFO: Accumulating evaluation results...
[02/19 09:56:52] ppdet.metrics.map_utils INFO: Per-category of VOC AP:
+-----+-----+
| category | AP | category | AP |
+-----+-----+
| head | 0.990 | helmet | 1.000 |
+-----+-----+

[02/19 09:56:52] ppdet.metrics.map_utils INFO: per-category PR curve has output to voc_pr_curve folder.
[02/19 09:56:52] ppdet.metrics.metrics INFO: mAP(0.50, 11point) = 99.50%
[02/19 09:56:52] ppdet.engine INFO: Total sample number: 11, average FPS: 1.543167959687751
```

图 5-21 PaddleYOLO 测试结果

5.4 目标检测 MindSpore 实战

与 PaddleYOLO 项目类似,这里使用 mindyolo 仓库^①来构建 YOLOv8 目标检测项目,数据集依旧使用之前做好的头(head)、头盔(helmet)、安全带(safetybelt)数据集 COVDevkit。

5.4.1 MindYOLO 工程

1. ~path/to/mindyolo
2. - /configs # 配置入口
3. - /demo # 推理 ./predict.py
4. - /deploy # 部署
5. - /docs # 文档、教程

^① <https://github.com/mindspore-lab/mindyolo>

```

6.      - /examples           # 使用案例(可当作参考)
7.      - /mindyolo           # mindyolo 包
8.      - /requirements
9.      - /tests               # 用于测试, 数据集可视化、loss 测试等
10.     - /tutorials           # 配置文件的说明(建议看)、云端应用的教程
11.     - README.md            # 工程介绍, 优先看
12.     - requirements.txt setup.py # 用于配置/构建项目环境
13.     - test.py               # 测试
14.     - train.py              # 训练

```

首先需要安装 requirements.txt 中的包, pip install -r requirements.txt。然后安装 MindYOLO, 官方教程中使用 pip install -e 指令通过 setup.py 来安装 MindYOLO。之前使用 setup.py 用的指令是 python setup.py install。这里就此差异做一个说明。

pip install -e, -e 或 --editable 表示可编辑模式(editable mode), 即以开发模式安装包。pip install -e 类似于 python setup.py develop(传统方法, 已经不再推荐, 但还是可以用), 会在 Python 解释器的 site-packages 目录中创建一个指向当前项目目录的符号链接(sym link), 生成一个记录了当前项目的路径 .egg-link 文件, 而不会直接复制代码。这样, 修改当前目录的源代码后, 无须重新安装, 修改会立即生效。

pip install 类似于 python setup.py install(传统方法, 已经不再推荐, 但还是可以用), 会根据 setup.py 文件中的配置, 将当前目录下的包安装到 Python 环境中。这个过程会将包的文件复制到虚拟环境(或系统环境)中的 site-packages 目录。pip install 会在 site-packages 目录中创建一个 .dist-info 文件夹, 其中包含包的元数据(如版本、依赖项等)。与开发模式不同, 常规安装会将包的所有文件复制到环境(对应虚拟环境的 site-packages 目录)中, 而不是创建符号链接。

到 mindyolo 目录下, 激活 mdspo, 执行 pip install -e 或者 pip install .。

由于 MindSpore 官方已经在 pypi 上发布了 MindYOLO 包, 可以直接用 pip install mindyolo 安装, 但是版本可能会比较旧, 推荐通过 setup.py 的安装方式。要避免虚拟环境中同时出现同一个包的不同版本的情况, 这里需要卸载通过 python setup.py install 安装的 MindYOLO, 指令为 pip uninstall mindyolo。这里以虚拟环境 mdspo 和包 mindyolo 为例介绍 pip uninstall mindyolo 做了什么事情。(mdspo) pip uninstall mindyolo 的逻辑就是到虚拟环境 mdspo 的文件夹 miniconda() 的安装目录下, 将 envs/mdspo/Lib/site-packages 文件夹下的 mindyolo, egg-link, /mindyolo, /mindyolo-xxx.dist-info 删除(这个删除操作类似于粉碎文件), 如果运行 pip uninstall mindyolo 指令的时候出现文件编码报错, 类似于图 5-22。

```

Found existing installation: mindyolo 0.4.0.dev0
ERROR: Exception:
Traceback (most recent call last):
  File "D:\anaconda\envs\mdspo\lib\site-packages\pip\_internal\cli\base_command.py", line 186, in _run_wrapper
    status = _inner_run()
  File "D:\anaconda\envs\mdspo\lib\site-packages\pip\_internal\cli\base_command.py", line 97, in _inner_run
    return self.run(options, args)
  File "D:\anaconda\envs\mdspo\lib\site-packages\pip\_internal\commands\uninstall.py", line 186, in run
    uninstall_pathset = req.uninstall(
  File "D:\anaconda\envs\mdspo\lib\site-packages\pip\_internal\req\req_install.py", line 722, in uninstall
    uninstalled_pathset = UninstallPathSet.from_dist(dist)
  File "D:\anaconda\envs\mdspo\lib\site-packages\pip\_internal\req\req_uninstall.py", line 524, in from_dist
    link_pointer = os.path.normcase(fh.readline().strip())
UnicodeDecodeError: 'gbk' codec can't decode byte 0xae in position 34: illegal multibyte sequence

```

图 5-22 文件编码报错

通常并不建议手动修改第三方库的源码。这个时候可以手动到 `path/to/envs/mdspo/Lib/site-packages` 下搜索 `mindyolo`, 删除相应文件(Windows 会将这些文件移动到回收站)。

5.4.2 MindYOLO 数据集

数据格式转换, 需要将我们的数据集格式转换成 MindYOLO 要求的数据集格式, 目标数据集格式如下。

```

1.  # target voc - ObjecDetection - dataset structure
2.  mdspo_voc:
3.  ~ /path/to/dataset/mdspo_voc
4.  - /VOCDevkit/VOC2007      # my dataset
5.  - /Annotations           # *.xml
6.  - /ImageSets
7.  - /Main                   # *.txt
8.  - /JPEGImages            # *.jpg
9.
10. - /labels                 # line:cls_id normed_x normed_y normed_w normed_h/n
11. - /train                  # *.txt
12. - /val                    # *.txt
13. - /test                   # *.txt
14. - /images
15. - /train                  # *.jpg
16. - /val                    # *.jpg
17. - /test                   # *.jpg
18. - train.txt               # path/to/ *.jpg
19. - val.txt                 # path/to/ *.jpg
20. - test.txt                # path/to/ *.jpg

```

`copy_files` 函数用于将复制过来的 VOCDevkit 数据集整理成目标数据集的格式。

```

1.  def copy_files(src_root = 'dataset/VOCdevkit/VOC2007',
2.                tgt_root = "mindyolo/dataset/mdspo_voc",
3.                mode = 'train'):
4.      os.makedirs(tgt_root, exist_ok = True)
5.      if mode == 'train': data_type = 'train'
6.      elif mode == 'val': data_type = 'val'
7.      else: data_type = 'test'
8.      # src dir
9.      src_img_dir = os.path.join(src_root, "JPEGImages")
10.     src_log_path = os.path.join(src_root, f"ImageSets/Main/{data_type}.txt")
11.     src_lbl_dir = os.path.join(src_root, "Annotations")
12.     # read data name
13.     def get_dataNames(logTxt = src_log_path):
14.         with open(logTxt, "r", encoding = 'utf-8') as f:
15.             return f.readlines()
16.     data_names = get_dataNames()
17.     # target img dir
18.     tgt_img_dir = os.path.join(tgt_root, f"images/{data_type}")
19.     os.makedirs(tgt_img_dir, exist_ok = True)
20.     # # copy images
21.     def copy_imgs(dir1, dir2, fnames):

```

```

22.     for fname in fnames:
23.         fname = fname.strip()
24.         src_img_path = os.path.join(dir1, f"{fname}.jpg")
25.         if os.path.isfile(src_img_path):
26.             tgt_img_path = os.path.join(dir2, f"{fname}.jpg")
27.             shutil.copyfile(src_img_path, tgt_img_path)
28.     copy_imgs(dir1 = src_img_dir, dir2 = tgt_img_dir, fnames = data_names)
29.     # target lbl dir
30.     tgt_lbl_dir = os.path.join(tgt_root, f"labels/{data_type}")
31.     os.makedirs(tgt_lbl_dir, exist_ok = True)
32.     for name_ in data_names:
33.         xml = name_.strip()
34.         tgt_lbl_path = os.path.join(tgt_lbl_dir, f"{xml}.txt")
35.         fil = open(tgt_lbl_path, "w", encoding = 'utf-8')
36.         src_xml_path = os.path.join(src_lbl_dir, f"{xml}.xml")
37.         if os.path.isfile(src_xml_path):
38.             convert_annotations(src_xml_path, src_img_dir, fil)
39.         fil.close()
40.
41.     copy_files(mode = 'train')

```

标签处理函数为 `convert_annotations`, 需要将 XML 标注文件中的标签转换成 YOLO 标签格式。

```

1.     def convert_annotations(lbl_filepath, img_data_dir, label_file):
2.         img_filename = os.path.basename(lbl_filepath).replace('.xml', '.jpg')
3.         tem_ = Image.open(os.path.join(img_data_dir, img_filename)).convert('RGB')
4.         w, h = tem_.size
5.         in_file = open(lbl_filepath, "r", encoding = 'utf-8')
6.         tree = ET.parse(in_file)
7.         root = tree.getroot()
8.         for obj in root.iter('object'):
9.             difficult = 0
10.            if obj.find('difficult') != None:
11.                difficult = obj.find('difficult').text
12.            cls = obj.find('name').text
13.            if cls not in my_classes or int(difficult) == 1: continue
14.            cls_id = my_classes.index(cls)
15.            xmlbox = obj.find('bndbox')
16.            b = (int(float(xmlbox.find('xmin').text)),
17.                int(float(xmlbox.find('ymin').text)),
18.                int(float(xmlbox.find('xmax').text)),
19.                int(float(xmlbox.find('ymax').text)))
20.            x_normed = (b[0] + b[2]) / 2 / w
21.            y_normed = (b[1] + b[3]) / 2 / h
22.            w_normed = abs(b[0] - b[2]) / w
23.            h_normed = abs(b[1] - b[3]) / h
24.            label_file.write(f"{cls_id} {x_normed} {y_normed} {w_normed} {h_normed}\n")

```

获取 `train.txt`、`val.txt`、`test.txt`, 同时需要整理数据文件名为数字类型, 如 `0001`、`000001` 等。函数 `get_num_fname` 如下。

```

1. def get_num_fname(data_dir = "mindyolo/dataset/mdspo_voc", mode = "train"):
2.     if mode == "train": data_t = "train"
3.     elif mode == "val": data_t = "val"
4.     else: data_t = "test"
5.     img_d = os.path.join(data_dir, f"images/{data_t}")
6.     lbl_d = os.path.join(data_dir, f"labels/{data_t}")
7.     ims = {name.split('.')[0] for name in os.listdir(img_d) if name.endswith('.jpg')}
8.     lbs = {name.split('.')[0] for name in os.listdir(lbl_d) if name.endswith('.txt')}
9.     # get train.txt val.txt test.txt
10.    fe = open(os.path.join("mindyolo/dataset/mdspo_voc", data_t + ".txt"), "w", encoding =
        'utf-8')
11.    if ims - lbs == set():
12.        start_num = 1
13.        for i, na in enumerate(ims, start = start_num):
14.            # fe.write(f"./images/{data_t}/{na}.jpg\n")
15.            new_name = f"{i:06d}"
16.            fe.write(f"..../dataset/images/{new_name}.jpg\n")
17.            src_img = os.path.join(img_d, f"{na}.jpg")
18.            dst_img = os.path.join(img_d, f"{new_name}.jpg")
19.            os.rename(src_img, dst_img)
20.            src_lbl = os.path.join(lbl_d, f"{na}.txt")
21.            dst_lbl = os.path.join(lbl_d, f"{new_name}.txt")
22.            os.rename(src_lbl, dst_lbl)
23.        fe.close()
24.    else: print(f"{ims - lbs = }")
25.    get_num_fname(mode = 'train')

```

5.4.3 MindYOLO 配置文件

训练配置文件 configs/mdspo_voc.yaml:

```

1. data:
2.     dataset_name: mdspo_voc
3.
4.     train_set: dataset/mdspo_voc/train.txt
5.     val_set: dataset/mdspo_voc/val.txt
6.     test_set: dataset/mdspo_voc/test.txt
7.
8.     nc: 3
9.
10.    # class names
11.    names: [ 'head', 'helmet', 'safetybelt' ]
12.
13.    train_transforms: []
14.    test_transforms: []

```

主配置文件 configs/yolov8/yolov8s_main.yaml 的详情如下。

```

1. __BASE__: [
2.     './mdspo_voc.yaml',          # 数据集配置
3.     './hyp.scratch.low.yaml',    # 优化器、损失、数据读取、数据增强 low 配置
4.     './yolov8 - base.yaml'      # 模型配置, sync_gn 等

```

```

5. ]
6.
7. # weights = ./preweights/yolov8-s_500e_mAP446-3086f0c9.ckpt
8.
9. overflow_still_update: False
10. network:
11.     depth_multiple: 0.33          # scales module repeats
12.     width_multiple: 0.50          # scales convolution channels
13.     max_channels: 1024

```

非 DDP 模式训练,设置 sync_bn 为 False,configs/yolov8/yolov8-base.yaml 中其余配置按需更新。

```

1. epochs: 500      # total train epochs
2. per_batch_size: 4
3. img_size: 640
4. iou_thres: 0.7
5. conf_free: True
6. sync_bn: False  # ddp True

```

5.4.4 运行

1. 训练

在训练之前,请检查配置文件中是否有中文,如有,请去掉。在 Windows 系统下用 CPU 训练。

```
1. python train.py --config ./configs/yolov8/yolov8s_main.yaml --device_target CPU
```

如果训练 loss 由正常变为 nan,如图 5-23 所示,可以尝试降低学习率 lr_init,如 0.0001。

```

2025-02-19 11:43:22,881 [INFO] Epoch 1/500, Step 6/6, imgsize (640, 640), loss: 9.0479, lbox: 5.7646, lcis: 0.2654, dfl: 3.0179, cur_lr: 0.09940075129270554
2025-02-19 11:43:22,917 [INFO] Epoch 1/500, Step 6/6, step time: 25752.12 ms
2025-02-19 11:43:23,392 [INFO] Saving model to ./runs/2025-02-19-11-40-43/weights/yolov8s_main-1_6.ckpt
2025-02-19 11:43:23,393 [INFO] Epoch 1/500, epoch time: 2.58 min.
2025-02-19 11:44:39,580 [WARNING] overflow, drop step, loss scale adjust to 1024.0
2025-02-19 11:44:39,582 [INFO] Epoch 2/500, Step 6/6, imgsize (640, 640), loss: nan, lbox: nan, lcis: nan, dfl: nan, cur_lr: 0.09880149364471436
2025-02-19 11:44:39,582 [INFO] Epoch 2/500, Step 6/6, step time: 12698.14 ms
2025-02-19 11:44:39,946 [INFO] Saving model to ./runs/2025-02-19-11-40-43/weights/yolov8s_main-2_6.ckpt
2025-02-19 11:44:39,946 [INFO] Epoch 2/500, epoch time: 1.28 min.
2025-02-19 11:45:44,725 [WARNING] overflow, drop step, loss scale adjust to 1024.0

```

图 5-23 loss nan

数据集路径下有一个 train.cache.npy 文件,这是 MindSpore 读取得到的数据集信息,可以通过以下方式解析这个文件。

```

1. import numpy as np
2. npy_path = r"dataset\mdspo_voc\train.cache.npy"
3. train_npy = np.load(npy_path, allow_pickle=True)
4. print(f>Data shape: {train_npy.shape}")      # Data shape: ()
5. print(f>Data type: {train_npy.dtype}")      # Data type: object
6. # train_npy 是一个零维数组(也称为标量数组)。在 NumPy 中,零维数组是一个单一的对象,
  # 没有维度
7. print(f>Data content: {train_npy}")          # 打印 train_npy 的详细内容
8. train_npy_items = train_npy.item()
9. print(f"{type(train_npy_items) = }")        # type(train_npy_items) = <class 'dict'>

```

随后可以得到 `train_npy_items` 字典,然后就可以访问这个嵌套字典,获取到图像路径、标注等信息。

2. 推理

训练过程的记录默认保存到了 `configs/run` 目录下,可以找到保存的 `ckpt` 模型权重文件。推理过程的结果会默认保存到 `configs/runs_infer` 目录下。推理指令如下:

```
1. python demo/predict.py -- config ./configs/yolov8/yolov8s_main.yaml -- weight /path_to_ckpt/WEIGHT.ckpt -- image_path /path/to/img -- device_target CPU
```

3. 测试

生成的 `dataset/mdspo_voc/annotations/instances_val2017.json` 文件用于测试,直接修改脚本 `mindspore_yolov8\mindyolo\examples\finetune_car_detection\crejson.py` 就可以。需要修改两个地方:更新数据集路径和类别映射。

```
1. # 设置数据集路径
2. dataset_path = "dataset/mdspo_voc"
3. images_path = os.path.join(dataset_path, "images/test")
4. labels_path = os.path.join(dataset_path, "labels/test")
5.
6. # 类别映射
7. categories = [
8.     {"id": 0, "name": "head"},
9.     {"id": 1, "name": "helmet"},
10.    {"id": 2, "name": "safetybelt"},
11.    # 添加更多类别
12. ]
```

生成 `dataset/mdspo_voc/annotations/instances_val2017.json` 文件:

```
1. # 为每个分区保存 JSON 文件
2. testdir = r"dataset/mdspo_voc/annotations"
3. os.makedirs(testdir, exist_ok = True)
4. with open(f"{testdir}/instances_val2017.json", "w") as json_file:
5.     json.dump(coco_format, json_file, indent = 4)
```

测试指令:

```
1. python test.py -- config ./configs/yolov8/yolov8s_main.yaml -- weight /path_to_ckpt/WEIGHT.ckpt -- device_target CPU
```

5.5 目标检测模型昇腾平台部署

通常,在服务器或计算机上训练好的目标检测模型若要投入实际应用,还需完成模型部署环节。模型部署是指将训练所得的模型转换为目标硬件平台支持或推荐的格式,并完成推理引擎的集成、性能优化与嵌入式运行环境适配等过程。本节将以华为 Atlas 200I DK A2 开发者套件为部署平台,分别介绍如何将 5.2~5.4 节中训练得到的目标检测模型 YOLO v8s 部署到该开发板上,完成从训练到边缘推理的完整闭环。

5.5.1 Atlas 200I DK A2 开发者套件介绍

Atlas 200I DK A2 是华为推出的一款高性能 AI 开发板,搭载昇腾(Ascend)310 AI 处理器,具备高达 8 TOPS(INT8)的 AI 算力,支持图像、视频等多模态数据的高效推理。该开发板提供丰富的接口(如 USB、HDMI、GPIO、MIPI CSI 等),并兼容 Ubuntu 系统及 CANN(Compute Architecture for Neural Network)软件栈,广泛适用于智能终端、教育实验、机器人、无人机、工业质检等边缘计算场景。Atlas 200I DK A2 外观如图 5-24 所示,基本规格如图 5-25 所示。



图 5-24 Atlas 200I DK A2 外观图

特征	规格
昇腾 AI 处理器	昇腾 310 系列 AI 处理器 <ul style="list-style-type: none"> 1 个 DaVinci V300 AI core(主频 500MHz) 4 个 TAISHAN V200M 处理器核(主频 1.0GHz)
AI 算力 ^a	<ul style="list-style-type: none"> 半精度(FP16): 4 TFLOPS 整数精度(INT8): 8 TOPS
内存	<ul style="list-style-type: none"> 类型: LPDDR4X 速率: 3200Mbps 位宽: 64bit 容量: 4GB 支持 ECC
存储	<ul style="list-style-type: none"> 内置 SPI flash 提供一个 Micro SD 卡接口, 类型为 SD 3.0, 向下兼容 SD 2.0 标准 推荐使用 SD 3.0 接口标准的 Micro SD 卡 提供一个 M.2 Key M 连接器, 可扩展 M.2224/2280 形态 SSD, 支持 NVMe
编解码能力	<ul style="list-style-type: none"> 支持 H.264/H.265 Decoder 硬件解码, 20 路 1080P(1920×1080)30FPS, YUV420 支持 H.264/H.265 Decoder 硬件解码, 2 路 4K(3840×2160)75FPS, YUV420 支持 H.264/H.265 Encoder 硬件编码, 12 路 1080P(1920×1080)30FPS, YUV420 支持 H.264/H.265 Encoder 硬件编码, 2 路 4K(3840×2160)50FPS, YUV420 JPEG 解码能力 1080P(1920×1080)512FPS, 编码能力 1080P(1920×1080)256FPS, 最大分辨率: 16384×16384, 最小分辨率 32×32
模组接口	提供一个 MXM 连接器用于连接不同规格的 Atlas 200I A2 加速模块
外设接口	<ul style="list-style-type: none"> 40Pin 扩展接口: 1 个 USB Type A 接口: 2 个 HDMI 接口: 2 个 (仅 HDMI0 支持图形桌面) USB Type C 接口: 1 个 M.2 Key M 连接器 (支持半长 (2242) 与全长 (2280)) Micro SD 卡接口: 1 个 M.2 Key E 接口: 1 个 MIPI-DSI 接口: 1 个, 4Lane MIPI-CSI 连接器: 2 个, 8Lane 风扇接口: 1 个 千兆网口: 2 个 板载 MIC: 1 个
功耗	<ul style="list-style-type: none"> 工作电压: 12V 典型功耗: 24W
结构尺寸(长×宽×高)	139mm×136mm×46mm
净重	330g
^a 稳定提供的峰值算力。	

图 5-25 Atlas 200I DK A2 基本规格

5.5.2 Atlas 200I DK A2 开发流程

华为 Atlas 200I DK A2 开发套件部署目标检测模型,开发流程可归纳如图 5-26 所示,

也可参考官方文档^①。

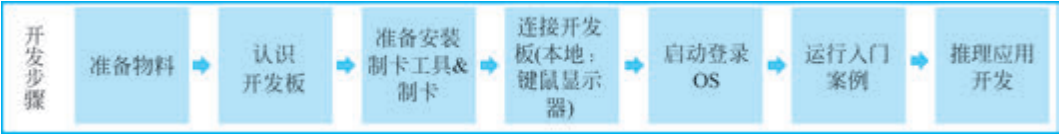


图 5-26 Atlas 200I DK A2 开发流程图

(1) 准备配件,配件详情见图 5-27。

硬件	是否需要额外准备	说明
Atlas 200I DK A2	否	开箱后的Atlas 200I DK A2包括套件主板和电源
Micro SD卡	是	SD卡用于装载镜像运行Atlas 200I DK A2。推荐使用Micro SD卡，容量推荐不小于64GB。
读卡器	是	需使用支持Micro SD卡的读卡器，用于插入SD卡烧录镜像。
PC (笔记本或台式机)	是	用于安装制卡工具、烧录镜像和远程连接Atlas 200I DK A2。 Windows PC配置要求： •操作系统：Windows 10、Windows 11 •具备USB或Type-C接口 •确保C盘剩余空间充足(大于10GB) Mac PC配置要求如下： •操作系统：macOS Big Sur

开发板登录方式	所需配件	优点
本机显示模式	显示器、HDMI连接线、键盘、鼠标、USB拓展坞	类windows、操作简单、桌面化开发
远程登录模式	Type-C数据线 或RJ45网络	配件少，操作稍复杂，linux命令行开发

电源适配器

Type-C数据线

开发者套件

容量64G以上SD卡

读卡器

图 5-27 Atlas 200I DK A2 所需配件

配件有：Atlas 200I DK A2 开发板、Micro SD 卡、读卡器、笔记本电脑或者计算机。

开发板登录方式：本机显示模式(本例采用)、远程登录模式。

Atlas 200I DK A2 开发板开机、关机、下电基本操作介绍如下。

开机：接上电自动开机，LED1 与 LED3 绿灯亮，网络正常连通，代表开发者套件已正常开机运行，如图 5-28 所示。



图 5-28 LED1 与 LED3 绿灯亮

关机：开机状态下，单击系统右下角关机图标关机，网络无法连通，LED1 与 LED3 绿灯灭，只有 D3 绿灯亮表示开发者套件已安全关机。

下电：待关机后，LED1 与 LED3 绿灯灭，从电源接口处拔掉电源线切断外部电源，将开发者套件完全下电。

^① https://www.hiascend.com/document/detail/zh/Atlas200IDKA2DeveloperKit/23.0.RC2/qs/qs_0018.html

(2) 镜像烧录流程如图 5-29 所示。



图 5-29 镜像烧录

镜像烧录好之后，就可以连接和登录到开发板的系统。连接并登录开发板模式本机显示效果如图 5-30 所示。



图 5-30 连接并登录开发板模式本机显示效果

5.5.3 Atlas 200I DK A2 离线推理——目标检测

Atlas 200I DK A2 支持的典型的 AI 开发场景有两种：“推理”和“训练+推理”，如表 5-1 所示。

表 5-1 Atlas 200I DK A2 支持的开发模式

AI 开发场景	开发模式	区 别	适 用 场 景
推理	离线推理	在终端设备上，将网络模型转换成离线格式，然后进行推理	推理性能需求高，不依赖网络、支持离线部署
训练+推理	云端训练+板推理	在云服务器（如华为云、阿里云等）上训练模型，然后将训练好的模型部署到终端设备上推理	大数据集，复杂网络结构，模型训练对算力需求高

这里演示离线推理的开发模式。离线推理主要包括两个步骤：模型转换和推理。

1. 模型转换

由于昇腾 AI 芯片原生支持的模型格式为 OM(Offline Model), 我们需要使用华为提供的 ATC(Ascend Tensor Compiler)工具将训练好的模型转换为 OM 格式。

ATC 工具介绍: 昇腾张量编译器(Ascend Tensor Compiler, ATC)是异构计算架构 CANN 体系下的模型转换工具, 它可以将开源框架的网络模型以及 Ascend IR 定义的单算子描述文件(JSON 格式)转换为昇腾 AI 处理器支持的 .om 格式离线模型。模型转换过程中, ATC 会进行算子调度优化、权重数据重排、内存使用优化等具体操作, 对原始的深度学习模型进行进一步的调优, 从而满足部署场景下的高性能需求, 使其能够高效执行在昇腾 AI 处理器上。

在将 PyTorch、PaddlePaddle 或 MindSpore 模型转换为 OM 模型之前, 通常先将其导出为 ONNX(Open Neural Network Exchange)格式。ONNX 是一种开放的神经网络模型表示标准, 旨在实现不同深度学习框架之间的模型互操作性。通过统一的中间表示, ONNX 能够简化模型在异构硬件平台(如昇腾、GPU、NPU)上的部署流程。在 CPU 端, 安装 ONNX 工具包仅需要在终端执行 `pip install onnx`。

• PyTorch YOLOv8s 转换为 ONNX 模型

```
1.  import torch
2.
3.  # 模型转换 -> onnx model ./model_data/torch_yolov8s.onnx
4.  # 创建虚拟输入(注意: batch_size=1, 通道=3, H=W=640)
5.  dummy_input = torch.randn(1, 3, input_shape[0], input_shape[1])
6.
7.  # 导出 ONNX 模型
8.  onnx_path = r"./model_data/torch_yolov8s.onnx"
9.  torch.onnx.export(
10.     infer_model,                # 加载了权重的 pytorch yolov8 模型
11.     dummy_input,
12.     onnx_path,
13.     export_params=True,          # 存储训练好的参数权重
14.     opset_version=12,           # 推荐使用 opset 11 或 12, 兼容性好
15.     do_constant_folding=True,   # 执行常量折叠优化
16.     input_names=['images'],      # 输入节点名称
17.     output_names=['output0'],   # 输出节点名称(YOLOv8 通常为单输出)
18.     dynamic_axes=None,          # 静态 shape, 便于 Atlas ATC 转换
19.     verbose=False
20. )
21.
22. print(f"✓ PyTorch 模型已成功导出为 ONNX: {onnx_path}")
```

• Paddle YOLOv8s 转换为 ONNX 模型

Paddle 为 Paddle 模型导出为 ONNX 模型设计了一个 Paddle2ONNX 工具, 该工具可使用 `pip install paddle2onnx` 安装。

首先在 paddle YOLO 工程下的 `output/yolov8_s_main` 目录中找到 `best_model`。

pdema、best_model. pdopt、best_model. pdparams, 转换得到静态 Paddle 模型之后才可转换成 ONNX。

```
1. # cd path/to/paddleYOLO
2. # conda activate envq # 切换到指定虚拟环境
3. python tools/export_model.py \
4.     -c configs/yolov8/yolov8_s_main.yml \
5.     -o weights=output/yolov8_s_main/best_model \
6.     --output_dir output/inference_model/yolov8s_paddle
```

然后就可以使用 paddle2onnx 加载静态 Paddle 模型转换成 ONNX 格式了：

```
1. paddle2onnx --model_dir output/inference_model/yolov8s_paddle/yolov8_s_main --model_
   filename model.pdmodel --params_filename model.pdparams --save_file ./output/
   paddle_yolov8s.onnx --opset_version 12
```

• MindSpore YOLOv8s 转换为 ONNX 模型

在 mindyolo 工程路径下执行下面的中断指令：

```
1. python deploy/export.py --config ./configs/yolov8/yolov8s_main.yaml --weight runs/
   2025.02.19-16.53.13/weights/EMA_yolov8s_main-476_24.ckpt --file_format ONNX --
   save_dir runs --device_target CPU
```

ATC 工具将 ONNX 模型转换为 om 模型：

```
1. atc --model=path/to/model_name.onnx \
2.     --framework=5 \
3.     --output=yolo \
4.     --input_format=NCHW \
5.     --input_shape="images:1,3,640,640" \
6.     --log=error \
7.     --soc_version=Ascend310B1
```

其中转换参数的含义如下。

- --model：输入模型路径。
- --framework：原始网络模型框架类型，5 表示 ONNX。
- --output：输出模型路径。
- --input_shape：指定模型输入数据的 shape。
- --log：日志级别。
- --soc_version：昇腾 AI 处理器型号。

成功后生成 yolov8s_640x640.om。

在 Atlas 200I DK A2 上开发和运行代码可使用 Jupyter 工具 JupyterLab 或者 JupyterNotebook, 可通过指令 pip install jupyter 安装, 使用示例如下：

```
1. conda activate your_env
2. cd path/to/unet
3. jupyter lab # 或者 jupyter notebook
```

然后就可以在浏览器上开发和运行代码了。

2. 推理与性能测试

在 Atlas 200I DK A2 开发者套件上部署 YOLOv8s 目标检测推理工程,实例化 YOLOv8s 目标检测推理模型有以下两种方式。

方式 1:“YOLOv8s 源码+YOLOv8s 模型权重”,如 pytorch_yolov8s 源码+pytorch_yolov8s.pth 权重、mindspore_yolov8s 源码+mindspore_yolov8s.ckpt 权重,源码可以参考 5.2 节。

方式 2: 加载包含了 YOLOv8s 模型结构以及权重的模型文件,如 pytorch_yolov8s.om、mindspore_yolov8s.om。

方式 1 需要将 unet 模型源码复制到 Atlas 200I DK A2 机器上,即本实验的 unet 人像分割部署工程的文件系统结构大致如图 5-31 和图 5-32 所示。

/ yolov8s / codes_yolov8s / mindyolo /			/ yolov8s / codes_yolov8s / pytorch_yolov8 /		
☐ Name	▲	Modi...	☐ Name	▲	Modi...
configs		9mo ago	img		2d ago
☒ deploy		9mo ago	model_data		4d ago
dist		9mo ago	nets		8mo ago
mindyolo		9mo ago	utils		8mo ago
mindyolo.egg-info		3d ago	infer.ipynb		17h ago
runs		5d ago	Y: cfg.yaml		4d ago
runs.ipynb		9mo ago	yolo_for_infer.py		2d ago
Y: mkdocs.yml		9mo ago			
requirements.txt		3d ago			
setup.py		9mo ago			

图 5-31 按方式 1 部署 YOLOv8s 目标检测推理工程文件结构示例 1

/ yolov8s / model_infer_yolov8s /		
☐ Name	▲	Mo...
test_img		1h ago
scripts.ipynb		1h ago
mindspore_yolov8s.om		1h ago
my_classes.txt		1h ago
pytorch_yolov8s.om		1h ago

图 5-32 按方式 1 部署 YOLOv8s 目标检测推理工程文件结构示例 2

在 ipynb 文件中开发,首先导入相关依赖:

```
1. from ais_bench.infer.interface import InferSession
2. import cv2
3. import numpy as np
4. from PIL import Image, ImageDraw, ImageFont
5. import time
```

其中 InferSession 是华为昇腾平台加载 om 模型的高级接口,封装好了 om 模型读取和模型推理的逻辑:

```

1.  om_path = "path/to/model_name.om"           # om 模型路径
2.  device_id = "0"                             # 只有一块 Ascend 芯片
3.  # 加载图片预处理得到 input_data
4.  input_data = preprocess_fn("path/to/image")
5.  # 模型加载
6.  om_model = InferSession(device_id, om_path)
7.  # 推理
8.  outs = om_model.infer(input_data)
9.  # 模型输出经过后处理得到最终结果

```

om 模型的推理代码示例如下：

```

1.  def infer_image_bgr(self, img_bgr, draw = True):
2.      h0, w0 = img_bgr.shape[:2]
3.      image_shape = (h0, w0)
4.      inp = self._preprocess_cv2_bgr(img_bgr)
5.      start_ = time.perf_counter()
6.      outputs = self.session.infer([inp])
7.      end_ = time.perf_counter()
8.      print(f"om - model - infer cost: {end_ - start_: .4f} s")
9.
10.     dbox = outputs[0]
11.     cls = outputs[1]
12.     origin_cls = outputs[2]
13.
14.     fm_shapes = [
15.         outputs[2].shape[-2:],
16.         outputs[3].shape[-2:],
17.         outputs[4].shape[-2:],
18.     ]
19.     self._ensure_anchors_and_strides(fm_shapes)
20.     anchors = self.anchors_np
21.     strides = self.strides_np
22.     decoded = self.bbox_util.decode_box((dbox, cls, origin_cls, anchors, strides))
23.     results = self.bbox_util.non_max_suppression(
24.         decoded, self.num_classes, self.input_shape, image_shape,
25.         self.letterbox_image, conf_thres = self.conf_thres, nms_thres = self.iou_thres
26.     )
27.     if results[0] is None:
28.         return None, None, None, None
29.     det = results[0]
30.     top_boxes = det[:, :4]
31.     top_conf = det[:, 4]
32.     top_label = det[:, 5].astype(np.int32)
33.     if not draw:
34.         return None, top_boxes, top_conf, top_label
35.     img_vis = img_bgr.copy()
36.     show_resultIm(top_boxes, top_conf, top_label)

```

推理结果示例如图 5-33 所示。



图 5-33 推理结果示例

练习

按照如下要求动手实现一个目标检测项目。

(1) 选择至少两种类型的检测目标如 target A 和 target B, 可自选常见物体(如不同的水果、树叶, 或者车牌识别等), 搜集对应的图片、标注并制作目标检测数据集(标注工具如 labelme、labelimg 等); 将数据集划分成训练集、测试集。

(2) 搭建如下所示的 SimpleYOLO 目标检测模型。

```

1.  num_classes = 2 # 假设
2.  Input Image : 3 × 128 × 128
3.  SimpleYOLO
4.  |— Backbone → (B, 64, 16, 16)
5.  |   |— Conv 3 × 3, 16ch, ReLU
6.  |   |   |— MaxPool2d(2)           → (B, 16, 64, 64)
7.  |   |— Conv 3 × 3, 32ch, ReLU
8.  |   |   |— MaxPool2d(2)           → (B, 32, 32, 32)
9.  |   |— Conv 3 × 3, 64ch, ReLU
10. |   |   |— MaxPool2d(2)           → (B, 64, 16, 16)
11. |   |— Backbone 输出             → (B, 64, 16, 16)
12. |
13. |— Neck → (B, 64, 16, 16)
14. |   |— Conv 3 × 3, 64ch, ReLU     → (B, 64, 16, 16)
15. |   |— Conv 3 × 3, 64ch, ReLU     → (B, 64, 16, 16)
16. |   |— Neck 输出                 → (B, 64, 16, 16)
17. |
18. |— Head → (B, 7, 16, 16)
19. |   |— Conv 1 × 1, (5 + num_classes) = 7ch

```

```

20. |      (x, y, w, h, conf, class)
21. |
22. └─ 最终输出  reshape / permute
23.   → (B, 16, 16, 7)  # S = 16
24.   - 4 个坐标 (x, y, w, h)
25.   - 1 个 confidence 置信度
26.   - 2 个类别概率

```

(3) 逐步构建整个目标检测工程。

- 模型部分代码,如 model.py。
- 数据集部分代码,包括数据集的读取和预处理,如 dataset.py。
- 数据增强部分代码,如 transforms.py。
- 损失函数部分代码,如 loss.py。
- 优化器、学习率策略部分代码,如 optimizer.py。
- 训练部分代码,如 train.py。
- 模型评估(mAP)部分代码,如 evaluate.py。
- 模型推理测试部分代码,如 infer.py。

(4) 利用训练集数据对 SimpleYOLO 目标检测模型进行训练,保存最佳模型权重为 bestModel.pth。

(5) 利用训练好的模型对测试集进行测试,计算相关评测指标如 mAP,并在测试图像上标注各类目标如 target A 和 target B 的预测位置。

(6) 如果所给的 SimpleYOLO 目标检测模型未能在你的数据集上表现出理想的性能,那么提升检测性能的方案有哪些?

(7) 对比 YOLOv8s 模型与 SimpleYOLO,尝试分析两个模型在网络结构设计上的特点。进行对比实验,比较 YOLOv8s 与 SimpleYOLO 在所构建数据集上的性能差异。