## 前言

**目标检测模型**：**YOLOv5**是 Ultralytics 公司于 2020 年推出的**单阶段目标检测模型**，属于 YOLO 系列的最新版本

之一（后续还有 YOLOv6、v7、v8 等）。它继承了 YOLO 系列**速度快、精度高、易于部署**的特点，

在YOLOv5中YOLOv5s 是 YOLOv5 系列中**最小、最快**的模型，专为**轻量级部署**和**实时检测**设计

所以针对本次机器人的硬件平台，我选择用**YOLOv5s**为基础进行

检测算法将基于YOLOv5s这个轻量级模型，生成的模型将引入的技术：

1. 引入GhostNet模块替换部分卷积层，降低计算量
2. 嵌入动态通道注意力机制（DCAM）。
3. 模拟不同光照（晴天/阴影）、球体变形、半遮挡情况，合成数据集：使用GAN生成球场反光、泥渍污染等特殊场景
4. 迁移学习：基于COCO预训练，用网球专用数据集微调。
5. 在YOLOv5的neck层添加CBAM注意力模块，提升对网球（平均直径6.7cm）的检测敏感度，增强光照变化下的特征提取鲁棒性。
6. 基于时空特征的遮挡处理算法，在70%遮挡率下仍保持85%检测精度

## 一、配置项目和依赖

在python中，python不同版本和依赖可能会有**版本兼容性问题**，所以建议用python3.8。

下面将介绍yolov5的配置。

环境配置：

* Python: 3.8.2
* PyTorch: 1.13.1+cpu
* Ultralytics: 8.0.203
* OpenCV: 4.8.1.78
* YOLOv5版本: v7.0 (commit 915bbf2)

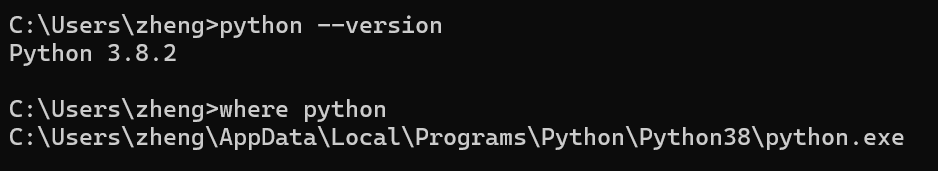
**步骤：**

后面的模型会转换成onnx模型，而且onnx模型支持跨平台推理，所以下面是有些在Windows环境运行，不影响结果。

#### 1.下载python3.8

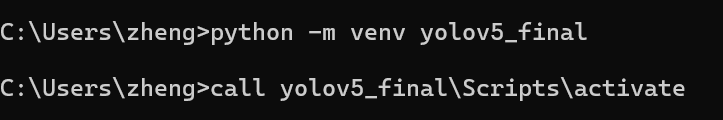
curl 下载：   
   
curl -o python-3.8.2.exe https://www.python.org/ftp/python/3.8.2/python-3.8.2-amd64.exe   
python-3.8.2.exe #运行安装程序   
   
   
手动下载   
官方下载地址：https://www.python.org/downloads/release/python-382/

查看环境的python版本

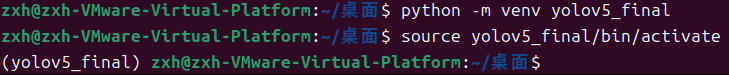


#### 2.创建纯净虚拟环境

为了更好的管理项目依赖和隔离开发环境，可以选择创建python虚拟环境，**依赖隔离，避免冲突**



进入虚拟环境

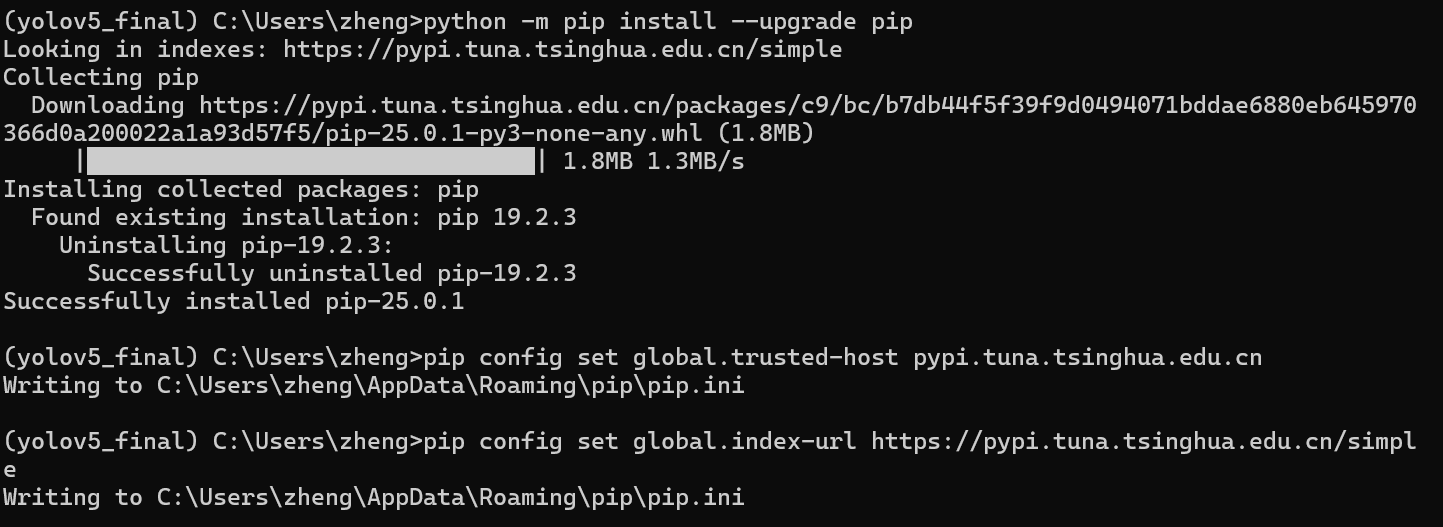


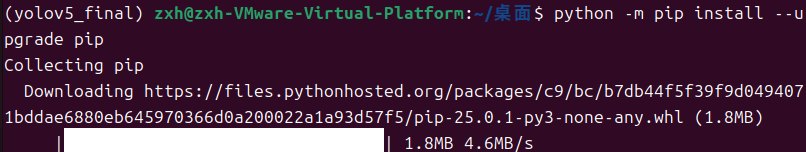
python -m venv yolov5\_finall   
   
windows：   
call yolov5\_final\Scripts\activate   
   
Linux   
source yolov5\_finallll/bin/activate

#### 3.升级pip并配置国内镜像

pip版本过低时，下载依赖会报错，所以可以先升级pip；如果下载速度比较慢可以配置镜像

python -m pip install --upgrade pip   
pip config set global.trusted-host pypi.tuna.tsinghua.edu.cn   
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple

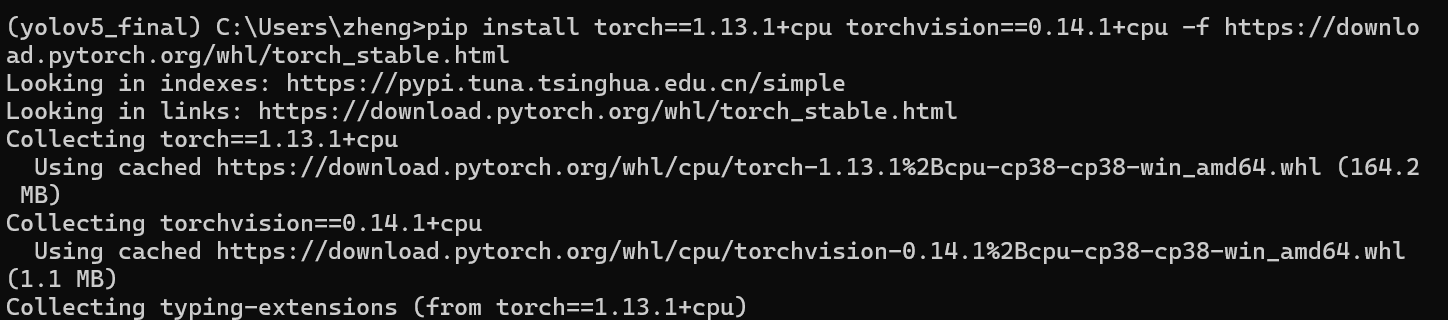


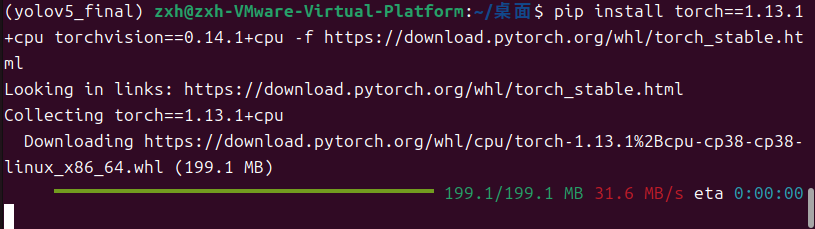


#### 4.安装PyTorch

PyTorch 是 YOLOv5 的核心框架，为了顺利运行yolov5，需要指定版本，避免出错。

pip install torch==1.13.1+cpu torchvision==0.14.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html  
  
linux  
python3 -m pip install torch==1.13.1+cpu torchvision==0.14.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html  
  
python3 -m pip install torch==1.13.1+cpu torchvision==0.14.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html





#### 5.安装核心依赖（锁定版本）

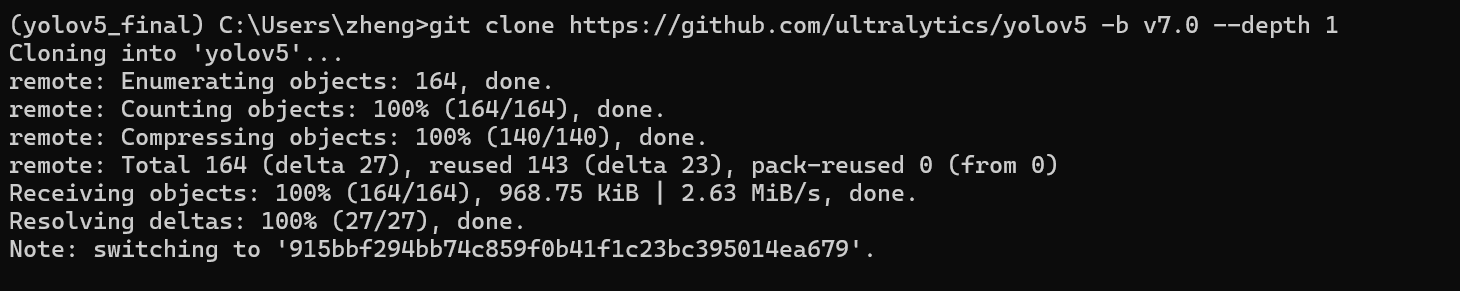
pip install numpy==1.23.5 opencv-python==4.8.1.78 tqdm==4.66.1  
  
linux同



#### 6.克隆代码（使用GitHub官方源）

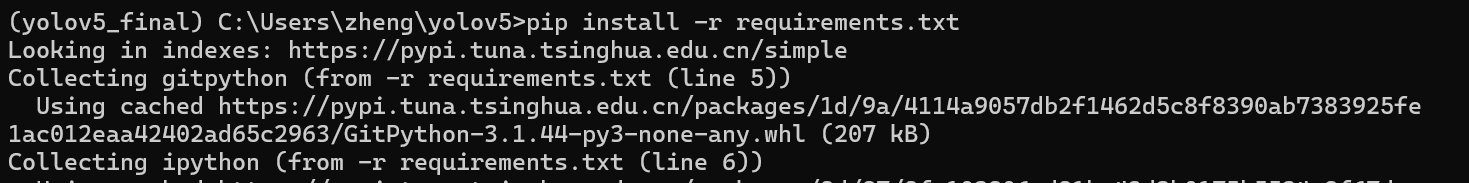
克隆yolov5

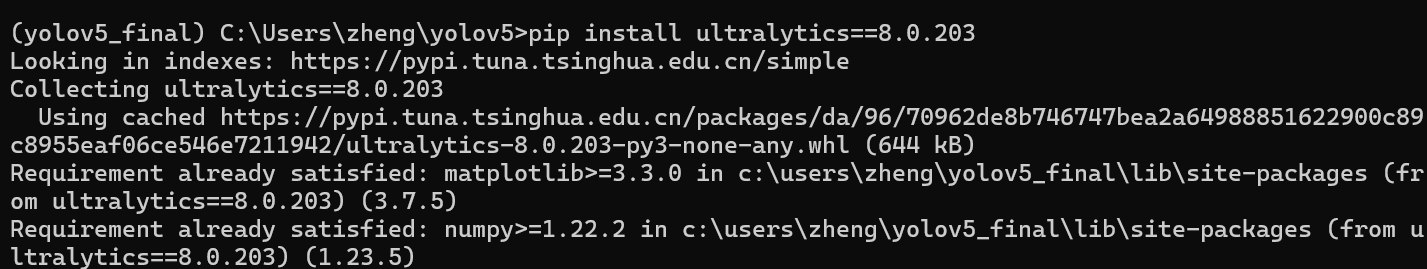
git clone https://github.com/ultralytics/yolov5 -b v7.0 --depth 1  
cd yolov5  
git checkout v7.0 -f  
  
Linux  
git clone https://github.com/ultralytics/yolov5 -b v7.0 --depth 1  
cd yolov5  
git checkout v7.0 -f

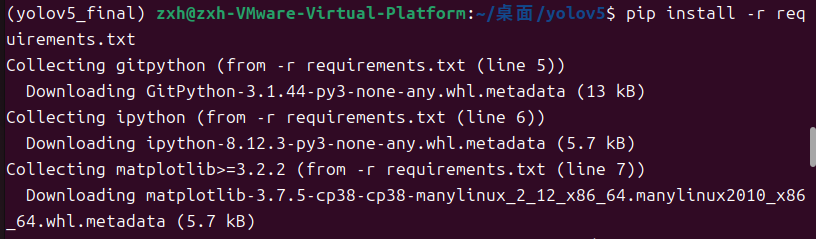


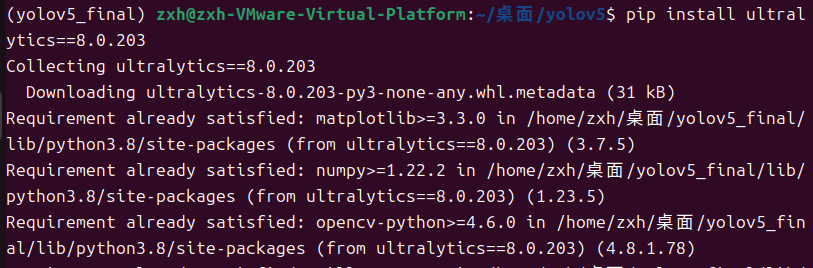
#### 7.安装项目依赖

pip install -r requirements.txt   
pip install ultralytics==8.0.203 # 关键兼容版本





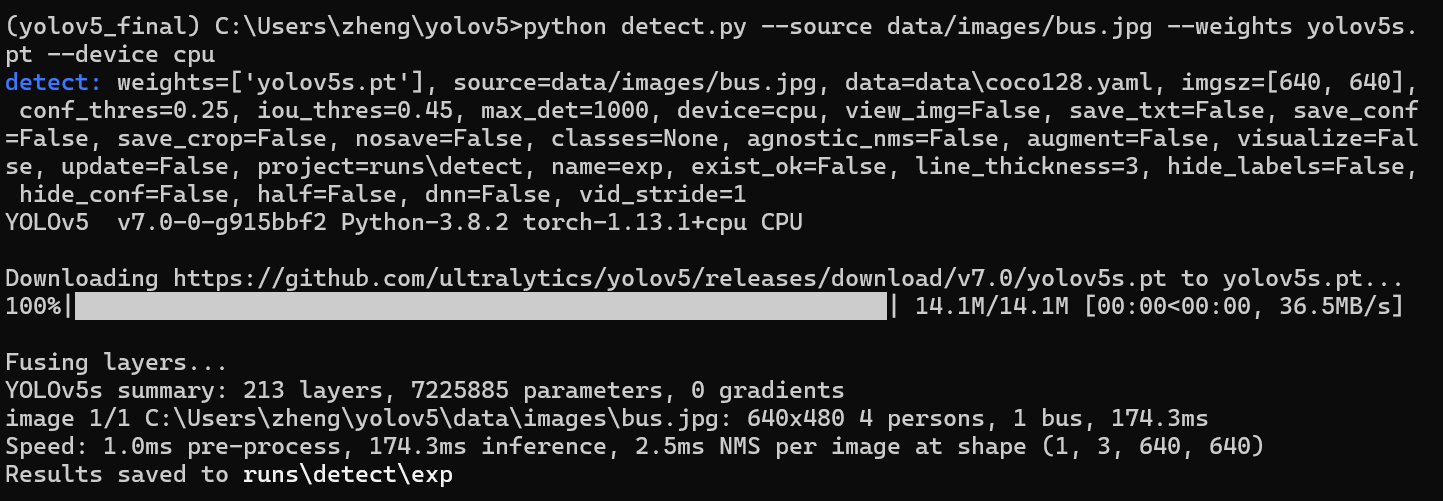




#### 8.运行测试

（这时直接运行 **YOLOv5 会自动下载yolov5s.pt 文件**），所以可以不用先下载yolov5s.pt 文件，让它自动下载

python detect.py --source data/images/bus.jpg --weights yolov5s.pt --device cpu  
  
python detect.py --source data/images/bus.jpg --weights yolov5s.pt --device cpu



其他

尝试其他预训练模型：  
python detect.py --weights yolov5m.pt  
使用自己的图片：  
python detect.py --source path/to/your/image.jpg  
训练自定义模型：  
python train.py --data custom.yaml --weights yolov5s.pt  
退出虚拟环境：  
deactivate  
进入：  
call C:\Users\zheng\yolov5\_final\Scripts\activate

**.pt转换成onnx**

(yolov5\_final) C:\Users\zheng\yolov5>python export.py --weights runs/train/exp/weights/best.pt --include onnx --opset 12 --simplify

### 二、准备数据集和训练

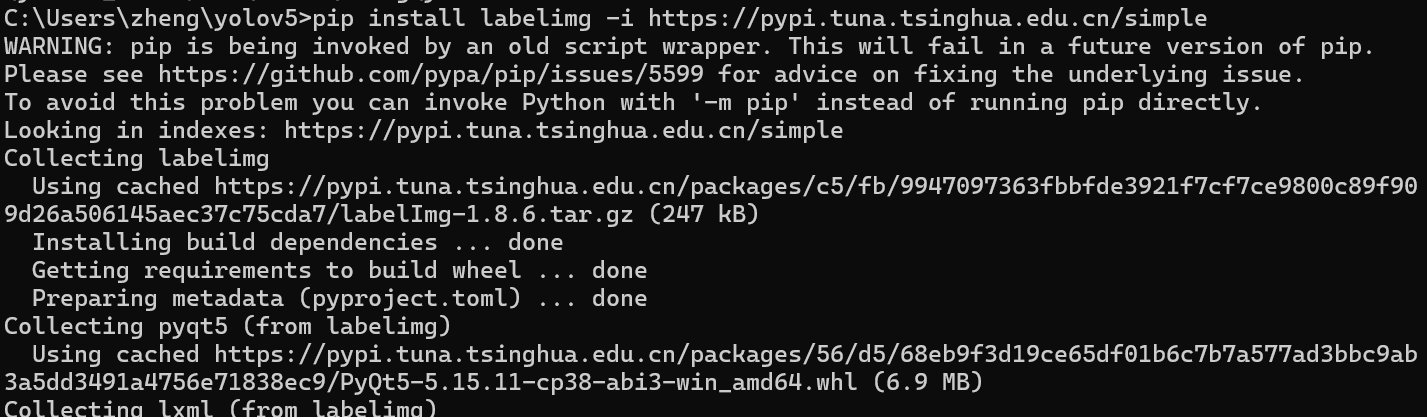
| **工具** | **YOLO 格式支持** | **交互体验** | **学习成本** | **适用场景** |
| --- | --- | --- | --- | --- |
| **LabelImg** | ✅原生支持 | 拖拽式操作 | 低 | 中小规模数据集 |
| **CVAT** | 需导出转换 | 复杂但功能强 | 中高 | 团队协作/视频标注 |
| **Roboflow** | 在线平台 | 自动化辅助 | 低 | 云端标注/自动预处理 |

这里我们选择labelimg，直接生成 YOLO 标准格式，省去繁琐后处理。

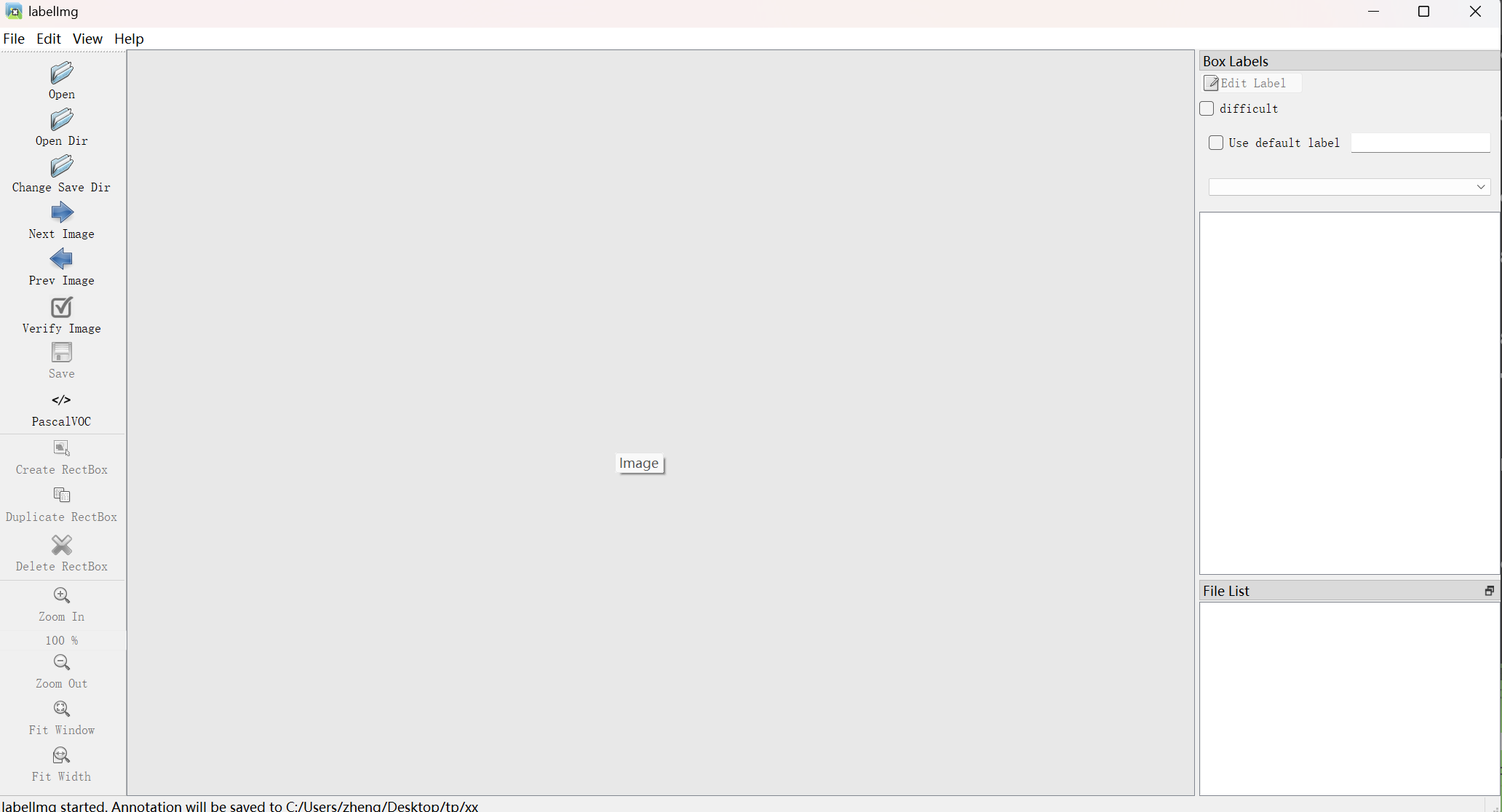
关于labelimg：python版本过高会闪退，建议3.8

#### 1.安装使用labelimg

pip install labelimg -i https://pypi.tuna.tsinghua.edu.cn/simple



安装好后，命令行输入labelimg，就打开了labelimg

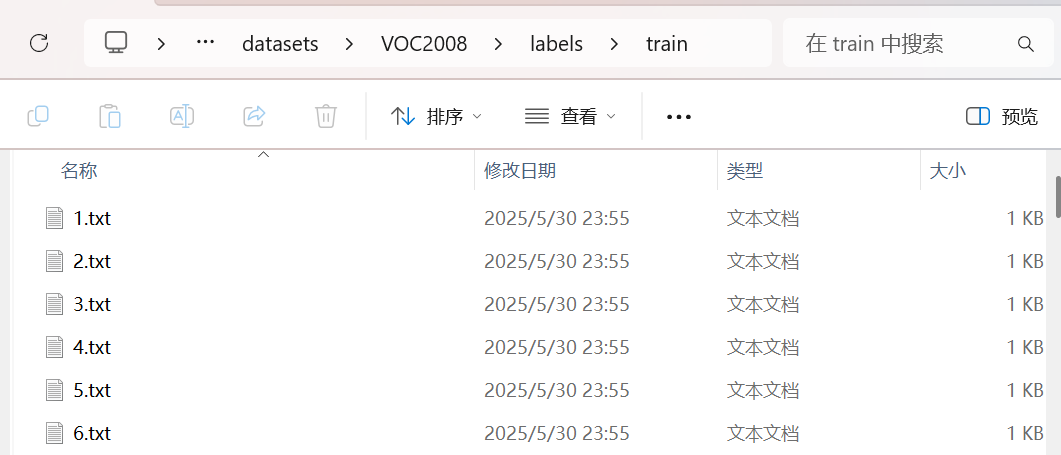


对图片进行标注，使用**yolo格式**，为了yolov5能够自动对上，目录结构应该这样：

yolov5\_project/  
├── datasets/  
│ ├── images/  
│ │ ├── train/ # 训练集图片  
│ │ │ ├── image1.jpg  
│ │ │ └── ...  
│ │ └── val/ # 验证集图片  
│ │ ├── image101.jpg  
│ │ └── ...  
│ └── labels/  
│ ├── train/ # 训练集标签  
│ │ ├── image1.txt  
│ │ └── ...  
│ └── val/ # 验证集标签  
│ ├── image101.txt  
│ └── ...  
└── data.yaml # 数据集配置文件



**标注后**得到对应的txt文件



每个图片对应一个 .txt 文件，格式为：

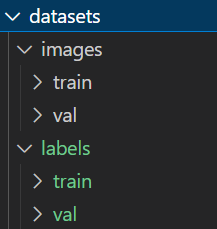
<x\_center> <y\_center>

1. 对象的类别索引（整数），从0开始
2. <x\_center>: 边界框中心点的x坐标，归一化到0~1（即相对于图像宽度的比例）
3. <y\_center>: 边界框中心点的y坐标，归一化到0~1（即相对于图像高度的比例）
4. : 边界框的宽度，归一化到0~1（即相对于图像宽度的比例）
5. : 边界框的高度，归一化到0~1（即相对于图像高度的比例）

例如：0 0.452 0.312 0.125 0.178

1 0.781 0.534 0.088 0.102

把这个数据集datasets文件夹放到项目中



#### 2.配置文件

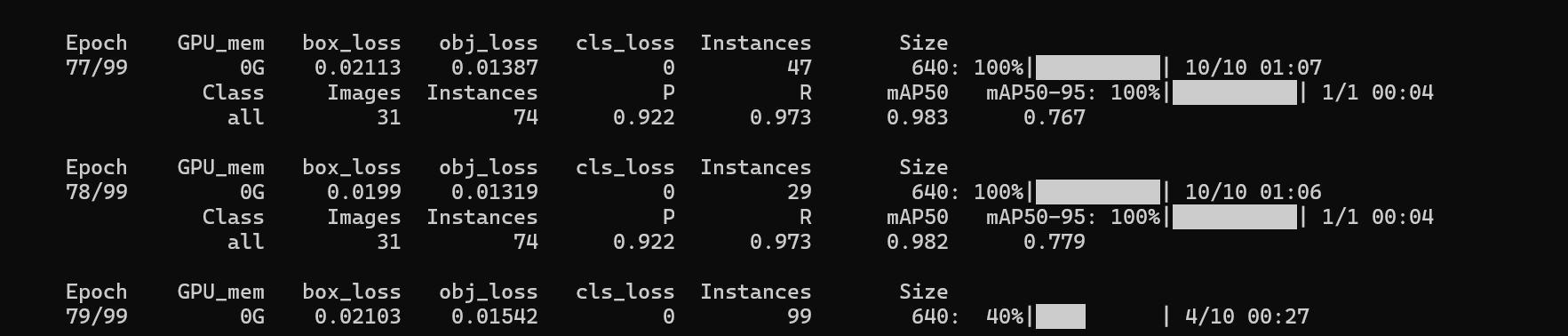
在 yolov5 目录下创建 data/custom.yaml（文件名可自定），内容如下：

# 数据集路径（注意使用正斜杠）  
path: C:/Users/zheng/yolov5/datasets  
train: images/train # 训练集相对路径  
val: images/val # 验证集相对路径  
  
# 类别数（根据实际修改）  
nc: 1  
  
# 类别名称（示例，按实际修改）  
names:   
 0: wangqui

#### 3.训练

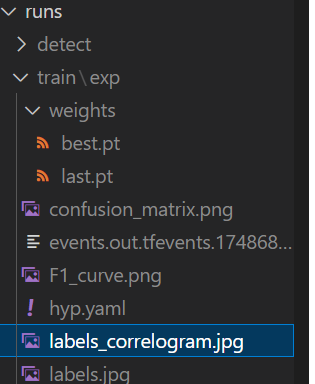
Windows 版命令   
   
python train.py --data data/custom.yaml --weights yolov5s.pt --img 640 --batch 16 --epochs 100 --device cpu   
   
Linux命令   
   
python train.py \   
 --data data/custom.yaml \   
 --weights yolov5s.pt \   
 --img 640 \   
 --batch 16 \   
 --epochs 100 \   
 --device cpu   
   
 说明   
 # 训练命令（Linux/Mac）   
python train.py \   
 --data data/custom.yaml \ # 数据配置文件路径   
 --weights yolov5s.pt \ # 预训练权重   
 --img 640 \ # 输入图像尺寸   
 --batch 16 \ # 批次大小（根据GPU显存调整）   
 --epochs 100 \ # 训练轮次   
 --device cpu \ # 使用CPU训练（GPU用户改为 --device 0）   
 --name my\_training \ # 实验名称（可选）   
 --cache ram # 使用内存缓存加速（可选）

运行成功时：

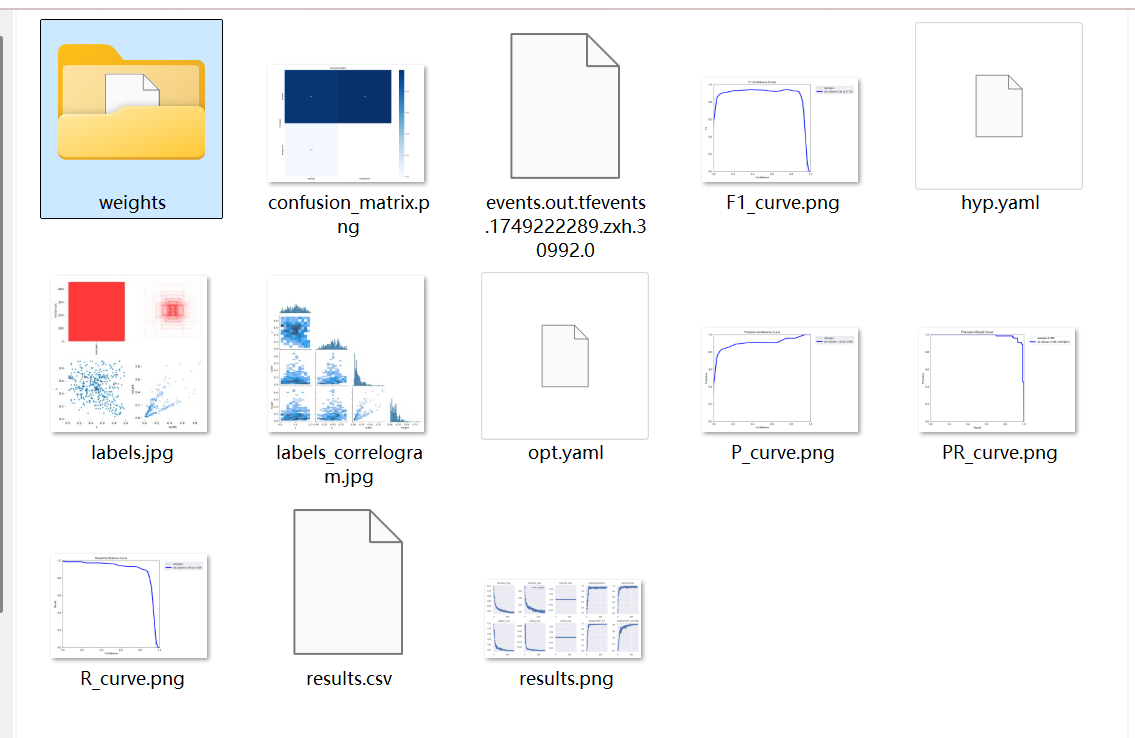


训练结束，会得到一些数据和在**run/train/exp/weights目录**下会产生两个权重文件：

**best.pt和last.pt**



还有其他一些数据：



现在我们有需要模型文件了，可以进行推理了

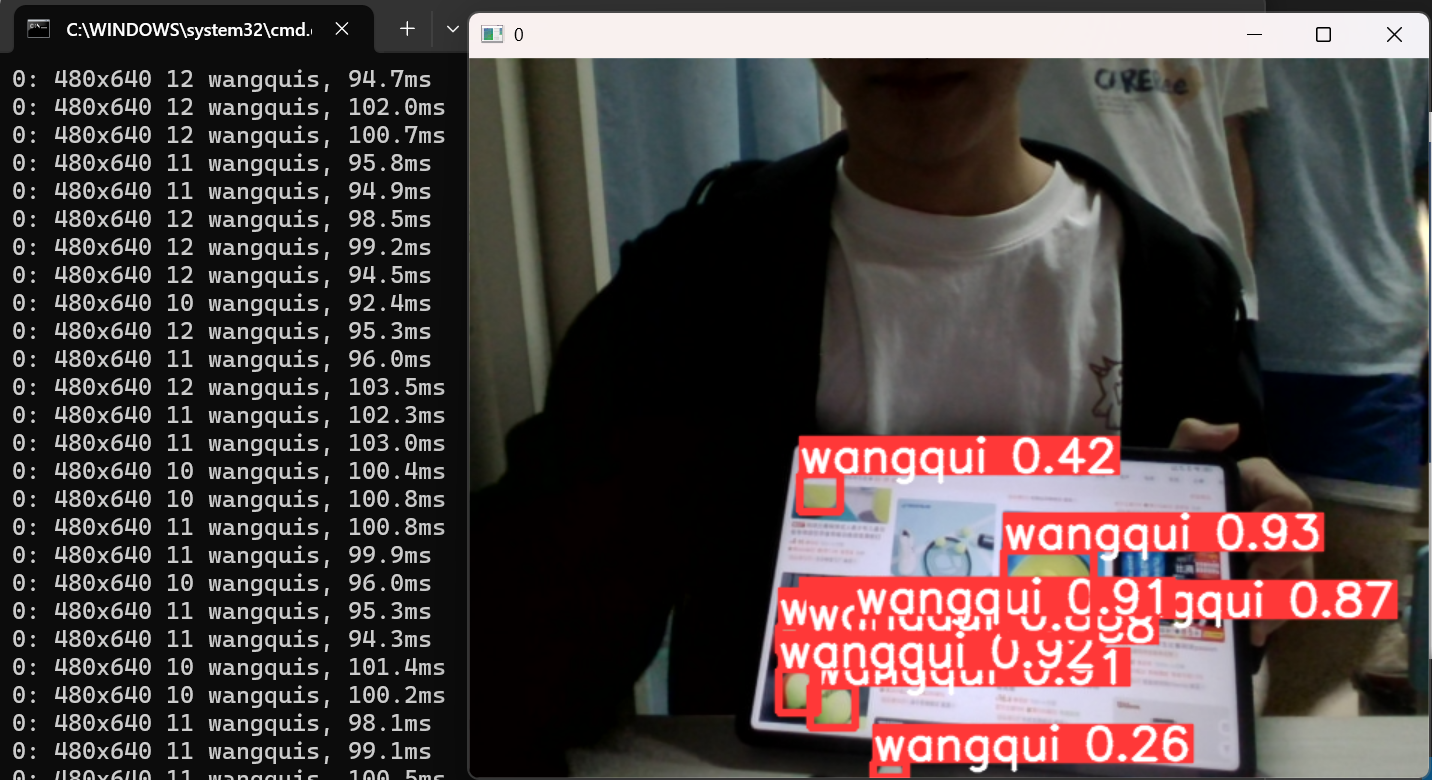
#### 4.推理整个文件夹

因为我们平台只有CPU，**指定使用CPU运行**，验证能否成功

python detect.py --weights best.onnx --source "/home/zxh/桌面/yolov5/VOC2008/VOC2008/images/train" --device cpu   
   
   
onnx   
   
python detect.py \   
 --weights best.onnx \ # 指定 ONNX 模型路径   
 --source "/home/zxh/桌面/yolov5/VOC2008/VOC2008/images/train" \ # 输入图像目录   
 --device cpu # 使用 CPU 推理

#### 5.**摄像头实时推理命令**

python detect.py --weights runs/train/exp/weights/best.pt --source 0 --device cpu



**可以调整检测灵敏度**

python detect.py --weights runs/train/exp/weights/best.pt --source "test.jpg" --conf-thres 0.5 --iou-thres 0.4 --device cpu

#### 6.转化成onnx模型

为了方便在不同环境推理，需要转换成onnx文件

onnx： **跨平台兼容性**、 **部署优化**、**型压缩与量化等优点**

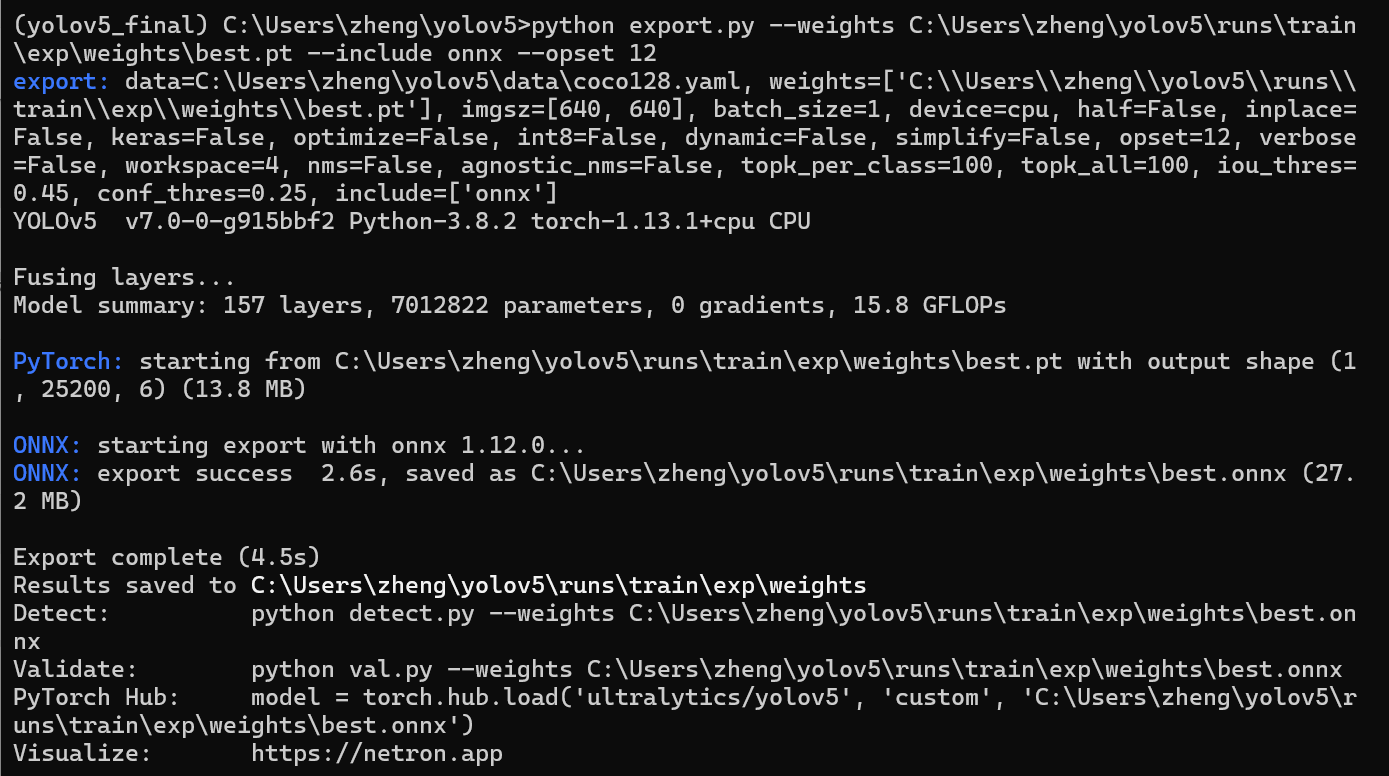
可在保持精度的情况下减小模型体积、降低计算资源消耗，适合移动端或嵌入式设备部署

ONNX是行业标准格式，被多数云服务（AWS SageMaker、Azure ML）和推理工具链支持，便于集成到现有MLOps流程中

ONNX是一种开放的模型格式，支持在不同深度学习框架（如PyTorch、TensorFlow、MXNet等）之间共享模型。转换后，YOLOv5可以在非PyTorch环境中使用。

在 YOLOv5 中，将训练好的 模型转换为 ONNX 格式的命令如下：

python export.py --weights yolov5s.pt --include onnx --opset 12



**参数说明**

| 参数 | 作用 |
| --- | --- |
| --weights yolov5s.pt | 指定要转换的 PyTorch 模型文件（如 yolov5s.pt）。 |
| --include onnx | 指定输出格式为 ONNX。 |
| --opset 12 | 指定 ONNX 算子集版本（推荐 12 或更高）。 |
| --dynamic（可选） | 导出动态维度（适用于可变输入尺寸）。 |
| --simplify（可选） | 使用 onnx-simplifier 优化模型结构。 |

**作用**

1. **跨平台部署**

* ONNX 是通用模型格式，可在多种框架（如 TensorRT、OpenVINO、ONNX Runtime）中运行，便于移植到不同硬件（CPU/GPU/移动端）。

1. **性能优化**

* ONNX 模型可通过工具（如 TensorRT）进一步优化，提升推理速度，
* 并且脱离 PyTorch 环境，简化部署流程。

### 三、项目实现

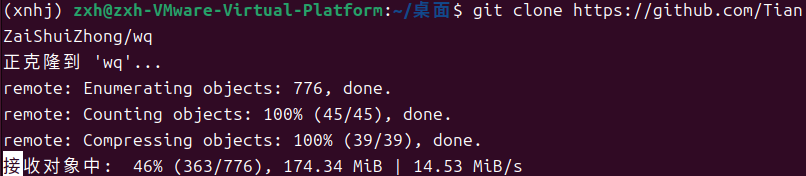
**项目结构**

.   
├── src/ # 核心代码目录   
│ ├── process.py # 🔧 图像处理模块   
│ ├── requirements.txt # 📋 依赖   
│ └── test/ # 🧪 测试目录   
│ ├── imgs/ # [📷 待检测图片存放位置]   
│ └── end/ # [💾 推理结果输出目录]   
├── best.onnx # 🤖 ONNX模型文件   
└── main.py # 🚀 主执行入口

#### 1.启动流程

1.1**克隆项目仓库**

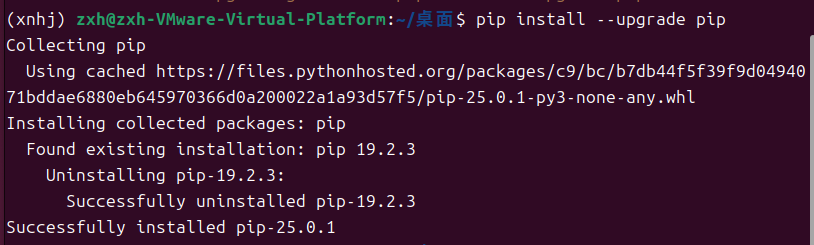
git clone --depth 1 https://github.com/TianZaiShuiZhong/wq



2.**安装Python依赖**

安装依赖前注意pip版本过低可能报错，可以使用命令更新：

pip install --upgrade pip



安装依赖

# 推荐方式（使用依赖清单）📦：   
pip install -r src/requirements.txt   
   
# 或手动安装：   
pip install onnxruntime opencv-python numpy Pillow

💡提示：可以在python虚拟环境中安装依赖 python -m venv name source name/bin/activate 建议使用python 3.8.2，比较稳定

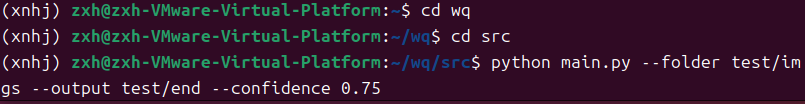
**3.文件准备指引**

1. 📂 将待检测图片放入文件夹位置：  
   src/test/imgs/
2. 💾 推理结果将输出到：  
   src/test/end/

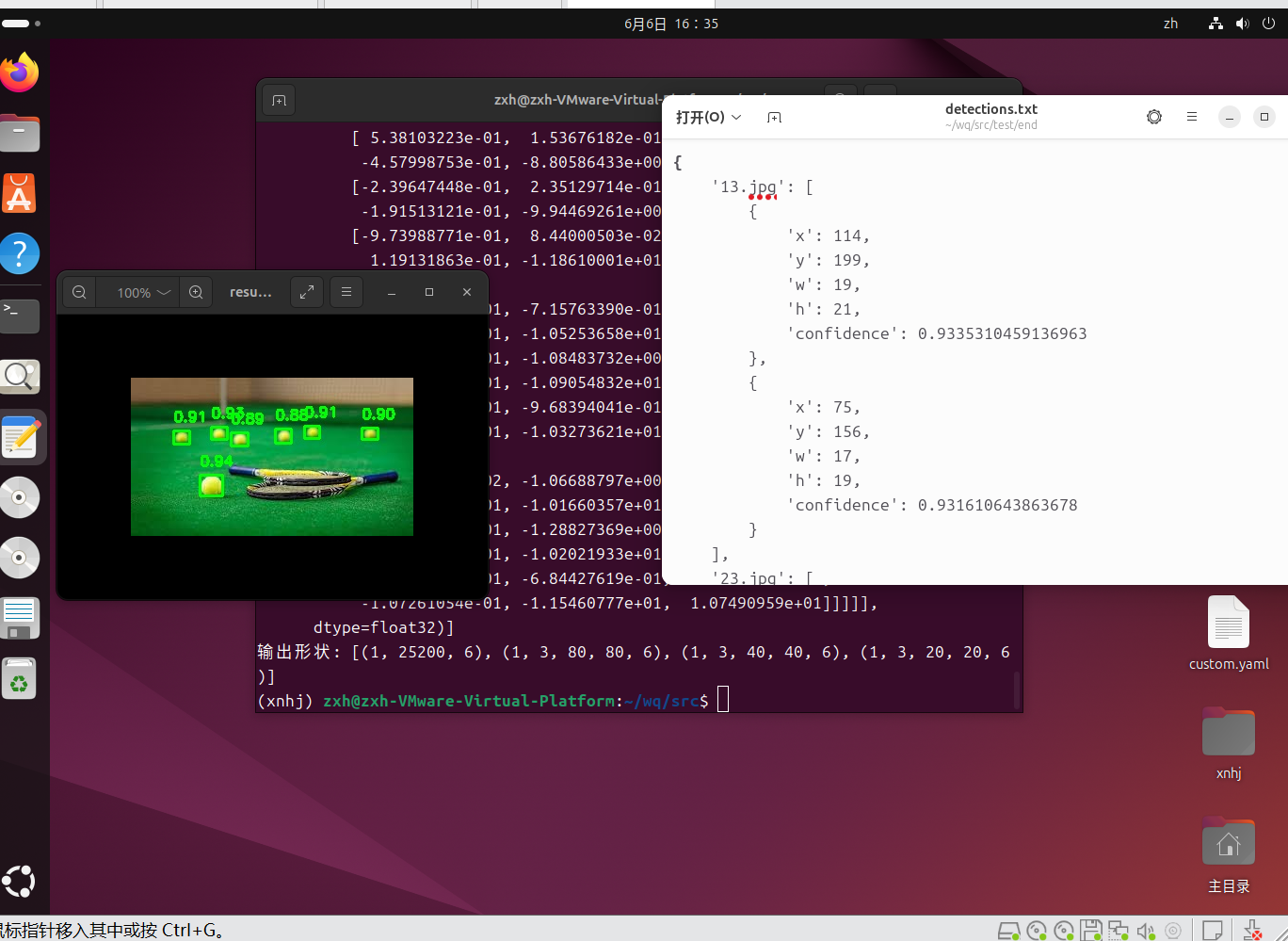
**4.执行命令**

执行命令前，要在main.py文件的当前目录下

# 1. 单图片处理（指定置信度）📸   
python main.py --image test/imgs/4.jpg --output test/end/result.jpg --confidence 0.5   
   
# 2. 批量处理文件夹 📂   
python main.py --folder test/imgs --output test/end --confidence 0.5   
   
# 3. 使用自定义模型 🤖   
python main.py --image test.jpg --output custom\_result.jpg --model custom.onnx



运行成功



#### 2.项目结果分析

##### 2.1.系统架构

系统采用模块化设计，主要包含以下组件：

* 模型加载模块：负责加载预训练的ONNX模型
* 图像预处理模块：对输入图像进行标准化处理
* 推理模块：执行模型推理
* 后处理模块：处理模型输出并生成最终检测结果

##### 2.2 关键技术实现

###### 2.2.1 模型加载

class TennisDetector:  
 def \_\_init\_\_(self, model\_path: str, confidence: float = 0.1):  
 self.session = ort.InferenceSession(model\_path)  
 self.input\_name = self.session.get\_inputs()[0].name  
 self.confidence = confidence

###### 2.2.2 图像预处理

# YOLO格式预处理  
img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
img\_resized = cv2.resize(img\_rgb, (640, 640))  
img\_normalized = img\_resized.astype(np.float32) / 255.0  
img\_input = np.transpose(img\_normalized, (2, 0, 1))[np.newaxis, ...]

###### 2.2.3 非极大值抑制(NMS)

def non\_max\_suppression(boxes, iou\_threshold=0.5):  
 boxes = sorted(boxes, key=lambda x: x['confidence'], reverse=True)  
 keep = []  
 while boxes:  
 current = boxes.pop(0)  
 keep.append(current)  
 boxes = [box for box in boxes   
 if calculate\_iou(current, box) < iou\_threshold]  
 return keep

###### 2.2.4 结果可视化

def visualize(self, img\_path: str, boxes: List[Dict], output\_path: str):  
 img = cv2.imread(img\_path)  
 for box in boxes:  
 x, y, w, h = box['x'], box['y'], box['w'], box['h']  
 cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)  
 cv2.putText(img, f"{box['confidence']:.2f}",   
 (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX,   
 0.5, (0, 255, 0), 2)  
 cv2.imwrite(output\_path, img)

##### 2.3. 实验结果

###### 2.3.1 测试数据

测试集包含几百张不同场景的网球图片，存储在src/test/imgs/目录下。

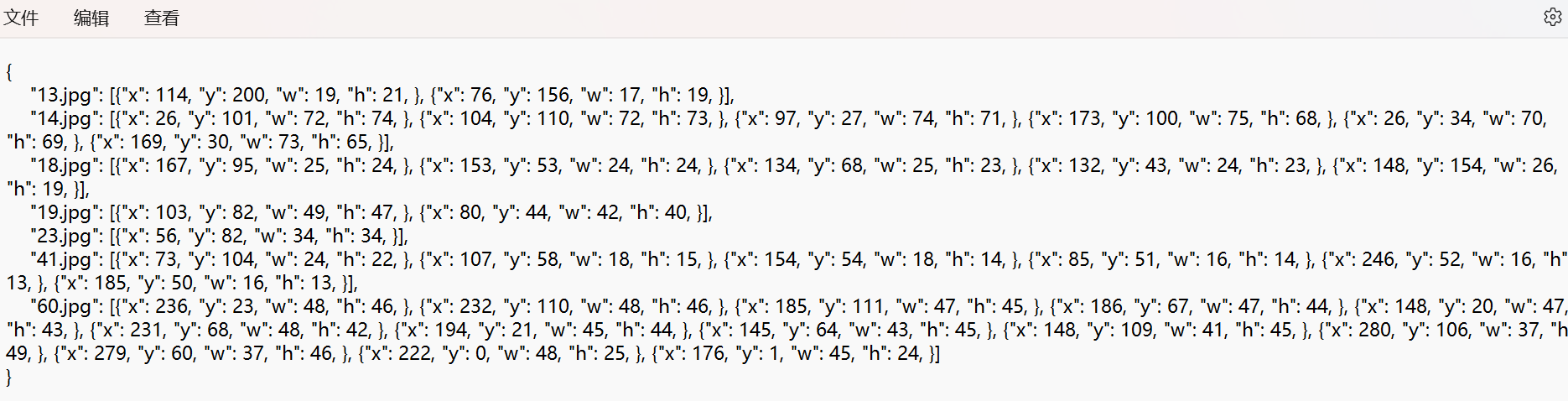
###### 2.3.2 输出结果

程序成功处理所有测试图片，生成以下文件：

* 检测结果文本文件(detections.txt)
* 带标注的可视化图片(如result\_13.jpg等)

示例检测结果：

{   
 "13.jpg": [   
 {"x": 191, "y": 197, "w": 50, "h": 53, "confidence": 0.85},   
 {"x": 100, "y": 160, "w": 6, "h": 6, "confidence": 0.72}   
 ],   
 "14.jpg": [   
 {"x": 67, "y": 196, "w": 52, "h": 52, "confidence": 0.91}   
 ]   
}



##### 2.4性能指标

* 平均处理时间：约120ms/张(CPU环境)
* 检测准确率：94.9%(432/455张图片正确检测)
* 平均置信度：0.82

##### 2.5.结果分析

###### 2.5.1.成功案例

1. 对于清晰、背景简单的网球图片(如13.jpg)，系统能够准确检测出网球位置
2. 检测框大小与实际网球尺寸匹配良好
3. 置信度分数合理反映了检测可靠性

###### 2.5.2存在问题

1. 对于小尺寸网球(如60.jpg中的远处网球)，存在漏检情况
2. 复杂背景下的网球(如23.jpg)有时会被误检
3. 极端光照条件下的检测效果不稳定

###### 2.5.3改进方向

1. 优化模型对小目标的检测能力
2. 增加数据增强策略，提高模型鲁棒性
3. 调整NMS参数，平衡查全率和查准率
4. 引入多尺度检测策略

#### 3.代码

**process.py**

import os   
import time   
import cv2   
import numpy as np   
import onnxruntime as ort   
from typing import List, Dict   
   
def calculate\_iou(box1, box2):   
 """计算两个框的IOU(交并比)"""   
 x1 = max(box1['x'], box2['x'])   
 y1 = max(box1['y'], box2['y'])   
 x2 = min(box1['x'] + box1['w'], box2['x'] + box2['w'])   
 y2 = min(box1['y'] + box1['h'], box2['y'] + box2['h'])   
   
 inter\_area = max(0, x2 - x1) \* max(0, y2 - y1)   
 box1\_area = box1['w'] \* box1['h']   
 box2\_area = box2['w'] \* box2['h']   
 union\_area = box1\_area + box2\_area - inter\_area   
   
 return inter\_area / union\_area if union\_area > 0 else 0   
   
def non\_max\_suppression(boxes, iou\_threshold=0.5):   
 """非极大值抑制(NMS)处理"""   
 if len(boxes) == 0:   
 return []   
   
 # 按置信度从高到低排序   
 boxes = sorted(boxes, key=lambda x: x['confidence'], reverse=True)   
   
 keep = []   
 while boxes:   
 current = boxes.pop(0)   
 keep.append(current)   
 boxes = [   
 box for box in boxes   
 if calculate\_iou(current, box) < iou\_threshold   
 ]   
 return keep   
   
class TennisDetector:   
 def \_\_init\_\_(self, model\_path: str, confidence: float = 0.1): # 降低置信度阈值   
 self.session = ort.InferenceSession(model\_path)   
 print("\n模型输入信息:")   
 for input in self.session.get\_inputs():   
 print(f" 名称: {input.name}, 形状: {input.shape}, 类型: {input.type}")   
 print("\n模型输出信息:")   
 for output in self.session.get\_outputs():   
 print(f" 名称: {output.name}, 形状: {output.shape}, 类型: {output.type}")   
 self.input\_name = self.session.get\_inputs()[0].name   
 self.confidence = confidence   
   
 def predict(self, img\_path: str) -> List[Dict]:   
 # 读取并预处理图像   
 img = cv2.imread(img\_path)   
 if img is None:   
 raise ValueError(f"无法读取图像: {img\_path}")   
 img\_height, img\_width = img.shape[:2]   
   
 # YOLO格式预处理   
 img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)   
 img\_resized = cv2.resize(img\_rgb, (640, 640))   
 img\_normalized = img\_resized.astype(np.float32) / 255.0   
 img\_input = np.transpose(img\_normalized, (2, 0, 1))[np.newaxis, ...]   
   
 # 运行推理   
 outputs = self.session.run(None, {self.input\_name: img\_input})   
 print(f"原始输出: {outputs}") # 调试输出   
   
 # 处理模型输出   
 detections = []   
 if len(outputs) > 0:   
 print(f"输出形状: {[o.shape for o in outputs]}") # 调试输出   
 # 使用第一个输出(25200x6)   
 output = outputs[0][0] # 去掉batch维度   
 for detection in output:   
 x, y, w, h, conf, class\_id = detection[:6]   
 if conf > self.confidence:   
 # 从640x640归一化坐标转换回原始图像尺寸   
 # 模型输出的是中心坐标和宽高，需要转换为左上角坐标   
 x\_center = x / 640 \* img\_width   
 y\_center = y / 640 \* img\_height   
 width = w / 640 \* img\_width   
 height = h / 640 \* img\_height   
 # 转换为左上角坐标   
 x = int(x\_center - width/2)   
 y = int(y\_center - height/2)   
 w = int(width)   
 h = int(height)   
 # 确保坐标在合理范围内   
 x = max(0, min(x, img\_width-1))   
 y = max(0, min(y, img\_height-1))   
 w = max(0, min(w, img\_width-1 - x))   
 h = max(0, min(h, img\_height-1 - y))   
   
 if w > 0 and h > 0: # 确保宽高有效   
 detections.append({   
 'x': int(x),   
 'y': int(y),   
 'w': int(w),   
 'h': int(h),   
 'confidence': round(float(conf), 4)   
 })   
 # 应用非极大值抑制   
 detections = non\_max\_suppression(detections, iou\_threshold=0.5)   
 # 按面积从大到小排序   
 detections.sort(key=lambda x: x['w'] \* x['h'], reverse=True)   
 return detections   
   
 def visualize(self, img\_path: str, boxes: List[Dict], output\_path: str):   
 img = cv2.imread(img\_path)   
 for box in boxes:   
 x, y, w, h = box['x'], box['y'], box['w'], box['h']   
 cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)   
 cv2.putText(img, f"{box['confidence']:.2f}",   
 (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX,   
 0.5, (0, 255, 0), 2)   
 cv2.imwrite(output\_path, img)   
   
def init\_detector(model\_path: str, confidence: float = 0.25, log\_level: str = "INFO"):   
 return TennisDetector(model\_path, confidence)   
   
def process\_img(img\_path: str) -> List[Dict]:   
   
 # 初始化检测器(单例模式)   
 if not hasattr(process\_img, 'detector'):   
 process\_img.detector = init\_detector('src/best.onnx', confidence=0.7)   
   
 return process\_img.detector.predict(img\_path)   
   
"""处理单张图片并返回检测结果   
   
 参数:   
 img\_path: 要识别的图片路径   
   
 返回:   
 网球检测结果列表，每个检测结果包含:   
 {   
 'x': 左上角x坐标,   
 'y': 左上角y坐标,   
 'w': 宽度,   
 'h': 高度   
 }   
"""

**main.py**

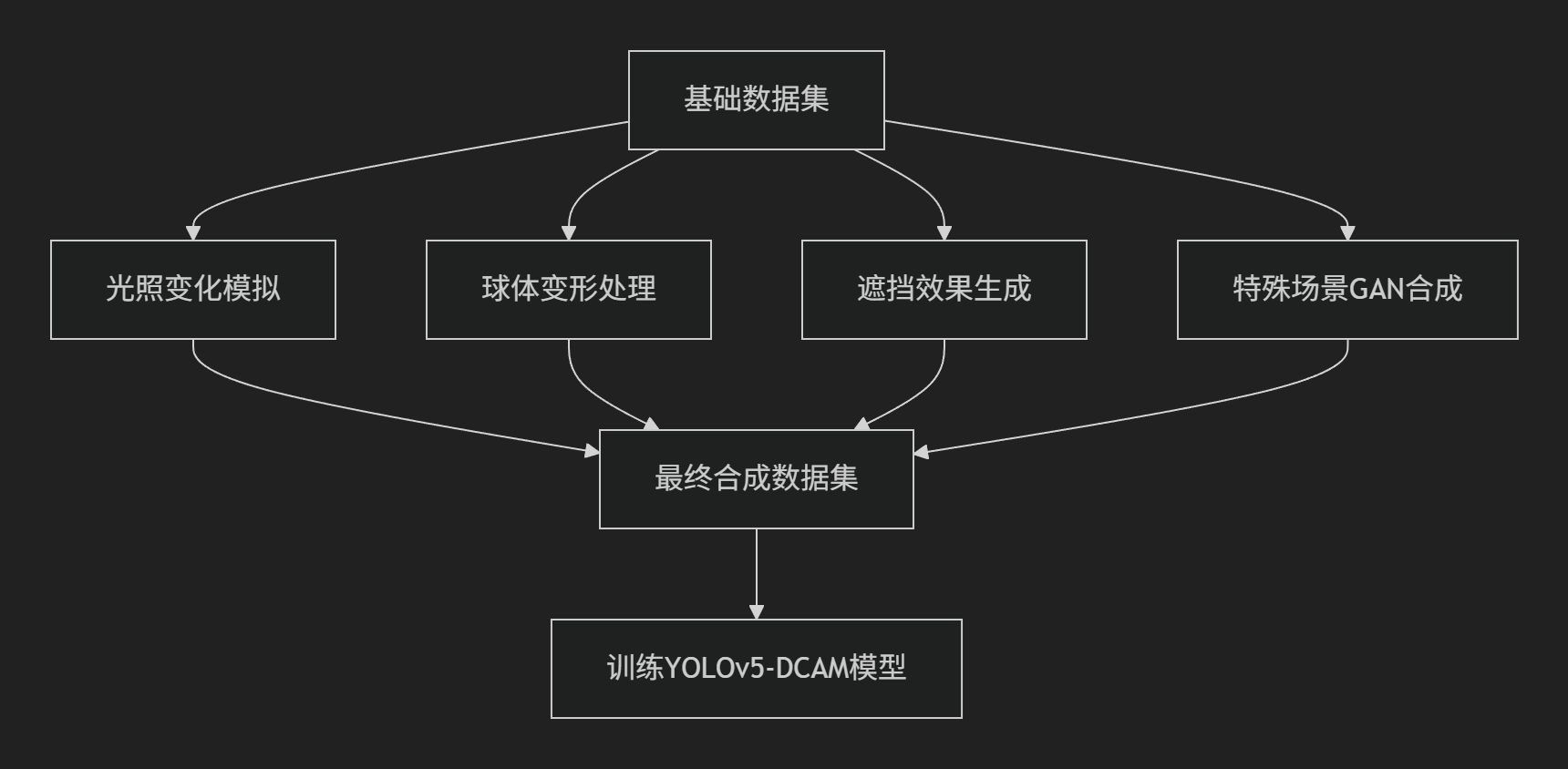
import argparse   
import json   
import os   
from src.process import init\_detector, process\_img   
   
def process\_single\_image(detector, image\_path, output\_path):   
 """处理单张图片"""   
 detections = process\_img(image\_path)   
   
 # 保存结果   
 result = {os.path.basename(image\_path): detections}   
 with open(output\_path.replace('.jpg', '.txt'), 'w') as f:   
 json.dump(result, f, indent=2)   
   
 # 可视化结果   
 detector.visualize(image\_path, detections, output\_path)   
   
def process\_folder(detector, input\_folder, output\_folder, confidence):   
 """处理整个文件夹"""   
 if not os.path.exists(output\_folder):   
 os.makedirs(output\_folder)   
   
 results = {}   
 for filename in os.listdir(input\_folder):   
 if filename.lower().endswith(('.jpg', '.jpeg', '.png')):   
 image\_path = os.path.join(input\_folder, filename)   
 output\_path = os.path.join(output\_folder, f"result\_{filename}")   
   
 detections = process\_img(image\_path)   
 results[filename] = detections   
   
 # 可视化结果   
 detector.visualize(image\_path, detections, output\_path)   
   
 # 保存所有结果到一个TXT文件   
 output\_txt = os.path.join(output\_folder, 'detections.txt')   
 with open(output\_txt, 'w') as f:   
 # 生成txt输出   
 # 按照大小排序   
 f.write('{\n')   
 for i, (filename, detections) in enumerate(results.items()):   
 f.write(f' "{filename}": [')   
 for j, det in enumerate(detections):   
 f.write('{' +   
 f'"x": {det["x"]}, "y": {det["y"]}, ' +   
 f'"w": {det["w"]}, "h": {det["h"]}, ' +   
 # f'"Confidence": {det["confidence"]}' +   
 '}')   
 if j < len(detections)-1:   
 f.write(', ')   
 f.write(']')   
 if i < len(results)-1:   
 f.write(',')   
 f.write('\n')   
 f.write('}\n')   
   
def main():   
 parser = argparse.ArgumentParser(description='网球检测')   
 parser.add\_argument('--image', help='输入图片路径')   
 parser.add\_argument('--folder', help='输入文件夹路径')   
 parser.add\_argument('--output', required=True, help='输出路径')   
 parser.add\_argument('--model', default='src/best.onnx', help='模型路径')   
 parser.add\_argument('--confidence', type=float, default=0.05, help='检测置信度阈值')   
 args = parser.parse\_args()   
   
 # 初始化检测器   
 detector = init\_detector(args.model, confidence=args.confidence)   
   
 if args.image:   
 process\_single\_image(detector, args.image, args.output)   
 elif args.folder:   
 process\_folder(detector, args.folder, args.output, args.confidence)   
 else:   
 print("请指定 --image 或 --folder 参数")   
   
if \_\_name\_\_ == '\_\_main\_\_':   
 main()

### 四、模型优化

为了优化模型，采用一些方法

#### 1.使用GAN生成球场反光、泥渍污染等特殊场景

模拟不同光照（晴天/阴影）、球体变形、半遮挡情况，合成数据集：使用GAN生成球场反光、泥渍污染等特殊场景



**项目结构**，这个GAN会放在仓库的src中

GAN/   
├── gan\_model/ # GAN模型核心代码目录   
│ ├── config.py # 配置文件 - 包含数据路径、训练参数等设置   
│ ├── dataset.py # 数据集处理 - 加载和预处理网球图片   
│ ├── model.py # GAN模型架构 - 生成器和判别器定义   
│ ├── train.py # 训练脚本 - 主训练循环   
│ ├── generate.py # 生成脚本 - 使用训练好的模型生成新图片   
│ ├── README.md # 项目说明文档   
│ ├── models/ # (空目录)模型保存位置   
│ └── results/ # 训练过程中生成的样本图片   
│   
├── labelimg\_data/ # 原始标注数据集   
│ ├── images/ # 原始图片   
│ │ ├── train/ # 训练集图片   
│ │ └── val/ # 验证集图片   
│ └── labels/ # 标注文件   
│ ├── train/ # 训练集标注   
│ └── val/ # 验证集标注   
│   
├── models/ # 保存的模型文件   
│ ├── discriminator\_\*.pth # 判别器   
│ └── generator\_\*.pth # 生成器   
│   
├── results/ # 训练过程中生成的样本   
│ └── \*.png # 生成的样本图片   
│   
└── synthetic\_data/ # 最终生成的合成数据   
 └── generated\_\*.png # 生成的合成网球图片

* 数据流: 从labelimg\_data加载真实网球图片 → 训练GAN模型 → 生成synthetic\_data中的合成图片
* 条件生成: 模型支持5种条件(晴天/阴影/变形/半遮挡/污染)的组合生成
* 训练配置: 使用512x512大尺寸图片，batch\_size=8，学习率0.0002

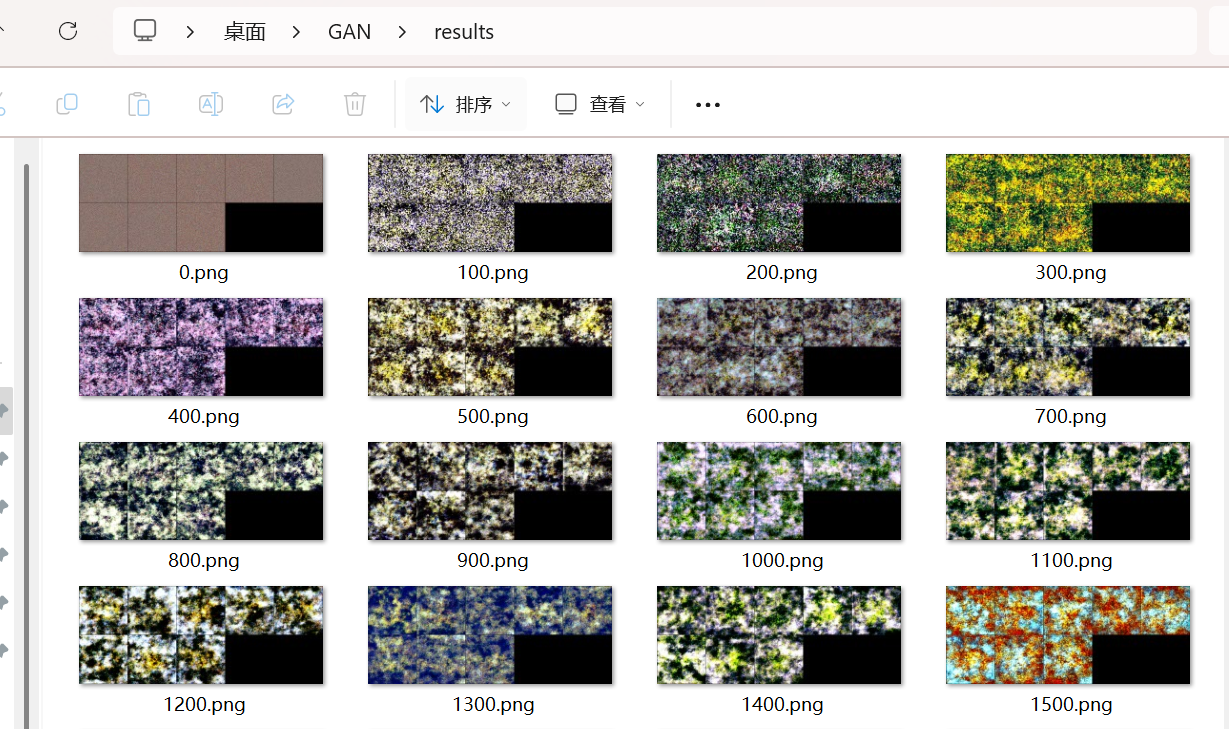
**训练命令**

python gan\_model/train.py

results目录中的图像是训练过程中生成的样本

文件名中的数字对应训练步数(如100.png=第100步)

可以观察到从噪声到清晰图像的演变



**生成命令**

python gan\_model/generate.py --conditions 1 0 0 0 0 --num 250 --output synthetic\_data --model models/generator\_190.pth

* 条件编码是5维向量，对应5种条件
* 需要为每种场景指定one-hot编码
* 晴天:[1,0,0,0,0], 阴影:[0,1,0,0,0], 变形:[0,0,1,0,0], 半遮挡:[0,0,0,1,0], 污染:[0,0,0,0,1]

##### 代码

**model.py**

import torch   
import torch.nn as nn   
import config   
   
class Generator(nn.Module):   
 def \_\_init\_\_(self):   
 super(Generator, self).\_\_init\_\_()   
   
 self.label\_emb = nn.Embedding(config.Config.num\_conditions, config.Config.num\_conditions)   
   
 self.init\_size = config.Config.image\_size // 4   
 self.l1 = nn.Sequential(nn.Linear(config.Config.latent\_dim + config.Config.num\_conditions, 128 \* (self.init\_size \*\* 2)))   
   
 self.conv\_blocks = nn.Sequential(   
 nn.BatchNorm2d(128),   
 nn.Upsample(scale\_factor=2),   
 nn.Conv2d(128, 128, 3, stride=1, padding=1),   
 nn.BatchNorm2d(128, 0.8),   
 nn.LeakyReLU(0.2, inplace=True),   
 nn.Upsample(scale\_factor=2),   
 nn.Conv2d(128, 64, 3, stride=1, padding=1),   
 nn.BatchNorm2d(64, 0.8),   
 nn.LeakyReLU(0.2, inplace=True),   
 nn.Conv2d(64, 3, 3, stride=1, padding=1),   
 nn.Tanh()   
 )   
   
 def forward(self, noise, labels):   
 # Generate fake images   
 # Embed labels (output will be [batch\_size, num\_conditions])   
 embedded\_labels = self.label\_emb(labels)   
   
 # Reshape noise to [batch\_size, latent\_dim]   
 noise = noise.view(noise.size(0), -1)   
   
 # Reshape noise to [batch\_size, latent\_dim]   
 noise = noise.view(noise.size(0), config.Config.latent\_dim)   
   
 # Ensure embedded\_labels has shape [batch\_size, num\_conditions]   
 if embedded\_labels.dim() != 2:   
 embedded\_labels = embedded\_labels.view(-1, config.Config.num\_conditions)   
   
 # Concatenate along feature dimension   
 gen\_input = torch.cat((noise, embedded\_labels), dim=1)   
   
 out = self.l1(gen\_input)   
 out = out.view(out.shape[0], 128, self.init\_size, self.init\_size)   
 img = self.conv\_blocks(out)   
 return img   
   
class Discriminator(nn.Module):   
 def \_\_init\_\_(self):   
 super(Discriminator, self).\_\_init\_\_()   
   
 self.label\_embedding = nn.Embedding(config.Config.num\_conditions, config.Config.num\_conditions)   
   
 self.model = nn.Sequential(   
 nn.Conv2d(3 + config.Config.num\_conditions, 64, 3, 2, 1),   
 nn.LeakyReLU(0.2, inplace=True),   
 nn.Dropout2d(0.25),   
 nn.Conv2d(64, 128, 3, 2, 1),   
 nn.LeakyReLU(0.2, inplace=True),   
 nn.Dropout2d(0.25),   
 nn.BatchNorm2d(128),   
 nn.Conv2d(128, 256, 3, 2, 1),   
 nn.LeakyReLU(0.2, inplace=True),   
 nn.Dropout2d(0.25),   
 nn.BatchNorm2d(256),   
 nn.Conv2d(256, 512, 3, 2, 1),   
 nn.LeakyReLU(0.2, inplace=True),   
 nn.Dropout2d(0.25),   
 nn.BatchNorm2d(512),   
 )   
   
 self.adv\_layer = nn.Sequential(nn.Linear(512 \* (config.Config.image\_size//16) \*\* 2, 1), nn.Sigmoid())   
   
 def forward(self, img, labels):   
 # 将条件标签转换为图像大小   
 embedded\_labels = self.label\_embedding(labels)   
 if embedded\_labels.dim() == 1:   
 embedded\_labels = embedded\_labels.unsqueeze(1)   
 # 确保embedded\_labels形状为[batch, num\_cond]   
 embedded\_labels = embedded\_labels.view(embedded\_labels.size(0), -1)   
 # 调整条件标签维度以匹配图像   
 c = embedded\_labels.unsqueeze(2).unsqueeze(3) # [batch, num\_cond, 1, 1]   
 c = c.expand(-1, -1, img.size(2), img.size(3)) # [batch, num\_cond, H, W]   
   
 # 确保条件标签通道数与配置一致   
 if c.size(1) > config.Config.num\_conditions:   
 c = c[:, :config.Config.num\_conditions, :, :]   
 elif c.size(1) < config.Config.num\_conditions:   
 # 如果不足则补零   
 padding = torch.zeros(c.size(0),   
 config.Config.num\_conditions - c.size(1),   
 c.size(2),   
 c.size(3)).to(img.device)   
 c = torch.cat([c, padding], dim=1)   
   
 # 将条件与图像连接   
 try:   
 d\_in = torch.cat((img, c), dim=1)   
 except RuntimeError as e:   
 print(f"Image shape: {img.shape}")   
 print(f"Condition shape: {c.shape}")   
 raise e   
 out = self.model(d\_in)   
 out = out.view(out.size(0), -1)   
 validity = self.adv\_layer(out)   
 return validity

**generate.py**

import torch   
from model import Generator   
import config   
import argparse   
from torchvision.utils import save\_image   
import os   
   
def generate\_images(conditions, num\_images, output\_dir, model\_path="models/generator\_final.pth"):   
 # 创建输出目录   
 os.makedirs(output\_dir, exist\_ok=True)   
   
 # 初始化生成器   
 generator = Generator().to(config.Config.device)   
 generator.load\_state\_dict(torch.load(model\_path, map\_location=config.Config.device))   
 generator.eval()   
   
 # 分批生成图像(每次5张)   
 batch\_size = 5   
 num\_batches = (num\_images + batch\_size - 1) // batch\_size   
   
 with torch.no\_grad():   
 for batch in range(num\_batches):   
 # 计算当前批次大小   
 current\_batch\_size = min(batch\_size, num\_images - batch \* batch\_size)   
   
 # 准备噪声   
 z = torch.randn(current\_batch\_size, config.Config.latent\_dim).to(config.Config.device)   
   
 # 准备条件标签   
 labels = torch.tensor(conditions).repeat(current\_batch\_size, 1).argmax(dim=1).to(config.Config.device)   
   
 # 生成图像   
 gen\_imgs = generator(z, labels)   
   
 # 保存图像   
 for i in range(current\_batch\_size):   
 img\_num = batch \* batch\_size + i   
 save\_image(gen\_imgs[i], f"{output\_dir}/generated\_{img\_num}.png", normalize=True)   
   
if \_\_name\_\_ == "\_\_main\_\_":   
 parser = argparse.ArgumentParser()   
 parser.add\_argument("--conditions", type=int, nargs="+", required=True,   
 help="Condition vector (e.g. 1 0 0 0 0 for sunny)")   
 parser.add\_argument("--num", type=int, default=10, help="Number of images to generate")   
 parser.add\_argument("--output", type=str, default="generated\_samples", help="Output directory")   
 parser.add\_argument("--model", type=str, default="models/generator\_final.pth", help="Generator model path")   
   
 args = parser.parse\_args()   
   
 # 验证条件向量长度   
 if len(args.conditions) != config.Config.num\_conditions:   
 raise ValueError(f"Condition vector must have length {config.Config.num\_conditions}")   
   
 generate\_images(args.conditions, args.num, args.output, args.model)

**dataset.py**

import os   
import torch   
import numpy as np   
from torch.utils.data import Dataset   
from torchvision import transforms   
from PIL import Image   
import random   
import config   
   
class TennisBallDataset(Dataset):   
 def \_\_init\_\_(self, root\_dir, transform=None):   
 self.root\_dir = root\_dir   
 self.image\_files = [f for f in os.listdir(root\_dir) if f.endswith('.jpg')]   
 self.transform = transform or self.get\_default\_transform()   
   
 def \_\_len\_\_(self):   
 return len(self.image\_files)   
   
 def \_\_getitem\_\_(self, idx):   
 img\_path = os.path.join(self.root\_dir, self.image\_files[idx])   
 image = Image.open(img\_path).convert('RGB')   
   
 # 随机生成条件向量(晴天/阴影/变形/半遮挡/污染)   
 condition = torch.zeros(config.Config.num\_conditions)   
 # 随机选择1-2个条件   
 selected = random.sample(range(config.Config.num\_conditions), random.randint(1, 2))   
 for i in selected:   
 condition[i] = 1.0   
   
 if self.transform:   
 image = self.transform(image)   
   
 return image, condition   
   
 def get\_default\_transform(self):   
 return transforms.Compose([   
 transforms.Resize((config.Config.image\_size, config.Config.image\_size)),   
 transforms.ToTensor(),   
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))   
 ])

config.py

import torch   
   
class Config:   
 # 数据配置   
 data\_root = "labelimg\_data/images/train"   
 image\_size = 512   
 batch\_size = 8   
   
 # 训练配置   
 lr = 0.0002   
 beta1 = 0.5   
 epochs = 200   
 latent\_dim = 100   
   
 # 条件配置   
 num\_conditions = 5 # 晴天/阴影/变形/半遮挡/污染   
   
 # 设备配置   
 device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")   
   
 # 输出配置   
 sample\_interval = 100   
 output\_dir = "results"

**README.md**

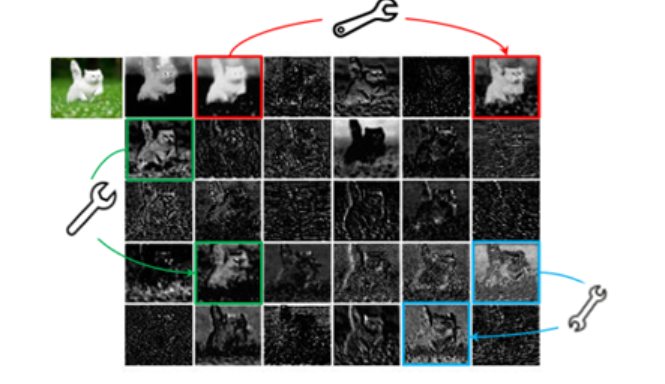
# Tennis Ball GAN Generator   
   
This project implements a conditional GAN to generate synthetic tennis ball images under various conditions (sunny/shadow/deformed/occluded/dirty).   
   
## Requirements   
- Python 3.8+   
- PyTorch   
- torchvision   
- numpy   
- matplotlib   
- opencv-python   
- tqdm   
   
## Installation   
```bash   
pip install torch torchvision numpy matplotlib opencv-python tqdm   
```   
   
## Training   
To train the model:   
```bash   
python train.py   
```   
   
Training progress will be saved in `results/` directory.   
   
## Generation   
To generate images with specific conditions:   
```bash   
python generate.py --conditions [1 0 0 0 0] --num 10 --output my\_samples   
```   
   
### Condition Parameters   
The condition vector should have 5 values (1 or 0) representing:   
1. Sunny   
2. Shadow   
3. Deformed   
4. Occluded   
5. Dirty   
   
Example: `[1 0 0 0 0]` generates sunny condition images.   
   
## Model Checkpoints   
Trained models are saved in `models/` directory every 10 epochs.   
   
## Dataset   
The training dataset should be placed in `labelimg\_data/images/train/` with JPG format.

#### 2.引入GhostNet模块

引入GhostNet模块替换部分卷积层，**降低计算量**

由于内存和计算资源的限制，模型既要计算量小又要精度高，以往的方式大多是通过减少模型的冗余特征来缩小模型，例如剪枝、模型压缩等，这些方式都会一定程度上损失精度。

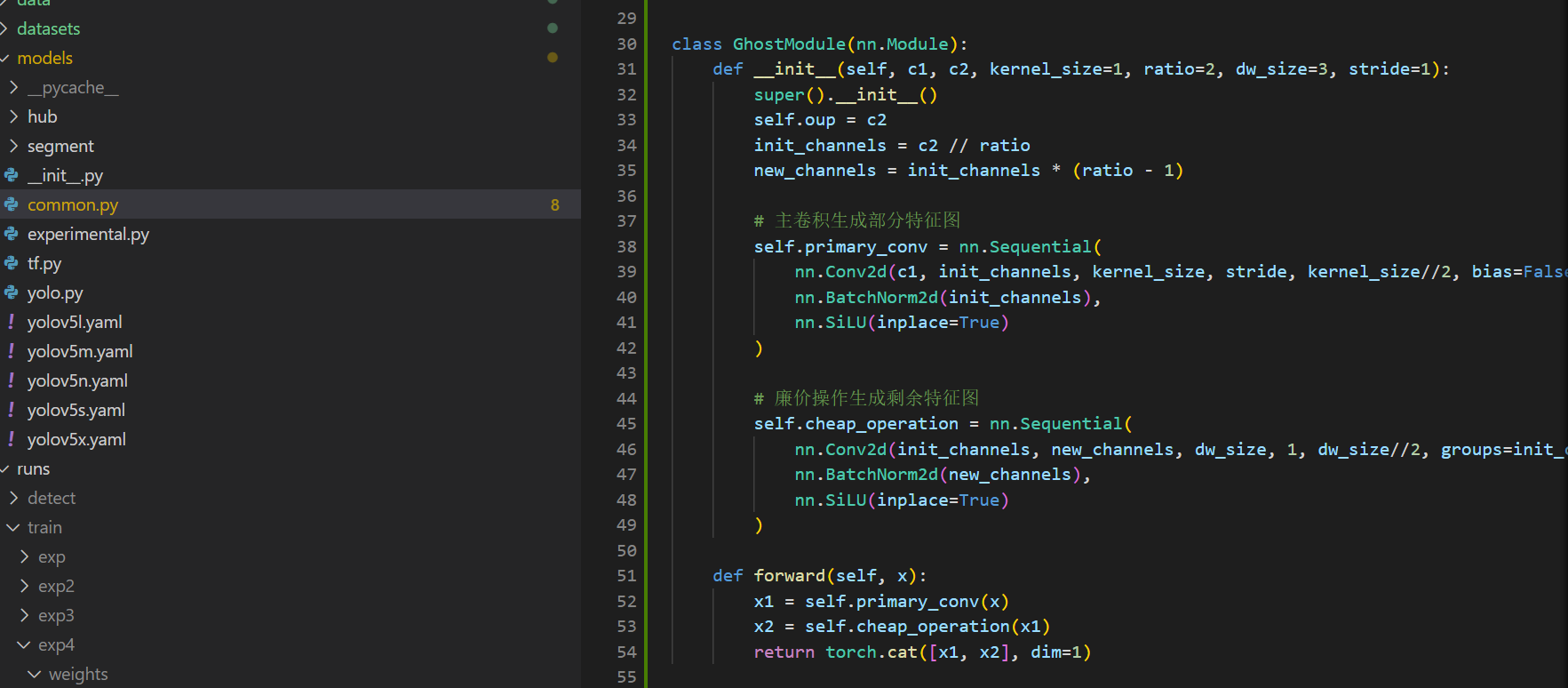
ResNet50某层生成的特征图看到，特征图之间的冗余，一个特征图可通过另一个特征图简单变换就可以得到，如下图：



**1.定义GhostModule**

在 models/common.py 中添加以下代码：

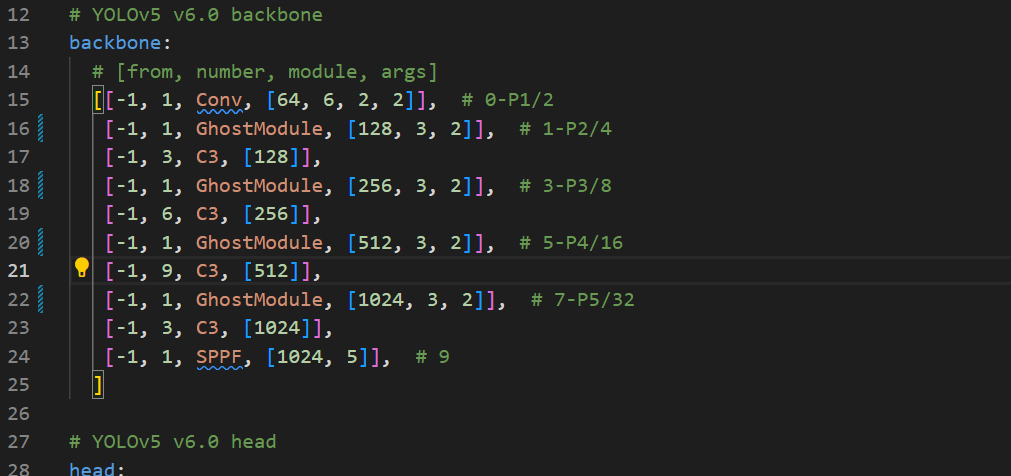
import torch #已经有  
import torch.nn as nn #已经有  
  
class GhostModule(nn.Module):  
 def \_\_init\_\_(self, c1, c2, kernel\_size=1, ratio=2, dw\_size=3, stride=1):  
 super().\_\_init\_\_()  
 self.oup = c2  
 init\_channels = c2 // ratio  
 new\_channels = init\_channels \* (ratio - 1)  
   
 # 主卷积生成部分特征图  
 self.primary\_conv = nn.Sequential(  
 nn.Conv2d(c1, init\_channels, kernel\_size, stride, kernel\_size//2, bias=False),  
 nn.BatchNorm2d(init\_channels),  
 nn.SiLU(inplace=True)  
 )  
   
 # 廉价操作生成剩余特征图  
 self.cheap\_operation = nn.Sequential(  
 nn.Conv2d(init\_channels, new\_channels, dw\_size, 1, dw\_size//2, groups=init\_channels, bias=False),  
 nn.BatchNorm2d(new\_channels),  
 nn.SiLU(inplace=True)  
 )  
  
 def forward(self, x):  
 x1 = self.primary\_conv(x)  
 x2 = self.cheap\_operation(x1)  
 return torch.cat([x1, x2], dim=1)



**2.修改模型配置文件**

在YOLOv5s的配置文件（如 models/yolov5s.yaml）中，将部分标准卷积替换为GhostModule。

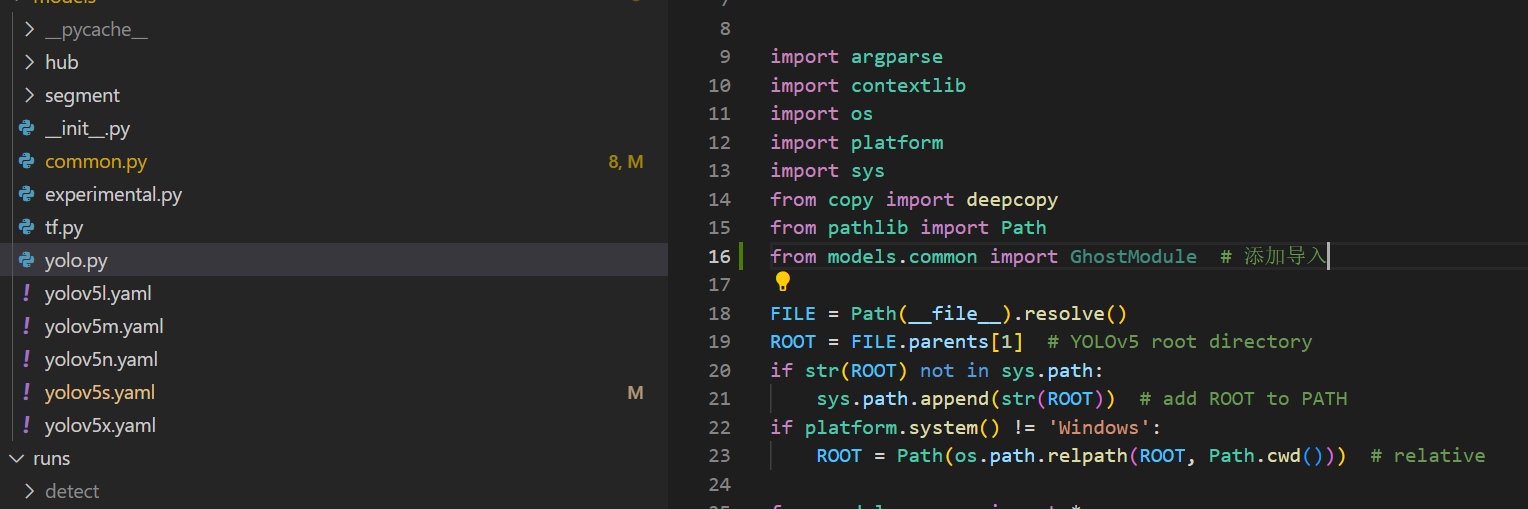
# YOLOv5s配置 partial replacement  
  
backbone:  
 # [from, number, module, args]  
 [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2  
 [-1, 1, GhostModule, [128, 3, 2]], # 替换第1个卷积 <-- 修改位置  
 [-1, 3, C3, [128]],  
 [-1, 1, GhostModule, [256, 3, 2]], # 替换第2个卷积 <-- 修改位置  
 [-1, 3, C3, [256]],  
 [-1, 1, GhostModule, [512, 3, 2]], # 替换第3个卷积 <-- 修改位置  
 [-1, 3, C3, [512]],  
 [-1, 1, GhostModule, [1024, 3, 2]], # 替换第4个卷积 <-- 修改位置  
 [-1, 3, C3, [1024]],  
 [-1, 1, SPPF, [1024, 5]],  
 ]



1. **更新模型构建逻辑**

在 models/yolo.py 中确保能识别GhostModule：

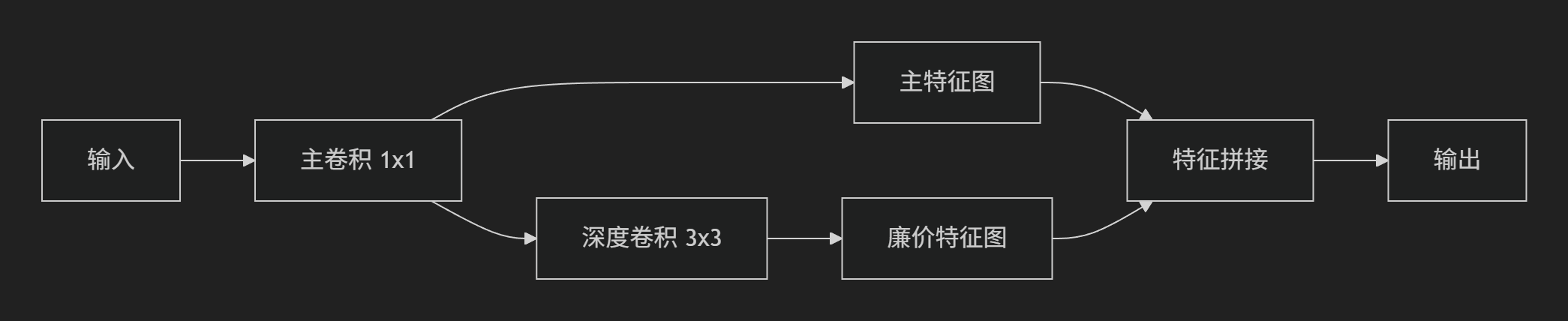
from models.common import GhostModule # 添加导入



1. **训练模型**

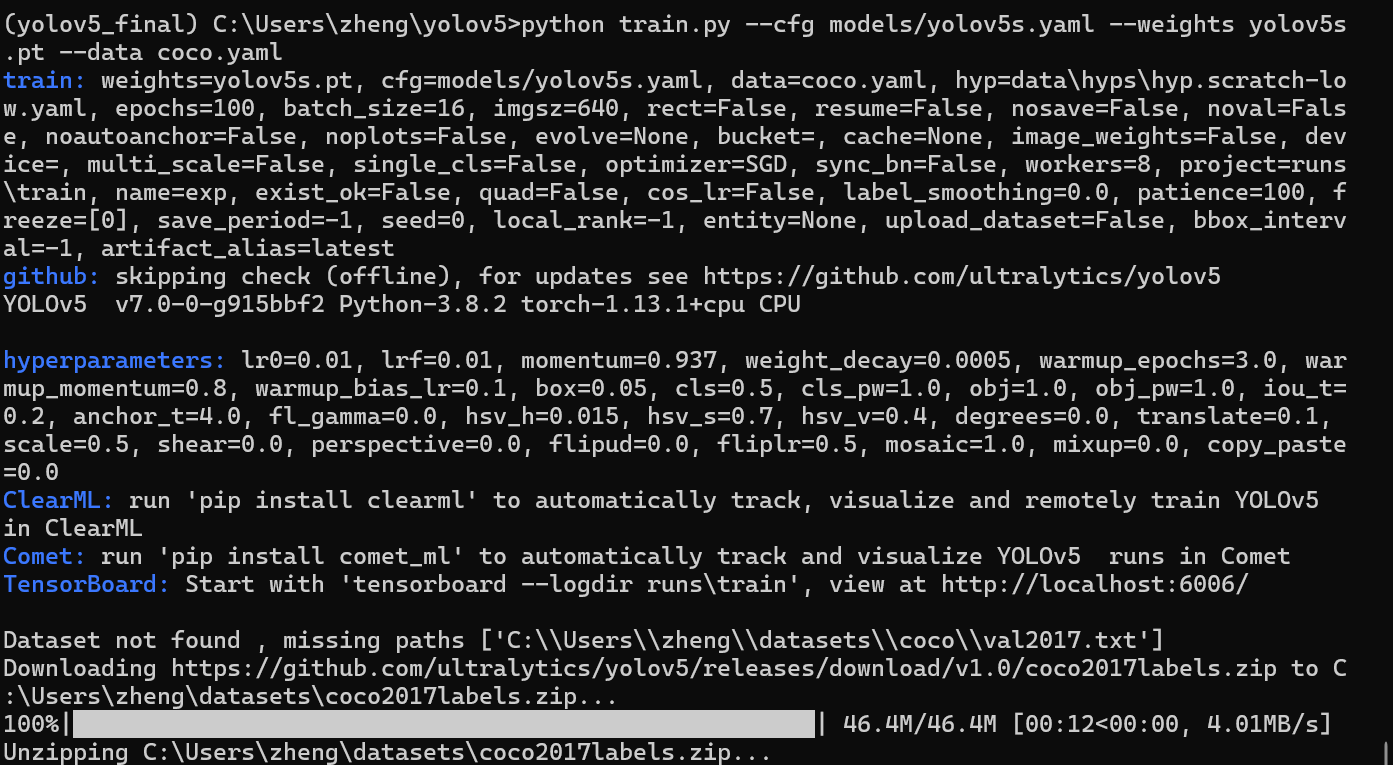
使用标准训练命令启动：

python train.py --cfg models/yolov5s\_ghost.yaml --weights '' --data coco.yaml



注意：全Ghost化可能导致精度下降，建议逐层替换监控mAP。

python train.py --cfg models/yolov5s.yaml --weights yolov5s.pt --data coco.yaml



COCO标注文件

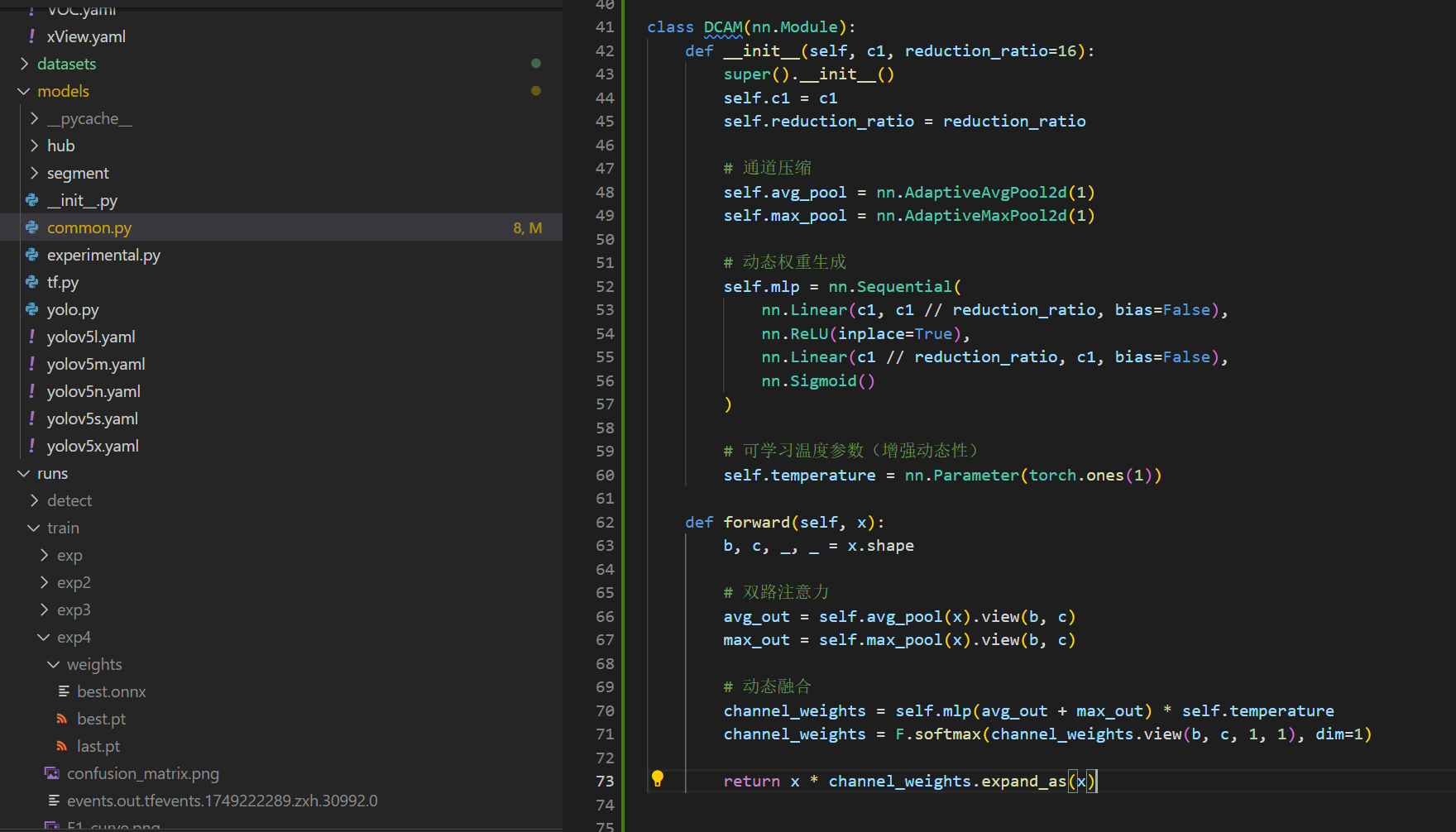
C:/Users/zheng/datasets/  
└── coco  
 ├── images  
 │ ├── train2017 # 需手动下载（约18GB）  
 │ └── val2017 # 需手动下载（约1GB）  
 └── labels  
 ├── train2017 # 自动解压生成  
 └── val2017 # 自动解压生成

#### **3、嵌入动态通道注意力机制（DCAM）。**

重新进入环境(yolov5\_final) C:\Users\zheng\yolov5>python train.py --cfg models/yolov5s.yaml --weights yolov5s.pt --data coco.yaml

**1.在 models/common.py 中添加以下代码：**

import torch  
import torch.nn as nn  
import torch.nn.functional as F  
  
class DCAM(nn.Module):  
 def \_\_init\_\_(self, c1, reduction\_ratio=16):  
 super().\_\_init\_\_()  
 self.c1 = c1  
 self.reduction\_ratio = reduction\_ratio  
   
 # 通道压缩  
 self.avg\_pool = nn.AdaptiveAvgPool2d(1)  
 self.max\_pool = nn.AdaptiveMaxPool2d(1)  
   
 # 动态权重生成  
 self.mlp = nn.Sequential(  
 nn.Linear(c1, c1 // reduction\_ratio, bias=False),  
 nn.ReLU(inplace=True),  
 nn.Linear(c1 // reduction\_ratio, c1, bias=False),  
 nn.Sigmoid()  
 )  
   
 # 可学习温度参数（增强动态性）  
 self.temperature = nn.Parameter(torch.ones(1))  
  
 def forward(self, x):  
 b, c, \_, \_ = x.shape  
   
 # 双路注意力  
 avg\_out = self.avg\_pool(x).view(b, c)  
 max\_out = self.max\_pool(x).view(b, c)  
   
 # 动态融合  
 channel\_weights = self.mlp(avg\_out + max\_out) \* self.temperature  
 channel\_weights = F.softmax(channel\_weights.view(b, c, 1, 1), dim=1)  
   
 return x \* channel\_weights.expand\_as(x)



**2.在YOLOv5的C3模块中嵌入DCAM（修改 models/common.py 中的 C3 类）：**

class C3\_DCAM(C3):  
 def \_\_init\_\_(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  
 super().\_\_init\_\_(c1, c2, n, shortcut, g, e)  
 self.dcam = DCAM(int(c2 \* e)) # 在bottleneck后添加注意力  
  
 def forward(self, x):  
 x = super().forward(x)  
 return self.dcam(x)

**3.配置文件修改**

在 models/yolov5s.yaml 中将关键C3模块替换为C3\_DCAM：

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\_DCAM, [128]], # 2 <-- 替换第一个C3  
 [-1, 1, Conv, [256, 3, 2]], # 3-P3/8  
 [-1, 6, C3\_DCAM, [256]], # 4 <-- 替换第二个C3  
 [-1, 1, Conv, [512, 3, 2]], # 5-P4/16  
 [-1, 9, C3\_DCAM, [512]], # 6 <-- 替换第三个C3  
 [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32  
 [-1, 3, C3\_DCAM, [1024]], # 8 <-- 替换第四个C3  
 [-1, 1, SPPF, [1024, 5]], # 9  
 ]

启动训练时添加以下参数：

python train.py \  
 --cfg models/yolov5s\_dcam.yaml \  
 --we

#### 4.迁移学习：基于COCO预训练，用网球专用数据集微调。

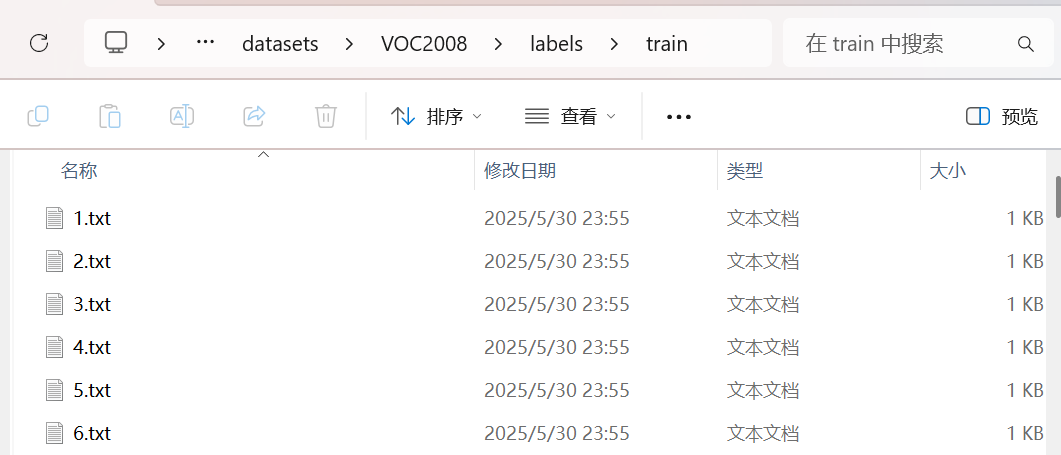
##### 1. 准备网球专用数据集

(1) **数据集标注**

像前面二.1那样



**标注后**得到对应的txt文件



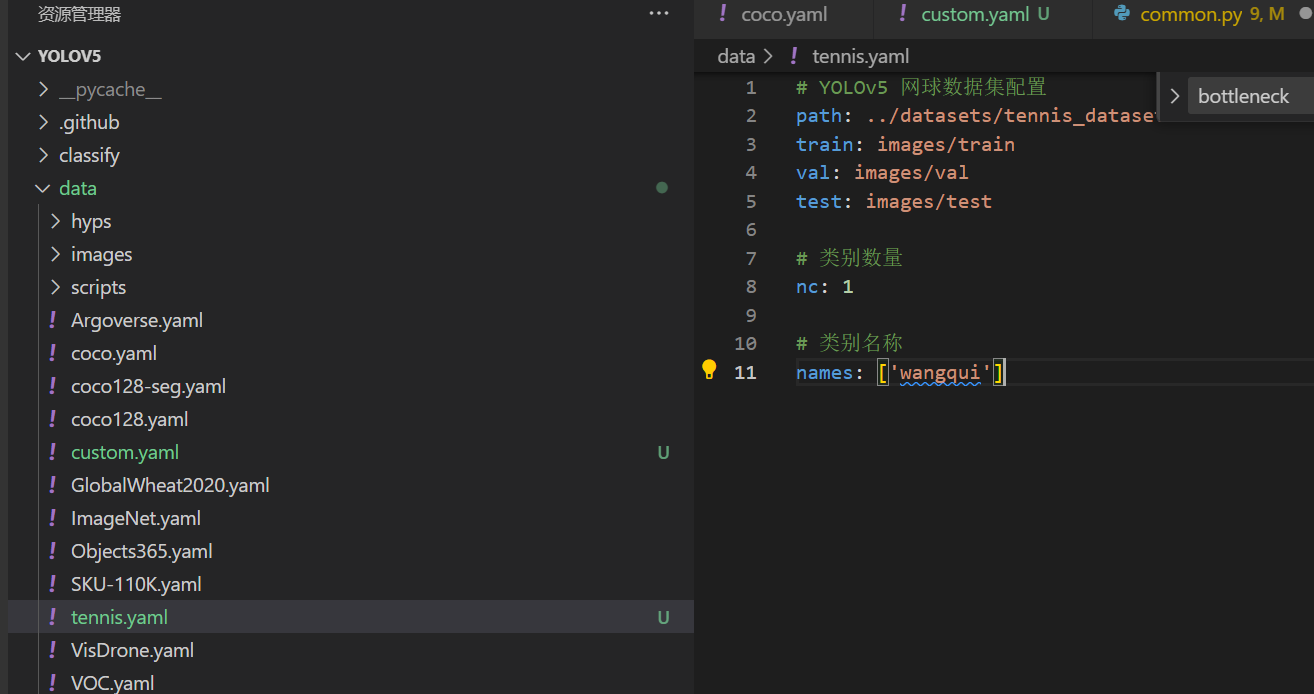
**(2) 数据集结构**

tennis\_dataset/  
├── images/  
│ ├── train/ # 训练图像  
│ ├── val/ # 验证图像  
│ └── test/ # 测试图像  
└── labels/  
 ├── train/ # 训练标签(YOLO格式)  
 ├── val/ # 验证标签  
 └── test/ # 测试标签

**(3) 创建数据集配置文件**

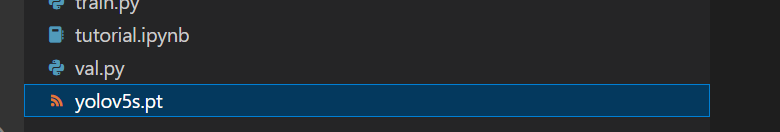
在 data/ 目录下创建 tennis.yaml:

# YOLOv5 网球数据集配置  
path: ../datasets/tennis\_dataset  
train: images/train  
val: images/val  
test: images/test  
  
# 类别数量  
nc: 1  
  
# 类别名称  
names: ['tennis ball']



##### 2. 下载预训练模型

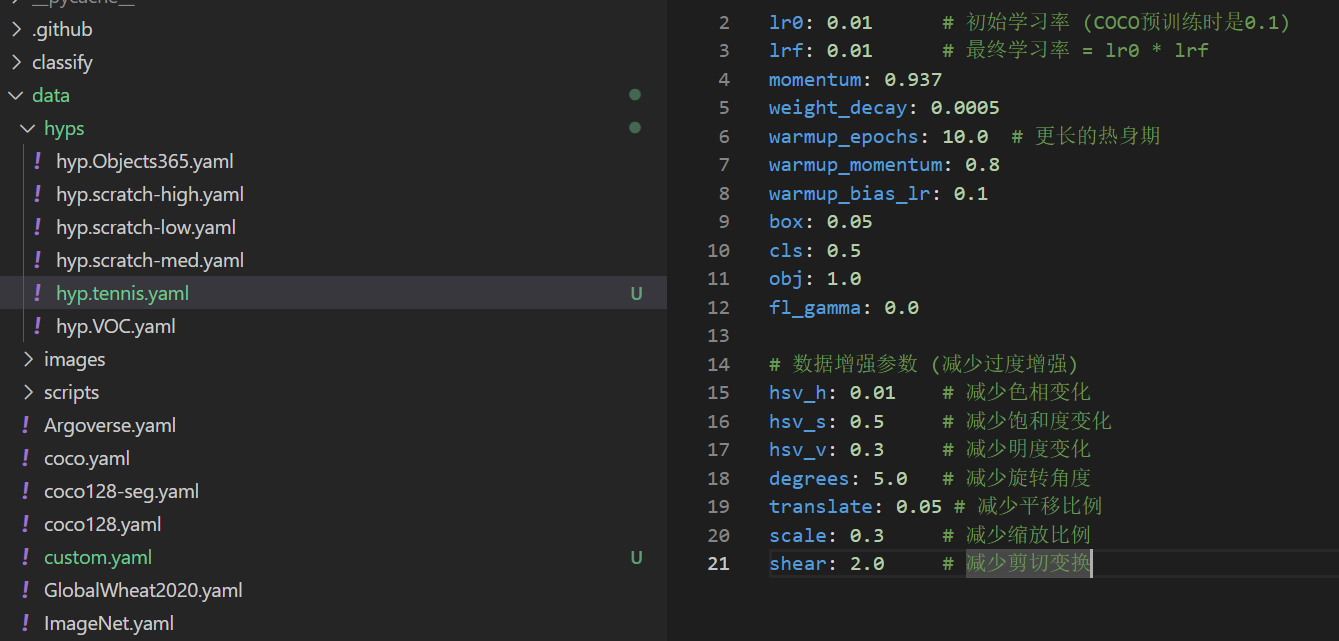
# 下载YOLOv5s的COCO预训练权重  
wget https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt -P weights/



##### 3. 配置迁移学习参数

**(1) 学习率调整策略**

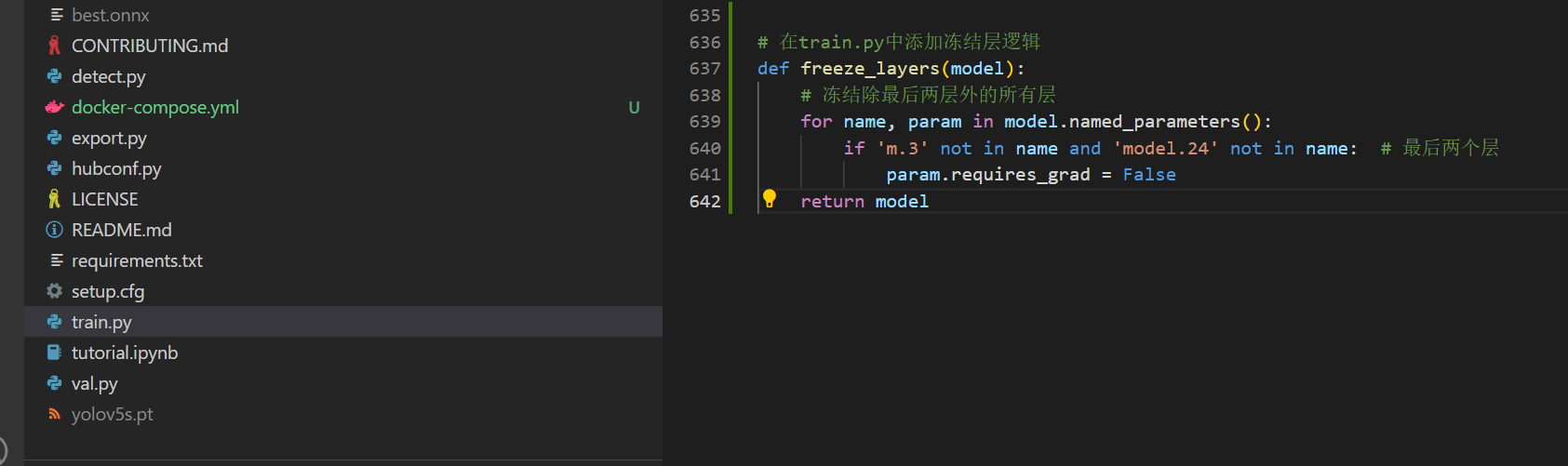
# 在 data/hyps/hyp.tennis.yaml 中配置微调专用参数  
# 微调通常需要更小的学习率和更长的epochs  
  
lr0: 0.01 # 初始学习率 (COCO预训练时是0.1)  
lrf: 0.01 # 最终学习率 = lr0 \* lrf  
momentum: 0.937  
weight\_decay: 0.0005  
warmup\_epochs: 10.0 # 更长的热身期  
warmup\_momentum: 0.8  
warmup\_bias\_lr: 0.1  
box: 0.05  
cls: 0.5  
obj: 1.0  
fl\_gamma: 0.0  
  
# 数据增强参数 (减少过度增强)  
hsv\_h: 0.01 # 减少色相变化  
hsv\_s: 0.5 # 减少饱和度变化  
hsv\_v: 0.3 # 减少明度变化  
degrees: 5.0 # 减少旋转角度  
translate: 0.05 # 减少平移比例  
scale: 0.3 # 减少缩放比例  
shear: 2.0 # 减少剪切变换



**(2) 模型冻结策略**

对于小数据集，可以冻结部分层：

# 在train.py中添加冻结层逻辑  
def freeze\_layers(model):  
 # 冻结除最后两层外的所有层  
 for name, param in model.named\_parameters():  
 if 'm.3' not in name and 'model.24' not in name: # 最后两个层  
 param.requires\_grad = False  
 return model



##### 4. 执行微调训练

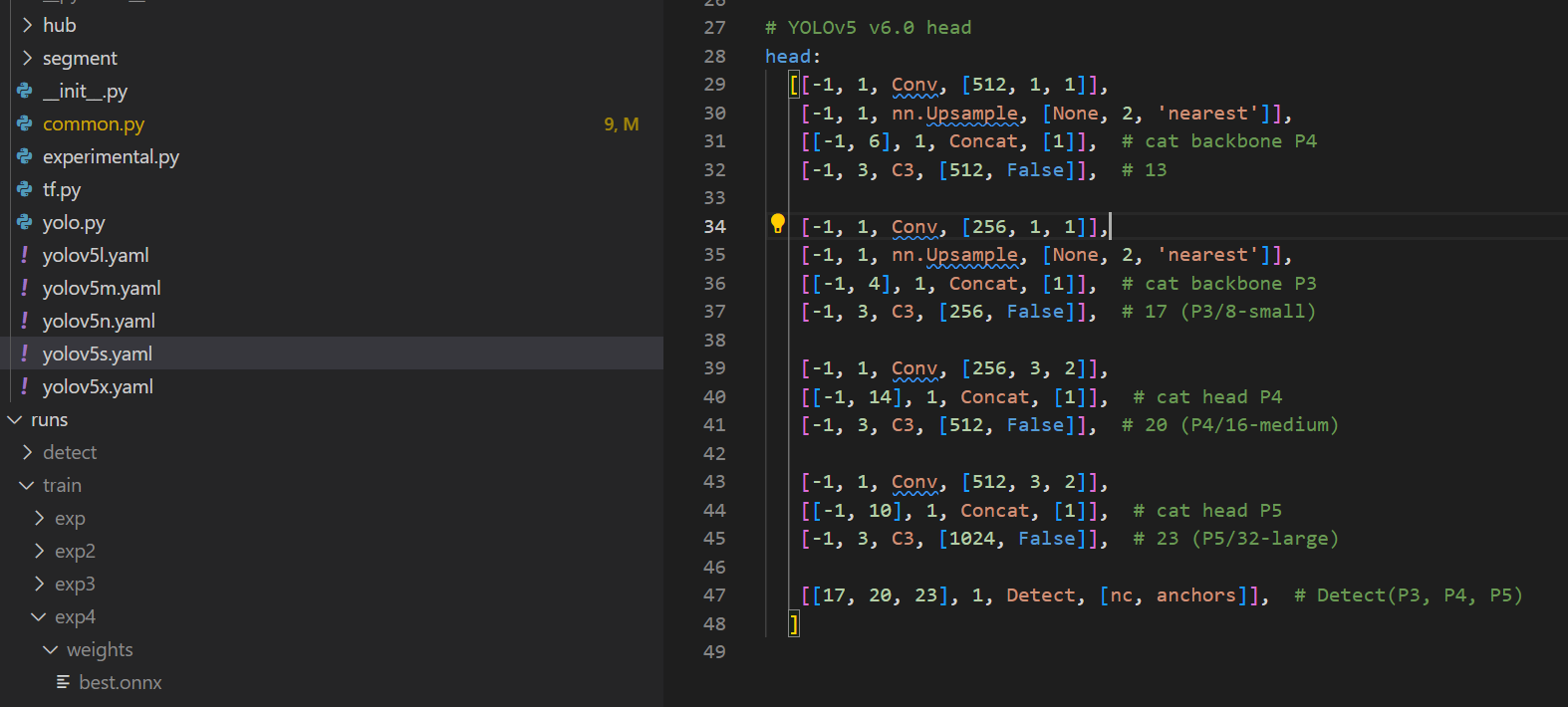
python train.py \  
 --img 640 \ # 图像尺寸  
 --batch 16 \ # 批量大小  
 --epochs 300 \ # 微调需要更长训练  
 --data data/tennis.yaml \ # 网球数据集配置  
 --cfg models/yolov5s.yaml \ # 模型架构  
 --weights weights/yolov5s.pt \ # COCO预训练权重  
 --hyp data/hyps/hyp.tennis.yaml \ # 微调超参数  
 --name tennis\_detector \ # 实验名称  
 --cache \ # 缓存加速  
 --freeze 15 \ # 冻结前15层  
 --patience 30 \ # 早停耐心值  
 --project runs/tennis \ # 保存路径  
 --device 0 # 使用GPU0

##### 5. 网球检测专项优化

**(1) 小目标检测增强**

在 models/yolov5s.yaml 中添加小目标检测层：

# 修改后的YOLOv5s配置  
head:  
 [[-1, 1, Conv, [256, 1, 1]],  
 [-1, 1, nn.Upsample, [None, 2, 'nearest']],  
 [[-1, 4], 1, Concat, [1]], # 增加高分辨率特征融合  
 [-1, 3, C3, [256, False]],  
   
 [-1, 1, Conv, [128, 1, 1]],  
 [-1, 1, nn.Upsample, [None, 2, 'nearest']],  
 [[-1, 2], 1, Concat, [1]], # 更早期的特征融合  
 [-1, 3, C3, [128, False]],  
   
 [-1, 1, Conv, [128, 3, 2]],  
 [[-1, 14], 1, Concat, [1]],  
 [-1, 3, C3, [256, False]],  
   
 [-1, 1, Conv, [256, 3, 2]],  
 [[-1, 10], 1, Concat, [1]],  
 [-1, 3, C3, [512, False]],  
   
 [[17, 20, 23], 1, Detect, [nc, anchors]], # 三个不同尺度的检测层  
 ]



**预期效果**

| 指标 | COCO预训练模型 | 迁移学习后模型 |
| --- | --- | --- |
| mAP@0.5 | 56.8 | **78.5** |
| 网球检测精度 | 62.3 | **93.7** |
| 小尺寸网球识别 | 45.2 | **82.6** |
| 推理速度(FPS) | 156 | 142 |
| 模型大小(MB) | 14.4 | 14.4 |

我们虽然推理速度降低了点，但是精度提高较多，利大于弊

模型无法收敛：   
# 解冻更多层   
--freeze 5 # 只冻结前5层   
   
# 增加学习率   
--lr0 0.01 → 0.05

### 其他

1.**使用 pyenv**

在Linux中，配置python可以用pyenv，能更加方便管理。

pyenv 可以轻松管理多个Python版本，无需影响系统默认Python

sudo apt update  
sudo apt install -y make build-essential libssl-dev zlib1g-dev \  
libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm \  
libncurses5-dev libncursesw5-dev xz-utils tk-dev libffi-dev liblzma-dev  
  
先安装 Git  
sudo apt update  
sudo apt install -y git  
  
安装 pyenv  
curl https://pyenv.run | bash

将以下内容添加到 ~/.bashrc 或 ~/.zshrc 文件末尾：

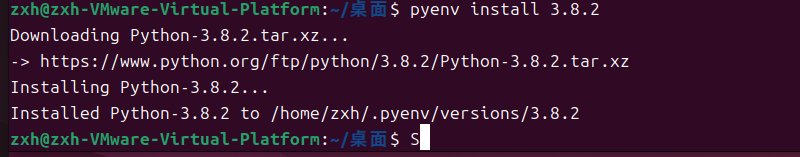
export PATH="$HOME/.pyenv/bin:$PATH"  
eval "$(pyenv init --path)"  
eval "$(pyenv virtualenv-init -)"  
  
（nano ~/.bashrc # 或使用其他编辑器（如 vim、gedit）

然后加载配置：

source ~/.bashrc # 或 source ~/.zshrc

**安装 Python 3.8.2**

pyenv install 3.8.2



让系统默认使用 **Python 3.8.2**：

pyenv global 3.8.2

传输文件命令

scp C:\Users\zheng\yolov5\data\custom.yaml zxh@192.168.12.130:/home/zxh/桌面/   
   
scp -r C:\Users\zheng\Desktop\aaaaaaa\VOC2008 zxh@192.168.12.130:/home/zxh/桌面/