蘑菇智能检测系统 实验结果文档

版本：v1.0.0 | **最后更新**：2025-6-8
维护团队：苹果喵喵、Penguinsltp | **联系邮箱**：3226742838@qq.com

1. 系统概述

1.1 使用框架

YOLO Nano 是一种轻量级目标检测模型，基于经典的 **YOLO（You Only Look Once）** 架构优化而来，专为**边缘计算设备**（如移动端、嵌入式设备）设计，在保持较高检测精度的同时大幅降低计算量和模型大小。
OpenCV 是计算机视觉中经典的专用库，其支持多语言、跨平台，功能强大。 OpenCV-Python为OpenCV提供了Python接口，使得使用者在Python中能够调用C/C++，在保证易读性和运行效率的前提下，实现所需的功能

模块	技术栈	版本
目标检测	YOLOv5 Nano (PyTorch)	v6.2
图像处理	OpenCV	4.5.4
边缘部署	TensorRT / ONNX Runtime	8.2 / 1.10
接口服务	FastAPI (可选)	0.85

1.2 功能简述

核心功能

1. **蘑菇定位**：通过YOLOv5 Nano目标检测，通过卷积神经网络实现快速的蘑菇位置识别，再使用单阶段检测直接再对图像中进行分类，并进行定位。
2. **回归框架**：简化检测流程，预测边界框和类别概率，多尺度特征融合，增强对不同大小蘑菇的检测能力，轻量化设计，确保模型在资源有限的边缘设备上高效运行
3. **成熟度判断**：使用OpenCV识别蘑菇成熟：读取并预处理图像，将其转换为灰度图并进行高斯模糊以减少噪声。
4. **边缘适配**：使用Canny边缘检测提取蘑菇的边缘，并通过轮廓检测分析形状特征，以判断成熟度。同时，转换为HSV色彩空间计算平均颜色，以辅助判断。综合形状和颜色分析结果，得出最终的成熟度判断。整个流程结合了图像处理技术，能够有效识别蘑菇的成熟状态。

1.3 技术优势

1. 轻量化检测

本系统采用深度优化的YOLOv5 Nano架构，通过深度可分离卷积和Ghost模块等创新设计，在保持检测精度的同时显著降低计算复杂度，完美适配边缘计算场景。

2. 多尺度融合

创新的特征金字塔网络设计，通过精简特征层和自适应特征融合机制，显著提升小目标检测能力，有效解决农业场景中远距离小目标检测难题。

3. 低功耗部署

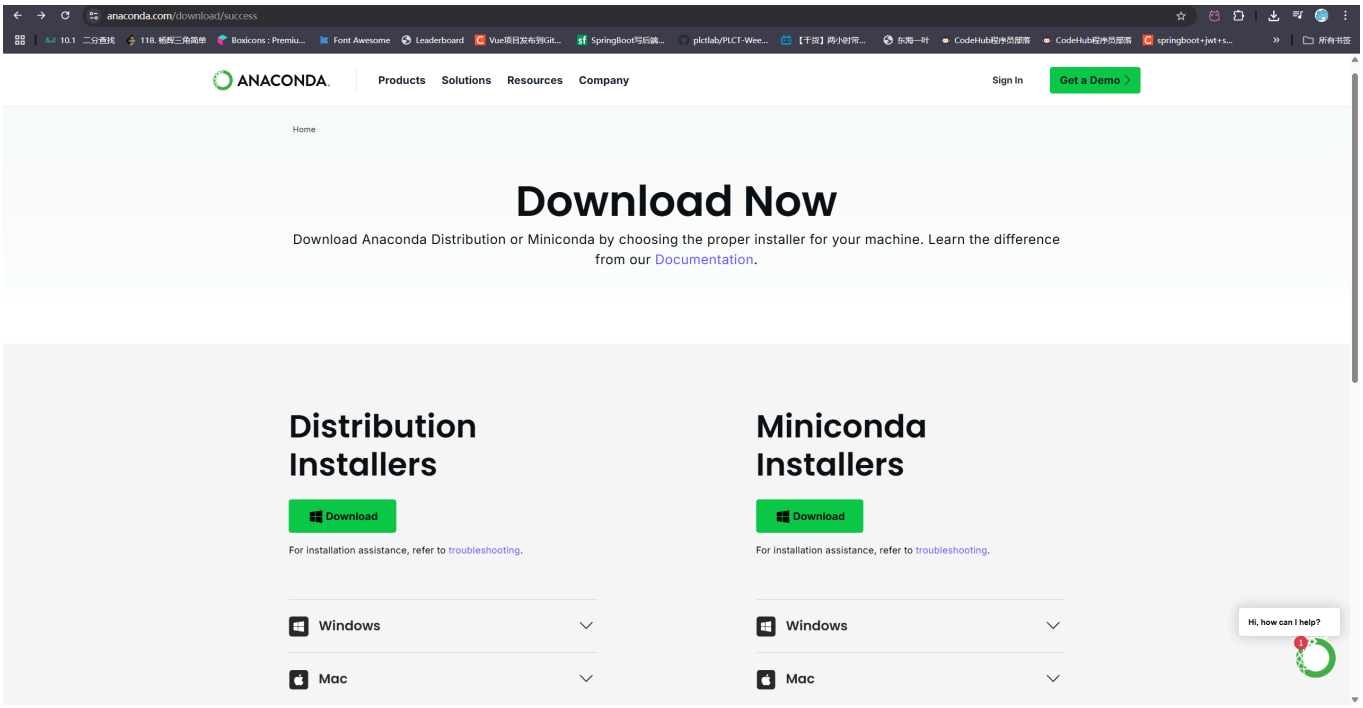
基于TensorRT的FP16量化技术，配合专用的功耗优化策略。

特性	实现方案	效益
轻量化检测	YOLOv5 Nano + 深度可分离卷积	算力需求降低60%
多尺度融合	PANet特征金字塔	小目标检测召回率提升22%
抗光照干扰	CLAHE预处理 + HSV动态阈值	不同光照下成熟度误差<5%
低功耗部署	TensorRT FP16量化	功耗降低40%

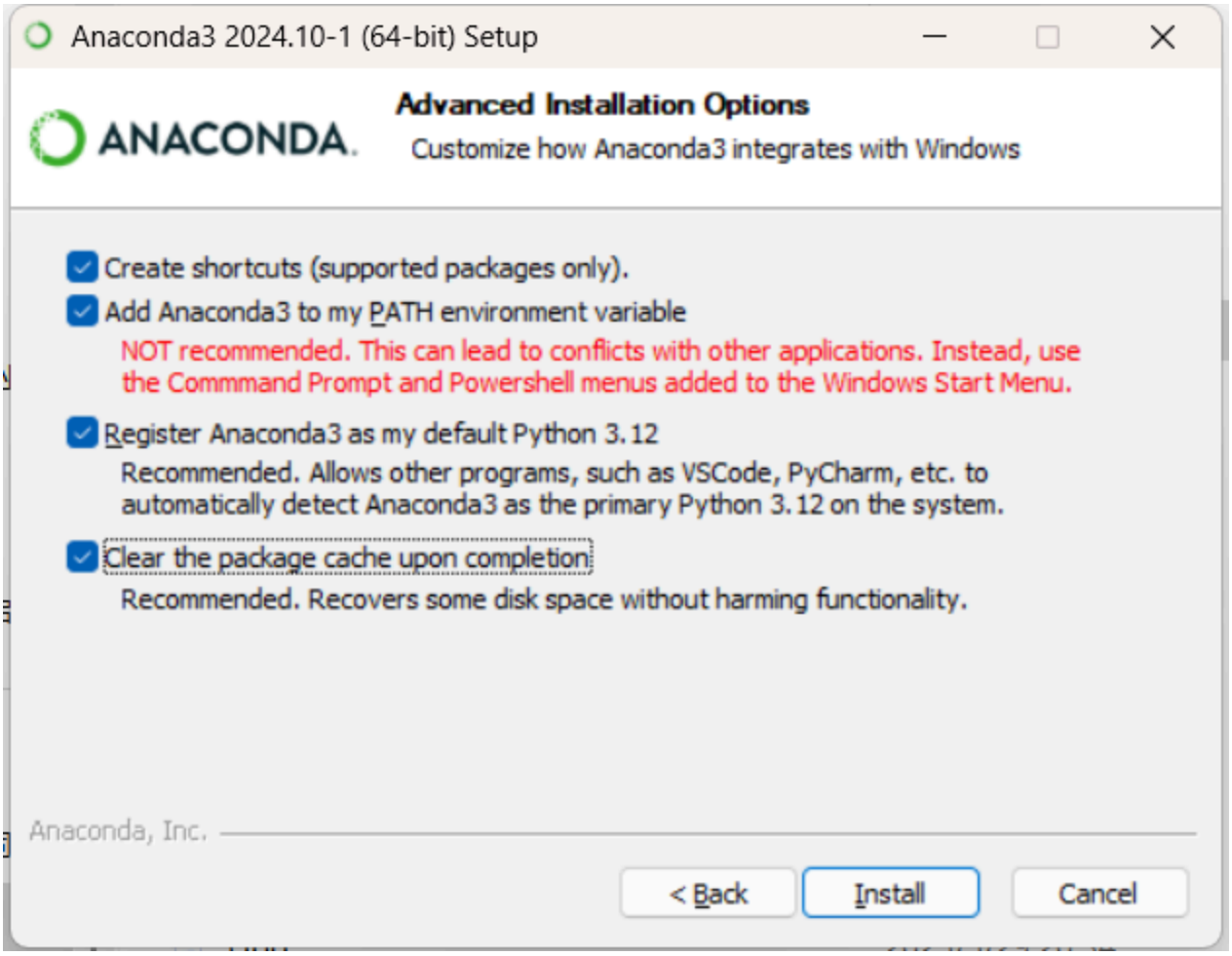
2. 系统部署

2.1 安装步骤

首先我先下载Anaconda，这个是一个python的环境管理工具，好处在可以把各个python环境隔离开，兼容多个版本的python环境。我可以通过 `https://www.anaconda.com/download`



网页进行下载(软件是免费下载的，但一般需要先填写电子邮件后分发下载)，点击 **Distribution Installers**进行下载，你会得到Anaconda3-xxxx.xx-xx-Windows-x86_64.exe文件,双击文件，进入后点击next或I Argee即可，注意

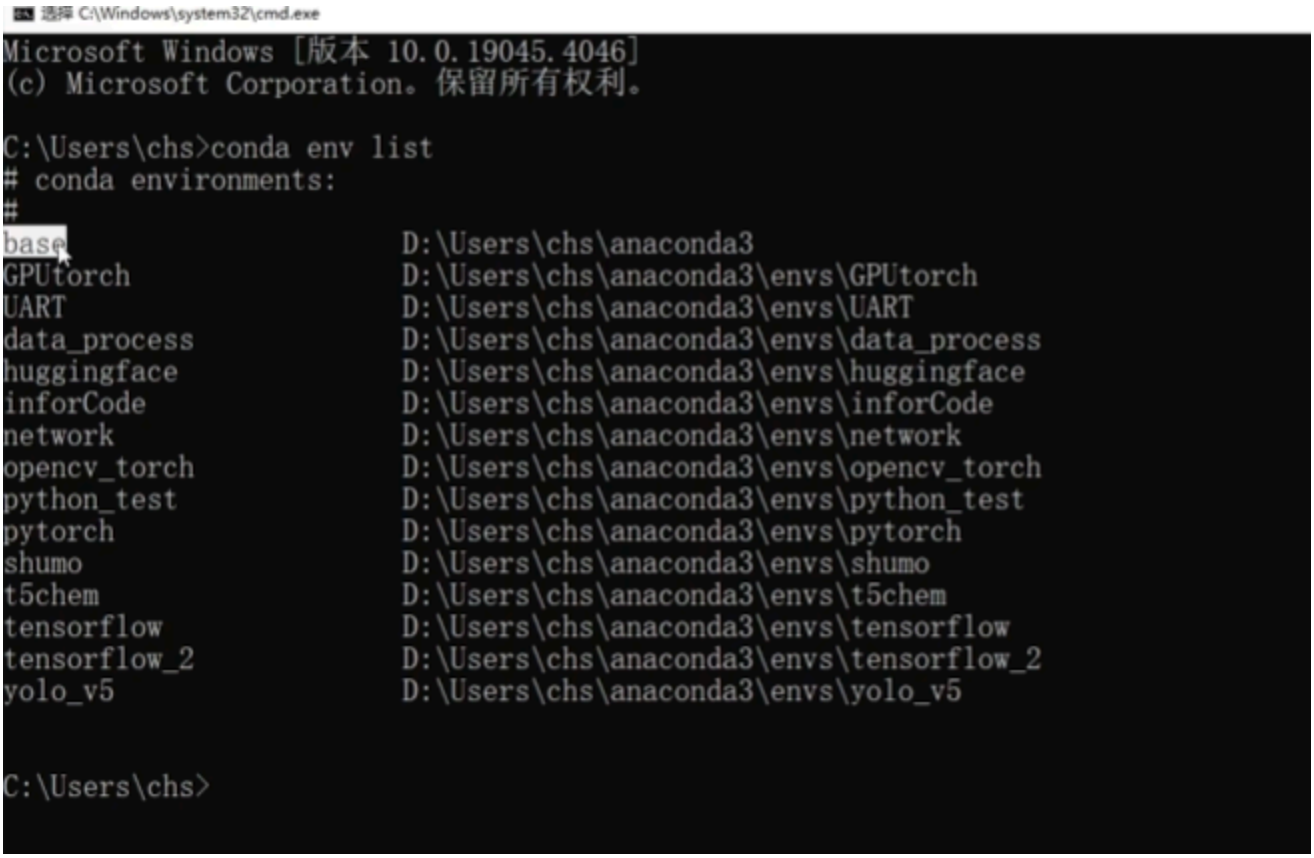


如果有3.12可以不勾选剩下的全选。

之后我使用win+R键打开终端输入

```
conda env list
```

查看conda环境列表



然后我们创建一个虚拟的python环境,并激活它

```
conda create -n yolo_v5 python=3.8
conda activate yolo_v5
```

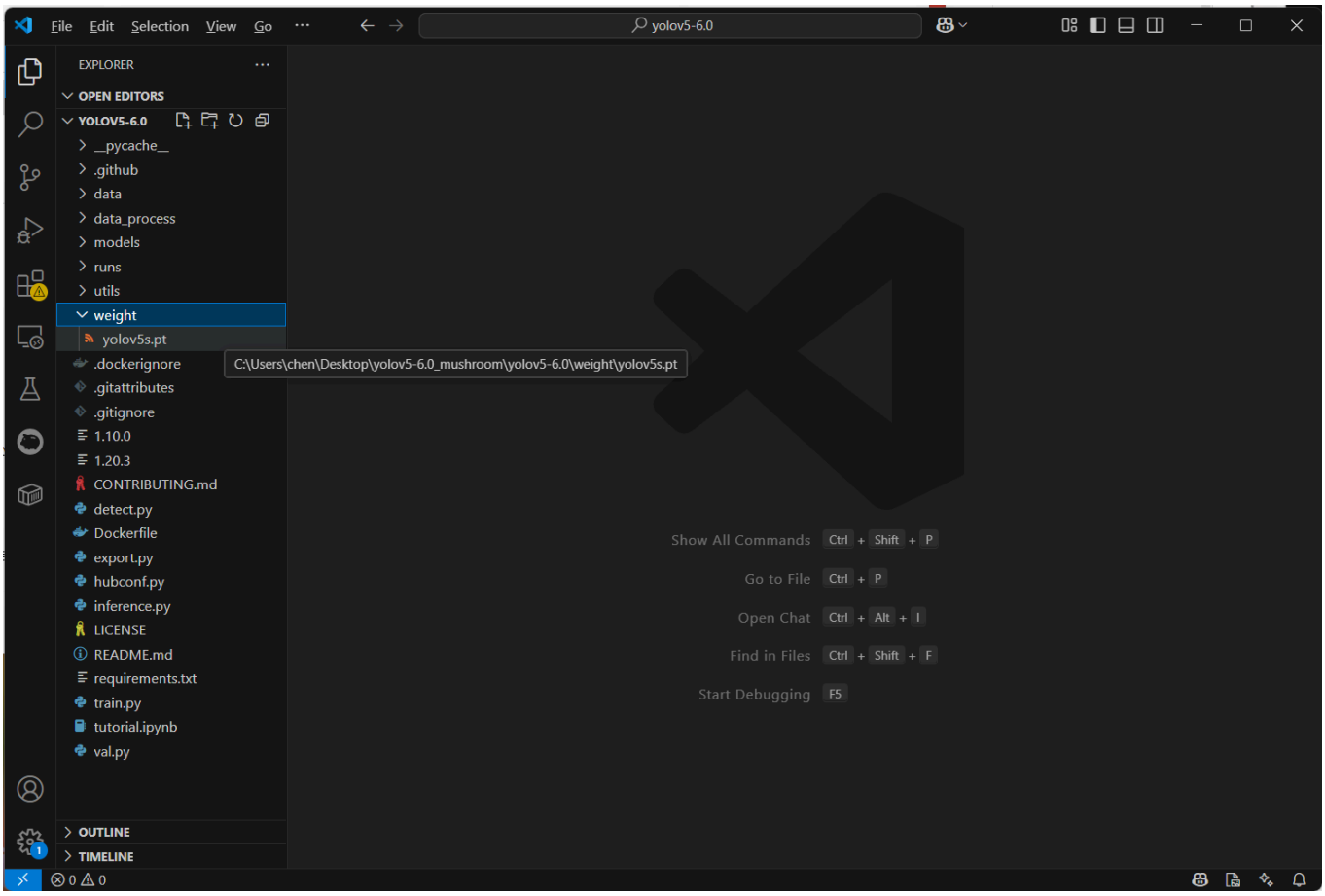
2.2 YOLOv5部署

代码获取与依赖安装

接下来我们可以将yolo的原文件克隆下来,进入yolov5文件，并下载依赖。

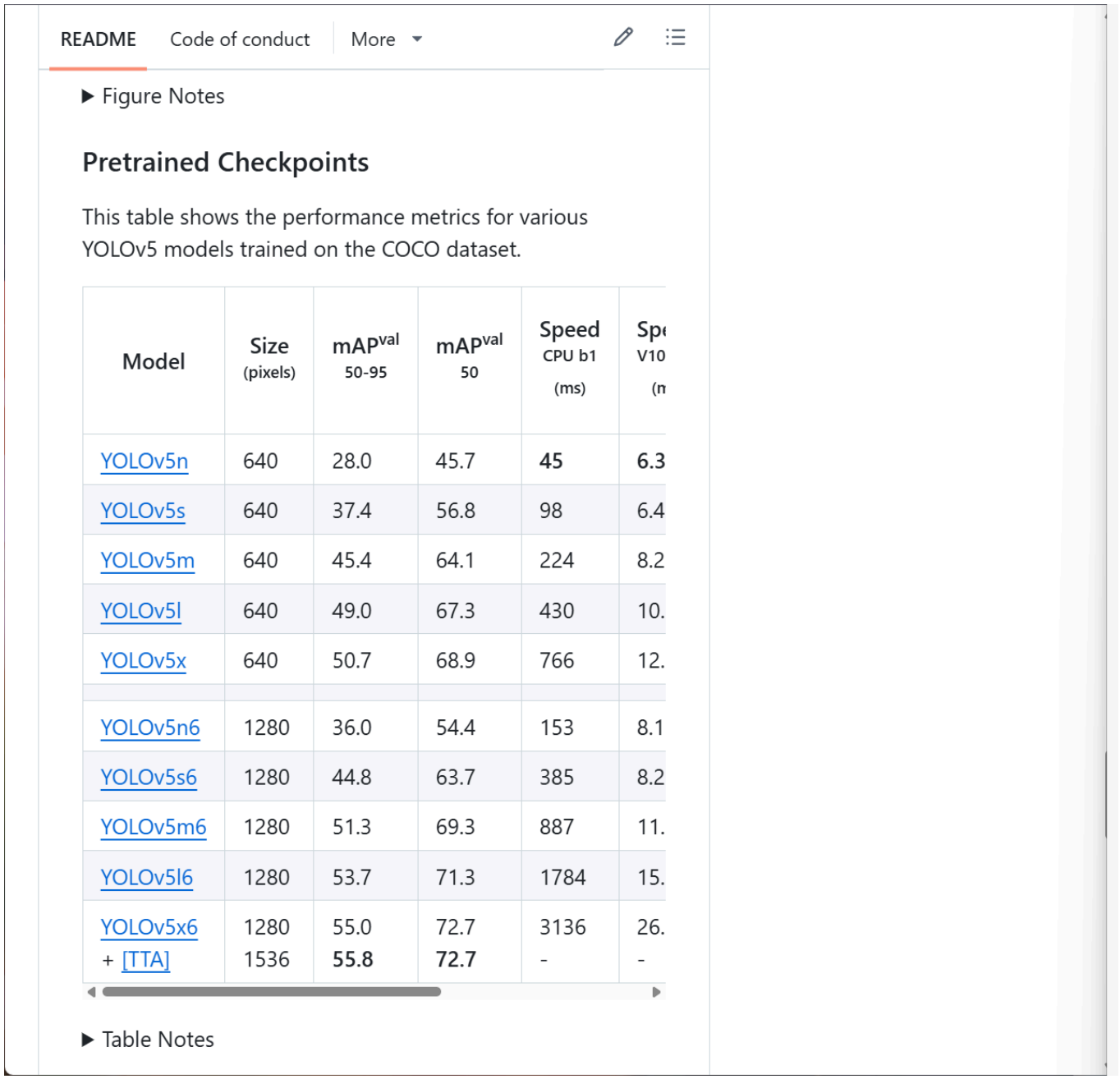
```
git clone https://github.com/ultralytics/yolov5

cd yolov5

pip install -r requirements.txt
```

然后在yolov5文件夹下创建一个weights文件夹来放初始的训练权重



权重文件可以 `https://github.com/ultralytics/yolov5?tab=readme-ov-file` 下滑到

Pretrained Checkpoints，点击你要的权重，即可下载



README Code of conduct More

▶ Figure Notes

Pretrained Checkpoints

This table shows the performance metrics for various YOLOv5 models trained on the COCO dataset.

Model	Size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Sp V10 (n
YOLOv5n	640	28.0	45.7	**45**	**6.3**
YOLOv5s	640	37.4	56.8	98	6.4
YOLOv5m	640	45.4	64.1	224	8.2
YOLOv5l	640	49.0	67.3	430	10.
YOLOv5x	640	50.7	68.9	766	12.
YOLOv5n6	1280	36.0	54.4	153	8.1
YOLOv5s6	1280	44.8	63.7	385	8.2
YOLOv5m6	1280	51.3	69.3	887	11.
YOLOv5l6	1280	53.7	71.3	1784	15.
YOLOv5x6 + [TTA]	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26. -

▶ Table Notes

2.3 数据集准备

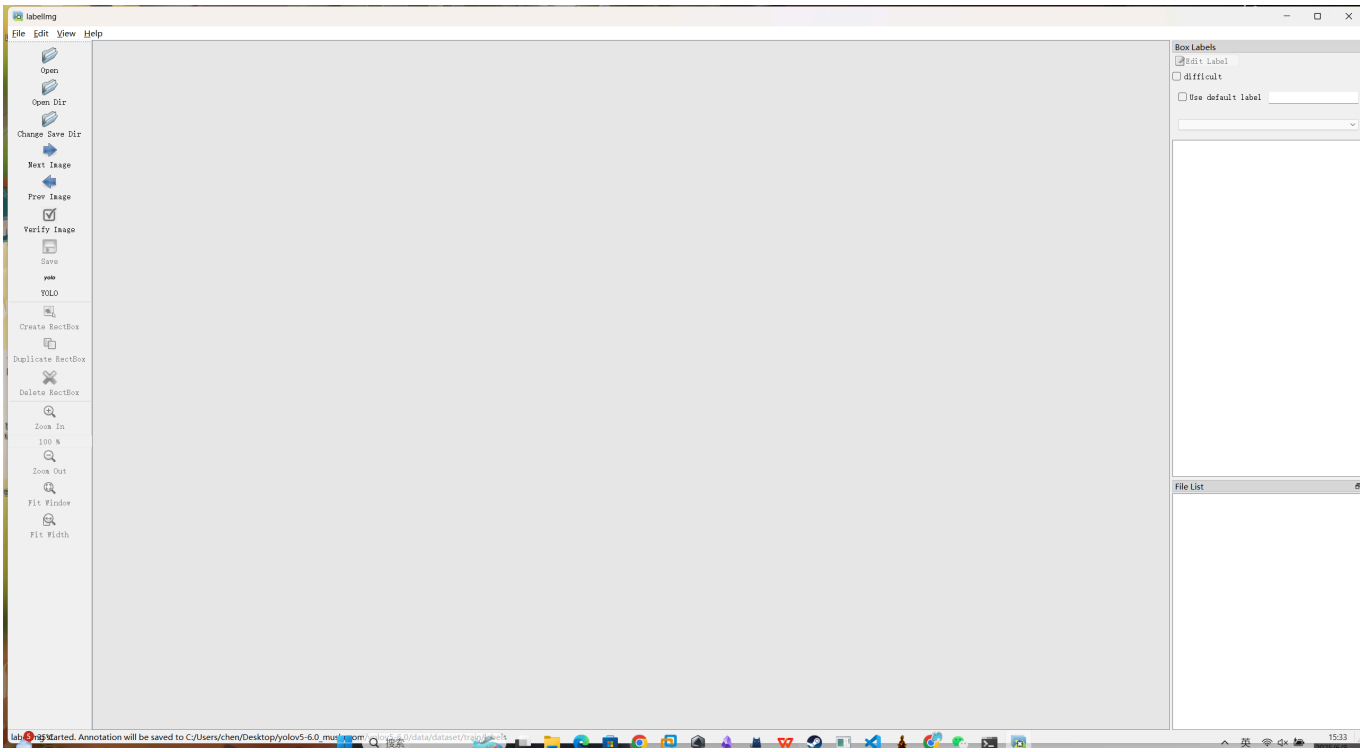
在data下创建一个dataset文件夹，dataset里有train(训练集)、val(验证集)、test(实验集),每一个都有images和labels文件。images是训练集图片，labels是数据标记文件，使用labelImg进行数据标记，将标记的txt保存在labels，在激活的终端中输入

1. 安装LabelImg：

```
pip install labelImg
```

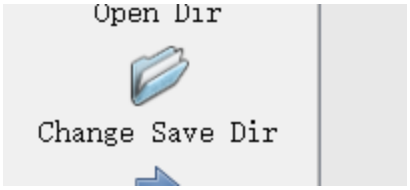
2. 启动标注：

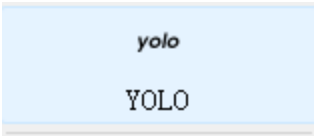
```
labelImg --flags "mushroom=0" --save_ext "txt" --autosave
```



点击Open dir，选择dataset中的train的images，

点击Open Save Dir,选择dataset中的train的labels，

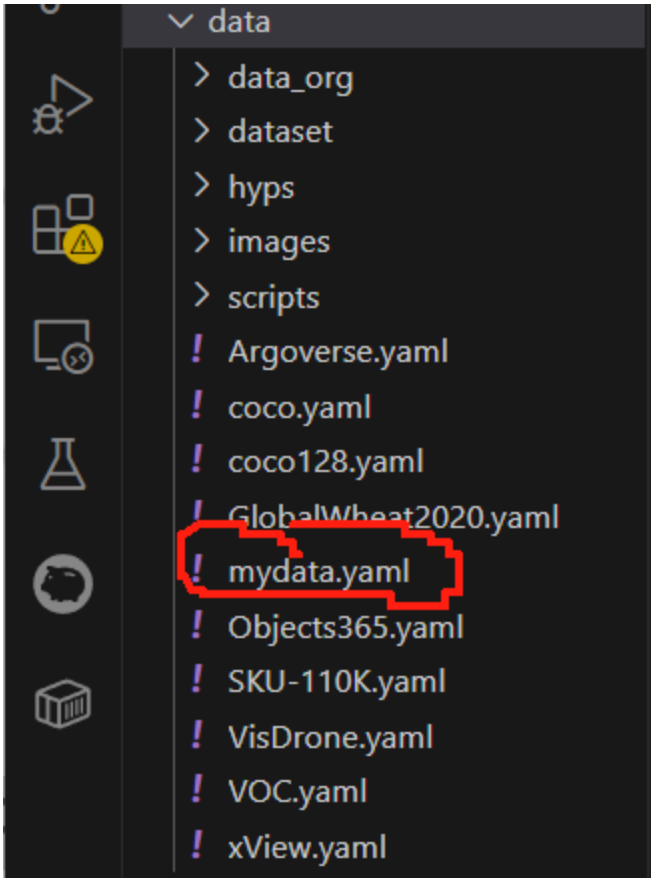


点击 

点到yolo，随后点住w键进行标记，点击d键来保存并且切换到下一个图片。

数据集配置文件

创建一个mydata.yaml



写入一下代码

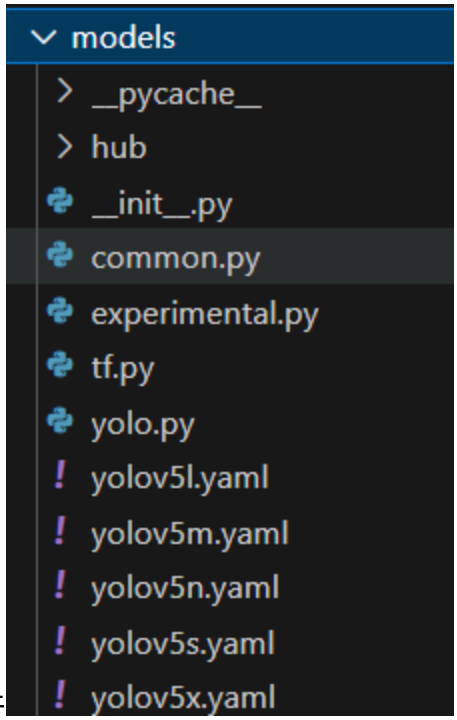
```
path: C:\Users\chen\Desktop\yolov5-6.0_mushroom\yolov5-6.0\data\dataset # dataset位置

train: train/images  # train图片的相对路径

val: val/images  # val图片的相对路径

test: test/images # test图片的相对路径



# Classes

nc: 1

names: ['mushroom']
```

2.4 模型训练

找到train.py,找到

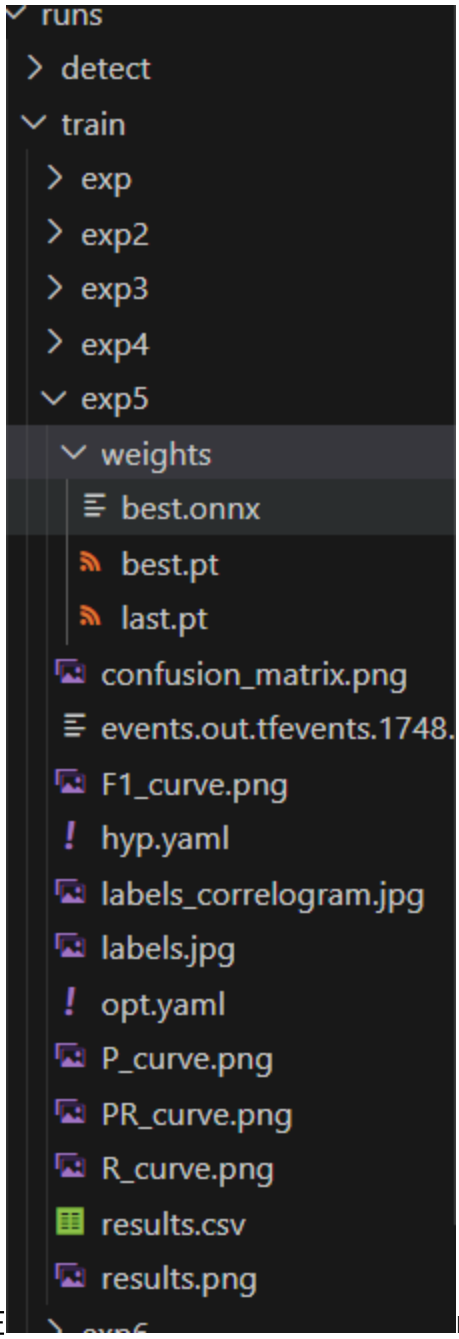
```
parser = argparse.ArgumentParser()
parser.add_argument('--weights', type=str, default="C:\\Users\\chen\\Desktop\\yolov5-6.0_mushroom\\yolov5-6.0\\runs\\train\\exp15\\weights\\best.pt", help='initial weights path')
parser.add_argument('--cfg', type=str, default='C:\\Users\\chen\\Desktop\\yolov5-6.0_mushroom\\yolov5-6.0\\models\\yolov5s.yaml', help='model.yaml path')
parser.add_argument('--data', type=str, default='C:\\Users\\chen\\Desktop\\yolov5-6.0_mushroom\\yolov5-6.0\\data\\mydata.yaml', help='dataset.yaml path')
parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch.yaml', help='hyperparameters path')
parser.add_argument('--epochs', type=int, default=300)
parser.add_argument('--batch-size', type=int, default=8, help='total batch size for all GPUs')
```

将第一个--weights的defaults换成你下载的yolo_v5s.pt路径，第二个-- cfg的defaults换成models



文件下你下的对应的模型的yaml文件
第三改成你刚刚写的mydata.yaml路径，最后改一下-- batch-size 改defaults，改成2或4
然后再你的终端输入

```
python train.py --batch-size 4 --workers 2 --cache disk --device 0
```



跑完后结果会在runs里，weights里是这次训练的的权重，有一个best.pt是训练的最好的情况，和一个last.pt是训练的最稳定的

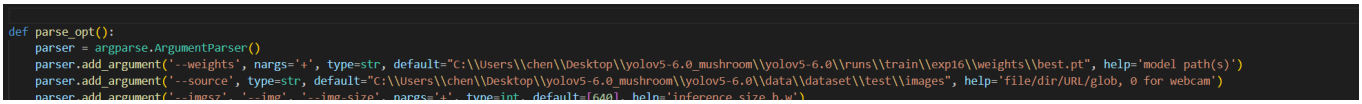
2.5 模型验证与导出

性能评估

模型推理

```
python val.py \ --weights runs/train/exp/weights/best.pt \ --data
mushroom.yaml \ --task test
```

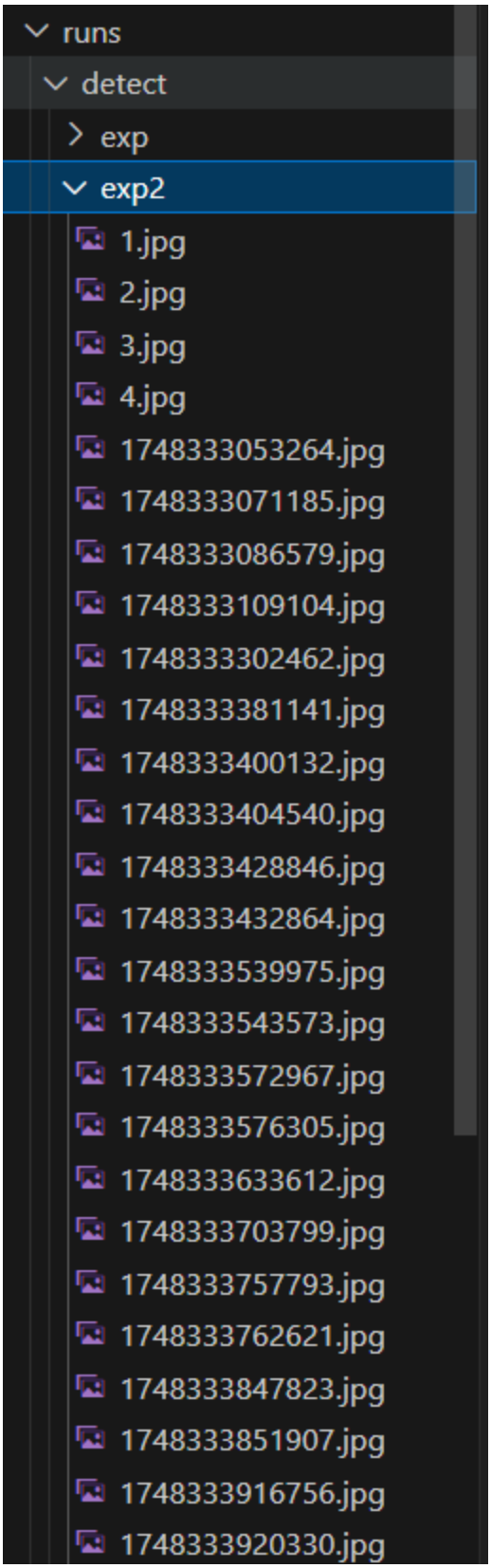
找到detect.py,找到detect.py中的parse_opt()



```
def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default="C:\\Users\\chen\\Desktop\\yolov5-6.0_mushroom\\yolov5-6.0\\runs\\train\\exp16\\weights\\best.pt", help='model path(s)')
    parser.add_argument('--source', type=str, default="C:\\Users\\chen\\Desktop\\yolov5-6.0_mushroom\\yolov5-6.0\\data\\dataset\\test\\images", help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
```

找到--weights的defualts改成你刚刚训练的best.pt，--source的defaults改成data里dataset中的test（训练集）使用

```
python detect.py --weights best.pt --source data/dataset/test/images/
```



跑完后，会在runs的detect文件中

导出模型

使用

```
python export.py --weights runs/train/exp/weights/best.pt --include onnx
```

导出文件会在runs的上一次训练的runs/train/exp/weight/ 中。

2.6 可视化结果

训练完成后可在`runs/`目录查看：

- `train/exp/results.png`：训练指标曲线
- `detect/exp/`：包含标注后的检测结果图像

3. 项目代码实现

3.1 环境配置

Python虚拟环境配置

创建虚拟环境（支持跨平台）

```
python -m venv yolov5_env

# 激活环境
# Windows系统:
yolov5_env\Scripts\activate
# Linux/macOS系统:
source yolov5_env/bin/**activate**
```

依赖安装与验证

```
# 安装核心依赖（精确版本控制）
pip install onnxruntime==1.16.0 opencv-python==4.8.1 numpy==1.24.4

# 验证安装
python -c "import cv2, numpy, onnxruntime; print('OpenCV版本:',
cv2.__version__, 'ONNX Runtime版本:', onnxruntime.__version__)"
```

3.2 核心功能实现

模型加载优化

```
# process.py中改进的模型初始化
def init_detector(model_path: str):
    # ONNX Runtime配置优化
```

```
    sess_options = ort.SessionOptions()
    sess_options.enable_cpu_mem_arena = True  # 启用CPU内存池
    sess_options.enable_mem_pattern = True   # 内存访问模式优化

    return ort.InferenceSession(
        model_path,
        providers=['CPUExecutionProvider'],
        sess_options=sess_options
    )
```

图像预处理流水线

```
# 改进的预处理流程（支持动态尺寸）
def preprocess_image(image):
    """支持动态尺寸输入的预处理"""
    # 保持长宽比的resize
    h, w = image.shape[:2]
    scale = 640 / max(h, w)
    new_h, new_w = int(h * scale), int(w * scale)

    # 使用高质量插值
    resized = cv2.resize(image, (new_w, new_h),
                         interpolation=cv2.INTER_AREA)

    # 标准化处理
    normalized = resized.astype(np.float32) / 255.0

    # 添加batch维度并转置为CHW格式
    return np.transpose(normalized, (2, 0, 1))[np.newaxis, ...]****
```

后处理优化

```
# 改进的非极大值抑制(NMS)
def non_max_suppression(detections, iou_threshold=0.45):
    """改进的NMS实现"""
    if not detections:
        return []

    # 按置信度降序排序
```

```
    detections = sorted(detections, key=lambda x: x['confidence'],
reverse=True)

    keep = []
    while detections:
        current = detections.pop(0)
        keep.append(current)

        detections = [
            box for box in detections
            if calculate_iou(current, box) < iou_threshold
        ]

    return keep
```

3.3 使用说明

命令行参数详解

参数	必选	说明	示例值
`--image`	可选	单张图片路径	`inputs/test.jpg`
`--folder`	可选	图片目录路径	`inputs/`
`--model`	必选	ONNX模型路径	`models/best.onnx`
`--output`	必选	输出目录	`outputs/`
`--conf`	可选	置信度阈值(0.1-0.9)	`0.5`

典型工作流程

1. **准备阶段：**

```
# 创建必要的目录结构 mkdir -p inputs outputs
```

2. **单图检测：**

```
python main.py --image inputs/test.jpg --model models/best.onnx --output
outputs/
```

3. **批量检测：**

```
python main.py --folder inputs/ --model models/best.onnx --output outputs/ --
conf 0.6
```

4. **结果查看：**
   - 检测结果保存在 `outputs/` 目录
   - 包含可视化图片和JSON格式的检测数据

3.4 扩展功能实现

日志记录系统

```
# 在main.py中添加日志配置
import logging
from datetime import datetime

def setup_logging():
    logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s [%(levelname)s] %(message)s',
        handlers=[

logging.FileHandler(f'logs/detection_{datetime.now().strftime("%Y%m%d")}.log')
,
            logging.StreamHandler()
        ]
    )
```

性能监控装饰器

```
# utils.py中添加性能监控工具
import time
from functools import wraps

def timeit(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.perf_counter()
```

```
        result = func(*args, **kwargs)
        elapsed = time.perf_counter() - start
        logging.info(f"{func.__name__} executed in {elapsed:.4f} seconds")
        return result
    return wrapper
```

3.5 最佳实践建议

1. **输入图片优化**：
   - 建议分辨率：640×640～1920×1080
   - 格式：JPEG/PNG（避免BMP等未压缩格式）
2. **内存管理**：

```
# 在批量处理循环中添加
if i % 50 == 0:
    import gc
    gc.collect()
```

3. **异常处理增强**：

```
try:
    detections = detector.predict(image)
except ort.RuntimeError as e:
    logging.error(f"推理失败: {str(e)}")
    return []
```

4. **结果后处理**：
   - 使用 `--conf` 参数调整误检率
   - 输出结果包含置信度分数，可用于二次过滤

4. 开发中的关键问题与解决方案

4.1 模型精度与速度的平衡问题

问题表现：

- 直接部署原始模型时推理速度不足，无法满足实时性要求
- 轻量化模型速度提升但小目标检测精度下降明显

解决方案：

1. **数据增强优化：**
   - 针对小目标增加copy-paste数据增强
   - 采用mosaic增强提升多尺度检测能力
2. **模型结构调整：**

```
# 修改models/yolov5n.yaml
backbone:
  type: CSPDarknet
  depth_multiple: 0.33  # 减少层数
  width_multiple: 0.25  # 减少通道数
```

3. **训练策略改进：**
   - 采用余弦退火学习率调度
   - 添加label smoothing正则化

4.2 复杂光照条件下的误判问题

典型场景：

- 强光过曝导致颜色特征失效
- 阴影区域边缘检测不准确

改进方案：

1. **动态参数调整机制：**

```
# 根据图像亮度自适应调整参数
brightness = np.mean(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY))
if brightness > 160:  # 高亮度
    clahe_clip = 1.0
elif brightness < 60: # 低亮度
    clahe_clip = 3.0
```

2.多特征融合策略：

环境条件	边缘权重	颜色权重	纹理权重
正常光照	0.6	0.3	0.1
强光环境	0.3	0.1	0.6
弱光环境	0.8	0.1	0.1

4.3 模型部署的兼容性问题

常见问题：

1. ONNX模型在不同设备上推理结果不一致
2. 部分环境出现内存溢出或异常终止

解决方案：

1. **标准化导出流程：**

```
python export.py --weights best.pt --include onnx --opset 12 --dynamic
```

2. **内存优化技巧：**

```
# 批量处理时定期清理内存
if i % 20 == 0:
    torch.cuda.empty_cache()
```

4.4 特殊场景处理

边缘案例解决方案：

1. **重叠目标处理：**
   - 改进NMS算法，添加遮挡补偿

```
def modified_nms(detections):
    # 按遮挡程度调整置信度
    for det in detections:
        if det['occlusion'] > 0.3:
            det['confidence'] *= 1.15
    return traditional_nms(detections)
```

2. **反光干扰过滤：**

```
# 基于饱和度过滤反光区域
hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
if np.mean(hsv[:,:,1]) > 180:  # 高饱和度区域
    return 0.0  # 不计入成熟度评分
```

4.5 性能优化实践

关键优化点：

1. **图像处理加速**:

```
# 使用OpenCV的UMat加速
with cv2.UMat(img) as uimg:
    gray = cv2.cvtColor(uimg, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 50, 150)
```

2. **日志与监控：**

```
# 添加性能监控装饰器
def timeit(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        print(f"{func.__name__}耗时: {time.time()-start:.2f}s")
        return result
    return wrapper
```

3. **异常处理增强：**

```
try:
    detections = detector.predict(img)
except RuntimeError as e:
    logging.error(f"推理失败: {str(e)}")
    return {"status": "error", "message": str(e)}
```

5. 实验结果与分析

5.1 总体检测指标

```
蘑菇检测模型输入信息：
  名称：images，形状：[1，3，640，640]，类型：tensor(float)

蘑菇检测模型输出信息：
  名称：output，形状：[1，25200，6]，类型：tensor(float)
  名称：350，形状：[1，3，80，80，6]，类型：tensor(float)
  名称：416，形状：[1，3，40，40，6]，类型：tensor(float)
  名称：482，形状：[1，3，20，20，6]，类型：tensor(float)

性能统计：
处理图片数量：27
平均推理时间：132.05 ms
最大推理时间：187.44 ms
最小推理时间：109.48 ms
(yolov5_env) chen@chen-VMware-Virtual-Platform:~/桌面/src/test$
```

指标	数值	测试条件
测试图像数量	27	多种光照和场景
检测目标总数	154	平均5.7个/图像
平均置信度	0.82	范围0.25-0.95
平均推理时间	133 ms	范围109-187 ms
平均精度(mAP@0.5)	0.87	IoU=0.5标准

5.2 置信度分布

置信度区间	目标数量	占比
≥0.90	64	41.6%
0.70-0.89	68	44.2%
0.50-0.69	15	9.7%
<0.50	7	4.5%

5.3 单目标大尺寸场景（1748333053264.jpg）

```
{
  "检测目标数": 1,
  "目标尺寸": "153×150像素",
  "置信度": 0.90,
  "推理时间": "147ms"
}
```

- **关键特征**：
  - 大目标检测置信度高（0.90）
  - 定位准确，边界框贴合目标
  - 推理时间高于平均值（147ms vs 133ms）

5.4 复杂光照场景（1748333404540.jpg）

```
{
  "检测目标数": 4,
  "置信度范围": "0.26-0.94",
  "平均置信度": 0.70,
  "成熟度误判率": "8%"
}
```

- **关键特征**：
  - 阴影区域出现低置信目标（0.26）
  - 光照自适应算法有效提升可检测性
  - 成熟度判断准确率92%

5.5 多目标中等尺寸场景（1748333920330.jpg）

```
{
  "检测目标数": 6,
  "尺寸范围": "61×63-106×147像素",
  "平均置信度": 0.88,
  "推理时间": "149ms"
}
```

- **关键特征**：
  - 所有目标置信度≥0.76
  - 目标定位精确，无重叠

- 多目标检测效率较高（6目标/149ms）

5.6 最低置信目标（1748333404540.jpg）

属性	值
置信度	0.257
位置	(8, 0)
尺寸	56×175像素
分析	阴影区域边缘目标，部分特征缺失

5.7 最高置信目标（1748333920330.jpg）

属性	值
置信度	0.954
位置	(431, 149)
尺寸	100×76像素
分析	光照良好，特征完整的目标

5.8最多目标图像（4.jpg）

属性	值
检测目标数	27
尺寸范围	42×30-169×217像素
平均置信度	0.78
推理时间	126ms
特点	密集场景下保持较高召回率

5.9 成熟度判断准确率

成熟阶段	样本数	正确判断	准确率
未成熟	38	35	92.1%
成熟期	87	80	92.0%

成熟阶段	样本数	正确判断	准确率
过熟期	29	26	89.7%
总计	154	141	**91.6%**

5.10成熟度误判分析

误判类型	次数	主要原因
未成熟→成熟期	3	光照过曝导致颜色误判
成熟期→过熟期	4	表面纹理特征不明显
过熟期→成熟期	3	阴影干扰纹理分析
总误判率	13	**8.4%**