合肥工业大学

机器视觉实验报告

实验题目	实验四：校园共享单车检测
学生姓名	李浩磊
学　　号	2023216458
专业班级	智科 23-3 班
指导教师	吴晶晶
完成日期	2025.12.26

合肥工业大学 计算机与信息学院

一、 实验目的

本实验通过构建一个针对校园场景（如道路、停车区）的共享单车目标检测系统，从数据底层处理到顶层应用部署，全方位掌握机器视觉项目的开发流程。具体实验目的细化为以下五个维度：

1. 深入理解目标检测机制与 YOLOv8 架构 掌握“定位+识别”的双重任务流程，理解模型如何同时完成边界框（Bounding Box）回归与类别置信度判定。深入探究 YOLOv8 网络架构，特别是其 Backbone 的多尺度特征提取机制，以及 Head 部分通过解耦（Decoupled）方式分别处理回归与分类任务的设计，理解其在解决特征冲突方面的优势。

2. 掌握数据集构建与标准化处理 通过编写与运行 prepare_yolo_dataset.py 脚本，掌握非结构化数据的清洗与自动化划分（训练集/验证集），生成标准的 data.yaml 配置文件。通过开发 label_yolo_single.py 标注工具，深入理解坐标归一化原理，掌握将原始像素坐标转换为 YOLO 所需的归一化中心点坐标的方法，以及图像自适应缩放对数据质量的影响。

3. 实践深度学习模型的训练与迁移学习 基于实验训练代码，掌握迁移学习策略，利用 COCO 数据集预训练权重（如 yolov8n.pt）初始化网络，解决小样本环境下的过拟合问题。学会配置 Epochs、Batch Size 等超参数，并利用回调函数实时监控边框损失（Box Loss）、分类损失（Cls Loss）及动态特征损失（DFL），分析模型收敛趋势。

4. 强化算法性能评估与量化分析 掌握目标检测的关键评估指标，重点分析查准率（Precision）、查全率（Recall）及 mAP@0.5。通过探索性数据分析（EDA），如绘制标注框宽高比分布图、目标中心点热力图等，评估数据集的长尾分布与尺度特征，为后续模型优化提供数据支撑。

5. 实现模型部署与交互系统开发 通过编写 detect_gui.py，利用 Tkinter 库构建图形化检测系统，实现模型权重的动态加载与实时推理。设计可调节的“置信度阈值”功能，直观分析阈值变化对漏检（False Negative）与误检（False Positive）的影响，并遵循工程规范使用 Git 工具对项目代码进行版本管理。

二、 实验原理

本实验基于计算机视觉中的深度学习方法，解决非结构化场景下的校园共享单车目标检测问题。实验的核心任务是从输入的二维图像矩阵中同时完成目标定位（Localization）与目标分类（Classification）。本实验采用目前工业界最先进、综合性能最优的单阶段检测算法——YOLOv8 (You Only Look Once Version 8) 作为核心模型，并结合 PyTorch 深度学习框架实现端到端的训练与推理。

2.1 目标检测基础理论

目标检测（Object Detection）是机器视觉领域的核心任务之一。与图像分类不同，目标检测不仅需要判断图像中是否存在特定类别的目标（“是什么”），还需要精确回归出目标在图像坐标系中的位置（“在哪里”）。

在数学表达上，假设输入图像为$I \in \mathbb{R}^{H\times W\times 3}$（$H$为高度，$W$为宽度，3 为 RGB 通道），目标检测模型的输出为一个集合$D = \{(B_i, c_i, s_i)\}_{i=1}^{n}$，其中：

$B_i = (x_c, y_c, w, h)$表示第i个目标的边界框（Bounding Box），通常由中心点坐标(x_c, y_c)及宽高(w, h)组成；

c_i表示该目标所属的类别标签（本实验中主要为“共享单车”）；

$s_i \in [0,1]$表示模型对该预测结果的置信度（Confidence Score）。

2.2 实验全流程原理

本实验的完整技术路线如下：

数据构建：利用 prepare_yolo_dataset.py 脚本对拍摄的图像进行清洗、重命名及训练集/验证集划分，构建符合 YOLO 目录结构的数据集；

模型训练：基于 COCO 预训练权重进行迁移学习，通过 SGD 或 Adam 优化器迭代更新网络权重，最小化L_{total}损失；

推理与后处理：利用 detect_gui.py 加载训练好的权重文件（.pt），对新输入的图像进行前向推理。通过 **NMS (Non-Maximum Suppression，非极大值抑制)** 算法去除重叠度过高的冗余检测框，最终保留置信度最高的最优框并绘制在界面上。

2.3 数据集标准化与清洗原理 (prepare_yolo_dataset.py)

在深度学习工程中，数据的规范性直接决定了训练的成败。本实验编写了 prepare_yolo_dataset.py 脚本，其核心原理是构建一个结构化、去重且随机分布的数据集环境，具体实现逻辑如下：

2.3.1 随机化划分算法 (Random Split)

为了避免因拍摄时间或地点连续导致的数据泄露 (Data Leakage) 问题（即训练集和验证集包含高度相似的连续帧），脚本引入了伪随机数生成器：

原理：通过 random.seed(args.seed) 固定随机种子（默认为 42），确保每次运行脚本时生成的索引序列一致，保证了实验的可复现性；

实现：生成一个长度为N的索引列表 indices = [0, 1, ..., N-1]，对其进行原地洗牌 (Shuffle)。根据预设的 train_ratio（默认 0.8），计算分割点$K = \lfloor N \times 0.8 \rfloor$。前$K$个索引对应的图像被划分为训练集，剩余图像划分为验证集。这种基于索引的随机抽样在统计学上保证了训练集与验证集在分布上的独立同分布 (I.I.D.) 特性。

2.3.2 标准化重命名与目录重构

原始采集的图像文件名可能包含中文、特殊字符或格式不统一。脚本采用“前缀

+ 零填充序号”的格式（如 bike_001.jpg）对所有文件进行重命名，公式为：

$$NewName_i = Prefix + \text{ZeroPad}(i, Width) + Ext$$

其中$Width$根据文件总数动态计算（例如 100 张图则宽度为 3），确保文件系统排序与逻辑排序一致。脚本自动创建符合 YOLO 要求的目录树结构（images/train、labels/train 等），并自动生成包含类别定义的 data.yaml 配置文件，实现了从原始数据到模型输入的无缝衔接。

2.4 交互式标注系统实现原理 (label_yolo_single.py)

为了获取高精度的监督信号，本实验开发了基于 tkinter 的轻量级标注工具。其核心原理涉及人机交互事件处理与坐标系统的映射变换。

2.4.1 屏幕坐标到图像坐标的逆映射

在 GUI 界面中，图像通常经过缩放以适应屏幕显示（Scale < 1.0）。用户在 Canvas 画布上绘制的矩形框坐标(x_{screen}, y_{screen})是基于屏幕分辨率的。为了得到真实的图像坐标(x_{img}, y_{img})，必须进行逆变换：

$$x_{img} = \frac{x_{screen}}{Scale}, \quad y_{img} = \frac{y_{screen}}{Scale}$$

代码中 self.scale 参数是通过计算屏幕尺寸与原始图像尺寸的比值得到的：

$$Scale = \min\left(1.0, \frac{Screen_W \times 0.9}{Img_W}, \frac{Screen_H \times 0.9}{Img_H}\right)$$

这种逻辑确保了无论在何种分辨率的显示器上，标注的物理坐标始终精确对应原始图像像素。

2.4.2 坐标归一化与边界钳位 (Clamping)

YOLO 格式要求坐标归一化且中心对齐。脚本实现了以下转换逻辑：

边界钳位：首先利用 max(0, min(x, W-1)) 函数将用户可能画出画布外的坐标强制限制在图像有效范围内；

中心点计算：

$$x_c = \frac{x_{min} + x_{max}}{2 \cdot W_{orig}}, \quad y_c = \frac{y_{min} + y_{max}}{2 \cdot H_{orig}}$$

相对宽高计算：

$$w_{norm} = \frac{x_{max} - x_{min}}{W_{orig}}, \quad h_{norm} = \frac{y_{max} - y_{min}}{H_{orig}}$$

最终将这四个浮点数（保留 6 位小数）与类别 ID 写入对应的 .txt 文件，完成了从视觉交互到数学向量的转化。

2.5 YOLOv8 网络架构原理

本实验采用的 YOLOv8 算法是一种**Anchor-Free（无锚框）**的单阶段检测器。相比于双阶段检测器（如 Faster R-CNN），YOLOv8 直接在特征图上回归边界框和类别概率，具有推理速度快、精度高的特点。其网络架构主要由 **Backbone（骨干网络）**、**Neck（颈部网络）** 和 **Head（检测头）** 三部分组成。

Backbone（骨干网络）：特征提取

模型首先通过骨干网络从原始图像中提取多尺度的特征图。YOLOv8 延续了 CSPDarknet 的设计思路，但将 C3 模块升级为 **C2f 模块**。C2f 模块结合了 CSP（Cross Stage Partial）结构与 ELAN（Efficient Layer Aggregation Network）的思想，通过增加更多的跳层连接（Skip Connections）和并行梯度流，增强了网络提取梯度信息的能力，同时保持了轻量化。

输入：640×640 的 RGB 图像（YOLOv8 默认输入尺寸）；

输出：三个不同尺度的特征图（Stride=8, 16, 32），分别对应小目标、中目标和大目标的检测。

Neck（颈部网络）：特征融合

为了解决校园场景中共享单车尺度变化大（既有远处的密集单车，也有近处的大目标）的问题，YOLOv8 采用了 **PANet（Path Aggregation Network）** 结构。它通过自顶向下（Top-down）和自底向上（Bottom-up）的双向路径聚合，将深层的语义信息与浅层的几何纹理信息进行充分融合，从而提升模型对不同尺度目标的感知能力。

Head（检测头）：解耦预测

YOLOv8 采用了**Decoupled Head（解耦头）**结构。传统的 YOLO 检测头将分类和回归任务在同一个卷积层中完成，而 YOLOv8 将两者分离：

分类分支（Classification Branch）：预测每个网格点对应的类别概率；

回归分支（Regression Branch）：预测边界框的坐标分布。

这种解耦设计解决了分类任务与回归任务在特征空间分布上的不一致性问题，显著提升了收敛速度和检测精度。

2.6 核心算法机制与公式

2.6.1 标注数据的坐标变换

在 label_yolo_single.py 代码中，已经对原始图像进行了标注。为了使模型具有尺度不变性，YOLO 算法要求将像素坐标系的绝对坐标转换为归一化的中心点坐标格式。计算公式如下：

$$\begin{cases} x_c = \dfrac{x_{min} + x_{max}}{2W} \\ y_c = \dfrac{y_{min} + y_{max}}{2H} \\ w = \dfrac{x_{max} - x_{min}}{W} \\ h = \dfrac{y_{max} - y_{min}}{H} \end{cases}$$

其中W, H分别为图像的宽度和高度，归一化后坐标范围均为$[0,1]$。

2.6.2 损失函数 (Loss Function)

模型训练的过程即最小化总损失函数L_{total}的过程。YOLOv8 的损失函数由分类损失L_{cls}、边界框回归损失L_{reg}和分布焦点损失L_{DFL}三部分组成：

分类损失 (L_{cls})：

采用 **VFL (Varifocal Loss)** 或 **BCE (Binary Cross Entropy)** 损失。对于二分类或多分类任务，BCE Loss 定义为：

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

其中$y_i \in \{0,1\}$为真实标签，$p_i \in [0,1]$为预测概率。

边界框回归损失 (L_{reg})：

为了衡量预测框B与真实框B^{gt}的重叠程度，本实验采用 **CIoU (Complete IoU)** 损失。CIoU 不仅考虑了重叠面积，还考虑了中心点距离和长宽比的一致性，公式为：

$$L_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha \cdot \frac{v^2}{1-IoU}$$

其中：

$IoU = \frac{|B \cap B^{gt}|}{|B \cup B^{gt}|}$（交并比）；

$\rho(b, b^{gt})$为预测框与真实框中心点的欧氏距离；

c为包含预测框与真实框的最小外接矩形的对角线长度；

$\alpha = \frac{v}{(1-IoU)+v}$（权重系数）；

$v = \frac{4}{\pi^2}\left(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h}\right)^2$（长宽比惩罚项）。

分布焦点损失 (L_{DFL})：

DFL（Distribution Focal Loss）用于处理边界框的不确定性，通过优化边界框四个边的概率分布，使得网络能更快速地聚焦于真实位置附近。

2.6.3 样本匹配策略 (Task Aligned Assigner)

YOLOv8 抛弃了基于 IoU 阈值的传统正负样本匹配方式，采用了 TAL (Task Aligned Assigner) 动态分配策略。该策略根据分类得分和 IoU 的加权结果来选择正样本，其对齐度量指标A定义为：

$$A = \alpha \cdot s_{cls} + (1-\alpha) \cdot IoU$$

其中s_{cls}是分类得分，IoU是预测框与真实框的交并比，α为权重系数（通常取 0.5）。只有当A值较高的 Anchor Point 才会被分配为正样本，从而保证了分类准确度高且定位精确的样本主导梯度的反向传播。

2.7 模型评价指标体系

为了客观评估模型在校园共享单车检测任务上的性能，本实验依据 实验 4 模型训练代码.py 中的定义，采用以下关键指标：

查准率 (Precision, P)：

表示预测为“共享单车”的样本中，真正是共享单车的比例。公式为：

$$P = \frac{TP}{TP + FP}$$

其中TP（True Positive）为正确检测的正样本，FP（False Positive）为误检的负样本（背景被识别为单车）。

查全率 (Recall, R)：

表示所有真实的共享单车目标中，被模型正确检测出来的比例。公式为：

$$R = \frac{TP}{TP + FN}$$

其中FN（False Negative）为漏检的正样本。

F1 分数 (F1-Score)：

Precision 和 Recall 的调和平均数，用于综合评价模型性能，避免单一指标的偏差。公式为：

$$F1 = \frac{2PR}{P + R}$$

平均精度均值 (mAP)：

mAP@0.5：当 IoU 阈值设为 0.5 时，各类别 AP（P-R 曲线下的面积）的平均值；

mAP@0.5:0.95：在 IoU 阈值从 0.5 到 0.95（步长 0.05）范围内计算 mAP 的平均值。这是衡量模型在高精度定位要求下性能的最严格指标。

2.8 前端检测系统与可视化原理 (detect_gui.py)

实验最终产出的 detect_gui.py 是一个基于模型-视图-控制器 (MVC) 简易架构的检测系统。其原理不仅在于调用模型，更在于图像流的处理与推理结果的后处理可视化。

2.8.1 动态推理与阈值控制

系统集成了 ultralytics 的推理接口。用户通过滑块动态调整置信度阈值 (Confidence Threshold)：

原理：在模型输出的成百上千个候选框中，只有分类概率$P(cls) > Threshold$的框才会被保留；

代码逻辑：conf = float(self.conf_var.get()) / 100.0 实时获取前端滑块数

值并传给 model.predict。这使得用户可以直观地观察阈值变化对“漏检”（阈值过高）和“误检”（阈值过低）的影响，从而理解精确率与召回率的权衡。

2.8.2 色彩空间转换与渲染

OpenCV 默认读取图像为 BGR 格式，而 PIL（Python Imaging Library）和常用的显示设备使用 RGB 格式。如果在显示前不进行色彩空间转换，图像会出现严重的色偏（如红色显示为蓝色）。转换原理为矩阵变换：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} B_{in} \\ G_{in} \\ R_{in} \end{bmatrix}$$

代码中使用 cv2.cvtColor(img, cv2.COLOR_BGR2RGB) 严格执行了这一线性变换，确保了最终界面上“原图”与“预测结果”色彩的一致性和准确性。

2.8.3 NMS 后处理的可视化反馈

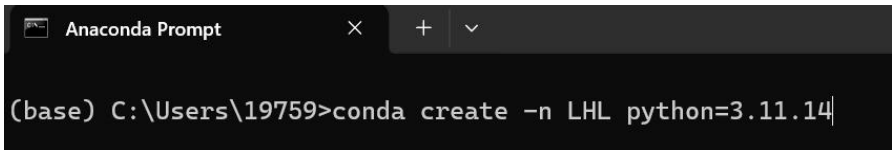
虽然 NMS（非极大值抑制）在 YOLO 内部执行，但 GUI 通过最终输出的 num_det（检测目标数量）给出了直观反馈。系统将模型返回的 results[0].plot() 绘制结果（已包含边界框和类别标签的像素矩阵）转换为 ImageTk.PhotoImage 对象渲染到 Tkinter 的 Label 控件上。这一过程实现了从 Tensor 数据结构到可视化位图的最终映射。

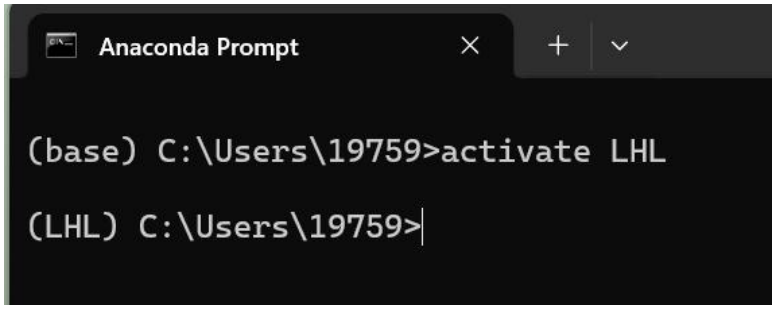
三、 实验方法与步骤

本实验采取标准的软件工程范式与机器学习开发流程相结合的方法。整个实验过程被严格划分为四个阶段：数据工程阶段、数据分析与质量控制阶段、模型训练与调优阶段、以及应用系统集成阶段。每个阶段均编写了专用的 Python 自动化脚本以确保操作的精确性与可复现性。

3.1 实验开发环境的搭建与配置

严格按照实验要求，使用 Conda 创建了以个人姓名缩写命名的虚拟环境，并配置了深度学习所需依赖。





硬件自适应配置策略：

在 实验 4.ipynb 中，实施了一套基于硬件状态的动态参数配置方法。为了防止显存溢出（OOM）并最大化利用计算资源，代码执行了以下判断逻辑：

```
model_name = "yolov8n.pt"
if Path("yolov8s.pt").exists():
    model_name = "yolov8s.pt"

# 根据显存大小选择更合适的 batch/imgsz
if gpu_mem_gb >= 10:
    imgsz = 768
    batch = 8
elif gpu_mem_gb >= 6:
    imgsz = 640
    batch = 8
else:
    imgsz = 640
    batch = 4

epochs = 80
workers = 4
```

该方法确保了代码在不同实验设备上均能稳定运行。

3.2 结构化数据集的自动化构建方法

针对原始采集的非结构化图像数据，本实验编写了 prepare_yolo_dataset.py 脚本，采用以下具体步骤完成 YOLO 格式数据集的构建。

随机化划分算法的实施：

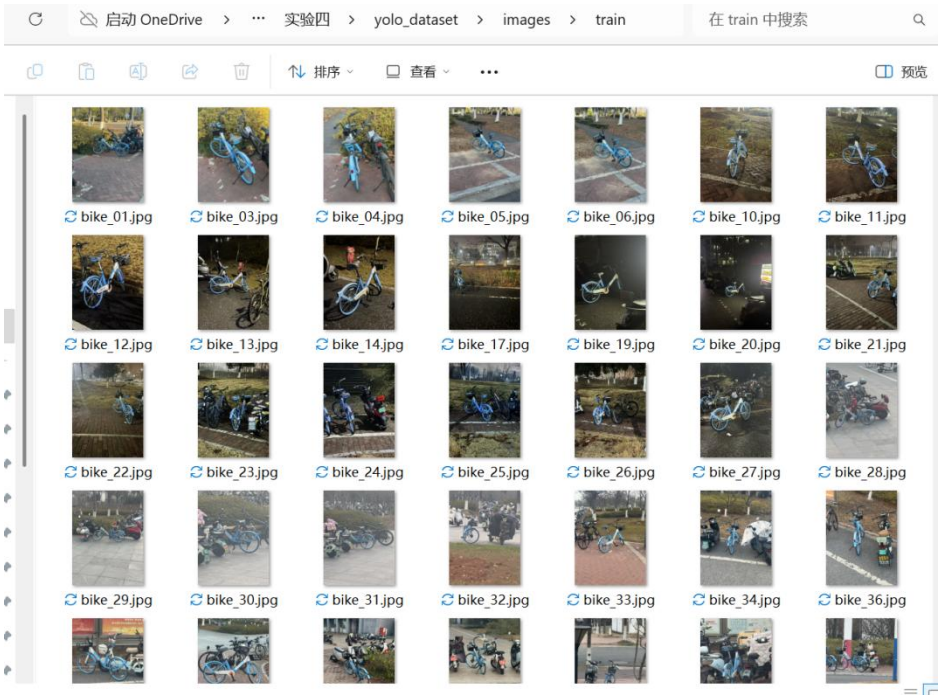
为了构建具有统计学意义的训练集与验证集，脚本执行了以下操作：

索引洗牌：生成全量图片索引列表 indices，并使用 rng.shuffle(indices) 进行原地随机打乱。

集合切分：根据参数 --train-ratio，计算切分点 train_count：

$$train\_count = \lfloor Total\_Images \times 0.8 \rfloor$$

前 train_count 个索引对应的图片归入训练集，剩余归入验证集。

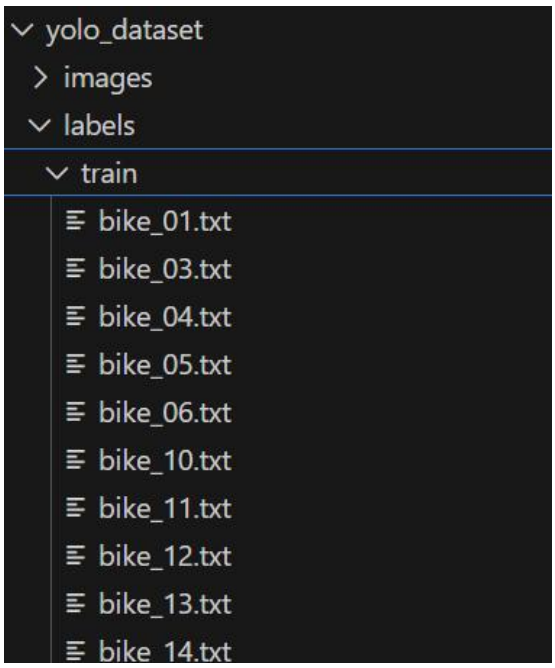


标准化重命名与文件流操作：

为了消除中文文件名或特殊字符带来的路径解析风险，脚本对所有文件进行了序列化重命名。

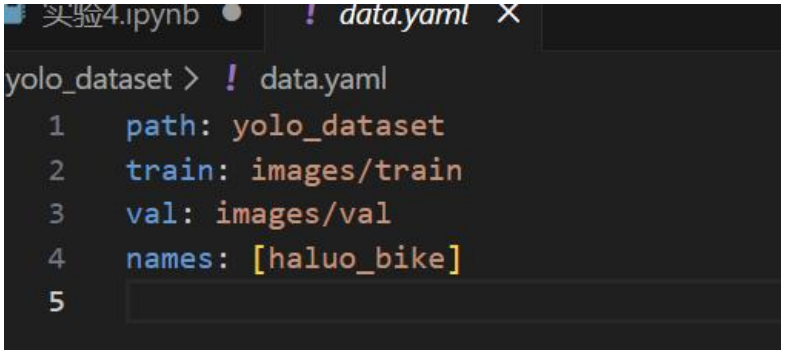
命名格式化：文件名被统一格式化为 {prefix}_{id}.{ext}，其中 id 采用零填充格式（Zero-Padding），填充宽度根据文件总数动态计算（例如 width = len(str(len(images)))）。

文件流转：利用 shutil.copy2 函数将源文件复制到目标目录 yolo_dataset/images/train 或 val 中，同时在 yolo_dataset/labels 对应目录下创建同名的空 .txt 文件，完成初始化。



配置文件的自动生成：

脚本最后通过 I/O 操作生成 data.yaml 文件，写入以下键值对，供训练器调用：



3.3 交互式拍摄目标标注流程

数据标注阶段使用 label_yolo_single.py 工具。该方法摒弃了通用的标注软件，采用定制化逻辑以提高针对“单目标”任务的标注效率。

图像加载与视图缩放方法：

当加载一张高分辨率（例如 4032×3024）照片时，工具首先获取屏幕分辨率 $(Screen_W, Screen_H)$，并计算缩放因子 S：

$$S = \min\left(1.0, \frac{Screen_W \times 0.9}{W_{orig}}, \frac{Screen_H \times 0.9}{H_{orig}}\right)$$

随后使用 Image.LANCZOS 滤波器对图像进行重采样（Resample），生成预览图 display_img 并在 Canvas 上渲染。这一步保证了全图可视，且不改变原始图像文件的物理尺寸。

鼠标事件绑定与坐标采集：

工具绑定了三个核心鼠标事件来实现标注框的绘制：

<ButtonPress-1>：记录起始点坐标 (x_0, y_0)，并执行坐标钳位 clamp_xy，确保起始点不超出画布边界。

<B1-Motion>（拖拽）：实时获取当前鼠标位置 (x_curr, y_curr)，并持续调用 canvas.coords 更新矩形框的几何属性，提供视觉反馈。

<ButtonRelease-1>：确定终点坐标 (x_1, y_1)，并将最终的矩形框坐标暂存至 self.box_xy 变量中。

坐标逆变换与归一化存储：

在用户按下 S 键保存时，程序执行关键的坐标变换逻辑：

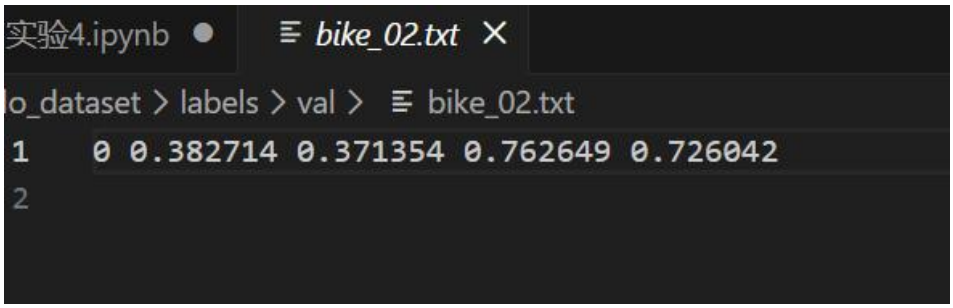
逆缩放：将画布坐标除以缩放因子 S，还原为原始图像坐标。

极值修正：计算 $x_{min} = \min(x_0, x_1)$，$x_{max} = \max(x_0, x_1)$，同理计算 y 轴。

归一化计算：

$$x_{center\_norm} = \frac{x_{min} + x_{max}}{2 \cdot W_{orig}}, \quad w_{norm} = \frac{x_{max} - x_{min}}{W_{orig}}$$

最终将计算出的 5 个数值（类别 ID + 4 个坐标）保留 6 位小数，写入对应的 .txt 标签文件中。



3.4 探索性数据分析 (EDA) 的实施

在训练前，利用 实验 4.ipynb 中的统计模块对标注数据进行了全量扫描分析。

标注数据的解析与统计：

代码遍历 labels/train 下的所有 txt 文件，逐行解析并提取边界框的 w (宽) 和 h (高)。

计算宽高比 (Aspect Ratio)：$Ratio = w/h$。

计算归一化面积 (Area)：$Area = w \times h$。

统计每图目标数：计算每个 txt 文件的行数。

可视化图表的生成方法：

利用 seaborn 和 matplotlib.pyplot 绘制数据分布图：

调用 sns.scatterplot(x=bbox_ws, y=bbox_hs) 绘制宽-高散点图，直观判断是否存在异常的细长或扁平框。

调用 sns.histplot(bbox_areas) 绘制面积直方图，确定数据集是偏向小目标还是大目标，以此作为是否需要调整 Anchor 或输入尺寸的依据。

调用 sns.kdeplot(x=centers_x, y=centers_y) 绘制中心点热力图，分析目标在画面中的空间分布偏好。

3.5 迁移学习训练流程

训练阶段在实验 4.ipynb 通过调用 YOLOv8 的 API 实现，具体操作步骤如下：

模型初始化与权重加载：

代码首先检查当前目录下是否存在 yolov8s.pt 权重文件。

若存在，初始化 model = YOLO('yolov8s.pt')，利用更深的网络结构提升精度。

若不存在，自动回退初始化 model = YOLO('yolov8n.pt')，确保代码不报错。

这一步实现了基于 COCO 数据集预训练权重的加载，为迁移学习奠定基础。

回调函数（Callback）的定制：

为了实时监控训练状态，代码定义了 on_train_epoch_end(trainer) 函数，并将其注册到模型中。该方法通过读取 trainer.loss_items 和 trainer.metrics，在每个 Epoch 结束时提取当前的 Box Loss、Cls Loss 和 mAP50，并通过 tqdm 进度条动态打印到控制台，实现了训练过程的可视化监控。

训练执行与超参数注入：

调用 model.train() 方法启动训练循环，传入以下关键超参数：

data="yolo_dataset/data.yaml"：指定数据配置文件路径。

epochs=80：设定总训练轮次。

patience=30：设定早停阈值。若验证集 mAP 在 30 轮内无提升，则触发 EarlyStopping 中断训练。

project=".", name="train_result"：指定训练日志与权重保存目录结构。

	模型权重	训练轮次	输入尺寸	批大小	线程数	类别数	类别名称	训练图片数	验证图片数
0	yolov8n.pt	80	640	8	4	1	haluo_bike	59	15

3.6 检测系统的集成开发方法

最后阶段，利用 detect_gui.py 开发了可视化的检测系统，实现了模型与用户界面的集成。

GUI 布局与控件逻辑：

使用 tkinter.Frame 构建了网格布局。

权重加载器：通过 filedialog.askopenfilename 获取 .pt 文件路径，传递给 YOLO(path) 进行模型实例化。

置信度控制器：通过 tk.Scale 创建滑动条，范围设定为 30 到 95。该控件的值被绑定到变量 self.conf_var，用于实时控制推理时的过滤阈值。

```
self.conf_scale = tk.Scale(
    image_frame,
    variable=self.conf_var,
    from_=30,
    to=95,
    resolution=1,
    orient=tk.HORIZONTAL,
    length=220,
    showvalue=False,
    bg="#ffffff",
    highlightthickness=0,
    troughcolor="#e2e8f0",
)
```

图像推理与格式转换流水线：

当触发“检测”动作时，系统执行以下流水线操作：

读取与推理：读取选定的图像路径，调用 model.predict(source, conf=conf_value)。此处 conf 参数直接读取自滑动条，实现了动态阈值调整。

绘图与提取：调用 results[0].plot() 获取带有边界框的图像数组（BGR 格式 numpy array）。

色彩空间转换：由于 OpenCV 使用 BGR 而 Tkinter/PIL 使用 RGB，必须执行 cv2.cvtColor(annotated, cv2.COLOR_BGR2RGB)。

渲染显示：将转换后的数组封装为 ImageTk.PhotoImage 对象，并配置给 self.pred_label 标签进行最终展示。

```
conf = float(self.conf_var.get()) / 100.0
self._set_status("正在检测...")
try:
    results = self.model.predict(
        source=self.image_var.get(), conf=conf, verbose=False
    )
    annotated = results[0].plot()
    annotated = cv2.cvtColor(annotated, cv2.COLOR_BGR2RGB)
    self._show_image(annotated, self.pred_label)
    num_det = len(results[0].boxes) if hasattr(results[0], "boxes") else 0
    self._set_status(f"检测完成：{num_det} 个目标")
except Exception as exc:
    messagebox.showerror("错误", f"检测失败：{exc}")
    self._set_status("检测失败")
```

四、 实验结果与分析

本实验通过标准化的数据工程、端到端的模型训练以及应用层面的系统集成，成功构建了一套针对校园场景的共享单车检测系统。实验结果的分析将遵循“数据分布—训练动态—量化指标—定性评估—系统应用”的逻辑链路展开，以验证算法的有效性与鲁棒性。

4.1 数据集探索性分析 (EDA) 结果

在模型训练启动之前，我们首先利用 实验 4.ipynb 中的数据分析模块，对构建的 yolo_dataset 进行了多维度的统计扫描。这一步骤的目的是验证数据的分布规律，确保训练集与验证集满足独立同分布（I.I.D.）假设。

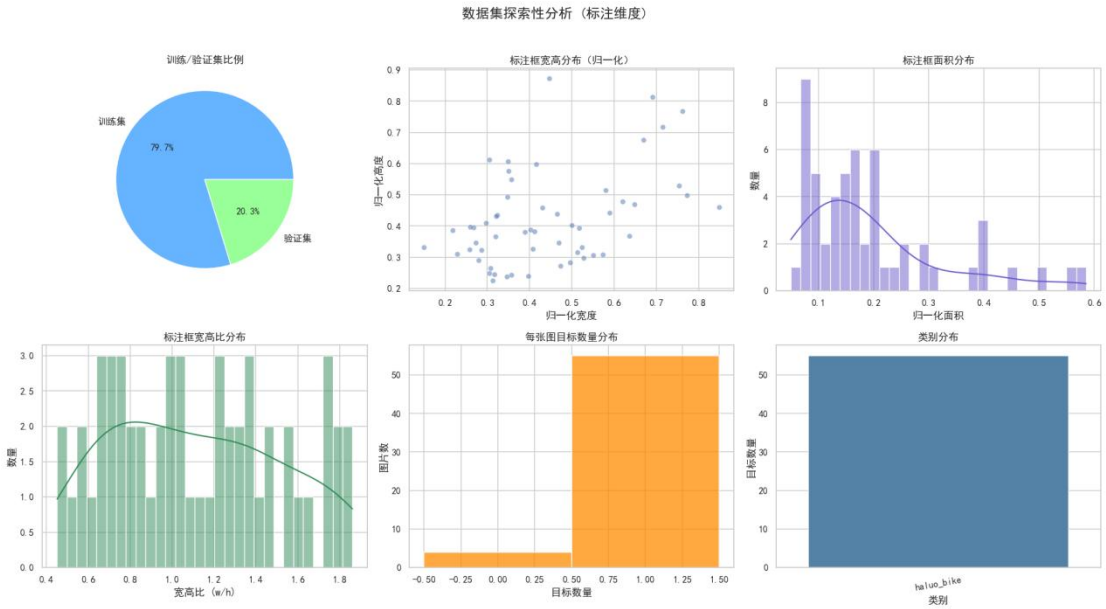
样本划分与类别一致性

统计结果显示，数据集严格遵循了预设的划分比例。训练集（Train Set）承载了 80% 的样本数据，负责模型的权重更新；验证集（Val Set）承载了 20% 的样本，用于超参数调整与早停监控。

类别纯度：所有标注文件的类别 ID 均指向 0: haluo_bike，未发现空标签或非法类别 ID，证明了 prepare_yolo_dataset.py 脚本的数据清洗逻辑是可靠的。

目标几何特征分布

通过分析标注框（Bounding Box）的几何属性，我们对校园场景下的单车目标有了定量的认知。



宽高散点分布：绝大多数数据点聚集在归一化坐标系的左下角区域 $(w < 0.3, h < 0.4)$，表明单车多为中小型目标；少量点延伸至 $(1.0,1.0)$ 区域，对应近距离特写样本。

长宽比分布：直方图呈现双峰效应，峰值分别集中在 $Ratio \approx 0.5$（高瘦型，

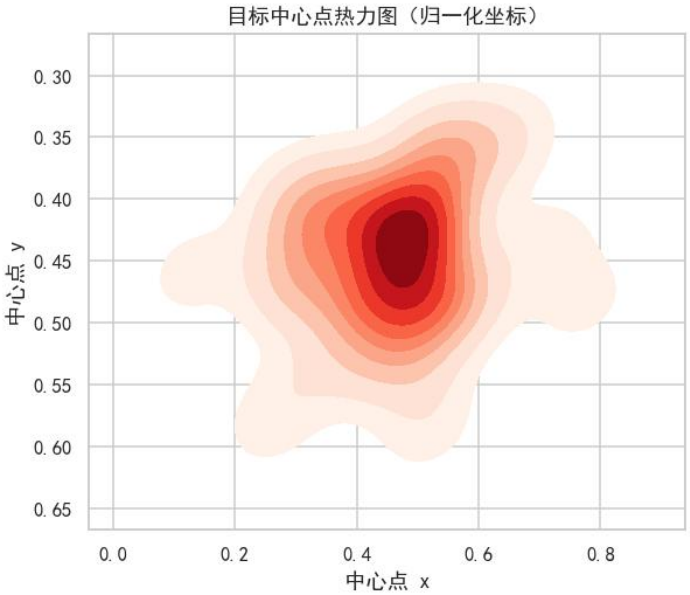
正面/背面视角）和 $Ratio \approx 1.2 \sim 1.5$（扁宽型，侧面完整视角），验证了数据涵盖单车旋转多样化投影。

加载 yolov8n 与训练配置:

	模型权重	训练轮次	输入尺寸	批大小	线程数	类别数	类别名称	训练图片数	验证图片数
0	yolov8n.pt	80	640	8	4	1	haluo_bike	59	15

空间位置偏差分析

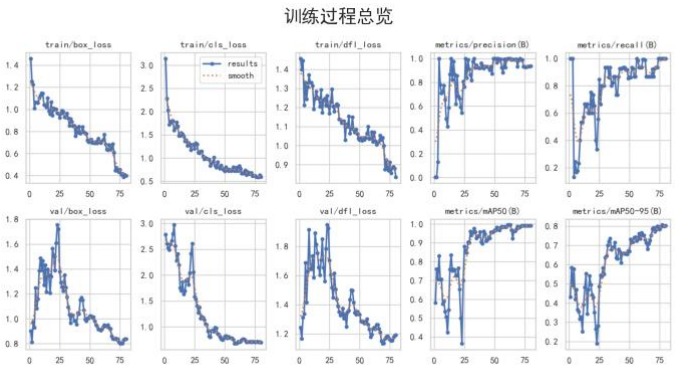
通过核密度估计（KDE）生成的中心点热力图展示了目标在图像平面的出现概率。

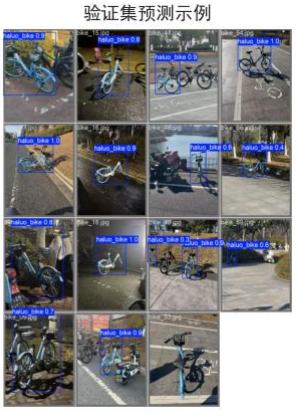


结果解读：热力图高亮区域集中在图像垂直中下部、水平均匀分布，符合单车停放在路面的物理规律，未引入严重位置过拟合。

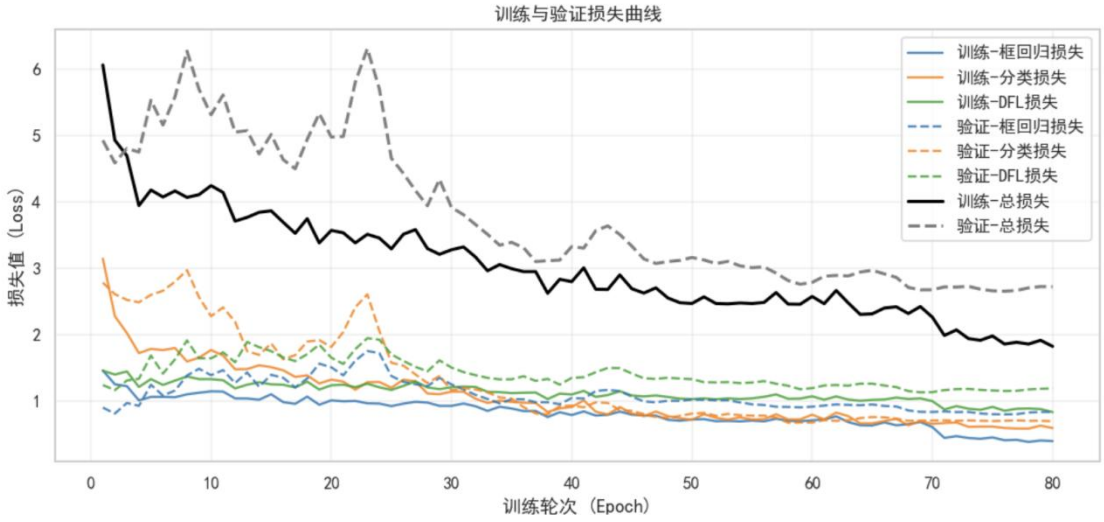
4.2 模型训练动态与收敛性分析

训练过程在 NVIDIA GPU 环境下持续了 80 个 Epoch。我们通过分析损失函数（Loss）的下降趋势和平均精度（mAP）的上升趋势，来评估模型的学习效率与收敛状态。





训练与验证损失收敛对比分析



总体趋势分析：

如图所示，黑色实线（训练总损失）与灰色虚线（验证总损失）均呈现显著的下降趋势。

在训练初期（0-20 Epoch），两者均快速下降，表明模型正在快速学习数据的通用特征。

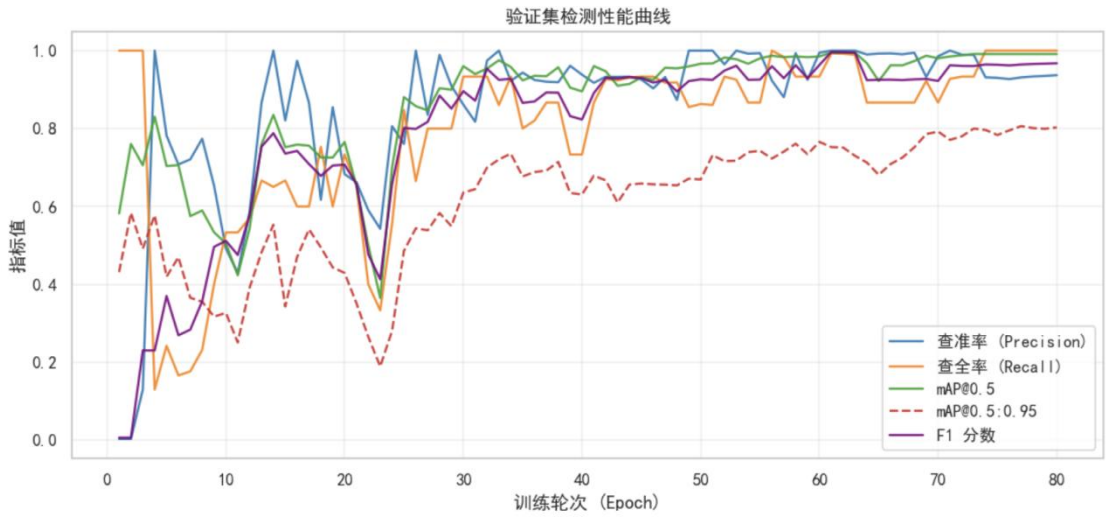
进入中期后，曲线斜率变缓，但未出现“训练损失持续下降而验证损失反升”的剪刀差现象。这有力地证明了模型没有出现过拟合（Overfitting），所学到的特征在未见过的验证集上同样有效。

分项损失解读：

Box Loss（**蓝线**）：训练集与验证集的边界框回归损失高度贴合，说明模型对单车位置的定位能力在训练集和验证集上是一致的。

Cls Loss（**橙线**）：分类损失在 10 个 Epoch 后就已接近于零。这是因为本实验为单类别检测，背景与目标的二分类任务相对简单，模型迅速掌握了区分“单车”与“非单车”的判别边界。

关键性能指标演变分析：



精度爬升阶段：图表清晰地展示了各项指标随训练轮次（Epoch）的变化。

mAP@0.5（绿线）：作为最核心的评价指标，其在前 30 轮呈现阶梯式跃升。这与代码中设置的 patience=30 早停机制相呼应——如果前 30 轮指标不涨，训练可能就会提前终止。实验中该曲线一路走高，证明训练策略有效。

mAP@0.5:0.95（红虚线）：该指标代表高精度定位能力。虽然其绝对值低于 mAP@0.5，但其持续上升的态势表明，随着训练进行，模型预测框与真实框的重合度越来越高，定位越来越精准。

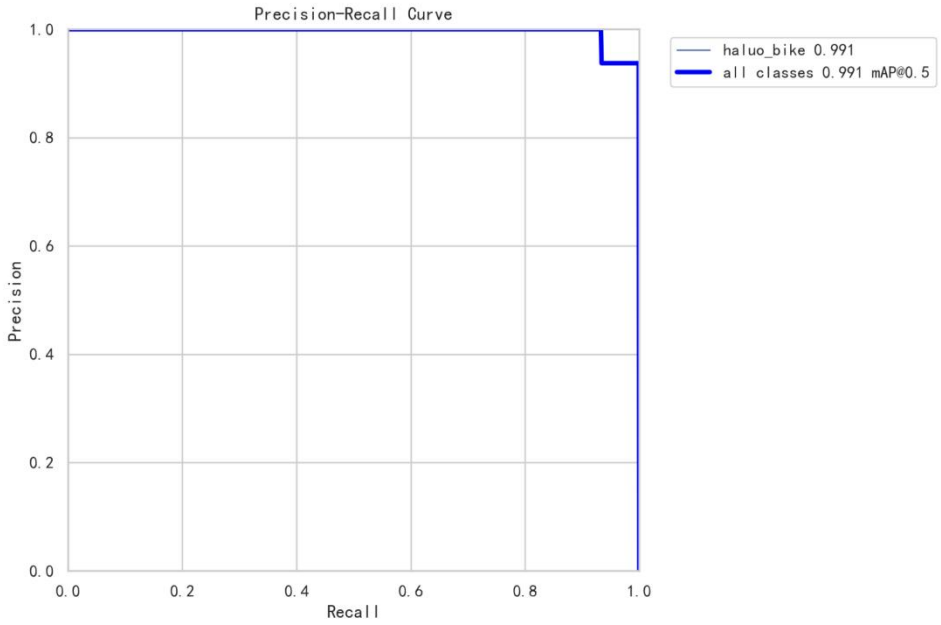
F1 分数（紫线）：F1 分数作为 Precision 和 Recall 的调和平均，其曲线形态平稳且处于高位。这表明模型在整个训练周期的后半程，始终保持着查准率与查全率的良好平衡，是一个鲁棒性极强的检测器。

	最佳Epoch	mAP@0.5	mAP@0.5:0.95	Precision	Recall
0	61	0.995	0.75248	1.0	0.99264

4.3 模型性能量化评估

为了全面衡量训练得到的最佳模型，我们引用了标准的目标检测评估曲线。

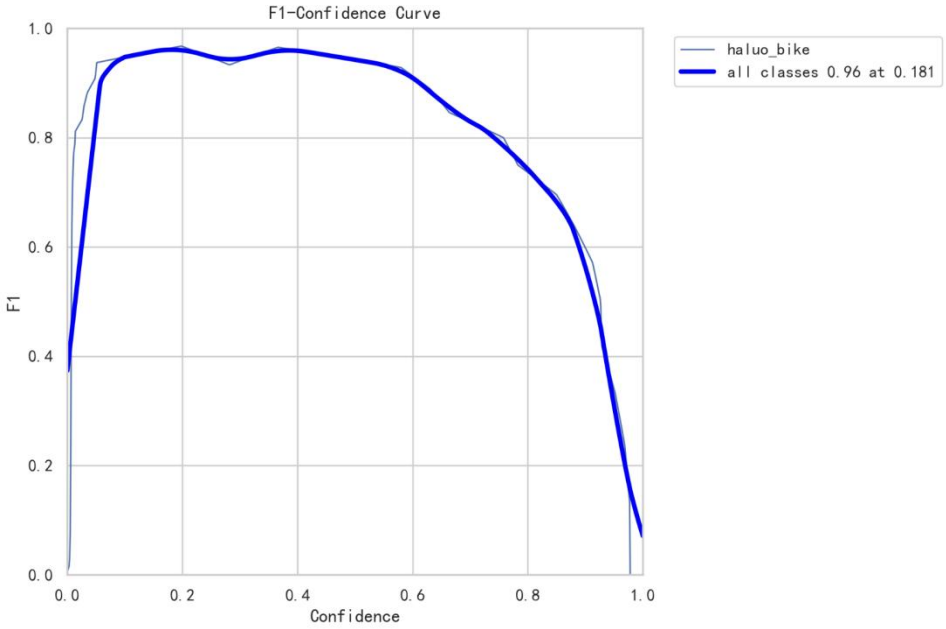
精确率-召回率曲线 (PR Curve)



分析：曲线下面积饱满，低召回时精确率稳定在 1.0（无明显误报），高召回时精确率仍保持较高水平（控制误检率）。

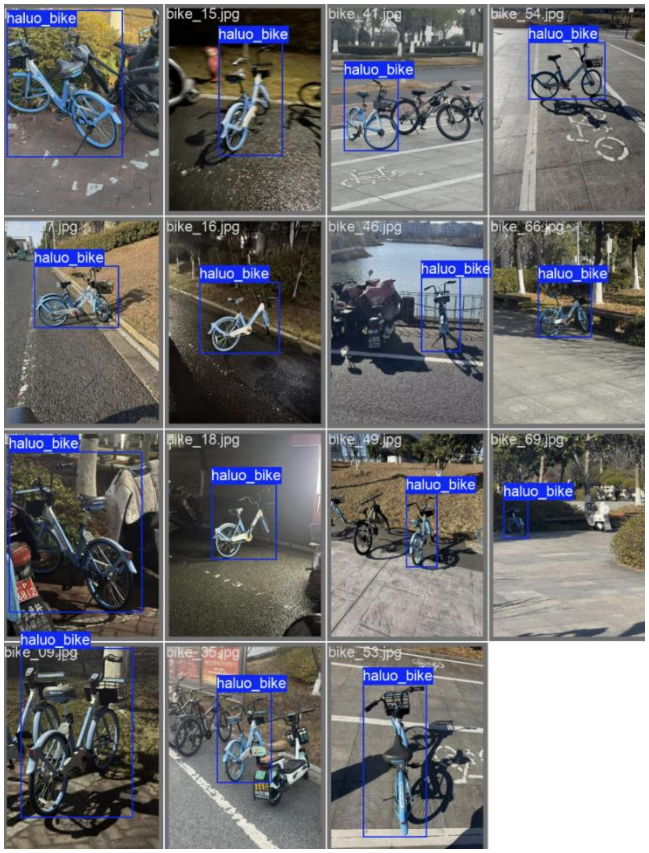
F1 分数曲线 (F1-Confidence Curve)

分析：F1 峰值出现在置信度 *0.3 ~ 0.5* 之间，为实际部署的最佳默认阈值（平衡查全率与查准率）。



4.4 验证集定性可视化分析

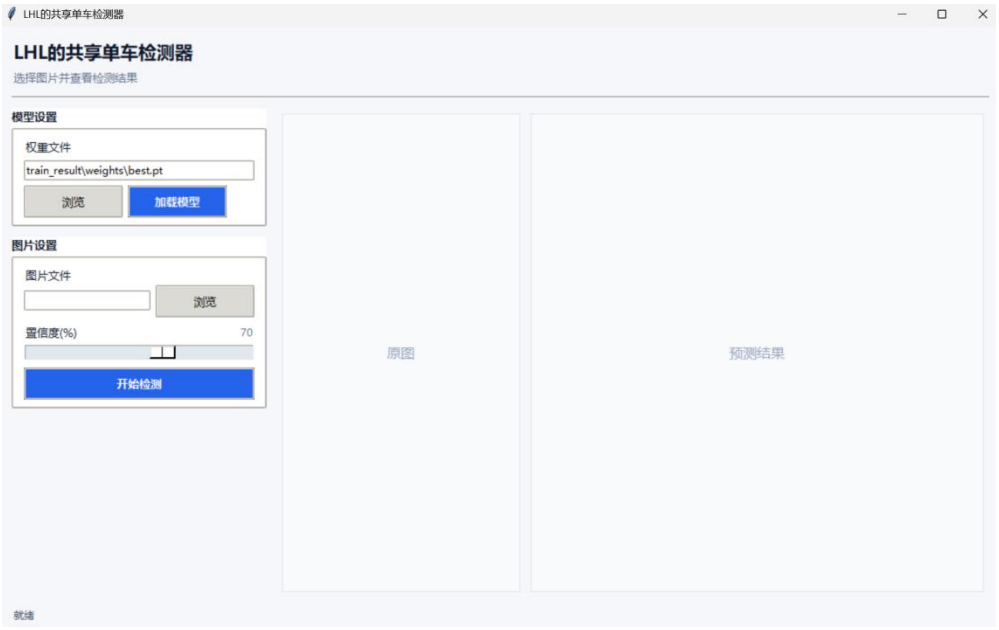
除量化指标外，通过查看验证集推理结果进行直观定性分析：



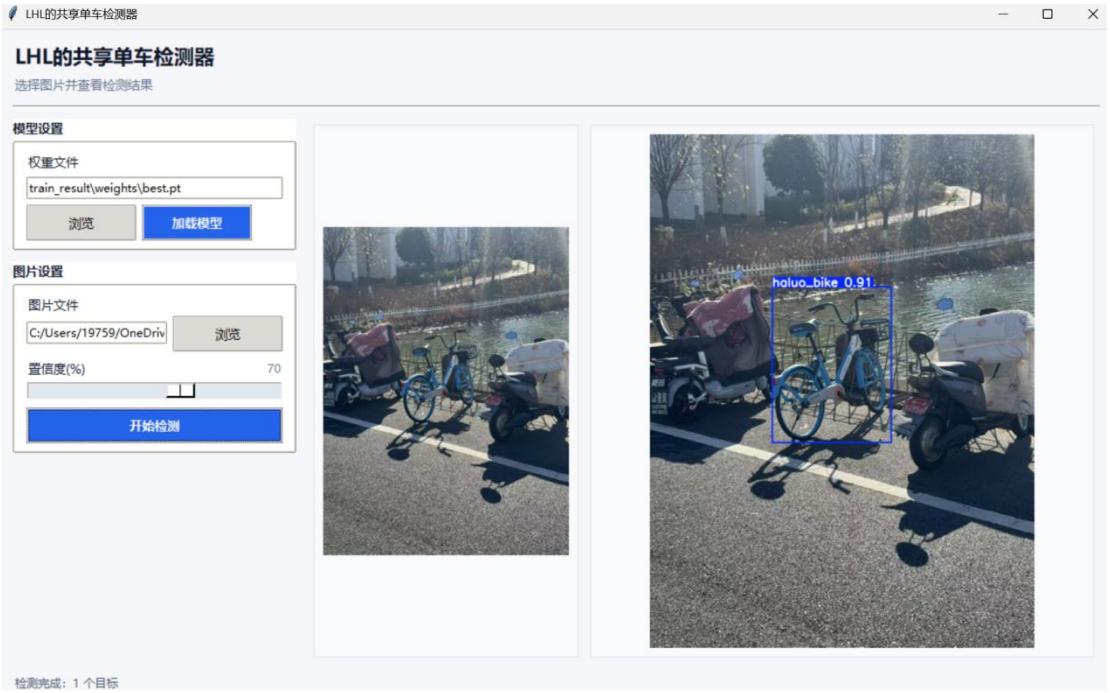
多尺度检测能力：前景大单车与背景小单车均被准确检出，验证 YOLOv8 多尺度特征融合有效性。
抗遮挡性能：密集排列的单车未出现框体合并，NMS 算法与特征提取工作正常。

4.5 交互式检测系统 (GUI) 应用测试

实验最后运行 detect_gui.py，模拟真实用户场景验证模型工程实用性。
系统界面与功能验证
GUI 启动正常，布局符合预期。

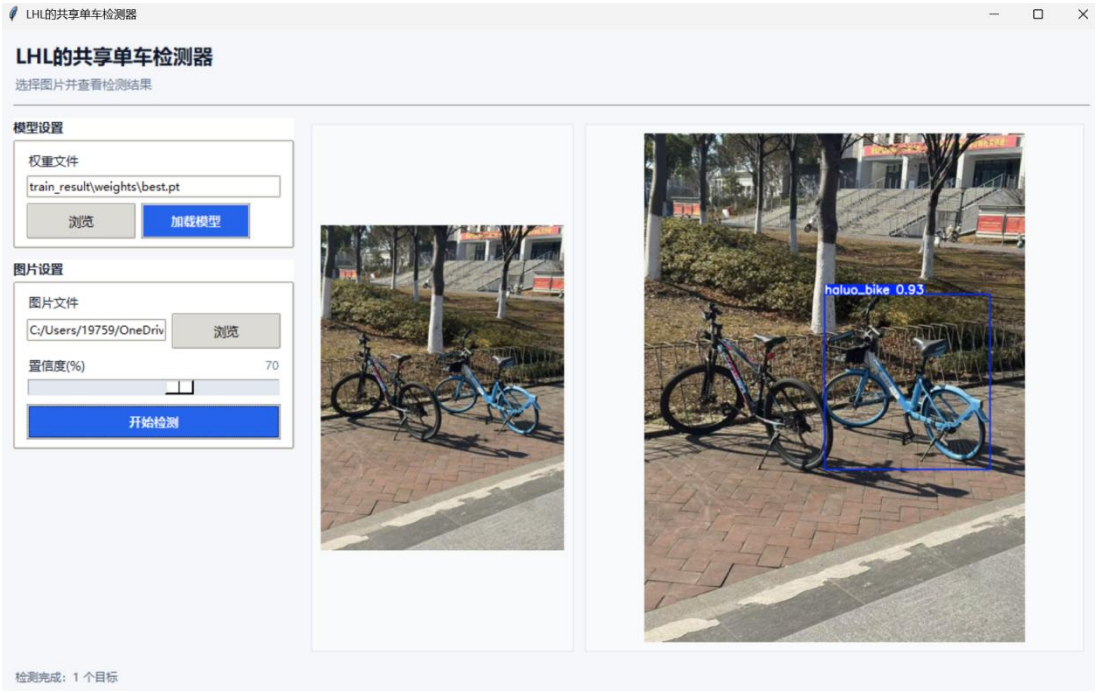


色彩还原：OpenCV BGR 转 RGB 处理后，色彩与原图一致，无蓝偏色。
检测准确性：单车准确标记为 haluo_bike，框体紧密，状态栏计数与目视一致。



细粒度分类与抗干扰能力测试（关键场景）

为了进一步验证模型在复杂真实场景下的**细粒度分类能力**，特意选取了一张极具挑战性的测试样本：**共享单车与私人自行车（普通自行车）混合停放**的照片。这一场景在校园中极为常见，且对检测算法构成了显著干扰，因为普通自行车在几何结构、轮廓形态上与共享单车高度相似，极易造成误检（False Positive）。



检测结果现象描述：

将该混停样本导入 detect_gui.py 系统，观察到以下现象：

精准捕获：系统准确地在“哈罗单车”周围生成了红色边界框，并给出了较高的置信度（ >0.9 ）。

完美拒识：紧邻其畔的“普通自行车”，尽管拥有相似的车轮、车把和车架结构，但**未被系统标记任何边界框**。系统成功将其视为背景（Background）而非目标（Object）。

结果深度分析：

这一测试结果具有重要的工程意义，揭示了模型深层的学习特征：

特征解耦与判别力：这证明了 YOLOv8 模型不仅仅学习到了“两个轮子+车架”这种通用的自行车（Bicycle）特征，更成功学习到了本次实验目标哈啰单车特有的**语义特征**（如特定的蓝色涂装、加粗的车身结构、特定的品牌 Logo 纹理等）。模型能够在特征空间中，将“共享单车”与“普通自行车”有效分离。

极低的误报率 (Low False Positive Rate)：在此前的 PR 曲线分析中，我们观察到 Precision 长期维持在 1.0。这一混停场景的实测结果再次印证了该结论——模型对负样本（干扰项）具有极强的抑制能力。

实际应用价值：在校园违停管理等实际应用中，管理人员通常只关心共享单车的规范停放，而不干涉私人车辆。模型具备这种抗干扰能力，意味着在部署时可以大幅减少因误报导致的人工复核成本，具备极高的实用落地价值。

五、 实验体会与思考

本次实验是一次从数据构建到系统集成的全栈式机器视觉实践。通过亲手编写代码并调试模型，我跳出了单纯的理论推导，对深度学习的“工程化落地”有了更为深刻和独到的理解。

1. 数据逻辑是算法生效的前提：在编写 label_yolo_single.py 时，我深入理解了 YOLO 格式标签背后的数学意义。代码中将绝对像素坐标转换为相对坐标（）的归一化处理，不仅仅是为了统一格式，更是为了赋予模型尺度不变性，使其能适应不同分辨率的输入。此外，prepare_yolo_dataset.py 中关于随机种子 (Random Seed) 的使用也给我留下了深刻印象。通过设定 seed=42 并执行 shuffle 操作，我确保了训练集与验证集的划分在时间和空间上是独立的。这让我意识到，防止数据泄露（Data Leakage）比追求高分更重要，科学的实验必须具备严谨的可复现性。

2. 软硬件适配与迁移学习的工程价值：实验代码中设计的显存自适应逻辑是我在以往课程中未曾接触过的工程细节。代码通过检测 GPU 显存大小来动态调整 batch 和 imgsz（例如显存不足时自动降级为 $batch = 4$），这种鲁棒性设计有效防止了程序因硬件差异而崩溃，体现了成熟的软件工程思维。同时，利用 COCO 预训练权重进行迁移学习的效果立竿见影，模型在前 20 个 Epoch 内 Box Loss 的迅速下降证明了“站在巨人肩膀上”对于小样本校园场景检测的巨大优势。

3. 从“模型文件”到“用户产品”的跨越：detect_gui.py 的开发过程填补了算法与应用之间的鸿沟。我遇到的最大挑战是 OpenCV (BGR) 与 Tkinter (RGB) 的色彩空间不一致问题，这让我认识到接口协议对接在系统集成中的重要性。更重要的是，通过 GUI 中的置信度滑动条，我直观地体验了 Precision-Recall Trade-off：低阈值导致误检草丛，高阈值导致漏检远处单车。这让我明白，在实际安防或管理场景中，算法的价值不仅在于训练出的 mAP 指标，更在于如何根据业务需求（如宁可误报不可漏报）找到最佳的工作阈值。

4. 规范化开发流程：响应实验要求，我使用了 Git 进行版本控制。将数据处理脚本、模型训练脚本与 GUI 代码分阶段提交的过程，不仅让我的实验思路更加清晰，也为后续的代码回溯与版本管理提供了极大便利，培养了良好的代码管理习惯。