武汉理工大学毕业设计（论文）

面向旱田除草作业的杂草识别算法研究

学院（系）：自动化学院

专业班级：自动化 2005 班

学生姓名：尹一达

指导教师：庞牧野

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：

年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密□，在　　年解密后适用本授权书

2、不保密☑　。

（请在以上相应方框内打“√”）

作者签名：　　年　　月　　日

导师签名：　　年　　月　　日

摘　要

杂草作为影响农作物生长的重要因素，其有效管理直接关系到粮食产量的稳定和提升。传统的杂草管理方式效率低下、劳动强度大，亟需新技术的介入。近年来，随着互联网、机器人与人工智能技术逐步走向成熟，特别是在智能除草领域，机器人利用先进的图像识别技术实现杂草的实时自动化识别与定位已成为可能。本研究立足于智能除草机器人的应用需求，旨在通过利用 YOLO 系列模型，探索高效、精准的旱田杂草识别方法，并为新模型的研究者和使用者提供实践方面的参考。

本文的主要工作有：

（1）回顾深度学习相关理论知识，建立杂草图像数据集，优化模型超参数并进行训练，设计动态识别算法。

（2）综合对比 YOLOv5 和 YOLOv9 在杂草识别任务中的表现，将性能较优者应用于除草机器人上。

（3）部署杂草识别系统，寻找较优的部署策略，使系统满足生产环境所需的所有要求。

对比实验的研究结果表明，与 YOLOv5s 相比，YOLOv9-c 在识别精度、识别效率、泛化程度、鲁棒性等方面表现欠佳。而基于 YOLOv5s 的杂草识别模型在各项指标上表现出色，识别准确率高达 98%，在实际应用中具有可行性和优越性。部署实验的测试结果表明，杂草识别系统满足实际生产环境所需的全部条件。

关键词：深度学习；YOLO；杂草识别；边缘计算；农业自动化

Abstract

Weeds significantly affect crop growth, impacting food production stability and enhancement. Traditional weed management methods are inefficient and labor-intensive, requiring new technologies. Recent advancements in the Internet, robotics, and AI, especially in intelligent weeding, enable real-time automated weed identification and localization using advanced image recognition.This study aims to explore efficient and accurate weed identification methods in dry fields using YOLO models, providing practical references for researchers and users.

Key tasks:

(1) Review deep learning theory, establish a weed image dataset, optimize model hyperparameters, train the model, and design a dynamic recognition algorithm.

(2) Compare YOLOv5 and YOLOv9 performance in weed identification, applying the better model to the weeding robot.

(3) Deploy the weed identification system, find the optimal deployment strategy, and ensure it meets production environment requirements.

Results show that YOLOv5s outperforms YOLOv9-c in accuracy, efficiency, generalization, and robustness, with a 98% identification accuracy. The system meets all production environment requirements.

Key Words: Deep Learning; YOLO; Weed Identification; Edge Computing; Agricultural Automation

目　录

第 1 章 绪论

1.1 研究背景和意义

杂草是指那些与当前农作物不同类且扩散迅速的植物，其对农业生产和生态环境有着深远的影响。尤其在中国这样一个农业大国，杂草问题显得尤为突出。中国地大物博，气候类型多样，适宜杂草生长的环境广泛。杂草种类繁多，分布范围广泛，几乎涵盖了全国所有农作物产区。东北地区的黑龙江、吉林，主要分布着猪殃殃(Galium aparine)和苍耳(Xanthium strumarium)；华北地区则常见有野燕麦(Avena fatua)和刺儿菜(Cirsium arvense)；长江流域广泛分布的有稗草(Echinochloa crus-galli)和狗尾草(Setaria viridis)；南方地区则以牛筋草(Eleusine indica)和稗子(Echinochloa crusgalli)为主。这些杂草不仅与作物争夺水分、养分和光照，还会导致作物减产甚至绝收。

而杂草的危害不仅局限于农业生产。它们还会对生态环境造成威胁，影响生物多样性。例如，紫茎泽兰(Eupatorium adenophorum)在西南地区的扩散，对当地的生态系统造成了严重破坏。这些入侵性杂草不仅改变了原有植被结构，还对土壤性质和水资源造成了负面影响。

人类与杂草的斗争可以追溯到农耕文明的起源。最早期的人类主要依靠手工除草，以石器和木棍为工具，通过人工拔除的方式来清理田间杂草。随着农业技术的发展，逐渐出现了一些简单的除草工具，如锄头和镰刀。在中国古代，《齐民要术》一书中就详细记载了各种除草方法和工具。

进入近现代，化学除草剂的发明和广泛应用大大提高了除草效率。20 世纪初，美国科学家首次合成了 2,4-D（一种常用的选择性除草剂），这种化学物质能够有效地杀死阔叶杂草，而对禾本科作物几乎没有影响。中国也在 20 世纪 50 年代开始引进和使用化学除草剂，极大地减轻了农民的劳动强度，提高了农业生产效率。然而，化学除草剂的长期使用也带来了环境污染和杂草抗药性等问题。

随着人们对环境保护和可持续发展的重视，生物除草（biological control）和生态除草（ecological weeding）方法开始受到关注。生物除草是利用天然敌人（如杂草病虫害、竞争性植物等）来控制杂草的生长，而生态除草则是通过调节农业生态系统中的生物群落结构，抑制杂草的生长。

进入 21 世纪，随着人工智能和机器人技术的快速发展，智能除草机器人逐渐成为农业领域的新宠。智能除草机器人能够通过搭载的摄像头和传感器，实时监测田间杂草的分布情况，并利用机器学习（machine learning）和计算机视觉（computer vision）技术，实现精准除草。

杂草识别技术是智能除草机器人的核心技术之一。通过对田间图像的分析，智能除草机器人能够准确识别不同种类的杂草，并根据其生长情况和分布特点，选择

最佳的除草策略。以 YOLOv5 为代表的目标检测算法在杂草识别中得到了广泛应用。YOLOv5（You Only Look Once version 5）是一种基于深度学习（deep learning）的实时目标检测系统，具有高效、精准的特点，能够在较短的时间内完成对大面积田地的杂草识别。使用自动化的杂草识别系统，能够减少对人工的依赖，提高除草效率，降低农业生产成本。智能除草机器人能够智能识别和定位杂草，能够精准施药或机械除草，进而减少化学除草剂的使用量，从而减缓对环境的污染和土壤的退化。

1.2 国内外研究现状

1.2.1 国外研究现状

早在 20 世纪 80 年代，美国开始研究杂草识别，并逐步扩展至室外环境，实现了从静态检测到实时跟踪的技术进步。这一技术演进不仅提高了农业自动化的效率，还为智能农业的发展奠定了基础。本文将按照时间顺序，详细介绍国外各阶段杂草识别研究的进展，并分析每个阶段的研究内容、技术特点、应用效果及其优缺点。

2004 年，丹麦奥尔胡斯大学的张寄尘等人开发了一款名为 HortiBot 的除草机器人。这款机器人利用图形算法来区分作物和杂草，展示了图像处理技术在农业中的应用潜力。这一创新不仅体现了机器人技术在农业中的前景，同时也为后来者提供了宝贵的研究方向和技术思路。随后，在 2017 年，澳大利亚昆士兰科技大学的 HallD[1]等人研发了 AgBotII 模块化除草机器人，这款机器人通过颜色分类来识别作物和杂草，并结合化学和机械方法进行除草，体现了多种技术手段的集成应用。通过这种多元化的技术手段，AgBotII 能够更高效地进行杂草处理，同时减少了对环境的负面影响。

此外，在 2017 年，Yalcin[2]等人使用预训练的 CNN 对 16 种植物物种进行分类，识别精度优于传统的支持向量机（SVM）模型，这进一步证明了深度学习技术在植物分类中的优势。预训练模型利用了大量已有的数据，提高了模型的初始性能，使得在特定任务上的表现更加出色。在 2018 年，Milioto[3]等人应用了 NDVI 颜色指数和卷积神经网络（CNN）分类器，使得杂草识别的精度达到了 99.42%和 99.66%，展示了深度学习技术在高精度识别中的强大能力。NDVI 颜色指数的应用使得植物的健康状况和生长状况得以量化，而卷积神经网络则大大提高了识别的效率和准确性。

到了 2020 年，Sabzi[4]等人设计了一种基于灰度共轭矩阵（GLCM）的模型，通过提取多种特征并比较了不同算法和分类器在区分作物和杂草时的表现，从而进一步优化了识别的准确性。这一研究展示了特征提取和算法优化在提高识别性能中的关键作用。同样在 2020 年，Hu[5]等人利用 Deep Weeds 数据集评估了其识别模型，

并结合其他多个数据集来提高识别效率，这显示出数据集多样性对模型训练的重要性。通过多样化的数据集，模型能够更好地适应不同的环境和条件，提高了实际应用的可靠性。

1.2.1 国内研究现状

近年来，随着农业智能化的不断推进，杂草识别技术在国内的研究受到了广泛关注并取得了显著发展。本文将按照时间顺序，详细介绍国内各阶段杂草识别研究的进展，并分析每个阶段的研究内容、技术特点、应用效果及其优缺点。

2019 年，彭明霞[6]等人提出了一种融合 FPN 的 Faster R-CNN 方法，用于自然光照下垂直拍摄的棉花杂草图像识别。FPN（特征金字塔网络）通过多尺度特征融合，提高了模型对不同尺寸目标的检测能力。通过训练和测试 1000 幅图片的数据集，平均识别准确率达 95.5%，优化后的 Faster R-CNN 在小目标识别中表现优于 YOLOv3。这表明 FPN 的引入在复杂背景下的目标检测中具有显著优势，特别是对于农业场景中常见的杂草检测任务。

2022 年，翟长远[7]等人针对甘蓝，研究了基于 YOLOv5s 和 MobileNet v3s 的在线识别模型 YOLO mdw，在不同天气条件下的识别准确率均超过 93%，处理时间较 YOLOv5s 缩短了 26.98%。该研究展示了轻量化模型在实时应用中的潜力，尤其是在需要快速处理和高准确率的农业环境中。MobileNet v3s 的低计算量和高效特性使其在资源受限的设备上也能表现优异。

2022 年，金小俊[8]等人提出了一种蔬菜苗期杂草识别方法，通过神经网络识别青菜，并利用颜色特征分割杂草。SSD 模型在检测速度和识别率上表现最佳，F1 值为 95.4%，平均精度为 98.1%。该方法结合了深度学习和传统图像处理技术，通过颜色特征进行预处理，显著提高了模型的检测效率。这种结合方法在大规模农业场景中具有广泛的应用前景。

2023 年，冀汶莉[9]等人提出了一种基于 YOLOv5 的轻量化杂草识别方法，采用 MSRCR 预处理和轻量级网络 PP-LCNet，结合 Ghost 卷积和注意力模块，提升了识别性能和效率，平均精度为 97.8%，内存占用显著减少。MSRCR（多尺度视网膜对比度恢复）预处理增强了图像的对比度和细节，使得模型在复杂背景下仍能保持高精度。PP-LCNet 和 Ghost 卷积的引入则进一步降低了计算复杂度和内存占用，使得该方法在资源有限的农业设备上也能高效运行。

1.3 技术路线及研究内容

1.3.1 技术路线

本论文的技术路线如图 1.1 所示。

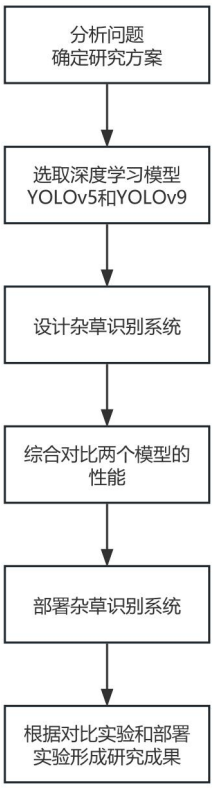

图 1.1 技术路线流程图

1.3.2 研究目标

本研究目的之一在于开发一种基于深度学习技术的杂草识别系统，并寻找最优的部署策略，以确保系统满足生产环境所需的所有要求，为智能除草机器人提供可靠的技术支持。目的之二在于测试 24 年最新版模型 YOLOv9 在杂草识别任务中的表现，旨在为新模型的研究者和使用者提供实践方面的参考。研究的具体目标包括：

（1）使用 YOLOv5 和 YOLOv9 在杂草识别任务中进行对比实验。

（2）建立杂草图像数据集。根据杂草识别任务的特点，对 YOLO 模型进行超参数优化和训练。设计动态识别算法，使模型能够进行动态检测。

（3）部署杂草识别系统，寻找较优的部署策略，确保系统满足生产环境所需的所有要求。

1.3.3 论文结构

为研究面向旱田自动化除草机器人的杂草识别问题，测试 YOLOv9 在杂草识别任务中的表现，我们从识别算法的选择、数据集的构建、对比实验的设计、实时检测算法的开发等方面展开研究，评估 YOLO 系列模型在旱田自动化除草中的可行性，测试新模型在杂草识别任务中的效率和质量，并部署杂草识别系统，使系统能够满足生产环境所需的所有要求。

第 1 章，绪论。本章介绍了面向旱田除草作业的杂草识别定位的研究背景及国内外研究现状，并确定技术路线和研究目标。

第 2 章，深度学习理论。本章依次介绍了卷积神经网络、YOLOv5 和 YOLOv9 的基本结构和工作原理，并确定选取 YOLOv5 和 YOLOv9 进行对比实验。

第 3 章，基于 YOLO 的杂草识别系统设计。本章依次介绍了数据集构建过程，模型超参数调优过程，模型评估方法和实时检测算法的实现。

第 4 章，对 YOLOv5 和 YOLOv9 在杂草识别任务中的性能进行对比。本章依次对两者在训练指标方面和识别表现方面进行了对比，最后进行了讨论分析。

第 5 章，对基于 YOLOv5 的杂草识别系统进行部署。本章详细描述了部署策略和部署算法，并对系统进行高强度的测试。

第 6 章，总结和展望。对上文研究内容和成果进行总结，并展望未来研究方向。

第 2 章 深度学习理论

2.1 卷积神经网络

2.1.1 卷积神经网络简介

卷积神经网络（Convolutional Neural Networks）是一类专门用于处理具有网格拓扑数据（如图像和视频）的深度学习模型。与传统的全连接神经网络（Fully Connected Neural Network, FCNN）不同，CNN 通过使用卷积层（Convolutional Layer）和池化层（Pooling Layer）来提取局部特征，并通过堆叠多个卷积层来学习更复杂的特征表示。CNN 在计算机视觉任务中表现出色，包括图像分类、物体检测和图像分割等。

卷积神经网络在杂草识别中被广泛应用，因为它们能够有效地处理图像数据并自动提取特征。杂草识别涉及从复杂的自然环境图像中区分杂草和作物，这需要强大的特征提取能力。CNN 通过卷积层、池化层和全连接层的组合，能够捕捉图像中的空间特征，如边缘、纹理和形状，显著提高识别准确性。此外，CNN 的参数共享和稀疏连接特性减少了计算复杂度，使其在处理大规模数据集时更高效。通过使用 CNN，可以实现对杂草的高效、精准识别，有助于提升农业自动化水平和减少除草成本。

2.1.2 卷积神经网络基本结构

CNN 的基本结构由输入层、多个卷积层、池化层和全连接层组成[10]。输入层接收原始数据，如图像的像素值。卷积层通过卷积操作提取局部特征，池化层则通过降采样操作减小特征图的尺寸，降低计算复杂度。全连接层将高维特征映射到输出类别，通过 Softmax 激活函数计算各类别的概率[11]。

卷积层的核心操作是卷积（Convolution）。对于二维图像输入，卷积操作可以描述为：

$$(I * K)(i,j) = \sum_m \sum_n I\,(i-m, j-n) \cdot K(m,n) \qquad (2.1)$$

其中，I是输入图像，K是卷积核，(i,j)是输出特征图的位置，m 和 n 是卷积核的索引。卷积核通过与输入图像的局部区域进行点积运算，生成特征图（Feature Map）。卷积核的参数通过反向传播算法（Backpropagation）进行学习，以提取有用的特征。

池化层（Pooling Layer）通过下采样操作减小特征图的尺寸，常见的池化操作包括最大池化（Max Pooling）和平均池化（Average Pooling）。最大池化选择池化

窗口中的最大值，平均池化则计算池化窗口的平均值。以最大池化为例，其操作定义为：

$$P(i,j) = \max_{m,n} I(i+m, j+n) \tag{2.2}$$

其中，P是池化后的特征图，I是输入特征图，(i,j)是输出位置索引，m 和 n 是池化窗口的位置索引。

全连接层（Fully Connected Layer）将卷积层和池化层输出的高维特征向量映射到最终的类别标签。全连接层的输出通过 Softmax 激活函数进行归一化，得到各类别的概率分布：

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.3}$$

其中，z_i是第 i 个类别的得分，n 是类别总数。

2.1.3 卷积神经网络工作原理

卷积神经网络基本的工作流程如图 2.1 所示。卷积神经网络通过逐层提取特征，实现对输入数据的层级化表示。首先，输入层接收原始图像数据并将其标准化处理。接着，卷积层通过多个卷积核对图像进行卷积操作，提取边缘、纹理等低级特征。随着卷积层的深入，提取的特征逐渐变得抽象和复杂，如形状和对象部分[12]。

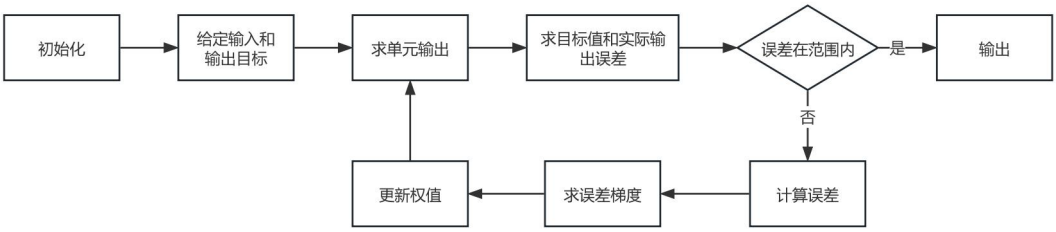

图 2.1 卷积神经网络工作流程图

池化层通过下采样操作减小特征图的尺寸，同时保留重要特征，降低计算复杂度。多次卷积和池化操作后，特征图的维度逐渐减小，但特征表达能力增强。最后，全连接层将提取的高维特征映射到输出类别，通过 Softmax 函数计算各类别的概率，实现分类任务。

CNN 的训练过程基于反向传播算法和梯度下降优化算法。首先，前向传播（Forward Propagation）计算每层的输出，直到得到最终的预测结果。然后，通过损失函数（如交叉熵损失）计算预测结果与真实标签之间的误差。最后，反向传播算法根据误差计算每个参数的梯度，并使用梯度下降算法更新参数，使得损失函数值逐渐减小，模型性能不断提升[13]。

卷积神经网络的优势在于其参数共享和局部连接机制，使得其在处理高维数据时具有较高的计算效率和较强的特征提取能力。因此，CNN 在计算机视觉领域得到了广泛应用，推动了图像识别、目标检测等任务的发展。

2.2 YOLOv5 算法

2.2.1 YOLOv5 简介

YOLO (You Only Look Once) 是在 CNN 基础上发展起来的对象检测算法.与传统的滑窗方法不同，YOLO 采用了一种端到端的检测方式。具体来说，YOLO 将图像划分成若干个网格，每个网格预测多个边界框及其类别概率。这一过程依赖于 CNN 对图像特征的提取。YOLOv5 诞生于 2020 年，作为一种先进的目标检测算法，具有高速、高精度的特点，非常适合应用于杂草识别任务。YOLOv5 可以在单次前向传播中同时检测多个杂草目标，并精确定位其位置，适用于实时检测需求。其高效的卷积神经网络架构使其在资源受限的农业设备上也能良好运行。

2.2.2 YOLOv5 基本结构

YOLOv5 的网络结构可以分为三个主要部分：骨干网络、颈部网络和头部网络，每个部分都有其独特的设计和功能[14]。

骨干网络(Backbone)采用了 CSP-Darknet53，这是对 YOLOv3 中使用的 Darknet53 的改进。通过引入交叉阶段部分网络（CSPNet, Cross Stage Partial Network）策略，CSP-Darknet53 有效减少了计算量和参数量，同时保持了高性能。具体来说，CSPNet 将基础层的特征图分为两部分，然后通过交叉阶段的方式进行融合。这种方法不仅保留了 DenseNet 的特征重用优势，还通过截断梯度流，减少了冗余梯度信息，从而提高了计算效率和推理速度.在公式上，CSP 的操作可以表示为：

$$Y = F(X_1) + H(X_2) \tag{2.4}$$

其中，X_1和X_2是输入特征图的两部分，F和H分别是两个不同的卷积操作。

颈部网络（Neck）通过结合快速空间金字塔池化（SPPF, Spatial Pyramid Pooling Fast）和改进的路径聚合网络（CSP-PANet, Path Aggregation Network）来实现多尺度特征融合。

SPPF 模块通过对输入特征进行多尺度池化，显著增加了感受野，而不会显著增加计算开销。具体实现是将最大池化层的输出依次传递给下一个最大池化层，从而提高计算效率。公式为：

$$Y_{SPPF} = [\text{MaxPool}_1(X), \text{MaxPool}_2(X), \text{MaxPool}_3(X)] \quad (2.5)$$

PANet 在 YOLOv5 中经过 CSP 改进后，进一步优化了特征融合。通过在 PANet 中引入 CSP 层，特征图的拼接操作变得更加高效，从而提升了多尺度特征的融合效果。这样的设计提高了模型的检测精度，并加快了处理速度。

头部网络（Head）延续了 YOLOv3 和 YOLOv4 的设计，负责生成边界框（bounding box）、物体置信度（objectness score）和类别概率（class probability）。它通过多个卷积层进行预测，输出三个不同尺度的预测结果，以适应不同大小的目标物体。具体的边界框预测公式如下：

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \quad (2.6)$$

其中，b_x和b_y是边界框的中心坐标，b_w和b_h是边界框的宽度和高度，t_x、t_y、t_w、t_h是网络输出，c_x和c_y是网格单元的左上角坐标，p_w和p_h是锚框的尺寸。

YOLOv5 在隐藏层中使用了 SiLU（Sigmoid Linear Unit）激活函数，其定义为：

$$\text{SiLU}(x) = x \cdot \sigma(x) \quad (2.7)$$

其中，$\sigma(x)$是 Sigmoid 函数。在输出层，使用了传统的 Sigmoid 激活函数来生成概率值。

YOLOv5 使用复合损失函数来优化模型，包括二分类交叉熵损失（BCE, Binary Cross Entropy）用于类别概率和置信度的计算，以及完全交并比损失（CIoU, Complete Intersection over Union）用于边界框的定位损失。总损失公式为：

$$Loss = \lambda_{class} \cdot BCE_{class} + \lambda_{obj} \cdot BCE_{obj} + \lambda_{bbox} \cdot CIoU_{bbox} \quad (2.8)$$

其中，λ_{class}、λ_{obj}和λ_{bbox}分别是类别损失、置信度损失和边界框损失的权重参数，确保在总损失中各部分的平衡。

2.2.3 YOLOv5 工作原理

在训练过程中，图像首先进行预处理，以确保其符合模型的输入要求。预处理包括图像尺寸调整和归一化，这些步骤是为了将不同尺寸和像素值范围的图像标准化，确保模型的输入一致性。此外，YOLOv5 使用了丰富的数据增强技术：

缩放方法（Scaling）通过调整图像的尺寸，增加数据的多样性，使模型能够更好地适应不同尺度的目标物体。这种增强方法在不改变图像内容的情况下，通过拉伸或压缩图像，生成多种不同尺寸的训练样本，从而增强模型对尺度变化的鲁棒性。

颜色空间调整方法（Color Space Adjustments）会调整图像的亮度、对比度、饱和度和色调等。这些调整可以使模型在不同光照条件下具有更好的适应性。色调（Hue）调整的公式为：

$$I' = I \cdot (1 + \Delta H) \tag{2.9}$$

其中，I是原始图像，ΔH是色调调整系数。亮度、对比度和饱和度的调整也是类似，通过比例因子对原始图像进行线性变换。

Mosaic 数据增强（Mosaic Augmentation）是 YOLOv5 中特别创新的一种数据增强方法，通过将四张图像随机拼接成一个新的图像，使得模型在一个训练批次内看到更多的目标分布和背景。这种方法有效解决了“小目标问题”，即模型对小目标检测不准确的问题。Mosaic 增强的公式为：

$$I_{mosaic} = \text{concat}(I_1, I_2, I_3, I_4) \tag{2.10}$$

其中，I_1, I_2, I_3, I_4分别是四个不同的图像块。

MixUp 方法通过线性插值两张图像及其标签，生成新的训练样本。其公式为：

$$I' = \lambda I_a + (1 - \lambda) I_b \tag{2.11}$$

$$y' = \lambda y_a + (1 - \lambda) y_b \tag{2.12}$$

其中，I_a和I_b是两张不同的图像，y_a和y_b是它们对应的标签，λ是介于 0 和 1 之间的随机数，用于控制插值的比例。

预处理后的图像被输入到模型的骨干网络 CSP-Darknet53 中，提取出多尺度特征。CSP-Darknet53 通过交叉阶段部分网络策略，有效地减小了计算量和参数量，同时保留了重要的特征信息。这些特征在经过骨干网络的多层卷积操作后，进入到颈部网络进行进一步的融合和多尺度处理。

颈部网络中包含快速空间金字塔池化（SPPF）和改进的路径聚合网络（CSP-PANet）。SPPF 模块通过对输入特征进行多尺度池化（multi-scale pooling），显著增加了感受野（receptive field），而不会显著增加计算开销。CSP-PANet 通过引入 CSP 层，进一步优化了特征融合过程，提高了模型的检测精度和速度。融合后的特征进入头部网络，生成预测的边界框、物体置信度和类别概率。

在头部网络中，YOLOv5 通过多个卷积层进行预测，并在三个不同尺度上输出结果，以适应不同大小的目标物体。模型使用复合损失函数来优化，包括类别损失、置信度损失和边界框损失。这些损失函数的加权和结合，确保了模型在多个目标检测任务中的精度和鲁棒性。

推理过程。图像首先经过与训练相同的预处理步骤，确保输入的一致性。预处理后的图像经过骨干网络和颈部网络，提取并融合多尺度特征。然后，这些特征被输入到头部网络，生成预测的边界框、物体置信度和类别概率。最后，模型进行非极大值抑制（NMS, Non-Maximum Suppression），这一过程用于去除重叠的预测框，保留置信度最高的框作为最终输出。

2.3 YOLOv9 算法

2.3.1 YOLOv9 简介

YOLOv9（You Only Look Once 第 9 版）是由 Chien-Yao Wang 等人于 2024 年 2 月推出的最新实时目标检测模型[15]。YOLOv9 在继承其前代模型优势的基础上，集成 PGI 和多功能 GELAN 架构，显著提升了模型的效率和准确性。

2.3.2 YOLOv9 结构的创新性

YOLOv9 的核心创新之一是广义高效层聚合网络（Generalized Efficient Layer Aggregation Network, GELAN）。GELAN 结合了 CSPNet（跨阶段部分网络）和 ELAN（高效层聚合网络）的优点，设计了一种能够在保持高准确度的同时显著减少参数量和计算量的新架构。

CSPNet 通过在网络中引入跨阶段的部分连接，优化了梯度路径规划（Gradient Path Planning），增强了特征提取的能力。具体来说，CSPNet 在每个阶段对特征图进行分割，一部分通过残差块（Residual Block），另一部分则直接连接到下一个阶段，确保信息流的多样性和充分性。ELAN 注重推理速度，通过叠加卷积层来实现快速推理。ELAN 的设计允许在多个网络分支中聚合变换操作，从而高效地捕捉多尺度特征。

在 CSPNet 和 ELAN 的基础上，GELAN 通过引入高效层聚合块（Efficient Layer Aggregation Blocks）和计算块（Computational Blocks）来平衡模型的性能和资源消

耗。这些块可以根据计算资源的限制进行替换，从而使 GELAN 架构能够在不同规模的模型和硬件上扩展。

YOLOv9 的另一项重要创新是可编程梯度信息（Programmable Gradient Information, PGI）技术。这项技术主要解决了轻量级模型在训练时梯度信息不可靠的问题。PGI 通过引入辅助可逆分支（Auxiliary Reversible Branches），在训练过程中保持与输入的可逆连接，确保梯度在网络中传播时不会退化。多层梯度集成（Multi-level Gradient Integration）技术通过在所有分支间整合梯度，避免不同辅助分支的干扰。这种方法确保了梯度的可靠性，特别是在轻量级模型中，显著提升了训练效果和模型的收敛速度。

2.3.3 YOLOv9 工作原理的创新性

在 YOLOv9 在训练和推理的基本工作原理上继承了 YOLO 系列的高效设计，但通过上述创新进一步优化了性能。在训练过程中，YOLOv9 采用 PGI 技术来提升梯度的可靠性。具体步骤包括：输入图像通过 GELAN 架构进行前向传播，逐层提取特征；在辅助可逆分支中，通过可逆连接保持梯度信息的完整性；使用多层梯度集成技术，将来自不同辅助分支的梯度进行整合，然后通过反向传播算法更新模型参数。通过这种方式，YOLOv9 能够在保持高准确度的同时，实现更快的训练速度和更好的模型收敛性。

在推理阶段，YOLOv9 依赖 GELAN 架构的高效特征提取和推理能力，实现实时目标检测。具体步骤如下：输入图像进行归一化处理，调整到模型所需的输入尺寸；通过 GELAN 架构逐层提取特征图；在检测头（Detection Head）中，使用 YOLO 算法的目标框回归和分类模块，对特征图进行处理，输出检测结果。

相比于 YOLOv5，YOLOv9 在架构和训练技术上的创新使其在保持实时性能的同时，进一步提升了精度和效率。总的来说，YOLOv9 的设计为实时目标检测提供了一个更加高效的解决方案。但是，由于 YOLOv9 是最新构建出的 YOLO 模型，缺少实践验证。因此，本研究使用 YOLO 系列中稳定性最好、准确性较高的 YOLOv5 与最新颖的 YOLOv9 进行同步训练和对比分析。对比实验一方面可以测验新模型在杂草识别任务中的效率和质量，一方面如果 YOLOv9 效果欠佳，可以使用 YOLOv5 补救除草机器人的需求。

第 3 章 基于 YOLO 的杂草识别系统设计

3.1 数据集构建

在旱田杂草识别的研究中，数据集的构建至关重要，它决定了模型训练的质量和识别的准确性。基于智能除草机器人的需求，杂草数据集至少要符合下列必要条件：（1）杂草拍摄条件：拍摄杂草的相机的姿态需要与除草机器人摄像头的姿态相吻合（2）杂草种类条件：根据上文中国旱田杂草的种类和分布的研究，我们需要构建对应杂草种类的数据集（3）杂草生长期条件：为了达到将杂草扼杀在萌芽的除草效果，杂草数据至少要涵盖出苗期（BBCH 00-09）和幼苗期（BBCH 10-19）。

本研究依赖于开源数据集 Weed25[16]，该数据集的构建过程具有严谨的科学方法和高度的代表性，能够满足上述必要条件和模型训练的各种常规需求。下面将介绍 Weed25 数据集的来源、数据采集方法、标注过程和数据预处理，确保数据的多样性和代表性。

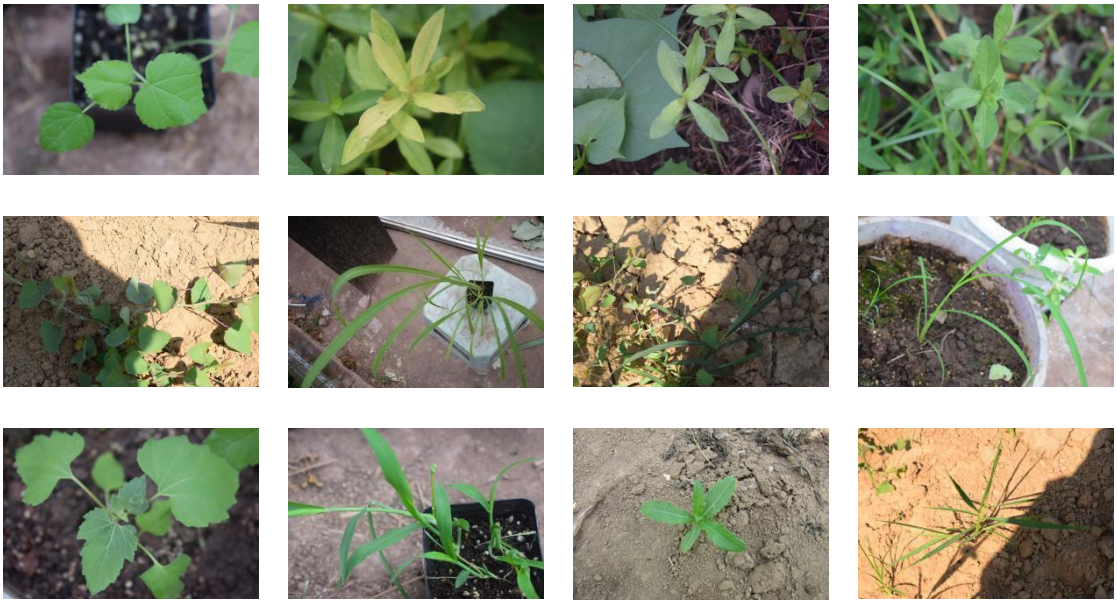

图 3.1 数据集中部分样本的图像

Weed25 数据集部分图像如图 3.1 所示，图像资源来自于中国重庆的田地和草坪，涵盖了东亚地区常见的 25 种杂草。这些图像采集于 2021 年 10 月至 2022 年 8 月之间，拍摄设备包括尼康 D5300 单反相机和华为手机，拍摄高度和角度大致在 30-50 厘米和 60°-90°之间，以确保尽可能垂直于杂草进行拍摄。采集时间分布在每天的 9:00 到 17:00 之间，包括晴天、阴天和雨天等不同光照条件，从而使数据集中的光照条件尽可能代表自然复杂环境中的实际情况。数据集包含 14,023 张图像，由来自 14 个科的 25 种杂草组成，每个科包含多个杂草种类。数据集涵盖不同生长阶段的杂草形态，大多数杂草图像采集于其生长阶段的二至九子叶期（BBCH 12-19）。

数据集中单张照片的分辨率 2992×2000，单张照片的平均大小为 1.85MB，数据集总量占 27GB。图像的分辨率越高，图像中包含的信息也就越多，高分辨率的图片数据可以提高模型的识别率。而对于计算资源有限的实验人员需要对数据集进行压缩处理，否则会消耗过多的时间成本甚至会出现内存不足的报错。

Weed25 只提供了.xml 格式的标签文件，而我们需要.txt 格式的标签文件来训练模型。由于开源数据集存在部分图像标注不当问题，我们首先将所有的图片和标签文件导入 roboflow.com 进行标注检查，手动修改标注不当的标签文件，再将所有的标签文件导出.xml 格式。最后，编写 python 脚本，将.xml 格式的标签文件全部转化为.txt 格式。

表 3.1 Weed25 数据集结构

科	名称	总数	训练集	验证集	测试集
禾本科	稗草	563	394	112	57
	马唐草	594	415	118	61
	狗尾草	552	386	110	56
莎草科	莎草	594	415	118	61
菊科	小蓬草	192	134	38	20
	田野蓟	565	395	113	57
	苍耳	745	521	149	75
	印度飞蓬	510	357	102	51
	鬼针草	612	428	122	62
	落葵	536	375	107	54
	藿香蓟	599	419	119	61
蓼科	白花蓼	671	469	134	68
	日本蓼	490	343	98	49
	中华蓼	390	273	78	39
苋科	空心莲子	637	445	127	65
	苋菜	742	519	148	75
十字花科	荠菜	224	156	44	24
马齿苋科	马齿苋	730	510	146	74
鸭跖草科	鸭跖草	562	393	112	57
藜科	藜	593	415	118	60
车前科	车前草	556	389	111	56
堇菜科	堇菜	523	366	104	53
茄科	龙葵	606	424	121	61
蔷薇科	蛇莓	615	430	123	62
锦葵科	苘麻	622	435	124	63
		14,023	9806	2796	1421

为了保证模型训练的科学性，我们将数据集按照 7:2:1 的比例分为训练集、验证集和测试集，具体划分为 9,806 张训练图像、2,796 张验证图像和 1,421 张测试图

像，如表 3.1 所示。这样的数据划分方式确保了训练、验证和测试过程中数据的充分性和代表性，能够有效避免过拟合，提高模型的泛化能力。

3.2 训练过程描述

3.2.1 训练环境概述

深度学习所使用的设备是一台高性能桌面工作设备。设备配备了第十一代 Intel Core i7-11700F 处理器，具有 8 核心，16 线程。系统内存为 32GB，能够处理大量数据并支持多任务并行操作。图形处理单元为 GeForce RTX 4090，这款显卡提供了卓越的浮点计算能力和大容量显存，适用于复杂深度学习模型的训练和推理任务。操作系统运行在 Windows 10 环境下，使用 Conda 来管理 Python 虚拟环境，确保软件包的一致性和依赖关系的稳定。

在软件环境配置方面，Python 版本为 3.8.15，深度学习框架选择 PyTorch，YOLOv5 模型选择的版本是 torch-1.12.1+cu116-cp38-cp38-win_amd64，YOLOv9 模型选择的版本是 torch-1.8.1+cu111-cp38-cp38-win_amd64。PyTorch 以其灵活性和高性能在学术研究和工业应用中广泛使用，支持多种神经网络架构和优化算法。

3.2.2 超参数配置和优化

YOLOv5 和 YOLOv9 的超参数设置如表 3.2 所示。

表 3.2 模型超参数配置

模型	YOLOv5s	YOLOv9-c
参数量（parameters, M）	7.0	50.9
批处理大小（batch size）	40	10
轮次（epochs）	70	70
工作线程（workers）	3	3
初始学习率（lr0）	0.01	0.01
最终学习率倍数（lrf）	0.01	0.01
优化器（optimizer）	SGD	SGD

YOLOv5 选择的具体预训练模型为 YOLOv5s。YOLOv9 选择的具体预训练模型为 YOLOv9-c 模型。由于 YOLOv9-c 模型参数量是 YOLOv5s 几倍的大小，所以两者的超参数设置会有所不同。在训练过程中，我们将 YOLOv5 模型批处理容量（batch size）设置为 30，将 YOLOv9 模型批处理容量（batch size）设置为 10，以平衡显存使用和训练效率，确保在 RTX 4090 显卡资源允许的范围内最大化数据处

理量。训练周期（epochs）均设定为 70 次，这一设定基于多次实验验证，既能确保模型充分训练，又能避免过度训练导致的过拟合。

YOLOv5 和 YOLOv9 均采用称为 OneCycleLR 的学习率调度策略，这种方法可以动态调整学习率，帮助模型更好地收敛。学习率是指在每次参数更新时，模型参数调整的步长。合理的学习率设置可以加速模型的收敛速度，并帮助模型找到全局最优解。初始学习率（lr0）设置为 0.01，这意味着在训练的初始阶段，每一步的参数更新幅度相对较大。这个过程可以通过式（3.1）表示。

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla L(\theta_t) \tag{3.1}$$

其中，θ_t表示第 t 次迭代的模型参数，η为学习率，$\nabla L(\theta_t)$为损失函数关于模型参数的梯度。该公式表示每次迭代时，模型参数根据当前梯度和学习率进行更新。

在 OneCycleLR 学习率调度策略中，lrf 参数决定了最终学习率相对于初始学习率的比例。这里的 lrf 设置为 0.01，意味着最终学习率将是初始学习率的 1%。OneCycleLR 策略通过三个阶段来动态调整学习率：首先，学习率从一个较低值逐步上升到一个较高值；然后，在中间阶段保持较高值；最后，再逐步降低到一个很小的值。这样可以使模型在初期快速接近最优解，并在后期细致调整参数以达到更好的泛化能力。这个过程可以通过式（3.2）表示。

$$lr_t = lr_{max} \cdot \left(1 + cos\left(\frac{t}{T} \cdot \pi\right)\right) \cdot 0.5 \tag{3.2}$$

其中，lr_t为第 t 次迭代的学习率，lr_{max}为最大学习率，T为总迭代次数。该公式描述了学习率如何在整个训练过程中动态变化。

YOLOv5 和 YOLOv9 均使用随机梯度下降（SGD）优化器，并结合动量和权重衰减，以提高模型训练的效率和效果。动量（momentum）是优化器中的一个参数，用于加速收敛和减少振荡。在每次参数更新时，动量会结合之前的梯度方向，从而在更新时考虑之前的动量，使模型能够在谷底附近平稳收敛。模型中的动量设置为 0.937，这一较高的动量值可以帮助模型更快地收敛，同时减小震荡。这个过程可以通过动量梯度下降公式（3.3）表示。

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla L(\theta_t) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{3.3}$$

其中，v_t为动量，γ为动量因子（0.937），η为学习率（lr0），$\nabla L(\theta_t)$为当前梯度。该公式表示在每次参数更新时，动量如何结合当前梯度和之前的动量。

YOLOv5 和 YOLOv9 中的权重衰减是一种正则化技术，用于防止模型过拟合。通过在损失函数中加入参数权重的 L2 正则化项，可以有效地限制参数值的增长，从而提高模型的泛化能力。在模型中，权重衰减项设置为 0.0005。这个过程可以通过权重衰减公式（3.4）表示。

$$\theta_{t+1} = \theta_t - \eta(\nabla L(\theta_t) + \lambda\theta_t) \quad (3.4)$$

其中，λ为权重衰减系数（0.0005）。该公式表示在每次参数更新时，如何将权重衰减应用于参数更新。

为了提高数据加载效率，我们设置了 3 个 workers 用于数据预处理的多线程操作。这种设置能够充分利用多核处理器的优势，提升数据加载速度，确保 GPU 在训练过程中的高利用率，避免因数据加载瓶颈而导致的训练停滞。

3.3 模型性能评估方法设计

在评估 YOLO 模型的性能时，我们采用了一系列精确的方法和计算公式，以确保模型在识别旱田杂草方面的有效性和可靠性。这些评估方法不仅包括对模型输出日志的分析，还涵盖了各项关键指标的计算，并通过多种曲线图进行可视化，以深入理解模型的行为和表现。

3.3.1 训练过程损失函数

我们需要明确模型输出日志中的各项损失函数，包括训练过程中的 train/box_loss、train/obj_loss、train/cls_loss 和验证过程中的 val/box_loss、val/obj_loss、val/cls_loss。

（1）train/box_loss 和 val/box_loss 衡量的是预测框与实际框的重叠程度。较低的 box_loss 意味着模型能够更准确地预测目标的位置。如果训练和验证损失都逐渐降低，说明模型在不断学习并改善对目标位置的预测。如果训练损失降低但验证损失不降甚至上升，则可能出现过拟合问题，说明模型在新数据上的泛化能力较差。

（2）train/obj_loss 和 val/obj_loss 衡量的是目标存在性的置信度误差。较低的 obj_loss 表明模型能够更准确地预测目标是否存在。如果 obj_loss 在训练和验证集上都降低，表示模型在判断目标存在性方面表现良好。如果验证集上的 obj_loss 不降，可能意味着模型在新数据上辨别目标存在性方面存在困难。

（3）train/cls_loss 和 val/cls_loss 衡量的是类别预测的准确性。较低的 cls_loss 表明模型能够更准确地识别目标的类别。在单类别检测任务中，这部分损失应较小。如果验证集上的 cls_loss 较高，则说明模型在实际应用中可能会出现分类错误。

（4）train/dfl_loss 和 val/dfl_loss 衡量的是模型在边界框预测精度上的表现。dfl_loss 基于分布式回归方法，通过预测每个坐标的概率分布来改进边界框定位的精度。较低的 dfl_loss 表明模型在精细定位边界框时表现良好，预测的坐标值更接近真实值。

通过监测这些损失函数的变化，可以评估模型的收敛情况和泛化能力。

3.3.2 模型性能评价参数

为了全面评估模型的识别能力，我们还需要计算并分析以下关键指标：精确率（Precision, P）、召回率（Recall, R）、平均精度（Mean Average Precision, mAP）。这些指标有助于量化模型的性能，具体如下：

（1）精确率（Precision, P） 衡量的是模型预测的所有正样本中实际为正的比例。公式为：

$$P = \frac{TP}{TP + FP} \times 100\% \tag{3.5}$$

其中，TP（True Positives）代表真实为正且被正确预测为正的样本数量，FP（False Positives）代表实际为负但被错误预测为正的样本数量。较高的精确率表明模型预测的正样本多数是正确的，误报较少。

（2）召回率（Recall, R） 衡量的是所有实际为正的样本中被正确预测为正的比例。公式为：

$$R = \frac{TP}{TP + FN} \times 100\% \tag{3.6}$$

其中，FN（False Negatives）代表实际为正但被错误预测为负的样本数量。较高的召回率表明模型能够识别出大多数实际存在的正样本，漏报较少。

（3）平均精度均值（Mean Average Precision, mAP） 综合衡量模型在不同置信度阈值下的表现。具体而言，mAP50 表示在 IOU（Intersection Over Union）阈值为 0.5 时计算的平均精度，而 mAP50-95 则是平均多个不同 IOU 阈值（从 0.5 到 0.95，以 0.05 为步长）下的精度。IOU 阈值是衡量预测框和真实框重叠程度的指标。

$$IOU = \frac{\text{预测框} \cap \text{真实框}}{\text{预测框} \cup \text{真实框}} \tag{3.7}$$

平均精度（AP）表示检测网络对某一类目标的检测效果。该值越大，整体检测效果越好。平均精度是通过绘制精确率-召回率（P-R）曲线并计算该曲线下面积（积分）得到的。AP 的计算公式为：

$$AP = \int_0^1 P\,(R)\,dR \tag{3.8}$$

平均精度均值（mAP）表示数据集中所有类别的平均精度的平均值。计算方式为所有类别的平均精度之和与所有类别数量的比值

$$mAP = \frac{1}{N}\sum_{i=1}^{N} A\,P_i \tag{3.9}$$

其中，AP_i是第 i 个类别的平均精度，N是类别总数。对于单一检测类别的杂草识别任务，mAP 计算将会简化为对唯一类别的 AP 的评估。

3.3.3 模型性能评价曲线

为了更加直观地分析模型性能，我们会生成一系列曲线图，包括 F1-Confidence 曲线、P-Confidence 曲线、R-Confidence 曲线和 P-R 曲线。

（1）P-Confidence 曲线 和 R-Confidence 曲线 分别展示了不同置信度阈值（Confidence）下的精确率和召回率变化情况。我们从 0 到 1 之间选取一系列不同的置信度阈值作为自变量。通过这两个曲线，可以选择最佳的置信度阈值，以平衡精确率和召回率。置信度阈值是模型预测某一检测为正样本的置信水平，置信度越高，模型越确定其预测结果。较高的置信度阈值通常会提高精确率但降低召回率，反之亦然。

（2）F1-Confidence 曲线 展示了不同置信度阈值下模型的 F1 值。F1 值是精确率和召回率的调和平均数，见式（3.10）

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \times 100\% \tag{3.10}$$

较高的 F1 值表明模型在精确率和召回率之间达到了良好的平衡。

（3）P-R 曲线（Precision-Recall Curve） 以召回率（R）为横坐标，精确率（P）为纵坐标，绘制不同置信度阈值下的精确率和召回率点，连接这些点形成 P-R 曲线。其展示了不同召回率水平下的精确率，帮助我们了解模型在不同召回率水平上的表现。通过观察 P-R 曲线，如果可以识别出在高召回率下保持较高精确率的区域，表明模型性能较优。

3.4 动态识别算法设计

为了满足除草机器人对动态识别的需求，本研究开发了一套基于 YOLO 的杂草识别模型的动态识别算法，通过捕捉和处理实时视频流，实现对旱田杂草的实时检测和定位。

动态识别算法的实现依赖于以下几个关键步骤：目标窗口定位、图像捕捉、图像预处理、模型推理、非极大值抑制（NMS）及结果显示。文中插入的代码段均为伪代码，仅供参考，程序可用源代码见附录 A。

（1）我们通过调用 win32gui 库中的 FindWindow 函数来定位目标窗口，并使用 GetWindowRect 函数获取窗口在屏幕上的位置。窗口的左上角和右下角的坐标分别为 $(x1, y1)$ 和 $(x2, y2)$，从而确定识别区域的矩形位置 rect。

（2）为了获取实时视频流中的图像帧，我们使用 PIL 库中的 ImageGrab.grab 函数，从指定的窗口区域截取当前帧的图像。并使用 np.array 函数将图像转换为 NumPy 数组 img0。这一过程为后续的图像处理和检测提供了基础数据。

（3）为了确保输入图像符合 YOLO 系列模型的输入要求，我们对图像进行了缩放和归一化等处理。首先，使用 letterbox 函数将图像调整为模型所需的尺寸，同时保持图像的纵横比。调整后的图像被转换为内存连续存储的数组，以提高处理速度。接着，将图像数据转换为 PyTorch 张量，并根据计算设备（CPU 或 GPU）的类型选择是否使用半精度浮点数（FP16）。图像数据随后被按照式（3.11）归一化至 $[0.0, 1.0]$ 的范围。添加批次维度以适应模型输入要求。使用 permute 方法改变张量维度顺序，以适应模型的通道顺序。

$$img = img/255.0 \tag{3.11}$$

（4）完成图像预处理后，处理后的图像被输入到模型中进行推理。模型通过前向传播过程，生成包含预测边界框和类别信息的输出张量。这一步骤是整个识别过程的核心，通过神经网络的计算，识别出图像中的潜在目标物体。

（5）为了提高检测结果的准确性，模型推理后需要对生成的边界框进行非极大值抑制（NMS）。NMS 算法通过设定置信度阈值 conf_thres 和交并比（IoU）阈值 iou_thres 来保留概率最高的检测结果，去除冗余的边界框。非极大值抑制的核心在于计算每个检测框的交并比，通过比较检测框之间的 IoU 值，过滤掉那些重叠度较高的冗余检测框，只保留最有可能的目标位置。这一步骤确保了检测结果的精确性和可靠性。

（6）最后，对检测结果进行处理。通过 scale_coords 函数将边界框从模型输入尺寸缩放回原始图像尺寸，在这一过程中，边界框的坐标被重新映射到原始图像

的尺寸上，以确保检测结果的准确性。使用 OpenCV 库中的绘制函数，将每个检测结果以矩形框的形式显示在图像上，并通过窗口展示处理后的图像。

为了实现对实时视频流的处理，上述步骤被封装在一个无限循环中，通过反复执行，实时更新和显示每帧图像。每帧图像都经过捕捉、预处理、推理和结果显示的完整流程，确保除草机器人能够实时、准确地检测和定位旱田中的杂草。

```
1.    im = ImageGrab.grab(bbox=rect)
2.    img0 = np.array(im)
3.    img = letterbox(img0, stride=stride)[0]
4.    img = np.ascontiguousarray(img)
5.    img = torch.from_numpy(img).to(device)
6.    img = img.half() if half else im.float()
7.    img /= 255
8.    if len(img.shape) == 3:  img = img[None]
9.    img = img.permute(0, 3, 1, 2)
10.   pred = model(img, augment=False, visualize=False)[0]
11.   pred = non_max_suppression(pred, conf_thres, iou_thres)
12.   for i, det in enumerate(pred):
13.   det[:, :4] = scale_coords(img.shape[2:], det[:, :4], img0.shape).round()
14.   for *xyxy, conf, cls in reversed(det):
15.   cv2.rectangle(img0, (int(xyxy[0]), int(xyxy[1])), (int(xyxy[2]), int(xyxy[3])),
    color, 3)
16.   tars.append((int(xyxy[0]), int(xyxy[1]), int(xyxy[2]), int(xyxy[3])))
```

第 4 章 YOLOv5 与 YOLOv9 性能对比

4.1 综合指标评估

4.1.1 测试集测评指标分析

模型训练结束后会通过测试集测评各项指标，结果如表 4.1 所示。

表 4.1 测试集测评结果

模型	YOLOv5s	YOLOv9-c
精确率（%）	0.94	0.847
召回率（%）	0.969	0.815
平均精度均值（%）	0.983	0.895
F1 分数	0.954	0.56
参数量（M）	7.0	50.9
训练时长（Min/epoch）	21.5	26.5

评估结果显示，YOLOv5s 比 YOLOv9-c 的训练速度更快。两个模型的精确率均表现优良，意味着模型在预测杂草存在时具有较高的准确性，误报率较低。两个模型的召回率均表现优良，这意味着模型能够识别出绝大多数的杂草，漏报率较低。YOLOv5s 和 YOLOv9-c 均获得了较优的识别精度。YOLOv5s 平均精度均值高达 0.983，表明模型在各种置信度阈值下都能保持较高的检测性能。YOLOv9-c 的 F1 分数与其 mAP 背离较大，说明 YOLOv9-c 的 F1 分数波动较高，模型精度在不同置信度阈值下无法保持稳定。在测试集指标的对比实验中，YOLOv5s 各项数据均优于 YOLOv9-c。

4.1.2 训练过程指标分析

模型在训练过程中会生成了 10 个参数随训练轮数（epochs）变化的曲线图。这些图表提供了对模型训练过程和最终性能的深刻洞察。

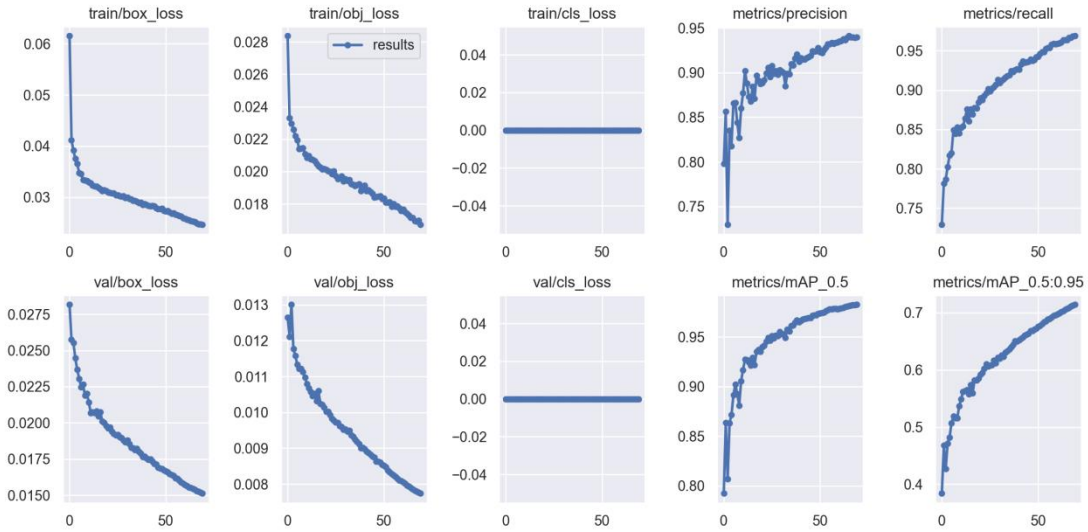

图 4.1 基于 YOLOv5 的杂草识别模型训练过程指标图

如图 4.1 所示，YOLOv5s 在训练和验证过程中的 box_loss、obj_loss 均呈现出逐渐降低的趋势并趋向于收敛，表明模型在不断学习并提高对目标位置、存在性的预测精度。其中，验证损失函数稳定下降，意味着模型具有良好的泛化能力，能够在新数据上保持较高的预测准确性。mAP_0.5、mAP_0.5:0.95 均稳定增长并趋向于收敛，表明训练过程的效果良好。识别精度在逐步增加，符合预期。

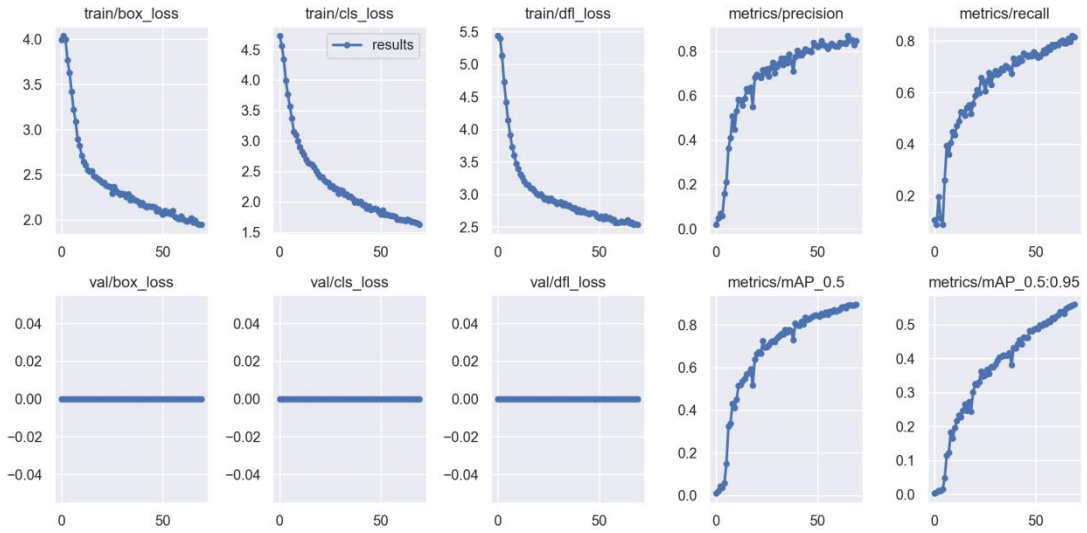

图 4.2 基于 YOLOv9 的杂草识别模型训练过程指标图

多次实验测试表明，因为未知原因，YOLOv9 的验证损失相关数据缺失，因此验证损失在这里不做讨论。如图 4.2 所示，YOLOv9-c 在训练过程中的 box_loss、dfl_loss 均呈现出逐渐降低的趋势并趋向于收敛，表明模型在不断学习并提高对目

标位置的预测精度。mAP_0.5、mAP_0.5:0.95 均稳定增长并趋向于收敛，表明训练过程的效果良好。识别精度在逐步增加，符合预期。

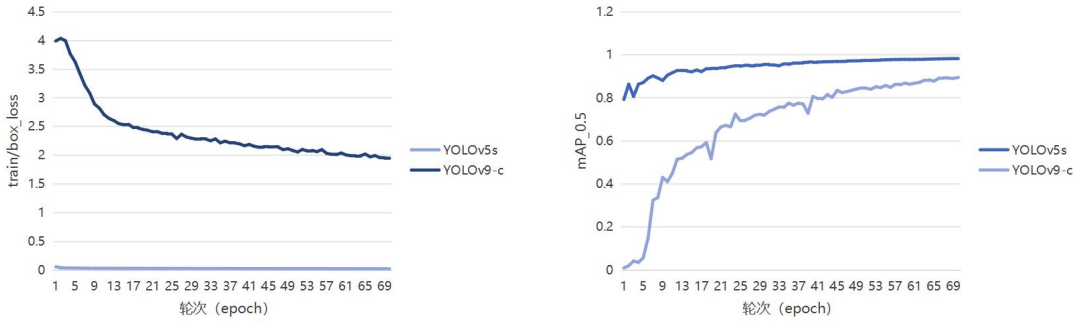

图 4.3 训练损失函数和平均精度均值的对比图

如图 4.3 比较显示，YOLOv5s 的训练损失值远远低于 YOLOv9-c，YOLOv5 的精度值远高于 YOLOv9-c。比较显示，YOLOv5s 比 YOLOv9-c 更快地实现收敛。在训练过程指标的对比实验中，YOLOv5s 各项数据均优于 YOLOv9-c，YOLOv9-c 略有不足。

4.1.3 性能评价曲线分析

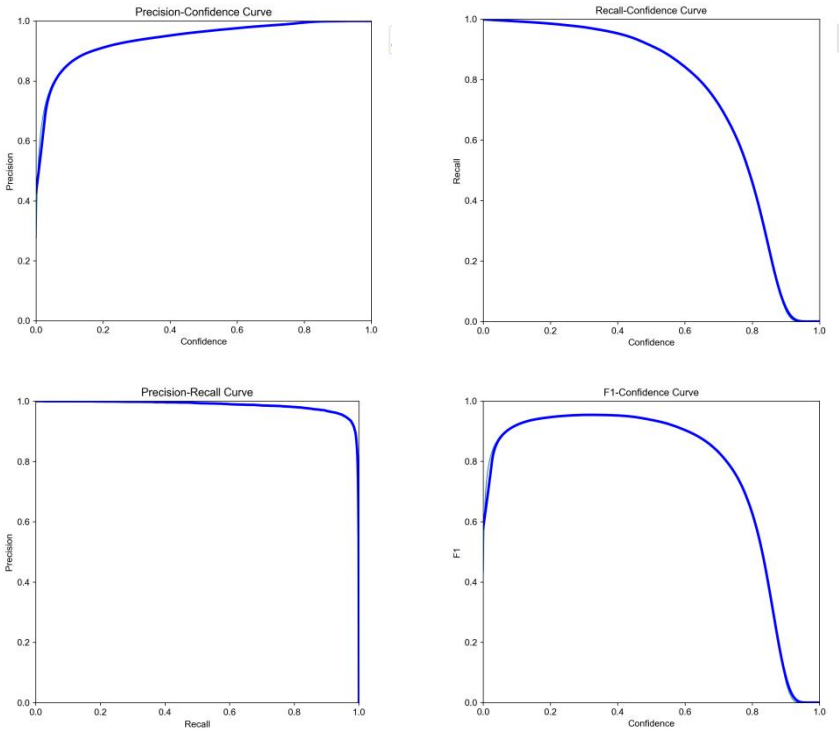

图 4.4 基于 YOLOv5 的杂草识别模型训练结果指标可视化图

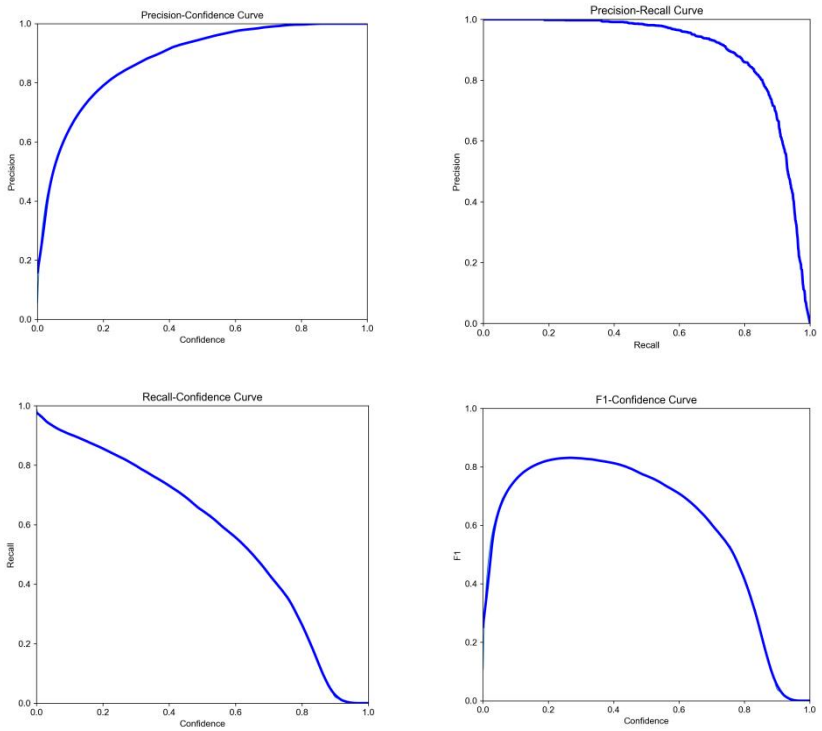

图 4.5 基于 YOLOv9 的杂草识别模型训练结果指标可视化图

如图 4.4,图 4.5 所示。YOLOv5s 的 F1 曲线图表明，在 0.1 至 0.5 的置信度阈值下，模型均能够维持高精确率和召回率，表示鲁棒性较好，稳定性较高。YOLOv9-c 的 F1 曲线图表明，只有在 0.3 左右的置信度阈值下，模型才能够维持高精确率和召回率，表示鲁棒性较差，稳定性不高。YOLOv5s 的 PR 曲线显示在高召回率（接近 1.0）时，模型精度仍然较高，表明模型几乎能够检测出所有目标，同时误检较少，意味着 YOLOv5s 模型在杂草识别任务中性能优异。YOLOv9-c 的 PR 曲线显示每次增加召回率时，精度都会等量下降，表示模型在保持高精度和高召回率方面存在明显的不足，意味着 YOLOv9-c 模型再杂草识别任务中性能欠佳。在训练结果指标的对比实验中，YOLOv5s 各项数据均优于 YOLOv9-c。

4.2 观察模型的识别表现

4.2.1 静态识别表现

我们使用测试集的数据观察两个模型的杂草检测效果

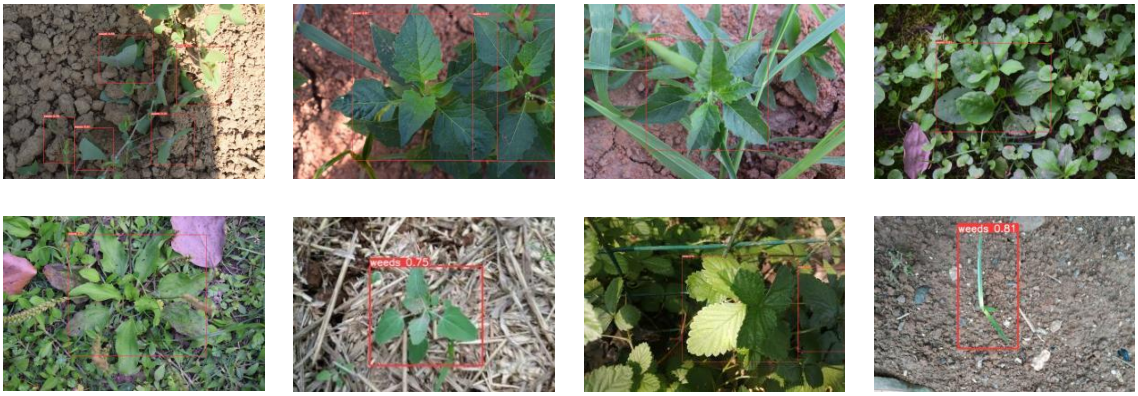

图 4.6 基于 YOLOv5 的杂草识别模型在测试集上识别的效果图

如图 4.6 显示，YOLOv5s 模型在旱田杂草识别任务中具有卓越的性能。模型不仅在训练过程中表现出良好的收敛性和稳定性，而且在实际应用中展示了高度的可靠性和精准度。

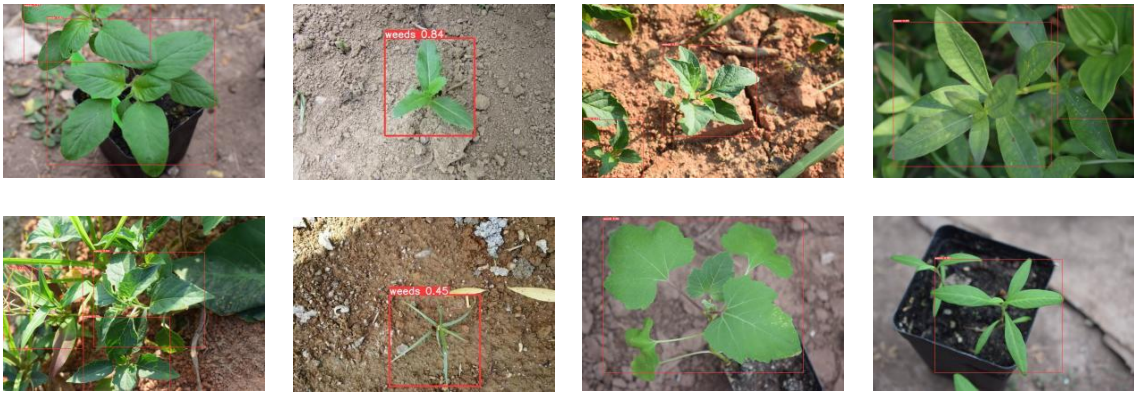

图 4.7 基于 YOLOv9 的杂草识别模型在测试集上识别的效果图

如图 4.7 显示，YOLOv9-c 模型在旱田杂草识别任务中同样具有优异的性能，模型在实际应用中展示了高度的可靠性和精准度。

最后我们比较两个模型的平均推理时长如图表 4.2 所示。

表 4.2 模型的平均推理时长和实时检测帧率

模型	YOLOv5s	YOLOv9-c
平均推理时长（ms/张）	7.9	28.9
实时检测帧率（FPS）	126.6	34.6

结果显示，YOLOv9-c 的推理速度过慢，但是目前帧率足够用于实时目标检测。然而如果使用性能较低的 GPU 和设备，YOLOv9-c 的实时检测帧率极大可能降到 30 以下，届时难以用于实时目标检测。

4.2.2 动态识别表现

由上文可得，如果使用性能较低的 GPU 和设备，YOLOv9-c 的实时检测帧率极大可能降到 30 以下，届时难以用于实时目标检测。因此，目前来说，YOLOv9 不可用在除草机器人上进行实时的杂草识别。

为了检验 YOLOv5s 杂草识别模型的实时检测能力，我们将模型应用于十段实景视频，这些视频分别在路边绿化带、院子和田间等多种环境下拍摄，分别展示了 10 种不同类型杂草的识别效果。视频均来自于网络，代表了实际应用场景中的复杂和多样的背景。这些视频充分考验了模型在不同场景中的适应能力和检测准确性。每段视频都具有动态的视觉内容，杂草的位置和形态不断变化，增加了检测的难度。

实验记录了 YOLOv5s 杂草识别模型在 10 段视频中有效框选目标的时长。我们通过计算得到 YOLOv5s 杂草识别模型在 10 段视频中平均框选了总时长的 89%。其中对于遮挡较少的视频，框选时长高达 99%。

图 4.8 基于 YOLOv5 的杂草识别模型动态识别效果（图像为视频截图，左侧为原视频，右侧为预测视频）

总而言之，YOLOv5 杂草识别模型在这 10 段视频中的表现优异，展示了 YOLOv5 在处理复杂背景和动态内容方面的卓越能力，进一步证明了将 YOLOv5 杂草识别模型应用于智能除草机器人具有可行性。

4.3 分析与讨论

4.3.1 对比实验总结

对比实验结果表明：在相同的计算设备上，YOLOv5s 比 YOLOv9-c 的训练速度更快。在精确率上，YOLOv5s 和 YOLOv9-c 的表现均优良，误报率较低。在召回率上，两个模型的表现均优良，漏报率较低。在平均精度上，YOLOv5s 的平均精度均值（0.983）较高，检测性能稳定；YOLOv9-c 的 F1 分数与 mAP 存在较大背离，不同置信度阈值下精度不稳定。在训练过程表现上，YOLOv5s 的各项指标优于 YOLOv9-c。YOLOv5s 的 box_loss 和 obj_loss 逐渐降低并趋向收敛，泛化能力良好。YOLOv5s 的 mAP_0.5 和 mAP_0.5:0.95 均稳定增长并趋向收敛，训练损失值远低于 YOLOv9-c，收敛速度更快。在 F1 曲线上，YOLOv5s 在 0.1 至 0.5 的置信度阈值下保持高精确率和召回率，鲁棒性和稳定性较高；YOLOv9-c 在 0.3 左右的置信度阈值下才能保持高精确率和召回率，鲁棒性较差。在 PR 曲线上，YOLOv5s 在高召回率时保持高精度，误检较少，表现优异；YOLOv9-c 的精度随召回率增加而等量下降，表现欠佳。在静态识别测试上，YOLOv5 在旱田杂草识别任务中表现卓越，具有良好的稳定性和可靠性。YOLOv9-c 的推理速度较慢，帧率在低性能设备上可能降到 30 以下，不适合实时检测。在动态识别测试上，YOLOv5 在十段不同场景的视频中均能准确稳定地识别杂草，平均框选时长高达 89%，展示了其处理复杂背景和动态内容的能力；YOLOv9-c 在低性能设备上难以满足实时检测要求，不适用于除草机器人进行实时检测。

显然，基于 YOLOv5 的杂草识别模型的性能最佳，能够胜任智能除草机器人的杂草实时检测任务。

4.3.2 对基于 YOLOv5 的杂草识别模型性能的讨论

在基于 YOLOv5 的杂草识别模型测试中，我们看到了令人满意的结果。然而，在分析实验结果的过程中，也发现了一些有趣的问题，这些问题揭示了当前模型的局限性。尽管这些问题对智能除草机器人的实际效果影响不大，但却为进一步的研究和改进提供了宝贵的方向。

（1）模型在识别实际杂草时表现出色，在各种动态视频中准确地定位了九种不同的杂草。然而，一个显著的问题是模型对人工生成式杂草图像或杂草 LOGO 的识别置信度甚至高于对真实杂草的识别。这表明模型在某些情况下可能会过度拟合某些特征，从而导致在非预期目标上的高置信度。这种现象的产生主要是因为模型在训练过程中学习到的特征可能在某些人造图像中得到了过度强化，导致模型误以为这些特征代表了真实杂草。

为了得到准确的生成式杂草图像误报率，我们使用 DELL-E 模型生成了 200 张杂草相关图像，并将这些图输入给 YOLOv5 杂草识别模型进行识别，识别效果如图 4.9 所示，最终测得误报结果如表 4.3 所示。

表 4.3 YOLOv5 杂草识别模型对生成式杂草的识别误报数据

生成式杂草数据总数	误报数	误报率
200	42	0.21

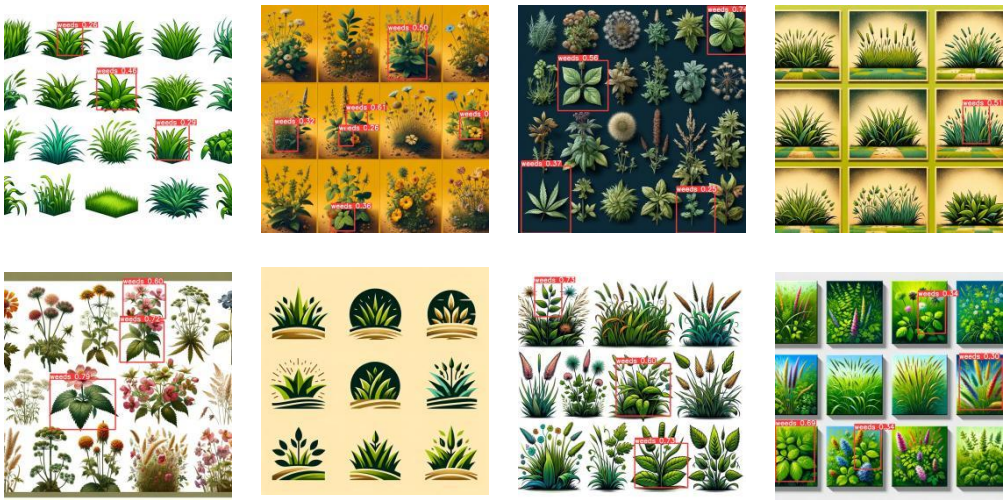

图 4.9 YOLOv5 杂草识别模型对生成式杂草的识别效果图

这种误识别的问题根源在于训练数据集缺乏多样性和泛化性。训练数据集中的杂草图像主要来自真实环境，而缺乏足够的人造杂草图像。因此，模型在面对这些非预期的图像时，往往无法正确区分其与真实杂草之间的差异。这种现象在深度学习领域被称为“域外泛化问题”，即模型在训练数据之外的未知数据域中的表现不佳。

如果要解决这个问题，我们需要在训练数据集中加入更多的人造杂草图像，通过数据增强技术和多样化的数据来源来提高模型的泛化能力。这将帮助模型更好地学习区分真实杂草与非预期目标之间的特征差异，从而提高模型在实际应用中的可靠性和鲁棒性。

（2）另一个发现的问题是模型对近地拍摄和杂草幼年期（二至九子叶期）的识别效果很好，但在高空无人机拍摄和中晚期生命周期杂草的识别上存在明显不足。这是由于训练数据集的专用性所导致的。模型训练过程中主要使用了近地拍摄和幼年期杂草的图像，导致模型在这些特定场景中的表现优异，但在其他场景中的表现受到限制。

这一问题揭示了当前数据集的局限性，特别是其在不同拍摄高度和杂草生长阶段的覆盖不足。高空无人机拍摄的图像由于分辨率较低和视角不同，对模型提出了

新的挑战。而中晚期杂草由于其形态和特征发生了显著变化，也需要不同的识别策略。

如果要解决这一问题，我们需要扩展训练数据集，涵盖更多不同拍摄高度和杂草生长阶段的图像，增加中晚期杂草的图像，通过覆盖整个杂草生命周期来提高模型的识别能力。我们还可以借助领域自适应技术，通过在不同数据域之间进行知识迁移和对齐，进一步提升模型在未知数据域中的表现。通过这种方式，可以在不大幅增加训练数据量的情况下，增强模型的适应性和鲁棒性。

4.3.3 对基于 YOLOv9 的杂草识别模型性能的讨论

由于 YOLOv9 的验证损失相关数据缺失，所以 YOLOv9 杂草识别模型的泛化性能得不到检验。由于 YOLOv5 对人工生成式杂草图像的泛化性能不够，因此我们同样可以测试 YOLOv9 杂草识别模型对人工生成式杂草图像的泛化性，并比较得出 YOLOv9 杂草识别模型的泛化性能，结果如表 4.4 所示。

表 4.4 YOLOv9 杂草识别模型对生成式杂草的识别误报数据

生成式杂草数据总数	误报数	误报率
200	48	0.24

基于数据得出 YOLOv9 杂草识别模型同样存在泛化性能不够的问题。

在基于 YOLOv9 的杂草识别模型的性能测试中，我们看到了不尽人意的结果：YOLOv9-c 模型在杂草识别任务中的各项指标数据均不如 YOLOv5s 模型。但是，这是情理之中的。YOLOv9 因为算法和模型的新颖性，必然会经历实践的考验。而通过对比实验，我们为 YOLOv9 的工程师和研究员们提供了宝贵的实践参考，这正是本研究的意义所在。

第 5 章 杂草识别系统部署

基于上文的研究结论，我们选用基于 YOLOv5 的杂草识别模型作为机器人的实时检测模型。接下来我们讨论如何将模型与除草机器人进行对接。

5.1 部署策略

5.1.1 部署方法的选择

在面向旱田除草机器人的杂草识别项目中，模型通常有几种常见的部署方法。

（1）本地部署：将模型直接部署在机器人控制程序上。这需要机器人本身搭载一个高性能的计算平台。但是大型显卡功耗较高，通常还需要强大的散热系统。在野外环境，尤其是旱田这种可能高温且缺乏电力的地方，提供稳定的电力和有效的散热是一种挑战。

（2）云端部署：将模型部署在云服务器上，机器人可以利用计算资源强大的云平台。但在广域网络上，机器人与云平台间的数据通信存在延迟甚至丢包，这对实时性要求较高的除草机器人来说是致命的。

（3）边缘计算部署：边缘计算是本地部署与云端部署之间的折中方案。边缘计算设备是指部署在网络边缘、靠近数据源的计算设备（服务器）。这样既可以解决本地部署上功耗问题，又可以解决云部署上网络通信延迟问题。因此，将模型部署在边缘计算设备上是最佳选择。而如果将边缘计算设备直接集成在机器人上则会出现与本地部署一样的问题。因此，我们的计划是在农田边缘建设边缘计算服务器设备箱，在农田区域内建设局域网全覆盖。这种方式可以有效解决机器人在电力、散热、数据延迟等方面的问题。

由于实验是在开发环境下，我们选用 DELL 笔记本（CPU:i5）模拟机器人设备，台式桌面工作站（CPU:i7+GPU:RTX4090）模拟边缘计算设备，两者通过局域网连接，服务器地址和端口设为 192.168.1.1:8001（可更改）。

5.1.2 编程语言选择与网络请求策略

机器人端使用与机器人控制算法相同的编程语言 Python 并将网络请求函数写在相机接口函数下以避免不同语言不同文件之间调用造成的延迟。程序使用循环函数对视频流数据进行帧分片，再通过网络请求将帧数据发送给服务端。

服务端使用与模型推理算法相同的编程语言 Python 并将服务器代码编写在推理算法下，以避免不同语言不同文件之间调用造成的延迟。程序将接收到的帧数据输入给模型进行推理，再将预测结果发送回机器人。识别结果为一个列表数据[[a,b,c,d],...]，其中[a,b,c,d]为识别到的对象之一。a 和 b 表示对象边界框的左上角的坐标，c 和 d 表示右下角的坐标。

5.1.3 通信协议和服务器框架的选择

边缘计算设备与机器人的通信可以通过有线连接和无线连接两种方式。有线连接中，可以通过 USD 方式连接，也可以通过以太网方式连接。无线连接可以通过 WIFI 连接，也可以通过蓝牙连接。对于在大面积田间自由移动的除草机器人来说，无线连接中的 WIFI 局域网连接是最优选择。

HTTP 是一种用于分布式、协作、超媒体信息系统的经典的应用层协议。POST 是 HTTP 的请求方法之一，用于向服务器发送大型复杂的数据。Flask 是一个轻量级的 Python Web 框架，它的核心非常小，却可以扩展以满足复杂的需求。在没有复杂业务逻辑的情况下，Flask 应用的响应时间通常很快，适用于基本的 HTTP 请求。WebSocket 是一个全双工通信协议，允许服务器和客户端之间实时双向数据传输。SocketIO 是基于 WebSocket 的服务端框架。因此，在具体的应用层协议和框架上，我们分别编写使用 HTTP 协议和 Flask 框架进行通信的程序和使用 WebSocket 协议和 SocketIO 框架进行通信的程序进行网络延迟比较。

（1）HTTP 协议和 Flask 框架：机器人端程序使用 read 函数读取图像二进制数据并使用 reqursts 库发送 post 请求，使用 response.json 接收识别结果。

```
    with open(file_path, 'rb') as img:
        response = requests.post(url, files={'file': img.read()}) #发送图片数据
        inference_result = response.json() #接收识别结果
```

服务端程序使用 Flask 库监听请求，并使用 return jsonify 响应识别结果。

```
    @app.route('/upload', methods=['GET', 'POST']) #接收图片数据
    def upload_file():
        images = request.files['file']
        inference_result = process_image(images) # 识别函数
        return jsonify(inference_result) #发送识别结果
```

（2）WebSocket 协议和 SocketIO 框架：机器人端程序我们使用使用 read 读取图像二进制数据并使用 emit 函数发送图像数据，使用 inference_result 函数监听识别结果。

```
    sio.emit('frame', image_file.read()) #发送图片数据
    @sio.event #接收识别结果
    def inference_result(data):
        inference_result=data
```

服务端程序使用 frame 函数监听图像数据，并使用 emit 函数发送识别结果。

```
    @sio.event #接收图片数据
    def frame(sid, data):
        inference_result=process_image(data) #识别函数
        sio.emit('inference_result', inference_result, to=sid) #发送识别结果
```

（3）我们编写了计时器代码以插入在主程序的各个函数中计算总延迟、网络

延迟、预处理延迟和推理延迟。其中，预处理延迟和推理延迟是流水线的固定速度，会引起掉帧现象。而网络延迟会导致数据传输不及时但不会引起掉帧现象。

```
    start_time = time.time()
    end_time = time.time()
    execution_time_ms1 = (end_time - start_time) * 1000
    print(f"Time Consuming: {execution_time_ms1:.2f} ms")
```

通过实验，我们得到了通过 HTTP 协议和 Flask 框架的网络通信延迟平均为 10.1ms，通过 WebSocket 协议和 SocketIO 框架的网络通信延迟平均为 557.6ms。因此我们选用 HTTP 协议和 Flask 框架作为机器人和服务器的通信协议。

5.2 机器人端算法设计

在开发环境中，我们使用一段视频代替相机的实时拍摄效果。实际上，如果应用于生产环境，也仅需更改一行代码，更改部分已在附录程序中标出。机器人端程序主要涉及以下步骤：逐帧读取视频、帧格式转换、通过 POST 请求发送图片数据、接收识别结果。文中插入的代码段均为伪代码，仅供参考，程序可用完整源代码见附录 B。

（1）我们使用 cv2.VideoCapture 函数打开指定路径的视频文件，定义了 frame_count 帧计数器，再定义了识别系统的主循环体。

（2）我们使用 cap.read 函数读取视频流中的帧图像。如果读取成功，ret 为 True，否则，ret 为 False，退出循环。

（3）为了适应 HTTP 协议要求，我们使用使用 cv2.imencode 函数将读取到的帧图像编码为 JPEG 格式。再使用 img_encoded.tobytes 函数将编码后的帧图像转换为字节数据。

（4）为了符合 requests 库的 API 要求，我们创建一个包含图像字节数据的字典 files，再使用 POST 请求将 files 上传到服务器。如果响应状态码为 200，表示识别成功，使用 response.json 方法将 JSON 字符串解码为列表即识别结果数据。程序进行下一轮循环。

```
    while True:
        ret, frame = cap.read() #逐帧读取视频
        if not ret: break
        _, img_encoded = cv2.imencode('.jpg', frame) #将帧编码为 JPEG 格式
        img_bytes = img_encoded.tobytes() #将 JPEG 图像转换为字节数据
        response = requests.post(url, files={'file': img_bytes}) #将图片字节数据发送 POST 请求
        if response.status_code == 200:  inference_result=response.json() #接收识别结果
```

5.3 服务端算法设计

服务端程序主要涉及以下步骤：创建 Flask 服务器、接收图片数据、将图片数据发送给识别函数进行预处理、推理等操作，最后将识别结果发送给机器人。文中插入的代码段均为伪代码，仅供参考，程序可用完整源代码见附录 C。

（1）服务器函数部分：我们首先创建一个 Flask 实例，再使用@app.route 定义路由监听器 upload，使用 read 函数读取 POST 请求中的图像内容，再调用函数 process_image 进行识别，最后使用 jsonify 方法将响应信息转换为 JSON 并返回给客户端。关于具体的通信内容，服务器收到的请求对象包含 device 摄像头编号、frame 当前帧编号和 file 帧图像，服务器返回的响应对象包含 device 摄像头编号、frame 当前帧编号和 result 识别结果列表。

（2）识别函数部分：由于与上文动态识别算法一致，此处不做详细描述。首先进行图像预处理，以适应模型输入的要求。利用 Image.open 读图像，通过 io.BytesIO 处理二进制数据，使用 np.array 函数将 PIL 图像转换为 NumPy 数组，便于进行数值操作。使用 letterbox 函数调整图像尺寸并填充以保持纵横比。将 NumPy 数组转换为 PyTorch 张量。根据 half 变量决定将数据类型转换为半精度或单精度浮点数。图像归一化到[0.0, 1.0]范围。添加批次维度以适应模型输入要求。使用 permute 改变张量维度顺序，以适应模型的通道顺序。最后，使用模型进行预测，并对模型输出进行后处理以得到最终的目标位置。使用 non_max_suppression 函数应用非极大值抑制（NMS）以去除冗余的检测框。后续调整检测框坐标，使其匹配原始图像的比例，再将每个检测框的坐标添加到 tars 列表中。

```
@app.route('/upload', methods=['GET', 'POST']) #服务器函数
def upload_file():
    image5 = request.files['file'].read()
    inference_result = process_image(image5)
    response_data = {'device':device,'frame':frame,'result':inference_result}
    return jsonify(response_data)
def process_image(img): #识别函数
    image = Image.open(io.BytesIO(img))
    img0 = np.array(image)
    ...#中间部分等同动态识别代码，故省略
    return tars
```

5.4 系统测试

为了测试我们部署的杂草识别系统的可靠性，本研究使用分辨率为 1920×1080，帧率为 30FPS 的 mp4 视频进行识别测试。系统稳定运行了 1 个小时以上，最后被手动停止运行。系统运行期间没有出现闪退、报错、掉帧、卡顿、数据阻塞等故障。

在超过 108,000 张帧图片上获得的平均延迟数据如表 5.1 所示。总延迟指在机器人端从视频的每帧被裁剪开始到最终获得该帧识别数据的总时长。网络延迟指进行一次往返传输的在网络链路上消耗的总时长。预处理延迟指在推理之前对图像数据进行预处理操作消耗的总时长。推理延迟指模型对图片进行推理操作和算法处理识别结果操作消耗的总时长。帧率由预处理延迟和推理延迟通过式（5.1）计算得到。通过表格数据可知，延迟达到了合格标准（小于 50ms），帧率达到了复杂环境作业标准（大于 30FPS）。

表 5.1 杂草识别系统延迟数据

平均总延迟（ms）	平均网络延迟（ms）	平均预处理延迟（ms）	平均推理延迟（ms）	平均帧率（FPS）
41.89	10.12	4.99	26.78	31.48

$$帧率 = \frac{1000}{预处理延迟+推理延迟} \tag{5.1}$$

根据第 4 章测试结果和本系统测试结果可以确定：本杂草识别系统具有低延迟、高帧率、高稳定性、高效率、高质量的特点，具备实际生产环境所需的全部条件。

第 6 章 总结与展望

5.1 研究总结

杂草管理作为农业生产中的重要环节，长期以来一直是农学研究的重要课题。随着互联网、机器人及人工智能技术的快速发展，杂草管理也逐步迈向自动化和智能化的方向。本研究一方面通过设计 YOLOv5 和 YOLOv9 的对比实验，验证了 YOLOv5 在旱田杂草识别任务中的优越性能，另一方面部署了杂草识别系统，使基于 YOLOv5 的杂草识别模型能够应用于实际生产环境中。研究结果不仅为智能除草机器人的应用提供了技术支持，也为 YOLOv9 模型的研究者和使用者提供了实践参考。本文研究成果如下：

（1）研究了深度学习理论的基础知识，重点阐述了卷积神经网络、YOLOv5 和 YOLOv9 的基本结构和工作原理。利用开源杂草图像数据集，进行了数据集构建。优化了超参数并进行模型训练。设计了动态识别算法，以满足在除草机器人上进行动态检测的需求。

（2）选取 YOLOv5 和 YOLOv9 进行对比实验。实验结果表明，在精确率、召回率和平均精度（mAP）方面，YOLOv5 均优于 YOLOv9。YOLOv5 在训练过程中的 box_loss 和 obj_loss 逐渐降低并趋向收敛，mAP_0.5 和 mAP_0.5:0.95 稳定增长并趋向收敛，显示出较强的鲁棒性和稳定性。而 YOLOv9 在这些指标上表现均不如 YOLOv5。YOLOv9 推理速度较慢，帧率在低性能设备上可能降到 30 以下，不适合实时检测。在静态和动态识别测试中，YOLOv5 均表现卓越。在 10 段不同场景的视频中，YOLOv5 能够准确稳定地识别杂草，框选时长高达总时长的 89%。而 YOLOv9 在低性能设备上难以满足实时检测要求，不适用于除草机器人实时识别任务。因此，YOLOv5 被证明是最佳的杂草识别模型，能够胜任智能除草机器人的杂草实时检测任务。

（3）指出了基于 YOLOv5 的杂草识别模型在人造杂草图像上存在域外泛化问题。指出了基于 YOLOv5 的杂草识别模型在特定场景中的表现优异，在其他场景中的表现受到限制。为未来的研究提供了方向。

（4）部署了杂草识别系统，并找到了较优的部署策略，使系统具有低延迟、高帧率、高稳定性、高效率、高质量的特点，从而使系统具备实际生产环境所需的全部条件。

5.2 研究不足与展望

本文对基于深度学习的杂草识别算法进行一系列的研究。本研究存在一些不足之处：

（1）本文为了在对比实验中控制变量，将两个模型的训练周期共同设为 70。由训练结果可知，70 个周期尚未能使 YOLOv9 模型达到最佳性能。所以 YOLOv9 的部分测试数据并不是其最优解。

（2）在杂草识别任务中，由于不同地区、气候、植被情况、拍摄视角等复杂背景条件繁多，训练模型需要包含足够多样性的杂草图像，以提高泛化能力。所以本文使用的开源数据集仍具有局限性。

（3）在杂草识别系统部署上，尚未进行更多的实验。在编程语言上还可以选用 C++或 Java；在传输方式上，还可以采用 USB4 直连方案。哪一种方案可以使系统性能更优，需要我们进行更多的研究。

展望未来，基于人工智能技术的智能除草机器人在农业生产中的应用前景广阔，但也面临诸多技术挑战。首先，现有的 YOLOv5 虽然在旱田杂草识别任务中表现优越，但其在更为复杂的农业场景中有一定局限性。未来的研究需要着重于改进模型的鲁棒性和泛化能力，使其在不同地区、气候、植被情况、拍摄视角等复杂背景条件下都能稳定工作。

随着深度学习技术的不断进步，新的模型架构和训练方法也在不断涌现。Transformer 作为一种新的神经网络架构，已经在自然语言处理领域取得了显著成效，例如我们所熟知的 GPT4 就是基于 Transformer 架构。大模型具有超高算力需求，这在各种实时检测任务中就无法满足。所以，若将大语言模型技术强行引入到对象检测任务中必然会导致过拟合。因此，算力问题和训练推理速度问题是人工智能技术正在面临的挑战，这需要我们在高性能计算设备方面有所革新。其次，模型架构也是重中之重，现有的深度网络架构究竟能否支撑起多模态和世界模型的体量呢？再此，笔者进行大胆假设。

第一，如果我们以断开能源作为“关机”的定义，那么人脑是 24 小时不间断工作的，直到被冷冻处理（科学性未经证实）或脑死亡。所以人类都要经历 10 多年的学习才能具备认知能力，那么机器呢？是否需要更长时间。第二，人脑在输出过程中会加深印象，在学习的过程中也在输出，这被称之为生成效应。而现阶段模型的训练和推理是分开的，这是否是深度神经网络架构的缺陷？无论如何，希望不久的将来人类会实现真正的多模态模型和世界模型。

参考文献

[1] Hall D, Dayoub F, Kulk J, et al. Toward unsupervised weed scouting for agricultural robotics[A]. IEEE International Conference on Robotics & Automation[C]. IEEE, 2017.

[2] H. Yalcin, S. Razavi. Plant classification using convolutional neural networks[A]. 2016 Fifth International Conference on Agro-Geoinformatics (Agro-Geoinformatics), Tianjin, China, 2016[C]. IEEE, 2016: 1-5.

[3] Milioto A, Lottes P, Stachniss C. Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs[C]. 2018 IEEE international conference on robotics and automation (ICRA). IEEE, 2018: 2229-2235.

[4] Sabzi, S., Abbaspour-Gilandeh, Y., Arribas, J. I. An automatic visible-range video weed detection, segmentation and classification prototype in potato field[J]. Heliyon, 2020, 6(5): e03685.

[5] Hu, K., Coleman, G., Zeng, S., et al. Graph weeds net: A graph-based deep learning method for weed recognition[J]. Computers and Electronics in Agriculture, 2020, 174: 105520.

[6] 彭明霞，夏俊芳，彭辉. 融合 FPN 的 Faster R-CNN 复杂背景下棉田杂草高效识别方法[J]. 农业工程学报，2019，35(20)：202-209.

[7] 翟长远，付豪，郑康，等. 基于深度学习的大田甘蓝在线识别模型建立与试验[J]. 农业机械学报，2022，53(04)：293-303.

[8] 金小俊，孙艳霞，于佳琳，等. 基于深度学习与图像处理的蔬菜苗期杂草识别方法[J]. 吉林大学学报(工学版)，2023，53(08)：2421-2429.

[9] 冀汶莉，刘洲，邢海花. 基于 YOLO v5 的农田杂草识别轻量化方法研究[J]. 农业机械学报，2024，55(01)：212-222+293.

[10]沈同平，王元茂，黄方亮，等. 基于深度学习技术的在线教学效果评价研究[J]. 河北北方学院学报(自然科学版)，2021，(03)：49-54.

[11]李莎莎. 基于深度回归神经网络的小目标和遮挡目标检测算法研究[D]. 河南大学，2022.

[12]屈志坚，高天姿，池瑞，等. 基于改进的 YOLOv3 接触网鸟巢检测与识别[J]. 华东交通大学学报，2021，(04)：78-86.

[13]严钦涛. 基于深度迁移学习的单样本人脸识别算法设计与 FPGA 验证[D]. 东南大学，2021.

[14]赛斌. 面向大规模人群行为规律挖掘的视频数据结构化与行人特征提取[D]. 国防科技大

学, 2021.

[15]Chien-Yao Wang, I-Hau Yeh, Hong-Yuan Mark Liao. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information[J]. arXiv preprint, 2024, 2402.13616.

[16]Wang P, Tang Y, Luo F, et al.Weed25: A deep learning dataset for weed identification[J]. Front. Plant Sci., 2022, 13:1053329.

附录 A　实时检测算法源代码

```
import win32gui
import torch
import cv2
import numpy as np
from PIL import ImageGrab
from utils.augmentations import letterbox
from utils.general import (non_max_suppression, scale_coords)
from models.experimental import attempt_load
# 置信率
conf_thres = 0.2
# 默认 0.45 效果较好
iou_thres = 0.5
# RGB  用来画框的颜色
color = (0, 255, 0)
# 权重文件名
weights = 'best.pt'
# 目标窗口类名
wnd_name = 'QQMusic_Daemon_Wnd'
# 找到目标窗口
hwnd = win32gui.FindWindow(wnd_name, None)
# 获取目标窗口位置-屏幕上
x1, y1, x2, y2 = win32gui.GetWindowRect(hwnd)
#x1, y1, x2, y2 =11, 224, 1281, 973
width = x2 - x1
height = y2 - y1
# 识别矩形区域位置
rect = (x1, y1, x2, y2)
def run():
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    print(device)
    # 判断设备类型并仅使用一张 GPU 进行测试
    half = device != 'cpu'
    # 载入模型
    model = attempt_load(weights, device=device)
    # 将模型的 stride 赋给 stride 变量 32
    stride = max(int(model.stride.max()), 32)  # model stride
    model.half()  # to FP16
    while True:
        # 记录识别对象坐标
        tars = []
        # 截取目标区域图像
        im = ImageGrab.grab(bbox=rect)
```

```
            # 将图像数据转换成 np 数组
            img0 = np.array(im)
            # 将图像缩放到指定尺寸
            img = letterbox(img0, stride=stride)[0]
            # 函数将一个内存不连续存储的数组转换为内存连续存储的数组,使得运行速度更快
            img = np.ascontiguousarray(img)
            # 把数组转换成张量,且二者共享内存
            img = torch.from_numpy(img).to(device)
            img = img.half() if half else im.float()
            # 压缩数据维度
            img /= 255  # 0 - 255 to 0.0 - 1.0
            if len(img.shape) == 3:
                img = img[None]
            # 对 tensor 进行转置
            img = img.permute(0, 3, 1, 2)
            # Inference 模型推理
            pred = model(img, augment=False, visualize=False)[0]
            # NMS 非极大值抑制 即只输出概率最大的分类结果
            pred = non_max_suppression(pred, conf_thres, iou_thres)
            # 处理预测识别结果
            for i, det in enumerate(pred):  # per image
                if len(det):
                    # Rescale boxes from img_size to im0 size
                    det[:, :4] = scale_coords(img.shape[2:], det[:, :4], img0.shape)
.round()
                    # 框选出检测结果
                    for *xyxy, conf, cls in reversed(det):
                        cv2.rectangle(img0, (int(xyxy[0]), int(xyxy[1])), (int(xyxy
[2]), int(xyxy[3])), color, 3)
                        # 添加识别对象坐标
                        tars.append((int(xyxy[0]), int(xyxy[1]), int(xyxy[2]), int(
xyxy[3])))
            # 调用 CV 显示结果图
            b, g, r = cv2.split(img0)
            image_1 = cv2.merge([r, g, b])
            resized_img = cv2.resize(np.array(image_1), (width, height))
            # 设置窗口可调整大小
            cv2.namedWindow("display", cv2.WINDOW_NORMAL)
            cv2.imshow("display", resized_img)
            cv2.waitKey(1)
    if __name__ == "__main__":
        run()
```

附录 B　系统部署机器人端算法源代码

```
import requests
import time
import cv2
url = 'http://192.168.31.118:5000/upload'
video_path = 'cscs.mp4'
cap = cv2.VideoCapture(video_path) # 打开视频文件 参数0表示默认摄像头
'''
if not cap.isOpened():
    print("Error: Could not open camera.")
    exit()
'''
frame_count = 0
while True:
    start_time = time.time()
    ret, frame = cap.read()
    if not ret:    #如果读取帧失败，退出循环
        break
    _, img_encoded = cv2.imencode('.jpg', frame) #将帧编码为JPEG格式
    img_bytes = img_encoded.tobytes() #将JPEG图像转换为字节数据
    files = {'file': img_bytes}  #将图片字节数据发送POST请求
    data = {'device': 0 ,'frame': frame_count}
    response = requests.post(url, files=files,data=data)
    if response.status_code == 200:
        end_time1 = time.time()
        execution_time_ms1 = (end_time1 - start_time) * 1000
        post_result=response.json()
        inference_result=post_result['result']
        print(f"ALL Delay Time Consuming: {execution_time_ms1:.2f} ms")
        print(f"Frame {frame_count} uploaded successfully.")
        print('Inference Result:', inference_result)
    else:
        print(f"Error uploading frame {frame_count}. Status code: {response.sta
tus_code}")
    frame_count += 1
# 释放视频对象
cap.release()
print("Video processing complete.")
```

附录 C　系统部署服务端算法源代码

```
import time
from PIL import Image
import torch
import numpy as np
from utils.augmentations import letterbox
from utils.general import (non_max_suppression, scale_coords)
from models.experimental import attempt_load
import io
from flask import Flask, request, jsonify
app = Flask(__name__)
@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return "No file part", 400
        file = request.files['file']
        m_device, frame_count=request.form.get('device', 'No Device'),request.form.get('frame', 'No Frame')
        if file:
            image12 = file.read()
            start_time = time.time()
            inference_data = process_image(image12)
            end_time = time.time()
            # 计算执行时间（以毫秒为单位）
            execution_time_ms = (end_time - start_time) * 1000
            print(f"ALL Recognition Time Consuming: {execution_time_ms:.2f} ms")
            response_data={'device':m_device,'frame':frame_count,'result':inference_data}
            return jsonify(response_data)
    return ''' '''
conf_thres = 0.2
iou_thres = 0.5
weights = 'best.pt'
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
half = device != 'cpu'
model = attempt_load(weights, device=device)
stride = max(int(model.stride.max()), 32)  # model stride
model.half()  # to FP16
def process_image(img):
    start_time2 = time.time()
```

```
        # 将二进制图像数据转换为 PIL 图像对象
        image = Image.open(io.BytesIO(img))
        # 将 PIL 图像对象转换为 NumPy 数组
        img0 = np.array(image)
        # 记录识别对象坐标
        tars = []
        # 将图像缩放到指定尺寸
        img = letterbox(img0, stride=stride)[0]
        # 函数将一个内存不连续存储的数组转换为内存连续存储的数组,使得运行速度更快
        img = np.ascontiguousarray(img)
        # 把数组转换成张量,且二者共享内存
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()
        # 压缩数据维度
        img /= 255  # 0 - 255 to 0.0 - 1.0
        if len(img.shape) == 3:
            img = img[None]
        # 对 tensor 进行转置
        img = img.permute(0, 3, 1, 2)
        end_time2 = time.time()
        # 计算执行时间（以毫秒为单位）
        execution_time_ms2 = (end_time2 - start_time2) * 1000
        print(f"Pretreatment Time Consuming: {execution_time_ms2:.2f} ms")
        # Inference 模型推理
        pred = model(img, augment=False, visualize=False)[0]
        # NMS 非极大值抑制 即只输出概率最大的分类结果
        pred = non_max_suppression(pred, conf_thres, iou_thres)
        # 处理预测识别结果
        for i, det in enumerate(pred):  # per image
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], img0.shape).ro
und()
                for *xyxy, conf, cls in reversed(det):
                    # 添加识别对象坐标
                    tars.append([int(xyxy[0]), int(xyxy[1]), int(xyxy[2]), int(xyxy
[3])])
        return tars
    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000 , debug=True)
```

致　谢

时光荏苒，四年的大学时光转瞬即逝。回顾这段美好的学习和生活经历，我心中充满了感激之情。在此，我要向那些在我成长道路上给予帮助和支持的人们表达最诚挚的感谢。

首先，我要感谢我的导师庞牧野教授。在论文的选题、研究方法以及写作过程中，庞牧野教授给予了我无微不至的指导和关怀。他渊博的学识、严谨的治学态度以及对学生的耐心指导，使我在学术研究的道路上受益匪浅。

其次，我要感谢各位授课老师和学院的教职工们。是你们在课堂上辛勤的教学和无私的奉献，使我得以不断充实自己的知识储备，开阔了视野，培养了科学的思维方式和严谨的学习态度。

同时，我也要感谢我的同学和朋友们。在这四年里，我们一起讨论问题、分享经验、相互鼓励。特别是在论文写作期间，大家给予了我许多宝贵的建议和帮助，使我能够顺利完成这篇论文。

最后，我要特别感谢我的家人。你们一直以来的理解、支持和鼓励，是我前进道路上最大的动力。无论遇到什么困难，你们始终是我坚强的后盾，让我能够勇敢地面对挑战。

在此，谨向所有关心、支持和帮助过我的人们，致以最诚挚的谢意！