Graduation Project (Thesis) of Wuhan University of Technology

Research on weed recognition algorithm for dryland weeding operation

School (Department): School of Automation

Specialized Class: Automation Class 2005

name of student ： Yin Yida

faculty adviser ： Pang Muye

Declaration of Originality for Academic Dissertation

I hereby solemnly declare that the submitted thesis represents my independent research achievement under the guidance of my supervisor. Except for the content explicitly cited in the text, this thesis does not include any other published or authored works by individuals or groups. I fully acknowledge that I shall bear all legal consequences of this declaration.

Author signature:

Year Month Day

Copyright License for Academic Theses

The author of this thesis fully understands the university's regulations regarding the protection and use of academic theses. I hereby consent to the university retaining and submitting copies and electronic versions of this thesis to relevant thesis management departments or institutions, and authorize access and borrowing of the thesis. I authorize the provincial-level outstanding bachelor's thesis selection committee to include all or part of this thesis in relevant databases for retrieval, and may preserve and compile this thesis through reproduction methods such as photocopying, microfilm, or scanning.

This thesis is classified as Confidential. This authorization shall remain valid until the year of declassification.

2. Not confidential (.

(Please check the corresponding box above)

Author signature: Year Month Day

Signature of Supervisor: Year Month Day

Author signature: Year Month Day

Signature of Supervisor: Year Month Day

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

catalogue

Chapter 1 Introduction

1.1 Research Background and Significance

Weeds refer to plants that are different from the current crops and spread rapidly, having a profound impact on agricultural production and the ecological environment. Especially in China, a major agricultural country, the weed problem is particularly prominent. China is vast and rich in resources, with diverse climate types, creating a wide range of environments suitable for weed growth. Weeds are diverse in species and widely distributed, almost covering all crop-producing areas in the country. In the northeastern regions of Heilongjiang and Jilin, the main weeds are Galium aparine and Xanthium strumarium; in the northern regions, wild oats (Avena fatua) and Cirsium arvense are common; in the Yangtze River basin, Echinochloa crus-galli and Setaria viridis are widely distributed; in the southern regions, Eleusine indica and Echinochloa crusgalli are predominant. These weeds not only compete with crops for water, nutrients, and light but also lead to reduced yields or even total crop failure.

The hazards of weeds extend beyond agricultural production. They also pose threats to the ecological environment and impact biodiversity. For instance, the spread of Eupatorium adenophorum in southwestern regions has caused severe damage to local ecosystems. These invasive weeds not only alter the original vegetation structure but also negatively affect soil properties and water resources.

The struggle between humans and weeds can be traced back to the origin of agricultural civilization. In the earliest days, humans mainly relied on manual weeding, using stone tools and wooden sticks as tools to clear weeds from the fields through manual uprooting. With the development of agricultural technology, some simple weeding tools gradually emerged, such as hoes and sickles. In ancient China, the book "Qi Min Yao Shu" (Essential Techniques for the Common People) detailed various weed removal methods and tools.

In modern times, the invention and widespread application of chemical herbicides have greatly improved weed control efficiency. In the early 20th century, American scientists first synthesized 2,4-D (a commonly used selective herbicide), which effectively kills broadleaf weeds while having almost no impact on grass crops. China also began introducing and using chemical herbicides in the 1950s, significantly reducing farmers' labor intensity and improving agricultural productivity. However, the long-term use of chemical herbicides has also brought about environmental pollution and weed resistance issues.

With increasing emphasis on environmental protection and sustainable development, biological control and ecological weeding methods have garnered attention. Biological control utilizes natural enemies (such as weed pests, diseases, and competitive plants) to suppress weed growth, whereas ecological weeding involves regulating the structure of biological communities within agricultural ecosystems to inhibit weed proliferation.

In the 21st century, with the rapid development of artificial intelligence and robotics, intelligent weeding robots have gradually become a new favorite in the agricultural sector. Equipped with cameras and sensors, these robots can monitor the distribution of weeds in the field in real time and utilize machine learning and computer vision technologies to achieve precise weeding.

Weed recognition technology is a core component of intelligent weeding robots. By analyzing field images, these robots can accurately identify various types of weeds and select optimal weeding strategies based on their growth patterns and distribution characteristics. The YOLOv5 (You Only Look Once version 5) target detection algorithm, widely used in weed recognition, is a real-time deep learning-based system renowned for its efficiency and precision. It enables rapid weed identification across large-scale fields. Automated weed recognition systems reduce reliance on manual labor, enhance weeding efficiency, and lower agricultural production costs. Intelligent weeding robots can autonomously identify and locate weeds, allowing precise pesticide application or mechanical weeding. This approach minimizes chemical herbicide usage, thereby mitigating environmental pollution and soil degradation.

1.2 Current Research Status at Home and Abroad

1.2.1 Current Status of International Research

As early as the 1980s, the United States initiated research on weed identification, which was later extended to outdoor environments, achieving technological advancements from static detection to real-time tracking. This evolution not only enhanced the efficiency of agricultural automation but also laid the foundation for smart agriculture. This article will chronologically detail the progress of weed identification research across different stages in foreign countries, analyzing the research content, technical features, application outcomes, and their respective advantages and disadvantages at each stage.

In 2004, researchers including Zhang Jichen from Aarhus University in Denmark developed HortiBot, a robotic weeding system that utilized pattern recognition algorithms to distinguish crops from weeds, demonstrating the potential of image processing technology in agriculture. This innovation not only showcased the prospects of robotics in farming but also provided valuable research directions and technical insights for future developments. Subsequently, in 2017, Hall D[1] and colleagues at Queensland University of Technology in Australia created AgBotII, a modular weeding robot that identified crops and weeds through color analysis while employing both chemical and mechanical methods for weed control, integrating multiple technological approaches. Through this diversified technical strategy, AgBotII achieved more efficient weed management while minimizing environmental impact.

Furthermore, in 2017, Yalcin[2] and colleagues employed pre-trained Convolutional Neural Networks (CNNs) to classify 16 plant species, achieving superior recognition accuracy compared to traditional Support Vector Machine (SVM) models. This further demonstrated the advantages of deep learning techniques in plant classification. The pre-trained model leveraged extensive existing data to enhance initial performance, resulting in enhanced performance on specific tasks. In 2018, Milioto[3] and colleagues applied the NDVI color index and Convolutional Neural Network (CNN) classifier, achieving weed recognition accuracies of 99.42% and 99.66%, respectively, showcasing the powerful capabilities of deep learning in high-precision identification. The application of the NDVI color index enabled quantification of plant health and growth status, while the Convolutional Neural Network significantly improved recognition efficiency and accuracy.

In 2020, Sabzi[4] and colleagues developed a gray-level conjugate matrix (GLCM)-based model that extracted multiple features and compared the performance of different algorithms and classifiers in distinguishing crops from weeds, thereby further optimizing recognition accuracy. This study demonstrated the critical role of feature extraction and algorithm optimization in enhancing recognition performance. Also in 2020, Hu[5] and team evaluated their recognition model using the Deep Weeds dataset and combined it with multiple other datasets to improve recognition efficiency, highlighting the importance of dataset diversity in model training. By utilizing diverse datasets, the model can better adapt to different environments and conditions, thereby enhancing the reliability of practical applications.

1.2.1 Current Domestic Research Status

In recent years, with the continuous advancement of agricultural intelligence, weed recognition technology has garnered widespread attention and achieved significant progress in domestic research. This paper will chronologically detail the advancements in weed recognition research across various stages in China, analyzing the research content, technical characteristics, application outcomes, as well as the advantages and disadvantages of each stage.

In 2019, Peng Mingxia [6] and colleagues proposed a Faster R-CNN method incorporating FPN for vertical-captured cotton weed image recognition under natural lighting. The feature pyramid network (FPN) enhances the model's detection capability for targets of varying sizes through multi-scale feature fusion. Training and testing on a dataset of 1,000 images achieved an average recognition accuracy of 95.5%, with the optimized Faster R-CNN outperforming YOLOv3 in small target detection. This demonstrates that FPN offers significant advantages in target detection under complex backgrounds, particularly for weed detection tasks common in agricultural scenarios.

In 2022, Zhai Changyuan [7] and colleagues developed the YOLO mdw model for cabbage recognition, leveraging YOLOv5s and MobileNet v3s. The model achieved over 93% accuracy across various weather conditions while reducing processing time by 26.98% compared to YOLOv5s. This study highlights the potential of lightweight models in real-time applications, particularly in agricultural settings requiring rapid processing and high accuracy. MobileNet v3s' low computational demands and high efficiency enable it to perform excellently on resource-constrained devices.

In 2022, Jin Xiaojun [8] and colleagues developed a weed recognition method for vegetable seedlings, using neural networks to identify leafy greens while segmenting weeds through color features. The SSD model demonstrated optimal performance in both detection speed and recognition accuracy, achieving an F1 score of 95.4% and average precision of 98.1%. By integrating deep learning with traditional image processing techniques and employing color feature preprocessing, this approach significantly enhanced model efficiency. The combined methodology holds broad application potential in large-scale agricultural scenarios.

In 2023, Ji Wenli[9] and colleagues developed a lightweight weed recognition method based on YOLOv5, employing MSRCR preprocessing and the lightweight PP-LCNet network. By integrating Ghost convolution and attention modules, the approach significantly improved recognition performance and efficiency, achieving an average accuracy of 97.8% while drastically reducing memory consumption. The

MSRCR (Multi-scale Retinal Contrast Restoration) preprocessing enhanced image contrast and detail, enabling the model to maintain high accuracy even in complex backgrounds. The incorporation of PP-LCNet and Ghost convolution further reduced computational complexity and memory usage, allowing the method to operate efficiently on resource-constrained agricultural equipment.

1.3 Technical Approach and Research Content

1.3.1 Technical Approach

The technical route of this paper is shown in Figure 1.1.

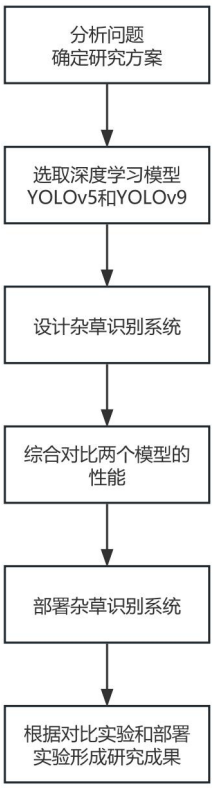


Figure 1.1 Technical Roadmap Flowchart

1.3.2 Research Objectives

One of the primary objectives of this study is to develop a deep learning-based weed recognition system and identify optimal deployment strategies to ensure the

system meets all requirements for production environments, thereby providing reliable technical support for intelligent weeding robots. The second objective is to evaluate the performance of the 24-year-old YOLOv9 model in weed recognition tasks, aiming to offer practical references for researchers and users of new models. The specific research goals include:

(1) Conduct comparative experiments using YOLOv5 and YOLOv9 for weed recognition tasks.

(2) Establish a weed image dataset. Based on the characteristics of weed recognition tasks, perform hyperparameter optimization and training for the YOLO model. Design a dynamic recognition algorithm to enable the model to conduct dynamic detection.

(3) Deploy the weed recognition system, identify optimal deployment strategies, and ensure the system fulfills all requirements for production environments.

1.3.3 Structure of the Thesis

To investigate weed recognition for automated dryland weeding robots and evaluate YOLOv9's performance in this task, we conducted comprehensive research covering algorithm selection, dataset construction, comparative experiments, and real-time detection algorithm development. This study assessed the feasibility of YOLO series models for automated dryland weeding, tested the efficiency and quality of the new model, and deployed a weed recognition system that fully meets all operational requirements in production environments.

Chapter 1, Introduction. This chapter introduces the research background and current status of weed identification and positioning for dryland weeding operations, both domestically and internationally, and defines the technical approach and research objectives.

Chapter 2: Deep Learning Theory. This chapter systematically introduces the fundamental architectures and operational principles of convolutional neural networks, YOLOv5, and YOLOv9, with the selection of YOLOv5 and YOLOv9 for comparative experiments.

Chapter 3 presents the design of a YOLO-based weed recognition system, detailing the data set construction process, model hyperparameter tuning, evaluation methods, and real-time detection algorithm implementation.

Chapter 4 compares the performance of YOLOv5 and YOLOv9 in weed recognition tasks. This chapter sequentially evaluates their training metrics and recognition performance, followed by a discussion and analysis.

Chapter 5 details the deployment of a weed recognition system based on YOLOv5. This chapter elaborates on the deployment strategy and algorithms, followed by rigorous testing of the system.

Chapter 6, Summary and Outlook. This section summarizes the research content and achievements presented above, and outlines future research directions.

Chapter 2: Deep Learning Theory

2.1 convolutional neural network

2.1.1 Introduction to Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are specialized deep learning models designed for processing grid-topology data such as images and videos. Unlike traditional Fully Connected Neural Networks (FCNNs), CNNs extract local features using convolutional layers and pooling layers, and learn more complex feature representations by stacking multiple convolutional layers. CNNs excel in computer vision tasks including image classification, object detection, and image segmentation.

Convolutional neural networks (CNNs) are widely used in weed recognition due to their ability to efficiently process image data and automatically extract features. Weed identification involves distinguishing weeds from crops in complex natural environment images, which requires robust feature extraction capabilities. By combining convolutional layers, pooling layers, and fully connected layers, CNNs can capture spatial features such as edges, textures, and shapes in images, significantly improving recognition accuracy. Additionally, the parameter sharing and sparse connection characteristics of CNNs reduce computational complexity, making them more efficient when processing large-scale datasets. Using CNNs enables efficient and accurate weed identification, helping to enhance agricultural automation levels and reduce weeding costs.

2.1.2 Basic Structure of Convolutional Neural Networks

The fundamental architecture of CNN comprises an input layer, multiple convolutional layers, a pooling layer, and a fully connected layer [10]. The input layer receives raw data such as pixel values from images. Convolutional layers extract local features through convolutional operations, while the pooling layer reduces feature map dimensions via downsampling to lower computational complexity. The fully connected layer maps high-dimensional features to output categories, calculating probability distributions for each category using the Softmax activation function [11].

The core operation of a convolutional layer is convolution. For two-dimensional image inputs, the convolution operation can be described as:

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

Here, represents $IK(i,j)$the input image, denotes the convolution kernel, and indicates the position of the output feature map, with m and n being the indices of the convolution kernel. The convolution kernel generates the feature map by performing dot product operations with local regions of the input image. The parameters of the convolution kernel are learned through backpropagation to extract useful features.

The pooling layer reduces the size of feature maps through downsampling operations. Common pooling methods include max pooling and average pooling. Max pooling selects the maximum value within the pooling window, while average pooling calculates the average value of the window. For example, max pooling is defined as:

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

Here, the $PI(i,j)$first represents the pooled feature map, the second the input feature map, and the third the output position index, where m and n denote the position indices of the pooling window.

The fully connected layer maps the high-dimensional feature vectors from the convolutional and pooling layers to the final class labels. Its output is normalized using the Softmax activation function, yielding probability distributions for each category.

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

Here, is $z_i$the score of the i-th category, and n is the total number of categories.

2.1.3 Working Principle of Convolutional Neural Networks

The fundamental workflow of convolutional neural networks (CNNs) is illustrated in Figure 2.1. CNNs achieve hierarchical representation of input data by progressively extracting features through multiple layers. The input layer first receives raw image data and applies normalization processing. Subsequently, convolutional layers perform convolution operations using multiple convolution kernels to extract low-level features such as edges and textures. As the convolution layers deepen, the extracted features become increasingly abstract and complex, including shapes and object components [12].

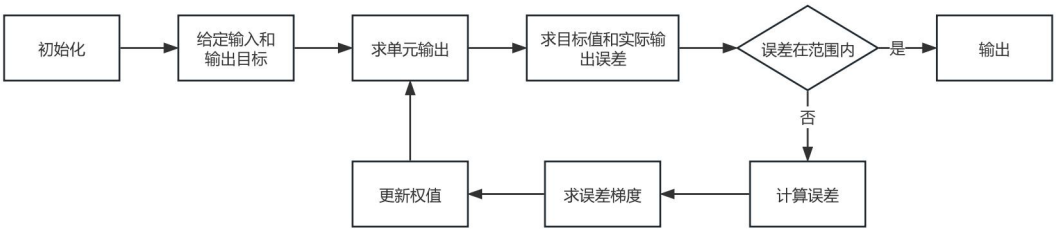


Figure 2.1 Workflow diagram of convolutional neural network

The pooling layer reduces the feature map size through downsampling while retaining key features, thereby lowering computational complexity. Through repeated convolution and pooling operations, the feature map's dimensions gradually decrease while its expressive power increases. Finally, the fully connected layer maps the extracted high-dimensional features to output categories, calculating probabilities for each category via the Softmax function to accomplish classification tasks.

CNN's training process employs backpropagation and gradient descent optimization algorithms. The forward propagation phase calculates layer outputs until the final prediction is obtained. The loss function (e.g., cross-entropy loss) then measures the error between predictions and true labels. The backpropagation algorithm computes gradients for each parameter based on this error, while the gradient descent algorithm updates parameters to progressively reduce the loss function value, thereby continuously improving model performance [13].

Convolutional neural networks (CNNs) excel in computational efficiency and feature extraction for high-dimensional data due to their parameter sharing and local connection mechanisms. Consequently, CNNs have been widely adopted in computer vision, driving advancements in tasks such as image recognition and object detection.

2.2YOLOv5 algorithm

2.2.1 Introduction to YOLOv5

YOLO (You Only Look Once) is an object detection algorithm developed based on CNN. Unlike traditional sliding window methods, YOLO employs an end-to-end detection approach. Specifically, it divides images into multiple grids, with each grid predicting multiple bounding boxes and their class probabilities. This process relies on CNN for image feature extraction. YOLOv5, launched in 2020, is an advanced object detection algorithm characterized by high speed and accuracy, making it particularly suitable for weed recognition tasks. YOLOv5 can simultaneously detect multiple weed targets in a single forward propagation and precisely locate their positions, meeting

real-time detection requirements. Its efficient convolutional neural network architecture ensures smooth operation even on resource-constrained agricultural equipment.

2.2.2 Basic Structure of YOLOv5

The network architecture of YOLOv5 comprises three primary components: the backbone network, neck network, and head network, each with distinct design and functional characteristics [14].

The backbone architecture employs CSP-Darknet53, an enhanced version of the Darknet53 model originally used in YOLOv3. By introducing the Cross-Stage Partial Network (CSPNet) strategy, CSP-Darknet53 achieves significant reductions in computational load and parameter size while maintaining high performance. Specifically, CSPNet divides the feature maps from the base layer into two parts and then fuses them through cross-stage processing. This approach not only preserves the feature reuse advantage of DenseNet but also reduces redundant gradient information by truncating gradient flows, thereby improving computational efficiency and inference speed. Mathematically, the CSP operation can be expressed as:

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

Here, and $X_1 X_2 F H$are two input feature maps, and are two distinct convolution operations.

The Neck network achieves multi-scale feature fusion by combining Spatial Pyramid Pooling Fast (SPPF) and the improved Path Aggregation Network (CSP-PANet).

The SPPF module significantly expands the receptive field through multi-scale pooling of input features without substantially increasing computational overhead. Specifically, it enhances computational efficiency by sequentially passing the output of the maximum pooling layer to the next maximum pooling layer. The formula is:

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

The CSP-enhanced YOLOv5 version of PANet further refines feature fusion. By incorporating a CSP layer into PANet, the feature map stitching process becomes more

efficient, significantly improving multi-scale feature integration. This design boosts the model's detection accuracy while accelerating processing speed.

The Head network, inheriting the design of YOLOv3 and YOLOv4, generates bounding boxes, objectness scores, and class probabilities. It performs predictions through multiple convolutional layers, producing outputs at three different scales to accommodate targets of varying sizes. The specific bounding box prediction formula is as follows:

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

Here, and $b_x b_y b_w b_h$ $t_x t_y t_w t_h$ $c_x c_y p_w p_h$ are the center coordinates of the bounding box, and are its width and height,,,, are the network outputs, and are the top-left coordinates of the grid cell, and are the anchor box dimensions.

YOLOv5 employs the SiLU (Sigmoid Linear Unit) activation function in its hidden layers, defined as:

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

The Sigmoid $\sigma(x)$function is used here. In the output layer, the traditional Sigmoid activation function is applied to generate probability values.

YOLOv5 employs a composite loss function to optimize the model, which includes Binary Cross Entropy (BCE) for calculating class probabilities and confidence scores, and Complete Intersection Over Union (CIoU) for bounding box localization. The total loss formula is:

$$Loss = \lambda_{class} \cdot BCE_{class} + \lambda_{obj} \cdot BCE_{obj} + \lambda_{bbox} \cdot CIoU_{bbox} \tag{2.8}$$

The weights $\lambda_{class}\lambda_{obj}\lambda_{bbox}$of category loss, confidence loss, and bounding box loss are respectively defined to ensure balanced contribution across all components in the total loss function.

2.2.3 Working Principle of YOLOv5

During training, images undergo preprocessing to meet the model's input requirements. This process involves image resizing and normalization, which standardize images with varying dimensions and pixel values to ensure consistent input for the model. Additionally, YOLOv5 employs a variety of data augmentation techniques:

Scaling enhances model robustness to scale variations by adjusting image dimensions and increasing data diversity. This augmentation method generates multiple training samples of different sizes through image stretching or compression without altering the original content, thereby improving the model's ability to handle scale changes.

Color Space Adjustments adjust the image's brightness, contrast, saturation, and hue. These adjustments enhance the model's adaptability under varying lighting conditions. The hue adjustment formula is:

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

This is the $I\Delta H$original image, and this is the tone adjustment coefficient. Adjustments to brightness, contrast, and saturation are similar, involving linear transformations of the original image using scaling factors.

Mosaic Augmentation, a groundbreaking data augmentation technique in YOLOv5, generates new images by randomly combining four original ones. This approach enables the model to observe a wider range of object distributions and backgrounds within a single training batch, effectively addressing the 'small object problem' —a common challenge where the model underperforms on small targets. The Mosaic augmentation formula is:

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

The four I_1, I_2, I_3, I_4image blocks are respectively.

The MixUp method generates new training samples by linearly interpolating two images and their labels. The formula is:

$$I' = \lambda I_a + (1 - \lambda) I_b \quad (2.11)$$

$$y' = \lambda y_a + (1 - \lambda) y_b \quad (2.12)$$

Here, and $I_a I_b y_a y_b \lambda$are two distinct images, and are their corresponding labels, while is a random number between 0 and 1 used to control the interpolation ratio.

The preprocessed images are fed into the model's backbone network CSP-Darknet53 to extract multi-scale features. By employing a cross-phase partial network strategy, CSP-Darknet53 effectively reduces computational load and parameter size while preserving critical feature information. These features then undergo multi-layer convolution operations in the backbone network before being transferred to the neck network for further fusion and multi-scale processing.

The neck network incorporates a fast spatial pyramid pooling (SPPF) and an enhanced path aggregation network (CSP-PANet). The SPPF module significantly expands the receptive field through multi-scale pooling of input features without substantially increasing computational overhead. CSP-PANet further optimizes feature fusion by introducing a CSP layer, thereby improving both detection accuracy and speed. The fused features are then fed into the head network to generate predicted bounding boxes, object confidence scores, and category probabilities.

In the head network, YOLOv5 performs prediction through multiple convolutional layers and outputs results at three different scales to accommodate objects of varying sizes. The model employs a composite loss function for optimization, incorporating category loss, confidence loss, and bounding box loss. By combining these loss functions with weighted coefficients, the model ensures both accuracy and robustness across multiple object detection tasks.

The reasoning process begins with images undergoing preprocessing steps identical to those used in training to ensure input consistency. After preprocessing, the images pass through the backbone network and neck network to extract and fuse multi-scale features. These features are then fed into the head network to generate predicted bounding boxes, object confidence scores, and category probabilities. Finally, the model performs non-maximum suppression (NMS) to eliminate overlapping predictions, retaining only the most confident bounding boxes as the final output.

2.3YOLOv9 algorithm

2.3.1 Introduction to YOLOv9

YOLOv9 (You Only Look Once, 9th Edition) is the latest real-time object detection model introduced by Chien-Yao Wang et al. in February 2024 [15]. Building upon the strengths of its predecessor, YOLOv9 incorporates the PGI and multi-task GELAN architectures, significantly enhancing the model's efficiency and accuracy.

2.3.2 Structural Innovation of YOLOv9

One of the core innovations of YOLOv9 is the Generalized Efficient Layer Aggregation Network (GELAN). GELAN combines the advantages of CSPNet (Cross-Phase Partial Network) and ELAN (Efficient Layer Aggregation Network) to design a novel architecture that significantly reduces parameter and computational costs while maintaining high accuracy.

CSPNet enhances gradient path planning by introducing cross-phase partial connections within the network, thereby improving feature extraction capabilities. Specifically, CSPNet segments feature maps at each stage, with some segments processed through residual blocks while others directly connect to the next stage, ensuring diverse and sufficient information flow. ELAN prioritizes inference speed by stacking convolutional layers for rapid computation. Its design enables aggregation transformation operations across multiple network branches, efficiently capturing multi-scale features.

Building upon CSPNet and ELAN, GELAN achieves a balance between model performance and resource consumption by incorporating Efficient Layer Aggregation Blocks and Computational Blocks. These modular components can be dynamically replaced according to computational resource constraints, enabling GELAN's architecture to scale across models and hardware platforms of varying scales.

Another groundbreaking innovation of YOLOv9 is its Programmable Gradient Information (PGI) technology, which tackles the reliability issue of gradients in lightweight models during training. PGI employs Auxiliary Reversible Branches to maintain reversible connections with inputs throughout training, ensuring gradients propagate without degradation. The Multi-level Gradient Integration technique further consolidates gradients across all branches, eliminating interference between auxiliary branches. This approach guarantees gradient reliability, particularly in lightweight

models, significantly enhancing training performance and accelerating model convergence.

2.3.3 Innovations in the Working Principle of YOLOv9

YOLOv9 inherits the efficient design principles of the YOLO series from its training and inference mechanisms, while further optimizing performance through the aforementioned innovations. During training, YOLOv9 employs PGI technology to enhance gradient reliability. The process involves: input images undergoing forward propagation through the GELAN architecture to extract features layer by layer; maintaining gradient integrity via reversible connections in auxiliary reversible branches; integrating gradients from different auxiliary branches using multi-layer gradient integration, followed by model parameter updates through backpropagation. This approach enables YOLOv9 to achieve faster training speeds and improved model convergence while maintaining high accuracy.

In the inference phase, YOLOv9 leverages the GELAN architecture's efficient feature extraction and inference capabilities to achieve real-time object detection. The process involves: normalizing input images to the model's required dimensions; extracting feature maps layer by layer through the GELAN architecture; and processing these feature maps in the Detection Head using YOLO's object bounding box regression and classification modules to generate detection results.

Compared to YOLOv5, YOLOv9's architectural and training innovations enhance precision and efficiency while maintaining real-time performance. Overall, YOLOv9's design provides a more efficient solution for real-time object detection. However, as YOLOv9 is the latest YOLO model, it lacks practical validation. Therefore, this study conducts synchronous training and comparative analysis between YOLOv5 (the most stable and accurate model in the YOLO series) and the newest YOLOv9. The comparative experiments serve dual purposes: assessing the new model's efficiency and quality in weed recognition tasks, and providing a fallback solution with YOLOv5 if YOLOv9 underperforms for weed-removal robot applications.

Chapter 3 Design of Weed Recognition System Based on YOLO

3.1 Dataset Construction

In the research of dryland weed identification, the construction of the dataset is crucial, as it determines the quality of model training and the accuracy of identification. Based on the requirements of intelligent weeding robots, the weed dataset must meet at least the following necessary conditions: (1) Weed photography conditions: The camera posture for photographing weeds must align with the camera posture of the weeding robot. (2) Weed species conditions: According to the aforementioned research on the species and distribution of China dryland weeds, we need to construct a dataset corresponding to the weed species. (3) Weed growth period conditions: To achieve the effect of weeding weeds at the budding stage, the weed dataset must at least cover the seedling stage (BBCH 00-09) and the seedling stage (BBCH 10-19).

This study relies on the open-source dataset Weed25 [16], which was constructed using rigorous scientific methods and high representativeness to meet the aforementioned requirements and various routine needs for model training. Below, we will introduce the source of the Weed25 dataset, data collection methods, annotation process, and data preprocessing to ensure data diversity and representativeness.

Figure 3.1 Images of selected samples from the dataset

Some images of the Weed25 dataset are shown in Figure 3.1. The image resources are sourced from fields and lawns in Chongqing, China, covering 25 common weed species in East Asia. These images were collected between October 2021 and August

2022 using a Nikon D5300 DSLR camera and a Huawei smartphone, with shooting heights and angles ranging approximately from 30-50 cm and 60°-90° to ensure vertical alignment with the weeds. The collection time spanned from 9:00 to 17:00 daily, including various lighting conditions such as sunny, cloudy, and rainy days, thereby representing the actual conditions in natural complex environments as closely as possible. The dataset contains 14,023 images, comprising 25 weed species from 14 families, with each family including multiple weed varieties. The dataset covers weed morphology at different growth stages, with most images collected during the second to ninth cotyledon stages (BBCH 12-19).

The dataset contains individual images with a resolution of 2992×2000 and an average file size of 1.85MB, totaling 27GB. Higher image resolution means more information can be captured, which helps improve the model's recognition accuracy. However, researchers with limited computing resources should compress the dataset to avoid excessive time consumption and potential memory shortages.

Weed25 only provided.xml format label files, whereas we required.txt format files for model training. Given that some images in the open-source dataset had incorrect annotations, we first imported all images and label files to roboflow.com for annotation verification. We manually corrected the mislabeled files and then exported all.xml files. Finally, we developed a Python script to convert all.xml files into.txt format.

Table 3.1 Structure of Weed25 dataset

a branch of academic or vocational study	name	tote	training set	validation set	test set
the grass family	Weeds	563	394	112	57
	Mangtang grass	594	415	118	61
	green bristlegrass	552	386	110	56
sedge family	nutgrass flatsedge	594	415	118	61
the composite family	Penthera	192	134	38	20
	Field thistle	565	395	113	57
	XanThium sibiricum	745	521	149	75
	Indian Feipeng	510	357	102	51
	sticktight	612	428	122	62
	Basella rubra	536	375	107	54

	L				
	Ageratum	599	419	119	61
Polygonaceae	Polygonum aviculare	671	469	134	68
	Japanese knotweed	490	343	98	49
	Polygonum cuspidatum	390	273	78	39
Amarantaceae	Hollow lotus seeds	637	445	127	65
	edible amaranth	742	519	148	75
Cruciferae	shepherd's purse	224	156	44	24
Portulacaceae	herba portulacae	730	510	146	74
Commelinaceae	Commelina communis L	562	393	112	57
Chenopodiaceae	lamb's-quarters	593	415	118	60
Plantaginaceae	plantago asitica Linn	556	389	111	56
Violaceae	Viola verecunda A	523	366	104	53
Solanaceae	black nightshade	606	424	121	61
Rosaceae	Indian strawberry	615	430	123	62
Malvaceae	piemarker	622	435	124	63
		14,023	9806	2796	1421

To ensure scientific model training, we divided the dataset into three sets in a 7:2:1 ratio: a training set of 9,806 images, a validation set of 2,796 images, and a test set of 1,421 images (as shown in Table 3.1). This data partitioning ensures adequate and representative data for training, validation, and testing, effectively preventing overfitting and enhancing the model's generalization ability.

3.2 Training Process Description

3.2.1 Overview of Training Environment

The deep learning system utilizes a high-performance desktop workstation. Powered by an 8-core, 16-thread 11th-gen Intel Core i7-11700F processor, it features 32GB of RAM for efficient data processing and multitasking. The GeForce RTX 4090

GPU delivers exceptional floating-point performance and large memory capacity, ideal for training and inference tasks in complex deep learning models. Running on Windows 10, the system employs Conda to maintain a consistent Python virtual environment, ensuring stable dependencies and package integrity.

In terms of software configuration, Python version 3.8.15 is used, with PyTorch as the deep learning framework. The YOLOv5 model is implemented using torch-1.12.1+cu116-cp38-cp38-win_amd64, while the YOLOv9 model employs torch-1.8.1+cu111-cp38-cp38-win_amd64. PyTorch is widely adopted in academic research and industrial applications for its flexibility and high performance, supporting various neural network architectures and optimization algorithms.

3.2.2 Hyperparameter Configuration and Optimization

The hyperparameter settings for YOLOv5 and YOLOv9 are presented in Table 3.2.

Table 3.2 Model Hyperparameter Configuration

model	YOLOv5s	YOLOv9-c
Parameters (M)	7.0	50.9
Batch size	40	10
epochs	70	70
Workers	3	3
Initial learning rate (lr0)	0.01	0.01
Final learning rate multiplier (lrf)	0.01	0.01
optimizer （optimizer）	SGD	SGD

The specific pre-trained model selected for YOLOv5 is YOLOv5s, while YOLOv9 employs the YOLOv9-c model. Given that YOLOv9-c has several times more parameters than YOLOv5s, their hyperparameter configurations differ. During training, we set the batch size to 30 for YOLOv5 and 10 for YOLOv9 to balance GPU memory usage and training efficiency, ensuring maximum data processing within the RTX 4090's resource limits. Both models were trained for 70 epochs—a setting validated through extensive experiments—to ensure adequate model training while preventing overfitting caused by overtraining.

Both YOLOv5 and YOLOv9 employ a learning rate scheduling strategy called OneCycleLR, which dynamically adjusts the learning rate to enhance model

convergence. The learning rate refers to the step size of parameter updates during each iteration. Properly configured learning rates can accelerate model convergence and help the model find the global optimum. The initial learning rate (lr0) is set to 0.01, indicating that the parameter update amplitude is relatively large in the early stages of training. This process can be expressed by Equation (3.1).

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

Here, represents $\theta_t \eta \nabla L(\theta_t)$the model parameter at iteration t, is the learning rate, and is the gradient of the loss function with respect to the model parameter. This formula indicates that the model parameter is updated at each iteration based on the current gradient and learning rate.

In the OneCycleLR learning rate scheduling strategy, the lrf parameter determines the final learning rate as a percentage of the initial learning rate. With lrf set to 0.01, the final learning rate is 1% of the initial rate. The OneCycleLR strategy dynamically adjusts the learning rate through three phases: first, a gradual increase from a low to a high value; then, maintaining the high value during the intermediate phase; and finally, a gradual decrease to a minimal value. This approach enables the model to rapidly approach the optimal solution in the early stages while making fine-tuned adjustments in later phases to enhance generalization performance. This process can be mathematically represented by Equation (3.2).

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

Here, is $lr_t lr_{max} T$the learning rate for the t-th iteration, is the maximum learning rate, and is the total number of iterations. This formula describes how the learning rate dynamically changes throughout the training process.

Both YOLOv5 and YOLOv9 employ the Stochastic Gradient Descent (SGD) optimizer, incorporating momentum and weight decay to enhance training efficiency and performance. Momentum, a key parameter in the optimizer, accelerates convergence while reducing oscillations. During parameter updates, momentum incorporates the previous gradient direction, allowing the model to maintain momentum during updates and achieve stable convergence near the valley. The model's momentum is set to 0.937, a high value that accelerates convergence while minimizing oscillations. This process can

be mathematically expressed through the momentum-based gradient descent formula (3.3).

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

where is $v_t \gamma \eta \nabla L(\theta_t)$the momentum, is the momentum factor (0.937), is the learning rate (lr0), and is the current gradient. This formula shows how the momentum combines the current gradient and the previous momentum during each parameter update.

Weight decay in YOLOv5 and YOLOv9 serves as a regularization technique to prevent model overfitting. By incorporating L2 regularization for parameter weights into the loss function, it effectively controls parameter value growth, thereby enhancing the model's generalization ability. In this model, the weight decay parameter is set to 0.0005. This process can be mathematically expressed through the weight decay formula (3.4).

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

The weight λdecay coefficient is 0.0005. This formula specifies how to apply weight decay during each parameter update.

To optimize data loading efficiency, we deployed three workers for multi-threaded preprocessing. This configuration fully leverages multi-core processor capabilities, significantly boosting loading speed while maintaining high GPU utilization during training. It effectively prevents training delays caused by data loading bottlenecks.

3.3 Design of Model Performance Evaluation Method

To evaluate the performance of the YOLO model, we implemented a series of precise methodologies and computational formulas to ensure its effectiveness and reliability in identifying weeds in dry fields. These evaluation approaches not only involve analyzing the model's output logs but also encompass the calculation of key metrics, with multiple visualizations provided through various curve plots to gain a comprehensive understanding of the model's behavior and performance.

3.3.1 Training Process Loss Function

We need to specify the loss functions in the model output log, including train/box_loss, train/obj_loss, and train/cls_loss during training, and val/box_loss, val/obj_loss, and val/cls_loss during validation.

(1) The metrics train/box_loss and val/box_loss measure the overlap between predicted and actual bounding boxes. Lower box_loss values indicate the model's improved accuracy in target localization. When both training and validation losses decrease progressively, it demonstrates continuous learning and refinement of target prediction. Conversely, if training loss declines while validation loss remains unchanged or even increases, overfitting may occur, reflecting the model's limited generalization ability to new data.

(2) The metrics train/obj_loss and val/obj_loss measure the confidence error in target detection. Lower obj_loss values indicate the model's improved accuracy in predicting target presence. When obj_loss decreases across both training and validation sets, it demonstrates the model's strong performance in target detection. Conversely, if obj_loss remains unchanged in the validation set, it may suggest the model struggles to identify targets in new data.

(3) The metrics train/cls_loss and val/cls_loss evaluate the accuracy of category prediction. Lower values of cls_loss indicate the model's enhanced ability to identify target categories. In single-category detection tasks, this loss component should be minimal. A higher cls_loss on the validation set suggests potential classification errors in real-world applications.

(4) The metrics train/dfl_loss and val/dfl_loss evaluate the model's performance in bounding box prediction accuracy. dfl_loss, based on distributed regression, improves bounding box localization precision by predicting the probability distribution of each coordinate. Lower dfl_loss values indicate better performance in fine-grained bounding box localization, with predicted coordinates more closely approximating real-world values.

By monitoring the changes of these loss functions, we can evaluate the convergence and generalization ability of the model.

3.3.2 Model Performance Evaluation Parameters

To comprehensively evaluate the model's recognition capability, we must calculate and analyze the following key metrics: Precision (P), Recall (R), and Mean Average Precision (mAP). These metrics help quantify the model's performance as detailed below:

(1) Precision (P) measures the proportion of all positive samples predicted by the model that are actually positive. The formula is:

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

Here, (True $TPFP$Positives) refers to the number of samples that are actually positive but correctly predicted as positive, while (False Positives) refers to the number of samples that are actually negative but incorrectly predicted as positive. A higher precision indicates that the model correctly predicts most positive samples with fewer false positives.

(2) Recall (R) measures the proportion of samples that are correctly predicted as positive among all actual positive samples. The formula is:

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

False Negatives FNrefer to samples that are actually positive but were incorrectly predicted as negative. A higher recall rate indicates the model effectively identifies most existing positive samples with minimal false negatives.

(3) Mean Average Precision (mAP) provides a comprehensive evaluation of the model's performance across different confidence thresholds. Specifically, mAP50 represents the average precision calculated at an IOU (Intersection Over Union) threshold of 0.5, while mAP50-95 denotes the average precision across multiple IOU thresholds (ranging from 0.5 to 0.95 with 0.05 increments). The IOU threshold serves as a metric for measuring the overlap between predicted and true bounding boxes.

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

Average Precision (AP) measures a detection network's performance in identifying a specific target category. A higher AP value indicates better overall detection performance. AP is calculated by plotting the Precision-Recall (P-R) curve and integrating the area under the curve. The AP calculation formula is:

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

Mean Average Precision (mAP) is the average of the average precision across all categories in the dataset. It is calculated as the sum of the average precision for all categories divided by the number of categories.

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

Here, denotes $AP_i N$mAPAPthe average precision of the i-th category, with the total number of categories. For a single detection category in weed recognition tasks, the computation can be simplified to evaluating just one category.

3.3.3 Model Performance Evaluation Curve

To provide a more intuitive analysis of model performance, we generate a series of curves, including the F1-Confidence curve, P-Confidence curve, R-Confidence curve, and P-R curve.

(1) The P-Confidence curve and R-Confidence curve demonstrate the precision-recall trade-off across different confidence thresholds (Confidence). We selected a range of confidence thresholds between 0 and 1 as independent variables. These curves help identify the optimal confidence threshold to balance precision and recall. Confidence threshold refers to the level of confidence in which a model predicts a detection as a positive sample. Higher confidence thresholds increase prediction certainty but may reduce precision, while lower thresholds improve recall at the expense of precision.

(2) The F1-Confidence curve displays the F1 values of the model at different confidence thresholds. F1 value is the harmonic mean of precision and recall, as shown in Equation (3.10).

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

The higher F1 value indicates that the model achieves a good balance between precision and recall.

(3) P-R Curve (Precision-Recall Curve) The P-R curve plots precision and recall values at different confidence thresholds, with recall (R) on the x-axis and precision (P) on the y-axis. This graphical representation demonstrates the model's performance across varying recall levels. By analyzing the curve, regions that maintain high precision at elevated recall levels indicate superior model performance.

3.4 Dynamic Recognition Algorithm Design

To meet the dynamic recognition requirements of weeding robots, this study developed a dynamic recognition algorithm based on YOLO for weed identification. By capturing and processing real-time video streams, the algorithm achieves real-time detection and localization of weeds in dry fields.

The implementation of the dynamic recognition algorithm relies on the following key steps: target window localization, image capture, image preprocessing, model inference, non-maximum suppression (NMS), and result display. The code snippets inserted in this paper are pseudocode for reference only, and the complete program is available in the source code in Appendix A.

(1) We locate the target window by invoking the FindWindow function in the win32gui library, then use the GetWindowRect function to obtain its screen coordinates. The coordinates of the window's upper-left and lower-right corners are (x1, y1) and (x2, y2), respectively, which determine the rectangular position rect of the recognition area.

(2) To capture image frames from real-time video streams, we utilize the ImageGrab.grab function in the PIL library to extract the current frame from the specified window area. The image is then converted into a NumPy array (img0) using the np.array function. This process provides foundational data for subsequent image processing and detection.

(3) To ensure the input images meet the requirements of the YOLO series models, we performed scaling and normalization processing. First, the image was resized to the model's required dimensions using the letterbox function while maintaining its aspect ratio. The adjusted image was then converted into a memory-continuous array to enhance processing speed. Next, the image data was transformed into a PyTorch tensor, with the use of half-precision floating-point (FP16) determined by the computing device type (CPU or GPU). The image data was subsequently normalized to the [0.0,1.0] range according to Equation (3.11). A batch dimension was added to accommodate the model's input requirements. The tensor dimensions were rearranged using the permute method to align with the model's channel order.

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

(4) After image preprocessing, the processed images are fed into the model for inference. The model generates an output tensor containing predicted bounding boxes

and category information through the forward propagation process. This step is the core of the entire recognition process, where the neural network calculates to identify potential target objects in the image.

(5) To enhance detection accuracy, non-maximum suppression (NMS) is applied to bounding boxes after model inference. The NMS algorithm retains the most probable detection results by setting confidence threshold conf_thres and IoU threshold iou_thres, while eliminating redundant bounding boxes. The core mechanism of NMS involves calculating the IoU ratio for each bounding box. By comparing IoU values between bounding boxes, it filters out redundant detections with high overlap, retaining only the most likely target locations. This step ensures the precision and reliability of the detection results.

(6) Finally, process the detection results. The scale_coords function rescales the bounding box from the model input dimensions back to the original image dimensions, re-mapping its coordinates to ensure detection accuracy. Using OpenCV's drawing function, each detection result is displayed as a rectangular box on the image, with the processed image shown through a window.

To process real-time video streams, the aforementioned steps are encapsulated in an infinite loop, where each frame is repeatedly executed, updated, and displayed in real time. Every frame undergoes a complete workflow encompassing capture, preprocessing, inference, and display, ensuring that the weeding robot can accurately detect and locate weeds in dry fields in real time.

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

Chapter 4 Performance Comparison of YOLOv5 and YOLOv9

4.1 Comprehensive Indicator Evaluation

4.1.1 Analysis of Evaluation Indicators for Test Sets

After the model training, the performance metrics are evaluated using the test set, with results presented in Table 4.1.

Table 4.1 Test Set Evaluation Results

model	YOLOv5s	YOLOv9-c
Accuracy (%)	0.94	0.847
recall (%)	0.969	0.815
Average precision (%)	0.983	0.895
F1 score	0.954	0.56
Parameter (M)	7.0	50.9
Training duration (min/epoch)	21.5	26.5

The evaluation results demonstrate that YOLOv5s achieves faster training speed than YOLOv9-c. Both models exhibit excellent precision rates, indicating high accuracy in detecting weeds with low false positive rates. Both models also demonstrate strong recall rates, meaning they can identify most weeds with low false negative rates. Both YOLOv5s and YOLOv9-c achieve superior recognition accuracy. YOLOv5s maintains high detection performance across various confidence thresholds, with an average precision score of 0.983. YOLOv9-c shows significant deviation from its mean absolute precision (mAP), indicating high F1 score fluctuations and unstable model accuracy across different confidence thresholds. In the comparative test set metrics, YOLOv5s outperforms YOLOv9-c in all evaluation metrics.

4.1.2 Analysis of Training Process Indicators

During the training process, the model generates 10 curves depicting how parameters evolve with the number of epochs. These charts provide profound insights into both the training process and the final performance of the model.

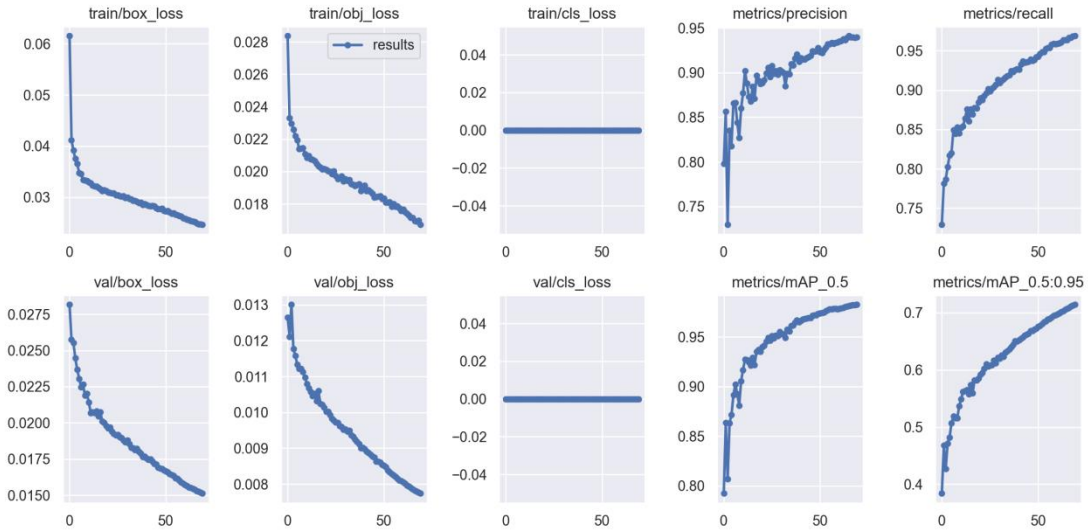


Figure 4.1 Metrics of the Weed Recognition Model Training Process Based on YOLOv5

As shown in Figure 4.1, both box_loss and obj_loss of YOLOv5s demonstrate a gradual decline and convergence during training and validation, indicating the model's continuous learning and improved prediction accuracy for target location and presence. The stable decrease in validation loss function reflects the model's strong generalization ability, maintaining high prediction accuracy on new data. Meanwhile, mAP_0.5 and mAP_0.5:0.95 show steady growth and convergence, confirming the training process's effectiveness. The recognition accuracy progressively increases, aligning with expectations.

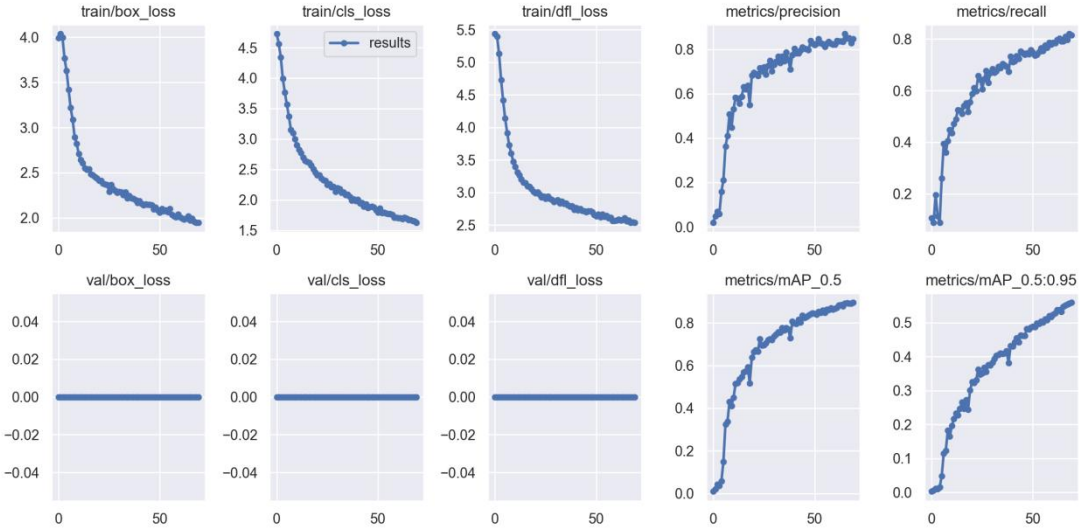


Figure 4.2 Metrics of the Weed Recognition Model Training Process Based on YOLOv9

Experimental results demonstrate that due to unknown reasons, validation loss data for YOLOv9 is missing, and thus this aspect is not discussed here. As shown in Figure 4.2, both box_loss and dfl_loss of YOLOv9-c during training show a gradual decline and convergence, indicating the model's continuous learning and improved target location prediction accuracy. Meanwhile, mAP_0.5 and mAP_0.5:0.95 demonstrate steady growth and convergence, confirming effective training. The recognition accuracy progressively improves, aligning with expectations.

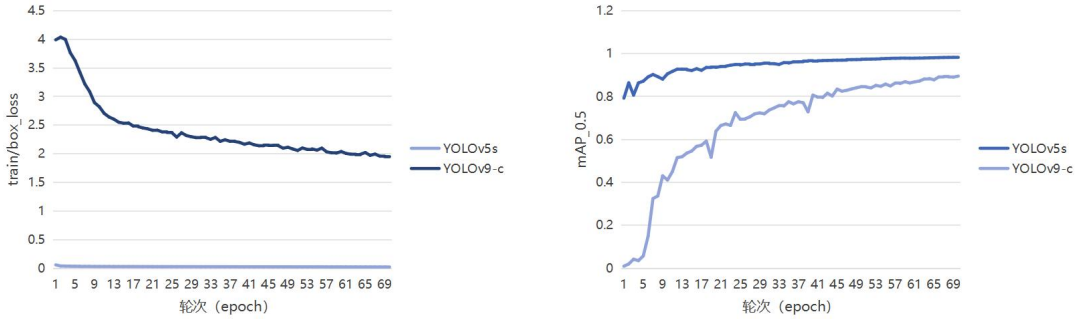
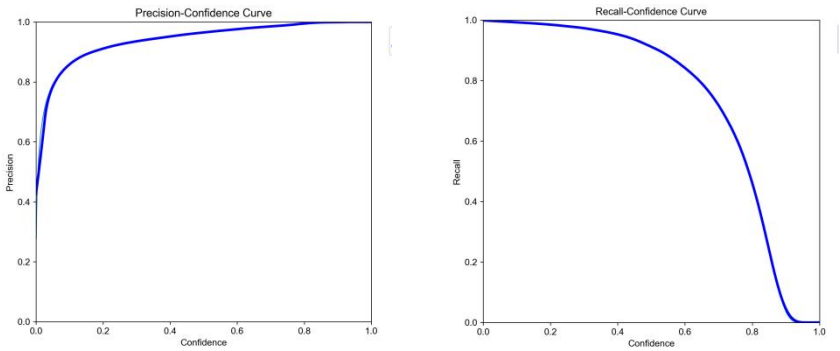


Figure 4.3 Comparison of Training Loss Function and Average Precision Mean

Figure 4.3 comparison shows that YOLOv5s achieves significantly lower training loss than YOLOv9-c, while YOLOv5 demonstrates substantially higher accuracy. The comparison also reveals that YOLOv5s converges faster than YOLOv9-c. In the benchmark experiments for training metrics, YOLOv5s outperforms YOLOv9-c across all metrics, with YOLOv9-c showing only marginal deficiencies.

4.1.3 Performance Evaluation Curve Analysis



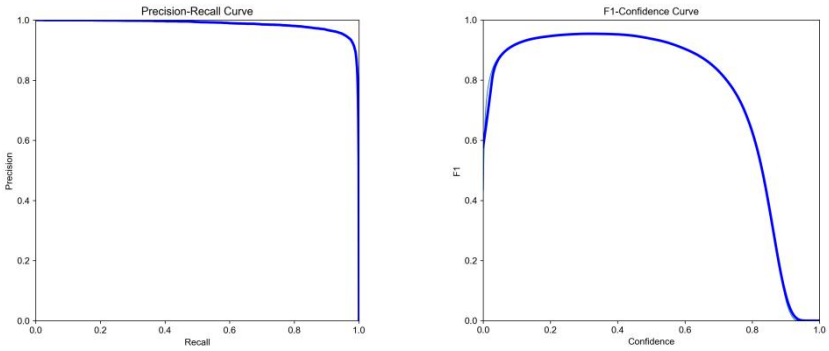


Figure 4.4 Visualization of Training Performance Metrics for YOLOv5-Based Weed Recognition Model

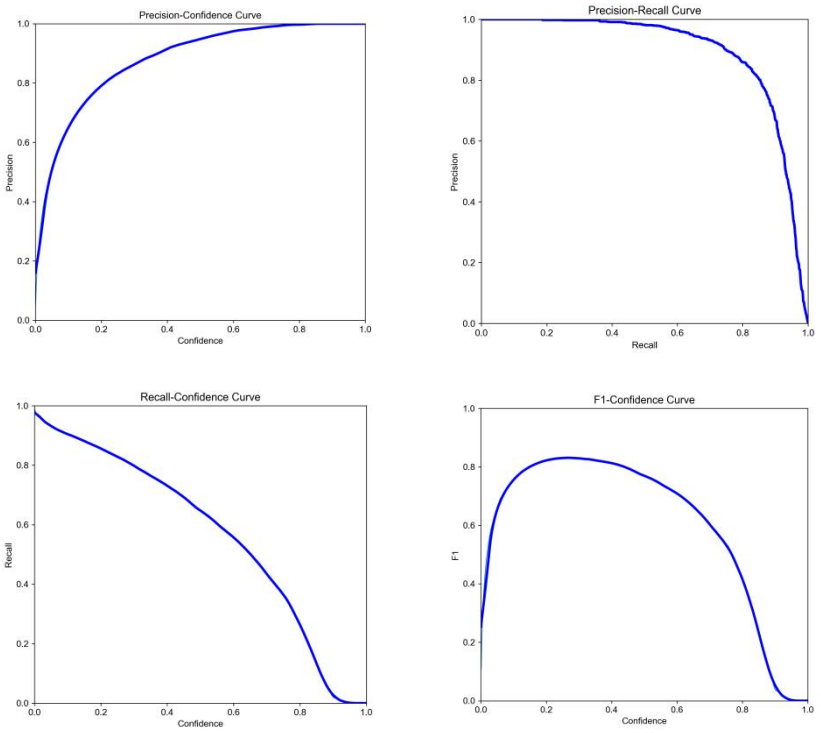


Figure 4.5 Visualization of Training Performance Metrics for YOLOv9-Based Weed Recognition Model

As shown in Figures 4.4 and 4.5, the F1 curves of YOLOv5s demonstrate that the model maintains high precision and recall rates across confidence thresholds from 0.1 to 0.5, indicating robust performance and stability. In contrast, the F1 curve of YOLOv9-c reveals that the model only achieves high precision and recall rates around a confidence threshold of 0.3, suggesting poor robustness and instability. The PR curves of YOLOv5s show that even at high recall rates (close to 1.0), the model maintains high accuracy, indicating it can detect nearly all targets with minimal false positives, demonstrating its superior performance in weed recognition tasks. The PR curves of YOLOv9-c, however,

reveal that accuracy decreases proportionally with increased recall rates, indicating a clear deficiency in maintaining both high precision and recall rates, which suggests suboptimal performance in weed recognition tasks. In the comparative experiments of training metrics, YOLOv5s outperforms YOLOv9-c across all evaluation metrics.

4.2 Identification Performance of the Observation Model

4.2.1 Static Recognition Performance

We evaluate the weed detection performance of two models using test set data.

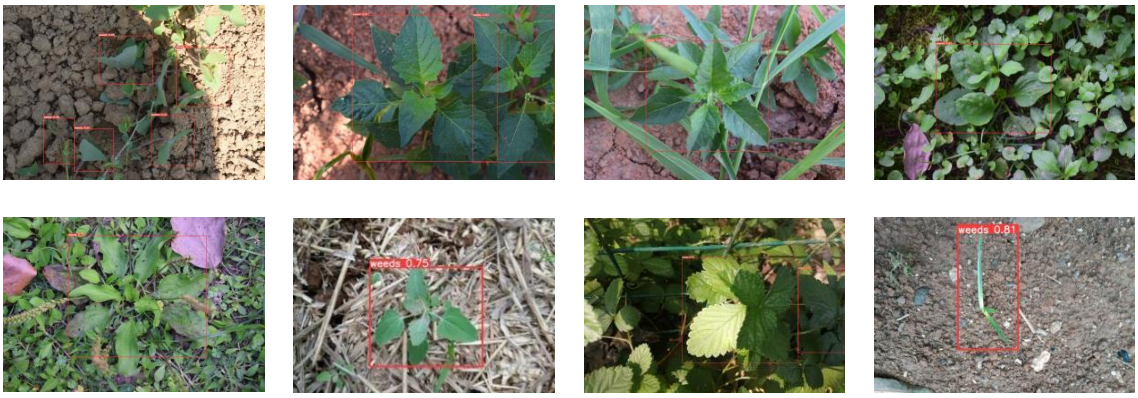


Figure 4.6 Performance evaluation of the YOLOv5-based weed recognition model on the test set

As shown in Figure 4.6, the YOLOv5s model demonstrates outstanding performance in dryland weed recognition tasks. The model not only exhibits excellent convergence and stability during training but also demonstrates high reliability and accuracy in practical applications.

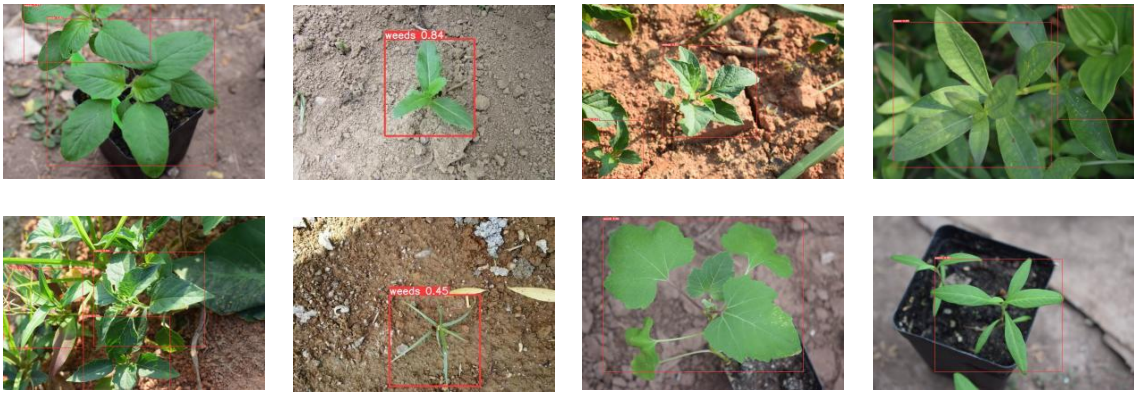


Figure 4.7 Performance evaluation of the YOLOv9-based weed recognition model on the test set

As shown in Figure 4.7, the YOLOv9-c model demonstrates outstanding performance in dryland weed recognition tasks, exhibiting high reliability and accuracy in practical applications.

Finally, we compare the average inference time of the two models as shown in Figure 4.2.

Table 4.2 Average inference time and real-time detection frame rate of the model

model	YOLOv5s	YOLOv9-c
Average inference time (ms per image)	7.9	28.9
Real-time FPS monitoring	126.6	34.6

The results indicate that YOLOv9-c exhibits slow inference speed, yet its current frame rate remains adequate for real-time object detection. However, when using lower-performance GPUs or devices, the real-time detection frame rate of YOLOv9-c may plummet below 30, rendering it unsuitable for real-time object detection.

4.2.2 Dynamic Recognition Performance

As shown above, when using low-performance GPUs and devices, YOLOv9-c's real-time detection frame rate may drop below 30, making it unsuitable for real-time object detection. Therefore, YOLOv9 is currently not applicable for real-time weed recognition in weeding robots.

To evaluate the real-time detection performance of the YOLOv5s weed recognition model, we applied it to ten real-world video clips filmed in diverse environments including roadside green belts, courtyards, and farmlands. These clips showcase the model's recognition accuracy for ten distinct types of weeds. The videos, sourced from online platforms, represent complex and varied background conditions typical in practical applications. Each clip features dynamic visual content with constantly changing weed positions and morphologies, significantly increasing detection challenges.

The experiment measured the duration of effective object detection by the YOLOv5s weed recognition model across 10 video clips. Our analysis revealed that the model achieved an average detection efficiency of 89% of total duration, with detection rates reaching up to 99% in videos with minimal object occlusion.

Figure 4.8 Dynamic performance of the YOLOv5-based weed recognition model (image: screenshot; left: original video; right: predicted video)

In summary, the YOLOv5 weed recognition model demonstrated outstanding performance across these 10 video clips, showcasing its exceptional capability in handling complex backgrounds and dynamic content. This further validates the feasibility of applying YOLOv5 weed recognition to intelligent weeding robots.

4.3 Analysis and Discussion

4.3.1 Summary of Comparative Experiments

Experimental results demonstrate that YOLOv5s achieves faster training speed than YOLOv9-c on the same computing resources. Both models exhibit excellent performance in accuracy with low false positive rates. Recall rates show comparable performance with minimal false negative rates. In terms of average precision, YOLOv5s maintains stable detection performance with a higher mean value (0.983), while YOLOv9-c shows significant divergence between F1 score and mAP, with unstable performance across different confidence thresholds. Training metrics reveal YOLOv5s outperforms YOLOv9-c: its box_loss and obj_loss progressively decrease and converge, indicating strong generalization ability. Both mAP_0.5 and mAP_0.5:0.95 of YOLOv5s demonstrate steady growth and convergence, with training loss values substantially lower than YOLOv9-c's, showing faster convergence. The F1 curve shows YOLOv5s maintains high accuracy and recall across confidence thresholds from 0.1 to 0.5,

demonstrating robustness and stability, whereas YOLOv9-c requires a confidence threshold around 0.3 to achieve similar performance. On the PR curve, YOLOv5s excels at maintaining high accuracy with minimal false negatives at high recall rates, while YOLOv9-c's accuracy decreases proportionally with increased recall, indicating suboptimal performance. In static recognition tests, YOLOv5 demonstrated outstanding performance in dry-field weed detection, exhibiting excellent stability and reliability. YOLOv9-c, however, has slower inference speeds, with frame rates potentially dropping below 30 on low-performance devices, making it unsuitable for real-time detection. In dynamic recognition tests, YOLOv5 accurately and consistently identified weeds across ten different video scenarios, achieving an average frame selection rate of 89%—demonstrating its capability to handle complex backgrounds and dynamic content. YOLOv9-c, however, fails to meet real-time detection requirements on low-performance devices and is therefore not suitable for real-time detection by weeding robots.

The YOLOv5-based weed recognition model demonstrates superior performance, effectively handling real-time weed detection for intelligent weeding robots.

4.3.2 Discussion on the Performance of YOLOv5-Based Weed Recognition Models

In the YOLOv5-based weed recognition model tests, we achieved satisfactory results. However, during the analysis of experimental outcomes, several intriguing issues were identified that highlight the current model's limitations. While these problems have minimal impact on the practical performance of intelligent weeding robots, they provide valuable insights for future research and improvements.

(1) The model demonstrates excellent performance in identifying real weeds, accurately locating nine distinct weed species across various dynamic video sequences. However, a notable issue emerges: the model's confidence level in recognizing artificially generated weed images or logos even surpasses that of authentic weeds. This indicates potential overfitting of certain features, leading to high confidence in unintended targets. This phenomenon primarily occurs because the features learned during training may be excessively reinforced in artificial images, causing the model to mistakenly identify these features as representing real weeds.

To obtain accurate false positive rates for generative weed images, we generated 200 weed-related images using the DELL-E model and fed them into the YOLOv5 weed recognition model. The recognition results are shown in Figure 4.9, with the final false positive rates presented in Table 4.3.

Table 4.3 False Positive Rates of YOLOv5 Weed Recognition Model for Generative Weeds

Total number of generative weed data	False alarm count	false alarm rate
200	42	0.21

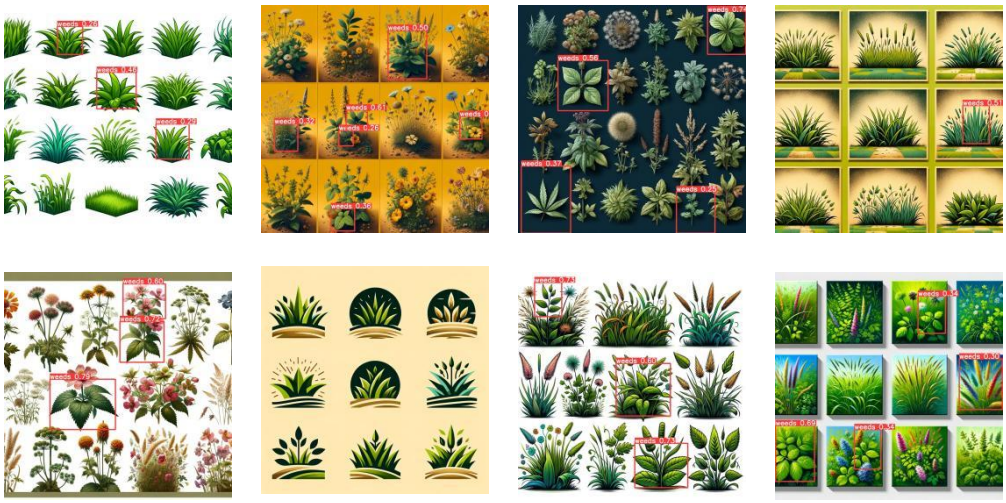


Figure 4.9 Demonstration of YOLOv5 weed recognition model's performance in identifying generative weeds

The root cause of this misidentification issue lies in the lack of diversity and generalization in the training dataset. The weed images in the dataset primarily originate from real-world environments, with insufficient artificial weed images. Consequently, the model often fails to accurately distinguish between these unexpected images and authentic weeds. This phenomenon is known as the "out-of-domain generalization problem" in deep learning, where the model underperforms on unfamiliar data domains beyond its training set.

To address this issue, we should incorporate more artificial weed images into the training dataset. By employing data augmentation techniques and diversifying data sources, we can enhance the model's generalization ability. This will enable the model to better distinguish feature differences between real weeds and unintended targets, thereby improving its reliability and robustness in practical applications.

(2) Another identified issue is that while the model performs well in recognizing near-ground shots and weeds in their early growth stages (2-9 leaf stages), it shows significant limitations when identifying aerial drone shots and mid-to-late life cycle weeds. This stems from the specialized nature of the training dataset. The model was

primarily trained using images of near-ground shots and early-stage weeds, which explains its strong performance in these specific scenarios but limited effectiveness in other contexts.

This issue highlights the limitations of the current dataset, particularly its inadequate coverage across varying shooting heights and weed growth stages. High-altitude drone imagery, with its lower resolution and unique viewing angles, presents new challenges for model development. Moreover, mid-to-late-stage weeds, which exhibit significant morphological and characteristic changes, require distinct recognition strategies.

To address this challenge, we need to expand the training dataset by incorporating images from various shooting heights and different weed growth stages, including more mid-to-late-stage weeds. This comprehensive coverage of the entire weed life cycle will enhance the model's recognition capabilities. Additionally, domain adaptation techniques can be employed to improve the model's performance in unknown data domains through knowledge transfer and alignment across different data domains. By adopting this approach, we can strengthen the model's adaptability and robustness without significantly increasing the training data volume.

4.3.3 Discussion on the Performance of YOLOv9-Based Weed Recognition Models

The generalization performance of the YOLOv9 weed recognition model cannot be evaluated due to missing validation loss data. Since YOLOv5's generalization performance on artificially generated weed images is insufficient, we also tested the model's generalization capability on such images and compared its performance. The results are shown in Table 4.4.

Table 4.4 False Positive Rates of YOLOv9 Weed Recognition Model for Generative Weeds

Total number of generative weed data	False alarm count	false alarm rate
200	48	0.24

The results show that the generalization performance of YOLOv9 model is not good.

In the performance evaluation of the YOLOv9-based weed recognition model, we observed suboptimal results: The YOLOv9-c model underperformed the YOLOv5s

model across all metrics in weed recognition tasks. This outcome was expected, as the novelty of YOLOv9's algorithm and model necessitates practical validation. Through comparative experiments, we provided valuable practical insights for YOLOv9's engineers and researchers, which constitutes the core significance of this study.

Chapter 5 Deployment of Weeds Identification System

Based on the aforementioned research findings, we selected a YOLOv5-based weed recognition model as the real-time detection model for the robot. Next, we will discuss how to integrate this model with the weeding robot.

5.1 Deployment Strategy

5.1.1 Deployment Method Selection

In the weed recognition project for dryland weeding robots, there are several common deployment methods for the model.

(1) On-site deployment: The model is directly deployed on the robot's control program. This requires the robot to be equipped with a high-performance computing platform. However, large graphics cards consume significant power and typically require robust cooling systems. In field environments, particularly in arid fields where high temperatures and power shortages are common, providing stable power supply and effective cooling poses a challenge.

(2) Cloud deployment: Deploying the model on cloud servers enables the robot to utilize the powerful computing resources of cloud platforms. However, in wide area networks (WAN), data communication between the robot and the cloud platform may experience latency or even packet loss, which is critical for weeding robots with stringent real-time requirements.

(3) Edge Computing Deployment: Edge computing serves as a balanced solution between on-premises and cloud-based deployments. Edge computing devices refer to computing equipment (servers) deployed at network edges near data sources. This approach addresses both power consumption issues in on-premises setups and network communication latency in cloud deployments. Therefore, deploying models on edge computing devices is the optimal choice. However, directly integrating edge computing devices into robots would replicate the same challenges as on-premises setups. Our plan involves constructing edge computing server enclosures at field edges and establishing

full local area network coverage within farmland areas. This method effectively resolves issues related to power supply, heat dissipation, and data latency for robots.

In the development environment, we employed a DELL laptop (i5 CPU) to simulate the robot device and a desktop workstation (i7 CPU + RTX 4090 GPU) for edge computing. Both devices were connected via LAN, with the server address and port set to 192.168.1.1:8001 (configurable).

5.1.2 Programming Language Selection and Network Request Strategy

The client side utilizes Python, the same programming language as the robot control algorithm, and implements the network request function within the camera interface function to prevent latency caused by cross-language file calls. The program employs loop functions to fragment video stream data into frames, which are then transmitted to the server via network requests.

The server utilizes Python, the same programming language as the model inference algorithm, to implement the server-side code within the inference framework, thereby eliminating latency caused by cross-language file calls. The program feeds the received frame data into the model for inference and then transmits the prediction results back to the robot. The recognition output is a list of data points [[a, b, c, d],...] where [a, b, c, d] represent one of the identified objects. a and b denote the coordinates of the object's upper-left corner bounding box, while c and d indicate the coordinates of its lower-right corner.

5.1.3 Selection of Communication Protocols and Server Framework

Edge computing devices can communicate with robots through both wired and wireless connections. Wired connections may utilize USB or Ethernet, while wireless options include WIFI or Bluetooth. For herbicide-removal robots operating freely across expansive fields, WIFI-based local area network (LAN) connections represent the optimal wireless solution.

HTTP is a classic application-layer protocol for distributed, collaborative, and hypermedia information systems. POST, one of HTTP's request methods, is used to send large and complex data to servers. Flask is a lightweight Python web framework with a minimal core that can be extended to meet complex needs. In the absence of complex business logic, Flask applications typically exhibit fast response times, making them suitable for basic HTTP requests. WebSocket is a full-duplex communication protocol that enables real-time bidirectional data transfer between servers and clients. SocketIO is

a server-side framework based on WebSocket. Therefore, to compare network latency, we developed two programs: one using HTTP protocols and the Flask framework for communication, and another employing WebSocket protocols and the SocketIO framework for communication.

(1) HTTP protocol and Flask framework: The robot program uses the read function to retrieve binary image data, sends a POST request via the requests library, and receives the recognition results through response.json.

```
    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
```

The server-side program uses the Flask library to handle requests and returns JSON responses with recognized results.

```
    @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 protocol and SocketIO framework: The robot-side program reads binary image data using read, transmits the data via emit, and monitors the recognition results through inference_result.

```
    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
```

The server-side program uses the frame function to monitor image data and the emit function to send the recognition results.

```
    @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) We implemented a timer mechanism to integrate into the main program's functions for calculating total latency, network latency, preprocessing latency, and inference latency. Notably, preprocessing and inference latencies are pipeline fixed rates that may cause frame drops, whereas network latency results in delayed data transmission without frame loss.

```
    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")
```

Our experiments show that the average network communication latency is 10.1ms when using HTTP protocol with Flask framework, while it reaches 557.6ms with WebSocket protocol and SocketIO framework. Therefore, we choose HTTP protocol with Flask framework as the communication protocol between the robot and the server.

5.2 Algorithm Design for Robot Side

In development environments, we utilize a video clip to simulate the camera's live recording. For production deployment, only one line of code modification is required, with the changes specified in the appendix program. The robot-side program primarily involves the following steps: frame-by-frame video reading, frame format conversion, image data transmission via POST requests, and recognition result reception. The code snippets provided are pseudocode for reference purposes only, with the complete source code available in Appendix B.

(1) We utilize the cv2.VideoCapture function to open the video file at the specified path, define a frame counter (frame_count), and implement the main loop of the recognition system.

(2) We use the cap.read function to read frame images from the video stream. If the read is successful, ret is True; otherwise, ret is False, and the loop exits.

(3) To comply with HTTP protocol requirements, we employ the cv2.imencode function to encode the read frame images into JPEG format, followed by the img_encoded.tobytes function to convert the encoded images into byte data.

(4) To comply with the requests library's API specifications, we generate a dictionary file containing image byte data and upload it to the server via a POST request. If the response status code is 200 (indicating successful recognition), the response.json method decodes the JSON string into a list of recognition results. The program then proceeds to the next iteration.

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

5.3 Server-side Algorithm Design

The server-side program primarily involves the following steps: establishing a Flask server, receiving image data, preprocessing the data through recognition functions, performing inference operations, and ultimately delivering the recognition results to the robot. The code snippets provided in this article are pseudocode for reference only, with the complete source code available in Appendix C.

(1) Server Function Module: We first create a Flask instance, then define a route listener 'upload' using@app.route. The read function retrieves image data from POST requests, followed by image recognition through process_image. The response is converted to JSON format via jsonify and returned to the client. The server's request contains device camera ID, current frame ID, and file image, while the response returns device camera ID, current frame ID, and a list of recognition results.

(2) Recognition Function Module: As this aligns with the dynamic recognition algorithm described earlier, we will not elaborate here. The process begins with image preprocessing to meet model input requirements. The Image.open library is used to read images, while io.BytesIO handles binary data processing. The PIL image is converted into a NumPy array using np.array for numerical operations. The letterbox function adjusts image dimensions and fills them to maintain aspect ratios. The NumPy array is then transformed into a PyTorch tensor. Data types are converted to half-precision or single-precision floating-point numbers based on the half variable. Images are normalized to the [0.0,1.0] range. A batch dimension is added to meet model input requirements. The permute function rearranges tensor dimensions to match the model's channel order. Finally, the model performs predictions, and post-processing of outputs yields final target locations. Non-maximum suppression (NMS) is applied using the non_max_suppression function to remove redundant bounding boxes. Subsequently, bounding box coordinates are adjusted to match the original image's scale, and each box's coordinates are appended to the tars list.

```
@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 System Testing

To evaluate the reliability of our deployed weed recognition system, this study employed MP4 videos with a resolution of 1920×1080 and a frame rate of 30FPS for recognition testing. The system operated stably for over one hour before being manually terminated. No system failures such as crashes, error reports, frame drops, lagging, or data congestion were observed during operation.

The average latency data obtained from over 108,000 frame images are presented in Table 5.1. Total latency refers to the total time from when each video frame is cropped at the robot end until the final recognition data is acquired. Network latency denotes the total time consumed for a round-trip transmission over the network link. Preprocessing latency represents the total time spent on image data preprocessing operations prior to inference. Inference latency refers to the total time required for the model to perform image inference and algorithmic processing to generate recognition results. The frame rate is calculated by combining preprocessing and inference latencies using Equation (5.1). As shown in the table, the latency meets the qualification standard (under 50ms), while the frame rate satisfies the operational standard for complex environments (over 30FPS).

Table 5.1 Delay data of weed identification system

Average total delay (ms)	Average network latency (ms)	Average preprocessing delay (ms)	Average inference delay (ms)	Average FPS
41.89	10.12	4.99	26.78	31.48

$$\text{frame rate} = \frac{1000}{\text{预处理延迟} + \text{推理延迟}} \tag{5.1}$$

According to the test results of Chapter 4 and the test results of the system, it can be determined that the weed recognition system has the characteristics of low delay, high frame rate, high stability, high efficiency and high quality, and has all the conditions required by the actual production environment.

Chapter 6 Summary and Outlook

6.1 Summary of the Study

Weeds management, as a critical component of agricultural production, has long been a key focus in agronomic research. With the rapid advancement of internet, robotics, and artificial intelligence technologies, weed management is increasingly moving toward automation and intelligent solutions. This study conducted comparative experiments between YOLOv5 and YOLOv9 to demonstrate YOLOv5's superior performance in dryland weed recognition tasks. Additionally, we deployed a weed recognition system to enable practical application of the YOLOv5-based model in real-world farming environments. The findings not only provide technical support for intelligent weeding robots but also offer practical references for researchers and users of YOLOv9 models. The key research outcomes are summarized as follows:

(1) The foundational knowledge of deep learning theory was studied, with a focus on elucidating the basic structures and working principles of convolutional neural networks, YOLOv5, and YOLOv9. Using an open-source weed image dataset, the dataset was constructed. Hyperparameters were optimized and model training was performed. A dynamic recognition algorithm was designed to meet the requirements for dynamic detection on weeding robots.

(2) Comparative experiments between YOLOv5 and YOLOv9 demonstrated that YOLOv5 outperforms YOLOv9 in accuracy, recall, and mean average precision (mAP). During training, YOLOv5 showed gradual reduction and convergence of box_loss and obj_loss, while mAP_0.5 and mAP_0.5:0.95 exhibited steady growth and convergence, indicating strong robustness and stability. In contrast, YOLOv9 underperformed on these metrics. YOLOv9's slower inference speed may cause frame rates to drop below 30 on low-performance devices, making it unsuitable for real-time detection. YOLOv5 excelled in both static and dynamic recognition tests. Across 10 video segments with diverse scenes, YOLOv5 achieved accurate and stable weed recognition, with bounding duration reaching 89% of total duration. YOLOv9 failed to meet real-time detection requirements on low-performance devices and is not suitable for real-time weed recognition tasks in robotic weeding systems. Therefore, YOLOv5 is proven to be the optimal weed recognition model for real-time detection in intelligent weeding robots.

(3) This study highlights the out-of-domain generalization issue in YOLOv5-based weed recognition models when applied to artificial weed images. It demonstrates that while these models exhibit superior performance in specific scenarios, their

effectiveness is constrained in other contexts, providing valuable insights for future research directions.

(4) The system of weed identification is deployed, and the optimal deployment strategy is found, which makes the system has the characteristics of low delay, high frame rate, high stability, high efficiency and high quality, so that the system has all the conditions required by the actual production environment.

6.2 Limitations and Future Directions

This paper conducts a series of studies on weed recognition algorithms based on deep learning. The research has some limitations:

(1) To control variables in the comparative experiment, this study set the training cycles for both models to 70. The training results indicate that 70 cycles are insufficient for the YOLOv9 model to achieve optimal performance. Therefore, some test data for YOLOv9 may not represent its best solution.

(2) In weed recognition tasks, the training model requires a sufficiently diverse set of weed images to enhance generalization capability, given the complex background conditions such as regional variations, climate, vegetation patterns, and camera angles. Therefore, the open-source dataset used in this study still has limitations.

(3) Further experiments are required for the deployment of the weed recognition system. Programming languages such as C++ or Java could be considered, and USB4 direct connection could be adopted for data transmission. To determine which solution would optimize system performance, additional research is warranted.

Looking ahead, AI-powered smart weeding robots hold vast potential in agricultural applications, yet face significant technical hurdles. While YOLOv5 excels at dry-field weed detection, its performance remains limited in complex farming environments. Future research should focus on enhancing the model's robustness and generalization capabilities to ensure stable operation across diverse conditions—including varying regions, climates, vegetation types, and camera angles.

With the continuous advancement of deep learning technologies, new model architectures and training methods are constantly emerging. As a novel neural network architecture, Transformers have achieved remarkable success in natural language processing, with the widely recognized GPT4 being a prime example based on this framework. However, large-scale language models require excessive computational power, which often proves insufficient for real-time detection tasks. Forcing the application of large language model technology to object detection tasks would

inevitably lead to overfitting. Consequently, computational power limitations and training-inference speed challenges remain critical issues in artificial intelligence development, necessitating innovations in high-performance computing infrastructure. Furthermore, model architecture design is paramount. Can existing deep network architectures effectively support the scale of multimodal data and world models? Here, the author makes a bold hypothesis.

First, if we define "shutting down" as energy cutoff, the human brain operates continuously 24/7 until cryopreservation (scientifically unverified) or brain death. Humans require over a decade of learning to develop cognitive abilities—so how about machines? Do they need even longer? Second, the human brain deepens impressions during output and simultaneously processes information during learning, a phenomenon called the generative effect. However, current models separate training and inference processes. Is this a limitation of deep neural network architectures? Regardless, we hope to see truly multimodal and world models in the near future.

References

[1] Hall D, Dayoub F, Kulk J, 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., 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] Peng Mingxia, Xia Junfang, Peng Hui. Efficient Recognition Method for Cotton Field Weeds in Complex Backgrounds with FPN-Integrated Faster R-CNN [J]. Journal of Agricultural Engineering, 2019,35(20):202-209.

[7] Zhai Changyuan, Fu Hao, Zheng Kang, et al. Establishment and experiment of an online recognition model for field cabbage based on deep learning [J]. Journal of Agricultural Machinery, 2022,53(04):293-303.

[8] Jin Xiaojun, Sun Yanxia, Yu Jialin, et al. A method for weed recognition in vegetable seedling stage based on deep learning and image processing [J]. Journal of Jilin University (Engineering Edition), 2023,53(08):2421-2429.

[9] Ji Wenli, Liu Zhou, Xing Haihua. Research on Lightweight Methods for Agricultural Weed Recognition Based on YOLO v5 [J]. Journal of Agricultural Machinery, 2024,55(01):212-222+293.

[10] Shen Tongping, Wang Yuanmao, Huang Fangliang, et al. Research on Online Teaching Effectiveness Evaluation Based on Deep Learning Technology [J]. Journal of Hebei North University (Natural Science Edition), 2021, (03):49-54.

[11] Li Shasha. Research on Small and Occluded Object Detection Algorithm Based on Deep Regression Neural Network [D]. Henan University, 2022.

[12] Qu Zhijian, Gao Tianzi, Chi Rui, et al. Improved YOLOv3-based detection and recognition of contact network nests [J]. Journal of East China Jiaotong University, 2021, (04):78-86.

[13] Yan Qintao. Design and FPGA Verification of Single-Subject Face Recognition Algorithm Based on Deep Transfer Learning [D]. Southeast University, 2021.

[14] Sai Bin. Structuring video data and extracting pedestrian features for large-scale crowd behavior analysis [D]. National University of Defense Technology, 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, Wang P, Tang Y, Luo F, et al.Weed25: A deep learning dataset for weed identification[J]. Front. Plant Sci., 2022, 13:1053329.

Appendix A Source Code of Real-time Detection Algorithm

```
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()
```

Appendix B: Source Code of Robot-side Algorithm for System Deployment

```
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.")
```

Appendix C: Source Code of Server-side Algorithm for System Deployment

```
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
```

```
38.   def process_image(img):
39.       start_time2 = time.time()
40.       # 将二进制图像数据转换为 PIL 图像对象
41.       image = Image.open(io.BytesIO(img))
42.       # 将 PIL 图像对象转换为 NumPy 数组
43.       img0 = np.array(image)
44.       # 记录识别对象坐标
45.       tars = []
46.       # 将图像缩放到指定尺寸
47.       img = letterbox(img0, stride=stride)[0]
48.       # 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
49.       img = np.ascontiguousarray(img)
50.       # 把数组转换成张量,且二者共享内存
51.       img = torch.from_numpy(img).to(device)
52.       img = img.half() if half else img.float()
53.       # 压缩数据维度
54.       img /= 255  # 0 - 255 to 0.0 - 1.0
55.       if len(img.shape) == 3:
56.           img = img[None]
57.       # 对 tensor 进行转置
58.       img = img.permute(0, 3, 1, 2)
59.       end_time2 = time.time()
60.       # 计算执行时间（以毫秒为单位）
61.       execution_time_ms2 = (end_time2 - start_time2) * 1000
62.       print(f"Pretreatment Time Consuming: {execution_time_ms2:.2f} ms")
63.       # Inference 模型推理
64.       pred = model(img, augment=False, visualize=False)[0]
65.       # NMS 非极大值抑制 即只输出概率最大的分类结果
66.       pred = non_max_suppression(pred, conf_thres, iou_thres)
67.       # 处理预测识别结果
68.       for i, det in enumerate(pred):  # per image
69.           if len(det):
70.               # Rescale boxes from img_size to im0 size
71.               det[:, :4] = scale_coords(img.shape[2:], det[:, :4], img0.shape).ro
    und()
72.               for *xyxy, conf, cls in reversed(det):
73.                   # 添加识别对象坐标
74.                   tars.append([int(xyxy[0]), int(xyxy[1]), int(xyxy[2]), int(xyxy
    [3])])
75.       return tars
76.   if __name__ == '__main__':
77.       app.run(host='0.0.0.0', port=5000 , debug=True)
```

Acknowledgements

Time flies, and the four years of university life have passed in the blink of an eye. Reflecting on this wonderful period of academic and personal growth, I am filled with profound gratitude. Here, I would like to express my sincerest thanks to those who have provided assistance and support throughout my journey of personal development.

First and foremost, I would like to express my deepest gratitude to my mentor, Professor Pang Muye. Throughout the thesis selection, research methodology, and writing process, Professor Pang provided me with meticulous guidance and unwavering support. His profound scholarship, rigorous academic approach, and patient mentorship have been invaluable to my academic journey.

Secondly, I would like to express my gratitude to all the instructors and faculty members of the college. It is through your dedicated teaching and selfless contributions in the classroom that I have been able to continuously enrich my knowledge, broaden my horizons, and cultivate a scientific mindset and rigorous learning attitude.

I would also like to express my gratitude to my classmates and friends. Over these four years, we have discussed issues, shared experiences, and encouraged each other. Particularly during the thesis writing process, everyone provided me with invaluable advice and assistance, enabling me to successfully complete this paper.

Finally, I would like to express my special gratitude to my family. Your consistent understanding, support, and encouragement have been the greatest motivation on my journey forward. No matter what difficulties I encounter, you have always been my steadfast support, enabling me to face challenges with courage.

I would like to express my deepest gratitude to all those who have cared for, supported and helped me.