.Problem Chosen :	A

2023 APMCM summary sheet

An Apple Recognition Model Based on Deep Learning and Image Processing

Abstract

"This paper addresses the issue of extracting and analyzing fruit image labels, identifying and analyzing apple image data, and establishing an apple recognition model to solve and visually present problems related to quantity calculation, precise localization, maturity level assessment, quality inspection, and individual identification.

Firstly, an apple attribute analysis model based on YOLOv5 was established, employing preprocessing using the imgaug library on the dataset in Appendix #1, achieving dataset augmentation and classification division. LabelImg was used for label preparation, followed by model training, balancing speed and accuracy attributes, selecting the optimal preprocessing model, adjusting parameters for optimal model performance, and enabling transfer learning.

Addressing different requirements for problems 1 & 2, the model computed and statistically visualized apple quantity and positions within the image set.

For problem 3 regarding maturity assessment, image segmentation based on the LAB color space was performed on the dataset. Validation using RGB values produced RGB mean distribution graphs and histograms for the distribution of R values. Due to the positive correlation between apple coloring and maturity, the histogram of R values was chosen to represent the maturity distribution histogram, resulting in visualizations of maturity levels.

For problem 4 regarding quality calculation, pixel area calculations were conducted based on image processing from problem three. By estimating the imaging area through pixel area estimation and abstracting apples into spherical shapes, apple volume was calculated using the formula for spherical volume.

Subsequently, a precise apple recognition model based on ResNet was established using dataset #2 comprising collected fruit images. Balancing sample data, preprocessing images using OpenCV, dividing the dataset into training, testing, and validation parts, setting learning rates, and training the model using cuda for forward propagation resulted in the optimal model.

For problem 5 concerning apple ID recognition, the chosen optimal model was utilized to classify fruits in Appendix #3, distinguishing apples from other fruits. Utilizing the Matplotlib library, computational results were visualized, generating histograms depicting apple ID distributions.

Finally, a discussion was conducted on the strengths, weaknesses, and practical feasibility of the models.

Keywords: Machine Learning, Deep Learning, YOLO, ResNet, OpenCV."

Contents

1 Introduction 1
1.1 Problem Background 1
1.2 Restatement of the Problem 1
1.3 Our Work 2
2. Assumptions and Justifications 3
3. Notations 3
4 Model I: Apple Property Analysis Model Based on YOLOv5 3
4.1 YOLOv5 working principle 3
4.2 Data Preparation 4
4.2.1 Data Set Preprocessing 4
4.3 Model Training 6
4.4 Solution for Question 1 9
4.5 Solution for Question 2 10
4.6 Solution for Question 3 11
4.7 Solution for Question 4 13
5 Model II: Accurate Apple Recognition Model Based on ResNet 15
5.1 ResNet working principle 15
5.2 Data Preparation 16
5.2.1 Data processing 16
5.2.2 Implementation model 16
5.3 Model Training 17
5.3.1 Training 17
5.3.2 Detection 17
5.4 Solution for Question 5 18
6. Model Evaluation and Further Discussion 18
6.1 Advantage 18
6.2 Disadvantage 19
References 1
Appendices 2

1 Introduction

1.1 Problem Background

With the advancement of the “Belt and Road Initiative”, China has increasingly emerged as a dominant global player in both the production and exportation of apples.

The continuous advancements in technology have injected renewed vigor into modern agriculture. The integration of intelligent harvesting robots into the traditional realm of manual apple picking has alleviated the predicament of existing labor shortages and, to a certain extent, addressed the crucial issue of enhancing production efficiency. Against the current backdrop, technological innovations in orchard management and fruit harvesting have assumed growing significance.

However, due to the intricate and non-structured disparities within orchard environments, widespread adoption and application of intelligent auxiliary equipment encounter significant hurdles. Enhancing existing technologies by extracting features and conducting mathematical modeling on apple image data to achieve precise apple identification and classification has emerged as a crucial industry demand.

1.2 Restatement of the Problem

Question 1

Perform data preprocessing and feature extraction on the provided dataset of ripe apple images (Attachment #1). Establish mathematical models to achieve individual identification and quantity statistics, culminating in the construction of histograms depicting apple distributions.

Question 2

Conduct recognition and assessment on the provided dataset of ripe apple images (Attachment #1). Establish a geometric coordinate system for the images, achieving precise localization of individual apples within the images. Represent this localization in the form of a two-dimensional scatter plot.

Question 3

Utilizing data preprocessing and feature extraction techniques on the provided dataset of ripe apple images (Attachment #1), establish mathematical models for intelligent assessment of apple ripeness. Visualize these assessments through corresponding distribution histograms.

Question 4

Based on data preprocessing and feature extraction from the provided dataset of ripe apple images (Attachment #1), establish mathematical models for intelligent apple ripeness detection. Illustrate these detections through respective distribution histograms.

Question 5

Based on preprocessing and feature extraction from the fruit harvesting image dataset (Attachment #2), train an apple identification model capable of distinguishing apples from other visually similar fruits. Achieve precise identification of apples using the images in

Attachment #3 as experimental subjects for species recognition tests, and generate histograms depicting the distribution of apple image IDs.

1.3 Our Work

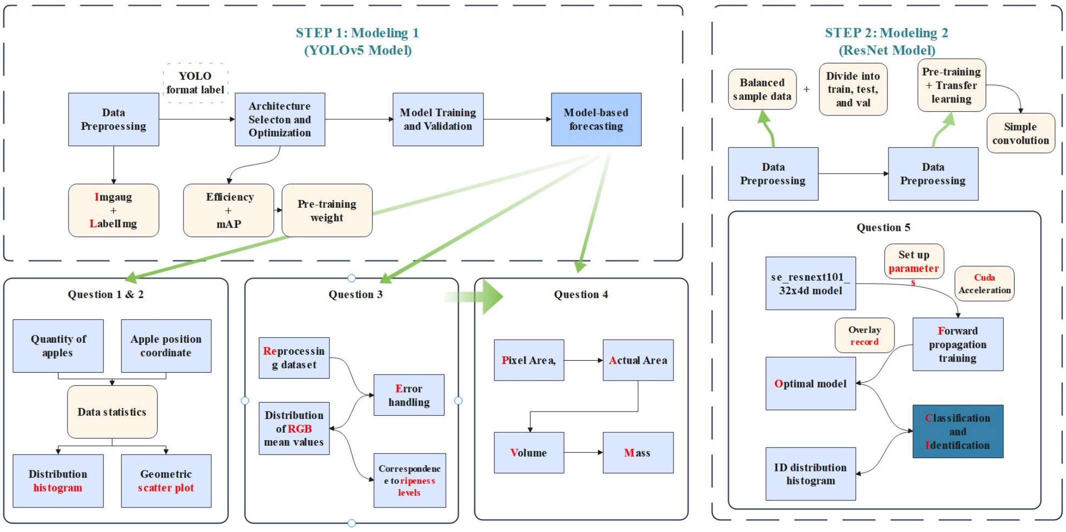

Figure 1： Working concepts

（1）

Through research on computer vision-based object detection techniques, the evaluation and selection of current object recognition algorithms are based on the balance between speed and accuracy, with real-time target identification being the evaluation basis for the model. This process concludes with the establishment of a predictive model.

Question 1&2

Based on the provided dataset of ripe apples, perform data preprocessing and manual labeling. Evaluate the recognition model comprehensively, select the most suitable pre-trained weights for transfer learning, and then proceed with model training and validation.

For different requirements of questions 1 and 2, the model is utilized to infer both the quantity of apples and their positional coordinates within the target image dataset. Subsequently, based on the obtained results, the necessary distribution histograms and geometric scatter plots are respectively constructed.

Question 3

Based on models from Questions 1 and 2, following the requirement for individual extraction and independent processing, the images are reprocessed to reconstruct RGB images, resulting in an RGB mean value distribution graph. Establishing criteria for apple ripeness, it is associated with the images to obtain a histogram depicting the distribution of ripeness levels across all apples.

Question 4

Based on the image processing results from Problem Three, the calculation of apple pixel area is conducted to estimate the actual area. Abstracting the apple's morphology, the conversion from area to volume and subsequently to mass is performed. Finally, the results are visualized through a histogram.

(2)

Question 5

Based on the requirements in question 5, reconfiguring parameters for the se_resnext101_32x4d model and comparing training performances to obtain the optimal model for classifying fruits in Attachment #3, distinguishing apples from other fruits. Based on the identified results, plot the required ID distribution histogram.

2. Assumptions and Justifications

- ✧ The pixel area of the reference image dataset equals the imaging area.
- ✧ Apples are uniformly referenced with an average density of 0.8.

3. Notations

Table 1 symbol description

Symbol	Description
mAP	Mean Average Precision
RGB	An industrial color standard based on three primary colors
Pixel area	The measurement of the total number of pixels within an area.
Imaging area	The total area covered or captured by an imaging process or system.

4 Model I: Apple Property Analysis Model Based on YOLOv5

4.1 YOLOv5 working principle

The YOLOv5 network is the culmination of the YOLO architecture series, characterized by high detection accuracy and fast inference speed. This network model employs an Anchor-based detection method and applies a single-stage detection approach. It efficiently achieves precise localization and classification of detected targets using a one-stage neural network, ensuring both high efficiency and accuracy.

YOLOv5 inherits and advances three key components of the YOLO series:

Backbone: Serving as the foundational part of the model, it utilizes a convolutional neural network to aggregate and extract image features across various image granularities.

Neck: It combines multiple target feature maps, consolidates enhanced feature representations, and delivers effective features to the prediction layers.

Head: Serving as the model's output segment, it generates final image regression predictions by predicting bounding box coordinates and class probabilities.

The three layers coordinate with each other, functioning as input->backbone->neck->head->output in the operation structure of the object detection model, aiming to achieve efficient real-time object detection. The working principle is illustrated in the diagram of Figure 2：

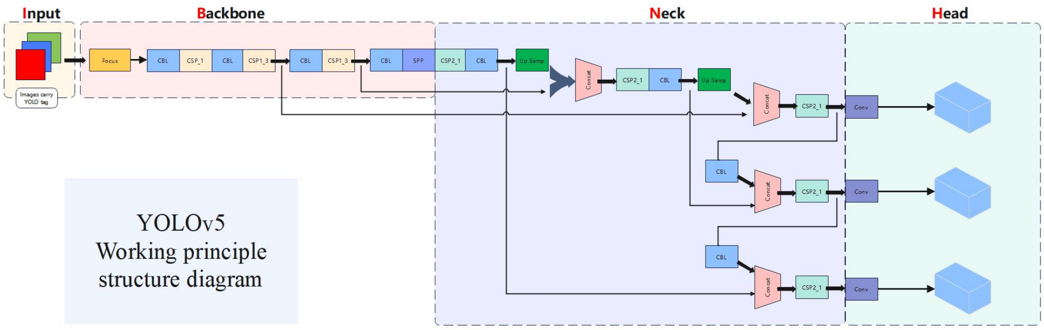

Figure 2:YOLOv5 Structure

4.2 Data Preparation

4.2.1 Data Set Preprocessing

To enhance the effectiveness of training and the reliability of sample data, before extracting image features and training the model, we utilized the provided collection of mature apple images from Attachment #1 as the dataset. Leveraging the imgaug data augmentation library, functions like Vertical Fliplr, Horizontal Fliplr Multiply, Cutout, among others, were employed to augment the dataset. This simulation allowed for the creation of apple picking recognition images under different natural lighting conditions and levels of branch obstruction. Subsequently, the dataset was divided into training and validation sets in a ratio of 9:1.

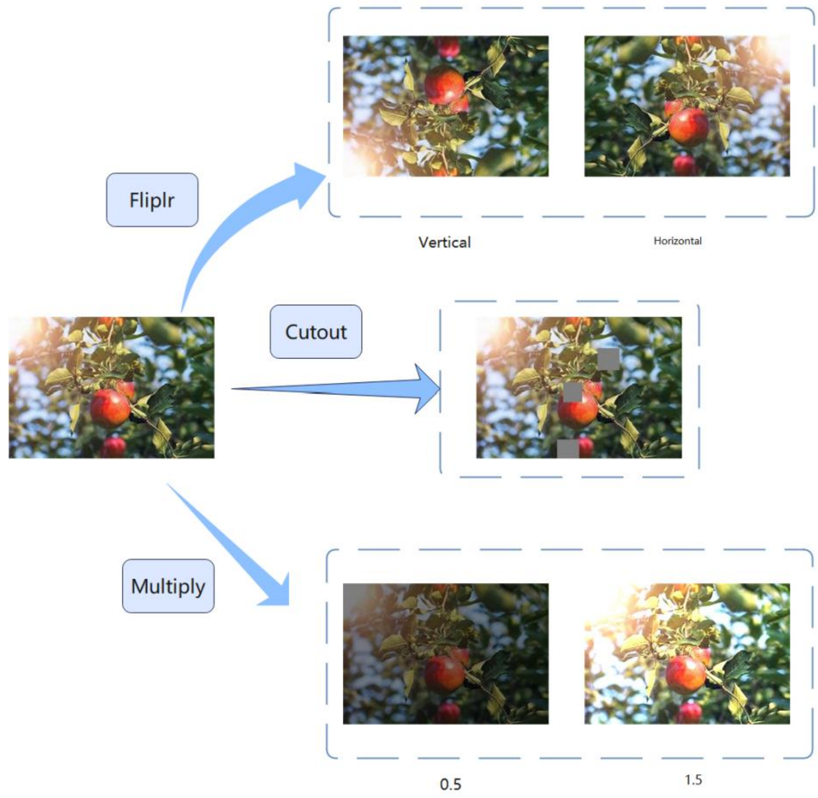

Figure 3:Preprocessing of data training set

4.2.2 Utilizing labeling for Data Annotation

Utilizing the preprocessed test set, LabelImg, an image annotation software, was chosen to label the visual apple objects in each sample image with bounding boxes. Consequently, yolo format labels for 200 images were generated and saved in .txt format.

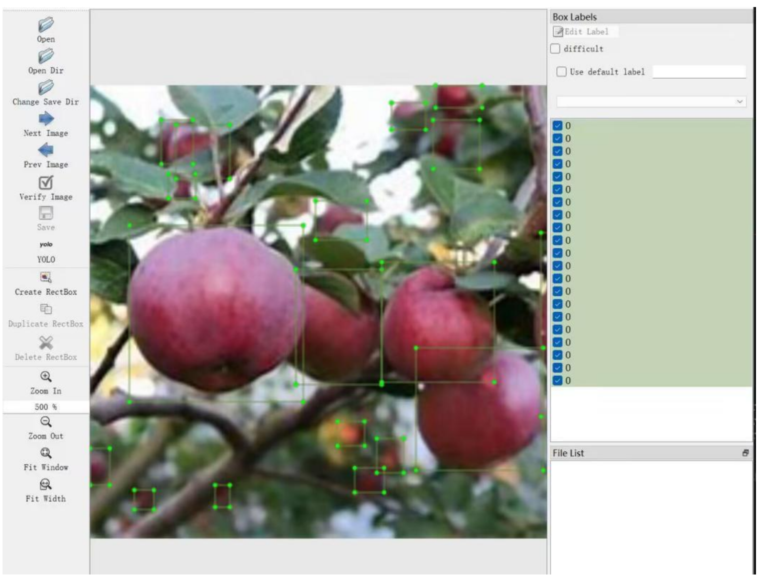

Figure 4 :Utilizing labelimg for Data Annotation

4.3 Model Training

- **Model Inference Environment**

YOLOv5 2022-11-12 Python-3.9.0 torch-1.11.0 CUDA:0 (NVIDIA GeForce RTX 2060, 6144MiB)

- **Preparation of transfer learning**

The YOLO-v5 object detection network comprises three architectures—Yolov5s, Yolov5m, Yolov5l. They share a common basic structure but differ in the feature extraction modules' specific positions within the network and the number of convolutional kernels [1]. These architectures exhibit a trend of increasing model depth and width parameters in sequence.

To comprehensively evaluate the accuracy, efficiency, and scalability of the recognition model and select the most suitable pre-trained weights for transfer learning, we conducted experiments on different network structures using the same dataset and hyperparameters. We recorded the algorithm's detection speed (ms) and mAP (%) .

This paper introduces precision (P), recall (R), and mean average precision (mAP) to assess the performance of the detection model. The expressions for P and R are as follows (1) and (2):

$$P = \frac{TP}{TP + FP} \tag{1}$$

$$R = \frac{TP}{TP + FN} \tag{2}$$

Where true positives (TP), false positives (FP), and false negatives (FN) represent correctly classified positive samples, misclassified negative samples, and misclassified positive samples, respectively.

AP denotes the average precision across all classes, calculated as the integral of P with respect to R, which corresponds to the area under the P-R curve; mAP is the mean average precision, obtained by computing the AP for each class and averaging them. The computation formulas are shown in (3) and (4):

$$AP = \frac{1}{N} \Sigma p(r) \tag{3}$$

$$mAP = \frac{1}{N} \sum_{N}^{i=1} AP_i \tag{4}$$

Where N represents the number of classes.[2]

The following comparative experimental results were obtained:

Table 2: Comparative Experimental Results of Performance and Efficiency with Different Pre-trained Weights

Parameter / Algorithm	P	R	mAP	Val/Loss	Speed (ms)
YOLOv5s	0.87995	0.82503	0.89403	0.035512	45.6
YOLOv5m	0.87729	0.82503	0.91312	0.033473	89.8
YOLOv5l	0.89208	0.82503	0.91873	0.033128	146.2

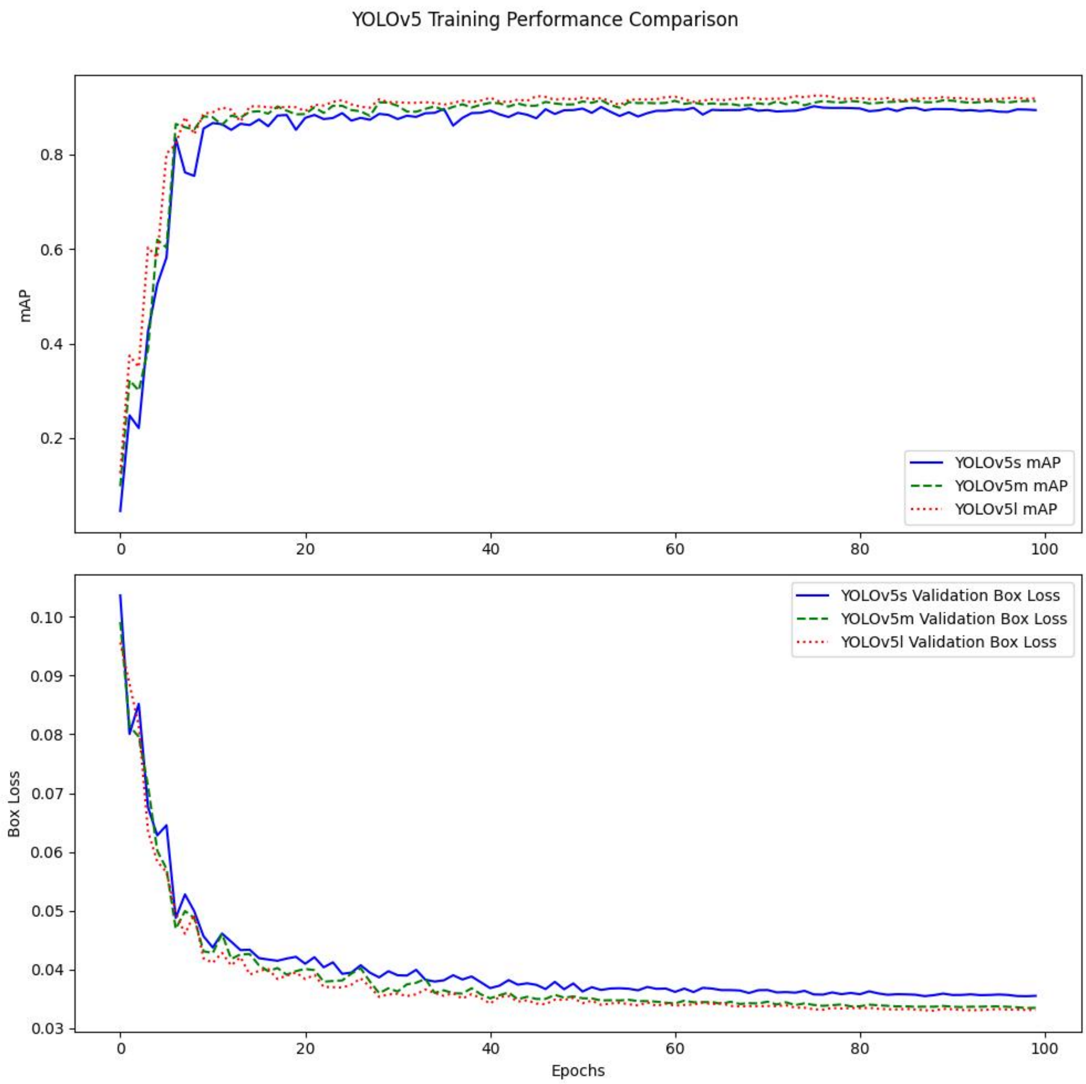

Figure 5: YOLOv5 Training Performance Comparison

The results demonstrate that under similar conditions, YOLO-v5L demonstrates high accuracy, whereas YOLO-v5s exhibits superior real-time performance.

Considering the intended application on robots, which typically have limited computational resources and memory constraints, demanding low memory usage for swift real-time responses and embedded deployment, a comprehensive assessment of detection performance, model weight size, and detection speed led us to select YOLO-v5s as the baseline network.

➢ **Training**

Based on the parameter settings, the input image area was set to 270x270, with a batch size of 32 for training over 100 epochs. The obtained training outcomes and optimal weights are illustrated in the figures below:

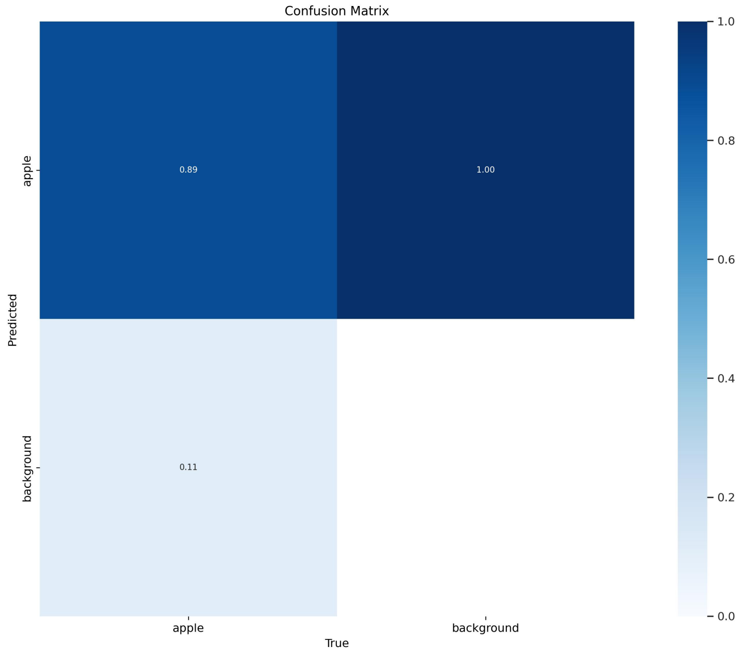

Figure 6:Confusion_Matrix

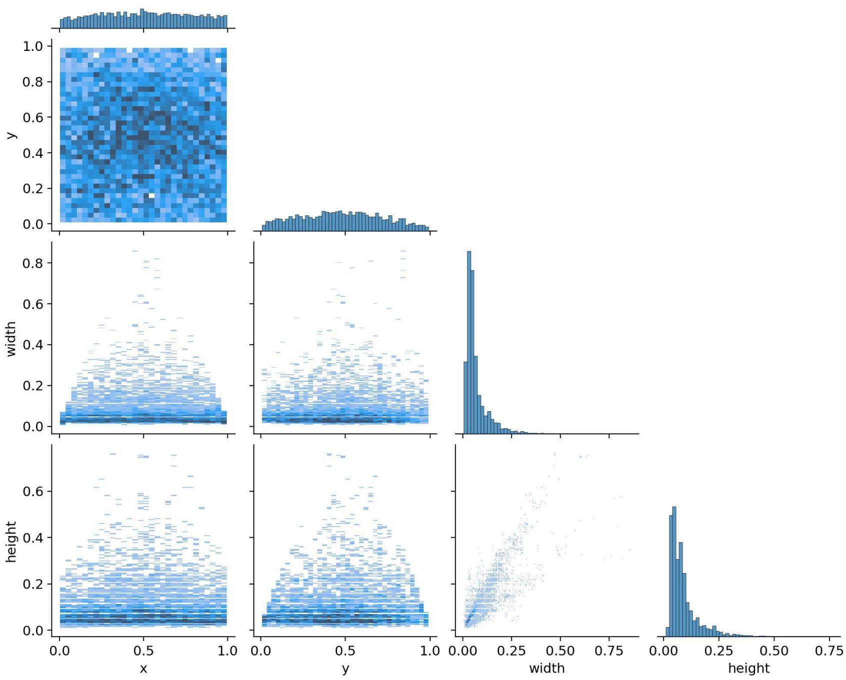

Figure 7:Label_Correlogram

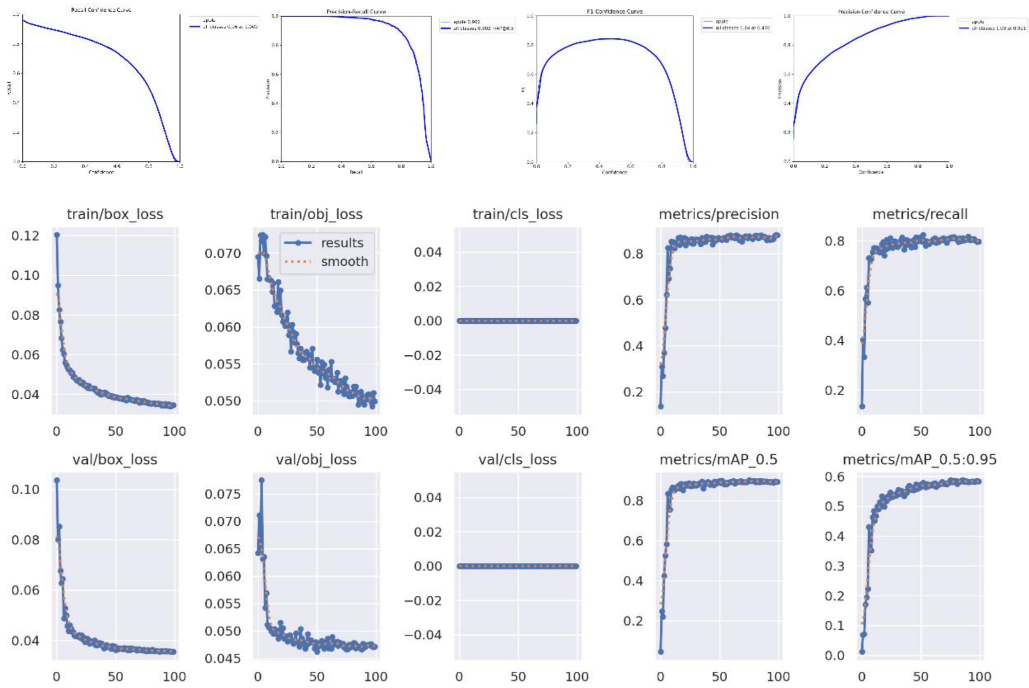

Figure 8:Training result

4.4 Solution for Question 1

Based on the generalized model obtained from transfer learning, the detected information from the target image dataset (Attachment #1) is retrieved using model code to read the rows of the annotated dataset. The successful identification and enumeration of the target apple count are accomplished. Utilizing the Matplotlib plotting library, the computational results are visualized, generating an apple count distribution histogram as illustrated in the following figures .

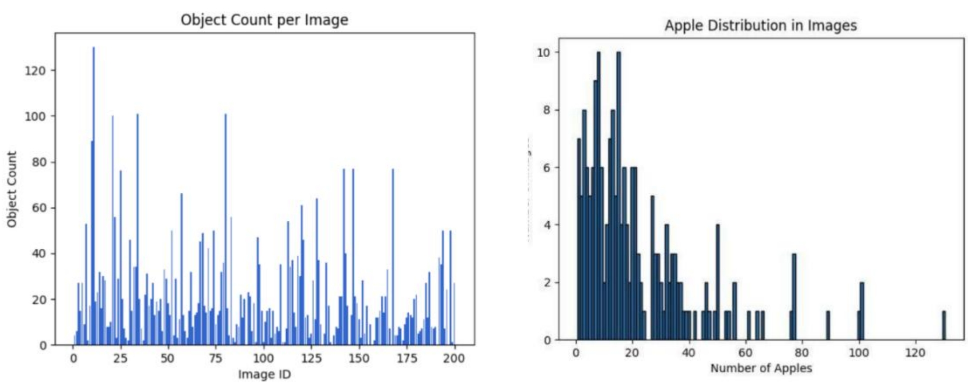

Figure 9：Apple quantity distribution

It can be inferred that the dataset in Attachment #1 exhibits a concentrated distribution of apple quantities per image within the range of 0 to 20. Moreover, as the number of apples increases, there is a fluctuating decline in the quantity of images.

4.5 Solution for Question 2

Based on the generalized model obtained through transfer learning, the objective is to detect target image data from Dataset #1. Establishing a two-dimensional coordinate system with the lower-left corner of the image as the origin, each image element within the dataset is correlated to the coordinate axes.

Utilizing the model code, the annotation dataset is read to extract all center point coordinates, pinpointing the location of apples in each image. Leveraging the Matplotlib library, individual scatter plots for each image and a comprehensive scatter plot encompassing all geometric coordinates of apples within the dataset are created, presenting a visual representation of the computational results as shown in Figure 10 & Figure11.

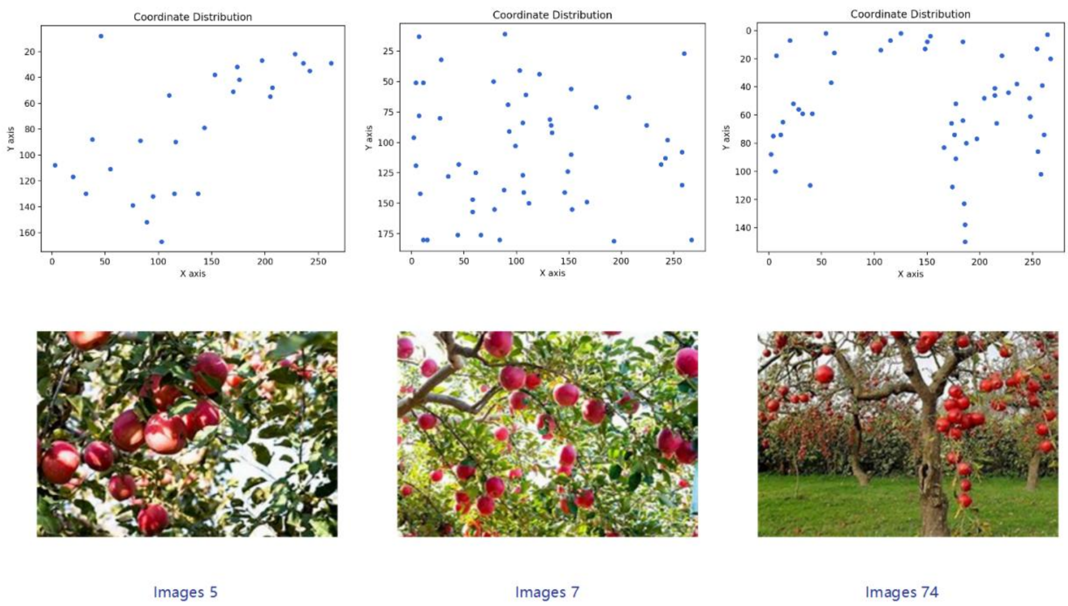

Figure 10 : Position visualization of individual images

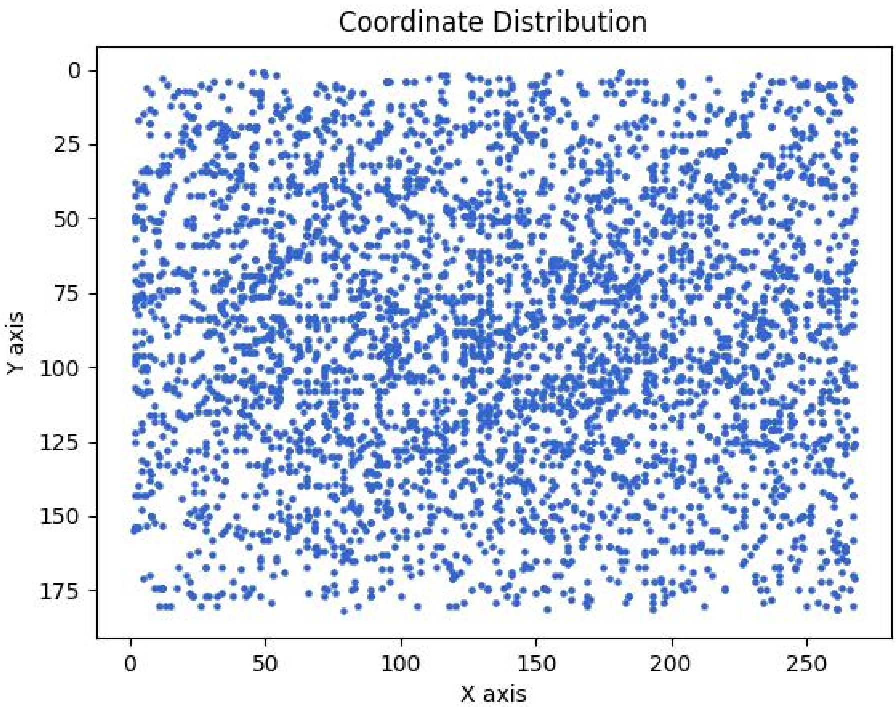

Figure 11：All apple geometry coordinates for the overall image dataset in Attachment 1

4.6 Solution for Question 3

Based on the model developed for questions 1 and 2, we've implemented an image processing methodology aimed at extracting segments containing apples within the images for individualized treatment. Adhering to the principle of independent processing, we've iteratively applied the following steps to each apple's image:

1.Extraction of Enclosed Images: Utilizing provided bounding box information, we isolated apple images corresponding to each specific bounding box from the original image.

2.Color Space Transformation: The extracted apple images were converted into the LAB color space, segmenting the images while preserving the 'a' component.

3.Morphological Operations: Applying morphological operations to the 'a' component to fill small gaps and eliminate isolated spots, ensuring the continuity and integrity of the images.

4.Reconstruction of RGB Images: Post-morphological operations, the processed 'a' component was reassembled into RGB images, using black as the background.

5.Computation of RGB Mean Values: The RGB mean values, excluding the black background, were calculated .The formula for calculating the RGB mean values is depicted as equations (5) through (8).

$$\text{MeanRed}\,(\text{R}) = \frac{\sum\limits_{i=1}^{N} R_i}{N} \tag{5}$$

$$\text{MeanGreen}\ (\text{G}) = \frac{\sum\limits_{N}^{i=1} G_i}{N} \tag{6}$$

$$\text{MeanBlue}\ (B) = \frac{\sum\limits_{N}^{i=1} B_i}{N} \tag{7}$$

Where, R_i, G_i, B_i represent the red, green, and blue channels of the i-th pixel respectively.

The total RGB mean can be created by combining the means of each channel:

$$\text{MeanRGB} = (\text{MeanRed}, \text{MeanGreen}, \text{MeanBlue}\ ) \tag{8}$$

Where A represents the area of the apple in square pixels, M denotes the degree of ripeness, which ranges between 0 and 1 (0 indicating complete unripeness and 1 indicating complete ripeness), and W signifies the weight of the apple. Equation (9) can be utilized to couple the area and degree of ripeness into the weight.

Then subject the solution to binary judgment. If all RGB mean values were zero, the image processing algorithm was reinitiated.

- If not all RGB mean values were zero, the process continued, displaying the resulting target's RGB mean values and the related RGB calculation formulas.

- If all RGB mean values were zero, an RGB segmentation algorithm was employed to retain segments where R was greater than B, and the difference between R and B fell between 90 and 110, followed by recalculating the image.[4]

The RGB mean values of apples across all images were iterated, statistically analyzed, and used to generate distribution plots for both RGB mean values and the mean value of the R channel, as depicted in Figures 12 and 13.

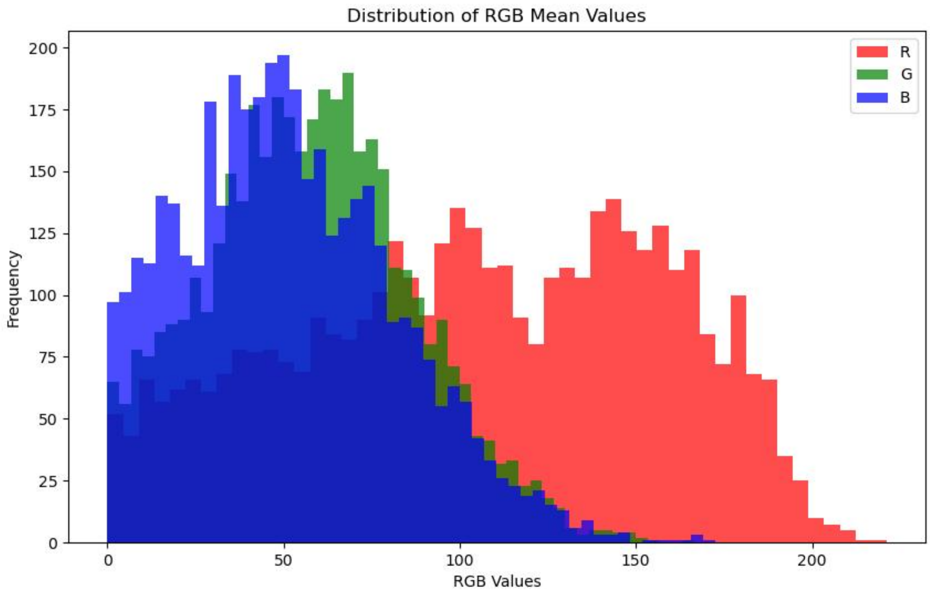

Figure 12：Distribution of RGB Mean Values

Due to the positive correlation between the coloring of apples and their maturity, we can represent the distribution histogram of maturity using R's distribution histogram:

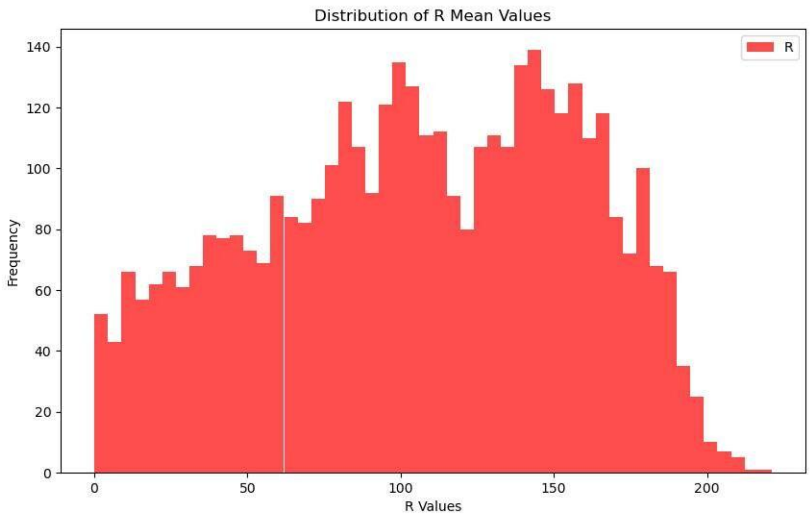

Figure 13：Distribution R Mean Values

4.7 Solution for Question 4

Based on the image processing results from the aforementioned Problem 3, calculations were conducted to determine the pixel area of apples. Through theoretical derivation and formula application (such as equations (9) to (11)), it was observed that due to the absence of physical ground truth in the given problem, an accurate representation of the actual apple area could not be obtained.

1. Normalized Image Coordinates: $\left(\frac{u}{f}, \frac{v}{f}\right)$

2. Normalized Camera Coordinates:

Utilize the inverse matrix of the camera intrinsic matrix K to convert normalized coordinates into normalized camera coordinates:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}_{\text{normalized}} = K^{-1} \begin{bmatrix} \frac{u}{f} \\ \frac{v}{f} \\ 1 \end{bmatrix} \tag{9}$$

3. Back-Projection to World Coordinates:

Utilize the camera's rotation matrix R and translation vector T to back-project normalized camera coordinates into world coordinates:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = R^{-1} \left(\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}_{\text{normalized}} - T \right) \tag{10}$$

$$\begin{bmatrix} X_{\text{real}} \\ Y_{\text{real}} \\ Z_{\text{real}} \end{bmatrix} = \frac{f}{1000} \times \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} \tag{11}$$

So assuming the pixel area of the reference image dataset is equal to the imaging area, estimated actual area is calculated using formulas (12) & (13).

$$S_r = \frac{f^2}{d^2} \cdot S_p \tag{12}$$

Where f represents the focal length, Sr stands for actual area, and Sp denotes pixel area.

$$V = \frac{4Sr}{3} \sqrt{\frac{sr}{\pi}} \tag{13}$$

Based on the above formulas, the resulting mass should be an expression involving d and f.

Abstracting the apple as a sphere, derivation from the formulas of sphere area and volume establishes the relationship between area and volume finding literature assuming an average density of 0.8 for apples.[5] Obtaining weight data and visualizing the results(Assuming f=50,b=1).

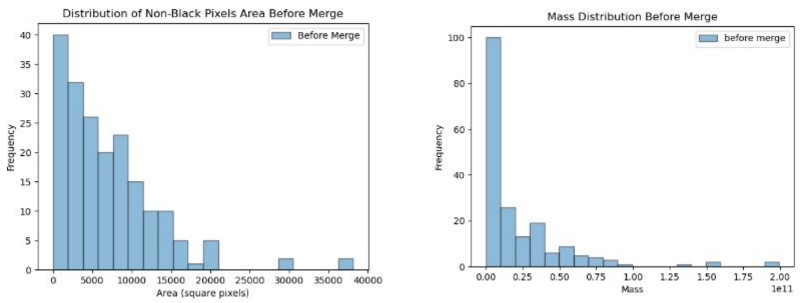

Figure 14： Area and mass distribution of a single image

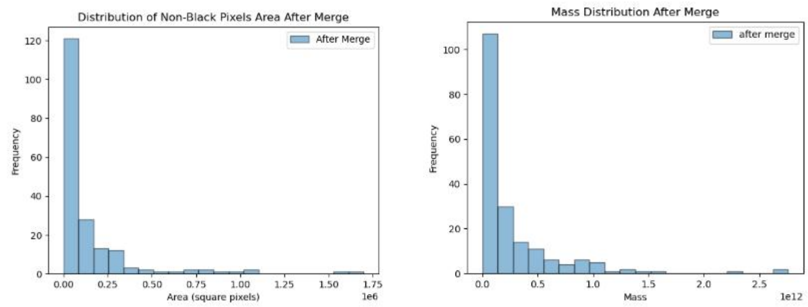

Figure 15： The area and mass distribution of fused image data set

5 Model II: Accurate Apple Recognition Model Based on ResNet

5.1 ResNet working principle

ResNet, as a type of deep neural network architecture, is centered around addressing the vanishing gradient problem in training deep networks by introducing residual connections or skip connections.

In conventional deep neural networks, increasing the network depth can lead to the problem of vanishing or exploding gradients, making the network challenging to train. ResNet introduces the concept of residual learning by employing residual blocks to construct the network. The mathematical expression for the output of traditional neural network layers is represented as (9), whereas ResNet modifies the network output as shown in (10), utilizing the residual mapping F(x) to learn the difference between the input and output.

$$H(x) \tag{9}$$

Where, H denotes the non-linear activation function, and x represents the input.

$$F(x) = H(x) + x \tag{10}$$

These residual blocks enable the network to learn residual mappings instead of directly learning low-level feature mappings. By incorporating skip connections or shortcuts, the residual blocks allow the input information to bypass one or more layers and be directly transmitted to deeper layers, facilitating the network's ability to learn identity mappings more easily.

The residual neural network consists of three main components: stacked residual units composed of convolutional layers, pooling layers, and fully connected layers. The convolutional layers are responsible for feature extraction, involving the convolution kernels sliding over the image to compute using local image information. Pooling layers perform feature reduction, compressing data while extracting useful ranges to generalize the model. The fully connected layers perform regression and classification on the sequentially extracted features, employing nonlinear combinations for output.[3]

5.2 Data Preparation

5.2.1 Data processing

Based on Dataset #2 of harvested fruit images, the images were processed using computer vision techniques, balancing the sample data. The dataset was then divided into three-dimensional segments named as train, test, and val.

5.2.2 Implementation model

Based on PyTorch, the deep neural network model engages in transfer learning for a classification dataset, improving upon the pre-trained version of ResNet,. Utilizing simple convolutions, it processes the raw features of input images, providing an initial input for deeper residual units, thereby establishing a starting point for the application of the network model.

The convolution operation involves the convolutional computation between the input image and the convolutional kernel, extracting specific feature information. The mathematical formula for the convolution operation is represented as in Equation (7):

$$Z_{ij} = \sum_{f}^{m=0} \sum_{f}^{n=0} X_{i+m,j+n} \cdot W_{m,n} + b \tag{11}$$

Where, Z represents the convolution result, X denotes the input feature map, W signifies the convolutional kernel, b stands for the bias term, f indicates the size of the convolutional kernel.

5.3 Model Training

5.3.1 Training

1. **Library Imports**: Import the necessary libraries and modules required for the task.
2. **Device Configuration**: Determine and select the computational device, either GPU or CPU, based on availability.
3. **Setting the Random Seed**: Ensure the reproducibility of experimental outcomes by establishing a specific random seed.
4. **Definition of Dataset Paths and Hyperparameters**: Specify crucial parameters such as dataset paths, model selection, input dimensions, paths for training and validation datasets, etc.
5. **Definition of Data Preprocessing Functions**: Develop image preprocessing methods for both training and validation datasets, involving techniques like cropping, flipping, normalization, and more.
6. **Definition of Custom Model Class**: Based on the chosen pre-trained model, construct a customized model. Modify the classifier component to accommodate the specific number of output categories for the intended task.
7. **Definition of Training and Validation Processes**: Encompassing data loading, forward propagation, loss computation, backward propagation of gradients, and optimizer updates among other steps.
8. **Definition of Function to Visualize Training Progress**: Create a function to visualize and present accuracy and loss metrics during both training and validation phases.
9. **Data Loading and Model Initialization**: Load datasets and initialize the model according to the specified paths and hyperparameters.
10. **Model Training and Preservation**: Iteratively execute training and validation processes. Save the model exhibiting the highest accuracy on the validation set. Additionally, record accuracy and loss metrics throughout the training phase.
11. **Outputting Training Completion Information**:Display comprehensive information regarding the completion of the training process. This includes the paths where the model and training logs are saved.

5.3.2 Detection

1. **Library Imports**: Necessary libraries and modules such as PyTorch, PIL (Pillow), tqdm, shutil, and those for assessing model performance, such as confusion matrix and classification report, are imported.
2. **Definition of Image Preprocessing Function**: Creation of a function to resize images to the same dimensions as during training through transformation.
3. **Loading Pre-trained Model**: Specifying the path of the pre-trained model and loading previously trained model parameters.
4. **Specifying the Directory of Images for Prediction and the Target Directory for Results**: Determining the folder containing images for prediction and the directory path to

save the resulting predictions.

5. **Iterating Through Images in the Folder**: Obtaining the filenames of all images within the folder designated for prediction.

6. **Conducting Predictions and Saving Results**: Iterating through each image, conducting predictions, and storing the results in the specified output folder. Concurrently, recording the true labels and predicted labels for subsequent performance assessment.

7. **Constructing a Confusion Matrix**: Utilizing the true and predicted labels to build a confusion matrix.

8. **Printing the Confusion Matrix and Classification Report**: Presenting the model's performance evaluation on the test set, encompassing the confusion matrix and classification report. The confusion matrix delineates the model's predictions across each class, while the classification report includes metrics like accuracy, recall, among others.

5.4 Solution for Question 5

Loading the se_resnext101_32x4d model integrated with the pretrainedmodels library, setting parameters such as learning rate, batch size, and epochs, conducting forward propagation training with cuda acceleration, comparing and recording the training performance, and overwriting to save the best-performing model.

Based on the selected optimal model, classify the fruits in Attachment #3, distinguishing between apples and other fruits. Utilize the Matplotlib library to visualize the computational results, obtaining an apple ID distribution histogram as shown in the Figure 16.(We divide all image indices into groups of 100, resulting in 200 groups. Then, we tally the frequency of identified apples within each group.)

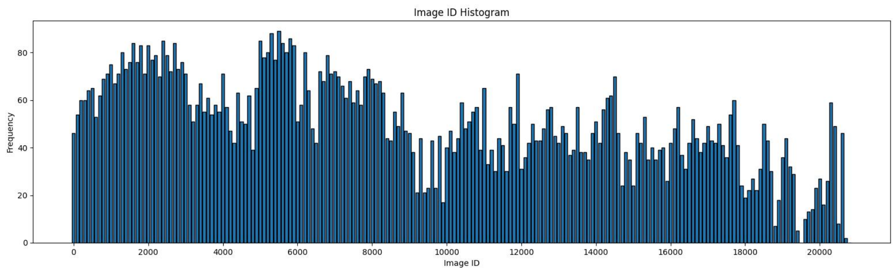

Figure 16:Apple ID distribution

7. Model Evaluation and Further Discussion

7.1 Advantage

- **Lightweightness**: The model is streamlined, allowing real-time responsiveness, making it suitable for foreground applications, and easily deployable in embedded devices.

- **Stability**: Built upon the mature YOLOv5 model, it amalgamates the significant advantages of the YOLO series. The training process exhibits stability, with negligible fluctuations, maintaining errors within an acceptable range.

- **Robustness**: The model demonstrates adaptability to various complex scenarios, showcasing operability across intricate datasets.

7.2 Disadvantage

Due to missing data in the dataset, we are unable to derive precise actual area and actual quality based on the model. However, our expectation is for the model to be feasible for real-world applications.

References

[1] Yan B, Fan P, Lei X, et al. A real-time apple targets detection method for picking robot based on improved YOLOv5[J]. Remote Sensing, 2021, 13(9): 1619.

[2] Yao J, Qi J, Zhang J, et al. A real-time detection algorithm for Kiwifruit defects based on YOLOv5[J]. Electronics, 2021, 10(14): 1711.

[3] 周涛, 刘赟璨, 陆惠玲, 等. ResNet 及其在医学图像处理领域的应用: 研究进展与挑战[J]. 电子与信息学报, 2022, 44(1): 149-167.

[4] Zhou R, Damerow L, Sun Y, et al. Using colour features of cv.'Gala'apple fruits in an orchard in image processing to predict yield[J]. Precision Agriculture, 2012, 13: 568-580.

[5] Vivek Venkatesh G, Iqbal S M, Gopal A, et al. Estimation of volume and mass of axi-symmetric fruits using image processing technique[J]. International journal of food properties, 2015, 18(3): 608-626.

Appendices

Appendice 1 Tools:

LabelImg、Pytorch、PyCharm

Appendice 2 Codes:

P1.1：

```
import os
import cv2
import imgaug.augmenters as iaa
import numpy as np

def adjust_brightness(image_path, output_folder, brightness_factor):
    # 读取图像
    image = cv2.imread(image_path)

    # 检查图像是否成功读取
    if image is None:
        print(f"无法读取图像: {image_path}")
        return

    print(f"图像成功加载: {image_path}")
    print(f"图像数据类型: {image.dtype}")
    print(f"图像形状: {image.shape}")

    # 定义增强器
    enhancer = iaa.Multiply(brightness_factor)

    # 对图像进行亮度调整
    adjusted_image = enhancer.augment_image(image)

    # 检查是否成功调整图像
    if adjusted_image is not None:
        # 获取原始文件名
        _, filename = os.path.split(image_path)

        # 构建保存路径
        output_path = os.path.join(output_folder, f"adjusted_{brightness_factor}_{filename}")

        # 保存调整后的图像
        cv2.imwrite(output_path, adjusted_image)
        print(f"图像成功保存至: {output_path}")
```

```python
        else:
            print(f"无法调整图像: {image_path}")

def flip_vertical(image_path, output_folder):
    image = cv2.imread(image_path)
    flipper = iaa.Flipud(1.0)
    flipped_image = flipper.augment_image(image)
    _, filename = os.path.split(image_path)
    output_path = os.path.join(output_folder, f"flipped_vertical_{filename}")
    cv2.imwrite(output_path, flipped_image)

def flip_horizontal(image_path, output_folder):
    image = cv2.imread(image_path)
    flipper = iaa.Fliplr(1.0)
    flipped_image = flipper.augment_image(image)
    _, filename = os.path.split(image_path)
    output_path = os.path.join(output_folder, f"flipped_horizontal_{filename}")
    cv2.imwrite(output_path, flipped_image)

def random_cutout(image_path, output_folder, cutout_size):
    image = cv2.imread(image_path)
    cutout = iaa.Cutout(nb_iterations=1, size=cutout_size)
    cutout_image = cutout.augment_image(image)
    _, filename = os.path.split(image_path)
    output_path = os.path.join(output_folder, f"cutout_{cutout_size}_{filename}")
    cv2.imwrite(output_path, cutout_image)

def random_rotation(image_path, output_folder):
    image = cv2.imread(image_path)
    rotator = iaa.Affine(rotate=(-45, 45))
    rotated_image = rotator.augment_image(image)
    _, filename = os.path.split(image_path)
    output_path = os.path.join(output_folder, f"rotated_{filename}")
    cv2.imwrite(output_path, rotated_image)

def process_brightness(input_folder, output_folder, brightness_factors):
    os.makedirs(output_folder, exist_ok=True)

    for filename in os.listdir(input_folder):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(input_folder, filename)

            for factor in brightness_factors:
                print(f"Processing brightness for {filename} with factor {factor}")
```

```python
                adjust_brightness(image_path, output_folder, factor)

def process_flips(input_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)

    for filename in os.listdir(input_folder):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(input_folder, filename)

            print(f"Processing flips for {filename}")
            flip_vertical(image_path, output_folder)
            flip_horizontal(image_path, output_folder)

def process_cutout(input_folder, output_folder, cutout_size):
    os.makedirs(output_folder, exist_ok=True)

    for filename in os.listdir(input_folder):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(input_folder, filename)

            print(f"Processing cutout for {filename} with size {cutout_size}")
            random_cutout(image_path, output_folder, cutout_size)

def process_rotation(input_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)

    for filename in os.listdir(input_folder):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(input_folder, filename)

            print(f"Processing rotation for {filename}")
            random_rotation(image_path, output_folder)

if __name__ == "__main__":
    input_folder = r"C:\Users\hp\Desktop\data\original\renamed_images"
    output_folder_brightness = "./enhanced_brightness"
    output_folder_flips = "./flipped_images"
    output_folder_cutout = "./cutout_images"
    output_folder_rotation = "./rotated_images"

    brightness_factors = [1.5, 0.5]
    cutout_size = (50, 50)

    # 如果文件夹不存在，创建它们
```

```
    if not os.path.exists(output_folder_flips):
        os.makedirs(output_folder_flips)

    if not os.path.exists(output_folder_cutout):
        os.makedirs(output_folder_cutout)

    if not os.path.exists(output_folder_rotation):
        os.makedirs(output_folder_rotation)

    # process_brightness(input_folder, output_folder_brightness, brightness_factors)
    # process_flips(input_folder, output_folder_flips)
    process_cutout(input_folder, output_folder_cutout, cutout_size)
    # process_rotation(input_folder, output_folder_rotation)
```

P1.2:

```
import os
import matplotlib.pyplot as plt

# 包含 YOLO 标签的文件夹路径
labels_folder = r'C:\Users\hp\Desktop\final_yolov5\images'

# 初始化一个字典以存储每个图像的对象计数
object_counts = {}

# 遍历每个标签文件
for i in range(1, 201):
    label_file = f"{i}.txt"
    label_path = os.path.join(labels_folder, label_file)

    # 检查文件是否存在
    if os.path.isfile(label_path):
        # 读取 YOLO 标签文件
        with open(label_path, 'r') as file:
            lines = file.readlines()

        # 计算标签文件中的对象数量
        num_objects = len(lines) if lines else 0    # 如果没有标签，将对象数量设置
为 0

        # 将计数添加到字典
        object_counts[i] = num_objects

# 绘制直方图
plt.bar(object_counts.keys(), object_counts.values(), color=(0.2, 0.4, 0.8))
plt.xlabel('Image ID')
```

```
plt.ylabel('Object Count')
plt.title('Object Count per Image')
plt.show()
```

P1.3:

```
import os
import matplotlib.pyplot as plt

def count_objects_in_labels(label_file_path):
    with open(label_file_path, 'r') as file:
        lines = file.readlines()

    # 每行代表一个目标
    return len(lines)

def analyze_folder(folder_path):
    counts = []

    for file_name in os.listdir(folder_path):
        if file_name.endswith('.txt'):
            label_file_path = os.path.join(folder_path, file_name)
            object_count = count_objects_in_labels(label_file_path)
            counts.append(object_count)

    return counts

def plot_distribution(x_values, y_values):
    plt.bar(x_values, y_values, width=1.0, edgecolor='black')
    plt.xlabel('Number of Apples')
    plt.ylabel('Number of Images')
    plt.title('Apple Distribution in Images')
    plt.show()

if __name__ == "__main__":
    folder_path = r'C:\Users\hp\Desktop\final_yolov5\images'   # 替换为你的文件夹路径

    object_counts = analyze_folder(folder_path)

    unique_counts = list(set(object_counts))
    unique_counts.sort()

    image_counts = [object_counts.count(count) for count in unique_counts]

    plot_distribution(unique_counts, image_counts)
```

P2:

```
import os
import matplotlib.pyplot as plt

# 文件夹路径，包含 YOLO 格式的标签文件
folder_path = r'C:\Users\hp\Desktop\final_yolov5\images'

# 初始化存储目标中心坐标的列表
centers = []

# 图像的宽度和高度
image_width, image_height = 270, 185

# 遍历文件夹中的每个标签文件
for file_name in os.listdir(folder_path):
    file_path = os.path.join(folder_path, file_name)

    # 检查文件是否为文本文件
    if file_name.endswith('.txt') and os.path.isfile(file_path):
        with open(file_path, 'r') as file:
            # 读取 YOLO 格式的标签
            lines = file.readlines()

            # 遍历标签中的每个目标
            for line in lines:
                data = line.strip().split()
                if len(data) == 5:   # 检查标签格式是否正确
                    class_label, x_center, y_center, width, height = map(float, data)

                    # 计算目标的中心坐标
                    center_x = int(x_center * image_width)
                    center_y = int(y_center * image_height)

                    # 将中心坐标添加到列表
                    centers.append((center_x, center_y))

# 将目标中心坐标分离为 x 和 y 坐标
x_coordinates, y_coordinates = zip(*centers)

# 绘制散点图（修改颜色为湖蓝色）
plt.scatter(x_coordinates, y_coordinates, s=5, c=(0.2, 0.4, 0.8), marker='o')
plt.xlabel('X axis')
plt.ylabel('Y axis')
```

```
plt.title('Coordinate Distribution')
plt.gca().invert_yaxis()
plt.show()
```

P3.1:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 输入和输出文件夹路径
input_folder = r"C:\Users\wangsiyu\Desktop\final_yolov5\single"
output_folder = r"C:\Users\wangsiyu\Desktop\final_yolov5\result"
# 确保输出文件夹存在
os.makedirs(output_folder, exist_ok=True)

# 获取所有图片文件
image_files = [file for file in os.listdir(input_folder) if file.endswith(('.jpg', '.jpeg', '.png'))]

mean_rgb_values = []

# 处理每张图片
for i, image_file in enumerate(image_files):
    # 构建图片文件的完整路径
    image_path = os.path.join(input_folder, image_file)

    # 读取图像
    image = cv2.imread(image_path)

    # 如果图像的 R 通道的值都为 0，则跳过该图像
    if np.all(image[:, :, 2] == 0):
        print(f"Skipping {image_file} because all R values are 0.")
        continue

    # 调整图像大小为 250x250 像素（如果需要的话）
    # image = cv2.resize(image, (250, 250))

    # 转换为 L*a*b*色彩空间
    lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2Lab)

    # 分割图像，保留 a*分量
    lab_channels = cv2.split(lab_image)
    _, threshold_image = cv2.threshold(lab_channels[1], 140, 255,
```

```
cv2.THRESH_BINARY)

            # 形态学操作，填充小孔和清除孤立斑点
            threshold_image = cv2.morphologyEx(threshold_image, cv2.MORPH_CLOSE,
cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))

            # 以黑色为背景重建 RGB 图像
            result_image = image.copy()
            result_image[np.where(threshold_image == 0)] = 0

            # 计算排除黑色背景的 RGB 值的均值
            mean_rgb = np.mean(result_image, axis=(0, 1))

            # 如果 RGB 均值都为 0，则重新处理图像
            if np.all(mean_rgb == 0):

                print(f"Reprocessing {image_file} because all RGB values are 0.")

           # 重新进行 RGB 分割，保留 R 大于 B 且 R 减 B 的值在 90-110 之间的像素
                mask = (image[:, :, 2] > image[:, :, 0]) & (image[:, :, 2] - image[:, :, 0] >= 60) &
(image[:, :, 2] - image[:, :, 0] <= 110)
                result_image = np.where(np.expand_dims(mask, axis=-1), image, 0)

                # 重新计算排除黑色背景的 RGB 值的均值
                mean_rgb = np.mean(result_image, axis=(0, 1))

                # 输出重新计算后的 RGB 均值
                print(f"Recomputed Mean RGB values for {image_file}: R={mean_rgb[2]},
G={mean_rgb[1]}, B={mean_rgb[0]}")

            if np.all(mean_rgb == 0):
                print(f"Skipping {image_file} because all RGB values are 0.")
                continue
            # 保存  RGB  均值
            mean_rgb_values.append(mean_rgb)

            # 输出 RGB 均值
            print(f"Mean RGB values for {image_file}: R={mean_rgb[2]}, G={mean_rgb[1]},
B={mean_rgb[0]}")



            # 保存处理后的图像到输出文件夹，以图片的序号和 RGB 均值命名
            original_name, extension = os.path.splitext(image_file)
```

```
        output_path = os.path.join(output_folder, f"{original_name}_R{int(mean_rgb[2])}_G{int(mean_rgb[1])}_B{int(mean_rgb[0])}{extension}")
        cv2.imwrite(output_path, result_image)


    # 转换为 NumPy 数组
    mean_rgb_values = np.array(mean_rgb_values)

    # 绘制 RGB 三个均值的分布图
    plt.figure(figsize=(10, 6))
    plt.hist(mean_rgb_values[:, 2], bins=50, color='red', alpha=0.7, label='R')
    plt.hist(mean_rgb_values[:, 1], bins=50, color='green', alpha=0.7, label='G')
    plt.hist(mean_rgb_values[:, 0], bins=50, color='blue', alpha=0.7, label='B')
    plt.xlabel('RGB Values')
    plt.ylabel('Frequency')
    plt.legend()
    plt.title('Distribution of RGB Mean Values')
    plt.savefig(os.path.join(output_folder, 'rgb_distribution.png'))
    plt.show()

    # 绘制 R 均值的分布图
    plt.figure(figsize=(10, 6))
    plt.hist(mean_rgb_values[:, 2], bins=50, color='red', alpha=0.7, label='R')
    plt.xlabel('R Values')
    plt.ylabel('Frequency')
    plt.legend()
    plt.title('Distribution of R Mean Values')
    plt.savefig(os.path.join(output_folder, 'r_distribution.png'))
    plt.show()
```

P3.2:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

def calculate_non_black_pixels(image):
    # 转换为 L*a*b*色彩空间
    lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2Lab)

    # 分割图像，保留 a*分量
    lab_channels = cv2.split(lab_image)
    _, threshold_image = cv2.threshold(lab_channels[1], 140, 255, cv2.THRESH_BINARY)
```

```
        # 在形态学操作前计算排除黑色背景的 RGB 值的均值
        mean_rgb = np.mean(image, axis=(0, 1))

        # 如果 RGB 均值都为 0，则重新处理图像
        if np.all(mean_rgb == 0):
            print(f"Reprocessing {image_file} because all RGB values are 0.")

            # 重新进行 RGB 分割，保留 R 大于 B 且 R 减 B 的值在 90-110 之间的像素
            mask = (image[:, :, 2] > image[:, :, 0]) & (image[:, :, 2] - image[:, :, 0] >= 60) & (image[:, :, 2] - image[:, :, 0] <= 110)
            image = np.where(np.expand_dims(mask, axis=-1), image, 0)

        # 形态学操作，填充小孔和清除孤立斑点
        threshold_image = cv2.morphologyEx(threshold_image, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))

        # 计算非黑色像素的数量
        non_black_pixels = np.count_nonzero(threshold_image)
        return non_black_pixels, threshold_image

    # 输入和输出文件夹路径
    input_folder = "C:/Users/wangsiyu/Desktop/Attachment 1/single"
    output_folder = "C:/Users/wangsiyu/Desktop/Attachment 1/area"

    # 确保输出文件夹存在
    os.makedirs(output_folder, exist_ok=True)

    # 获取所有图片文件
    image_files = [file for file in os.listdir(input_folder) if file.endswith(('.jpg', '.jpeg', '.png'))]

    # 存储每组的像素面积总和
    area_sum_before_merge = {}
    area_sum_after_merge = {}

    # 存储每组的文件名
    grouped_filenames = {}

    # 处理每张图片
    for i, image_file in enumerate(image_files):
        # 构建图片文件的完整路径
        image_path = os.path.join(input_folder, image_file)
```

```
# 读取图像
image = cv2.imread(image_path)

# 如果图像的 R 通道的值都为 0，则跳过该图像
if np.all(image[:, :, 2] == 0):
    print(f"Skipping {image_file} because all R values are 0.")
    continue

# 调整图像大小为 250x250 像素（如果需要的话）
# image = cv2.resize(image, (250, 250))

# 提取文件名中的数字，第二次遇到下划线就停止提取
x_value = ""
for char in image_file.split('_')[1]:
    if char.isdigit():
        x_value += char
    else:
        break

# 计算非黑色像素的数量
non_black_pixels, threshold_image = calculate_non_black_pixels(image)

# 输出非黑色像素的面积
print(f"Image: {image_file}, Non-Black Pixels Area: {non_black_pixels} square pixels")

# 更新合并前的像素面积总和
area_sum_before_merge.setdefault(x_value, 0)
area_sum_before_merge[x_value] += non_black_pixels

# 更新组内的文件名列表
grouped_filenames.setdefault(x_value, [])
grouped_filenames[x_value].append(image_file)

# 以黑色为背景重建 RGB 图像
result_image = image.copy()
result_image[np.where(threshold_image == 0)] = 0

# 在文件名中添加像素面积大小
output_filename = f"result_{x_value}_{non_black_pixels}sq_{image_file}"
output_path = os.path.join(output_folder, output_filename)
cv2.imwrite(output_path, result_image)
```

```
        # 保存数字和总像素面积到文本文件
        text_output_path = os.path.join(output_folder, f"summary_{x_value}.txt")
        with open(text_output_path, 'w') as text_file:
            text_file.write(f"Number: {x_value}\n")
            text_file.write(f"Total Pixels Area: {area_sum_before_merge[x_value]} square pixels\n")

        print(f"Processed image saved at: {output_path}")
        print(f"Summary saved at: {text_output_path}")

    # 统计合并后的像素面积总和
    for x_value, filenames in grouped_filenames.items():
        area_sum_after_merge[x_value] = sum(area_sum_before_merge[filename.split('_')[1]] for filename in filenames)

    # 绘制合并前的像素面积分布图
    plt.hist(list(area_sum_before_merge.values()), bins=20, edgecolor='black', alpha=0.5, label='Before Merge')

    # 输出合并前的像素面积分布图
    plt.title('Distribution of Non-Black Pixels Area Before Merge')
    plt.xlabel('Area (square pixels)')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()

    # 绘制合并后的像素面积分布图
    plt.hist(list(area_sum_after_merge.values()), bins=20, edgecolor='black', alpha=0.5, label='After Merge')

    # 输出合并后的像素面积分布图
    plt.title('Distribution of Non-Black Pixels Area After Merge')
    plt.xlabel('Area (square pixels)')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()
```

P4.1:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math  # 导入 math 模块以使用 sqrt 函数
```

```
# 函数：计算非黑色像素
def calculate_non_black_pixels(image):
    # 转换为 L*a*b* 色彩空间
    lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2Lab)

    # 分割图像，保留 a* 分量
    lab_channels = cv2.split(lab_image)
    _, threshold_image = cv2.threshold(lab_channels[1], 140, 255, cv2.THRESH_BINARY)

    # 在形态学操作前计算排除黑色背景的 RGB 值的均值
    mean_rgb = np.mean(image, axis=(0, 1))

    # 如果 RGB 均值都为 0，则重新处理图像
    if np.all(mean_rgb == 0):
        print(f"重新处理 {image_file}，因为所有 RGB 值都为 0。")

        # 重新进行 RGB 分割，保留 R 大于 B 且 R 减 B 的值在 90-110 之间的像素
        mask = (image[:, :, 2] > image[:, :, 0]) & (image[:, :, 2] - image[:, :, 0] >= 60) & (image[:, :, 2] - image[:, :, 0] <= 110)
        image = np.where(np.expand_dims(mask, axis=-1), image, 0)

    # 形态学操作：闭运算，填充小孔和清除孤立斑点
    threshold_image = cv2.morphologyEx(threshold_image, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))

    # 计算非黑色像素的数量
    non_black_pixels = np.count_nonzero(threshold_image)
    return non_black_pixels, threshold_image

# 输入和输出文件夹路径
input_folder = r"C:\Users\wangsiyu\Desktop\final_yolov5\single"
output_folder = r"C:\Users\wangsiyu\Desktop\final_yolov5\mass"

# 确保输出文件夹存在
os.makedirs(output_folder, exist_ok=True)

# 获取所有图片文件
image_files = [file for file in os.listdir(input_folder) if file.endswith(('.jpg', '.jpeg', '.png'))]

# 存储每组的质量总和
```

```
mass_sum_before_merge = {}
mass_sum_after_merge = {}

# 存储每组的文件名
grouped_filenames = {}

# 参数 f 和 d
f =50    # 替换为实际值
d =1    # 替换为实际值

# 处理每张图片
for i, image_file in enumerate(image_files):
    # 构建图片文件的完整路径
    image_path = os.path.join(input_folder, image_file)

    # 读取图像
    image = cv2.imread(image_path)

    # 如果图像的 R 通道的值都为 0，则跳过该图像
    if np.all(image[:, :, 2] == 0):
        print(f"跳过 {image_file}，因为所有 R 值都为 0。")
        continue

    # 调整图像大小为 250x250 像素（如果需要的话）
    # image = cv2.resize(image, (250, 250))

    # 提取文件名中的数字，第二次遇到下划线就停止提取
    x_value = ""
    for char in image_file.split('_')[1]:
        if char.isdigit():
            x_value += char
        else:
            break

    # 计算非黑色像素的数量
    non_black_pixels, threshold_image = calculate_non_black_pixels(image)

    # 输出非黑色像素的面积
    print(f"图像: {image_file}，非黑色像素面积: {non_black_pixels} 平方像素")

    # 更新合并前的质量总和
    mass_sum_before_merge.setdefault(x_value, 0)

    # 使用指定的公式计算质量
```

```
        s = non_black_pixels * f * f / d / d
        v = 4 * s * math.sqrt(s) / 3 / math.sqrt(3.1415926)
        mass = v * 0.8

        # 更新当前组的总质量
        mass_sum_before_merge[x_value] += mass

        # 更新当前组的文件名列表
        grouped_filenames.setdefault(x_value, [])
        grouped_filenames[x_value].append(image_file)

        # 以黑色为背景重建 RGB 图像
        result_image = image.copy()
        result_image[np.where(threshold_image == 0)] = 0

        # 在文件名中添加像素面积大小
        output_filename = f"result_{x_value}_{non_black_pixels}sq_{image_file}"
        output_path = os.path.join(output_folder, output_filename)
        cv2.imwrite(output_path, result_image)

        # 保存数字和总质量到文本文件
        text_output_path = os.path.join(output_folder, f"summary_{x_value}.txt")
        with open(text_output_path, 'w') as text_file:
            text_file.write(f"编号: {x_value}\n")
            text_file.write(f"总质量: {mass_sum_before_merge[x_value]}\n")

        print(f"处理后的图像保存在: {output_path}")
        print(f"摘要保存在: {text_output_path}")

    # 计算合并后的总质量
    for x_value, filenames in grouped_filenames.items():
        mass_sum_after_merge[x_value] =
sum(mass_sum_before_merge[filename.split('_')[1]] for filename in filenames)

    # 绘制合并前的质量分布图
    plt.hist(list(mass_sum_before_merge.values()), bins=20, edgecolor='black', alpha=0.5,
label='before merge')
    plt.title('Mass Distribution Before Merge')
    plt.xlabel('Mass')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()

    # 绘制合并后的质量分布图
```

```
plt.hist(list(mass_sum_after_merge.values()), bins=20, edgecolor='black', alpha=0.5, label='after merge')
plt.title('Mass Distribution After Merge')
plt.xlabel('Mass')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

P4.2:

```
import os
import cv2
import numpy as np

# 输入和输出文件夹路径
labels_folder = r"C:\Users\wangsiyu\Desktop\final_yolov5\final_yolov5\images"
output_folder = r"C:\Users\wangsiyu\Desktop\final_yolov5\single"

# 确保输出文件夹存在
os.makedirs(output_folder, exist_ok=True)

# 获取所有标签文件
label_files = [file for file in os.listdir(labels_folder) if file.endswith(".txt")]

# 处理每个标签文件
for label_file in label_files:
    # 构建标签文件的完整路径
    label_path = os.path.join(labels_folder, label_file)

    # 读取图像
    image_path = label_path.replace(".txt", ".jpg")
    image = cv2.imread(image_path)

    # 读取标签信息
    with open(label_path, "r") as f:
        lines = f.readlines()

    for line in lines:
        # 解析 YOLO 格式的标签
        parts = line.strip().split()
        x_center, y_center, width, height = map(float, parts[1:])

        # 计算矩形框的坐标
        x_min = int((x_center - width / 2) * image.shape[1])
```

```
            y_min = int((y_center - height / 2) * image.shape[0])
            x_max = int((x_center + width / 2) * image.shape[1])
            y_max = int((y_center + height / 2) * image.shape[0])

            # 提取苹果区域图像
            apple_region = image[y_min:y_max, x_min:x_max]

            # 对苹果区域进行处理（可以根据需要添加更多的图像处理操作）

            #  保存处理后的苹果区域图像
            output_path = os.path.join(output_folder, f"processed_{label_file.replace('.txt', '_')}_box{x_min}_{y_min}_{x_max}_{y_max}.jpg")
            cv2.imwrite(output_path, apple_region)
```

P5.1:

```
import os
import torch
from torchvision import transforms
from PIL import Image
from train import CustomModel  # Ensure the correct import of the CustomModel class
from tqdm import tqdm
import shutil
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Define preprocessing transform
img_size = 270  # Same as the image size used during training
transform = transforms.Compose([
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
])

# Specify the model path
model_path = r"C:\Users\hp\Desktop\checkpoints\6\resnet50_pretrained_270\resnet50_29epochs_accuracy096449_weights.pth"  # Modify to your model file path

# Load the model
model = CustomModel(model_name="resnet50", out_features=5, pretrained=True)  # Modify parameters based on your actual situation
model.load_state_dict(torch.load(model_path))
model.eval()

# Specify the folder containing images to predict and the target folder to save results
input_folder = r"E:\APMCC\Attachment\Attachment 3"  # Modify to your input folder
```

```
path
    output_folder = r"E:\APMCC\Attachment\1127_0247"    # Modify to your output folder path

    # Iterate over each image in the folder
    image_files = [f for f in os.listdir(input_folder) if f.endswith(('.jpg', '.jpeg', '.png'))]

    # Store true labels and predicted labels
    true_labels = []
    predicted_labels = []

    for image_file in tqdm(image_files, desc="Predicting"):
        image_path = os.path.join(input_folder, image_file)

        # Read the image and make predictions
        img = Image.open(image_path).convert("RGB")
        img = transform(img)
        img = img.unsqueeze(0)    # Add a dimension to match the model's input requirements

        with torch.no_grad():
            output = model(img)

        # Get the prediction result
        predicted_class = torch.argmax(torch.softmax(output, dim=1)).item()
        true_class = # Real label, obtain based on the actual situation

        # Store true labels and predicted labels
        true_labels.append(true_class)
        predicted_labels.append(predicted_class)

        # Determine the save path
        save_path = os.path.join(output_folder, f"class_{predicted_class}", image_file)

        # Create the save path folder (if it does not exist)
        os.makedirs(os.path.dirname(save_path), exist_ok=True)

        # Copy the file to the save path
        shutil.copy(image_path, save_path)

    # Build the confusion matrix
    conf_matrix = confusion_matrix(true_labels, predicted_labels)

    # Print the confusion matrix and classification report
```

```
print("Confusion Matrix:")
print(conf_matrix)

print(r"\nClassification Report:")
print(classification_report(true_labels, predicted_labels))
```

P5.2:

```
import os
import shutil

image_path = r'E:\APMCC\Attachment\apple\images'   # 图片文件
txt_path = r'E:\APMCC\Attachment\apple\labels'   # 标签文件
new_file_path = r'E:\APMCC\Attachment\apple_split'   # 划分数据后的文件
train_rate = 0.8   # 训练集比例
val_rate = 0.2   # 验证集比例

# 将有对应标签的图片找出来，放到新文件夹下
labels = []
for label in os.listdir(txt_path):
    labels.append(os.path.splitext(label)[0])
for image_name in os.listdir(image_path):
    image_name = os.path.splitext(image_name)[0]

    if image_name in labels:
        image_name = image_name + ".jpg"
        shutil.copy(image_path + '/' + image_name, new_file_path)

# 计算训练集与验证集数量
images = []
for image in os.listdir(new_file_path):
    images.append(image)
    total = len(images)
    train_images = images[0:int(train_rate * total)]
    val_images = images[int(train_rate * total):int((train_rate + val_rate) * total)]

# 图片-train
for image in train_images:
    print(image)
    old_path = new_file_path + '/' + image
    new_path1 = new_file_path + '/' + 'images' + '/' + 'train'
    # new_path1 = new_file_path + '/' + 'train' + '/' + 'images'
    if not os.path.exists(new_path1):
        os.makedirs(new_path1)
    # new_path = new_path1 + '/' + image
```

```
        shutil.copy(old_path, new_path1)

# 图片-val
for image in val_images:
        old_path = new_file_path + '/' + image
        new_path1 = new_file_path + '/' + 'images' + '/' + 'val'
        # new_path1 = new_file_path + '/' + 'val' + '/' + 'images'
        if not os.path.exists(new_path1):
                os.makedirs(new_path1)
        # new_path = new_path1 + '/' + image
        shutil.copy(old_path, new_path1)

#   标签-train
images1 = []
for image in os.listdir(new_file_path + '/' + 'images' + '/' + 'train'):
        images1.append(os.path.splitext(image)[0])
for label_name in os.listdir(txt_path):
        label_name = os.path.splitext(label_name)[0]
        if label_name in images1:
                label_name = label_name + ".txt"
                label_train_path = new_file_path + '/' + 'labels' + '/' + 'train'
                if not os.path.exists(label_train_path):
                        os.makedirs(label_train_path)
                shutil.copy(txt_path + '/' + label_name, label_train_path)
                shutil.copy(txt_path + '/' + 'classes.txt', label_train_path)

# 标签-val
images2 = []
for image in os.listdir(new_file_path + '/' + 'images' + '/' + 'val'):
        images2.append(os.path.splitext(image)[0])
for label_name in os.listdir(txt_path):
        label_name = os.path.splitext(label_name)[0]
        if label_name in images2:
                label_name = label_name + ".txt"
                label_val_path = new_file_path + '/' + 'labels' + '/' + 'val'
                if not os.path.exists(label_val_path):
                        os.makedirs(label_val_path)
                shutil.copy(txt_path + '/' + label_name, label_val_path)
                shutil.copy(txt_path + '/' + 'classes.txt', label_val_path)

# 删除新文件夹下对应标签的图片
for name in os.listdir(new_file_path):
        if name.endswith('.jpg'):
                os.remove(os.path.join(new_file_path, name))
```

```
if image_name in labels:
    image_name = image_name + ".jpg"
shutil.copy(image_path + '/' + image_name, new_file_path)

# 计算训练集与验证集数量
images = []
for image in os.listdir(new_file_path):
    images.append(image)
total = len(images)
train_images = images[0:int(train_rate * total)]
val_images = images[int(train_rate * total):int((train_rate + val_rate) * total)]

# 图片-train
for image in train_images:
    print(image)
old_path = new_file_path + '/' + image
new_path1 = new_file_path + '/' + 'images' + '/' + 'train'
# new_path1 = new_file_path + '/' + 'train' + '/' + 'images'
if not os.path.exists(new_path1):
    os.makedirs(new_path1)
# new_path = new_path1 + '/' + image
shutil.copy(old_path, new_path1)

# 图片-val
for image in val_images:
    old_path = new_file_path + '/' + image
new_path1 = new_file_path + '/' + 'images' + '/' + 'val'
# new_path1 = new_file_path + '/' + 'val' + '/' + 'images'
if not os.path.exists(new_path1):
    os.makedirs(new_path1)
# new_path = new_path1 + '/' + image
shutil.copy(old_path, new_path1)

# 标签-train
images1 = []
for image in os.listdir(new_file_path + '/' + 'images' + '/' + 'train'):
    images1.append(os.path.splitext(image)[0])
for label_name in os.listdir(txt_path):
    label_name = os.path.splitext(label_name)[0]
if label_name in images1:
    label_name = label_name + ".txt"
label_train_path = new_file_path + '/' + 'labels' + '/' + 'train'
if not os.path.exists(label_train_path):
```

```
        os.makedirs(label_train_path)
    shutil.copy(txt_path + '/' + label_name, label_train_path)
    shutil.copy(txt_path + '/' + 'classes.txt', label_train_path)

    # 标签-val
    images2 = []
    for image in os.listdir(new_file_path + '/' + 'images' + '/' + 'val'):
        images2.append(os.path.splitext(image)[0])
    for label_name in os.listdir(txt_path):
        label_name = os.path.splitext(label_name)[0]
    if label_name in images2:
        label_name = label_name + ".txt"
    label_val_path = new_file_path + '/' + 'labels' + '/' + 'val'
    if not os.path.exists(label_val_path):
        os.makedirs(label_val_path)
    shutil.copy(txt_path + '/' + label_name, label_val_path)
    shutil.copy(txt_path + '/' + 'classes.txt', label_val_path)

    # 删除新文件夹下对应标签的图片
    for name in os.listdir(new_file_path):
        if name.endswith('.jpg'):
            os.remove(os.path.join(new_file_path, name))
```

P5.3:

```
import os
import os.path as osp
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
from torchvision import models
from torchvision import datasets
import pretrainedmodels
import timm
from tqdm import tqdm
from torchutils import MetricMonitor, calculate_f1_macro, calculate_recall_macro, accuracy, adjust_learning_rate
import matplotlib.pyplot as plt
import numpy as np

if torch.cuda.is_available():
    device = torch.device('cuda:0')
```

```
    else:
        device = torch.device('cpu')
    print(f'Using device: {device}')

    # Set a fixed random seed for reproducibility
    seed = 42
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = True

    data_path = r"E:\APMCC\Attachment\Attachment 2_split"    # todo Dataset path
    # Note: Please split the dataset before executing

    # Hyperparameter settings
    params = {
        'model': 'resnet50',    # Choose a pretrained model
        "img_size": 270,    # Image input size
        "train_dir": osp.join(data_path, "train"),    # todo Training set path
        "val_dir": osp.join(data_path, "val"),    # todo Validation set path
        'device': device,    # Device
        'lr': (1e-3)*2,    # Learning rate
        'batch_size': 32,    # Batch size
        'num_workers': 0,    # Number of processes
        'epochs': 50,    # Number of epochs
        "save_dir": "../checkpoints/6",    # todo Save path
        "pretrained": True,
        "num_classes": len(os.listdir(osp.join(data_path, "train"))),    # Number of classes,
adaptively obtain the number of classes
        'weight_decay': 1e-5    # Learning rate decay
    }

    def get_torch_transforms(img_size):
        # Define image transformations for training and validation
        train_transform = transforms.Compose([
            transforms.RandomResizedCrop(img_size),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])

        val_transform = transforms.Compose([
```

```
            transforms.Resize(int(img_size * 1.1)),
            transforms.CenterCrop(img_size),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])

        return {'train': train_transform, 'val': val_transform}

    # Define the model
    class CustomModel(nn.Module):
        def __init__(self, model_name=params['model'],
out_features=params['num_classes'], pretrained=True):
            super().__init__()

            if pretrained:
                self.model = pretrainedmodels.__dict__[model_name](num_classes=1000,
pretrained='imagenet')
            else:
                self.model = pretrainedmodels.__dict__[model_name](num_classes=1000,
pretrained=None)

            # Modify the classifier
            if model_name.startswith("res"):
                n_features = self.model.last_linear.in_features
                self.model.last_linear = nn.Linear(n_features, out_features)
            elif model_name.startswith("vit"):
                n_features = self.model.head.in_features
                self.model.head = nn.Linear(n_features, out_features)
            else:
                n_features = self.model.last_linear.in_features
                self.model.last_linear = nn.Linear(n_features, out_features)

            print(self.model)

        def forward(self, x):
            x = self.model(x)
            return x

    # Define the training process
    def train(train_loader, model, criterion, optimizer, epoch, params):
        metric_monitor = MetricMonitor()
        model.train()
        nBatch = len(train_loader)
        stream = tqdm(train_loader)
```

```
        for i, (images, target) in enumerate(stream, start=1):
            images = images.to(params['device'], non_blocking=True)
            target = target.to(params['device'], non_blocking=True)
            output = model(images)
            loss = criterion(output, target.long())
            f1_macro = calculate_f1_macro(output, target)
            recall_macro = calculate_recall_macro(output, target)
            acc = accuracy(output, target)
            metric_monitor.update('Loss', loss.item())
            metric_monitor.update('F1', f1_macro)
            metric_monitor.update('Recall', recall_macro)
            metric_monitor.update('Accuracy', acc)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            lr = adjust_learning_rate(optimizer, epoch, params, i, nBatch)
            stream.set_description(
                "Epoch: {epoch}. Train.      {metric_monitor}".format(
                    epoch=epoch,
                    metric_monitor=metric_monitor)
            )
        return metric_monitor.metrics['Accuracy']["avg"], metric_monitor.metrics['Loss']["avg"]

    # Define the validation process
    def validate(val_loader, model, criterion, epoch, params):
        metric_monitor = MetricMonitor()
        model.eval()
        stream = tqdm(val_loader)
        with torch.no_grad():
            for i, (images, target) in enumerate(stream, start=1):
                images = images.to(params['device'], non_blocking=True)
                target = target.to(params['device'], non_blocking=True)
                output = model(images)
                loss = criterion(output, target.long())
                f1_macro = calculate_f1_macro(output, target)
                recall_macro = calculate_recall_macro(output, target)
                acc = accuracy(output, target)
                metric_monitor.update('Loss', loss.item())
                metric_monitor.update('F1', f1_macro)
                metric_monitor.update("Recall", recall_macro)
                metric_monitor.update('Accuracy', acc)
                stream.set_description(
                    "Epoch: {epoch}. Validation. {metric_monitor}".format(
```

```
                    epoch=epoch,
                    metric_monitor=metric_monitor)
            )
        return metric_monitor.metrics['Accuracy']["avg"], metric_monitor.metrics['Loss']["avg"]

    # Display training curves
    def show_loss_acc(acc, loss, val_acc, val_loss, save_dir):
        plt.figure(figsize=(8, 8))
        plt.subplot(2, 1, 1)
        plt.plot(acc, label='Training Accuracy')
        plt.plot(val_acc, label='Validation Accuracy')
        plt.legend(loc='lower right')
        plt.ylabel('Accuracy')
        plt.ylim([min(plt.ylim()), 1])
        plt.title('Training and Validation Accuracy')

        plt.subplot(2, 1, 2)
        plt.plot(loss, label='Training Loss')
        plt.plot(val_loss, label='Validation Loss')
        plt.legend(loc='upper right')
        plt.ylabel('Cross Entropy')
        plt.title('Training and Validation Loss')
        plt.xlabel('epoch')
        save_path = osp.join(save_dir, "results.png")
        plt.savefig(save_path, dpi=100)

    if __name__ == '__main__':
        accs = []
        losss = []
        val_accs = []
        val_losss = []
        data_transforms = get_torch_transforms(img_size=params["img_size"])
        train_transforms = data_transforms['train']
        valid_transforms = data_transforms['val']
        train_dataset = datasets.ImageFolder(params["train_dir"], train_transforms)
        valid_dataset = datasets.ImageFolder(params["val_dir"], valid_transforms)
        if params['pretrained'] == True:
            save_dir = osp.join(params['save_dir'], params['model']+"_pretrained_" + str(params["img_size"]))
        else:
            save_dir = osp.join(params['save_dir'], params['model'] + "_nopretrained_" + str(params["img_size"]))
        if not osp.isdir(save_dir):
```

```
        os.makedirs(save_dir)
        print("save dir {} created".format(save_dir))
    train_loader = DataLoader(
        train_dataset, batch_size=params['batch_size'], shuffle=True,
        num_workers=params['num_workers'], pin_memory=True,
    )
    val_loader = DataLoader(
        valid_dataset, batch_size=params['batch_size'], shuffle=False,
        num_workers=params['num_workers'], pin_memory=True,
    )
    print(train_dataset.classes)
    model = CustomModel(model_name=params['model'],
out_features=params['num_classes'],
                    pretrained=params['pretrained'])
    model = model.to(params['device'])
    criterion = nn.CrossEntropyLoss().to(params['device'])
    optimizer = torch.optim.AdamW(model.parameters(), lr=params['lr'],
weight_decay=params['weight_decay'])
    best_acc = 0.0
    for epoch in range(1, params['epochs'] + 1):
        acc, loss = train(train_loader, model, criterion, optimizer, epoch, params)
        val_acc, val_loss = validate(val_loader, model, criterion, epoch, params)
        accs.append(acc)
        losss.append(loss)
        val_accs.append(val_acc)
        val_losss.append(val_loss)
        if val_acc >= best_acc:
            save_path = osp.join(save_dir,
f"{params['model']}_{epoch}epochs_accuracy{acc:.5f}_weights.pth")
            torch.save(model.state_dict(), save_path)
            best_acc = val_acc
            show_loss_acc(accs, losss, val_accs, val_losss, save_dir)
    print("Training completed. Model and training logs saved in: {}".format(save_dir))
```