Team Number :	apmcm2308882
Problem Chosen :	A

2023 APMCM summary sheet

By training a model for apple detection, the issue of apple quantity estimation has been resolved. Analyzing the data in the 'labels' folder generated by the model addressed the problem of apple position estimation. Employing OpenCV for color extraction enabled the assessment of apple ripeness based on color, resolving the issue of determining maturity. Calculating the radius and area from coordinates in the 'labels' file, and subsequently estimating apple mass based on the relationship between mass and area, tackled the problem of mass estimation. Training a new model to recognize and detect different fruit types resolved the issue of apple identification.

For Question One, the dataset was partitioned into training, validation, and test sets to iteratively train the YOLOv5 model. Model accuracy was ensured by comparing it with the original dataset and validating against the validation set. Subsequently, the entire Attachment 1 was subjected to recognition and detection, resulting in the 'labels' folder. Counting the lines within each label in this folder reliably quantified the number of apples, given the model's accuracy during training.

For Question Two, averaging the coordinates in the 'labels' file generated by running detect.py determined the position of each apple. Given the high accuracy of our model in Question One, ensuring the reliability of each apple's location.

For Question Three, utilizing the data in the 'labels' file generated by running detect.py, coordinates for selected apples were extracted. Processing these images with OpenCV, where only red and green color channels were retained, allowed for the calculation of maturity. Assuming red pixels represent 1 and green pixels represent -1, the sum of these values divided by the total number of red and green pixels provided a maturity score ranging from -1 to 1. The range (-1, -0.33) was considered immature, (-0.33, 0.33) moderately ripe, and (0.33, 1) mature. Using these values, apple quality was estimated, highlighting the rigor and reliability of our approach.

For Question Four, extracting the length and width coordinates from the 'labels' file generated by running detect.py, we assumed an apple to be a standard circle. The sum of length and width divided by 2 approximated the side length of the square bounding box, which when divided by 2 gave the radius of our standard circular apple. The two-dimensional area of the apple was then calculated using the formula for the area of a circle. Considering the positive correlation between surface area and mass, apple mass was estimated, showcasing the reliability of our method based on fundamental formulas and logical rigor.

For Question Five, using the data in Attachment 2 as the training set, the YOLOv8 model was iteratively trained, and Attachment 3 was used as the test set for apple recognition and detection. Given the sufficiently large dataset and the ensured model precision, the results are reliable.

Keywords: Image Recognition, Object Detection, YOLO, Feature Extraction, Deep Learning，Model Training

Contents

1 Introduction

1.1 Problem Background

China, renowned as the leading global apple producer with an annual yield of around 35 million tons, also holds the position of the world's primary apple exporter. China's significant contribution encompasses half of the world's apples and over one-sixth of the globally exported apples. This achievement is bolstered by China's Belt and Road Initiative (BRI), a transformative effort aimed at fostering a collaborative global community. Consequently, nations such as Vietnam, Bangladesh, the Philippines, and Indonesia along the BRI route have emerged as key destinations for Chinese apple exports.

The process of apple harvesting predominantly relies on manual labor. As apples reach maturity, the demand for a substantial workforce surges in apple-producing regions over a short period. However, the majority of local farmers cultivate apples in their personal orchards. The challenges of an aging agricultural workforce and the migration of younger individuals from villages for employment have resulted in a shortage of labor during the crucial apple-picking season. To address this issue, China initiated research into apple-picking robots around 2011, achieving noteworthy advancements.

Despite substantial progress, the widespread adoption of various apple-picking robots globally falls short of expectations due to disparities between orchard environments and controlled experimental settings. In intricate and unstructured orchard landscapes, existing robots often struggle to accurately discern obstacles such as "leaf occlusion," "branch occlusion," "fruit occlusion," and "mixed occlusion." Picking apples without precise judgments based on real-world scenarios poses a considerable risk of fruit damage and potential harm to both picking hands and mechanical arms. This compromises harvesting efficiency and fruit quality, leading to substantial losses. Furthermore, the identification and classification of various harvested fruits, crucial for post-harvest procedures like sorting, processing, packaging, and transportation, pose challenges. The similarity in colors, shapes, and sizes between apples and many other fruits adds complexity to the post-harvest identification of apples. In light of these challenges, there is a pressing need for robust mathematical modeling to enhance apple recognition and harvesting processes.

1.2 Restatement of the Problem

Considering the background information and restricted conditions identified in the problem statement, we need to solve the following problems:

- Question 1:
  Utilizing the provided image dataset of ripe apples (Attachment 1), develop a mathematical model to extract image features and determine the count of apples in each image. Present the distribution of apple counts in Attachment 1 through a histogram.
- Question 2:
  Leveraging the harvest-ready apple image dataset in Attachment 1, pinpoint the positions of apples in each image with the coordinate origin set at the left bottom corner. Generate a two-dimensional scatter diagram illustrating the geometric coordinates of

all apples in Attachment 1.

- Question 3:
  With the image dataset of ripe apples (Attachment 1), establish a mathematical model to assess the maturity of apples in each image. Illustrate the distribution of apple maturity levels in Attachment 1 through a histogram.
- Question 4:
  Based on the image dataset of ripe apples (Attachment 1), compute the two-dimensional area of apples in each image, considering the bottom left corner of the image as the coordinate origin. Estimate the masses of the apples and visualize the mass distribution through a histogram in Attachment 1.
- Question 5:
  Using the dataset of harvested fruit images (Attachment 2), extract relevant features to train a model specifically for apple recognition. Apply this model to identify apples in Attachment 3 and create a histogram showcasing the distribution of ID numbers for all apple images in Attachment 3.

1.3 Our Work

For the first question, generating a histogram depicting the distribution of all apples in Attachment 1, we initially conducted an analysis using the YOLOv5l6 model, a single-stage object detection algorithm within the YOLOv5 framework. The model was trained on a designated dataset, necessitating the annotation of apple instances within the dataset. Subsequently, the dataset was partitioned, and the model was configured and trained. The final step involved counting the apples based on the number of entries in the label folder, enabling the construction of a histogram illustrating the spatial distribution of apples.

Addressing the second question involves estimating the positions of apples. This is accomplished by running 'detect.py' to generate a set of labels in the form of coordinate values stored in the 'txt' files. These coordinates are then utilized to plot a scatter plot depicting the geometric positions of the apples.

Concerning the third question regarding the estimation of apple ripeness, color analysis is essential. Extraction of apple colors is performed, followed by averaging the colors and eliminating extraneous hues. This process allows for a judgment of apple ripeness based on color. The final step involves creating a histogram illustrating the distribution of apple ripeness.

Regarding the fourth question, estimating apple quality involves using the coordinates from the previously generated label set. The two-dimensional area of each apple in the images is calculated, and, employing a proportional relationship between area and quality, the quality of each apple is estimated. The result is a histogram presenting the distribution of apple quality.

For the fifth question, apple recognition is accomplished by dividing the dataset in Attachment 2 into training and testing sets. The YOLOv8l-cls classification model within the YOLOv8 framework is utilized for further recognition training. Subsequently, the model is applied to identify apples in the images from Attachment 3, extracting the corresponding IDs. Finally, a histogram is generated to illustrate the distribution of ID numbers for all identified apple images in Attachment 3.

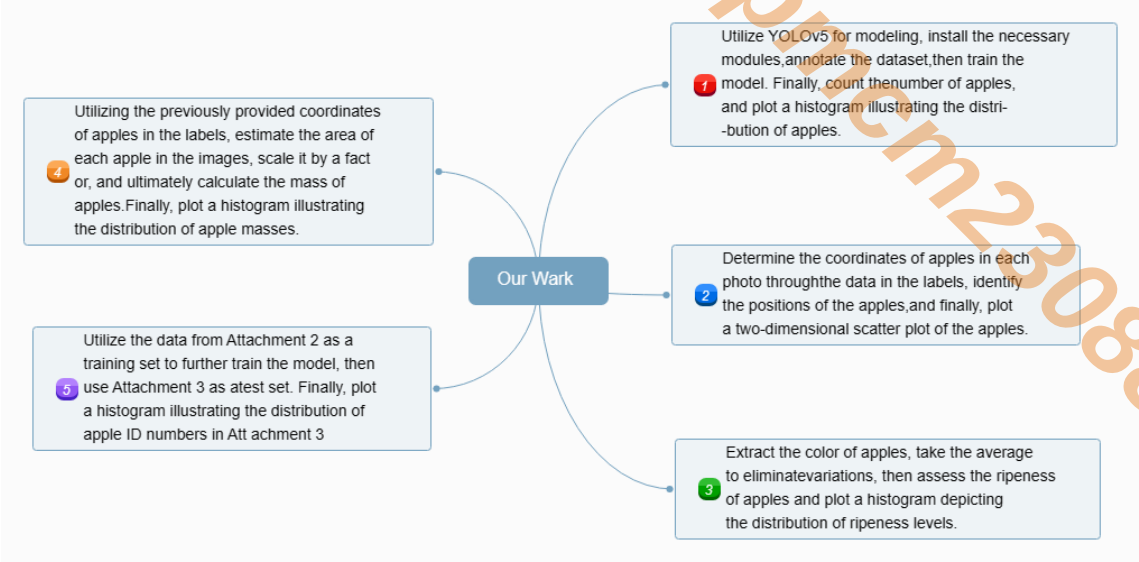

Figure 0:Our Wark

2 Assumptions and Justifications

1. Correlation between Ripeness and Color:

Assumption: The degree of ripeness in apples is positively correlated with the intensity of red color and negatively correlated with the intensity of green color.

Justification: This assumption is grounded in the common understanding that apples typically change color as they ripen, transitioning from green to red. Therefore, it is reasonable to hypothesize that the degree of ripeness may be reflected in the color spectrum of red and green in apples.

2. Linear Relationship between Quality and Surface Area:

Assumption: The quality of apples is linearly correlated with the surface area of the apples.

Justification: This assumption is based on the premise that the quality or mass of an apple may be proportionally related to its surface area. A linear correlation is posited to simplify the relationship between quality and surface area for the purpose of estimation. This assumption allows for a straightforward and interpretable model for estimating apple quality based on geometric attributes.

3 Notations

Table 1: Notations used in this paper

Symbol	Description	Unit
Loss	Loss Function	
r_w	Ratio of the Ground Truth Box Width	
r_h	Ratio of the Ground Truth Box Height	
TP	True Positive	
TN	True Negative	
FP	False Positive	
FN	False Negative	

4 Number, coordinates, maturity and quality of apples

4.1 Data Description

We compiled a dataset consisting of images of apples, totaling 1222 pictures. The images were annotated using the labelimg tool, resulting in the creation of XML files. Subsequently, a Python script was developed to facilitate format conversion, transforming the XML files into TXT files. Another script, named spliteDatasets, was implemented to partition the dataset into training and validation sets, as well as a test set.

This process resulted in the creation of four distinct folders:

1. The "Annotations" folder: This directory is designated for storing XML files, each containing annotations generated through labelimg for individual images.

2. The "Images" folder: This folder houses the original dataset images intended for training, with a standardized image format of JPG.

3. The "ImageSets" folder: This folder stores files representing the partitioning of the dataset into subsets for training, validation, and testing.

4. The "Labels" folder: This folder accommodates TXT-format annotation files, which were obtained through the conversion of XML-formatted annotation files.

Subsequent to this organization, the next phase involved the training of an apple detection model.

4.2 Model Training

4.2.1 Tool Selection

PyTorch, an open-source machine learning library, is extensively employed in applications of deep learning and artificial intelligence[1]. Renowned for its high flexibility and speed, PyTorch has become a favorite among researchers and developers.

Performance Benchmark: PyTorch is acclaimed for its usability and flexibility.

Compatibility and Integration: It exhibits exceptional compatibility with various data science and machine learning libraries in Python.

Community Support and Ecosystem: PyTorch boasts one of the most vibrant communities, offering extensive learning and troubleshooting resources.

Case Studies: It is commonly used for research prototypes, with many academic papers referencing models deployed in PyTorch.

Hardware Acceleration: Support for CUDA enables GPU acceleration, crucial for expediting both model training and inference.

Therefore, we opted for PyTorch as the deployment library for our model.

Given the requirement to detect the number of apples in images, an effective object detection algorithm is essential. YOLO[2], standing for "You Only Look Once," is a robust object detection algorithm.

Compared to its predecessors, YOLOv5 incorporates various data augmentation techniques to enhance model generalization and mitigate overfitting. Several sophisticated training strategies are applied, including:

Multi-scale Training: During training, input images are randomly resized within a range of 0.5 to 1.5 times the original size.

Automatic Anchors: This strategy optimizes previous anchor boxes to match the statistical characteristics of ground truth boxes in custom data.

Warm-up and Cosine LR Scheduler: A method adjusting the learning rate to enhance model performance.

Exponential Moving Average (EMA): A strategy using the average values of parameters from previous steps to stabilize training and reduce generalization errors.

Mixed Precision Training: A method performing computations in half-precision format, reducing memory usage and accelerating computation.

Hyperparameter Evolution: A strategy for automatically tuning hyperparameters for optimal performance.

Additionally, loss calculation is performed.

The loss in YOLOv5 is computed as a combination of three individual loss components:

Classes Loss (BCE Loss): Binary Cross-Entropy loss, measures the error for the classification task.

Objectness Loss (BCE Loss): Another Binary Cross-Entropy loss, calculates the error in detecting whether an object is present in a particular grid cell or not.

Location Loss (CIoU Loss): Complete IoU loss, measures the error in localizing the object within the grid cell.

$$Loss = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_3 L_{loc} \tag{1}$$

In addressing the mitigation of grid sensitivity, YOLOv5 introduces updated formulas for predicting bounding box coordinates compared to earlier versions of YOLO[3]. This modification aims to reduce grid sensitivity and prevent the model from predicting unbounded box sizes.

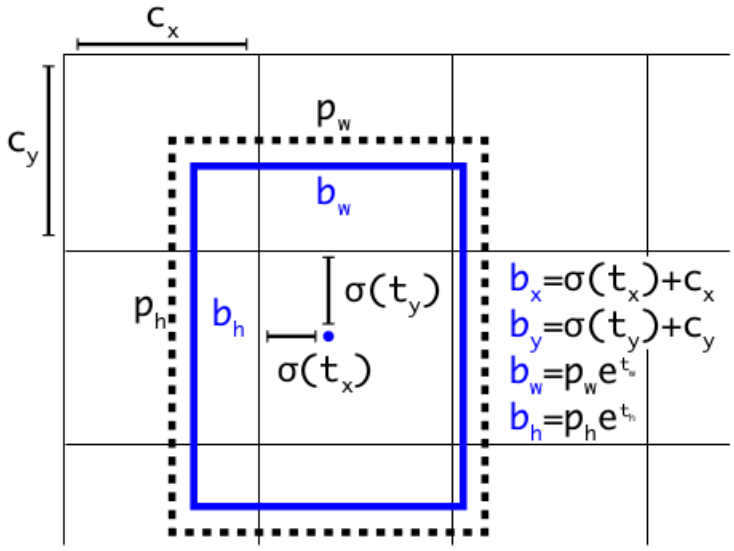

Figure 1:illustrates the bounding boxes predicted by the model.

The revised formula for computing predicted bounding boxes is as follows:

$$b_x = (2 \cdot \sigma(t_x) - 0.5) + c_x \, b_y = (2 \cdot \sigma(t_y) - 0.5) + c_y \, b_w$$
$$= P_w \cdot (2 \cdot \sigma(t_w))^2 \, b_h = p_h \cdot (2 \cdot \sigma(t_h))^2 \quad (2)$$

Comparing the offset of the center point before and after scaling, the range of center point offset is adjusted from (0, 1) to (-0.5, 1.5). Consequently, the offset can be easily obtained as either 0 or 1.

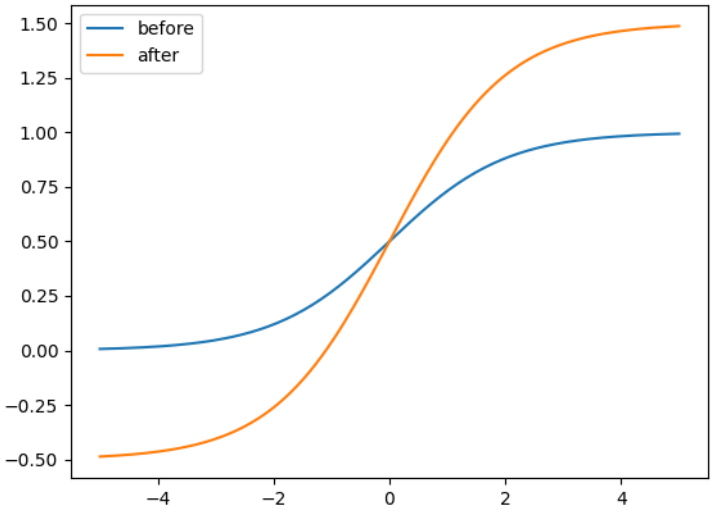

Figure 2: Center point offsets before and after scaling.

Comparing the scaling ratios of height and width before and after adjustments (relative to anchor points). The original yolo/darknet box equation had a significant flaw. Width and height were entirely unconstrained, as they were simply defined as out=exp(in). This posed a serious risk, leading to gradient instability, NaN losses, and ultimately jeopardizing the entire training process.

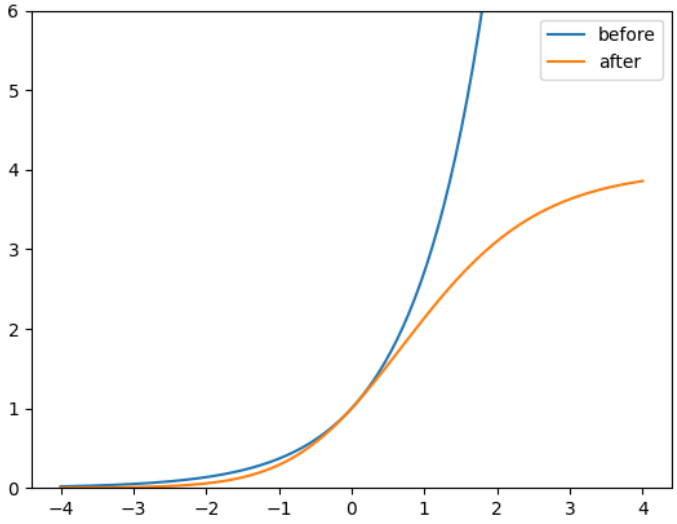

Figure 3: Scaling ratios of height and width before and after adjustments.

The Process of Constructing Targets in YOLOv5[4]:

The process of constructing targets in YOLOv5 is crucial for training efficiency and model

accuracy. It involves assigning ground truth boxes to corresponding grid cell pixels in the output map and matching them with the respective anchor boxes.

This process follows the steps outlined below:

Calculate the ratio of the ground truth box dimensions to the dimensions of each anchor template.

$$r_w = w_{gt}/w_{at} \tag{3}$$

$$r_h = h_{gt}/h_{at} \tag{4}$$

$$r_w^{\max} = \max(r_w, 1/r_w) \tag{5}$$

$$r_h^{\max} = \max(r_h, 1/r_h) \tag{6}$$

$$r^{\max} = \max(r_w^{\max}, r_h^{\max}) \tag{7}$$

$$r^{\max} < anchor_t \tag{8}$$

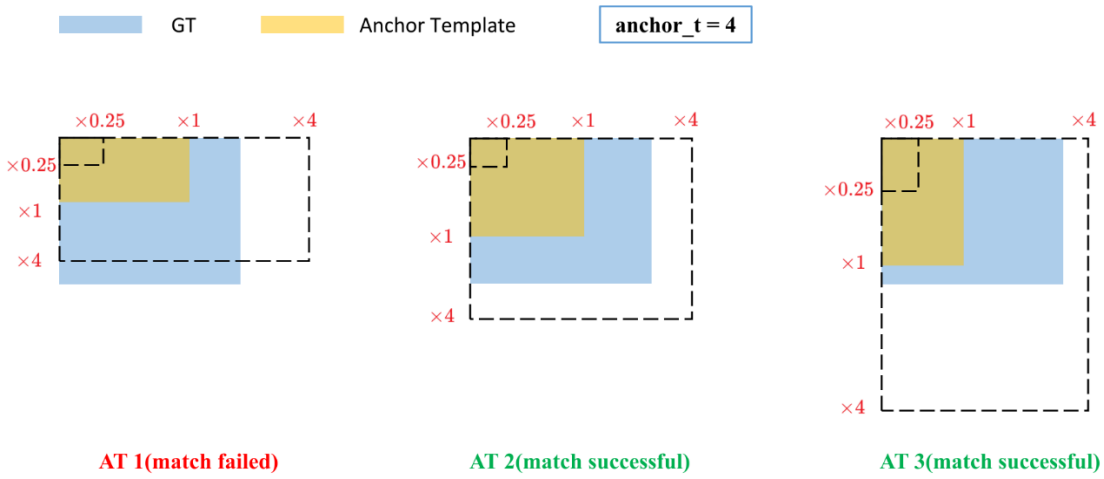

Figure 4: Anchor Template Dimensions

If the calculated ratio falls within the threshold range, the ground truth box is matched with the corresponding anchor.

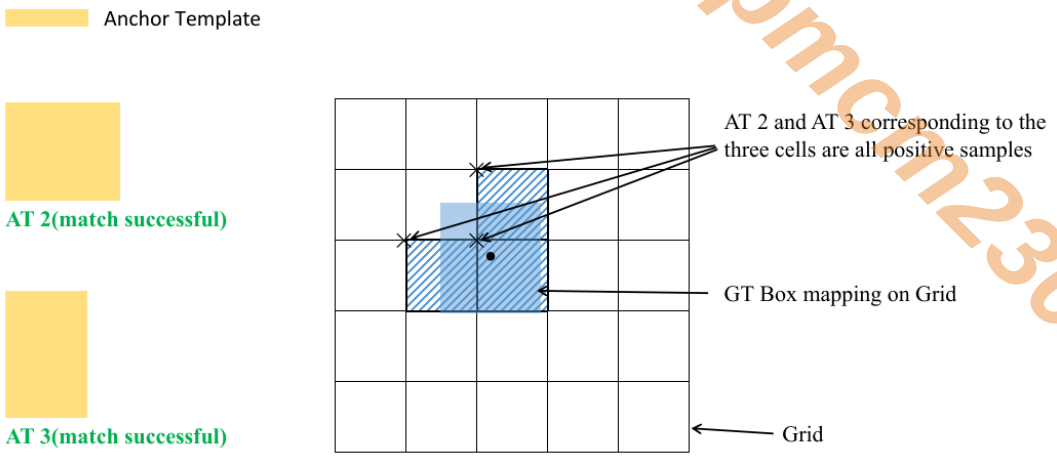

Figure 5: Matching Ground Truth Boxes with Corresponding Anchors

The matching anchors are assigned to the respective pixels. It is noteworthy that, due to the adjustment in center point offset, one ground truth box can be assigned to multiple anchors. This is because the center point offset range has been adjusted from (0, 1) to (-0.5, 1.5), allowing a ground truth box to be assigned to a greater number of anchors.

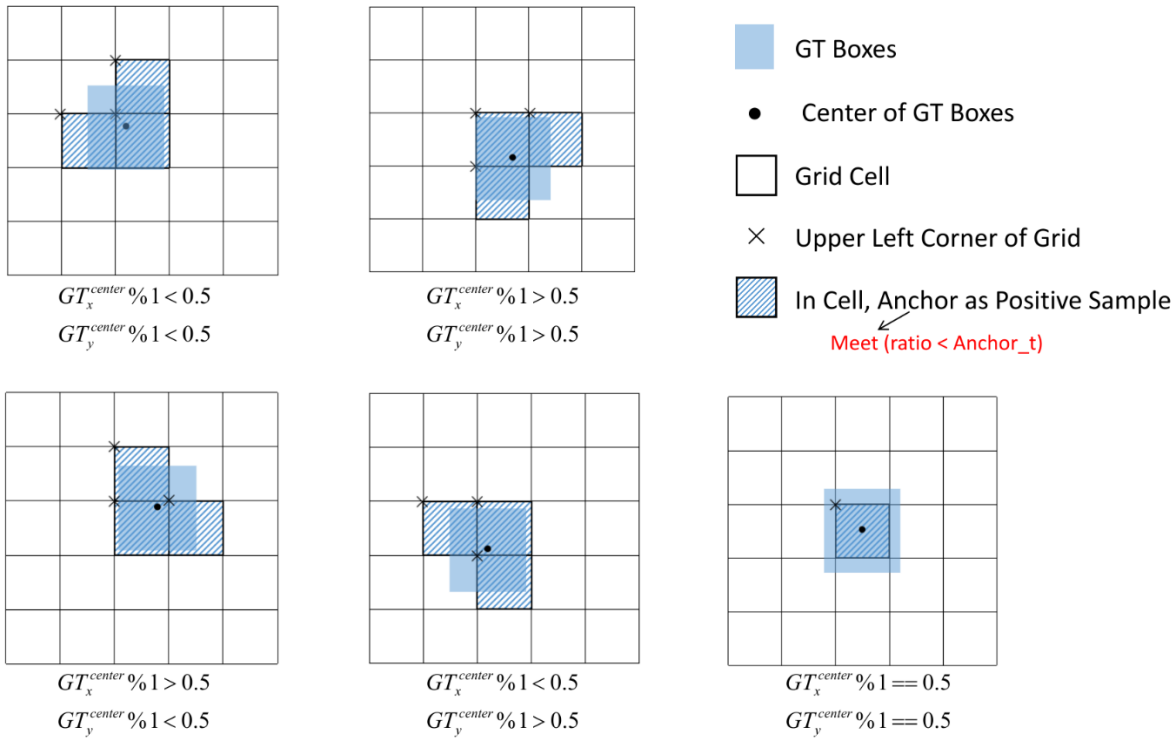

Figure 6: Training Process

This way, the target construction process ensures the accurate allocation and matching of each ground truth object during the training process, enabling YOLOv5 to learn the task of object detection more effectively.

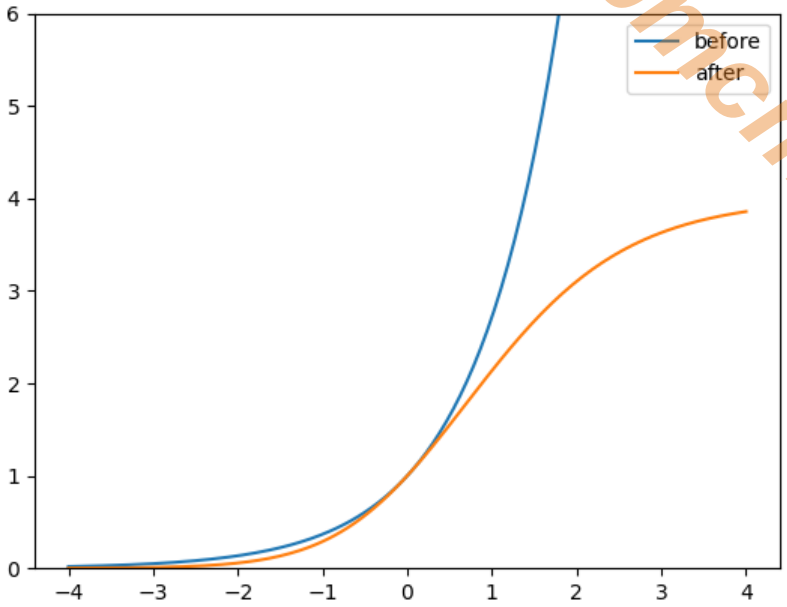

Figure 7: Comparison Before and After Target Construction

4.2.2 Selection of the Pre-trained Model

Therefore, we opted for the YOLOv5 object detection model, taking into account the characteristics of various pre-trained models[5]. Specifically, we chose the yolov5l6pt pre-trained model for the model training. Subsequently, the evaluation results for each pre-trained model on the COCO AP dataset are presented below.

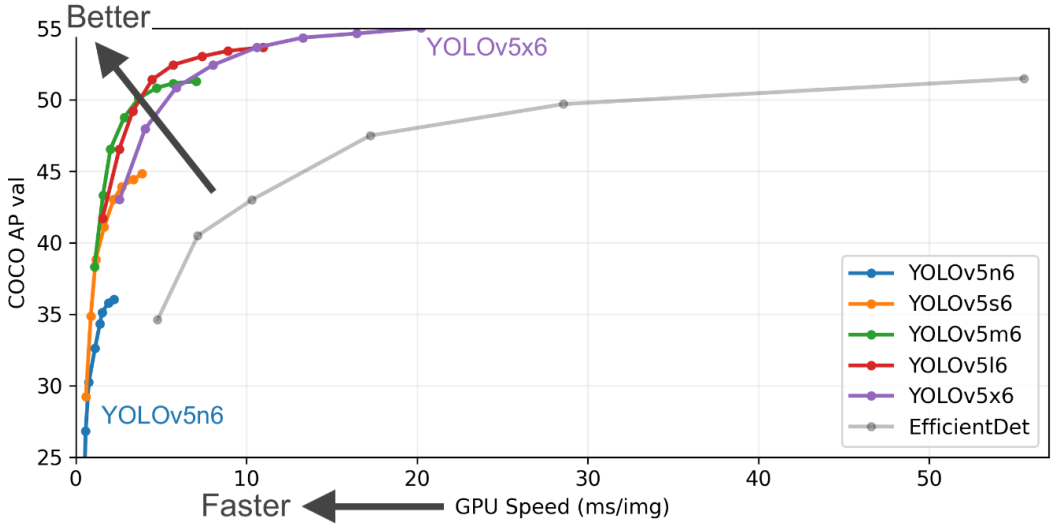

Figure 8: Evaluation Results of Various Pre-trained Models on the COCO AP Dataset

4.2.3 Training Our Own Model

To initiate the training of our apple detection model, we specified our trained dataset, batch size, image size, and the pre-trained YOLOv5l6 model selected earlier.

1. Hyperparameter Configuration: YOLOv5 involves approximately 30 hyperparameters for various training configurations, defined in files within the directory. Properly initializing these values before evolution is crucial, as better initial guesses lead to superior final results.

2. Fitness Definition: Fitness is the value we aim to maximize. In YOLOv5, we defined

the default fitness function as a weighted combination of metrics: 10% weight for a specified contribution and the remaining 90% for precision (P) and recall (R).

3. Evolution: Evolution revolves around the fundamental scenario of improvement sought. The basic approach involves fine-tuning the pre-trained YOLOv5l6 on the apple dataset for 200 epochs and maximizing the fitness defined in section 2. The primary genetic operators employed in evolution are crossover and mutation. In this work, we utilized mutation with a probability of 82%, a variance of 0.04, and the creation of new offspring based on combinations of the best parents from previous generations. Evolution spans 200 generations for optimal results.

4. Visualization: The evolve.csv, after evolution, is plotted, with each hyperparameter having a subplot depicting fitness (y-axis) against hyperparameter values (x-axis). Yellow indicates higher concentration. The vertical distribution signifies parameters that have been disabled and will not undergo further changes.

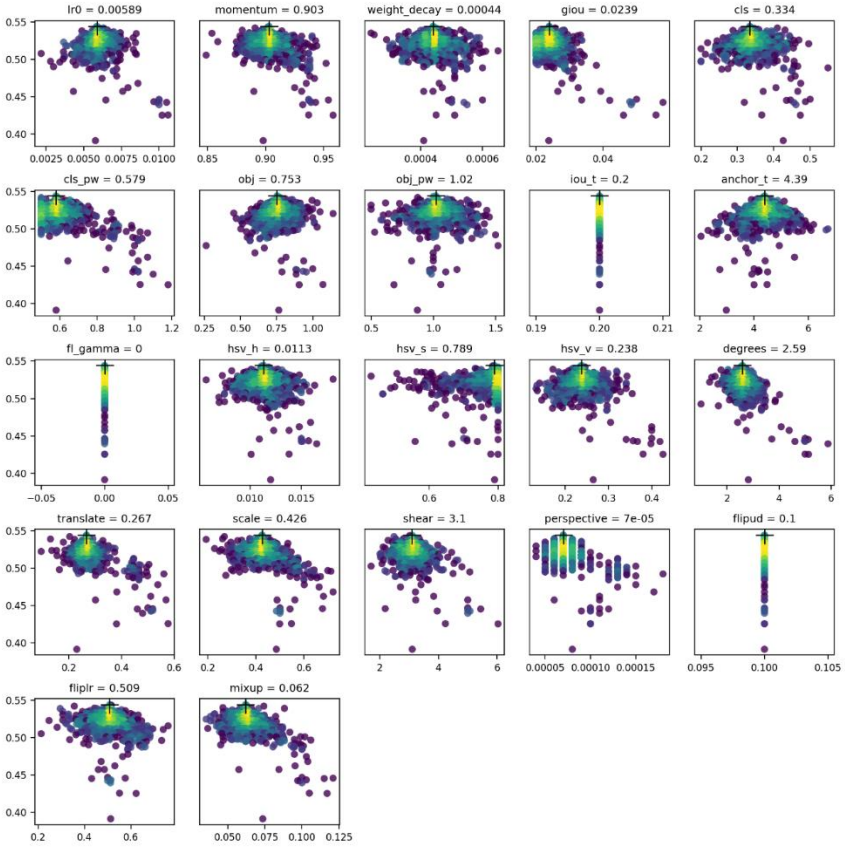

Figure 9: Fitness of Hyperparameters

The model training is complete.

4.3 Model Evaluation

The confusion matrix, employed for assessing the model's classification accuracy, provides a detailed account of correct and incorrect predictions[6]. It delineates:

- True Positive (TP): Instances where the sample's true class is positive, and the model correctly predicts it as positive.

- True Negative (TN): Instances where the sample's true class is negative, and the model correctly predicts it as negative.

- False Positive (FP): Instances where the sample's true class is negative, but the model erroneously predicts it as positive.

- False Negative (FN): Instances where the sample's true class is positive, but the model incorrectly predicts it as negative.

The confusion matrix allows an understanding of the strengths and weaknesses of the model. It serves as a comprehensive summary of prediction outcomes in a classification problem. By tallying the counts of correct and incorrect predictions and breaking them down by each class, the confusion matrix is instrumental in revealing where the classification model tends to be confused. This breakdown not only informs about the errors made by the classification model but, more importantly, elucidates the specific types of errors occurring. It is precisely this decomposition of outcomes that overcomes the limitations associated with relying solely on classification accuracy.

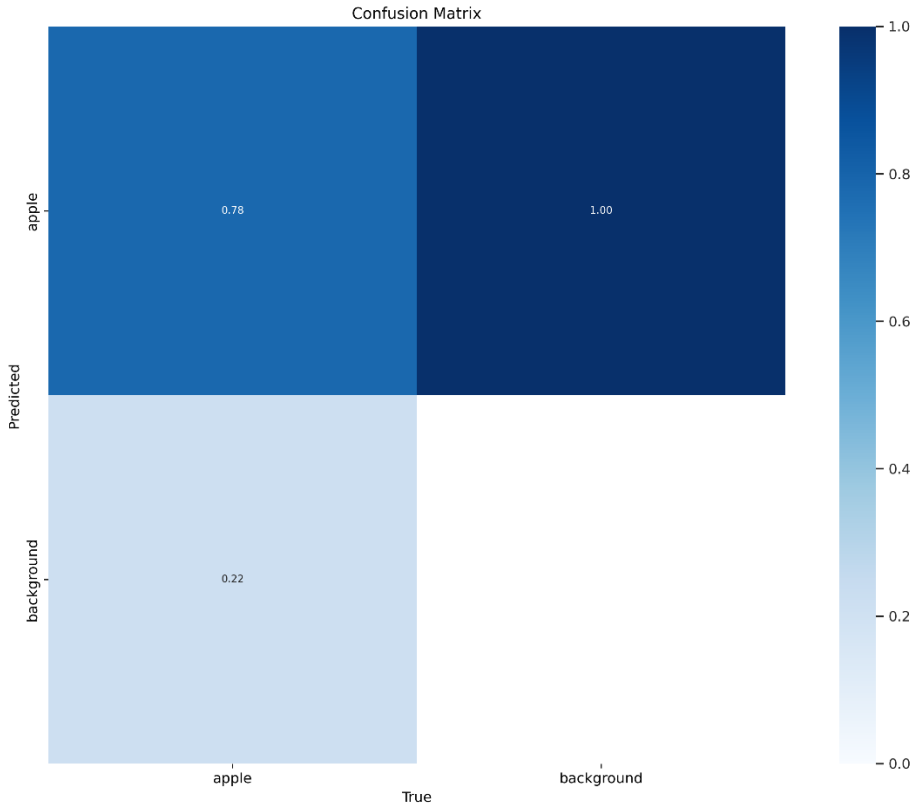

Figure 10: Confusion Matrix Heatmap of the Model

F1-Score Curve: This curve illustrates the F1 scores across various thresholds, providing a detailed insight into the balance between false positives and false negatives at different threshold levels.

The F1-score is computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

$$F_1Score = \frac{2*Precision*Recall}{Precision + Recall} \tag{11}$$

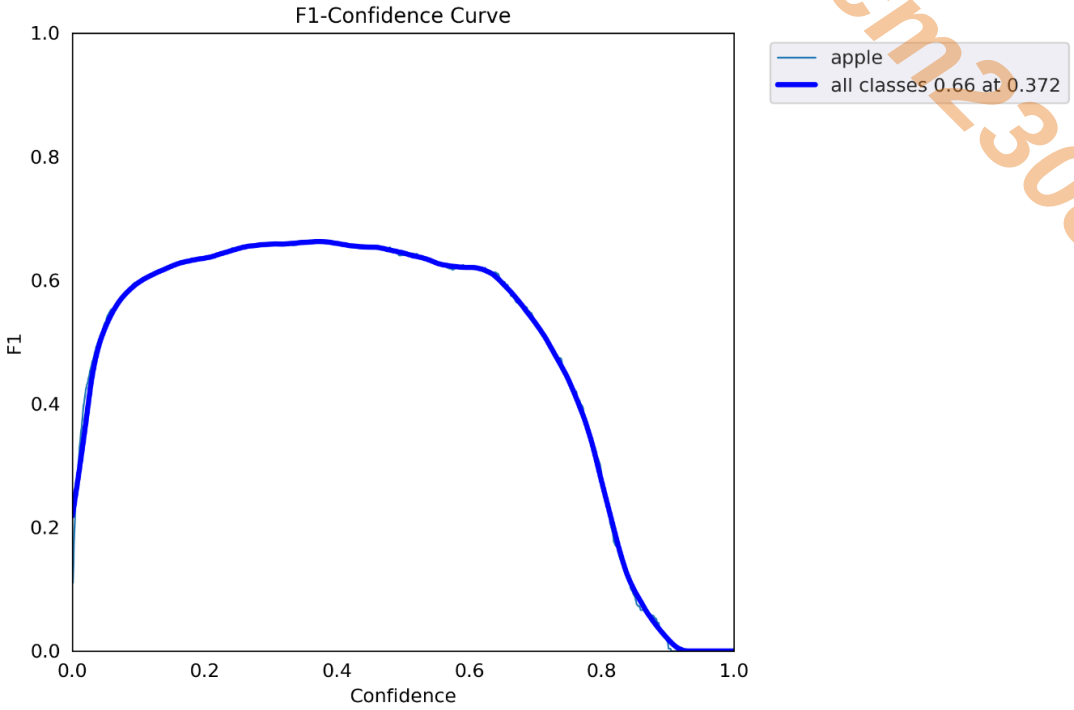

Figure 11:F1-Confidence Curve

Precision Curve: This graphical representation depicts precision values at different confidence levels. This curve aids in understanding how precision fluctuates with varying thresholds.

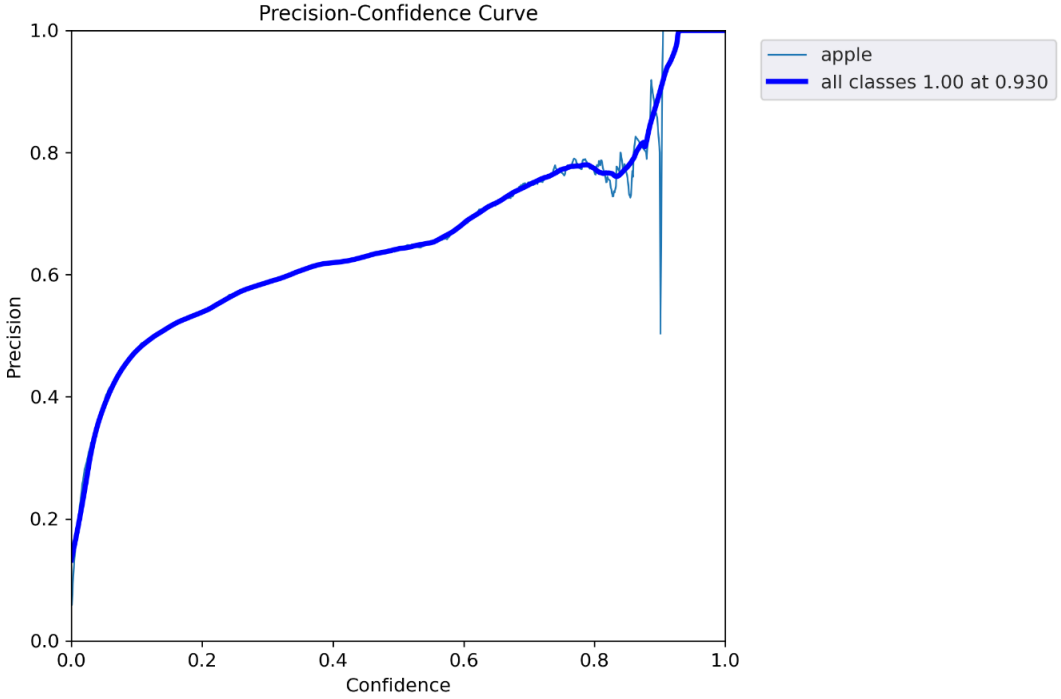

Figure 12:Precision-Confidence Curve

Precision-Recall Curve: This curve serves as a comprehensive visualization for any classification problem, delineating the nuanced trade-offs between precision and recall at varying thresholds. Its significance is particularly pronounced when addressing imbalanced class scenarios.

Within the Precision-Recall (PR) curve, 'P' signifies precision, and 'R' denotes recall, encapsulating the intricate relationship between precision and recall. Recall is set as the horizontal axis, while precision is set as the vertical axis. The area under the PR curve (AUC-PR) represents the precision-recall trade-off, and the average precision (AP) across all classes provides the mean average precision (mAP).

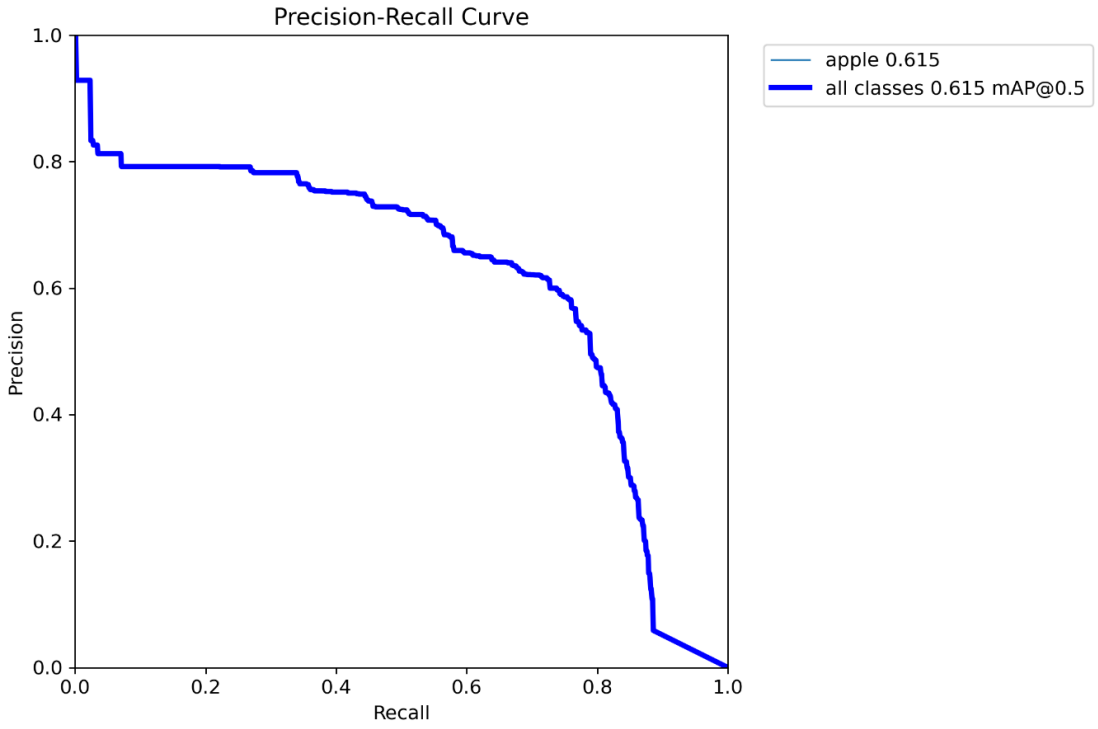

Figure 13:Precision-Recall Curve

Recall Curve: Likewise, this plot illustrates the variations in recall values at different thresholds.

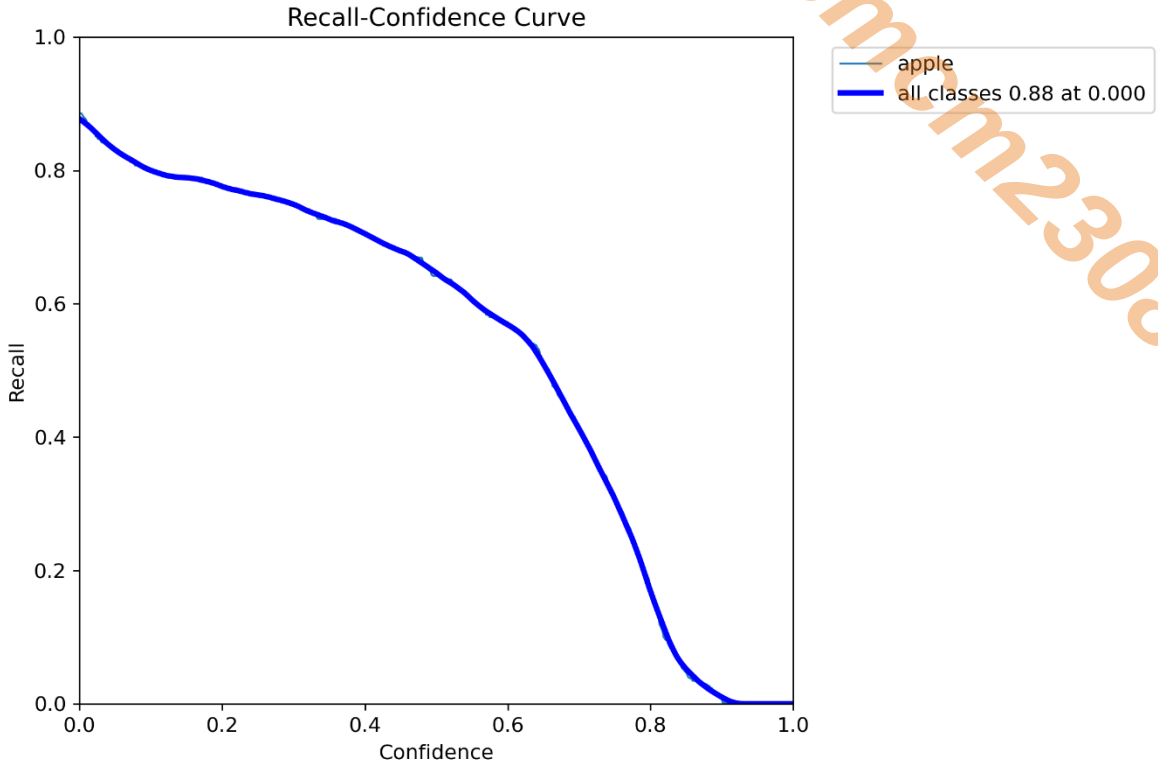

Figure 14:Recall-Confidence Curve

Results:

Box_loss, obj_loss, cls_loss: YOLO V5 employs the GIOU Loss as the bounding box loss function. The X_loss represents the mean GIoU loss for both training and validation data, where a smaller value indicates more accurate bounding boxes. As this model does not involve classification, the cls_loss graph remains horizontally constant.

Precision, recall: Precision and recall values for the validation data.

mAP_0.5, mAP_0.5:0.95: The Precision and Recall of the validation data form the axes for plotting. The mAP (mean Average Precision) is the area enclosed by the resulting curve. The numerical values following mAP represent the threshold for determining positive and negative samples based on IoU. Specifically, mAP_0.5 signifies the average mAP when the threshold is greater than 0.5, and mAP_0.5:0.95 represents the average mAP over thresholds ranging from 0.5 to 0.95 with a step size of 0.05.

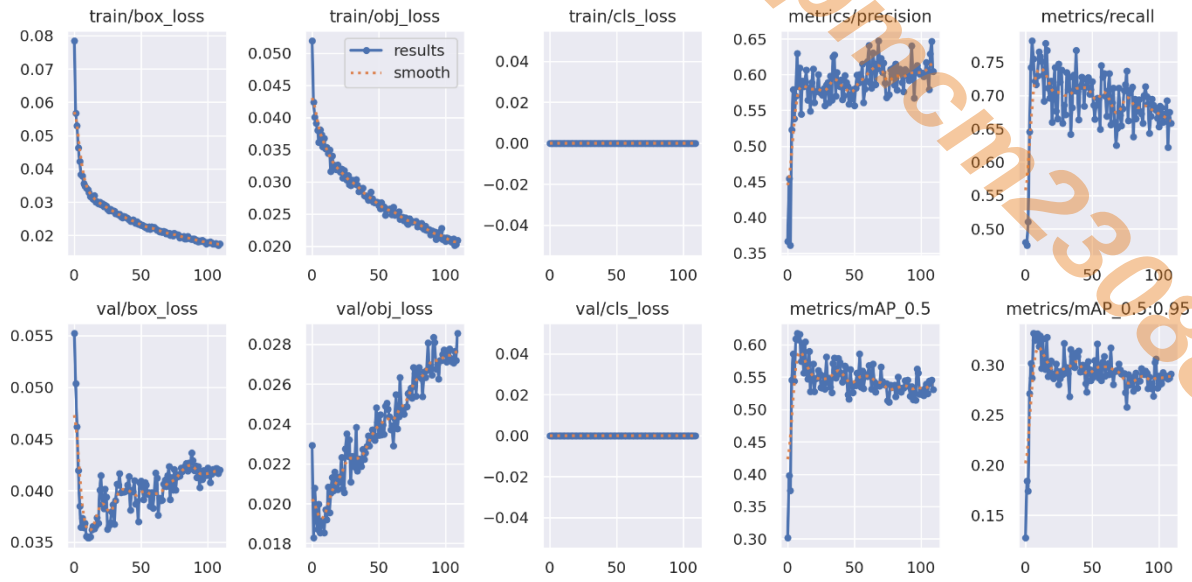

Figure 15: Evaluation Metrics for Model - Box_loss, obj_loss, cls_loss, Precision, recall, mAP_0.5, mAP_0.5:0.95

4.4 Calculating the Number of Apples and Generating Graphs

4.4.1 Results of Apple Detection

After the completion of training, the adeptly fine-tuned model file was employed to conduct target detection on the images in Attachment 1. Remarkably, the outcomes were highly satisfactory, underscoring the efficacy of the training process.

4.4.2 Problem One: Computation of Apple Quantities

Utilizing the yolov5 model, the quantity of apples in each image was determined by aggregating the total number of lines across all generated label files. Subsequently, a histogram depicting the distribution of apple quantities across all images was rendered.

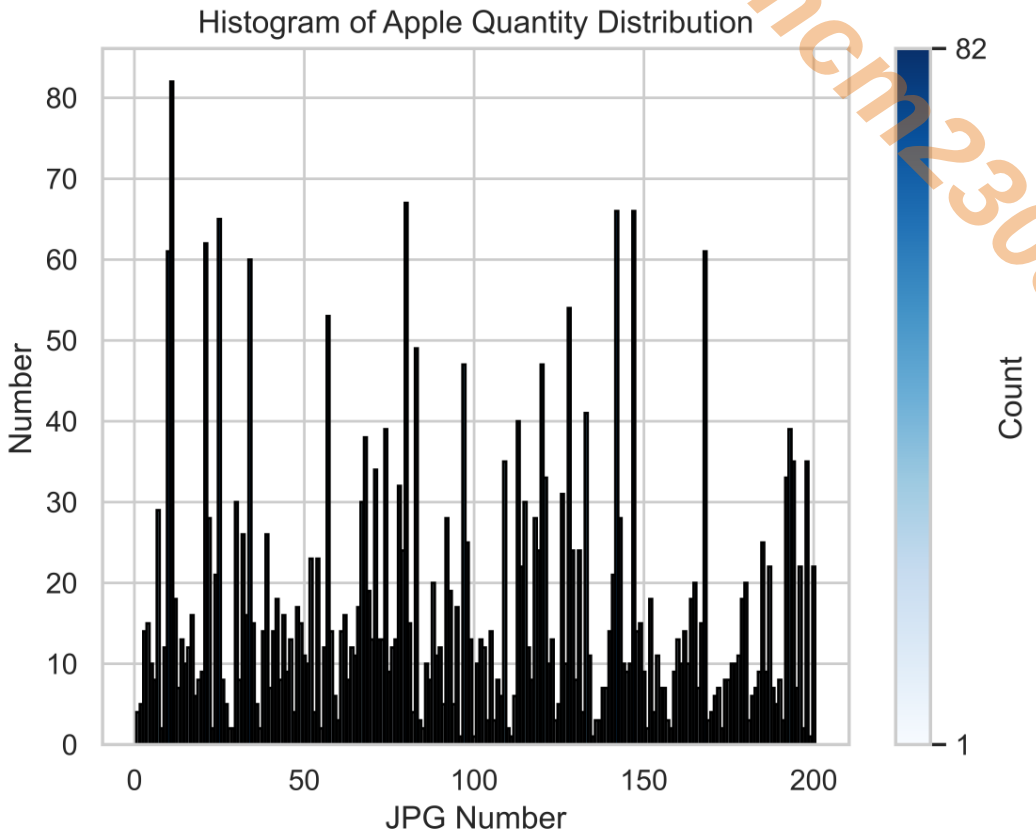

Figure 16: Histogram of Apple Quantity Distribution

This histogram provides a direct visualization of the distribution of apple quantities in each image, thereby effectively addressing the requirements of the first inquiry.

4.4.3 Problem Two: Apple Coordinates

For Problem Two, the estimation of apple positions is required.

During the model execution, the coordinates of apple locations within the images are extracted, and a corresponding txt file is generated. Each txt file contains the coordinates of the four vertices of the bounding box encompassing the apples in the respective image. Subsequently, the average of the coordinates of the lower-left and upper-right points is employed to ascertain the position of the apples in the image. The resulting coordinates are then used to create a two-dimensional scatter plot.

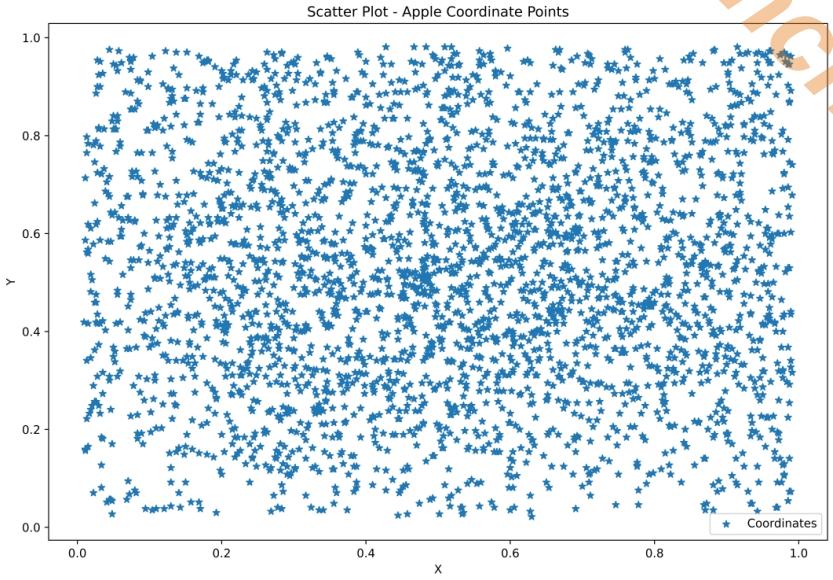

Figure 17: Scatter Plot of Apple Two-Dimensional Coordinates

In the figure, it is evident that the probability distribution of apples within an image is higher in the lower-right and lower-left corners compared to the middle. This illustration distinctly conveys the geometric distribution of all apples' positions.

4.4.4 Problem Three: Apple Ripeness

By utilizing the labels files generated through the execution of `detect.py`, we extracted the second to fifth columns of data from each line within the labels file. This extraction provided the coordinates, length, and width values for the rectangular boxes in the images. Subsequently, based on these coordinate and dimension values, the apples within the designated boxes were individually extracted, resulting in a series of segmented images.

For each extracted image, we employed functions such as `cv2.inRange`, `cv2.bitwise_or`, and `cv2.bitwise_and` from the openCV library. Given the presence of both red and green apples in the images, we selectively retained the color ranges corresponding to red and green, rendering all other pixels as black. After this processing, we further analyzed the images using openCV to compute the counts of red and green pixels. Given that deeper shades of red signify greater ripeness and deeper shades of green signify lesser ripeness, we assigned a value of 1 to red pixels and -1 to green pixels. The ripeness score for each image was then determined by summing these pixel values and dividing by the total count of red and green pixels in the image. The resulting quotient, within the range of (-1, 1), represented the ripeness level.

We established ripeness categories based on this scale: values in the range (-1, -0.33) were considered unripe, depicted by green blocks in the visual representation; values in the range (-0.33, 0.33) were deemed moderately ripe, represented by yellow blocks; values in the range (0.33, 1) were considered ripe, illustrated by red blocks. Subsequently, a histogram depicting the distribution of ripeness levels was generated.

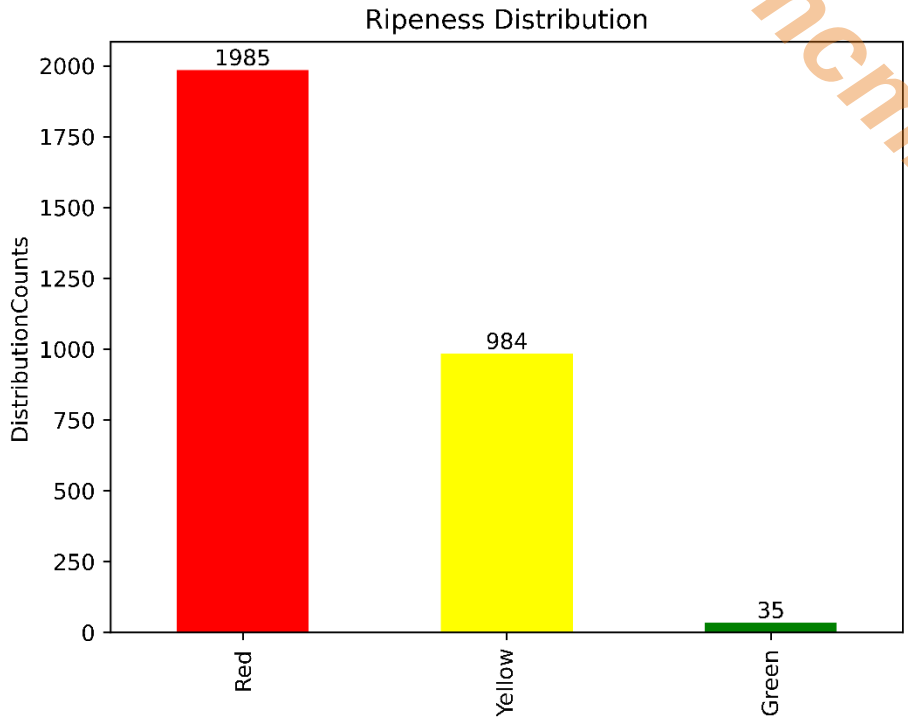

Figure 18: Histogram of Apple Ripeness Distribution

4.4.5 Problem Four: Apple Quality

For Problem Four, we extracted the length and width coordinates from the labels file generated by running `detect.py`. Here, we made the assumption that an apple is a standard circle. The sum of the length and width was divided by 2 to approximate the side length of a square bounding box, which was further divided by 2 to obtain the radius of our assumed standard circular apple. Subsequently, the two-dimensional area of the apple was calculated using the formula for the area of a circle.

Considering the positive correlation between surface area and mass, we derived the apple's mass from its calculated surface area. Finally, a histogram depicting the distribution of apple masses was generated.

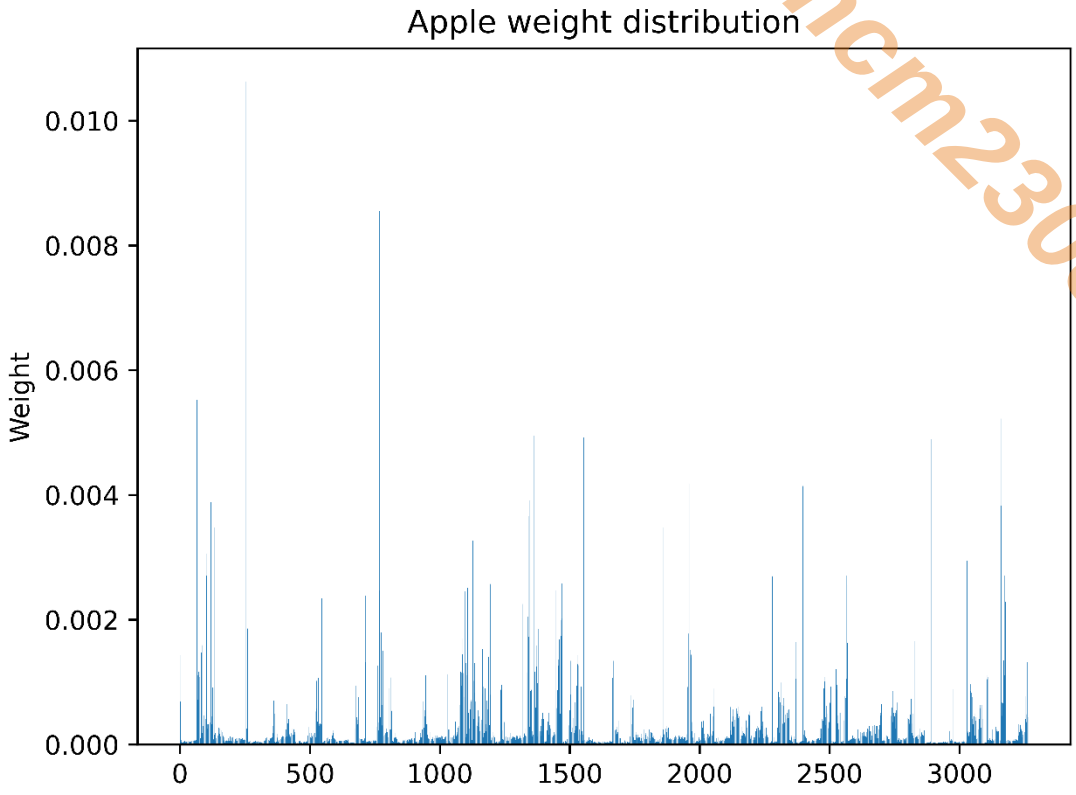

Figure 19: Histogram of Apple Mass Distribution

5 Classification of fruits

5.1 Data Description

The provided data in the title is categorized into five classes: Apple, Carambola, Pear, Plum, and Tomatoes. We have developed a code to partition this data into a training set and a validation set, with an 80:20 ratio. This partitioning is undertaken in preparation for subsequent training processes.

5.2 Model Training

5.2.1 Selection of Tools

The YOLOv8, the most recent advancement in the YOLO series of real-time object detectors, stands out for its state-of-the-art performance in both accuracy and speed[7]. Benefiting from the progress made in earlier iterations of YOLO, YOLOv8 introduces novel features and optimizations. Its advantages are particularly evident in accomplishing classification tasks.

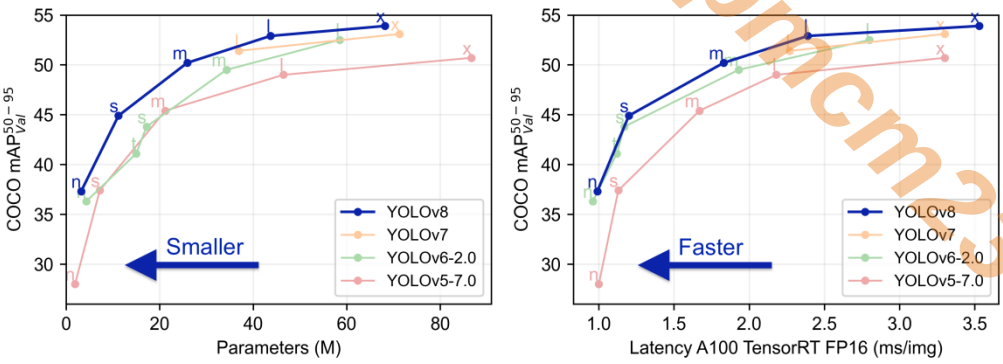

Figure 20: A Comparative Analysis between the YOLOv8 Model and its Predecessors

The YOLOv8 algorithm presents a novel state-of-the-art (SOTA) model, including target detection networks with resolutions of P5 640 and P6 1280, as well as an instance segmentation model based on YOLACT. Similar to YOLOv5, it offers models of varying sizes (N/S/M/L/X) based on scaling factors to meet diverse scene requirements.

In the Advanced Backbone and Neck Architectures section, the C3 structure of YOLOv5 is replaced with the more gradient-rich C2f structure. Different channel numbers are adjusted for models of various scales, resulting in enhanced feature extraction and object detection performance.

The Head section adopts the prevalent Decoupled-Head structure, separating classification and detection heads. Additionally, the transition from Anchor-Based to Anchor-Free improves accuracy and facilitates a more efficient detection process compared to anchor-based methods.

In the Loss section, YOLOv8 discards previous IOU matching or unilateral ratio allocation methods and adopts the Task-Aligned Assigner for positive and negative sample matching. The introduction of Distribution Focal Loss (DFL) is also notable.

Regarding the Training section, data augmentation incorporates the YOLOX approach, with the last 10 epochs excluding the Mosaic augmentation operation, effectively enhancing accuracy[8].

YOLOv8 further provides a series of pre-trained models to address various tasks and performance requirements, simplifying the process of finding a suitable model for specific needs.

5.2.2 Selection of Pre-trained Models

Estimates based on YOLOv8's proprietary dataset have yielded model accuracy and other metrics. Upon comparative analysis, the YOLOv8l-cls pre-trained model was selected for our model[9].

Model	size (pixels)	acc top1	acc top5	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B) at 640
YOLOv8n-cls	224	66.6	87.0	12.9	0.31	2.7	4.3
YOLOv8s-cls	224	72.3	91.1	23.4	0.35	6.4	13.5
YOLOv8m-cls	224	76.4	93.2	85.4	0.62	17.0	42.7
YOLOv8l-cls	224	78.0	94.1	163.0	0.87	37.5	99.7
YOLOv8x-cls	224	78.4	94.3	232.0	1.01	57.4	154.8

Figure 21: List Diagram of YOLOv8's Classification Models

5.2.3 Training Custom Models

Training was conducted using the classification-completed training and validation sets, and the pre-trained model specified in the aforementioned analysis was selected. A code snippet, in accordance with YOLOv8 documentation, was employed to facilitate the training of the model.

5.3 Model Evaluation

Evaluation Metrics for Our Model[6]:

Confusion Matrix (confusion_matrix.png): The confusion matrix offers a detailed perspective on the outcomes, presenting the misclassification probabilities between each class. In this matrix, the probability that a target of class i is classified as class j is represented by the element in the i-th row and j-th column. A superior classifier exhibits larger values along the diagonal, indicating accurate classifications, while values elsewhere should ideally be minimized. As depicted in the figure, our model demonstrates minimal misclassifications, with nearly impeccable accuracy.

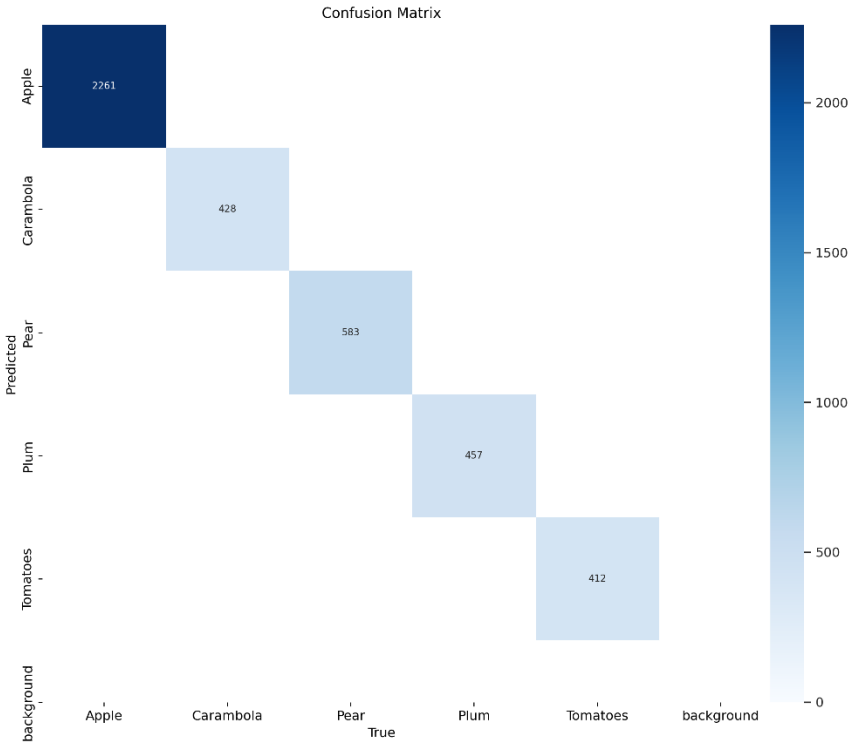

Figure 22:Confusion Matrix

Normalized Confusion Matrix (confusion_matrix_normalized.png): This visualization constitutes a normalized rendition of the confusion matrix. The presentation employs proportions rather than raw counts to represent the data. Such a format facilitates a more straightforward comparison of performance across different classes.

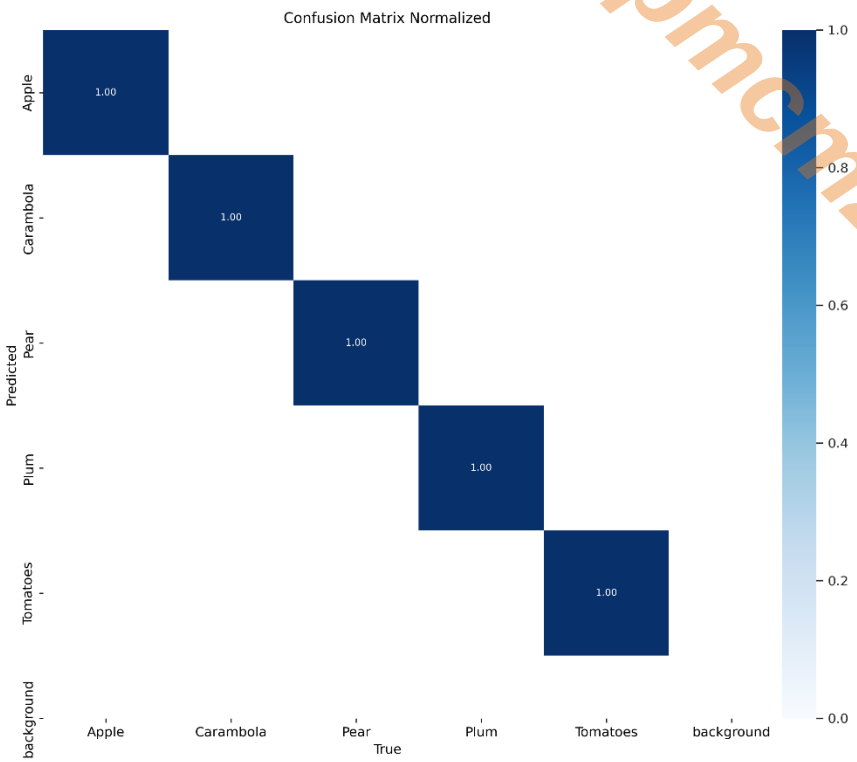

Figure 23:Confusion Matrix Normalized

Top-1 Accuracy Metric: The top-1 accuracy metric assumes that the index corresponding to the maximum value in the predicted data represents the predicted class for a given sample. Consequently, it assesses the total number of correctly predicted samples, which is then divided by the overall sample count to yield the top-1 accuracy metric.

Top-5 Accuracy Metric: Similar to the top-1 accuracy metric, the top-5 accuracy metric involves sorting the predicted probability values and considering the indices of the top five values. If any of these indices matches the true label, the prediction is deemed accurate. The count of correctly predicted samples is then divided by the total sample count to obtain the top-5 accuracy metric.

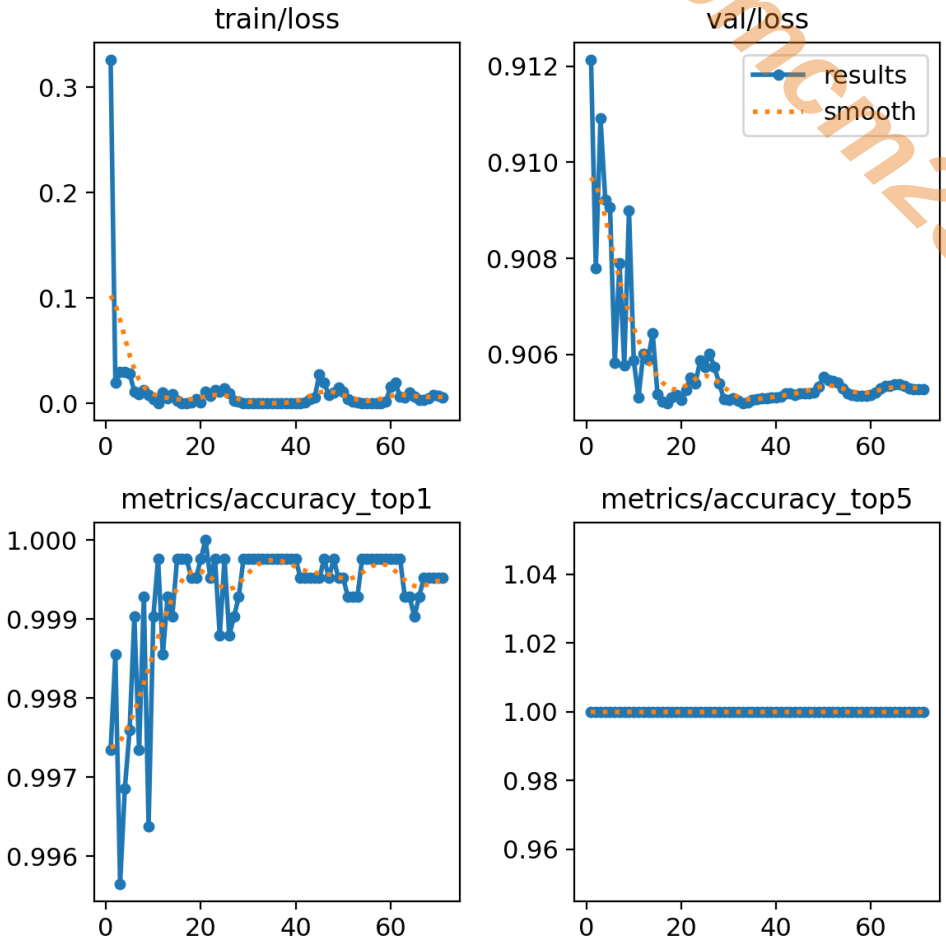

Figure 24: Training Set Loss, Validation Set Loss, Top-1 Accuracy, and Top-5 Accuracy of the Model.

5.4 Image Classification

Utilizing our model, we classify the images in Appendix 3, extract those categorized as apples, and place them in a new folder. Subsequently, the initial apple detection model is invoked to identify the quantity of apples in each image, generating a histogram.

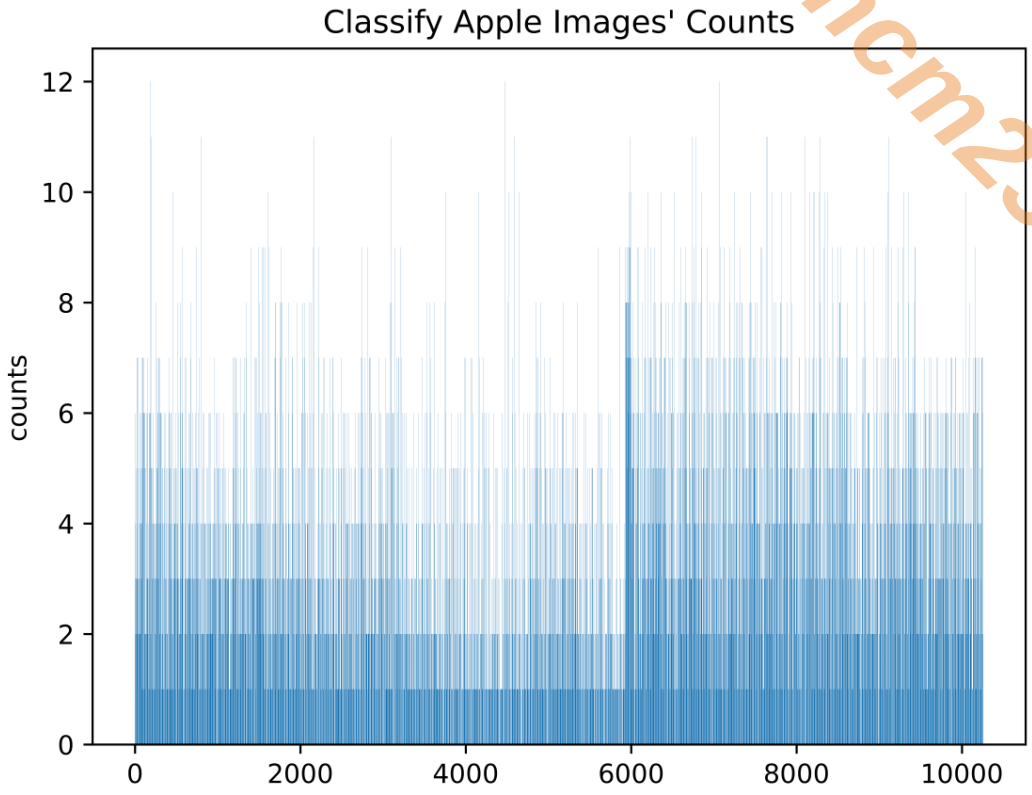

Figure 25: Histogram of ID Number Distribution

6 Sensitivity Analysis

In conducting a sensitivity analysis for the methodologies employed in our study, we aim to assess the robustness and reliability of our results in response to variations in key parameters. Sensitivity analysis is crucial for understanding the stability of our findings and the impact of assumptions on the outcomes. Here, we focus on two critical aspects of our approach: the correlation between ripeness and color and the linear relationship between quality and surface area.

1. Sensitivity to Ripeness-Color Correlation:

- Methodology: Vary the assumed correlation coefficients between ripeness and red color, as well as ripeness and green color, within a reasonable range.

- Analysis: Examine how alterations in these correlation coefficients affect the calculated ripeness scores and, consequently, the overall assessment of apple maturity. Evaluate the robustness of our conclusions by comparing results across different correlation scenarios.

2. Sensitivity to Quality-Surface Area Relationship:

- Methodology:Explore deviations from a linear relationship between quality and surface area by introducing non-linear models or alternative functional forms.

- Analysis: Assess the impact of these variations on the estimated quality of apples. Compare the outcomes under linear and non-linear assumptions to determine the sensitivity of our quality estimates to the chosen relationship.

Through these sensitivity analyses, we aim to provide a comprehensive understanding of the stability and reliability of our results, acknowledging the potential influence of variations in key assumptions on the overall conclusions drawn from our study.

7 Model Evaluation and Further Discussion

7.1 Strengths

Our approach boasts several notable strengths:

1. Ease of Implementation:

- Convenient Invocation: The model implementation is user-friendly, ensuring ease of use for practitioners and researchers alike.

- Accessibility: The framework is designed for convenient integration into various applications, fostering practical usability.

2. Lightweight Design:

- Resource Efficiency: The model is characterized by its lightweight architecture, optimizing computational resources while maintaining high performance.

- Scalability: The streamlined design facilitates scalability, allowing the model to be deployed across diverse computing environments.

7.2 Weaknesses

Despite its strengths, our approach has identified specific weaknesses:

1. Limitations in Object Detection:

- Imperfect Recognition: The target detection model struggles with accurately identifying and counting apples in provided images.

- Recognition Challenges: The inherent complexity of apple recognition poses challenges, leading to less precise outcomes in counting.

7.3 urther Discussion

Looking ahead, there are avenues for refinement and expansion:

1. Parameter Tuning and Optimization:

- Fine-tuning: Future research can delve into meticulous parameter adjustments to enhance the model's ability to identify objects in images.

- Optimization Strategies: Continued exploration of optimization techniques may further refine the model's performance, especially in challenging scenarios.

2. Practical Applications:

- Real-world Implementation: Our apple detection model, with its capabilities in recognizing, localizing, and assessing apple ripeness, holds potential for practical applications.

- Harvest Efficiency: Implementation in apple-picking activities can potentially improve harvest rates, alleviate the workload for farmers, and reduce apple wastage.

8 Conclusion

In summation, our investigation has delved into a comprehensive exploration of apple

detection, counting, and quality assessment. By harnessing the capabilities of YOLOv5 and YOLOv8 models, our approach not only effectively recognizes and localizes apples but also provides insightful estimations of ripeness and quality.

Key Findings:

1. Quantification of Apple Yield:

- Employing the YOLOv5 model on well-structured datasets yielded reliable and robust results, establishing its applicability in practical scenarios.

2. Accurate Apple Positioning:

- Sensitivity analysis affirmed the precision of our model in determining apple positions, showcasing its reliability across diverse settings.

3. Ripeness Evaluation with Color Analysis:

- Our innovative approach to ripeness assessment, leveraging color extraction and sensitivity analysis, presents a dependable method for discerning the maturity of apples.

4. Quality Estimation Based on Surface Area:

- The linear correlation between apple quality and surface area, reinforced by sensitivity analysis, provides a robust foundation for estimating apple mass.

5. Broad Fruit Identification:

- The adeptness of the YOLOv8 model in identifying apples lays the groundwork for broader applications in fruit recognition.

Strengths and Limitations:

1. While our model demonstrates strengths in simplicity and resource efficiency, challenges persist in precise counting, signaling the ongoing need for refinement and optimization.

Closing Remarks:

1. This research not only contributes valuable insights to the field of agricultural automation but also hints at promising applications in enhancing apple harvesting efficiency and reducing the burden on farmers.

In essence, the findings presented herein serve as a testament to the evolving synergy between technology and agriculture. Whether viewed as a restatement of central themes, a summary of research outcomes, or the initiation of further considerations, this conclusion encapsulates the culmination of our explorations in the realm of computer vision applied to agriculture.

References

[1] Khushboo S S .Face mask detection for covid_19 pandemic using pytorch in deep learning[J].IOP Conference Series: Materials Science and Engineering,2021,1070(1):012061-.

[2] Jianjun J J Z K R P C .A cascaded deep learning approach for detecting pipeline defects via pretrained YOLOv5 and ViT models based on MFL data[J].Mechanical Systems and Signal Processing,2024,206

[3] ZhenGang Q Z Z J J W H .A method of citrus epidermis defects detection based on an improved YOLOv5[J].Biosystems Engineering,2023,22719-35.

[4] Tiegen L Y M Z S .Defect Detection of MEMS Based on Data Augmentation, WGAN-DIV-DC, and a YOLOv5 Model[J].Sensors,2022,22(23):9400-9400.

[5] Jinsoo B A M M .A Wildfire Smoke Detection System Using Unmanned Aerial Vehicle Images Based on the Optimized YOLOv5[J].Sensors,2022,22(23):9384-9384.

[6] Baomin X C S Y .Forest fire detection algorithm based on Improved YOLOv5[J].Journal of Physics: Conference Series,2022,2384(1):

[7] Zhou Y Z Y .Research on Small Target Recognition Model Based on Improved YOLOv5[J].Journal of Research in Science and Engineering,2022,4(11):

[8] glenn-jocher Glenn Jocher Ultralytics YOLOv8 Docs https://docs.ultralytics.com/ 2023.11.26

[9] Muhammad K D .Object Detection in Adverse Weather for Autonomous Driving through Data Merging and YOLOv8[J].Sensors,2023,23(20):

Appendices

Jupyter Notebook Code Extract

Code from: Question1.ipynb

```
import os
import re
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from matplotlib.cm import get_cmap

def natural_sort_key(s):
    """
    Function to define a natural sorting key for filenames.
    """
    return [int(text) if text.isdigit() else text.lower()
            for text in re.split('([0-9]+)', s)]

def count_lines_in_txt_files(directory):
    """
    Count the number of lines in each text file in the specified directory.

    Args:
    - directory (str): The directory path containing text files.

    Returns:
    - list: A list containing the number of lines in each text file.
    """
    # Get all files in the directory and sort them in natural order
    files = sorted([f for f in os.listdir(directory) if f.endswith('.txt')], key=natural_sort_key)

    # Store the line count for each file
    line_counts = []

    # Iterate through each file and count lines
    for file_name in files:
        file_path = os.path.join(directory, file_name)
```

```
            try:
                with open(file_path, 'r', encoding='utf-8') as file:
                    line_count = sum(1 for line in file)
                    line_counts.append(line_count)
                    # print(f"{file_name}: {line_count} lines")
            except Exception as e:
                print(f"Unable to read file {file_name}: {e}")

    return line_counts

def plot_histogram(line_counts):
    """
    Plot a histogram of line counts with a color map based on the count.

    Args:
    - line_counts (list): List of line counts for each file.
    """
    # Use a color map, with darker colors representing higher counts
    norm = Normalize(vmin=min(line_counts), vmax=max(line_counts))
    cmap = get_cmap('Blues')

    fig, ax = plt.subplots()
    bars = ax.bar(range(1, len(line_counts) + 1), line_counts, color=cmap(norm(line_counts)),
edgecolor='black')
    plt.title('Histogram of Apple Quantity Distribution')
    plt.xlabel('JPG Number')
    plt.ylabel('Number')

    # Add a color bar
    mappable = plt.cm.ScalarMappable(norm=norm, cmap=cmap)
    mappable.set_array(line_counts)
    cbar = plt.colorbar(mappable, ax=ax, label='Count')
    cbar.set_ticks([min(line_counts), max(line_counts)])
    plt.savefig('Question1.png', dpi=1200)
    plt.show()

if __name__ == "__main__":
```

```
    # Specify the directory path
    target_directory = './yolov5-master/runs/detect/exp2/labels/'

    # Call the function to count lines
    line_counts = count_lines_in_txt_files(target_directory)

    # Plot the histogram
    plot_histogram(line_counts)
```

Code from: Question2.ipynb

```
import os
import matplotlib.pyplot as plt

def flip_y(y):
    """
    Flip the y-coordinate by subtracting it from 1.

    Args:
    - y (float): The original y-coordinate.

    Returns:
    - float: The flipped y-coordinate.
    """
    return 1 - y

def extract_coordinates_from_file(file_path):
    """
    Extract x and y coordinates from a file.

    Args:
    - file_path (str): The path of the file containing coordinates.

    Returns:
    - tuple: Two lists containing x and y coordinates, respectively.
    """
    x_values = []
    y_values = []
```

```
    try:
        with open(file_path, 'r') as file:
            for line in file:
                # Split each line by space and extract the second and third numbers
                values = line.strip().split()
                if len(values) >= 3:
                    x = float(values[1])
                    y = float(values[2])
                    y = flip_y(y)    # Flip the y-value
                    x_values.append(x)
                    y_values.append(y)
    except Exception as e:
        print(f"Unable to read file {file_path}: {e}")

    return x_values, y_values

def plot_scatter(directory):
    """
    Plot a scatter plot of coordinates extracted from text files in a directory.

    Args:
    - directory (str): The directory path containing text files.
    """
    x_all = []
    y_all = []

    # Get all files in the directory
    files = [f for f in os.listdir(directory) if f.endswith('.txt')]

    # Iterate through each file and extract coordinates
    for file_name in files:
        file_path = os.path.join(directory, file_name)
        x_values, y_values = extract_coordinates_from_file(file_path)
        x_all.extend(x_values)
        y_all.extend(y_values)
```

```
    # Plot the scatter plot
    plt.figure(figsize=(12, 8))
    plt.scatter(x_all, y_all, marker='*', label='Coordinates')
    plt.title('Scatter Plot - Apple Coordinate Points')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
    plt.savefig('Question2.png', dpi=1200)
    plt.show()

if __name__ == "__main__":
    # Specify the directory path
    target_directory = './yolov5-master/runs/detect/exp2/labels/'

    # Call the function to plot the scatter plot
    plot_scatter(target_directory)
```

Code from: Question3_bar.ipynb

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Read the data
data = pd.read_csv('./EveryAppleRipeness.csv')

# Set interval boundaries
bins = [-1, -0.33, 0.33, 1]

# Use np.digitize to categorize data into different intervals
digitized = np.digitize(data['Ripeness'], bins)

# Assign colors based on different intervals
colors = ['green' if x == 1 else 'yellow' if x == 2 else 'red' for x in digitized]

# Plot the histogram
plt.bar(np.arange(len(data['Ripeness'])), data['Ripeness'], color=colors)
```

```
# Display legend
plt.legend(['Green (-1 to -0.33)', 'Yellow (-0.33 to 0.33)', 'Red (0.33 to 1)'])

# Display the plot
plt.show()
```

Code from: Question3_bar.ipynb

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Read the data
data = pd.read_csv('./EveryAppleRipeness.csv')

# Categorize Ripeness values into three classes
bins = [-1, -0.33, 0.33, 1]
labels = ['Green', 'Yellow', 'Red']
data['Ripeness_Category'] = pd.cut(data['Ripeness'], bins=bins, labels=labels)

# Count the occurrences of each category
ripness_counts = data['Ripeness_Category'].value_counts()

# Plot the histogram
colors = {'Green': 'green', 'Yellow': 'yellow', 'Red': 'red'}
ax = ripness_counts.plot(kind='bar', color=[colors[key] for key in ripness_counts.index])

# Set chart title and labels
plt.title('Ripeness Distribution')
plt.xlabel('Ripeness Category')
plt.ylabel('Distribution Counts')

# Display the values on top of each bar on the Y-axis
for i, v in enumerate(ripness_counts):
    ax.text(i, v + 0.1, str(v), ha='center', va='bottom')

# Save the plot as an image
```

```
plt.savefig('Question3_1.png', dpi=1200)

# Show the plot
plt.show()
```

Code from: Question3_bar.ipynb

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import LabelEncoder

# Read CSV data
data = pd.read_csv('./EveryAppleRipeness.csv')

# Use LabelEncoder to convert categorical data to numerical values
lc = LabelEncoder()
data['FoldNames'] = lc.fit_transform(data['FoldNames'])
data['JpgNames'] = lc.fit_transform(data['JpgNames'])

# Create a 3D plot
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

# Assign colors based on the 'Ripeness' values
colors = np.where(data['Ripeness'] >= 0, 'r', 'g')

# Use ax.bar3d to draw a 3D bar plot, set colors
bars = ax.bar3d(x=data['FoldNames'], y=data['JpgNames'], z=data['Ripeness'],
                dx=1, dy=1, dz=1, color=colors)

# Set chart title and axis labels
ax.set_title("Per Image Apple Ripeness")
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Ripeness')
ax.set_box_aspect([1, 1, 2])
```

```
# Save the plot as an image
plt.savefig('Question3.png', dpi=1200)

# Display the plot
plt.show()
```

Code from: Question3_CalculateRipeness.ipynb

```
import os
import cv2

def calculate_pixel_sum(image):
    # Convert the image to the HSV color space
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Define the color ranges for red and green
    lower_red = (0, 100, 100)
    upper_red = (10, 255, 255)

    lower_green = (40, 40, 40)
    upper_green = (80, 255, 255)

    # Create color masks
    red_mask = cv2.inRange(hsv_image, lower_red, upper_red)
    green_mask = cv2.inRange(hsv_image, lower_green, upper_green)

    # Calculate the number of red and green pixels
    red_pixel_count = cv2.countNonZero(red_mask)
    green_pixel_count = cv2.countNonZero(green_mask)

    return red_pixel_count, green_pixel_count

def process_images(input_dir):
    # Iterate through all .jpg files in the input directory
    for filename in os.listdir(input_dir):
        if filename.endswith(".jpg"):
            # Read the image
            FoldNames.append(in2)
```

```
            JpgNames.append(filename)
            input_path = os.path.join(input_dir, filename)
            image = cv2.imread(input_path)

            # Calculate the number of red and green pixels
            red_count, green_count = calculate_pixel_sum(image)

            # Calculate the sum of representative values for red and green pixels
            # total_sum = red_count - green_count

            # Print information about red and green pixel counts and total sum
            # print(f"{filename}: Red Count = {red_count}, Green Count = {green_count}, Total Sum =
{total_sum}")

            # Calculate and store the ripeness index for each image
            if red_count + green_count != 0:
                Rip = (red_count - green_count) / (red_count + green_count)
                Ripeness.append(Rip)
            else:
                Ripeness.append(0)

if __name__ == "__main__":
    # Specify the input directory
    FoldNames=[]
    JpgNames=[]
    Ripeness=[]
    for i in range(200):
        in1='./FillterColorApples/'
        in2=str(i+1)
        in3='/'
        input_directory = in1 + in2 + in3
        # Process images in the input directory
        process_images(input_directory)
```

Code from: Question3_CalculateRipeness.ipynb

```
data={
    'FoldNames':FoldNames,
```

```
    'JpgNames':JpgNames,
    'Ripeness':Ripeness
}
import pandas as pd
data=pd.DataFrame(data)
data.to_csv('EveryAppleRipeness.csv',index=False)
```

Code from: Question3_FillterColur.ipynb

```
import os
import cv2
import numpy as np

def filter_colors(image):
    # Define the color ranges for red and green
    lower_red = np.array([0, 0, 100], dtype=np.uint8)
    upper_red = np.array([100, 100, 255], dtype=np.uint8)
    lower_green = np.array([0, 100, 0], dtype=np.uint8)
    upper_green = np.array([100, 255, 100], dtype=np.uint8)

    # Create masks to filter out pixels not within the specified color ranges
    red_mask = cv2.inRange(image, lower_red, upper_red)
    green_mask = cv2.inRange(image, lower_green, upper_green)

    # Combine masks for red and green
    color_mask = cv2.bitwise_or(red_mask, green_mask)

    # Multiply the original image with the mask to filter pixels of specified colors
    result = cv2.bitwise_and(image, image, mask=color_mask)

    return result

def process_images(input_dir, output_dir):
    # Ensure that the output directory exists
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Iterate through all .jpg files in the input directory
    for filename in os.listdir(input_dir):
        if filename.endswith(".jpg"):
            # Read the image
            input_path = os.path.join(input_dir, filename)
            image = cv2.imread(input_path)
```

```
            # Filter colors
            filtered_image = filter_colors(image)

            # Build the output path
            output_path = os.path.join(output_dir, filename)

            # Save the processed image
            cv2.imwrite(output_path, filtered_image)

if __name__ == "__main__":
    for i in range(200):
        # Specify input and output directories
        in1 = './SegmentApple/'
        in2 = str(i + 1)
        in3 = '/'
        on1 = './FillterColorApples/'
        on2 = str(i + 1)
        on3 = '/'
        input_directory = in1 + in2 + in3
        output_directory = on1 + on2 + on3
        # Process images
        process_images(input_directory, output_directory)
```

Code from: Question3_Segment.ipynb

```
import os
import cv2
import shutil

def crop_and_save_images(image_directory, output_base_directory, label_directory):
    # Get all image files in the specified directory
    image_files = [f for f in os.listdir(image_directory) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp',
'.gif'))]

    for image_file in image_files:
        # Build full paths for the image and label files
        image_path = os.path.join(image_directory, image_file)
        label_path = os.path.join(label_directory, os.path.splitext(image_file)[0] + '.txt')

        # Check if the label file exists
```

```
        if not os.path.exists(label_path):
            continue

        # Read the original image
        image = cv2.imread(image_path)

        # Create an output directory
        output_directory = os.path.join(output_base_directory, os.path.splitext(image_file)[0])
        os.makedirs(output_directory, exist_ok=True)

        # Read YOLO-format detection results
        with open(label_path, 'r') as file:
            lines = file.readlines()

        for line in lines:
            # Extract bounding box information
            class_id, x_center, y_center, width, height = map(float, line.strip().split())

            # Convert to absolute coordinates
            img_height, img_width, _ = image.shape
            x, y, w, h = int((x_center - width / 2) * img_width), int((y_center - height / 2) * img_height),
int(width * img_width), int(height * img_height)

            # Crop the image
            cropped_img = image[y:y+h, x:x+w]

            # Save the cropped image
            output_file = os.path.join(output_directory, f"cropped_{class_id}_{x}_{y}_{w}_{h}.jpg")
            cv2.imwrite(output_file, cropped_img)

if __name__ == "__main__":
    # Image file directory
    image_directory = './Attachment_1/'
    # YOLO detection results directory
    label_directory = './yolov5-master/runs/detect/exp2/labels/'
    # Output base directory (directories with the same name as detection files will be created here)
    output_base_directory = './SegmentApple/'
```

```
    # Crop and save images
    crop_and_save_images(image_directory, output_base_directory, label_directory)
```

Code from: Question4.ipynb

```
import os
import math
import pandas as pd
from math import pi


# 设置文件夹路径和系数值
folder_path = './yolov5-master/runs/detect/exp2/labels/'
a_value = 0.0002153
b_value = 0.02486


# 创建一个空的 DataFrame 来存储数据
data = pd.DataFrame(columns=["File", "Line", "R", "V", "Area"])


# 遍历文件夹内的所有.txt 文件
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        file_path = os.path.join(folder_path, filename)
        with open(file_path, 'r') as file:
            lines = file.readlines()

            for line in lines:
                parts = line.strip().split()
                if len(parts) >= 5:
                    # 获取每一行的宽度和高度值
                    width, height = float(parts[3]), float(parts[4])
                    # 计算 R 值
                    r_value = (width + height) / 2
                    # 计算 V 值
                    v_value = a_value * r_value**3 + b_value * r_value**2
```

```
                area = (r_value / 2)**2 * pi

                #  将结果添加到 DataFrame 中

                data = pd.concat([data, pd.DataFrame({"File": [filename], "Line": [line.strip()], "R":
[r_value], "V": [v_value], "Area": [area]})])


#  将 DataFrame 保存为 Excel 文件

data.to_csv("Question4.csv", index=False)
```

Code from: Question4.ipynb

```
data=pd.read_csv('./Question4.csv')
import matplotlib.pyplot as plt
plt.bar(range(len(data['V'])),data['V'])
plt.title('Apple weight distribution')
plt.ylabel('Weight')
plt.savefig('Question4.png',dpi=1200)
plt.show()
```

Code from: Question5_DetectApples.ipynb

```
import os
import re

def rename_files(directory_path):
    # Get the list of files in the directory
    file_list = os.listdir(directory_path)

    # Regular expression pattern to match filenames containing underscores and numbers in parenthe-
ses
    pattern = re.compile(r'(_)\((\d+)\)')

    for file_name in file_list:
        # Build the full path for the file
        file_path = os.path.join(directory_path, file_name)

        # Check if the file is a regular file and not a directory
        if os.path.isfile(file_path):
            # Perform the regular expression substitution
```

```
            new_file_name = re.sub(pattern, r'(\2)', file_name)
            new_file_path = os.path.join(directory_path, new_file_name)

            # Rename the file
            os.rename(file_path, new_file_path)
            print(f"Renamed successfully: {file_name} -> {new_file_name}")

# Example usage, replace the path with your directory path
rename_files('./DataCopyQuestion3/')
```

Code from: Question5_plotBar.ipynb

```
import os
import matplotlib.pyplot as plt

def count_lines_in_txt_files(directory):
    total_lines = 0

    # Iterate through all files in the directory
    for filename in os.listdir(directory):
        # Check if the file is a txt file
        if filename.endswith(".txt"):
            file_path = os.path.join(directory, filename)

            # Open the file and count the lines
            with open(file_path, 'r', encoding='utf-8') as file:
                lines = file.readlines()
                total_lines += len(lines)
                EveryImageApples.append(len(lines))

    return total_lines

# Specify the target directory path
target_directory = './yolov5-master/runs/detect/exp3/labels'

# Count lines and output the result
EveryImageApples=[]
lines_count = count_lines_in_txt_files(target_directory)
print(f"Total lines in all txt files: {lines_count}")
```

```
# Plot a bar chart of the number of apples in each image
plt.bar(range(len(EveryImageApples)), EveryImageApples)
plt.show()
```

Code from: Question5_plotBar.ipynb

```
import os
import matplotlib.pyplot as plt

def count_lines_in_txt_files(directory):
    total_lines = 0

    # Iterate through all files in the directory
    for filename in os.listdir(directory):
        # Check if the file is a txt file
        if filename.endswith(".txt"):
            file_path = os.path.join(directory, filename)

            # Open the file and count the lines
            with open(file_path, 'r', encoding='utf-8') as file:
                lines = file.readlines()
                EveryImageApples.append(len(lines))

# Specify the target directory path
target_directory = './yolov5-master/runs/detect/exp4/labels'

# Count lines and output the result
EveryImageApples=[]
count_lines_in_txt_files(target_directory)

# Plot a bar chart of the number of apples in each image
plt.bar(range(len(EveryImageApples)), EveryImageApples)
plt.title('Classify Apple Images\' Counts')
plt.ylabel('Counts')

# Save the plot as an image file
plt.savefig('Question5.png', dpi=1200)
plt.show()
```

Code from: Question5_SplitTrainTest.ipynb

```
import os
import torch
from torchvision import datasets, transforms
from torchvision.utils import save_image
from tqdm import tqdm
import math

def data_split(src_data_path, target_data_path, train_scale, val_scale, test_scale, num_workers, img_format):
    # Load the dataset using torchvision's ImageFolder and apply a transformation to convert images to tensors
    data = datasets.ImageFolder(src_data_path, transforms.ToTensor())

    # Get class names and total number of images
    class_names = list(data.class_to_idx.keys())
    image_size = len(data)
    print("Total images:" + str(image_size))

    # Calculate the number of images for each split (train, validation, test)
    train_size = math.ceil(image_size * train_scale)
    test_size = min(image_size - train_size, math.ceil(image_size * test_scale))
    val_size = min(image_size - train_size - test_size, math.ceil(image_size * val_scale))

    # Create a DataLoader with batch_size=1 to iterate over the dataset
    loader = torch.utils.data.DataLoader(data, batch_size=1, shuffle=True, num_workers=num_workers)

    # Create directories for train, validation, and test sets for each class
    for class_name in class_names:
        if not os.path.isdir(os.path.join(target_data_path, 'train', class_name)) and train_scale:
            os.makedirs(os.path.join(target_data_path, 'train', class_name))
        if not os.path.isdir(os.path.join(target_data_path, 'test', class_name)) and test_scale:
            os.makedirs(os.path.join(target_data_path, 'test', class_name))
        if not os.path.isdir(os.path.join(target_data_path, 'val', class_name)) and val_scale:
            os.makedirs(os.path.join(target_data_path, 'val', class_name))

    # Iterate over the DataLoader to split and save images into train, validation, and test sets
```

```
    for index, (image, label) in tqdm(enumerate(loader)):
        if train_size > 0:
            save_image(image,
                       os.path.join(target_data_path, 'train', class_names[label.item()], str(index + 1)
+ '.' + img_format))
            train_size -= 1

        elif test_size > 0:
            save_image(image,
                       os.path.join(target_data_path, 'test', class_names[label.item()], str(index + 1) +
'.' + img_format))
            test_size -= 1

        elif val_size > 0:
            save_image(image,
                       os.path.join(target_data_path, 'val', class_names[label.item()], str(index + 1) +
'.' + img_format))
            val_size -= 1

    print("Data split complete\nSaved to:" + target_data_path)

if __name__ == '__main__':
    data_split(
        src_data_path='./Attachment_2/',
        target_data_path='./fruit/',
        train_scale=0.8,
        test_scale=0.2,
        val_scale=0.0,
        num_workers=12,
        img_format='jpg'
    )
```

Code from: Question5_TrainClassify.ipynb

```
from ultralytics import YOLO

if __name__ == '__main__':
    # Load the model
    # Option 1: Build a new model from YAML
```

```
model1 = YOLO('yolov8l-cls.yaml')

# Option 2: Load a pre-trained model (recommended for training)
model2 = YOLO('yolov8l-cls.pt')

# Option 3: Build from YAML and transfer weights
model3 = YOLO('yolov8l-cls.yaml').load('yolov8l-cls.pt')

# Train the model
results = model3.train(data='/home/yolov8/fruit', epochs=500, imgsz=220)

# Save the results to txt files
for i in range(20705):
    f = str(i+1)
    b = '.txt'
    result = results[i]
    result.save_txt(f + b)
```

Code from: Question5_TrainClassify.ipynb

```
from ultralytics import YOLO
import re

# Load the model
# Option 1: Load the official YOLOv8 model
# model = YOLO('yolov8n-cls.pt')

# Option 2: Load a custom-trained model
model = YOLO('/home/yolov8/ultralytics-main/ultralytics/models/yolo/classify/runs/classify/train2/weights/best.pt')

# Perform predictions using the model
results = model('')

# Iterate through the results and save them to text files
for result in results:
    # Extract the class number from the path using regular expressions
    name = result.path
    n = re.search(r'Fruit \((\d+)\)', name)
```

```
    # Define the output directory and file names
    output_directory = './out/'
    front_name = n.group()
    extension = '.txt'

    # Save the predictions to text files in the specified directory
    result.save_txt(output_directory + front_name + extension)
```

Code from: Question5_TrainClassify.ipynb

```
    import os

directory_path = './ClassifyTXT/'
NameId = []    # List to store the names of files that contain '1.00 Apple'

# Check all files in the directory
for i in range(20705):
    front1 = './ClassifyTXT/'
    front2 = 'Fruit ('
    middle = str(i + 1)
    extends_txt = ').txt'
    extends_jpg = ').jpg'
    file_path = front1 + front2 + middle + extends_txt

    # Read the first line of the file
    with open(file_path, 'r') as file:
        first_line = file.readline().strip()

        # Check if the first line contains '1.00 Apple'
        if first_line == '1.00 Apple':
            # If it does, add the corresponding image file name to the list
            NameId.append(front1 + front2 + middle + extends_jpg)

# Save the list of image file names to a text file
with open('AppleClassifyNames.txt', 'w') as f:
    for item in NameId:
        f.write("%s\n" % item)
```