ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/387818793

Deep learning-based real-time detection and classification of tomato ripeness stages using YOLOv8 on raspberry Pi

Article *in* Engineering Research Express · January 2025

DOI: 10.1088/2631-8695/ada720

CITATIONS
2

READS
410

10 authors, including:

Md. Nahiduzzaman
Rajshahi University of Engineering and Technology
57 PUBLICATIONS **1,412** CITATIONS
SEE PROFILE

Rusab Sarmun
University of Dhaka
17 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Amith Khandakar
Qatar University
237 PUBLICATIONS **9,888** CITATIONS
SEE PROFILE

Md. Ahasan Atick Faisal
Qatar University
26 PUBLICATIONS **340** CITATIONS
SEE PROFILE

All content following this page was uploaded by Rusab Sarmun on 22 January 2025.

The user has requested enhancement of the downloaded file.

Engineering Research
Express

INCLUSIVE PUBLISHING
TRUSTED SCIENCE

PURPOSE-LED
PUBLISHING™

PAPER

Deep learning-based real-time detection and classification of tomato ripeness stages using YOLOv8 on raspberry Pi

To cite this article: Md Nahiduzzaman *et al* 2025 *Eng. Res. Express* **7** 015219

View the article online for updates and enhancements.

You may also like

- Biospeckle image processing algorithms for non-destructive differentiation between maturity and ripe stages of Indian climacteric fruits and evaluation of their ripening period
S Kumari and A K Nirala

- Strawberry ripeness calibrated 2D colour lookup table for field-deployable computer vision
Imran Mohamed, David Williams, Richard Stevens et al.

- Identification Of Ripe And Unripe Citrus Fruits Using Artificial Neural Network
Mrs Rex Fiona, Shreya Thomas, Isabel J Maria et al.

This content was downloaded from IP address 103.121.11.22 on 22/01/2025 at 16:19

Engineering Research Express

RECEIVED
29 October 2024

REVISED
18 December 2024

ACCEPTED FOR PUBLICATION
7 January 2025

PUBLISHED
16 January 2025

PAPER

Deep learning-based real-time detection and classification of tomato ripeness stages using YOLOv8 on raspberry Pi

Md Nahiduzzaman[1,2], Rusab Sarmun[2,3], Amith Khandakar[2], Md Ahsan Atick Faisal[2], Munshi Sajidul Islam[2], Mohammad Kaoser Alam[2], Tawsifur Rahman[2], Nasser Al-Emadi[2], M Murugappan[4,5,*] and Muhammad E H Chowdhury[2,*]

[1] Department of Electrical & Computer Engineering, Rajshahi University of Engineering & Technology, Rajshahi 6204, Bangladesh
[2] Department of Electrical Engineering, Qatar University, Doha 2713, Qatar
[3] Department of Electrical & Electronics Engineering, Dhaka University, Dhaka 1000, Bangladesh
[4] Intelligent Signal Processing (ISP) Research Lab, Department of Electronics and Communication Engineering, Kuwait College of Science and Technology, Block 4, Doha, 13133, Kuwait
[5] Department of Electronics and Communication Engineering, Vels Institute of Sciences, Technology, and Advanced Studies, Chennai, Tamil Nadu, India
[*] Authors to whom any correspondence should be addressed.

E-mail: m.murugappan@kcst.edu.kw and mchowdhury@qu.edu.qa

Keywords: tomato ripeness detection, YOLOv8, object detection, automated harvesting, embedded system

Abstract
The automated detection of tomato ripeness is critical in crop management and harvesting. In most earlier works, tomato image ripeness detection has been based upon a limited set of images and binary classification (ripe and unripe). This study uses the cutting-edge YOLOv8 object detection algorithm and a comprehensive dataset to propose an accurate real-time system for detecting and classifying tomato ripeness (multi-class). Based on two open-source datasets (Kaggle and Internet-sourced), we developed and tested the proposed system. In this method, a comprehensive tomato image dataset is curated, YOLOv8 models are built, seamlessly integrated into an embedded system (Raspberry Pi4), then evaluated and validated. The model shows exceptional performance in detecting three distinct classes of ripeness: unripe, partially ripe, and ripe. It surpasses existing state-of-the-art models in both accuracy and efficiency. Based on the Kaggle dataset, our model achieves an average precision at 50 of 0.808, with F1-scores of 0.80, 0.65, and 0.796 for unripe, partially ripe, and ripe classes, respectively. It achieves mAP at 50 of 0.725 and F1-scores of 0.747 (unripe), 0.652 (partially ripe), and 0.72 (ripe) for the corresponding classes of the Internet-sourced Dataset, exceeding current state-of-the-art models. Finally, the proposed tomato ripeness detection algorithm is implemented on the Raspberry Pi 4 system and exhibits notable performance. With the integration of YOLOv8 into an embedded system (Respbeery Pi4), it can be used to improve efficiency and reduce labor costs in tomato-picking robots, helping to revolutionize agricultural practices.

1. Introduction and related works

The tomato (Solanum Lycopersicon L.) is one of the world's most important cash crops and the second largest crop in the world [1]. There is an abundance of potassium, folate, fiber, and vitamins A, C, and K in tomatoes. Tomato consumption has also been linked to a reduced risk of certain cancers, cardiovascular disease, osteoporosis, and other conditions [2]. Furthermore, tomatoes are also deeply rooted in their native Latin America and many other regions around the world. Furthermore, they play an important role in classic Italian dishes. Therefore, understanding the tomato crop's maturity stage is imperative for several reasons. These include determining which tomatoes should be shipped first and stored for the longest period. In addition to this, tomatoes are a type of product whose color characteristics are heavily relied upon to recognize their maturity level. Due to the amount of labor involved, picking them by hand takes considerable time, effort, and

© 2025 IOP Publishing Ltd. All rights, including for text and data mining, AI training, and similar technologies, are reserved.

money. Due to the worldwide trend of an aging population, high labor costs and a shortage of available workers are major issues in today's economy. A person with formal tomato sorting training should be employed to perform this task. Individual differences in sorting abilities make this method unreliable. By developing tomato ripeness-based detection systems, smart farm machinery will be able to harvest tomatoes automatically, apply drugs in targeted ways, and many other tasks. Artificial Intelligence (AI) advances will also allow previously labor-intensive tasks to be automated, such as picking [3]. The accuracy with which a robot's vision system can locate objects directly relates to its ability to pick them up. In recent decades, non-destructive computer vision has been used in the food industry and precision agriculture for inspecting and grading produce including fruits and vegetables [4–6]. A tomato's color characteristics are strongly correlated with its maturity; therefore, computer vision may be able to assess tomato maturity by analyzing its color characteristics. Therefore, one of the most crucial aspects of automatic fruit harvesting is accurately detecting ripe fruits based on their color in their natural habitat.

The past few decades have seen a plethora of detection and classification methods being developed, but many of these methods need to be more precise and reliable [7–17]. Consequently, this poses a significant challenge to the widespread use of sorting robots in industry. Fruit detection algorithms are becoming more likely to be successful as Computer Vision, Machine Learning, and Deep Learning technologies improve. Liu *et al* developed a method for detecting tomatoes based on YOLOv3, which incorporates dense architecture but replaces a rectangular bounding box with a circular bounding box (CBbox) [18]. Meanwhile, the author developed a CBbox that outperformed the Intersection-over-Union (IoU) for Non-Maximum Suppression (NMS). There were 966 images containing 3,465 tomatoes collected by the authors. Their model was trained on 725 images (2,553 tomatoes) and tested on 241 images (912 tomatoes). In their study, they achieved a precision of 94.75%, a recall of 93.09%, an F1-score of 93.91%, and an Average Precision (AP) of 96.4%. Magalhães *et al* employed five deep learning models: SSD (Single-Shot MultiBox Detector)—MobileNetv2, SSD—InceptionV2, SSD—ResNet50, SSD—ResNet101, and YOLOv4 tiny architectures, to detect tomatoes in various conditions such as occluded, overlapped, and so on [19]. The researchers collected 297 images with three classes (unripe, reddish, and red tomatoes) from unripehouses. They then augmented the images to create 23,021 images. In terms of precision, recall, f1-score, and mean AP (mAP), SSD-MobileNetv2 produced the highest values of 84.37%, 54.4%, 66.15%, and 51.46%, respectively.

The YOLOv3, YOLOv4, YOLODenseNet, and YOLOMixNet models were also developed by Lawal to detect tomatoes in occlusion, clusters of tomatoes, illumination variation, and shading conditions [20]. The authors captured 1,698 images and performed data augmentation to create 5,147 images. Only ripe and unripe classes were examined, and YOLOMixNet had the best IoU, recall, and mAP, which were 77.7%, 91%, and 98.4%, respectively. By fusing residual neural networks from YOLOv4 with R-CSPDarknet53, Zheng *et al* developed a new backbone network to detect tomatoes in the natural environment [1]. In addition, Contextual Spatial Pyramid Pooling (C-SPP) has been proposed to enable feature information reuse and multi-scale fusion. A total of 1,698 mature and immature tomato images were captured for the training of their models. They obtained precision, recall, and mean accuracy of 87.6%, 88.79%, and 94.45%, respectively. Su *et al* proposed a different approach to detect four types of tomato maturity, namely SYOLOV3-MobileNetV1, based on depth-wise separable convolutions and squeeze and excite attention mechanisms [5]. Moreover, they used mosaic data augmentation, K-means clustering, and SE methods to improve the accuracy of their models. As part of the preprocessing, they used data augmentation to create 5,147 synthetic images to train their models, resulting in an optimistic F1 of 94.9% and a mAP of 97.5%.

A YOLO V5 framework was constructed by Egi *et al* using three separate networks: CSPDarknet as the backbone, PANet as the neck, and the YOLO Layer as the head for detecting tomatoes without considering any conditions [21]. In order to count tomatoes, the Deep-Sort algorithm was assisted by the Kalman filter. In this study, they considered three classes of images (flowers, unripe tomatoes, and red tomatoes) and achieved precision, recall, and mean average precision of 74.1%, 57%, and 63%, respectively. Additionally, Ge *et al* incorporated the Shufflenetv2 module as a backbone network to reuse and incorporate image features, the bi-directional feature pyramid network framework in the Neck section, and the YOLOv5-Deepsort framework for tomato detection [22]. In addition, they analyzed 3,818 images of tomatoes from a real-time video, and their Deepsort model achieved 98.4% recall and 97.58 mAP. In contrast, Wang *et al* applied four models, namely a faster RCNN, an SSD, a YOLOv3, and a YOLO-Dense, to detect anomalies in tomatoes while taking occlusion, shading, and other factors into consideration [23]. Based on 15,000 tomatoes, they found that YOLO-Dense had a mAP of 96.41%, while different methods of data augmentation were taken into account. Based on the Darknet-19 classification model and YOLOv2 network, Zhao *et al* developed a detection algorithm to identify healthy and diseased tomatoes from 225 images [3]. By using different pre-processing image enhancement techniques, we achieved accuracy, recall, and mAP of 96%, 97%, and 91%, respectively. In order to distinguish between the four stages of tomato ripeness, Nugroho *et al* proposed three models: the faster region-based convolutional neural network (faster R-CNN), the SSD, and the YOLO [24]. With an average accuracy of 99.955% and a mAP of 90.8%, they got the highest accuracy using R-CNN from 400 images collected from Kaggle.

In addition to YOLO-based detection, some researchers investigated other types of detection algorithms. A series of preprocessing steps were performed by Wan *et al* to separate the tomato from its background, including threshold segmentation, noise cancellation, and image contour extraction [2]. Based on the color analysis method developed for capturing feature hue values, the authors then used back propagation neural networks (BPNNs) to classify tomatoes. The researchers collected 150 tomato images with three classes (unripe, orange, and red), trained the model with 102 images, and then tested it with 48 images, and achieved 99.31% accuracy. Chen *et al* converted the images into grayscale images and then used the Otsu algorithm to remove the complex background from the images [4]. In the following step, binary images were optimized using morphological operations, and original contours were extracted using the Canny operator. Finally, a modified Hough transform was applied to the images in order to recognize the tomato and extract only its true contour. Additionally, they collected 89 tomatoes using an industrial camera under illumination variation and occlusion conditions and achieved 94.33%, 96.5%, and 97.97% IoU, precision, and recall, respectively. Liu *et al* developed a tomatoDet model where the authors used Deep Layer Aggregation-34 (DLA-34) as its foundation and added a Convolutional Block Attention Module (CBAM) to enhance the feature expression ability before utilizing a radius head to detect tomatoes in the same natural conditions as Chen *et al* [25]. As a result of training their models with 725 images and testing them with 241 images, they got accuracy rates of 95.7%, recall rates of 94.3%, and AP rates of 98.16 percent. The greedy non-maximum suppression method, the greedy non-maximum merging method, and the greedy non-maximum merging method are used as methods of post-processing by Tureková *et al* [26]. Then, a faster R-CNN model using a ResNet-50 backbone was used to detect tomatoes from 50 images with precision, recall, and AP of 82.12%, 84.08, and 88.4%, respectively.

Existing studies have mostly focused on categorizing ripe and unripe tomatoes as ripe and unripe, respectively. The binary approach ignores the multiclass classification problem, which involves identifying not only tomatoes that are ripe and unripe but also their flowers. The pivotal stage of partially ripe tomatoes has often been overlooked. This stage is crucial for the agricultural industry, especially for long-distance shipments, as partially ripe tomatoes continue to ripen during transport, reducing the risk of damage typically associated with shipping fully ripe tomatoes, which are susceptible to bruising and spoilage. A more accurate measurement of tomato ripeness has direct implications for sustainability. It is critical to harvest tomatoes at the right stage of ripeness to minimize waste due to spoilage during transport or storage. Consequently, waste is reduced, resources are conserved, and energy is saved by not growing and transporting these discarded tomatoes. Furthermore, precise harvesting can reduce the number of trips across fields, save fuel, and reduce unripehouse gas emissions.

Our study aims to fill this gap in the literature by creating a classification system that identifies unripe (unripe), partially ripe, and red (ripe) tomatoes. Various tomato images from multiple sources were gathered to ensure a balanced representation of ripeness stages. Every image was annotated and validated by subject-matter experts to ensure accuracy and reliability. The computational complexity and processing time of detection algorithms are noted challenges in existing studies. Because of this, embedded systems, such as the Raspberry Pi, cannot be implemented using such algorithms. We have adopted the YOLOv8 model, an advanced object detection algorithm recognized for its excellent performance, efficiency, and reduced computation time. In particular, our model was implemented on the Raspberry Pi, which demonstrates its real-world utility. With the YOLOv8 model integrated with Raspberry Pi, farmers have access to a tool that reduces their manual workload and minimizes errors in tomato harvesting processes, ensuring maximum yields and minimizing waste.

To summarize, our system provides a detailed, efficient, and actionable solution for tomato classification that overcomes previous research barriers. By combining the YOLOv8 model with Raspberry Pi, our approach has the potential to redefine agricultural practices, leading to greater productivity, decreased waste, and improved crop quality, which will contribute to a more sustainable tomato harvesting pipeline.

2. Materials and methods

In order to address the critical challenges associated with tomato ripeness detection and classification, we propose a novel approach leveraging the cutting-edge object detection algorithm YOLOv8, as shown in figure 1. This method is meticulously designed with four primary phases that ensure optimal performance and robustness:

1. Developing a comprehensive tomato image dataset from open-source datasets with a wide range of ripeness levels, lighting conditions, and tomato varieties for training and validating YOLOv8 models.

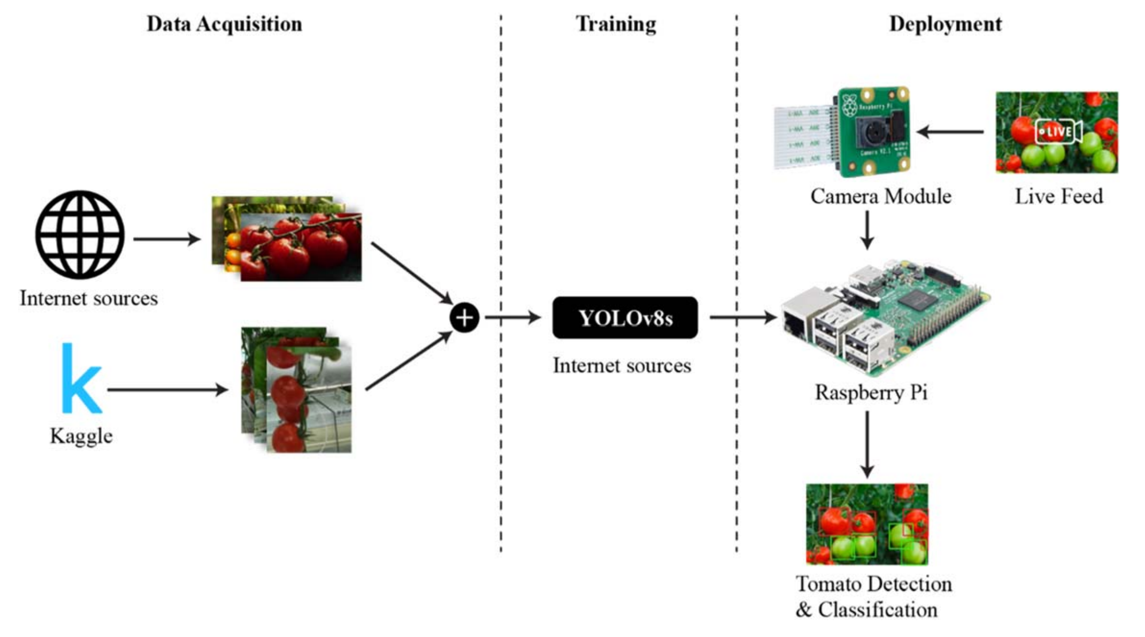


Figure 1. Proposed framework for tomato ripeness detection in a real-world environment.

2. Employing the YOLOv8 model to accurately detect and classify tomato ripeness stages with extraordinary recall and mean average precision (mAP) of 0.5 while minimizing computational overhead and processing latency. In real-world scenarios, this provides an efficient and effective method for detecting tomato ripeness.

3. The state-of-the-art model can be seamlessly integrated into a Raspberry Pi-based embedded system to enable the development of an automated tomato-picking system that dramatically reduces labor costs and damage associated with manual harvesting.

4. Demonstrate the accuracy, efficiency, and adaptability of the proposed framework in comparison with existing methods.

A framework like this could have wide-ranging implications for the agricultural sector, with immense potential. We hope to revolutionize the tomato harvesting pipeline by streamlining and automating the tomato ripeness detection process, resulting in more sustainable and cost-effective production practices. In the following section, we will provide a detailed exposition of the proposed framework. We will elucidate the methodology and techniques employed at each phase to achieve the framework's objectives. As a result of our in-depth exploration, the framework can be readily replicated and implemented in real-world applications, enhancing tomato ripeness detection and classification.

2.1. Data preparation and annotation

Data preparation plays an important role in detection algorithms since the quality and accuracy of input data directly influence the model's performance. A detection algorithm's effectiveness depends on its ability to understand complex patterns and distinctive characteristics that distinguish one class from another based on the training data [27]. In the presence of noise, incompleteness, inconsistent noise or outliers, the model will most likely produce erroneous or inconsequential patterns, leading to suboptimal generalization when applied to unknown data. It is critical to prepare a well-prepared dataset to ensure that the tomato ripeness detection model is robust and accurate. In this study, two different data sources were used to boost the generalization ability of the model. The first data source included 387 tomato images collected from various online resources, encompassing 6,369 instances. A variety of tomato types, stages of ripeness, lighting conditions, and background contexts were meticulously chosen for the images. Another source was the Kaggle dataset, which contained 359 images and 2,008 instances [28]. In the Kaggle dataset, only images were provided without any accompanying annotations.

We established a rigorous annotation process for tomato ripeness labels in order to ensure their accuracy and consistency. The images from both sources were first carefully annotated by an expert using 'labelImg' tool, who classified the tomatoes into unripe (unripe), partially ripe, and ripe. It has also been clarified that each image is paired with a corresponding .txt file that captures the bounding box information for each tomato ripeness category: unripe (unripe) labeled as '0', partially ripe labeled as '1', and red (ripe) labeled as '2'. We aim to provide

Table 1. Data splitting into training, testing, and validation set for both sources.

Data Sources	Class Types	Instances	Training	Testing	Validation
Internet	Unripe	2,426	1,402	668	356
	Partially Ripe	1,313	782	342	189
	Ripe	2,630	1,738	611	281
	Total	**6,369**	**3,922**	**1,621**	**826**
Kaggle	Unripe	380	286	55	39
	Partially Ripe	236	165	38	33
	Ripe	1,392	946	239	207
	Total	**2,008**	**1,397**	**332**	**279**

readers with a thorough explanation of image annotation, labeling, and subsequent training of YOLO models through this elucidation. As a result of this initial annotation, a second expert reviewed and verified the labels, effectively implementing a double-blind review process. As a result of this rigorous approach, the annotations were reliable, and human error was minimized.

A double-blind validation system was incorporated to ensure the integrity of the annotations. The system worked as follows:

- Blinding of the annotation: After the primary expert completed the annotation, the identity of this expert and the specific annotations were concealed from a secondary expert reviewing the paper.
- Secondary Review: An expert who is unaware of the primary annotator's decisions independently reviewed the images and made annotations.
- Comparison and Validation: Any discrepancies between the annotations of the two experts were discussed and reconciled to ensure the utmost accuracy of the labels.

As a result of this procedure, we not only established an effective checks-and-balances system but also ensured that any biases or errors from one expert could not affect the final dataset. By using a rigorous double-blind approach, the annotations were more reliable, reducing the possibility of human error.

After the annotations were finalized, the Dataset was divided into three subsets so the YOLOv8 model could be trained, tested, and validated. The model learned underlying patterns and relationships between the input features and ripeness categories by training 70% of the images. In addition, 20% of the images were reserved for testing, enabling the model to be evaluated on previously unexplored data. Finally, 10% of the images were validated, allowing the model's hyperparameters to be fine-tuned and generalization capabilities to be assessed. Based on three ripeness categories, table 1 presents a comprehensive data distribution analysis across each dataset source. Figure 2 shows some sample images from both samples.

2.2. Run-time preprocessing

CLAHE (Contrast Limited Adaptive Histogram Equalization): Given the variety of lighting conditions seen in our collected images, standardizing brightness and improving contrast was critical. As a result, we used CLAHE to ensure localized contrast enhancement on smaller tiles or regions of the image. More specifically:

- **Tile Size (or Grid Size):** Typically, an 8×8 grid divides the image into 64 regions. During histogram equalization, each of these regions is treated independently, which makes it particularly useful for images with differing lighting conditions.
- **Clip Limit:** The clip limit was set to 2.0 by default. By setting this value, the amplification of the contrast is restricted so the bin count is not greater than the clip limit multiplied by the bin's average. Due to this limitation, overamplification of contrast and possible noise amplification are prevented.

  The CLAHE method was implemented with these parameters so that tomatoes with subtle color variances or in shadowed regions could be accurately detected. Consequently, our model was able to adapt to a variety of lighting conditions.
- **Gaussian Blur:** By using Gaussian blur, noise was reduced and features were enhanced:
- **Kernel Size:** Typically, a 5×5 kernel is used. As the kernel size is crucial, a larger kernel would result in a more pronounced blur, which might not be suitable for all images.

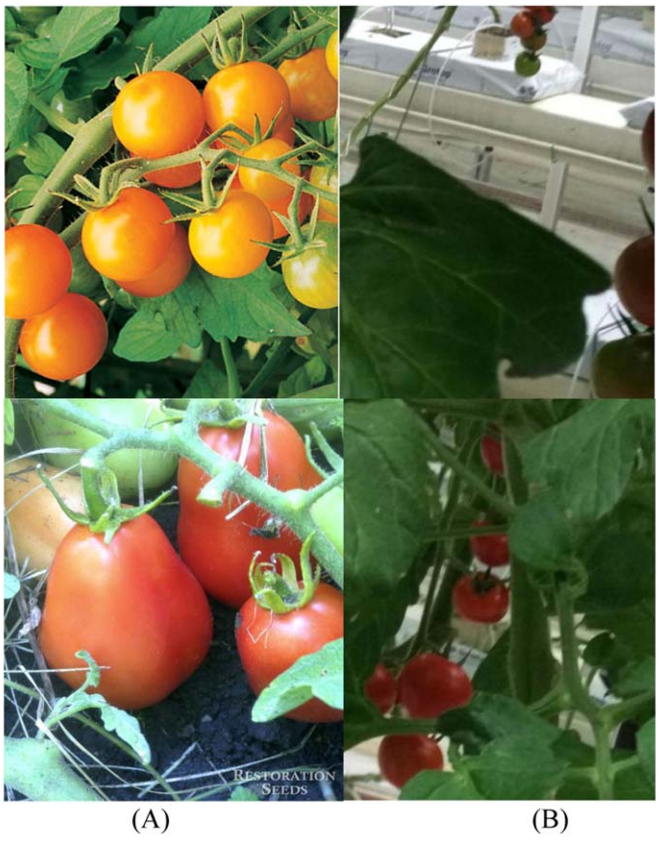


Figure 2. Some images from (A) Internet and (B) Kaggle dataset.

- **Standard Deviation (σ):** Our algorithm was given a standard deviation value of 0 for both X and Y directions which allowed us to calculate the optimal standard deviation based on the kernel size.

These parameters helped accentuate the color and boundaries of the tomatoes, making them easier to identify during detection, and reducing the risk of false positives due to image noise..

2.3. YOLOv8

On January 11, 2023, Ultralytics introduced its latest state-of-the-art object detection model, YOLOv8, following the success of its earlier versions (YOLOv3 and YOLOv5) [29]. YOLOv8 includes several new features, including enhanced detection of objects, segmentation of images, and classification. Additionally, it exhibits significant improvements in accuracy, speed, and model complexity. Based on a revamped architecture and the integration of loss functions like CIoU and DFL for bounding box detection, along with binary cross-entropy for classification loss, YOLOv8 offers improved performance, especially for small objects [30]. By improving architecture and leveraging cutting-edge training approaches, it outperforms its predecessors on the COCO dataset [31], achieving enhanced accuracy with fewer parameters [32].

A new convolution block is incorporated into the YOLOv8 backbone to replace conventional 6×6 convolutions with more efficient 3×3 convolutions. As a general rule, smaller convolutions are more efficient, as they extract more granular features, while reducing the number of parameters, making the model more efficient and potentially faster. As a result of this modification, there are fewer parameters, the architecture is simplified, and the model may run faster without compromising its accuracy. A major improvement of YOLOv8 is the elimination of anchor boxes. In earlier YOLO versions, it was necessary to carefully select anchor boxes based on their size and ratio, making the optimization process difficult. The model predicts object locations based on their centers, circumventing the need for predefined anchor boxes.

The model would predict multiple bounding boxes for each anchor per grid cell if anchor boxes were used. The Non-Maximum Suppression (NMS) process would then be complicated by the large number of overlapped boxes for a single object. As a result, the anchor-free approach speeds up the inference process and simplifies the post-processing of the NMS results. The primary building block of the YOLO architecture previously used only the output from the final bottleneck layer. However, the C2F block now concatenates output from all bottleneck

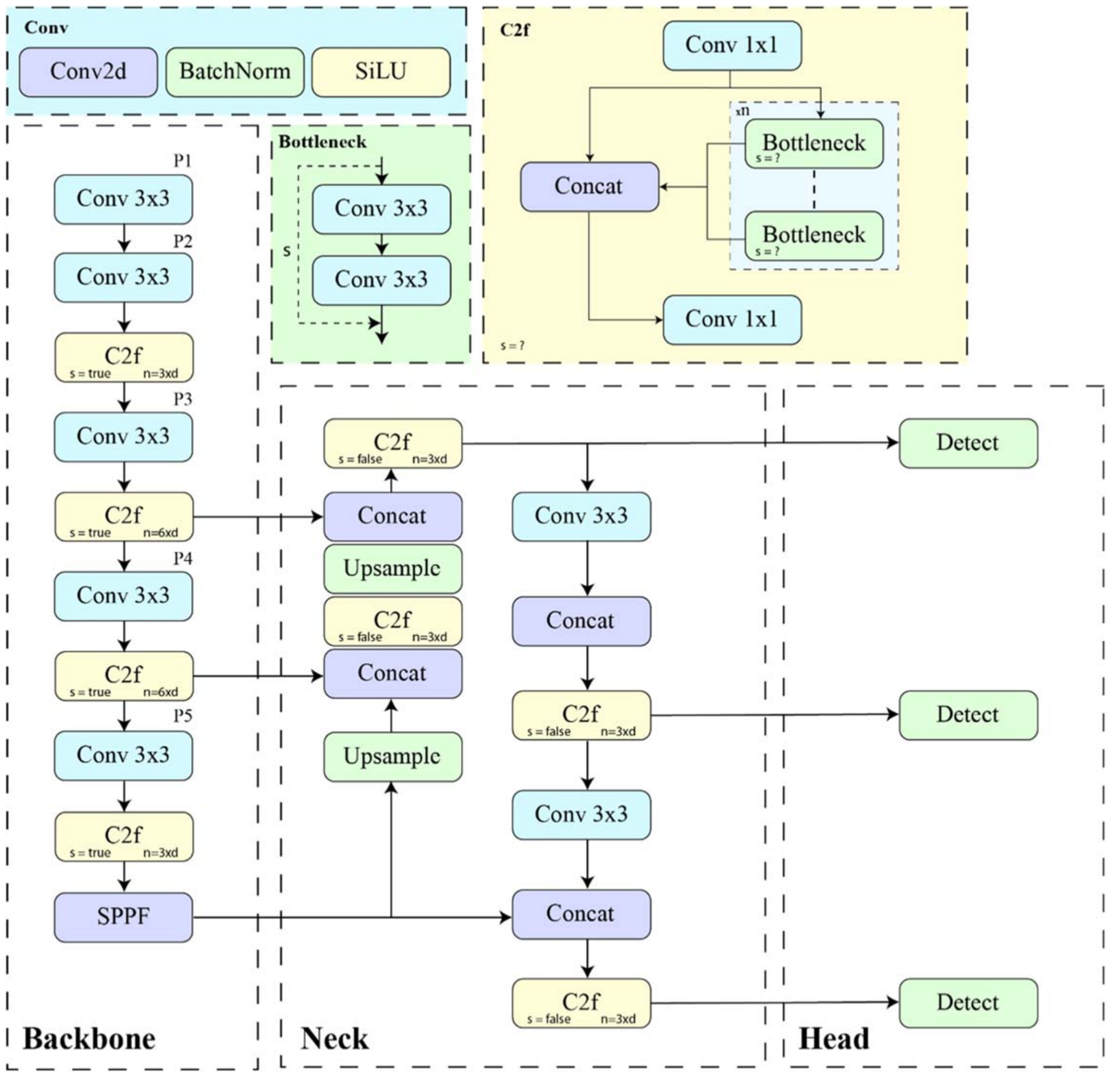


Figure 3. YOLOv8 architecture for accurate tomato ripeness detection with appropriate labels.

layers. This allows the network to access and utilize information from multiple stages, resulting in a richer, more comprehensive flow of data. Figure 3 illustrates the YOLOv8 architecture with appropriate labeling [33].

This study used the YOLOv8s (small) model for tomato ripeness detection. The figure shows that the YOLOv8 architecture consists of three main components: Backbone, Neck, and Head, designed for efficient and precise object detection. The Backbone extracts key features from input images through a series of Conv 3×3 layers and C2f modules, enhanced by Bottleneck layers for efficient feature aggregation and an SPPF (Spatial Pyramid Pooling Fast) layer to improve feature resolution. The Neck refines these features and fuses information from different scales using upsampling, concatenation, Conv 3×3 layers, and additional C2f modules, enabling robust multi-scale feature representation. Finally, the Head performs detection at multiple scales, utilizing specialized layers to predict bounding boxes and object classes with high accuracy. This architecture is well-suited for tasks like tomato ripeness detection, leveraging its efficient multi-scale feature processing and precise detection capabilities.

We assessed the YOLOv8s models with a high-quality, pre-processed tomato dataset in order to ensure that there were enough training samples for each class. The models were trained for 100 iterations, using stochastic gradient descent (SGD) as the optimizer and a callback with a patience of 20 as the callback method. Depending on the level of ripeness of the tomato, there may be specific differences in color and texture. The hyperparameters of our model have been fine-tuned after meticulous tuning to better capture these nuances. The learning rate was set to 0.01 and the batch size to 8. As mentioned in the previous section, our model performance has been enhanced by several run-time preprocessing techniques. Several key metrics were used to evaluate the models' performance, including precision, recall, mean average precision at 50% intersection over union (mAP@50), and computational resources (e.g., training time, memory usage, and inference speed).

Several experiments were conducted to analyze the impact of model size and complexity on overall performance, taking into account factors such as false positives, false negatives, and computational resources. Based on this thorough evaluation, we identified the optimal YOLOv8s model for effectively detecting tomato ripeness and classifying it with appropriate labels while balancing accuracy and computational efficiency. A comprehensive analysis of this study is important for the development of reliable tomato detection systems and can contribute to the advancement of agricultural robots for picking tomatoes.

2.3.1. Experimental setup

The experimental setup was implemented on Google Colab, leveraging the computational power of an NVIDIA Tesla T4 GPU with 15 GB of memory, supported by a dual-core Intel Xeon CPU operating at 2.00 GHz and 26 GB of RAM. The software environment consisted of Python 3.9.16 and PyTorch 1.13, ensuring compatibility with the chosen machine learning framework and libraries. All models were trained using the following hyperparameters and settings:

- **Epochs:** 100 (maximum number of epochs for training).
- **Early Stopping Patience:** 20 epochs (to terminate training if no improvement is observed).
- **Batch Size:** 8 images per batch (with -1 available for auto-batch adjustment if required).
- **Image Size:** 640 pixels (uniform input size as an integer or specified as width and height).
- **Data Loading Workers:** 8 threads (for efficient data loading).
- **Optimizer:** Stochastic Gradient Descent (SGD)
- **Mosaic Augmentation:** Disabled in the final 10 epochs.
- **Close Mosaic:** 10 (number of final epochs with mosaic augmentation disabled).

2.4. Hardware deployment

Using Raspberry Pi as a deployment target, we deployed our model. Raspberry Pi is a small, affordable, single-board computer developed by the Raspberry Pi Foundation. This platform is popular due to its ability to run ML/DL algorithms without requiring a dedicated computer or internet connection, enabling edge computing. Due to its size limitations and the lack of internet access, Raspberry Pi is a great deployment target. A number of recent works [34–36] have exploited this for edge deployment of ML/DL algorithms. The Raspberry Pi model 4 is the latest version of the Raspberry Pi, and has been selected for this particular project. The Raspberry Pi v4 comes with a 64-bit ARM Cortex-A72 processor, 1/2/4/8 GB of LPDDR4 RAM (the 4GB model was used in this study), WiFi, Bluetooth, HDMI, and a CSI Camera port. The board is very portable since it is powered by a single-cell Li-Ion battery and has a small footprint of 2.5' x 2.2' x 0.47'.

A heavy model like YOLO (You Only Look Once) on a Raspberry Pi presents several significant challenges due to the Raspberry Pi's limited computational power compared to GPUs or high-end CPUs. Raspberry Pi devices, even the more capable Raspberry Pi 4, have limited computational power compared to modern desktop and server machines. YOLOv4, as a deep neural network with a large number of parameters, requires substantial computational resources. In the case of high-resolution images or videos, running such a model on a Raspberry Pi can be slow and fails to achieve real-time performance. Additionally, YOLO models typically have a large memory footprint, whereas Raspberry Pi devices typically have a limited amount of RAM. The loading of large models and processing of images or videos can lead to memory exhaustion, which may cause the application to crash or slow down significantly. Finally, achieving real-time performance (e.g., 30 frames per second for video processing) can be challenging, especially for high-resolution inputs. There are also several limitations, including limited library support, high power consumption, and no dedicated GPU. All these obstacles had to be overcome in order to deploy our model to the Raspberry Pi. First, the lower performance of the YOLOv8 model was mitigated by using the YOLOv8s model, which has significantly fewer computing resources and is optimized for low-powered devices. The limited RAM issue was resolved by using a comparatively high-end Raspberry Pi with 4GB of RAM, which was sufficient for our model.

Without access to a dedicated GPU, achieving real-time performance posed a challenge. To address this, we optimized our software pipeline using multithreading, which distributed tasks across multiple threads and improved processing speed. With these optimizations, we achieved an inference time of less than 500ms, corresponding to 2 FPS. This frame rate was intentionally chosen to balance energy efficiency and thermal management, ensuring optimal performance on low-power devices like the Raspberry Pi. Running intensive models like YOLOv8 over extended periods can generate significant heat, leading to thermal throttling. To

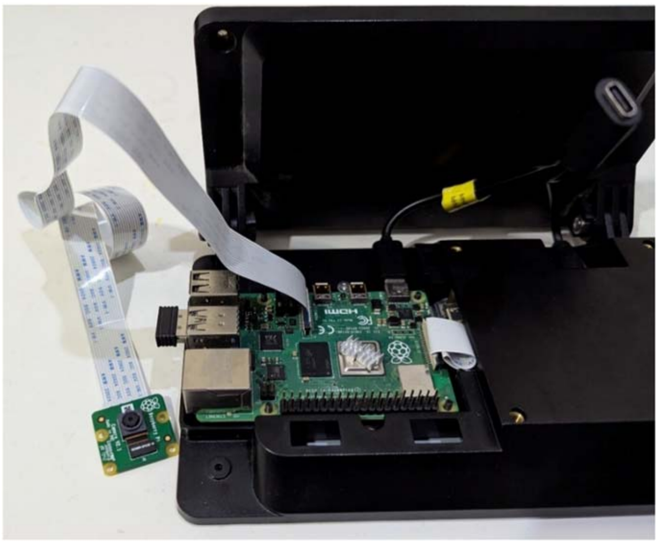


Figure 4. Hardware Setup.

prevent this, we implemented a robust cooling solution using a high-quality aluminum heatsink and a cooling fan, maintaining CPU temperatures below 65°C during inference and ensuring stable and efficient operation.

Figure 4 illustrates the hardware setup for the proposed model on the Raspberry Pi 4 with a camera module. The process of deploying a YOLOv8 model on a Raspberry Pi 4 with a Raspberry Pi Camera Module involves several steps, including setting up the Raspberry Pi, installing the necessary software and libraries, and configuring the camera module. First, the Raspbian OS was installed on the device, which is the official operating system. Next, Python and OpenCV software dependencies were installed. A headless version of the OpenCV library was installed, which is more suitable for low-end devices without all of its functionalities. Our next step was to configure the Raspberry Pi interfacing options to enable the camera module for Raspberry Pi. Then, we installed the Pytorch library, which is needed to run the YOLOv8 model. There was no pre-built binary for the ARM CPU, so the library had to be compiled from source code. Finally, we transferred the pre-trained YOLOv8s weights to the Raspberry Pi storage, which will be loaded during inference. During inference, the images were captured directly from the Raspberry Pi camera module at a resolution of 960 × 640 pixels. Nevertheless, the images were cropped to 640 by 640 before feeding them to the model. During the inference process, these images were fed to the model, and no additional pre-processing steps were required.

It is crucial to balance accuracy and speed when deploying on a Raspberry Pi, as there are trade-offs to consider. Due to Raspberry Pi's lower performance, we had to deploy the smaller YOLOv8s model, which has slightly lower performance than the large model. In addition to hardware acceleration, input image resolution, preprocessing steps, and thermal conditions, several other factors affect real-time performance. As discussed earlier, we utilized minimal pre-processing steps, multithreading, and thermal cooling to reduce the inference time. Additionally, the framerate was kept low, close to 2FPS, to ensure the model ran smoothly. However, the Raspberry Pi can still be an effective platform for detecting and classifying tomato ripeness on-device, if these challenges are taken into account and appropriate optimization strategies are employed. According to figure 5, Raspberry Pi 4 inference times are compared to those of a laptop computer with an Intel Core i5 processor. Raspberry Pi 4 CPUs have four cores and a clock at 1.8GHz, while laptop CPUs have eight cores and a clock at 4.2GHz. Raspberry Pi inference time increases by 4–5 times. However, we were able to achieve near real-time detection with the YOLOv8s model running in Raspberry Pi with an inference time of 665.04 ms.

3. Assessment metrics and implementation

A variety of metrics, such as the Mean Average Precision (mAP), Precision, Recall, and F1-Score, were used to evaluate the performance of the YOLOv8 models. According to equations (1) and (2), recall measures the number of real positives detected, whereas precision evaluates the ratio of accurately identified positives to all predicted positives [37, 38]. In equation (3), the F1-Score is defined as the harmonic mean of precision and recall.

$$Precision = \frac{T_P}{T_P + F_P} \tag{1}$$

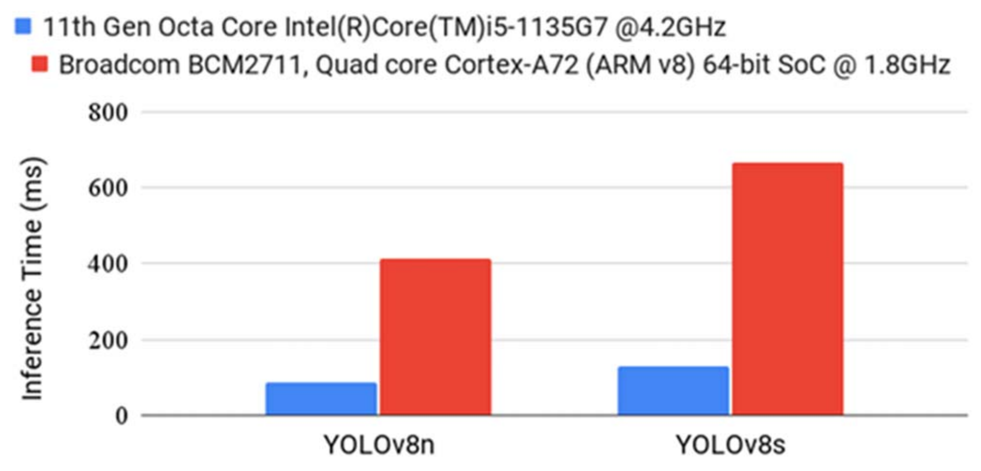


Figure 5. Comparison of inference times between a Raspberry Pi 4 and a laptop computer with Intel Core i5 CPU.

$$Recall = \frac{T_P}{T_N \quad + \quad F_P} \tag{2}$$

$$F1 - Score = 2*\frac{\text{Precision}*\text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

Average Precision (AP) is defined as the area under the Precision–Recall curve, while mAP is defined as the mean of AP across all classes. A widely used metric, these metrics are defined in equations (5) and (6) [39].

$$AP = \int_0^1 p(r)dr \tag{5}$$

$$mAP = \frac{1}{n}\sum_{i=1}^{n} AP_i \tag{6}$$

Where true positives, true negatives, false positives, and false negatives were represented by T_P, T_N, F_P and F_N, respectively. n is the number of classes, AP_i is the average precision for class i and p(r) signifies the precision at recall r. It is imperative to define true positives before calculating these metrics. The Intersection over Union (IoU) metric is employed to achieve this. Essentially, it is the ratio between the intersection and union of the prediction bounding box and the ground truth bounding box. As a result of the formula below, IoU is calculated:

$$IoU = \frac{A \cap B}{A \cup B} \tag{7}$$

Where, A and B are ground truth and prediction bounding boxes. A higher IoU value indicates a closer match between the prediction and ground truth bounding boxes [40]. Predictions are considered true positive if the IoU score equals or exceeds 0.5, as is the standard practice in object detection [41–43]. A prediction with an IoU score below 0.5 is classified as a false positive. An image is labeled false negative if none of the predictions match the bounding box of the ground truth. Only the prediction with the highest confidence score is considered a true positive if multiple predictions exceed the IoU threshold for the same ground truth. All other predictions are considered false positives. A mAP calculated using a threshold of 0.5 is referred to as a mAP@50. Additionally, the YOLOv8s model's classification efficacy was evaluated using a confusion matrix (CM).

4. Numerical results and discussion

A comprehensive examination and discussion of the outcomes of two datasets are presented in this section. Our primary objective is to combine these datasets seamlessly for the training of the cutting-edge YOLOv8s model, and then analyze the results from each dataset and the combined dataset comprehensively. Our approach ensures robustness and generalizability by allocating 75% of total images from each dataset for training, 15% for testing, and 10% for validation.

4.1. The kaggle dataset

This section presents a comprehensive analysis of the performance of YOLOv8s, which was trained on 1,397 instance distributions evenly distributed across the unripe (286), partially ripe (165), and ripe (946) categories. During validation, 279 specimens were used, including 39 unripe, 33 partly ripe, and 207 ripe specimens. We

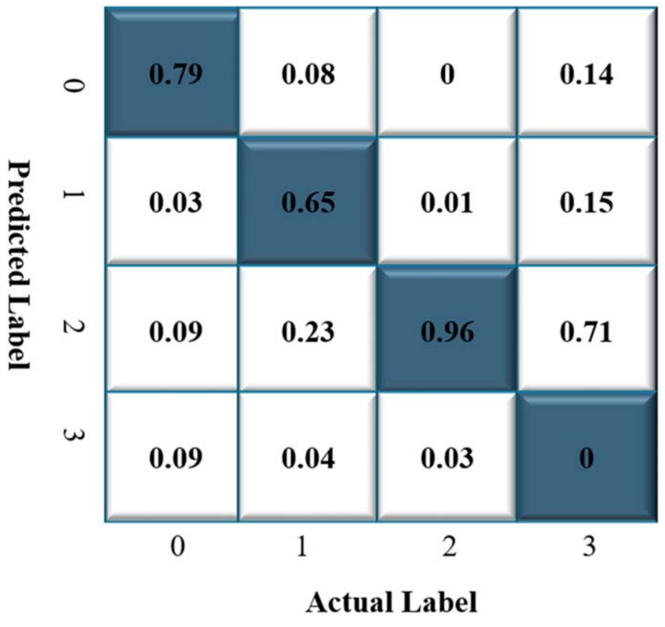


Figure 6. Confusion matrices for the Kaggle dataset for the YOLOv8s model (0: unripe, 1: partially ripe, 2: ripe, and 3: background).

Table 2. Yolov8s prediction on Kaggle dataset performance breakdown.

Class	Instances	Precision	Recall	F1-score	mAP@50
All	332	0.735	0.769	0.752	0.808
Unripe	55	0.80	0.80	0.80	0.853
Partially Ripe	38	0.682	0.62	0.650	0.74
Ripe	239	0.722	0.887	0.796	0.898

tested the model on 71 images containing 332 instances (unripe: 55, partially ripe: 38, and ripe: 239). Figure 6 shows a class-specific performance evaluation of the Kaggle dataset was conducted using the confusion matrix. Based on table 2, F1 scores for unripe, partially ripe, and ripe fruits are 0.80, 0.65, and 0.796, respectively. It can be attributed to the inherent difficulty of distinguishing between unripe and partially ripe samples, which leads to classification confusion within the model, that the partially ripe class exhibits a lower F1-score.

Based on figure 7, YOLOv8s delivers a mean average precision (mAP) at 50 of 0.896, which is commendable. Figure 8 illustrates the F1-Confidence Curve for the tomato ripeness detection task using the YOLOv8s model. The plot shows the variation of F1-score with respect to different confidence thresholds for each ripeness class: Unripe (blue curve), Partially Ripe (orange curve), and Ripe (green curve). The thick blue curve represents the aggregated F1-score across all classes. At lower confidence thresholds, the F1-score is higher due to the inclusion of more predictions, increasing recall. As the confidence threshold increases, the F1-score gradually declines, reflecting the trade-off between precision and recall. For the aggregated performance across all classes, the maximum F1-score of 0.84 is achieved at a confidence threshold of 0.354, as indicated by the thick blue curve. In figure 9, we illustrate the decreasing and optimizing trends for val/box_loss and val/cls_loss after training for 50 epochs.

Results presented in this section not only demonstrate the impressive performance of the YOLOv8s model but also provide insights into the underlying factors that contribute to its success. In addition, this study indicates the potential for widespread application of the model in high-impact applications like automatic tomato picking. The importance of determining tomato ripeness is further highlighted.

4.2. Internet-sourced dataset

This section evaluates the performance of the YOLOv8s model, which was trained using a dataset compiled from the internet consisting of 3,922 instances, categorized as unripe (1,402), partially ripe (782), and ripe (1,738). To ensure robust validation, we used 826 samples, including 356 unripe, 189 partially ripe, and 281 ripe samples. We assessed the model's effectiveness using 79 images with 1,621 instances (unripe: 668, partially ripe: 342, and ripe: 611).

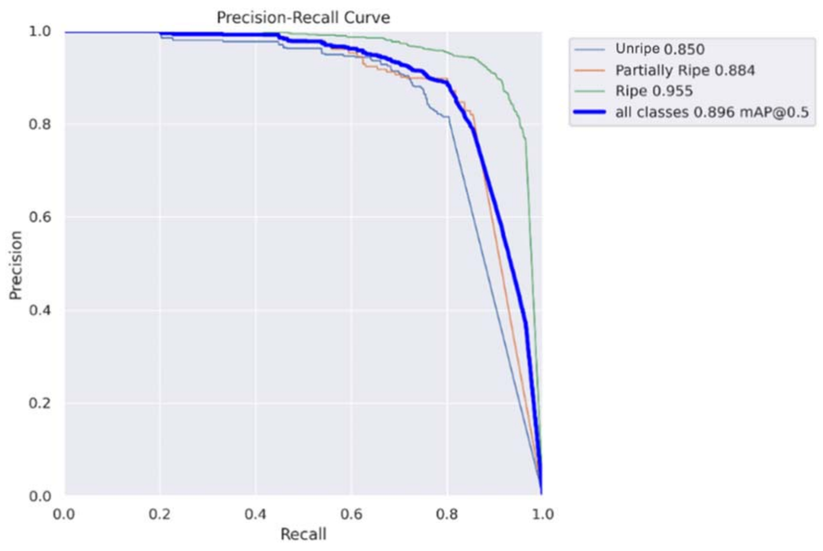


Figure 7. The precision–recall curve for Kaggle dataset for YOLOv8s model.

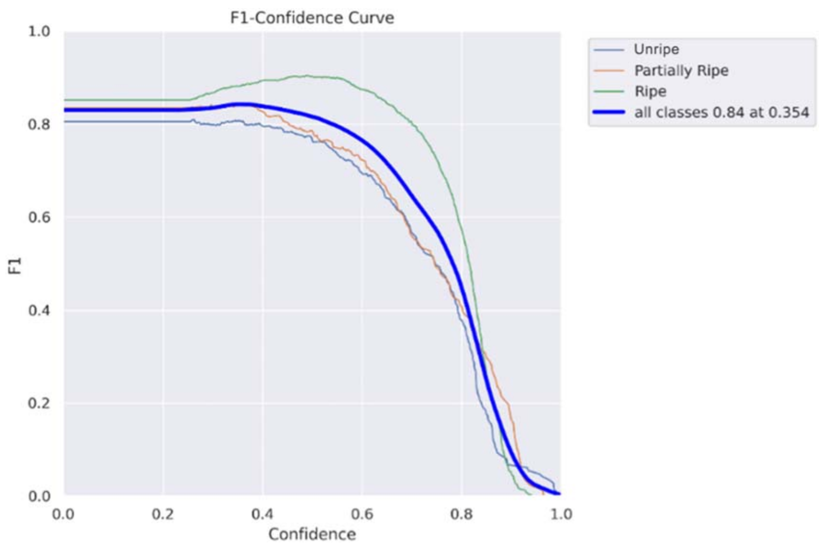


Figure 8. The F1-confidence curve for Kaggle dataset for YOLOv8s model.

According to table 3, the average precision, recall, and F1-scores are 0.747, 0.652, and 0.72, respectively, indicating a satisfactory performance. The confusion matrix of the Yolo v8x model is shown in figure 10. Figure 11 shows the YOLOv8s model's mAP at 50, which is 0.725, which highlights its commendable performance. As shown in figure 12, the F1-confidence curve provides further insight into the model's classification capabilities.

In figure 13, we highlight the decreasing trends and optimization of val/box_loss and val/cls_loss after 50 epochs. The comprehensive analysis of YOLOv8s' performance on the Internet-sourced Dataset not only demonstrates the model's effectiveness, but also allows us to better understand the underlying factors that contribute to its success.

4.3. Combined dataset

YOLOv8s's performance was evaluated using a combined dataset, combining internet-sourced and Kaggle-sourced data. The model's performance improved noticeably after this integration, as shown in table 4, with mAP@50 increasing from 0.725 to 0.745. Moreover, the combined dataset demonstrated 0.751, 0.689, and 0.719 F1-scores, respectively, indicating an increase in precision, recall, and F1-score. Figure 14 shows the confusion matrics of the Yolov8s model for the combined dataset.

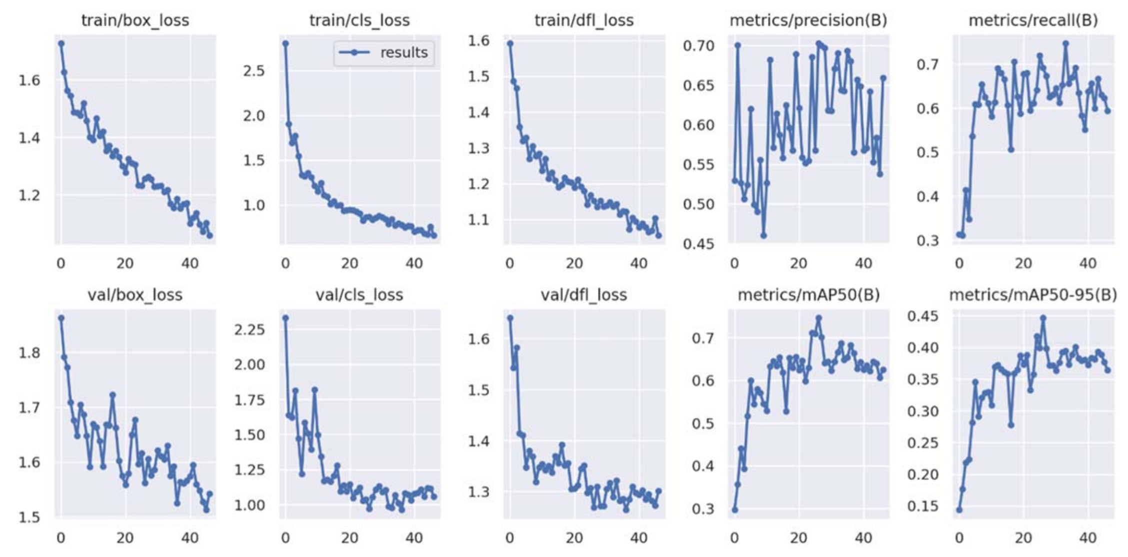
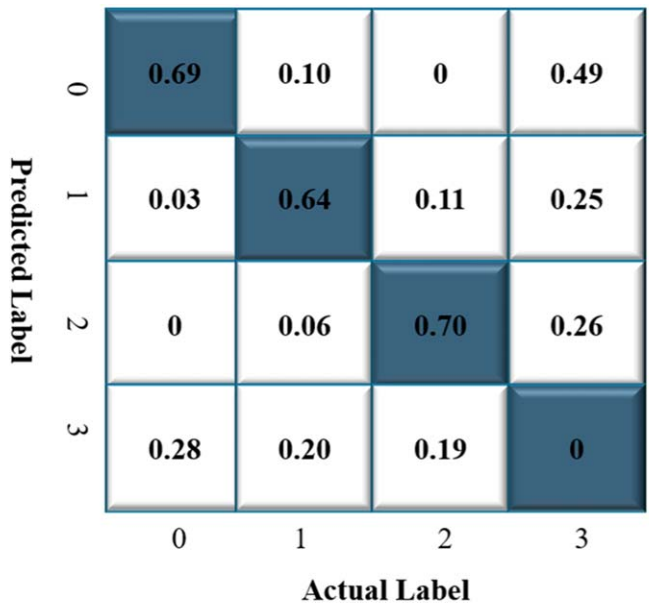


Figure 9. Different performance results Kaggle dataset for YOLOv8s model.



	0	1	2	3
0	0.69	0.10	0	0.49
1	0.03	0.64	0.11	0.25
2	0	0.06	0.70	0.26
3	0.28	0.20	0.19	0

Figure 10. Confusion matrices for Internet-sourced Dataset for YOLOv8s model (0: unripe, 1: partially ripe, 2: ripe, and 3: background).

Table 3. Yolov8s prediction on Internet-sourced dataset performance breakdown.

Class	Instances	Precision	Recall	F1-score	mAP@50
All	1,621	0.747	0.652	0.72	0.725
Unripe	668	0.739	0.637	0.684	0.717
Partially Ripe	342	0.647	0.638	0.643	0.664
Ripe	611	0.853	0.687	0.758	0.794

Despite this, an in-depth examination of the F1-scores indicates that unripe and partially ripe tomatoes have lower F1-scores of 0.698 and 0.658, respectively, in comparison with ripe tomatoes' F1-score of 0.801. There is a discrepancy between unripe and partially ripe tomatoes due to their visual similarity, making it difficult for the detection model to distinguish between them. Consequently, the model's performance is diminished. A number of studies have classified tomatoes into ripe and unripe categories rather than separating them into unripe and

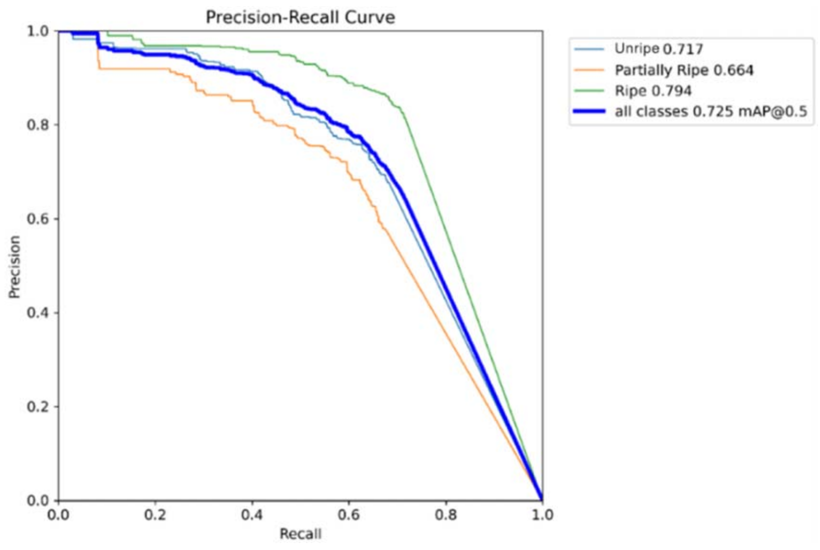


Figure 11. The precision–recall curve for Internet-sourced Dataset for YOLOv8s model.

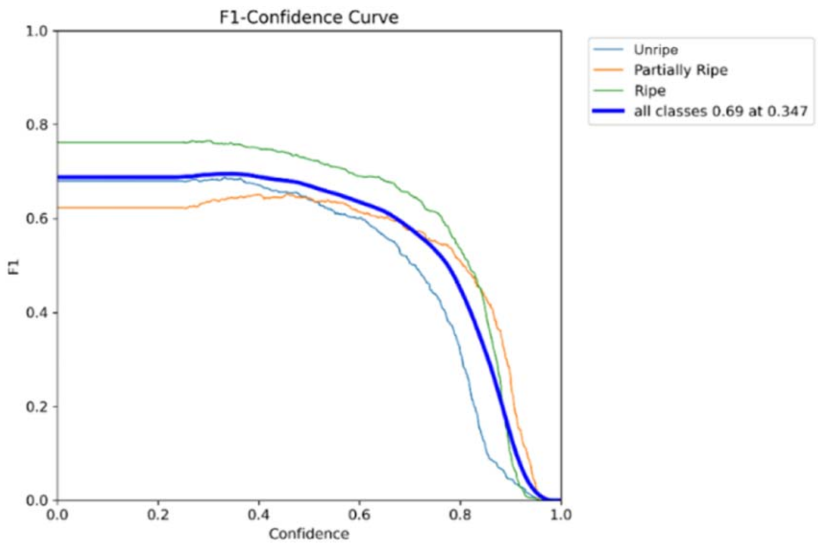


Figure 12. The F1-confidence curve for Internet-sourced Dataset for YOLOv8s model.

partially ripe categories. Nevertheless, it is important to distinguish between unripe and partially ripe fruit. The precision–recall curve and F1 confidence curve of the Yolov8s model for combined dataset is shown in figures 15 and 16, respectively. Figure 17 illustrates the differences in performance metrics between the various classifications, providing a comprehensive understanding of the YOLOv8s model's efficacy on the combined dataset. Figure 18 shows some images from the validation set.

4.3.1. Comparison with other state-of-the-art models

To evaluate the performance of our approach, we compared it against several state-of-the-art object detection models, including YOLOv8s, YOLOv5s, YOLOv5l [44], RTDETR-l, RTDETR-x [45], and YOLOv9c[46]. The comparison was conducted based on the mAP@50 metric, as shown in table 5, across three ripeness classes: Unripe, Partially Ripe, and Ripe, along with the overall performance (All).

Table 6 highlights the mAP@50 values for each model. Among the models, YOLOv8s achieved the highest overall mAP@50 score of 0.745, demonstrating its robust detection performance across all ripeness categories. It also showed the best performance for the Ripe category with an mAP@50 of 0.845. However, for the Unripe category, YOLOv9c performed slightly better, achieving an mAP@50 of 0.725, indicating its strength in detecting less mature tomatoes. YOLOv5l also exhibited strong performance, achieving an overall mAP@50 of 0.739, closely trailing YOLOv8s. Additionally, YOLOv5s delivered competitive results with an overall mAP@50

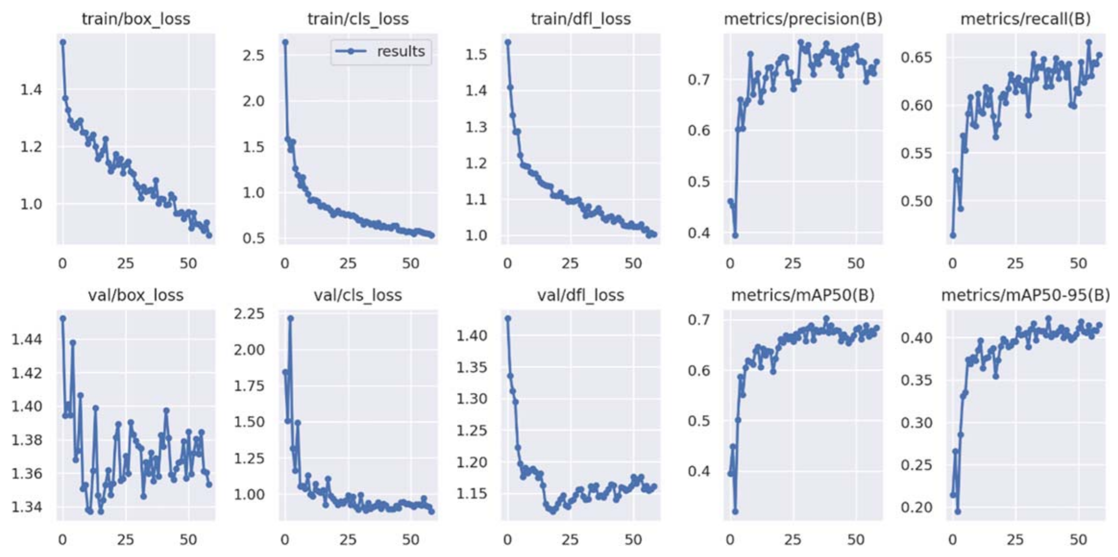


Figure 13. Different performance results Internet-sourced Dataset for YOLOv8s model.

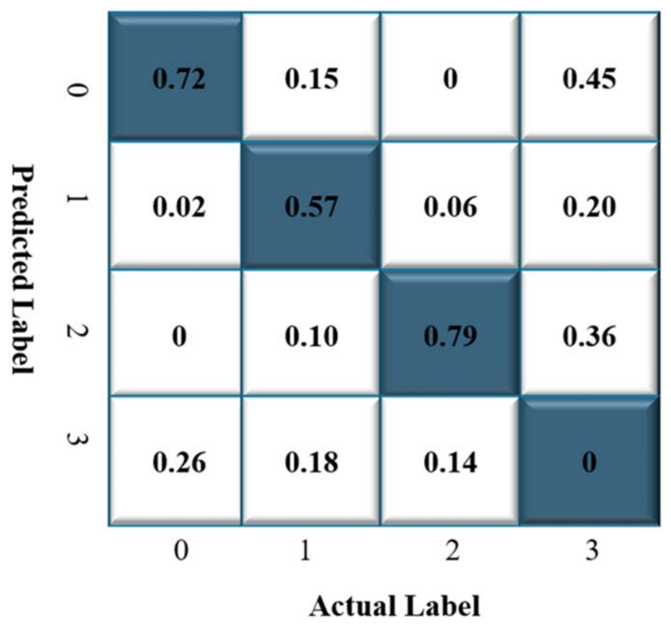


Figure 14. Confusion matrices for combined Dataset for YOLOv8s model (0: unripe, 1: partially ripe, 2: ripe, and 3: background).

Table 4. Yolov8s prediction on combined dataset performance breakdown.

Class	Instances	Precision	Recall	F1-score	mAP@50
All	1,953	0.751	0.689	0.719	0.745
Unripe	723	0.716	0.678	0.696	0.717
Partially Ripe	380	0.716	0.608	0.658	0.672
Ripe	850	0.820	0.783	0.801	0.845

of 0.725, making it a viable lightweight alternative. RTDETR models (RTDETR-l and RTDETR-x) demonstrated moderate performance in comparison, with RTDETR-l outperforming RTDETR-x in all categories except for the Partially Ripe category. RTDETR-l achieved the best mAP@50 score for Unripe but lagged behind the YOLO models on other categories. From figure 19 and figure 20, it can be observed that the prediction from YOLOv8s

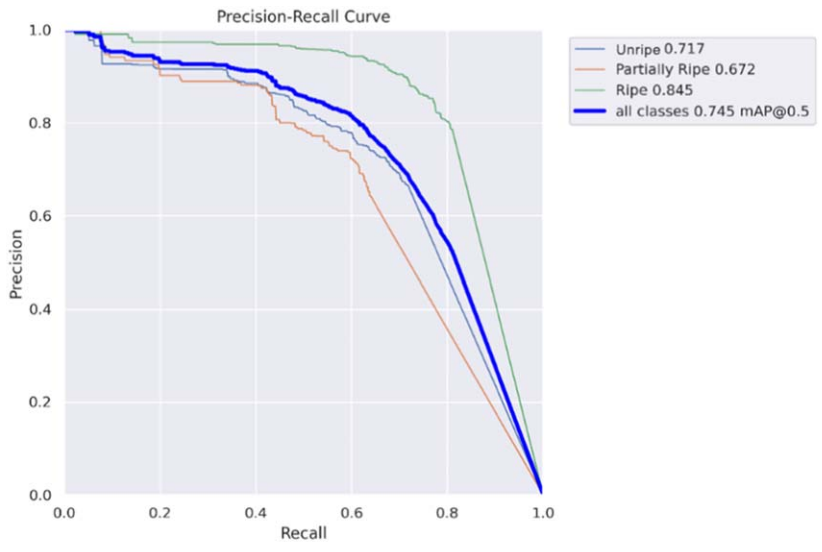


Figure 15. The precision–recall curve for combined Dataset for YOLOv8s model.

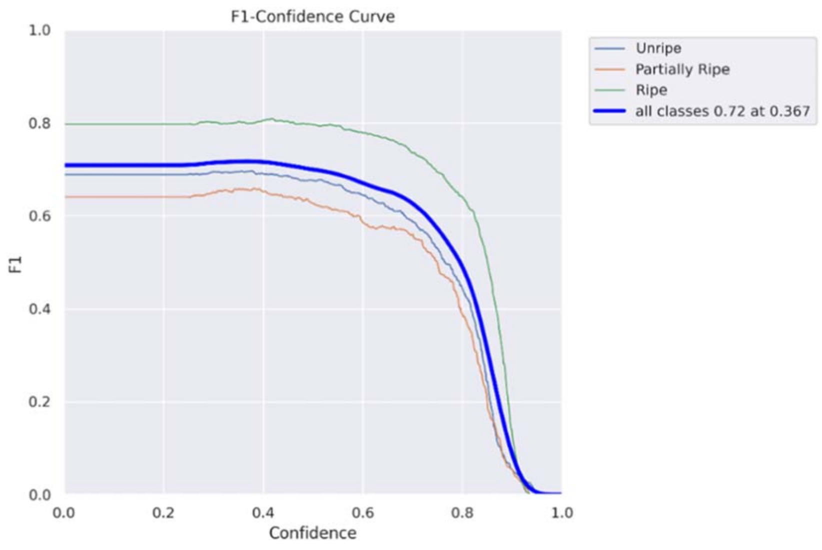


Figure 16. The F1-confidence curve for combined Dataset for YOLOv8s model.

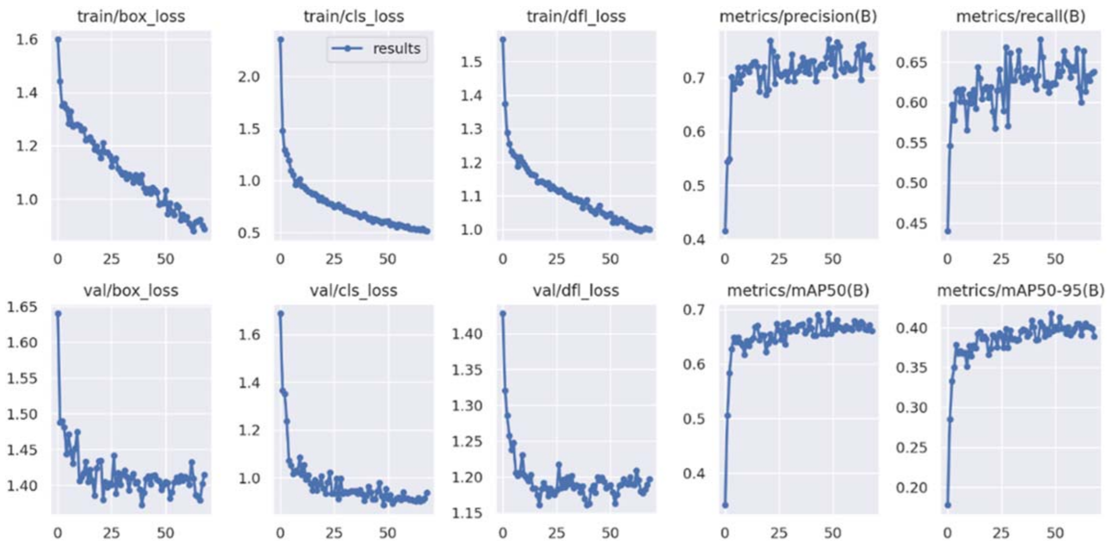
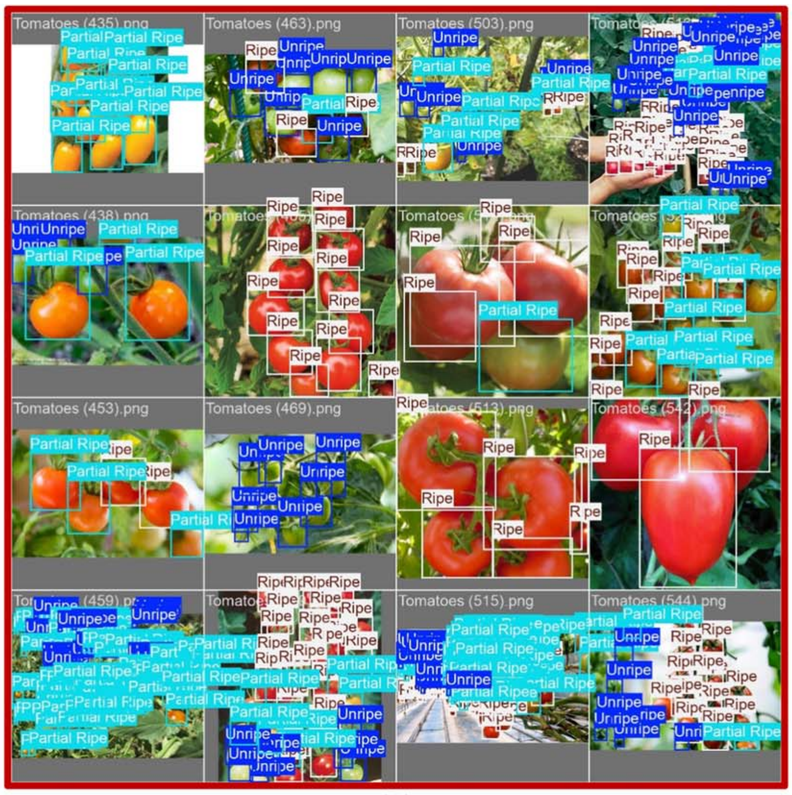


Figure 17. Different performance results combined Dataset for YOLOv8s model.


(A)

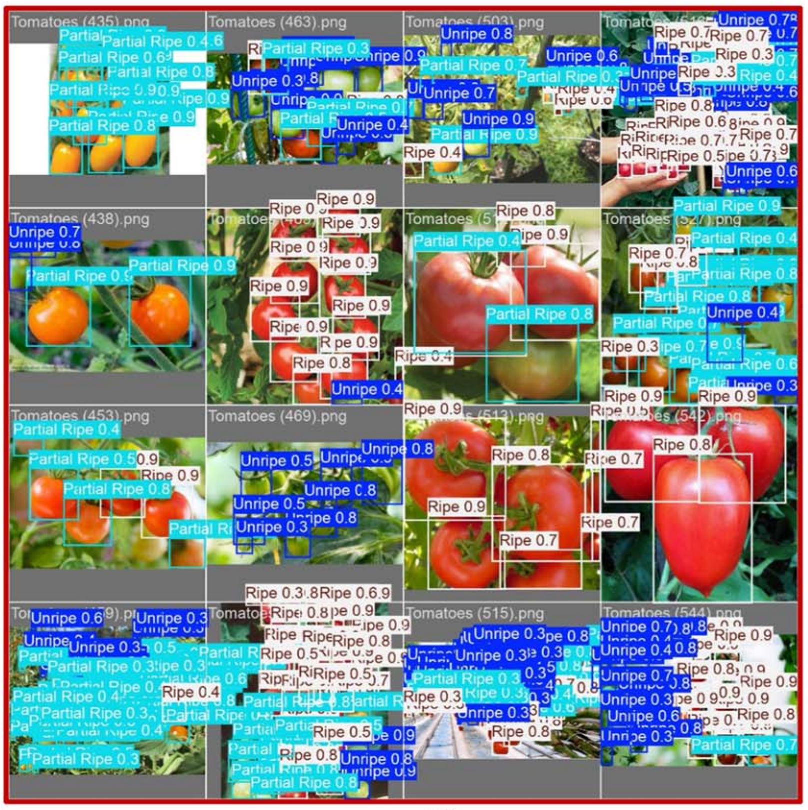

(B)

Figure 18. Sample images of (A) Actual (B) Predicted for the combined Dataset.

are cleaner and closely align with the ground truth while other models frequently overpredict both in dense and less dense scenes. In conclusion, YOLOv8s consistently outperformed other state-of-the-art models in overall performance, making it the most effective model for tomato ripeness detection.

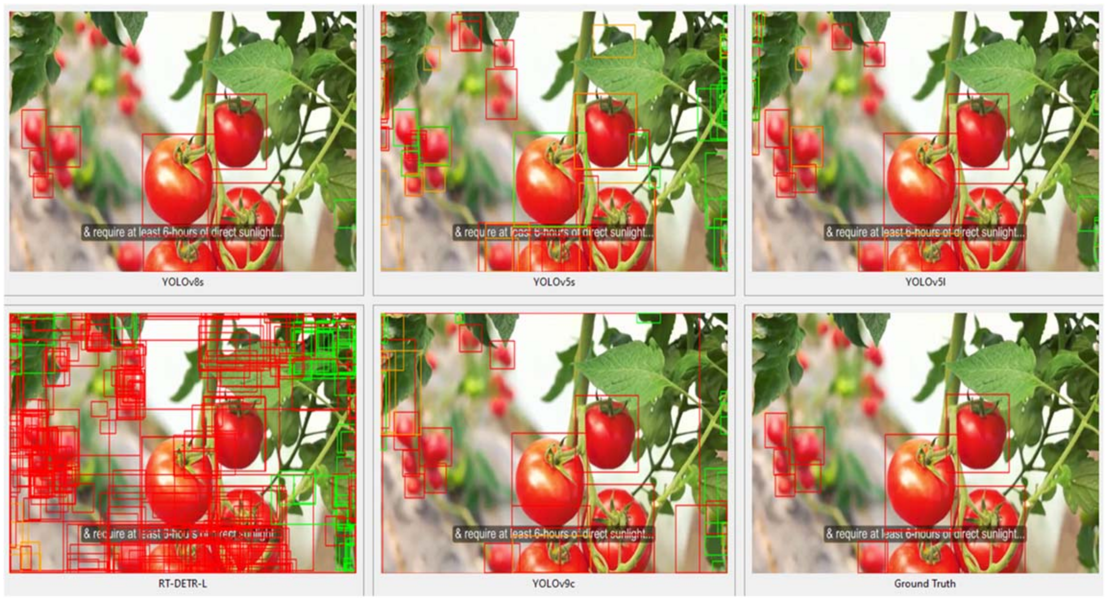


Figure 19. Comparison of model performance on less dense scenes (Green: Unripe; Orange: Partially Ripe; Red: Ripe).

Table 5. Comparison with other models on Combined Dataset.

Model	Unripe	Partially Ripe	Ripe	All
YOLOv8s	0.717	0.672	0.845	0.745
YOLOv5s	0.695	0.668	0.811	0.725
YOLOv5l	0.716	0.657	0.843	0.739
RTDETR-l	0.739	0.621	0.820	0.727
RTDETR-x	0.691	0.607	0.823	0.707
YOLOv9c	0.725	0.656	0.839	0.740

4.4. Prototype deployment and performance

We tested a proof-of-concept deployment of our tomato detection system using the YOLOv8s model implemented on a Raspberry Pi. The setup involved the Raspberry Pi processing video footage of tomatoes played on a TV screen. Despite the constrained computational resources of the Raspberry Pi, the system performed reasonably well, achieving near real-time detection. The device successfully identified tomatoes in the video stream with consistent accuracy, demonstrating the feasibility of deploying lightweight models in resource-constrained environments. Figure 21 illustrates the deployment setup and detection results. A video demonstration of the system is provided at the following link: https://youtu.be/CDy3DOMHUv4.

5. Performance comparison with existing literature

This section compares the performance of our YOLOv8s model with a variety of state-of-the-art (SOTA) models. The study is notable for addressing the detection of three distinct tomato ripeness classes: unripe, partially ripe, and ripe. Consequently, the detection results differ significantly from previous studies.Many previous studies have focused on binary classification (ripe and unripe tomatoes), with Lawal's study achieving the highest mAP@50 of 99.5%. Although the difference between ripe and unripe tomatoes may seem simple, their model comprises 53 layers with 63 million parameters, taking 52 milliseconds to detect a single tomato [20]. Contrary to this, our model contains only 11.11 million parameters (nearly 5 times fewer) and takes only 17.6 milliseconds to detect. In our study, we aim to develop a simple, lightweight model that can be seamlessly integrated into embedded systems. Implementing our model on a Raspberry Pi shows its potential for real-world deployment in tomato-picking robots and other agricultural applications. Moreover, some researchers have explored three-class detection (flower, ripe, and unripe), but none have considered partially ripe tomatoes. In this category, Egi *et al* [21] achieved the best mAP@50 at 63%. However, their model has 88 million parameters. With only 11.11 million parameters (approximately 8 times fewer), our YOLOv8s model performs nearly as well as theirs with a mAP@50 of 80.8% (17% higher) on the Kaggle dataset and 74.5% (11% higher) on

Table 6. Performance comparison with SOTA models.

Authors	Dataset Count Instance and Images	Class type	Model	Layers	Parameters	Testing Time	Precision (in %)	Recall (in %)	mAP@50
Zhao *et al* [3]	225 Images	2-class (Healthy and Diseased tomatoes)	YOLOv2	29	50 M	—	96.00	97.00	91.00
Magalhães *et al* [19]	297 Images	2-class (Unripe and reddish tomatoes)	SSD MobileNetv2	—	—	16.44ms	84.37	54.40	51.46
Lawal [20]	425 Images	2-class (ripe and unripe)	YOLOv3	53	63 M	52ms	97.00	99.30	99.50
Zheng *et al* [1]	1,698 Tomato Instances	2-class (mature and immature)	R-CSPDarknet53 &CSPP	—	—	—	87.60		
Egi *et al* [21]	1,097 Images and 6,957 Tomato Instances	3-class (flower, unripe tomato, and red tomato)	YOLOv5	25	88M	—	74.10	57.00	63.00
Our Dataset (Proposed)	**387 Images and 6,368 Tomato Instances**	**3-class (Ripe, partial and unripe) Custom**	**YOLOv8s**	**224**	**11.11 M**	**17.6 ms**	**74.70**	**65.20**	**72.50**
Kaggle dataset with our Proposed model	**359 Images and 2,008 Tomato Instances**	**3-class (Ripe, partial and unripe) Kaggle**	**YOLOv8s**	**224**	**11.11 M**	**17.6 ms**	**73.50**	**76.90**	**80.80**
Combined Dataset (Proposed)	**746 Images and 8,376 Tomato Instances**	**3-class (Ripe, partial and unripe) Combined**	**YOLOv8s**	**224**	**11.11 M**	**17.6 ms**	**75.10**	**68.90**	**74.50**

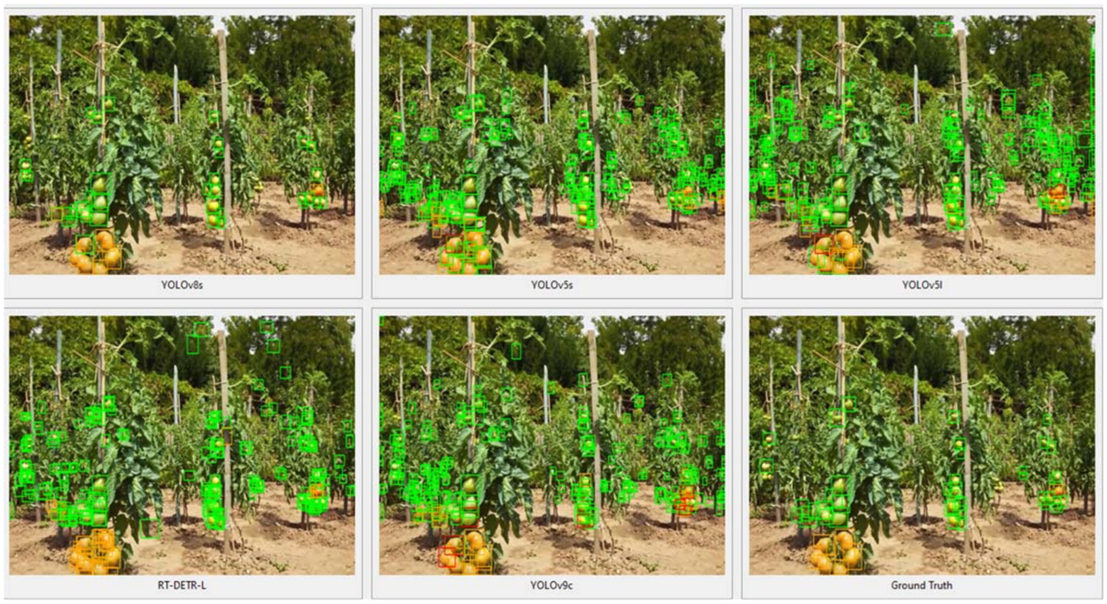


Figure 20. Comparison of Model performance on dense scenes (Green: Unripe; Orange: Partially Ripe; Red: Ripe).

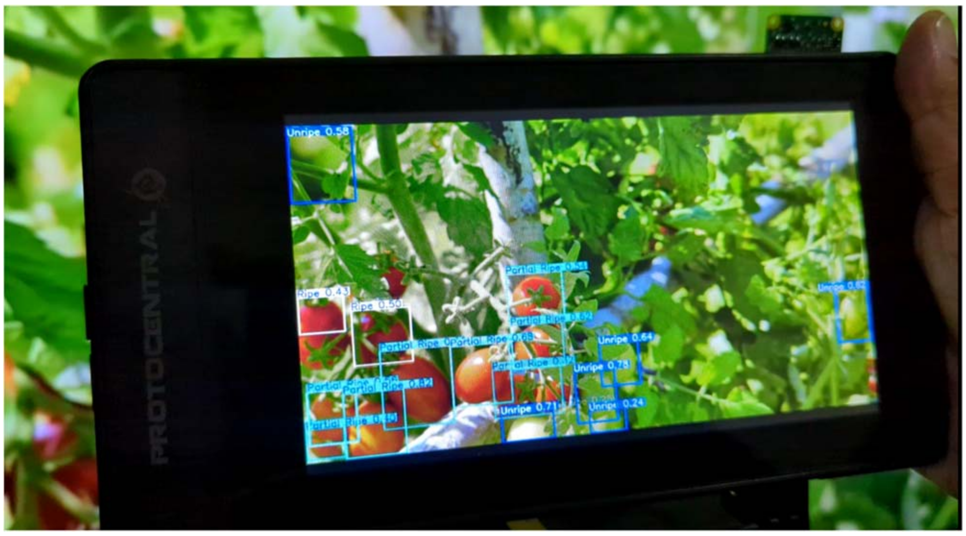


Figure 21. Performance of hardware prototype.

the combined dataset. In addition to demonstrating the robustness of our model, these results also demonstrate that it is suitable for real-world applications, reinforcing its position as a leading solution for detecting tomato ripeness.

In summary, our YOLOv8s model outperforms existing SOTA models in detecting tomato ripeness across three classes, while maintaining a lean, efficient architecture. With its simplicity and lightweight nature, our model is an ideal candidate for deployment on embedded systems, allowing new agricultural applications to be created and setting a new standard.

6. Conclusion

This study presents a groundbreaking approach for detecting tomato ripeness using the YOLOv8 object detection algorithm. Compared to existing state-of-the-art models, our model is more accurate and efficient, making it a pioneering solution. The YOLOv8 model was trained on a comprehensive tomato image dataset and was able to detect three distinct ripeness classes: unripe, partially ripe, and ripe. The results obtained from the Kaggle dataset demonstrated a mean average precision (mAP) at 50 of 0.808, along with F1-scores of 0.80

(unripe), 0.65 (partially ripe), and 0.796 (ripe). Using the Internet-sourced Dataset, the model achieved a mAP@50 of 0.725, with F1-scores of 0.747 (unripe), 0.652 (partially ripe), and 0.72 (ripe). We are confident that our model is effective and robust based on these results.Additionally, we seamlessly integrated the YOLOv8 model into a Raspberry Pi-based embedded system, paving the way for automated tomato picking. By integrating these systems, we can reduce the need for manual labor, reduce labor costs, and minimize crop damage during harvest. In real-world applications, our model's lightweight and efficient architecture makes it practical to deploy. As a result of our study, a new benchmark in tomato ripeness detection has been established, enabling advancements in crop management and automated harvesting systems. The model's exceptional accuracy and efficiency, demonstrated through rigorous evaluation and validation, reinforce its superiority over existing methods. As part of future work, we plan to incorporate different tomato varieties and environmental conditions into the Dataset to enhance the model's adaptability. Additionally, further research can explore the integration of advanced computer vision techniques and conduct field trials to validate the model's performance on a variety of farms. Beyond tomato harvesting, our approach has implications for other fruit and vegetable classification tasks, contributing to the advancement of smart farming and precision agriculture.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://kaggle.com/datasets/andrewmvd/tomato-detection.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

ORCID iDs

Amith Khandakar https://orcid.org/0000-0001-7068-9112
M Murugappan https://orcid.org/0000-0002-5839-4589
Muhammad E H Chowdhury https://orcid.org/0000-0003-0744-8206

References

[1] Zheng T, Jiang M, Li Y and Feng M 2022 Research on tomato detection in natural environment based on RC-YOLOv4 *Comput. Electron. Agric.* **198** 107029
[2] Wan P, Toudeshki A, Tan H and Ehsani R 2018 A methodology for fresh tomato maturity detection using computer vision *Comput. Electron. Agric.* **146** 43–50
[3] Zhao J and Qu J 2019 Healthy and diseased tomatoes detection based on YOLOv2 in*Human Centered Computing: 4th Int. Conf., HCC 2018, Mérida, Mexico, December, 5–7, 2018, Revised Selected Papers 4* (Springer) 347–53
[4] Chen J *et al* 2021 Detecting ripe fruits under natural occlusion and illumination conditions *Comput. Electron. Agric.* **190** 106450
[5] Su F, Zhao Y, Wang G, Liu P, Yan Y and Zu L 2022 Tomato maturity classification based on SE-YOLOv3-MobileNetV1 network under nature greenhouse environment *Agronomy* **12** 1638
[6] Chowdhury M E *et al* 2021 Automatic and reliable leaf disease detection using deep learning techniques *AgriEngineering* **3** 294–312
[7] Rupanagudi S R, Ranjani B, Nagaraj P and Bhat V G 2014 A cost effective tomato maturity grading system using image processing for farmers *2014 Int. Conf. on Contemporary Computing and Informatics (IC 3I)* (IEEE) 7–12
[8] Abisha S *et al* 2023 Brinjal leaf diseases detection based on discrete shearlet transform and deep convolutional neural network *PLoS One* **18** e0284021
[9] Kurtulmus F, Lee W S and Vardar A 2011 Green citrus detection using eigenfruit, color and circular gabor texture features under natural outdoor conditions *Comput. Electron. Agric.* **78** 140–9
[10] Martis J E *et al* 2024 Novel hybrid quantum architecture-based lung cancer detection using chest radiograph and computerized tomography images *Bioengineering* **11** 799
[11] Murugappan M, Nagarajan R and Yaacob S 2009 Appraising human emotions using time frequency analysis based EEG alpha band features *2009 Innovative Technologies in Intelligent Systems and Industrial Applications* 70–5
[12] Karthikeyan P *et al* 2012 EMG signal based human stress level classification using wavelet packet transform *Communications in Computer and Information Science* (*Trends in Intelligent Robotics, Automation, and Manufacturing*) vol 330 ed S G Ponnambalam, J Parkkinen and K C Ramanathan (Springer) IRAM 2012 (https://doi.org/10.1007/978-3-642-35197-6_26)
[13] Hemalakshmi G R *et al* 2024 PE-Ynet: a novel attention-based multi-task model for pulmonary embolism detection using CT pulmonary angiography (CTPA) scan images *Phys. Eng. Sci. Med.* **47** 863–80
[14] Ahamed M F *et al* 2024 Automated colorectal polyps detection from endoscopic Images using MultiResUNet framework with attention guided segmentation *Hum-Cent. Intell. Syst.* **4** 299–315
[15] Wang G-Q *et al* 2024 A high-accuracy and lightweight detector based on a graph convolution network for strip surface defect detection *Adv. Eng. Inf.* **59** 102280

[16] Wang G *et al* 2024 High-accuracy and lightweight weld surface defect detector based on graph convolution decoupling head *Meas. Sci. Technol.* **35** 105025
[17] Wang G-Q *et al* 2023 YOLO-MSAPF: multiscale alignment fusion with parallel feature filtering model for high accuracy weld defect detection *IEEE Trans. Instrum. Meas.* **72** 1–14
[18] Liu G, Nouaze J C, Touko Mbouembe P L and Kim J H 2020 YOLO-tomato: a robust algorithm for tomato detection based on YOLOv3 *Sensors* **20** 2145
[19] Magalhães S A *et al* 2021 Evaluating the single-shot multibox detector and YOLO deep learning models for the detection of tomatoes in a greenhouse *Sensors* **21** 3569
[20] Lawal O M 2021 Development of tomato detection model for robotic platform using deep learning *Multimedia Tools Appl.* **80** 26751–72
[21] Egi Y, Hajyzadeh M and Eyceyurt E 2022 Drone-computer communication based tomato generative organ counting model using YOLO V5 and deep-sort *Agriculture* **12** 1290
[22] Ge Y *et al* 2022 Tracking and counting of tomato at different growth period using an improving YOLO-deepsort network for inspection robot *Machines* **10** 489
[23] Wang X and Liu J 2021 Tomato anomalies detection in greenhouse scenarios based on YOLO-Dense *Frontiers in Plant Science* **12** 634103
[24] Nugroho D P, Widiyanto S and Wardani D T 2022 Comparison of deep learning-based object classification methods for detecting tomato ripeness *International Journal of Fuzzy Logic and Intelligent Systems* **22** 223–32
[25] Liu G, Hou Z, Liu H, Liu J, Zhao W and Li K 2022 TomatoDet: Anchor-free detector for tomato detection *Frontiers in Plant Science* **13** 942875
[26] Turečková A *et al* 2022 Slicing aided large scale tomato fruit detection and counting in 360-degree video data from a greenhouse *Measurement* **204** 111977
[27] Zhang S, Zhang C and Yang Q 2003 Data preparation for data mining *Appl. Artif. Intell.* **17** 375–81
[28] Larxel 2020 https://kaggle.com/datasets/ andrewmvd/tomato-detection (accessed on 2 Jan 2024)
[29] Glenn J, Ayush C and Jing Q (2023) *YOLO by Ultralytics (version 8.0. 0) GitHub* https://github.com/ultralytics/ultralytics (accessed on 10 Mar 2024)
[30] Terven J, Córdova-Esparza D-M and Romero-González J-A 2023 A comprehensive review of yolo architectures in computer vision: from yolov1 to yolov8 and yolo-nas *Machine Learning and Knowledge Extraction* **5** 1680–716
[31] Lin T-Y *et al* 2014 Microsoft coco: common objects in context in*Computer Vision–ECCV 2014: 13th European Conf., Zurich, Switzerland, September 6–12, 2014, Proc., Part V 13* (Springer) 740–55
[32] Waxmann G J a S Ultralytics YOLOv8 Docs. 2023 https://docs.ultralytics.com/models/yolov8 (accessed on 2 Jan 2024)
[33] Solawetz J 2023 "What is yolov8? the ultimate guide", Roboflow Blog, (https://blog.roboflow.com/whats-new-in-yolov8/)
[34] Curtin B H and Matthews S J 2019 Deep learning for inexpensive image classification of wildlife on the Raspberry Pi *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conf. (UEMCON)* (IEEE) 0082–7
[35] Bhosale Y H and Patnaik K S 2022 IoT deployable lightweight deep learning application for COVID-19 detection with lung diseases using RaspberryPi *2022 Int. conference on IoT and blockchain technology (ICIBT)* (IEEE) 1–6
[36] Dewangan D K and Sahu S P 2020 Deep learning-based speed bump detection model for intelligent vehicle system using raspberry Pi *IEEE Sens. J.* **21** 3570–8
[37] Swets J A 1988 Measuring the accuracy of diagnostic systems *Science* **240** 1285–93
[38] Powers D M 2020 Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation *arXiv preprint arXiv 2010.16061*
[39] Padilla R, Netto S L and Da Silva E A 2020 A survey on performance metrics for object-detection algorithms *2020 international conference on systems, signals and image processing (IWSSIP)* (IEEE) 237–42
[40] Rezatofighi H, Tsoi N, Gwak J, Sadeghian A, Reid I and Savarese S 2019 Generalized intersection over union: a metric and a loss for bounding box regression *Proc. of the IEEE/CVF conference on computer vision and pattern recognition* 658–66
[41] Wu P, Liu A, Fu J, Ye X and Zhao Y 2022 Autonomous surface crack identification of concrete structures based on an improved one-stage object detection algorithm *Eng. Struct.* **272** 114962
[42] Jin Y, Lu Y, Zhou G, Liu Q and Wang Y 2023 Glass wool defect detection using an improved YOLOv5 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* 4385–94
[43] Zhang L, Ding G, Li C and Li D 2023 DCF-Yolov8: an improved algorithm for aggregating low-level features to detect agricultural pests and diseases *Agronomy* **13** 2012
[44] Jocher G, Stoken A, Borovec J and NanoCode012 (2020) ultralytics/yolov5: v3.1 - bug fixes and performance improvements *Zenodo* (accessed on 5 Mar 2024) (https://doi.org/10.5281/zenodo.4154370)
[45] Zhao Y *et al* 2024 Detrs beat yolos on real-time object detection in*Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition* 16965–74
[46] Wang C-Y, Yeh I-H and Liao H-Y M 2025 Yolov9: learning what you want to learn using programmable gradient information in*European Conf. on Computer Vision* (Springer) 1–21

View publication stats