sustainability

MDPI

Article

Detecting Plant Diseases Using Machine Learning Models

Nazar Kohut [1], Oleh Basystiuk [1], Nataliya Shakhovska [1,2] and Nataliia Melnykova [1,*]

[1] Department of Artificial Intelligence, Lviv Polytechnic National University, 79013 Lviv, Ukraine; nazar.kohut.mknssh.2024@lpnu.ua (N.K.); oleh.a.basystiuk@lpnu.ua (O.B.); nataliya.b.shakhovska@lpnu.ua (N.S.)
[2] Department of Civil and Environmental Engineering, Brunel University of London, Uxbridge UB8 3PH, UK
* Correspondence: nataliia.i.melnykova@lpnu.ua

Abstract: Sustainable agriculture is pivotal to global food security and economic stability, with plant disease detection being a key challenge to ensuring healthy crop production. The early and accurate identification of plant diseases can significantly enhance agricultural practices, minimize crop losses, and reduce the environmental impacts. This paper presents an innovative approach to sustainable development by leveraging machine learning models to detect plant diseases, focusing on tomato crops—a vital and globally significant agricultural product. Advanced object detection models including YOLOv8 (minor and nano variants), Roboflow 3.0 (Fast), EfficientDetV2 (with EfficientNetB0 backbone), and Faster R-CNN (with ResNet50 backbone) were evaluated for their precision, efficiency, and suitability for mobile and field applications. YOLOv8 nano emerged as the optimal choice, offering a mean average precision (MAP) of 98.6% with minimal computational requirements, facilitating its integration into mobile applications for real-time support to farmers. This research underscores the potential of machine learning in advancing sustainable agriculture and highlights future opportunities to integrate these models with drone technology, Internet of Things (IoT)-based irrigation, and disease management systems. Expanding datasets and exploring alternative models could enhance this technology's efficacy and adaptability to diverse agricultural contexts.

Keywords: object detection; computer vision; YOLO; YOLOv8; EfficientDet; Faster R-CNN; CNN; agriculture; diseases

Academic Editors: Olexander Barmak, Iurii Krak, Eduard Manziuk and Pavlo Radiuk

Received: 28 November 2024
Revised: 23 December 2024
Accepted: 24 December 2024
Published: 27 December 2024

Citation: Kohut, N.; Basystiuk, O.; Shakhovska, N.; Melnykova, N. Detecting Plant Diseases Using Machine Learning Models. *Sustainability* **2025**, *17*, 132. https://doi.org/10.3390/su17010132

1. Introduction

Tomatoes are among the most widely grown and economically significant crops worldwide, and ensuring their health is crucial for optimal crop yields and high-quality produce. Tomatoes are one of the foremost cultivated crops globally, with far-reaching economic significance and indispensable contributions to global food security. The cultivation of tomatoes spans diverse agro-climatic zones, reflecting their adaptability and popularity among growers and consumers. However, this widespread cultivation has challenges, and maintaining tomato plant health is critical to ensure optimal yields and the production of high-quality fruits [1].

Disease detection and management have emerged as paramount considerations in safeguarding the health and productivity of tomato plants. The early detection of potential health issues represents a cornerstone in effectively managing diseases as it allows for timely intervention strategies to mitigate their impact. Often, the earliest manifestations of distress become apparent on the leaves, where subtle discolorations, spotting, or other anomalies serve as harbingers of impending disease spread throughout the plant.

Although environmental stressors such as fluctuating soil moisture, prolonged periods of rainfall, or heavy dews can induce abnormalities in tomato fruits (examples of tomato fruit abnormalities are presented in Figure 1), it is primarily on the foliage that the initial signs of disease manifest. Therefore, focusing on the early detection of symptoms on leaves is key to effective disease management [2].

Figure 1. Examples of tomato fruit abnormalities.

In this context, understanding the intricate interplay between environmental factors, plant physiology, and disease dynamics has assumed paramount importance. By unraveling the mechanisms underlying disease development and propagation in tomato plants, we can devise targeted approaches to bolster plant resilience, minimize disease incidence, and ultimately enhance agricultural sustainability and food security. This paper delves into the nuances of disease detection and management in tomatoes, elucidating key principles and strategies to empower growers to combat plant diseases and optimize crop yields.

Sunscald occurs when plants lose foliage due to disease or insect feeding. Blossom end rot is caused by calcium deficiency due to inconsistent watering, drought, or excessive nitrogen fertilizers. Cracks form from excess moisture, high temperatures, and are caused by heavy rains after a long dry period with high humidity [3]. Machine learning and artificial intelligence advancements have paved the way for efficient and precise methods of detecting tomato plant diseases. These technologies are especially useful in creating fast and accurate models suitable for use in the field. In particular, mobile applications and embedded devices provide real-time monitoring and intervention opportunities, aiding farmers in addressing issues promptly and maintaining crop health.

Early and efficient disease detection in tomato plants cannot be overstated, as it directly influences crop health, productivity, and economic viability. To address this imperative, our study undertook a comprehensive evaluation of several advanced and established models for disease detection. Among the advanced models scrutinized were YOLOv8's [4,5] minor and nano variants, renowned for its speed and accuracy in object detection tasks, and Roboflow 3.0 object detection (Fast), a cutting-edge solution tailored for real-time applications. Additionally, we investigated established models such as EfficientDetV2 [6] with an EfficientNetB0 [7] backbone and Faster R-CNN with a ResNet50 backbone, renowned for their robust performance in computer vision tasks.

In the context of this study, we developed a mobile-based system on Ionic Angular and TypeScript, using models in the ONNX format for offline inference. This system allows farmers to detect diseases in tomato plants in real-time, even in areas with limited or no Internet connectivity. This practical approach enables on-field applications to benefit from disease monitoring.

Through rigorous performance evaluation encompassing metrics of speed and accuracy, we endeavored to identify the optimal solutions for integration into our mobile-based system. By elucidating the strengths and limitations of each model, we aim to equip farmers and agricultural stakeholders with actionable insights to enhance disease monitoring and management practices. Ultimately, our endeavor seeks to empower farmers with technological tools that facilitate proactive disease intervention, fostering sustainable agricultural practices and ensuring food security in a rapidly evolving world.

2. Materials and Methods

Over the years, researchers have explored various techniques to tackle the challenge of tomato disease identification. Most efforts have centered around image classification and object detection methods. Image classification relies on analyzing a single leaf per image for accurate diagnosis, but this becomes problematic when multiple leaves are in the frame, each potentially carrying different diseases. On the other hand, object detection provides a more versatile solution, allowing the model to pinpoint and identify diseased areas within an image regardless of the number of leaves present. In this paper, we approached tomato disease identification as an object detection problem because this method aligns better with the task's inherent complexity, allowing us to accurately assess plant health by detecting multiple diseases within a single image.

Building on this approach, the MobileNetV2-YOLOv3 model, coupled with the GIoU loss function, was proposed to detect gray leaf mold disease in tomatoes. While this technique demonstrated strong performance in detecting a specific disease, its focus on a single type of disease must fully address our broader challenge: simultaneously identifying multiple diseases in tomato leaves.

In line with advancements in object detection, the Mobile Ghost with Attention YOLO network (MGA-YOLO) based on YOLOv5 was constructed [8] for the recognition of apple leaf diseases using a custom dataset, the Apple Leaf Disease Object Detection dataset (ALDOD), which combines data from the Plant Pathology 2021-FGVC8 [9] and Plant Pathology 2020-FGVC7 [10] datasets from Kaggle. This dataset encompasses four classes: healthy, rust, scab, and black rot. Although this research presented an innovative approach using a newer modification of the YOLO architecture, its focus on apple leaf diseases deviated from the scope of our work, which aims to detect diseases in tomato leaves. Despite the emphasis on apple leaf diseases, this approach and model could be adapted for detecting diseases in tomato leaves and disease identification in other crops.

Another study utilized a modified YOLOv5 architecture to detect plant diseases [11], particularly rubber tree diseases collected from a rubber plantation in Shengli State Farm, Maoming City, China. This model introduced the InvolutionBottleneck in place of the Bottleneck module within the C3 module and added an SE module to the last layer of the Backbone, which allowed for a weighted merging of image features of powdery mildew and anthracnose, thereby enhancing the network's performance with minimal additional cost. Moreover, the authors altered the loss function from generalized intersection over union (GIoU) to efficient intersection over union (EIoU).

The researchers compared their modified YOLOv5 architecture with the original YOLOv5 network and the YOLOX_nano model. Their results demonstrated a mean average precision (MAP) increase of 5.4% compared to the original YOLOv5, indicating a significant improvement in performance on the test results. While this research focused on a different crop and set of diseases, it offers valuable insights into potential improvements for plant disease detection through innovative YOLOv5 modifications. Nonetheless, the study utilized older versions of YOLO, which may not perform as well as newer architecture iterations. Therefore, there is room for improvement by leveraging these newer versions, which tend to outperform older versions.

A recent study [12] on tea leaf disease detection and identification examined the YOLOv7 (YOLO-T) architecture, which was evaluated against the YOLOv5 model. Although this research achieved improved performance, it required more time for training compared to YOLOv5. Despite leveraging a newer version of YOLO, the study's focus on tea leaves presents a different challenge from tomato leaf disease detection. Still, like previous research, this study addressed a different problem than ours, reflecting the diverse applications of advanced object detection techniques across plant species and diseases.

YOLOv8 small was used for a similar task involving the prediction of ripe, unripe, and diseased tomato fruits [13]. The approach incorporated depthwise separable convolutions (DSConv), an integrated dual-path gated (DPAG) attention module, and a feature enhancement module (FEM), with the authors evaluating the performance of each change both individually and collectively. The study highlighted notable differences in performance across different YOLO models including Faster R-CNN (80.8%), SSD (76.7%), YOLOv4 (88.4%), YOLOv5 (91.2%), YOLOv7 (91.6%), and the standard YOLOv8 (91.9%).

Despite the advancements demonstrated by the YOLOv8 model in this study, the research focused on identifying disease presence in tomatoes rather than tomato leaves. This focus on fruits rather than leaves makes the model well-suited for aiding tomato gathering. However, it also limits the model's applicability to detect diseases at earlier stages, which presents a different challenge from our task of identifying diseases in tomato leaves.

Advancements in machine learning and artificial intelligence have paved the way for efficient and precise methods of detecting tomato plant diseases. These technologies are especially useful in creating fast and accurate models suitable for use in the field. In particular, mobile applications and embedded devices provide opportunities for real-time monitoring and intervention, aiding farmers in addressing issues promptly and maintaining crop health [14].

The importance of early and efficient disease detection in tomato plants cannot be overstated, as it directly influences the crop health, productivity, and economic viability. To address this imperative, our study undertook a comprehensive evaluation of several advanced and established models for disease detection [15]. Among the advanced models scrutinized were YOLOv8 in its small and nano variants, renowned for speed and accuracy in object detection tasks, and Roboflow 3.0 object detection (Fast), a cutting-edge solution tailored for real-time applications. Additionally, we investigated established models such as EfficientDetV2 with an EfficientNetB0 backbone and Faster R-CNN with a ResNet50 backbone, renowned for their robust performance in computer vision tasks [16].

In the context of this study, we developed a mobile-based system on Ionic Angular and TypeScript, using models in the ONNX [17] format for offline inference. This system allows farmers to detect diseases in tomato plants in real-time, even in areas with limited or no Internet connectivity. This practical approach enables on-field applications to benefit from disease monitoring.

To identify optimal solutions for early and efficient disease detection in tomato plants, this paper compared several advanced models including YOLOv8 in its small and nano variants and Roboflow 3.0 object detection (Fast) [18]. We also examined established models such as EfficientDetV2 with an EfficientNetB0 backbone and Faster R-CNN with a ResNet50 backbone. By evaluating these models' performance in terms of speed and accuracy, our aim was to determine which was the most suitable for implementation in our mobile-based system.

In summary, the literature review illustrates diverse strategies for plant disease detection. Previous studies have relied on older models like YOLO, but recent research indicates that more recent iterations of YOLO consistently achieve superior performance. In tomato disease detection using leaf images, existing research has focused on a limited number of disease classes. Similar trends can be observed in studies examining diseases in other plants.

This work focused heavily on YOLO, with our primary model being YOLOv8. This choice was driven by YOLO, a one-stage model that generally offers faster performance than two-stage models like Faster R-CNN [19].

Another factor to consider was the use of datasets like PlantVillage, which were collected in controlled laboratory settings with simplistic backgrounds in some studies.

This could result in suboptimal performance [20,21] in real-world scenarios, where models must contend with diverse and complex environments found in the field.

Overall, while there have been meaningful advances in the field, there is room for improvement by leveraging the latest versions of object detection models and using more diverse and representative datasets to enhance the detection and classification of multiple diseases in tomato leaves.

This study leveraged YOLOv8 due to its one-stage architecture, which combines high accuracy with faster inference times, outperforming older versions and two-stage models like Faster R-CNN. YOLOv8's versatility, combined with robust training on diverse datasets, allows for the improved detection of multiple diseases in tomato leaves under real-world conditions. By adopting this state-of-the-art model, the research aimed to address gaps in the field, providing a scalable, efficient, and accurate solution for tomato disease identification.

3. Results

This section describes the dataset selection process, data preprocessing, and the training methods used for our models. We compare the performance of the YOLOv8 nano and minor variants and present the results achieved with these models.

3.1. Dataset

The choice of dataset is one of the most critical choices when training any machine learning model. As mentioned in previous sections, using data that does not accurately represent real-world scenarios can lead to unexpected outcomes. After reviewing the available datasets, we found several options including classification datasets such as PlantVillage and Tomato Disease Multiple Sources and object detection datasets such as the Tomato-Village dataset. Both types could have been better for our specific task.

Classification datasets like PlantVillage and Tomato Disease Multiple Sources typically contain one leaf per image. They require extensive relabeling to be re-purposed for object detection, making them less suitable for our task.

Regarding the object detection datasets, many of them needed more images or classes required for our task. However, during our evaluation, two datasets stood out. One was the Tomato-Village dataset, which provides separate annotations for classification and object detection tasks. However, this dataset presented particular challenges despite its potential, as depicted in Figure 2. The object detection labels often featured tiny bounding boxes, capturing images of leaves that a human observer would typically not consider necessary enough to label. Given a leaf positioned centrally and closer to the photographer's viewpoint, a human would naturally prioritize labeling it over leaves on the left or right. This discrepancy may have arisen due to the dataset's primary emphasis on classification, potentially leading to the automated generation of object detection labels.

Therefore, we decided to explore another option that better suited our needs. We ultimately chose to use a dataset from Roboflow, which presented a comprehensive collection of 4132 photographs depicting tomato leaves afflicted with eight distinct diseases: early blight, late blight, leaf miner, leaf mold, mosaic virus, septoria, spider mites, yellow leaf curl virus, alongside a class representing healthy plants. The authors collected the images, each with dimensions of 512 by 512 pixels, across both controlled laboratory settings and natural environments. Figure 3 illustrates the visual representations.

Considering this distribution, we decided to train the model using the available data, assuming that such a distribution may provide sufficient information for practical model training.

Figure 2. Examples of unusual object detection labels in the Tomato-Village dataset.

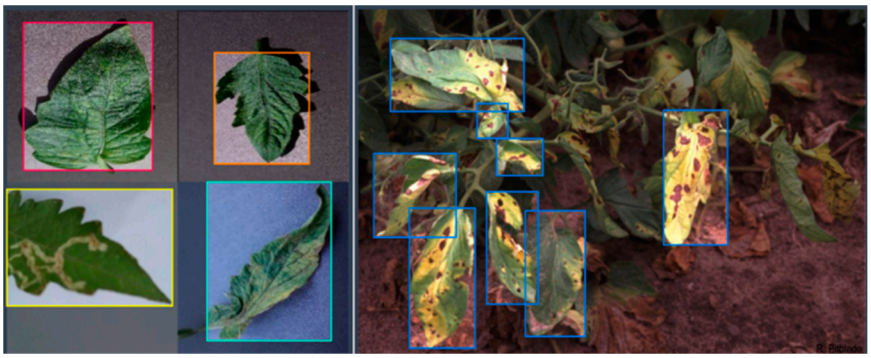


Figure 3. Examples of images gathered in the laboratory and natural environment.

3.2. Algorithms

Before initiating model training, it is essential to assess the class distribution imbalance as it can significantly impact the training dynamics and outcome accuracy. Within our dataset, most classes demonstrated a relatively balanced distribution, as seen in Figure 4, hovering around 1500 instances each. However, there were notable exceptions: yellow leaf curl virus exhibited the highest frequency with 2131 occurrences, while spider mites was slightly underrepresented with 1232 instances.

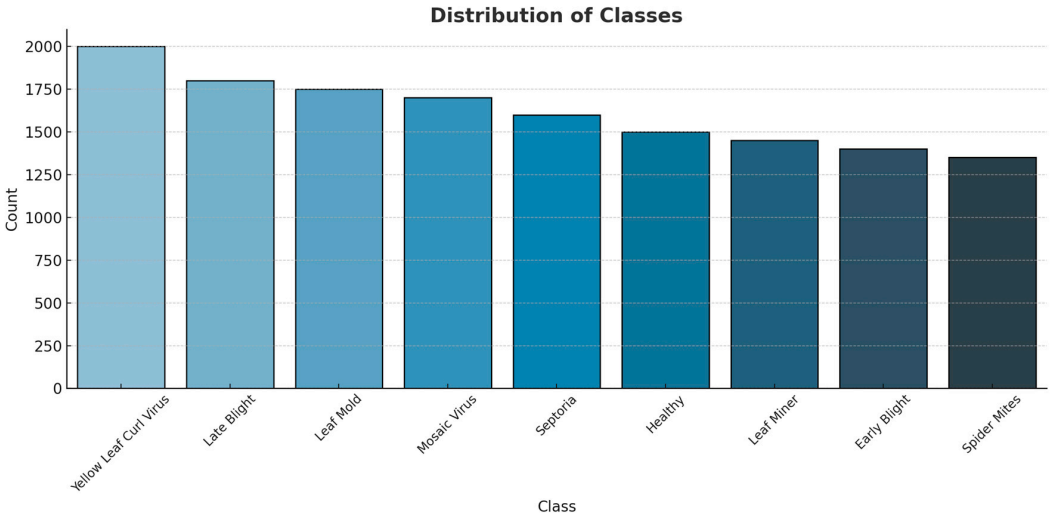


Figure 4. Class distribution.

Considering this distribution, we decided to train the model using the available data, assuming that such a distribution may provide sufficient information for practical model training.

In previous sections, we explained why it was essential to consider this problem as an object detection problem. Now that we have chosen the dataset, we may review the number of objects in each image in our data in Figure 5.

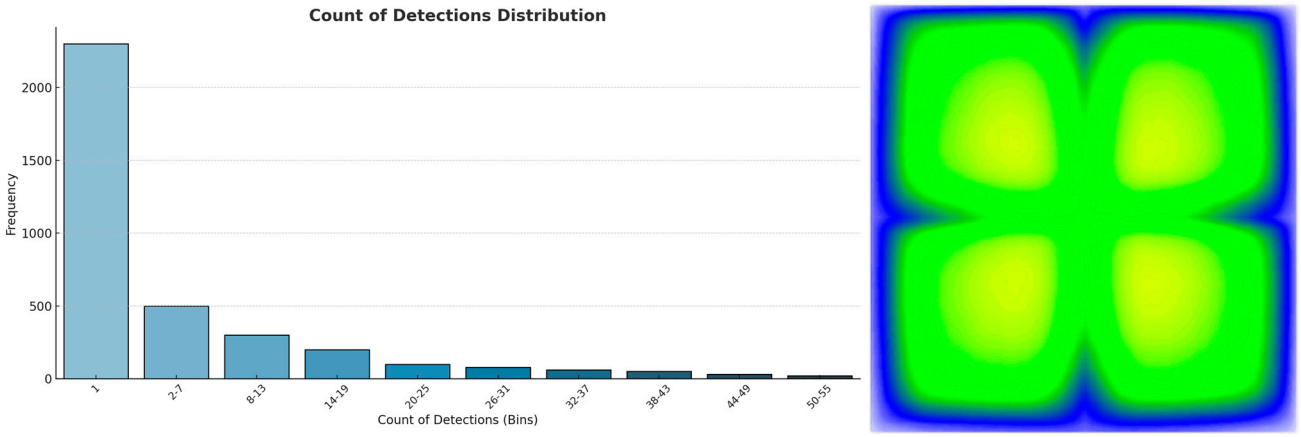


Figure 5. Dataset visualization: (**a**) Histogram of object count; (**b**) Annotation heatmap.

More than half of the images contained multiple objects, but it is important to note that our dataset included images from laboratory environments that typically contain a single leaf per image. These laboratory images are often combined into sets of four within the dataset, as illustrated in the first image in Figure 3, producing composite images containing four objects. Therefore, the actual number of images with multiple objects may be lower than that suggested by the histogram. Additionally, the annotation heatmap showed a higher concentration of images from laboratory settings than field settings. This highlights a potential need for a more balanced representation of field images, though this research does not address this aspect and leaves it for future work.

To further prove our point, we meticulously sifted through the dataset and discovered that the quantity of images captured in the fields ranged between 1300 and 1500. Notably, a small portion of these images had been deliberately placed against the background in one color (see Figure 6).

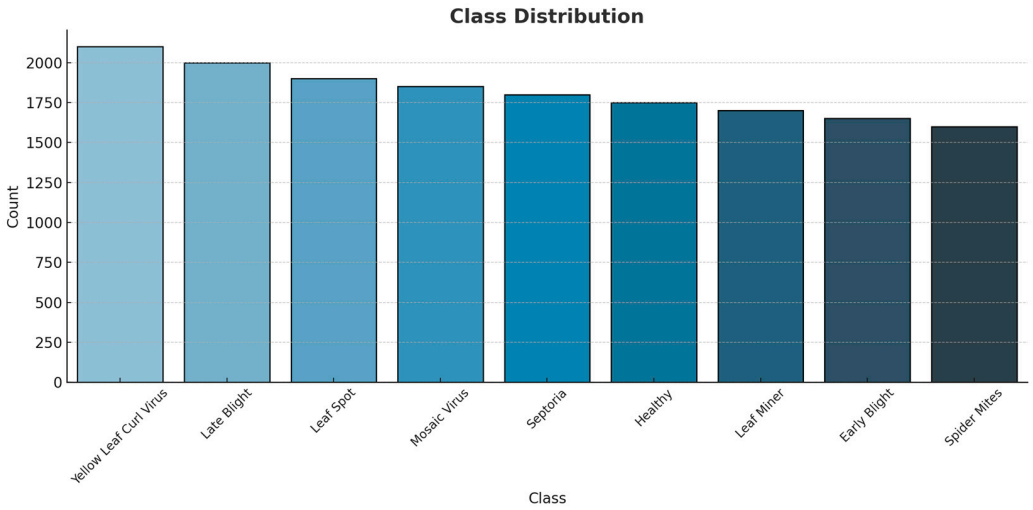


Figure 6. Examples of images from fields but with artificially created backgrounds.

A notable aspect of this dataset pertains to the origins of specific images, with a portion sourced from literary materials such as books and potential websites (see Figure 7 for visual examples).

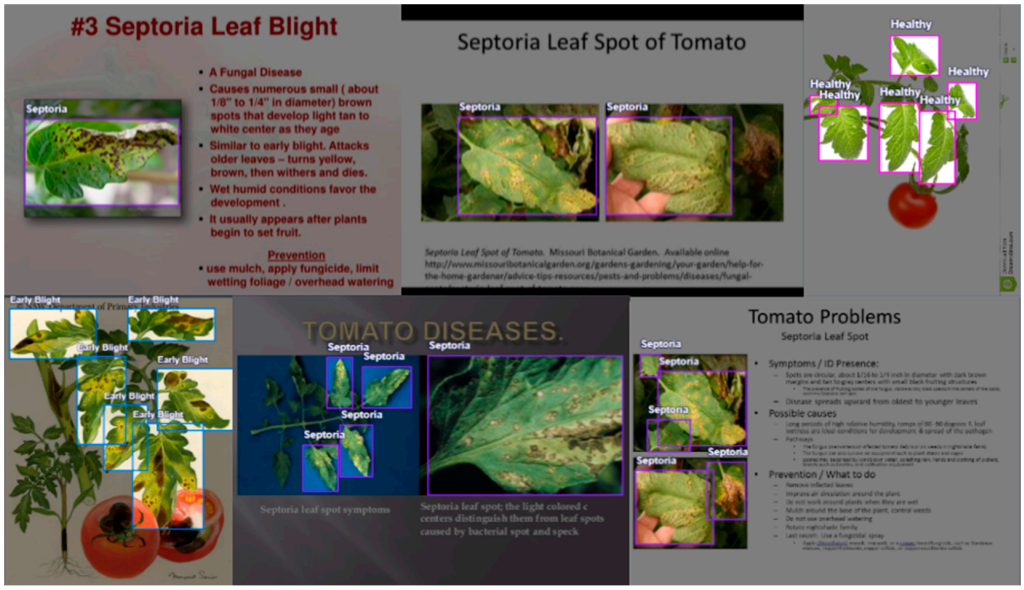


Figure 7. Examples of images that came from books and websites.

3.3. YOLOv8

In this section, we delve into YOLOv8 [17], the primary model of focus in this research. YOLOv8 represents a significant advancement in object detection technology, enhancing detection accuracy and efficiency compared to its predecessors. One notable enhancement is the incorporation of a novel neural network architecture that leverages both the feature pyramid network (FPN) [18] and the path aggregation network (PAN) [19]. This architecture enables effective feature capture across various scales and resolutions, which is crucial for detecting objects of diverse sizes and shapes.

The feature pyramid network (see Figure 8 for visual examples) component in YOLOv8 gradually reduces the spatial resolution of the input image while simultaneously increasing the number of feature channels. This process produces feature maps well-suited for detecting objects at various scales and resolutions [20]. In addition, the path aggregation network (PAN) architecture complements this by aggregating features from multiple network levels by using skip connections, which enhances the model's ability to capture fine-grained details.

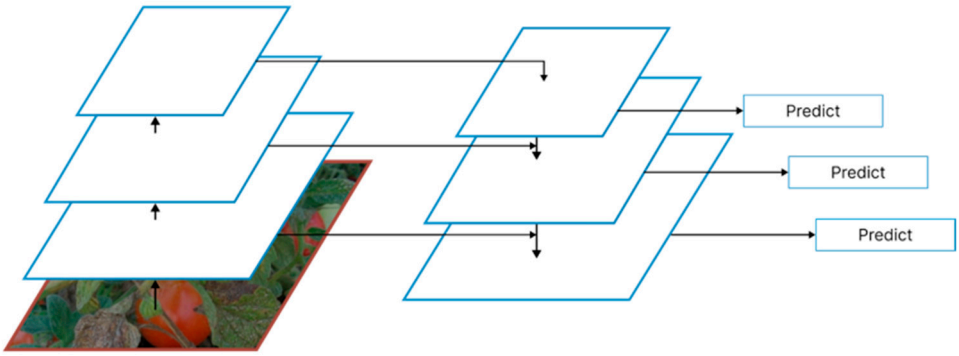


Figure 8. Feature pyramid network architecture.

Compared to the previous model, YOLOv5, which has three output heads, YOLOv8 employs an anchor-free detection mechanism that directly predicts the center of an object

instead of the offset from a known anchor box, reducing the number of box predictions and expediting the post-processing process. However, it is worth noting that YOLOv8 is slightly slower than YOLOv5 in object detection speed [21].

YOLOv8 encompasses a range of models tailored to diverse computational resources and deployment scenarios including nano, minor, medium, large, and extra-large variants. In this study, we emphasized deploying YOLOv8 on mobile devices, primarily focusing on achieving real-time performance (see Figure 9 for visual examples) for applications requiring swift object detection [22]. To align with our objectives, we exclusively employed the nano and miniature versions of the YOLOv8 model, as these versions are meticulously crafted to address the computational constraints inherent in mobile devices while ensuring optimal performance in object detection tasks.

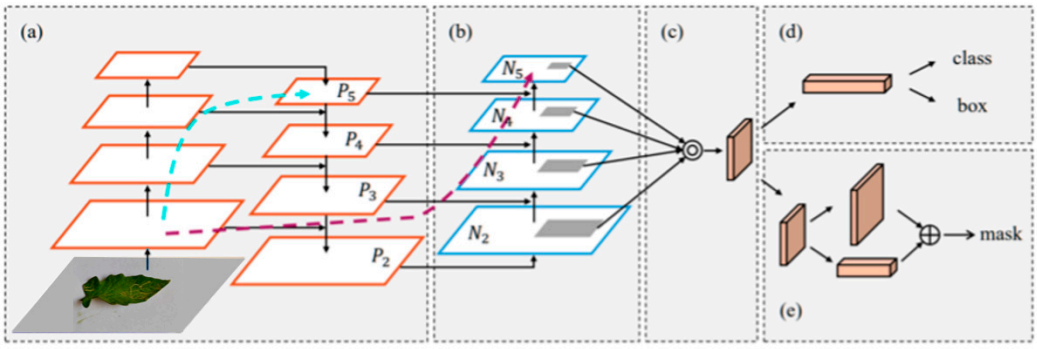


Figure 9. Illustration of the framework. (**a**) FPN backbone, (**b**) bottom-up path augmentation, (**c**) adaptive feature pooling, (**d**) box branch, and (**e**) fully-connected fusion.

Incorporating YOLOv8 into a mobile application involves key considerations such as image preprocessing, model format compatibility, and post-processing steps like non-maximum suppression (NMS). Image preprocessing can be managed using custom implementations of various operations or OpenCV.js [23]. We preferred OpenCV.js due to its efficiency and extensive capabilities.

TensorFlow.js [24] and ONNX [25] were considered for the model format, but we chose ONNX for its lightweight nature. Although TensorFlow.js offers a broad range of operations and would eliminate the need to choose between custom implementations and OpenCV.js, its use in a mobile application could have been more optimal. Since NMS is not included in either model format, we implemented it ourselves to handle the post-processing tasks.

The lightweightness of ONNX implies that the machine learning model training should be carried out using Python and its frameworks. The trained model is then saved in a format supported by the chosen framework and converted to the onyx format. After this, the steps recommended by the documentation should be followed. We describe this flow visually in Figure 9.

The general flow of detecting diseases by photo (see Figure 10) begins with the user uploading a picture from the gallery or capturing one using the camera. Subsequently, a loader is displayed while the image undergoes preprocessing in the background to match the format expected by YOLOv8. Following this, an inference is conducted, generating multiple detection boxes. A confidence threshold filters these boxes, and non-maximum suppression (NMS) is applied to retain only the relevant ones.

The uploaded image undergoes resizing using linear interpolation, followed by normalization and conversion of each channel to a format compatible with YOLOv8. The processed image is then transformed into a tensor and fed into the model, utilizing weights loaded from a file with the onyx extension. In our application setup, we used Web Assembly as the execution provider and loaded the YOLO model from the previously saved model in ONNX format with opset18.

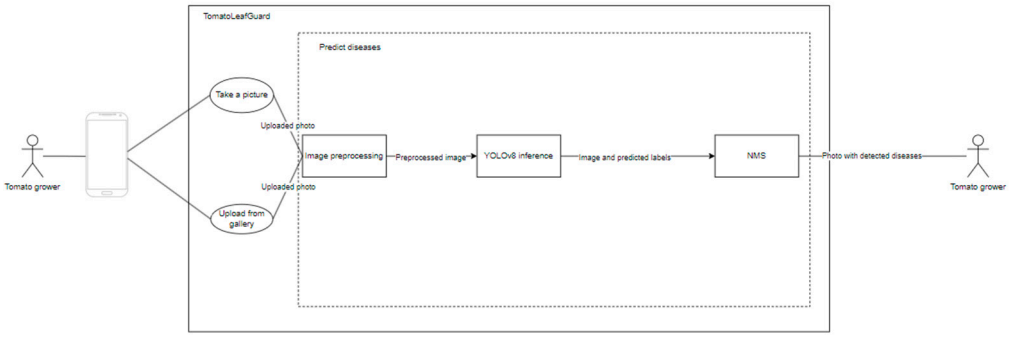


Figure 10. The general flow of predicting tomato diseases is shown in the photo on the mobile.

The output of YOLO is transformed into an array of boxes, where each box comprises coordinates, a label, and a probability. Subsequently, the array is sorted based on probabilities and filtered by a confidence score threshold. Intersection over union (IoU) is then applied to select only the relevant boxes. The detailed flow after an image is uploaded can be seen in Figure 11.

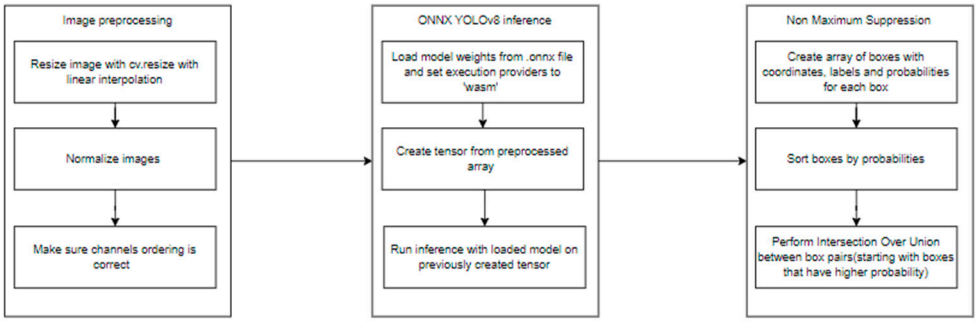


Figure 11. Detailed application process flow.

3.4. Image Preprocessing and Model Training

During the transfer of the dataset chosen in the previous section to our project on Roboflow, inadvertent removal of some images occurred, resulting in the final dataset size remaining at 4129 images. Following data acquisition, the dataset was partitioned into three subsets for subsequent model training: a training set comprising 3096 images (75%), a validation set consisting of 633 pictures (15%), and a test set comprising 400 images (10%). No preprocessing or data augmentation techniques were applied during the dataset transfer process.

As for our baseline model, we opted for Roboflow's pre-trained object detection model [26], explicitly leveraging Roboflow 3.0 Object Detection (Fast) with initial weights sourced from the COCO dataset.

Figures 12 and 13 showcase the learning curves for these YOLOv8 models. We employed COCO-pre-trained nano and small models, training them on images resized to 640 × 640 over 80 epochs. Notably, during training, we incorporated mosaic augmentation, wherein multiple images are merged into a single mosaic image to enhance model robustness and generalization.

Mosaic was not the only augmentation we applied, as we used Ultralitics CLI, which by default applies some other transformations under the hood. To break down the augmentation pipeline, we created a simplified diagram illustrating the flow of augmentation steps including those that may not be utilized by default (as depicted in Figure 14).

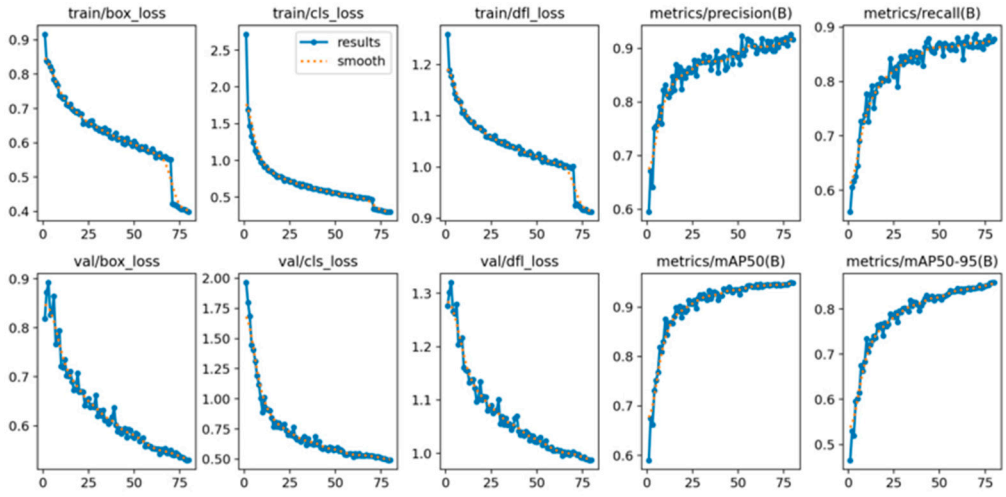


Figure 12. Sample of learning curves for YOLOv8 nano trained throughout 80 epochs.

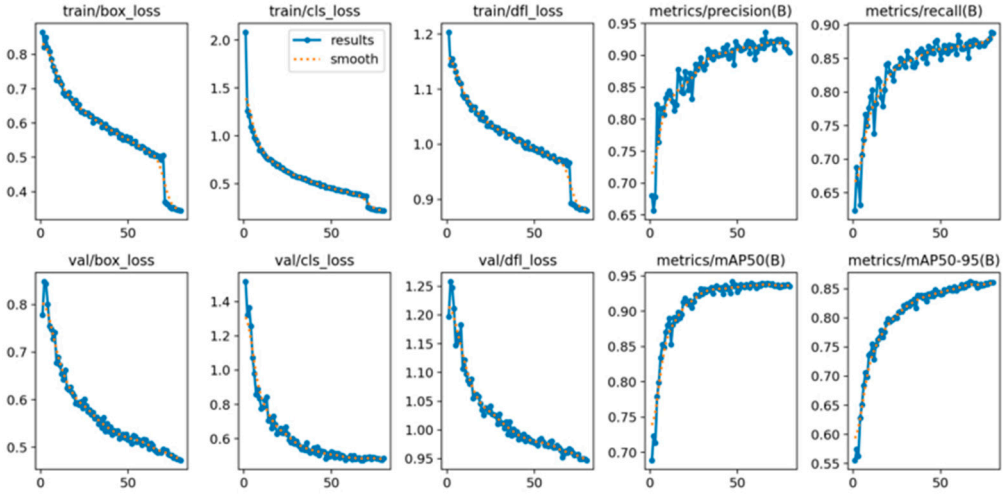


Figure 13. Learning curves for YOLOv8 small trained throughout 80 epochs.

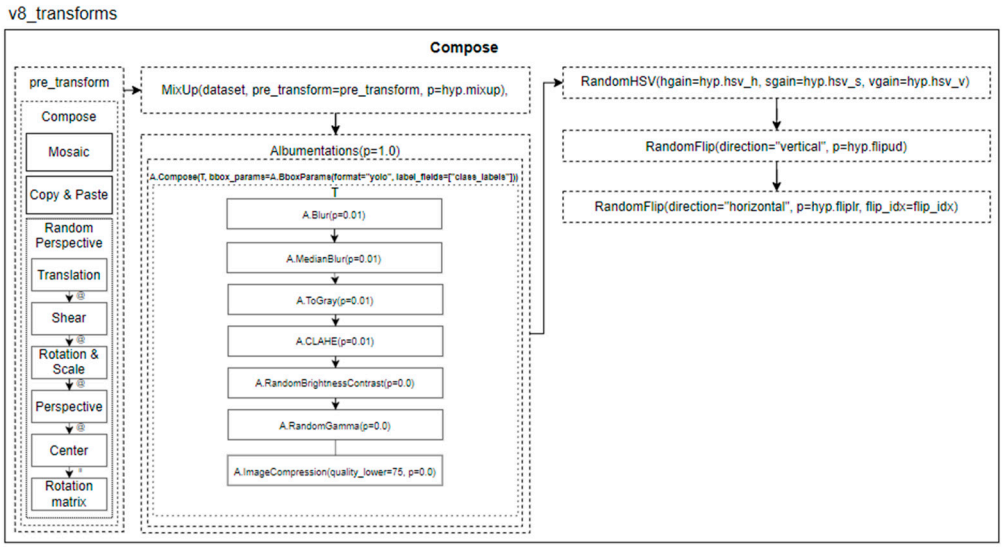


Figure 14. Detailed application process flow.

The second transformation applied by default is translation, followed by scaling with default parameters of 0.1 and 0.5, respectively. Translation introduces shifts in images, horizontally and vertically, by up to 10% of their width and height, thereby diversifying

their spatial characteristics. Similarly, scaling enables random size alterations within a range of ±50% of the original dimensions, fostering variability in image sizes and contents. The next steps include applying MixUp and Albumentations followed by hue, saturation, and brightness (value) adjustments, and flipping the images horizontally with a probability of 50%. Visual representation of the augmentation steps applied during our training is depicted in Figure 15.

Figure 15. Detailed application pipeline.

Augmentation is a part of the training pipeline and is performed during the data loader creation step, which involves data retrieval and augmentation. A simplified training pipeline is depicted in Figure 16; however, it should be noted that our diagram does not cover all aspects of the pipeline, such as repeating all augmentation steps except mosaic augmentation on the last ten epochs, performed to ensure model stability.

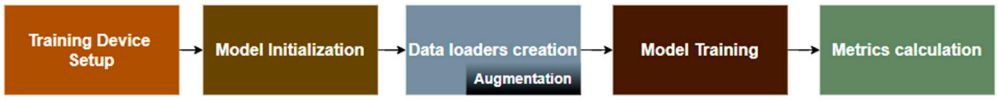


Figure 16. Simplified YOLOv8 training pipeline.

We trained an EfficientDet model with an EfficientNetB0 backbone, utilizing a slightly modified version of the Roboflow training notebook.

Before training and evaluation, we slightly modified the source code we downloaded during the initial execution of the original notebook. Our training configuration consisted of 20 epochs, with a batch size of 16 and an image size of 512 pixels. The learning rate was set to 0.0001, and validation was performed every epoch. During training, no early stopping was triggered due to a patience value of 0 and a minimum delta of 0.

During our research, we also trained another model: Faster R-CNN, which employs the ResNet50 backbone and was implemented using the detection framework. We modified the provided notebook to tailor the training process to our dataset and specific hyperparameters. We utilized the CO-CO-Detection/faster_rcnn_R_50_C4_1x.yaml model weights and set the batch size to 16 images per batch. The base learning rate was fixed at 0.001, and we integrated a warm-up strategy spanning 500 iterations. Training extended over 15 epochs, equivalent to approximately 2902 iterations. Additionally, we introduced a learning rate schedule, adjusting the learning rate at specific intervals: 1550, 1800, 2200, and 2600 iterations. This involved reducing the learning rate by a factor (Gamma solver) of 0.5 at each designated iteration.

Next, this was converted into an array of boxes containing coordinates, a label, and a probability. Subsequently, the array was sorted based on probabilities and filtered by a confidence score threshold. Intersection over union (IoU) was then applied to select only the relevant boxes. The detailed flow after an image is uploaded can be seen in Figure 11.

3.5. Results

In this section, we review the evaluation results of the trained models and the inference time of different models.

We evaluated each model on the test set, analyzing metrics such as the mean average precision (MAP) across a range of intersection over union (IOU) thresholds from 50 to 95 and at an IOU threshold of 50. Additionally, we independently assessed the average precision for each class.

Given the importance of inference speed for our specific task, we used the YOLOv8 nano version. Its mean average precision (MAP) slightly differs from the miniature version while offering superior model size and inference time, as demonstrated in Figure 17.

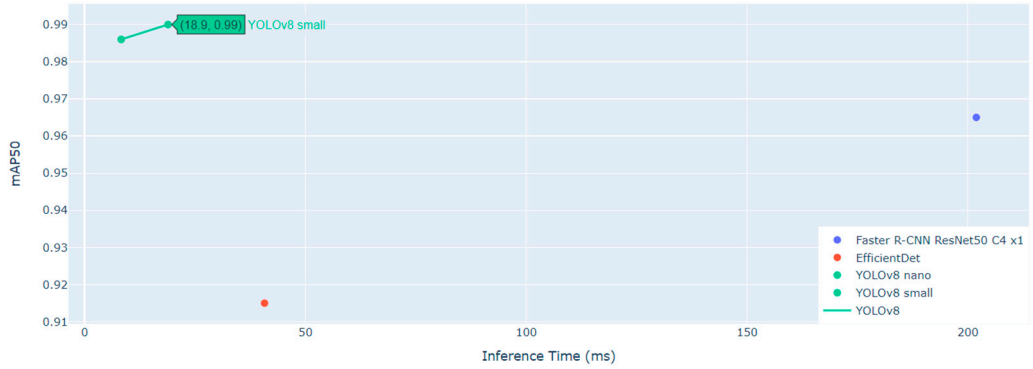


Figure 17. Model performance regarding MAP50 and inference time on the Tesla T4 GPU.

Another key evaluation metric is confusion matrices, which help identify model weaknesses and potential areas for improvement. We calculated confusion matrices across different data splits for training, testing (see Figure 18), and validation (see Figure 19). We include the validation matrix here because the class distribution was more balanced in this set compared to the test set, where the healthy and spider mites classes were underrepresented.

The models were trained using Google Colaboratory using GPUs and a Tesla T4 video card (see configuration in Table 1).

Table 1. Training and inference hardware specification.

#	Unit	Specification
1	RAM	12.67 GB
2	Processor	Intel Xeon(R) CPU @ 2.20GHz
3	Processor count	2
4	Graphic card model	Nvidia Tesla T4
5	Graphic card memory	15 GB

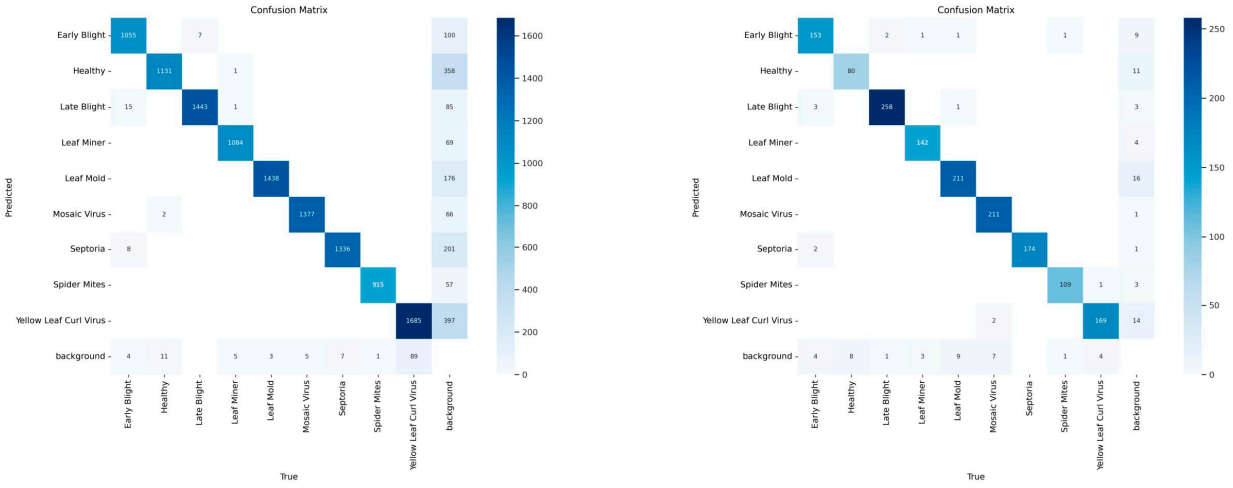
Model training can be carried out without or with the initial weights [27]. In our case, we always used model weights on the COCO dataset as the initial model weights, as this approach should reduce the number of iterations required to train the model.

The results of these evaluations are presented in Table 2 including information about each model's number of parameters (specified in millions) and model size (specified in megabytes).

YOLOv8 training was performed using the Ultralitics CLI tool [28], while EfficientDet with the EfficientNetB0 backbone was trained using a slightly modified version of the notebook file provided by Roboflow [29]. Before training, the source code of the file and repository was modified to calculate additional metrics and use a custom (created) dataset. Regarding Faster R-CNN with the ResNet50 backbone, this was trained using the Detectron2 framework, based on the Roboflow notebook [30].

In this section, we examine the results of the trained models and the inference time of the different models. We evaluated each model on the test set, analyzing metrics such as MAP at different object-over-object intersection (IOU) thresholds from 50 to 95 as well as at an IOU threshold of 50. Additionally, we estimated the average accuracy for each

class independently. The results of these evaluations are presented in Table 2, which also contains information about the number of parameters of each model (in millions) and the size of the model (in megabytes).



(**a**) (**b**)

Figure 18. Confusion matrices were calculated for the YOLOv8 nano model as: (**a**) YOLOv8 nano model on the train dataset; (**b**) YOLOv8 nano model on the test dataset.

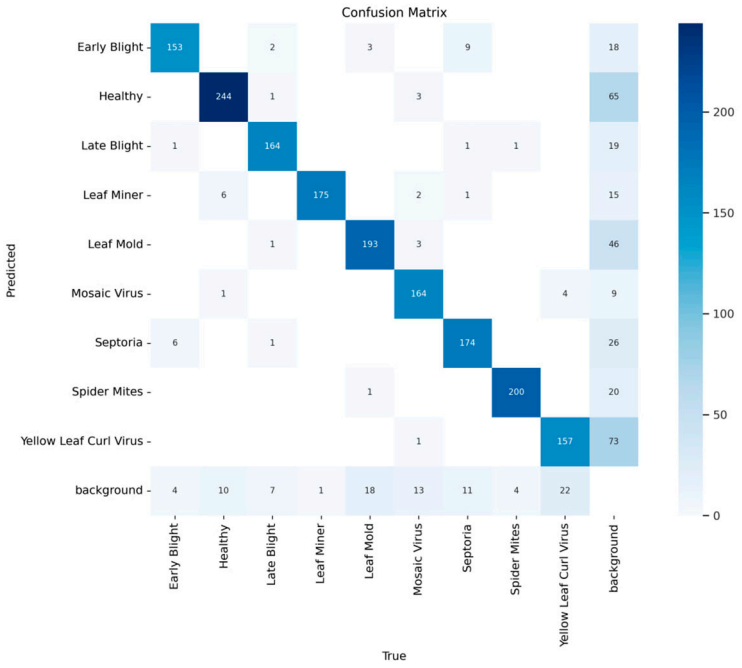


Figure 19. Confusion matrices were calculated for the YOLOv8 nano model on the validation dataset.

It should be noted that this study did not examine these aspects in more depth and leaves them for future research and further elaboration.

YOLOv8 small demonstrated the highest accuracy but had slower prediction times, making it less suitable for mobile applications. YOLOv8 nano, with an accuracy only slightly lower (1% in MAP50:95 and 0.4% in MAP50), was chosen for the mobile application due to its efficiency and reduced parameter count. It only occasionally misclassified healthy and diseased leaves as background, which is less critical than disease misclassification, and prompts users to upload additional images if necessary. Faster R-CNN ranked

third in accuracy but exhibited significantly lower speed and accuracy compared to the YOLOv8 models.

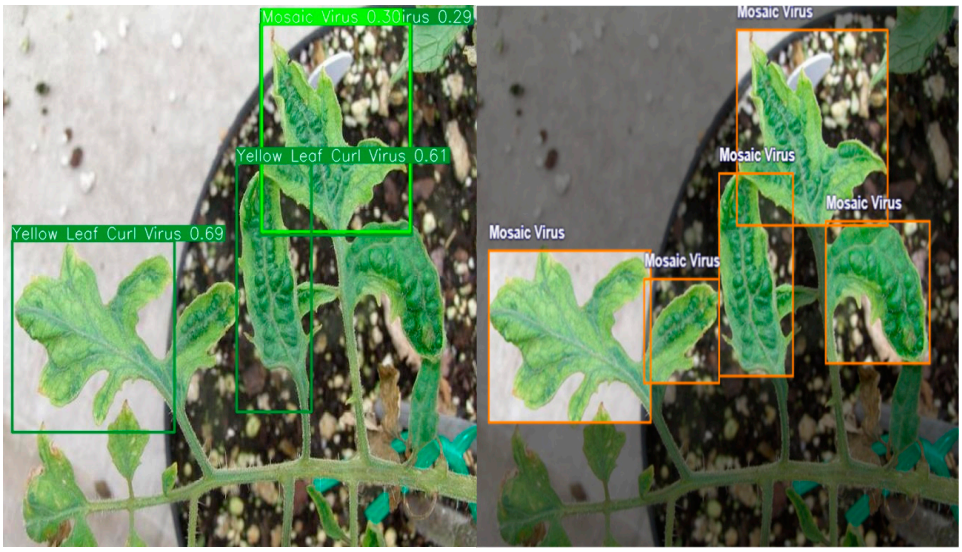
Possible future improvements could include enriching the dataset by adding more field images and creating a more balanced dataset. Another option could be to include another feature detection dataset, such as Tomato-Village, or to improve the dataset used in this study by taking into account any class imbalance that may arise during the merging step (see Figures 20–22). Another area for potential improvement is to investigate different models and their possible improvements as well as experiment with different augmentation configurations.

Table 2. Model comparison based on the test data evaluation including the expected input dimensions for each trained model.

Model	Params	Dimensions	MAP 50:95	MAP50	Model Size	Model Size (ONNX)
Roboflow 3.0 object detection(Fast)	-	512	-	0.933	-	-
Faster R-CNN (ResNet50 C4 x1)	33.8	640	0.84	0.965	252	-
EfficientDet (EfficientNetB0)	4.14	512	0.76	0.915	16.2	-
YOLOv8 nano	3.01	640	0.95	0.986	5.97	11.68
YOLOv8 small	11.14	640	0.96	0.990	21.44	42.66

As a result, YOLOv8 nano often misclassifies healthy and diseased leaves as background, which is a problem that needs attention. However, from the perspective of our task, this behavior is desirable compared to the misclassification of diseases, since the lack of results will force users to download another image if they suspect a problem with a particular plant.

In the case of false predictions, which occurred less frequently, it was found that they mostly occurred on images from the field with uneven backgrounds, which emphasizes the need to expand the dataset with more field images.

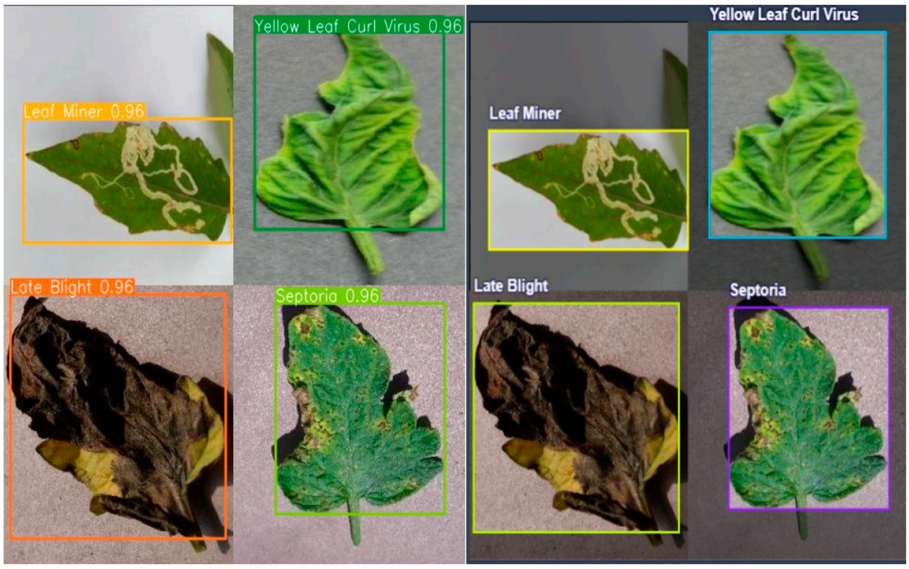


(**a**) (**b**)

Figure 20. Results of disease detection using the trained YOLOv8 nano model: (**a**) leaf photographs in field conditions and (**b**) labeled image from the created dataset.

(**a**) (**b**)

Figure 21. Results of disease detection using the trained YOLOv8 nano model: (**a**) leaf photographs in field conditions and (**b**) labeled image from the created dataset.



(**a**) (**b**)

Figure 22. Results of disease detection using the trained YOLOv8 nano model: (**a**) leaf photographs in laboratory conditions and (**b**) labeled image from the created dataset.

4. Discussion

Our comparative analysis revealed notable variations in the performance of the examined models, particularly concerning accuracy and inference speed. YOLOv8 small emerged as the top-performing model, exhibiting superior accuracy metrics compared to other models evaluated in this study. Its robust performance underscores its efficacy in accurately detecting diseases in tomato plants, making it a promising candidate for integration into our mobile-based system. However, it is essential to note that YOLOv8 nano, despite ranking slightly lower in accuracy, offers distinct advantages regarding model size and parameter count. This lightweight variant presents a compelling option for our application, considering its efficient use of computational resources and suitability for deployment in resource-constrained environments.

In contrast, the EfficientDet and Faster R-CNN models demonstrated comparatively lower accuracy and slower inference times relative to YOLOv8. Although Faster R-CNN with a ResNet50 backbone exhibited promising accuracy results, its inference time and model size may pose challenges for real-time deployment in mobile applications. Nonetheless, the potential for further enhancement by exploring alternative backbone architectures warrants consideration in future investigations. These limitations emphasize the need for the continued exploration of lightweight yet accurate architectures, particularly those that balance performance and efficiency for real-time applications.

Several avenues for future research and improvement have emerged from our study findings. Firstly, enriching the dataset by incorporating additional images from diverse geographical regions and environmental conditions could enhance the robustness and generalizability of the trained models. Furthermore, addressing class imbalances within the dataset, such as integrating complementary object detection datasets like the Tomato Leaf Disease Dataset, may yield more comprehensive and representative training data.

Additionally, exploring different model architectures and augmentation strategies could further improve the accuracy and inference speed.

5. Conclusions

In conclusion, YOLOv8 small delivered top-tier results in terms of accuracy, while YOLOv8 nano ranked second with a 1% difference in MAP50:95 and 0.4% in MAP50. Meanwhile, the nano version boasts nearly a four times smaller model size and parameter count than the minor variant, making it a better fit for our application. Additionally, EfficientDet and Faster R-CNN displayed lower accuracy and slower inference times than YOLOv8. However, Faster R-CNN with a ResNet50 backbone reached a MAP50 of 96.5%, with the potential for further improvement using different backbones. Nonetheless, its inference time and model size may not be suitable for mobile applications. Future enhancements may enrich the dataset by adding more images from the field and creating a more balanced dataset. Another option could be incorporating a different object detection dataset, such as relabeling Tomato-Village, either to use it independently or to enhance the dataset employed in this research while managing any class imbalance that may emerge during the merge. Other avenues for potential improvement are an investigation of various models and their possible enhancements as well as experimenting with different augmentation configurations.

Author Contributions: Conceptualization, N.K. and O.B.; methodology, O.B.; software, N.K.; validation, N.S. and O.B.; formal analysis, O.B.; investigation, N.K.; resources, O.B.; data curation, N.M.; writing—original draft preparation, N.K.; writing—review and editing, O.B. and N.M.; visualization, O.B.; supervision, N.S.; project administration, N.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the following links: https://github.com/nazarkohut/tomato-disease-detection-train; https://www.kaggle.com/datasets/emmarex/plantdisease; https://www.kaggle.com/datasets/cookiefinder/tomato-disease-multiple-sources; https://github.com/mamta-joshi-gehlot/Tomato-Village.

Acknowledgments: The authors would like to thank the Armed Forces of Ukraine for providing the security to perform this work. This work was only possible because of the resilience and courage of the Ukrainian Army.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chen, Z.; Wu, R.; Lin, Y.; Li, C.; Chen, S.; Yuan, Z.; Chen, S.; Zou, X. Plant Disease Recognition Model Based on Improved YOLOv5. *Agronomy* **2022**, *12*, 365. [CrossRef]
2. Soeb, M.J.A.; Jubayer, M.F.; Tarin, T.A.; Al Mamun, M.R.; Ruhad, F.M.; Parven, A.; Mubarak, N.M.; Karri, S.L.; Meftaul, I.M. Tea leaf disease detection and identification based on YOLOv7 (YOLO-T). *Sci. Rep.* **2023**, *13*, 6078. [CrossRef]
3. Liu, J.; Wang, X. Early recognition of tomato gray leaf spot disease based on MobileNetv2-YOLOv3 model. *Plant Methods* **2020**, *16*, 83. [CrossRef] [PubMed]
4. Wang, Y.; Wang, Y.; Zhao, J. MGA-YOLO: A lightweight one-stage network for apple leaf disease detection. *Front. Plant Sci.* **2022**, *13*, 927424. [CrossRef] [PubMed]
5. Yang, G.; Wang, J.; Nie, Z.; Yang, H.; Yu, S. A Lightweight YOLOv8 Tomato Detection Algorithm Combining Feature Enhancement and Attention. *Agronomy* **2023**, *13*, 1824. [CrossRef]
6. Tan, M.; Le, Q.V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114. Available online: https://proceedings.mlr.press/v97/tan19a.html (accessed on 1 November 2024).
7. Bistroń, M.; Piotrowski, Z. Optimization of Imaging Reconnaissance Systems Using Super-Resolution: Efficiency Analysis in Interference Conditions. *Sensors* **2024**, *24*, 7977. [CrossRef]
8. Hughes, D.; Salathé, M. An Open-Access Repository of Images on Plant Health to Enable the Development of Mobile Disease Diagnostics. 2015. Available online: https://arxiv.org/ftp/arxiv/papers/1511/1511.08060.pdf (accessed on 1 November 2024).
9. Ahmad, A.; Saraswat, D.; El Gamal, A. A survey on using deep learning techniques for plant disease diagnosis and recommendations for developing appropriate tools. *Smart Agric. Technol.* **2023**, *3*, 100083. [CrossRef]
10. Lespinats, S.; Colange, B.; Dutykh, D. *Nonlinear Dimensionality Reduction Techniques: A Data Structure Preservation Approach*; Springer International Publishing: Berlin/Heidelberg, Germany, 2022. [CrossRef]
11. Thapa, R.; Zhang, K.; Snavely, N.; Belongie, S.; Khan, A. Plant Pathology 2021-FGVC8. Kaggle. 2021. Available online: https://kaggle.com/competitions/plant-pathology-2021-fgvc8 (accessed on 1 November 2024).
12. Kaeser-Chen, C.; Pathology, F.; Maggie; Dane, S. Plant Pathology 2020-FGVC7. Kaggle. 2020. Available online: https://kaggle.com/competitions/plant-pathology-2020-fgvc7 (accessed on 1 November 2024).
13. Arun Pandian, J.; Gopal, G.; Huang, M.-L.; Chang, Y.-H. Tomato Disease Multiple Sources. Kaggle. 2022. Available online: https://www.kaggle.com/dsv/4270691 (accessed on 1 November 2024).
14. Geoffroy, H.; Berger, J.; Colange, B.; Lespinats, S.; Dutykh, D. The use of dimensionality reduction techniques for fault detection and diagnosis in an AHU unit: Critical assessment of its reliability. *J. Build. Perform. Simul.* **2023**, *16*, 249–267. [CrossRef]
15. Gehlot, M.; Saxena, R.K.; Gandhi, G.C. "Tomato-Village": A dataset for end-to-end tomato disease detection in a real-world environment. *Multimed. Syst.* **2023**, *29*, 3305–3328. [CrossRef]
16. Bryan. Tomato Leaf Disease Dataset. Roboflow Universe. 2023. Available online: https://universe.roboflow.com/bryan-b56jm/tomato-leaf-disease-ssoha (accessed on 15 April 2024).
17. Wang, C.-Y.; Bochkovskiy, A.; Liao, H.-Y.M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv* **2022**, arXiv:2207.02696.
18. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *arXiv* **2015**, arXiv:1512.03385. [CrossRef]
19. Noyan, M.A. Uncovering bias in the PlantVillage dataset. *arXiv* **2022**. [CrossRef]
20. Reis, D.; Kupec, J.; Hong, J.; Daoudi, A. Real-Time Flying Object Detection with YOLOv8. *arXiv* **2023**, arXiv:2305.09972. [CrossRef]
21. Solawetz, J. How to Train Detectron2 on Custom Object Detection Data. Roboflow Blog. 2020. Available online: https://blog.roboflow.com/how-to-train-detectron2/ (accessed on 1 November 2024).
22. Jocher, G.; Chaurasia, A.; Qiu, J. "Ultralytics YOLO" (Version 8.0.0). Ultralytics. 2023. Available online: https://github.com/ultralytics/ultralytics (accessed on 1 November 2024).
23. Zhang, H.; Cisse, M.; Dauphin, Y.N.; Lopez-Paz, D. Mixup: Beyond Empirical Risk Minimization. 2018. Available online: https://arxiv.org/pdf/1710.09412.pdf (accessed on 1 November 2024).
24. Smilkov, D.; Thorat, N.; Assogba, Y.; Yuan, A.; Kreeger, N.; Yu, P.; Zhang, K.; Cai, S.; Nielsen, E.; Soergel, D.; et al. TensorFlow.js: Machine Learning for the Web and Beyond. 2019. Available online: https://arxiv.org/pdf/1901.05350.pdf (accessed on 1 November 2024).
25. ONNX Runtime Developers. ONNX Runtime (Version: 1.17.0). 2021. Available online: https://onnxruntime.ai/ (accessed on 1 November 2024).
26. Taheri, S.; Vedienbaum, A.; Nicolau, A.; Hu, N.; Haghighat, M.R. OpenCV.js: Computer vision processing for the open web platform. In Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18), Amsterdam, The Netherlands, 12–15 June 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 478–483. [CrossRef]
27. Kohut, N.; Kohut, A.; Basystiuk, O. Model Weights. 2024. Available online: https://github.com/nazarkohut/tomato-disease-detection-models (accessed on 1 November 2024).

28. Ahmed, N.; Jahir, I.R.; Rashedul, M.I. Enhanced Human Activity Recognition Based on Smartphone Sensor Data Using Hybrid Feature Selection Model. *Sensors* **2020**, *20*, 317. [CrossRef] [PubMed]
29. Lin, T.-Y.; Dollr, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature Pyramid Networks for Object Detection. 2017. Available online: https://arxiv.org/pdf/1612.03144.pdf (accessed on 1 November 2024).
30. Carranza-Garca, M.; Torres-Mateo, J.; Lara-Bentez, P.; Garca-Gutirrez, J. On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data. *Remote Sens.* **2021**, *13*, 89. [CrossRef]