Optimizing Disease Detection Models in School Greenhouses: An AI and IoT-Based Approach for Smart Agriculture

Jihene Rezgui[*$], Léane Lafleur-Hébert[*], Enric Soldevila[*$], Yousra Azmour[*], Matéo Tardy[*]
Laboratoire **R**echerche **I**nformatique **M**aisonneuve (LRIMa) [*], University of Montreal[$]
Montreal, Canada
jrezgui@cmaisonneuve.qc.ca

***Abstract* – In recent years, the integration of Internet of Things technologies and machine learning models in agriculture has significantly advanced smart farming practices. This paper presents our research on enhancing disease detection in tomato plants within a greenhouse environment using advanced object detection models. Collaborating with the University of Montreal greenhouse, we developed a realistic dataset comprising images from both the PlantVillage repository and over 1,900 manually labeled leaf images taken from the greenhouse. Using this dataset, we evaluated and compared three object detection models: Faster R-CNN, YOLOv10, and SSD, to accurately detect and classify tomato leaf diseases. Our approach enables us to train a model on a more realistic set of images, facilitating automatic and earlier disease detection for farmers. Our results show that the Faster R-CNN model with a ResNet-50 backbone and Feature Pyramid Network is the most effective for detecting diseased leaves, achieving a mAP50 score of 93.13%.**

Keywords: Agricultural IoT, Machine Learning, Smart Farming, Disease Detection, Greenhouse Monitoring.

I. Introduction

The rapid advancement of Internet of Things (IoT) technologies [1], coupled with machine learning (ML) methodologies, has ushered in a new era of smart agriculture, where precision farming techniques enhance productivity and sustainability [2]. Smart farming systems leverage IoT sensors and data-driven models to monitor crop health, optimize resource usage, and automate agricultural processes. Among the most critical applications of these technologies is the early detection and management of plant diseases, which can significantly impact crop yields and quality [3]. Tomato plants, a staple in global agriculture, are particularly vulnerable to a variety of diseases that can rapidly spread in greenhouse environments. Traditional disease detection methods rely on manual inspection, which is time-consuming, labor-intensive, and prone to human error. This highlights the need for automated, reliable, and efficient solutions to identify and manage plant diseases in real-time. In our previous work [4], we explored the integration of IoT and ML technologies in smart agriculture, focusing on the detection of plant diseases in a real greenhouse environment. We augmented the Plant Village dataset from Kaggle [5] and compared the accuracy of YOLOv8 and Faster-RCNN models. This paper builds on that research by further enhancing the dataset with new images, benchmarking additional object detection models, and adding disease classification capabilities. We focused on three state-of-the-art object detection models: Faster R-CNN, YOLOv10, and SSD, each offering unique advantages for detecting plant diseases. Our primary goal is to determine the most effective model for this specific application, considering factors such as detection accuracy, training speed, prediction speed, and the ability to handle the complex and varied conditions found in a greenhouse. Our research was conducted in collaboration with the Sustainability Unit of the University of Montreal (UdeM), where we utilized their industrial school greenhouse, referred to as the *UdeM greenhouse* in this paper, to collect a robust dataset of tomato leaves, both healthy and diseased (see **Fig. 1**), combined with images from the PlantVillage dataset.

Fig.1. School UdeM greenhouse

Our contributions in this paper can be summarized as follows: **(1)** we selected over 8,500 images from the PlantVillage dataset and categorized them using Roboflow; **(2)** we framed and labeled 1,900 tomato leaves with their diseases from approximately 50 images taken in the UdeM greenhouse; **(3)** we augmented our dataset using various transformations; **(4)** we trained and benchmarked three object detection models on our dataset; **(5)** we fine-tuned the hyperparameters of these models to compare their performances; and (**6**) we discussed the results of the three models through extensive simulations.

Outline: Section II gives a brief overview of similar research done in the field of tomato leaf disease detection. Section III presents the transformation and preparation of the data used to train the models. Section IV explains our choices of object detection models. Section V shows our results. Finally, section VI concludes the paper.

II. Related work

Numerous researchers have presented their solutions for optimizing disease detection models in tomatoes [6-9]. Notably, studies involving Faster R-CNN have explored various Convolutional Neural Network (CNN) models, such as

Region-based CNN (R-CNN) and Fast R-CNN, to detect diseases on tomato leaves using the PlantVillage dataset [7]. Although the PlantVillage dataset is diverse, it does not accurately represent real-life agricultural conditions due to its lack of comprehensive environmental data. To address this, we collected and labeled images from the UdeM greenhouse to augment the PlantVillage dataset. For SSD, a notable approach focuses on detecting tomato leaf defects by categorizing leaves into two broad classes: "healthy" and "unhealthy," with the latter further divided into "Bacterial Spot" and "Yellow Leaf Curl Virus." This work [8] utilizes the MobileNetV2 backbone of SSD for model development. While effective, this method's limited disease categories do not fully capture the diversity of real-life agricultural conditions. In contrast, our dataset includes ten distinct categories—"Healthy Tomato," "EarlyBlight," "BacterialSpot," "LeafMold," "Mosaic Virus," "SeptoriaLeafSpot," "SpiderMites," "TargetSpot," "TomatoLateBlight," and "YellowLeafCurlVirus"—making our system more applicable to real-world scenarios.

To the best of our knowledge, there are no papers using versions of YOLO more recent than YOLOv8 [9] for tomato leaf disease detection. Therefore, we are among the first to experiment with YOLOv10 in this context.

III. Case study: UdeM School Greenhouse

We conducted our research in an industrial school greenhouse. This section highlights the key differences to our previous work [4], focusing on the improvements and refinements we have made. The section is organized into two main subsections:

(1) **Dataset preparation** subsection covers the methods and processes involved in gathering the data for our dataset, which is used to train the models later. It describes the original structure of the dataset, the modifications made to its structure, and the image transformations applied to enhance it.

(2) **The models used** subsection briefly outlines the different models employed in our approach, emphasizing the changes we made compared to previous work to better address our specific needs. It discusses these modifications in more detail, including the theoretical aspects, in the next section.

A. Dataset preparation

Similar to our previous work, the initial foundation of our dataset is composed of images from the Kaggle repository named PlantVillage. We began by selecting 10 directories from this repository, which include various images of tomato leaves, either healthy or exhibiting one of the 9 different types of diseases. Previously, our dataset was categorized into only three groups: tomato leaf, sick tomato leaf, and other leaf. In this new version, we have removed the "other leaf" category and expanded the dataset into 10 specific categories. This allows the models to not only identify diseased leaves but also determine the type of disease affecting them. The categories now include healthy tomato leaf, leaf mold, early blight, septoria leaf spot, bacterial spot, spider mites, mosaic virus, yellow leaf curl virus, target spot, and late blight.

However, the images of PlantVillage feature a single leaf per image, which is not representative of a greenhouse environment where tomato plants often have clusters of leaves grouped together and overlapping. To address this issue and make the dataset more relevant to our needs, we augmented it with images taken directly from the UdeM greenhouse, manually drawing bounding boxes around the tomato leaves and labeling them according to their respective disease categories. This adjustment made the dataset reflect the conditions found in a greenhouse, providing a more comprehensive and realistic training set for our models. At the time of this paper, the distribution of images from each source is illustrated in the accompanying graphs. The first graph (**Fig.2**) compares the quantity of diseased leaves between the PlantVillage dataset, and the images collected from the UdeM greenhouse. The second graph (**Fig. 3**) represents the quantity and percentage of leaves from each source, showing how the leaf categories are distributed between the PlantVillage dataset and the UdeM greenhouse images.

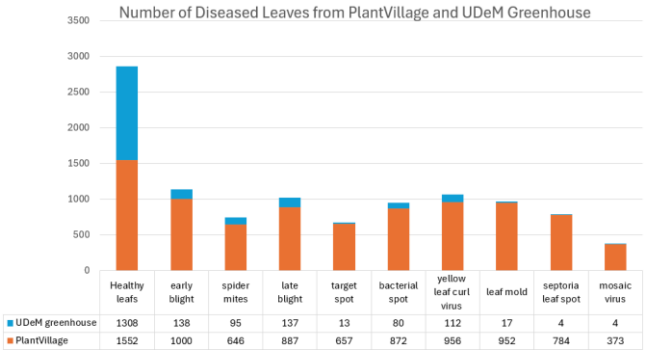

Fig.2. Number of Diseased Leaves from PlantVillage and UdeM Greenhouse

We acknowledge that there is currently an imbalance in the dataset. To address this imbalance and enhance the diversity of the dataset, we will continuously add more images from the UdeM greenhouse using a Raspberry Pi camera. This ongoing effort aims to create a more balanced and representative dataset, improving the robustness and accuracy of our models. We utilized a comprehensive set of image transformations to augment our dataset, enhancing its quality, diversity and realism. The transformations applied are detailed in **Table 1.** below.

Flip	horizontal and vertical flip
Crop	Minimum zoom of 0% and maximum zoom of 24%
Rotation	Between -30° and +30°
Shear	±27° horizontal and ±29° vertical
Brightness	Adjustments between -36% and +0%
Blur	Up to 1.2 pixels
Mosaic	Applied to combine multiple images into one

Table 1. Transformations applied to dataset

Also, for each training image, three additional images were generated by randomly applying various transformations. This approach increases the dataset size. Finally, the dataset was divided into three subsets for training, train (82%), test (8%), valid (8%), as illustrated in **Fig. 4**. To maintain consistency and ensure comparability with our previous research, we aimed to keep the dataset split and structure as similar as possible.

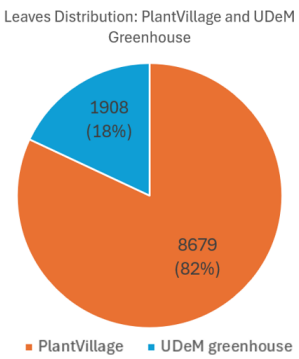

Fig.3. Leaves Distribution: PlantVillage and UdeM Greenhouse

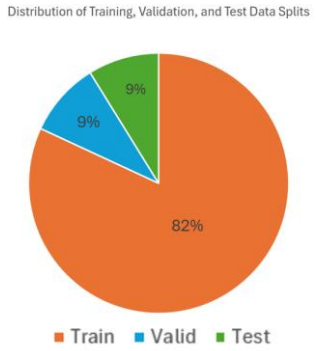

Fig.4. Distribution of Training, Validation and Test Data Splits

B. The Models

For the object detection models used in our study, we decided to implement 3 different models to determine which one would be most beneficial in a real greenhouse context. These models will be evaluated and compared based on several criteria, including their ability to accurately identify smaller objects, detection speed, and overall precision. We chose these criteria because, in a greenhouse environment, it is crucial for the model to accurately detect small objects like leaves, which can vary significantly in size.

The first model will be the same as in [4], which is Faster R-CNN with ResNet as the backbone. In this study, we continue to use ResNet-50 with a Feature Pyramid Network (FPN) as a reference point, like the work in [4], but we also introduce and compare two alternative backbones: ResNet-101 and ResNet-152, both without FPN. We decided to compare these models to determine which backbone would provide better precision for our research moving forward.

The second model will be YOLOv10, an upgraded version of YOLOv8. This newer version shows improvements in detecting smaller objects due to key advancements, such as dual label assignments [10] that boost accuracy and enhanced downsampling techniques that preserve fine details.

The third model in our study, is SSD (Single Shot Detector), specifically using two variants: ssd300_vgg16 and ssdlite320_mobilenet_v3_large. We selected these SSD models to explore how well they balance speed and accuracy, especially in detecting the smaller, more detailed features of tomato leaves that are crucial for identifying diseases.

IV. Our Greenhouse AI models

A. Faster-RCNN

The first model is Faster R-CNN (Region-based Convolutional Neural Network). In our previous work, we used Faster R-CNN with a ResNet-50 version 2 backbone combined with a Feature Pyramid Network (FPN) to achieve high precision in object detection. To explore whether an alternative backbone could be better for the model's precision, we decided to evaluate ResNet-101 and ResNet-152 backbones, this time without FPN, and compare them against the ResNet-50 with FPN used in our earlier study. The ResNet-50 with FPN architecture is known for balancing accuracy and computational efficiency by leveraging multi-scale feature maps. In contrast, ResNet-101 and ResNet-152 offer deeper networks compared to ResNet-50, with 101 and 152 layers, respectively, compared to 50 layers. For a fair comparison, all three models were trained on the same dataset, with identical parameters: 25 epochs, a batch size of 8, and an image size of 128.

In the analysis of the models' performance, we observe two key metrics: mAP50 and mAP50:90. The first metric, mAP50, reflects the accuracy of object detection at an IoU threshold of 50%, while the second, mAP50:95, measures accuracy across a range of IoU thresholds from 50% to 95%. The ResNet-50 with FPN model demonstrates the smallest gap between these two metrics, suggesting it maintains consistent performance in both detection accuracy and localization precision. This indicates that the ResNet-50 with FPN model is particularly effective in balancing both accurate object detection and precise localization. Conversely, models using ResNet-101 and ResNet-152 show a more pronounced gap between the metrics. This gap implies that, although these models may achieve higher overall detection accuracy, their ability to perform well at stricter localization thresholds is less robust compared to ResNet-50 with FPN.

Having concluded that ResNet-50 with FPN is the most suitable model for our study, we will now train it to use different parameters from those used to compare it with ResNet-101 and ResNet-152. This will allow us to optimize its performance and effectively compare it with the other two models, YOLOv10 and SSD. For the result section we used a batch size of 6, an image size of 640, and 50 epochs. However, the training process stopped early at epoch 17 due to early stopping, which was activated after the validation performance failed to improve for 10 consecutive epochs.

B. YOLOv10

The second model is YOLO (You Only Look Once). In our previous research paper, we used YOLOv8. Since the release of that paper, researchers at Tsinghua University built YOLOv10 using the Ultralystics Python package, the same one as YOLOv8.

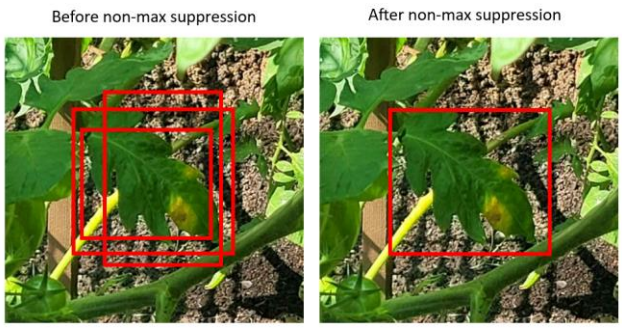

Fig.5. Example of NMS technique

There are a few differences between the version 8 and version 10 that made us choose version 10 for this research. For example, YOLOv8 uses Non-Maximum Suppression (NMS) (**Fig.5**) as a post-processing step to remove duplicate bounding boxes around detected objects. This ensures that each object is represented by only one bounding box and not multiple. However, the use of NMS adds computational cost during the training and inference. During training in YOLOv8, for each object in the image, there's multiple potential bounding boxes assigned to it, this is called the one-to-many assignment strategy. Then once the model is making a prediction, they use NMS to filter out all the redundant bounding boxes, so that the most accurate one is the one kept. Furthermore, YOLOv10 introduces a new approach that eliminates the need for NMS, hence the name "NMS-free." This is achieved through a new training strategy that employs dual label assignments and a consistent matching metric.

The introduction of dual label assignments in YOLOv10 combines the best of both one-to-many and one-to-one matching during training, as illustrated in this schema. The first head is the one-to-many assignment is the same as the one used in YOLOv8, making multiple bounding boxes. The second head is the one-to-one on the contrary, making one box per object. Both heads contribute to the model training and to make sure that they are harmonized YOLOv10 uses a uniform matching metric ($m(\alpha,\beta) = s \cdot p^{\alpha} \cdot IoU(\hat{b} \cdot b)^{\beta}$) [10]. When it's time to make a prediction, only the one-to-one will be used to make it only one box, no need for the post-processing NMS.

Another example of change is the downsampling technique. Downsampling refers to the process of reducing the spatial dimensions of feature maps in a convolutional neural network while also adjusting the number of channels. In YOLOv8, this is achieved using standard 3x3 convolutions with a stride of 2, which simultaneously reduces the height and width of the feature maps by half and increases the number of channels. This approach, while effective, performs both spatial reduction and channel transformation in a single step, which can be computationally intensive.

In contrast, YOLOv10 introduces a more efficient method known as spatial-channel decoupled downsampling. This technique separates the spatial and channel operations into two distinct steps. First, it uses a pointwise convolution (1x1) to handle channel transformations without affecting the spatial dimensions. Then, a depthwise convolution (3x3 with stride 2) is employed to reduce the spatial dimensions. This separation allows for more efficient computation and better preservation of fine details in the feature maps. For our case, this means YOLOv10 can more effectively detect smaller objects, such as leaves in a greenhouse, by retaining crucial details during the downsampling process.

For the result section we use the model YOLOv10-B. We train this model with a batch size of 16, 50 epochs and 640 as the image size.

C. SSD (Single Shot Detector)

The SSD (Single Shot Detector) is our newly added model to our research. The SSD adds valuable diversity to our study on tomato leaf disease detection. SSD is renowned for its real-time detection capabilities, which makes it particularly advantageous for applications requiring swift and efficient processing. This model stands out for its ability to perform object localization and classification in a single pass through the network, similar to YOLO [11]. This approach significantly reduces computational complexity compared to two-stage detectors like Faster R-CNN.

In the SSD architecture, two main components work together to perform object detection. First is the backbone, it's a pre-trained convolutional neural network, such as MobileNetV3 in ssdlite320_mobilenet_v3_large or VGG-16 in ssd300_vgg16, that serves as a feature extractor. It processes the input image and generates feature maps at various levels of detail. These feature maps capture important information about the objects in the image, such as their shapes, textures, and locations.

Second is the SSD head that is attached to the feature maps produced by the backbone. It consists of additional convolutional layers designed to predict the presence of objects at different scales. The SSD head generates bounding box coordinates and class scores for each object directly from these feature maps. By applying this head to multiple feature maps, SSD can detect objects of various sizes and positions within a single forward pass through the network. **Fig.6** displays the first few layers (white boxes) that are the backbone and the last few layers (blue boxes) that are the SSD head.

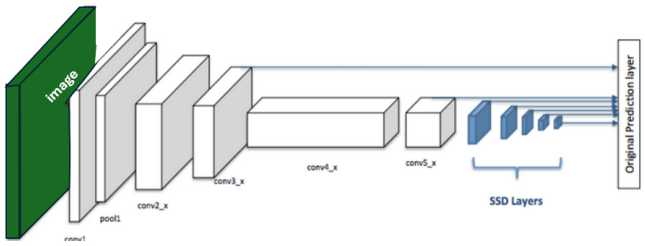

Fig.6. Modified image of a design of a convolutional neural network integrated with an SSD detector. [11]

While the combination of the backbone and SSD head allows for efficient object detection, detecting small objects like tomato leaves presents unique challenges. The smaller the object, the more grid cells are needed to effectively detect it. This is because smaller objects occupy less space in the image, so a finer grid allows the model to capture more detailed information and better localize these small objects. In SSD, this is achieved by using higher-resolution feature maps that provide more grid cells, enabling the detection of smaller

objects like individual tomato leaves. Therefore, in the context of the UdeM greenhouse we have to use higher-resolution feature maps concerning the tomato leaves.

We used a default configuration for bounding boxes, which are also known as anchor boxes. These default boxes are predefined at multiple scales and aspect ratios for each feature map cell in the SSD architecture. They are crucial for the SSD model to predict object locations and classes efficiently.

In our study, we employed two variants of the SSD model to compare their performance in detecting diseases in tomato leaves: ssd300_vgg16 and ssdlite320_mobilenet_v3_large.

The first model, ssd300_vgg16, utilizes the VGG-16 backbone, known for its simplicity and effectiveness in feature extraction. This model is designed to provide a good balance between detection accuracy and computational efficiency, making it suitable for scenarios where moderate speed and high precision are required.

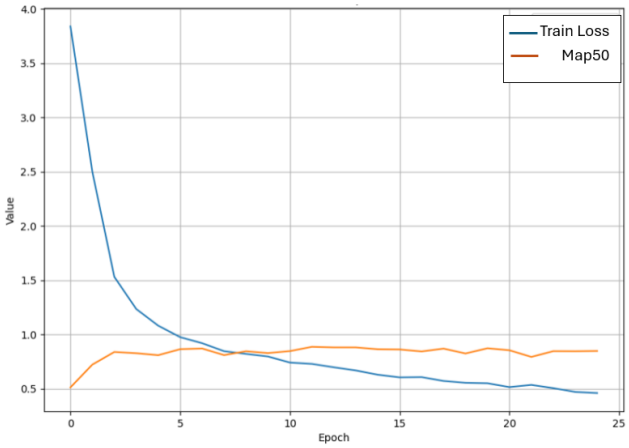

Fig.7. The Training loss and the mAP50 of ssdlite320_mobilenet_v3_large over the epochs.

On the other hand, we also used ssdlite320_mobilenet_v3_large, which incorporates the MobileNetV3 backbone [12]. This variant is optimized for mobile and embedded devices, offering a lighter, faster alternative without compromising too much on accuracy. The ssdlite320_mobilenet_v3_large model is particularly advantageous in real-time applications, where rapid detection is crucial, such as in a field setting where quick identification of diseased leaves can significantly impact crop management decisions.

By comparing these two models, we aimed to evaluate how different backbones and SSD configurations affect the detection of small objects, like tomato leaves, and determine which approach offers the best trade-off between speed and accuracy for our specific application.

For a fair comparison, the two models were trained on the same dataset, with identical parameters: 25 epochs, a batch size of 32, and an image size of 640.

We chose ssdlite320_mobilenet_v3_large for its high mAP50 (0.8493) and low training loss (0.4619) as shown in **Fig.7**. In comparison to the ssd300_vgg16 where its mAP50 was 0.7438 and its training loss was 2.3897. Furthermore, **Fig.8** shows a sudden drop in the mAP50 going from 0.70 to 0.10 and a sudden gain in the training loss indicating that the ssd300_vgg16 has some stability issues during training when exposed to randomness caused by our data shuffling which adds to our choice of not picking the ssd300_vgg16 model.

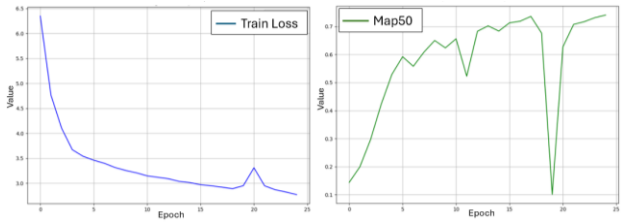

Fig.8. The training loss and the mAP50 over the epochs of ssd300_vgg16.

V. Simulations Results

The following subsections will present the results of our research for the 3 models.

A. Faster R-CNN

For the model using ResNet-50 with FPN, the average mAP50 score across all the epochs came to 92.13%, and the average mAP came to 90.83%. The graph (**Fig.9**) below illustrates the different components of the training loss over the epochs.

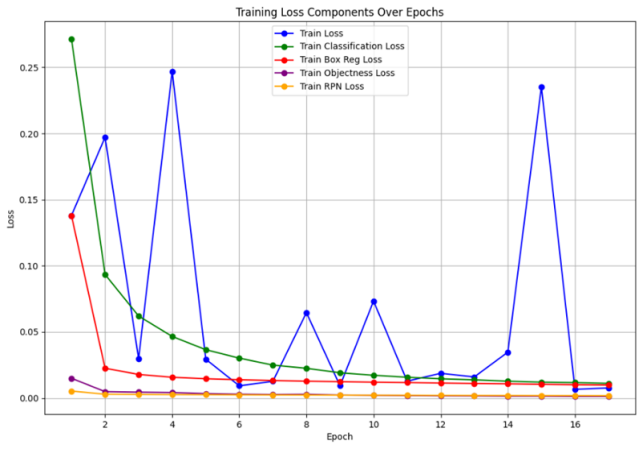

Fig.9 Training Loss components throughout the epochs for Faster-RCNN

For this model the time of training is about 9 hours and 30 minutes and for prediction of one image it's between 250 and 350 milliseconds on average.

B. YOLOv10

For the model using YOLOv10-B, the average mAP50 score across all the epochs came to 91.68%. The graph (**Fig.10**) below illustrates the different components of the training loss over the epochs.

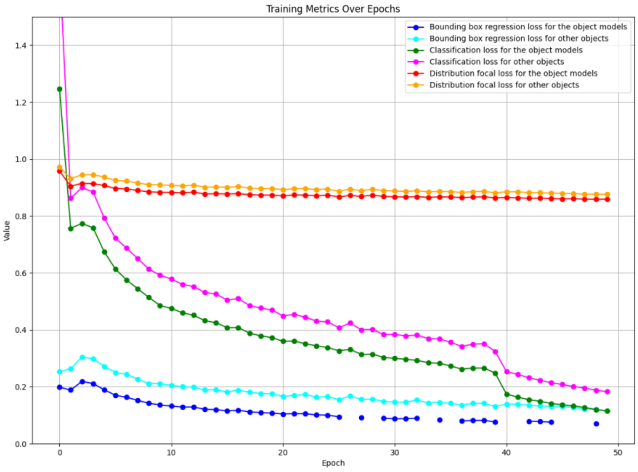

Fig.10. Training Metrics throughout the epochs for YOLOv10

For this model the time of training is about 7 hours and 45 minutes and for prediction of one image it's between 750 and 850 milliseconds on average.

C. SSD

For the model using ssdlite320_mobilenet_v3_large, the average mAP50 reached to 84.93%. The graph in **Fig.11** illustrates the different components of the training loss over the epochs.

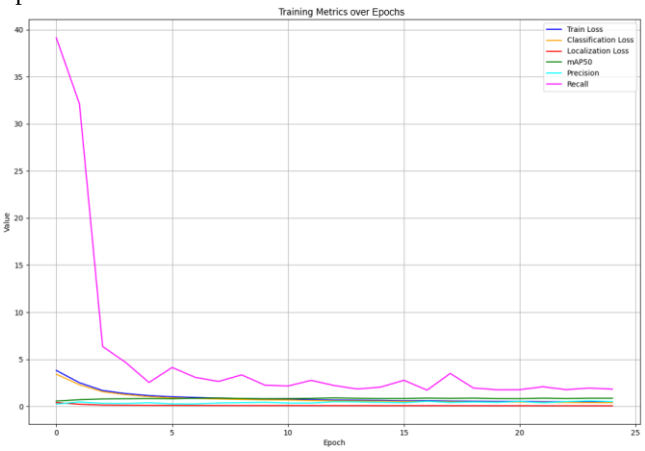

Fig.11 Training metrics throughout the epochs for the ssdlite320_mobilenet_v3_large model.

For this model, the training time is approximately 1 hour and 3 minutes, and the prediction time for a single image is between 291 and 350 milliseconds on average.

D. Comparison of all the models

1. Metrics Comparison

The following **Table 2.** show the comparison of all the relevant metrics generated from the training of all three models. As we can see, although Faster-RCNN takes the longest to train, it is overall the highest performing model that we trained with a map50 score of 92.13% and prediction time varying between 250 and 350 ms.

Moreover, our Faster-RCNN achieved an average mAP of 90.83% compared to the average mAP of 60% of our last article, representing a significant improvement of 30.83%.

Metric	Faster-RCNN	YOLOv10	SSD
map50 score	92.13%	91.68%	84.93%
Training time	9h30	7h45	1h03
Prediction time (ms)	250-350	750-850	291-350

Table 2. Metrics comparison between all three models.

2. Practical tests

To further test the models, we selected two different images from the greenhouse (one with larger leaves and one with smaller leaves) that the models had never seen before, to evaluate which model detects the most diseased leaves. Table. 3 below compares the number of leaves detected and individually framed by each model.

	Faster R-CNN	YOLOv10	SSD
Image with smaller leaves	8/15 sick leaves (53%)	0/15 sick leaves (0%)	0/15 sick leaves (0%)
Image with bigger leaves	11/20 sick leaves (55%)	3/20 sick leaves (15%)	5/20 sick leaves (25%)

Table 3. Number of sick leaves detected by each model.

From our results, we can once again conclude that Faster R-CNN is the best-performing model. It consistently detected and accurately framed individual leaves, rather than grouping them into large bounding boxes. Notably, in the case of images with smaller leaves, Faster R-CNN managed to frame the leaves separately but incorrectly identified the disease. In contrast, the SSD model more accurately identified the disease but framed the entire cluster of leaves rather than individual ones.

3. Analysis discussion

While our approach has shown promise, several areas for improvement have been identified, particularly related to the challenges of handling large clusters of tomato leaves and managing the surrounding noise in images. One significant challenge is the presence of dense clusters of tomato leaves in greenhouse settings, which complicates the accurate

identification and separation of individual leaves. To address these challenges, we propose the following improvements:

(1) Expanding the Dataset: To enhance the model's robustness and accuracy, like we mentioned earlier, we will continue to augment our dataset with additional images captured directly from the UdeM greenhouse. This will better represent the real conditions where the model will be deployed, leading to a more balanced and accurate training set.

(2) Improving Annotation Consistency: We previously employed both polygon and bounding box annotations for labeling leaves in our images in Roboflow. However, moving forward, we will only use bounding box annotations since it offers a significant advantage by capturing not just the leaf but also its surrounding context, including any background noise. This approach is crucial for preserving the natural variations around the leaf, which is essential for the model's ability to generalize and perform effectively in a greenhouse environment. In contrast, polygon annotations tend to "extract" the object from the image, isolating it and removing the surrounding context by placing it against a black background, similar to the dataset from PlantVillage, where each image contains a single leaf against a plain background. By using bounding boxes, we retain the complete image and simply highlight each leaf, ensuring that the model is trained to recognize leaves in realistic conditions.

VI. CONCLUSION

This study has highlighted the challenges and potential of using object detection models like SSD, Faster R-CNN, and YOLOv10 for disease detection in greenhouse environments. Our results show that while each model has its strengths, handling dense clusters of tomato leaves and managing surrounding noise remain significant challenges. To address these issues, we propose expanding the dataset with additional images from the UdeM school greenhouse and improving annotation consistency by focusing on bounding box annotations. These steps are essential for enhancing the models' robustness and accuracy in real-world applications.

Moving forward, we plan to deploy the most promising model in a real-time monitoring system within the school greenhouse. This system will leverage the model's capabilities to provide proactive disease management. Additionally, implementing a closed-loop feedback mechanism and cloud-based alerting functionalities will ensure continuous model improvement and effective disease detection in real-time, thereby providing greenhouse operators with timely and actionable insights.

ACKNOWLEDGMENT

We would like to thank Innovations ALIVEcode inc. for financially supporting this research. We would also like to thank Hannick Nadine Anoutsa Zangue, Alexandre Beaudoin and Claude Dagenais who manage the *School Greenhouse* project of the University of Montreal.

REFERENCES

[1] K. B. A. Bakar, F. T. Zuhra, B. Isyaku and S. B. Sulaiman, "A Review on the Immediate Advancement of the Internet of Things in Wireless Telecommunications," in IEEE Access, vol. 11, pp. 21020-21048, 2023, doi: 10.1109/ACCESS.2023.3250466.

[2] Bongiovanni, R., Lowenberg-Deboer, J. Precision Agriculture and Sustainability. Precision Agriculture 5, 359–387 (2004). https://doi.org/10.1023/B:PRAG.0000040806.39604.aa

[3] Martinelli, F., Scalenghe, R., Davino, S. et al. Advanced methods of plant disease detection. A review. Agron. Sustain. Dev. 35, 1–25 (2015). https://doi.org/10.1007/s13593-014-0246-1

[4] Y. Lakhdari, E. Soldevila and J. Rezgui,, "Detection of plant diseases in an industrial greenhouse: Development, Validation & Exploitation", accepted IEEE ISNCC 2023, Qatar.

[5] PlantVillage Kaggle repository https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset [last visited 21 August 2024]

[6] Sandro A. Magalhães, Luís Castro, Germano Moreira, Filipe N. Santos, mário Cunha, Jorge Dias, António P. Moreira. (2021). Evaluating the Single-Shot MultiBox Detector and YOLO Deep Learning Models for the Detection of Tomatoes in a Greenhouse. https://doi.org/10.3390/s21103569

[7] Alruwaili, M.; Siddiqi, M.H.; Khan, A.; Azad, M.; Khan, A.; Alanazi, S. RTF-RCNN. (2022). An Architecture for Real-Time Tomato Plant Leaf Diseases Detection in Video Streaming Using Faster-RCNN. https://doi.org/10.3390/bioengineering9100565

[8] Brucal, S., De Jesus, L., De Los Santos, J., Mendoza, M., Harion, K., Reyes, G., Nevalasca, D., & Reyes, J. (2023). Development of Tomato Leaf Disease Detection using Single Shot Detector (SSD) Mobilenet V2. International Journal Of Computing Sciences Research, 7, 1857-1869. www.stepacademic.net/ijcsr/article/view/405

[9] Zhong, Y. (2024). Tomato Leaf Disease Identification Based on Yolov8. International Journal of Computer Science and Information Technology, 3(2), 265-276. https://doi.org/10.62051/ijcsit.v3n2.30

[10] Ao Wang, Hui Chen, et al. (2024). YOLOv10: Real-Time End-to-End Object Detection. https://doi.org/10.48550/arXiv.2405.14458

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu (2016). SSD: Single Shot MultiBox Detector. https://doi.org/10.48550/arXiv.1512.02325

[12] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam. (2019). "Searching for MobileNetV3". https://doi.org/10.48550/arXiv.1905.02244