Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 525–530
DOI: 10.15439/2024F2492
ISSN 2300-5963 ACSIS, Vol. 39

LeAF: Leveraging Deep Learning for Agricultural Pest Detection and Classification for Farmers

Aditya Sengupta
Email: adityasngpta@gmail.com

***Abstract*—Farmers face many challenges while growing crops such as monitoring and maintaining plant health. Key indicators of poor plant health are plant anomalies such as pests, plant disease, and weeds, which can decrease crop yield. Over 40% of global crop production is lost to plant anomalies, costing $220 billion annually. As the global population and demand for food increases, farmers will have to grow more food, making manual surveying for plant anomalies increasingly difficult. This forces farmers to excessively and indiscriminately apply more fertilizers and pesticides across their whole fields, often to both healthy and unhealthy plants, unnecessarily wasting acres worth of chemicals and increasing chemical contamination of food and environmental footprint of agriculture as the chemicals release greenhouse gases after their application and leak into ecosystems. Recent advances in deep learning with Convolutional Neural Networks (CNNs) allow using imaging data to solve this problem. LeAF aims to provide farmers with an end-to-end system to survey crops on the field and take targeted actions to maintain plant health. By focusing on agricultural pests, this paper demonstrates the following capabilities for the visual perception sub-system of LeAF: (1) use CNNs on field images to get plant-specific data with bounding box based detection and classification about plant anomalies at human-level accuracy and (2) combine detection and classification functionality into a single compact distilled model that can run on farmer accessible mobile phones or in embedded devices in agricultural tractors and robots with low latency and high throughput to enable real-time processing on video feeds. With lightweight and accurate plant anomaly detection and classification, LeAF addresses plant health management challenges faced by farmers, empowering them with actionable insights to enhance productivity while minimizing chemical usage and its environmental impact.**

I. Introduction

Modern day agriculture has an enormous environmental footprint. In 2022, agriculture accounted for approximately 26% of global greenhouse gas emissions, used 50% of global inhabitable land, and accounted for 70% of all freshwater usage. [1]. This means the agriculture industry has one of the largest impacts on climate change. Over the next few decades, agriculture's environmental footprint will only get larger because about 60% more food needs to be grown to feed the increasing world population [2]. This means immediate and effective steps need to be taken to increase the efficiency of agriculture to improve yields, conserve resources, and reduce the environmental impacts.

A. Problem

Farmers face challenges with continuously monitoring and maintaining plant health. Key indicators of poor plant health are plant anomalies such as pests, plant disease, and weeds, which can decrease crop yield. Over 40% of global crop production is lost to plant anomalies, costing $220 billion annually [3], [4].

Farmers often use chemicals to treat plant anomalies and ensure healthy crops, high yields, and good quality food for consumers. However, a lot of the emissions from agriculture come from the use of these chemicals (pesticides, herbicides, and fertilizers) to maintain plant health and treat plant anomalies (pests, weeds, and plant disease, respectively). Chemical-related greenhouse gas emissions such as nitrous oxide and methane can have up to a 300x higher global warming impact than carbon dioxide [5]. The chemicals emit greenhouse gases in their manufacturing process (creating harmful ground-level ozone), transportation to farms (fuel emissions from trucks), and even after they are applied to crops as they stimulate the production of nitrous oxide and methane in the soil. Furthermore, chemicals often leak into environments surrounding farms and into the ocean, harming wildlife and disrupting food chains.

These chemicals also have further downstream harmful impact. For example, on top of yield losses from pests, an additional $60 billion dollars is spent on 1 billion pounds of pesticides annually in the US alone [6]. Furthermore, pesticide residues on food lead to 20,000+ new cases of cancer every year and contamination in nature from pesticides results in 80+ million fish and bird deaths annually [7]. However, with more infestations of invasive pests and the cultivation of more crops, the use of chemicals is only increasing [8].

The use of chemicals also increases production costs for farmers, increasing the cost of food for consumers. Not only are 78% of the world's poor people farmers [9], their income is also declining because of falling crop prices since 2014 despite inflation [10]. Reducing chemical costs can help farmers keep up with decreasing crop sales revenue.

Most farmers are forced to apply chemicals across their whole field to both healthy and unhealthy crops because they have no data on where the pest infestations are. This wastes acres worth of chemicals. In order to reduce the use of chemicals, they need to be used more efficiently to still counteract plant anomalies and allow for healthy yield. A farmer could survey their crops themselves and analyze plant anomalies manually, but they might not have enough knowledge about the types of plant anomalies, whether they are helpful, benign, or malignant to the farm ecosystem, or which chemicals to use in order to treat them. In addition, many farmers have acres of land which makes self-surveying

unfeasible due to the shortage of time and labor to make accurate observations on a daily or weekly basis. With such a lack of data on plant anomalies in their field, farmers are less aware of the efficacy of their techniques, which may lead to bad yields despite the use of chemicals.

This highlights the need for automation in the agricultural industry to detect and identify plant anomalies and help farmers with treatment strategies that minimize the use of chemicals, cost, and harm to the environment. A solution which can provide farmers with such data will pave the way for better quality yields while using less chemicals in a more efficient and effective manner and minimizing costs and environmental impact.

B. Related Work

There is limited work on using machine learning to identify plant anomalies; these also have gaps that are addressed by LeAF. The IP102 pest classification dataset [11] is used in the same paper to train a classification model with accuracy significantly below human level (49.4% with ResNet-50). IP102 also contains more than 100 classes, which requires a larger model to achieve this accuracy while trading off inference latency. Since there are thousands of pests that need larger model to classify accurately, it is instead more effective to use a smaller, lightweight model tuned specifically for local pests, as in LeAF, since farmers only deal with a few major pests in their area.

For plant disease classification, AMaizeD focuses on corn diseases [12]. For weed classification, recent work is limited to weeds that impact bell peppers [13].

The problem with these models is that they have low accuracy and are not generalizable to different types of anomalies for other plants. For example, if you give AMaizeD a non-corn disease image, it will still classify it as a specific corn disease. This is especially dangerous for pests because the model might classify an unknown but helpful insect as a pest, leading to the farmer applying useless pesticides. Moreover, these models focus on classification, not detection, so they don't provide bounding boxes to pinpoint the anomalies. When dealing with multiple anomalies and surveying multiple crops, knowing where the anomaly is in the image is necessary for explainability, to keep a count of anomalies, and avoid both over and under counting.

LeAF, on the other hand, uses transfer learning to quickly customize for regional and local scenarios with small datasets, provides human level classification and detection accuracy for plant anomalies, classifies unseen pests as "unknown" to avoid incorrect treatment, and is also amenable to expanding supported types of anomalies with incremental retraining.

C. Goals & Benefits of LeAF

LeAF advances plant anomaly detection and mapping for farmers, offering an end-to-end solution that enhances efficiency and minimizes chemical usage. By leveraging emerging agricultural robots or tractors equipped with cameras, LeAF captures and analyzes crop images, initially focusing on pests while designing to expand to diseases and weeds. By consolidating various plant anomaly detection tasks into a single lightweight model per anomaly type, the solution can run effectively on embedded devices in the edge, such as the EarthSense TerraSentia robot [14]. This streamlined approach not only ensures accuracy but also enables rapid tuning to different farming regions.

The ultimate objective of the LeAF end-to-end system [15] is to provide farmers with actionable insights, including treatment suggestions and cost-effectiveness estimates, through a user-friendly natural language based multimodal interface. By correlating plant anomaly data with mapped field arrays, LeAF enables farmers to monitor anomalies on a day-to-day basis and track the efficacy of their interventions, facilitating informed decision-making and optimizing crop yield.

To achieve this, LeAF utilizes precise yet lightweight model architectures like ResNet-18 [16] and YOLOv8 [17], trained on agricultural images sourced from the iNaturalist library [18]. The camera feed obtained while surveying the field (with robots or tractors, for example) is processed for anomaly detection and classification, with anomalies tracked with location awareness across the field. This individual plant level data is then mapped to a field array, facilitating analysis by an agriculture domain adapted Large Language Model (LLM) for treatment suggestions and Q&A support.

By empowering farmers with real-time insights and optimized chemical application strategies, LeAF aims to minimize environmental impact and maximizes crop yield, ultimately enhancing agricultural practices for a sustainable future. This paper covers the visual perception component of LeAF, involving plant anomaly detection and classification, and focuses on pests for concrete model development and evaluation.

II. PEST DETECTION AND CLASSIFICATION

The visual perception component of LeAF detects and classifies pests based on a given number of pest classes. The classes that LeAF aims to detect depend on the crop being grown and the geographical region of the world. For the prototype, I target ten main types of pests for corn and soybean crops grown in the midwestern region of the United States.

A. Initial Pipeline Overview

The solution comprises two stages: pest detection and classification. Initially, I develop a pipeline using foundation models for pest detection (with bounding box), evaluate its performance, and refine it. Subsequently, I focus on gathering datasets for training and evaluating a specialized pest classification model.

Given the challenge of obtaining large labeled datasets, I leverage foundation models like GroundingDINO [19] for pest detection. Although these models provide bounding boxes for pests, they lack fine-tuned classification accuracy for diverse agricultural pests. To address this, I utilize cropped images from GroundingDINO to feed into a specialized, more efficient classification model like ResNet-18. This two-stage pipeline, depicted in Figure 1, first localizes potential pests using

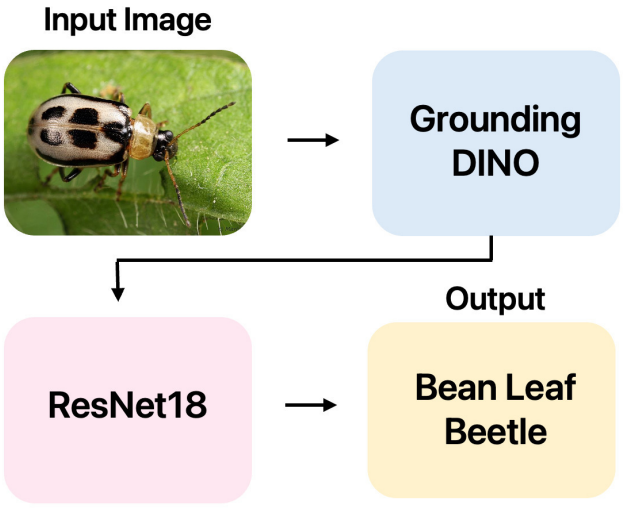

Fig. 1. Initial Pipeline with GroundingDINO and ResNet-18

GroundingDINO and then classifies them using ResNet-18. This approach mitigates data scarcity and enables recognition of unseen pest classes through an 'unknown' class.

Further exploration involves distilling this pipeline into a single, lightweight YOLOv8n model for mobile and edge deployment, as discussed in Section III.

B. Datasets & Training Prerequisites

After figuring out what the ML pipeline needs to input and output, building the initial pipeline, testing it, and coming up with a more refined solution, the next step is to train a custom model to classify pests. For training this and the eventual tweaking of the final pipeline, I first need to establish a dataset containing images of pests and their labels.

1) Identifying Pest Classes: LeAF focuses on the 10 most harmful pests affecting corn and soybean crops in the Midwestern United States, as identified by an UIUC entomology study [20]. These classes include Bean Leaf Beetle, Grape Colaspis, Japanese Beetle, Northern Corn Rootworm (CRW), Southern CRW, Western CRW, Grasshopper, Cloverworm/Looper, Stink Bug, and Dectes Stem Borer. This selection prioritizes the needs of farmers in this agriculturally significant region.

2) Curating Initial Dataset: Training machine learning models requires substantial data, which is often scarce for specific pest classes. LeAF leverages iNaturalist [18], a crowdsourced platform with hundreds of millions of images of various species, to curate a dataset of relevant pest images for model training.

The iNaturalist dataset includes images of all 10 of the above pest classes, so I created the training dataset by downloading images from iNaturalist. Utilizing iNaturalist's export tool, I filtered images meeting research-grade standards and consensus thresholds. I ensured a balanced distribution, with around 1,000 images per class (total 10,000 images across all classes), and split them into 80% for training, 10% for validation, and 10% for test.

3) Device Platform: For training and inference, I use Python 3.10 and the PyTorch ML framework on multiple hardware platforms, including Google Colab with Nvidia T4 GPUs, on-prem cluster of Nvidia A100 GPUs, and M1 Mac Mini desktop.

C. Training

With this pipeline structure of GroundingDINO feeding into a smaller CNN, it was now time to select the CNN model structure and custom train it on the pest dataset. Initially, I started with a ResNet-18 model [21], which is the smallest in ResNet family with one of the most simple CNN structures.

1) Initial Training Process: I initialized a ResNet-18 model pre-trained on ImageNet and replaced its last layer with a 10-class fully connected layer for pest classification [21]. Leveraging transfer learning, I utilized previously learned features to expedite training with less data. This approach also enables selective training of the final layer, conserving computational resources and accelerating training.

2) Enhancing Accuracy: Training ResNet-18 on the dataset for 100 epochs yielded a stagnant accuracy around 60%. Despite comparable training and validation accuracy, both were below human accuracy levels (which are above 90%), indicating underfitting. Experimenting with a larger model, ResNet-101, raised accuracy to nearly 80%, but extended training time and inference cost and latency beyond feasibility. Opting for computational efficiency, I retained the smaller ResNet-18 model and focused on the following optimizations to reach target accuracy of 90%+.

3) Adjusting/Augmenting Data to Increase Accuracy: After reassessing the pipeline based use case, I realized that ResNet-18 only needed to do inference on already cropped pest images from GroundingDINO's output bounding boxes. Instead of the initial pest images, using cropped images addresses underfitting because the model will get less confused about the distracting image backgrounds and more focused on the pest and its features, since the image is essentially more zoomed in to the pest after cropping. Therefore, for the training and validation data, I fed all the images through GroundingDINO and cropped them to the bounding boxes for pests in the image. This made the input images for the ResNet-18 smaller and more focused, allowing the model to pick up on the features of the pest, therefore increasing accuracy.

Dataset	Train Accuracy	Validation Accuracy
Original	66.67%	65.71%
Cropped	82.60%	81.29%

TABLE I
RESNET-18 MODEL ACCURACY AFTER 10 EPOCHS OF TRAINING ON ORIGINAL AND CROPPED PEST DATASETS.

This change increased accuracy from 60% to 80% as shown in Table I. After this, I made smaller improvements to the training procedure by adjusting hyperparameters to further increase accuracy as follows.

4) Adjusting Training Hyperparameters: To further increase the accuracy from 80% to human levels (90%+) and speed up training, I adjusted many of the training hyperparameters for the ResNet-18 model. The adjustments in this section are incremental and cumulative. For example, if the optimizer was changed to Adam, then all future testing includes this change. In addition, all results were achieved by training for 10 epochs (unless otherwise specified). Training accuracy was evaluated based on correct/incorrect classifications on the training dataset and validation accuracy is based on the validation dataset. Here are the hyperparameters that were tuned:

Optimizer and Learning Rate: I started by using the Stochastic Gradient Descent (SGD) training optimizer. In addition to the other optimizations, this was giving about an 85% accuracy. However, when experimenting with other optimizers, I found the Adam Optimizer to work best. Next, I tuned the learning rate. While the learning rate is less important with the presence of optimizers which can adapt the learning rate based on the change in loss, it still plays an important role in providing an estimate or range. I found the perfect balance of decrease in loss to optimal end-accuracy at a learning rate of 0.001. Specifically, using the Adam optimizer helped the model jump to 91% accuracy, which now met the threshold of human performance. These optimizer results are shown in Table II and the learning rate results are shown in Table III.

Optimizer	Train Accuracy	Validation Accuracy
SGD	82.60%	81.29%
Adam	89.26%	88.97%

TABLE II
RESNET-18 MODEL ACCURACY USING SGD AND ADAM OPTIMIZERS.

Learning Rate	Train Accuracy	Validation Accuracy
0.01	89.26%	88.97%
0.001	91.60%	90.96%
0.0001	90.51%	90.18%

TABLE III
RESNET-18 MODEL ACCURACY USING DIFFERENT LEARNING RATES FOR ADAM OPTIMIZER.

Batch Size: Batch size is an important training hyperparameter to adjust. On one hand, increasing batch size improves training throughput (by increasing compute utilization) but requires more memory. On the other hand, increasing batch size may have negative effects on regularization, training convergence, and model quality. I found that a batch size of 32 was optimal as it maximized CPU and GPU utilization for the amount of RAM available, allowing for fast training. This batch size was also favorable for obtaining high accuracy. These results are shown in Table IV.

Layers to Train: As mentioned before, larger models like the ResNet-101 took more time and resources to train than smaller models like ResNet-18. One way to make the training process even faster is to train only the last (fully connected) layer of the model. This works because with a pre-trained

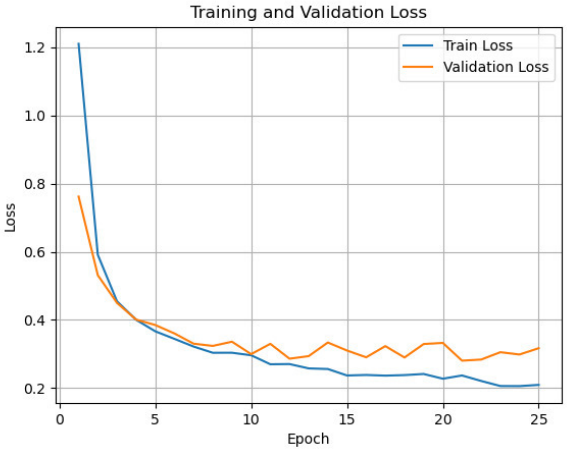

Fig. 2. Training and validation loss over 25 epochs. The validation loss flattens out after 10 epochs.

ResNet-18, the previous pre-training on ImageNet still helped tune the earlier layers to classify features present in any image like edges, contours, texture, and color. The only thing new is the classification of pests rather than everyday objects.

Training only the last layer decreases training time by 8x and produces almost identical accuracy results (within 1%), allowing to train for more epochs and tune other hyperparameters faster. The results are shown in Table V.

Number of Epochs: I found that 10 epochs were optimal for training this model by looking at the graphs plotting train and validation loss from 10 epochs (see Figure 2). Even though initial epochs allow bigger decreases in loss, this flattens out over time. After 10 epochs, the validation loss oscillates and does not decrease noticeably. While the training loss still decreases slowly, it is not necessary to train beyond this point because this means the model starts to overfit and make changes to weights that don't generalize well with new data.

5) Summary: Table VI summarizes the key milestones in the journey from 60% to 91%+ accuracy.

6) Other Model Architectures: I experimented with other model architectures like EfficientNet, MobileNet, ViT, and SqueezeNet. However, since this model is being deployed on the edge, it must be as lightweight as possible while still maintaining 90+% accuracy. Therefore, I had to decide on the tradeoff between accuracy and computational resources. The accuracy and inference times for the different models are in Table VII.

ResNet-18 exhibits the best efficiency-to-accuracy ratio among tested architectures, striking an optimal balance. Despite having more parameters than MobileNet, ResNet-18's performance speed remains comparable, indicating additional optimizations beyond parameter count. Post training, ResNet-18 achieved a 91.43% classification accuracy on the validation set comprising 1,000 cropped images.

In summary, the initial model pipeline (depicted in Figure 1) employs GroundingDINO to generate bounding box detec-

Batch Size	Average Training Time Per Epoch	Train Accuracy	Validation Accuracy
16	52.35 seconds	87.91%	87.56%
32	36.40 seconds	91.73%	91.43%
64	38.50 seconds	91.32%	90.87%

TABLE IV
RESNET-18 MODEL ACCURACY USING DIFFERENT BATCH SIZES.

Layer to Train	Train Accuracy	Validation Accuracy	Average Training Time Per Epoch
Train Only Last Layer	91.32%	90.87%	36.35 seconds
Train All Layers	91.45%	90.66%	273.46 seconds

TABLE V
RESNET-18 MODEL ACCURACY WHEN TRAINING LAST VS ALL LAYERS.

Tuning Technique	Validation Accuracy
Original Accuracy	65.71%
Data Filtering/Augmentation	81.29%
Adam Optimizer	88.97%
Learning Rate 0.001	90.96%
Batch Size 32	91.29%
Train 10 Epochs (Last Layer)	91.43%

TABLE VI
IMPACT OF TUNING TECHNIQUE ON VALIDATION ACCURACY (AFTER 10 EPOCHS OF TRAINING).

Model	Validation Accuracy	Inference Time per 1K images (sec)
ResNet-18	91.29%	36.35
MobileNetV2 S	87.65%	32.47
EfficientNetV2 L	93.32%	79.73
ViT B16	95.53%	126.82
SqueezeNet	89.96%	50.34

TABLE VII
PERFORMANCE OF DIFFERENT MODEL ARCHITECTURES.

tions, followed by ResNet-18 for pest classification, obtaining an accuracy of 90%+.

D. Example Detection Results

Figure 3 shows some sample results for detection and classification. The pipeline even performs well on tough examples like a brown grasshopper camouflaged in brown leaves, and a Corn Rootworm (CRW) camouflaged in a leaf and flower. In addition, for images where there is no pest, the pipeline works correctly because those images are filtered by the GroundingDINO model (which is why there wasn't a need to train a separate class for "no pests in image" scenarios on the ResNet-18).

III. DISTILLED LIGHTWEIGHT MODEL FOR MOBILE AND EDGE DEVICES

While the GroundingDINO and ResNet-18 based model pipeline has good accuracy, it is slow and takes 3 seconds per image on an Nvidia T4 GPU in Google Colab. To enable this to run in real time on an embedded device like a Raspberry Pi or Nvidia Jetson Nano, the pipeline needs to be shrunk drastically.

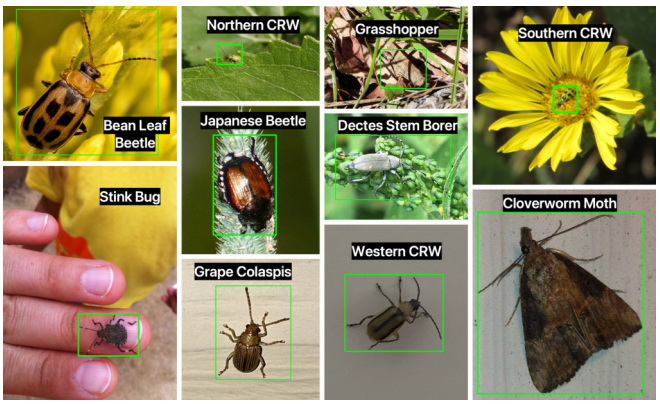

Fig. 3. Sample detection and classification results on test images with GroundingDINO and ResNet-18 pipeline

A. YOLOv8n instead of GroundingDINO and ResNet-18

The main bottleneck is the GroundingDINO model, which has 172M parameters compared to ResNet-18 which has only 11M parameters. One efficient model that can replace both is YOLO [17], which is an efficient one-shot computer vision model that can output bounding boxes and classifications. I will use YOLOv8n (Nano), latest lightweight version of YOLO with only 3.2M parameters (0.17% of the initial pipeline), which also enables huge increase in speed.

B. Model Distillation

Integrating YOLOv8n into the refined pipeline may challenge accuracy due to its smaller size. However, GroundingDINO's comprehensive bounding box training exceeds our need for pest classification alone. To transfer GroundingDINO's bounding box knowledge to YOLOv8n, I employ model distillation. This entails a larger foundation model annotating a task-specific dataset, from which a smaller student model learns. The previously annotated iNaturalist dataset using GroundingDINO, containing cropped images based on their bounding box coordinates to isolate pests with class labels, is used to train YOLOv8n. I used the PyTorch based Ultralytics API [22] to train the model for 10 epochs in about 30 minutes on a Mac Mini M1.

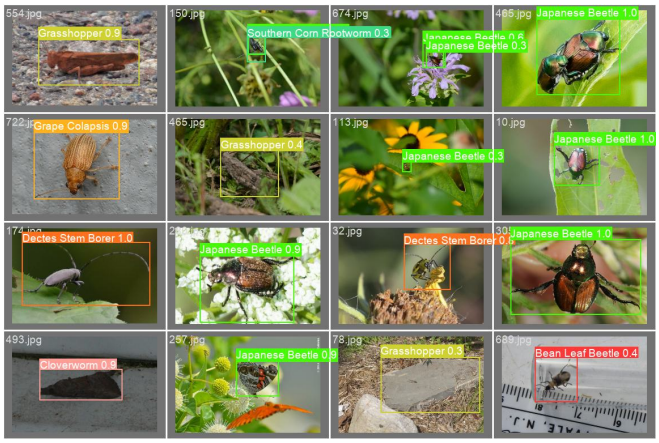

Fig. 4. Sample Detections and Classifications from YOLO Model (with confidence scores)

C. Evaluation on Pest Images

Testing the output of this YOLO model on some sample pest images yielded accurate bounding box results, shown in Figure 4. The model now takes only about 10 milliseconds per image on a mobile phone, compared to 3 seconds per image in initial pipeline on Nvidia T4 GPU, which is a 300x reduction in inference latency. This allows YOLO to perform real-time inference on video at 30 fps even on a mobile phone. The corresponding decrease in model size is 600x.

The model size and inference latency reduction benefits of the distilled model come with accuracy preservation comparable to that of the initial pipeline with GroundingDINO and ResNet-18 (as per the mAP metric). The distilled model has mAP50 of 0.81 (computed over IoU $\geq 50\%$). For the original pipeline, since GroundingDINO is a foundation model trained on vast amounts of data, it outputs perfect bounding boxes, so all of the detections satisfy the 50% overlap requirement of mAP50. Hence, the mAP calculation for initial pipeline simplifies to $mAP = \frac{1}{n}\sum_{i=1}^{n} Precision_i * Recall_i$ ($n =$ #classes). This gives an mAP of 0.82 for the original pipeline, hence mAP of the distilled model is very close, thus successfully preserving accuracy.

D. Handling Unknown Pest Classes

The initially trained YOLO struggled to classify out-of-distribution (OOD) pests, often misclassifying them as the most similar trained class. Detecting unknown pests becomes challenging, as class probabilities tend to dominate a single class rather than being spread across multiple classes.

Given the vast number of pest species globally, training YOLO to classify every pest is impractical, especially for edge deployment. I explored various techniques to detect unknown pests, ultimately settling on augmenting the original YOLO model with an 'unknown' pest class and trained it using a random distribution of pest images from the IP102 dataset.

This enhancement prevents misclassification of beneficial insects like ladybugs as harmful pests by assigning them to the unknown class. Farmers can then manually review these images to determine their relevance. If numerous unknown pests emerge during deployment, the YOLO model can be extended to include additional pest classes, by incrementally expanding the last fully connected layer and retraining.

IV. Conclusion

LeAF offers lightweight, accurate pest detection and classification on mobile devices by distilling YOLO based model from Internet scale datasets using GroundingDINO foundational model for bounding box labeling and custom trained Resnet-18 model for classification. This visual perception capability for agricultural pests is part of the end-to-end LeAF system [15] that includes field level mapping for plant-by-plant analysis combined with LLM-powered analysis and recommendations to empower farmers with actionable insights for efficient plant health management. Informed by ongoing deployments and farmer feedback, LeAF is being further refined to ensure its effectiveness in promoting environmentally sustainable and productive agriculture.

References

[1] H. Ritchie, P. Rosado, and M. Roser, "Environmental impacts of food production," *Our World in Data*, 2022.
[2] J. G. D. Silva, "Feeding the world sustainably," *United Nations Chronicle*, 2012.
[3] "Invasive pest spread another fallout from climate change, UN-backed study finds," *United Nations*, 2021.
[4] N. Bhalla, "40% of global crop production is lost to pests. and it's getting worse," *World Economic Forum*, 2021.
[5] J. Garthwaite, "Why laughing gas is a growing climate problem," *Stanford News*, 2020.
[6] S. LaMotte, "Reducing pesticides in food: Major food manufacturers earn an F grade," *CNN*, 2023.
[7] A. Pariona, "Top pesticide using countries," *WorldAtlas*, 2017.
[8] M. Tudi, H. D. Ruan, L. Wang, J. Lyu, R. Sadler, D. Connell, C. Chu, and D. T. Phung, "Agriculture development, pesticide application and its impact on the environment," *National Library of Medicine*, 2021.
[9] "For Up to 800 Million Rural Poor, a Strong World Bank Commitment to Agriculture," *World Bank*, 2019.
[10] W. A. Reinsch, T. Denamiel, and E. Kerstens, "Climate change and U.S. agricultural exports," *CSIS*, 2023.
[11] X. Wu and et al., "IP102: A Large-Scale Benchmark Dataset for Insect Pest Recognition," *IEEE CVPR*, 2019.
[12] A. Mall, S. Kabra, A. Lhila, and P. Ajmera, "AMaizeD: an end to end pipeline for automatic maize disease detection," *ICST*, 2023.
[13] A. Subeesh and et al., "Deep Convolutional Neural Network Models for Weed Detection in Polyhouse Grown Bell Peppers," *Artificial Intelligence in Agriculture*, 2022.
[14] "EarthSense TerraSentia," https://www.earthsense.co/robotics.
[15] A. Sengupta, "LeAF: Leveraging Deep Learning for Plant Anomaly Detection and Classification for Farmers with Large Language Models for Natural Language Interaction & BRANCH Robot-Based Deployment," *IEEE CVPR Computer Vision for Science*, June 2024.
[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE CVPR*, 2016.
[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *IEEE CVPR*, 2016.
[18] "iNaturalist," https://www.inaturalist.org/.
[19] S. Liu and et al., "Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection," *arXiv.org*, 2024.
[20] A. Decker and et al., "2022 applied research results field crop disease and insect management," *UIUC Technical Report*, 2022.
[21] "ResNet18 - Torchvision main documentation," https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html.
[22] "Ultralytics YOLO," https://docs.ultralytics.com/.