Using Transfer Learning to Adapt MobileNet for General Plant Disease Detection on Irregular Images

Medhya Goel
Stanford University
450 Jane Stanford Way, Stanford, CA 94305
medhya@stanford.edu

Abstract

Early detection of plant diseases is critical for ensuring global food security, particularly in low-resource settings where expert diagnosis is not readily available. While deep learning models trained on curated datasets like PlantVillage have shown promise, they often fail to generalize to real-world field conditions due to significant domain shifts. In this work, we explore transfer learning techniques to adapt a lightweight MobileNetV2 model—pretrained on ImageNet and fine-tuned on PlantVillage—for robust disease classification on PlantDoc, a more diverse and noisy dataset. We evaluate four finetuning strategies: classifier-only, full-network finetuning (with and without augmentation), and a two-phase finetuning approach. Our results show that two-phase finetuning achieves the highest weighted F1 score (49.89%) and outperforms a zero-shot baseline by over 15 percentage points in accuracy. We attribute these improvements to targeted adaptation of higher-level layers while preserving transferable low-level features. Our study demonstrates the effectiveness of lightweight transfer learning pipelines for mobile deployment in field settings and highlights promising directions for future work in robust agricultural diagnostics.

1. Introduction

Each year, the world loses between 20 and 40% of crops to pests and diseases, costing the global economy $220 billion a year [9]. Beyond the financial impact, these losses pose a serious threat to global food security. With the world population projected to reach 9.8 billion, the United Nations estimates that we will need to increase food production by 50% to meet demand in 2050 [4]. In this context, early and accurate identification of plant diseases is essential not only for reducing crop losses, but also for ensuring sustainable agricultural practices.

A major barrier to achieving widespread, accurate disease diagnosis is the lack of accessible diagnostic tools for non-experts. Farmers—particularly in low-resource or rural environments—often rely on visual inspection to identify diseases, which can be unreliable due to the variability in how symptoms manifest across crop types, growth stages, and environmental conditions. Misdiagnosis can lead to inappropriate or excessive use of chemical treatments, compounding economic losses and contributing to environmental degradation. Thus, there is a pressing need for scalable, reliable, and field-deployable disease recognition systems that do not depend on expert knowledge.

Recent advances in computer vision and deep learning offer a promising solution. Convolutional neural networks (CNNs) have achieved remarkable success in image classification tasks, and have been applied to problems such as identifying skin lesions, detecting diabetic retinopathy, and diagnosing pneumonia from X-rays. Similar techniques have been applied to agriculture, most notably using curated datasets like PlantVillage (introduced in 2015 [5]). However, models trained on clean, lab-like datasets often fail to generalize well to real-world field conditions where images may be blurry, uncentered, occluded, poorly lit, or contain complex backgrounds. Further, different geographical locations may have very different diseases that they are interested in, but farmers and communities in these regions may not have enough compute or data to train a model to identify these diseases from scratch. This domain shift highlights a core challenge in building field-ready models and motivates the use of transfer learning to improve robustness under distributional shifts.

In this project, we investigate the use of various transfer learning techniques to adapt a lightweight CNN architecture—specifically MobileNetV2–trained on the clean PlantVillage dataset to the more realistic and noisy PlantDoc dataset, for the task of classifying crop diseases from images. Our objective is to build a model that is not only accurate, but also efficient and robust enough to be deployed on mobile devices for in-field diagnosis using transfer learning. The input to our system is an image of a plant leaf captured in uncontrolled conditions, and the output is the predicted crop species and disease class (or crop species and "healthy" if no disease is detected). We evaluate our approach on the PlantDoc dataset to assess accuracy, generalization, and suitability for real-world use.

2. Related Work

2.1. Traditional Machine Learning Approaches

Early efforts in plant disease detection relied on traditional machine learning algorithms, which typically involved manual feature extraction followed by classification using models like support vector machines (SVMs) or decision trees. For instance, Mohanty *et al*. [8] utilized handcrafted features such as color and texture to classify plant diseases, achieving moderate accuracy. However, these methods often struggled with generalization due to their reliance on manually engineered features and limited capacity to capture complex patterns in image data.

2.2. Convolutional Neural Networks (CNNs)

The advent of CNNs revolutionized image-based plant disease detection by enabling automatic feature extraction and end-to-end learning. Sladojevic *et al*. [14] demonstrated the effectiveness of CNNs in classifying 13 different plant diseases with high accuracy. Similarly, Ferentinos [3] achieved over 99% accuracy using deep CNNs on a dataset of 87,848 images. Despite their success, these models often require substantial computational resources and large labeled datasets, limiting their applicability in resource-constrained environments.

2.3. Evolution of MobileNet-Based Approaches to Plant Disease Classification

To address resource limitations, researchers have proposed making use of more lightweight CNNs like MobileNet and EfficientNet. These architectures offer a balance between performance and computational efficiency that make them ideal for deployment on mobile and embedded devices.

Various research teams have tried finetuning MobileNet to classify plant diseases, usually focusing on a particular plant species. One of the earliest teams to publish about this technique, Zaki *et al*. [16] achieved over 90% accuracy in classifying the subset of tomato leaf diseases within the PlantVillage dataset, using a MobileNet V2 architecture that they finetuned on PlantVillage images. Vinuta M. S. *et al*. [6] (2019) extended this technique to a broader subset of PlantVillage, achieving over 90% accuracy while identifying 15 disease classes spanning 8 plant species. These studies demonstrate the effectiveness of MobileNet when trained and tested on clean, controlled data. Wang *et al*. (2023) [15] introduced an ultra-lightweight efficient network (ULEN) incorporating MobileNet's depthwise separable convolutions for plant disease and pest detection. The model, with approximately 100,000 parameters, achieved competitive accuracy while being highly efficient for real-time applications. The architecture was designed to be trained from scratch, focusing on efficiency.

Some work has also been done to build atop the architecture of MobileNet and other pretrained models. Shafik *et al*. [12] introduce models that integrate early fusion with pretrained CNNs (including MobileNet) finetuned by deep feature extraction on PlantVillage dataset with considerable success (97% accuracy). However, modifying the architecture of these pretrained models can increase the size and complexity, reducing the benefit of MobileNet's intentional lightweight structure.

Importantly, despite the gains in efficiency achieved by reducing the size and complexity of the CNNs used for plant disease classification, most of these models are not generalizable to real world data. Most existing work focuses on finetuning models on curated datasets like PlantVillage, which contain clean, centered, and well-lit images. This limits generalization to real-world, in-field conditions that involve occlusion, lighting variation, multiple leaves, and overlapping symptoms. Additionally, class imbalance, data scarcity for rare diseases, and difficulty in distinguishing visually similar conditions (e.g., nutrient deficiency versus fungal infection) hinder robust performance.

2.4. Transfer Learning for Irregular Plant Disease Datasets

While MobileNet-based models show high accuracy on curated data, their generalization to field data is underexplored. Few works combine MobileNet's efficiency with explicit transfer learning pipelines targeting real-world robustness across many disease types. Additionally, most literature ignores fine-grained classification on datasets like PlantDoc.

Beyond classification, object detection and segmentation models aim to localize disease symptoms within images. YOLO (You Only Look Once) models, known for their speed and accuracy, have been applied to plant disease detection. For instance, a recent study [10] finetuned YOLOv7 and YOLOv8 and tested them on the PlantVillage dataset as well as a more irregular Cassava disease dataset, achieving high precision and real-time performance. Miao *et al*. [7] similarly tune YOLOv8 on the PlantDoc dataset, achieving a precision of over 0.7. However, these models require extensive annotated data for training and may struggle with detecting diseases at early stages or under occlusions. This exploration also explores only a relatively narrow set of irregular data (specifically cassava) rather than a broader range of plant species and diseases.

In 2022, Aarizou and Merah [1] proposed transfer learning on a hybrid dataset of regular (PlantVillage) images and irregular (EdenLibrary) images to finetune various classifiers (AlexNet, ResNet, etc.) pretrained on ImageNet to classify plants as healthy or unhealthy. Although informative, such coarse-grained labels limit real-world utility—farmers often already know if a plant is unhealthy; identifying the disease is what informs treatment. Moreover, heavier models like ResNet are less suitable for on-device deployment.

3. Methods

3.1. Codebase

This project builds on top of a MobileNetV2 model that has been pretrained on ImageNet and fine-tuned on the PlantVillage dataset to classify 38 plant disease classes. The pretrained model is available at `https://huggingface.co/linkanjarad/mobilenet_v2_1.0_224-plant-disease-identification` [2]. All training code, augmentation pipelines, logging, and evaluation metrics were implemented from scratch in Google Colab using PyTorch and Hugging Face Transformers. The base MobileNetV2 weights were loaded from HuggingFace.

3.2. MobileNetV2 Architecture

MobileNetV2 is a convolutional neural network designed for efficient inference on mobile and embedded devices [11]. It achieves computational efficiency by combining two key design elements: depthwise separable convolutions and inverted residual bottleneck blocks (see Figure 2).

Depthwise separable convolutions decompose a standard convolution into a depthwise convolution (applying a single filter per input channel) followed by a pointwise 1×1 convolution that mixes information across channels. This decomposition significantly reduces the number of computations and parameters, costing $h_i \cdot w_i \cdot d_i \left(k^2 + d_j\right)$, which is a factor of k^2 or 8-9 times less than standard convolutions, with only a small decrease in accuracy.

The central innovation of MobileNetV2 is the use of inverted residuals with linear bottlenecks. In traditional residual connections (as in ResNet), the input is passed through a block that expands its representation before the result is added back to the input (see Figure 1). MobileNetV2 inverts this idea: it first expands the input to a higher-dimensional space using a pointwise convolution, applies a depthwise convolution in that space, and then projects it back down to a lower-dimensional output using another pointwise convolution.

Mathematically, for input $x \in \mathbb{R}^{H \times W \times d}$ (with H height, W width, d channels), the inverted residual block performs:

$$x' = \text{Conv}_{1\times1}^{\text{expand}}(x) \rightarrow \text{DWConv}_{3\times3}(x') \rightarrow \text{Conv}_{1\times1}^{\text{project}}(x')$$

If the input and output dimensions match, a residual connection is added.

This structure allows the network to maintain representational power with fewer parameters, while the linear bottleneck avoids unnecessary non-linearities (ReLU) in low-dimensional spaces that can destroy information. These inverted residuals are especially well-suited for mobile applications, as they maximize the ratio of accuracy to FLOPs.

With fewer parameters and a latency-friendly design, MobileNetV2 provides a lightweight yet expressive backbone, making it a strong choice for deploying disease classification models in the field using mobile devices or edge hardware.

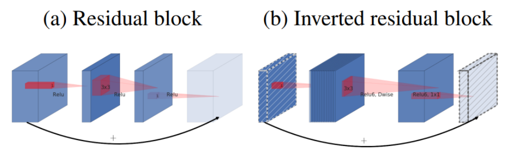
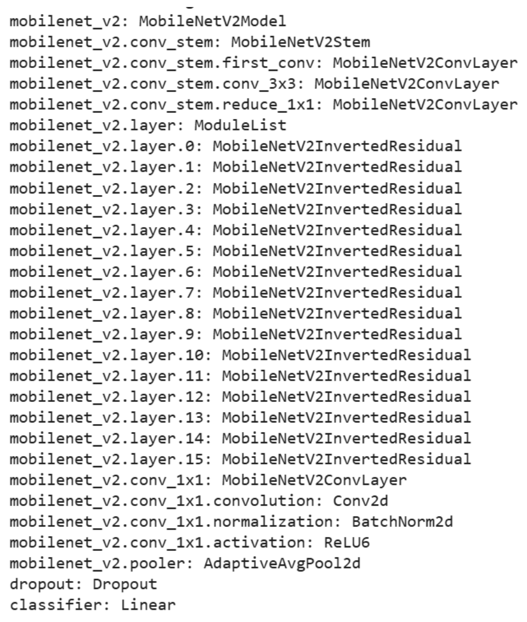


Figure 1. Residual Block and Inverted Residual Block (Figure from [11])



```
mobilenet_v2: MobileNetV2Model
mobilenet_v2.conv_stem: MobileNetV2Stem
mobilenet_v2.conv_stem.first_conv: MobileNetV2ConvLayer
mobilenet_v2.conv_stem.conv_3x3: MobileNetV2ConvLayer
mobilenet_v2.conv_stem.reduce_1x1: MobileNetV2ConvLayer
mobilenet_v2.layer: ModuleList
mobilenet_v2.layer.0: MobileNetV2InvertedResidual
mobilenet_v2.layer.1: MobileNetV2InvertedResidual
mobilenet_v2.layer.2: MobileNetV2InvertedResidual
mobilenet_v2.layer.3: MobileNetV2InvertedResidual
mobilenet_v2.layer.4: MobileNetV2InvertedResidual
mobilenet_v2.layer.5: MobileNetV2InvertedResidual
mobilenet_v2.layer.6: MobileNetV2InvertedResidual
mobilenet_v2.layer.7: MobileNetV2InvertedResidual
mobilenet_v2.layer.8: MobileNetV2InvertedResidual
mobilenet_v2.layer.9: MobileNetV2InvertedResidual
mobilenet_v2.layer.10: MobileNetV2InvertedResidual
mobilenet_v2.layer.11: MobileNetV2InvertedResidual
mobilenet_v2.layer.12: MobileNetV2InvertedResidual
mobilenet_v2.layer.13: MobileNetV2InvertedResidual
mobilenet_v2.layer.14: MobileNetV2InvertedResidual
mobilenet_v2.layer.15: MobileNetV2InvertedResidual
mobilenet_v2.conv_1x1: MobileNetV2ConvLayer
mobilenet_v2.conv_1x1.convolution: Conv2d
mobilenet_v2.conv_1x1.normalization: BatchNorm2d
mobilenet_v2.conv_1x1.activation: ReLU6
mobilenet_v2.pooler: AdaptiveAvgPool2d
dropout: Dropout
classifier: Linear
```

Figure 2. MobileNetV2 structure

3.3. Baseline

As a baseline, we use zero-shot learning to run inference on the PlantDoc dataset using the baseline model (MobileNetV2 pretrained on ImageNet and finetuned on PlantVillage). We map the model's 38 PlantVillage classes to the 29 PlantDoc classes using a dictionary and consider predictions of unmapped classes (classes that exist in PlantVillage but not in PlantDoc) to be incorrect .

We also run inference on the PlantVillage dataset on which our Baseline Model was originally finetuned in order to sanity-check that the Baseline Model does achieve high accuracy and performance on the finetuned dataset. We do not do any mapping here since the model's existing classifier head already outputs to the PlantVillage dataset classes.

3.4. Transfer Learning and Fine-Tuning

To transfer the PlantVillage-pretrained MobileNetV2 to conduct inference effectively on the PlantDoc dataset, which contains fewer classes, we first substitute an appropriately-sized and randomly initialized classifier head in the model. Then, we experiment with several transfer learning strategies:

- **Single-phase fine-tuning with feature extraction**: update only the classifier weights while finetuning on unaugmented PlantDoc data.
- **Single-phase fine-tuning of all weights**: updating all weights with unaugmented PlantDoc data.
- **Single-phase fine-tuning of all weights with augmentation**: updating all weights while finetuning on augmented PlantDoc data (see 3.5).
- **Two-phase fine-tuning**: (1) fine-tune only the classifier layer while freezing all other layers, then (2) unfreeze dropout and last three inverted-residual blocks, and fine-tune these higher-level convolutional layers, dropout layer, and classifier weights.

We randomly split the PlantDoc training dataset into an 80/20 split to create a training and validation split and verified that the class distribution of all three splits is similar. We also overfit the finetuning pipeline to a single training example to verify the correctness of the finetuning pipeline.

3.5. Justification for Selective Unfreezing

In the second phase of our two-stage fine-tuning process, we chose to unfreeze only the final few layers of the MobileNetV2 backbone—specifically layers 13 through 15, the final 1×1 convolutional layer ($conv_1x1$), and the classifier and dropout layers. This selective unfreezing strategy was motivated by the need to balance adaptation to the new domain (PlantDoc) with retention of the general visual features learned from the much larger and cleaner PlantVillage dataset.

MobileNetV2, like other convolutional architectures, learns hierarchical features, with earlier layers capturing low-level information (e.g., edges, color gradients) and later layers capturing high-level task-specific patterns (e.g., leaf texture, disease spots). Because the low-level features are often transferable across domains, we freeze the earlier layers to preserve their representations and avoid overfitting to the limited and noisy PlantDoc training data.

We unfreeze only the final three layers to allow the model to adapt its high-level representations to the new dataset, where disease cues may appear under different lighting, backgrounds, and leaf poses than in PlantVillage. This provides enough capacity for domain-specific adaptation while keeping the majority of the network stable, reducing the risk of catastrophic forgetting and overfitting. Additionally, we reduced the learning rate in Phase 2 (to 2×10^{-5}) to accommodate the increased number of trainable parameters while allowing fine-grained updates to these deeper layers without destabilizing the pretrained weights.

3.6. Data Augmentation

Given the significant domain shift between datasets, we applied aggressive augmentations to the PlantDoc training set:

- `RandomResizedCrop`(224, scale=(0.6, 1.0))
- `RandomHorizontalFlip`(p=0.5)
- `RandomRotation`(20°)
- `ColorJitter`(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.05)
- `RandomErasing`(p=0.25)

These augmentations aim to simulate the variability found in real-world field data and reduce overfitting.

3.7. Loss Function and Optimization

We use standard cross-entropy loss for multi-class classification. Optimization is performed with the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$) and a learning rate of 5×10^{-4}. A linear warmup schedule is used to avoid early divergence. Each training configuration was run for 5–10 epochs with a batch size of 16 (due to memory constraints) and checkpointed by validation accuracy on Google Colab's A100 GPU.

3.8. Evaluation Metrics

We evaluate models using accuracy, precision, and F1 score. All metrics are computed on the PlantDoc test split using standard scikit-learn definitions. Given that both the PlantVillage and PlantDoc datasets are highly imbalanced, we chose to report the weighted F1 score because it accounts for the support (i.e., number of true instances) of each class. This is especially important in real-world plant disease classification settings, where certain diseases—like healthy leaves or common infections—occur far more frequently than rare conditions. While macro-averaged F1 would give equal importance to all classes, it can be misleading when some classes have very few samples, as performance on these rare classes can disproportionately affect the metric. By contrast, the weighted average gives a more representative measure of overall model performance, reflecting the model's ability to correctly classify both common and rare classes in proportion to their real-world frequency in the dataset.

4. Dataset and Features

We used the PlantDoc dataset, which was introduced in 2020 [13], to finetune the baseline model. This dataset contains 2342 examples in the training split and 236 examples in the test split. We randomly split the training dataset further into a training and validation dataset with an 80/20 ratio (1873 training examples and 469 validation examples). The examples in this dataset covered 29 classes and were a subset of the 35 classes in the original PlantVillage dataset. The PlantDoc dataset is available at https://huggingface.co/datasets/agyaatcoder/PlantDoc, and the PlantVillage dataset is available at https://huggingface.co/datasets/GVJahnavi/PlantVillage_dataset. As demonstrated in Figure 4, PlantVillage images are reliably individual leaves, centered in the frame, against a plain, uniform, decently-lit background. On the other hand, as displayed in Figure 3, PlantDoc data range from stock images containing the fruit of the plant in addition to the leaves, leaves with messy and confusing backgrounds, and off-center samples, in addition to some clean, centered, well-lit images. Both the PlantVillage and PlantDoc datasets are heavily right-skewed and highly imbalanced, with long tails of underrepresented classes (see Figures 6 and 5). However, each class is placed in a relatively similar spot in each distribution (i.e. the relative proportion of healthy potato leaves in both datasets is about the same).

In order to work with the finetuned model, we preprocessed images in the same way that the model preprocessed images during finetuning. More specifically, we resized images to 256 x 256 pixels, took the center crop of 224 x 224 pixels, and normalized the pixel values. We did not extract handcrafted features. Instead, we leveraged the pretrained MobileNetV2 backbone to learn hierarchical visual features directly from the image data.

In order to sanity check our procedure, we overfitted the finetuning pipeline to a single example and verified the accuracy of the unmodified model we were experimenting with on the original PlantVillage dataset on which it was trained. This verification ensured that our data preprocessing pipeline was aligned with the original training regime, and that our implementation could recover near-perfect accuracy on a memorized training example.

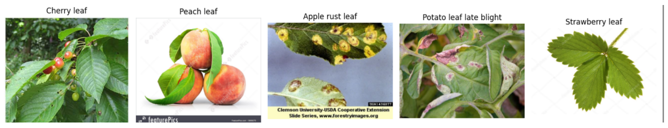


Figure 3. Example PlantDoc images

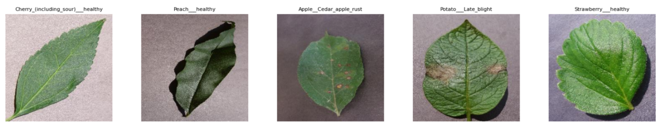


Figure 4. Example PlantVillage images

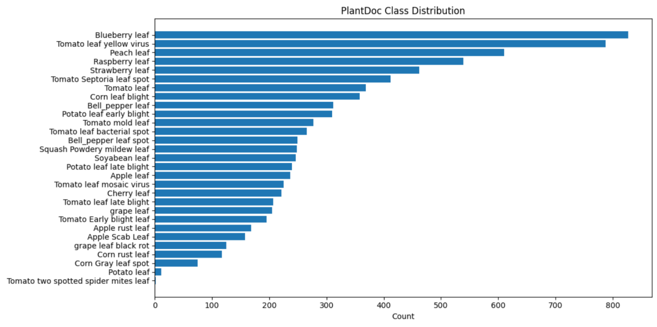


Figure 5. PlantDoc dataset distribution

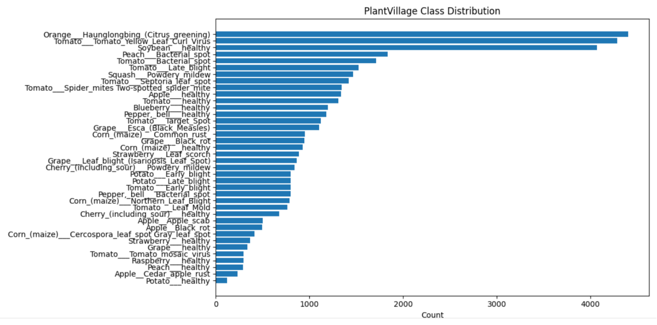


Figure 6. PlantVillage dataset distribution

5. Experiments, Results, and Discussion

5.1. Experiments

Outside of the baseline, we ran four experiments after conducting hyperparameter sweeps to estimate the correct hyperparameter range for each technique. For learning rates, we tried values between $1 * 10^{-8}$ and $1 * 10^{-2}$. Ultimately, for finetuning just the classifier using unaugmented data, we chose to train for 5 epochs with a learning rate of $1 * 10^{-4}$ and weight decay of 0.01 after experimenting with smaller learning rates and not seeing a significant decrease in training loss. To investigate what would happen if we mitigated the possibility of overfitting, we tried unfreezing all layers and finetuning all weights of the underlying model.

For finetuning all layers using unaugmented data, we found that a similar training setup but with a slightly higher learning rate of $5 * 10^{-4}$ was most effective. We used the same hyperparameters to finetune all layers using augmented data.

Finally, for multiphase fine-tuning, we used a lower learning rate of $5 * 10^{-5}$ to train the classifier head for 3 epochs, and then reduced the learning rate further to $2*10^{-5}$ to fine-tune a larger subset of model layers for another 3 epochs. In earlier experiments, we observed that a linear learning rate schedule yielded the best performance; however, for the second stage of multiphase fine-tuning, we adopted a cosine annealing schedule. This choice was motivated by the need for a smoother decay of the learning rate as the model adapted higher-level convolutional layers. The cosine schedule gradually reduces the learning rate in a non-linear fashion, which helps prevent catastrophic forgetting of previously learned representations while allowing more stable convergence during fine-tuning of deeper network components.

Strategy	**Accuracy (%)**	**Precision (%)**	**F1 (%)**
Zero-shot	34.55%	33.02%	29.88%
Fine-tuned classifier (unaugmented)	15.68%	23.02%	10.64%
Fine-tuned (unaugmented)	32.63%	31.33%	25.13%
Fine-tuned (augmented)	25.42%	24.15%	19.80%
Two-phase tuning	51.69%	54.15%	49.89%

Table 1. Comparison of model performances on the PlantDoc test set.

5.2. Results & Discussion

Table 1 summarizes the comparative performance across methods. The zero-shot baseline achieved a modest accuracy of 34.55%, showing the pretrained model's limited ability to generalize from the clean PlantVillage dataset to the more challenging PlantDoc images. The most common errors were misclassifications of corn diseases (although the model accurately identified that the leaves in question were corn leaves) and over-prediction of various types of healthy leaves as healthy blueberry leaves. See Figure 21 for a comparison of the healthy leaf failure case.

Finetuning only the classifier head resulted in a substantial drop in F1 (10.64%). There are several possible explanations for this. More layers may simply be needed to adapt to the target distribution. The classifier could also be overfitted to the small finetuning dataset, as is suggested by the rapidly decreasing training loss and more slowly decreasing validation loss. Additionally, there could be a significant misalignment between the frozen features learned from PlantVillage and the features needed to accurately predict PlantDoc. These issues could be exacerbated by the underlying class imbalance in the training and tuning data.

Surprisingly, fine-tuning all weights on unaugmented data (F1: 25.13%) outperformed finetuning on augmented data (F1: 19.80%), suggesting that naive augmentation did not provide robustness benefits in the low-data regime. We believe that the augmentation parameters may have been too aggressive, and certain ”standard” augmentation techniques like color jitter may have interfered with inference based on key features of the data. For example, the brown color of a leaf may have been critical to its correct classification, and color jitter randomization may have confused the model on this front. The best performance came from the two-phase strategy, which achieved 51.69% accuracy and 49.89% F1—substantially outperforming all other methods.

Figure 11 shows the training and validation loss curves for all strategies. While all methods achieved steady training loss reductions, the validation loss curves indicate that overfitting was common in single-phase setups. Two-phase fine-tuning produced the smoothest and lowest validation loss, reinforcing its effectiveness in learning transferable disease representations while avoiding overfitting.

Figures 23 and 22 show the confusion matrices for the zero-shot and two-phase models. The zero-shot model frequently confuses disease types within the same species (e.g., early vs. late blight in potatoes), while the two-phase model demonstrates significantly better class separation and reduced confusion.

Figures 12 and 13 display class visualizations for ”Potato with Early Blight” generated via gradient ascent. The zero-shot model attends to noisy background textures, while the two-phase model focuses more clearly on blight-characteristic lesion patterns, indicating more biologically meaningful representations.

Figure 17 presents saliency maps on a test image with powdery mildew. The zero-shot model's saliency is diffusely distributed, whereas the two-phase model sharply highlights mildew-covered areas, suggesting that it has learned to attend to disease-localized evidence more effectively.

Together, these quantitative and qualitative results support our hypothesis that carefully staged fine-tuning—particularly two-phase transfer learning—can substantially improve model generalization to noisy, real-world agricultural datasets.

6. Conclusion/Future Work

In this project, we investigated how well a MobileNetV2 model—pretrained on ImageNet and finetuned on the clean, lab-controlled PlantVillage dataset—could be adapted to perform robustly on the more challenging and diverse PlantDoc dataset. We evaluated several transfer learning strategies, including zero-shot inference, single-stage finetuning (with and without augmentation), and a staged, two-phase finetuning approach. Among these, two-phase finetuning proved the most effective, yielding the highest gains in accuracy, precision, and F1 score relative to the zeroshot baseline, while maintaining modest computational demands.

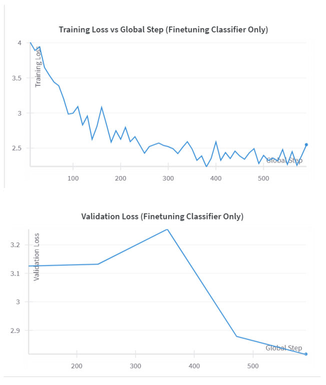


Figure 7. *
Fine-tuning Classifier Only

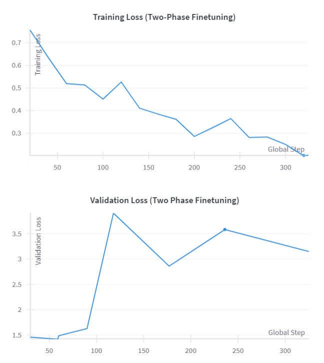


Figure 8. *
Two-Phase Fine-tuning

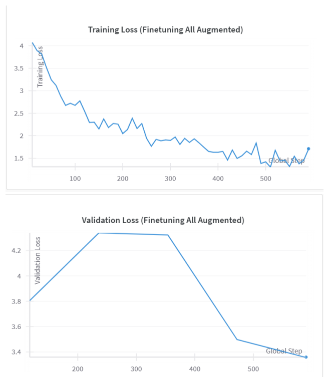


Figure 9. *
Fine-tuned (Augmented)

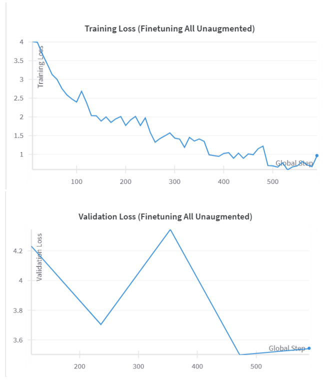


Figure 10. *
Fine-tuned (Unaugmented)

Figure 11. Training and validation loss curves for each fine-tuning strategy. Each column corresponds to one strategy, with training loss on top and validation loss below.

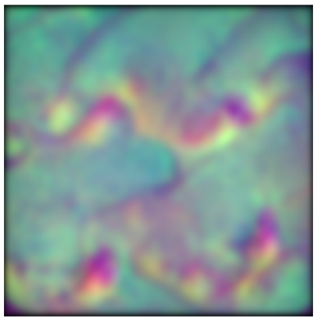


Figure 12. Class Visualization of "Potato with Early Blight" by Baseline Model

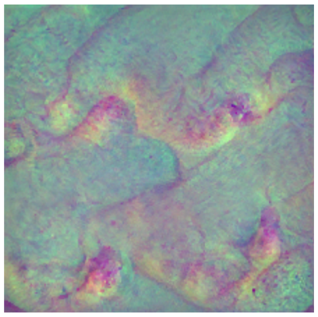


Figure 13. Class Visualization of "Potato with Early Blight" by Two-Phase Model

Figure 14. *
Original Image

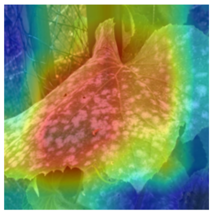


Figure 15. *
Zero-shot Model Saliency

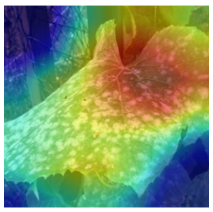


Figure 16. *
Two-Phase Model Saliency

Figure 17. Saliency maps comparing original image, baseline model, and two-phase finetuned model.

This approach allowed the model to better disentangle plant species recognition from disease classification, resulting in reduced confusion across visually similar disease classes. Qualitative tools such as saliency maps and class visualizations further confirmed that the model was learning to attend to domain-relevant visual features—such as lesion patterns, discolorations, and leaf textures—suggesting a degree of generalizable disease understanding.

Looking ahead, with more computational resources and

Figure 18. *
Blueberry Leaf

Figure 19. *
Peach Leaf

Figure 20. *
Apple Leaf

Figure 21. Zeroshot often misclassifies healthy fruit leaves as blueberry leaves.

time, a natural next step would be to systematically explore the tradeoff between accuracy and compute cost in low-data, low-resource scenarios. Another promising direction would be to assess transferability when the target dataset contains novel classes absent from the source dataset, which better reflects real-world deployment conditions. Finally, recent advances in multimodal models such as Vision-Language Models (VLMs), as well as generative approaches for synthetic data augmentation, offer intriguing possibilities for improving plant disease classification in under-resourced settings with minimal annotation effort.

7. Contributions & Acknowledgements

This project was completed by the sole author, Medhya Goel. This work was completed only for CS 231N and the author did not work with any other collaborators. Thanks to the teaching team for their guidance and feedback.

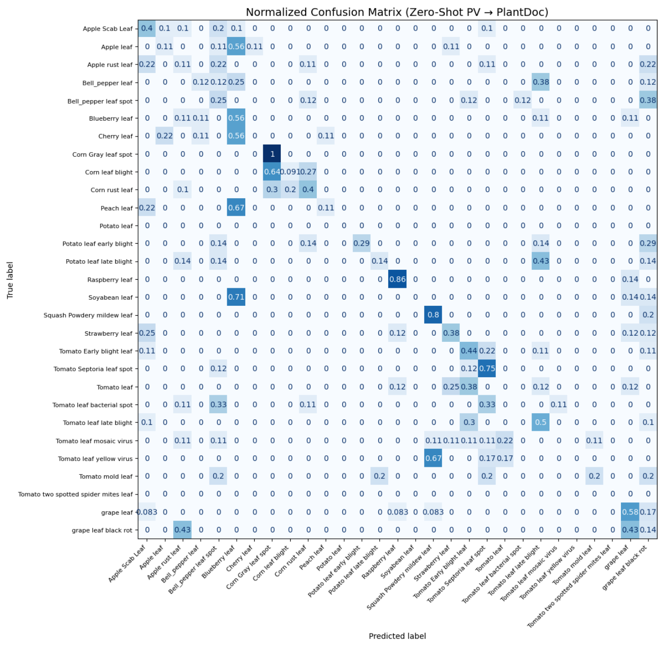


Figure 22. Baseline model confusion matrix on PlantDoc

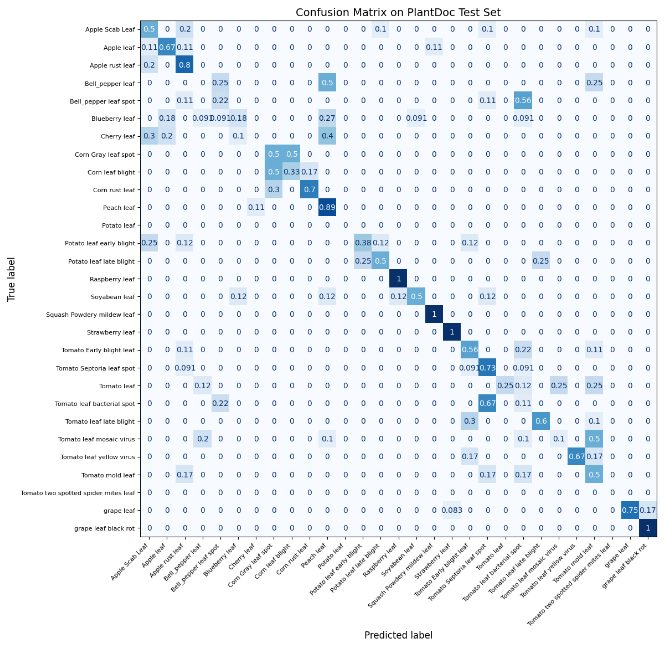


Figure 23. Two-Phase Model confusion matrix on PlantDoc

References

[1] A. Aarizou and M. Merah. Transfer learning for plant disease detection on complex images. In *2022 7th International Conference on Image and Signal Processing and their Applications (ISPA)*, pages 1–6, 2022. 2

[2] L. Anjarad. mobilenet_v2_1.0_224-plant-disease-identification. https://huggingface.co/linkanjarad/mobilenet_v2_1.0_224-plant-disease-identification, 2023. 3

[3] K. Ferentinos. Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, 145:311–318, 02 2018. 2

[4] Food and Agriculture Organization of the United Nations. Plant production and protection division, 2024. Accessed May 20, 2025. 1

[5] D. Hughes and M. Salathe. An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing. *arXiv preprint arXiv:1511.08060*, 11 2015. 1

[6] R. Kharbanda, Rajani, R. Pareek, and V. Ms. Crop monitoring: Using mobilenet models. *International Research Journal of Engineering and Technology*, 6, 05 2019. 2

[7] Y. Miao, W. Meng, and Z. X. Serpensgate-yolov8: an enhanced yolov8 model for accurate plant disease detection. *Frontiers in Plant Science*, 15, 1 2025. 2

[8] S. P. Mohanty, D. P. Hughes, and M. Salathé. Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, Volume 7 - 2016, 2016. 2

[9] National Institute of Food and Agriculture. Researchers helping protect crops from pests, 2021. Accessed May 20, 2025. 1

[10] B. Sambana, H. S. Nnadi, M. A. Wajid, N. O. Fidelia, C. Camacho-Zuñiga, H. D. Ajuzie, and E. M. Onyema. An efficient plant disease detection using transfer learning approach. *Scientific Reports*, 15, 5 2025. 2

[11] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks . In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, Los Alamitos, CA, USA, June 2018. IEEE Computer Society. 3

[12] W. Shafik, A. Tufail, C. D. S. Liyanage, and R. A. A. H. M. Apong. Using transfer learning-based plant disease classification and detection for sustainable agriculture. *BMC Plant Biology*, 24, 2 2024. 2

[13] D. Singh, N. Jain, P. Jain, P. Kayal, S. Kumawat, and N. Batra. Plantdoc: A dataset for visual plant disease detection. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, CoDS COMAD 2020, page 249–253, New York, NY, USA, 2020. Association for Computing Machinery. 5

[14] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic. Deep neural networks based recognition of plant diseases by leaf image classification. *Computational Intelligence and Neuroscience*, 2016(1):3289801, 2016. 2

[15] B. Wang, C. Zhang, Y. Li, C. Cao, D. Huang, and Y. Gong. An ultra-lightweight efficient network for image-based plant disease and pest infection detection. *Precision Agriculture*, 24:1836–1861, 4 2023. 2

[16] S. Z. M. Zaki, M. A. Zulkifley, M. M. Stofa, N. A. M. Kamari, and N. A. Mohamed. Classification of tomato leaf diseases using mobilenet v2. *IAES International Journal of Artificial Intelligence*, 9:290–296, 06 2020. 2

Package	**Version**
datasets	3.6.0
transformers	4.52.3
matplotlib	3.10.0
collections	builtin
PIL	11.2.1
torchcam	0.4.0
torchvision	0.21.0+cu124
torch	2.6.0+cu124
numpy	2.0.2
cv2	4.11.0
tqdm	4.67.1
wandb	0.19.11
os	builtin
google.colab	0.0.1a2
evaluate	0.4.3
datetime	builtin
sklearn	1.6.1

Table 2. Packages and versions used in the project.