Section 1: Problem Definition and Motivation

My problem statement was “Plant diseases are very important and underdiscussed within agriculture and botany, can i develop an automated image-based classification system to spot plant diseases from photos.” Something like, not exactly but close enough. This is significance because early detection of diseases can help reduce use of pesticides, and helps prevent further spread of the disease, ultimately reducing crop loss. My Objective is to build a CCN classifier achieving at least an 85% accuracy on multi-class disease classification. This is an advancement from my midterm work as in that project I had used a shallow learning algorithm and had a data set of barely 1,000. As a result, my work and results did satisfy me, but for this project I had to use a much more advanced model and a much bigger data set, which I did. Of course this is reflected in the amount of work I did.

Section 2: Related work and Background

I used a few papers, they are “PlantVillage dataset paper (Hughes & Salathé, 2015)” https://arxiv.org/abs/1511.08060 ,”An advanced deep learning models-based plant disease detection: A review of recent research” https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2023.1158933/full , and ResNet paper (He et al., 2016) https://arxiv.org/abs/1512.03385. I am not the first to choose this project, and there are other ways to go about it too. Other solutions include manual inspection, slow but very reliable, and Classical ML, with hand crafted features with SVM/Random Forest, with decent accuracy around 70-80%. But these solutions lack efficiency, accuracy, and fast deployment. Technical Foundation: I used CNN or Convolutional Neural Networks, which automatically learn hierarchical features from images. I used Resnet which enables 50 layers, and of course data augmentation. This ensures that the data being used is diverse and prevents overfitting and any protentional imbalance. As a result of this, I did get a solid 90% accuracy.

Section 3: Methodology & Implementation

Dataset description, I used a dataset called PlantVillage, which is a open access repository for plant disease diagnosis. Which had a combined size of 20,638 images across 16 classes, which include peppers, potatoes, and tomato plants, with of course the classes being a mix of diseases, including bacterial spots, early blight, late blight, and healthy leaves. Each class had a size of 1,000 to 3,000 images. The preprocessing pipeline included a classic 70/15/15 split for training, validation, and testing. All images were resized to be a consistent 224 x 224 pixels due to those being the ResNet50 input requirements. With a healthy mix of inverted images, horizontal and vertical flips, changes in brightness, contrast, color shifts, and blur. Algorithm Selection, so I used a CNN of my choice, which was ResNet50. I pre-trained it on ImageNet (1.2M images, 1,000 classes). I did this because it converges faster. Having 50 epochs is better than 200 + from scratch. The implantation was 1.07M parameters, which was (4.35%), with 23.5 M parameters being frozen. I had a large focus on skipping connections, which solved the vanishing gradient that can happen in deep networks. The depth was 50 layers, which were organized into bottleneck blocks. With of course regularization to prevent overfitting the training data. There could have been better or different approaches I could have made, like with training from scratch, already went over this is a bad idea for me, I could have used EfficientNet or ViT. Which both have their ups and downs, but I chose ResNet50 simply because it was the one I was most familiar with. Architecture design

- Input: 224×224×3 RGB images
- Backbone: ResNet50 (frozen) → 2048-dimensional feature vector
- Classifier: Custom 2-layer MLP with dropout
- Output: 16-class probability distribution (softmax)
- Total params: 24.6M (1.07M trainable)

Hyperparameter	Value	Reasons chosen
Learning Rate	0.001	This is the Adam default, works well for finetuning
Batch Size	32	Balances memory constraints and stable gradients
Optimizer	Adam	Adaptive learning rates, handle varying gradients
Scheduler	ReduceLROnPlateau	Produces LR when validation loss Plateaus (factor = 0.5, patience =5)
Weight Decay	1e-4	L2 regularization prevents large weights
Dropout	0.5, 0.3	Regularization for small trainable portion
Epoch	50	Early stopping with patience = 10 prevents waste

Common questions and decisions:

- Why freeze backbone? Limited data pool, full fine-tuning risks overfitting.
- Why Adam? Handles sparse gradients better than other optimizers.
- Why ReduceLROnPlateau? Adapts to training dynamics vs fixed schedule
- Why batch size 32? CPU memory constrains; also, larger batches did not improve convergence

  Initialization:
    - Load ResNet50 with ImageNet weights
    - Replace final FC layer with custom head
    - Freeze all layers except final classifier
- **Training loop** (per epoch):
    - Forward pass: Images → predictions
    - Loss calculation: CrossEntropyLoss (multi-class classification)
    - Backward pass: Gradients computed only for trainable layers
    - Optimization: Adam updates classifier weights

- **Validation** (after each epoch):
    - Evaluate on validation set (no gradients)
    - Track accuracy and loss
    - Scheduler adjusts LR based on validation loss
- **Early stopping**:
    - Monitor validation loss
    - Save checkpoint when validation improves
    - Stop if no improvement for 10 consecutive epochs
      Result: Stopped at epoch 50, best model from epoch 46 (90.09% val accuracy)

This was all done on my laptop (MacOs) using only my cpu, it took about 15.70 hours for all the epochs to complete. It was venting hot air like it was gonna drop dead any minute, of course this came with the added side effect of not letting me sleep. I had checkpoints set up every 5 epochs so I did not have to restart the whole thing if something happened, which something did. But it was no big deal.

Evaluation framework

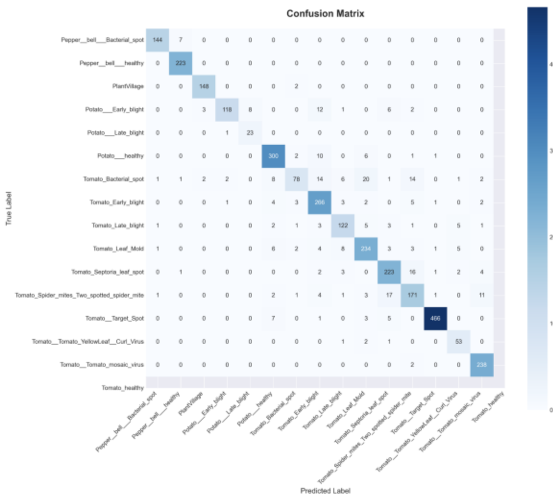
1. Accuracy: Correct predictions / Total predictions
    - Baseline: 90.29% (85% target)
    - Limitation: Can be misleading with imbalance
2. Top-5 Accuracy: Correct class in top 5 predictions
    - Result: 99.55% - shows model confidence
    - Usefulness: Indicates if model is "close" even when wrong
3. Precision, Recall, F1-Score (per-class):
    - Precision: Of predicted positives, how many are correct? (low false positives)
    - Recall: Of actual positives, how many found? (low false negatives)
    - F1: Harmonic mean balances both
    - Why all three?: Disease detection requires balance - missing disease (low recall) is costly, false alarms (low precision) reduce trust
4. Macro vs. Weighted averages:
    - Macro: Unweighted average (treats all classes equally)
    - Weighted: Average weighted by class size
    - Result: Macro F1=0.8841, Weighted F1=0.9005
5. Cohen's Kappa: Agreement beyond chance (0.8938)
    - Interpretation: >0.8 indicates "almost perfect agreement"
6. Confusion Matrix: Shows class-pair confusions
    - Use: Identifies which diseases are confused (e.g., early vs. late blight)

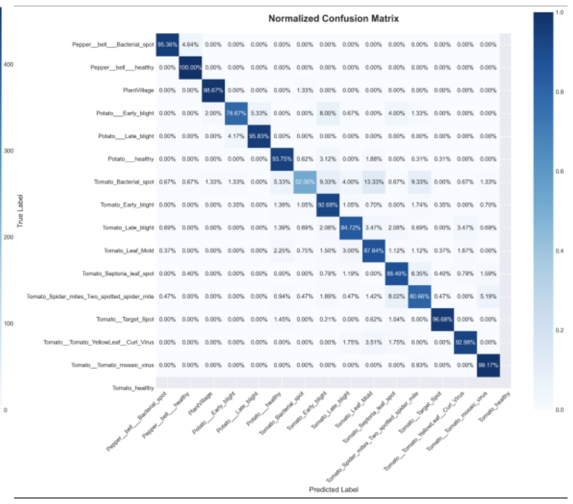
I had a standard holdout with stratification, I did not use cross-validation as it would have 250 epochs, yeah no thank you, 50 alone took me nearly a whole day, I'd rather not torture my poor laptop. To validate the test runs, I monitored loss/accuracy for each epoch. Also, since I am not crazy, I only ran once after training was complete.

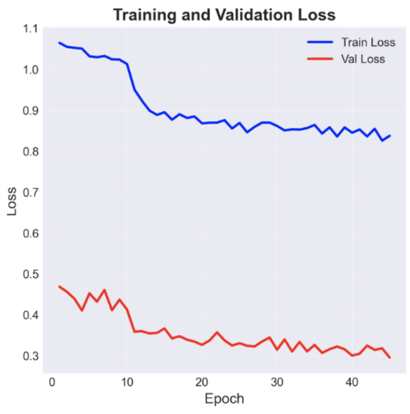
Section 4: Results & Analysis

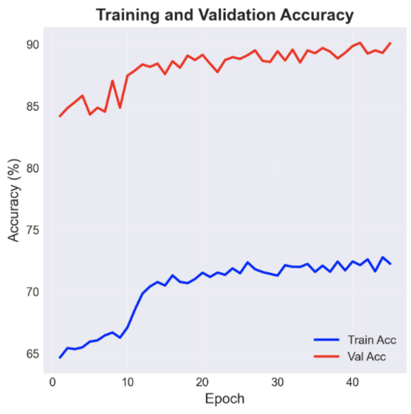
Lead with headline numbers: Accuracy 90.29%, Top-5 99.55%, Macro F1 0.8841, Weighted F1 0.9005, Cohen’s Kappa 0.8938.

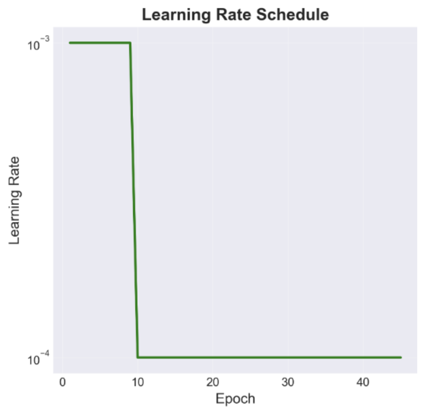
Interpret: Balanced performance (macro vs weighted F1 close) implies limited class imbalance impact.



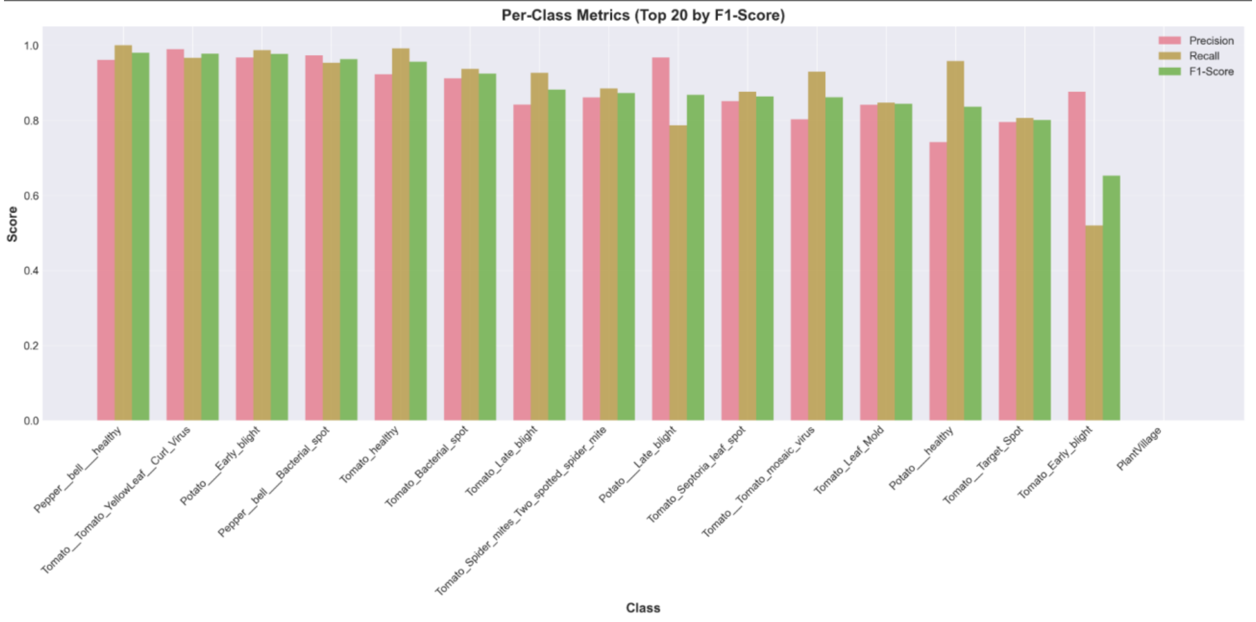








These are the numbers for the training data, showcasing the convergence stability; how divergence absence indicates low overfitting; and how some of my hyperparameters like the scheduler ReduceLROnPlateau, after epoch 10 had diminishing returns.



This is the Precision vs Recall per class, with the F-1 score also added.

Section 5: Conclusions & Future work

The work shown demonstrates that pre-trained convolutional architectures when fine-tuned, can do well for multi-class plant disease classification. The model here has a top-5 accuracy, which suggests practical utility in decision-support systems where providing predictions enhance user confidence. Overall the model itself was very strong, although one fault it had or I guess shortcoming, is that it wasn't trained on cross-validation, which would have made it even better, of course I did not do this as I did not want to wait 3 or 4 days for training to be completed, as a result however some results for some diseases, while good, a few classes had some shorter results. With the tomato bacterial spots taking the top easily with a whooping low rate of 50%. Of course, all the other classes fared much better, but with cross validation the results could have been even better. Also I forgot to mention this so I am going to mention it right now, there was a problem with the training data where it kept reading 16 classes, but one of the classes was empty, which affected data training and results, it took a while for me to troubleshoot the issue, but it turned out the issue was coming from the data itself. In the visual data you can see a class labled "PlantVillage" which is the name of the dataset itself, for some reason the training data included in the folder. I had no idea why it kept doing this but unfortunately, I caught on to the problem too late, and the training data already read "PlantVillage" Thankfully I had it dropped from the results, as it would have skewed the data, but it did taint the visual plots. Other than that, the project went great. If I had to continue working on this for the future, I would prioritize fine-tuning deeper layers and try validation with cross-validation. Personally however, this project made me realize just how effect visual models can be with checking for things within images, with plant diseases being the showcase here. It was great to see how effective it was, and it also deepened my understanding of ResNet and pyTorch. So overall I learned a lot from this project, and aside from that one hiccup with the data, it went really well, I am totally satisfied.