Computer Vision Project: Plant Disease Detection

Patrick Guerin

April 5, 2025

1 Introduction

For my final project, I developed three models for the detection of plant disease. I utilized the Plant Village dataset available on Kaggle, which contains over 150,000 images across 13 plant species, each exhibiting various disease conditions. I will use several pre-trained models from the PyTorch library to build and evaluate different classifiers by comparing there performance on different metrics.

2 The Dataset and Data Augmentation

As previously mentioned, the Plant Village dataset comprises over 150,000 images of various plant species exhibiting a wide range of diseases. In addition to the original images, the dataset provides grayscale versions and segmented copies where the background has been removed. However, since the the grayscale and segmented images are copies of the original, we can only use a third of the dataset to train on. These images are organized into folders by class, where each class corresponds to a specific plant species and disease combination—-examples include *Apple Black Rot* or *Blueberry Healthy*. In total, the Plant Village dataset consists of 38 distinct classes spanning 13 plant species.

In order to effectively train a model, the dataset must be separated into test, train and validation subsets. I used the Python package `split-folders` to divide the data into 80% training, 10% validation and 10% testing split. This tool requires an input directory, an output directory, and a specified ratio, and it handles the partitioning accordingly.

For data augmentation, I applied a series of transformations to the training data in order to make the data invariant to common variations like rotations and cropping. Specifically, these transformations included a random rotation, a random horizontal flip, and random resizing and cropping and normalization of the image tensor. For the testing and validation data only the normalization was applied.

Finally, the training, validation, and test datasets were loaded using PyTorch's `torchvision.datasets.ImageFolder` class. The resulting datasets were passed into data loaders using `torch.utils.data.DataLoader` with a batch size of 32. The training set was shuffled to further improve generalization during model training.

3 Building The Models

3.1 Training Method and Module

In order to begin training, I first implemented a training framework. I adapted the the `CIFARModule` from lab 5 and modified it to record accuracy and loss at the test, train and validation steps.
Since plant disease classification is a multi-class classification problem, I used the cross-entropy loss function. Cross-entropy is defined as:

$$H(p|q)) = -\sum_i p_i log(q_i)$$

where p_i is the true probability distribution and q_i is the predict probability distribution for class i. this loss function is well suited for for classification tasks as it quantifies the difference between the predicted and actual distributions, which encourages confident correct predictions.
The choice of optimizer is also important, since the optimizer must be both efficient and effective. I selected AdamW, an enhanced version of the Adam optimizer that introduces a decoupled weight decay term. This addition prevents weights from growing too large, reducing the likelihood of overfitting. AdamW retains the benefits of Adam such as momentum and adaptive learning rates while improving generalization.

I further customized the trainer method to include two additions:

1.) A CSV logger was added using PyTorch Lightning to save and organize training metrics.

2.)An early stopping callback was add to halt training once validation performance plateaued, again to mitigate overfitting.

Additionally, before training, I appended a dropout layer and a bottleneck layer to each model to improve regularization. Since the selected models are pre-trained on large-scale image datasets, I only trained the classification layer to prevent overfitting.

3.2 Picking Models

Given the potential application of plant disease classification in mobile devices, model efficiency and computational cost are important considerations. For this reason, I selected three lightweight but effective pre-trained convolution neural networks: ResNet18, MobileNet V3, and EfficientNet. Each model offers a balance between accuracy and efficiency, making them suitable for use on resource limited hardware.

ResNet18 was the first model I evaluated. It is composed of 18 layers that utilize residual blocks, a concept used to address the vanishing gradient problem in deep neural networks. Residual Blocks can be expressed as:

$$y = F(x) + x$$

where x is the input and $F(x)$ is the residual function (this is composed of convolution layers), and y is the output of the block. This architecture allows the model to learn the residual functions mapping instead of the direct mapping, which improves convergence and allowing deeper networks to be trained efficiently. ResNet18 is the smallest member of the ResNet family and demonstrates strong performance for computer vision tasks making it a strong candidate.

The second model evaluated was MobileNet V3, which was specifically designed for mobile applications. It incorporates several key innovations to boost performance such as, depthwise separable convolutions, inverted residual blocks, squeeze-and-excitation (SE) modules, and the hard-swish activation function. Inverted residual blocks expand the feature space before compressing it again, as opposed to traditional residual blocks that narrow the feature space before expanding. These design choices significantly reduce the number of parameters enhancing efficiency. MobileNet V3 trades a slight reduction in accuracy for improvements in speed, making it highly suitable for use on mobile devices.

Lastly, I tested EfficientNet, a model family developed by Google. EfficientNet builds upon the design of MobileNet but introduces compound scaling, a method for scaling a models width, depth and resolution using a set of predefined coefficients. compound scaling can be summarized as:

$$depth = \alpha^n, width = \beta^n, resolution = \gamma^n$$

for some scaling factor n and constants α, β and γ found through neural architecture search. EfficentNet's performance and accuracy makes it particularly appealing applications like plant disease classification.

4 Results

4.1 Performance Metrics

To evaluate the models I used four performance metrics: confusion matrix, accuracy, recall and F1-score.

The confusion matrix provides a class by class breakdown of performance, highlighting both accurate predictions and common misclassifications. Although confusion matrices are typically annotated, I omitted this step due the large number of classes making it difficult to read.

Accuracy, the ratio of correct predictions to total predictions, is a commonly used metric but can be misleading in the presence of class imbalance as is the case with the Plant Village dataset. Some classes contain over 5,000 images, while others have fewer than 400, potentially biasing the metric toward larger classes.

Recall measures the model's ability to correctly identify positive samples. This metric is especially important in the context of plant disease detection, where failing to identify a diseased plant could have consequences. For instance, if the model fails to recognize a diseased plant, it could spread affliction to other plants.

The F1-score, provides a balanced metric that considers both false positives and false negatives. It is particularly useful when both types of errors are critical, as in this classification task.

4.2 ResNet18 Metrics

The confusion matrix for ResNet18 indicates strong overall performance across most classes. However, there are two notable exceptions: Tomato Early Blight and Tomato Mosaic Virus.

In the case of Tomato Early Blight, a small but significant number of samples were misclassified as healthy tomatoes. This misclassification may stem from the subtle visual differences between diseased and healthy samples, making it difficult for the model to distinguish between the two.

For Tomato Mosaic Virus, the model correctly identified only about 40% -45% of the cases. Most remaining predictions were distributed across other tomato disease classes. This likely results from the class being underrepresented in the dataset and similarity to other tomato classes, making it harder for the model to learn features.

The validation accuracy plot shows rapid improvement during the first three epochs, peaking at around 91%, then stabilizing around 90%. This plateau is expected, given that only the classification layer was tuned while the rest of the model remained frozen. Overall, ResNet18 showed solid performance, with minor weaknesses for some classes.

ResNet18 Recall: 0.8722
ResNet18 F1-score: 0.8816
ResNet18 Accuracy: 0.9110

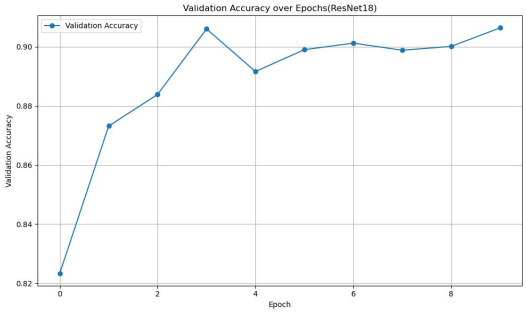

Figure 1: ResNet18 Accuracy Graph

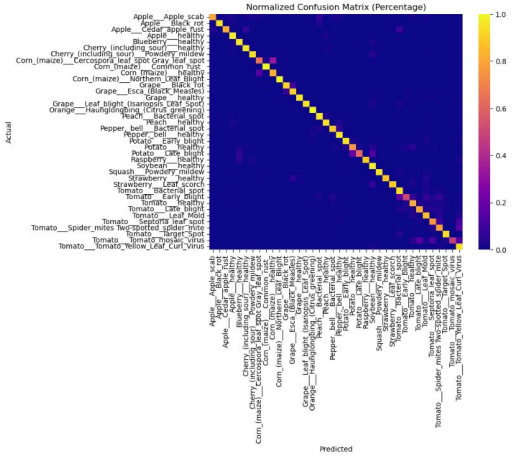

Figure 2: ResNet18 Normalized Confusion Matrix

4.3 MobileNet Metrics

EfficientNet achieved the best performance among the models evaluated. The confusion matrix shows consistently accurate predictions across nearly all classes, with a single noticeable weaknesses.

The only somewhat significant confusion occurred between healthy corn and corn affected by gray leaf spot, which is consistent with the other models and likely due

to high visual similarity between the two classes.

The validation accuracy curve shows a sharp increase between epochs 1 and 2, a temporary dip between epochs 2 and 6, followed by a final jump to peak performance. Training halted after the seventh epoch due to the early stopping condition, indicating that further improvements were unlikely. Overall, EfficientNet stands out for its high accuracy, recall, and generalization capability, confirming it as the most effective model for this classification task.

MobileNet Recall: 0.8632
MobileNet F1-score: 0.8726
MobileNet Accuracy: 0.9073

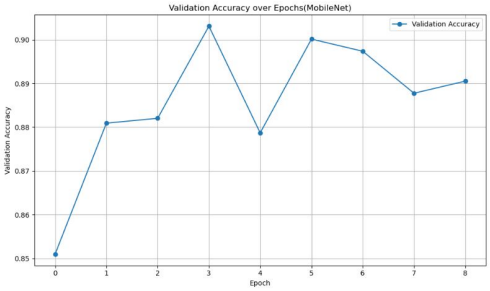

Figure 3: MobileNet Accuracy Graph

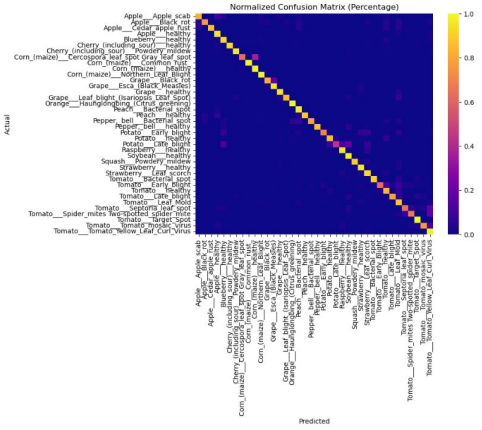

Figure 4: MobileNet Normalized Confusion Matrix

4.4 EfficientNet Metrics

finally we look at EfficientNet looking at EfficientNet Confusion matrix we see that EfficientNet preforms best out of all the models. There are not really specific classes that under preform, the only thing of note is that the model will sometimes classify healthy corn as corn with gray leaf spot and vice versa, however this is also a problem with both ResNet18 and MobileNet so it is not a outlier with performance.

Next we need to look at the accuracy graph, here we see a sharp rise in accuracy from the first epoch to the second followed by a sharp drop from epochs 2 to 6 and

final a sharp jump back up to about maximum accuracy. The fact that the model truncated at the seventh epoch indicates that performance stopped getting better very quickly. Finally looking at the Accuracy, Recall and F1 metrics we again see that it out preforms both ResNet18 and MobileNet. Overall EfficientNet is a very strong model with great performance and speed.

EfficientNet Recall: 0.9135
EfficientNet F1-score: 0.9196
EfficientNet Accuracy: 0.9370

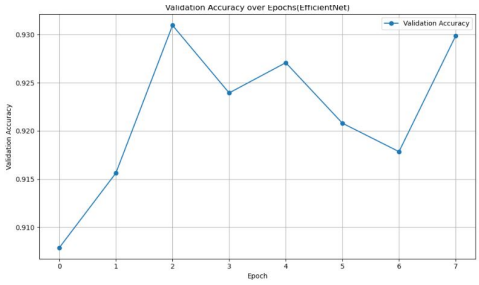

Figure 5: EfficientNet Accuracy Graph

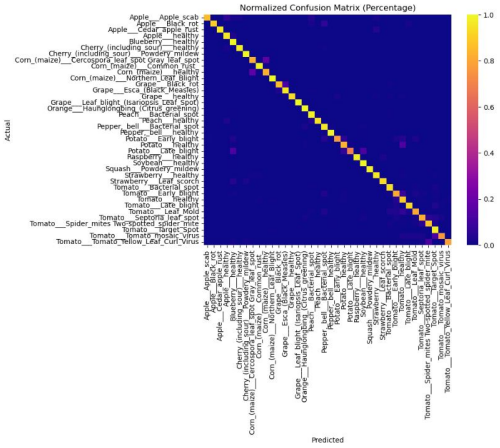

Figure 6: EfficientNet Normalized Confusion Matrix

5 Improvements and Conclusion

In this report, I outlined the development and evaluation of three deep learning models—ResNet18, MobileNet V3, and EfficientNet for the task of plant disease classification. I discussed the architecture and design rationale behind each model, along with their respective advantages and trade-offs. After conducting a comparative analysis based on multiple performance metrics, EfficientNet emerged as the most effective model, achieving the highest scores in accuracy, recall, and F1-score.

Despite these strong results, there are still opportunities for improvement. One major limitation lies in the Plant Village dataset itself. The dataset is imbalanced, with some classes containing over 5,000 images while others have fewer than 400. This imbalance can bias the model toward larger classes and reduce performance on underrepresented ones. A possible solution would be to reduce the size of larger classes.

Additionally, certain plant species in the dataset are represented by only one class (e.g., "healthy" or a single disease), which limits the model's ability to learn differences in plant health for certain species. A more diverse dataset with richer class distributions would improve robustness of the model.

Lastly, to better analyze class-level performance, the confusion matrix could be broken down into sub matrices by plant species, enabling more focused evaluation and error analysis for each speieces.

6 Resources

GitHub Repository