**4. Modeling**

**4.1. Choose model types**

For this project, we selected ResNet-18 as the primary model architecture. ResNet (Residual Network) is well-known for its ability to train very deep networks using residual connections, which help mitigate the vanishing gradient problem. Among the ResNet family, ResNet-18 offers a good trade-off between performance and computational efficiency, making it suitable for deployment in resource-constrained environments, such as mobile or embedded devices commonly used in agricultural monitoring.

The model was initialized with pretrained weights from ImageNet, leveraging the benefits of transfer learning to improve convergence speed and generalization on our crop disease dataset. The original classification head (fully connected layer) of ResNet-18 was replaced with a new linear layer matching the number of disease categories in our dataset.

While ResNet-18 was the main focus, alternative lightweight architectures like MobileNet and EfficientNet were considered for future work due to their smaller model sizes and faster inference speeds. These would be beneficial for real-time detection tasks on edge devices, but were not implemented in this phase of the project.

**4.2. Transfer learning using pretrained models**

To accelerate training and enhance model performance, we applied transfer learning by initializing the ResNet-18 model with pretrained weights from the ImageNet dataset. This approach leverages rich feature representations learned from a large-scale, general-purpose image classification task, which is particularly effective when working with limited or domain-specific datasets such as plant disease images.

In our implementation, we retained all convolutional layers of ResNet-18 to preserve the learned low- and mid-level visual features (e.g., edges, textures, patterns). The original fully connected (fc) classification layer was replaced with a new nn.Linear layer whose output dimension corresponds to the number of target classes in our dataset.

This fine-tuning strategy allowed us to adapt the model to our specific classification task (identifying healthy vs. diseased crops) while maintaining the benefits of the pretrained backbone. Training was performed on all layers of the network, allowing the model to adjust its parameters to the domain-specific characteristics of crop leaf images.

This use of transfer learning significantly reduced training time and improved generalization compared to training a model from scratch.

**4.3. Ensemble models for improved robustness**

In this phase of the project, ensemble techniques were not applied. The current implementation focused on training and evaluating a single model architecture, namely ResNet-18, using transfer learning.

**4.4. Train the model with augmented data**

The training was conducted for 15 epochs using the Adam optimizer with a learning rate of 0.001, batch size of 64, and input size of 224×224. The dataset was split with an 80/20 ratio for training and testing, and the training data was further divided into 80% for training and 20% for validation using PyTorch's random\_split.

During training, the model's performance was tracked across both the training and validation sets. Accuracy and loss values were recorded after each epoch. The best model checkpoint was saved based on the highest validation accuracy.

Below is a detailed summary of the training process:

Table 1: ResNet18 - Training and Validation Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Train Loss** | **Train Acc** | **Val Loss** | **Val Acc** |
| 1 | 0.16660 | 0.94753 | 0.09899 | 0.96857 |
| 2 | 0.07035 | 0.97697 | 0.03452 | 0.98840 |
| 3 | 0.05140 | 0.98318 | 0.03277 | 0.98909 |
| 4 | 0.03922 | 0.98707 | 0.03322 | 0.98906 |
| 5 | 0.03319 | 0.98926 | 0.06739 | 0.98118 |
| 6 | 0.02702 | 0.99128 | 0.02622 | 0.99200 |
| 7 | 0.02410 | 0.99229 | 0.09062 | 0.97553 |
| 8 | 0.01983 | 0.99343 | 0.02233 | 0.99231 |
| 9 | 0.01715 | 0.99439 | 0.03509 | 0.99027 |
| 10 | 0.01770 | 0.99430 | 0.02737 | 0.99171 |
| 11 | 0.01423 | 0.99565 | 0.02791 | 0.99079 |
| 12 | 0.01267 | 0.99612 | 0.01320 | 0.99637 |
| 13 | 0.01312 | 0.99589 | 0.03716 | 0.98975 |
| 14 | 0.01103 | 0.99660 | 0.01221 | 0.99603 |
| 15 | 0.01074 | 0.99639 | 0.03489 | 0.98967 |

To better visualize the numerical results, the training and validation performance are illustrated using the following plots:

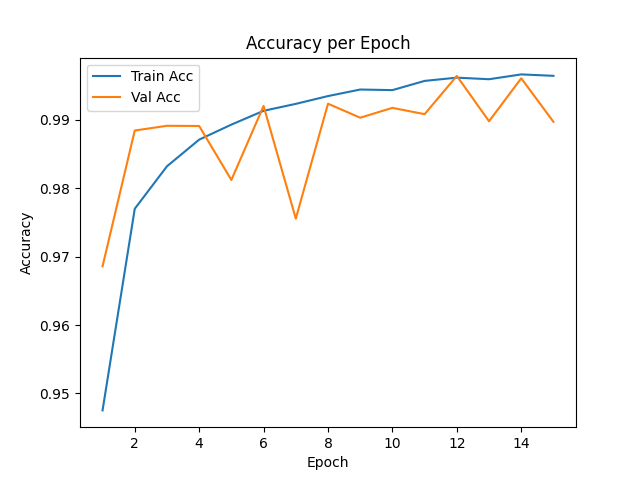


Figure 1: ResNet18 - Training and Validation Accuracy per Epoch

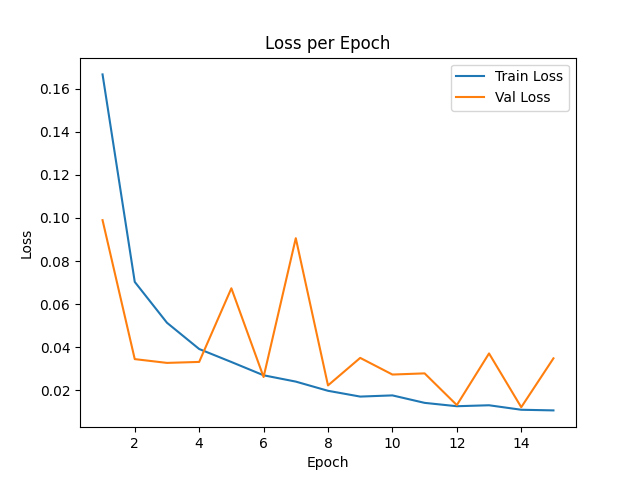


Figure 2: ResNet18 - Training and Validation Loss per Epoch

**4.5. Tune hyperparameters**

Upon analyzing the training and validation metrics from the previous section, we observed that while the model achieved exceptionally high accuracy on the training set (reaching over 99.6% by epoch 15), the validation accuracy plateaued and occasionally fluctuated, with slight drops in some epochs (e.g., epoch 5, 7, and 13). Moreover, the validation loss exhibited notable spikes, particularly at epochs 5 and 7, despite the continuous decrease in training loss.

These patterns indicate a mild overfitting phenomenon, where the model becomes increasingly tailored to the training data but generalizes less effectively to unseen validation samples. This imbalance suggests that the model might have learned noise or non-generalizable patterns from the training set, especially given its rapid convergence.

To address this, further tuning of hyperparameters is essential. Adjustments such as modifying the learning rate, experimenting with smaller batch sizes, increasing or reducing the number of training epochs, or implementing regularization strategies can help in enhancing generalization performance. Additionally, exploring architectural improvements or incorporating dropout layers could also reduce the risk of overfitting.

To mitigate the mild overfitting observed during the initial training phase and further enhance the model's generalization capability, a series of targeted adjustments were made to both the hyperparameters and model architecture. These changes are outlined as follows:

**4.5.1. Reducing the Batch Size**

* **What changed**: The batch size used during training was decreased from 64 to 32.
* **Why this helps**: A smaller batch size introduces more noise into the gradient estimation process, which can act as a regularizer and help prevent the model from overfitting. By reducing the batch size, the model is less likely to converge prematurely to sharp minima that generalize poorly on unseen data.

**4.5.2. Enhancing the Final Fully Connected Layer with a Multi-layer Structure**

* **What changed**: The original fully connected (FC) layer at the end of the ResNet18 model was replaced with a more complex structure. Instead of directly mapping the final feature vector to the class scores, an intermediate layer was inserted. This new structure included a linear transformation followed by batch normalization, ReLU activation, dropout regularization, and finally another linear layer to output class probabilities.
* **Why this helps**: The new FC structure increases the model’s capacity to learn richer representations, while also integrating mechanisms like Batch Normalization and Dropout to improve training stability and reduce overfitting. Dropout randomly deactivates neurons during training, forcing the model to be more robust, while Batch Normalization helps in smoothing the optimization landscape, accelerating convergence, and reducing internal covariate shift.

**4.5.3. Adding a Learning Rate Scheduler**

* **What changed**: A learning rate scheduler was introduced to dynamically reduce the learning rate after every 5 training epochs by a factor of 10.
* **Why this helps**: Gradually reducing the learning rate allows the model to take large optimization steps early in training and finer adjustments later on. This strategy prevents the model from overshooting optimal minima and encourages smoother convergence. Especially in the later epochs where overfitting typically starts to emerge, a smaller learning rate helps in fine-tuning the model more precisely without causing significant over-adaptation to the training data.

Here is the result after tuning hyperparameters and the model:

Table 2: Training and Validation Results After Tuning Hyperparameters and Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Train Loss** | **Train Accuracy** | **Val Loss** | **Val Accuracy** |
| 1 | 0.29320 | 0.91140 | 0.10726 | 0.96523 |
| 2 | 0.11777 | 0.96309 | 0.09479 | 0.96863 |
| 3 | 0.08205 | 0.97377 | 0.03590 | 0.98872 |
| 4 | 0.06391 | 0.97999 | 0.03738 | 0.98782 |
| 5 | 0.05343 | 0.98344 | 0.05575 | 0.98230 |
| 6 | 0.01160 | 0.99622 | 0.00741 | 0.99758 |
| 7 | 0.00600 | 0.99816 | 0.00711 | 0.99778 |
| 8 | 0.00439 | 0.99845 | 0.00793 | 0.99752 |
| 9 | 0.00338 | 0.99899 | 0.00677 | 0.99807 |
| 10 | 0.00318 | 0.99899 | 0.00804 | 0.99781 |
| 11 | 0.00212 | 0.99932 | 0.00621 | 0.99850 |
| 12 | 0.00170 | 0.99950 | 0.00558 | 0.99862 |
| 13 | 0.00138 | 0.99951 | 0.00583 | 0.99856 |
| 14 | 0.00121 | 0.99961 | 0.00592 | 0.99859 |
| 15 | 0.00124 | 0.99966 | 0.00555 | 0.99856 |

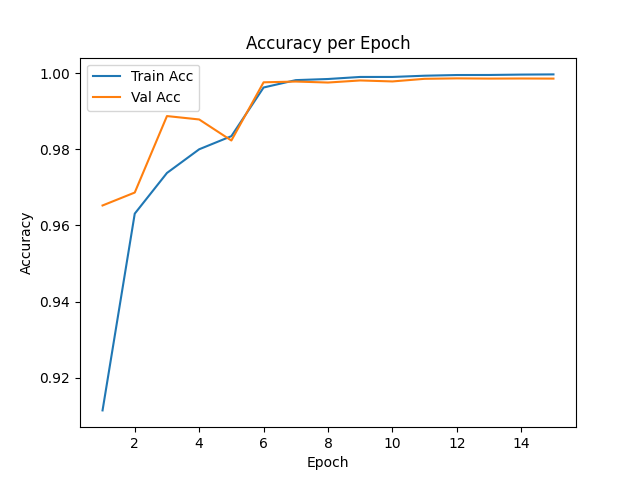


Figure 3: Training and Validation Accuracy After Tuning Hyperparameters and Model

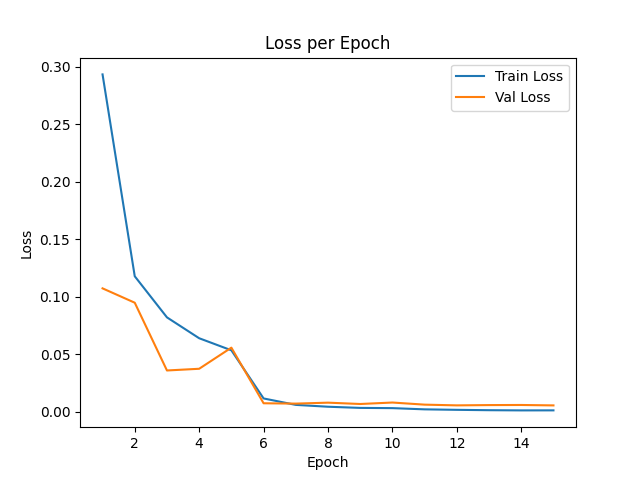


Figure 4: Training and Validation Loss After Tuning Hyperparameters and Model

The refinements made to the model architecture and training configuration have clearly contributed to improved overall performance. Compared to the previous results, the updated training exhibits:

* A significant reduction in validation loss, reaching as low as 0.00555 in the final epoch, indicating a much better generalization capability.
* Higher and more stable validation accuracy, consistently above 99.7% from epoch 6 onwards, peaking at 99.86%.
* Minimal gap between training and validation accuracy, suggesting that the model no longer suffers from the mild overfitting observed previously.
* Early and smooth convergence, with the model stabilizing around optimal performance after just 6 epochs.

In summary, these enhancements have effectively addressed the earlier signs of overfitting, leading to a more robust and well-generalized model that achieves excellent accuracy while maintaining stability across epochs.

**5. Evaluation**

**5.1. Use standard metrics: Accuracy, Precision, Recall, F1-score, Confusion Matrix**

To evaluate the performance of the plant disease classification model, we used two primary metrics: Accuracy and Confusion Matrix, which together provide both a global and detailed view of model performance.

* Accuracy measures the proportion of correctly predicted samples out of the total number of samples. This metric gives a concise overview of how well the model performs overall. For our dataset with 38 classes, a high accuracy indicates the model has learned to distinguish effectively between different plant diseases.
* Confusion Matrix gives a more granular analysis by displaying how predictions are distributed across the actual classes. Each row of the matrix represents the actual class, while each column represents the predicted class. The diagonal elements indicate correct predictions, and the off-diagonal elements reveal specific misclassifications.

By examining the confusion matrix, we can:

* Identify which diseases are frequently confused with others.
* Assess whether there is class imbalance in predictions.
* Prioritize improvements on specific class pairs where confusion is common.

These two metrics complement each other — accuracy provides a high-level summary, while the confusion matrix reveals the detailed classification behavior across all categories.

For Accuracy, the results have already been discussed in the Modeling section, where we observed significant improvements after tuning the model and hyperparameters. The final validation accuracy reached 99.85%, indicating strong overall performance.

As for the Confusion Matrix, the results are presented below. This matrix illustrates how well the model performs across all 38 disease classes by comparing the predicted labels against the true labels:

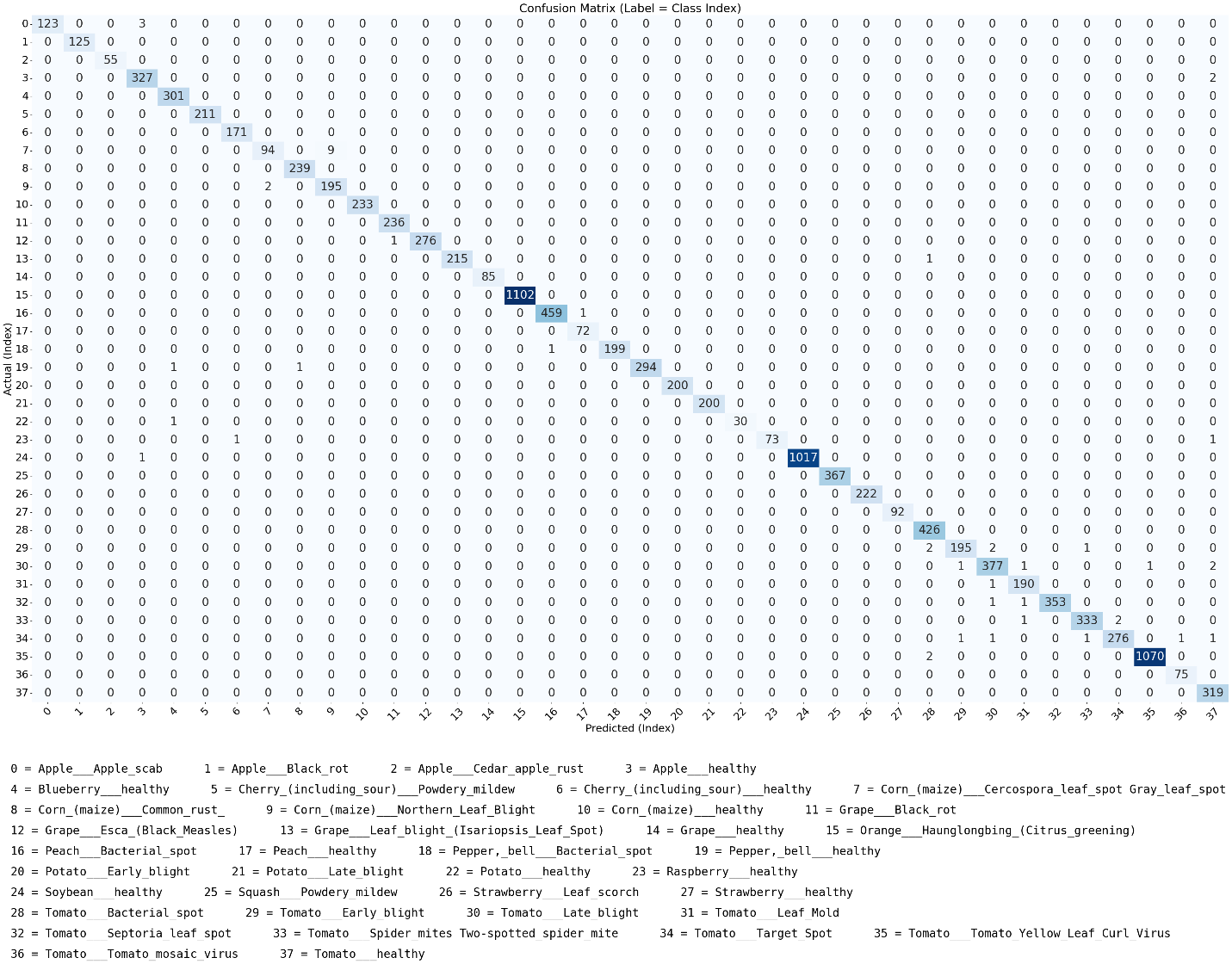


Figure 5: Confusion Matrix

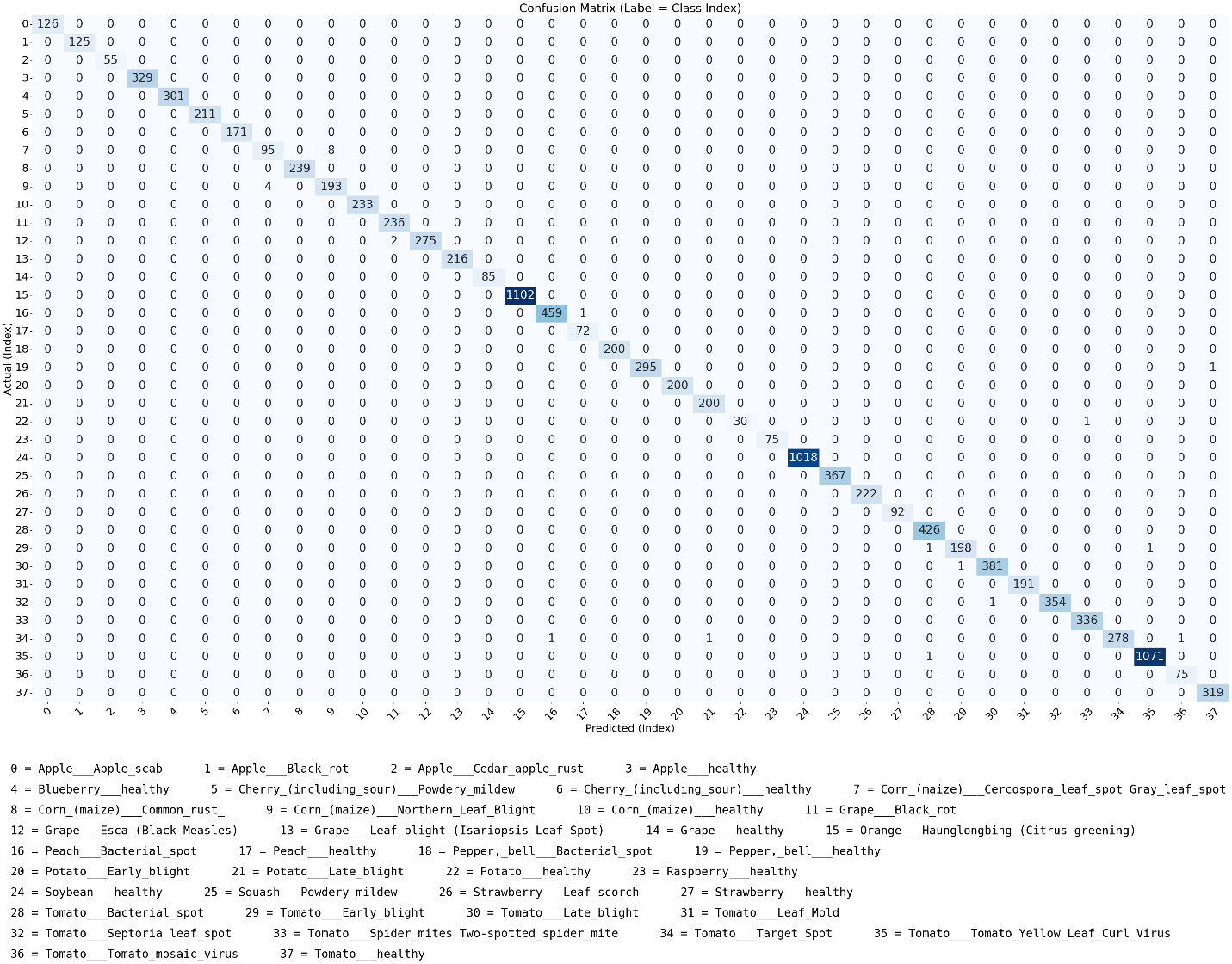


Figure 6: Confusion Matrix After Tuning Hyperparameters and Model

**5.2. Analyze misclassifications and confusion between diseases**

A deeper inspection of the confusion matrices from both training phases provides insight into how the model evolved in terms of class-wise accuracy and robustness.

**Before Tuning Hyperparameters and Model:**

In the initial training phase, the model already demonstrated strong convergence behavior with a high validation accuracy (approximately 99.64% at peak). The confusion matrix from this phase shows that the model generally distinguished well between the 38 plant disease classes.

However, due to slight overfitting—as evidenced by a gap between training and validation accuracy in the later epochs—there were still several isolated misclassifications.

The number of confused classes was not large, and the degree of confusion for each was relatively low (i.e., only a few samples per class were misclassified). Nonetheless, these inconsistencies indicated potential for further optimization.

**After Tuning Hyperparameters and Model:**

Following the application of several model and training refinements—including reducing the batch size, expanding the fully connected layer, and scheduling the learning rate—the confusion matrix reveals clear improvements:

* The total number of misclassified samples across all classes significantly decreased.
* Several class pairs that previously showed confusion are now correctly classified.
* The overall distribution is tighter along the diagonal, indicating more consistent and confident predictions across the entire validation set.

This improvement highlights the effectiveness of hyperparameter tuning in addressing overfitting and refining the model's ability to distinguish between subtle intra-class variations. In particular, the deeper and regularized fully connected head may have enabled the model to learn more discriminative representations, while the learning rate schedule helped stabilize training and avoid overshooting optimal weights.

Ultimately, the reduction in confusion between disease classes directly supports better diagnostic reliability, which is crucial for practical deployment in agriculture or field settings.

**5.3. Compare different models and configurations**

In addition to ResNet18, we also experimented with training the same dataset using MobileNetV2, a lightweight convolutional neural network architecture optimized for mobile and embedded vision applications. We kept the hyperparameters consistent across both experiments to ensure a fair comparison:

* Optimizer: Adam
* Learning Rate: 0.001
* Batch Size: 32
* Epochs: 15
* Input Size: 224×224

The training and validation results for MobileNetV2 are summarized below:

Table 3: MobileNet - Training and Validation Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Train Loss** | **Train Accuracy** | **Val Loss** | **Val Accuracy** |
| 1 | 0.1878 | 94.29% | 0.0682 | 97.90% |
| 2 | 0.0857 | 97.30% | 0.0732 | 97.82% |
| 3 | 0.0639 | 97.99% | 0.0289 | 99.09% |
| 4 | 0.0533 | 98.32% | 0.0223 | 99.37% |
| 5 | 0.0456 | 98.56% | 0.0281 | 99.02% |
| 6 | 0.0391 | 98.75% | 0.0193 | 99.44% |
| 7 | 0.0351 | 98.88% | 0.0137 | 99.55% |
| 8 | 0.0314 | 98.98% | 0.0151 | 99.54% |
| 9 | 0.0291 | 99.09% | 0.0139 | 99.56% |
| 10 | 0.0271 | 99.11% | 0.0136 | 99.60% |
| 11 | 0.0235 | 99.26% | 0.0127 | 99.61% |
| 12 | 0.0226 | 99.29% | 0.0114 | 99.63% |
| 13 | 0.0223 | 99.29% | 0.0189 | 99.40% |
| 14 | 0.0201 | 99.35% | 0.0152 | 99.61% |
| 15 | 0.0199 | 99.38% | 0.0077 | 99.75% |

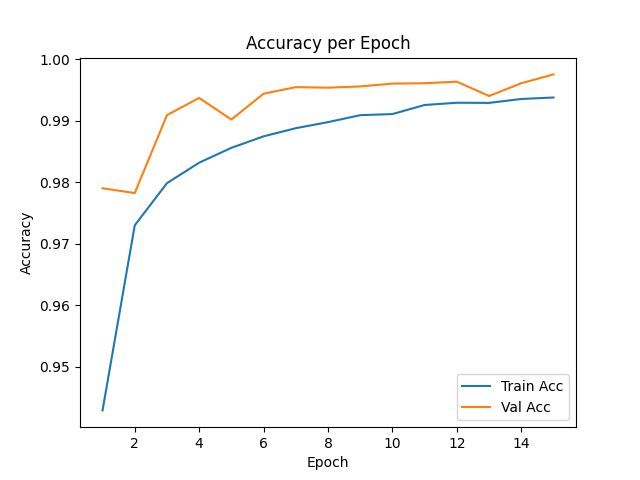


Figure 7: MobileNet - Training and Validation Accuracy per Epoch

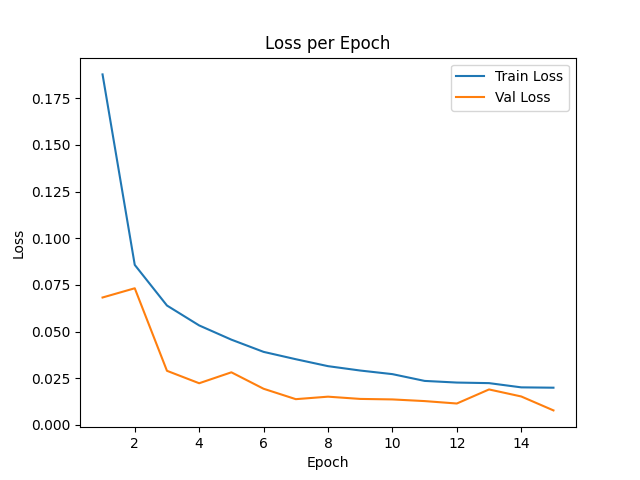


Figure 8: MobileNet - Training and Validation Loss per Epoch

Both ResNet18 and MobileNetV2 exhibited excellent performance on the plant disease classification task, achieving very high training and validation accuracy. However, ResNet18 consistently outperformed MobileNetV2 across several important aspects.

ResNet18 reached a slightly higher final validation accuracy of 99.86% (vs. MobileNetV2’s 99.75%) and demonstrated more stable convergence with minimal fluctuation in validation loss during the last few epochs. In addition, ResNet18 achieved lower overall validation loss, indicating stronger confidence in its predictions.

Moreover, ResNet18 was able to achieve this performance with smaller loss values earlier in training, suggesting better feature extraction and generalization from the outset.

On the other hand, MobileNetV2 still remains a very strong contender, particularly when considering resource constraints. It offers a highly competitive accuracy with significantly fewer parameters and lighter computation, making it well-suited for deployment on edge devices or mobile platforms.

In summary, while ResNet18 slightly edges out MobileNetV2 in terms of raw accuracy and stability, MobileNetV2 presents an excellent trade-off between performance and efficiency, and would be a strong choice in scenarios where computational resources are limited.