BILKENT UNIVERSITY CS 464 – INTRODUCTION TO MACHINE LEARNING

GROUP 23

FINAL PROJECT REPORT

PLANT DISEASE CLASSIFICATION

Group Members:

Kerem Er - 22102369

Asım Adil Can - 22103729

Gün Taştan - 22101850

Begüm Kunaç - 22103838

Erdem Pülat - 22103566

Introduction

Our objective is to develop a robust classification system that can accurately identify three distinct plant conditions:

- Healthy plants with no visible disease symptoms
- Plants affected by powdery mildew, characterized by white fungal marks on leaves
- Plants suffering from rust disease, identified by yellow, rust-like pathogenic fungal growth

The ability to automatically detect these conditions can significantly improve plant health monitoring, enable early intervention, and support more efficient agricultural practices. We aim to determine the most effective method for this classification task by comparing multiple machine learning and deep learning approaches.

Dataset Analysis

The dataset is structured into three primary categories that reflect different plant health conditions:

1. **Class Distribution**:
    - Healthy: Plants showing no visible disease symptoms
    - Powdery: Plants affected by fungal disease show white marks
    - Rusty: Plants displaying yellow, rust-like symptoms

Figure 1: Sample Images from Each Class

2. **Dataset Structure**:

    The dataset is pre-organized into three distinct directories:

    - Training set (1322 total images, 458 healthy, 430 rustic disease, and 434 powdery disease)
    - Testing set (150 total images, 50 healthy, 50 rustic disease, 50 powdery diseases)
    - Validation set (60 total images, 20 healthy, 20 rustic disease, 20 powdery disease)

This pre-defined split eliminates the need for manual data partitioning.

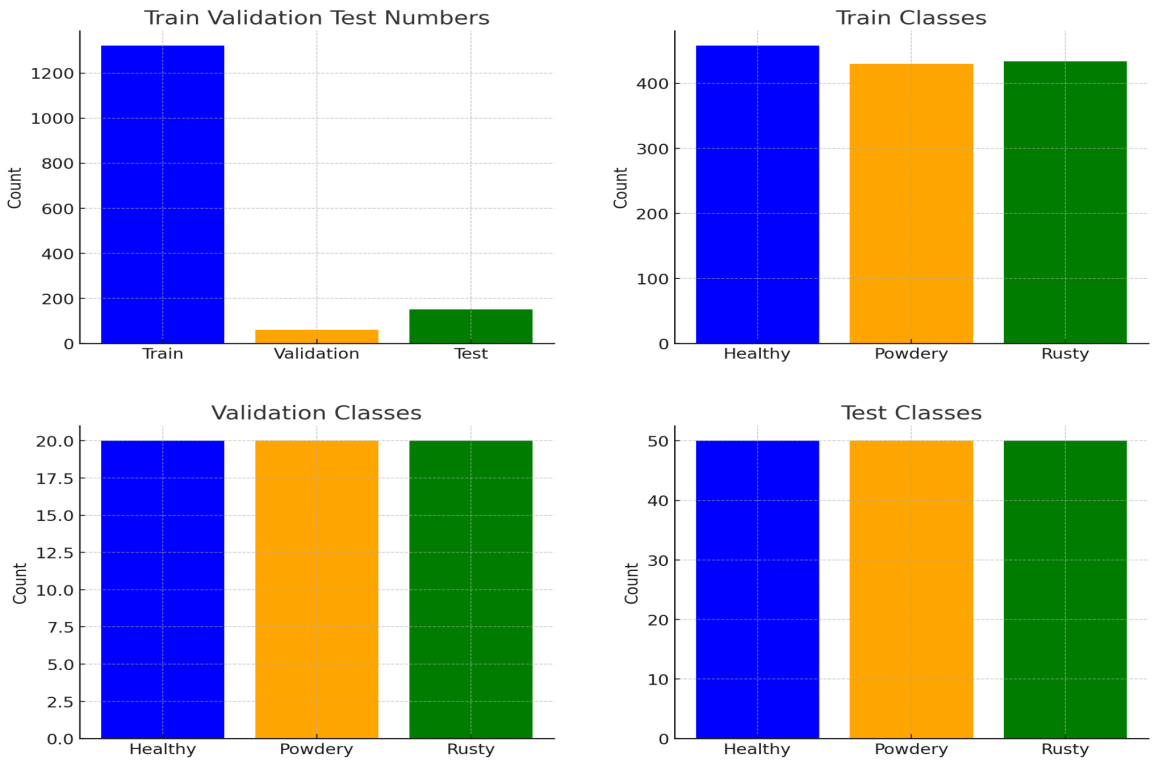


Figure 2: Bar Charts for Number of Images in Data Splits

3. **Image Characteristics**:

   Each image captures plant leaves exhibiting their respective health conditions The dataset contains visual representations of:

   - Normal, healthy leaf tissue
   - White fungal growth characteristic of powdery mildew
   - Yellow/brown discoloration indicative of rust disease

4. **Image Size**:
   - Each image has different sizes but every image has a size greater than 1024 x 1024 pixels.

Dataset Link: https://www.kaggle.com/code/vad13irt/plant-disease-classification/notebook

Dataset Preprocessing

Our preprocessing pipeline was designed to standardize the plant disease images and prepare them for optimal model training. The implementation utilizes PyTorch's torch-vision library and consists of several key steps:

Image Standardization

1. **Resizing**: All images are resized to 128x128 pixels to ensure:
    - Consistent input dimensions for the neural network
    - Reduced computational overhead
    - Preserved aspect ratio while maintaining important visual features
2. **Normalization**:
    - Images are converted to PyTorch tensors and normalized using mean (0.5) and standard deviation (0.5) values for each RGB channel
    - This transformation scales pixel values to the range [-1, 1]
    - Normalization helps achieve Faster convergence during training. Better numerical stability. Consistent input range across all images

In addition, for Adaboost, the normalization range was between [0,1], and NumPy was used instead of PyTorch.

Coding Environment

The coding environment, tools, libraries and repositories used for training and evaluation of the models for plant disease classification is as follows:

1. **Google Colab**:
    - The project was executed in Google Colab, providing a cloud-based environment with GPU support, enabling faster training and experimentation.
2. **Google Drive Integration**:
    - The dataset and model weights were accessed and stored via Google Drive to provide consistent data management and persistence.
3. **NumPy**:
    - Used for numerical computations and handling arrays, providing support for preprocessing data and data manipulation.
4. **PIL**:
    - Used for image processing tasks such as loading and manipulating image data.
5. **OpenCv:**
    - OpenCV was used for image processing tasks such as reading images, resizing them, edge detection before AdaBoost, and extracting color histogram features. It ensured that images were prepared consistently for feature extraction and model input.
6. **PyTorch**:
    - **torch**: The main library was used to define and train the models, using its dynamic computational graph and GPU acceleration.
    - **torch.nn and torch.nn.functional**: Provided modules for creating the model architectures, defining layers, activation functions, and loss functions (Cross-Entropy). [1]
    - **torch.optim**: Implemented optimization algorithms like Stochastic Gradient Descent and Adam optimizer for training. [1]

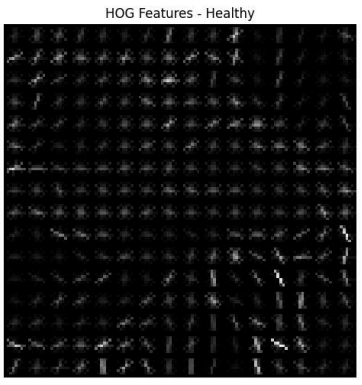
- **torch.utils.data.DataLoader**: Facilitated batch loading and shuffling of data for training, validation, and testing. [1]
- **torchvision.models.resnet18**(pretrained=True):

7. **Torchvision**:
    - **Datasets**: The ImageFolder class was used to load and structure the dataset from a directory of images.
    - **Transforms**: Applied preprocessing operations such as resizing, tensor conversion, and normalization to prepare images for model input.
8. **Matplotlib**:
    - Used for visualizing data and model performance metrics, helping in analysis and debugging.
9. **Scikit-Image**:
    - The Histogram of Oriented Gradients (HOG) feature extraction was performed using the hog module from Scikit-Image. This library enabled capturing texture and shape patterns from grayscale images, which were critical for classification tasks
10. **Scikit-Learn**:
    - The project utilized SVC for Support Vector Machine implementation, RandomForestClassifier for Random Forest classification, and AdaBoostClassifier for AdaBoost implementation. Performance evaluation was conducted using metrics such as classification_report, accuracy_score, f1_score, precision_score, and recall_score. Preprocessing steps included encoding labels with LabelEncoder and normalizing features with StandardScaler to ensure consistent scaling. Lastly, GridSearchCV was used for hyperparameter optimization.

Details of the Trained Models

Support Vector Machine

Support Vector Machines were employed in this project to classify plant images into three categories: Healthy, Rust, and Powdery. The model works by identifying an optimal hyperplane in the feature space that separates these classes. The input features for the SVM were derived from two key sources: Histogram of Oriented Gradients (HOG) and color histograms. HOG features captured texture and shape patterns from the grayscale images, while color histograms represented the color distribution across RGB channels. These features were concatenated to create a comprehensive representation of each image and were subsequently normalized using StandardScaler to ensure consistent scaling, improving the SVM's convergence and classification performance.



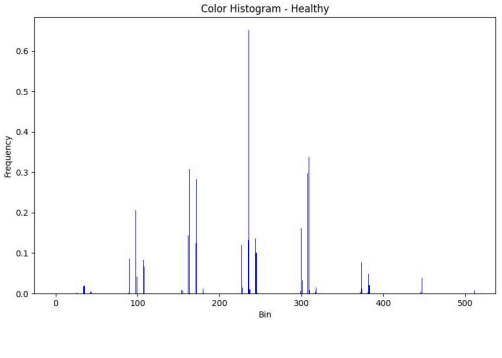


Figure 3: Visualization of Input Image, Extracted HOG Features, and Corresponding Color Histogram for a Healthy Leaf Sample

Random Forest

Random Forests were utilized in this project to classify plant images into three categories: Healthy, Rust, and Powdery. The model operates by constructing an ensemble of decision trees, where each tree is trained on a random subset of the data and features. Predictions from all trees are aggregated to make the final classification, improving accuracy and reducing the risk of overfitting. The input features for Random Forest were derived from two primary sources: Histogram of Oriented Gradients (HOG) and color histograms. HOG features captured essential texture and shape patterns from grayscale images, while color histograms represented the RGB channel distributions of the images. These features were concatenated to form a unified representation for each image, ensuring the model leveraged both structural and color information for classification. Additionally, StandardScaler was used to normalize the features, ensuring uniform scaling and enhancing model performance.

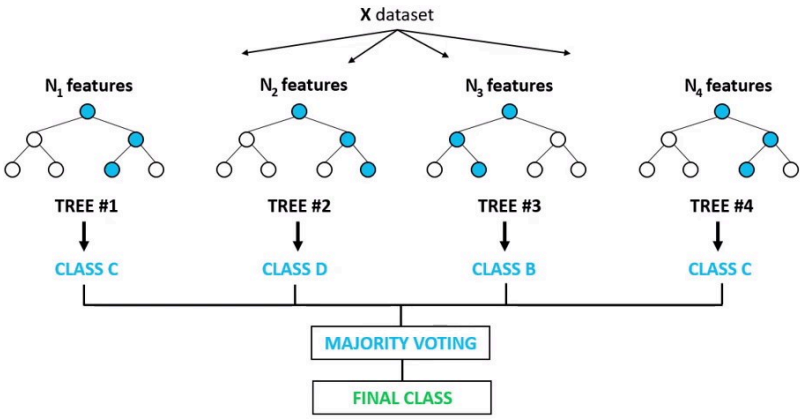


Figure 4: Illustration of the Random Forest Algorithm

Adaboost

The Adaptive Boosting (AdaBoost) algorithm mainly revolves around iteratively training a series of weak learners -commonly shallow decision trees- and adjusting the focus of each subsequent learner to the mistakes of the previous ones. At each iteration, a tree is fit to the data, with increased weights on samples that were misclassified by the latest model, thus forcing the next tree to prioritize those harder examples. The final prediction is obtained through a weighted vote of all weak learners [1].

In our experiments, we relied on decision trees with limited depth as the base learners for AdaBoost. These trees, often called “decision stumps” when max_depth is set to 1, are computationally inexpensive. The general process included initializing uniform weights on the training samples, fitting the first decision tree, measuring its weighted error, weight update, and a subsequent refit of the next tree to address the misclassified cases more directly. This cycle continued until the specified number of estimators (n_estimators) was reached. We did not implement an algorithm from scratch thanks to *scikit-learn*, which provided the necessary implementation for both the base estimator and the AdaBoost algorithm.

Convolutional Neural Networks (CNN)

The first deep learning model used for the plant disease classification task is Convolutional Neural Networks (CNNs). An input layer, convolutional layers, pooling layers, fully connected layers, and an output layer comprises the typical CNN. In order to extract hierarchical features, a CNN analyzes an image through a number of layers. The image is input as a 3D array of pixel values (width, height, and color channels). Initial layers create feature maps that emphasize the existence of features edges, textures, or basic patterns, as they move across the image. The feature maps are then given non-linear activation functions , which ensures the network is capable of learning complex, non-linear patterns. The model becomes computationally efficient when pooling layers are used to minimize the spatial dimensions of the feature maps while keeping the most important information. The resulting high-level feature representations are flattened into a 1D vector and supplied into fully connected layers after going through a number of convolutional and pooling layers. The output layer's softmax function, which generates a probability distribution across all potential classes, is used in these layers to transfer the retrieved features to particular class probabilities. The model uses backpropagation and optimization methods to modify its filters and weights during training to reduce classification error. Different architectures with varying numbers of convolutional layers and hyperparameter combinations are conducted in this project. The optimal design for achieving the best classification performance in the validation set is determined by comparing various architectures [1].

Specifications of CNN with 2 Convolutional Layers:

- Input images are resized to 128 x 128 pixels and normalization of the pixel values into the range [-1, 1] is done.
- First convolutional layer consists of 12 kernels with 5 x 5 size.
- Second convolutional layer has 18 kernels with 3 x 3 size.
- ReLU activation function is used,
- Max pooling with stride of 2, is selected.

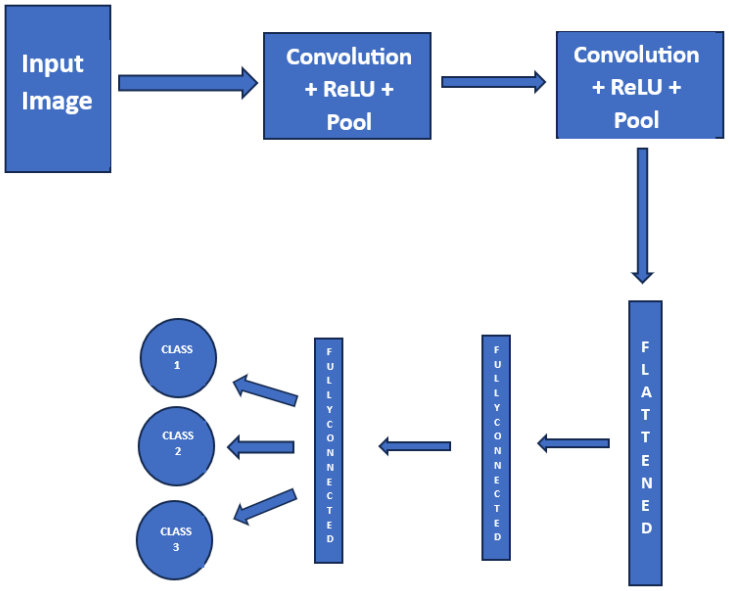


Figure 5: Convolutional Architecture with 2 Layers

Specifications of CNN with 3 Convolutional Layers:

- Input images are resized to 128 x 128 pixels and normalization of the pixel values into the range [-1, 1] is done.
- First convolutional layer consists of 12 kernels with 5 x 5 size.
- Second convolutional layer has 18 kernels with 3 x 3 size.
- Third convolutional layer has 24 kernels with 3 x 3 size.
- ReLU activation function is used,
- Max pooling with stride of 2, is selected.

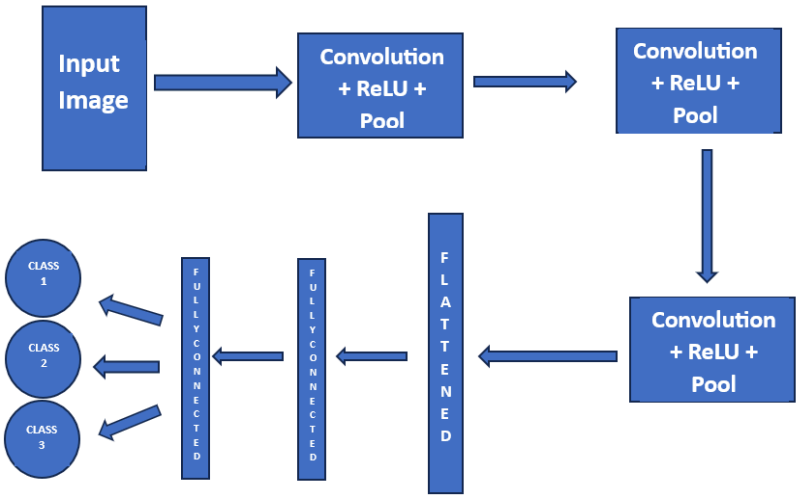


Figure 6: Convolutional Architecture with 3 Layers

Transfer Learning

Transfer learning is useful in cases of specific scenarios, including limited datasets or when data collection and labeling are an issue. In our dataset, we have 1,322 images in the training, 60 images in validation, and 150 images in the test set. This may be acceptable for some simple model training; still, we think that this might not suffice for the training of deep convolutional neural networks without any risks of underfitting or overfitting.

Also, pre-trained models are pre-trained for vast image datasets, one of which is ImageNet, consisting of more than one million labeled images falling into 1,000 categories. "Knowledge" learned by every pre-trained model is then learning to extract generic features, like edges, textures, and shapes, which themselves will easily generalize to newer tasks, in our instance, plant disease classification like rustic disease, powdery disease and healthy plants. Changing the output layer according to our number of classes (3) and connecting the last fully connected layer into this new output layer is realized. Freezing the previous layers and replacing the last layer of the model with new weights is expected to improve the accuracy of classification tasks because of the relatively small number of images in the dataset.

Transfer Learning Model Selections

We chose ResNet-18 several features such as it is a light network, consisting of 18 layers only, hence making the computation very efficient and capable of maintaining the capture of intricate features of any image. ResNet proposes shortcut connections as a mitigation of the vanishing gradient problem by allowing the model to scale in-depth and learn useful representations. Also, the architecture has already been shown to generalize well across datasets of different nature and is thus ideal for transfer learning on limited data. Below is an image of layer representation of ResNet-18 [2]

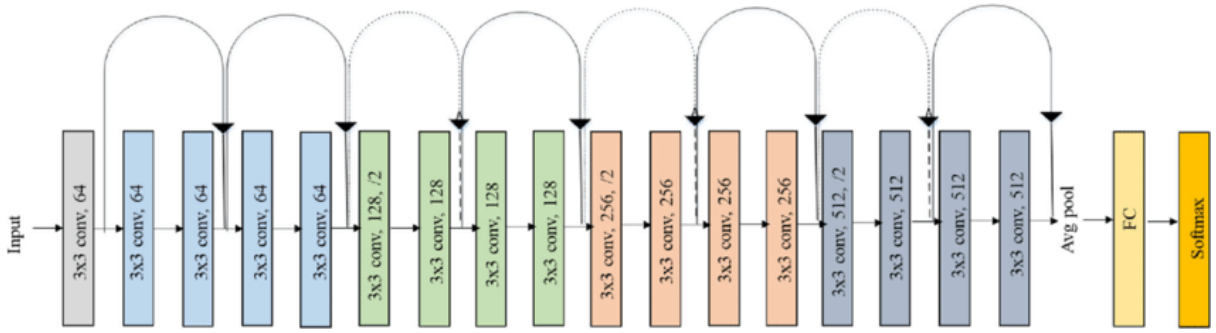


Figure 7: ResNet-18 layer illustration[3]

The next selected model is VGG16 which consists of CNNs in its architecture. There are 13 convolutional layers and 3 fully connected layers in this model. The model uses stacked convolutional layers followed with max pooling layers with progressively increasing depth. Accurate and precise predictions are obtained by the model's ability to learn complex hierarchical representations of visual characteristics due to this design [4].

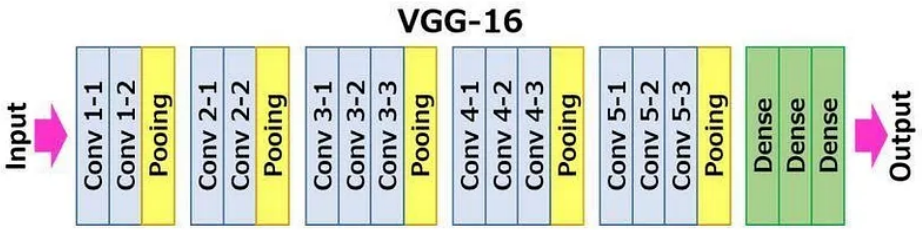


Figure 8: VGG-16 Layer Illustration [4]

Experimental Setup for Models

Support Vector Machine

The SVM model was fine-tuned through hyperparameter optimization using Grid Search, a systematic method to identify the best combination of parameters. A 5-fold cross-validation approach was adopted during this process, and the weighted F1-score was used as the evaluation metric to account for class imbalances. The optimization involved varying key parameters such as the kernel type, regularization strength, and kernel-specific parameters. The kernels explored included Radial Basis Function, Linear, and Polynomial kernels. Regularization strength was adjusted using the parameter C with values ranging from 0.01 to 100, balancing the trade-off between the model's margin width and classification accuracy. For the Polynomial kernel, the degree parameter was varied between 2, 3, and 4. For both RBF and Polynomial kernels, the gamma parameter, which controls the influence of individual training points, was tested with values including 0.001, 0.01, 0.1, 1, and "scale."

Parameter Grid Used:

- Kernel: Radial Basis Function, Linear, Polynomial
- C (Regularization parameter): 0.01, 0.1, 1, 10, 100
- Degree: 2, 3, 4
- Gamma: 0.001, 0.01, 0.1, 1, scale

The parameter grid explored in this study included a wide range of values to allow the SVM to adapt effectively to the dataset's characteristics. The kernels played a crucial role in defining the decision boundary's complexity, with Linear kernels handling simple linear separable data, Polynomial kernels capturing higher-order relationships, and RBF kernels handling non-linear data distributions by projecting it into a higher-dimensional space. The regularization parameter C helped control overfitting by penalizing models that fit the training data too closely, ensuring a balance between flexibility and generalization. Gamma, particularly for RBF and Polynomial kernels, determined how far the influence of a single training example extended, allowing the model to focus on either local or global data patterns depending on the dataset's nature. The parameters were adapted from sample image classification tasks and discussions available on platforms such as Kaggle and Stack Overflow.

Thus, hyperparameter tuning allowed for a systematic exploration of the model's performance across different configurations, ensuring an optimal balance between accuracy and generalization.

The best parameters for SVM, through Grid Search, were {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}. In this instance, the RBF kernel was selected because it handles nonlinear data by mapping it into a higher-dimensional space that will enable the model to learn complicated relationships in the dataset. The regularization parameter C = 10 gives a good balance between correct classification of the training data and the smoothness of the decision boundary, hence reducing overfitting. The gamma='scale' automatically sets the contribution of each training point according to the features of the dataset, and this allows the model to generalize effectively by focusing on relevant local and global patterns. This gave the best performance by using the flexibility of the RBF kernel, appropriate regularization and adaptive scaling of features to handle dataset characteristics.

On the validation dataset, the SVM achieved an accuracy of 78.33% and a weighted F1-score of 77.97%. The class-wise performance metrics reveal that the model achieved its highest precision (86%) and F1-score (0.90) for the Powdery class, reflecting its ability to reliably distinguish this category. However, the model faced challenges with the Rust class, achieving a recall of 65% and an F1-score of 0.68, indicating some misclassification in identifying this category. The overall macro-averaged and weighted metrics confirm consistent performance across the three classes with minimal imbalance effects.

Table 1: SVM - Validation Results

Class	Precision	Recall	F1-Score	Support
Healthy	0.75	0.75	0.75	20
Powdery	0.86	0.95	0.9	20
Rusty	0.72	0.65	0.68	20
Macro Average	0.78	0.78	0.78	60
Weighted Average	0.78	0.78	0.78	60

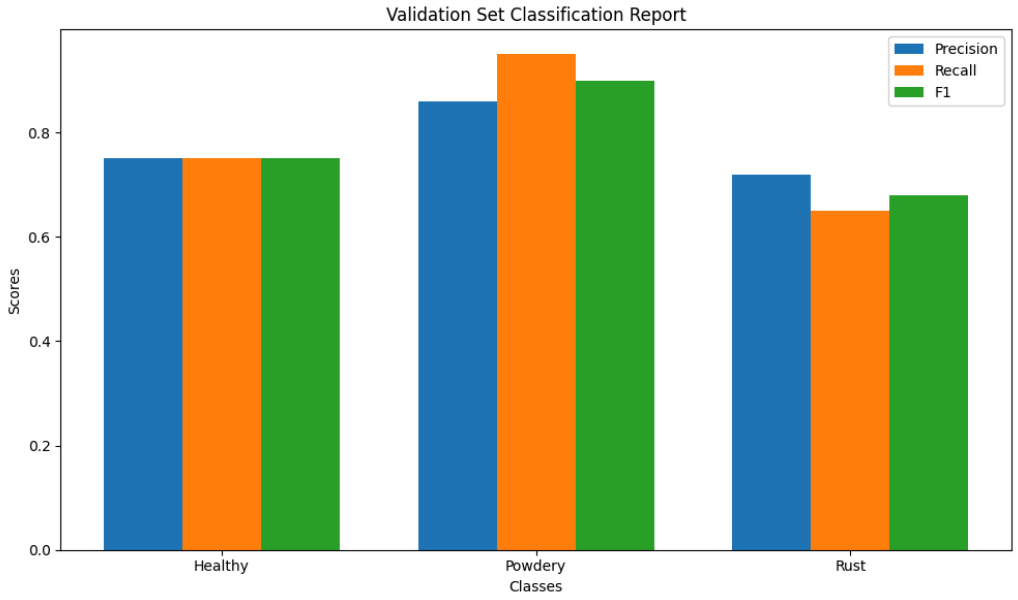


Figure 9: Classification Performance Metrics on Validation Set for SVM Model

Random Forest

The Random Forest model was fine-tuned using Grid Search to identify the optimal combination of hyperparameters. A 5-fold cross-validation strategy was implemented during this process, and the weighted F1-score was used as the evaluation criterion to handle class imbalances effectively. Key hyperparameters tuned during this process included the number of estimators, maximum depth of each tree, minimum samples required for a split, and leaf nodes. The parameter grid was designed to explore a diverse set of configurations, allowing the model to adapt effectively to the dataset's unique characteristics. The parameters were adapted from sample image classification tasks and discussions available on platforms such as Kaggle and Stack Overflow like in SVM.

Parameter Grid Used:

- n_estimators: 100, 200, 300
- max_depth: 10, 20, None
- min_samples_split: 2, 5, 10
- min_samples_leaf: 1, 2, 4
- max_features: sqrt, log2, None
- criterion: gini, entropy

The parameter grid enabled the systematic exploration of Random Forest configurations. The number of estimators determined the size of the ensemble, with larger ensembles generally providing more robust predictions at the cost of additional computational resources. The maximum depth of the trees controlled their complexity, preventing overfitting on the training data. The minimum samples required for splitting and at leaf nodes provided regularization, ensuring the trees were not overly specialized. The max_features parameter influenced the feature subsets considered for each split, with "sqrt" and "log2" providing controlled randomness to enhance generalization. The criterion parameter defined the

splitting metric, with "gini" focusing on minimizing impurity and "entropy" focusing on information gain.

The best hyperparameters were found for the Random Forest model: criterion was set to 'gini', meaning the model splits nodes based on Gini impurity. The max_depth parameter was left as None to let the trees grow until all leaves are pure or contain less than the minimum samples required for a split, ensuring that the model captures all possible patterns in the data. sqrt' was set for the max_features parameter, which at each split considers a random subset of features equal to the square root of the number of features. This ensures diversity in trees and hence increases generalization. The min_samples_leaf was set to 1, which allows the leaf node to contain just one sample, hence the model was allowed to learn even the very specific patterns. min_samples_split=2 allowed growth of trees when any node had at least two samples to consider splitting it further for higher flexibility during the tree growth. Finally, n_estimators=200 resulted in a considerable balance between the robustness of the prediction accuracy and computational efficiency. This combination of hyperparameters allows the Random Forest model to generalize well on this data, with a computationally feasible solution.

On the validation dataset, the RF model achieved an accuracy of 96.67% and a weighted F1-score of 96.66%. Class-wise performance metrics indicate near-perfect precision and recall for the Rust class, with an F1-score of 1.00, demonstrating the model's ability to consistently identify Rust with no misclassification. The Healthy and Powdery classes also displayed strong performance, each achieving an F1-score of 0.95. These results highlight the model's ability to distinguish between the three classes with minimal error on the validation data.

Table 2: RF - Validation Results

Class	Precision	Recall	F1-Score	Support
Healthy	0.91	1.0	0.95	20
Powdery	1.0	0.9	0.95	20
Rusty	1.0	1.0	1.0	20
Macro Average	0.97	0.97	0.97	60
Weighted Average	0.97	0.97	0.97	60

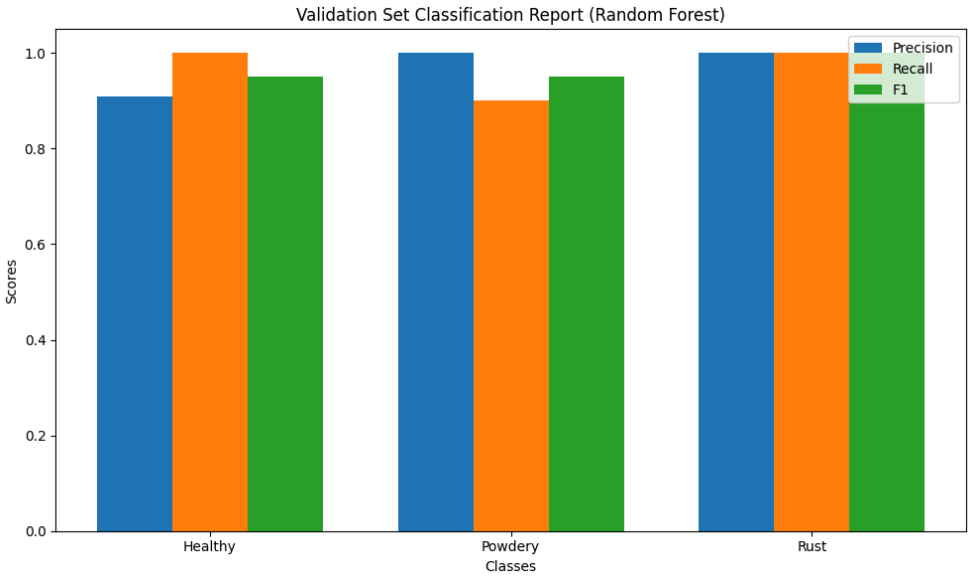


Figure 10: Classification Performance Metrics on Validation Set for RF Model

Adaboost

The training-validation-test split remained the same as provided to us.

We conducted a different preprocessing pipeline for this method that resized each plant image to 128 x 128 pixels and normalized its pixel values to the [0,1] range. From each image, we extracted features by combining color histograms (using 8×8×8 bins for the RGB channels) with edge maps derived from the Canny algorithm (via OpenCV). This approach produced a tabular feature vector capturing color distribution and shape information. This feature extraction process is crucial for the model to classify accurately.

Figure 11: Edge Map Of An Image

In tuning AdaBoost, we were interested in n_estimators (the number of boosting rounds), learning_rate (scaling the contribution of each tree), and max_depth (the capacity of each decision tree). Initially, we explored these parameters via manual sweeps, systematically varying one parameter at a time while holding the others fixed. Our base configuration was max_depth=1, learning_rate = 1, n_estimators = 100.

Accuracy was a straightforward metric since our dataset was balanced among the three classes. However, accuracy alone does not reveal everything. We tracked both training accuracy—an indicator of how well the model fits its training data—and validation accuracy—our signal for selecting the best hyperparameters. Whenever the gap between training and validation accuracy became large, we viewed it as a potential sign of overfitting, prompting us to adjust parameters accordingly (for example, by not pushing n_estimators too high or depth too deep). Thus, we noted the resulting validation accuracy to identify trends such as improvements when increasing n_estimators or adjusting the learning_rate.

As a side note, a base learner of max_depth=1 is common in AdaBoost, but there is potential for improvement by allowing slightly deeper trees (e.g., max_depth=2 or 3) if it does not induce overfitting. By carefully monitoring training versus validation accuracy, we found that increasing the depth from 1 to 2 often enhanced accuracy without significantly harming the model's generalization. This outcome was confirmed by examining how accuracy changed with each possible depth value. (See figures below)

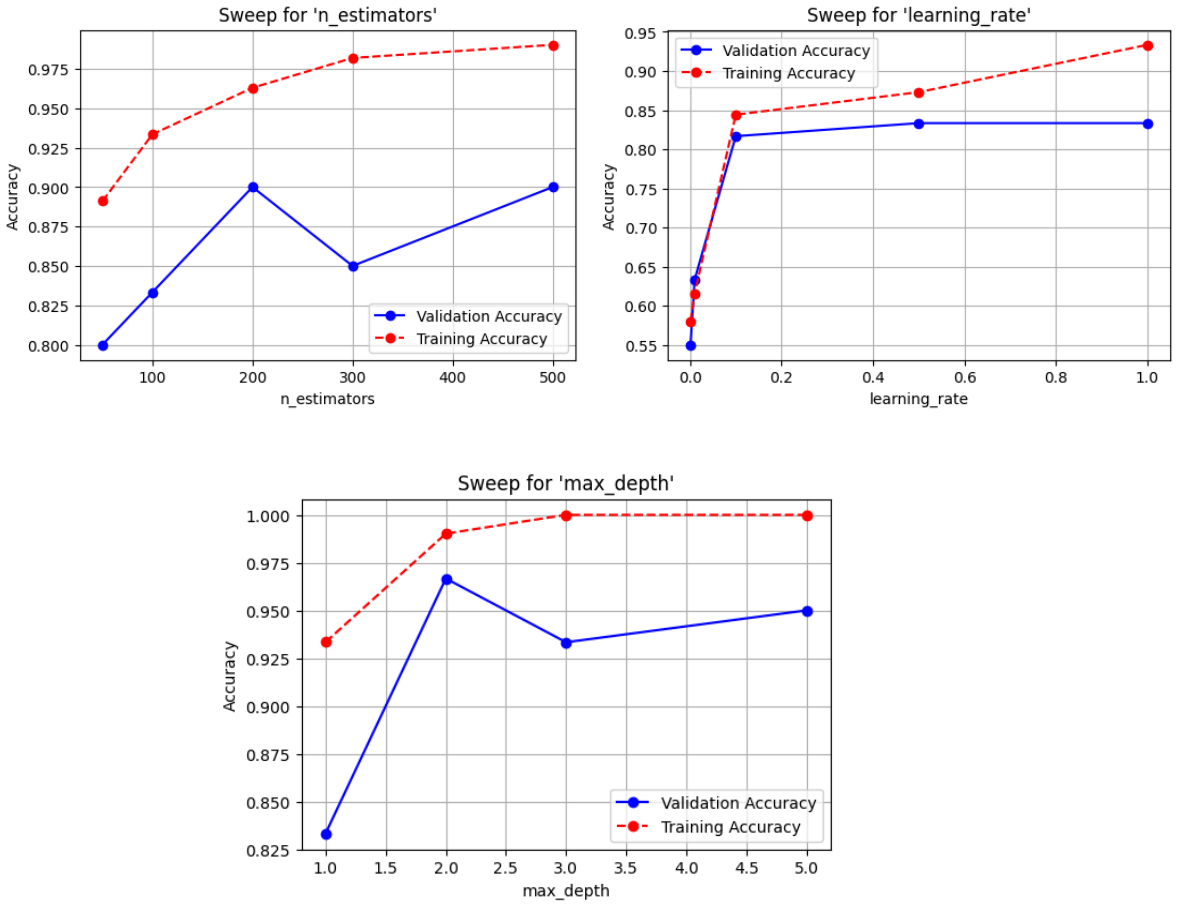


Figure 12: Accuracy of the model throughout sweeps

The figures above show the overfitting and the underfitting cases for each parameter. As mentioned, 2 was the sweet spot for max_depth, and any increase resulted in overfitting, while max_depth=1 was underfitting. For learning_rate, 0.001 and 0.01 showed severe underfitting, while learning rates above 0.1 did not cause a noticeable change. Lastly, n_estimators showed underfitting until 200 and overfitting after it. As a result of this tuning process, our final classifier was the one with the base configuration with the max_depth set to two.

CNN:

Some parameters of the model are fixed to observe the effect of batch size and learning rate individually. In the network stochastic gradient descent is used with momentum-based update rule. For the loss function, cross entropy loss is used to measure the performance of the classification network. Additionally, L2 regularization is used to prevent the model from overfitting to the training data. The model has been run for 25 epochs. The created algorithm stores and selects the best parameters that give the least loss on validation data and tests are conducted with optimal parameters. Fixed parameters of the network and optimal batch size, learning rate values can be seen in the hyperparameter tuning section. Finding the optimal parameters that result in highest accuracy on validation data is the purpose of this section.

Table 3: Fixed Coefficients of the CNN

Momentum Coefficient	0.9
Epoch Number	25
L2 Regularization Coefficient	0.0035

CNN with 2 Layer

Table 4: Hyperparameter Tuning for Batch Size 2 Layer

Batch Size	Learning Rate	Validation Loss	Validation Accuracy
16	0.3125	0.4410	89.33%
32	0.3125	0.6742	85.33%
64	0.3125	0.9817	80.00%
128	0.3125	0.7042	76.67%

Table 5: Hyperparameter Tuning for Learning Rate 2 Layer

Learning Rate	Batch Size	Validation Loss	Validation Accuracy
0.02250	16	0.3953	89.17%
0.02625	16	0.4432	84.00%
0.03125	16	0.3694	89.51%
0.03500	16	0.7785	77.57%

Hyperparameter tuning is completed with different batch size and learning rates and optimal value for batch size is 16 and for the learning rate 0.03125. Highest accuracy is obtained as 89.51% on validation data with these parameters. Adding another convolutional layer may increase the performance of the network by increasing complexity. On the other hand, increasing complexity may cause overfitting to the training data by memorizing all the aspects of the training data. In the next section, performance of the 3 layer CNN will be introduced.

CNN with 3 Layer

Table 6: Hyperparameter Tuning for Batch Size 3 Layer

Batch Size	Learning Rate	Validation Loss	Validation Accuracy
16	0.3125	0.2946	91.33%
32	0.3125	0.2211	93.33%
64	0.3125	0.5036	88.00%
128	0.3125	1.2837	69.67%

Table 7: Hyperparameter Tuning for Leaning Rate 3 Layer

Learning Rate	Batch Size	Validation Loss	Validation Accuracy
0.02250	32	0.4439	92.67%
0.02625	32	0.4473	83.33%
0.03125	32	0.2531	95.00%
0.03500	32	0.3849	86.33%

According to tables provided above, optimal batch size and learning rate values are 32, 0.03125 respectively. In these hyperparameter tuning tables, there are two experiments with the same batch size and learning rate which are 32, 0.03125 respectively.This learning rate achieves convergence without overshooting, and this batch size yields the best compromise between stability and generalization. Greatest accuracy on validation data is achieved as 95.00%. The reason that the validation accuracy and loss are different is, when the train function is called, the parameters of the convolution layers and linear layers are initialized randomly. This is the cause of the minor difference between validation loss and validation accuracy in those cases. Furthermore, in some experiments the validation loss and validation accuracy does not change as they should be. According to the learning rate table, validation loss decreases when learning rate changes from 0.02250 to 0.03500, even if validation accuracy is also less. The reason is, cross entropy loss is used which belongs to the probabilities of predictions. To clarify it, there is a difference in terms of loss between predicting 0.5 for a class and 0.99 for a class even if they classify the correct class.
Best hyperparameters and layer numbers for the CNN are shown below:

Table 8: Optimal Parameters for CNN

Learning Rate	Batch Size	Layer Number	Validation Accuracy
0.03125	32	3	95.00%

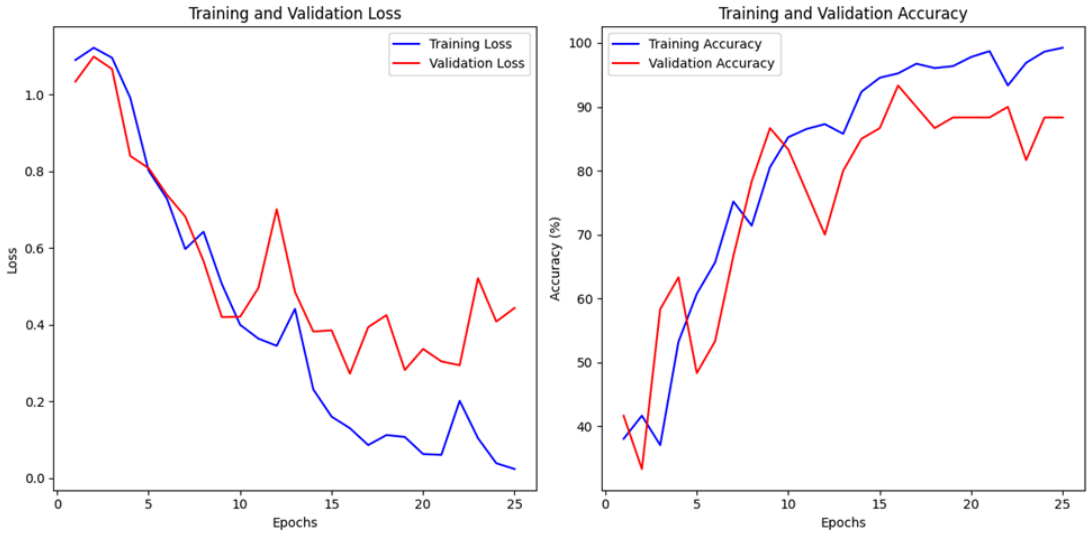


Figure 13: Loss and Accuracy Plots for Optimal Parameters Selected for CNN

Loss and accuracy result with optimal hyperparameters of the network, over the epochs can be seen from the plots above. Training loss and accuracy are changed in a more stable way than the validation loss and accuracy. As there is a limited size of validation data, this is the main cause of the spikes in the validation plots. Single classification error may lead to significant change in loss and accuracy as data samples are low in number. After the 16th epoch, the training accuracy of the model becomes close to the 100% which means

overfitting to the training data may occur if the network is trained more. Additionally, training loss continues to decrease but contrary to that, validation loss fluctuates too much after this epoch. This is a sign of overfit at this point. Even this architecture is more complex (which increases the risk of overfitting) than CNN with 2 layers, at 16'th epoch CNN with 3 layers achieved the highest validation accuracy. However, the implemented algorithm selects the parameters of the network that gives the highest validation accuracy, therefore, overfitting does not affect optimal parameters selected.

Transfer Learning (Resnet 18):

Number of Epochs runned: 20
Optimizer: Adam Optimizer
Optimal Batch size: 32
Optimal Learning rate: 0.0005

	Training Loss	Validation Loss	Validation Accuracy (%)
Batch Size 16	0.2	0.265	90.33
Batch Size 32	0.176	0.255	91.67
Batch Size 64	0.16	0.273	89.0

Figure 14: Batch sizes compared (Learning rate fixed at 0.0005)

Learning Rate	Epoch	Validation Loss	Validation Accuracy (%)
0.0001	20.0	0.32	85.0
0.0005	20.0	0.255	91.67
0.001	20.0	0.345	88.33
0.005	20.0	0.45	83.0

Figure 15: Learning rates compared

In order to achieve the best performance, the hyperparameter tuning for the transfer learning model included batch size and learning rate. Batch sizes of 16, 32, and 64 were compared, with the learning rate fixed at 0.0005. Figure 2: Results of batch size tuning. The results showed that the lowest validation loss of 0.255, the highest validation accuracy of 91.67%, and best test accuracy of 92.0% were achieved for the batch size of 32. Then, keeping the batch size fixed at 32, the learning rate tuning was done by testing the following learning rates: 0.0001, 0.0005, 0.001, and 0.005. Figure 3: It can be realized that a learning rate of 0.0005 had the best performance with a validation loss of 0.255, a validation accuracy of 91.67%, and a test accuracy of 92.0%. These results show that careful hyperparameter tuning is really important for model performance optimization and can lead to robust results.

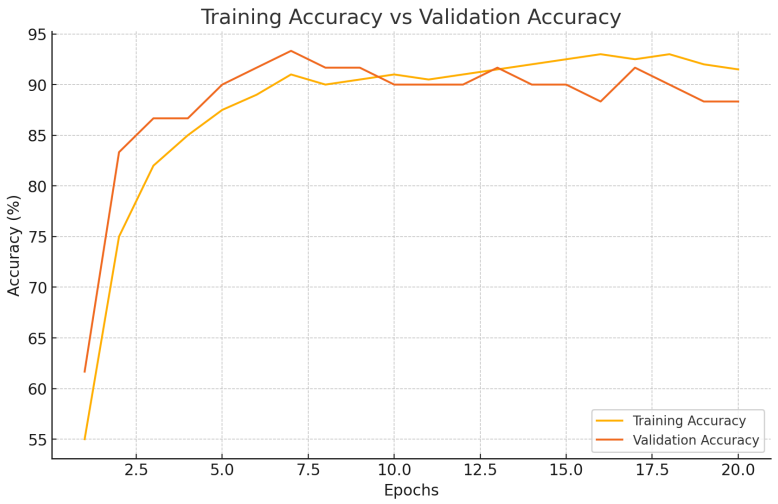


Figure 16: Training accuracy vs Validation Accuracy

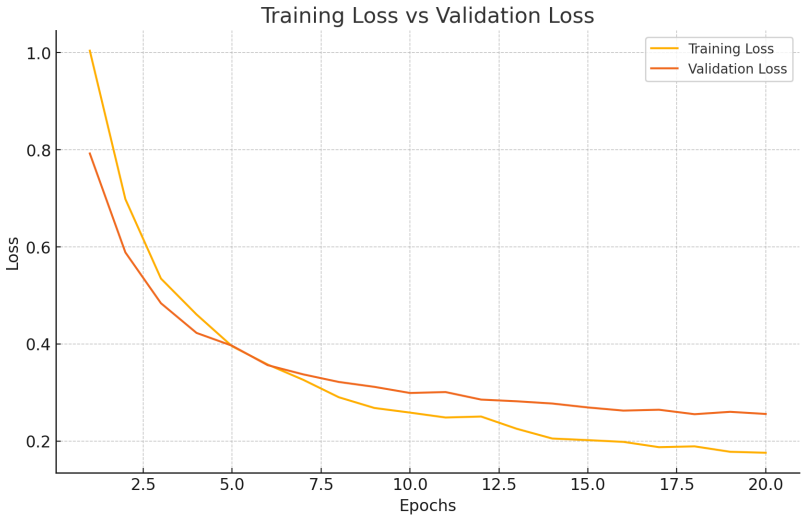


Figure 17: Training accuracy vs Validation Accuracy

These graphs show the performance of training and validation of the model in 20 epochs.

Figure 16: Training vs. validation accuracy. The accuracy of both training and validation increases smoothly for the initial epochs and then stabilizes around 90–95%. This implies that the model is learning well and not overfitting seriously, as the validation accuracy follows the training accuracy quite well.

Figure 17: illustrates the training and validation loss, both of which are continuously going down. This reflects effective optimization whereby the model generalizes well to unseen data without overfitting. That will reflect a well-trained model with good convergence characteristics.

Transfer Learning (VGG-16):

For transfer learning, a pre-trained VGG16 model is employed, in which the final fully connected layer is changed to match the three target classes while the convolutional layers are frozen to preserve previously learned features. Only the last layer is optimized during the training procedure, which uses a cross-entropy loss function and the Adam optimizer with

different learning rate and batch size values across 25 epochs. For the assessment, the model with the highest validation accuracy is considered to be optimal.

Table 9: Hyperparameter Tuning for Batch Size of VGG-16

Batch Size	Learning Rate	Validation Loss	Validation Accuracy
16	0.00075	0.2231	94.09%
32	0.00075	0.2131	94.93%
64	0.00075	0.2248	93.94%
128	0.00075	0.2321	92.73%

Table 10: Hyperparameter Tuning for Learning Rate of VGG-16

Learning Rate	Batch Size	Validation Loss	Validation Accuracy
0.0005	32	0.2331	94.09%
0.00075	32	0.2304	94.19%
0.001	32	0.2392	93.11%
0.00125	32	0.2548	91.75%

The hyperparameter tunings for the VGG-16 model helped us to select the optimal parameters. When the batch size equals to 32 and learning rate is equivalent to 0.00075, the model achieves its highest accuracy on validation data. Best balance of stability and generalization is obtained by this batch size and convergence without overshooting is reached with this learning rate. Loss function is selected as cross entropy loss and adam optimizer is used as the optimization algorithm. A total of 25 epochs, the model has been run. Model obtained its highest validation accuracy with 94.93% and 0.2131 loss. Nevertheless, there is not a huge difference between the validation accuracies of different hyperparameter selections so that model is robust to small changes in hyperparameters.

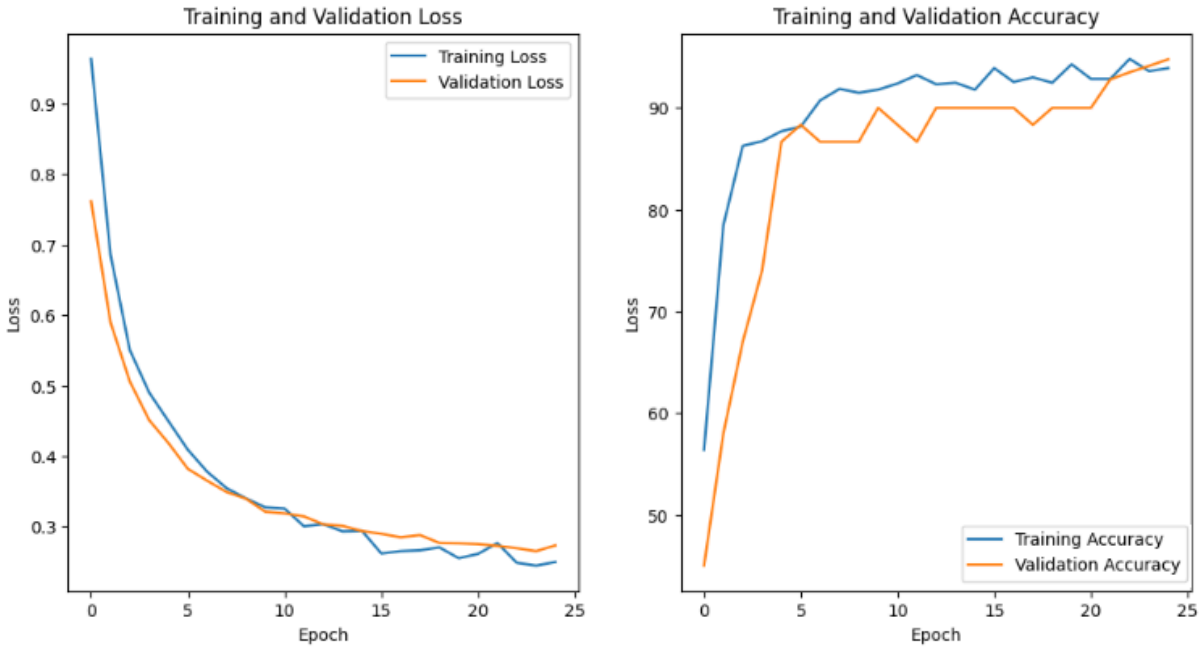


Figure 18: Train and Validation Plots for VGG-16 Model

Training and validation losses are decreasing in a stable manner and they begin to become flat around epoch 20 - 25. For the first 5 epochs, the model learns incredibly fast and achieves 88.17% accuracy on validation. From the plots it can be understood that the VGG-16 model is able to learn quickly and converges smoothly. Differences between training and validation plots are not much so that model does not overfit to the training data. The best parameters are saved and selected by highest validation accuracy. At 25th epoch, the model achieved its highest validation accuracy by 94.93% accuracy. With these best parameters, the model will be tested on test data and confusion matrices and performance metrics will be reported.

Test Results of Models

Results of SVM Model

The best-performing configuration was determined through a systematic Grid Search process, identifying the optimal hyperparameters as C=10, gamma='scale', and kernel='rbf'. These settings represent a regularization parameter of 10, a radial basis function (RBF) kernel for handling non-linear separability, and a scaled gamma value that adapts the influence of individual training examples based on feature space dimensions.

The test dataset results indicate an accuracy of 76.00% and a weighted F1-score of 75.54%, suggesting the model generalizes reasonably well to unseen data. The Healthy class was identified with 90% recall, demonstrating strong sensitivity in detecting healthy plants. However, similar to the validation results, the Rust class exhibited lower recall at 60%, reflecting persistent challenges in correctly identifying this category. The macro-averaged F1-score of 0.76 highlights comparable performance across classes, while the weighted F1-score underscores the model's ability to account for class distributions effectively.

Table 11: SVM - Test Results

Class	Precision	Recall	F1-Score	Support
Healthy	0.71	0.9	0.8	50
Powdery	0.78	0.78	0.78	50
Rusty	0.81	0.6	0.69	50
Macro Average	0.77	0.76	0.76	150
Weighted Average	0.77	0.76	0.76	150

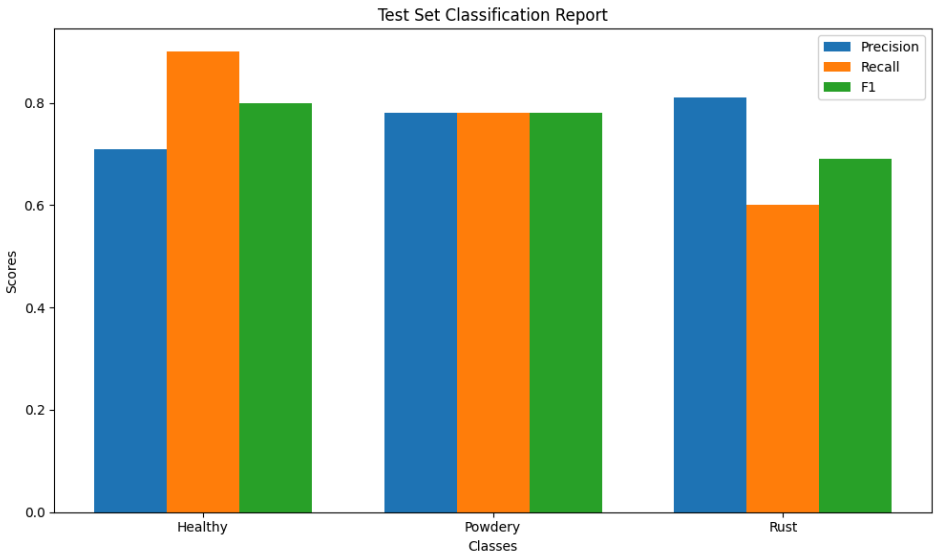


Figure 19: Classification Performance Metrics on Test Set for SVM Model

In summary, the SVM model displayed robust performance for the Powdery and Healthy categories while showing limitations in identifying Rust.

Results of RF Model

Through hyperparameter tuning, the optimal configuration was identified as criterion='gini', max_depth=None, max_features='sqrt', min_samples_leaf=1, min_samples_split=2, and n_estimators=200. This configuration uses the Gini impurity criterion for node splits, no maximum tree depth to allow full growth, square root of the features at each split for diversity, and 200 trees in the ensemble to enhance stability.

The RF model demonstrated strong generalization when evaluated on the test dataset, achieving an accuracy of 92.00% and a weighted F1-score of 92.09%. The Rust class continued to exhibit good performance with an F1-score of 0.96, underscoring the model's capability to accurately detect this class even on unseen data. The Powdery class achieved an F1-score of 0.92, while the Healthy class showed slightly lower but still successful performance with an F1-score of 0.89. The macro-averaged F1-score of 0.92 indicates consistent performance across classes, while the weighted F1-score reflects the model's sensitivity to class distributions in the test dataset.

Table 12: RF - Test Results

Class	Precision	Recall	F1-Score	Support
Healthy	0.84	0.94	0.89	50
Powdery	0.96	0.88	0.92	50
Rusty	0.98	0.94	0.96	50
Macro Average	0.92	0.92	0.92	150
Weighted Average	0.92	0.92	0.92	150

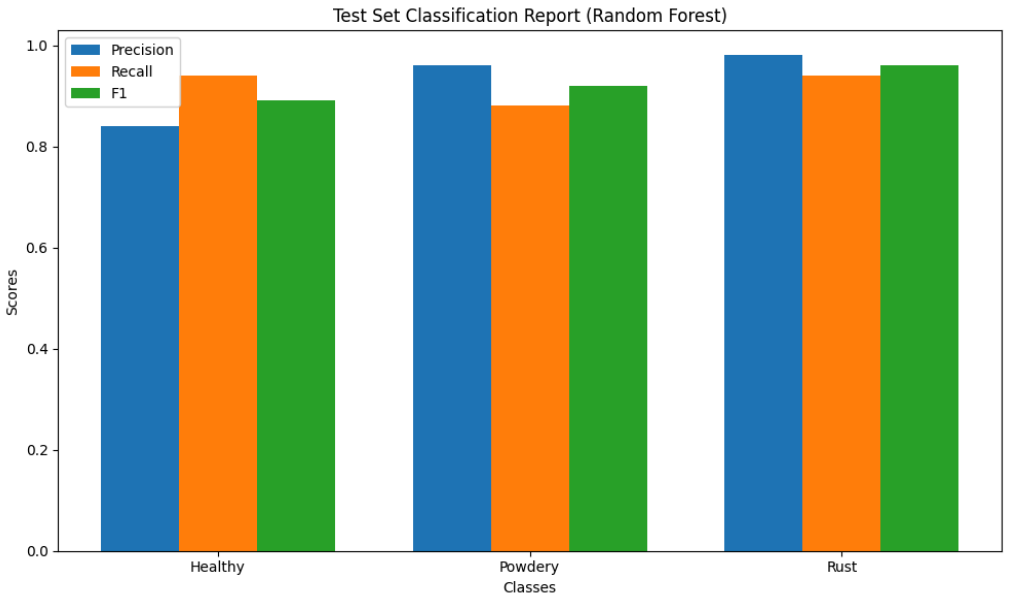


Figure 20: Classification Performance Metrics on Test Set for RF Model

In summary, the Random Forest model displayed exceptional classification capabilities, particularly excelling in the detection of the Rust class while maintaining high performance

for the other categories. Its ensemble-based design contributed to its robustness and ability to handle class imbalances effectively. The high accuracy and F1-scores across both validation and test datasets demonstrate that the RF model is well-suited for this classification task.

Results of CNN:

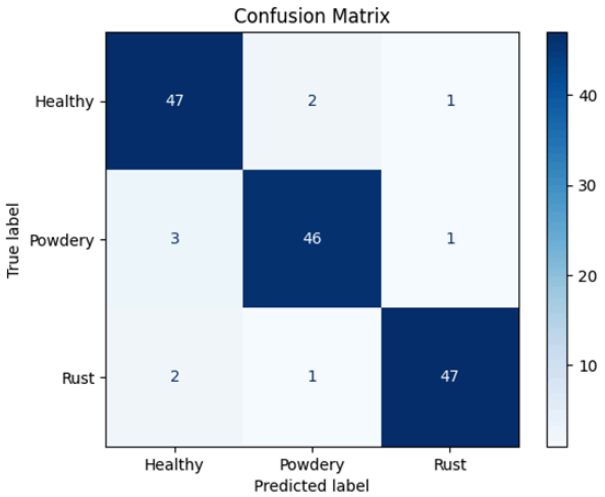


Figure 21: Confusion Matrix of CNN with Optimal Parameters

Above is the confusion matrix for the CNN test result with optimal parameters. 93.33% test accuracy is attained with optimal parameters. Test accuracy is 1.67% less than the validation accuracy which seems normal. The model is good at generalizing instead of overfitting. Predicting the powdery class as healthy results in the largest classification error. Apart from that, the model performed well on these predictions, and the classification errors are nearly uniform.

Table 13: Performance Metrics of CNN with Optimal Parameters

Class	Precision	Recall	F1-score	Support
Healthy	0.90	0.94	0.92	50
Powdery	0.94	0.92	0.93	50
Rusty	0.96	0.94	0.95	50
Macro Average	0.93	0.93	0.93	150
Weighted Average	0.93	0.93	0.93	150

There is not a significant performance gap in the predictions of classes. Highest precision and F1-score are in the Rusty class and it can be stated that the model distinguishes this class slightly better. Healthy class has relatively lower precision on it which corresponds to occasional false positives. This situation can also be observed from the confusion matrix that healthy samples are misclassified with Powdery and Rusty classes more than the contrary circumstances. The high macro and weighted averages confirm the model's robustness across all classes.

Results of Transfer Learning Models

Results for Resnet 18

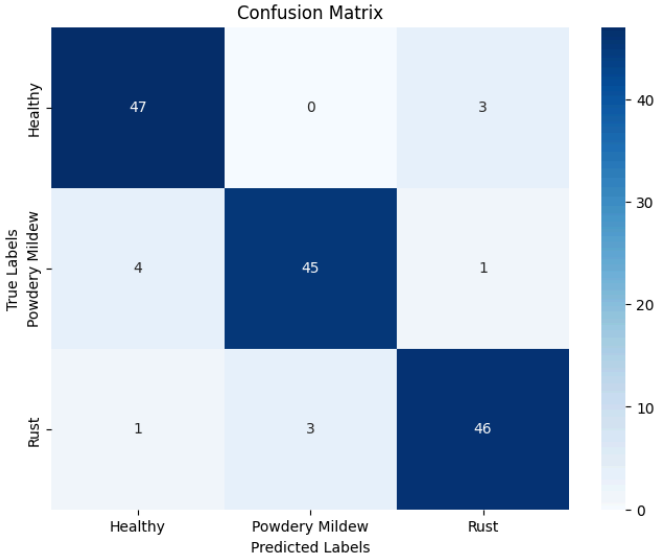


Figure 23: Confusion Matrix

	Precision	Recall	F1-Score	Support
Healthy	0.903846	0.94	0.921569	50.0
Powdery Mildew	0.9375	0.9	0.918367	50.0
Rust	0.92	0.92	0.92	50.0
accuracy	0.92	0.92	0.92	0.92
macro avg	0.920449	0.92	0.919979	150.0
weighted avg	0.920449	0.92	0.919979	150.0

Figure 24: Precision scores of transfer learning model

Test Loss: 0.2942 Test Accuracy: 92.00%

The results depict the strong performance of the model in classifying the three plant health conditions: Healthy, Powdery Mildew, and Rust. The confusion matrix is a good representation of how most of the samples were correctly classified by the model, while a few of them were misclassified. For instance, 47 out of 50 "Healthy" samples were correctly classified and 3 were misclassified. Results from these studies indicate that Powdery Mildew yields, out of 56 data inputs of the class, correctly predicted 45 samples against

misclassification of 4 while Rust correctly classified 46 with a total number misclassified as 4 on other classes.

The precision, recall, and F1-score of each class further give more assurance of the reliability of this model. The precision score in all classes is more than 90%, proving the low rate of the false positive. Recall values of 90% and above are indicative of the low false negative rate; hence, the model correctly predicts most instances for each class. Its F1-score is 0.92, while the general test accuracy is 92.00%, showing a balanced performance on all classes. Its test loss stands at 0.2942; thus, this model provides minimal classification errors but still strongly generalizes well to unseen data. These results validate the suitability of the model with regard to accuracy in plant view.

Results for VGG-16

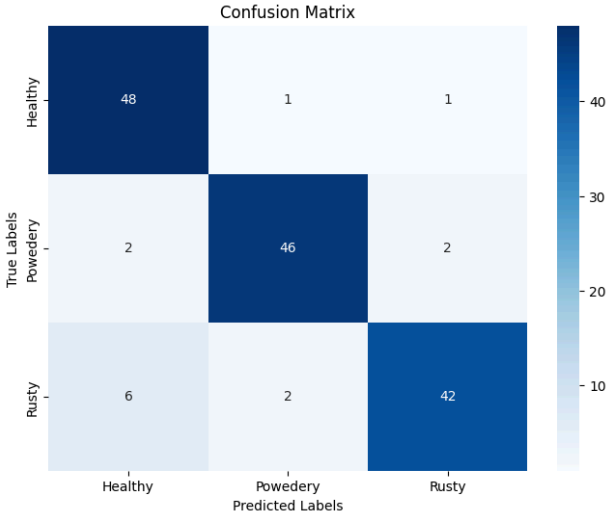


Figure 25: Confusion Matrix for Test Data VGG-16

Number of diagonal elements is high in this confusion matrix and this indicates a high accuracy on test data. Test accuracy is attained by 90.67%. The most significant misclassification occurs when the model predicts Healthy class instead of Rusty by 6 times. Rusty and Healthy classes may contain similar and overlapping features inside their images. This seems to be the reason that model struggles to differentiate these two classes.

Table 14: Performance Metrics of CNN with 3 Layers

Class	Precision	Recall	F1-Score	Support
Healthy	0.86	0.96	0.91	50
Powdery	0.94	0.92	0.93	50
Rusty	0.93	0.84	0.88	50
Macro Average	0.91	0.91	0.91	150
Weighted Average	0.91	0.91	0.91	150

Even model shows a successful overall performance with 90.67% accuracy on test data, Powdery becomes the easiest class to be differentiated and Rusty becomes the most challenging class to be distinguished. Precision of Healthy is the lowest that means predictions on this class are relatively less correct. The most important misclassification is mentioned in the confusion matrix part and this is the main reason for the 0.86 precision score on Healthy class. On the other hand, in the Healthy class the recall is maximum, which means the model captures almost all true Healthy samples with very few false negatives. Recall of Rusty class is 0.84 and it is the lowest recall score among classes. The same misclassification situation mentioned in the confusion matrix resulted in this 0.84 recall. With macro and weighted averages being 0.91, it can be seen that the model does not behave biased towards a class.

Results for AdaBoost:

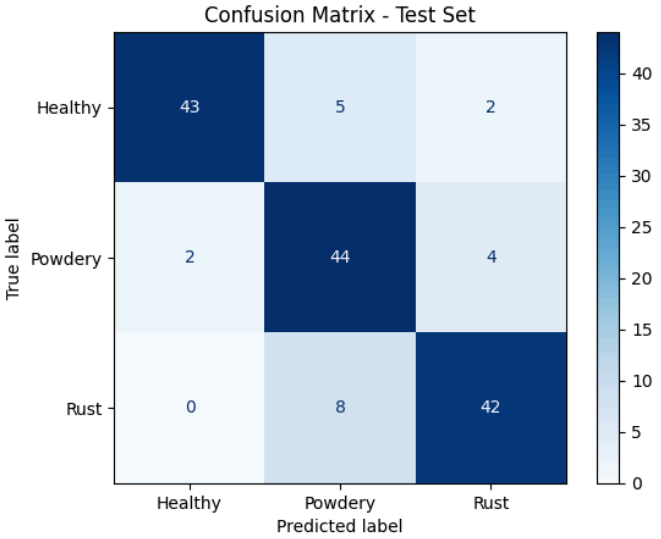


Figure 26: Confusion Matrix Of The Model Predictions On The Test Set

Our best results were when using AdaBoost with base parameters set at n_estimators=100, learning_rate=1.0, and a base decision tree of max_depth=2, as mentioned in the hyperparameter tuning section. The difference between max_depth=1 and max_depth=2 proved decisive in capturing subtle distinctions among the three classes while avoiding overfitting.

Upon concluding the hyperparameter tuning, we evaluated the final model on the test set, which was held out during training and validation. The final model's accuracy on the test set was 86%, consistent with the validation performance, suggesting no severe overfitting had occurred. We also generated a confusion matrix, which revealed that the Rust class was more challenging to differentiate. Specifically, a small cluster of misclassifications occurred between the Powdery and Rusty classes, likely attributable to visual similarities in color or texture. Below, you can see three misclassified examples that were classified as Powdery but were Rusty. These are hard samples to predict even with human eyes, especially the second image, as its perspective is not from the top of the leaf.

Figure 27: Images Misclassified as Powdery but Were Rusty

We interpret these results to imply that while a small tree depth can be enough in many boosting contexts, a moderate increase (from depth 1 to depth 2) yielded a noticeable boost in overall accuracy. This reinforces the notion that a minimal but well-chosen expansion in model complexity can address challenging examples without suffering from an explosion in variance or overfitting.

Final Comparison and Discussion of Models

In the final comparison, each model is tested on test data with their optimal parameters on validation accuracy. Comparing the different machine learning models, including those involving deep learning methods, really underlines how good each model is when working out the classification of plant diseases. Moving on to the comparisons provided in the machine learning models section, it should be noted that Random Forest achieved 92% on testing versus a 76.00% obtained from the Support Vector Machine and 86.00% obtained from Adaboost. In the domain of convolutional neural networks, accuracies of 93.33% outperform transfer learning models like VGG-16 with ResNet-18, who are 90.67% and 92% respectively. Although ResNet18 and VGG-16 extend their pre-trained networks to generalize well into the dataset, the potency of a custom architecture performed very well in this setting.

Table 15: Comparison of Models on Test Accuracy

Model	Test Accuracy
Support Vector Machine	76.00%
Random Forest	92.00%
Adaboost	86.00%
Convolutional Neural Network	93.33%
Transfer Learning (Resnet 18)	92.00%
Transfer Learning (VGG-16)	90.67%

These models have very different computational efficiencies due to their differences in both complexity and training requirements. Machine learning models that are usually efficient computationally, especially on smaller data, including Support Vector Machines and Random Forest. While generally fast, Support Vector Machines are much slower on very large datasets or high-dimensional data, since solving a quadratic optimization problem is required; it does enjoy relatively fast inference, though. Random Forest, because of its ensemble structure, is computationally more intensive in training due to the creation of several decision trees, but the inference remains fast. Adaboost has medium efficiency. It trains weak learners sequentially, which may increase the time of training but keeps the inference relatively fast.

Deep learning models, such as CNN and transfer learning approaches, including ResNet-18 and VGG-16, are computationally intensive. They contain a large number of parameters with a requirement for matrix operations. Among them, ResNet-18 is usually more efficient than VGG-16 since the residual connections help to avoid the vanishing gradient problem and make the learning easier. In particular, VGG-16 has a much deeper architecture and lots of more parameters; thus, much more computational resources are needed for both training and inference.

Overally, some traditional models such as random forests will fit best at very minimal computation, and deep learning, at the total cost of much higher computations, achieves far superior and high-quality results for highly complex information datasets. For this specific

classification task Random Forest, CNN and Transfer Learning (Resnet 18) are the most accurate models. The optimal model may be chosen according to computational resources. If computational efficiency does not matter, CNN or Transfer Learning are successful deep learning models but if computational load is important, Random Forest would be a better option.

References:

[1] Towards Data Science, "Convolutional Neural Networks Explained," *Towards Data Science*, Dec. 21, 2021. [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939. [Accessed: Dec. 25, 2024].

[2] “Resnet18,” resnet18 - Torchvision main documentation, https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html (accessed Dec. 28, 2024).

[3] Original resnet-18 architecture | download scientific diagram, https://www.researchgate.net/figure/Original-ResNet-18-Architecture_fig1_336642248 (accessed Dec. 28, 2024).

[4] Great Learning Team, "Everything You Need to Know About VGG16," *Medium*, May 7, 2021. [Online]. Available: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918. [Accessed: Dec. 27, 2024].

[X] Hastie, Trevor; Rosset, Saharon; Zhu, Ji; Zou, Hui (2009). "Multi-class AdaBoost". *Statistics and Its Interface*. **2** (3): 349–360. doi:10.4310/sii.2009.v2.n3.a8. ISSN 1938-7989.