# Plant Disease Classifier
# Group 23

Asım Adil Can - 22103729
Erdem Pülat - 22103566
Gün Taştan - 22101850
Kerem Er - 22102369
Begüm Kunaç - 22103838

# Project Overview

**Significance:**

Early detection and accurate classification of plant diseases are essential for timely intervention and effective disease management. By developing a robust classification system that identifies plant health conditions, our project aims to:

- Improve plant health monitoring.
- Enable early intervention to minimize crop loss.
- Enhance agricultural efficiency through automation

# Objective:

To develop a machine learning-based classification system capable of accurately identifying three distinct plant conditions:

- **Healthy Plants:** No visible disease symptoms.
- **Powdery Mildew:** Characterized by white fungal growth on leaves.
- **Rust Disease:** Identified by yellow, rust-like discoloration on leaves.

Our goal is to compare multiple machine learning and deep learning approaches to determine the most effective method for this classification task.

# Dataset

**Class Distribution:**

- **Healthy:** Plants showing no visible disease symptoms.
- **Powdery:** Plants affected by white fungal growth.
- **Rusty:** Plants displaying yellow to orange discoloration on its surface.

**Dataset Structure:**

- **Training Set:** 1322 images (458 healthy, 430 rusty, 434 powdery).
- **Testing Set:** 150 images (50 healthy, 50 rusty, 50 powdery).
- **Validation Set:** 60 images (20 healthy, 20 rusty, 20 powdery).

# Dataset Properties

**Image Characteristics:**

- **Content:** Healthy tissue, white fungal growth (powdery mildew), and yellow discoloration (rust).

- **Image Sizes:** Image sizes are not fixed, resolution of images are in good quality.

**Dataset Link:** Plant Disease Classification Dataset

# Sample Images from Dataset

Healthy Class

Powdery Class

Rusty Class

# Dataset Preprocessing

**Preprocessing Pipeline:**

1. **Image Standardization:**
   - Resized all images to **128x128 pixels** for consistent input dimensions and reduced computational overhead.
   - Used PyTorch's **torchvision.transforms** for resizing.
2. **Normalization:**
   - Images normalized with mean and standard deviation of 0.5 for each RGB channel.
   - Transformed pixel values to the range **[-0.5, 0.5]** to achieve faster convergence and better numerical stability.

# What did we use?

**Libraries we used:**

- NumPy
- OpenCV
- Matplotlib
- Scikit-learn
- pandas
- Pytorch

**Additional tools:**

- Github
- Google Colab
- Kaggle

**Models we trained:**

- SVM
- Random Forest
- Adaboost Classifier
- Transfer Learning
- CNN

# Support Vector Machine

**Preprocessing Steps for SVM:**

**Feature Extraction:**

- **HOG Features:** Captured texture and shape patterns from grayscale images.
- **Color Histograms:** Represented color distribution in the RGB channels.

**Feature Concatenation:** Combined HOG and color histogram features for a richer representation.

**Normalization:** Features were scaled using StandardScaler for better model performance.

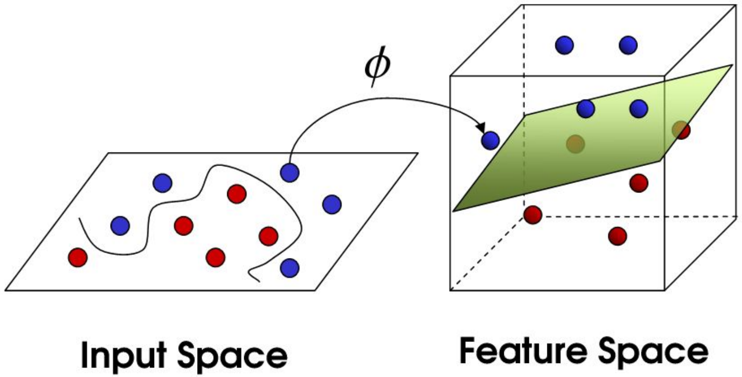
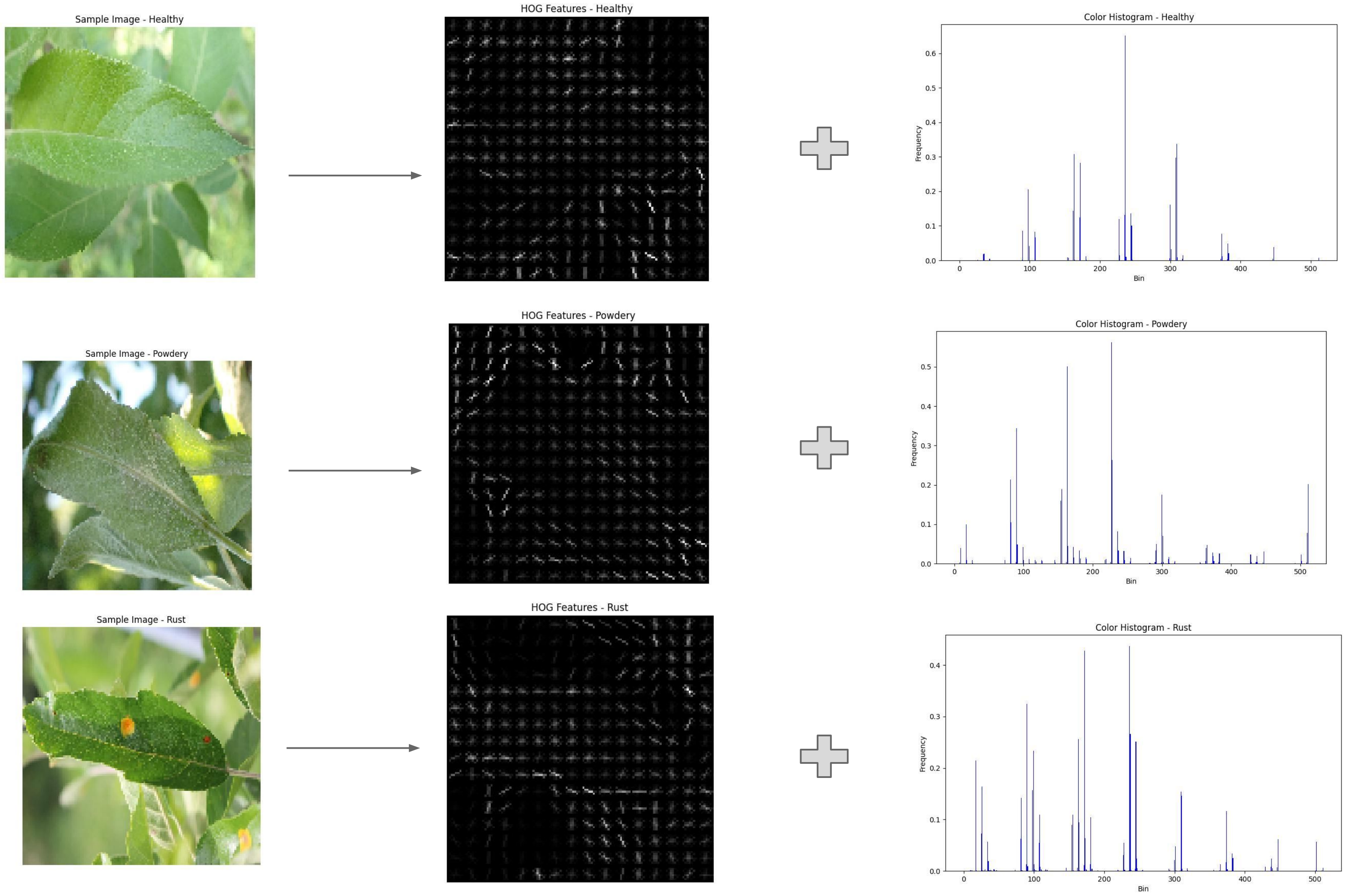



Figure 1: SVM illustration

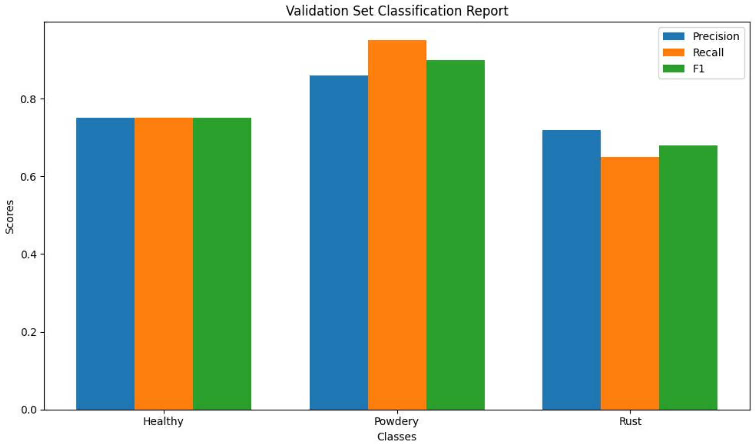

Sample Image - Healthy
HOG Features - Healthy
Color Histogram - Healthy
Sample Image - Powdery
HOG Features - Powdery
Color Histogram - Powdery
Sample Image - Rust
HOG Features - Rust
Color Histogram - Rust
Frequency
Bin

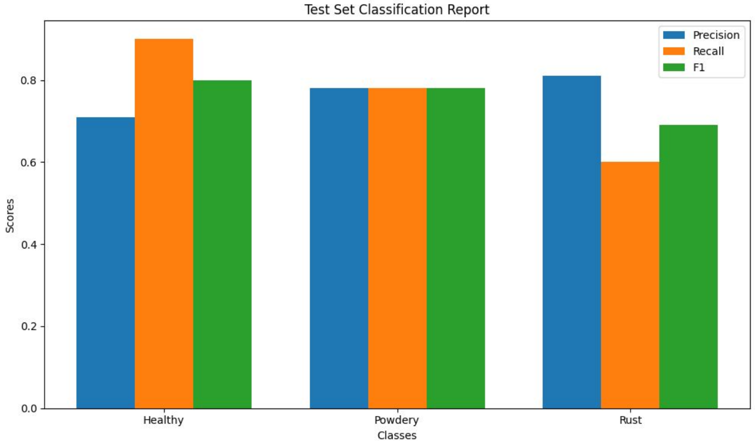
# SVM - Experimental Setup and Parameters

- **Grid Search:**
  - Performed hyperparameter optimization using 5-fold cross-validation.
- **Evaluation Criteria:**
  - Weighted F1-score.
- **Parameter Grid:**
  - **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

# SVM - Results

- **Best Parameters:**
  - **Kernel:** Radial Basis Function
  - **C (Regularization parameter)** 10
  - **Gamma:** scale
- **Validation Results:**
  - Accuracy: 0.7833
  - Weighted F1-Score:  0.7797
- **Test Results:**
  - Accuracy: 0.76
  - Weighted F1-Score: 0.7554



# Random Forest

**Preprocessing Steps for RF:**

1. **Feature Extraction:**
   - **HOG Features:** Captured texture and shape patterns from grayscale images.
   - **Color Histograms:** Represented color distribution in the RGB channels.
2. **Feature Concatenation:** Combined HOG and color histogram features for a richer representation.
3. **Normalization:** Features were scaled using StandardScaler for better model performance.

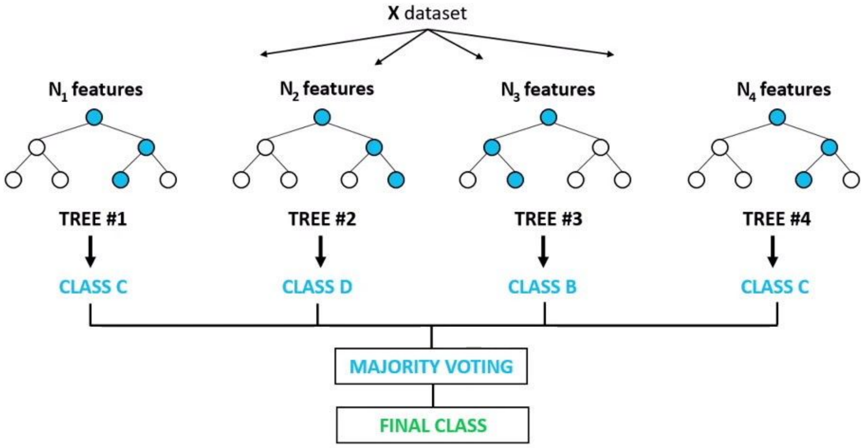


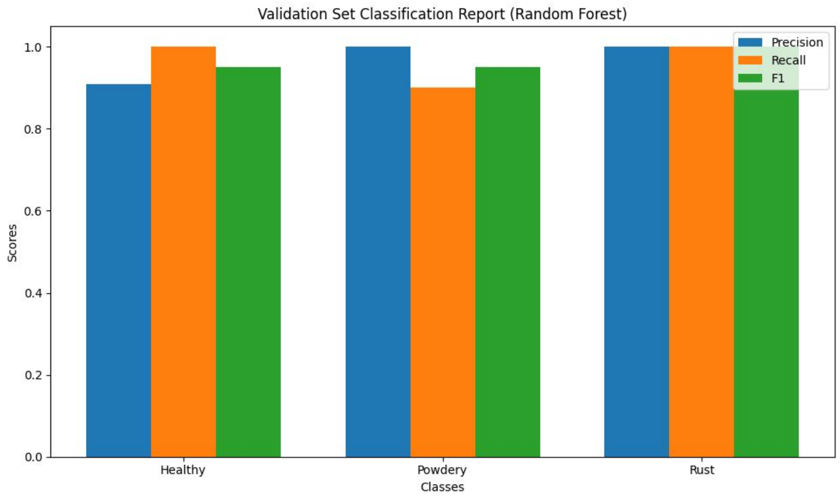

Figure 3: Random Forest Illustration

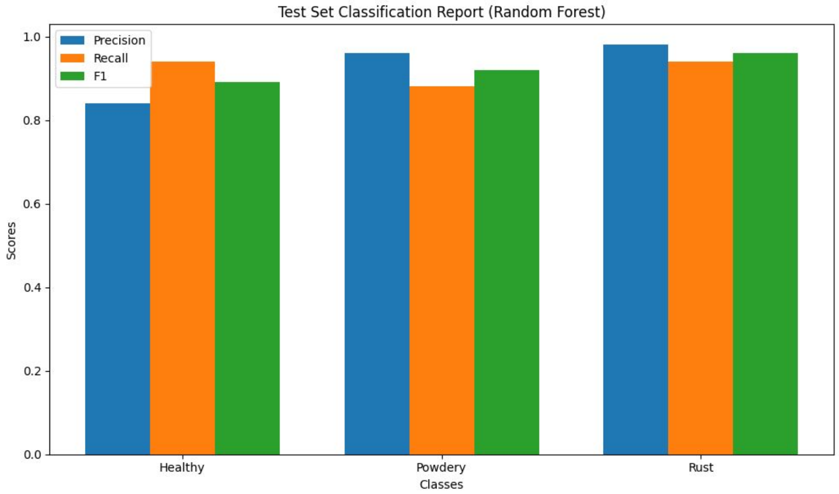
# Random Forest - Experimental Setup and Parameters

- **Grid Search:**
    - Performed hyperparameter optimization using 5-fold cross-validation.
- **Evaluation Criterion:**
    - Weighted F1-score.
- **Parameter Grid:**
    - 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

# RF Results

- **Best Parameters:**
  - criterion: gini
  - max_depth: None
  - max_features: sqrt
  - min_samples_leaf: 1
  - min_samples_split: 2
  - n_estimators: 200
- **Validation Results:**
  - Accuracy: 0.9667
  - Weighted F1-Score: 0.9666
- **Test Results:**
  - Accuracy: 0.9200
  - Weighted F1-Score: 0.9209



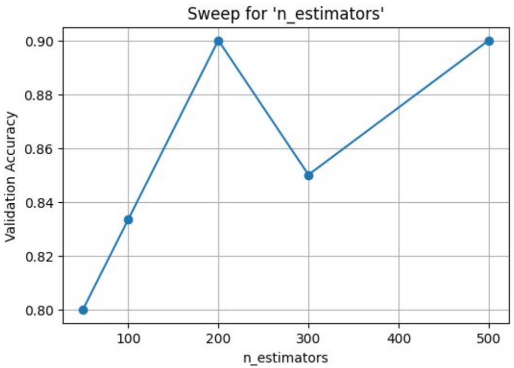



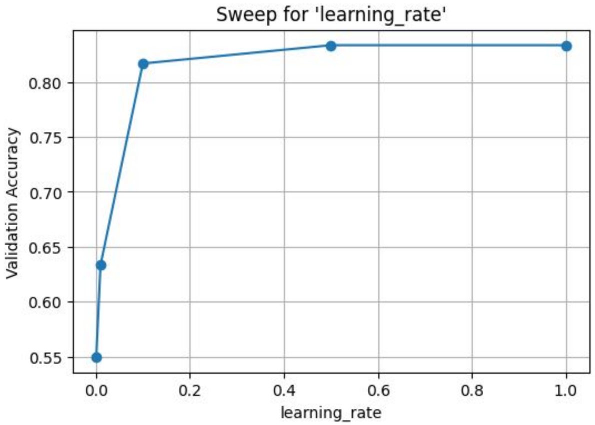
# Adaboost & The Core Mechanism

- AdaBoost stands for *Adaptive Boosting*.
- Initialize equal weights for all training examples.
- Train a weak learner; measure error on weighted samples.
- Increase weights for misclassified samples; decrease for correct ones.
- Repeat for multiple rounds and sum each learner's contributions.
- Sequentially focuses on misclassified examples to reduce errors over rounds.
- Our weak learner: Decision Trees, with depth as a hyperparameter.
- Feature Extraction: Same as in SVM, combination of grayscale edge-features and color histograms

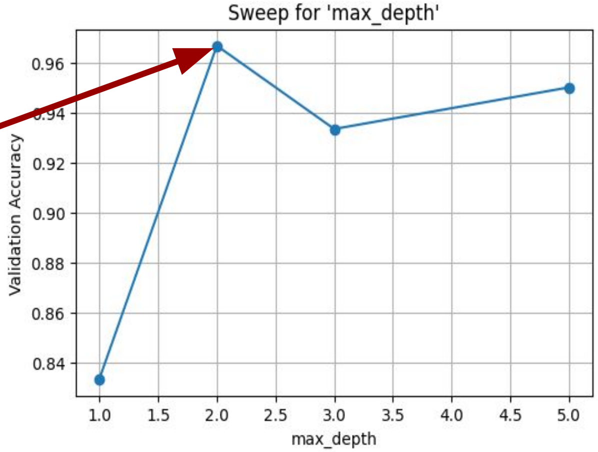
# Adaboost Hyperparameters & Tuning

**Base : # of estimators : 100, Learning Rate: 1.0, Max Depth: 1**



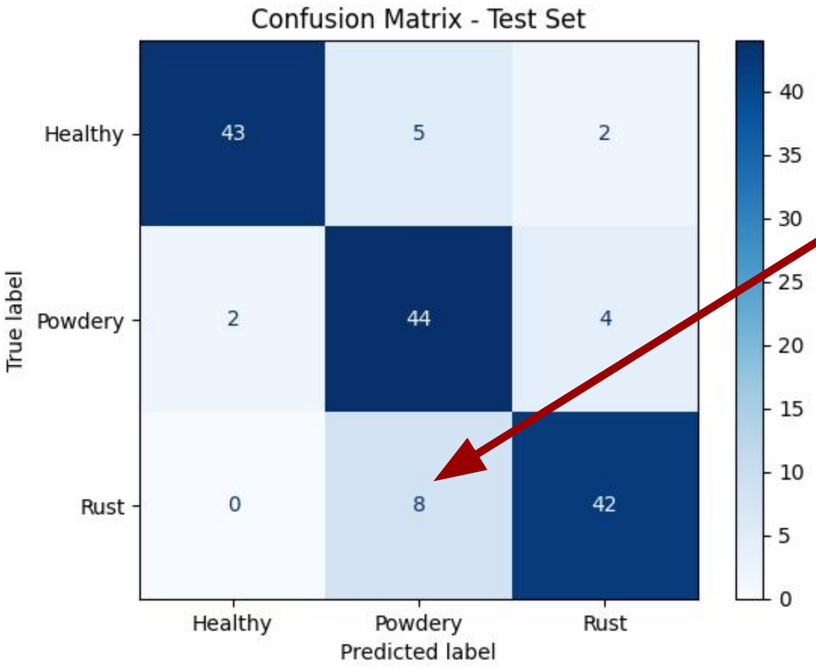




**The Winner!**



# Adaboost Results

**Test Accuracy : 86% (Tested with the winner of the hyperparameters)**




**Main Problem**

# Transfer learning Model

What is transfer learning?

Which Model We Used?

**ResNet-18**

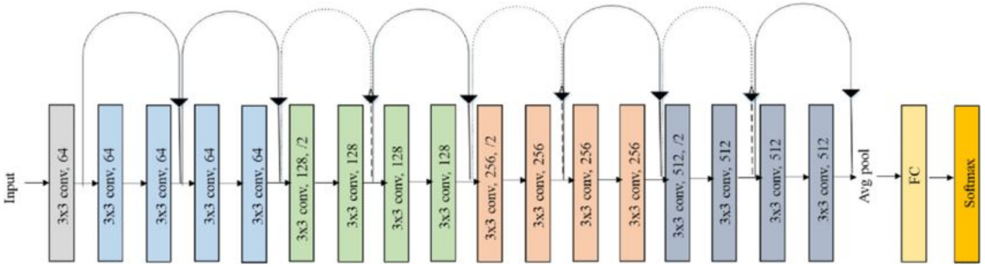



Figure 4: Resnet 18 structure

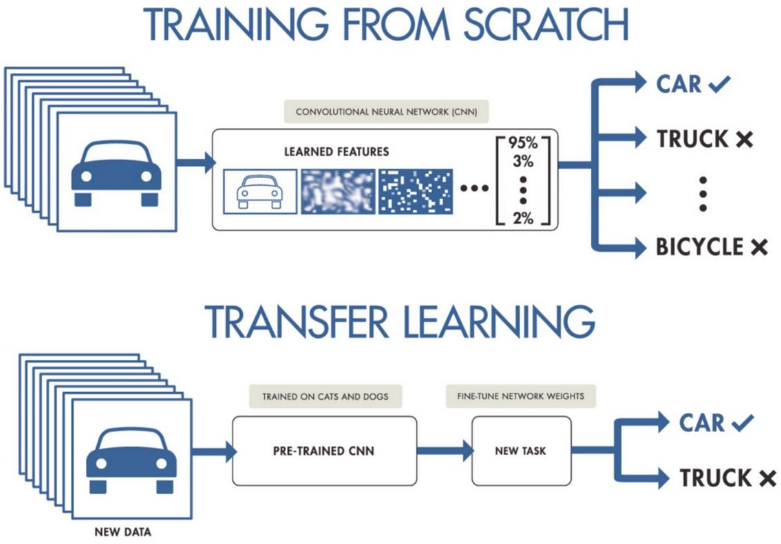



Figure 3: Transfer learning illustration

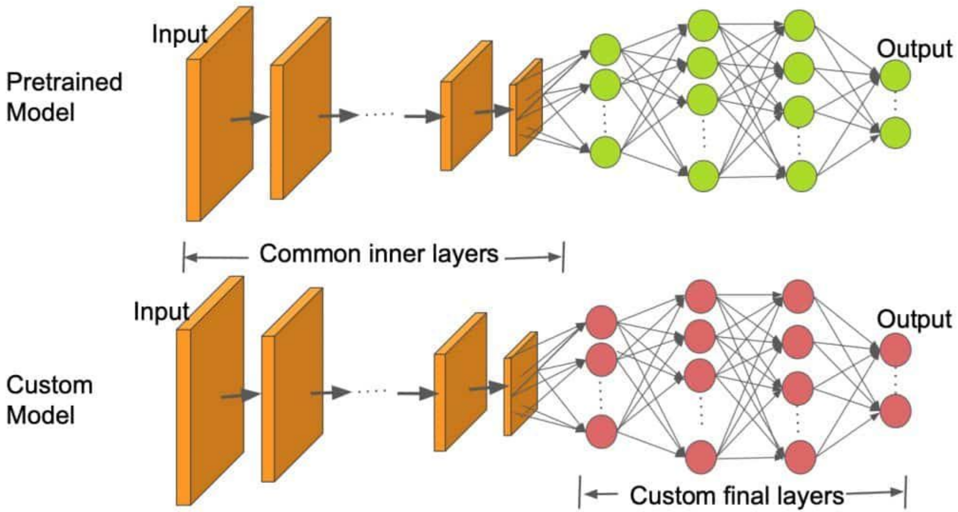
# How does Transfer Learning work?

Select a Pre-trained Model

Freeze the Base Layers

Change the final output layer

Fine-Tuning



# Hyperparameter tuning

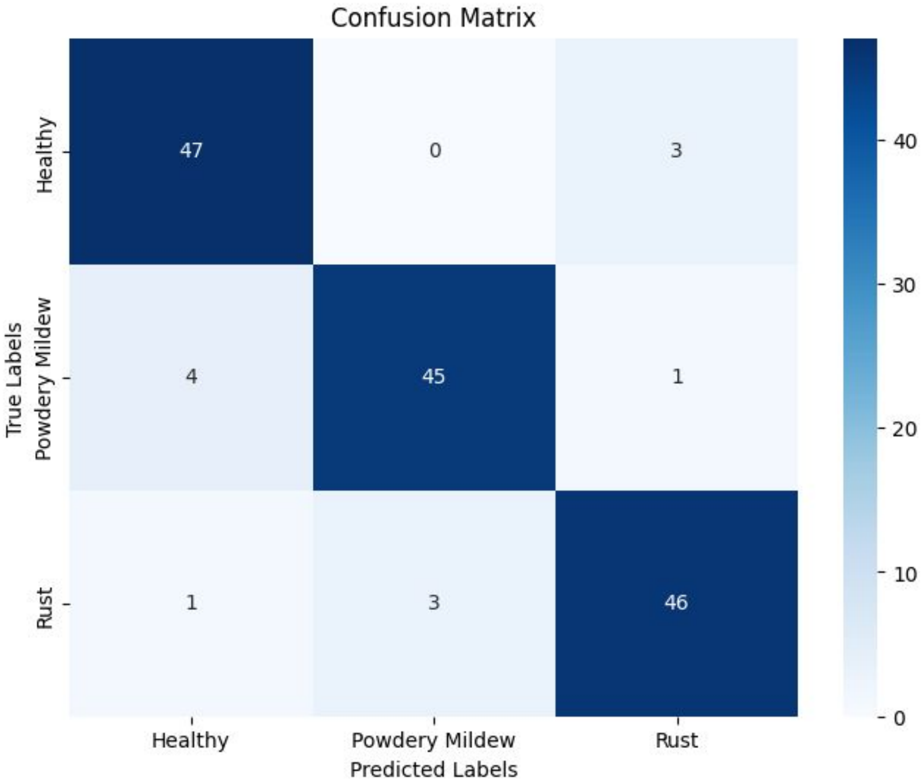
| | Training Loss | Validation Loss | Validation Accuracy (%) | Test Accuracy (%) |
|---|---|---|---|---|
| Batch Size 16 | 0.2 | 0.265 | 90.33 | 91.0 |
| Batch Size 32 | 0.176 | 0.255 | 91.67 | 92.0 |
| Batch Size 64 | 0.16 | 0.273 | 89.0 | 89.67 |

Learning rate fixed at 0.0005

| Learning Rate | Epoch | Validation Loss | Validation Accuracy (%) | Test Accuracy (%) |
|---|---|---|---|---|
| 0.0001 | 20.0 | 0.32 | 85.0 | 84.0 |
| 0.0005 | 20.0 | 0.255 | 91.67 | 92.0 |
| 0.001 | 20.0 | 0.345 | 88.33 | 86.0 |
| 0.005 | 20.0 | 0.45 | 83.0 | 81.5 |

Batch size fixed at 32

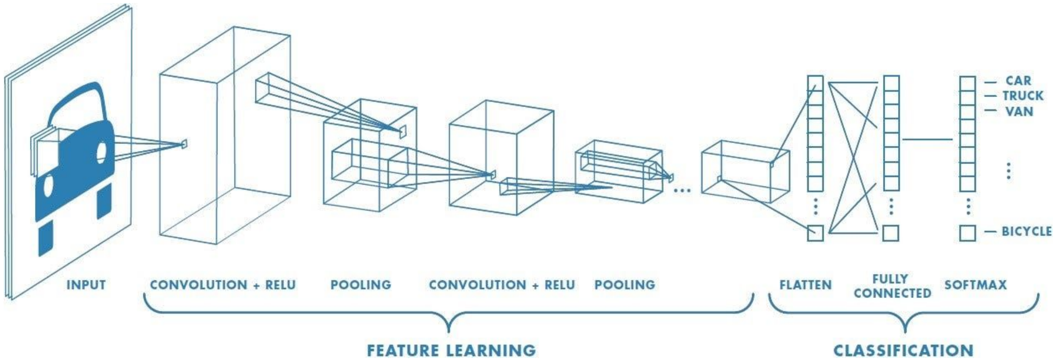
# Results on transfer learning




| | 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 |

Test Loss: 0.2942 Test Accuracy: 92.00%

# CNN STRUCTURE




**First Convolution Kernel Size:** (12 x 5 x 5)

**Second Convolution Kernel Size:** (18 x 3 x 3)

**Third Convolution Kernel Size:** (24 x 3 x 3)

**Activation Function:** ReLU

**Pooling:** Max Pooling (2 x 2)

# Hyperparameter Tuning

**Fixed Parameters During Hyperparameter Tuning:**

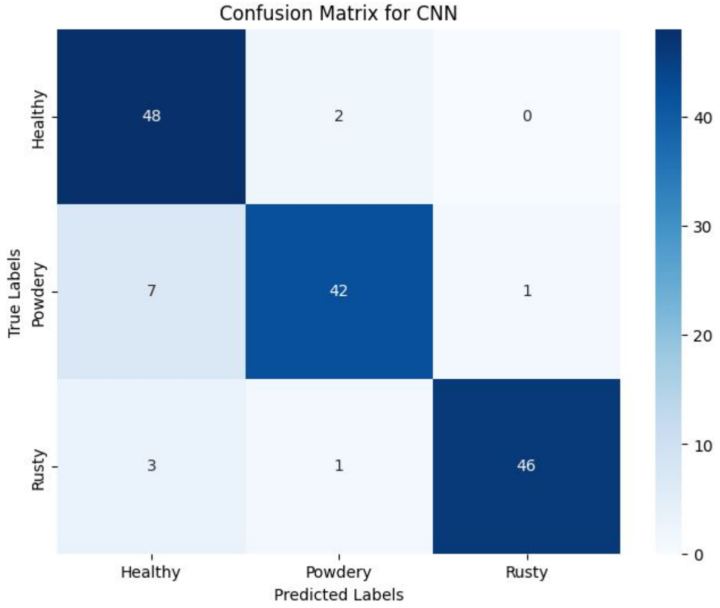
- Stochastic Gradient Descent Optimizer
- Momentum Coefficient = 0.9
- Epoch Number = 20
- Learning Rate = 0.1 in tuning of Batch Size
- Batch Size = 16 in tuning of Learning Rate

| Batch Size | Training Loss | Validation Loss | Validation Accuracy | Test Accuracy |
|---|---|---|---|---|
| 16 | 0.2894 | 0.4591 | 92.16% | 91.44% |
| 32 | 0.3071 | 0.4998 | 83.12% | 83.16% |
| 64 | 0.3123 | 0.5027 | 81.67% | 80.44% |

| Learning Rate | Training Loss | Validation Loss | Validation Accuracy | Test Accuracy |
|---|---|---|---|---|
| 0.005 | 0.3707 | 0.5218 | 81.94% | 78.29% |
| 0.01 | 0.2894 | 0.4593 | 92.00% | 91.67% |
| 0.0125 | 0.3092 | 0.4871 | 88.33% | 87.45% |

**Optimal Batch Size and Learning Rate Values** = (16, 0.01)

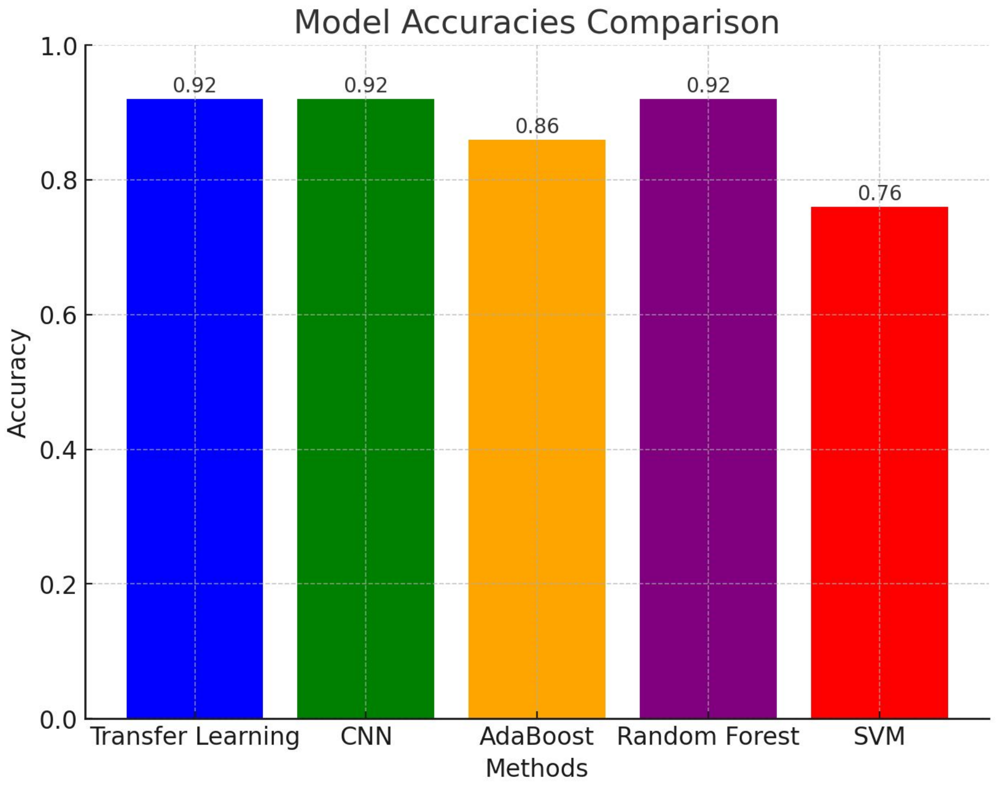
# CNN RESULTS




| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Healthy | 0.83 | 0.96 | 0.89 | 50 |
| Powdery | 0.93 | 0.82 | 0.88 | 50 |
| Rusty | 0.98 | 0.92 | 0.95 | 50 |
| Macro Avg | 0.91 | 0.91 | 0.91 | 150 |
| Weighted Avg | 0.91 | 0.91 | 0.91 | 150 |

**Test Accuracy:** 91.67 %

# BEST Performing Model



# Challenges

## Limited Dataset Size

***Our dataset was relatively small to other datasets, that's why we used more models to evaluate the performance***

# References


[1] NeuralNine, “Image Classification CNN in PyTorch,” YouTube, https://www.youtube.com/watch?v=CtzfbUwrYGI&t=1353s (accessed Nov. 17, 2024).

[2] Jorgecardete, “Convolutional Neural Networks: A comprehensive guide,” Medium, https://medium.com/thedeephub/convolutional-neural-networks-a-comprehensive-guide-5cc0b5eae175 (accessed Nov. 17, 2024).

[3] T. N. Fatyanosa and M. Aritsugi, “Effects of the number of hyperparameters on the performance of Ga-CNN,” *2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, pp. 144–153, Dec. 2020. doi:10.1109/bdcat50828.2020.00016

[4] Ciresan, Dan & Meier, Ueli & Masci, Jonathan & Gambardella, Luca Maria & Schmidhuber, Jürgen. (2011). Flexible, High Performance Convolutional Neural Networks for Image Classification. *International Joint Conference on Artificial Intelligence* IJCAI-2011. 1237-1242. 10.5591/978-1-57735-516-8/IJCAI11-210.

[5] “Image classification using machine learning: Support Vector Machine (SVM),” Medium, Oct. 2020. Available: https://medium.com/analytics-vidhya/image-classification-using-machine-learning-support-vector-machine-svm-dc7a0ec92e01. Accessed: Dec. 21, 2024.

[6]“Image classification using sklearn random forest,” Kaggle, Oct. 2020. Available: https://www.kaggle.com/code/kkhandekar/image-classification-using-sklearn-randomforest. Accessed: Dec. 21, 2024.