Project Report

Pattern Recognition (CSCI-B 655)

Spring 2025

Submitted by

Md Rifat Ullah

Luddy School of Informatics, Computing, and Engineering
Indiana University Indianapolis

INDIANA UNIVERSITY
INDIANAPOLIS

Plant Disease Detection using Image Classification

1. Introduction

Plant diseases significantly impact agricultural productivity and food security. Early detection of plant diseases can play a crucial role in minimizing crop loss and reducing the use of pesticides. This project focuses on the classification of plant health conditions using image data and machine learning techniques. By developing an image-based classification system, we aim to assist in identifying whether a plant is healthy or suffering from common diseases, enabling timely intervention and promoting environmental sustainability. The overarching goal is to prevent disease spread, reduce deforestation, and lower the cost of plant-based goods.

---

2. Dataset Information

The dataset was sourced from Kaggle (https://www.kaggle.com/datasets/rashikrahmanpritom/plant-disease-recognition-dataset/discussion/360986) and is titled "Plant Disease Recognition Dataset." It comprises a total of 1,532 labeled images, categorized into three classes:

- Healthy
- Powdery
- Rust

The dataset was divided into training (85%), testing (10%), and validation (5%) subsets. Preprocessing steps included:

- Image resizing
- Normalization (scaling pixel values)
- Augmentation (e.g., rotation, zoom, flip)

These steps ensured consistency in input dimensions and enhanced the model's ability to generalize.

---

3. Experiments and Models

Four machine learning models were implemented and evaluated:

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN) with Bidirectional LSTM
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)

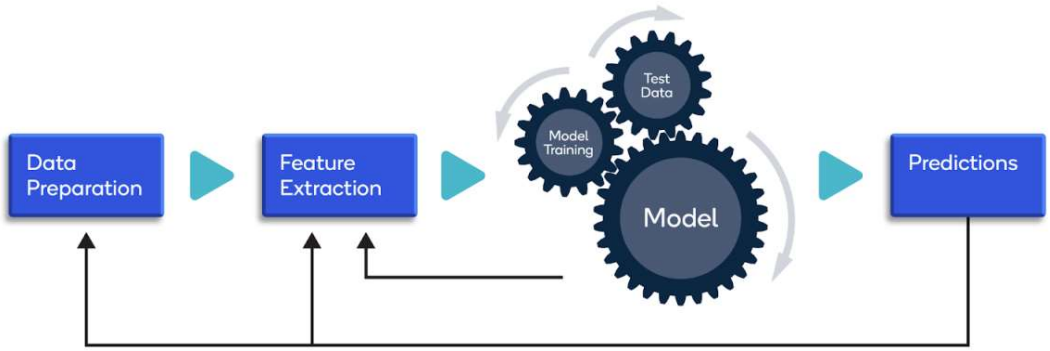


Fig 1: The figure shows how machine learning model train and predict the output.

How does CNN, RNN, SVM and KNN works with image data:

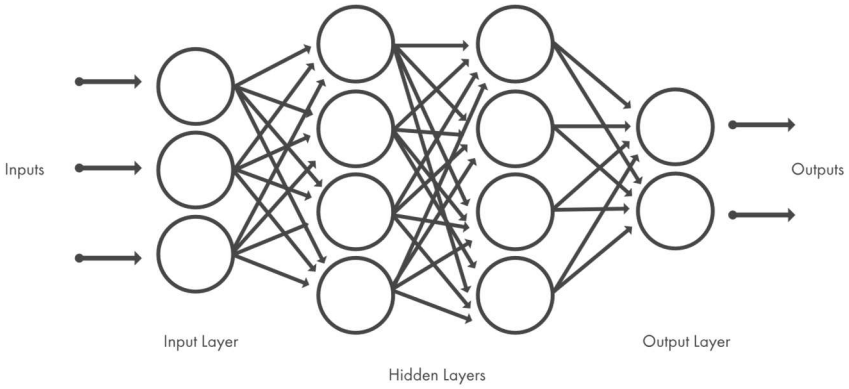


Fig 2: Example of a Neural Network

This subsection explains how each of the Algorithms works with image datasets:

1. Convolutional Neural Network (CNN)

CNN:

CNN is a deep learning model specifically designed for image and spatial data. It automatically extracts hierarchical features from images using layers that detect edges, textures, shapes, and object parts.

How CNN works with images:

1. **Input Image:** CNN takes a 2D or 3D image.
2. **Convolution Layers:** Apply filters (kernels) to detect local patterns like edges, corners.
3. **Activation (ReLU):** Introduces non-linearity.
4. **Pooling Layers:** Downsample feature maps to reduce computation and preserve important info (e.g., MaxPooling).
5. **Flattening:** Converts final feature maps into a 1D vector.
6. **Dense Layers:** Perform final classification using learned features.
7. **Softmax Layer:** Outputs class probabilities.

CNN with images:

CNN keeps the spatial structure of images intact, making it ideal for classification, object detection, and segmentation.

---

2. Recurrent Neural Network (RNN)

RNN:

RNN is a neural network designed to handle sequential data, like text or time-series, by maintaining internal memory via hidden states.

How RNN works with images:

- RNNs are not natively built for images.
- To use RNN on image data:
    1. Flatten or reshape the image into a sequence (e.g., rows or columns).
        - Example: A 64x64x3 image → 64 time steps of 192-length vectors.
    2. Feed this sequence into an RNN (e.g., LSTM or GRU).
    3. The RNN processes each "step" in order, building a temporal understanding.
    4. Output is passed to a dense layer for classification.

Limitations:

- Reshaping breaks spatial structure → poor learning.
- RNNs do not exploit local pixel relationships.
- Slower to train than CNNs.
- Not recommended for image classification.

---

3. Support Vector Machine (SVM)

SVM:

SVM is a classical machine learning algorithm that finds the optimal hyperplane that separates data points from different classes with the maximum margin.

How SVM works with images:

1. **Preprocessing:** All images are resized and flattened into 1D feature vectors.
    - Example: 64x64x3 image → 12,288 features.
2. **Feature space:** Each image becomes a point in this high-dimensional space.
3. **Training:** SVM tries to find the best linear (or kernel-based) boundary to separate classes.
4. **Prediction:** New images are also flattened and classified using this decision boundary.

Challenges:

- High-dimensional data → can be slow, memory-heavy.
- Loses spatial structure.
- Works better with dimensionality reduction (e.g., PCA) or CNN-extracted features.

---

4. K-Nearest Neighbors (KNN)

KNN:

KNN is a lazy learning algorithm that classifies samples based on the majority label of the K most similar training examples.

How KNN works with images:

1. **Preprocessing:** Resize and flatten images into 1D vectors.
2. **Distance Calculation:** For a test image, compute the Euclidean distance to every training image.
3. **K Neighbors:** Identify the K closest training samples.
4. **Voting:** Assign the label that appears most frequently among the neighbors.

Example:

- Given a 64x64 image → flatten to 12,288 pixels.
- Use K=5: find the 5 closest vectors in the training set and pick the most common label.

Limitations:

- No learning phase → slow at prediction time.
- High memory usage.
- Loses spatial relationships.
- Sensitive to pixel intensity differences and noise.

Comparison Summary

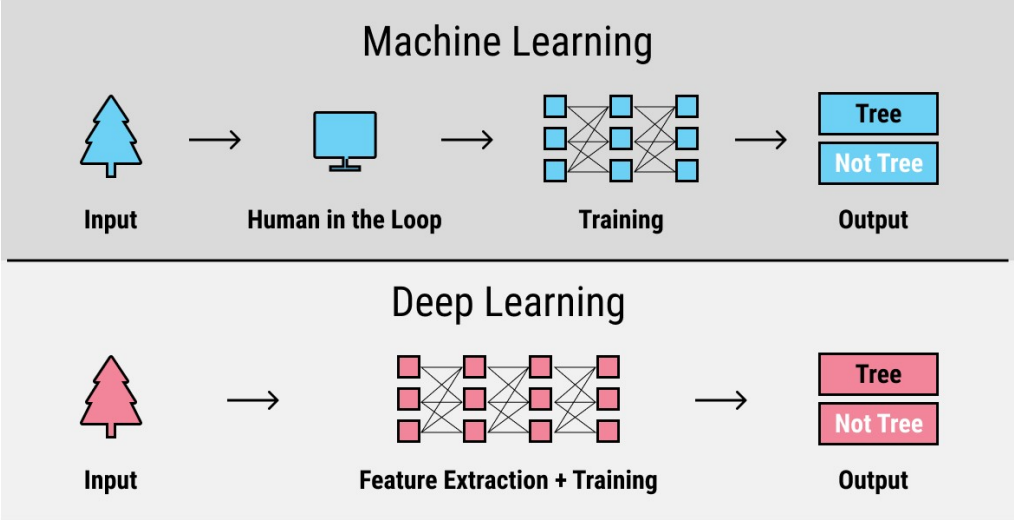


Fig 3: Difference between Machine Learning and Deep Learning. Machine Learning needs human to select the features. On the other hand, Deep Learning does it without humans help.

Algorithm	Learns Spatial Features	Needs Flattening	Handles Raw Pixels	Suitable for Images
CNN	Yes	No	Yes	Best suited
RNN	No	Yes	But poorly	Not suitable
SVM	No	Yes	Yes	Limited use
KNN	No	Yes	Yes	Good for small, simple datasets

Model Hyperparameters:

- **CNN:**
  - Batch size (train generator): 32

- Batch size (model training): 16
- Epochs: 25
- Activation functions: ReLU and Softmax

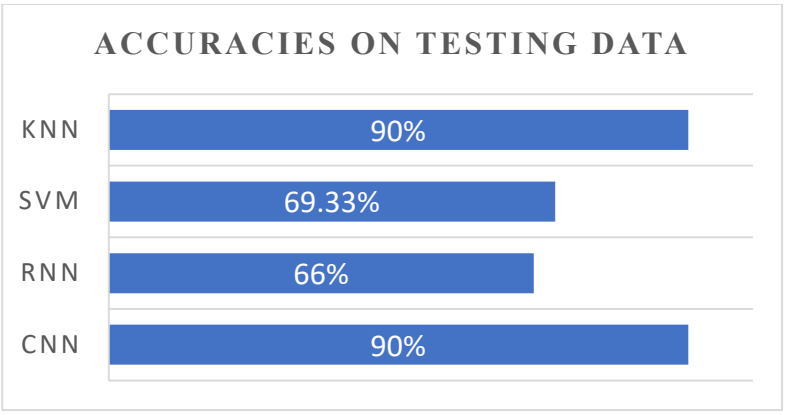
- **RNN:**
  - Bidirectional LSTM
  - Batch size: 32
  - Epochs: 25
  - Optimizer: Adam
  - Loss: Categorical Cross entropy
- **KNN:**
  - Neighbors: 5
  - Distance Metric: Euclidean
- **SVM:**
  - Kernel: Linear

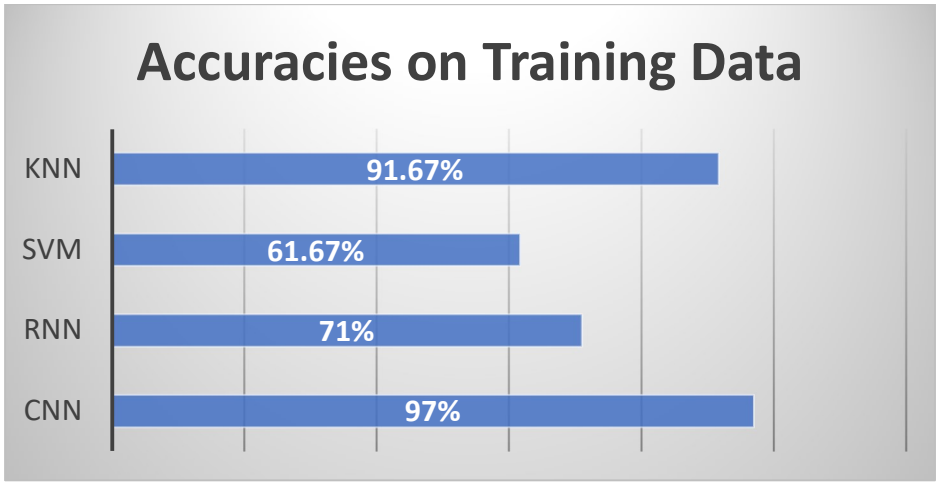
---

4. Results and Result Analysis

Model	Training Accuracy	Testing Accuracy
CNN	97%	90%
RNN	71%	66%
SVM	61.67%	69.33%
KNN	91.67%	90%

Result Comparison:





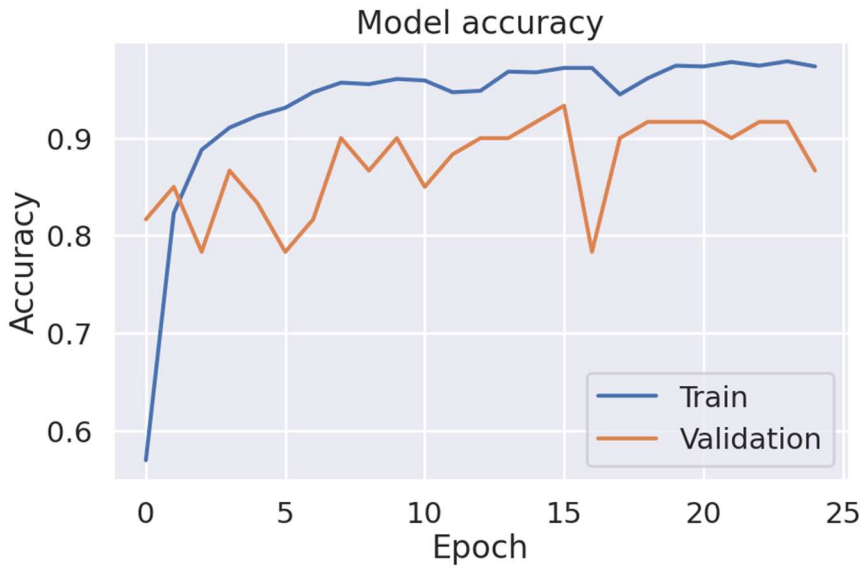


Fig 4: shows the CNN training & validation accuracies at each epoch.

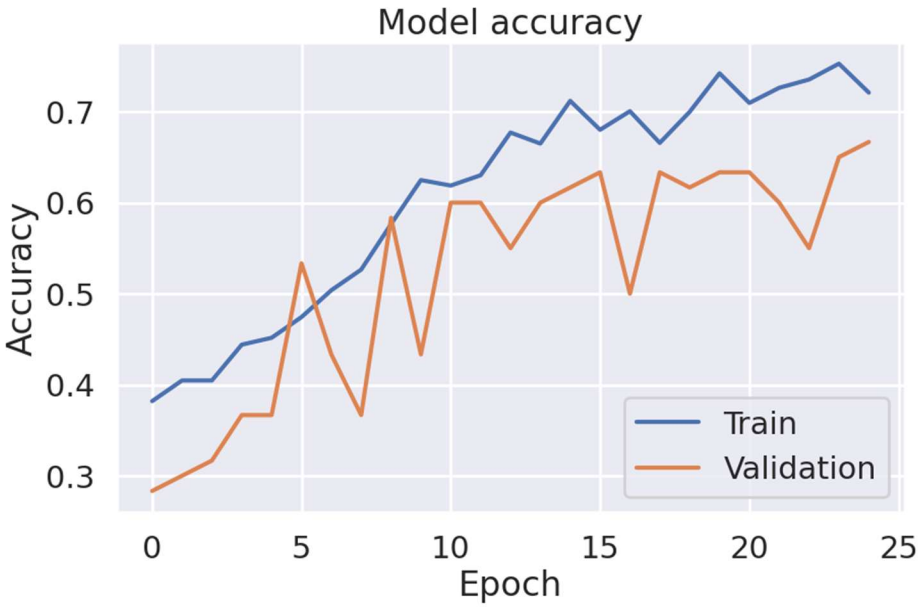


Fig 5: shows the RNN training & validation accuracies at each epoch.

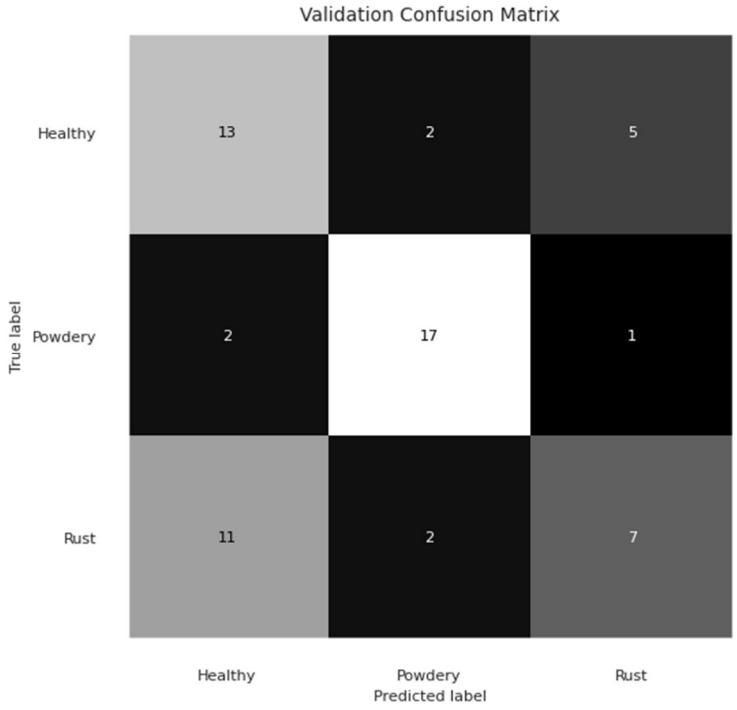


Fig 6: SVM confusion matrix on validation data.

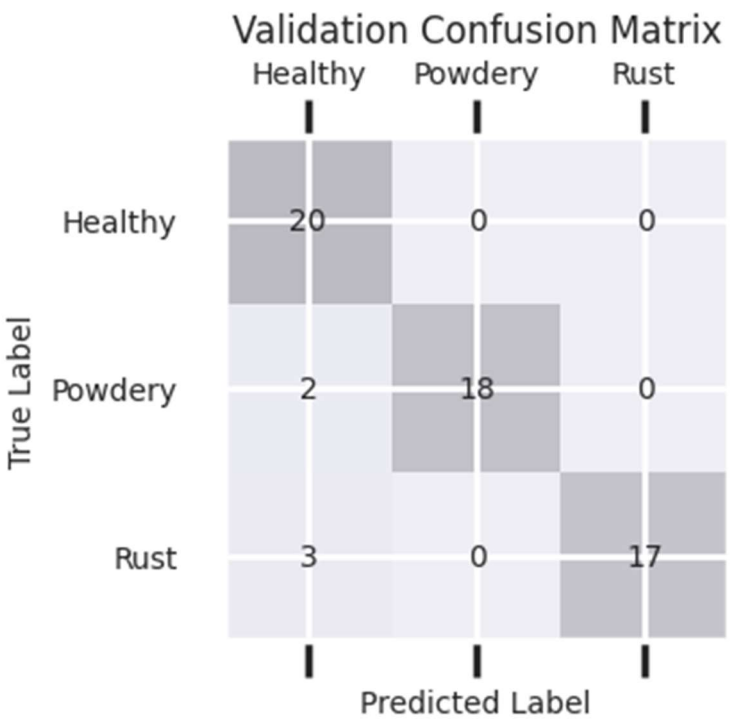


Fig 7: KNN confusion matrix on validation data.

Result Analysis:

- **CNN** achieved the highest training accuracy (97%) and high testing accuracy (90%). This indicates effective learning, though the gap suggests mild overfitting due to the small dataset.
- **KNN** showed excellent generalization, with both training and testing accuracies around 91.67% and 90%, respectively. As a non-parametric method, KNN excels with well-separated and clean datasets, benefiting from the similarity of pixel patterns (not learned features) across image samples.
- **SVM** had relatively low training accuracy (61.67%) but performed better on the test set (69.33%), indicating underfitting yet decent generalization. SVMs are effective when data is linearly separable or when supported with dimensionality reduction.
- **RNN** performed poorly, with 71% training and 66% testing accuracy. This confirms that RNNs are not suitable for image classification tasks, as they are designed for sequential data (e.g., time series, natural language). Reshaping images into sequences disrupts their spatial integrity, leading to suboptimal performance.

5. Limitations

- The limited dataset size may have restricted the full potential of deep learning models, especially CNN and RNN.
- Hardware constraints prolonged training times and possibly hindered experimentation with deeper architectures or extensive hyperparameter tuning.
- The RNN architecture was not ideally suited for image-based tasks, contributing to poor performance.
- No dimensionality reduction or feature engineering was applied before training the SVM and KNN models, which could have improved their results further.

THE END.