A Project Report

on

PADDY CROP DISEASE DETECTION USING MACHINE LEARNING

Submitted in partial fulfillment of the requirements

for the award of degree of

BACHELOR OF TECHNOLOGY

in

Information Technology

by

G. Harshitha (19WH1A1205)

B. Keerthi (19WH1A1208)

Ch. Mithiksha (19WH1A1222)

T. Ramya Sri (19WH1A1247)

Under the esteemed guidance of

Ms. M. Sudha Rani

Assistant Professor

Department of Information Technology

BVRIT HYDERABAD College of Engineering for Women

Rajiv Gandhi Nagar, Nizampet Road, Bachupally, Hyderabad – 500090

(Affiliated to Jawaharlal Nehru Technological University Hyderabad)

(NAAC 'A' Grade & NBA Accredited- ECE, EEE, CSE & IT)

June, 2023

DECLARATION

We hereby declare that the work presented in this project entitled " **PADDY CROP DISEASE DETECTION USING MACHINE LEARNING** " submitted towards completion of the project in IV year II sem of B.Tech IT at "BVRIT HYDERABAD College of Engineering for Women", Hyderabad is an authentic record of our original work carried out under the esteemed guidance of **Ms. M. Sudha Rani, Assistant Professor**, Department of IT.

G. Harshitha (19WH1A1205)

B. Keerthi (19WH1A1208)

Ch. Mithiksha (19WH1A1222)

T. Ramya Sri (19WH1A1247)

BVRIT HYDERABAD

College of Engineering for Women

Rajiv Gandhi Nagar, Nizampet Road, Bachupally, Hyderabad – 500090

(Affiliated to Jawaharlal Nehru Technological University Hyderabad)

(NAAC 'A' Grade & NBA Accredited- ECE, EEE, CSE & IT)

CERTIFICATE

This is to certify that the Major-project report on **" PADDY CROP DISEASE DETECTION USING MACHINE LEARNING"** is a bonafide work carried out by **Ms. G. Harshitha (19WH1A1205), Ms. B. Keerthi (19WH1A1208), Ms. Ch. Mithiksha (19WH1A1222), Ms. T. Ramya Sri (19WH1A1247)** in the partial fulfillment for the award of B.tech degree in **Information Technology, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad** affiliated to the Jawaharlal Nehru Technological University Hyderabad under my guidance and supervision. The results embodied in the major-project work have not been submitted to any other university or institute for the award of any degree or diploma.

Internal Guide
Ms. M. Sudha Rani
Assistant Professor
Department of IT

Head of the Department
Dr. Aruna Rao S L
Professor & HoD
Department of IT

External Examiner

ACKNOWLEDGEMENT

We would like to express our profound gratitude and thanks to **Dr. K. V. N. Sunitha, Principal, BVRIT HYDERABAD** for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. Aruna Rao S L, Professor & Head, Department of Information Technology, BVRIT HYDERABAD** for all the timely support, constant guidance and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide **Ms. M. Sudha Rani, Assistant Professor, Department of IT, BVRIT HYDERABAD** for her constant guidance, encouragement and moral support throughout our project.

Finally, we would also like to thank our Project Coordinators **Dr. P. Kayal, Associate Professor and Ms. K. S. Niraja, Assistant Professor**, all the faculty and staff of the Department of IT who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

G. Harshitha (19WH1A1205)

B. Keerthi (19WH1A1208)

Ch. Mithiksha (19WH1A1222)

T. Ramya Sri (19WH1A1247)

ABSTRACT

Now a days, farmers are facing loss in crop production due to many reasons. One of the major problem is crop diseases. This is due to lack of knowledge about the disease and pesticides or insecticides available in order to control the disease. But finding the disease and providing best remedies requires expert opinion or prior knowledge. This is time consuming and expensive. Therefore highly effective and cost efficient method is needed for fast detection of crop disease. To solve the above issue a Machine Learning model is developing by using Support Vector Machine(SVM) algorithm for disease detection. By analyzing images of paddy leaves, the model extracts relevant features and applies trained model to detect and classify diseases. This model calculates the affected area in the leaf by using cv2 module. The input layer takes the images of the diseased and healthy rice plant and output layer gives disease detection and affected area in the leaf. The remedies provide proper information regarding the pesticide or insecticide to be used in order to cure the disease.

LIST OF FIGURES

LIST OF ABBREVIATIONS

Acronym	Abbreviation
SVM	Support Vector Machine
KNN	K-Nearest Neighbors
VGG	Visual Geometry Group
OpenCV	Open Source Computer Vision Library

TABLE OF CONTENTS

1. INTRODUCTION

In India most of its economy comes from Agriculture itself and it is the second largest producer of wheat and rice. Agriculture plays an important role in the economic growth of every country so it is necessary to ensure its development. Indian agriculture is composed of many crops like paddy, wheat, sugarcane, oilseeds, vegetables and fruits. Due to emerging growth in the population agriculture sector needs a better advancement using the latest technologies. Paddy is one of the important food crops in the world and India is a largest producer of paddy. Paddy is infected by some disease which is caused by fungi, bacteria and viruses. So farmers lose a 37 % estimated average of paddy due to paddy crop diseases every year.

For a better yield, the crop should be healthy. Major issue is farmers are unable to identify the disease properly and they do not know the proper preventive methods in order to control the disease. In recent years, the spread of various diseases in rice plants has increased. There is a variety of plant pathogens such as viral, bacterial, fungal and these can damage different plant parts above and below the ground. However, some abiotic factors such as water, light, radiation, temperature, humidity, atmosphere, acidity, and soil also affect the growth of the plant. Therefore highly effective and cost efficient method is needed for fast detection of crop disease.There is a number of remedial measures are available like varieties of pesticides or insecticides to control the crop diseases and increase the crop production.

But finding the current disease and providing best remedies requires expert opinion or prior knowledge in order to control the disease. This is time consuming and expensive. The presence of disease on the plant is reflected on leaves by showing some specific symptoms related to that disease.This specific symptoms act as a features in order to detect the particular disease. Using these features we are developing a machine learning model which is less expensive and provides accurate results with less span of time while detecting the paddy crop diseases and provides the best remedies such as insecticides or pesticides in order to control the disease. So, it is need of the hour to detect such diseases as earliest as possible.

Paddy crop disease detection systems based on machine learning techniques typically involve the use of computer vision algorithms and image processing tools. These systems employ cameras or other imaging devices to capture images of paddy plants or their specific parts, such as leaves or stems. The acquired images are then processed using sophisticated algorithms that extract relevant features, such as color, texture, shape, and lesion patterns, from the images.

The proposed system aims to leverage the power of machine learning algorithms to accurately identify various diseases that affect paddy crops. By analyzing images of paddy leaves, the system extracts relevant features and applies classification models to determine the presence of diseases. Once the diseases and affected areas are identified, the system offers appropriate remedies or preventive measures to address the specific diseases. These remedies may include recommended fungicides, pesticides, or cultural practices to alleviate the disease and promote healthier crop growth.

The advantages of using machine learning-based paddy crop disease detection systems are manifold. Firstly, they enable rapid and accurate detection of diseases, allowing farmers to take timely preventive or curative measures. Secondly, these systems can detect diseases at an early stage, even before visible symptoms appear, thereby improving the chances of successful disease management. Additionally, machine learning models can be continuously updated and refined as new data becomes available, enhancing their accuracy and reliability over time.

In conclusion, the integration of machine learning and image processing techniques has opened up new possibilities for paddy crop disease detection. These advanced systems offer faster, more objective, and accurate disease diagnosis, enabling farmers to make informed decisions and mitigate the impact of diseases on their crops. As technology continues to advance, the development of robust and user-friendly disease detection tools for paddy crops holds immense potential for enhancing agricultural productivity and ensuring food security in the face of evolving plant diseases.

1.1 Objective

Sometimes farmers are forced to sell the crop even though it is contaminated with disease and when consumed by general public results in diseases. So this is where our model comes into action it quickly diagnoses the disease of the plant and provides instant results and solutions which keeps up the quality of crop. Using this model, they were able to determine the disease, the level of disease, and the remedies required to cure the disease.

1.2 Problem Definition

Developing a Machine Learning model that detects the paddy crop leaf disease, identifies the affected area in the leaf and suggests remedies to cure the disease.

1.3 Modules

1. Pre-Processing
2. Feature Extraction
3. Model Training
4. Model Evaluation

2. LITERATURE SURVEY

Shruthi U, Nagaveni V, Raghavendra B K, et al.[1] proposed “A review on machine learning classification techniques for plant disease detection”. They have implemented the model using different techniques. The different classification methods are Artificial Neural Network (ANN) classification technique, K-nearest neighbour classification technique, Convolutional Neural Network classification, fuzzy classifier and Support Vector Machine (SVM) classification methods. From the above mentioned techniques they concluded that Convolution Neural Network provides high accuracy compared to other methods and detect more number of diseases in multiple crops. But the model is unable to identify the affected area in the leaf.

V. Vanitha, et al.[2] proposed “Rice disease detection using deep learning” which proposes an automatic disease detection using deep neural network. They developed a model which is capable of detecting 3 different disease of paddy and also detect the healthy leaf image. Bacterial Leaf Blight disease, Brown Spot disease Sheath Blight disease occurs on rice leaf. The symptoms of different diseases are seen at different parts of the rice plant such as leaf, stem and grain. So, they have considered all these parts while capturing images. To prevent the model from being confused between dead parts and diseased parts of rice plant, they have collected enough images of dead leaf, dead stem and dead grain of rice plants. Images of the dead parts of the plant are compared with the dead parts of the healthy plant. They consider a total of three classes. The simultaneous occurrence of diseases has not been considered. The dataset was trained with three CNN architectures and achieved a high accuracy of 89.53 %. Sometimes it is unable to identify the important fetaures in the dead parts of the diseased leaf.

Shamim Mahbub, Md. Abu Nasim, Md. Jahid Hasan, Md. Shahin Alom, et al.[3] proposed “Rice disease identification and classification by integrating support vector machine with deep convolutional neural network”. Their model is developed to identify the rice disease and help the farmers to take proper decision to control the disease and also help them to increase production. Here they considered nine different diseases of rice which are most commonly affected diseases mainly in Bangladesh as well as other countries. These are Bacterial Leaf Blight, Rice Blast, Brown spot, False Smut, Leaf Smut, Leaf Scald, Sheath Blight, Tungro and Healthy. Generally, the color, shape & size of rice blast disease depends on varietal resistance, environmental conditions and the age of the

lesions. The pathogen mostly attacked in these organ of the rice plants and those organs are Leaf and Collar, and Neck and Node. Though they put this two affected organs in same class but they trained them separately. They have built a AI model by integration of Support Vector Machine (SVM). This model identifies and classifies nine types of rice diseases with an accuracy of 81.5%. The model doesn't suggests any remedies to cure the disease.

Anuradha, et al.[4] badge proposed "Crop disease detection using Machine learning: Indian Agriculture". This model depicts how diseases affect yield and how machine learning techniques assisted in detecting the disease and assisting farmers in taking the necessary action. Periodically images are obtained by remote sensing. RGB values of the monitored images are extracted and compared with threshold images. If the threshold is greater or less than given value, histogram analysis and edge detection techniques are used to identify particular plant diseases. They considered wheat crop for their model. They used canny's edge detection algorithm for the efficient detection of crop disease by taking image of crop.

Mr Ramachandra Hebbar, Mr. P. V Vinod, Shima Ramesh, Niveditha.M, Pooja R, Shashank.N, Prasad Bhat.N, et al.[5] proposed "Plant disease detection using machine learning". They proposed a model which includes various phases of implementation namely dataset creation, feature extraction,training the classifier and classification. They created datasets of diseased and healthy leaves. Those images are collectively trained under Random Forest for classification. For extracting features of an image they have used Histogram of an Oriented Gradient (HOG). So, finally they concluded that using machine learning to train the large data sets gives a clear way to detect the disease present in plants in a colossal scale.

Md. Ashiqul Islam, Md. Nymur Rahman Shuvo, Muhammad Shamsojjaman Shazid Hasan, Md. Shahadat Hossain, Tania Khatun, et al.[6] proposed "Automated Convolutional Neural Network Based Approach for Paddy Leaf Disease Detection". This model is to provide the best results for paddy leaf disease detection through an automated detection approach with the deep learning CNN models that can achieve the highest accuracy instead of the traditional lengthy manual disease detection process where the accuracy is also greatly questionable. It has analyzed four models such as VGG-19, Inception-Resnet-V2, ResNet-101, Xception, and achieved better accuracy from

Inception-ResNet-V2 is 92.68 %.

Nargis Parven, Muhammad Rashiduzzaman, Nasrin Sultana, Md. Touhidur Rahman, Md. Ismail Jabiullah et.al[7] proposed “Detection and Recognition of Paddy Plant Leaf Diseases using Machine Learning Technique”. The main aim of this model is quick and perfect detection of various paddy leaf diseases in the primary stage. To develop this system at first unnecessary background of captured images are eliminated using mask, then output is fed into segmentation part and separate the group of object for feature extraction. After that, extracted the color and calculated texture features for disease portion using color feature and texture feature then create trained data-set and classified this feature using SVM and finally identify the diseases. Sometimes it is difficult to calculate accurate texture features.

Tejas Tawde, Lobhas Verekar, Shailendra Aswale, Kunal Deshmukh, Ajay Reddy et.al[8] proposed “Rice Plant Disease Detection and Classification Techniques : A Survey”. This research paper provides a survey on various techniques and briefly discusses significant aspects of different classifiers and techniques used to detect rice diseases. Papers of the last decade are studied thoroughly, including the work carried on various rice plant diseases, and a survey is presented based on essential aspects. The survey focuses to distinguish different methods based on the classifier used. The survey gives insights of the different techniques used for the identification of rice plant disease. In addition, a hardware prototype and model using Convolutional Neural Network (CNN) is proposed that detects rice disease. It further identifies the type of rice disease into rice blast, rice blight, brown spots, leaf smut, tungro and sheath blight.

Muhammad Shoaib, Babar Shah, Shaker EI-Sappagh, Akhtar Ali, Asad Ullah et.al[9] proposed “An advanced deep learning models-based plant disease detection: A review of recent research”. In this paper, the recent advancements in the use of ML and DL techniques for the identification of plant diseases are explored. This study addresses the challenges and limitations associated with using ML and DL for plant disease identification, such as issues with data availability, imaging quality, and the differentiation between healthy and diseased plants. It provides valuable insights for plant disease detection researchers, practitioners, and industry professionals by offering

solutions to these Deep challenges and limitations, providing a comprehensive understanding of the current state of research in this field, highlighting the benefits and limitations of these methods, and proposing potential solutions to overcome the challenges of their implementation.

Amritha Haridasan, Jeena Thomas, Ebin Deni Raj et.al[10] proposed “Deep learning system for paddy plant disease detection and classification”. The system is for the recognition of rice plant diseases adopts a computer vision-based approach that employs the techniques of image processing, machine learning, and deep learning, reducing the reliance on conventional methods to protect paddy crops from diseases like bacterial leaf blight, false smut, brown leaf spot, rice blast, and sheath rot, the five primary diseases that frequently plague the Indian rice fields. Following image pre-processing, image segmentation is employed to determine the diseased section of the paddy plant, with the diseases listed above being identified purely on the basis of their visual contents. An integration of a support vector machine classifier and convolutional neural networks are used to recognize and classify specific varieties of paddy plant diseases. With ReLU and softmax functions, the suggested deep learning-based strategy attained the highest validation accuracy of 0.9145. Following recognition, a predictive remedy is recommended, which can assist agriculture-related individuals and organizations in taking suitable measures to combat these diseases.

3. SYSTEM ANALYSIS & DESIGN

3.1 Architecture Design

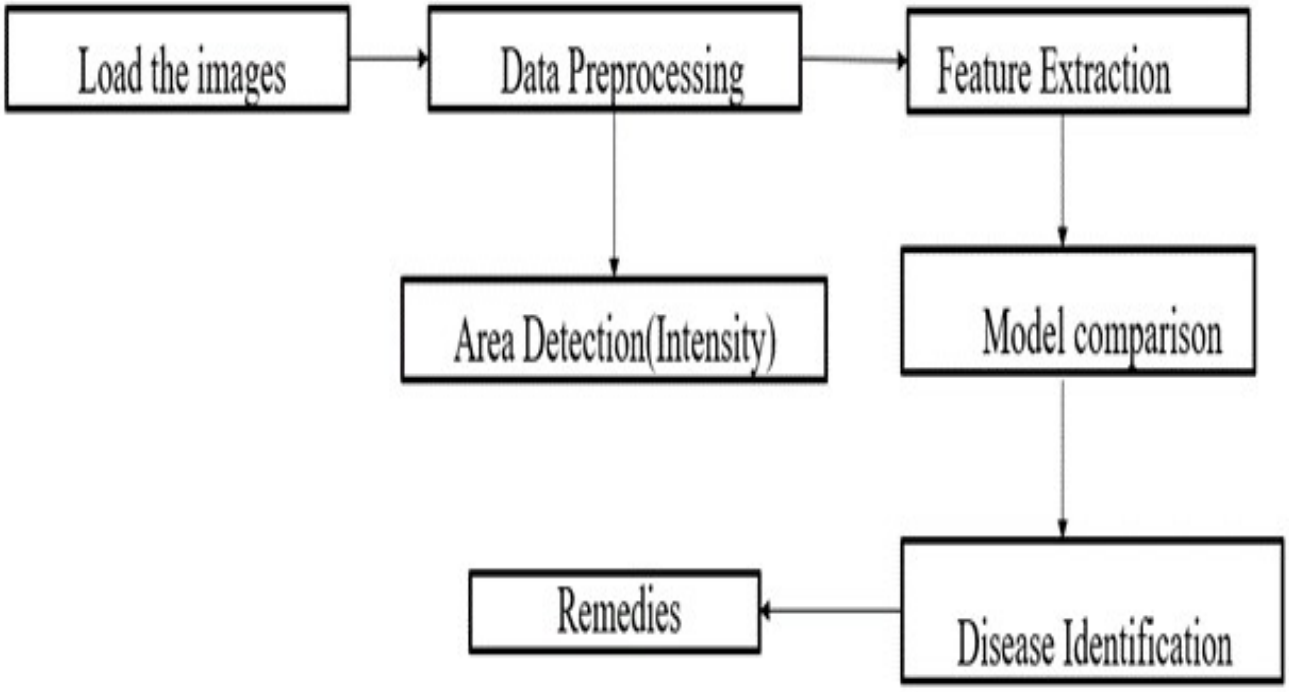


Figure 3.1.1: Block Diagram of Paddy Crop Disease Detection

1. Data Preprocessing: The code reads the images from the dataset directory using ‘cv2.imread‘ and resizes them to a fixed size of 224x224 pixels using ‘cv2.resize‘. The images are normalized by dividing the pixel values by 255.0 to bring them within the range of [0, 1]. The labels are encoded using scikit-learn’s ‘LabelEncoder‘ to convert categorical labels into numerical format. The dataset is shuffled using ‘shuffle‘ from scikit-learn to ensure randomness.

2. Feature Extraction: The VGG16 model from TensorFlow's Keras applications is loaded without the fully connected layers using 'VGG16' function. The pre-trained VGG16 model is used to extract features from the preprocessed images using 'base_model.predict'. The extracted features are flattened to create feature vectors for each image using 'reshape'.

3. Model Comparison:

• The SVM (Support Vector Machine) classifier is used for disease classification.

• Grid search is performed using 'GridSearchCV' to find the best hyperparameters for the SVM model.

• The best SVM model is obtained from the grid search results.

• Accuracy, classification report, and confusion matrix are computed for evaluating the SVM model's performance.

• The Decision Tree classifier and K-Nearest Neighbors (KNN) classifier are also used for disease classification.

• Accuracy and confusion matrix are calculated for the Decision Tree and KNN models.

• The Random Forest classifier is used as another model for disease classification.

• Accuracy and confusion matrix are computed for the Random Forest model.

4. Disease Identification: - After training the SVM model, an example image is loaded, preprocessed, and passed through the trained model to predict the disease label. - The predicted label is transformed back to the original disease name using 'label_encoder.inverse_transform'. - The predicted disease name is displayed, and the image is visualized along with the predicted disease name. - From the preprocessed image, the level of area is affected in the leaf is also identified. - Remedies specific to each disease can be printed based on the predicted disease name.

Overall, the project demonstrates a pipeline for preprocessing the dataset, extracting features using the VGG16 model, training multiple classifiers, evaluating their performance, and using the trained model for disease identification in new images.

3.2 Technologies

Scikit-learn

Scikit-learn, also known as sklearn, is a popular machine learning library in Python. It provides a wide range of tools and algorithms for tasks such as classification, regression, clustering, and dimensionality reduction. In this project, scikit-learn is used for the following purposes:

1. Data preprocessing: Scikit-learn's ‘LabelEncoder()‘ is employed to encode the categorical class labels into numerical format for training the machine learning models. This step ensures that the labels can be properly processed by the models.

2. Dataset splitting: The ‘train_test_split()‘ function from scikit-learn is used to split the dataset into training and testing sets. This is a common practice in machine learning to evaluate the performance of the trained models on unseen data.

3. Model evaluation: Scikit-learn provides various evaluation metrics and functions to assess the performance of machine learning models. In the code, the ‘accuracy_score()‘ and ‘classification_report()‘ functions are used to calculate the accuracy and generate a classification report for the SVM model.

4. Hyperparameter tuning: The ‘GridSearchCV()‘ function from scikit-learn is utilized for hyperparameter tuning of the SVM model. It performs an exhaustive search over a specified parameter grid and selects the best combination of hyperparameters based on cross-validation performance.

5. Alternative models: Scikit-learn is also used to train alternative machine learning models such as Decision Tree, K-Nearest Neighbors (KNN), and Random Forest. These models are instantiated from scikit-learn's respective classes (‘DecisionTreeClassifier()‘, ‘KNeighborsClassifier()‘, ‘RandomForestClassifier()‘), trained on the extracted features, and evaluated using the ‘score()‘ method.

Overall, scikit-learn enhances the functionality of the code by providing powerful tools for data preprocessing, model training, evaluation, and hyperparameter tuning, allowing for a comprehensive and robust machine learning workflow.

Keras

Keras is a popular open-source deep learning framework that simplifies the development of neural network models. It provides a user-friendly, high-level API for building and training models, abstracting away low-level details. With Keras, we can easily define and stack layers to create complex architectures. It supports both sequential and functional model building, making it flexible for various tasks. Keras is backed by TensorFlow, allowing you to leverage TensorFlow's ecosystem and resources. It also provides GPU acceleration for faster training. Overall, Keras is a powerful tool that enables rapid prototyping and experimentation in machine learning.

VGG16 Model

VGG16 is a pre-trained convolutional neural network (CNN) model that is widely used for image classification tasks. It is part of the VGG (Visual Geometry Group) model family, which was developed by the Visual Geometry Group at the University of Oxford.

In this project, the VGG16 model from the 'tensorflow.keras.applications' module is used for feature extraction. Here's a breakdown of the VGG16-related steps in the code:

1. Model loading: The 'VGG16()' function is called to load the pre-trained VGG16 model. It takes several optional arguments, such as 'weights', 'include_top', and 'input_shape'. In the code, the 'weights' argument is set to ''imagenet'', which initializes the model with weights pre-trained on the ImageNet dataset. The 'include_top' argument is set to 'False' to exclude the fully connected layers at the top of the network. The 'input_shape' argument specifies the shape of the input images (224x224 pixels with 3 color channels).

2. Feature extraction: The loaded VGG16 model is used to extract features from the input images. The 'predict()' method is called on the VGG16 model, passing the training and testing images as inputs. This returns the output feature maps generated by the VGG16 model for each image.

3. Flattening: The extracted feature maps are reshaped using the ‘reshape()‘ function to convert them into a flat feature vector. This step is necessary to prepare the features for input into a classifier.

By utilizing the pre-trained VGG16 model, the code benefits from the rich representation learned by the model on a large-scale dataset like ImageNet. The extracted features can then be used as input to train and evaluate different classifiers, such as Support Vector Machines (SVM), Decision Trees, K-Nearest Neighbors (KNN), or Random Forest, as demonstrated in the code. This approach allows for leveraging the powerful feature extraction capabilities of VGG16 and applying them to the specific image classification task at hand.

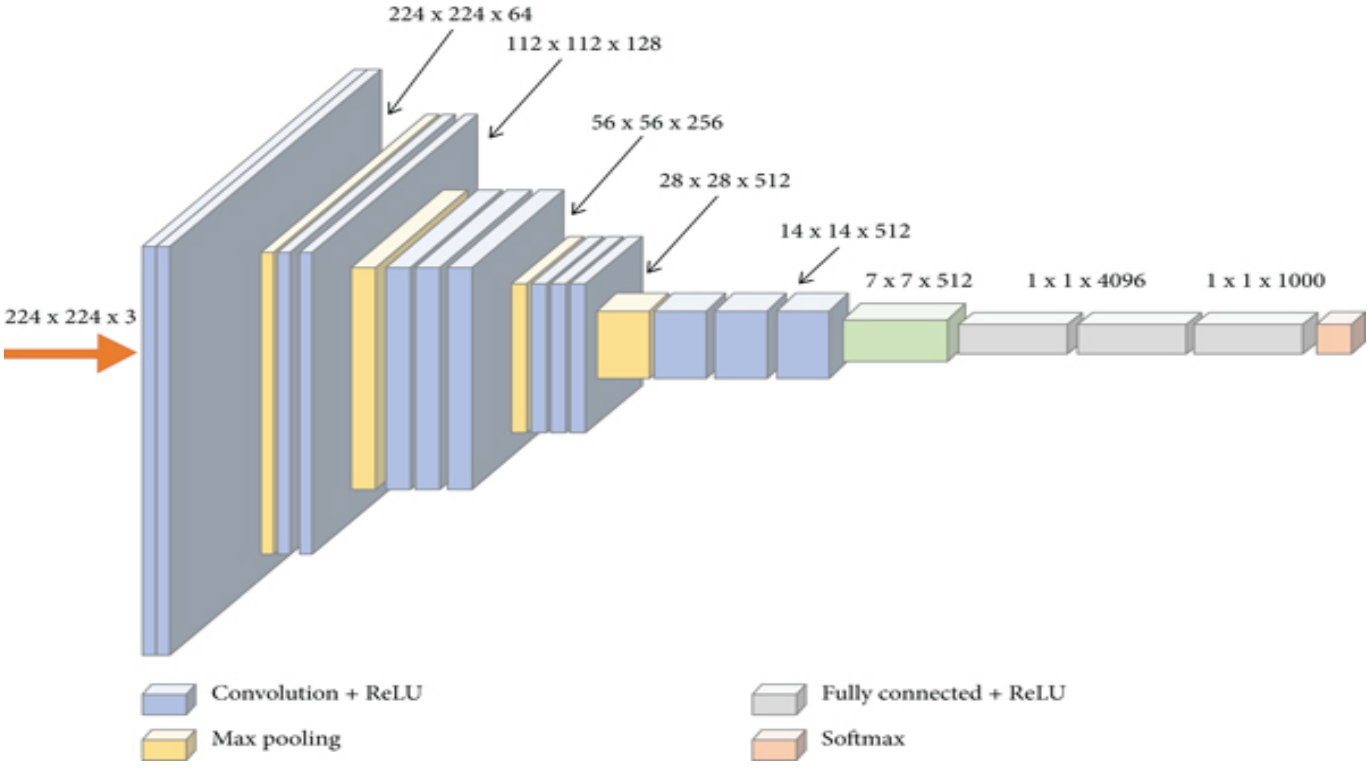


Figure 3.2.1 VGG16 Model Architecture

OpenCV

OpenCV (Open Source Computer Vision Library) is utilized in the project to handle image-related tasks such as loading, resizing, and preprocessing. Specifically, OpenCV is used to:

1. Load images: OpenCV's 'cv2.imread()' function is used to read image files from the dataset directories.

2. Resize images: The 'cv2.resize()' function is applied to resize the loaded images to a fixed size (e.g., 224x224). This step ensures that all images have the same dimensions, which is often necessary for training machine learning models.

3. Image preprocessing: OpenCV is involved in normalizing the pixel values of the images using the expression 'images.astype('float32') / 255.0'. This operation scales the pixel intensities to a range of [0, 1], which is a common practice for training neural networks.

4. Display images: The 'plt.imshow()' function, which internally uses OpenCV, is employed to display the resized images after prediction. This allows visual inspection of the predicted disease label alongside the corresponding image.

OpenCV plays a crucial role in efficiently handling image data and performing necessary operations for image classification tasks in the given code.

Figure 3.2.2: OpenCV

3.3 UML Diagrams

UML (Unified Modeling Language) is a visual language that provides a way for software engineers and developers to construct, document, and visualize software systems. These visual representations help software developers to understand potential outcomes or errors in programs, saving time and enhancing collaboration among team members.

Usecase Diagram

A usecase diagram is a graphical depiction of a user interactions with a system. Use-case diagrams describe the high-level functions and scope of a system. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

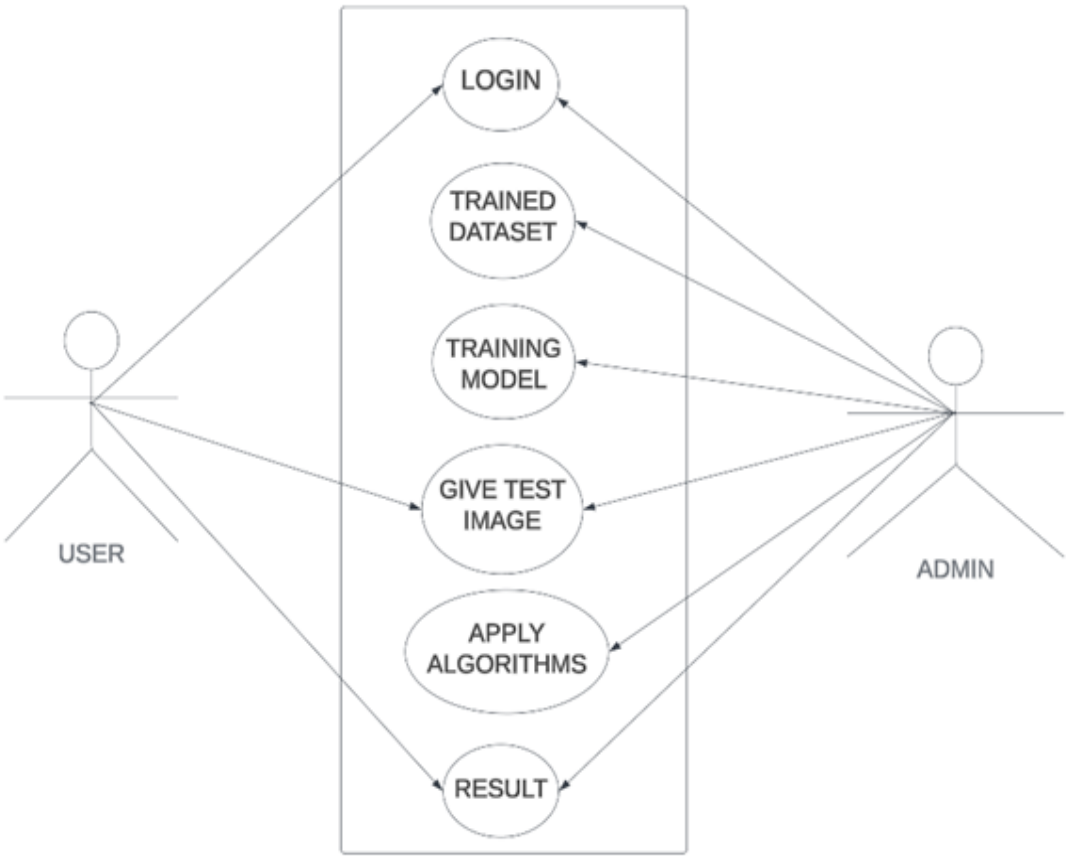


Figure 3.3.1: Usecase Diagram

It represents the user or farmer who interacts with the paddy crop disease detection system. The user captures an image of a paddy crop using a camera or imaging device.This module receives the captured image and performs necessary processing operations like resizing, noise reduction, and enhancement. The preprocessed image is prepared for further analysis and feature extraction. This module extracts relevant features from the preprocessed image. The module analyzes the preprocessed image and extracts the desired features required for disease classification.

This component consists of machine learning algorithms, such as SVM, KNN, decision trees, Random Forest which have been trained on a dataset of labeled images representing healthy and diseased paddy plants. It classifies the paddy crop based on the extracted features. The machine learning classifier analyzes the extracted features and predicts whether the paddy crop is healthy or diseased, and in some cases, identifies the specific disease affecting the crop.This module provides the final output to the user, presenting the classification results and any relevant information about the detected disease, if applicable.

Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how and in what order a group of objects works together. Sequence diagrams describe the work flow in a systematic order. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagram gives the information about what is happening in the each phase. These are widely used to document a system's requirements and to flush out a system's design.

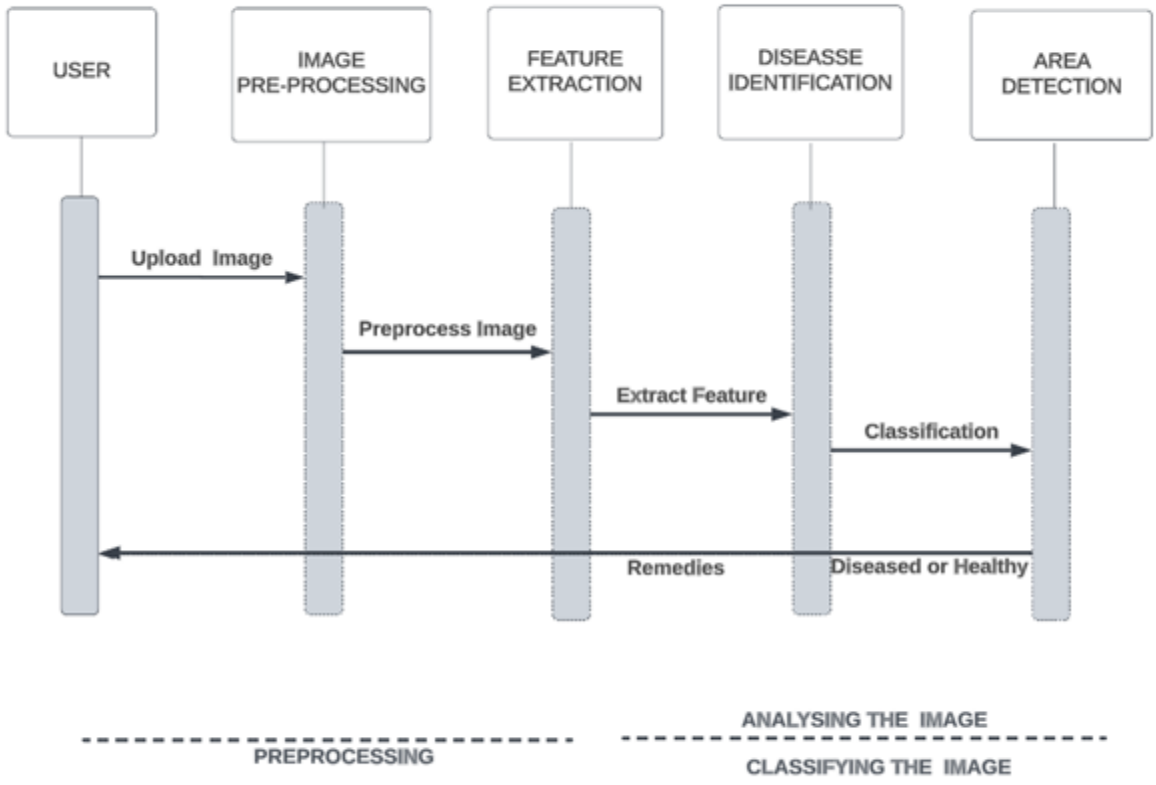


Figure 3.3.2: Sequence Diagram

The diagram starts with the User initiating the disease detection process. The user captures an image of a paddy crop and sends it to the Paddy Crop Disease Detection System. Upon receiving the image, the system passes it to the Image Processing Module. This module performs necessary processing operations, such as resizing, noise reduction, and enhancement, on the image. The message labeled Image Preprocessing indicates the transfer of the preprocessed image from the system to the image processing module. Once the image is preprocessed, it is sent to the Feature Extraction Module.

This module analyzes the preprocessed image and extracts relevant features. The message labeled Extract Features represents the transfer of the preprocessed image to the feature extraction module. After feature extraction, the extracted features are passed to the Machine Learning Classifier component.

This component incorporates machine learning algorithms, such as SVMs, KNN, Random Forest or decision trees, which have been trained on a dataset of labeled images. The classifier uses the extracted features to classify the paddy crop as either healthy or diseased and, in some cases, identify the specific disease affecting the crop. The message labeled Disease Identification indicates the transfer of the extracted features to the machine learning classifier.

Upon completing the disease classification, the classifier sends the classification results to the Output Module. This module prepares the final output, which includes the classification outcome (healthy or diseased) and any relevant information about the detected disease and the level of area affected in the leaf will suggest the remedies based on the disease.

3.4 System Requirements

Hardware

1) Processor - Intel core i5
2) Memory(RAM) - 8GB
3) Hard Disk - 500GB

Software

1) Operating System: Windows 10
2) Python 3.6
3) IDE: Google Colab

4. MODULES

Image Pre-processing

Data preprocessing is the process of preparing and transforming raw data to make it suitable for analysis and modeling tasks. In the above code, data preprocessing steps are performed to prepare the dataset for training a machine learning model for classifying paddy crop leaf diseases.

Here is an explanation of the data preprocessing steps performed in the code:

1. Loading Images: The code uses the OpenCV library to load images from the dataset. It iterates over each class folder and reads the image files within each folder.

2. Resizing Images: The loaded images are resized to a fixed size of 224x224 pixels using the ‘cv2.resize()‘ function. Resizing the images ensures that all images have the same dimensions, which is a requirement for many machine learning models.

3. Creating Image and Label Lists: Empty lists, ‘images‘ and ‘labels‘, are initialized to store the preprocessed images and their corresponding labels. The resized images are appended to the ‘images‘ list, and the class folder names are appended to the ‘labels‘ list.

4. Conversion to Numpy Arrays: The ‘images‘ and ‘labels‘ lists are converted to numpy arrays using ‘np.array()‘. Numpy arrays provide efficient storage and manipulation of the data.

5. Normalization: The pixel values of the images are normalized to the range [0, 1] by dividing them by 255.0. Normalization helps in standardizing the input data and can improve model training and convergence.

6. Label Encoding: The class labels are encoded using the ‘LabelEncoder‘ from scikit-learn. Label encoding converts the class labels into numeric values, which is required by many machine learning algorithms.

7. Data Splitting: The preprocessed dataset is split into training and testing sets using the

‘train_test_split()‘ function from scikit-learn. This splitting helps evaluate the trained model’s performance on unseen data.

The data preprocessing steps in the above code ensure that the input images are resized, normalized, and transformed into a suitable format (numpy arrays) for further model training. Additionally, label encoding and data splitting help in preparing the labels and splitting the data into separate sets for training and evaluation.

Feature Extraction

Feature extraction is the process of transforming raw data (in this case, images) into a representative set of numerical features that capture relevant information for the given task. In the above code, feature extraction is performed using a pre-trained VGG16 model.

1. Pre-trained VGG16 Model: The code utilizes the VGG16 model, which is a deep convolutional neural network (CNN) that has been trained on a large image dataset. The model has learned to extract hierarchical and discriminative features from images.

2. Loading the Pre-trained Model: The code loads the pre-trained VGG16 model using the 'VGG16' class from the Keras library. The 'include_top=False' argument ensures that the fully connected layers (top layers) of the model are not loaded, leaving only the convolutional layers.

3. Extracting Features: The pre-trained VGG16 model is used to extract features from the preprocessed images. The 'base_model.predict()' method is applied to the preprocessed training and testing images, which results in obtaining the feature vectors for each image.

4. Benefits of Feature Extraction: By using a pre-trained model for feature extraction, we can leverage the model's ability to capture high-level image features. This saves time and computational resources compared to training a deep learning model from scratch on a large dataset. The pre-trained model has already learned general image representations, which can be useful for a similar task of classifying paddy crop leaf diseases.

5. Transfer Learning: Feature extraction is a common technique in transfer learning. By utilizing the learned features from a pre-trained model, we can transfer the knowledge gained on a source task (e.g., ImageNet dataset) to a target task (paddy crop leaf disease classification). This can improve the model's performance, especially when the target task has limited training data.

In summary, feature extraction using a pre-trained model like VGG16 helps in capturing relevant and discriminative features from images. It leverages the learned representations from a large dataset, saving time and resources. These extracted features can be used as input for training a separate classifier (in this case, SVM) to perform the classification task effectively.

Model Training

Model training is the process of training a machine learning model on a labeled dataset to learn the underlying patterns and relationships between input features (in this case, extracted image features) and their corresponding target labels (crop leaf disease classes). The goal is to optimize the model's parameters so that it can make accurate predictions on unseen data.

In the above code, four algorithms are used for model training: Support Vector Machines (SVM), Decision Tree, K-Nearest Neighbors (KNN), and Random Forest. Let's discuss why these algorithms were chosen and their usefulness in the given code:

1) Support Vector Machines (SVM) :

SVMs are effective in solving classification problems, making them suitable for detecting whether a paddy crop is healthy or diseased based on extracted features.It begins by initializing an SVM classifier using svm = SVC(). This creates an instance of the SVM classifier with default hyperparameter settings. The SVM classifier will be trained to classify the paddy crops.

To improve the SVM classifier's performance, the code performs hyperparameter tuning using the GridSearchCV class. It specifies a parameter grid, param_grid, which contains different values of the C and gamma hyperparameters for the SVM classifier. GridSearchCV performs an exhaustive search over the parameter grid and selects the combination of hyperparameters that yields the best performance based on cross-validation.

Once the hyperparameters are tuned, the code trains the SVM model using the best estimator obtained from the hyperparameter tuning process. The flattened training features and corresponding labels are used to train the SVM model with svm_model.fit(train_features_flat, train_labels).

After training the SVM model, the code predicts the labels for the test set using predicted_labels = svm_model.predict(test_features_flat). The SVM model utilizes the learned decision boundary to classify the test set samples as either healthy or diseased paddy crops.

The code evaluates the performance of the SVM model by calculating the accuracy of the predictions using accuracy_score(test_labels, predicted_labels). It also generates a classification report using classification_report to provide metrics such as precision, recall, and F1-score, giving insights into the model's performance across different classes.

Finally, the code visualizes the confusion matrix using confusion_matrix(test_labels, predicted_labels) and matplotlib. The confusion matrix provides a summary of the model's predictions, indicating the number of true and false predictions for each class. The visualization helps in understanding the model's performance and the types of errors it makes in classifying the paddy crop diseases.

In summary, the code demonstrates the use of SVMs for paddy crop disease detection by tuning the hyperparameters, training the model, making predictions, evaluating the performance, and visualizing the results.

- Reason for Use: SVM is a powerful and versatile algorithm for classification tasks, especially when dealing with high-dimensional feature spaces.

- Usefulness: SVM is effective in separating different classes in feature space by finding an optimal hyperplane that maximizes the margin between classes. It can handle complex decision boundaries and works well with limited training data. SVM is used in the code to train a model based on the extracted features and their corresponding labels.

2) Decision Tree :

Decision Tree classifier, a machine learning algorithm capable of performing classification tasks. The Decision Tree algorithm constructs a tree-like model where each internal node represents a feature or attribute, and each leaf node represents a class label.

First, the code initializes a Decision Tree classifier using tree = DecisionTreeClassifier(). This creates an instance of the DecisionTreeClassifier class with default hyperparameter settings.

Next, the code trains the Decision Tree model using the training features and corresponding labels. The flattened training features and labels are used to fit the Decision Tree model with tree.fit(train_features_flat, train_labels). The model learns the relationships between the features and the class labels, creating a decision tree structure.

The code then uses the trained Decision Tree model to make predictions on the test set using predicted_labels = tree.predict(test_features_flat). The Decision Tree model traverses the learned tree structure, evaluating the features of the test samples to assign class labels.

To evaluate the performance of the Decision Tree model, the code calculates the accuracy score using accuracy_d = tree.score(test_features_flat, test_labels). The accuracy score measures the proportion of correctly predicted labels in the test set.

Additionally, the code generates a confusion matrix using confusion_matrix(test_labels, predicted_labels). The confusion matrix provides a summary of the model's predictions, showing the counts of true and false predictions for each class. The matrix is then visualized as a heatmap using matplotlib.

In summary, the Decision Tree algorithm in the code plays a key role in training a model for paddy crop disease detection. It learns the patterns and relationships in the training data to create a decision tree structure. This structure is then used to make predictions on the test set, and the accuracy and confusion matrix help evaluate the model's performance. The Decision Tree algorithm

provides interpretability as the resulting tree structure can be examined to gain insights into the decision-making process.

- Reason for Use: Decision trees are interpretable and intuitive models that make decisions based on a series of if-else conditions on feature values.

- Usefulness: Decision trees can capture complex relationships between features and labels. They are useful for both classification and regression tasks. In the code, a decision tree classifier is trained using the extracted features and labels. Decision trees are relatively fast to train and provide insights into feature importance.

3) K-Nearest Neighbors (KNN) :

KNN initially imports the KNeighborsClassifier class from scikit-learn, which represents the KNN classifier. Reshapes the training and testing feature vectors to flatten them into a 1D array. Then Initializes a KNN classifier with the parameter n_neighbors=5, indicating that the algorithm should consider the five nearest neighbors to make predictions.

Then trains the KNN model using the flattened training features and corresponding labels. The model learns the patterns and relationships in the training data by storing the feature-label associations.It uses the trained KNN model to predict the labels for the test set based on the nearest neighbors. The algorithm finds the k nearest neighbors of each test sample by calculating the distances between the test samples and the training samples. It assigns the class label that is most prevalent among those k neighbors to each test sample.

Calculates the accuracy of the KNN model by comparing the predicted labels with the true labels of the test set. The accuracy is the proportion of correctly predicted labels. Computes the confusion matrix, which summarizes the performance of the KNN model by showing the counts of true positive, true negative, false positive, and false negative predictions for each class. The confusion matrix helps evaluate the model's ability to correctly classify different classes.

In summary, the KNN algorithm in the provided code trains a model based on the K-nearest neighbors of the training samples. It then uses this model to make predictions on the test set and evaluates its performance using accuracy and the confusion matrix. The KNN algorithm is a simple yet effective method for classification tasks, relying on the similarities between samples to make predictions.

- Reason for Use: KNN is a simple and non-parametric algorithm that classifies data points based on the majority vote of their nearest neighbors.

- Usefulness: KNN is effective when the decision boundary is not well-defined or linear. It can handle multi-class classification and adapt to different data distributions. In the code, KNN is trained using the extracted features and labels to classify the paddy crop leaf diseases based on their nearest neighbors in the feature space.

4) Random Forest :

It imports the RandomForestClassifier class from scikit-learn, which represents the Random Forest classifier. Reshapes the training and testing feature vectors to flatten them into a 1D array. Then initializes a Random Forest classifier with the parameter n_estimators=100, indicating the number of decision trees to be used in the ensemble. The random_state=42 ensures reproducibility of results.

It trains the Random Forest model using the flattened training features and corresponding labels. The model builds multiple decision trees, each trained on a random subset of the training data. It uses the trained Random Forest model to predict the labels for the test set. Each decision tree in the ensemble independently predicts a label, and the final prediction is determined by majority voting or averaging, depending on the task (classification or regression).

Calculates the accuracy of the Random Forest model by comparing the predicted labels with the true labels of the test set. The accuracy is the proportion of correctly predicted labels. Computes the confusion matrix, which summarizes the performance of the Random Forest model by showing the counts of true positive, true negative, false positive, and false negative predictions for each class. The confusion matrix helps evaluate the model's ability to correctly classify different classes.

In summary, the Random Forest algorithm in the provided code builds an ensemble of decision trees using the Random Forest classifier. It trains multiple trees on random subsets of the training data and combines their predictions to make final predictions. Random Forest is a powerful machine learning algorithm known for its ability to handle complex datasets and mitigate overfitting.

- Reason for Use: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions.

- Usefulness: Random Forest can handle high-dimensional feature spaces, capture complex interactions between features, and reduce overfitting compared to individual decision trees. It is

robust to noisy data and provides good generalization. In the code, a Random Forest classifier is trained using the extracted features and labels for disease classification.

The choice of these algorithms provides a diverse set of models with different characteristics. By using multiple algorithms, the code aims to compare their performance and identify the most suitable model for classifying paddy crop leaf diseases based on the given dataset. Each algorithm has its strengths and weaknesses, and their performance can vary depending on the specific characteristics of the dataset.

Overall, by training multiple models, the code explores different approaches to find the model that provides the best accuracy and generalization for classifying the crop leaf diseases.

Model Evaluation

Model evaluation is the process of assessing the performance and effectiveness of a trained machine learning model using various metrics and techniques. It helps determine how well the model generalizes to unseen data and how accurately it predicts the target labels.

Model evaluation is performed to compare the performance of the four algorithms (SVM, Decision Tree, KNN, and Random Forest) in classifying paddy crop leaf diseases. The accuracy of each algorithm is calculated and compared to determine which algorithm performs the best.

SVM obtained the highest accuracy among the four algorithms. This indicates that SVM achieved the highest percentage of correct predictions on the test set compared to the other algorithms. This result suggests that SVM is the most effective algorithm for classifying paddy crop leaf diseases in the given dataset.

The usefulness of this finding is that it provides insights into the performance of different algorithms for the specific task of crop leaf disease classification. By identifying the algorithm with the highest accuracy, we can choose that algorithm for deployment in real-world applications. Additionally, knowing the accuracy of each algorithm allows us to make informed decisions about the choice of algorithm based on the desired level of accuracy and other evaluation metrics.

• Area Detection

The image performs image processing and analysis to calculate the affected area of a disease in a rice leaf image.

1. Import necessary libraries:
    - ‘cv2‘: OpenCV library for image processing.
    - ‘numpy‘: Library for numerical operations.
    - ‘cv2_imshow‘: A utility function from Google Colab for displaying images.

2. Load the leaf image:
    - The ‘cv2.imread()‘ function is used to read the image from the specified file path.

3. Convert the image to HSV color space:

• The ‘cv2.cvtColor()‘ function is used to convert the image from the BGR color space (default in OpenCV) to the HSV color space. This conversion allows for better color-based segmentation.

4. Define the lower and upper color thresholds for disease-affected areas:

• The lower and upper color values define a range of HSV values that represent the color characteristics of the disease-affected areas in the image. These values need to be adjusted based on the specific characteristics of the disease being detected.

5. Threshold the image to get disease-affected areas:

• The ‘cv2.inRange()‘ function is used to create a binary mask, where white pixels represent the disease-affected areas and black pixels represent the rest of the image, based on the defined color thresholds.

6. Apply morphological operations to improve the mask:

• Morphological operations, such as closing, are applied to the binary mask using the ‘cv2.morphologyEx()‘ function to remove noise and fill in gaps in the detected areas.

7. Find contours in the mask:

• The ‘cv2.findContours()‘ function is used to find the contours (boundaries) of the disease-affected areas in the binary mask.

8. Draw contours on the original image:

• The ‘cv2.drawContours()‘ function is used to draw the detected contours on the original image. In this code, blue contours are drawn around the disease-affected areas.

9. Calculate the total leaf area:

• The total leaf area is calculated by multiplying the height (number of rows) and width (number of columns) of the original image.

10. Initialize a variable to store the total affected area and iterate over the contours:

• A variable ‘affected_area‘ is initialized to 0 to store the cumulative affected area.

• The code iterates over each contour and calculates the area using the ‘cv2.contourArea()‘ function. The length and width of the bounding rectangle around each contour are also calculated using ‘cv2.boundingRect()‘.

11. Calculate the length and width in millimeters:

• Assuming you have the pixel resolution or scale factor in mm/pixel, the code multiplies the length and width of the bounding rectangle by the pixel resolution to calculate the length and width in millimeters.

12. Update the total affected area and calculate the affected area in mm$\hat{2}$:

• The total affected area is updated by adding the area of each contour.

• Affected area in mm$\hat{2}$ is calculated by multiplying the length and width in millimeters.

13. Print the affected area:

• The total affected area in mm$\hat{2}$, total affected area in pixels, and affected area percentage are printed.

14. Calculate the total disease area:

• The code calculates the total disease area by multiplying the affected area percentage with the total area of the leaf.

15. Display the image with disease-affected areas highlighted:

• The ‘cv2_imshow()‘ function is used to display the processed image with the disease-affected areas highlighted.

It provides information about the affected area in terms of pixels, millimeters, and percentage, which can be useful for analyzing the severity of the disease and monitoring its progression.

5. IMPLEMENTATION & RESULTS

5.1 CODE

```
import os
import cv2
import numpy as np

dataset_path = "/content/drive/MyDrive/rice_leaf_diseases_1"

images = []
labels = []

class_to_disease = {
    "Bacterial leaf blight": "Bacterial Leaf Blight",
    "Brown spot": "Brown Spot",
    "Leaf smut": "Leaf Smut",
    "Healthy": "Healthy"
}

for class_folder in os.listdir(dataset_path):
    class_path = os.path.join(dataset_path, class_folder)
    if os.path.isdir(class_path):
        for image_file in os.listdir(class_path):
            image_path = os.path.join(class_path, image_file)
            image = cv2.imread(image_path)
            if image is not None:
                image = cv2.resize(image, (224, 224))
                images.append(image)
                labels.append(class_to_disease[class_folder])

images = np.array(images)
labels = np.array(labels)

disease_names = np.unique(labels)

for i, disease_name in enumerate(disease_names):
    print("Disease Name:", disease_name, " - Label:", i)
```

```
Disease Name: Bacterial Leaf Blight  - Label: 0
Disease Name: Brown Spot  - Label: 1
Disease Name: Healthy  - Label: 2
Disease Name: Leaf Smut  - Label: 3
```

Figure 5.1.1: Dataset Loading and Preprocessing

This module loads and preprocesses the dataset. It iterates over each class folder, reads the images, resizes them, and appends them to the images list. The corresponding class folder names are used as labels and appended to the labels list. Finally, the lists are converted to numpy arrays.

```
[ ] class_names = np.unique(labels)
    print("Class names:", class_names)

    Class names: ['Bacterial Leaf Blight' 'Brown Spot' 'Healthy' 'Leaf Smut']
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

# Normalize pixel values to [0, 1]
images = images.astype('float32') / 255.0

label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)
```

Figure 5.1.2: Label Encoding and Train-Test Split

This module performs label encoding on the categorical labels using LabelEncoder. The pixel values of the images are normalized to the range [0, 1]. The dataset is shuffled using shuffle for randomization. Then, it is split into training and testing sets using train_test_split from sklearn.

```python
import matplotlib.pyplot as plt

label_to_class = {label_idx: class_name for label_idx, class_name in enumerate(label_encoder.classes_)}

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(8, 8))
axes = axes.ravel()

for i in range(9):
    rand_index = np.random.randint(0, len(images))
    axes[i].imshow(images[rand_index])
    label_idx = labels[rand_index]
    class_name = label_to_class[label_idx]
    axes[i].set_title(f"{class_name}-{label_idx}")
    axes[i].axis('off')

plt.tight_layout()
plt.show()
```

Figure 5.1.3: Visualization

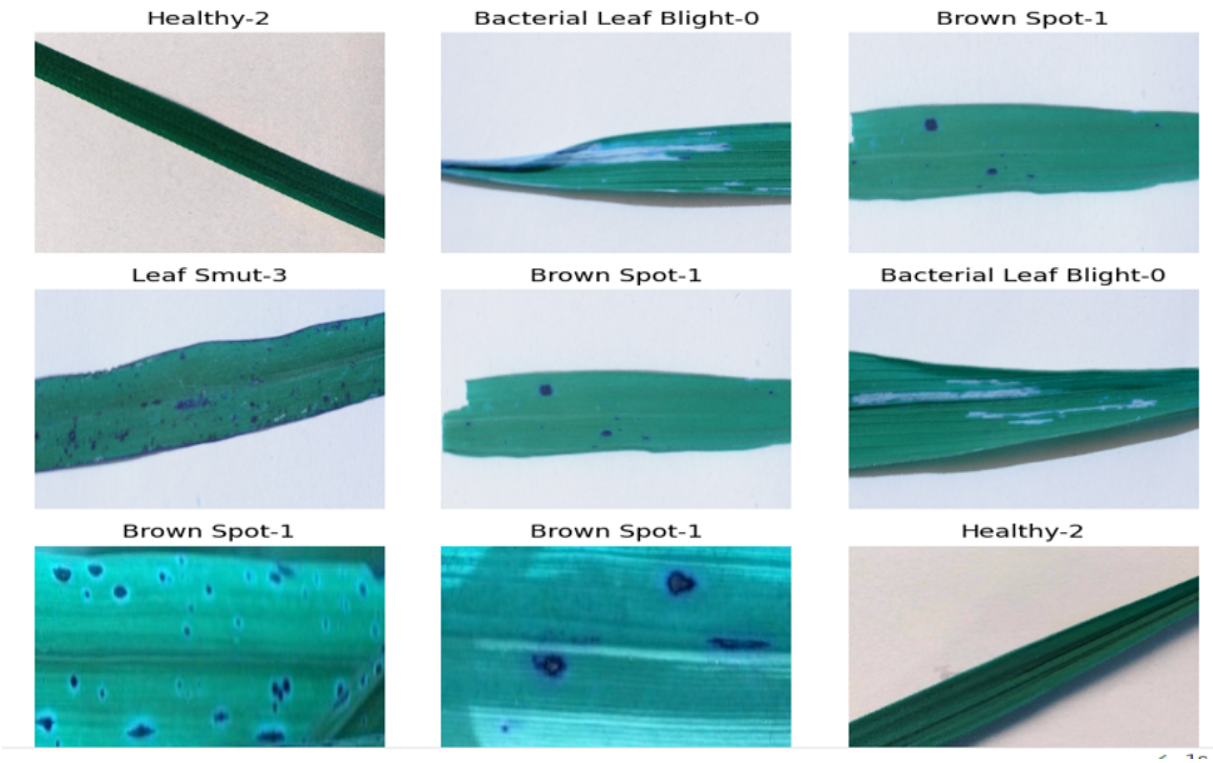


Figure 5.1.4 visualization output

Figure 5.1.5: Feature Extraction using VGG16

This module uses the pre-trained VGG16 model to extract features from the images. The VGG16 model is loaded without the fully connected layers. The predict method is used to extract features from both the training and testing images.

Model Training using SVM

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Flatten the feature vectors
train_features_flat = train_features.reshape(train_features.shape[0], -1)

# Initialize an SVM classifier
svm = SVC()

param_grid = {'C': [1, 10, 100], 'gamma': [0.1, 0.01, 0.001]}


grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(train_features_flat, train_labels)

# Get the best SVM model
svm_model = grid_search.best_estimator_
# Train the SVM model
svm_model.fit(train_features_flat, train_labels)
```

```
SVC
SVC(C=1, gamma=0.001)
```

Figure 5.1.6 Training using SVM Classifier

This module trains an SVM classifier on the extracted features. The feature vectors are reshaped to be flattened. An SVM classifier is initialized, and a parameter grid is defined for hyperparameter tuning using GridSearchCV. Grid search with 5-fold cross-validation is performed to find the best hyperparameters. The best SVM model is obtained, and it is trained on the training features and labels.

Model Evaluation using SVM

```
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix

# Reshape the test features
test_features_flat = test_features.reshape(test_features.shape[0], -1)

# Predict labels for the test set
predicted_labels = svm_model.predict(test_features_flat)

# Calculate accuracy and other evaluation metrics
accuracy = accuracy_score(test_labels, predicted_labels)
report = classification_report(test_labels, predicted_labels, target_names=label_encoder.classes_)

print("Accuracy:", accuracy)
print(confusion_matrix(test_labels,predicted_labels))

cm = confusion_matrix(test_labels, predicted_labels)
classes = np.unique(test_labels)

# Plot the confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Greens)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
# Label the matrix with the counts
thresh = cm.max() / 2.0
for i, j in np.ndindex(cm.shape):
    plt.text(j, i, cm[i, j], ha='center', va='center', color='white' if cm[i, j] > thresh else 'black')

# Add axis labels
plt.xlabel('Predicted Label')
plt.ylabel('Test Label')

# Show the plot
plt.show()
```

Figure 5.1.7 Evaluation of SVM Classifier

This module evaluates the SVM classifier on the testing set. It reshapes the testing features, predicts labels using the best SVM model, and calculates accuracy, classification report, and confusion matrix. It also demonstrates the prediction of a sample image using the trained SVM model.

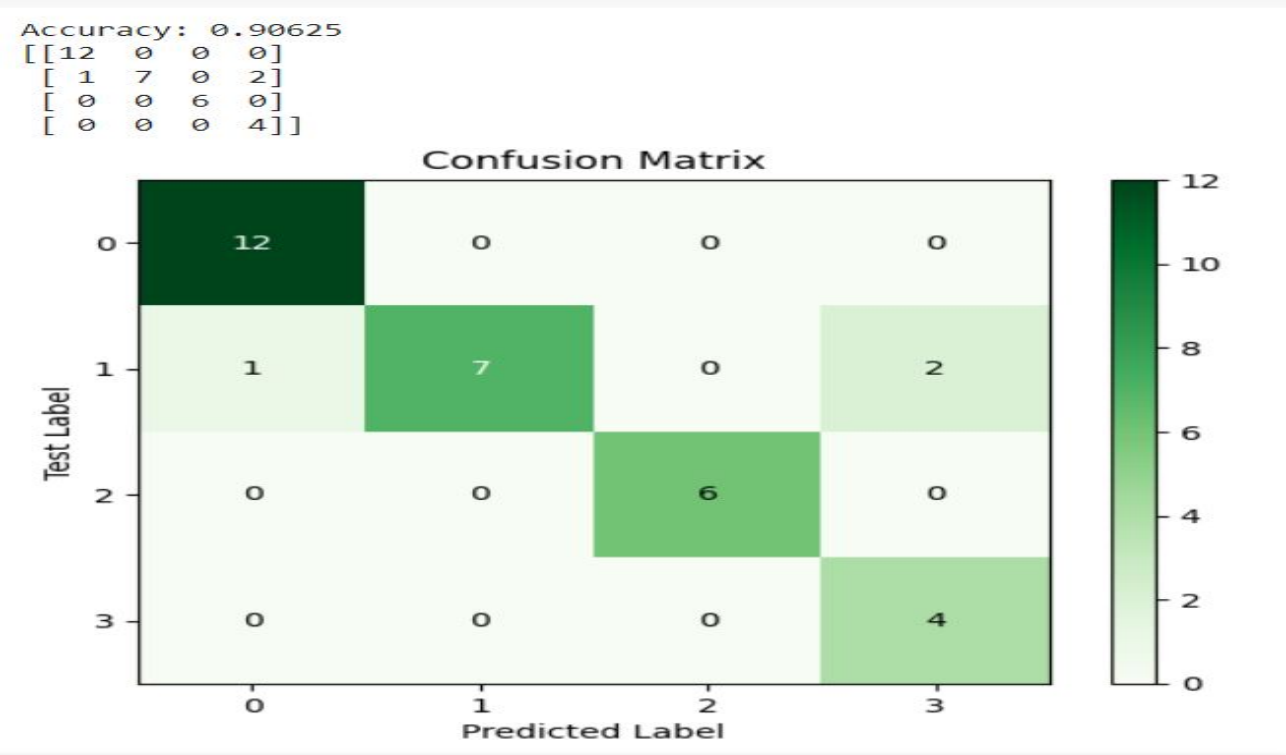


Figure 5.1.8: Accuracy and Confusion Matrix of SVM Classifier

Figure 5.1.9 Prediction of Bacterial Leaf Blight Disease using SVM Classifier

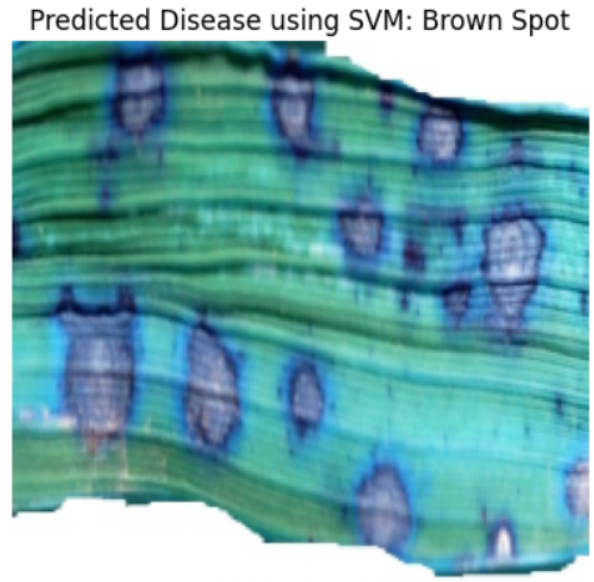
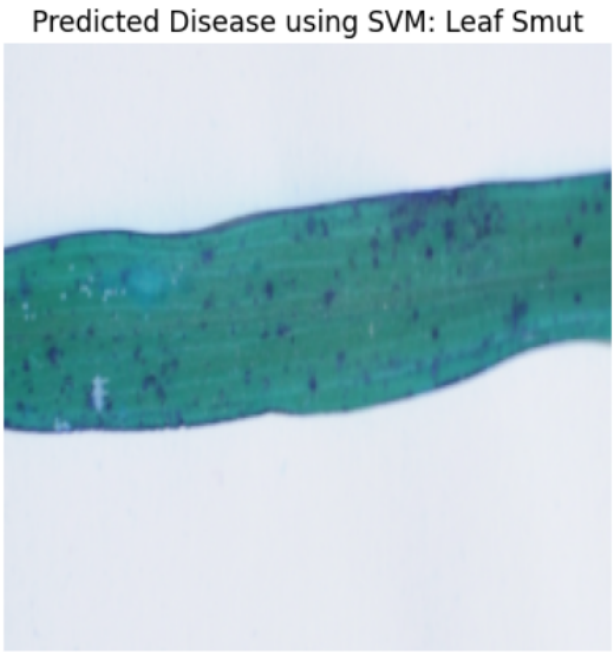


Figure 5.1.10 Prediction of Brownspot Disease using SVM Classifier

Figure 5.1.11: Prediction of Healthy Leaf with SVM Classifier




```
Remedies :
Treat paddy seeds with recommended fungicides like carbendazim or thiophanate-methyl before planting.
Apply fungicides during the early stages of the crop as a preventive measure against leaf smut.
```

Figure 5.1.12 Prediction of Leaf Smut Disease using SVM Classifier

Model Training using Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

train_features_flat = train_features.reshape(train_features.shape[0], -1)
test_features_flat = test_features.reshape(test_features.shape[0], -1)

tree = DecisionTreeClassifier()

tree.fit(train_features_flat, train_labels)

Dt_model = tree.fit(train_features_flat, train_labels)

accuracy_d= tree.score(test_features_flat, test_labels)
print("Accuracy:", accuracy_d)
```

Figure 5.1.13 Model Training using Decision Tree Classifier

This module trains a decision tree classifier on the extracted features. The training and testing features are reshaped. A decision tree classifier is initialized, trained on the training features and labels, and evaluated for accuracy. It also demonstrates the prediction of a sample image using the trained decision tree model.

```
print("Accuracy:", accuracy_d)
print(confusion_matrix(test_labels,predicted_labels))
dt = confusion_matrix(test_labels, predicted_labels)
classes = np.unique(test_labels)

# Plot the confusion matrix
plt.imshow(dt, interpolation='nearest', cmap=plt.cm.Greens)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
# Label the matrix with the counts
thresh = dt.max() / 2.0
for i, j in np.ndindex(dt.shape):
    plt.text(j, i, dt[i, j], ha='center', va='center', color='white' if dt[i, j] > thresh else 'black')

# Add axis labels
plt.xlabel('Predicted Label')
plt.ylabel('Test Label')

# Show the plot
plt.show()
```

Figure 5.1.14 Model Evaluation using Decision Tree Classifier

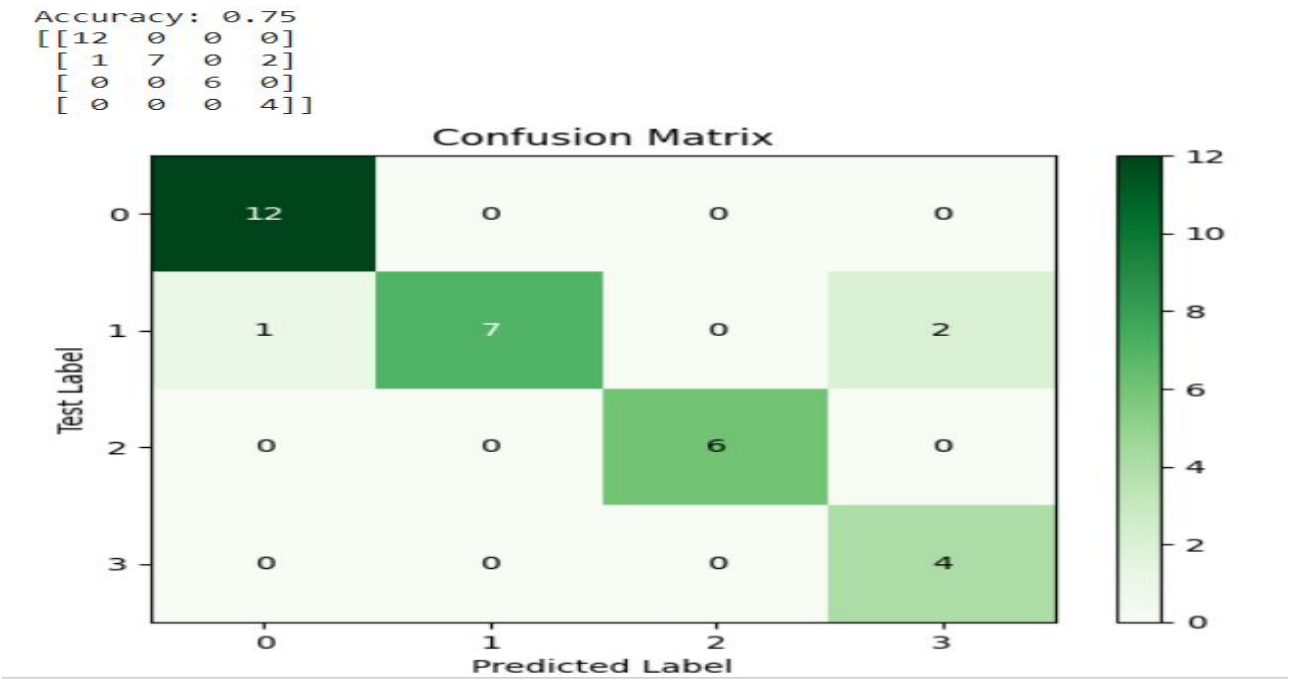


Figure 5.1.15: Accuracy and Confusion Matrix of Decision Tree Classifier


```
Remedies :
Use disease-free seeds and rotate paddy with non-host crops.
Apply copper-based sprays (such as Bordeaux mixture or copper hydroxide) or bactericides like streptomycin sulfate.
```

Figure 5.1.16 Prediction of Bacterial Leaf Blight Disease with Decision Tree Classifier

Figure 5.1.17 Prediction of Brownspot Disease with Decision Tree Classifier

Figure 5.1.18: Prediction of Healthy Image with Decision Tree Classifier

Figure 5.1.19 Prediction of Leaf Smut Disease with Decision Tree Classifier

Model Training using KNN

```
[ ] from sklearn.neighbors import KNeighborsClassifier

    train_features_flat = train_features.reshape(train_features.shape[0], -1)
    test_features_flat = test_features.reshape(test_features.shape[0], -1)

    knn = KNeighborsClassifier(n_neighbors=5)

    knn.fit(train_features_flat, train_labels)

    knn_model = knn.fit(train_features_flat, train_labels)

    accuracy_knn = knn.score(test_features_flat, test_labels)
```

Figure 5.1.20 Model Training using KNN Classifier

This module trains a K-Nearest Neighbors (KNN) classifier on the extracted features. A KNN classifier with 5 neighbors is initialized, trained on the training features and labels, and evaluated for accuracy. It also demonstrates the prediction of a sample image using the trained KNN model.

```
print("Accuracy:", accuracy_knn)
print(confusion_matrix(test_labels,predicted_labels))
kcm = confusion_matrix(test_labels, predicted_labels)
classes = np.unique(test_labels)

# Plot the confusion matrix
plt.imshow(kcm, interpolation='nearest', cmap=plt.cm.Greens)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
# Label the matrix with the counts
thresh = kcm.max() / 2.0
for i, j in np.ndindex(kcm.shape):
    plt.text(j, i, kcm[i, j], ha='center', va='center', color='white' if kcm[i, j] > thresh else 'black')

# Add axis labels
plt.xlabel('Predicted Label')
```

Figure 5.1.21 Model Evaluation using KNN Classifier

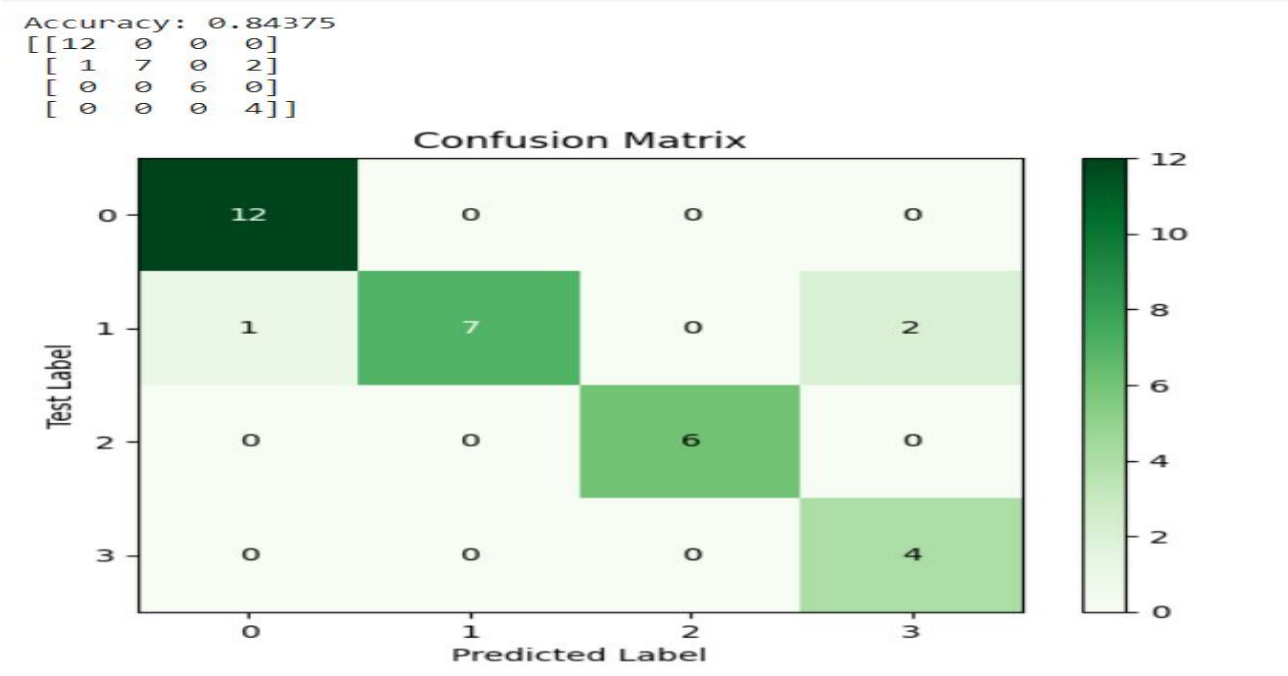


Figure 5.1.22: Accuracy and Confusion Matrix of KNN Classifier

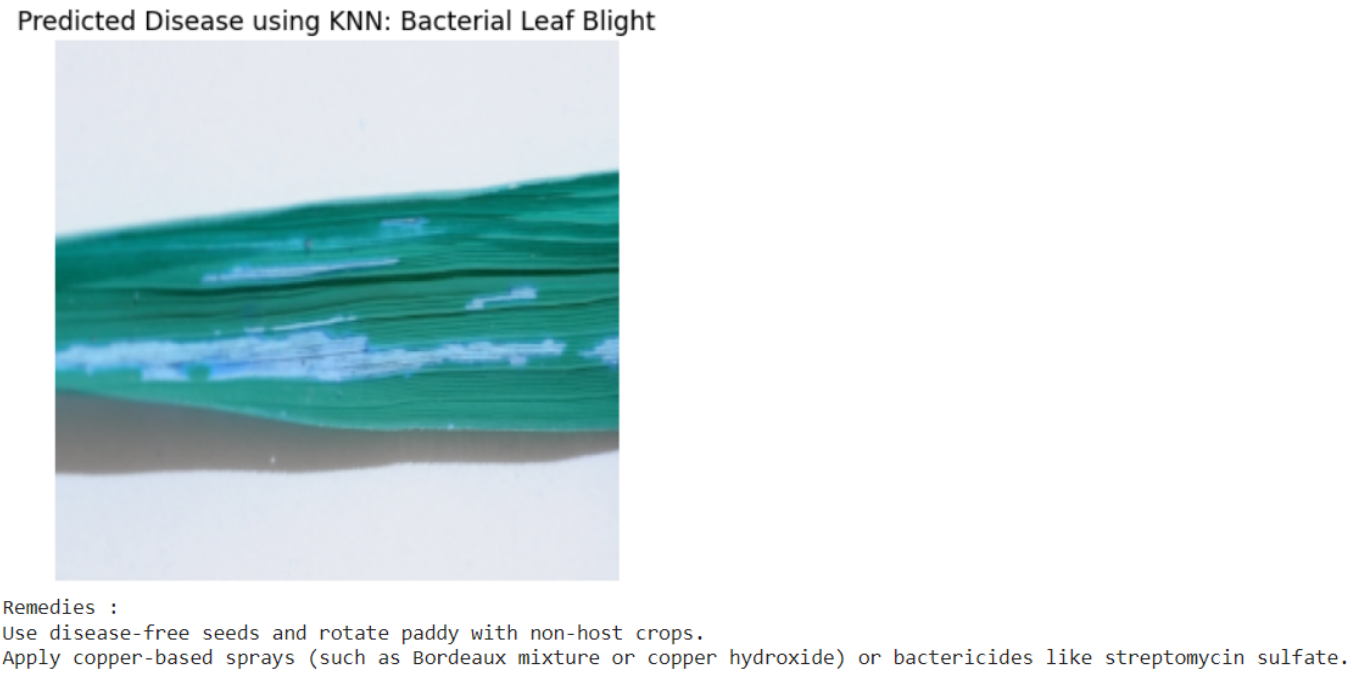


Figure 5.1.23: Prediction of Bacterial Leaf Blight Disease with KNN Classifier

Figure 5.1.24 Prediction of Brownspot Disease with KNN Classifier

Figure 5.1.25 Prediction of Healthy Image with KNN Classifier

Figure 5.1.26 Prediction of Leaf Smut Disease with KNN Classifier

Model Training using Random Forest

```
from sklearn.ensemble import RandomForestClassifier

train_features_flat = train_features.reshape(train_features.shape[0], -1)
test_features_flat = test_features.reshape(test_features.shape[0], -1)

rf = RandomForestClassifier(n_estimators=100, random_state=42)

rf.fit(train_features_flat, train_labels)
rf_model = rf.fit(train_features_flat, train_labels)

accuracy_r = rf.score(test_features_flat, test_labels)
print("Accuracy:", accuracy_r)
```

Figure 5.1.27 Model Training using Random Forest Classifier

```
print("Accuracy:", accuracy_r)
print(confusion_matrix(test_labels,predicted_labels))

rdf = confusion_matrix(test_labels, predicted_labels)
classes = np.unique(test_labels)

# Plot the confusion matrix
plt.imshow(rdf, interpolation='nearest', cmap=plt.cm.Greens)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
# Label the matrix with the counts
thresh = rdf.max() / 2.0
for i, j in np.ndindex(kcm.shape):
    plt.text(j, i, rdf[i, j], ha='center', va='center', color='white' if rdf[i, j] > thresh else 'black')
# Add axis labels
plt.xlabel('Predicted Label')
plt.ylabel('Test Label')
# Show the plot
plt.show()
```

Figure 5.1.28 Model Evaluation using Random Forest Classifier

```
Accuracy: 0.8125
[[12  0  0  0]
 [ 1  7  0  2]
 [ 0  0  6  0]
 [ 0  0  0  4]]
```

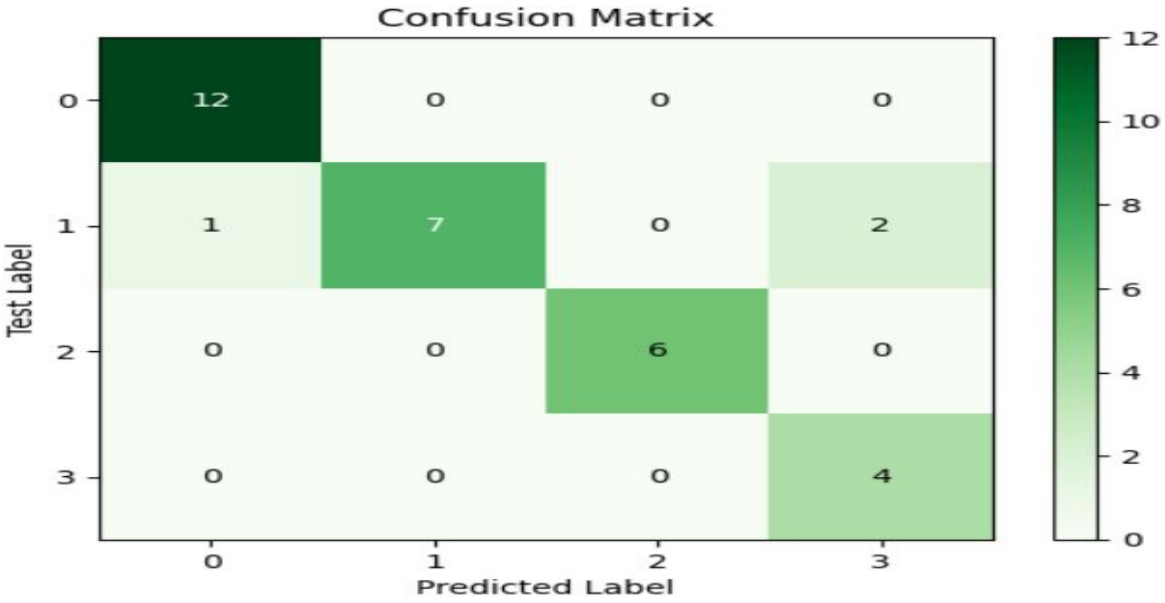
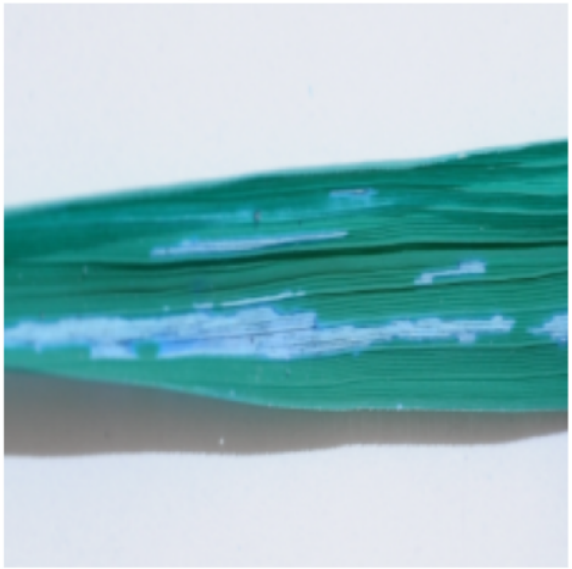


Figure 5.1.29 Accuracy and Confusion Matrix of Random Forest Classifier

Predicted Disease using Random Forest: Bacterial Leaf Blight



```
Remedies :
Use disease-free seeds and rotate paddy with non-host crops.
Apply copper-based sprays (such as Bordeaux mixture or copper hydroxide) or bactericides like streptomycin sulfate.
```

Figure 5.1.30 Prediction of Bacterial Leaf Blight Disease with Random Forest Classifier

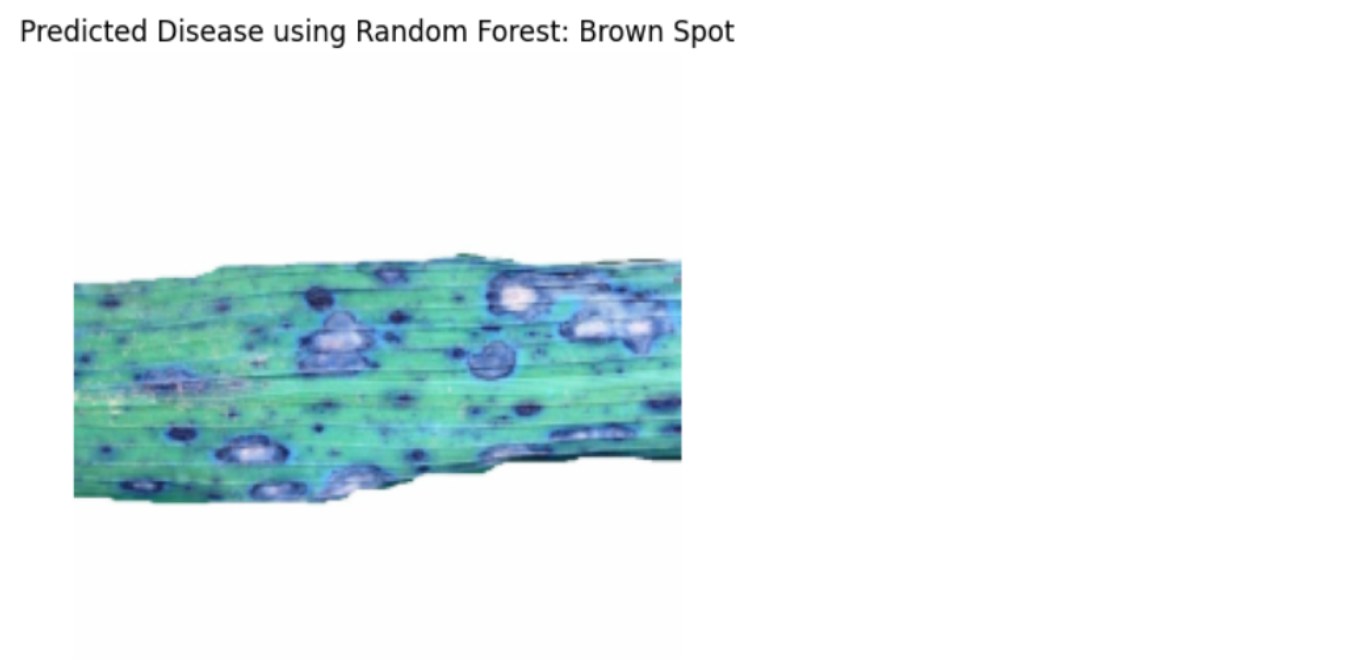


Figure 5.1.31 Prediction of Brownspot Disease with Random Forest Classifier

Figure 5.1.32 Prediction of Healthy Image with Random Forest Classifier

Predicted Disease using Random Forest: Leaf Smut

```
Remedies :
Treat paddy seeds with recommended fungicides like carbendazim or thiophanate-methyl before planting.
Apply fungicides during the early stages of the crop as a preventive measure against leaf smut.
```

Figure 5.1.33 Prediction of Leaf Smut Disease with Random Forest Classifier

```python
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

# Load the leaf image
image = cv2.imread('/content/drive/MyDrive/rice_leaf_diseases_1/Bacterial leaf blight/DSC_0372.JPG')

# Convert the image to HSV color space
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Defining the lower and upper color thresholds for disease-affected areas
lower_color = np.array([20, 50, 50])  # Adjusting these values according to  specific disease characteristics
upper_color = np.array([40, 255, 255])  # Adjusting these values according to specific disease characteristics

# Threshold the image to get disease-affected areas
mask = cv2.inRange(hsv, lower_color, upper_color)

# Apply a morphological operation to improve the mask
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

#contour_image = np.zeros_like(image)
# Find contours in the mask
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Draw contours on the original image
cv2.drawContours(image, contours, -1, (255, 0, 0), 6)  # Draw blue contours around disease-affected areas
```

```python
cv2.drawContours(image, contours, -1, (255, 0, 0), 6)  # Draw blue contours around disease-affected areas

# Calculate the total leaf area
total_area = image.shape[0] * image.shape[1]
#print(total_area)
# Initialize a variable to store the total affected area
affected_area = 0

# Iterate over the contours
for contour in contours:
    # Calculate the area of each contour
    area = cv2.contourArea(contour)

    x, y, w, h = cv2.boundingRect(contour)
    length = h
    width = w

    # Assuming you have the pixel resolution or scale factor in mm/pixel
    pixel_resolution_mm = 0.1  # Pixel resolution in mm/pixel

    # Calculate the length and width in millimeters
    length_mm = length * pixel_resolution_mm
    width_mm = width * pixel_resolution_mm

    # Update the total affected area
    affected_area += area
```

```
    affected_area += area

   # Calculate the affected area in mm^2
    affected_area_mm2 = length_mm * width_mm

# Print the length and width of the affected area
#print("Length:  {} mm".format(length_mm))
#print("Width: {} mm" .format(width_mm))

# Print the affected area
print("Total affected Area in mm^2 : {} mm^2".format(affected_area_mm2))

# Print the total affected area
print("Total affected area in pixels :", affected_area, "pixels")

# Calculate the affected area percentage
affected_area_percentage = (affected_area / total_area) * 100

# Print the affected area percentage
print("Total affected area in percentage:", affected_area_percentage, "%")

Disease_Area  = (affected_area_percentage / 100) * total_area

# Display the image with disease-affected areas highlighted
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 5.1.34 Area Detection of Diseased Image

This code segment takes an image of a leaf and detects disease-affected areas in the image. It converts the image to the HSV color space and defines color thresholds to identify the affected areas. It then applies image thresholding and morphological operations to improve the mask. Contours are extracted from the mask, and these contours represent the boundaries of the disease-affected areas. The code calculates the area, length, and width of each contour. Assuming a known pixel resolution, it converts the length and width to millimeters. The affected area is calculated in millimeters squared. Finally, the code displays the original image with the disease-affected areas highlighted and prints the affected area in millimeters squared, pixels, and percentage of the total area.

Figure 5.1.35: AREA INTENSITY DETECTION FOR BACTERIAL LEAF BLIGHT DISEASE IMAGE

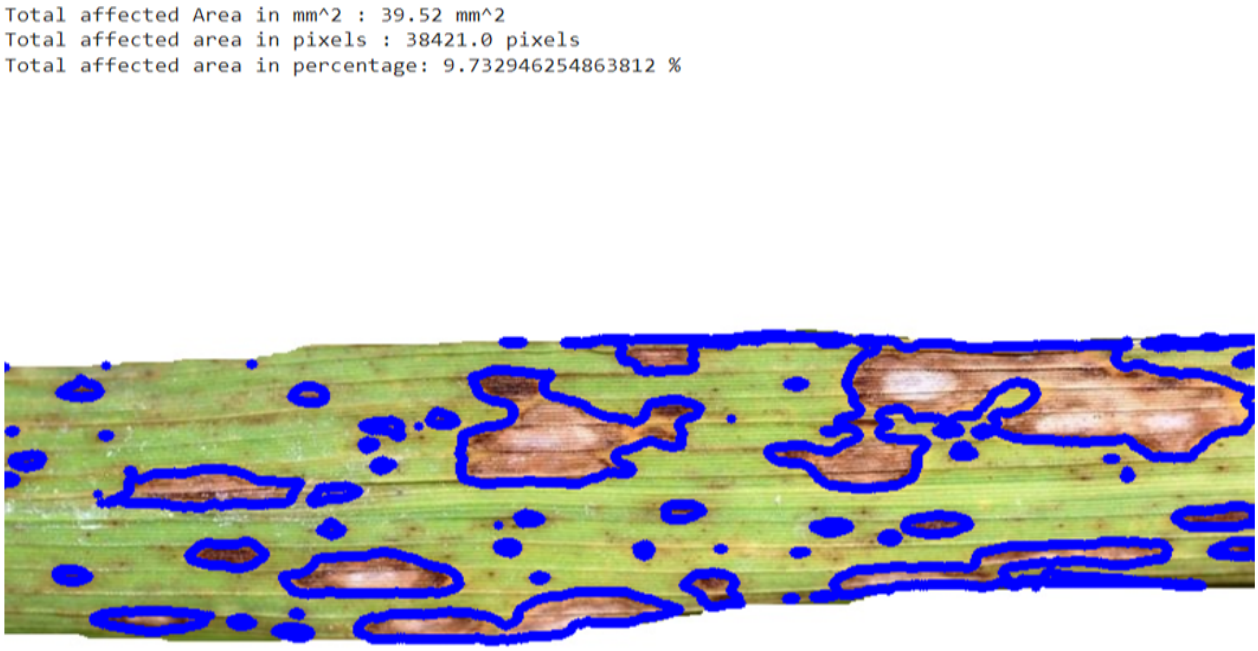


Figure 5.1.36: AREA INTENSITY DETECTION FOR BROWN SPOT DISEASE IMAGE

Figure 5.1.37: AREA INTENSITY DETECTION FOR HEALTHY IMAGE

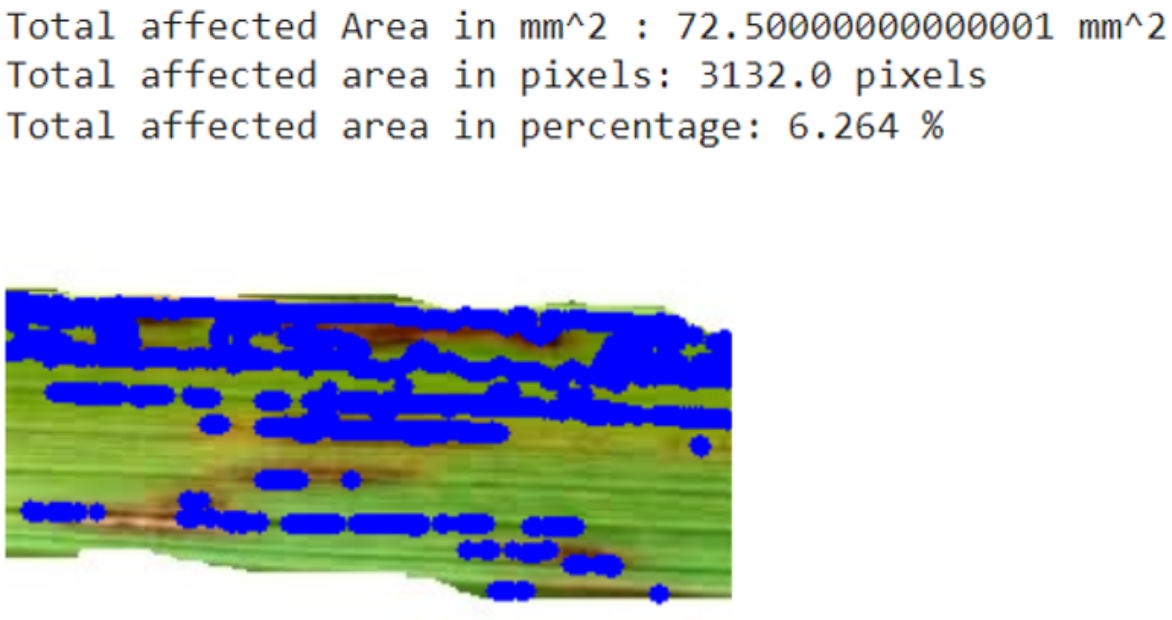


Figure 5.1.38: AREA INTENSITY DETECTION FOR LEAF SMUT DISEASE IMAGE

6. CONCLUSION AND FUTURE SCOPE

As farmers are facing issues with disease identification, level of the disease and unable to find effective remedies to cure the diseases. To solve the issues, a machine learning model is developed that detects the disease and the calculates the affected area. For better accuracy, the model is trained with different algorithms such as Support Vector Machine(SVM), K-Nearest Neigbor(KNN), Random Forest, Decision Tree. Among all these SVM gives highest accuracy. Based on the disease identified, remedies are suggested. The remedies provide proper information regarding the pesticide or insecticide to be used in order to cure the disease.

In Future, there is scope for developing models capable of identifying multiple diseases simultaneously. This would provide a comprehensive assessment of crop health and enable farmers to implement holistic disease management strategies.

REFERENCES

[1] Mrs. Shruti U, Dr. Nagaveni V, Dr. Raghavendra B K, “A review on machine learning classification techniques for plant disease detection” International Conference on Advanced Computing & Communication Systems (ICACCS), 2021.

[2] Md. Ashiqul Islam, Md. Nymur Rahman Shuvo, Muhammad Shamsojjaman Shazid Hasan, Md. Shahadat Hossain, Tania Khatun, “Automated Convolutional Neural Network Based Approach for Paddy Leaf Disease Detection” , IJACSA, 2021.

[3] Tejas Tawde, Lobhas Verekar, Shailendra Aswale, Kunal Deshmukh, Ajay Reddy, “Rice Plant Disease Detection and Classification Techniques : A Survey”, IJERT, 2021.

[4] V. Vanitha “Rice disease detection using machine learning”, International Journal of Recent Technology and Engineering, IJRTE, Feb 2020.

[5] Nargis Parven, Muhammad Rashiduzzaman, Nasrin Sultana, Md. Touhidur Rahman, Md.Ismail Jabiullah, “Detection and Recognition of Paddy Plant Leaf Diseases using Machine Learning Technique”, IJIEE, 2020.

[6] Md. Jahid Hasan, Shamim Mahbub, Md. Abu Nasim, Md. Shahin Alom “Rice disease identification and classification by integrating support vector machine with deep convolutional neural network”, IJRTE, 2019.

[7] Anuradha Badge, “Crop disease detection using Machine learning: Indian Agriculture”, IJRTE, 2019.

[8] Mr M Sivakumar, S Suriya, S Santhana Hari, Dr P Renuga, S Karthikeyan “Detection of plant disease by leaf image using convolutional neural network”, IJRTE, 2019.

[9] Priyanka B Raj, Dr Neha Mangal, Pooja R ,Soumya G Hegde “Paddy leaf disease detection using image processing and machine learning”, IJRTE, 2019.

[10] Tenzin Chokey, Sarika Jain “Quality assessment of crops using machine learning technique”, IJRTE, 2019.

[11] Mr Ramachnadra Hebbar, Mr.P.V Vinod, Shima Ramesh, Niveditha.M, Pooja R, Shashank.N, Prasad Bhat.N, “Plant disease detection using machine learning”, IEEE, 2018.

[12] Muhammad Shoaib, Babar Shah, Shaker EI-Sappagh, Akhtar Ali, Asad Ullah, “An advanced deep learning models-based plant disease detection: A review of recent research”, IJERT,2018.

[13] Amritha Haridasan, Jeena Thomas, Ebin Deni Raj, “Deep learning system for paddy plant disease detection and classification”, IJIEE, 2018.

[14] Anjali K, Mrs Divya unni, Arya M.S, “Detection of unhealthy plant leaves using image processing and genetic algorithm with arduino”, ICPSCC, 2018.

[15] Manish Sah, Taohidul Islam,Rudra Roy Choudhry,Sudipto Baral “A faster technique on rice disease detection using image processing of affected area in agro-field”, ICICCT, 2018