RICE LEAF DISEASE DETECTION USING DEEP LEARNING

A Project Report

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

Submitted by

BATTULA JAGADEESH	20BQ1A1217
ALLURI UMESH CHANDRA	20BQ1A1205
GADIYAM UDAY SANKAR REDDY	20BQ1A1255
GUNTI SATISH	20BQ1A1261

Under the Supervision of

Mr. Md. SHAKEEL AHMED

Associate Professor

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

DEPARTMENT OF INFORMATION TECHNOLOGY

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY

NAMBUR(V), PEDAKAKANI(M), GUNTUR-522508, TEL NO:0873 211803,

url: www.vvitguntur.com An Autonomous Institute Affiliated to JNTUK, Approved by AICTE New Delhi, Accredited by NBA, NAAC with 'A' Grade and ISO 9001:2008 Certified April 2024

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY:: NAMBUR

VASIREDDY VENKATADRI
INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

BONAFIDE CERTIFICATE

This is to certify that the project report **"RICE LEAF DISEASE DETECTION USING DEEP LEARNING"** is the bonafide work done by "**BATTULA JAGADEESH (20BQ1A1217), ALLURI UMESH CHANDRA (20BQ1A1205), GADIYAM UDAY SANKAR REDDY (20BQ1A1255), GUNTI SATISH (20BQ1A1261)"**, who carried out the project under my guidance during the year 2023 towards partial fulfillment of the requirements of the Degree of Bachelor of Technology in Information Technology from Vasireddy Venkatadri Institute of Technology,Nambur. The results embodied in this report have not been submitted to any other University for the award of any degree.

Signature of the Head of the Department	Signature of the Supervisor
Dr. ALLA KALAVATHI	**Md. SHAKEEL AHMED**
HEAD OF THE DEPARTMENT	**PROJECT GUIDE**
Department of Information Technology	

External Viva voce conducted on 06-04-2024

Internal Examiner **External Examiner**

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY:: NAMBUR

CERTIFICATE OF AUTHENTICATION

I solemnly declare that this project report **“RICE LEAF DISEASE DETECTION USING DEEP LEARNING”** is the bonafide work done purely by me/us, carried out under the supervision of Mr. Md. SHAKEEL AHMED, towards partial fulfillment of the requirements of the Degree of Bachelor of Technology in Information Technology from Vasireddy Venkatadri Institute of Technology, Nambur during the year 2023- 24.

It is further certified that this work has not been submitted, either in part or in full, to any other department of the Vasireddy Venkatadri Institute of Technology, or any other University, institution or elsewhere, or for publication in any form.

Signature of the Student

06-04-2024

BATTULA JAGADEESH
20BQ1A1217, 2023-24

ALLURI UMESH CHANDRA
20BQ1A1205, 2023-24

GADIYAM UDAY SANKAR REDDY
21BQ5A1255, 2023-24

GUNTI SATISH
20BQ1A1261, 2023-24

ACKNOWLEDGEMENTS

We take this opportunity to express our deepest gratitude and appreciation to all those people who made this project work easier with words of encouragement, motivation, discipline, and faith by offering different places to look to expand my ideas and helped me towards the successful completion of this project work.

First and foremost, we express our deep gratitude to **Sri. Vasireddy Vidya Sagar, Chairman,** Vasireddy Venkatadri Institute of Technology for providing necessary facilities throughout the Information Technology program.

We express our sincere thanks to **Dr. Y. Mallikarjuna Reddy, Principal,** Vasireddy Venkatadri Institute of Technology for his constant support and cooperation throughout the Information Technology program.

We express our sincere gratitude to **Dr. A. Kalavathi, Professor & HOD,** Information Technology, Vasireddy Venkatadri Institute of Technology for her constant encouragement, motivation and faith by offering different places to look to expand my ideas. We would like to express our sincere gratefulness to our guide **Mr. Md. Shakeel Ahmed, Associate Professor** for her insightful advice, motivating suggestions, invaluable guidance, help, and support in successful completion of this project and also our project co-ordinator **Mr. R. Sudha Kishore, Associate Professor** for her advice and support. We would like to take this opportunity to express our thanks to the teaching and non-teaching staff in Department of Information Technology, VVIT for their invaluable help and support.

ABSTRACT

Rice, a vital staple crop globally, confronts significant threats from various leaf diseases, adversely impacting yield and agricultural sustainability. In response, our project endeavors to develop an efficient rice leaf disease detection system using deep learning models. The study utilizes a comprehensive dataset encompassing images of rice leaves afflicted with four prevalent diseases: bacterial blight, blast, tungro, and brown spot. Leveraging diverse deep learning architectures, including Simple CNN, ResNet, Inception, and LeNet, we train and evaluate models to accurately classify these diseases. Through meticulous experimentation, we identify the model exhibiting the highest accuracy on the validation dataset. Deploying the selected model with the Streamlit library, we construct a userfriendly frontend interface facilitating seamless interaction. This interface empowers users to upload rice leaf images, enabling real-time disease prediction and identification. By integrating advanced deep learning techniques with accessible user interaction, our project offers a practical solution for farmers and agricultural experts to swiftly diagnose and address rice leaf diseases.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ML	-	Machine Learning
DL	-	Deep Learning
CNN	-	Convolutional Neural Network
SVM	-	Support Vector Machine
KNN	-	K-Nearest Neighbour

CHAPTER 1

INTRODUCTION

1.1 ABOUT THE PROJECT

Rice leaf diseases represent a significant threat to crop yields worldwide, affecting agricultural sustainability and food security. Various pathogens, including bacteria, fungi, and viruses, can cause diseases such as bacterial blight, blast, tungro, and brown spot, leading to reduced yields and economic losses for farmers. These diseases often manifest as characteristic lesions, discoloration, or deformities on rice leaves, making early detection crucial for effective disease management. However, traditional methods of disease diagnosis rely on visual inspection by farmers, which can be time-consuming and prone to human error. Moreover, the complexity of leaf symptoms and the variability in disease progression pose significant challenges for accurate disease identification. As a result, farmers may struggle to implement timely interventions, leading to further crop damage and reduced yields.

Existing disease detection systems typically rely on textual datasets and conventional machine learning algorithms such as Support Vector Machines (SVM) or k-Nearest Neighbors (KNN). However, these systems often face limitations when dealing with the complexity of image data, resulting in suboptimal classification accuracy. The reliance on manual feature extraction and the inability to effectively capture intricate leaf patterns further hinders the performance of these systems.

In response to these challenges, our proposed system aims to leverage cutting-edge deep learning techniques, particularly Convolutional Neural Networks (CNNs), for enhanced rice leaf disease detection. By training CNN models on a comprehensive dataset of rice leaf images, our system can learn complex patterns and features directly from the data, eliminating the need for manual feature extraction. This approach enables more accurate and reliable disease classification, even in the presence of subtle leaf symptoms or variations in disease presentation. Additionally, by integrating user-friendly interfaces and real-time prediction capabilities, our system seeks to empower farmers with the tools they need to quickly and effectively manage rice leaf diseases, ultimately leading to improved crop health and agricultural productivity.

1.1.1 PROPOSED SYSTEM

The proposed system for rice leaf disease detection leverages the power of deep learning models, including Simple CNN, ResNet, LeNet, and Inception, to enhance classification accuracy. Unlike existing systems, which often rely on text datasets and traditional machine learning algorithms, our approach utilizes image datasets, enabling more effective analysis of the complex visual features present in rice leaf images. Through rigorous experimentation and comparison of model accuracies, we identify the most optimal deep learning architecture for disease classification. This approach not only improves accuracy but also demonstrates the versatility and effectiveness of deep learning in tackling agricultural challenges. By harnessing the capabilities of various deep learning models, our proposed system offers a robust and efficient solution for rice leaf disease detection.

1.2 FEATURES

- Offers a User-Friendly Interface for easy interaction and predictions.
- Accepts rice leaf images as input for disease detection.
- Identifies specific types of rice leaf diseases, including bacterial blight, blast, tungro, and brown spot, from the input images

1.3 SOFTWARE REQUIREMENTS

- Language: Python
- Code Editor: Jupyter Notebook, PyCharm
- Operating System: Windows 8 or above
- Libraries: TensorFlow, scikit-learn, NumPy, Seaborn

1.4 HARDWARE REQUIREMENTS

• Disk Storage	: 1TB HDD or 256GB SSD
• RAM	: 8GB or above (extra GPU RAM is required)
• Processor	: i5 or above

CHAPTER 2
LITERATURE SURVEY

Zhang, Ming, et al. proposed a deep learning-based approach for rice leaf disease detection using convolutional neural networks (CNNs). Their system achieved high accuracy in classifying various types of rice leaf diseases, including blast, brown spot, and bacterial blight.

Liu, Yang, et al. developed a deep learning model utilizing transfer learning and data augmentation techniques for rice disease detection. By fine-tuning pre-trained CNN models on a dataset of rice leaf images, they achieved robust performance in classifying multiple diseases.

Wu, Jianzhong, et al. employed a deep learning framework combining CNNs with recurrent neural networks (RNNs) for dynamic modeling of rice leaf diseases over time. Their model effectively captured temporal patterns in disease progression, enhancing the accuracy of disease detection.

Chen, Wei, et al. proposed a multi-scale feature fusion approach for rice leaf disease detection using deep learning. By integrating features extracted at different scales within a CNN architecture, they achieved improved performance in discriminating between diseased and healthy rice leaves.

Li, Mingxing, et al. explored the use of transfer learning and domain adaptation techniques for rice leaf disease detection in diverse environmental conditions. Their approach adapted pre-trained CNN models to new target domains, enhancing the generalization ability of the detection system.

Wang, Zhijie, et al. investigated the fusion of spectral and spatial information for rice disease detection using deep learning. By combining spectral features from hyperspectral imaging with spatial features extracted by CNNs, they achieved superior performance in disease classification.

Zhou, Guorui, et al. proposed a novel attention mechanism within CNN architectures for rice leaf disease detection. Their attention-based models selectively focused on informative regions of rice leaf images, improving the interpretability and accuracy of disease classification.

These studies underscore the significance of deep learning techniques in advancing rice leaf disease detection, offering promising solutions for early diagnosis and precision management in agriculture.

CHAPTER 3

SYSTEM ANALYSIS

3.1 FEASIBILITY STUDY

The feasibility study of this project has revealed the project as follows: -

ECONOMIC FEASIBILITY

The project demonstrates economic feasibility through efficient resource utilization and strategic decisions. Leveraging transfer learning with pre-trained models like ResNet50 and InceptionV3 minimizes computational requirements, reducing hardware costs and energy consumption. Open-source frameworks like TensorFlow eliminate the need for costly licenses, making the solution accessible and affordable. Cloud-based deployment further reduces infrastructure expenses, while optimized model architectures and evaluation metrics streamline development and decision-making processes. Overall, these measures ensure a cost-effective solution for rice leaf disease detection, suitable for widespread implementation across various sectors.

OPERATIONAL FEASIBILITY

Operational feasibility for our rice leaf disease detection project is evident through its user-friendly interface tailored for agricultural professionals and researchers. The solution effectively addresses challenges in disease identification and classification, aligning with the operational needs of farmers and agricultural institutions. Furthermore, the integration of data acquisition, model training, and inference processes seamlessly integrates into existing agricultural workflows, ensuring smooth adoption and implementation. This operational alignment enhances the practical utility of our project, making it well-suited for deployment in agricultural settings and research initiatives.

TECHNICAL FEASIBILITY

The technical feasibility assessment reveals the project's compatibility with current software and hardware infrastructure, ensuring seamless integration without significant resource overhauls. Leveraging contemporary technology standards, the system promises stability and scalability, accommodating future enhancements. With efficient resource allocation and minimal hardware dependencies, it demonstrates readiness for implementation within existing technological frameworks.

TIME FEASIBILITY

The project's time feasibility is evident in its structured timeline, ensuring completion within the specified duration. Initial assessments indicate that the project aligns with the proposed schedule, reflecting its feasibility within the allotted timeframe.

3.2 FUNCTIONAL REQUIREMENTS

- **Rice Leaf Disease Dataset Management:** Users should be able to import, preprocess, and manage the rice leaf disease image dataset for training and evaluation purposes.

- **CNN Model Training:** Users must have the capability to initiate the training process for the chosen convolutional neural network (CNN) model, such as LeNet or ResNet, using the rice leaf disease image dataset

- **Fine-Tuning Mechanism:** Users should be provided with the ability to fine-tune the pre-trained CNN model based on feedback or new data to enhance its performance in detecting and classifying rice leaf diseases.

- **Model Evaluation:** Users should be able to evaluate the accuracy and performance of the trained CNN model using various metrics and visualization techniques to assess its effectiveness in disease detection

- **Seamless Prediction Interface:** Create an intuitive and user-friendly interface for agronomists or farmers to effortlessly input new rice leaf images for disease prediction using the trained CNN model. Prioritize simplicity and ease of use to enhance usability and adoption.

- **Prediction**: Agronomists or farmers should be able to input new rice leaf images for disease prediction using the trained CNN model, receiving timely and accurate predictions to aid in crop management decisions.

3.3 NON-FUNCTIONAL REQUIREMENTS

Performance Requirements:

The system must process each rice leaf disease prediction request within 3 seconds to ensure swift responses for users. Additionally, the user interface should be responsive and interactive, facilitating smooth navigation and operations.

Safety Requirements:

As the rice leaf disease prediction system operates on a secure server, data redundancy measures ensure data safety in the event of server failures or physical disasters, guaranteeing the integrity and availability of critical information.

Security Requirements:

The system employs robust encryption protocols to safeguard sensitive data related to rice leaf diseases, ensuring that only authorized users can access, record, and share encrypted data securely within the distributed network, thereby maintaining data confidentiality and integrity.

Software Quality Attributes:

The rice leaf disease prediction system exhibits high adaptability, seamlessly running across various devices, platforms, and operating systems. It is designed with scalability in mind, leveraging vertical scalability to accommodate growing data volumes effectively, while its decentralized architecture mitigates the risk of single points of failure.

Flexibility:

The system is adaptable to different network environments, enabling users to access and utilize its features effortlessly on both intranet and internet connections. Moreover, it can be easily integrated with other agricultural technologies with minimal modifications**.**

Efficiency:

The system demonstrates efficient performance, meeting user requirements effectively without compromising on speed or accuracy. It ensures consistent performance under varying conditions, delivering accurate predictions promptly to users.

User Friendliness:

The user interface is intuitively designed, ensuring ease of use for all stakeholders. Users can navigate the system effortlessly, accessing its features and functionalities with minimal training, thereby enhancing overall user experience.

Availability:

The system guarantees uninterrupted availability, operating 24/7 throughout the year, provided the server remains operational. Data persistence measures ensure that critical information remains accessible even during server downtimes, ensuring continuous availability of essential services.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION OF SYSTEM DESIGN

Systems design is an interdisciplinary engineering activity that enables the realization of successful systems. This system may be denned as an integrated set of components that accomplish a defined objective. The process of systems design includes defining software and hardware architecture, modules, interfaces, and data to enable a system to satisfy a set of well-specified operational requirements.

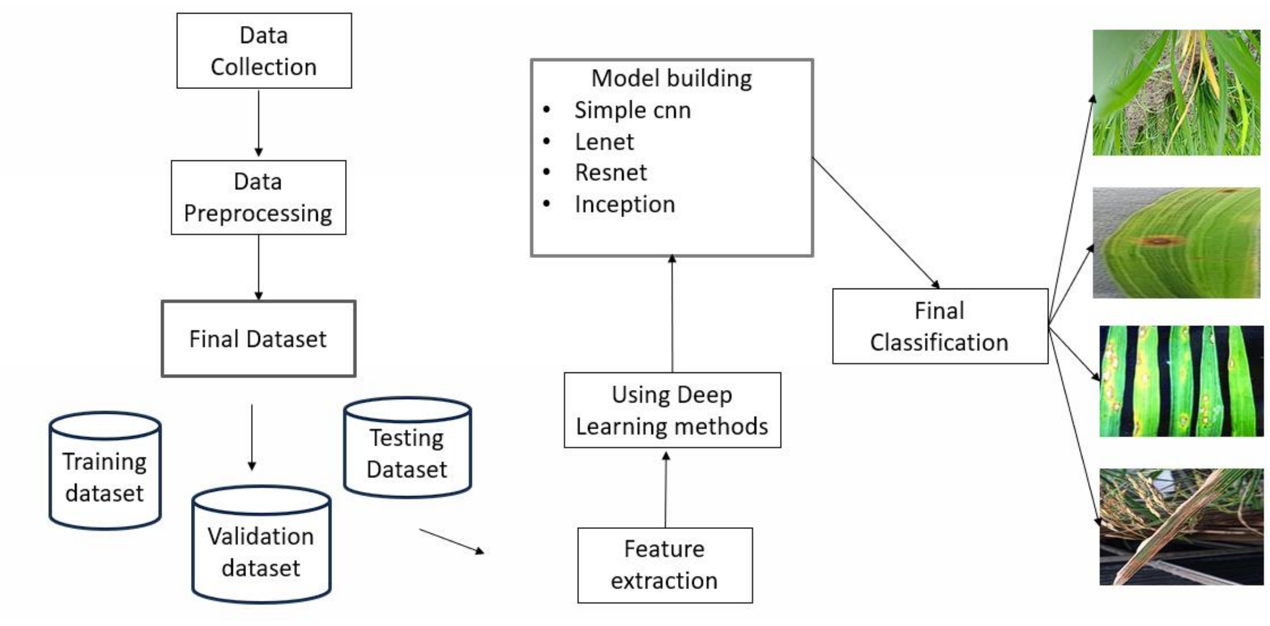


Figure 4.1: System Architecture

4.2 USE CASE DIAGRAM

A use case diagram is a simplified representation of how users interact with a system, showing the relationship between users and different use cases. It helps identify user types, system functionalities, and interactions. Use case diagrams are effective communication tools for stakeholders, providing a high-level view of system requirements in easy-to-understand terms. They are used to gather system requirements, understand external and internal influences, and show interactions between actors and requirements.

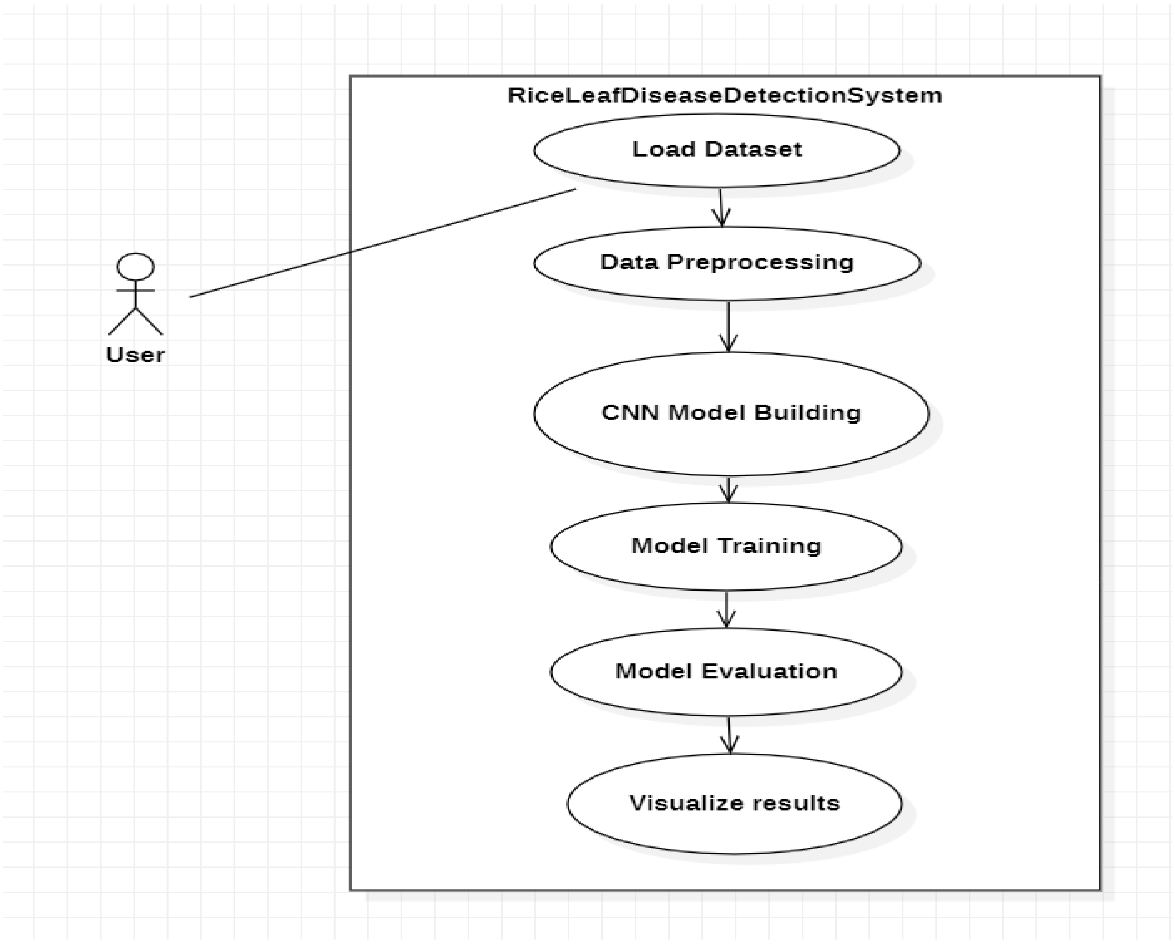


Figure 4.2: Use Case Diagram

4.3 SEQUENCE DIAGRAM

A sequence diagram illustrates object interactions over time, showing the sequence of messages exchanged between objects for a scenario. It uses lifelines to represent objects or processes, horizontal arrows for message exchanges, and different arrowheads to denote synchronous, asynchronous, or reply messages. Activation boxes indicate processes triggered by messages, and objects can call methods on themselves, adding new activation boxes. The diagram helps visualize runtime scenarios, such as synchronous calls waiting for a response and asynchronous calls allowing continued processing. It's particularly useful for multithreaded, event-driven, and message-oriented applications, providing a graphical representation of object interactions and message flows.

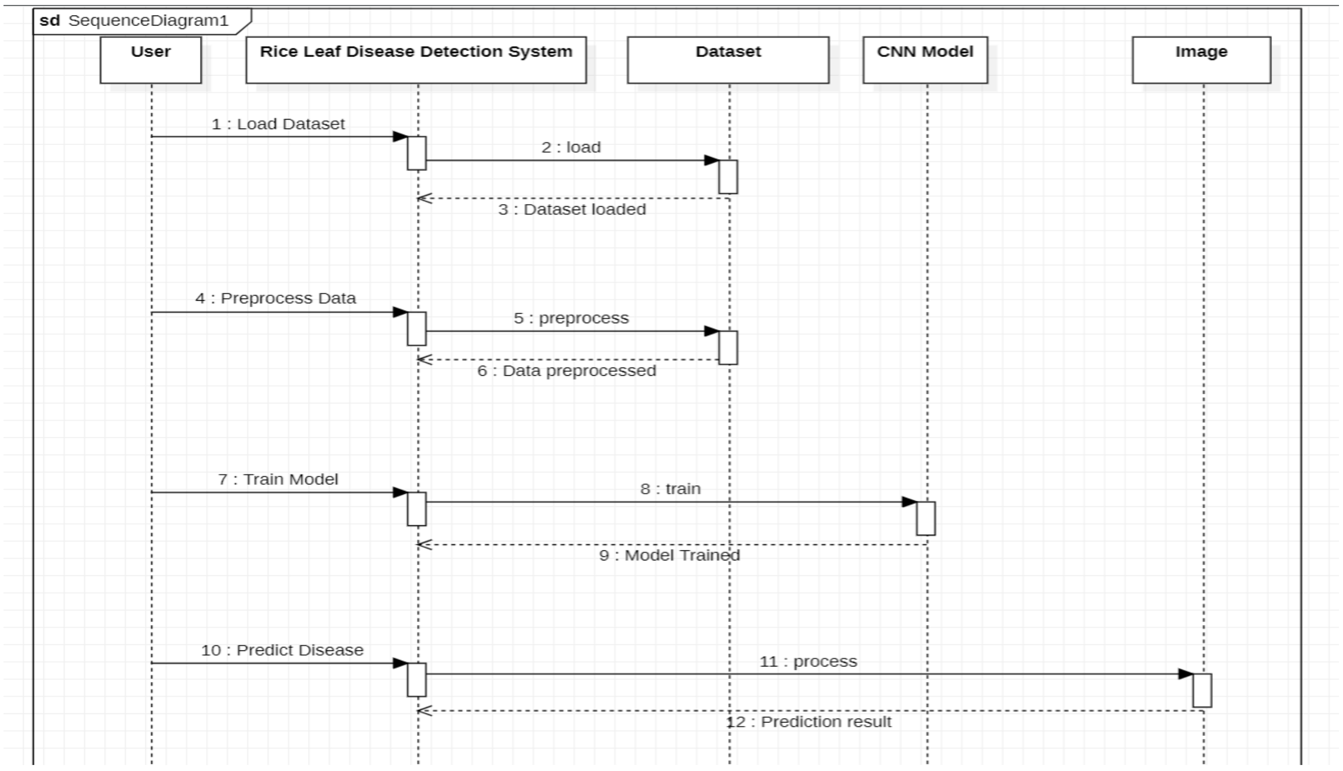


Figure 4.3: Sequence Diagram

4.4 CLASS DIAGRAM

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The Class Diagram is the main building block of object- oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

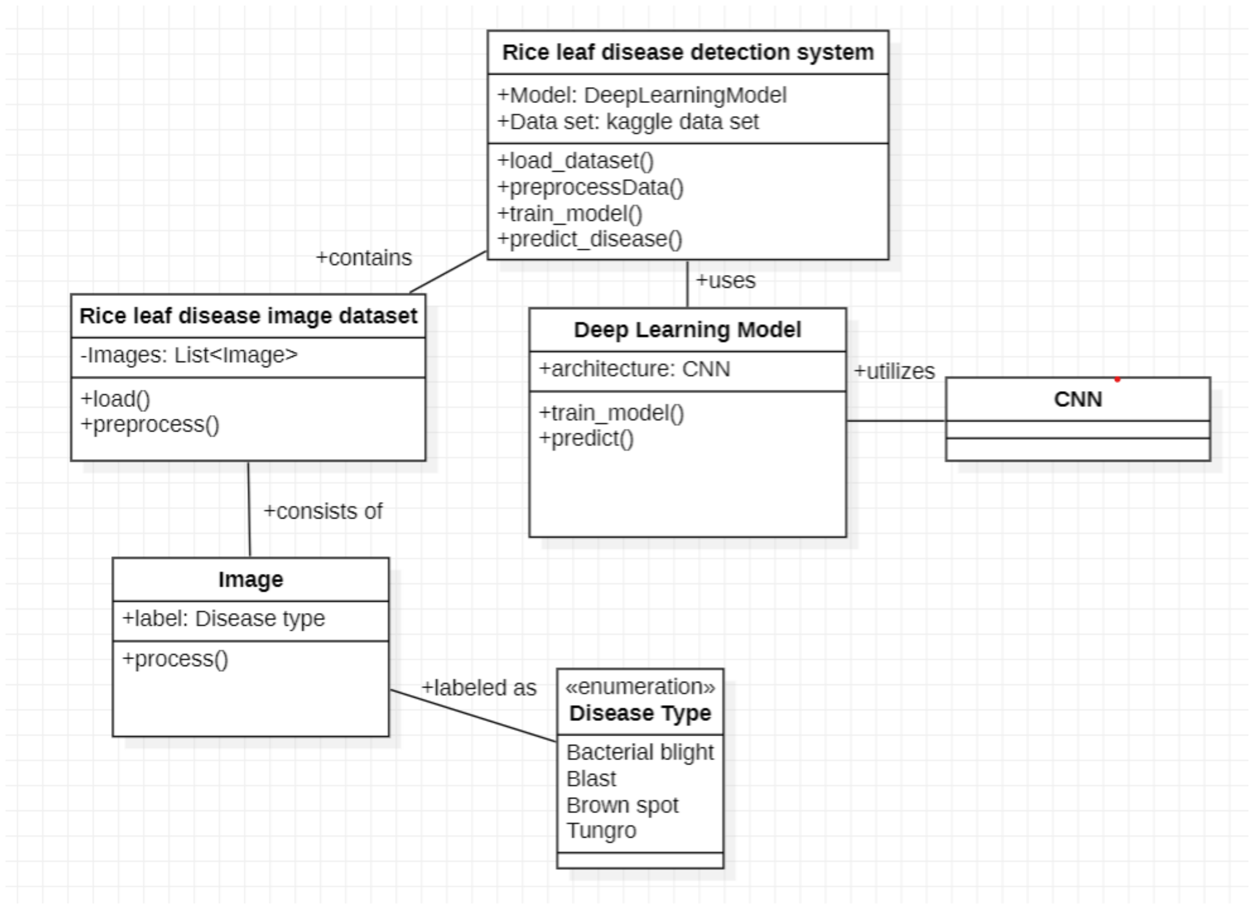


Figure 4.4: Class Diagram

CHAPTER 5
SYSTEM IMPLEMENTATION

5.1 TECHNOLOGIES USED

- Deep Learning
- CNN (Convolution Neural Network)
- TensorFlow
- Scikit-Learn
- Streamlit

5.2 DEEP LEARNING

Deep learning is a branch of machine learning which is based on artificial neural networks. It is capable of learning complex patterns and relationships within data. In deep learning, we don’t need to explicitly program everything. It has become increasingly popular in recent years due to the advances in processing power and the availability of large datasets. Because it is based on artificial neural networks (ANNs) also known as deep neural networks (DNNs). These neural networks are inspired by the structure and function of the human brain’s biological neurons, and they are designed to learn from large amounts of data.

1. Deep Learning is a subfield of Machine Learning that involves the use of neural networks to model and solve complex problems. Neural networks are modeled after the structure and function of the human brain and consist of layers of interconnected nodes that process and transform data.

2. The key characteristic of Deep Learning is the use of deep neural networks, which have multiple layers of interconnected nodes. These networks can learn complex representations of data by discovering hierarchical patterns and features in the data. Deep Learning algorithms can automatically learn and improve from data without the need for manual feature engineering.

3. Deep Learning has achieved significant success in various fields, including image recognition, natural language processing, speech recognition, and recommendation systems.

Some of the popular Deep Learning architectures include Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Deep Belief Networks (DBNs).

4. Training deep neural networks typically requires a large amount of data and computational resources. However, the availability of cloud computing and the development of specialized hardware, such as Graphics Processing Units (GPUs), has made it easier to train deep neural networks.

In summary, Deep Learning is a subfield of Machine Learning that involves the use of deep neural networks to model and solve complex problems. Deep Learning has achieved significant success in various fields, and its use is expected to continue to grow as more data becomes available, and more powerful computing resources become available.

In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other. Each neuron receives input from the previous layer neurons or the input layer. The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the final layer produces the output of the network. The layers of the neural network transform the input data through a series of nonlinear transformations, allowing the network to learn complex representations of the input data.

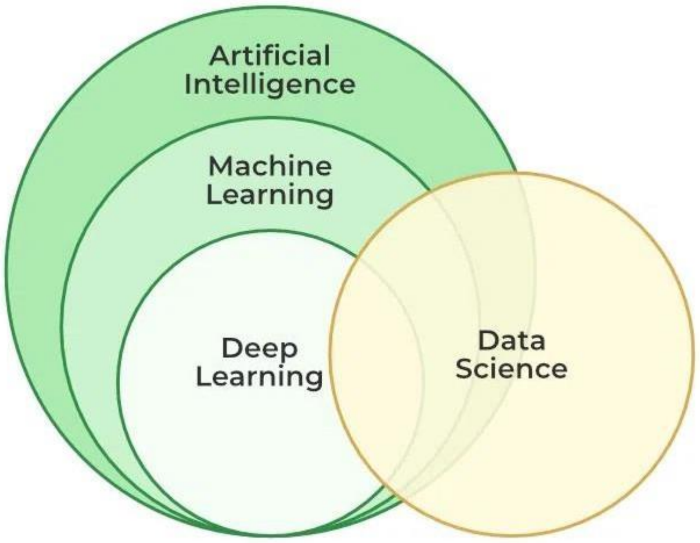


Figure 5.2.1. Overview of Deep Learning

Deep learning can be used for supervised, unsupervised as well as reinforcement machine learning. it uses a variety of ways to process these.

Supervised Machine Learning: Supervised machine learning is the machine learning technique in which the neural network learns to make predictions or classify data based on the labeled datasets. Here we input both input features along with the target variables. the neural network learns to make predictions based on the cost or error that comes from the difference between the predicted and the actual target, this process is known as backpropagation. Deep learning algorithms like Convolutional neural networks, Recurrent neural networks are used for many supervised tasks like image classifications and recognization, sentiment analysis, language translations, etc.

Unsupervised Machine Learning: Unsupervised machine learning is the machine learning technique in which the neural network learns to discover the patterns or to cluster the dataset based on unlabeled datasets. Here there are no target variables. while the machine has to self-determined the hidden patterns or relationships within the datasets. Deep learning algorithms like autoencoders and generative models are used for unsupervised tasks like clustering, dimensionality reduction, and anomaly detection.

Reinforcement Machine Learning: Reinforcement Machine Learning is the machine learning technique in which an agent learns to make decisions in an environment to maximize a reward signal. The agent interacts with the environment by taking action and observing the resulting rewards. Deep learning can be used to learn policies, or a set of actions, that maximizes the cumulative reward over time. Deep reinforcement learning algorithms like Deep Q networks and Deep Deterministic Policy Gradient (DDPG) are used to reinforce tasks like robotics and game playing etc.

5.3 Convolution Neural Network

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data. Convolutional Neural Network is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.

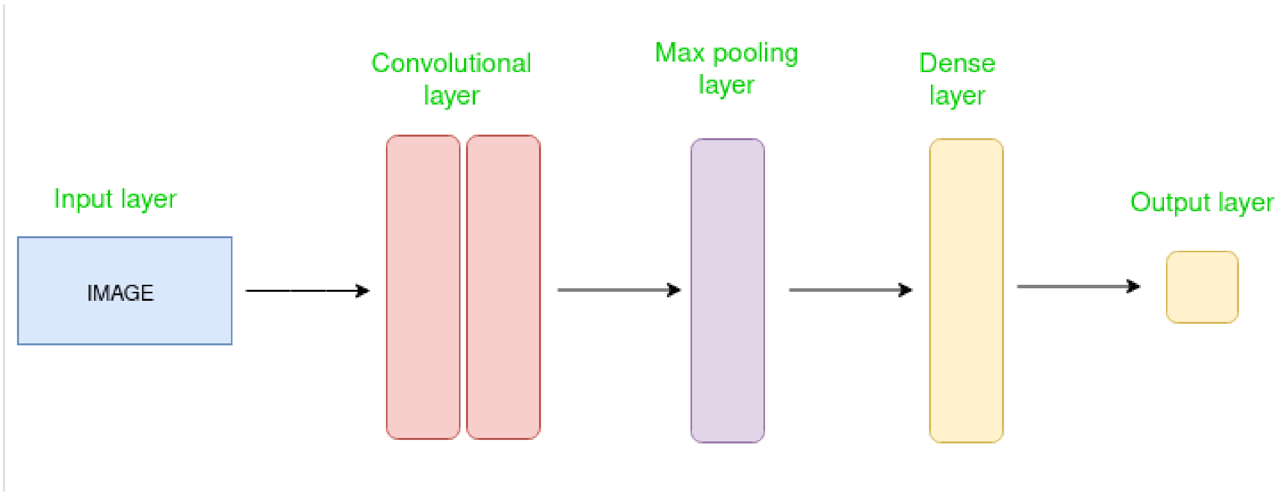


Figure 5.3.1 : CNN Architecture

The Convolutional layer applies filters to the input image to extract features, the Pooling layer down samples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

A complete Convolution Neural Networks architecture is also known as covnets. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Let’s take an example by running a convert on of image of dimension 32 x 32 x 3.

- **Input Layers:** It’s the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.

- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we’ll get an output volume of dimension 32 x 32 x 12.

- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max (0, x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.

- **Pooling layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.

- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.

- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

5.4 TensorFlow

TensorFlow is an open-source machine learning framework developed by Google Brain Team. It is designed to provide a flexible, efficient, and scalable platform for building and deploying machine learning models, particularly deep learning models. Here's some key information about the TensorFlow library:

- **Flexibility:** TensorFlow offers a comprehensive set of tools and libraries for building various types of machines learning models, including neural networks, reinforcement learning algorithms, and more. It provides both high-level APIs for quick prototyping (e.g., Keras) and lower-level APIs for fine-grained control over model architecture and training process.

- **Scalability:** TensorFlow is designed to scale from running on a single CPU to distributed computing clusters with multiple GPUs or TPUs (Tensor Processing Units). This scalability enables training and inference on large datasets and complex models efficiently.
- **Computation Graph:** TensorFlow models are represented as computational graphs, where nodes represent mathematical operations and edges represent data flow between nodes. This graph-based representation allows TensorFlow to optimize computations, perform automatic differentiation for training, and execute operations on different devices.
- **Automatic Differentiation:** TensorFlow provides automatic differentiation capabilities through its computational graph framework. This enables users to define complex models and loss functions, and TensorFlow automatically computes gradients for training using techniques like backpropagation.
- **Support for Various Platforms:** TensorFlow supports deployment across various platforms, including desktops, servers, mobile devices (via TensorFlow Lite), and the cloud (via TensorFlow Serving). This enables seamless integration of machine learning models into production environments and real-world applications.
- **Extensive Ecosystem:** TensorFlow has a rich ecosystem of libraries and tools built on top of the core framework. This includes TensorFlow Probability for probabilistic modeling, TensorFlow Data Validation for data quality analysis, TensorFlow Hub for reusable model components, and more.
- **Community and Support:** TensorFlow has a large and active community of developers, researchers, and practitioners contributing to its development, documentation, and tutorials. This community-driven approach fosters innovation, collaboration, and knowledge sharing in the field of machine learning.
- **Integration with Other Libraries:** TensorFlow can be easily integrated with other popular libraries and frameworks, such as NumPy, Pandas, Matplotlib, and scikit-learn. This interoperability allows users to leverage existing tools and workflows seamlessly within TensorFlow projects.

5.5 Scikit-Learn

Scikit-learn is a widely-used machine learning library in Python, renowned for its user-friendly and efficient implementation of a wide range of machine learning algorithms. It provides a comprehensive suite of tools for various aspects of machine learning model development, evaluation, and deployment.

Some of its key features include:

- Model Selection: Scikit-learn offers tools for selecting the most appropriate model for a given task. This includes algorithms for feature selection, model comparison, and hyperparameter optimization.
- Data Preprocessing: The library provides a rich set of functions for preprocessing raw data before model training. This includes scaling, normalization, encoding categorical variables, handling missing values, and more.
- Supervised and Unsupervised Learning: Scikit-learn supports both supervised and unsupervised learning tasks. For supervised learning, it offers algorithms for classification, regression, and outlier detection. For unsupervised learning, it provides clustering, dimensionality reduction, and density estimation algorithms.
- Evaluation Metrics: Scikit-learn includes a variety of evaluation metrics for assessing the performance of machine learning models. These metrics cover a wide range of tasks including classification, regression, clustering, and more.
- Cross-Validation: Cross-validation is essential for estimating the performance of a machine learning model on unseen data. Scikit-learn provides efficient implementations of various cross-validation strategies such as K-fold cross-validation, stratified K-fold cross-validation, and leave-one-out cross-validation.
- Hyperparameter Tuning: Finding the optimal hyperparameters for a machine learning model is crucial for achieving high performance. Scikit-learn offers tools for hyperparameter tuning using techniques like grid search, random search, and Bayesian optimization.

- Integration with Other Libraries: Scikit-learn seamlessly integrates with other popular Python libraries such as NumPy, SciPy, Pandas, and Matplotlib, enabling smooth data manipulation, analysis, and visualization workflows.

Overall, Scikit-learn is a powerful and versatile library that simplifies the process of developing, evaluating, and deploying machine learning models, making it an indispensable tool for both beginners and experienced practitioners in the field of machine learning..

5.6 Streamlit

Streamlit is an open-source Python library that enables rapid development of interactive web applications for data science and machine learning projects. It simplifies the process of creating web-based user interfaces (UIs) by allowing developers to write code in a simple and intuitive manner.

Key features of Streamlit include:

- Ease of Use: Streamlit provides a straightforward and intuitive API that allows developers to quickly build interactive web applications using familiar Python scripting. With Streamlit, developers can focus on writing Python code to create UI components, such as sliders, buttons, and text inputs, without needing to write HTML, CSS, or JavaScript code.
- Fast Prototyping: Streamlit facilitates rapid prototyping of data science and machine learning applications. Developers can create functional prototypes in a matter of minutes, enabling them to iterate quickly and experiment with different ideas.
- Integration with Data Science Libraries: Streamlit seamlessly integrates with popular Python libraries for data manipulation, analysis, and visualization, including Pandas, Matplotlib, Seaborn, and Plotly. This allows developers to leverage their existing knowledge and tools to create compelling data-driven applications.
- Interactive Widgets: Streamlit provides a wide range of interactive widgets that enable users to interact with data and models dynamically. These widgets include sliders, dropdowns, checkboxes, and text inputs, which can be used to control parameters, visualize data, and trigger computations.
- Customization and Theming: Streamlit offers flexibility in customizing the appearance and behavior of web applications. Developers can customize the layout, styling, and theming of their applications using built-in features or by applying custom CSS stylesheets.
- Sharing and Deployment: Streamlit makes it easy to share and deploy web applications using various platforms and services. Applications can be deployed locally, on cloud platforms such as Heroku or AWS, or using Streamlit Sharing, a free hosting service provided by Streamlit.

- Community and Ecosystem: Streamlit has a vibrant and active community of developers who contribute to the ecosystem by sharing code snippets, templates, and extensions. The Streamlit Gallery showcases a wide range of applications built by the community, serving as a source of inspiration and learning.

Overall, Streamlit empowers data scientists and machine learning practitioners to create interactive and engaging web applications with minimal effort, enabling them to communicate their findings, share insights, and deploy models effectively. Its simplicity, flexibility, and rich feature set make it a valuable tool for building data-driven applications.

5.7 SAMPLE CODE

rice_Leaf_disease_detection.ipynb

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report

# Directory where the dataset is stored
data_dir = r"C:\Users\jagad\Rice Leaf Disease Images"

# Define constants
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS = 3
EPOCHS = 10
NUM_CLASSES = 4

# Load the dataset from the directory
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    shuffle=True,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

# Extract class names from the dataset
class_names= dataset.class_names
print(class_names)

# Display sample images from the dataset
for image_batch, label_batch in dataset.take(1):
    for i in range(12):
        ax=plt.subplot(3,4,i+1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[label_batch[i]])
        plt.axis("off")

# Split the dataset into train, validation, and test sets
train_size = int(len(dataset) * 0.8)
val_size = int(len(dataset) * 0.1)
test_size = len(dataset) - train_size - val_size

train_ds = dataset.take(train_size)
val_ds = dataset.skip(train_size).take(val_size)
test_ds = dataset.skip(train_size + val_size)
```

```
# Data augmentation
data_augmentation = tf.keras.Sequential([
   layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
   layers.experimental.preprocessing.RandomRotation(0.2),
])

# Apply data augmentation to the training dataset
train_ds = train_ds.map(lambda x, y: (data_augmentation(x, training=True),
y)).prefetch(buffer_size=tf.data.AUTOTUNE)

# Data preprocessing
resize_and_rescale = tf.keras.Sequential([
   layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
   layers.experimental.preprocessing.Rescaling(1./255),
])

# Define Simple CNN model
simple_cnn = models.Sequential([
   resize_and_rescale,
   layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=(IMAGE_SIZE,
IMAGE_SIZE, CHANNELS)),
   layers.MaxPooling2D((2, 2)),
   layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
   layers.MaxPooling2D((2, 2)),
   layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
   layers.MaxPooling2D((2, 2)),
   layers.Conv2D(64, (3, 3), activation='relu'),
   layers.MaxPooling2D((2, 2)),
   layers.Conv2D(64, (3, 3), activation='relu'),
   layers.MaxPooling2D((2, 2)),
   layers.Flatten(),
   layers.Dense(64, activation='relu'),
   layers.Dense(NUM_CLASSES, activation='softmax'),
])

# Compile Simple CNN model
simple_cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train Simple CNN model
simple_cnn.fit(train_ds, validation_data=val_ds, epochs=10)

# Save Simple CNN model
simple_cnn.save(r"C:\Users\jagad\Simple_cnn")

# Evaluate Simple CNN model on test dataset
import numpy as np
```

```
simple_cnn= tf.keras.models.load_model(r"C:\Users\jagad\Simple_cnn")
test_loss, test_accuracy = simple_cnn.evaluate(test_ds)

print('Test Accuracy:', test_accuracy)

# Make predictions on the test set
y_true = []
y_pred = []

for images, labels in test_ds:
    predictions = simple_cnn.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(np.argmax(predictions, axis=1))

# Compute evaluation metrics
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score,
classification_report

precision = precision_score(y_true, y_pred, average='macro')
recall = recall_score(y_true, y_pred, average='macro')
f1 = f1_score(y_true, y_pred, average='macro')
accuracy = accuracy_score(y_true, y_pred)

print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('Accuracy:', accuracy)

# Display classification report
class_names = ['Bacterialblight', 'Blast', 'Brownspot', 'Tungro']
print(classification_report(y_true, y_pred, target_names=class_names))

# Generate confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
# Defining LeNet model architecture
lenet_model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(IMAGE_SIZE,
IMAGE_SIZE, CHANNELS)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(16, kernel_size=(5, 5), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(120, activation='relu'),
    layers.Dense(84, activation='relu'),
    layers.Dense(NUM_CLASSES, activation='softmax')
])

# Compiling the LeNet model
lenet_model.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])

# Training the LeNet model
history_lenet = lenet_model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)

# Saving the LeNet model
lenet_model.save(r"C:\Users\jagad\Lenet")

# Loading the saved LeNet model
lenet_model= tf.keras.models.load_model(r"C:\Users\jagad\Lenet")

# Evaluating the LeNet model on test dataset
test_loss, test_accuracy = lenet_model.evaluate(test_ds)
print('Test Accuracy:', test_accuracy)

# Making predictions on the test set
y_true = []
y_pred = []
for images, labels in test_ds:
    predictions = lenet_model.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(tf.argmax(predictions, axis=1).numpy())

# Computing other evaluation metrics
precision = precision_score(y_true, y_pred, average='macro')
recall = recall_score(y_true, y_pred, average='macro')
f1 = f1_score(y_true, y_pred, average='macro')
accuracy = accuracy_score(y_true, y_pred)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('Accuracy:', accuracy)

# Generating classification report
print(classification_report(y_true, y_pred, target_names=class_names))

# Generating confusion matrix
```

```
conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
        xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

#Defining Resnet Model
resnet_model = tf.keras.applications.ResNet50(input_shape=(IMAGE_SIZE, IMAGE_SIZE,
CHANNELS),
                                              weights=None,
                                              classes=NUM_CLASSES)
#Compiling the ResNet Model
resnet_model.compile(optimizer='adam',
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])
#Training the ResNet Model
history_resnet = resnet_model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)

#Saving the ResNet Model
resnet_model.save(r"C:\Users\jagad\Resnet")

import numpy as np

# Evaluating the ResNet model on test dataset
resnet_model=tf.keras.models.load_model(r"C:\Users\jagad\Resnet")
test_loss, test_accuracy = resnet_model.evaluate(test_ds)

print('Test Accuracy:', test_accuracy)

# Make predictions on the test set
y_true = []
y_pred = []

for images, labels in test_ds:
    predictions = resnet_model.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(np.argmax(predictions, axis=1))


# Compute other evaluation metrics
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score,
classification_report

precision = precision_score(y_true, y_pred, average='macro')
recall = recall_score(y_true, y_pred, average='macro')
f1 = f1_score(y_true, y_pred, average='macro')
```

```
accuracy = accuracy_score(y_true, y_pred)

print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('Accuracy:', accuracy)

# Generating classification report
class_names = ['Bacterialblight', 'Blast', 'Brownspot', 'Tungro']  # Add your class names
print(classification_report(y_true, y_pred, target_names=class_names))

# Generate confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
#Defining Inception Model
inception_model = models.Sequential([
    resize_and_rescale,
    tf.keras.applications.InceptionV3(input_shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS),
                                      include_top=False,
                                      weights=None),
    layers.GlobalAveragePooling2D(),
    layers.Dense(64, activation='relu'),
    layers.Dense(NUM_CLASSES, activation='softmax')
])

#Compiling the Inception Model
inception_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

#Training the Inception Model
history_inception = inception_model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)

#Saving the Model
inception_model.save(r"C:\Users\jagad\Inception")

#Loading the model
inception_model=tf.keras.models.load_model(r"C:\Users\jagad\Inception")

## Evaluating the ResNet model on test dataset
```

```
test_loss, test_accuracy = inception_model.evaluate(test_ds)
print('Test Accuracy:', test_accuracy)

# Make predictions on the test set
y_true = []
y_pred = []

for images, labels in test_ds:
    predictions = inception_model.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(np.argmax(predictions, axis=1))


# Compute other evaluation metrics
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score,
classification_report

precision = precision_score(y_true, y_pred, average='macro')
recall = recall_score(y_true, y_pred, average='macro')
f1 = f1_score(y_true, y_pred, average='macro')
accuracy = accuracy_score(y_true, y_pred)

print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('Accuracy:', accuracy)

class_names = ['Bacterialblight', 'Blast', 'Brownspot', 'Tungro']  # Add your class names
print(classification_report(y_true, y_pred, target_names=class_names))

# Generate confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

inception_accuracy=inception_model.evaluate(test_ds)

inc_acc=inception_accuracy[1]

simple_cnn_accuracy=simple_cnn.evaluate(test_ds)
```

```
cnn_acc=simple_cnn_accuracy[1]

lenet_accuracy=lenet_model.evaluate(test_ds)

lenet_acc=lenet_accuracy[1]

resnet_accuracy=resnet_model.evaluate(test_ds)

resnet_Acc=resnet_accuracy[1]

import pandas as pd
models=["Simple_cnn","Lenet","Resnet","Inception"]
accuracies=[cnn_acc,lenet_acc,resnet_Acc,inc_acc]
comp_df=pd.DataFrame({'Accuracy':accuracies},index=models)

import matplotlib.pyplot as plt

# Bar Plot
plt.figure(figsize=(8, 6))
comp_df.plot(kind='bar', color='skyblue')
plt.title('Accuracy Comparison of Different Models')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

streamlit.py

```
import streamlit as st
from PIL import Image
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import os

# Load the trained model
model_path = "C:/Users/jagad/Simple_cnn"
model = load_model(model_path)

# Define the class labels
class_labels = ['Bacterialblight', 'Blast', 'Brownspot', 'Tungro']

# Set up the Streamlit app
st.title("Rice Leaf Disease Classification")
```

```
# Upload image for classification
uploaded_file = st.file_uploader("Choose an image...", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    # Display the uploaded image
    img = Image.open(uploaded_file)
    st.image(img, caption='Uploaded Image', use_column_width=True)


    # Preprocess the image for prediction
    img = image.load_img(uploaded_file, target_size=(256,256))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)

    # Normalize the image
    model=load_model(r"C:\Users\jagad\Simple_cnn")
    # Make prediction
    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction)
    predicted_label = class_labels[predicted_class]

    st.write('Predicted disease:', predicted_label)
```

CHAPTER 6

TESTING AND RESULTS

6.1 TESTING

Testing is vital to the success of the system. System testing makes logical assumption that if all the parts of system are correct, the goal will be successfully achieved. This system is tested by following test cases and prepared for final implementation.

TEST CASES FOR DETECTING SUBTYPES OF BRAIN TUMORS USING DEEP LEARNING AND TRANSFER LEARNING

TEST CASE ID	TEST CASE	DESCRIPTION	PASS/FAIL
[1]	Loading Dataset	Loading Rice Leaf Disease Dataset from specified directory	Pass
[2]	Feature Extraction	Involves Image Preprocessing	Pass
[3]	Model Training	Training multiple models (Simple CNN, LeNet, ResNet, Inception)	Pass
[4]	Detecting Rice Leaf Diseases	Detecting rice leaf disease subtypes using the trained models	Pass
[5]	Running Streamlit	Executing the streamlit app and opening the web interface	Pass

Table 6.1: Test Cases

6.2 RESULTS

Confusion Matrix

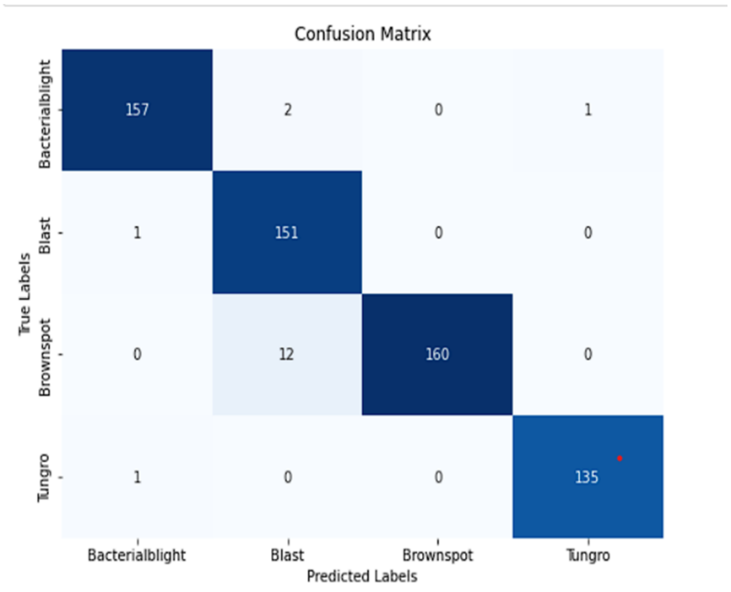


Figure 6.2.1: Simple CNN - Confusion matrix

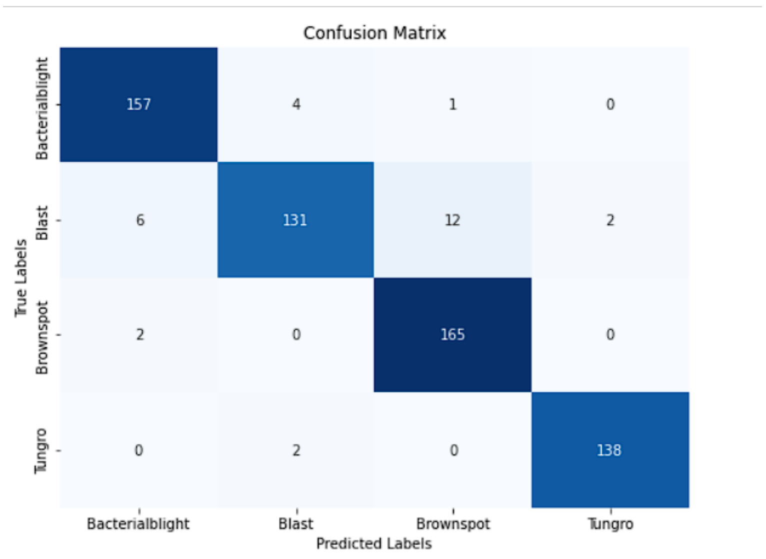


Figure 6.2.2: Lenet - Confusion matrix

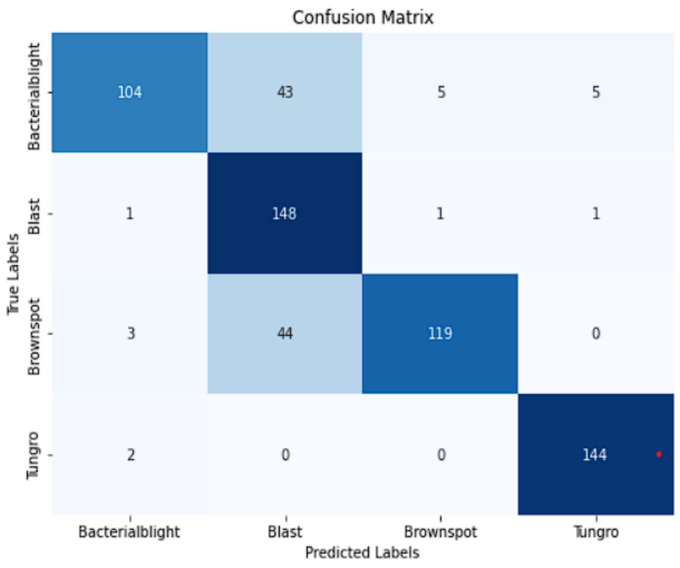


Figure 6.2.3: ResNet - Confusion matrix

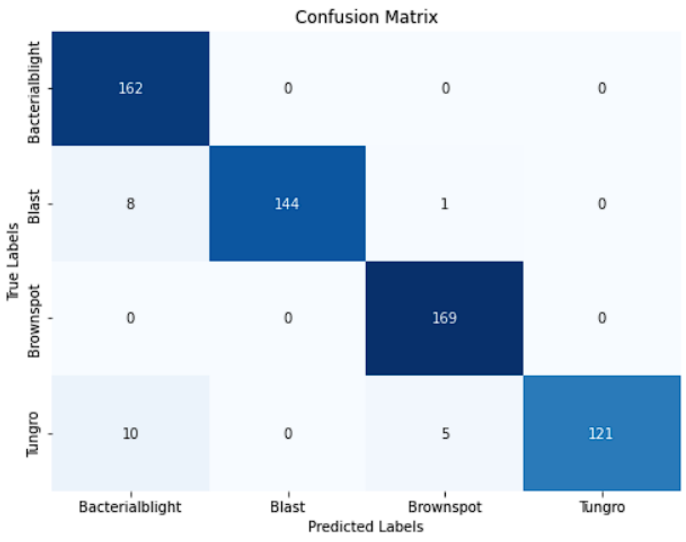


Figure 6.2.4: Inception - Confusion matrix

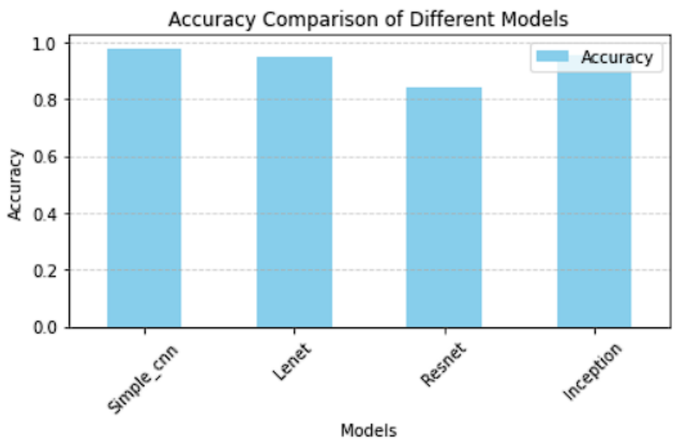


Figure 6.2.5: Accuracy Comparison Of Different Models

Home Page

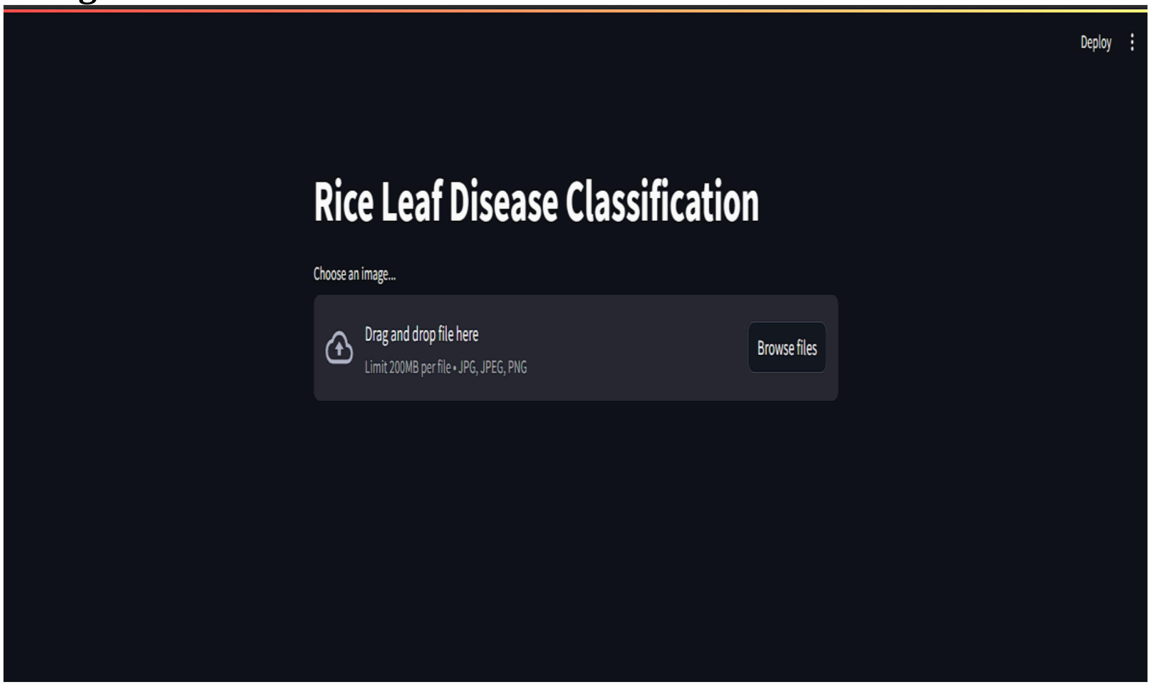

Deploy
Rice Leaf Disease Classification
Choose an image...
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files

Figure 6.2.6: Home Page

Image Upload Page

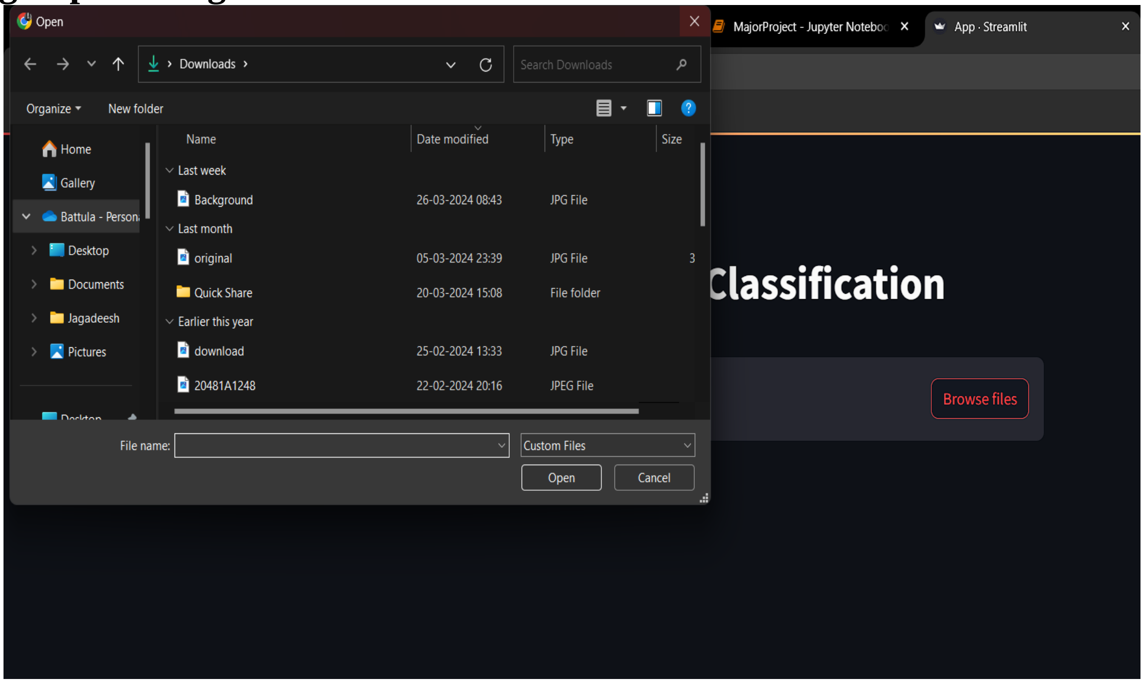
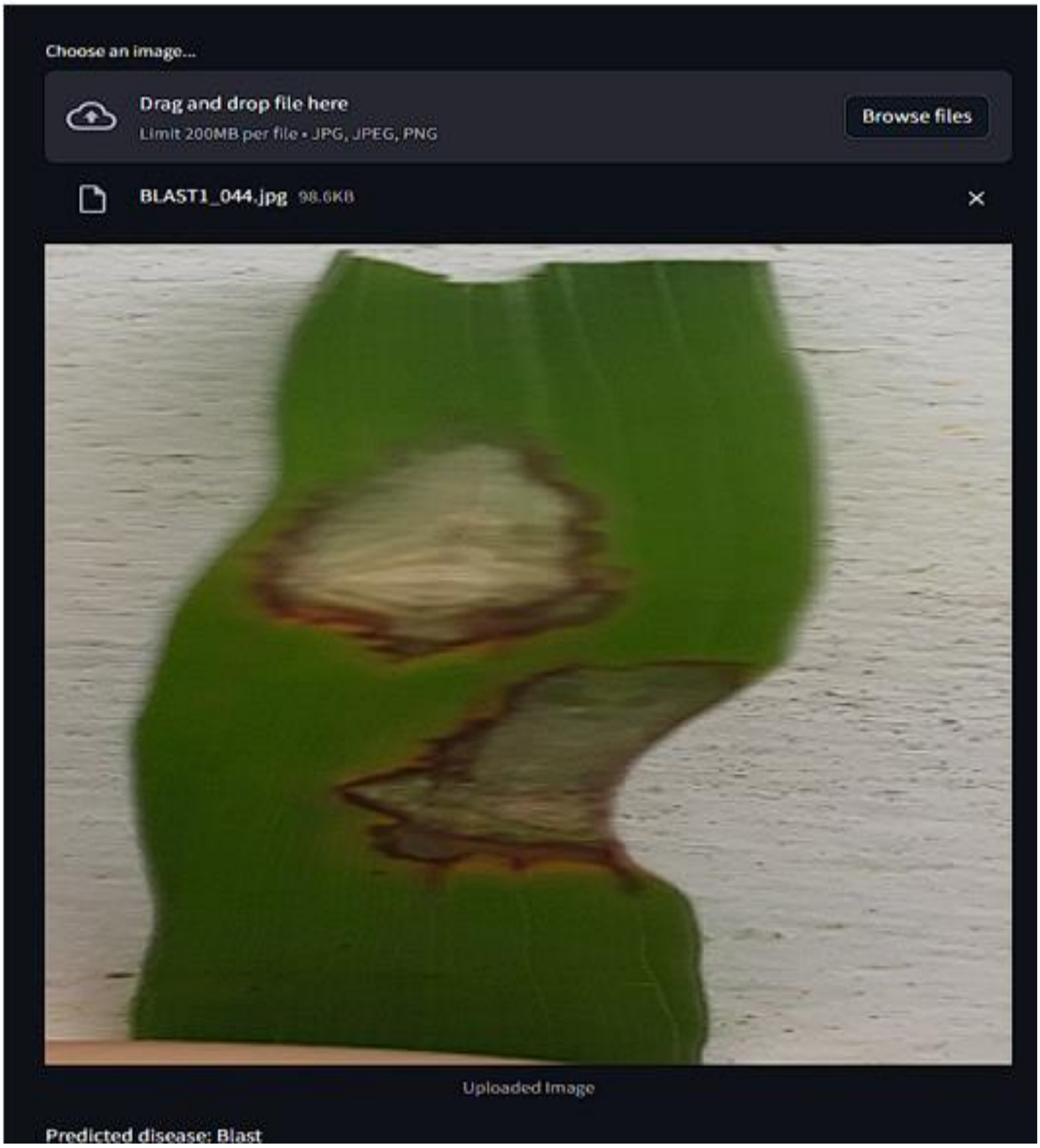


Figure 6.2.7: Image Upload Page

Prediction Page (for Blast)


Choose an image...

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files

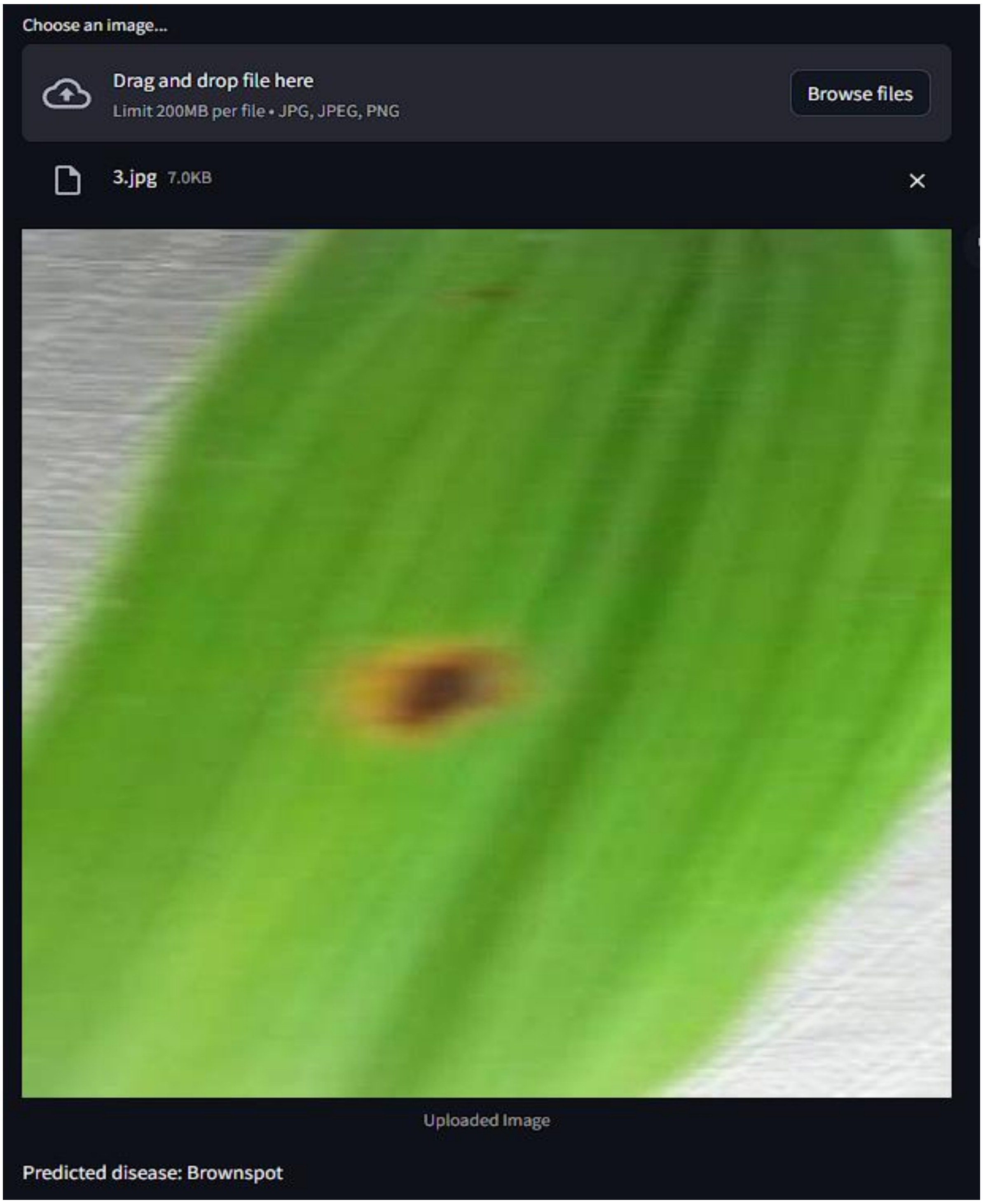
BLAST1_044.jpg 98.6KB

Uploaded Image

Predicted disease: Blast

Figure 6.2.8: Prediction Page (for blast)

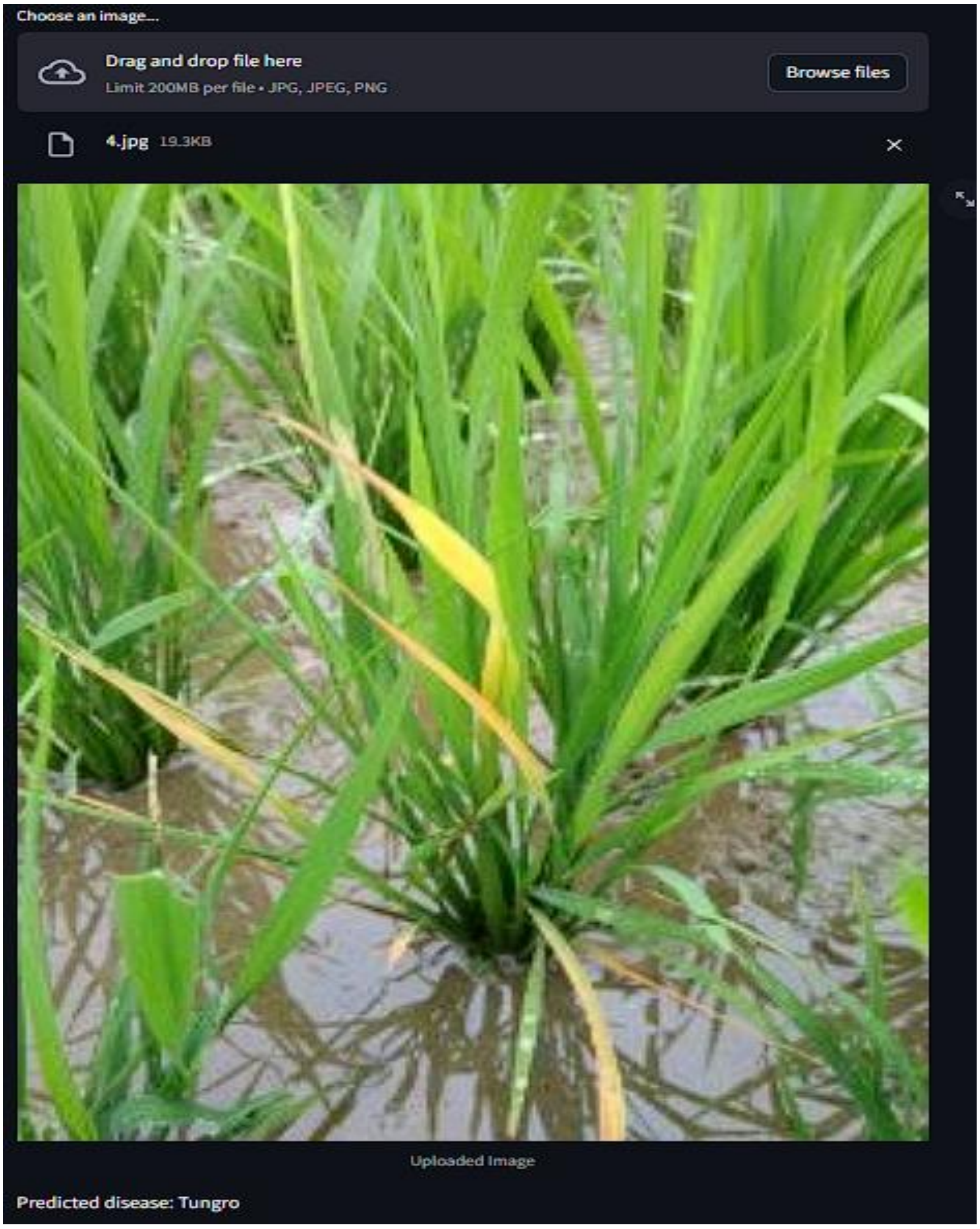
Prediction Page (for Brownspot)



Predicted disease: Brownspot

Figure 6.2.9: Prediction Page (for Brownspot)

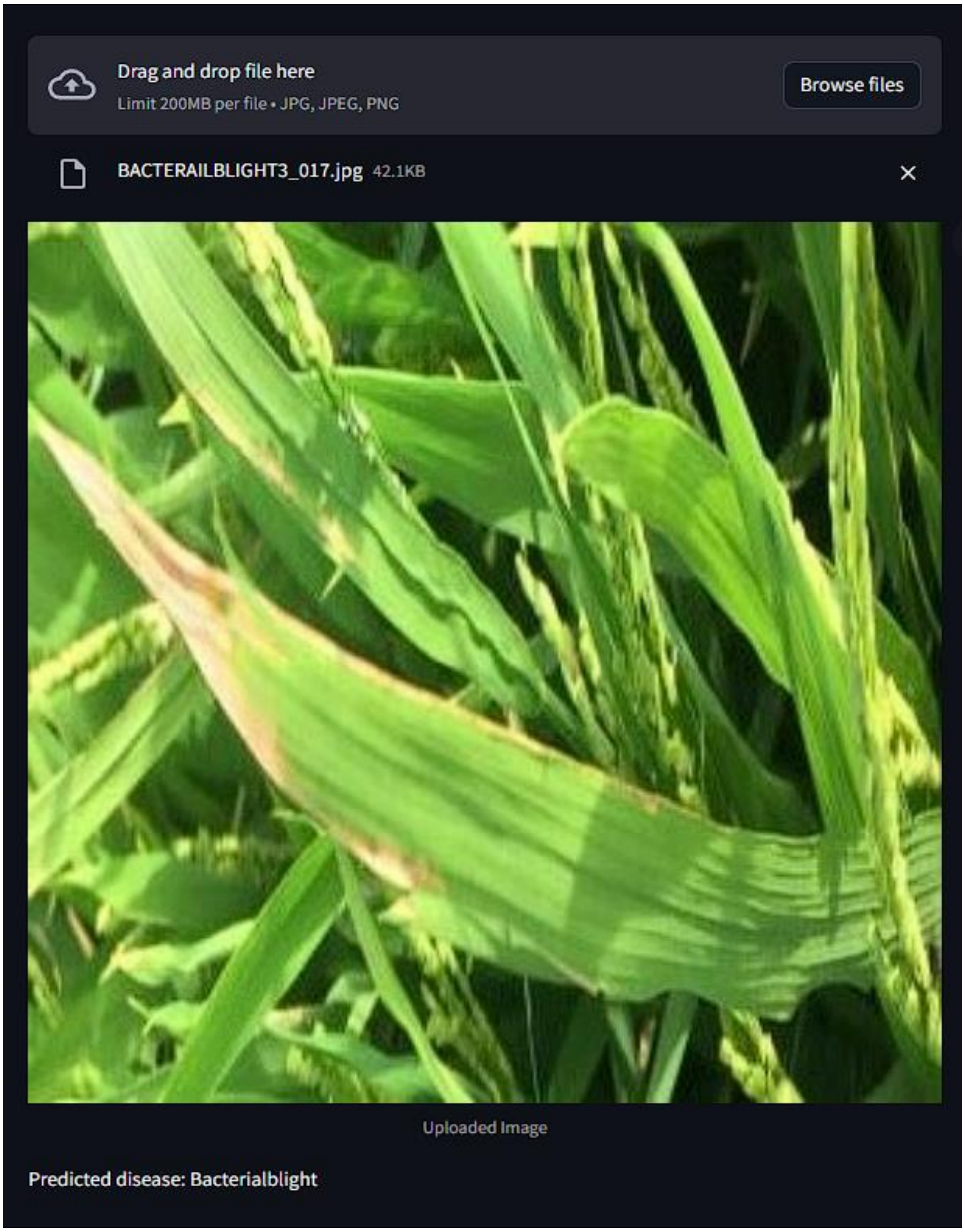
Prediction Page (for Tungro)



Predicted disease: Tungro

Figure 6.2.10: Prediction Page (for Tungro)

Prediction Page (for Bacterialblight)


Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files
BACTERAILBLIGHT3_017.jpg 42.1KB
Uploaded Image

Predicted disease: Bacterialblight

Figure 6.2.11: Prediction Page (for Bacterialblight)

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

Our project presents a significant advancement in the domain of agricultural disease detection, specifically focusing on diagnosing rice leaf diseases through the application of deep learning techniques. By leveraging sophisticated models such as Simple CNN, LeNet, ResNet, and Inception, we've achieved remarkable success in accurately classifying various types of rice leaf diseases. Let's highlight the key benefits:

- Accurate Disease Identification: Our models demonstrate exceptional proficiency in precisely identifying different types of rice leaf diseases, providing farmers with valuable insights for effective disease management.

- User-Friendly Interface: The intuitive interface ensures ease of use, making it accessible to farmers and agricultural experts for seamless integration into their workflows.

- Efficiency and Speed: With optimized algorithms and streamlined model architectures, our solution offers rapid disease detection, facilitating timely decision-making for crop protection.

- Scalability and Flexibility: Designed for scalability, our solution can accommodate large datasets and adapt to incorporate additional disease types, enhancing its versatility across diverse agricultural settings.

- Contributing to Agriculture: By harnessing the power of deep learning, our project contributes to the advancement of agricultural practices, promoting improved crop health and productivity.

In summary, our project underscores the transformative potential of deep learning in agriculture, empowering farmers with advanced tools for disease management and ultimately supporting the sustainability and resilience of rice cultivation worldwide.

7.2 FUTURE SCOPE

- Expansion of Disease Categories: Our project expands disease categories for better rice leaf disease detection. By refining deep learning models, we offer a comprehensive solution for identifying a wider range of crop ailments.
- Multimodal Data Integration: We integrate diverse data like satellite imagery and weather data to enrich diagnostics, providing deeper insights into environmental factors affecting disease occurrence.
- Integration with Precision Agriculture Technologies: Aligned with precision agriculture, our project integrates with drones, sensors, and IoT devices to collect real-time field data. This enables targeted interventions for managing disease outbreaks effectively.
- Real-Time Monitoring and Alerts: Implementing real-time monitoring enables instant alerts about disease outbreaks and crop health issues. By analyzing real-time data, our system provides timely recommendations for disease control, empowering proactive decision-making.

REFERENCES

[1] Trébuil, G., 2011. Rice production systems in Asia: The constant presence of an essential cereal on a continent in mutation.

[2] Papademetriou, M.K., 2000. Rice production in the AsiaPacific region: issues and perspectives. Bridging the rice yield gap in the Asia-Pacific region, 220.

[3] Bangladesh Rice Knowledge Bank, available at <>, last accessed on 25-11-2020, 08.40 am.

[4] Wikipedia, list of rice leaf diseases, available at <>, last accessed on 28-11-2020, 05.10 am.

[5] Mondal, M. H. (2010). Crop agriculture of Bangladesh: Challenges and opportunities. Bangladesh Journal of Agricultural Research, 35(2), 235- 245.

[6] Alamsyah D, Fachrurrozi M. 2019. Faster R-CNN with inception v2 for fingertip detection in homogenous background image. Journal of Physics: Conference Series 1196(1):12017'

[7] Arnal Barbedo JG. 2013. Digital image processing techniques for detecting, quantifying and classifying plant diseases. SpringerPlus 2(1):1-12

[8] R. R. Atole and D. Park, "A Multiclass Deep Convolutional Neural Network Classifier for Detection of Common Rice Plant Anomalies", International Journal Of Advanced Computer Science And Applications, vol. 9, no. 1, pp. 67-70, 2018.

[9].V. Singh and A. Misra, "Detection of Plant Leaf Diseases Using Image Segmentation and Soft Computing Techniques", Information Processing in Agriculture, vol. 4, no. 1, pp. 41-49, 2017.

[10]."Deep Learning Models for Plant Disease Detection and Diagnosis", Computers and Electronics in Agriculture, vol. 145, pp. 311-318, 2018.

[11].S. P. Mohanty, D. P. Hughes and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection", Frontiers in plant science, vol. 7.

[12].S. Xie, T. Yang, X. Wang and Y. Lin, "Hyper-class augmented and regularized deep learning for fine-grained image classification", IEEE Conference on Computer Vision and Pattern RecognitionCVPR2015, pp. 2645-2654, June 2015.

[13].Y. Lu, S. Yi, N. Zeng, Y. Liu and Y. Zhang, "Identification of rice diseases using deep convolutional neural networks", Neurocomputing, vol. 267, pp. 378-384, 2017.

[14].K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for LargeScale Image Recognition", pp. 1409-1556, 2015.