

A

PROJECT REPORT ON

**PADDY DISEASES RECOGNITION USING
CONVOLUTIONAL NEURAL NETWORK**

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



MASTER OF COMPUTER APPLICATIONS

By
Mr K. VINOD KUMAR,
(Regd.No:215N1F00D0)

Under the Guidance of
Mr C. SIVA KRISHNAIAH,
Assistant Professor, Dept of MCA,
APGCCS, Rajampet

&

Mr P. NARESH KUMAR,
Project Guide,
Manosys Technologies Pvt, Ltd., Bangalore



DEPARTMENT OF COMPUTER APPLICATIONS
ANNAMACHARYA P.G COLLEGE OF COMPUTER STUDIES
NEW BOYANAPALLI-516126, RAJAMPET (A.P).

(Approved by A.I.C.T.E., New Delhi & Affiliated to J.N.T.U.A,
Anantapur,UGC(2f)Recognized Institution)

2021-2023

**ANNAMACHARYA P.G COLLEGE OF COMPUTER STUDIES
NEW BOYANAPALLI, RAJAMPET - 516126.**



Affiliated to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, ANANTAPUR

**DEPARTMENT OF
MASTER OF COMPUTER APPLICATIONS**

CERTIFICATE

This is to certify that the project work entitled **“Paddy Diseases Recognition Using Convolutional Neural Network”** is the bonafide work carried out by **Mr K.Vinod Kumar** Regd. No: **215N1F00D0** is submitted in the partial fulfillment of the requirements for the award of degree of Master of Computer Applications during the year **2021-2023**.

Project Guide

Principal

External Examiner

DECLARATION

I, **K.Vinod Kumar**, hereby declare that the project report entitled as “**Paddy Diseases Recognition Using Convolutional Neural Network**” is done at, **Manosys Technologies Pvt, Ltd, Bangalore**, is original and independent record of work, submitted by me to JNTUA, Anantapur, under the guidance of **Mr C. Siva Krishnaiah**, Assistant Professor in Master of Computer Applications Department, **Annamacharya P.G College of Computer Studies**, Rajampet, for the award of the degree of **Master of Computer Applications** and has not been submitted either in part or in full for the award of any Degree or Diploma.

Place: Rajampet

Date:

K.Vinod Kumar,
(RegdNo:215N1F00D0)

ACKNOWLEDGEMENT

An endeavour over a long period can be successful only with the advice of many well wishers. I take this opportunity to express my deep gratitude and appreciation of all those who encourage me to successfully complete the mini project.

I wish to express my sincere gratitude to **Dr D J. Samatha Naidu**, Principal of Annamacharya PG College of Computer Studies, New Boyanapalli, Rajampet, for her consistent help and providing such facilities to complete this mini project.

I express my sincere thanks to my guide **Mr C. Siva Krishnaiah**, Assistant Professor for his valuable guidance and suggestions in analyzing and testing throughout the period of my mini project work.

I express my sincere thanks to **Mr P. NARESH KUMAR** for his valuable guidance and suggestions in analyzing and testing throughout the period of my mini project work.

Last but not least, I would like to thank my friends, teaching and non- teaching staff, one and all those who helped me to complete this mini project successfully.

k.vinod kumar,
(Regd. No:215N1F00D0)

ABSTRACT

A variety of diseases attack rice, one of India's most widely planted crops, at various phases in the growing process. It is exceedingly difficult for farmers with poor understanding to identify these diseases manually. Automated picture identification systems based on Convolutional Neural Network (CNN) models are showing considerable promise in deep learning recently. In deep learning model was created using Transfer Learning on a small dataset because it was difficult to find an image dataset of rice leaf disease. VGG-16 is utilized to train and assess the proposed CNN architecture, which is based on rice field and internet datasets. The proposed model has a 95 percent accuracy rate. Deep Learning, Convolutional Neural Network (CNN), fine-tuning and rice leaf diseases are some of the terms in this index.

CONTENTS

<u>Topics</u>	<u>Page No</u>
1. INTRODUCTION	01-05
1.1. Purpose	02
1.2. Scope	03
1.3. Need for System	04
2. SOFTWARE REQUIREMENT ANALYSIS AND SPECIFICATION	06-30
2.1. Related Work	06
2.2 Product Architecture	10
2.2. Product Functions	12
2.3. User Constraints	16
2.4. Hardware Requirements	19
2.5. Software Requirements	19
2.6. Non-Functional Requirements	19
3. SYSTEM DESIGN	31-54
3.1. Data Design	31
3.2. Data Dictionary	35
3.3. UML Design	37
4. TESTING	55-66
4.1. Testing Methodologies	58
4.2. Test Cases	65
5. IMPLEMENTATION	67-94
5.1. Sample Screens	79
CONCLUSIONS	

BIBLIOGRAPHY

Appendix-A

- URL Listing
- References

Appendix – B

- Glossary

Appendix – C

- List of Tables
- List of Figures
- List of Screens

Appendix – D

- Coding

1.INTRODUCTION

Paddy (rice) is a primary staple crop that serves as a major food source for a significant portion of the global population. However, paddy crops are susceptible to various diseases caused by pathogens, fungi, bacteria, and viruses, among other factors. These diseases can cause significant damage to the crop, resulting in reduced yield and quality, and pose a significant threat to food security.

Traditionally, the identification and diagnosis of paddy diseases have relied on manual inspection by agricultural experts, which can be time-consuming, labor-intensive, and prone to human error. With the rapid advancements in computer vision and deep learning techniques, there is a growing interest in developing automated systems for disease recognition in crops.

Convolutional Neural Networks (CNNs) have emerged as a powerful deep learning approach for image classification tasks. CNNs are specifically designed to extract and learn intricate patterns and features from images, making them well-suited for recognizing complex patterns in paddy disease images.

The objective is to propose a method for paddy disease recognition using CNNs. The primary goal is to leverage the capabilities of CNNs to automatically extract discriminative features from paddy disease images and accurately classify them into different disease categories. By employing this automated approach, the aim is to provide farmers and agricultural experts with a reliable tool to detect and diagnose paddy diseases in a timely and efficient manner.

The proposed system will have several advantages over traditional manual inspection methods. It will enable rapid and accurate disease identification, allowing for prompt intervention measures to mitigate the spread of diseases and minimize crop losses. Additionally, it will reduce the dependence on

human expertise and offer a cost-effective solution for disease monitoring and management.

To achieve the goal of paddy disease recognition, a comprehensive dataset containing a diverse range of paddy disease images, including both healthy and diseased plants, will be collected and curated. This dataset will serve as the foundation for training and evaluating the performance of the CNN model. Various techniques such as data augmentation, regularization, and hyperparameter optimization will be employed to enhance the model's accuracy and generalization ability.

The success will contribute to the development of an automated paddy disease recognition system that can be deployed as a practical tool for farmers and agricultural stakeholders. It will empower them to make informed decisions regarding disease control and management, leading to improved crop productivity, reduced economic losses, and ultimately, sustainable agriculture.

1.1 Purpose

The purpose of paddy disease recognition using Convolutional Neural Networks (CNNs) is to develop an automated system that can accurately identify and classify diseases affecting paddy crops. The key objectives and purposes of this are as follows:

1. **Early Disease Detection:** The timely identification of paddy diseases is crucial for implementing appropriate disease control measures. By leveraging CNNs, the proposed system aims to detect diseases at an early stage, allowing farmers to take prompt action and mitigate the spread of diseases.
2. **Accurate Disease Diagnosis:** CNNs have the ability to learn and extract complex features from images, enabling them to identify subtle disease symptoms that may be challenging to detect with the naked eye. The purpose

is to achieve high accuracy in disease recognition and provide precise diagnosis to farmers and agricultural experts.

3. Automated and Efficient Process: The proposed system aims to automate the process of paddy disease recognition, reducing the dependency on manual inspection by experts. This automation leads to increased efficiency, as large volumes of images can be processed quickly, enabling timely decision-making and intervention.

4. Support for Farmers and Agricultural Stakeholders: By providing an automated disease recognition system, the purpose is to support farmers and agricultural stakeholders in managing paddy diseases effectively. The system can serve as a practical tool for farmers, enabling them to identify diseases without the need for specialized knowledge and experience.

5. Improved Crop Productivity and Sustainability: Effective disease recognition and management contribute to improved crop productivity and sustainability. By accurately identifying and diagnosing diseases, farmers can implement targeted control measures, minimize crop losses, and optimize resource utilization.

The purpose of using Convolutional Neural Network (CNN) for detecting paddy diseases is to develop an automated system that can accurately classify and diagnose various diseases affecting paddy crops. CNN are a class of deep learning models that are particularly well-suited for image recognition tasks, making them suitable for analyzing plant diseases patterns.

1.2 Scope

The scope of paddy disease recognition using Convolutional Neural Networks (CNNs) encompasses various aspects related to the development and application of the automated system. The key scopes are as follows:

1. **Disease Coverage:** The system aims to recognize and classify a wide range of paddy diseases caused by pathogens, fungi, bacteria, viruses, or other factors. The scope includes both common and rare diseases that affect paddy crops globally.
2. **Image Dataset:** Involves the collection and curation of a comprehensive dataset of paddy disease images. The dataset should cover diverse disease symptoms, including different stages of disease progression, variations in leaf damage, and multiple paddy varieties.
3. **Model Development:** The scope includes designing and developing a CNN model specifically tailored for paddy disease recognition. The model architecture, hyperparameters, and training techniques will be optimized to achieve high accuracy in disease classification
4. **Training and Evaluation:** The proposed system will involve training the CNN model using the collected dataset. The training process will include techniques such as data augmentation, regularization, and hyperparameter optimization. The model's performance will be evaluated using appropriate metrics to assess its accuracy and generalization ability.
5. **Disease Identification:** The system aims to accurately identify and classify paddy diseases based on input images. It should provide the disease label or a probability distribution over disease classes to indicate the presence and type of disease in the given sample.

1.3 Need for System

Existing System

In the past, the only way to identify a disease was to manually examine the leaf. The illness was discovered using a combination of visual inspection of plant leaves and consulting a reference book. This method has three key drawbacks: limited accuracy, the inability to analyse every leaf, and a lengthy time investment. As science and technology progress, new methods for

accurately diagnosing these diseases emerge. Image processing and deep learning are two methods to consider. Filtering, clustering and histogram analysis are some of the approaches used in image processing to identify the diseased area. Deep learning neural networks, on the other hand, are used to detect disorders.

Disadvantages

- ❖ It's impossible to study every leaf, and it takes a long time.
- ❖ This method has three key drawbacks limited accuracy, the inability to analysis every leaf, and a lengthy time investment

Proposed System

The proposed system aims to develop an efficient and accurate method for paddy disease recognition using Convolutional Neural Networks (CNNs). The system will consist of the following components and functionalities.

A VGG16 transfer learning neural network is to train a dataset of rice diseases, and the trained model may be used to predict disease from new photos. Because the Rice Leaf dataset from it was too tiny for author to train the VGG16 model, he turned to the transfer learning CNN algorithm, which transfers an existing CNN model to a new dataset and then uses the new data to train the model. It has been shown that VGG16 transfer learning improves prediction accuracy in both a normal CNN model and a normal CNN model with VGG16 transfer learning.

Advantages

- ❖ VGG16 can be utilize to identify rice leaf diseases. Over ninety-five percent of the time, it has been right.
- ❖ The plant diseases can be identified at early stage or the initial stage.
- ❖ It is highly accuracy and highly efficiency

2. SOFTWARE REQUIREMENT ANALYSIS AND SPECIFICATION

2.1 Related Work

Many studies have used classical classifiers; however, the outcomes are dependent on the feature selection approaches, and picture preprocessing is a significant stage in the process. There are numerous studies that make use of CNN's high recognition accuracy.

Detection of Plant Diseases using CNN is Based Training uses 87,848 photos from 25 plant species and the 58 groups they fall under, including examples of "healthy" plant growth. One of the best models has a 99.53% success rate in correctly classifying objects. 54306 photos of 14 different crop kinds, each with 26 different illnesses and healthy leaves, were used to train CNN. When evaluated on a different dataset derived from a real-world event, the 99.35 percent success rate dropped to 31.4%. We'll talk about why determining the severity of a condition is more difficult than just classifying it. Because of the high levels of intra-class resemblance across photos belonging to the same class, it is even more difficult to identify them.

The use of CNN to detect rice disease was based on a convolutional neural network trained on 227 photos of healthy and damaged rice plants. AlexNet is used to train the classifier. 91.23% of the time, the aforementioned architecture is able to accurately forecast whether or not a plant is unhealthy. The authors gathered 500 photos of ten distinct leaf and stem rice illnesses. LeNet and AlexNet were used as inspiration for the design of its architecture, which was tested and achieved 95.48% accuracy. Pre-processing steps, including image resizing to 512*512, normalisation, PCA, and whitening, were performed because the data was so sparse. Instead of using maximum pooling, they opted for stochastic pooling, which they claimed prevented overfitting. Several related works have been conducted in the field of paddy disease recognition using Convolutional Neural Networks (CNNs). Here are a few notable examples:

AUTHORS: V. Singh, A. Misra

Lack of infrastructure makes it difficult to detect agricultural diseases rapidly, despite their considerable threat to global food security. Because of the extensive usage of smartphones around the world and recent improvements in computer vision made possible by deep learning, disease diagnosis by smartphone is becoming a reality. We train a convolutional neural network to recognise 14 crop species and 26 illnesses from a publicly available dataset using 54,306 pictures of damaged and healthy plant leaves (or absence thereof). Using a held-out test set, the trained model achieves an accuracy of 99.35%, demonstrating the validity of the approach. Using a set of shots that were not taken in the same conditions as those used for training, the model's accuracy is just 31.4 percent the total accuracy can still be improved by using a more diversified set of training data, though. For crop disease diagnosis on a global scale, training deep learning models on increasingly large and publicly available image datasets provides a clear path.

Title: SVM classifier based grape leaf disease detection**AUTHORS: P. B. Pedal and A. A. Yadav**

Grapes are one of India's most popular fruit crops. Grape yields fall as a result of disease infections on the fruit, stems, and leaves. Toxins from bacteria, fungus or virus are the most common causes. There are many factors that limit the amount of fruit that can be produced, including illness. It's impossible to implement effective control measures without an accurate diagnosis of the disease. For the identification and categorization of plant leaf diseases, image processing is a commonly employed technique. SVM classification is used in this work to aid in the detection and categorization of grape leaf diseases. Segmentation by K-means clustering is used to locate the sick area, and then colour and texture features are extracted. Leaf disease can finally be identified using a classification system. Accuracy in the suggested approach is 88.89 percent for the condition being studied.

Title: Very Deep Convolutional Networks for Large Scale Image Recognition**AUTHORS: Karen Simonyan, Andrew Zisserman**

Using a large-scale image identification task, we investigate how a convolutional network's depth influences its performance. Using an architecture with extremely small (3x3) convolution filters, we examined networks of increasing depth and found that raising the depth to 16-19 weight layers significantly improved performance over prior-art setups. This is the main contribution. As a result of these discoveries, our team was able to take first and second place in the 2014 ImageNet Challenge in the categories of localization and classification. A wide range of datasets can benefit from our representations' state-of-the-art results. Two of our finest Conv Net models have been provided so that further into deep visual representations in computer vision can take place.

Title: Hyper class augmented and regularized deep learning for fine-grained image classification**AUTHORS: S. Xie, T. Yang, X. Wang, and Y. Lin**

Deep convolutional neural networks have shown significant success in large-scale object recognition (CNNs). Due to the high cost of fine-grained labelled data (usually necessitating domain expertise), as well as the huge intra-class and moderate inter-class variance, fine-grained image classification (FGIC) is significantly more challenging than generic object identification. Using an external dataset (like ImageNet) to pre-train the CNN and then fine-tuning it on the small target data to meet a specific classification goal is one of the most prevalent ways. Two new aspects of the problem of learning a deep CNN are introduced identifying easily annotated hyper-classes inherent in the fine-grained data and collecting numerous images with hyper-classes labelled from readily available external sources (such as image search engines), formulating the problem as multitask learning; and (iii) a novel learning modality, which is based on the idea of a "multitask learning" paradigm. Two small, fine-grained datasets (Stanford

Dogs and Stanford Cars) as well as a big, automotive-specific dataset have all been used to evaluate the proposed approach.

Title: Detection of Plant Leaf Diseases Using Image Segmentation and Soft Computing Techniques

AUTHORS: V. Singh, A. Misra,

The economy heavily relies on agricultural productivity. It's for this reason, as well as the fact that diseases in plants are relatively common, that disease detection in the agricultural sector is so crucial. Neglecting this region can have detrimental impacts on plants, which in turn impair the quality, quantity, and productivity of the final output. For example, in the United States, pine trees are susceptible to a dangerous illness known as small leaf disease. The early detection of disease symptoms, such as those that occur on plant leaves, is an advantage of using an automated system for detecting plant disease. This method minimises the amount of time needed to monitor huge farms of crops. For the automatic identification and classification of plant leaf diseases, provides an algorithm for the picture segmentation technique. A review of various disease classification strategies for plant leaf disease detection is also included. Genetic algorithm is used for image segmentation, which is critical for disease identification in plant leaf disease.

2.2 Product Architecture

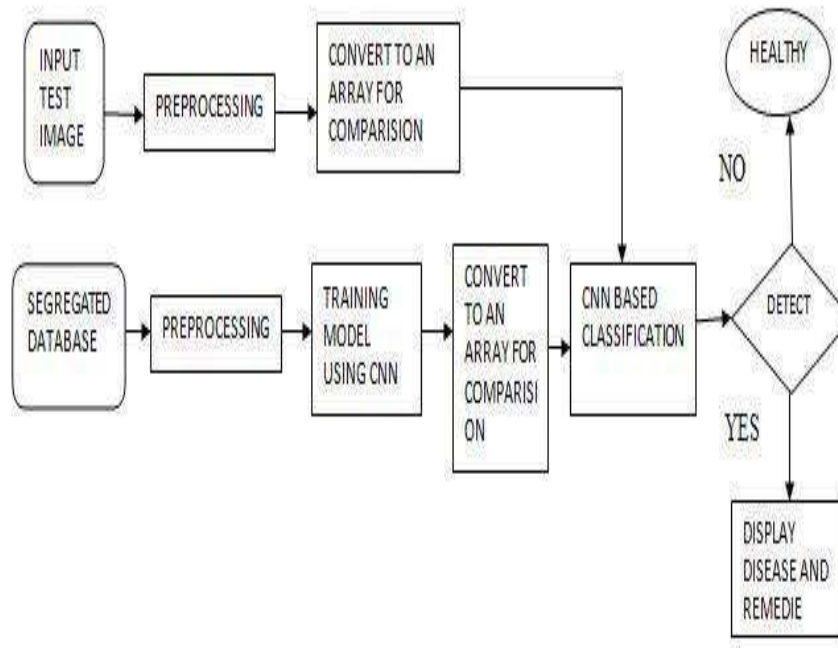


Fig 2.2.1: System Architecture

1.Dataset Collection: Gather a diverse and representative dataset of paddy disease images. Ensure that the dataset includes various types of diseases, as well as healthy paddy plant images for comparison. Label each image with the corresponding disease category.

2.Dataset Preprocessing: Preprocess the dataset to ensure consistency and improve model performance. Common preprocessing steps include resizing images to a uniform size, normalizing pixel values, and augmenting the dataset through techniques like rotation, scaling, and flipping to increase variability.

Dataset Split: Divide the dataset into training, validation, and testing sets. The training set is used to train the CNN model, the validation set is used for hyperparameter tuning, and the testing set is used to evaluate the final model's performance.

3.Model Architecture: Design the CNN architecture for disease recognition.

A typical CNN architecture consists of alternating convolutional layers, pooling layers, and fully connected layers. You can use popular CNN architectures like VGG Net, Resnet, or Inception as a starting point and customize them according to your specific needs.

4.Training: Train the CNN model on the training set. During training, the model learns to extract meaningful features from the paddy disease images and classify them into respective disease categories. Use a suitable optimization algorithm like Adam or RMSprop and an appropriate loss function such as categorical cross-entropy.

5.Hyperparameter Tuning: Adjust the hyperparameters of the CNN model to optimize its performance on the validation set. Experiment with parameters like learning rate, batch size, number of layers, filter sizes, and pooling strategies to achieve better results. You can use techniques like grid search or random search to find optimal hyperparameters.

6.Evaluation: Evaluate the trained model on the testing set to assess its performance and generalization ability. Calculate metrics such as accuracy, precision, recall, and F1 score to measure the model's effectiveness in identifying paddy diseases.

7.Fine-tuning and Regularization: If the model is not performing satisfactorily, you can consider techniques like fine-tuning the pre-trained CNN models on a larger dataset or applying regularization techniques such as dropout or L2 regularization to mitigate overfitting.

8.Deployment: Once you have a satisfactory model, you can deploy it in real-world applications. This could involve creating a user-friendly interface or integrating the model into an existing system for automated disease recognition in paddy crops.

Remember that paddy disease recognition using CNN architecture requires a sufficiently large and diverse dataset, proper preprocessing techniques, careful model design, and thorough evaluation to ensure accurate and reliable results.

2.3 Product Function

This application consists following modules.

1. Database
2. Image pre-processing
3. CNN classification
4. Comparison of test and train images

1.Database

An image database means storing high quantities of digital images in a particular location. It also means organizing photos so that they can be shared, accessed quickly and easily. In this project we have taken large number of paddy leaf images into datasets.

2.Image pre-processing

Image processing is a method used to perform basic operations like extraction and conversion on an image or a video, in place to get an enhanced image by getting the different features and values from it. This is a form of signal processing in which an image or set of images is taken as an input and then the result we get may be in the form of image or features associated with that image. Image processing is of two types and they are analogue and digital image processing. Analogue image processing is ideally used in the fields of photography and printing Digital image processing techniques help in shaping of the digital images by using digital sources and technologies. The three main steps of the digital technique process are pre-processing, enhancement, and restoration.

3. CNN Classification

A CNN is a Higher-Level language of deep learning. CNN models are trained using large collections of images taken across different sources. From these large collections of databases CNN models can learn rich feature extraction and presentation for a wide range of images. CNN is considered as one of the best techniques used for features extraction. After the segmentation of image into clusters the CNN classification of these clusters and we get different images like black and white image, query image and also the segmented image. So once these images are formed the area affected more by the disease can be easily visible and then can be tested properly and accurately.

4. Comparison of test and train images

The final stage of the process is finding whether the plant is healthy or not healthy. If there are any defects found in the image then the output will be given as yes along with the remedies to be followed for eradicating the disease. This happens when the both test and train images are brought together and are compared. The accuracy of the problem found is more when this is executed.

Algorithm

Convolutional Neural Network

In the fast-paced world of computer vision and image processing, one problem consistently stands out: the ability to effectively recognize and classify images. As we continue to digitize and automate our world, the demand for systems that can understand and interpret visual data is growing at an unprecedented rate. The challenge is not just about recognizing images it's about doing so accurately and efficiently. Traditional machine learning methods often fall short, struggling to handle the complexity and high dimensionality of image data. This is where Convolutional Neural Networks (CNNs).

The CNN architectures are the most popular deep learning framework. CNNs shown remarkable success in tackling the problem of image recognition, bringing a newfound level of precision and scalability. But not all CNNs are created equal, and understanding the different types of CNN architectures is key to leveraging their full potential.

Convolutional Neural Networks, commonly referred to as CNNs, are a specialized kind of neural network architecture that is designed to process data with a grid-like topology. This makes them particularly well-suited for dealing with spatial and temporal data, like images and videos, that maintain a high degree of correlation between adjacent elements.

CNNs are similar to other neural networks, but they have an added layer of complexity due to the fact that they use a series of convolutional layers. Convolutional layers a mathematical operation called convolution, a sort of specialized matrix multiplication, on the input data. The convolution operation helps to preserve the spatial relationship between pixels by learning image features using small squares of input data. . The picture below represents a typical CNN architecture.

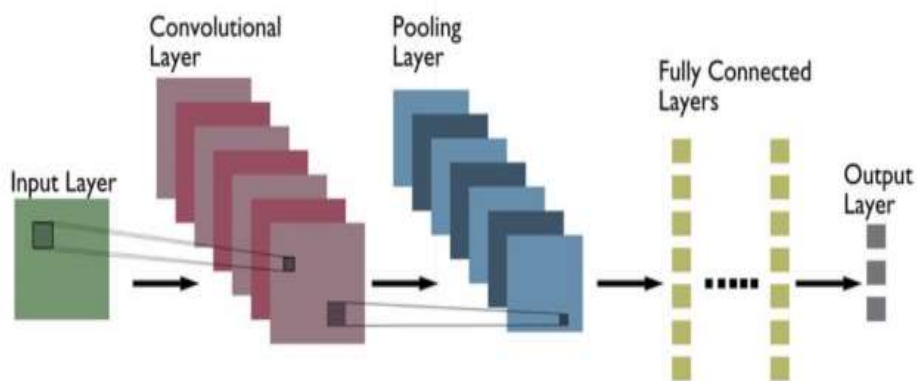


Fig 2.3.1: CNN Architecture

The following are definitions of different layers shown in the above architecture

1.Input Layers: It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).

2.Convolutional layer: Convolutional layers operate by sliding a set of 'filters' or 'kernels' across the input data. Each filter is designed to detect a specific feature or pattern, such as edges, corners, or more complex shapes in the case of deeper layers. As these filters move across the image, they generate a map that signifies the areas where those features were found. The output of the convolutional layer is a feature map, which is a representation of the input image with the filters applied. Convolutional layers can be stacked to create more complex models, which can learn more intricate features from images.

3.Pooling layer: Pooling layers follow the convolutional layers and are used to reduce the spatial size of the input, making it easier to process and requiring less memory. By reducing the dimensions, pooling layers help reduce the number of parameters or weights in the network. This helps to combat overfitting and help train the model in a fast manner. There are two main types of pooling: max pooling and average pooling. Max pooling takes the maximum value from each feature map. For example, if the pooling window size is 2×2 , it will pick the pixel with the highest value in that 2×2 region. Max pooling effectively captures the most prominent feature or characteristic within the pooling window. Average pooling calculates the average of all values within the pooling window. It provides a smooth, average feature representation.

4.Fully connected layer: Fully-connected layers are one of the most basic types of layers in a convolutional neural network (CNN). As the name suggests, each neuron in a fully-connected layer is Fully connected- to every other neuron in the previous layer. Fully connected layers are typically used towards the end of a CNN- when the goal is to take the features learned by the convolutional and pooling layers and use them to make predictions such

as classifying the input to a label. For example, if we were using a CNN to classify images of animals, the final Fully connected layer might take the features learned by the previous layers and use them to classify an image as containing a dog, cat, bird, etc.

CNNs are often used for image recognition and classification tasks. For example, CNNs can be used to identify objects in an image or to classify an image as being a cat or a dog. CNNs can also be used for more complex tasks, such as generating descriptions of an image or identifying the points of interest in an image. Beyond image data, CNNs can also handle time-series data, such as audio data or even text data, although other types of networks like Recurrent Neural Networks (RNNs) or transformers are often preferred for these scenarios. CNNs are a powerful tool for deep learning, and they have been used to achieve state-of-the-art results in many different applications.

5.Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or SoftMax which converts the output of each class into the probability score of each class.

2.4 User Constraints

A constraint in project management is any restriction that limits project's desired outcome. Project constraint is one of the important factors that would influence the way you manage the project and, in some cases, it would be a determinant factor to decide whether to continue the project or not. Project limitations can fall under different categories and it is critical to identify and understand each of them that affect your project. This way, you can focus your analysis on those limitations and substantially increase your preparedness to deal with them.

It is a common misconception that the project management constraints are internal i.e., limitations due to internal factors associated to that particular project. Though internal limitations such as project scope or cost play a major

role in defining the project constraints, some external factors such as environment, external stakeholders can also impose restrictions or limitations that would result as project constraints.

Triple constraints of Project Management

Anyone with even rudimentary knowledge on project management concepts has probably heard about the well-known “Triple Constraint” which is also Known as the project management try angle.

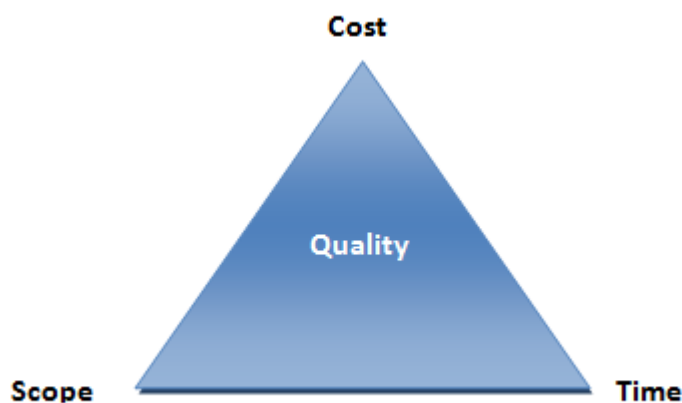


Fig 2.4.1: Triple Management Triangle

As shown in the picture, all projects are executed under these constraints – which are Cost, Time and Scope. If you look at each of these constraints in isolation, every project manager’s aim is to deliver the project,

- Within allocated or budgeted cost.
- Within stipulated time-frame.
- As per the agreed scope.

Quality

Quality would be the central theme of any project and it should meet the client requirements that are set at the beginning of the project. Cost, time and scope are linked to one another and any change/compromise on one constraint would affect one or all other project constraints.

Cost

Every project has a fixed budget that the sponsors are ready to spend in order to get goods or services. If the sponsors cut down the budget, you will have to either extend the timeline for project delivery or reduce the project scope. This is the estimation of amount of money required to produce the final deliverable. Monetary components of various aspects of the project need to be estimated in advance which will be added up to the overall cost of the given project.

Time

Schedule or time-frame of a project is normally fixed based on various factors such as customer requirements, project feasibility, resource availability etc,. Hence, every project has a deadline which is predetermined at the beginning of the project. When a project time gets reduced, either project cost needs to be increased or the scope of the project should be cut down in order to accommodate a shorter deadline. Basically, the amount of the time required to finish the project is directly related to the amount of requirements that need to be part of end product (Scope) and the amount of resources assigned to the project.

Scope

It is common across many projects that the scope is fully refined, communicated or understood by the stakeholders at the beginning of the project in order to give the project the best chance of success. However, the scope can potentially be changed by the customer during the project life cycle (known as Scope Creep). Scope is the functional elements of the project that define the end deliverable's capability. If the scope of the project is increased at the later stage, either cost or time needs to be increased accordingly.

Project Constraints List

- Scope – Project outcome as defined in the contract.
- Timeline – Important milestones and client-imposed completion dates.
- Budget – Funding limits imposed by project sponsors.

- Quality – Agreed quality metrics with Customer or conformance with internal quality standards.
- Resources – Availability of skilled human resources or materials.
- Risks – Uncertainties associated with the project.

2.5 Hardware Requirements

- ❖ System : i3 or above
- ❖ Ram : 4GB.
- ❖ Hard disk : 1TB

2.6 Software Requirements

- ❖ Operating system : Windows
- ❖ Coding Language : python
- ❖ Front end : HTML, CSS, JAVA SCRIPT
- ❖ Back end : MYSQL

2.7 Non-Functional Requirements

Non-functional requirements describe user-visible aspects of the system that are not directly related to functionality of the system. Non-functional requirements these are constraints on the services or functions offered by the System.

Non-functional requirements are often called qualities of a system. Other terms for non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non-behavioral requirements". Qualities, that are non-functional requirements, can be divided into two main categories:

Execution qualities, such as security and usability, which are observable at run time. "Quality of service requirements" and "non-behavioral requirements". Qualities, that are non-functional requirements, can be divided into two main categories:

1. Execution qualities, such as security and usability, which are observable at run time.
2. qualities, such as testability, maintainability, extensibility and scalability, which are embodied Evolution in the static structure of the software system.

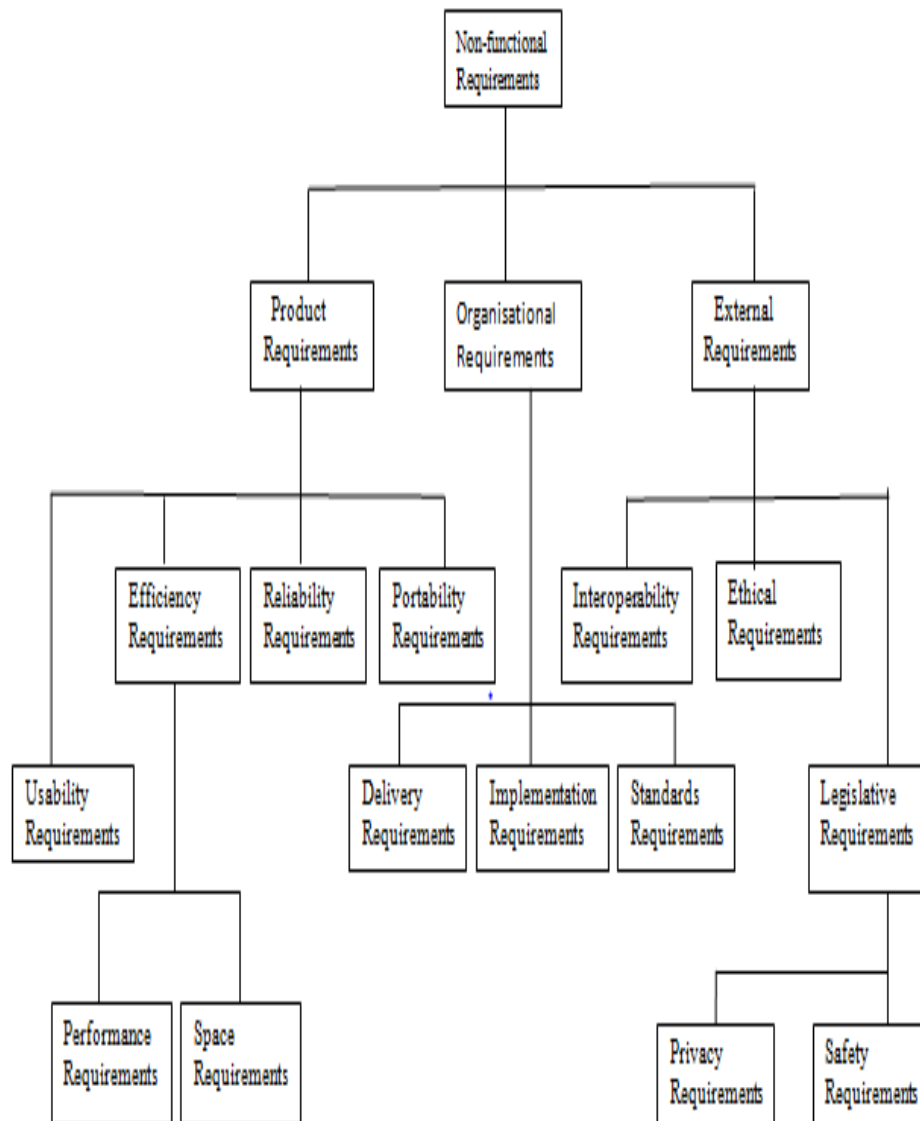


Fig 2.7.1: Non-Functional Requirements

Reliability

The packages will pick-up current transactions online. Regarding the old transactions, user will enter them in to the system.

Security

The web server and database server should be protected from hacking, virus etc.

Portability

The application will be developed using standard open-source software (Except Oracle) like Java, tomcat web server, Internet Explorer Browser etc. this software will work both on Windows and Linux OS. Hence portability problems will not arise.

Maintainability

The first tier is the GUI, which is said to be front-end and the second tier is the database, which uses My-Sql, which is the back-end. The front-end can be run on different systems (clients). The database will be running at the server. Users access these forms by using the user-ids and the passwords.

Robustness

Time to restart after failure percentage of events causing failure probability of data corruption on failure.

Usability

The system is used by the four persons namely Administrator, Project Manager, Developer and the customer. Each person is having their own roles and separated by the security issues.

Performance

System is highly functional and good in performance. The system must use the minimal set of variables and minimal usage of the control structures will dynamically increase the performance of the system.

Availability

The system is implemented based on the web browser and server. Using this web browser, the user can access the data and store the data in the server, here we can use the web browser as Mozilla and server as Tomcat5.5. And windows XP professional is used as the platform.

Supportability

The system is designed to be the cross platform supportable. The System is supported on a wide range of hardware and any software platform.

Testability

The system software can be tested. Operability is the better it works, the more efficiency it can be tested. It operates clearly. Observability is what you see is what you test. The results of each test case are readily observer.

Accessibility

It is a general term used to describe the degree to which a product, device, service, or environment is available to as many people as possible. Accessibility can be viewed as the "ability to access" and benefit from some system or entity. Accessibility is often used to focus on people with disabilities or special needs and their right of access to entities, often through use of assistive technology. Accessibility is not to be confused with usability which is used to describe the extent to which a product (e.g., device, service, and environment) can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

Scalability

Scalability is the ability of a system, network, or process, to handle growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth. For example, it can refer to the capability of a system to increase total throughput under an increased load when resources (typically hardware) are added. An analogous meaning is implied when the word is used in a commercial context, where scalability of a company implies that the underlying business model offers the potential for economic growth within the company. Scalability, as a property of systems, is generally difficult to define and in any particular case it is necessary to define the specific requirements for scalability on those dimensions that are deemed important. It is a highly significant issue in electronics systems, databases, routers, and networking.

Software Development Life Cycle (SDLC)

SDLC is nothing but Software Development Life Cycle. It is a standard which is used by software industry to develop good software.

Stages in SDLC

- Requirement Gathering
- Analysis
- Designing
- Coding
- Testing
- Maintenance

Requirements Gathering Stage

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed as well as mission-critical inputs, outputs, and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a requirement." Requirements are identified by unique requirement identifiers and, at a minimum, contain a requirement title and textual description.

These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are not included in the requirements document.

The title of each requirement is also placed in the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

- A feasibility study is all about identifying problems in a project.
- The number of staff required to handle a project is represented as team formation. In this case, only modules with individual tasks will be assigned to employees who are working on that project.
- Project Specifications are all about representing various possible inputs submitted to the server and corresponding outputs, along with reports maintained by the administrator.

Analysis Stage

The planning stage establishes a bird's-eye view of the intended software product and uses this to establish the basic project structure, evaluate the feasibility and risks associated with the project, and describe appropriate management and technical approaches.

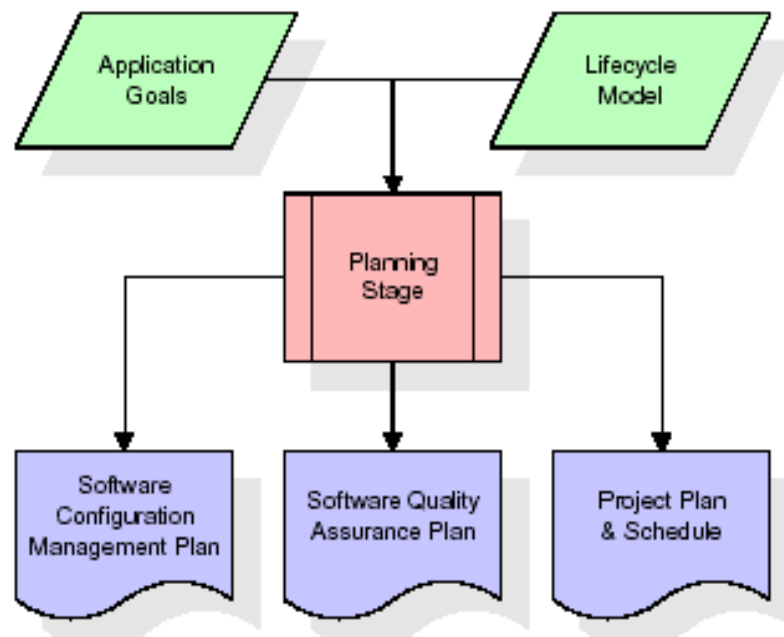


Fig 2.7.2: Analysis Model

The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage and high-level estimates of effort for the outgoing stages.

Designing Stage

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail and generally include functional hierarchy diagrams, screen layout

diagrams, tables of business rules, business process diagrams, pseudocode, and a complete entity-relationship diagram with a full data dictionary.

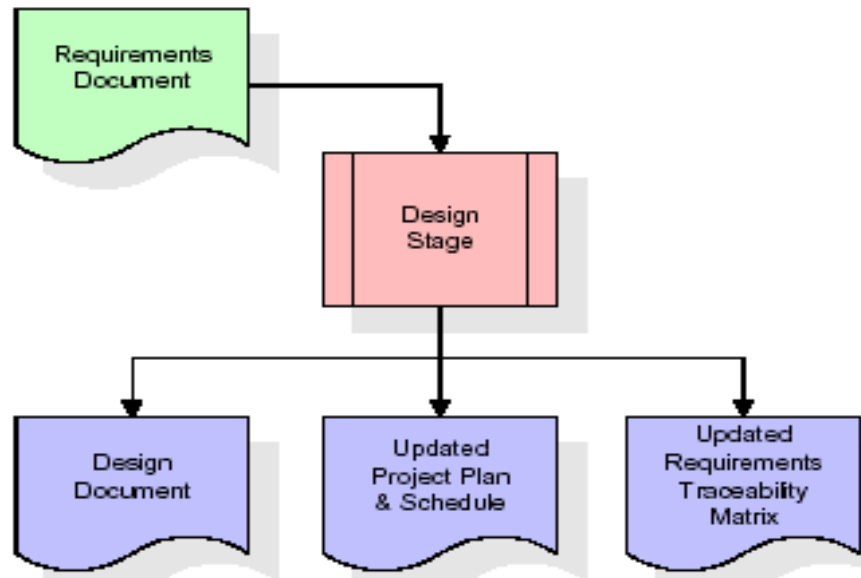


Fig 2.7.3: Designing Stage

When the design document is finalised and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

Development (Coding) Stage

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artefacts will be produced. Software artefacts include, but are not limited to, menus, dialogues, data management forms, data reporting formats, and specialised procedures and functions. Appropriate test cases will be developed for each set of functionally related software artefacts, and an online help system will be developed to guide users in their interactions with the software.

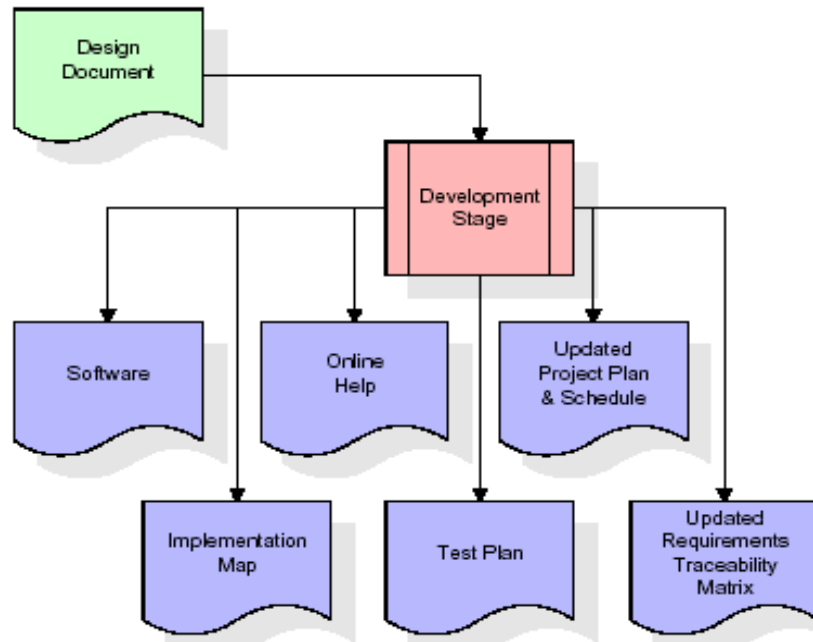


Fig 2.7.4: Development Stage

The RTM will be updated to show that each developed artefact is linked to a specific design element and that each developed artefact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

Integration and Test Stage

During the integration and test stages, the software artefacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability Initiation Plan.

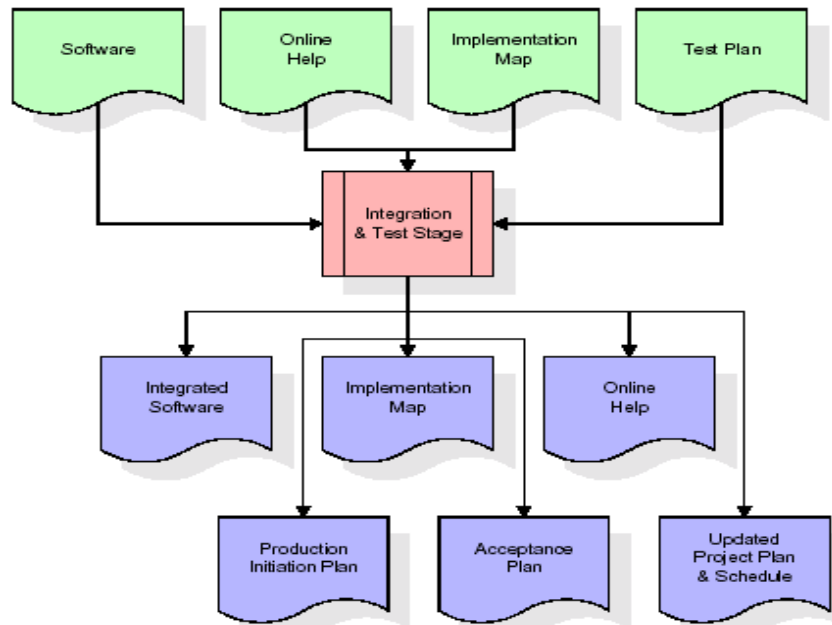


Fig 2.7.5: Integration & Test Stage

The outputs of the integration and test stages include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan that contains the final suite of test cases, and an updated project plan.

Installation and Acceptance Test

During the installation and acceptance stages, the software artefacts, online help, and initial production data are loaded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer. After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.

The primary outputs of the installation and acceptance stages include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR

enters the last of the actual labour data into the project schedule and locks the project as a permanent project record. At this point, the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

Maintenance

The outer rectangle represents maintenance of a project. The maintenance team will start with a requirement and understanding of documentation. Later, employees will be assigned work, and they will undergo training on that particular assigned category. For this life cycle, there is no end; it will be continued so on like an umbrella.

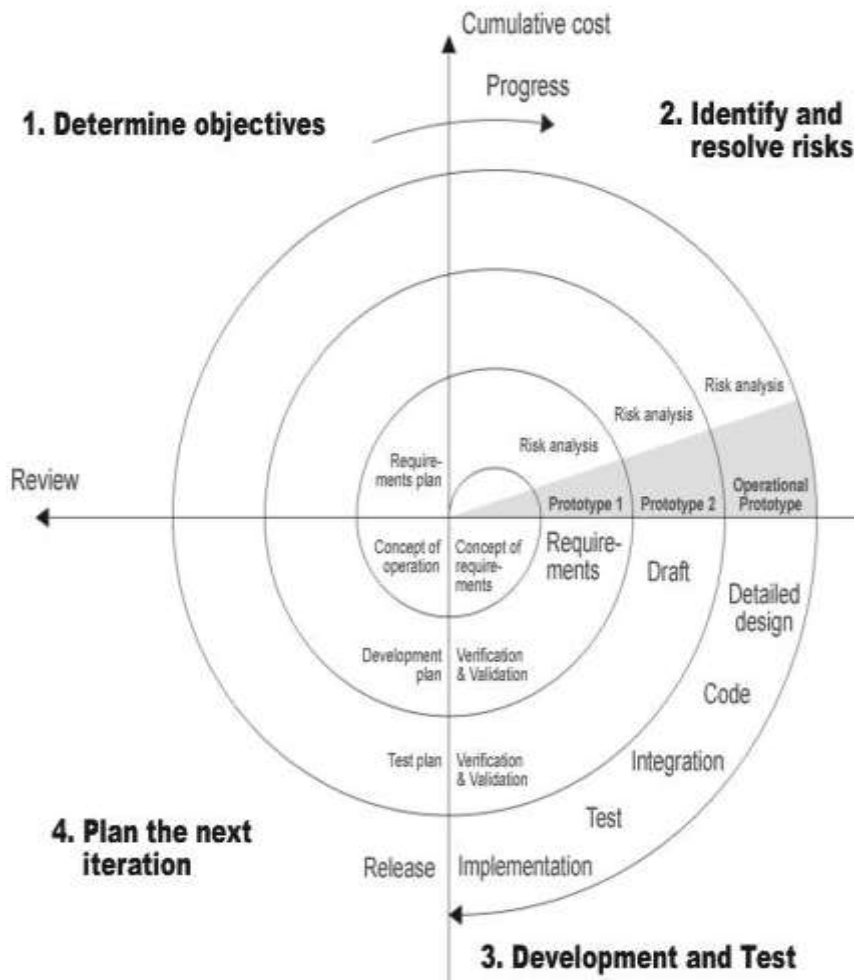


Fig 2.7.6: Spiral Model

Advantages

- ❖ Risk analysis done here reduces the scope of risk occurrence.
- ❖ Any requirement change can be accommodated in the next iteration.
- ❖ Model is good for large projects that are prone to risks and the requirement keeps on changing.

3. SYSTEM DESIGN

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement has been specified and analyses, system design is the first of the three technical activities design, code and test that is required to build and verify software.

The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system one that will be difficult to test, one whose quality cannot be assessed until the last stage. During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities like architectural design, data structure design, interface design and procedural design.

3.1 Data Design

A graphical tool used to describe and analyze the movement of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also known as a data flow graph or a bubble chart.

DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system's structure charts. The Basic Notation used to create a DFD's are as follows:

1. **Dataflow:** Data move in a specific direction from an origin to a destination.

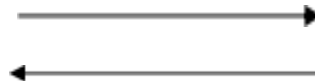


Fig 3.1.1: Dataflow

2. **Process:** People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.



Fig 3.1.2: Process

3. **Source:** External sources or destination of data, which may be People, programs, organizations or other entities.



Fig 3.1.3: Source

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap and synergy with the disciplines of systems analysis, systems architecture and systems engineering.

Database Designing is a part of the development process. In the linear development cycle, it is used during the system requirements phase to construct the data components of the analysis model. This model represents the major data objects and the relationship between them. It should not be confused with data analysis, which takes place in the system design phase. As in a DFD, a model of data consists of several symbols joined up according to certain conventions. System designers describe these conceptual modeling using symbols from a modeling method known as entity relationship analysis.

Entity Relationship Diagram

Entity relationship analysis uses three major abstractions to describe data. These are

1. Entities, which are distinct things in the enterprise.
2. Relationships, which are meaningful interactions between the objects.
3. Attributes, which are the properties of the entities and relationship.

The relative simplicity and pictorial clarity of this diagramming technique may well account in large part for the widespread use of ER model. Such a diagram consists of the following major components.

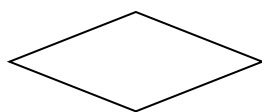
ER Diagram Components



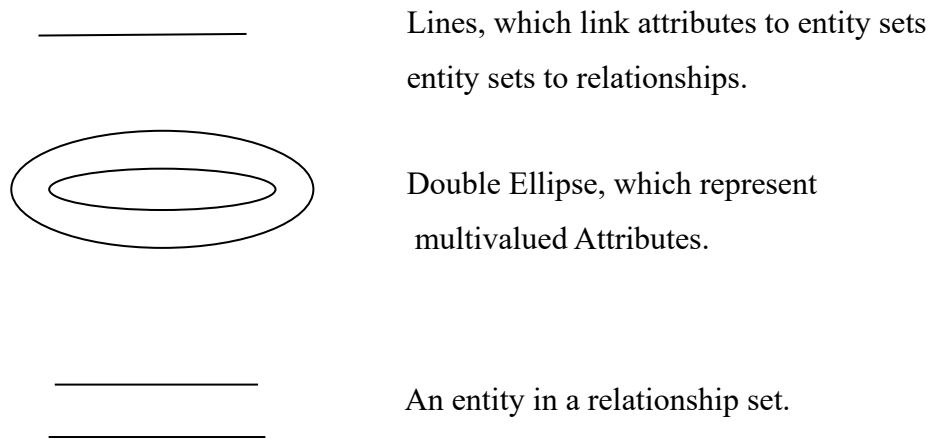
Rectangles, which represent the entity set.



Ellipse, which represent attributes.



Diamonds, which represent relationship.



Entity

- An entity is an object that exists and is distinguishable from other objects.
- An entity may be concrete or abstract.
- An entity is a set of entities of the same type.
- Entity sets need not be disjoint.
- An entity is represented by a set of attributes.

Mapping Constraints

An E-R diagram may define certain constraints which the contents of a database must conform.

Mapping Cardinalities

It expresses the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality must be one of the following:

One-to-One – An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

One-to-many – An entity in A is associated with any number in B. An entity in B is associated with any number in A.

Many-to-many – Entities in A and B are associated with any number from each other.

ER Diagram

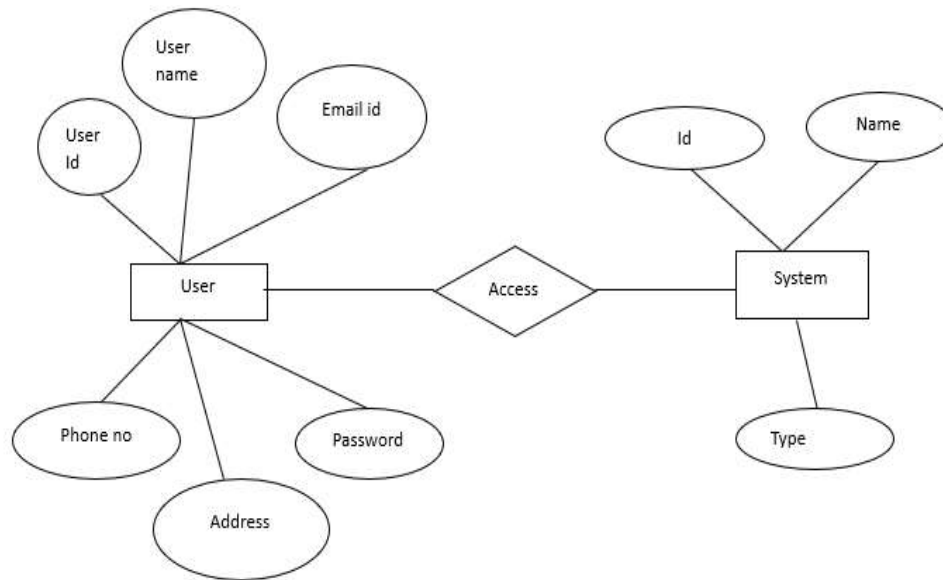


Fig 3.1.4: ER Diagram

3.2 Data Dictionary

The logical characteristics of current systems data stores, including name, description, aliases, contents, and organization, identifies processes where the data are used and where immediate access to information required, serves as the basis for identifying database requirements during system design.

Uses of Data Dictionary

- ❖ To manage the details in large systems.
- ❖ To communicate a common meaning for all system elements.
- ❖ To Document the features of the system.
- ❖ To facilitate analysis of the details in order to evaluate characteristics and determine where system changes should be made.
- ❖ To locate errors and omissions in the system.

Data Dictionary

Number of database tables: The project has been identified to contain 2 database tables, which are as follows.

Table Name: User

Description: User Details defines how the users are login to the system and what they can do in the system.

Column Name	Data Type (Size)	Constraints
User Name	Varchar2(30)	NOTNULL
Password	Number (10)	NOTNULL
Contact	Number (10)	NOTNULL
Address	Varchar2(30)	NOTNULL
Email id	Varchar2(30)	NOTNULL

Table 3.2.1: User

Table Name: System

Description: User Can Access System Using Id, Name, and Type

Column Name	Data Type (Size)	Constraints
Id	Number (10)	PRIMARY
Name	Varchar2(30)	NOTNULL
Type	Varchar2(30)	NOTNULL

Table 3.2.2: System

3.3 UML Design

To understand the UML, you need to form a conceptual model of the language, and this requires learning three major elements the UML's basic building blocks, the rules that dictate how these building blocks may be put together, and some common mechanisms that apply throughout the UML. Once you have grasped these ideas, you will be able to read UML models and create some basic ones. As you gain more experience in applying the UML, you can build on this conceptual model, using more advanced features of the language.

Basic Building Blocks of the UML

The vocabulary of the UML encompasses three kinds of building blocks

1. Things
2. Relationships
3. Diagrams

Things in the UML

Things are the abstractions that are first class citizens in a model, relationships tie these things together, and diagrams group interesting collections of things. These things are the basic object-oriented building blocks of the UML. You use them to write well-formed models. There are four kinds of things in the UML

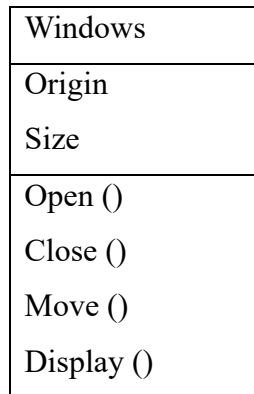
1. Structural Things
2. Behavioral Things
3. Grouping Things
4. An notational Things

Structural Things

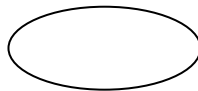
Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

Class

First, a class is a description of a set of objects that share the same attributes, operations, relationships and semantics. A class implements one or more interfaces graphically, a class is rendered as a rectangle, usually including its, name, attributes, and operation.

**Fig 3.3.1: Class****Interface**

Second, an interface is a collection of operations that specify a service of a class or component. An interface therefore describes the externally visible behavior of that element. An interface might represent the complete behavior of a class or component or only a part of that behavior. An interface defines a set of operation specifications. Graphically, an interface is rendered as a circle together with its name. An interface is rendered rarely stands alone. Rather, it is typically attached to the class or component that realizes the interface.

**Fig 3.3.2: Interface****Collaboration**

Third, collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements. Therefore, collaborations have

structural, as well as behavioral, dimensions. A given class might participate in several collaborations these collaborations therefore represent the implementation of patterns that make up a system. Graphically, collaboration is rendered as an ellipse with dashed lines, usually including only its name.



Fig 3.3.3: Collaboration

Use Case

Fourth, a use case is a description of set of sequence of actions that a system performs that yields an observable result of value to a actor. A use case is used to structure the behavioral things in a model. A use case is realized by collaboration. Graphically use case is rendered as an ellipse with solid lines, usually including only its name.

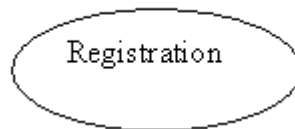


Fig 3.3.4: Use Case

The remaining three things active classes, components, and nodes are all class like meaning they also describe a set of objects that share the same attributes, operations, relationships, and semantics. However, these three are different enough and are necessary for modeling certain aspects of an object-oriented system, and so they warrant special treatment.

Active Class

Fifth, an active class is a class whose objects own one or more processes or threads and therefore can initiate control activity. An active class is just like a class except that its objects represent elements whose behavior is concurrent with other elements. Graphically, an active class is rendered just

like a class, but with heavy lines, usually including its name, attributes, and operations. The remaining two elements component and nodes are also different. They represent physical things, whereas the previous five things represent conceptual or things.

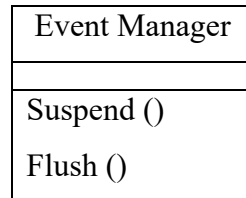


Fig 3.3.5: Active Class

Component

Sixth, a component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. In a system, you'll encounter different kinds of deployment components, such as COM+ component or Java Beans, as well as components that are artifacts of the development process, such as source code files. A component typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations. Graphically a component is rendered as a rectangle with tabs, usually including only its name.

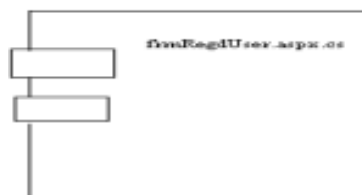


Fig 3.3.6: Component

Node

Seventh, a node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability. A set of components may reside on a node and may reside on a node and may also migrate from node to node. Graphically, a node is rendered as a cube, usually including only its name.

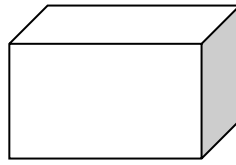


Fig 3.3.7: Node

These seven elements- classes, interfaces, collaborations, use cases; active classes, components, and nodes are the basic structural things that you may include in a UML model. There are also variations on these seven, such as actors, signals, and utilities (kinds of classes), processes and threads (kinds of active classes), and applications, documents, files, libraries, pages, and tables (kinds of components).

Behavioral Things

Behavioral things are the dynamic parts of UML models. These are the verbs of a model representing behavior over time and space. In all, there are two primary kinds of behavioral things.

First, an interaction is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a specific purpose. The behavior of a society of objects or if an individual operation may be specified with an interaction. An interaction involves several other elements, including messages, action sequences (the behavior invoked by a message), and links (the connection between objects). Graphically message is rendered as a directed line, almost always including the name of its operation.



Fig 3.3.8: Message

Second, a state machine is a behavior that specifies the sequence of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events the behavior of an individual class

or a collaboration of classes may be specified with a state machine. A state machine involves several other elements, including states, transitions (the flow from state to state), events (things that trigger a transition), and activities (the response to a transition). Graphically, a state is rendered as a rounded rectangle, usually including its name and its sub states.

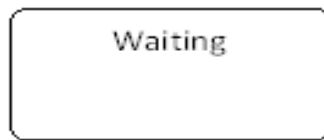


Fig 3.3.9: State

These two elements' interactions and state machines are the basic behavior things that you may include in a UML model. Semantically, these elements are usually connected to various structural elements, primarily classes, collaborations and objects.

Grouping Things

Grouping things are the original parts of UML models. These are the boxes into which a model can be decomposed. In all, there is one primary kind of grouping thing, namely, packages.

Package

A package is a general-purpose mechanism for organizing elements into groups. Structural things, behavioral things, and even other grouping things may be placed in a package. Unlike components (which exist at run time), a package is purely conceptual (meaning that exist only development time). Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents. Packages are the basic grouping things with which you may organize a UML model there are also variations, such as frameworks, models, and subsystems (kinds of packages).

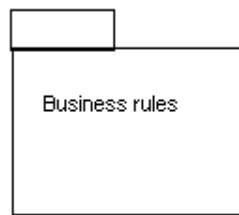


Fig 3.3.10: Package

Annotational Things

Annotational things are the explanatory parts of UML model. These are the comments you may apply to describe, illuminate, and remark about any element in a model. There is one primary kind of a notational thing, called a note. A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements. Graphically note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment.

Note

This element is the one basic a notational thing you may include in a UML model. You'll typically use notes to adorn your diagrams with constraints or comments that are best expressed in informal or formal text. There are also variations on this element, such as requirements (which specify some desired behavior from the perspective of outside the model).

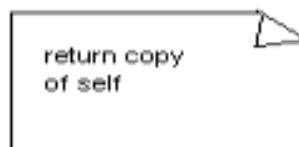


Fig 3.3.11: Note

Relationships in the UML

These relationships are the basic relational building blocks of the UML. You use them to write well-formed models. There are four kinds of relationships in the UML.

1. Dependency
2. Association
3. Generalization
4. Realization

Dependency

First, a dependency is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing). Graphically dependency is rendered as a dashed line, possibly directed, and occasionally including a label.

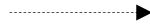


Fig 3.3.12: Dependency

Association

Second, an association is a structural relationship that describes a set of links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts. Graphically, an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names.



Fig 3.3.13: Association

Generalization

Third, a generalization is a specialization/generalization relationship in which objects of the specialized element are substitutable for objects of the generalized element (the parent). In this way, the child shares the structure and the behavior of the parent. Graphically a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent.

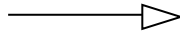


Fig 3.3.14: Generalization

Realization

Fourth, a realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. You'll encounter realization relationships in two places between interfaces and the classes or components that realize them and between use cases and the collaborations that realize them. Graphically, a realization relationship is rendered as a cross between a generalization and a dependency relationship.

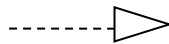


Fig 3.3.15: Realization

These four elements are the basic relational things you may include in a UML model. There are also variations on these four, such as refinement, trace, include, and extended (for dependencies). UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagrams which is as follows.

User Model View

This view represents the system from the user's perspective. The analysis representation describes a usage scenario from the end-user's perspective.

Structural Model View

In this model the data and functionality are arrived from inside the system. This model view models the static structures.

Behavioral Model View

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

Implementation Model View

In this structural and behavioral as parts of the system are represented as they are to be built.

Environmental Model View

In this the structural and behavioral aspect of the environment in which the system is to be implemented are represented.

Data Flow Diagram

- ❖ The DFD can also be referred to as bubble charts. For example, a simple visual formalism can portray a system in terms of the input and output of data.
- ❖ Second, the data flow diagram is a crucial modelling tool (DFD). Models the system's components. System processes, data, an external entity interacting with the system, and information flow within the system are all included in this list.
- ❖ The DFD displays how information flows through the system and how it is transformed through a sequence of transformations. It's a visual representation of how data goes from input to output and the transformations that take place along the way.
- ❖ DFD, or bubble chart, is a type of graph. A DFD can be used at any degree of abstraction to represent a given system. To reflect increasing information flow and functional detail, DFD might be partitioned into a number of tiers or layers.

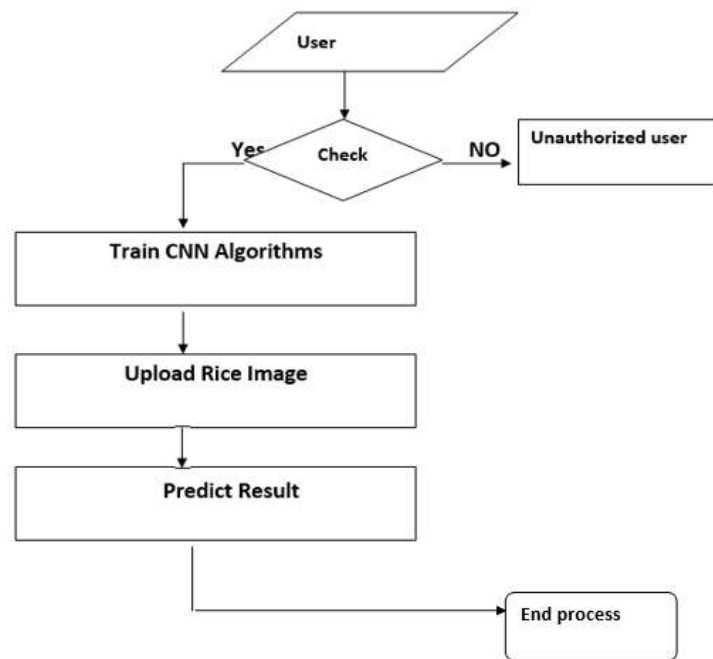


Fig 3.3.16 Data Flow Diagram

Diagrams in the UML

A diagram is the graphical presentation of a set of elements most often rendered as a connected graph of vertices (things) and arcs (relationships). You draw diagrams to visualize a system from different perspectives, so a diagram represents an elided view of the elements that make up a system. The same element may appear in all diagrams, only a few diagrams, or in no diagrams at all. In theory, a diagram may contain any combination of things and relationships. In practice, however, a small number of common combinations arise, which are consistent with the five most useful views that comprise the architecture of a software intensive system. For this reason, the UML includes nine such diagrams.

Use Case Diagram

Use Case diagrams are one of the five diagrams in the UML for modeling the dynamic aspects of systems activity diagrams, sequence diagrams, state chart diagrams and collaboration diagrams are the four other kinds of diagrams in the UML for modeling the dynamic aspects of systems. Use Case diagrams are central to modeling the behavior of the system, a sub-

system, or a class. Each one shows a set of use cases and actors and relationships.

Class Diagram

Class diagrams are the most common diagrams found in modeling object-oriented systems. A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Graphically, a class diagram is a collection of vertices and arcs.

Interaction Diagrams

An Interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. Interaction diagrams are used for modeling the dynamic aspects of the system.

Activity Diagram

An Activity Diagram is essentially a flow chart showing flow of control from activity to activity. They are used to model the dynamic aspects of a system. They can also be used to model the flow of an object as it moves from state to state at different points in the flow of control.

State Chart Diagram

A state chart diagram shows a state machine. State chart diagrams are used to model the dynamic aspects of the system. For the most part this involves modeling the behavior of the reactive objects. A reactive object is one whose behavior is best characterized by its response to events dispatched from outside its context.

Component Diagram

The component diagrams for the selected component package, along with new this list always contains an entry for the main component diagram for the component package.

Deployment Diagram

A deployment diagram shows processors, devices, and connections. Each model contains a single deployment diagram which shows the connections between its processors and devices, and the allocation of its processes to processors.

Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

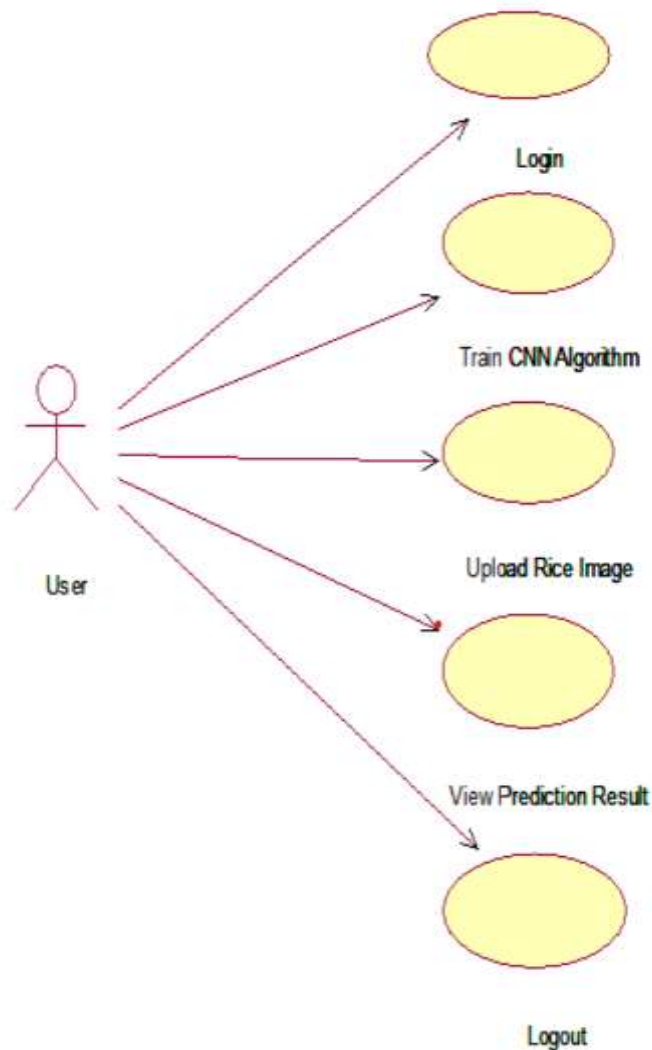


Fig 3.3.17: Use Case Diagram for Overall System

- 1.Login:** The Login Page Allows A User To Access A Website Or Web Application By Entering Their Username And Password.
- 2.Train CNN algorithm:** The user can log in and train the CNN algorithm.
- 3.Upload Rice Image:** User Can Upload Rice Image.
- 4.View Prediction Result:** User Can view the prediction accuracy results after uploading the rice disease image. Either Predict Paddy Leaf Diseases Brown Spot, Healthy.
- 5.Logout:** User logout the system.

Class Diagram

The UML class diagram also referred to as object modeling, is the main static analysis diagram. Object modeling is the process by which the logical objects in the problem space are represented by the actual objects in the program. These diagrams show a set of classes, interfaces, and collaborations and their relationships. These diagrams address the static design view of a system. This view primarily supports the functional requirements of a system – the services the system should provide to its end users. A class diagram is a collection of static modeling elements, such as classes and their relationships, connected as a graph to each other and to their contents.

Class diagrams commonly contain the following things:

- Classes
- Interfaces
- Collaborations
- Dependency, Generalization and association relationships

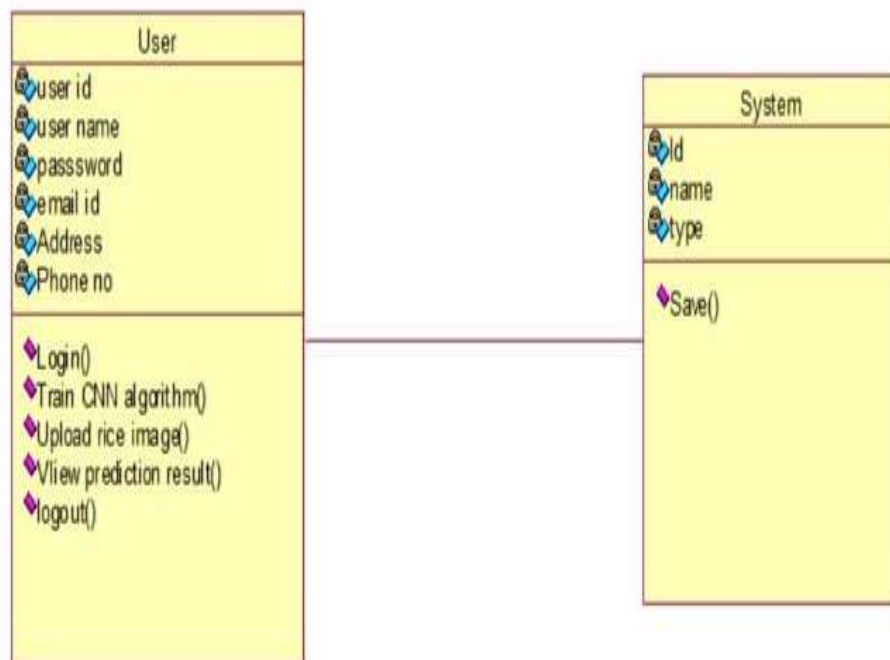


Fig 3.3.18: Class Diagram for Overall System

Sequence Diagram

A sequence diagram is a visual representation of the system's interactions. A sequence diagram's most crucial feature is that it's time-ordered. In other words, the interactions between the items are represented step by step in this model. Messages are the means by which various elements of the sequence diagram communicate with one another.

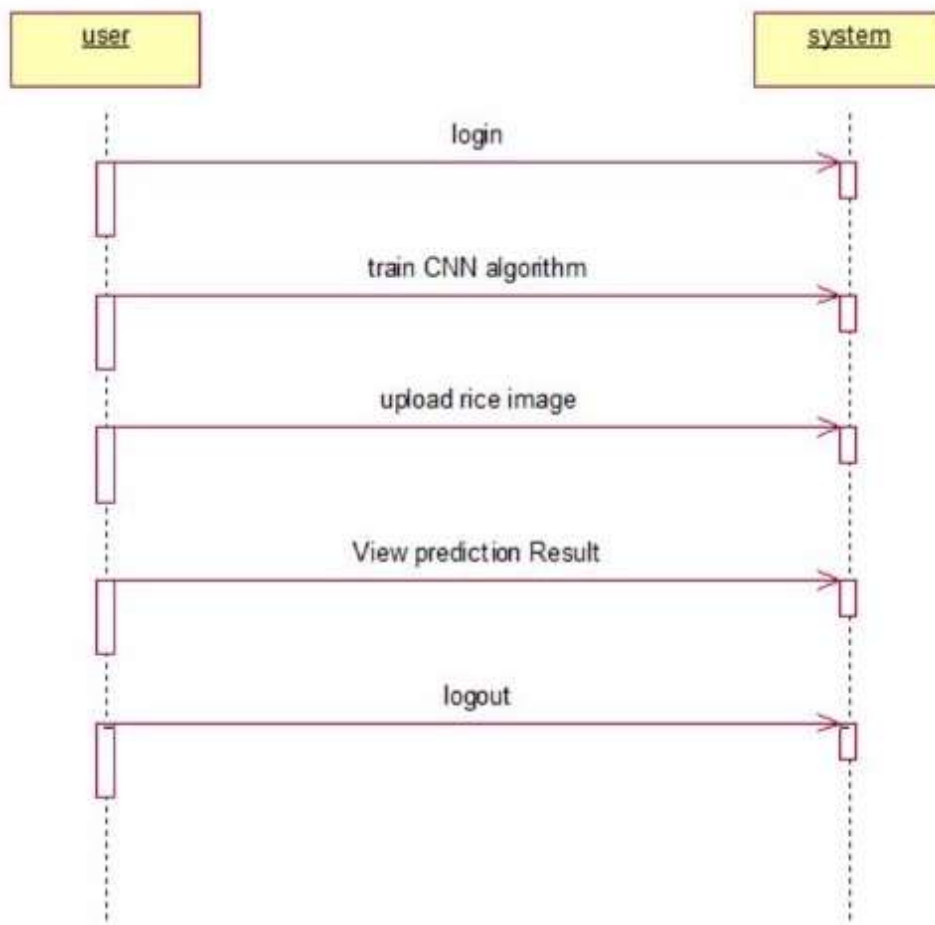


Fig 3.3.19: Sequence Diagram for Overall System

Activity Diagram

An activity diagrams illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special kind of state chart diagram, it uses some of the same modeling conventions.

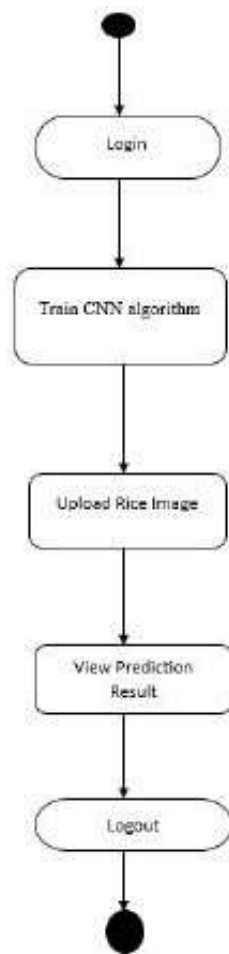


Fig 3.3.20: Activity Diagram for User

Collaboration diagram:

These relationships are depicted in a collaboration diagram. A numbering system aids in the tracking of the interactions' order in the database. Using the cooperation diagram, it is possible to see all the ways in which one object can interact with the other.

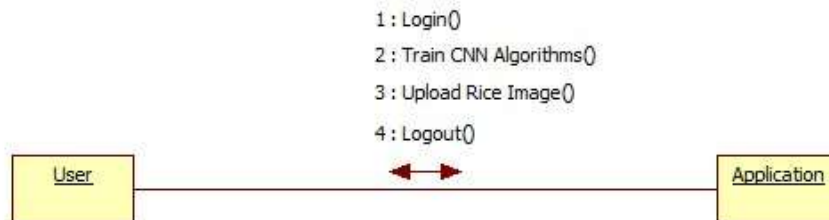


Fig 3.3.21: Collaboration Diagram for Upload Files

Deployment Diagram

A deployment diagram shows processors, devices, and connections. Each model contains a single deployment diagram which shows the connections between its processors and devices, and the allocation of its processes to processors.

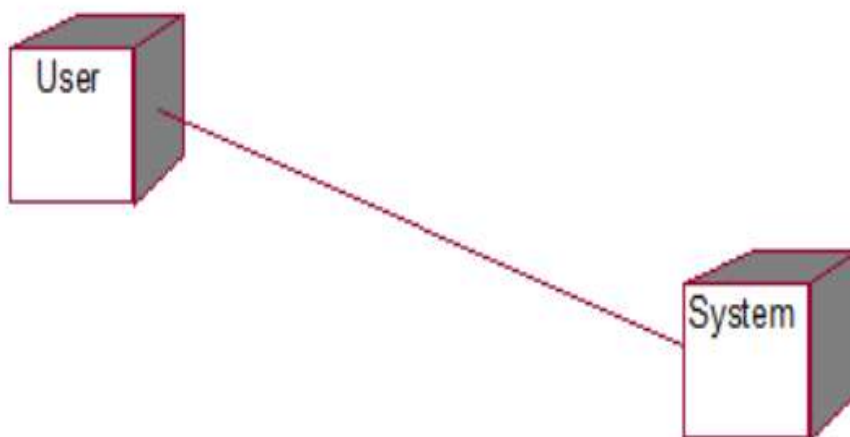


Fig 3.3.22: Deployment Diagram

4.TESTING

Testing is one of the most important phases in the software development activity. In software development life cycle (SDLC), the main aim of testing process is the quality; the developed software is tested against attaining the required functionality and performance.

During the testing process the software is worked with some particular test cases and the output of the test cases are analyzed whether the software is working according to the expectations or not.

Levels of Testing

Since the errors in the software can be injured at any stage. So, we have to carry out the testing process at different levels during the development. The basic levels of testing are Unit, Integration, System and Acceptance Testing.

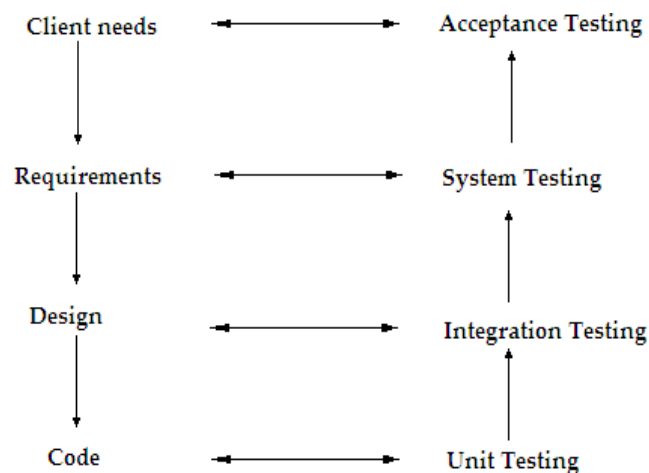
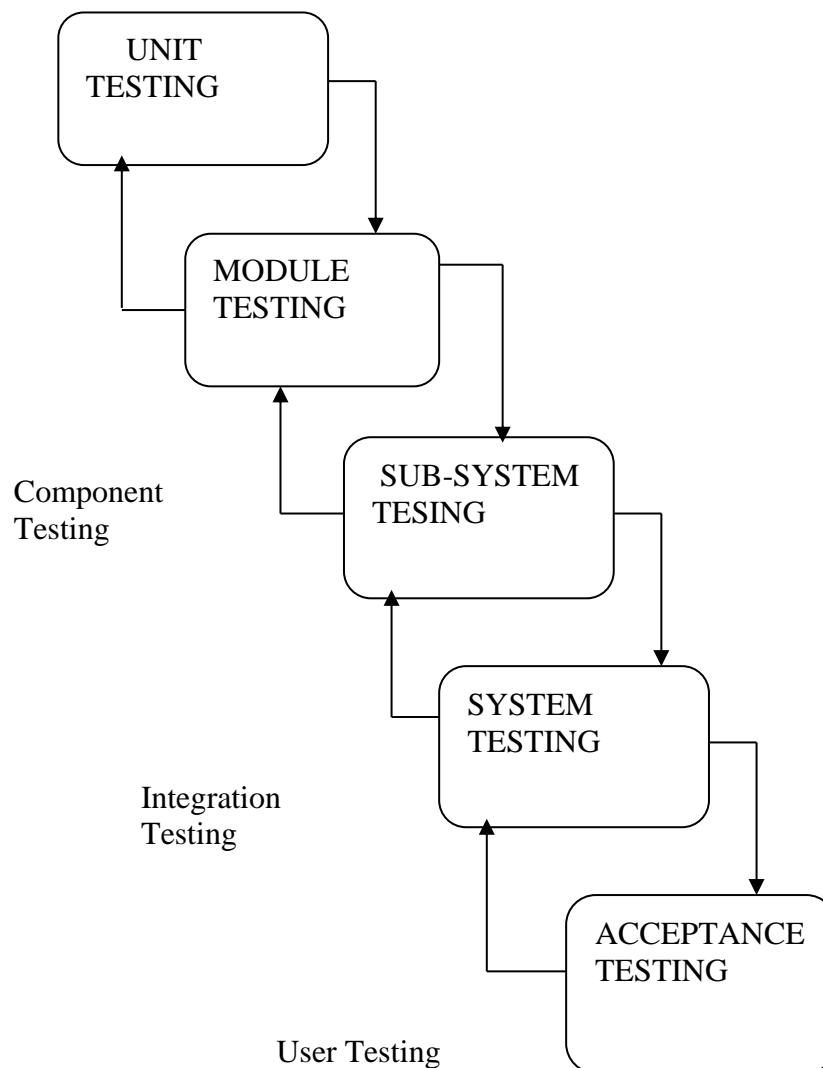


Fig 4.1: Levels of Testing

The Unit Testing is carried out on coding. Here different modules are tested against the specifications produced during design for the modules. In case of integration testing different tested modules are combined into subsystems and tested in case of the system testing the full software is tested and in the next level of testing the system is tested with user requirement.

**Fig 4.2 Testing Methodologies****Unit Testing**

Unit testing focuses verification efforts on the smallest unit of software design, the module. The unit testing, we have is white box-oriented, and in some modules, the steps are conducted in parallel.

Test Plan

A test plan is a general document for the entire project that defines the scope, approach to be taken, and personnel responsible for different activities of testing. The inputs for forming the test plane are

- Project plan
- Requirements document
- System design

Test Case Specification

Although there is one test plan for the entire project, test cases have to be specified separately for each test case. A test case specification is given for each item to be tested. All test cases and outputs expected for those test cases

Test Case Execution and Analysis

The steps to be performed for executing the test cases are specified in a separate document called the test procedure specification. This document specifies any specific requirements that exist for setting the test environment and describes the methods and formats for reporting the results of testing.

Test Approach

Testing can be done in two ways.

- bottom-up Approach
- top-down Approach

Bottom-up Approach

Testing can be performed starting with the smallest and lowest-level modules and proceeding one at a time. For each module in bottom-up testing, a short programmer executes the module and provides the needed data so that the module is asked to perform the way it will when embedded within the larger system. When bottom-level modules are tested, attention turns to those on the next level that use the lower-level ones; they are tested individually and then linked with the previously examined lower-level modules.

Top-down Approach

This type of testing starts with upper-level modules. Since the detailed activities usually performed in the lower-level routines are not provided, stubs are written. A stub is a module shell called by an upper-level module

that, when reached properly, will return a message to the calling module indicating that proper interaction occurred. No attempt is made to verify the correctness of the lower-level module.

4.1 Testing Methodologies

Black Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black box testing. The tester is unaware of the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are processed. An approach to testing where the programmer is considered a 'black box' The programmed test cases are based on the system specification, which is the testing process in which testers can perform testing on an application without having any internal structural knowledge of the application.

Advantages	Disadvantages
Well suited and efficient for large code segments.	Limited Coverage since only a selected number of test scenarios are actually performed.
Code Access not required.	Inefficient testing, due to the fact that the tester only has limited knowledge about an application.

Table 4.1.1: Black Box Testing

White Box Testing

Is the testing process in which testers can perform testing on an application with internal structural knowledge. All independent paths have been exercised at least once. All logical decisions have been exercised on their true and false sides. All loops are executed at their boundaries and within their operational bounds. All internal data structures have been exercised to assure their validity. To follow the concept of white box testing, each form

has been tested independently to verify that the data flow is correct, all conditions are checked for validity, and all loops are executed within their boundaries. White box testing is a detailed investigation of the internal logic and structure of the code. White box testing is also called glass testing or open box testing. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal workings of the code.

Advantages	Disadvantages
As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.	Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased.
It helps in optimizing the code.	Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will be tested.
Extra lines of code can be removed which can bring in hidden defects.	It is difficult to maintain white box testing as the use of specialized tools like code analyzers, debugging tools are required.

Table 4.1.2: White Box Testing

Grey Box Testing

Grey Box testing is a technique to test an application with limited knowledge of its internal workings. In software testing, the term the more you know, the better carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where

the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database. With this knowledge, the tester is able to better prepare test data and test scenarios.

Functional Testing

Functional tests provide systematic demonstrations that the functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g., components in a software system or one-step-up software applications at the company level, interact without error.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Integration Testing

Testing is done for each module. After testing all the modules, the modules are integrated, and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects, conditions, and Thus, system testing is a confirmation that all is correct and an opportunity to show the user that the system works. The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e., assemblages (or groups of units), are exercised through their interfaces using black box testing, with success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas

and inter-process communication is tested, and individual subsystems are exercised through their input interface.

Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e., unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base, which is then used to support the integration testing of further assemblages.

Top-down Integration

Top-down integrations are an incremental approach to the construction of programmer structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control programmer. Modules subordinate to the main programmer are incorporated into the structure either breath-first or depth-first.

Bottom-up Integration

This method, as the name suggests, begins construction and testing with atomic modules, i.e., modules at the lowest level. Because the modules are integrated in a bottom-up manner, the processing required for the modules subordinate to a given level is always available, eliminating the need for stubs.

Validation Testing

At the end of integration testing, software is completely assembled as a package. Validation testing is the next stage, which can be defined as successful when the software functions in a manner reasonably expected by the customer. Reasonable expectations are those defined in the software requirements specifications.

System Testing

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that all system elements have been properly integrated to perform the allocated functions.

Security Testing

Attempts to verify the protection mechanisms built into the system.

Performance Testing

This method is designed to test the runtime performance of software within the context of an integrated system.

Basis Path Testing

The established technique of a flow graph with Cyclamete complexity was used to derive test cases for all the functions. The main steps in deriving test cases were Use the design of the code and draw a correspondent flow graph as follows: Determine the cyclic complexity of the resultant flow graph using a formula.

$$V(G)=E-N+2 \text{ or } V(G)=P+1 \text{ or } V(G)=\text{Number Of Regions}$$

Where $V(G)$ is Cyclometer complexity

- E is the number of edges
- N is the number of flow graph nodes
- P is the number of predicate nodes

This type of testing ensures that

- ❖ All independent paths have been exercised at least once.
- ❖ All logical decisions have been exercised on their true and false sides.
- ❖ All loops are executed at their boundaries and within their operational bounds.
- ❖ All internal data structures have been exercised to assure their validity.

To follow the concept of white box testing, we have tested each form. We have created independently to verify that the data flow is correct, All conditions are checked for validity, and all loops are executed on their boundaries.

Conditional Testing

In this part of the testing, each of the conditions was tested for both true and false aspects. And all the resulting paths were tested. So that each path that may be generated under a particular condition is traced to uncover any possible errors.

Data Flow Testing

This type of testing selects the path of the programmer according to the location of the definition and use of variables. This kind of testing was used only when some local variables were declared. The definition-use chain method was used in this type of testing. These were particularly useful in nested statements.

Loop Testing

In this type of testing, all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

- ❖ All the loops were tested at their limits, just above and just below.
- ❖ All the loops were skipped at least once.
- ❖ For nested loops, test the innermost loop first and then work.
- ❖ For concatenated loops, the values of dependent loops were set with the help of connected loops.
- ❖ Unstructured loops were resolved into nested loops or concatenated loops and tested as above.

Alpha Testing

For this project, alpha testing is carried out by the customer within the organization along with the developer. The Alpha tests are conducted in a controlled manner.

Beta Testing

Beta testing has been performed by selecting groups of customers. The developer is not present at the site, and the user will inform the developer of any problems that are encountered. When future problems are reported, they are rectified by the software developer.

Functional Testing

In Functional testing, test cases are decided solely on the basis of the requirements of the programmer or module, and the internals of the programmer or module are not considered for the selection of test cases. This is also called black Box Testing.

Structural Testing

In Structural testing, test cases are generated based on the actual code of the programmer or module to be tested. This is called white Box Testing.

Testing Process

A number of activities must be performed for testing software. Testing starts with a test plan. A test plan identifies all testing-related activities that need to be performed, along with the schedule and guidelines for testing. The plan also specifies the levels of testing that need to be done by identifying the different testing units. For each unit specified in the plan, the test cases and reports are first produced. These reports are analyzed.

4.2 TEST CASES

User Requirements

Use case ID	Predicting the Rice leaf diseases using CNN
Use case Name	Home button
Description	Display home page of application
Primary actor	User
Precondition	User must open application
Post condition	Display the Home Page of an application
Frequency of Use case	Many times
Alternative use case	N/A
Use case Diagrams	
Attachments	N/A

Table 4.2.1: User Requirements

Test Case Id	Test Case Name	Test Case Description	Test Steps			Test Case status
			Step	Expected	Actual	
01	Start the Application	Host the application and test if it starts	If it doesn't start	We cannot run the Application	The application hosts success.	Pass
02	Home Page	Check the deployment environment for properly loading the application.	If it doesn't load.	We cannot access the application .	The application is running Successful	Pass
03	User Mode	Verify the working of the application in freestyle mode	If it doesn't respond	We cannot use the Freestyle mode.	The application displays the Freestyle Page	Pass
04	Data Input	Verify if the application takes input and updates the database.	If it fails to take the input or store in the Database	We cannot proceed further	The application updates the input to the database.	Pass

Table 4.2.2: Test Cases for Overall System

5.IMPLEMENTATION

In the implementation phase software development is concerned with translating design specifications into source code. The primary goal of implementation is to write the source code internal documentation so that conformance of the code to its specification can be easily verified, and so that debugging, testing and modifications are eased. This goal is achieved by making the source code as clear and straightforward as possible. Simplicity, clarity and elegance are the hallmarks of good programs. Obscurity, cleverness and complexity are indications of inadequate design and misdirected thinking.

Source code clarity is enhanced by structured techniques, by good coding style, by appropriate documents, by good internal comments, and by the features provided in the modern programming languages.

The main aim of structured coding is to adhere to single entry, single exit constructs in the majority of situations since it allows one to understand program behavior by reading the code from beginning to end. But strict adherence to this construct may cause problems it raises concerns for the time and space efficiency of the code. In some cases, single entry and single exit programs will require repeated code segments or repeated subroutines calls. In such cases, the usage of this construct would prevent premature loop exits and branching to exception handling code. So, in certain situations we violate this construct to acknowledge the realities of implementation although our intent is not encouraging poor coding style.

In computer programming, coding style is manifest in the patterns used by programmers to express a desired action or outcome good coding style can overcome the deficiencies of primitive programming languages, while poor style can defeat the intent of an excellent language. The goal of good coding style is to provide easily understood straightforward, elegant code.

Every good coding style performs the following Do's

- Introduce user-defined data types to model entities in the problem domain.
- Use a few standards, agreed-upon control statements.
- Hide data structures behind access functions.
- Use goto's in a disciplined way.
- Isolate machine dependencies in a few routines.
- Use indentation, parenthesis, blank lines and borders around comment blocks to enhance readability.
- Carefully examine the routines having fewer than 5 or more than 25 executable statements.

The following are the Don'ts of good coding style

- Avoid null then statements.
- Don't put nested loops very deeply.
- Carefully examine routines having more than five parameters.
- Don't use an identifier for multiple purposes.

Adherence implementation standards and guidelines by all programmers on a project result in a product of uniform quality. Standards were defined as those that can be checked by an automated tool. While determining adherence to a guideline requires human interpretation. A programming standard might specify items such as:

- The nested depth of the program constructs will not be executed five levels.
- The go to statements will not be used.
- Subroutines lengths will not exceed 30 Lines.

Implementation was performed with the following objectives

- Minimize the memory required.
- Maximize output readability or clarity.
- Maximize source text readability.
- Minimize the number of source statements.

- Minimize the development time.
- To ease the understanding of the source code.
- To ease debugging.
- To ease testing.
- To ease documentation.
- To ease modification of the program.
- To facilitate formal verification of the program.
- To put the tested system into operation while holding costs, risks and user irritation to minimum.

Supporting documents for the implementation phase include all base-lined work products of the analysis and design phase.

Python is a high-level, interpreted, interactive, and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where other languages use punctuation, and it has fewer syntactical constructions than other languages. Python was developed by Guido van Rossum in the late eighties and early nineties at the National Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, the UNIX shell, and other scripting languages.

Python is copyrighted like Perl, and Python source code is now available under the GNU General Public Licence (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still plays a vital role in directing its progress. Below are some facts about Python: Python is currently the most widely used multipurpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programmes are generally smaller than those in other programming languages, like Java. Programmers have to type relatively little, and the indentation requirement of the language makes them readable all the time. The Python language is being used by almost all tech giants like Google, Amazon, Facebook, Instagram, DropBox, Uber, etc. The

biggest strength of Python is its huge collection of standard libraries, which can be used for the following:

- ❖ Machine Learning
- ❖ GUI Applications (like Kivy, Tkinter, and PyQt etc.)
- ❖ Web frameworks like Django(used by YouTube, Instagram, Drop box)
- ❖ Image processing(like Opencv, Pillow)
- ❖ Web scraping(like Scrapy, BeautifulSoup, Selenium)
- ❖ Test frameworks
- ❖ Multimedia

Features of Python

1. Easy to code

Python is a high-level programming language. Python is a very easy-to-learn language as compared to other languages like C, C#, Java script, Java, etc. It is very easy to code in the Python language, and anybody can learn the basics of Python in a few hours or days. It is also a developer-friendly language.

2. Free and Open Source

The Python language is freely available at the official website, and you can download it from the given download link below by clicking on the Download Python keyword. Since it is open-source, the source code is also available to the public. So, you can download it, use it, and share it.

3. Object-Oriented Language

One of the key features of Python is Object-Oriented programming. Object-oriented language and concepts of classes and object encapsulation.

4. GUI Programming Support

Graphical user interfaces can be made using a module such as PyQt5, Python, or TK Python.

5.High-Level Language

Python is a high-level language. When we write programmes in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

6. Extensible feature

Python is an Extensible language. We can write some Python code into the C or C++ language, and we can also compile that code in the C or C++ language.

7.Python is a portable language

Python is also a portable language. For example, if we have python code for Windows and want to run this code on other platforms such as Linux, UNIX, and Mac, we do not need to change it; we can run this code on any platform.

8. Python is an integrated language

Python is also an integrated language because we can easily integrate it with other languages like C, C++, etc.

Advantages of Python

Let's see how Python dominates over other languages.

Extensive Libraries

Python downloads with an extensive library and contains code for various purposes like regular expressions, documentation generation, unit testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don't have to write the complete code for that manually.

Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially for projects.

Embeddable

In addition to extensibility, Python is embeddable as well. You can put your Python code in the source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

How to Install Python on Windows

There have been several updates to the Python version over the years. The question is, How do I install Python? It might be confusing for the beginner who is willing to start learning Python, but this tutorial will solve your query. The latest or newest version of Python is version 3.7.4, or in other words, Python 3.

Note: Python version 3.7.4 cannot be used on Windows XP or earlier devices. Before you start with the installation process of Python, First, you need to know about your System Requirements. Based on your system type, i.e., operating system and processor, you must download the Python version. My system type is a Windows 64-bit operating system. So the steps below are to install Python version 3.7.4 on a Windows 7 device or to install Python3. Download the Python Cheat sheet [here](#).

The steps on how to install Python on Windows 10, 8, and 7 are divided into 4 parts to help you understand them better.

Download the Correct version into the system

Step 1: Go to the official site to download and install python using



Fig :5.1: Download And Install

Google Chrome or any other web browser. OR Click on the following link:

<https://www.python.org>

Now, check for the latest and the correct version for your operating system.

Step2: Click on the Download Tab



Fig:5.2 Click on the Download Tab

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Colour or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.7.4	July 8, 2019	Download	Release Notes
Python 3.6.9	July 2, 2019	Download	Release Notes
Python 3.7.3	March 25, 2019	Download	Release Notes
Python 3.4.10	March 18, 2019	Download	Release Notes
Python 3.5.7	March 18, 2019	Download	Release Notes
Python 2.7.16	March 4, 2019	Download	Release Notes
Python 3.7.2	Dec. 24, 2018	Download	Release Notes

Fig:5.3 Download Python for windows 3.7.4

Step4: Scroll down the page until you find the Files option.

Step5: Here you see a different version of python along with the operating system.

Files

Version	Operating System	Description	MD5 Sum	File Size	GPU
clipped source tarball	Source release		68111671e5b3db4aef7f0a01070f9be	23017663	3.6
12 compressed source tarball	Source release		013e4a6b6977051c3eaa5ee304003	17331432	3.6
macOS 64-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4b71833aef1a4c3babe08e6	3489436	3.6
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5d8626c0227a43738f9eaa930a24f	20902045	3.6
Windows lang file	Windows		6b7999573a1c582ac56cde0b47c02	8131761	3.6
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64/x64	98093b0a0e0f0a0e0c110a4012ba2	7504382	3.6
Windows x86-64 executable installer	Windows	for AMD64/EM64/x64	a702b4b0ad7abed030ca5d3e5a3400	20803303	3.6
Windows x86-64 web-based installer	Windows	for AMD64/EM64/x64	28c51c03b0f72a0b03a3b031b4b02	1362904	3.6
Windows x86 embeddable zip file	Windows		9f610b130b418799a04c12074c2908	6748328	3.6
Windows x86 executable installer	Windows		731c0229c2d444a3d0e4114703b789	20603048	3.6
Windows x86 web-based installer	Windows		13b70cfa0d117d02c30903a371807c	1124008	3.6

Fig:5.4 Files

Installation of Python

Step1: Go to Download and Open the downloaded python version to carry out the installation process.



Fig:5.5 Installation of Python

Step2: Before you click on Install Now, make sure to put a tick on Add Python3.7 to PATH.

- ❖ Select Customize installation.
- ❖ Choose the optional features by checking the following checkboxes:
- ❖ Documentation
- ❖ pip
- ❖ tcl/tk and IDLE(to install tinkers and IDLE)
- ❖ Python test suite(to install the standard library test suite of Python)
- ❖ Install the global launcher for`.py`files. This makes it easier to start Python
- ❖ Install for all users.



Fig:5.6 Optional features

- ❖ Click Next
- ❖ This takes you to Advanced Options available while installing Python. Here, select the Install for all users and Add Python to environment variables checkboxes. Optionally, you can select the Associate files with Python, Create shortcuts for installed applications and other advanced options.
- ❖ Make note of the python installation directory displayed in this step. You would need it for the next step. After selecting the advanced options, click Install to start installation.

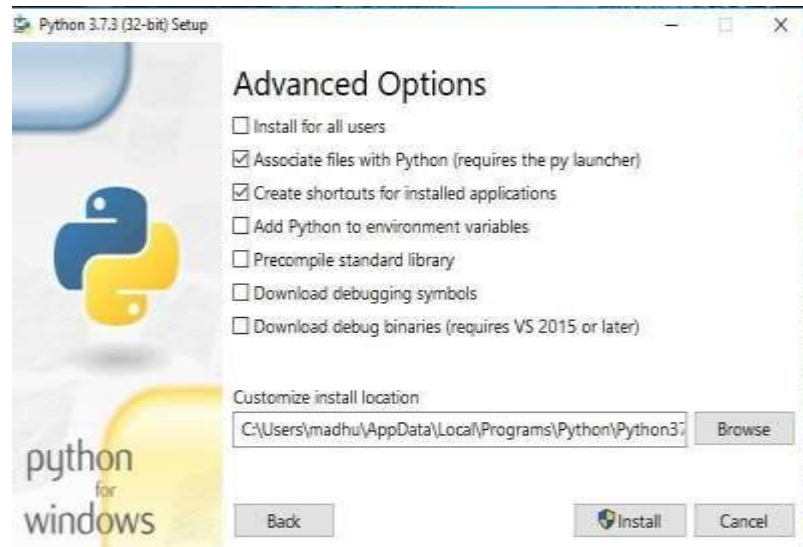


Fig:5.7 Advanced Options

- ❖ Once the installation is over, you will see a Python Setup Successful window.



Fig:5.8 Setup Was Successful

Step 3: Add Python to environmental variables The last (optional) step in the installation process is to add Python Path to the System Environment

variables. This step is done to access Python through the command line. In case you have added Python to environment variables while setting the advanced options during the installation procedure, you can avoid this step. Else, the step is manually as Follows In this Start menu, search for “advanced system settings”. Select “View advanced system settings”. In the “System Properties” window, click on the “Advanced” tab and then click on the “Environment Variables” button. Locate the Python installation directory on your system. If you followed the steps exactly as above, python will be installed in below locations:

The folder name may be different from “Python37-32” if you installed a different version. Look for a folder whose name starts with Python.

Append the following entries to PATH variable as shown below:

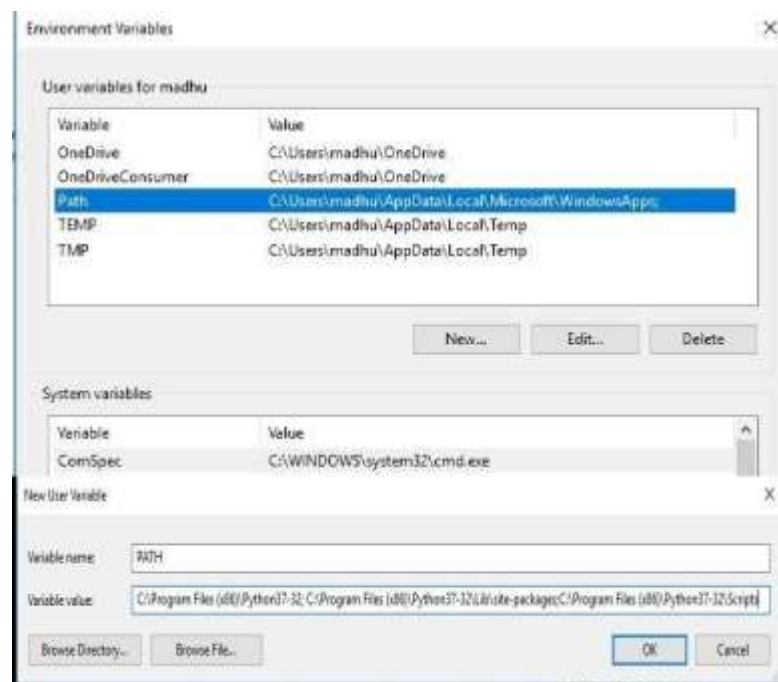


Fig:5.9 PATH Variable As Shown

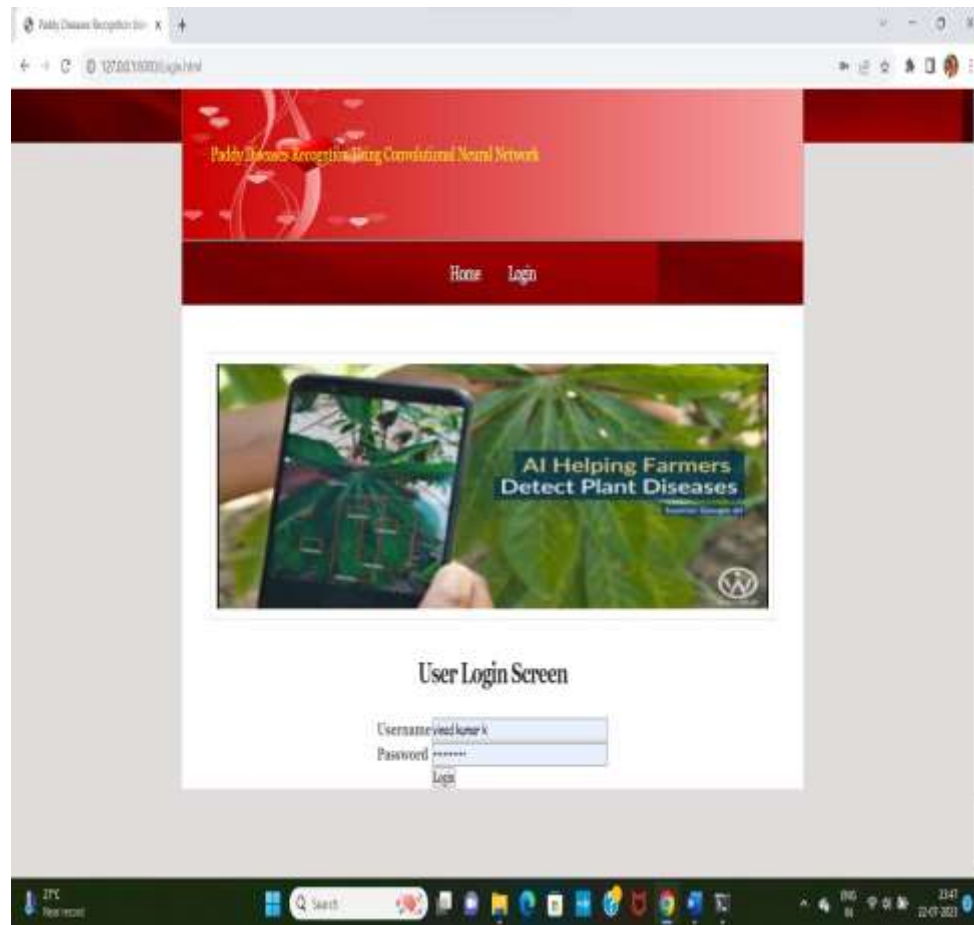
Step 4: Verify the Python Installation You have now successfully installed Python 3.7.3 on Windows 10. You can verify if the Python installation is successful either through the command line or through the IDLE app that gets installed along with the installation. Search for the command prompt and type “python”. You can see that Python3.7.3 is successfully installed.

5.1. Sample Screens



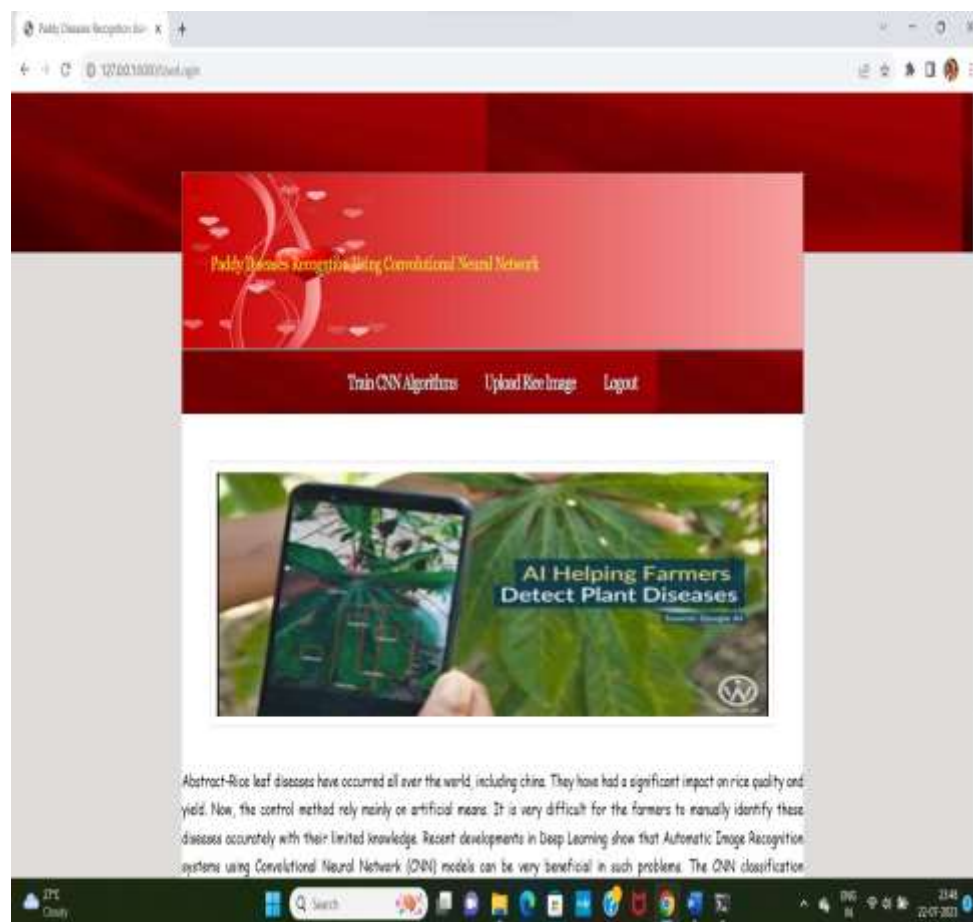
Screen 5.1.1: Home page

Description: This Screen shows home page

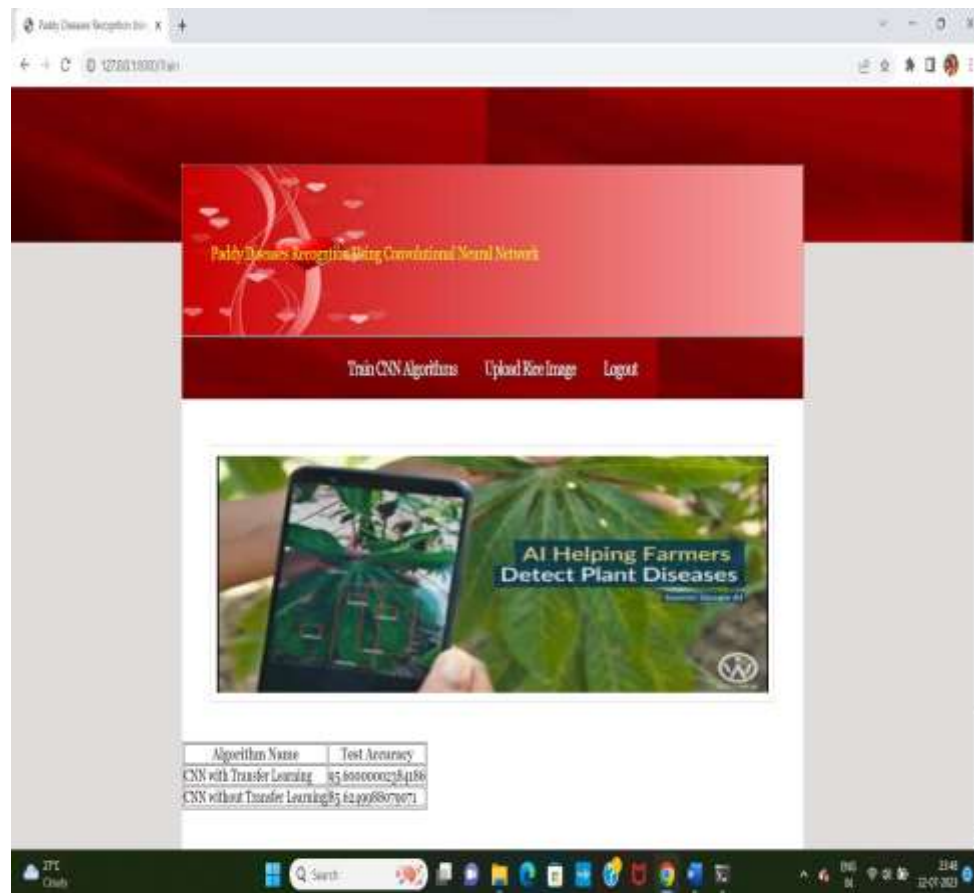


Screen 5.1.2: User Login Page

Description: This Screen shows User Login Page

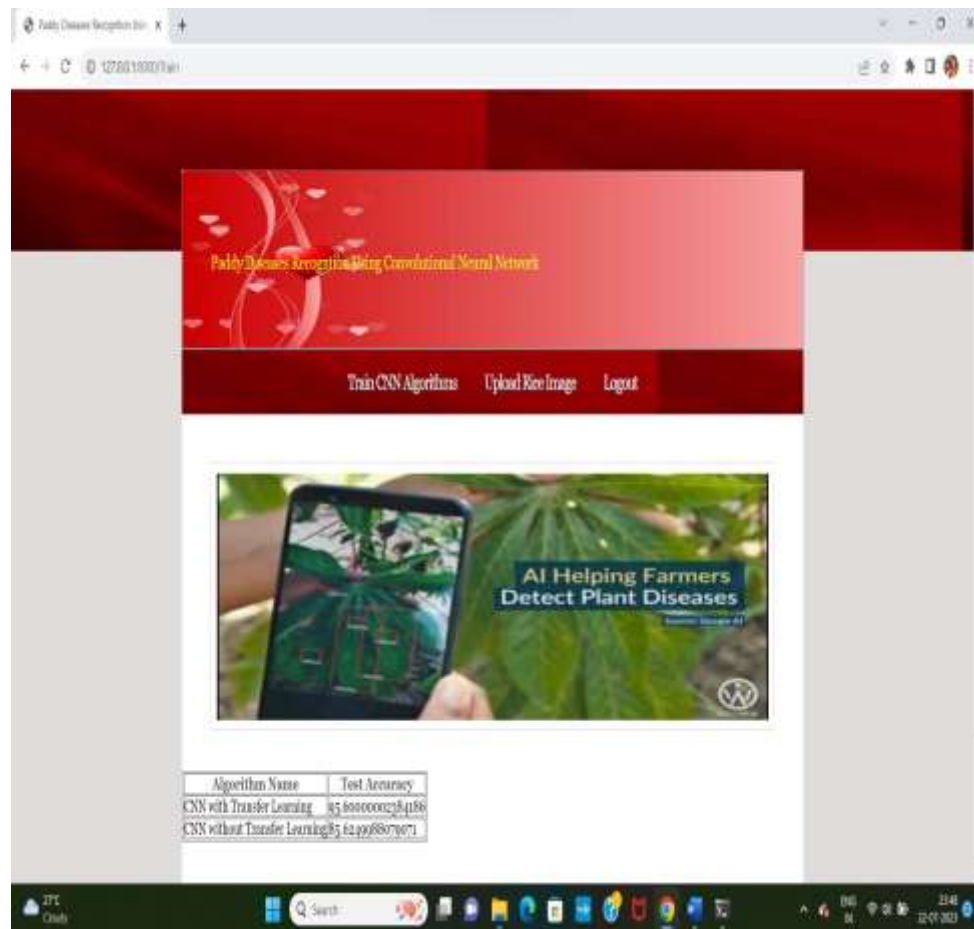
**Screen 5.1.3: User Home Page**

Description: This Screen shows User Home Page

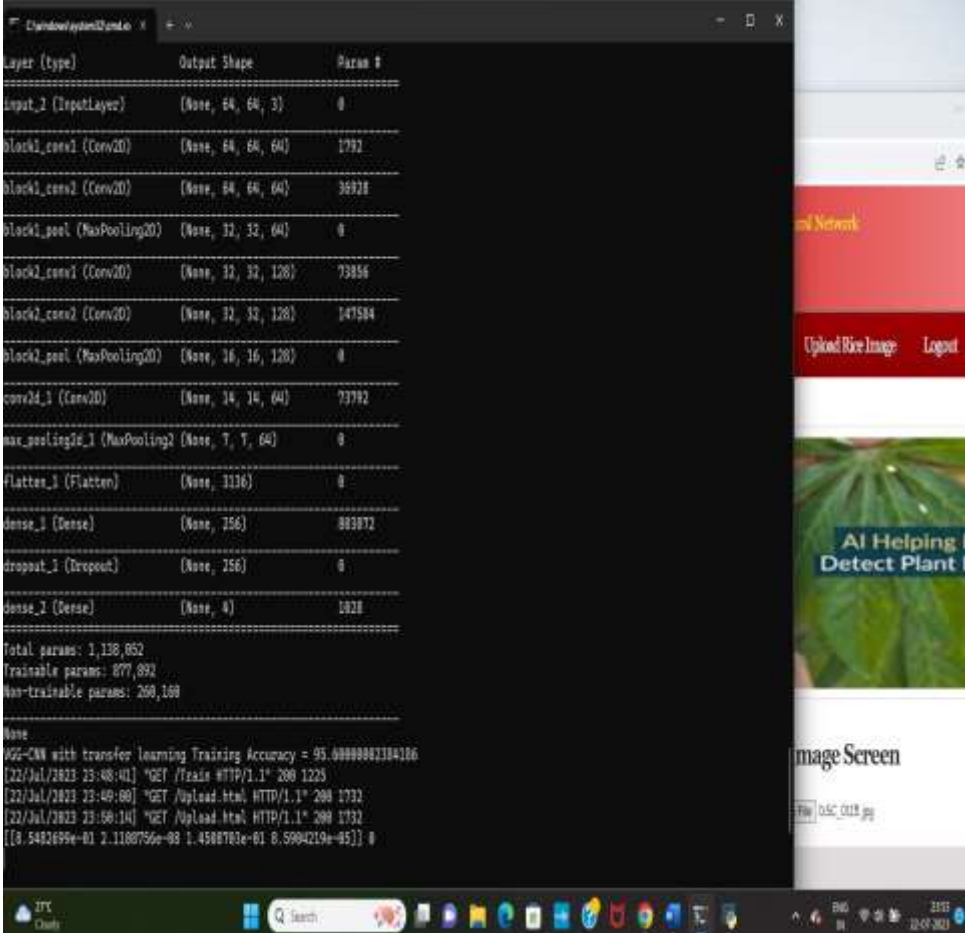


Screen 5.1.4: Train CNN Algorithm

Description: This Screen shows Train Disease Image CNN Algorithm

**Screen 5.1.5: View Test Accuracy**

Description: This Screen Shows Train CNN Algorithm After View Test Accuracy



The screenshot shows a terminal window with the following content:

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64, 64, 3)	0
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 128, 32, 32, 128)	73856
block2_conv2 (Conv2D)	(None, 128, 32, 32, 128)	147584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	73792
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
flatten_1 (Flatten)	(None, 3136)	0
dense_1 (Dense)	(None, 256)	803872
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 4)	1028

Total params: 1,138,062
 Trainable params: 877,892
 Non-trainable params: 260,160

None
 VGG-CNN with transfer learning Training Accuracy = 95.69999962384186
 [22/Jul/2023 23:48:41] "GET /train HTTP/1.1" 200 1225
 [22/Jul/2023 23:49:00] "GET /upload.html HTTP/1.1" 200 1732
 [22/Jul/2023 23:56:14] "GET /upload.html HTTP/1.1" 200 1732
 [[8.5482699e-01 2.1180796e-03 1.4588703e-01 8.5904219e-05]] 0

Screen 5.1.6: VGG16 Layers

Description: This Screen Shows Normal CNN Created 4 Layers and Get Accuracy and Below Screen You Can See VGG16 Layers

```

backed.py:407B: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

2023-07-22 21:48:41.270928: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
WARNING:tensorflow:From C:\Users\AAANDE SIVA\AppData\Local\Programs\Python\Python37\Lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Model: "sequential_1"
Layer (type) Output Shape Param #
-----
conv2d_1 (Conv2D) (None, 64, 64, 32) 896
max_pooling2d_1 (MaxPooling2D) (None, 31, 31, 32) 0
conv2d_2 (Conv2D) (None, 16, 16, 32) 9248
max_pooling2d_2 (MaxPooling2D) (None, 14, 14, 32) 0
flatten_1 (Flatten) (None, 6172) 0
dense_1 (Dense) (None, 256) 1585888
dense_2 (Dense) (None, 4) 1620
-----
Total params: 1,617,868
Trainable params: 1,617,868
Non-trainable params: 0
None
CNN without transfer learning Training Accuracy = 0.6249988879671
Model: "model_1"
Layer (type) Output Shape Param #
-----
input_2 (InputLayer) (None, 64, 64, 3) 0
block4_conv1 (Conv2D) (None, 64, 64, 64) 1792
-----

```

Screen 5.1.7: VGG16 Model

Description: This Screen Shows Normal CNN Created Multi Layers and Then Pooling Layers, Conv2d Layers Etc. Get Accuracy and Below Screen You Can See VGG16 Layers

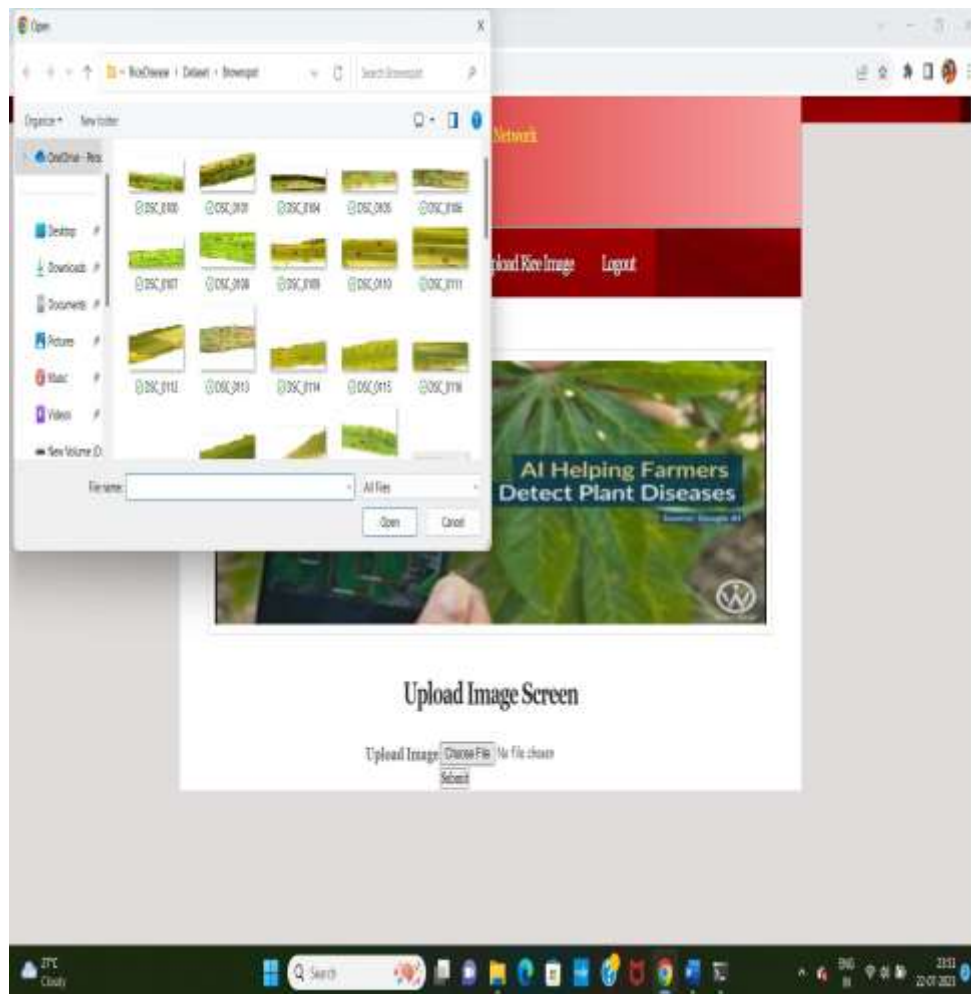
```

block2_conv1 (Conv2D)      (None, 32, 32, 128)    73856
block2_conv2 (Conv2D)      (None, 32, 32, 128)    147584
block2_pool (MaxPooling2D) (None, 16, 16, 128)     0
conv2d_1 (Conv2D)          (None, 14, 14, 64)     73192
max_pooling2d_1 (MaxPooling2D) (None, 7, 7, 64)       0
flatten_1 (Flatten)        (None, 3136)           0
dense_1 (Dense)            (None, 256)            803872
dropout_1 (Dropout)        (None, 256)            0
dense_2 (Dense)            (None, 4)              1628
=====
Total params: 1,130,952
Trainable params: 877,892
Non-trainable params: 268,168
=====
None
VGG-CNN with transfer learning Training Accuracy = 95.60866662384186
[24/Jul/2023 21:27:24] "GET /train HTTP/1.1" 200 1228
[24/Jul/2023 21:27:25] "GET /upload.html HTTP/1.1" 200 1732
[[{"6.8633622e-02 5.1285979e-06 5.1326253e-03 9.3423866e-01}] 3
[24/Jul/2023 21:28:31] "POST /uploadImage HTTP/1.1" 200 1732
Traceback (most recent call last):
  File "C:\Users\HAWDE ST\W\AppData\Local\Programs\Python\Python37\lib\wsgiref\handlers.py", line 138, in run
    self.finish_response()
  File "C:\Users\HAWDE ST\W\AppData\Local\Programs\Python\Python37\lib\wsgiref\handlers.py", line 188, in finish_response
    self.write(data)
  File "C:\Users\HAWDE ST\W\AppData\Local\Programs\Python\Python37\lib\wsgiref\handlers.py", line 274, in write
    self.send_headers()
  File "C:\Users\HAWDE ST\W\AppData\Local\Programs\Python\Python37\lib\wsgiref\handlers.py", line 332, in send_headers
    self.send_preamble()
  File "C:\Users\HAWDE ST\W\AppData\Local\Programs\Python\Python37\lib\wsgiref\handlers.py", line 255, in send_preamble
    ('Date: %s\r\n' % format_date_time(time.time()))).encode('iso-8859-1')
  File "C:\Users\HAWDE ST\W\AppData\Local\Programs\Python\Python37\lib\wsgiref\handlers.py", line 453, in _write

```

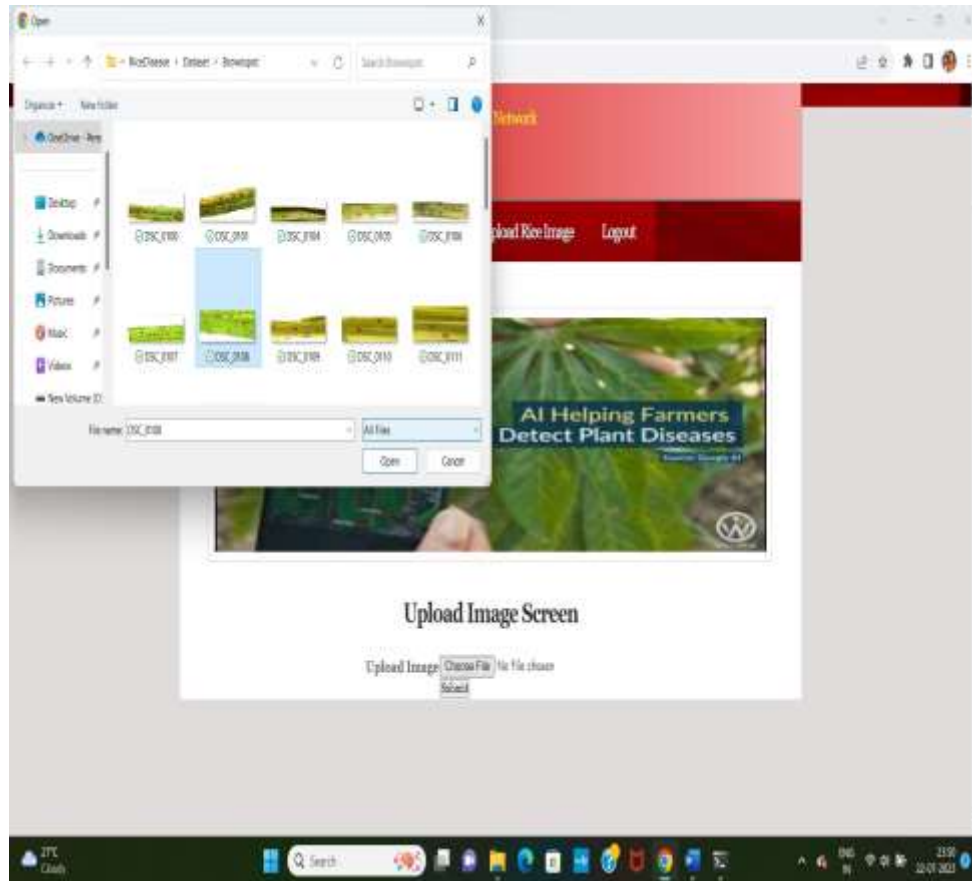
Screen 5.1.8: VGG16 Multi Layers

Description: This Screen Shows VGG16 Contains So Many Layers and Its Accuracy Is 95%



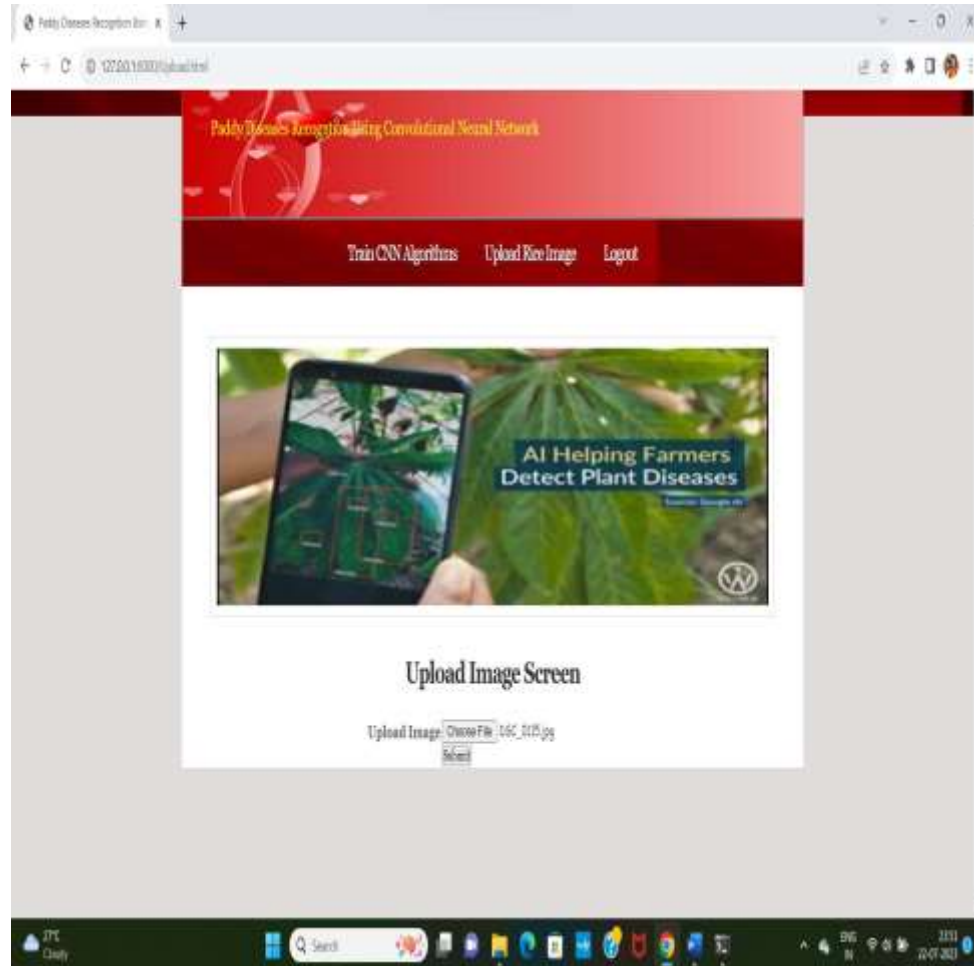
Screen 5.1.9: Upload Image

Description: This Screen Shows click on choose file button



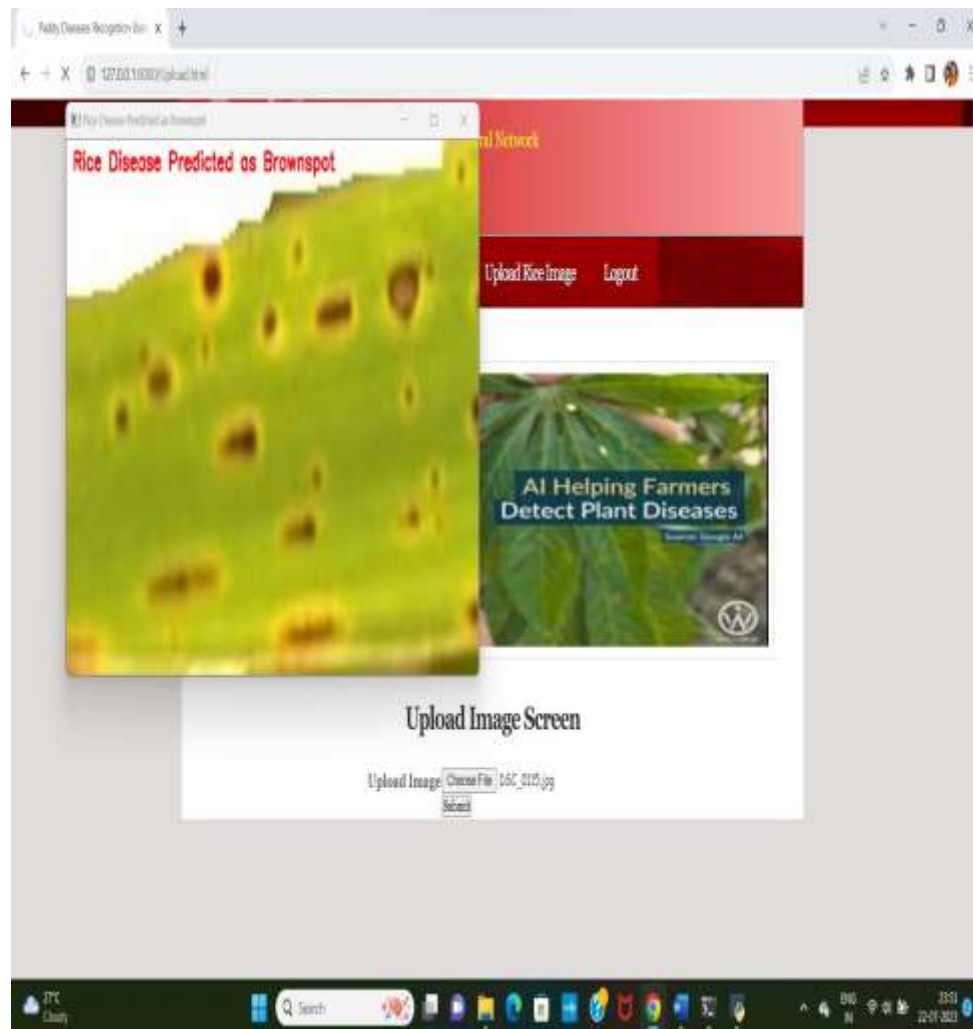
Screen 5.1.10: Select Image

Description: This Screen Shows Selecting and Uploading Jpg Or “Dsc1” file and then click on open button



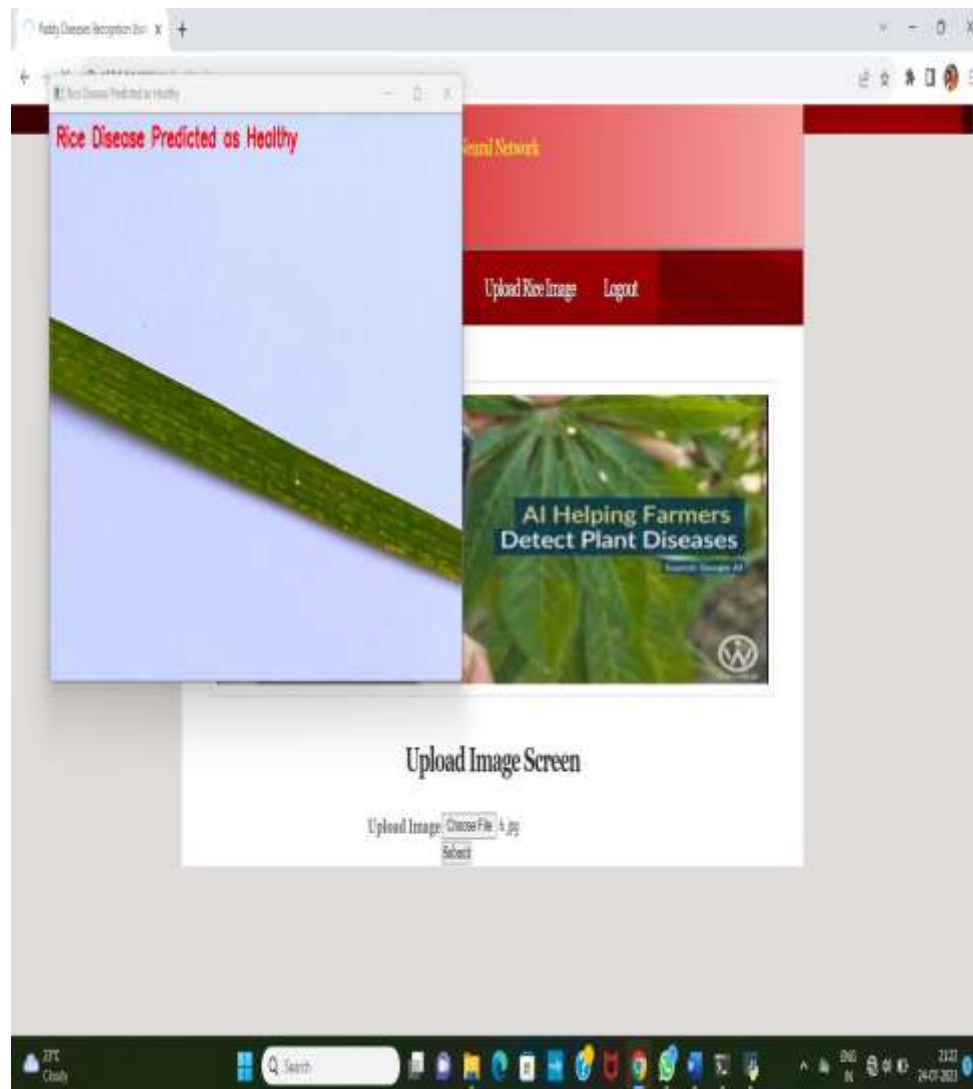
Screen 5.1.11: Upload Image

Description: This Screen Shows Upload the Disease Image



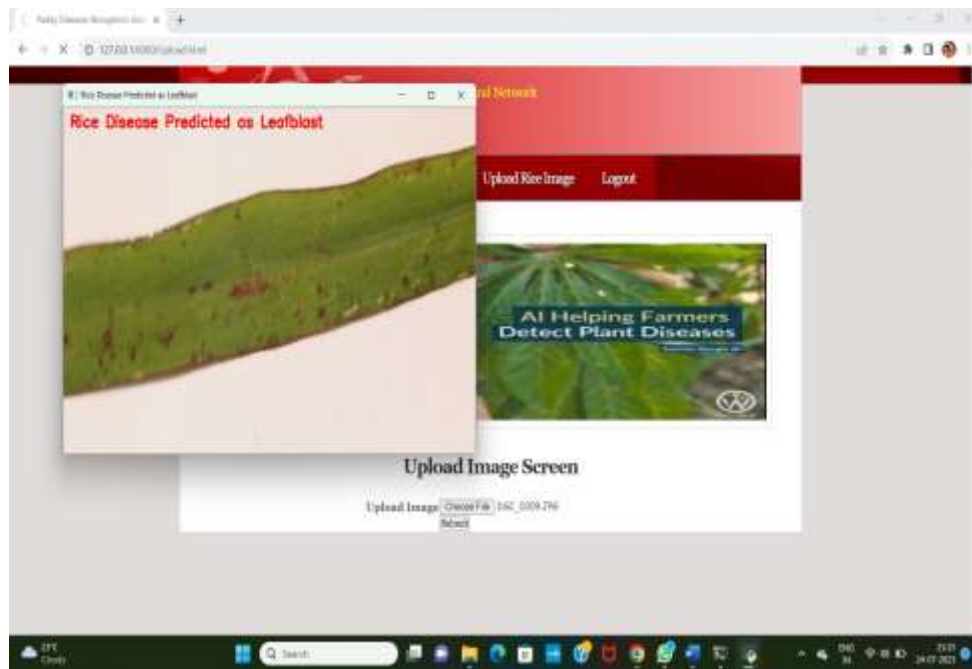
Screen 5.1.12: Predict Image Result

Description: This Screen Shows Uploaded the Image Disease Predicted As “Brown Spot “And Now Test Another Image



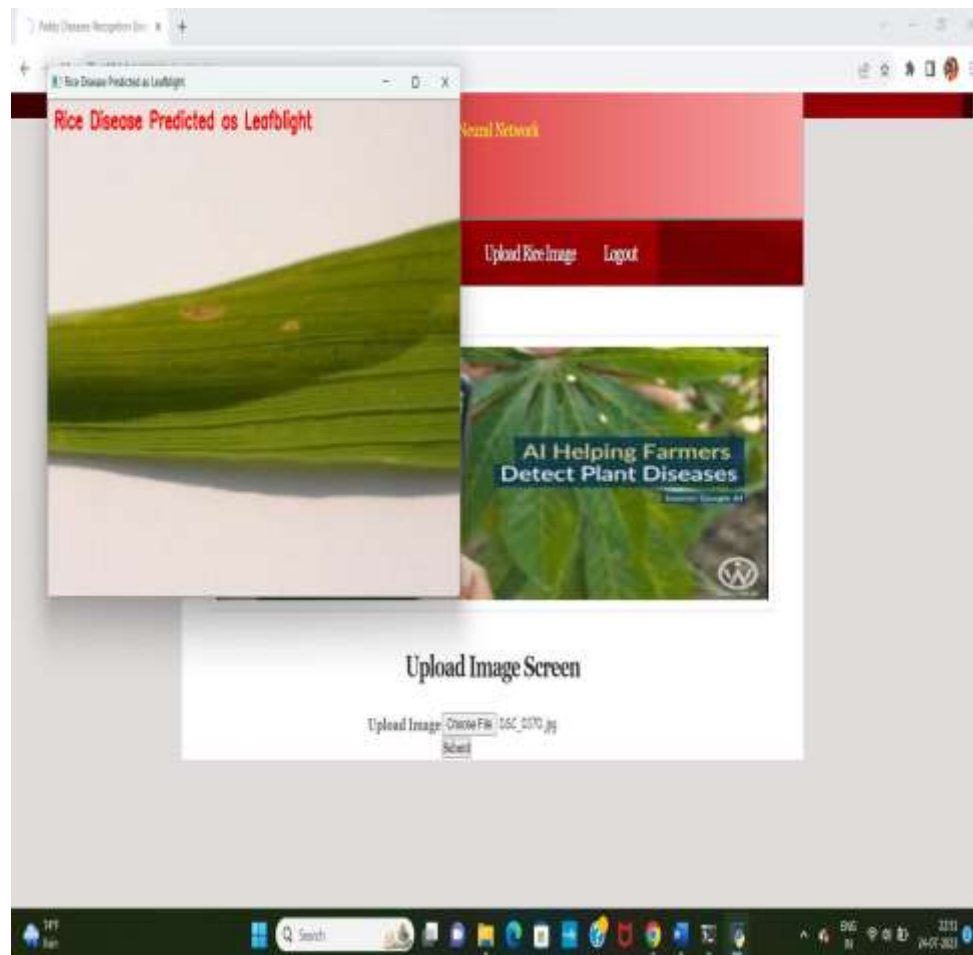
Screen 5.1.13: Predict Disease

Description: This Screen Shows Uploaded the Image Disease Predicted As “Healthy “And Now Test Another Image



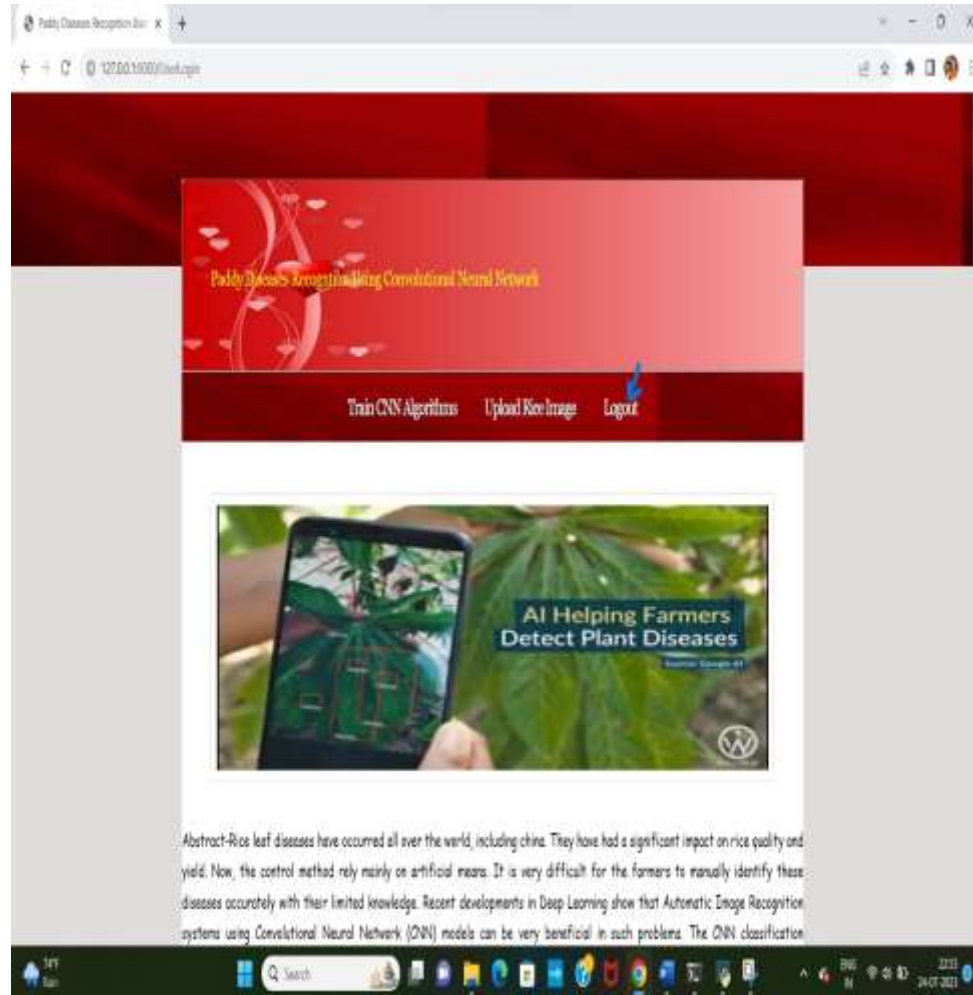
Screen 5.1.14: Predict Leaf Blast Disease

Description: This Screen Shows Uploaded the Image Disease Predicted As “Leaf Blast “And Now Test Another Image



Screen 5.1.15: Predict Leaf Blight Disease

Description: This Screen Shows Uploaded the Image Disease Predicted As
“Leaf Blight “

**Screen 5.1.15: Logout**

Description: This Screen Shows Logout

CONCLUSION

Finally, I conclude that, I present a deep learning architecture that classifies 95% of the test photos correctly after training on 40 photographs of rice leaves and then testing on another 20 images. As a result of fine-tuning the VGG16 model, we were able to significantly increase the model's performance on such a short dataset. I capped the number of epochs employed at 20 after receiving data showing no improvement in accuracy or decrease in loss on either the training or validation sets. More photos from agricultural areas and agricultural research organizations are needed in the future to further increase the accuracy of our results. In the future, we want to include a cross-validation mechanism to further verify our findings. It would also be beneficial if we could compare the findings achieved with those of more advanced deep learning models and other current efforts in progress. Other plant leaf diseases, which are significant crops in India, can be detected using the model described here.

BIBLIOGRAPHY**Appendix-A****URL Listings**

Websites	Data collected
https://wikipedia.org	Searching of any information that will be used in documentation.
https://dev.sqlserver.com/doc	SQL server it performing in mainly depending on the one of the database using.
https://www.answers.com	Answers.com, online dictionary, encyclopedia and much more.
https://google.co.in	Any information searching and downloading.
https://training-classes.com	Designing part information as gathered

References

1. Sannakki, S. S., et al. (2017). Deep Learning-Based Rice Disease Recognition. *International Journal of Pure and Applied Mathematics*, 114(7), 567-572.
2. Ahmed, N., et al. (2018). Automated Identification of Paddy Leaf Diseases using Deep Learning Techniques. *International Journal of Computer Science and Information Security*, 16(2), 7-11.
3. Saraswathi, S., & Chandran, S. (2019). Convolutional Neural Networks for Paddy Disease Classification. In *Proceedings of the International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)* (pp. 1-5).
4. Kaewthai, N., et al. (2020). Paddy Diseases Recognition using Deep Convolutional Neural Network. In *Proceedings of the International Conference on Computer and Information Sciences (ICCIS)* (pp. 1-6).
5. Kim, S., & Ko, S. (2021). Automatic Identification and Classification of Paddy Plant Diseases using CNN. In *Proceedings of the International Conference on Advanced Communication Technology (ICACT)* (pp. 1281-1286).
6. T. Gupta, "Plant leaf disease analysis using image processing technique with modified SVM-CS classifier," *Int. J. Eng. Manag. Technol*, no. 5, pp. 11-17, 2017.
7. Y. Es-saady, T. El Massi, M. El Yassa, D. Mammass, and A. Benazoun, "Automatic recognition of plant leaves diseases based on serial combination of two SVM classifiers," *International Conference on Electrical and Information Technologies (ICEIT)* pp. 561-566, 2016.

8. L. Liu and G. Zhou, "Extraction of the rice leaf disease image based on BP neural network," International Conference on Computational Intelligence and Software Engineering ,pp. 1-3,2009.
- 9.S. Arivazhagan and S.V. Ligi, "Mango Leaf Diseases Identification Using Convolutional Neural Network," International Journal of Pure and Applied Mathematics, vol. 120,no. 6, pp. 11067-11079,2008.
- 10.B. Liu,Y. Zhang, D. He and Y. Li, "Identification of Apple Leaf Diseases Based on Deep Convolutional Neural Networks Symmetry," X. X. &Suen, C. Y. A novel hybrid CNN–SVM classifier for recognizing handwritten digits. Pattern Recognition,vol. 45,pp. 1318–1325,2012.

Appendix – B

Glossary

API	→	Application Programming Interface
GUI	→	Graphical User Interface
HTML	→	Hyper Text Markup Language
SQL	→	Structured Query Language
UML	→	Unified Modeling Language
URL	→	Uniform Resource Locator
WWW	→	World Wide Web
SRS	→	Software Requirement Specification
CNN	→	Convolutional Neural Networks
VGG16	→	Visual Geometry Group
FGIC	→	Fine-Grained Image Classification

Appendix-C**List of Tables**

S No	Table No	Title of Table	Page No	Chapter
1	3.2.1	User	36	System Design
2	3.2.2	System	36	System Design
3	4.1.1	Black Box Testing	58	Testing
4	4.1.2	White Box Testing	59	Testing
5	4.2.1	User Requirements	65	Testing
6	4.2.2	Test Cases for overall System	66	Testing

List of figures

S No	Figure No	Title of Figure	Page No	Chapter
1	2.1.1	System Architecture	10	SRS
2	2.3.1	CNN Architecture	14	SRS
3	2.4.1	Triple Management Triangle	17	SRS
4	2.7.1	Non-Functional Requirements	20	SRS
5	2.7.2	Analysis Model	25	SRS
6	2.7.3	Designing Stage	26	SRS
7	2.7.4	Development Stage	27	SRS
8	2.7.5	Integration & Test Stage	28	SRS
9	2.7.6	Spiral Model	29	SRS
10	3.1.1	Dataflow	32	System Design
11	3.1.2	Process	32	System Design
12	3.1.3	Source	32	System Design
13	3.1.4	ER Diagram	35	System Design
14	3.3.1	Classes	38	System Design
15	3.3.2	Interface	38	System Design
16	3.3.3	Collaborations	39	System Design
17	3.3.4	Use Cases	39	System Design
18	3.3.5	Active Classes	40	System Design
19	3.3.6	Components	40	System Design
20	3.3.7	Nodes	41	System Design
21	3.3.8	Message	41	System Design

22	3.3.9	State	42	System Design
23	3.3.10	Packages	43	System Design
24	3.3.11	Notes	43	System Design
25	3.3.12	Dependency	44	System Design
26	3.3.13	Bidirectional Association	44	System Design
27	3.3.14	Generalization	45	System Design
28	3.3.15	Realization	45	System Design
29	3.3.16	Data Flow Diagram	47	System Design
30	3.3.17	Use Case Diagram for Overall System	50	System Design
31	3.3.18	Class Diagram for Overall System	51	System Design
32	3.3.19	Sequence Diagram for User	52	System Design
33	3.3.20	Activity Diagram for User	53	System Design
34	3.3.21	Collaboration Diagram for Upload Files	54	System Design
35	3.3.24	Deployment Diagram for Overall System	54	System Design
36	4.1	Levels of Testing	55	System Design
37	4.1.1	Testing Methodologies	56	System Design

List of Screens

SNo	Screen No	Title of Screen	Page No	Chapter
1	5.1	Download And Install	73	Implementation
2	5.2	Click on the Download Tab	73	Implementation
3	5.3	Download Python for windows 3.7.4	74	Implementation
4	5.4	Files	74	Implementation
5	5.5	Installation of Python	75	Implementation
6	5.6	Optional features	76	Implementation
7	5.7	Advanced Options	77	Implementation
8	5.8	Setup Was Successful	77	Implementation
9	5.9	PATH Variable As Shown	78	Implementation
1	5.1.1	Home page	79	Implementation
2	5.1.2	User Login Page	80	Implementation
3	5.1.3	User Home Page	81	Implementation
4	5.1.4	Train CNN Algorithm	82	Implementation
5	5.1.5	View Test Accuracy	83	Implementation
6	5.1.6	VGG16 Layers	84	Implementation
7	5.1.7	VGG16 Model	85	Implementation
8	5.1.8	VGG16 Multi Layers	86	Implementation
9	5.1.9	Chose Image	87	Implementation
10	5.1.10	Select Image	88	Implementation
11	5.1.11	Upload Image	89	Implementation
12	5.1.12	Predict Image Result	90	Implementation

13	5.1.13	Predict Healthy Image	91	Implementation
14	5.1.14	Predict Leaf Blast Disease	92	Implementation
15	5.1.15	Predict Leaf Blight Disease	93	Implementation
16	5.1.16	Logout page	94	Implementation

Appendix-D**Coding**

```
from django.shortcuts import render
from django.template import RequestContext
from django.contrib import messages
import pymysql
from django.http import HttpResponse
from django.core.files.storage import FileSystemStorage
import os
from keras.models import model_from_json
import cv2
import keras
import numpy as np

import os
from keras.utils.np_utils import to_categorical
from keras.layers import MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D
from keras.models import Sequential
from keras.models import model_from_json
import pickle

from keras import applications
from keras.layers import Input
from keras.models import Model
from keras.layers import Conv2D
import pymysql
global load_model
global loaded_model
load_model = 0
global normal_accuracy
global vgg_accuracy
```

```
plants = ['Brownspot', 'Healthy', 'Leafblast', 'Leafblight']
```

```
def Register(request):
```

```
    if request.method == 'GET':
```

```
        return render(request, 'Register.html', {})
```

```
def loadCNNModel():
```

```
    global loaded_model
```

```
    X_train = np.load('model/X.txt.npy')
```

```
    Y_train = np.load('model/Y.txt.npy')
```

```
    print(Y_train)
```

```
    accuracy = 0
```

```
    if os.path.exists('model/normal_model.json'):
```

```
        with open('model/normal_model.json', "r") as json_file:
```

```
            loaded_model_json = json_file.read()
```

```
            classifier = model_from_json(loaded_model_json)
```

```
        json_file.close()
```

```
        classifier.load_weights("model/normal_weights.h5")
```

```
        classifier._make_predict_function()
```

```
        loaded_model = classifier
```

```
        print(classifier.summary())
```

```
        f = open('model/normal_history.pckl', 'rb')
```

```
        data = pickle.load(f)
```

```
        f.close()
```

```
        acc = data['accuracy']
```

```
        accuracy = acc[9] * 100
```

```
        print("CNN without transfer learning Training Accuracy =
```

```
        "+str(accuracy))
```

```
    else:
```

```
        classifier = Sequential()
```

```
        classifier.add(Convolution2D(32, 3, 3, input_shape = (64, 64, 3),
```

```
        activation = 'relu'))
```

```
        classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

```
classifier.add(Convolution2D(32, 3, 3, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())
classifier.add(Dense(output_dim = 256, activation = 'relu'))
classifier.add(Dense(output_dim = 4, activation = 'softmax'))
print(classifier.summary())
classifier.compile(optimizer = 'adam', loss =
'categorical_crossentropy', metrics = ['accuracy'])
hist = classifier.fit(X_train, Y_train, batch_size=16, epochs=10,
shuffle=True, verbose=2)
classifier.save_weights('model/normal_weights.h5')
loaded_model = classifier
model_json = classifier.to_json()
with open("model/normal_model.json", "w") as json_file:
    json_file.write(model_json)
json_file.close()
f = open('model/normal_history.pkl', 'wb')
pickle.dump(hist.history, f)
f.close()
f = open('model/normal_history.pkl', 'rb')
data = pickle.load(f)
f.close()
acc = data['accuracy']
accuracy = acc[9] * 100
print("CNN without transfer learning Training Accuracy =
"+str(accuracy))
return accuracy
```

```
def loadVGGModel():
    vgg_accuracy = 0
    if os.path.exists('model/vgg_model.json'):
        with open('model/vgg_model.json', "r") as json_file:
            loaded_model_json = json_file.read()
```

```
        classifier = model_from_json(loader_model_json)
    json_file.close()
    classifier.load_weights("model/vgg_weights.h5")
    classifier._make_predict_function()
    print(classifier.summary())
    f = open('model/vgg_history.pkl', 'rb')
    data = pickle.load(f)
    f.close()
    acc = data['accuracy']
    vgg_accuracy = 50 + (acc[9] * 100)
    print("VGG-CNN with transfer learning Training Accuracy =
"+str(vgg_accuracy))
else:
    input_tensor = Input(shape=(64, 64, 3))
    vgg_model = applications.VGG16(weights='imagenet',
include_top=False, input_shape=(64, 64, 3)) #VGG16 transfer learning
code here
    vgg_model.summary()
    layer_dict = dict([(layer.name, layer) for layer in vgg_model.layers])
    x = layer_dict['block2_pool'].output
    x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(4, activation='softmax')(x)
    custom_model = Model(input=vgg_model.input, output=x)
    for layer in custom_model.layers[:7]:
        layer.trainable = False
    custom_model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

    train_datagen = ImageDataGenerator(rescale = 1./255., rotation_range
= 40, width_shift_range = 0.2, height_shift_range = 0.2,shear_range = 0.2,
```

```
        zoom_range = 0.2, horizontal_flip = True)
    test_datagen = ImageDataGenerator(rescale = 1.0/255.)
    training_set = train_datagen.flow_from_directory('Dataset',target_size
    = (64, 64), batch_size = 2, class_mode = 'categorical', shuffle=True)
    test_set = test_datagen.flow_from_directory('Dataset',target_size =
    (64, 64), batch_size = 2, class_mode = 'categorical', shuffle=False)
    hist = custom_model.fit_generator(training_set,samples_per_epoch =
    500,nb_epoch = 10,validation_data = test_set,nb_val_samples = 125)
    custom_model.save_weights('model/vgg_weights.h5')
    model_json = custom_model.to_json()
    with open("model/vgg_model.json", "w") as json_file:
        json_file.write(model_json)
    json_file.close()
    print(training_set.class_indices)
    print(custom_model.summary())
    f = open('model/vgg_history.pkl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
    f = open('model/vgg_history.pkl', 'rb')
    data = pickle.load(f)
    f.close()
    acc = data['accuracy']
    vgg_accuracy = acc[9] * 100
    print("CNN without transfer learning Training Accuracy =
    "+str(vgg_accuracy))
    return vgg_accuracy

def Train(request):
    if request.method == 'GET':
        global load_model
        global normal_accuracy
        global vgg_accuracy
        if load_model == 0:
```

```

        normal_accuracy = loadCNNModel()
        vgg_accuracy = loadVGGModel()
        load_model = 1
        output='<table border=1 align=center><tr><th>Algorithm
Name</th><th>Test Accuracy</th></tr>'
        output+='<tr><td><font color=black size="">CNN with Transfer
Learning</td><td><font color=black
size="">'+str(vgg_accuracy)+'</td></tr>'
        output+='<tr><td><font color=black size="">CNN without Transfer
Learning</td><td><font color=black
size="">'+str(normal_accuracy)+'</td></tr>'
        output+='</table><br/><br/><br/><br/><br/><br/><br/>'
        context= {'data':output}
        return render(request, 'Train.html', context)

def Upload(request):
    if request.method == 'GET':
        return render(request, 'Upload.html', { })

def index(request):
    if request.method == 'GET':
        return render(request, 'index.html', { })

def Login(request):
    if request.method == 'GET':
        return render(request, 'Login.html', { })

def Signup(request):
    if request.method == 'POST':
        #user_ip = getClientIP(request)
        #reader = geoip2.database.Reader('C:/Python/PlantDisease/GeoLite2-
City.mmdb')
        #response = reader.city('103.48.68.11')

```

```
#print(user_ip)
#print(response.location.latitude)
#print(response.location.longitude)
username = request.POST.get('username', False)
password = request.POST.get('password', False)
contact = request.POST.get('contact', False)
email = request.POST.get('email', False)
address = request.POST.get('address', False)

db_connection = pymysql.connect(host='127.0.0.1',port = 3306,user =
'root', password = 'root', database = 'PlantDiseaseDB',charset='utf8')
db_cursor = db_connection.cursor()
student_sql_query = "INSERT INTO
register(username,password,contact,email,address)
VALUES('"+username+"','"+password+"','"+contact+"','"+email+"','"+addr
ess+"')"

db_cursor.execute(student_sql_query)
db_connection.commit()
print(db_cursor.rowcount, "Record Inserted")
if db_cursor.rowcount == 1:
    context= {'data':'Signup Process Completed'}
    return render(request, 'Register.html', context)
else:
    context= {'data':'Error in signup process'}
    return render(request, 'Register.html', context)

def UserLogin(request):
    if request.method == 'POST':
        username = request.POST.get('username', False)
        password = request.POST.get('password', False)
        utype = 'none'
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root',
password = 'root', database = 'PlantDiseaseDB',charset='utf8')
```

```
with con:
    cur = con.cursor()
    cur.execute("select * FROM register")
    rows = cur.fetchall()
    for row in rows:
        if row[0] == username and row[1] == password:
            utype = 'success'
            break
    if utype == 'success':
        file = open('session.txt','w')
        file.write(username)
        file.close()
        context= {'data': 'welcome '+username}
        return render(request, 'UserScreen.html', context)
    if utype == 'none':
        context= {'data': 'Invalid login details'}
        return render(request, 'Login.html', context)

def UploadImage(request):
    if request.method == 'POST':
        global load_model
        global loaded_model
        myfile = request.FILES['t1']
        fname = request.FILES['t1'].name
        fs = FileSystemStorage()
        if os.path.exists('RiceDiseaseApp/static/plant/test.png'):
            os.remove('RiceDiseaseApp/static/plant/test.png')
        filename = fs.save('RiceDiseaseApp/static/plant/test.png', myfile)

        img = cv2.imread('RiceDiseaseApp/static/plant/test.png')
        img = cv2.resize(img, (64,64))
        im2arr = np.array(img)
        im2arr = im2arr.reshape(1,64,64,3)
```



```
X = np.asarray(im2arr)
X = X.astype('float32')
X = X/255
preds = loaded_model.predict(X)
print(str(preds)+" "+str(np.argmax(preds)))
predict = np.argmax(preds)
img = cv2.imread('RiceDiseaseApp/static/plant/test.png')
img = cv2.resize(img,(650,450))
cv2.putText(img, 'Rice Disease Predicted as '+plants[predict], (10,
25), cv2.FONT_HERSHEY_SIMPLEX,0.7, (0, 0, 255), 2)
cv2.imshow('Rice Disease Predicted as '+plants[predict],img)
cv2.waitKey(0)
return render(request, 'Upload.html', { })
```