International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Fruit Classifier Based On Machine Learning Analysis

Ranjitha.B(234012101325)

ABSTRACT

The project aims to develop an intelligent system capable of classifying different types of fruits based on their images. The model utilizes Convolutional Neural Networks (CNNs) to identify and categorize fruits such as apples, bananas, oranges, and more, offering high accuracy in distinguishing between various fruit types. In addition to classification, the system also evaluates the quality of the fruits, identifying features such as ripeness, blemishes, and texture, which are crucial for determining fruit quality. The model is trained on a large dataset of fruit images, enabling it to assess both visual and physical characteristics effectively. The project combines computer vision and deep learning techniques to automate the fruit classification and quality inspection process, which can significantly benefit industries like agriculture, retail, and food processing by improving product selection, reducing waste, and enhancing consumer satisfaction.

1.INTRODUCTION

The Fruit Classifier and Quality Using Deep Learning project aims to develop an intelligent system that can automatically identify and classify different types of fruits based on images. The system uses Convolutional Neural Networks (CNN) to process and classify fruit images into categories, such as apples, bananas, oranges, and more. By leveraging deep learning techniques, the project can accurately predict the fruit type, offering significant improvements in automated sorting systems for grocery stores and warehouses. Additionally, the project incorporates a quality assessment component, where the system not only classifies the fruit but also evaluates its quality. It uses image processing to detect defects, such as bruises, cuts, or ripeness, providing a score or label that indicates the fruit's condition. This feature can be particularly valuable for the agriculture and food industry, where ensuring the quality of produce is crucial for consumer satisfaction. Overall, this project combines the power of deep learning and computer vision to create an efficient, automated fruit classification and quality assessment system. It has potential applications in various sectors, including agriculture, retail, and supply chain management, aiming to improve productivity, reduce waste, and enhance consumer experience by offering high-quality produce.

ARTIFICIAL INTELLINGENCE:

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines programmed to think, reason, and make decisions like humans. It involves designing algorithms that allow computers and systems to perform tasks such as problem-solving, learning from data, understanding natural language, and recognizing patterns. AI can range from simple systems, like a chatbot that answers

basic questions, to complex applications, such as autonomous vehicles and advanced robotics. The goal of AI is to create systems that can perform tasks that typically require human intelligence.

AI is broadly categorized into two types: Narrow AI and General AI. Narrow AI focuses on performing specific tasks and is widely used today in applications like facial recognition, recommendation systems, and language translation. General AI, on the other hand, aims to replicate human-like intelligence that can perform a wide range of tasks and adapt to new situations—something that remains a long-term research goal. Machine Learning (ML), a subset of AI, plays a critical role by enabling systems to learn and improve from experience without being explicitly programmed.

The impact of AI spans various industries, including healthcare, finance, education, and entertainment. In healthcare, AI aids in diagnosing diseases and personalizing treatment plans. In finance, it detects fraud and automates trading. Despite its numerous benefits, AI also raises concerns about ethical implications, job displacement, and decision-making transparency. As AI

2.1 PROBLEM STATEMENT

Problem Definition: Fruit Classifier Using Machine Learning

Objective

Develop a machine learning-based system capable of accurately identifying and classifying different types of fruits from images. The classifier should be robust enough to handle variations in lighting, orientation, background, and fruit maturity levels.

The objective of this project is to develop an advanced deep learning med system for the Classifications and quality assent of fruits. By revenging state-of-the-art convolutional neural networks (CNN, the system aims to accurately identify different types of fruits and evaluate their quality based on visual features such as cud or, texture, shape, and surface blemishes. The project involves several key stages, including data collection and preprocessing, model development, training and optimization, and deployment. A comprehensive dataset of fruit images encompassing various types and quality grades is gathered and proposed to ensure subset model training. CNN architectures are then designed and trained to classify fruits and assess their quality. The models are rigorously tested and fine-tuned to achieve high accuracy and reliability. Upon successful validation, the trained models deployed into an application suitable for red-time usage in articulable fields marketplaces grocery stores, and food supply chains

Scope

- Input: Images of various fruits captured under different conditions.
- Output: Predicted fruit category (e.g., apple, banana, orange, strawberry, etc.).
- Performance Metrics: Accuracy, precision, recall, F1-score.

Key Components

Data Collection

- Gather a diverse dataset of labeled fruit images from online sources, datasets, or custom photography.
- Ensure variety in fruit types, sizes, colors, and environmental conditions.

Data Preprocessing

- Resize images to a consistent dimension.
- Normalize pixel values for uniformity.
- Augment data through rotations, flips, and brightness adjustments to improve robustness.

Feature Extraction

- Use raw pixel data or extract features like color histograms, shape descriptors, texture features.
- Alternatively, employ deep learning models (e.g., Convolutional Neural Networks) to automatically learn features.

Model Selection

- Traditional ML models: Support Vector Machines (SVM), Random Forests, k-Nearest Neighbours (k-NN).
- Deep Learning models: CNN architectures like VGG, ResNet, or custom models.

Training and Validation

- Split dataset into training, validation, and test sets.
- Tune hyperparameters to optimize performance.

Evaluation

- Test the model on unseen data.
- Use metrics like confusion matrix, accuracy, precision, recall, and F1-score to assess performance.

Deployment

- Integrate the trained model into a user-friendly application or API for real-time fruit classification.

Challenges & Considerations

- Handling similar-looking fruits to prevent misclassification.
- Ensuring the dataset is representative of real-world scenarios.
- Balancing model complexity with computational efficiency for deployment.

Summary

The fruit classifier aims to leverage machine learning techniques to automate fruit identification, aiding applications in agriculture, retail, and consumer electronics, enhancing efficiency and accuracy in fruit recognition tasks.

2.2 EXISTING SYSTEM

A traditional fruit classification and quality assessment system typically involves human inspectors evaluating fruits based on visual and tactile cues, such as color, size, shape, and texture. These systems rely on expert knowledge to categorize fruits into different grades or qualities, often using a set of predefined standards. In recent years, automated systems have been introduced, where machine vision and sensors are used to analyze fruits for classification and quality evaluation. These systems employ computer vision techniques to assess features like color, firmness, and external defects. Machine learning algorithms are often used to train models based on large datasets of fruit images, enabling the system to classify fruits and determine quality with high accuracy and speed. Such automated systems reduce the reliance on human judgment, increase processing speed, and improve consistency in the grading process.

However, despite the advances in automation, traditional methods still hold value in certain settings, particularly in smaller-scale or manual operations where technology adoption may be limited. These traditional systems are often more flexible, requiring less upfront investment and offering a more tactile, hands-on approach to quality evaluation. Still, the growing shift toward digital and automated solutions is gradually replacing the need for traditional inspection methods, especially in large-scale operations where efficiency and precision are paramount.

DISADVANTAGES:

- Limited Dataset: Fruit classifiers require large, diverse datasets to be accurate. If the dataset is not representative of all fruit types, varieties, or conditions, the model may perform poorly on unseen data.

- Environmental Factors: The quality of fruits can vary based on environmental conditions, such as lighting, temperature, or background clutter. A classifier may struggle with accuracy in suboptimal conditions (e.g., poor lighting or images with heavy shadows).

- Texture and Shape Variations: Fruits of the same variety can vary significantly in size, shape, and texture. These variations can make it challenging for the model to classify and assess quality accurately.

- Handling Overfitting: If the model is trained on a small or biased dataset, it may overfit, meaning it performs well on training data but fails to generalize to new, unseen fruits or environments.

- Real-Time Processing Challenges: Deploying the classifier in real-time environments (e.g., in a supermarket or farm setting) can be computationally expensive, requiring powerful hardware and causing delays in processing.

- Misclassification of Damaged Fruits: The system may struggle to identify fruits with slight damage or deformities, leading to misclassification, especially in cases where the damage is subtle.

- Multimodal Data Requirement: To assess fruit quality comprehensively, combining visual, tactile, and sometimes even sensory (e.g., smell, taste) data might be required, which is harder to capture using just image data.

2.3 PROPOSED SYSTEM

Fruit Classifier und Quality Assessment Using Deep Learning To address the limitations of traditional fruit classification and quality assessment methods, propose an advanced system leveraging deep learning technologies. Gather a comprehensive dataset of fruit images encompassing various types and quality levels. Collect images from diverse sources including agricultural fields, marketplaces, and controlled environments. Ensure a wide range of fruit types and quality conditions are represented. Preprocess the images by resizing normalizing and augmenting to enhance the dataset and improve model robustness. Utilize convolutional neural networks CNN for feature extraction and image domification

ADVANTAGES

Efficiency and Scalability Automated processing allows for high throughput, significantly reducing the time and labor required for sorting and grading fruit

3.REQUIRMENT ANALYSIS

3.1 HARDWARE REQUIREMENTS:

CPU type	I5
Ram size	4GB
Hard disk capacity	80 GB
Keyboard type	Internet keyboard
Monitor type	15 Inch colour monitor
CD -drive type	52xmax

3.2 SOFTWARE REQUIREMENTS:

Operating System	Windows 7or later
Simulation Tool	Visual Studio Code
Documentation	Ms – Office

REQUIREMENT ANALYSIS

Requirement Analysis is a crucial phase in the software development lifecycle, focused on identifying, understanding, and documenting the needs and expectations of stakeholders for a project. It serves as the foundation for all subsequent stages of development, ensuring that the final product aligns with the desired goals. This process involves gathering requirements from various stakeholders, including clients, users, and internal teams, and converting them into detailed specifications that guide design and implementation.

A well-conducted requirement analysis minimizes ambiguities, reduces risks, and ensures a clear vision of the project's objectives. During this phase, various techniques such as interviews, surveys, brainstorming sessions, and workshops are employed to gather functional and non-functional requirements. Functional requirements define the specific features or functionalities the system must perform, such as user authentication or report generation, while non-functional requirements specify constraints like performance, scalability, and security.

These requirements are then analyzed for feasibility, prioritized based on importance, and validated with stakeholders to confirm accuracy and completeness. Tools like use case diagrams, user stories, or requirement traceability matrices are often used to structure and visualize the information effectively. Requirement Analysis is iterative and collaborative in nature, involving continuous feedback and adjustments. This ensures that the requirements remain relevant as the project evolves and new insights are gained. The outcome of this process is typically a well-defined Requirements Specification Document (RSD) or a Software Requirements Specification (SRS), which acts as a reference point for designers, developers, and testers throughout the project lifecycle. By bridging the gap between business needs and technical implementation, requirement analysis plays a pivotal role in delivering high-quality software solutions that meet user expectations.

PYTHON

Python is a high-level, interpreted programming language that emphasizes code readability and simplicity. Created by Guido van Rossum in 1991, Python is known for its clean syntax, which makes it easy to learn and use for both beginners and experienced developers. Its versatility allows it to be employed across various domains, such as web development, data analysis, artificial intelligence, machine learning, and scientific computing. With an extensive standard library and a vibrant ecosystem of third-party packages, Python is a powerful tool for tackling a wide range of software tasks. Python's simplicity and readability are key features that set it apart. It supports multiple programming paradigms, including object-oriented, procedural, and functional programming. The language is platform-independent, meaning code written in Python can run on any operating system with minimal modification.

Python also provides dynamic typing and garbage collection, which reduces the complexity of memory management. Its robust community continually develops new libraries and frameworks, such as Django for web development, NumPy for numerical computing, and TensorFlow for machine learning, making Python adaptable to emerging technologies.Python's widespread adoption is attributed to its ease of use and applicability across diverse fields. It is extensively used in data science for data manipulation, visualization, and predictive modeling. In web development, Python's frameworks enable rapid prototyping and scalable solutions. Additionally, Python plays a crucial role in artificial intelligence and automation, helping developers build intelligent systems and streamline workflows. Its popularity among developers, combined with strong community support, has cemented Python as one of the most in-demand and influential programming languages in the tech industry today.

Features in Python

There are many features in Python, some of which are discussed below

- Easy to code
- Free and Open Source

- Object-Oriented Language
- GUI Programming Support
- High-Level Language
- Extensible feature
- Python is Portable language
- Python is Integrated language
- Interpreted Language

ANACONDA

Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source packages can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator,[12] as a graphical alternative to the command line interface (CLI).

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages. It will install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation of, for example, Google Tensorflow, can find that it stops working having used pip to install a different package that requires a different version of the dependent numpy library than the one used by Tensorflow. In some cases, the package may appear to work but produce different results in detail.

In contrast, conda analyses the current environment including everything currently installed, and, together with any version limitations specified (e.g. the user may wish to have Tensorflow version 2,0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Open source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or the user's own private repository or mirror, using the conda install command. Anaconda, Inc. compiles and builds the packages available in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed.

Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories.

The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda.

Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.
The following applications are available by default in Navigator:[16]

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glue
- Orange
- RStudio
- Visual Studio Code

JUPYTER NOTEBOOK

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.
Jupyter Notebook can connect to many kernels to allow programming in different languages. By default, Jupyter Notebook ships with the IPython kernel. As of the 2.3 release[11][12] (October 2014), there are currently 49 Jupyter-compatible kernels for many programming languages, including Python, R, Julia and Haskell.
The Notebook interface was added to IPython in the 0.12 release[14] (December 2011), renamed to Jupyter notebook in 2015 (IPython 4.0 – Jupyter 1.0). Jupyter Notebook is similar to the notebook interface of other programs such as Maple, Mathematica, and SageMath, a computational interface style that originated with Mathematica in the 1980s. According to *The Atlantic*, Jupyter interest overtook the popularity of the Mathematica notebook interface in early 2018.

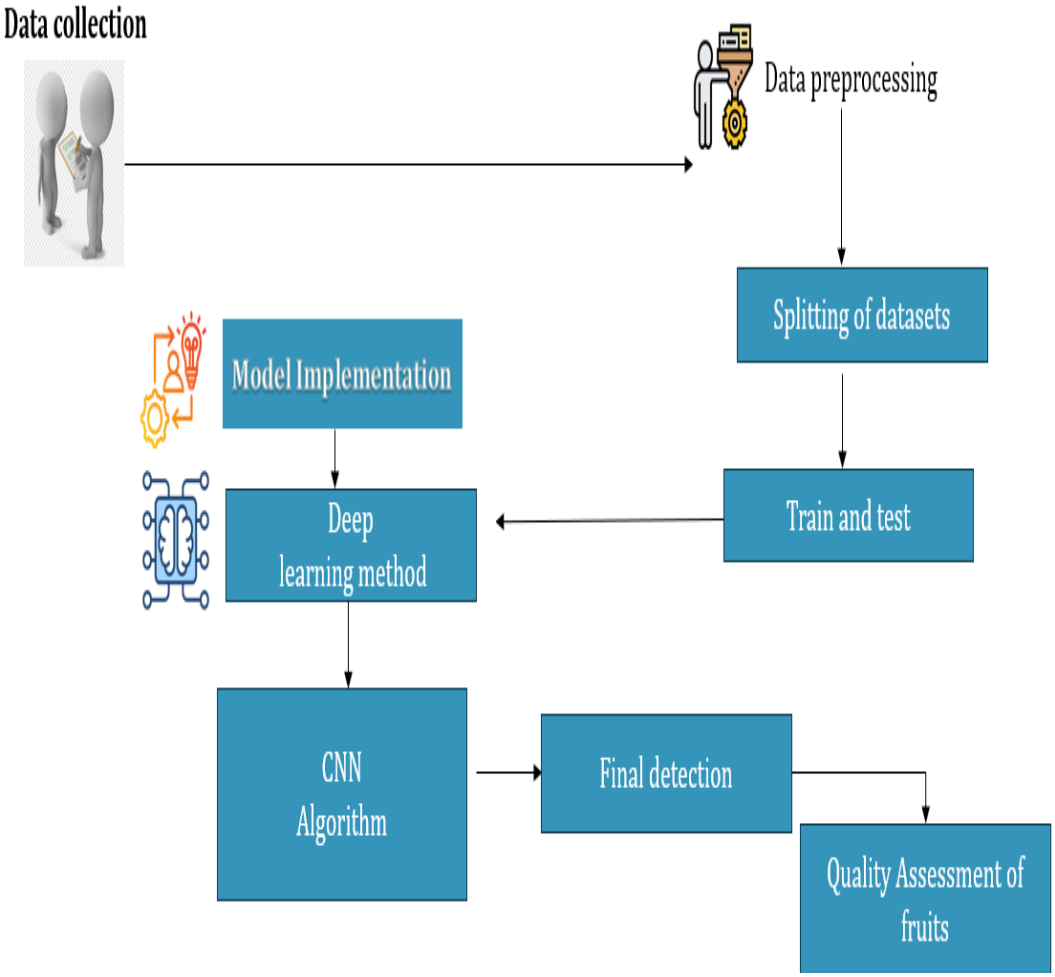
VISUAL STUDIO CODE

Visual Studio Code (VS Code) is a lightweight, open-source code editor developed by Microsoft, designed to streamline the process of writing, debugging, and managing code. It supports a wide range of

programming languages and frameworks, making it a versatile tool for developers. VS Code offers a rich ecosystem of extensions that enhance functionality, such as syntax highlighting, code completion, debugging tools, version control integration (like Git), and language-specific features. Its intuitive user interface, built-in terminal, and seamless integration with cloud-based services have made it a popular choice for software development, web development, and data science. With cross-platform compatibility, it can run on Windows, macOS, and Linux, providing developers with flexibility and efficiency in their workflow.

4.DESIGN AND IMPLEMENTATION

4.1ARCHITECTURE DIAGRAM



.2 DATAFLOW DIAGRAM

Level 0: Context Diagram (High-level DFD)

This diagram shows the system at a high level, focusing on interactions between external entities and the fruit classification system.

Entities:

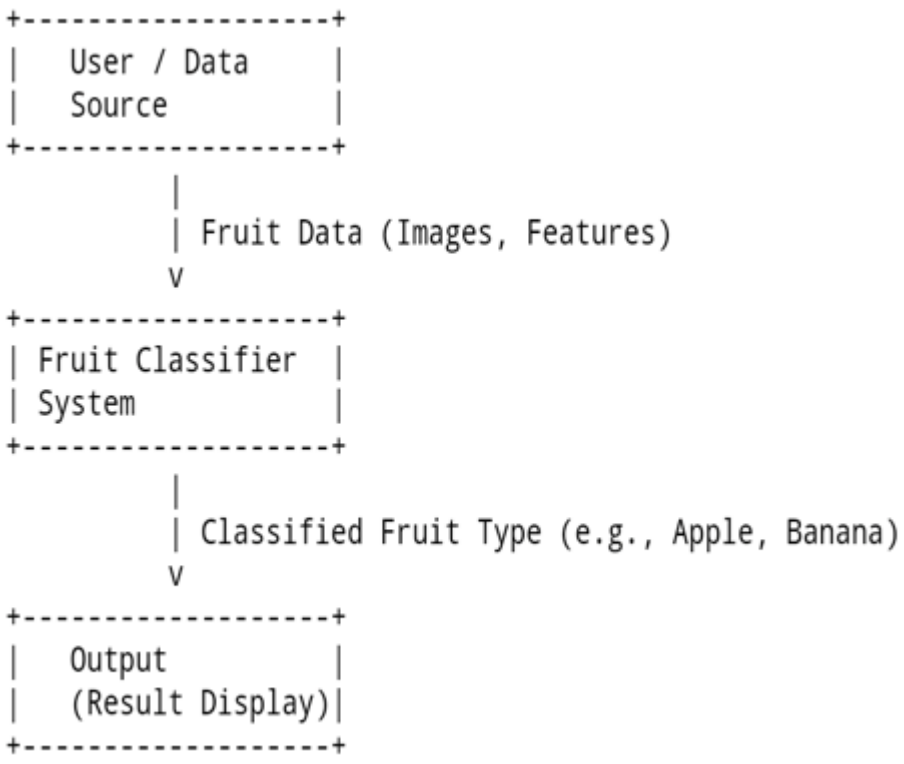
User (or Data Source) – Provides the raw input data (images or features of fruits). Fruit Classifier System – This is the machine learning-based system that processes the input data to classify the fruit.

Process:

Fruit Classification – The main process where the input data is processed using a machine learning model to classify the fruit.

Data Flow:

Input: Fruit data (images, features, etc.) from the User/Data Source. Output: Classified fruit type (e.g., Apple, Banana, Orange).



Level 1: Decomposition Diagram (Detailed DFD)

In this level, we break down the Fruit Classifier System into smaller sub-processes.

Data Input Collection:

Collects the raw fruit data from the user, which could be images, videos, or sensor data. **Input**: Raw fruit data.

Output: Data to be preprocessed.

Preprocessing:

Handles raw data, preparing it for classification by normalizing or resizing images, cleaning data, or scaling numerical features.

- Input: Raw data.
- Output: Preprocessed data (ready for feature extraction or direct use).

Feature Extraction:

- If necessary, this step extracts specific features from images or other data (e.g., texture, color, shape).

Input: Preprocessed data.

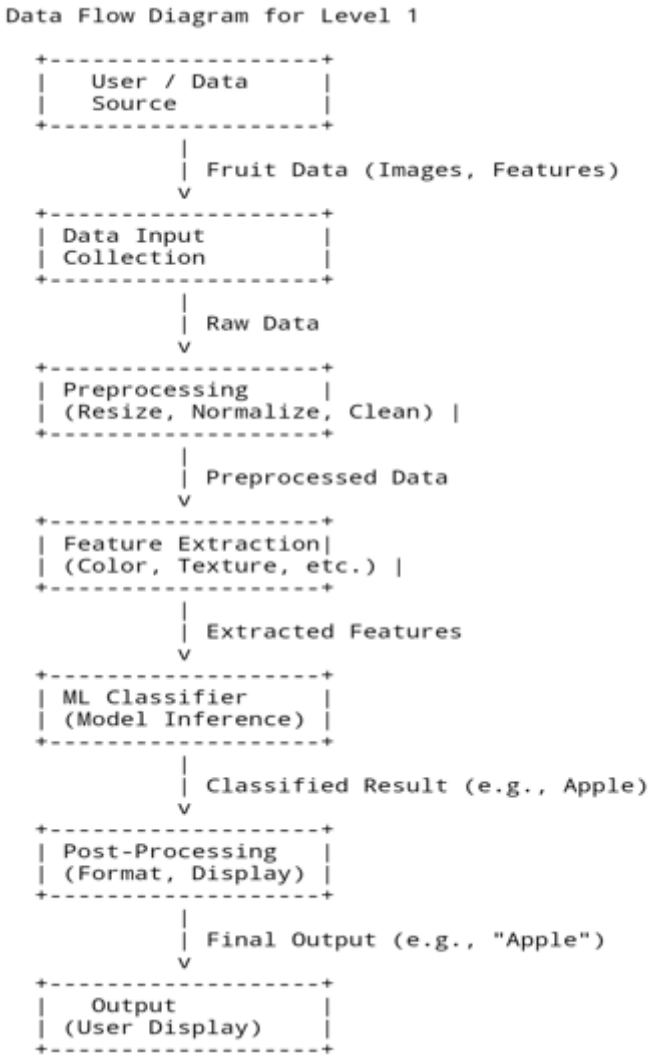
- **Output**: Extracted features (e.g., color histograms, texture, shape).
- Model Inference (Machine Learning Classifier):
- This is where the trained machine learning model classifies the fruit using the preprocessed data or extracted features.
- **Input**: Features or preprocessed data.
- **Output**: Classification result (e.g., "Apple," "Banana").
- **Post-processing:**

The classification result might be enhanced or formatted for output, such as displaying confidence scores or labels to the user.

- Input: Classification result.
- Output: Display the final result to the user.
- **Post-Processing:**

Label the prediction (e.g., "Confidence: 90%").

- Display the result to the user in a suitable format.
- Example of Data Flow at Level 2
- **Image Preprocessing:** Resize, normalize.
- **Feature Extraction**: Extract shape, color, and texture features.
- **ML Model**: Use the model to classify the fruit type based on features.
- **Post-Processing:** Display results with confidence scores.



Level 2: Further Decomposition (Optional)

At Level 2, we can break down the sub-processes into more specific tasks. For example, Preprocessing might involve multiple steps:

Image Preprocessing:

Resize, crop, or normalize images.

Convert to grayscale or extract channels.

Feature Extraction:

For images, this could include extracting texture (e.g., HOG, Gabor filters), shape, or color histograms.

Machine Learning Model:

Training: The model is trained using labeled fruit data (images with known classifications).

Inference: The trained model performs predictions based on input features

Post-Processing:

Label the prediction (e.g., “Confidence: 90%”).

Display the result to the user in a suitable format.

Example of Data Flow at Level 2

Image Preprocessing: Resize, normalize.

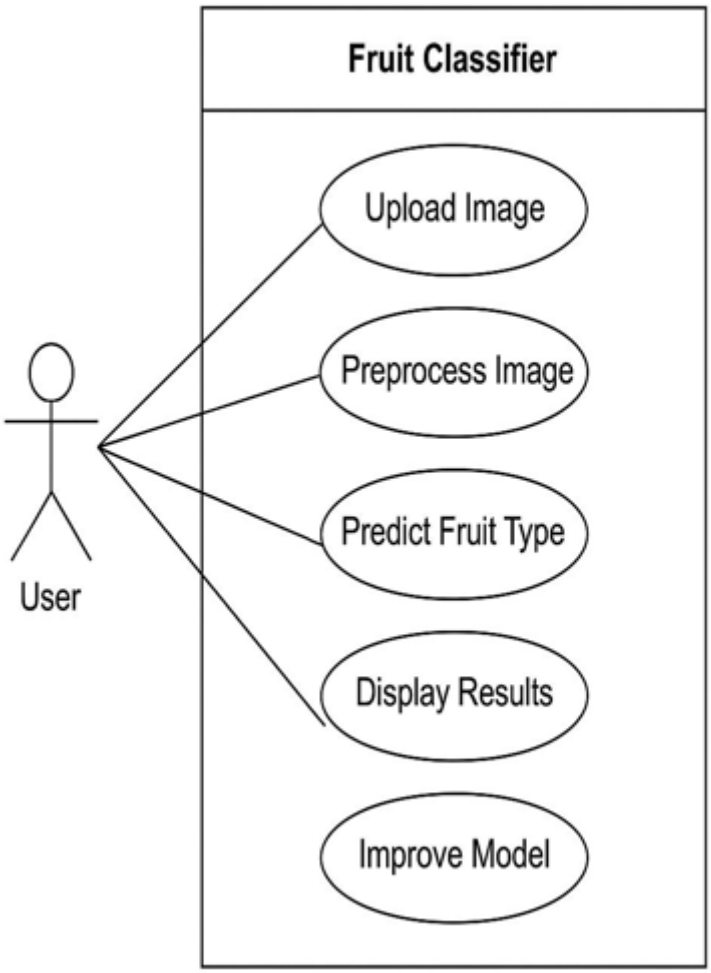
Feature Extraction: Extract shape, color, and texture features.

ML Model: Use the model to classify the fruit type based on features.
Post-Processing: Display results with confidence scores.

4.3 USE CASE DIAGRAM

▸ USER CASE DIAGRAM:

▸ Actors: User, System (ML Model)
Use Cases: Upload Image, Preprocess Image, Predict Fruit Type, Display Results, Improve Model



4.4 SEQUENCE DIAGRAM

▸ Shows the step-by-step interaction between components.

1. User uploads an image

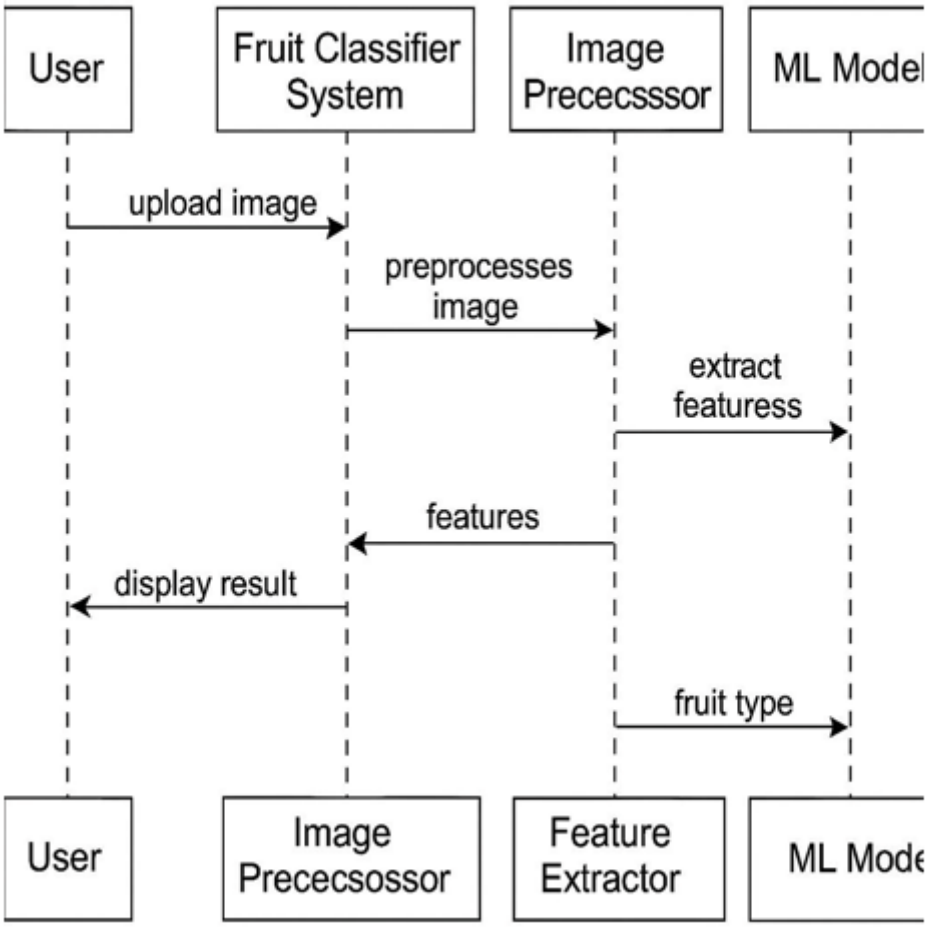
2. Image Processor preprocesses image

3. Feature Extractor extracts key features

4. Classifier Model predicts fruit type

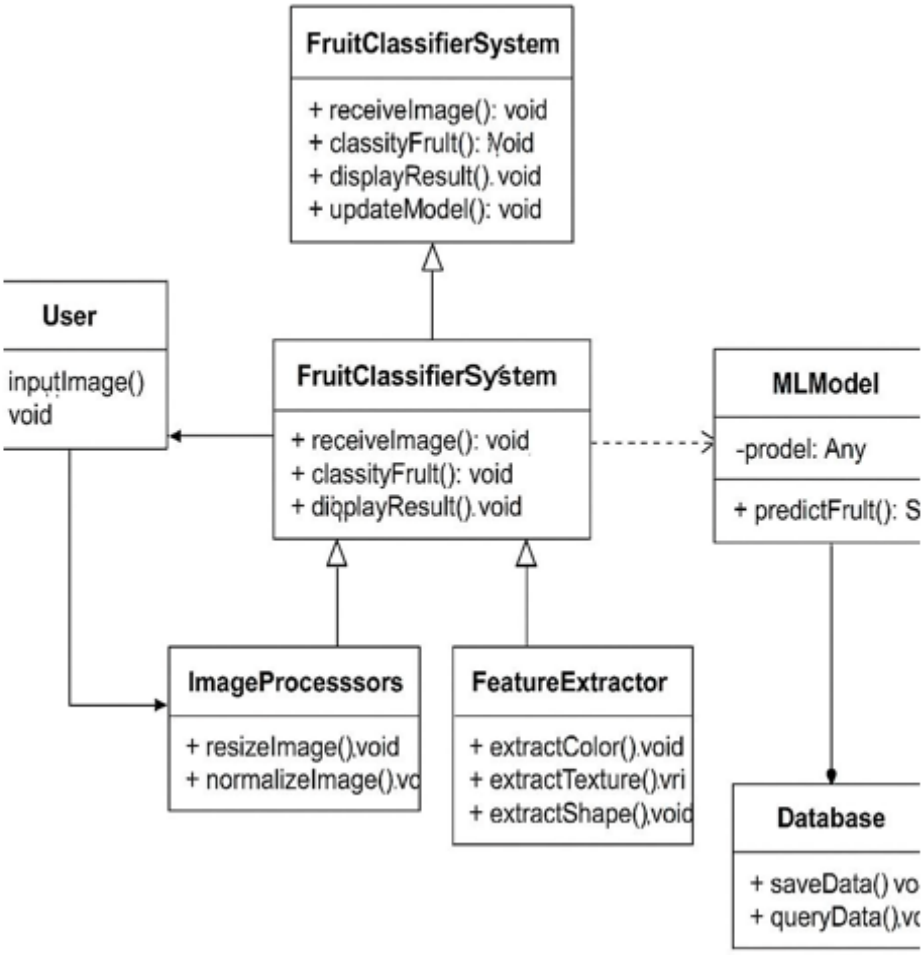
5. Database stores/retrieves data

6. System displays result to user



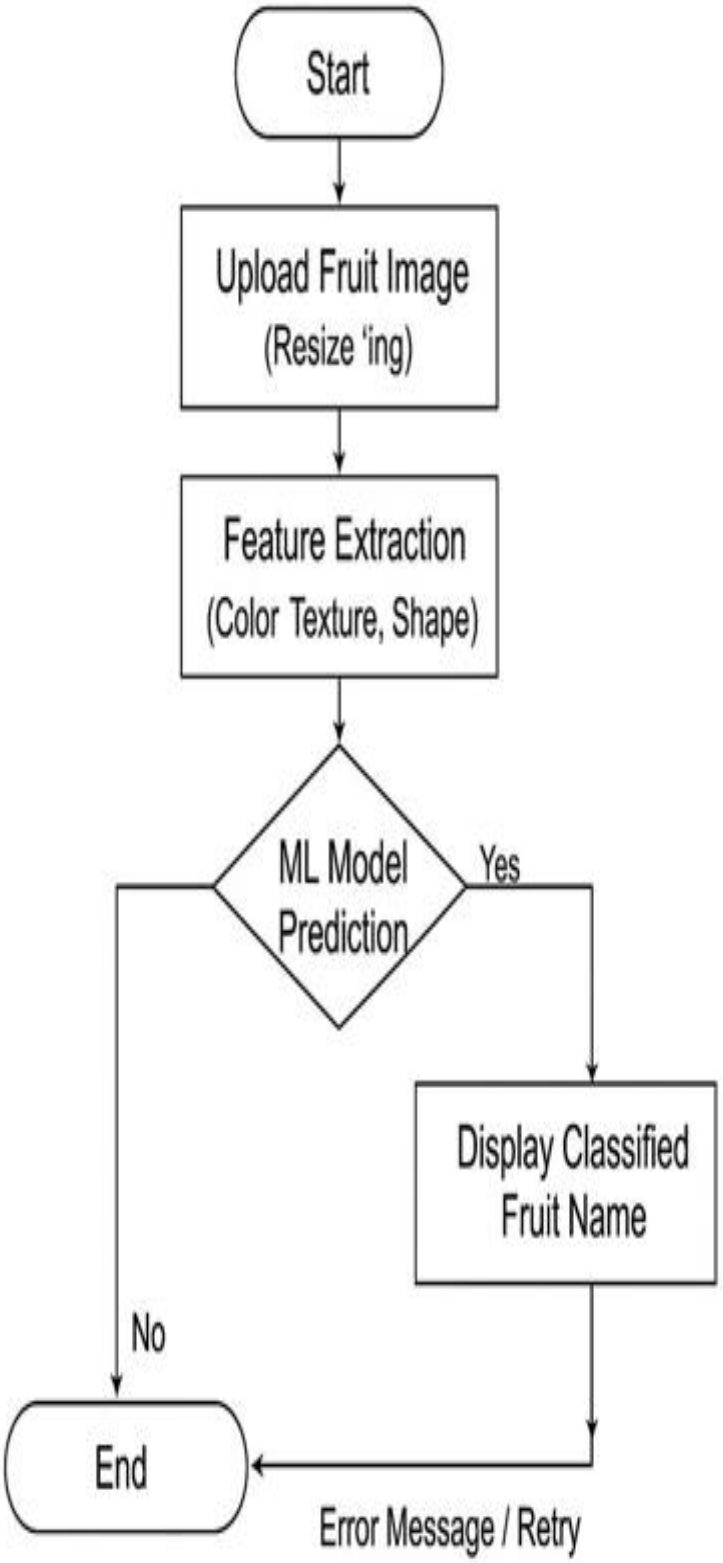
4.5 COLLABORATIVE DIAGRAM

- Classes and Relationships:
- User (Interacts with the system)
- Fruit Classifier System (Main system handling input and output)
- Image Preprocessor (Handles image resizing, normalization)
- Feature Extractor (Extracts color, texture, and shape features)
- ML Model (Trained model for classification)
- Database (Stores labeled fruit data and user queries)



4.5 DATABASE DESIGN

- Represents workflow of fruit classification:
- 1. Start → Upload image
- 2. Preprocessing → Resize, De-noise
- 3. Feature Extraction → Extract color, shape, texture
- 4. ML Model Prediction → Classify fruit
- 5. Store result in database
- 6. Display result to user
- 7. End


Start
Upload Fruit Image
(Resize 'ing)
Feature Extraction
(Color Texture, Shape)
ML Model
Prediction
Yes
No
Display Classified
Fruit Name
End
Error Message / Retry

5.TESTING

5.1 INTRODUCTION

Testing is a crucial phase in software development that ensures the functionality, performance, and reliability of an application. It involves systematically identifying and fixing defects before the software is deployed. Testing can be manual or automated, and it follows various methodologies, such as unit testing, integration testing, system testing, and user acceptance testing (UAT). By implementing thorough testing strategies, developers can minimize errors, enhance security, and improve user experience.

There are different types of testing, each serving a unique purpose. Functional testing ensures that the software meets specified requirements, while non-functional testing evaluates aspects such as performance, scalability, and security. White-box testing examines internal structures, whereas black-box testing focuses on external functionalities without knowledge of the internal code. Automated testing uses scripts and tools to run repetitive tests efficiently, whereas manual testing requires human intervention to validate software behavior. A well-structured test plan helps teams detect and resolve issues early, reducing the cost of fixing defects later in development.

Testing plays a vital role in delivering high-quality software and maintaining customer satisfaction. It helps prevent critical failures, reduces downtime, and ensures compliance with industry standards. Proper testing enhances software performance, making it robust and reliable under various conditions. Without adequate testing, software may contain undetected bugs that could lead to security vulnerabilities or operational failures. As software development evolves, continuous testing and DevOps practices integrate testing into the development pipeline, ensuring faster releases and improved software quality.

5.2 TYPES OF TESTS

UNIT TESTING

Unit testing is a software testing technique that focuses on testing individual components or functions of a program in isolation. These tests are typically automated and are designed to ensure that specific sections of code function correctly. By testing small, independent units of a system, developers can identify and fix errors early in the development cycle, reducing the risk of larger system failures. The primary goal of unit testing is to validate that each unit of software performs as expected under various conditions. Each test case is designed to cover different scenarios, including edge cases, normal inputs, and unexpected inputs. This helps in verifying the correctness of code, improving its maintainability, and enabling easier refactoring.

Unit tests also serve as documentation for the expected behavior of functions, making it easier for new developers to understand and work with the codebase. Unit testing is an essential part of test-driven development (TDD), where tests are written before the actual code implementation. This approach ensures that new code does not introduce regressions and that modifications maintain expected functionality. While unit testing alone does not guarantee a bug-free application, it significantly reduces the likelihood of defects by catching issues early. Combined with integration and system testing, unit testing forms the foundation of a robust software quality assurance process.

INTEGRATION TESTING

Integration Testing is a type of software testing where individual components or modules are combined and tested as a group to ensure they work together correctly. Unlike unit testing, which focuses on isolated components, integration testing checks how these components interact, exchange data, and function as a whole. It helps identify defects in communication between modules, such as incorrect data passing, API failures, or interface mismatches. This testing phase is crucial, especially in complex systems where multiple services, databases, or third-party integrations are involved. There are several approaches to integration testing, including Big Bang, Top-Down, Bottom-Up, and Hybrid testing.

The Big Bang approach integrates all components at once and tests the system as a whole, while Top-Down and Bottom-Up approaches incrementally test higher-level and lower-level modules, respectively. Hybrid testing combines both methods for better flexibility. Testers use stubs (temporary replacements for missing components) and drivers (test modules that provide input) to simulate incomplete parts of the system during testing.

Integration testing helps detect issues early in development, reducing costly fixes later. It ensures the system functions as intended when different modules interact, improving software reliability and performance. Automated tools like Selenium, Postman, and Junit can assist in running integration tests efficiently. Proper integration testing is essential for delivering seamless, high-quality software that works across different environments and meets user expectations.

FUNCTIONAL TEST

A Functional test is a type of software testing that evaluates whether an application meets its specified requirements by testing its features and functionalities. It focuses on verifying that each function of the software behaves as expected, based on predefined test cases. Functional testing typically involves testing user interactions, system workflows, and business logic to ensure the software performs correctly under normal and edge-case scenarios. This type of testing is usually conducted using both manual and automated techniques to validate various aspects of the application.

Functional testing is carried out by simulating real-world use cases and checking whether the system responds as intended. It follows a black-box testing approach, meaning the testers do not need to understand the internal code structure; instead, they assess input and output behaviors. Key types of functional testing include unit testing, integration testing, system testing, and user acceptance testing (UAT). Test cases are designed based on functional requirements, ensuring that user expectations and business goals are met efficiently.

The importance of functional testing lies in preventing defects and ensuring a seamless user experience. It helps identify issues related to incorrect calculations, broken workflows, UI inconsistencies, or integration failures. By thoroughly testing functionalities before deployment, companies can reduce the risk of software failures, improve quality assurance, and enhance customer satisfaction. Automated functional testing tools, such as Selenium, Cypress, and Test Complete, aid in performing repetitive tests efficiently, making the process faster and more reliable.

SYSTEM TEST

System testing is a crucial phase in the software development lifecycle (SDLC) where the entire application is tested as a complete system. It ensures that the integrated components function as expected in a real-world environment. The objective of system testing is to validate the application against the

specified requirements and identify any defects before deployment. This testing is performed after integration testing and before user acceptance testing (UAT), covering both functional and non-functional aspects of the system.System testing involves verifying end-to-end workflows, data integrity, security, performance, and compatibility across different environments. It20sabilies various types of testing such as functional testing, performance testing, security testing,20sability testing, and recovery testing. Test cases are designed based on real-world scenarios to ensure that the system meets business objectives. Testers use automated and manual techniques to validate software behavior under different conditions, ensuring that all components interact correctly.

WHITE BOX TESTING

White Box Testing, also known as Clear Box Testing, Glass Box Testing, or Structural Testing, is a software testing technique where the internal structure, code, and logic of the application are examined. Unlike Black Box Testing, which focuses on functionality without knowledge of the internal code, White Box Testing requires a deep understanding of the program's source code. Testers use this method to verify control flow, data flow, loops, and security vulnerabilities within the application. It is commonly performed by developers or software engineers during unit testing or integration testing phases. This testing approach involves techniques such as statement coverage, branch coverage, path coverage, and condition coverage, ensuring every line of code is tested at least once. By using White Box Testing, testers can identify hidden bugs, optimize code performance, and ensure the security of an application. Various tools like Junit, Selenium, CppUnit, and PyTest are used to automate and streamline the testing process. Since it requires coding knowledge, White Box Testing is best suited for experienced testers who can analyze and interpret the software's architecture and logic.

BLACK BOX TESTING.

Black Box Testing is a software testing method that evaluates the functionality of an application without examining its internal code, structure, or implementation details. Testers interact with the software by providing inputs and verifying outputs to ensure it meets the specified requirements. This approach mimics the end-user experience, making it useful for detecting functional issues, usability concerns, and missing features. Since testers do not require programming knowledge, it allows independent validation of software functionality by quality assurance teams.

This testing technique is classified into several types, including functional testing, non-functional testing, and regression testing. Functional testing checks whether the system performs as expected based on given specifications. Non-functional testing evaluates performance, security, and usability aspects. Regression testing ensures that new updates do not introduce defects into existing functionalities. Common methods used in black box testing include equivalence partitioning, boundary value analysis, and decision table testing, which help optimize test coverage and efficiency. Black Box Testing is widely applied in different stages of the software development life cycle, particularly in system testing and acceptance testing. It is beneficial in ensuring that an application meets user expectations, regulatory requirements, and business objectives. However, since it does not analyze internal code logic, it may not uncover hidden defects like memory leaks or security vulnerabilities that require white box testing techniques. Despite this limitation, black box testing remains a crucial part of delivering high-quality, reliable, and user-friendly softwareTesting is a crucial phase in software development that ensures the functionality, performance, and reliability of an application. It involves systematically identifying and fixing defects before the

software is deployed. Testing can be manual or automated, and it follows various methodologies, such as unit testing, integration testing, system testing, and user acceptance testing (UAT). By implementing thorough testing strategies, developers can minimize errors, enhance security, and improve user experience.There are different types of testing, each serving a unique purpose. Functional testing ensures that the software meets specified requirements, while non-functional testing evaluates aspects such as performance, scalability, and security. White-box testing examines internal structures, whereas black-box testing focuses on external functionalities without knowledge of the internal code. Automated testing uses scripts and tools to run repetitive tests efficiently, whereas manual testing requires human intervention to validate software behavior. A well-structured test plan helps teams detect and resolve issues early, reducing the cost of fixing defects later in development.

Testing plays a vital role in delivering high-quality software and maintaining customer satisfaction. It helps prevent critical failures, reduces downtime, and ensures compliance with industry standards. Proper testing enhances software performance, making it robust and reliable under various conditions. Without adequate testing, software may contain undetected bugs that could lead to security vulnerabilities or operational failures. As software development evolves, continuous testing and DevOps practices integrate testing into the development pipeline, ensuring faster releases and improved software quality.

Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

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.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

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.

Test Results: All the test cases mentioned above passed successfully. No defects encountered

5.3 TEST CASES

TEST PLAN ID	TESTING MODULE	TESTING TYPE	DURATION	TEST DESCRIPTION
TP1	Admin Login	Unit Testing	1 hour	Admin correct Login name and Password than Login
TP2	Dataset Upload	Validation Testing	3 hours	To Upload a New Dataset to Database Storage.
TP3	List out Uploaded Dataset	Integration Testing	1 hour	To view a Uploaded all data's
TP4	View all user information	Block Box Testing	3 hours	Only view for admin side, in overall results
TP5	Admin view form Graph	Block Box Testing	4 hours	Admin view form positive or negative result
TP6	User New Registration	Unit Testing	3 hours	User new registration a address and user full details in enter the registration form
TP7	User search cities	Unit Testing	2 hours	User first login than search
TP8	View the result	Integration Testing	1 hours	View the result and details
TP9	User check status	Unit Testing	2 hours	User check status and result

5.4Test Data

5.4.1User New Registration

TEST CASE ID	TEST PLAN ID	TEST CASE DESCRIPTION	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC1	TP1	User registration New Account	User enter the all text field for correct Details	Your Account was Registered successfully	Your Account was Registered successfully	Success
TC2	TP1	Without filling text field	User wrong entered details	Display an error message “Please fill Out this field”	Your Account is Registered successfully	Fail
TC3	TP1	Input to invalid Email id	User mail id invalid mail id or already created	Display error message “Please Provide a valid Email id”	Your Account is Registered successfully	Fail

5.4.2 Admin Upload datasets and view Results

TEST CASE ID	TEST PLAN ID	TEST CASE DESCRIPTION	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC1	TP2	Admin add dataset	Upload	Your Upload was successfully	Upload was successfully	Success
TC2	TP2	Login a Invalid Admin name and Password	Login	Display an error message “Invalid User name and password”	Login was successfully	Fail

TC3	TP2	View overall Results	View Overall result Tagging	View The Results	View The Results in	Success

User Search

TEST CASE ID	TEST PLAN ID	TEST CASE DESCRIPTION	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC1	TP3	User Search Cities	View result	successfully	successfully	Success
TC2	TP3	Login an Invalid Admin name and Password	User name or password incorrect	Display an error message “Invalid User name and password”	Login was successfully	Fail
TC3	TP3	View overall Results	View Overall result Tagging	View The Results	View The Results in	Success

5.5 TEST REPORT

TEST CASE ID	TEST PLAN ID	TEST CASE DESCRIPTION	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC1	TP4	Login an Invalid User name and Password	User name or password incorrect	Display an error message “Invalid User name and password”	Login was successfully	Fail

6.SYSTEM IMPLEMENTATION

6.1 MODULE DESCRIPTION

Module Description for Fruit Classifier Based on Machine Learning Analysis

TheFruit Classifier Module is a core component of the system designed to accurately identify and categorize different fruits from input images using machine learning techniques. This module encompasses all stages from data ingestion to output prediction, ensuring a seamless and efficient classification process.

Data Acquisition and Preprocessing

- Image Input: Accepts images of fruits captured via camera or uploaded files.
- Preprocessing Steps:
- Resize images to a standardized resolution.
- Normalize pixel intensity values for uniformity.
- Apply data augmentation techniques (rotation, flipping, brightness adjustments) to enhance model robustness.

Feature Extraction

Traditional Methods: Extract color histograms, shape descriptors, and texture features.
Deep Learning Approach: Utilize convolutional neural networks (CNNs) to automatically learn relevant features directly from raw images.

Model Training and Optimization

Model Selection: Implements classifiers such as CNN architectures (e.g., VGG, ResNet), SVM, Random Forest, or k-NN.

Training Process

- Uses labeled datasets to train the model.
- Employs validation techniques like cross-validation to tune hyperparameters.
- Performance Evaluation: Monitors accuracy, precision, recall, and F1-score on validation data.

Prediction and Classification

- Inference: Processes new input images through the trained model.
- Output: Provides the predicted fruit category with confidence scores.

Post-processing and Output Delivery

- Presents classification results in a user-friendly format.
- Optionally, overlays predicted labels on input images or provides additional information about the fruit.

Model Maintenance

- Includes capabilities for retraining with new data.

- Implements performance monitoring to detect model drift over time.

Summary

This module is the intelligent core of the fruit classification system, transforming raw images into accurate fruit labels through advanced machine learning techniques. It ensures high accuracy, adaptability to new data, and efficient processing suitable for real-world applications.

6.2 ALGORITHMS

Convolutional Neural Networks (CNN)

Introduction to Deep Learning

Deep learning is a subset of machine learning that focuses on algorithms inspired by the structure and function of the brain, known as artificial neural networks. It has gained prominence due to its superior performance in various domains such as image recognition, speech processing, and natural language processing.

What is CNN?

A Convolutional Neural Network (CNN) is a type of deep neural network designed to process data with a grid-like topology, such as images. CNNs are widely used in visual recognition tasks and are known for their ability to automatically and adaptively learn spatial hierarchies of features from input images.

Architecture of CNN

The architecture of a CNN is typically composed of a sequence of layers, each performing specific operations. These include convolution layers, activation functions, pooling layers, and fully connected layers.

Convolution Layer

This layer applies a number of filters to the input image, creating feature maps that represent different aspects of the image such as edges, textures, and shapes.

Activation Function (ReLU)

ReLU (Rectified Linear Unit) is commonly used as an activation function in CNNs. It introduces non-linearity into the model by outputting zero for negative values and the input itself for positive values.

Pooling Layer

Pooling layers are used to reduce the spatial dimensions of the feature maps. This helps in reducing the number of parameters and computation in the network.

Fully Connected Layer

These layers connect every neuron in one layer to every neuron in another layer and are typically used towards the end of the CNN to perform classification based on the extracted features.

Mathematics Behind CNN

CNN operations involve mathematical concepts such as matrix multiplication and convolution. The convolution operation applies filters to the input to produce feature maps. The learning process involves adjusting the filter values using backpropagation and gradient descent to minimize the error.

Advantages of CNN

CNNs require fewer parameters compared to fully connected networks, they are effective in automatically learning spatial hierarchies of features, and are highly efficient for image processing tasks.

Applications of CNN

CNNs are used in a variety of applications including image classification, object detection, face recognition, and medical image analysis. Their ability to learn from raw image data makes them powerful tools for computer vision tasks.

Example: Handwritten Digit Recognition (MNIST)

The MNIST dataset is a collection of 28x28 pixel images of handwritten digits. CNNs can be trained to achieve high accuracy on this dataset by learning to recognize the distinct features of each digit.

Implementation Using Python & TensorFlow

Here's a simple CNN implementation for digit classification:

```python
import tensorflow as tf
from tensorflow.keras import layers, models
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Challenges and Limitations

CNNs can be data-hungry and computationally intensive. They may also suffer from overfitting if not properly regularized, and they require substantial hardware resources for training large models.

Conclusion & Future Scope

CNNs have revolutionized the field of computer vision. With advancements in hardware and algorithms, their applications are expected to expand further into fields such as autonomous driving, augmented reality, and beyond.

CNN Variants

There are several CNN variants developed to improve performance and adapt to specific tasks. Some popular ones include:

- LeNet-5: One of the earliest CNNs, used for digit recognition.
- AlexNet: A deep CNN that won the ImageNet competition in 2012.
- VGGNet: Known for its simplicity and use of 3x3 filters.
- GoogLeNet (Inception): Introduced the inception module to optimize computation.
- ResNet: Utilizes residual connections to train very deep networks.

Detailed Example: CNN on CIFAR-10

CIFAR-10 is a dataset of 60,000 32x32 color images in 10 classes. Here's a step-by-step breakdown of using CNN on this dataset:

1. Load and preprocess the data.
2. Build a CNN model using Conv2D, MaxPooling, Flatten, Dense layers.
3. Compile the model using categorical crossentropy and an optimizer like Adam.
4. Train the model and evaluate its accuracy.

Visualizing CNN Filters

One important advantage of CNNs is their interpretability. The learned filters in the convolution layers can be visualized to understand what features the model is learning. Early layers might detect edges and textures, while deeper layers detect shapes and objects.

Transfer Learning

Transfer learning allows using a pretrained CNN model and fine-tuning it for a new but similar task. This is useful when there’s limited data available. Common pretrained models include VGG16, ResNet50, and MobileNet.

CNN in Real-World Applications

CNNs are used in many industries:

- Healthcare: Detecting diseases from X-rays and MRIs.
- Automotive: Used in self-driving car systems for object detection.
- Agriculture: Classifying crop diseases.
- Retail: Customer behavior analysis via video feeds.

Tools and Frameworks

Popular tools and frameworks for building CNNs include:

- TensorFlow
- Keras
- PyTorch
- OpenCV (for preprocessing)
- scikit-learn (for evaluation and ML utilities)

Summary of CNN Workflow

The general workflow for using CNNs is:

1. Data Collection and Preprocessing
2. Designing the CNN Architecture
3. Training the Model
4. Evaluation and Optimization
5. Deployment

Future Trends in CNN Research

Emerging research trends in CNNs include:

- Efficient models for edge computing
- Self-supervised learning
- Hybrid models combining CNNs with transformers
- Explainable AI for better interpretability

RAIN FOREST ALGORITHM

Creating a fruit classifier using a machine learning approach with a Random Forest algorithm involves several key steps. Here's a comprehensive overview to help you develop such a system:

1. Data Collection

Gather a labeled dataset of fruit images. You can use public datasets like:

- Fruits 360 Dataset
- Custom datasets collected via camera or smartphone

Ensure the dataset contains diverse images of various fruits under different conditions.

2. Data Preprocessing

- **Resize Images:** Standardize image sizes (e.g., 100x100 pixels).
- **Color Space Conversion:** Convert images to a suitable color space (RGB or HSV).
- **Data Augmentation:** Apply transformations (rotation, flipping, zoom) to increase dataset diversity
- **Label Encoding:** Convert fruit labels into numerical format.

3. Feature Extraction

Since Random Forests work with tabular data, you'll need to extract features from images:

- **Color Histograms:** Distribution of colors.
- **Texture Features:** Using methods like Local Binary Patterns (LBP).
- **Shape Features:** Contours, aspect ratio, perimeter, etc.
- Alternatively, you can use deep features from pre-trained CNNs (transfer learning).

- Collect and preprocess data.
- Extract meaningful features.
- Train a Random Forest classifier.

- Evaluate and optimize the model.
- Deploy for real-world fruit classificatio

7.CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

In conclusion, the Fruit Classifier and Quality system using deep learning demonstrates a significant advancement in automating the classification and quality assessment of fruits. By leveraging convolutional neural networks (CNNs), the system is able to accurately classify various fruit types based on images, ensuring efficient sorting and categorization. Additionally, the integration of deep learning techniques allows for the analysis of fruit quality by detecting defects, ripeness levels, and overall condition, providing valuable insights for both producers and consumers. This system not only enhances the accuracy and speed of fruit classification but also plays a vital role in reducing human error and improving the overall efficiency of agricultural supply chains. The combination of classification and quality evaluation in one framework showcases the potential of AI in the food industry, promoting better standards of quality control and ensuring a consistent supply of fresh, high-quality produce.

7.2 FUTURE ENHANCEMENT

A Fruit Classifier and Quality Detection System using deep learning leverages advanced techniques to not only identify various types of fruits but also assess their quality. By employing Convolutional Neural Networks (CNNs), the system can analyze images of fruits and classify them into categories based on features such as color, shape, texture, and size. Additionally, the quality assessment involves evaluating ripeness, blemishes, firmness, and other physical attributes to determine whether the fruit meets specific quality standards. The system can be trained using a large dataset of labeled fruit images with quality annotations, improving its accuracy through feature enhancement techniques like image augmentation, transfer learning, and data normalization. By utilizing these techniques, the model becomes more robust, able to handle variations in lighting, angles, and backgrounds, ensuring reliable classification and quality prediction in real-world scenarios.

8.APPENDICES

8.1 CODING

UPLOAD DATASETS

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title></title>
   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
crossorigin="anonymous">
   <style media="screen">
      html, body {
         height: 100vh;
         margin: 0;
         padding: 0;
      }
      .jumbotron {
         margin-top: 20px;
         background-image: url("../static/wl3.jpg");
         background-size: cover;
         color: rgb(0, 0, 0);
         height: 100%;
         display: flex;
         justify-content: flex-end;
         align-items: center;
      }
         .jumbotron h1,
         .jumbotron strong {
         color: #ff0000; /* Change this to the color you desire */
   }
   </style>
</head>
<body style="background-image: linear-gradient(to right bottom, #ffffff);
 text-align: center; align:center; background-color:#ffffff; background-repeat:no-repeat;">

   <div class="jumbotron">
      {% block content %}
      {% endblock %}
```

```
    </div>

    <!-- JS, Popper.js, and jQuery -->
                <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
                    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" integrity="sha384-
OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"
crossorigin="anonymous"></script>
</body>
</html>
```

HYBERNATE CONFIGURATION FILE

```
!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>CNN vs Random Forest Algorithm Comparison</title>
  <style>
    @import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600;700&display=swap');

    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Poppins', sans-serif;
      background-image: url('./static/wl4.jpg');
      background-size: cover;
      background-attachment: fixed;
      background-position: center;
      color: #333;
      position: relative;
      min-height: 100vh;
    }
```

```
body::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(255, 255, 255, 0.85);
  z-index: -1;
}

   /* Button Container Styling */
.button-container {
  display: flex;
  justify-content: center;
  width: 100%;
  margin: 30px 0;
}

/* Button Styling */
.btn-primary {
  background: linear-gradient(135deg, #3498db, #9b59b6);
  color: white;
  border: none;
  padding: 12px 30px;
  font-size: 1rem;
  font-weight: 600;
  font-family: 'Poppins', sans-serif;
  border-radius: 50px;
  cursor: pointer;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
  transition: all 0.3s ease;
  text-transform: uppercase;
  letter-spacing: 1px;
}

.btn-primary:hover {
  transform: translateY(-5px);
  box-shadow: 0 7px 20px rgba(0, 0, 0, 0.3);
  background: linear-gradient(135deg, #2980b9, #8e44ad);
}
```

```
.btn-primary:active {
  transform: translateY(-2px);
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
}

/* Focus state for accessibility */
.btn-primary:focus {
  outline: none;
  box-shadow: 0 0 0 3px rgba(52, 152, 219, 0.4), 0 4px 15px rgba(0, 0, 0, 0.2);
}
.page-container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 40px 20px;
}

header {
  text-align: center;
  margin-bottom: 50px;
  position: relative;
}

h1 {
  font-size: 2.5rem;
  color: #2c3e50;
  margin-bottom: 15px;
  position: relative;
  display: inline-block;
}

h1::after {
  content: '';
  position: absolute;
  bottom: -10px;
  left: 50%;
  transform: translateX(-50%);
  width: 80px;
  height: 4px;
  background: linear-gradient(90deg, #3498db, #9b59b6);
  border-radius: 2px;
}

.subtitle {
```

```
  font-size: 1.2rem;
  color: #7f8c8d;
  font-weight: 300;
}

.container {
  display: flex;
  flex-wrap: wrap;
  gap: 30px;
  justify-content: center;
}

.column {
  display: flex;
  flex-direction: column;
  gap: 30px;
  flex: 1;
  max-width: calc(50% - 15px);
}

.column-header {
  text-align: center;
  font-size: 1.8rem;
  margin-bottom: 15px;
  color: #2c3e50;
  font-weight: 600;
  padding: 15px 0;
  border-radius: 10px;
  background: linear-gradient(135deg, rgba(255,255,255,0.9), rgba(255,255,255,0.6));
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
  height: 70px;
  display: flex;
  align-items: center;
  justify-content: center;
}

.image-box {
  background: rgba(255, 255, 255, 0.9);
  border-radius: 15px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
  overflow: hidden;
  transition: all 0.3s ease;
  height: 450px;
```

```
  display: flex;
  flex-direction: column;
}

.image-box:hover {
  transform: translateY(-10px);
  box-shadow: 0 15px 35px rgba(0, 0, 0, 0.15);
}

.image-title {
  font-weight: 600;
  font-size: 1.2rem;
  padding: 20px;
  text-align: center;
  color: #2c3e50;
  background: linear-gradient(to right, #f5f7fa, #c3cfe2);
  flex-shrink: 0;
  height: 80px;
  display: flex;
  align-items: center;
  justify-content: center;
}

.image-content {
  flex: 1;
  position: relative;
  overflow: hidden;
  display: flex;
  align-items: center;
  justify-content: center;
}

.image-box img {
  width: 100%;
  height: auto;
  max-height: 100%;
  object-fit: contain;
  display: block;
  transition: transform 0.3s ease;
}

.image-box:hover img {
  transform: scale(1.05);
```

```
}

.image-caption {
  padding: 20px;
  text-align: center;
  font-size: 0.95rem;
  color: #555;
  line-height: 1.6;
  flex-shrink: 0;
  height: 80px;
  overflow-y: auto;
  display: flex;
  align-items: center;
  justify-content: center;
}

.summary {
  background: rgba(255, 255, 255, 0.95);
  padding: 30px;
  border-radius: 15px;
  margin-bottom: 50px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
}

.summary h2 {
  color: #2c3e50;
  margin-bottom: 15px;
  position: relative;
}

.summary h2::after {
  content: '';
  position: absolute;
  bottom: -8px;
  left: 0;
  width: 60px;
  height: 3px;
  background: linear-gradient(90deg, #3498db, #9b59b6);
  border-radius: 2px;
}

.summary p {
  line-height: 1.8;
```

```
    margin-bottom: 15px;
    color: #555;
   }

   @media (max-width: 900px) {
    .column {
     flex: 0 0 100%;
     max-width: 100%;
    }

    h1 {
     font-size: 2rem;
    }

    .image-box {
     height: auto;
    }

    .image-content {
     height: 300px;
    }
   }

   @media (max-width: 480px) {
    h1 {
     font-size: 1.5rem;
    }

    .column-header {
     font-size: 1.4rem;
    }
   }
  </style>
</head>
<body>
  <div class="page-container">
   <header>
    <h1>Comparison Between CNN and Random Forest Algorithm</h1>
    <p class="subtitle">Performance analysis for image classification tasks</p>
   </header>

   <div class="summary">
    <h2>Algorithm Comparison Summary</h2>
```

This page presents a visual comparison between Convolutional Neural Networks (CNN) and Random Forest (RF) algorithms for image classification. Both algorithms were evaluated on their ability to classify images by category and quality, with performance metrics including accuracy, precision, recall, and F1-score.
The confusion matrices provide detailed insights into how each algorithm performs across different classes, highlighting where they excel and where they might need improvement.

<h2 class="column-header">Convolutional Neural Network (CNN)</h2>

CNN Accuracy

<img src="./static/results/cnn_report.png" alt="CNN Accuracy">

Overall classification performance metrics for CNN model

CNN Accuracy Category

<img src="./static/results/CNN/CNN_Accuracy_Category.png" alt="CNN Accuracy Category">

Category-specific accuracy measurements for CNN model

CNN Accuracy Quality

<img src="./static/results/CNN/CNN_Accuracy_Quality.png" alt="CNN Accuracy Quality">

Quality-specific accuracy measurements for CNN model

CNN Category Confusion Matrix

<img src="./static/results/CNN/CNN_Category_Confusion.png" alt="CNN Category Confusion Matrix">

```
            <div class="image-caption">Visualizes the CNN model's performance across different categories</div>
        </div>

        <div class="image-box">
          <div class="image-title">CNN Quality Confusion Matrix</div>
          <div class="image-content">
              <img src="./static/results/CNN/CNN_Quality_Confusion.png" alt="CNN Quality Confusion Matrix">
          </div>
           <div class="image-caption">Visualizes the CNN model's performance across different quality levels</div>
        </div>
      </div>

      <div class="column">
        <h2 class="column-header">Random Forest (RF)</h2>

        <div class="image-box">
          <div class="image-title">Random Forest Accuracy</div>
          <div class="image-content">
            <img src="./static/results/rf_report.png" alt="Random Forest Accuracy">
          </div>
             <div class="image-caption">Overall classification performance metrics for Random Forest model</div>
        </div>

        <div class="image-box">
          <div class="image-title">Random Forest Accuracy Category</div>
          <div class="image-content">
              <img src="./static/results/RF/Accuracy_Category_RF.png" alt="Random Forest Accuracy Category">
          </div>
          <div class="image-caption">Category-specific accuracy measurements for Random Forest</div>
        </div>

        <div class="image-box">
          <div class="image-title">Random Forest Accuracy Quality</div>
          <div class="image-content">
           <img src="./static/results/RF/Accuracy_RF_Quality.png" alt="Random Forest Accuracy Quality">
          </div>
             <div class="image-caption">Quality-specific accuracy measurements for Random Forest model</div>
```

```
      </div>

      <div class="image-box">
        <div class="image-title">Random Forest Category Confusion Matrix</div>
        <div class="image-content">
            <img src="./static/results/RF/Confusion_Category_RF.png" alt="Random Forest Category Confusion Matrix">
        </div>
        <div class="image-caption">Visualizes the Random Forest model's performance across different categories</div>
      </div>

      <div class="image-box">
        <div class="image-title">Random Forest Quality Confusion Matrix</div>
        <div class="image-content">
        <img src="./static/results/RF/Confusion_Quality_RF.png" alt="Random Forest Quality Confusion Matrix">
        </div>
        <div class="image-caption">Visualizes the Random Forest model's performance across different quality levels</div>
      </div>
    </div>
  </div>
 </div>
 <br>
  <form action="{{ url_for('home') }}" method="GET">
    <button type="submit" class="btn btn-primary">Back to Home</button>
  </form>
  <br>
</body>
</html>
```

HOME PAGE

```
{% extends "Base.html" %}

{% block content %}
<hr>
<div class="container">
  <form action="{{ url_for('pred') }}" method="POST" enctype="multipart/form-data">
    <h1><strong>Fruit Disease Prediction</strong>.</h1>
    <br><br><br>
    <h4>Click below to upload your <strong>image</strong>.</h4>
```

```
      <div class="form-group">
        <label>
          <input type="file" name="file" id="file" style="display:none" accept="image/*" required>
          <img src="../static/upload5.png" width="50%">
        </label>
      </div>

      <div class="container">
        <div class="form-group form-check">
          <input type="checkbox" class="form-check-input" id="exampleCheck1" required>
         <label class="form-check-label" for="exampleCheck1">Make sure you uploaded images related to <strong>1. Apples</strong>, <strong>2. Bananas</strong>, <strong>3. Oranges</strong></label>
        </div>
      </div>
      <br>
       <input type="submit" class="btn btn-secondary" name="Submit and predict" value="Submit and Predict">
   </form>

   <!-- Comparison Button -->
   <br>
   <form action="{{ url_for('compare') }}" method="GET">
      <button type="submit" class="btn btn-primary">Comparison</button>
   </form>
   <br>
</div>
{% endblock %}
```

DATASET PRED3 CLASS

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">

<!-- JS, Popper.js, and jQuery -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>
```

```
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" integrity="sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI" crossorigin="anonymous"></script>

<style>
* {box-sizing: border-box}

.container1 {
  width: 100%;
  background-color: #d3d3d330;
}

.skills {
  text-align: right;
  padding-top: 10px;
  padding-bottom: 10px;
  color: white;
}

.apple {width: {{fruit_dict['apple']}}%; background-color: #8B0000;}
.banana {width: {{fruit_dict['banana']}}%; background-color: #B8860B;}
.orange {width: {{fruit_dict['orange']}}%; background-color: #FF8C00;}
.fresh {width: {{rotten[0]}}%; background-color: #228B22;}
.rotten {width: {{rotten[1]}}%; background-color: #696969;}
body {
  text-align: center;
  align:center;
  background-repeat: no-repeat;
  background-size: cover;
                                                                    background-image:
url('https://static.vecteezy.com/system/resources/thumbnails/033/107/945/small_2x/fresh-fruits-food-background-web-banner-with-copy-space-generative-ai-photo.jpg');
}
</style>
</head>
<body>

<br>
```

```
<h4 class="display-4">Prediction <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-bar-chart" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
  <path fill-rule="evenodd" d="M4 11H2v3h2v-3zm5-4H7v7h2V7zm5-5h-2v12h2V2zm-2-1a1 1 0 0 0-1 1v12a1 1 0 0 0 1 1h2a1 1 0 0 0 1-1V2a1 1 0 0 0-1-1h-2zM6 7a1 1 0 0 1 1-1h2a1 1 0 0 1 1 1v7a1 1 0 0 1-1 1H7a1 1 0 0 1-1-1V7zm-5 4a1 1 0 0 1 1-1h2a1 1 0 0 1 1 1v3a1 1 0 0 1-1 1H2a1 1 0 0 1-1-1v-3z"/>
</svg></h4>

<hr>
<div class="container" style="background-color: rgba(236, 235, 232, 0.292);">
<div class="row">
  <div class="col-sm">
    <ul>
      <h3>1. Fruit Classifier <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-arrow-down-square" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
  <path fill-rule="evenodd" d="M14 1H2a1 1 0 0 0-1 1v12a1 1 0 0 0 1 1h12a1 1 0 0 0 1-1V2a1 1 0 0 0-1-1zM2 0a2 2 0 0 0-2 2v12a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V2a2 2 0 0 0-2-2H2z"/>
  <path fill-rule="evenodd" d="M4.646 7.646a.5.5 0 0 1 .708 0L8 10.293l2.646-2.647a.5.5 0 0 1 .708.708l-3 3a.5.5 0 0 1-.708 0l-3-3a.5.5 0 0 1 0-.708z"/>
  <path fill-rule="evenodd" d="M8 4.5a.5.5 0 0 1 .5.5v5a.5.5 0 0 1-1 0V5a.5.5 0 0 1 .5-.5z"/>
</svg></h3>
    </ul>
    <hr>
    <br>
   <h4 align='left'><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-percent" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
  <path fill-rule="evenodd" d="M13.442 2.558a.625.625 0 0 1 0 .884l-10 10a.625.625 0 1 1-.884-.884l10-10a.625.625 0 0 1 .884 0zM4.5 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5zm7 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5z"/>
</svg> Apple</h4>
  <div class="container1">
    <div class="skills apple">{{fruit_dict['apple']}} %</div>
  </div>
  <br>

   <h4 align='left'><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-percent" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
  <path fill-rule="evenodd" d="M13.442 2.558a.625.625 0 0 1 0 .884l-10 10a.625.625 0 1 1-.884-.884l10-10a.625.625 0 0 1 .884 0zM4.5 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5zm7 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5z"/>
</svg> Banana</h4>
  <div class="container1">
    <div class="skills banana">{{fruit_dict['banana']}} %</div>
  </div>
```

```
 <br>

  <h4 align='left'><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-percent" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
 <path fill-rule="evenodd" d="M13.442 2.558a.625.625 0 0 1 0 .884l-10 10a.625.625 0 1 1-.884-.884l10-10a.625.625 0 0 1 .884 0zM4.5 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5zm7 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5z"/>
</svg> Orange</h4>
 <div class="container1">
  <div class="skills orange">{{fruit_dict['orange']}} %</div>
 </div>
 <br>
 <ul>
  <h3>2. Quality Classifier <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-arrow-down-square" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
 <path fill-rule="evenodd" d="M14 1H2a1 1 0 0 0-1 1v12a1 1 0 0 0 1 1h12a1 1 0 0 0 1-1V2a1 1 0 0 0-1-1zM2 0a2 2 0 0 0-2 2v12a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V2a2 2 0 0 0-2-2H2z"/>
  <path fill-rule="evenodd" d="M4.646 7.646a.5.5 0 0 1 .708 0L8 10.293l2.646-2.647a.5.5 0 0 1 .708.708l-3 3a.5.5 0 0 1-.708 0l-3-3a.5.5 0 0 1 0-.708z"/>
 <path fill-rule="evenodd" d="M8 4.5a.5.5 0 0 1 .5.5v5a.5.5 0 0 1-1 0V5a.5.5 0 0 1 .5-.5z"/>
</svg></h3>
 </ul>
 <hr>
 <br>
  <h4 align='left'><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-percent" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
 <path fill-rule="evenodd" d="M13.442 2.558a.625.625 0 0 1 0 .884l-10 10a.625.625 0 1 1-.884-.884l10-10a.625.625 0 0 1 .884 0zM4.5 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5zm7 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5z"/>
</svg> Fresh</h4>
 <div class="container1">
  <div class="skills fresh">{{rotten[0]}} %</div>
 </div>
 <br>

  <h4 align='left'> <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-percent" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
 <path fill-rule="evenodd" d="M13.442 2.558a.625.625 0 0 1 0 .884l-10 10a.625.625 0 1 1-.884-.884l10-10a.625.625 0 0 1 .884 0zM4.5 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5zm7 6a1.5 1.5 0 1 0 0-3 1.5 1.5 0 0 0 0 3zm0 1a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5z"/>
</svg>
Rotten</h4>
 <div class="container1">
```

```
   <div class="skills rotten">{{rotten[1]}} %</div>
  </div>
<br>

  </div>
  <br>

 <div class="col-sm">
 <h3>Uploaded Image <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-file-earmark-arrow-up-fill" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
   <path fill-rule="evenodd" d="M2 3a2 2 0 0 1 2-2h5.293a1 1 0 0 1 .707.293L13.707 5a1 1 0 0 1 .293.707V13a2 2 0 0 1-2 2H4a2 2 0 0 1-2-2V3zm7 2V2l4 4h-3a1 1 0 0 1-1-1zM6.354 9.854a.5.5 0 0 1-.708-.708l2-2a.5.5 0 0 1 .708 0l2 2a.5.5 0 0 1-.708.708L8.5 8.707V12.5a.5.5 0 0 1-1 0V8.707L6.354 9.854z"/>
</svg></h3>

 <div class="container1">
  <img src=" {{plot_url}}" alt='Your uploaded image' height="70%" >
  </div>
<br>

   </div></div>
   <br><br>
</div>
<hr>

<br><br>

</body>
</html>
```

DATABASE CON

```
import os
import sys
import numpy as np
from flask import Flask, render_template, request
from werkzeug.utils import secure_filename
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
from PIL import Image, ImageFile
import utils
from io import BytesIO
```

```
import matplotlib.pyplot as plt
import base64

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/compare')
def compare():
    return render_template('compare.html')  # or whatever logic/page you want

@app.route('/Prediction', methods=['GET','POST'])
def pred():
    if request.method=='POST':
        file = request.files['file']
        org_img, img= utils.preprocess(file)

        print(img.shape)
        fruit_dict=utils.classify_fruit(img)
        rotten=utils.check_rotten(img)

        img_x=BytesIO()
        plt.imshow(org_img/255.0)
        plt.savefig(img_x,format='png')
        plt.close()
        img_x.seek(0)
        plot_url=base64.b64encode(img_x.getvalue()).decode('utf8')

    return render_template('Pred3.html', fruit_dict=fruit_dict, rotten=rotten, plot_url=plot_url)

if __name__=='__main__':
    app.run(debug=True)
```

REQUIREMENTS TEXT

```
absl-py==2.1.0
astunparse==1.6.3
cachetools==5.3.3
certifi==2024.2.2
charset-normalizer==3.3.2
```

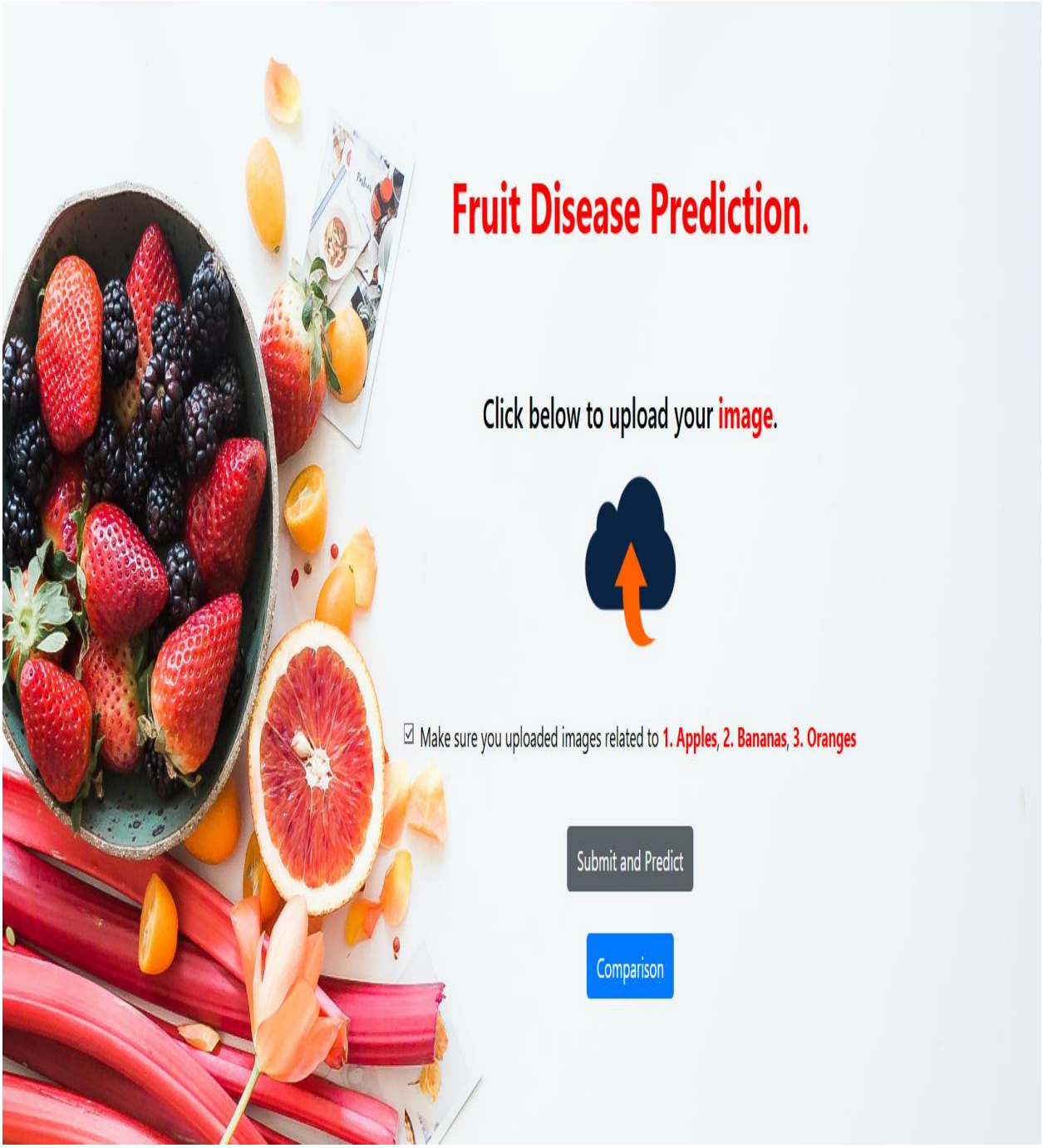
click==8.1.7
colorama==0.4.6
cycler==0.11.0
Flask==2.2.5
flatbuffers==24.3.25
fonttools==4.38.0
gast==0.4.0
google-auth==2.29.0
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.62.2
h5py==3.8.0
idna==3.7
importlib-metadata==6.7.0
itsdangerous==2.1.2
Jinja2==3.1.3
keras==2.11.0
kiwisolver==1.4.5
libclang==18.1.1
Markdown==3.4.4
MarkupSafe==2.1.5
matplotlib==3.5.3
numpy==1.21.6
oauthlib==3.2.2
opt-einsum==3.3.0
packaging==24.0
Pillow==9.5.0
protobuf==3.19.6
pyasn1==0.5.1
pyasn1-modules==0.3.0
pyparsing==3.1.2
python-dateutil==2.9.0.post0
requests==2.31.0
requests-oauthlib==2.0.0
rsa==4.9
six==1.16.0
tensorboard==2.11.2
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.11.0
tensorflow-estimator==2.11.0
tensorflow-intel==2.11.0
tensorflow-io-gcs-filesystem==0.31.0

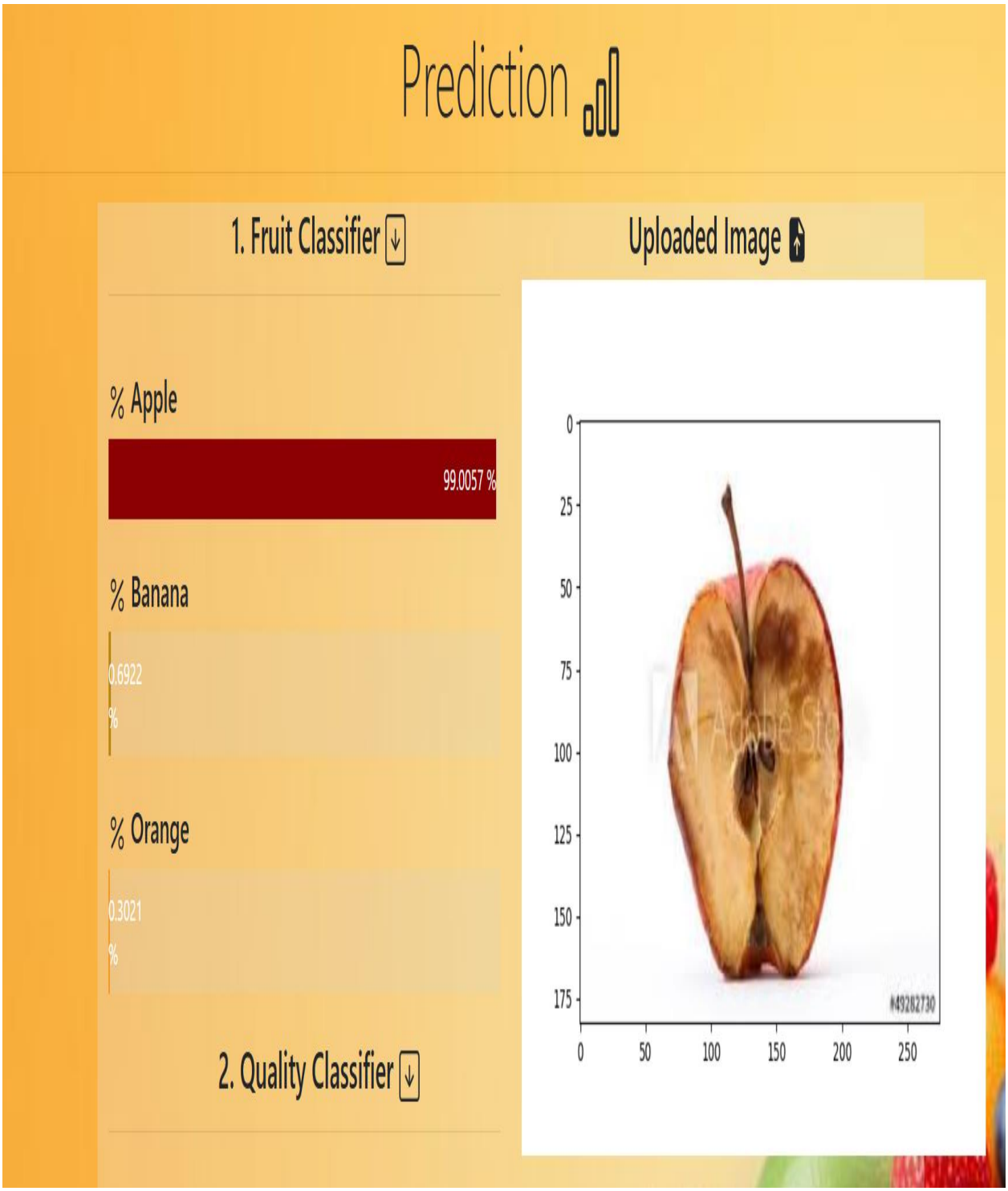
termcolor==2.3.0
typing-extensions==4.7.1
urllib3==2.0.7
Werkzeug==2.2.3
wrapt==1.16.0
zipp==3.15.0

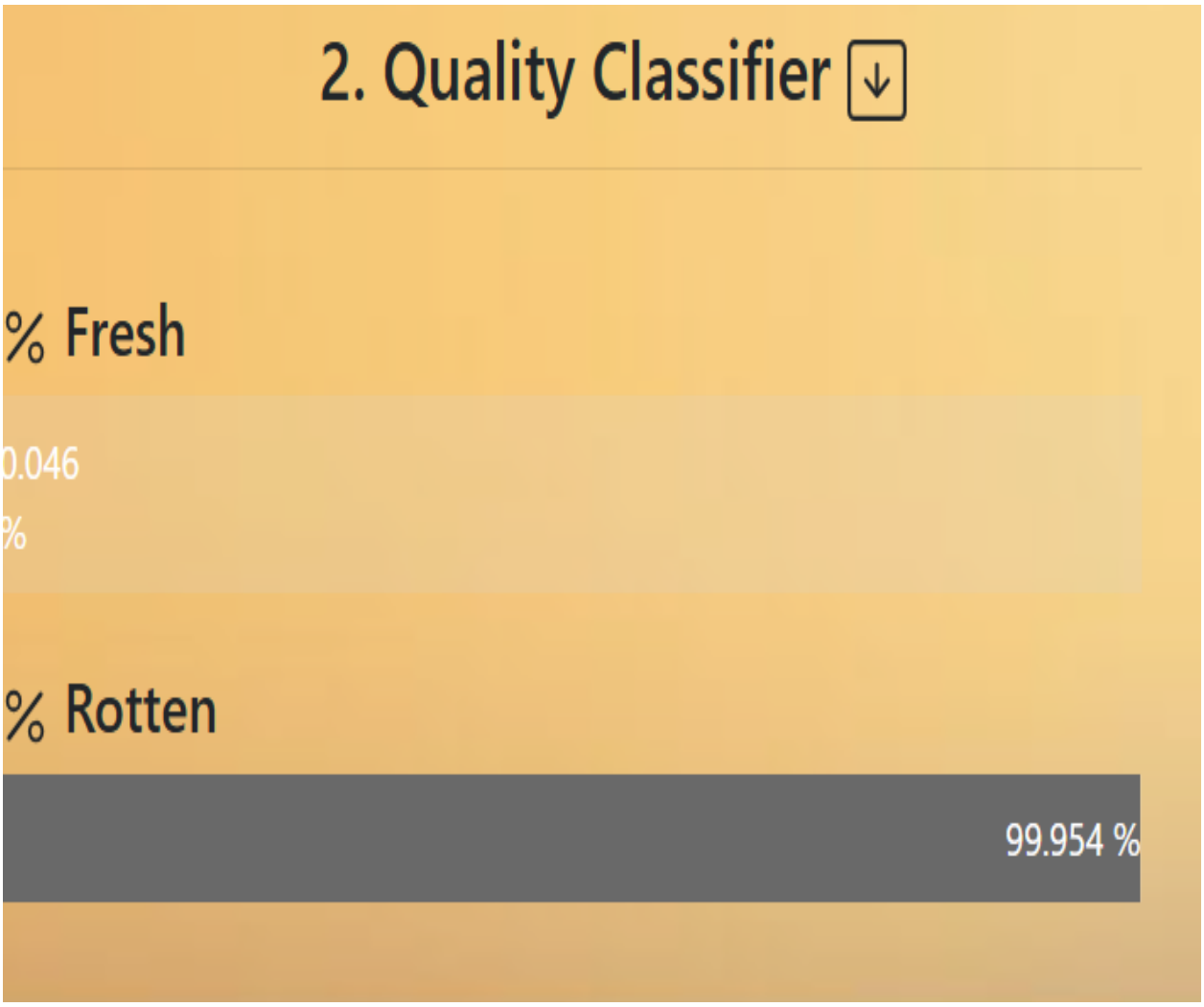
PROFILE UPDATE

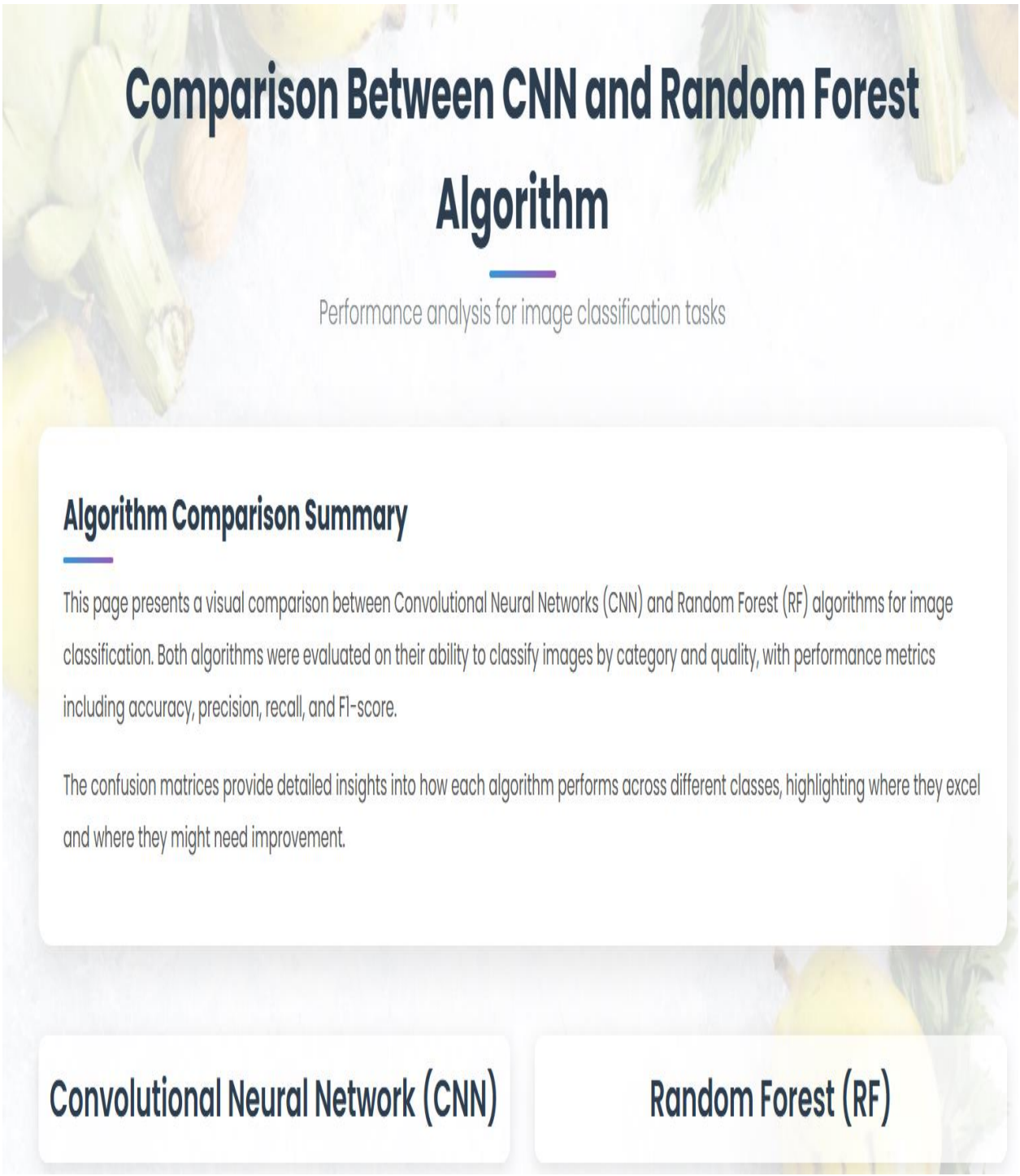
```
rom tensorflow.keras.models import load_model
import numpy as np
from tensorflow.keras.preprocessing import image
from PIL import Image, ImageFile
from io import BytesIO

quality_model=load_model('model/local_rotten_lr2_final.h5')
clf_model=load_model('model/local_fruit_final.h5')

def preprocess(file):
    ImageFile.LOAD_TRUNCATED_IMAGES =False
    org_img=Image.open(BytesIO(file.read()))
    org_img.load()
    img=org_img.resize((100,100), Image.ANTIALIAS)

    img=image.img_to_array(img)
    org_img=image.img_to_array(org_img)
    return org_img, np.expand_dims(img,axis=0)

def check_rotten(img):
                                return                [round(100*quality_model.predict(img)[0][0],3),round(100*(1-
quality_model.predict(img)[0][0]),3)]

def classify_fruit(img):
    fru_dict={}
    fru_dict['apple']=round(clf_model.predict(img)[0][0]*100,4)
    fru_dict['banana']=round(clf_model.predict(img)[0][1]*100,4)
    fru_dict['orange']=round(clf_model.predict(img)[0][2]*100,4)

    for value in fru_dict:
     if fru_dict[value]<=0.001:
        fru_dict[value]=0.00

    return fru_dict
```

8.2 SCREENSHOTS




Prediction
1. Fruit Classifier
Uploaded Image
% Apple
99.0057 %
% Banana
0.6922
%
% Orange
0.3021
%
2. Quality Classifier
0
25
50
75
100
125
150
175
0
50
100
150
200
250


2. Quality Classifier
% Fresh
0.046
%
% Rotten
99.954 %


Comparison Between CNN and Random Forest Algorithm
Performance analysis for image classification tasks
Algorithm Comparison Summary
This page presents a visual comparison between Convolutional Neural Networks (CNN) and Random Forest (RF) algorithms for image classification. Both algorithms were evaluated on their ability to classify images by category and quality, with performance metrics including accuracy, precision, recall, and F1-score.
The confusion matrices provide detailed insights into how each algorithm performs across different classes, highlighting where they excel and where they might need improvement.
Convolutional Neural Network (CNN)
Random Forest (RF)

CNN Accuracy

[REPORT]

	precision	recall	f1-score	support
apple	0.97	0.95	0.96	37
banana	0.83	0.95	0.88	20
orange	0.97	0.90	0.93	31
accuracy			0.93	88
macro avg	0.92	0.93	0.93	88
weighted avg	0.94	0.93	0.93	88

Accuracy: 0.9318181818181818

Overall classification performance metrics for CNN model

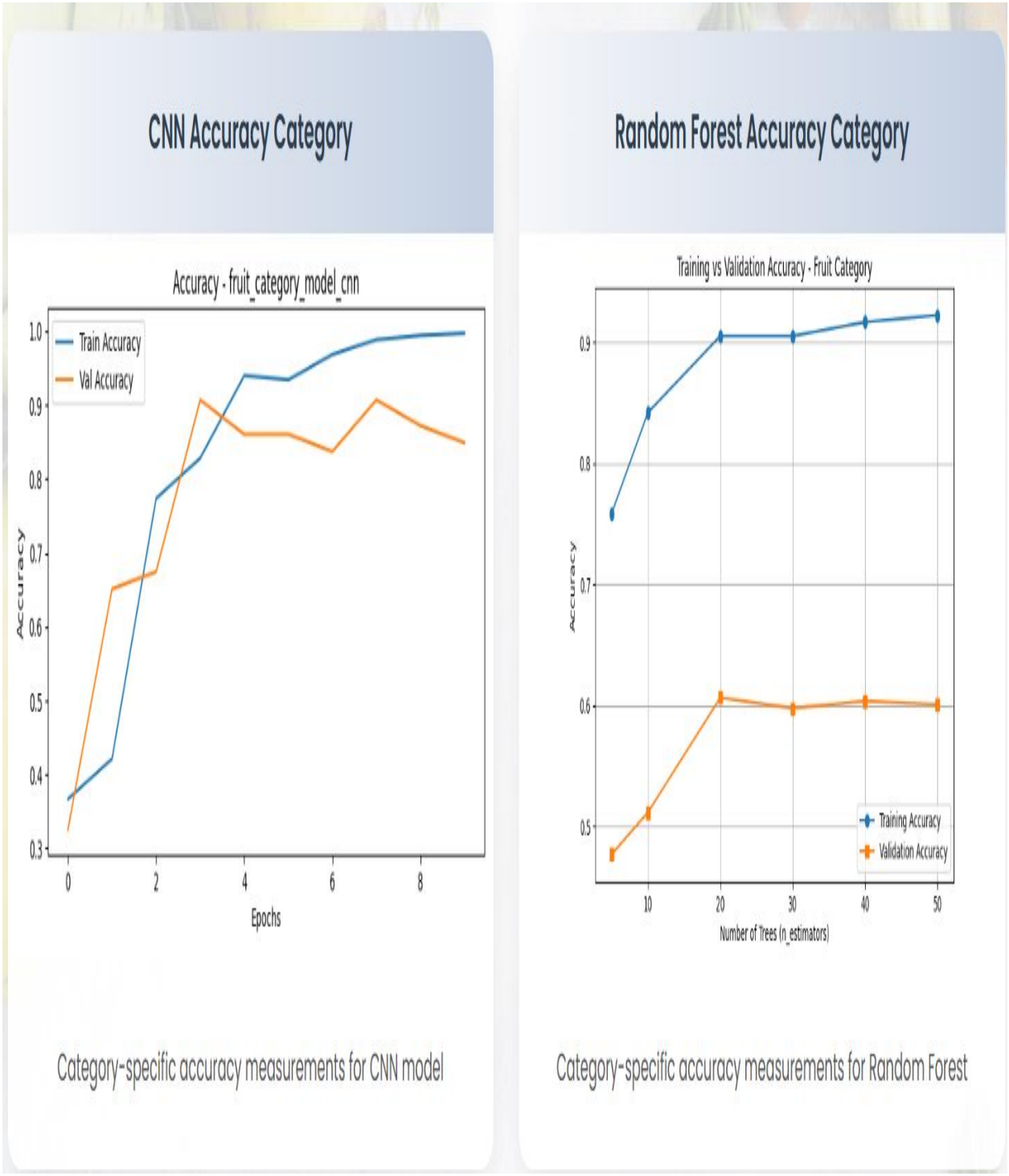
Random Forest Accuracy

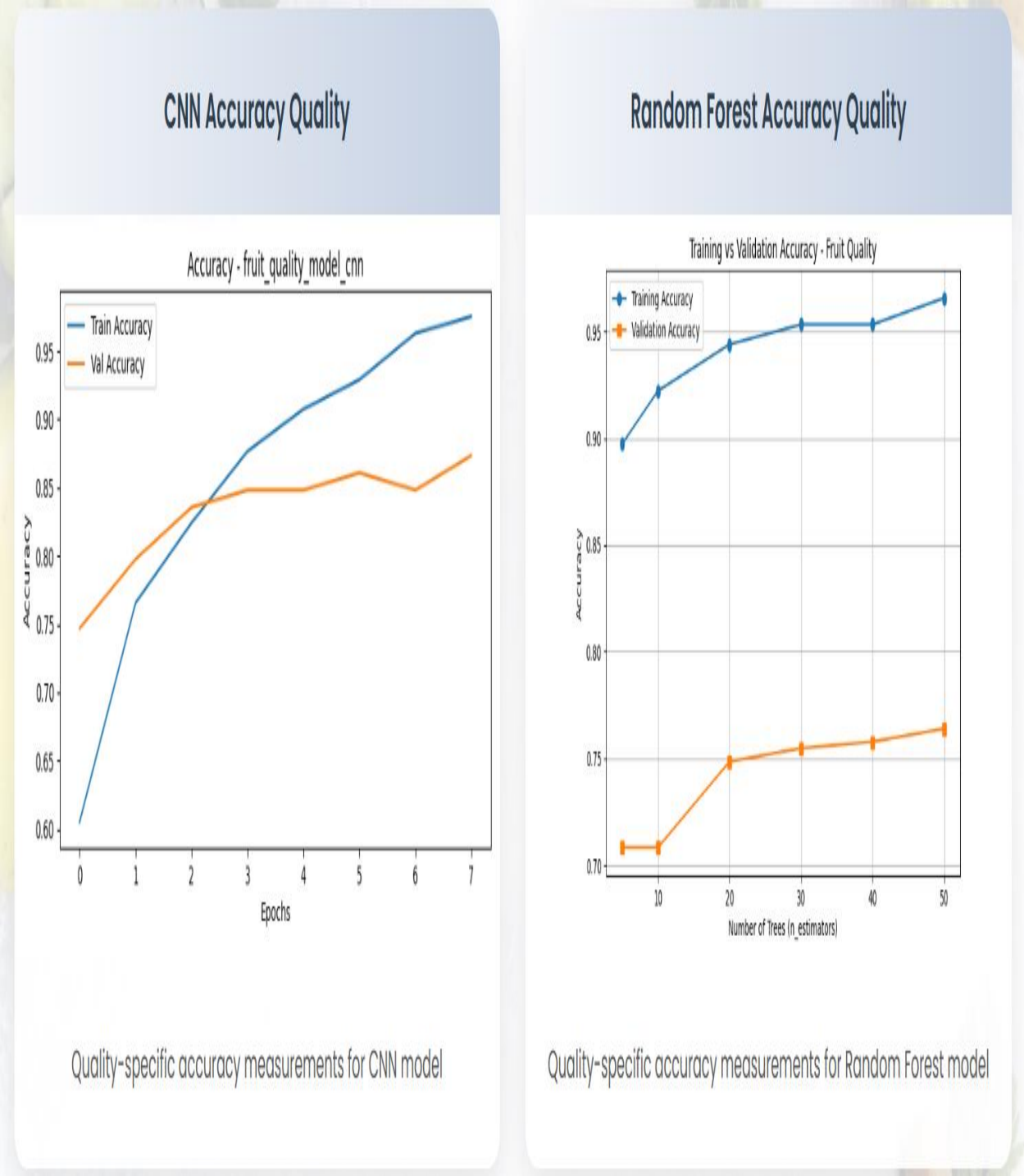
REPORT]

	precision	recall	f1-score	support
apple	0.73	0.51	0.60	37
banana	0.28	0.55	0.37	20
orange	0.52	0.39	0.44	31
accuracy			0.48	88
macro avg	0.51	0.48	0.47	88
eighted avg	0.56	0.48	0.49	88

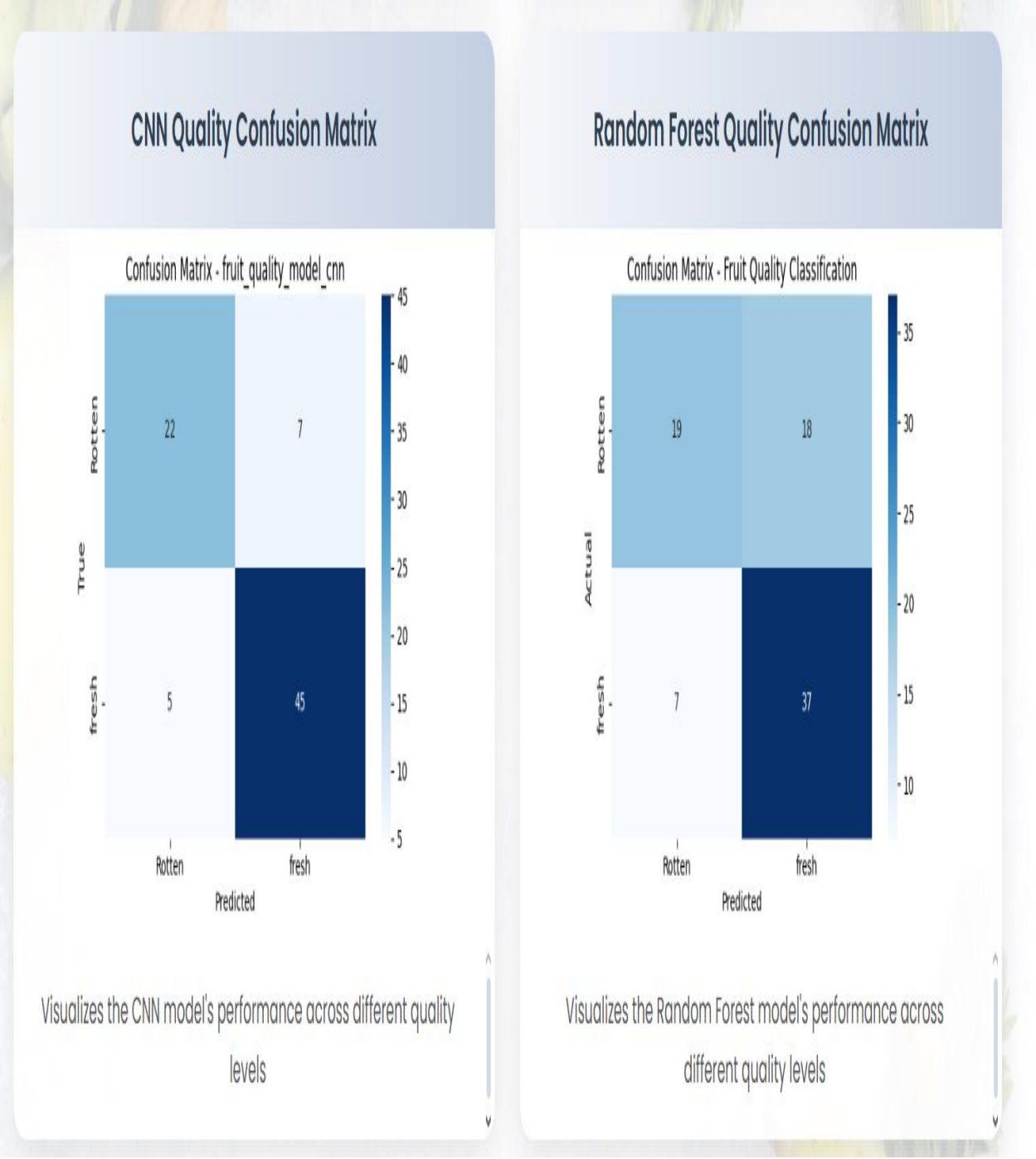
ccuracy: 0.4772727272727273

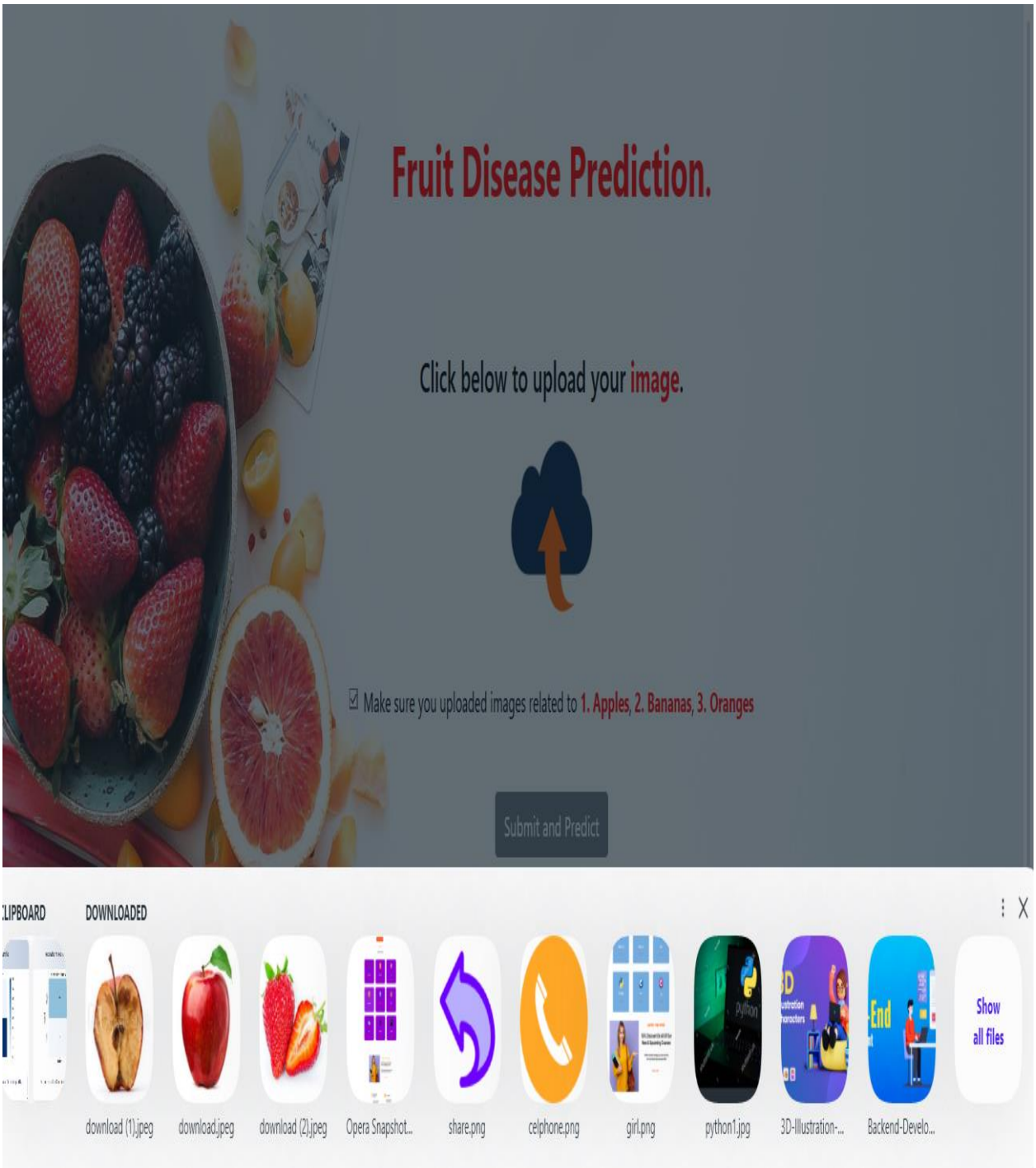
Overall classification performance metrics for Random Forest model

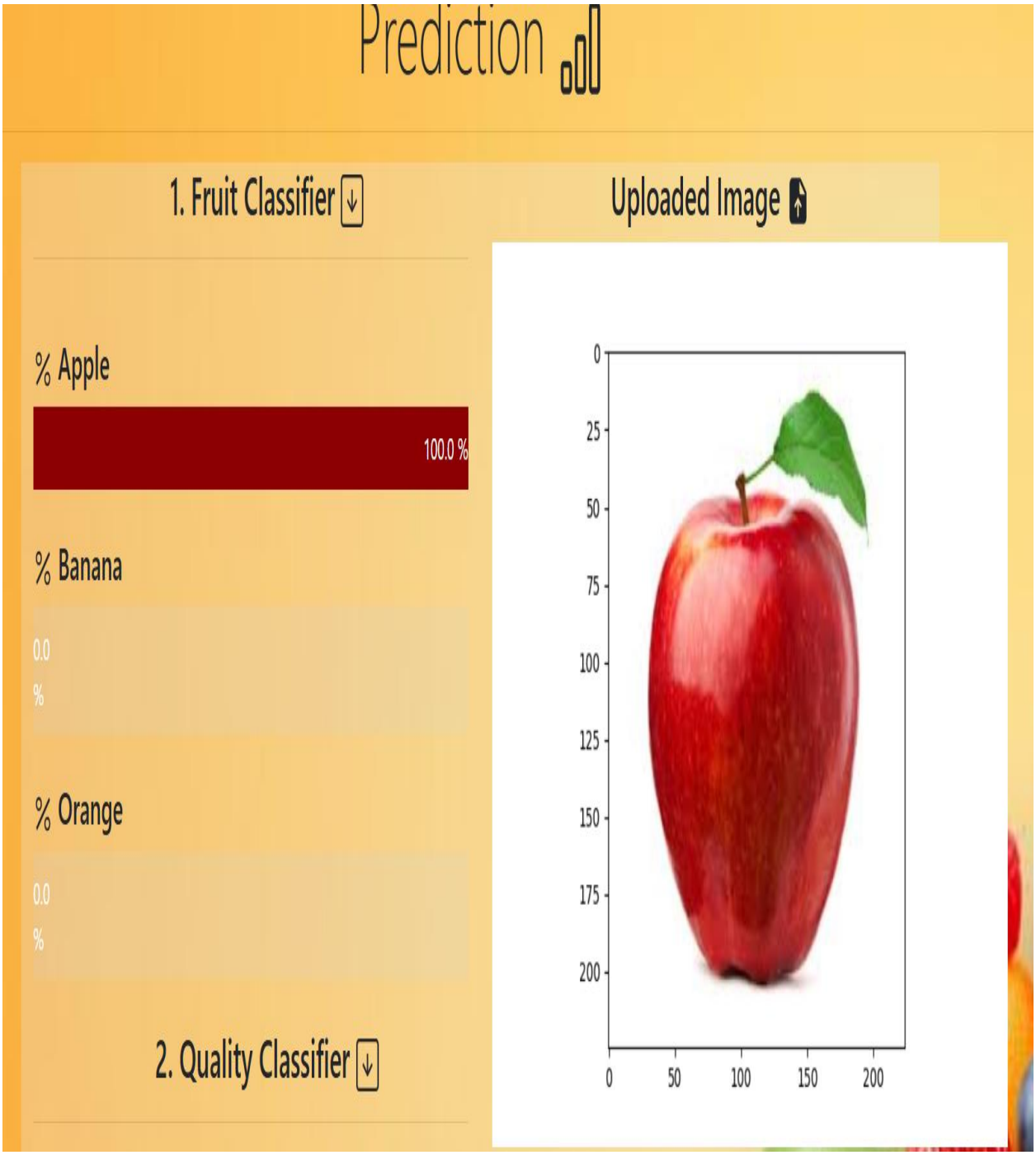

CNN Accuracy Category
Accuracy - fruit_category_model_cnn
Train Accuracy
Val Accuracy
Accuracy
Epochs
Category-specific accuracy measurements for CNN model
Random Forest Accuracy Category
Training vs Validation Accuracy - Fruit Category
Accuracy
Training Accuracy
Validation Accuracy
Number of Trees (n_estimators)
Category-specific accuracy measurements for Random Forest

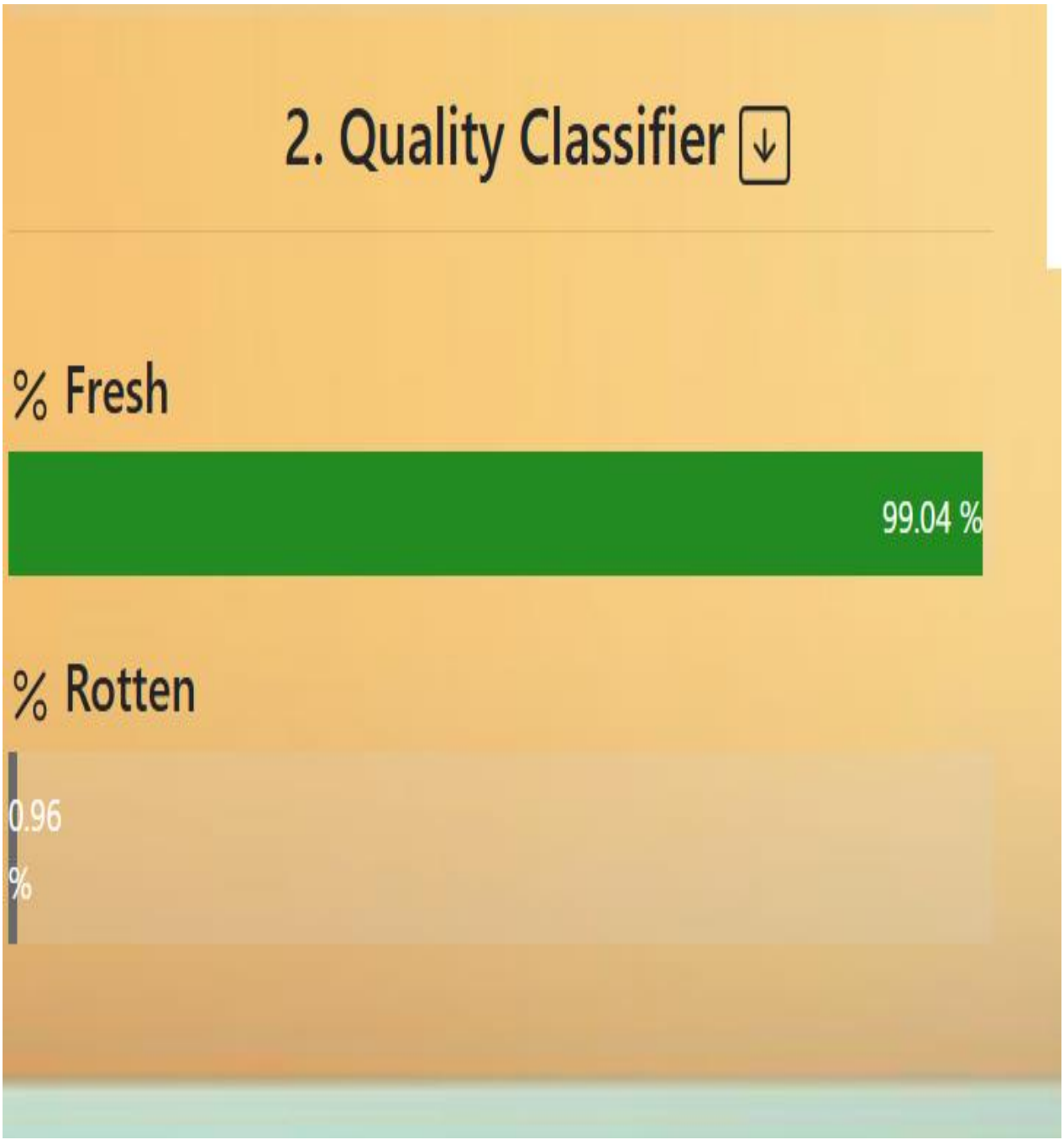

CNN Accuracy Quality
Accuracy - fruit_quality_model_cnn
Train Accuracy
Val Accuracy
Accuracy
Epochs
Quality-specific accuracy measurements for CNN model
Random Forest Accuracy Quality
Training vs Validation Accuracy - Fruit Quality
Training Accuracy
Validation Accuracy
Accuracy
Number of Trees (n_estimators)
Quality-specific accuracy measurements for Random Forest model


CNN Category Confusion Matrix
Confusion Matrix - fruit_category_model_cnn
True
apple
banana
orange
25
0
3
2
19
8
0
0
29
apple
banana
orange
Predicted
25
20
15
10
5
0
Visualizes the CNN model's performance across different categories
Random Forest Category Confusion Matrix
Confusion Matrix - Fruit Category Classification
Actual
apple
banana
orange
16
13
8
4
11
5
2
16
13
apple
banana
orange
Predicted
16
14
12
10
8
6
4
2
Visualizes the Random Forest model's performance across different categories

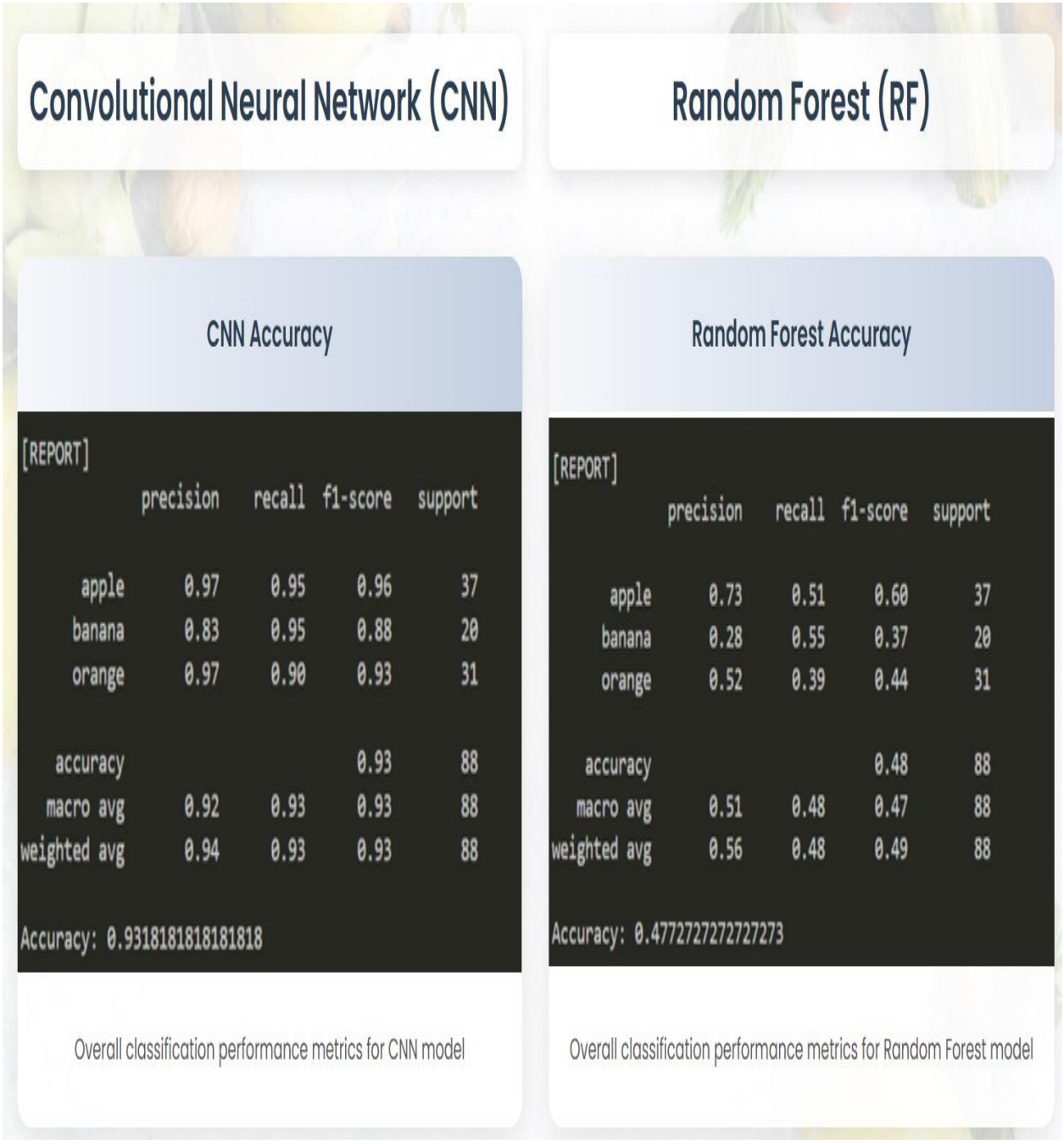

CNN Quality Confusion Matrix
Confusion Matrix - fruit_quality_model_cnn
True
Rotten
fresh
22
7
5
45
Rotten
fresh
Predicted
45
40
35
30
25
20
15
10
5
Visualizes the CNN model's performance across different quality levels
Random Forest Quality Confusion Matrix
Confusion Matrix - Fruit Quality Classification
Actual
Rotten
fresh
19
18
7
37
Rotten
fresh
Predicted
35
30
25
20
15
10
Visualizes the Random Forest model's performance across different quality levels


Fruit Disease Prediction.
Click below to upload your image.
Make sure you uploaded images related to 1. Apples, 2. Bananas, 3. Oranges
Submit and Predict
DOWNLOADED
download (1).jpeg
download.jpeg
download (2).jpeg
Opera Snapshot...
share.png
celphone.png
girl.png
python1.jpg
3D-Illustration-...
Backend-Develo...
Show all files

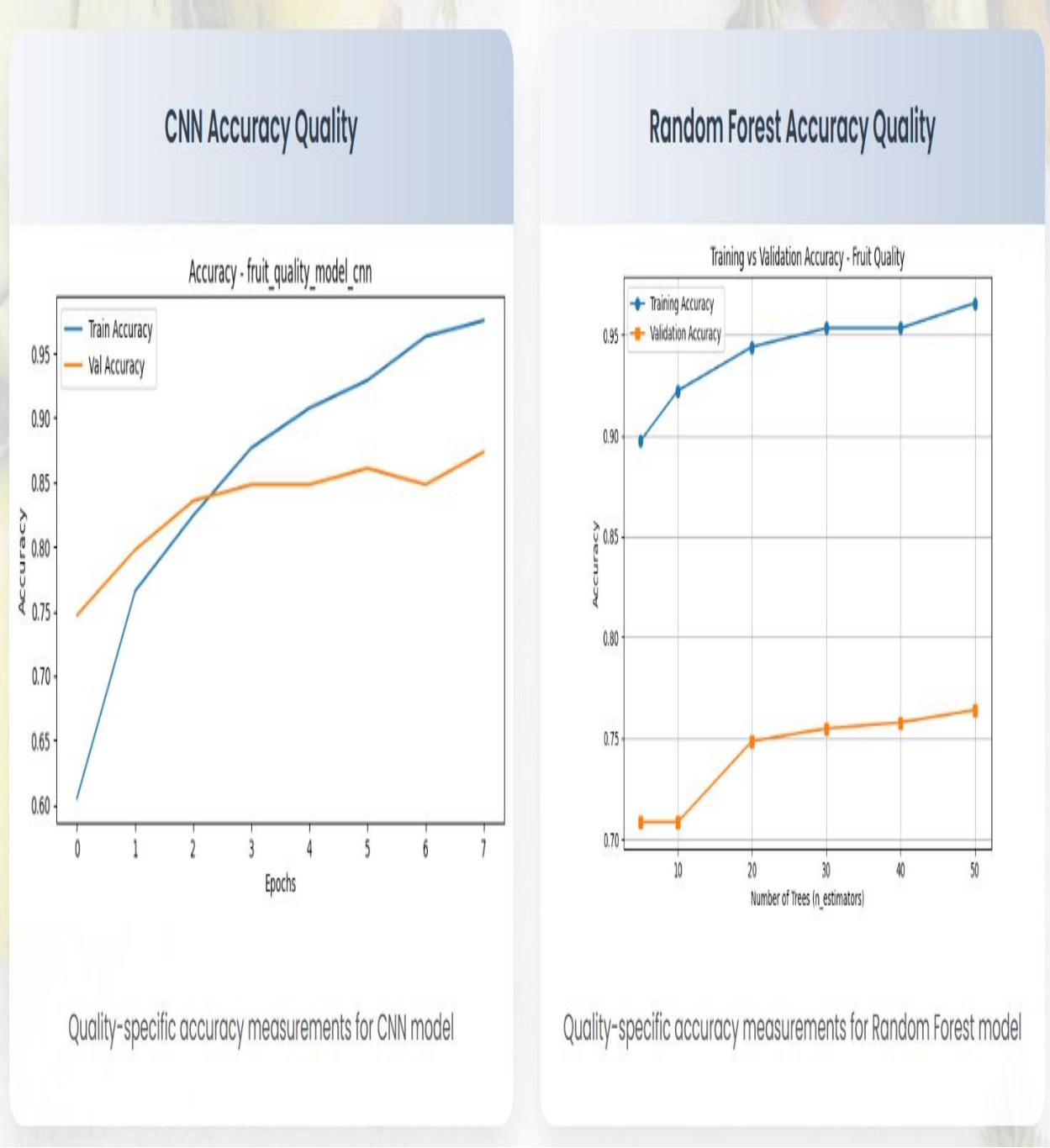

Prediction
1. Fruit Classifier
Uploaded Image
% Apple
100.0 %
% Banana
0.0
%
% Orange
0.0
%
2. Quality Classifier
0
25
50
75
100
125
150
175
200
0
50
100
150
200


2. Quality Classifier
% Fresh
99.04 %
% Rotten
0.96
%

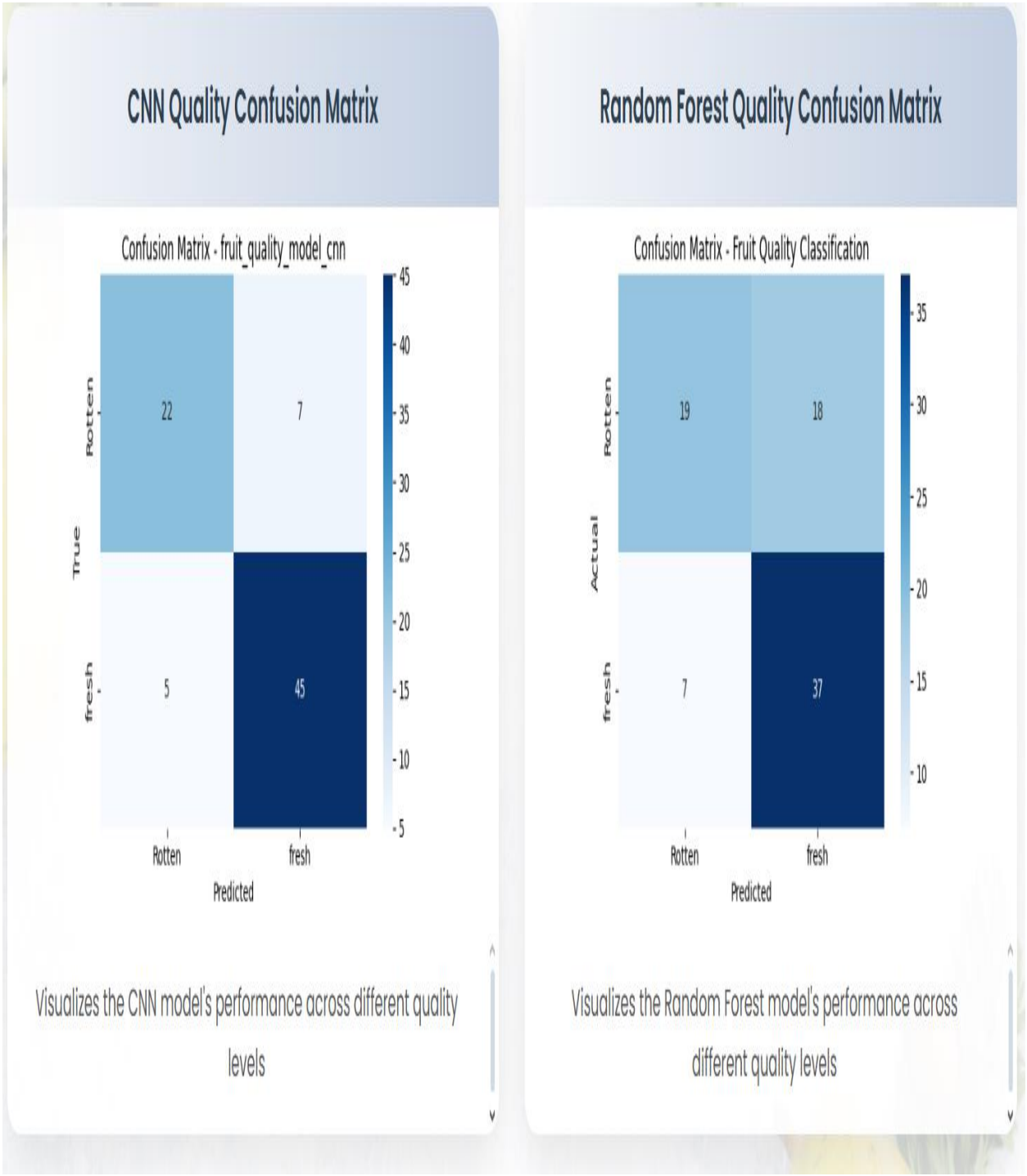

Comparison Between CNN and Random Forest Algorithm
Performance analysis for image classification tasks
Algorithm Comparison Summary
This page presents a visual comparison between Convolutional Neural Networks (CNN) and Random Forest (RF) algorithms for image classification. Both algorithms were evaluated on their ability to classify images by category and quality, with performance metrics including accuracy, precision, recall, and F1-score.
The confusion matrices provide detailed insights into how each algorithm performs across different classes, highlighting where they excel and where they might need improvement.


Convolutional Neural Network (CNN)
Random Forest (RF)
CNN Accuracy
[REPORT]
precision recall f1-score support
apple 0.97 0.95 0.96 37
banana 0.83 0.95 0.88 20
orange 0.97 0.90 0.93 31
accuracy 0.93 88
macro avg 0.92 0.93 0.93 88
weighted avg 0.94 0.93 0.93 88
Accuracy: 0.9318181818181818
Overall classification performance metrics for CNN model
Random Forest Accuracy
[REPORT]
precision recall f1-score support
apple 0.73 0.51 0.60 37
banana 0.28 0.55 0.37 20
orange 0.52 0.39 0.44 31
accuracy 0.48 88
macro avg 0.51 0.48 0.47 88
weighted avg 0.56 0.48 0.49 88
Accuracy: 0.4772727272727273
Overall classification performance metrics for Random Forest model


CNN Accuracy Category
Accuracy - fruit_category_model_cnn
Train Accuracy
Val Accuracy
Accuracy
Epochs
Category-specific accuracy measurements for CNN model
Random Forest Accuracy Category
Training vs Validation Accuracy - Fruit Category
Accuracy
Number of Trees (n_estimators)
Training Accuracy
Validation Accuracy
Category-specific accuracy measurements for Random Forest


CNN Accuracy Quality
Accuracy - fruit_quality_model_cnn
Train Accuracy
Val Accuracy
Accuracy
Epochs
Quality-specific accuracy measurements for CNN model
Random Forest Accuracy Quality
Training vs Validation Accuracy - Fruit Quality
Training Accuracy
Validation Accuracy
Accuracy
Number of Trees (n_estimators)
Quality-specific accuracy measurements for Random Forest model


CNN Category Confusion Matrix
Confusion Matrix - fruit_category_model_cnn
True
apple
banana
orange
25
0
3
2
19
8
0
0
29
apple
banana
orange
Predicted
25
20
15
10
5
0
Visualizes the CNN model's performance across different categories
Random Forest Category Confusion Matrix
Confusion Matrix - Fruit Category Classification
Actual
apple
banana
orange
16
13
8
4
11
5
2
16
13
apple
banana
orange
Predicted
16
14
12
10
8
6
4
2
Visualizes the Random Forest model's performance across different categories


CNN Quality Confusion Matrix
Confusion Matrix - fruit_quality_model_cnn
True
Rotten
fresh
22
7
5
45
Rotten
fresh
Predicted
45
40
35
30
25
20
15
10
5
Visualizes the CNN model's performance across different quality levels
Random Forest Quality Confusion Matrix
Confusion Matrix - Fruit Quality Classification
Actual
Rotten
fresh
19
18
7
37
Rotten
fresh
Predicted
35
30
25
20
15
10
Visualizes the Random Forest model's performance across different quality levels

BIBILOGRAPHY

1.Alston, J.M.; Pardey, P.G. Agriculture in the Global Economy. J. Econ. Perspect. 2014, 28, 121–146.
2.Contribution of Agriculture Sector Towards GDP Agriculture Has Been the Bright Spot in the Economy despite COVID-19. Available online: https://www.pib.gov.in/indexd.aspx (accessed on 29 September 2022).
3.Li, L.; Zhang, S.; Wang, B. Plant Disease Detection and Classification by Deep Learning—A Review. IEEE Access 2021, 9, 56683–56698.
4.Dawod, R.G.; Dobre, C. Upper and Lower Fruit Side Detection with Machine Learning Methods. Sensors 2022, 22, 2696.
5.Khan, M.A.; Akram, T.; Sharif, M.; Javed, K.; Raza, M.; Saba, T. An automated system for cucumber Fruit diseased spot detection and classification using improved saliency method and deep features selection. Multimed. Tools Appl. 2020, 79, 18627–18656.
6.Scientist, D.; Bengaluru, T.M.; Nadu, T. Rice Plant Disease Identification Using Artificial Intelligence. Int. J. Electr. Eng. Technol. 2020, 11, 392–402.
7.Dubey, S.R.; Jalal, A.S. Adapted Approach for Fruit Disease Identification using Images. In Image Processing: Concepts, Methodologies, Tools, and Applications; IGI Global: Hershey, PA, USA, 2013; pp. 1395–1409.
8.Yun, S.; Xianfeng, W.; Shanwen, Z.; Chuanlei, Z. PNN based crop disease recognition with Fruit image features and meteorological data. Int. J. Agric. Biol. Eng. 2015, 8, 60–68.
9.Li, G.; Ma, Z.; Wang, H. Image Recognition of Grape Downy Mildew and Grape. In Proceedings of the International Conference on Computer and Computing Technologies in Agriculture, Beijing, China, 29–31 October 2011; pp. 151–162.
10. Rauf, H.T.; Saleem, B.A.; Lali, M.I.U.; Khan, M.A.; Sharif, M.; Bukhari, S.A.C. A citrus fruits and leaves dataset for detection and classification of citrus diseases through machine learning. Data Brief 2019, 26, 104340.