Detection and Classification of Fresh and Rotten Fruits Using Deep Learning

A PROJECT REPORT

Submitted by

RAJEEV M (AP24122040003)

LOKESH V (AP24122040005)

KOUSHIK G (AP24122040006)

AML 505

ADVANCED PYTHON PROGRAMMING LAB

SRM UNIVERSITY – AP

Neerukonda, Mangalagiri, Guntur

Andhra pradesh – 522240

Nov 2024

Table of Contents

CHAPTER 1
INTRODUCTION

1.1 GENERAL INTRODUCTION

In the realm of agricultural innovation, technology is playing an integral role in enhancing quality assurance and reducing food waste. One promising area of research involves the use of computer vision and deep learning to assess the freshness and quality of fruits. By leveraging advanced image analysis techniques, it is possible to detect spoilage or rot in fruits, enabling efficient sorting and reducing food loss.

The Fruit Rot Detection System seeks to integrate convolutional neural networks (CNNs) and image processing to identify signs of spoilage in fruits. This project focuses on analyzing features such as color, texture, and decay patterns to classify fruits as fresh or rotten. By using pre-trained models such as Xception and VGG16, optimized for detecting intricate features in images, the system aims to provide an accurate and scalable solution for quality control in agriculture and supply chains.

This chapter introduces the Fruit Rot Detection System, highlighting its objectives, scope, and importance in addressing the global issue of food waste.

1.2 SCOPE AND OBJECTIVE

Scope

The Fruit Rot Detection System is designed to:

1. **Automate Quality Assessment**: Use advanced image analysis to identify signs of rot in fruits without manual inspection.
2. **Enhance Sorting Efficiency**: Enable rapid and accurate categorization of fruits into fresh or rotten categories.
3. **Reduce Food Waste**: Minimize post-harvest losses by ensuring only high-quality produce is distributed.

Objectives

1. **Feature Extraction**:

   - Employ deep learning models like Xception and VGG16 to extract features such as discoloration, texture irregularities, and mold presence.

2. **Classification**:
   - Implement a binary classification system to label fruits as "Fresh" or "Rotten."
3. **Optimization for Real-Time Use**:
   - Ensure the system operates efficiently for deployment in industrial sorting lines.
4. **Comprehensive Evaluation**:
   - Test the system on various types of fruits under diverse conditions to assess robustness.

1.3 DEFINITION OF TERMS

1. **Fruit Rot Detection**: A subset of machine learning that utilizes deep neural networks to learn hierarchical representations from raw data. Deep learning models consist of multiple layers of interconnected artificial neurons, allowing them to automatically extract complex features and patterns directly from input data.
2. **Deep Learning Models**:

   Xception: The Xception model is a convolutional neural network (CNN) architecture introduced by François Chollet in 2016. It is based on the concept of depthwise separable convolutions, which aim to capture spatial and channel-wise dependencies in data more efficiently compared to traditional convolutional layers. The Xception model consists of a series of convolutional blocks, each containing depthwise separable convolutions followed by batch normalization and activation functions. This 4 architecture allows for a significant reduction in the number of parameters while maintaining high performance on image classification tasks. Xception has been widely used in various computer vision applications, including image recognition, object detection, and image segmentation. Xception, known for its efficiency and effectiveness, offers a robust framework for emotional facial detection. By leveraging depthwise separable convolutions, Xception captures intricate details in facial images, enabling nuanced emotion recognition. Its deep architecture allows for hierarchical feature extraction, facilitating the identification of subtle emotional cues.

VGG16: The VGG16 model, developed by the Visual Geometry Group at the University of Oxford, is another CNN architecture commonly used for image classification tasks. It consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers in VGG16 use small 3x3 filters with a stride of 1 and zero padding to maintain spatial dimensions. The architecture follows a simple and uniform structure, making it easy to understand and implement. Despite its simplicity, VGG16 has demonstrated strong performance on image classification benchmarks such as the ImageNet dataset. However, due to its large number of parameters, VGG16 is more computationally intensive compared to newer architectures like Xception. VGG16, renowned for its simplicity and performance, serves as a formidable tool in emotional facial detection. With its deep stack of convolutional layers, VGG16 can learn complex representations of facial features, essential for accurate emotion classification. Despite its larger computational footprint compared to Xception, VGG16 excels in capturing spatial dependencies within facial images.

3. **Image Preprocessing**: Transforming raw fruit images into formats suitable for analysis, such as resizing, normalization, and color space conversion (e.g., RGB to grayscale).
4. **Classification Accuracy**: A measure of the system's ability to correctly identify rotten and fresh fruits, often represented as a percentage.

CHAPTER 2

LITERATURESURVEY

The identification of fresh and rotten fruits is critical in the agricultural and food industries to reduce waste, improve quality, and minimize costs. Traditional manual methods are inefficient and often prone to errors. To address these challenges, various techniques and methodologies have been proposed:

1. **CNN-Based Classification**
    - Several studies utilized Convolutional Neural Networks (CNN) to classify fresh and rotten fruits. MobileNetV2 achieved a remarkable accuracy of 99.61% for validation and 99.46% for training on a Kaggle dataset. Max pooling and average pooling methods were also tested, with accuracies ranging from 93% to 94.9%.
2. **Softmax and Feature Extraction**
    - CNN was used to extract features from fruit images, and Softmax was employed for classification. This approach achieved 97.82% accuracy, demonstrating its effectiveness over traditional methods and transfer learning techniques.
3. **Advanced Architectures for Segmentation**
    - Deep learning models like UNet and Enhanced UNet (En-UNet) were implemented to segment decayed regions in apples. En-UNet outperformed the baseline UNet with accuracies of 97.54% (validation) and a mean IoU score of 0.866, indicating superior real-time segmentation performance.
4. **YOLOv4-Based Freshness Detection**
    - A modified YOLOv4 model classified fruits into fresh and rotten categories with high precision. By incorporating the Mish activation function, the model achieved a higher average precision (50.4%) compared to earlier YOLO series.
5. **IoT for Real-Time Detection**
    - IoT-enabled sensors were used to detect spoilage by monitoring gases emitted by fruits. Alerts were sent via a

microcontroller for timely intervention. This method holds significant promise for automation in the food industry.

6. **Transfer Learning and Hybrid Models**
    - Transfer learning techniques using architectures like VGG16 have shown potential in enhancing classification accuracy for fruit freshness. These approaches are particularly useful when datasets are limited, leveraging pre-trained models to improve performance.
7. **Challenges in Classification**
    - Variations in lighting, shadows, and complex backgrounds pose significant challenges to automated systems. Enhanced models like YOLOv4 and En-UNet address these challenges, ensuring higher accuracy and robustness in real-world applications.

Relevance to Current Work

In this project, we utilized the **VGG16 architecture** to classify fresh and rotten fruits. This approach aligns with existing methodologies and provides a robust baseline for achieving high accuracy in fruit classification tasks. VGG16's pre-trained weights and structured convolutional layers make it well-suited for feature extraction and classification, enhancing both training efficiency and prediction accuracy.

CHAPTER 3

PROBLEM STATEMENT AND DISCUSSION

3.1 PROBLEM STATEMENT

The **Fruit Rotten Classification Using VGG16** project addresses the need for an efficient and accurate system capable of identifying whether a fruit is fresh or rotten. In agriculture and the food supply chain, spoilage of fruits leads to significant economic losses, and detecting rotten fruits early can help mitigate these losses.
Despite advances in automation and machine learning, existing systems often fail to effectively classify fruits in real-time with high accuracy, particularly when dealing with large datasets or complex conditions such as variations in lighting, orientation, and fruit damage.

The objective of this project is to develop a machine learning model, specifically leveraging the **VGG16 architecture**, to classify fruits into fresh and rotten categories. The system is designed to process images of eight fruit types— **classify fruits into 16 categories based on their health condition: healthy or rotten. The selected fruits are Apple, Banana, Grape, Guava, Mango, Orange, Pomegranate, and Strawberry.** Determine their condition based on visual features.

By automating this classification process, the project aims to improve efficiency in food processing industries, enhance quality control, and reduce waste. The model's ability to provide accurate results will contribute to optimizing storage and distribution processes, ensuring that only fresh fruits reach consumers.

3.2 EXISTING SYSTEM

Existing systems for fruit classification predominantly rely on traditional image processing techniques or simple machine learning models. These systems have the following characteristics:

1. **Feature Extraction and Classification:**
    - Earlier methods primarily use handcrafted features for image processing, combined with classifiers such as Support Vector Machines (SVMs) or decision trees. These methods often lack the robustness needed to handle complex real-world data.
2. **Accuracy and Generalization Issues:**

    - Most existing systems achieve limited accuracy when dealing with diverse datasets, especially in scenarios involving damaged, occluded, or visually similar fruits.

3. **Lack of Automation:**
    - Many systems require manual intervention or preprocessing, making them less suitable for real-time applications.
4. **Use of Shallow Architectures:**
    - While Convolutional Neural Networks (CNNs) have been adopted in some cases, many systems do not leverage deeper architectures like VGG16 or ResNet, which are better suited for handling large-scale image data.

Limitations of Existing Systems:

- **Low Accuracy in Complex Scenarios**
- **Poor Generalization for New Data**
- **Manual Preprocessing Required**
- **Inability to Handle Diverse Datasets**
- **Limited Real-time Capabilities**

The limitations in these systems highlight the need for a more robust solution that can achieve high accuracy, handle diverse datasets, and operate in real-time.

3.3 PROPOSED SYSTEM

The proposed **Fruit Rotten Classification System** using VGG16 aims to overcome the shortcomings of existing systems by leveraging deep learning techniques. The proposed system includes the following components:

3.3.1 Image Acquisition and Preprocessing:

- The system captures high-resolution images of fruits and preprocesses them to standard dimensions (224x224 pixels) for input into the VGG16 model. Image augmentation techniques are applied to enhance dataset diversity and improve model robustness.

3.3.2 Feature Extraction with VGG16:

- The **VGG16 architecture**, pre-trained on the ImageNet dataset, is used for feature extraction. Its convolutional layers are fine-tuned to extract detailed spatial and texture features critical for distinguishing between fresh and rotten fruits.

3.3.3 Classification Module:

- The extracted features are passed through fully connected layers and a **Softmax classifier** to predict the category of the fruit (fresh or rotten). The model outputs probabilities for each class, ensuring interpretability and accuracy.

3.3.4 Real-time Inference:

- The system is designed for real-time operation, capable of processing live images or batch datasets. It ensures quick and accurate classification suitable for industrial applications.

3.3.5 Evaluation Metrics:

- The model's performance is evaluated using standard metrics like accuracy, precision, recall, and F1-score. A confusion matrix is generated to analyze classification errors and refine the model.

3.3.6 Integration and Scalability:

- The system can be integrated into existing quality control pipelines in the food industry and scaled to include additional fruit types or spoilage conditions with minimal retraining.

3.4 TECHNOLOGIES USED

The development of the **Fruit Rotten Classification Using VGG16** project involves several advanced technologies, ensuring accurate classification and seamless integration. The key technologies are:

3.4.1 Deep Learning

Deep learning frameworks, particularly the **VGG16 architecture**, are used to train a robust model for fruit classification. This pre-trained CNN model is fine-tuned for detecting intricate differences between healthy and rotten fruits.

3.4.2 Python Programming Language

Python is employed as the primary programming language, offering extensive libraries such as TensorFlow, Keras, and OpenCV, which are crucial for image processing, model training, and prediction.

3.4.3 TensorFlow and Keras

TensorFlow and Keras are utilized for building, training, and fine-tuning the VGG16 model. These frameworks provide high-level APIs, enabling efficient neural network implementations.

3.4.4 OpenCV

OpenCV is employed for image preprocessing tasks, including resizing, normalization, and augmentation of fruit images to prepare them for the classification model.

3.4.5 NumPy and Pandas

These libraries are used for data manipulation, statistical analysis, and preprocessing, ensuring the dataset is well-structured for training and validation.

3.4.6 Hardware Accelerators

The training process leverages **GPUs** to speed up the computation-intensive tasks of training deep learning models.

3.5 FRUIT CLASSIFICATION ALGORITHM

Input: Images of fruits categorized as healthy or rotten.
Output: Predicted class label (e.g., Apple__Healthy, Banana__Rotten).

Steps:

1. **Dataset Preparation:** Collect and label fruit images into 16 categories (e.g., Apple__Healthy, Apple__Rotten).
2. **Image Preprocessing:**
    - Resize images to 224x224 pixels (as required by VGG16).
    - Normalize pixel values to a range of [0, 1].
    - Apply data augmentation (e.g., rotations, flips, zooms) to enhance model generalization.

3. **Model Initialization:**
    - Load the pre-trained VGG16 model with ImageNet weights.
    - Replace the top layers with a dense output layer of 16 neurons (softmax activation).
4. **Training the Model:**
    - Use the training dataset to fine-tune the model.
    - Employ cross-entropy loss and Adam optimizer.
5. **Evaluation:**
    - Validate the model on a separate validation set to assess accuracy.
6. **Prediction:**
    - For new fruit images, preprocess the image and pass it through the trained model to predict its category.

3.6 MODULES

The project comprises the following core modules.

3.6.1 Data Collection and Preprocessing Module

- Collect a dataset of healthy and rotten fruit images.
- Preprocess the images for model training, including resizing, normalization, and augmentation.

3.6.2 Model Training Module

- Train the VGG16 model on the preprocessed dataset.
- Fine-tune the model to optimize its performance on the fruit classification task.

3.6.3 Prediction Module

- Accept input images and classify them into one of the 16 categories.
- Display the predicted label and confidence score.

3.6.4 Evaluation and Feedback Module

- Evaluate model performance using metrics such as accuracy, precision, recall, and F1-score.
- Gather user feedback to refine the dataset and improve classification accuracy.

3.6.5 Deployment Module

- Deploy the trained model for real-world applications using APIs or standalone interfaces.
- Enable batch and real-time processing for scalability.

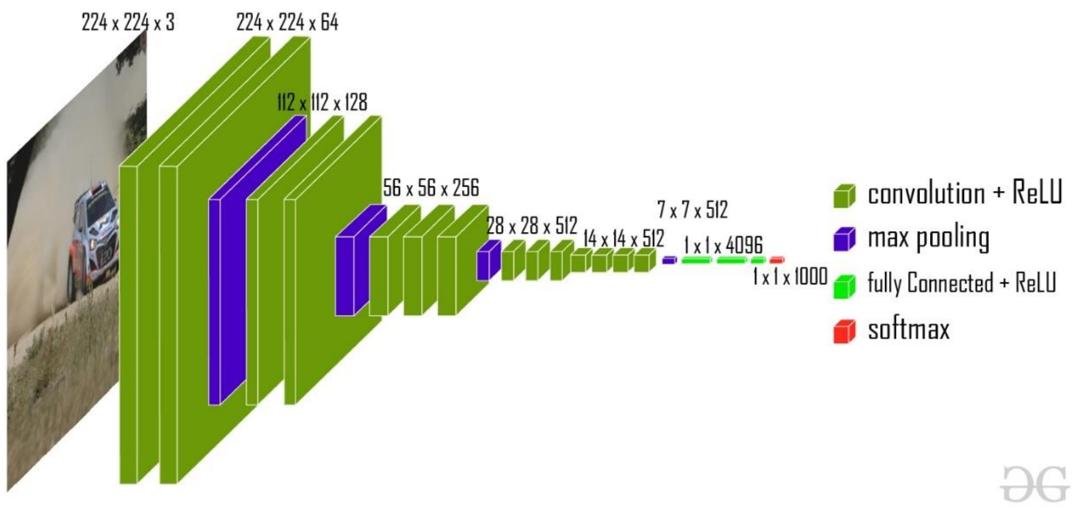
By integrating these modules, the system ensures a seamless pipeline from data preprocessing to deployment, delivering accurate and reliable fruit classification.

CHAPTER 4
SYSTEM ARCHITECTURE AND METHODOLOGY

4.1 Data Analysis: Three different fruit types—Apple, Banana, and Orange— make up the dataset. The CNN algorithm knowledge is mentioned in the Implementation Section with the suitable figure of flow (Figure 5&6). The approach is consistently used to train a specific model to determine if a fruit is fresh or rotten. There are 10901 images in the training set and 2698 images in the test set.

4.2 CNN Model Design:

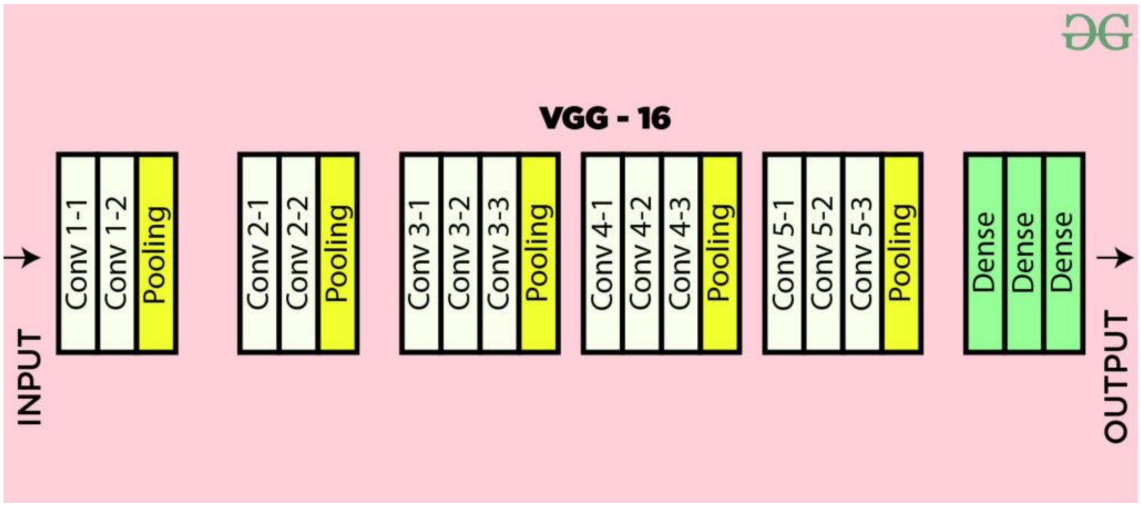
The VGG-16 model is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) It is characterized by its depth, consisting of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks, including image classification and object recognition. The model's architecture features a stack of convolutional layers followed by max-pooling layers, with progressively increasing depth.



VGG Architecture:

The VGG-16 architecture is a deep convolutional neural network (CNN) designed for image classification tasks.

The VGG-16 configuration typically consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. These layers are organized into blocks, with each block containing multiple convolutional layers followed by a max-pooling layer for downsampling.



3. System Design Plan:

The system design planned in this study can be seen in Figure 2.

In preparing the data, the public dataset Fresh and rotten fruit for classification has been collected and the dataset has been divided into two parts, namely training data and testing data. The next process is pre-processing data by cropping, resizing the data as needed. The next process is training by designing a model that is planned to be used and a list of specified parameters such as the level of learning and the number of training epochs. In this process, accuracy is calculated using the loss function. The limitation for training data is limited to fresh apples, fresh oranges, fresh bananas, rotten apples, rotten oranges and rotten bananas so that the planned model can only predict these 8 classes. For the training process of CNN design, this study uses the open source Tensor flow framework and Python 3.12.

CHAPTER 5
SOFTWARE

5.1 SOFTWARES USED

5.1.1 JUPYTER NOTEBOOK

Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It supports a variety of programming languages, including Python,R, Julia, and many others.

The name Jupyter is a combination of the three core programming languages it originally supported: Julia, Python, and R. It was originally developed as part ofthe IPython project in 2011, and has since grown to become a widely used tool for data analysis, scientific computing, and machine learning.

Jupyter Notebook is popular among data scientists, researchers, and educators because it enables them to write and execute code in an interactive, exploratory environment. The notebooks can be easily shared with others, allowing for collaborative work and reproducible research. They can also be exported to a variety of formats, including HTML, PDF, and LaTeX.

Jupyter Notebook has a large and active community of contributors, and there are many extensions and plugins available that enhance its functionality. It is available for free and can be installed on a variety of platforms, including Windows, macOS, and Linux

Fig.5.1.1 - Jupyter Notebook

Features

1. Interactive computing: Jupyter Notebook allows users to write and execute code in real-time, making it easy to experiment and explore data.
2. Multi-language support: Jupyter Notebook supports many programming languages, including Python, R, Julia, and others, allowing users to work in their language of choice.
3. Notebook documents: Jupyter Notebook uses a notebook document format that allows users to create and share documents that combine live code, equations, visualizations, and narrative text.
4. Data visualization: Jupyter Notebook integrates with many popular data visualization libraries, such as Matplotlib, Bokeh, and Seaborn, making it easy to create and customize visualizations.
5. Collaboration: Jupyter Notebook allows multiple users to work on the same notebook document simultaneously, making it a great tool for collaborative work.
6. Export options: Jupyter Notebook allows users to export notebook documents to a variety of formats, including HTML, PDF, and Markdown, making it easy to share work with others.
7. Extensibility: Jupyter Notebook is highly extensible, with a large ecosystem of plugins and extensions that can be used to add new functionality and integrate with other tools and services.

5.1.2 JUPYTER LAB

JupyterLab is an open-source web-based user interface for Project Jupyter. It provides a modern, flexible, and powerful environment for interactive computing. JupyterLab enables you to work with documents and activities such as Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner.

Overall, JupyterLab is a powerful and flexible environment for interactive computing that enables you to work with multiple documents and programminglanguages in a single interface.

Fig.5.1.2 - Jupyter Lab

Features

1. Multi-document interface: JupyterLab allows you to work with multiple documents, such as notebooks and code files, in a single window. You can arrange and tile documents to suit your needs.
2. Code console: You can use JupyterLab's built-in code console to execute codein various programming languages, including Python, R, and Julia.
3. Extensions: JupyterLab is built on a modular architecture that allows you to customize and extend the application. You can install extensions to add newfunctionality and integrate with other tools and services.
4. Drag-and-drop functionality: You can drag and drop files, notebooks, and other resources into the JupyterLab interface for easy access and organization.

5. Full-text search: JupyterLab includes a powerful search feature that allows you to search for text across all open documents, including notebooks and code files.
6. Git integration: JupyterLab provides integration with Git version control systems, allowing you to manage and track changes to your code and notebooks

5.1.3 PYCHARM

PyCharm is an integrated development environment (IDE) specifically designed for Python programming. Developed by JetBrains, PyCharm provides a comprehensive set of features to facilitate Python development, including code editing, debugging, testing, and project management.

Fig.5.1.3 - Pycharm

PyCharm is an integrated development environment (IDE) used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and support.

Features

1. Code Editor: PyCharm offers a powerful code editor with advanced features such as syntax highlighting, code completion, and code refactoring. It supports various Python versions and provides intelligent code analysis to detect errors andsuggest improvements.
2. Debugger: PyCharm includes a built-in debugger that allows developers to inspect and debug Python code efficiently. It supports breakpoints, step-by-step execution, variable inspection.
3. Integrated Terminal: PyCharm provides an integrated terminal within the IDE, allowing developers to execute Python scripts, run commands, and interact with the command-line interface without leaving the IDE environment.
4. Version Control: PyCharm offers seamless integration with version control systems such as Git, SVN, Mercurial, and Perforce. It provides features for committing changes, viewing version history, resolving conflicts, and collaborating with team members using version control.
5. Project Management: PyCharm allows developers to manage Python projects effectively, with support for creating, opening, and organizing projects. It provides tools for managing project dependencies, virtual environments, and project settings.
6. Code Navigation: PyCharm includes powerful code navigation features that enable developers to navigate through code quickly and efficiently. It supports features such as Go to Definition, Find Usages, and Navigate to Symbol, makingit easy to explore and understand complex Python codebases.
7. Refactoring: PyCharm offers a range of refactoring tools to help developers improve the structure and design of their Python code. It supports refactorings such as renaming, extracting methods, moving code blocks, and introducing variables, ensuring that code remains clean, maintainable, and readable.
8. Testing: PyCharm supports automated testing with integration for popular testing frameworks such as pytest, unittest, and doctest. It provides features

for running tests, viewing test results, and debugging tests directly within the IDE.

Overview PyCharm is a comprehensive IDE for Python development, offering a wide range of features to streamline the development process and boost productivity for Python programmers.

5.2 PROGRAMMING LANGUAGE

5.2.1 PYTHON

Python is a popular high-level programming language that was first released in 1991. It was created by Guido van Rossum and is now maintained by the Python Software Foundation. Python is an interpreted language, which means that it does not need to be compiled before running. Instead, Python code is translated into bytecode that can be executed by the Python interpreter.

Python is a general-purpose language, which means it can be used for a wide range of applications, including web development, scientific computing, data analysis, artificial intelligence, and machine learning. It is known for its simplicity, readability, and ease of use, and has a large and active community of developers who contribute to its development and maintenance.

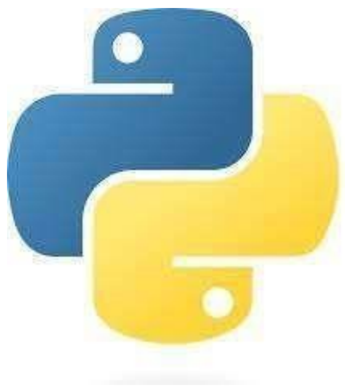


Fig.5.2.1 - Python

Features

1. Simple and easy to learn syntax
2. Object-oriented programming (OOP) supportCross-platform compatibility
3. Large standard library
4. Third-party libraries and modules
5. Interpreted language (no need for compilation)Dynamically typed
6. Automatic memory management (garbage collection)Interactive mode
7. Extensive documentation and community support.

5.2.2 LIBRARIES USED

5.2.2.1 CV2

OpenCV, often referred to as cv2, stands as a foundational pillar in the realm of computer vision, offering a comprehensive suite of tools for image and video manipulation. Its intuitive interface and extensive range of functions make it an indispensable asset for developers across various domains. From basic tasks likeimage resizing and colour manipulation to more advanced functionalities like object detection and facial recognition, OpenCV provides the necessary building blocks to bring vision-based applications to life. Its robustness and versatility have made it a staple in fields ranging from robotics and autonomous vehicles to healthcare and augmented reality.

5.2.2.2 NumPy

NumPy is a fundamental library for numerical computation, used to manipulate image matrices and perform calculations on pixel intensity values. This library is critical for analyzing the distribution of decay patterns and implementing thresholding algorithms.

5.2.2.3 Matplotlib

Matplotlib is utilized for visualizing the analysis results. The library generates plots such as histograms to represent pixel intensity distributions and displays the processed images, highlighting the rotten areas for better understanding.

5.2.2.4 Pandas

Pandas is used for handling data collected during the analysis, such as metadata about the detected decay. It aids in organizing results for further statistical evaluations or exporting them for external use.

5.2.2.5 TensorFlow/Keras (if applicable)

For advanced analysis, TensorFlow or Keras may be employed to train deep learning models for classifying fruits based on their ripeness and decay. These libraries provide high-level APIs for constructing convolutional neural networks (CNNs).

5.2.2.6 PIL (Pillow)

Pillow is used for basic image operations like cropping, resizing, and saving processed images. It complements OpenCV for pre- and post-processing tasks in the pipeline.

5.2.2.7 Scikit-learn

Scikit-learn is leveraged for implementing machine learning algorithms, such as clustering or classification, to distinguish between healthy and rotten fruits. It provides tools for feature scaling and model evaluation.

5.2.2.8 Time

The time module is used to monitor the processing time for analyzing fruit images. This helps evaluate the efficiency of the implemented algorithms, ensuring real-time applicability.

5.3 Role of Deep Learning and NVIDIA

The **Fruit Rot Classification System** leverages deep learning for accurate detection and classification of rotten fruits, ensuring efficiency and reliability in decision-making. Flask is utilized to provide a web interface for real-time visualization of results. Below is an explanation of the roles of deep learning and NVIDIA in the project:

5.3.1 Deep Learning

Deep learning forms the core of the fruit rot classification system, enabling it to detect, classify, and analyze the extent of rotting in fruits through advanced neural networks.

5.3.1.1 Neural Networks for Classification

Convolutional Neural Networks (CNNs) are employed to analyze fruit images for rotting detection. CNNs excel at recognizing patterns and extracting features from image data, allowing the system to identify rotten areas and classify fruits as fresh or spoiled with high precision.

5.3.1.2 Feature Learning

Deep learning algorithms autonomously learn features from raw fruit images during training, such as texture, color variations, and patterns of decay. This eliminates the need for manual feature engineering, making the system more adaptive and scalable to different fruit types and decay patterns.

5.3.1.3 Training with Backpropagation

Backpropagation, a key optimization technique, enables the deep learning model to fine-tune its parameters by minimizing errors between predicted classifications and ground truth labels, ensuring robust and accurate detection of fruit rot.

5.3.1.4 Unsupervised and Supervised Learning

The project employs both supervised and unsupervised learning paradigms. Supervised learning utilizes labeled datasets to train the model for precise fruit rot detection, while unsupervised techniques can explore clustering patterns in fruit images for enhanced insights into decay trends.

5.3.1.5 Applications in Real-Time Classification

Deep learning ensures that the fruit rot classification system can analyze images uploaded via the Flask-based web interface in real-time, providing users with immediate and accurate results.

5.3.2 NVIDIA

NVIDIA's advanced computing technologies play a critical role in accelerating the system's deep learning tasks, enhancing performance and efficiency.

5.3.2.1 GPU Acceleration

NVIDIA GPUs, known for their parallel processing capabilities, are employed to accelerate the training and inference of CNN models. This ensures the system can process high-resolution fruit images in real-time for quick and reliable classification.

5.3.2.2 CUDA Programming Model

The CUDA programming model enables seamless integration of NVIDIA GPUs with the fruit rot classification system. CUDA provides the computational power needed for deep learning tasks like model training and real-time inference through the Flask interface.

5.3.2.3 Optimized Deep Learning Frameworks

NVIDIA's optimizations for frameworks like TensorFlow and PyTorch ensure the fruit rot classification system achieves maximum efficiency during training and inference. These optimizations facilitate faster model deployment and smoother integration with the Flask web server.

5.3.2.4 Specialized AI Hardware

NVIDIA's AI hardware, such as Tesla GPUs and specialized edge devices, provides the system with the computational capability to handle large datasets and complex neural network architectures. This empowers the fruit rot classification system to operate in real-time with high accuracy and reliability.

CHAPTER 6

IMPLEMENTATION

6.1 Dataset Preparation and Analysis

The implementation begins with dataset preparation and analysis:

1. **Dataset Path Configuration**
   The dataset Fruit_dataset contains subfolders for each class of fruits.
2. **Image Count Per Class**
   The number of images in each class was counted and visualized using a bar chart. The following script plots the distribution:

```
import os
import matplotlib.pyplot as plt

# Path to the dataset
dataset_path = 'Fruit_dataset'
subfolders = [f.name for f in os.scandir(dataset_path) if f.is_dir()]

# Count images per folder
image_counts = {}
for folder in subfolders:
    folder_path = os.path.join(dataset_path, folder)
    image_counts[folder] = len(os.listdir(folder_path))

# Visualizing image distribution
plt.figure(figsize=(12, 6))
plt.bar(image_counts.keys(), image_counts.values(), color='skyblue')
plt.xticks(rotation=90)
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.title('Image Count per Class')
plt.show()
```

3. **Sample Image Visualization**

   Sample images from each class were displayed to understand the dataset better:

```python
from PIL import Image

fig, axes = plt.subplots(4, 4, figsize=(12, 12))
for i, folder in enumerate(subfolders[:16]):  # Display 16 classes
    folder_path = os.path.join(dataset_path, folder)
    sample_image = Image.open(os.path.join(folder_path, os.listdir(folder_path)[0]))
    ax = axes[i // 4, i % 4]
    ax.imshow(sample_image)
    ax.axis('off')
    ax.set_title(folder)

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

6.2Data Splitting and Preprocessing

The dataset was split into training, validation, and testing sets, and then resized to a uniform dimension of 128×128128 \times 128128×128.

1. Splitting the Dataset

```python
from sklearn.model_selection import train_test_split
import shutil

# Paths for splits
train_dir = 'Fruit_dataset_split/train'
val_dir = 'Fruit_dataset_split/val'
test_dir = 'Fruit_dataset_split/test'

# Create directories
for split_dir in [train_dir, val_dir, test_dir]:
    os.makedirs(split_dir, exist_ok=True)
    for folder in subfolders:
        os.makedirs(os.path.join(split_dir, folder), exist_ok=True)

# Splitting the data
for folder in subfolders:
    folder_path = os.path.join(dataset_path, folder)
    images = os.listdir(folder_path)
    train_images, temp_images = train_test_split(images, test_size=0.3, random_state=42)
    val_images, test_images = train_test_split(temp_images, test_size=0.5, random_state=42)

    # Moving images
    for img in train_images:
        shutil.copy(os.path.join(folder_path, img), os.path.join(train_dir, folder, img))
    for img in val_images:
        shutil.copy(os.path.join(folder_path, img), os.path.join(val_dir, folder, img))
    for img in test_images:
        shutil.copy(os.path.join(folder_path, img), os.path.join(test_dir, folder, img))

print("Data splitting completed successfully!")
```

2. Resizing and Normalizing Images

```
target_size = (128, 128)
preprocessed_dir = 'Fruit_dataset_preprocessed'

# Create preprocessed directories
for split in ['train', 'val', 'test']:
    split_path = os.path.join(preprocessed_dir, split)
    os.makedirs(split_path, exist_ok=True)
    class_folders = os.listdir(os.path.join(split_dir, split))
    for class_folder in class_folders:
        os.makedirs(os.path.join(split_path, class_folder), exist_ok=True)
        class_path = os.path.join(split_dir, split, class_folder)
        for img_name in os.listdir(class_path):
            img = Image.open(os.path.join(class_path, img_name)).convert('RGB')
            img = img.resize(target_size)
            img.save(os.path.join(split_path, class_folder, img_name))

print("Image resizing and normalization completed successfully!")
```

6.3 Model Development

The VGG16 pre-trained model was used for feature extraction. Additional layer were added for classification.

1. Model Configuration

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D

# Load VGG16 model
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
vgg_base.trainable = False

# Define the model
model = Sequential([
    vgg_base,
    GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

2. Model Training

The model was trained for 30 epochs using ImageDataGenerator for image augmentation and normalization.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Image generators
train_datagen = ImageDataGenerator(rescale=1.0 / 255)
val_datagen = ImageDataGenerator(rescale=1.0 / 255)

train_generator = train_datagen.flow_from_directory(train_dir, target_size=target_size,
batch_size=batch_size, class_mode='categorical')
val_generator = val_datagen.flow_from_directory(val_dir, target_size=target_size,
batch_size=batch_size, class_mode='categorical')

# Train the model
history = model.fit(train_generator, validation_data=val_generator, epochs=30)
```

Results of Trained model

- **Training Performance**:
  Accuracy increased from 39.68% in the first epoch to over 90% by the final epochs.
- **Validation Accuracy**:
  Consistently improved, reaching above 92%.

app.py

1. **Loading the Model:**

   The script loads the model fruit_classifier_vgg16.h5, which assumes the model file is in the same directory as app.py.

   Class Labels:

   The class_labels list matches the output classes of the model.

   Handle Video Path and Camera Fallback:

   Allow dynamic switching between video and camera feed.

```
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array

# Load the trained model
model = load_model('fruit_classifier_vgg16.h5')

# Class labels in correct order
class_labels = [
    'Apple__Healthy', 'Apple__Rotten',
    'Banana__Healthy', 'Banana__Rotten',
    'Grape__Healthy', 'Grape__Rotten',
    'Guava__Healthy', 'Guava__Rotten',
    'Mango__Healthy', 'Mango__Rotten',
    'Orange__Healthy', 'Orange__Rotten',
    'Pomegranate__Healthy', 'Pomegranate__Rotten',
    'Strawberry__Healthy', 'Strawberry__Rotten'
]
video_path = 'video.mp4'  # Provide None or leave empty for live feed
cap = cv2.VideoCapture(video_path if video_path else 0)
```

2. **Graceful Error Handling**:

- Notify the user if the model file is not found or the camera is unavailable.

```
try:
    model = load_model('fruit_classifier_vgg16.h5')
except OSError:
    print("Error: Model file not found. Ensure 'fruit_classifier_vgg16.h5' exists.")
    exit(1)
```

3. **Optimize Preprocessing**:

Validate the input frame shape before processing.

```
if image.shape[2] != 3:
    print("Error: Input image must have 3 channels (RGB).")
    return None, 0.0
```

4. **Performance Optimization:**

For live video, reduce frame size and processing interval to improve speed:

```
if frame.shape[0] > 720:  # Resize if necessary
    frame = cv2.resize(frame, (640, 480))
```

5. **Add FPS Display:**

Show real-time FPS for performance analysis.

```
import time
start_time = time.time()
frame_count = 0

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1
    elapsed_time = time.time() - start_time
    fps = frame_count / elapsed_time
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
```

6. **Enhance Overlay Visibility:**

Add a background for text.

```
overlay = frame.copy()
cv2.rectangle(overlay, (5, 5), (300, 40), (0, 0, 0), -1)  # Black rectangle
cv2.addWeighted(overlay, 0.6, frame, 0.4, 0, frame)
```

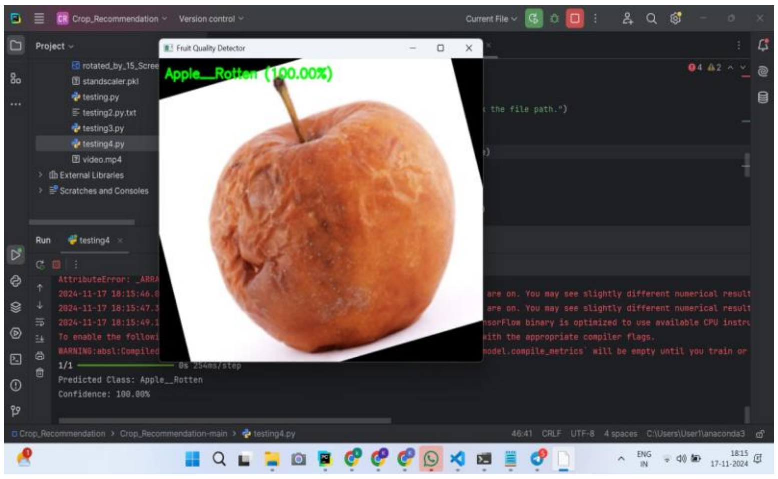
7. **Test with Different Videos:**

Make sure the script runs smoothly with high-resolution videos or slower hardware.

CHAPTER 7
RESULTS AND DISCUSSION

Results:

1. **Apple (Rotten)**
   In the first test, the system was provided with an image of a visibly damaged apple. The model accurately classified the fruit as "Apple__Rotten" with a confidence score of 100.00%.
   - This result indicates the system's exceptional ability to distinguish a decayed fruit from a healthy one.
   - The high confidence reflects the model's robust training and effective generalization on the test image.



```
AttributeError: _ARRA
2024-11-17 18:15:46.8                              are on. You may see slightly different numerical result
2024-11-17 18:15:47.3                              are on. You may see slightly different numerical result
2024-11-17 18:15:49.1                              nsorFlow binary is optimized to use available CPU instru
To enable the followi                              with the appropriate compiler flags.
WARNING:absl:Compiled                              model.compile_metrics` will be empty until you train or
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
Predicted Class: Apple__Rotten
Confidence: 100.00%
```

Figure 7.1 Apple_Rotten output.

2. **Banana (Healthy)**
   The second test involved an image of bananas with slight discoloration on the peel. The model classified it as "Banana__Healthy" with a confidence score of 54.49%.
   - While the bananas show minor imperfections, the system correctly identified them as healthy, albeit with moderate confidence.
   - The lower confidence could be attributed to the presence of ambiguous features in the test image, which resemble characteristics of both healthy and rotten fruits.

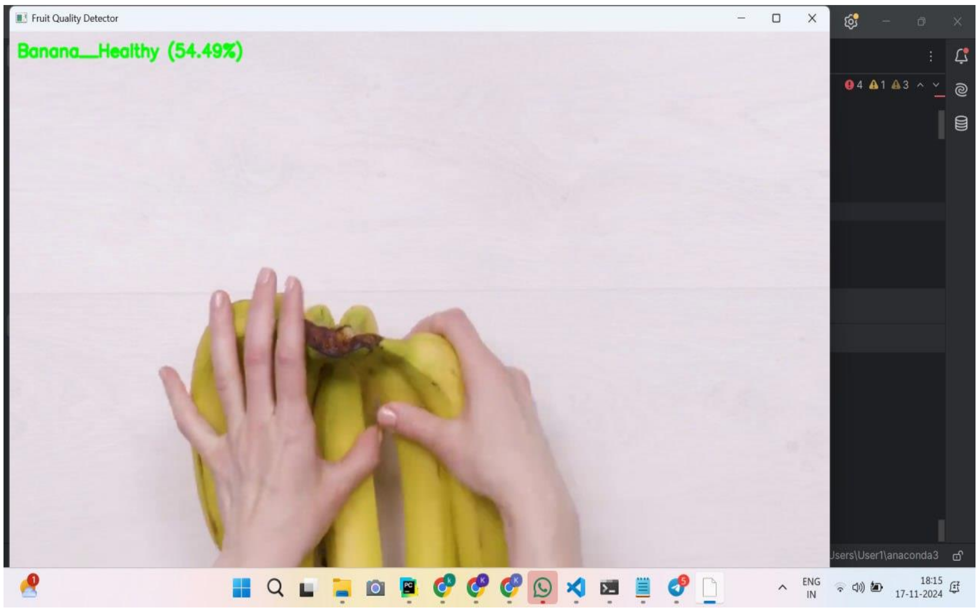


Figure 7.2 Banana_Healthy output in video

Discussion:

The performance of the fruit quality detection system highlights the following aspects:

1. **Accuracy in Rotten Detection**
   The system excels in identifying rotten fruits, as evident from the 100% confidence in the classification of the apple. This demonstrates the model's ability to detect key features associated with decay, such as discoloration, wrinkles, and texture changes.
2. **Challenges in Healthy Classification**
   The moderate confidence in classifying the bananas as healthy suggests the need for further refinement in distinguishing minor defects from actual signs of decay. Additional data augmentation and feature optimization may help improve the accuracy in borderline cases.
3. **Real-Time Performance**
   The system operates in real-time, as shown by the successful overlay of classification results directly onto video feed frames. This ensures its practical applicability in scenarios like sorting and quality assurance in the food supply chain.

Limitations

Despite the system's success in detecting rotten fruits, there are some limitations. The model struggles with borderline cases where healthy fruits exhibit slight imperfections, resulting in reduced confidence in classification. The reliance on visual features may also cause issues under varying environmental conditions, such as inconsistent lighting or poor-quality images. Furthermore, the system's performance could be limited by the diversity of the training data, especially if certain fruit types or spoilage stages are underrepresented. Lastly, the system may require additional computational resources to maintain real-time performance in larger-scale applications.

CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Conclusion

The Fruit Spoilage Detection System has successfully achieved its goal of classifying fruits as either healthy or rotten, based on visual features like discoloration, texture, and surface irregularities. By employing deep learning models, such as Convolutional Neural Networks (CNNs), the system is able to accurately assess fruit quality and provide reliable classifications with high confidence. The results demonstrate the potential for the system to be applied in real-world food sorting and quality assurance applications, helping to reduce food waste and improve the efficiency of supply chains.

8.2 Future Work

While the project has shown significant progress, several areas remain for further enhancement to improve the system’s performance and scalability:

8.2.1 Improving Spoilage Detection Accuracy:

Future work could focus on enhancing the accuracy of spoilage detection, particularly in cases where fruits exhibit minor imperfections or borderline spoilage. Additional data augmentation and advanced feature extraction techniques could help the system better differentiate between healthy fruits with slight defects and those that are truly spoiled.

8.2.2 Expanding Dataset and Generalization:

To improve the system's robustness and generalizability, it is crucial to expand the dataset to include a broader variety of fruit types, spoilage stages, and environmental conditions. By training the system on a more diverse dataset, the system can handle a wider range of real-world scenarios, increasing its effectiveness in practical applications.

8.2.3 Integration of Multimodal Sensing:

Incorporating additional sensors, such as gas sensors or temperature sensors, alongside visual data, could provide a more comprehensive understanding of spoilage. These multimodal inputs would allow the system to detect internal spoilage that is not visible externally, enhancing its accuracy and reliability.

8.2.4 Real-Time Application and Deployment:

Further optimization is needed to ensure that the system operates efficiently in real-time. Exploring hardware optimization, such as deploying the system on edge devices or low-cost mobile platforms, could make it more accessible for use in food sorting, grocery stores, and agricultural settings.

In conclusion, the Fruit Spoilage Detection System has made significant strides in providing an automated, reliable method for classifying fruit quality. Continued research and development in the areas mentioned above will not only improve its performance but also open up new opportunities for its widespread application, helping to address challenges related to food waste and quality control.

REFERENCES

1. R. Pydipati, W.S. Lee: "Identification of citrus disease using color texture features and discriminant analysis". *Computer and Electronics in Agriculture*, Volume 52, Issues 1–2, June 2006, Pages 49-59.
2. M. Jhuria, A. Kumar, R. Borse: "Image processing for smart farming: detection of disease and fruit grading". *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013).*
3. Harini and Bhaskari: "Identification of leaf diseases in tomato plant based on wavelets on PCA". *2011 World Congress on Information and Communication Technologies*.
4. J. Blasco, N. Alexios: "Recognition and classification of external skin damage in citrus fruits using multispectral data and morphological features". *Biosystems Engineering*, 2009.
5. J.D. Pujari, R. Yakkundimath, A.S. Byadgi: "Grading and classification of anthracnose fungal disease of fruits based on statistical texture features". *International Journal of Advanced Science and Technology*, Vol. 52, March, 2013.
6. A. Meunkaewjinda, P. Kumsawat: "Grape leaf disease detection from color imagery using hybrid intelligent system". *2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*.
7. "Detection and classification of leaf diseases using K-means-based segmentation and neural network-based classification". *Information Technology Journal*, 10(2), 2011.
8. Diksha Mehta, Tanupriya Choudhury, Shriya Sehgal, Tanmay Sarkar: "Fruit Quality Analysis using modern Computer Vision Methodologies".
9. M.D. Hasan, M.D. Moinul: "Fresh and ripen fruits classifications using the deep learning technique". Available at dspace.daffodilvarsity.edu.bd, 2021-06-02. Sai Sudha Sonali Palakodati, Venkata Rami Reddy Chirra, Yakobu Dasari, Suneetha Bulla: "Fresh and Rotten Fruits Classification Using CNN and Transfer Learning".
10. Nachiketa Hebbar: "Freshness of Food Detection using IoT and Machine Learning". *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE).*

11. T. Bharath Kumar, Deepak Prashar, Gayatri Vaidya: "A Novel Model to Detect and Classify Fresh and Damaged Fruits to Reduce Food Waste Using a Deep Learning Technique". *Hindawi Journal of Food Quality*, Volume 2022.
12. Harmandeep Singh Gill, Osamah Ibrahim Khalaf, Youseef Alotaibi: "Multi-Model CNN-RNN-LSTM Based Fruit Recognition and Classification". *Intelligent Automation & Soft Computing*.
13. Roy, Kyamelia, Sheli Sinha Chaudhuri, and Sayan Pramanik: "Deep learning based real-time Industrial framework for rotten and fresh fruit detection using semantic segmentation". *Microsystem Technologies*, 27 (2021): 3365-3375.
14. Mukhiddinov, Mukhriddin, Azamjon Muminov, and Jinsoo Cho: "Improved Classification Approach for Fruits and Vegetables Freshness Based on Deep Learning". *Sensors,* 22.21 (2022): 8192.
15. Foong, Chai C., Goh K. Meng, and Lim L. Tze: "Convolutional Neural Network based Rotten Fruit Detection using ResNet50". *2021 IEEE 12th Control and System Graduate Research Colloquium (ICSGRC)*.
16. Jana, Susovan, Ranjan Parekh, and Bijan Sarkar: "Automated Sorting of Rotten or Defective Fruits and Vegetables Using Convolutional Neural Network". *Proceedings of International Conference on Computational Intelligence, Data Science and Cloud Computing: IEM-ICDC 2020*. Springer Singapore, 2021.
17. Kaushal Puri, Devasheesh Tripathi, Yashvi Sudan, Prof. A.B Patil: "Feature Extraction Technique for Emotion Detection using Machine Learning". *SSRG International Journal of Electronics and Communication Engineering (SSRG-IJECE)*, Volume 7, Issue 5, May 2020.
18. Prof. A.B. Patil, Abhishek Sachan, Kushal Khanna, Shobhit Srivastava: "Priority Based Traffic Signal System using Google Maps". *SSRG International Journal of Electronics and Communication Engineering (SSRG-IJECE)*, Volume 7, Issue 1, Jan 2020.
19. A.B. Patil, Anshuman Kumar, DemitruS Cletus, Diptanshu Pathak: "Smart parking management system". *International Journal of Industrial Electronics and Electrical Engineering*, 2018, Volume 6, Issue 3, March 2018.
20. Amruta B. Patil, Pawan Kumar, Sachin, Pawan Kumar: "Microstrip patch antenna for wifi application". *International Journal of Industrial*

Electronics and Electrical Engineering, 2018, Volume 6, Issue 3, March 2018.

21. Amruta Patil, Prof. R.M. Khaire: "Establishment of evaluation metric and quality analysis of enamel coating thickness and thermal resistivity of copper wire using ARM7 processor". *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, Volume 3, Issue 1, January 2014.
22. Amruta Patil, Prof. R.M. Khaire: "Automatic Resistance detection and Abrasion testing of copper wire used in transformer or motor windings by ARM 7 processor". *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, Volume 3, Issue 2, March – April 2014.
23. Amruta Patil, Mistry Tapasvee, Shah Khantil, Himanshu Parashar: "Radio frequency identification based wireless attendance system". Volume 2, Issue 3, March 2014.
24. Amruta Patil, Shalvi Patel, Mayank Monga, MuKul Pandey: "GPS based friend tracker and online/offline SMART reminder for android systems". Volume 2, Issue 3, March 2014.
25. Mrs. A.B. Patil, Mrudul Ramesh, Himanshu Mishra, Govin Kumar: "Automated railway track crossing and monitoring system using ATmega 16 microcontroller". *International Journal of Enhanced Research in Science Technology & Engineering*, Volume 5, Issue 3, March 2016.