1. INTRODUCTION

1.1 INTRODUCTION

Precision agriculture has revolutionized modern farming practices by focusing on optimizing resource allocation, enhancing productivity, and minimizing environmental impact. A critical aspect involves assessing crop attributes, particularly fruit ripeness, influencing harvest timing and quality. Accurate ripeness detection is vital for informed decision-making in harvesting, post-harvest handling, and marketing. Timely assessments maximize yield, minimize waste, and optimize the supply chain for consumer satisfaction, taste, and nutrition.

Vision-based methods, leveraging visual cues like colour and texture, offer non-destructive, real-time, scalable solutions for fruit ripeness detection. Deep learning, especially Convolutional Neural Networks (CNNs), shows promise in automatically learning complex features from large datasets. Challenges persist, including achieving high accuracy in diverse scenarios with variations in lighting, fruit sizes, and occlusions.

This study proposes a vision-based deep learning approach using CNNs to automatically learn discriminative features for accurate fruit ripeness classification. Custom datasets for training, validation, and testing will ensure method effectiveness. Challenges include developing efficient, lightweight models for deployment in resource-constrained devices like drones or embedded systems. Addressing these issues is crucial for advancing fruit ripeness detection in precision agriculture.

1.2 NECESSITY

1.2.1 **Precision Agriculture Advancements:** Advanced technologies are crucial for precision agriculture, including machine vision-based pomegranate ripeness detection systems, which aim to optimize resource utilization and improve agricultural efficiency.

1.2.2 **Optimizing Harvesting Practices:** Accurate pomegranate ripeness determination is crucial for optimizing harvesting practices, maximizing yield and minimizing waste, thus requiring a reliable system.

1.2.3 **Supply Chain Optimization:** Precise ripeness detection contributes to the optimization of the supply chain. Ensuring that consumers receive fruits with optimal taste, texture, and nutritional value enhances the overall quality of agricultural produce in the market.

1.2.4 **Enhancing Post-Harvest Handling:** Accurate ripeness detection supports effective post-harvest handling and storage logistics. It prevents spoilage and extends the shelf life of pomegranates, reducing economic losses and enhancing the efficiency of post-harvest processes.

1.2.5 **Meeting Market Demands:** Aligning with consumer preferences for high-quality produce, the project ensures that the market receives fruits with optimal ripeness, taste, and nutritional

content. This responsiveness to market demands enhances the competitiveness of agricultural products.

1.2.6 **Overcoming Limitations of Traditional Methods:** Traditional methods for assessing ripeness may be subjective and prone to errors. A machine vision-based approach, particularly utilizing deep learning, overcomes these limitations by providing objective, data-driven, and potentially more accurate results.

1.3 OBJECTIVE

The primary objectives of this project include:

1.3.1 **Dataset Creation:** Generate a diverse dataset of pomegranate images, covering ripeness stages, lighting variations, and potential occlusions, serving as the foundation for training and validating the machine vision system.

1.3.2 **CNN Implementation:** Design and implement a Convolutional Neural Network (CNN) to automatically extract pertinent features from pomegranate images, fine-tuning the model for high accuracy across different ripeness scenarios.

1.3.3 **Real-time Optimization:** Optimize the machine vision system for real-time performance, addressing processing speed, resource efficiency, and adaptability for practical deployment in agricultural settings.

1.3.4 **Environmental Robustness:** Investigate and implement strategies to enhance the system's robustness against environmental variables, including varying lighting conditions and occlusions, ensuring consistent accuracy in real-world agricultural scenarios.

1.3.5 **Device Adaptation:** Adapt the machine vision system for deployment on resource-constrained devices like drones or embedded systems in precision agriculture. Develop efficient, lightweight models that balance accuracy with the limited computational capabilities of these devices.

1.4 THEME

The initial focus is on software, where an advanced mobile application will be created using machine vision and deep learning algorithms. This app will accurately assess pomegranate ripeness through image analysis. Subsequently, the project transitions to hardware, aiming to deploy the ripeness detection system on resource-constrained devices like drones or embedded systems used in precision agriculture. This dual-phase approach ensures a holistic solution, combining cutting-edge software capabilities with practical and efficient hardware deployment for real-world agricultural applications.

2. LITERATURE SURVEY

2.1 Implementation of Fruit Detection System and using Computer Vision:

In this paper the author has used *TensorFlow* library to build a neural network for the classification and grading of fruits, we have also used *TensorFlow* library to create our CNN model for classification of the fruit & author have used *OpenCV* library for the real time detection of fruits using camera and classification of fruits is done using *support vector machine* algorithm. In our project we have also used *support vector machine* algorithm and *OpenCV* library[1]

2.2 Ripe-Unripe: Machine Learning based Ripeness Classification:

In this paper the author has used the Faster Region-based Convolutional Neural Network (Faster RCNN) with InceptionV2 for ripeness. We have referred the methodology to develop a CNN model for fruit ripeness detection and also classification and regression algorithm are used.[2]

2.2 Autonomous Fruit Detection Using Image Processing for an Agricultural Robotic System:

In this paper the author discusses about detection of the ripe and turning tomato fruits using computer vision and image processing techniques such as *OpenCV* & *HSV* color space, Moreover the paper also compares different classification techniques concludes that the KNN classifier for image segmentation has the higher accuracy of 99.33%. [3]

2.4 Detection of Fruit Ripeness Using Image Processing:

In this book the author has developed an image processing system for assessing fruit ripeness using color analysis in the Lab color space. The system compares two colour-separated images through correlation, focusing on colour information. Histogram comparison enhances contrast and intensity distribution, determining fruit ripeness levels and distinguishing between ripe and unripe states. We will be focusing on using this to distinguish fruit according to ripeness to grade them. [4]

3. SYSTEM DEVELOPMENT

3.1 Methodology :

The proposed system endeavours to construct a machine-learning model utilizing the Convolutional Neural Network (CNN) approach to differentiate between ripe and unripe fruits in real-time through Object Detection methodology. A well-suited dataset is imperative for the efficacy of any machine learning implementation, and in our project, it comprises images of various fruits gathered from open sources on the internet. To enhance the dataset, we apply data augmentation and image preprocessing techniques, ensuring a larger and more uniform set of data. The CNN algorithm is deployed on the augmented dataset to train the model. CNN, emulating human neuron functionality, autonomously extracts diverse features from the data, utilizing them to calculate the likelihood of each class during predictions. Consequently, when an image is input into the system, the model can accurately identify the fruit and its ripeness.

3.2 System Architecture :

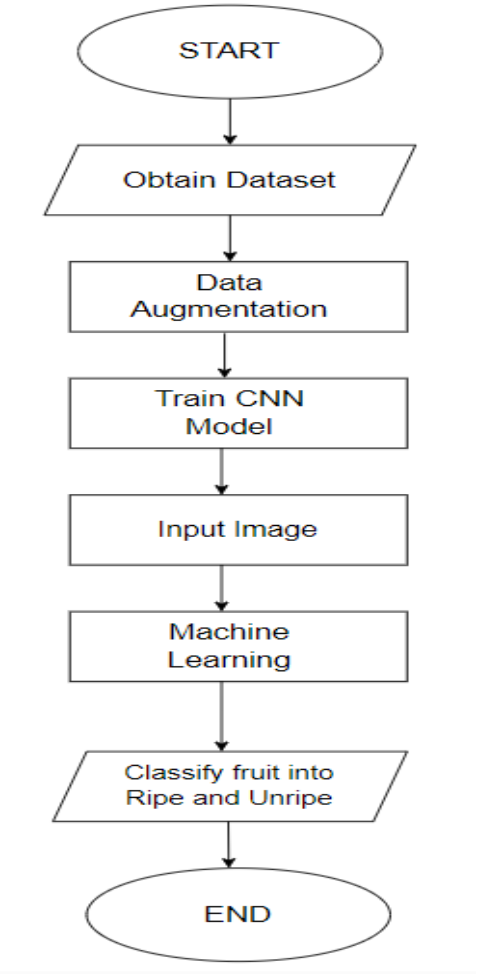

Fig. 3.2 Flowchart of the proposed system architecture

Presented in Figure 3.2 , the proposed system architecture is detailed in this section, outlining a pipeline that includes dataset acquisition, data augmentation, convolutional neural network (CNN) model training, input image processing, machine learning, and classification of fruits into ripe and unripe

categories. Commencing a machine learning project involves obtaining a dataset, and in this instance, images of both ripe and unripe fruits are collected from diverse sources such as the Internet, Github, and Kaggle. Following dataset collection, data augmentation is performed to enhance the dataset's diversity and prevent overfitting, achieved through transformations like rotations and flips. For real-time processing, captured images undergo preprocessing steps like resizing and background removal. Machine learning, a facet of artificial intelligence, involves training models to draw conclusions from data patterns. Here, a CNN model is employed to classify fruits based on ripeness.

3.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) represent a category of neural networks primarily employed for image classification tasks. As such, we have chosen CNNs as the technique for fruit classification into ripe or unripe categories, forming the basis for our machine learning model. The utilization of pre-trained models facilitates the implementation of CNN, creating a trained system proficient in classifying identified fruits. CNN offers advantages over Artificial Neural Networks (ANN) in image classification by concurrently executing both feature selection and extraction.

Subsequently, the next step involves training a CNN model using the acquired dataset and any augmented data. Throughout the training process, the model acquires the ability to discern patterns within the data, enabling it to classify fruits (ripe or unripe) for each input image. Once trained, the model is capable of classifying new fruit images. The input image is fed into the trained model, generating a prediction regarding the fruit's ripeness based on the model's output. This classification process is automated through a computer program utilizing the predictions generated by the trained model.

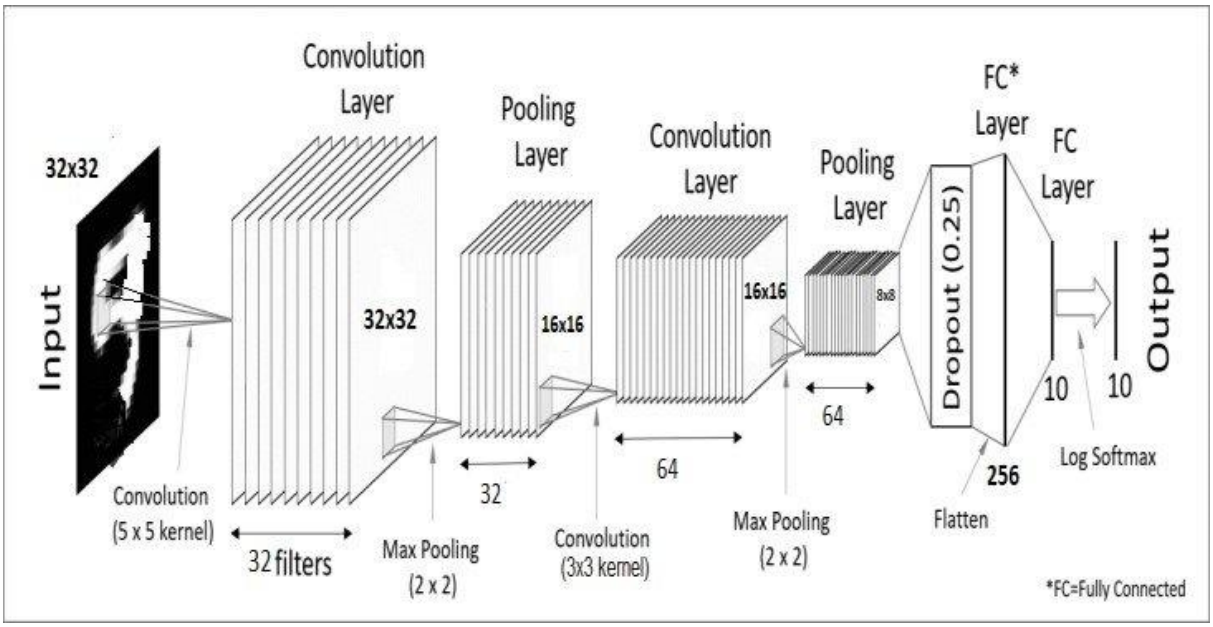

Fig 3.3 Design Architecture of CNN

3.3.1 Architecture/ Development Stages of Convolutional Neural Network :

Input Layer: This is where your network takes in the raw input data. In image-based tasks, each neuron in this layer represents a pixel value of the image.

Convolutional Layer (5x5): The convolutional layer applies filters (small 5x5 grids in this case) to the input image. These filters slide over the input, helping the network detect patterns like edges, textures, or other features.

Convolutional Layer: Another convolutional layer is applied, possibly with a different set of filters. These layers progressively extract more complex features from the input data.

Pooling Layer (32): Pooling layers down sample the spatial dimensions of the previous layers. Max pooling, for example, takes the maximum value from a group of values, reducing the size of the data while preserving important features. The '32' refer to the number of filters used.

Convolutional Layer (3x3): Similar to the earlier convolutional layers, this layer continues to extract higher-level features from the down sampled input.

Pooling Layer (64): Another pooling layer further reduces the spatial dimensions, and '64' indicate the number of filters used.

Dropout Layer: Dropout is a regularization technique. During training, randomly selected neurons are ignored, which helps prevent overfitting by forcing the network to not rely too much on any particular set of neurons.

Flatten Layer: The flatten layer converts the multi-dimensional output of the previous layers into a one-dimensional vector. This flattened vector is then fed into the fully connected layers.

Fully Connected (Dense) Layers: These layers connect every neuron to every neuron in the previous and next layers. The fully connected layers use the information learned by the previous layers to make predictions.

Output Layer: The final layer produces the output of the network. For example, in a classification task, it might represent the different classes, and the neuron with the highest activation indicates the predicted class.

3.4 Max Pooling in Convolutional Neural Networks (CNNs):

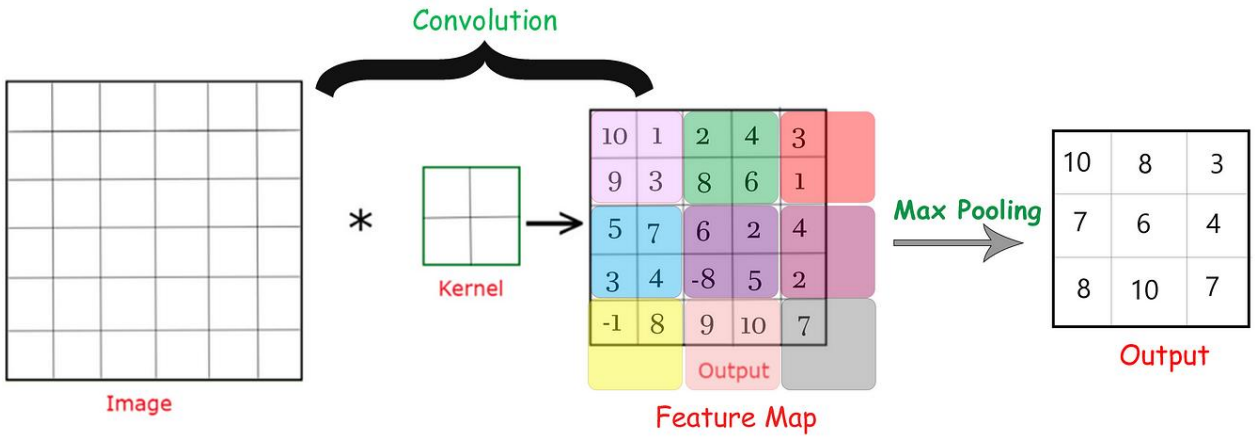

Fig 3.4.1 Max Pooling in CNN Development

3.4.1 Overview: Max pooling is a down-sampling operation commonly used in Convolutional Neural Networks (CNNs) after convolutional layers. It plays a crucial role in reducing the spatial dimensions of the input volume, which helps in decreasing the computational complexity of the network and controlling overfitting.

3.4.2 How Max Pooling Works: Max pooling divides the input into non-overlapping rectangles and, for each sub-region, outputs the maximum value. The process involves sliding a window (usually 2x2 or 3x3) across the input and selecting the maximum value within each window. The output is a down-sampled version of the input with reduced spatial dimensions.

3.4.3 Advantages of Max Pooling:

Spatial Hierarchical Representation: Max pooling retains the most significant information from the input. By selecting the maximum value, it captures the most activated features in a local region. This helps maintain the hierarchical representation of features throughout the network.

Translation Invariance: Max pooling provides a degree of translation invariance. Since it selects the maximum value within a region, the exact location of the feature in that region is less critical. This can make the network more robust to variations in the position of features within the input.

Reduction of Computational Complexity: By reducing the spatial dimensions, max pooling reduces the number of parameters and computations in subsequent layers. This is beneficial for managing computational resources, especially in deeper networks where the number of parameters can become substantial.

Noise Robustness: Max pooling can help the network become more robust to noise in the input. By selecting the maximum activation, it is less sensitive to small variations or noise in the local region.

3.4.4 Reasons for Use:

Down-sampling: Max pooling is primarily used for down-sampling the spatial dimensions of the input volume. This is essential for managing computational complexity and memory requirements, especially in large-scale CNNs.

Feature Selection: It helps in selecting the most important features from each local region, contributing to the extraction of essential information while discarding less relevant details.

Hierarchy Preservation: Max pooling contributes to preserving the hierarchical structure of features in the network, allowing it to learn and recognize increasingly complex patterns in higher layers.

4. MACHINE LEARINING MODEL DEVELOPMENT

4.1 Creating data set of Fruits in three Separate folder as training, testing & validation Upload the Dataset to Google Drive and Mount the Drive in Google Collaboratory.

```
from google.colab import drive
drive.mount('/content/drive')
```

4.2 Importing Required Libraries

```
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

import numpy as np powerful numerical computing library in Python. It is commonly used for array operations and mathematical computations.

import tensorflow as tf: TensorFlow is widely used for building and training machine learning models.

from keras.preprocessing.image import ImageDataGenerator: This class is often used for data augmentation and preprocessing of image data during model training.

import matplotlib.pyplot as plt: It is commonly used for creating visualizations, including plots and charts.

4.3 Data Preprocessing

4.3.1 Training Image preprocessing

```
training_set = tf.keras.utils.image_dataset_from_directory(
    train_data_dir,
    labels="inferred",
    label_mode="categorical",
    color_mode="rgb",
    batch_size=batch_size,
    image_size=image_size,
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False)
```

image_dataset_from_directory: This function is part of TensorFlow's Keras utilities and is used to create a dataset from images stored in a directory. It automatically infers labels, applies categorical encoding, and performs other configurations specified in the parameters.

Parameters like **labels**, **label_mode**, **color_mode**, **batch_size**, **image_size**, etc., are set according to the requirements of your training data. For example, setting **label_mode** to "categorical" indicates that the labels are one-hot encoded.

4.3.2 Validation Image Preprocessing

This runs the same code but the directory is changed for the validation folder

4..4 Model Selection - Convolutional Neural Network (CNN)

```
cnn = tf.keras.models.Sequential()
```

cnn is an instance of a Sequential model, and you can add layers to it using methods like **add()**. This model can be used to define and build a Convolutional Neural Network (CNN) architecture for tasks such as image classification.

4.5 Building Convolution Layer

```
# Add the first convolutional layer
cnn.add(tf.keras.layers.Conv2D(
    filters=32,
    kernel_size=3,
    padding='same',
    activation='relu',
    input_shape=[64, 64, 3]
))

# Add the second convolutional layer
cnn.add(tf.keras.layers.Conv2D(
    filters=32,
    kernel_size=3,
    activation='relu'
))

# Add a max pooling layer
cnn.add(tf.keras.layers.MaxPool2D(
    pool_size=2,
    strides=2
))
```

tf.keras.layers.Conv2D: This layer creates a convolutional layer with specified parameters like the number of filters, kernel size, padding, and activation function.

tf.keras.layers.MaxPool2D: This layer adds a max pooling operation for spatial down-sampling.

The **input_shape** parameter is specified only in the first convolutional layer, indicating the shape of the input data. Subsequent layers automatically infer the input shape from the output shape of the previous layer.

4.5 Avoid Overfitting

```
cnn.add(tf.keras.layers.Dropout(0.5)) #To avoid overfitting
```

cnn.add(tf.keras.layers.Dropout(0.5)) adds a Dropout layer to the Convolutional Neural Network (CNN). The purpose of using Dropout is to mitigate overfitting in the model during the training phase.

Overfitting occurs when a model learns not only the underlying patterns in the training data but also captures noise and random fluctuations that are specific to the training set.

Role of Dropout: Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of input units to zero during each update during training. Essentially, it "drops out" some neurons during training.

4.6 Compiling and Training Phase

```
cnn.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accura
cy'])
cnn.summary()
```

This Command Code Will Evaluate the Trained data on the Sequential Model and the results displayed are attached below in Figure 4.6

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 96, 96, 16)        1216

 max_pooling2d (MaxPooling2D  (None, 48, 48, 16)       0
 )

 dropout (Dropout)           (None, 48, 48, 16)        0

 conv2d_1 (Conv2D)           (None, 44, 44, 32)        12832

 max_pooling2d_1 (MaxPooling  (None, 22, 22, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 22, 22, 32)        0

 conv2d_2 (Conv2D)           (None, 18, 18, 64)        51264

 max_pooling2d_2 (MaxPooling  (None, 9, 9, 64)         0
 2D)

 dropout_2 (Dropout)         (None, 9, 9, 64)          0

 conv2d_3 (Conv2D)           (None, 5, 5, 128)         204928

 max_pooling2d_3 (MaxPooling  (None, 2, 2, 128)        0
 2D)

 dropout_3 (Dropout)         (None, 2, 2, 128)         0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 64)                32832

 dropout_4 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 16)                1040

 dropout_5 (Dropout)         (None, 16)                0

 dense_2 (Dense)             (None, 12)                204

=================================================================
Total params: 304,316
Trainable params: 304,316
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.6 Trained Model Summary

4.7 Evaluating Model

```
#Training set Accuracy
train_loss, train_acc = cnn.evaluate(training_set)
print('Training accuracy:', train_acc)
```

Results : `38/38 [==============================] - 51s 1s/step - loss: 0.2270 - accuracy: 0.9522`

```
Training accuracy: 0.9522212743759155
```

```
#Validation set Accuracy
val_loss, val_acc = cnn.evaluate(validation_set)
print('Validation accuracy:', val_acc)
```

Validation set accuracy is a metric used to evaluate the performance of a machine learning model during training, particularly in the context of classification tasks.

```
5/5 [==============================] - 6s 215ms/step - loss: 0.5112 - accuracy: 0.8978
Validation accuracy: 0.8978102207183838
```

4.8 Saving Trained Model as .h5 extension

```
cnn.save('trained_model.h5')
```

5. PERFORMANCE ANALYSIS

5.1 Accuracy Visualization of Training & Validation Model

5.1.1 Training Accuracy Visualization using Matplotlib

Figure 5.1 Training Accuracy Visualization

Definition: Training accuracy is a metric that measures the correctness of a machine learning model's predictions on the training dataset. It is calculated by comparing the model's predictions to the actual labels in the training set. A high training accuracy indicates that the model is effectively learning and memorizing patterns present in the training data.

Epochs : An epoch refers to one complete pass through the entire training dataset. Multiple epochs are used in the training process to allow the model to adjust its internal parameters iteratively. As the model goes through each epoch, it refines its ability to make predictions on the training data.

The x-axis represents the number of training epochs. An epoch is one complete pass through.

The y-axis represents the training accuracy of the model.

Visualizing training accuracy helps practitioners make decisions about the training process. It aids in determining when to stop training to avoid overfitting, understanding the model's convergence, and adjusting hyperparameters to achieve the desired balance between training accuracy and generalization.

5.1.2 Validation Accuracy Visualization using Matplotlib

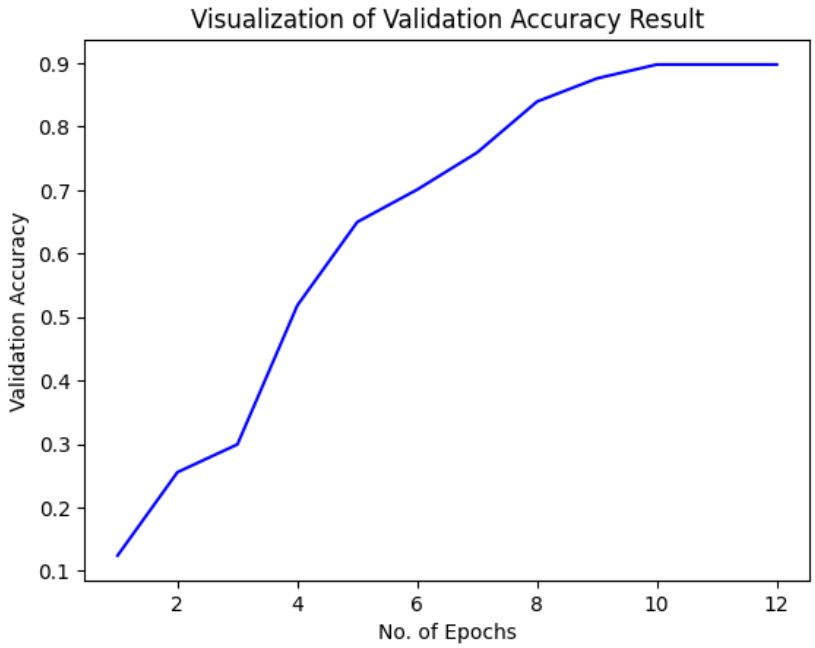

Figure 5.2 Validation Accuracy Visualization

Validation accuracy is a metric used to evaluate the performance of a machine learning model on a separate dataset that it has not seen during training. This dataset, known as the validation set, serves as a proxy for real-world, unseen data. The accuracy is calculated by comparing the model's predictions to the actual labels in the validation set.

Interpreting the Plot:

- **Increasing Validation Accuracy:** A consistent upward trend in validation accuracy indicates that the model is learning useful patterns in the data and generalizing well.
- **Fluctuations:** Occasional fluctuations may be normal, especially in complex datasets. However, persistent oscillations might suggest that the model is struggling to find stable patterns.
- **Plateau or Decrease:** A plateau or a decrease in validation accuracy may signal overfitting, where the model is fitting the training data too closely and failing to generalize.

The x-axis represents the number of training epochs. An epoch is one complete pass through.

The y-axis represents the validation accuracy of the model.

6. ANDROID APPLICATION DEVELOPMENT

6.1 Goal:

Early Disease Detection: The primary goal of the application is to enable early detection of diseases in pomegranate crops. This can significantly reduce the impact of diseases on crop yield and quality.

User-Friendly Interface: Designing a user-friendly interface that allows farmers, even those with limited technical knowledge, to easily navigate and use the application. This includes simple image capture and analysis functionalities.

Accurate Diagnosis: Implementing a robust CNN model that can accurately diagnose various diseases affecting pomegranate crops. The model should be trained on a diverse dataset to ensure its effectiveness across different regions and conditions.

Real-Time Notifications: Providing real-time notifications to farmers regarding the status of their crops, including alerts for potential disease outbreaks. This feature enhances the proactive management of crops.

Scope of the Thesis: The scope of this thesis extends to the development and evaluation of a mobile application for pomegranate crop disease detection using CNN.

6.2 Key components of the thesis include:

Pomegranate crop images, including healthy plants and those affected by various diseases.

Model Development: Designing and implementing a Convolutional Neural Network for image-based disease detection in pomegranate crops. The model should be trained and fine-tuned to achieve high accuracy.

Mobile Application Development: Creating an intuitive mobile application that integrates the trained CNN model for on-the-go disease detection. The application should be compatible with Android platforms.

Performance Evaluation: Conducting thorough evaluations to assess the accuracy, sensitivity, and specificity of the developed model. Comparisons with existing methods or models can also be considered.

User Feedback and Usability Testing: Gathering feedback from potential users, especially farmers, to assess the usability and practicality of the application in real-world scenarios.

6.3 Theory:

This project presents an Android application designed for pomegranate farmers and growers to accurately detect pomegranate diseases using a built-in machine learning model. Developed in Java language, the application leverages the accessibility and convenience of mobile devices to empower users with a valuable tool for early disease identification and prevention.

6.4 Android Platform Overview

Android is a popular mobile operating system used on a wide range of smartphones and tablets. It provides a robust and versatile platform for developing applications with diverse functionalities. Its open-source nature allows for custom adaptations and extensive resource availability for developers.

6.4.1 Application Structure

The application consists of several key files, each containing specific functionalities:

6.4.2 MainActivity.java:

This primary file defines the central activity displayed on the user's screen. It manages user interaction and interfaces with other components of the application.

onCreate(): Initializes the user interface with buttons, information fields, and displays.

classifyImage(): Processes captured images, prepares them for the model, and sends them for disease prediction.

onActivityResult(): Manages outcomes after taking a picture, including showing the captured image and triggering disease classification.

6.4.2 activity_main.xml:

This file defines the layout for the application's user interface. It specifies the arrangement and visual elements displayed on the screen.

6.4.3 AboutActivity.java and about.xml:

These files handle the information section of the application, showcasing details about the project, its functionalities, and relevant details for users.

6.4.4 model.tflite and Model.java:

These files relate to the machine learning model responsible for disease.

model.tflite: contains the trained machine learning model in a compressed format.

Model.java: interacts with the model, prepares model inputs from captured images, runs model predictions, and retrieves outputs for disease identification.

Additional Files:

The application may include additional files for resources like images, icons, or other supporting functionalities depending on the specific implementation.

6.4.5 Main files Description:

MainActivity.java Description

The *MainActivity.java* file is the core component of the Pomegranate Disease Detection Android application. This class is responsible for managing the main user interface, capturing images from the device's camera, and initiating the image classification process using a pre-trained machine learning model.

User Interface Initialization

The *onCreate* method, executed when the activity is created, sets up the user interface elements such as buttons, text views, and the image display. The layout is defined in the *activity_main.xml* file, which includes an *ImageButton* for additional information, a Button for capturing images, and various *TextViews* for displaying classification results.

```
ImageButton infoButton = findViewById(R.id.infoButton);
result = findViewById(R.id.result);
confidence = findViewById(R.id.confidence);
imageView = findViewById(R.id.imageView);
picture = findViewById(R.id.button);
```

Camera Integration

The picture button's click listener initiates the camera capture process. If the application has the necessary camera permissions, it launches the camera intent to capture an image. The result is handled in the *onActivityResult* method.

```
picture.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Launch camera if we have permission
        if          (checkSelfPermission(Manifest.permission.CAMERA)          ==
PackageManager.PERMISSION_GRANTED) {
            Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            startActivityForResult(cameraIntent, 1);
        } else {
            // Request camera permission if we don't have it.
            requestPermissions(new String[]{Manifest.permission.CAMERA}, 100);
        }
    }
});
```

Image Classification

The *classifyImage* method is responsible for processing the captured image using a TensorFlow Lite machine learning model. It initializes the model, prepares the input image data, performs inference, and displays the classification results on the user interface.

```
public void classifyImage(Bitmap image) {
    try {
        // Initialize the TensorFlow Lite model
        Model model = Model.newInstance(getApplicationContext());

        // Prepare the input data for the model
        TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1,
224, 224, 3}, DataType.FLOAT32);
        ByteBuffer byteBuffer = ByteBuffer.allocateDirect(4 * imageSize *
imageSize * 3);
        // ... (image data preparation)

        // Run model inference
        Model.Outputs outputs = model.process(inputFeature0);
        TensorBuffer outputFeature0 =
outputs.getOutputFeature0AsTensorBuffer();

        // Process the output to determine the classification result
        // Display results on the UI
        // ... (result processing)

        // Release model resources
        model.close();
    } catch (IOException e) {
        // Handle exception
    }
}
```

Handling Camera Result

The *onActivityResult* method is triggered when the camera captures an image successfully. It retrieves the captured image, resizes it, and then calls the *classifyImage* method for inference.

```
@Override
public void onActivityResult(int requestCode, int resultCode, @Nullable Intent
data) {
    if (requestCode == 1 && resultCode == RESULT_OK) {
        // Process the captured image
        Bitmap image = (Bitmap) data.getExtras().get("data");
        // ... (image processing)

        // Perform image classification
        classifyImage(image);
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

6.5 UI Designing

The MainActivity.xml layout file defines the visual structure and user interface components for the main screen of the Pomegranate Disease Detection Android application. The design emphasizes a clean and intuitive user experience, incorporating various elements to facilitate image capture, classification, and result presentation.

Layout Structure

RelativeLayout:

Serves as the root layout for positioning child views relative to each other.

Enables flexible arrangement of UI components.

LinearLayout (Header):

Positioned at the top of the screen.

Vertical orientation for stacking child views.

Contains an ImageView displaying a pomegranate image as a decorative header.

Styled with a background color for visual appeal.

Button (Take Picture):

Positioned at the center of the screen, aligned to the bottom.

A prominent, styled button with bold text to initiate the image capture process.

Background tint provides a visually appealing color.

ImageView (Captured Image):

Centered horizontally in the layout.

Displays the captured image or a placeholder with a defined size and border.

Visibility set to "visible" by default.

TextViews (Classification Information):

Display classification-related information below the captured image.

"Classified as" text with bold styling and appropriate margins.

"Result" text showing the classification output with color-coded styling.

"Confidences" text indicating the confidence level of the classification.

ImageButton (Information Button):

Positioned at the top-right corner.

Provides additional information about the application.

Styled with a drawable logo for recognition.

Styling and Aesthetics

Color Scheme:

Utilizes a consistent color scheme, including a soft background color for the header and a distinct background tint for the "Take Picture" button.

Result text is styled with a specific color for easy differentiation.

Text Styling:

Uses bold styling for key textual elements, enhancing readability and emphasis.

\Appropriate text sizes ensure a balanced and visually pleasing layout.

Image Presentation:

The captured image is presented in a defined square shape with a border, contributing to a cohesive and organized appearance.

User Interaction

Button Interaction:

The "Take Picture" button serves as the primary interaction point for initiating the image capture process.

Information Button:

The information button provides users with additional details about the application, enhancing user engagement and understanding.

The MainActivity.xml layout demonstrates a thoughtful and visually appealing design, considering both functionality and aesthetics. The arrangement of elements and consistent styling contribute to a user-friendly interface, ensuring a positive user experience in the context of pomegranate disease detection.

Fig. 6.1 Application UI

The Application UI of “AnarMitra” determines the image of the fruits and classifies the fruit based on the trained ML model.

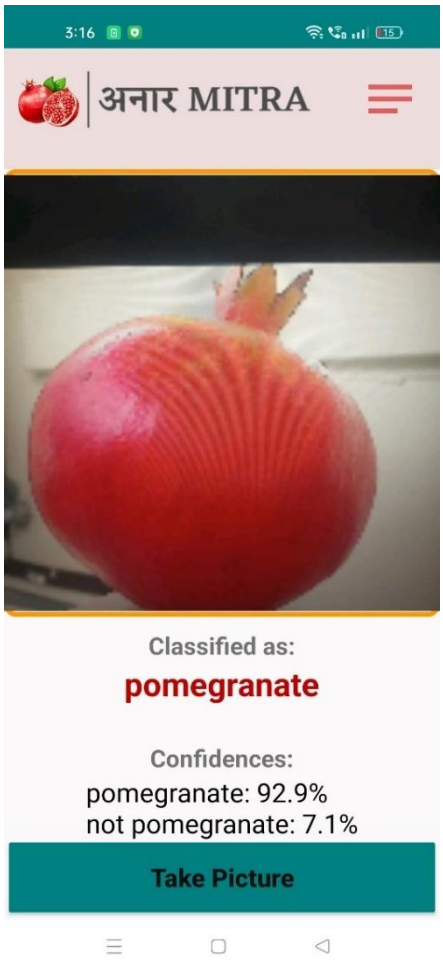

Fig 6.2 Image Classification Percentage Confidence

7. Conclusion

7.1 Conclusion

In the culmination of the ***"Machine Vision Based Pomegranate Ripeness Detection System"*** project, a pioneering exploration into the realms of agricultural technology has unfolded, with the primary objective of enhancing fruit quality assessment. This endeavour has seen the convergence of machine vision, deep learning, and agricultural science to create a holistic system that goes beyond mere detection, reaching into the nuanced assessment of pomegranate ripeness. The significance of this project extends beyond the technical intricacies, delving into the potential to revolutionize agricultural practices and improve yield quality.

Key Achievements are as follows

7.1.1 Precision in Ripeness Classification: The machine vision system has showcased remarkable precision in classifying pomegranates based on their ripeness, providing a valuable tool for farmers to optimize harvest timings.

7.1.2 Cross-disciplinary Integration: The successful integration of machine vision technology with agricultural science highlights the potential for cross-disciplinary collaborations, ushering in a new era of smart farming practices.

7.1.3 Real-time Monitoring Capability: The system's real-time monitoring capabilities empower farmers with instantaneous insights into the ripeness levels of pomegranates, facilitating timely decision-making in harvesting.

7.2 Future Scope:

The future scope of the "Machine Vision Based Pomegranate Ripeness Detection System" envisions a range of advancements and expansions that can further elevate its impact on agricultural practices and technology. Key areas of future exploration include:

7.2.1 Multi-Spectral Imaging: Integrate multi-spectral imaging technology to capture a broader spectrum of information about pomegranate fruits. This could enable the system to analyze additional features, such as internal quality, nutritional content, or early signs of diseases.

7.2.2 Automated Harvesting Systems: Collaborate with agricultural robotics and automation experts to seamlessly integrate the ripeness detection system with automated harvesting technologies. This could lead to the development of intelligent harvesting robots that selectively pick ripe fruits based on real-time assessments.

7.2.3 Disease Detection and Prevention: Expand the scope of the system to detect early signs of diseases or pests affecting pomegranate trees. By incorporating disease recognition models, the system could provide proactive recommendations for disease prevention and control measures.

7.2.4 Weather and Climate Integration: Incorporate real-time weather and climate data into the ripeness detection algorithm. This integration would allow the system to consider environmental factors that influence fruit ripening, providing more accurate predictions and recommendations to farmers.

7.2.5 Blockchain Traceability: Explore the integration of blockchain technology to establish a transparent and traceable supply chain. This could involve recording each stage of the fruit's journey, from ripeness detection to harvesting and distribution, ensuring quality control and authenticity.

7.3 Applications

The "Machine Vision Based Pomegranate Ripeness Detection System" holds immense potential for diverse applications across the agricultural landscape and beyond. Some notable applications include:

7.3.1 Precision Farming: The system can empower farmers with precise information about the ripeness of pomegranate fruits, enabling optimized harvesting schedules and resource allocation. This, in turn, contributes to increased yield quality and reduced waste.

7.3.2 Supply Chain Optimization: Facilitate supply chain management by providing accurate ripeness information. Distributors and retailers can better plan inventory, logistics, and marketing strategies based on the real-time ripeness status of available pomegranates.

7.3.3 Quality Assurance in Processing Units: Processing units and fruit packaging facilities can leverage the system to ensure the quality of incoming pomegranate batches. This application enhances quality control measures and supports the production of consistent and high-quality processed products.

7.3.4 Educational and Research Purposes: Serve as a valuable tool for agricultural research institutions and educational organizations. The system can aid researchers in studying fruit ripening patterns, disease dynamics, and the impact of environmental factors on crop quality.

7.3.5 Technological Innovation in Agriculture: Foster technological innovation within the agricultural sector. The project can stimulate the creation of new technologies and methodologies, establishing a precedent for the integration of machine vision and artificial intelligence in modern farming practices.

REFERENCES

[1] A. Dhandrave, V.T. Gaikwad “Implementation of Fruit Detection System and Checking Fruit Quality using Computer Vision,” IRJET, vol. 8, no. 6, pp. 4116-4120, June 2021.

[2] B. Rodrigues, R. Kansara, S. Singh, D. Save, S. Parihar “Ripe-Unripe: Machine Learning based Ripeness Classification” presented at the the 5th International Conference on Intelligent Computing and Control Systems (ICICCS 2021), Madurai, India, 2021.

[3] Z. Al-Mashhadani; B. Chandrasekaran, “Autonomous Ripeness Detection Using Image Processing for an Agricultural Robotic System*,” IEEE, pp.* 0743 - 0748, 2020

[4] R. Saragih, Andi W. R. Emanuel “Banana Ripeness Classification Based on Deep Learning using Convolutional Neural Network,” presented at the 3rd East Indonesia Conference on Computer and Information Technology (EIConCIT*), IEEE, Surabaya, Indonesia*, 2021,

[5] Fruit 360 Dataset : https://www.kaggle.com/datasets/moltean/fruits Version - 2020.05.18.0