Ripe-Sense: Mango Quality Grading

With Image Analysis and Deep Learning

By:
Lakshya Singh
Mayank M
Akshanth Sai K
Jahnavi Jain

Ripe-Sense: Mango Quality Grading with Image Analysis and Deep Learning

Mangoes are one of the most popular tropical fruits, consumed and exported worldwide. However, assessing mango quality and ripeness is a challenging task that requires expertise and experience. Traditional methods of grading mangoes are subjective, time-consuming, and prone to human error, leading to inconsistencies in the evaluation process. In recent years, advancements in computer vision and deep learning techniques have opened up new opportunities for automating fruit quality assessment.

"Ripe-Sense" is a deep learning project aimed at revolutionizing the mango quality grading process using image analysis and advanced deep learning algorithms. The objective is to develop a robust and accurate system that can automatically assess the quality and ripeness of mangoes based on their visual appearance.
The project leverages the power of deep learning models, such as convolutional neural networks (CNNs), to extract meaningful features from mango images. These models are trained on large datasets of annotated mango images, where each image is labeled with its corresponding quality grade and ripeness stage. By learning from these labeled examples, the deep learning models can generalize and make predictions on unseen mango images.

The benefits of Ripe-Sense are numerous. It provides a fast, objective, and consistent grading process, reducing human bias and errors. It also enables farmers to optimize their harvest and post-harvest operations, ensuring that mangoes are picked at the right stage of ripeness. For distributors and retailers, the system facilitates better inventory management and quality control. Finally, for consumers, Ripe-Sense ensures they receive mangoes that meet their preferences and expectations.

Aim:

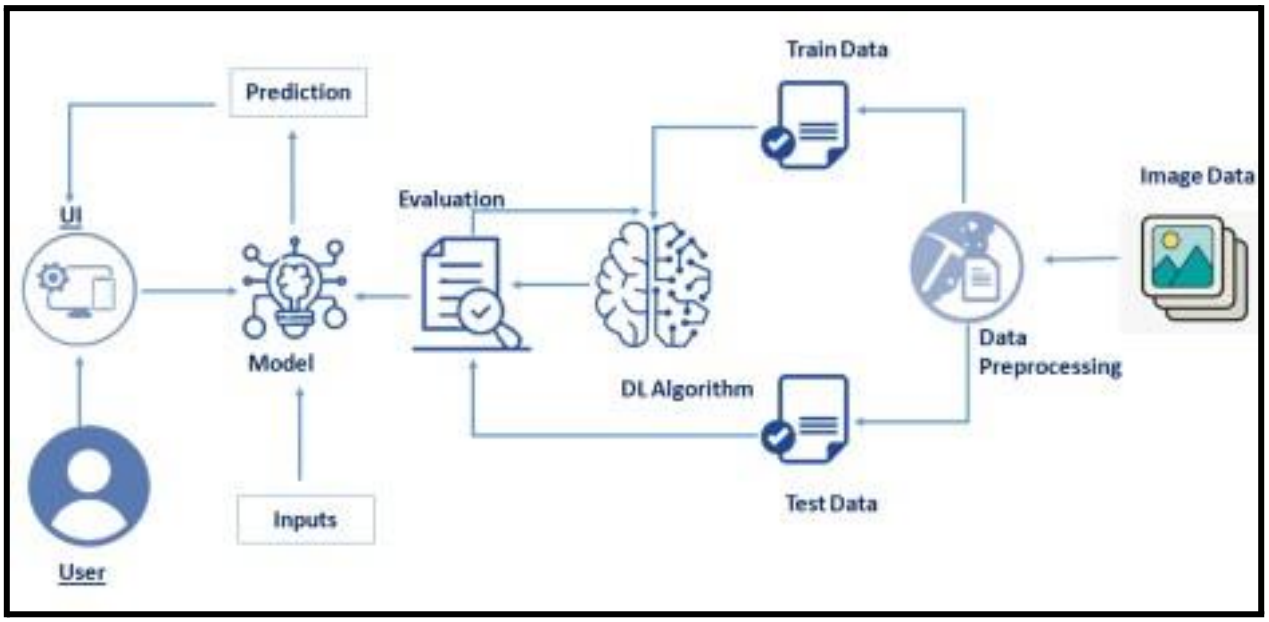
The aim of the project "Ripe-Sense: Mango Quality Grading with Image Analysis and Deep Learning" is to develop a deep learning model that can accurately classify mangoes into three categories: perfectly ripen, over ripen, and under ripen, based on their visual characteristics. The model aims to provide an automated and reliable solution for mango quality assessment, enabling farmers, distributors, and consumers to make informed decisions about mangoes' ripeness and suitability for consumption.

Purpose of doing this Project:

The purpose of the project "Ripe-Sense: Mango Quality Grading with Image Analysis and Deep Learning" is to provide an automated and accurate solution for mango quality grading. By training a deep learning model on three grading classifications (perfectly ripen, over ripen, and under ripen), the project aims to enable farmers, distributors, and consumers to quickly and objectively assess the ripeness of mangoes using just an image. Deploying the model using Flask enhances accessibility, allowing users to utilize the system through a user-friendly web interface, thereby improving efficiency and decision-making in the mango industry.

Technical Architecture:

The technical architecture of the Ripe-Sense project, titled "Mango Quality Grading with Image Analysis and Deep Learning," combines deep learning techniques and web development using Flask. The project focuses on training a deep learning model to classify mangoes into three grading categories: perfectly ripen, over ripen, and under ripen. The process begins with data collection and preprocessing, where mango images are gathered and standardized through resizing, cropping, and normalization. The collected dataset is then utilized to train a deep learning model, such as a convolutional neural network (CNN), which learns to extract relevant features from the mango images. Once the model is trained, it is deployed using Flask, a Python web framework. Flask enables the creation of a user-friendly web interface where users can upload images of mangoes. The deployed model processes the uploaded image and classifies it into one of the three grading categories. The resulting classification is displayed to the user in real-time, providing valuable insights into the quality of the mango. This technical architecture seamlessly integrates deep learning, image analysis, and Flask deployment to create an efficient and accessible system for mango quality grading.



Pre-requisites:

To complete this project, you must require following software's, concepts, and packages

- **Anaconda Navigator:**

  - Refer to the link below to download anaconda navigator - Link:

    https://www.youtube.com/watch?v=5mDYijMfSzs

- Python packages:

  - Open anaconda prompt as administrator

  - Type "pip install tensorflow" (make sure you are working on Python 64 bit)

  - Type "pip install flask".

- **Deep Learning Concepts**

  - CNN: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add

  - Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project we will:

- Know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Knowhow to pre-process/clean the data using different data preprocessing techniques.
- Know how to build a web application using Flask framework.

Project Flow:

- User interacts with the UI (User Interface) to upload the image as input
- Uploaded image is analyzed by the model which is integrated
- Once model analyses the uploaded image, the prediction food recipe is showcased on the UI to accomplish this, we have to complete all the activities and tasks listed below

o Data Collection. o Collect the dataset or Create the dataset
o Data Preprocessing.
o Import the ImageDataGenerator library o Configure ImageDataGenerator class
o Apply ImageDataGenerator functionality to Trainset and Testset
o Model Building o Import the model building Libraries
o Initializing the model
o Adding Input Layer

- o Adding Hidden Layer
- o Adding Output Layer
- o Configure the Learning Process
- o Training the model
- o Save the Model
- o Test the Model

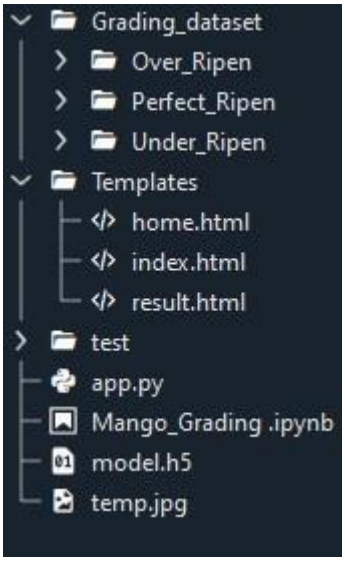
o Application Building

o Create an HTML file

o Build Python Code

Project Structure:

Create project folder which contains files as shown below:



- The data obtained 3 folders containing 200 images of respective type of graded mango each, which is used one for training, testing, validation dataset.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- app.py file is used for routing purposes using scripting.
- model.h5 is the saved model. This will further be used in the Flask integration.

Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

The business problem addressed by the "Ripe-Sense: Mango Quality Grading with Image Analysis and Deep Learning" project is the subjective and time-consuming nature of mango quality assessment. Traditional methods of grading mangoes rely on human expertise and are prone to errors and inconsistencies. This leads to challenges for farmers, distributors, and consumers in determining the ripeness and quality of mangoes. The project aims to automate and standardize the mango grading process by developing a deep learning model that can accurately classify mangoes into three categories: perfectly ripen, over ripen, and under ripen. By providing an objective and efficient solution, Ripe-Sense enables stakeholders in the mango industry to make informed decisions about the quality of mangoes, optimize operations, and enhance the overall mango consumption experience.

Activity 2: Business Requirements

1. **Accurate Mango Quality Grading:** The deep learning model should be trained to classify mangoes into three categories - perfectly ripen, over ripen, and under ripen - with a high degree of accuracy. The model should be able to assess mango quality based on visual characteristics captured in the input image.
2. **Real-Time Classification:** The model should provide real-time classification results to ensure quick and efficient grading. The system should process the uploaded mango image and provide the grading classification promptly, allowing stakeholders to make immediate decisions based on the quality assessment.
3. **Compatibility:** The deployed system should be compatible with different web browsers and devices, ensuring accessibility for a wide range of users.
4. **Reliability:** The system should be robust and reliable, ensuring consistent and accurate generation of wood textures, with minimal errors or inconsistencies.
5. **Trust:** The model should be designed in such a way that it develops trust among the users, especially the advertisers and content creators, who rely on the predicted adviews for their marketing strategies.
6. **User-Friendly Interface:** The deployment using Flask should provide a user-friendly web interface where stakeholders, such as farmers, distributors, and consumers, can easily interact with the system. The interface should allow users to upload mango images, receive classification results, and view the grading category assigned to each mango.
7. **Seamless Deployment:** The deployment using Flask should be smooth and seamless, ensuring that the model is easily accessible and operational. The system should handle multiple requests concurrently and provide consistent and reliable performance.
8. **Scalability:** The solution should be scalable to handle a growing number of mango images and users. As the usage of the system increases, it should be able to accommodate the additional load without compromising on performance or accuracy.
9. **Flexibility for Future Enhancements:** The system should be designed with flexibility in mind, allowing for future enhancements and improvements. This includes the ability to incorporate additional grading categories or expand the dataset to further refine the mango quality assessment capabilities of the model.

Activity 3: Literature Survey

The literature survey conducted for the Ripe-Sense project focused on existing research and studies related to image analysis, deep learning, and fruit quality grading. Several papers and articles were reviewed to understand the methodologies, techniques, and algorithms employed in similar projects. The survey covered topics such as convolutional neural networks (CNNs), image preprocessing, feature extraction, and classification models. Key findings from the literature survey guided the selection of appropriate models and techniques for mango quality grading, ensuring the project's alignment with current advancements in the field.

Activity 4: Social or Business Impact.

Social Impact:

The Ripe-Sense project has significant social impact as it addresses the challenges faced by stakeholders in the mango industry. By providing an automated and accurate mango quality grading system, it empowers farmers, distributors, and consumers to make informed decisions about mangoes' ripeness and suitability for consumption. This helps reduce food waste by ensuring optimal utilization of mangoes at

different stages of ripeness. Additionally, the project contributes to improved market transparency and consumer satisfaction, enhancing overall confidence and trust in the mango supply chain.

Business Impact:

The Ripe-Sense project brings significant business impact to the mango industry. By automating the mango quality grading process, it enables stakeholders to streamline operations, reduce manual efforts, and enhance efficiency. Farmers can optimize harvesting and sorting, distributors can make informed decisions on stock management, and consumers can confidently select mangoes based on their desired ripeness. This leads to improved supply chain management, reduced losses, enhanced customer satisfaction, and ultimately, increased profitability for businesses involved in the mango trade.

Milestone 2: Data Collection

Machine Learning & Deep Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset. **Activity 1: Collect the dataset**

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/saurabhshahane/mango-varieties-classification

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Milestone 3: Image Preprocessing

Activity 1: Import the ImageDataGenerator library

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from keras

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

Activity 2: Configure ImageDataGenerator class

The ImageDataGenerator class in Keras is used for real-time data augmentation during model training. It allows the user to perform various image transformations such as rotation, zooming, shifting, and flipping on the input data, which can help prevent overfitting and improve model performance. The class can be configured to apply different types and levels of data augmentation, and it supports both binary and categorical data. The ImageDataGenerator class is a powerful tool for improving the accuracy and robustness of deep learning models, particularly in computer vision applications.

Activity 3 :Apply ImageDataGenerator functionality to Trainset and Testset

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split=0.2)
```

For Training dataset

The code defines an ImageDataGenerator object for data augmentation. It applies various transformations to the training images, including rescaling, rotation, shifting, shearing, zooming, and horizontal flipping. The validation_split parameter specifies the portion of the data to be used for validation. The train_generator is then created using the flow_from_directory() function, specifying the directory path, target size, batch size, class mode as categorical, and subset as 'training'. This generator will generate batches of augmented training data on the fly for training the model.

For Validation Dataset

Here, we have created a validation_generator using the same ImageDataGenerator object. It generates batches of validation data from the specified directory path. The target_size, batch_size, class_mode, and subset parameters are set similarly to the train_generator. This generator will provide batches of validation data for evaluating the model's performance during training.

```
train_generator = train_datagen.flow_from_directory(
        data_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='categorical',
        subset='training')

validation_generator = train_datagen.flow_from_directory(
        data_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='categorical',
        subset='validation')

test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
        data_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='categorical')
```

For Test Dataset

In the given code, a test_generator using the ImageDataGenerator object is created. It rescales the pixel values of the images by dividing them by 255. The generator generates batches of test data from the specified directory path. The target_size, batch_size, and class_mode parameters are set to match the requirements of the model. This generator will provide batches of test data for evaluating the final trained model's performance.

```
Found 480 images belonging to 3 classes.
Found 120 images belonging to 3 classes.
Found 600 images belonging to 3 classes.
```

These are the outputs from the flow_from_directory() method of ImageDataGenerator. The first line shows that there are 480 images in the training set, the second line shows that there are 120 images in the validation set, and the third line shows that there are 600 images in the test set. All sets have 3 classes in total.

Activity 3 : Sample Image Visualization

```
import matplotlib.pyplot as plt

def show_image_samples(generator):
    # Get the class indices from the generator
    class_indices = generator.class_indices

    # Get the class names from the class indices
    class_names = list(class_indices.keys())

    # Get a sample batch from the generator
    images, labels = next(generator)

    # Plot the images and their labels
    plt.figure(figsize=(10, 10))
    for i in range(min(images.shape[0], 25)):
        plt.subplot(5, 5, i + 1)
        plt.imshow(images[i])
        plt.title(class_names[np.argmax(labels[i])])
        plt.axis('off')
    plt.show()
```

```
show_image_samples(train_generator)
```

The show_image_samples function takes a generator as input and displays a grid of sample images along with their corresponding labels. It retrieves the class indices and names from the generator, extracts a batch of images and labels, and then plots them using Matplotlib. Each image is displayed with its associated class name, and the resulting grid is shown with a size of 5x5. This function helps in visually inspecting the data and verifying the correctness of the generator.

Milestone 4: Model Building

Activity 1 : Importing the Model Building Libraries

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import pathlib

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.python.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
```

Activity 2: Building a EfficientNet-based Transfer Learning Model Architecture

EfficientNet

```
[26] import os
     import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.keras import regularizers, layers, Model
     from tensorflow.keras.applications import EfficientNetB2
     from tensorflow.keras.optimizers import Adamax
     from keras.callbacks import ReduceLROnPlateau, EarlyStopping


     sdir = '/content/drive/MyDrive/Grading_dataset'
     classlist = os.listdir(sdir)
     filepaths = []
     labels = []
     for c in classlist:
         classpath = os.path.join(sdir, c)
         for f in os.listdir(classpath):
             filepaths.append(os.path.join(classpath, f))
             labels.append(c)

     df = pd.DataFrame({'filepaths': filepaths, 'labels': labels})
```

This code block creates a model for mango quality grading using transfer learning with the EfficientNet architecture. The pre-trained EfficientNet model is loaded with its weights from the ImageNet dataset. The model is then constructed by adding each layer from the EfficientNet model to a new Sequential

model. The added layers are frozen, and additional layers are appended, including a flatten layer, dense layers with ReLU activation, batch normalization, dropout regularization, and a final dense layer with a softmax activation for classification into three mango ripeness categories.

Activity 3: Model Compilation and Training

```
def scalar(img): return img

trgen = ImageDataGenerator(preprocessing_function=scalar, horizontal_flip=True)
tvgen = ImageDataGenerator(preprocessing_function=scalar)
train_gen = trgen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                      class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=batch_size)
test_gen = tvgen.flow_from_dataframe(test_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                     class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=test_batch_size)
valid_gen = tvgen.flow_from_dataframe(valid_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                      class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=batch_size)
classes = list(train_gen.class_indices.keys())
class_count = len(classes)
train_steps = np.ceil(len(train_gen.labels) / batch_size)

base_model = EfficientNetB2(include_top=False, weights="imagenet", input_shape=(height, width, channels), pooling='max')
x = layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(base_model.output)
x = layers.Dense(256, kernel_regularizer=regularizers.l2(0.016), activity_regularizer=regularizers.l1(0.006),
                 bias_regularizer=regularizers.l1(0.006), activation='relu')(x)
x = layers.Dropout(rate=0.45, seed=123)(x)
output = layers.Dense(class_count, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output)
model.compile(optimizer=Adamax(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
Found 480 validated image filenames belonging to 3 classes.
Found 60 validated image filenames belonging to 3 classes.
Found 60 validated image filenames belonging to 3 classes.
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb2_notop.h5
31790344/31790344 [==============================] - 1s 0us/step
```

This code the model is compiled using the Adam optimizer, categorical cross-entropy loss function, and accuracy metric. The model is then trained using the fit() function, specifying the training generator, number of epochs (100 in this case), validation data generator, and verbose mode set to 1 for displaying training progress. The training process updates the model's weights and biases iteratively over the specified number of epochs, evaluating the performance on the validation data during training.

Milestone

```
epochs = 100
callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=1, verbose=1),
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True, verbose=1)
]
history = model.fit(train_gen, epochs=epochs, verbose=1, callbacks=callbacks, validation_data=valid_gen,
                    validation_steps=None, shuffle=False, initial_epoch=0)
```

```
Epoch 60/100
16/16 [==============================] - 7s 408ms/step - loss: 0.3186 - accuracy: 1.0000 - val_loss: 0.3338 - val_accuracy: 0.9833 - lr: 5.0000e-04
Epoch 61/100
16/16 [==============================] - 7s 399ms/step - loss: 0.3100 - accuracy: 0.9979 - val_loss: 0.3253 - val_accuracy: 0.9833 - lr: 5.0000e-04
Epoch 62/100
16/16 [==============================] - 7s 414ms/step - loss: 0.3029 - accuracy: 0.9979 - val_loss: 0.3210 - val_accuracy: 0.9833 - lr: 5.0000e-04
Epoch 63/100
16/16 [==============================] - 7s 401ms/step - loss: 0.3033 - accuracy: 0.9958 - val_loss: 0.3207 - val_accuracy: 0.9833 - lr: 5.0000e-04
Epoch 64/100
16/16 [==============================] - 7s 404ms/step - loss: 0.2955 - accuracy: 0.9979 - val_loss: 0.3030 - val_accuracy: 0.9833 - lr: 5.0000e-04
Epoch 65/100
16/16 [==============================] - ETA: 0s - loss: 0.2900 - accuracy: 0.9979
Epoch 65: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
16/16 [==============================] - 7s 400ms/step - loss: 0.2900 - accuracy: 0.9979 - val_loss: 0.3111 - val_accuracy: 0.9833 - lr: 5.0000e-04
Epoch 66/100
16/16 [==============================] - 7s 400ms/step - loss: 0.2832 - accuracy: 0.9958 - val_loss: 0.3028 - val_accuracy: 0.9833 - lr: 2.5000e-04
Epoch 67/100
16/16 [==============================] - 7s 415ms/step - loss: 0.2756 - accuracy: 1.0000 - val_loss: 0.2982 - val_accuracy: 0.9833 - lr: 2.5000e-04
Epoch 68/100
16/16 [==============================] - 6s 398ms/step - loss: 0.2781 - accuracy: 0.9979 - val_loss: 0.2956 - val_accuracy: 0.9833 - lr: 2.5000e-04
Epoch 69/100
16/16 [==============================] - ETA: 0s - loss: 0.2723 - accuracy: 1.0000
Epoch 69: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
16/16 [==============================] - 6s 391ms/step - loss: 0.2723 - accuracy: 1.0000 - val_loss: 0.2997 - val_accuracy: 0.9833 - lr: 2.5000e-04
Epoch 70/100
16/16 [==============================] - 7s 414ms/step - loss: 0.2661 - accuracy: 1.0000 - val_loss: 0.2942 - val_accuracy: 0.9833 - lr: 1.2500e-04
Epoch 71/100
16/16 [==============================] - 7s 400ms/step - loss: 0.2763 - accuracy: 0.9958 - val_loss: 0.2935 - val_accuracy: 0.9833 - lr: 1.2500e-04
Epoch 72/100
16/16 [==============================] - 7s 412ms/step - loss: 0.2734 - accuracy: 0.9979 - val_loss: 0.2931 - val_accuracy: 0.9833 - lr: 1.2500e-04
Epoch 73/100
16/16 [==============================] - 7s 398ms/step - loss: 0.2683 - accuracy: 1.0000 - val_loss: 0.2894 - val_accuracy: 0.9833 - lr: 1.2500e-04
Epoch 74/100
16/16 [==============================] - 7s 399ms/step - loss: 0.2632 - accuracy: 1.0000 - val_loss: 0.2833 - val_accuracy: 0.9833 - lr: 1.2500e-04
Epoch 75/100
16/16 [==============================] - ETA: 0s - loss: 0.2637 - accuracy: 0.9979
Epoch 75: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
16/16 [==============================] - 7s 413ms/step - loss: 0.2637 - accuracy: 0.9979 - val_loss: 0.2833 - val_accuracy: 0.9833 - lr: 1.2500e-04
Epoch 76/100
16/16 [==============================] - 6s 398ms/step - loss: 0.2707 - accuracy: 0.9937 - val_loss: 0.2810 - val_accuracy: 0.9833 - lr: 6.2500e-05
Epoch 77/100
16/16 [==============================] - ETA: 0s - loss: 0.2708 - accuracy: 0.9958
Epoch 77: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
16/16 [==============================] - 7s 405ms/step - loss: 0.2708 - accuracy: 0.9958 - val_loss: 0.2810 - val_accuracy: 0.9833 - lr: 6.2500e-05
Epoch 78/100
16/16 [==============================] - ETA: 0s - loss: 0.2658 - accuracy: 0.9979
Epoch 78: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
16/16 [==============================] - 7s 400ms/step - loss: 0.2658 - accuracy: 0.9979 - val_loss: 0.2811 - val_accuracy: 0.9833 - lr: 3.1250e-05
Epoch 79/100
16/16 [==============================] - ETA: 0s - loss: 0.2603 - accuracy: 1.0000
Epoch 79: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.
16/16 [==============================] - 7s 403ms/step - loss: 0.2603 - accuracy: 1.0000 - val_loss: 0.2812 - val_accuracy: 0.9833 - lr: 1.5625e-05
Epoch 80/100
16/16 [==============================] - ETA: 0s - loss: 0.2594 - accuracy: 1.0000
Epoch 80: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.
Restoring model weights from the end of the best epoch: 77.
16/16 [==============================] - 7s 412ms/step - loss: 0.2594 - accuracy: 1.0000 - val_loss: 0.2815 - val_accuracy: 0.9833 - lr: 7.8125e-06
Epoch 80: early stopping
```

5: Visualization of Training and Validation Performance.

```
import matplotlib.pyplot as plt

# plot accuracy for both Validation and Training Datasets
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

```python
def show_image_samples(generator):
    # Get the class indices from the generator
    class_indices = generator.class_indices

    # Get the class names from the class indices
    class_names = list(class_indices.keys())

    # Get a sample batch from the generator
    images, labels = next(generator)

    # Plot the images and their labels
    plt.figure(figsize=(10, 10))
    for i in range(min(images.shape[0], 25)):
        plt.subplot(5, 5, i + 1)
        plt.imshow(images[i])
        plt.title(class_names[np.argmax(labels[i])])
        plt.axis('off')
    plt.show()
```

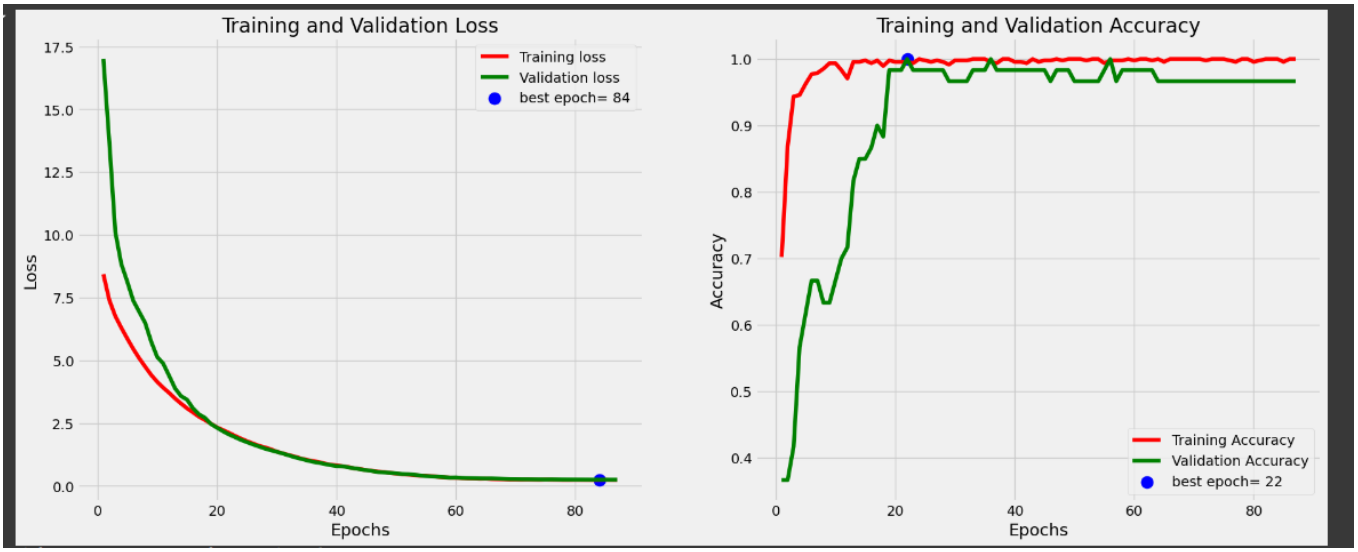
```python
from collections import Counter

# Check the class distribution
class_counts = train_generator.class_indices
print(f"Class indices: {class_counts}")

# Check class counts
train_labels = train_generator.classes
print(Counter(train_labels))
```

```
Class indices: {'Over_Ripen': 0, 'Perfect_Ripen': 1, 'Under_Ripen': 2}
Counter({0: 160, 1: 160, 2: 160})
```

The code block is used to plot the accuracy and loss values over epochs for both the training and validation datasets. The first plot shows the accuracy values, while the second plot shows the loss values. The x-axis represents the number of epochs, and the y-axis represents the accuracy or loss values. The plot helps visualize the model's performance during training, allowing us to analyze trends, identify overfitting or underfitting, and determine the optimal number of epochs for training the model.



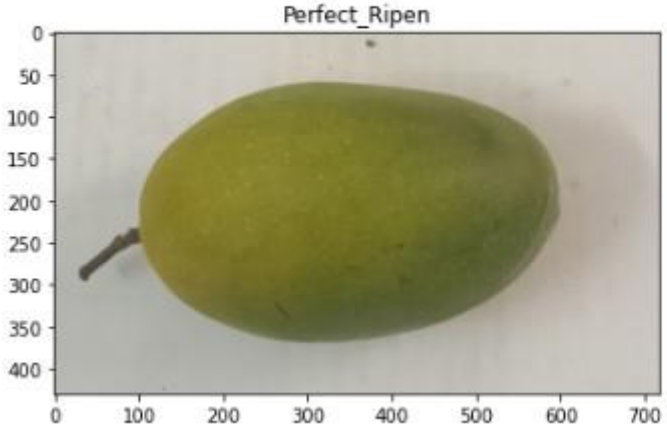
Milestone

```
tr_plot(history,0)
subject='mangos'
acc=model.evaluate( test_gen, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
msg=f'accuracy on the test set is {acc:5.2f} %'
model.save('eff_model.h5')
```

Activity 2: Predicting mango variety from an uploaded image

In this code, an image is loaded using the load_img function from TensorFlow's keras.preprocessing.image module. The image is then converted into a numpy array using img_to_array. The preprocessing steps, such as scaling the pixel values, are applied to the image array to match the preprocessing done during training. The array is reshaped to match the input shape expected by the model. The model then makes a prediction on the preprocessed image by calling model.predict. The predicted class is obtained by finding the index of the maximum value in the prediction array. Finally, the predicted class and the test image are displayed using matplotlib.

```
1/1 [==============================] - 0s 314ms/step
Predicted class: Perfect_Ripen
```



6 : Performance Testing

Activity 1: Evaluating model on validation set

In this code the model's performance is evaluated on a validation dataset. The loss and accuracy metrics are calculated using the evaluate function, taking the validation_generator as input. The resulting accuracy is then printed with a formatted string, displaying the validation accuracy in percentage format with two decimal places.

```
loss, accuracy = model.evaluate(test_gen)
print('Validation Accuracy: {:.2f}%'.format(accuracy*100))
```

```
1/1 [==============================] - 3s 3s/step - loss: 0.3210 - accuracy: 0.9667
Validation Accuracy: 96.67%
```

The result indicates that the model achieved an accuracy of 96.67% on the validation dataset. The loss value of 0.3210 suggests that the model's predictions were reasonably accurate. These metrics indicate that the model is performing well in classifying mangoes into their respective ripeness categories.

Activity 2: Evaluating model on test set

In this code the model's performance is evaluated on a test dataset. The loss and accuracy metrics are calculated using the evaluate function, taking the test_generator as input. The resulting accuracy is then printed with a formatted string, displaying the test accuracy in percentage format with two decimal places.

```
loss, accuracy = model.evaluate(valid_gen)
print('Test Accuracy: {:.2f}%'.format(accuracy*100))
```

```
2/2 [==============================] - 0s 198ms/step - loss: 0.2810 - accuracy: 0.9833
Test Accuracy: 98.33%
```

The result shows that the model achieved a test accuracy of 98.33%. With a loss value of 0.2810, the model's predictions were highly accurate. This indicates that the model has effectively learned to classify mangoes into their respective ripeness categories, demonstrating its strong performance on the test dataset.

Milestone 6: Model Deployment

Activity 1: Save and load the best model

```
acc=model.evaluate( test_gen, batch_size=test_batch_size, verbose=1, steps=test_steps
msg=f'accuracy on the test set is {acc:5.2f} %'
model.save('/content/drive/MyDrive/eff_model.h5')
```

```
1/1 [==============================] - 3s 3s/step - loss: 0.3210 - accuracy: 0.9667
```

We save the model into a file named mango_classification_model.h5

Milestone 7: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building HTML pages:

For this project we create two HTML files namely

• Index.html

Milestone

• Results.html

• Home.html

And we will save them in the templates folder.

Activity 2.2: Build Python code

Create a new app.py file which will be store in the Flask folder.

- Import the necessary Libraries.

```
from flask import Flask, request, jsonify, render_template
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import base64
```

- This code loads a pre-trained machine learning model for mango classification, which has been saved in the file 'model.h5'.

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

- This code creates a new instance of a Flask web application using the Flask class from the Flask library. The __name__ argument specifies the name of the application's module or package.

```
app = Flask(__name__)
```

- The home function is a route handler in Flask that is associated with the default URL route '/'. When a user accesses the root URL of the web application, Flask invokes this function. Inside the function, it calls the render_template function to render and display the 'index.html' template, which represents the home page of the web application. This route handler is responsible for handling requests to the home page and returning the corresponding HTML content to the user's browser.
- **Flask Route for Prediction**

The predict function is a route handler in Flask that is associated with the '/predict' URL route and accepts POST requests. When a user submits an image file through a form, Flask invokes this function. Inside the function, it retrieves the image file from the request, saves it temporarily, and preprocesses the image using the preprocess_image function. Then, it makes a prediction using the trained model. Based on the prediction result, it converts the predicted class to a human-readable string. The image is then loaded and converted to a base64-encoded string for rendering in the HTML template. Finally, it renders the 'result.html' template with the prediction and the image to display the prediction result to the user.

```python
@app.route('/predict', methods=['POST'])
def predict():
    # Get the image file from the request
    file = request.files['file']

    # Save the file to a temporary location
    file_path = 'temp.jpg'
    file.save(file_path)

    # Preprocess the image
    img_array = preprocess_image(file_path)

    # Make a prediction
    prediction = model.predict(img_array)

    # Convert the prediction to a string
    if np.argmax(prediction) == 0:
        prediction_str = 'Over-Riped'
    elif np.argmax(prediction) == 1:
        prediction_str = 'Perfectly Riped'
    else:
        prediction_str = 'Under Riped'

    # Load the image and convert it to a base64-encoded string
    with open(file_path, 'rb') as f:
        img_base64 = base64.b64encode(f.read()).decode()

    # Render the HTML template with the prediction and the image
    return render_template('result.html', prediction=prediction_str, image=img_base64)
```

Main Function:

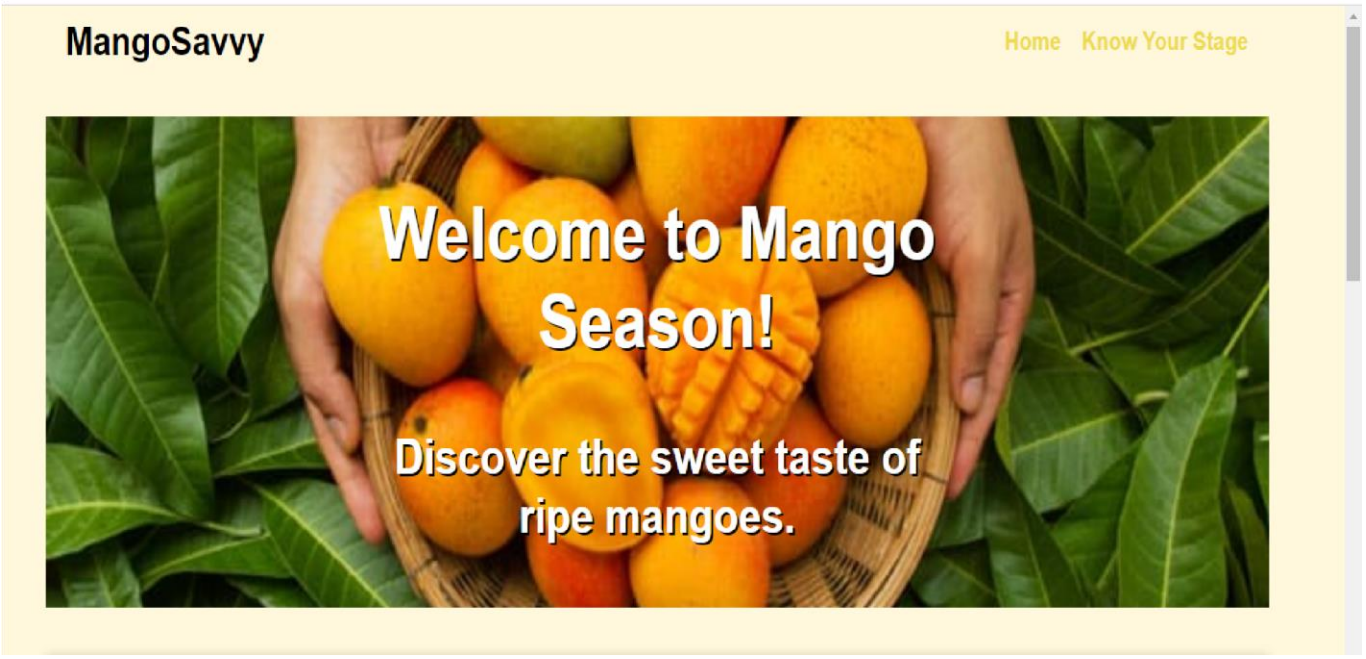
This code runs the Flask application if the script is being executed directly (i.e. not imported as a module in another script). The if __name__ == '__main__': line checks if the script is the main module being executed, and if so, runs the Flask application using the app.run() method. This method starts the Flask development server, allowing the application to be accessed via a web browser at the appropriate URL.

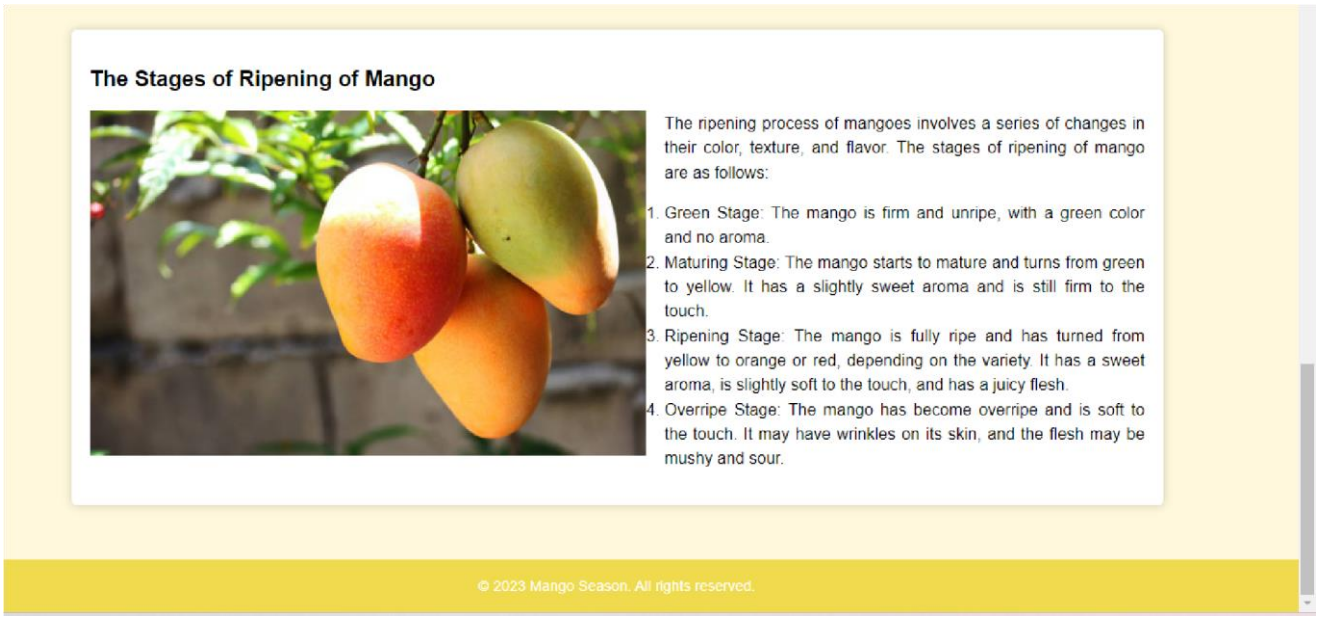
```python
if __name__ == '__main__':
    app.run()
```

Activity 2.3: Run the Web Application

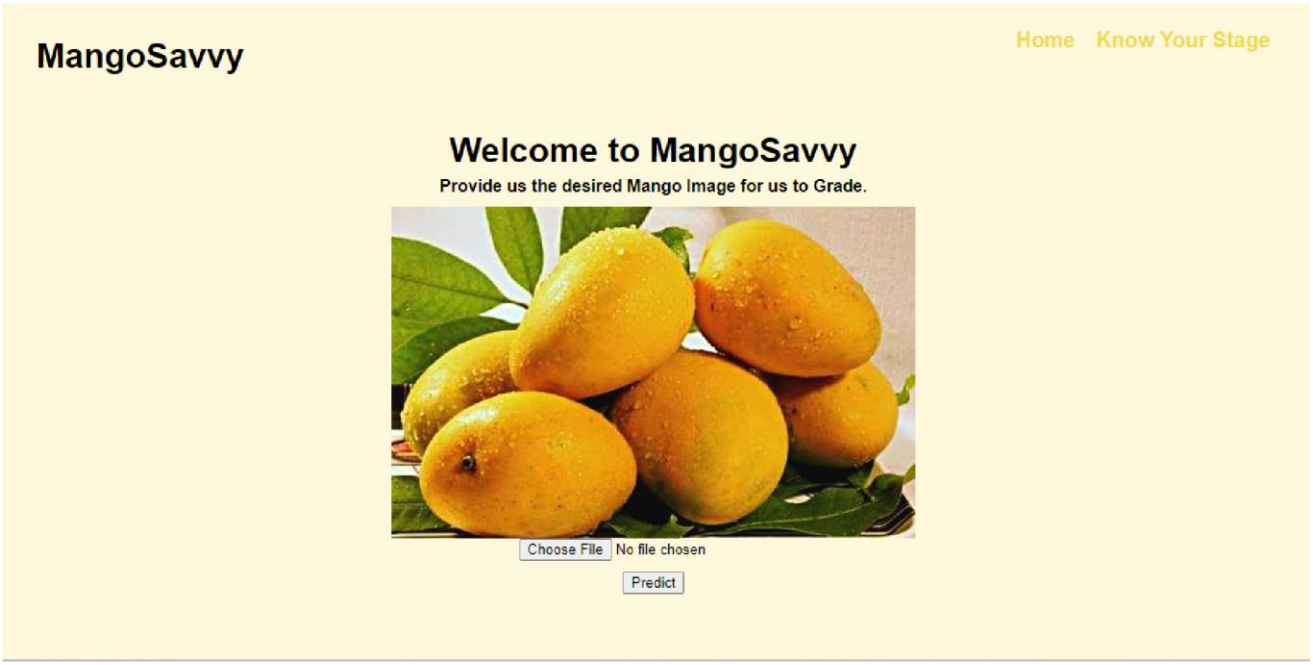
When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it in the browser.

When we paste the URL in a web browser, our home.html page will open. It contains a welcome page and information about - the ‘Importance of Defining the Ripening Stage of Mango’& ‘The Stages of Ripening of Mango’. There are two navigation buttons on the top right- ‘Home’, and ‘Know Your Stage ’.

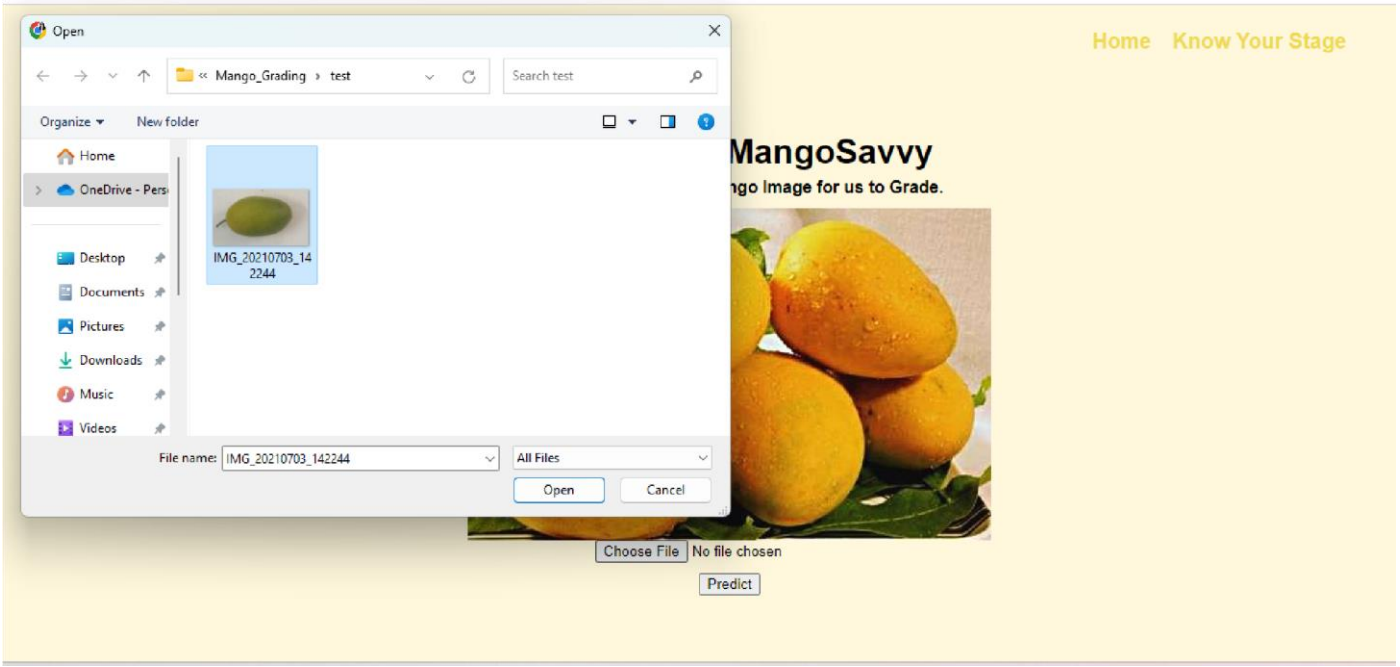




When you click on Know Your Stage button you will be redirected to index.html page where you'll be asked to give the image of mango to graded as input.



Choose the desired input image-



After hitting the predict button, you will be redirected to result.html page, where you will find information about the precautionary measures to be taken while grading the mangoes and also information about the Deep Learning model that has been used.

MangoSavvy

Home Know Your Stage

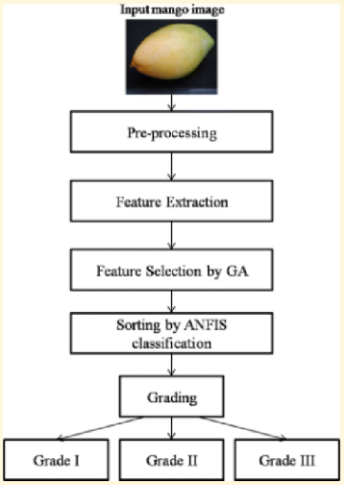
Welcome to MangoSavvy

Scroll down to see the results

Precautionary measures to be taken while Grading Mangos

When grading mangos, it is important to follow certain precautionary measures to ensure accuracy and safety. These measures include:

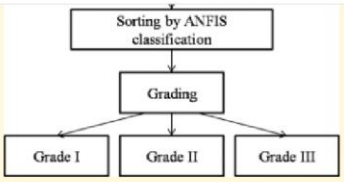
- Wearing protective gloves to avoid contamination
- Using clean and sanitized grading equipment
- Inspecting each mango carefully for defects
- Properly handling and storing graded mangos



Steps taken while grading a mango using Deep Learning

The grading of mangos using deep learning involves the following steps:

1. Collecting a large dataset of mango images
2. Labeling the images with corresponding ripeness grades
3. Training a deep learning model on the labeled dataset
4. Applying the trained model to predict the ripeness of new mango images
5. Verifying the predictions and refining the model if necessary



3. Training a deep learning model on the labeled dataset
4. Applying the trained model to predict the ripeness of new mango images
5. Verifying the predictions and refining the model if necessary

Graded-Quality: Perfectly Riped

The given mango is **‘Perfectly Riped’**.

To get to the home page again, hit on the ‘Home’ button in the top right corner and you will be redirected to the home.html page again.

Link to GitHub Repository:

https://github.com/SkilledSpark/Ripe-Sense