Ripe-Sense: Mango Quality Grading

With Image Analysis and DeepLearning

By:

Lakshya Singh

Mayank M

Akshanth Sai K

Jahnavi Jain

Index

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

1. Introduction

1.1 Project Overviews

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.

1.2 Objectives

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.

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.

By the end of this project, we will:

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

2. Project Initialization and Planning Phase

2.1 Define Problem Statement

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.

2.2 Project Proposal (Proposed Solution)

To complete this project, the following software, concepts, and packages are required:

- **Anaconda Navigator**: Refer to the link below to download Anaconda Navigator - Link: [Anaconda Navigator](https://www.youtube.com/watch?v=5mDYijMfSzs)
- **Python packages**:
  - Open Anaconda prompt as administrator
  - Type `pip install tensorflow` (ensure you are working on Python 64 bit)
  - Type `pip install flask`
- **Deep Learning Concepts**:
  - CNN: [Basics of the Classic CNN](https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add)
  - Flask Basics: [Flask Tutorial](https://www.youtube.com/watch?v=lj4I_CvBnt0)

2.3 Initial Project Planning

Project Structure:

- The data obtained consists of 3 folders containing 200 images of respective types of graded mango each, which is used for training, testing, and validation datasets.

- A Flask application will be built requiring 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 be used in the Flask integration.

Milestone 1: Define Problem/Problem Understanding

Activity 1: Specify the Business Problem

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

- **Accurate Mango Quality Grading**: The model should classify mangoes into three categories - perfectly ripen, over ripen, and under ripen - with high accuracy based on visual characteristics.
- **Real-Time Classification**: The model should provide real-time classification results.
- **Compatibility**: The system should be compatible with different web browsers and devices.
- **Reliability**: The system should be robust and reliable.
- **Trus**t: The model should develop trust among users, especially advertisers and content creators.
- **User-Friendly Interface**: The deployment using Flask should provide a user-friendly web interface.
- **Seamless Deployment**: The deployment using Flask should be smooth and seamless.
- **Scalability**: The solution should be scalable to handle a growing number of mango images and users.
- **Flexibility for Future Enhancements**: The system should allow for future enhancements and improvements.

Activity 3: Literature Survey

The literature survey focused on existing research related to image analysis, deep learning, and fruit quality grading. Key findings from the 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 project empowers stakeholders to make informed decisions about mangoes' ripeness and suitability for consumption, reducing food waste and improving market transparency and consumer satisfaction.
- **Business Impact**: The project streamlines operations, reduces manual efforts, and enhances efficiency, leading to improved supply chain management, reduced losses, enhanced customer satisfaction, and increased profitability.

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified

Machine Learning & Deep Learning depend heavily on data. This section guides on how to download the dataset. The data for this project is downloaded from [Kaggle](https://www.kaggle.com/datasets/saurabhshahane/mango-varieties-classification).

Data Collection

Collect the Dataset

- Download the dataset from [Kaggle](https://www.kaggle.com/datasets/saurabhshahane/mango-varieties-classification).

3.2 Data Quality Report

Analyze the Data

- Read and understand the data using visualization techniques.

3.3 Data Preprocessing

Image Preprocessing

Import the ImageDataGenerator Library

- Import the `ImageDataGenerator` class from Keras.

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

Configure ImageDataGenerator Class

- Configure the `ImageDataGenerator` class for real-time data augmentation during model training.

Apply ImageDataGenerator Functionality to Train set and Test set

- Training Dataset: Configure and apply various transformations to the training images.
- Validation Dataset: Create a `validation_generator` for batches of validation data.
- Test Dataset: Create a `test_generator` for batches of test data.

```
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)
```

```
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')
```

Sample Image Visualization

- The `show_image_samples` function displays a grid of sample images along with their corresponding labels for visual inspection and verification.

```
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()
```

4. Model Development Phase

4.1 Model Selection Report

Importing the Model Building Libraries

- Import necessary libraries for model building.

```
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
```

Building an EfficientNet-based Transfer Learning Model Architecture

- Create a model for mango quality grading using transfer learning with the EfficientNet 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})
```

4.2 Initial Model Training Code, Model Validation and Evaluation Report

Compile and train the model using the Adam optimizer, categorical cross-entropy loss function, and accuracy metric.

```
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
```

```
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 0.00012500000593371814.
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 6.250000296685907e-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 3.9062501855369196e-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. Model Optimization and Tuning Phase

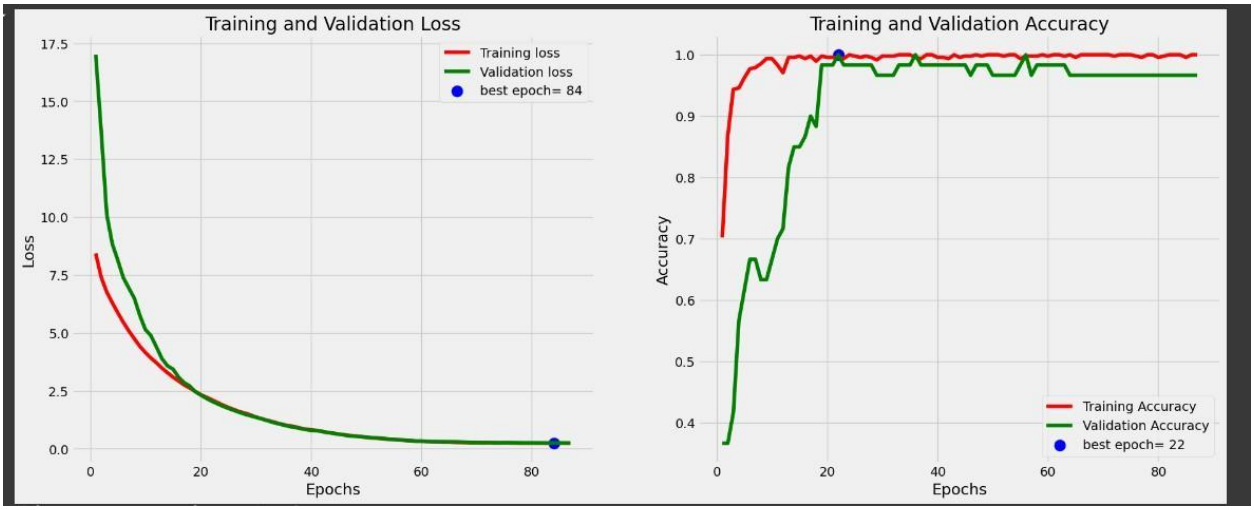
5.1 Tuning Documentation

Visualization of Training and Validation Performance

```
def tr_plot(tr_data, start_epoch):
    #Plot the training and validation data
    tacc=tr_data.history['accuracy']
    tloss=tr_data.history['loss']
    vacc=tr_data.history['val_accuracy']
    vloss=tr_data.history['val_loss']
    Epoch_count=len(tacc)+ start_epoch
    Epochs=[]
    for i in range (start_epoch ,Epoch_count):
        Epochs.append(i+1)
    index_loss=np.argmin(vloss)#  this is the epoch with the lowest validation loss
    val_lowest=vloss[index_loss]
    index_acc=np.argmax(vacc)
    acc_highest=vacc[index_acc]
    plt.style.use('fivethirtyeight')
    sc_label='best epoch= '+ str(index_loss+1 +start_epoch)
    vc_label='best epoch= '+ str(index_acc + 1+ start_epoch)
    fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(20,8))
    axes[0].plot(Epochs,tloss, 'r', label='Training loss')
    axes[0].plot(Epochs,vloss,'g',label='Validation loss' )
    axes[0].scatter(index_loss+1 +start_epoch,val_lowest, s=150, c= 'blue', label=sc_label)
    axes[0].set_title('Training and Validation Loss')
    axes[0].set_xlabel('Epochs')
    axes[0].set_ylabel('Loss')
    axes[0].legend()
    axes[1].plot (Epochs,tacc,'r',label= 'Training Accuracy')
    axes[1].plot (Epochs,vacc,'g',label= 'Validation Accuracy')
    axes[1].scatter(index_acc+1 +start_epoch,acc_highest, s=150, c= 'blue', label=vc_label)
    axes[1].set_title('Training and Validation Accuracy')
    axes[1].set_xlabel('Epochs')
    axes[1].set_ylabel('Accuracy')
    axes[1].legend()
    plt.tight_layout
    #plt.style.use('fivethirtyeight')
    plt.show()
```

```
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')
```

Plot accuracy and loss values over epochs for both the training and validation datasets.



5.2 Final Model Selection Justification

Predicting Mango Variety from an Uploaded Image

Load and preprocess an image for model prediction.

6. Results

6.1 Output Screenshots

Performance Testing

Evaluating Model on Validation Set

- Evaluate the model's performance on the validation dataset, achieving an accuracy of 96.67%.

```
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%
```

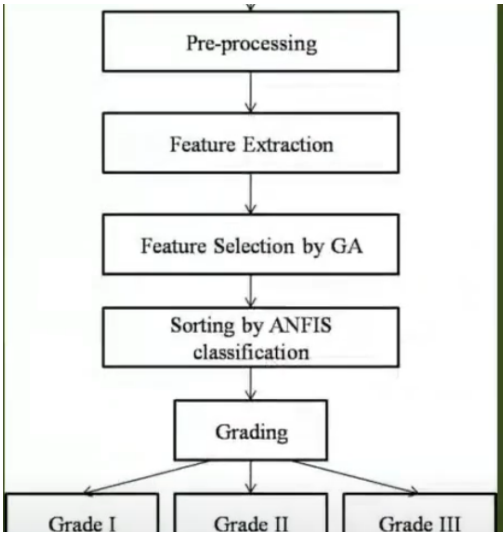
Evaluating Model on Test Set

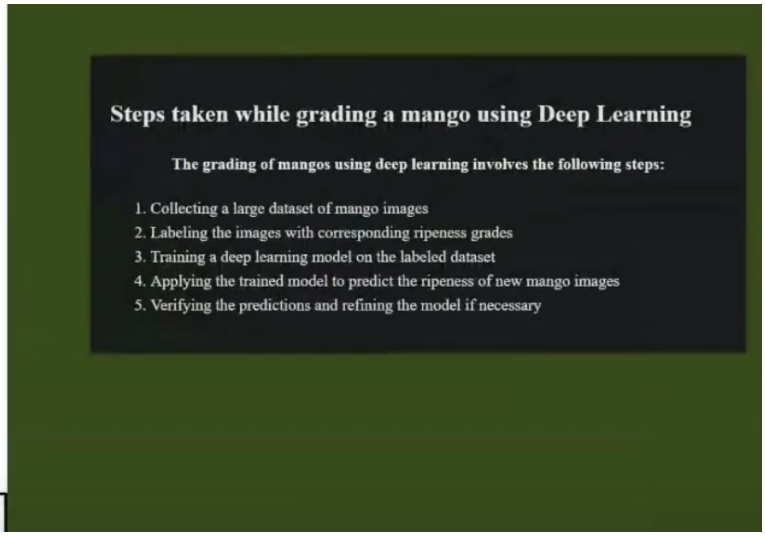
- Evaluate the model's performance on the test dataset, achieving an accuracy of 98.33%.

```
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%
```

Web Application





7. Advantages & Disadvantages

- **Advantages**: Automated, objective, accurate, and efficient mango quality grading process.
- **Disadvantages**: Dependence on quality and quantity of training data, computational resources required for deep learning models.

8. Conclusion

The "Ripe-Sense" project successfully developed a deep learning model that accurately classifies mangoes into perfectly ripen, over ripen, and under ripen categories. The system provides an automated, objective, and efficient solution for mango quality grading, benefiting farmers, distributors, and consumers alike. The project demonstrates the potential of deep learning and computer vision in transforming traditional agricultural practices and enhancing the quality of produce in the market.

9. Future Scope

- **Dataset Expansion**: Increasing the dataset size and diversity for better model generalization.
- **Model Improvements**: Experimenting with different architectures and hyperparameters.
- **Deployment Enhancements**: Improving the web application for wider accessibility.
- **Real-Time Implementation**: Integrating the system with IoT devices for real-time mango grading.

10. Appendix

10.1. Source Code (Flask)

```python
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

app = Flask(__name__)

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

# Define a function to preprocess the input image
def preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.
    return img_array

# Define the home page
@app.route('/')
def home():
    return render_template('index.html')

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

@app.route('/predict', methods=['POST'])
```

```python
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('results.html', prediction=prediction_str, image=img_base64)



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

10.2. GitHub & Project Demo Link

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

Project Demo Link: https://drive.google.com/file/d/1QO3Qrb7m2U4aPgKEomYUD1gixnw1PmRZ/view?usp=sharing