MangoNet-

A VGG16-based Neural Network for Mango Classification

Project Hand-Out
-Kamya Paliwal

MangoNet: A VGG16-based Neural Network for Mango Classification

Mango is one of the most popular fruits around the world, and it comes in many different varieties, each with its unique taste, texture, and appearance. In this project, we aim to develop a deep learning model for the classification of mangoes based on their variety. The model uses the VGG16 architecture, which is a powerful convolutional neural network (CNN) for image recognition tasks.

The model is trained on a dataset of various mango images, including popular varieties like dosehri, fajri, Chaunsa, Anwar Ratool, and others. The dataset is preprocessed to ensure that the images are standardized and ready for the model to use. The VGG16 architecture is used as the basis for the neural network, with modifications made to the output layer to accommodate the classification task.

Overall, MangoNet is a powerful tool for mango classification that demonstrates the potential of deep learning and computer vision techniques in the field of agriculture. The project highlights the importance of accurate and efficient fruit classification, which can improve the quality of fruit production and facilitate trade and commerce.

Aim:

The aim of this project is to develop MangoNet, a VGG16-based neural network for the accurate classification of different types of mangoes based on input images. The model will be deployed using Flask, providing an accessible tool for farmers, traders, and mango enthusiasts to identify different mango varieties.

Purpose of doing this Project:

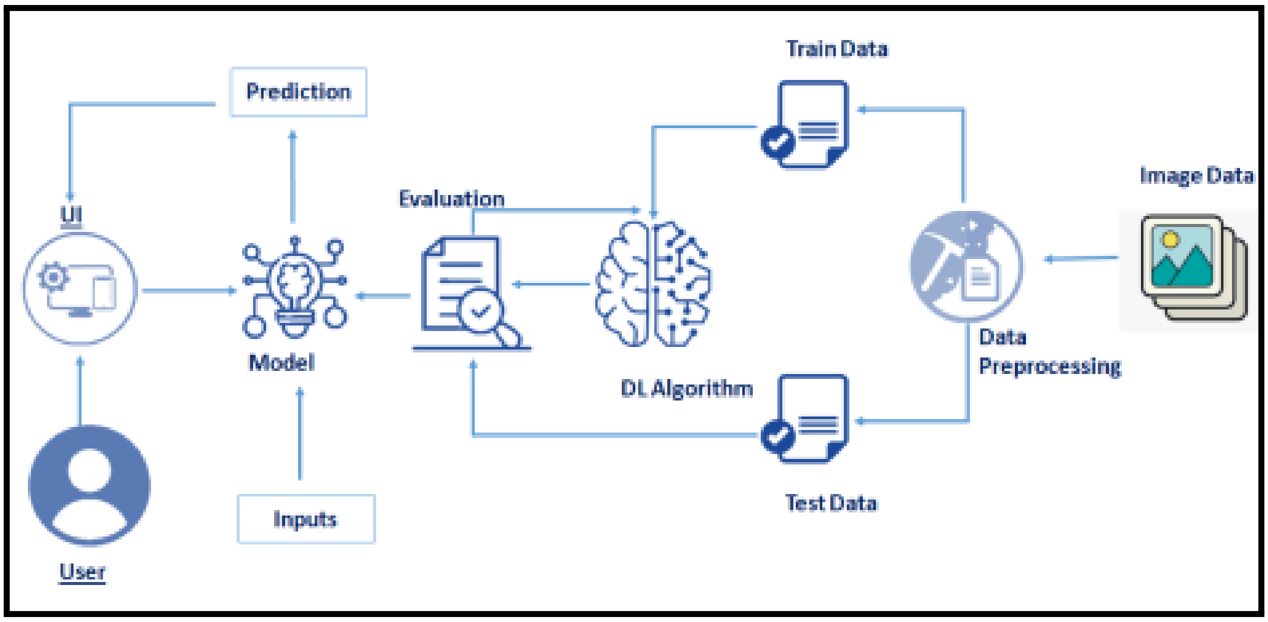
The purpose of this project is to demonstrate the effectiveness of deep learning and computer vision techniques in the field of agriculture, specifically for the classification of mangoes. MangoNet provides an accessible and accurate tool for farmers, traders, and mango enthusiasts to identify different mango varieties, facilitating trade and commerce and improving the quality of fruit production. Additionally, the project serves as a proof-of-concept for the potential applications of deep learning in the agricultural industry.

Technical Architecture:

The MangoNet project uses the VGG16 architecture, a deep convolutional neural network for image recognition tasks. The model is trained on a large dataset of mango images, which are preprocessed for standardization and augmented for improved model generalization. The output layer of the VGG16 model is modified to accommodate the classification task, resulting in a model that accurately classifies different types of mangoes based on input images. The trained model is saved as a file for deployment on a remote server.

The deployment architecture involves hosting the MangoNet model on a Flask web server. The Flask app provides a user-friendly web interface for users to upload mango images and receive predictions of their variety.

MangoNet provides an accessible and accurate tool for the classification of mangoes, with potential applications in agriculture and food industries.



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 you 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:

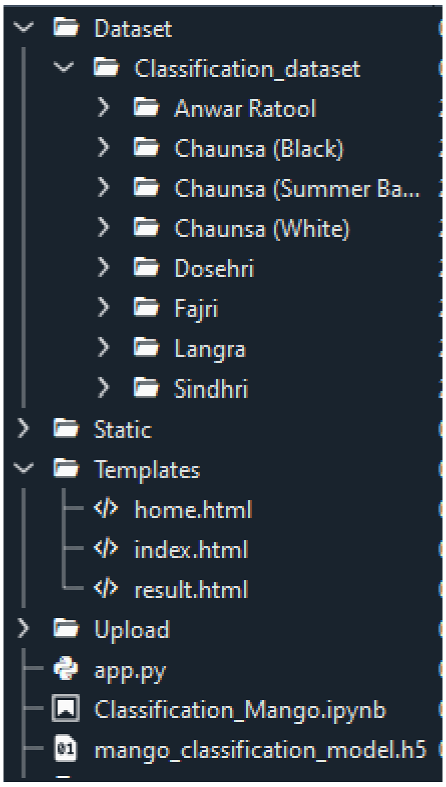
- User interacts with the UI (User Interface) to upload the image as input
- Uploaded image is analyzed by the model which is integrated
- Once model analyses the uploaded image, the prediction food recipe is showcased on the UI to accomplish this, we have to complete all the activities and tasks listed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



```
Dataset
  Classification_dataset
    Anwar Ratool
    Chaunsa (Black)
    Chaunsa (Summer Ba...
    Chaunsa (White)
    Dosehri
    Fajri
    Langra
    Sindhri
Static
Templates
  home.html
  index.html
  result.html
Upload
app.py
Classification_Mango.ipynb
mango_classification_model.h5
```

- The data obtained 8 folders containing 200 images of respective type of 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.
- mango_classification_model.h5 is the saved model. This will further be used in the Flask integration.
- Whenever we upload an image to predict, that images is saved in uploads folder.

Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

he business problem MangoNet solves is the accurate and efficient classification of mangoes. The current visual inspection process by trained personnel can be time-consuming and subjective, leading to errors, inconsistencies, and inefficiencies in mango production and distribution. MangoNet offers an automated and objective solution using deep learning techniques to classify mangoes based on their images, improving efficiency, reducing costs, and increasing the accuracy and consistency of mango classification, benefiting both producers and consumers.

Activity 2: Business Requirements

1. Accurate and efficient classification of mangoes based on their images using deep learning techniques and VGG16 architecture.
2. The ability to handle input images of different resolutions and orientations.
3. The ability to classify multiple types of mangoes, such as dosehri, fajri, Chaunsa, Anwar Ratool, etc.
4. The model should be easy to deploy using Flask to allow users to interact with the model through a web interface.
5. The system should be scalable and able to handle a large volume of mango images.
6. The system should be reliable, with minimal downtime or errors.
7. The system should be cost-effective and provide a significant improvement in efficiency and accuracy compared to manual classification methods.

- **Accurate and reliable information:** The regression models used for predicting adviews should be accurate and reliable since any false information can have severe consequences. The right symptoms and engagement metrics should be linked to the right adviews for providing the most precise output.
- **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.
- **Compliance:** The model should comply with all the relevant laws and regulations set by the Central Drug Standard Control Organization, Ministry of Health, etc.
- **User-friendly interface:** The model should have a user-friendly interface to make it easy for users to input relevant metrics and obtain an estimated number of adviews.

Activity 3: Literature Survey

Several studies have been conducted on fruit classification using deep learning techniques. The VGG16 architecture has been widely used for image classification tasks. Similar projects have been developed for banana and apple classification. However, there is a lack of research on mango classification using deep learning. This project aims to fill that gap by developing a MangoNet model based on the VGG16 architecture for accurate and efficient mango classification.

Activity 4: Social or Business Impact.

Social Impact:

The social impact of the MangoNet project is mainly in the agricultural industry. The classification of mango types can help farmers to identify and sort the different types of mangoes, ensuring better pricing and marketing opportunities. Accurate classification can also aid in improving the supply chain and reducing food waste. This project can also be useful for researchers studying mango varieties and their growth patterns, ultimately helping to improve mango cultivation techniques. Overall, the MangoNet project can have a positive impact on the agriculture industry, leading to better income for farmers and more sustainable food practices.

Business Impact:

The social impact of MangoNet could be to help farmers in identifying and sorting the different types of mangoes, thereby increasing their market value and reducing food waste. The model could also aid in promoting the export of high-quality mangoes. From a business standpoint, MangoNet could potentially be utilized by mango exporters, retailers, and distributors to classify and sort their mangoes efficiently. This could lead to increased sales, improved supply chain management, and enhanced customer satisfaction.

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.

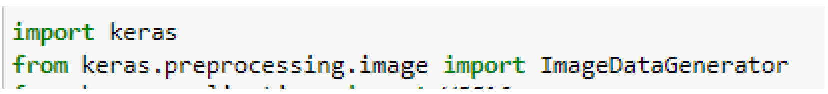
Milestone 3: Image Preprocessing

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

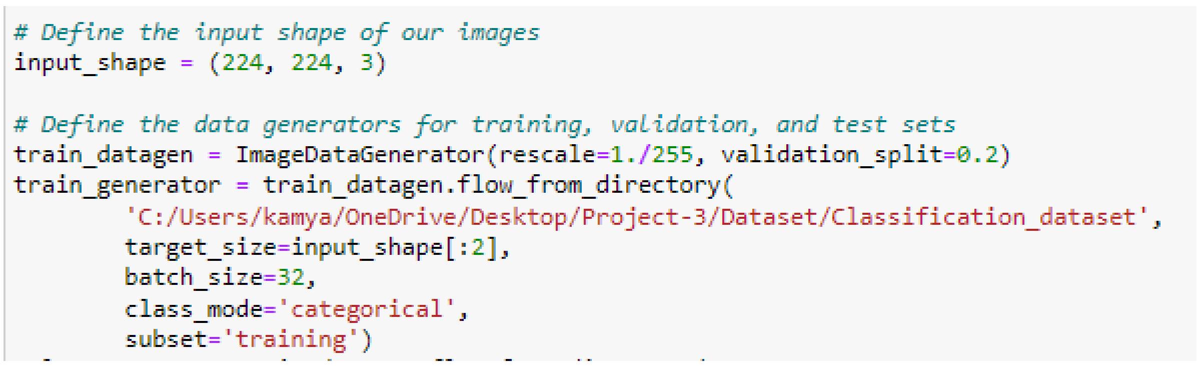


```
import keras
from keras.preprocessing.image import ImageDataGenerator
```

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.

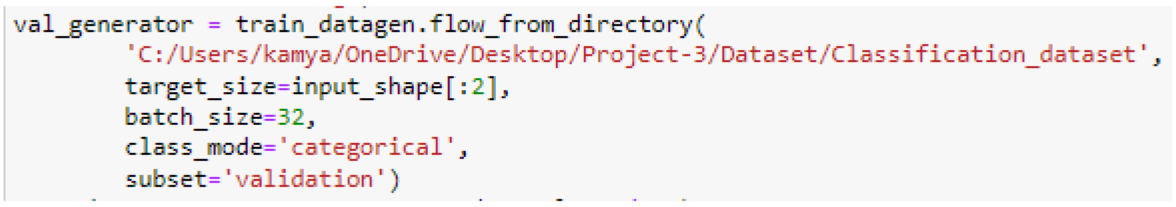
Activity 3 :Apply ImageDataGenerator functionality to Trainset and Testset



```
# Define the input shape of our images
input_shape = (224, 224, 3)

# Define the data generators for training, validation, and test sets
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_generator = train_datagen.flow_from_directory(
        'C:/Users/kamya/OneDrive/Desktop/Project-3/Dataset/Classification_dataset',
        target_size=input_shape[:2],
        batch_size=32,
        class_mode='categorical',
        subset='training')
```

For Training dataset

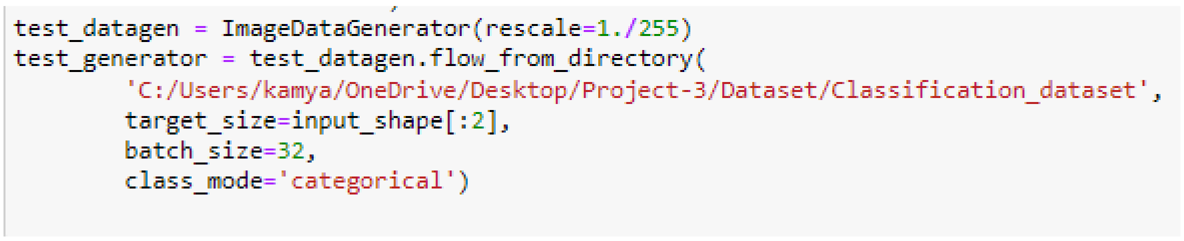
The code defines the input shape of the images as 224x224x3 and creates an ImageDataGenerator object for training data with rescaling factor of 1/255 and 20% of the data for validation. Then, it creates a generator object for training data with target size of (224, 224), batch size of 32, categorical class mode, and a subset of 'training' which indicates that it should use only the training data for the generator.



```
val_generator = train_datagen.flow_from_directory(
        'C:/Users/kamya/OneDrive/Desktop/Project-3/Dataset/Classification_dataset',
        target_size=input_shape[:2],
        batch_size=32,
        class_mode='categorical',
        subset='validation')
```

For Validation Dataset

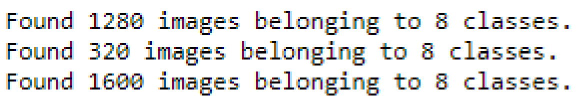
Here, we have defined a validation data generator using the ImageDataGenerator class. The generator will load images from the same directory as the training data generator, but will use a different subset of the data (defined by the "subset" parameter) for validation. The other parameters such as the target size of images, batch size, and class mode are the same as the training data generator.



```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
        'C:/Users/kamya/OneDrive/Desktop/Project-3/Dataset/Classification_dataset',
        target_size=input_shape[:2],
        batch_size=32,
        class_mode='categorical')
```

For Test Dataset

In the given code, a test data generator is defined using the ImageDataGenerator class. The test data generator takes images from the directory specified in the flow_from_directory method and rescales them by a factor of 1./255. The target_size argument specifies the input shape of the images, batch_size specifies the number of images to be generated in each batch and class_mode specifies the type of class labels that the generator should return.

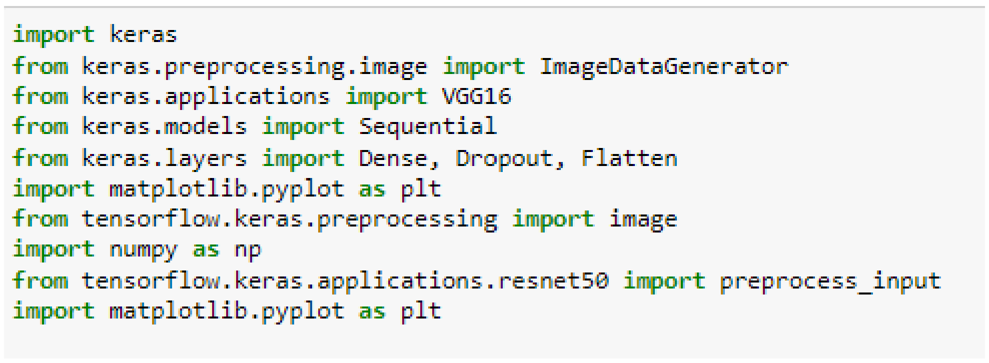


```
Found 1280 images belonging to 8 classes.
Found 320 images belonging to 8 classes.
Found 1600 images belonging to 8 classes.
```

These are the outputs from the flow_from_directory() method of ImageDataGenerator. The first line shows that there are 1280 images in the training set, the second line shows that there are 320 images in the validation set, and the third line shows that there are 1600 images in the test set. All sets have 8 classes in total.

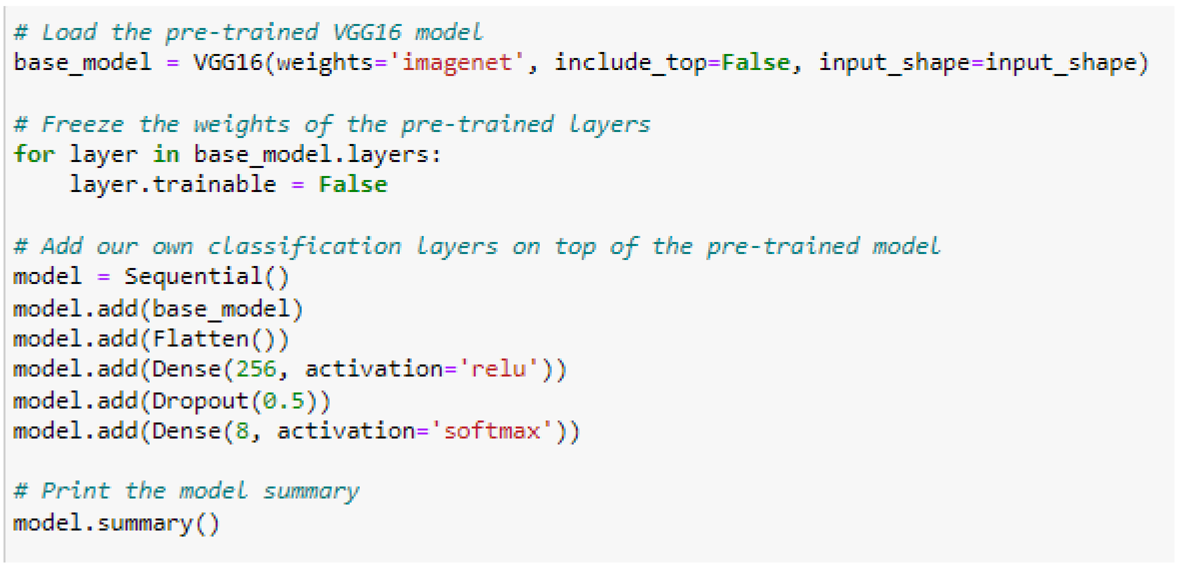
Milestone 4: Model Building

Activity 1 : Importing the Model Building Libraries

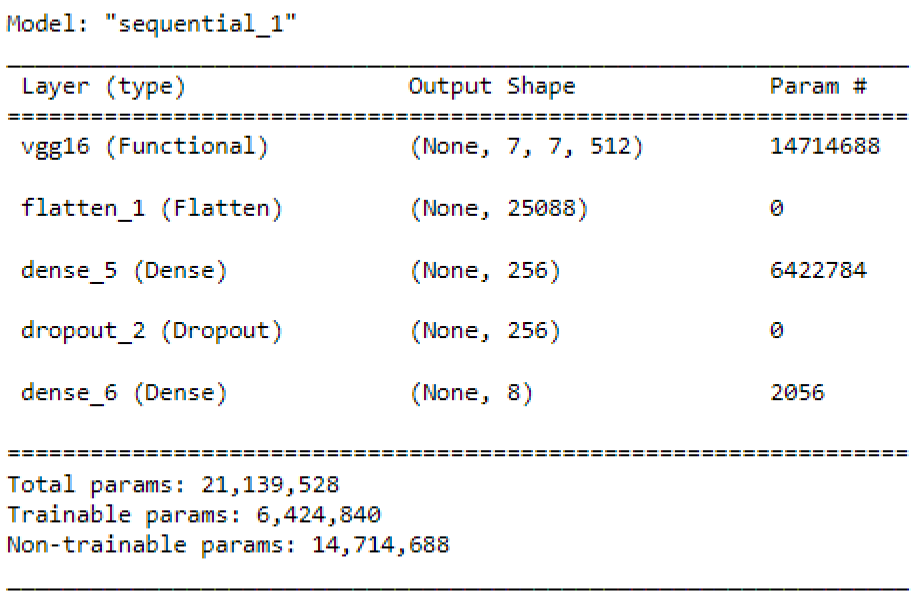


```
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
import numpy as np
from tensorflow.keras.applications.resnet50 import preprocess_input
import matplotlib.pyplot as plt
```

Activity 2: Building a VGG16-based neural network for mango classification with transfer learning.



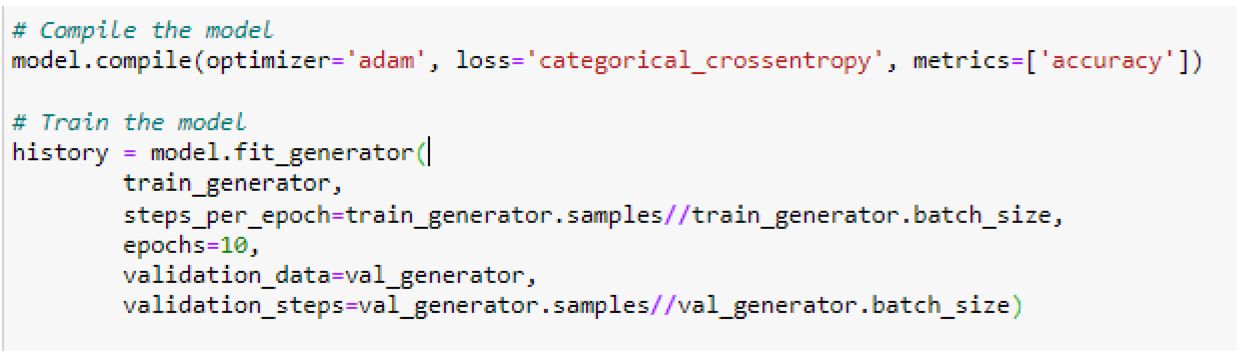
```
# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

# Freeze the weights of the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

# Add our own classification layers on top of the pre-trained model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(8, activation='softmax'))

# Print the model summary
model.summary()
```



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_1 (Flatten)         (None, 25088)             0

 dense_5 (Dense)             (None, 256)               6422784

 dropout_2 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 8)                 2056

=================================================================
Total params: 21,139,528
Trainable params: 6,424,840
Non-trainable params: 14,714,688
_________________________________________________________________
```

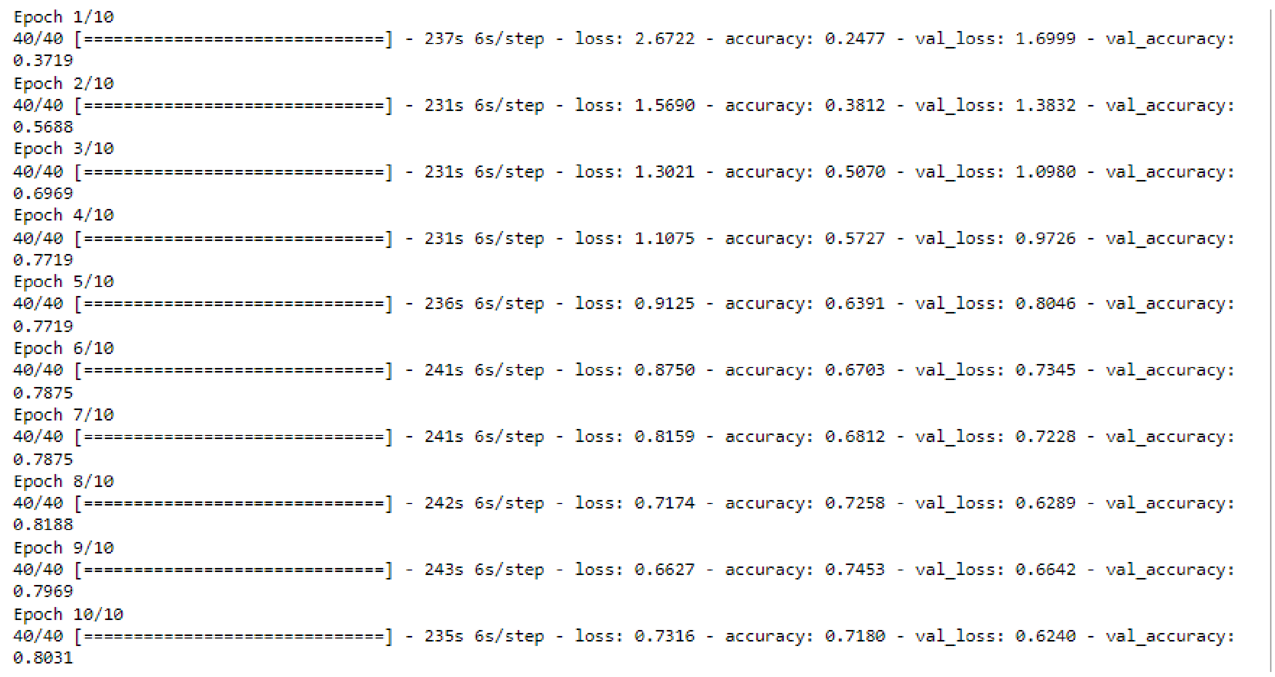
This code block defines a deep learning model for image classification using a pre-trained VGG16 convolutional neural network as the base model. The pre-trained layers' weights are frozen to leverage their feature extraction capabilities, and new classification layers are added on top. The added layers include a flatten layer, two dense layers, and a dropout layer for regularization. The model has a softmax activation function to classify input images into one of eight mango types. The model's architecture and parameters are printed with the summary method.

Activity 3: Training the Model using Fit Generator.



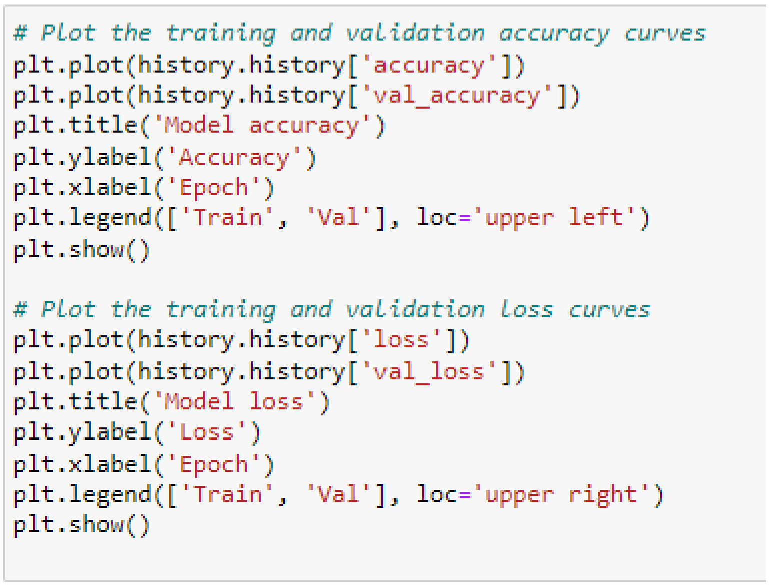
```python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples//train_generator.batch_size,
        epochs=10,
        validation_data=val_generator,
        validation_steps=val_generator.samples//val_generator.batch_size)
```

This code compiles and trains the neural network model for mango classification using the ImageDataGenerator and VGG16 architecture. The model is compiled with the Adam optimizer, categorical cross-entropy loss, and accuracy as the evaluation metric. The model is trained for 10 epochs using the training and validation data generators created earlier, and the training history is stored in the 'history' variable. The number of steps per epoch and validation steps is calculated based on the batch size and number of samples in the respective data generators.



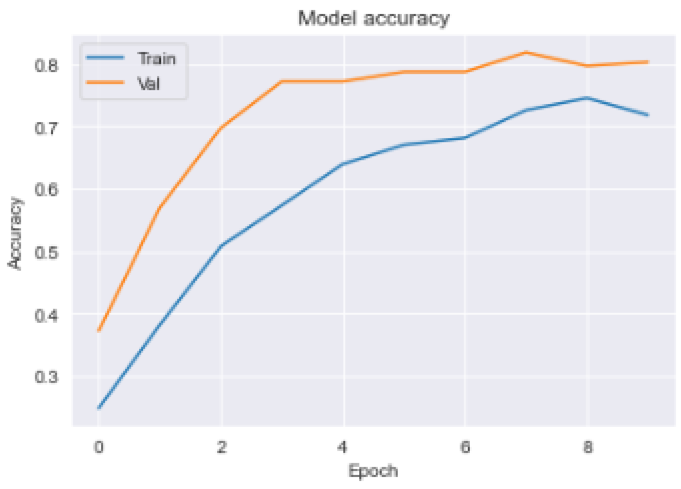
```
Epoch 1/10
40/40 [==============================] - 237s 6s/step - loss: 2.6722 - accuracy: 0.2477 - val_loss: 1.6999 - val_accuracy:
0.3719
Epoch 2/10
40/40 [==============================] - 231s 6s/step - loss: 1.5690 - accuracy: 0.3812 - val_loss: 1.3832 - val_accuracy:
0.5688
Epoch 3/10
40/40 [==============================] - 231s 6s/step - loss: 1.3021 - accuracy: 0.5070 - val_loss: 1.0980 - val_accuracy:
0.6969
Epoch 4/10
40/40 [==============================] - 231s 6s/step - loss: 1.1075 - accuracy: 0.5727 - val_loss: 0.9726 - val_accuracy:
0.7719
Epoch 5/10
40/40 [==============================] - 236s 6s/step - loss: 0.9125 - accuracy: 0.6391 - val_loss: 0.8046 - val_accuracy:
0.7719
Epoch 6/10
40/40 [==============================] - 241s 6s/step - loss: 0.8750 - accuracy: 0.6703 - val_loss: 0.7345 - val_accuracy:
0.7875
Epoch 7/10
40/40 [==============================] - 241s 6s/step - loss: 0.8159 - accuracy: 0.6812 - val_loss: 0.7228 - val_accuracy:
0.7875
Epoch 8/10
40/40 [==============================] - 242s 6s/step - loss: 0.7174 - accuracy: 0.7258 - val_loss: 0.6289 - val_accuracy:
0.8188
Epoch 9/10
40/40 [==============================] - 243s 6s/step - loss: 0.6627 - accuracy: 0.7453 - val_loss: 0.6642 - val_accuracy:
0.7969
Epoch 10/10
40/40 [==============================] - 235s 6s/step - loss: 0.7316 - accuracy: 0.7180 - val_loss: 0.6240 - val_accuracy:
0.8031
```

Milestone 5: Visualization of Training and Validation Performance.

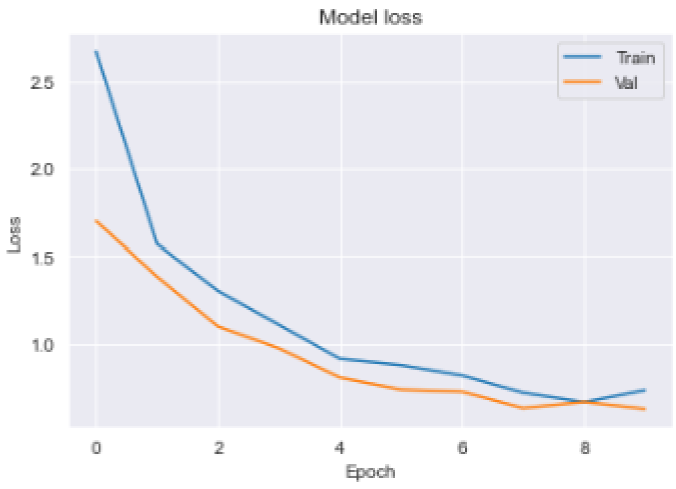


```python
# Plot the training and validation accuracy curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

# Plot the training and validation loss curves
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```

This code block plots the training and validation accuracy curves and the training and validation loss curves for a machine learning model. The graphs allow us to visualize how the model performed during training and validation over each epoch. The training and validation accuracy curves indicate how accurate the model's predictions were, while the training and validation loss curves indicate how well the model was able to minimize its loss function during training.

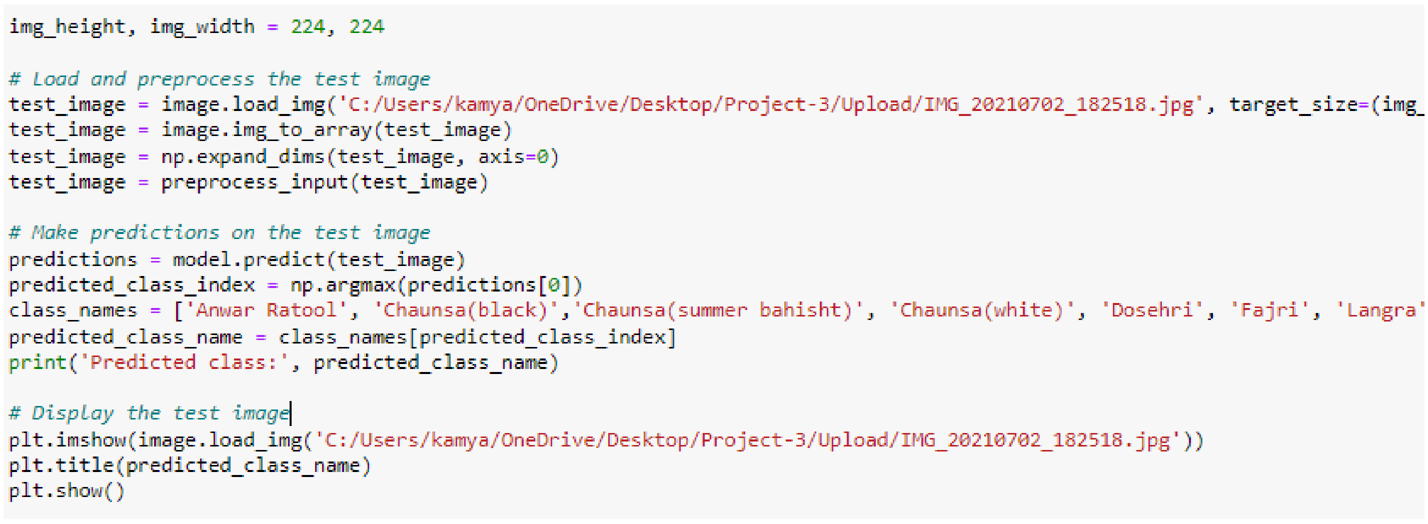


Model Accuracy



Model Accuracy

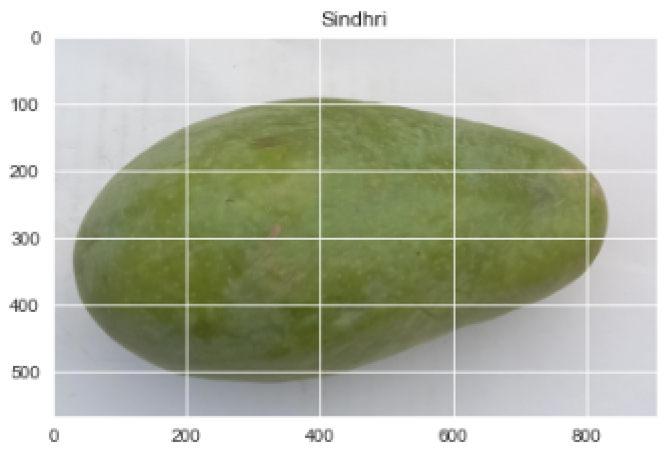
Activity 2: Predicting mango variety from an uploaded image



```
img_height, img_width = 224, 224

# Load and preprocess the test image
test_image = image.load_img('C:/Users/kamya/OneDrive/Desktop/Project-3/Upload/IMG_20210702_182518.jpg', target_size=(img_
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
test_image = preprocess_input(test_image)

# Make predictions on the test image
predictions = model.predict(test_image)
predicted_class_index = np.argmax(predictions[0])
class_names = ['Anwar Ratool', 'Chaunsa(black)','Chaunsa(summer bahisht)', 'Chaunsa(white)', 'Dosehri', 'Fajri', 'Langra'
predicted_class_name = class_names[predicted_class_index]
print('Predicted class:', predicted_class_name)

# Display the test image
plt.imshow(image.load_img('C:/Users/kamya/OneDrive/Desktop/Project-3/Upload/IMG_20210702_182518.jpg'))
plt.title(predicted_class_name)
plt.show()
```

This code loads a test image, preprocesses it, makes a prediction using a pre-trained model, and displays the predicted class along with the original image. The predicted class is determined by the highest probability value in the predicted output. The class names are pre-defined and are associated with the corresponding index value in the predicted output. Finally, the image is displayed with the predicted class as the title.

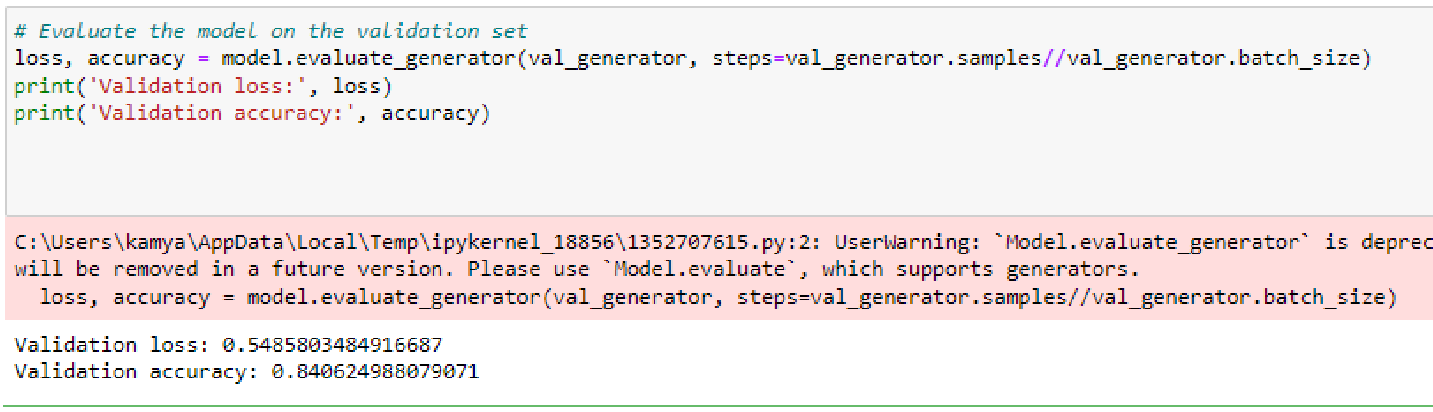
```
1/1 [==============================] - 0s 124ms/step
Predicted class: Sindhri
```



Milestone 6 : Performance Testing

Activity 1: Evaluating model on validation set

This code evaluates the performance of the trained model on the validation set and prints the validation loss and accuracy. The evaluate_generator method computes the loss and accuracy of the model on the validation set, based on the predictions and actual targets.



```
# Evaluate the model on the validation set
loss, accuracy = model.evaluate_generator(val_generator, steps=val_generator.samples//val_generator.batch_size)
print('Validation loss:', loss)
print('Validation accuracy:', accuracy)
```

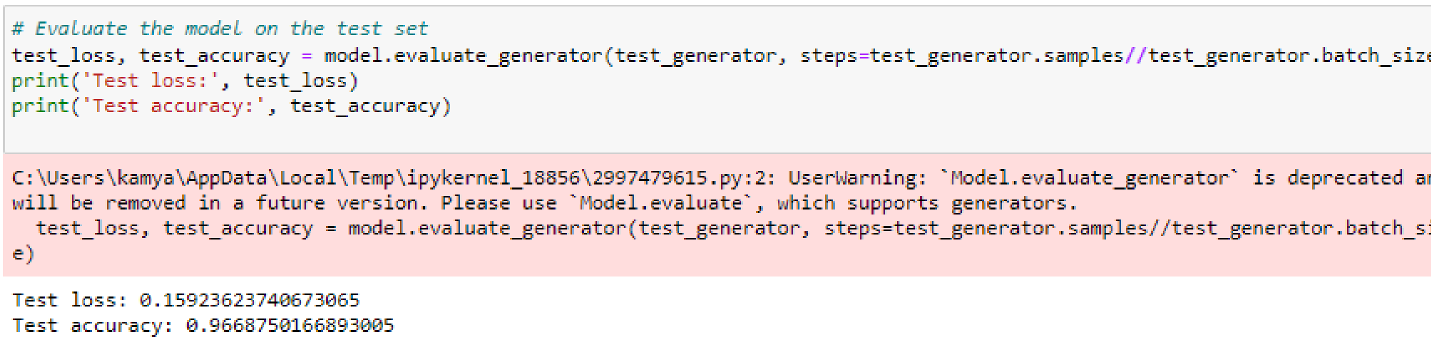
```
C:\Users\kamya\AppData\Local\Temp\ipykernel_18856\1352707615.py:2: UserWarning: `Model.evaluate_generator` is deprec
will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  loss, accuracy = model.evaluate_generator(val_generator, steps=val_generator.samples//val_generator.batch_size)
```

```
Validation loss: 0.5485803484916687
Validation accuracy: 0.840624988079071
```

The validation loss and accuracy of the trained model are printed using the evaluate_generator function, which takes the validation data as input and calculates the loss and accuracy on the validation set. The obtained validation loss of 0.548 and accuracy of 0.840 indicate that the model is performing well on the unseen validation data. A low validation loss indicates that the model is not overfitting to the training data, while a high validation accuracy indicates that the model is generalizing well to the validation set. Therefore, these metrics help us to assess the performance of the model and fine-tune it for better accuracy on unseen data.

Activity 2: Evaluating model on test set

This code evaluates the performance of the trained model on the test set and prints the test loss and accuracy. The evaluate_generator method computes the loss and accuracy of the model on the test set, based on the predictions and actual targets.



```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate_generator(test_generator, steps=test_generator.samples//test_generator.batch_size
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
C:\Users\kamya\AppData\Local\Temp\ipykernel_18856\2997479615.py:2: UserWarning: `Model.evaluate_generator` is deprecated an
will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  test_loss, test_accuracy = model.evaluate_generator(test_generator, steps=test_generator.samples//test_generator.batch_si
e)
```

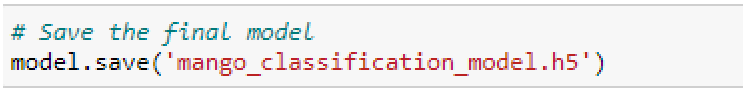
```
Test loss: 0.15923623740673065
Test accuracy: 0.9668750166893005
```

The test loss of a machine learning model is a measure of how well the model is able to generalize to new, unseen data. A lower test loss indicates better generalization performance. In this case, the test loss of the model is 0.159, which is relatively low and suggests that the model is performing well on new, unseen data.

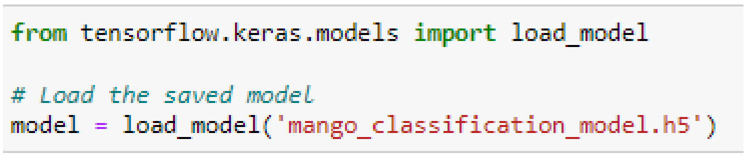
The test accuracy of the model is a measure of how often the model correctly predicts the label of a new sample. A higher test accuracy indicates better prediction performance. In this case, the test accuracy of the model is 0.967, which is relatively high and suggests that the model is making accurate predictions on new, unseen data.

Milestone 6: Model Deployment

Activity 1: Save and load the best model



```python
# Save the final model
model.save('mango_classification_model.h5')
```



```python
from tensorflow.keras.models import load_model

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

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

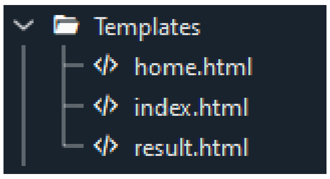
● Run the web application

Activity 2.1: Building HTML pages:

For this project we create two HTML files namely

• Index.html
• Results.html
• Home.html

And we will save them in the templates folder.

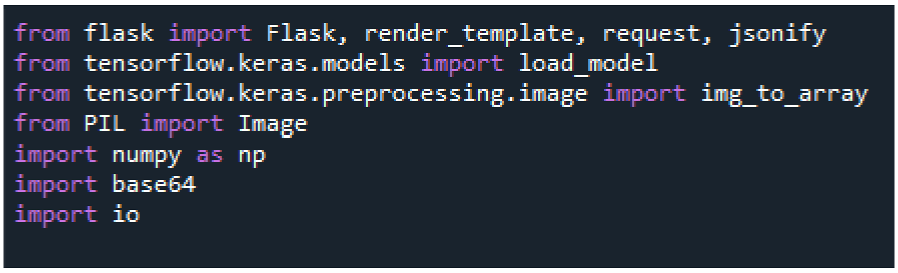


```
Templates
├ home.html
├ index.html
└ result.html
```

Activity 2.2: Build Python code

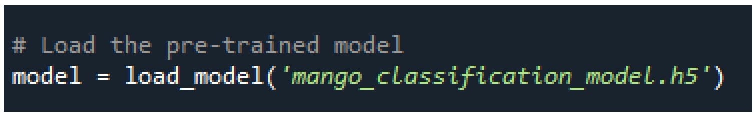
Create a new app.py file which will be store in the Flask folder.

● Import the necessary Libraries.



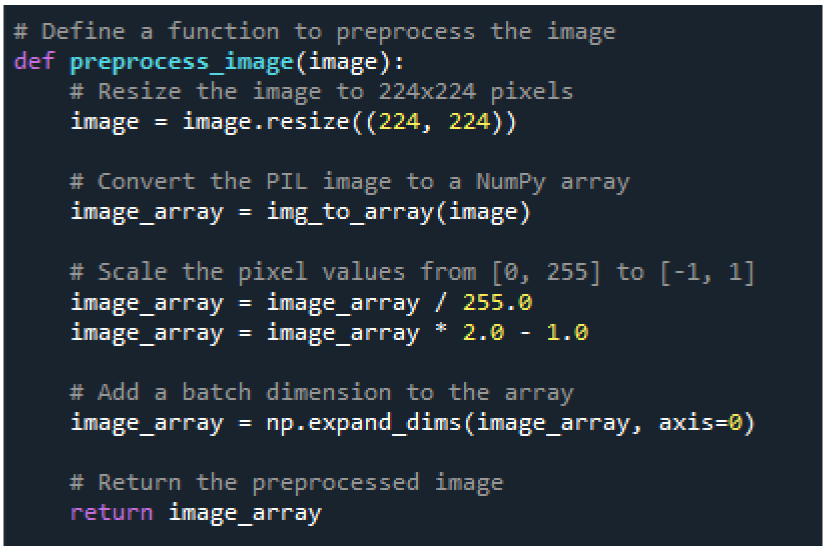
```python
from flask import Flask, render_template, request, jsonify
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
from PIL import Image
import numpy as np
import base64
import io
```

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



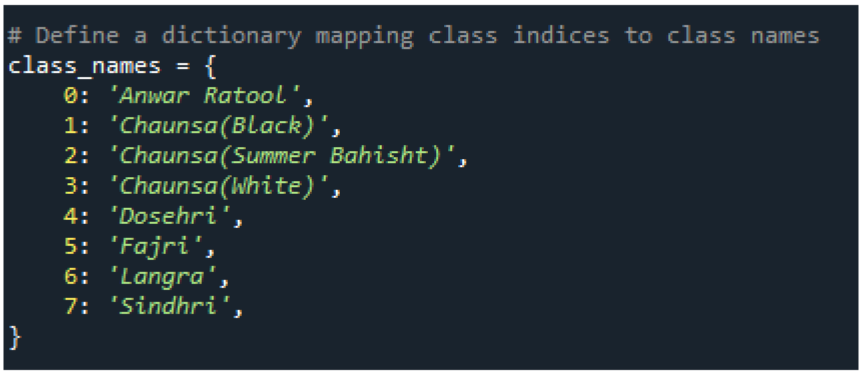
```
# Load the pre-trained model
model = load_model('mango_classification_model.h5')
```

- This code defines a function called preprocess_image which takes an image as input and performs several pre-processing steps on it. First, it resizes the image to 224x224 pixels. Then, it converts the image from a PIL image object to a NumPy array. The pixel values of the array are then scaled from [0, 255] to [-1, 1]. Finally, a batch dimension is added to the array to prepare it for input to the neural network. The function returns the preprocessed image as a NumPy array. This function can be used to preprocess images before feeding them to the pre-trained model for classification.



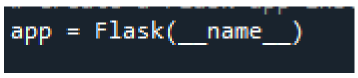
```
# Define a function to preprocess the image
def preprocess_image(image):
    # Resize the image to 224x224 pixels
    image = image.resize((224, 224))

    # Convert the PIL image to a NumPy array
    image_array = img_to_array(image)

    # Scale the pixel values from [0, 255] to [-1, 1]
    image_array = image_array / 255.0
    image_array = image_array * 2.0 - 1.0

    # Add a batch dimension to the array
    image_array = np.expand_dims(image_array, axis=0)

    # Return the preprocessed image
    return image_array
```

- This dictionary maps the integer class indices to their corresponding class names. It is used to translate the output of the model, which is a predicted class index, to the actual class name. In this case, the model was trained to classify different varieties of mangoes, and the dictionary provides the names of the different mango varieties. For example, if the model outputs a class index of 2, which corresponds to 'Chaunsa(Summer Bahisht)', the dictionary can be used to get the actual name of the mango variety. This helps to provide a more meaningful output to the user.



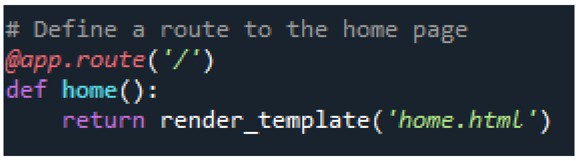
```
# Define a dictionary mapping class indices to class names
class_names = {
    0: 'Anwar Ratool',
    1: 'Chaunsa(Black)',
    2: 'Chaunsa(Summer Bahisht)',
    3: 'Chaunsa(White)',
    4: 'Dosehri',
    5: 'Fajri',
    6: 'Langra',
    7: 'Sindhri',
}
```

- 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.



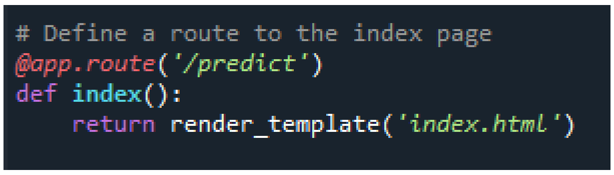
```
app = Flask(__name__)
```

- This defines a route to the home page of the web application. When a user visits the base URL of the application (e.g. https://example.com/), Flask will execute the home() function and return the content of the home.html template. The render_template() function is used to render the HTML template, which can include dynamic content based on data passed to it from the Flask application.



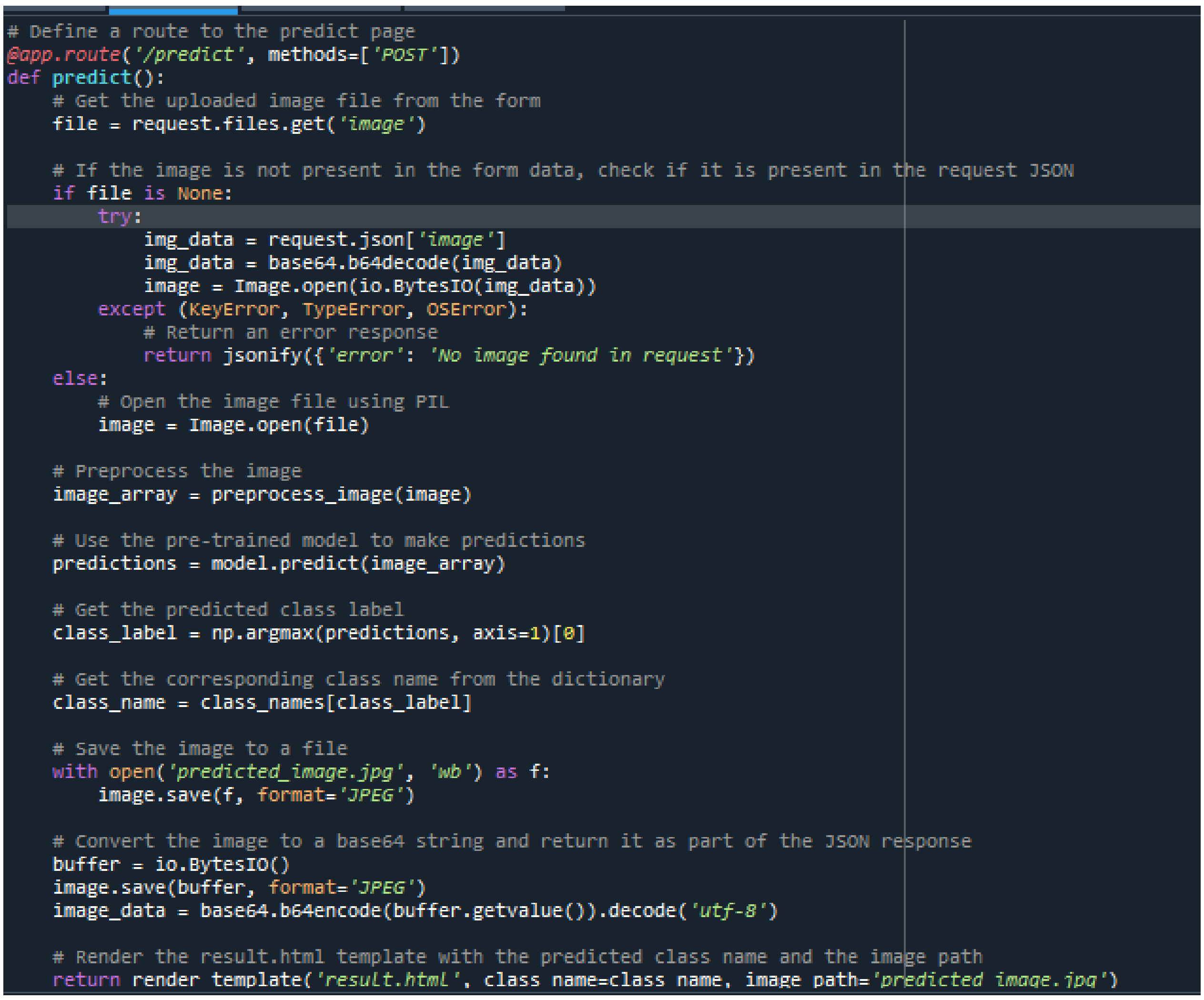
```python
# Define a route to the home page
@app.route('/')
def home():
    return render_template('home.html')
```

- This code defines a route using Flask for the /predict URL. When a user navigates to this URL, the index() function is called, which renders an HTML template called index.html. This template likely contains a form for the user to upload an image to be classified.



```python
# Define a route to the index page
@app.route('/predict')
def index():
    return render_template('index.html')
```

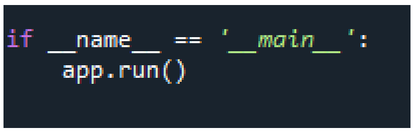
- This code defines a route for predicting the class of a mango image uploaded by the user through a form. The code checks if an image file was uploaded in the form data, if not, it tries to find the image in the request JSON. The image is then preprocessed using the preprocess_image function, and the pre-trained model is used to make a prediction on the image. The predicted class label is obtained, and the corresponding class name is retrieved from the class_names dictionary. The predicted image is saved to a file and converted to a base64 string to be included in the JSON response. Finally, the result.html template is rendered with the predicted class name and the path to the predicted image.



```python
# Define a route to the predict page
@app.route('/predict', methods=['POST'])
def predict():
    # Get the uploaded image file from the form
    file = request.files.get('image')

    # If the image is not present in the form data, check if it is present in the request JSON
    if file is None:
        try:
            img_data = request.json['image']
            img_data = base64.b64decode(img_data)
            image = Image.open(io.BytesIO(img_data))
        except (KeyError, TypeError, OSError):
            # Return an error response
            return jsonify({'error': 'No image found in request'})
    else:
        # Open the image file using PIL
        image = Image.open(file)

    # Preprocess the image
    image_array = preprocess_image(image)

    # Use the pre-trained model to make predictions
    predictions = model.predict(image_array)

    # Get the predicted class label
    class_label = np.argmax(predictions, axis=1)[0]

    # Get the corresponding class name from the dictionary
    class_name = class_names[class_label]

    # Save the image to a file
    with open('predicted_image.jpg', 'wb') as f:
        image.save(f, format='JPEG')

    # Convert the image to a base64 string and return it as part of the JSON response
    buffer = io.BytesIO()
    image.save(buffer, format='JPEG')
    image_data = base64.b64encode(buffer.getvalue()).decode('utf-8')

    # Render the result.html template with the predicted class name and the image path
    return render_template('result.html', class_name=class_name, image_path='predicted_image.jpg')
```

Main Function:

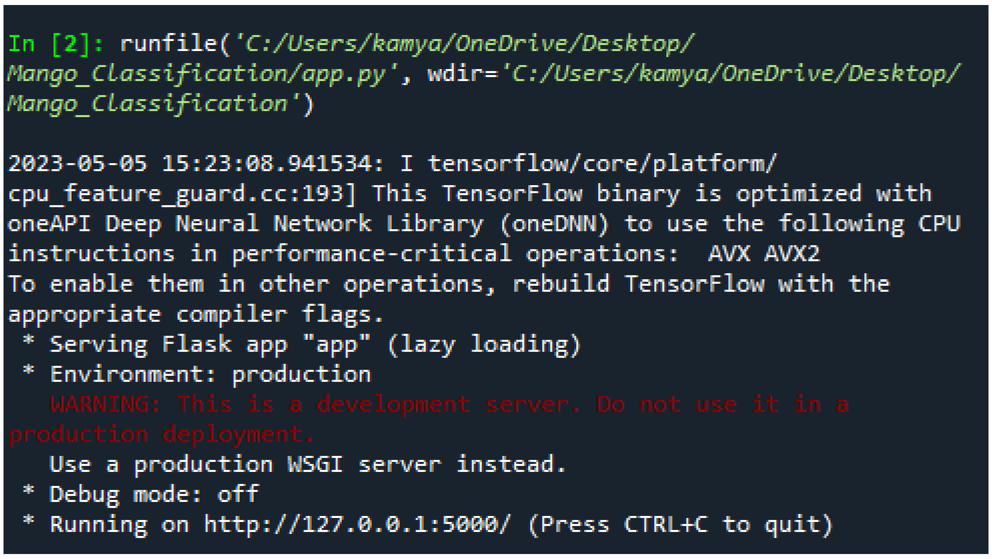
This code runs the Flask application if the script is being executed directly (i.e. not imported as a module in another script). The if __name__ == '__main__': line checks if the script is the main module being executed, and if so, runs the Flask application using the app.run() method. This method starts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



```
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.



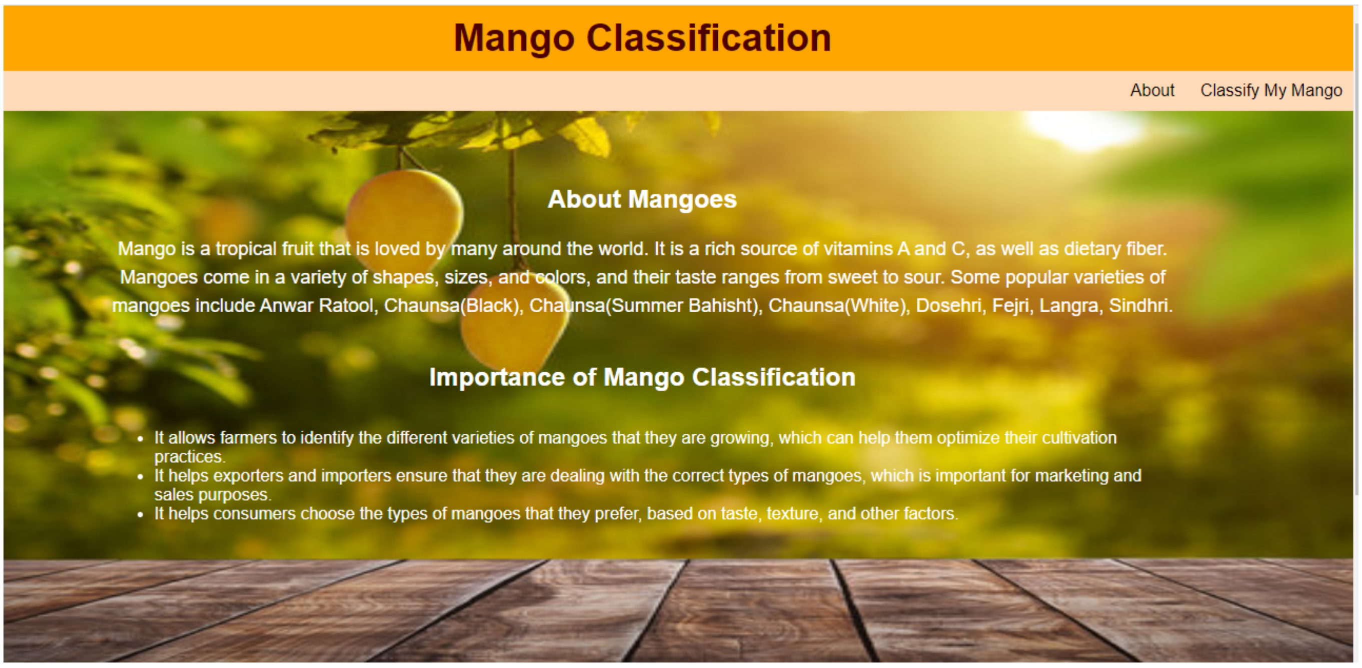
```
In [2]: runfile('C:/Users/kamya/OneDrive/Desktop/
Mango_Classification/app.py', wdir='C:/Users/kamya/OneDrive/Desktop/
Mango_Classification')

2023-05-05 15:23:08.941534: I tensorflow/core/platform/
cpu_feature_guard.cc:193] This TensorFlow binary is optimized with
oneAPI Deep Neural Network Library (oneDNN) to use the following CPU
instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a
production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

When we paste the URL in a web browser, our home.html page will open. It contains information about Mango and various varieties of it.

There are two navigation button on top right- ‘About’, ‘ClassifyMy Mango’.

If you click on the Classify My Mango button on home page you will be redirected to the index.html page.

Our home.html looks as shown below:



Mango Classification

About Classify My Mango

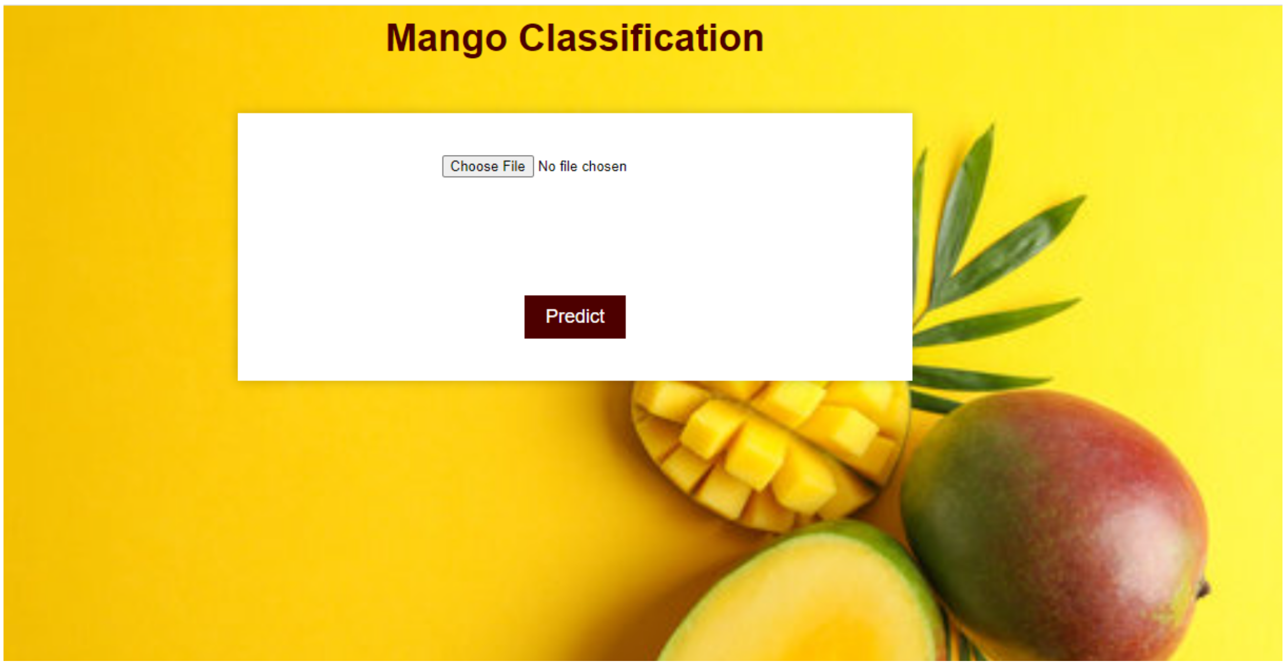
About Mangoes

Mango is a tropical fruit that is loved by many around the world. It is a rich source of vitamins A and C, as well as dietary fiber. Mangoes come in a variety of shapes, sizes, and colors, and their taste ranges from sweet to sour. Some popular varieties of mangoes include Anwar Ratool, Chaunsa(Black), Chaunsa(Summer Bahisht), Chaunsa(White), Dosehri, Fejri, Langra, Sindhri.

Importance of Mango Classification

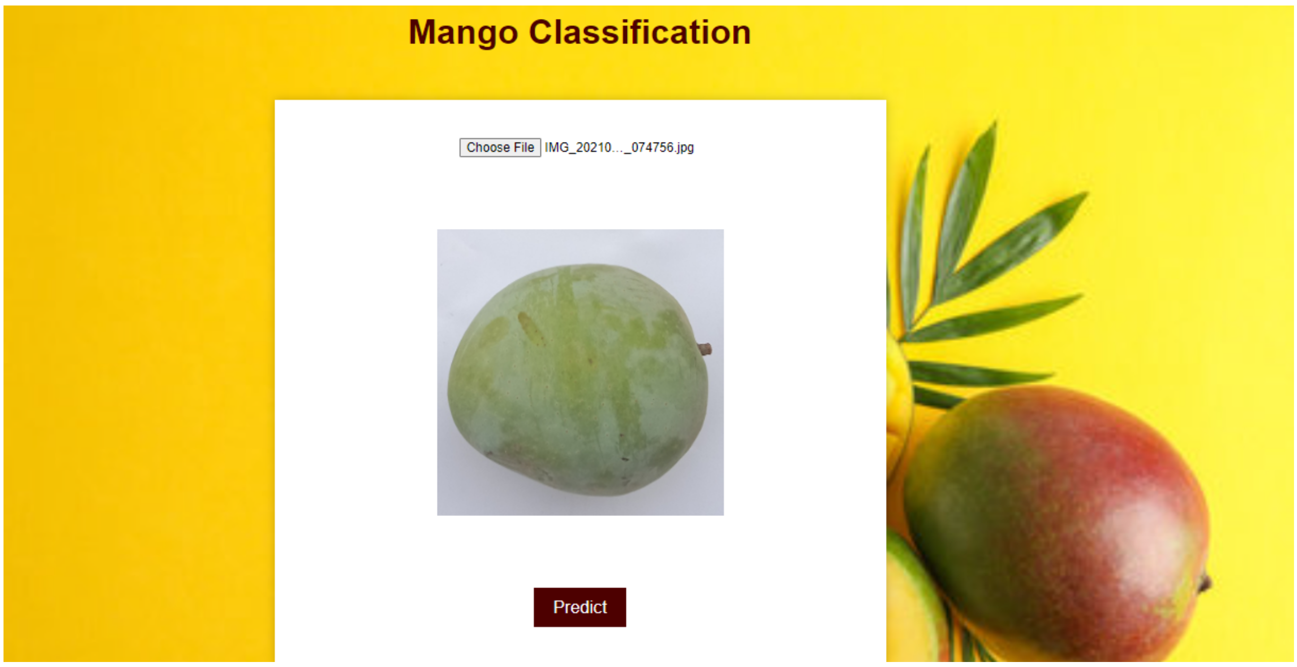
- It allows farmers to identify the different varieties of mangoes that they are growing, which can help them optimize their cultivation practices.
- It helps exporters and importers ensure that they are dealing with the correct types of mangoes, which is important for marketing and sales purposes.
- It helps consumers choose the types of mangoes that they prefer, based on taste, texture, and other factors.

Our index.html page looks like this:

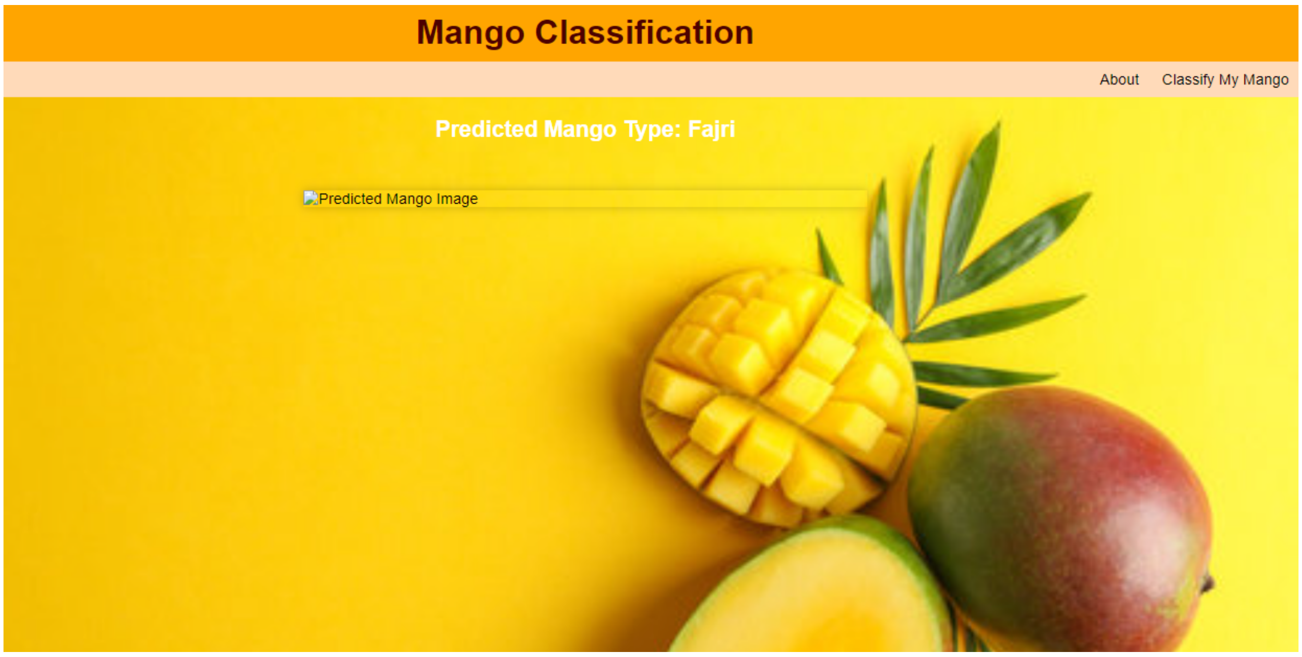


Mango Classification

Choose your image in ‘Choose File’ button and choose the image you want to get the type name of.



Now hit on Predict button. The model will preprocess the image and classify it into one of the 8 Varieties of Mangoes.



Predicted Mango Type: Fajri

The given image is that of **‘Fajri’** type.

To classify the image of new mango, hit on ‘Classify My Mango’ button on top right corner and you will be redirected to index.html page again.

Link to GitHub Repository:

https://github.com/Kamya-Paliwal/MangoNet-A-VGG16-based-Neural-Network-for-Mango-Classification