**Fruit Image Classification Model Based on MobileNetV2 with Deep Transfer Learning Technique**

### **Table of Contents**

1. **Introduction**
   * Overview of the Project
   * Objectives
   * Tools and Libraries Used
2. **Setting Up the Environment**
   * Installing TensorFlow and Keras
   * Installing Flask and Flask-CORS
   * Setting Up a Virtual Environment
   * Directory Structure
3. **Data Preparation**
   * Organizing the Dataset
   * Image Data Preprocessing
   * Creating Training and Validation Generators
4. **Building the Convolutional Neural Network (CNN)**
   * Initializing the Model
   * Adding Convolutional Layers
   * Adding Pooling Layers
   * Flattening the Data
   * Adding Dense Layers
   * Compiling the Model
   * Summary of the Model Architecture
5. **Training the Model**
   * Configuring Training Parameters
   * Fitting the Model on Data
   * Monitoring Training Progress with Accuracy and Loss
6. **Evaluating the Model**
   * Plotting Training and Validation Accuracy
   * Evaluating Model Performance on Test Data
   * Saving the Trained Model
7. **Creating a Flask Web Application**
   * Setting Up Flask
   * Enabling Cross-Origin Resource Sharing (CORS)
   * Loading the Trained Model in Flask
8. **Handling Image Uploads**
   * Defining the Image Preparation Function
   * Writing the Image Upload Endpoint
   * Predicting the Class of Uploaded Images
   * Handling Errors and Edge Cases
9. **Serving HTML Content**
   * Creating and Serving the HTML File
10. **Running the Flask Application**
    * Starting the Flask Server
    * Testing the API Endpoints
11. **Deployment**
    * Preparing for Deployment
    * Deploying to a Cloud Platform (e.g., Heroku, AWS)
12. **Conclusion**
    * Summary of the Project
    * Future Work and Enhancements

### **Detailed Documentation Content**

**1. Introduction**

**Overview of the Project:** This project aims to build an image classification system for fruits using a Convolutional Neural Network (CNN) implemented in TensorFlow and Keras. The system is deployed as a web application using Flask, allowing users to upload images of fruits and receive predictions about the fruit type.

**Objectives:**

* Develop a CNN model to classify images of fruits.
* Train and evaluate the model on a dataset of fruit images.
* Create a Flask web application to serve the model and handle image uploads.
* Deploy the web application for public use.

**Tools and Libraries Used:**

* **TensorFlow and Keras:** For building and training the CNN.
* **Flask:** For creating the web application.
* **Flask-CORS:** To handle Cross-Origin Resource Sharing.
* **PIL:** For image processing.
* **Matplotlib:** For plotting training results.

**2. Setting Up the Environment**

**Installing TensorFlow and Keras:** To install TensorFlow and Keras, run the following command:

pip install tensorflow

**Installing Flask and Flask-CORS:** To install Flask and Flask-CORS, run:

pip install flask flask-cors

**Setting Up a Virtual Environment:** It is recommended to use a virtual environment to manage dependencies. You can create and activate a virtual environment as follows:

python -m venv env

source env/bin/activate # On Windows use `env\Scripts\activate`

**Directory Structure:** The project directory should be organized as follows:

project\_root/

|-- archive/

| |-- train/

| |-- test/

|-- app.py

|-- fruit\_classifier.h5

|-- index.html

**3. Data Preparation**

**Organizing the Dataset:** The dataset should be organized into training and validation directories, with subdirectories for each class of fruit.

**Image Data Preprocessing:** The ImageDataGenerator class from Keras is used to rescale the pixel values of images and generate batches of augmented image data.

train\_datagen = ImageDataGenerator(rescale=1./255)

validation\_datagen = ImageDataGenerator(rescale=1./255)

**Creating Training and Validation Generators:** Generators are created to read images from the directories and preprocess them for training and validation.

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(100, 100),

batch\_size=32,

class\_mode='categorical'

)

validation\_generator = validation\_datagen.flow\_from\_directory(

validation\_dir,

target\_size=(100, 100),

batch\_size=32,

class\_mode='categorical'

)

**4. Building the Convolutional Neural Network (CNN)**

**Initializing the Model:** A Sequential model is initialized.

model = Sequential()

**Adding Convolutional Layers:** Convolutional layers are added to extract features from the images.

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(100, 100, 3)))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

**Adding Pooling Layers:** Pooling layers reduce the spatial dimensions of the feature maps.

**Flattening the Data:** The data is flattened to feed it into fully connected layers.

model.add(Flatten())

**Adding Dense Layers:** Dense layers are added for classification.

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(train\_generator.num\_classes, activation='softmax'))

**Compiling the Model:** The model is compiled with the Adam optimizer and categorical crossentropy loss.

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

**Summary of the Model Architecture:** The model summary provides an overview of the layers and parameters.

model.summary()

**5. Training the Model**

**Configuring Training Parameters:** The model is trained with the training generator and validated with the validation generator.

history = model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples // validation\_generator.batch\_size,

epochs=10

)

**Monitoring Training Progress with Accuracy and Loss:** Training and validation accuracy are plotted to monitor progress.

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label='val\_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0, 1])

plt.legend(loc='lower right')

plt.show()

**6. Evaluating the Model**

**Plotting Training and Validation Accuracy:**

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label='val\_accuracy')

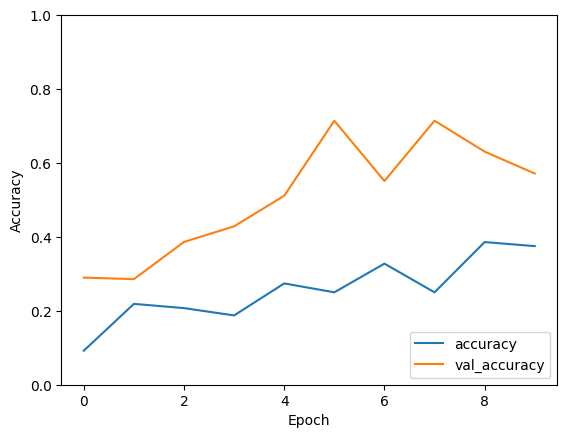
plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0, 1])

plt.legend(loc='lower right')

plt.show()



**Evaluating Model Performance on Test Data:**

test\_loss, test\_acc = model.evaluate(validation\_generator, verbose=2)

print('\nTest accuracy:', test\_acc)

**Saving the Trained Model:**

model.save("fruit\_classifier.h5")

**7. Creating a Flask Web Application**

**Setting Up Flask:**

from flask import Flask, request, jsonify, send\_file

from flask\_cors import CORS

app = Flask(\_\_name\_\_)

CORS(app)

**Enabling Cross-Origin Resource Sharing (CORS):**

CORS(app)

**Loading the Trained Model in Flask:**

from tensorflow.keras.models import load\_model

model = load\_model("./fruit\_classifier.h5")

**8. Handling Image Uploads**

**Defining the Image Preparation Function:**

def prepare\_image(img):

if img.mode != "RGB":

img = img.convert("RGB")

img = img.resize((100, 100))

img\_array = np.array(img) / 255.0

img\_array = np.expand\_dims(img\_array, axis=0)

return img\_array

**Writing the Image Upload Endpoint:**

@app.route("/upload", methods=["POST"])

def upload\_image():

if "file" not in request.files:

return jsonify({"error": "No file part"}), 400

file = request.files["file"]

if file.filename == "":

return jsonify({"error": "No selected file"}), 400

try:

if file:

img = Image.open(io.BytesIO(file.read()))

img\_array = prepare\_image(img)

predictions = model.predict(img\_array)

predicted\_class = np.argmax(predictions, axis=1)

class\_labels = ["apple", "banana", "beetroot", "bell pepper", "cabbage", "capsicum", "carrot", "cauliflower", "chilli pepper", "corn", "cucumber", "eggplant", "garlic", "ginger", "grapes", "jalepeno", "kiwi", "lemon", "lettuce", "mango", "onion", "orange", "paprika", "pear", "peas", "pineapple", "pomegranate", "potato", "raddish", "soy beans", "spinach", "sweetcorn", "sweetpotato", "tomato", "turnip", "watermelon"]

predicted\_label = class\_labels[predicted\_class[0]]

return jsonify({"fruit": predicted\_label}), 200

except Exception as e:

return jsonify({"error": "Error processing image"}), 500

return jsonify({"error": "Invalid file format"}), 400

**Handling Errors and Edge Cases:**

if "file" not in request.files:

return jsonify({"error": "No file part"}), 400

file = request.files["file"]

if file.filename == "":

return jsonify({"error": "No selected file"}), 400

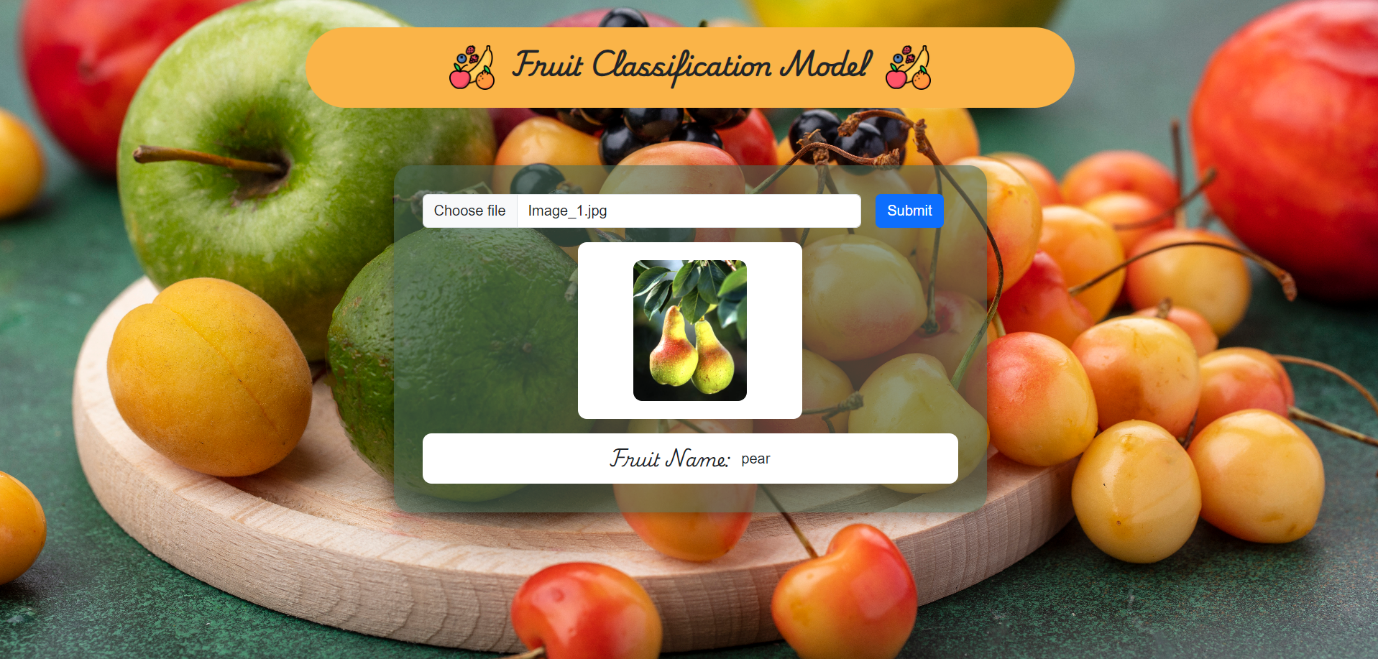
**9. Serving HTML Content**

**Creating and Serving the HTML File:**

@app.route("/")

def serve\_html():

return send\_file("./index.html")



**10. Running the Flask Application**

**Starting the Flask Server:**

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**Testing the API Endpoints:** Test the endpoints using tools like Postman or Curl.

**11. Deployment**

**Preparing for Deployment:** Ensure all dependencies are listed in requirements.txt.

pip freeze > requirements.txt

**Deploying to a Cloud Platform:** Follow the platform-specific instructions for deploying a Flask app.

**12. Conclusion**

**Summary of the Project:** The project successfully implements a CNN for fruit classification and deploys it as a web application.

**Future Work and Enhancements:**

* Improve model accuracy with more data and fine-tuning.
* Implement a more user-friendly front-end.
* Add support for more image formats.