#Python codes for Fruit Classification Model

#----------------------------

#connecting drive with colab

#----------------------------

from google.colab import drive

drive.mount('/content/drive')

# Follow the instructions for uploading drive and setting permissions.

#

#-------------------------------

import os

import numpy as np

from PIL import Image

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# -----------------------------------------

# STEP 1: SET PARAMETERS

# -----------------------------------------

data\_dir = "/content/drive/MyDrive/FruitsDataSet" # Path to your main dataset folder

!pwd

img\_height = 64

img\_width = 64

# Lists to store image data and labels

images = []

labels = []

# Optional: list subfolders (fruit classes) explicitly, or get them automatically

fruit\_classes = [

folder for folder in os.listdir(data\_dir)

if os.path.isdir(os.path.join(data\_dir, folder))

# and not folder.startswith('.') # uncomment if you have hidden/system folders

]

# -----------------------------------------

# STEP 2: LOAD & PREPROCESS IMAGES

# -----------------------------------------

for class\_name in fruit\_classes:

class\_folder = os.path.join(data\_dir, class\_name)

# Go through each image in the class folder

for img\_file in os.listdir(class\_folder):

# Only process valid image formats

if img\_file.lower().endswith(('.png', '.jpg', '.jpeg')):

img\_path = os.path.join(class\_folder, img\_file)

try:

# 1. Open the image

img = [Image.open](http://Image.open/)(img\_path)

# 2. Convert to RGB (3 channels) to ensure uniform shape

img = img.convert("RGB")

# 3. Resize to a consistent size (64x64)

img = img.resize((img\_width, img\_height))

# 4. Convert the image to a NumPy array

img\_array = np.array(img) # shape => (64, 64, 3)

# Store the processed image and the corresponding label

images.append(img\_array)

labels.append(class\_name)

except Exception as e:

print(f"Error loading {img\_path}: {e}")

# Convert the lists to NumPy arrays

images = np.array(images, dtype="float32") # shape => (num\_images, 64, 64, 3)

labels = np.array(labels)

print("Loaded images shape:", images.shape) # e.g. (NUM\_SAMPLES, 64, 64, 3)

print("Loaded labels shape:", labels.shape)

# -----------------------------------------

# STEP 3: NORMALIZE PIXEL VALUES [0,255] -> [0,1]

# -----------------------------------------

images = images / 255.0

# -----------------------------------------

# STEP 4: ENCODE LABELS (STRING TO INT)

# -----------------------------------------

unique\_labels = np.unique(labels)

label\_to\_idx = {label: idx for idx, label in enumerate(unique\_labels)}

encoded\_labels = np.array([label\_to\_idx[lbl] for lbl in labels])

# -----------------------------------------

# STEP 5: SPLIT INTO TRAIN & TEST

# -----------------------------------------

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

images,

encoded\_labels,

test\_size=0.2,

random\_state=42

)

print("X\_train shape:", X\_train.shape)

print("y\_train shape:", y\_train.shape)

print("X\_test shape:", X\_test.shape)

print("y\_test shape:", y\_test.shape)

# -----------------------------------------

# STEP 6: BUILD A SIMPLE CNN MODEL

# -----------------------------------------

num\_classes = len(unique\_labels)

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_height, img\_width, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dense(num\_classes, activation='softmax') # one output node per class

])

model.compile(

optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy']

)

model.summary()

# -----------------------------------------

# STEP 7: TRAIN THE MODEL

# -----------------------------------------

history = [model.fit](http://model.fit/)(

X\_train, y\_train,

epochs=10,

validation\_split=0.2,

batch\_size=32

)

# -----------------------------------------

# STEP 8: EVALUATE THE MODEL

# -----------------------------------------

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {test\_acc\*100:.2f}%")

# -----------------------------------------

# STEP 9: Plot

# -----------------------------------------

import matplotlib.pyplot as plt

# (Optional) Plot training results or test predictions

# Plot training & validation accuracy

plt.figure(figsize=(8, 5))

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

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

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

[plt.show](http://plt.show/)()

#------------------------

#STEP 10 Random Check

#------------------------

# Pick a random test image

rand\_index = random.randint(0, len(X\_test)-1)

sample\_image = X\_test[rand\_index]

actual\_label\_index = y\_test[rand\_index]

actual\_label = unique\_labels[actual\_label\_index]

# Predict

predictions = model.predict(np.expand\_dims(sample\_image, axis=0))

predicted\_label\_index = np.argmax(predictions)

predicted\_label = unique\_labels[predicted\_label\_index]

plt.imshow(sample\_image)

plt.title(f"Actual: {actual\_label}, Predicted: {predicted\_label}")

[plt.show](http://plt.show/)()