```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
test_dir = "/content/drive/MyDrive/FruitinAmazon/FruitinAmazon/test"
train_dir = "/content/drive/MyDrive/FruitinAmazon/FruitinAmazon/train"
```

```
import tensorflow as tf

# Define image size and batch size
img_height = 128  # Example image height
img_width = 128   # Example image width
batch_size = 32
validation_split = 0.2  # 80% training, 20% validation
```

```
rescale = tf.keras.layers.Rescaling(1./255)  # Normalize pixel values to [0, 1]
```

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    labels='inferred',
    label_mode='categorical',
    image_size=(img_height, img_width),
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=True,
    validation_split=validation_split,
    subset='training',
    seed=123
)

# Apply the normalization (Rescaling) to the dataset
train_ds = train_ds.map(lambda x, y: (rescale(x), y))

# Create validation dataset with normalization
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    labels='inferred',
    label_mode='categorical',
    image_size=(img_height, img_width),
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=False,
    validation_split=validation_split,
    subset='validation',
    seed=123
)

# Apply the normalization (Rescaling) to the validation dataset
val_ds = val_ds.map(lambda x, y: (rescale(x), y))
```

```
Found 90 files belonging to 6 classes.
Using 72 files for training.
Found 90 files belonging to 6 classes.
Using 18 files for validation.
```

```
model = tf.keras.Sequential([

    tf.keras.layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu', input_shape=(img_height, img_width, 3)),

    tf.keras.layers.MaxPooling2D((2, 2), strides=2),

    tf.keras.layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu'),

    tf.keras.layers.MaxPooling2D((2, 2), strides=2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation='relu'),
```

```
    tf.keras.layers.Dense(6, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

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

# Model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 128)	4,194,432
dense_1 (Dense)	(None, 6)	774

Total params: 4,205,350 (16.04 MB)

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint_callback = ModelCheckpoint(
    'best_model.h5',
    monitor='val_loss',
    save_best_only=True,
    mode='min',
    verbose=1
)

early_stopping_callback = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True,
    verbose=1
)


history =model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=20,
    batch_size=16,
    callbacks=[checkpoint_callback, early_stopping_callback]
)
```

Show hidden output

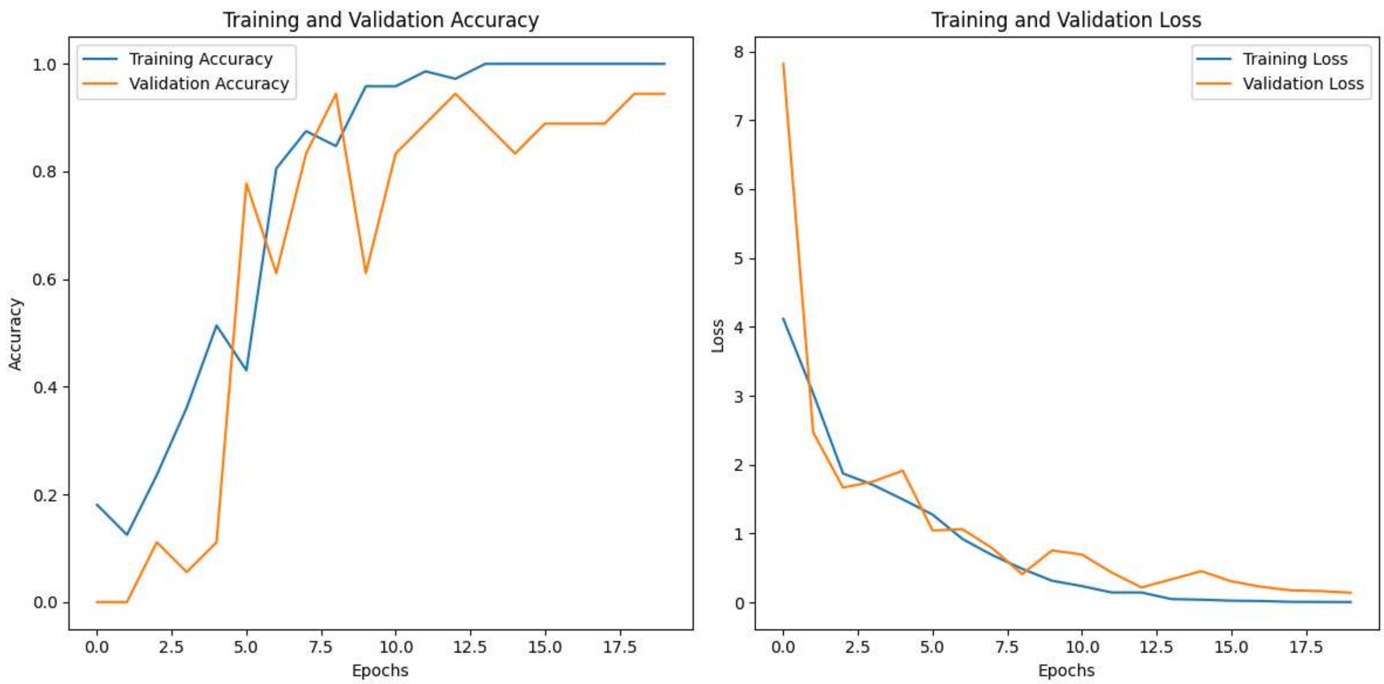
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))

# Plot training accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='best')

# Plot training and validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
```

```
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='best')

plt.tight_layout()
plt.show()
```




```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='categorical',
    image_size=(img_height, img_width),
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=False,
)

# Apply normalization (Rescaling) to the test dataset
test_ds = test_ds.map(lambda x, y: (rescale(x), y))

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_ds)
model.save("fruit_classification_model.h5")
# Print the evaluation results
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
Found 30 files belonging to 6 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - accuracy: 0.7667 - loss: 1.1949
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
Test Loss: 1.1949249505996704
Test Accuracy: 0.7666666507720947
```

```
import numpy as np
from sklearn.metrics import classification_report

# List of class labels (replace with your actual labels)
class_names = ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']

# Get predictions on the test set
predictions = model.predict(test_ds)
```

```
# Convert predicted probabilities to class labels
predicted_labels = np.argmax(predictions, axis=-1)

# Get true labels from the test dataset
true_labels = []
for _, labels in test_ds:
    # For categorical labels, labels are already one-hot encoded
    true_labels.extend(np.argmax(labels.numpy(), axis=-1))  # Convert one-hot encoded labels to class indices
true_labels = np.array(true_labels)

# Print classification report with actual class names
report = classification_report(true_labels, predicted_labels, target_names=class_names)
print(report)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
              precision    recall  f1-score   support

        acai       0.62      1.00      0.77         5
     cupuacu       0.80      0.80      0.80         5
    graviola       0.71      1.00      0.83         5
     guarana       1.00      0.40      0.57         5
     pupunha       0.83      1.00      0.91         5
      tucuma       1.00      0.40      0.57         5

    accuracy                           0.77        30
   macro avg       0.83      0.77      0.74        30
weighted avg       0.83      0.77      0.74        30
```

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing import image

# Define image size (same as during training)
img_height = 128  # Example image height
img_width = 128   # Example image width

# List of class labels (replace this with your actual class labels)
class_names = ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']

# Path to the test image
test_image_path = "/content/drive/MyDrive/FruitinAmazon/FruitinAmazon/test/tucuma/download (1).jpeg"

# Load the image and resize to the target size
img = image.load_img(test_image_path, target_size=(img_height, img_width))  # Use the same target size as during training

# Convert the image to a numpy array
img_array = image.img_to_array(img)

# Add batch dimension (since the model expects a batch of images)
img_array = np.expand_dims(img_array, axis=0)  # Shape becomes (1, img_height, img_width, 3)

# Apply the same rescaling (normalization) as in training
rescale = tf.keras.layers.Rescaling(1./255)  # Normalize pixel values to [0, 1]
img_array = rescale(img_array)  # Apply rescaling

# Make prediction using the model
predictions = model.predict(img_array)

# Convert predicted probabilities to class label
predicted_class_index = np.argmax(predictions, axis=-1)[0]  # Get the class with the highest probability
predicted_class_name = class_names[predicted_class_index]  # Map to the actual class name

print(f"The predicted class is: {predicted_class_name}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
The predicted class is: tucuma
```