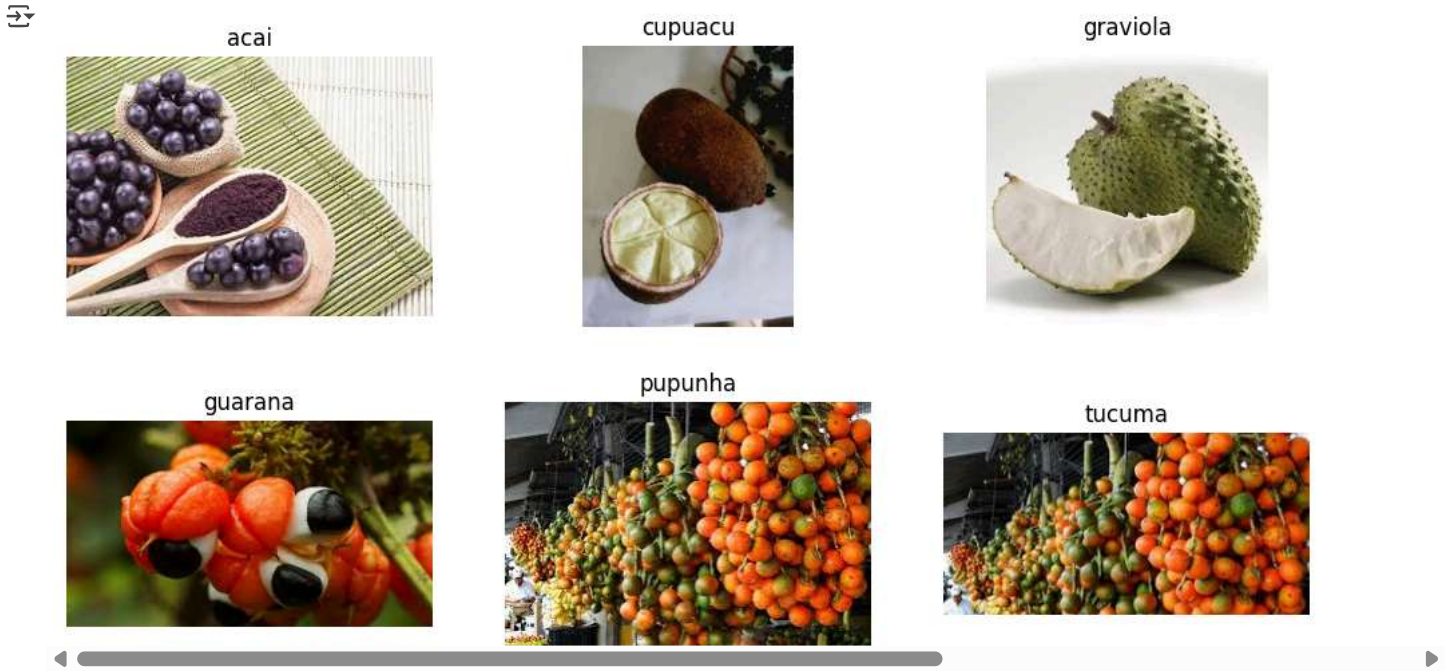
```
import os
import random
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Rescaling
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import classification_report


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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

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


class_names = os.listdir(train_dir)
print(f"Classes: {class_names}")
```

```
Classes: ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']
```

```
def visualize_images(train_dir, class_names):
    fig, axes = plt.subplots(2, len(class_names) // 2, figsize=(12, 6))
    axes = axes.flatten()
    for i, class_name in enumerate(class_names):
        class_path = os.path.join(train_dir, class_name)
        img_name = random.choice(os.listdir(class_path))
        img_path = os.path.join(class_path, img_name)
        img = load_img(img_path)
        axes[i].imshow(img)
        axes[i].set_title(class_name)
        axes[i].axis("off")
    plt.show()

visualize_images(train_dir, class_names)
```




```
damagedImages = []
for class_name in class_names:
    class_path = os.path.join(train_dir, class_name)
    for img_name in os.listdir(class_path):
        img_path = os.path.join(class_path, img_name)
        try:
```

```
            img = load_img(img_path)  # Try opening the image
        except (IOError, SyntaxError):
            damagedImages.append(img_path)
            os.remove(img_path)
            print(f"Damaged image removed: {img_path}")

if not damagedImages:
    print("No Damaged Images Found.")
```

```
No Damaged Images Found.
```

```
img_height, img_width = 128, 128
batch_size = 32
validation_split = 0.2


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

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

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=False,
    validation_split=validation_split,
    subset='validation',
    seed=123
)
```

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

```
rescale = Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (rescale(x), y))
val_ds = val_ds.map(lambda x, y: (rescale(x), y))


num_classes = len(class_names)

model = Sequential([
    Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(img_height, img_width, 3)),
    MaxPooling2D((2,2), strides=2),

    Conv2D(32, (3,3), activation='relu', padding='same'),
    MaxPooling2D((2,2), strides=2),

    Flatten(),
    Dense(64, activation='relu'),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')
])

model.summary()
```

```
/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: "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, 64)	2,097,216
dense_1 (Dense)	(None, 128)	8,320
dense_2 (Dense)	(None, 6)	774

Total params: 2,116,454 (8.07 MB)
Trainable params: 2,116,454 (8.07 MB)
Non-trainable params: 0 (0.00 B)

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


callbacks = [
    ModelCheckpoint("best_model.h5", save_best_only=True, monitor="val_accuracy", mode="max"),
    EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
]

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=250,
    batch_size=16,
    callbacks=callbacks
)
```

```
Epoch 8/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 436ms/step - accuracy: 0.4184 - loss: 1.3238 - val_accuracy: 0.3889 - val_loss: 1.4637
Epoch 9/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 450ms/step - accuracy: 0.5365 - loss: 1.1700 - val_accuracy: 0.7222 - val_loss: 1.0137
Epoch 10/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 453ms/step - accuracy: 0.7509 - loss: 1.0756 - val_accuracy: 0.1667 - val_loss: 1.5077
Epoch 11/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 529ms/step - accuracy: 0.6033 - loss: 1.0180 - val_accuracy: 0.1111 - val_loss: 1.7260
Epoch 12/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 797ms/step - accuracy: 0.8030 - loss: 0.7922 - val_accuracy: 0.6667 - val_loss: 0.7558
Epoch 13/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 629ms/step - accuracy: 0.8073 - loss: 0.6804 - val_accuracy: 0.7778 - val_loss: 0.9439
Epoch 14/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 446ms/step - accuracy: 0.8533 - loss: 0.5674 - val_accuracy: 0.7778 - val_loss: 0.9379
Epoch 15/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 356ms/step - accuracy: 0.7778 - loss: 0.5781WARNING:absl:You are saving your model as an HDF5 file via `m
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 511ms/step - accuracy: 0.7917 - loss: 0.5616 - val_accuracy: 0.8889 - val_loss: 0.5829
Epoch 16/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 538ms/step - accuracy: 0.9783 - loss: 0.3468 - val_accuracy: 0.8889 - val_loss: 0.4961
Epoch 17/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 517ms/step - accuracy: 0.9232 - loss: 0.3322 - val_accuracy: 0.7222 - val_loss: 0.7110
Epoch 18/250
```

```
Epoch 26/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 444ms/step - accuracy: 1.0000 - loss: 0.0088 - val_accuracy: 0.8889 - val_loss: 0.4564
Epoch 27/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 512ms/step - accuracy: 1.0000 - loss: 0.0365 - val_accuracy: 0.8889 - val_loss: 0.4286
Epoch 28/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 746ms/step - accuracy: 1.0000 - loss: 0.0078 - val_accuracy: 0.7778 - val_loss: 0.4642
Epoch 29/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 614ms/step - accuracy: 1.0000 - loss: 0.0102 - val_accuracy: 0.7778 - val_loss: 0.4571
Epoch 30/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 445ms/step - accuracy: 1.0000 - loss: 0.0068 - val_accuracy: 0.8889 - val_loss: 0.3800
Epoch 31/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 432ms/step - accuracy: 1.0000 - loss: 0.0034 - val_accuracy: 0.8889 - val_loss: 0.4029
Epoch 32/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 440ms/step - accuracy: 1.0000 - loss: 0.0068 - val_accuracy: 0.8889 - val_loss: 0.4392
Epoch 33/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 514ms/step - accuracy: 1.0000 - loss: 0.0060 - val_accuracy: 0.8889 - val_loss: 0.4088
Epoch 34/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 696ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 0.8333 - val_loss: 0.4067
Epoch 35/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 729ms/step - accuracy: 1.0000 - loss: 0.0032 - val_accuracy: 0.7778 - val_loss: 0.5035
```

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

test_ds = test_ds.map(lambda x, y: (rescale(x), y))
test_loss, test_accuracy = model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Found 30 files belonging to 6 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step - accuracy: 0.8333 - loss: 0.8369
Test Accuracy: 83.33%
```

```
model.save("final_model.h5")
loaded_model = tf.keras.models.load_model("final_model.h5")
```

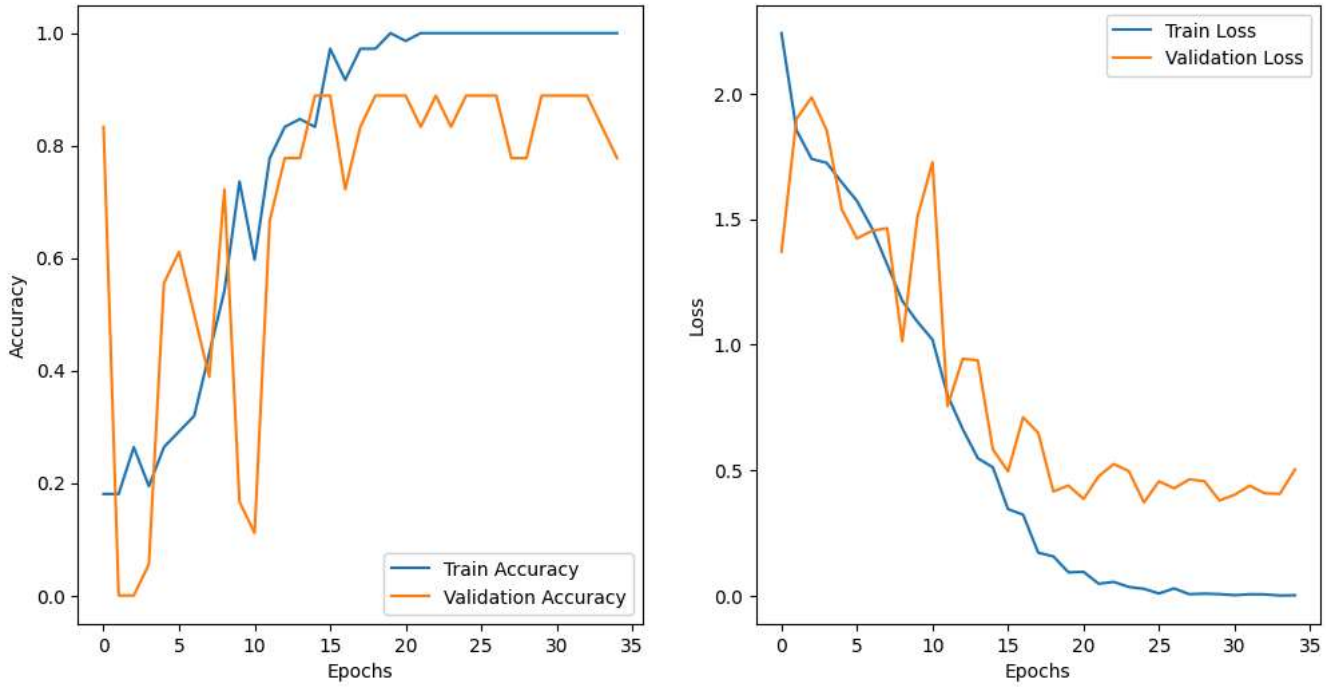
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
```

```
y_true = []
y_pred = []

for images, labels in test_ds:
    preds = loaded_model.predict(images)
    y_pred.extend(np.argmax(preds, axis=1))
    y_true.extend(labels.numpy())

print(classification_report(y_true, y_pred, target_names=class_names))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
              precision    recall  f1-score   support

        acai       0.71      1.00      0.83         5
     cupuacu       0.67      0.80      0.73         5
    graviola       0.83      1.00      0.91         5
     guarana       1.00      0.80      0.89         5
     pupunha       1.00      1.00      1.00         5
      tucuma       1.00      0.40      0.57         5

    accuracy                           0.83        30
   macro avg       0.87      0.83      0.82        30
weighted avg       0.87      0.83      0.82        30
```

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



Start coding or generate with AI.