Worksheet 6 output:

```
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, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import load_img, ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
from sklearn.metrics import classification_report
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D, Input
from tensorflow.keras.models import Model
```

```
[4] 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).
```

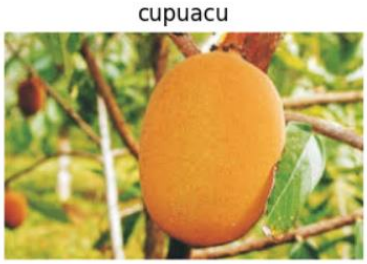
Task-1

```
train_dir = "/content/drive/MyDrive/Workshop5_AI/FruitinAmazon/train"
test_dir = "/content/drive/MyDrive/Workshop5_AI/FruitinAmazon/test"
```

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

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

```
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.
```

```
[9] img_height, img_width = 128, 128
    batch_size = 32
    validation_split = 0.2
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=validation_split,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2]
)

val_datagen = ImageDataGenerator(rescale=1./255, validation_split=validation_split)
```

```
[11] train_ds = train_datagen.flow_from_directory(
        train_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='sparse',
        subset='training',
        shuffle=True,
        seed=123
    )
```

```
Found 72 images belonging to 6 classes.
```

```
val_ds = val_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='sparse',
    subset='validation',
    shuffle=False,
    seed=123
)
```

```
Found 18 images belonging to 6 classes.
```

```
[13] num_classes = len(class_names)

    model = Sequential([
        Conv2D(32, (3,3), activation='relu', padding='same', kernel_regularizer=l2(0.001), input_shape=(img_height, img_width, 3)),
        BatchNormalization(),
        Conv2D(64, (3,3), activation='relu', padding='same', kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        MaxPooling2D((2,2)),
        Dropout(0.25),

        Conv2D(128, (3,3), activation='relu', padding='same', kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        MaxPooling2D((2,2)),
        Dropout(0.4),

        Flatten(),
        Dense(256, activation='relu', kernel_regularizer=l2(0.001)),
```

```
    BatchNormalization(),
    Dropout(0.5),
    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`/`input_dim` argument to a layer. When using Se
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	896
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
conv2d_1 (Conv2D)	(None, 128, 128, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 64)	256
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0
dropout (Dropout)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 64, 64, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 128)	0
dropout_1 (Dropout)	(None, 32, 32, 128)	0
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 256)	33,554,688
batch_normalization_3 (BatchNormalization)	(None, 256)	1,024
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 6)	1,542

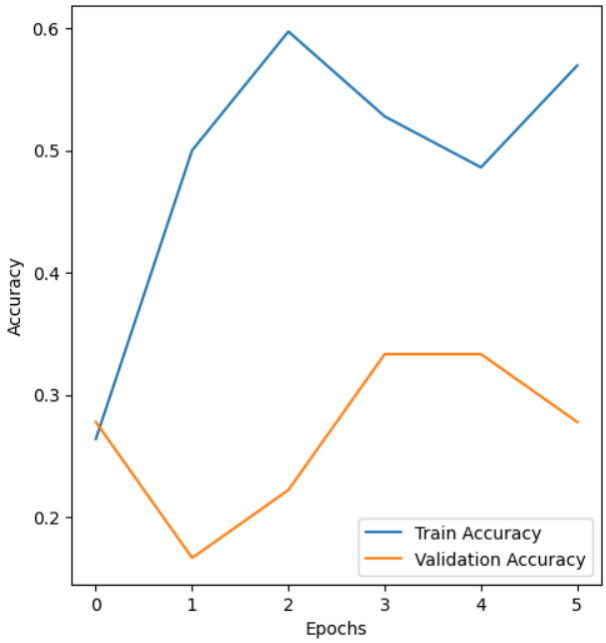
Total params: 33,651,398 (128.37 MB)
Trainable params: 33,650,438 (128.37 MB)
Non-trainable params: 960 (3.75 KB)

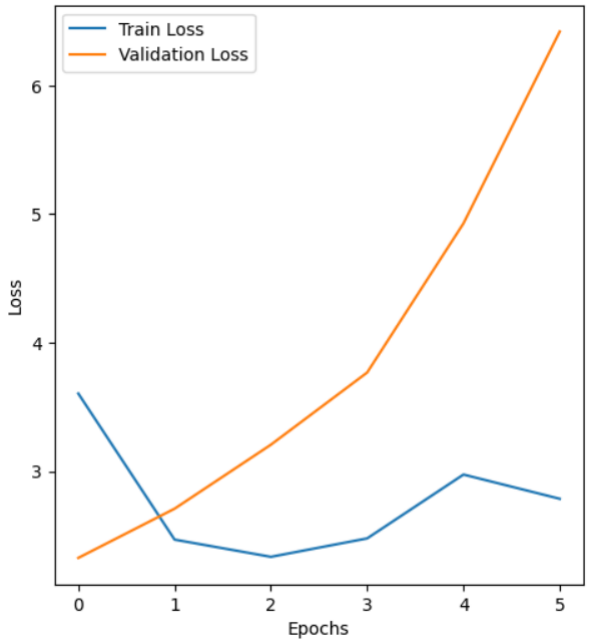
```
[14] model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

```
callbacks = [
    ModelCheckpoint("best_model.h5", save_best_only=True, monitor="val_accuracy", mode="max"),
    EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)
]

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=30,
    batch_size=16,
    callbacks=callbacks
)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)`
  self._warn_if_super_not_called()
Epoch 1/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.2171 - loss: 3.5672WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`
3/3 ━━━━━━━━━━━━━━━━━━━━ 29s 8s/step - accuracy: 0.2288 - loss: 3.5763 - val_accuracy: 0.2778 - val_loss: 2.3253 - learning_rate: 0.0010
Epoch 2/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 16s 7s/step - accuracy: 0.4938 - loss: 2.3590 - val_accuracy: 0.1667 - val_loss: 2.7081 - learning_rate: 0.0010
Epoch 3/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 16s 4s/step - accuracy: 0.6033 - loss: 2.3251 - val_accuracy: 0.2222 - val_loss: 3.2056 - learning_rate: 0.0010
Epoch 4/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.5676 - loss: 2.3569WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`
3/3 ━━━━━━━━━━━━━━━━━━━━ 21s 7s/step - accuracy: 0.5576 - loss: 2.3870 - val_accuracy: 0.3333 - val_loss: 3.7672 - learning_rate: 0.0010
Epoch 5/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 37s 4s/step - accuracy: 0.4868 - loss: 2.9064 - val_accuracy: 0.3333 - val_loss: 4.9269 - learning_rate: 5.0000e-04
Epoch 6/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 17s 4s/step - accuracy: 0.6128 - loss: 2.6247 - val_accuracy: 0.2778 - val_loss: 6.4192 - learning_rate: 5.0000e-04
```

```
[16] test_datagen = ImageDataGenerator(rescale=1./255)
     test_ds = test_datagen.flow_from_directory(
         test_dir,
         target_size=(img_height, img_width),
         batch_size=batch_size,
         class_mode='sparse',
         shuffle=False
     )
     test_loss, test_accuracy = model.evaluate(test_ds)
     print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

     Found 30 images belonging to 6 classes.
     1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.5000 - loss: 2.3028
     Test Accuracy: 50.00%

[17] 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 considered legacy. We recommend using instead
     WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

[18] y_true = test_ds.classes
     y_pred = np.argmax(loaded_model.predict(test_ds), axis=1)

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

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

          guarana       0.36      1.00      0.53         5
         graviola       0.00      0.00      0.00         5
             acai       0.40      0.40      0.40         5
          cupuacu       0.00      0.00      0.00         5
           tucuma       1.00      0.80      0.89         5
          pupunha       0.57      0.80      0.67         5

         accuracy                           0.50        30
        macro avg       0.39      0.50      0.41        30
     weighted avg       0.39      0.50      0.41        30

     /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicte
       _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
     /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicte
       _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
     /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicte
       _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

     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()
```





Task-2

```
base_model = MobileNetV2(input_shape=(img_height, img_width, 3), include_top=False, weights='imagenet')
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_128_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

+ Code + Text

```
inputs = Input(shape=(img_height, img_width, 3))
x = base_model(inputs, training=False)
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)
outputs = Dense(num_classes, activation='softmax')(x)
model = Model(inputs, outputs)

model.summary()
```

Model: "functional_16"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 128, 128, 3)	0
mobilenetv2_1.00_128 (Functional)	(None, 4, 4, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 128)	163,968
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 6)	774

Total params: 2,422,726 (9.24 MB)
Trainable params: 164,742 (643.52 KB)
Non-trainable params: 2,257,984 (8.61 MB)

+ Code + Text

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

```python
callbacks = [
    ModelCheckpoint("best_model_t1.h5", save_best_only=True, monitor="val_accuracy", mode="max"),
    EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)
]

# Train the model (only top layers)
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=30,
    callbacks=callbacks
)
```

```
Epoch 1/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 266ms/step - accuracy: 0.0793 - loss: 3.0859WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mod
3/3 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.0838 - loss: 3.0459 - val_accuracy: 0.4444 - val_loss: 1.4776 - learning_rate: 0.0010
Epoch 2/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 598ms/step - accuracy: 0.3889 - loss: 1.6125WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mod
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 902ms/step - accuracy: 0.3958 - loss: 1.6210 - val_accuracy: 0.5000 - val_loss: 1.0801 - learning_rate: 0.0010
Epoch 3/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 593ms/step - accuracy: 0.5852 - loss: 1.1856WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mod
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 894ms/step - accuracy: 0.5778 - loss: 1.1940 - val_accuracy: 0.7222 - val_loss: 0.9430 - learning_rate: 0.0010
Epoch 4/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 734ms/step - accuracy: 0.7000 - loss: 0.9217 - val_accuracy: 0.6667 - val_loss: 0.9452 - learning_rate: 0.0010
Epoch 5/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 378ms/step - accuracy: 0.6868 - loss: 0.7543WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mod
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 684ms/step - accuracy: 0.6818 - loss: 0.7629 - val_accuracy: 0.8889 - val_loss: 0.7323 - learning_rate: 0.0010
Epoch 6/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 442ms/step - accuracy: 0.7943 - loss: 0.6844 - val_accuracy: 0.8889 - val_loss: 0.5397 - learning_rate: 0.0010
```

```
Epoch 8/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 745ms/step - accuracy: 0.8215 - loss: 0.4730 - val_accuracy: 0.8333 - val_loss: 0.4606 - learning_rate: 0.0010
Epoch 9/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 439ms/step - accuracy: 0.8394 - loss: 0.3705 - val_accuracy: 0.8889 - val_loss: 0.4559 - learning_rate: 0.0010
Epoch 10/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 735ms/step - accuracy: 0.9069 - loss: 0.2700 - val_accuracy: 0.8889 - val_loss: 0.5043 - learning_rate: 0.0010
Epoch 11/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 504ms/step - accuracy: 0.8743 - loss: 0.2694 - val_accuracy: 0.8889 - val_loss: 0.5666 - learning_rate: 0.0010
Epoch 12/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 689ms/step - accuracy: 0.9510 - loss: 0.1962 - val_accuracy: 0.8889 - val_loss: 0.5913 - learning_rate: 0.0010
Epoch 13/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 895ms/step - accuracy: 0.9651 - loss: 0.1586 - val_accuracy: 0.8889 - val_loss: 0.5931 - learning_rate: 5.0000e-04
Epoch 14/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 761ms/step - accuracy: 0.9264 - loss: 0.2280 - val_accuracy: 0.8889 - val_loss: 0.5772 - learning_rate: 5.0000e-04
```

```python
test_loss, test_accuracy = model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 520ms/step - accuracy: 0.9000 - loss: 0.4587
Test Accuracy: 90.00%
```

+ Code + Text

```python
model.save("final_model_t1.h5")

loaded_model = tf.keras.models.load_model("final_model_t1.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 considered legacy. We recommend using instead
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

```python
y_true = test_ds.classes
y_pred_probs = loaded_model.predict(test_ds)
y_pred = np.argmax(y_pred_probs, axis=1)

print("Inference Output: First 20 Samples:")
for i in range(20):
    true_label = class_names[int(y_true[i])]
    pred_label = class_names[int(y_pred[i])]
    print(f"{i+1}. True: {true_label} - Predicted: {pred_label}")
```

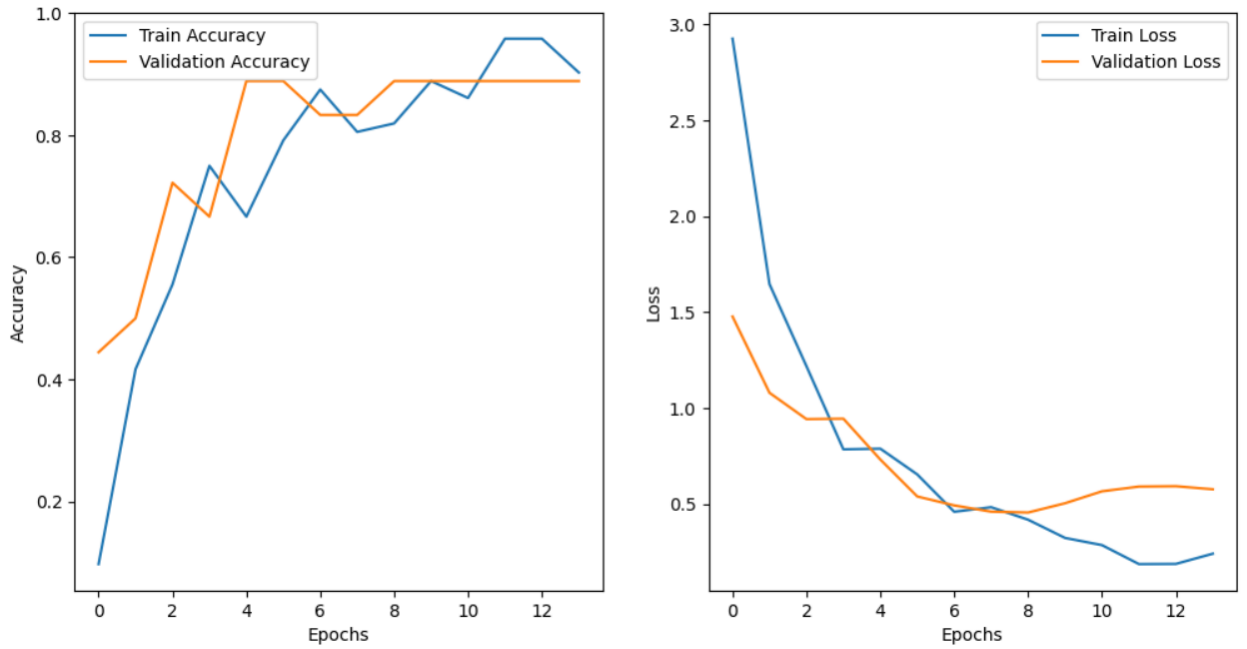
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Inference Output: First 20 Samples:
1. True: guarana - Predicted: guarana
2. True: guarana - Predicted: guarana
3. True: guarana - Predicted: acai
4. True: guarana - Predicted: tucuma
5. True: guarana - Predicted: guarana
6. True: graviola - Predicted: graviola
7. True: graviola - Predicted: graviola
8. True: graviola - Predicted: graviola
9. True: graviola - Predicted: graviola
10. True: graviola - Predicted: graviola
11. True: acai - Predicted: acai
12. True: acai - Predicted: acai
13. True: acai - Predicted: acai
14. True: acai - Predicted: acai
15. True: acai - Predicted: acai
16. True: cupuacu - Predicted: cupuacu
17. True: cupuacu - Predicted: cupuacu
18. True: cupuacu - Predicted: cupuacu
```

```
19. True: cupuacu - Predicted: cupuacu
20. True: cupuacu - Predicted: cupuacu
```

```
[ ] print("Classification Report:")
    print(classification_report(y_true, y_pred, target_names=class_names))
```

```
Classification Report:
              precision    recall  f1-score   support

      guarana       1.00      0.60      0.75         5
     graviola       1.00      1.00      1.00         5
         acai       0.83      1.00      0.91         5
      cupuacu       0.83      1.00      0.91         5
       tucuma       0.83      1.00      0.91         5
      pupunha       1.00      0.80      0.89         5

     accuracy                           0.90        30
    macro avg       0.92      0.90      0.89        30
 weighted avg       0.92      0.90      0.89        30
```

```
[28] 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()
```


Train Accuracy
Validation Accuracy
Accuracy
Epochs
Train Loss
Validation Loss
Loss
Epochs