Feature Extraction

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

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

```
import os
# Define dataset path
train_dir = "/content/drive/MyDrive/Final-Year
AI/week6/FruitinAmazon/train"
# Get class names (subdirectories)
class_names = sorted(os.listdir(train_dir))
if not class_names:
  print("No class directories found in the train folder!")
else:
  print(f"Found {len(class_names)} classes: {class_names}")
```

```
Found 6 classes: ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha',
'tucuma']
```

```
from PIL import Image, UnidentifiedImageError
corrupted_images = [] # List to store corrupted images path
# Loop through each class folder and check for corrupted images
for class_name in class_names:
  class_path = os.path.join(train_dir, class_name)
  if os.path.isdir(class_path): # Ensure it's a valid directory
    images = os.listdir(class_path)
    for img_name in images:
        img_path = os.path.join(class_path, img_name)
        try:
          with Image.open(img_path) as img:
            img.verify() # Verify image integrity
        except (IOError, UnidentifiedImageError):
            corrupted_images.append(img_path)

# Print results
if corrupted_images:
  print("\nCorrupted Images Found:")
  for img in corrupted_images:
      print(img)
else:
  print("\nNo corrupted images found.")
```

```
No corrupted images found.
```

```
# Dictionary to store class counts
class_counts = {}
```

```
for class_name in class_names:
    class_path = os.path.join(train_dir, class_name)
    if os.path.isdir(class_path):
        images = [img for img in os.listdir(class_path) if
img.lower().endswith(('.png', '.jpg', '.jpeg'))]
        class_counts[class_name] = len(images)  # Count images in each
class

# Print Class Balance
print("\nClass Distribution:")
print("=" * 45)
print(f"{'Class Name':<25}{'Valid Image Count':>15}")
print("=" * 45)
for class_name, count in class_counts.items():
    print(f"{class_name:<25}{count:>15}")
print("=" * 45)
```

```
Class Distribution:
=============================================
Class Name                Valid Image Count
=============================================
acai                                     15
cupuacu                                  15
graviola                                 15
guarana                                  15
pupunha                                  15
tucuma                                   15
=============================================
```

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
from sklearn.metrics import classification_report, confusion_matrix

# Set dataset path
train_dir = "/content/drive/MyDrive/Final-Year
AI/week6/FruitinAmazon/train"
test_dir = "/content/drive/MyDrive/Final-Year
AI/week6/FruitinAmazon/test"

# Image settings
img_size = (224, 224)  # Higher resolution
batch_size = 32
```

```
# Improved Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    zoom_range=0.3,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    brightness_range=[0.8, 1.2],
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Load Data
train_data = train_datagen.flow_from_directory(train_dir,
target_size=img_size, batch_size=batch_size, class_mode='categorical')
test_data = test_datagen.flow_from_directory(test_dir,
target_size=img_size, batch_size=batch_size, class_mode='categorical',
shuffle=False)

# Load Pretrained Model (MobileNetV2)
base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
base_model.trainable = False  # Freeze initial layers

# Define Model
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(train_data.class_indices),
activation='softmax')
])

# Compile Model
opt = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3)
```

```
# Train Model
history = model.fit(train_data, validation_data=test_data, epochs=70,
callbacks=[early_stopping, lr_scheduler])

# Save the trained model
model.save("/content/drive/MyDrive/Final-Year
AI/week6/fruit_classification_model_v2.h5")

# Load the saved model
loaded_model =
tf.keras.models.load_model("/content/drive/MyDrive/Final-Year
AI/week6/fruit_classification_model_v2.h5")

# Re-evaluate the model
y_true = test_data.classes
y_pred = np.argmax(loaded_model.predict(test_data), axis=1)
class_labels = list(test_data.class_indices.keys())

print("Re-evaluated Model Performance:")
print(classification_report(y_true, y_pred,
target_names=class_labels))

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels, fmt='d')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Plot Training Performance
plt.figure(figsize=(12,4))

# Accuracy Plot
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.title("Accuracy Curve")

# Loss Plot
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.title("Loss Curve")

plt.show()
```

```
Found 90 images belonging to 6 classes.
Found 30 images belonging to 6 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/mobilenet_v2/
mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

/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)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

Epoch 1/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 24s 7s/step - accuracy: 0.1576 - loss: 3.2465
- val_accuracy: 0.2000 - val_loss: 2.5893 - learning_rate: 1.0000e-04
Epoch 2/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.1927 - loss: 2.9962
- val_accuracy: 0.2000 - val_loss: 2.4425 - learning_rate: 1.0000e-04
Epoch 3/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 3s/step - accuracy: 0.3202 - loss: 2.6213
- val_accuracy: 0.2333 - val_loss: 2.2951 - learning_rate: 1.0000e-04
Epoch 4/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 4s/step - accuracy: 0.3252 - loss: 2.5511
- val_accuracy: 0.2667 - val_loss: 2.1560 - learning_rate: 1.0000e-04
Epoch 5/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.4636 - loss: 2.0591
- val_accuracy: 0.2667 - val_loss: 2.0275 - learning_rate: 1.0000e-04
Epoch 6/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.3452 - loss: 2.2257
- val_accuracy: 0.2667 - val_loss: 1.9106 - learning_rate: 1.0000e-04
Epoch 7/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.3614 - loss: 2.2059
- val_accuracy: 0.3000 - val_loss: 1.8011 - learning_rate: 1.0000e-04
Epoch 8/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 13s 4s/step - accuracy: 0.4435 - loss: 2.0387
- val_accuracy: 0.3333 - val_loss: 1.7017 - learning_rate: 1.0000e-04
Epoch 9/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 19s 3s/step - accuracy: 0.5641 - loss: 1.6340
- val_accuracy: 0.4000 - val_loss: 1.6088 - learning_rate: 1.0000e-04
Epoch 10/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.6098 - loss: 1.4344
- val_accuracy: 0.4000 - val_loss: 1.5295 - learning_rate: 1.0000e-04
```

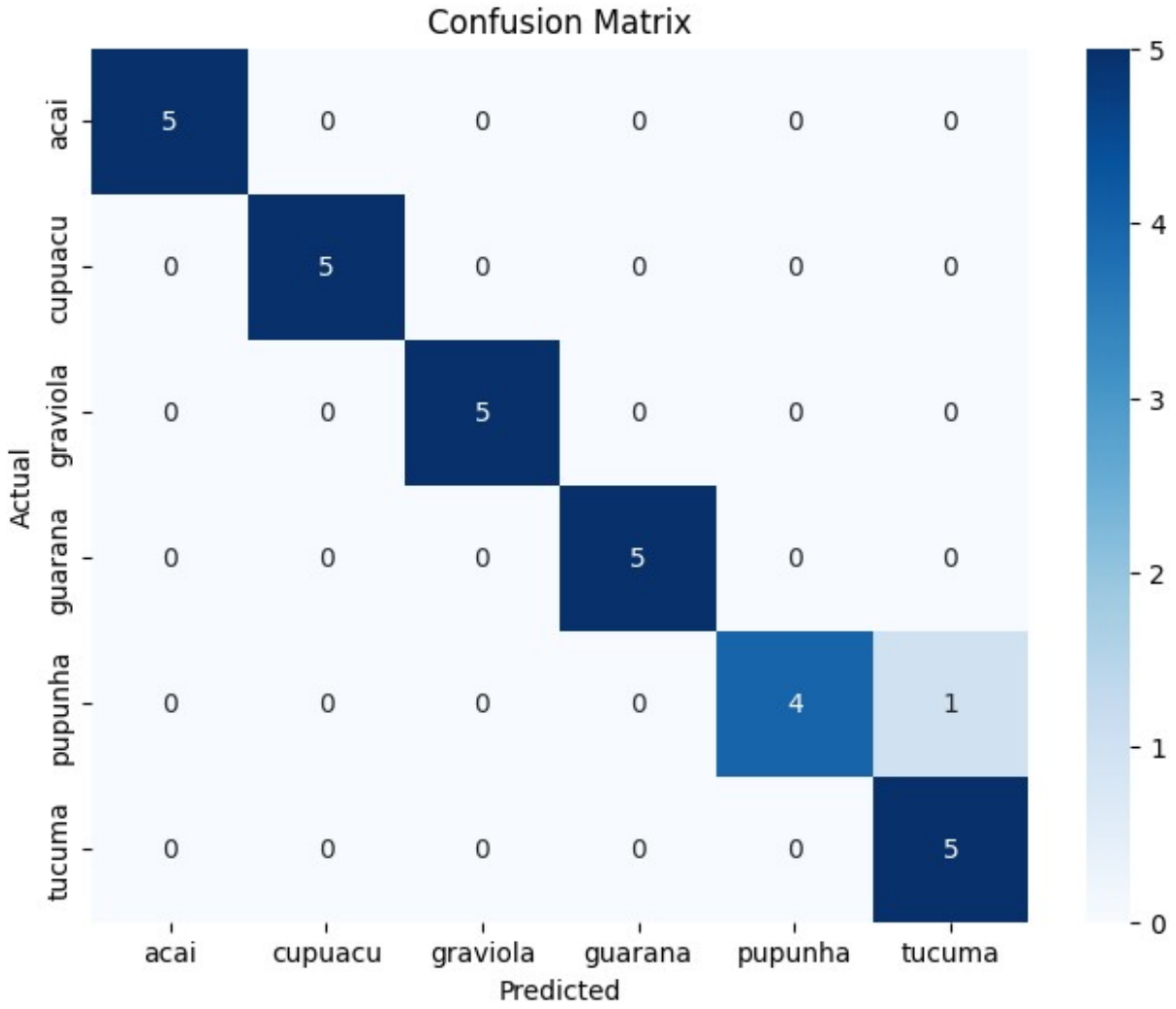
```
Epoch 11/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.6553 - loss: 1.4566
- val_accuracy: 0.4667 - val_loss: 1.4573 - learning_rate: 1.0000e-04
Epoch 12/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.7240 - loss: 1.1766
- val_accuracy: 0.5333 - val_loss: 1.3931 - learning_rate: 1.0000e-04
Epoch 13/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.6424 - loss: 1.5219
- val_accuracy: 0.7000 - val_loss: 1.3303 - learning_rate: 1.0000e-04
Epoch 14/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.6732 - loss: 1.3758
- val_accuracy: 0.7333 - val_loss: 1.2739 - learning_rate: 1.0000e-04
Epoch 15/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.7133 - loss: 1.4870
- val_accuracy: 0.7667 - val_loss: 1.2240 - learning_rate: 1.0000e-04
Epoch 16/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.8490 - loss: 0.9814
- val_accuracy: 0.8000 - val_loss: 1.1809 - learning_rate: 1.0000e-04
Epoch 17/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 3s/step - accuracy: 0.7845 - loss: 1.0657
- val_accuracy: 0.8000 - val_loss: 1.1408 - learning_rate: 1.0000e-04
Epoch 18/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.6809 - loss: 1.1347
- val_accuracy: 0.8667 - val_loss: 1.1053 - learning_rate: 1.0000e-04
Epoch 19/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.7695 - loss: 1.0225
- val_accuracy: 0.8667 - val_loss: 1.0740 - learning_rate: 1.0000e-04
Epoch 20/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.7812 - loss: 0.9667
- val_accuracy: 0.8667 - val_loss: 1.0466 - learning_rate: 1.0000e-04
Epoch 21/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.7873 - loss: 1.0597
- val_accuracy: 0.8667 - val_loss: 1.0205 - learning_rate: 1.0000e-04
Epoch 22/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 21s 9s/step - accuracy: 0.7655 - loss: 1.1665
- val_accuracy: 0.8667 - val_loss: 0.9955 - learning_rate: 1.0000e-04
Epoch 23/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.7880 - loss: 1.1417
- val_accuracy: 0.8667 - val_loss: 0.9746 - learning_rate: 1.0000e-04
Epoch 24/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8814 - loss: 0.8700
- val_accuracy: 0.8667 - val_loss: 0.9556 - learning_rate: 1.0000e-04
Epoch 25/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.8563 - loss: 0.8627
- val_accuracy: 0.9000 - val_loss: 0.9363 - learning_rate: 1.0000e-04
Epoch 26/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.8707 - loss: 0.8195
- val_accuracy: 0.9000 - val_loss: 0.9167 - learning_rate: 1.0000e-04
Epoch 27/70
```

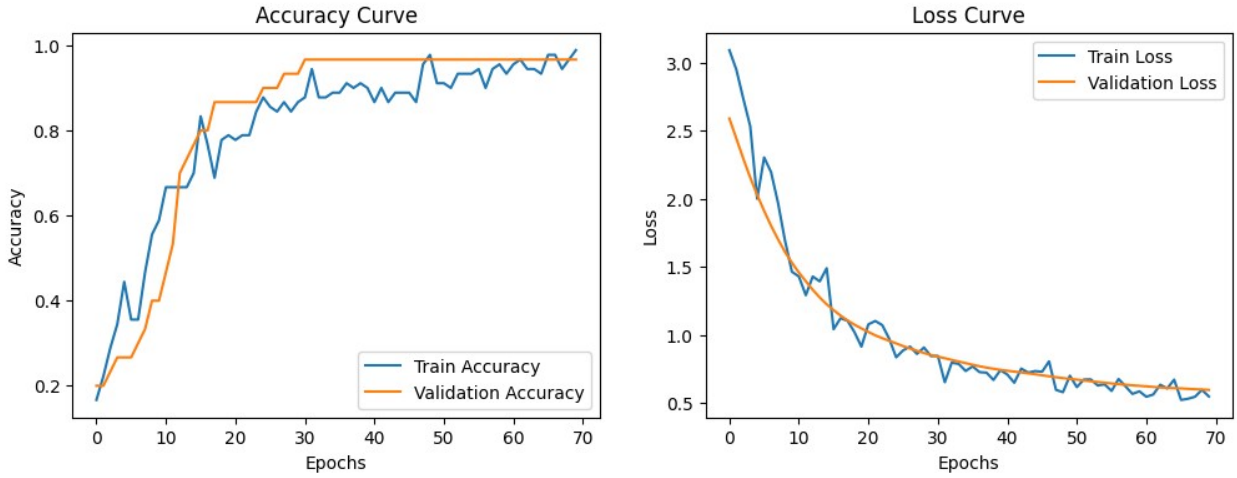
```
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step - accuracy: 0.8267 - loss: 0.9411
- val_accuracy: 0.9000 - val_loss: 0.8985 - learning_rate: 1.0000e-04
Epoch 28/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 3s/step - accuracy: 0.8422 - loss: 0.8735
- val_accuracy: 0.9333 - val_loss: 0.8825 - learning_rate: 1.0000e-04
Epoch 29/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8022 - loss: 0.9745
- val_accuracy: 0.9333 - val_loss: 0.8660 - learning_rate: 1.0000e-04
Epoch 30/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.8747 - loss: 0.8380
- val_accuracy: 0.9333 - val_loss: 0.8509 - learning_rate: 1.0000e-04
Epoch 31/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.8938 - loss: 0.8071
- val_accuracy: 0.9667 - val_loss: 0.8388 - learning_rate: 1.0000e-04
Epoch 32/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.9444 - loss: 0.6356
- val_accuracy: 0.9667 - val_loss: 0.8274 - learning_rate: 1.0000e-04
Epoch 33/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step - accuracy: 0.8703 - loss: 0.7900
- val_accuracy: 0.9667 - val_loss: 0.8150 - learning_rate: 1.0000e-04
Epoch 34/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step - accuracy: 0.8775 - loss: 0.8160
- val_accuracy: 0.9667 - val_loss: 0.8031 - learning_rate: 1.0000e-04
Epoch 35/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8908 - loss: 0.7369
- val_accuracy: 0.9667 - val_loss: 0.7912 - learning_rate: 1.0000e-04
Epoch 36/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.8830 - loss: 0.7909
- val_accuracy: 0.9667 - val_loss: 0.7795 - learning_rate: 1.0000e-04
Epoch 37/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.9008 - loss: 0.7214
- val_accuracy: 0.9667 - val_loss: 0.7683 - learning_rate: 1.0000e-04
Epoch 38/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.8730 - loss: 0.7600
- val_accuracy: 0.9667 - val_loss: 0.7583 - learning_rate: 1.0000e-04
Epoch 39/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.8912 - loss: 0.6922
- val_accuracy: 0.9667 - val_loss: 0.7499 - learning_rate: 1.0000e-04
Epoch 40/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.9039 - loss: 0.7016
- val_accuracy: 0.9667 - val_loss: 0.7417 - learning_rate: 1.0000e-04
Epoch 41/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.8508 - loss: 0.7408
- val_accuracy: 0.9667 - val_loss: 0.7353 - learning_rate: 1.0000e-04
Epoch 42/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.9085 - loss: 0.6646
- val_accuracy: 0.9667 - val_loss: 0.7288 - learning_rate: 1.0000e-04
Epoch 43/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.8312 - loss: 0.8093
```

```
- val_accuracy: 0.9667 - val_loss: 0.7229 - learning_rate: 1.0000e-04
Epoch 44/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.8865 - loss: 0.7537
- val_accuracy: 0.9667 - val_loss: 0.7161 - learning_rate: 1.0000e-04
Epoch 45/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.8859 - loss: 0.7446
- val_accuracy: 0.9667 - val_loss: 0.7093 - learning_rate: 1.0000e-04
Epoch 46/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.9093 - loss: 0.6935
- val_accuracy: 0.9667 - val_loss: 0.7022 - learning_rate: 1.0000e-04
Epoch 47/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 3s/step - accuracy: 0.8591 - loss: 0.8368
- val_accuracy: 0.9667 - val_loss: 0.6948 - learning_rate: 1.0000e-04
Epoch 48/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 4s/step - accuracy: 0.9648 - loss: 0.5800
- val_accuracy: 0.9667 - val_loss: 0.6878 - learning_rate: 1.0000e-04
Epoch 49/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.9772 - loss: 0.5806
- val_accuracy: 0.9667 - val_loss: 0.6811 - learning_rate: 1.0000e-04
Epoch 50/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.9201 - loss: 0.6930
- val_accuracy: 0.9667 - val_loss: 0.6755 - learning_rate: 1.0000e-04
Epoch 51/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.9244 - loss: 0.6052
- val_accuracy: 0.9667 - val_loss: 0.6702 - learning_rate: 1.0000e-04
Epoch 52/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8914 - loss: 0.7051
- val_accuracy: 0.9667 - val_loss: 0.6636 - learning_rate: 1.0000e-04
Epoch 53/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9315 - loss: 0.6542
- val_accuracy: 0.9667 - val_loss: 0.6582 - learning_rate: 1.0000e-04
Epoch 54/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9459 - loss: 0.6128
- val_accuracy: 0.9667 - val_loss: 0.6525 - learning_rate: 1.0000e-04
Epoch 55/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.9260 - loss: 0.6123
- val_accuracy: 0.9667 - val_loss: 0.6471 - learning_rate: 1.0000e-04
Epoch 56/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.9527 - loss: 0.5752
- val_accuracy: 0.9667 - val_loss: 0.6420 - learning_rate: 1.0000e-04
Epoch 57/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.9188 - loss: 0.6369
- val_accuracy: 0.9667 - val_loss: 0.6367 - learning_rate: 1.0000e-04
Epoch 58/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.9449 - loss: 0.6252
- val_accuracy: 0.9667 - val_loss: 0.6323 - learning_rate: 1.0000e-04
Epoch 59/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9692 - loss: 0.5445
- val_accuracy: 0.9667 - val_loss: 0.6283 - learning_rate: 1.0000e-04
```

```
Epoch 60/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.9163 - loss: 0.5957
- val_accuracy: 0.9667 - val_loss: 0.6243 - learning_rate: 1.0000e-04
Epoch 61/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.9492 - loss: 0.5571
- val_accuracy: 0.9667 - val_loss: 0.6207 - learning_rate: 1.0000e-04
Epoch 62/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step - accuracy: 0.9560 - loss: 0.5764
- val_accuracy: 0.9667 - val_loss: 0.6163 - learning_rate: 1.0000e-04
Epoch 63/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.9401 - loss: 0.6107
- val_accuracy: 0.9667 - val_loss: 0.6123 - learning_rate: 1.0000e-04
Epoch 64/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.9371 - loss: 0.6191
- val_accuracy: 0.9667 - val_loss: 0.6095 - learning_rate: 1.0000e-04
Epoch 65/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 11s 3s/step - accuracy: 0.9432 - loss: 0.6219
- val_accuracy: 0.9667 - val_loss: 0.6073 - learning_rate: 1.0000e-04
Epoch 66/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9772 - loss: 0.5162
- val_accuracy: 0.9667 - val_loss: 0.6055 - learning_rate: 1.0000e-04
Epoch 67/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.9707 - loss: 0.5387
- val_accuracy: 0.9667 - val_loss: 0.6025 - learning_rate: 1.0000e-04
Epoch 68/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.9437 - loss: 0.5455
- val_accuracy: 0.9667 - val_loss: 0.5997 - learning_rate: 1.0000e-04
Epoch 69/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.9638 - loss: 0.5931
- val_accuracy: 0.9667 - val_loss: 0.5974 - learning_rate: 1.0000e-04
Epoch 70/70
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9823 - loss: 0.5882
- val_accuracy: 0.9667 - val_loss: 0.5947 - learning_rate: 1.0000e-04

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 the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
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.

1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Re-evaluated Model Performance:
              precision    recall  f1-score   support

        acai       1.00      1.00      1.00         5
     cupuacu       1.00      1.00      1.00         5
    graviola       1.00      1.00      1.00         5
```

guarana	1.00	1.00	1.00	5
pupunha	1.00	0.80	0.89	5
tucuma	0.83	1.00	0.91	5
accuracy			0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30






```
# Task 2: Transfer Learning with MobileNetV2

import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense,
Dropout
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Set image size compatible with MobileNetV2
IMAGE_SIZE = (128, 128)
BATCH_SIZE = 32

# Prepare ImageDataGenerators
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Final-Year AI/week6/FruitinAmazon/train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    '/content/drive/MyDrive/Final-Year AI/week6/FruitinAmazon/test',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

# Load the base model
```

```
base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(128, 128, 3))

# Freeze all layers in the base model
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers on top
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(train_generator.num_classes, activation='softmax')
(x)

# Final model
model_tl = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model_tl.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history_tl = model_tl.fit(
    train_generator,
    validation_data=val_generator,
    epochs=5
)

# Evaluate model performance
val_loss_tl, val_acc_tl = model_tl.evaluate(val_generator)
print(f"Validation Accuracy: {val_acc_tl:.4f}")

# Generate predictions and classification report
y_true = val_generator.classes
y_pred_probs = model_tl.predict(val_generator)
y_pred = np.argmax(y_pred_probs, axis=1)

# Classification Report
class_labels = list(val_generator.class_indices.keys())
report = classification_report(y_true, y_pred,
target_names=class_labels)
print("\nClassification Report:\n")
print(report)

# Inference output (show first 10 predictions)
print("\nSample Inference Results:")
for i in range(10):
    print(f"True: {class_labels[y_true[i]]}, Predicted:
{class_labels[y_pred[i]]}")
```

```
Found 90 images belonging to 6 classes.
Found 30 images belonging to 6 classes.
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 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

/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)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

Epoch 1/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.1645 - loss: 2.4283
- val_accuracy: 0.6000 - val_loss: 1.4170
Epoch 2/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 601ms/step - accuracy: 0.4587 - loss:
1.5166 - val_accuracy: 0.7000 - val_loss: 1.0946
Epoch 3/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 664ms/step - accuracy: 0.6771 - loss:
0.8324 - val_accuracy: 0.7333 - val_loss: 0.8344
Epoch 4/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 688ms/step - accuracy: 0.8343 - loss:
0.5422 - val_accuracy: 0.8000 - val_loss: 0.7121
Epoch 5/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 625ms/step - accuracy: 0.8418 - loss:
0.4392 - val_accuracy: 0.8333 - val_loss: 0.6202
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 491ms/step - accuracy: 0.8333 - loss:
0.6202
Validation Accuracy: 0.8333
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

Classification Report:

              precision    recall  f1-score   support

        acai       0.67      0.80      0.73         5
     cupuacu       1.00      1.00      1.00         5
    graviola       1.00      1.00      1.00         5
     guarana       0.83      1.00      0.91         5
     pupunha       1.00      0.60      0.75         5
      tucuma       0.60      0.60      0.60         5

    accuracy                           0.83        30
   macro avg       0.85      0.83      0.83        30
weighted avg       0.85      0.83      0.83        30
```

```
Sample Inference Results:
True: acai, Predicted: acai
True: acai, Predicted: acai
True: acai, Predicted: acai
True: acai, Predicted: tucuma
True: acai, Predicted: acai
True: cupuacu, Predicted: cupuacu
True: cupuacu, Predicted: cupuacu
True: cupuacu, Predicted: cupuacu
True: cupuacu, Predicted: cupuacu
True: cupuacu, Predicted: cupuacu
```