1. Connecting Drive

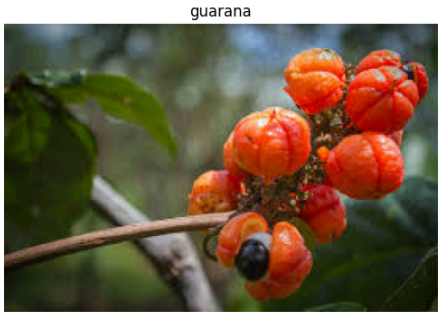
```
# prompt: connect drive

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

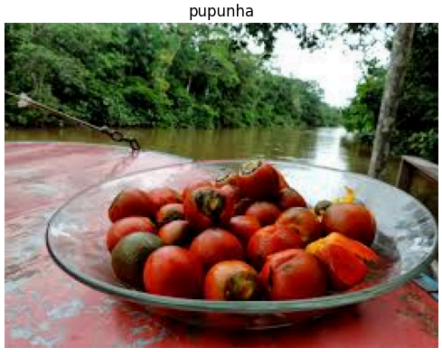
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

2. Imports and Setup

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image
import random
from sklearn.metrics import classification_report

# Set random seed for reproducibility
tf.random.set_seed(123)
np.random.seed(123)

# Define dataset paths
train_dir = "/content/drive/MyDrive/AI&ML-Level6/Week - 5/Week - 5 - Image Classification wi
test_dir = "/content/drive/MyDrive/AI&ML-Level6/Week - 5/Week - 5 - Image Classification wit


# Task 1 - Data Understanding and Visualization
train_dir = "/content/drive/MyDrive/AI&ML-Level6/Week - 5/Week - 5 - Image Classification wi

# Get class names
class_names = sorted(os.listdir(train_dir))

# Select one random image from each class
images = []
titles = []
for class_name in class_names[:6]:  # Assuming 6 classes as per sample report
    class_path = os.path.join(train_dir, class_name)
    if os.path.isdir(class_path):
        image_files = os.listdir(class_path)
        if image_files:
            random_image = random.choice(image_files)
            image_path = os.path.join(class_path, random_image)
            images.append(Image.open(image_path))
```

```
                titles.append(class_name)

    # Display images in 2x3 grid
    plt.figure(figsize=(15, 10))
    for i in range(len(images)):
        plt.subplot(2, 3, i+1)
        plt.imshow(images[i])
        plt.title(titles[i])
        plt.axis('off')
    plt.tight_layout()
    plt.show()

    print("Observation: The dataset contains various fruit images from Amazon, each representing
```

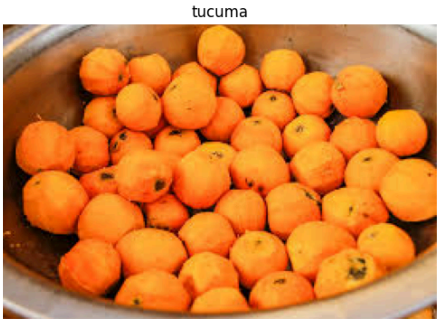




Observation: The dataset contains various fruit images from Amazon, each representing a

Task 1 - Data Understanding and Visualization

```
# Get class names
class_names = sorted(os.listdir(train_dir))

# Select one random image from each class
images = []
titles = []
for class_name in class_names:
    class_path = os.path.join(train_dir, class_name)
    if os.path.isdir(class_path):
        image_files = os.listdir(class_path)
        if image_files:
            random_image = random.choice(image_files)
            image_path = os.path.join(class_path, random_image)
            images.append(Image.open(image_path))
            titles.append(class_name)

# Display images in 2x3 grid
plt.figure(figsize=(15, 10))
for i in range(len(images)):
    plt.subplot(2, 3, i+1)
    plt.imshow(images[i])
    plt.title(titles[i])
    plt.axis('off')
plt.tight_layout()
plt.show()

print("Observation: The dataset contains images of six Amazonian fruits: acai, cupuacu, grav
```



Observation: The dataset contains images of six Amazonian fruits: acai, cupuacu, graviol

Check for Corrupted Images

```
def check_corrupted_images(directory):
    corrupted_images = []

    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path):
            for image_file in os.listdir(class_path):
                image_path = os.path.join(class_path, image_file)
                try:
                    img = Image.open(image_path)
                    img.verify()  # Verify image integrity
                except (IOError, SyntaxError) as e:
                    corrupted_images.append(image_path)
                    print(f"Removed corrupted image: {image_path}")

    if not corrupted_images:
        print("No corrupted images found.")
    return corrupted_images

corrupted = check_corrupted_images(train_dir)
```

```
No corrupted images found.
```

Task 2 - Loading and Preprocessing Data

```
img_height = 128
img_width = 128
batch_size = 16
validation_split = 0.2

# Create preprocessing layer
rescale = tf.keras.layers.Rescaling(1./255)

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

```
# Load validation dataset
val_ds_raw = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=False,
    validation_split=validation_split,
    subset='validation',
    seed=123
)

# Load test dataset
test_ds_raw = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=False
)

# Get number of classes and class names before applying map
num_classes = len(train_ds_raw.class_names)
class_names = train_ds_raw.class_names
print(f"Number of classes: {num_classes}")
print(f"Class names: {class_names}")

# Apply normalization
train_ds = train_ds_raw.map(lambda x, y: (rescale(x), y))
val_ds = val_ds_raw.map(lambda x, y: (rescale(x), y))
test_ds = test_ds_raw.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.
Found 30 files belonging to 6 classes.
Number of classes: 6
Class names: ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']
```

Task 3 - Implement CNN Model

```
model = keras.Sequential([
    # Convolutional Layer 1
    layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu',
```

```
                input_shape=(img_height, img_width, 3)),
    layers.MaxPooling2D((2, 2), strides=2),

    # Convolutional Layer 2
    layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu'),
    layers.MaxPooling2D((2, 2), strides=2),

    # Fully Connected Layers
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

# Display model summary
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_7 (Conv2D)	(None, 64, 64, 32)	9,248
max_pooling2d_7 (MaxPooling2D)	(None, 32, 32, 32)	0
flatten_3 (Flatten)	(None, 32768)	0
dense_9 (Dense)	(None, 64)	2,097,216
dense_10 (Dense)	(None, 128)	8,320
dense_11 (Dense)	(None, 6)	774

Total params: 2,116,454 (8.07 MB)

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

Task 4 - Train Model

```
# Define callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint(
        'best_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        mode='max'
    ),
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=20,
        restore_best_weights=True
    )
]

# Train the model
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=50,
    batch_size=16,
    callbacks=callbacks
)
```

```
Epoch 3/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 341ms/step - accuracy: 0.5367 - loss: 1.3554 - val_accura
Epoch 4/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step - accuracy: 0.7215 - loss: 0.9653WARNING:absl:
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 363ms/step - accuracy: 0.7147 - loss: 0.9674 - val_accura
Epoch 5/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 258ms/step - accuracy: 0.7094 - loss: 0.7346WARNING:absl:
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 332ms/step - accuracy: 0.7092 - loss: 0.7342 - val_accura
Epoch 6/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 342ms/step - accuracy: 0.8602 - loss: 0.4293 - val_accura
Epoch 7/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 308ms/step - accuracy: 0.8972 - loss: 0.2842WARNING:absl:
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 424ms/step - accuracy: 0.9051 - loss: 0.2793 - val_accura
Epoch 8/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 538ms/step - accuracy: 0.9578 - loss: 0.1519 - val_accura
Epoch 9/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 317ms/step - accuracy: 0.9954 - loss: 0.0611 - val_accura
Epoch 10/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 310ms/step - accuracy: 0.9786 - loss: 0.0690 - val_accura
Epoch 11/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 390ms/step - accuracy: 1.0000 - loss: 0.0236 - val_accura
Epoch 12/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 343ms/step - accuracy: 0.9737 - loss: 0.0447 - val_accura
Epoch 13/50
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 331ms/step - accuracy: 1.0000 - loss: 0.0055 - val_accura
Epoch 17/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 310ms/step - accuracy: 1.0000 - loss: 0.0056 - val_accura
Epoch 18/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 309ms/step - accuracy: 1.0000 - loss: 0.0023 - val_accura
Epoch 19/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 341ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accura
Epoch 20/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 521ms/step - accuracy: 1.0000 - loss: 0.0010 - val_accura
Epoch 21/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 339ms/step - accuracy: 1.0000 - loss: 7.7969e-04 - val_ac
Epoch 22/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 339ms/step - accuracy: 1.0000 - loss: 8.1992e-04 - val_ac
Epoch 23/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 343ms/step - accuracy: 1.0000 - loss: 5.6080e-04 - val_ac
Epoch 24/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 309ms/step - accuracy: 1.0000 - loss: 4.8817e-04 - val_ac
Epoch 25/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 318ms/step - accuracy: 1.0000 - loss: 4.7920e-04 - val_ac
Epoch 26/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 527ms/step - accuracy: 1.0000 - loss: 4.5928e-04 - val_ac
Epoch 27/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 323ms/step - accuracy: 1.0000 - loss: 4.2036e-04 - val_ac
Epoch 28/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 310ms/step - accuracy: 1.0000 - loss: 3.5722e-04 - val_ac
Epoch 29/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 321ms/step - accuracy: 1.0000 - loss: 3.3024e-04 - val_ac
```

Task 5 - Evaluate Model

```
# Evaluate on test set
test_loss, test_acc = model.evaluate(test_ds)
print(f"Test accuracy: {test_acc:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 0.7375 - loss: 0.7328
Test accuracy: 0.7000
Test loss: 0.8313
```

Task 6 - Save and Load Model

```
# Save the model
model.save('final_model.h5')

# Load the model
loaded_model = keras.models.load_model('final_model.h5')

# Re-evaluate loaded model
loaded_test_loss, loaded_test_acc = loaded_model.evaluate(test_ds
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.savi
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 0.7375 - loss: 0.7328
Loaded model test accuracy: 0.7000
Loaded model test loss: 0.8313
```

Task 7 - Predictions, Classification Report, and Visualization

```
# Get predictions for the entire test set
all_predictions = []
all_labels = []

for images, labels in test_ds:
    predictions = model.predict(images)
    predicted_labels = np.argmax(predictions, axis=1)
    all_predictions.extend(predicted_labels)
    all_labels.extend(labels.numpy())

# Generate classification report
print("Classification Report:")
print(classification_report(all_labels, all_predictions,
                           target_names=class_names))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
Classification Report:
              precision    recall  f1-score   support

        acai       0.56      1.00      0.71         5
     cupuacu       0.83      1.00      0.91         5
    graviola       1.00      0.60      0.75         5
     guarana       0.67      0.40      0.50         5
     pupunha       0.75      0.60      0.67         5
      tucuma       0.60      0.60      0.60         5

    accuracy                           0.70        30
   macro avg       0.73      0.70      0.69        30
weighted avg       0.73      0.70      0.69        30
```

Visualization of Training Metrics

```
# Plot training history
plt.figure(figsize=(12, 4))

# Plot accuracy
plt.subplot(1, 2, 1)
```