Task 1: Data Understanding and Visualization:

```
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).
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import load_model
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from PIL import Image
```

1. Load and visualize images from a dataset stored in directories, where each subdirec-

tory represents a class.

```
# Training and testing directory
train_dir = "/content/drive/MyDrive/Level 6/AI & ML/w5/FruitinAmazon/train"
test_dir = "/content/drive/MyDrive/Level 6/AI & ML/w5/FruitinAmazon/test"

img_height, img_width = 128, 128  # Increased resolution

def load_images_from_directory(directory):
    images = []
    labels = []
    class_names = sorted(os.listdir(directory))  # Ensure consistent label order
```

```
        class_dict = {class_name: idx for idx, class_name in enumerate(class_names)}

        for class_name in class_names:
            class_path = os.path.join(directory, class_name)
            if not os.path.isdir(class_path):
                continue

            for img_name in os.listdir(class_path):
                img_path = os.path.join(class_path, img_name)
                try:
                    img = Image.open(img_path)
                    img = img.resize((img_width, img_height), Image.LANCZOS)  # LANCZOS for sharper resizing
                    images.append(np.array(img))
                    labels.append(class_dict[class_name])
                except Exception as e:
                    print(f"Error loading image {img_path}: {e}")

        return np.array(images), np.array(labels), class_names

    # Load training images
    X, y, class_names = load_images_from_directory(train_dir)

    # Normalize pixel values to [0,1]
    X = X / 255.0

    # Convert labels to categorical
    y = to_categorical(y, num_classes=len(class_names))

    # Split data into training and validation sets
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    # Display some sample images
    def display_sample_images(X, y, class_names, rows=2, cols=5):
        fig, axes = plt.subplots(rows, cols, figsize=(10, 5))
        axes = axes.flatten()

        for i in range(rows * cols):
            idx = np.random.randint(len(X))
            axes[i].imshow(X[idx], interpolation='nearest')  # Ensure sharp display
            axes[i].set_title(class_names[np.argmax(y[idx])])
```

```
        axes[i].axis('off')

    plt.tight_layout()
    plt.show()

# Display sample images from training set
display_sample_images(X_train, y_train, class_names)
```

2. Check for Corrupted Image:

```
import os
from PIL import Image
```

```
# Training directory
train_dir = "/content/drive/MyDrive/Level 6/AI & ML/w5/FruitinAmazon/train"

def remove_corrupted_images(directory):
    corrupted_images = []

    # Iterate through each class subdirectory
    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if not os.path.isdir(class_path):
            continue

        # Iterate through each image in the class subdirectory
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)

            try:
                # Attempt to open the image
                img = Image.open(img_path)
                img.verify()  # Verify the image is valid
            except (IOError, SyntaxError) as e:
                # If an error occurs, it's a corrupted image
                corrupted_images.append(img_path)
                os.remove(img_path)  # Remove corrupted image
                print(f"Removed corrupted image: {img_path}")

    # Report if no corrupted images were found
    if not corrupted_images:
        print("No corrupted images found.")

# Call the function to check and remove corrupted images
remove_corrupted_images(train_dir)
```

```
No corrupted images found.
```

Task 2: Loading and Preprocessing Image Data in keras:

```
# Define image size and batch size
img_height = 128
img_width = 128
batch_size = 32
validation_split=0.2 #80% training , 20% validation
# Create preprocessing layer for normalization
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='int',
    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='int',
    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.
```

Task 3 - Implement a CNN with

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam

# Define the CNN + Fully Connected Network model
model = Sequential()

# Convolutional Layer 1
model.add(Conv2D(32, (3, 3), padding='same', strides=1, activation='relu', input_shape=(128, 128, 3)))

# Max Pooling Layer 1
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))

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

# Max Pooling Layer 2
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))

# Flatten the output from the convolutional layers
model.add(Flatten())

# Hidden Layer 1 - 64 neurons
model.add(Dense(64, activation='relu'))
```

```
# Hidden Layer 2 - 128 neurons
model.add(Dense(128, activation='relu'))

# Output Layer (Number of classes = len(class_names))
model.add(Dense(len(class_names), activation='softmax'))

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

# Model Summary
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)

Explanation of the Layers: Convolutional Layers (Conv2D) and Max Pooling Layers (MaxPooling2D): These layers are the same as in the previous CNN model. They extract features from the image and reduce spatial dimensions.

Flatten Layer:

The Flatten() layer reshapes the output from the convolutional layers into a 1D vector that can be passed to the fully connected layers.

Hidden Layers:

Dense Layer 1: Has 64 neurons, with ReLU activation. This layer learns the relationships between the features extracted by the convolutional layers.

Dense Layer 2: Has 128 neurons, also with ReLU activation. This further processes the features learned in the first hidden layer.

Output Layer:

The number of neurons is equal to the number of classes (i.e., len(class_names)).

Softmax activation is used for multi-class classification, where the model outputs probabilities for each class.

Model Compilation: Optimizer: Adam optimizer is used for gradient descent.

Loss function: categorical_crossentropy is used for multi-class classification.

Metrics: Accuracy is used to evaluate the model's performance.

Task 4: Compile the Model

```
# Compile the model
model.compile(
    optimizer='adam',  # Adam optimizer
    loss='sparse_categorical_crossentropy',  # Use 'categorical_crossentropy' if labels are one-hot encoded
    metrics=['accuracy']  # Accuracy metric
)
```

Task 4: Train the Model

```
# Define callbacks
# ModelCheckpoint: Save the best model based on validation accuracy
checkpoint_callback = ModelCheckpoint(
    'best_model.h5',  # File path to save the best model
    monitor='val_loss',  # Monitor validation loss (could also use 'val_accuracy')
    save_best_only=True,  # Save only the best model
    mode='min',  # Minimize the validation loss
    verbose=1  # Print a message when the model is saved
)

# EarlyStopping: Stop training if validation loss doesn't improve for a given number of epochs
early_stopping_callback = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=10,  # Stop after 10 epochs with no improvement
    restore_best_weights=True,  # Restore the weights of the best model
    verbose=1  # Print a message when training stops
)

# Train the model using model.fit() with callbacks
history = model.fit(
    X_train,  # Training data
    y_train,  # Training labels
    epochs=250,  # Number of epochs
    batch_size=16,  # Batch size
    validation_data=(X_val, y_val),  # Validation data
    callbacks=[checkpoint_callback, early_stopping_callback]  # Callbacks for saving the best model and early stopping
)
```

```
Epoch 1/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step - accuracy: 0.9229 - loss: 0.4336
Epoch 1: val_loss improved from inf to 1.25242, saving model to best_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
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 331ms/step - accuracy: 0.9219 - loss: 0.4412 - val_accuracy: 0.4444 - val_loss: 1.2524
Epoch 2/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step - accuracy: 0.8556 - loss: 0.4137
Epoch 2: val_loss did not improve from 1.25242
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 307ms/step - accuracy: 0.8565 - loss: 0.4065 - val_accuracy: 0.5556 - val_loss: 1.2639
Epoch 3/250
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step - accuracy: 0.9580 - loss: 0.2650
Epoch 3: val_loss improved from 1.25242 to 1.11293, saving model to best_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
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 330ms/step - accuracy: 0.9534 - loss: 0.2711 - val_accuracy: 0.5556 - val_loss: 1.1129
Epoch 4/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 459ms/step - accuracy: 0.9382 - loss: 0.1844
Epoch 4: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 556ms/step - accuracy: 0.9369 - loss: 0.1864 - val_accuracy: 0.3333 - val_loss: 1.1843
Epoch 5/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step - accuracy: 0.9899 - loss: 0.1281
Epoch 5: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 301ms/step - accuracy: 0.9893 - loss: 0.1284 - val_accuracy: 0.5000 - val_loss: 1.4356
Epoch 6/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step - accuracy: 1.0000 - loss: 0.0720
Epoch 6: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 287ms/step - accuracy: 1.0000 - loss: 0.0683 - val_accuracy: 0.3889 - val_loss: 1.3739
Epoch 7/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 273ms/step - accuracy: 1.0000 - loss: 0.0175
Epoch 7: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 313ms/step - accuracy: 1.0000 - loss: 0.0182 - val_accuracy: 0.4444 - val_loss: 1.5099
Epoch 8/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step - accuracy: 1.0000 - loss: 0.0132
Epoch 8: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 301ms/step - accuracy: 1.0000 - loss: 0.0136 - val_accuracy: 0.5000 - val_loss: 1.6705
Epoch 9/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 440ms/step - accuracy: 1.0000 - loss: 0.0075
Epoch 9: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 531ms/step - accuracy: 1.0000 - loss: 0.0076 - val_accuracy: 0.5556 - val_loss: 1.6649
Epoch 10/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 433ms/step - accuracy: 1.0000 - loss: 0.0176
Epoch 10: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 527ms/step - accuracy: 1.0000 - loss: 0.0166 - val_accuracy: 0.6111 - val_loss: 1.4880
Epoch 11/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step - accuracy: 1.0000 - loss: 0.0036
Epoch 11: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 297ms/step - accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 0.5000 - val_loss: 1.6830
Epoch 12/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step - accuracy: 1.0000 - loss: 0.0021
Epoch 12: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 292ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.4444 - val_loss: 1.6431
Epoch 13/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step - accuracy: 1.0000 - loss: 0.0020
Epoch 13: val_loss did not improve from 1.11293
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 280ms/step - accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 0.5000 - val_loss: 1.6917
```

```
Epoch 13: early stopping
Restoring model weights from the end of the best epoch: 3.
```

```
# Remove one-hot encoding (to_categorical)
X, y, class_names = load_images_from_directory(train_dir)

# Normalize pixel values to [0,1]
X = X / 255.0

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Compilation using sparse_categorical_crossentropy
model.compile(
    optimizer='adam',  # Adam optimizer
    loss='sparse_categorical_crossentropy',  # For integer labels
    metrics=['accuracy']  # Accuracy metric
)

# Define callbacks
checkpoint_callback = ModelCheckpoint(
    'best_model.h5',  # File path to save the best model
    monitor='val_loss',  # Monitor validation loss
    save_best_only=True,  # Save only the best model
    mode='min',  # Minimize the validation loss
    verbose=1  # Print a message when the model is saved
)

early_stopping_callback = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=10,  # Stop after 10 epochs with no improvement
    restore_best_weights=True,  # Restore the weights of the best model
    verbose=1  # Print a message when training stops
)

# Train the model using model.fit() with callbacks
history = model.fit(
    X_train,  # Training data
    y_train,  # Training labels
```

```
    y_train,  # Training labels
    epochs=250,  # Number of epochs
    batch_size=16,  # Batch size
    validation_data=(X_val, y_val),  # Validation data
    callbacks=[checkpoint_callback, early_stopping_callback]  # Callbacks for saving the best model and early stopping
)
```

```
Epoch 1/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 284ms/step - accuracy: 0.1979 - loss: 2.1042
Epoch 1: val_loss improved from inf to 1.81966, saving model to best_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 cor
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 409ms/step - accuracy: 0.1927 - loss: 2.1035 - val_accuracy: 0.2222 - val_loss: 1.8197
Epoch 2/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 452ms/step - accuracy: 0.2382 - loss: 1.7286
Epoch 2: val_loss improved from 1.81966 to 1.65113, saving model to best_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 cor
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 586ms/step - accuracy: 0.2355 - loss: 1.7257 - val_accuracy: 0.3333 - val_loss: 1.6511
Epoch 3/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 450ms/step - accuracy: 0.4149 - loss: 1.5336
Epoch 3: val_loss improved from 1.65113 to 1.47348, saving model to best_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 cor
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 549ms/step - accuracy: 0.4060 - loss: 1.5305 - val_accuracy: 0.3889 - val_loss: 1.4735
Epoch 4/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step - accuracy: 0.5486 - loss: 1.3130
Epoch 4: val_loss improved from 1.47348 to 1.34499, saving model to best_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 cor
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 331ms/step - accuracy: 0.5567 - loss: 1.3089 - val_accuracy: 0.5556 - val_loss: 1.3450
Epoch 5/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step - accuracy: 0.6951 - loss: 0.9427
Epoch 5: val_loss did not improve from 1.34499
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 295ms/step - accuracy: 0.6881 - loss: 0.9440 - val_accuracy: 0.3889 - val_loss: 1.6149
Epoch 6/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step - accuracy: 0.8181 - loss: 0.7474
Epoch 6: val_loss improved from 1.34499 to 1.11054, saving model to best_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 cor
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 313ms/step - accuracy: 0.8183 - loss: 0.7379 - val_accuracy: 0.6111 - val_loss: 1.1105
Epoch 7/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 833ms/step - accuracy: 0.8993 - loss: 0.4183
Epoch 7: val_loss did not improve from 1.11054
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8999 - loss: 0.4152 - val_accuracy: 0.4444 - val_loss: 1.2217
Epoch 8/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 577ms/step - accuracy: 0.9646 - loss: 0.2434
Epoch 8: val_loss did not improve from 1.11054
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 653ms/step - accuracy: 0.9635 - loss: 0.2409 - val_accuracy: 0.5556 - val_loss: 1.2669
Epoch 9/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 504ms/step - accuracy: 0.9778 - loss: 0.1330
Epoch 9: val_loss did not improve from 1.11054
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 604ms/step - accuracy: 0.9769 - loss: 0.1326 - val_accuracy: 0.5556 - val_loss: 1.2621
Epoch 10/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 394ms/step - accuracy: 1.0000 - loss: 0.0604
Epoch 10: val_loss did not improve from 1.11054
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 470ms/step - accuracy: 1.0000 - loss: 0.0593 - val_accuracy: 0.3889 - val_loss: 1.7675
Epoch 11/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 568ms/step - accuracy: 1.0000 - loss: 0.0329
Epoch 11: val_loss did not improve from 1.11054
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 662ms/step - accuracy: 1.0000 - loss: 0.0336 - val_accuracy: 0.4444 - val_loss: 1.7860
Epoch 12/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 395ms/step - accuracy: 0.9899 - loss: 0.0307
Epoch 12: val_loss did not improve from 1.11054
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 481ms/step - accuracy: 0.9893 - loss: 0.0314 - val_accuracy: 0.5000 - val_loss: 1.5862
Epoch 13/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 536ms/step - accuracy: 1.0000 - loss: 0.0141
Epoch 13: val_loss did not improve from 1.11054
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 596ms/step - accuracy: 1.0000 - loss: 0.0153 - val_accuracy: 0.3889 - val_loss: 1.8181
```

Task 5: Evaluate the Model

```
from tensorflow.keras.preprocessing import image_dataset_from_directory

# Load the test data (assuming the test data is in a similar format to the training data)
test_ds = image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),  # Ensure test images are resized to match training images
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=False
)

# Apply normalization to the test dataset (same as training and validation datasets)
```

```
test_ds = test_ds.map(lambda x, y: (rescale(x), y))

# Evaluate the model on the test dataset
test_loss, test_accuracy = model.evaluate(test_ds)

# Print the results
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
Found 30 files belonging to 6 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 574ms/step - accuracy: 0.7000 - loss: 1.0521
Test Loss: 1.0521148443222046
Test Accuracy: 0.699999988079071
```

```
# Save the model to an .h5 file
model.save('my_model.keras')
```

```
# Load the model in the Keras format
loaded_model = load_model('my_model.keras')
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/saving/saving_lib.py:757: UserWarning: Skipping variable loading for optimizer 'rmspro
  saveable.load_own_variables(weights_store.get(inner_path))
```

```
# Evaluate the loaded model on the test dataset
test_loss, test_accuracy = loaded_model.evaluate(test_ds)

# Print the results
print(f"Test Loss (after reloading): {test_loss}")
print(f"Test Accuracy (after reloading): {test_accuracy}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 625ms/step - accuracy: 0.7000 - loss: 1.0521
Test Loss (after reloading): 1.0521148443222046
Test Accuracy (after reloading): 0.699999988079071
```

Task 7: Predictions and Classification Report

```
import numpy as np
from sklearn.metrics import classification_report
import tensorflow as tf
import os

# Get class names from the directory structure
class_names = sorted(os.listdir(test_dir))  # List of class names

# Get the test dataset (make sure it's in the same format as train_ds)
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
)

# Get true labels from the test dataset
true_labels = np.concatenate([y.numpy() for _, y in test_ds], axis=0)

# Make predictions on the test dataset
predictions = loaded_model.predict(test_ds)

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

# Ensure true_labels and predicted_labels are 1D arrays
true_labels = true_labels.flatten()
predicted_labels = predicted_labels.flatten()

# Generate the classification report
report = classification_report(true_labels, predicted_labels, target_names=class_names)
```

```
# Print the classification report
print(report)
```

```
Found 30 files belonging to 6 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 358ms/step
              precision    recall  f1-score   support

        acai       0.75      0.60      0.67         5
     cupuacu       1.00      0.40      0.57         5
    graviola       0.42      1.00      0.59         5
     guarana       1.00      1.00      1.00         5
     pupunha       0.83      1.00      0.91         5
      tucuma       1.00      0.20      0.33         5

    accuracy                           0.70        30
   macro avg       0.83      0.70      0.68        30
weighted avg       0.83      0.70      0.68        30
```

Start coding or generate with AI.