```
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).
```

Simple CNN Implemented using Keras.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

# Load a sample dataset (MNIST for simplicity)
(x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()

# Normalize and reshape data
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
x_train = np.expand_dims(x_train, axis=-1)  # Add channel dimension
x_test = np.expand_dims(x_test, axis=-1)

# Define a simple CNN model with an explicit Input layer
model = keras.Sequential([
    keras.Input(shape=(28, 28, 1)),  # Explicit Input layer
    layers.Conv2D(32, (3, 3), activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation="relu"),
    layers.Dense(10, activation="softmax")  # 10 classes for MNIST
digits
])

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

# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=32,
validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")

# Make predictions
```

```
predictions = model.predict(x_test[:5])
predicted_labels = np.argmax(predictions, axis=1)

# Print results
print("Predicted labels:", predicted_labels)
print("Actual labels: ", y_test[:5])
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 58s 30ms/step - accuracy: 0.9084 -
loss: 0.2913 - val_accuracy: 0.9842 - val_loss: 0.0488
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 30ms/step - accuracy: 0.9862 -
loss: 0.0421 - val_accuracy: 0.9911 - val_loss: 0.0281
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 30ms/step - accuracy: 0.9918 -
loss: 0.0266 - val_accuracy: 0.9895 - val_loss: 0.0284
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9935 -
loss: 0.0186 - val_accuracy: 0.9899 - val_loss: 0.0316
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9954 -
loss: 0.0132 - val_accuracy: 0.9916 - val_loss: 0.0290
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9885 - loss:
0.0384
Test accuracy: 0.9916
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step
Predicted labels: [7 2 1 0 4]
Actual labels:  [7 2 1 0 4]
```

Exercise.

Task 1: Data Understanding and Visualization:

• Get the list of class directories from the train folder.

• Select one image randomly from each class

• Display the images in a grid format with two rows using matplotlib.

```
import os
import random
import matplotlib.pyplot as plt
from PIL import Image

train_path = '/content/drive/MyDrive/Final-Year
AI/week5/FruitinAmazon/train'
test_path = '/content/drive/MyDrive/Final-Year
AI/week5/FruitinAmazon/test'
```

```python
class_dirs = [d for d in os.listdir(train_path)
              if os.path.isdir(os.path.join(train_path, d))]
class_dirs.sort()  # Sort alphabetically for consistent ordering

print("Found classes:", class_dirs)
```

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

```python
num_classes = len(class_dirs)
cols = (num_classes + 1) // 2

plt.figure(figsize=(15, 8))
plt.suptitle("Random Sample from Each Class", fontsize=16, y=1.05)

for i, class_name in enumerate(class_dirs):
    # Get all images in the class directory
    class_path = os.path.join(train_path, class_name)
    images = [f for f in os.listdir(class_path)
             if os.path.isfile(os.path.join(class_path, f))]

    # Select a random image
    if images:  # Only proceed if there are images in the directory
        random_image = random.choice(images)
        img_path = os.path.join(class_path, random_image)

        # Open and display the image
        try:
            img = Image.open(img_path)

            # Create subplot
            plt.subplot(2, cols, i+1)
            plt.imshow(img)
            plt.title(class_name, pad=10)
            plt.axis('off')

        except Exception as e:
            print(f"Error loading image {img_path}: {e}")
    else:
        print(f"No images found in class: {class_name}")

plt.tight_layout()
plt.show()
```

What did you Observe?

Answer: The output lists six Amazonian fruit classes: acal, cupuacu, graviola, guarana, pupunha, and tucuma. The list provides a clear overview of the different fruit types contained in your dataset's training folder.

```
from PIL import ImageFile


def check_and_remove_corrupted_images(directory):
    corrupted_images = []

    for root, _, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)

            if not file.lower().endswith(('.png', '.jpg', '.jpeg',
'.gif', '.bmp')):
                continue

            try:
                with Image.open(file_path) as img:
                    img.verify()

                with Image.open(file_path) as img:
                    img.load()
            except (IOError, SyntaxError,
Image.DecompressionBombError) as e:
```

```
                print(f"Removed corrupted image: {file_path} - Error: 
{str(e)}")
                corrupted_images.append(file_path)
                os.remove(file_path)

    return corrupted_images


corrupted = check_and_remove_corrupted_images(train_path)

if not corrupted:
    print("No corrupted images found.")
else:
    print(f"\nTotal corrupted images removed: {len(corrupted)}")

No corrupted images found.
```

Task 2: Loading and Preprocessing Image Data in keras:

```
import tensorflow as tf

train_dir = '/content/drive/MyDrive/Final-Year 
AI/week5/FruitinAmazon/train'
img_height, img_width = 128, 128
batch_size = 32
validation_split = 0.2
seed = 123

train_ds_unmapped = 
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=seed
)

val_ds_unmapped = 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=seed
)

class_names = train_ds_unmapped.class_names
print("Class names:", class_names)

normalization = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds_unmapped.map(lambda x, y: (normalization(x), y))
val_ds = val_ds_unmapped.map(lambda x, y: (normalization(x), y))

for images, labels in train_ds.take(1):
    print("\nFirst training batch:")
    print("Images shape:", images.shape)
    print("Labels shape:", labels.shape)
    print("Pixel value range: ({:.2f}, {:.2f})".format(
        tf.reduce_min(images).numpy(),
        tf.reduce_max(images).numpy()
    ))
```

```
Found 90 files belonging to 6 classes.
Using 72 files for training.
Found 90 files belonging to 6 classes.
Using 18 files for validation.
Class names: ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha',
'tucuma']

First training batch:
Images shape: (32, 128, 128, 3)
Labels shape: (32,)
Pixel value range: (0.00, 1.00)
```

Task 3 - Implement a CNN with

```
import tensorflow as tf
from tensorflow.keras import layers, models


def create_cnn_model(input_shape=(128, 128, 3), num_classes=6):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), strides=1, padding='same',
activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2), strides=2),

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

        layers.Flatten(),
```

```python
        layers.Dense(64, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model

model = create_cnn_model(input_shape=(img_height, img_width, 3),
num_classes=len(class_names))

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

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 Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Model: "sequential_4"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ conv2d_8 (Conv2D)                     │ (None, 128, 128, 32)         │
896 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_8 (MaxPooling2D)        │ (None, 64, 64, 32)           │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_9 (Conv2D)                     │ (None, 64, 64, 32)           │
9,248 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_9 (MaxPooling2D)        │ (None, 32, 32, 32)           │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ flatten_4 (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)

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

 Non-trainable params: 0 (0.00 B)
```

Task 4:

Compile the Model

```
# Compile the model with recommended settings
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

extra_metrics = [
    tf.keras.metrics.SparseTopKCategoricalAccuracy(k=2,
name='top2_accuracy'),
    tf.keras.metrics.SparseCategoricalCrossentropy(name='xentropy')
]

# Verify compilation
print("Model successfully compiled!")
print("Optimizer:", model.optimizer.get_config()['name'])
print("Loss function:", model.loss)
print("Metrics:", [m.name for m in model.metrics])

Model successfully compiled!
Optimizer: adam
Loss function: sparse_categorical_crossentropy
Metrics: ['loss', 'compile_metrics']
```

Train the Model

```python
import numpy as np
from sklearn.metrics import classification_report

callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath='best_model.keras',
        monitor='val_accuracy',
        save_best_only=True,
        mode='max',
        verbose=1
    ),
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=15,
        restore_best_weights=True,
        verbose=1
    )
]

# Train the model
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=250,
    batch_size=16,
    callbacks=callbacks,
    verbose=1
)

# Plot training history
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

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

```
plt.tight_layout()
plt.show()
```

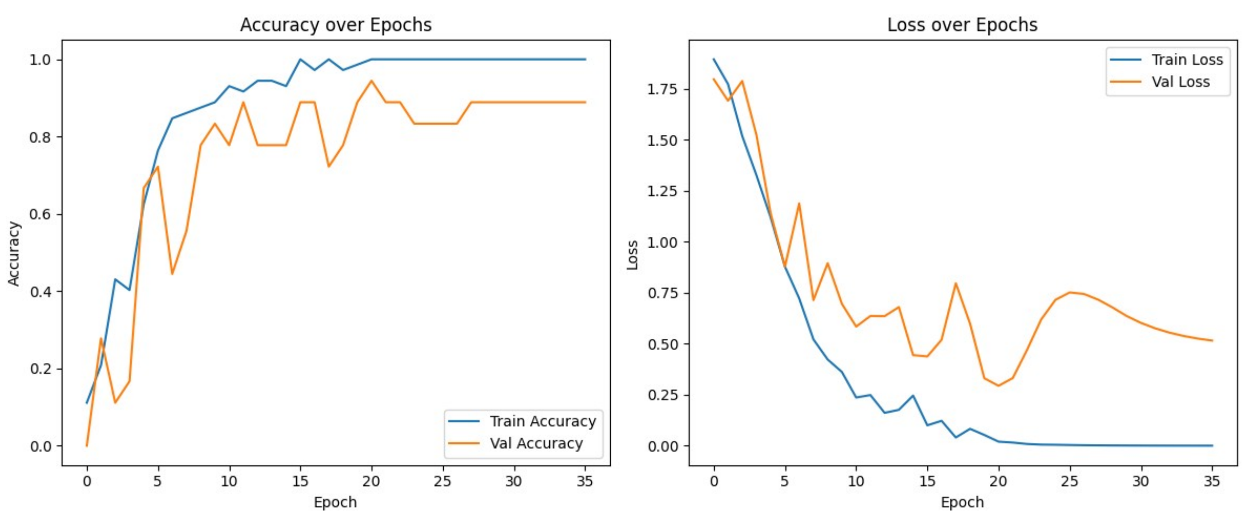
```
Epoch 1/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 446ms/step - accuracy: 0.1100 - loss: 
1.8583
Epoch 1: val_accuracy improved from -inf to 0.00000, saving model to 
best_model.keras
3/3 ━━━━━━━━━━━━━━━━━━━━ 5s 777ms/step - accuracy: 0.1102 - loss: 
1.8674 - val_accuracy: 0.0000e+00 - val_loss: 1.7968
Epoch 2/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 364ms/step - accuracy: 0.2049 - loss: 
1.7972
Epoch 2: val_accuracy improved from 0.00000 to 0.27778, saving model 
to best_model.keras
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 600ms/step - accuracy: 0.2057 - loss: 
1.7912 - val_accuracy: 0.2778 - val_loss: 1.6919
Epoch 3/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 350ms/step - accuracy: 0.4664 - loss: 
1.5320
Epoch 3: val_accuracy did not improve from 0.27778
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 437ms/step - accuracy: 0.4575 - loss: 
1.5286 - val_accuracy: 0.1111 - val_loss: 1.7885
Epoch 4/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 334ms/step - accuracy: 0.3843 - loss: 
1.3496
Epoch 4: val_accuracy did not improve from 0.27778
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 435ms/step - accuracy: 0.3889 - loss: 
1.3437 - val_accuracy: 0.1667 - val_loss: 1.5228
Epoch 5/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 351ms/step - accuracy: 0.5677 - loss: 
1.1632
Epoch 5: val_accuracy improved from 0.27778 to 0.66667, saving model 
to best_model.keras
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 519ms/step - accuracy: 0.5820 - loss: 
1.1520 - val_accuracy: 0.6667 - val_loss: 1.1380
Epoch 6/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 467ms/step - accuracy: 0.8380 - loss: 
0.7867
Epoch 6: val_accuracy improved from 0.66667 to 0.72222, saving model 
to best_model.keras
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 722ms/step - accuracy: 0.8194 - loss: 
0.8095 - val_accuracy: 0.7222 - val_loss: 0.8786
Epoch 7/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 610ms/step - accuracy: 0.8449 - loss: 
0.7197
Epoch 7: val_accuracy did not improve from 0.72222
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 768ms/step - accuracy: 0.8455 - loss: 
0.7204 - val_accuracy: 0.4444 - val_loss: 1.1885
Epoch 8/250
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 340ms/step - accuracy: 0.8600 - loss:
0.5518
Epoch 8: val_accuracy did not improve from 0.72222
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 443ms/step - accuracy: 0.8602 - loss:
0.5442 - val_accuracy: 0.5556 - val_loss: 0.7141
Epoch 9/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 348ms/step - accuracy: 0.8646 - loss:
0.4324
Epoch 9: val_accuracy improved from 0.72222 to 0.77778, saving model
to best_model.keras
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 510ms/step - accuracy: 0.8672 - loss:
0.4301 - val_accuracy: 0.7778 - val_loss: 0.8952
Epoch 10/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 346ms/step - accuracy: 0.9109 - loss:
0.3255
Epoch 10: val_accuracy improved from 0.77778 to 0.83333, saving model
to best_model.keras
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 582ms/step - accuracy: 0.9054 - loss:
0.3346 - val_accuracy: 0.8333 - val_loss: 0.6960
Epoch 11/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 655ms/step - accuracy: 0.9248 - loss:
0.2322
Epoch 11: val_accuracy did not improve from 0.83333
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 832ms/step - accuracy: 0.9262 - loss:
0.2333 - val_accuracy: 0.7778 - val_loss: 0.5845
Epoch 12/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 481ms/step - accuracy: 0.9253 - loss:
0.2306
Epoch 12: val_accuracy improved from 0.83333 to 0.88889, saving model
to best_model.keras
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 639ms/step - accuracy: 0.9232 - loss:
0.2351 - val_accuracy: 0.8889 - val_loss: 0.6365
Epoch 13/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 350ms/step - accuracy: 0.9554 - loss:
0.1390
Epoch 13: val_accuracy did not improve from 0.88889
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 454ms/step - accuracy: 0.9527 - loss:
0.1447 - val_accuracy: 0.7778 - val_loss: 0.6357
Epoch 14/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 357ms/step - accuracy: 0.9450 - loss:
0.1791
Epoch 14: val_accuracy did not improve from 0.88889
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 526ms/step - accuracy: 0.9449 - loss:
0.1784 - val_accuracy: 0.7778 - val_loss: 0.6802
Epoch 15/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 355ms/step - accuracy: 0.9091 - loss:
0.3189
Epoch 15: val_accuracy did not improve from 0.88889
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 526ms/step - accuracy: 0.9145 - loss:
```

```
0.3007 - val_accuracy: 0.7778 - val_loss: 0.4445
Epoch 16/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 1.0000 - loss:
0.1004
Epoch 16: val_accuracy did not improve from 0.88889
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 542ms/step - accuracy: 1.0000 - loss:
0.1004 - val_accuracy: 0.8889 - val_loss: 0.4383
Epoch 17/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 624ms/step - accuracy: 0.9751 - loss:
0.1256
Epoch 17: val_accuracy did not improve from 0.88889
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 780ms/step - accuracy: 0.9744 - loss:
0.1249 - val_accuracy: 0.8889 - val_loss: 0.5200
Epoch 18/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 573ms/step - accuracy: 1.0000 - loss:
0.0410
Epoch 18: val_accuracy did not improve from 0.88889
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 666ms/step - accuracy: 1.0000 - loss:
0.0410 - val_accuracy: 0.7222 - val_loss: 0.7967
Epoch 19/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 350ms/step - accuracy: 0.9595 - loss:
0.0986
Epoch 19: val_accuracy did not improve from 0.88889
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 442ms/step - accuracy: 0.9627 - loss:
0.0948 - val_accuracy: 0.7778 - val_loss: 0.5992
Epoch 20/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 335ms/step - accuracy: 0.9797 - loss:
0.0649
Epoch 20: val_accuracy did not improve from 0.88889
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 435ms/step - accuracy: 0.9813 - loss:
0.0620 - val_accuracy: 0.8889 - val_loss: 0.3314
Epoch 21/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 351ms/step - accuracy: 1.0000 - loss:
0.0200
Epoch 21: val_accuracy improved from 0.88889 to 0.94444, saving model
to best_model.keras
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 584ms/step - accuracy: 1.0000 - loss:
0.0201 - val_accuracy: 0.9444 - val_loss: 0.2941
Epoch 22/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 583ms/step - accuracy: 1.0000 - loss:
0.0153
Epoch 22: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 731ms/step - accuracy: 1.0000 - loss:
0.0155 - val_accuracy: 0.8889 - val_loss: 0.3322
Epoch 23/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 367ms/step - accuracy: 1.0000 - loss:
0.0103
Epoch 23: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 463ms/step - accuracy: 1.0000 - loss:
```

```
0.0100 - val_accuracy: 0.8889 - val_loss: 0.4699
Epoch 24/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 351ms/step - accuracy: 1.0000 - loss:
0.0060
Epoch 24: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 436ms/step - accuracy: 1.0000 - loss:
0.0060 - val_accuracy: 0.8333 - val_loss: 0.6201
Epoch 25/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 399ms/step - accuracy: 1.0000 - loss:
0.0051
Epoch 25: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 567ms/step - accuracy: 1.0000 - loss:
0.0051 - val_accuracy: 0.8333 - val_loss: 0.7158
Epoch 26/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 352ms/step - accuracy: 1.0000 - loss:
0.0047
Epoch 26: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 447ms/step - accuracy: 1.0000 - loss:
0.0045 - val_accuracy: 0.8333 - val_loss: 0.7520
Epoch 27/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 599ms/step - accuracy: 1.0000 - loss:
0.0032
Epoch 27: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 774ms/step - accuracy: 1.0000 - loss:
0.0031 - val_accuracy: 0.8333 - val_loss: 0.7444
Epoch 28/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 353ms/step - accuracy: 1.0000 - loss:
0.0023
Epoch 28: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 455ms/step - accuracy: 1.0000 - loss:
0.0023 - val_accuracy: 0.8889 - val_loss: 0.7164
Epoch 29/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 357ms/step - accuracy: 1.0000 - loss:
0.0017
Epoch 29: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 454ms/step - accuracy: 1.0000 - loss:
0.0017 - val_accuracy: 0.8889 - val_loss: 0.6786
Epoch 30/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 375ms/step - accuracy: 1.0000 - loss:
0.0016
Epoch 30: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 542ms/step - accuracy: 1.0000 - loss:
0.0016 - val_accuracy: 0.8889 - val_loss: 0.6363
Epoch 31/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 369ms/step - accuracy: 1.0000 - loss:
0.0013
Epoch 31: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 537ms/step - accuracy: 1.0000 - loss:
0.0013 - val_accuracy: 0.8889 - val_loss: 0.6025
```

```
Epoch 32/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 394ms/step - accuracy: 1.0000 - loss:
9.4255e-04
Epoch 32: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 574ms/step - accuracy: 1.0000 - loss:
9.6477e-04 - val_accuracy: 0.8889 - val_loss: 0.5761
Epoch 33/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 590ms/step - accuracy: 1.0000 - loss:
8.5979e-04
Epoch 33: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 769ms/step - accuracy: 1.0000 - loss:
8.7561e-04 - val_accuracy: 0.8889 - val_loss: 0.5549
Epoch 34/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 350ms/step - accuracy: 1.0000 - loss:
8.7243e-04
Epoch 34: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 445ms/step - accuracy: 1.0000 - loss:
8.6277e-04 - val_accuracy: 0.8889 - val_loss: 0.5385
Epoch 35/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 355ms/step - accuracy: 1.0000 - loss:
7.8817e-04
Epoch 35: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 527ms/step - accuracy: 1.0000 - loss:
7.8008e-04 - val_accuracy: 0.8889 - val_loss: 0.5257
Epoch 36/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 352ms/step - accuracy: 1.0000 - loss:
6.1371e-04
Epoch 36: val_accuracy did not improve from 0.94444
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 458ms/step - accuracy: 1.0000 - loss:
6.3433e-04 - val_accuracy: 0.8889 - val_loss: 0.5160
Epoch 36: early stopping
Restoring model weights from the end of the best epoch: 21.
```



Task 5: Evaluate the Model

```
# Load test dataset
test_dir = '/content/drive/MyDrive/Final-Year
AI/week5/FruitinAmazon/test'
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    label_mode='int'
).map(lambda x, y: (normalization(x), y))

# Evaluate on test set
test_loss, test_acc = model.evaluate(test_ds)
print(f'\nTest Accuracy: {test_acc:.4f}')
print(f'Test Loss: {test_loss:.4f}')
```

```
Found 30 files belonging to 6 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 269ms/step - accuracy: 0.6333 - loss:
0.7824

Test Accuracy: 0.6333
Test Loss: 0.7824
```

Task 6: Save and Load the Model

```
model.save('fruit_classifier.keras', include_optimizer=False)

# Load the saved model
loaded_model = tf.keras.models.load_model('fruit_classifier.keras')

# Verify loaded model
loaded_loss, loaded_acc = loaded_model.evaluate(test_ds)
print(f'\nLoaded Model Test Accuracy: {loaded_acc:.4f}')
print(f'Loaded Model Test Loss: {loaded_loss:.4f}')
```

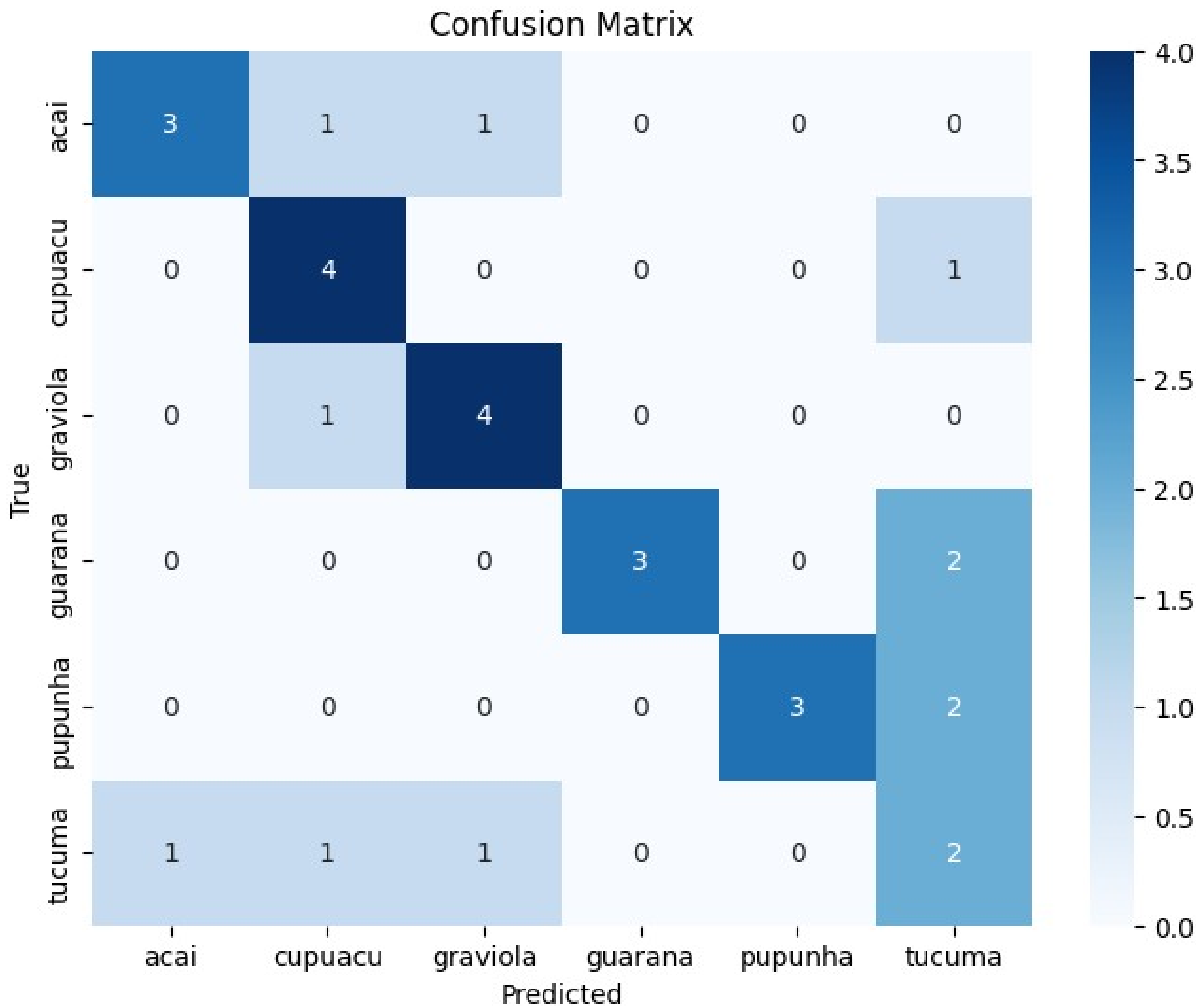
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 575ms/step - accuracy: 0.6333 - loss:
0.7824

Loaded Model Test Accuracy: 0.6333
Loaded Model Test Loss: 0.7824
```

Task 7: Predictions and Classification Report

```
import numpy as np
from sklearn.metrics import classification_report

y_true = []
y_pred = []

for images, labels in test_ds:
    y_true.extend(labels.numpy())
```

```
    y_pred.extend(np.argmax(loaded_model.predict(images), axis=1))

# Classification report
print('\nClassification Report:')
print(classification_report(
    y_true,
    y_pred,
    target_names=class_names
))

# Confusion matrix visualization
from sklearn.metrics import confusion_matrix
import seaborn as sns

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 262ms/step

Classification Report:
              precision    recall  f1-score   support

        acai       0.75      0.60      0.67         5
     cupuacu       0.57      0.80      0.67         5
    graviola       0.67      0.80      0.73         5
     guarana       1.00      0.60      0.75         5
     pupunha       1.00      0.60      0.75         5
      tucuma       0.29      0.40      0.33         5

    accuracy                           0.63        30
   macro avg       0.71      0.63      0.65        30
weighted avg       0.71      0.63      0.65        30
```


Confusion Matrix
acai
cupuacu
graviola
guarana
pupunha
tucuma
True
Predicted
3 1 1 0 0 0
0 4 0 0 0 1
0 1 4 0 0 0
0 0 0 3 0 2
0 0 0 0 3 2
1 1 1 0 0 2
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0