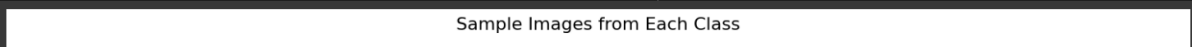
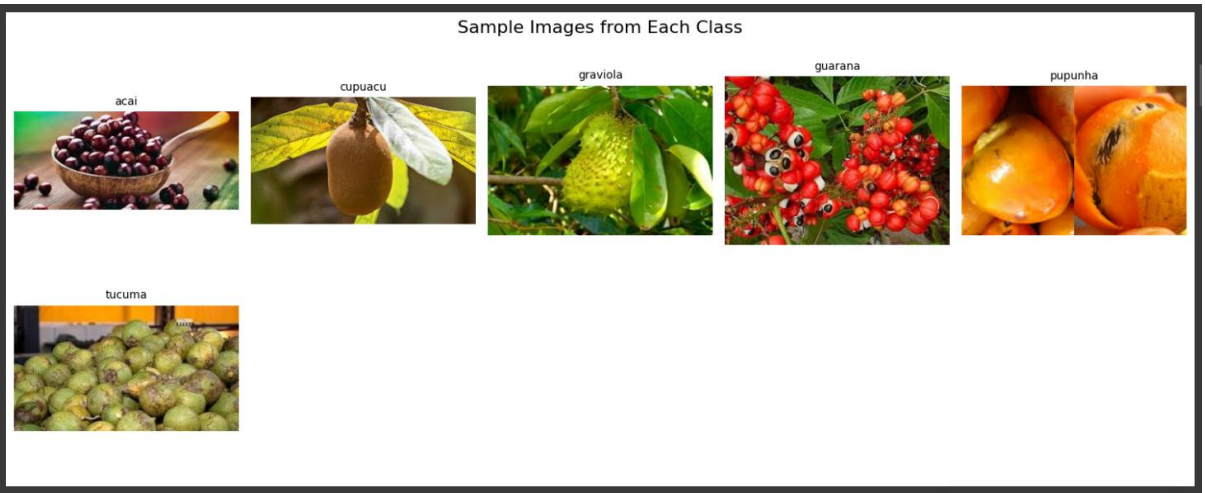
```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

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

```
import os
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

train_dir = "/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train"

class_names = sorted([d for d in os.listdir(train_dir) if os.path.isdir(os.path.join(train_dir, d))])

if not class_names:
    raise ValueError("No class directories found in the train folder. Check dataset path!")

selected_images = []
selected_labels = []

for class_name in class_names:
    class_path = os.path.join(train_dir, class_name)
    image_files = [f for f in os.listdir(class_path) if f.endswith(('png', 'jpg', 'jpeg'))]

    if image_files:
        random_image = random.choice(image_files)
        selected_images.append(os.path.join(class_path, random_image))
        selected_labels.append(class_name)
```

```
    if image_files:
        random_image = random.choice(image_files)
        selected_images.append(os.path.join(class_path, random_image))
        selected_labels.append(class_name)

num_classes = len(selected_images)
if num_classes == 0:
    raise ValueError("No images found in any class folder. Please check dataset.")

cols = min(5, num_classes)
rows = (num_classes // cols) + (num_classes % cols > 0)

fig, axes = plt.subplots(rows, cols, figsize=(15, 6))
fig.suptitle("Sample Images from Each Class", fontsize=16)

for i, ax in enumerate(axes.flat):
    if i < num_classes:
        img = mpimg.imread(selected_images[i])
        ax.imshow(img)
        ax.set_title(selected_labels[i], fontsize=10)
        ax.axis("off")
    else:
        ax.axis("off")

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





What did you Observe?

What did you Observe?

Each image represents a different class, confirming correct dataset structure. Images vary in resolution, lighting, and orientation. Some classes may have fewer images, indicating dataset imbalance. Preprocessing (resizing, normalization) may be needed for consistency.

```
import os
from PIL import Image
train_dir = "/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train"

corrupted_images = []

for class_name in sorted(os.listdir(train_dir)):
    class_path = os.path.join(train_dir, class_name)

    if os.path.isdir(class_path):
        for image_name in os.listdir(class_path):
            image_path = os.path.join(class_path, image_name)

            try:

                with Image.open(image_path) as img:
                    img.verify()
            except (IOError, SyntaxError):

                corrupted_images.append(image_path)
                os.remove(image_path)
                print(f"Removed corrupted image: {image_path}")
```

```
[7]                 os.remove(image_path)
                    print(f"Removed corrupted image: {image_path}")

if not corrupted_images:
    print("No Corrupted Images Found.")
```

```
No Corrupted Images Found.
```

```
import tensorflow as tf

img_height = 128
img_width = 128
batch_size = 32
validation_split = 0.2

rescale = tf.keras.layers.Rescaling(1./255)

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
)
```

```
    shuffle=True,
    validation_split=validation_split,
    subset='training',
    seed=123
)

train_ds = train_ds.map(lambda x, y: (rescale(x), y))

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
)

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.
```

```
Using 18 files for validation.
```

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

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu', input_shape=(128, 128, 3)))
model.add(layers.MaxPooling2D((2, 2), strides=2))
model.add(layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu'))
model.add(layers.MaxPooling2D((2, 2), strides=2))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))

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. Whe
  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

```
/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. Whe
  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)
Non-trainable params: 0 (0.00 B)
```

```
Total params: 2,116,454 (8.07 MB)
Trainable params: 2,116,454 (8.07 MB)
Non-trainable params: 0 (0.00 B)
```

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

```python
import tensorflow as tf

callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        'RenjenSherpa.h5',
        monitor='val_loss',
        save_best_only=True,
        mode='min',
        verbose=1
    ),
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    )
]
```

```
    ),
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    )
]

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=250,
    batch_size=16,
    callbacks=callbacks
)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 495ms/step - accuracy: 0.9332 - loss: 0.2403 - val_accuracy: 0.7222 - val_loss: 0.7414
Epoch 8/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 579ms/step - accuracy: 1.0000 - loss: 0.0634
Epoch 8: val_loss did not improve from 0.45226
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 726ms/step - accuracy: 1.0000 - loss: 0.0622 - val_accuracy: 0.8333 - val_loss: 0.4709
Epoch 9/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 548ms/step - accuracy: 1.0000 - loss: 0.0283
Epoch 9: val_loss improved from 0.45226 to 0.44183, saving model to RenjenSherpa.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 u
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 762ms/step - accuracy: 1.0000 - loss: 0.0287 - val_accuracy: 0.8333 - val_loss: 0.4418
Epoch 10/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 351ms/step - accuracy: 1.0000 - loss: 0.0199
Epoch 10: val_loss improved from 0.44183 to 0.43879, saving model to RenjenSherpa.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 u
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 564ms/step - accuracy: 1.0000 - loss: 0.0199 - val_accuracy: 0.8333 - val_loss: 0.4388
```

```
Epoch 12: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 508ms/step - accuracy: 1.0000 - loss: 0.0095 - val_accuracy: 0.8333 - val_loss: 0.4717
Epoch 13/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 337ms/step - accuracy: 1.0000 - loss: 0.0071
Epoch 13: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 423ms/step - accuracy: 1.0000 - loss: 0.0071 - val_accuracy: 0.8333 - val_loss: 0.4861
Epoch 14/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 360ms/step - accuracy: 1.0000 - loss: 0.0059
Epoch 14: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 537ms/step - accuracy: 1.0000 - loss: 0.0058 - val_accuracy: 0.8333 - val_loss: 0.4894
Epoch 15/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 571ms/step - accuracy: 1.0000 - loss: 0.0041
Epoch 15: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 698ms/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 0.8333 - val_loss: 0.4807
Epoch 16/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 337ms/step - accuracy: 1.0000 - loss: 0.0039
Epoch 16: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 508ms/step - accuracy: 1.0000 - loss: 0.0038 - val_accuracy: 0.8333 - val_loss: 0.4751
Epoch 17/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 333ms/step - accuracy: 1.0000 - loss: 0.0027
Epoch 17: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 421ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 0.8333 - val_loss: 0.4822
Epoch 18/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 335ms/step - accuracy: 1.0000 - loss: 0.0025
Epoch 18: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 421ms/step - accuracy: 1.0000 - loss: 0.0025 - val_accuracy: 0.8333 - val_loss: 0.4973
Epoch 19/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 326ms/step - accuracy: 1.0000 - loss: 0.0021
Epoch 19: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 493ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 0.8333 - val_loss: 0.5223
Epoch 20/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 561ms/step - accuracy: 1.0000 - loss: 0.0023
Epoch 20: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 721ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.8333 - val_loss: 0.5556
Epoch 20: early stopping
Restoring model weights from the end of the best epoch: 10.
```

```
test_dir = '/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/test'

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=False
)

test_loss, test_acc = model.evaluate(test_ds)

print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_acc}")
```

```
Found 30 files belonging to 6 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step - accuracy: 0.7333 - loss: 118.4351
Test Loss: 118.4350814819336
Test Accuracy: 0.7333333492279053
```

```
model.save('RenjenSherpa_Model.h5')
print("Model saved successfully!")
```

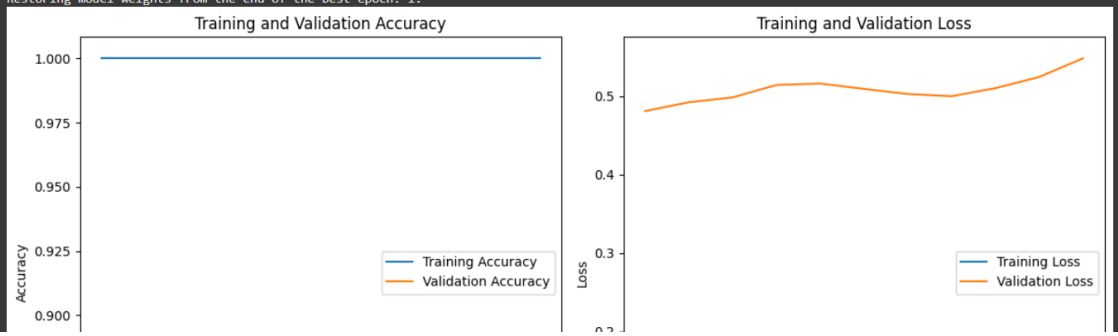
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend usir
Model saved successfully!
```

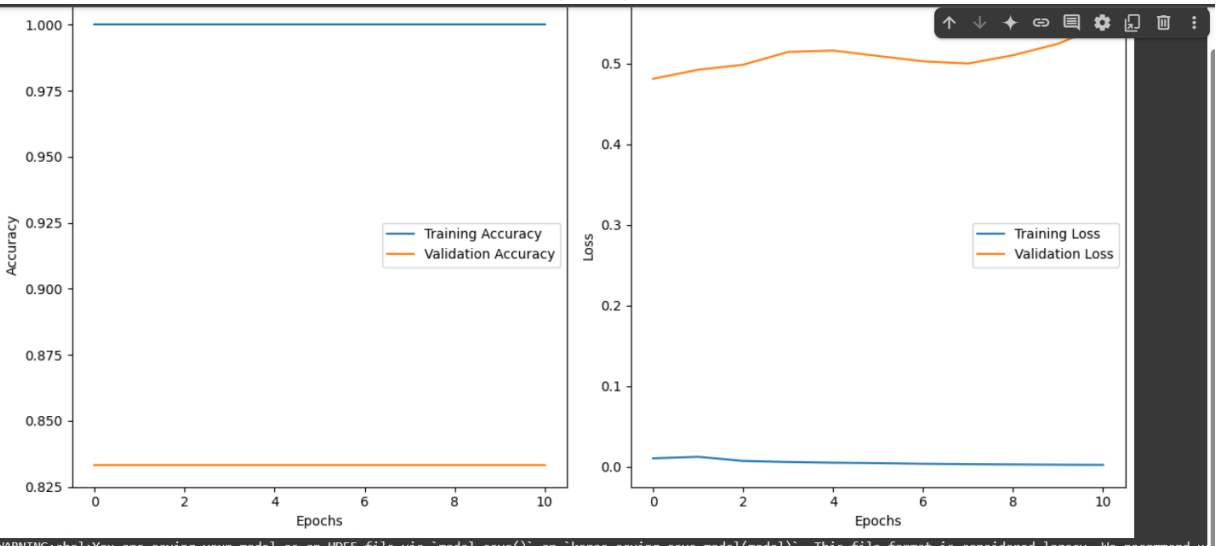
```
import numpy as np
import tensorflow as tf
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

test_images, test_labels = [], []

for images, labels in test_ds:
    test_images.append(images)
    test_labels.append(labels)

test_images = np.concatenate(test_images, axis=0)
test_labels = np.concatenate(test_labels, axis=0)

predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)

print("Classification Report:")
print(classification_report(test_labels, predicted_labels))

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=250,
    batch_size=16,
    callbacks=callbacks
)

plt.figure(figsize=(12, 6))
```

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

model.save('RenjenSherpa_Model.h5')
print("Model saved as 'samip_maharjan_model.h5'")
```

```
Epoch 7: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 420ms/step - accuracy: 1.0000 - loss: 0.0034 - val_accuracy: 0.8333 - val_loss: 0.5028
Epoch 8/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 346ms/step - accuracy: 1.0000 - loss: 0.0028
Epoch 8: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 463ms/step - accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 0.8333 - val_loss: 0.5000
Epoch 9/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 564ms/step - accuracy: 1.0000 - loss: 0.0026
```

```
Epoch 8: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 463ms/step - accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 0.8333 - val_loss: 0.5000
Epoch 9/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 564ms/step - accuracy: 1.0000 - loss: 0.0026
Epoch 9: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 672ms/step - accuracy: 1.0000 - loss: 0.0026 - val_accuracy: 0.8333 - val_loss: 0.5103
Epoch 10/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 341ms/step - accuracy: 1.0000 - loss: 0.0020
Epoch 10: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 443ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 0.8333 - val_loss: 0.5246
Epoch 11/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 337ms/step - accuracy: 1.0000 - loss: 0.0021
Epoch 11: val_loss did not improve from 0.43879
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 424ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 0.8333 - val_loss: 0.5483
Epoch 11: early stopping
Restoring model weights from the end of the best epoch: 1.
```






```
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 u
Model saved as 'samip_maharjan_model.h5'
```