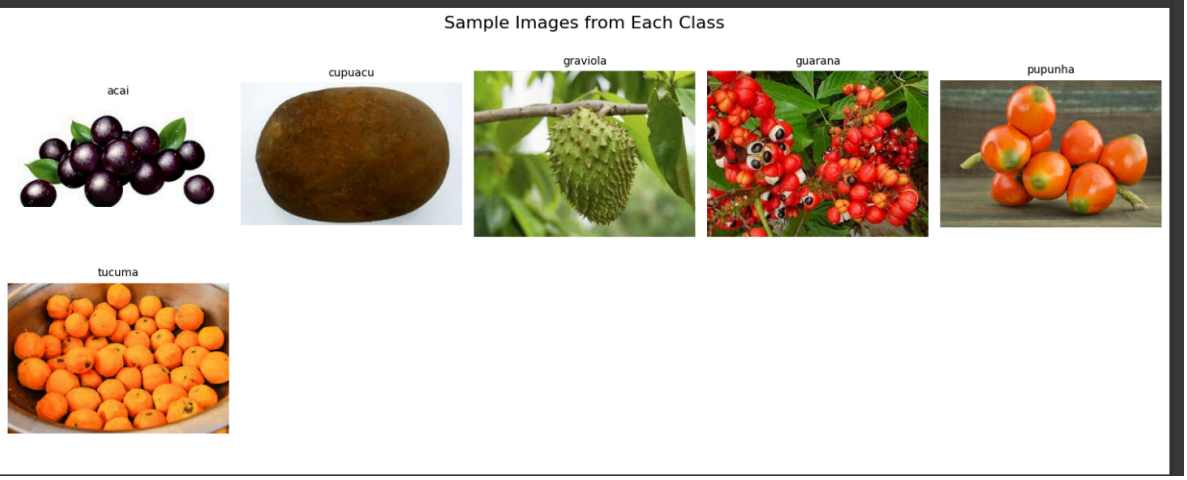
```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

```
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)
```

```
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()
```

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



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}")

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

```
No Corrupted Images Found.
```

```python
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
)

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.
```

```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. When using Sequential models, prefer using an `Input(shape)` object as the first
  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)
```

```python
import tensorflow as tf

callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        'Renjen Sherpa.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
    )
]

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

```
Epoch 1/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 699ms/step - accuracy: 1.0000 - loss: 0.0685
Epoch 1: val_loss improved from inf to 0.73160, saving model to Renjen Sherpa.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 using instead the native Keras format, e.g. `mod
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 965ms/step - accuracy: 1.0000 - loss: 0.0690 - val_accuracy: 0.8333 - val_loss: 0.7316
Epoch 2/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 1.0000 - loss: 0.0604
Epoch 2: val_loss improved from 0.73160 to 0.51434, saving model to Renjen Sherpa.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 using instead the native Keras format, e.g. `mod
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 584ms/step - accuracy: 1.0000 - loss: 0.0577 - val_accuracy: 0.8333 - val_loss: 0.5143
Epoch 3/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step - accuracy: 1.0000 - loss: 0.0291
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 512ms/step - accuracy: 1.0000 - loss: 0.0343 - val_accuracy: 0.8333 - val_loss: 0.3957
Epoch 5/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 646ms/step - accuracy: 1.0000 - loss: 0.0148
Epoch 5: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 804ms/step - accuracy: 1.0000 - loss: 0.0150 - val_accuracy: 0.8333 - val_loss: 0.4545
Epoch 6/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 573ms/step - accuracy: 1.0000 - loss: 0.0134
Epoch 6: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 662ms/step - accuracy: 1.0000 - loss: 0.0136 - val_accuracy: 0.8333 - val_loss: 0.4896
Epoch 7/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 363ms/step - accuracy: 1.0000 - loss: 0.0092
Epoch 7: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 533ms/step - accuracy: 1.0000 - loss: 0.0092 - val_accuracy: 0.8333 - val_loss: 0.4899
Epoch 8/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 374ms/step - accuracy: 1.0000 - loss: 0.0076
Epoch 8: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 542ms/step - accuracy: 1.0000 - loss: 0.0074 - val_accuracy: 0.8333 - val_loss: 0.4763
Epoch 9/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 1.0000 - loss: 0.0057
Epoch 9: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 539ms/step - accuracy: 1.0000 - loss: 0.0056 - val_accuracy: 0.8333 - val_loss: 0.4708
Epoch 10/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 356ms/step - accuracy: 1.0000 - loss: 0.0049
Epoch 10: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 459ms/step - accuracy: 1.0000 - loss: 0.0048 - val_accuracy: 0.8333 - val_loss: 0.4599
Epoch 11/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 369ms/step - accuracy: 1.0000 - loss: 0.0035
Epoch 11: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 507ms/step - accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 0.8333 - val_loss: 0.4466
Epoch 12/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 661ms/step - accuracy: 1.0000 - loss: 0.0031
Epoch 12: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 836ms/step - accuracy: 1.0000 - loss: 0.0031 - val_accuracy: 0.8333 - val_loss: 0.4494
Epoch 13/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step - accuracy: 1.0000 - loss: 0.0027
Epoch 13: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 467ms/step - accuracy: 1.0000 - loss: 0.0027 - val_accuracy: 0.8333 - val_loss: 0.4607
Epoch 14/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 382ms/step - accuracy: 1.0000 - loss: 0.0022
Epoch 14: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 470ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.8333 - val_loss: 0.4657
Epoch 14: early stopping
Restoring model weights from the end of the best epoch: 4.
```

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

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 120 files belonging to 2 classes.
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 166ms/step - accuracy: 0.2110 - loss: 1154.1299
Test Loss: 1195.381591796875
Test Accuracy: 0.1916666626930368
```

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

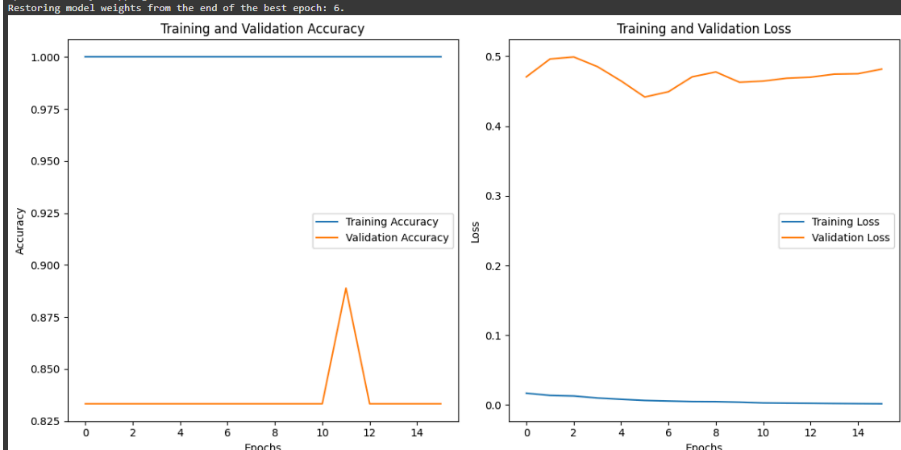
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mod
Model saved successfully!
```

```python
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.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()
```

```python
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('RenjenSherpaModel.h5')
print("Model saved as 'RenjenSherpaModel.h5'")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 195ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.25      0.17      0.20        30
           1       0.69      0.20      0.31        90
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.00      0.00      0.00         0

    accuracy                           0.19       120
   macro avg       0.16      0.06      0.09       120
weighted avg       0.58      0.19      0.28       120

Epoch 1/250
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` paramete
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` paramete
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` paramete
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 1.0000 - loss: 0.0187
Epoch 1: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 462ms/step - accuracy: 1.0000 - loss: 0.0182 - val_accuracy: 0.8333 - val_loss: 0.4706
Epoch 2/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 1.0000 - loss: 0.0132
Epoch 2: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 459ms/step - accuracy: 1.0000 - loss: 0.0133 - val_accuracy: 0.8333 - val_loss: 0.4962
Epoch 3/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 571ms/step - accuracy: 1.0000 - loss: 0.0119
Epoch 3: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 751ms/step - accuracy: 1.0000 - loss: 0.0121 - val_accuracy: 0.8333 - val_loss: 0.4992
Epoch 4/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 610ms/step - accuracy: 1.0000 - loss: 0.0106
Epoch 4: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 750ms/step - accuracy: 1.0000 - loss: 0.0104 - val_accuracy: 0.8333 - val_loss: 0.4853
Epoch 5/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 1.0000 - loss: 0.0078
Epoch 5: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 545ms/step - accuracy: 1.0000 - loss: 0.0078 - val_accuracy: 0.8333 - val_loss: 0.4648
Epoch 6/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 1.0000 - loss: 0.0062
Epoch 6: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 459ms/step - accuracy: 1.0000 - loss: 0.0063 - val_accuracy: 0.8333 - val_loss: 0.4417
Epoch 7/250
```

```
Epoch 15: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 556ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 0.8333 - val_loss: 0.4751
Epoch 16/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 480ms/step - accuracy: 1.0000 - loss: 0.0017
Epoch 16: val_loss did not improve from 0.39568
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 660ms/step - accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.8333 - val_loss: 0.4817
Epoch 16: early stopping
Restoring model weights from the end of the best epoch: 6.
```




```
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. `m
Model saved as 'RenjenSherpaModel.h5'
```

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # 20% of the data used for validation
)

train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 72 images belonging to 6 classes.
Found 18 images belonging to 6 classes.
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

val_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(
        '/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical')

val_generator = val_datagen.flow_from_directory(
        '/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical')

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
```

29s completed at 12:12 AM

```python
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(6, activation='softmax')
])

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

# Fit model
history = model.fit(train_generator, epochs=10, validation_data=val_generator)
```

```
/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 a
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 62, 62, 32)	896
batch_normalization (BatchNormalization)	(None, 62, 62, 32)	128
max_pooling2d_2 (MaxPooling2D)	(None, 31, 31, 32)	0
dropout (Dropout)	(None, 31, 31, 32)	0
conv2d_3 (Conv2D)	(None, 29, 29, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 29, 29, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0

conv2d_3 (Conv2D)	(None, 29, 29, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 29, 29, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_3 (Dense)	(None, 128)	1,605,760
batch_normalization_2 (BatchNormalization)	(None, 128)	512
dropout_2 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 6)	774

Total params: 1,626,823 (6.21 MB)
Trainable params: 1,626,374 (6.20 MB)
Non-trainable params: 448 (1.75 KB)

```
/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` c
  self._warn_if_super_not_called()
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 895ms/step - accuracy: 0.2108 - loss: 3.1151 - val_accuracy: 0.2667 - val_loss: 1.7871
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 547ms/step - accuracy: 0.5039 - loss: 1.7635 - val_accuracy: 0.2222 - val_loss: 1.8114
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 810ms/step - accuracy: 0.4992 - loss: 1.7595 - val_accuracy: 0.2444 - val_loss: 1.7385
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 902ms/step - accuracy: 0.6384 - loss: 1.1229 - val_accuracy: 0.2222 - val_loss: 1.7235
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 552ms/step - accuracy: 0.5099 - loss: 1.1949 - val_accuracy: 0.2222 - val_loss: 1.7471
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 567ms/step - accuracy: 0.6693 - loss: 1.0955 - val_accuracy: 0.2111 - val_loss: 1.7637
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 556ms/step - accuracy: 0.4616 - loss: 1.3470 - val_accuracy: 0.2222 - val_loss: 1.7951
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 611ms/step - accuracy: 0.5882 - loss: 1.3256 - val_accuracy: 0.2111 - val_loss: 1.8007
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.6133 - loss: 1.1672 - val_accuracy: 0.2000 - val_loss: 1.8276
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 545ms/step - accuracy: 0.6723 - loss: 0.9993 - val_accuracy: 0.1889 - val_loss: 1.9064
```

Task 2: Transfer Learning using VGG16

```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense

# Load base VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

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

# Add custom top layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)

predictions = Dense(6, activation='softmax')(x)

# Create model
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

train_generator_vgg = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical')

val_generator_vgg = val_datagen.flow_from_directory(
    '/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical')
```

```
val_generator_vgg = val_datagen.flow_from_directory(
    '/content/drive/MyDrive/Artificial intelligence and machine learning/Worksheet5/FruitinAmazon/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical')

# Train the model
history_vgg = model.fit(train_generator_vgg, epochs=5, validation_data=val_generator_vgg)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "functional_9"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense_5 (Dense)	(None, 1024)	525,312
dense_6 (Dense)	(None, 6)	6,150

```
 Total params: 15,246,150 (58.16 MB)
 Trainable params: 531,462 (2.03 MB)
 Non-trainable params: 14,714,688 (56.13 MB)
Found 90 images belonging to 6 classes.
Found 90 images belonging to 6 classes.
Epoch 1/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 119s 47s/step - accuracy: 0.1648 - loss: 1.8706 - val_accuracy: 0.3778 - val_loss: 1.5796
Epoch 2/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 112s 48s/step - accuracy: 0.3219 - loss: 1.5890 - val_accuracy: 0.6778 - val_loss: 1.3846
Epoch 3/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 114s 46s/step - accuracy: 0.6226 - loss: 1.4361 - val_accuracy: 0.6667 - val_loss: 1.2689
Epoch 4/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 140s 46s/step - accuracy: 0.6690 - loss: 1.2910 - val_accuracy: 0.6667 - val_loss: 1.1098
Epoch 5/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 140s 59s/step - accuracy: 0.7220 - loss: 1.1739 - val_accuracy: 0.7111 - val_loss: 1.0277
```


block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense_5 (Dense)	(None, 1024)	525,312
dense_6 (Dense)	(None, 6)	6,150

Total params: 15,246,150 (58.16 MB)
Trainable params: 531,462 (2.03 MB)
Non-trainable params: 14,714,688 (56.13 MB)
Found 90 images belonging to 6 classes.
Found 90 images belonging to 6 classes.
Epoch 1/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 119s 47s/step - accuracy: 0.1648 - loss: 1.8706 - val_accuracy: 0.3778 - val_loss: 1.5796
Epoch 2/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 112s 48s/step - accuracy: 0.3219 - loss: 1.5890 - val_accuracy: 0.6778 - val_loss: 1.3846
Epoch 3/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 114s 46s/step - accuracy: 0.6226 - loss: 1.4361 - val_accuracy: 0.6667 - val_loss: 1.2689
Epoch 4/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 140s 46s/step - accuracy: 0.6690 - loss: 1.2910 - val_accuracy: 0.6667 - val_loss: 1.1098
Epoch 5/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 140s 59s/step - accuracy: 0.7220 - loss: 1.1739 - val_accuracy: 0.7111 - val_loss: 1.0277

```python
# Evaluate and Classification Report
import numpy as np
from sklearn.metrics import classification_report

# Predict and report
Y_pred = model.predict(val_generator_vgg)
y_pred = np.argmax(Y_pred, axis=1)
print("Classification Report:")
print(classification_report(val_generator_vgg.classes, y_pred, target_names=list(val_generator_vgg.class_indices.keys())))
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 57s 19s/step
Classification Report:
              precision    recall  f1-score   support

        acai       0.00      0.00      0.00        15
     cupuacu       0.12      0.20      0.15        15
    graviola       0.08      0.07      0.07        15
     guarana       0.40      0.40      0.40        15
     pupunha       0.10      0.07      0.08        15
      tucuma       0.20      0.27      0.23        15

    accuracy                           0.17        90
   macro avg       0.15      0.17      0.16        90
weighted avg       0.15      0.17      0.16        90
```

Training and Validation Accuracy (left):

Training accuracy (blue line) remains consistently at 1.0 (100%) across all epochs Validation accuracy (orange line) fluctuates between approximately 0.825 and 0.89 There's a noticeable dip in validation accuracy around epochs 6-9

Training and Validation Loss (right):

Training loss (blue line) remains very close to 0 throughout training Validation loss (orange line) fluctuates between approximately 0.3 and 0.55 in Image 1, while it ranges between 0.32 and 0.37 in Image 2

Image 1 also displays additional information in the console output:

"Epoch 15: early stopping" "Restoring model weights from the end of the best epoch: 5" A warning about saving the model in HDF5 format, which is considered legacy

in both plot but difference is one is in 5 epoch other is in 6

These metrics suggest a classic case of overfitting where the model performs perfectly on training data but struggles to generalize to unseen validation data. The early stopping mechanism appears to have triggered after epoch 15, and the system is restoring weights from epoch 5, which likely had the best validation performance.