Fruits Classification Training Script

Step 1. Check if CUDA Is Enabled

In [1]:
```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
print("Is CUDA available:", tf.test.is_built_with_cuda())
print("GPUs available:", tf.config.list_physical_devices('GPU'))
```

```
TensorFlow version: 2.10.0
Is CUDA available: True
GPUs available: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Step 2: Counting and Analyzing the Data

In [2]:
```
import os

def count_images(base_path, folder_name):
    path = os.path.join(base_path, folder_name)
    fruit_folders = [folder for folder in os.listdir(path) if os.path.isdir(os.path

    image_counts = {}

    for fruit in fruit_folders:
        fruit_folder_path = os.path.join(path, fruit)
        images = [img for img in os.listdir(fruit_folder_path) if img.endswith(('.p
        image_counts[fruit] = len(images)

    return image_counts

# Define paths
base_directory = 'Data'
train_folder = 'train'
test_folder = 'test'

# Get image counts
train_image_counts = count_images(base_directory, train_folder)
test_image_counts = count_images(base_directory, test_folder)

# Display the counts and calculate percentages
print("Image distribution across train and test folders:")
for fruit in train_image_counts:
    total_images = train_image_counts[fruit] + (test_image_counts[fruit] if fruit i
    test_percentage = (test_image_counts[fruit] / total_images * 100) if fruit in t
    print(f"{fruit}: Train = {train_image_counts[fruit]}, Test = {test_image_counts
          f"Test % = {test_percentage:.2f}%")
```

```
Image distribution across train and test folders:
AppleRed: Train = 83, Test = 20, Test % = 19.42%
Banana: Train = 560, Test = 140, Test % = 20.00%
Orange: Train = 560, Test = 140, Test % = 20.00%
Pineapple: Train = 560, Test = 140, Test % = 20.00%
Pomelo: Train = 560, Test = 140, Test % = 20.00%
```

Step 3: Set Up Data Augmentation and Data Generators

In [3]:

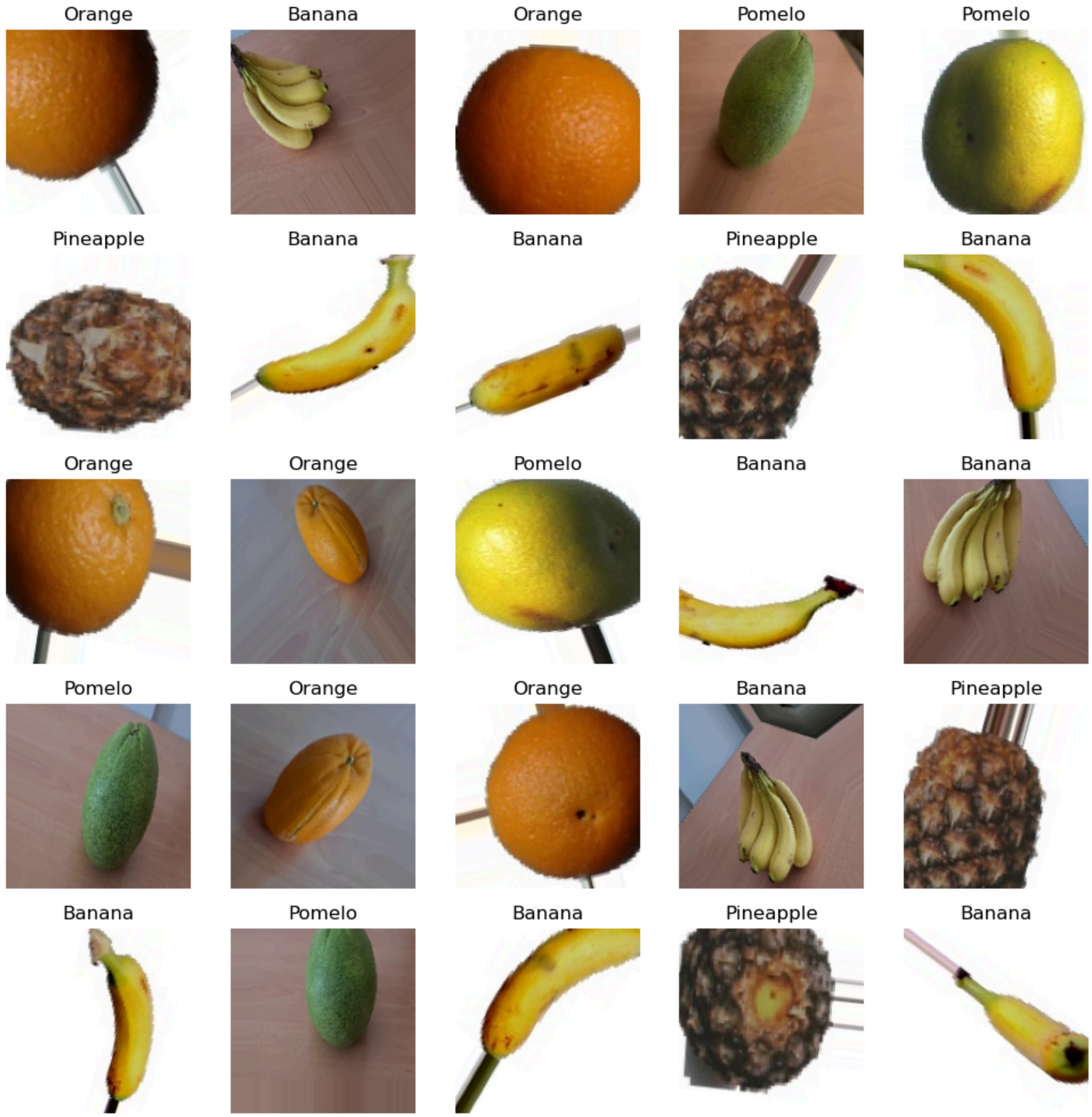
```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the path to your dataset
base_dir = 'Data'
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')

# Set up data augmentation configuration
train_datagen = ImageDataGenerator(
    rescale=1./255,         # Normalize pixel values
    rotation_range=40,      # Random rotations
    width_shift_range=0.2, # Random horizontal shifts
    height_shift_range=0.2,# Random vertical shifts
    shear_range=0.2,        # Shear transformations
    zoom_range=0.2,         # Random zoom
    horizontal_flip=True,   # Random horizontal flips
    fill_mode='nearest'     # Strategy for filling newly created pixels
)

test_datagen = ImageDataGenerator(rescale=1./255)  # Only rescale for testing data

# Prepare data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),  # Resize images to 150x150
    batch_size=32,
    class_mode='categorical'  # Multi-class labels
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 2323 images belonging to 5 classes.
Found 580 images belonging to 5 classes.
```

4. Visualizing Augmented Images

In [4]:

```
import matplotlib.pyplot as plt
import numpy as np

# Function to plot images in a 5x5 grid with labels
def plot_images(images_arr, labels_arr):
    fig, axes = plt.subplots(5, 5, figsize=(10, 10))  # Increase the subplot size t
    axes = axes.flatten()
    for img, label, ax in zip(images_arr, labels_arr, axes):
        ax.imshow(img)
        ax.axis('off')
        ax.set_title(label)
    plt.tight_layout()
    plt.show()

# Get multiple batches to increase the likelihood of variety
images = []
labels = []
for _ in range(5):  # Get 5 batches to have a variety, each batch typically has 32
    imgs, lbls = next(train_generator)
    images.extend(imgs)
    labels.extend(lbls)

# Choose 25 images randomly to display
indices = np.random.choice(range(len(images)), 25, replace=False)
selected_images = np.array(images)[indices]
selected_labels = np.array(labels)[indices]

# Convert one-hot labels to class names
class_names = list(train_generator.class_indices.keys())  # Get class names from th
selected_labels = [class_names[np.argmax(label)] for label in selected_labels]

# Plot the selected images and their labels
plot_images(selected_images, selected_labels)
```



Step 5: Defining the CNN Model Architecture

In [12]:

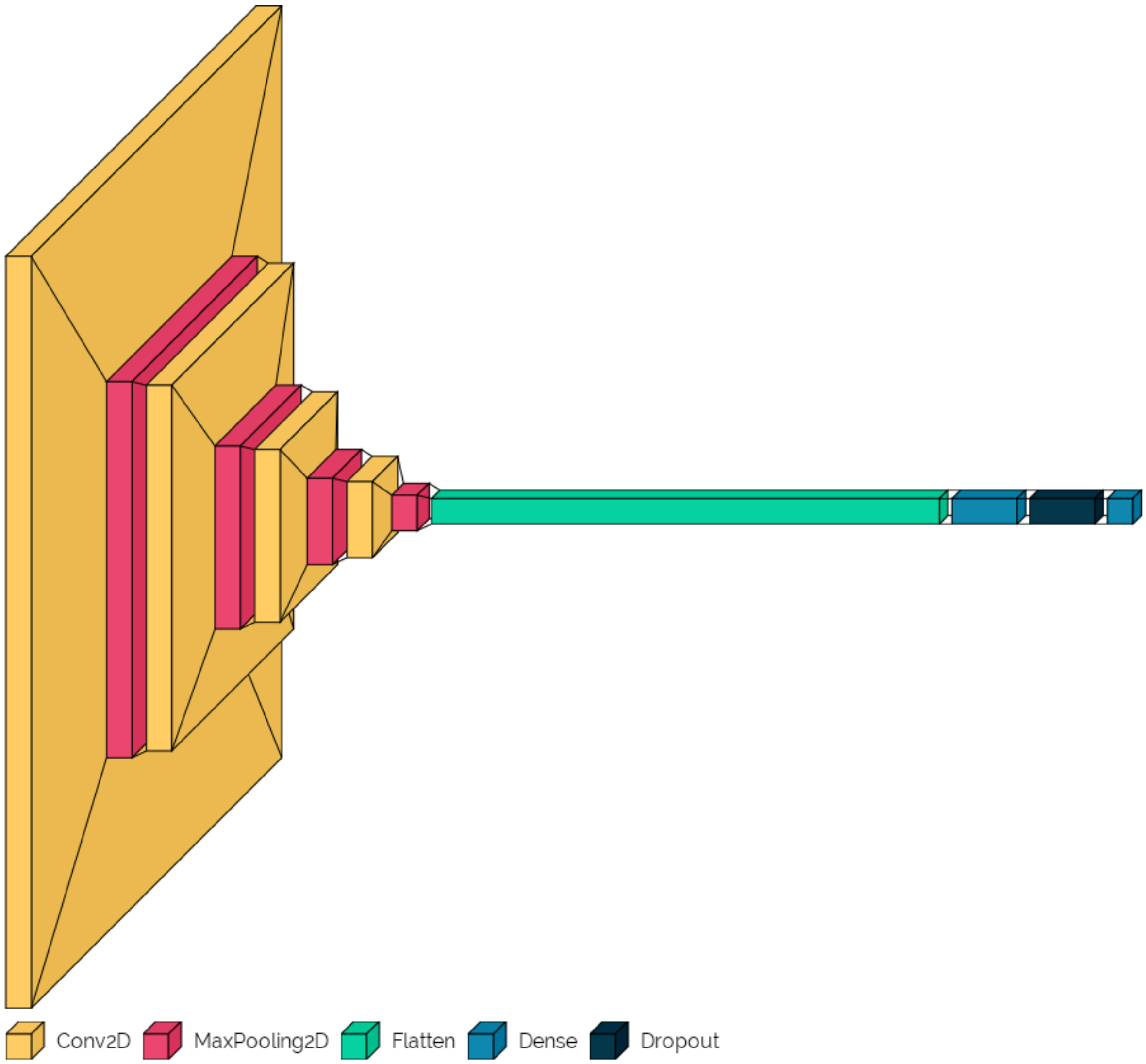
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Define the model
model = Sequential([
    # First convolutional layer
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2, 2),
    # Second convolutional layer
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2, 2),
    # Third convolutional layer
```

```python
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2, 2),
    # Fourth convolutional layer
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2, 2),
    # Flatten the results to feed into a dense layer
    Flatten(),
    # 512 neuron hidden layer
    Dense(512, activation='relu'),
    Dropout(0.5),
    # Output layer with a single neuron for each class
    Dense(len(train_generator.class_indices), activation='softmax')
])

# Model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 148, 148, 32)      896

 max_pooling2d_4 (MaxPooling  (None, 74, 74, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_6 (MaxPooling  (None, 17, 17, 128)      0
 2D)

 conv2d_7 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_7 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 flatten_1 (Flatten)         (None, 6272)              0

 dense_2 (Dense)             (None, 512)               3211776

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,455,173
Trainable params: 3,455,173
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```python
from PIL import ImageFont
import visualkeras

# Load a specific font
font = ImageFont.truetype('font.ttf', 16)  # Adjust the path and size as needed

# Generate the model visual with a custom font
visualkeras.layered_view(model, legend=True, font=font)
```

Out[13]:



Step 6. Compile the Model

In [14]:
```python
from tensorflow.keras.optimizers import RMSprop

# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(learning_rate=0.001),
              metrics=['accuracy'])
```

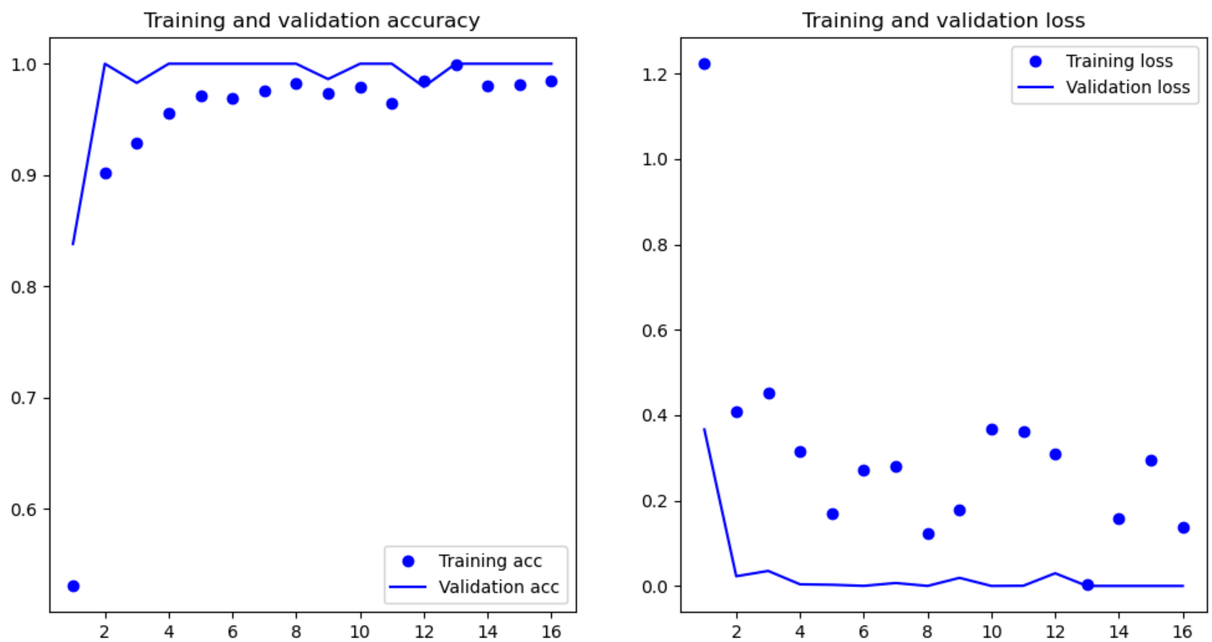
Step 7: Train the CNN Model

Code to Train the Model and Plot Performance

In [15]:

```
import matplotlib.pyplot as plt

from tensorflow.keras.callbacks import EarlyStopping

# Define the Early Stopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor the validation loss
    patience=3,          # Number of epochs with no improvement after which trainin
    verbose=1,
    restore_best_weights=True  # Restores model weights from the epoch with the bes
)

# Fit the model with Early Stopping
history = model.fit(
    train_generator,
    epochs=20,  # Maximum number of epochs (might stop earlier)
    validation_data=test_generator,
    verbose=1,
    callbacks=[early_stopping]  # Include the Early Stopping callback
)

# Function to plot training and validation accuracy and loss
def plot_training_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1, len(acc) + 1)

    # Plot accuracy
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()

# Call the function to plot the training and validation accuracy and loss
plot_training_history(history)
```

```
Epoch 1/20
73/73 [==============================] - 19s 249ms/step - loss: 1.2232 - accuracy:
0.5303 - val_loss: 0.3665 - val_accuracy: 0.8379
Epoch 2/20
73/73 [==============================] - 18s 243ms/step - loss: 0.4079 - accuracy:
0.9019 - val_loss: 0.0227 - val_accuracy: 1.0000
Epoch 3/20
73/73 [==============================] - 19s 253ms/step - loss: 0.4512 - accuracy:
0.9290 - val_loss: 0.0352 - val_accuracy: 0.9828
Epoch 4/20
73/73 [==============================] - 18s 247ms/step - loss: 0.3158 - accuracy:
0.9557 - val_loss: 0.0038 - val_accuracy: 1.0000
Epoch 5/20
73/73 [==============================] - 19s 252ms/step - loss: 0.1692 - accuracy:
0.9712 - val_loss: 0.0028 - val_accuracy: 1.0000
Epoch 6/20
73/73 [==============================] - 18s 251ms/step - loss: 0.2712 - accuracy:
0.9686 - val_loss: 2.2949e-04 - val_accuracy: 1.0000
Epoch 7/20
73/73 [==============================] - 18s 239ms/step - loss: 0.2793 - accuracy:
0.9759 - val_loss: 0.0069 - val_accuracy: 1.0000
Epoch 8/20
73/73 [==============================] - 16s 224ms/step - loss: 0.1241 - accuracy:
0.9819 - val_loss: 6.4933e-05 - val_accuracy: 1.0000
Epoch 9/20
73/73 [==============================] - 17s 228ms/step - loss: 0.1785 - accuracy:
0.9737 - val_loss: 0.0190 - val_accuracy: 0.9862
Epoch 10/20
73/73 [==============================] - 16s 224ms/step - loss: 0.3691 - accuracy:
0.9789 - val_loss: 1.8114e-05 - val_accuracy: 1.0000
Epoch 11/20
73/73 [==============================] - 17s 229ms/step - loss: 0.3631 - accuracy:
0.9647 - val_loss: 4.2530e-04 - val_accuracy: 1.0000
Epoch 12/20
73/73 [==============================] - 16s 222ms/step - loss: 0.3090 - accuracy:
0.9845 - val_loss: 0.0297 - val_accuracy: 0.9793
Epoch 13/20
73/73 [==============================] - 17s 232ms/step - loss: 0.0043 - accuracy:
0.9987 - val_loss: 1.9712e-06 - val_accuracy: 1.0000
Epoch 14/20
73/73 [==============================] - 17s 226ms/step - loss: 0.1578 - accuracy:
0.9802 - val_loss: 1.4396e-05 - val_accuracy: 1.0000
Epoch 15/20
73/73 [==============================] - 16s 224ms/step - loss: 0.2946 - accuracy:
0.9815 - val_loss: 3.8292e-06 - val_accuracy: 1.0000
Epoch 16/20
73/73 [==============================] - ETA: 0s - loss: 0.1369 - accuracy: 0.9845Re
storing model weights from the end of the best epoch: 13.
73/73 [==============================] - 17s 226ms/step - loss: 0.1369 - accuracy:
0.9845 - val_loss: 6.5129e-06 - val_accuracy: 1.0000
Epoch 16: early stopping
```



Save Model

In [16]:
```
# Save the model
model.save('fruit_classifier_model.h5')  # Saves the model in HDF5 format
```

Step 8: Load and Use the Model for Predictions

In [24]:
```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import os
import random

# Load the saved model
model = load_model('fruit_classifier_model.h5')

# Function to load and prepare the image
def load_and_prepare_image(file_path):
    img = image.load_img(file_path, target_size=(150, 150))
    img_array = image.img_to_array(img)
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    return img_array_expanded_dims / 255.0

# Base directory for test images
base_dir = 'Data/test'

# Get class directories
class_directories = [os.path.join(base_dir, d) for d in os.listdir(base_dir) if os.

# Iterate over each class directory
```

```python
for class_dir in class_directories:
    # Get all files in directory
    files = [os.path.join(class_dir, f) for f in os.listdir(class_dir) if f.endswit

    # Select a random file
    random_file = random.choice(files)

    # Prepare the image
    prepared_image = load_and_prepare_image(random_file)

    # Make a prediction
    predictions = model.predict(prepared_image)
    predicted_class_index = np.argmax(predictions[0])
    confidence = predictions[0][predicted_class_index]

    # Get class name from directory
    class_name = os.path.basename(class_dir)

    # Display the prediction and confidence
    print(f"Class: {class_name} - Predicted fruit: {class_labels[predicted_class_in
```

```
1/1 [==============================] - 0s 115ms/step
Class: AppleRed - Predicted fruit: AppleRed with Confidence: 1.00%
1/1 [==============================] - 0s 59ms/step
Class: Banana - Predicted fruit: Banana with Confidence: 1.00%
1/1 [==============================] - 0s 20ms/step
Class: Orange - Predicted fruit: Orange with Confidence: 1.00%
1/1 [==============================] - 0s 20ms/step
Class: Pineapple - Predicted fruit: Pineapple with Confidence: 1.00%
1/1 [==============================] - 0s 19ms/step
Class: Pomelo - Predicted fruit: Pomelo with Confidence: 1.00%
```