

















Fruits Classification by ResNet

Table of Contents

Project Overview 	2
Project Objectives 	2
Technologies & Techniques 	2
Dataset Preparation	3
Data Loading 	3
Data Preprocessing & Augmentation 	3
Data Splitting 	3
Model Architecture: Transfer Learning with ResNet	3
Introduction to Convolutional Neural Networks (CNNs) 	3
ResNet and Residual Learning 	3
Customizing the Model for Fruit Classification 	4
Training Process 	4
Compilation and Loss Functions 	4
Optimization Techniques 	4
Callbacks and Regularization 	4
Evaluation and Results 	5
Metrics and Visualizations	5
Inference & Usage 	5
Full Source Code 	5
Step 1 Libraries	5
Step 2 Load Data	6
Step 3 Name of Classes	6
Step 4 Showing Images	7
Step 5 Configure Dataset For Performance	8
Step 6 Data Augmentation	9
Step 7 ResNET Model For Transfer Learning	9
Step 8 Build Model	9
Step 9 Train Model	18
Step 10 Accuracy Plot	19
Step 11 Loss Plot	20
Step 12 Predict Images	20
Conclusion & Future Work 	22
Conclusion	22
Future Work	23

Project Overview

The **Fruits Classification by ResNet** project demonstrates the power of transfer learning by applying a pre-trained ResNet model to classify images of various fruits. By leveraging advanced deep learning techniques and data augmentation, this project provides an end-to-end solution—from data preprocessing to model evaluation and real-world inference.

Project Objectives

- **Accurate Classification:** Develop a model capable of distinguishing between different fruit types.
 - **Efficient Use of Transfer Learning:** Utilize a pre-trained ResNet model to reduce training time and improve feature extraction.
 - **Robust Data Handling:** Implement thorough data preprocessing and augmentation strategies to enhance model performance.
 - **Comprehensive Evaluation:** Analyze model performance using multiple metrics and visualization techniques.
 - **Scalable Inference:** Provide a framework to deploy the model for real-time fruit classification.
-

Technologies & Techniques

- **Deep Learning & Transfer Learning:** Utilizing pre-trained models (ResNet) to harness learned features from large datasets.
 - **Convolutional Neural Networks (CNNs):** A class of deep neural networks that excels in processing image data.
 - **Data Augmentation:** Techniques like rotations, flips, and scaling to artificially expand the training dataset.
 - **Optimization Algorithms:** Use of optimizers (e.g., Adam) to minimize the loss function during training.
 - **Regularization & Callbacks:** Techniques such as early stopping and learning rate scheduling to prevent overfitting and improve training efficiency.
 - **Visualization Tools:** Libraries like Matplotlib for plotting training progress, loss curves, and performance metrics.
-

Dataset Preparation

Data Loading 📁

- **Source:** Fruit image dataset containing various classes.
- **Process:** Images are loaded from directories, using libraries such as `pandas` for metadata handling and image processing libraries (e.g., `PIL` or `OpenCV`) for reading image files.

Data Preprocessing & Augmentation 🛠️

- **Resizing & Normalization:** Images are resized to a uniform shape and normalized to ensure consistency.
- **Augmentation Techniques:**
 - **Rotations & Flips:** Randomly rotate and flip images to simulate different orientations.
 - **Scaling & Cropping:** Adjust image scales and perform random crops to create diversity.
 - **Why Augment?** Augmentation increases the dataset size and diversity, helping the model generalize better.

Data Splitting ✂️

- **Training, Validation, Testing:** The dataset is split into three subsets to ensure the model is trained, validated, and tested on distinct data, reducing overfitting and ensuring robust evaluation.

Model Architecture: Transfer Learning with ResNet

Introduction to Convolutional Neural Networks (CNNs) 🤖

- **CNN Basics:** CNNs consist of convolutional layers that automatically learn spatial hierarchies of features from images.
- **Key Components:**
 - **Convolutional Layers:** Extract features using filters.
 - **Pooling Layers:** Reduce spatial dimensions while retaining important information.
 - **Fully Connected Layers:** Interpret features for classification tasks.

ResNet and Residual Learning 🔄

- **What is ResNet?**
 - ResNet (Residual Network) introduces skip connections or residual blocks to alleviate the vanishing gradient problem in deep networks.

- **Residual Blocks:** These blocks allow the model to learn identity functions, making it easier to optimize deeper networks.
- **Benefits:**
 - Improved training stability.
 - Better accuracy through deep architectures without degradation.

Customizing the Model for Fruit Classification 🍇

- **Pre-trained Backbone:** The model uses a ResNet (e.g., ResNet-50) pre-trained on a large-scale dataset (like ImageNet) to leverage already-learned features.
- **Adding Custom Layers:**
 - **Global Average Pooling:** Reduces each feature map to a single number, summarizing the presence of features.
 - **Dense Layers:** A fully connected layer to interpret these features and output probabilities for each fruit class.
- **Fine-Tuning:**
 - **Freezing Layers:** Initially, lower layers of ResNet are frozen to preserve learned features.
 - **Unfreezing:** Later, selected layers may be unfrozen for fine-tuning on the fruit dataset.

Training Process 🔥

Compilation and Loss Functions 📄

- **Loss Function:** Categorical Cross-Entropy is used for multi-class classification.
- **Compilation:** The model is compiled with an optimizer (commonly Adam) to minimize the loss function.

Optimization Techniques ⚙️

- **Adam Optimizer:** Combines the benefits of RMSProp and Momentum, offering adaptive learning rates.
- **Batch Training:** Training is performed in mini-batches, improving computational efficiency and stability.

Callbacks and Regularization ⌚

- **Early Stopping:** Monitors validation loss and stops training when performance ceases to improve, preventing overfitting.
- **Learning Rate Scheduling:** Adjusts the learning rate during training to help the model converge faster.
- **Dropout:** Optional dropout layers might be used in custom layers to further reduce overfitting.

Evaluation and Results

Metrics and Visualizations

- **Accuracy & Loss Curves:**
 - Training and validation accuracy curves help assess model performance over epochs.
 - Loss curves indicate how well the model is learning and if overfitting occurs.
 - **Confusion Matrix:**
 - Provides insights into which fruit classes are misclassified, guiding further improvements.
 - **Visualization Tools:**
 - **Matplotlib/Seaborn:** Used to generate plots for training progress, confusion matrices, and sample predictions.
-

Inference & Usage

- **Real-Time Predictions:** Once trained, the model can classify new fruit images by preprocessing them in the same way as the training data.
 - **Deployment Considerations:**
 - The model can be exported and deployed in web or mobile applications.
 - Integration with REST APIs for real-time inference is possible.
 - **Usage Flow:**
 1. Load the trained model.
 2. Preprocess the incoming image.
 3. Predict the fruit class and display the result.
-

Full Source Code

Step 1 | Libraries

```
# import libraries
import os
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import layers, callbacks
import warnings
warnings.filterwarnings("ignore")
```

Step 2 | Load Data

```
# Load dataset
train_dir = '/kaggle/input/fruits/fruits-360_dataset_100x100/fruits-360/Training'
test_dir = '/kaggle/input/fruits/fruits-360_dataset_100x100/fruits-360/Test'
```

```
# Load training dataset without preprocessing
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    validation_split=0.2,
    subset='training',
    batch_size=32,
    image_size=(100, 100),
    seed=123,
    shuffle=True,
)
```

```
# Load validation dataset without preprocessing
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    validation_split=0.2,
    subset='validation',
    batch_size=32,
    image_size=(100, 100),
    seed=42,
)
```

```
Found 70491 files belonging to 141 classes.
Using 56393 files for training.
Found 70491 files belonging to 141 classes.
Using 14098 files for validation.
```

Step 3 | Name of Classes

```
# Get the class names from the subdirectories in the training directory
class_names = sorted(os.listdir(train_dir))
```

```
# Print the class names and the number of classes
print("Class Names:", class_names)
num_classes = len(class_names)
print("Number of Classes:", num_classes)
```

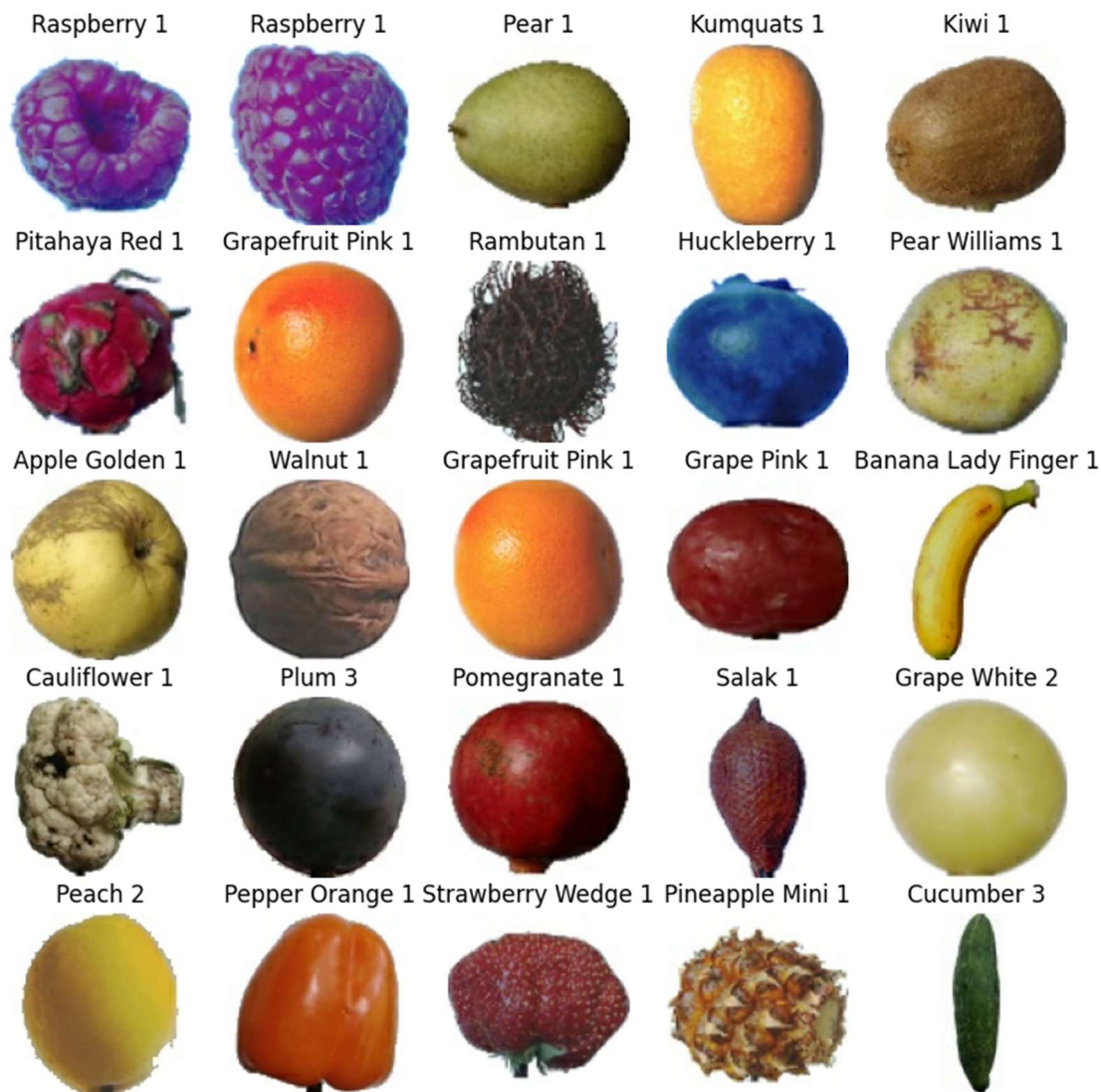
```
Class Names: ['Apple 6', 'Apple Braeburn 1', 'Apple Crimson Snow 1',
'Apple Golden 1', 'Apple Golden 2', 'Apple Golden 3', 'Apple Granny Smith
1', 'Apple Pink Lady 1', 'Apple Red 1', 'Apple Red 2', 'Apple Red 3',
'Apple Red Delicious 1', 'Apple Red Yellow 1', 'Apple Red Yellow 2',
```

```
'Apple hit 1', 'Apricot 1', 'Avocado 1', 'Avocado ripe 1', 'Banana 1',
'Banana Lady Finger 1', 'Banana Red 1', 'Beetroot 1', 'Blueberry 1',
'Cabbage white 1', 'Cactus fruit 1', 'Cantaloupe 1', 'Cantaloupe 2',
'Carambula 1', 'Carrot 1', 'Cauliflower 1', 'Cherry 1', 'Cherry 2',
'Cherry Rainier 1', 'Cherry Wax Black 1', 'Cherry Wax Red 1', 'Cherry Wax
Yellow 1', 'Chestnut 1', 'Clementine 1', 'Cocos 1', 'Corn 1', 'Corn Husk
1', 'Cucumber 1', 'Cucumber 3', 'Cucumber Ripe 1', 'Cucumber Ripe 2',
'Dates 1', 'Eggplant 1', 'Eggplant long 1', 'Fig 1', 'Ginger Root 1',
'Granadilla 1', 'Grape Blue 1', 'Grape Pink 1', 'Grape White 1', 'Grape
White 2', 'Grape White 3', 'Grape White 4', 'Grapefruit Pink 1',
'Grapefruit White 1', 'Guava 1', 'Hazelnut 1', 'Huckleberry 1', 'Kaki 1',
'Kiwi 1', 'Kohlrabi 1', 'Kumquats 1', 'Lemon 1', 'Lemon Meyer 1', 'Limes
1', 'Lychee 1', 'Mandarine 1', 'Mango 1', 'Mango Red 1', 'Mangostan 1',
'Maracuja 1', 'Melon Piel de Sapo 1', 'Mulberry 1', 'Nectarine 1',
'Nectarine Flat 1', 'Nut Forest 1', 'Nut Pecan 1', 'Onion Red 1', 'Onion
Red Peeled 1', 'Onion White 1', 'Orange 1', 'Papaya 1', 'Passion Fruit 1',
'Peach 1', 'Peach 2', 'Peach Flat 1', 'Pear 1', 'Pear 2', 'Pear 3', 'Pear
Abate 1', 'Pear Forelle 1', 'Pear Kaiser 1', 'Pear Monster 1', 'Pear Red
1', 'Pear Stone 1', 'Pear Williams 1', 'Pepino 1', 'Pepper Green 1',
'Pepper Orange 1', 'Pepper Red 1', 'Pepper Yellow 1', 'Physalis 1',
'Physalis with Husk 1', 'Pineapple 1', 'Pineapple Mini 1', 'Pitahaya Red
1', 'Plum 1', 'Plum 2', 'Plum 3', 'Pomegranate 1', 'Pomelo Sweetie 1',
'Potato Red 1', 'Potato Red Washed 1', 'Potato Sweet 1', 'Potato White 1',
'Quince 1', 'Rambutan 1', 'Raspberry 1', 'Redcurrant 1', 'Salak 1',
'Strawberry 1', 'Strawberry Wedge 1', 'Tamarillo 1', 'Tangelo 1', 'Tomato
1', 'Tomato 2', 'Tomato 3', 'Tomato 4', 'Tomato Cherry Red 1', 'Tomato
Heart 1', 'Tomato Maroon 1', 'Tomato Yellow 1', 'Tomato not Ripened 1',
'Walnut 1', 'Watermelon 1', 'Zucchini 1', 'Zucchini dark 1']
Number of Classes: 141
```

Step 4 | Showing Images

```
# Function to display images from a dataset
def show_images(dataset, class_names):
    plt.figure(figsize=(10, 10))
    for images, labels in dataset.take(1): # Take one batch from the
dataset
        for i in range(25): # Display 25 images
            ax = plt.subplot(5, 5, i + 1)
            plt.imshow(images[i].numpy().astype("uint8"))
            plt.title(class_names[labels[i]])
            plt.axis("off")

# Show images from training dataset
show_images(train_ds, class_names)
plt.show()
```



Step 5 | Configure Dataset For Performance

The variable **AUTOTUNE** in TensorFlow's `tf.data.experimental` module is used to dynamically tune the buffer size during data processing for optimal performance. In the given code, the training and validation datasets are cached and pre-fetched using **AUTOTUNE** as the buffer size, enabling faster data loading and processing during model training.

for faster training which takes less time and less memory for training
`AUTOTUNE = tf.data.experimental.AUTOTUNE`



```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```


Step 6 | Data Augmentation

When you don't have a large image dataset, it's a good practice to artificially introduce sample diversity by applying random, yet realistic, transformations to the training images, such as rotation and horizontal flipping. This helps expose the model to different aspects of the training data and reduce [overfitting](#). You can learn more about data augmentation in this [tutorial](#).

```
# Augmentation for training dataset
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2)
])
```

Step 7 | ResNet Model For Transfer Learning

 **ResNet, short for Residual Network, is a popular model for transfer learning in deep learning. Its architecture, with skip connections and residual blocks, allows for efficient training of deep neural networks, making it highly effective in tasks such as image classification and object detection.** 

```
# Using resNET model of transfer Learning for training
base_model = tf.keras.applications.ResNet50(include_top=False,
weights='imagenet', input_shape=(100, 100, 3))
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ————— 0s 0us/step

```
preprocess_input = tf.keras.applications.resnet.preprocess_input
base_model.trainable = False
```

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(num_classes)
```

Step 8 | Build Model

```
inputs = tf.keras.Input(shape=(100, 100, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)

model.compile(
    optimizer=optimizer,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

Let's take a look at the base model architecture

```
base_model.summary()
```

Model: "resnet50"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100, 100, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 106, 106, 3)	0	input_layer[0][0]
conv1_conv (Conv2D)	(None, 50, 50, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalizatio...	(None, 50, 50, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 50, 50, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 52, 52, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 25, 25, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 25, 25, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalizatio...	(None, 25, 25, 64)	256	conv2_block1_1_c...
conv2_block1_1_relu (Activation)	(None, 25, 25, 64)	0	conv2_block1_1_b...
conv2_block1_2_conv (Conv2D)	(None, 25, 25, 64)	36,928	conv2_block1_1_r...
conv2_block1_2_bn (BatchNormalizatio...	(None, 25, 25, 64)	256	conv2_block1_2_c...
conv2_block1_2_relu (Activation)	(None, 25, 25, 64)	0	conv2_block1_2_b...
conv2_block1_0_conv (Conv2D)	(None, 25, 25, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 25, 25, 256)	16,640	conv2_block1_2_r...
conv2_block1_0_bn (BatchNormalizatio...	(None, 25, 25, 256)	1,024	conv2_block1_0_c...

conv2_block1_3_bn (BatchNormalizatio...	(None, 25, 25, 256)	1,024	conv2_block1_3_c...
conv2_block1_add (Add)	(None, 25, 25, 256)	0	conv2_block1_0_b... conv2_block1_3_b...
conv2_block1_out (Activation)	(None, 25, 25, 256)	0	conv2_block1_add...
conv2_block2_1_conv (Conv2D)	(None, 25, 25, 64)	16,448	conv2_block1_out...
conv2_block2_1_bn (BatchNormalizatio...	(None, 25, 25, 64)	256	conv2_block2_1_c...
conv2_block2_1_relu (Activation)	(None, 25, 25, 64)	0	conv2_block2_1_b...
conv2_block2_2_conv (Conv2D)	(None, 25, 25, 64)	36,928	conv2_block2_1_r...
conv2_block2_2_bn (BatchNormalizatio...	(None, 25, 25, 64)	256	conv2_block2_2_c...
conv2_block2_2_relu (Activation)	(None, 25, 25, 64)	0	conv2_block2_2_b...
conv2_block2_3_conv (Conv2D)	(None, 25, 25, 256)	16,640	conv2_block2_2_r...
conv2_block2_3_bn (BatchNormalizatio...	(None, 25, 25, 256)	1,024	conv2_block2_3_c...
conv2_block2_add (Add)	(None, 25, 25, 256)	0	conv2_block1_out... conv2_block2_3_b...
conv2_block2_out (Activation)	(None, 25, 25, 256)	0	conv2_block2_add...
conv2_block3_1_conv (Conv2D)	(None, 25, 25, 64)	16,448	conv2_block2_out...
conv2_block3_1_bn (BatchNormalizatio...	(None, 25, 25, 64)	256	conv2_block3_1_c...
conv2_block3_1_relu (Activation)	(None, 25, 25, 64)	0	conv2_block3_1_b...
conv2_block3_2_conv (Conv2D)	(None, 25, 25, 64)	36,928	conv2_block3_1_r...
conv2_block3_2_bn (BatchNormalizatio...	(None, 25, 25, 64)	256	conv2_block3_2_c...
conv2_block3_2_relu (Activation)	(None, 25, 25, 64)	0	conv2_block3_2_b...
conv2_block3_3_conv (Conv2D)	(None, 25, 25, 256)	16,640	conv2_block3_2_r...
conv2_block3_3_bn (BatchNormalizatio...	(None, 25, 25, 256)	1,024	conv2_block3_3_c...
conv2_block3_add (Add)	(None, 25, 25, 256)	0	conv2_block2_out... conv2_block3_3_b...

conv2_block3_out (Activation)	(None, 25, 25, 256)	0	conv2_block3_add...
conv3_block1_1_conv (Conv2D)	(None, 13, 13, 128)	32,896	conv2_block3_out...
conv3_block1_1_bn (BatchNormalizatio...	(None, 13, 13, 128)	512	conv3_block1_1_c...
conv3_block1_1_relu (Activation)	(None, 13, 13, 128)	0	conv3_block1_1_b...
conv3_block1_2_conv (Conv2D)	(None, 13, 13, 128)	147,584	conv3_block1_1_r...
conv3_block1_2_bn (BatchNormalizatio...	(None, 13, 13, 128)	512	conv3_block1_2_c...
conv3_block1_2_relu (Activation)	(None, 13, 13, 128)	0	conv3_block1_2_b...
conv3_block1_0_conv (Conv2D)	(None, 13, 13, 512)	131,584	conv2_block3_out...
conv3_block1_3_conv (Conv2D)	(None, 13, 13, 512)	66,048	conv3_block1_2_r...
conv3_block1_0_bn (BatchNormalizatio...	(None, 13, 13, 512)	2,048	conv3_block1_0_c...
conv3_block1_3_bn (BatchNormalizatio...	(None, 13, 13, 512)	2,048	conv3_block1_3_c...
conv3_block1_add (Add)	(None, 13, 13, 512)	0	conv3_block1_0_b... conv3_block1_3_b...
conv3_block1_out (Activation)	(None, 13, 13, 512)	0	conv3_block1_add...
conv3_block2_1_conv (Conv2D)	(None, 13, 13, 128)	65,664	conv3_block1_out...
conv3_block2_1_bn (BatchNormalizatio...	(None, 13, 13, 128)	512	conv3_block2_1_c...
conv3_block2_1_relu (Activation)	(None, 13, 13, 128)	0	conv3_block2_1_b...
conv3_block2_2_conv (Conv2D)	(None, 13, 13, 128)	147,584	conv3_block2_1_r...
conv3_block2_2_bn (BatchNormalizatio...	(None, 13, 13, 128)	512	conv3_block2_2_c...
conv3_block2_2_relu (Activation)	(None, 13, 13, 128)	0	conv3_block2_2_b...
conv3_block2_3_conv (Conv2D)	(None, 13, 13, 512)	66,048	conv3_block2_2_r...
conv3_block2_3_bn (BatchNormalizatio...	(None, 13, 13, 512)	2,048	conv3_block2_3_c...
conv3_block2_add (Add)	(None, 13, 13, 512)	0	conv3_block1_out... conv3_block2_3_b...

conv3_block2_out (Activation)	(None, 13, 13, 512)	0	conv3_block2_add...
conv3_block3_1_conv (Conv2D)	(None, 13, 13, 128)	65,664	conv3_block2_out...
conv3_block3_1_bn (BatchNormalizatio...	(None, 13, 13, 128)	512	conv3_block3_1_c...
conv3_block3_1_relu (Activation)	(None, 13, 13, 128)	0	conv3_block3_1_b...
conv3_block3_2_conv (Conv2D)	(None, 13, 13, 128)	147,584	conv3_block3_1_r...
conv3_block3_2_bn (BatchNormalizatio...	(None, 13, 13, 128)	512	conv3_block3_2_c...
conv3_block3_2_relu (Activation)	(None, 13, 13, 128)	0	conv3_block3_2_b...
conv3_block3_3_conv (Conv2D)	(None, 13, 13, 512)	66,048	conv3_block3_2_r...
conv3_block3_3_bn (BatchNormalizatio...	(None, 13, 13, 512)	2,048	conv3_block3_3_c...
conv3_block3_add (Add)	(None, 13, 13, 512)	0	conv3_block2_out... conv3_block3_3_b...
conv3_block3_out (Activation)	(None, 13, 13, 512)	0	conv3_block3_add...
conv3_block4_1_conv (Conv2D)	(None, 13, 13, 128)	65,664	conv3_block3_out...
conv3_block4_1_bn (BatchNormalizatio...	(None, 13, 13, 128)	512	conv3_block4_1_c...
conv3_block4_1_relu (Activation)	(None, 13, 13, 128)	0	conv3_block4_1_b...
conv3_block4_2_conv (Conv2D)	(None, 13, 13, 128)	147,584	conv3_block4_1_r...
conv3_block4_2_bn (BatchNormalizatio...	(None, 13, 13, 128)	512	conv3_block4_2_c...
conv3_block4_2_relu (Activation)	(None, 13, 13, 128)	0	conv3_block4_2_b...
conv3_block4_3_conv (Conv2D)	(None, 13, 13, 512)	66,048	conv3_block4_2_r...
conv3_block4_3_bn (BatchNormalizatio...	(None, 13, 13, 512)	2,048	conv3_block4_3_c...
conv3_block4_add (Add)	(None, 13, 13, 512)	0	conv3_block3_out... conv3_block4_3_b...
conv3_block4_out (Activation)	(None, 13, 13, 512)	0	conv3_block4_add...
conv4_block1_1_conv (Conv2D)	(None, 7, 7, 256)	131,328	conv3_block4_out...

conv4_block1_1_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block1_1_c...
conv4_block1_1_relu (Activation)	(None, 7, 7, 256)	0	conv4_block1_1_b...
conv4_block1_2_conv (Conv2D)	(None, 7, 7, 256)	590,080	conv4_block1_1_r...
conv4_block1_2_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block1_2_c...
conv4_block1_2_relu (Activation)	(None, 7, 7, 256)	0	conv4_block1_2_b...
conv4_block1_0_conv (Conv2D)	(None, 7, 7, 1024)	525,312	conv3_block4_out...
conv4_block1_3_conv (Conv2D)	(None, 7, 7, 1024)	263,168	conv4_block1_2_r...
conv4_block1_0_bn (BatchNormalizatio...	(None, 7, 7, 1024)	4,096	conv4_block1_0_c...
conv4_block1_3_bn (BatchNormalizatio...	(None, 7, 7, 1024)	4,096	conv4_block1_3_c...
conv4_block1_add (Add)	(None, 7, 7, 1024)	0	conv4_block1_0_b... conv4_block1_3_b...
conv4_block1_out (Activation)	(None, 7, 7, 1024)	0	conv4_block1_add...
conv4_block2_1_conv (Conv2D)	(None, 7, 7, 256)	262,400	conv4_block1_out...
conv4_block2_1_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block2_1_c...
conv4_block2_1_relu (Activation)	(None, 7, 7, 256)	0	conv4_block2_1_b...
conv4_block2_2_conv (Conv2D)	(None, 7, 7, 256)	590,080	conv4_block2_1_r...
conv4_block2_2_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block2_2_c...
conv4_block2_2_relu (Activation)	(None, 7, 7, 256)	0	conv4_block2_2_b...
conv4_block2_3_conv (Conv2D)	(None, 7, 7, 1024)	263,168	conv4_block2_2_r...
conv4_block2_3_bn (BatchNormalizatio...	(None, 7, 7, 1024)	4,096	conv4_block2_3_c...
conv4_block2_add (Add)	(None, 7, 7, 1024)	0	conv4_block1_out... conv4_block2_3_b...
conv4_block2_out (Activation)	(None, 7, 7, 1024)	0	conv4_block2_add...
conv4_block3_1_conv (Conv2D)	(None, 7, 7, 256)	262,400	conv4_block2_out...

conv4_block3_1_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block3_1_c...
conv4_block3_1_relu (Activation)	(None, 7, 7, 256)	0	conv4_block3_1_b...
conv4_block3_2_conv (Conv2D)	(None, 7, 7, 256)	590,080	conv4_block3_1_r...
conv4_block3_2_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block3_2_c...
conv4_block3_2_relu (Activation)	(None, 7, 7, 256)	0	conv4_block3_2_b...
conv4_block3_3_conv (Conv2D)	(None, 7, 7, 1024)	263,168	conv4_block3_2_r...
conv4_block3_3_bn (BatchNormalizatio...	(None, 7, 7, 1024)	4,096	conv4_block3_3_c...
conv4_block3_add (Add)	(None, 7, 7, 1024)	0	conv4_block2_out... conv4_block3_3_b...
conv4_block3_out (Activation)	(None, 7, 7, 1024)	0	conv4_block3_add...
conv4_block4_1_conv (Conv2D)	(None, 7, 7, 256)	262,400	conv4_block3_out...
conv4_block4_1_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block4_1_c...
conv4_block4_1_relu (Activation)	(None, 7, 7, 256)	0	conv4_block4_1_b...
conv4_block4_2_conv (Conv2D)	(None, 7, 7, 256)	590,080	conv4_block4_1_r...
conv4_block4_2_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block4_2_c...
conv4_block4_2_relu (Activation)	(None, 7, 7, 256)	0	conv4_block4_2_b...
conv4_block4_3_conv (Conv2D)	(None, 7, 7, 1024)	263,168	conv4_block4_2_r...
conv4_block4_3_bn (BatchNormalizatio...	(None, 7, 7, 1024)	4,096	conv4_block4_3_c...
conv4_block4_add (Add)	(None, 7, 7, 1024)	0	conv4_block3_out... conv4_block4_3_b...
conv4_block4_out (Activation)	(None, 7, 7, 1024)	0	conv4_block4_add...
conv4_block5_1_conv (Conv2D)	(None, 7, 7, 256)	262,400	conv4_block4_out...
conv4_block5_1_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block5_1_c...
conv4_block5_1_relu (Activation)	(None, 7, 7, 256)	0	conv4_block5_1_b...

conv4_block5_2_conv (Conv2D)	(None, 7, 7, 256)	590,080	conv4_block5_1_r...
conv4_block5_2_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block5_2_c...
conv4_block5_2_relu (Activation)	(None, 7, 7, 256)	0	conv4_block5_2_b...
conv4_block5_3_conv (Conv2D)	(None, 7, 7, 1024)	263,168	conv4_block5_2_r...
conv4_block5_3_bn (BatchNormalizatio...	(None, 7, 7, 1024)	4,096	conv4_block5_3_c...
conv4_block5_add (Add)	(None, 7, 7, 1024)	0	conv4_block4_out... conv4_block5_3_b...
conv4_block5_out (Activation)	(None, 7, 7, 1024)	0	conv4_block5_add...
conv4_block6_1_conv (Conv2D)	(None, 7, 7, 256)	262,400	conv4_block5_out...
conv4_block6_1_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block6_1_c...
conv4_block6_1_relu (Activation)	(None, 7, 7, 256)	0	conv4_block6_1_b...
conv4_block6_2_conv (Conv2D)	(None, 7, 7, 256)	590,080	conv4_block6_1_r...
conv4_block6_2_bn (BatchNormalizatio...	(None, 7, 7, 256)	1,024	conv4_block6_2_c...
conv4_block6_2_relu (Activation)	(None, 7, 7, 256)	0	conv4_block6_2_b...
conv4_block6_3_conv (Conv2D)	(None, 7, 7, 1024)	263,168	conv4_block6_2_r...
conv4_block6_3_bn (BatchNormalizatio...	(None, 7, 7, 1024)	4,096	conv4_block6_3_c...
conv4_block6_add (Add)	(None, 7, 7, 1024)	0	conv4_block5_out... conv4_block6_3_b...
conv4_block6_out (Activation)	(None, 7, 7, 1024)	0	conv4_block6_add...
conv5_block1_1_conv (Conv2D)	(None, 4, 4, 512)	524,800	conv4_block6_out...
conv5_block1_1_bn (BatchNormalizatio...	(None, 4, 4, 512)	2,048	conv5_block1_1_c...
conv5_block1_1_relu (Activation)	(None, 4, 4, 512)	0	conv5_block1_1_b...
conv5_block1_2_conv (Conv2D)	(None, 4, 4, 512)	2,359,808	conv5_block1_1_r...
conv5_block1_2_bn (BatchNormalizatio...	(None, 4, 4, 512)	2,048	conv5_block1_2_c...

conv5_block1_2_relu (Activation)	(None, 4, 4, 512)	0	conv5_block1_2_b...
conv5_block1_0_conv (Conv2D)	(None, 4, 4, 2048)	2,099,200	conv4_block6_out...
conv5_block1_3_conv (Conv2D)	(None, 4, 4, 2048)	1,050,624	conv5_block1_2_r...
conv5_block1_0_bn (BatchNormalizatio...	(None, 4, 4, 2048)	8,192	conv5_block1_0_c...
conv5_block1_3_bn (BatchNormalizatio...	(None, 4, 4, 2048)	8,192	conv5_block1_3_c...
conv5_block1_add (Add)	(None, 4, 4, 2048)	0	conv5_block1_0_b... conv5_block1_3_b...
conv5_block1_out (Activation)	(None, 4, 4, 2048)	0	conv5_block1_add...
conv5_block2_1_conv (Conv2D)	(None, 4, 4, 512)	1,049,088	conv5_block1_out...
conv5_block2_1_bn (BatchNormalizatio...	(None, 4, 4, 512)	2,048	conv5_block2_1_c...
conv5_block2_1_relu (Activation)	(None, 4, 4, 512)	0	conv5_block2_1_b...
conv5_block2_2_conv (Conv2D)	(None, 4, 4, 512)	2,359,808	conv5_block2_1_r...
conv5_block2_2_bn (BatchNormalizatio...	(None, 4, 4, 512)	2,048	conv5_block2_2_c...
conv5_block2_2_relu (Activation)	(None, 4, 4, 512)	0	conv5_block2_2_b...
conv5_block2_3_conv (Conv2D)	(None, 4, 4, 2048)	1,050,624	conv5_block2_2_r...
conv5_block2_3_bn (BatchNormalizatio...	(None, 4, 4, 2048)	8,192	conv5_block2_3_c...
conv5_block2_add (Add)	(None, 4, 4, 2048)	0	conv5_block1_out... conv5_block2_3_b...
conv5_block2_out (Activation)	(None, 4, 4, 2048)	0	conv5_block2_add...
conv5_block3_1_conv (Conv2D)	(None, 4, 4, 512)	1,049,088	conv5_block2_out...
conv5_block3_1_bn (BatchNormalizatio...	(None, 4, 4, 512)	2,048	conv5_block3_1_c...
conv5_block3_1_relu (Activation)	(None, 4, 4, 512)	0	conv5_block3_1_b...
conv5_block3_2_conv (Conv2D)	(None, 4, 4, 512)	2,359,808	conv5_block3_1_r...
conv5_block3_2_bn (BatchNormalizatio...	(None, 4, 4, 512)	2,048	conv5_block3_2_c...

conv5_block3_2_relu (Activation)	(None, 4, 4, 512)	0	conv5_block3_2_b...
conv5_block3_3_conv (Conv2D)	(None, 4, 4, 2048)	1,050,624	conv5_block3_2_r...
conv5_block3_3_bn (BatchNormalizatio...	(None, 4, 4, 2048)	8,192	conv5_block3_3_c...
conv5_block3_add (Add)	(None, 4, 4, 2048)	0	conv5_block2_out... conv5_block3_3_b...
conv5_block3_out (Activation)	(None, 4, 4, 2048)	0	conv5_block3_add...

Total params: 23,587,712 (89.98 MB)

Trainable params: 0 (0.00 B)

Non-trainable params: 23,587,712 (89.98 MB)

Step 9 | Train Model

Define EarlyStopping callback

```
early_stopping = callbacks.EarlyStopping(patience=3)
```

Train the model with early stopping

```
history = model.fit(
    train_ds,
    epochs=10,
    validation_data=val_ds,
    callbacks=[early_stopping] # Add the EarlyStopping callback to the
training process
)
```

Epoch 1/10

1763/1763 ————— 150s 53ms/step - accuracy: 0.4786 - loss: 2.5426 - val_accuracy: 0.9685 - val_loss: 0.2653

Epoch 2/10

1763/1763 ————— 84s 48ms/step - accuracy: 0.9507 - loss: 0.2607 - val_accuracy: 0.9912 - val_loss: 0.1031

Epoch 3/10

1763/1763 ————— 84s 48ms/step - accuracy: 0.9806 - loss: 0.1178 - val_accuracy: 0.9961 - val_loss: 0.0560

Epoch 4/10

1763/1763 ————— 85s 48ms/step - accuracy: 0.9897 - loss: 0.0673 - val_accuracy: 0.9970 - val_loss: 0.0353

Epoch 5/10

1763/1763 ————— 84s 48ms/step - accuracy: 0.9942 - loss: 0.0434 - val_accuracy: 0.9987 - val_loss: 0.0236

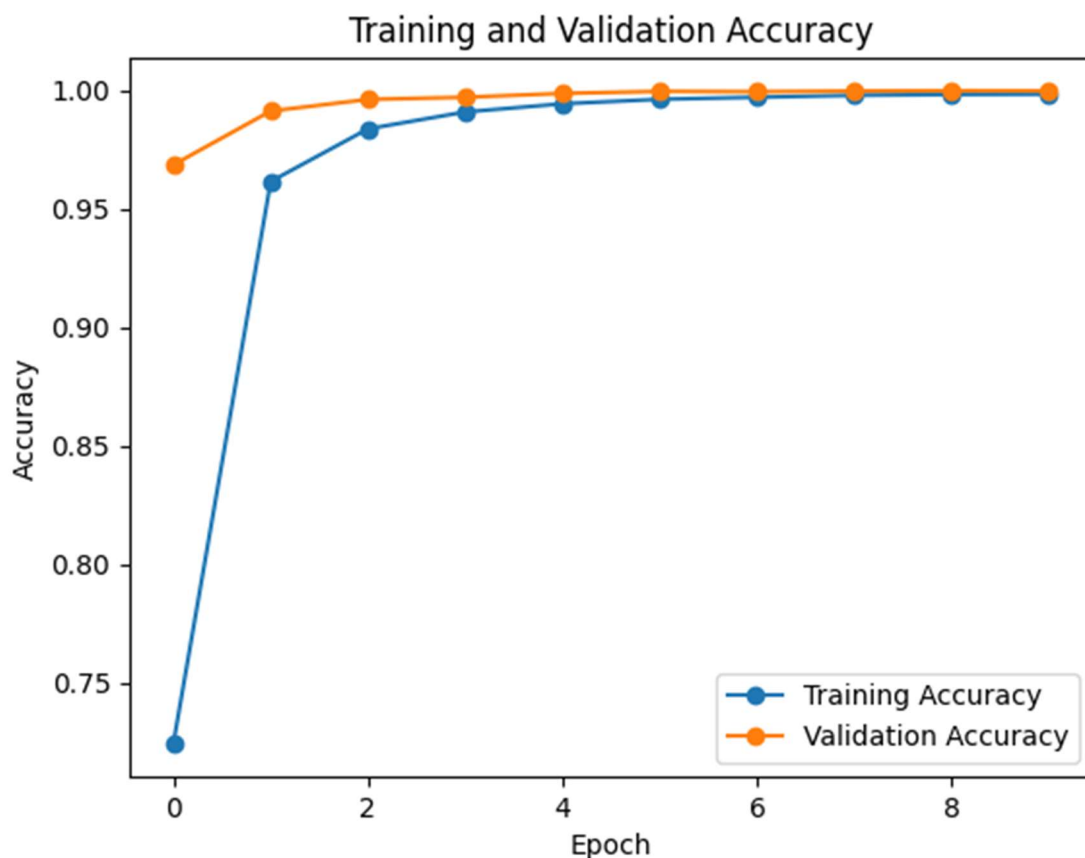
Epoch 6/10

1763/1763 ————— 85s 48ms/step - accuracy: 0.9957 - loss: 0.0301 - val_accuracy: 0.9995 - val_loss: 0.0173

Epoch 7/10
1763/1763 ————— 86s 49ms/step - accuracy: 0.9968 - loss: 0.0225 - val_accuracy: 0.9994 - val_loss: 0.0131
Epoch 8/10
1763/1763 ————— 84s 48ms/step - accuracy: 0.9977 - loss: 0.0172 - val_accuracy: 0.9996 - val_loss: 0.0102
Epoch 9/10
1763/1763 ————— 84s 48ms/step - accuracy: 0.9982 - loss: 0.0147 - val_accuracy: 0.9998 - val_loss: 0.0084
Epoch 10/10
1763/1763 ————— 85s 48ms/step - accuracy: 0.9980 - loss: 0.0123 - val_accuracy: 0.9997 - val_loss: 0.0067

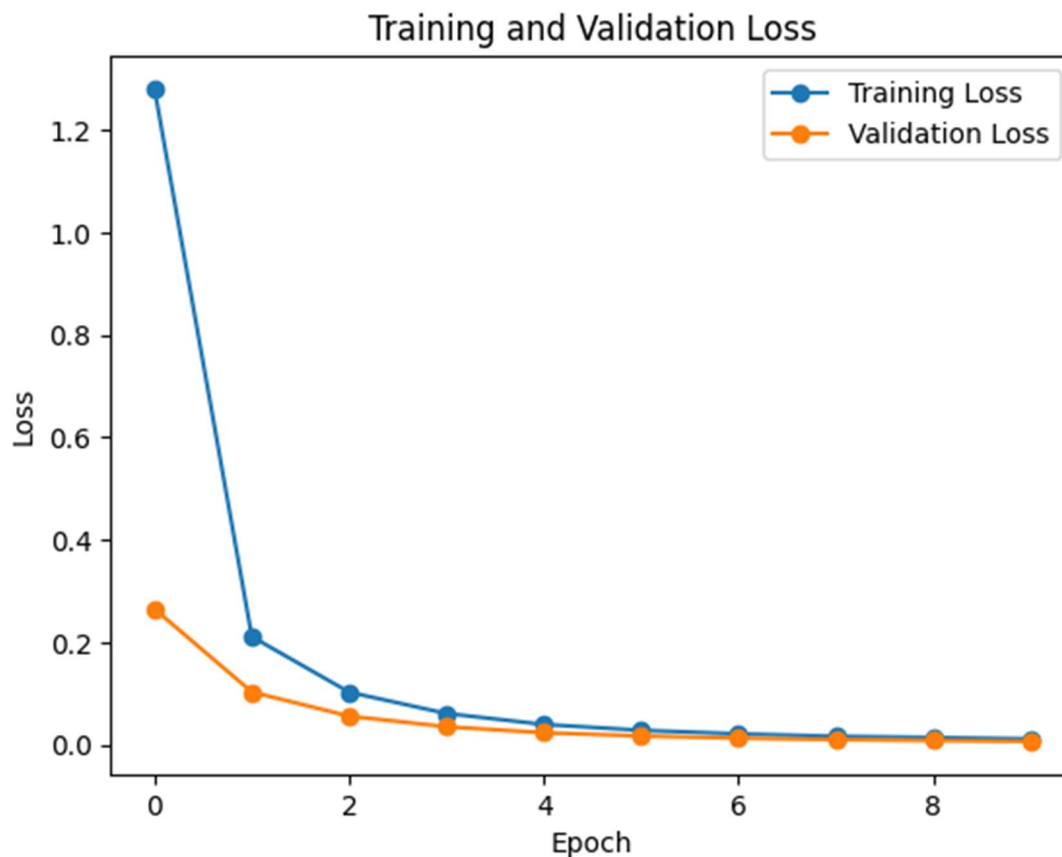
Step 10 | Accuracy Plot

```
# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy',
marker='o')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy',marker='o')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```



Step 11 | Loss Plot

```
# Plot training history
plt.plot(history.history['loss'], label='Training Loss',marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss',marker='o')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



Step 12 | Predict Images

```
def predict(model, img):
    img_array = tf.keras.utils.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100*(np.max(predictions[0])), 0)
    return predicted_class, confidence
```

```

import matplotlib.pyplot as plt

# Assuming val_ds is your validation dataset
plt.figure(figsize=(15, 15))
for images, labels in val_ds.take(1):
    for i in range(25):
        ax = plt.subplot(5, 5, i+1) # Adjust the subplot layout as per
your preference
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class}, \n Predicted:
{predicted_class}.\n Confidence: {confidence}%")

        plt.axis('off')

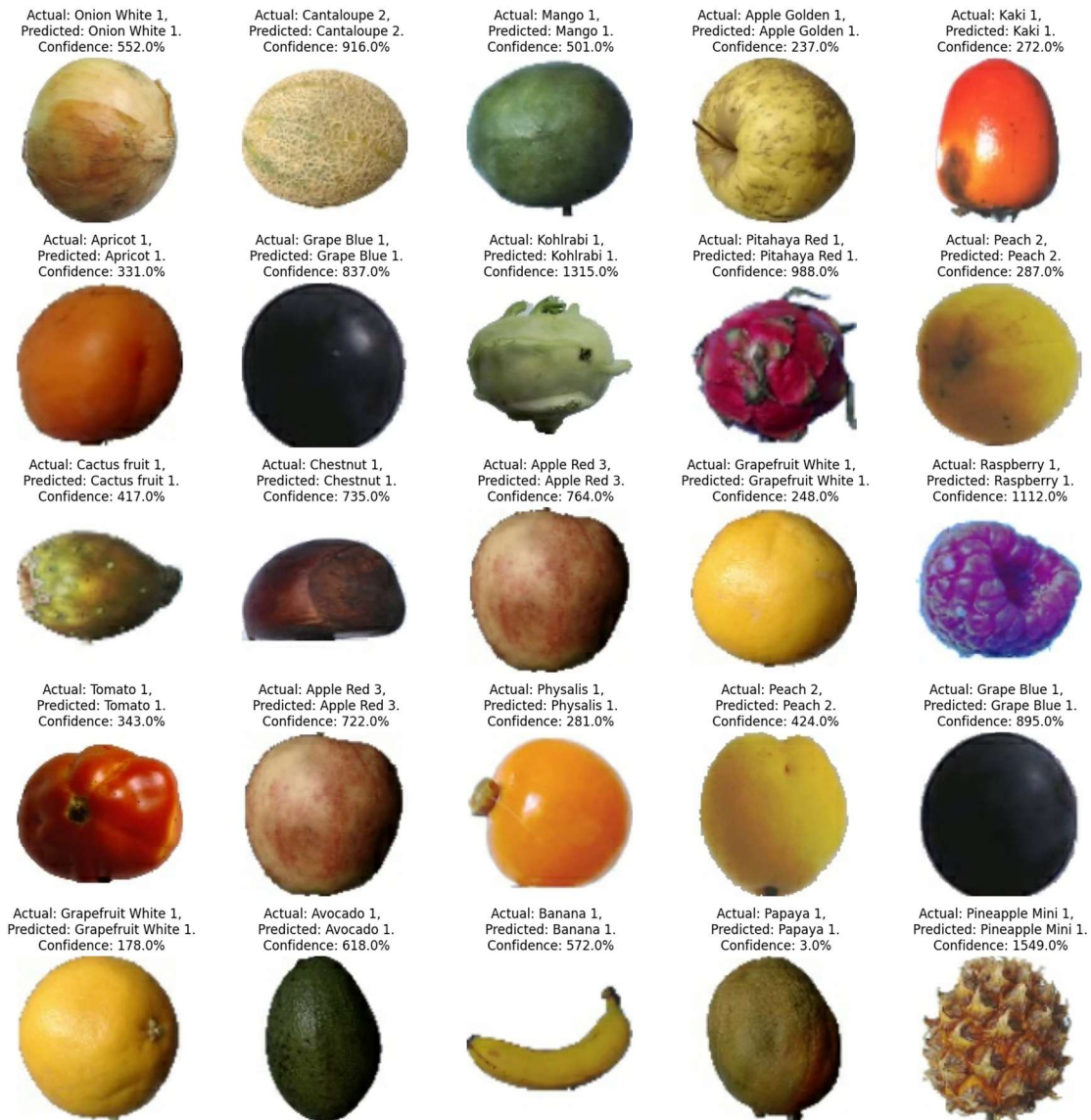
plt.tight_layout()
plt.show()

```

```

1/1 ————— 2s 2s/step
1/1 ————— 0s 24ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 26ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 24ms/step
1/1 ————— 0s 24ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 22ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 22ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 23ms/step

```



Conclusion & Future Work 🎉

Conclusion

- **Key Takeaways:**
 - The project effectively demonstrates how transfer learning with ResNet can be used for fruit classification.
 - Detailed data preprocessing, augmentation, and model fine-tuning techniques were critical in achieving high accuracy.
- **Impact:**
 - Highlights the practical applications of deep learning in image classification tasks.
 - Provides a framework that can be adapted for similar classification problems in various domains.

Future Work

- **Model Improvements:**
 - Experiment with deeper or alternative architectures (e.g., DenseNet, EfficientNet).
 - Explore more sophisticated data augmentation strategies.
- **Deployment:**
 - Develop a user-friendly interface for real-time fruit classification.
 - Implement further optimizations to reduce model latency.
- **Research:**
 - Investigate the effects of additional features or multimodal data (e.g., texture, color histograms) on classification performance.