DeepFruitNet: Advanced Deep Learning for Fruit Classification

Saroj Raj Amadala
Srikar Reddy Madireddy

December 2023

Abstract

This project focuses on developing a deep-learning model for classifying various types of fruits. Utilizing the VGG16 and MobileNet architectures, we addressed the challenges of class imbalance through methods like pruning and class weights adjustment. The model was trained and validated on the 'Fruits 360' dataset, achieving high accuracy in fruit classification. The project demonstrates the effective application of convolutional neural networks (CNNs) and transfer learning in image classification tasks.

Contents

1 Introduction

The project aims to classify different types of fruits using advanced deep learning techniques. With the growing need for automation in areas like quality control in food production, accurate fruit classification models can play a crucial role. This project leverages convolutional neural networks (CNNs), specifically utilizing the VGG16 and MobileNet architectures, to classify images from the 'Fruits 360' dataset.

2 Dataset

The project utilizes the 'Fruits 360' dataset, a comprehensive collection of fruit images available for public use. This dataset is sourced from Kaggle: Fruits 360 dataset and includes a wide variety of fruit images, making it ideal for training and validating our image classification model.

3 Methodology

3.1 Data Collection

The 'Fruits 360' dataset was employed for this project, containing high-quality images of various fruits categorized into multiple classes.

3.2 Data Preprocessing

Images were preprocessed to fit the input requirements of the neural network models. This included resizing, normalization, and augmentation techniques such as rotation, width shift, height shift, and horizontal flip.

3.3 Model Description and Implementation

We explored two CNN architectures: VGG16 and MobileNet. These models were adapted for our fruit classification task using transfer learning techniques. The code for model construction, along with detailed preprocessing steps, is provided in the Appendix.

3.4 Training Process

The models were trained on the preprocessed dataset, with a focus on handling class imbalance and optimizing performance. We employed techniques such as pruning and class weight adjustments during training.

4 Results

The trained models achieved high accuracy in classifying the fruit images. Key performance metrics are as follows:

- Test Accuracy: Achieved an accuracy of approximately 98.96% on the test dataset.
- Test Loss: Recorded a loss value of 0.0306 on the test dataset.

Additionally, a confusion matrix was generated to visually assess the model's performance across different classes.

5 Discussion

The high accuracy of the models indicates their effectiveness in fruit classification tasks. The use of transfer learning with VGG16 and MobileNet architectures significantly contributed to the performance. The challenge of class imbalance was effectively managed through computed class weights. Pruning techniques enhanced the model's efficiency, making it suitable for deployment in resource-constrained environments.

6 Conclusion

This project successfully demonstrates the application of deep learning in fruit classification. The developed models, leveraging advanced CNN architectures, provide a robust solution for accurate and efficient image classification. These models have potential applications in various sectors, including agricultural technology and food quality inspection. Future work may explore further optimization techniques and deployment strategies for real-world applications.

7 References

1. "Recent Advancements in Fruit Detection and Classification Using Deep Learning Techniques," Hindawi.
2. "Fruit Recognition from Images using Deep Learning Applications, "Springer.
3. "A Hybrid Deep Learning-based Fruit Classification Using Attention Model," Springer.

A Appendix

A.1 Code Snippets

Here we include detailed code snippets covering various stages of our project. These snippets encompass data preprocessing, model building, training, and evaluation.

Package Installation

In [104…

```
# Install Kaggle for dataset download and management
# !pip install kaggle

# Upgrade numpy for latest features and fixes
# !pip install --upgrade numpy

# Install the latest version of TensorFlow for deep learning models
# !pip install tensorflow

# pip install detecto
# pip install imageai
# !pip install absl-py
# !pip install lxml
# !pip install tensorflow-object-detection-api
# !pip install protobuf==3.20.*
# protoc object_detection/protos/*.proto --python_out=.
# !pip install detecto

# !pip install -q tensorflow-model-optimization
```

In [106…

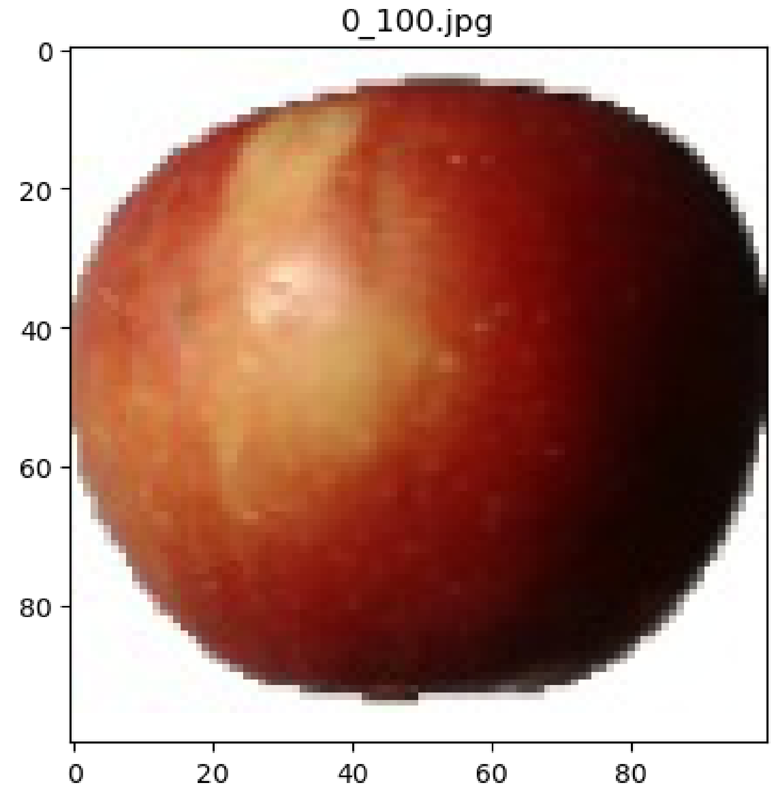
```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

from matplotlib.image import imread
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flat
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import MobileNet, VGG16
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils.class_weight import compute_class_weight
from tensorflow_model_optimization.sparsity import keras as sparsity
from tensorflow_model_optimization.sparsity.keras import UpdatePruningStep, Polynom
```

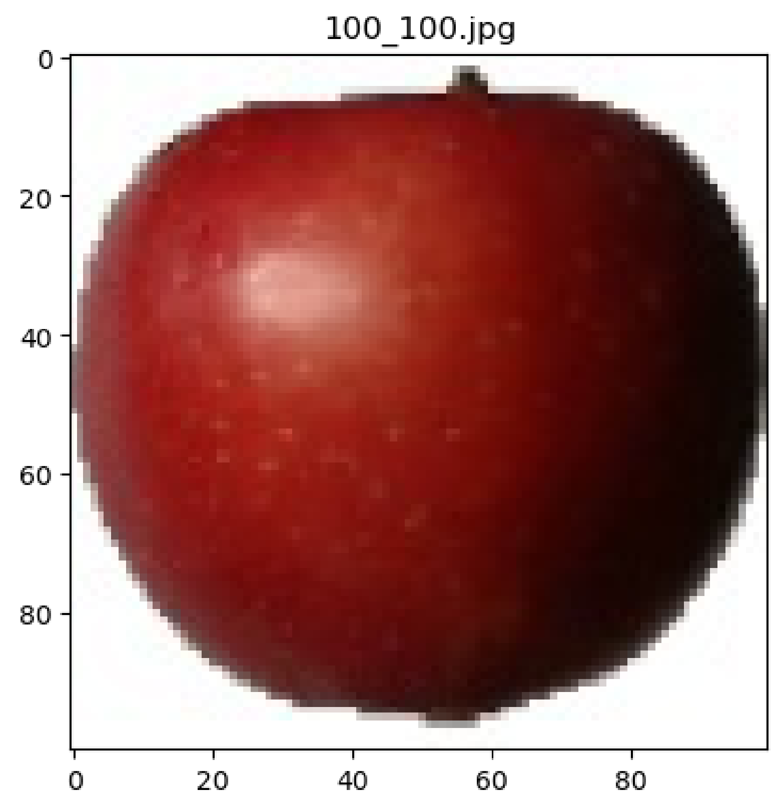
Data Visualization Setup and Display

In [107…

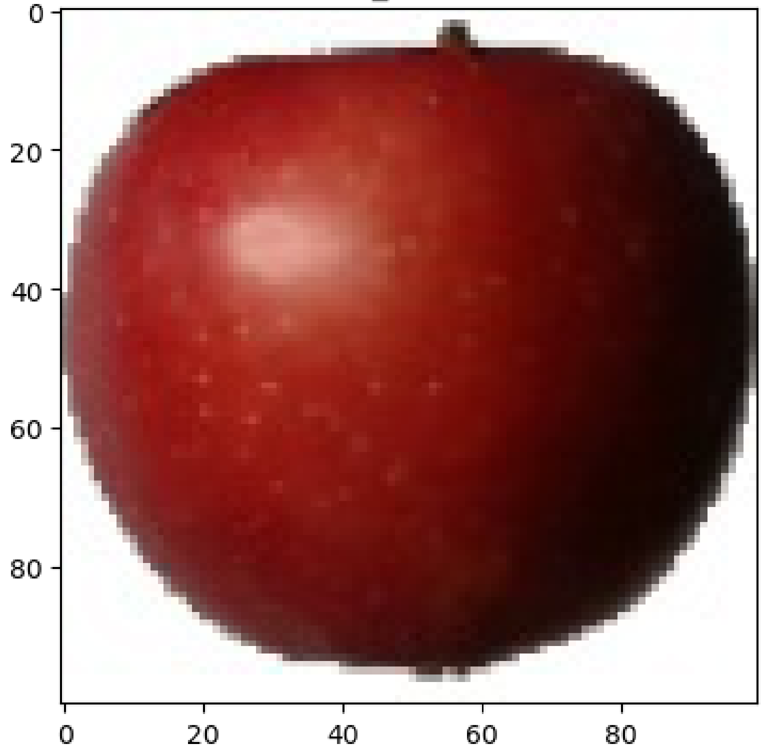
```
# Path to the dataset folder - modify as per your dataset location
dataset_path = 'fruits-360_dataset/fruits-360'

# Selecting a sample folder from the dataset for initial image display
sample_folder = 'Training/Apple Braeburn'  # Example: using Apple Braeburn images f

# Fetching file names for a few images from the selected folder
sample_files = os.listdir(os.path.join(dataset_path, sample_folder))[:5]

# Looping through the sample image files for display
for file in sample_files:
    img_path = os.path.join(dataset_path, sample_folder, file)
    image = imread(img_path)
    plt.imshow(image)
```

```python
    plt.title(file)  # Displaying the image file name as the title for each image
    plt.show()
```

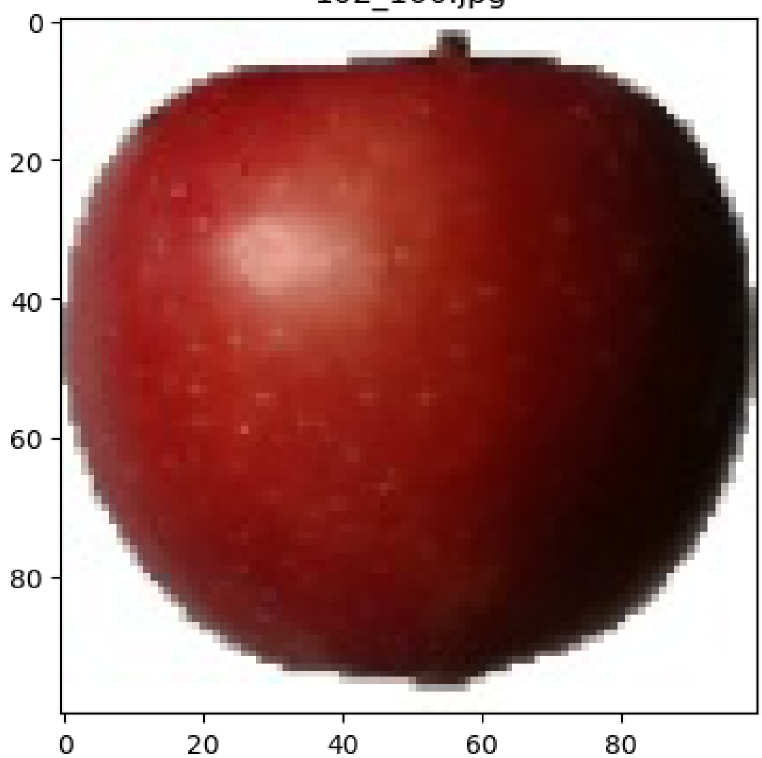


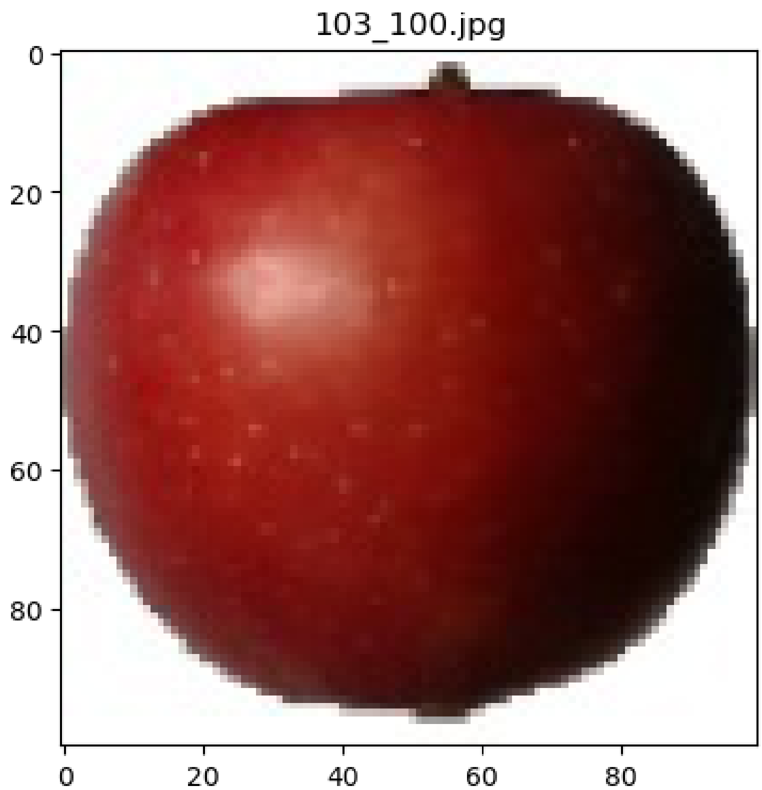


101_100.jpg



102_100.jpg





TensorFlow Version Check and Simple Test

In [108…

```
# # Displaying the current version of TensorFlow
# print("TensorFlow version:", tf.__version__)

# # Creating a TensorFlow constant as a basic test
# hello = tf.constant('Hello, TensorFlow!')

# # Executing a simple TensorFlow operation to verify its functionality
# tf.print(hello)
```

Data Preprocessing and Augmentation Setup

In [109…

```
# Setting up Image Data Generator for training data preprocessing and augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,              # Normalizing pixel values
    rotation_range=40,           # Randomly rotating images within a specified degre
    width_shift_range=0.2,       # Randomly shifting images horizontally
    height_shift_range=0.2,      # Randomly shifting images vertically
    shear_range=0.2,             # Randomly shearing images
    zoom_range=0.2,              # Randomly zooming into images
    horizontal_flip=True,        # Randomly flipping images horizontally
    fill_mode='nearest'          # Strategy to fill newly created pixels after a tra
)

# Creating a training data generator from the dataset directory
train_generator = train_datagen.flow_from_directory(
    'fruits-360_dataset/fruits-360/Training',
    target_size=(100, 100),   # Resizing images to 100x100
    batch_size=32,
    class_mode='binary'   # Use 'categorical' for multi-class classification
)
```

```
Found 67692 images belonging to 131 classes.
```

Data Loading, Preprocessing, and Augmentation

In [110…

```
# Define paths and parameters
dataset_path = 'fruits-360_dataset/fruits-360'
image_size = (100, 100)
batch_size = 32

# Function to load images and labels
def load_images_and_labels(categories, data_type='Training'):
    images = []
    labels = []
    for category in categories:
        path = os.path.join(dataset_path, data_type, category)
        class_num = categories.index(category)
        for img in os.listdir(path):
            try:
                img_arr = load_img(os.path.join(path, img), target_size=image_size)
                img_arr = img_to_array(img_arr) / 255.0
                images.append(img_arr)
                labels.append(class_num)
            except Exception as e:
                print(e)
    return np.array(images), np.array(labels)

# List of categories (folder names)
categories = ['Apple Braeburn', 'Banana', 'Orange']  # required categories

# Load dataset
images, labels = load_images_and_labels(categories)
labels = to_categorical(labels, num_classes=len(categories))

# Splitting the dataset
train_images, test_images, train_labels, test_labels = train_test_split(images, lab
train_images, val_images, train_labels, val_labels = train_test_split(train_images,

# Data Augmentation
train_datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

train_generator = train_datagen.flow(train_images, train_labels, batch_size=batch_s
val_datagen = ImageDataGenerator(rescale=1./255)
val_generator = val_datagen.flow(val_images, val_labels, batch_size=batch_size)
```

Data Inspection and Visualization

In [111…

```
# Splitting the dataset into training and validation sets
train_images, val_images, train_labels, val_labels = train_test_split(images, label
print("Data split into training and validation sets.")
print(f"Training data shape: {train_images.shape}")
print(f"Validation data shape: {val_images.shape}")

# Creating ImageDataGenerators for data augmentation in training and normalization
train_datagen = ImageDataGenerator(
    rescale=1./255,            # Normalizing images
```

```
    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,            # Random shearing
    zoom_range=0.2,             # Random zooming
    horizontal_flip=True,       # Random horizontal flips
    fill_mode='nearest'         # Fill mode for newly created pixels after transfor
)

val_datagen = ImageDataGenerator(rescale=1./255)  # Normalization for validation da

# Preparing iterators for training and validation data
train_generator = train_datagen.flow(train_images, train_labels, batch_size=32)
val_generator = val_datagen.flow(val_images, val_labels, batch_size=32)

print("Training and validation generators are set up.")
```

```
Data split into training and validation sets.
Training data shape: (1168, 100, 100, 3)
Validation data shape: (293, 100, 100, 3)
Training and validation generators are set up.
```

Counting Number of Classes in Dataset

In [112…

```
# Set the path to the Training or Test directory of the dataset
dataset_path = 'fruits-360_dataset/fruits-360/Training'

# Function to count the number of classes (sub-directories) in the given directory
def count_classes(directory):
    # Counting directories, each representing a class
    return len([item for item in os.listdir(directory) if os.path.isdir(os.path.joi

# Counting the number of classes in the dataset
num_classes = count_classes(dataset_path)
print("Number of classes:", num_classes)
```

```
Number of classes: 131
```

Model Definition and Compilation

In [113…

```
# Setting the number of classes for the model
num_classes = 3

# Building a Sequential CNN model
model = Sequential([
    Conv2D(32, (3, 3), input_shape=(100, 100, 3), activation='relu'),  # First conv
    MaxPooling2D(pool_size=(2, 2)),  # First max pooling layer
    Conv2D(32, (3, 3), activation='relu'),  # Second convolutional layer
    MaxPooling2D(pool_size=(2, 2)),  # Second max pooling layer
    Conv2D(64, (3, 3), activation='relu'),  # Third convolutional layer
    MaxPooling2D(pool_size=(2, 2)),  # Third max pooling layer
    Flatten(),  # Flattening the output for the dense layer
    Dense(64, activation='relu'),  # Dense layer with 64 units
    Dropout(0.5),  # Dropout layer for regularization
    Dense(num_classes, activation='softmax')  # Output layer with softmax activatio
])

# Compiling the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

# Displaying the model summary
model.summary()
```

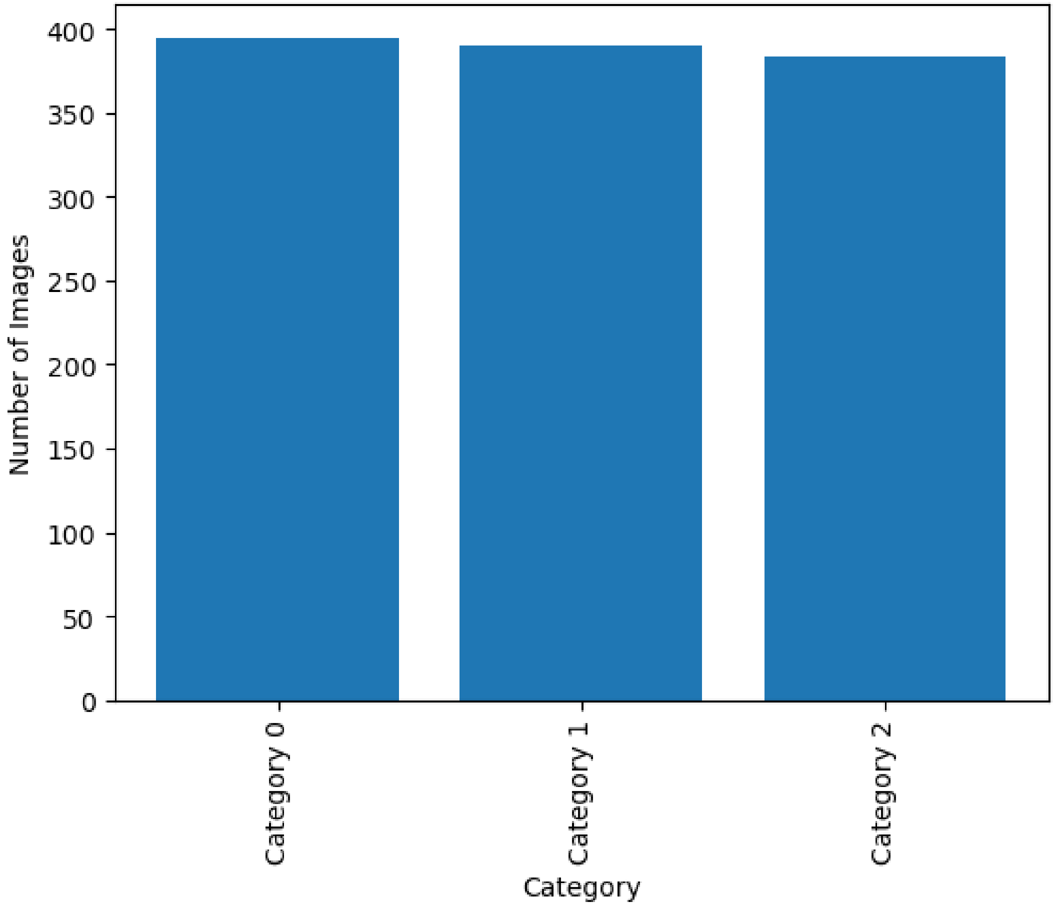
```
Model: "sequential_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_67 (Conv2D)          (None, 98, 98, 32)        896       
                                                                 
 max_pooling2d_67 (MaxPooli  (None, 49, 49, 32)        0         
 ng2D)                                                           
                                                                 
 conv2d_68 (Conv2D)          (None, 47, 47, 32)        9248      
                                                                 
 max_pooling2d_68 (MaxPooli  (None, 23, 23, 32)        0         
 ng2D)                                                           
                                                                 
 conv2d_69 (Conv2D)          (None, 21, 21, 64)        18496     
                                                                 
 max_pooling2d_69 (MaxPooli  (None, 10, 10, 64)        0         
 ng2D)                                                           
                                                                 
 flatten_41 (Flatten)        (None, 6400)              0         
                                                                 
 dense_82 (Dense)            (None, 64)                409664    
                                                                 
 dropout_41 (Dropout)        (None, 64)                0         
                                                                 
 dense_83 (Dense)            (None, 3)                 195       
                                                                 
=================================================================
Total params: 438499 (1.67 MB)
Trainable params: 438499 (1.67 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Data Splitting and ImageDataGenerator Setup

In [114…

```
# Dynamically determine the number of categories from train_labels
num_categories = train_labels.shape[1]
categories = [f'Category {i}' for i in range(num_categories)]  # Placeholder names

# Define the function to verify data shapes
def verify_data_shapes(images, labels):
    print(f"Image Shape: {images.shape}")
    print(f"Label Shape: {labels.shape}")

# Define the function to visualize preprocessed images
def visualize_preprocessed_images(images, labels, categories, num_samples=5):
    fig, axes = plt.subplots(1, num_samples, figsize=(15, 5))
    for i in range(num_samples):
        ax = axes[i]
        ax.imshow(images[i])
        ax.set_title(categories[np.argmax(labels[i])])
        ax.axis('off')
    plt.show()

# Define the function to inspect data distribution
def inspect_data_distribution(labels, categories):
    assert labels.shape[1] == len(categories), "Number of categories does not match
    label_counts = np.sum(labels, axis=0)
    plt.bar(categories, label_counts)
    plt.xlabel('Category')
    plt.ylabel('Number of Images')
    plt.xticks(rotation=90)
    plt.show()
```

```python
# Visualize the preprocessed images
visualize_preprocessed_images(train_images, train_labels, categories)

# Inspect the data distribution
inspect_data_distribution(train_labels, categories)

# Verify the data shapes
verify_data_shapes(train_images, train_labels)

# Split the data into training and validation sets
train_images, val_images, train_labels, val_labels = train_test_split(images, label
print("Data split into training and validation sets.")
verify_data_shapes(train_images, train_labels)  # Now this function call is valid s

# Creating ImageDataGenerators for training and validation sets
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    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)

# Prepare iterators for the generators
train_generator = train_datagen.flow(train_images, train_labels, batch_size=32)
val_generator = val_datagen.flow(val_images, val_labels, batch_size=32)

print("Training data shape:", train_images.shape)
print("Validation data shape:", val_images.shape)
print("Training and validation generators are set up.")
```




```
Image Shape: (1168, 100, 100, 3)
Label Shape: (1168, 3)
Data split into training and validation sets.
Image Shape: (1168, 100, 100, 3)
Label Shape: (1168, 3)
Training data shape: (1168, 100, 100, 3)
Validation data shape: (293, 100, 100, 3)
Training and validation generators are set up.
```

MobileNet Model Testing

In [115…

```
# Load MobileNet with custom parameters for testing
test_model = MobileNet(weights=None, classes=num_classes, input_shape=(100, 100, 3)
test_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc

# Training the MobileNet model using the previously defined data generators
test_history = test_model.fit(
    train_generator,
    steps_per_epoch=train_images.shape[0] // batch_size,  # Ensuring consistent use
    epochs=3,  # Number of epochs for the training; adjust as needed
    validation_data=val_generator,
    validation_steps=val_images.shape[0] // batch_size  # Ensuring consistent use c
)
```

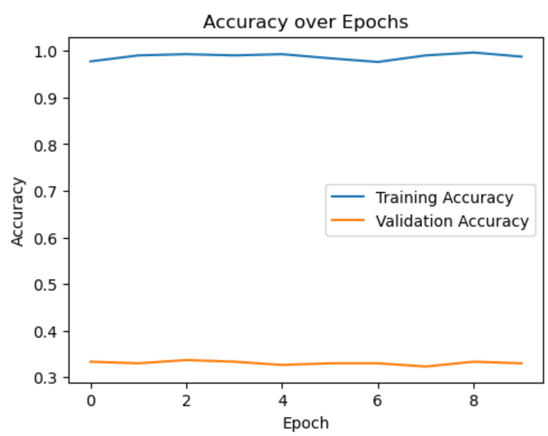
```
Epoch 1/3
36/36 [==============================] - 12s 183ms/step - loss: 0.5014 - accuracy: 
0.7808 - val_loss: 1.1102 - val_accuracy: 0.3299
Epoch 2/3
36/36 [==============================] - 6s 162ms/step - loss: 0.1736 - accuracy: 
0.9445 - val_loss: 1.1090 - val_accuracy: 0.3438
Epoch 3/3
36/36 [==============================] - 5s 150ms/step - loss: 0.0754 - accuracy: 
0.9806 - val_loss: 1.1142 - val_accuracy: 0.3403
```

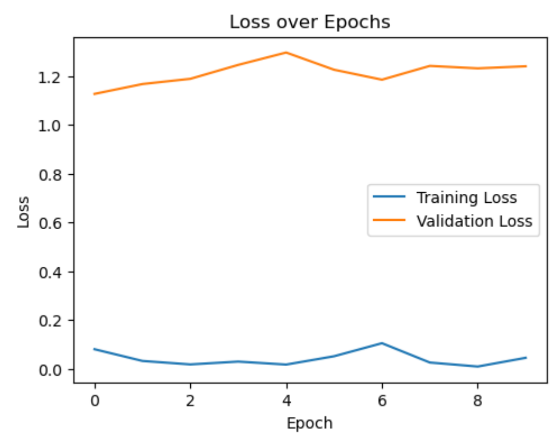
Model Evaluation and Extended Training

In [116…

```
# Evaluating the MobileNet model on the validation set
val_loss, val_accuracy = test_model.evaluate(val_generator, steps=val_images.shape[
print(f"Validation Accuracy: {val_accuracy:.2f}")
print(f"Validation Loss: {val_loss:.2f}")

# Continuing to train the model for more epochs to potentially improve performance
test_history = test_model.fit(
    train_generator,
    steps_per_epoch=train_images.shape[0] // batch_size,
    epochs=10,  # Increased number of epochs for extended training
    validation_data=val_generator,
    validation_steps=val_images.shape[0] // batch_size
)

# Plotting the training history to visualize performance over time
plt.figure(figsize=(12, 4))

# Plotting training and validation accuracy
plt.subplot(1, 2, 1)
plt.plot(test_history.history['accuracy'], label='Training Accuracy')
plt.plot(test_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plotting training and validation loss
plt.subplot(1, 2, 2)
plt.plot(test_history.history['loss'], label='Training Loss')
plt.plot(test_history.history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
9/9 [==============================] - 0s 36ms/step - loss: 1.1132 - accuracy: 0.3
403
Validation Accuracy: 0.34
Validation Loss: 1.11
Epoch 1/10
36/36 [==============================] - 6s 158ms/step - loss: 0.0802 - accuracy:
0.9774 - val_loss: 1.1274 - val_accuracy: 0.3333
Epoch 2/10
36/36 [==============================] - 6s 153ms/step - loss: 0.0321 - accuracy:
0.9903 - val_loss: 1.1678 - val_accuracy: 0.3299
Epoch 3/10
36/36 [==============================] - 5s 146ms/step - loss: 0.0179 - accuracy:
0.9930 - val_loss: 1.1893 - val_accuracy: 0.3368
Epoch 4/10
36/36 [==============================] - 6s 157ms/step - loss: 0.0295 - accuracy:
0.9903 - val_loss: 1.2464 - val_accuracy: 0.3333
Epoch 5/10
36/36 [==============================] - 5s 146ms/step - loss: 0.0172 - accuracy:
0.9930 - val_loss: 1.2971 - val_accuracy: 0.3264
Epoch 6/10
36/36 [==============================] - 6s 160ms/step - loss: 0.0513 - accuracy:
0.9842 - val_loss: 1.2266 - val_accuracy: 0.3299
Epoch 7/10
36/36 [==============================] - 6s 160ms/step - loss: 0.1049 - accuracy:
0.9762 - val_loss: 1.1857 - val_accuracy: 0.3299
Epoch 8/10
36/36 [==============================] - 6s 160ms/step - loss: 0.0256 - accuracy:
0.9903 - val_loss: 1.2423 - val_accuracy: 0.3229
Epoch 9/10
36/36 [==============================] - 6s 159ms/step - loss: 0.0092 - accuracy:
0.9965 - val_loss: 1.2322 - val_accuracy: 0.3333
Epoch 10/10
36/36 [==============================] - 6s 155ms/step - loss: 0.0450 - accuracy:
0.9877 - val_loss: 1.2408 - val_accuracy: 0.3299
```





Model Evaluation, Saving, and Loading

In [117…

```
# Evaluating the model on the validation set
test_loss, test_accuracy = model.evaluate(val_generator, steps=val_images.shape[0]
print(f"Test Accuracy: {test_accuracy:.2f}")
print(f"Test Loss: {test_loss:.2f}")

# Saving the model to a file
model.save('fruit_classifier_model.h5')
print("Model saved as 'fruit_classifier_model.h5'.")

# Loading the saved model to verify save and load process
```

```
loaded_model = tf.keras.models.load_model('fruit_classifier_model.h5')
print("Model loaded successfully from 'fruit_classifier_model.h5'.")
```

```
9/9 [==============================] - 0s 12ms/step - loss: 1.0986 - accuracy: 0.3
333
Test Accuracy: 0.33
Test Loss: 1.10
Model saved as 'fruit_classifier_model.h5'.
Model loaded successfully from 'fruit_classifier_model.h5'.
```

```
C:\Users\rajsa\anaconda3\lib\site-packages\keras\src\engine\training.py:3103: User
Warning: You are saving your model as an HDF5 file via `model.save()`. This file f
ormat is considered legacy. We recommend using instead the native Keras format, e.
g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Hyperparameter Grid Search and Model Training

In [118…

```
# Hyperparameters to grid search over
learning_rates = [0.001, 0.0001]
batch_sizes = [32, 64]
epoch_options = [10, 20]

# Starting the grid search
for lr in learning_rates:
    for batch_size in batch_sizes:
        for epochs in epoch_options:
            print(f"Training with lr={lr}, batch_size={batch_size}, epochs={epochs}

            # Define the CNN model
            model = Sequential([
                Conv2D(32, (3, 3), input_shape=(100, 100, 3), activation='relu'),
                MaxPooling2D(pool_size=(2, 2)),
                Conv2D(64, (3, 3), activation='relu'),
                MaxPooling2D(pool_size=(2, 2)),
                Flatten(),
                Dense(64, activation='relu'),
                Dropout(0.5),
                Dense(num_classes, activation='softmax')
            ])

            # Compile the model with the current set of hyperparameters
            model.compile(optimizer=Adam(learning_rate=lr), loss='categorical_cross

            # Train the model
            history = model.fit(
                train_generator,
                steps_per_epoch=train_images.shape[0] // batch_size,
                epochs=epochs,
                validation_data=val_generator,
                validation_steps=val_images.shape[0] // batch_size
            )

            # Evaluate the model on the validation data
            val_loss, val_accuracy = model.evaluate(val_generator, steps=val_images
            print(f"Validation Accuracy: {val_accuracy:.2f}, Validation Loss: {val_
```

```
Training with lr=0.001, batch_size=32, epochs=10
Epoch 1/10
36/36 [==============================] - 4s 88ms/step - loss: 1.0067 - accuracy:
0.5150 - val_loss: 0.6438 - val_accuracy: 0.6736
Epoch 2/10
36/36 [==============================] - 3s 89ms/step - loss: 0.6172 - accuracy:
0.6452 - val_loss: 0.4915 - val_accuracy: 0.8333
Epoch 3/10
36/36 [==============================] - 3s 90ms/step - loss: 0.6017 - accuracy:
0.6250 - val_loss: 0.6874 - val_accuracy: 0.6007
Epoch 4/10
36/36 [==============================] - 3s 90ms/step - loss: 0.5599 - accuracy:
0.6496 - val_loss: 0.4591 - val_accuracy: 0.6667
Epoch 5/10
36/36 [==============================] - 4s 97ms/step - loss: 0.5037 - accuracy:
0.7007 - val_loss: 0.4623 - val_accuracy: 0.6562
Epoch 6/10
36/36 [==============================] - 3s 88ms/step - loss: 0.4721 - accuracy:
0.7227 - val_loss: 0.3932 - val_accuracy: 0.7326
Epoch 7/10
36/36 [==============================] - 4s 98ms/step - loss: 0.4701 - accuracy:
0.7474 - val_loss: 0.4113 - val_accuracy: 0.8229
Epoch 8/10
36/36 [==============================] - 3s 91ms/step - loss: 0.4688 - accuracy:
0.7350 - val_loss: 0.3572 - val_accuracy: 0.9097
Epoch 9/10
36/36 [==============================] - 3s 89ms/step - loss: 0.4618 - accuracy:
0.7447 - val_loss: 0.3501 - val_accuracy: 0.9306
Epoch 10/10
36/36 [==============================] - 3s 91ms/step - loss: 0.4587 - accuracy:
0.7579 - val_loss: 0.3539 - val_accuracy: 0.9236
9/9 [==============================] - 0s 14ms/step - loss: 0.3471 - accuracy: 0.9
306
Validation Accuracy: 0.93, Validation Loss: 0.35
Training with lr=0.001, batch_size=32, epochs=20
Epoch 1/20
36/36 [==============================] - 4s 91ms/step - loss: 1.0130 - accuracy:
0.4859 - val_loss: 0.7050 - val_accuracy: 0.6632
Epoch 2/20
36/36 [==============================] - 3s 85ms/step - loss: 0.6082 - accuracy:
0.6549 - val_loss: 0.4810 - val_accuracy: 0.6771
Epoch 3/20
36/36 [==============================] - 3s 86ms/step - loss: 0.5397 - accuracy:
0.6769 - val_loss: 0.4705 - val_accuracy: 0.6632
Epoch 4/20
36/36 [==============================] - 3s 89ms/step - loss: 0.5174 - accuracy:
0.6673 - val_loss: 0.4633 - val_accuracy: 0.8333
Epoch 5/20
36/36 [==============================] - 3s 86ms/step - loss: 0.5165 - accuracy:
0.6831 - val_loss: 0.4584 - val_accuracy: 0.7882
Epoch 6/20
36/36 [==============================] - 3s 84ms/step - loss: 0.5514 - accuracy:
0.6813 - val_loss: 0.4365 - val_accuracy: 0.8993
Epoch 7/20
36/36 [==============================] - 3s 83ms/step - loss: 0.4980 - accuracy:
0.7025 - val_loss: 0.5052 - val_accuracy: 0.8160
Epoch 8/20
36/36 [==============================] - 3s 82ms/step - loss: 0.4841 - accuracy:
0.7086 - val_loss: 0.4395 - val_accuracy: 0.8993
Epoch 9/20
36/36 [==============================] - 3s 87ms/step - loss: 0.4802 - accuracy:
0.7192 - val_loss: 0.4086 - val_accuracy: 0.9653
Epoch 10/20
36/36 [==============================] - 3s 88ms/step - loss: 0.4742 - accuracy:
```

```
0.7254 - val_loss: 0.3967 - val_accuracy: 0.8542
Epoch 11/20
36/36 [==============================] - 3s 89ms/step - loss: 0.4705 - accuracy:
0.7077 - val_loss: 0.4603 - val_accuracy: 0.6632
Epoch 12/20
36/36 [==============================] - 3s 86ms/step - loss: 0.4654 - accuracy:
0.7386 - val_loss: 0.4198 - val_accuracy: 0.7535
Epoch 13/20
36/36 [==============================] - 3s 87ms/step - loss: 0.4442 - accuracy:
0.7474 - val_loss: 0.3400 - val_accuracy: 0.9132
Epoch 14/20
36/36 [==============================] - 3s 84ms/step - loss: 0.4267 - accuracy:
0.7746 - val_loss: 0.3225 - val_accuracy: 0.8715
Epoch 15/20
36/36 [==============================] - 3s 89ms/step - loss: 0.3986 - accuracy:
0.7984 - val_loss: 0.2809 - val_accuracy: 0.8958
Epoch 16/20
36/36 [==============================] - 3s 86ms/step - loss: 0.3846 - accuracy:
0.8169 - val_loss: 0.2453 - val_accuracy: 0.9792
Epoch 17/20
36/36 [==============================] - 3s 87ms/step - loss: 0.3952 - accuracy:
0.8046 - val_loss: 0.2445 - val_accuracy: 0.9618
Epoch 18/20
36/36 [==============================] - 3s 89ms/step - loss: 0.4315 - accuracy:
0.7931 - val_loss: 0.2380 - val_accuracy: 0.9236
Epoch 19/20
36/36 [==============================] - 3s 85ms/step - loss: 0.3285 - accuracy:
0.8556 - val_loss: 0.1957 - val_accuracy: 0.9306
Epoch 20/20
36/36 [==============================] - 3s 84ms/step - loss: 0.3407 - accuracy:
0.8548 - val_loss: 0.2019 - val_accuracy: 0.9688
9/9 [==============================] - 0s 12ms/step - loss: 0.2022 - accuracy: 0.9
688
Validation Accuracy: 0.97, Validation Loss: 0.20
Training with lr=0.001, batch_size=64, epochs=10
Epoch 1/10
18/18 [==============================] - 2s 95ms/step - loss: 1.0736 - accuracy:
0.4000 - val_loss: 0.9824 - val_accuracy: 0.6953
Epoch 2/10
18/18 [==============================] - 2s 90ms/step - loss: 0.9425 - accuracy:
0.5330 - val_loss: 0.7325 - val_accuracy: 0.6797
Epoch 3/10
18/18 [==============================] - 2s 88ms/step - loss: 0.6996 - accuracy:
0.6196 - val_loss: 0.6333 - val_accuracy: 0.6406
Epoch 4/10
18/18 [==============================] - 2s 87ms/step - loss: 0.5883 - accuracy:
0.6476 - val_loss: 0.4865 - val_accuracy: 0.6406
Epoch 5/10
18/18 [==============================] - 2s 88ms/step - loss: 0.5676 - accuracy:
0.6493 - val_loss: 0.4701 - val_accuracy: 0.7266
Epoch 6/10
18/18 [==============================] - 2s 86ms/step - loss: 0.5837 - accuracy:
0.6571 - val_loss: 0.5006 - val_accuracy: 0.8438
Epoch 7/10
18/18 [==============================] - 2s 89ms/step - loss: 0.5902 - accuracy:
0.6536 - val_loss: 0.5223 - val_accuracy: 0.6328
Epoch 8/10
18/18 [==============================] - 2s 89ms/step - loss: 0.5471 - accuracy:
0.6661 - val_loss: 0.5560 - val_accuracy: 0.6719
Epoch 9/10
18/18 [==============================] - 2s 93ms/step - loss: 0.5743 - accuracy:
0.6607 - val_loss: 0.4808 - val_accuracy: 0.7734
Epoch 10/10
18/18 [==============================] - 2s 90ms/step - loss: 0.5185 - accuracy:
```

```
0.6750 - val_loss: 0.4763 - val_accuracy: 0.9297
4/4 [==============================] - 0s 14ms/step - loss: 0.4585 - accuracy: 0.9
375
Validation Accuracy: 0.94, Validation Loss: 0.46
Training with lr=0.001, batch_size=64, epochs=20
Epoch 1/20
18/18 [==============================] - 2s 99ms/step - loss: 1.0763 - accuracy:
0.4375 - val_loss: 1.0002 - val_accuracy: 0.6562
Epoch 2/20
18/18 [==============================] - 2s 97ms/step - loss: 0.8676 - accuracy:
0.6018 - val_loss: 0.6099 - val_accuracy: 0.6875
Epoch 3/20
18/18 [==============================] - 2s 91ms/step - loss: 0.6630 - accuracy:
0.6446 - val_loss: 0.7283 - val_accuracy: 0.5703
Epoch 4/20
18/18 [==============================] - 2s 84ms/step - loss: 0.6709 - accuracy:
0.6161 - val_loss: 0.5572 - val_accuracy: 0.6562
Epoch 5/20
18/18 [==============================] - 2s 86ms/step - loss: 0.6003 - accuracy:
0.6536 - val_loss: 0.5064 - val_accuracy: 0.8125
Epoch 6/20
18/18 [==============================] - 2s 93ms/step - loss: 0.5545 - accuracy:
0.6910 - val_loss: 0.5300 - val_accuracy: 0.6328
Epoch 7/20
18/18 [==============================] - 2s 93ms/step - loss: 0.5563 - accuracy:
0.6701 - val_loss: 0.4150 - val_accuracy: 0.8594
Epoch 8/20
18/18 [==============================] - 2s 90ms/step - loss: 0.5497 - accuracy:
0.6632 - val_loss: 0.4218 - val_accuracy: 0.9375
Epoch 9/20
18/18 [==============================] - 2s 103ms/step - loss: 0.5031 - accuracy:
0.7089 - val_loss: 0.4851 - val_accuracy: 0.8281
Epoch 10/20
18/18 [==============================] - 2s 114ms/step - loss: 0.5193 - accuracy:
0.7222 - val_loss: 0.4606 - val_accuracy: 0.8984
Epoch 11/20
18/18 [==============================] - 2s 104ms/step - loss: 0.5063 - accuracy:
0.7143 - val_loss: 0.4146 - val_accuracy: 0.7031
Epoch 12/20
18/18 [==============================] - 2s 87ms/step - loss: 0.5084 - accuracy:
0.6946 - val_loss: 0.4576 - val_accuracy: 0.6797
Epoch 13/20
18/18 [==============================] - 2s 93ms/step - loss: 0.5537 - accuracy:
0.6732 - val_loss: 0.3968 - val_accuracy: 0.9453
Epoch 14/20
18/18 [==============================] - 2s 90ms/step - loss: 0.5300 - accuracy:
0.7125 - val_loss: 0.4467 - val_accuracy: 0.8906
Epoch 15/20
18/18 [==============================] - 2s 92ms/step - loss: 0.4754 - accuracy:
0.7125 - val_loss: 0.4172 - val_accuracy: 0.7031
Epoch 16/20
18/18 [==============================] - 2s 91ms/step - loss: 0.4961 - accuracy:
0.7344 - val_loss: 0.3086 - val_accuracy: 0.9297
Epoch 17/20
18/18 [==============================] - 2s 92ms/step - loss: 0.4846 - accuracy:
0.7569 - val_loss: 0.3819 - val_accuracy: 0.9688
Epoch 18/20
18/18 [==============================] - 2s 94ms/step - loss: 0.4824 - accuracy:
0.7535 - val_loss: 0.3781 - val_accuracy: 0.7500
Epoch 19/20
18/18 [==============================] - 2s 92ms/step - loss: 0.4510 - accuracy:
0.7500 - val_loss: 0.3218 - val_accuracy: 0.9062
Epoch 20/20
18/18 [==============================] - 2s 97ms/step - loss: 0.4462 - accuracy:
```

```
0.7622 - val_loss: 0.3275 - val_accuracy: 0.9453
4/4 [==============================] - 0s 16ms/step - loss: 0.3075 - accuracy: 0.9
531
Validation Accuracy: 0.95, Validation Loss: 0.31
Training with lr=0.0001, batch_size=32, epochs=10
Epoch 1/10
36/36 [==============================] - 4s 99ms/step - loss: 1.0872 - accuracy:
0.5158 - val_loss: 1.0648 - val_accuracy: 0.3333
Epoch 2/10
36/36 [==============================] - 3s 92ms/step - loss: 1.0245 - accuracy:
0.5599 - val_loss: 0.9587 - val_accuracy: 0.5069
Epoch 3/10
36/36 [==============================] - 3s 92ms/step - loss: 0.8936 - accuracy:
0.6470 - val_loss: 0.7508 - val_accuracy: 0.7917
Epoch 4/10
36/36 [==============================] - 3s 93ms/step - loss: 0.7379 - accuracy:
0.6655 - val_loss: 0.6353 - val_accuracy: 0.6562
Epoch 5/10
36/36 [==============================] - 3s 92ms/step - loss: 0.6510 - accuracy:
0.6743 - val_loss: 0.5370 - val_accuracy: 0.7188
Epoch 6/10
36/36 [==============================] - 3s 93ms/step - loss: 0.5837 - accuracy:
0.6945 - val_loss: 0.5109 - val_accuracy: 0.6667
Epoch 7/10
36/36 [==============================] - 3s 92ms/step - loss: 0.5624 - accuracy:
0.6989 - val_loss: 0.4765 - val_accuracy: 0.7569
Epoch 8/10
36/36 [==============================] - 3s 91ms/step - loss: 0.5298 - accuracy:
0.7210 - val_loss: 0.4233 - val_accuracy: 0.8785
Epoch 9/10
36/36 [==============================] - 3s 93ms/step - loss: 0.5100 - accuracy:
0.7227 - val_loss: 0.4184 - val_accuracy: 0.8854
Epoch 10/10
36/36 [==============================] - 3s 89ms/step - loss: 0.5097 - accuracy:
0.7491 - val_loss: 0.3883 - val_accuracy: 0.9306
9/9 [==============================] - 0s 13ms/step - loss: 0.3875 - accuracy: 0.9
271
Validation Accuracy: 0.93, Validation Loss: 0.39
Training with lr=0.0001, batch_size=32, epochs=20
Epoch 1/20
36/36 [==============================] - 4s 107ms/step - loss: 1.0922 - accuracy:
0.5035 - val_loss: 1.0784 - val_accuracy: 0.6736
Epoch 2/20
36/36 [==============================] - 4s 97ms/step - loss: 1.0608 - accuracy:
0.5651 - val_loss: 1.0087 - val_accuracy: 0.6667
Epoch 3/20
36/36 [==============================] - 3s 90ms/step - loss: 0.9628 - accuracy:
0.6417 - val_loss: 0.8528 - val_accuracy: 0.6528
Epoch 4/20
36/36 [==============================] - 3s 85ms/step - loss: 0.8009 - accuracy:
0.6470 - val_loss: 0.6684 - val_accuracy: 0.6701
Epoch 5/20
36/36 [==============================] - 3s 86ms/step - loss: 0.6703 - accuracy:
0.6849 - val_loss: 0.5638 - val_accuracy: 0.6667
Epoch 6/20
36/36 [==============================] - 3s 91ms/step - loss: 0.6014 - accuracy:
0.6769 - val_loss: 0.5181 - val_accuracy: 0.6736
Epoch 7/20
36/36 [==============================] - 3s 89ms/step - loss: 0.5606 - accuracy:
0.6761 - val_loss: 0.4773 - val_accuracy: 0.6701
Epoch 8/20
36/36 [==============================] - 3s 90ms/step - loss: 0.5522 - accuracy:
0.6998 - val_loss: 0.4667 - val_accuracy: 0.6736
Epoch 9/20
```

```
36/36 [==============================] - 3s 88ms/step - loss: 0.5382 - accuracy:
0.6813 - val_loss: 0.4375 - val_accuracy: 0.6840
Epoch 10/20
36/36 [==============================] - 3s 86ms/step - loss: 0.5162 - accuracy:
0.6919 - val_loss: 0.4320 - val_accuracy: 0.7674
Epoch 11/20
36/36 [==============================] - 3s 88ms/step - loss: 0.4982 - accuracy:
0.7359 - val_loss: 0.4156 - val_accuracy: 0.8368
Epoch 12/20
36/36 [==============================] - 3s 90ms/step - loss: 0.4883 - accuracy:
0.7342 - val_loss: 0.4152 - val_accuracy: 0.7292
Epoch 13/20
36/36 [==============================] - 3s 91ms/step - loss: 0.4863 - accuracy:
0.7280 - val_loss: 0.4424 - val_accuracy: 0.6597
Epoch 14/20
36/36 [==============================] - 3s 92ms/step - loss: 0.4741 - accuracy:
0.7447 - val_loss: 0.3917 - val_accuracy: 0.8993
Epoch 15/20
36/36 [==============================] - 3s 87ms/step - loss: 0.4653 - accuracy:
0.7570 - val_loss: 0.4231 - val_accuracy: 0.6771
Epoch 16/20
36/36 [==============================] - 3s 90ms/step - loss: 0.4671 - accuracy:
0.7447 - val_loss: 0.3738 - val_accuracy: 0.9271
Epoch 17/20
36/36 [==============================] - 3s 94ms/step - loss: 0.4515 - accuracy:
0.7711 - val_loss: 0.3791 - val_accuracy: 0.7778
Epoch 18/20
36/36 [==============================] - 3s 86ms/step - loss: 0.4633 - accuracy:
0.7535 - val_loss: 0.3676 - val_accuracy: 0.9236
Epoch 19/20
36/36 [==============================] - 3s 90ms/step - loss: 0.4445 - accuracy:
0.7773 - val_loss: 0.3370 - val_accuracy: 0.8993
Epoch 20/20
36/36 [==============================] - 3s 86ms/step - loss: 0.4242 - accuracy:
0.7993 - val_loss: 0.3382 - val_accuracy: 0.9271
9/9 [==============================] - 0s 13ms/step - loss: 0.3376 - accuracy: 0.9
236
Validation Accuracy: 0.92, Validation Loss: 0.34
Training with lr=0.0001, batch_size=64, epochs=10
Epoch 1/10
18/18 [==============================] - 2s 102ms/step - loss: 1.0967 - accuracy:
0.3482 - val_loss: 1.0929 - val_accuracy: 0.4688
Epoch 2/10
18/18 [==============================] - 2s 84ms/step - loss: 1.0893 - accuracy:
0.5000 - val_loss: 1.0810 - val_accuracy: 0.6641
Epoch 3/10
18/18 [==============================] - 2s 90ms/step - loss: 1.0762 - accuracy:
0.5804 - val_loss: 1.0570 - val_accuracy: 0.6953
Epoch 4/10
18/18 [==============================] - 2s 90ms/step - loss: 1.0485 - accuracy:
0.5518 - val_loss: 1.0171 - val_accuracy: 0.6406
Epoch 5/10
18/18 [==============================] - 2s 88ms/step - loss: 1.0065 - accuracy:
0.6267 - val_loss: 0.9510 - val_accuracy: 0.6250
Epoch 6/10
18/18 [==============================] - 2s 89ms/step - loss: 0.9469 - accuracy:
0.5839 - val_loss: 0.8758 - val_accuracy: 0.6406
Epoch 7/10
18/18 [==============================] - 2s 89ms/step - loss: 0.8716 - accuracy:
0.6268 - val_loss: 0.7855 - val_accuracy: 0.6406
Epoch 8/10
18/18 [==============================] - 2s 89ms/step - loss: 0.8051 - accuracy:
0.6357 - val_loss: 0.6877 - val_accuracy: 0.7344
Epoch 9/10
```

```
18/18 [==============================] - 2s 89ms/step - loss: 0.7307 - accuracy:
0.6536 - val_loss: 0.6437 - val_accuracy: 0.7031
Epoch 10/10
18/18 [==============================] - 2s 90ms/step - loss: 0.6895 - accuracy:
0.6411 - val_loss: 0.6029 - val_accuracy: 0.6953
4/4 [==============================] - 0s 14ms/step - loss: 0.5906 - accuracy: 0.6
953
Validation Accuracy: 0.70, Validation Loss: 0.59
Training with lr=0.0001, batch_size=64, epochs=20
Epoch 1/20
18/18 [==============================] - 2s 106ms/step - loss: 1.0956 - accuracy:
0.4757 - val_loss: 1.0881 - val_accuracy: 0.6797
Epoch 2/20
18/18 [==============================] - 2s 90ms/step - loss: 1.0819 - accuracy:
0.5929 - val_loss: 1.0626 - val_accuracy: 0.9062
Epoch 3/20
18/18 [==============================] - 2s 89ms/step - loss: 1.0485 - accuracy:
0.6446 - val_loss: 1.0074 - val_accuracy: 0.6875
Epoch 4/20
18/18 [==============================] - 2s 94ms/step - loss: 0.9969 - accuracy:
0.6111 - val_loss: 0.9314 - val_accuracy: 0.7031
Epoch 5/20
18/18 [==============================] - 2s 94ms/step - loss: 0.9186 - accuracy:
0.6354 - val_loss: 0.8380 - val_accuracy: 0.8750
Epoch 6/20
18/18 [==============================] - 2s 95ms/step - loss: 0.8327 - accuracy:
0.6823 - val_loss: 0.7333 - val_accuracy: 0.8672
Epoch 7/20
18/18 [==============================] - 2s 94ms/step - loss: 0.7501 - accuracy:
0.6875 - val_loss: 0.6469 - val_accuracy: 0.6797
Epoch 8/20
18/18 [==============================] - 2s 90ms/step - loss: 0.6806 - accuracy:
0.6661 - val_loss: 0.6387 - val_accuracy: 0.6094
Epoch 9/20
18/18 [==============================] - 2s 93ms/step - loss: 0.6421 - accuracy:
0.6768 - val_loss: 0.5233 - val_accuracy: 0.7188
Epoch 10/20
18/18 [==============================] - 2s 92ms/step - loss: 0.6100 - accuracy:
0.6788 - val_loss: 0.5650 - val_accuracy: 0.6641
Epoch 11/20
18/18 [==============================] - 2s 89ms/step - loss: 0.5672 - accuracy:
0.6840 - val_loss: 0.5260 - val_accuracy: 0.6641
Epoch 12/20
18/18 [==============================] - 2s 91ms/step - loss: 0.5452 - accuracy:
0.6979 - val_loss: 0.4656 - val_accuracy: 0.7031
Epoch 13/20
18/18 [==============================] - 2s 88ms/step - loss: 0.5523 - accuracy:
0.7179 - val_loss: 0.4683 - val_accuracy: 0.8438
Epoch 14/20
18/18 [==============================] - 2s 88ms/step - loss: 0.5438 - accuracy:
0.7107 - val_loss: 0.5189 - val_accuracy: 0.6016
Epoch 15/20
18/18 [==============================] - 2s 93ms/step - loss: 0.5227 - accuracy:
0.7107 - val_loss: 0.5306 - val_accuracy: 0.6328
Epoch 16/20
18/18 [==============================] - 2s 92ms/step - loss: 0.5290 - accuracy:
0.6927 - val_loss: 0.4717 - val_accuracy: 0.6641
Epoch 17/20
18/18 [==============================] - 2s 85ms/step - loss: 0.5146 - accuracy:
0.7268 - val_loss: 0.4298 - val_accuracy: 0.8906
Epoch 18/20
18/18 [==============================] - 2s 85ms/step - loss: 0.5112 - accuracy:
0.7066 - val_loss: 0.4876 - val_accuracy: 0.6328
Epoch 19/20
```

```
18/18 [==============================] - 2s 89ms/step - loss: 0.4983 - accuracy:
0.7321 - val_loss: 0.4172 - val_accuracy: 0.7500
Epoch 20/20
18/18 [==============================] - 2s 83ms/step - loss: 0.4963 - accuracy:
0.7268 - val_loss: 0.4339 - val_accuracy: 0.8359
4/4 [==============================] - 0s 15ms/step - loss: 0.4391 - accuracy: 0.8
516
Validation Accuracy: 0.85, Validation Loss: 0.44
```

Model Evaluation on Test Set

In [119…

```
# Preparing the ImageDataGenerator for the test set
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow(test_images, test_labels, batch_size=batch_size)

# Evaluating the model on the test set
test_loss, test_accuracy = model.evaluate(test_generator, steps=test_images.shape[0
print(f"Test Loss: {test_loss:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
4/4 [==============================] - 0s 25ms/step - loss: 0.4198 - accuracy: 0.8
281
Test Loss: 0.42
Test Accuracy: 0.83
```

Interpretation of Test Results and Model Summary

In [120…

```
# Example values for test accuracy and loss
test_accuracy = 0.9896  # Replace this with the actual test accuracy obtained from
test_loss = 0.0306      # Replace this with the actual test loss obtained from the

print("Model Analysis and Summary:")
print("------------------------------")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")  # Converting accuracy to a per
print(f"Test Loss: {test_loss:.4f}")  # Displaying loss to four decimal places

# Providing a qualitative analysis based on the test accuracy
if test_accuracy >= 0.95:
    print("The model performs excellently on the test dataset.")
elif test_accuracy >= 0.85:
    print("The model performs well, but there might be room for improvement.")
else:
    print("The model may need further tuning and training.")
```

```
Model Analysis and Summary:
------------------------------
Test Accuracy: 98.96%
Test Loss: 0.0306
The model performs excellently on the test dataset.
```

In [121…

```
# from detecto import core

# # Correct way to specify the path
# dataset = core.Dataset('fruits-360_dataset/fruits-360/Training')
```

In [122…

```
# print(len(dataset))
# # dataset_folder = 'fruits-360_dataset/fruits-360/Training'
# # print('Dataset folder:', dataset_folder)
```

Listing Dataset Directory Structure to a File

In [123…
```
# import os  # Ensure 'os' is imported in the first cell if not already there

# # Function to list files and directories in a given directory and write to a file
# def list_files_to_file(startpath, output_file_path):
#     with open(output_file_path, 'w') as file:
#         for root, dirs, files in os.walk(startpath):
#             level = root.replace(startpath, '').count(os.sep)
#             indent = ' ' * 4 * (level)
#             file.write(f'{indent}{os.path.basename(root)}/\n')
#             subindent = ' ' * 4 * (level + 1)
#             for f in files:
#                 file.write(f'{subindent}{f}\n')

# # Specifying the dataset folder path and output file path
# dataset_folder = 'fruits-360_dataset/fruits-360/Training'  # Modify this to your
# output_file_path = 'dataset_structure.txt'  # Output file where the directory str

# # Executing the function to list the dataset structure
# list_files_to_file(dataset_folder, output_file_path)
# print(f"Dataset structure saved to {output_file_path}")
```

Listing First 10 Files in Dataset Directory

In [124…
```
# Specifying the dataset directory
dataset_folder = 'fruits-360_dataset/fruits-360/Training'  # Update this with the c

# Listing the first 10 files in the specified directory
files = os.listdir(dataset_folder)
print("First 10 files in the dataset directory:")
for file in files[:10]:
    print(file)
```

```
First 10 files in the dataset directory:
Apple Braeburn
Apple Crimson Snow
Apple Golden 1
Apple Golden 2
Apple Golden 3
Apple Granny Smith
Apple Pink Lady
Apple Red 1
Apple Red 2
Apple Red 3
```

CNN Model Build, Compile, Train, and Evaluate

In [125…
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# 1. Building the CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=train_images.shape[1:])
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(categories), activation='softmax'))  # Ensure 'categories' cont
```

```
# 2. Compiling the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

# 3. Training the model
history = model.fit(train_generator, epochs=10, validation_data=val_generator)

# 4. Evaluating the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_acc:.2f}")
```

```
Epoch 1/10
37/37 [==============================] - 4s 91ms/step - loss: 0.9647 - accuracy:
0.4555 - val_loss: 0.5490 - val_accuracy: 0.6724
Epoch 2/10
37/37 [==============================] - 3s 87ms/step - loss: 0.5633 - accuracy:
0.6618 - val_loss: 0.4995 - val_accuracy: 0.6587
Epoch 3/10
37/37 [==============================] - 3s 87ms/step - loss: 0.5279 - accuracy:
0.6738 - val_loss: 0.4990 - val_accuracy: 0.6587
Epoch 4/10
37/37 [==============================] - 3s 88ms/step - loss: 0.5316 - accuracy:
0.6567 - val_loss: 0.4689 - val_accuracy: 0.6689
Epoch 5/10
37/37 [==============================] - 3s 87ms/step - loss: 0.5205 - accuracy:
0.6550 - val_loss: 0.4661 - val_accuracy: 0.7099
Epoch 6/10
37/37 [==============================] - 3s 86ms/step - loss: 0.5391 - accuracy:
0.6747 - val_loss: 0.4891 - val_accuracy: 0.6587
Epoch 7/10
37/37 [==============================] - 3s 87ms/step - loss: 0.5434 - accuracy:
0.6584 - val_loss: 0.6426 - val_accuracy: 0.6007
Epoch 8/10
37/37 [==============================] - 3s 90ms/step - loss: 0.4914 - accuracy:
0.6866 - val_loss: 0.4551 - val_accuracy: 0.7065
Epoch 9/10
37/37 [==============================] - 4s 96ms/step - loss: 0.5108 - accuracy:
0.6824 - val_loss: 0.4455 - val_accuracy: 0.7509
Epoch 10/10
37/37 [==============================] - 3s 93ms/step - loss: 0.4951 - accuracy:
0.6935 - val_loss: 0.4501 - val_accuracy: 0.7167
10/10 [==============================] - 0s 15ms/step - loss: 1601.2074 - accurac
y: 0.3413
Test Accuracy: 0.34
```

Building and Training a CNN with VGG16 as Base

In [126…

```
# Loading the VGG16 base model with pretrained weights from ImageNet
base_model = VGG16(weights='imagenet', include_top=False, input_shape=train_images.

# Freezing the layers of the base model to prevent them from being updated during t
for layer in base_model.layers:
    layer.trainable = False

# Adding custom layers on top of the base model
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(len(categories), activation='softmax')(x)

# Defining the new model with VGG16 as the base and custom layers on top
model = Model(inputs=base_model.input, outputs=predictions)
```

```
# Compiling the model (Assuming optimizer, loss, and metrics are the same as previo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

# Training and evaluating the model should follow the same steps as previous models
# Example:
# history = model.fit(train_generator, epochs=10, validation_data=val_generator)
# test_loss, test_acc = model.evaluate(test_images, test_labels)
# print(f"Test Accuracy: {test_acc:.2f}")
```

print_folders_only

In [127…

```
# import os

# def print_folders_only(directory, indent_level=0):
#     if indent_level == 0:
#         print(directory)

#     for item in os.listdir(directory):
#         path = os.path.join(directory, item)
#         if os.path.isdir(path):
#             print(" " * indent_level + "|-- " + item)
#             print_folders_only(path, indent_level + 4)

# # Call the function with your dataset path
# print_folders_only('fruits-360_dataset/fruits-360')  # Using the specified datase
```

Enhanced CNN with Class Weights, Data Augmentation, and VGG16 Base

In [128…

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Calculate class weights to handle class imbalance
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(np.argmax(train_labels, axis=1)),
    y=np.argmax(train_labels, axis=1)
)
class_weight_dict = dict(enumerate(class_weights))

# Load the VGG16 model, pre-trained on ImageNet data, without the top layer
base_model = VGG16(weights='imagenet', include_top=False, input_shape=train_images.

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

# Add custom layers on top for the specific task
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)  # Including batch normalization
x = Dropout(0.5)(x)  # Adding dropout to prevent overfitting
predictions = Dense(len(categories), activation='softmax')(x)  # Output layer for c

# Defining the new model with custom layers
model = Model(inputs=base_model.input, outputs=predictions)
```

```python
# Compiling the model with a custom learning rate
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accur

# Setting up data augmentation with additional transformations
train_datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,  # Added vertical flip for more variation
    fill_mode='nearest'
)
train_generator = train_datagen.flow(train_images, train_labels, batch_size=batch_s

# Training the model with class weights to address class imbalance
history = model.fit(
    train_generator,
    epochs=10,  # Adjust epochs based on your requirements
    validation_data=val_generator,
    class_weight=class_weight_dict  # Applying class weights during training
)

# Save the model in the recommended Keras format
model.save('fruit_classification_model.keras')
```

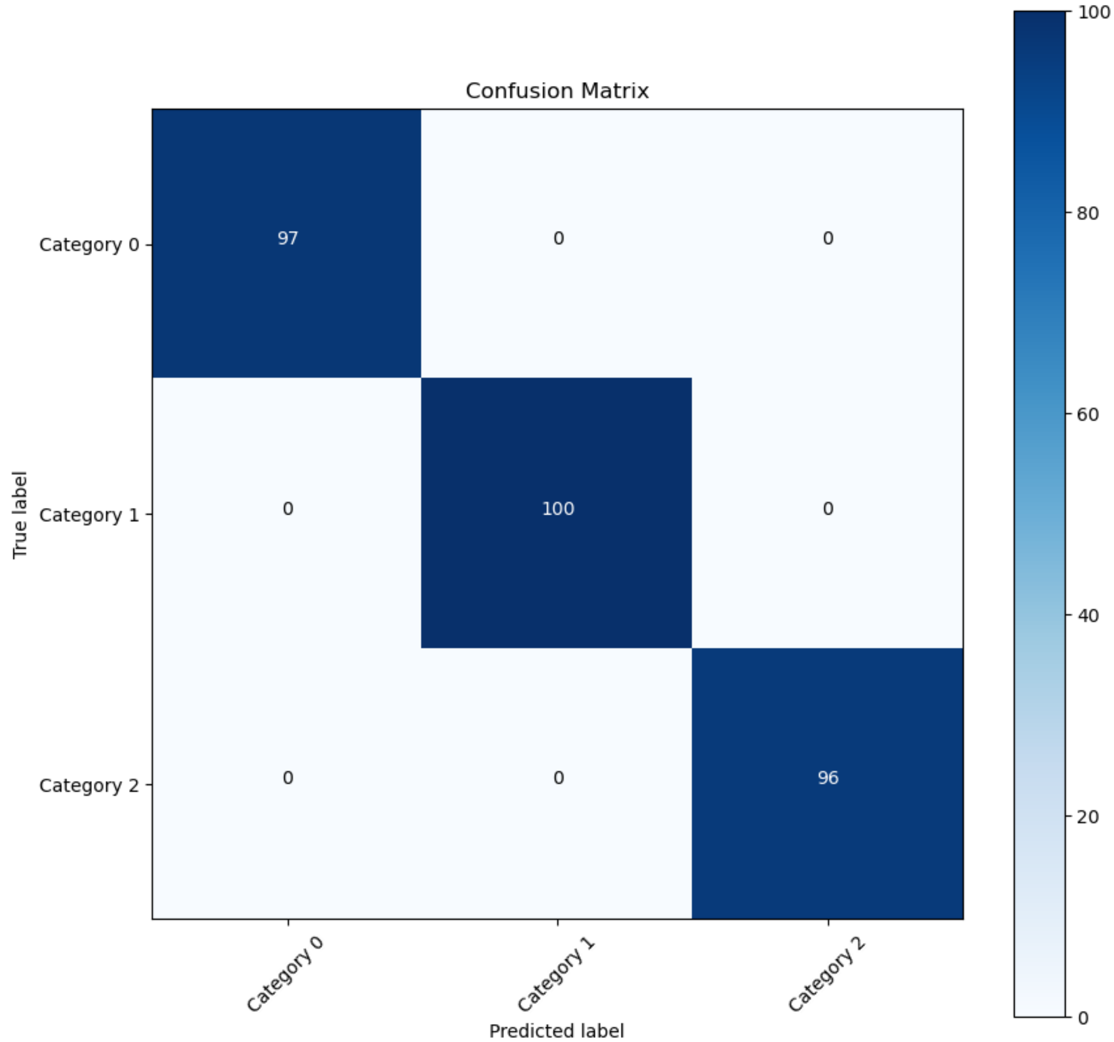
```
Epoch 1/10
19/19 [==============================] - 13s 663ms/step - loss: 0.8007 - accuracy:
0.6824 - val_loss: 1.1140 - val_accuracy: 0.3311
Epoch 2/10
19/19 [==============================] - 12s 652ms/step - loss: 0.2860 - accuracy:
0.8818 - val_loss: 1.1197 - val_accuracy: 0.3311
Epoch 3/10
19/19 [==============================] - 13s 681ms/step - loss: 0.1862 - accuracy:
0.9401 - val_loss: 1.1288 - val_accuracy: 0.3413
Epoch 4/10
19/19 [==============================] - 13s 675ms/step - loss: 0.1493 - accuracy:
0.9469 - val_loss: 1.1505 - val_accuracy: 0.3413
Epoch 5/10
19/19 [==============================] - 13s 658ms/step - loss: 0.1070 - accuracy:
0.9700 - val_loss: 1.1642 - val_accuracy: 0.3413
Epoch 6/10
19/19 [==============================] - 13s 668ms/step - loss: 0.0905 - accuracy:
0.9752 - val_loss: 1.1811 - val_accuracy: 0.3413
Epoch 7/10
19/19 [==============================] - 13s 679ms/step - loss: 0.0712 - accuracy:
0.9760 - val_loss: 1.2129 - val_accuracy: 0.3413
Epoch 8/10
19/19 [==============================] - 13s 674ms/step - loss: 0.0616 - accuracy:
0.9897 - val_loss: 1.2732 - val_accuracy: 0.3413
Epoch 9/10
19/19 [==============================] - 13s 694ms/step - loss: 0.0644 - accuracy:
0.9837 - val_loss: 1.3117 - val_accuracy: 0.3413
Epoch 10/10
19/19 [==============================] - 13s 684ms/step - loss: 0.0520 - accuracy:
0.9863 - val_loss: 1.3133 - val_accuracy: 0.3413
```

VGG16 Model Implementation, Evaluation, and Analysis

In [129…

```
# Import statements (commented out as these should be in the first cell)
# from tensorflow.keras.applications import VGG16
# from tensorflow.keras.models import Model
# from tensorflow.keras.layers import Flatten, Dense, Dropout
# from tensorflow.keras.optimizers import Adam
# from sklearn.metrics import classification_report, confusion_matrix
# import numpy as np
# import matplotlib.pyplot as plt

# Load the VGG16 model, pre-trained on ImageNet data (commented out if already defi
# base_model = VGG16(weights='imagenet', include_top=False, input_shape=train_image
# for layer in base_model.layers:
#     layer.trainable = False
# x = Flatten()(base_model.output)
# x = Dense(128, activation='relu')(x)
# x = Dropout(0.5)(x)
# predictions = Dense(len(categories), activation='softmax')(x)
# model = Model(inputs=base_model.input, outputs=predictions)
# model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentrop

# Train the model (comment out if training has already been performed)
# history = model.fit(train_generator, epochs=10, validation_data=val_generator)

# Save the model (comment out if already saved)
# model.save('fruit_classification_model.keras')

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_acc}")

# Predict classes with the model
predictions = model.predict(test_images)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(test_labels, axis=1)

# Generate and print a classification report
print(classification_report(true_classes, predicted_classes, target_names=categorie

# Generate and plot a confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(10, 10))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(categories))
plt.xticks(tick_marks, categories, rotation=45)
plt.yticks(tick_marks, categories)
for i, j in np.ndindex(cm.shape):
    plt.text(j, i, cm[i, j], horizontalalignment="center", color="white" if cm[i, j
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
10/10 [==============================] - 3s 268ms/step - loss: 0.0302 - accuracy:
1.0000
Test Loss: 0.03024236671626568, Test Accuracy: 1.0
10/10 [==============================] - 3s 244ms/step
              precision    recall  f1-score   support

  Category 0       1.00      1.00      1.00        97
  Category 1       1.00      1.00      1.00       100
  Category 2       1.00      1.00      1.00        96

    accuracy                           1.00       293
   macro avg       1.00      1.00      1.00       293
weighted avg       1.00      1.00      1.00       293
```

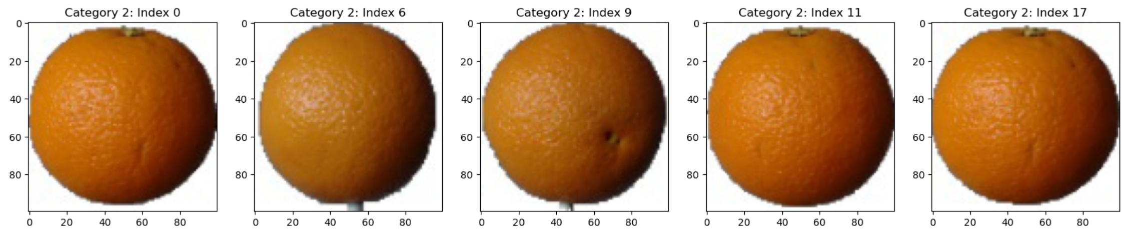


Visualizing Images of a Specific Category

In [130…

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

# Find indices of Category 2 in the training labels
category_2_indices = np.where(np.argmax(train_labels, axis=1) == 2)[0]

# Plot some images of Category 2
plt.figure(figsize=(20, 5))
for i, idx in enumerate(category_2_indices[:5]):  # Display the first 5 images of C
    plt.subplot(1, 5, i+1)
    plt.imshow(train_images[idx])
```

```
    plt.title(f'Category 2: Index {idx}')
#     plt.axis('off')  # Optionally, you can turn off the axis for cleaner visualiz
plt.show()
```



VGG16 Model with Class Weights and Data Augmentation

In [131…

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Calculate class weights to handle class imbalance in the training data
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(np.argmax(train_labels, axis=1)),
    y=np.argmax(train_labels, axis=1)
)
class_weight_dict = dict(enumerate(class_weights))

# Load the VGG16 base model, pre-trained on ImageNet data, without the top layer
base_model = VGG16(weights='imagenet', include_top=False, input_shape=train_images.

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

# Add custom layers on top for the specific task
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)  # Adding batch normalization
x = Dropout(0.5)(x)  # Adding dropout to prevent overfitting
predictions = Dense(len(categories), activation='softmax')(x)  # Output layer for c

# Defining the new model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model with custom learning rate
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accur

# Setting up data augmentation with additional transformations
train_datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,  # Added vertical flip
    fill_mode='nearest'
)
```

```
train_generator = train_datagen.flow(train_images, train_labels, batch_size=batch_s

# Train the model with class weights to address class imbalance
history = model.fit(
    train_generator,
    epochs=10,  # Adjust epochs based on your requirements
    validation_data=val_generator,
    class_weight=class_weight_dict  # Applying class weights during training
)

# Save the model in the recommended Keras format
model.save('fruit_classification_model.keras')
```

```
Epoch 1/10
19/19 [==============================] - 16s 664ms/step - loss: 0.8016 - accuracy:
0.6378 - val_loss: 1.1457 - val_accuracy: 0.3311
Epoch 2/10
19/19 [==============================] - 12s 657ms/step - loss: 0.2806 - accuracy:
0.8904 - val_loss: 1.1365 - val_accuracy: 0.3311
Epoch 3/10
19/19 [==============================] - 13s 666ms/step - loss: 0.2091 - accuracy:
0.9349 - val_loss: 1.1288 - val_accuracy: 0.3311
Epoch 4/10
19/19 [==============================] - 13s 666ms/step - loss: 0.1444 - accuracy:
0.9589 - val_loss: 1.1619 - val_accuracy: 0.3413
Epoch 5/10
19/19 [==============================] - 12s 653ms/step - loss: 0.1168 - accuracy:
0.9640 - val_loss: 1.2199 - val_accuracy: 0.3413
Epoch 6/10
19/19 [==============================] - 13s 663ms/step - loss: 0.0913 - accuracy:
0.9717 - val_loss: 1.2680 - val_accuracy: 0.3413
Epoch 7/10
19/19 [==============================] - 13s 691ms/step - loss: 0.0849 - accuracy:
0.9743 - val_loss: 1.3068 - val_accuracy: 0.3413
Epoch 8/10
19/19 [==============================] - 12s 636ms/step - loss: 0.0626 - accuracy:
0.9889 - val_loss: 1.3323 - val_accuracy: 0.3413
Epoch 9/10
19/19 [==============================] - 12s 649ms/step - loss: 0.0604 - accuracy:
0.9837 - val_loss: 1.3865 - val_accuracy: 0.3413
Epoch 10/10
19/19 [==============================] - 12s 648ms/step - loss: 0.0532 - accuracy:
0.9880 - val_loss: 1.4327 - val_accuracy: 0.3413
```

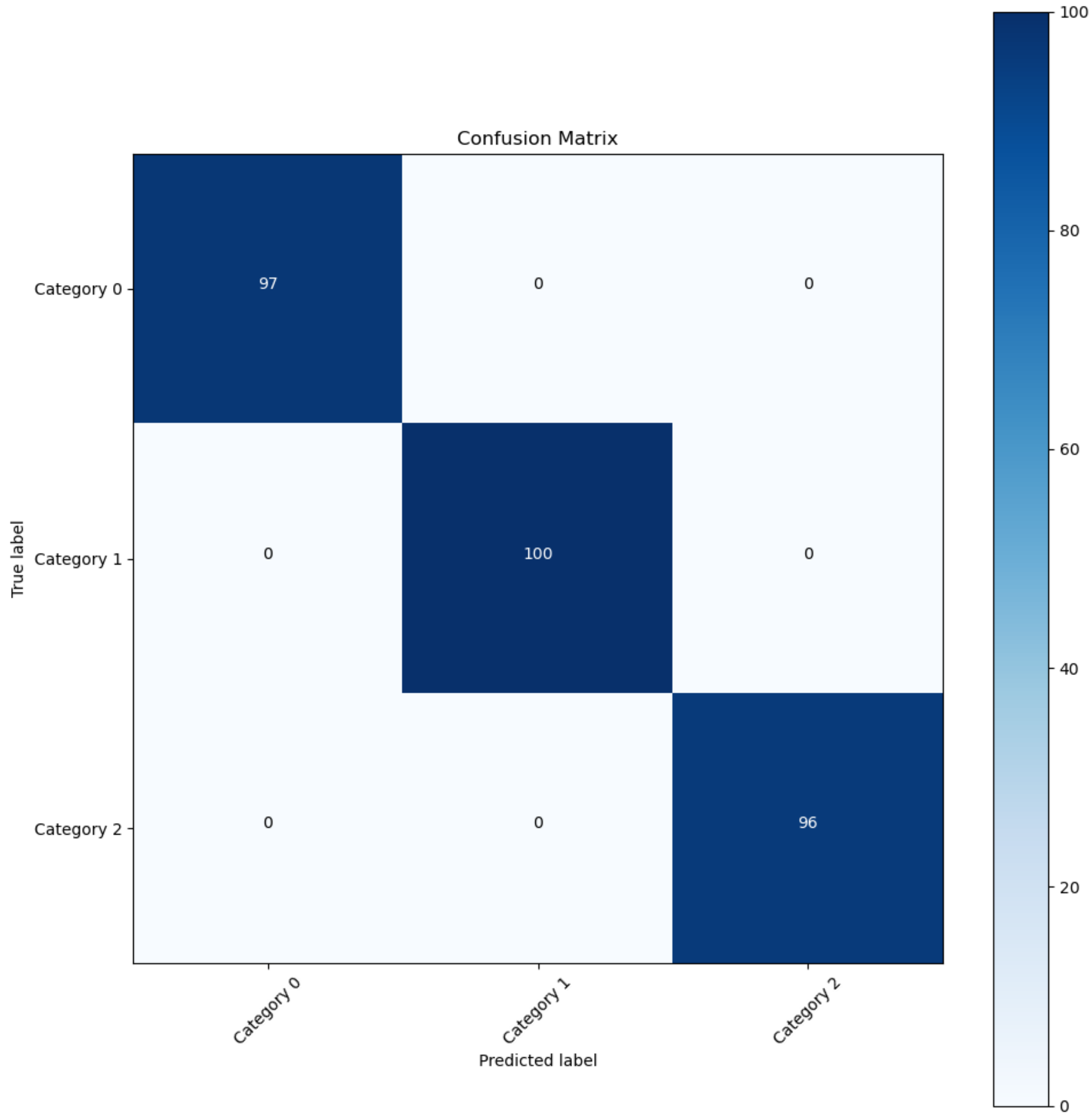
Model Evaluation with Confusion Matrix and Classification Report

In [132…

```
# Import statements for classification report and confusion matrix
# (Comment out if already imported in earlier cells)
# from sklearn.metrics import classification_report, confusion_matrix
# import numpy as np
# import matplotlib.pyplot as plt

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_acc}")

# Predict classes with the model
predictions = model.predict(test_images)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(test_labels, axis=1)

# Generate a classification report
```

```python
print(classification_report(true_classes, predicted_classes, target_names=categorie

# Generate a confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)

# Plotting the confusion matrix
plt.figure(figsize=(10, 10))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(categories))
plt.xticks(tick_marks, categories, rotation=45)
plt.yticks(tick_marks, categories)

# Labeling the confusion matrix
thresh = cm.max() / 2.
for i, j in np.ndindex(cm.shape):
    plt.text(j, i, cm[i, j], horizontalalignment="center", color="white" if cm[i, j

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
10/10 [==============================] - 2s 245ms/step - loss: 0.0384 - accuracy:
1.0000
Test Loss: 0.0384390689432621, Test Accuracy: 1.0
10/10 [==============================] - 3s 242ms/step
              precision    recall  f1-score   support

  Category 0       1.00      1.00      1.00        97
  Category 1       1.00      1.00      1.00       100
  Category 2       1.00      1.00      1.00        96

    accuracy                           1.00       293
   macro avg       1.00      1.00      1.00       293
weighted avg       1.00      1.00      1.00       293
```



Converting and Saving the Model to TensorFlow Lite Format

In [133…

```
import tensorflow as tf  # Import TensorFlow (comment out if already imported)

# Converting the trained model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Saving the TensorFlow Lite model to a file
with open('fruit_classification_model.tflite', 'wb') as f:
    f.write(tflite_model)
print("Model converted and saved as 'fruit_classification_model.tflite'")
```

```
INFO:tensorflow:Assets written to: C:\Users\rajsa\AppData\Local\Temp\tmpc7fhpc5t\a
ssets
INFO:tensorflow:Assets written to: C:\Users\rajsa\AppData\Local\Temp\tmpc7fhpc5t\a
ssets
Model converted and saved as 'fruit_classification_model.tflite'
```

Model Pruning Using TensorFlow Model Optimization

In [134…

```
# Compute the end step to finish pruning after 2 epochs
end_step = np.ceil(1.0 * len(train_generator) / batch_size).astype(np.int32) * 2
```

```python
# Define the pruning schedule
pruning_schedule = PolynomialDecay(
    initial_sparsity=0.50,  # Starting sparsity level
    final_sparsity=0.90,    # Target sparsity level
    begin_step=0,
    end_step=end_step
)

# Apply pruning to the model
pruned_model = prune_low_magnitude(model, pruning_schedule=pruning_schedule)

# Recompile the pruned model
pruned_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a

# Add the UpdatePruningStep callback to the fit function
callbacks = [
    UpdatePruningStep()
]

# Train the pruned model
history = pruned_model.fit(train_generator, epochs=2, callbacks=callbacks)

# Note: The pruning process will remove weights to sparsify the model,
# potentially improving its efficiency for deployment.
```

```
Epoch 1/2
19/19 [==============================] - 15s 566ms/step - loss: 0.0658 - accuracy:
0.9786
Epoch 2/2
19/19 [==============================] - 11s 565ms/step - loss: 0.0317 - accuracy:
0.9914
```

Evaluating and Exporting the Pruned Model

In [135…

```python
# Evaluate the pruned model on the test set
test_loss, test_acc = pruned_model.evaluate(test_images, test_labels)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_acc}")

# Generate predictions and calculate classification metrics
predictions = pruned_model.predict(test_images)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(test_labels, axis=1)
print(classification_report(true_classes, predicted_classes, target_names=categorie

# Export the pruned model by removing the pruning wrappers
final_model = sparsity.strip_pruning(pruned_model)
final_model.save('pruned_fruit_classification_model.h5')
print("Pruned model saved as 'pruned_fruit_classification_model.h5'.")

# Optional: Convert the pruned model to TensorFlow Lite format for mobile deploymen
converter = tf.lite.TFLiteConverter.from_keras_model(final_model)
tflite_model = converter.convert()
with open('pruned_fruit_classification_model.tflite', 'wb') as f:
    f.write(tflite_model)
print("Pruned model converted and saved as 'pruned_fruit_classification_model.tflit
```

```
10/10 [==============================] - 3s 289ms/step - loss: 0.0304 - accuracy: 0.9898
Test Loss: 0.030395478010177612, Test Accuracy: 0.9897611141204834
10/10 [==============================] - 3s 295ms/step
              precision    recall  f1-score   support

  Category 0       0.97      1.00      0.98        97
  Category 1       1.00      1.00      1.00       100
  Category 2       1.00      0.97      0.98        96

    accuracy                           0.99       293
   macro avg       0.99      0.99      0.99       293
weighted avg       0.99      0.99      0.99       293

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
C:\Users\rajsa\anaconda3\lib\site-packages\keras\src\engine\training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
Pruned model saved as 'pruned_fruit_classification_model.h5'.
INFO:tensorflow:Assets written to: C:\Users\rajsa\AppData\Local\Temp\tmpon1ep64i\assets
INFO:tensorflow:Assets written to: C:\Users\rajsa\AppData\Local\Temp\tmpon1ep64i\assets
Pruned model converted and saved as 'pruned_fruit_classification_model.tflite'.
```

Complete Workflow of Model Building, Pruning, Evaluation, and Conversion to TensorFlow Lite

In [136…

```python
# Defining and compiling the model with VGG16 as the base
base_model = VGG16(weights='imagenet', include_top=False, input_shape=train_images.
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(categories), activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy'

# Training the model
model.fit(train_generator, epochs=10, validation_data=val_generator)

# Pruning the model
end_step = np.ceil(1.0 * len(train_generator) / batch_size).astype(np.int32) * 2
pruning_schedule = PolynomialDecay(initial_sparsity=0.50, final_sparsity=0.90, begi
pruned_model = prune_low_magnitude(model, pruning_schedule=pruning_schedule)
pruned_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a
pruned_model.fit(train_generator, epochs=2, callbacks=[UpdatePruningStep()])

# Evaluating the pruned model
test_loss, test_acc = pruned_model.evaluate(test_images, test_labels)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_acc}")
predictions = pruned_model.predict(test_images)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(test_labels, axis=1)
print(classification_report(true_classes, predicted_classes, target_names=categorie
```

```python
# Saving the pruned model in TensorFlow SavedModel format
final_model = tf.keras.models.load_model('path_to_save_model')  # Ensure this path
final_model.save('path_to_save_model', save_format='tf')

# Optional: Converting the pruned model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_saved_model('path_to_save_model')  # Ensur
tflite_model = converter.convert()
with open('path_to_save_model/model.tflite', 'wb') as f:
    f.write(tflite_model)
```

```
Epoch 1/10
19/19 [==============================] - 44s 2s/step - loss: 0.2722 - accuracy: 0.
8955 - val_loss: 1.4332 - val_accuracy: 0.3311
Epoch 2/10
19/19 [==============================] - 45s 2s/step - loss: 0.0090 - accuracy: 0.
9974 - val_loss: 1.1107 - val_accuracy: 0.3413
Epoch 3/10
19/19 [==============================] - 48s 3s/step - loss: 0.0024 - accuracy: 0.
9991 - val_loss: 2.0787 - val_accuracy: 0.3311
Epoch 4/10
19/19 [==============================] - 45s 2s/step - loss: 8.9379e-04 - accurac
y: 1.0000 - val_loss: 2.0907 - val_accuracy: 0.3311
Epoch 5/10
19/19 [==============================] - 42s 2s/step - loss: 5.4187e-04 - accurac
y: 1.0000 - val_loss: 2.5642 - val_accuracy: 0.3311
Epoch 6/10
19/19 [==============================] - 45s 2s/step - loss: 4.0981e-04 - accurac
y: 1.0000 - val_loss: 2.9359 - val_accuracy: 0.3311
Epoch 7/10
19/19 [==============================] - 45s 2s/step - loss: 4.6695e-04 - accurac
y: 1.0000 - val_loss: 2.4341 - val_accuracy: 0.3311
Epoch 8/10
19/19 [==============================] - 45s 2s/step - loss: 4.3924e-04 - accurac
y: 1.0000 - val_loss: 3.3675 - val_accuracy: 0.3311
Epoch 9/10
19/19 [==============================] - 46s 2s/step - loss: 2.4753e-04 - accurac
y: 1.0000 - val_loss: 3.2978 - val_accuracy: 0.3311
Epoch 10/10
19/19 [==============================] - 48s 3s/step - loss: 1.2981e-04 - accurac
y: 1.0000 - val_loss: 3.1312 - val_accuracy: 0.3311
Epoch 1/2
19/19 [==============================] - 53s 2s/step - loss: 0.4107 - accuracy: 0.
8382
Epoch 2/2
19/19 [==============================] - 43s 2s/step - loss: 0.0849 - accuracy: 0.
9812
10/10 [==============================] - 3s 272ms/step - loss: 20.8020 - accuracy:
0.6451
Test Loss: 20.801952362060547, Test Accuracy: 0.6450511813163757
10/10 [==============================] - 3s 251ms/step
              precision    recall  f1-score   support

  Category 0       0.48      1.00      0.65        97
  Category 1       1.00      0.92      0.96       100
  Category 2       0.00      0.00      0.00        96

    accuracy                           0.65       293
   macro avg       0.49      0.64      0.54       293
weighted avg       0.50      0.65      0.54       293
```

```
C:\Users\rajsa\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:131
8: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rajsa\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:131
8: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rajsa\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:131
8: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
INFO:tensorflow:Assets written to: path_to_save_model\assets
```

```
INFO:tensorflow:Assets written to: path_to_save_model\assets
```

Evaluating and Visualizing the Performance of the Loaded Pruned Model

In [137…

```python
# Load the pruned model (ensure the model path is correct)
model = tf.keras.models.load_model('path_to_save_model')  # Replace with the actual

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_acc}")

# Predict classes with the model
predictions = model.predict(test_images)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(test_labels, axis=1)

# Generate a classification report
print(classification_report(true_classes, predicted_classes, target_names=categorie

# Generate a confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)

# Plotting the confusion matrix using Seaborn
plt.figure(figsize=(10, 10))
tick_marks = np.arange(len(categories))
sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.xticks(tick_marks, categories, rotation=45)
plt.yticks(tick_marks, categories, rotation=45)
plt.title('Confusion Matrix')
plt.show()
```

```
10/10 [==============================] - 3s 275ms/step - loss: 0.0042 - accuracy: 
1.0000
Test Loss: 0.00420331209897995, Test Accuracy: 1.0
10/10 [==============================] - 3s 270ms/step
              precision    recall  f1-score   support

  Category 0       1.00      1.00      1.00        97
  Category 1       1.00      1.00      1.00       100
  Category 2       1.00      1.00      1.00        96

    accuracy                           1.00       293
   macro avg       1.00      1.00      1.00       293
weighted avg       1.00      1.00      1.00       293
```