```
# -*- coding: utf-8 -*-
"""
Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

This Google Colab notebook provides a comprehensive deep learning project
to identify rotten fruits and vegetables using transfer learning.

It covers:
1. Setting up the environment.
2. Data loading and preprocessing.
3. Data augmentation for improved model generalization.
4. Building a classification model using pre-trained MobileNetV2 (transfer learning).
5. Training the model.
6. Evaluating the model's performance.
7. Making predictions on new images.
8. Saving the trained model.

Author: Your AI Assistant
Date: July 16, 2025
"""

# --- 1. Environment Setup and Imports ---
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
import pathlib
import shutil

print(f"TensorFlow Version: {tf.__version__}")
print(f"GPU Available: {tf.config.list_physical_devices('GPU')}")

# --- 2. Data Collection and Preparation ---
# For this example, we will simulate a dataset or use a small public one.
# In a real project, you would upload your dataset to Colab or mount Google Drive.

# Option 1: Simulate a small dataset structure for demonstration
# This creates dummy image files and directories.
def create_dummy_dataset(base_dir='data', num_images_per_class=50):
    """Creates a dummy dataset structure for demonstration purposes."""
    classes = ['rotten', 'fresh']
    for cls in classes:
        class_path = os.path.join(base_dir, cls)
        os.makedirs(class_path, exist_ok=True)
        for i in range(num_images_per_class):
            # Create a dummy image file (e.g., a blank text file)
            with open(os.path.join(class_path, f'image_{i:03d}.txt'), 'w') as f:
                f.write(f"Dummy image content for {cls} {i}")
    print(f"Dummy dataset created at: {base_dir}")

# Clean up previous dummy data if it exists
if os.path.exists('data'):
    shutil.rmtree('data')

# Create a small dummy dataset for demonstration
create_dummy_dataset()

# Define dataset directory
data_dir = pathlib.Path('data')

# Parameters for data loading
```

```
IMG_HEIGHT = 224
IMG_WIDTH = 224
BATCH_SIZE = 32

# Simulate data for demonstration if actual loading fails
num_classes = 2 # rotten, fresh
num_train_samples = 100
num_val_samples = 20

# Create dummy data tensors
dummy_train_images = tf.random.normal((num_train_samples, IMG_HEIGHT, IMG_WIDTH, 3))
dummy_train_labels    =    tf.one_hot(tf.random.uniform((num_train_samples,),    minval=0,    maxval=num_classes,
dtype=tf.int32), depth=num_classes)
dummy_val_images = tf.random.normal((num_val_samples, IMG_HEIGHT, IMG_WIDTH, 3))
dummy_val_labels    =    tf.one_hot(tf.random.uniform((num_val_samples,),    minval=0,    maxval=num_classes,
dtype=tf.int32), depth=num_classes)

train_ds = tf.data.Dataset.from_tensor_slices((dummy_train_images, dummy_train_labels)).batch(BATCH_SIZE)
val_ds = tf.data.Dataset.from_tensor_slices((dummy_val_images, dummy_val_labels)).batch(BATCH_SIZE)
class_names = ['fresh', 'rotten']

# --- 3. Data Augmentation ---
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal_and_vertical"),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),
])

preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input

def prepare_data(image, label):
    image = data_augmentation(image)
    image = preprocess_input(image)
    return image, label

augmented_train_ds = train_ds.map(prepare_data, num_parallel_calls=tf.data.AUTOTUNE)
processed_val_ds = val_ds.map(lambda x, y: (preprocess_input(x), y), num_parallel_calls=tf.data.AUTOTUNE)

AUTOTUNE = tf.data.AUTOTUNE
augmented_train_ds = augmented_train_ds.cache().prefetch(buffer_size=AUTOTUNE)
processed_val_ds = processed_val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# --- 4. Model Building (Transfer Learning) ---
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(IMG_HEIGHT, IMG_WIDTH, 3),
    include_top=False,
    weights='imagenet'
)

base_model.trainable = False

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')

inputs = tf.keras.Input(shape=(IMG_HEIGHT, IMG_WIDTH, 3))
x = base_model(inputs, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])

model.summary()

# --- 5. Training the Model ---
initial_epochs = 10
history = model.fit(augmented_train_ds,
                    epochs=initial_epochs,
                    validation_data=processed_val_ds)

# --- Optional: Fine-tuning the entire model ---
base_model.trainable = True
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate / 10),
              metrics=['accuracy'])

fine_tune_epochs = 10
total_epochs = initial_epochs + fine_tune_epochs

history_fine = model.fit(augmented_train_ds,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=processed_val_ds)

# --- 6. Evaluating the Model ---
loss, accuracy = model.evaluate(processed_val_ds)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")

acc = history.history['accuracy'] + history_fine.history['accuracy']
val_acc = history.history['val_accuracy'] + history_fine.history['val_accuracy']
loss = history.history['loss'] + history_fine.history['loss']
val_loss = history.history['val_loss'] + history_fine.history['val_loss']

plt.figure(figsize=(10, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

# --- 7. Making Predictions ---
```

```
for images, labels in processed_val_ds.take(1):
    predictions = model.predict(images)
    predicted_classes = np.argmax(predictions, axis=1)
    true_classes = np.argmax(labels.numpy(), axis=1)

    print(f"True labels (first 5): {true_classes[:5]}")
    print(f"Predicted labels (first 5): {predicted_classes[:5]}")

# --- 8. Saving the Model ---
model_save_path = 'rotten_fruit_classifier_model'
tf.saved_model.save(model, model_save_path)
print(f"Model saved to: {model_save_path}")
```