Image Classification using Custom CNN and EfficientNet

Introduction

This project aims to perform image classification using a Custom Convolutional Neural Network (CNN) and EfficientNet. The objective is to compare the performance of a simple, custom-built CNN with a pre-trained, efficient model like EfficientNet. We will evaluate the models based on accuracy, training time, and complexity.

Dataset

The dataset used for this project is available on Mendeley Data. It comprises various images classified into predefined categories. We will preprocess these images, train our models on them, and then evaluate the models' performance.

Environment Setup

First, ensure that you have the necessary libraries installed. These include TensorFlow, NumPy, Matplotlib, and Sklearn.

```python
!pip install tensorflow numpy matplotlib sklearn
```

Step 1: Download and Prepare the Dataset

1. **Download the Dataset**:
- Visit [Mendeley Data](https://data.mendeley.com/datasets/brfgw46wzb/1) and download the dataset.
- Upload the dataset to your Google Drive.

2. **Mount Google Drive**:
- Mount your Google Drive to access the dataset.

```python
from google.colab import drive
drive.mount('/content/drive')
```

3. **Extract and Preprocess the Dataset**:
- Extract the dataset and set up data generators for training and validation.

```python
import os
import zipfile

# Define paths
data_dir = '/content/drive/My Drive/Lab03 Dataset/CoLeaf DATASET'  # replace with your
dataset path
img_height, img_width = 224, 224
batch_size = 32

# Data augmentation and normalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

Step 2: Implement the Models

Custom CNN Model

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

def create_custom_cnn(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    return model

# Model parameters
input_shape = (img_height, img_width, 3)
num_classes = len(train_generator.class_indices)

custom_cnn = create_custom_cnn(input_shape, num_classes)
custom_cnn.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

EfficientNet Model

```python
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout

efficientnet_base = EfficientNetB0(weights='imagenet', include_top=False,
input_shape=input_shape)
x = efficientnet_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
```

```
predictions = Dense(num_classes, activation='softmax')(x)

model_efficientnet = Model(inputs=efficientnet_base.input, outputs=predictions)
model_efficientnet.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

Step 3: Train the Models

```python
# Training Custom CNN
history_custom_cnn = custom_cnn.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)

# Training EfficientNet
history_efficientnet = model_efficientnet.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

Step 4: Evaluate and Compare the Models

```python
import matplotlib.pyplot as plt

def plot_history(history, title, filename):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(title)
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.savefig(filename)
    plt.close()

# Plotting the training history
plot_history(history_custom_cnn, 'Custom CNN Accuracy', 'custom_cnn_accuracy.png')
plot_history(history_efficientnet, 'EfficientNet Accuracy', 'efficientnet_accuracy.png')
```

```
# Evaluate on validation data
custom_cnn_performance = custom_cnn.evaluate(validation_generator)
efficientnet_performance = model_efficientnet.evaluate(validation_generator)

custom_cnn_accuracy = custom_cnn_performance[1] * 100
efficientnet_accuracy = efficientnet_performance[1] * 100
```

Step 5: Model Complexity and Summary

```python
print("Custom CNN Summary:")
custom_cnn.summary()

print("EfficientNet Summary:")
model_efficientnet.summary()
```

Results and Discussion

Training and Validation Accuracy:
- Custom CNN Accuracy: 45.69%
- EfficientNet Accuracy: 10.15%

Model Complexity:
- **Custom CNN**: A relatively simple architecture with fewer layers and parameters.
- **EfficientNet**: A more complex and deeper architecture with more parameters.