Crop Disease Detection Model

Use case: Crop Disease Prediction using RGB Images. Classify disease presence from standard images of crops using deep learning models.

Model used :EfficientNetB4

---

1. Project Overview

- A CNN model for plant disease detection using transfer learning (EfficientNetB4).
- Two-stage training: initial with frozen base, then fine-tuning selected layers.
- Uses the PlantVillage dataset (39 classes).

---

2. Model code and working

Download and Extract Dataset:

```
!wget -O "dataset.zip" "<https://data.mendeley.com/public-files/datasets/tyw
btsjrjv/files/b4e3a32f-c0bd-4060-81e9-6144231f2520/file_downloaded>"
!unzip dataset.zip -d data
```

Split Data (Python):

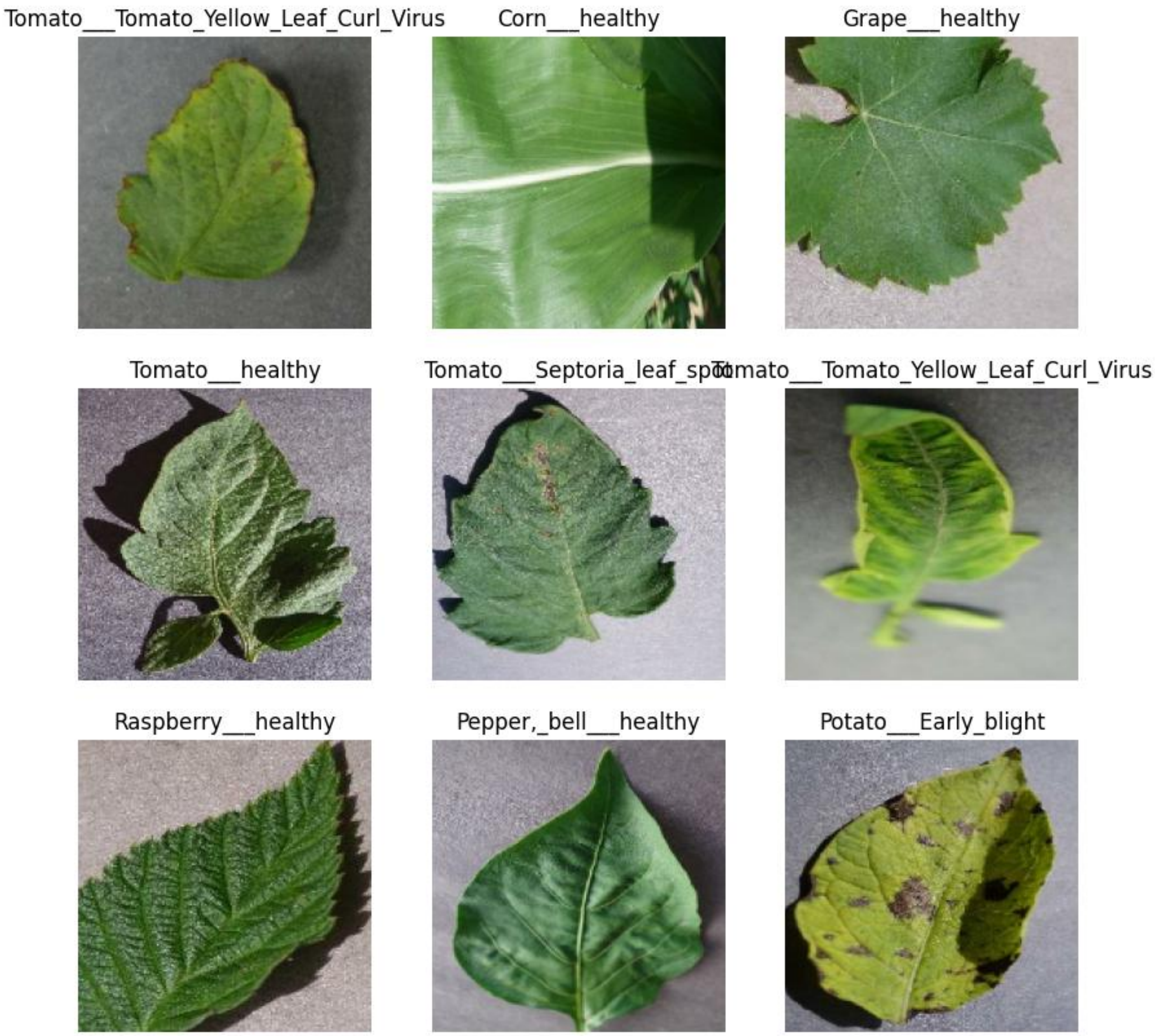
```
import os
import shutil
import random
from tqdm import tqdm

original_dataset_path = "data/plantvillage_dataset"
output_base = "data/split"
train_dir = os.path.join(output_base, "train")
```

```
val_dir = os.path.join(output_base, "val")
test_dir = os.path.join(output_base, "test")

train_split     0.8
val_split     0.1
test_split     0.1

for split_dir in [train_dir, val_dir, test_dir]:
    os.makedirs(split_dir, exist_ok=True)

for class_name in tqdm(os.listdir(original_dataset_path)):
    class_path = os.path.join(original_dataset_path, class_name)
    if not os.path.isdir(class_path):
        continue
    images = [img for img in os.listdir(class_path) if img.lower().endswith(('.jpg',
'.jpeg', '.png'))]
    random.shuffle(images)
    total = len(images)
    train_end, val_end = int(total * train_split), int(total * (train_split + val_split))
    train_imgs, val_imgs, test_imgs = images[:train_end], images[train_end:val_
end], images[val_end:]
    for folder, files in zip([train_dir, val_dir, test_dir], [train_imgs, val_imgs, test_i
mgs]):
        os.makedirs(os.path.join(folder, class_name), exist_ok=True)
        for img in files:
            shutil.copy2(os.path.join(class_path, img), os.path.join(folder, class_na
me, img))
print("Dataset split into train, val, and test sets!")
```

Data Loading and Preprocessing

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

```
train_dir = "/content/dataset/train"
validation_dir = "/content/dataset/val"
test_dir = "/content/dataset/test"

BATCH_SIZE     32
IMG_SIZE     (160, 160)

train_dataset  =  tf.keras.utils.image_dataset_from_directory(
   train_dir, shuffle=True, batch_size=BATCH_SIZE, image_size=IMG_SIZE)
validation_dataset = tf.keras.utils.image_dataset_from_directory(
   validation_dir, shuffle=True, batch_size=BATCH_SIZE, image_size=IMG_SIZ
E)
test_dataset =
   tf.keras.utils.image_dataset_from_directory( test_dir,
   batch_size=BATCH_SIZE, image_size=IMG_SIZE)

# Visualize a few images with labels
class_names = train_dataset.class_names
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
   for i in range(9):
      ax = plt.subplot(3, 3, i + 1)
      plt.imshow(images[i].numpy().astype("uint8")) plt.title(class_names[labels[i]])
      plt.axis("off")

# Prefetch for performance
AUTOTUNE      tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)

preprocess_input  =  tf.keras.applications.efficientnet.preprocess_input
```


Tomato___Tomato_Yellow_Leaf_Curl_Virus
Corn___healthy
Grape___healthy
Tomato___healthy
Tomato___Septoria_leaf_spot
Tomato___Tomato_Yellow_Leaf_Curl_Virus
Raspberry___healthy
Pepper,_bell___healthy
Potato___Early_blight

```
from tensorflow.keras import layers, models

IMG_SHAPE     IMG_SIZE + (3,)
base_model = tf.keras.applications.EfficientNetB4(
    input_shape=IMG_SHAPE, include_top=False, weights='imagenet')

base_model.trainable      False  # Freeze base for transfer learning

inputs = tf.keras.Input(shape=IMG_SHAPE)
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.2)(x)
outputs = layers.Dense(len(class_names), activation='softmax')(x)

model = models.Model(inputs, outputs)
```

Training Strategy

```
model.compile( optimizer=tf.keras.optimizers.Adam()
    ,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')]
)


# Stage 1 training: base frozen
initial_epochs     6
history = model.fit( train_dataset,
    epochs=initial_epochs,
    validation_data=validation_dataset
)



# Stage 2: fine-tune deeper layers
```

```
base_model.trainable      True
fine_tune_at      100
for layer in base_model.layers[:fine_tune_at]:
   layer.trainable      False

model.compile( optimizer=tf.keras.optimizers.
   Adam(),
   loss=tf.keras.losses.SparseCategoricalCrossentropy(),
   metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')]
)

fine_tune_epochs      10
total_epochs = initial_epochs + fine_tune_epochs

history_fine =
   model.fit( train_dataset,
   epochs=total_epochs,
   initial_epoch=history.epoch[-1] + 1,
   validation_data=validation_dataset
)
```

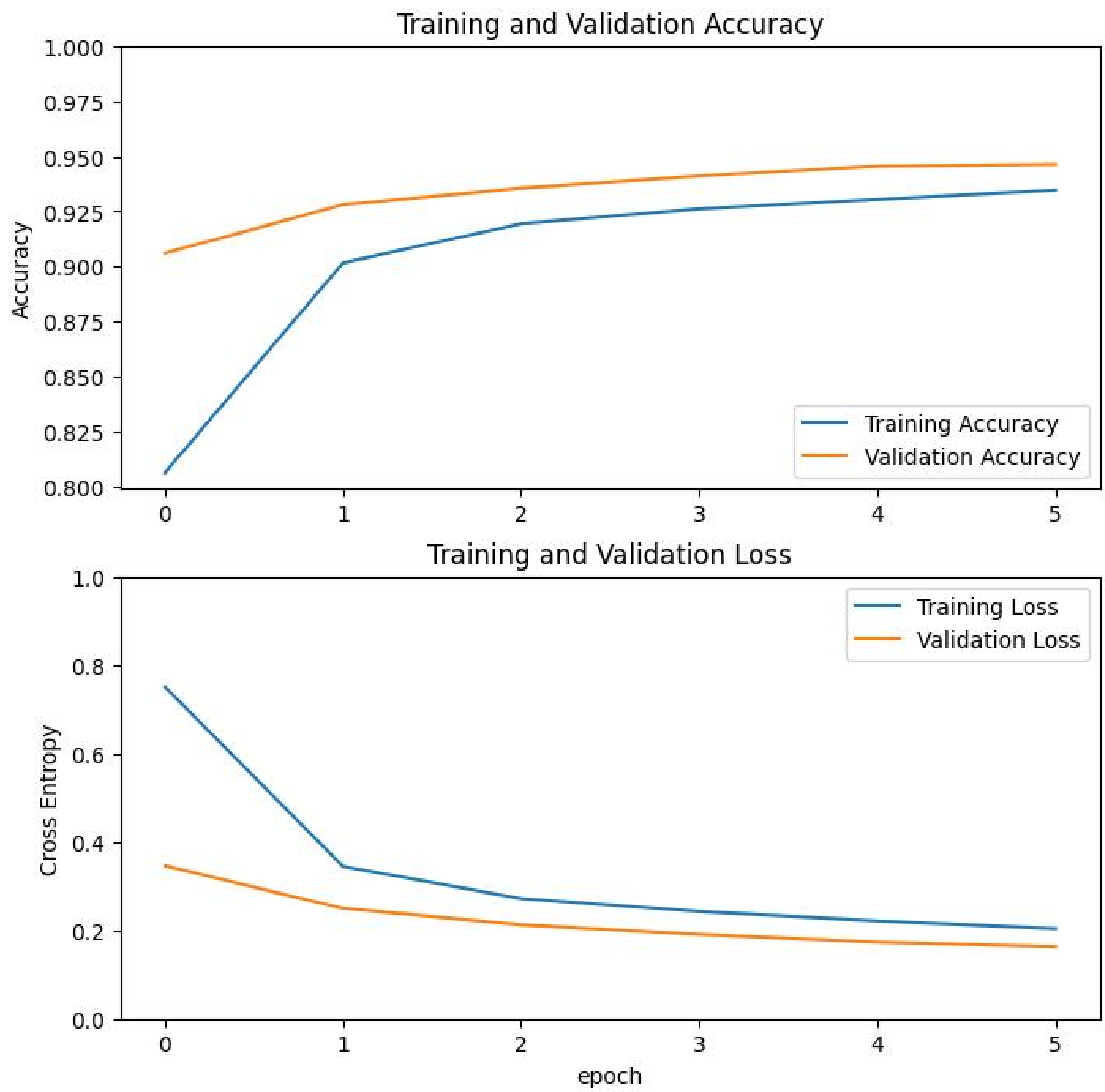
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/6
1537/1537 ━━━━━━━━━━━━━━━━━━━━ 156s 77ms/step - accuracy: 0.6928 - loss: 1.2347 - val_accuracy: 0.9060 - val_loss: 0.3458
Epoch 2/6
1537/1537 ━━━━━━━━━━━━━━━━━━━━ 103s 65ms/step - accuracy: 0.8957 - loss: 0.3725 - val_accuracy: 0.9280 - val_loss: 0.2492
Epoch 3/6
1537/1537 ━━━━━━━━━━━━━━━━━━━━ 142s 65ms/step - accuracy: 0.9184 - loss: 0.2810 - val_accuracy: 0.9355 - val_loss: 0.2124
Epoch 4/6
1537/1537 ━━━━━━━━━━━━━━━━━━━━ 100s 65ms/step - accuracy: 0.9255 - loss: 0.2491 - val_accuracy: 0.9410 - val_loss: 0.1906
Epoch 5/6
1537/1537 ━━━━━━━━━━━━━━━━━━━━ 98s 64ms/step - accuracy: 0.9293 - loss: 0.2245 - val_accuracy: 0.9456 - val_loss: 0.1731
Epoch 6/6
1537/1537 ━━━━━━━━━━━━━━━━━━━━ 143s 64ms/step - accuracy: 0.9349 - loss: 0.2062 - val_accuracy: 0.9464 - val_loss: 0.1626
```

Training Monitoring & Visualization

```
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=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.axvline(x=initial_epochs-1, color='r', linestyle='--', label='Start Fine Tunin
g')
plt.legend(loc='lower right')
```

```
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.axvline(x=initial_epochs-1, color='r', linestyle='--', label='Start Fine Tunin g')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```


Training and Validation Accuracy
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
Accuracy
0
1
2
3
4
5
Training Accuracy
Validation Accuracy
Training and Validation Loss
1.0
0.8
0.6
0.4
0.2
0.0
Cross Entropy
0
1
2
3
4
5
epoch
Training Loss
Validation Loss

Evaluation & Results

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy:', accuracy)

# Predict and visualize results
image_batch, label_batch = next(iter(test_dataset))
predictions = model.predict_on_batch(image_batch)
predicted_labels = tf.argmax(predictions, axis=1)

plt.figure(figsize=(10, 10)) for
i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image_batch[i].numpy().astype("uint8"))
    true_label = class_names[label_batch[i]]
    pred_label = class_names[predicted_labels[i]]
    plt.title(f"Pred: {pred_label}\\nTrue: {true_label}")
    plt.axis("off")
```

Model Deployment

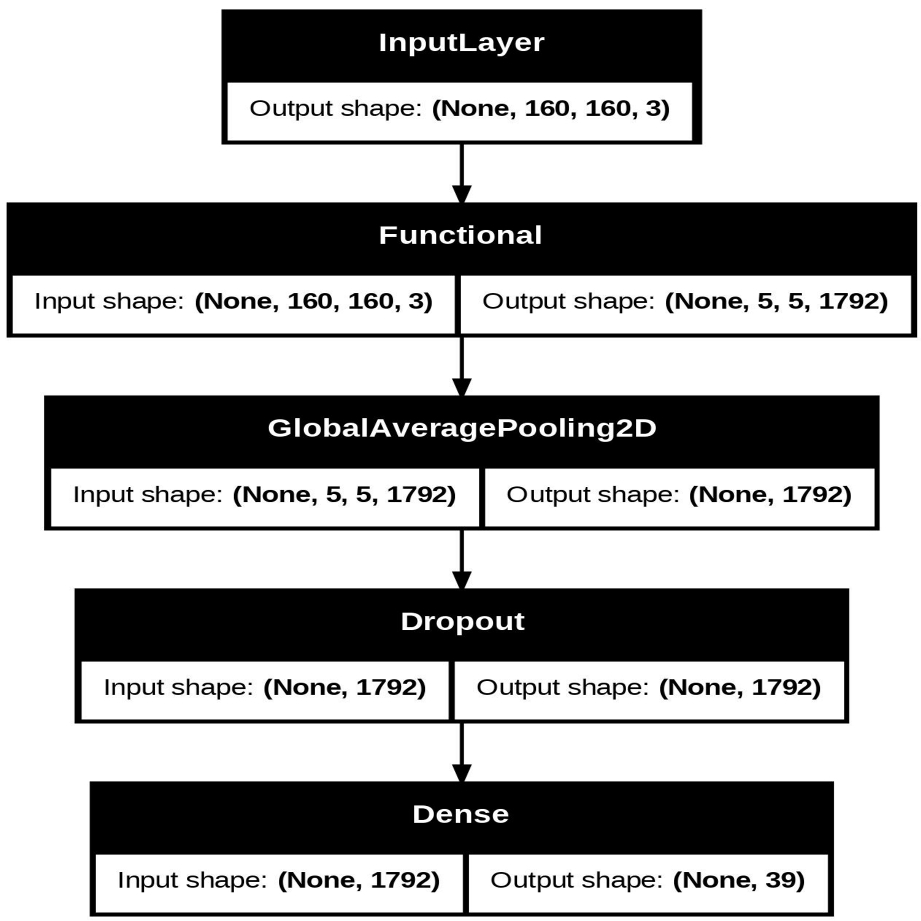
```
model.save("model.keras")
```

3. Model Architecture

Base Architecture: EfficientNetB4

The model implements **EfficientNetB4** as the backbone architecture, which represents a state-of-the-art

convolutional neural network optimized for efficiency and accuracy.



Architecture Specifications:

Total Parameters: 19M parameters

Input Resolution: 160 160 3

Output Classes: Multiple plant disease categories

Activation Functions: Swish (primary), Softmax (output)

Normalization: Batch Normalization throughout

4. Why EfficientNetB4 Was Selected

Efficiency-Accuracy Trade-off:

EfficientNetB4 provides an optimal balance between computational efficiency and classification accuracy. The compound scaling method systematically scales network width, depth, and resolution using a simple yet effective compound coefficient.

Transfer Learning Benefits:

Pre-trained Weights: Leverages ImageNet pre-training for robust feature extraction

Domain Adaptation: Plant imagery shares visual characteristics with natural images

Reduced Training Time: Significantly faster convergence compared to training from scratch.

Mobile-Friendly Architecture:

Lightweight Design: Suitable for deployment on resource-constrained devices

Optimized Operations: Mobile inverted bottlenecks reduce computational overhead

Efficient Memory Usage: Lower memory footprint compared to traditional CNNs

Proven Performance in Agricultural Applications:

Strong performance on image classification tasksEffective feature extraction for plant disease detection

Robust to variations in lighting, angle, and image quality

Scalability Considerations:

Easy to scale up/down based on computational requirements

Consistent architecture across different model sizes

Well-supported in TensorFlow/Keras ecosystem

---

5. Model Evaluation Report

Classification Metrics:

```
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
```

Accuracy: 0.9569
Precision: 0.9598
Recall: 0.9569
MSE: 4.6521
Confusion Matrix:

```
[[100 0 0 ... 0 0 0]
 [ 1 95 0 ... 0 0 0]
 [ 0 0 100 ... 0 0 0]
 ...
[ 0 0 0 ... 534 0 0]
 [ 0 0 0 ... 0 100 0]
[ 0 0 0 ... 0 0 159]]
```

Intersection over Union(IoU):

IoU for class Apple__Apple_scab:
0.8621 IoU for class Apple____
Black_rot: 0.9048
IoU for class Apple___Cedar_apple_rust:
0.9901 IoU for class Apple_____healthy:
0.7961
IoU for class Background_without_leaves: 0.9569
IoU for class Blueberry___healthy: 0.9869
IoU for class Cherry___Powdery_mildew: 0.9528
IoU for class Cherry___healthy: 0.9706
IoU for class Corn__Cercospora_leaf_spot Gray_leaf_spot:
0.7667 IoU for class Corn_____Common_rust: 0.9833
IoU for class Corn___Northern_Leaf_Blight: 0.7500
IoU for class Corn___healthy: 0.9915

IoU for class Grape___Black_rot: 0.7584
IoU for class Grape___Esca_(Black_Measles): 0.9574
IoU for class Grape___Leaf_blight_(Isariopsis_Leaf_Spot): 0.9908
IoU for class Grape___healthy: 0.6863
IoU for class Orange___Haunglongbing_(Citrus_greening): 0.9892
IoU for class Peach___Bacterial_spot: 0.9660
IoU for class Peach___healthy: 0.5727
IoU for class Pepper,_bell___Bacterial_spot: 0.9208
IoU for class Pepper,_bell___healthy: 0.9156
IoU for class Potato___Early_blight: 0.9608
IoU for class Potato___Late_blight: 0.8738
IoU for class Potato___healthy: 0.9412
IoU for class Raspberry__healthy: 0.9802
IoU for class Soybean__healthy: 0.9883
IoU for class Squash___Powdery_mildew: 0.9946
IoU for class Strawberry___Leaf_scorch: 0.8346
IoU for class Strawberry___healthy: 0.8100
IoU for class Tomato___Bacterial_spot: 0.9464
IoU for class Tomato___Early_blight: 0.6275
IoU for class Tomato__Late_blight: 0.8679
IoU for class Tomato__Leaf_Mold:
0.9216
IoU for class Tomato___Septoria_leaf_spot: 0.9239
IoU for class Tomato___Spider_mites Two-spotted_spider_mite: 0.8889
IoU for class Tomato___Target_Spot: 0.8232
IoU for class Tomato__Tomato_Yellow_Leaf_Curl_Virus:
0.9871 IoU for class Tomato___Tomato_mosaic_virus: 0.9709
IoU for class Tomato__healthy: 0.9876

Optimization techniques used to increase the performance

Performance Enhancement Techniques used

1. Data Pipeline Optimization

python

```
AUTOTUNE      tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Benefits:

Overlaps data preprocessing with model execution

Reduces training time by 15-30%

Eliminates I/O bottlenecks

2. Transfer Learning Strategy

Pre-trained Backbone: EfficientNetB4 with ImageNet weights

Fine-tuning Approach: Adapted final layers for plant disease classification

Reduced Training Time: 6 initial epochs vs. 50+ from scratch

3. Batch Size OptimizationBatch Size: 32 (balanced for memory and convergence)

Memory Efficiency: Prevents OOM errors on limited GPU memory

Gradient Stability: Sufficient samples for stable gradient updates

4. Batch Size OptimizationBatch Size: 32 (balanced for memory and convergence)

Memory Efficiency: Prevents OOM errors on limited GPU memory

Gradient Stability: Sufficient samples for stable gradient updates

5. Image Resolution Selection

Input Size: 160 160 pixels

Computational Efficiency: Faster inference while maintaining accuracy

Mobile Deployment: Suitable for real-time applications

6. Model Architecture Efficiency

EfficientNet Scaling: Compound scaling for optimal resource utilization

Squeeze-and-Excitation: Channel attention mechanism

Mobile Inverted Bottlenecks: Reduced parameter count

Deployed model: https://hackathonpavaman.streamlit.app/

Source code : https://github.com/bmuralisridharan/Hackathon_Pavaman