In [21]:
```
#Importing Libraries
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

In [22]:
```
tf.config.list_physical_devices('GPU')
```

Out[22]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Loading Data

In [23]:
```
#Dataset Path
train_dir = 'C:/Users/reddy/OneDrive/Desktop/HELLO/22 ML Projects/Plant Disease Det
```

In [24]:
```
# Data augmentation and generator setup (with 20% validation split)
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    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'
)

# Training generator
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

# Validation generator
val_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 44371 images belonging to 39 classes.
Found 11077 images belonging to 39 classes.
```

In [25]:
```
# Dynamically determine the number of classes from the folder structure
num_classes = len(train_generator.class_indices)
print("Number of classes in the dataset:", num_classes)
```

```
Number of classes in the dataset: 39
```

Transfer Learning

In [26]:
```
base_model = MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
base_model.trainable = False  # Freeze the base model initially
```

In [27]:
```
model = Sequential([
    base_model,
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')  # Output layer now has 39 units
])
```

In [28]:
```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [29]:
```
# Define callbacks for early stopping and saving the best model.
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

checkpoint = ModelCheckpoint(
    'best_model.h5',
    monitor='val_loss',
    save_best_only=True
)
```

In [30]:
```
# Initial training phase: train only the top layers.
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size,
    epochs=30,
    callbacks=[early_stop, checkpoint]
)
```

```
Epoch 1/30
1386/1386 [==============================] - 736s 527ms/step - loss: 1.5794 - accura
cy: 0.5542 - val_loss: 0.6429 - val_accuracy: 0.8168
Epoch 2/30
1386/1386 [==============================] - 329s 238ms/step - loss: 0.9745 - accura
cy: 0.6991 - val_loss: 0.4784 - val_accuracy: 0.8583
Epoch 3/30
1386/1386 [==============================] - 310s 224ms/step - loss: 0.8566 - accura
cy: 0.7344 - val_loss: 0.4178 - val_accuracy: 0.8640
Epoch 4/30
1386/1386 [==============================] - 294s 212ms/step - loss: 0.7879 - accura
cy: 0.7562 - val_loss: 0.3845 - val_accuracy: 0.8748
Epoch 5/30
1386/1386 [==============================] - 299s 216ms/step - loss: 0.7624 - accura
cy: 0.7631 - val_loss: 0.3746 - val_accuracy: 0.8776
Epoch 6/30
1386/1386 [==============================] - 296s 213ms/step - loss: 0.7291 - accura
cy: 0.7726 - val_loss: 0.3599 - val_accuracy: 0.8960
Epoch 7/30
1386/1386 [==============================] - 293s 212ms/step - loss: 0.7126 - accura
cy: 0.7817 - val_loss: 0.3625 - val_accuracy: 0.8998
Epoch 8/30
1386/1386 [==============================] - 292s 211ms/step - loss: 0.6920 - accura
cy: 0.7875 - val_loss: 0.3490 - val_accuracy: 0.8964
Epoch 9/30
1386/1386 [==============================] - 292s 211ms/step - loss: 0.6825 - accura
cy: 0.7912 - val_loss: 0.3231 - val_accuracy: 0.9083
Epoch 10/30
1386/1386 [==============================] - 292s 211ms/step - loss: 0.6697 - accura
cy: 0.7950 - val_loss: 0.3501 - val_accuracy: 0.8929
Epoch 11/30
1386/1386 [==============================] - 294s 212ms/step - loss: 0.6615 - accura
cy: 0.7944 - val_loss: 0.3149 - val_accuracy: 0.9048
Epoch 12/30
1386/1386 [==============================] - 295s 213ms/step - loss: 0.6568 - accura
cy: 0.7986 - val_loss: 0.3165 - val_accuracy: 0.9063
Epoch 13/30
1386/1386 [==============================] - 295s 213ms/step - loss: 0.6423 - accura
cy: 0.8011 - val_loss: 0.3122 - val_accuracy: 0.9087
Epoch 14/30
1386/1386 [==============================] - 293s 211ms/step - loss: 0.6320 - accura
cy: 0.8035 - val_loss: 0.3194 - val_accuracy: 0.9033
Epoch 15/30
1386/1386 [==============================] - 293s 212ms/step - loss: 0.6322 - accura
cy: 0.8065 - val_loss: 0.3244 - val_accuracy: 0.9062
Epoch 16/30
1386/1386 [==============================] - 292s 211ms/step - loss: 0.6233 - accura
cy: 0.8101 - val_loss: 0.2968 - val_accuracy: 0.9114
Epoch 17/30
1386/1386 [==============================] - 291s 210ms/step - loss: 0.6263 - accura
cy: 0.8099 - val_loss: 0.3271 - val_accuracy: 0.9009
Epoch 18/30
1386/1386 [==============================] - 292s 211ms/step - loss: 0.6159 - accura
cy: 0.8092 - val_loss: 0.2954 - val_accuracy: 0.9114
Epoch 19/30
1386/1386 [==============================] - 292s 211ms/step - loss: 0.6151 - accura
```

```
cy: 0.8121 - val_loss: 0.2866 - val_accuracy: 0.9175
Epoch 20/30
1386/1386 [==============================] - 292s 211ms/step - loss: 0.6129 - accura
cy: 0.8134 - val_loss: 0.3062 - val_accuracy: 0.9107
Epoch 21/30
1386/1386 [==============================] - 296s 214ms/step - loss: 0.6065 - accura
cy: 0.8168 - val_loss: 0.2881 - val_accuracy: 0.9177
Epoch 22/30
1386/1386 [==============================] - 293s 212ms/step - loss: 0.5918 - accura
cy: 0.8179 - val_loss: 0.2763 - val_accuracy: 0.9165
Epoch 23/30
1386/1386 [==============================] - 293s 211ms/step - loss: 0.6010 - accura
cy: 0.8174 - val_loss: 0.2934 - val_accuracy: 0.9175
Epoch 24/30
1386/1386 [==============================] - 298s 215ms/step - loss: 0.5906 - accura
cy: 0.8197 - val_loss: 0.3012 - val_accuracy: 0.9136
Epoch 25/30
1386/1386 [==============================] - 293s 212ms/step - loss: 0.5909 - accura
cy: 0.8218 - val_loss: 0.2903 - val_accuracy: 0.9144
Epoch 26/30
1386/1386 [==============================] - 294s 212ms/step - loss: 0.5811 - accura
cy: 0.8224 - val_loss: 0.2913 - val_accuracy: 0.9143
Epoch 27/30
1386/1386 [==============================] - 412s 297ms/step - loss: 0.5926 - accura
cy: 0.8194 - val_loss: 0.2781 - val_accuracy: 0.9252
```

Fine-Tuning

In [31]:
```
#Freezing 1st 100 layers
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

In [32]:
```
# Recompile the model with a lower learning rate for fine-tuning.
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [33]:
```
# Fine-tune the model for additional epochs.
history_fine = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size,
    epochs=10,  # Adjust epochs as needed based on validation performance.
    callbacks=[early_stop, checkpoint]
)
```

```
Epoch 1/10
1386/1386 [==============================] - 472s 338ms/step - loss: 0.5684 - accura
cy: 0.8251 - val_loss: 0.2660 - val_accuracy: 0.9210
Epoch 2/10
1386/1386 [==============================] - 479s 345ms/step - loss: 0.5374 - accura
cy: 0.8342 - val_loss: 0.2553 - val_accuracy: 0.9259
Epoch 3/10
1386/1386 [==============================] - 504s 363ms/step - loss: 0.5321 - accura
cy: 0.8365 - val_loss: 0.2597 - val_accuracy: 0.9237
Epoch 4/10
1386/1386 [==============================] - 507s 366ms/step - loss: 0.5200 - accura
cy: 0.8393 - val_loss: 0.2604 - val_accuracy: 0.9254
Epoch 5/10
1386/1386 [==============================] - 563s 406ms/step - loss: 0.5191 - accura
cy: 0.8405 - val_loss: 0.2531 - val_accuracy: 0.9242
Epoch 6/10
1386/1386 [==============================] - 509s 367ms/step - loss: 0.5231 - accura
cy: 0.8379 - val_loss: 0.2597 - val_accuracy: 0.9262
Epoch 7/10
1386/1386 [==============================] - 548s 395ms/step - loss: 0.5153 - accura
cy: 0.8405 - val_loss: 0.2533 - val_accuracy: 0.9255
Epoch 8/10
1386/1386 [==============================] - 474s 342ms/step - loss: 0.5135 - accura
cy: 0.8433 - val_loss: 0.2557 - val_accuracy: 0.9256
Epoch 9/10
1386/1386 [==============================] - 344s 248ms/step - loss: 0.5048 - accura
cy: 0.8448 - val_loss: 0.2565 - val_accuracy: 0.9249
Epoch 10/10
1386/1386 [==============================] - 374s 270ms/step - loss: 0.5134 - accura
cy: 0.8426 - val_loss: 0.2553 - val_accuracy: 0.9254
```

In [34]:

```python
print("Training complete. Best model saved as 'best_model.h5'.")
```

```
Training complete. Best model saved as 'best_model.h5'.
```

Examples

In [38]:

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

# Get a single batch of images and labels from the validation generator.
# If you prefer, you could also use your training generator or load images from dis
x_val, y_val = next(val_generator)

# Use the trained model to predict the classes for these images.
predictions = model.predict(x_val)

# Convert the prediction probabilities into class labels (indices).
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(y_val, axis=1)

# Retrieve class names from the training generator's mapping.
# This assumes that your train_generator was set up to automatically map folder nam
class_indices = train_generator.class_indices
# Invert the dictionary to map indices back to class names.
```

```
class_names = {v: k for k, v in class_indices.items()}

# Plot a grid of 9 sample images along with their predicted and true labels.
plt.figure(figsize=(12, 12))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(x_val[i])
    plt.title(f"Pred: {class_names[predicted_classes[i]]}\nTrue: {class_names[true_
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 0s 27ms/step
```