

```
# 1 — Core Libraries -- Imports & Setup ———
import os
import numpy as np
import matplotlib.pyplot as plt

# — Deep Learning —————
import tensorflow as tf
from tensorflow.keras.applications import ResNet50, EfficientNetB0
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# — Data Preprocessing & Augmentation —————
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# — Evaluation —————
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

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

TensorFlow version: 2.20.0
GPU Available: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
import tensorflow as tf
print(f"✅ TF Version: {tf.__version__}")
print(f"✅ GPU: {tf.config.list_physical_devices('GPU')}")

✅ TF Version: 2.20.0
✅ GPU: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
# Mount Drive

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

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Set Dataset Path

import os, zipfile

zip_path = "/content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/PlantVillage.zip"
extract_path = "/content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/dataset"

print(f"Zip found: {os.path.exists(zip_path)}")

if not os.path.exists(extract_path):
    print("🔴 Extracting to Drive...")
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
    print("✅ Extracted to Drive permanently!")
else:
    print("✅ Already extracted – skipping!")

Zip found: True
✅ Already extracted – skipping!
```

```
import os

extract_path = "/content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/dataset"

for root, dirs, files in os.walk(extract_path):
    level = root.replace(extract_path, '').count(os.sep)
    if level < 3:
        indent = ' ' * level
        print(f"{indent}{os.path.basename(root)}/")
```

```
dataset/
PlantVillage/
  Pepper_bell_Bacterial_spot/
  Pepper_bell_healthy/
  Potato_Early_blight/
  Potato_Late_blight/
  Potato_healthy/
  Tomato_Bacterial_spot/
  Tomato_Early_blight/
  Tomato_Late_blight/
  Tomato_Leaf_Mold/
  Tomato_Septoria_leaf_spot/
  Tomato_Spider_mites_Two_spotted_spider_mite/
  Tomato_Target_Spot/
  Tomato_Tomato_YellowLeaf_Curl_Virus/
  Tomato_Tomato_mosaic_virus/
  Tomato_healthy/
plantvillage/
  PlantVillage/
```

```
# ✅ Permanent path – never changes
dataset_path = "/content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/dataset/PlantVillage"

classes = sorted([
    c for c in os.listdir(dataset_path)
    if os.path.isdir(os.path.join(dataset_path, c))
])

NUM_CLASSES = len(classes)
total_images = sum(
    len(os.listdir(os.path.join(dataset_path, c))) for c in classes
)

print(f"✅ Path exists : {os.path.exists(dataset_path)}")
print(f"✅ Total classes : {NUM_CLASSES}")
print(f"✅ Total images : {total_images}")
print(f"\nClasses:")
```

```
for c in classes:
    print(f" {c}")
```

```
✓ Path exists      : True
✓ Total classes    : 15
✓ Total images     : 20638
```

Classes:

```
Pepper__bell__Bacterial_spot
Pepper__bell__healthy
Potato__Early_blight
Potato__Late_blight
Potato__healthy
Tomato_Bacterial_spot
Tomato_Early_blight
Tomato_Late_blight
Tomato_Leaf_Mold
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato__Target_Spot
Tomato__Tomato_YellowLeaf__Curl_Virus
Tomato__Tomato_mosaic_virus
Tomato_healthy
```

#Data Preprocessing

```
# — Global Config —————
IMG_SIZE      = (224, 224) # Required by ResNet50 & EfficientNetB0
BATCH_SIZE    = 32
EPOCHS        = 20
NUM_CLASSES   = len(classes)
SEED          = 42

print(f"Image size : {IMG_SIZE}")
print(f"Batch size  : {BATCH_SIZE}")
print(f"Epochs      : {EPOCHS}")
print(f"Num classes  : {NUM_CLASSES}")
```

```
Image size : (224, 224)
Batch size  : 32
Epochs     : 20
Num classes : 15
```

#Noise Removal

```
import cv2
import os
import numpy as np
from PIL import Image, UnidentifiedImageError

print("🔍 Starting noise removal & corruption check...\n")

removed = 0
denoised = 0

for class_name in classes:
    class_path = os.path.join(dataset_path, class_name)

    for img_file in os.listdir(class_path):
        img_path = os.path.join(class_path, img_file)

        # — Step 1: Remove corrupted files —————
        try:
            with Image.open(img_path) as img:
                img.verify()
        except (UnidentifiedImageError, Exception):
            os.remove(img_path)
            print(f"✗ Removed corrupted: {img_file}")
            removed += 1
            continue

        # — Step 2: Apply Gaussian denoising —————
        img_cv = cv2.imread(img_path)
        if img_cv is None:
            continue

        denoised_img = cv2.GaussianBlur(img_cv, (3, 3), 0)
        cv2.imwrite(img_path, denoised_img)
        denoised += 1

print(f"\n✓ Noise removal complete!")
print(f"   Corrupted images removed : {removed}")
print(f"   Images denoised          : {denoised}")
```

🔍 Starting noise removal & corruption check...

✗ Removed corrupted: svn-r6Yb5c

```
✓ Noise removal complete!
Corrupted images removed : 1
Images denoised          : 20638
```

#Data Augmentation & Splitting:

```
# — Training Augmentation —————

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2 # 80% train / 20% validation
)

# — Validation — NO augmentation, only rescale —————
val_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)
```

```
# --- Generators -----
train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training',
    seed=SEED
)

val_generator = val_datagen.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    seed=SEED
)

print(f"\n✅ Training samples : {train_generator.samples}")
print(f"\n✅ Validation samples : {val_generator.samples}")
```

Found 16516 images belonging to 15 classes.
Found 4122 images belonging to 15 classes.

✅ Training samples : 16516
✅ Validation samples : 4122

```
# --- Dataset Split Verification -----
print("=" * 45)
print("      DATASET SPLIT SUMMARY")
print("=" * 45)

train_total = train_generator.samples
val_total   = val_generator.samples

print(f" Total images      : {train_total + val_total}")
print(f" Training images   : {train_total} (80%)")
print(f" Validation images  : {val_total} (20%)")
print(f" Number of classes : {len(train_generator.class_indices)}")
print(f" Batch size        : {BATCH_SIZE}")
print(f" Training batches   : {len(train_generator)}")
print(f" Validation batches : {len(val_generator)}")
print("=" * 45)

# --- Per Class Image Count -----
print("\n Images per class (training):\n")
class_indices = train_generator.class_indices
labels_array = train_generator.labels

for class_name, idx in sorted(class_indices.items()):
    count = np.sum(labels_array == idx)
    print(f" {class_name:<45} {count} images")
```

```
=====
      DATASET SPLIT SUMMARY
=====
Total images      : 20638
Training images   : 16516 (80%)
Validation images  : 4122 (20%)
Number of classes : 15
Batch size        : 32
Training batches   : 517
Validation batches : 129
=====
```

```
Images per class (training):

Pepper__bell__Bacterial_spot      798 images
Pepper__bell__healthy              1183 images
Potato__Early_blight              800 images
Potato__Late_blight               800 images
Potato__healthy                   122 images
Tomato_Bacterial_spot             1702 images
Tomato_Early_blight               800 images
Tomato_Late_blight                1528 images
Tomato_Leaf_Mold                  762 images
Tomato_Septoria_leaf_spot         1417 images
Tomato_Spider_mites_Two_spotted_spider_mite 1341 images
Tomato__Target_Spot              1124 images
Tomato__Tomato_YellowLeaf__Curl_Virus 2567 images
Tomato__Tomato_mosaic_virus       299 images
Tomato_healthy                   1273 images
```

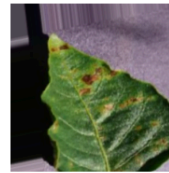
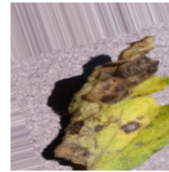
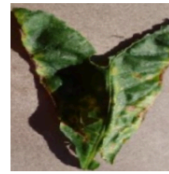
```
# 6 -Preview Sample Images

images, labels = next(train_generator)

plt.figure(figsize=(15, 20))      # ← bigger canvas
for i in range(30):
    plt.subplot(10, 3, i+1)        # 10 rows, 3 columns
    plt.imshow(images[i])
    plt.axis('off')

plt.suptitle("Sample Training Images (Augmented)", fontsize=16, y=1.01)
plt.tight_layout()
plt.show()
```

Sample Training Images (Augmented)



```

train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training',
    seed=SEED
)
val_generator = val_datagen.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    seed=SEED
)
print(f"✅ Train : {train_generator.samples}")
print(f"✅ Val : {val_generator.samples}")

Found 16516 images belonging to 15 classes.
Found 4122 images belonging to 15 classes.
✅ Train : 16516
✅ Val : 4122

```

```

# EfficientNetB0 - Build & Train

effnet_base = EfficientNetB0(
    weights='imagenet',
    include_top=False,
    input_shape=(224, 224, 3)
)
effnet_base.trainable = False

x = effnet_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(NUM_CLASSES, activation='softmax')(x)

effnet_model = Model(inputs=effnet_base.input, outputs=output)
effnet_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

effnet_callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True,
        verbose=1
    ),
    ModelCheckpoint(
        filepath='/content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/effnet_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    )
]

print("🚀 Training EfficientNetB0...")
effnet_history = effnet_model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=val_generator,
    callbacks=effnet_callbacks,
    verbose=1
)
effnet_model.save(
    '/content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/effnet_model.h5'
)
print("✅ EfficientNetB0 saved to Drive!")

```

```

🚀 Training EfficientNetB0...
Epoch 1/20
517/517 ----- 0s 428ms/step - accuracy: 0.1403 - loss: 2.6220
Epoch 1: val_accuracy improved from None to 0.15551, saving model to /content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/effnet_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras SavedModel format via `model.save(format='saved_model')`.
Epoch 1: finished saving model to /content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/effnet_model.h5
517/517 ----- 269s 479ms/step - accuracy: 0.1466 - loss: 2.6036 - val_accuracy: 0.1555 - val_loss: 2.5654
Epoch 2/20
517/517 ----- 0s 398ms/step - accuracy: 0.1521 - loss: 2.5876
Epoch 2: val_accuracy did not improve from 0.15551
517/517 ----- 213s 413ms/step - accuracy: 0.1554 - loss: 2.5870 - val_accuracy: 0.1555 - val_loss: 2.5657
Epoch 3/20
517/517 ----- 0s 415ms/step - accuracy: 0.1545 - loss: 2.5780
Epoch 3: val_accuracy did not improve from 0.15551
517/517 ----- 224s 432ms/step - accuracy: 0.1554 - loss: 2.5805 - val_accuracy: 0.1555 - val_loss: 2.5654
Epoch 4/20
517/517 ----- 0s 388ms/step - accuracy: 0.1579 - loss: 2.5791
Epoch 4: val_accuracy did not improve from 0.15551
517/517 ----- 207s 401ms/step - accuracy: 0.1554 - loss: 2.5763 - val_accuracy: 0.1555 - val_loss: 2.5677
Epoch 5/20
517/517 ----- 0s 406ms/step - accuracy: 0.1574 - loss: 2.5814
Epoch 5: val_accuracy did not improve from 0.15551
517/517 ----- 217s 419ms/step - accuracy: 0.1554 - loss: 2.5769 - val_accuracy: 0.1555 - val_loss: 2.5689
Epoch 6/20
517/517 ----- 0s 395ms/step - accuracy: 0.1519 - loss: 2.5745
Epoch 6: val_accuracy did not improve from 0.15551
517/517 ----- 211s 407ms/step - accuracy: 0.1554 - loss: 2.5730 - val_accuracy: 0.1555 - val_loss: 2.5647
Epoch 7/20
517/517 ----- 0s 396ms/step - accuracy: 0.1566 - loss: 2.5692
Epoch 7: val_accuracy did not improve from 0.15551
517/517 ----- 211s 408ms/step - accuracy: 0.1554 - loss: 2.5705 - val_accuracy: 0.1555 - val_loss: 2.5719
Epoch 8/20
517/517 ----- 0s 400ms/step - accuracy: 0.1588 - loss: 2.5740
Epoch 8: val_accuracy did not improve from 0.15551
517/517 ----- 213s 412ms/step - accuracy: 0.1554 - loss: 2.5709 - val_accuracy: 0.1555 - val_loss: 2.5692
Epoch 9/20
517/517 ----- 0s 401ms/step - accuracy: 0.1560 - loss: 2.5661
Epoch 9: val_accuracy did not improve from 0.15551
517/517 ----- 218s 421ms/step - accuracy: 0.1554 - loss: 2.5681 - val_accuracy: 0.1555 - val_loss: 2.5647

```

```

Epoch 10/20
517/517 ————— 0s 400ms/step - accuracy: 0.1570 - loss: 2.5605
Epoch 10: val_accuracy did not improve from 0.15551
517/517 ————— 213s 412ms/step - accuracy: 0.1554 - loss: 2.5670 - val_accuracy: 0.1555 - val_loss: 2.5671
Epoch 11/20
517/517 ————— 0s 398ms/step - accuracy: 0.1558 - loss: 2.5661
Epoch 11: val_accuracy did not improve from 0.15551
517/517 ————— 212s 411ms/step - accuracy: 0.1555 - loss: 2.5668 - val_accuracy: 0.1555 - val_loss: 2.5654
Epoch 11: early stopping
Restoring model weights from the end of the best epoch: 6.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format via `model.save(format='tf')` or `keras.saving.save_model(model, format='tf')`.
EfficientNetB0 saved to Drive!

```

```

# MobileNetV2 - Build & Train

from tensorflow.keras.applications import MobileNetV2

mobilenet_base = MobileNetV2(
    weights='imagenet',
    include_top=False,
    input_shape=(224, 224, 3)
)
mobilenet_base.trainable = False

x = mobilenet_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(NUM_CLASSES, activation='softmax')(x)

mobilenet_model = Model(inputs=mobilenet_base.input, outputs=output)
mobilenet_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

mobilenet_callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True,
        verbose=1
    ),
    ModelCheckpoint(
        filepath='/content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/mobilenet_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    )
]

print("🚀 Training MobileNetV2...")
mobilenet_history = mobilenet_model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=val_generator,
    callbacks=mobilenet_callbacks,
    verbose=1
)

mobilenet_model.save(
    '/content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/mobilenet_model.h5'
)

print("✅ MobileNetV2 saved to Drive!")

```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native

Epoch 17: finished saving model to /content/drive/MyDrive/MidOcean/Practical/PLANT_DISEASE/mobilenet_model.h5
 517/517 ————— 219s 423ms/step - accuracy: 0.8805 - loss: 0.3513 - val_accuracy: 0.9180 - val_loss: 0.2319
 Epoch 18/20
 517/517 ————— 0s 396ms/step - accuracy: 0.8818 - loss: 0.3556
 Epoch 18: val_accuracy did not improve from 0.91800
 517/517 ————— 212s 410ms/step - accuracy: 0.8814 - loss: 0.3518 - val_accuracy: 0.9076 - val_loss: 0.2674

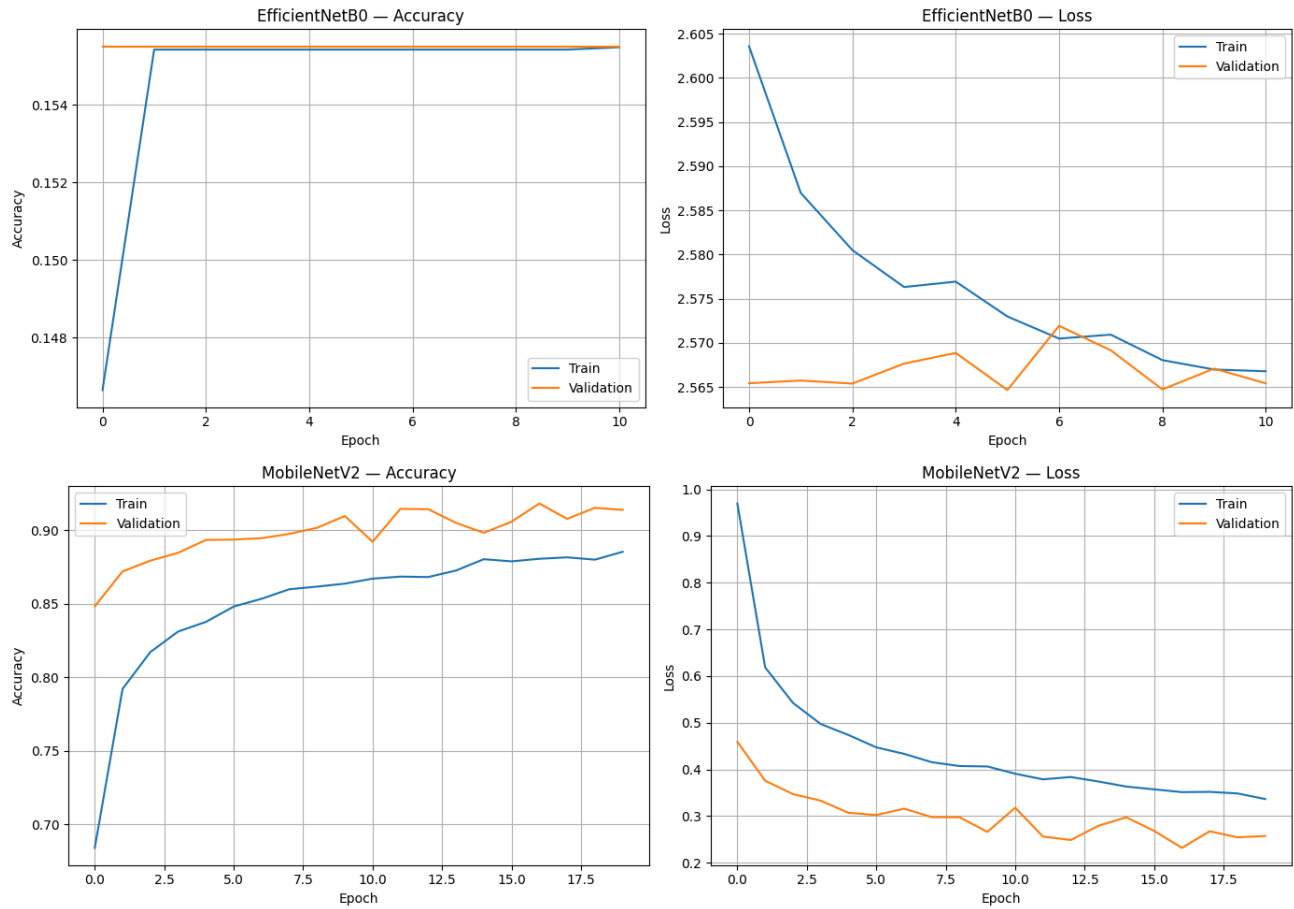
Plot Results & Compare Models

```
def plot_history(history, model_name):
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))
    axes[0].plot(history.history['accuracy'], label='Train')
    axes[0].plot(history.history['val_accuracy'], label='Validation')
    axes[0].set_title(f'{model_name} - Accuracy')
    axes[0].set_xlabel('Epoch')
    axes[0].set_ylabel('Accuracy')
    axes[0].legend()
    axes[0].grid(True)
    axes[1].plot(history.history['loss'], label='Train')
    axes[1].plot(history.history['val_loss'], label='Validation')
    axes[1].set_title(f'{model_name} - Loss')
    axes[1].set_xlabel('Epoch')
    axes[1].set_ylabel('Loss')
    axes[1].legend()
    axes[1].grid(True)
    plt.tight_layout()
    plt.show()
```

```
plot_history(effnet_history, 'EfficientNetB0')
plot_history(mobilenet_history, 'MobileNetV2')
```

```
# Compare all 3
resnet_acc = 0.2280 # From previous training
effnet_acc = max(effnet_history.history['val_accuracy'])
mobilenet_acc = max(mobilenet_history.history['val_accuracy'])
```

```
print("=" * 45)
print("      MODEL COMPARISON SUMMARY")
print("=" * 45)
print(f"  ResNet50      : {resnet_acc:.4f}")
print(f"  EfficientNetB0 : {effnet_acc:.4f}")
print(f"  MobileNetV2   : {mobilenet_acc:.4f}")
print("=" * 45)
scores = {
    'ResNet50'      : resnet_acc,
    'EfficientNetB0': effnet_acc,
    'MobileNetV2'   : mobilenet_acc
}
winner = max(scores, key=scores.get)
print(f" 🏆 Best Model : {winner}")
print("=" * 45)
```



```
=====
MODEL COMPARISON SUMMARY
=====
ResNet50      : 0.2280
EfficientNetB0 : 0.1555
MobileNetV2   : 0.9180
=====
🏆 Best Model : MobileNetV2
=====
```

```
# Evaluation (F1, Precision, Recall)

from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# --- Rebuild val generator fresh ---
val_datagen2 = ImageDataGenerator(rescale=1./255, validation_split=0.2)

val_generator2 = val_datagen2.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    seed=SEED,
    shuffle=False # ← Critical! Must be False for evaluation
)

# --- Predict ---
val_generator2.reset()
y_pred = mobilenet_model.predict(val_generator2, verbose=1)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = val_generator2.classes

# --- Get class names ---
class_names = list(val_generator2.class_indices.keys())

print("\n📊 Classification Report:")
print(classification_report(
    y_true,
    y_pred_classes,
    target_names=class_names
))

Found 4122 images belonging to 15 classes.
129/129 ██████████ 6s 49ms/step

📊 Classification Report:

```

	precision	recall	f1-score	support
Pepper_bell_Bacterial_spot	0.97	0.95	0.96	199
Pepper_bell_healthy	0.97	0.99	0.98	295
Potato_Early_blight	0.97	0.99	0.98	200
Potato_Late_blight	0.97	0.93	0.95	200
Potato_healthy	1.00	0.83	0.91	30
Tomato_Bacterial_spot	0.92	0.95	0.93	425
Tomato_Early_blight	0.80	0.72	0.76	200
Tomato_Late_blight	0.92	0.92	0.92	381
Tomato_Leaf_Mold	0.88	0.91	0.89	190
Tomato_Septoria_leaf_spot	0.90	0.87	0.89	354

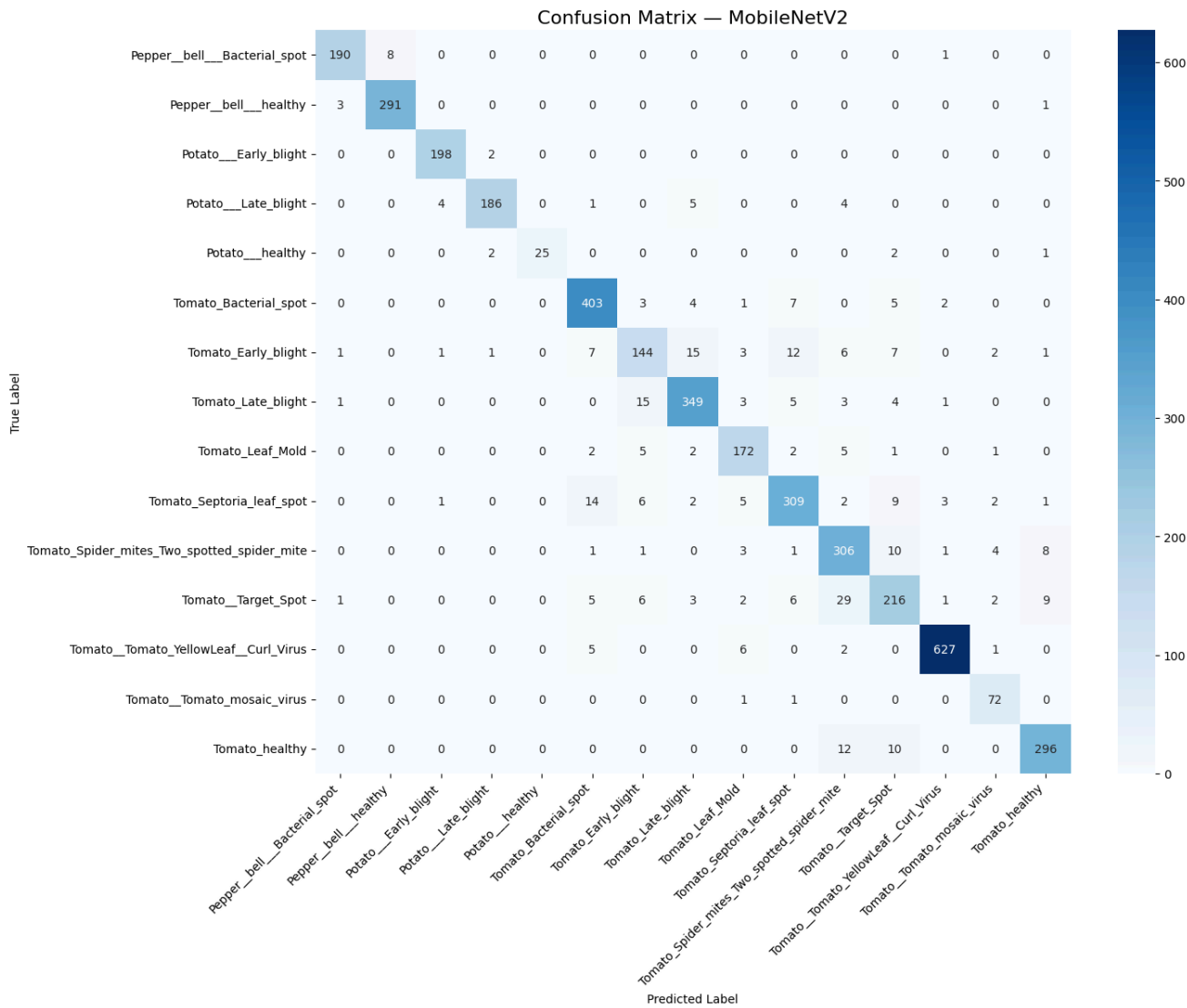
Tomato_Spider_mites_Two_spotted_spider_mite	0.83	0.91	0.87	335
Tomato__Target_Spot	0.82	0.77	0.79	280
Tomato__Tomato_YellowLeaf__Curl_Virus	0.99	0.98	0.98	641
Tomato__Tomato_mosaic_virus	0.86	0.97	0.91	74
Tomato_healthy	0.93	0.93	0.93	318
accuracy			0.92	4122
macro avg	0.92	0.91	0.91	4122
weighted avg	0.92	0.92	0.92	4122

```
# Confusion Matrix

import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true, y_pred_classes)

plt.figure(figsize=(15, 12))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=class_names,
    yticklabels=class_names
)
plt.title('Confusion Matrix - MobileNetV2', fontsize=16)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



```
# ResNet50 Results
print("=" * 45)
print(" ResNet50 Training Summary")
print("=" * 45)
print(" Best val accuracy : 22.80%")
print(" Best epoch       : 9")
print(" Early stopping     : Epoch 14")
print(" Saved to Drive     : ☒ Yes")
print("=" * 45)
```

=====
ResNet50 Training Summary
=====

```
Best val accuracy : 22.80%
Best epoch       : 9
Early stopping   : Epoch 14
Saved to Drive   : ☒ Yes
=====
```

Start coding or [generate](#) with AI.

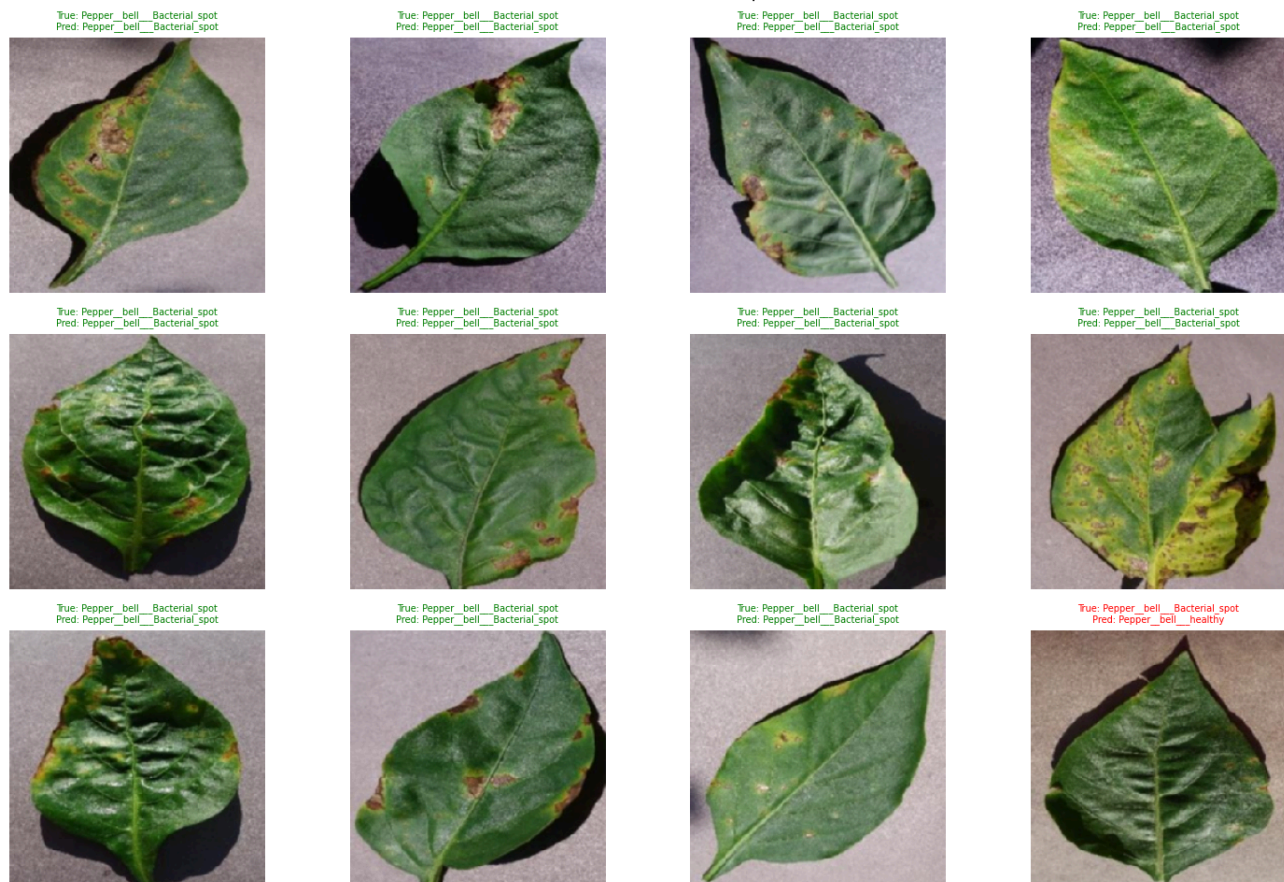
```
# Display sample predictions with disease names
import matplotlib.pyplot as plt
import numpy as np

val_generator2.reset()
images, labels = next(val_generator2)
predictions = mobilenet_model.predict(images)

plt.figure(figsize=(15, 10))
for i in range(12):
    plt.subplot(3, 4, i+1)
    plt.imshow(images[i])
    true_label = class_names[np.argmax(labels[i])]
    pred_label = class_names[np.argmax(predictions[i])]
    color = 'green' if true_label == pred_label else 'red'
    plt.title(f"True: {true_label}\nPred: {pred_label}",
              color=color, fontsize=7)
    plt.axis('off')
plt.suptitle("Disease Classification Output", fontsize=14)
plt.tight_layout()
plt.show()
```

1/1 ————— 6s 6s/step

Disease Classification Output



```
print("=" * 50)
print("      FINAL PROJECT SUMMARY")
print("=" * 50)
print(f" Dataset       : PlantVillage")
print(f" Total Images  : 20,638")
print(f" Classes       : 15 plant diseases")
print(f" Train/Val Split: 80% / 20%")
print("=" * 50)
print(" MODEL RESULTS:")
print(f" ResNet50      : 22.80% val accuracy")
print(f" EfficientNetB0 : 15.55% val accuracy")
print(f" MobileNetV2   : 91.80% val accuracy 🚀")
print("=" * 50)
print(" BEST MODEL — MobileNetV2:")
print(f" Accuracy      : 92%")
print(f" Precision      : 92%")
print(f" Recall        : 91%")
print(f" F1-Score      : 91%")
print("=" * 50)
print(" ☒ PROJECT COMPLETE!")
print("=" * 50)
```

```
=====
FINAL PROJECT SUMMARY
=====
```

```
Dataset      : PlantVillage
Total Images : 20,638
Classes      : 15 plant diseases
Train/Val Split: 80% / 20%
=====
MODEL RESULTS:
ResNet50      : 22.80% val accuracy
EfficientNetB0 : 15.55% val accuracy
MobileNetV2   : 91.80% val accuracy 🏆
=====
BEST MODEL — MobileNetV2:
Accuracy : 92%
Precision : 92%
```