EfficientNETB3 Model Documentation

1. Model Overview: -

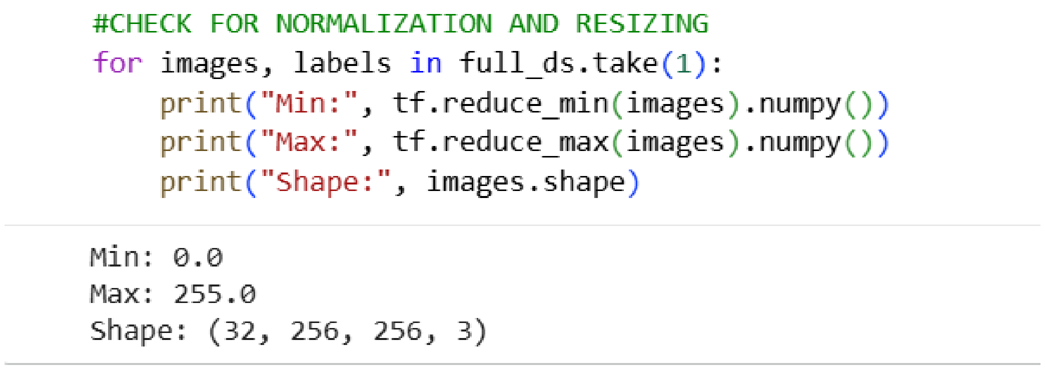
The model is a deep convolutional neural network based on **EfficientNetB3**, adapted for multi-class plant disease classification using transfer learning and fine-tuning. The architecture leverages pre-trained ImageNet weights to extract rich visual features and a custom classification head for domain-specific learning.

2. Model Architecture: -

2.1 Base Network:

Pre-training: ImageNet

Input Shape: 256 × 256 × 3



```
#CHECK FOR NORMALIZATION AND RESIZING
for images, labels in full_ds.take(1):
    print("Min:", tf.reduce_min(images).numpy())
    print("Max:", tf.reduce_max(images).numpy())
    print("Shape:", images.shape)
```

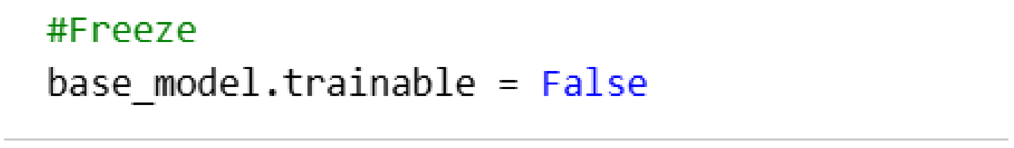
```
Min: 0.0
Max: 255.0
Shape: (32, 256, 256, 3)
```

Top Layers: Removed (include_top = False)

→ The top layers of EfficientNetB3 were removed because they are designed to classify 1000 ImageNet classes, while this task requires a new classifier adapted to a different number of target classes.

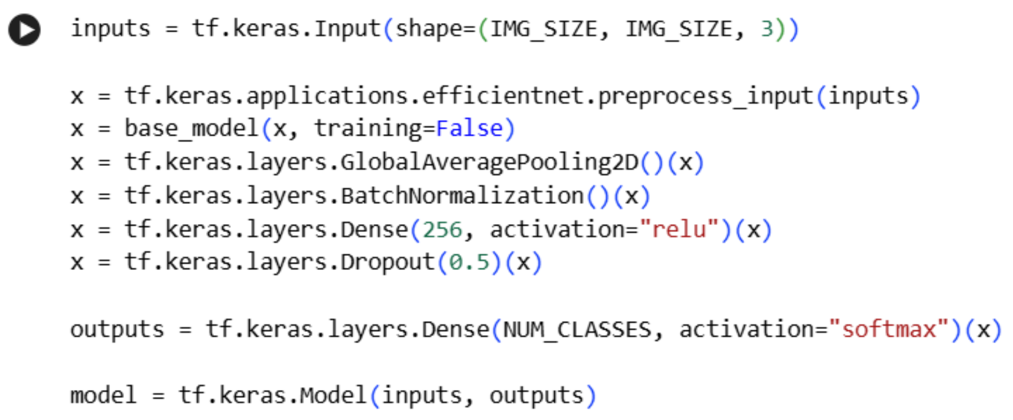
2.2 Feature Extractor: (Transfer learning)

The EfficientNetB3 backbone is used as a **feature extractor** during initial training. All convolutional layers are frozen to preserve learned ImageNet representations.



```
#Freeze
base_model.trainable = False
```

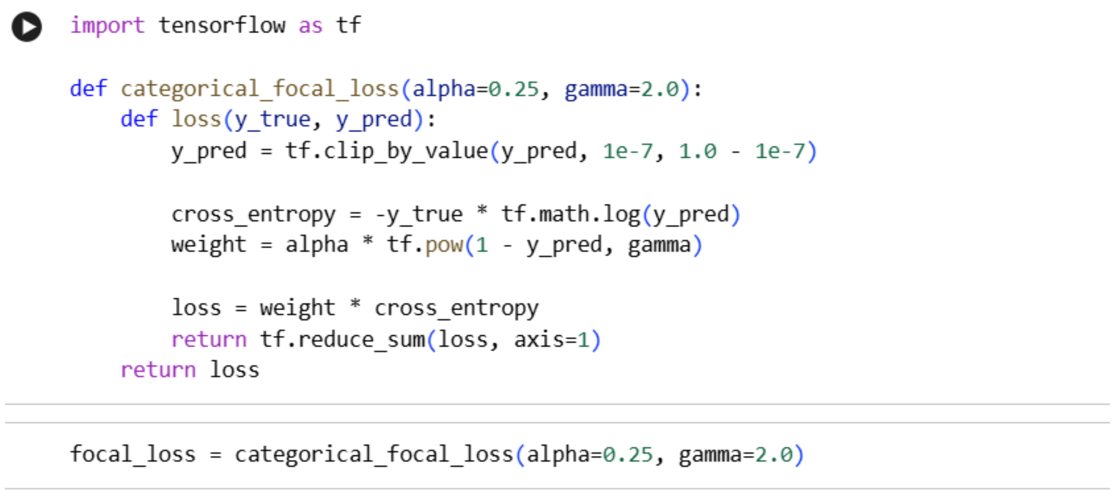
2.3 Classification Head:



```
inputs = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

x = tf.keras.applications.efficientnet.preprocess_input(inputs)
x = base_model(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.Dense(256, activation="relu")(x)
x = tf.keras.layers.Dropout(0.5)(x)

outputs = tf.keras.layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = tf.keras.Model(inputs, outputs)
```

- **The input layer** receives RGB images of size IMG_SIZE×IMG_SIZE×3IMG Before feature extraction, the images are normalized using **the EfficientNet preprocessing function** to match the input distribution used during ImageNet pre-training.
- **training=False**, ensuring that frozen layers and batch normalization statistics **remain unchanged** during feature extraction.
- **Global Average Pooling** is applied to convert the spatial feature maps into a single feature vector, significantly reducing the number of parameters and minimizing overfitting.
- **Batch Normalization** is then used to stabilize training and improve convergence.
- **A fully connected dense layer** with 256 neurons and **ReLU** activation learns high-level, task-specific representations.
- **Dropout** with a rate of 0.5 is applied as a regularization technique to reduce overfitting.
- **A dense output layer** with **softmax** activation produces a **probability** distribution over the target classes, enabling multi-class classification.

3. Loss Function: -

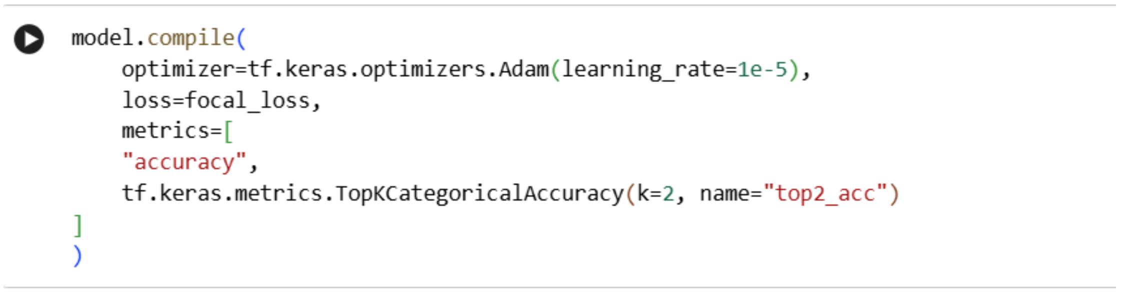


```
import tensorflow as tf

def categorical_focal_loss(alpha=0.25, gamma=2.0):
    def loss(y_true, y_pred):
        y_pred = tf.clip_by_value(y_pred, 1e-7, 1.0 - 1e-7)

        cross_entropy = -y_true * tf.math.log(y_pred)
        weight = alpha * tf.pow(1 - y_pred, gamma)

        loss = weight * cross_entropy
        return tf.reduce_sum(loss, axis=1)
    return loss
```

```
focal_loss = categorical_focal_loss(alpha=0.25, gamma=2.0)
```

- **α (Alpha):** The parameter α controls the overall weight of the focal loss and determines how strongly the model is penalized for misclassified samples.
- **γ (Gamma):** The parameter γ controls the focusing effect of the loss function by reducing the contribution of easy, well-classified samples and emphasizing harder examples.
- Categorical Focal Loss was employed to address class imbalance and improve learning from hard-to-classify samples. Unlike standard categorical cross-entropy, which is dominated by easy examples, focal loss reduces the contribution of well-classified samples and places greater emphasis on misclassified and difficult instances, leading to more robust and balanced model training.

4. Optimization Strategy:



```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss=focal_loss,
    metrics=[
    "accuracy",
    tf.keras.metrics.TopKCategoricalAccuracy(k=2, name="top2_acc")
]
)
```

The Adam optimizer is employed due to its adaptive learning rate and efficient convergence. A low learning rate of 1×10^{-5} is selected to enable stable fine-tuning without significantly altering the pre-trained weights.

5. Training Strategy:

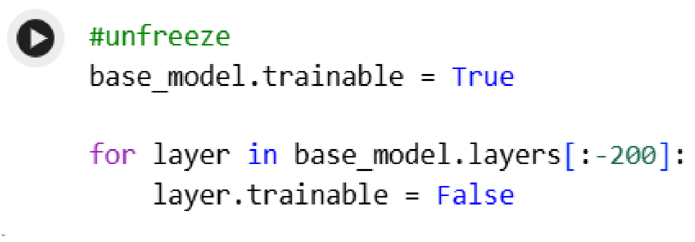
5.1 Stage 1 – Feature Extraction

- EfficientNetB3 backbone **fully frozen**
- Only classification head is trained
- Prevents overfitting on small datasets
- Stabilizes early training

5.2 Stage 2 – Fine-Tuning

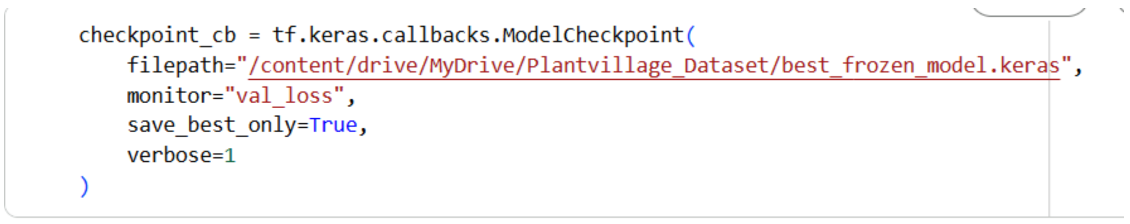
- Last **200 layers** of EfficientNetB3 unfrozen
- Earlier layers remain frozen

- Lower learning rate applied
- Enables task-specific feature refinement

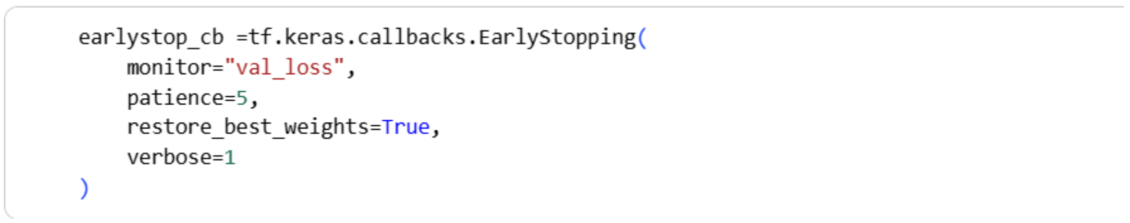


```
#unfreeze
base_model.trainable = True

for layer in base_model.layers[:-200]:
    layer.trainable = False
```

6. Regularization Techniques:



```
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint(
    filepath="/content/drive/MyDrive/Plantvillage_Dataset/best_frozen_model.keras",
    monitor="val_loss",
    save_best_only=True,
    verbose=1
)
```

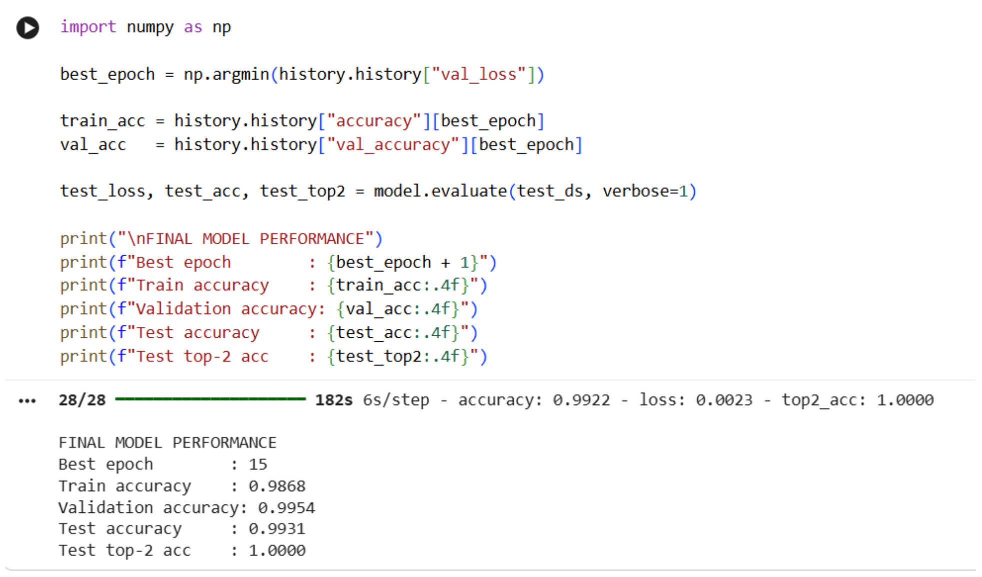


```
earlystop_cb =tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True,
    verbose=1
)
```

Early stopping is applied to prevent overfitting by halting training when validation loss no longer improves, while model **checkpointing** saves the best-performing model based on validation loss.

7. Model accuracy:

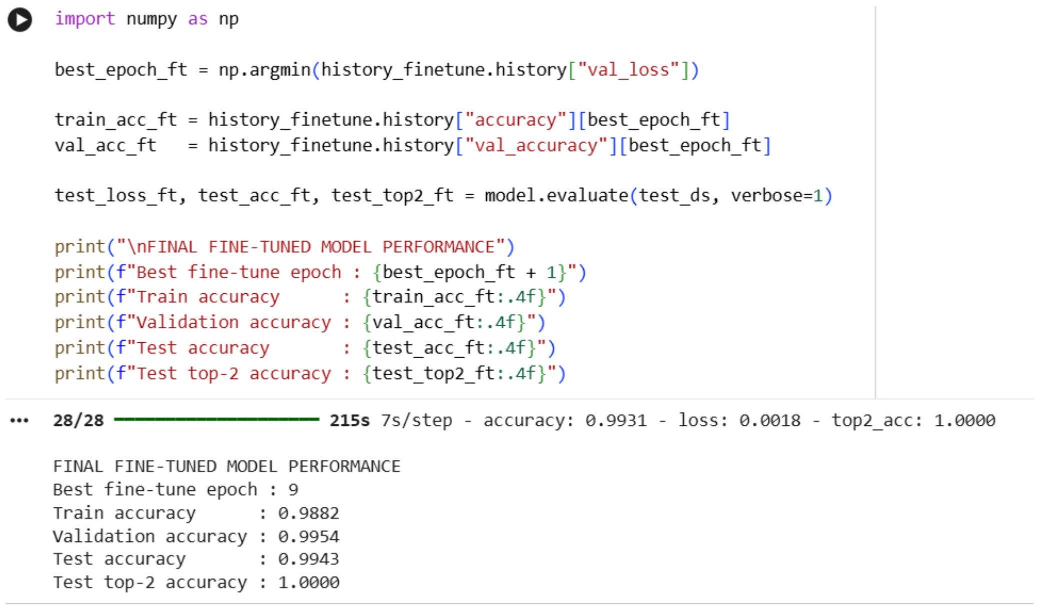
Frozen – training:



```
import numpy as np

best_epoch = np.argmin(history.history["val_loss"])

train_acc = history.history["accuracy"][best_epoch]
val_acc   = history.history["val_accuracy"][best_epoch]

test_loss, test_acc, test_top2 = model.evaluate(test_ds, verbose=1)

print("\nFINAL MODEL PERFORMANCE")
print(f"Best epoch        : {best_epoch + 1}")
print(f"Train accuracy    : {train_acc:.4f}")
print(f"Validation accuracy: {val_acc:.4f}")
print(f"Test accuracy     : {test_acc:.4f}")
print(f"Test top-2 acc    : {test_top2:.4f}")
```

```
28/28 ━━━━━━━━━━━━━━━━━━━━ 182s 6s/step - accuracy: 0.9922 - loss: 0.0023 - top2_acc: 1.0000

FINAL MODEL PERFORMANCE
Best epoch        : 15
Train accuracy    : 0.9868
Validation accuracy: 0.9954
Test accuracy     : 0.9931
Test top-2 acc    : 1.0000
```

Fine-tuning-training:



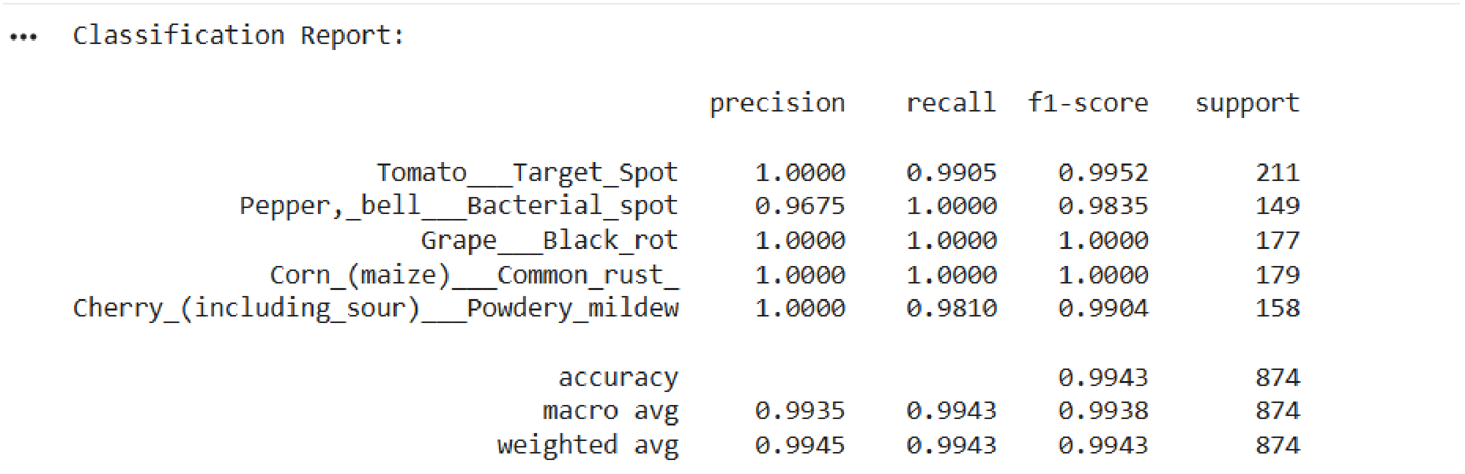
```
import numpy as np

best_epoch_ft = np.argmin(history_finetune.history["val_loss"])

train_acc_ft = history_finetune.history["accuracy"][best_epoch_ft]
val_acc_ft   = history_finetune.history["val_accuracy"][best_epoch_ft]

test_loss_ft, test_acc_ft, test_top2_ft = model.evaluate(test_ds, verbose=1)

print("\nFINAL FINE-TUNED MODEL PERFORMANCE")
print(f"Best fine-tune epoch : {best_epoch_ft + 1}")
print(f"Train accuracy      : {train_acc_ft:.4f}")
print(f"Validation accuracy : {val_acc_ft:.4f}")
print(f"Test accuracy       : {test_acc_ft:.4f}")
print(f"Test top-2 accuracy : {test_top2_ft:.4f}")
```

```
28/28 ━━━━━━━━━━━━━━━━━━━━ 215s 7s/step - accuracy: 0.9931 - loss: 0.0018 - top2_acc: 1.0000

FINAL FINE-TUNED MODEL PERFORMANCE
Best fine-tune epoch : 9
Train accuracy      : 0.9882
Validation accuracy : 0.9954
Test accuracy       : 0.9943
Test top-2 accuracy : 1.0000
```

8.Evaluation:

-classification report:

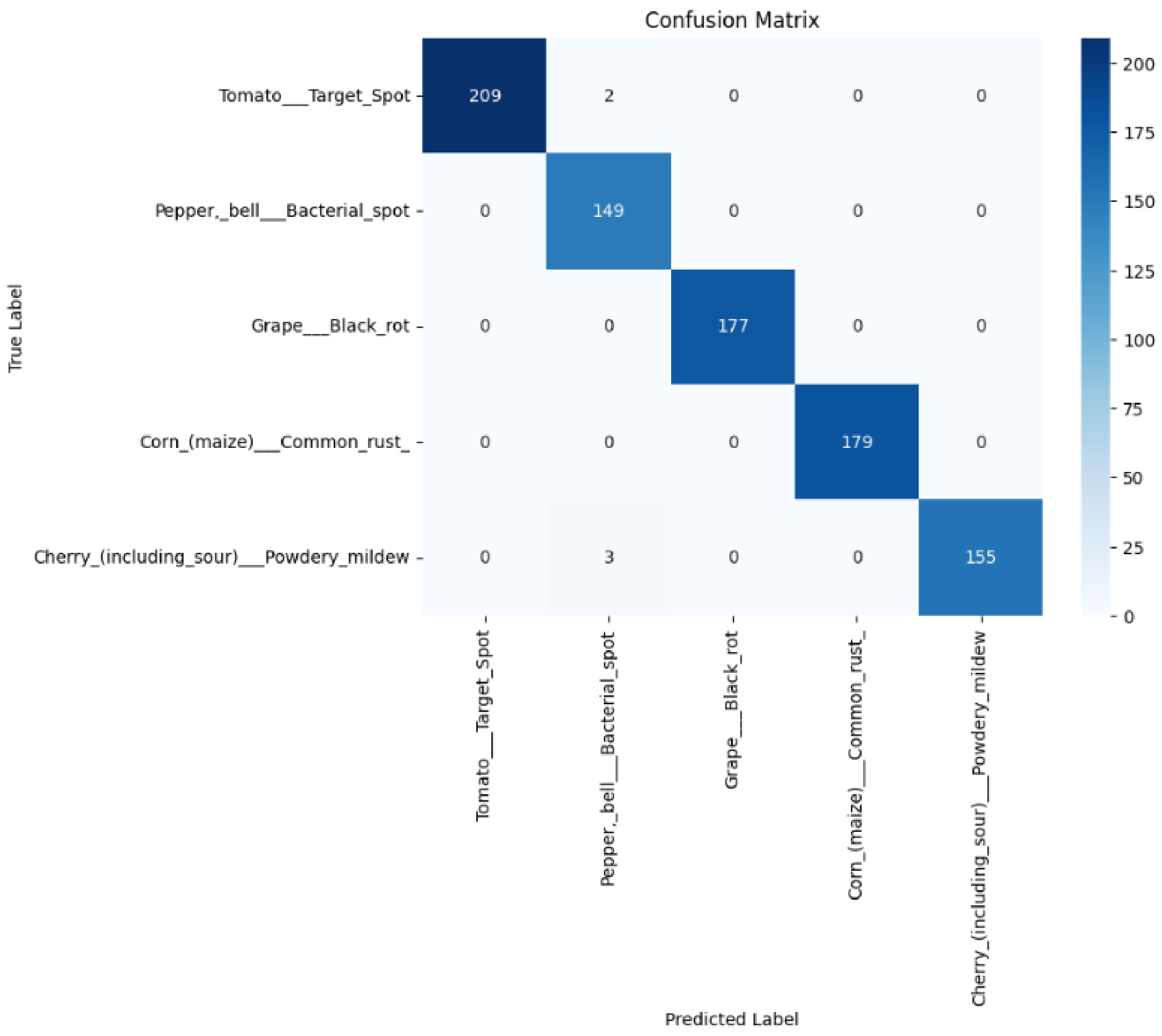
The classification report shows consistently high precision, recall, and F1-scores across all classes, with an overall accuracy of 99.43%, demonstrating robust and well-balanced classification performance.



```
Classification Report:

                                           precision    recall  f1-score   support

                       Tomato___Target_Spot     1.0000    0.9905    0.9952       211
               Pepper,_bell___Bacterial_spot     0.9675    1.0000    0.9835       149
                         Grape___Black_rot     1.0000    1.0000    1.0000       177
                  Corn_(maize)___Common_rust_     1.0000    1.0000    1.0000       179
    Cherry_(including_sour)___Powdery_mildew     1.0000    0.9810    0.9904       158

                                  accuracy                         0.9943       874
                                 macro avg     0.9935    0.9943    0.9938       874
                              weighted avg     0.9945    0.9943    0.9943       874
```

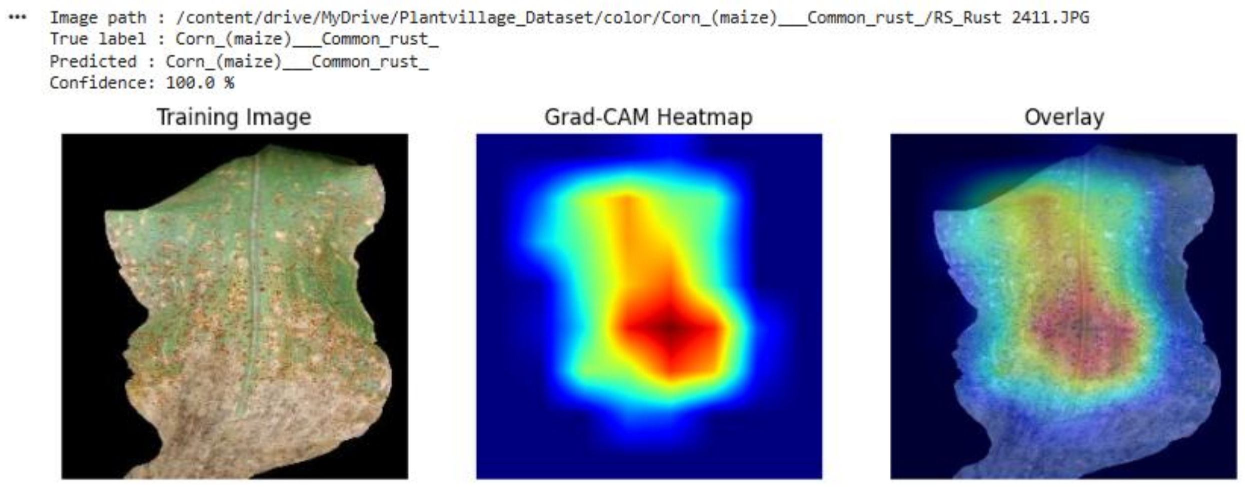
-confusion matrix:

The confusion matrix shows that most predictions lie along the diagonal, indicating high classification accuracy with only minor misclassifications between visually similar classes.



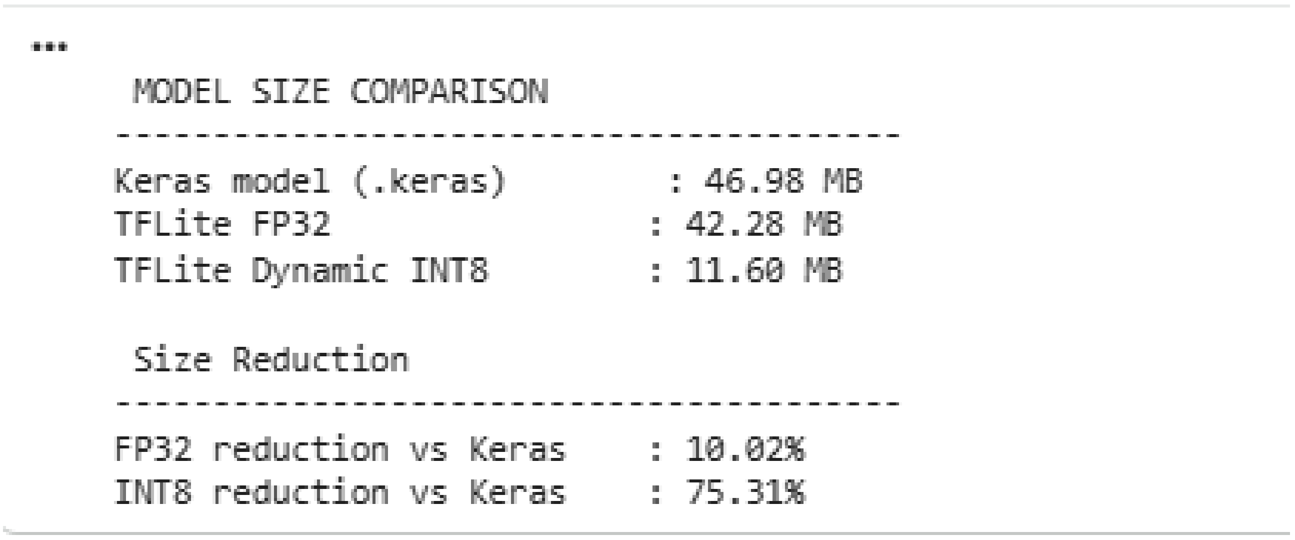
-CRAD-CAM:

The Grad-CAM visualization highlights disease-affected regions of the leaf, confirming that the model bases its prediction on relevant pathological features



```
Image path : /content/drive/MyDrive/Plantvillage_Dataset/color/Corn_(maize)___Common_rust_/RS_Rust 2411.JPG
True label : Corn_(maize)___Common_rust_
Predicted : Corn_(maize)___Common_rust_
Confidence: 100.0 %
```

9.TF-lite:-



```
...
      MODEL SIZE COMPARISON
     ----------------------------------------
     Keras model (.keras)         : 46.98 MB
     TFLite FP32                 : 42.28 MB
     TFLite Dynamic INT8         : 11.60 MB

      Size Reduction
     ----------------------------------------
     FP32 reduction vs Keras     : 10.02%
     INT8 reduction vs Keras     : 75.31%
```

Model compression significantly reduces storage requirements for deployment. While the original Keras model occupies 46.98 MB, conversion to TensorFlow Lite FP32 reduces the size by 10.02%. Applying TensorFlow Lite dynamic INT8 quantization further compresses the model to 11.60 MB, achieving a 75.31% size reduction and making it suitable for edge and mobile applications with limited resources.

Why using efficientnetb3 ?

- EfficientNet-B3 is well suited for this task due to its ability to capture both low-level and high-level features effectively.
- The EfficientNet architecture employs **compound scaling**, which uniformly scales network depth, width, and input resolution. This results in improved feature representation while using fewer parameters compared to traditional convolutional neural networks
- EfficientNet-B3 offers higher representational power than smaller variants (e.g., B0–B2) while remaining more computationally efficient than larger variants (e.g., B4–B7), making it ideal for medium-sized agricultural datasets.