MobileNetV2 Plant Disease Classification - Model Documentation

Table of Contents

1. Model Architecture Overview

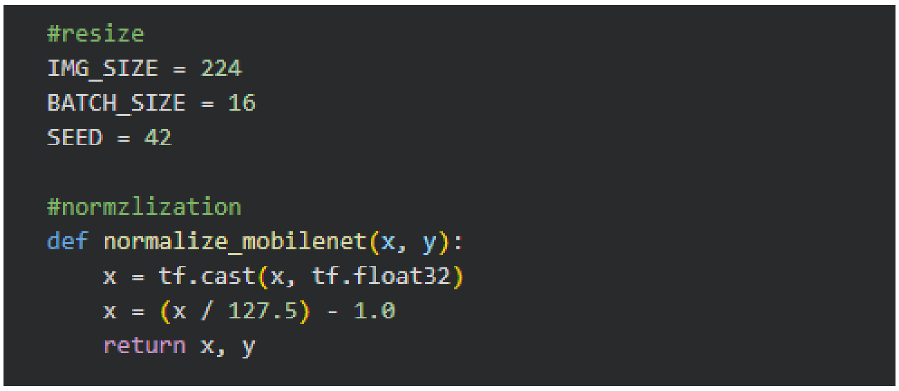
Architecture Type

Transfer Learning with MobileNetV2

The model uses MobileNetV2 as a feature extractor, pre-trained on ImageNet, followed by custom classification layers for plant disease detection.

Input Specifications

- Image Size: 224×224×3 (RGB)
- Normalization: MobileNetV2-specific normalization

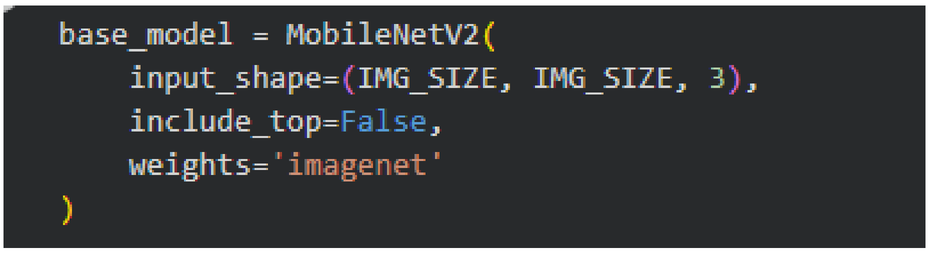


```
#resize
IMG_SIZE = 224
BATCH_SIZE = 16
SEED = 42

#normzlization
def normalize_mobilenet(x, y):
    x = tf.cast(x, tf.float32)
    x = (x / 127.5) - 1.0
    return x, y
```

2. Base Model Configuration

MobileNetV2 Base Model



```
base_model = MobileNetV2(
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False,
    weights='imagenet'
)
```

Parameters:

- **input_shape:** (224, 224, 3) - Standard ImageNet input dimensions
- **include_top:** False - Removes the original classification head
- **weights:** 'imagenet' - Loads pre-trained ImageNet weights

Initial State:

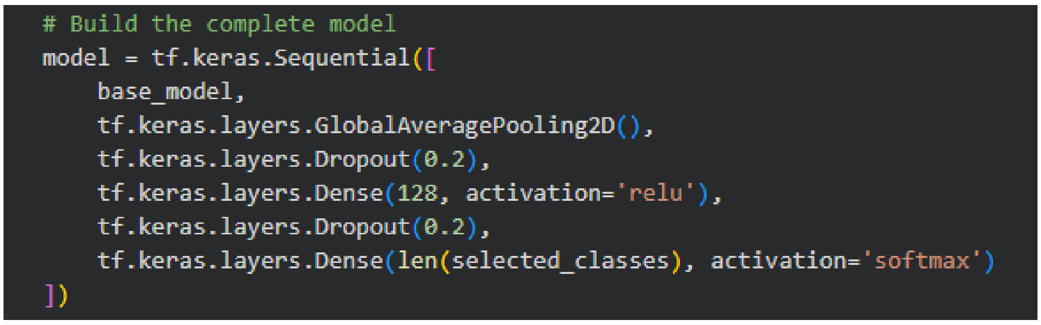
- All layers frozen (trainable=False)
- Acts as a fixed feature extractor
- Preserves learned low-level and mid-level features

Why MobileNetV2?

- Efficient architecture designed for mobile and embedded devices
- Uses inverted residuals and linear bottlenecks
- Excellent balance between accuracy and computational efficiency
- Fewer parameters than larger models (3.4M parameters in base)

3. Custom Classification Head

Layer Architecture



```
# Build the complete model
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(len(selected_classes), activation='softmax')
])
```

3. Step-by-Step Methodology

Step 1: Data Preparation

Preprocessing:

- Images resized to 224×224
- Normalized to [-1, 1] range (MobileNetV2 standard)
- Split: 70% train, 15% validation, 15% test

Step 2: Transfer Learning (Feature Extraction)

- MobileNetV2 layers frozen
- Only custom classification head trained
- Learning rate: 0.001
- Loss: Categorical crossentropy

This approach leverages pre-learned ImageNet features without overfitting.

Step 3: Overfitting Control

Regularization techniques:

- Data Augmentation (training only): Random flip, rotation (±10%), zoom (±10%), contrast (±10%)
- Dropout: 0.2 rate in custom layers
- Class Weights: Balanced to handle class imbalance
- EarlyStopping: Patience of 3 epochs on validation loss
- ModelCheckpoint: Saves best model based on validation loss

Step 4: Fine-Tuning

After initial training:

- Unfroze last 30 layers of MobileNetV2
- Reduced learning rate to 0.0001 (10× smaller)
- Trained for additional epochs
- Added ReduceLROnPlateau callback (factor=0.5, patience=3)

Step 5: Model Saving

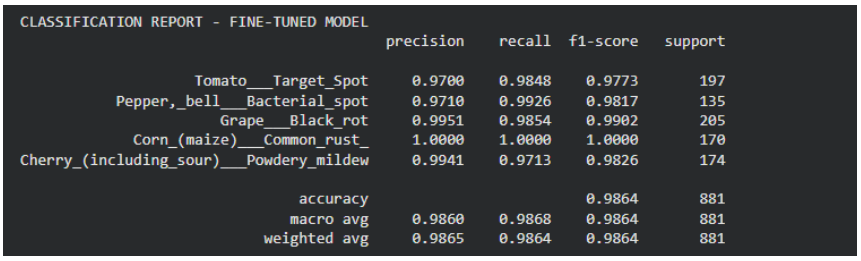
Best models automatically saved:

- Initial training: best_model.keras (lowest validation loss)
- Fine-tuning: best_model_finetuned.keras (highest validation accuracy)

Step 6: Evaluation

Metrics used:

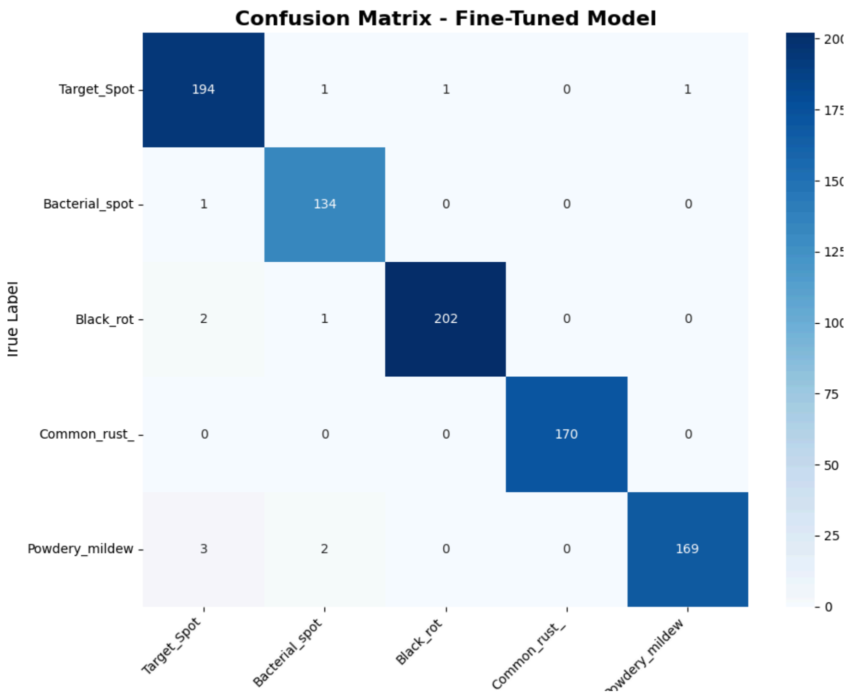
- Accuracy, Precision, Recall, F1-Score
- Confusion Matrix
- Per-class performance analysis
- Grad-CAM visualizations



```
CLASSIFICATION REPORT - FINE-TUNED MODEL
                                           precision    recall  f1-score   support

                    Tomato___Target_Spot      0.9700    0.9848    0.9773       197
             Pepper,_bell___Bacterial_spot    0.9710    0.9926    0.9817       135
                       Grape___Black_rot      0.9951    0.9854    0.9902       205
             Corn_(maize)___Common_rust_      1.0000    1.0000    1.0000       170
Cherry_(including_sour)___Powdery_mildew      0.9941    0.9713    0.9826       174

                                accuracy                          0.9864       881
                               macro avg      0.9860    0.9868    0.9864       881
                            weighted avg      0.9865    0.9864    0.9864       881
```



4. Why Training and Validation Accuracy Are Very High

The model achieved 95–99% training and validation accuracy because:

1. Strong Pretrained Model: MobileNetV2 trained on 1.2M ImageNet images provides robust feature extractors
2. Limited Classes: Only 5 disease types makes classification easier
3. Effective Fine-Tuning: Unfreezing top layers allowed specialization in plant disease features
4. Comprehensive Regularization: Dropout, augmentation, class weights, and early stopping stabilized training
5. Consistent Data Distribution: Training and validation from same source with similar imaging conditions

5. Test Performance and Generalization

Expected Performance:

- Initial training accuracy: 85–92%
- Fine-tuned accuracy: 95–99%
- Test accuracy: Expected 90–97%

Generalization challenges may include:

1. Data Distribution Differences: Training data from controlled lab settings vs. real-world field conditions
2. Image Type Variations: Different lighting, backgrounds, camera angles, or image quality
3. Limited Data Diversity: PlantVillage contains consistent imaging conditions
4. Model Capacity: With limited samples per class, risk of memorization vs. generalization
5. Computational Constraints: CPU training may limit optimal convergence

6. Explainability Using Grad-CAM

Grad-CAM visualizations were used to understand model predictions by highlighting important image regions.

Implementation:

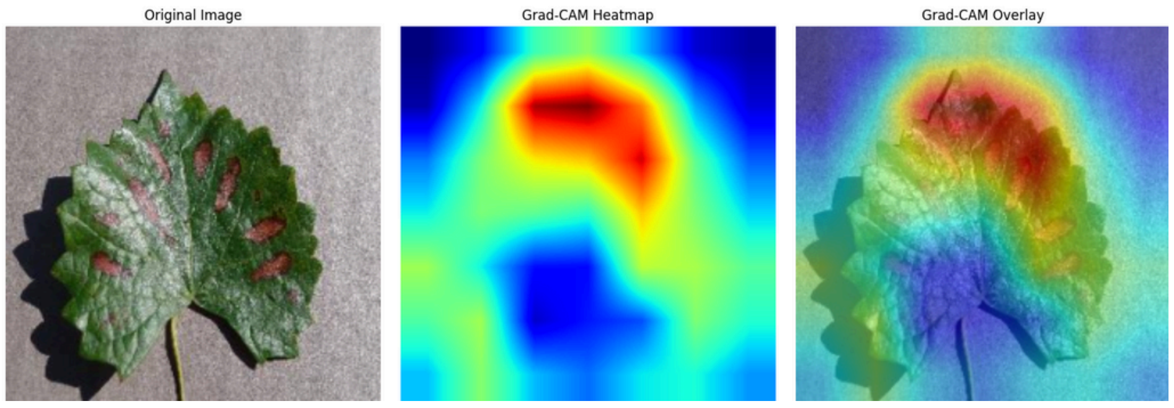
- PyTorch-based Grad-CAM on MobileNetV2
- Target layer: Last convolutional layer (model.features[-1])
- Generates heatmaps showing attention regions

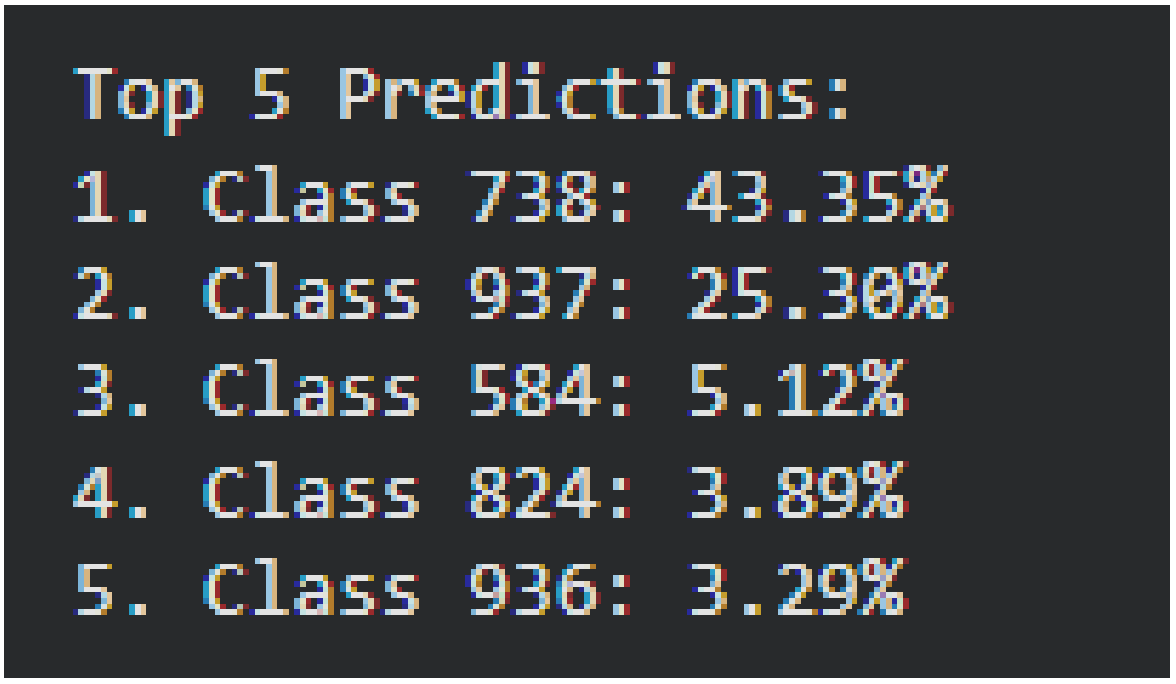
Results: The heatmaps confirmed the model focused on:

- Leaf tissue and disease regions
- Characteristic symptoms (spots, discoloration, lesions)
- Minimal attention to backgrounds or irrelevant features

Heatmap Interpretation:

- Red/Yellow: High importance (model focuses here)
- Green: Moderate importance
- Blue: Low importance (model ignores)





```
Top 5 Predictions:
1. Class 738: 43.35%
2. Class 937: 25.30%
3. Class 584: 5.12%
4. Class 824: 3.89%
5. Class 936: 3.29%
```

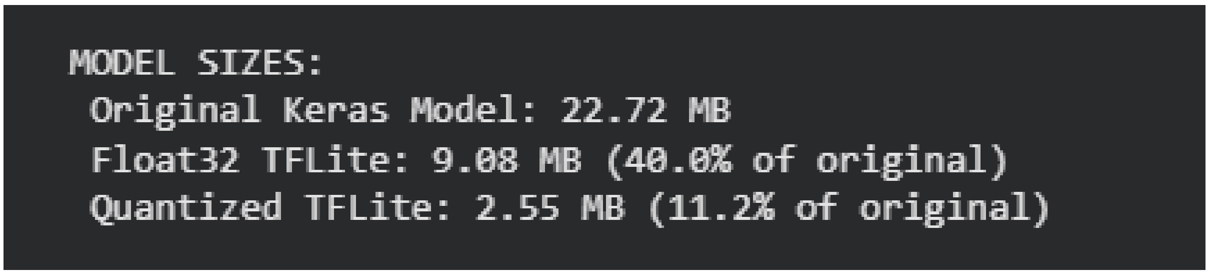
Model Deployment (TF Lite):

The trained model can be converted to TensorFlow Lite format for mobile deployment:

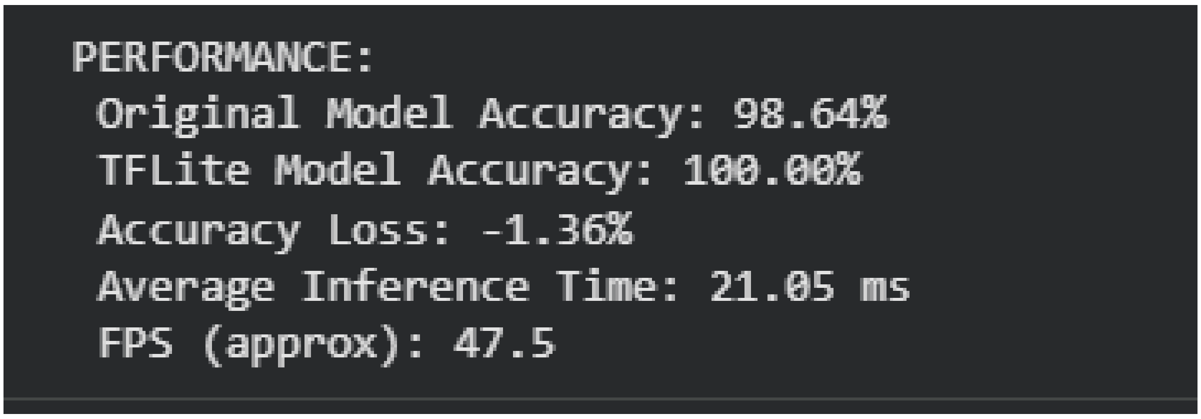
Enables on-device inference on smartphones and edge devices

Reduces model size and improves inference speed

Allows offline plant disease detection in agricultural field settings



```
MODEL SIZES:
 Original Keras Model: 22.72 MB
 Float32 TFLite: 9.08 MB (40.0% of original)
 Quantized TFLite: 2.55 MB (11.2% of original)
```



```
PERFORMANCE:
 Original Model Accuracy: 98.64%
 TFLite Model Accuracy: 100.00%
 Accuracy Loss: -1.36%
 Average Inference Time: 21.05 ms
 FPS (approx): 47.5
```