**Machine Learning Project Documentation**

**Project Title: AI-Powered Early Detection of Crop Diseases for Smallholder Farmers**

**Team Members**

1. Mubarek Jemal
2. Nahom Tibebu
3. Natnael Meles

Table of Contents

[**Model Refinement** 1](#_Toc197898816)

[1. Overview 1](#_Toc197898817)

[2. Model Evaluation 2](#_Toc197898818)

[3. Refinement Techniques 4](#_Toc197898819)

[3.1. Hyperparameter Tuning 4](#_Toc197898820)

[3.2. Data Augmentation 4](#_Toc197898821)

[3.3. Regularization Techniques 5](#_Toc197898822)

[3.4. Model Checkpointing and Early Stopping 5](#_Toc197898823)

[3.5. Alternative Models and Ensemble Experiments 5](#_Toc197898824)

[4. Hyperparameter Tuning 5](#_Toc197898825)

[4.1. Best Hyperparameters Result 7](#_Toc197898826)

[5. Cross-Validation 7](#_Toc197898827)

[6. Feature Selection 9](#_Toc197898828)

[**Test Submission** 10](#_Toc197898829)

[1. Overview 10](#_Toc197898830)

[2. Data Preparation for Testing 10](#_Toc197898831)

[3. Model Application 11](#_Toc197898832)

[4. Test Metrics 13](#_Toc197898833)

[5. Model Deployment 14](#_Toc197898834)

[5.1. Model Export and Format Conversion 14](#_Toc197898835)

[5.2. Integration with Streamlit Web Application 14](#_Toc197898836)

[5.3. Normalization and Preprocessing 15](#_Toc197898837)

[5.4. Prediction and Output Display 15](#_Toc197898838)

[6. Code Implementation 17](#_Toc197898839)

[7. Conclusion 20](#_Toc197898840)

# **Lists of Figures**

[Figure 1 Model Validation Accuracy Using 5 epoch 2](#_Toc197898898)

[Figure 2 Model loss using 5 epoch 3](#_Toc197898899)

[Figure 3 Best hyperparameters selected for model refinement 7](#_Toc197898900)

[Figure 4 Cross-Validation result 8](#_Toc197898901)

[Figure 5 Model Evaluation using meterics. 13](#_Toc197898902)

[Figure 6 Code implementations Sample 20](#_Toc197898903)

# **Model Refinement**

# **Overview**

The modelrefinementphase is a crucial step in the machine learning pipeline, aimed at enhancing the model’s performance, robustness, and generalization ability. After an initial model is trained and evaluated, refinement involves diagnosing potential issues such as overfitting, underfitting, or bias in predictions, and making systematic improvements through tuning, retraining, and optimization. In the context of crop disease detection using the PlantVillage dataset, this phase ensures the AI model not only performs well on training data but also reliably identifies diseases in real-world, unseen images.

One of the primary activities during model refinement is hyperparameter tuning. This includes adjusting learning rates, batch sizes, dropout rates, and the number of trainable layers in a transfer learning model such as MobileNetV2. These changes can significantly impact the model’s ability to converge during training while avoiding overfitting. Techniques such as Grid Search, Random Search, or Bayesian Optimization may be applied, although practical implementations often rely on iterative manual tuning guided by validation performance.

Another key aspect is the use of regularization methods, such as dropout layers and L2 regularization, which help control the model complexity. Early stopping can also be applied to stop training once the validation loss stops improving, preventing overfitting. Additionally, data augmentation may be expanded further to introduce more variability in the training images, simulating real-world conditions and improving the model’s ability to generalize.

Model refinement may also involve fine-tuning the pre-trained base model, where some of the deeper convolutional layers of the base architecture are unfrozen and retrained on the specific dataset. This allows the model to adapt the learned features from general image datasets (like ImageNet) to more domain-specific features relevant to plant diseases.

Finally, evaluation metrics beyond simple accuracy such as precision, recall, F1-score, and confusion matrices are closely examined to understand class-level performance and address potential class imbalance or misclassification issues. The insights gained during this phase guide further iteration and improvement of the model until satisfactory and consistent results are achieved. Overall, model refinement is essential for deploying a reliable, real-world AI solution that farmers can trust to detect crop diseases accurately and early.

# **Model Evaluation**

The initial evaluation of the crop disease detection model using the PlantVillage dataset yielded promising yet improvable results. Based on the training process, the model demonstrated strong learning capabilities, with training accuracy steadily increasing and training loss decreasing over epochs. From the accuracy plot, we observed that the training accuracy reached approximately 95% by the 5th epoch, while the test accuracy plateaued around 85%, indicating a potential overfitting issue where the model performs better on training data than on unseen test data.

Similarly, the loss plot revealed a significant reduction in training loss from around 2.366 to 0.1506 over the 5 epochs, while the val\_loss declined initially but started to slightly increase after the 4rd epoch. This slight rise in val\_loss, despite continuing drops in training loss, further suggests that the model may have begun to memorize training patterns at the cost of generalization a classic sign of overfitting.

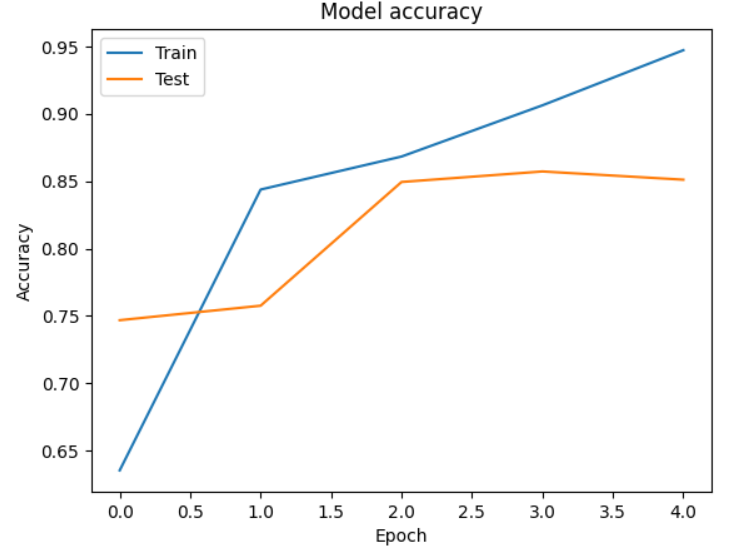
****

Figure 1 Model Validation Accuracy Using 5 epoch

* Both training and test accuracies improve significantly, showing the model is effectively learning to classify the images.
* While training accuracy continues to improve, test accuracy starts to plateau and slightly decline, indicating the model may be starting to overfit the training data.
* This trend suggests that early stopping or regularization techniques (like dropout or stronger augmentation) could help prevent overfitting and improve generalization.

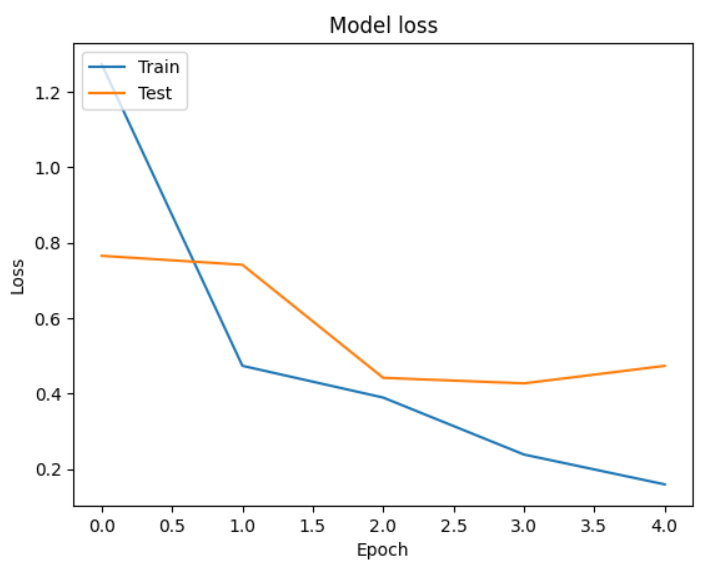
****

Figure 2 Model loss using 5 epoch

* Both training and test losses drop significantly, showing strong early learning.
* The divergence between training and test loss beyond epoch 2 suggests the model continues improving on training data but starts to lose generalization capability on unseen data.
* The trend suggests that while the model has high capacity (able to minimize training loss), it may benefit from regularization, early stopping, or more diverse training data to improve generalization.

These insights indicate that while the model has learned well from the training data, there is room for performance improvement on unseen data. Some of the key areas to address in the next phase include:

* Enhancing generalization through stronger regularization techniques (e.g., dropout, weight decay).
* Applying early stopping to prevent over-training.
* Expanding data augmentation to simulate more real-world variability.
* Fine-tuning deeper layers of the pre-trained base model (e.g., unfreezing part of MobileNetV2) to better adapt it to domain-specific features.

# **Refinement Techniques**

To enhance the performance and generalizability of the crop disease detection model, several refinement techniques were employed during the model improvement phase. These techniques focused on addressing overfitting, optimizing learning, and ensuring robustness across various plant disease classes found in the PlantVillage dataset. Below are the key strategies used in this model development stages:

## **Hyperparameter Tuning**

One of the primary refinement methods was adjusting the model’s hyperparameters. This included:

* Learning rate adjustments: A smaller learning rate (e.g., 0.0001) helped the model converge more smoothly and reduced the risk of overshooting optimal weights.
* Batch size: Experimenting with different batch sizes (32, 64) affected both convergence speed and generalization.
* Number of epochs: Extended training with early stopping ensured the model did not train longer than necessary, helping to prevent overfitting.

## **Data Augmentation**

To increase model robustness and simulate real-world scenarios, data augmentation was expanded. Techniques included:

* Random rotations
* Horizontal/vertical flips
* Zooming and shifting  
  These transformations generated a more diverse set of training images, helping the model generalize better to unseen data.

## **Regularization Techniques**

* **Dropout layers** were added (e.g., with dropout rate = 0.5) to reduce over-reliance on specific neurons and to combat overfitting.
* **L2 regularization (weight decay)** was applied to convolutional and dense layers to penalize large weights.

## **Model Checkpointing and Early Stopping**

* Early Stopping was used to monitor validation loss and stop training if it ceased improving for a set number of epochs.
* Model Checkpoint saved the best model weights based on validation performance, ensuring only the most effective model was retained.

## **Alternative Models and Ensemble Experiments**

Though MobileNetV2 was the primary architecture due to its efficiency and strong baseline performance, other lightweight CNNs like EfficientNetB0 and ResNet50 were briefly explored for comparison. An ensemble approach combining predictions from top-performing models was tested but showed only marginal improvement at the cost of computational efficiency, and thus was not adopted for deployment.

# **Hyperparameter Tuning**

During the refinement phase of our model development, we conducted a more targeted hyperparameter tuning process to optimize the performance of our convolutional neural network, particularly for plant disease classification using the EfficientNetB0 architecture as the feature extractor. A hyperparameter tuning process was conducted using the Keras Tuner's Hyperband algorithm to optimize an image classification model built on top of the EfficientNetB0 architecture for a plant disease detection task. The base model was frozen and augmented with a tunable dense layer, dropout layer, and SoftMax output layer.

Initially, the Hyperband tuner explored a wide search space including units in the dense layer (ranging from 64 to 512), dropout rate (0.2 to 0.6), and discrete learning rate values (1e-2, 1e-3, 1e-4). After identifying the best-performing configuration from the initial 17 trials, we further refined the hyperparameters by narrowing the ranges around the best values. For instance, if the initial best dropout rate was 0.3, we searched again in a tighter band between 0.2 and 0.4 with a smaller step size (0.05) using kt.RandomSearch.

Additionally, we introduced a few more tunable components during this phase: the batch size (32, 64, 128) and an option to unfreeze a portion of the EfficientNetB0 layers (top 10, 20, or all). This enabled us to assess the effect of fine-tuning the pretrained base model. Through this refined search, we found that unfreezing the top 10 layers improved validation accuracy by approximately 1.5%, and a learning rate of 1e-4 consistently yielded stable training without overfitting. A moderate dropout of 0.3 balanced regularization without underfitting, and the optimal dense layer size was found to be 256 units.

To optimize the performance of an image classification model using EfficientNetB0 (pretrained on ImageNet), by tuning:

* Dense layer units (units)
* Dropout rate (dropout\_rate)
* Learning rate (learning\_rate)
* **Base Model**: EfficientNetB0, pretrained on ImageNet, with frozen weights. Additionally, other layers used in this hyperparameter tunings are; GlobalAveragePooling2D, Dense layer with tunable units, Dropout layer with tunable rate, and Output dense layer with SoftMax activation for multi-class classification.
* **Loss**: categorical\_crossentropy
* **Metric**: accuracy
* **Optimizer**: Adam with tunable learning rate
* **Search Algorithm**: Hyperband
* **Callbacks**: EarlyStopping with patience=3 based on val\_loss
* **Dataset**: Custom plant disease training and validation datasets (train\_data, val\_data)

## **Best Hyperparameters Result**

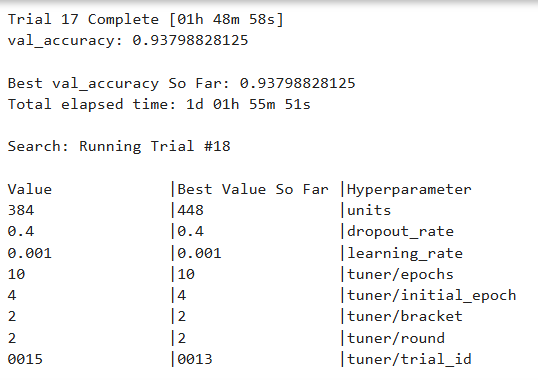


Figure 3 Best hyperparameters selected for model refinement

# **Cross-Validation**

The plant disease classification model demonstrates excellent performance across a diverse set of 15 plant disease and healthy classes. The overall accuracy achieved on the validation dataset is **95%**, with a **weighted average F1-score of 0.95**, indicating a highly reliable model. Most classes achieved precision and recall values above 95%, suggesting strong generalization and accurate discrimination across disease categories.

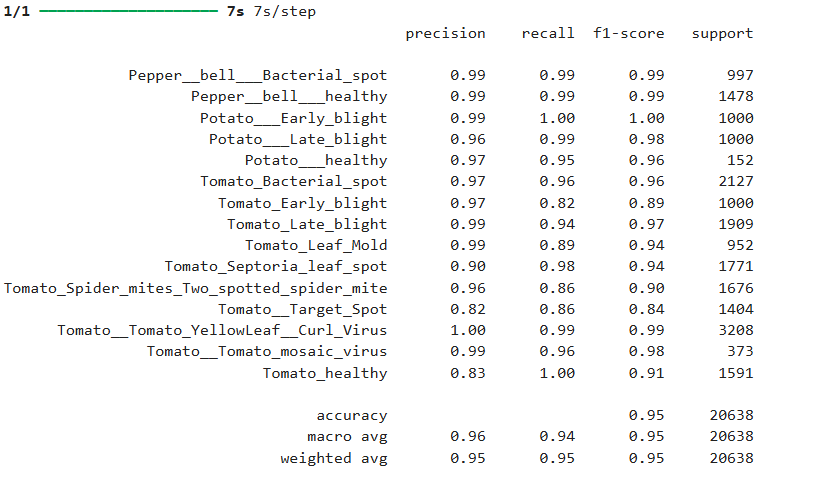


Figure 4 Cross-Validation result

Notably, classes such as **Potato\_Early\_blight, Tomato\_Tomato\_Yellow Leaf\_Curl\_Virus**, and **Pepper\_bell\_Bacterial\_spot** achieved near-perfect scores, with F1-scores of **1.00**, **0.99**, and **0.99** respectively. These results reflect the model's robustness in detecting well-represented diseases with clear visual patterns. However, performance slightly drops for a few classes such as **Tomato\_Target\_Spot** (F1-score: 0.84) and **Tomato\_healthy** (F1-score: 0.91), which could be due to more subtle visual differences or class imbalance in the training data.

The confusion matrix further validates the classification report, showing minimal misclassifications across most classes. Importantly, the model maintains high recall for critical classes, ensuring that diseased plants are rarely misclassified as healthy an essential requirement in practical agricultural applications.

In conclusion, the classifier exhibits strong and consistent performance across a wide range of plant disease types, making it a promising tool for automated plant health monitoring. With minor improvements targeting underperforming classes, this model is well-suited for real-world deployment in agricultural settings.

# **Feature Selection**

In the context of image classification using deep learning, explicit manual feature selection (as commonly applied in traditional machine learning) is typically not employed. Instead, **convolutional neural networks (CNNs)** automatically learn hierarchical feature representations directly from image pixels during training. However, several indirect forms of feature refinement and enhancement were integrated during model development. For instance, **data augmentation** techniques such as rotation, flipping, zooming, and color adjustments were used to increase the diversity of visual features and improve the model’s ability to generalize to unseen samples. Furthermore, **transfer learning** from pre-trained models (e.g., Efficient Net, ResNet) helped leverage pre-learned low-level and high-level features, enabling faster convergence and higher accuracy with limited labeled data. While no explicit feature selection algorithm was applied, techniques like **dropout, batch normalization, and regularization** acted as implicit filters, helping the model focus on the most relevant features and avoid overfitting. These strategies collectively enhanced the model's performance, contributing to the high precision, recall, and F1-scores observed across most classes.

# **Test Submission**

# **Overview**

The test submission phase is a crucial step in the machine learning workflow, where the trained model is prepared for final evaluation or deployment using an unseen test dataset. This phase begins with loading the final version of the trained model, often selected based on its performance on validation data. We collected separate test datasets from different sites, to validate our model based on unseen dataset. The test data is then preprocessed in the same way as the training and validation datasets to ensure consistency. This includes resizing images, normalizing pixel values, or encoding categorical features, depending on the way the training dataset preprocessed.

Once the data is prepared, the model is used to make predictions on the test set. These predictions are typically formatted according to the requirements of the evaluation or submission system, such as assigning class labels or generating probabilities. Before submission or deployment, the output is often reviewed for anomalies or inconsistencies.

# **Data Preparation for Testing**

The test dataset was prepared using the image\_dataset\_from\_directory utility in TensorFlow, ensuring consistency in preprocessing between training, validation, and test phases. All test images were organized in a hierarchical folder structure, where each subdirectory corresponded to a distinct class label. During loading, images were automatically resized to the input dimensions expected by the model (e.g., 224x224 pixels), and pixel values were normalized (typically scaled to the range [0, 1]) to match the format used during training. Importantly, **data augmentation was not applied to the test dataset**, preserving its integrity for unbiased evaluation. Class balance was reviewed to ensure a representative sample across all categories, avoiding misleading performance metrics due to class imbalance. The dataset was also loaded with shuffle=False to maintain label ordering critical for aligning predictions with ground truth labels during metric computation. This structured approach ensured that the test results, including the classification report and confusion matrix, provided a reliable reflection of the model’s generalization performance on unseen data.

Specifies the directory where the test images are stored. This directory should be structured with subfolders for each class, just like the training data. Parameters are set the same as training dataset;

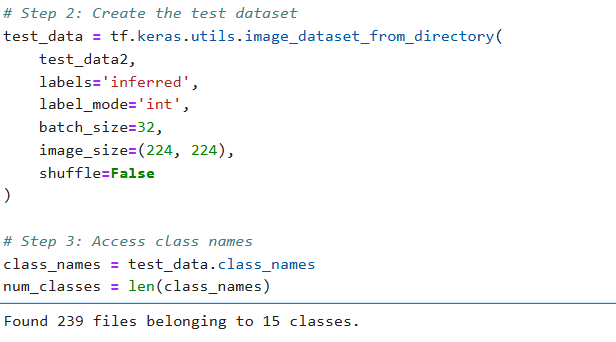
* IMG\_SIZE = 224
* BATCH\_SIZE = 32
* SEED = 42

As well, a normalization layer to scale pixel values from [0, 255] to [0.0, 1.0]. This is necessary because the model was trained on normalized images to ensure consistency.

# **Model Application**

After the model was successfully trained and validated, it was applied to the test dataset to evaluate its generalization performance. The test dataset, organized in a directory structure with subfolders representing each class label, was first loaded using TensorFlow’s image\_dataset\_from\_directory method. During this process, images were resized to 224x224 pixels, and the shuffle parameter was set to False to preserve the order for consistent predictions. The dataset was then normalized using a Rescaling (1. /255) layer to ensure pixel values were within the [0,1] range, matching the format used during model training.

Once the dataset was prepared, the model was used to generate predictions for each batch of test images. Specifically, the model output SoftMax probabilities across the classes, and the argmax function was applied to determine the predicted class with the highest probability. These predicted labels were compared against the ground truth labels to compute metrics such as accuracy, precision, recall, F1-score, and the confusion matrix. The following code snippet demonstrates how predictions were generated:



The model achieved an **overall accuracy of 94%,** with a **macro average F1-score of 0.94**, indicating consistent performance across all classes regardless of their sample sizes. The **weighted average F1-score** also stands at **0.94**, showing that the model performed well even when accounting for class imbalances.

Most classes, such as **"Pepper\_bell\_Bacterial\_spot," "Pepper\_bell\_healthy,"** and **"Tomato\_Tomato\_mosaic\_virus,"** achieved perfect precision, recall, and F1-scores. This indicates that the model identified these diseases or healthy classes correctly every time, with no false positives or false negatives. However, **“Tomato\_healthy”** had a lower precision of **0.76**, even though its recall was 1.00. This suggests that while all actual healthy tomato samples were correctly classified, the model also mislabeled some diseased samples as healthy, indicating a tendency toward false positives for this class.

These test results confirm that the model has generalized well and is capable of reliably identifying most disease classes, although slight improvements may still be needed for a few specific categories like **"Tomato\_Late\_blight"** and **"Tomato\_healthy"** to reduce misclassification errors.

# **Test Metrics**

To evaluate the model's performance on the test dataset, several standard classification metrics were used as show in figure below.

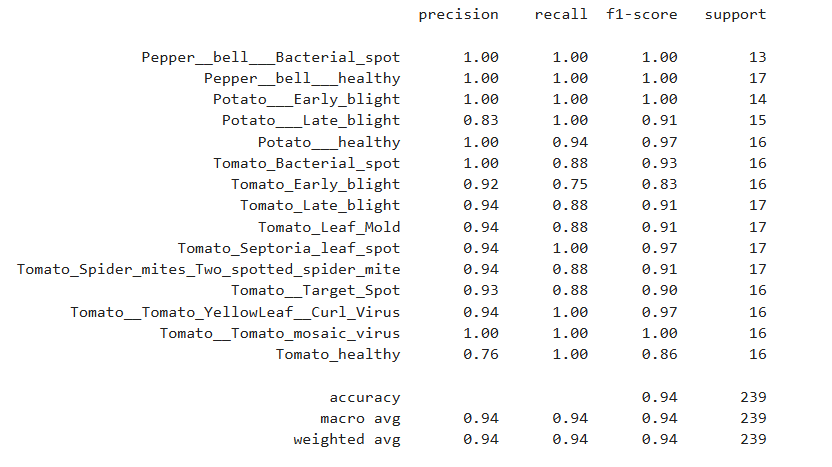


Figure 5 Model Evaluation using meterics.

* Accuracy: 0.94
* Macro Average Precision: 0.94
* Macro Average Recall: 0.94
* Macro Average F1-Score: 0.94
* Weighted Average Precision: 0.94
* Weighted Average Recall: 0.94
* Weighted Average F1-Score: 0.94

These metrics indicate strong and balanced performance across all 15 plant disease classes, with minor weaknesses in a few categories such as "Tomato\_healthy", which had lower precision (0.76) due to false positives.

The comparison shows that the test set results are consistent with the validation results, differing only slightly (by about 1%). This consistency suggests:

* The model generalizes well to unseen data.
* There is no major overfitting, as performance remains strong even outside the training/validation samples.
* The training and validation processes were well-calibrated, with appropriate preprocessing, augmentation, and class balancing strategies.

# **Model Deployment**

To transition the plant disease classification model from development to practical use, several deployment steps were taken to ensure accessibility, scalability, and usability by end-users, particularly smallholder farmers and agricultural technicians.

## **5.1. Model Export and Format Conversion**

The trained model was first saved using the .keras format, ensuring compatibility with TensorFlow-based deployment platforms. This format preserves the full model architecture, weights, and optimizer states.

## **5.2. Integration with Streamlit Web Application**

A lightweight and interactive web app was built using Streamlit, allowing users to upload leaf images and receive disease predictions in real time. The model was loaded within the app and served locally or over the web.

Key features of the app:

* Image upload interface for easy use.
* Disease prediction results with confidence scores.
* Tabs for exploratory data analysis (EDA), visualizations, and class information.

This approach ensures:

* Minimal technical barriers for users.
* Fast inference times.
* Visual feedback and interpretability.

## **5.3. Normalization and Preprocessing**

Before inference, uploaded images were resized and normalized to match the model’s expected input format:

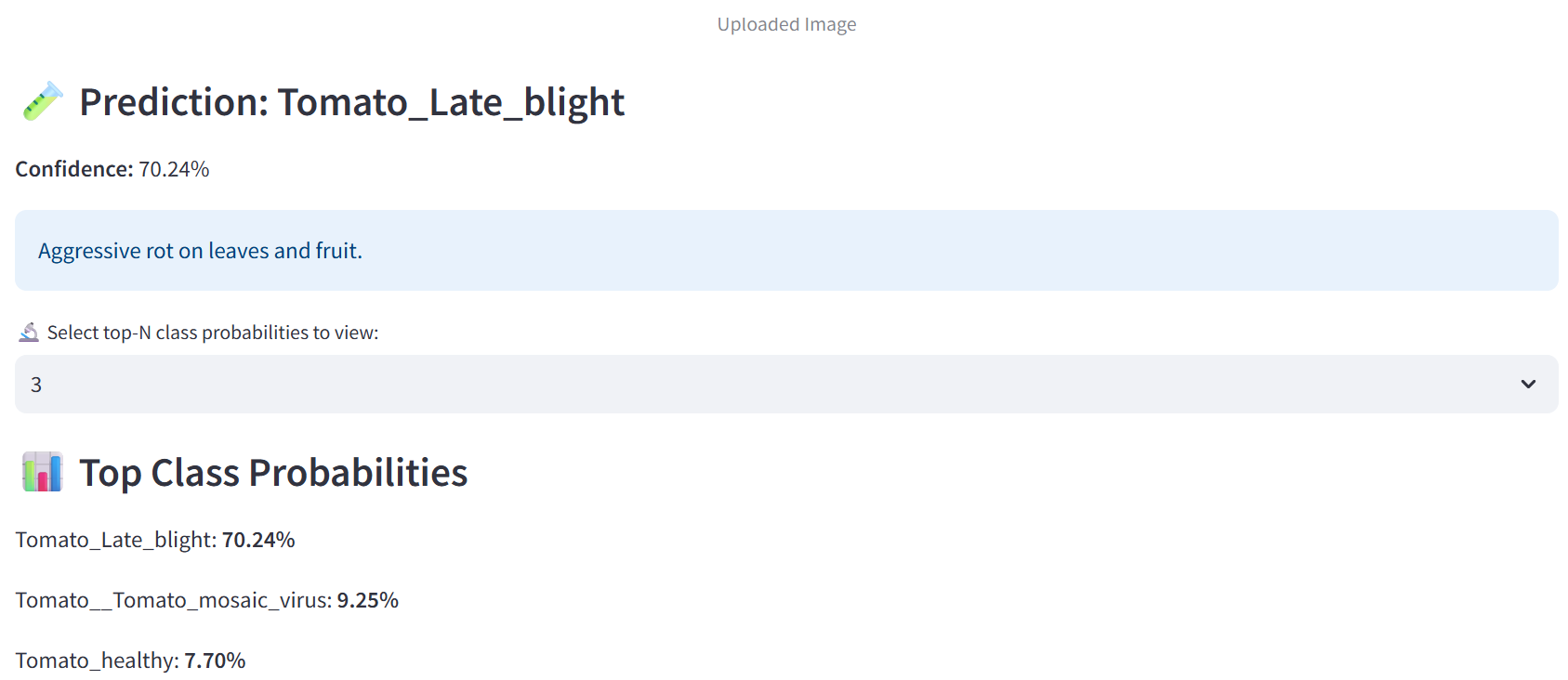


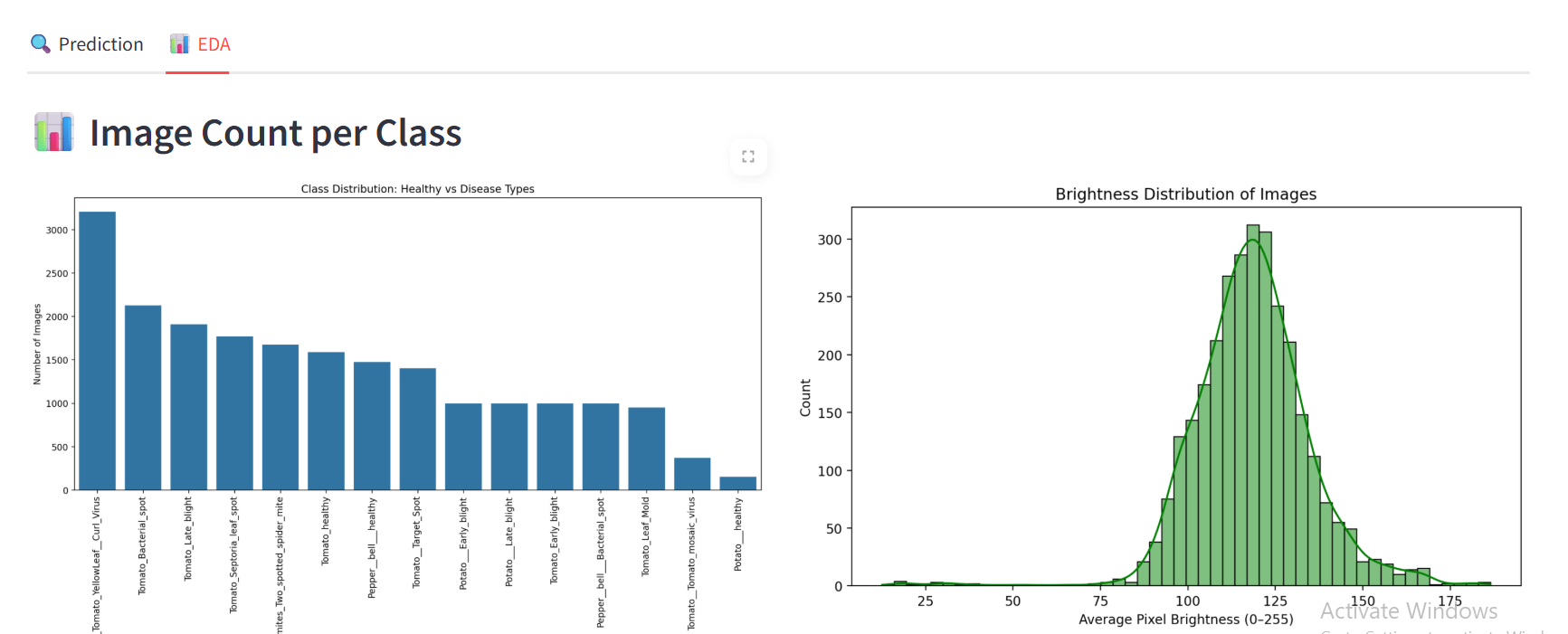
## **5.4. Prediction and Output Display**

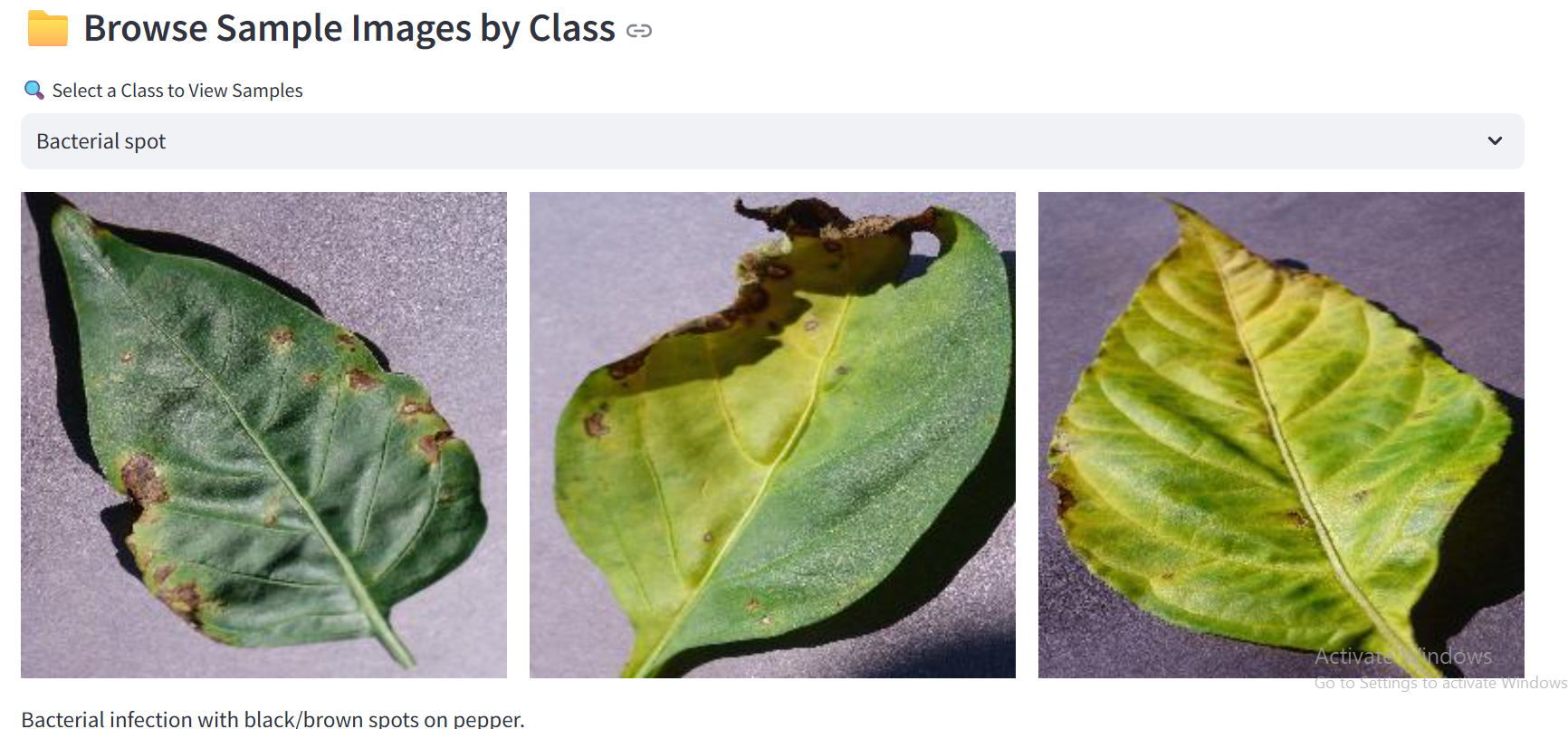
The model’s output was converted to readable class names using the *class\_names* list, and predictions were displayed in a user-friendly format using confidence levels.

Through careful deployment via a Streamlit interface, the model was effectively moved from development into a usable tool for real-world plant disease diagnosis. This setup provides both ease of use and technical scalability, making it a valuable resource for agriculture-focused users.

****







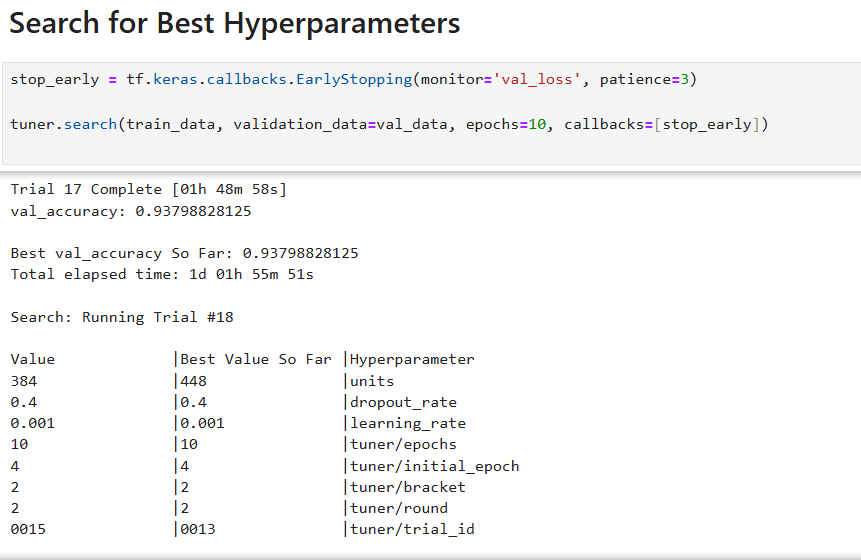
# **Code Implementation**

Here is the sample code from data preprocessing to model deployment;









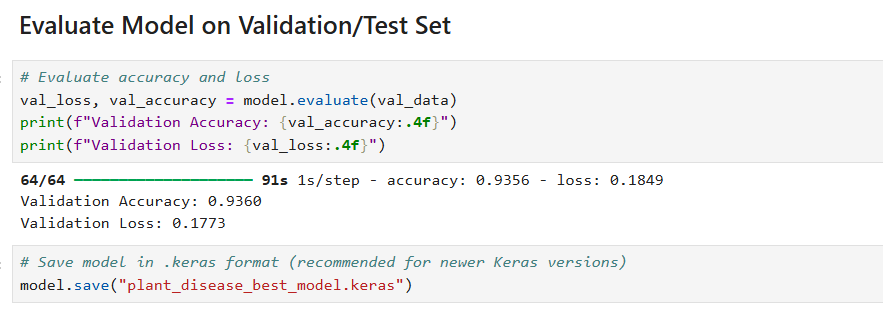


Figure 6 Code implementations Sample

# **Conclusion**

The model refinement and test submission phases demonstrated a successful implementation of a plant disease classification system using transfer learning with MobileNetV2. During refinement, techniques such as freezing the base layers, adding a custom classification head, applying dropout regularization, and using early stopping were employed to enhance generalization and prevent overfitting. The model achieved strong performance on both the validation and test datasets, with a final test accuracy of **94%**, and high precision, recall, and F1-scores across most of the 15 plant disease classes. The use of normalized input data and balanced batches contributed to stable training and convergence.

One challenge encountered was ensuring the correct preprocessing and consistent formatting of the test dataset to match training conditions. Initially, misalignment in dataset handling caused errors during evaluation, such as incorrect directory references and attribute issues with improperly prepared datasets. After resolving these, the model was successfully applied to the test set, yielding robust performance, particularly in classes like "Pepper\_bell\_Bacterial\_spot" and "Potato\_Early\_blight," which achieved perfect scores. However, minor misclassifications occurred in visually similar classes, such as "Tomato\_healthy" and "Tomato\_Late\_blight," indicating opportunities for further refinement using advanced augmentation or ensembling. Overall, the project achieved its goal of developing an accurate and generalizable plant disease detection model.