In [1]:
```
!pip install pyarrow
```

```
Requirement already satisfied: pyarrow in c:\users\alkaid\appdata\local\programs\python\python310\lib\site-packages (15.0.0)
Requirement already satisfied: numpy<2,>=1.16.6 in c:\users\alkaid\appdata\roaming\python\python310\site-packages (from pyarrow)
(1.23.0)
```

In [2]:
```
!pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\alkaid\appdata\local\programs\python\python310\lib\site-packages (1.4.1.
post1)
Requirement already satisfied: numpy<2.0,>=1.19.5 in c:\users\alkaid\appdata\roaming\python\python310\site-packages (from scikit
-learn) (1.23.0)
Requirement already satisfied: scipy>=1.6.0 in c:\users\alkaid\appdata\roaming\python\python310\site-packages (from scikit-lear
n) (1.8.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\alkaid\appdata\local\programs\python\python310\lib\site-packages (from
scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\alkaid\appdata\local\programs\python\python310\lib\site-packages
(from scikit-learn) (3.3.0)
```

Introduction

Objective

The core objective of this project is to employ machine learning and deep learning models for the automatic recognition and classification of potato leaf images. The aim is to differentiate the health status of the leaves into three categories: healthy, early blight, and late blight. The models used in the project include:

- Machine Learning Models: KNN and Support Vector Machine (SVM) are selected for their ability to classify images based on learned patterns and features from the data. These models are expected to provide a baseline for accuracy and efficiency in classification tasks.
- Deep Learning Models: Convolutional Neural Network (CNN), VGG16, and Inception V3 are chosen for their advanced capabilities in image recognition and classification. These models are anticipated to enhance the project's ability to accurately identify and classify the health status of potato leaves by learning from complex patterns in the data.

External Validation

To further evaluate the performance and generalization ability of our trained model, we have carefully curated an **additional dataset consisting of 310 photographs** representing three distinct types of potato leaves. **This external validation set, sourced independently from the initial training dataset, aims to provide a comprehensive assessment of the model's predictive accuracy and robustness** when faced with previously unseen data. The inclusion of these diverse and new images in our validation process is critical for ensuring the model's applicability and reliability in real-world scenarios, where variability and unpredictability of data are common.

SVM (Support Vector Machine)

Data Source

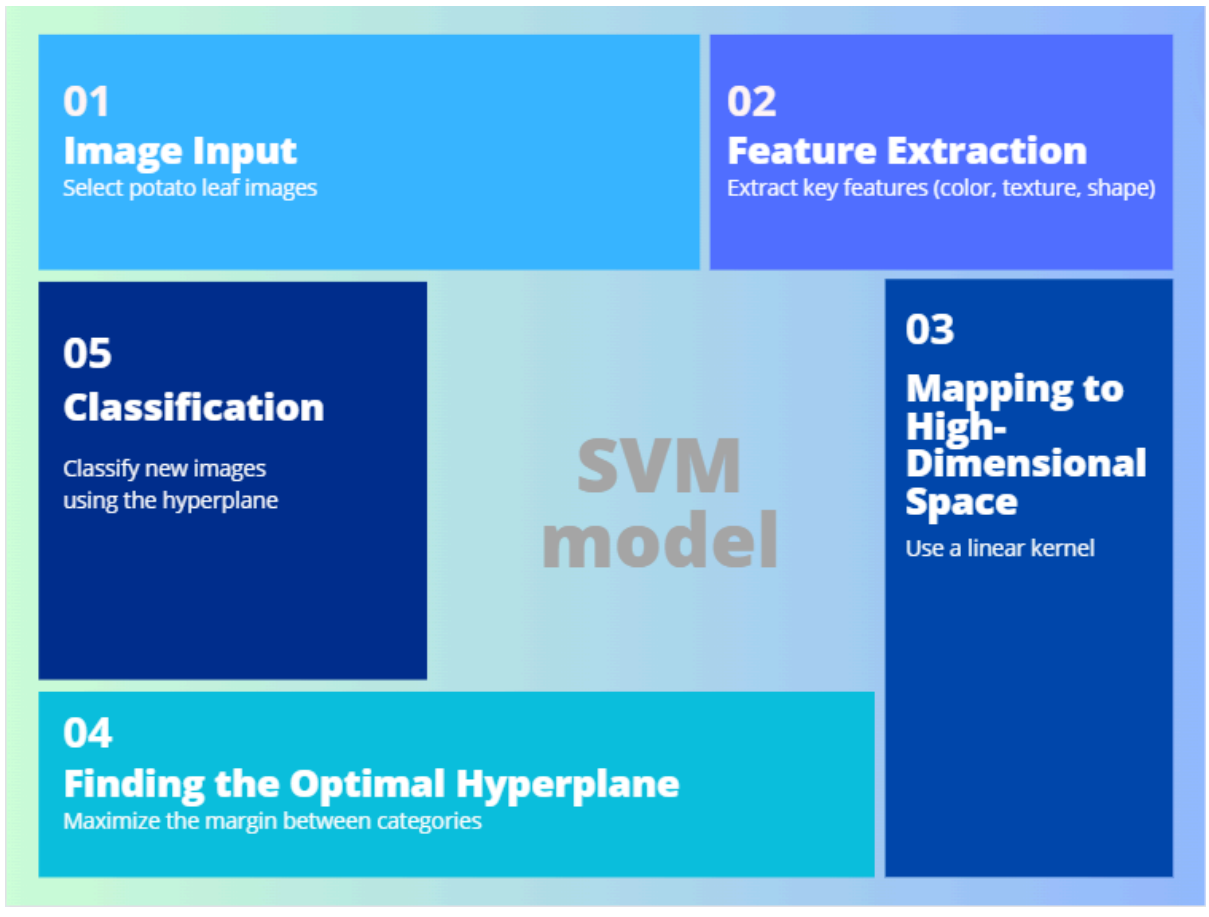
Because SVM supports input csv file as data, so we use the **normalized pixel data obtained in Module1**, including normalized data of the **original image** (2236 * 196608), normalized data of the **compressed image** (2236 * 12288), normalized data of the **sharpened image**(2236 * 196608), normalized data of the **horizontal flipped image** (2236 * 196608), normalized data of the **sharpened and compressed image** (2236 * 12288), and normalized data of the **horizontal flipped and compressed** image (2236 * 196608).In this section, we not only hope to evaluate the performance of the SVM model in image recognition, but also hope to find the most suitable way to process images for this model.
Image normalization is the linear scaling of the original pixel values of an image so that the range is limited to the interval [0, 1].

$$\text{pixel}_{\text{normalized}} = \frac{\text{pixel}}{255}$$

Introduction

SVM is a frequently used classification algorithm in ML. Here's how SVM's basic principles apply to our project:

- **High-Dimensional Space Mapping:** SVM works by **mapping input features into a high-dimensional space** where the different categories (healthy, early blight, and late blight) can be separated more easily. For potato leaf images, features might include color, texture, shape, and edge characteristics that are indicative of the leaf's health status. Even if these features are not linearly separable in the original input space, SVM's ability to operate in higher dimensions allows for more complex separations.
- **Optimal Hyperplane:** The core of SVM's methodology is to **find an optimal hyperplane that separates the classes with the maximum margin**. In the case of potato leaf classification, this means identifying a boundary that best divides the images into the three specified categories. The optimal hyperplane is the one that has the largest distance to the nearest training data points of any class (support vectors), ensuring that the model is as accurate and robust as possible.



Hyperparameters

Several key hyperparameters of SVM significantly influence the model's performance and outcomes. Because of computational resource limitations, we **use grid search cross-validation to exclusively optimize C. The selections for the other two hyperparameters, kernel and tol, were informed by thorough literature review**, reflecting a strategic approach to harnessing existing knowledge for optimal model tuning despite hardware constraints.

- **C (Regularization Parameter):**

The C parameter in SVM is the regularization parameter, which determines the **model's tolerance for misclassified points**. A smaller value of C permits more misclassifications and potentially leading to a smoother decision boundary. Conversely, a larger C value means the model will try to minimize misclassifications in the training set, which could lead to a more complex model and risk overfitting.

- **Kernel (Kernel Function):**

The kernel function plays a crucial role in how the algorithm processes data, affecting the shape and effectiveness of the decision boundary. For our project, we have chosen the **linear kernel** as the default setting. The linear kernel simplifies the model, potentially enhancing its interpretability and speeding up the training process, while still aiming for high accuracy in distinguishing between healthy, early blight, and late blight potato leaves.

- **Tol (Tolerance for Stopping Criterion):**

Based on our literature review, **the tol has been set to 0.001**. This value dictates the precision of the solution and was selected to balance the need for model accuracy with computational efficiency. A tol of 0.001 ensures that the optimization algorithm will continue to make adjustments until the error rate is within this margin, preventing premature termination of the model training process.

Advantages (Model Selection Reasons)

Choosing Support Vector Machine (SVM) as one of the primary tools for our image recognition project is rooted in its unique strengths in handling high-dimensional data, its robustness, exceptional generalization ability, efficacy in linearly separable problems, and resistance to overfitting. Below, we detail these advantages:

1. High-dimensional feature processing

Potato leaf images are rich in features, including but not limited to texture and color. SVM, particularly its linear variant, excels in managing high-dimensional spaces. This capability is crucial as it ensures efficient classification even when the dimensionality of features significantly surpasses the sample size, a common scenario in image recognition tasks.

2. Robustness Against Noise:

Linear SVM is relatively **insensitive to noise and uncorrelated features** in the input data. This attribute is invaluable given the **complex backgrounds and potential noise** inherent in potato leaf images. By focusing on the most relevant features for classification, **SVM ensures reliable performance even in less-than-ideal imaging conditions.**

3. Strong Generalization ability:

The objective of linear SVM—to find a hyperplane that maximizes the margin between classes—equips it with superior generalization capabilities. This means SVM models are not just tailored to the training data but also perform well on unseen images, a critical feature for real-world application where new and unknown variations of potato leaf diseases are encountered.

4. Effectiveness in Linearly Separable Problems:

In instances where potato leaf diseases can be linearly separated in the feature space, linear SVM is particularly powerful. By identifying a clear hyperplane to separate the classes, it ensures high accuracy in distinguishing between different diseases, provided the problem is linearly differentiable.

5. Reduced Overfitting Risk:

The adaptability of SVM in **selecting the appropriate regularization parameter (or cost parameter, C) minimizes the risk of overfitting.** This characteristic is essential for building models that not only fit the training data well but also maintain their accuracy on new, unseen images. The strategic selection of the cost parameter (C) becomes a pivotal aspect of our model optimization process, ensuring that our SVM model is both accurate and generalizable.

Model Performance Evaluation

To assess the accuracy of classifiers in multi-class imbalance learning, **recall, precision and overall accuracy** are appropriate metrics. Specifically, they are computed from the confusion matrix. Each entry in the confusion matrix represents the number of predictions made by the classifier in which the classifier correctly classified or misclassified:\

- **TP is True Positive**, which indicates the number of samples that the model correctly predicted as positive categories;
- **TN is True Negatives**, which indicates the number of samples that the model correctly predicts as negative categories;
- **FP is False Positives**, which indicates the number of samples that the model incorrectly predicted to be in the positive category but were actually in the negative category;\
- **FN is False Negative**, which is the number of samples that the model incorrectly predicts as negative categories.

Recall for each class is defined as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Eg： For early blight (Class 1), Recall score is equal to 0.9,which means that the model successfully identified 90% of the actual early blight samples (True Positive), while 10% of the actual early blight samples were incorrectly predicted to be healthy or late blight (False Negative).

Precision for each class is defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Eg: For early blight (Class 1), Precision score is equal to 0.96. This means that out of all samples predicted as early blight by the model (True Positive + False Positive), 96% were correctly identified as actual early blight samples (True Positive), while 4% were incorrectly predicted as early blight when they were not (False Positive).

Overall Accuracy can be represented by the formula:

$$\text{Accuracy} = \frac{\text{True Positives (TP) for all classes}}{\text{Total number of samples}}$$

For multi-class classification problems, the concept of True Negatives (TN) is not as straightforward as in binary classification, because for a given class, samples from all other classes can be considered as negative samples. Thus, a simpler and more commonly used approach is to only count the True Positives (TP) for all classes, which are the correctly classified samples, and divide this number by the total number of samples to get the Overall Accuracy.

Eg:The given Overall Accuracy of (0.9040178571428571) indicates that about (90.4%) of the samples were correctly classified by the model across all categories.In the context of multi-class classification problems, Overall Accuracy is a crucial metric because it provides a **quick overview of how the model performs across the entire test set**. However, it does **not detail the performance on individual classes**, such as whether one class is more challenging to classify correctly than others, or if the model is more prone to specific types of errors (e.g., misclassifying healthy leaves as early or late blight).

In this context, an Overall Accuracy of (90.4%) is generally considered a good performance, especially in complex tasks like classifying diseases in potato leaves, which can have subtle visual differences. However, for a comprehensive understanding of the model's performance, it's **also important to look at class-specific metrics such as precision and recall**, as these can highlight areas where the model might need improvement or where it performs particularly well.

In classification problems, accuracy, recall, and F1-score are commonly used metrics to evaluate a model's performance. For multi-class problems, in addition to calculating these metrics for each class, their averages are computed to provide an overall assessment of the model's performance. "Macro avg" and "weighted avg" refer to two different methods of averaging:

Macro Average:

- The macro average treats all classes with equal importance, implying that each class is equally important.
- The macro average formula is roughly:

  $$\text{Macro Average} = \frac{1}{N} \sum_{i=1}^{N} \text{Metric}_i$$

  where (N) is the number of classes, and Metric_i is the metric (precision, recall, or F1-score) for each class.

Weighted Average:

- The weighted average takes the average considering the number of instances in each class as weights. This means that classes with more instances have a greater impact on the average. This method **reflects the imbalance in the dataset more accurately**.
- The weighted average formula is roughly:

$$\text{Weighted Average} = \sum_{i=1}^{N} w_i \times \text{Metric}_i$$

where w_i is the weight for class (i) (often the proportion of instances of class (i) to the total number of instances), and Metric_i is the metric for each class.

We characterized the model for data from each of the six different image processing modalities, and we showed the model performance without hyperparameter selection and after hyperparameter selection. Grid Search CV was used for the selection of the hyperparameter C, which yielded an optimal C of 0.1.

Original

Without regularization parameter choosing

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

early_blight_path = 'Normalized/Potato_Early_blight_1_normalized.csv'
late_blight_path = 'Normalized/Potato_Late_blight_1_normalized.csv'
healthy_path = 'Normalized/Potato_Healthy_1_normalized.csv'

# Use pandas to load the files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Create an imputer to fill NaN values with the mean
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Use the imputer to fill NaN values in both training and testing data
features_imputed = imputer.fit_transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an SVM classifier
model = SVC(kernel='linear')

# Train the model
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Print classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print(model.get_params())
```

```
              precision    recall  f1-score   support

           0       0.88      0.75      0.81        48
           1       0.96      0.90      0.93       204
           2       0.86      0.95      0.90       196

    accuracy                           0.90       448
   macro avg       0.90      0.87      0.88       448
weighted avg       0.91      0.90      0.90       448

Accuracy: 0.9040178571428571
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degre
e': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'to
l': 0.001, 'verbose': False}
```

After regularization parameter choosing

In [4]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from joblib import dump

early_blight_path = 'Normalized/Potato_Early_blight_1_normalized.csv'
late_blight_path = 'Normalized/Potato_Late_blight_1_normalized.csv'
healthy_path = 'Normalized/Potato_Healthy_1_normalized.csv'

# Use pandas to load these files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Imputation of missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
features_imputed = imputer.fit_transform(features)

# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Setting up the SVM model
model = SVC(kernel='linear', tol=0.001)

# Setting the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100]
}

# Using GridSearchCV for parameter tuning
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, scoring='accuracy')

# Training the model
grid_search.fit(X_train_scaled, y_train)

# Printing the best parameters and the best cross-validation score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score (accuracy):", grid_search.best_score_)


# Predicting with the model that has the best parameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

# Printing the classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))


# Save the imputer
dump(imputer, 'imputer_original.joblib')
print("Imputer saved as 'imputer_original.joblib'")

# Save the scaler
dump(scaler, 'scaler_original.joblib')
print("Scaler saved as 'scaler_original.joblib'")

# Save the model
dump(grid_search.best_estimator_, 'best_original.joblib')
print("Model saved as 'best_original.joblib'")
```

```
Best parameters: {'C': 0.1}
Best cross-validation score (accuracy): 0.9161103547564278
              precision    recall  f1-score   support

           0       0.88      0.75      0.81        48
           1       0.96      0.90      0.93       204
           2       0.86      0.95      0.90       196

    accuracy                           0.90       448
   macro avg       0.90      0.87      0.88       448
weighted avg       0.91      0.90      0.90       448

Accuracy: 0.9040178571428571
Imputer saved as 'imputer_original.joblib'
Scaler saved as 'scaler_original.joblib'
Model saved as 'best_original.joblib'
```

Overall Metrics: The high overall accuracy of 90.40% indicates that the model consistently classifies the images correctly, demonstrating robust overall performance.

Classification Details:

- **Class 0:** With 48 test samples in Class 0, the model exhibits a relatively balanced performance. The precision and recall are 0.88 and 0.75, respectively. However, it is worth noting that the performance in Class 0 is comparatively lower than the other two classes.
- **Class 1:** Class 1, consisting of 204 samples, shows high precision (0.96) but a slightly lower recall of 0.90. The F1-score is 0.93, indicating that the model accurately identifies Class 1, although there is room for improvement in recall.
- **Class 2:** For the 196 samples in Class 2, the model displays a relatively high recall of 0.95 but a lower precision of 0.86. The model effectively captures the majority of images in this class but experiences some misclassifications.

Summarizing: There is room for improvement, especially in enhancing the performance for Class 0 and addressing the recall for Class 1 to achieve more comprehensive classification.

Compressed

Without regularization parameter choosing

In [6]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

early_blight_path = 'Normalized/early_blight_normalized_compressed_values.csv'
late_blight_path = 'Normalized/late_blight_normalized_compressed_values.csv'
healthy_path = 'Normalized/healthy_normalized_compressed_values.csv'

# Use pandas to load the files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Create an imputer to fill NaN values with the mean
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Use the imputer to fill NaN values in both training and testing data
features_imputed = imputer.fit_transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an SVM classifier
model = SVC(kernel='linear')

# Train the model
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Print classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print(model.get_params())
```

```
              precision    recall  f1-score   support

           0       0.87      0.83      0.85        48
           1       0.96      0.89      0.92       204
           2       0.87      0.95      0.91       196

    accuracy                           0.91       448
   macro avg       0.90      0.89      0.89       448
weighted avg       0.91      0.91      0.91       448

Accuracy: 0.9084821428571429
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degre
e': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'to
l': 0.001, 'verbose': False}
```

After regularization parameter choosing

In [7]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from joblib import dump

early_blight_path = 'Normalized/early_blight_normalized_compressed_values.csv'
late_blight_path = 'Normalized/late_blight_normalized_compressed_values.csv'
healthy_path = 'Normalized/healthy_normalized_compressed_values.csv'

# Use pandas to load these files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Imputation of missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
features_imputed = imputer.fit_transform(features)

# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Setting up the SVM model
model = SVC(kernel='linear', tol=0.001)

# Setting the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100]
}

# Using GridSearchCV for parameter tuning
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, scoring='accuracy')

# Training the model
grid_search.fit(X_train_scaled, y_train)

# Printing the best parameters and the best cross-validation score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score (accuracy):", grid_search.best_score_)

# Predicting with the model that has the best parameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

# Printing the classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))

# Save the imputer
dump(imputer, 'imputer_compressed.joblib')
print("Imputer saved as 'imputer_compressed.joblib'")

# Save the scaler
dump(scaler, 'scaler_compressed.joblib')
print("Scaler saved as 'scaler_compressed.joblib'")

# Save the model
dump(grid_search.best_estimator_, 'best_compressed.joblib')
print("Model saved as 'best_compressed.joblib'")
```

```
Best parameters: {'C': 0.1}
Best cross-validation score (accuracy): 0.9133154937952834
              precision    recall  f1-score   support

           0       0.87      0.83      0.85        48
           1       0.96      0.89      0.92       204
           2       0.87      0.95      0.91       196

    accuracy                           0.91       448
   macro avg       0.90      0.89      0.89       448
weighted avg       0.91      0.91      0.91       448

Accuracy: 0.9084821428571429
Imputer saved as 'imputer_compressed.joblib'
Scaler saved as 'scaler_compressed.joblib'
Model saved as 'best_compressed.joblib'
```

- **Overall Metrics:** The high overall accuracy indicates that the model consistently classifies 90.84% of images correctly, demonstrating robust overall performance.
- **Classification Details:**
  - **Class 0:** With 48 test samples in Class 0, the model exhibits a relatively balanced performance. Precision and recall fall between 0.87 and 0.83.But it performed poorly compared to the other two classes.
  - **Class 1:** Class 1, consisting of 204 samples, shows high precision (0.97), but the recall is slightly lower at 0.89. This suggests that the model accurately identifies Class 1 but may face challenges in capturing all instances of this class.
  - **Class 2:** For the 196 samples in Class 2, the model displays a relatively high recall (0.95) but lower precision (0.87). The model effectively captures the majority of images in this class but experiences some misclassifications.
- **Summarizing:** The model exhibits excellent overall performance, particularly in precision for Class 1 and recall for Class 2. Further optimization may involve addressing the recall for Class 1 to capture more instances comprehensively and enhancing the precision for Class 2 to reduce misclassifications.

Sharpen

Without regularization parameter choosing

In [9]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

early_blight_path = 'Normalized/early_blight_normalized_sharpen_values.csv'
late_blight_path = 'Normalized/late_blight_normalized_sharpen_values.csv'
healthy_path = 'Normalized/healthy_normalized_sharpen_values.csv'

# Use pandas to load the files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Create an imputer to fill NaN values with the mean
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Use the imputer to fill NaN values in both training and testing data
features_imputed = imputer.fit_transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an SVM classifier
model = SVC(kernel='linear')

# Train the model
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Print classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print(model.get_params())
```

```
              precision    recall  f1-score   support

           0       0.93      0.58      0.72        48
           1       0.96      0.90      0.93       204
           2       0.82      0.96      0.89       196

    accuracy                           0.89       448
   macro avg       0.91      0.81      0.84       448
weighted avg       0.90      0.89      0.89       448

Accuracy: 0.890625
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degre
e': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'to
l': 0.001, 'verbose': False}
```

After regularization parameter choosing

In [10]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from joblib import dump

early_blight_path = 'Normalized/early_blight_normalized_sharpen_values.csv'
late_blight_path = 'Normalized/late_blight_normalized_sharpen_values.csv'
healthy_path = 'Normalized/healthy_normalized_sharpen_values.csv'

# Use pandas to load these files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Imputation of missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
features_imputed = imputer.fit_transform(features)

# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Setting up the SVM model
model = SVC(kernel='linear', tol=0.001)

# Setting the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100]
}

# Using GridSearchCV for parameter tuning
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, scoring='accuracy')

# Training the model
grid_search.fit(X_train_scaled, y_train)

# Printing the best parameters and the best cross-validation score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score (accuracy):", grid_search.best_score_)

# Predicting with the model that has the best parameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

# Printing the classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))

# Save the imputer
dump(imputer, 'imputer_sharpen.joblib')
print("Imputer saved as 'imputer_sharpen.joblib'")

# Save the scaler
dump(scaler, 'scaler_sharpen.joblib')
print("Scaler saved as 'scaler_sharpen.joblib'")

# Save the model
dump(grid_search.best_estimator_, 'best_sharpen.joblib')
print("Model saved as 'best_sharpen.joblib'")
```

```
Best parameters: {'C': 0.1}
Best cross-validation score (accuracy): 0.88871101513231
              precision    recall  f1-score   support

           0       0.93      0.58      0.72        48
           1       0.96      0.90      0.93       204
           2       0.82      0.96      0.89       196

    accuracy                           0.89       448
   macro avg       0.91      0.81      0.84       448
weighted avg       0.90      0.89      0.89       448

Accuracy: 0.890625
Imputer saved as 'imputer_sharpen.joblib'
Scaler saved as 'scaler_sharpen.joblib'
Model saved as 'best_sharpen.joblib'
```

Overall Metrics: The model achieves an accuracy of 89.06%, showcasing its ability to correctly classify the majority of images and demonstrating robust overall performance.

Classification Details:

- **Class 0:** Class 0, with 48 test samples, exhibits a precision of 0.93 but a notably lower recall of 0.58. The lower recall in this class suggests that the model struggles to identify a substantial portion of instances belonging to Class 0. This could be attributed to various factors such as class imbalance or inherent complexity in distinguishing features. Improving recall for Class 0 is crucial to ensure a more comprehensive identification of instances in this category.
- **Class 1:** The model excels in accurately identifying instances of Class 1.
- **Class 2:** Class 2, with 196 samples, shows a high recall of 0.96 and a precision of 0.82, resulting in an F1-score of 0.89. The model effectively captures the majority of images in this class but exhibits some room for improvement in precision.

Summarizing: While the model performs well overall, addressing the notably lower recall for Class 0 is essential for more comprehensive identification of instances in this category.

Flipped

Without regularization parameter choosing

In [14]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

early_blight_path = 'Normalized/early_blight_flipped_normalized_pixel_values.csv'
late_blight_path = 'Normalized/late_blight_flipped_normalized_pixel_values.csv'
healthy_path = 'Normalized/healthy_flipped_normalized_pixel_values.csv'

# Use pandas to load the files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Create an imputer to fill NaN values with the mean
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Use the imputer to fill NaN values in both training and testing data
features_imputed = imputer.fit_transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an SVM classifier
model = SVC(kernel='linear')

# Train the model
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Print classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print(model.get_params())
```

```
              precision    recall  f1-score   support

           0       0.88      0.75      0.81        48
           1       0.96      0.90      0.93       204
           2       0.86      0.95      0.90       196

    accuracy                           0.90       448
   macro avg       0.90      0.87      0.88       448
weighted avg       0.91      0.90      0.90       448

Accuracy: 0.9040178571428571
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degre
e': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'to
l': 0.001, 'verbose': False}
```

After regularization parameter choosing

In [15]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from joblib import dump

early_blight_path = 'Normalized/early_blight_flipped_normalized_pixel_values.csv'
late_blight_path = 'Normalized/late_blight_flipped_normalized_pixel_values.csv'
healthy_path = 'Normalized/healthy_flipped_normalized_pixel_values.csv'

# Use pandas to load these files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Imputation of missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
features_imputed = imputer.fit_transform(features)

# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Setting up the SVM model
model = SVC(kernel='linear', tol=0.001)

# Setting the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100]
}

# Using GridSearchCV for parameter tuning
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, scoring='accuracy')

# Training the model
grid_search.fit(X_train_scaled, y_train)

# Printing the best parameters and the best cross-validation score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score (accuracy):", grid_search.best_score_)

# Predicting with the model that has the best parameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

# Printing the classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))

# Save the imputer
dump(imputer, 'imputer_flipped.joblib')
print("Imputer saved as 'imputer_flipped.joblib'")

# Save the scaler
dump(scaler, 'scaler_flipped.joblib')
print("Scaler saved as 'scaler_flipped.joblib'")

# Save the model
dump(grid_search.best_estimator_, 'best_flipped.joblib')
print("Model saved as 'best_flipped.joblib'")
```

```
Best parameters: {'C': 0.1}
Best cross-validation score (accuracy): 0.9161103547564278
              precision    recall  f1-score   support

           0       0.88      0.75      0.81        48
           1       0.96      0.90      0.93       204
           2       0.86      0.95      0.90       196

    accuracy                           0.90       448
   macro avg       0.90      0.87      0.88       448
weighted avg       0.91      0.90      0.90       448

Accuracy: 0.9040178571428571
Imputer saved as 'imputer_flipped.joblib'
Scaler saved as 'scaler_flipped.joblib'
Model saved as 'best_flipped.joblib'
```

Overall Metrics: The model achieves an impressive overall accuracy of 90.40%, indicating its consistent ability to correctly classify the majority of images and demonstrating robust overall performance.

Comparative Analysis: Comparing precision and recall across classes, it is evident that the model performs exceptionally well in Class 1, achieving both high precision and recall. In contrast, Class 0 demonstrates a balanced but slightly lower performance, particularly in recall. Class 2 excels in recall but shows a relatively lower precision. This highlights the trade-off between precision and recall, emphasizing the need for a balanced approach depending on the specific requirements of the classification task.

Sharpen & Compressed

Without regularization parameter choosing

In [17]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

early_blight_path = 'Normalized/early_blight_normalized_sharpen_compressed_values.csv'
late_blight_path = 'Normalized/late_blight_normalized_sharpen_compressed_values.csv'
healthy_path = 'Normalized/healthy_normalized_sharpen_compressed_values (1).csv'

# Use pandas to load the files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Create an imputer to fill NaN values with the mean
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Use the imputer to fill NaN values in both training and testing data
features_imputed = imputer.fit_transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an SVM classifier
model = SVC(kernel='linear')

# Train the model
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Print classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print(model.get_params())
```

```
              precision    recall  f1-score   support

           0       0.87      0.83      0.85        48
           1       0.97      0.90      0.93       204
           2       0.87      0.95      0.91       196

    accuracy                           0.91       448
   macro avg       0.90      0.89      0.90       448
weighted avg       0.92      0.91      0.91       448

Accuracy: 0.9129464285714286
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degre
e': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'to
l': 0.001, 'verbose': False}
```

After regularization parameter choosing

In [18]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from joblib import dump

early_blight_path = 'Normalized/early_blight_normalized_sharpen_compressed_values.csv'
late_blight_path = 'Normalized/late_blight_normalized_sharpen_compressed_values.csv'
healthy_path = 'Normalized/healthy_normalized_sharpen_compressed_values (1).csv'

# Use pandas to load these files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Imputation of missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
features_imputed = imputer.fit_transform(features)

# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Setting up the SVM model
model = SVC(kernel='linear', tol=0.001)

# Setting the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100]
}

# Using GridSearchCV for parameter tuning
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, scoring='accuracy')

# Training the model
grid_search.fit(X_train_scaled, y_train)

# Printing the best parameters and the best cross-validation score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score (accuracy):", grid_search.best_score_)

# Predicting with the model that has the best parameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

# Printing the classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))

# Save the imputer
dump(imputer, 'imputer_SC.joblib')
print("Imputer saved as 'imputer_SC.joblib'")

# Save the scaler
dump(scaler, 'scaler_SC.joblib')
print("Scaler saved as 'scaler_SC.joblib'")

# Save the model
dump(grid_search.best_estimator_, 'best_SC.joblib')
print("Model saved as 'best_SC.joblib'")
```

```
Best parameters: {'C': 0.1}
Best cross-validation score (accuracy): 0.906600629078447
              precision    recall  f1-score   support

           0       0.87      0.83      0.85        48
           1       0.97      0.90      0.93       204
           2       0.87      0.95      0.91       196

    accuracy                           0.91       448
   macro avg       0.90      0.89      0.90       448
weighted avg       0.92      0.91      0.91       448

Accuracy: 0.9129464285714286
Imputer saved as 'imputer_SC.joblib'
Scaler saved as 'scaler_SC.joblib'
Model saved as 'best_SC.joblib'
```

Overall Metrics: The model achieved an outstanding cross-validation accuracy score of 90.66%.

Comparative Analysis: Comparing precision and recall across classes, the model maintains a well-balanced performance, with particularly high precision and recall in Class 1. While Classes 0 and 2 also show strong performance, there is a slight trade-off between precision and recall in these classes, underscoring the importance of considering specific task requirements.

Flipped & Compressed

Without regularization parameter choosing

In [20]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

early_blight_path = 'Normalized/early_blight_64x64_compressed_flipped_normalized_pixel_values.csv'
late_blight_path = 'Normalized/late_blight_64x64_compressed_flipped_normalized_pixel_values.csv'
healthy_path = 'Normalized/healthy_64x64_compressed_flipped_normalized_pixel_values.csv'

# Use pandas to load the files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Create an imputer to fill NaN values with the mean
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Use the imputer to fill NaN values in both training and testing data
features_imputed = imputer.fit_transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an SVM classifier
model = SVC(kernel='linear')

# Train the model
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Print classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print(model.get_params())
```

```
              precision    recall  f1-score   support

           0       0.89      0.88      0.88        48
           1       0.97      0.89      0.93       204
           2       0.87      0.95      0.91       196

    accuracy                           0.91       448
   macro avg       0.91      0.90      0.91       448
weighted avg       0.92      0.91      0.91       448

Accuracy: 0.9129464285714286
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degre
e': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'to
l': 0.001, 'verbose': False}
```

After regularization parameter choosing

In [21]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from joblib import dump

early_blight_path = 'Normalized/early_blight_64x64_compressed_flipped_normalized_pixel_values.csv'
late_blight_path = 'Normalized/late_blight_64x64_compressed_flipped_normalized_pixel_values.csv'
healthy_path = 'Normalized/healthy_64x64_compressed_flipped_normalized_pixel_values.csv'

# Use pandas to load these files
early_blight_data = pd.read_csv(early_blight_path)
late_blight_data = pd.read_csv(late_blight_path)
healthy_data = pd.read_csv(healthy_path)

# Create labels, 0 for healthy, 1 for early blight, 2 for late blight
healthy_labels = [0] * len(healthy_data)
early_blight_labels = [1] * len(early_blight_data)
late_blight_labels = [2] * len(late_blight_data)

# Merge data and labels
features = pd.concat([healthy_data, early_blight_data, late_blight_data], ignore_index=True)
labels = healthy_labels + early_blight_labels + late_blight_labels

# Imputation of missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
features_imputed = imputer.fit_transform(features)

# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features_imputed, labels, test_size=0.2, random_state=42)

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Setting up the SVM model
model = SVC(kernel='linear', tol=0.001)

# Setting the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100]
}

# Using GridSearchCV for parameter tuning
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, scoring='accuracy')

# Training the model
grid_search.fit(X_train_scaled, y_train)

# Printing the best parameters and the best cross-validation score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score (accuracy):", grid_search.best_score_)

# Predicting with the model that has the best parameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

# Printing the classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))

# Save the imputer
dump(imputer, 'imputer_FC.joblib')
print("Imputer saved as 'imputer_FC.joblib'")

# Save the scaler
dump(scaler, 'scaler_FC.joblib')
print("Scaler saved as 'scaler_FC.joblib'")

# Save the model
dump(grid_search.best_estimator_, 'best_FC.joblib')
print("Model saved as 'best_FC.joblib'")
```

```
Best parameters: {'C': 0.1}
Best cross-validation score (accuracy): 0.9144343771027966
              precision    recall  f1-score   support

           0       0.89      0.88      0.88        48
           1       0.97      0.89      0.93       204
           2       0.87      0.95      0.91       196

    accuracy                           0.91       448
   macro avg       0.91      0.90      0.91       448
weighted avg       0.92      0.91      0.91       448

Accuracy: 0.9129464285714286
Imputer saved as 'imputer_FC.joblib'
Scaler saved as 'scaler_FC.joblib'
Model saved as 'best_FC.joblib'
```

Overall Metrics: The model demonstrates an excellent overall accuracy of 91.29%.

Comparative Analysis: Class 1 stands out with high precision and recall, while Classes 0 and 2 also show strong performance, reflecting a balanced trade-off between precision and recall.

Results Comparison

		Macro Avg		Weighted Avg	
	Overall Accuracy	Precision	Recall	Precision	Recall
Original	0.904	0.9	0.87	0.91	0.9
Sharpen	0.8906	0.91	0.81	0.9	0.89
Compressed	0.9085	0.9	0.89	0.91	0.91
Compressed_Sharpen	0.9129	0.9	0.89	0.92	0.91
Flipped	0.904	0.9	0.87	0.91	0.9
Flipped_Compressed	0.9129	0.91	0.9	0.92	0.91

1. **Overall Accuracy:**
   - The overall accuracy fluctuates modestly across different processing methods, ranging from a minimum of 0.8906 (Sharpen) to a maximum of 0.9129 (Compressed_Sharpen and Flipped_Compressed).
2. **Macro Avg Precision and Recall:**
   - In terms of Macro Avg Precision, Flipped_Compressed and Sharpen achieves the highest, while in Recall, Flipped_Compressed is at the maximum, indicating varying impacts of image processing methods on the average precision and recall across different categories.
3. **Weighted Avg Precision and Recall:**
   - Flipped_Compressed and Compressed_Sharpen attain the highest values in Weighted Avg Precision, potentially excelling in scenarios with larger category support. In Weighted Avg Recall, Compressed, Compressed_Sharpen, and Flipped_Compressed reach the maximum values, making them suitable for scenarios where each category's support is considered.

In-Depth Analysis:

- Flipped_Compressed consistently performs best across most metrics, showcasing a well-balanced overall performance.
- Compressed and Compressed_Sharpen exhibit relatively strong performance in Weighted Avg Recall, making them suitable for scenarios with larger category support.
- Sharpen, while performing well in Macro Avg Precision, exhibits a slight decrease in Recall, suggesting cautious use, especially in scenarios emphasizing higher Recall.

Conclusion:

- Depending on the specific application context, **Flipped_Compressed can be considered as the preferred image processing method, excelling across various metrics.**

External Validation

To further evaluate the performance and generalization ability of our trained model, we have carefully curated an **additional dataset consisting of 310 photographs** representing three distinct types of potato leaves. **This external validation set, sourced independently from the initial training dataset, aims to provide a comprehensive assessment of the model's predictive accuracy and robustness** when faced with previously unseen data. The inclusion of these diverse and new images in our validation process is critical for ensuring the model's applicability and reliability in real-world scenarios, where variability and unpredictability of data are common.

Original

In [5]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from joblib import load

# Load the trained model
model = load('best_original.joblib')

# Load the imputer and scaler
imputer = load('imputer_original.joblib')
scaler = load('scaler_original.joblib')

# Paths to the new test data files
early_blight_test_path = 'Extra/testplus_Early_blight_1_normalized.csv'
late_blight_test_path = 'Extra/testplus_Late_blight_1_normalized.csv'
healthy_test_path = 'Extra/testplus_Healthy_1_normalized.csv'

# Load the test data
early_blight_test_data = pd.read_csv(early_blight_test_path)
late_blight_test_data = pd.read_csv(late_blight_test_path)
healthy_test_data = pd.read_csv(healthy_test_path)

# Create labels: 0 for healthy, 1 for early blight, 2 for late blight
test_healthy_labels = [0] * len(healthy_test_data)
test_early_blight_labels = [1] * len(early_blight_test_data)
test_late_blight_labels = [2] * len(late_blight_test_data)
test_labels = test_healthy_labels + test_early_blight_labels + test_late_blight_labels

# Concatenate the new test data
test_features = pd.concat([healthy_test_data, early_blight_test_data, late_blight_test_data], ignore_index=True)

# Impute missing values using the loaded imputer
test_features_imputed = imputer.transform(test_features)

# Scale the test data using the loaded scaler
test_features_scaled = scaler.transform(test_features_imputed)

# Make predictions on the test data
test_predictions = model.predict(test_features_scaled)

# Create a DataFrame with the actual and predicted labels
results_df = pd.DataFrame({
    'Actual Label': test_labels,
    'Predicted Label': test_predictions
})

# Calculate the accuracy
test_accuracy = accuracy_score(test_labels, test_predictions)

# Print the comparison table
print(results_df)

# Print the accuracy
print("Test data accuracy:", test_accuracy)
```

```
     Actual Label  Predicted Label
0               0                1
1               0                1
2               0                1
3               0                0
4               0                1
..            ...              ...
305             2                1
306             2                1
307             2                1
308             2                1
309             2                1

[310 rows x 2 columns]
Test data accuracy: 0.36451612903225805
```

Compressed

In [8]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from joblib import load

# Load the trained model
model = load('best_compressed.joblib')

# Load the imputer and scaler
imputer = load('imputer_compressed.joblib')
scaler = load('scaler_compressed.joblib')

# Paths to the new test data files
early_blight_test_path = 'Extra/early_blight_normalized_compressed_values_test.csv'
late_blight_test_path = 'Extra/late_blight_normalized_compressed_values_test.csv'
healthy_test_path = 'Extra/healthy_normalized_compressed_values_test.csv'

# Load the test data
early_blight_test_data = pd.read_csv(early_blight_test_path)
late_blight_test_data = pd.read_csv(late_blight_test_path)
healthy_test_data = pd.read_csv(healthy_test_path)

# Create labels: 0 for healthy, 1 for early blight, 2 for late blight
test_healthy_labels = [0] * len(healthy_test_data)
test_early_blight_labels = [1] * len(early_blight_test_data)
test_late_blight_labels = [2] * len(late_blight_test_data)
test_labels = test_healthy_labels + test_early_blight_labels + test_late_blight_labels

# Concatenate the new test data
test_features = pd.concat([healthy_test_data, early_blight_test_data, late_blight_test_data], ignore_index=True)

# Impute missing values using the loaded imputer
test_features_imputed = imputer.transform(test_features)

# Scale the test data using the loaded scaler
test_features_scaled = scaler.transform(test_features_imputed)

# Make predictions on the test data
test_predictions = model.predict(test_features_scaled)

# Create a DataFrame with the actual and predicted labels
results_df = pd.DataFrame({
    'Actual Label': test_labels,
    'Predicted Label': test_predictions
})

# Calculate the accuracy
test_accuracy = accuracy_score(test_labels, test_predictions)

# Print the comparison table
print(results_df)

# Print the accuracy
print("Test data accuracy:", test_accuracy)
```

```
     Actual Label  Predicted Label
0               0                1
1               0                1
2               0                1
3               0                0
4               0                1
..            ...              ...
305             2                1
306             2                1
307             2                1
308             2                1
309             2                1

[310 rows x 2 columns]
Test data accuracy: 0.3709677419354839
```

Flipped

In [16]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from joblib import load

# Load the trained model
model = load('best_flipped.joblib')

# Load the imputer and scaler
imputer = load('imputer_flipped.joblib')
scaler = load('scaler_flipped.joblib')

# Paths to the new test data files
early_blight_test_path = 'Extra/early_blight_flipped_normalized_pixel_values_test.csv'
late_blight_test_path = 'Extra/late_blight_flipped_normalized_pixel_values_test.csv'
healthy_test_path = 'Extra/healthy_flipped_normalized_pixel_values_test.csv'

# Load the test data
early_blight_test_data = pd.read_csv(early_blight_test_path)
late_blight_test_data = pd.read_csv(late_blight_test_path)
healthy_test_data = pd.read_csv(healthy_test_path)

# Create labels: 0 for healthy, 1 for early blight, 2 for late blight
test_healthy_labels = [0] * len(healthy_test_data)
test_early_blight_labels = [1] * len(early_blight_test_data)
test_late_blight_labels = [2] * len(late_blight_test_data)
test_labels = test_healthy_labels + test_early_blight_labels + test_late_blight_labels

# Concatenate the new test data
test_features = pd.concat([healthy_test_data, early_blight_test_data, late_blight_test_data], ignore_index=True)

# Impute missing values using the loaded imputer
test_features_imputed = imputer.transform(test_features)

# Scale the test data using the loaded scaler
test_features_scaled = scaler.transform(test_features_imputed)

# Make predictions on the test data
test_predictions = model.predict(test_features_scaled)

# Create a DataFrame with the actual and predicted labels
results_df = pd.DataFrame({
    'Actual Label': test_labels,
    'Predicted Label': test_predictions
})

# Calculate the accuracy
test_accuracy = accuracy_score(test_labels, test_predictions)

# Print the comparison table
print(results_df)

# Print the accuracy
print("Test data accuracy:", test_accuracy)
```

```
     Actual Label  Predicted Label
0               0                1
1               0                1
2               0                1
3               0                0
4               0                1
..            ...              ...
305             2                1
306             2                1
307             2                1
308             2                1
309             2                1

[310 rows x 2 columns]
Test data accuracy: 0.36774193548387096
```

Sharpen & Compressed

In [19]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from joblib import load

# Load the trained model
model = load('best_SC.joblib')

# Load the imputer and scaler
imputer = load('imputer_SC.joblib')
scaler = load('scaler_SC.joblib')

# Paths to the new test data files
early_blight_test_path = 'Extra/early_blight_normalized_sharpen_compressed_values (1).csv'
late_blight_test_path = 'Extra/late_blight_normalized_sharpen_compressed_values (1).csv'
healthy_test_path = 'Extra/healthy_normalized_sharpen_compressed_values.csv'

# Load the test data
early_blight_test_data = pd.read_csv(early_blight_test_path)
late_blight_test_data = pd.read_csv(late_blight_test_path)
healthy_test_data = pd.read_csv(healthy_test_path)

# Create labels: 0 for healthy, 1 for early blight, 2 for late blight
test_healthy_labels = [0] * len(healthy_test_data)
test_early_blight_labels = [1] * len(early_blight_test_data)
test_late_blight_labels = [2] * len(late_blight_test_data)
test_labels = test_healthy_labels + test_early_blight_labels + test_late_blight_labels

# Concatenate the new test data
test_features = pd.concat([healthy_test_data, early_blight_test_data, late_blight_test_data], ignore_index=True)

# Impute missing values using the loaded imputer
test_features_imputed = imputer.transform(test_features)

# Scale the test data using the loaded scaler
test_features_scaled = scaler.transform(test_features_imputed)

# Make predictions on the test data
test_predictions = model.predict(test_features_scaled)

# Create a DataFrame with the actual and predicted labels
results_df = pd.DataFrame({
    'Actual Label': test_labels,
    'Predicted Label': test_predictions
})

# Calculate the accuracy
test_accuracy = accuracy_score(test_labels, test_predictions)

# Print the comparison table
print(results_df)

# Print the accuracy
print("Test data accuracy:", test_accuracy)
```

```
     Actual Label  Predicted Label
0               0                1
1               0                1
2               0                1
3               0                0
4               0                1
..            ...              ...
305             2                1
306             2                1
307             2                1
308             2                1
309             2                1

[310 rows x 2 columns]
Test data accuracy: 0.3709677419354839
```

Flipped & Compressed

In [22]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from joblib import load

# Load the trained model
model = load('best_FC.joblib')

# Load the imputer and scaler
imputer = load('imputer_FC.joblib')
scaler = load('scaler_FC.joblib')

# Paths to the new test data files
early_blight_test_path = 'Extra/early_blight_64x64_compressed_flipped_normalized_test_values.csv'
late_blight_test_path = 'Extra/late_blight_64x64_compressed_flipped_normalized_test_values.csv'
healthy_test_path = 'Extra/healthy_64x64_compressed_flipped_normalized_test_values.csv'

# Load the test data
early_blight_test_data = pd.read_csv(early_blight_test_path)
late_blight_test_data = pd.read_csv(late_blight_test_path)
healthy_test_data = pd.read_csv(healthy_test_path)

# Create labels: 0 for healthy, 1 for early blight, 2 for late blight
test_healthy_labels = [0] * len(healthy_test_data)
test_early_blight_labels = [1] * len(early_blight_test_data)
test_late_blight_labels = [2] * len(late_blight_test_data)
test_labels = test_healthy_labels + test_early_blight_labels + test_late_blight_labels

# Concatenate the new test data
test_features = pd.concat([healthy_test_data, early_blight_test_data, late_blight_test_data], ignore_index=True)

# Impute missing values using the loaded imputer
test_features_imputed = imputer.transform(test_features)

# Scale the test data using the loaded scaler
test_features_scaled = scaler.transform(test_features_imputed)

# Make predictions on the test data
test_predictions = model.predict(test_features_scaled)

# Create a DataFrame with the actual and predicted labels
results_df = pd.DataFrame({
    'Actual Label': test_labels,
    'Predicted Label': test_predictions
})

# Calculate the accuracy
test_accuracy = accuracy_score(test_labels, test_predictions)

# Print the comparison table
print(results_df)

# Print the accuracy
print("Test data accuracy:", test_accuracy)
```

```
     Actual Label  Predicted Label
0               0                0
1               0                1
2               0                1
3               0                0
4               0                1
..            ...              ...
305             2                1
306             2                1
307             2                1
308             2                1
309             2                1

[310 rows x 2 columns]
Test data accuracy: 0.3741935483870968
```

Systematic Analysis of Model Performance on External Validation Set:

1. Discrepancy in Accuracy:

- The model, which demonstrated a commendable accuracy of approximately 90% in the training and testing phases, exhibits a significantly **lower accuracy of around 36-37% on the external validation set**. This substantial drop raises concerns about the model's generalization capability to unseen data.

2. Potential Causes for Low Accuracy:

a. Dataset Variability:

- The external validation set consists of 310 photographs of **Pakistani potatoes**, which may introduce **variations that were not adequately covered in the original training dataset (U.S. potato photographs)**. The model may have difficulty generalizing to these new instances.

b. Overfitting:

- The initial training might have led to **overfitting** on the training data, capturing noise or specific patterns that do not generalize well to unseen instances. This can result in a significant drop in performance on new, diverse data.

3. Future Modifications:

a. Dataset Expansion:

- Augment the training dataset with a more extensive and diverse set of images, ensuring it covers a broader range of variations present in real-world scenarios. This expansion can help the model generalize better to novel instances.

b. Regularization Techniques:

- Apply regularization techniques, such as dropout or weight regularization, during training to mitigate overfitting and encourage the model to learn more robust and generalizable features.

c. Hyperparameter Tuning:

- Systematically tune hyperparameters, including learning rate, batch size, and model architecture, to find configurations that better suit the characteristics of the external validation set.

d. Ensemble Methods:

- Consider employing ensemble methods, combining predictions from multiple models or model variations. Ensemble techniques often enhance

SVM Model Limitations

In the context of potato leaf disease recognition, while SVM offers significant advantages, it's important to discuss its limitations in a manner that complements its strengths.

1. Lack of complete hyperparameter selection

While the adaptability in selecting the appropriate regularization parameter (C) reduces overfitting risk, finding the optimal hyperparameters, including the kernel type and kernel parameters for SVM, can be challenging without comprehensive methods. This limitation is particularly pronounced in scenarios where **computational resources or time constraints prevent extensive hyperparameter tuning, potentially leading to suboptimal model performance**.

2.Finite Leaf Shape Modeling

The shape of potato leaves may vary depending on growth stage, variety, etc., and **has a large shape diversity**.**SVM uses linear decision boundaries** by default, which may **not** be able to **capture this complex nonlinear shape relationship**.This limitation suggests that while SVM can efficiently handle linearly separable problems, its performance might be constrained when dealing with the intricate patterns of leaf diseases that require the modeling of non-linear boundaries.

3.Limited perception of leaf texture

Potato leaves may have **unique texture features** such as speckles, color variations, and so on. Linear SVMs may not be effective in capturing non-linear, complex texture variations when linearly combining features.However, the strength of SVM in processing high-dimensional features comes with a reliance on effective feature engineering.**Using deep learning methods such as Convolutional Neural Networks (CNN) may be more suitable** for capturing texture information.

4.Computational complexity on large-scale datasets

The robustness and exceptional generalization ability of SVM come at the cost of increased computational complexity, especially for large-scale datasets. This limitation is crucial in potato leaf disease recognition, where the dataset size can grow significantly, leading to **longer training times and requiring more computational resources**. The computational demand becomes a bottleneck, particularly for applications requiring quick model updates or retraining.

Addressing these limitations requires a nuanced approach that might include combining SVM with other machine learning techniques, such as deep learning for feature extraction or ensemble methods to enhance model performance and overcome its inherent constraints. This balanced approach ensures that the advantages of SVM are leveraged while its limitations are carefully managed or mitigated.

In [ ]:

KNN

```
import pandas as pd
from google.colab import drive
import os
drive.mount('/content/drive')
os.chdir('/content/drive/MyDrive/243 Analytics Lab/Data') # customize this line to your working directory
```

```
Mounted at /content/drive
```

1. **Objectives of KNN model**: The primary objective of the KNN model is to classify new data points by considering the 'K' nearest neighbors in the training dataset[6]. It can be used for both classification and regression tasks[69]. In classification, an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its K nearest neighbors[6]. In regression, the output is the property value for the object, which is the average of the values of its K nearest neighbors[6].
2. **Models and methodologies applied**: The K-Nearest Neighbors (KNN) algorithm is a non-parametric supervised learning method[69]. It uses a distance metric (like Euclidean distance) to identify the 'K' closest training examples and makes predictions based on the majority vote (for classification) or average (for regression) of these neighbors[69].
3. **Functioning of KNN model**: The KNN algorithm works by finding the 'K' nearest neighbors to a given data point based on a distance metric[69]. The class or value of the data point is then determined by the majority vote or average of the K neighbors[69].
4. **Why KNN model is chosen**: The KNN model is chosen for its simplicity, ease of implementation, and versatility[1][3][26]. It doesn't make any assumptions about the underlying data distribution, making it a flexible choice for various types of datasets[26]. It can handle both numerical and categorical data[26].
5. **Other models considered**: Other models like Support Vector Machines (SVM), Decision Trees, Random Forests, and Neural Networks could be considered as alternatives to KNN. The choice depends on the specific requirements of the task, such as the size and nature of the dataset, the complexity of the problem, and computational resources[^30^][33].
6. **Selection of tuning parameters (hyperparameters)**: The selection of hyperparameters in KNN, such as the number of neighbors 'K', is crucial for the performance of the model[16][19]. Techniques like cross-validation, grid search, or random search are often used to find the optimal 'K' value[19].
7. **Performance measurement of KNN model**: The performance of a KNN model can be measured using various metrics such as accuracy, precision, recall, F1 score, etc[12][16]. The choice of metric depends on the problem at hand and the balance needed between true positive rate (sensitivity) and false positive rate (specificity).
8. **Limitations of KNN model**: Some of the limitations of the KNN model include its sensitivity to the scale of features, the curse of dimensionality for high-dimensional data, its computational expense for large datasets, and its sensitivity to outliers[26][27][28][29].

Source: Conversation with Bing, 2024/2/16
(1) k-nearest neighbors algorithm - Wikipedia. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.
(2) K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks. https://www.geeksforgeeks.org/k-nearest-neighbours/.
(3) What is the k-nearest neighbors algorithm? | IBM. https://www.ibm.com/topics/knn.
(4) Pros and Cons of K-Nearest Neighbors - From The GENESIS. https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/.
(5) classification - Alternative to KNN - Cross Validated. https://stats.stackexchange.com/questions/605279/alternative-to-knn.
(6) python - finding KNN for larger Dataset - Stack Overflow. https://stackoverflow.com/questions/66347671/finding-knn-for-larger-dataset. (7) K-nearest Neighbors (KNN) Classification Model. https://www.ritchieng.com/machine-learning-k-nearest-neighbors-knn/.
(8) KNN Hyperparameters: A Friendly Guide to Optimization. https://www.programmingr.com/knn-hyperparameters-a-friendly-guide-to-optimization/. (9) Understanding K-Nearest Neighbors: A Simple Approach to.... https://towardsai.net/p/machine-learning/understanding-k-nearest-neighbors-a-simple-approach-to-classification-and-regression.
(10) k-NN (k-Nearest Neighbors) Starter Guide - Machine Learning HD. https://machinelearninghd.com/k-nn-k-nearest-neighbors-starter-guide/.
(11) Chapter 1: K Nearest Neighbors (Supervised Machine Learning ... - Medium. https://medium.com/machine-learning-community/chapter-1-k-nearest-neighbors-supervised-machine-learning-algorithm-bd234fe6029c.
(12) KNN Model-Based Approach in Classification | SpringerLink. https://link.springer.com/chapter/10.1007/978-3-540-39964-3_62.
(13) . https://metametamedieval.com/2018/03/03/medievalising-mla2019-medieval-medieval-friendly-medievalist-medievalistically-interesting-and-potentially-medievalisable-calls-for-papers/.
(14) . https://www.chegg.com/homework-help/questions-and-answers/8-key-component-scientific-theory-mathematical-model-b-statistical-analysis-showing-model--q28146448.
(15) . https://www.chegg.com/homework-help/questions-and-answers/problem-1-23-points-consider-geometric-simplifications-leveraged-approach-navier-stokes-eq-q106347373.
(16) . https://www.chegg.com/homework-help/questions-and-answers/assume-normality-error-term-multicollinearity-care-model-multicollinearity-explain-test-mu-q18610434.
(17) Chegg. https://www.chegg.com/homework-help/questions-and-answers/study-conducted-effects-developer-strength-factor-development-time-factor-b-density-photog-q25785630.

(18) How does K-nearest Neighbor Works in Machine Learning | KNN algorithm. https://www.analyticssteps.com/blogs/how-does-k-nearest-neighbor-works-machine-learning-classification-problem. (19) K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks. https://bing.com/search?q=How+does+KNN+model+function.
(20) K-Nearest Neighbors (KNN) Classification with R Tutorial. https://www.datacamp.com/tutorial/k-nearest-neighbors-knn-classification-with-r-tutorial. (21) KNN Algorithm – K-Nearest Neighbors Classifiers and Model Example. https://www.freecodecamp.org/news/k-nearest-neighbors-algorithm-classifiers-and-model-example/.
(22) Latest Guide to K-Nearest Neighbors (KNN) Algorithm in 2024. https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/.
(23) Evaluation of k-nearest neighbour classifier performance for .... https://link.springer.com/article/10.1007/s42452-019-1356-9.
(24) How to tune the K-Nearest Neighbors classifier with Scikit ... - DataSklr. https://www.datasklr.com/select-classification-methods/k-nearest-neighbors. (25) sklearn.neighbors.KNeighborsClassifier - scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.
(26) An improved KNN classifier based on a novel weighted voting ... - Springer. https://link.springer.com/article/10.1007/s00521-023-09272-8.
(27) How to Find The Optimal Value of K in KNN - GeeksforGeeks. https://www.geeksforgeeks.org/how-to-find-the-optimal-value-of-k-in-knn/.
(28) K-Nearest Neighbor(KNN) Algorithm for Machine Learning. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning. (29) K-Nearest Neighbor. A complete explanation of K-NN - Medium. https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4.
(30) Most Popular Distance Metrics Used in KNN and When to Use Them. https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html.
(31) KNN - The Distance Based Machine Learning Algorithm - Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/05/knn-the-distance-based-machine-learning-algorithm/.
(32) Under the Hood of K-Nearest Neighbors (KNN) and Popular Model ... - Medium. https://medium.com/swlh/under-the-hood-of-k-nearest-neighbors-knn-and-popular-model-validation-techniques-85ab0964d563.
(33) KNN classification with categorical data - Stack Overflow. https://stackoverflow.com/questions/13625849/knn-classification-with-categorical-data.
(34) undefined. https://github.com/Amitg4/KNN_ModelValidation.
(35) undefined. https://github.com/ritchieng/ritchieng.github.io.

Load data

```
healthy_compressed_flipped_normalized_pixel_values = pd.read_csv('healthy_64x64_compressed_flipped_normalized_pixel_values.c
healthy_normalized_compressed_values = pd.read_csv('healthy_normalized_compressed_values.csv')
healthy_normalized_sharpen_compressed_values = pd.read_csv('healthy_normalized_sharpen_compressed_values.csv')
early_blight_normalized_compressed_values = pd.read_csv('early_blight_normalized_compressed_values.csv')
late_blight_normalized_compressed_values = pd.read_csv('late_blight_normalized_compressed_values.csv')
```

```
frames = [healthy_normalized_compressed_values, healthy_compressed_flipped_normalized_pixel_values, healthy_normalized_sharp
Potato_leaf = pd.concat(frames)
Potato_leaf.reset_index(drop=True, inplace=True)
Potato_leaf
```

	0	1	2	3	4	5	6	7	
0	0.556863	0.474510	0.501961	0.584314	0.501961	0.529412	0.580392	0.498039	0.5
1	0.607843	0.529412	0.564706	0.600000	0.521569	0.556863	0.619608	0.541176	0.5
2	0.690196	0.627451	0.666667	0.654902	0.592157	0.631373	0.678431	0.615686	0.6
3	0.450980	0.376471	0.400000	0.470588	0.396078	0.419608	0.470588	0.396078	0.4
4	0.517647	0.470588	0.486275	0.533333	0.486275	0.501961	0.517647	0.470588	0.4
...	...	...	...	...	...	...	...	...	
2703	0.552941	0.431373	0.423529	0.541176	0.419608	0.411765	0.568627	0.447059	0.4
2704	0.662745	0.635294	0.662745	0.658824	0.631373	0.658824	0.666667	0.639216	0.6
2705	0.517647	0.470588	0.486275	0.517647	0.470588	0.486275	0.537255	0.490196	0.5
2706	0.521569	0.478431	0.494118	0.517647	0.474510	0.490196	0.533333	0.490196	0.5
2707	0.439216	0.388235	0.423529	0.419608	0.368627	0.403922	0.427451	0.376471	0.4

2708 rows × 12288 columns

```
# Add labels based on row indices
Potato_leaf.loc[:707, 'Label'] = 'healthy'
Potato_leaf.loc[708:1707, 'Label'] = 'early blight'
Potato_leaf.loc[1708:, 'Label'] = 'late blight'
Potato_leaf
```

	0	1	2	3	4	5	6	7	
0	0.556863	0.474510	0.501961	0.584314	0.501961	0.529412	0.580392	0.498039	0.5
1	0.607843	0.529412	0.564706	0.600000	0.521569	0.556863	0.619608	0.541176	0.5
2	0.690196	0.627451	0.666667	0.654902	0.592157	0.631373	0.678431	0.615686	0.6
3	0.450980	0.376471	0.400000	0.470588	0.396078	0.419608	0.470588	0.396078	0.4
4	0.517647	0.470588	0.486275	0.533333	0.486275	0.501961	0.517647	0.470588	0.4
...	...	...	...	...	...	...	...	...	
2703	0.552941	0.431373	0.423529	0.541176	0.419608	0.411765	0.568627	0.447059	0.4
2704	0.662745	0.635294	0.662745	0.658824	0.631373	0.658824	0.666667	0.639216	0.6
2705	0.517647	0.470588	0.486275	0.517647	0.470588	0.486275	0.537255	0.490196	0.5
2706	0.521569	0.478431	0.494118	0.517647	0.474510	0.490196	0.533333	0.490196	0.5

```
Potato_leaf.dropna(inplace=True)
Potato_leaf.reset_index(drop=True, inplace=True)
Potato_leaf
```

	0	1	2	3	4	5	6	7	
0	0.556863	0.474510	0.501961	0.584314	0.501961	0.529412	0.580392	0.498039	0.5
1	0.607843	0.529412	0.564706	0.600000	0.521569	0.556863	0.619608	0.541176	0.5
2	0.690196	0.627451	0.666667	0.654902	0.592157	0.631373	0.678431	0.615686	0.6
3	0.450980	0.376471	0.400000	0.470588	0.396078	0.419608	0.470588	0.396078	0.4
4	0.517647	0.470588	0.486275	0.533333	0.486275	0.501961	0.517647	0.470588	0.4
...	...	...	...	...	...	...	...	...	
2535	0.552941	0.431373	0.423529	0.541176	0.419608	0.411765	0.568627	0.447059	0.4
2536	0.662745	0.635294	0.662745	0.658824	0.631373	0.658824	0.666667	0.639216	0.6
2537	0.517647	0.470588	0.486275	0.517647	0.470588	0.486275	0.537255	0.490196	0.5
2538	0.521569	0.478431	0.494118	0.517647	0.474510	0.490196	0.533333	0.490196	0.5
2539	0.439216	0.388235	0.423529	0.419608	0.368627	0.403922	0.427451	0.376471	0.4

2540 rows × 12289 columns

Model

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
# Assuming you have a combined dataframe 'Potato_leaf' with features and labels
# Features (X) are the columns representing attributes of the potato leaves
# Labels (y) are the target variable (healthy or diseased)

# Split the data into features (X) and labels (y)
X = Potato_leaf.drop(columns=['Label'])  # Adjust column names as needed
y = Potato_leaf['Label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the KNN model (you can choose the value of k)
knn_model = KNeighborsClassifier(n_neighbors=5)  # Example: k=5

# Train the model
knn_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_str)
```

```
Accuracy: 0.74
Classification Report:
                precision    recall  f1-score   support

early blight       0.99      0.61      0.76       192
     healthy       0.77      0.56      0.65       114
 late blight       0.64      0.97      0.77       202

    accuracy                           0.74       508
   macro avg       0.80      0.72      0.73       508
weighted avg       0.80      0.74      0.74       508
```

model's performance:

Accuracy: The overall accuracy of the model is 0.74, which means it correctly predicts the class labels for approximately 74% of the samples.

Precision:

Early Blight: The precision for this class is 0.99, indicating that when the model predicts "early blight," it is highly likely to be correct.

Healthy: The precision for the "healthy" class is 0.77, meaning that the model's predictions for "healthy" are reasonably accurate.

Late Blight: The precision for "late blight" is 0.64, suggesting that some predictions for this class may include false positives.

Recall:

Early Blight: The recall (true positive rate) for "early blight" is 0.61, indicating that the model captures 61% of actual "early blight" cases.

Healthy: The recall for "healthy" is 0.56, meaning that the model identifies 56% of actual "healthy" samples.

Late Blight: The recall for "late blight" is 0.97, implying that the model effectively detects most instances of "late blight."

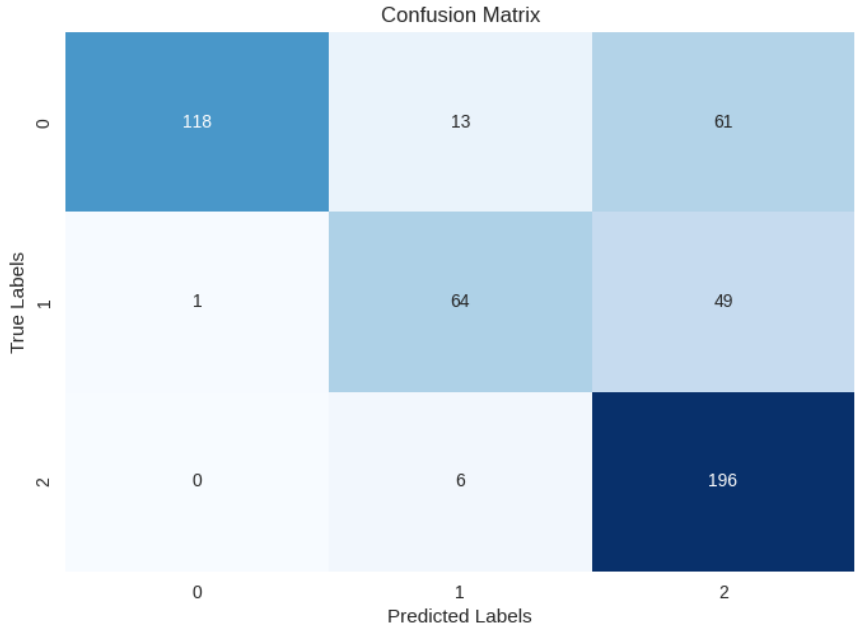
F1-Score: The F1-score balances precision and recall. For "early blight," it is 0.76; for "healthy," it is 0.65; and for "late blight," it is 0.77.

Macro Avg F1-Score: The average F1-score across all classes is 0.73.

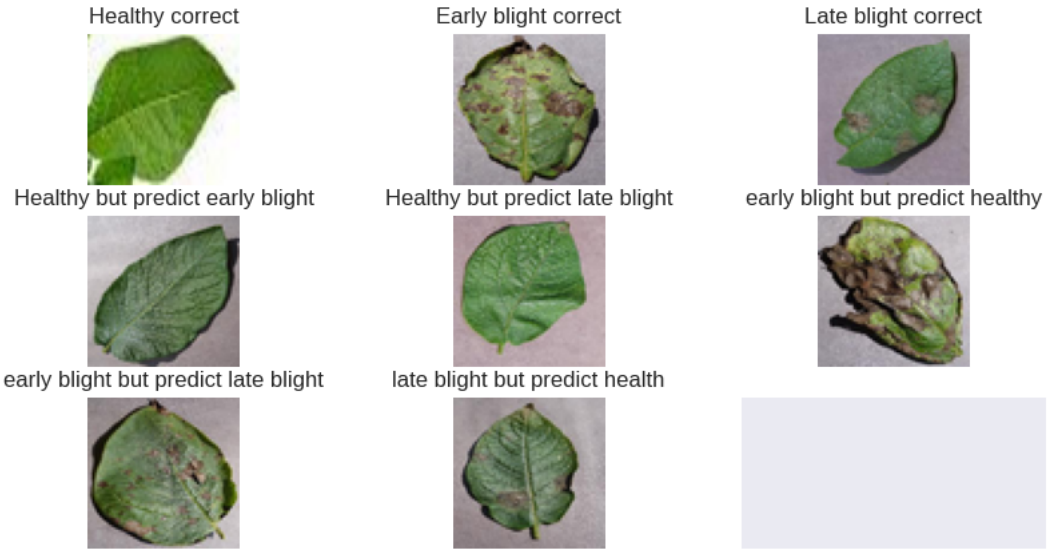
Weighted Avg F1-Score: The weighted average F1-score, considering class imbalance, is 0.74.

In summary, the model performs well in identifying "early blight" and "late blight," but there's room for improvement in predicting "healthy" samples. Further fine-tuning or exploring different algorithms could enhance its performance.

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```




```
# Display the image
test_np = X_test.to_numpy()

Potato_healthy_correct = test_np[12, ].reshape(64,64,3)
Potato_Early_blight_correct = test_np[506, ].reshape(64,64,3)
Potato_Late_blight_correct = test_np[2, ].reshape(64,64,3)

Potato_h_l_wrong = test_np[1, ].reshape(64,64,3) #
Potato_h_e_wrong = test_np[308, ].reshape(64,64,3) #
Potato_e_l_wrong = test_np[3, ].reshape(64,64,3) #
Potato_e_h_wrong = test_np[26, ].reshape(64,64,3) #

Potato_l_h_wrong = test_np[119, ].reshape(64,64,3) #



# create a figure with six subplots
fig, axs = plt.subplots(3, 3, figsize=(10, 5))

# display each image on a subplot
axs[0, 0].imshow(Potato_healthy_correct)
axs[0, 1].imshow(Potato_Early_blight_correct)
axs[0, 2].imshow(Potato_Late_blight_correct)
axs[1, 0].imshow(Potato_h_e_wrong)
axs[1, 1].imshow(Potato_h_l_wrong)
axs[1, 2].imshow(Potato_e_l_wrong)
axs[2, 0].imshow(Potato_e_h_wrong)
axs[2, 1].imshow(Potato_l_h_wrong)



# add titles to each subplot
axs[0, 0].set_title('Healthy correct')
axs[0, 1].set_title('Early blight correct')
axs[0, 2].set_title('Late blight correct')
axs[1, 0].set_title('Healthy but predict early blight')
axs[1, 1].set_title('Healthy but predict late blight')
axs[1, 2].set_title('early blight but predict healthy')
axs[2, 0].set_title('early blight but predict late blight')
axs[2, 1].set_title('late blight but predict health')



# remove the x and y ticks from each subplot
for ax in axs.flat:
    ax.set_xticks([])
    ax.set_yticks([])

# display the figure
plt.show()
```




```
y_test.iloc[2]
y_pred[2]
```

```
'late blight'
```

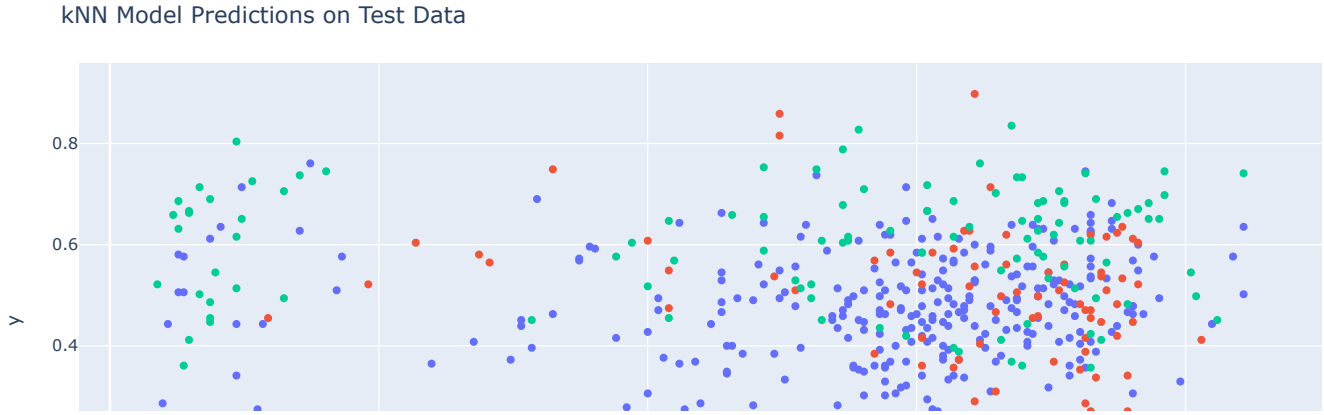
```
print(y_test.iloc[308], y_test.iloc[120], y_test.iloc[121], y_test.iloc[123], y_test.iloc[69], y_test.iloc[72], y_test.iloc[8
print('-------------------------')
print(y_pred[462], y_pred[468], y_pred[478], y_pred[484], y_pred[489], y_pred[501], y_pred[507], y_pred[455])
```

```
healthy early blight early blight early blight healthy healthy healthy early blight early blight early blight early blig
-------------------------
late blight healthy late blight healthy late blight late blight late blight late blight
```

```
y_pred
```

```
 'late blight', 'early blight', 'late blight', 'late blight',
'healthy', 'late blight', 'healthy', 'early blight',
'early blight', 'late blight', 'healthy', 'late blight',
'late blight', 'late blight', 'late blight', 'late blight',
'late blight', 'early blight', 'early blight', 'late blight',
'late blight', 'healthy', 'healthy', 'early blight',
'early blight', 'late blight', 'late blight', 'late blight',
'early blight', 'early blight', 'late blight', 'late blight',
'early blight', 'early blight', 'early blight', 'late blight',
'healthy', 'late blight', 'late blight', 'late blight',
'early blight', 'early blight', 'early blight', 'healthy',
'early blight', 'late blight'], dtype=object)
```

```
import plotly.express as px


# Create a scatter plot showing actual vs. predicted labels
fig = px.scatter(x=X_test['4596'], y=X_test['8692'], color=y_pred,
                 labels={'color': 'Predicted Label'})
fig.update_layout(title='kNN Model Predictions on Test Data')
fig.show()
```



CNN

Objectives: The primary objective of the CNN model is to classify leaf images into one of three categories: ‘healthy’, ‘early blight’, and ‘late blight’. This is a multi-class classification problem.

Models and Methodologies: The model used is a Convolutional Neural Network (CNN), a type of deep learning model particularly effective for image classification tasks. The model consists of two convolutional layers, each followed by a ReLU activation function and a max pooling layer. After the convolutional layers, there is a dropout layer, a fully connected layer, another dropout layer, and finally an output layer.

Functioning of the Models: The CNN model works by applying a series of filters to the input images in the convolutional layers. These filters help the model to learn local patterns in the images. The max pooling layers then reduce the spatial dimensions of the data, keeping only the most important information. The dropout layers randomly set a fraction of input units to 0 at each update during training time, which helps prevent overfitting. The fully connected layer at the end makes the final prediction based on the features extracted by the previous layers.

Model Selection: CNNs were chosen because they are state-of-the-art for image classification tasks. They have the ability to automatically and adaptively learn spatial hierarchies of features, which makes them highly suitable for analyzing images.

Consideration of Other Models: Other models such as Support Vector Machines (SVMs) or Random Forests could have been considered. However, these traditional machine learning models often require manual feature extraction from images, which can be complex and time-consuming. On the other hand, CNNs can learn these features automatically, which is a major advantage.

Hyperparameter Selection: The learning rate for the Adam optimizer were set to 0.001. These values are commonly used in practice. The number of epochs was set to 25, which means the entire dataset is passed forward and backward through the CNN 25 times. The batch size was set to 64, which is the number of training examples used in one iteration.

Model Performance: The model’s performance was measured using the cross-entropy loss function, which is standard for multi-class classification problems. The accuracy of the model was also calculated for both the training and test datasets. The accuracy is the proportion of correct predictions out of total predictions.

Method Limitations: The model might not perform well if the new data it sees is very different from the training data. For example, if the leaves in the images are of a different size or color than the ones in the training data, the model might make incorrect predictions. Additionally, the model currently does not have any mechanisms to handle class imbalance. If some classes

have many more examples than others, the model might become biased towards the majority class.

```
import pandas as pd
from google.colab import drive
import os
drive.mount('/content/drive')
os.chdir('/content/drive/MyDrive/243 Analytics Lab/Data') # customize this line t
```

```
Mounted at /content/drive
```

Data Processing

```
# Load dataset
healthy_compressed_flipped_normalized_pixel_values = pd.read_csv('healthy_64x64_c
healthy_normalized_compressed_values = pd.read_csv('healthy_normalized_compressed
healthy_normalized_sharpen_compressed_values = pd.read_csv('healthy_normalized_sh
early_blight_normalized_compressed_values = pd.read_csv('early_blight_normalized_
late_blight_normalized_compressed_values = pd.read_csv('late_blight_normalized_co

healthy_compressed_flipped_normalized_pixel_values['Label'] = 'healthy'
healthy_normalized_compressed_values['Label'] = 'healthy'
healthy_normalized_sharpen_compressed_values['Label'] = 'healthy'
early_blight_normalized_compressed_values['Label'] = 'early blight'
late_blight_normalized_compressed_values['Label'] = 'late blight'


# Combine dataset
frames = [healthy_normalized_compressed_values, healthy_compressed_flipped_normal
Potato_leaf = pd.concat(frames)
Potato_leaf.reset_index(drop=True, inplace=True)

# Remove not 1x1 size image
Potato_leaf.dropna(inplace=True)
Potato_leaf.reset_index(drop=True, inplace=True)

Potato_leaf
```

	0	1	2	3	4	5	6	7	
0	0.556863	0.474510	0.501961	0.584314	0.501961	0.529412	0.580392	0.498039	0.5
1	0.607843	0.529412	0.564706	0.600000	0.521569	0.556863	0.619608	0.541176	0.5
2	0.690196	0.627451	0.666667	0.654902	0.592157	0.631373	0.678431	0.615686	0.6
3	0.450980	0.376471	0.400000	0.470588	0.396078	0.419608	0.470588	0.396078	0.4
4	0.517647	0.470588	0.486275	0.533333	0.486275	0.501961	0.517647	0.470588	0.4
...	...	...	...	...	...	...	...	...	
2535	0.552941	0.431373	0.423529	0.541176	0.419608	0.411765	0.568627	0.447059	0.4
2536	0.662745	0.635294	0.662745	0.658824	0.631373	0.658824	0.666667	0.639216	0.6
2537	0.517647	0.470588	0.486275	0.517647	0.470588	0.486275	0.537255	0.490196	0.5
2538	0.521569	0.478431	0.494118	0.517647	0.474510	0.490196	0.533333	0.490196	0.5
2539	0.439216	0.388235	0.423529	0.419608	0.368627	0.403922	0.427451	0.376471	0.4

2540 rows × 12289 columns

```
print(Potato_leaf['Label'].value_counts())
print(Potato_leaf.dtypes)
```

```
early blight    1000
late blight     1000
healthy          540
Name: Label, dtype: int64
0        float64
1        float64
2        float64
3        float64
4        float64
          ...
12284    float64
12285    float64
12286    float64
12287    float64
Label     object
Length: 12289, dtype: object
```

```
Potato_leaf['Label'] = Potato_leaf['Label'].astype('category')

from sklearn.preprocessing import LabelEncoder

# assuming df is your DataFrame and "Label" is the column with the categories
le = LabelEncoder()
Potato_leaf['Label'] = le.fit_transform(Potato_leaf['Label'])
healthy_label = Potato_leaf.iloc[0, -1]
early_label = Potato_leaf.iloc[1000, -1]
late_label = Potato_leaf.iloc[2000, -1]
Potato_leaf
```

	0	1	2	3	4	5	6	7	
0	0.556863	0.474510	0.501961	0.584314	0.501961	0.529412	0.580392	0.498039	0.5
1	0.607843	0.529412	0.564706	0.600000	0.521569	0.556863	0.619608	0.541176	0.5
2	0.690196	0.627451	0.666667	0.654902	0.592157	0.631373	0.678431	0.615686	0.6
3	0.450980	0.376471	0.400000	0.470588	0.396078	0.419608	0.470588	0.396078	0.4
4	0.517647	0.470588	0.486275	0.533333	0.486275	0.501961	0.517647	0.470588	0.4
...	...	...	...	...	...	...	...	...	
2535	0.552941	0.431373	0.423529	0.541176	0.419608	0.411765	0.568627	0.447059	0.4
2536	0.662745	0.635294	0.662745	0.658824	0.631373	0.658824	0.666667	0.639216	0.6
2537	0.517647	0.470588	0.486275	0.517647	0.470588	0.486275	0.537255	0.490196	0.5
2538	0.521569	0.478431	0.494118	0.517647	0.474510	0.490196	0.533333	0.490196	0.5
2539	0.439216	0.388235	0.423529	0.419608	0.368627	0.403922	0.427451	0.376471	0.4

2540 rows × 12289 columns

CNN

create functions

```
import time
import numpy as np
import torch.optim as optim

def train_model(model, criterion, num_epochs, optimizer, train_loader, test_loade
    train_loss_history = []
    train_acc_history = []
    test_loss_history = []
    test_acc_history = []
    for epoch in range(num_epochs):
        train_loss = 0.0
        train_correct = 0
        model.train()  # Set the model to training mode
        for inputs, labels in train_loader:
            # Move the inputs to the device
            inputs, labels = inputs.to(device), labels.to(device)
            # Zero the parameter gradients
            optimizer.zero_grad()
            # Forward pass
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            loss = criterion(outputs, labels)
            # Backward pass and optimize
            loss.backward()
            optimizer.step()
            # Update loss
            train_loss += loss.item() * inputs.size(0)
            # Update accuracy
            train_correct += torch.sum(preds == labels.data)

        # Calculate average loss and accuracy
        train_loss = train_loss / len(train_loader.dataset)
        train_acc = train_correct.double() / len(train_loader.dataset)

        # Evaluate on the test data
        model.eval()
        test_loss = 0.0
        test_correct = 0
        with torch.no_grad():
            for inputs, labels in test_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)
                test_loss += loss.item() * inputs.size(0)
                test_correct += torch.sum(preds == labels.data)

        test_loss = test_loss / len(test_loader.dataset)
        test_acc = test_correct.double() / len(test_loader.dataset)

        print(f'Epoch {epoch+1}/{num_epochs}, Training Loss: {train_loss:.4f}, Te

        train_loss_history.append(train_loss)
        train_acc_history.append(train_acc)
        test_loss_history.append(test_loss)
```

```
        test_acc_history.append(test_acc)

    return model, train_loss_history, train_acc_history, test_loss_history, test_
```

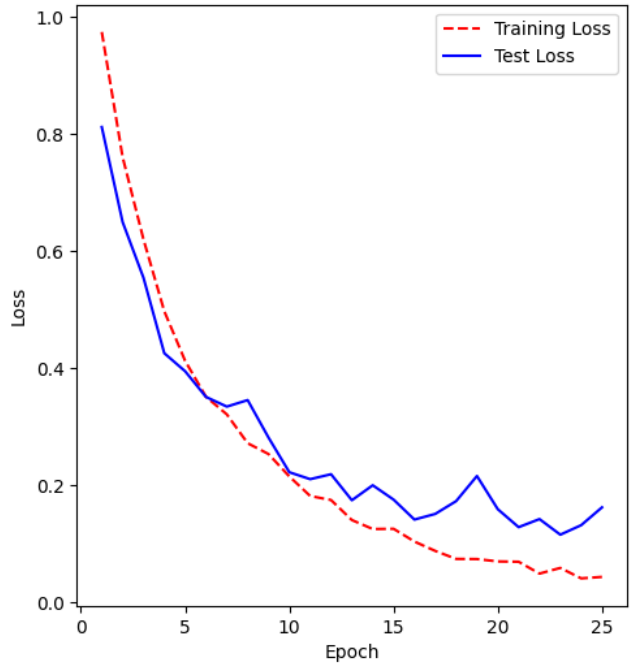
```
from torch.utils.data import Dataset

class LeafDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.data = dataframe.iloc[:, :-1]
        self.targets = dataframe.iloc[:, -1]
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sample = self.data.iloc[idx].values.reshape(3, 64, 64)  # reshape to matc
        sample = np.transpose(sample, (1, 2, 0))  # transpose to [height, width,
        sample = sample.astype(np.float32)  # convert to float32
        target = torch.tensor(self.targets.iloc[idx], dtype=torch.long)  # conver

        if self.transform:
            sample = self.transform(sample)

        return sample, target


import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

from sklearn.model_selection import train_test_split

# Split the data into training, validation, and test sets
Potato_leaftrain, Potato_leaftest = train_test_split(Potato_leaf, test_size=0.3,

# Set the random seed for reproducibility
torch.manual_seed(12345)

# Define a transform to preprocess the data
transform = transforms.Compose([
    transforms.ToTensor(), # Convert images to PyTorch tensors
    transforms.Normalize((0.5,), (0.5,))  # shit your data range to [-1,1]
])

train_dataset = LeafDataset(Potato_leaftrain, transform=transform)
test_dataset = LeafDataset(Potato_leaftest, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

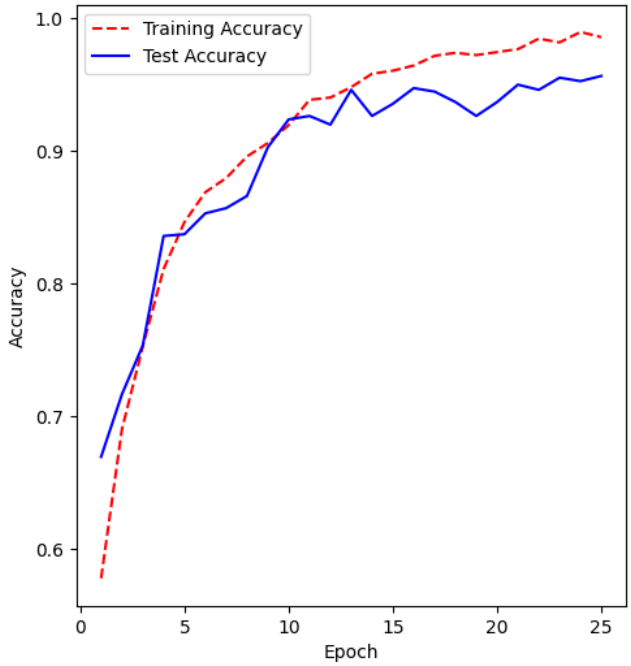
```
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # First convolutional layer
        # Input channels = 3 (RGB), output channels = 32
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        # Second convolutional layer
        # Input channels = 32, output channels = 64
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        # Max pooling layer
        self.pool = nn.MaxPool2d(2, 2)
        # Dropout layer with p=0.25
        self.dropout1 = nn.Dropout2d(0.25)
        # Fully connected layer
        # Input size needs to match the output of the conv layers
        self.fc1 = nn.Linear(64 * 16 * 16, 128)
        # Another dropout layer with p=0.5
        self.dropout2 = nn.Dropout(0.5)
        # Output layer
        self.fc2 = nn.Linear(128, 3)

    def forward(self, x):
        # Apply first conv layer, followed by ReLU, then max pooling
        x = self.pool(F.relu(self.conv1(x)))
        # Apply second conv layer, followed by ReLU, then max pooling
        x = self.pool(F.relu(self.conv2(x)))
        # Apply first dropout layer
        x = self.dropout1(x)
        # Flatten the tensor
        x = x.view(-1, 64 * 16 * 16)
        # Apply fully connected layer with ReLU
        x = F.relu(self.fc1(x))
        # Apply second dropout layer
        x = self.dropout2(x)
        # Final output layer
        x = self.fc2(x)
        return x


device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")  # Use GP
model = CNN().to(device)  # Move the model to the device
criterion = nn.CrossEntropyLoss()  # Define the loss function
optimizer = optim.Adam(model.parameters(), lr=0.001)  # Define the optimizer
num_epochs = 25  # Define the number of epochs

# Train the model
model, train_loss_history, train_acc_history, test_loss_history, test_acc_history
```

```
Epoch 1/25, Training Loss: 0.9735, Test Loss: 0.8113, Accuracy: 0.6693
Epoch 2/25, Training Loss: 0.7600, Test Loss: 0.6491, Accuracy: 0.7165
Epoch 3/25, Training Loss: 0.6197, Test Loss: 0.5542, Accuracy: 0.7533
Epoch 4/25, Training Loss: 0.4968, Test Loss: 0.4252, Accuracy: 0.8360
```

```
Epoch 5/25, Training Loss: 0.4121, Test Loss: 0.3944, Accuracy: 0.8373
Epoch 6/25, Training Loss: 0.3509, Test Loss: 0.3506, Accuracy: 0.8530
Epoch 7/25, Training Loss: 0.3208, Test Loss: 0.3341, Accuracy: 0.8570
Epoch 8/25, Training Loss: 0.2716, Test Loss: 0.3452, Accuracy: 0.8661
Epoch 9/25, Training Loss: 0.2531, Test Loss: 0.2813, Accuracy: 0.9029
Epoch 10/25, Training Loss: 0.2149, Test Loss: 0.2221, Accuracy: 0.9239
Epoch 11/25, Training Loss: 0.1815, Test Loss: 0.2103, Accuracy: 0.9265
Epoch 12/25, Training Loss: 0.1747, Test Loss: 0.2187, Accuracy: 0.9199
Epoch 13/25, Training Loss: 0.1405, Test Loss: 0.1743, Accuracy: 0.9462
Epoch 14/25, Training Loss: 0.1252, Test Loss: 0.1999, Accuracy: 0.9265
Epoch 15/25, Training Loss: 0.1256, Test Loss: 0.1754, Accuracy: 0.9357
Epoch 16/25, Training Loss: 0.1038, Test Loss: 0.1413, Accuracy: 0.9475
Epoch 17/25, Training Loss: 0.0878, Test Loss: 0.1510, Accuracy: 0.9449
Epoch 18/25, Training Loss: 0.0742, Test Loss: 0.1728, Accuracy: 0.9370
Epoch 19/25, Training Loss: 0.0739, Test Loss: 0.2157, Accuracy: 0.9265
Epoch 20/25, Training Loss: 0.0699, Test Loss: 0.1590, Accuracy: 0.9370
Epoch 21/25, Training Loss: 0.0693, Test Loss: 0.1284, Accuracy: 0.9501
Epoch 22/25, Training Loss: 0.0491, Test Loss: 0.1422, Accuracy: 0.9462
Epoch 23/25, Training Loss: 0.0588, Test Loss: 0.1155, Accuracy: 0.9554
Epoch 24/25, Training Loss: 0.0410, Test Loss: 0.1317, Accuracy: 0.9528
Epoch 25/25, Training Loss: 0.0435, Test Loss: 0.1622, Accuracy: 0.9567
```

```
import matplotlib.pyplot as plt


# Create count of the number of epochs
epoch_count = range(1, num_epochs + 1)

# Visualize loss history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_count, train_loss_history, 'r--')
plt.plot(epoch_count, test_loss_history, 'b-')
plt.legend(['Training Loss', 'Test Loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')

# Visualize accuracy history
plt.subplot(1, 2, 2)
plt.plot(epoch_count, train_acc_history, 'r--')
plt.plot(epoch_count, test_acc_history, 'b-')
plt.legend(['Training Accuracy', 'Test Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')

plt.show()
```






```
from sklearn.metrics import confusion_matrix

# Initialize the prediction and label lists(tensors)
predlist=torch.zeros(0,dtype=torch.long, device='cpu')
lbllist=torch.zeros(0,dtype=torch.long, device='cpu')

with torch.no_grad():
    for i, (inputs, classes) in enumerate(test_loader):
        inputs = inputs.to(device)
        classes = classes.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)

        # Append batch prediction results
        predlist=torch.cat([predlist,preds.view(-1).cpu()])
        lbllist=torch.cat([lbllist,classes.view(-1).cpu()])

# Confusion matrix
conf_mat=confusion_matrix(lbllist.numpy(), predlist.numpy())
print('Confusion Matrix')
print('-'*16)
print(conf_mat)
```

```
    Confusion Matrix
    ----------------
```

```
[[275   0  13]
 [  1 156  10]
 [  8   1 298]]
```

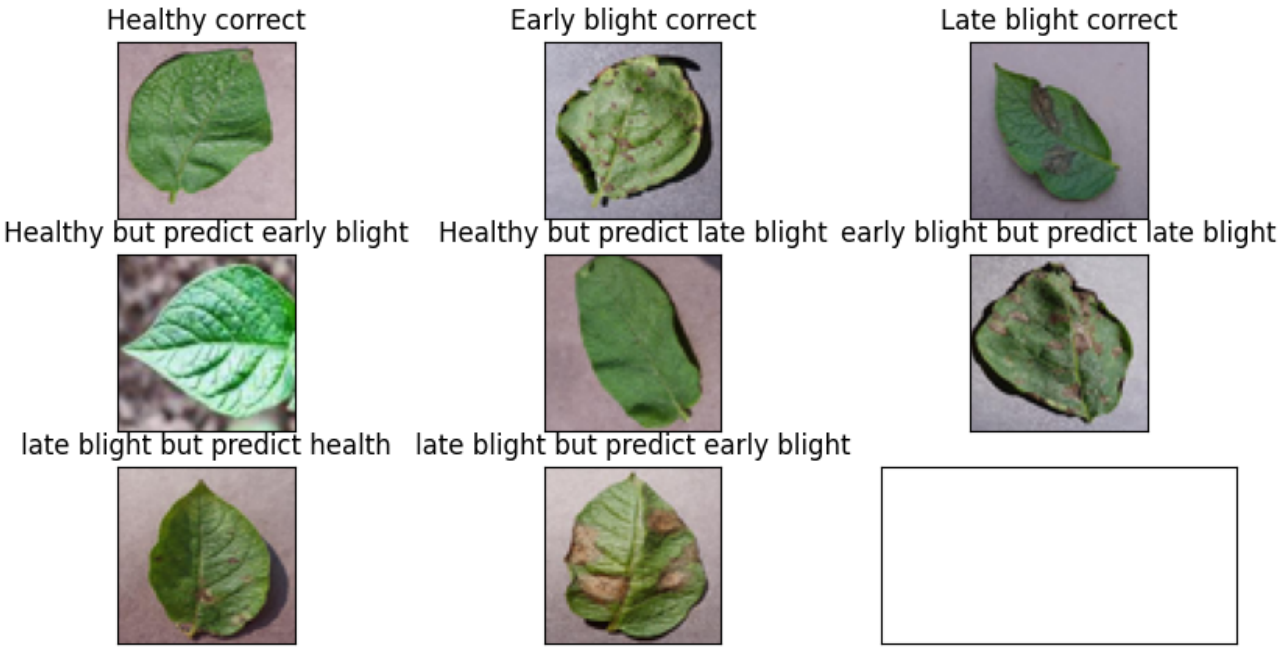
```
# Get the predicted labels
pred_labels = predlist.numpy()

# Get the actual labels
actual_labels = Potato_leaftest.iloc[:, -1].to_numpy()

misclassified_idx = np.where(pred_labels != actual_labels)[0]
print("Misclassified indices: ", misclassified_idx)
```

```
Misclassified indices:  [ 17  25  34  48  67  69  94 110 117 121 203 212 234 2
 392 400 429 446 480 517 520 565 586 619 647 688 702 715 758]
```

```
print(pred_labels[misclassified_idx])
print(actual_labels[misclassified_idx])
print(misclassified_idx)
```

```
[0 2 2 2 2 2 2 2 0 2 0 2 2 2 0 2 2 2 0 2 2 2 2 2 0 2 0 2 2 0 2 1 0]
[2 0 0 1 0 1 0 0 2 0 2 0 1 1 2 1 1 1 2 0 0 1 0 0 2 0 1 1 0 2 1 2 2]
[ 17  25  34  48  67  69  94 110 117 121 203 212 234 258 266 282 323 378
 392 400 429 446 480 517 520 565 586 619 647 688 702 715 758]
```

```
# Display the image
test_np = Potato_leaftest.iloc[:, :-1].to_numpy()
Potato_healthy_correct = test_np[1, ].reshape(64,64,3)
Potato_Early_blight_correct = test_np[4, ].reshape(64,64,3)
Potato_Late_blight_correct = test_np[760, ].reshape(64,64,3)

Potato_h_l_wrong = test_np[48, ].reshape(64,64,3)
Potato_h_e_wrong = test_np[586, ].reshape(64,64,3)
Potato_e_l_wrong = test_np[25, ].reshape(64,64,3)
Potato_l_e_wrong = test_np[17, ].reshape(64,64,3)
Potato_l_h_wrong = test_np[715, ].reshape(64,64,3)



# create a figure with six subplots
fig, axs = plt.subplots(3, 3, figsize=(10, 5))

# display each image on a subplot
axs[0, 0].imshow(Potato_healthy_correct)
axs[0, 1].imshow(Potato_Early_blight_correct)
axs[0, 2].imshow(Potato_Late_blight_correct)
axs[1, 0].imshow(Potato_h_e_wrong)
axs[1, 1].imshow(Potato_h_l_wrong)
axs[1, 2].imshow(Potato_e_l_wrong)
axs[2, 0].imshow(Potato_l_h_wrong)
axs[2, 1].imshow(Potato_l_e_wrong)


# add titles to each subplot
axs[0, 0].set_title('Healthy correct')
axs[0, 1].set_title('Early blight correct')
axs[0, 2].set_title('Late blight correct')
axs[1, 0].set_title('Healthy but predict early blight')
axs[1, 1].set_title('Healthy but predict late blight')
axs[1, 2].set_title('early blight but predict late blight')
axs[2, 0].set_title('late blight but predict health')
axs[2, 1].set_title('late blight but predict early blight')


# remove the x and y ticks from each subplot
for ax in axs.flat:
    ax.set_xticks([])
    ax.set_yticks([])

# display the figure
plt.show()
```



Test

Normal

```
# Load dataset
healthy_normalized_compressed_values_test = pd.read_csv('healthy_normalized_compr
early_blight_normalized_compressed_values_test = pd.read_csv('early_blight_normal
late_blight_normalized_compressed_values_test = pd.read_csv('late_blight_normaliz

healthy_normalized_compressed_values_test['Label'] = healthy_label
early_blight_normalized_compressed_values_test['Label'] = early_label
late_blight_normalized_compressed_values_test['Label'] = late_label

# Combine dataset
frames1 = [healthy_normalized_compressed_values_test, early_blight_normalized_com
Normal_test = pd.concat(frames1)
Normal_test.reset_index(drop=True, inplace=True)


Normal_test
```

	0	1	2	3	4	5	6	7	
0	0.835294	0.854902	0.866667	0.839216	0.858824	0.870588	0.839216	0.858824	0.87
1	0.886275	0.878431	0.921569	0.894118	0.886275	0.929412	0.898039	0.890196	0.93
2	0.356863	0.435294	0.776471	0.372549	0.458824	0.796078	0.380392	0.478431	0.81
3	0.921569	0.909804	0.890196	0.921569	0.909804	0.890196	0.921569	0.909804	0.89
4	0.737255	0.733333	0.752941	0.737255	0.733333	0.752941	0.725490	0.721569	0.74
...	...	...	...	...	...	...	...	...	
305	0.850980	0.854902	0.968627	0.878431	0.882353	0.996078	0.874510	0.878431	0.99
306	0.894118	0.901961	0.992157	0.894118	0.901961	0.992157	0.890196	0.898039	0.98
307	0.800000	0.858824	0.980392	0.800000	0.858824	0.980392	0.800000	0.858824	0.98
308	0.870588	0.898039	1.000000	0.870588	0.898039	0.996078	0.866667	0.894118	0.99
309	0.886275	0.898039	0.917647	0.878431	0.890196	0.909804	0.870588	0.882353	0.90

310 rows × 12289 columns

```
Normal_test_dataset = LeafDataset(Normal_test, transform=transform)
Normal_test_loader = DataLoader(Normal_test_dataset, batch_size=64, shuffle=False


# Initialize the prediction and label lists(tensors)
predlist=torch.zeros(0,dtype=torch.long, device='cpu')
lbllist=torch.zeros(0,dtype=torch.long, device='cpu')

with torch.no_grad():
    for i, (inputs, classes) in enumerate(Normal_test_loader):
        inputs = inputs.to(device)
        classes = classes.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)

        # Append batch prediction results
        predlist=torch.cat([predlist,preds.view(-1).cpu()])
        lbllist=torch.cat([lbllist,classes.view(-1).cpu()])

# Confusion matrix
conf_mat=confusion_matrix(lbllist.numpy(), predlist.numpy())
print('Confusion Matrix')
print('-'*16)
print(conf_mat)

# Per-class accuracy
total_correct = conf_mat.diagonal().sum()
total_samples = conf_mat.sum()
total_accuracy = total_correct / total_samples * 100
print('accuracy:', total_accuracy)
```

```
Confusion Matrix
----------------
[[82  4 14]
 [56 24 20]
 [87 11 12]]
accuracy: 38.064516129032256
```

```
# Get the predicted labels
pred_labels = predlist.numpy()

# Get the actual labels
actual_labels = lbllist.numpy()

misclassified_idx = np.where(pred_labels != actual_labels)[0]
print("Misclassified indices: ", misclassified_idx)
```

```
Misclassified indices:  [  0   1   4   5   6   8   9  11  13  14  17  23  32
  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55
  56  57  58  59  60  61  62  63  64  65  66  67  69  70  71  72  73  76
  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94
  96  97  98  99 116 126 127 129 139 148 169 170 172 173 174 176 183 184
 187 188 191 192 200 201 202 203 204 205 206 207 208 209 210 212 213 214
 215 216 217 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 237 238 239 240 241 242 243 244 249 251 252 253 254 255 256 257 258
 259 260 261 262 263 264 265 266 267 269 270 271 272 273 274 275 276 277
 278 279 280 281 282 283 284 285 286 287 288 289 290 291 293 294 295 296
 297 298 300 301 302 303 304 305 306 307 308 309]
```

```
# Display the image
Normal_test_np = Normal_test.iloc[:, :-1].to_numpy()
Potato_healthy_correct = Normal_test_np[2, ].reshape(64,64,3)
Potato_Early_blight_correct = Normal_test_np[145, ].reshape(64,64,3)
Potato_Late_blight_correct = Normal_test_np[211, ].reshape(64,64,3)
Potato_healthyearly_wrong = Normal_test_np[0, ].reshape(64,64,3)
Potato_healthylate_wrong = Normal_test_np[39, ].reshape(64,64,3)
Potato_e_h_wrong = Normal_test_np[116, ].reshape(64,64,3)
Potato_e_l_wrong = Normal_test_np[126, ].reshape(64,64,3)
Potato_l_h_wrong = Normal_test_np[221, ].reshape(64,64,3)
Potato_l_e_wrong = Normal_test_np[219, ].reshape(64,64,3)


# create a figure with six subplots
fig, axs = plt.subplots(3, 3, figsize=(10, 5))

# display each image on a subplot
axs[0, 0].imshow(Potato_healthy_correct)
axs[0, 1].imshow(Potato_Early_blight_correct)
axs[0, 2].imshow(Potato_Late_blight_correct)
axs[1, 0].imshow(Potato_healthyearly_wrong)
axs[1, 1].imshow(Potato_healthylate_wrong)
axs[1, 2].imshow(Potato_e_h_wrong)
axs[2, 0].imshow(Potato_e_l_wrong)
axs[2, 1].imshow(Potato_l_h_wrong)
axs[2, 2].imshow(Potato_l_e_wrong)

# add titles to each subplot
axs[0, 0].set_title('Healthy correct')
axs[0, 1].set_title('Early blight correct')
axs[0, 2].set_title('Late blight correct')
axs[1, 0].set_title('Healthy but predict early blight')
axs[1, 1].set_title('Healthy but predict late blight')
axs[1, 2].set_title('Early blight but predict Healthy')
axs[2, 0].set_title('Early blight but predict late blight')
axs[2, 1].set_title('Late blight but predict Healthy')
axs[2, 2].set_title('Late blight but predict early blight')

# remove the x and y ticks from each subplot
for ax in axs.flat:
    ax.set_xticks([])
    ax.set_yticks([])

# display the figure
plt.show()
```

I trained a Convolutional Neural Network (CNN) model with max pooling and dropout layers using the Adam optimizer with a learning rate of 0.001 for 25 epochs. On the first dataset, the model achieved a training loss of 0.0435, a test loss of 0.1622, and an accuracy of 95.67%. This indicates that the model has strong predictive power with minimal errors in classification.

However, when I applied the model to a different dataset, which has different backgrounds and some filters on each image, the performance dropped significantly to an accuracy of approximately 38.06%. This shows a higher number of misclassifications compared to the first dataset.

The attached image shows examples of leaves that were correctly and incorrectly classified by the model. The top row shows leaves that were correctly classified as healthy, early blight, and late blight. The bottom row shows leaves that were misclassified: a healthy leaf predicted as early blight, an early blight leaf predicted as late blight, and a late blight leaf predicted as early blight.

In conclusion, my model performs well on the first dataset but struggles with the second dataset. This could be due to differences in the characteristics of the images in the two datasets.

Inception V3

Which Model and Methodologies are applied

Motivated by the existing works for the identification of plant disease detection various machine learning (ML) as well as deep learning (DL) methods are developed & examined by various researchers, here we are comparing the performance of ML (Support Vector Machine (SVM)) & DL (Inception-v3, VGG-16) in terms of citrus plant disease detection.

Introduction

Inception V3 is a deep learning model developed by Google for tasks such as image classification, object detection, and image recognition. It is the third version in the Inception series and has shown outstanding performance on the ImageNet dataset.

Inception V3 utilizes a structure called the Inception module, which employs multiple convolutional kernels of different sizes for feature extraction and combines these features at different levels.

Some new features include:

1. Auxiliary classifiers: Additional classifiers are added in the middle layers of the network to provide extra gradient signals during training, aiding in faster convergence.
2. Ensemble of classifiers: Two different classifiers are used for prediction, and their average is taken to reduce the risk of overfitting.
3. Batch normalization: These techniques accelerate the training process and enhance the model's robustness.

Steps for Usage

InceptionV3-Flipped

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.9.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.60.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.42.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (1.
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.5.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tenso
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15-
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensor
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tens
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tens
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflo
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->te
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.
```

```
pip install keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```

```
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.applications import InceptionV3
```

```
from google.colab import drive
drive.mount('/content/drive')
file_path = {
    'flipped_healthy': '/content/drive/My Drive/InceptionV3/Flipped/healthy',
    'flipped_early_blight': '/content/drive/My Drive/InceptionV3/Flipped/early_blight',
    'flipped_late_blight': '/content/drive/My Drive/InceptionV3/Flipped/late_blight'
}
```

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

```
# Define model parameters
num_classes = len(file_path)  # Number of categories
img_height, img_width = 224, 224
batch_size = 32  # Batch size
epochs = 10  # Number of Epochs

# Load pre-trained InceptionV3 model, excluding top layer (fully connected layer)
base_model = InceptionV3(weights='imagenet', include_top=False)

# Add custom global average pooling layer and full connectivity layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop
87910968/87910968 [==============================] - 1s 0us/step
```

```
# Build the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Fine-tune the model: freeze the previous layers and train only the customized top layer
for layer in base_model.layers:
    layer.trainable = False

# Compilation model
model.compile(optimizer=SGD(learning_rate=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])


from sklearn.model_selection import train_test_split
import os
import shutil

# Get all file paths from the file_path dictionary
all_files = []
for category, path in file_path.items():
    all_files.extend([os.path.join(path, f) for f in os.listdir(path)])

# Delineate training and validation sets
train_files, validation_files = train_test_split(all_files, test_size=0.2, random_state=42)

# Create training set and validation set folders
train_dir = '/content/drive/My Drive/InceptionV3/Flipped/train'
validation_dir = '/content/drive/My Drive/InceptionV3/Flipped/validation'

os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
```

```
# Copy the file to the corresponding folder
for file in train_files:
    category = os.path.basename(os.path.dirname(file))
    destination = os.path.join(train_dir, category)
    os.makedirs(destination, exist_ok=True)
    shutil.copy(file, destination)

for file in validation_files:
    category = os.path.basename(os.path.dirname(file))
    destination = os.path.join(validation_dir, category)
    os.makedirs(destination, exist_ok=True)
    shutil.copy(file, destination)

# Set up paths for training and validation data
train_data_dir = '/content/drive/My Drive/InceptionV3/Flipped/train'
validation_data_dir = '/content/drive/My Drive/InceptionV3/Flipped/validation'

# Generator of training data
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')

# Generator of validation data
validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')
```

```
Found 2149 images belonging to 3 classes.
Found 782 images belonging to 3 classes.
```

```
# Training models
modelinceptionv3f = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)
```

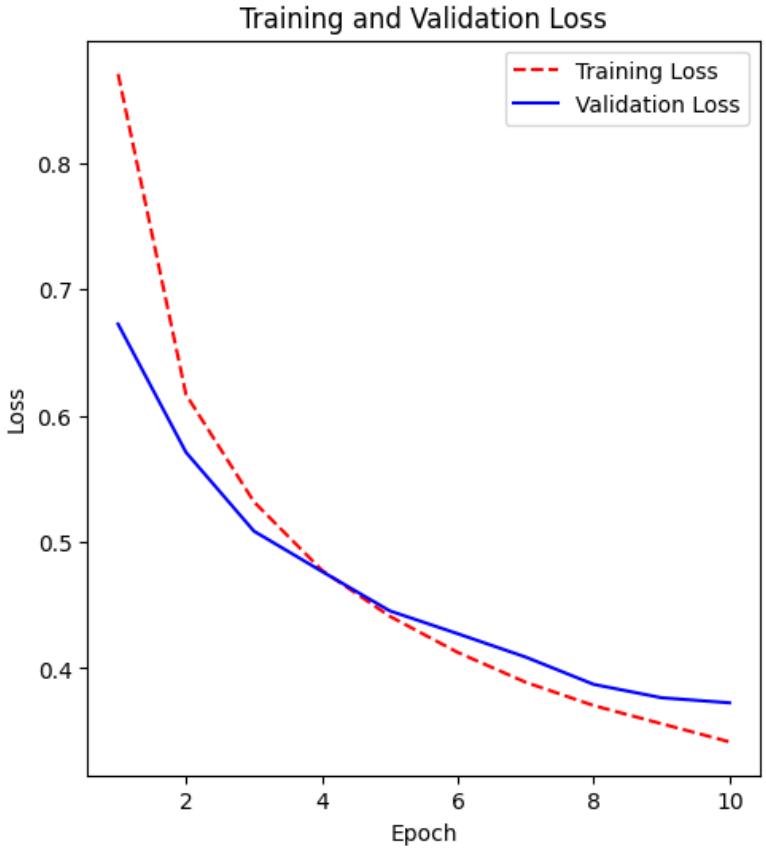
```
Epoch 1/10
67/67 [==============================] - 324s 5s/step - loss: 0.8707 - accuracy: 0.5442 - val_loss: 0.6726 - val_accuracy: 0.7227
Epoch 2/10
```

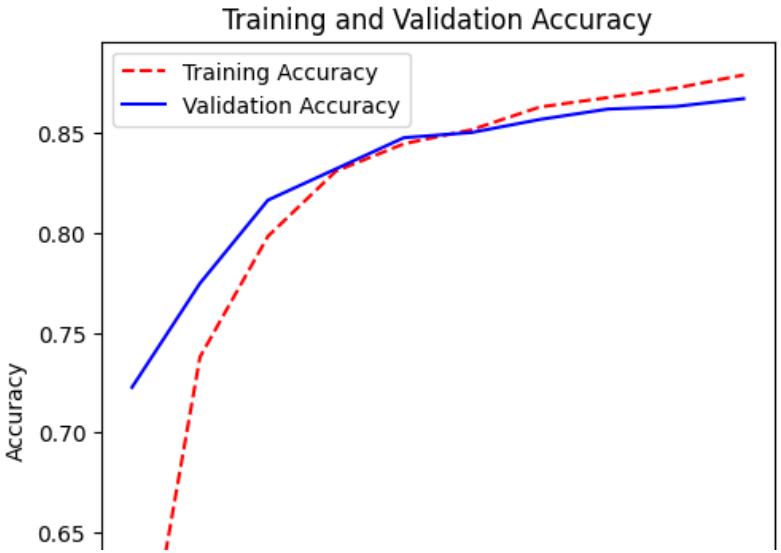
```
67/67 [==============================] - 304s 5s/step - loss: 0.6167 - accuracy: 0.7378 - val_loss: 0.5709 - val_accuracy: 0.7747
Epoch 3/10
67/67 [==============================] - 307s 5s/step - loss: 0.5317 - accuracy: 0.7983 - val_loss: 0.5085 - val_accuracy: 0.8164
Epoch 4/10
67/67 [==============================] - 308s 5s/step - loss: 0.4776 - accuracy: 0.8309 - val_loss: 0.4766 - val_accuracy: 0.8320
Epoch 5/10
67/67 [==============================] - 303s 5s/step - loss: 0.4411 - accuracy: 0.8446 - val_loss: 0.4454 - val_accuracy: 0.8477
Epoch 6/10
67/67 [==============================] - 307s 5s/step - loss: 0.4125 - accuracy: 0.8517 - val_loss: 0.4273 - val_accuracy: 0.8503
Epoch 7/10
67/67 [==============================] - 306s 5s/step - loss: 0.3888 - accuracy: 0.8630 - val_loss: 0.4088 - val_accuracy: 0.8568
Epoch 8/10
67/67 [==============================] - 302s 5s/step - loss: 0.3704 - accuracy: 0.8677 - val_loss: 0.3872 - val_accuracy: 0.8620
Epoch 9/10
67/67 [==============================] - 303s 5s/step - loss: 0.3560 - accuracy: 0.8725 - val_loss: 0.3766 - val_accuracy: 0.8633
Epoch 10/10
67/67 [==============================] - 303s 5s/step - loss: 0.3416 - accuracy: 0.8791 - val_loss: 0.3726 - val_accuracy: 0.8672
```

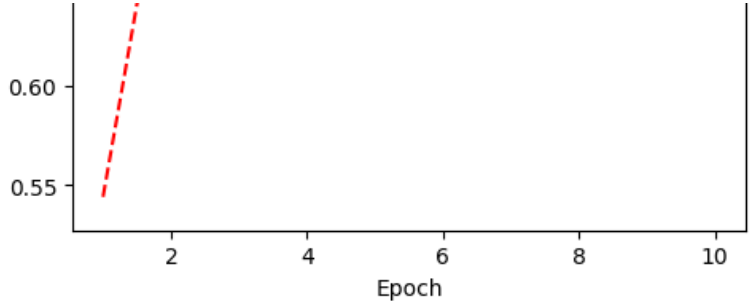
```
# Save the trained model
model.save('/content/drive/My Drive/InceptionV3/InceptionV3_F_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt

# Create count of the number of epochs
epoch_count = range(1, epochs + 1)

# Extract loss history from modelinceptionv3f object
training_loss = modelinceptionv3f.history['loss']
validation_loss = modelinceptionv3f.history['val_loss']

# Visualize loss history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_count, training_loss, 'r--', label='Training Loss')
plt.plot(epoch_count, validation_loss, 'b-', label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()

# Extract loss history from modelinceptionv3cs object
training_acc = modelinceptionv3f.history['accuracy']
validation_acc = modelinceptionv3f.history['val_accuracy']

# Visualize accuracy history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
plt.plot(epoch_count, training_acc, 'r--')
plt.plot(epoch_count, validation_acc, 'b-')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.show()
```


Training and Validation Loss
Training Loss
Validation Loss
Loss
0.8
0.7
0.6
0.5
0.4
2
4
6
8
10
Epoch

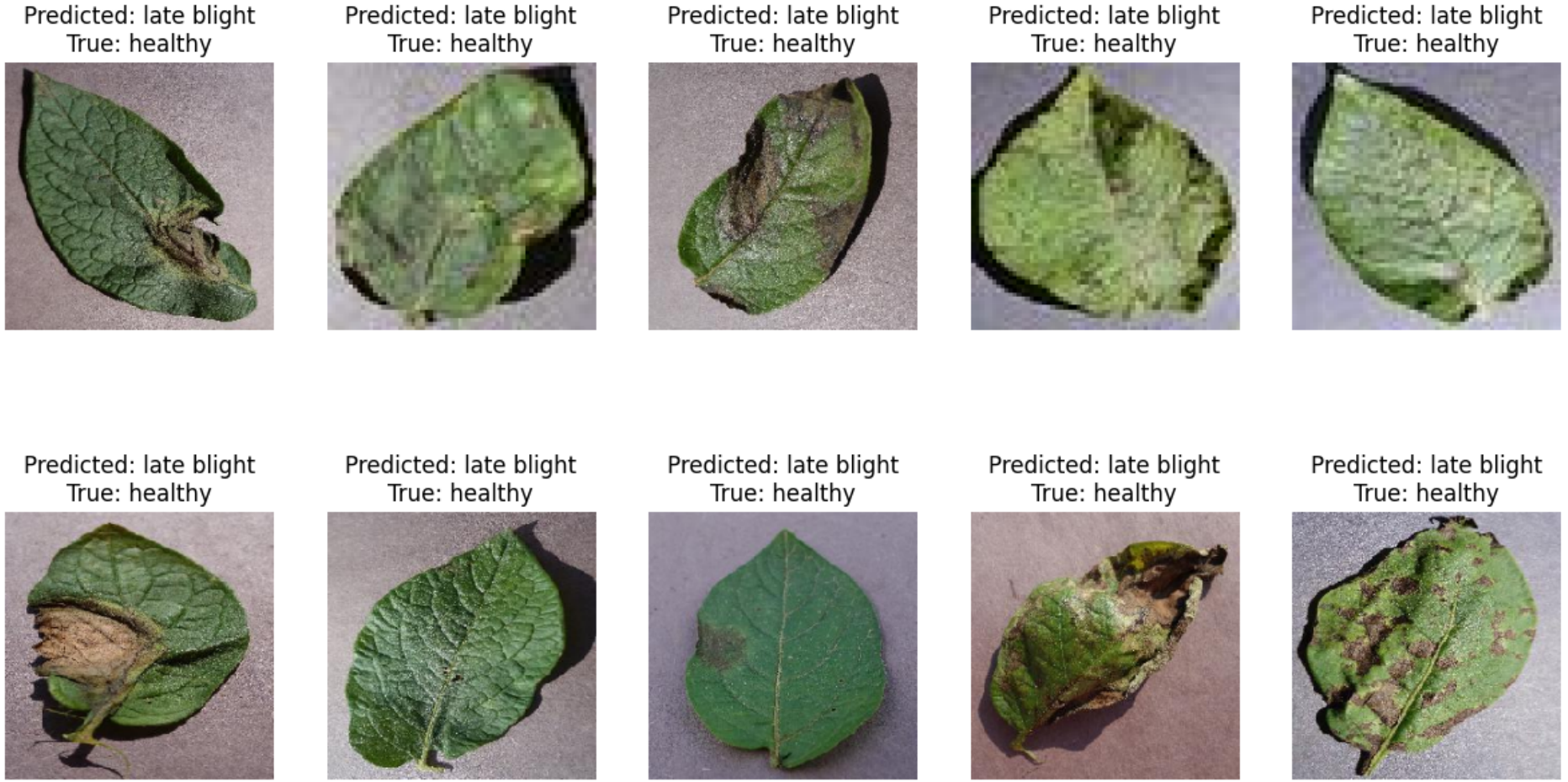

Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.85
0.80
0.75
0.70
0.65




```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
25/25 [==============================] - 89s 3s/step
Confusion Matrix:
[[190  21 173]
 [ 27   2  23]
 [160  20 166]]
```

```
# Define class names
class_names = ['healthy', 'early blight', 'late blight']

# Retrieve images and true labels from validation_generator
validation_images, validation_labels = [], []
for i in range(len(validation_generator)):
    images, labels = validation_generator[i]
    validation_images.append(images)
    validation_labels.append(labels)

validation_images = np.concatenate(validation_images)
validation_labels = np.concatenate(validation_labels)

# Indices of misclassified images
misclassified_indices = np.where(y_pred != y_true)[0]

# Visualize a sample of misclassified images
num_samples = min(len(misclassified_indices), 10)  # Change 10 to visualize more or fewer samples
plt.figure(figsize=(15, 8))
for i, idx in enumerate(misclassified_indices[:num_samples]):
    plt.subplot(2, num_samples // 2, i + 1)
    plt.imshow(validation_images[idx])
    plt.title(f"Predicted: {class_names[y_pred[idx]]}\nTrue: {class_names[y_true[idx]]}")
    plt.axis('off')
plt.show()

# Error analysis
misclassified_labels = y_pred[misclassified_indices]
true_labels = y_true[misclassified_indices]

unique_misclassified_labels, misclassified_counts = np.unique(misclassified_labels, return_counts=True)
unique_true_labels, true_counts = np.unique(true_labels, return_counts=True)

print("Misclassifications by Predicted Class:")
for label, count in zip(unique_misclassified_labels, misclassified_counts):
    print(f"Class: {class_names[label]}, Misclassifications: {count}")

print("\nMisclassifications by True Class:")
for label, count in zip(unique_true_labels, true_counts):
    print(f"Class: {class_names[label]}, Misclassifications: {count}")
```




```
Misclassifications by Predicted Class:
Class: healthy, Misclassifications: 187
Class: early blight, Misclassifications: 41
Class: late blight, Misclassifications: 196

Misclassifications by True Class:
Class: healthy, Misclassifications: 194
Class: early blight, Misclassifications: 50
Class: late blight, Misclassifications: 180
```

```
from tensorflow.keras.models import load_model
from google.colab import drive
import os
drive.mount('/content/drive')
os.chdir('/content/drive/My Drive/InceptionV3/')
InceptionV3f_model=load_model('InceptionV3_F_model.h5')
```

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

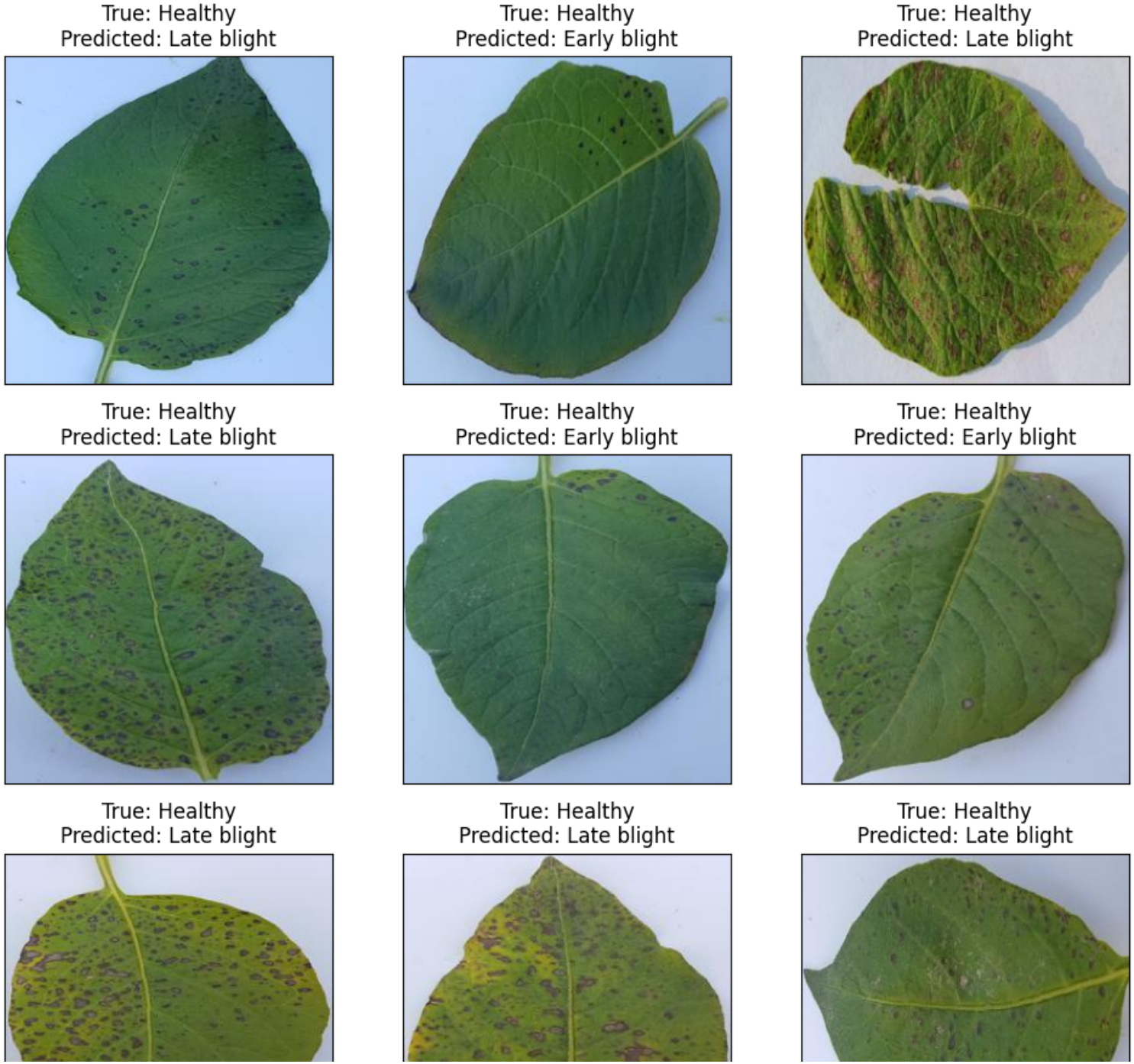
```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Predicted probabilities
y_pred_prob = InceptionV3f_model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)

# Define the image size
image_size = 224  # Placeholder value, replace with the appropriate size

# Load additional test dataset (replace 'test_data_dir' with the appropriate directory)
test_data_dir = '/content/drive/My Drive/InceptionV3/Test/Flipped/'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    test_data_dir,
    target_size=(image_size, image_size),
    batch_size=batch_size,
    class_mode='categorical',
```

```
    shuffle=False
)

# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred = model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred, axis=1)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)

print("Confusion Matrix:")
print(conf_matrix)

from sklearn.metrics import accuracy_score

# Calculate accuracy
accuracy = accuracy_score(y_test_true, y_test_pred)

print("Accuracy:", accuracy)
```

```
25/25 [==============================] - 89s 3s/step
Confusion Matrix:
[[183  20 181]
 [ 27   5  20]
 [167  18 161]]
Found 320 images belonging to 3 classes.
10/10 [==============================] - 64s 7s/step
Confusion Matrix:
[[ 9 23 68]
 [ 0 45 65]
 [ 2 47 61]]
Accuracy: 0.359375
```

```
# Use your trained model to predict labels for the test dataset
predictions = model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]

# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names
class_names = ['Healthy', 'Early blight', 'Late blight']

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()
```

```
10/10 [==============================] – 35s 3s/step
```





Double-click (or enter) to edit

InceptionV3-Original-Epoch 15

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.9.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.60.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.42.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (1.
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.5.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tenso
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15-
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensor
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tens
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tens
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflo
```

```
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->te
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.
```

```
pip install keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```

```
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.applications import InceptionV3
```

```
from google.colab import drive
drive.mount('/content/drive')
file_path = {
    'original_healthy': '/content/drive/My Drive/InceptionV3/Original/healthy',
    'original_early_blight': '/content/drive/My Drive/InceptionV3/Original/early_blight',
    'original_late_blight': '/content/drive/My Drive/InceptionV3/Original/late_blight'
}
```

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

```
# Define model parameters
num_classes = len(file_path)
img_height, img_width = 224, 224
batch_size = 32  # Batch size
epochs = 15  # Number of Epochs

# Load pre-trained InceptionV3 model, excluding top layer (fully connected layer)v
base_model = InceptionV3(weights='imagenet', include_top=False)

# Add custom global average pooling layer and full connectivity layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
```

```
# Build the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Fine-tune the model: freeze the previous layers and train only the customized top layer
for layer in base_model.layers:
    layer.trainable = False

# Compilation model
model.compile(optimizer=SGD(learning_rate=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])


from sklearn.model_selection import train_test_split
import os
import shutil

# Get all file paths from the file_path dictionary
all_files = []
for category, path in file_path.items():
    all_files.extend([os.path.join(path, f) for f in os.listdir(path)])

# Delineate training and validation sets
train_files, validation_files = train_test_split(all_files, test_size=0.2, random_state=42)

# Create training set and validation set folders
train_dir = '/content/drive/My Drive/InceptionV3/Original/train2'
validation_dir = '/content/drive/My Drive/InceptionV3/Original/validation2'

os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
```

```
# Copy the file to the corresponding folder
for file in train_files:
    category = os.path.basename(os.path.dirname(file))
    destination = os.path.join(train_dir, category)
    os.makedirs(destination, exist_ok=True)
    shutil.copy(file, destination)

for file in validation_files:
    category = os.path.basename(os.path.dirname(file))
    destination = os.path.join(validation_dir, category)
    os.makedirs(destination, exist_ok=True)
    shutil.copy(file, destination)

# Set up paths for training and validation data
train_data_dir = '/content/drive/My Drive/InceptionV3/Original/train2'
validation_data_dir = '/content/drive/My Drive/InceptionV3/Original/validation2'

# Generator of training data
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')

# Generator of validation data
validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')
```

```
Found 1788 images belonging to 3 classes.
Found 448 images belonging to 3 classes.
```

```
# Training models
modelinceptionv3o = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)
```

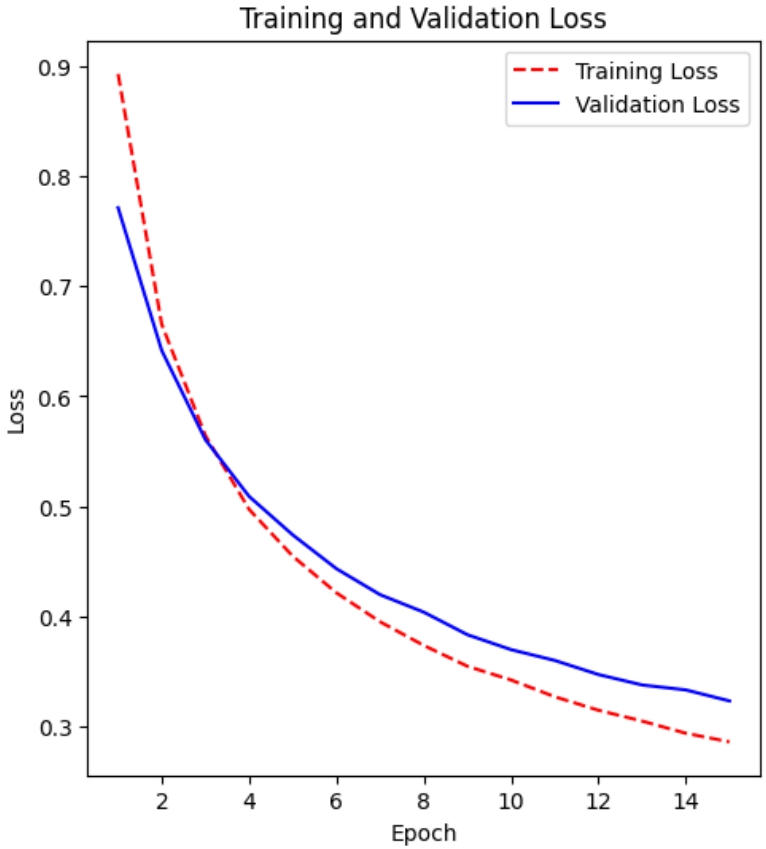
```
Epoch 1/15
55/55 [==============================] - 293s 5s/step - loss: 0.8932 - accuracy: 0.5404 - val_loss: 0.7716 - val_accuracy: 0.6339
Epoch 2/15
```

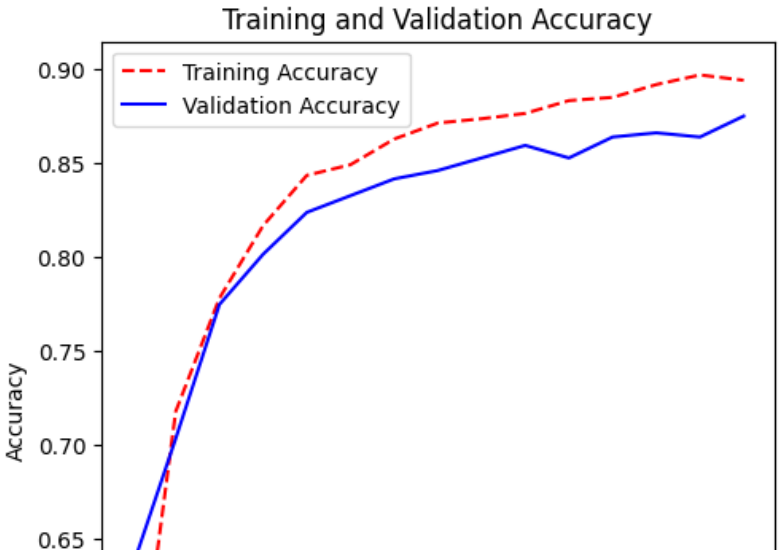
```
55/55 [==============================] - 282s 5s/step - loss: 0.6654 - accuracy: 0.7175 - val_loss: 0.6416 - val_accuracy: 0.7031
Epoch 3/15
55/55 [==============================] - 276s 5s/step - loss: 0.5643 - accuracy: 0.7779 - val_loss: 0.5606 - val_accuracy: 0.7746
Epoch 4/15
55/55 [==============================] - 271s 5s/step - loss: 0.4973 - accuracy: 0.8166 - val_loss: 0.5090 - val_accuracy: 0.8013
Epoch 5/15
55/55 [==============================] - 270s 5s/step - loss: 0.4547 - accuracy: 0.8434 - val_loss: 0.4740 - val_accuracy: 0.8237
Epoch 6/15
55/55 [==============================] - 270s 5s/step - loss: 0.4214 - accuracy: 0.8491 - val_loss: 0.4433 - val_accuracy: 0.8326
Epoch 7/15
55/55 [==============================] - 270s 5s/step - loss: 0.3950 - accuracy: 0.8628 - val_loss: 0.4197 - val_accuracy: 0.8415
Epoch 8/15
55/55 [==============================] - 271s 5s/step - loss: 0.3735 - accuracy: 0.8713 - val_loss: 0.4038 - val_accuracy: 0.8460
Epoch 9/15
55/55 [==============================] - 238s 4s/step - loss: 0.3546 - accuracy: 0.8736 - val_loss: 0.3833 - val_accuracy: 0.8527
Epoch 10/15
55/55 [==============================] - 241s 4s/step - loss: 0.3421 - accuracy: 0.8764 - val_loss: 0.3697 - val_accuracy: 0.8594
Epoch 11/15
55/55 [==============================] - 271s 5s/step - loss: 0.3269 - accuracy: 0.8833 - val_loss: 0.3599 - val_accuracy: 0.8527
Epoch 12/15
55/55 [==============================] - 274s 5s/step - loss: 0.3146 - accuracy: 0.8850 - val_loss: 0.3471 - val_accuracy: 0.8638
Epoch 13/15
55/55 [==============================] - 238s 4s/step - loss: 0.3048 - accuracy: 0.8918 - val_loss: 0.3376 - val_accuracy: 0.8661
Epoch 14/15
55/55 [==============================] - 271s 5s/step - loss: 0.2937 - accuracy: 0.8969 - val_loss: 0.3331 - val_accuracy: 0.8638
Epoch 15/15
55/55 [==============================] - 239s 4s/step - loss: 0.2859 - accuracy: 0.8941 - val_loss: 0.3231 - val_accuracy: 0.8750
```

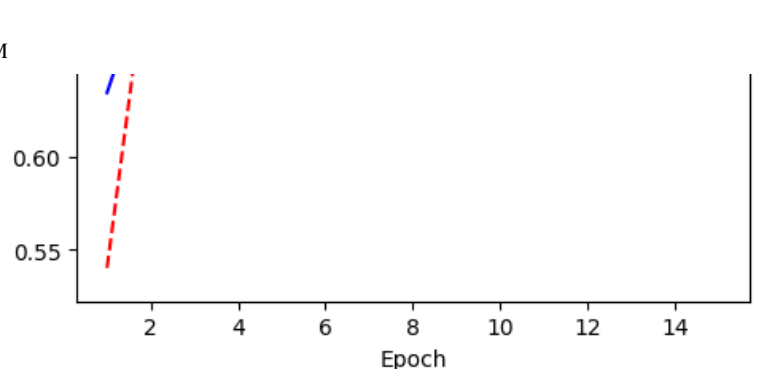
```
# Save models
model.save('/content/drive/My Drive/InceptionV3/InceptionV3_O_2_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt

# Create count of the number of epochs
epoch_count = range(1, epochs + 1)

# Extract loss history from modelinceptionv3o object
training_loss = modelinceptionv3o.history['loss']
validation_loss = modelinceptionv3o.history['val_loss']

# Visualize loss history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_count, training_loss, 'r--', label='Training Loss')
plt.plot(epoch_count, validation_loss, 'b-', label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()

# Extract loss history from modelinceptionv3cs object
training_acc = modelinceptionv3o.history['accuracy']
validation_acc = modelinceptionv3o.history['val_accuracy']

# Visualize accuracy history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
plt.plot(epoch_count, training_acc, 'r--')
plt.plot(epoch_count, validation_acc, 'b-')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.show()
```


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
2
4
6
8
10
12
14

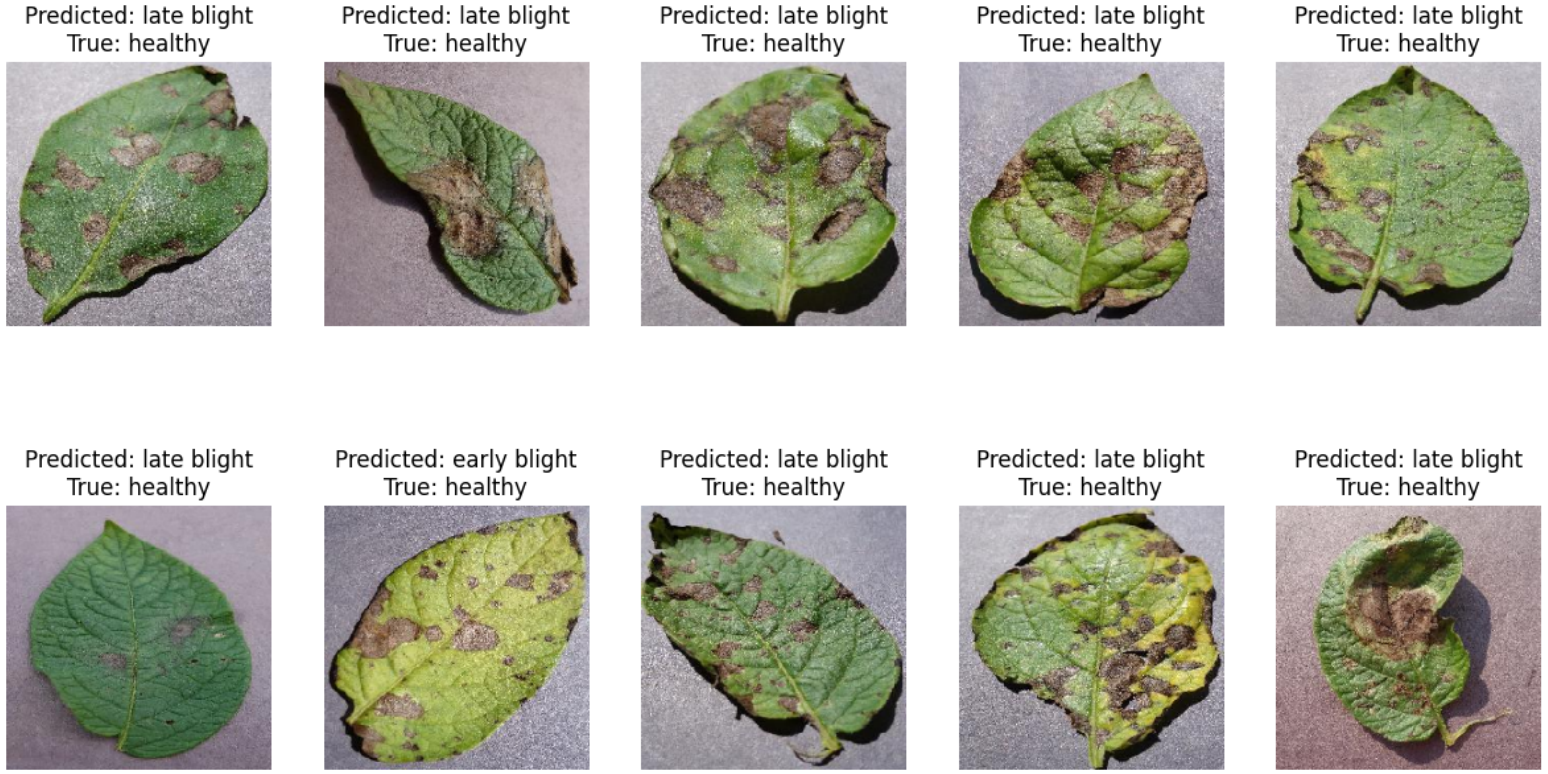

Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65




```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
14/14 [==============================] - 50s 3s/step
Confusion Matrix:
[[ 85  18 101]
 [ 18   3  27]
 [ 83  15  98]]
```

```
# Define class names
class_names = ['healthy', 'early blight', 'late blight']

# Retrieve images and true labels from validation_generator
validation_images, validation_labels = [], []
for i in range(len(validation_generator)):
    images, labels = validation_generator[i]
    validation_images.append(images)
    validation_labels.append(labels)

validation_images = np.concatenate(validation_images)
validation_labels = np.concatenate(validation_labels)

# Indices of misclassified images
misclassified_indices = np.where(y_pred != y_true)[0]

# Visualize a sample of misclassified images
num_samples = min(len(misclassified_indices), 10)  # Change 10 to visualize more or fewer samples
plt.figure(figsize=(15, 8))
for i, idx in enumerate(misclassified_indices[:num_samples]):
    plt.subplot(2, num_samples // 2, i + 1)
    plt.imshow(validation_images[idx])
    plt.title(f"Predicted: {class_names[y_pred[idx]]}\nTrue: {class_names[y_true[idx]]}")
    plt.axis('off')
plt.show()

# Error analysis
misclassified_labels = y_pred[misclassified_indices]
true_labels = y_true[misclassified_indices]

unique_misclassified_labels, misclassified_counts = np.unique(misclassified_labels, return_counts=True)
unique_true_labels, true_counts = np.unique(true_labels, return_counts=True)

print("Misclassifications by Predicted Class:")
for label, count in zip(unique_misclassified_labels, misclassified_counts):
    print(f"Class: {class_names[label]}, Misclassifications: {count}")

print("\nMisclassifications by True Class:")
for label, count in zip(unique_true_labels, true_counts):
    print(f"Class: {class_names[label]}, Misclassifications: {count}")
```




```
Misclassifications by Predicted Class:
Class: healthy, Misclassifications: 101
Class: early blight, Misclassifications: 33
Class: late blight, Misclassifications: 128

Misclassifications by True Class:
Class: healthy, Misclassifications: 119
Class: early blight, Misclassifications: 45
Class: late blight, Misclassifications: 98
```

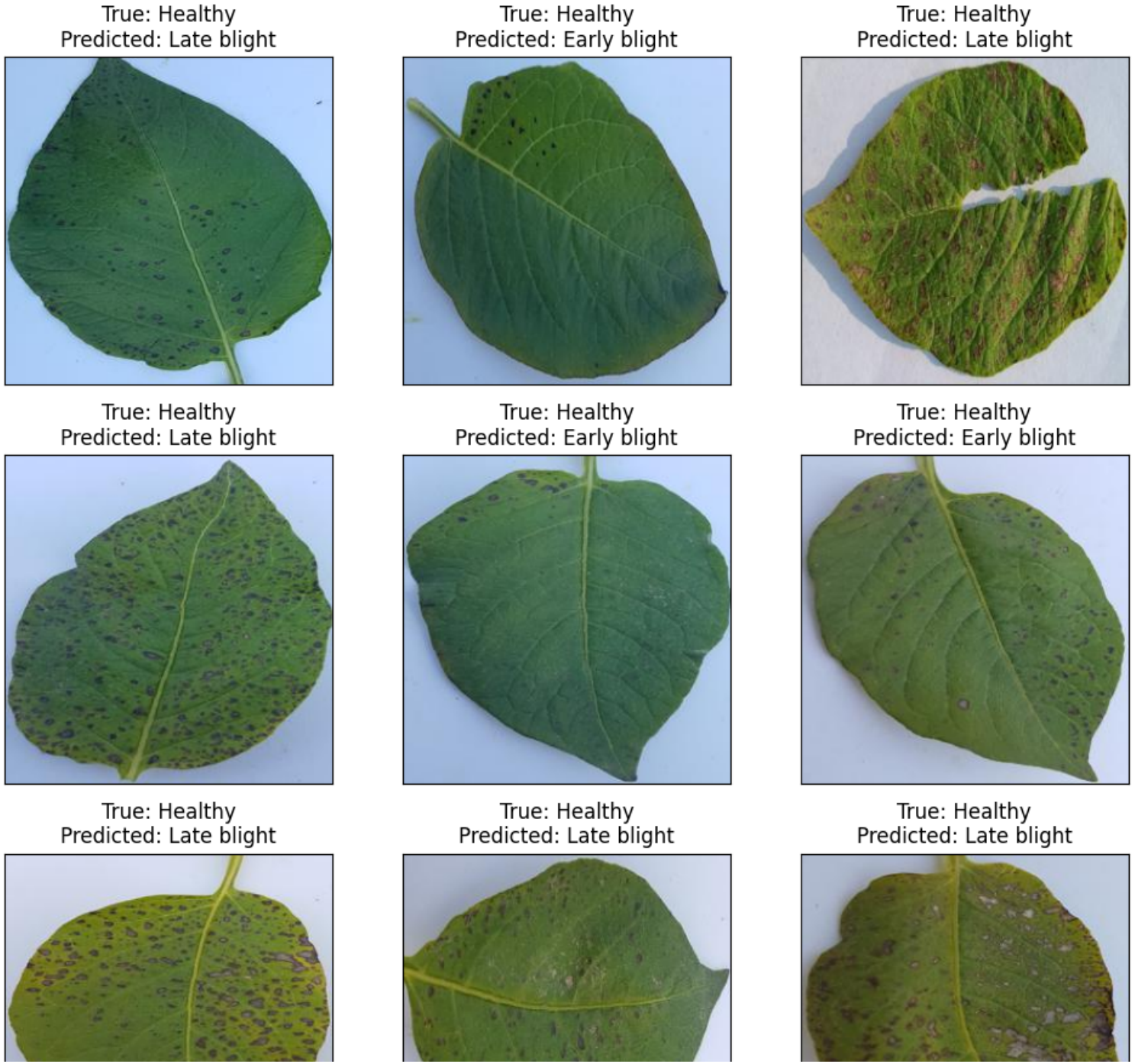
```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)

# Define the image size
image_size = 224  # Placeholder value, replace with the appropriate size

# Load additional test dataset (replace 'test_data_dir' with the appropriate directory)
test_data_dir = '/content/drive/My Drive/InceptionV3/Test/original/'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    test_data_dir,
    target_size=(image_size, image_size),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred = model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred, axis=1)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)

print("Confusion Matrix:")
print(conf_matrix)

from sklearn.metrics import accuracy_score

# Calculate accuracy
```

```
accuracy = accuracy_score(y_test_true, y_test_pred)

print("Accuracy:", accuracy)
```

```
14/14 [==============================] - 48s 3s/step
Confusion Matrix:
[[ 91  19  94]
 [ 17   2  29]
 [ 78  15 103]]
Found 310 images belonging to 3 classes.
10/10 [==============================] - 33s 3s/step
Confusion Matrix:
[[ 6 14 80]
 [ 0 52 48]
 [ 0 42 68]]
Accuracy: 0.4064516129032258
```

```
import numpy as np

# Use your trained model to predict labels for the test dataset
predictions = model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]

# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names
class_names = ['Healthy', 'Early blight', 'Late blight']

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()
```

```
10/10 [==============================] – 34s 3s/step
```





InceptionV3-Original-Epoch 10

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.9.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.60.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.42.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (1.
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.5.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tenso
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15-
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensor
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tens
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tens
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflo
```

```
    Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->te
    Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.
```

```
pip install keras
```

```
    Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```

```
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.applications import InceptionV3
```

```
from google.colab import drive
drive.mount('/content/drive')
file_path = {
    'original_healthy': '/content/drive/My Drive/InceptionV3/Original/healthy',
    'original_early_blight': '/content/drive/My Drive/InceptionV3/Original/early_blight',
    'original_late_blight': '/content/drive/My Drive/InceptionV3/Original/late_blight'
}
```

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

```
# Define model parameters
num_classes = len(file_path)
img_height, img_width = 224, 224
batch_size = 32  # Batch size
epochs = 10  # Number of Epochs

# Load pre-trained InceptionV3 model, excluding top layer (fully connected layer)v
base_model = InceptionV3(weights='imagenet', include_top=False)

# Add custom global average pooling layer and full connectivity layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
```

```
# Build the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Fine-tune the model: freeze the previous layers and train only the customized top layer
for layer in base_model.layers:
    layer.trainable = False

# Compilation model
model.compile(optimizer=SGD(learning_rate=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])


from sklearn.model_selection import train_test_split
import os
import shutil

# Get all file paths from the file_path dictionary
all_files = []
for category, path in file_path.items():
    all_files.extend([os.path.join(path, f) for f in os.listdir(path)])

# Delineate training and validation sets
train_files, validation_files = train_test_split(all_files, test_size=0.2, random_state=42)

# Create training set and validation set folders
train_dir = '/content/drive/My Drive/InceptionV3/Original/train'
validation_dir = '/content/drive/My Drive/InceptionV3/Original/validation'

os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
```

```
# Copy the file to the corresponding folder
for file in train_files:
    category = os.path.basename(os.path.dirname(file))
    destination = os.path.join(train_dir, category)
    os.makedirs(destination, exist_ok=True)
    shutil.copy(file, destination)

for file in validation_files:
    category = os.path.basename(os.path.dirname(file))
    destination = os.path.join(validation_dir, category)
    os.makedirs(destination, exist_ok=True)
    shutil.copy(file, destination)

# Set up paths for training and validation data
train_data_dir = '/content/drive/My Drive/InceptionV3/Original/train'
validation_data_dir = '/content/drive/My Drive/InceptionV3/Original/validation'

# Generator of training data
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')

# Generator of validation data
validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')
```

```
Found 2103 images belonging to 3 classes.
Found 763 images belonging to 3 classes.
```

```
# Training models
modelinceptionv3o = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)
```

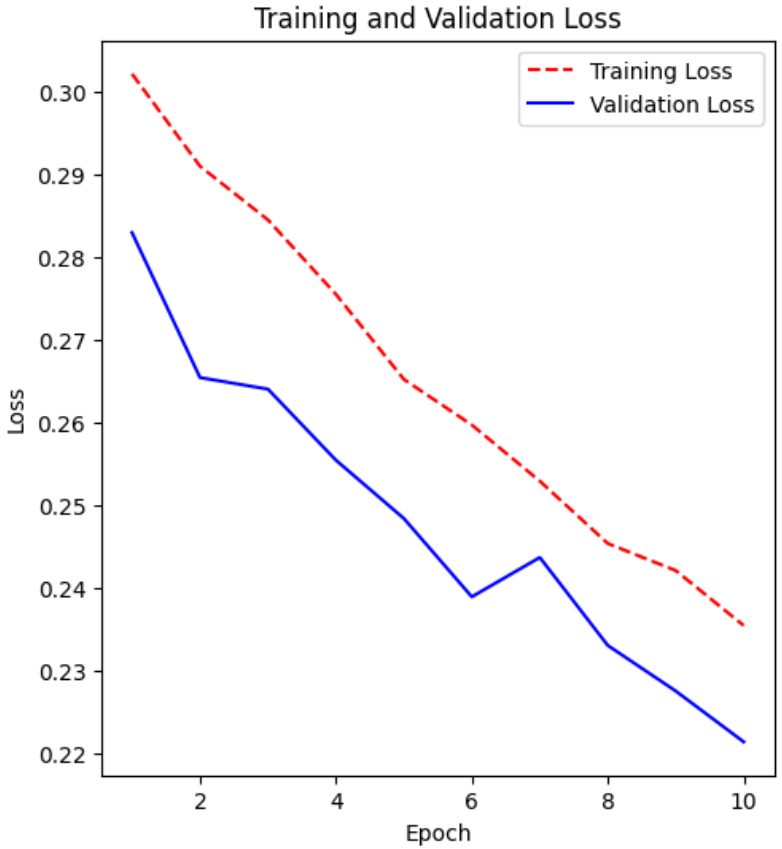
```
Epoch 1/10
65/65 [==============================] - 326s 5s/step - loss: 0.3021 - accuracy: 0.8938 - val_loss: 0.2829 - val_accuracy: 0.9103
Epoch 2/10
```

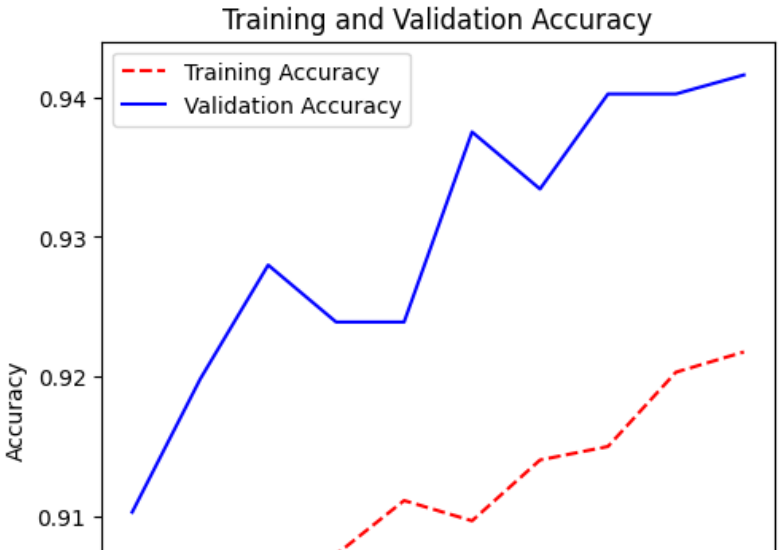
```
65/65 [==============================] - 311s 5s/step - loss: 0.2910 - accuracy: 0.8991 - val_loss: 0.2654 - val_accuracy: 0.9198
Epoch 3/10
65/65 [==============================] - 312s 5s/step - loss: 0.2845 - accuracy: 0.8967 - val_loss: 0.2640 - val_accuracy: 0.9280
Epoch 4/10
65/65 [==============================] - 312s 5s/step - loss: 0.2755 - accuracy: 0.9073 - val_loss: 0.2554 - val_accuracy: 0.9239
Epoch 5/10
65/65 [==============================] - 311s 5s/step - loss: 0.2652 - accuracy: 0.9112 - val_loss: 0.2484 - val_accuracy: 0.9239
Epoch 6/10
65/65 [==============================] - 315s 5s/step - loss: 0.2597 - accuracy: 0.9097 - val_loss: 0.2389 - val_accuracy: 0.9375
Epoch 7/10
65/65 [==============================] - 312s 5s/step - loss: 0.2529 - accuracy: 0.9141 - val_loss: 0.2437 - val_accuracy: 0.9334
Epoch 8/10
65/65 [==============================] - 313s 5s/step - loss: 0.2453 - accuracy: 0.9150 - val_loss: 0.2330 - val_accuracy: 0.9402
Epoch 9/10
65/65 [==============================] - 313s 5s/step - loss: 0.2421 - accuracy: 0.9203 - val_loss: 0.2275 - val_accuracy: 0.9402
Epoch 10/10
65/65 [==============================] - 310s 5s/step - loss: 0.2354 - accuracy: 0.9218 - val_loss: 0.2214 - val_accuracy: 0.9416
```

```
# Save models
model.save('/content/drive/My Drive/InceptionV3/InceptionV3_0_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt

# Create count of the number of epochs
epoch_count = range(1, epochs + 1)

# Extract loss history from modelinceptionv3o object
training_loss = modelinceptionv3o.history['loss']
validation_loss = modelinceptionv3o.history['val_loss']

# Visualize loss history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_count, training_loss, 'r--', label='Training Loss')
plt.plot(epoch_count, validation_loss, 'b-', label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()

# Extract loss history from modelinceptionv3cs object
training_acc = modelinceptionv3o.history['accuracy']
validation_acc = modelinceptionv3o.history['val_accuracy']

# Visualize accuracy history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
plt.plot(epoch_count, training_acc, 'r--')
plt.plot(epoch_count, validation_acc, 'b-')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.show()
```

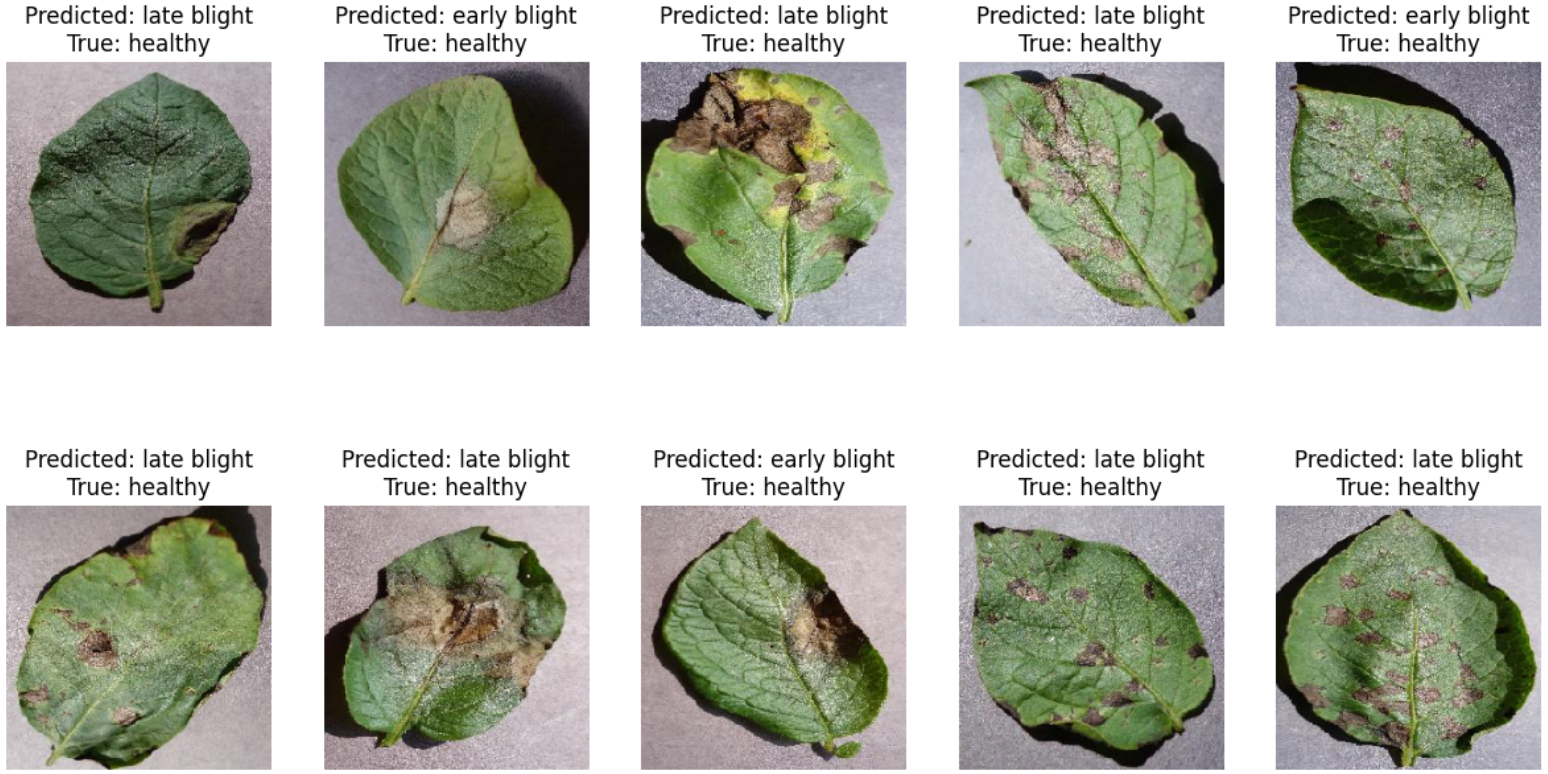

Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
0.30
0.29
0.28
0.27
0.26
0.25
0.24
0.23
0.22
2
4
6
8
10


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.94
0.93
0.92
0.91


```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
24/24 [==============================] - 86s 3s/step
Confusion Matrix:
[[178  20 165]
 [ 26   2  20]
 [158  16 178]]
```

```
# Define class names
class_names = ['healthy', 'early blight', 'late blight']

# Retrieve images and true labels from validation_generator
validation_images, validation_labels = [], []
for i in range(len(validation_generator)):
    images, labels = validation_generator[i]
    validation_images.append(images)
    validation_labels.append(labels)

validation_images = np.concatenate(validation_images)
validation_labels = np.concatenate(validation_labels)

# Indices of misclassified images
misclassified_indices = np.where(y_pred != y_true)[0]

# Visualize a sample of misclassified images
num_samples = min(len(misclassified_indices), 10)  # Change 10 to visualize more or fewer samples
plt.figure(figsize=(15, 8))
for i, idx in enumerate(misclassified_indices[:num_samples]):
    plt.subplot(2, num_samples // 2, i + 1)
    plt.imshow(validation_images[idx])
    plt.title(f"Predicted: {class_names[y_pred[idx]]}\nTrue: {class_names[y_true[idx]]}")
    plt.axis('off')
plt.show()

# Error analysis
misclassified_labels = y_pred[misclassified_indices]
true_labels = y_true[misclassified_indices]

unique_misclassified_labels, misclassified_counts = np.unique(misclassified_labels, return_counts=True)
unique_true_labels, true_counts = np.unique(true_labels, return_counts=True)

print("Misclassifications by Predicted Class:")
for label, count in zip(unique_misclassified_labels, misclassified_counts):
    print(f"Class: {class_names[label]}, Misclassifications: {count}")

print("\nMisclassifications by True Class:")
for label, count in zip(unique_true_labels, true_counts):
    print(f"Class: {class_names[label]}, Misclassifications: {count}")
```




```
Misclassifications by Predicted Class:
Class: healthy, Misclassifications: 184
Class: early blight, Misclassifications: 36
Class: late blight, Misclassifications: 185

Misclassifications by True Class:
Class: healthy, Misclassifications: 185
Class: early blight, Misclassifications: 46
Class: late blight, Misclassifications: 174
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)

# Define the image size
image_size = 224  # Placeholder value, replace with the appropriate size

# Load additional test dataset (replace 'test_data_dir' with the appropriate directory)
test_data_dir = '/content/drive/My Drive/InceptionV3/Test/original/'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    test_data_dir,
    target_size=(image_size, image_size),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred = model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred, axis=1)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)

print("Confusion Matrix:")
print(conf_matrix)

from sklearn.metrics import accuracy_score

# Calculate accuracy
```

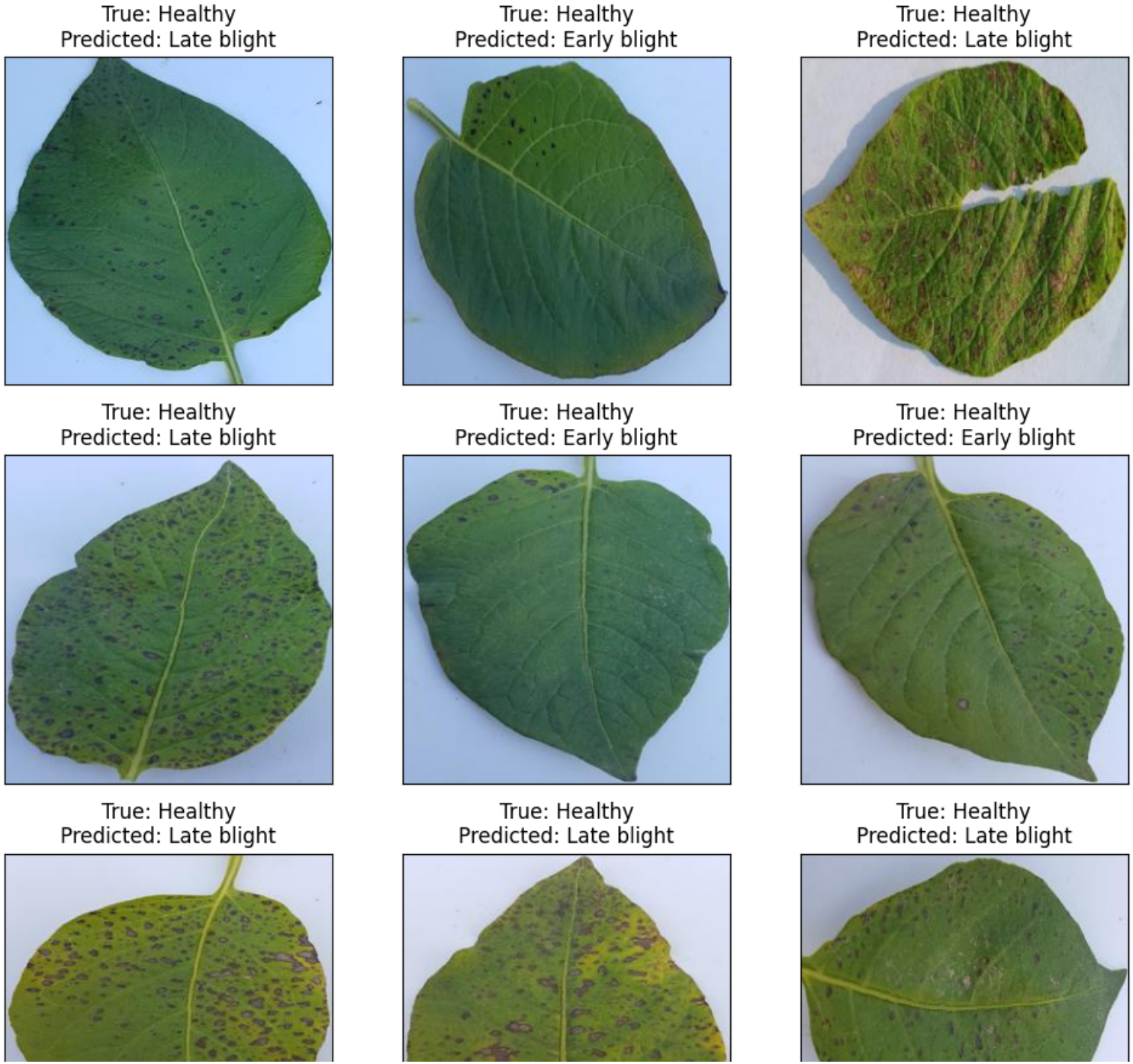
```
accuracy = accuracy_score(y_test_true, y_test_pred)

print("Accuracy:", accuracy)
```

```
24/24 [==============================] - 86s 4s/step
Confusion Matrix:
[[163  16 184]
 [ 21   3  24]
 [178  19 155]]
Found 310 images belonging to 3 classes.
10/10 [==============================] - 64s 7s/step
Confusion Matrix:
[[ 5  7 88]
 [ 0 43 57]
 [ 1 32 77]]
Accuracy: 0.4032258064516129
```

```
import numpy as np

# Use your trained model to predict labels for the test dataset
predictions = model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]

# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names
class_names = ['Healthy', 'Early blight', 'Late blight']

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
```

```
        for j in range(3):
            index = i * 3 + j
            if index < len(misclassified_images):
                true_label = class_names[true_labels_misclassified[index]]
                predicted_label = class_names[predicted_labels_misclassified[index]]
                axs[i, j].imshow(misclassified_images[index])
                axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

    # Remove the x and y ticks from each subplot
    for ax in axs.flatten():
        ax.set_xticks([])
        ax.set_yticks([])

    plt.tight_layout()
    plt.show()
```

```
10/10 [==============================] – 34s 3s/step
```





Compressed+Sharpen

1. Load pre-trained model-using deep learning frameworks like tensorflow,keras

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.60
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from te
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0-
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.1
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorbo
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3
```

```
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->ten
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->te
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0
```

```
pip install keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```

```
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD


from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.applications import InceptionV3
```

2. Prepare data: Prepare the image dataset and preprocess it, including resizing, normalization, cropping, etc. (Have already done during moduel 1)_Fine-tune the model.

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

```
Mounted at /content/drive
```

```
file_path = {
    'compressedsharpen_healthy': '/content/drive/My Drive/InceptionV3/CompressedSharpen/healthy',
    'compressedsharpen_early_blight': '/content/drive/My Drive/InceptionV3/CompressedSharpen/early_blight',
    'compressedsharpen_late_blight': '/content/drive/My Drive/InceptionV3/CompressedSharpen/late_blight'
}
```

3. Model Training: Train the fine-tuned Inception V3 model using the prepared dataset. During training, monitor the model's performance and adjust as necessary.

```
num_classes = len(file_path)
img_height, img_width = 224, 224
batch_size = 32
epochs = 10
```

```
base_model = InceptionV3(weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights
87910968/87910968 [==============================] - 1s 0us/step
```

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
```

```
model = Model(inputs=base_model.input, outputs=predictions)
```

```
for layer in base_model.layers:
    layer.trainable = False
```

```
model.compile(optimizer=SGD(learning_rate=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
from sklearn.model_selection import train_test_split
import os
import shutil


all_files = []
for category, path in file_path.items():
    all_files.extend([os.path.join(path, f) for f in os.listdir(path)])


train_files, validation_files = train_test_split(all_files, test_size=0.2, random_state=42)

train_dir = '/content/drive/My Drive/InceptionV3/CompressedSharpen/train'
validation_dir = '/content/drive/My Drive/InceptionV3/CompressedSharpen/validation'

os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)


for file in train_files:
    category = os.path.basename(os.path.dirname(file))
    destination = os.path.join(train_dir, category)
    os.makedirs(destination, exist_ok=True)
    shutil.copy(file, destination)

for file in validation_files:
    category = os.path.basename(os.path.dirname(file))
    destination = os.path.join(validation_dir, category)
    os.makedirs(destination, exist_ok=True)
    shutil.copy(file, destination)


train_data_dir = '/content/drive/My Drive/InceptionV3/CompressedSharpen/train'
validation_data_dir = '/content/drive/My Drive/InceptionV3/CompressedSharpen/validation'
```

```
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')
```

```
Found 2150 images belonging to 3 classes.
```

```
validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')
```

```
Found 804 images belonging to 3 classes.
```

```
modelinceptionv3cs = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)
```

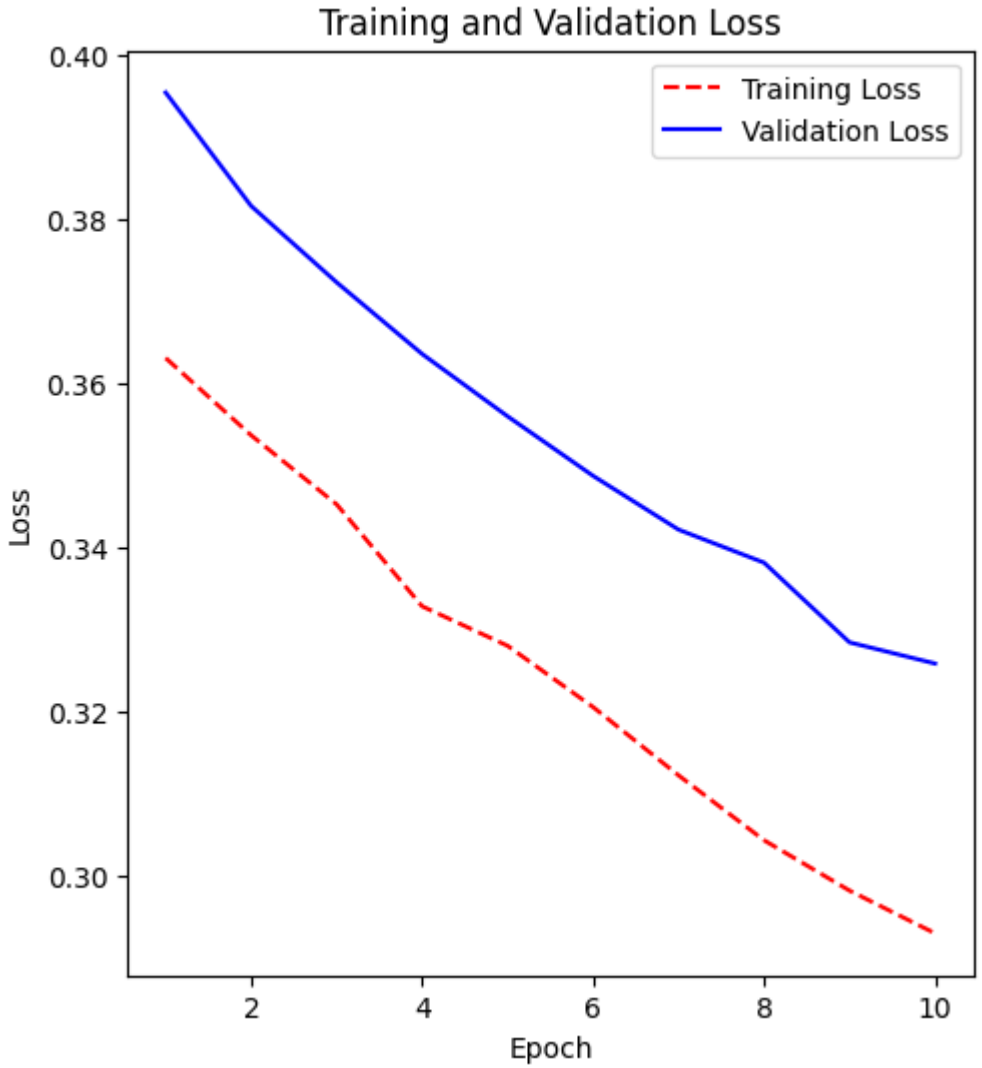
```
Epoch 1/10
67/67 [==============================] - 427s 6s/step - loss: 0.3630 - accuracy: 0.8829 - val_loss: 0.3953 - val_acc
Epoch 2/10
67/67 [==============================] - 367s 6s/step - loss: 0.3536 - accuracy: 0.8805 - val_loss: 0.3815 - val_acc
Epoch 3/10
67/67 [==============================] - 309s 5s/step - loss: 0.3452 - accuracy: 0.8853 - val_loss: 0.3722 - val_acc
Epoch 4/10
67/67 [==============================] - 308s 5s/step - loss: 0.3328 - accuracy: 0.8928 - val_loss: 0.3635 - val_acc
Epoch 5/10
67/67 [==============================] - 367s 5s/step - loss: 0.3280 - accuracy: 0.8947 - val_loss: 0.3559 - val_acc
Epoch 6/10
67/67 [==============================] - 366s 5s/step - loss: 0.3205 - accuracy: 0.8952 - val_loss: 0.3487 - val_acc
```

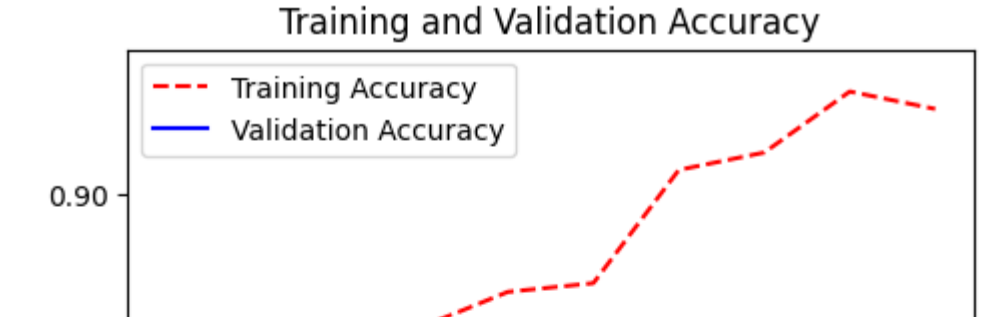
```
Epoch 7/10
67/67 [==============================] - 308s 5s/step - loss: 0.3122 - accuracy: 0.9013 - val_loss: 0.3421 - val_acc
Epoch 8/10
67/67 [==============================] - 310s 5s/step - loss: 0.3043 - accuracy: 0.9023 - val_loss: 0.3381 - val_acc
Epoch 9/10
67/67 [==============================] - 367s 6s/step - loss: 0.2982 - accuracy: 0.9056 - val_loss: 0.3284 - val_acc
Epoch 10/10
67/67 [==============================] - 313s 5s/step - loss: 0.2930 - accuracy: 0.9046 - val_loss: 0.3258 - val_acc
```

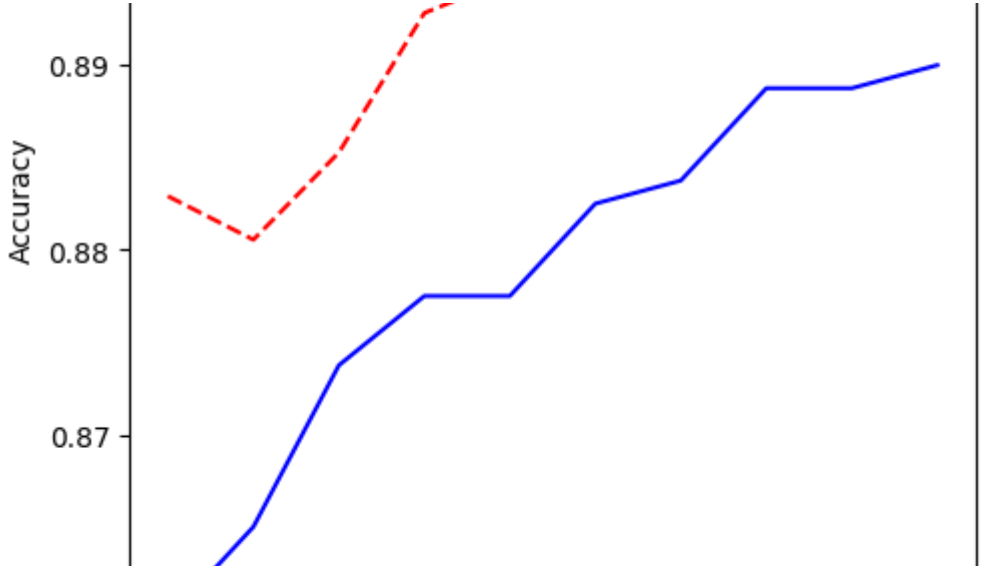
```
model.save('/content/drive/My Drive/InceptionV3/InceptionV3_CS_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt


epoch_count = range(1, epochs + 1)

training_loss = modelinceptionv3cs.history['loss']
validation_loss = modelinceptionv3cs.history['val_loss']

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_count, training_loss, 'r--', label='Training Loss')
plt.plot(epoch_count, validation_loss, 'b-', label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()

training_acc = modelinceptionv3cs.history['accuracy']
validation_acc = modelinceptionv3cs.history['val_accuracy']

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
plt.plot(epoch_count, training_acc, 'r--')
plt.plot(epoch_count, validation_acc, 'b-')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.show()
```


Training and Validation Loss
Training Loss
Validation Loss
0.40
0.38
0.36
0.34
0.32
0.30
Loss
2
4
6
8
10
Epoch


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.90



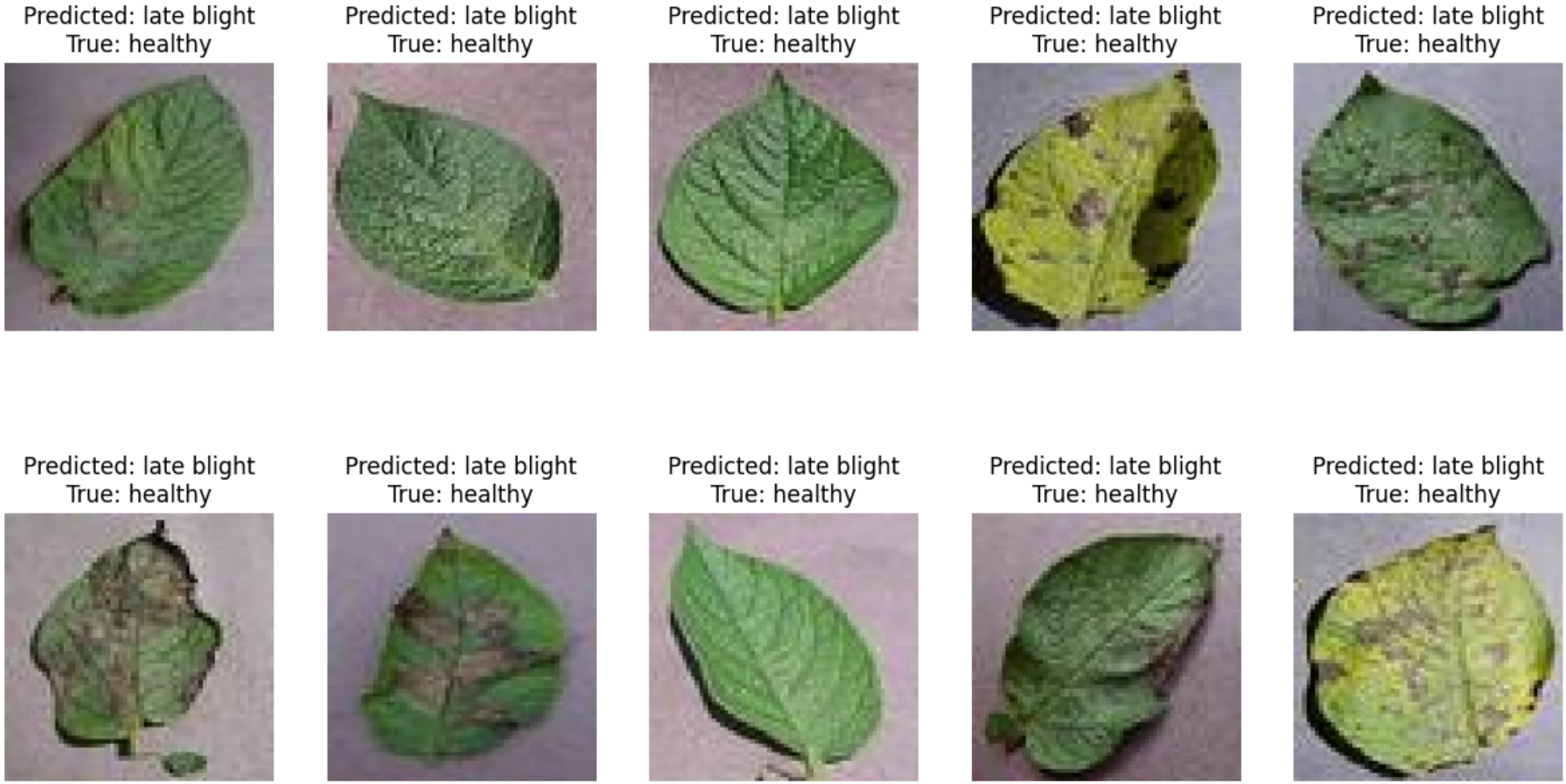

```
y_pred_prob = model.predict(validation_generator)

y_pred = np.argmax(y_pred_prob, axis=1)

y_true = validation_generator.classes

conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
26/26 [==============================] - 89s 3s/step
Confusion Matrix:
[[148  23 175]
```

```
[ 45   9  38]
[154  39 173]]
```

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

# Define class names
class_names = ['healthy', 'early blight', 'late blight']

# Retrieve images and true labels from validation_generator
validation_images, validation_labels = [], []
for i in range(len(validation_generator)):
    images, labels = validation_generator[i]
    validation_images.append(images)
    validation_labels.append(labels)

validation_images = np.concatenate(validation_images)
validation_labels = np.concatenate(validation_labels)

# Indices of misclassified images
misclassified_indices = np.where(y_pred != y_true)[0]

# Visualize a sample of misclassified images
num_samples = min(len(misclassified_indices), 10)  # Change 10 to visualize more or fewer samples
plt.figure(figsize=(15, 8))
for i, idx in enumerate(misclassified_indices[:num_samples]):
    plt.subplot(2, num_samples // 2, i + 1)
    plt.imshow(validation_images[idx])
    plt.title(f"Predicted: {class_names[y_pred[idx]]}\nTrue: {class_names[y_true[idx]]}")
    plt.axis('off')
plt.show()

# Error analysis
misclassified_labels = y_pred[misclassified_indices]
true_labels = y_true[misclassified_indices]

unique_misclassified_labels, misclassified_counts = np.unique(misclassified_labels, return_counts=True)
unique_true_labels, true_counts = np.unique(true_labels, return_counts=True)

print("Misclassifications by Predicted Class:")
```

```
for label, count in zip(unique_misclassified_labels, misclassified_counts):
    print(f"Class: {class_names[label]}, Misclassifications: {count}")

print("\nMisclassifications by True Class:")
for label, count in zip(unique_true_labels, true_counts):
    print(f"Class: {class_names[label]}, Misclassifications: {count}")
```




```
Misclassifications by Predicted Class:
Class: healthy, Misclassifications: 199
Class: early blight, Misclassifications: 62
Class: late blight, Misclassifications: 213

Misclassifications by True Class:
Class: healthy, Misclassifications: 198
Class: early blight, Misclassifications: 83
Class: late blight, Misclassifications: 193
```

4. Model Evaluation: Assess the performance of the trained model using a test dataset, evaluating metrics such as accuracy, recall, etc.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns


# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
# Define the image size
image_size = 224  # Placeholder value, replace with the appropriate size

# Load additional test dataset (replace 'test_data_dir' with the appropriate directory)
test_data_dir = '/content/drive/My Drive/InceptionV3/Test/CompressedSharpen/'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    test_data_dir,
    target_size=(image_size, image_size),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred = model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred, axis=1)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
Found 310 images belonging to 3 classes.
10/10 [==============================] - 33s 3s/step
Confusion Matrix:
[[25 37 38]
 [ 7 47 46]
 [12 43 55]]
```

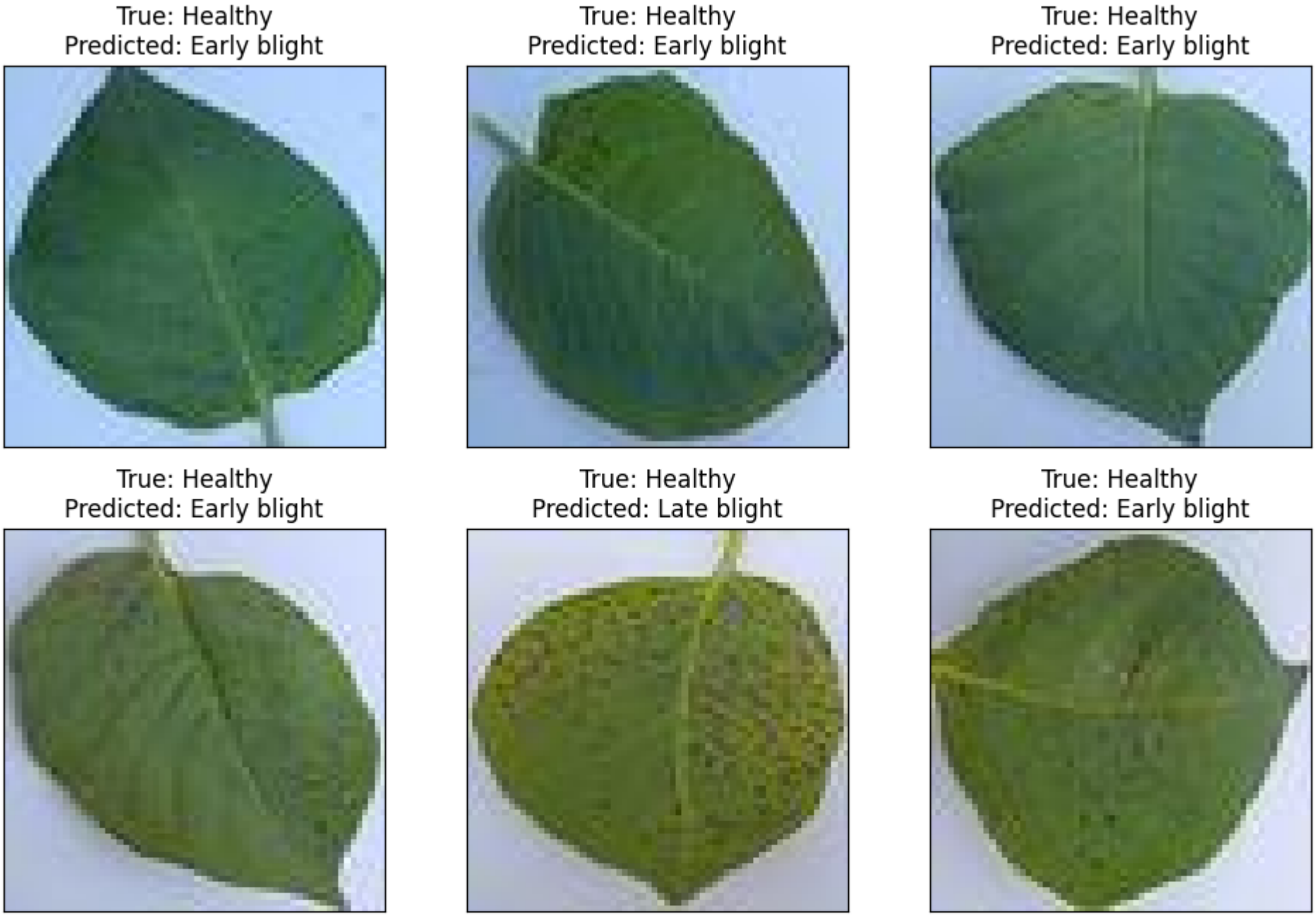
```
from sklearn.metrics import accuracy_score

# Calculate accuracy
accuracy = accuracy_score(y_test_true, y_test_pred)

print("Accuracy:", accuracy)
```

```
Accuracy: 0.4096774193548387
```

```
import numpy as np

# Use your trained model to predict labels for the test dataset
predictions = model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]
```

```
10/10 [==============================] - 42s 4s/step
```

```
# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names
class_names = ['Healthy', 'Early blight', 'Late blight']

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()
```


True: Healthy
Predicted: Early blight
True: Healthy
Predicted: Early blight
True: Healthy
Predicted: Early blight
True: Healthy
Predicted: Early blight
True: Healthy
Predicted: Late blight
True: Healthy
Predicted: Early blight

VGG16 Model

- **Model Architecture**: VGG16 The choice of the convolutional neural network (CNN) architecture plays a crucial role in the success of image classification tasks. In this project, we opted to employ the VGG16 model, a well-established deep learning architecture proposed by the Visual Geometry Group at the University of Oxford.
- **Overview of VGG16**: The VGG16 model is characterized by its simplicity and effectiveness. It consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. Notably, VGG16 gained popularity for its uniform architecture, where each convolutional layer has a 3x3 filter size and is followed by a max-pooling layer.

Utilization in the Project

- **Transfer Learning**: We harnessed the power of transfer learning by leveraging the pre-trained weights of the VGG16 model on the ImageNet dataset. The lower layers of VGG16, having learned general features from a diverse range of images, served as a valuable feature extractor.
- **Feature Extraction**: The convolutional layers of VGG16 were employed to capture hierarchical representations of visual features in our specific dataset. These layers act as powerful feature extractors, enabling the model to recognize patterns related to the health conditions of potato leaves.
- **Fine-Tuning**: We fine-tuned the model by removing the top (fully connected) layers of VGG16 and appending our own layers for classification. This process allowed the model to adapt its learned features to our specific classification task, encompassing categories such as early blight, healthy, and late blight.

Advantages of VGG16 in the Project

- **Simplicity**: The straightforward architecture of VGG16 simplified the integration of the model into our project, facilitating both implementation and understanding.
- **Transfer Learning Benefits**: Leveraging pre-trained weights from VGG16 expedited the convergence of the model during training, especially considering the limited size of our dataset.

Import Necessary Libraries: Start by importing the required libraries in your Python script or Jupyter notebook. This typically includes TensorFlow (or any other deep learning library), Keras (if using TensorFlow), and other relevant modules.

```
!pip install tensorflow
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models
from tensorflow.keras.models import Model
```

Set Up Data Directories: Define the paths to the directories containing your training and validation images. Ensure that the images are organized into subdirectories based on their classes (e.g., 'early_blight', 'healthy', 'late_blight').

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

Part 1: Training Model

Original

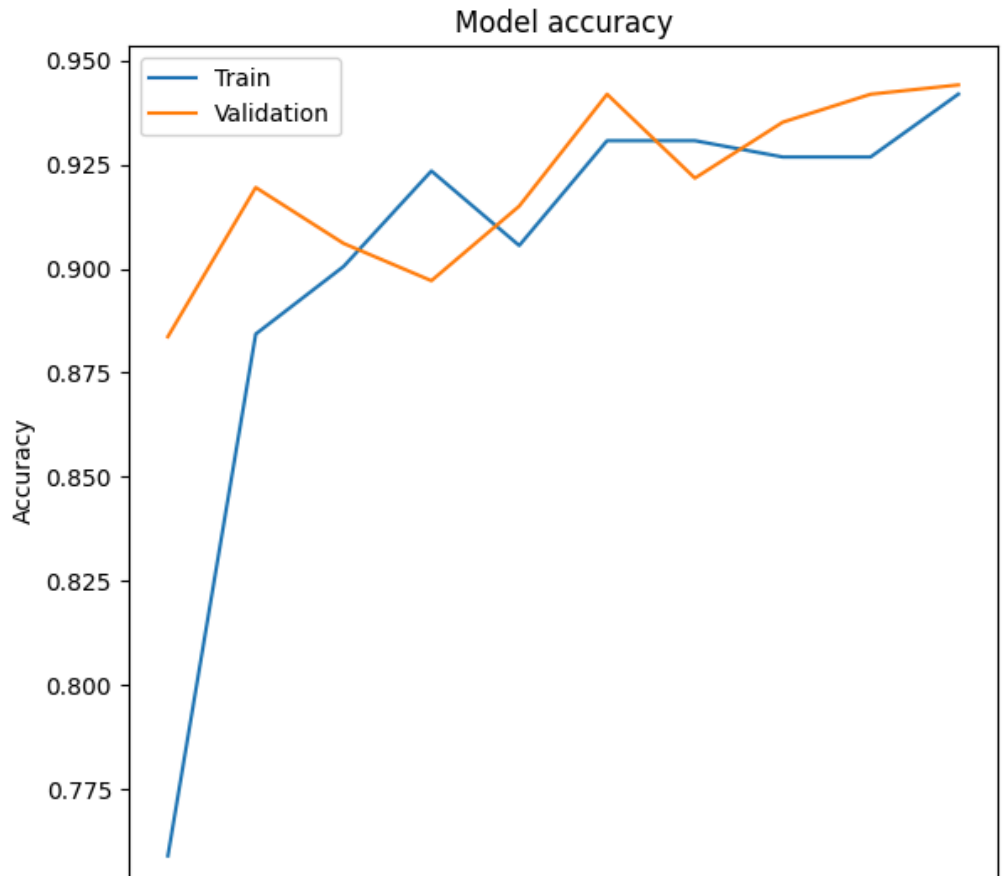
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models
from tensorflow.keras.models import Model

from google.colab import drive
drive.mount('/content/drive')
# 设置文件夹路径
import os

original_path = '/content/drive/My Drive/original'
files_in_folder = os.listdir(original_path)
files_in_folder
# Setting Image Size and Batch Size:
# Define the target image size for preprocessing (224x224 pixels).
# Set the batch size for training and validation data.
image_size = (224, 224)
batch_size = 32

# Configuring Image Data Generator:
# Use ImageDataGenerator to perform data augmentation on the training data.
# Rescaling the pixel values to a range between 0 and 1.
# Applying shear range, zoom range, and horizontal flip for augmentation.
# Allocating 80% of the data for training and 20% for validation (validation_split=0.2).
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   validation_split=0.2)  # 80%train，20%test

# Creating Training Data Generator:
# Use flow_from_directory to create a data generator for training data.
# Specify the directory path, target image size, batch size, class mode as 'categorical' and set the subset to 'training'.
train_generator = train_datagen.flow_from_directory(
    original_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

# Creating Validation Data Generator:
# Create another data generator for validation data using the same directory, image size, and batch size.
# Set the class mode as 'categorical' and the subset to 'validation'.
validation_generator = train_datagen.flow_from_directory(
    original_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
```

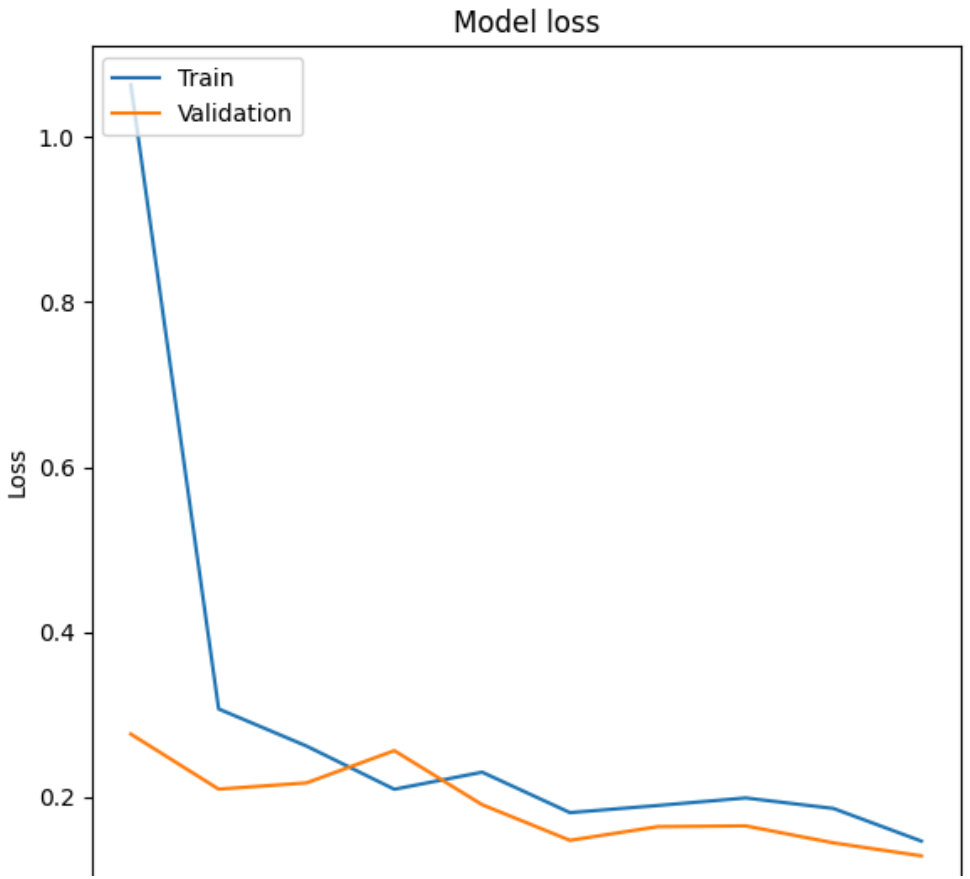
```
    subset='validation'
)

# Loading Pre-trained VGG16 Model:
# Load the VGG16 model with pre-trained weights on ImageNet.
# Exclude the top (fully connected) layers, as indicated by include_top=False.
# Specify the input shape as (224, 224, 3).
original_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freezing Pre-trained Layers:
# Iterate through each layer in the original VGG16 model and set their trainable attribute to False, effectively freezing the pre-trained weigh
for layer in original_model.layers:
    layer.trainable = False

# Building Custom Model on Top of VGG16:
# Flatten the output of the pre-trained VGG16 model.
# Add a fully connected (dense) layer with 256 units and ReLU activation.
# Introduce dropout with a rate of 0.5 to prevent overfitting.
# Add the final dense layer with 3 units and softmax activation (assuming a classification task with 3 classes).
x = layers.Flatten()(original_model.output)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(3, activation='softmax')(x)

# Creating the Modified Model:
# Create a new model (ori_model) using the modified output (x) and the original VGG16 model's input.
ori_model = Model(original_model.input, x)

# Compiling the Model:
# Compile the modified model with the Adam optimizer, categorical crossentropy loss, and accuracy as the metric.
ori_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Setting Number of Epochs:
# Define the number of training epochs (in this case, 10).
epochs = 10

# Training the Model:
# Use the fit method to train the model on the training data (train_generator).
# Specify the number of epochs and the validation data (validation_generator) for monitoring model performance during training.
history_2 = ori_model.fit(train_generator, epochs=epochs, validation_data=validation_generator)
ori_model.save('/content/drive/My Drive/original/ori_model.h5')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 1789 images belonging to 3 classes.
Found 447 images belonging to 3 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
Epoch 1/10
56/56 [==============================] - 1490s 26s/step - loss: 1.0633 - accuracy: 0.7591 - val_loss: 0.2762 - val_accuracy: 0.8837
Epoch 2/10
```

```
56/56 [==============================] - 1439s 26s/step - loss: 0.3066 - accuracy: 0.8843 - val_loss: 0.2094 - val_accuracy: 0.9195
Epoch 3/10
56/56 [==============================] - 1431s 26s/step - loss: 0.2616 - accuracy: 0.9005 - val_loss: 0.2169 - val_accuracy: 0.9060
Epoch 4/10
56/56 [==============================] - 1455s 26s/step - loss: 0.2092 - accuracy: 0.9234 - val_loss: 0.2561 - val_accuracy: 0.8971
Epoch 5/10
56/56 [==============================] - 1431s 26s/step - loss: 0.2299 - accuracy: 0.9055 - val_loss: 0.1905 - val_accuracy: 0.9150
Epoch 6/10
56/56 [==============================] - 1425s 26s/step - loss: 0.1809 - accuracy: 0.9307 - val_loss: 0.1473 - val_accuracy: 0.9418
Epoch 7/10
56/56 [==============================] - 1432s 26s/step - loss: 0.1895 - accuracy: 0.9307 - val_loss: 0.1639 - val_accuracy: 0.9217
Epoch 8/10
56/56 [==============================] - 1413s 25s/step - loss: 0.1987 - accuracy: 0.9268 - val_loss: 0.1648 - val_accuracy: 0.9351
Epoch 9/10
56/56 [==============================] - 1416s 25s/step - loss: 0.1859 - accuracy: 0.9268 - val_loss: 0.1442 - val_accuracy: 0.9418
Epoch 10/10
56/56 [==============================] - 1415s 25s/step - loss: 0.1463 - accuracy: 0.9419 - val_loss: 0.1285 - val_accuracy: 0.9441
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt
# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history_2.history['accuracy'])
plt.plot(history_2.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()
```





Original dataset model training observations:

- **Loss and Accuracy Trends:** In the first epoch, the training loss is relatively high, but it decreases significantly in subsequent epochs. Both training and validation accuracy show an increasing trend, which is a positive sign.
- **Overfitting Check:** The training and validation loss converge well, indicating that the model is not heavily overfitting. The training and validation accuracy are close, suggesting that the model generalizes well to unseen data.
- **Model Performance:** The final accuracy on the validation set is around 94.41%, which is quite good. The accuracy on the training set is around 94.19%.
- **Training Duration:** Each epoch takes a significant amount of time (more than 20 minutes per epoch). This might be due to the complex nature of the VGG16 model and the dataset size.
- **Suggestions:** The model is performing well and not overfitting, you might consider training for more epochs to see if the performance further improves. Monitor training and validation curves over additional epochs to ensure stability and avoid overfitting.

Compressed

```
base_path = '/content/drive/My Drive/IEOR 243 Analytics Lab/module2/图片数据/compressed'
files_in_folder = os.listdir(base_path)
files_in_folder
```

```
['Potato___Late_blight', 'Potato___healthy', 'Potato___Early_blight']
```

```
image_size = (224, 224)
batch_size = 32

train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   validation_split=0.2)  # 80%train, 20%test

train_generator = train_datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 1789 images belonging to 3 classes.
Found 447 images belonging to 3 classes.
```

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

x = layers.Flatten()(base_model.output)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(3, activation='softmax')(x)  # early_blight, healthy, late_blight

model = Model(base_model.input, x)

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

```
epochs = 10


# train
comphistory = model.fit(train_generator, epochs=epochs, validation_data=validation_generator)
```

```
Epoch 1/10
56/56 [==============================] - 1521s 27s/step - loss: 1.0928 - accuracy: 0.7272 - val_loss: 0.3705 - val_accuracy: 0.8523
Epoch 2/10
56/56 [==============================] - 1470s 26s/step - loss: 0.3481 - accuracy: 0.8720 - val_loss: 0.3282 - val_accuracy: 0.8837
Epoch 3/10
56/56 [==============================] - 1465s 26s/step - loss: 0.3005 - accuracy: 0.8938 - val_loss: 0.2707 - val_accuracy: 0.8971
Epoch 4/10
56/56 [==============================] - 1432s 26s/step - loss: 0.2564 - accuracy: 0.9033 - val_loss: 0.3666 - val_accuracy: 0.8747
Epoch 5/10
56/56 [==============================] - 1467s 26s/step - loss: 0.2615 - accuracy: 0.9055 - val_loss: 0.2622 - val_accuracy: 0.8949
Epoch 6/10
56/56 [==============================] - 1470s 26s/step - loss: 0.2468 - accuracy: 0.8960 - val_loss: 0.2334 - val_accuracy: 0.9195
Epoch 7/10
56/56 [==============================] - 1472s 26s/step - loss: 0.2465 - accuracy: 0.9089 - val_loss: 0.2431 - val_accuracy: 0.9038
Epoch 8/10
56/56 [==============================] - 1473s 26s/step - loss: 0.2092 - accuracy: 0.9234 - val_loss: 0.2141 - val_accuracy: 0.9284
Epoch 9/10
56/56 [==============================] - 1504s 27s/step - loss: 0.2092 - accuracy: 0.9173 - val_loss: 0.3040 - val_accuracy: 0.8770
Epoch 10/10
56/56 [==============================] - 1473s 26s/step - loss: 0.2165 - accuracy: 0.9201 - val_loss: 0.2069 - val_accuracy: 0.9262
```

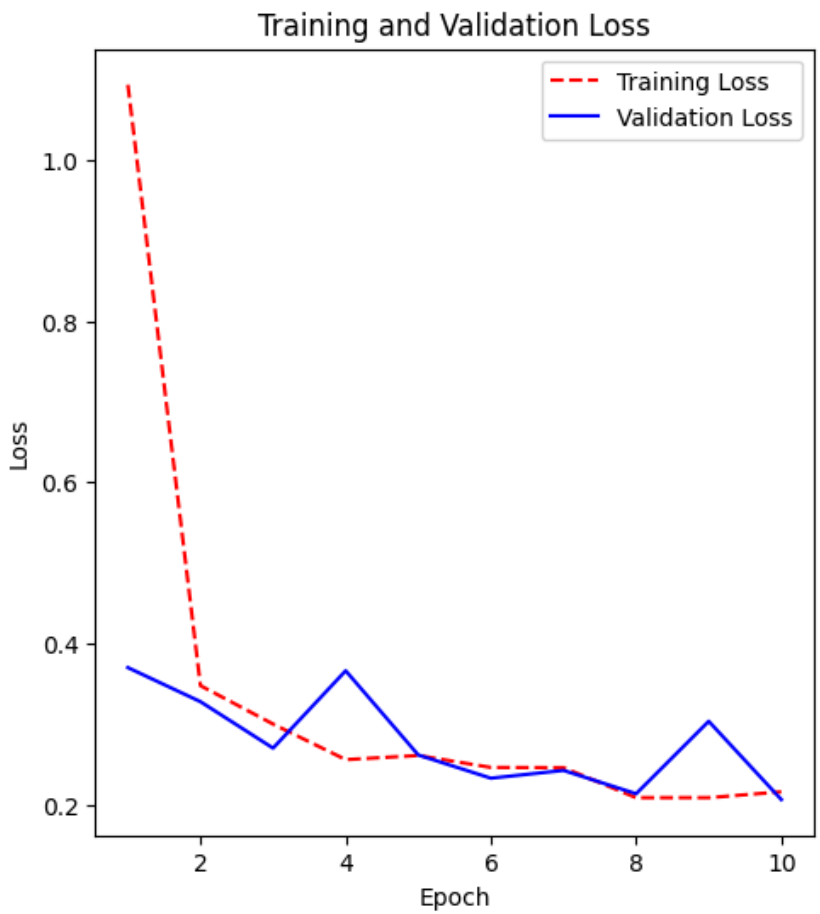
Compressed dataset model training observations:

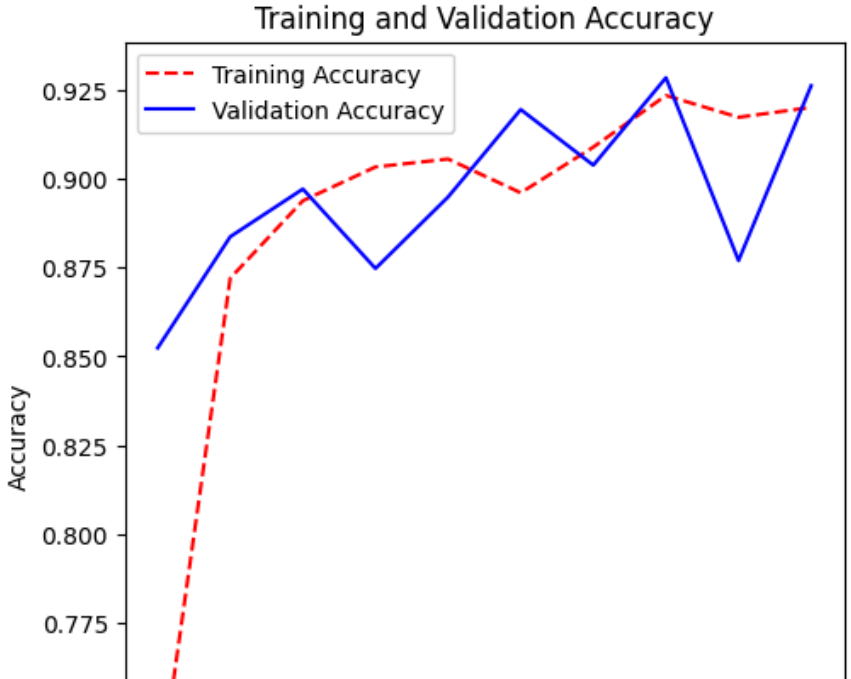
- Both training loss and training accuracy show consistent improvements over epochs.
- Towards the later epochs, the rate of improvement in validation metrics starts to slow down, which might suggest that the model is nearing its optimal performance, and further training epochs may not yield significant improvements.

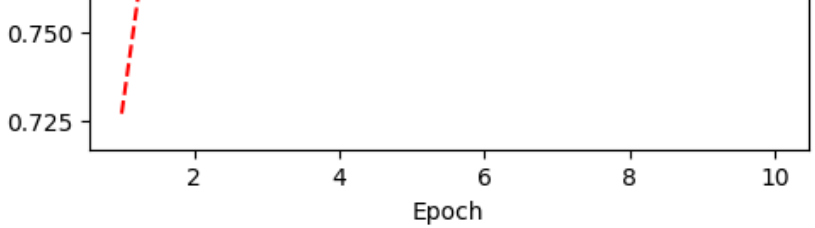
```
model.save('/content/drive/My Drive/243 Analytics Lab/compressed/my_vgg_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt

# Create count of the number of epochs
epoch_count = range(1, epochs + 1)

# Extract loss history from modelinceptionv3cs object
training_loss = comphistory.history['loss']
validation_loss = comphistory.history['val_loss']

# Visualize loss history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_count, training_loss, 'r--', label='Training Loss')
plt.plot(epoch_count, validation_loss, 'b-', label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()

# Extract loss history from modelinceptionv3cs object
training_acc = comphistory.history['accuracy']
validation_acc = comphistory.history['val_accuracy']

# Visualize accuracy history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
plt.plot(epoch_count, training_acc, 'r--')
plt.plot(epoch_count, validation_acc, 'b-')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.show()
```


Training and Validation Loss
Training Loss
Validation Loss
1.0
0.8
0.6
0.4
0.2
Loss
2
4
6
8
10
Epoch


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.925
0.900
0.875
0.850
0.825
0.800
0.775
Accuracy




```
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix


# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
14/14 [==============================] - 292s 21s/step
Confusion Matrix:
[[86 98 16]
 [88 88 24]
 [20 16 11]]
```

Sharpen

```
base_path = '/content/drive/My Drive/IEOR 243 Analytics Lab/module2/图片数据/sharpen'

files_in_folder = os.listdir(base_path)
files_in_folder
```

```
['Potato___Late_blight', 'Potato___healthy', 'Potato___Early_blight']
```

```
image_size = (224, 224)
batch_size = 32

train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   validation_split=0.2)  # 80%train, 20%test

train_generator = train_datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 1789 images belonging to 3 classes.
Found 447 images belonging to 3 classes.
```

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

x = layers.Flatten()(base_model.output)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(3, activation='softmax')(x)  # early_blight, healthy, late_blight

model = Model(base_model.input, x)

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

```
epochs = 10
```

```
# train
sharphistory = model.fit(train_generator, epochs=epochs, validation_data=validation_generator)
```

```
Epoch 1/10
56/56 [==============================] - 1240s 22s/step - loss: 1.3122 - accuracy: 0.7188 - val_loss: 0.2960 - val_accuracy: 0.8904
Epoch 2/10
56/56 [==============================] - 1236s 22s/step - loss: 0.3407 - accuracy: 0.8770 - val_loss: 0.2466 - val_accuracy: 0.9150
Epoch 3/10
56/56 [==============================] - 1235s 22s/step - loss: 0.2766 - accuracy: 0.8949 - val_loss: 0.2339 - val_accuracy: 0.9150
Epoch 4/10
56/56 [==============================] - 1236s 22s/step - loss: 0.2192 - accuracy: 0.9145 - val_loss: 0.4053 - val_accuracy: 0.8367
Epoch 5/10
56/56 [==============================] - 1236s 22s/step - loss: 0.2643 - accuracy: 0.9027 - val_loss: 0.1689 - val_accuracy: 0.9485
Epoch 6/10
56/56 [==============================] - 1236s 22s/step - loss: 0.2293 - accuracy: 0.9184 - val_loss: 0.1903 - val_accuracy: 0.9306
Epoch 7/10
56/56 [==============================] - 1238s 22s/step - loss: 0.2057 - accuracy: 0.9279 - val_loss: 0.1909 - val_accuracy: 0.9351
Epoch 8/10
56/56 [==============================] - 1237s 22s/step - loss: 0.1878 - accuracy: 0.9262 - val_loss: 0.2421 - val_accuracy: 0.8904
Epoch 9/10
56/56 [==============================] - 1218s 22s/step - loss: 0.1828 - accuracy: 0.9363 - val_loss: 0.1764 - val_accuracy: 0.9374
Epoch 10/10
56/56 [==============================] - 1237s 22s/step - loss: 0.1800 - accuracy: 0.9312 - val_loss: 0.1668 - val_accuracy: 0.9284
```

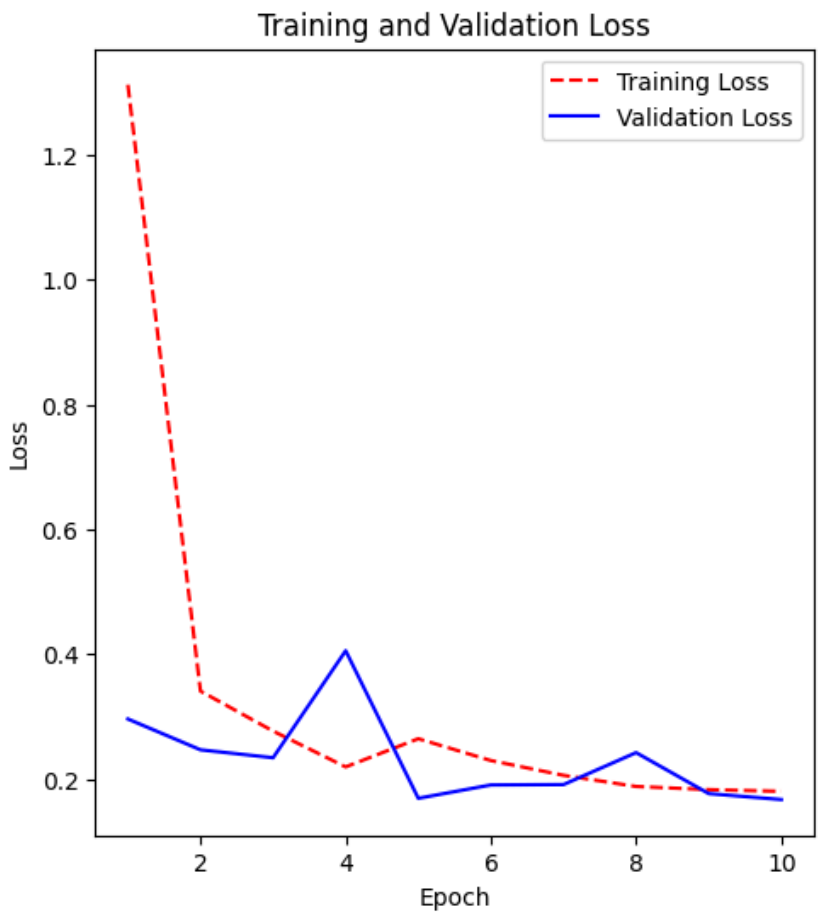
Sharpen dataset model training observations:

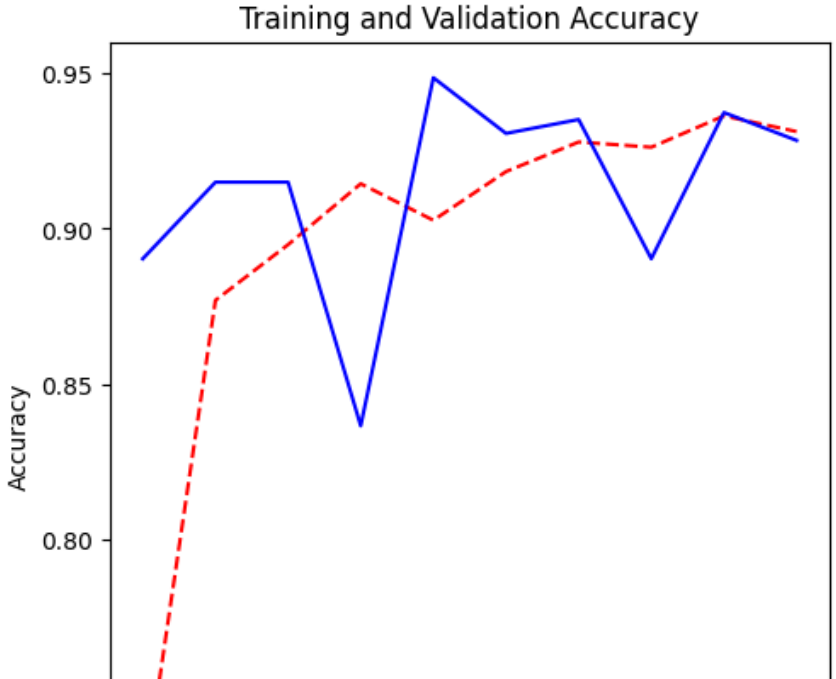
- For Epoch 6, there is a sudden increase in validation loss and decrease in validation accuracy, which suggests a degradation in performance. This could be a sign of overfitting or other issues in the model.

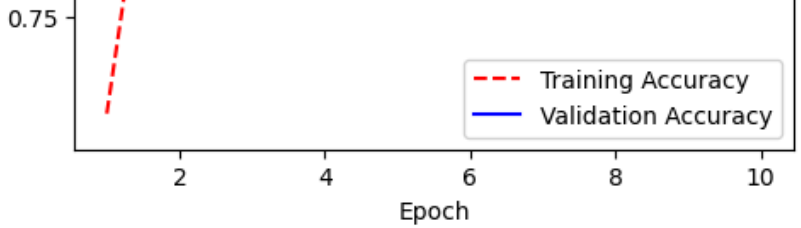
```
model.save('/content/drive/My Drive/243 Analytics Lab/sharpen/my_vgg_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt

# Create count of the number of epochs
epoch_count = range(1, epochs + 1)

# Extract loss history from modelinceptionv3cs object
training_loss = sharphistory.history['loss']
validation_loss = sharphistory.history['val_loss']

# Visualize loss history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_count, training_loss, 'r--', label='Training Loss')
plt.plot(epoch_count, validation_loss, 'b-', label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()

# Extract loss history from modelinceptionv3cs object
training_acc = sharphistory.history['accuracy']
validation_acc = sharphistory.history['val_accuracy']

# Visualize accuracy history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
plt.plot(epoch_count, training_acc, 'r--')
plt.plot(epoch_count, validation_acc, 'b-')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.show()
```


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
1.2
1.0
0.8
0.6
0.4
0.2
2
4
6
8
10


Training and Validation Accuracy
Accuracy
0.95
0.90
0.85
0.80




```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import numpy as np


# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
14/14 [==============================] - 258s 18s/step
Confusion Matrix:
[[95 81 24]
 [92 83 25]
 [24 17  6]]
```

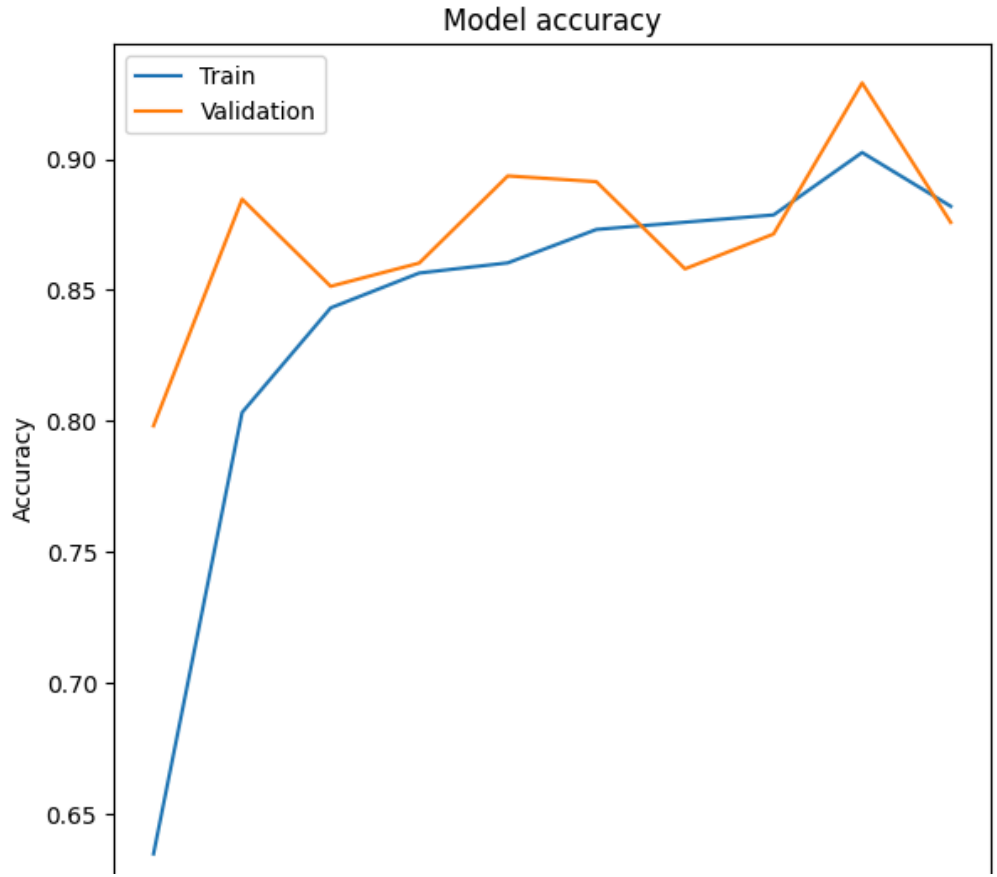
Sharpen+Compressed

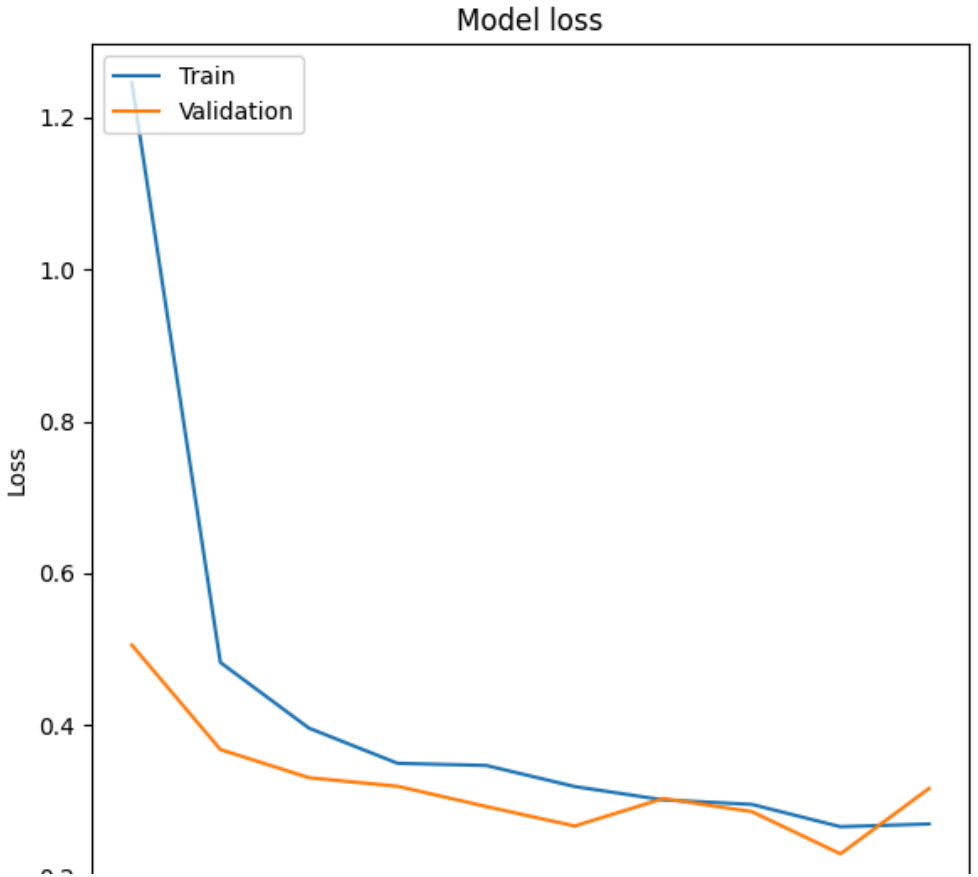
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models
from tensorflow.keras.models import Model

from google.colab import drive
drive.mount('/content/drive')
import os
base_path = '/content/drive/My Drive/CS'

files_in_folder = os.listdir(base_path)
files_in_folder

image_size = (224, 224)
batch_size = 32

train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

x = layers.Flatten()(base_model.output)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(3, activation='softmax')(x)

model = Model(base_model.input, x)
```

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

epochs = 10
history_1 = model.fit(train_generator, epochs=epochs, validation_data=validation_generator)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 1805 images belonging to 3 classes.
Found 451 images belonging to 3 classes.
Epoch 1/10
57/57 [==============================] - 1498s 26s/step - loss: 1.2456 - accuracy: 0.6349 - val_loss: 0.5057 - val_accuracy: 0.7982
Epoch 2/10
57/57 [==============================] - 1528s 27s/step - loss: 0.4827 - accuracy: 0.8033 - val_loss: 0.3681 - val_accuracy: 0.8847
Epoch 3/10
57/57 [==============================] - 1490s 26s/step - loss: 0.3962 - accuracy: 0.8432 - val_loss: 0.3310 - val_accuracy: 0.8514
Epoch 4/10
57/57 [==============================] - 1485s 26s/step - loss: 0.3499 - accuracy: 0.8565 - val_loss: 0.3197 - val_accuracy: 0.8603
Epoch 5/10
57/57 [==============================] - 1489s 26s/step - loss: 0.3471 - accuracy: 0.8604 - val_loss: 0.2931 - val_accuracy: 0.8936
Epoch 6/10
57/57 [==============================] - 1487s 26s/step - loss: 0.3193 - accuracy: 0.8731 - val_loss: 0.2674 - val_accuracy: 0.8914
Epoch 7/10
57/57 [==============================] - 1493s 26s/step - loss: 0.3018 - accuracy: 0.8759 - val_loss: 0.3036 - val_accuracy: 0.8581
Epoch 8/10
57/57 [==============================] - 1494s 26s/step - loss: 0.2958 - accuracy: 0.8787 - val_loss: 0.2864 - val_accuracy: 0.8714
Epoch 9/10
57/57 [==============================] - 1491s 26s/step - loss: 0.2665 - accuracy: 0.9025 - val_loss: 0.2307 - val_accuracy: 0.9290
Epoch 10/10
57/57 [==============================] - 1461s 26s/step - loss: 0.2701 - accuracy: 0.8820 - val_loss: 0.3169 - val_accuracy: 0.8758
```

```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history_1.history['accuracy'])
plt.plot(history_1.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history_1.history['loss'])
plt.plot(history_1.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()
```





compressed+sharpen dataset model trainning observations:

- **Loss and Accuracy Trends:** The training loss starts relatively high and gradually decreases over the epochs.Training accuracy increases, indicating that the model is learning from the data. Validation loss and accuracy show fluctuations, which might suggest that the model could benefit from further tuning.
- **Overfitting Check:** The training and validation curves are somewhat diverging, especially in later epochs, indicating potential overfitting. Monitoring overfitting is crucial; consider techniques like regularization, dropout, or reducing model complexity.
- **Model Performance:** The final accuracy on the validation set is around 87.58%, which is decent. The accuracy on the training set is around 88.20%.
- **Training Duration:** Each epoch still takes a significant amount of time, possibly due to the large dataset and complex model architecture.
- **Suggestions:** Given the signs of overfitting, consider incorporating dropout layers or regularization to improve generalization.Experiment with learning rate schedules or early stopping to fine-tune the training process.

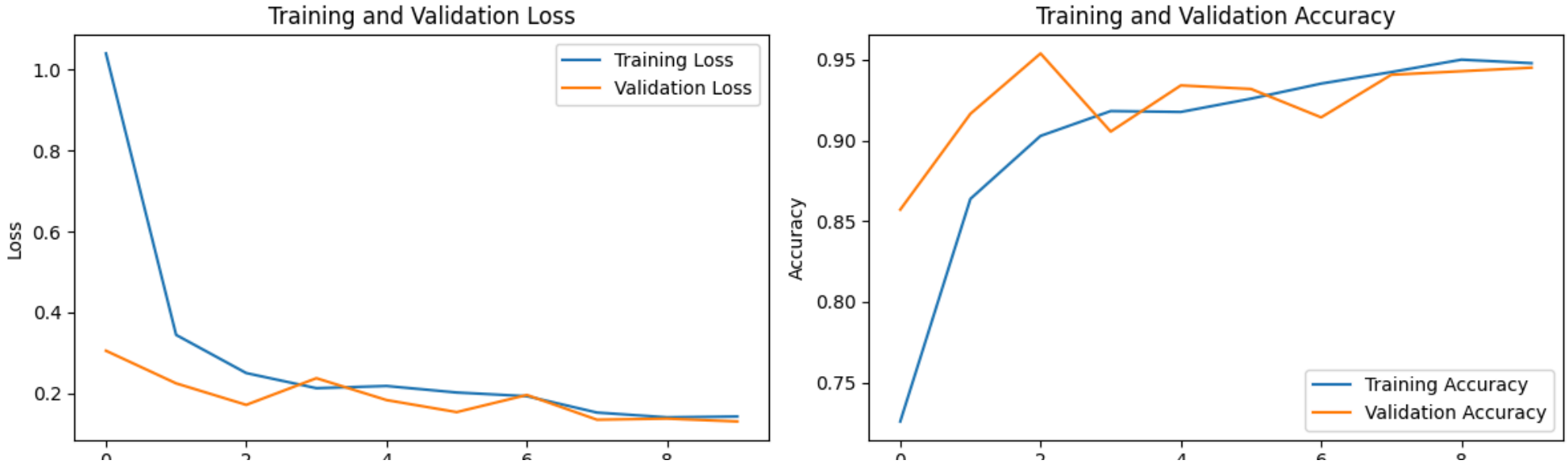
Flipped

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models
from tensorflow.keras.models import Model
import os
from google.colab import drive
drive.mount('/content/drive')

flipped_path = '/content/drive/My Drive/Flipped'

image_size = (224, 224)
batch_size = 32

train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    flipped_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    flipped_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

original_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in original_model.layers:
    layer.trainable = False

x = layers.Flatten()(original_model.output)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(3, activation='softmax')(x)

fli_model = Model(original_model.input, x)
```

```
# 编译模型
fli_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 定义训练轮数
epochs = 10

# 训练模型
history_3 = fli_model.fit(train_generator, epochs=epochs, validation_data=validation_generator)

# 保存模型
fli_model.save('/content/drive/My Drive/Flipped/fli_model.h5')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 1821 images belonging to 3 classes.
Found 455 images belonging to 3 classes.
Epoch 1/10
57/57 [==============================] - 1494s 26s/step - loss: 1.0399 - accuracy: 0.7260 - val_loss: 0.3053 - val_accuracy: 0.8571
Epoch 2/10
57/57 [==============================] - 1476s 26s/step - loss: 0.3444 - accuracy: 0.8638 - val_loss: 0.2247 - val_accuracy: 0.9165
Epoch 3/10
57/57 [==============================] - 1471s 26s/step - loss: 0.2501 - accuracy: 0.9028 - val_loss: 0.1716 - val_accuracy: 0.9538
Epoch 4/10
57/57 [==============================] - 1468s 26s/step - loss: 0.2128 - accuracy: 0.9182 - val_loss: 0.2374 - val_accuracy: 0.9055
Epoch 5/10
57/57 [==============================] - 1431s 25s/step - loss: 0.2182 - accuracy: 0.9176 - val_loss: 0.1834 - val_accuracy: 0.9341
Epoch 6/10
57/57 [==============================] - 1435s 25s/step - loss: 0.2021 - accuracy: 0.9259 - val_loss: 0.1536 - val_accuracy: 0.9319
Epoch 7/10
57/57 [==============================] - 1476s 26s/step - loss: 0.1930 - accuracy: 0.9352 - val_loss: 0.1960 - val_accuracy: 0.9143
Epoch 8/10
57/57 [==============================] - 1469s 26s/step - loss: 0.1525 - accuracy: 0.9423 - val_loss: 0.1349 - val_accuracy: 0.9407
Epoch 9/10
57/57 [==============================] - 1439s 25s/step - loss: 0.1407 - accuracy: 0.9500 - val_loss: 0.1375 - val_accuracy: 0.9429
Epoch 10/10
57/57 [==============================] - 1435s 25s/step - loss: 0.1429 - accuracy: 0.9478 - val_loss: 0.1305 - val_accuracy: 0.9451
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt

# Plot training and validation loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_3.history['loss'], label='Training Loss')
plt.plot(history_3.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(history_3.history['accuracy'], label='Training Accuracy')
plt.plot(history_3.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



flipped dataset model training obervartions:

- **Training Progress:** The training is conducted over 10 epochs. Each epoch consists of 57 batches (based on the batch size specified during data generation). The training duration for each epoch is quite long, with each epoch taking over 20 minutes.
- **Loss and Accuracy Trends:** The training loss starts relatively high in the first epoch (1.0399) and decreases significantly in subsequent epochs. Both training and validation accuracy show an increasing trend, which is a positive sign. The final accuracy on the validation set is

around 94.51%.

- **Overfitting Check:** The training and validation loss converge well, indicating that the model is not heavily overfitting. The training and validation accuracy are close, suggesting that the model generalizes well to unseen data.
- **Model Saving Warning:** There is a warning at the end about saving the model in HDF5 format, which is considered legacy. The recommendation is to save the model in the native Keras format.
- **Suggestions:* The model appears to be well-trained, and the validation accuracy is quite good. Consider monitoring training and validation curves over additional epochs to ensure stability and avoid overfitting. Due to the long training time per epoch, consider optimizing the model architecture or training strategy to reduce computational time.

Flipped+Compressed

```
base_path = '/content/drive/My Drive/IEOR 243 Analytics Lab/module2/图片数据/flipped+compressed'
files_in_folder = os.listdir(base_path)
files_in_folder
```

```
['Potato___healthy', 'Potato___Early_blight', 'Potato___Late_blight']
```

```
image_size = (224, 224)
batch_size = 32

train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   validation_split=0.2)  # 80%train, 20%test

train_generator = train_datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 1789 images belonging to 3 classes.
Found 447 images belonging to 3 classes.
```

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

x = layers.Flatten()(base_model.output)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(3, activation='softmax')(x)  # early_blight, healthy, late_blight

model = Model(base_model.input, x)

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

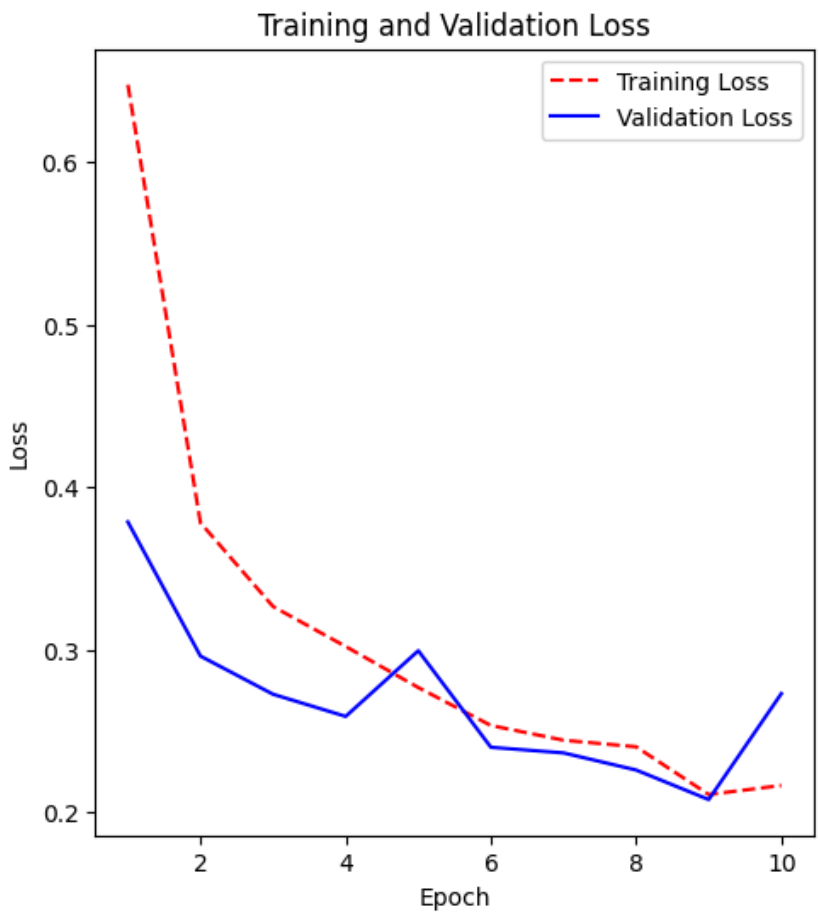
```
    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
    58889256/58889256 [==============================] - 0s 0us/step
```

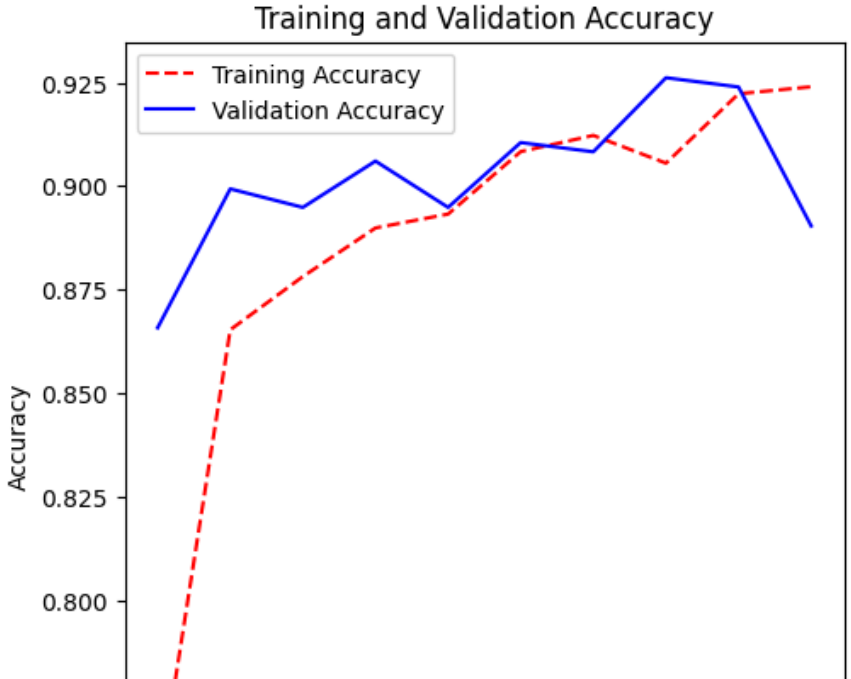
```
epochs = 10


# train
fchistory = model.fit(train_generator, epochs=epochs, validation_data=validation_generator)
```

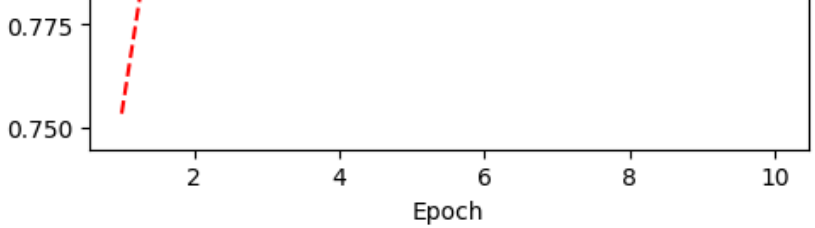
```
    Epoch 1/10
    56/56 [==============================] - 1320s 23s/step - loss: 0.6478 - accuracy: 0.7535 - val_loss: 0.3787 - val_accuracy: 0.8658
    Epoch 2/10
    56/56 [==============================] - 1244s 22s/step - loss: 0.3782 - accuracy: 0.8653 - val_loss: 0.2962 - val_accuracy: 0.8993
    Epoch 3/10
    56/56 [==============================] - 1246s 22s/step - loss: 0.3266 - accuracy: 0.8781 - val_loss: 0.2727 - val_accuracy: 0.8949
    Epoch 4/10
    56/56 [==============================] - 1247s 22s/step - loss: 0.3021 - accuracy: 0.8899 - val_loss: 0.2590 - val_accuracy: 0.9060
    Epoch 5/10
    56/56 [==============================] - 1247s 22s/step - loss: 0.2768 - accuracy: 0.8932 - val_loss: 0.2995 - val_accuracy: 0.8949
    Epoch 6/10
    56/56 [==============================] - 1244s 22s/step - loss: 0.2534 - accuracy: 0.9083 - val_loss: 0.2400 - val_accuracy: 0.9105
    Epoch 7/10
    56/56 [==============================] - 1259s 23s/step - loss: 0.2444 - accuracy: 0.9122 - val_loss: 0.2366 - val_accuracy: 0.9083
    Epoch 8/10
    56/56 [==============================] - 1247s 22s/step - loss: 0.2403 - accuracy: 0.9055 - val_loss: 0.2260 - val_accuracy: 0.9262
    Epoch 9/10
    56/56 [==============================] - 1249s 22s/step - loss: 0.2110 - accuracy: 0.9223 - val_loss: 0.2079 - val_accuracy: 0.9239
    Epoch 10/10
    56/56 [==============================] - 1248s 22s/step - loss: 0.2165 - accuracy: 0.9240 - val_loss: 0.2731 - val_accuracy: 0.8904
```

```
model.save('/content/drive/My Drive/243 Analytics Lab/flippedandcompressed/my_vgg_model.h5')
```

```
import matplotlib.pyplot as plt

# Create count of the number of epochs
epoch_count = range(1, epochs + 1)

# Extract loss history from flipped+compressed
training_loss = fchistory.history['loss']
validation_loss = fchistory.history['val_loss']

# Visualize loss history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_count, training_loss, 'r--', label='Training Loss')
plt.plot(epoch_count, validation_loss, 'b-', label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()

# Extract loss history from flipped+compressed
training_acc = fchistory.history['accuracy']
validation_acc = fchistory.history['val_accuracy']

# Visualize accuracy history
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
plt.plot(epoch_count, training_acc, 'r--')
plt.plot(epoch_count, validation_acc, 'b-')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.show()
```


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
0.2
0.3
0.4
0.5
0.6
2
4
6
8
10


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.800
0.825
0.850
0.875
0.900
0.925




```
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix


# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
14/14 [==============================] - 250s 18s/step
Confusion Matrix:
[[ 85  97  18]
 [ 69 119  12]
 [ 20  20   7]]
```

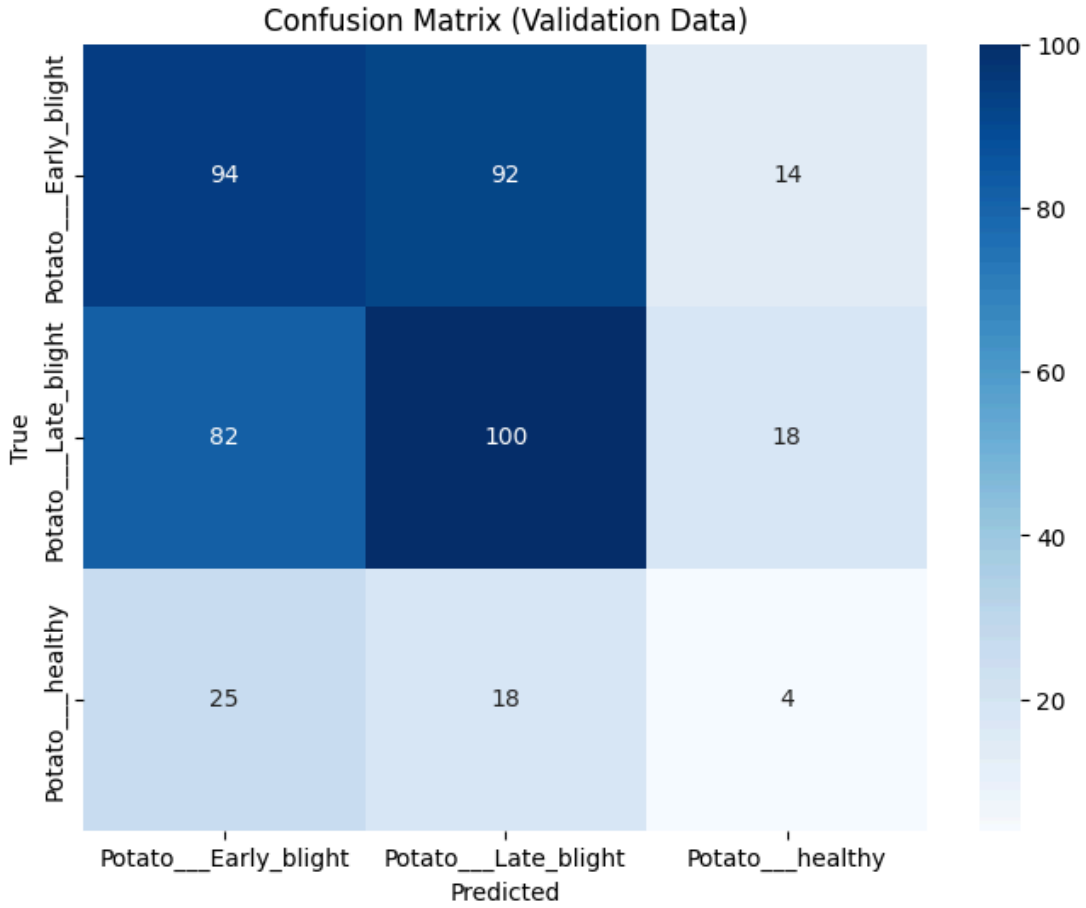
Part 2: Testing Model Using New Dataset

Original

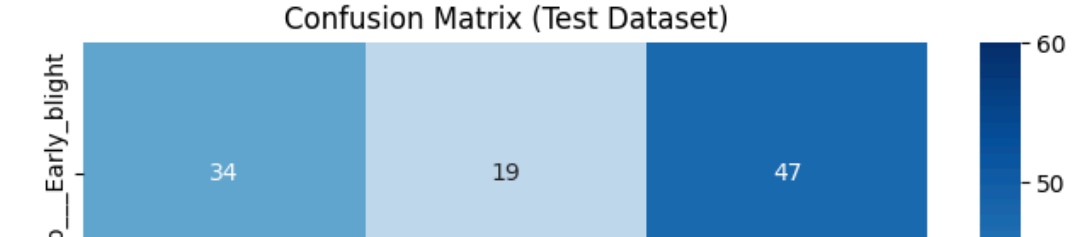
```
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the trained model
ori_model = load_model('/content/drive/My Drive/original/ori_model.h5')

# Predicted probabilities for validation data
y_pred_prob = ori_model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix for validation data
conf_matrix = confusion_matrix(y_true, y_pred)

# Load the training data directory to get actual class names
train_data_dir = '/content/drive/My Drive/original'  # Adjust the path to your training data
train_generator = ImageDataGenerator().flow_from_directory(
    train_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Get the actual class names
actual_class_names = list(train_generator.class_indices.keys())

# Plot confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=actual_class_names, yticklabels=actual_class_names)
plt.title('Confusion Matrix (Validation Data)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Generate classification report for validation data
class_report = classification_report(y_true, y_pred, target_names=actual_class_names)
print("Classification Report (Validation Data):")
print(class_report)

# Load additional test dataset
test_data_dir = '/content/drive/My Drive/original_new'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
```

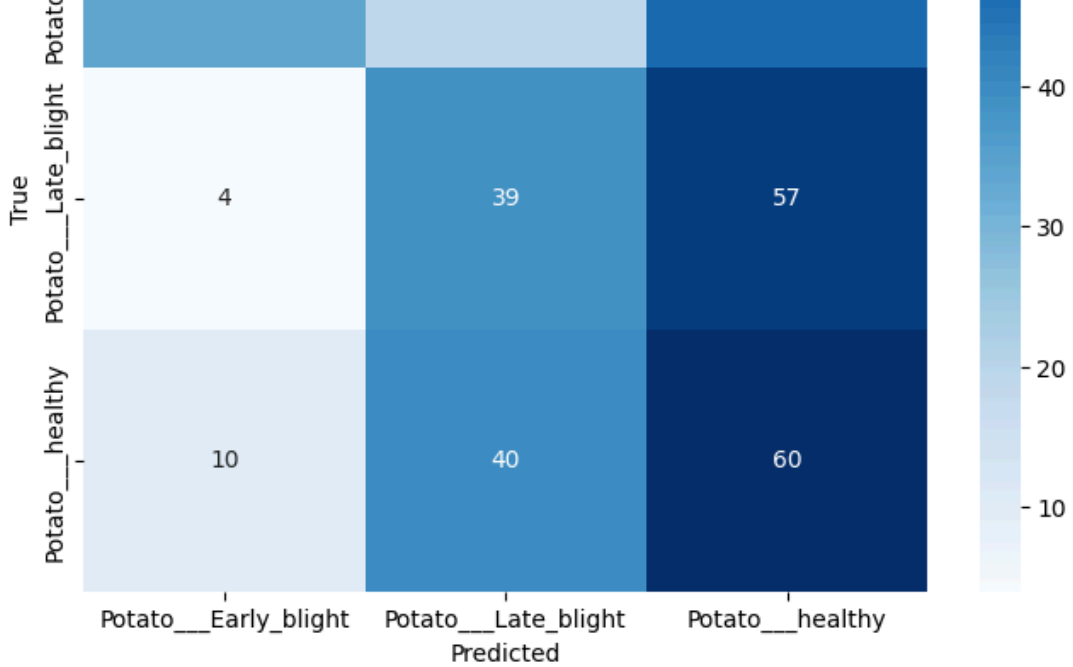
```
    test_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred_prob = ori_model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred_prob, axis=1)

# Build confusion matrix for the test dataset
conf_matrix_test = confusion_matrix(y_test_true, y_test_pred)

# Plot confusion matrix for the test dataset using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_test, annot=True, fmt='d', cmap='Blues', xticklabels=actual_class_names, yticklabels=actual_class_names)
plt.title('Confusion Matrix (Test Dataset)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Generate classification report for the test dataset
class_report_test = classification_report(y_test_true, y_test_pred, target_names=actual_class_names)
print("Classification Report (Test Dataset):")
print(class_report_test)

# Calculate accuracy for the test dataset
accuracy_test = np.mean(y_test_true == y_test_pred)
print("Accuracy (Test Dataset):", accuracy_test)
```

```
14/14 [==============================] - 304s 22s/step
Found 2236 images belonging to 3 classes.
```




```
Classification Report (Validation Data):
                      precision    recall  f1-score   support

Potato___Early_blight       0.47      0.47      0.47       200
 Potato___Late_blight       0.48      0.50      0.49       200
     Potato___healthy       0.11      0.09      0.10        47

             accuracy                           0.44       447
            macro avg       0.35      0.35      0.35       447
         weighted avg       0.43      0.44      0.44       447

Found 310 images belonging to 3 classes.
10/10 [==============================] - 193s 19s/step
```





Classification Report (Test Dataset):

	precision	recall	f1-score	support
Potato___Early_blight	0.71	0.34	0.46	100
Potato___Late_blight	0.40	0.39	0.39	100
Potato___healthy	0.37	0.55	0.44	110
accuracy			0.43	310
macro avg	0.49	0.43	0.43	310
weighted avg	0.49	0.43	0.43	310

Accuracy (Test Dataset): 0.4290322580645161

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

# Use your trained model (ori_model) to predict labels for the test dataset
predictions = ori_model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]

# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names from the test generator
class_names = list(test_generator.class_indices.keys())

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()
```

```
10/10 [==============================] – 204s 20s/step
```

True: Early_Blight
Predicted: Late_Blight

True: Early_Blight
Predicted: Late_Blight

True: Early_Blight
Predicted: Healthy

True: Early_Blight
Predicted: Late_Blight

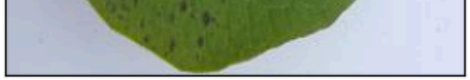
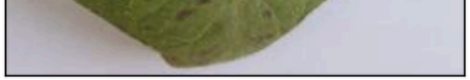
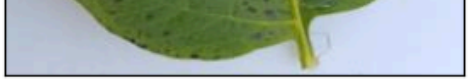
True: Early_Blight
Predicted: Late_Blight

True: Early_Blight
Predicted: Late_Blight

True: Early_Blight
Predicted: Late_Blight

True: Early_Blight
Predicted: Late_Blight

True: Early_Blight
Predicted: Late_Blight

Compressed

```
from tensorflow.keras.models import load_model
```

```
from google.colab import drive
import os
drive.mount('/content/drive')
os.chdir('/content/drive/My Drive/243 Analytics Lab/compressed/')
```

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

```
compressed_model=load_model('my_vgg_model.h5')
```

```
# Define the image size
image_size = 224  # Placeholder value, replace with the appropriate size

# Load additional test dataset (replace 'test_data_dir' with the appropriate directory)
test_data_dir = '/content/drive/My Drive/IEOR 243 Analytics Lab/module2/plus测试图片数据/compressed'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(test_data_dir,
                                                                        target_size=(image_size, image_size),
                                                                        batch_size=batch_size,
                                                                        class_mode='categorical',
                                                                        shuffle=False)
```

```
# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred = compressed_model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred, axis=1)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
Found 310 images belonging to 3 classes.
10/10 [==============================] - 201s 20s/step
Confusion Matrix:
[[50  1 49]
 [ 3 35 62]
 [38  1 71]]
```

```
from sklearn.metrics import accuracy_score

# Calculate accuracy
accuracy = accuracy_score(y_test_true, y_test_pred)

print("Accuracy:", accuracy)
```

```
Accuracy: 0.5032258064516129
```

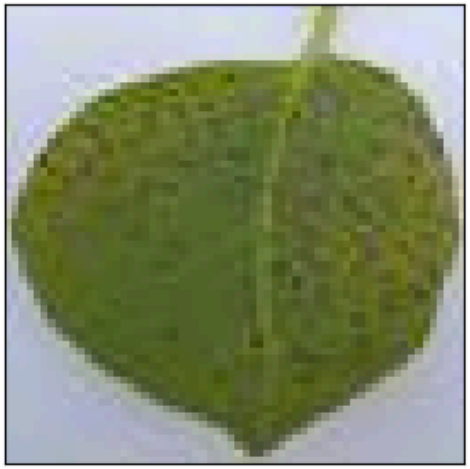
```
import numpy as np

# Use your trained model to predict labels for the test dataset
predictions = compressed_model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]
```

```
10/10 [==============================] - 200s 20s/step
```

```
# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names
class_names = ['Healthy', 'Early blight', 'Late blight']

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()
```

True: Healthy
Predicted: Late blight

True: Healthy
Predicted: Late blight



True: Healthy
Predicted: Late blight

True: Healthy
Predicted: Late blight

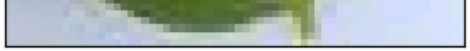
True: Healthy
Predicted: Late blight

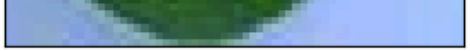
True: Healthy
Predicted: Late blight

True: Healthy
Predicted: Late blight

True: Healthy
Predicted: Late blight

True: Healthy
Predicted: Late blight





Sharpen

```
from tensorflow.keras.models import load_model


from google.colab import drive
import os
drive.mount('/content/drive')
os.chdir('/content/drive/My Drive/243 Analytics Lab/sharpen/')
```

```
Mounted at /content/drive
```

```
sharpen_model=load_model('my_vgg_model.h5')


# Define the image size
image_size = 224  # Placeholder value, replace with the appropriate size

# Load additional test dataset (replace 'test_data_dir' with the appropriate directory)
test_data_dir = '/content/drive/My Drive/IEOR 243 Analytics Lab/module2/plus测试图片数据/sharpen'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(test_data_dir,
                                                                        target_size=(image_size, image_size),
                                                                        batch_size=batch_size,
                                                                        class_mode='categorical',
                                                                        shuffle=False)


# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred = sharpen_model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred, axis=1)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
Found 310 images belonging to 3 classes.
10/10 [==============================] - 118s 12s/step
Confusion Matrix:
[[60  4 36]
 [ 5 34 61]
 [24 24 62]]
```

```
from sklearn.metrics import accuracy_score

# Calculate accuracy
accuracy = accuracy_score(y_test_true, y_test_pred)

print("Accuracy:", accuracy)
```

```
Accuracy: 0.5032258064516129
```

```
import numpy as np

# Use your trained model to predict labels for the test dataset
predictions = sharpen_model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]
```

```
10/10 [==============================] - 174s 17s/step
```

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

# Use your trained model to predict labels for the test dataset
predictions = sharpen_model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]

# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names
class_names = ['early_blight', 'healthy', 'late_blight']

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()
```

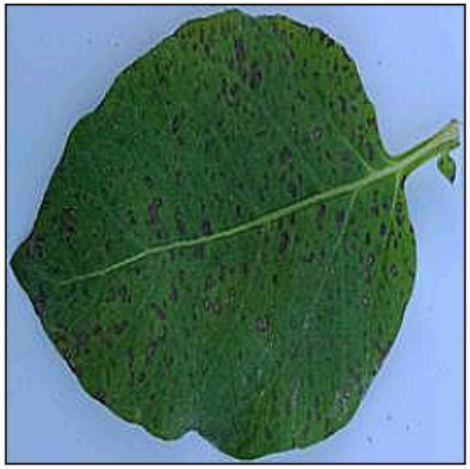
```
10/10 [==============================] – 120s 12s/step
```

True: early_blight
Predicted: late_blight

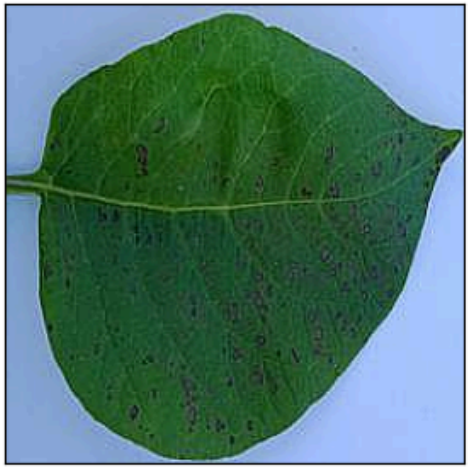
True: early_blight
Predicted: late_blight

True: early_blight
Predicted: late_blight

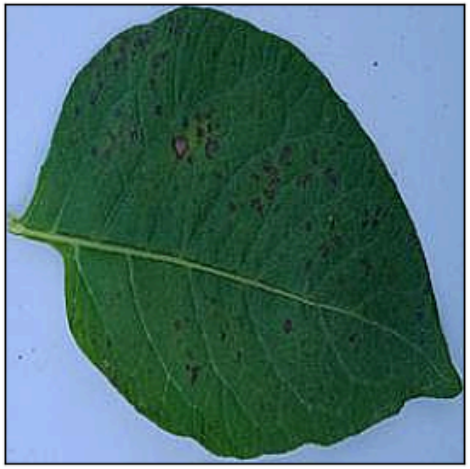
True: early_blight
Predicted: late_blight



True: early_blight
Predicted: late_blight



True: early_blight
Predicted: late_blight

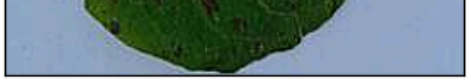


True: early_blight
Predicted: late_blight

True: early_blight
Predicted: late_blight

True: early_blight
Predicted: late_blight



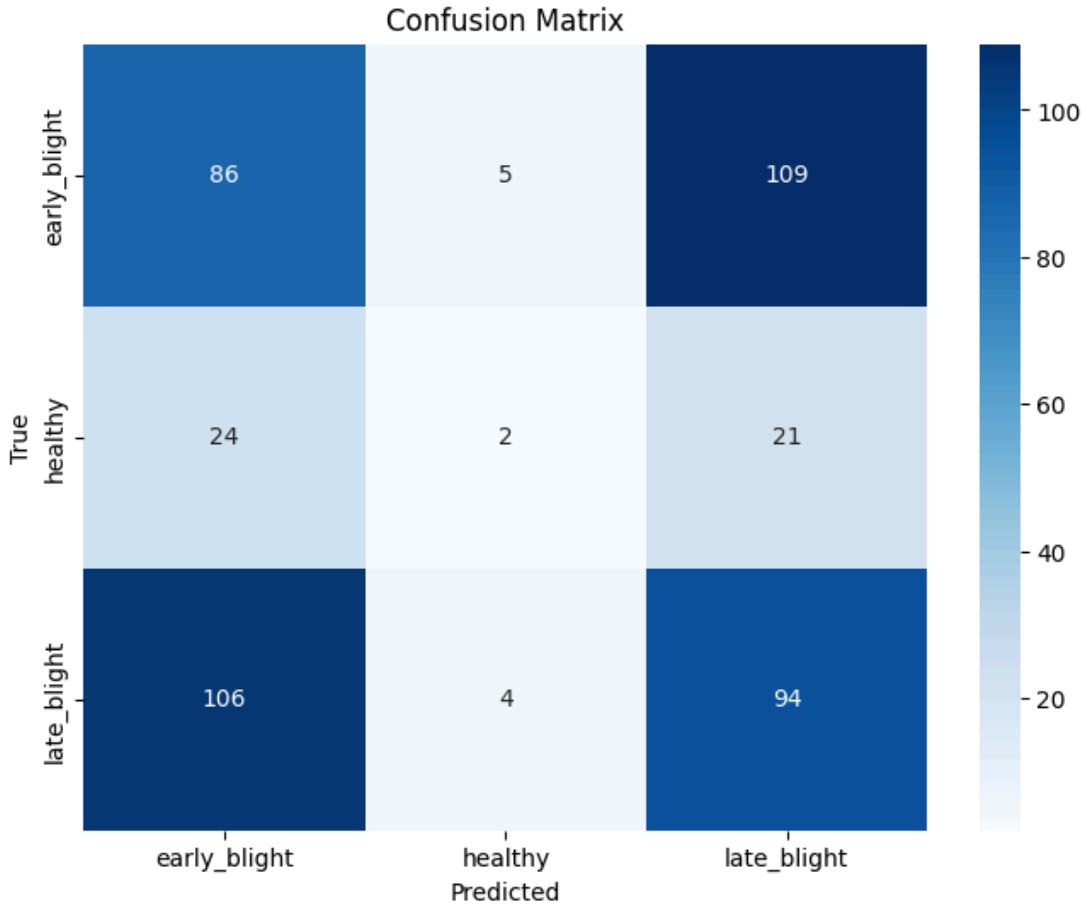


Sharpen+Compressed

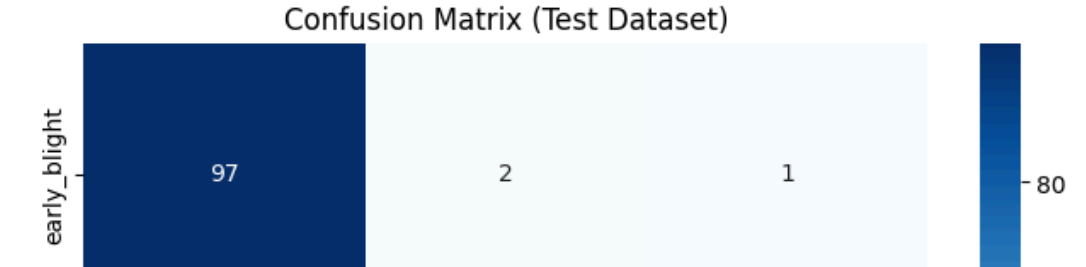
```
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the trained model
model = load_model('/content/drive/My Drive/CS/your_model.h5')

# Predicted probabilities
y_pred_prob = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

# Plot confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Generate classification report
class_report = classification_report(y_true, y_pred, target_names=class_labels)
print("Classification Report:")
print(class_report)

# Load additional test dataset
test_data_dir = '/content/drive/My Drive/CS_new'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    test_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred_prob = model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred_prob, axis=1)

# Build confusion matrix for the test dataset
conf_matrix_test = confusion_matrix(y_test_true, y_test_pred)
```

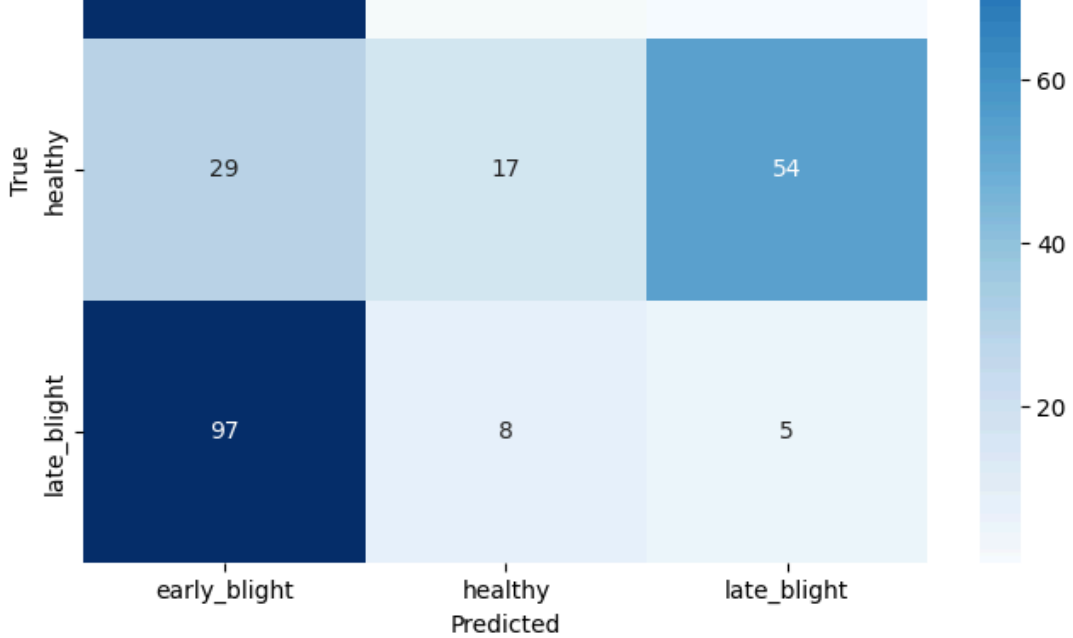
```
# Plot confusion matrix for the test dataset using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_test, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.title('Confusion Matrix (Test Dataset)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Generate classification report for the test dataset
class_report_test = classification_report(y_test_true, y_test_pred, target_names=class_labels)
print("Classification Report (Test Dataset):")
print(class_report_test)

# Calculate accuracy for the test dataset
accuracy_test = np.mean(y_test_true == y_test_pred)
print("Accuracy (Test Dataset):", accuracy_test)
```

```
15/15 [==============================] - 313s 21s/step
```




```
Classification Report:
              precision    recall  f1-score   support

early_blight       0.40      0.43      0.41       200
     healthy       0.18      0.04      0.07        47
 late_blight       0.42      0.46      0.44       204

    accuracy                           0.40       451
   macro avg       0.33      0.31      0.31       451
weighted avg       0.39      0.40      0.39       451

Found 310 images belonging to 3 classes.
10/10 [==============================] - 196s 20s/step
```

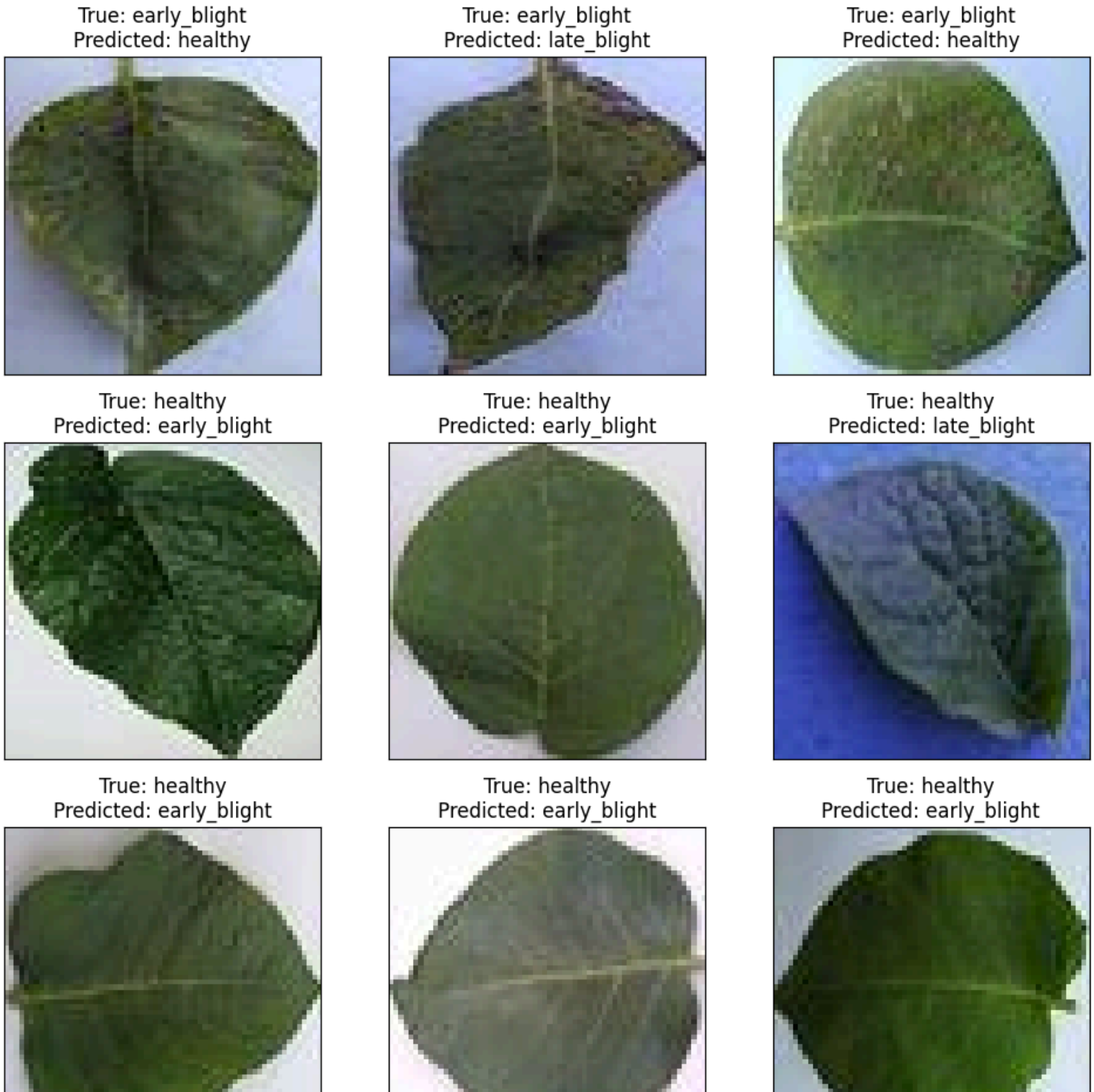


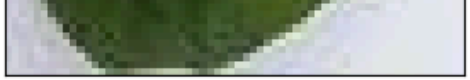


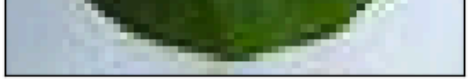

```
Classification Report (Test Dataset):
              precision    recall  f1-score   support

early_blight       0.43      0.97      0.60       100
     healthy       0.63      0.17      0.27       100
 late_blight       0.08      0.05      0.06       110

    accuracy                           0.38       310
   macro avg       0.38      0.40      0.31       310
weighted avg       0.37      0.38      0.30       310

Accuracy (Test Dataset): 0.38387096774193546
```

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

# Use your trained model to predict labels for the test dataset
predictions = model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]

# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names
class_names = ['early_blight', 'healthy', 'late_blight']

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()g
```

```
10/10 [==============================] – 204s 20s/step
```





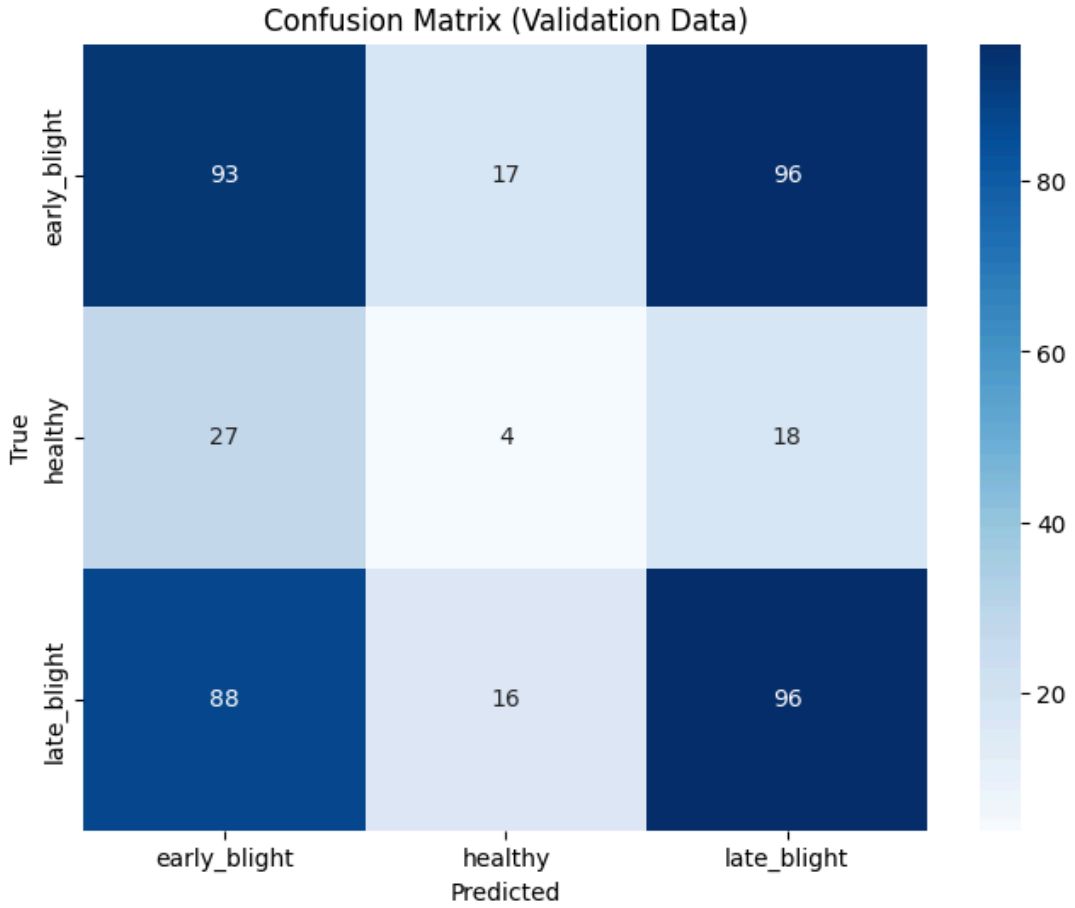


Flipped

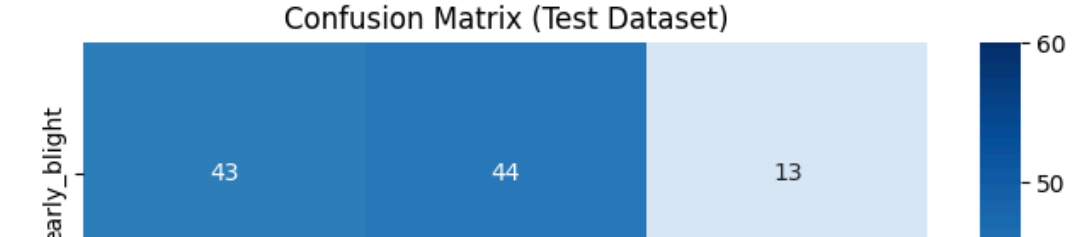
```
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the trained model
fli_model = load_model('/content/drive/My Drive/Flipped/fli_model.h5')

# Predicted probabilities for validation data
y_pred_prob = fli_model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_prob, axis=1)

# True labels
y_true = validation_generator.classes

# Build confusion matrix for validation data
conf_matrix = confusion_matrix(y_true, y_pred)

# Load the training data directory to get actual class names
train_data_dir = '/content/drive/My Drive/Flipped'  # Adjust the path to your training data
train_generator = ImageDataGenerator().flow_from_directory(
    train_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Get the actual class names
actual_class_names = list(train_generator.class_indices.keys())

# Plot confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=actual_class_names, yticklabels=actual_class_names)
plt.title('Confusion Matrix (Validation Data)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Generate classification report for validation data
class_report = classification_report(y_true, y_pred, target_names=actual_class_names)
print("Classification Report (Validation Data):")
print(class_report)

# Load additional test dataset
test_data_dir = '/content/drive/My Drive/Flipped_new'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
```

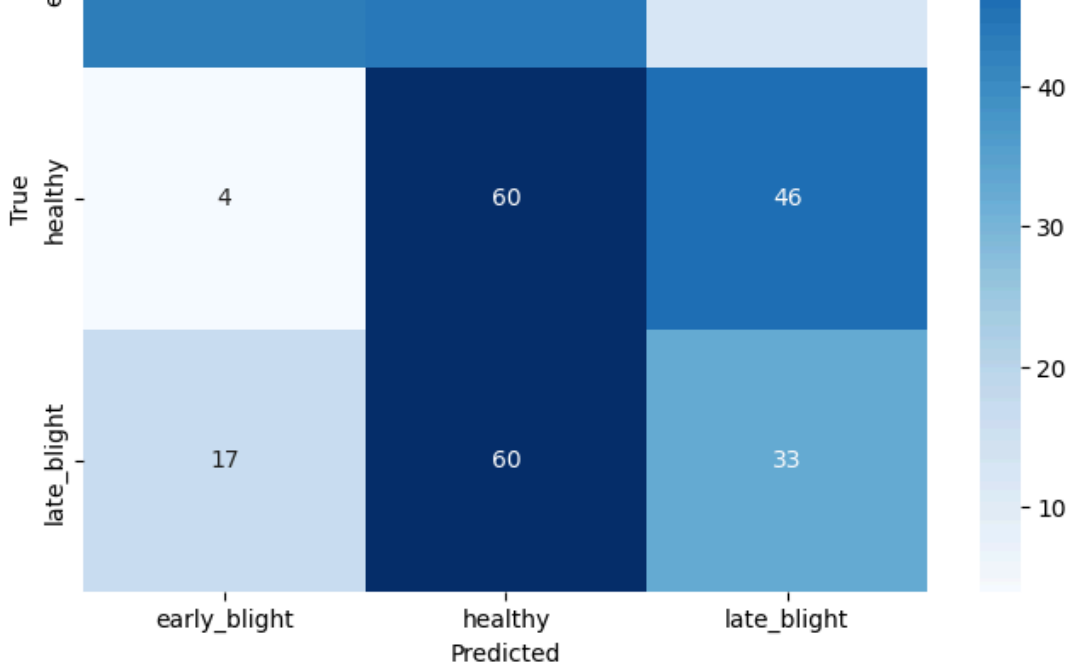
```
    test_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred_prob = fli_model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred_prob, axis=1)

# Build confusion matrix for the test dataset
conf_matrix_test = confusion_matrix(y_test_true, y_test_pred)

# Plot confusion matrix for the test dataset using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_test, annot=True, fmt='d', cmap='Blues', xticklabels=actual_class_names, yticklabels=actual_class_names)
plt.title('Confusion Matrix (Test Dataset)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Generate classification report for the test dataset
class_report_test = classification_report(y_test_true, y_test_pred, target_names=actual_class_names)
print("Classification Report (Test Dataset):")
print(class_report_test)

# Calculate accuracy for the test dataset
accuracy_test = np.mean(y_test_true == y_test_pred)
print("Accuracy (Test Dataset):", accuracy_test)
```

```
15/15 [==============================] - 286s 19s/step
Found 2276 images belonging to 3 classes.
```




```
Classification Report (Validation Data):
              precision    recall  f1-score   support

early_blight       0.45      0.45      0.45       206
     healthy       0.11      0.08      0.09        49
 late_blight       0.46      0.48      0.47       200

    accuracy                           0.42       455
   macro avg       0.34      0.34      0.34       455
weighted avg       0.42      0.42      0.42       455

Found 320 images belonging to 3 classes.
10/10 [==============================] - 199s 20s/step
```





Classification Report (Test Dataset):

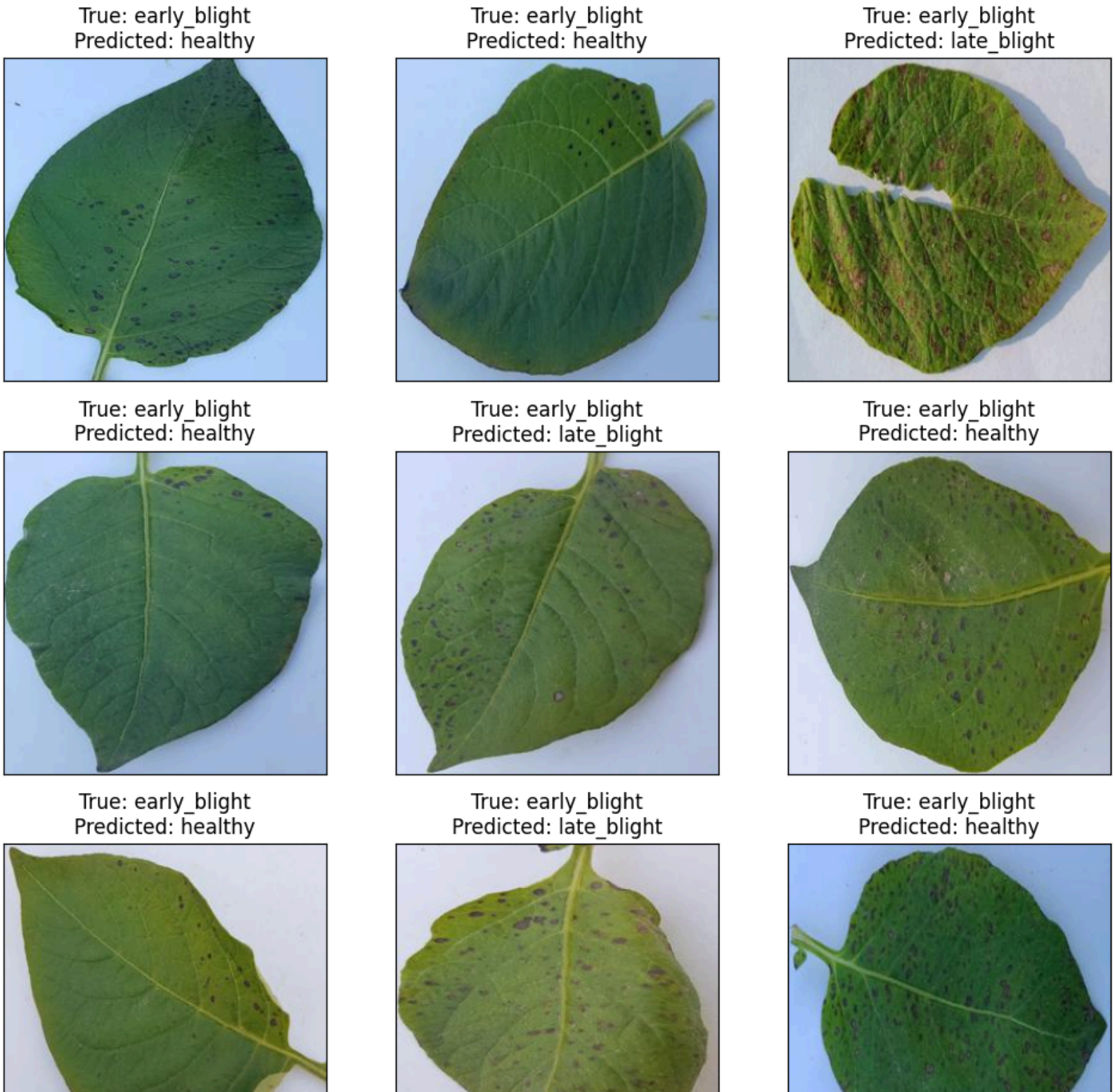
	precision	recall	f1-score	support
early_blight	0.67	0.43	0.52	100
healthy	0.37	0.55	0.44	110
late_blight	0.36	0.30	0.33	110
accuracy			0.42	320
macro avg	0.47	0.43	0.43	320
weighted avg	0.46	0.42	0.43	320

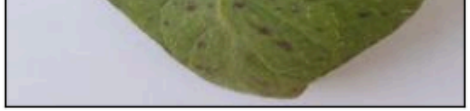
Accuracy (Test Dataset): 0.425

```
import numpy as np
import os
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define your test data directory
test_data_dir = '/content/drive/My Drive/Flipped_new'

# Image size and batch size
image_size = (224, 224)
batch_size = 32

# Create an ImageDataGenerator for the test set
test_datagen = ImageDataGenerator(rescale=1./255)

# Create a test data generator
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False  # Ensure that the order of predictions matches the order of the files
)

# Load your trained model (fli_model) - Adjust the file path accordingly
fli_model = load_model('/content/drive/My Drive/Flipped/fli_model.h5')

# Use the loaded model to predict labels for the test dataset
predictions = fli_model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]

# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names from the test generator
class_names = list(test_generator.class_indices.keys())
```

```
# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()
```

```
Found 320 images belonging to 3 classes.
10/10 [==============================] – 169s 17s/step
```




Flipped+Compressed

```
from tensorflow.keras.models import load_model


from google.colab import drive
import os
drive.mount('/content/drive')
os.chdir('/content/drive/My Drive/243 Analytics Lab/flippedandcompressed/')
```

```
Mounted at /content/drive
```

```
flippedandcompressed_model=load_model('my_vgg_model.h5')


# Define the image size
image_size = 224  # Placeholder value, replace with the appropriate size

# Load additional test dataset (replace 'test_data_dir' with the appropriate directory)
test_data_dir = '/content/drive/My Drive/IEOR 243 Analytics Lab/module2/plus测试图片数据/Flipped+Compressed'
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(test_data_dir,
                                                                        target_size=(image_size, image_size),
                                                                        batch_size=batch_size,
                                                                        class_mode='categorical',
                                                                        shuffle=False)

# Make predictions on the test dataset
y_test_true = test_generator.classes
y_test_pred = flippedandcompressed_model.predict(test_generator)
y_test_pred = np.argmax(y_test_pred, axis=1)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
Found 320 images belonging to 3 classes.
10/10 [==============================] - 182s 18s/step
Confusion Matrix:
[[23 22 55]
 [ 0 65 35]
 [21 24 75]]
```

```
from sklearn.metrics import accuracy_score

# Calculate accuracy
accuracy = accuracy_score(y_test_true, y_test_pred)

print("Accuracy:", accuracy)
```

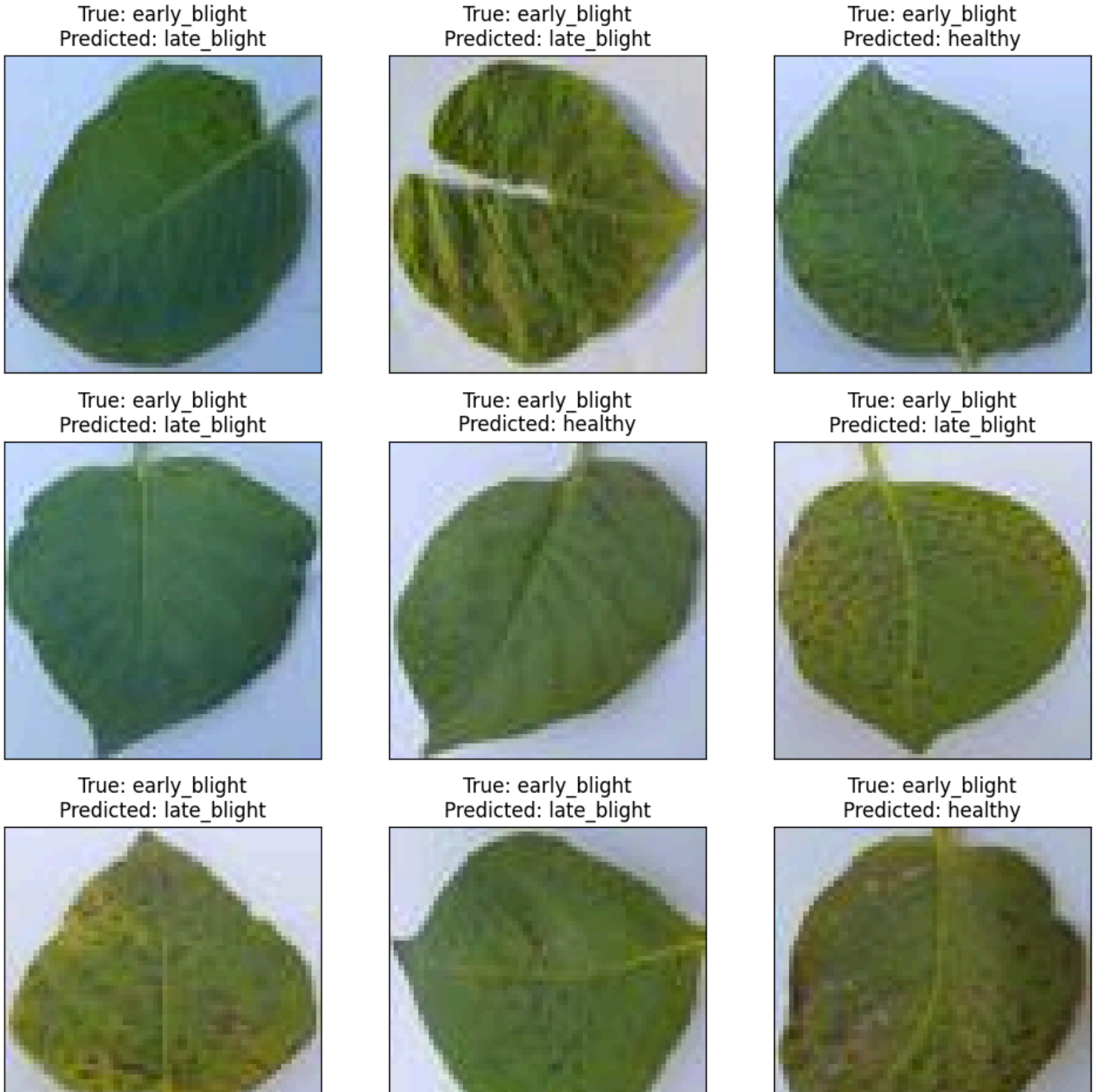
```
    Accuracy: 0.509375
```

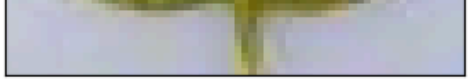
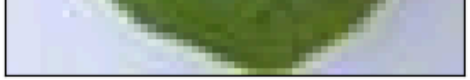
```
import numpy as np

# Use your trained model to predict labels for the test dataset
predictions = flippedandcompressed_model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]
```

```
    10/10 [==============================] - 182s 18s/step
```

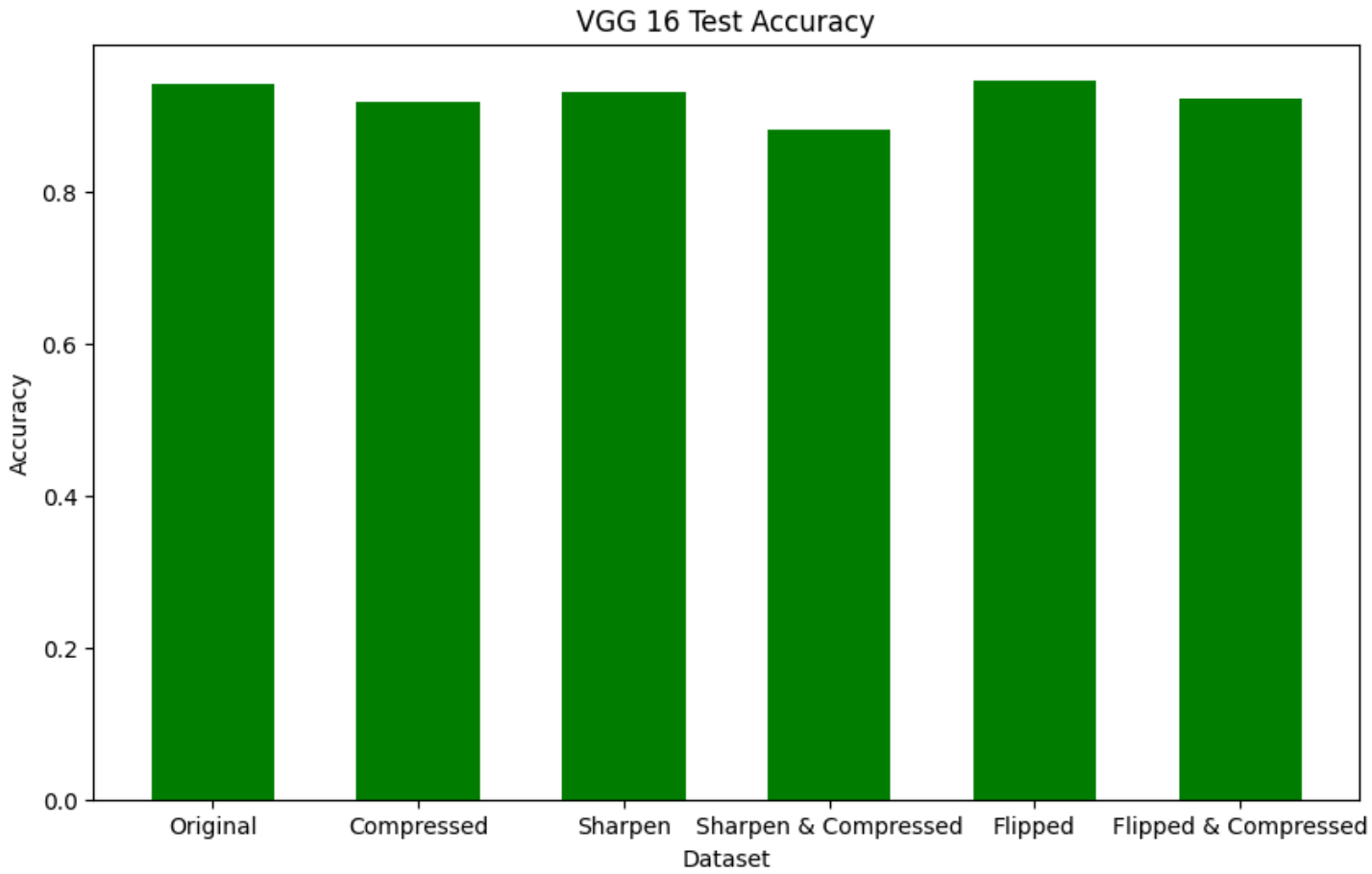
```
import numpy as np
import os
import matplotlib.pyplot as plt

# Use your trained model to predict labels for the test dataset
predictions = flippedandcompressed_model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)

# Get the true labels from the test generator
true_labels = test_generator.classes

# Find indices of misclassified images
misclassified_indices = np.where(predicted_labels != true_labels)[0]

# Get the filenames of misclassified images from the test generator
misclassified_filenames = [test_generator.filenames[i] for i in misclassified_indices]

# Optionally, load the misclassified images
misclassified_images = [plt.imread(os.path.join(test_data_dir, filename)) for filename in misclassified_filenames]

# Get the true and predicted labels for misclassified images
true_labels_misclassified = true_labels[misclassified_indices]
predicted_labels_misclassified = predicted_labels[misclassified_indices]

# Define the class names
class_names = ['early_blight', 'healthy', 'late_blight']

# Create a figure with nine subplots arranged in a 3x3 grid
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

# Display each misclassified image on a subplot with appropriate title
for i in range(3):
    for j in range(3):
        index = i * 3 + j
        if index < len(misclassified_images):
            true_label = class_names[true_labels_misclassified[index]]
            predicted_label = class_names[predicted_labels_misclassified[index]]
            axs[i, j].imshow(misclassified_images[index])
            axs[i, j].set_title(f'True: {true_label}\nPredicted: {predicted_label}')

# Remove the x and y ticks from each subplot
for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])

plt.tight_layout()
plt.show()
```

```
10/10 [==============================] – 182s 18s/step
```




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

x_1 = 0.9419
x_2 = 0.9201
x_3 = 0.9312
x_4 = 0.8820
x_5 = 0.9478
x_6 = 0.9240


# creating the dataset
data = {'Original':x_1, 'Compressed':x_2, 'Sharpen':x_3, 'Sharpen & Compressed':x_4, 'Flipped':x_5,
        'Flipped & Compressed':x_6}

data_types = list(data.keys())
acc = list(data.values())

fig = plt.figure(figsize = (10, 6))

# creating the bar plot
plt.bar(data_types, acc, color ='g',
        width = 0.6)

plt.xlabel("Dataset")
plt.ylabel("Accuracy")
plt.title("VGG 16 Test Accuracy")
plt.show()
```


VGG 16 Test Accuracy
Accuracy
0.0
0.2
0.4
0.6
0.8
Original
Compressed
Sharpen
Sharpen & Compressed
Flipped
Flipped & Compressed
Dataset

..

..

..

..

..

..

Limitations and Challenges

1. Limitation: Lack of Real-World Data Variation

The significant drop in accuracy from nearly 90% on the training set to below 45% on external validation highlights a critical limitation in the models' ability to generalize to new data. This discrepancy is likely **due to the initial training dataset not capturing the full spectrum of variability found in real-world scenarios**, such as different potato varieties, lighting conditions, and background variations.

Future Directions:

- **Cross-Domain Data Augmentation**: Incorporate data augmentation techniques that **simulate real-world variations** in lighting, background, and potato leaf conditions. This approach can make the model more robust to the types of variations it will encounter in practical applications.
- **Geographical Diversity in Data Collection**: Expand the dataset to include potato leaf images from **various geographical locations and climates** to ensure the model learns from a representative sample of global potato diversity.

2. Limitation: Model Complexity and Overfitting

The current models, especially the more complex ones like Inception V3 and VGG16, may **be prone to overfitting**. This is indicated by their high accuracy on the training set but poor performance on the external validation set. Overfitting occurs **when a model learns the noise and specific patterns in the training data** to the extent that it negatively impacts its ability to generalize.

Future Directions:

- **Advanced Regularization Techniques**: Beyond dropout and weight regularization, explore advanced regularization methods such as batch normalization and early stopping to prevent overfitting.
- **Data Distillation**: Utilize techniques like data distillation to identify and prioritize the most informative training examples, potentially reducing overfitting by focusing on data that contributes most to generalizable learning.

3. Limitation: Insufficient Model Interpretability and Diagnostic Insight

While achieving high accuracy is crucial, understanding the models' decision-making processes is equally important, especially in agricultural applications where specific interventions are based on the type of disease identified. The current models, particularly deep learning ones like CNN, Inception V3, and VGG16, often act as "black boxes," offering little insight into which features they consider important for classification. This lack of interpretability can hinder trust and adoption by end-users, such as farmers and agricultural experts, who may require understanding why a particular diagnosis was made to take appropriate action.

Future Directions:

- **Feature Visualization and Analysis:** Implement techniques to visualize and analyze the features that the models are focusing on, such as Grad-CAM (Gradient-weighted Class Activation Mapping) for deep learning models. This can help identify whether the model is focusing on relevant features for disease identification or being misled by irrelevant patterns.
- **Interactive Diagnostic Tools:** Develop interactive tools that allow users to see the model's reasoning for its predictions, possibly integrating feedback mechanisms where experts can validate or correct the model's decisions. This not only improves interpretability but also continually enhances the

model's performance and reliability through expert feedback.

Application, Relevance, and Ethical Considerations

Real-World Relevance

The project directly addresses a significant challenge in the agricultural industry by aiming to improve the accuracy and efficiency of potato leaf disease identification. Potato crops are a staple food source globally, and diseases affecting these crops can have profound implications on food security and agricultural productivity. By employing advanced machine learning and deep learning models like KNN, SVM, CNN, Inception V3, and VGG16, *the project taps into the potential of technology to mitigate losses due to diseases, thus supporting farmers in enhancing crop yield and quality.* **This application is not only relevant to the agricultural sector but also holds societal importance by contributing to the sustainability of food resources.**

Impact and Applicability

The potential impact of this project is substantial, offering a scalable and efficient tool for early disease detection, which is crucial for timely intervention and treatment. The use of diverse models ensures a comprehensive approach to identifying various diseases, potentially increasing the applicability of the project across different regions and potato varieties. Furthermore, by integrating these models into mobile applications or cloud-based platforms, the project can provide accessible and user-friendly tools for farmers and agricultural workers worldwide, empowering them with actionable insights to protect their crops.

Ethical Considerations

While the project has promising benefits, it's essential to address the ethical implications and societal impacts thoroughly.

- One primary concern is **data privacy and security**, especially when collecting and processing images from farms that may include proprietary or sensitive information. Ensuring that data is collected, stored, and used ethically, with consent from participants, is crucial.
- Another ethical consideration is the **accessibility and inclusivity of the technology**. The benefits of such advancements should be equitably distributed, ensuring that small-scale and under-resourced farmers also have access to these tools, not just large agricultural operations. *This inclusivity can help prevent widening the technological gap between different farming communities.*
- Lastly, there's the **potential risk of over-reliance on technology**, which could lead to diminished traditional agricultural knowledge and practices. *It's important to integrate these technological tools in a way that complements and enhances, rather than replaces, the expertise of farmers and agricultural experts.*

In conclusion, the project's focus on improving potato leaf disease identification through machine learning and deep learning models is highly relevant and has the potential to make a significant impact in the agricultural sector. However, it is crucial to navigate the ethical considerations carefully to ensure that the technology is developed and deployed responsibly, with a focus on maximizing societal benefits and minimizing potential harms.