Capstone Project Data Preparation/Feature Engineering

Project Title: Plant Disease Detection Using Deep Learning

Team Members

- Dawit Tadesse Hailu
- Chol Moses Malueth
- Fahadi Mugigayi

1. Overview

Data preparation and feature engineering phase are crucial for our machine learning project as it ensures that the data is in the best possible shape for training a model. This phase involved collecting, cleaning, and transforming data to create meaningful features that the model can learn from. Proper data preparation enhances the model's accuracy, robustness, and generalizability.

2. Data Collection

Source: The primary dataset for this project is the PlantVillage dataset, which contains 55,448 images of healthy and diseased plant leaves across 39 crop categories. This dataset is known for its diversity and quality, making it ideal for training robust deep-learning models.

Preprocessing Steps:

- **Data Augmentation**: To simulate various environmental conditions and increase dataset variability, techniques such as scaling, shifting, rotation and flipping were applied.
- **Image Standardization**: All images were resized to ensure uniformity.
- **Image Scaling:** To make sure it's easy for the learning process to go smoothly, the values of the RGB pixel values were scaled.

3. Data Cleaning

Folder Restructuring for our convenience: The folders are renamed after being downloaded to make them easy to work with.

```
import os

# Define the paths
old_folder_path = './training_set'
new_folder_path = './Dataset'
old_subfolder_path = os.path.join(new_folder_path,
'Plant_leave_diseases_dataset_without_augmentation')
new_subfolder_path = os.path.join(new_folder_path, 'all_data')

# Rename the main folder
if os.path.exists(old_folder_path):
    os.rename(old_folder_path, new_folder_path)
    print(f'Renamed folder "{old_folder_path}" to "{new_folder_path}".')

# Rename the subfolder
if os.path.exists(old_subfolder_path):
    os.rename(old_subfolder_path, new_subfolder_path)
    print(f'Renamed subfolder "{old_subfolder_path}" to "{new_subfolder_path}".')
```

Steps Taken:

- **Handling Missing Values**: Checked for any missing values or corrupted images in the dataset. Any incomplete data entries were removed or replaced.
- **Outliers**: Inspected the dataset for outliers, such as images with incorrect labels or abnormal features, and cleaned these entries to prevent skewed results.

```
import os
import cv2
import numpy as np
```

```
# Handling Missing Values
def remove_incomplete_data(image_dir):
    for root, dirs, files in os.walk(image_dir):
        for file in files:
            try:
                img = cv2.imread(os.path.join(root, file))
                if img is None:
                    os.remove(os.path.join(root, file))
            except Exception as e:
                print(f"Error reading {file}: {e}")
                os.remove(os.path.join(root, file))

remove_incomplete_data('./Dataset/all_data')
```

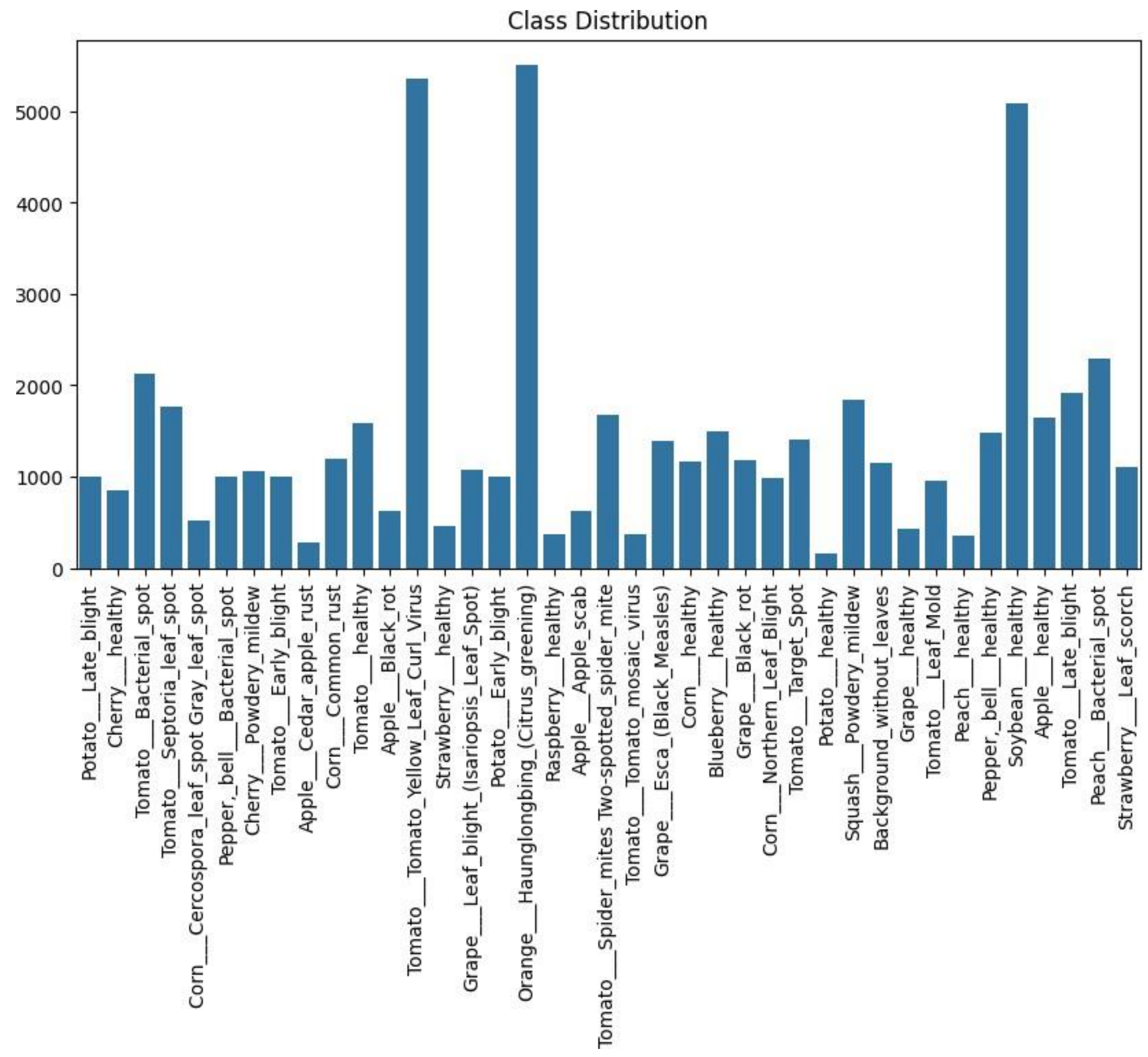
```
# Outliers
def remove_outliers(image_dir):
    for root, dirs, files in os.walk(image_dir):
        for file in files:
            try:
                img = cv2.imread(os.path.join(root, file))
                if img is None or img.shape[0] < 100 or img.shape[1]
< 100:
                    os.remove(os.path.join(root, file))
            except Exception as e:
                print(f"Error processing {file}: {e}")
                os.remove(os.path.join(root, file))

remove_outliers('./Dataset/all_data')
```

---

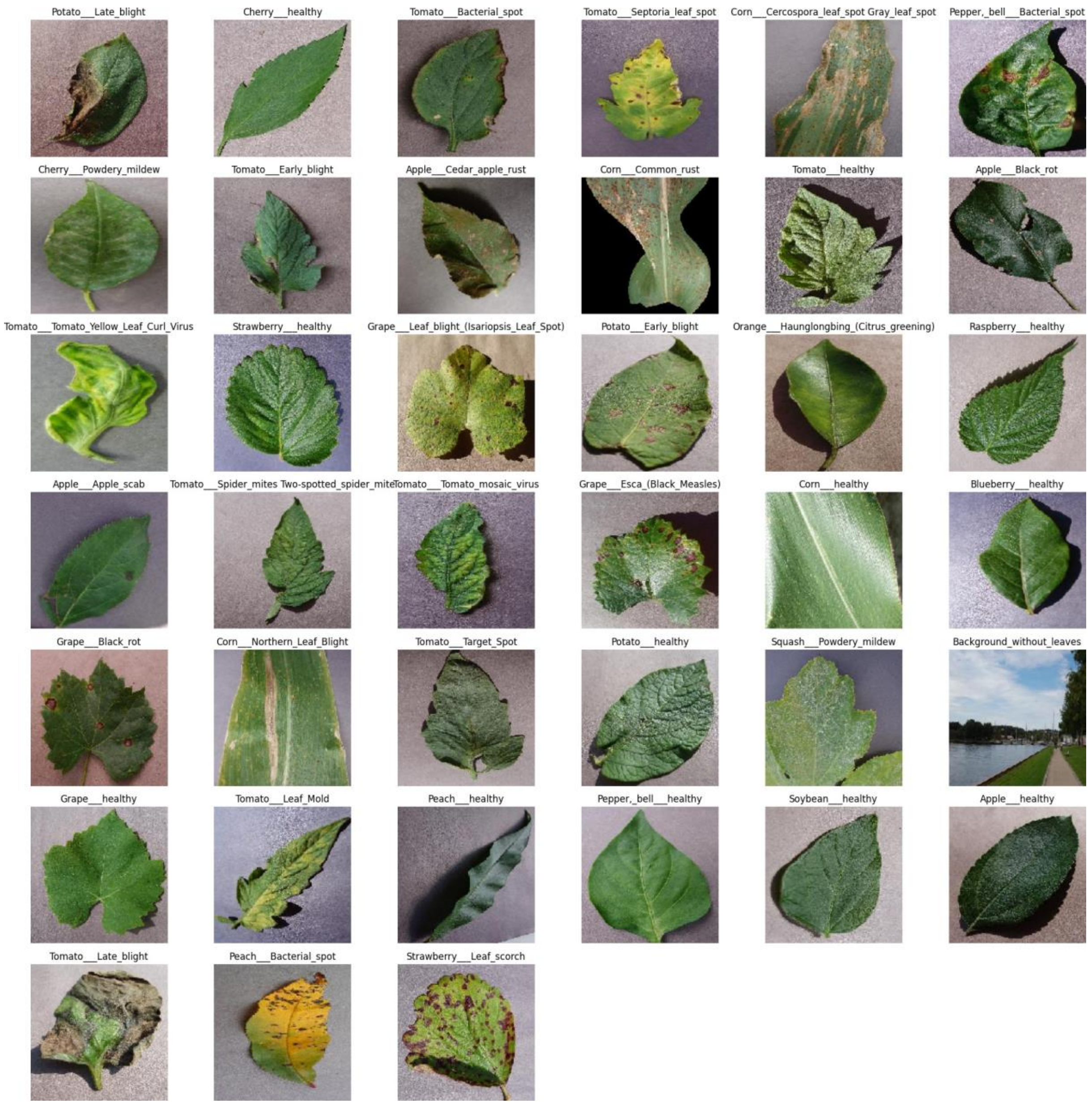
4. Exploratory Data Analysis (EDA)

Visualizations and Insights:

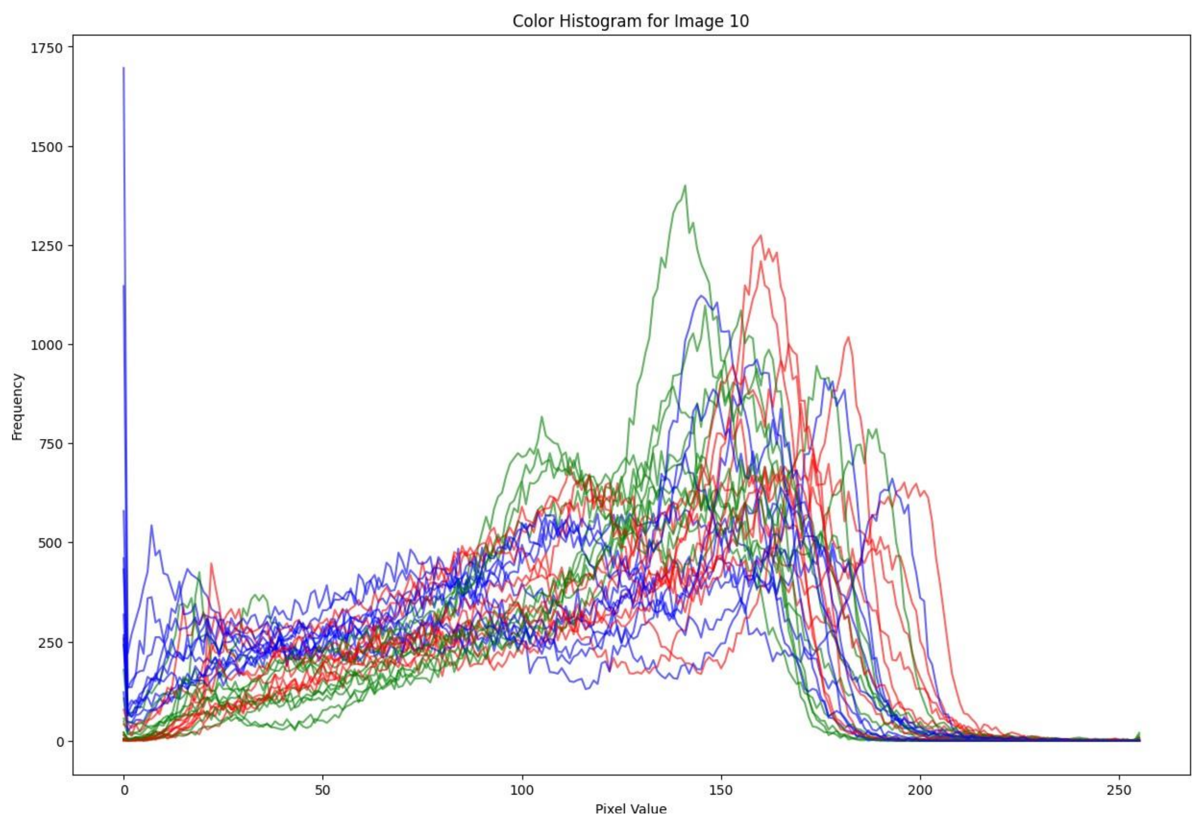
- **Class Distribution**: Visualized the number of images per plant class (healthy vs. diseased) to ensure balanced data.



- **Image Samples**: Displayed sample images from different classes to understand the variations in leaf appearances.



- **Color Histograms**: Analyzed the color distributions within the images to detect any patterns or anomalies and reveal any imbalances in the color channels.



```
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from tensorflow.keras.preprocessing.image import load_img,
img_to_array
```

```
# Class Distribution
def visualize_class_distribution(image_dir):
    class_counts = Counter()
    for root, dirs, files in os.walk(image_dir):
        for dir_name in dirs:
            class_counts[dir_name] =
len(os.listdir(os.path.join(root, dir_name)))

    plt.figure(figsize=(10, 5))
    sns.barplot(x=list(class_counts.keys()),
y=list(class_counts.values()))
    plt.xticks(rotation=90)
    plt.title('Class Distribution')
    plt.show()
```

```
visualize_class_distribution('./Dataset/all_data')
```

```
# Image Samples
def display_sample_images(image_dir, class_names):
    plt.figure(figsize=(20, 20))  # Increase the figure size for
better visibility
    for idx, class_name in enumerate(class_names):
        class_path = os.path.join(image_dir, class_name)
        sample_images = os.listdir(class_path)
        if sample_images:
            img_name = sample_images[0]  # Pick the first image in
the class directory
            img_path = os.path.join(class_path, img_name)
            img = load_img(img_path, target_size=(150, 150))
            plt.subplot(7, 6, idx + 1)  # Adjust the subplot grid
for better layout
            plt.imshow(img)
            plt.axis('off')
            plt.title(class_name)
    plt.tight_layout()
    plt.show()

class_names = os.listdir('./Dataset/all_data')
display_sample_images('./Dataset/all_data', class_names)
```

```
# Color Histograms
def plot_color_histograms(image_dir, num_images=10):
    plt.figure(figsize=(15, 10))
    image_files = [os.path.join(root, file) for root, dirs, files in
os.walk(image_dir) for file in files][:num_images]
    for i, img_path in enumerate(image_files):
        img = img_to_array(load_img(img_path))
        for j, color in enumerate(['r', 'g', 'b']):
            hist, bins = np.histogram(img[:, :, j], bins=256,
range=(0, 256))
            plt.plot(hist, color=color, alpha=0.6)
        plt.title(f'Color Histogram for Image {i+1}')
        plt.xlabel('Pixel Value')
        plt.ylabel('Frequency')
    plt.show()
```

```
plot_color_histograms('./Dataset/all_data')
```

5. Feature Engineering and Data Transformation

Transform Process:

- **Image Augmentation**: Applied transformations like rotation, scaling, and flipping to create more training examples and improve model robustness.

Scaling and Normalization:

- **Scaling**: Resized images to a uniform size of 240x240 pixels.
- **Normalization**: Normalized pixel values to the range [0, 1] to facilitate faster convergence during training.

Rationale: These features were engineered to provide the model with diverse and informative inputs that enhance its ability to distinguish between healthy and diseased leaves under various conditions.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint

TRAINING_DIR = training_data_dir2
# Experiment
train_datagen = ImageDataGenerator(rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(TRAINING_DIR,
                                                    batch_size=100,

class_mode='categorical',

target_size=(240, 240))
```

6. Encoding

- **Label Encoding**: Converted categorical labels (healthy/diseased) into numerical format for model compatibility. This will come in handy after we are done with training and when finally making predictions after deployment to get the actual names.

```
import json

# Extract class indices and corresponding labels
class_indices = train_generator.class_indices
labels = dict((v,k) for k,v in class_indices.items())

# Save labels to a JSON file
with open('labels.json', 'w') as f:
    json.dump(labels, f)
```

Model Exploration

1. Model Selection

Chosen Model: Convolutional Neural Network (CNN)

Rationale: CNNs are particularly well-suited for image classification tasks due to their ability to automatically learn spatial hierarchies of features from images. They have shown high accuracy in similar tasks and can efficiently handle the complexity of plant disease detection.

Strengths:

- Excellent at feature extraction and pattern recognition in images.
- Robust to variations in image quality and background.
- Scalable and adaptable to different datasets.

Weaknesses:

- Requires a large amount of labeled data for training.
- Computationally intensive, requiring significant processing power.

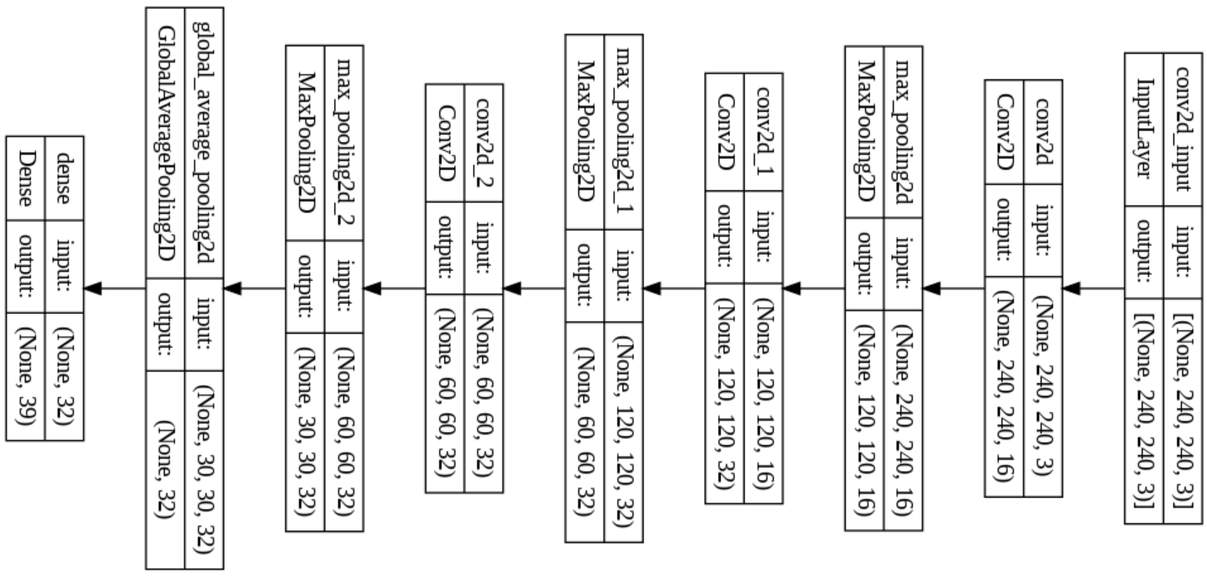
2. Model Training

Training Details:

- **Framework**: TensorFlow Keras
- **Architecture**: Simple CNN with multiple convolutional layers followed by pooling layers and fully connected layers.

- **Hyperparameters**: Learning rate, batch size, number of epochs, dropout rate.
- **Cross-Validation**: Used k-fold cross-validation to ensure the model's robustness and generalization capability.

Rationale: We start with the simplest architecture we can come up with and scale up through experimentation to find the right balance in order to not underfit and also not overfit.


conv2d_input | InputLayer | input: | [(None, 240, 240, 3)] | output: | [(None, 240, 240, 3)]
conv2d | Conv2D | input: | (None, 240, 240, 3) | output: | (None, 240, 240, 16)
max_pooling2d | MaxPooling2D | input: | (None, 240, 240, 16) | output: | (None, 120, 120, 16)
conv2d_1 | Conv2D | input: | (None, 120, 120, 16) | output: | (None, 120, 120, 32)
max_pooling2d_1 | MaxPooling2D | input: | (None, 120, 120, 32) | output: | (None, 60, 60, 32)
conv2d_2 | Conv2D | input: | (None, 60, 60, 32) | output: | (None, 60, 60, 32)
max_pooling2d_2 | MaxPooling2D | input: | (None, 60, 60, 32) | output: | (None, 30, 30, 32)
global_average_pooling2d | GlobalAveragePooling2D | input: | (None, 30, 30, 32) | output: | (None, 32)
dense | Dense | input: | (None, 32) | output: | (None, 39)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense,  Dropout

model = Sequential()

# First Convolutional Layer
model.add(Conv2D(filters=16, kernel_size=3, padding='same',
activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=2))

# Second Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

# Third Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

# Global Average Pooling and Fully Connected Layer
model.add(GlobalAveragePooling2D())
model.add(Dense(39, activation='softmax'))

model.summary()
```

3. Model Evaluation

Evaluation Metrics:

- **Accuracy**: Proportion of correctly classified samples.
- **Precision and Recall**: Measures of the model's ability to correctly identify positive cases and avoid false negatives.
- **F1-Score**: Harmonic mean of precision and recall.
- **Confusion Matrix**: Visual representation of the model's performance across different classes.

```
from sklearn.metrics import confusion_matrix, precision_score,
recall_score, accuracy_score

# Predict the labels
y_pred = model.predict(validation_generator)
y_pred_classes = np.argmax(y_pred, axis=1)

# Get the true labels
y_true = validation_generator.classes

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate precision
precision = precision_score(y_true, y_pred_classes,
average='weighted')
print(f"Precision: {precision}")

# Calculate recall
recall = recall_score(y_true, y_pred_classes, average='weighted')
print(f"Recall: {recall}")

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred_classes)
print(f"Accuracy: {accuracy}")
```

4. Code Implementation

Data Preparation/Feature Engineering first run MAIN Code Snippets:

Downloaded the Dataset inside google colab to access GPU resources as it was quite slow when running on our computers due to the CNN requiring GPU!!!

```
!wget -O training_set.zip
'https://data.mendeley.com/public-files/datasets/tywbtsjrjv/files/d5
652a28-c1d8-4b76-97f3-72fb80f94efc/file_downloaded'

print("Download complete!")
import zipfile

extract_dir = '/content/training_set'
filename = 'training_set.zip'

with zipfile.ZipFile(filename, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

print("Extraction complete!")

import os

os.listdir()
```

```
import os

# Define the paths
old_folder_path = './training_set'
new_folder_path = './Dataset'
old_subfolder_path = os.path.join(new_folder_path,
'Plant_leave_diseases_dataset_without_augmentation')
new_subfolder_path = os.path.join(new_folder_path, 'all_data')

# Rename the main folder
if os.path.exists(old_folder_path):
    os.rename(old_folder_path, new_folder_path)
    print(f'Renamed folder "{old_folder_path}" to "{new_folder_path}".')

# Rename the subfolder
if os.path.exists(old_subfolder_path):
    os.rename(old_subfolder_path, new_subfolder_path)
    print(f'Renamed subfolder "{old_subfolder_path}" to "{new_subfolder_path}".')
```

```
os.listdir("./Dataset")
```

```
import re
from pathlib import Path
import shutil, sys
import random
from shutil import copyfile
import numpy as np

main_dir =  './Dataset'
all_data_dir = './Dataset/all_data'
training_data_dir = './Dataset/traindataset'
training_data_dir2 = './Dataset/train'
testing_data_dir = './Dataset/test'
valid_data_dir = './Dataset/valid'

def split_data(SOURCE, TRAINING, TESTING, SPLIT_SIZE):
    cat_list = os.listdir(SOURCE)
    os.mkdir(TRAINING)
    os.mkdir(TESTING)
    if SPLIT_SIZE < 0.1 or SPLIT_SIZE > 0.9:
        raise Exception("SPLIT SIZE is greater than needed")
    for category in cat_list:
        os.mkdir(TRAINING + '/' + category)
        os.mkdir(TESTING + '/' + category)
        category_image_names = os.listdir(SOURCE + '/' + category)
        training_length = int(len(category_image_names) *
SPLIT_SIZE)
        testing_length = int(len(category_image_names) -
training_length)
        shuffled_set = random.sample(category_image_names,
len(category_image_names))
        training_set = shuffled_set[0:training_length]
        testing_set = shuffled_set[training_length:]
        for filename in training_set:
            this_file = SOURCE + '/' + category + '/' + filename
            destination = TRAINING + '/' + category + '/' + filename
            copyfile(this_file, destination)
        for filename in testing_set:
            this_file = SOURCE + '/' + category + '/' + filename
            destination = TESTING + '/' + category + '/' + filename
            copyfile(this_file, destination)
```

```
split_data(all_data_dir, training_data_dir2, testing_data_dir, 0.7)

def count_files(directory):
    total_files = 0
    for root, dirs, files in os.walk(directory):
        total_files += len(files)
    return total_files

print(f"all files: {count_files(all_data_dir)}")
print(f"training: {count_files(training_data_dir2)}")
print(f"testing: {count_files(testing_data_dir)}")

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint

TRAINING_DIR = training_data_dir2

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

train_generator = train_datagen.flow_from_directory(TRAINING_DIR,
                                                    batch_size=100,
class_mode='categorical',
target_size=(240, 240))

VALIDATION_DIR = testing_data_dir

validation_datagen = ImageDataGenerator(rescale=1./255,
                                        rotation_range=40,
                                        width_shift_range=0.2,
                                        height_shift_range=0.2,
                                        shear_range=0.2,
                                        zoom_range=0.2,
                                        horizontal_flip=True,
                                        fill_mode='nearest')

validation_generator =
validation_datagen.flow_from_directory(VALIDATION_DIR,

batch_size=100,
class_mode='categorical',

target_size=(240, 240))
```

Model Exploration MAIN Code Snippets:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D,
GlobalAveragePooling2D, Dense

model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, padding='same',
activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=128, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=256, kernel_size=2,
kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(GlobalAveragePooling2D())
model.add(Dense(39, activation='softmax'))

model.summary()

from tensorflow.keras.utils import plot_model

plot_model(model, to_file='model_structure.png', show_shapes=True,
show_layer_names=True)
```

```
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
checkpoint = ModelCheckpoint(filepath='model_checkpoint.h5',
                             save_best_only=True,
                             monitor='val_loss',
                             mode='min')
history = model.fit(train_generator,
                    epochs=12,
                    verbose=1,
                    validation_data=validation_generator,
                    callbacks=[checkpoint])

from google.colab import files
files.download('model_checkpoint.h5'
import matplotlib.pyplot as plt

acc =  history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.show()

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

These code snippets illustrate the key steps in the data preparation, feature engineering, model training, and evaluation processes. At each step, we included comments explaining the purpose and functionality of the code. This implementation is our comprehensive approach to building a deep learning model for plant disease detection using the PlantVillage dataset.