Minor Project Report

On

Image Processing for Plant Health Monitoring and Classification

ICT-497

B.TECH IN INFORMATION TECHNOLOGY

2021-25

University School of Information and Communication Technology

Guru Gobind Singh Indraprastha University

Guided By:

Dr. Anuradha Chug
Associate Professor

SUBMITTED BY:

Ujjwal Gupta
00516401521
BTECH IT(7th Sem)

DECLARATION

This is to certify that Project Report entitled "Image Processing for Plant Health Monitoring and Classification "which is submitted by me in partical fulfillment of the requirement for the award of degree B.Tech in Information Technology to USICT, GGSIP University, Dwarks, Delhi comprises only my original work and due acknowledgement has been made in the text to all other material used.

Signature of the Student

Date:

CERTIFICATE

I, Ujjwal Gupta, Enroll No. 00516401521 certify that the Project Report (BTECH-IT) entitled "Image Processing for Plant Health Monitoring and Classification" is done by me and it is an authentic work carried out by me at USICT, IPU Main Campus, Dwarka. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Signature of the Student

Date:

Certified that the Project Report (BTECH-IT) entitled "Image Processing for Plant Health Monitoring and Classification" done by Mr Ujjwal Gupta Roll No. 00516401521, is completed under my guidance.

Signature of the Guide

Date:

Name of the Guide: Dr. Anuradha Chug

Designation: Associate Professor

Address: USICT, IPU Main Campus,

Dwarka Sector-16

ACKNOWLEDGEMENT

I am pleased to present this project report titled Image Processing for Plant Health Monitoring and Classification as a culmination of dedicated efforts of myself, my parents, Dr. Anuradha Mam and my friends.

I express my deep appreciation to Dr. Anuradha Chug for her guidance, mentorship, and constructive feedback.

Their insights have been pivotal in steering the project in the right direction and enhancing its quality.

Furthermore, I extend my thanks to USICT, Dwarka for their support, be it through resources, expertise, or collaborative opportunities.

Lastly, my gratitude goes to my family and friends for their unwavering encouragement and understanding during this demanding phase of the project.

This report reflects our collective dedication to advancing knowledge and making a meaningful impact. It is my hope that the insights presented herein contribute positively to the relevant field and pave the way for future research and developments.

I extend my sincere thanks once again for the unwavering support and guidance that I have received throughout this endeavor.

Warm regards,

Ujjwal Gupta

ABSTRACT

Image processing techniques have developed a powerful tool for plant health monitoring and classification. This project report presents a work of image processing methodologies employed in plant health monitoring, with a focus on automated classification. The project work represents approaches of image pre-processing, segmentation, image augmentation and developing deep learning model which are used for disease identification. The pre-processing step in image analysis aims to enhance the quality of image data by eliminating background clutter, noise, and any unwanted distortions. Image segmentation is the most crucial stage in image analysis. It involves dividing an image into uniform regions based on specific criteria, ideally corresponding to real objects within the scene. Due to the high variability in plant disease appearance and environmental conditions, augmenting the dataset is necessary to increase model robustness. For classification, a deep learning model (CNN-based) is made to be trained on the augmented and pre-processed dataset. CNNs are highly effective in image classification tasks due to their ability to automatically extract relevant features from raw images. Lastly the identification of pre-processing algorithms which gives optimal result.

TABLE OF CONTENT

Contents

INTRODUCTION

Agriculture plays a crucial role in feeding the world's population, and the health of crops directly impacts agricultural productivity. Plant diseases can significantly reduce production and quality, which can have a disastrous effect on agricultural output.

Traditional methods for detecting plant diseases often involve manual inspection by experts, which can be time-consuming, expensive, and prone to human error.

This project aims to develop a deep learning model using various image pre-processing and image segmentation techniques that identifies whether plant is healthy or suffering from some disease.

OBJECTIVES

The main objectives of this project are:

1. **Develop an Automated Image-Based Plant Disease Detection Model:** Create a deep learning-based model that utilizes plant images to automatically detect and classify plant diseases, thereby reducing the need for manual inspection.
2. **Analysing various Image Pre-processing and Image Segmentation Techniques:** Implement image pre-processing techniques to improve the quality of input images, and implement image segmentation techniques to identify diseased areas in leaf images.
3. **Handle Variability in Image Data:** Design the deep learning model to effectively manage variability in plant images, including differences in lighting conditions, viewing angles, and noise, ensuring robust disease detection across diverse imaging scenarios.

SCOPE OF WORK

This project focuses on designing and implementing a deep learning-based system for **plant disease classification** with an emphasis on **image processing techniques**.

Image Preprocessing:

- The preprocessing step in image analysis aims to enhance the quality of image data by eliminating background clutter, noise, and any unwanted distortions. This process improves the visibility of important features, making the images more suitable for further processing and analysis. [3]
- Image preprocessing will involve techniques like **bilateral smooth, contrast correction, Gaussian Sharpen, Histogram Equalize etc.**

Image Segmentation

- Image segmentation is the most crucial stage in image analysis. It involves dividing an image into uniform regions based on specific criteria, ideally corresponding to real objects within the scene [1].
- Image segmentation is carried out to differentiate between the affected and unaffected areas of leaf when applied on an image.
- Image Segmentation will involve techniques like **Bradley Local Thresholding, K-means Clustering, Blob Transform.**

Image Augmentation:

- Due to the high variability in plant disease appearance and environmental conditions, augmenting the dataset is necessary to increase model robustness. Augmentation helps simulate different lighting conditions, orientations, and shapes to improve the model's ability to generalize.
- Techniques used for data augmentation include: **Rotation, Flipping**

Convolutional Neural Network Model(CNN)

- For classification, a deep learning model (CNN-based) is made to be trained on the augmented and pre-processed dataset. CNNs are highly effective in image classification tasks due to their ability to automatically extract relevant features from raw images.
- Model Architecture: A CNN architecture with multiple convolutional layers, followed by pooling layers, fully connected layers, and a final softmax output layer.
- Training: The model was trained using a labeled dataset of plant images containing both healthy and diseased samples.
- Evaluation: The performance of the model was evaluated using metrics such as accuracy on a validation dataset.

PROBLEM AND CHALLENGES

- Exploring and understand various image pre-processing and segmentation techniques and implementing them from scratch
- Incorporating these techniques as a custom pre-processing layer in tensor-flow neural network maintaining both performance speed and accuracy of model

DEVELOPMENT ENVIRONMENT

The Standards below are the ones which will be used while developing and thus are not the minimum requirements.

- **Hardware Environments:**

OS	Windows 11
Processor	Intel i5
RAM	16GB
SSD	500GB

Table 1 Hardware Environment

- **Software Environments:**

IDE	Jupyter Notebook or VS Code
Programming Language	Python (3.10 version 64-bit)
Librararies	TensorFlow
	Open-CV
	Scikit-Learn
	Pandas/Numpy

Table 2 Software Environment

METHODOLOGY AND IMPLEMENTATION

Exploring, understanding and implementing various image preprocessing, image segmentation and image augmentation techniques.

Image Pre-processing techniques explored and implemented are:

1. Bilateral Smoothing**:** Bilateral smoothing is an advanced non-linear image preprocessing technique that reduces noise while preserving important edge details. It works by applying a bilateral filter, which combines a spatial Gaussian blur and a range Gaussian blur[2]. It calculates the spatial kernel based on the defined diameter and adjusts this kernel based on the neighborhood of each individual pixel then, it identifies the region of interest and computes the range kernel based on the intensity differences between the pixel in the region of interest and its surrounding pixels; these two kernels are then multiplied to obtain a combined kernel that reflects both spatial and intensity considerations, followed by normalizing the result to ensure the final pixel value maintains the image's brightness In the context of plant leaves disease detection, bilateral smoothing effectively cleans up leaf images by reducing irrelevant noise from the images.

```
def bilateral_filter(image, diameter, sigma_color, sigma_space):
    image = image.astype(np.float32)
    height, width, channels = image.shape
    smoothed_image = np.zeros_like(image)
    half_diameter = diameter // 2
    spatial_kernel = np.zeros((diameter, diameter), dtype=np.float32)
    for i in range(diameter):
        for j in range(diameter):
            x = i - half_diameter
            y = j - half_diameter
            spatial_kernel[i, j] = np.exp(-(x**2 + y**2) / (2 *
sigma_space**2))
    for y in range(height):
        for x in range(width):
            for c in range(channels):
```

```
                y_min = max(y - half_diameter, 0)
                y_max = min(y + half_diameter + 1, height)
                x_min = max(x - half_diameter, 0)
                x_max = min(x + half_diameter + 1, width)
                region = image[y_min:y_max, x_min:x_max, c]
                intensity_diff = region - image[y, x, c]
                range_kernel = np.exp(-(intensity_diff**2) / (2 *
sigma_color**2))
                spatial_kernel_expanded = spatial_kernel[
(y_min - y + half_diameter):(y_max - y + half_diameter),
(x_min - x + half_diameter):(x_max - x + half_diameter)]
                bilateral_kernel = spatial_kernel_expanded * range_kernel
                bilateral_kernel /= bilateral_kernel.sum()
                smoothed_image[y, x, c] = np.sum(bilateral_kernel * region)
   return smoothed_image.astype(np.uint8)
```

2. Contrast Correction: Contrast correction is an image preprocessing technique used to adjust the contrast and brightness of an image. It works by applying a simple formula: corrected_pixel=α×original_pixel+β, where α controls the contrast and β adjusts the brightness. This technique helps enhance important features in the image by making dark regions darker and bright regions brighter, thereby improving the visual distinction between different parts of the image. In the context of plant leaf disease detection, contrast correction is useful for highlighting subtle variations in leaf texture, color, making it easier to detect disease symptoms such as spots, discolorations, and other anomalies.

```
def contrast_correction(image_path, output_path, alpha, beta):
    image = cv2.imread(image_path)
    if image is None:
        raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    image = image.astype(np.float32)
    corrected_pixel = alpha * original_pixel + beta
    corrected_image = alpha * image + beta
    corrected_image = np.clip(corrected_image, 0, 255).astype(np.uint8)
```

3. Gaussian Sharpening: Gaussian Sharpening is an image preprocessing technique that enhances the edges and details of an image by applying a Gaussian filter to blur the image and then subtracting this blurred version from the original image. This process works by reducing the low-frequency components of the image while preserving high-frequency details, which results in sharper edges and more pronounced features. In the context of plant leaves disease detection, Gaussian Sharpen is particularly useful for improving the visibility of subtle leaf features

```
def gaussian_kernel(size, sigma):
    kernel = np.zeros((size, size), dtype=np.float32)
    center = size // 2
    for x in range(size):
        for y in range(size):
            diff = (x - center) ** 2 + (y - center) ** 2
            kernel[x, y] = np.exp(-diff / (2 * sigma ** 2))
    kernel /= (2 * np.pi * sigma ** 2)
    kernel /= kernel.sum()
    return kernel
def convolve(image, kernel):
    if len(image.shape) == 3:
        channels = image.shape[2]
        output = np.zeros_like(image)
        for c in range(channels):
            output[:, :, c] = convolve(image[:, :, c], kernel)
        return output
    else:
        image_height, image_width = image.shape
        kernel_height, kernel_width = kernel.shape
        pad_height = kernel_height // 2
        pad_width = kernel_width // 2
        padded_image = np.pad(image, ((pad_height, pad_height), (pad_width,
pad_width)), mode='constant')
        output = np.zeros_like(image)
        for i in range(image_height):
```

```
            for j in range(image_width):
                output[i, j] = np.sum(padded_image[i:i+kernel_height, j:j+kernel_width] * kernel)
        return output
def gaussian_blur(image, sigma):
    size = int(2 * np.ceil(3 * sigma) + 1)
    kernel = gaussian_kernel(size, sigma)
    return convolve(image, kernel)
def add_weighted(image1, weight1, image2, weight2, gamma):
    return np.clip(weight1 * image1 + weight2 * image2 + gamma, 0, 255).astype(np.uint8)
def sharpen_image(image, sigma):
    blurred = gaussian_blur(image, sigma)
    sharpened_image = add_weighted(image, 1.5, blurred, -0.5, sigma)
    return sharpened_image
def gaussian_sharpen(input_path, output_path, sigma):
    image = cv2.imread(input_path)
    sharpened_image = sharpen_image(image, sigma)
    cv2.imwrite(output_path, sharpened_image)
    cv2.imshow('Original Image', image)
    cv2.imshow('Sharpened Image', sharpened_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

4. **Histogram Equalisation:** Histogram equalization is an image preprocessing technique that enhances the contrast of an image by redistributing the intensity levels. It works by calculating the cumulative distribution function (CDF) from histogram of the pixel intensities, normalising it, mapping the original pixel values to new values based on the CDF, which spreads out the most frequent intensity values and effectively enhances the overall contrast of the image. In the context of plant leaves disease detection, histogram equalization is particularly useful for improving the visibility of subtle leaf features.

```
def equalize_histogram(channel):
    hist, bins = np.histogram(channel.flatten(), 256, [0, 256])
    cdf = (cdf - cdf.min()) * 255 / (cdf.max() - cdf.min())
    cdf = np.ma.filled(cdf, 0).astype('uint8')
```

```
    equalized_channel = cdf[channel]

    return equalized_channel
```

Image Segmentation techniques explored and implemented are:

1. Bradley Local Thresholding**:** Bradley Local Thresholding is an adaptive thresholding technique used for image segmentation that calculates a local threshold for each pixel based on the average intensity of its neighboring pixels within a defined window. The method involves scanning the image in small regions, computing the mean intensity of each local neighbourhood, and then determining the threshold by applying a fraction of this mean to create a binary image that distinguishes foreground from background. In the context of plant leaves disease detection, Bradley Local Thresholding effectively isolates diseased areas or anomalies in leaf images from the background, allowing for more accurate analysis of leaf health and facilitating subsequent feature extraction and classification processes.

```
def bradley_local_thresholding(image, window_size=15, t=0.15):

    image = np.array(image)

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    integral_image = cv2.integral(gray)

    thresholded_image = np.zeros_like(gray, dtype=np.uint8)

    height, width = gray.shape

    half_window = window_size // 2

    for y in range(height):

        for x in range(width):

            x1 = max(x - half_window, 0)

            x2 = min(x + half_window, width - 1)

            y1 = max(y - half_window, 0)

            y2 = min(y + half_window, height - 1)

            local_sum = integral_image[y2 + 1, x2 + 1] - integral_image[y1,
x2 + 1] - integral_image[y2 + 1, x1] + integral_image[y1, x1]

            num_pixels = (x2 - x1 + 1) * (y2 - y1 + 1)

            local_threshold = (1.0 - t) * (local_sum / num_pixels)

            if gray[y, x] < local_threshold:

                thresholded_image[y, x] = 0

            else:

                thresholded_image[y, x] = 255
```

```
    thresholded_image_rgb = cv2.cvtColor(thresholded_image,
cv2.COLOR_GRAY2BGR)

    return thresholded_image_rgb
```

2. K-means Clustering: **K-means clustering** is an unsupervised learning algorithm used for image segmentation that partitions an image into distinct regions based on pixel intensity values. The process begins by selecting a predetermined number of clusters (K) and randomly initializing K centroids, after which each pixel is assigned to the nearest centroid based on its intensity, forming clusters of similar pixels. In plant leaves disease detection, K-means clustering helps isolate different regions of a leaf image, such as healthy and diseased areas, by grouping similar color or intensity values, making it easier to identify patterns or anomalies indicative of disease and facilitating further analysis and classification.

```
def initialize_centers(pixels, k):

    np.random.seed(42)

    random_indices = np.random.choice(pixels.shape[0], k, replace=False)

    centers = pixels[random_indices]

    return centers

def assign_clusters(pixels, centers):

    distances = np.linalg.norm(pixels[:, np.newaxis] - centers, axis=2)

    labels = np.argmin(distances, axis=1)

    return labels

def recompute_centers(pixels, labels, k):

    centers = np.array([pixels[labels == i].mean(axis=0) for i in
range(k)])

    return centers

def k_means_clustering(image_path, output_path, k=3, max_iters=100, tol=1e-
4):

    original_image = cv2.imread(image_path)

    if original_image is None:

        raise FileNotFoundError("Image not found. Please provide a valid
image path.")

    original_image_rgb = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

    pixels = original_image_rgb.reshape((-1, 3))

    pixels = np.float32(pixels)

    centers = initialize_centers(pixels, k)
```

```
    for i in range(max_iters):
        labels = assign_clusters(pixels, centers)
        new_centers = recompute_centers(pixels, labels, k)
        if np.linalg.norm(new_centers - centers) < tol:
            break
        centers = new_centers
    segmented_image =
centers[labels].reshape(original_image_rgb.shape).astype(np.uint8)
    cv2.imwrite(output_path, cv2.cvtColor(segmented_image,
cv2.COLOR_RGB2BGR))
    palette = np.uint8(centers)
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 3, 1)
    plt.imshow(original_image_rgb)
    plt.title("Original Image")
    plt.axis("off")
    plt.subplot(1, 3, 2)
    plt.imshow(segmented_image)
    plt.title(f"K-means Segmented Image (k={k})")
    plt.axis("off")
    plt.subplot(1, 3, 3)
    plt.imshow([palette])
    plt.title("Cluster Colors")
    plt.axis("off")
    plt.show()
```

3. Blob transform**:** Blob transform is a technique used in image segmentation to detect and identify regions or "blobs" that stand out from the rest of the image, such as areas that differ in intensity, color, or texture. It works by analyzing the image to find connected regions that match certain criteria (like size or intensity), helping to isolate important areas. In the context of plant leaf disease detection, blob transform is useful for identifying diseased spots, lesions, or discoloration on leaves, which may appear as distinct blobs. By segmenting these affected areas from healthy parts of the leaf, it aids in diagnosing plant diseases more accurately.

```
params = cv2.SimpleBlobDetector_Params()
params.filterByArea = True
params.minArea = 100
params.maxArea = 5000
params.filterByCircularity = False
params.filterByConvexity = False
params.filterByInertia = False
def blob_detection(image, params):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    detector = cv2.SimpleBlobDetector_create(params)
    keypoints = detector.detect(gray)
    im_with_keypoints = cv2.drawKeypoints(image, keypoints, np.array([]),
(0, 0, 255),
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return im_with_keypoints
```

Image Augmentation techniques explored and implemented are:

1. Rotation**:** Rotation is an image augmentation technique that involves rotating an image by a certain angle, either clockwise or counterclockwise, to create new variations of the original image. This technique is beneficial in plant leaves disease detection as it helps to generate a diverse dataset by providing multiple perspectives of the same leaf, regardless of its orientation in the original image. By augmenting the dataset with rotated images, machine learning models can become more robust and better at recognizing disease symptoms, as they learn to identify features and patterns from different angles, ultimately improving the accuracy and reliability of the disease detection process.
2. Horizontal And Vertical Flips**:** Horizontal and vertical flips are image augmentation techniques that involve mirroring an image along the horizontal or vertical axis, respectively, to create new versions of the original image. Horizontal flipping involves flipping the image left to right, while vertical flipping involves flipping it top to bottom. In the context of plant leaves disease detection, these techniques are valuable as they increase the diversity of the training dataset by providing multiple orientations of the same leaf, allowing machine learning models to learn to recognize disease symptoms regardless of the leaf's position or orientation. This augmentation enhances the model's

robustness and improves its ability to accurately identify various diseases by ensuring that it is trained on a comprehensive representation of the data.

```
def rotate_image(image, angle):
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w, h))
    return rotated
def horizontal_flip(image):
    return cv2.flip(image, 1)
def vertical_flip(image):
    return cv2.flip(image, 0)
```

OUTPUT

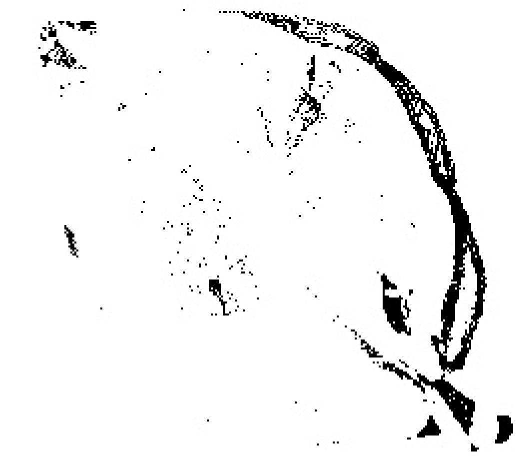
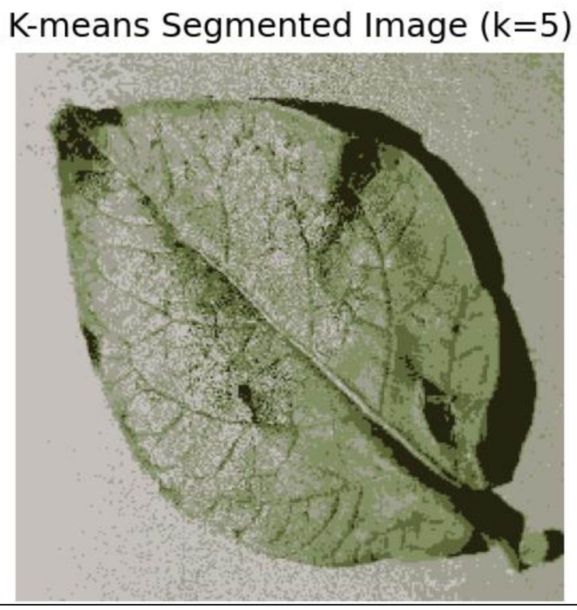
Image Preprocessing

Input Image	
Bilateral Smoothing	
Contrast Correction	

Gaussian Sharpening	
Histogram Equalize	

Table 3 Image Preprocessing Output

Image Segmentation

Input Image	
Bradley Local Thresholding	
K-means Clustering	

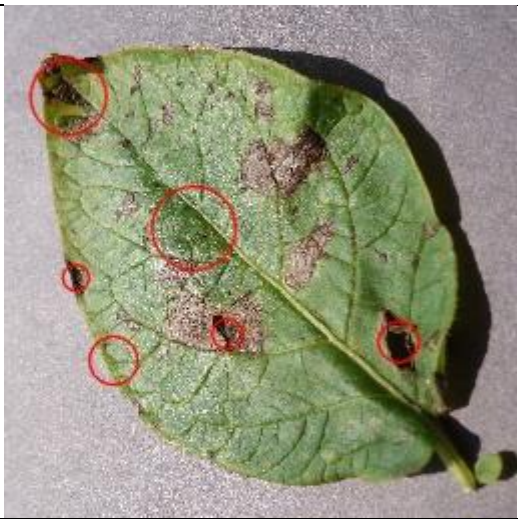
	
Blob transform	

Table 4 Image Segmentation Output

Image Augmentation

Input Image	

Image Rotation	
Horizontal Flips	
Vertical Flips	

Table 5 Image Augmentation Output

DEEP LEARNING MODEL CODE

```
import tensorflow as tf

from tensorflow import keras

from keras import models,layers

import matplotlib.pyplot as plt

import numpy as np

import cv2

IMAGE_SIZE = 256

BATCH_SIZE = 32

CHANNELS = 3

EPOCHS=5

dataset = tf.keras.preprocessing.image_dataset_from_directory(

    "PlantVillage",

    image_size= (IMAGE_SIZE, IMAGE_SIZE),

    batch_size=BATCH_SIZE,

    shuffle=True,

)

def
get_dataset_partition_tf(ds,train_split=0.8,val_split=0.1,test_split=0.1,sh
uffle=True,shuffle_size=10000):

    ds_size = len(ds)

    if shuffle:

        ds = ds.shuffle(shuffle_size,seed=12)

    train_size = int(train_split* ds_size)

    val_size = int(val_split*ds_size)

    train_ds = ds.take(train_size)

    test_ds = ds.skip(train_size)

    val_ds = test_ds.skip(val_size)

    test_ds = test_ds.take(val_size)

    return train_ds,val_ds,test_ds

train_ds,val_ds,test_ds = get_dataset_partition_tf(dataset)

train_ds = train_ds.cache()

train_ds = train_ds.shuffle(1000)

train_ds = train_ds.prefetch(buffer_size= tf.data.AUTOTUNE)

val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=
tf.data.AUTOTUNE)
```

```
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=
tf.data.AUTOTUNE)
```

```
for image_batch, label_batch in dataset.take(1):

    print(f"Shape of image batch: {image_batch.shape}")

    print(f"Number of images in this batch: {image_batch.shape[0]}")

    print(image_batch[0].shape)

print(f"Total number of batches: {len(dataset)}")

print(f"Total number of images: {len(dataset) * BATCH_SIZE}")
```

Output

```
Shape of image batch: (32, 256, 256, 3)
Number of images in this batch: 32
(256, 256, 3)
Total number of batches: 68
Total number of images: 2176
```

```
@register_keras_serializable()

class BilateralSmoothingLayer(tf.keras.layers.Layer):

    def __init__(self, diameter=15, sigma_color=75, sigma_space=75,
**kwargs):

        super(BilateralSmoothingLayer, self).__init__(**kwargs)

        self.diameter = diameter

        self.sigma_color = sigma_color

        self.sigma_space = sigma_space

    def call(self, inputs):

        def bilateral_smooth(image):

            image = image.numpy().astype(np.float32)

            smoothed_image = cv2.bilateralFilter(image, self.diameter,
self.sigma_color, self.sigma_space)

            return smoothed_image

        smoothed_batch = tf.map_fn(

            lambda img: tf.py_function(func=bilateral_smooth, inp=[img],
Tout=tf.float32),

            inputs,

           fn_output_signature=tf.TensorSpec(shape=inputs.shape[1:],
dtype=tf.float32)

        )

        return smoothed_batch
```

```
@register_keras_serializable()
class BradleyLocalThresholdingLayer(tf.keras.layers.Layer):
    def __init__(self, block_size=15, threshold=0.1, **kwargs):
        super(BradleyLocalThresholdingLayer, self).__init__(**kwargs)
        self.block_size = block_size
        self.threshold = threshold

    def call(self, inputs):
        def bradley_threshold(image):
            image = image.numpy().astype(np.uint8)
            gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            mean_c = cv2.adaptiveThreshold(
                gray_image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, self.block_size, self.threshold
            )
            rgb_image = cv2.cvtColor(mean_c, cv2.COLOR_GRAY2RGB)
            return rgb_image
        thresholded_batch = tf.map_fn(
            lambda img: tf.py_function(func=bradley_threshold, inp=[img],
Tout=tf.float32),
            inputs,
           fn_output_signature=tf.TensorSpec(shape=(inputs.shape[1],
inputs.shape[2], 3), dtype=tf.float32)
        )
        return thresholded_batch
```

```
@register_keras_serializable()
class ContrastCorrectionLayer(tf.keras.layers.Layer):
    def __init__(self, alpha=1.0, beta=0.0, **kwargs):
        super(ContrastCorrectionLayer, self).__init__(**kwargs)
        self.alpha = alpha
        self.beta = beta
    def call(self, inputs):
        def contrast_correction(image):
            image = image.numpy().astype(np.float32)
            corrected_image = cv2.convertScaleAbs(image, alpha=self.alpha,
beta=self.beta)
```

```
            return corrected_image
        corrected_batch = tf.map_fn(
            lambda img: tf.py_function(func=contrast_correction, inp=[img], 
Tout=tf.float32),
            inputs,
            fn_output_signature=tf.TensorSpec(shape=inputs.shape[1:], 
dtype=tf.float32)
        )
        return corrected_batch
```

```
@register_keras_serializable()
class GaussianSharpeningLayer(tf.keras.layers.Layer):
    def __init__(self, kernel_size=5, sigma=1.0, alpha=1.5, **kwargs):
        super(GaussianSharpeningLayer, self).__init__(**kwargs)
        self.kernel_size = kernel_size
        self.sigma = sigma
        self.alpha = alpha
    def call(self, inputs):
        def gaussian_sharpen(image):
            image = image.numpy().astype(np.float32)
            blurred = cv2.GaussianBlur(image, (self.kernel_size, 
self.kernel_size), self.sigma)
            sharpened = cv2.addWeighted(image, self.alpha, blurred, -0.5, 
0)
            return sharpened
        sharpened_batch = tf.map_fn(
            lambda img: tf.py_function(func=gaussian_sharpen, inp=[img], 
Tout=tf.float32),
            inputs,
            fn_output_signature=tf.TensorSpec(shape=inputs.shape[1:], 
dtype=tf.float32)
        )
        return sharpened_batch
```

```
@register_keras_serializable()
class HistogramEqualizationLayer(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super(HistogramEqualizationLayer, self).__init__(**kwargs)
```

```
    def call(self, inputs):
        def histogram_equalize(image):
            image = image.numpy().astype(np.uint8)
            if len(image.shape) == 3 and image.shape[2] == 3:
                image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
                image[:, :, 0] = cv2.equalizeHist(image[:, :, 0])
                image = cv2.cvtColor(image, cv2.COLOR_YCrCb2RGB)
            else:
                image = cv2.equalizeHist(image)
            return image
        equalized_batch = tf.map_fn(
            lambda img: tf.py_function(func=histogram_equalize, inp=[img],
Tout=tf.float32),
            inputs,
            fn_output_signature=tf.TensorSpec(shape=inputs.shape[1:],
dtype=tf.float32)
        )
        return equalized_batch
```

```
@register_keras_serializable()
class KMeansClusteringLayer(tf.keras.layers.Layer):
    def __init__(self, n_clusters=3, **kwargs):
        super(KMeansClusteringLayer, self).__init__(**kwargs)
        self.n_clusters = n_clusters
    def call(self, inputs):
        def kmeans_clustering(image):
            image = image.numpy().astype(np.float32)
            pixel_values = image.reshape((-1, 3))
            pixel_values = np.float32(pixel_values)
            criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
100, 0.2)
            _, labels, centers = cv2.kmeans(pixel_values, self.n_clusters,
None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
            centers = np.uint8(centers)
            segmented_image = centers[labels.flatten()]
            segmented_image = segmented_image.reshape(image.shape)
            return segmented_image
```

```
        clustered_batch = tf.map_fn(
            lambda img: tf.py_function(func=kmeans_clustering, inp=[img],
Tout=tf.float32),
            inputs,
            fn_output_signature=tf.TensorSpec(shape=inputs.shape[1:],
dtype=tf.float32)
        )
        return clustered_batch
```

```
@register_keras_serializable()
class BlobSegmentationLayer(tf.keras.layers.Layer):
    def __init__(self, min_area=100, max_area=1000, **kwargs):
        super(BlobSegmentationLayer, self).__init__(**kwargs)
        self.min_area = min_area
        self.max_area = max_area
    def call(self, inputs):
        def blob_segmentation(image):
            image = image.numpy().astype(np.uint8)
            gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            _, binary_image = cv2.threshold(gray_image, 127, 255,
cv2.THRESH_BINARY)
            params = cv2.SimpleBlobDetector_Params()
            params.filterByArea = True
            params.minArea = self.min_area
            params.maxArea = self.max_area
            detector = cv2.SimpleBlobDetector_create(params)
            keypoints = detector.detect(binary_image)
            blank = np.zeros((1, 1))
            blobs = cv2.drawKeypoints(image, keypoints, blank, (0, 0, 255),
            cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
            return blobs
        segmented_batch = tf.map_fn(
            lambda img: tf.py_function(func=blob_segmentation, inp=[img],
Tout=tf.float32),
            inputs,
            fn_output_signature=tf.TensorSpec(shape=inputs.shape[1:],
dtype=tf.float32)
```

```
        )
        return segmented_batch
```

```
resize_and_rescale = tf.keras.Sequential([
    layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.Rescaling(1./255)
])
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
])
```

```
n_classes = 3
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
model1 = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    resize_and_rescale,
    data_augmentation,
    BilateralSmoothingLayer(diameter=15, sigma_color=75, sigma_space=75),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax')
])
model1.compile(
```

```
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
model1.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
sequential_1 (Sequential)	(None, 256, 256, 3)	0
bilateral_smoothing_layer (BilateralSmoothingLayer)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_3 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_4 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 64)	16,448
dense_1 (Dense)	(None, 3)	195

Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)

```
n_classes = 3
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
model2 = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    resize_and_rescale,
```

```
    data_augmentation,
    BradleyLocalThresholdingLayer(block_size=15, threshold=0.1),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax')
])
model2.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
model2.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
sequential_1 (Sequential)	(None, 256, 256, 3)	0
bradley_local_thresholding_lay… (BradleyLocalThresholdingLayer)	(None, 256, 256, 3)	0
conv2d_6 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_7 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_8 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_8 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_9 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_9 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_10 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_10 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_11 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_11 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_1 (Flatten)	(None, 256)	0
dense_2 (Dense)	(None, 64)	16,448
dense_3 (Dense)	(None, 3)	195

```
Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)
```

```
n_classes = 3
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
model3 = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    resize_and_rescale,
    data_augmentation,
    ContrastCorrectionLayer(alpha=1.5, beta=0.0),
    layers.Conv2D(32, (3, 3), activation='relu'
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax')
])
model3.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
model3.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
sequential_1 (Sequential)	(None, 256, 256, 3)	0
contrast_correction_layer (ContrastCorrectionLayer)	(None, 256, 256, 3)	0
conv2d_12 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_12 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_13 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_13 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_14 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_14 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_15 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_15 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_16 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_16 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_17 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_17 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_2 (Flatten)	(None, 256)	0
dense_4 (Dense)	(None, 64)	16,448
dense_5 (Dense)	(None, 3)	195

```
Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)
```

```
n_classes = 3
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
model4 = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    resize_and_rescale,
    data_augmentation,
    GaussianSharpeningLayer(kernel_size=5, sigma=1.0, alpha=1.5),
    layers.Conv2D(32, (3, 3), activation='relu'
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax')
])
model4.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
model4.summary()
```

```
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
sequential_1 (Sequential)	(None, 256, 256, 3)	0
gaussian_sharpening_layer (GaussianSharpeningLayer)	(None, 256, 256, 3)	0
conv2d_18 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_18 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_19 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_19 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_20 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_20 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_21 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_21 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_22 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_22 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_23 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_23 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_3 (Flatten)	(None, 256)	0
dense_6 (Dense)	(None, 64)	16,448
dense_7 (Dense)	(None, 3)	195

```
Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)
```

```
n_classes = 3
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
model5 = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    resize_and_rescale,
    data_augmentation,
    HistogramEqualizationLayer(),
    layers.Conv2D(32, (3, 3), activation='relu'
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax')
])
model5.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
model5.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
sequential_1 (Sequential)	(None, 256, 256, 3)	0
histogram_equalization_layer (HistogramEqualizationLayer)	(None, 256, 256, 3)	0
conv2d_24 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_24 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_25 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_25 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_26 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_26 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_27 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_27 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_28 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_28 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_29 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_29 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_4 (Flatten)	(None, 256)	0
dense_8 (Dense)	(None, 64)	16,448
dense_9 (Dense)	(None, 3)	195

```
Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)
```

```
n_classes = 3
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
model6 = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    resize_and_rescale,
    data_augmentation,
    KMeansClusteringLayer(n_clusters=3),
    layers.Conv2D(32, (3, 3), activation='relu'
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax')
])
model6.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
model6.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
sequential_1 (Sequential)	(None, 256, 256, 3)	0
k_means_clustering_layer (KMeansClusteringLayer)	(None, 256, 256, 3)	0
conv2d_30 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_30 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_31 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_31 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_32 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_32 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_33 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_33 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_34 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_34 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_35 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_35 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_5 (Flatten)	(None, 256)	0
dense_10 (Dense)	(None, 64)	16,448
dense_11 (Dense)	(None, 3)	195

Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)

```
n_classes = 3
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
model7 = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    resize_and_rescale,
    data_augmentation,
    BlobSegmentationLayer(min_area=100, max_area=1000),
    layers.Conv2D(32, (3, 3), activation='relu'
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax')
])
model7.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
model7.summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
sequential_1 (Sequential)	(None, 256, 256, 3)	0
blob_segmentation_layer (BlobSegmentationLayer)	(None, 256, 256, 3)	0
conv2d_36 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_36 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_37 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_37 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_38 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_38 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_39 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_39 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_40 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_40 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_41 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_41 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_6 (Flatten)	(None, 256)	0
dense_12 (Dense)	(None, 64)	16,448
dense_13 (Dense)	(None, 3)	195

```
Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)
```

```
history1 = model1.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
    verbose= 1
)
```

```
Epoch 1/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.4403 - loss: 0.9488 - val_accuracy: 0.5898 - val_loss: 0.8007
Epoch 2/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 138s 3s/step - accuracy: 0.6164 - loss: 0.7535 - val_accuracy: 0.7695 - val_loss: 0.5927
Epoch 3/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 143s 3s/step - accuracy: 0.7220 - loss: 0.6074 - val_accuracy: 0.8398 - val_loss: 0.3854
Epoch 4/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 169s 3s/step - accuracy: 0.8431 - loss: 0.3798 - val_accuracy: 0.8672 - val_loss: 0.3098
Epoch 5/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 167s 3s/step - accuracy: 0.8637 - loss: 0.3319 - val_accuracy: 0.9023 - val_loss: 0.2339
```

```
model1.save("model1.keras")
```

```
history2 = model2.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
    verbose= 1
)
```

```
Epoch 1/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 128s 2s/step - accuracy: 0.4034 - loss: 3.8175 - val_accuracy: 0.4727 - val_loss: 0.9392
Epoch 2/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.4751 - loss: 0.9275 - val_accuracy: 0.4336 - val_loss: 0.9433
Epoch 3/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.4428 - loss: 0.9204 - val_accuracy: 0.4727 - val_loss: 0.9429
Epoch 4/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 93s 2s/step - accuracy: 0.4850 - loss: 0.9028 - val_accuracy: 0.4727 - val_loss: 0.9464
Epoch 5/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.4928 - loss: 0.9098 - val_accuracy: 0.4727 - val_loss: 0.9402
```

```
model2.save("model2.keras")
```

```
history3 = model3.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
```

```
    verbose= 1
)
```

```
Epoch 1/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 98s 2s/step - accuracy: 0.5338 - loss: 0.8936 - val_accuracy: 0.6836 - val_loss: 0.6785
Epoch 2/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 93s 2s/step - accuracy: 0.7199 - loss: 0.6132 - val_accuracy: 0.8125 - val_loss: 0.4364
Epoch 3/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.8260 - loss: 0.3947 - val_accuracy: 0.8711 - val_loss: 0.3077
Epoch 4/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.8679 - loss: 0.3022 - val_accuracy: 0.8906 - val_loss: 0.2708
Epoch 5/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.9208 - loss: 0.2239 - val_accuracy: 0.9062 - val_loss: 0.2373
```

```
model3.save("model3.keras")
```

```
history4 = model4.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
    verbose= 1
)
```

```
Epoch 1/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 155s 3s/step - accuracy: 0.4560 - loss: 0.9445 - val_accuracy: 0.5117 - val_loss: 0.8510
Epoch 2/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.6592 - loss: 0.6901 - val_accuracy: 0.7734 - val_loss: 0.5790
Epoch 3/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 121s 2s/step - accuracy: 0.7763 - loss: 0.5053 - val_accuracy: 0.7969 - val_loss: 0.4361
Epoch 4/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.8402 - loss: 0.3768 - val_accuracy: 0.8711 - val_loss: 0.3035
Epoch 5/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.8939 - loss: 0.2630 - val_accuracy: 0.8984 - val_loss: 0.2427
```

```
model4.save("model4.keras")
```

```
history5 = model5.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
    verbose= 1
)
```

```
Epoch 1/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 119s 2s/step - accuracy: 0.4517 - loss: 1.0345 - val_accuracy: 0.4727 - val_loss: 26.2887
Epoch 2/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 94s 2s/step - accuracy: 0.4478 - loss: 0.9312 - val_accuracy: 0.4766 - val_loss: 25.5411
Epoch 3/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 84s 2s/step - accuracy: 0.4426 - loss: 0.9065 - val_accuracy: 0.4805 - val_loss: 26.2211
Epoch 4/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.4637 - loss: 0.8950 - val_accuracy: 0.4805 - val_loss: 28.2390
Epoch 5/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.4464 - loss: 0.8995 - val_accuracy: 0.7148 - val_loss: 66.5611
```

```
model5.save("model5.keras")
```

```
history6 = model6.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
    verbose= 1
)
```

```
Epoch 1/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 232s 4s/step - accuracy: 0.4774 - loss: 1.0923 - val_accuracy: 0.4336 - val_loss: 1.0765
Epoch 2/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 210s 4s/step - accuracy: 0.4580 - loss: 1.0695 - val_accuracy: 0.4336 - val_loss: 1.0578
Epoch 3/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 215s 4s/step - accuracy: 0.4728 - loss: 1.0483 - val_accuracy: 0.4336 - val_loss: 1.0412
Epoch 4/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 207s 4s/step - accuracy: 0.4707 - loss: 1.0362 - val_accuracy: 0.4336 - val_loss: 1.0271
Epoch 5/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 216s 4s/step - accuracy: 0.4843 - loss: 1.0165 - val_accuracy: 0.4336 - val_loss: 1.0147
```

```
model6.save("model6.keras")
```

```
history7 = model7.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
    verbose= 1
)
```

```
Epoch 1/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 104s 2s/step - accuracy: 0.4541 - loss: 1.0476 - val_accuracy: 0.4336 - val_loss: 0.9625
Epoch 2/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 98s 2s/step - accuracy: 0.4835 - loss: 0.9089 - val_accuracy: 0.4336 - val_loss: 1.0055
Epoch 3/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 115s 2s/step - accuracy: 0.4535 - loss: 0.9286 - val_accuracy: 0.3789 - val_loss: 1.1485
Epoch 4/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 122s 2s/step - accuracy: 0.4381 - loss: 0.8958 - val_accuracy: 0.3867 - val_loss: 2.1114
Epoch 5/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 118s 2s/step - accuracy: 0.4771 - loss: 0.9021 - val_accuracy: 0.5898 - val_loss: 1.7729
```

```
model7.save("model7.keras")
```

```
scores1 = model1.evaluate(test_ds)
print(f"Testing Accuracy: {scores1[1]}")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 24s 993ms/step - accuracy: 0.9031 - loss: 0.2683
Testing Accuracy: 0.9114583134651184
```

```
scores2 = model2.evaluate(test_ds)
print(f"Testing Accuracy: {scores2[1]}")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 440ms/step - accuracy: 0.4040 - loss: 1.0168
Testing Accuracy: 0.46875
```

```
scores3 = model3.evaluate(test_ds)
print(f"Testing Accuracy: {scores3[1]}")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 427ms/step - accuracy: 0.9126 - loss: 0.2546
Testing Accuracy: 0.9114583134651184
```

```
scores4 = model4.evaluate(test_ds)

print(f"Testing Accuracy: {scores4[1]}")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 425ms/step - accuracy: 0.9226 - loss: 0.2413
Testing Accuracy: 0.9114583134651184
```

```
scores5 = model5.evaluate(test_ds)

print(f"Testing Accuracy: {scores5[1]}")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 455ms/step - accuracy: 0.6548 - loss: 66.7426
Testing Accuracy: 0.6354166865348816
```

```
scores6 = model6.evaluate(test_ds)

print(f"Testing Accuracy: {scores6[1]}")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 13s 2s/step - accuracy: 0.4137 - loss: 1.0105
Testing Accuracy: 0.4166666567325592
```

```
scores7 = model7.evaluate(test_ds)

print(f"Testing Accuracy: {scores7[1]}")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 2s 408ms/step - accuracy: 0.5984 - loss: 1.7488
Testing Accuracy: 0.5572916865348816
```

```
accuracies = [scores1[1], scores2[1], scores3[1], scores4[1], scores5[1], scores6[1], scores7[1]]

sum_accuracies = sum(acc for acc in accuracies)

weighted_accuracies = []

for acc in accuracies:

    if acc <= 0.5:

        weighted_accuracies.append(acc * 0.2)

    else:

        weighted_accuracies.append(acc * (acc / sum_accuracies))

average_weighted_accuracy = sum(weighted_accuracies)

print(f"Average Weighted Accuracy: {average_weighted_accuracy}")
```

```
Average Weighted Accuracy: 0.8433892390492357
```

```
class_names = dataset.class_names
class_names
```

```
['early_blight', 'healthy', 'late_blight']
```

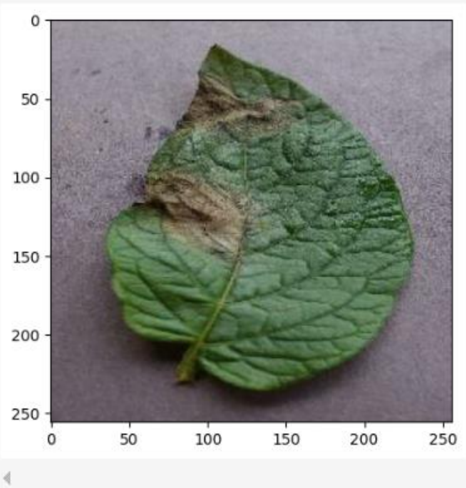
```
for image_batch, labels_batch in test_ds.take(1):

    first_image = image_batch[0].numpy().astype('uint8')

    first_label = labels_batch[0]
```

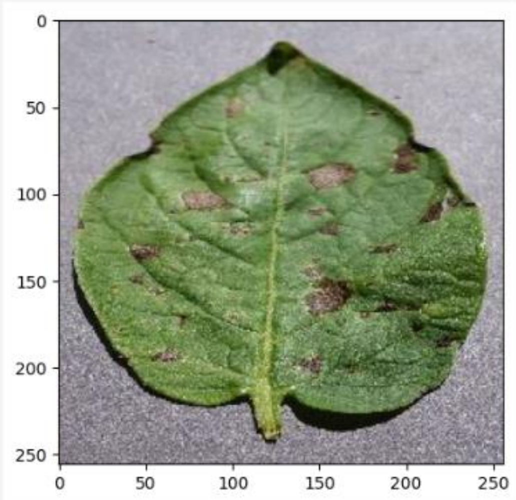
```python
print("First image to predict")
plt.imshow(first_image)
plt.show()
print("First image's actual label:", class_names[first_label.numpy()])
predictions = [
    np.argmax(model1.predict(image_batch)[0]),
    np.argmax(model2.predict(image_batch)[0]),
    np.argmax(model3.predict(image_batch)[0]),
    np.argmax(model4.predict(image_batch)[0]),
    np.argmax(model5.predict(image_batch)[0]),
    np.argmax(model6.predict(image_batch)[0]),
    np.argmax(model7.predict(image_batch)[0])
]
for i, prediction in enumerate(predictions):
    print(f"Model {i+1} predicted label: {class_names[prediction]}")
overall_prediction = max(set(predictions), key=predictions.count)
print("Overall predicted label:", class_names[overall_prediction])
```

```
First image to predict
```



```
First image's actual label: late_blight
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 481ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 485ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 477ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 451ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 450ms/step
Model 1 predicted label: late_blight
Model 2 predicted label: early_blight
Model 3 predicted label: late_blight
Model 4 predicted label: late_blight
Model 5 predicted label: late_blight
Model 6 predicted label: late_blight
Model 7 predicted label: late_blight
----------------------------------
Overall predicted label: late_blight
```

```
First image to predict
```



```
First image's actual label: early_blight
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 835ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 588ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 463ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 394ms/step
Model 1 predicted label: early_blight
Model 2 predicted label: early_blight
Model 3 predicted label: early_blight
Model 4 predicted label: early_blight
Model 5 predicted label: early_blight
Model 6 predicted label: late_blight
Model 7 predicted label: late_blight
----------------------------------
Overall predicted label: early_blight
```

CONCLUSION

1. Implemented wide array of image preprocessing and image segmentation techniques is crucial for enhancing the accuracy of plant disease classification.
2. Noise reduction techniques like bilateral smoothing ensures that the model focuses on the important areas of the image, such as diseased parts in leaf, and not on irrelevant background noise.
3. Image Enhancement techniques like contrast correction and Gaussian Sharpening improve the visibility of plant diseases, enabling the system to detect important features.
4. Image Segmentation techniques such as Bradley Local Thresholding, K-means Clustering, and Blob Transform, are vital for isolating affected areas from healthy regions. This segmentation allows for more focused and accurate disease detection and classification.
5. Implemented a deep learning ensemble model incorporating these techniques as a custom preprocessing layer in Tensor flow network which successfully classifies potato plant images into 3 classes **early_blight, healthy, late_blight** with overall accuracy of 84.34 percent.
6. The optimal result on the current dataset is given by following image processing techniques
    - **Bilateral Smoothing**
    - **Gaussian Sharpen**
    - **Contrast Correction**

ADVANTAGES

1. **Efficiency:** This model if further developed can automate the disease detection process, reducing the time and effort required by human experts.
2. **Accuracy:** Deep learning models, especially CNNs, can achieve high accuracy in classifying plant diseases, even with subtle visual differences.
3. **Scalability:** The model can be scaled to handle multiple plant species and diseases, improving its applicability in diverse agricultural settings.
4. **Early Detection:** Identifying and classifying plant diseases can offer wider environmental and social benefits, while also improving crop quality and productivity.

REFERENCES

[1] Abdulateef, S.K. and Salman, M.D., 2021. A Comprehensive Review of Image Segmentation Techniques. *Iraqi Journal for Electrical & Electronic Engineering*, *17*(2).

[2] Barash, D. and Comaniciu, D., 2004. A common framework for nonlinear diffusion, adaptive smoothing, bilateral filtering and mean shift. *Image and Vision Computing*, *22*(1), pp.73-81.

[3] Oo, Y.M. and Htun, N.C., 2018. Plant leaf disease detection and classification using image processing. *International Journal of Research and Engineering*, *5*(9), pp.516-523.