**PLANT DISEASE DETECTION SYSTEM FOR SUSTAINABLE AGRICULTURE**

**PROBLEM STATEMENT**

Develop a CNN-based model capable of detecting and classifying plant diseases from images of leaves of various crops such as apples, cherry, grapes, and corn. The model should accurately identify both healthy and diseased leaves while predicting the specific type of disease. This system will aid in precision agriculture by enabling early detection and effective disease management.

**AIM**

To design and implement a CNN-based model that accurately detects and classifies plant diseases from leaf images, identifying both healthy and diseased conditions. The system aims to support precision agriculture by enabling early diagnosis and improving crop management practices.

**OBJECTIVE**

The objectives of this project are to:

**Dataset Acquisition and Preparation:**

• Collect a diverse dataset of leaf images representing various plant species, including healthy and diseased samples.

• Clean and Preprocess the images to ensure uniform size, resolution, and quality.

• Label the images with corresponding plant species and disease types.

**Exploratory Data Analysis**:

• Analyze dataset characteristics such as image size, color distribution, and class balance.

**Feature Extraction:**

• Extract visual features such as leaf texture, color patterns, and vein structures using CNN layers.

• Experiment with different CNN architectures and transfer learning models to improve feature

extraction.

**Model Development:**

• Train and evaluate deep learning models, focusing on CNN architectures for image classification.

• Optimize model hyperparameters such as learning rate and number of layers to enhance

performance.

**Model Evaluation:**

• Assess model performance using metrics like accuracy, precision, recall, F1-score, and confusion

matrix.

• Identify performance gaps, refine the model, and iterate to improve accuracy and generalization.

**About Data**

This dataset consists of about 87K rgb images of healthy and diseased crop leaves which is categorized into 38 different classes. The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purpose.

**Data source** : Data Set Link https://www.kaggle.com/datasets/vipoooool/new-plant-diseases dataset/data?select=New+Plant+Diseases+Dataset%28Augmented%29

**Tools Used**

**1. Python:**

• Utilized python in plant disease detection project to build a convolutional neural network (CNN) using frameworks like TensorFlow or Keras. It enables loading, preprocessing, and augmenting image datasets efficiently.

**2. Google Colab Description**

• Utilized Google Colab for an interactive and collaborative coding environment, providing a cloud-based platform for efficient model development.

• Google Colab facilitated seamless execution of deep learning models, offering access to powerful hardware like GPUs and TPUs for faster training.

• The platform also enabled easy sharing and collaboration, allowing team members to work together on model development, documentation, and visualization in real time.

**3. Framework – Steamlit**

• Used Streamlit which is an open-source Python framework for data scientists and AI/ML engineers to deliver interactive data apps – in only a few lines of code.

**Findings and Insights**

**Data Preparation:**

• Dataset Loading: The datasets, consisting of leaf images from different plants, are loaded into the environment for training and testing purposes. The images are likely stored in separate directories for healthy and diseased leaves.

• Class Identification: The dataset includes multiple classes corresponding to healthy and different types of plant diseases, allowing the model to distinguish between them.

**Model Training:**

• CNN Model: A Convolutional Neural Network (CNN) model is trained on the labeled leaf images to learn visual patterns associated with healthy and diseased conditions.

• Hyperparameter Tuning: The CNN architecture and hyperparameters (e.g., learning rate, number of filters, kernel size, batch size) are optimized to improve the model's accuracy in classifying diseases and healthy conditions.

**Prediction Function:**

• Image Preprocessing: The prediction function preprocesses input images by resizing them to a fixed size (e.g., 224x224 pixels) and reshaping them to be compatible with the CNN model.

• Model Prediction: The trained model predicts the disease type or identifies the leaf as healthy based on the extracted features.

• Output: The predicted disease type or healthy condition is displayed, helping to assess the health of the plant.

**Evaluation:**

• Testing: The model's performance is evaluated on a separate testing dataset to determine its ability to generalize to unseen data, ensuring it can accurately classify plant diseases in real world scenarios.

• Metrics: Evaluation metrics like accuracy, precision, recall used to measure the model’s classification performance and effectiveness in identifying healthy and diseased plant leaves. Overall, the project demonstrates a functional workflow for plant disease detection using CNNs, though further details about the CNN architecture, hyperparameter tuning, and additional evaluation metrics could offer a more comprehensive understanding.

**CLEANING AND PREPROCESSING THE DATA**

Clean the irrelevant data to avoid corruptions and errors. Then process the data to train the model more effectively.

**1.Import Necessary Libraries:**

Ensure you have all the required libraries imported.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import cv2

import tensorflow as tf

**2.Define Paths and Parameters**:

Set the path to your dataset and the image size.

train\_path = '/content/new plant diseases dataset(augmented)/New Plant Diseases Dataset(Augmented)/train'

valid\_path = '/content/new plant diseases dataset(augmented)/New Plant Diseases Dataset(Augmented)/valid'

size = 224 # Image size to resize to

**3.Check and Resize Images**:

Create a function to check if images can be read and resize them to the desired size. This helps in handling any missing or corrupt images.

def check\_and\_resize\_images(directory, size):

for root, dirs, files in os.walk(directory):

for file in files:

if file.endswith('.jpg') or file.endswith('.png'):

img\_path = os.path.join(root, file)

img = cv2.imread(img\_path)

if img is None:

print(f"Image not readable: {img\_path}")

else:

img\_resized = cv2.resize(img, (size, size))

cv2.imwrite(img\_path, img\_resized)

check\_and\_resize\_images(train\_path, size)

check\_and\_resize\_images(valid\_path, size)

**4.Set Up ImageDataGenerator:**

Create training and validation data generators using TensorFlow’s Keras API. These generators will handle data augmentation and normalization.

train\_generator = tf.keras.preprocessing.image.ImageDataGenerator(

rotation\_range=90,

width\_shift\_range=0.0,

height\_shift\_range=0.0,

shear\_range=0.0,

zoom\_range=0.0,

horizontal\_flip=False,

vertical\_flip=False,

rescale=1/255.0,

validation\_split=0.1

)

train\_data = train\_generator.flow\_from\_directory(

train\_path,

batch\_size=164,

target\_size=(size, size),

subset="training",

color\_mode='rgb',

class\_mode='categorical',

shuffle=True

)

val\_generator = tf.keras.preprocessing.image.ImageDataGenerator(

rescale=1/255.0,

validation\_split=0.1

)

val\_data = val\_generator.flow\_from\_directory(

valid\_path,

batch\_size=164,

target\_size=(size, size),

subset="validation",

color\_mode='rgb',

class\_mode='categorical',

shuffle=False

)

**5.Visualize a Batch of Images:**

It’s always good practice to visualize a batch of images to ensure they are loaded and preprocessed correctly.

def plot\_images(generator):

x\_batch, y\_batch = next(generator)

fig, axes = plt.subplots(1, 5, figsize=(20, 20))

axes = axes.flatten()

for img, ax in zip(x\_batch, axes):

ax.imshow(img)

ax.axis('off')

plt.tight\_layout()

plt.show()

plot\_images(train\_data)

**PERFORM EXPLORATORY DATA ANALYSIS (EDA) TO UNDERSTAND DATA PATTERNS**

* Load the Data: Utilize the existing data generators to load and examine the data.
* Class Distribution: Analyze the distribution of classes.
* Image Properties: Inspect the properties of the images, such as dimensions and color channels.
* Visualize Samples: Display a few images from each class to understand the variations.
* Check for Missing or Corrupt Images: Ensure all images are readable.

**1.Load the Data**

Utilize the existing train\_data and val\_data generators to access the images and labels.

**2. Class Distribution**

Check the distribution of classes in the training and validation datasets.

import matplotlib.pyplot as plt

# Get class labels from the generator

class\_labels = list(train\_data.class\_indices.keys())

# Count the number of samples per class

train\_counts = train\_data.classes

val\_counts = val\_data.classes

# Plot the class distribution

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.hist(train\_counts, bins=len(class\_labels), alpha=0.7, color='blue', edgecolor='black')

plt.title('Training Set Class Distribution')

plt.xlabel('Class')

plt.ylabel('Count')

plt.xticks(range(len(class\_labels)), class\_labels, rotation=90)

plt.subplot(1, 2, 2)

plt.hist(val\_counts, bins=len(class\_labels), alpha=0.7, color='green', edgecolor='black')

plt.title('Validation Set Class Distribution')

plt.xlabel('Class')

plt.ylabel('Count')

plt.xticks(range(len(class\_labels)), class\_labels, rotation=90)

plt.tight\_layout()

plt.show()

**3.Image Properties**

Inspect the properties of a few images from the dataset to understand their dimensions, color channels, etc.

import numpy as np

# Get a batch of images and labels

images, labels = next(train\_data)

# Display image properties

print(f'Image shape: {images[0].shape}')

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

print(f'Image data type: {images.dtype}')

# Display the range of pixel values

print(f'Pixel value range: [{images.min()}, {images.max()}]')

**4. Visualize Samples**

Display a few images from each class to understand the variations within classes.

def plot\_samples(generator, class\_labels):

fig, axes = plt.subplots(4, 4, figsize=(12, 12))

axes = axes.flatten()

for i in range(16):

img, label = generator[i]

ax = axes[i]

ax.imshow(img)

label\_idx = np.argmax(label)

ax.set\_title(class\_labels[label\_idx])

ax.axis('off')

plt.tight\_layout()

plt.show()

plot\_samples(train\_data, class\_labels)

**5. Check for Missing or Corrupt Images**

Ensure all images can be loaded properly and are not corrupted.

def check\_images(directory):

for root, dirs, files in os.walk(directory):

for file in files:

if file.endswith('.jpg') or file.endswith('.png'):

img\_path = os.path.join(root, file)

try:

img = cv2.imread(img\_path)

if img is None:

print(f"Image not readable: {img\_path}")

except Exception as e:

print(f"Error reading {img\_path}: {e}")

check\_images(train\_path)

check\_images(valid\_path)

**SPLIT DATASETS INTO TRAINING, VALIDATION, TEST SETS**

**Code for Splitting Datasets**

1. **Import Necessary Libraries**: Import the libraries required for data processing, image manipulation, and model training.
2. **Define Paths and Parameters**: Set the path to your dataset and the image size.
3. **Check and Resize Images**: Create a function to check if images can be read and resize them to the desired size.
4. **Set Up ImageDataGenerator**: Create data generators for training, validation, and test sets.

#libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import cv2

import tensorflow as tf

# Paths and parameters

dataset\_path = '/content/new plant diseases dataset(augmented)/New Plant Diseases Dataset(Augmented)'

train\_path = os.path.join(dataset\_path, 'train')

valid\_path = os.path.join(dataset\_path, 'valid')

test\_path = os.path.join(dataset\_path, 'test')

size = 224 # Image size to resize to

# Function to check and resize images

def check\_and\_resize\_images(directory, size):

for root, dirs, files in os.walk(directory):

for file in files:

if file.endswith('.jpg') or file.endswith('.png'):

img\_path = os.path.join(root, file)

img = cv2.imread(img\_path)

if img is None:

print(f"Image not readable: {img\_path}")

else:

img\_resized = cv2.resize(img, (size, size))

cv2.imwrite(img\_path, img\_resized)

# Check and resize images

check\_and\_resize\_images(train\_path, size)

check\_and\_resize\_images(valid\_path, size)

check\_and\_resize\_images(test\_path, size)

# ImageDataGenerator setup

data\_generator = tf.keras.preprocessing.image.ImageDataGenerator(

rotation\_range=90,

width\_shift\_range=0.0,

height\_shift\_range=0.0,

shear\_range=0.0,

zoom\_range=0.0,

horizontal\_flip=False,

vertical\_flip=False,

rescale=1/255.0,

validation\_split=0.1 # Use this for splitting training and validation sets

)

# Training data generator

train\_data = data\_generator.flow\_from\_directory(

train\_path,

batch\_size=164,

target\_size=(size, size),

subset="training",

color\_mode='rgb',

class\_mode='categorical',

shuffle=True

)

# Validation data generator from train directory

val\_data = data\_generator.flow\_from\_directory(

train\_path,

batch\_size=164,

target\_size=(size, size),

subset="validation",

color\_mode='rgb',

class\_mode='categorical',

shuffle=False

)

# Test data generator

test\_generator = tf.keras.preprocessing.image.ImageDataGenerator(

rescale=1/255.0

).flow\_from\_directory(

test\_path,

batch\_size=164,

target\_size=(size, size),

color\_mode='rgb',

class\_mode='categorical',

shuffle=False

)

# Plot images to verify

def plot\_images(generator):

x\_batch, y\_batch = next(generator)

fig, axes = plt.subplots(1, 5, figsize=(20, 20))

axes = axes.flatten()

for img, ax in zip(x\_batch, axes):

ax.imshow(img)

ax.axis('off')

plt.tight\_layout()

plt.show()

# Plot a batch of training images

plot\_images(train\_data)

**Explanation:**

1. **Importing Libraries**: Import all necessary libraries for data processing, image manipulation, and model training.
2. **Checking and Resizing Images**: A function to iterate through your dataset, check if images are readable, and resize them to the desired size (224x224).
3. **Creating ImageDataGenerators**:
   * data\_generator: This handles the image augmentation and normalization for training and validation sets.
   * train\_data: The training data generator that uses 90% of the data.
   * val\_data: The validation data generator that uses 10% of the data.
   * test\_generator: The test data generator that reads images from the test directory.

**Conclusion:**

By following these steps, you have prepared a clean and well-organized dataset that is ready for training a machine learning model to detect plant diseases. This ensures that your model training process will be efficient and effective, leading to more accurate predictions and better performance.

The plant disease detection project successfully developed a CNN model capable of accurately classifying leaf images into healthy and diseased categories for plants like apple, cherry, grape, and corn. The model showcases the potential of deep learning to aid in early disease detection, promoting efficient agricultural practices.