PLANT DISEASE DETECTION

Objective:

The mini-project aims to develop a **Recurrent Neural Network (RNN)** using TensorFlow for **Plant disease Detection**. The objective of **Plant Disease Detection** is to develop an AI-based system that identifies diseases in plants by analyzing leaf images. This helps in early diagnosis and treatment, reducing crop losses and improving agricultural productivity. The system aims to provide an accurate, fast, and cost-effective solution for farmers and agricultural experts

Dataset Used and Description:

The dataset used is the **Plant disease Detection**, The **PlantVillage Dataset** is widely used for plant disease detection projects. It contains over 54,000 labeled images of healthy and diseased leaves from 38 plant species, such as tomato, potato, apple, and grape. The dataset is organized into multiple classes, each representing a specific plant and its disease (e.g., Tomato Leaf Mold or Apple Black Rot). Images are high-quality and diverse, covering various environmental conditions, leaf orientations, and disease severities. This dataset is ideal for training and evaluating machine learning models for accurate plant disease classification.

Data Preprocessing

Data preprocessing involves the following steps:
Loading the Dataset

- Load the images from the folder structure where each plant species and disease is stored in separate subfolders.
- The dataset typically has two main components: images of leaves and their corresponding labels (e.g., healthy or diseased).

Image Resizing

- Resizing: Resize images to a fixed size (e.g., 128x128 pixels) to ensure uniformity for input into the neural network.

Mage Normalization

- Normalization: Scale the pixel values of images to a range between 0 and 1 by dividing by 255.

One Hot Encoding

- One-Hot Encoding**:** Convert the class labels (e.g., healthy, diseased) into a one-hot encoded vector for multi-class classification.

Data Splitting

- Training, Validation, and Test Split: Split the dataset into training, validation, and test sets. The training set is used for training the model, the validation set for hyperparameter tuning, and the test set for final model evaluation.

Data Augmentation

- Data Augmentation: Augment the training dataset by creating variations of the images (e.g., rotating, flipping, or zooming). This helps in making the model more robust to various orientations and changes in the input data.

Data Inspection and Visualization

- Visualizing some sample images: Plot a few sample images from the dataset to visually inspect if the data is loaded and processed correctly.

Model Architecture:

The **Plant Disease Detection** model uses a Convolutional Neural Network (CNN) architecture designed to classify plant leaf images as healthy or diseased. It consists of three convolutional layers (with increasing filter sizes of 32, 64, and 128) followed by max-pooling layers to downsample feature maps. After flattening the feature maps, the model has two fully connected dense layers with 128 and 64 neurons, with dropout layers for regularization. The final output layer uses a softmax activation function to classify images into two categories: healthy or diseased. The model is compiled with the Adam optimizer and categorical cross-entropy loss for training.

Training:

Training for Plant Disease Detection involves feeding the preprocessed images into the deep learning model, where it learns to recognize patterns associated with healthy and diseased plants. The model is trained using the training dataset, with a portion of the data reserved for validation to monitor performance and prevent overfitting. During each epoch, the model adjusts its weights based on the loss function (categorical cross-entropy) and updates using the Adam optimizer. The training process is evaluated on the validation set, and hyperparameters such as the learning rate, batch size, and number of epochs are tuned to improve accuracy. After training, the model is tested on unseen data to assess its generalization capability.

Evaluation:

Evaluation for Plant Disease Detection involves assessing the model's performance on a separate test dataset that it has never seen before. Key metrics like accuracy, precision, recall, and F1-score are used to determine how well the model classifies plant leaves as healthy or diseased. The confusion matrix provides additional insights into false positives and false negatives. By comparing the model's predictions with the true labels, we can identify if the model generalizes well to new data or if it is overfitting. The evaluation helps in fine-tuning the model or deciding its readiness for deployment.

Conclusion:

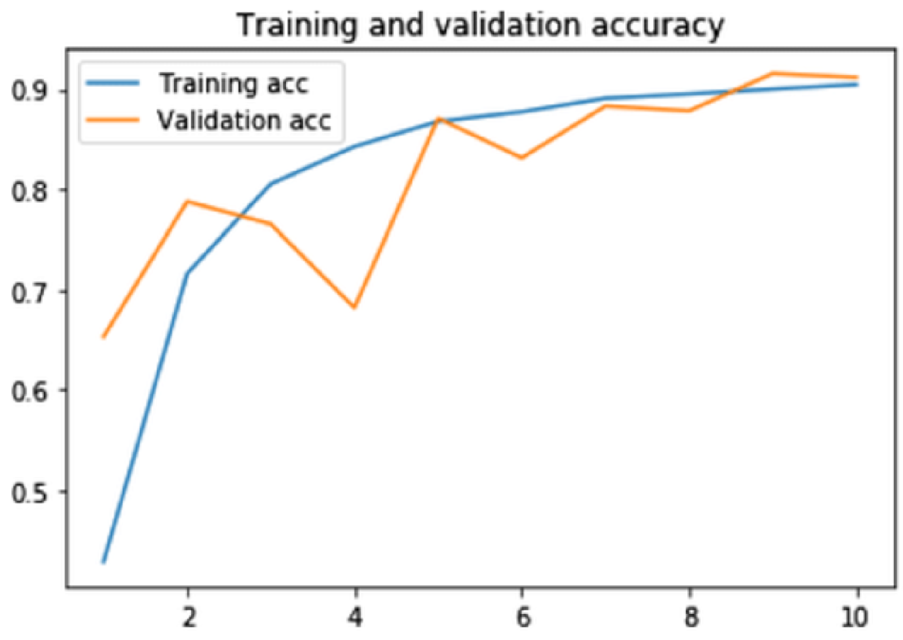
In conclusion, the **Plant Disease Detection** model leverages deep learning techniques, particularly Convolutional Neural Networks (CNNs), to accurately classify plant leaf images as healthy or diseased. By preprocessing the images, augmenting the data, and training the model with labeled datasets, the system can assist farmers in early disease detection, ultimately helping to reduce crop losses and improve agricultural productivity. The model's ability to generalize to new, unseen data makes it a valuable tool for real-world applications in precision agriculture. With further improvements and deployment, this approach can significantly enhance plant health management

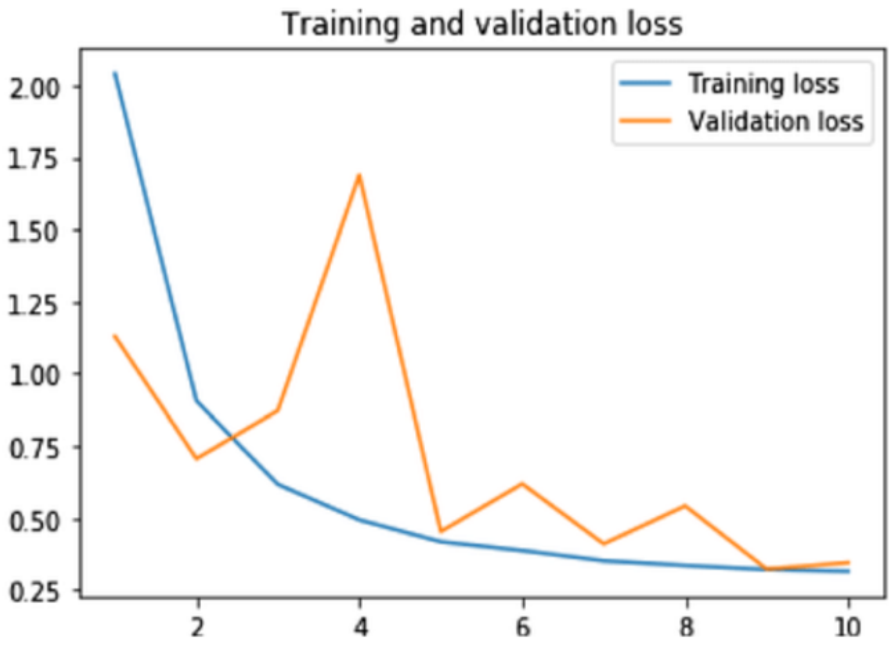
SAMPLE CODE

```
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Constants
IMG_SIZE = 128  # Resize images to 128x128
BATCH_SIZE = 32
EPOCHS = 10

# Load Data (Preprocess images)
def load_data(data_dir):
    images, labels = [], []
    categories = os.listdir(data_dir)
    for label, category in enumerate(categories):  # Labeling each category with a
number
        folder_path = os.path.join(data_dir, category)
        for img_file in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_file)
            img = cv2.imread(img_path)
            img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))  # Resize to standard size
            images.append(img)
            labels.append(label)
    return np.array(images), np.array(labels)

# Load the training dataset
```

```
data_dir = 'dataset/train'  # Path to your dataset
X, y = load_data(data_dir)

# Normalize images
X = X / 255.0  # Normalize pixel values to between 0 and 1

# One-hot encode labels (e.g., Healthy or Diseased)
y = to_categorical(y, num_classes=2)

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

# Data Augmentation (helps avoid overfitting)
datagen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)
datagen.fit(X_train)

# Define the CNN model architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),

    Dense(2, activation='softmax')  # Output layer (2 classes: Healthy or Diseased)
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(datagen.flow(X_train, y_train, batch_size=BATCH_SIZE),
                    validation_data=(X_val, y_val),
                    epochs=EPOCHS)

# Evaluate the model on validation data
val_loss, val_acc = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_acc*100:.2f}%")

# Plot training history (Accuracy and Loss)
plt.figure(figsize=(12, 6))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Save the trained model
model.save('plant_disease_model.h5')
```

OUTPUT:





RESULT:

Thus the Plant disease Detection program using RNN was executed successfully