Develop and Implement - CNN based application

PLANT DISEASE IMAGE CLASSIFIER-CNN

Problem Statement:

Plant diseases pose a significant threat to global agriculture, leading to reduced crop yields, economic losses, and increased risks to food security. The traditional methods of identifying plant diseases rely heavily on expert analysis, manual inspections, and laboratory testing, which can be time-consuming, expensive, and inaccessible to many farmers, particularly in rural areas.

Accurate and timely detection of plant diseases is crucial for minimizing these impacts. Early detection allows farmers to take targeted measures to prevent the spread of disease and minimize crop damage, thus improving agricultural productivity and sustainability.

This project aims to develop a **Convolutional Neural Network (CNN)-based model** capable of detecting and classifying plant diseases from images of leaves. By utilizing cutting-edge deep learning techniques, the proposed solution seeks to provide an accessible, automated, and highly accurate diagnostic tool for farmers and agricultural stakeholders. The ultimate goal is to empower farmers to take proactive measures, reduce losses, and ensure food security through scalable and efficient disease detection.

Dataset Used

Dataset Name: PlantVillage Dataset

- **Source**: Open-access repository containing labeled images of healthy and diseased leaves across multiple plant species.
- **Dataset Features**:
  - ~54,000 labeled images.
  - Categories: Healthy and diseased leaves for 38 plant types (e.g., tomato, potato, grape).
  - Format: RGB images.
  - Preprocessing: Images resized to a uniform size (e.g., 224x224 pixels).

Implementation

1. Preprocessing

- Resized images to a consistent dimension (e.g., 224x224 or 128x128).
- Augmented data to increase dataset diversity (rotation, flipping, zooming, etc.).
- Normalized pixel values to [0,1] range for faster convergence.

2. CNN Architecture

- Input Layer: Accepts preprocessed leaf images.

- Convolutional Layers: Extract spatial features using filters (e.g., 3x3 kernels).
- Pooling Layers: Applied max-pooling for dimensionality reduction.
- Dropout: Added dropout layers to prevent overfitting.
- Dense Layers: Fully connected layers for final classification.
- Output Layer: Softmax activation for multiclass classification.

3. Model Training

- Framework: TensorFlow/Keras.
- Optimizer: Adam optimizer.
- Loss Function: Categorical Crossentropy.
- Metrics: Accuracy and Loss.
- Epochs: Trained over 25–50 epochs (adjustable).
- Validation Split: 20% of the data reserved for validation.

4. Testing and Evaluation

- Test the model on unseen images.
- Metrics: Accuracy, precision, recall, F1-score.

CODE:

```
# Set seeds for reproducibility
import random
random.seed(0)
import numpy as np
np.random.seed(0)
import tensorflow as tf
tf.random.set_seed(0)


import os
import json
from zipfile import ZipFile
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models


!pip install kaggle


kaggle_credentails = json.load(open("kaggle.json"))

# setup Kaggle API key as environment variables
os.environ['KAGGLE_USERNAME'] = kaggle_credentails["username"]
os.environ['KAGGLE_KEY'] = kaggle_credentails["key"]
```

```
!kaggle datasets download -d abdallahalidev/plantvillage-dataset

!ls

# Unzip the downloaded dataset
with ZipFile("plantvillage-dataset.zip", 'r') as zip_ref:
    zip_ref.extractall()

print(os.listdir("plantvillage dataset"))
print(len(os.listdir("plantvillage dataset/segmented")))
print(os.listdir("plantvillage dataset/segmented")[:5])
print(len(os.listdir("plantvillage dataset/color")))
print(os.listdir("plantvillage dataset/color")[:5])
print(len(os.listdir("plantvillage dataset/grayscale")))
print(os.listdir("plantvillage dataset/grayscale")[:5])
```

['color', 'grayscale', 'segmented']
38
['Apple___Black_rot', 'Corn_(maize)___Northern_Leaf_Blight','Cherry_(including_sour)___Powdery_mildew', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Grape___Black_rot']
38
['Apple___Black_rot', 'Corn_(maize)___Northern_Leaf_Blight','Cherry_(including_sour)___Powdery_mildew', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Grape___Black_rot']
38
['Apple___Black_rot', 'Corn_(maize)___Northern_Leaf_Blight','Cherry_(including_sour)___Powdery_mildew', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Grape___Black_rot']

```
print(len(os.listdir("plantvillage dataset/color/Grape___healthy")))
print(os.listdir("plantvillage dataset/color/Grape___healthy")[:5])
```

423
['69a50f71-f8ee-4280-ac3e-c1d68e950c2e___Mt.N.V_HL 6165.JPG', '5e8fe4a1-4b7c-44d6-bee6-2a97360a09f1___Mt.N.V_HL 9044.JPG', '51665d49-a91d-4790-a705-425d3bd9e62a___Mt.N.V_HL 6172.JPG', '22b7199f-581e-4330-b14f-4cb7b382340a___Mt.N.V_HL 6115.JPG', 'f8f6d1ee-deb5-4b3c-a92e-2583e2d6d590___Mt.N.V_HL 6122.JPG']

```
# Dataset Path
base_dir = 'plantvillage dataset/color'

image_path = '/content/plantvillage dataset/color/Apple___Cedar_apple_rust/025b2b9a-0ec4-4132-96ac-7f2832d0db4a___FREC_C.Rust 3655.JPG'

# Read the image
img = mpimg.imread(image_path)
print(img.shape)
```

```
# Display the image
plt.imshow(img)
plt.axis('off')  # Turn off axis numbers
plt.show()
```

```
image_path = '/content/plantvillage dataset/color/Apple___Cedar_apple_rust/025b2b9a-0ec4-4132-96ac-7f2832d0db4a___FREC_C.Rust 3655.JPG'
# Read the image
img = mpimg.imread(image_path)
print(img)

[[[179 175 176]
  [181 177 178]
  [184 180 181]
  ...
  [115 112 105]
  [108 105  98]
  [101  98  91]]

 [[176 172 173]
  [177 173 174]
  [178 174 175]
  ...
  [113 110 103]
  [111 108 101]
  [109 106  99]]
```

```
 [[180 176 177]
  [180 176 177]
  [180 176 177]
  ...
  [108 105  98]
  [111 108 101]
  [114 111 104]]

 ...

 [[137 128 119]
  [131 122 113]
  [125 116 107]
  ...
  [ 74  65  48]
  [ 74  65  48]
  [ 73  64  47]]

 [[136 127 118]
  [132 123 114]
  [128 119 110]
  ...
  [ 77  69  50]
  [ 75  67  48]
  [ 75  67  48]]

 [[133 124 115]
  [133 124 115]
  [132 123 114]
  ...
  [ 81  73  54]
  [ 80  72  53]
  [ 79  71  52]]]

# Image Parameters
img_size = 224
batch_size = 32

# Image Data Generators
data_gen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2  # Use 20% of data for validation
)

# Train Generator
train_generator = data_gen.flow_from_directory(
    base_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    subset='training',
```

```
    class_mode='categorical'
)
```

Found 43456 images belonging to 38 classes.

```
# Validation Generator
validation_generator = data_gen.flow_from_directory(
    base_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    subset='validation',
    class_mode='categorical'
)
```

Found 10849 images belonging to 38 classes.

```
# Model Definition
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))

# model summary

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
flatten (Flatten)	(None, 186624)	0
dense (Dense)	(None, 256)	47,776,000
dense_1 (Dense)	(None, 38)	9,766

Total params: 47,805,158 (182.36 MB)
Trainable params: 47,805,158 (182.36 MB)
Non-trainable params: 0 (0.00 B)

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

```
# Training the Model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,  # Number of steps per epoch
    epochs=5,  # Number of epochs
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size  # Validation steps
)
```

Epoch 1/5
1358/1358 [==============================] - 108s 76ms/step - loss: 0.9791 - accuracy: 0.7328 - val_loss: 0.4846 - val_accuracy: 0.8465
Epoch 2/5
1358/1358 [==============================] - 104s 77ms/step - loss: 0.2812 - accuracy: 0.9110 - val_loss: 0.4477 - val_accuracy: 0.8655
Epoch 3/5
1358/1358 [==============================] - 106s 78ms/step - loss: 0.1362 - accuracy: 0.9553 - val_loss: 0.4321 - val_accuracy: 0.8863
Epoch 4/5
1358/1358 [==============================] - 103s 76ms/step - loss: 0.0891 - accuracy: 0.9708 - val_loss: 0.5433 - val_accuracy: 0.8715
Epoch 5/5
1358/1358 [==============================] - 109s 81ms/step - loss: 0.0761 - accuracy: 0.9760 - val_loss: 0.5091 - val_accuracy: 0.8828

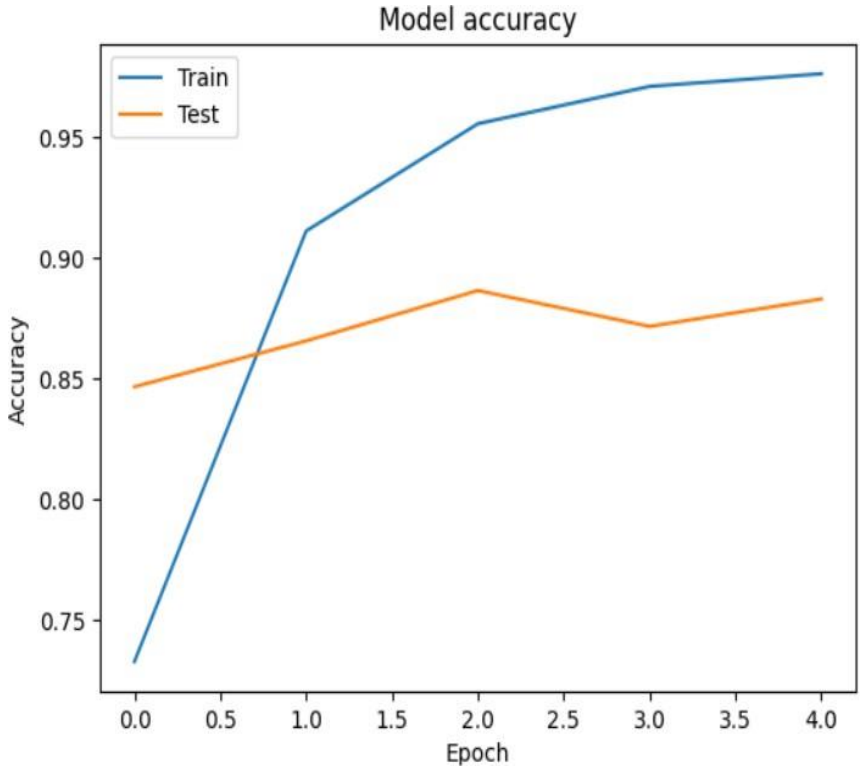
```
# Model Evaluation
print("Evaluating model...")
val_loss, val_accuracy = model.evaluate(validation_generator,
steps=validation_generator.samples // batch_size)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

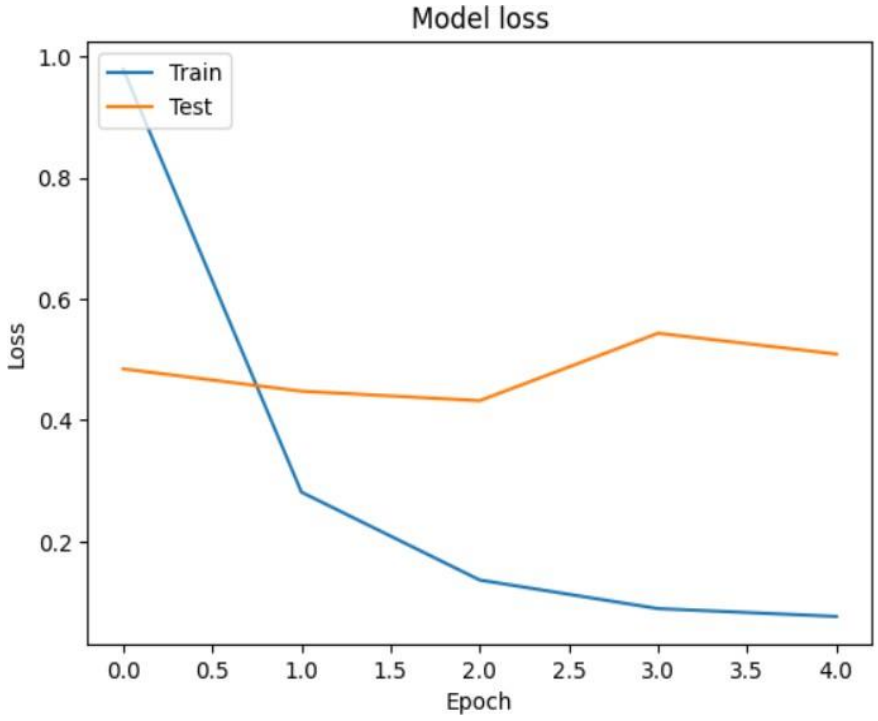
Evaluating model...
339/339 [==============================] - 19s 57ms/step - loss: 0.5091 - accuracy: 0.8828
Validation Accuracy: 88.28%

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
```

```
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```





Function to Load and Preprocess the Image using Pillow

```
def load_and_preprocess_image(image_path, target_size=(224, 224)):
    # Load the image
    img = Image.open(image_path)
    # Resize the image
    img = img.resize(target_size)
    # Convert the image to a numpy array
    img_array = np.array(img)
    # Add batch dimension
    img_array = np.expand_dims(img_array, axis=0)
    # Scale the image values to [0, 1]
    img_array = img_array.astype('float32') / 255.
    return img_array

# Function to Predict the Class of an Image
def predict_image_class(model, image_path, class_indices):
    preprocessed_img = load_and_preprocess_image(image_path)
    predictions = model.predict(preprocessed_img)
    predicted_class_index = np.argmax(predictions, axis=1)[0]
    predicted_class_name = class_indices[predicted_class_index]
    return predicted_class_name



# Create a mapping from class indices to class names
class_indices = {v: k for k, v in train_generator.class_indices.items()}


# saving the class names as json file
json.dump(class_indices, open('class_indices.json', 'w'))

# Example Usage
image_path = '/content/test_apple_black_rot.JPG'
#image_path = '/content/test_blueberry_healthy.jpg'
#image_path = '/content/test_potato_early_blight.jpg'
predicted_class_name = predict_image_class(model, image_path, class_indices)

# Output the result
print("Predicted Class Name:", predicted_class_name)
```

OUTPUT:

1/1 [==============================] - 0s 266ms/step
Predicted Class Name: Apple___Black_rot