CROP DISEASE DETECTION

Deep learning based object detection framework.

Introduction

Crop diseases are a major threat to food security, but their rapid identification remains difficult in many parts of the world due to the lack of the necessary infrastructure. But the recent developments in the terms of Computer Vision lead me into thinking that why can't we develop a deep learning model to test whether a plant is healthy or it has any disease. Hence I built a deep learning Model Using CNN to classify the plants using their leaves. There has been Research Papers previously on this topic, I have referred to those papers and tried to enhance my model to achieve higher accuracy.

Experimental Setup

Dataset: I have used a public Dataset from Kaggle for training the model. Dataset has 20638 images classified into 15 categories.

Dataset Link: https://www.kaggle.com/datasets/arjuntejaswi/plant-village.

Importing Dataset:

```
!kaggle datasets download -d arjuntejaswi/plant-village
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kagg
Downloading plant-village.zip to /content
 99% 324M/329M [00:10<00:00, 38.4MB/s]
100% 329M/329M [00:10<00:00, 32.9MB/s]
```

Creating Tensorflow pipeline for the dataset:

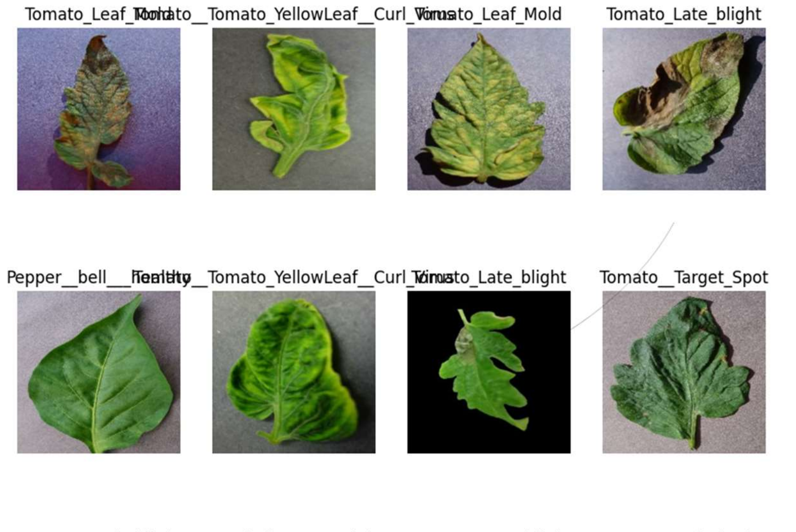
```
import tensorflow as tf
from tensorflow.keras import layers,models
import matplotlib.pyplot as plt
impor (module) imblearn
from imblearn.over_sampling import SMOTE
```

```
[6] Batch_Size=32
    image_size=256
    df=tf.keras.preprocessing.image_dataset_from_directory(
        "PlantVillage",
        shuffle=True,
        image_size=(image_size,image_size),
        batch_size=Batch_Size,
    )
```

```
Found 20638 files belonging to 15 classes.
```

Checking Dataset:

```python
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in df.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



Hyperparameters Used: I have Used Batch_Size= ’32’ for the dataset and Epochs= ’40’.

Preprocessing: I have used standard Image_size=256 x256 pixels and preprocessing Steps Included rescaling and reshaping of the given images.

```python
resizing=tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(256,256),
    layers.experimental.preprocessing.Rescaling(1.0/255)
])
```

Hardware and Software Equipments Used:

I have Used T4 GPU provided by google colab and the frameworks used are Tensorflow, Libraries used are numpy,pandas, Sequential,Models,Flatten,Conv2D,Maxpooling2D, etc.....

Training the Model:

Train_Test_Split:

From the total Dataset I took 80% of the images for training and 10% for the validation and 10%for the testing phase.

```
train_size=int(0.8*len(df))
train_ds=df.take(train_size)
vali_size=int(0.1*len(df))
test_size=int(0.1*len(df))
test_ds=df.skip(train_size)
vali_ds=test_ds.take(vali_size)
test_ds=test_ds.skip(vali_size)
```

Data Augmentation:

Data augmentation is a technique commonly used in machine learning and deep learning to increase the size and diversity of a training dataset without collecting new data.

Since Our data is in the form of Images, I have used Random_flip and Random_rotation techniques for the data augmentation such that model Recognizes when the picture is given in reverse direction or in any flipped direction.

```
augmentation=tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

Model Architecture:

In the Model Architecture I chose to have 6 Layers of Convolution each with 64 filters and size of ‘3x3’, for each convolution layer i have used ‘relu’ as activation function.

For each Convolution Layer I Have Used Maxpooling Layer using size of ‘2x2’.

Then I have Used one Dense Hidden Layer with 64 Nodes and activation Function as ‘Relu’, and at last I have output Layer with 15 neurons and activation function as ‘Softmax’.

```
input_shape=(Batch_Size,image_size,image_size,3)
model = models.Sequential([
    resizing,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (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(15, activation='softmax'),
])


model.build(input_shape)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential_1 (Sequential)   (32, 256, 256, 3)         0         
                                                                 
 conv2d (Conv2D)             (32, 254, 254, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928     
                                                                 
 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928     
                                                                 
 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928     
                                                                 
 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928     
                                                                 
 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (32, 256)                 0         
                                                                 
 dense (Dense)               (32, 64)                  16448     
                                                                 
 dense_1 (Dense)             (32, 15)                  975       
                                                                 
=================================================================
Total params: 184,527
Trainable params: 184,527
Non-trainable params: 0
_________________________________________________________________
```

Training Parameters:

Metrics: I have Used Accuracy as the metric as it is a straightforward metric to understand and interpret. It's easy to communicate to non-technical stakeholders, making it a popular choice for presenting results and performance.

Loss_Function: My Choice of Loss_Function was SparseCategoricalCrossEntropy Since it is a loss function commonly used in multi-class classification tasks, where the target labels are integers instead of one-hot encoded vectors. It is particularly useful when dealing with large datasets with a significant number of classes, as it helps in reducing memory usage and computational complexity.

Optimizer: My Choice of optimizer was Adam Since It's main aim is to adapt the learning rates of individual parameters based on their past gradients and momentum, allowing it to converge quickly and efficiently in many cases.

```
model.compile(
    optimizer='adam',
    loss='SparseCategoricalCrossentropy',
    metrics=['accuracy']
)
```

Model.Fit:

```
history=model.fit(train_ds,
                  batch_size=Batch_Size,
                  validation_data=vali_ds,
                  verbose=1,
                  epochs=40,)
```

Training Epochs:

```
516/516 [==============================] - 197s 382ms/step - loss: 0.2562 - accuracy: 0.9118 - val_loss: 0.4716 - val_accuracy: 0.8589
Epoch 14/40
516/516 [==============================] - 195s 377ms/step - loss: 0.2490 - accuracy: 0.9133 - val_loss: 0.2846 - val_accuracy: 0.9111
Epoch 15/40
516/516 [==============================] - 198s 383ms/step - loss: 0.2383 - accuracy: 0.9164 - val_loss: 0.5485 - val_accuracy: 0.8403
Epoch 16/40
516/516 [==============================] - 197s 380ms/step - loss: 0.2152 - accuracy: 0.9248 - val_loss: 0.3702 - val_accuracy: 0.8809
Epoch 17/40
516/516 [==============================] - 198s 384ms/step - loss: 0.2022 - accuracy: 0.9282 - val_loss: 0.2276 - val_accuracy: 0.9277
Epoch 18/40
516/516 [==============================] - 197s 381ms/step - loss: 0.1938 - accuracy: 0.9333 - val_loss: 0.2043 - val_accuracy: 0.9351
Epoch 19/40
516/516 [==============================] - 196s 379ms/step - loss: 0.1854 - accuracy: 0.9351 - val_loss: 0.2031 - val_accuracy: 0.9360
Epoch 20/40
516/516 [==============================] - 194s 375ms/step - loss: 0.1857 - accuracy: 0.9375 - val_loss: 0.3048 - val_accuracy: 0.8921
Epoch 21/40
516/516 [==============================] - 196s 380ms/step - loss: 0.1706 - accuracy: 0.9430 - val_loss: 0.2079 - val_accuracy: 0.9268
Epoch 22/40
516/516 [==============================] - 197s 382ms/step - loss: 0.1705 - accuracy: 0.9408 - val_loss: 0.2792 - val_accuracy: 0.9062
Epoch 23/40
516/516 [==============================] - 194s 376ms/step - loss: 0.1749 - accuracy: 0.9386 - val_loss: 0.2548 - val_accuracy: 0.9165
Epoch 24/40
516/516 [==============================] - 194s 375ms/step - loss: 0.1597 - accuracy: 0.9436 - val_loss: 0.1894 - val_accuracy: 0.9360
Epoch 25/40
516/516 [==============================] - 202s 392ms/step - loss: 0.1635 - accuracy: 0.9442 - val_loss: 0.1658 - val_accuracy: 0.9463
Epoch 26/40
516/516 [==============================] - 196s 380ms/step - loss: 0.1563 - accuracy: 0.9483 - val_loss: 0.1669 - val_accuracy: 0.9419
Epoch 27/40
516/516 [==============================] - 203s 393ms/step - loss: 0.1499 - accuracy: 0.9496 - val_loss: 0.2263 - val_accuracy: 0.9346
Epoch 28/40
516/516 [==============================] - 194s 375ms/step - loss: 0.1387 - accuracy: 0.9509 - val_loss: 0.1958 - val_accuracy: 0.9312
Epoch 29/40
516/516 [==============================] - 195s 378ms/step - loss: 0.1431 - accuracy: 0.9508 - val_loss: 0.2506 - val_accuracy: 0.9194
Epoch 30/40
516/516 [==============================] - 196s 378ms/step - loss: 0.1256 - accuracy: 0.9575 - val_loss: 0.2557 - val_accuracy: 0.9199
Epoch 31/40
516/516 [==============================] - 194s 376ms/step - loss: 0.1319 - accuracy: 0.9523 - val_loss: 0.2841 - val_accuracy: 0.9121
Epoch 32/40
516/516 [==============================] - 192s 372ms/step - loss: 0.1456 - accuracy: 0.9500 - val_loss: 0.3057 - val_accuracy: 0.9121
Epoch 33/40
516/516 [==============================] - 194s 375ms/step - loss: 0.1147 - accuracy: 0.9601 - val_loss: 0.1795 - val_accuracy: 0.9463
Epoch 34/40
516/516 [==============================] - 191s 370ms/step - loss: 0.1288 - accuracy: 0.9550 - val_loss: 0.2756 - val_accuracy: 0.9097
Epoch 35/40
516/516 [==============================] - 192s 372ms/step - loss: 0.1203 - accuracy: 0.9586 - val_loss: 0.1602 - val_accuracy: 0.9512
Epoch 36/40
516/516 [==============================] - 191s 369ms/step - loss: 0.1186 - accuracy: 0.9599 - val_loss: 0.1289 - val_accuracy: 0.9590
Epoch 37/40
516/516 [==============================] - 198s 383ms/step - loss: 0.1169 - accuracy: 0.9595 - val_loss: 0.1558 - val_accuracy: 0.9487
Epoch 38/40
516/516 [==============================] - 197s 382ms/step - loss: 0.1186 - accuracy: 0.9589 - val_loss: 0.1893 - val_accuracy: 0.9390
Epoch 39/40
516/516 [==============================] - 190s 368ms/step - loss: 0.1089 - accuracy: 0.9633 - val_loss: 0.1507 - val_accuracy: 0.9507
Epoch 40/40
516/516 [==============================] - 191s 370ms/step - loss: 0.1156 - accuracy: 0.9606 - val_loss: 0.1899 - val_accuracy: 0.9414
```

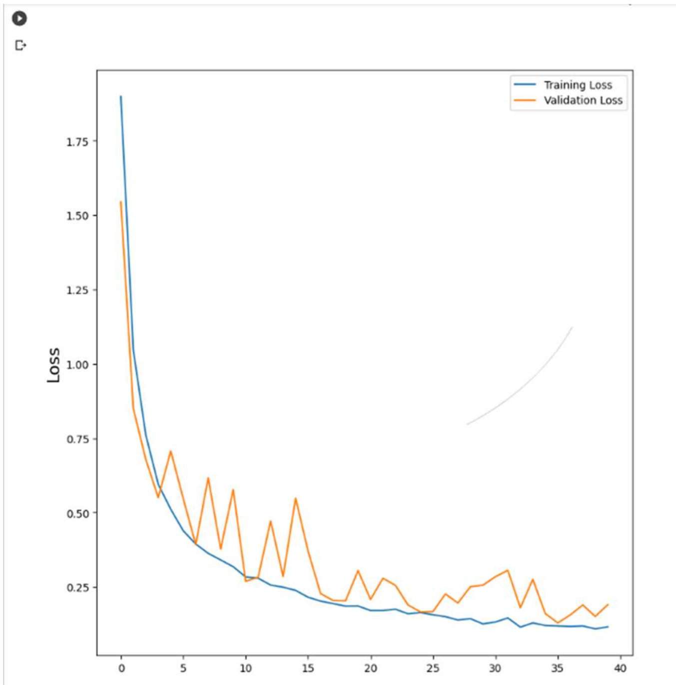
Results:

Test_Accuracy: I have achieved an approximate test accuracy of 93%.

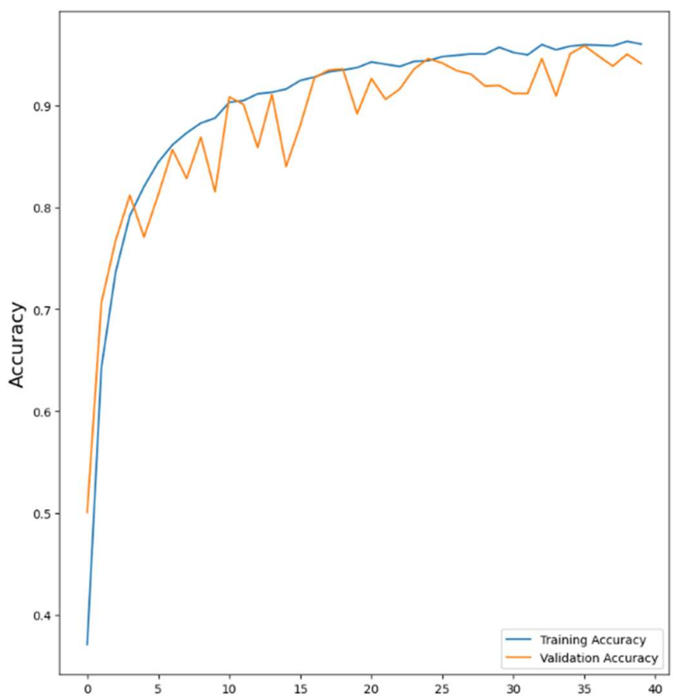
```
[ ] model.evaluate(test_ds)

65/65 [==============================] - 19s 35ms/step - loss: 0.2133 - accuracy: 0.9321
[0.21325252950191498, 0.9321463108062744]
```

Training Loss vs Validation Loss Curve:



Training Accuracy vs Validation Accuracy:



Manual Testing:

```
import cv2
img1=cv2.imread('/content/PlantVillage/Potato___healthy/07dfb451-4378-49d1-b699-33a5fc49ff07___RS_HL 5399.JPG')
img1.shape
```

```
(256, 256, 3)
```

```
import numpy as np
img1=np.expand_dims(img1,axis=0)
predicted=model.predict(img1)
```

```
1/1 [==============================] - 0s 25ms/step
```

```
predicted
```

```
array([[2.2975782e-05, 9.9990416e-01, 1.1414668e-06, 2.6586786e-06,
        4.7464178e-05, 2.2900604e-08, 9.8815099e-07, 1.8516944e-07,
        7.5865055e-14, 8.3385459e-08, 9.2509342e-09, 2.0324494e-05,
        4.6804347e-09, 5.4100700e-12, 1.1221420e-08]], dtype=float32)
```

```
from tensorflow.keras.models import load_model
from google.colab import files
model.save('plant_disease_model8.h5')
```

Conclusion:

"In conclusion, I successfully developed a deep learning model for plant disease detection. The model demonstrated promising results, achieving high accuracy on the test dataset. Automated disease detection has significant potential to benefit farmers and improve agricultural practices by enabling early detection and intervention. While the model performed well, there is room for future improvements, including fine-tuning hyperparameters and Handling Class Imbalances In a Good Manner. My work contributes to the field of plant health and lays the foundation for further advancements in automated disease detection in agriculture."

References:

- A comprehensive review on detection of plant disease using machine learning and deep learning approaches.
  Link:https://www.sciencedirect.com/science/article/pii/S2665917422000757
- Construction of deep learning-based disease detection model in plants.
  Link:https://www.nature.com/articles/s41598-023-34549-2
- An advanced deep learning models-based plant disease detection: A review of recent research
  Link:https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10070872/
- An Overview of The Research on Plant Leaves Disease detection using Image Processing Techniques.
  Link:https://www.researchgate.net/publication/314436486_An_Overview_of_the_Research_on_Plant_Leaves_Disease_detection_using_Image_Processing_Techniques

Thank You