Importing Libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Data Preprocessing

Training Image Preprocessing

```
training_set= tf.keras.utils.image_dataset_from_directory(
    r'/content/drive/MyDrive/train',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
)
```

```
Found 70295 files belonging to 38 classes.
```

Validation Image Preprocessing

```
validation_set= tf.keras.utils.image_dataset_from_directory(
    r'/content/drive/MyDrive/valid',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
)
```

```
Found 17572 files belonging to 38 classes.
```

```
training_set
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 128, 128, 3), dtype=tf.float32, name=None),
TensorSpec(shape=(None, 38), dtype=tf.float32, name=None))>
```

```
for x,y in training_set:
    print(x,x.shape)
    print(y,y.shape)
    break
```

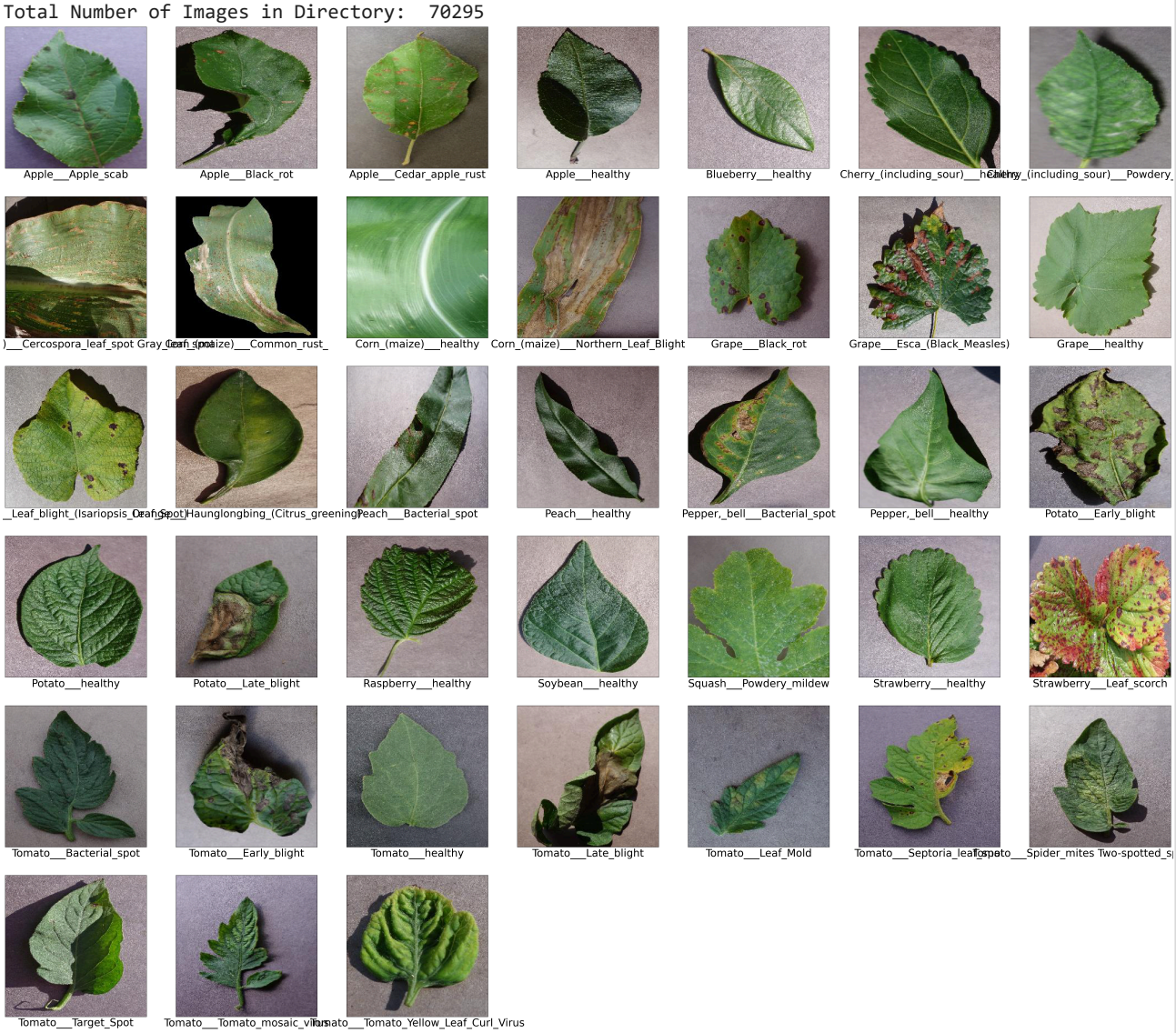
```
  -                    -
  ...
  [128.5  126.5  139.5 ]
  [130.75 128.75 141.75]
  [143.25 141.25 154.25]]

 [[153.75 151.75 162.75]
  [159.75 157.75 168.75]
  [149.5  147.5  158.5 ]
  ...
  [131.75 129.75 142.75]
  [132.75 130.75 143.75]
  [136.   134.   147.  ]]

 ...

 [[191.5  194.5  203.5 ]
  [187.   190.   199.  ]
  [189.75 192.75 201.75]
  ...
  [162.5  160.5  174.5 ]
  [164.   162.   176.  ]
  [161.5  159.5  173.5 ]]

 [[187.5  190.5  199.5 ]
  [184.25 187.25 196.25]
  [187.25 190.25 199.25]
  ...
  [163.25 161.25 175.25]
  [162.75 160.75 174.75]
  [166.5  164.5  178.5 ]]

 [[180.75 183.75 192.75]
  [186.75 189.75 198.75]
  [183.75 186.75 195.75]
  ...
  [167.   165.   179.  ]
  [171.   169.   183.  ]
  [174.75 172.75 186.75]]]


[[[118.75 110.75 107.75]
  [123.   115.   112.  ]
  [125.   117.   114.  ]
  ...
  [181.5  174.5  182.5 ]
  [184.   176.5  186.  ]
  [184.75 177.   187.25]]

 [[122.25 114.25 111.25]
  [124.75 116.75 113.75]
  [124.75 116.75 113.75]
  ...
  [175.5  168.5  175.5 ]
  [185.25 178.5  184.75]
  [181.25 174.5  181.  ]]

 [[124.5  116.5  113.5 ]
  [121.5  113.5  110.5 ]
  [123.75 115.75 112.75]
```

```
from PIL import Image
import os
import matplotlib.pyplot as plt
```

```
train_dir = r'/content/drive/MyDrive/train'
Diseases_classes = os.listdir(train_dir)
```

```
plt.figure(figsize=(60,60), dpi=200)
cnt = 0
plant_names = []
tot_images = 0

for i in Diseases_classes:
    cnt += 1
    plant_names.append(i)
    plt.subplot(7,7,cnt)

    image_path = os.listdir(train_dir + "/" + i)
    print("The Number of Images in " +i+ ":", len(image_path), end= " ")
    tot_images += len(image_path)

    img_show = plt.imread(train_dir + "/" + i + "/" + image_path[0])

    plt.imshow(img_show)
    plt.xlabel(i,fontsize=30)
    plt.xticks([])
    plt.yticks([])


print("\nTotal Number of Images in Directory: ", tot_images)
```

```
The Number of Images in Apple___Apple_scab: 2016 The Number of Images in Apple___Black_rot: 1987 The Numb
Total Number of Images in Directory:  70295
```



To avoid overshooting Loss Function:

1.Choose small learning rate default 0.001 here we have taken 0.0001
2.There may be chance of underfitting so increase number of neuron
3.Add more Convolutional Layer to extract more feature from images there may be possibilty that model unable to capture relevant feature or model is confusing due to lack of feature so feed with more feature

```
from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Flatten,Dropout
from tensorflow.keras.models import Sequential
```

Building Model

```
model=Sequential()
```

Building Convolutional Layer

```
model.add(Conv2D(filters=32,kernel_size=3,padding='same',activation='relu',input_shape=[128,128,3]))
model.add(Conv2D(filters=32,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
C:\Users\Dhanush\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning
  super().__init__(
```

```
model.add(Conv2D(filters=64,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=64,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
model.add(Conv2D(filters=128,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=128,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
model.add(Conv2D(filters=256,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=256,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
model.add(Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=512,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
model.add(Dropout(0.25))
```

```
model.add(Flatten())
```

```
model.add(Dense(units=1024,activation='relu'))
```

```
model.add(tf.keras.layers.Dropout(0.4)) #To avoid overfitting
```

```
model.add(Dense(units=38,activation='softmax'))
```

Compiling Model

```
model.compile(optimizer=tf.keras.optimizers.Adam(
    learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 128, 128, 32)	896
conv2d_11 (Conv2D)	(None, 126, 126, 32)	9,248
max_pooling2d_5 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_12 (Conv2D)	(None, 63, 63, 64)	18,496
conv2d_13 (Conv2D)	(None, 61, 61, 64)	36,928
max_pooling2d_6 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_14 (Conv2D)	(None, 30, 30, 128)	73,856
conv2d_15 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_7 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_16 (Conv2D)	(None, 14, 14, 256)	295,168
conv2d_17 (Conv2D)	(None, 12, 12, 256)	590,080
max_pooling2d_8 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_18 (Conv2D)	(None, 6, 6, 512)	1,180,160
conv2d_19 (Conv2D)	(None, 4, 4, 512)	2,359,808
max_pooling2d_9 (MaxPooling2D)	(None, 2, 2, 512)	0
dropout_2 (Dropout)	(None, 2, 2, 512)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_2 (Dense)	(None, 1024)	2,098,176
dropout_3 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 38)	38,950

Training model

```
training_history=model.fit(x=training_set,validation_data=validation_set,epochs=10)
```

```
Epoch 1/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 1817s 825ms/step - accuracy: 0.3924 - loss: 2.1580 - val_accuracy: 0.8433
Epoch 2/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 1675s 763ms/step - accuracy: 0.8327 - loss: 0.5353 - val_accuracy: 0.9016
Epoch 3/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 2390s 1s/step - accuracy: 0.9054 - loss: 0.3031 - val_accuracy: 0.9392 - v
Epoch 4/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 3476s 2s/step - accuracy: 0.9329 - loss: 0.2059 - val_accuracy: 0.9534 - v
Epoch 5/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 2955s 1s/step - accuracy: 0.9522 - loss: 0.1448 - val_accuracy: 0.9466 - v
Epoch 6/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 3117s 1s/step - accuracy: 0.9603 - loss: 0.1246 - val_accuracy: 0.9537 - v
Epoch 7/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 3223s 1s/step - accuracy: 0.9722 - loss: 0.0872 - val_accuracy: 0.9603 - v
Epoch 8/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 3633s 2s/step - accuracy: 0.9759 - loss: 0.0737 - val_accuracy: 0.9591 - v
Epoch 9/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 3440s 2s/step - accuracy: 0.9781 - loss: 0.0695 - val_accuracy: 0.9668 - v
Epoch 10/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 3253s 1s/step - accuracy: 0.9801 - loss: 0.0598 - val_accuracy: 0.9634 - v
```

Evaluating Model

```
train_loss, train_acc = model.evaluate(training_set)
print('Training accuracy:', train_acc)
```

```
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 990s 451ms/step - accuracy: 0.9889 - loss: 0.0352
Training accuracy: 0.9897148013114929
```

```
val_loss, val_acc = model.evaluate(validation_set)
print('Validation accuracy:', val_acc)
```

```
550/550 ━━━━━━━━━━━━━━━━━━━━ 280s 508ms/step - accuracy: 0.9639 - loss: 0.1144
Validation accuracy: 0.963350772857666
```

Saving Model

```
# Save the model with a valid filepath extension (.h5)
model.save('C:/Users/Dhanush/trained_plant_disease_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(moc
```

```
training_history.history #Return Dictionary of history
```

```
{'accuracy': [0.5956611633300781,
  0.8547691702842712,
  0.9128530025482178,
  0.9373354911804199,
  0.9533963799476624,
  0.9642080068588257,
  0.9716765284538269,
  0.9757024049758911,
  0.978632926940918,
  0.9812788963317871],
 'loss': [1.3765381574630737,
  0.4630456864833832,
  0.2732311189174652,
  0.19423291087150574,
  0.1412355750799179,
  0.11078231781721115,
  0.08759018033742905,
  0.0742606595158577,
  0.06640901416540146,
  0.05726907402276993],
 'val_accuracy': [0.8433303236961365,
  0.9016048312187195,
  0.9392215013504028,
  0.9534486532211304,
  0.9466196298599243,
  0.9537332057952881,
  0.9603345990180969,
  0.9591395258903503,
  0.966765284538269,
  0.963350772857666],
 'val_loss': [0.4923146665096283,
  0.30608975887298584,
  0.1950557827949524,
  0.1455708146095276,
  0.1590263545513153,
  0.15049678087234497,
  0.1253540813922882,
  0.14174355566501617,
  0.10798969864845276,
  0.11913156509399414]}
```

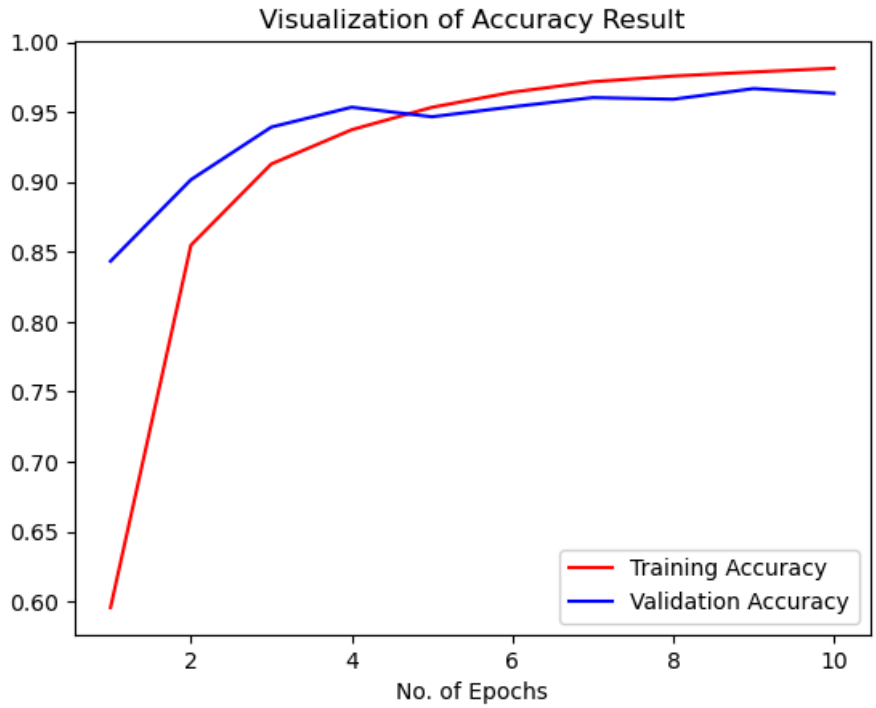
```
import json
with open('training_hist.json','w') as f:
  json.dump(training_history.history,f)
```

```
print(training_history.history.keys())
```

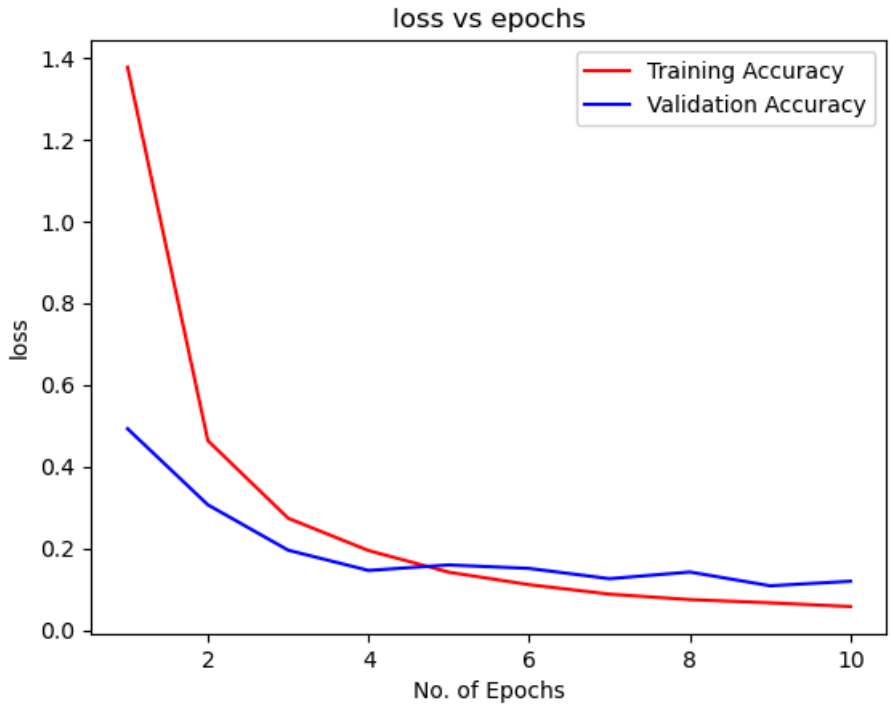
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

Accuracy Visualization

```
epochs = [i for i in range(1,11)]
plt.plot(epochs,training_history.history['accuracy'],color='red',label='Training Accuracy')
plt.plot(epochs,training_history.history['val_accuracy'],color='blue',label='Validation Accuracy')
plt.xlabel('No. of Epochs')
plt.title('Visualization of Accuracy Result')
plt.legend()
plt.show()
```




```
plt.plot(epochs,training_history.history['loss'],color='red',label='Training Accuracy')
plt.plot(epochs,training_history.history['val_loss'],color='blue',label='Validation Accuracy')
plt.xlabel('No. of Epochs')
plt.ylabel('loss')
plt.title('loss vs epochs')
plt.legend()
plt.show()
```



Other metrics for model evaluation

```
class_name = validation_set.class_names
```

```
print(validation_set.class_names)
```

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy', 'Blueberry___h
```

```
test_set = tf.keras.utils.image_dataset_from_directory(
r'/content/drive/MyDrive/valid',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=1,
    image_size=(128, 128),
    shuffle=False,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

```
Found 17572 files belonging to 38 classes.
```

```
y_pred = model.predict(test_set)
predicted_categories = tf.argmax(y_pred, axis=1)
```

```
17572/17572 ━━━━━━━━━━━━━━━━━━━━ 563s 32ms/step
```

```
true_categories = tf.concat([y for x, y in test_set], axis=0)
Y_true = tf.argmax(true_categories, axis=1)
```

```
Y_true
```

```
<tf.Tensor: shape=(17572,), dtype=int64, numpy=array([ 0,  0,  0, ..., 37, 37, 37], dtype=int64)>
```

```
predicted_categories
```

```
<tf.Tensor: shape=(17572,), dtype=int64, numpy=array([ 0,  0,  0, ..., 37, 37, 37], dtype=int64)>
```

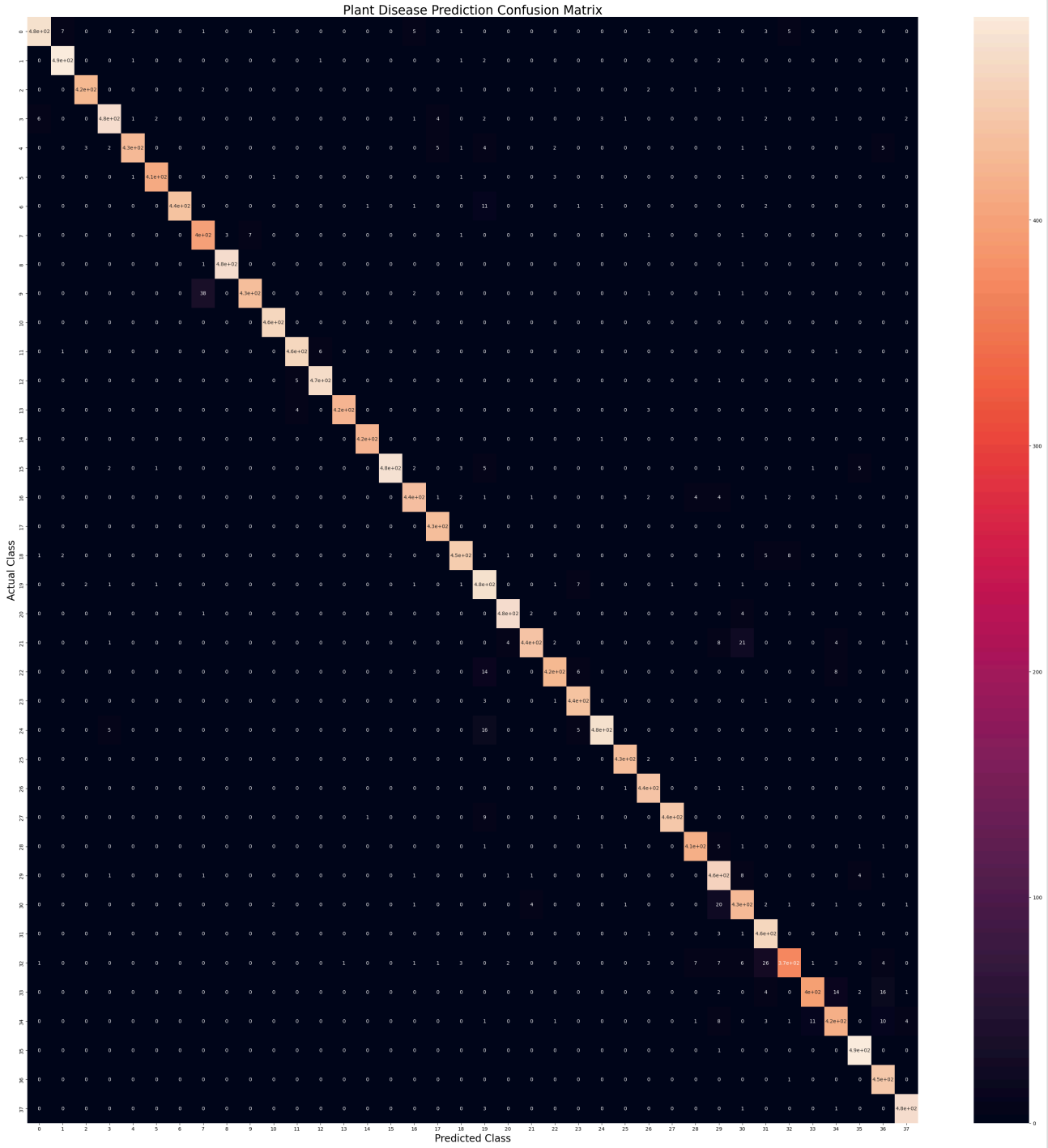
```
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(Y_true,predicted_categories)
```

```
# Precision Recall Fscore
print(classification_report(Y_true,predicted_categories,target_names=class_name))
```

```
                                                precision    recall  f1-score   support

                                Apple___Apple_scab       0.98      0.95      0.96       504
                                 Apple___Black_rot       0.98      0.99      0.98       497
                          Apple___Cedar_apple_rust       0.99      0.97      0.98       440
                                   Apple___healthy       0.98      0.95      0.96       502
                               Blueberry___healthy       0.99      0.95      0.97       454
          Cherry_(including_sour)___Powdery_mildew       0.99      0.98      0.98       421
                 Cherry_(including_sour)___healthy       1.00      0.96      0.98       456
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot       0.90      0.97      0.93       410
                       Corn_(maize)___Common_rust_       0.99      1.00      0.99       477
               Corn_(maize)___Northern_Leaf_Blight       0.98      0.91      0.95       477
                         Corn_(maize)___healthy       0.99      1.00      1.00       465
                              Grape___Black_rot       0.98      0.98      0.98       472
                   Grape___Esca_(Black_Measles)       0.99      0.99      0.99       480
        Grape___Leaf_blight_(Isariopsis_Leaf_Spot)       1.00      0.98      0.99       430
                                 Grape___healthy       1.00      1.00      1.00       423
          Orange___Haunglongbing_(Citrus_greening)       1.00      0.96      0.98       503
                           Peach___Bacterial_spot       0.96      0.95      0.96       459
                                  Peach___healthy       0.98      1.00      0.99       432
                    Pepper,_bell___Bacterial_spot       0.97      0.95      0.96       478
                           Pepper,_bell___healthy       0.86      0.96      0.91       497
                            Potato___Early_blight       0.98      0.98      0.98       485
                             Potato___Late_blight       0.98      0.92      0.95       485
                                 Potato___healthy       0.97      0.93      0.95       456
                              Raspberry___healthy       0.96      0.99      0.97       445
                                Soybean___healthy       0.99      0.95      0.97       505
                          Squash___Powdery_mildew       0.98      0.99      0.99       434
                         Strawberry___Leaf_scorch       0.96      0.99      0.98       444
                             Strawberry___healthy       1.00      0.98      0.99       456
                          Tomato___Bacterial_spot       0.97      0.97      0.97       425
                            Tomato___Early_blight       0.87      0.96      0.91       480
                             Tomato___Late_blight       0.89      0.93      0.91       463
                               Tomato___Leaf_Mold       0.90      0.99      0.94       470
                      Tomato___Septoria_leaf_spot       0.94      0.85      0.89       436
     Tomato___Spider_mites Two-spotted_spider_mite       0.97      0.91      0.94       435
                             Tomato___Target_Spot       0.92      0.91      0.92       457
           Tomato___Tomato_Yellow_Leaf_Curl_Virus       0.97      1.00      0.99       490
                     Tomato___Tomato_mosaic_virus       0.92      1.00      0.96       448
                                 Tomato___healthy       0.98      0.99      0.98       481

                                         accuracy                           0.96     17572
                                        macro avg       0.96      0.96      0.96     17572
                                     weighted avg       0.96      0.96      0.96     17572
```

Confusion Matrix

```
plt.figure(figsize=(40, 40))
sns.heatmap(cm,annot=True,annot_kws={"size": 10})

plt.xlabel('Predicted Class',fontsize = 20)
plt.ylabel('Actual Class',fontsize = 20)
plt.title('Plant Disease Prediction Confusion Matrix',fontsize = 25)
plt.show()
```




```
Start coding or generate with AI.
```

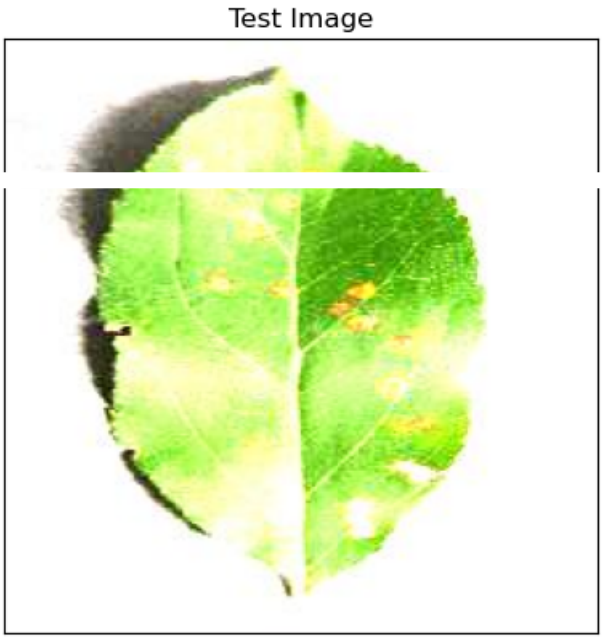
```
cnn = tf.keras.models.load_model(r'C:\Users\Dhanush\trained_plant_disease_model.h5')
```

```
Start coding or generate with AI.
```

Visualizing and Performing prediction on single leaf

```
#Test Image Visualization
import cv2
image_path = r'/content/drive/MyDrive/test/test/AppleCedarRust1.JPG'
# Reading an image in default mode
img = cv2.imread(image_path)
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB) #Converting BGR to RGB
```

```
# Displaying the image
plt.imshow(img)
plt.title('Test Image')
plt.xticks([])
plt.yticks([])
```




```
image = tf.keras.preprocessing.image.load_img(image_path,target_size=(128,128))
input_arr = tf.keras.preprocessing.image.img_to_array(image)
input_arr = np.array([input_arr])  # Convert single image to a batch.
predictions = cnn.predict(input_arr)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 754ms/step
```

```
print(predictions)
```

```
[[1.67697287e-08 6.60685430e-07 9.99862671e-01 9.36219546e-10
  3.30470289e-06 2.79887646e-09 1.41133916e-09 1.22803343e-07
  6.05909004e-11 3.10912407e-10 3.90011634e-10 3.39333274e-05
  6.67333211e-07 7.79392018e-09 2.69829425e-09 2.97298630e-08
  6.63106846e-07 5.47271606e-09 9.15810597e-06 6.44825332e-06
```