Plant Leaf Diseases using Tensorflow

Nishchal Krishnappa(0609414)

MS in Business Analytics

Golden Gate university

Table of Contents

Executive Summary

Plant diseases may reduce agricultural output and food security. Leaf disease, caused by fungus, bacteria, and viruses, is a frequent plant disease. Poor crop yields, quality, and economic losses may result from these diseases.

Effective leaf disease management and prevention need early identification and precise diagnosis. Image recognition and machine learning have allowed automated systems to swiftly and correctly diagnose leaf diseases. These platforms enable farmers and agricultural specialists choose treatment and preventative methods.

in this project, we apply machine learning skill to predict the disease in plant.over approach leverage python, along with libraries like tensorflow,pandas, seaborn, matplolib, and various other libraries to process and analyze the image of leaf. For agricultural systems to survive, plant disease detection and management research must continue. We can safeguard our crops and maintain food security for future generations by using cutting-edge science and technology.

Introduction to the data

I have extracted the data from Kaggal website which contains train, validation, and test image in which they are 14 different plants leaf which are tomato, straw Berry, Squash, Soy Bean, Raspberry, Potato, Pepper, Peach, Orange, Grape, Corn, Cheery, Blue Berry, Apple this all plant continue different type of disease

'Apple___Apple_scab',

'Apple___Black_rot',

'Apple___Cedar_apple_rust',

'Apple___healthy',

'Blueberry___healthy',

'Cherry_(including_sour)___Powdery_mildew',

'Cherry_(including_sour)___healthy',

'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',

'Corn_(maize)___Common_rust_',

'Corn_(maize)___Northern_Leaf_Blight',

'Corn_(maize)___healthy',

'Grape___Black_rot',

'Grape___Esca_(Black_Measles)',

'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',

'Grape___healthy',

'Orange___Haunglongbing_(Citrus_greening)',

'Peach___Bacterial_spot',

'Peach___healthy',

'Pepper,_bell___Bacterial_spot',

'Pepper,_bell___healthy',

'Potato___Early_blight',

'Potato___Late_blight',

'Potato___healthy',

'Raspberry___healthy',

'Soybean___healthy',

'Squash___Powdery_mildew',

'Strawberry___Leaf_scorch',

'Strawberry___healthy',

'Tomato___Bacterial_spot',

'Tomato___Early_blight',

'Tomato___Late_blight',

'Tomato___Leaf_Mold',

'Tomato___Septoria_leaf_spot',

'Tomato___Spider_mites Two-spotted_spider_mite',

'Tomato___Target_Spot',

'Tomato___Tomato_Yellow_Leaf_Curl_Virus',

'Tomato___Tomato_mosaic_virus',

'Tomato___healthy'

each disease contains above 2000 and more images which well is useful for training over model and also uses validation image so model learns at best.

Image Integration Process

First, I will import all required libraries.

```
[1]: import tensorflow as tf
     import matplotlib.pyplot as plt
     import pandas as pd
     import seaborn as sns
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.layers import Conv2D
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import MaxPool2D
     from tensorflow.keras.layers import Flatten
     from tensorflow.keras.layers import Dropout
     from sklearn.metrics import classification_report
     from sklearn.metrics import confusion_matrix
     import warnings
     warnings.filterwarnings('ignore')
```

Second I have loaded image data using Tensorful and Keras.

```
[2]: df_train = tf.keras.utils.image_dataset_from_directory(
         "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,
         pad_to_aspect_ratio=True,
     )
     Found 69768 files belonging to 38 classes.
```

Train

- Function: 'tf.keras.utils.image_dataset_from_directory' is a TensorFlow utility to load image data from a directory and prepare it for training a neural network model.
- Directory: The directory named "train" is specified, indicating this is where the training images are located.
- Labels: The labels are "inferred" from the directory structure.
- Label Mode: Set to "categorical", meaning the labels will be one-hot encoded.
- Class Names: Default value 'None' is used, which will automatically infer class names from the subdirectory names in the 'train' directory.
- Color Mode: Set to "RGB", indicating that the images are in color.
- Batch Size: The data batches are configured to have a size of 32.
- Image Size: The dimensions of each picture are adjusted to 128 by 128 pixels.
- Shuffle: Set to 'True', meaning the data will be shuffled.
- Seed: No specific random seed is provided ('None').
- Validation Split: No split is specified for validation data, which is indicated by 'None'.
- Subset: Not applicable, as no validation split is being used.
- Interpolation: Set to "bilinear" for resizing images.
- Follow Links: Set to 'False', so symbolic links will not be followed.
- Crop to Aspect Ratio: Set to 'False', so images will not be cropped to maintain the aspect ratio.
- Pad to Aspect Ratio: Set to 'True', so images will be padded to maintain aspect ratio if necessary.

after the process the code recogized 69768 file the belong to 38 classes means datset has been successfully created with image from 38 different categories.

```
[3]: df_valid = tf.keras.utils.image_dataset_from_directory(
         "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,
         pad_to_aspect_ratio=True,
     )
     Found 17452 files belonging to 38 classes.
```

validation

- Function: tf.keras.utils.image_dataset_from_directory is again used, similar to the previous code snippet for the training dataset, but this time for the validation dataset.
- Directory: The "valid" directory is specified, indicating the location of validation images.
- Labels: Set to "inferred", where the labels are automatically inferred from the subdirectory names within the "valid" directory.
- Label Mode: The "categorical" setting denotes that labels will be one-hot encoded.
- Class Names: Not specified, so class names will be automatically inferred as before.
- Color Mode: "rgb", meaning the images are color images.
- Batch Size: Set to 32, determining the number of images to process at once during model validation.
- Image Size: Resizing images to (128, 128) pixels.
- Shuffle: True, meaning the order of images will be shuffled.
- Seed: Not set, which means a random seed will be used for shuffling.
- Validation Split: Not applicable since this dataset is already for validation.
- Subset: Not specified since the validation set is already determined.
- Interpolation: "bilinear", used for image size scaling.
- Follow Links: False, indicating symbolic links in the directory will not be followed.
- Crop to Aspect Ratio: False, meaning images will not be cropped to maintain their aspect ratio.
- Pad to Aspect Ratio: True, so images will be padded as needed to maintain the aspect ratio.

after the process, the code recogized 17452 file the belong to 38 classes means datset has been successfully created with image from 38 different categories.

Sequential Model

```
[4]: model = Sequential()

[5]: 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))

[6]: 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))

[7]: 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))

[8]: 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))

[9]: 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))
```

1. the image of a plant leaf is input. it must have a fixed size,and it's usually in RGB color mode which mean red, green, blue.

2. Conv2D in this image get n-number of filter where it detects different features or patterns in the image, such as edges, color, or texture which is important for recongnizing various disease symptoms like spots in the leaf

3. Relu introduce non-linearities into the model. which activate function helps the model to account for the non-linear relationship between the features.

4. MaxPool2D the layers downsample the image by reducing its dimensions while preserving the most important feature detected by the filters. this helpto reduce computational load and control overfitting.

5. flattening after several convolutional and pooling layers, the high-level features of the plant image will be flattened into a one-dimensional array to serve as input to fully connected layers.

6. connecting layers is a dense layers further process the feature by combing them in various ways to make final prediction to a decision-making process where all the learned patterns are weighed and lead to a disease classification.

7. the last layers is typically with softmax activation function if its a multi-class classification.it gives a outputs a probability distribution over the diffrent possible dieseases and the highest probability determines the model's final prediction.

check for overfitting

```
[10]: model.add(Dropout(0.25))

[11]: model.add(Flatten())

[12]: model.add(Dense(units=2000,activation='relu'))

[13]: model.add(Dropout(0.4))

[14]: model.add(Dense(units = 38, activation='softmax'))

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

In the Keras model, the code is configuring a neural network to identify plant diseases. The network includes dropout layers to prevent overfitting, which is critical due to the complexity of the patterns it must learn from the plant images. After flattening the convolutional layer outputs, it employs a dense layer with 2000 neurons to capture a wide array of features and patterns indicative of various plant diseases. Another dropout is then used to further mitigate overfitting before proceeding to the output layer, which comprises 38 neurons corresponding to the number of plant disease classes it can identify. This layer uses the softmax activation to output probabilities across these classes. The network is compiled with the Adam optimizer, a modest learning rate for steady convergence, and it optimizes for classification accuracy using categorical crossentropy as the loss function, which is standard for multi-class classification problems.

Summary

This picture illustrates the architectural blueprint of a deep learning model specifically designed to excel in image classification tasks, with a particular focus on identifying plant diseases. This model utilizes convolutional layers, namely the `Conv2D` layers, to extract characteristics from images.

These layers progressively capture more complex features as we go deeper into the network. `Conv2D` layer, a pooling layer (`MaxPooling2D`) is used to reduce the spatial dimensions of the feature maps. This enhances computing efficiency and reduces the likelihood of overfitting.

The `Dropout` layers included into the design selectively deactivate a fraction of neurons during training, thereby mitigating overfitting by discouraging the network from excessively depending on any one neuron.

The `Flatten` layer converts the 2D feature maps into a 1D feature vector, making it possible to connect to `Dense` layers, which are fully connected neural networks. The `Dense` layer with 2000 units acts as a feature combiner, interpreting the features extracted to make a final prediction.

Finally, the network concludes with a `Dense` output layer with 38 units (each representing a class of plant disease) and uses a `softmax` activation function to generate a probability distribution across these classes. The model is compiled with the Adam optimizer, uses categorical crossentropy for loss (which is typical for multi-class problems), and focuses on 'accuracy' as the primary metric.

Overall, this model represents a complex and capable CNN, designed to discern subtle patterns indicative of different plant diseases from input images, with a total of over 8.8 million parameters, making it a powerful tool for automatic plant disease diagnosis.

```
[16]: model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	896
conv2d_1 (Conv2D)	(None, 126, 126, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_2 (Conv2D)	(None, 63, 63, 64)	18,496
conv2d_3 (Conv2D)	(None, 61, 61, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_4 (Conv2D)	(None, 30, 30, 128)	73,856
conv2d_5 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_6 (Conv2D)	(None, 14, 14, 256)	295,168
conv2d_7 (Conv2D)	(None, 12, 12, 256)	590,080
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_8 (Conv2D)	(None, 6, 6, 512)	1,180,160
conv2d_9 (Conv2D)	(None, 4, 4, 512)	2,359,808
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 512)	0
dropout (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 2000)	4,098,000
dropout_1 (Dropout)	(None, 2000)	0
dense_1 (Dense)	(None, 38)	76,038

Total params: 8,886,262 (33.90 MB)
Trainable params: 8,886,262 (33.90 MB)
Non-trainable params: 0 (0.00 B)

model learning

```
[17]: train_hist = model.fit(x=df_train,validation_data=df_valid,epochs=5)
Epoch 1/5
2181/2181 ━━━━━━━━━━━━━━━━━━━━ 1948s 889ms/step - accuracy: 0.4221 - loss: 2.0219 - val_accuracy: 0.8399 - val_loss: 0.5270
Epoch 2/5
2181/2181 ━━━━━━━━━━━━━━━━━━━━ 1720s 788ms/step - accuracy: 0.8344 - loss: 0.5217 - val_accuracy: 0.9014 - val_loss: 0.3128
Epoch 3/5
2181/2181 ━━━━━━━━━━━━━━━━━━━━ 1735s 795ms/step - accuracy: 0.9043 - loss: 0.2925 - val_accuracy: 0.9253 - val_loss: 0.2333
Epoch 4/5
2181/2181 ━━━━━━━━━━━━━━━━━━━━ 1716s 787ms/step - accuracy: 0.9342 - loss: 0.1998 - val_accuracy: 0.9283 - val_loss: 0.2268
Epoch 5/5
2181/2181 ━━━━━━━━━━━━━━━━━━━━ 1812s 831ms/step - accuracy: 0.9506 - loss: 0.1484 - val_accuracy: 0.9436 - val_loss: 0.1733

evalution
```

The model undergoes 5 epochs of training, during which we notice a rise in training accuracy and a reduction in training loss. This indicates that the model is progressively learning and improving its capacity to accurately categorize the data. we can also mention there is no sign of overfitting in the learning process. In validation accuracy this is the accuracy on a separate dataset not used during training the validation accuracy starts at 0.8399 and increases to 0.9436 when it comes to loss it also decreases from 0.5270 to 0.1733, showing the model's improving performance on the validation set.

```
[18]: train_loss, train_accuracy = model.evaluate(df_train)
      2181/2181 ━━━━━━━━━━━━━━━━━━━━ 394s 181ms/step - accuracy: 0.9689 - loss: 0.0936

[19]: print("training_loss:",train_loss)
      training_loss: 0.08464037626981735

[20]: val_loss,val_acc = model.evaluate(df_valid)
      546/546 ━━━━━━━━━━━━━━━━━━━━ 93s 170ms/step - accuracy: 0.9427 - loss: 0.1764

[21]: print("validation:",val_loss)
      validation: 0.17334075272083282

[22]: model.save("trained_model.keras")

[23]: train_hist.history

[23]: {'accuracy': [0.6174033880233765,
        0.85845947265625,
        0.9154913425445557,
        0.9409041404724121,
        0.9564986824989319],
       'loss': [1.2803746461868286,
        0.439273864030838,
        0.25865307450294495,
        0.17899763584136963,
        0.1307966262102127],
       'val_accuracy': [0.839903712272644,
        0.9014439582824707,
        0.925280749797821,
        0.928317666053772,
        0.9436167478561401],
       'val_loss': [0.5269742012023926,
        0.31284603476524353,
        0.23331646621227264,
        0.22682936489582062,
        0.17334066331386566]}
```

Creating JSON file

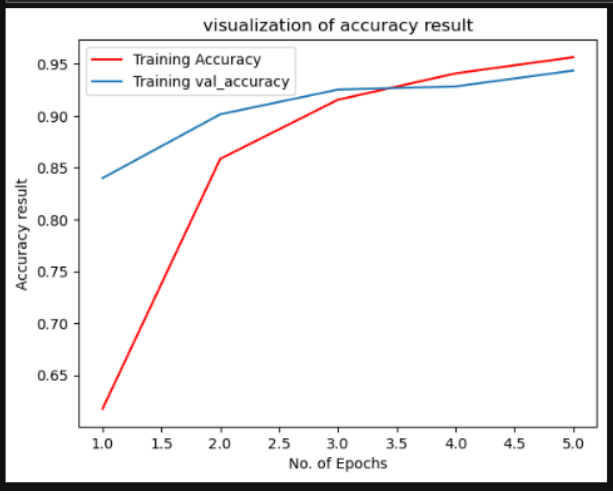
we create a JSON file for accuracy, loss, val_accuracy,val_los and plot the graph for training accuracy and training val_accuracy as well as loss and val_accuracy.

```
[24]: #Recording history in json
      import json
      with open("training_hist.json","w") as f:
          json.dump(train_hist.history,f)
```

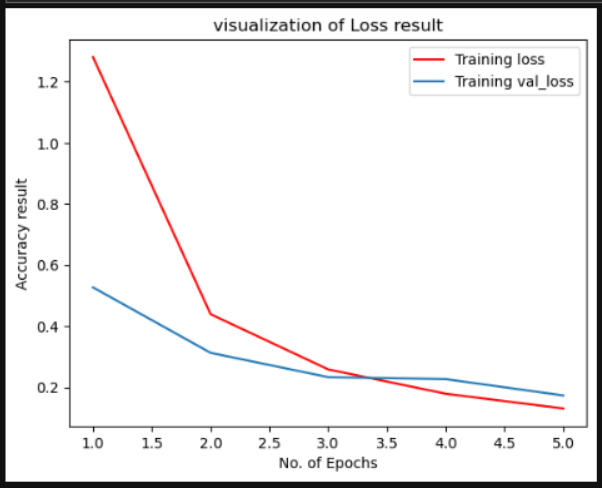
the graph would indicate that as the number of epochs increases, both the training and validation accuracies improve, suggesting that the model is learning from the data. Initially, there's a notable difference between training accuracy and validation accuracy, with training accuracy being lower. However, over subsequent epochs, the training accuracy swiftly improves, suggesting the model is effectively capturing the patterns within the training data.

A convergence of training and validation accuracy values is a positive sign, indicating that the model is generalizing well rather than memorizing the training data (overfitting). By the final epoch, validation accuracy slightly surpasses training accuracy, which is an excellent outcome. Typically, one might expect the training accuracy to be higher because the model is directly learning from this data. However, in this case, the higher validation accuracy could imply either a well-generalizing model or a validation set that happens to contain examples the model finds easier to classify correctly.

[26]:
```
epochs = [i for i in range(1,6)]
plt.plot(epochs,train_hist.history["accuracy"],color = "red",label = "Training Accuracy")
plt.plot(epochs,train_hist.history["val_accuracy"],label = "Training val_accuracy")
plt.xlabel("No. of Epochs")
plt.ylabel("Accuracy result")
plt.title("visualization of accuracy result")
plt.legend()
plt.show()
```



[40]:
```
epochs1 = [i for i in range(1,6)]
plt.plot(epochs,train_hist.history["loss"],color = "red",label = "Training loss")
plt.plot(epochs,train_hist.history["val_loss"],label = "Training val_loss")
plt.xlabel("No. of Epochs")
plt.ylabel("Accuracy result")
plt.title("visualization of Loss result")
plt.legend()
plt.show()
```



Seeing validation class name

```
[27]: class_name = df_valid.class_names
      class_name

[27]: ['Apple___Apple_scab',
       'Apple___Black_rot',
       'Apple___Cedar_apple_rust',
       'Apple___healthy',
       'Blueberry___healthy',
       'Cherry_(including_sour)___Powdery_mildew',
       'Cherry_(including_sour)___healthy',
       'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
       'Corn_(maize)___Common_rust_',
       'Corn_(maize)___Northern_Leaf_Blight',
       'Corn_(maize)___healthy',
       'Grape___Black_rot',
       'Grape___Esca_(Black_Measles)',
       'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
       'Grape___healthy',
       'Orange___Haunglongbing_(Citrus_greening)',
       'Peach___Bacterial_spot',
       'Peach___healthy',
       'Pepper,_bell___Bacterial_spot',
       'Pepper,_bell___healthy',
       'Potato___Early_blight',
       'Potato___Late_blight',
       'Potato___healthy',
       'Raspberry___healthy',
       'Soybean___healthy',
       'Squash___Powdery_mildew',
       'Strawberry___Leaf_scorch',
       'Strawberry___healthy',
       'Tomato___Bacterial_spot',
       'Tomato___Early_blight',
       'Tomato___Late_blight',
       'Tomato___Leaf_Mold',
       'Tomato___Septoria_leaf_spot',
       'Tomato___Spider_mites Two-spotted_spider_mite',
       'Tomato___Target_Spot',
       'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
       'Tomato___Tomato_mosaic_virus',
       'Tomato___healthy']
```

Testing

The test_df created using tf.keras.utils.image_dataset_from_directory suggests the test dataset is prepared in a similar fashion to the training and validation datasets, with the same image resizing and color settings.

When the model.predict(test_df) function is called, it generates predictions for each input from the test dataset. The predictions, y_pred, are in the form of arrays containing probabilities corresponding to the likelihood that a given input belongs to each of the 38 classes the model is trained to recognize. Each array represents one test example and the probabilities sum to 1.0 across all potential classes for that example.

The values in y_pred show a high degree of confidence (values close to 1 or 0) for certain classes in some predictions. For instance, the first entry has a high probability for the first class, and similarly high confidence is observed in other entries for other classes.

This suggests that the model is quite certain about its predictions, indicating clear decision boundaries have been learned for these particular cases. Such high confidence is desired, yet it's essential to assess whether these probabilities correspond to correct classifications, which would require comparing y_pred against true labels. If the model's high-confidence predictions are accurate, it suggests robust learning and good generalization. However, if they are not matching the true labels, this might be a case of overfitting to the training data or a lack of

[29]:
```
test_df = tf.keras.utils.image_dataset_from_directory(
    "valid",
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=False,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
    pad_to_aspect_ratio=True,
)
```

```
Found 17452 files belonging to 38 classes.
```

[30]:
```
y_pred = model.predict(test_df)
y_pred
```

```
546/546 ━━━━━━━━━━━━━━━━━━━━ 93s 168ms/step
```

[30]:
```
array([[9.9315697e-01, 1.6451147e-03, 8.5298765e-05, ..., 3.0048350e-09,
        2.7868265e-09, 2.0375921e-08],
       [9.9978298e-01, 6.1737155e-05, 7.5652511e-06, ..., 1.4620013e-09,
        2.3213390e-08, 1.8891106e-09],
       [9.9998820e-01, 1.9614790e-06, 8.9938339e-06, ..., 2.4964713e-12,
        7.3237395e-11, 5.8171207e-10],
       ...,
       [1.7441233e-14, 3.7415052e-16, 1.6658319e-11, ..., 5.7008100e-15,
        1.5104671e-14, 1.0000000e+00],
       [8.2563418e-14, 1.8444940e-16, 8.8548474e-12, ..., 4.1491718e-15,
        3.6885346e-15, 1.0000000e+00],
       [4.2780222e-16, 8.1393993e-18, 1.5395696e-16, ..., 1.0593089e-16,
        5.1826736e-15, 1.0000000e+00]], dtype=float32)
```

[31]:
```
y_pred.shape
```

[31]:
```
(17452, 38)
```

[32]:
```
PREDICTED_CATEGORIES = tf.argmax(y_pred,axis=1)
```

[33]:
```
PREDICTED_CATEGORIES
```

[33]:
```
<tf.Tensor: shape=(17452,), dtype=int64, numpy=array([ 0,  0,  0, ..., 37, 37, 37], dtype=int64)>
```

[34]:
```
true_categories = tf.concat([y for x,y in test_df],axis=0)
true_categories
```

[34]:
```
<tf.Tensor: shape=(17452, 38), dtype=float32, numpy=
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)>
```

[35]:
```
y_true = tf.argmax(true_categories,axis=1)
y_true
```

[35]:
```
<tf.Tensor: shape=(17452,), dtype=int64, numpy=array([ 0,  0,  0, ..., 37, 37, 37], dtype=int64)>
```

Predicted categories

Check for recall, f1-Score, Support, Precision

```
[37]: print(classification_report(y_true,PREDICTED_CATEGORIES,target_names=class_name))
```

	precision	recall	f1-score	support
Apple___Apple_scab	0.99	0.87	0.93	504
Apple___Black_rot	0.99	0.95	0.97	497
Apple___Cedar_apple_rust	0.99	0.95	0.97	440
Apple___healthy	0.88	0.95	0.91	502
Blueberry___healthy	0.94	0.99	0.96	454
Cherry_(including_sour)___Powdery_mildew	0.97	0.97	0.97	421
Cherry_(including_sour)___healthy	0.90	1.00	0.94	456
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	0.79	0.95	0.86	290
Corn_(maize)___Common_rust_	0.97	0.99	0.98	477
Corn_(maize)___Northern_Leaf_Blight	0.98	0.85	0.91	477
Corn_(maize)___healthy	0.96	1.00	0.98	465
Grape___Black_rot	0.95	0.90	0.92	472
Grape___Esca_(Black_Measles)	0.91	0.99	0.95	480
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	0.98	1.00	0.99	430
Grape___healthy	1.00	0.99	0.99	423
Orange___Haunglongbing_(Citrus_greening)	0.98	0.95	0.96	503
Peach___Bacterial_spot	0.96	0.91	0.94	459
Peach___healthy	0.97	0.99	0.98	432
Pepper,_bell___Bacterial_spot	0.97	0.92	0.95	478
Pepper,_bell___healthy	0.96	0.95	0.95	497
Potato___Early_blight	0.91	0.99	0.95	485
Potato___Late_blight	0.98	0.91	0.94	485
Potato___healthy	0.97	0.97	0.97	456
Raspberry___healthy	0.97	0.99	0.98	445
Soybean___healthy	0.96	0.98	0.97	505
Squash___Powdery_mildew	0.97	0.96	0.97	434
Strawberry___Leaf_scorch	0.96	0.97	0.96	444
Strawberry___healthy	0.98	1.00	0.99	456
Tomato___Bacterial_spot	0.97	0.88	0.93	425
Tomato___Early_blight	0.85	0.90	0.87	480
Tomato___Late_blight	0.89	0.87	0.88	463
Tomato___Leaf_Mold	0.89	0.97	0.92	470
Tomato___Septoria_leaf_spot	0.98	0.72	0.83	436
Tomato___Spider_mites Two-spotted_spider_mite	0.88	0.97	0.92	435
Tomato___Target_Spot	0.81	0.88	0.85	457
Tomato___Tomato_Yellow_Leaf_Curl_Virus	0.98	0.98	0.98	490
Tomato___Tomato_mosaic_virus	1.00	0.86	0.92	448
Tomato___healthy	0.95	1.00	0.97	481
accuracy			0.94	17452
macro avg	0.94	0.94	0.94	17452
weighted avg	0.95	0.94	0.94	17452

This report provides detailed performance metrics such as precision, recall, f1-score, and support for each class.

Precision measures the accuracy of positive predictions for each class, while recall (or sensitivity) assesses the model's ability to find all the positive samples. The F1-score combines precision and recall into a single metric that balances both.

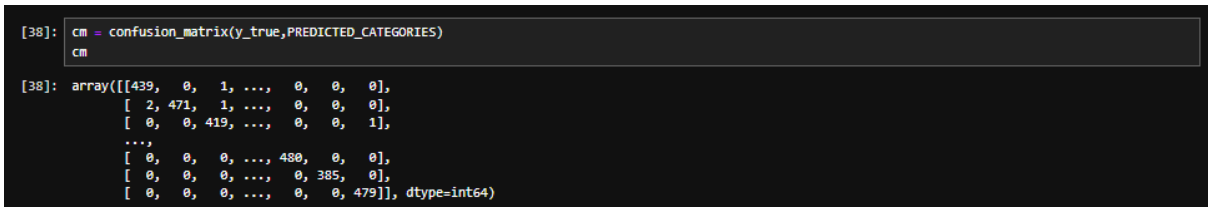
The high precision and recall values across most classes suggest that the model is performing exceptionally well at correctly identifying the various plant diseases, as well as being healthy. The F1-scores, which are also high, confirm this balanced performance between precision and recall.high macro-average F1-score of 0.94, indicating very good overall performance. The weighted average takes class imbalance into account and is also high at 0.94, showing that the model is robust even when class distribution varies.

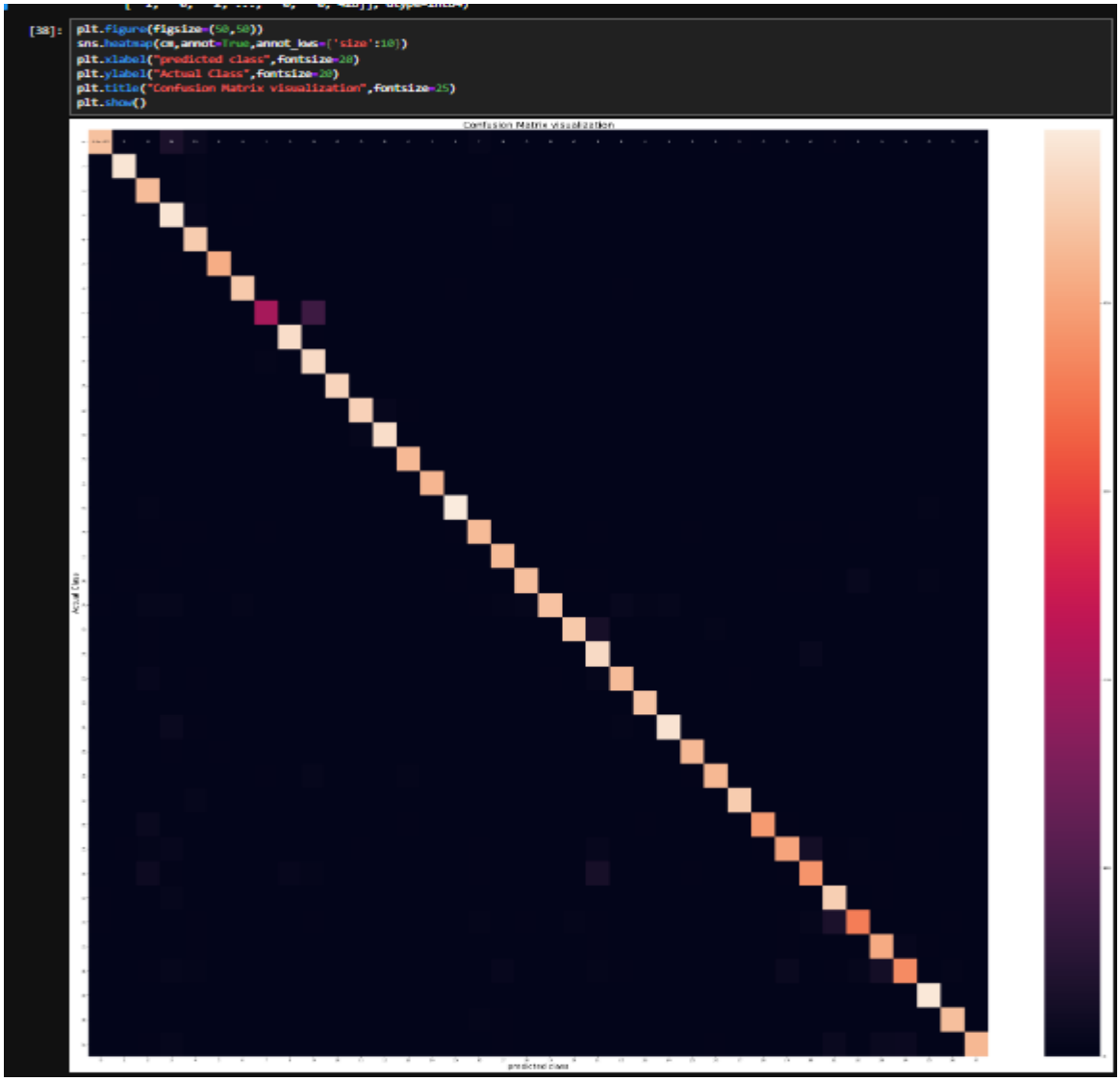
Confusion Maxtrix visualization

test by passing image path of test from test file.The array is then expanded by one dimension to create a batch containing a single image, as neural network models typically expect batches of data for prediction. This is evidenced by the use of np.expand_dims on the first axis of the array.

Following preprocessing, the model makes a prediction, resulting in an array of probabilities across 38 different classes, which likely correspond to various plant diseases and a healthy class. The shape of the prediction (1, 38) confirms it's for a single image across 38 classes.

Finally, np.argmax is used to extract the index of the highest probability, which indicates the model's chosen class for the image. The index of 37 suggests that the model has classified the image into the last of the 38 classes, potentially indicating a 'healthy' class if ordered sequentially.





Check the model on test Folder to check its predicting right

```
[2]: model = tf.keras.models.load_model("trained_model.keras")
```

I have saved the trained_model in the train program and loaded that in a new test notebook to checking if the model is working fine.

[3]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	896
conv2d_1 (Conv2D)	(None, 126, 126, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_2 (Conv2D)	(None, 63, 63, 64)	18,496
conv2d_3 (Conv2D)	(None, 61, 61, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_4 (Conv2D)	(None, 30, 30, 128)	73,856
conv2d_5 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_6 (Conv2D)	(None, 14, 14, 256)	295,168
conv2d_7 (Conv2D)	(None, 12, 12, 256)	590,080
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_8 (Conv2D)	(None, 6, 6, 512)	1,180,160
conv2d_9 (Conv2D)	(None, 4, 4, 512)	2,359,808
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 512)	0
dropout (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 2000)	4,098,000
dropout_1 (Dropout)	(None, 2000)	0
dense_1 (Dense)	(None, 38)	76,038

Total params: 26,658,788 (101.70 MB)
Trainable params: 8,886,262 (33.90 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 17,772,526 (67.80 MB)

Has you can see the summary is similar compared to the train dataset

```
[42]: import cv2
      image_path = "test/test/TomatoHealthy1.JPG"

[43]: #reading Image
      img = cv2.imread(image_path)
      img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

      plt.imshow(img)
      plt.title("Test Image")
      plt.xticks([ ])
      plt.yticks([ ])
      plt.show()
```


```
[4]: import cv2
     image_path = "test/test/AppleScab1.JPG"

[5]: #reading Image
     img = cv2.imread(image_path)
     img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

     plt.imshow(img)
     plt.title("Test Image")
     plt.xticks([ ])
     plt.yticks([ ])
     plt.show()
```

Here I have load the image using import cv2 and give the image path.

```
[44]: image = tf.keras.preprocessing.image.load_img(image_path,target_size=(128,128))
      input_arr = tf.keras.preprocessing.image.img_to_array(image)
      print(input_arr.shape)

      (128, 128, 3)

[45]: input_arr = np.expand_dims(input_arr,axis=0)

[46]: predication = model.predict(input_arr)
      predication,predication.shape

      1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step

[46]: (array([[2.6160475e-08, 1.0570507e-09, 4.7001902e-10, 6.7146391e-07,
               9.0455221e-10, 3.8139081e-07, 8.4450308e-11, 8.9966307e-12,
               1.9780863e-11, 1.9253292e-11, 1.0341833e-11, 9.9258156e-13,
               4.0644745e-09, 1.0376028e-12, 8.7438612e-13, 6.4980252e-12,
               3.5878495e-10, 3.8728430e-12, 5.8444960e-10, 1.4323974e-10,
               6.5809587e-08, 2.0969624e-09, 1.7181141e-09, 4.2142071e-12,
               1.4407876e-10, 4.1731028e-08, 9.3404708e-11, 4.0997095e-11,
               2.0516921e-11, 7.3041923e-10, 8.0292892e-08, 9.7433451e-07,
               2.0295685e-08, 4.8810245e-10, 2.8820002e-06, 3.0873227e-12,
               1.1110839e-11, 9.9999487e-01]], dtype=float32),
        (1, 38))

[47]: result_index = np.argmax(predication)
      result_index

[47]: 37
```

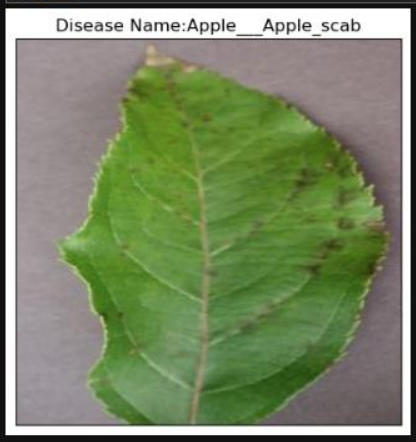
Providing class which we got from validation

```
[48]: class_name = ['Apple___Apple_scab',
       'Apple___Black_rot',
       'Apple___Cedar_apple_rust',
       'Apple___healthy',
       'Blueberry___healthy',
       'Cherry_(including_sour)___Powdery_mildew',
       'Cherry_(including_sour)___healthy',
       'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
       'Corn_(maize)___Common_rust_',
       'Corn_(maize)___Northern_Leaf_Blight',
       'Corn_(maize)___healthy',
       'Grape___Black_rot',
       'Grape___Esca_(Black_Measles)',
       'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
       'Grape___healthy',
       'Orange___Haunglongbing_(Citrus_greening)',
       'Peach___Bacterial_spot',
       'Peach___healthy',
       'Pepper,_bell___Bacterial_spot',
       'Pepper,_bell___healthy',
       'Potato___Early_blight',
       'Potato___Late_blight',
       'Potato___healthy',
       'Raspberry___healthy',
       'Soybean___healthy',
       'Squash___Powdery_mildew',
       'Strawberry___Leaf_scorch',
       'Strawberry___healthy',
       'Tomato___Bacterial_spot',
       'Tomato___Early_blight',
       'Tomato___Late_blight',
       'Tomato___Leaf_Mold',
       'Tomato___Septoria_leaf_spot',
       'Tomato___Spider_mites Two-spotted_spider_mite',
       'Tomato___Target_Spot',
       'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
       'Tomato___Tomato_mosaic_virus',
       'Tomato___healthy']
```

Result

```
[49]: model_predicaton = class_name[result_index]
      plt.imshow(img)
      plt.title(f"Disease Name:{model_predicaton}")
      plt.xticks([ ])
      plt.yticks([ ])
      plt.show()
```


```
[11]: model_predicaton = class_name[result_index]
      plt.imshow(img)
      plt.title(f"Disease Name:{model_predicaton}")
      plt.xticks([ ])
      plt.yticks([ ])
      plt.show()
```




```
[12]: model_predicaton

[12]: 'Apple___Apple_scab'
```

According to this model, it provides the name of the illness and the name of the plant.

References

(n.d.). Retrieved from https://keras.io/api/data_loading/image/

Agudelo, M. B. (2008). *A case study concerning the extraction and identification of plant-parasitic nematodes.* APS.

Jajoo, P. (26 May 2022). *A Case Study on Machine Learning Techniques for Plant Disease Identification*. Springer Link.

SAMIR. (Dataset). *New Plant Diseases Dataset.* Kaggle. Retrieved from https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset/code

https://github.com/nishchal69/Plant-leaf-Diseases-/tree/main