"Enhancing the performance of Plant Diseases Detection with Leaf Images by using Explainable Artificial Intelligence Techniques.":

Introduction Plants disease results in significant decrement in both the quantity and quality of farming products as well as productivity and economic losses.The Identification of plant disease is now getting more attention in monitoring vast fields of crops. Early detection of plant disease is very crucial since they have an impact on the development of affected species. There are numerous Machine Learning models that have been used to identify and categorise plant diseases, but recent years have seen tremendous advancements in Deep Learning, a branch of Machine Learning.This area of research appears to have a great deal of potential for increased accuracy. In this paper we have presented a plant disease detection system which will facilitate the progress in agriculture by predicting the disease and also suggest treatment to cure the plant disease.And these predictions are evaluated by using Explainable AI models such as Local Interpretable Model-Agnostic Explanations (LIME). This paper also include the image acquisition, image preprocessing, feature extraction,data augmentation,neural network based classification and Explainable AI.

" Training Model "

```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
import matplotlib.pyplot as plt
import numpy as np
import os
```

```
#basic cnn
# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution    128 -> 256
classifier.add(Conv2D(32, (3, 3), input_shape = (256, 256, 3), activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dense(units = 12, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

Data Augmentation : In leaf disease detection, collection and labelling a large number of disease images require lots of manpower, material resources and financial resources. For some certain plant diseases, their onset period is shorter, it is difficult to collect them. To artificially expand the size of our dataset and enhance the functionality of our model, we employed the Data Augmentation techniques.To be more precise, we randomly translated, rotated, and flipped the photos in our dataset in addition to changing the brightness and contrast levels. The main motivation for using data augmentation was due to the small size of our available dataset and complication of the issue we were researching. We were able to build a more reliable model by employing data augmentation to ten-fold the amount of our dataset.

```
train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)
```

```
test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('C:/Users/Jayraj/Prototype MK4/Dataset/train', # relative path from working directoy
                                                 target_size = (256, 256),
                                                 batch_size = 6, class_mode = 'categorical')
valid_set = test_datagen.flow_from_directory('C:/Users/Jayraj/Prototype MK4/Dataset/val', # relative path from working directoy
                                             target_size = (256, 256),
                                          batch_size = 3, class_mode = 'categorical')

labels = (training_set.class_indices)
print(labels)
```

```
Found 231 images belonging to 12 classes.
Found 238 images belonging to 12 classes.
{'Canker_fruits-disease': 0, 'Caterpillar worms leaf Disease': 1, 'Faint color fruit disease': 2, 'InitialBurn leaf Disease': 3, 'M
```

```
classifier.fit_generator(training_set,
                         steps_per_epoch = 20,
                         epochs = 50,
                         validation_data=valid_set
                         )
```

```
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 1.0110 - accuracy: 0.6833 - val_loss:
Epoch 8/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 1.0664 - accuracy: 0.6239 - val_loss:
Epoch 9/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.9955 - accuracy: 0.6581 - val_loss:
Epoch 10/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 1.0895 - accuracy: 0.6583 - val_loss:
Epoch 11/50
20/20 [==============================] - 24s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 1.1204 - accuracy: 0.6410 - val_loss:
Epoch 12/50
20/20 [==============================] - 23s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 1.0644 - accuracy: 0.5917 - val_loss:
Epoch 13/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 1.1094 - accuracy: 0.6923 - val_loss:
Epoch 14/50
20/20 [==============================] - 25s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 1.2087 - accuracy: 0.5833 - val_loss:
Epoch 15/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.9272 - accuracy: 0.6417 - val_loss:
Epoch 16/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.9331 - accuracy: 0.6496 - val_loss:
Epoch 17/50
20/20 [==============================] - 24s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.7591 - accuracy: 0.7333 - val_loss:
Epoch 18/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.8291 - accuracy: 0.7009 - val_loss:
Epoch 19/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.8305 - accuracy: 0.7094 - val_loss:
Epoch 20/50
20/20 [==============================] - 23s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.7228 - accuracy: 0.7583 - val_loss:
Epoch 21/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.6135 - accuracy: 0.8034 - val_loss:
Epoch 22/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.6120 - accuracy: 0.7265 - val_loss:
Epoch 23/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.6662 - accuracy: 0.7833 - val_loss:
Epoch 24/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.7670 - accuracy: 0.7521 - val_loss:
Epoch 25/50
20/20 [==============================] - 25s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.6868 - accuracy: 0.7667 - val_loss:
Epoch 26/50
20/20 [==============================] - 24s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.5550 - accuracy: 0.8083 - val_loss:
Epoch 27/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.6245 - accuracy: 0.8120 - val_loss:
Epoch 28/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.7273 - accuracy: 0.7167 - val_loss:
Epoch 29/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.5845 - accuracy: 0.8417 - val_loss:
Epoch 30/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 5.7000 - loss: 0.5899 - accuracy: 0.7895 - val_loss:
Epoch 31/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.8022 - accuracy: 0.7000 - val_loss:
Epoch 32/50
20/20 [==============================] - 22s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.6965 - accuracy: 0.8000 - val_loss:
Epoch 33/50
20/20 [==============================] - 23s 1s/step - batch: 9.5000 - size: 5.8500 - loss: 0.4499 - accuracy: 0.8376 - val_loss:
Epoch 34/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.7657 - accuracy: 0.7417 - val_loss:
Epoch 35/50
20/20 [==============================] - 21s 1s/step - batch: 9.5000 - size: 6.0000 - loss: 0.6291 - accuracy: 0.7833 - val_loss:
```

```
classifier_json=classifier.to_json()
with open("model1.json", "w") as json_file:
    json_file.write(classifier_json)
# serialize weights to HDF5
    classifier.save_weights("my_model_weights.h5")
    classifier.save("model.h5")
    print("Saved model to disk")
```

```
Saved model to disk
```