University Institute of Engineering
Department of Electronics & Communication Engineering

Experiment No. :- 10

Student Name: Priyanshu Mathur **UID:** 20BEC1073
Branch: Electronics and Communication **Section/Group:** A
Semester: 7th **Date of Performance:** 15/11/2023
Subject Name: AIML **Subject Code:** 20ECA-445

1. **Aim of the practical:** Write a program to predict Leaf Disease Detection and Analysis.

2. **Tool Used:** Google Colab

3. **Theory:** The leaf disease detection process using Artificial Intelligence and Machine Learning (AIML) starts with collecting a diverse dataset of healthy and diseased plant leaves. After data pre-processing, including cleaning and augmentation, pre-trained Convolutional Neural Networks (CNNs) are employed for feature extraction to enable effective pattern recognition. Machine learning techniques, focusing on feature extraction and classification, are widely utilized in plant disease detection. These methods extract image features (color, texture, shape) to train classifiers distinguishing between healthy and diseased plants. While successful for various diseases and stress symptoms, these techniques face challenges in identifying subtle symptoms and early-stage diseases. The integration of ML and DL techniques in plant disease detection is a rapidly advancing field, showing promising results and ongoing efforts to enhance model robustness and accuracy.

4. **Steps for experiment/practical:**

Step 1: - Open Google Colab

Step 2: - Create a new notebook

Step 3: - Write the code given below and run it.

Program Code:

Plot :-
```
import os
import cv2
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

from tensorflow import keras
from keras.models import Sequential
from keras.layers import
Conv2D,MaxPooling2D,Dense,Flatten,Dropout
```

2) Data loading and exploration
```
fpath = "../input/plantdisease/PlantVillage/"
random_seed = 111

categories = os.listdir(fpath)
def load_images_and_labels(categories):
    img_lst=[]
    labels=[]
    for index, category in enumerate(categories):
        for image_name in
os.listdir(fpath+"/"+category)[:300]:
            file_ext = image_name.split(".")[-1]
            if (file_ext.lower() == "jpg") or (file_ext.lower()
== "jpeg"):
```

```
            #print(f"\nCategory = {category}, Image name =
{image_name}")
            img =
cv2.imread(fpath+"/"+category+"/"+image_name)
            img = cv2.cvtColor(img,
cv2.COLOR_BGR2RGB)

            img_array = Image.fromarray(img, 'RGB')

            #resize image to 227 x 227 because the input
image resolution for AlexNet is 227 x 227
            resized_img = img_array.resize((227, 227))

            img_lst.append(np.array(resized_img))

            labels.append(index)
    return img_lst, labels

images, labels = load_images_and_labels(categories)
images = np.array(images)
labels = np.array(labels)
```

- **Display few random images from dataset with their label**

```
def display_rand_images(images, labels):
    plt.figure(1 , figsize = (19 , 10))
    n = 0
    for i in range(9):
        n += 1
        r = np.random.randint(0 , images.shape[0] , 1)
```

```
        plt.subplot(3 , 3 , n)
        plt.subplots_adjust(hspace = 0.3 , wspace = 0.3)
        plt.imshow(images[r[0]])

        plt.title('Plant label : {}'.format(labels[r[0]]))
        plt.xticks([])
        plt.yticks([])

    plt.show()

display_rand_images(images, labels)
```

3) Prepare data for CNN model training

- **Step 1 - shuffle the data loaded from the dataset**

```
#1-step in data shuffling

#get equally spaced numbers in a given range
n = np.arange(images.shape[0])


#shuffle all the equally spaced values in list 'n'
np.random.seed(random_seed)
np.random.shuffle(n)
#2-step in data shuffling

#shuffle images and corresponding labels data in both the
lists
images = images[n]
labels = labels[n]
```

```
images = images.astype(np.float32)
labels = labels.astype(np.int32)
images = images/255
display_rand_images(images, labels)
x_train, x_test, y_train, y_test = train_test_split(images,
labels, test_size = 0.2, random_state = random_seed)

display_rand_images(x_train, y_train)
model=Sequential()

#1 conv layer
model.add(Conv2D(filters=96,kernel_size=(11,11),strides
=(4,4),padding="valid",activation="relu",input_shape=(
227,227,3)))

#1 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())

#2 conv layer
model.add(Conv2D(filters=256,kernel_size=(5,5),strides=
(1,1),padding="valid",activation="relu"))

#2 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())

#3 conv layer
```

```
model.add(Conv2D(filters=384,kernel_size=(3,3),strides=
(1,1),padding="valid",activation="relu"))

#4 conv layer
model.add(Conv2D(filters=384,kernel_size=(3,3),strides=
(1,1),padding="valid",activation="relu"))

#5 conv layer
model.add(Conv2D(filters=256,kernel_size=(3,3),strides=
(1,1),padding="valid",activation="relu"))

#3 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())


model.add(Flatten())

#1 dense layer
model.add(Dense(4096,input_shape=(227,227,3),activatio
n="relu"))

model.add(Dropout(0.4))

model.add(BatchNormalization())

#2 dense layer
model.add(Dense(4096,activation="relu"))
```

```
model.add(Dropout(0.4))

model.add(BatchNormalization())

#3 dense layer
model.add(Dense(1000,activation="relu"))

model.add(Dropout(0.4))

model.add(BatchNormalization())

#output layer
model.add(Dense(20,activation="softmax"))

model.summary()
model.compile(optimizer="adam",
loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
model.fit(x_train, y_train, epochs=100)
loss, accuracy = model.evaluate(x_test, y_test)

print(loss,accuracy)
pred = model.predict(x_test)

pred.shape
plt.figure(1 , figsize = (19 , 10))
n = 0

for i in range(9):
   n += 1
```

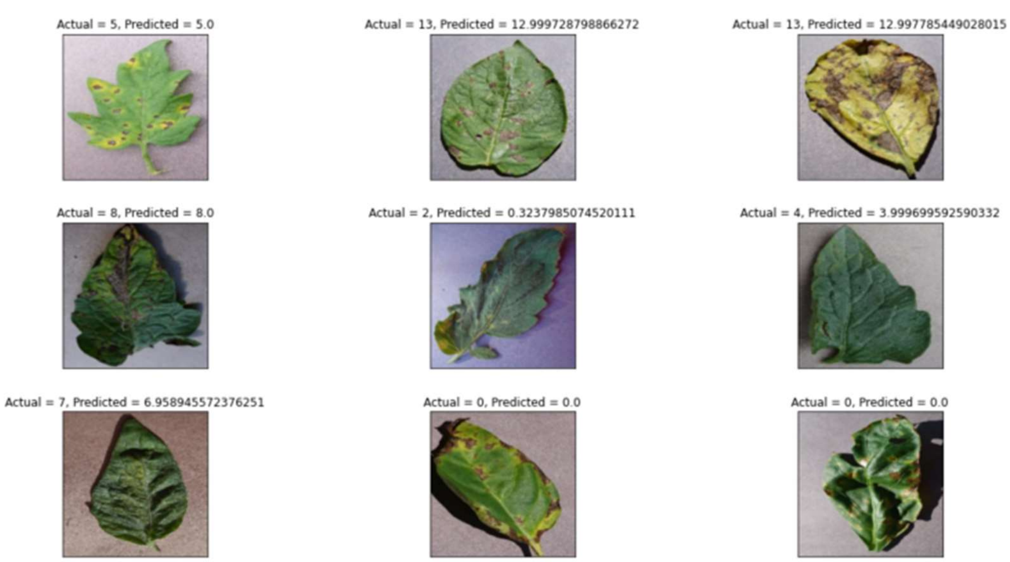
```
    r = np.random.randint( 0, x_test.shape[0], 1)

    plt.subplot(3, 3, n)
    plt.subplots_adjust(hspace = 0.3, wspace = 0.3)

    plt.imshow(x_test[r[0]])
    plt.title('Actual = {}, Predicted =
{}'.format(y_test[r[0]] ,
y_test[r[0]]*pred[r[0]][y_test[r[0]]]) )
    plt.xticks([]) , plt.yticks([])

plt.show()
```

Result and Discussion:-



Learning outcomes (What I have learnt):

- We have learned about how convolution works.
- We have learned about how max poling works.