**Experiment No. 3 : Plant Disease prediction using CNN**

import numpy as np  
import pickle  
import cv2  
from os import listdir  
from sklearn.preprocessing import LabelBinarizer  
from keras.models import Sequential  
from keras.layers.normalization import Batch Normalization  
from keras.layers.convolutional import Conv2D  
from keras.layers.convolutional import MaxPooling2D  
from keras.layers.core import Activation, Flatten, Dropout, Dense  
from keras import backend as K  
from keras.preprocessing.image import ImageDataGenerator  
from keras.optimizers import Adam  
from keras.preprocessing import image  
from keras.preprocessing.image import img\_to\_array  
from sklearn.preprocessing import MultiLabelBinarizer  
from sklearn.model\_selection import train\_test\_split  
import matplotlib.pyplot as plt  
import tensorflow

EPOCHS = 25  
INIT\_LR= 1e-3  
BS = 32  
default image\_size = tuple((256, 256))  
image\_size = 0  
directory\_root ='./input/plantvillage/’  
width=256  
height=256  
depth=3

def convert\_image\_to\_array(image\_dir):

try:  
image = cv2.imread(image\_dir)  
if image is not None:  
image = cv2.resize (image, default\_image\_size)  
return img\_to\_array(image)  
else:  
return np.array ([])  
except Exception as e:  
print(f“Error : {e}")  
  
return None  
image\_list, label\_list = [],[]

try:  
print("[INFO] Loading images..”)  
root\_dir = listdir (directory\_root)

for directory in root\_dir :  
# remove .DS\_Store from list  
if directory == ".DS\_Store":  
root\_dir.remove (directory)

for plant\_folder in root\_dir :  
plant\_disease\_folder\_list = listdir(f" {directory\_root}/(plant\_folder}”)

for disease\_folder in plant\_disease\_folder list:  
# remove .DS\_Store from list  
if disease folder == ".DS\_Store" :  
plant\_disease \_folder\_list.remove (disease\_folder)

for plant\_disease\_folder in plant\_disease\_folder\_list:  
print(f”[INFO] Processing {plant disease\_folder}..")  
plant\_disease\_image\_list = listdir(f"{directory\_root}/{plant\_folder}/{plant\_disease\_folder} /")

for single\_plant\_disease\_image in plant\_disease\_image\_list :  
if single\_plant\_disease\_image == ".DS\_ Store":  
plant\_disease\_image\_list.remove(single\_plant\_disease\_image)

for image in plant\_disease\_image\_list[:200]:  
image\_directory = f”{directory\_root}/{plant, folder}/{plant\_disease\_folder}/{image}"  
if image\_directory.endswith (".jpg") == True or image\_directory.endswith(".JPG") == True:  
Image\_list.append (convert\_image\_to\_array(image\_directory))

label\_list.append (plant\_disease\_folder)  
print("[INFO] Image loading completed")  
except Exception as e:  
print(f”Error:{e}")

image\_size = len(image\_list)

label\_binarizer = LabelBinarizer()  
image\_labels = label\_binarizer.fit\_transform(label\_list)  
pickle.dump(label\_binarizer,open('label\_transform.pkl', 'wb'))  
n\_classes = len(label\_binarizer.classes\_)

print(label\_binarizer.classes)

np\_image\_list = np.array (image\_list, dtype=np.float16) / 225.0

print("[NFO] Spliting data to train, test")  
X\_train, x\_test, y\_train, y\_test = train\_test\_split(np\_image\_list, image\_labels, test\_size=0.2, random state = 42)

aug = ImageDataGenerator(  
rotation\_range=25, width\_shift\_range=0.1,  
height\_shift\_range=0.1, shear\_range=0.2,  
zoom\_range=0.2,horizontal\_flip=True,fill mode="nearest")

model = Sequential()  
inputShape = (height, width, depth)  
 chanDim=-1  
if K.image\_data\_format() == "channels\_first":  
inputShape = (depth, height, width)  
 chan Dim=1  
model.add (Conv2D(32, (3, 3), padding="same",input\_shape=inputShape))  
model.add(Activation("relu'"))  
model.add (Batch Normalization(axis=chanDim))  
model.add (MaxPooling2D(pool\_size=(3, 3)))  
model.add (Dropout(0.25))  
model.add (Conv2D (64, (3, 3), padding="'same'"))  
model.add (Activation(“relu"))  
model.add(BatchNormalization(axis=chanDim))  
model.add(Conv2D(64,(3, 3),padding="'same"))

model.add(Activation(“relu")  
model.add (Batch Normalization(axis=chanDim))  
model.add(MaxPooling2D(poo\_size=(2, 2)))  
model.add(Dropout(0.25))  
model.add(Conv2D(128, (3, 3),. padding="'same"))  
model.add(Activation("relu"))  
model.add(BatchNormalization(axis=chan Dim))  
model.add(Conv2D (128, (3, 3),padding="same"))  
model.add(Activation(“relu"))  
model.add(BatchNormalization(axis=chanDim))  
model.add(MaxPooling2D(poolsize=(2, 2))  
model.add(Dropout(0.25))  
model.add(Flatten())  
model.add(Dense(1024))  
model.add(Activation("relu"))  
model.add(BatchNormalization())  
model.add(Dropout(0.5))  
model.add (Dense(n\_classes))  
model.add (Activation("'softmax"))

opt= Adam(Ir=INIT\_LR, decay=INIT\_LR / EPOCHS)  
# distribution  
model.compile(loss="binary\_crossentropy", optimizer=opt,metrics=["accuracy"])  
# train the network  
print("[INFO] training network..")

history = model.fit \_generator(  
aug.flow(x\_train, y\_train, batch \_size=BS),  
validation\_data=(x test, y\_test),  
steps\_per\_epoch=len(x\_train) // BS,  
epochs=EPOCHS, verbose=1

)  
acc = history.history['acc']  
val\_acc = history. history['val\_acc']  
loss = history.history['loss']  
val \_loss = history.history ['val\_loss']  
epochs = range(1, len(acc) + 1)

#Train and validation accuracy  
plt.plot(epochs, acc, 'b', label=’Training accuracy')

plt.plot(epochs, val\_acc, 'r', label=’Validation accurarcy')  
plt.title(‘Training and Validation accurarcy')  
plt.legend()

plt.figure()  
#Train and validation loss  
plt.plot(epochs, loss, 'b', label=’Training loss’)  
plt.plot(epochs, val\_loss, 'r, label=’Validation loss')  
plt.title(‘Training and Validation loss')  
plt.legend()  
plt.show()

print("[INFO] Calculating model accuracy")  
scores = model.evaluate(x\_test, y\_test)  
print(f"Test Accuracy: {scores[1]\*100}")

# save the model to disk  
print("[INFO] Saving model..")  
pickle.dump(model,open('cnn\_model.pkl', 'wb')

Accuracy: 96.77%

[INFO] Calculating model accuracy  
591/591[=====-===] - 2s 3ms/step  
Test Accuracy : 96.773830807551 92