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
```