Experiment No. 3 : Plant Disease prediction using CNN (PART 1)

```
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(lr=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
```

Experiment No. 3 : Classification of MNIST Fashion dataset using CNN(PART 2)

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from keras.utils import to_categorical
from matplotlib.pyplot import figure, show
import warnings
import seaborn as sns
warnings.flterwarnings(ignore')
import matplotlib.style as style
```

```
from sklearn.model_selection import train_test_split
from keras.layers import Input, Concatenate, concatenate, Dense, Embedding, Dropout,
Conv2D, MaxPooling2 D
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, History
from keras.layers import Dropout, Flatten, GlobalAveragePooling2D, Activation

from sklearn,preprocessing import LabelEncoder, OneHotEncoder
from keras.preprocessing.image import imageDataGenerator
from sklearn.model_selection import train_test_split
from keras.applications.resnet50 import ResNet50
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint
from keras.applications.vggl6 import VGG16
from keras.utils import to_categorical
from sklearn.utils import class_weight
from keras.layers.normalization import BatchNormalization
from matplotlib import pyplot as plt
from keras import backend as K
from keras.optimizers import SGD
from keras.models import Model
import seaborn as sns
import numpy as np
import argparse
import time
import glob
import cv2
import numpy
import os
```

```
import glob

ímport sys

import os

import json

import pprint

import warnings

warnings.flterwarnings('ignore')


#Load Data

!curl -L -0 https://www.dropbox.com/s/heyqll2my8uwotq/fashionmaistzip

!unzip fashionmnist:zip


#Load training and test data using dataframes from Pandas.

train = pd. read_csv("fashion-mnist_train.csv")

test = pd.read csv("fashion-mnist_test.csv'")


Img_rows, img_cols = 28, 28

input_shape = (img_rows, img_cols, 1)

X= train.iloc[:,1:]

Y= traln.iloc[:,:1]

X_ test = test.iloc[:, 1:]

Y_test = test.iloc[:,: 1]


#Normalization

X= np.asarray(X).reshape (X.shape [0], img_rows,img_cols, 1)

X_test = np.asarray(X_ test).reshape(X_test.shape [0], img_rows,ímg_cols,1)
```

```
X= (255. - X) /255.
X_test = (255. - X_test) / 255.

#Number of classes
classes = len(Y['label'],value _counts())

print("Number of features: ", X.shape[1])
print("Number of train samples: ", Xshape [0])
print("Number of test samples: ", X test.shape [0])
OUT:
Number of features: 28
Number of train samples: 60000
Number of test samples: 10000

#Training
Y_test = to_categorical(Y_test)
Y= to_categorical (Y)
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, stratify=Y, test_size=0.2,
random_state=66)
Irr = ReducelLROn Plateau (monitor='val_loss', factor=0.1, patience=2, verbose=1,
epsilon=1e-3, mode= ‘min')
early_stopping = EarlyStopping(monitor='val loss',patience=5,verbose=0, mode='auto’)
checkpoint = ModelCheckpoint("checkpoint.hdf5", monitor='val_acc', verbose=1,
save_best_only=True, mode='max’)
batch size = 64
epochs = 10
```

```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

# define the grid search parameters
batch_size = [16, 32, 64, 80]
epochs = [10, 25, 50]
param_grid = dict (batch_size=batch_size, epochs=epochs)
model = KerasClassifier(build_fn=model_basic, verbose=0)
Grid=GridSearchCV(estimator=model, param_grid=param_grid,n_jobs1, cv=3)
grid_result = grid.fit(X_train, Y_train)

# summarize results
print("Best: %using %s" % (grid_result.best_score, grid_result.bestparams_))
means = grid_result.cv_results ['mean_test_score']
stds = grid_result.cv_results ['std_test_score']
params =grid_result.cv_results ['params']
for mean, stdev, param in zip(means, stds, params):
print("%f (%f) with: %r" % (mean, stdev, param))

#Model
def model_basic(classes=classes,optimizer='adam'):
kernel_size = (3,3)
dropout = 0.25
pool_size = (2,2)
inputs = Input(shape=(img_rows, img_cols, 1))
```

```
y= Conv2D (filters=32, kernel_size=kernel_size,activation='relu',padding='same’) (inputs)
y= MaxPooling2D(pool_size=pool_size,strides=(2,2)) (y)
y= Dropout(dropout)(y)

y= Flatten()(y)

y= Dense(256,activation=’relu')(y)
y= BatchNormalization()(y)
y= Dropout(dropout)(y)
outputs = Dense(classes, activation='softmax’)(y)

model = Model(inputs=inputs, outputs=outputs)
model.compile (optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'|)
return model
basic_model = model_basic()

import warnings
warnings.filterwarnings('ignore')
history = basic_model.fit(X_train, Y_train, batch size=batch_size, epochs=epochs,
verbose=1, validation_data=(X_val, Y_val)

OUT:
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [=======]-82s Zms/step- loss: 0.4390 - acc:.8470 - val_loss: 0.4019val _acc:
0.8618
```

Epoch 2/10

48000/48000 [=========]-82s 2ms/step - loss: 0.3458 - acc: 0.8776 - val_loss: 0.3046

val_acc: 0.8932

Epoch 3/10

48000/48000 [========] -82s 2ms/step - loss: 0.3110 - acc: 0.8883 - val_loss: 0.2947 -

val_acc: 0.8953

Epoch 4/10

48000/48000 [===]-81s 2ms/step - loss: 0.2935 - acc: 0.8937 - val_loss: 0.2772 -

val_acc: 0.9024

Epoch 5/10

48000/48000 [=========]-85s 2ms/step- loss: 0.2710 - acc: 0.9030 - val_loss: 0.2855-

val_acc: 0.8952

Epoch 6/10[==========| -84s 2ms/step - loss: 0.2592 - acc: 0.9067 - val _loss: 0.2574-

val_acc: 0.9063

Epoch 7/10

48000/48000 [==========]-85s 2ms/step-loss: 0.2450 - acc: 0.9107 - val_loss: 0.2773 -

val acc: 0.8998

Epoch 8/10

48000/48000 [=======] - 84s 2ms/step - loss: 0.2338 - acc: 0.9147 – val_loss: 0.2833 -

val_acc: 0.8955

Epoch 9/10

48000/48000 [========]-82s 2ms/step - loss: 0.2239 - acc: 0.9174 – val_loss: 0.2553 -

val_acc: 0.9127

Epoch 10/10

48000/48000 [========]-84s 2ms/step - loss: 0.2153 - acc: 0.9207 - val loss: 0.2564 -

val_acc: 0.9123

score = basic_model.evaluate(X_test,Y_test, verbose=0)

```
print("Test loss:", score[0])

print("Test accuracy:', score[1])
```

OUT:

```
Test loss: 0.2463475521683693

Test accuracy: 0.9137
```