from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

from tkinter.filedialog import askopenfilename

import pandas as pd

import random

import numpy as np

from keras.utils.np\_utils import to\_categorical

from keras.layers import MaxPooling2D

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Convolution2D

from keras.models import Sequential

from keras.models import model\_from\_json

import pickle

import os

from sklearn import preprocessing

from keras.optimizers import SGD

import matplotlib.pyplot as plt

from numpy import dot

from numpy.linalg import norm

main = tkinter.Tk()

main.title("Neural-Network-Based Root Mean Delay Spread Model for Ubiquitous Indoor Internet-of-Things Scenarios")

main.geometry("1300x1200")

global filename

global classifier

global X, Y

global rms

def uploadDataset():

global filename

filename = filedialog.askdirectory(initialdir=".")

pathlabel.config(text=filename)

text.delete('1.0', END)

text.insert(END,filename+" loaded\n\n");

def preprocessDataset():

text.delete('1.0', END)

global X, Y

X = []

Y = []

for root, dirs, directory in os.walk(filename):

for j in range(len(directory)):

name = os.path.basename(root)

dataset = pd.read\_csv(root+"/"+directory[j],sep=";")

dataset = dataset.values

dataset = dataset[:,0:(dataset.shape[1]-1)]

#dataset = preprocessing.normalize(dataset)

if name == 'Lab139\_7.1':

Y.append(1)

X.append(dataset)

if name == 'Corridor\_rm155\_7.1':

X.append(dataset)

Y.append(0)

X = np.asarray(X)

Y = np.asarray(Y)

print(X.shape)

indices = np.arange(X.shape[0])

np.random.shuffle(indices)

X = X[indices]

Y = Y[indices]

Y1 = to\_categorical(Y)

text.insert(END,"Total files founds in dataset is : "+str(X.shape[0])+"\n")

text.insert(END,"Total vector values founds in each file is : "+str(X.shape[1])+"\n")

text.insert(END,"Measurements vector available for Corridor & LAB area\n")

X = X.reshape(40,25,24,5)

print(X.shape)

def buildModel():

global rms

global classifier

text.delete('1.0', END)

if os.path.exists('model/model.json'):

with open('model/model.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

classifier = model\_from\_json(loaded\_model\_json)

classifier.load\_weights("model/model\_weights.h5")

classifier.make\_predict\_function()

print(classifier.summary())

f = open('model/history.pckl', 'rb')

data = pickle.load(f)

rms = data

f.close()

loss = data['loss']

loss = loss[9]

text.insert(END,"Neural Network RMS Delay found from Measurement = "+str(loss)+"\n")

else:

classifier = Sequential()

classifier.add(Convolution2D(32, 3, 3, input\_shape = (25, 24, 5), activation = 'relu')) #defining input layer with 25 X 24 X 5 size with 32 filters

classifier.add(MaxPooling2D(pool\_size = (2, 2))) #defining output layer of 32 filter

classifier.add(Convolution2D(32, 3, 3, activation = 'relu')) #defining hidden layer with 32 filters

classifier.add(MaxPooling2D(pool\_size = (2, 2))) #defining output for hidden layer

classifier.add(Flatten())

classifier.add(Dense(output\_dim = 256, activation = 'relu')) #output layer with 256 neurons

classifier.add(Dense(output\_dim = 2, activation = 'sigmoid')) #prediction output of two classes

print(classifier.summary())

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['mse'])#compiling neural network and asking to calculate rmse

hist = classifier.fit(X, Y, batch\_size=2, epochs=10, shuffle=True, verbose=2)#fitting or training neural network with given X input data and Y predicted output

classifier.save\_weights('model/model\_weights.h5')

model\_json = classifier.to\_json()

with open("model/model.json", "w") as json\_file:

json\_file.write(model\_json)

f = open('model/history.pckl', 'wb')

pickle.dump(hist.history, f)

f.close()

f = open('model/history.pckl', 'rb')

data = pickle.load(f)

rms = data

f.close()

loss = data['loss']

loss = loss[9]

text.insert(END,"Neural Network RMS Delay found from Measurement = "+str(loss)+"\n")

def predict():

temp = []

text.delete('1.0', END)

filename = filedialog.askopenfilename(initialdir="testSamples")

test = pd.read\_csv(filename,sep=";")

test = test.values

test = test[:,0:(test.shape[1]-1)]

XX = np.reshape(X, (X.shape[0],(X.shape[1]\*X.shape[2]\*X.shape[3])))

temp = np.asarray(test)

temp = temp.ravel()

count1 = 0

count2 = 0

for i in range(len(XX)):

predict\_score = dot(XX[i], temp)/(norm(XX[i])\*norm(temp))

if i < 10 and predict\_score < 1.0:

count1 = count1 + 1

if i>=10 and predict\_score < 1.0:

count2 = count2 + 1

if count1 < 5:

text.insert(END,"Input file vector measurement suitable for CORRIDOR AREA\n")

else:

text.insert(END,"Input file vector measurement suitable for LAB AREA\n")

def delayGraph():

delay = rms['loss']

plt.figure(figsize=(10,6))

plt.grid(True)

plt.xlabel('Iterations')

plt.ylabel('RMS Delay')

plt.plot(delay, 'ro-', color = 'green')

plt.legend(['RMS Delay'], loc='upper left')

#plt.xticks(wordloss.index)

plt.title('RMS Delay Graph')

plt.show()

font = ('times', 15, 'bold')

title = Label(main, text='Neural-Network-Based Root Mean Delay Spread Model for Ubiquitous Indoor Internet-of-Things Scenarios',anchor=W, justify=CENTER)

title.config(bg='yellow4', fg='white')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 13, 'bold')

upload = Button(main, text="Upload Vector Network Analyzer Dataset", command=uploadDataset)

upload.place(x=50,y=100)

upload.config(font=font1)

pathlabel = Label(main)

pathlabel.config(bg='yellow4', fg='white')

pathlabel.config(font=font1)

pathlabel.place(x=50,y=150)

processButton = Button(main, text="Preprocess Dataset", command=preprocessDataset)

processButton.place(x=50,y=200)

processButton.config(font=font1)

buildButton = Button(main, text="Build RMS Neural Network Model", command=buildModel)

buildButton.place(x=50,y=250)

buildButton.config(font=font1)

graphButton = Button(main, text="RMS Delay Graph", command=delayGraph)

graphButton.place(x=50,y=300)

graphButton.config(font=font1)

#predictButton = Button(main, text="Predict Environment from Measurement", command=predict)

#predictButton.place(x=50,y=350)

#predictButton.config(font=font1)

font1 = ('times', 12, 'bold')

text=Text(main,height=15,width=78)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=450,y=100)

text.config(font=font1)

main.config(bg='magenta3')

main.mainloop()