# -\*- coding: utf-8 -\*-

"""luna.ipynb

Automatically generated by Colaboratory.

Original file is located at colab

from google.colab import drive

drive.mount('/content/drive')

!wget "https://zenodo.org/record/3723295/files/seg-lungs-LUNA16.zip"

!unzip "/content/seg-lungs-LUNA16.zip"

#increase ram

#d=[]

#while(1)

# d.append('1')

root\_path = '/content/drive/My Drive/luna'

!pip install SimpleITK

import SimpleITK as sitk

import numpy as np

import csv

import os

from PIL import Image

from glob import glob

import matplotlib.pyplot as plt

from tqdm import tqdm

from random import shuffle

import pandas as pd

from tensorflow import keras as k

from tensorflow.keras.models import Sequential ,Model

from tensorflow.keras.layers import Conv2D, MaxPooling2D,AveragePooling3D,AveragePooling2D,MaxPooling3D,Conv3D, add

from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense ,Input ,BatchNormalization, GlobalAveragePooling3D

from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler

from tensorflow.keras.preprocessing import image

from tensorflow.keras import applications, regularizers

cand\_path = os.path.join(root\_path,'/content/drive/My Drive/sorted.csv')

#cand\_path = os.path.join(root\_path,'/content/drive/My Drive/sorted.csv')

cands = pd.read\_csv(cand\_path , header=0)

#cands = pd.read\_csv(cand\_path , header=0)

cands.head()

cands["class"].shape

cands.shape[0]/len(np.unique(cands['LNDbID'].values))

def load\_itk\_image(filename):

itkimage = sitk.ReadImage(filename)

numpyImage = sitk.GetArrayFromImage(itkimage)

numpyOrigin = np.array(list(reversed(itkimage.GetOrigin())))

numpySpacing = np.array(list(reversed(itkimage.GetSpacing())))

return numpyImage, numpyOrigin, numpySpacing

def readCSV(filename):

lines = []

with open(filename, "rb") as f:

csvreader = csv.reader(f)

for line in csvreader:

lines.append(line)

return lines

def worldToVoxelCoord(worldCoord, origin, spacing):

stretchedVoxelCoord = np.absolute(worldCoord - origin)

voxelCoord = stretchedVoxelCoord / spacing

return voxelCoord

def normalizePlanes(npzarray):

maxHU = 400.

minHU = -1000.

npzarray = (npzarray - minHU) / (maxHU - minHU)

npzarray[npzarray>1] = 1.

npzarray[npzarray<0] = 0.

return npzarray

voxelWidth = 64

train\_paths = []

for i in range(1,9):

train\_paths += glob(os.path.join(os.path.join(root\_path, 'subset'+str(i)), "\*.mhd"))

val\_paths = glob(os.path.join(os.path.join(root\_path, 'subset0'), "\*.mhd")) + glob(os.path.join(os.path.join(root\_path, 'subset9'), "\*.mhd"))

numpyImage, numpyOrigin, numpySpacing = load\_itk\_image(train\_paths[0])

def get\_batches\_data(paths, voxelWidth, batch\_size):

while True:

count = 0

first = True

for path in paths:

numpyImage, numpyOrigin, numpySpacing = load\_itk\_image(path)

for cand in (cands.values):

worldCoord = np.asarray([float(cand[3]),float(cand[2]),float(cand[1])])

voxelCoord = worldToVoxelCoord(worldCoord, numpyOrigin, numpySpacing)

nod\_x = numpyImage[int(voxelCoord[0]-voxelWidth/2):int(voxelCoord[0]+voxelWidth/2),

int(voxelCoord[1]-voxelWidth/2):int(voxelCoord[1]+voxelWidth/2),

int(voxelCoord[2]-voxelWidth/2):int(voxelCoord[2]+voxelWidth/2)]

nod\_x = normalizePlanes(nod\_x)

if nod\_x.shape == (voxelWidth,voxelWidth,voxelWidth):

nod\_x = np.expand\_dims(nod\_x,axis=3)

if(cand[4]==0):

nod\_y = np.array([1,0])

#weight = 0.5

else:

nod\_y = np.array([0,1])

#weight = 0.5

if first:

X\_train = np.ones([batch\_size, voxelWidth, voxelWidth, voxelWidth, 1])

X\_train[count,:,:,:,:] = nod\_x

Y\_train = np.ones([batch\_size, 2])

Y\_train[count,:] = nod\_y

#loss\_weight = np.ones([batch\_size, 1])

#loss\_weight[count,:] = weight

first = False

count += 1

else:

X\_train[count,:,:,:,:] = nod\_x

Y\_train[count,:] = nod\_y

#loss\_weight[count,:] = weight

count +=1

if count >= batch\_size :

count = 0

first = True

yield (X\_train, Y\_train)

else:

continue

if count>0 and count < batch\_size:

yield (X\_train[:count], Y\_train[:count])

#train\_gen= get\_batches\_train(nodule\_path,non\_nodule\_path, 64, 16 )

#val\_gen= get\_batches\_val(nodule\_path,non\_nodule\_path, 64, 16 )

#from numpy import asarray

#from numpy import save

#from numpy import load

# save to npy file

#save('/content/drive/My Drive/xtrainlndb.npy', xtrain)

#save('/content/drive/My Drive/ytrainlndb.npy', ytrain)

#xtraintrain = load ('/content/drive/My Drive/xtraintrain.npy')

#ytraintrain = load ('/content/drive/My Drive/ytraintrain.npy')

#xtrainval = load ('/content/drive/My Drive/xtrainval.npy')

#ytrainval = load ('/content/drive/My Drive/ytrainval.npy')

#xtrainlndb = load ('/content/drive/My Drive/xtrainlndb.npy')

#ytrainlndb = load ('/content/drive/My Drive/ytrainlndb.npy')

#xtrainlndbval = load ('/content/drive/My Drive/xtrainlndbval.npy')

#ytrainlndbval = load ('/content/drive/My Drive/ytrainlndbval.npy')

#xtrain = np.vstack((xtrainlndb, xtrainlndbval))

#ytrain = np.vstack((ytrainlndb, ytrainlndbval))

gen = get\_batches\_data(train\_paths, 64, 16)

count = 0

for element in gen:

print(element[0].shape, element[1].shape)

break

#Lnet

model = Sequential()

model.add(Conv3D(32, kernel\_size=(5, 5, 5), strides=(1, 1,1),

activation='relu',

input\_shape=[64,64,64,1]))

model.add(Dropout(0.4))

model.add(MaxPooling3D(pool\_size=(2, 2,2), strides=(2, 2,2)))

model.add(Conv3D(64, (5, 5,5), activation='relu'))

model.add(Dropout(0.4))

model.add(MaxPooling3D(pool\_size=(2, 2,2)))

model.add(Flatten())

model.add(Dropout(0.4))

model.add(Dense(1000, activation='relu'))

model.add(Dropout(0.4))

model.add(Dense(2, activation='softmax'))

model.summary()

model.compile(loss=k.losses.categorical\_crossentropy,

optimizer=k.optimizers.SGD(lr=0.001),

metrics=['accuracy'])

model\_checkpoint = ModelCheckpoint('/content/drive/My Drive/lenetexcel.hdf5', monitor='loss', save\_best\_only=True)

model.load\_weights('/content/drive/My Drive/lenetexcel.hdf5')

batch\_size = 20

train\_gen = get\_batches\_data(train\_paths, voxelWidth, batch\_size)

val\_gen = get\_batches\_data(val\_paths, voxelWidth, batch\_size)

historylenet= model.fit(train\_gen, steps\_per\_epoch=813, epochs=30, verbose=1, validation\_data= val\_gen,

validation\_steps=200, callbacks=[model\_checkpoint], shuffle= True)

# summarize history for accuracy

plt.plot(acc)

plt.plot(valacc)

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(loss)

plt.plot(valoss)

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

len(valoss)

#new

def makeNew():

Sequential()

nClasses = 2

x = Input(shape=(64,64,64,1))

conv1= Conv3D(32, (3,3,3), padding='same', activation= 'relu')(x)

conv1= Conv3D(64, (3,3,3), padding='same', activation= 'relu')(conv1)

max1 = MaxPooling3D(pool\_size=(2,2,2), strides= None, padding='valid')(conv1)

conv2= Conv3D(128, (3,3,3), padding='same', activation= 'relu')(max1)

conv2= Conv3D(128, (3,3,3), padding='same', activation= 'relu')(conv2)

max2 = MaxPooling3D(pool\_size=(2,2,2), strides= None, padding='valid')(conv2)

conv3= Conv3D(256, (3,3,3), padding='same', activation= 'relu')(max2)

conv3= Conv3D(256, (3,3,3), padding='same', activation= 'relu')(conv3)

conv3= Conv3D(256, (3,3,3), padding='same', activation= 'relu')(conv3)

max3 = MaxPooling3D(pool\_size=(2,2,2), strides= None, padding='valid')(conv3)

conv4= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(max3)

conv4= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(conv4)

conv4= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(conv4)

max4 = MaxPooling3D(pool\_size=(2,2,2), strides= None, padding='valid')(conv4)

conv5= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(max4)

conv5= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(conv5)

conv5= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(conv5)

max5 = MaxPooling3D(pool\_size=(2,2,2), strides= None, padding='valid')(conv5)

conv6= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(max5)

conv6= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(conv6)

conv6= Conv3D(512, (3,3,3), padding='same', activation= 'relu')(conv6)

max6 = MaxPooling3D(pool\_size=(2,2,2), strides= None, padding='valid')(conv6)

flat = Flatten()(max6)

dense= Dense(4096,activation="relu")(flat)

flat= Dropout(0.5)(dense)

dense= Dense(4096,activation="relu")(flat)

flat= Dropout(0.5)(dense)

dense= Dense(nClasses, activation="softmax")(flat)

New = Model(inputs=x, outputs=dense)

New.compile(loss=k.losses.categorical\_crossentropy,

optimizer=k.optimizers.Adam(lr=0.001),

metrics=['accuracy'])

return New

New = makeNew()

New.summary()

model\_checkpoint = ModelCheckpoint('/content/drive/My Drive/New.hdf5', monitor='loss', save\_best\_only=True)

New.load\_weights('/content/drive/My Drive/New.hdf5')

historyNew= New.fit(xtrain, ytrain, batch\_size= 16, epochs=30, verbose=1,

validation\_split=0.2, callbacks=[model\_checkpoint])

def inceptionlayer(prev):

tower\_1 = Conv3D(64, (1,1,1), padding='same', activation='relu', activity\_regularizer= regularizers.l2(0.001))(prev)

tower\_1 = Conv3D(64, (3,3,3), padding='same', activation='relu', activity\_regularizer= regularizers.l2(0.001))(tower\_1)

tower\_2 = Conv3D(64, (1,1,1), padding='same', activation='relu', activity\_regularizer= regularizers.l2(0.001))(prev)

tower\_2 = Conv3D(64, (5,5,5), padding='same', activation='relu', activity\_regularizer= regularizers.l2(0.001))(tower\_2)

tower\_3 = MaxPooling3D((3,3,3), strides=(1,1,1), padding='same')(prev)

tower\_3 = Conv3D(64, (1,1,1), padding='same', activation='relu', activity\_regularizer= regularizers.l2(0.001))(tower\_3)

output = k.layers.concatenate([tower\_1, tower\_2, tower\_3], axis = 3)

return output

#GOOGLENET

def makegooglenet():

x = Input(shape=(64,64,64,1))

conv1= Conv3D(32,kernel\_size=(7,7,7),activation='relu', activity\_regularizer= regularizers.l2(0.001))(x)

conv1= Dropout(0.3)(conv1)

max1 = MaxPooling3D(pool\_size=(2,2,2))(conv1)

conv2= Conv3D(32,kernel\_size=(3,3,3),activation='relu', activity\_regularizer= regularizers.l2(0.001))(max1)

conv2= Dropout(0.3)(conv2)

max2 = MaxPooling3D(pool\_size=(2,2,2))(conv2)

incp1= inceptionlayer(max2)

incp1= Dropout(0.3)(incp1)

#incp2= inceptionlayer(incp1)

#incp2= Dropout(0.3)(incp2)

#max3 = MaxPooling3D(pool\_size=(2,2,2))(incp2)

#incp3= inceptionlayer(max3)

#incp3= Dropout(0.3)(incp3)

#incp4= inceptionlayer(incp3)

#incp4= Dropout(0.3)(incp4)

#max4 = MaxPooling3D(pool\_size=(2,2,2))(incp4)

#incp5= inceptionlayer(max4)

#incp5= Dropout(0.3)(incp5)

#incp6= inceptionlayer(incp5)

#incp6= Dropout(0.3)(incp6)

avg1= AveragePooling3D(pool\_size=(2,2,2))(incp1)

flat = Flatten()(avg1)

flat= Dropout(0.3)(flat)

dense= Dense(2,activation="softmax", activity\_regularizer= regularizers.l2(0.001))(flat)

googlenet = Model(inputs=x, outputs=dense)

googlenet.compile(loss=k.losses.categorical\_crossentropy,

optimizer=k.optimizers.Adam(lr=0.00001),

metrics=['accuracy'])

return googlenet

googlenet = makegooglenet()

googlenet.summary()

model\_checkpoint = ModelCheckpoint('/content/drive/My Drive/googlenet.hdf5', monitor='loss', save\_best\_only=True)

googlenet.load\_weights('/content/drive/My Drive/googlenet.hdf5')

batch\_size = 32

val\_gen = get\_batches\_data(val\_paths, voxelWidth, batch\_size)

train\_gen = get\_batches\_data(train\_paths, voxelWidth, batch\_size)

historygoogle= googlenet.fit(train\_gen, steps\_per\_epoch=813, epochs=30, verbose=1, validation\_data= val\_gen,

validation\_steps=91, callbacks=[model\_checkpoint], shuffle = True)

plotmymodel(historygoogle)

print(historygoogle.history)

#model.load\_weights('weights3d\_lnet.hdf5')

histroylenet= model.fit(xtrain, ytrain,

batch\_size=1000,

epochs=30,

verbose=1,

validation\_split=0.1,

callbacks=[model\_checkpoint])

model.save\_weights('weights3d\_new.hdf5')

model.summary()

from sklearn.metrics import confusion\_matrix

predicted = vanilla3d.predict(train\_gen)

pre = np.argmax(predicted,axis=1)

tru = np.argmax(train\_gen,axis=1)

confusion\_matrix(tru, pre)

del xtrain

del ytrain

xtrainval.shape

#trial

def maketrial():

x = Input(shape=(64,64,64,1))

conv0= Conv3D(32,kernel\_size=(3,3,3),padding="same",activation='sigmoid')(x)

conv0= Dropout(0.7)(conv0)

avg1 = AveragePooling3D(pool\_size=(2,1,1))(conv0)

bat0= BatchNormalization()(avg1)

conv1= Conv3D(32,kernel\_size=(3,3,3),padding="same",activation='sigmoid')(bat0)

conv1= Dropout(0.7)(conv1)

max1 = MaxPooling3D(pool\_size=(2,2,2))(conv1)

bat1= BatchNormalization()(max1)

conv2= Conv3D(64,kernel\_size=(3,3,3),padding="same",activation='sigmoid')(bat1)

conv2= Dropout(0.7)(conv2)

max2 = MaxPooling3D(pool\_size=(2,2,2))(conv2)

bat2= BatchNormalization()(max2)

conv3= Conv3D(128,kernel\_size=(3,3,3),padding="same",activation='sigmoid')(bat2)

flat = Flatten()(conv3)

flat= Dropout(0.7)(flat)

dense= Dense(2,activation="sigmoid")(flat)

flat= Dropout(0.7)(dense)

dense= Dense(2,activation="softmax")(flat)

trialmodel = Model(inputs=x, outputs=dense)

trialmodel.compile(loss=k.losses.categorical\_crossentropy,

optimizer=k.optimizers.Adamax(lr= 0.01),

metrics=['accuracy'])

return trialmodel

trial = maketrial()

trial.summary()

model\_trial = ModelCheckpoint('/content/drive/My Drive/trialluna.hdf5', monitor='loss', save\_best\_only=True)

trial.load\_weights('/content/drive/My Drive/trialluna.hdf5')

historytrial= trial.fit(xtrain, ytrain, batch\_size= 16, epochs=30, verbose=1,

validation\_split=0.2, callbacks=[model\_trial], shuffle=True)

len(xtrain)

#Vanilla3d

def makevanilla():

x = Input(shape=(64,64,64,1))

conv0= Conv3D(32,kernel\_size=(3,3,3),padding="same",activation='sigmoid', activity\_regularizer= regularizers.l2(0.05))(x)

conv0= Dropout(0.4)(conv0)

avg1 = AveragePooling3D(pool\_size=(2,1,1))(conv0)

conv1= Conv3D(32,kernel\_size=(3,3,3),padding="same",activation='sigmoid', activity\_regularizer= regularizers.l2(0.05))(avg1)

conv1= Dropout(0.4)(conv1)

max1 = MaxPooling3D(pool\_size=(2,2,2))(conv1)

conv2= Conv3D(64,kernel\_size=(3,3,3),padding="same",activation='sigmoid', activity\_regularizer= regularizers.l2(0.05))(max1)

conv2= Dropout(0.4)(conv2)

max2 = MaxPooling3D(pool\_size=(2,2,2))(conv2)

conv3= Conv3D(128,kernel\_size=(3,3,3),padding="same",activation='sigmoid', activity\_regularizer= regularizers.l2(0.05))(max2)

#conv3= Dropout(0.2)(conv3)

#max3 = MaxPooling3D(pool\_size=(2,2,2))(conv3)

#bat3= BatchNormalization()(max3)

#conv4= Conv3D(256,kernel\_size=(3,3,3),padding="same",activation='sigmoid', activity\_regularizer= regularizers.l2(0.001))(bat3)

#conv4= Dropout(0.2)(conv4)

#max4 = MaxPooling3D(pool\_size=(2,2,2))(conv4)

#bat4= BatchNormalization()(max4)

#conv5= Conv3D(256,kernel\_size=(3,3,3),padding="same",activation='sigmoid', activity\_regularizer= regularizers.l2(0.001))(bat4)

#conv5= Dropout(0.2)(conv5)

#max5 = MaxPooling3D(pool\_size=(2,2,2))(conv5)

#bat5= BatchNormalization()(max5)

#conv6= Conv3D(512,kernel\_size=(3,3,3),padding="same",activation='sigmoid', activity\_regularizer= regularizers.l2(0.001))(bat5)

flat = Flatten()(conv3)

flat= Dropout(0.4)(flat)

dense= Dense(2,activation="softmax", activity\_regularizer= regularizers.l2(0.05))(flat)

vanillamodel = Model(inputs=x, outputs=dense)

vanillamodel.compile(loss=k.losses.categorical\_crossentropy,

optimizer=k.optimizers.Adam(learning\_rate=0.0001),

metrics=['accuracy'])

return vanillamodel

vanilla3d = makevanilla()

vanilla3d.summary()

model\_checkpoint = ModelCheckpoint('/content/drive/My Drive/vanilla.hdf5', monitor='loss', save\_best\_only=True)

vanilla3d.load\_weights('/content/drive/My Drive/vanilla.hdf5')

batch\_size = 16

val\_gen = get\_batches\_data(val\_paths, voxelWidth, batch\_size)

train\_gen = get\_batches\_data(train\_paths, voxelWidth, batch\_size)

historyVanilla= vanilla3d.fit(train\_gen, steps\_per\_epoch=700,epochs=30, verbose=1, validation\_data= val\_gen,

validation\_steps=111, callbacks=[model\_checkpoint], shuffle=True)

#model1

def makemodel1():

Sequential()

x = Input(shape=(64,64,64,1))

conv0= Conv3D(20,kernel\_size=(5,5,5),padding="same",activation='relu', activity\_regularizer= regularizers.l2(0.01))(x)

conv0= Dropout(0.4)(conv0)

max1 = MaxPooling3D(pool\_size=(2,2,2))(conv0)

#conv1= Conv3D(50,kernel\_size=(5,5,5),padding="same",activation='relu', activity\_regularizer= regularizers.l2(0.01))(max1)

#conv1= Dropout(0.3)(conv1)

#max1 = MaxPooling3D(pool\_size=(2,2,2))(conv1)

flat = Flatten()(max1)

flat= Dropout(0.4)(flat)

dense = Dense (500, activation= "relu", activity\_regularizer= regularizers.l2(0.01))(flat)

flat= Dropout(0.5)(dense)

dense= Dense(2,activation="softmax", activity\_regularizer= regularizers.l2(0.01))(flat)

model1 = Model(inputs=x, outputs=dense)

model1.compile(loss=k.losses.categorical\_crossentropy,

optimizer=k.optimizers.Adam(learning\_rate=0.1),

metrics=['accuracy'])

return model1

model1 = makemodel1()

model1.summary()

model\_checkpoint = ModelCheckpoint('/content/drive/My Drive/lenett.hdf5', monitor='loss', save\_best\_only=True)

model1.load\_weights('/content/drive/My Drive/lenetdel.hdf5')

batch\_size = 64

val\_gen = get\_batches\_data(val\_paths, voxelWidth, batch\_size)

train\_gen = get\_batches\_data(train\_paths, voxelWidth, batch\_size)

historymodel1= model1.fit(train\_gen, steps\_per\_epoch=132,epochs=30, verbose=1, validation\_data= val\_gen,

validation\_steps=33, callbacks=[model\_checkpoint], shuffle=True)

from sklearn.metrics import confusion\_matrix

predicted = vanilla3d.predict(train\_gen)

pre = np.argmax(predicted,axis=1)

tru = np.argmax(ytrain[:],axis=1)

confusion\_matrix(tru, pre)

plotmymodel(historyVanilla)

def plotmymodel(history):

# list all data in history

print(history.history.keys())

# summarize history for accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

plotmymodel(historyvanilla)

!wget "https://github.com/abdullahtarek/Early-Detection-of-lung-cancer-using-machine-learning/raw/master/weights3d\_googlenet.hdf5"

model\_checkpoint = ModelCheckpoint('todel/weights3d\_googlenet.hdf5', monitor='loss', save\_best\_only=True)

googlenet.load\_weights('/content/drive/My Drive/weights3d\_googlenet.hdf5')

model.predict(X\_train)

xtrain.shape

model\_checkpoint = ModelCheckpoint('todel/weights3d\_googlenet.hdf5', monitor='loss', save\_best\_only=True)

historyGoogle= googlenet.fit(xtrain, ytrain,

batch\_size=20,

epochs=20,

verbose=1,

validation\_split=0.1,

callbacks=[model\_checkpoint])

plotmymodel(historyGoogle)

from sklearn.metrics import confusion\_matrix

predicted = googlenet.predict(xtrain[:])

pre = np.argmax(predicted,axis=1)

tru = np.argmax(ytrain[:],axis=1)

confusion\_matrix(tru, pre)

ytrain

plotmymodel(historyGoogle)