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

"""malignancyTraining.ipynb

Automatically generated by Colaboratory.

Original file is located at colab

import xml.etree.ElementTree as ET

import SimpleITK as sitk

import tensorflow as tf

import numpy as np

import os

import pandas as pd

import matplotlib.pyplot as plt

import math

import scipy.ndimage

from tqdm import tqdm

from glob import glob

from random import shuffle

import keras as k

from keras.models import Sequential ,Model

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

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

from keras.callbacks import ModelCheckpoint, LearningRateScheduler

from keras.preprocessing import image

from keras import applications

from keras.optimizers import SGD, RMSprop, Adam, Adagrad, Adadelta

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

outdir = "cropsout/"

count=0

for xmln in tqdm(glob("tcia-lidc-xml/\*/\*.xml")):

tree = ET.parse(xmln)

root = tree.getroot()

try:

seriesID = (root.find("{http://www.nih.gov}ResponseHeader")).find("{http://www.nih.gov}SeriesInstanceUid")

path = glob("../subset\*/"+seriesID.text+".mhd")

if len(path)>0:

for reading in root.findall("{http://www.nih.gov}readingSession"):

for unblindedread in reading.findall("{http://www.nih.gov}unblindedReadNodule"):

if unblindedread.find("{http://www.nih.gov}characteristics") !=None:

sumx=0

sumy=0

sumz=0

malignancy =int(unblindedread.find("{http://www.nih.gov}characteristics").find("{http://www.nih.gov}malignancy").text)

countz=0

countxy=0

for roi in unblindedread.findall("{http://www.nih.gov}roi"):

sumz= sumz+float(roi.find("{http://www.nih.gov}imageZposition").text)

countz = countz+1

for edge in roi.findall("{http://www.nih.gov}edgeMap"):

sumx=sumx + float(edge.find(("{http://www.nih.gov}xCoord")).text)

sumy=sumy+float(edge.find(("{http://www.nih.gov}yCoord")).text)

countxy = countxy+1

sumx=sumx/countxy

sumy=sumy/countxy

sumz=sumz/countz

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

worldCoord = np.asarray([sumz,sumy,sumx])

voxelCoord = worldToVoxelCoord(worldCoord, numpyOrigin, numpySpacing)

voxelCoord = np.asarray([voxelCoord[0],sumy,sumx])

voxelWidth = 64

patch = 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)]

patch = normalizePlanes(patch)

desired= malignancy

if(desired==0):

yy = [0,0,0,0,0,1]

if(desired==1):

yy = [0,0,0,0,1,0]

if(desired==2):

yy = [0,0,0,1,0,0]

if(desired==3):

yy = [0,0,1,0,0,0]

if(desired==4):

yy = [0,1,0,0,0,0]

if(desired==5):

yy = [1,0,0,0,0,0]

np.save(outdir+str(count)+"X.npy",patch)

np.save(outdir+str(count)+"Y.npy",yy)

count= count+1

#print(sumz,sumx,sumy,malignancy)

except:

print("NOT IN DATASET")

xtrain , ytrain= [] ,[]

l1 = glob(outdir+"\*X.npy")

ll = l1[:]

shuffle(ll)

for x in ll:

y = x[:-5] +"Y" + x[-4:]

xx= np.load(x)

yy = np.load(y)

if xx.shape == (64,64,64):

xx = np.expand\_dims(xx,axis=3)

xtrain.append(xx)

ytrain.append(yy)

xtrain = np.array(xtrain)

ytrain = np.array(ytrain)

xtrain.shape , ytrain.shape

model = Sequential()

model.add(Conv2D(32, kernel\_size=(5, 5), strides=(1, 1),

activation='relu',

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

model.add(Dropout(0.4))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

model.add(Conv2D(64, (5, 5), activation='relu'))

model.add(Dropout(0.4))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dropout(0.4))

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

model.add(Dropout(0.4))

model.add(Dense(6, activation='softmax'))

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

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

metrics=['accuracy'])

model\_checkpoint = ModelCheckpoint('malignancy\_crop.hdf5', monitor='loss', save\_best\_only=True)

model.load\_weights("malignancy\_crop.hdf5")

model.fit(xtrain, ytrain,

batch\_size=1000,

epochs=100,

verbose=1,

validation\_split=0.1,

callbacks=[model\_checkpoint])

count0=0

count1=0

count2=0

count3=0

count4=0

count5=0

for y in ytrain:

if np.argmax(y)==0:

count0=count0+1

if np.argmax(y)==1:

count1=count1+1

if np.argmax(y)==2:

count2=count2+1

if np.argmax(y)==3:

count3=count3+1

if np.argmax(y)==4:

count4=count4+1

if np.argmax(y)==5:

count5=count5+1

print(count0,count1,count2,count3,count4,count5)

img\_rows = 64

img\_cols=64

channels=64

num\_classes = 6

INIT\_LR = 1e-3

middle\_layers\_activation = "relu"

last\_layer\_activation = "softmax"

batch\_size = 500

epochs = 30

input\_shape = (img\_rows, img\_cols, channels)

model1 = Sequential()

model1.add(Conv2D(20, (5, 5), padding="same",

input\_shape=input\_shape))

model1.add(Activation(middle\_layers\_activation))

model1.add(Dropout(0.2))

model1.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

model1.add(Conv2D(50, (5, 5), padding="same"))

model1.add(Activation(middle\_layers\_activation))

model1.add(Dropout(0.2))

model1.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

model1.add(Flatten())

model1.add(Dense(500))

model1.add(Activation(middle\_layers\_activation))

model1.add(Dropout(0.3))

model1.add(Dense(num\_classes))

model1.add(Activation(last\_layer\_activation))

opt = Adam(lr=INIT\_LR, decay=INIT\_LR / epochs)

model1.compile(loss="binary\_crossentropy",

optimizer=opt,

metrics=['accuracy'])

model\_checkpoint = ModelCheckpoint('lenetdel.hdf5', monitor='loss', save\_best\_only=True)

model1.load\_weights('lenet.hdf5')

history = model1.fit(xtrain, ytrain,

batch\_size = 60, epochs=100, shuffle=True, validation\_split=0.1 , callbacks = [model\_checkpoint])

from sklearn.metrics import confusion\_matrix

predicted = model1.predict(xtrain[:])

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

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

confusion\_matrix(tru, pre)

#SEA born

def plotmymodel(history):

# list all data in history

print(history.history.keys())

# summarize history for accuracy

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

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

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(history)

def inceptionlayer(prev):

tower\_1 = Conv3D(64, (1,1,1), padding='same', activation='relu')(prev)

tower\_1 = Conv3D(64, (3,3,3), padding='same', activation='relu')(tower\_1)

tower\_2 = Conv3D(64, (1,1,1), padding='same', activation='relu')(prev)

tower\_2 = Conv3D(64, (5,5,5), padding='same', activation='relu')(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')(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')(x)

conv1= Dropout(0.3)(conv1)

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

conv2= Conv3D(32,kernel\_size=(3,3,3),activation='relu')(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))(incp4)

flat = Flatten()(avg1)

flat= Dropout(0.3)(flat)

dense= Dense(num\_classes,activation="softmax")(flat)

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

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

optimizer=k.optimizers.Adam(lr=0.0001),

metrics=['accuracy'])

return googlenet

googlenet = makegooglenet()

googlenet.summary()

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

googlenet.load\_weights('weights3d\_malignancy\_googlenet.hdf5')

history= googlenet.fit(xtrain, ytrain,

batch\_size=20,

epochs=5,

verbose=1,

validation\_split=0.1,

callbacks=[model\_checkpoint])

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)

plotmymodel(history)

#Vanilla3d

def makevanilla():

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

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

conv0= Dropout(0.2)(conv0)

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

conv1= Conv3D(32,kernel\_size=(3,3,3),padding="same",activation='relu')(avg1)

conv1= Dropout(0.2)(conv1)

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

conv2= Conv3D(64,kernel\_size=(3,3,3),padding="same",activation='relu')(max1)

conv2= Dropout(0.2)(conv2)

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

conv3= Conv3D(128,kernel\_size=(3,3,3),padding="same",activation='relu')(max2)

conv3= Dropout(0.2)(conv3)

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

conv4= Conv3D(256,kernel\_size=(3,3,3),padding="same",activation='relu')(max3)

conv4= Dropout(0.2)(conv4)

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

conv5= Conv3D(256,kernel\_size=(3,3,3),padding="same",activation='relu')(max4)

conv5= Dropout(0.2)(conv5)

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

conv6= Conv3D(512,kernel\_size=(3,3,3),padding="same",activation='relu')(max5)

flat = Flatten()(conv6)

flat= Dropout(0.2)(flat)

dense= Dense(num\_classes,activation="softmax")(flat)

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

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

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

metrics=['accuracy'])

return vanillamodel

vanilla3d = makevanilla()

vanilla3d.summary()

model\_checkpoint = ModelCheckpoint('weights3d\_maliganancy\_vanila.hdf5', monitor='loss', save\_best\_only=True)

vanilla3d.fit(xtrain, ytrain,

batch\_size=20,

epochs=10,

verbose=1,

validation\_split=0.1,

callbacks=[model\_checkpoint])