# Brain Tumor Detection Using a Convolutional Neural Network

( SEGMENTATION PART )

import numpy as np

import cv2

#%matplotlib inline

#from google.colab.patches import cv\_imshow

#from matplotlib import pyplot as plt

from matplotlib import pyplot as plt

from skimage.morphology import extrema

from skimage.morphology import watershed as skwater

def ShowImage(title,img,ctype):

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

if ctype=='bgr':

b,g,r = cv2.split(img) # get b,g,r

rgb\_img = cv2.merge([r,g,b]) # switch it to rgb

plt.imshow(rgb\_img)

elif ctype=='hsv':

rgb = cv2.cvtColor(img,cv2.COLOR\_HSV2RGB)

plt.imshow(rgb)

elif ctype=='gray':

plt.imshow(img,cmap='gray')

elif ctype=='rgb':

plt.imshow(img)

else:

raise Exception("Unknown colour type")

plt.axis('off')

plt.title(title)

plt.show()

#insert image name

img = cv2.imread('brains1.png')

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

ShowImage('Brain MRI',gray,'gray')

ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH\_OTSU)

ShowImage('Thresholding image',thresh,'gray')

ret, markers = cv2.connectedComponents(thresh)

#Get the area taken by each component. Ignore label 0 since this is the background.

marker\_area = [np.sum(markers==m) for m in range(np.max(markers)) if m!=0]

#Get label of largest component by area

largest\_component = np.argmax(marker\_area)+1 #Add 1 since we dropped zero above

#Get pixels which correspond to the brain

brain\_mask = markers==largest\_component

brain\_out = img.copy()

#In a copy of the original image, clear those pixels that don't correspond to the brain

brain\_out[brain\_mask==False] = (0,0,0)

#save image in brain folder

#and then give the name of image here

img = cv2.imread('brains1.png')

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)

# noise removal

kernel = np.ones((3,3),np.uint8)

opening = cv2.morphologyEx(thresh,cv2.MORPH\_OPEN,kernel, iterations = 2)

# sure background area

sure\_bg = cv2.dilate(opening,kernel,iterations=3)

# Finding sure foreground area

dist\_transform = cv2.distanceTransform(opening,cv2.DIST\_L2,5)

ret, sure\_fg = cv2.threshold(dist\_transform,0.7\*dist\_transform.max(),255,0)

# Finding unknown region

sure\_fg = np.uint8(sure\_fg)

unknown = cv2.subtract(sure\_bg,sure\_fg)

# Marker labelling

ret, markers = cv2.connectedComponents(sure\_fg)

# Add one to all labels so that sure background is not 0, but 1

markers = markers+1

# Now, mark the region of unknown with zero

markers[unknown==255] = 0

markers = cv2.watershed(img,markers)

img[markers == -1] = [255,0,0]

im1 = cv2.cvtColor(img,cv2.COLOR\_HSV2RGB)

ShowImage('segmented image',im1,'gray')

brain\_mask = np.uint8(brain\_mask)

kernel = np.ones((8,8),np.uint8)

closing = cv2.morphologyEx(brain\_mask, cv2.MORPH\_CLOSE, kernel)

ShowImage('Closing', closing, 'gray')

brain\_out = img.copy()

#In a copy of the original image, clear those pixels that don't correspond to the brain

brain\_out[closing==False] = (0,0,0)

# Brain Tumor Detection Using a Convolutional Neural Network

(CLASSIFICATION PART)

import tensorflow as tf

from tensorflow.keras.layers import Conv2D, Input, ZeroPadding2D, BatchNormalization, Activation, MaxPooling2D, Flatten, Dense

from tensorflow.keras.models import Model, load\_model

from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import f1\_score

from sklearn.utils import shuffle

import cv2

import imutils

import numpy as np

import matplotlib.pyplot as plt

import time

from os import listdir

%matplotlib inline

def crop\_brain\_contour(image, plot=False):

    #import imutils

    #import cv2

    #from matplotlib import pyplot as plt

    # Convert the image to grayscale, and blur it slightly

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    gray = cv2.GaussianBlur(gray, (5, 5), 0)

    # Threshold the image, then perform a series of erosions +

    # dilations to remove any small regions of noise

    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH\_BINARY)[1]

    thresh = cv2.erode(thresh, None, iterations=2)

    thresh = cv2.dilate(thresh, None, iterations=2)

    # Find contours in thresholded image, then grab the largest one

    cnts = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

    cnts = imutils.grab\_contours(cnts)

    c = max(cnts, key=cv2.contourArea)

    # Find the extreme points

    extLeft = tuple(c[c[:, :, 0].argmin()][0])

    extRight = tuple(c[c[:, :, 0].argmax()][0])

    extTop = tuple(c[c[:, :, 1].argmin()][0])

    extBot = tuple(c[c[:, :, 1].argmax()][0])

    # crop new image out of the original image using the four

extreme points (left, right, top, bottom)

    new\_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]

    if plot:

        plt.figure()

        plt.subplot(1, 2, 1)

        plt.imshow(image)

        plt.tick\_params(axis='both', which='both',

                        top=False, bottom=False, left=False, right=False,

                        labelbottom=False, labeltop=False,

labelleft=False, labelright=False)

        plt.title('Original Image')

        plt.subplot(1, 2, 2)

        plt.imshow(new\_image)

        plt.tick\_params(axis='both', which='both',

                        top=False, bottom=False, left=False, right=False,

                        labelbottom=False, labeltop=False,

labelleft=False, labelright=False)

        plt.title('Cropped Image')

        plt.show()

    return new\_image

ex\_img = cv2.imread('yes/Y1.jpg')

ex\_new\_img = crop\_brain\_contour(ex\_img, True)

def load\_data(dir\_list, image\_size):

    """

    Read images, resize and normalize them.

    Arguments:

        dir\_list: list of strings representing file directories.

    Returns:

        X: A numpy array with shape = (#\_examples, image\_width,

image\_height, #\_channels)

        y: A numpy array with shape = (#\_examples, 1)

    """

    # load all images in a directory

    X = []

    y = []

    image\_width, image\_height = image\_size

    for directory in dir\_list:

        for filename in listdir(directory):

            # load the image

            image = cv2.imread(directory + '\\' + filename)

            # crop the brain and ignore the unnecessary rest

part of the image

            image = crop\_brain\_contour(image, plot=False)

            # resize image

            image = cv2.resize(image, dsize=(image\_width, image\_height),

interpolation=cv2.INTER\_CUBIC)

            # normalize values

            image = image / 255.

            # convert image to numpy array and append it to X

            X.append(image)

            # append a value of 1 to the target array if the image

            # is in the folder named 'yes', otherwise append 0.

            if directory[-3:] == 'yes':

                y.append([1])

            else:

                y.append([0])

    X = np.array(X)

    y = np.array(y)

    # Shuffle the data

    X, y = shuffle(X, y)

    print(f'Number of examples is: {len(X)}')

    print(f'X shape is: {X.shape}')

    print(f'y shape is: {y.shape}')

    return X, y

augmented\_path = 'augmented data/'

# augmented data (yes and no) contains both the original and the new

generated examples

augmented\_yes = augmented\_path + 'yes'

augmented\_no = augmented\_path + 'no'

IMG\_WIDTH, IMG\_HEIGHT = (240, 240)

X, y = load\_data([augmented\_yes, augmented\_no], (IMG\_WIDTH, IMG\_HEIGHT))

def plot\_sample\_images(X, y, n=50):

    """

    Plots n sample images for both values of y (labels).

    Arguments:

        X: A numpy array with shape = (#\_examples, image\_width,

image\_height, #\_channels)

        y: A numpy array with shape = (#\_examples, 1)

    """

    for label in [0,1]:

        # grab the first n images with the corresponding

y values

 equal to label

        images = X[np.argwhere(y == label)]

        n\_images = images[:n]

        columns\_n = 10

        rows\_n = int(n/ columns\_n)

        plt.figure(figsize=(20, 10))

        i = 1 # current plot

        for image in n\_images:

            plt.subplot(rows\_n, columns\_n, i)

            plt.imshow(image[0])

            # remove ticks

     plt.tick\_params(axis='both', which='both',

top=False, bottom=False, left=False, right=False,

     labelbottom=False, labeltop=False, labelleft=False, labelright=False)

            i += 1

        label\_to\_str = lambda label: "Yes" if label == 1 else "No"

        plt.suptitle(f"Brain Tumor: {label\_to\_str(label)}")

        plt.show()

plot\_sample\_images(X, y)

def split\_data(X, y, test\_size=0.2):

    """

    Splits data into training, development and test sets.

    Arguments:

        X: A numpy array with shape = (#\_examples, image\_width,

image\_height, #\_channels)

        y: A numpy array with shape = (#\_examples, 1)

    Returns:

        X\_train: A numpy array with shape = (#\_train\_examples,

image\_width, image\_height, #\_channels)

        y\_train: A numpy array with shape = (#\_train\_examples, 1)

        X\_val: A numpy array with shape = (#\_val\_examples,

image\_width, image\_height, #\_channels)

        y\_val: A numpy array with shape = (#\_val\_examples, 1)

        X\_test: A numpy array with shape = (#\_test\_examples,

image\_width, image\_height, #\_channels)

        y\_test: A numpy array with shape = (#\_test\_examples, 1)

    """

    X\_train, X\_test\_val, y\_train, y\_test\_val = train\_test\_split(X, y,

test\_size=test\_size)

    X\_test, X\_val, y\_test, y\_val = train\_test\_split(X\_test\_val,

y\_test\_val, test\_size=0.5)

    return X\_train, y\_train, X\_val, y\_val, X\_test, y\_test

X\_train, y\_train, X\_val, y\_val, X\_test, y\_test = split\_data(X, y, test\_size=0.3)

print ("number of training examples = " + str(X\_train.shape[0]))

print ("number of development examples = " + str(X\_val.shape[0]))

print ("number of test examples = " + str(X\_test.shape[0]))

print ("X\_train shape: " + str(X\_train.shape))

print ("Y\_train shape: " + str(y\_train.shape))

print ("X\_val (dev) shape: " + str(X\_val.shape))

print ("Y\_val (dev) shape: " + str(y\_val.shape))

print ("X\_test shape: " + str(X\_test.shape))

print ("Y\_test shape: " + str(y\_test.shape))

# Nicely formatted time string

def hms\_string(sec\_elapsed):

    h = int(sec\_elapsed / (60 \* 60))

    m = int((sec\_elapsed % (60 \* 60)) / 60)

    s = sec\_elapsed % 60

    return f"{h}:{m}:{round(s,1)}"

def compute\_f1\_score(y\_true, prob):

    # convert the vector of probabilities to a target vector

    y\_pred = np.where(prob > 0.5, 1, 0)

    score = f1\_score(y\_true, y\_pred)

    return score

def build\_model(input\_shape):

    """

    Arugments:

        input\_shape: A tuple representing the shape of the input of the

model. shape=(image\_width, image\_height, #\_channels)

    Returns:

        model: A Model object.

    """

    # Define the input placeholder as a tensor with shape input\_shape.

    X\_input = Input(input\_shape) # shape=(?, 240, 240, 3)

    # Zero-Padding: pads the border of X\_input with zeroes

    X = ZeroPadding2D((2, 2))(X\_input) # shape=(?, 244, 244, 3)

    # CONV -> BN -> RELU Block applied to X

    X = Conv2D(32, (7, 7), strides = (1, 1), name = 'conv0')(X)

    X = BatchNormalization(axis = 3, name = 'bn0')(X)

    X = Activation('relu')(X) # shape=(?, 238, 238, 32)

    # MAXPOOL

    X = MaxPooling2D((4, 4), name='max\_pool0')(X) # shape=(?, 59, 59, 32)

    # MAXPOOL

    X = MaxPooling2D((4, 4), name='max\_pool1')(X) # shape=(?, 14, 14, 32)

    # FLATTEN X

    X = Flatten()(X) # shape=(?, 6272)

    # FULLYCONNECTED

    X = Dense(1, activation='sigmoid', name='fc')(X) # shape=(?, 1)

    # Create model. This creates your Keras model instance,

you'll use this instance to train/test the model.

    model = Model(inputs = X\_input, outputs = X,

name='BrainDetectionModel')

    return model

IMG\_SHAPE = (IMG\_WIDTH, IMG\_HEIGHT, 3)

model = build\_model(IMG\_SHAPE)

model.summary()

model.compile(optimizer='adam', loss='binary\_crossentropy',

metrics=['accuracy'])

# tensorboard

log\_file\_name = f'brain\_tumor\_detection\_cnn\_{int(time.time())}'

tensorboard = TensorBoard(log\_dir=f'logs/{log\_file\_name}')

# checkpoint

# unique file name that will include the epoch and the validation

(development) accuracy

filepath="cnn-parameters-improvement-{epoch:02d}-{val\_acc:.2f}"

# save the model with the best validation (development) accuracy till now

checkpoint = ModelCheckpoint("models/{}.model".format(filepath, monitor='val\_acc', verbose=1, save\_best\_only=True, mode='max'))

start\_time = time.time()

model.fit(x=X\_train, y=y\_train, batch\_size=32, epochs=3,

validation\_data=(X\_val, y\_val), callbacks=[tensorboard, checkpoint])

end\_time = time.time()

execution\_time = (end\_time - start\_time)

print(f"Elapsed time: {hms\_string(execution\_time)}")

start\_time = time.time()

model.fit(x=X\_train, y=y\_train, batch\_size=32, epochs=3, validation\_data=(X\_val, y\_val), callbacks=[tensorboard, checkpoint])

end\_time = time.time()

execution\_time = (end\_time - start\_time)

print(f"Elapsed time: {hms\_string(execution\_time)}")

start\_time = time.time()

model.fit(x=X\_train, y=y\_train, batch\_size=32, epochs=3, validation\_data=(X\_val, y\_val), callbacks=[tensorboard, checkpoint])

end\_time = time.time()

execution\_time = (end\_time - start\_time)

print(f"Elapsed time: {hms\_string(execution\_time)}")

history = model.history.history

for key in history.keys():

    print(key)

def plot\_metrics(history):

    train\_loss = history['loss']

    val\_loss = history['val\_loss']

    train\_acc = history['acc']

    val\_acc = history['val\_acc']

    # Loss

    plt.figure()

    plt.plot(train\_loss, label='Training Loss')

    plt.plot(val\_loss, label='Validation Loss')

    plt.title('Loss')

    plt.legend()

    plt.show()

    # Accuracy

    plt.figure()

    plt.plot(train\_acc, label='Training Accuracy')

    plt.plot(val\_acc, label='Validation Accuracy')

    plt.title('Accuracy')

    plt.legend()

    plt.show()

plot\_metrics(history)

best\_model = load\_model

(filepath='models/cnn-parameters-improvement-23-0.91.model')

best\_model.metrics\_names

loss, acc = best\_model.evaluate(x=X\_test, y=y\_test)

print (f"Test Loss = {loss}")

print (f"Test Accuracy = {acc}")

y\_test\_prob = best\_model.predict(X\_test)

f1score = compute\_f1\_score(y\_test, y\_test\_prob)

 print(f"F1 score: {f1score}")

y\_val\_prob = best\_model.predict(X\_val)

f1score\_val = compute\_f1\_score(y\_val, y\_val\_prob)

print(f"F1 score: {f1score\_val}")

y\_val\_prob = best\_model.predict(X\_val)

f1score\_val = compute\_f1\_score(y\_val, y\_val\_prob)

print(f"F1 score: {f1score\_val}")

def data\_percentage(y):

    m=len(y)

    n\_positive = np.sum(y)

    n\_negative = m - n\_positive

    pos\_prec = (n\_positive\* 100.0)/ m

    neg\_prec = (n\_negative\* 100.0)/ m

    print(f"Number of examples: {m}")

    print(f"Percentage of positive examples: {pos\_prec}%, number of pos examples: {n\_positive}")

    print(f"Percentage of negative examples: {neg\_prec}%, number of neg examples: {n\_negative}")

# the whole data

data\_percentage(y)

print("Training Data:")

data\_percentage(y\_train)

print("Validation Data:")

data\_percentage(y\_val)

print("Testing Data:")

data\_percentage(y\_test)