

Medical image processing – ex 3 , ITAMAR ZAFRANI 308334465

חלק 1:

בניית ROI של הכבד:

תיאור הפונקציה:

Input: CT scan, aorta segmentation

Output: saving 3d rectangle that is superset of the liver

```
def find_ROI(CT_scan, aorta_seg):
    print("looking for roi")

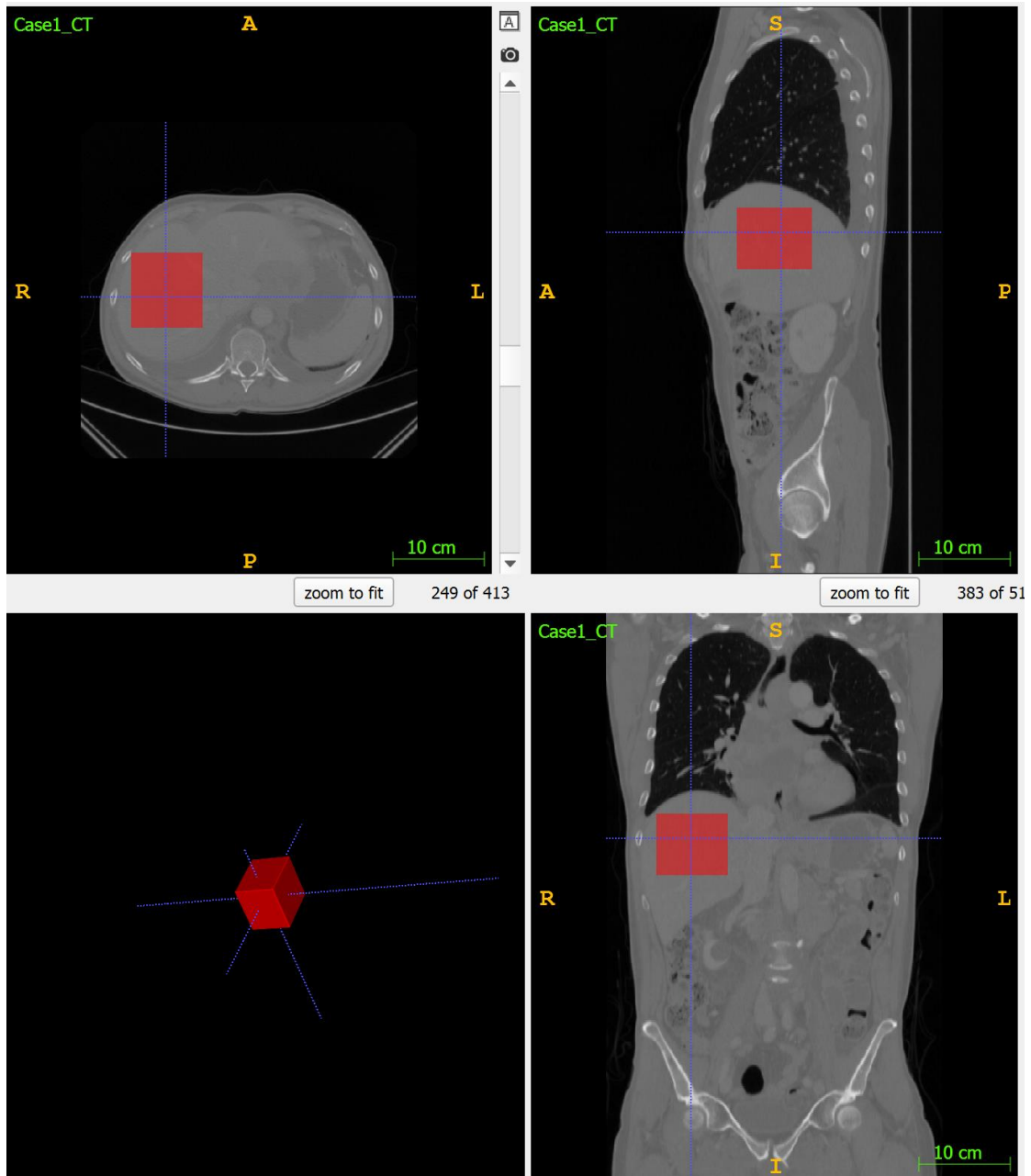
    aorta = nib.load(aorta_seg)
    aorta_data = aorta.get_data()
    img = nib.load(CT_scan)
    img_data = img.get_data()
    tuple_of_aorta_seg = np.nonzero(aorta_data)
    aorta_x = tuple_of_aorta_seg[0]
    aorta_y = tuple_of_aorta_seg[1]
    aorta_z = tuple_of_aorta_seg[2]
    minimal_z = np.amin(aorta_z)
    maximal_z = np.amax(aorta_z)
    minimal_x = np.amin(aorta_x)
    maximal_x = np.amax(aorta_x)
    minimal_y = np.amin(aorta_y)
    maximal_y = np.amax(aorta_y)
    half_z = int(np.floor((minimal_z + maximal_z) / 2))
    quarter_z = int(np.floor((minimal_z + maximal_z) / 4))
    eighth_z = int(np.floor((minimal_z + maximal_z) / 8))
    third_z = int(np.floor((minimal_z + maximal_z) / 3))
    tenth_z = int(np.floor((minimal_z + maximal_z) / 10))
    tent_to_2 = int(tenth_z / 2)
    seventh_z = int((minimal_z + maximal_z) / 7)
    sixth_z = int((minimal_z + maximal_z) / 6)
    fifth_z = int((minimal_z + maximal_z) / 5)
    print(third_z)
    third_z = 2 * third_z

    img_data[half_z] = 0
    print(minimal_z, eighth_z, quarter_z)
    img_data[int(1.2 * maximal_x):int(1.6 * maximal_x), int(1.2 * minimal_y):maximal_y,
    minimal_z + fifth_z:half_z + tent_to_2] = 1
    name_format = img.get_filename().split('.')[0] + "ROI.nii.gz"
    nib.save(img, name_format)

    return img_data
```

בפונקציה אני מוצא את נתוני האורתה, ולפי מיקומים של החתכים של האורתה בניתי חתכים יחסיים לאורתה בהם נמצא הכבד. היחסים נקבעו בעיקר בניסוי וטעיה, כך שהמטרה הייתה שכל ה ROI יהיה מוכל ממש בתוך הכבד, על מנת שכל ה seeds ההתחלתיים שלי יהיו מתוך הכבד.

להלן ה ROI שהתקבל:



חלק 2: מימוש פונקציות :

findSeeds:

input: ct scan, roi

output: 200 seeds

מימוש הפונקציה:

```
def findSeeds(CT scan, ROI):
    wanted_points = 200

    img = nib.load(CT scan)
    img_data = img.get_data()
    seeds = copy.deepcopy(img_data)
    seeds[:] = 0
    img_roi = nib.load(ROI)
    roi_data = img_roi.get_data()
    tuple_of_roi_seg = np.nonzero(roi_data)
    roi_x = tuple_of_roi_seg[0]
    roi_y = tuple_of_roi_seg[1]
    roi_z = tuple_of_roi_seg[2]
```

```

minimal_z = np.amin(roi_z)
maximal_z = np.amax(roi_z)
minimal_x = np.amin(roi_x)
maximal_x = np.amax(roi_x)
minimal_y = np.amin(roi_y)
maximal_y = np.amax(roi_y)
z_axis = list(range(minimal_z, maximal_z))
print(z_axis)
x_axis = list(range(minimal_x, maximal_x))
y_axis = list(range(minimal_y, maximal_y))

cur_points = 0
while cur_points <= wanted_points:
    # print("cur_points ", cur_points)
    z_point = random.choice(z_axis)
    x_point = random.choice(x_axis)
    y_point = random.choice(y_axis)

    # print ("x y z ", x_point , y_point , z_point )
    gray_level = img_data[x_point, y_point, z_point]
    print(gray_level)
    if (gray_level > 0) and (gray_level < 150):
        seeds[x_point, y_point, z_point] = 1
        cur_points += 1
    else:
        print("point has not been choosefd")
        continue

img_data[:, :] = seeds
name_format = img.get_filename().split('.')[0] + "seeds.nii.gz"
nib.save(img, name_format)
return 0

```

במימוש הפונקציה הגרלתי באמצעות ספריית random בכל איטרציה 3 נקודות, אחת מכל ציר, מתוך הROI שנתון. בדקתי האם הנקודה נמצאת בערכי האפור המתאימים לכבד, ואם כן, הוספתי אותה.

multipleSeedsRG

input: CT scan, ROI

output: liver segmentation

מימוש הפונקציה:

```

def multipleSeedsRG(CT_scan, Aorta):
    img = nib.load(CT_scan)
    ROI = img.get_filename().split('.')[0] + "ROI.nii.gz"

    Find_ROI(CT_scan, Aorta)

    img_data = img.get_data().astype(np.uint8)
    xdim, ydim, zdim = img_data.shape
    FindSeeds(CT_scan, ROI)

    checked_voxels = np.zeros((xdim, ydim, zdim))
    checked_voxels.astype(np.uint8)
    new_neighbours_voxels = np.zeros((xdim, ydim, zdim)).astype(np.uint8)
    new_neighbours_voxels.astype(np.uint8)
    seeds_name_format = img.get_filename().split('.')[0] + "seeds.nii.gz"

    seeds = nib.load(seeds_name_format)
    seeds_data = seeds.get_data()
    checked_voxels[seeds_data == 1] = 1
    print("Checked voxels after insert seeds", np.sum(checked_voxels), " while seeds is", np.sum(seeds_data))
    next_segment = morphology.dilation(seeds_data, morphology.cube(3, np.uint8))
    new_neighbours_voxels = np.subtract(next_segment, seeds_data)
    new_neighbours_voxels[checked_voxels == 1] = 0
    checked_voxels[new_neighbours_voxels == 1] = 1
    print("Checked voxels after insert first neighbours", np.sum(checked_voxels), " while seeds is", np.sum(seeds_data))
    iteration = 0
    iterations = []
    new_voxels_added_to_list = []
    new_voxels_added_to_segmentation = []
    while (np.sum(new_neighbours_voxels) > 0):
        iteration += 1
        print("sum of seeds ", np.sum(seeds_data == 1), "iteration", iteration)
        mean = np.mean(img_data[seeds_data == 1])

        print("mean is", mean)
        std = np.std(img_data[seeds_data == 1])
        neighbours_grays = np.zeros((xdim, ydim, zdim))

        # print ("ng shape ", new_neighbours_voxels.shape)

        neighbours_grays[new_neighbours_voxels == 1] = img_data[new_neighbours_voxels == 1]
        seeds_before = np.sum(seeds_data)
        seeds_data[np.absolute(neighbours_grays - mean) < 20] = 1
        seeds_after = np.sum(seeds_data)
        seeds_just_added = seeds_after - seeds_before
        print("seeds added", seeds_just_added)
        # print ("seeds sum after insertion via mean", np.sum(seeds_data))

        next_segment = morphology.dilation(seeds_data, morphology.cube(3, np.uint8))
        # print("seeds sum after dilation", np.sum(seeds_data))
        new_neighbours_voxels = np.subtract(next_segment, seeds_data)
        # print("new neighbors after sumstract dialtion from seeds", np.sum(new_neighbours_voxels))
        new_neighbours_voxels[checked_voxels == 1] = 0
        # print ("new neighbors after subtract who checked ", np.sum(new_neighbours_voxels))

        checked_voxels[new_neighbours_voxels == 1] = 1
        iterations.append(iteration)
        new_voxels_added_to_list.append(np.sum(new_neighbours_voxels))
        new_voxels_added_to_segmentation.append(seeds_just_added)

        if (iteration == 100):
            break
        # if(iteration%30==0):
        #
        #     name_format = img.get_filename().split('.')[0] + "after adding" + str(np.sum(seeds_data==1))+ "in iteration" + str(iteration) + ".nii.gz"
        #     nib.save(seeds, name_format)

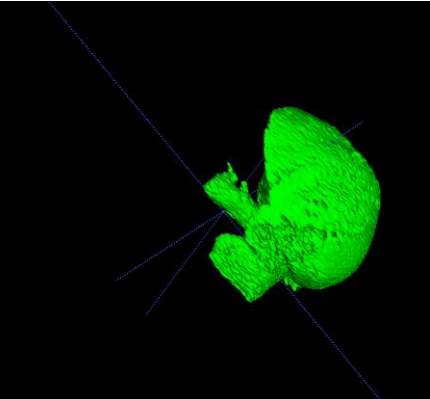
    name_format = img.get_filename().split('.')[0] + "Aftersegmentation50.nii.gz"
    nib.save(seeds, name_format)
    lines = plt.plot(iterations, new_voxels_added_to_list, iterations, new_voxels_added_to_segmentation, 'o')
    plt.setp(lines[0], linewidth = 4)
    plt.setp(lines[1], linewidth = 2)
    plt.legend(("new_voxels_added_to_list", "new_voxels_added_to_segmentation"), loc='upper right')
    plt.title("data")
    plt.show()
    return 0

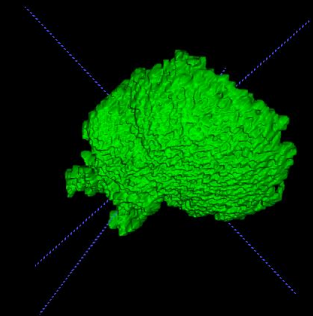
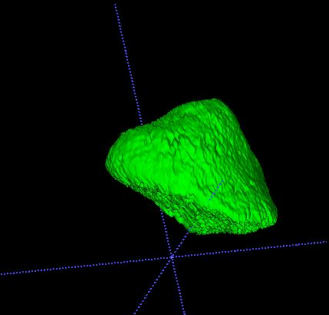
```

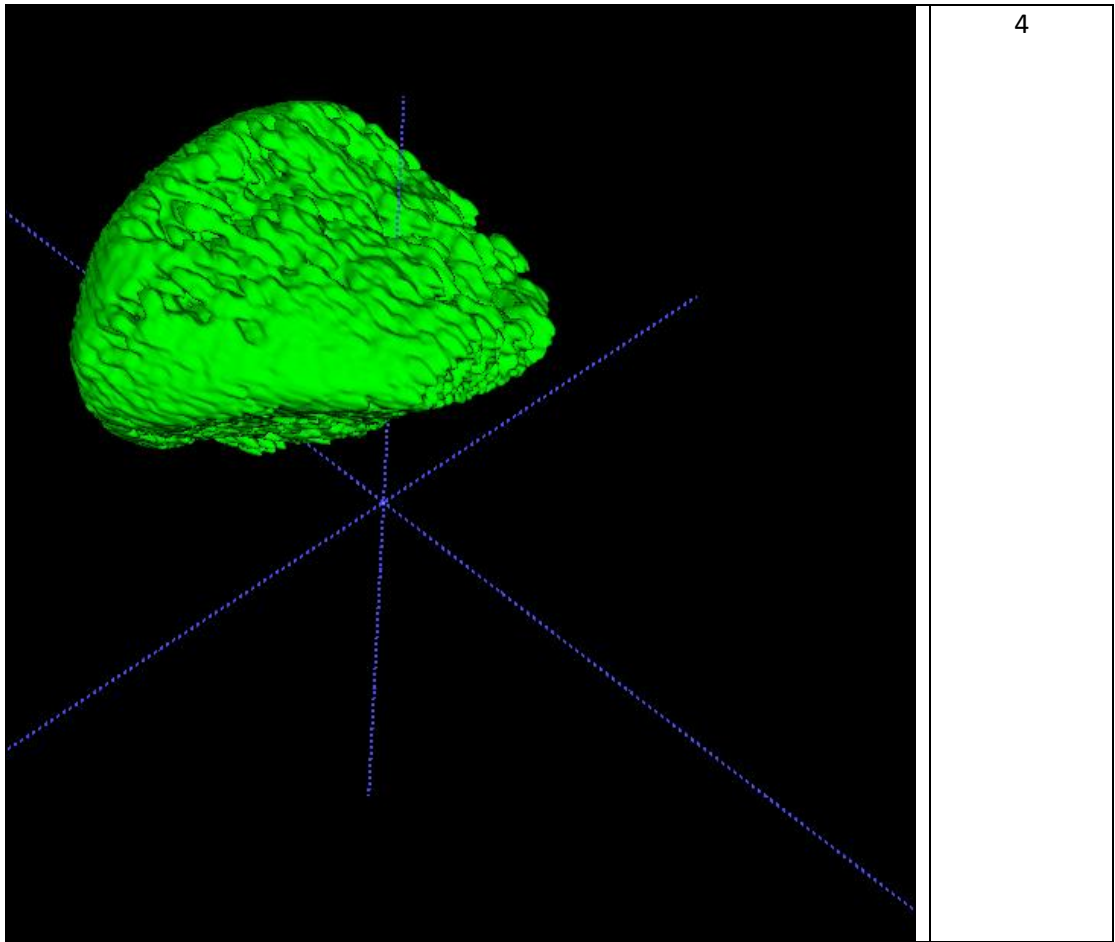
תיאור המימוש:

המימוש מבוסס על האלגוריתם הנלמד בשיעור:

- (1) מייצרים list של כל הseeds
 - (2) מייצרים מערך מועתק של הseeds שישמש אותנו לזכור את כל הseeds שכבר בדקנו. למערך קוראים checked.
 - (3) מוצאים את השכנים של הseeds
 - a. באמצעות dialition במצב cube אנחנו מייצרים קוביה סביב כל פיקסל
 - b. באמצעות numpy.substract אנחנו מחסרים מהdialition את הseeds המקוריים ונשארים רק עם השכנים
 - c. מחסרים את הseeds שמתוייגים בchecked
 - d. מעדכנים את המערך checked להכיל את כל השכנים שאנחנו בודקים
 - (4) בודקים מי עונה על התנאי. בחרתי שהתנאי יהיה שהמרחק מהmean יהיה קטן (בערך מוחלט) מ-20 (ערכי אפור)
 - (5) השכנים שעונים על התנאי נכנסים לרשימת "השכנים החדשה" ולרשימה של הseeds.
 - (6) כל עוד רשימת השכנים החדשים לא ריקה, חזור ל-3
 - (7) לאחר 60 איטרציות, עצור. זוהי היוריסטיקה מתבקשת לאחר לילות שחייתי לסגמנטציה.
- מכיוון שאברים סמוכים בעלי רמות אפור דומות, לאחר ההאלגוריתם אני מבצע מספר פעולות מורפולוגיות (erusion, dialition, remove small objects) על מנת להישאר עם כבד ובלבד.

SEGMENTATION			CT
			1

			2
			3



חלק ג' – הערכת הסגמנטציה:

תיאור הפונקציה evaluateSegmentation

Input: segmentation, ground truth

Output: VOD, DC

את הפונקציה מימשנו במקור בתרגיל קודם, והוספתי שינוי מתבקש

טבלת תוצאות:

DC	VOD	Ct
0.797	0.601	1
לא היו ground Truth		2
		3
		4