Medical Image processing

Ex 1

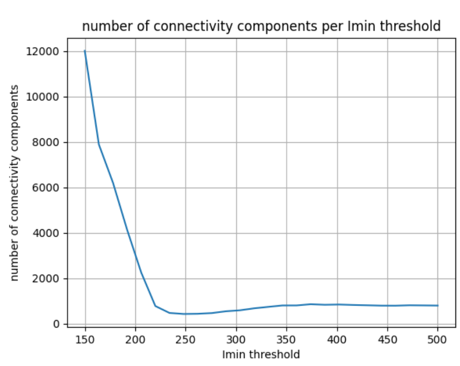
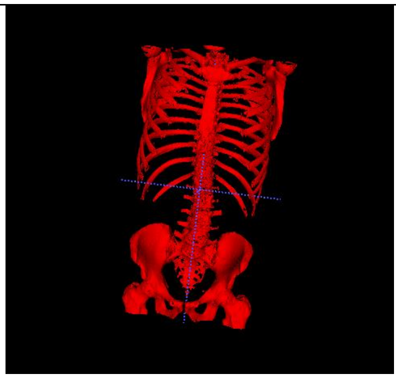
Amitai Ovadia 312244254

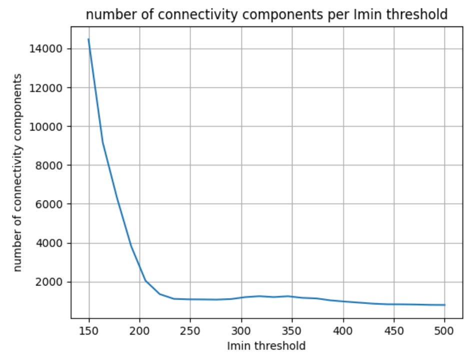
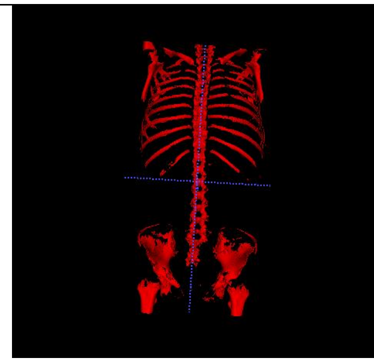
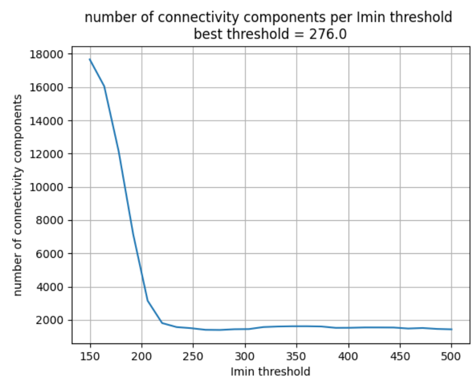
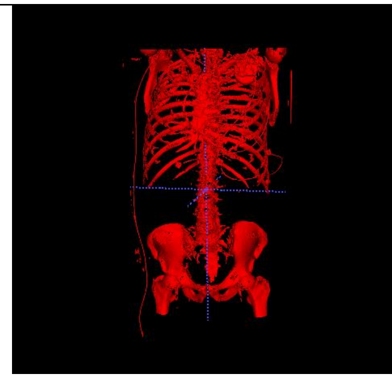
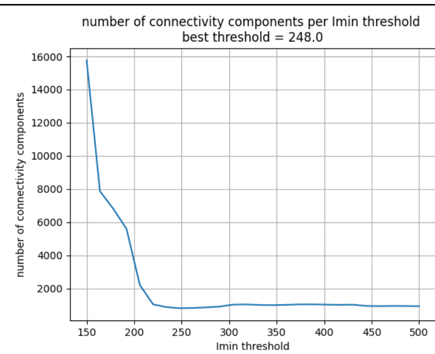
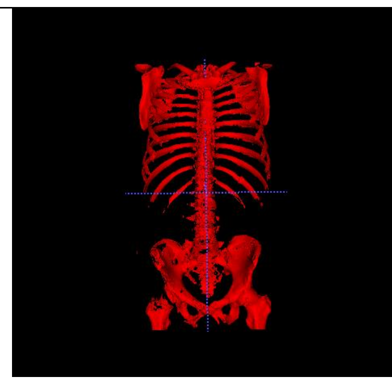
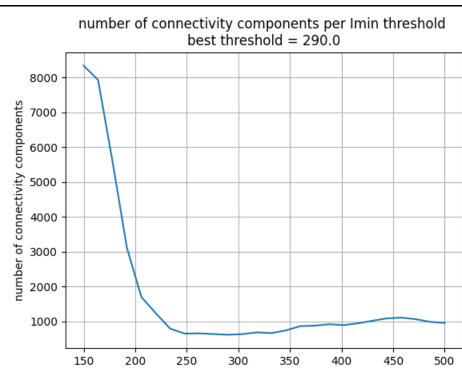
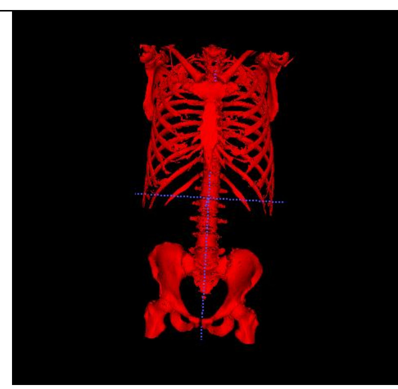
<u>Part 1:</u>

Design:

- The solution thresholds all voxel values in the CT between minimum and maximum values using np.where function.
- It performs the thresholding for all values between [150, 500] in jumps of 14 and returns the result with the lowest number of connectivity components
- Then I performed morphological operations such as remove small objects, remove small holes and binary closing for several iterations, and stopped after a fixed amount of iterations so I won't erode all the image. Then I saved the resulting image.

Results:

CT1	Min threshold 248		

<table>
<tr><td>CT2</td><td>Min
threshold
500</td><td>
</td><td></td></tr>
<tr><td>CT3</td><td>Min
threshold
276</td><td>
</td><td></td></tr>
<tr><td>CT4</td><td>Min
threshold
248</td><td>
</td><td></td></tr>
<tr><td>CT5</td><td>Min
threshold
290</td><td>
</td><td></td></tr>
</table>

libraries:

- Skimage :
- Numpy

Methods:

class BonesSegmentation;

Function: SkeletonTHFinder Takes the CT in the nifty_file and segments the skeleton using threshold and morphological operations.
Input: self.nifty_file_path
Output: saves the new skeleton segmentation as a nifty file

Function: get_best_threshold Searching through the threshold space and returning the threshold that gives the lowest number of connected components in the 3D CT image
Input: self.nifty_file_path
Output: best_threshold, int representing the best minimum threshold

Function: SegmentationByTH loads the CT image and thresholds it according to Imin and Imax values
Input: Imin: minimal threshold Imax: maximal threshold
Output: save the segmented nifty file and returns 1 is succeeded 0 if failed

Part 2:

Design:

- CT scan is cut, and the remaining slices are the ones that include L1
- CT is cut again, according to assumed Aorta location in x, y: close the front of L1.
- Assuming that the Aorta is round, or close to round I performed the following trick:
- Each of this cut CT scans is then analyzed using the Circles Hough Transform. The supplied parameters are the search radii: between 7 and 20 pixels,

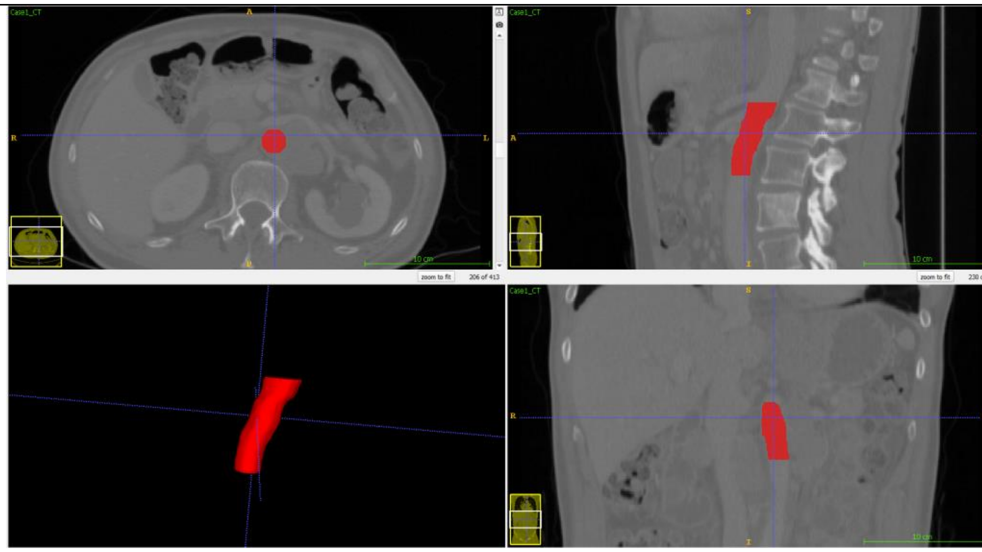
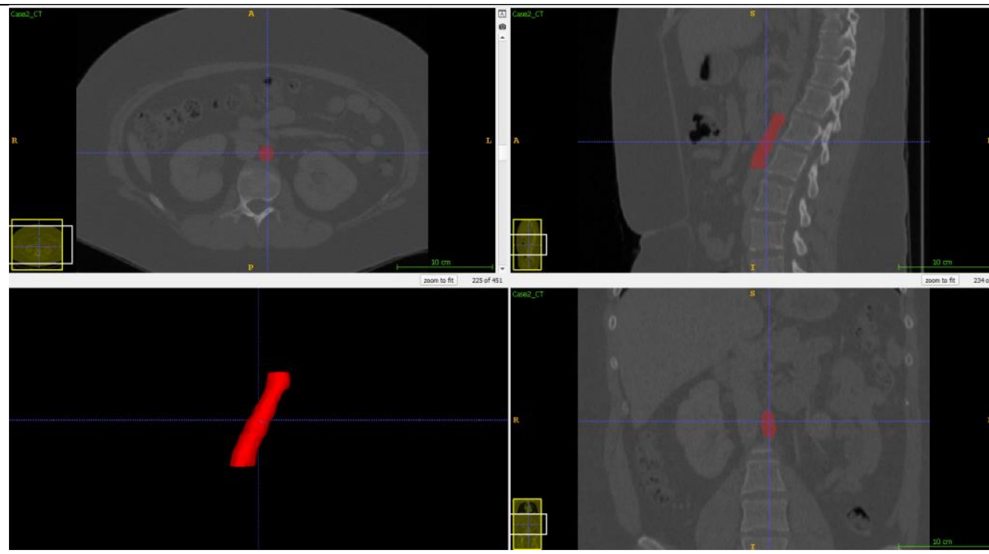
  and the number of desired circles is 20.

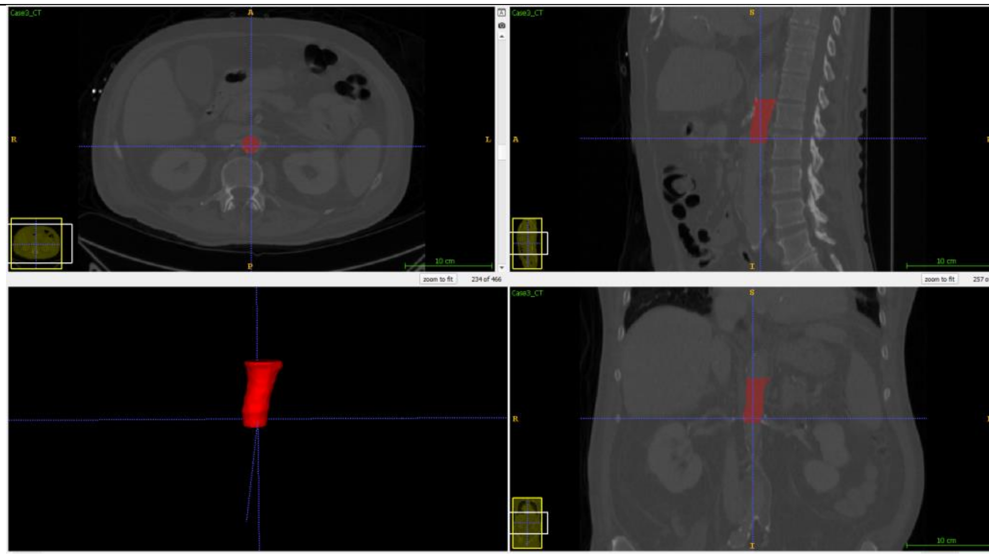
  overall I got 20 circle candidates for Aorta location per slice,

  ordered by their likelihood.
- Then in order to choose the most fitting circle for Aorta location in each slice I assumed that the Aorta is more or less in the same location in consecutive slices and used the following method:
- Fixing the first chosen circle in the lowest slice to be the most likely circle,

  I then chose the above circle, from the 20 different candidates, to be the one that is closest to the previous circle in terms of location and radius.
- I did this iteratively, choosing for every slice it's Aorta location circle according to it's previous neighbor.
- If no close circle was found, according to a specified error rate, a circle was then created in the location and radius of the previous circle (almost never happened).
- Then I created the Aorta segmentation by filling the circles per slice and stacking them on top of each other, and preformed binary closing in order to smooth the result.

Results:

CT1	DICE = 0.93 VOD = 0.12	
CT2	DICE = 0.82 VOD = 0.30	

CT3	DICE = 0.87 VOD = 0.22	
CT4	DICE = 0.91 VOD = 0.16	

Library:

- Morphology, transform, draw, feature, exposure and measure from skimage
- Numpy

Methods:

class AortaSegmentation:

Function: get_L1_boundaries(self) find x,y,z boundaries of L1 to cut the CT accordingly
Input: self.L1_img
Output: min_x, max_x, min_y, max_y, min_z, max_z

Function: AortaSegmentation(self) Segment Aorta image and save segmentation, uses the function self.find_Aorta()
Input: self.L1_img, self.CT_img
Output: saves the Aorta segmentation

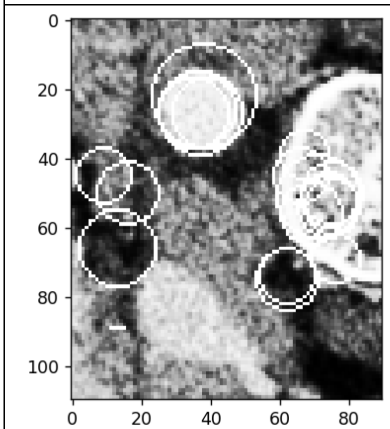
Function: find_Aorta(self) finds all Aorta pixels in CT: first per slice and then for all 3D image uses the functions: self.find_possible_Aorta_circles() self.extract_best_Aorta_circles(circles) self.construct_3d_Aurta_segmentation(final_circles)
Input: self
Output: segmented_Aorta

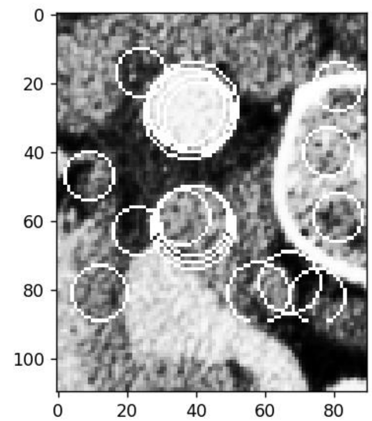
Function: find_possible_Aorta_circles(self)

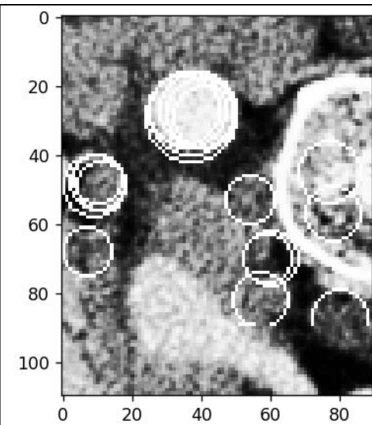
for each L1 slice finds 20 possible Aorta's circle locations
using hough transform

Input: self.cut_CT_img

Output: circles: a list of circles information arrays per slice
In the images the all the candidate circles for the Aorta locations in a specific slice is encircled in white.





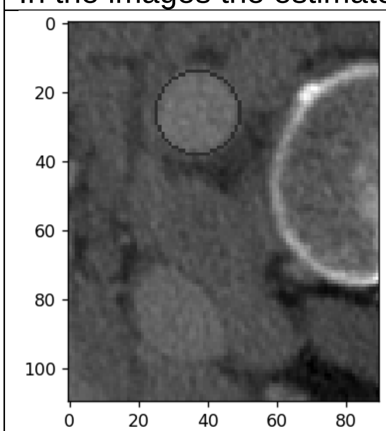


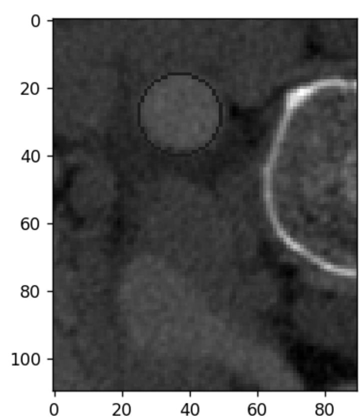
Function: extract_best_Aorta_circles(self, circles)

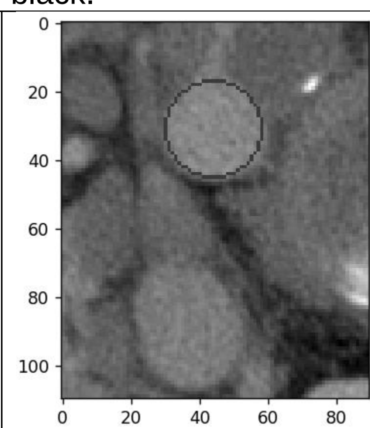
extract the most fitting Aorta circle for each slice

Input: circles: the output of find_possible_Aorta_circles

Output: final_circles: an array of the selected 1 circle per slice for Aorta location
In the images the estimated Aorta locations is encircled in black.







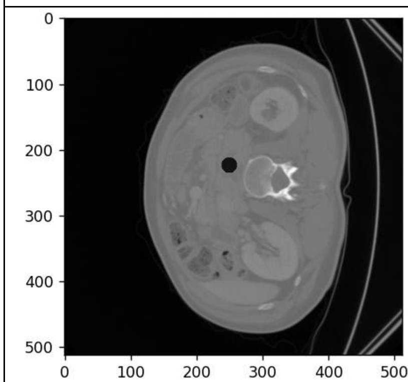
Function: construct_3d_Aurta_segmentation(self, final_circles)

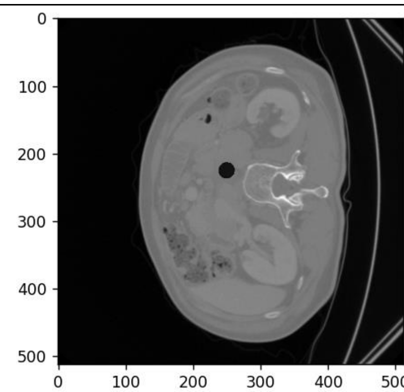
construct 3D Aorta segmentation from Aorta circles information from each slice

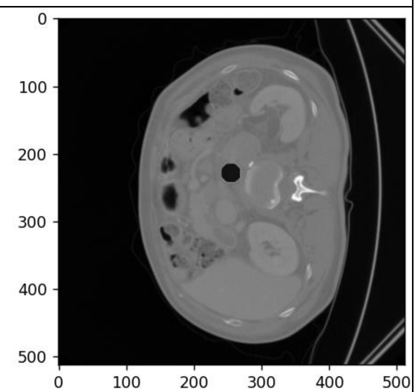
Input: final_circles: Aorta circles information from each slice

Output: 3D Aorta segmentation

The images are of the Aurta estimated location in some CT slices







Function: evaluateSegmentation(GT_seg, est_seg)

evaluate performance of segmentation compared to the ground truth segmentation

Input: GT_seg: ground truth segmentation
est_seg: computed segmentation

Output:
DICE ((2 * intersection_size) / (size_A + size_B))
and
VOD (1 - intersection_size/union_size)
scores for evaluating the computed segmentation