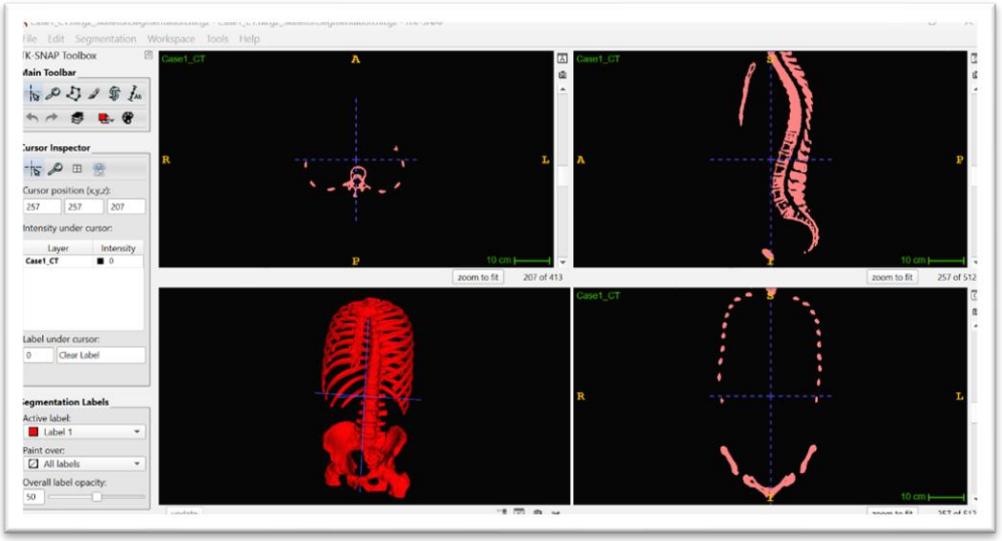
MEDICAL IMAGE PROCESSING (67705):

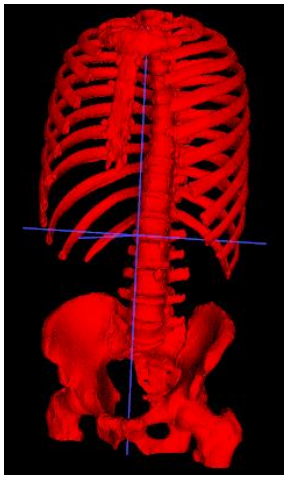
EXERCISE 1- PART 1

BY: AMIT HALBREICH, ID: 208917393

Case 1 CT

Segmentation Model Results

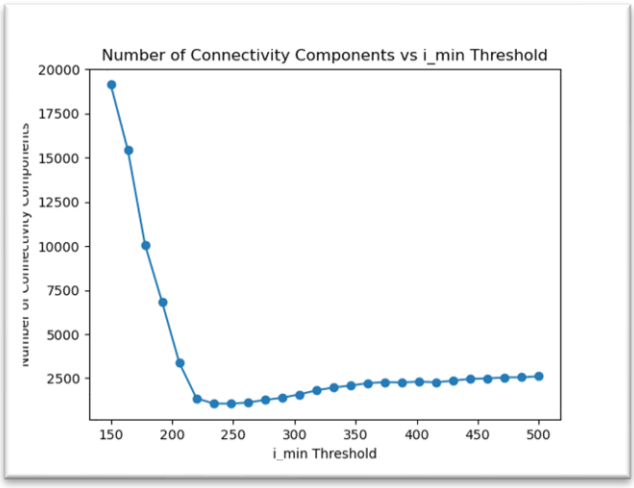




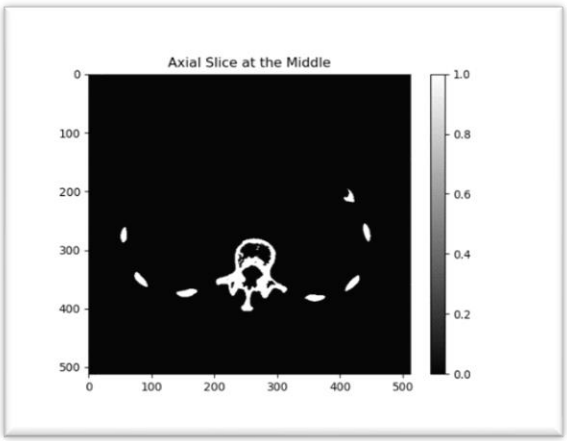
Final i min *& Connectivity Components Results*

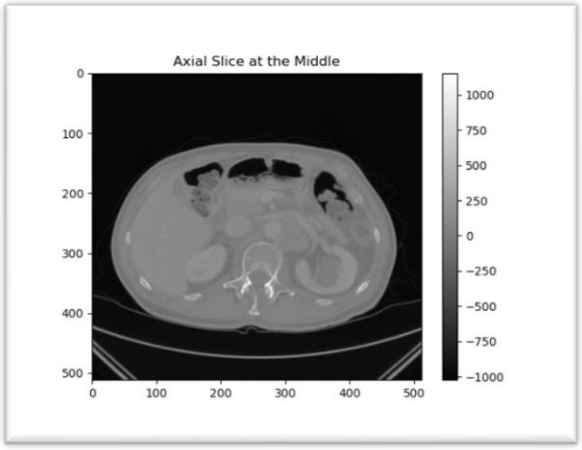
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Py
Working on nifti file name: MIP Data\Case1_CT.nii.gz
Selected i min is: 248
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

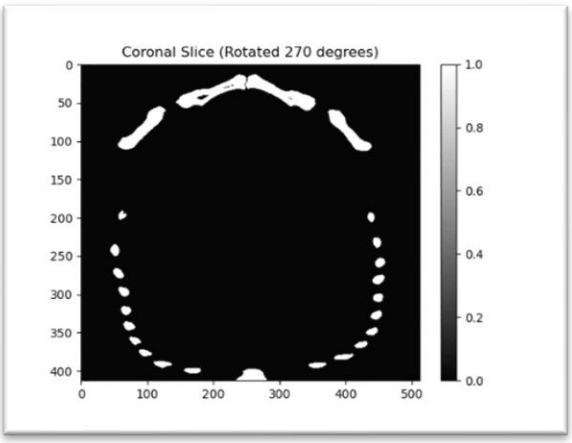
Graph: threshold value (index) VS Number of Connectivity Components

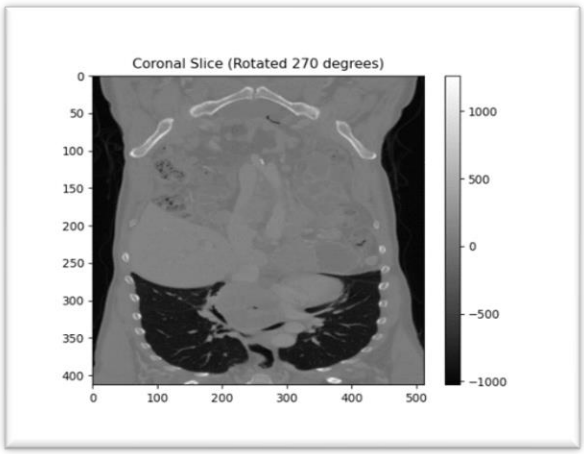


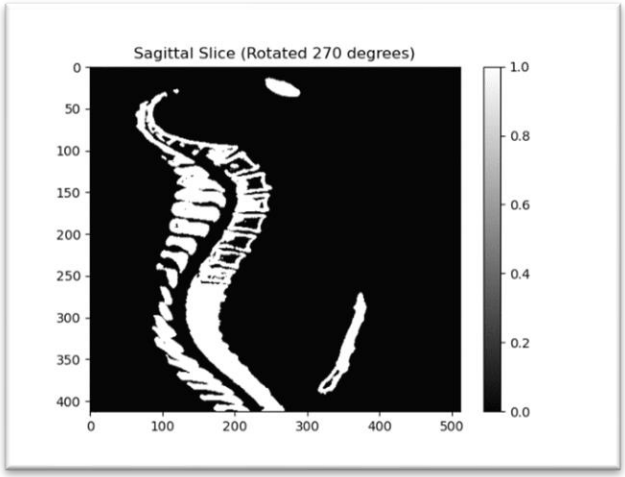
Middle CT Examples of Slices – Case 1

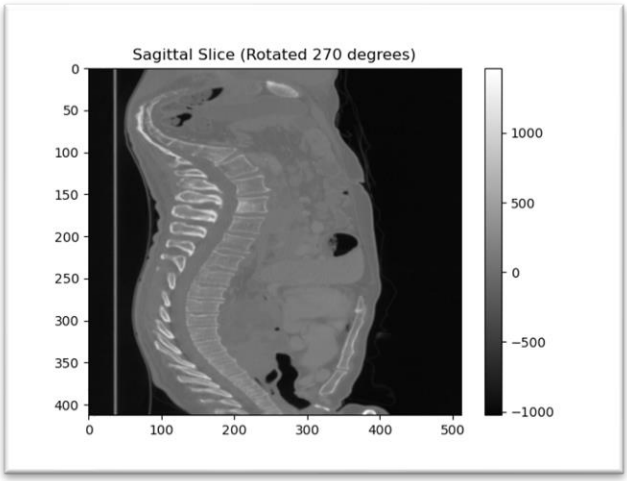










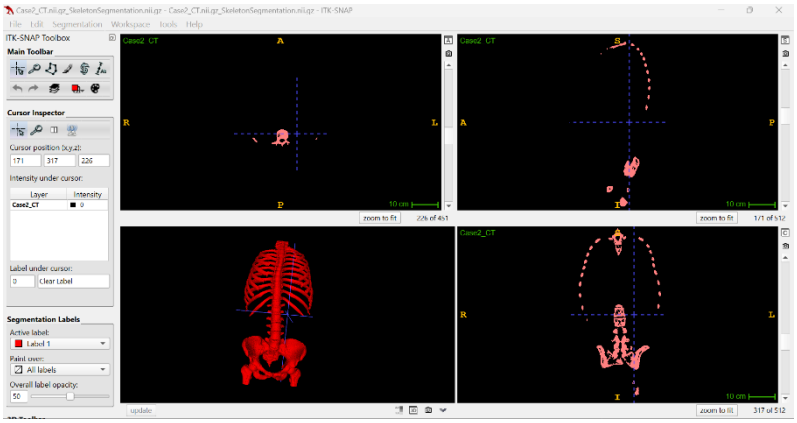


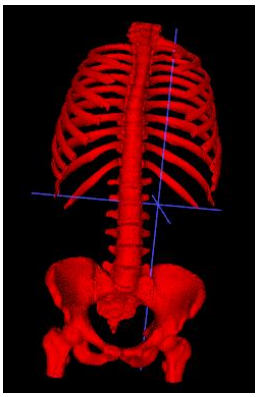
The results are very pleasing and clean in regular high-resolution cases 1-5 – we can observe a clear model of the skeleton in the region of Thorax and Abdomen – Clear Rib Cage, Sternum, Hip Bones and Spine with all vertebras.

I also included Axial, Coronal & Sagittal Slices from the middle of the NIFTI File to demonstrate the threshold selection and post-processing results after filling holed, removing CC, use binary operations from morphology library etc.

Case 2 CT

Segmentation Model Results

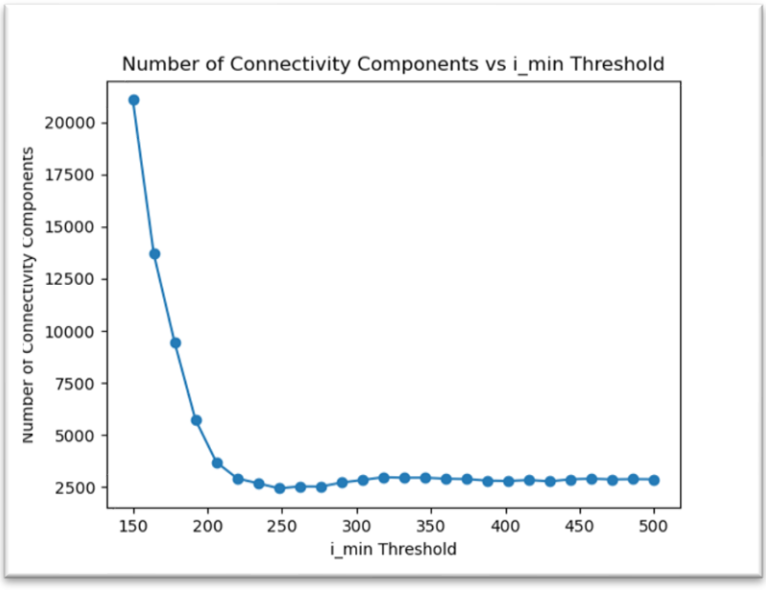




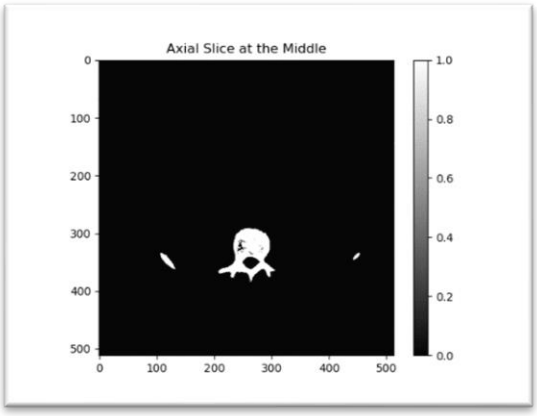
Final i_min & Connectivity Components results

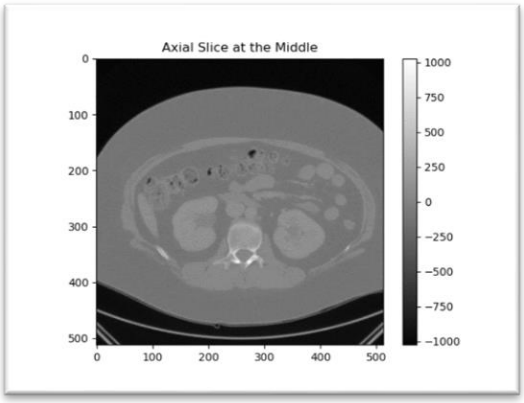
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Py
Working on nifti file name: MIP Data\Case2_CT.nii.gz
Selected i min is: 248
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

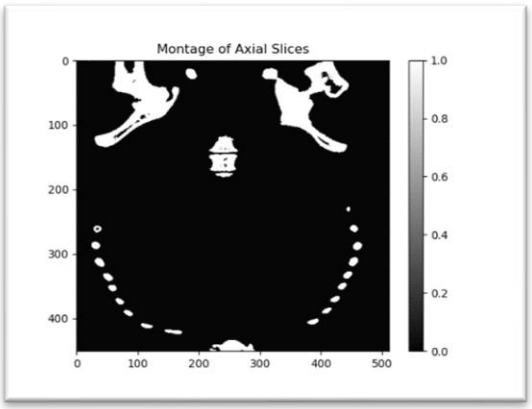
Graph: threshold value (index) VS Number of Connectivity Components

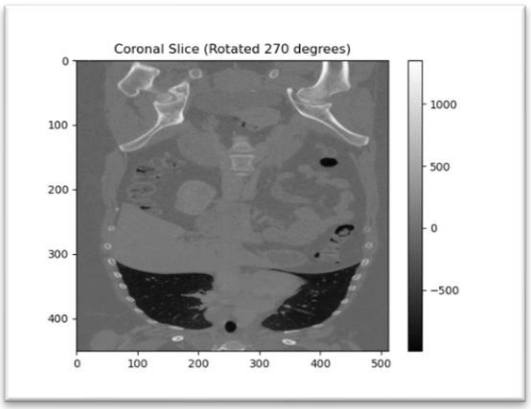


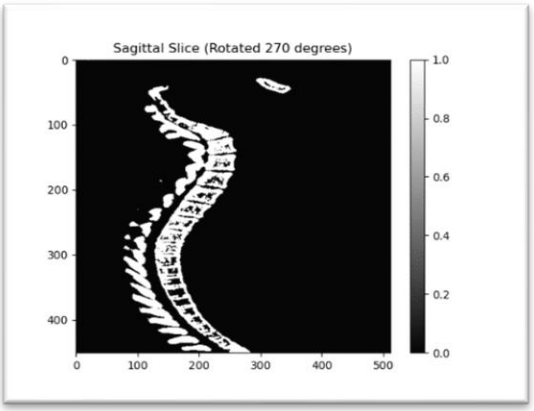
Middle CT Examples of Slices – Case 2

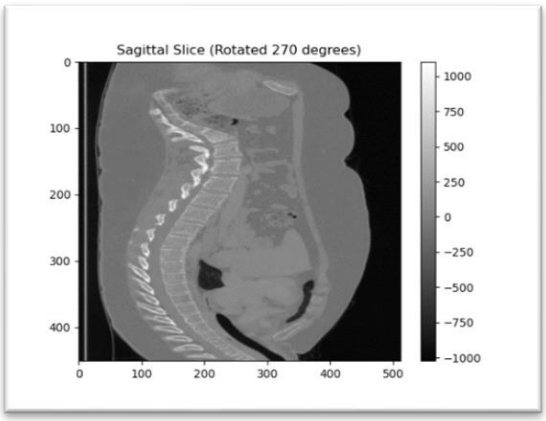






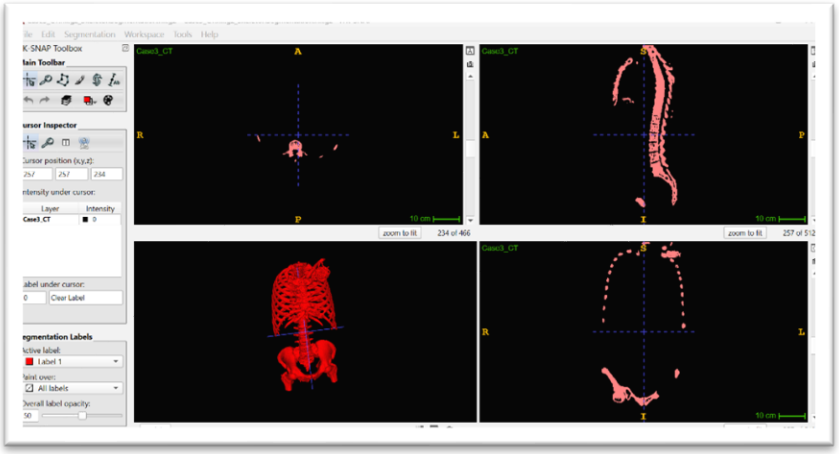


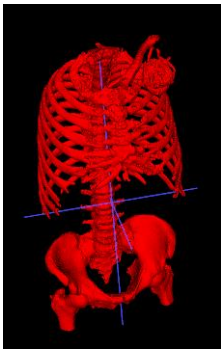




Case 3 CT

Segmentation Model Results

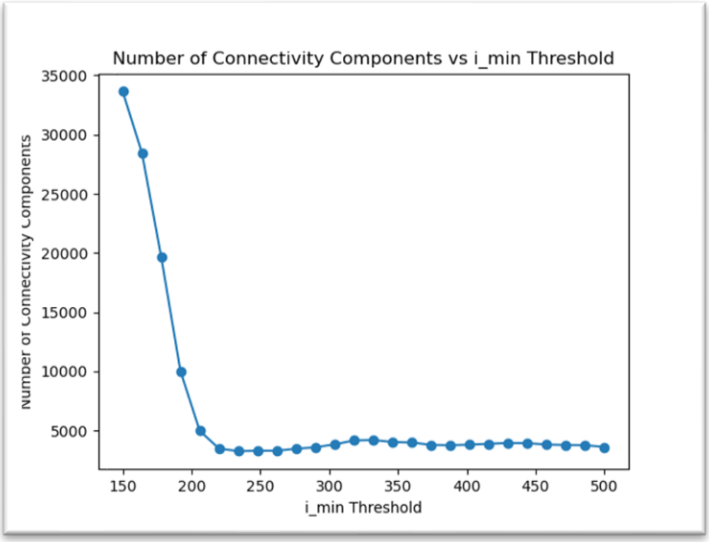




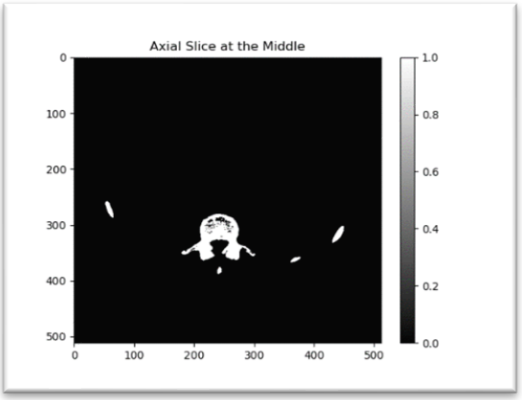
Final i_min & Connectivity Components Results

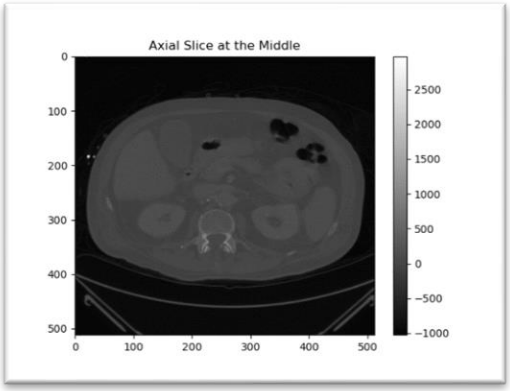
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Py
Working on nifti file name: MIP Data\Case3_CT.nii.gz
Selected i min is: 234
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

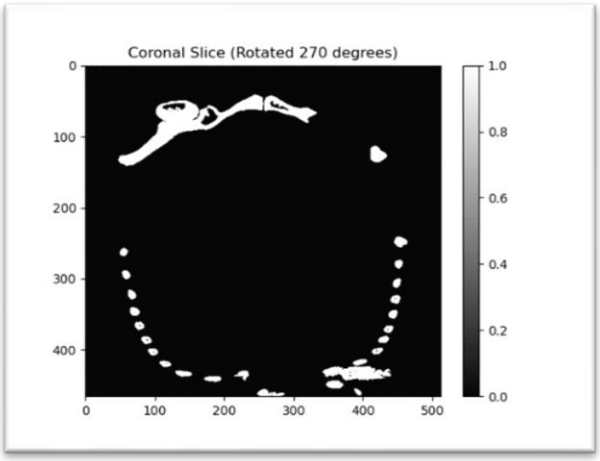
Graph: threshold value (index) VS Number of Connectivity Components

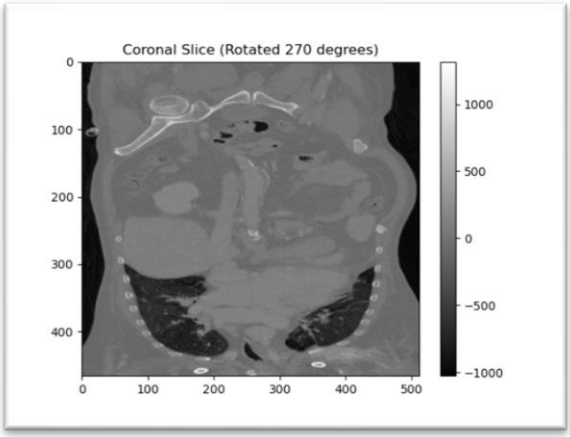


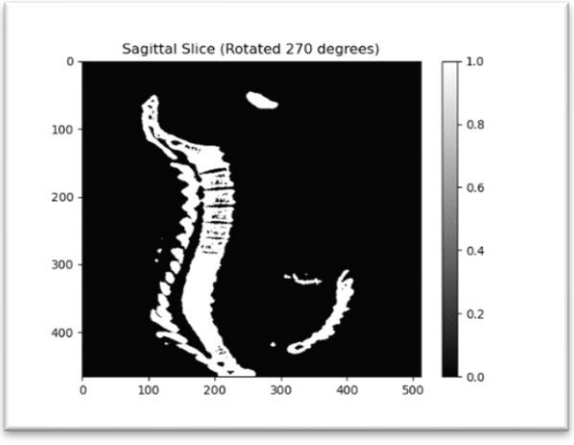
Middle CT Examples of Slices – Case 3

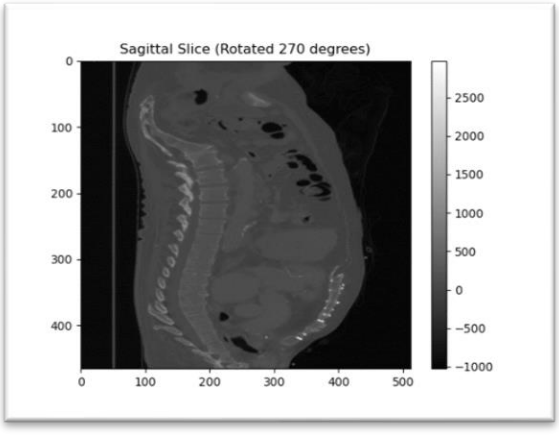






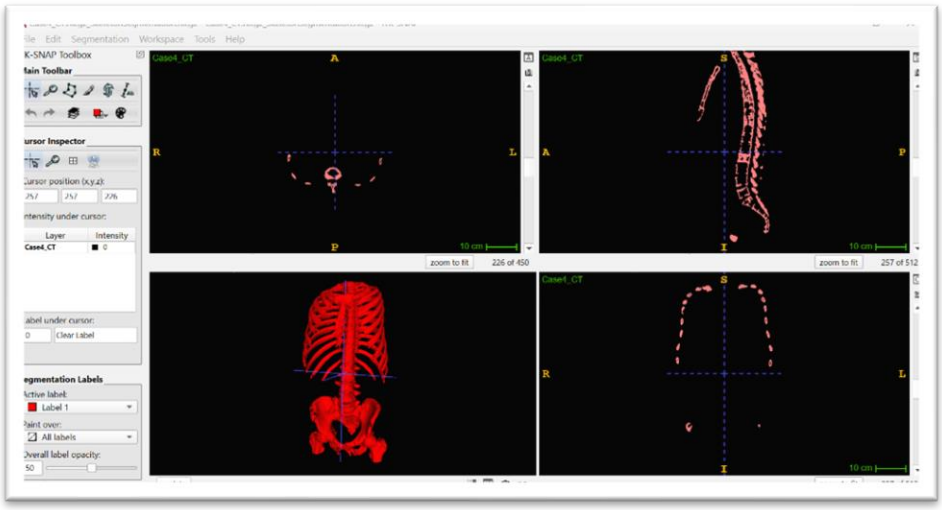


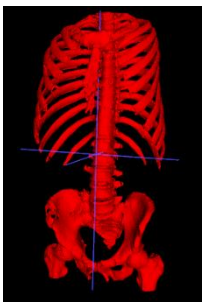




Case 4 CT

Segmentation Model Results

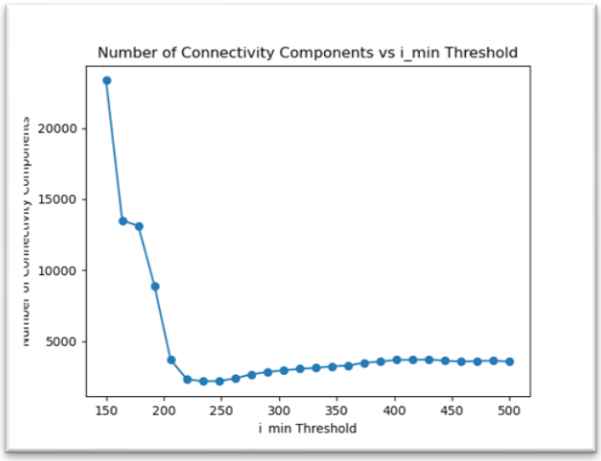




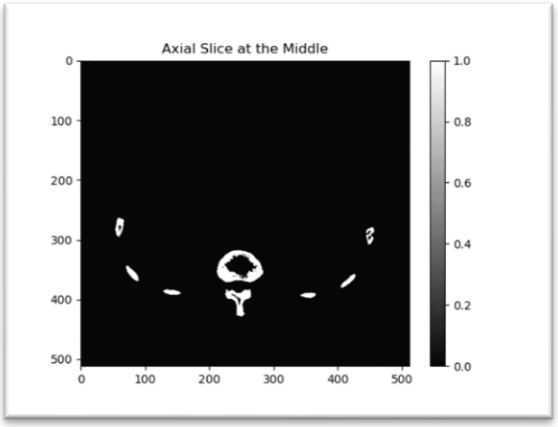
Final i min & *Connectivity Components Results*

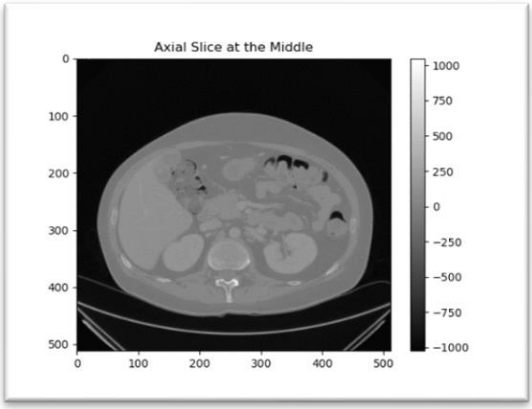
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Py
Working on nifti file name: MIP Data\Case4_CT.nii.gz
Selected i min is: 234
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

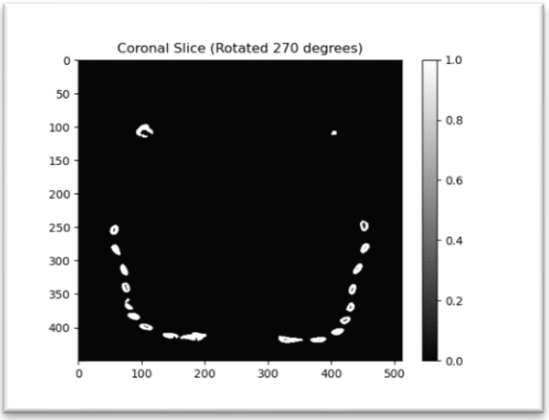
Graph: threshold value (index) VS Number of Connectivity Components

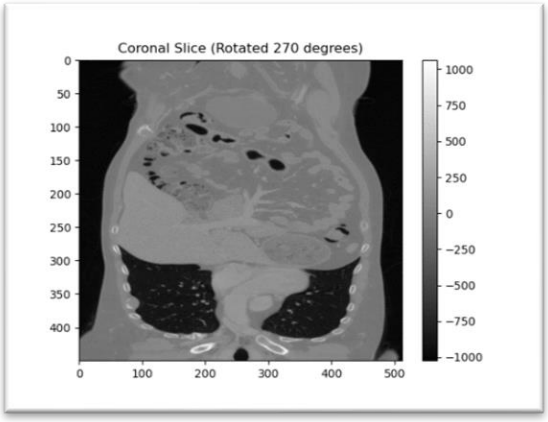


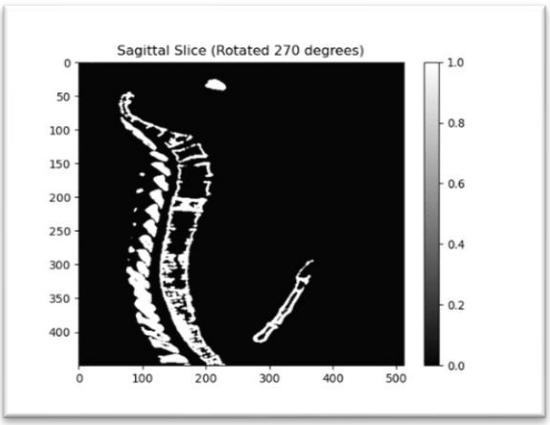
Middle CT Examples of Slices – Case 4

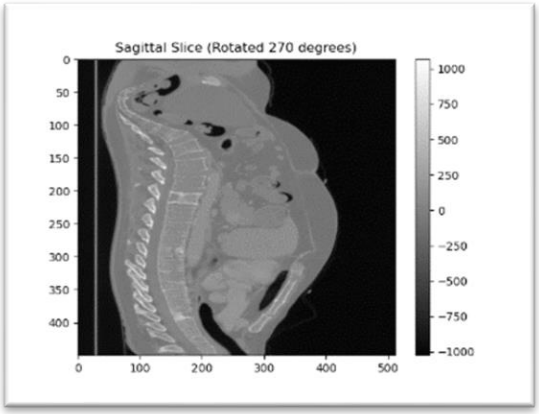






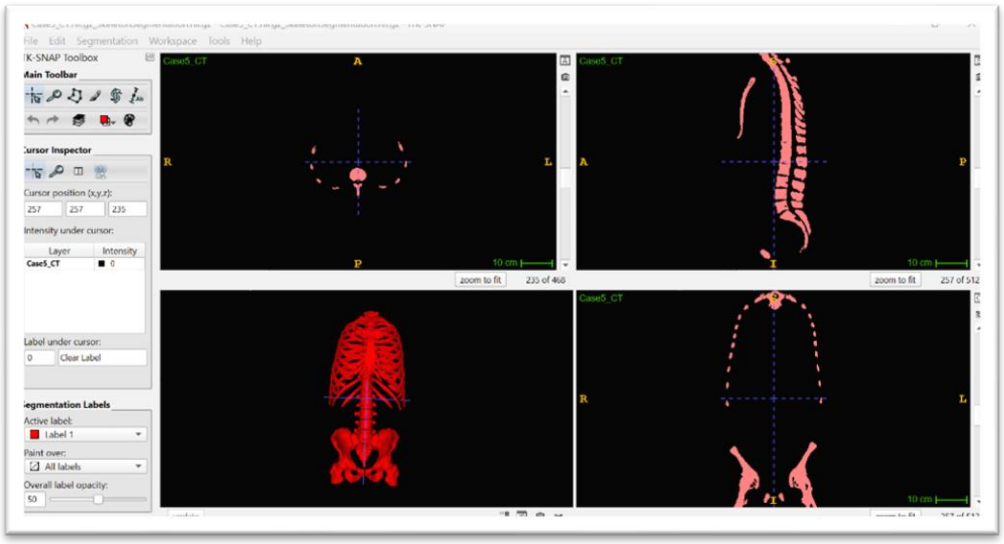


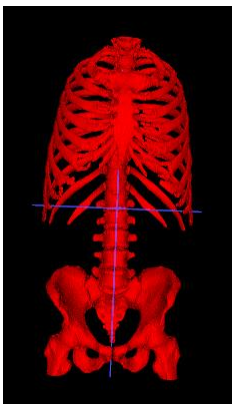




Case 5 CT

Segmentation Model Results

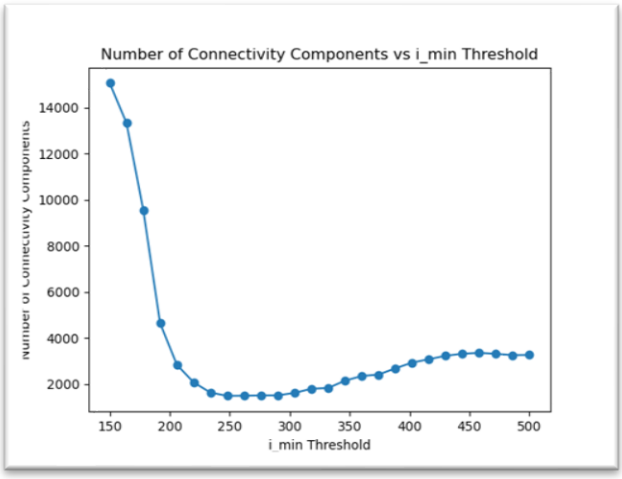




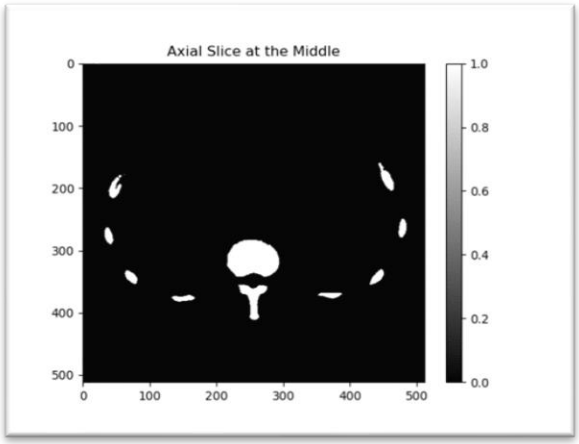
Final i_min & Connectivity Components Results

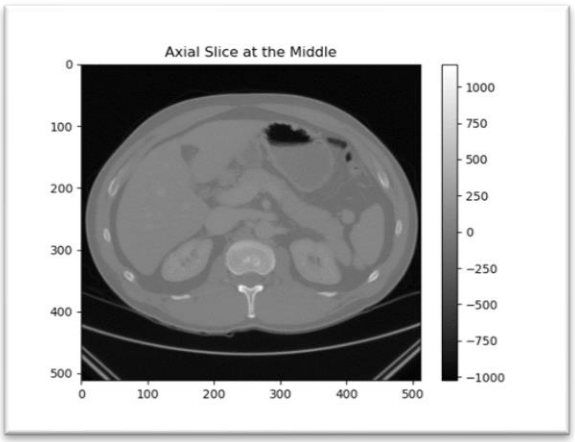
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Py
Working on nifti file name: MIP Data\Case5_CT.nii.gz
Selected i min is: 248
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

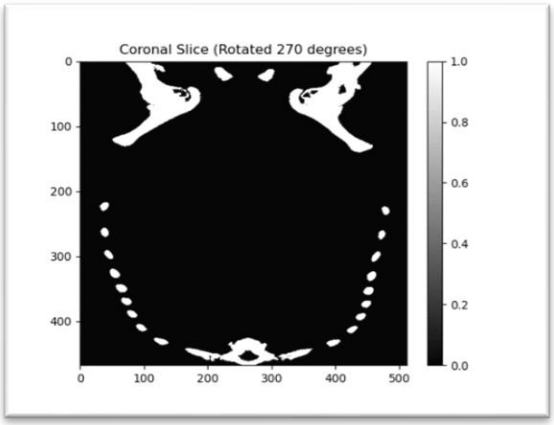
Graph: threshold value (index) VS Number of Connectivity Components

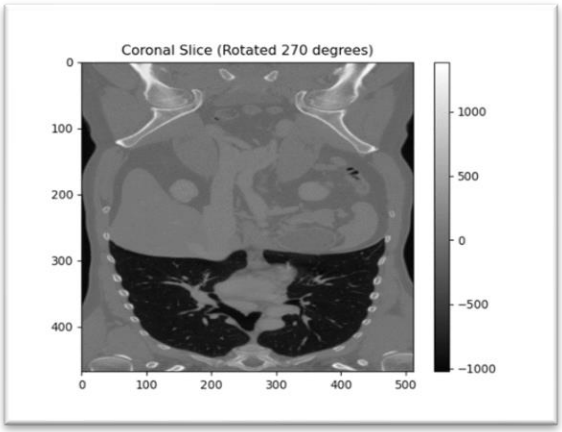


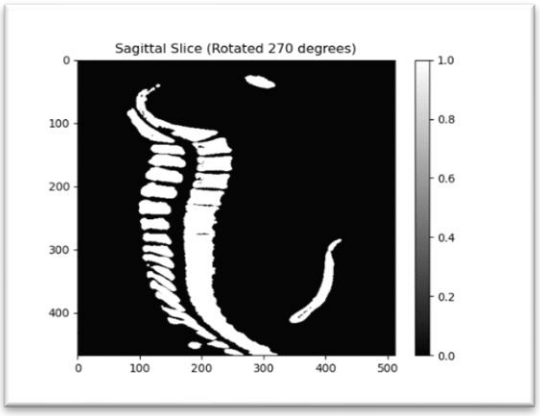
Middle CT Examples of Slices – Case 5

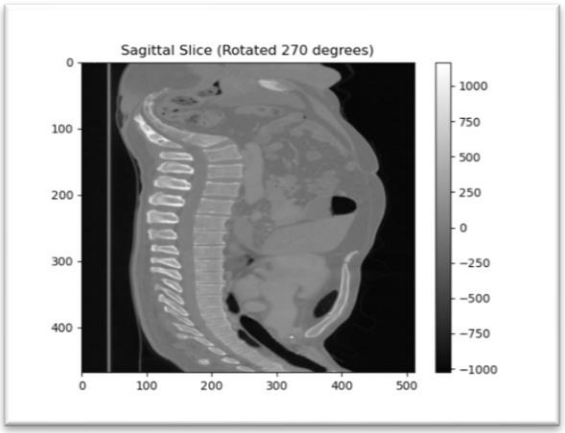






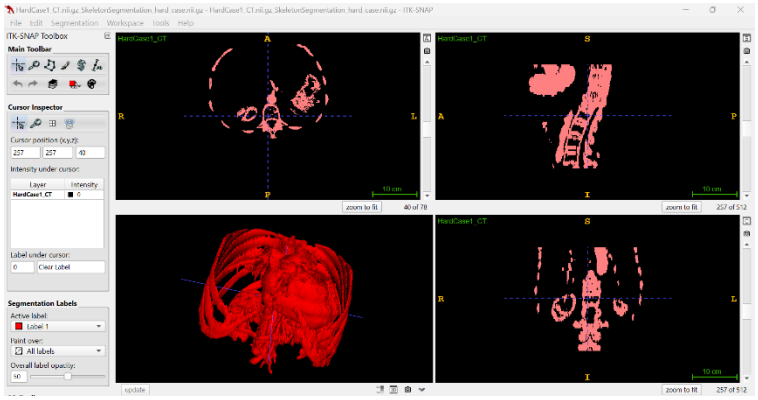


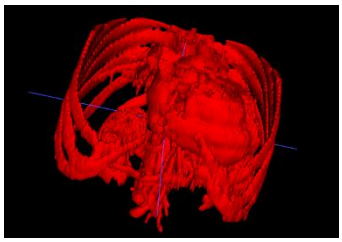




Hard Case 1 CT

Segmentation Model Results

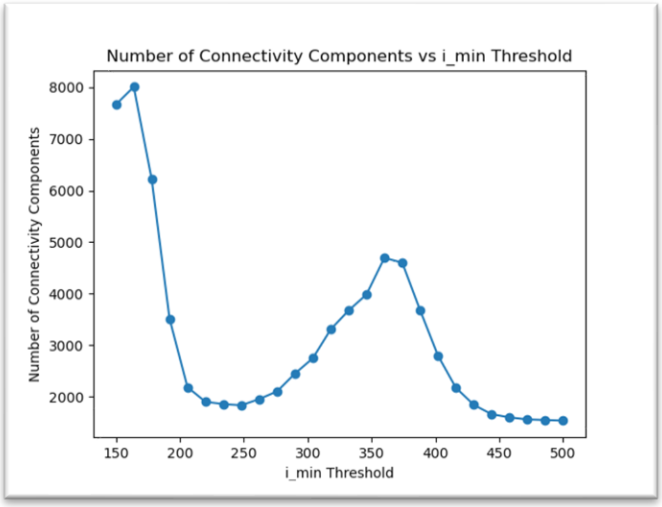




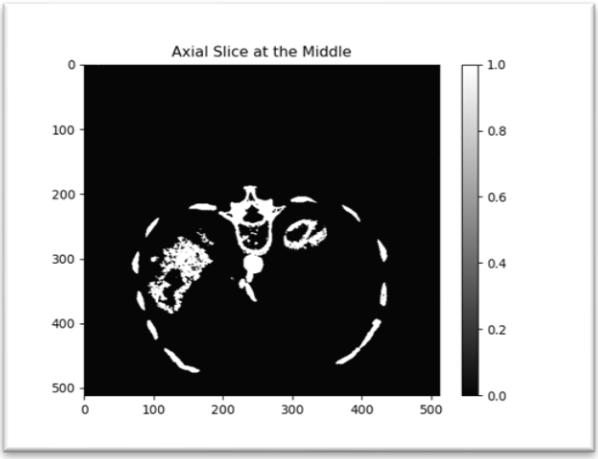
Final i_min & Connectivity Components Results

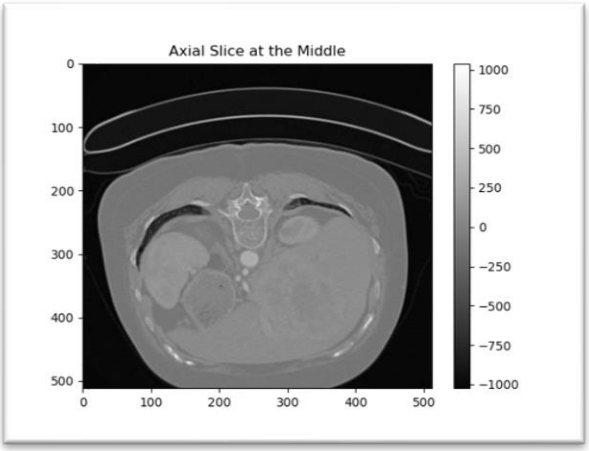
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Pychar
Working on nifti file name: MIP Data\HardCase1_CT.nii.gz
Selected i min is: 500
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

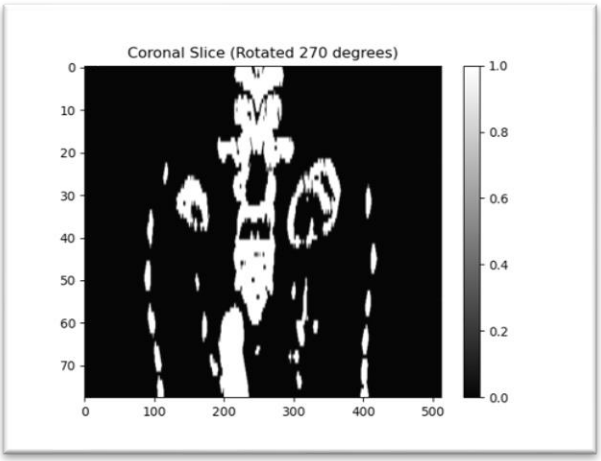
Graph: threshold value (index) VS Number of Connectivity Components

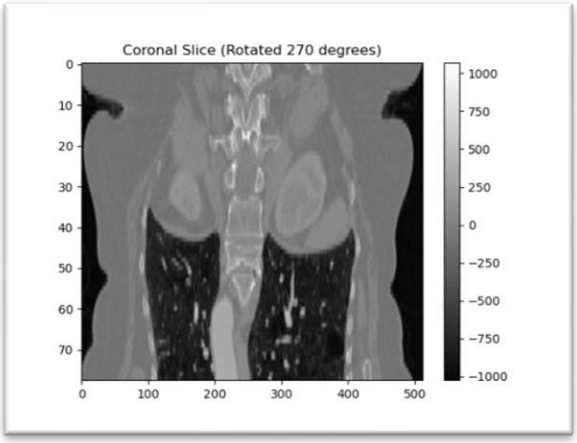


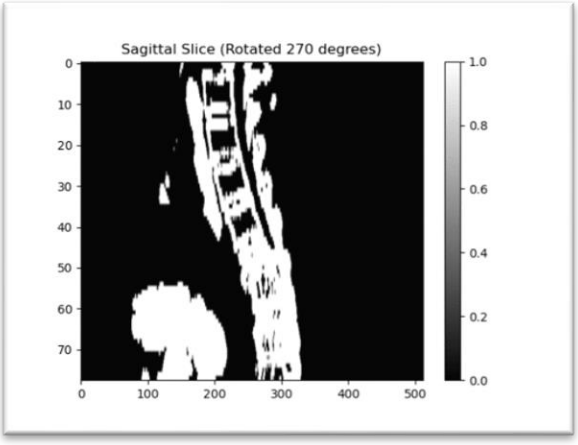
Middle CT Examples of Slices – Case 5 – Results with 1 Component





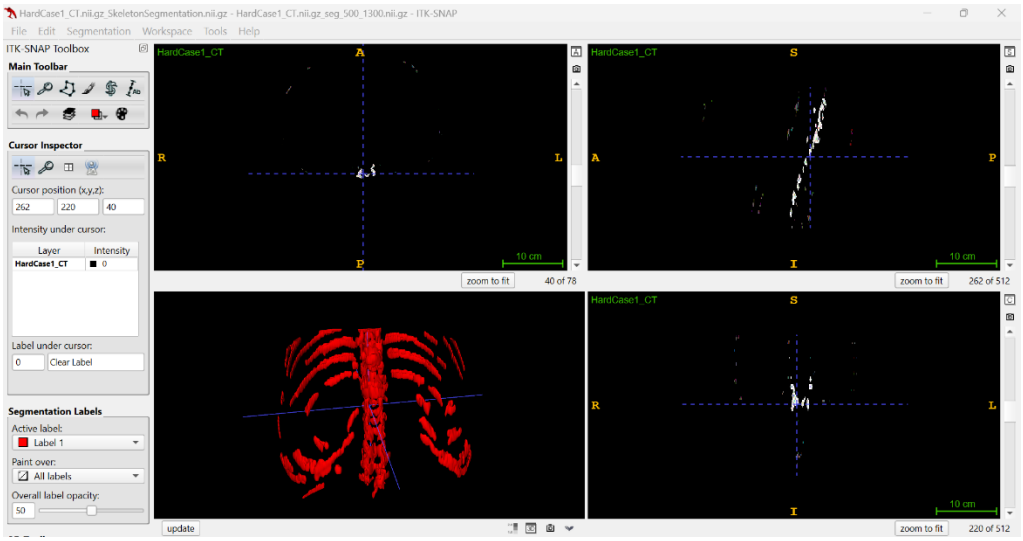


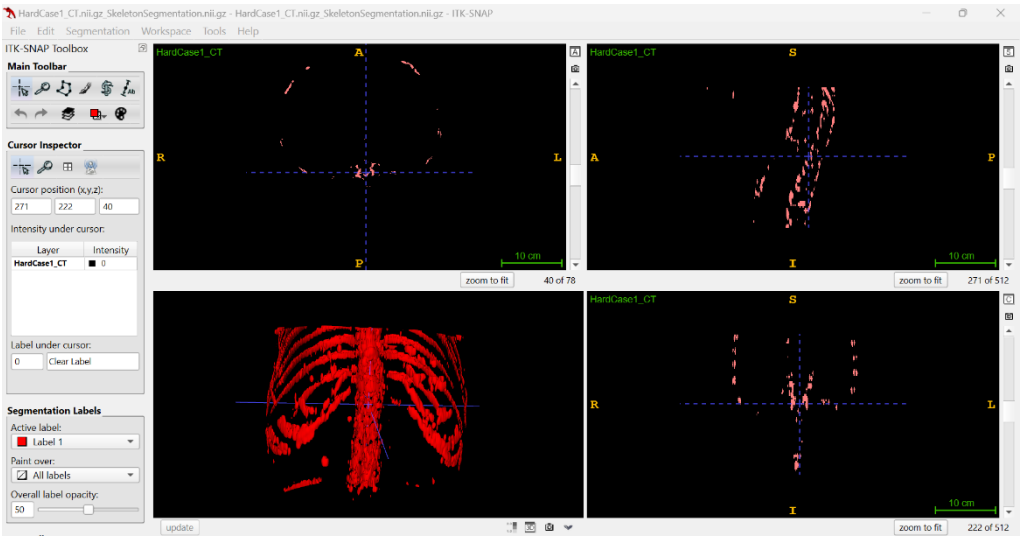




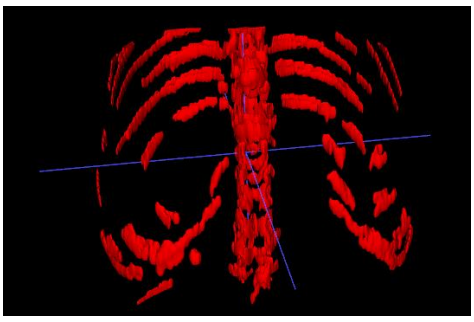
Segmentation Model Results with a few Connectivi Components:

Skeleton Only – good results but more CC

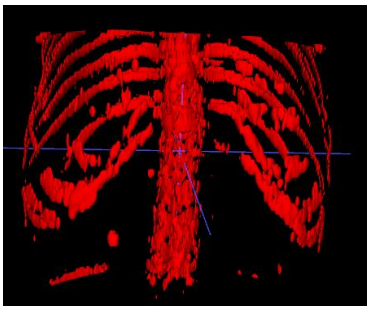




With remove_small_objects (min_size=64)

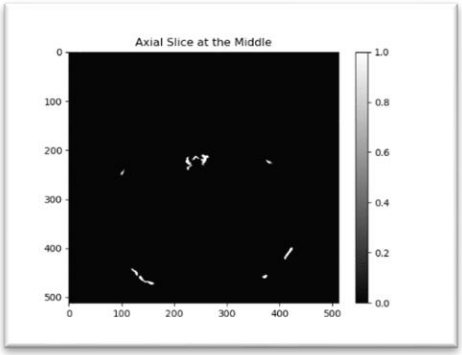


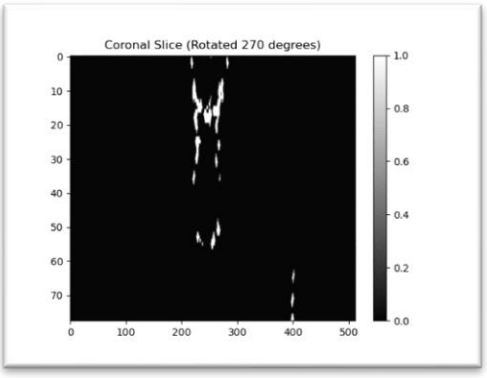
Without remove_small_objects function

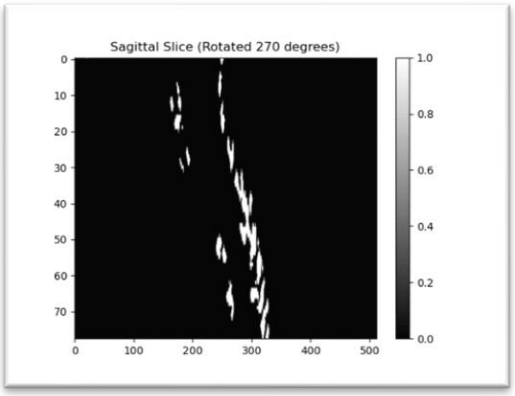


```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Pychar
Working on nifti file name: MIP Data\HardCase1_CT.nii.gz
Selected i min is: 500
Number of connectivity components: 292
Image data converted successfully.

Process finished with exit code 0
```

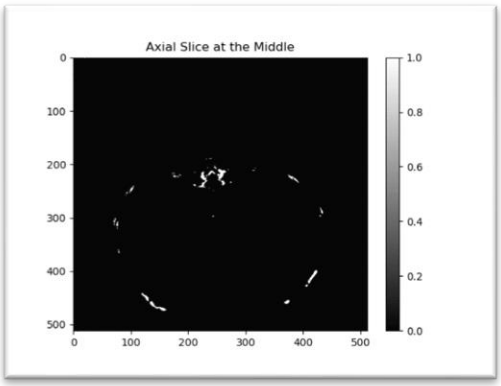
With remove_small_objects (min_size=64)

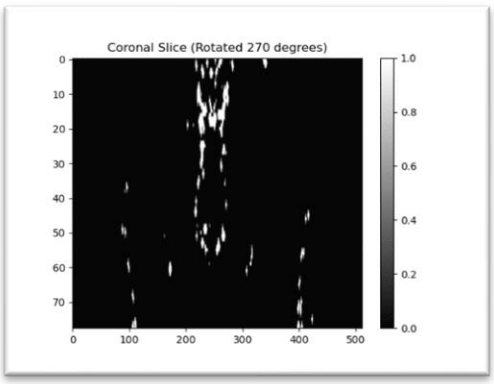


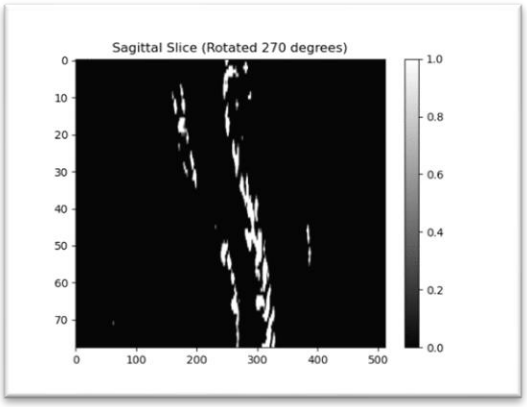




Without remove_small_objects function

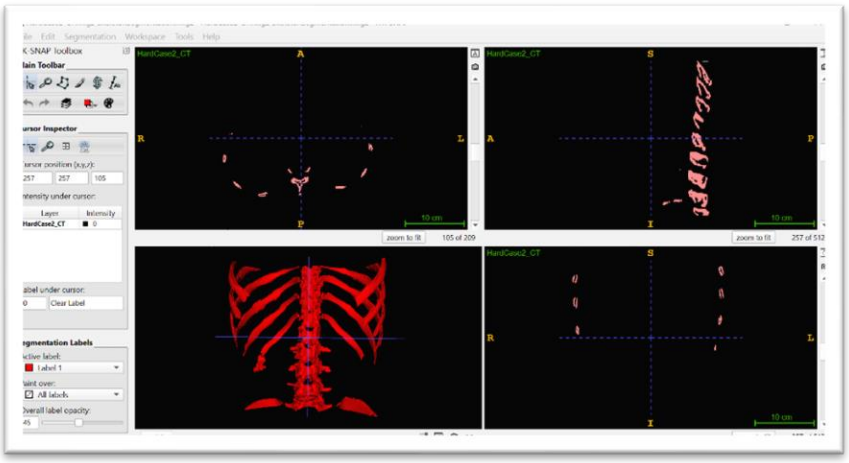


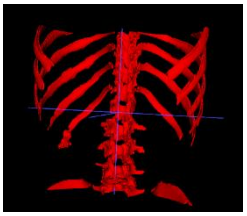




Hard Case 2 CT

Segmentation Model Results

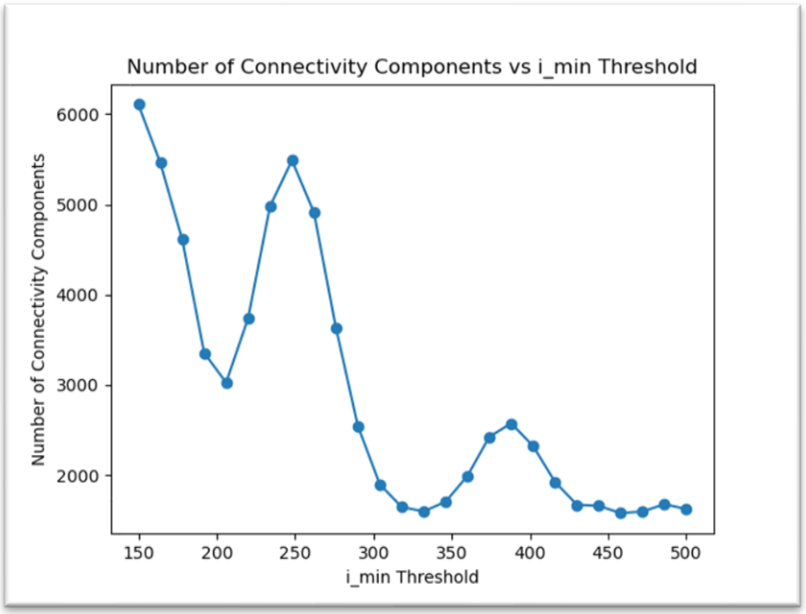




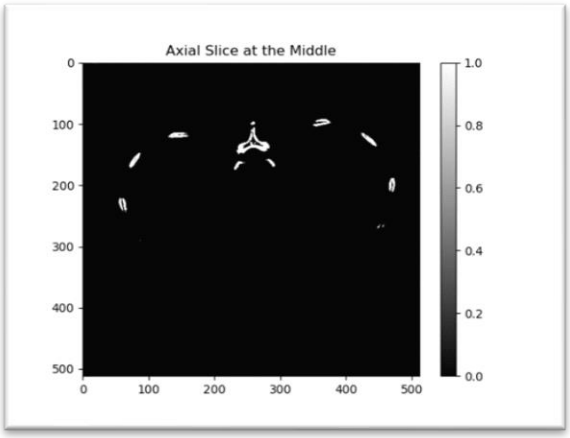
Final i_min & Connectivity Components Results

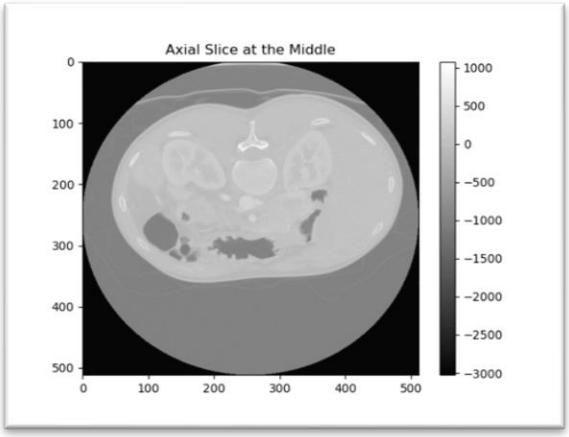
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Pychar
Working on nifti file name: MIP Data\HardCase2_CT.nii.gz
Selected i min is: 458
Number of connectivity components: 16
Image data converted successfully.

Process finished with exit code 0
```

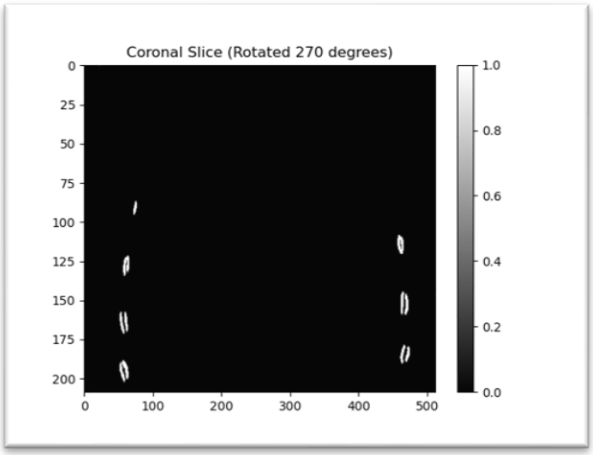
Graph: threshold value (index) VS Number of Connectivity Components

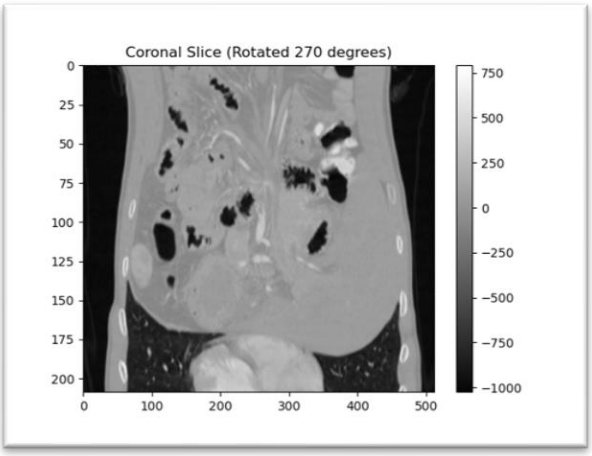


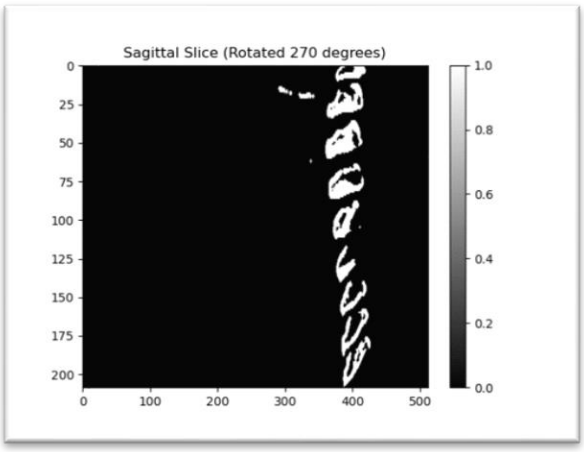
Middle CT Examples of Slices – Hard Case 2

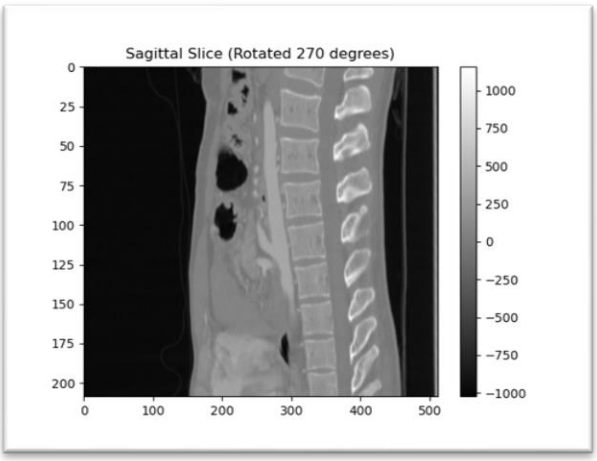






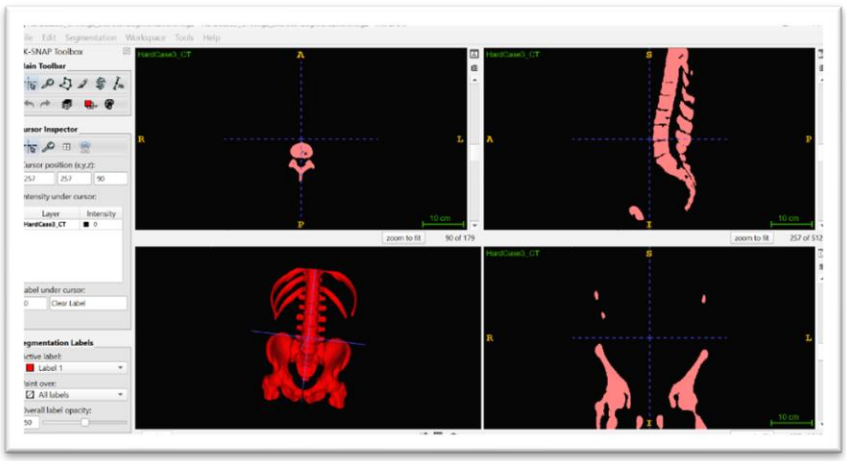


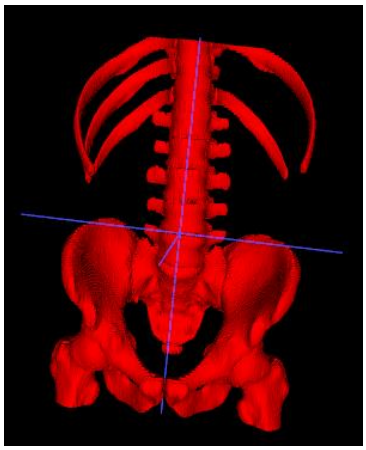




Hard Case 3 CT

Segmentation Model Results

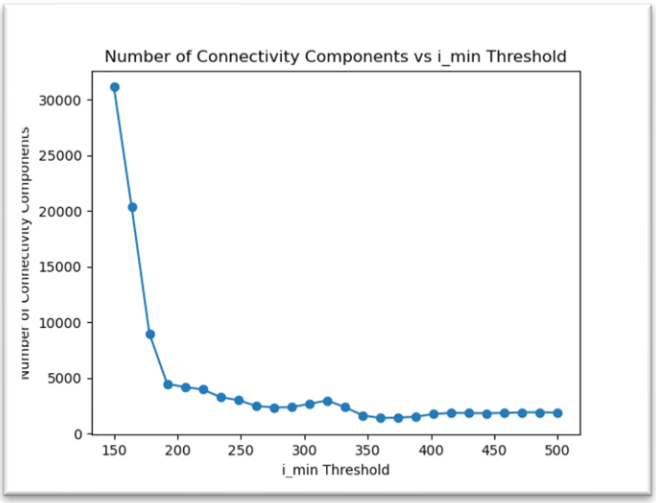




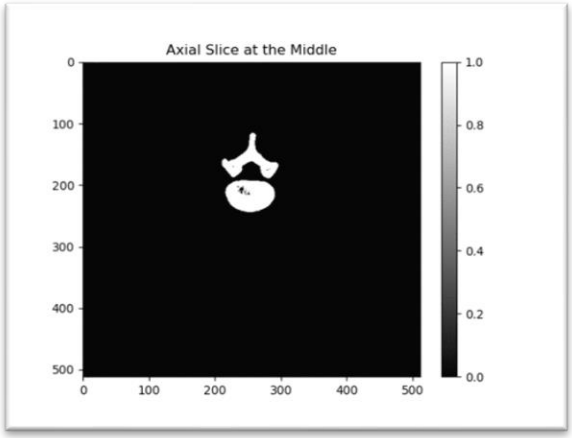
Final i_min & Connectivity Components Results

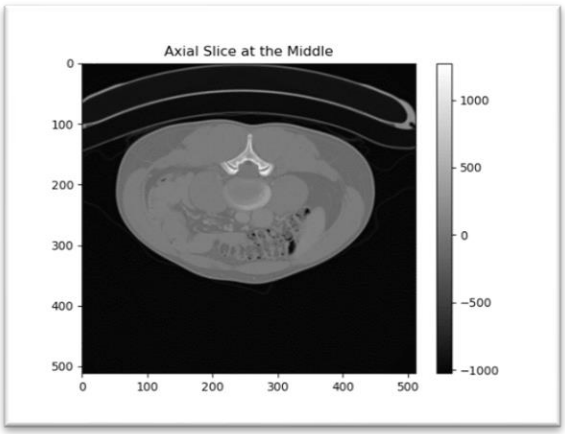
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Pychar
Working on nifti file name: MIP Data\HardCase3_CT.nii.gz
Selected i min is: 360
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

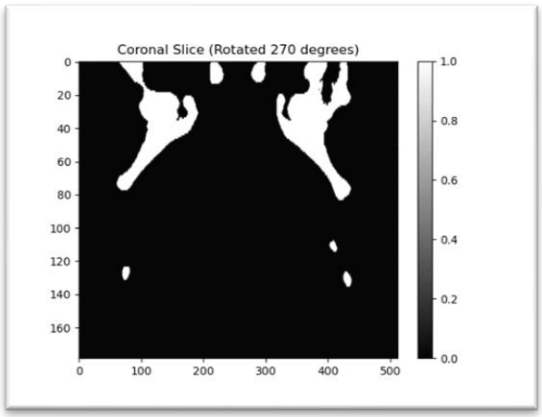
Graph: threshold value (index) VS Number of Connectivity Components

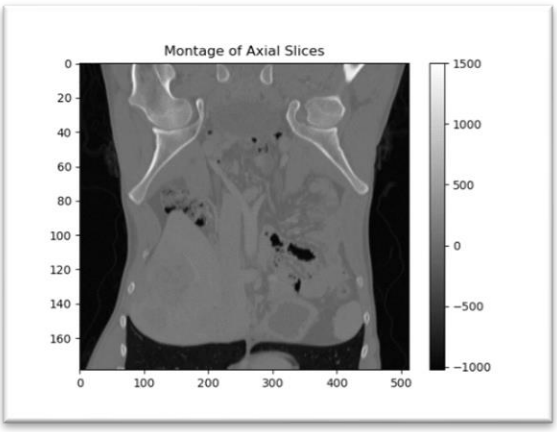


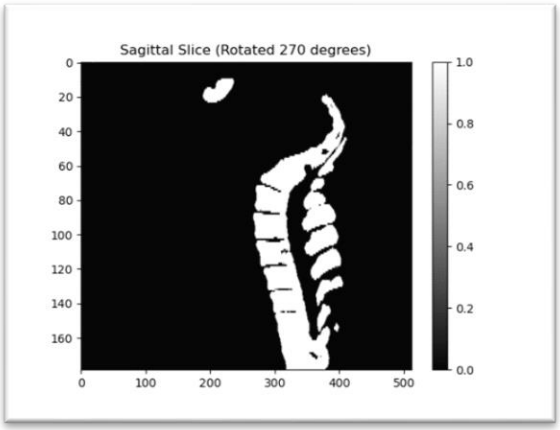
Middle CT Examples of Slices – Hard Case 3

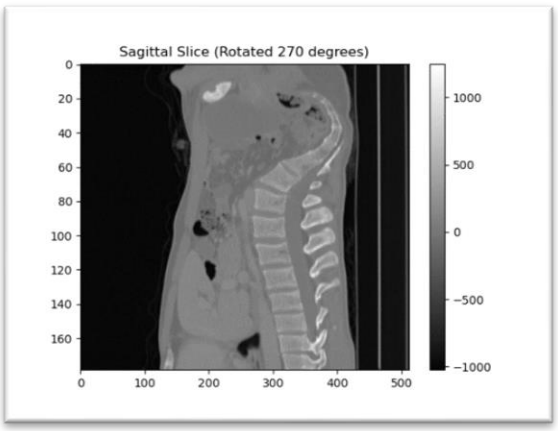






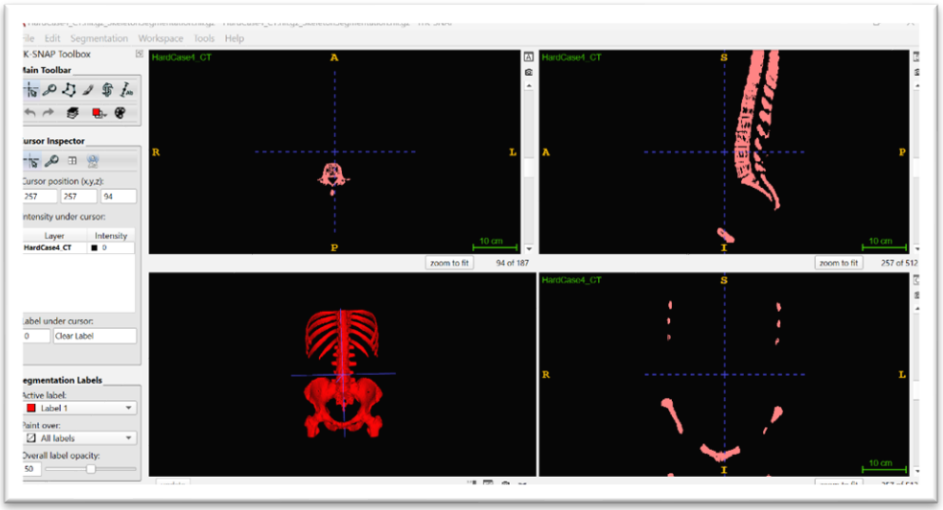


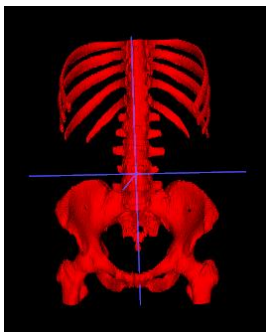




Hard Case 4 CT

Segmentation Model Results

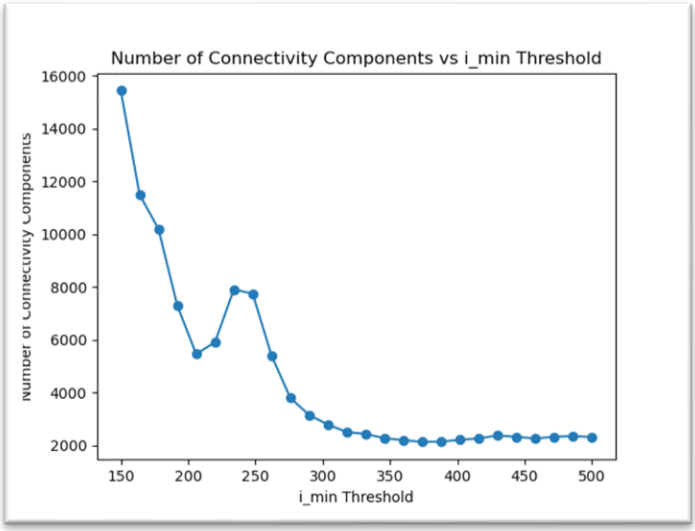




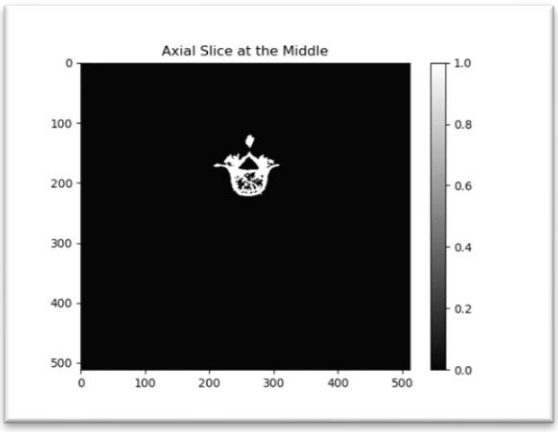
Final i_min & Connectivity Components Results

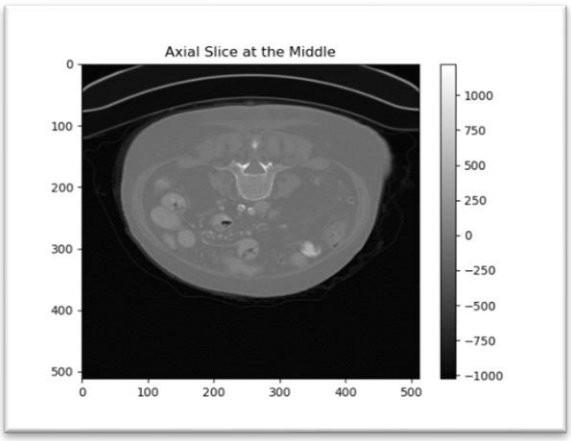
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Pychar
Working on nifti file name: MIP Data\HardCase4_CT.nii.gz
Selected i min is: 374
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

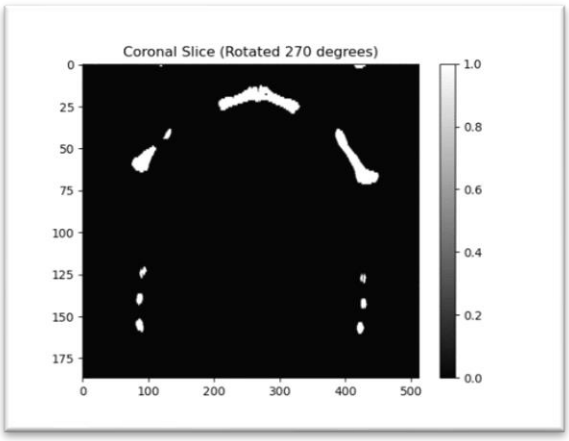
Graph: threshold value (index) VS Number of Connectivity Components

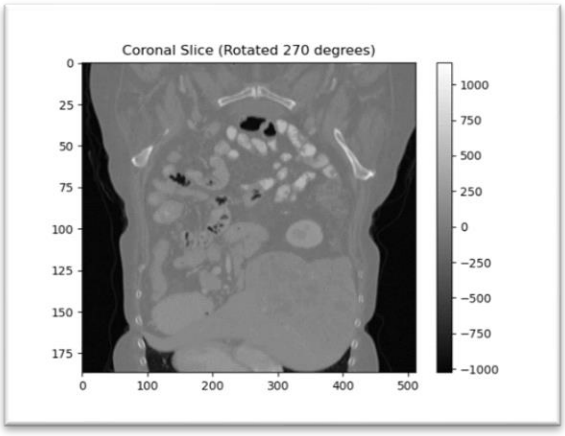


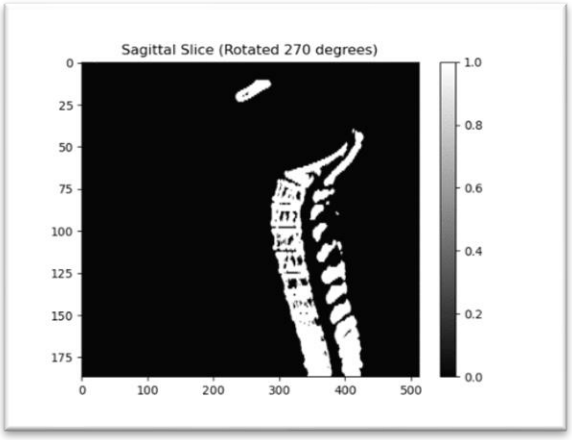
Middle CT Examples of Slices – Hard Case 4

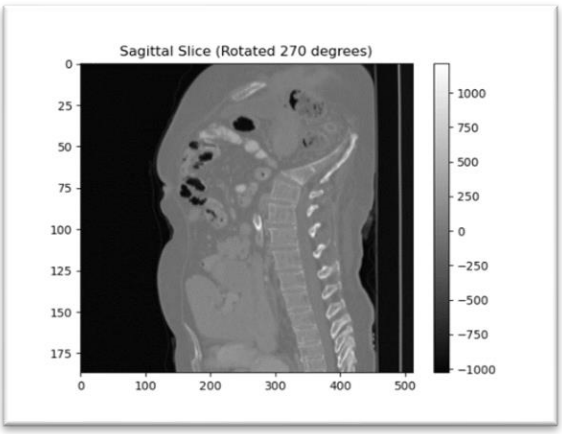










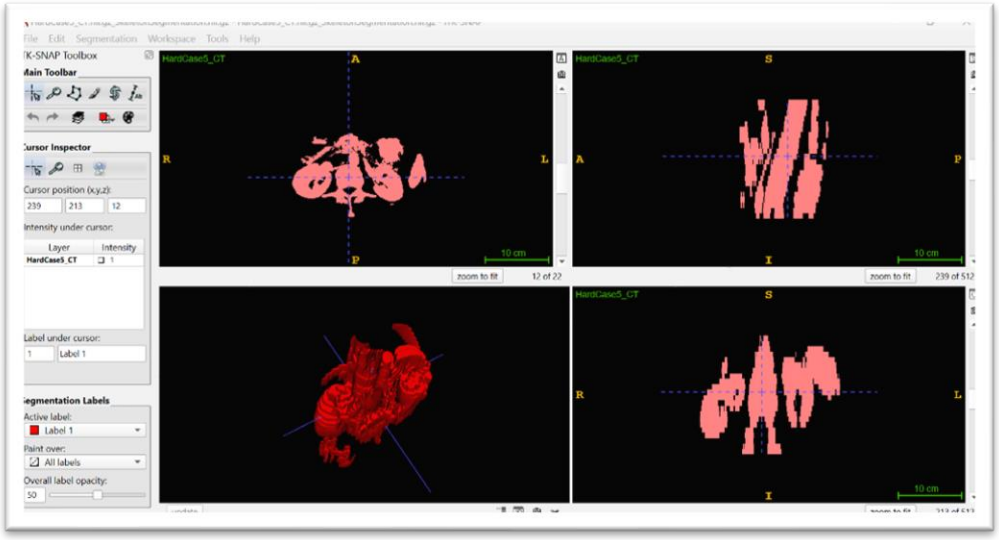


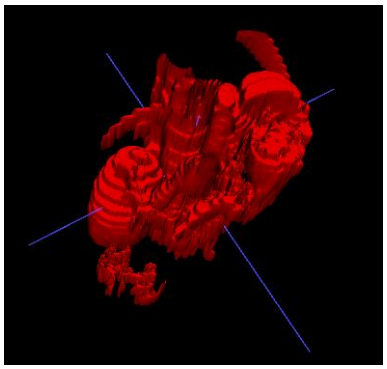
Hard Case 5 CT – With Regular Case Segmentation Code

Segmentation Results

With 1 Connectivity Component We can see organs and very Bad Resulution,

I tried cleaning it, but managed to get only with few CC only skeleton model

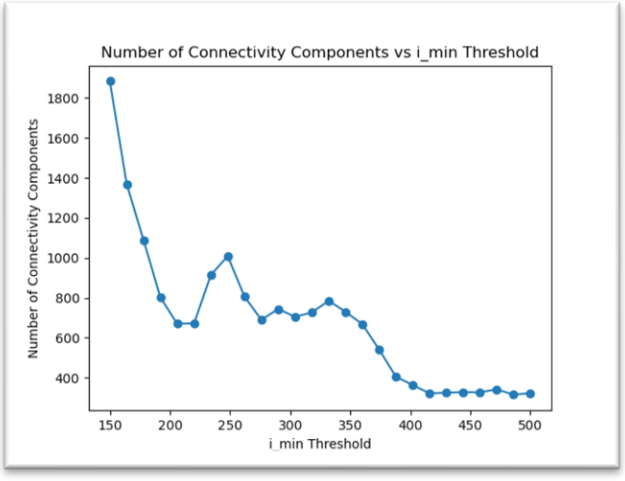




Final i min *& Connectivity Components Results*

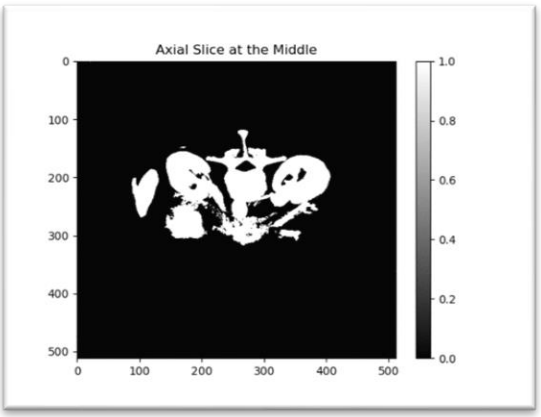
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Pychar
Working on nifti file name: MIP Data\HardCase5_CT.nii.gz
Selected i min is: 486
Number of connectivity components: 1
Image data converted successfully.

Process finished with exit code 0
```

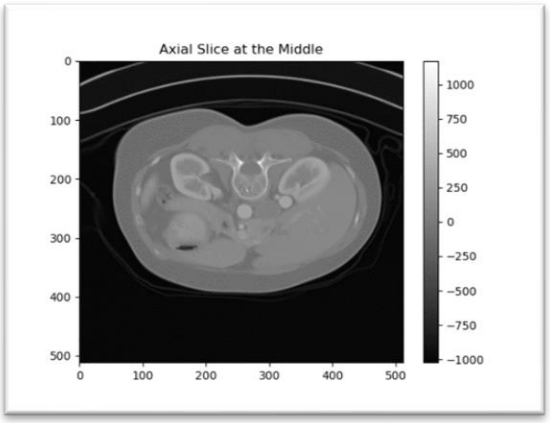
Graph: threshold value (index) VS Number of Connectivity Components

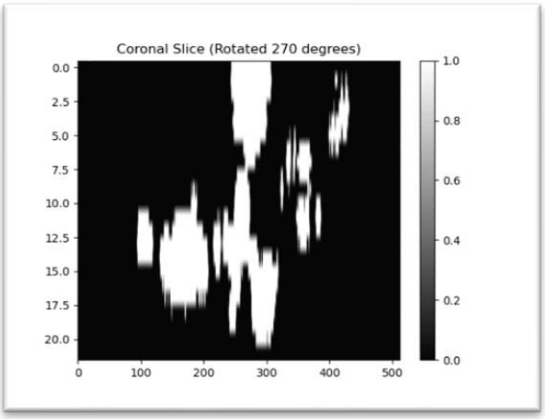


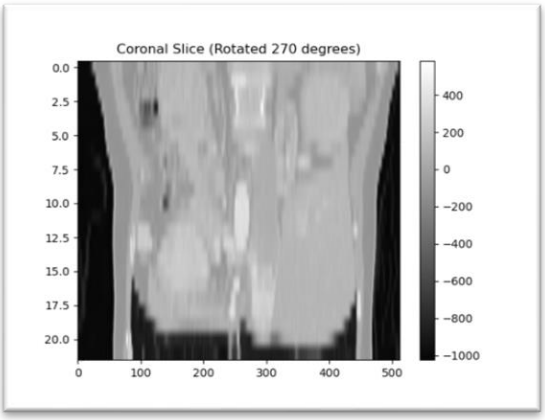
Middle CT Examples of Slices – Hard Case 5:

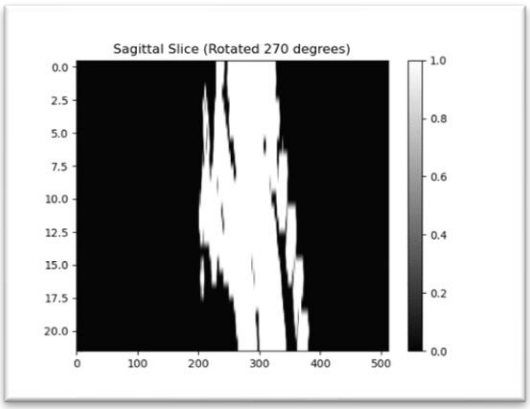
1 *Connectivity Component with Bad Seperation*

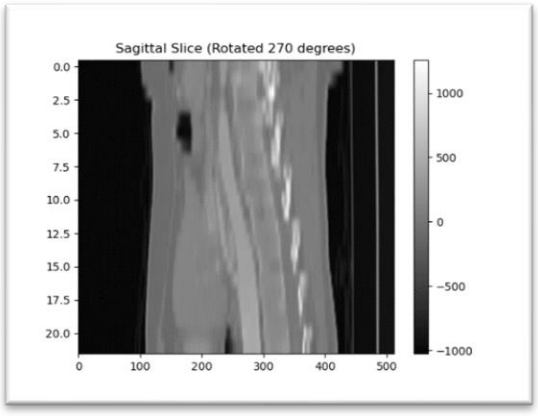






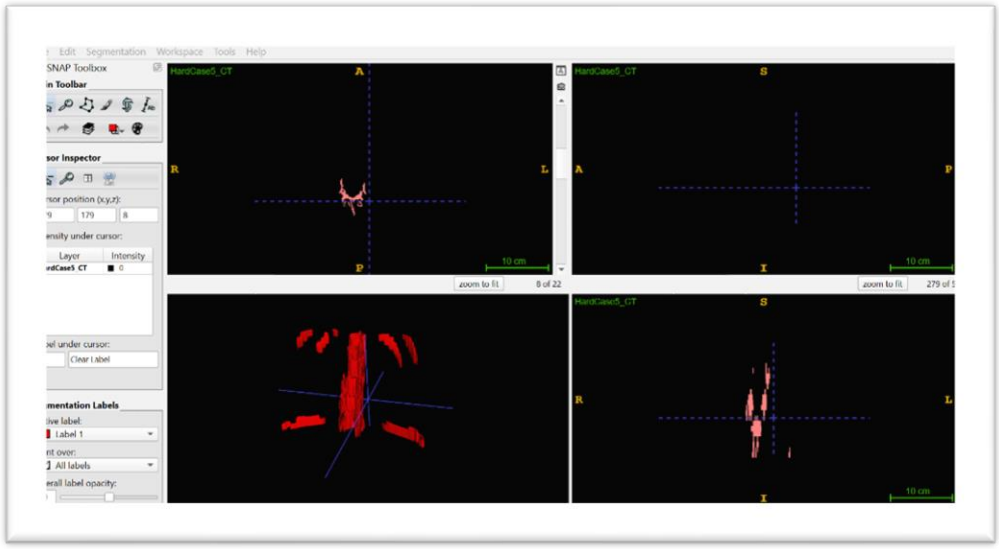


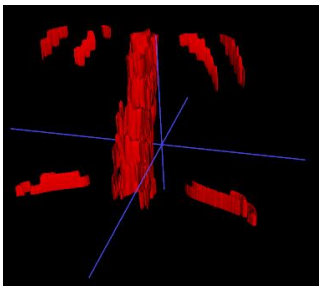




Hard Case 5 CT – With Hard Case Segmentation Adjusted Code

Segmentation Results

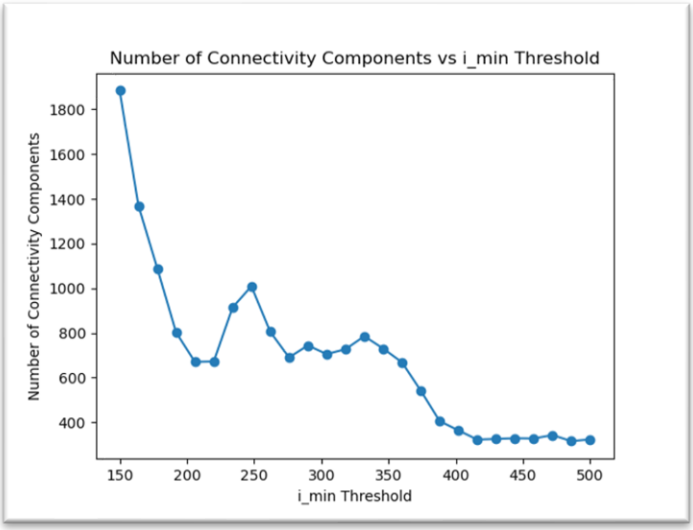




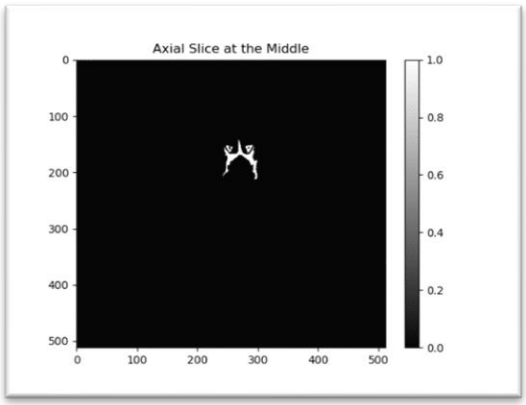
Final i_min & Connectivity Components Results

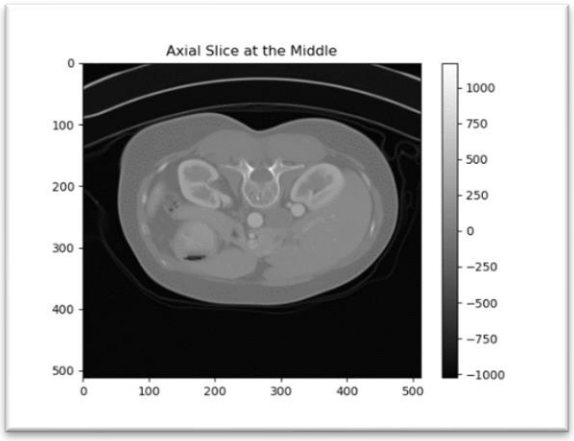
```
C:\Users\amith\anaconda3\python.exe C:/Users/amith/Pychar
Working on nifti file name: MIP Data\HardCase5_CT.nii.gz
Selected i min is: 486
Number of connectivity components: 11
Image data converted successfully.

Process finished with exit code 0
```

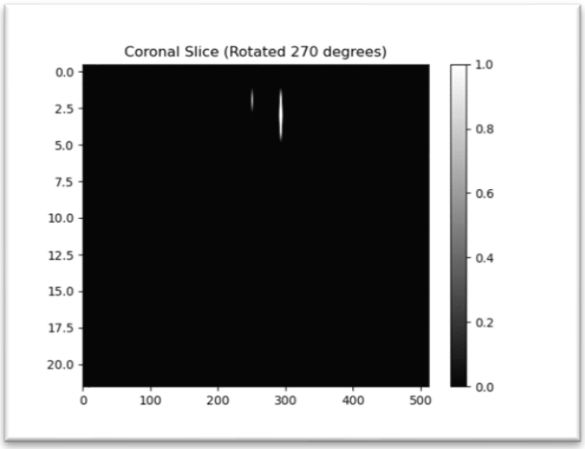
Graph: threshold value (index) VS Number of Connectivity Components

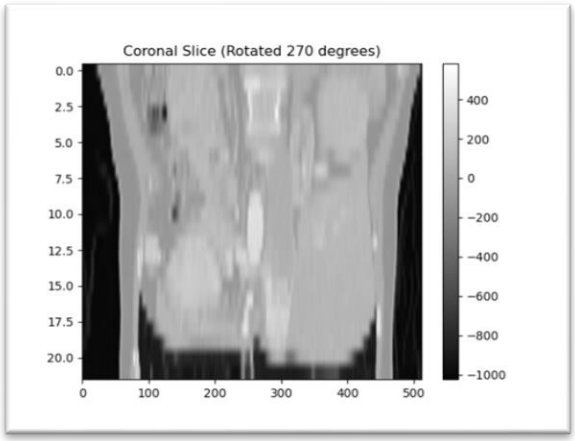


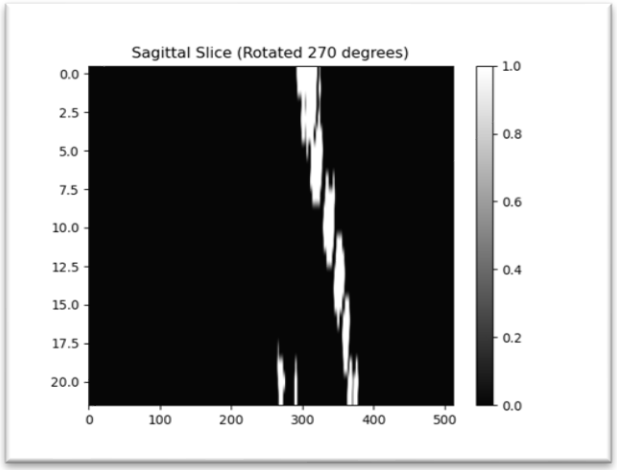
Middle CT Examples of Slices – Hard Case 5 Adjusted Algorithm for hard cases

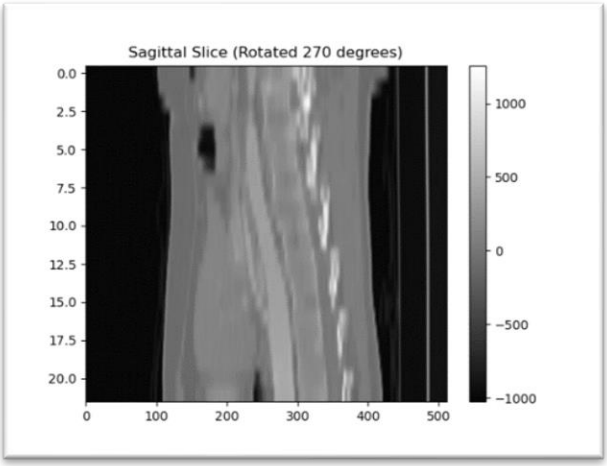












The results were very bad for regular cases segmentation code with one component because of the bad resolution so I changed the code, such that I will get a few CC but without organs that aren't part of the skeleton, I tried a few versions and played with parameters – eventually this is what I got and it's only skeleton but bits of it – its hard to produce good results in cases 1 and 5 of the Hard Cases because of low resolution and a low amount of Axial Slices especially in Hard Case 5.

Library Functions use & Explanation for specific use

Library functions I used from skimage.morphology and scipy.ndimage skimage.measure:

```
import nibabel as nib
import matplotlib.pyplot as plt
import numpy as np
from scipy.ndimage import label as lab, binary_dilation
from scipy.ndimage import binary_closing, binary_fill_holes, find_objects
from skimage.morphology import remove_small_objects, binary_dilation, \
    binary_opening, binary_closing, remove_small_holes, \
    binary_erosion
from skimage.measure import label
```

I used mainly functions in this order to clean the regular cases of the skeleton's CT in the following order:

1. **remove_small_objects:** A morphological operation uses to get rid of small CC groups.

2. **binary_fill_holes:** a morphological operation that uses to fill the shape of the skeleton model if it has small holes in it.

3. **binary_closing:** a morphological operation uses to close small holes or gaps in binary images combining dilation and erosion processes.

4.**binary_dilation:** a morphological operation uses to expand the boundaries of foreground (white) regions in a binary image.

5**. binary_fill_holes:** use the function again to fill holes in the model and make it more complete.

6.**remove_small_holes**: Sometimes I preferred using this function. A morphological operation used in image processing to eliminate small holes (foreground regions) within the larger connected components of a binary image.

The justifications for the fact I used these morphological operations in this specific order after finding the best threshold is as follows:

- First, I wish to remove all small objects that represents small CC Group that may be n organ, a vessel – vein/artery, a nerve or any other part which is not part of the skeleton.
- Afterwards I wanted to fill the holes that may be created in the model.
  I tried both Binary Closing and Binary Opening and Closing was better than opening for the exercise's purposes.
- I use dilation in order to fill and take also pixels that may be part of the skeleton but may have been counted out in the process.
- Finally, I want to fill small holes so the skeleton model would be complete.

For hard cases I did a 'Trial and Error' process because each case required different techniques and order of morphological in order to get good results.

Some of the cases like Hard Case 1 & Hard Case 5 required to give result of more than 1 CC because results with 1 CC were constructing models with organs in

addition to the skeleton which disqualifies the purpose of the exercise – we were requested to remove any organs with threshold and morphological operations so as a result we will get the most clear & clean model of the skeleton for the specific case – This is the reason that for low resolution CT scans with low number of Axial Slices the number of CC is bigger to try to eliminate every part that is not a bone (part of the skeleton).

Methods' Description

Method*: *segmentation_by_th

```
def segmentation_by_th(nifti_file_path, i_min, i_max):
    """
    This function is given as inputs a grayscale NIFTI file (.nii.gz) and
    two integers – the minimal and maximal thresholds. The function
    generates a segmentation NIFTI file of the same dimensions, with a
    binary segmentation – 1 for voxels between i_min and i_max ,
    0 otherwise. This segmentation NIFTI file is saved under the name
    <nifti_file>_seg_<i_min >_<i_max >.nii.gz.
    :param nifti_file: Path to the input NIFTI file.
    :return: The function returns 1 if successful,
    and 0 otherwise. The function raises descriptive errors when returning 0.
    """
    try:
        # Load the NIFTI file
        nifti_data = nib.load(nifti_file_path)

        # Getting a pointer to the data
        img_data = nifti_data.get_fdata()

        # Create boolean masks for values below i_min and above or equal to
        # i_max
        low_values_flags = img_data < i_min
        high_values_flags = img_data >= i_max

        # Set values outside the range [i_min, i_max] to 0 and 1
        img_data[low_values_flags] = 0
        img_data[high_values_flags] = 1

        # Create a new NiftiImage
        segmented_nifti = nib.Nifti1Image(img_data, nifti_data.affine)

        # Save the segmented NIFTI file
        output_filename = f"{nifti_file_path}_seg_{i_min}_{i_max}.nii.gz"
        nib.save(segmented_nifti, output_filename)
        return 1  # Successfully segmented and saved

    except Exception as e:
        print(f"Error: {e}")
        return 0  # Return 0 if an error occurs
```

Method*: *skeleton_th_finder

```
def skeleton_th_finder(nifti_file):
    """
    Performs thresholding, segmentation, and post-processing to extract the
    skeleton CT cases with preferably 1 Connectivity Component for good
    resolution CT scans and a few for bad resolution CT scans to avoid Model
    Construction with organs in addition to the skeleton,
    scans.
    :param nifti_file: Path to the input NIFTI file.
    :return: The selected optimal threshold value. The function raises
    descriptive errors when returning 0.
    """
    try:
        # Range of candidate i_min thresholds
        i_min_range = range(INITIAL_THRESHOLD, FINAL_THRESHOLD, STEP)
```

```
        num_components = []

        # Iterate over candidate i_min thresholds
        for i_min_candidate in i_min_range:
            # Perform segmentation using segmentation_by_th
            segmentation_by_th(nifti_file, i_min_candidate, 1300)

            # Load the segmented NIFTI file
            seg_file_path = f"{nifti_file}_seg_{i_min_candidate}_1300.nii.gz"
            segmented_data = nib.load(seg_file_path).get_fdata()

            # Count the number of connectivity components
            labeled_array, num_labels = lab(segmented_data)

            # Append the number of components to the list
            num_components.append(num_labels)

        # Plot the results
        plt.plot(i_min_range, num_components, marker='o')
        plt.xlabel('i_min Threshold')
        plt.ylabel('Number of Connectivity Components')
        plt.title('Number of Connectivity Components vs i_min Threshold')
        plt.show()

        # Calculate i_min Threshold final value
        i_min_final = INITIAL_THRESHOLD + STEP * np.argmin(num_components) \
            if num_components[0] != np.min(num_components) \
            else INITIAL_THRESHOLD
        print("Selected i_min is: {}".format(i_min_final))

        # Perform post-processing to get a single connectivity component
        segmentation_by_th(nifti_file, i_min_final, 1300)

        # Load the segmented NIFTI file after post-processing
        seg_file_path_post = f"{nifti_file}_seg_{i_min_final}_1300.nii.gz"
        segmented_data_post = nib.load(seg_file_path_post).get_fdata()
        post_processed_data = segmented_data_post
        if "HardCase" in nifti_file:
            # Remove small objects
            if "1" in nifti_file:
                post_processed_data = binary_fill_holes(
                    binary_closing(post_processed_data))
                post_processed_data = binary_dilation(post_processed_data)
            if "2" in nifti_file:
                post_processed_data = remove_small_objects(
                    segmented_data_post.astype(bool), min_size=1200)
                post_processed_data = binary_fill_holes(
                    binary_closing(post_processed_data))
                post_processed_data = binary_dilation(post_processed_data)
            if "3" in nifti_file:
                post_processed_data = binary_dilation(
                    segmented_data_post)
                post_processed_data = binary_closing(
                    post_processed_data)
                post_processed_data = binary_fill_holes(post_processed_data)
                post_processed_data = remove_small_objects(
                    post_processed_data.astype(bool), min_size=5000)
                # Apply morphological operations for post-processing
                post_processed_data = binary_dilation(
                    binary_fill_holes(post_processed_data))
                # Remove small objects
                post_processed_data = remove_small_objects(
                    post_processed_data.astype(bool), min_size=5000)
            if "4" in nifti_file:
                post_processed_data = binary_dilation(
                    segmented_data_post)
                post_processed_data = binary_closing(
                    post_processed_data)
                post_processed_data = binary_fill_holes(post_processed_data)
                post_processed_data = remove_small_objects(
                    post_processed_data.astype(bool), min_size=5000)
                # Apply morphological operations for post-processing
```

```
                post_processed_data = binary_dilation(
                    binary_fill_holes(post_processed_data))
                # Remove small objects
                post_processed_data = remove_small_objects(
                    post_processed_data.astype(bool), min_size=5000)
            if "5" in nifti_file:
                post_processed_data = remove_small_objects(
                    segmented_data_post.astype(bool), min_size=64)
                post_processed_data = binary_fill_holes(
                    binary_closing(post_processed_data))
                post_processed_data = binary_dilation(post_processed_data)
        else:
            if "4" in nifti_file:
                # Remove small objects
                post_processed_data = remove_small_objects(
                    segmented_data_post.astype(bool), min_size=500000)

                # Apply morphological operations for post-processing
                post_processed_data = binary_fill_holes(
                    binary_closing(post_processed_data))

                post_processed_data = binary_fill_holes(post_processed_data)
            elif "3" in nifti_file:
                # Remove small objects
                post_processed_data = remove_small_objects(
                    segmented_data_post.astype(bool), min_size=400000)

                # Apply morphological operations for post-processing for case 3
                post_processed_data = binary_opening(post_processed_data)
                post_processed_data = binary_fill_holes(post_processed_data)

                # Remove small objects
                post_processed_data = remove_small_objects(
                    post_processed_data.astype(bool), min_size=180000)

                post_processed_data = binary_fill_holes(post_processed_data)
                post_processed_data = binary_closing(post_processed_data)
                post_processed_data = binary_fill_holes(post_processed_data)

                # Remove small objects
                post_processed_data = remove_small_objects(
                    post_processed_data.astype(bool), min_size=200000)
            else:
                # Remove small objects
                post_processed_data = remove_small_objects(
                    segmented_data_post.astype(bool), min_size=300000)

                # Apply morphological operations for post-processing
                post_processed_data = binary_fill_holes(
                    binary_closing(post_processed_data))

                post_processed_data = binary_dilation(
                    binary_fill_holes(post_processed_data))

        # Save the final segmentation NIFTI file
        final_output_filename = f"{nifti_file}_SkeletonSegmentation.nii.gz"
        final_segmented_nifti = nib.Nifti1Image(
            post_processed_data.astype(np.uint8), nib.load(nifti_file).affine)
        nib.save(final_segmented_nifti, final_output_filename)

        res, num_components = label(post_processed_data, return_num=True)
        print("Number of connectivity components: {}".format(num_components))

        # Save the final segmentation NIFTI file
        final_output_filename = f"{nifti_file}_SkeletonSegmentation.nii.gz"
        final_segmented_nifti = nib.Nifti1Image(
            post_processed_data.astype(np.uint8), nib.load(nifti_file).affine)
        nib.save(final_segmented_nifti, final_output_filename)
        return i_min_final

    except Exception as e:
```

```
        print(f"Error: {e}")
        return None
```

Method: nifti_to_img_converter

```
def nifti_to_img_converter(nifti_file_path, save_path='output_img_data.npy'):
    """
   Function Description:
   Converts a NIFTI file to image data and generates visualizations, including
axial, sagittal, and coronal slices,
   a montage of axial slices, and saves the entire image data. The visualizations
are saved as PNG files with relevant
   titles.

   Visualization Outputs:
   - Axial Slice at the Middle
   - Sagittal Slice (Rotated 270 degrees)
   - Coronal Slice (Rotated 270 degrees)
   - Montage of Axial Slices at the Middle

   Additionally, the entire image data is saved as an output file.

   :param nifti_file_path: Path to the input NIFTI file.
   :param save_path: Path to save the output image data file (default:
         'output_img_data.npy').
   :return: Image data as an array or None if an error occurs.
   """
    try:
        # Load the NIFTI file
        nifti_data = nib.load(nifti_file_path)

        # Get the data
        img_data = nifti_data.get_fdata()

        # Show and save the axial slice using matplotlib
        axial_slice = np.rot90(img_data[:, :, img_data.shape[2] // 2], k=3)
        plt.imshow(axial_slice, cmap='gray', aspect='auto')
        plt.title('Axial Slice at the Middle')
        plt.colorbar()
        plt.savefig(save_path.replace('.npy', '_axial_slice.png'))
        plt.show()

        # Show and save the sagittal slice using matplotlib (rotated by 270
degrees)
        sagittal_slice = np.rot90(img_data[img_data.shape[0] // 2, :, :], k=3)
        plt.imshow(sagittal_slice, cmap='gray', aspect='auto')
        plt.title('Sagittal Slice (Rotated 270 degrees)')
        plt.colorbar()
        plt.savefig(save_path.replace('.npy', '_sagittal_slice.png'))
        plt.show()

        # Show and save the coronal slice using matplotlib (rotated by 270
degrees)
        coronal_slice = np.rot90(img_data[:, img_data.shape[1] // 2, :], k=3)
        plt.imshow(coronal_slice, cmap='gray', aspect='auto')
        plt.title('Coronal Slice (Rotated 270 degrees)')
        plt.colorbar()
        plt.savefig(save_path.replace('.npy', '_coronal_slice.png'))
        plt.show()

        # Create a montage of slices along the axial axis (change axis as needed)
        montage = np.transpose(img_data, (2, 0, 1))

        # Display and save the montage using matplotlib
        plt.imshow(montage[:, :, montage.shape[2] // 2], cmap='gray',
aspect='auto')
        plt.title('Montage of Axial Slices')
        plt.colorbar()
        plt.savefig(save_path.replace('.npy', '_axial_montage.png'))
```

```
        plt.show()

        # Save the entire image using numpy's save function
        np.save(save_path, img_data)

        return img_data  # Return image data as an array

    except Exception as e:
        print(f"Error: {e}")
        return None  # Return None if an error occurs
```