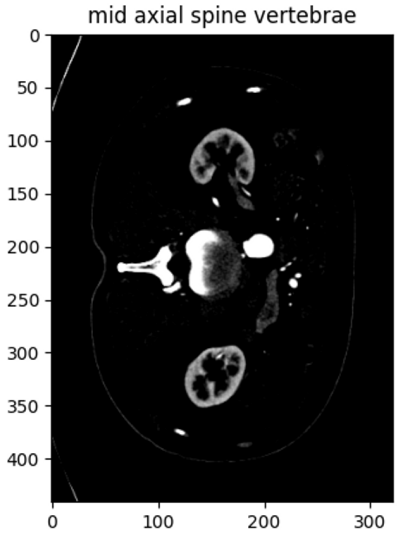
HC Assignment2

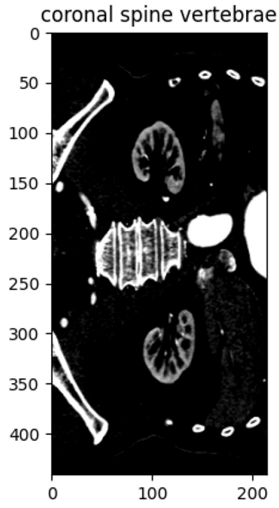
The code is available at https://github.com/sunningmbzuai/HC701/tree/main/ass2

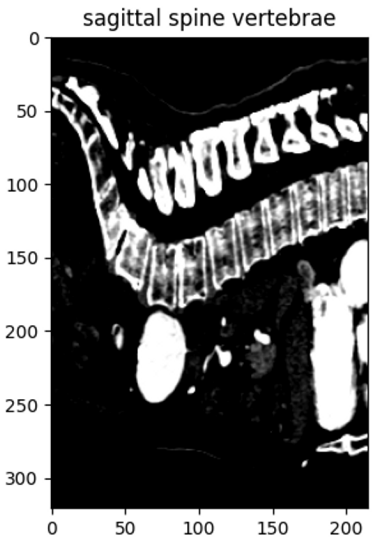
All codes are in assignment2.ipynb, except task2.2 is implemented by task2.py

Task 1.1

Best intensity: min_intensity=60, max_intensity=250







Task 1.2

task1.2.1 Watershed algorithm

The watershed algorithm (Vincent & Soille, P, 1991) (Wang & Liu, 2017) (Chen, Chen, Huang, Zhang, & Wang, 2017) (Ghose, Mitra, & Chakraborty, 2019) (Li, et al., 2020)is a popular image segmentation technique used to identify and isolate individual objects within an image. In the context of spine vertebrae segmentation, the algorithm is particularly useful for separating overlapping or adjacent vertebrae from one another.

The algorithm works by treating the image as a topographic map, where each pixel is assigned an elevation value based on its intensity or other image features. The algorithm then identifies "watershed lines" along the edges of objects in the image by simulating the flow of water over the topographic surface. These watershed lines represent the boundaries between adjacent objects in the image.

To segment spine vertebrae using the watershed algorithm, a pre-processing step is typically required to enhance the contrast and eliminate noise in the image. This may involve techniques such as histogram equalization, filtering, or thresholding.

Next, the algorithm is applied to the pre-processed image, with the watershed lines identified to separate the individual vertebrae. The resulting segmented image can then be post-processed to eliminate any remaining noise or artifacts, and to refine the boundaries of the individual vertebrae.

One potential challenge of using the watershed algorithm for spine vertebrae segmentation is that it can sometimes result in over-segmentation or under-segmentation, where adjacent vertebrae are erroneously separated or merged together. This can be addressed through the use of additional techniques, such as region merging or morphological operations, to refine the segmentation.

Overall, the watershed algorithm is a powerful and flexible tool for spine vertebrae segmentation, particularly when dealing with complex or overlapping structures in the image. However, its success depends on careful pre-processing, parameter tuning, and post-processing to ensure accurate and reliable results

Bibliography

Vincent, L., & Soille, P. (1991). Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Wang, J., & Liu, H. (2017). A review of image segmentation using watershed algorithm. *Journal of Imaging.*

Chen, Z., Chen, Y., Huang, S., Zhang, X., & Wang, J. (2017). A review of image segmentation methods based on the watershed algorithm. *Journal of Physics: Conference Series.*

Ghose, S., Mitra, A., & Chakraborty, C. (2019). A review on image segmentation techniques with remote sensing applications. *International Journal of Remote Sensing.*

Li, Y., Qin, J., Li, H., Li, J., Zhang, H., Li, X., & Liu, H. (2020). A review of recent progress in spine vertebrae segmentation. *Journal of Healthcare Engineering.*

task 1.2.2 Find locating boxes for each spine vertebra

Description

1. Segment the medical image using the watershed algorithm
2. Limit the intensity of the resulting segmented image to a specific range.
3. Calculate the minimum value for each column, and identify local minimum points to partition the image into zones.
4. For each zone, locate a box to identify the vertebra within the zone.

Code

```
### watershed algorithm
import SimpleITK as sitk
import numpy as np

# use watershed algorithm to get segmentation
min_intensity = 100
max_intensity = 300

clipped_img = img_data.clip(min_intensity,max_intensity)
img4 = clipped_img[img_data.shape[0]//2, : , :]
image = sitk.GetImageFromArray(img4)

# Preprocessing to enhance the image
# Here, we apply a Gaussian smoothing filter and a histogram equalization filter
image_smooth = sitk.SmoothingRecursiveGaussian(image, 1.0)
image_eq = sitk.AdaptiveHistogramEqualization(image_smooth)

# Thresholding to separate the vertebrae from the background
threshold_filter = sitk.BinaryThresholdImageFilter()
```

```
threshold_filter.SetLowerThreshold(200) # Adjust the threshold value as needed
threshold_filter.SetUpperThreshold(2000) # Adjust the threshold value as needed
threshold_filter.SetInsideValue(1)
threshold_filter.SetOutsideValue(0)
image_threshold = threshold_filter.Execute(image_eq)

# Morphological operations to remove small objects and fill holes
closing_filter = sitk.BinaryMorphologicalClosingImageFilter()
closing_filter.SetKernelRadius(2)
image_closing = closing_filter.Execute(image_threshold)

opening_filter = sitk.BinaryMorphologicalOpeningImageFilter()
opening_filter.SetKernelRadius(2)
image_opening = opening_filter.Execute(image_closing)

# Connected component analysis to separate individual vertebrae
cc_filter = sitk.ConnectedComponentImageFilter()
image_cc = cc_filter.Execute(image_opening)

# Extract the largest connected component (which should be the vertebrae)
label_shape_stats = sitk.LabelShapeStatisticsImageFilter()
label_shape_stats.Execute(image_cc)
labels = label_shape_stats.GetLabels()
largest_label = None
largest_size = 0
for label in labels:
    size = label_shape_stats.GetNumberOfPixels(label)
    if size > largest_size:
        largest_label = label
        largest_size = size
image_largest_cc = sitk.RelabelComponent(image_cc, largest_label)

# Get the segmentation as a numpy array for further processing or visualization
segmentation = sitk.GetArrayFromImage(image_largest_cc)
```

```
### limit intensity of segmentation to a specific range
segmentation[segmentation>16] = 0
segmentation[segmentation<4] = 0
segmentation[:80,:] = 0
img4[segmentation>16] = 300
img4[segmentation<4] = 300
img4[:80,:] = 300
plt.imshow(img4, cmap='gray') # Figure1
```

```
### find local min to partition zones
x, y = [],[]
for j in range(img4.shape[1]):
    x.append(j)
    y.append(np.min(img4[:,j]))
plt.plot(x,y) # Figure2

win = 20
split_points=[]
for j in range(int(img4.shape[1]/win )+1):
    min = np.min(y[j*win:(j+1)*win])
    if min < 120:
        idx = y[j*win:(j+1)*win].index(min)
        split_points.append(j*win+idx)
split_img = img4.copy()
split_img[:,split_points] = 100
plt.imshow(split_img, cmap='gray') # Figure2
```

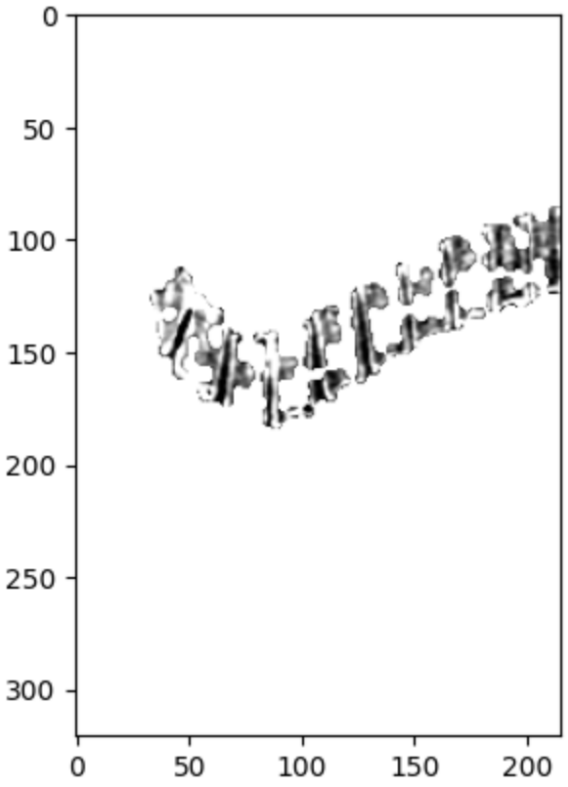


Figure1: Segmentation within specific range

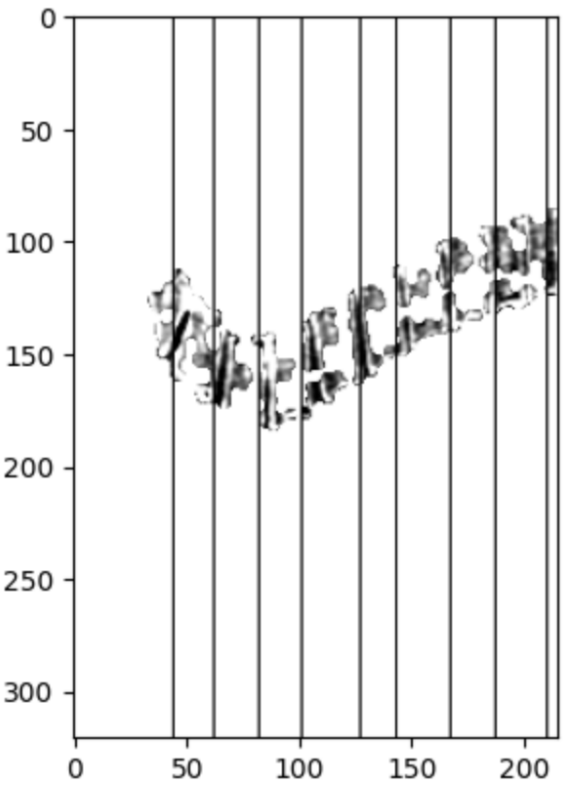


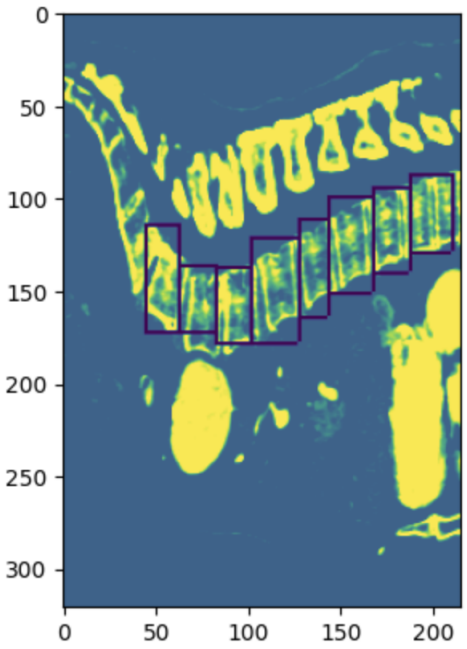
Figure2: partitioning image into zones

```
### find box in each zone
boxes = []
old_max_idx = 0
old_min_idx = 0
old_j = 0
for j in split_points:
    if old_max_idx == 0:
        old_max_idx = np.max(np.where(img4[:,j]<300))
        old_min_idx = np.min(np.where(img4[:,j]<300))
        old_j = j
    else:
        max_idx = np.max(np.where(img4[:,j]<300))
        min_idx = np.min(np.where(img4[:,j]<300))

        l = [old_j,np.min([old_min_idx,min_idx])]
        r = [j,np.max([old_max_idx,max_idx])]
        boxes.append(l+r)
        old_max_idx = max_idx
        old_min_idx = min_idx
        old_j = j
```

```
### show boxes in original image
import SimpleITK as sitk
import numpy as np

# Convert the image to a numpy array for drawing
image_array = img3

# Draw boxes on the image array
for box in boxes:
    x1, y1, x2, y2 = box
```

```
    image_array[y1:y2, x1:x1+2] = 0
    image_array[y1:y2, x2:x2+2] = 0
    image_array[y1:y1+2, x1:x2] = 0
    image_array[y2:y2+2, x1:x2] = 0

# Convert the image array back to a SimpleITK image
plt.imshow(image_array)
```

Result



The number of vertebra

We detect 8 pieces of vertebra in this image

task 1.2.3 Remove all non-vertebra regions from the 2D image

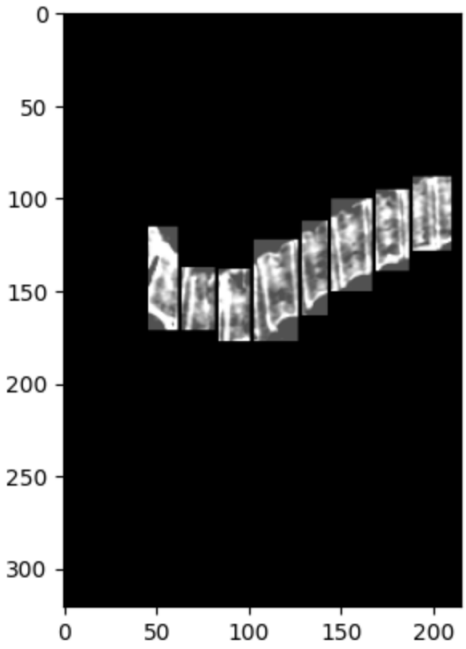
Description:

Once the boxes of each vertebra have been obtained, we proceed to mask out the pixels outside the boxes. This ensures that only the pixels within the boxes are displayed

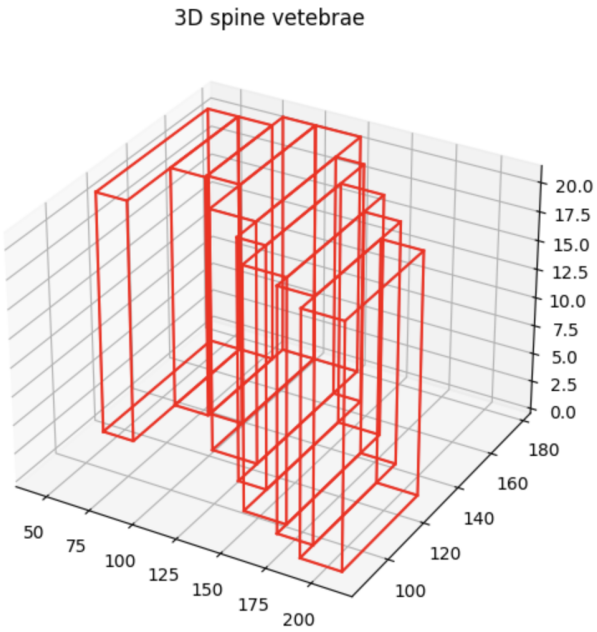
Code

```
### remove non-vertebra regions from 2D image
import SimpleITK as sitk
import numpy as np

# Convert the image to a numpy array for drawing
image_array = img3
# mask image
mask_img = np.zeros(img3.shape)

# remain pixels within boxes
for box in boxes:
    x1, y1, x2, y2 = box
    mask_img[y1:y2,x1:x2] = img3[y1:y2,x1:x2]
plt.imshow(mask_img,cmap='gray')
```

Result



Task 1.2.3 3D boxes for spine vertebrae



Task 2.1

class	dataset type	sample number	range of filename
Healthy	train	3040	h0995.png ~ h5000.png
Healthy	test	760	h0001.png ~ h0993.png

class	dataset type	sample number	range of filename
TB	train	640	tb0250.png ~ tb1199.png
TB	test	160	tb0003.png ~ tb0248.png

Task 2.2

Experiment Setting

Model architecture: ResNet50

ResNet50 is a convolutional neural network architecture that has demonstrated state-of-the-art performance in image classification tasks. The main benefit of ResNet50 is its ability to train very deep neural networks with hundreds of layers without encountering the vanishing gradient problem, which is a common issue that arises when training very deep neural networks.

ResNet50 achieves this by introducing residual connections, which allow the network to bypass certain layers and make direct connections between input and output layers. This architecture enables the network to learn more complex features and make better predictions, leading to improved accuracy in image classification tasks.

Data Augmentation: RandomHorizontalFlip and RandomVerticalFlip (Flip in the table), Color Jitter.

> **ColorJitter(brightness=0,contrast=0,saturation=0,hue=0)** is a type of transformation that changes the color of an image by applying random perturbations to its hue, saturation, brightness, and contrast. This technique is used to create additional training data that can help improve the accuracy and robustness of machine learning models.
>
> **RandomHorizontalFlip(p=0.5)/RandomVerticalFlip(p=0.5)** involves randomly flipping images horizontally/vertically along the x-axis/y-axis. This technique can be applied to image datasets during training to increase the size of the dataset and improve the model's ability to generalize to new, unseen data. By randomly flipping images horizontally/vertically, the model learns to recognize the same object or pattern.

Exp	Data Augmentation	Network	Hyperparameter	Parameter Num	FLOPS
1	Normalization Flip	ResNet50	depth_list=[0, 1, 2] expand_ratio_list=[0.2, 0.25, 0.35] width_mult_list=[0.65, 0.8, 1.0]	46061090	235.205M
2	Normalization Flip	ResNet50	dropout_rate=0.5 depth_list=[0, 1, 2] expand_ratio_list=[0.2, 0.25, 0.35] width_mult_list=[0.65, 0.8, 1.0]	46061090	235.205M
3	Normalization Color Jitter Flip	ResNet50	dropout_rate=0.5 depth_list=[0, 1, 2] expand_ratio_list=[0.2, 0.25, 0.35]	46061090	235.205M

			width_mult_list= [0.65, 0.8, 1.0]		
4	Normalization Color Jitter Flip	ResNet50	dropout_rate=0.5 depth_list=[0, 1] expand_ratio_list= [0.2, 0.25] width_mult_list= [0.65, 0.8]	14228282	142.557M
5	Normalization Color Jitter Flip	ResNet50	dropout_rate=0.5 depth_list=[0, 1] expand_ratio_list= [0.2, 0.3] width_mult_list= [0.8, 0.8]	18469978	148.892M

Exp	Observation	Decision	Decision Explanation
1	The testing loss is also abnormally increasing.	add drop out	model is overfitting, add drop out to reduce overfitting
2	The testing accuracy dramatically changes	add color jitter data augmentation	model is overfitting, add color jitter to reduce overfitting
3	The testing accuracy still dramatically changes	reduce the number of layer	model is overfitting, reduce model layers to reduce overfitting
4	The best accuracy deceases.	increase the width of layer	model is underfitting, increase model scale
5	achieve best testing accuracy	Finish	Finish

Experiment Results

Experiment	Test Accuracy	F1 score	Confusion Matrices
1	91.63%	0.9033	[[752 8] [79 81]]
2	90.22%	0.8878	[[756 4] [86 74]]
3	92.5%	0.9167	[[759 1] [68 92]]
4	73.26%	0.7271	[[645 115] [131 29]]
5	**96.96**%	**0.9691**	**[[752 8] [20 140]]**

Top2 Training Accuracy and Loss Figures

Top1: Experiment5

Top2: Experiment3


train_loss
icy-sky-1
0.8
0.6
0.4
0.2
0
epoch
0
5
10
15
train_accuracy
icy-sky-1
99
98
97
96
95
94
93
epoch
0
5
10
15
test_loss
icy-sky-1
10
8
6
4
2
0
Step
10
20
30
test_accuracy
icy-sky-1
80
60
40
20
0
Step
10
20
30