CS 590 Challenge2: Cortical thickness challenge

Author: Ye Cheng

Project Description

This project is completed in Python. Nibabel, numpy, skimage, and CV2 packages are used in the project. In this project, the cerebral cortex is segmented by identifying the gray level of the image and contour. After obtaining the segment of the cerebral cortex: Method 1 determines the thickness by determining whether there are zero pixels around non-zero pixels. Method 2 uses cv2.distanceTransform to show the thickness.

The looking_down script and the looking_front script Is a segment and thickness map of a single slice of 2 views (default way). The looking_down_144 script and the looking_front_176 are used to correct the default contour function to get a correct thickness map. The all map_front script and the all map_down script will traverse all slices through two views.

Specific Steps

Segment:

First, in this project, we use nibabel.load to load the CT data and pick the slice to segment.

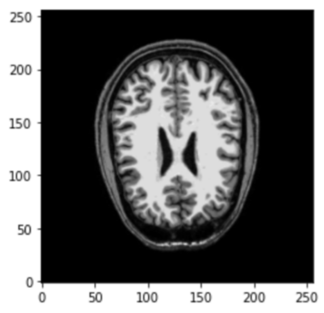
```
#choice one image number num
def choice_image(image,num):
    numpy = image.get_fdata()
    numpy = numpy.T
    numpy = numpy[:,num]
    return numpy
```

```
image = nib.load('./raw_t1_subject_02.nii.gz')
image_brain_numpy = choice_image(image,100)
```

Then, we can adjust the image through the CT intensity value (Hounsfield unit) and through the window level and window width. This may be helpful for the later segment (X-ray absorption is measured by Hounsfield unit (HU) (Hounsfield, CT inventor), which is mainly used to measure radiation intensity).

```
# Functions that display CT slices with level, window
def Pretreatment(npslice, level, window):
    max = level + window/2
    min = level - window/2
    npslice = npslice.clip(min,max)
    plt.imshow(np.flip(npslice,axis=0), cmap="gray", origin="lower")
    return npslice
```

```
image_brain_numpy = Pretreatment(image_brain_numpy,40,80)
```



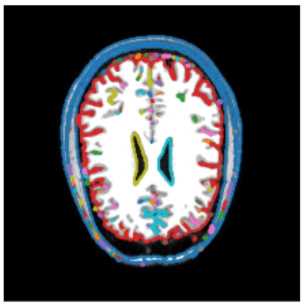
The window level and window width selected for different parts are different. Window level40 and window width80 are to observe the cerebral.

Then, binarization is performed according to the intensity threshold of the CT image. Through the skimage. measure.find_contours () method, which can find the relevant contour in the image and return a complete contour list containing the pixel information coordinates of each contour.

```python
#Binarization of an image according to its intensity
def intensity_seg(ct_numpy, min, max):
    clipped = ct_numpy.clip(min, max)

    clipped[clipped != max] = 1
    clipped[clipped == max] = 0
    return measure.find_contours(clipped, 0.99)
```

Define show_ contour function and read the contour one by one from the contour list returned by the intensity_seg function

```python
def show_contour(image, contours, name=None ):
    fig, ax = plt.subplots()
    #ax.imshow(image.T, cmap=plt.cm.gray)
    ax.imshow(image, cmap=plt.cm.gray)
    # Each contour in the returned contour list from  measure.find_contours
    # diff colour
    for contour in contours:
        ax.plot(contour[:, 1], contour[:, 0], linewidth=2)
    ax.set_xticks([])
    ax.set yticks([])
```



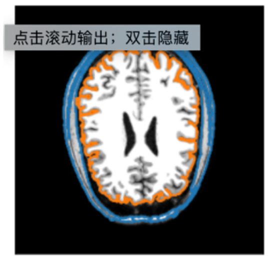
In the above contour map, we found too many shape contours in the image. However, we only want the outline of the cerebral, so we need to exclude other outlines. First, set the contour closure determination function, which requires that the cerebral contour must be closed. Here, we also need to calculate the starting point distance in the coordinate list of each wheel list to assist in the determination. Delete the contour if the start and end points are not closed.

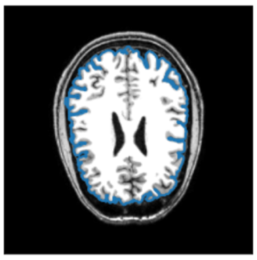
```python
#Euclidean distance between points of Contour array
def contour_distance(contour):

    dx = contour[0, 1] - contour[-1, 1]
    dy = contour[0, 0] - contour[-1, 0]
    #Euclidean distance between the first coordinate point and the last coordinate point
    return np.sqrt(np.power(dx, 2) + np.power(dy, 2))

# Determine true or false according to contour closure
def set_is_closed(contour):
    if contour_distance(contour) < 1:
        return True
    else:
        return False
```

In this way, we can remove many interference contours, but we still have a skull contour that hasn't been removed, so we will calculate the outline size, sort it, and delete the skull outline. We can check all the contours and decide how to delete the interference contour (find_allcerebral (contours)). For example, the skull contours in the looking down view is larger and complete, so the larger outline has to deleted. The skull contours in the front view is incomplete, so the smaller contour has to deleted. In looking_ front_ 176 and looking_ down_ 144 provides two examples as solutions that are not suitable for the above two view default situation (see challenges and problems part).

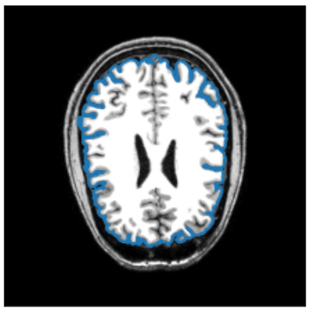
```python
def find_brain(contours):
    body_and_brain_contours = []
    vol_contours = []

    for contour in contours:
        hull = ConvexHull(contour)

        # Set pixel limit
        if hull.volume > 2000 and set_is_closed(contour):
            body_and_brain_contours.append(contour)
            vol_contours.append(hull.volume)

    # Discard body contour
    if len(body_and_brain_contours) == 1:

        return body_and_brain_contours

    elif len(body_and_brain_contours) > 1:
        vol_contours, body_and_brain_contours = \
        (list(t) for t in zip(*sorted(zip(vol_contours,
                                          body_and_brain_contours))))
        # Excluding the body contour, leaving only the brain contour
        body_and_brain_contours.pop(-1)
        if len(body_and_brain_contours) != 1:
             body_and_brain_contours.pop(0)
        return body_and_brain_contours


def find_allbrain(contours):
    body_and_brain_contours = []
    vol_contours = []

    for contour in contours:
        hull = ConvexHull(contour)

        # Set pixel limit
        if hull.volume > 2000 and set_is_closed(contour):
            body_and_brain_contours.append(contour)
            vol_contours.append(hull.volume)

    # Discard body contour
    return body_and_brain_contours
```

We can see all the contours and we have correctly removed the interference contour. Sometimes we need to modify the function to delete the interference contour correctly, but this time the default removal function is right.

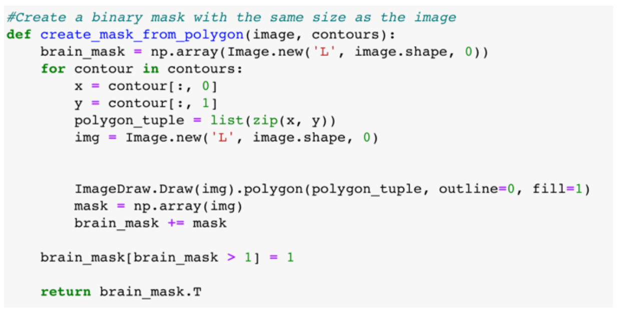




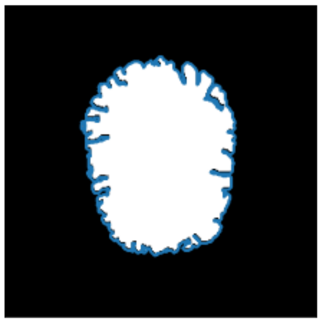
Finally, we can get the contour of the cerebral.



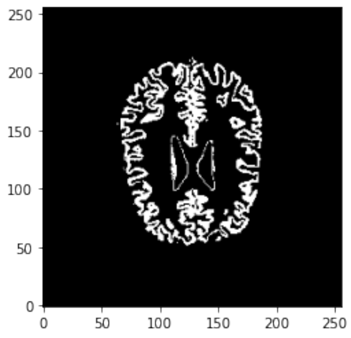
After that, we need to convert the cerebral contour set into a binary mask of the cerebral, so we have the target area of the cerebral.



```python
#Create a binary mask with the same size as the image
def create_mask_from_polygon(image, contours):
    brain_mask = np.array(Image.new('L', image.shape, 0))
    for contour in contours:
        x = contour[:, 0]
        y = contour[:, 1]
        polygon_tuple = list(zip(x, y))
        img = Image.new('L', image.shape, 0)


        ImageDraw.Draw(img).polygon(polygon_tuple, outline=0, fill=1)
        mask = np.array(img)
        brain_mask += mask

    brain_mask[brain_mask > 1] = 1

    return brain_mask.T
```



Then, we obtained the cerebral cortex by screening the intensity of Hounsfield unit. First, we multiply the CT image and the cerebral mask pixel by pixel to isolate the cerebral region. Then, we set the element whose multiplication result is 0 to -1000 (that is, the air intensity in HU). Finally, only the intensity of 40-80 is reserved as the cerebral cortex.

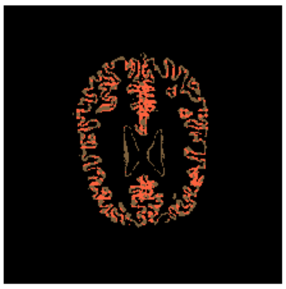


Thickness:

There are two methods to display thickness in this project.

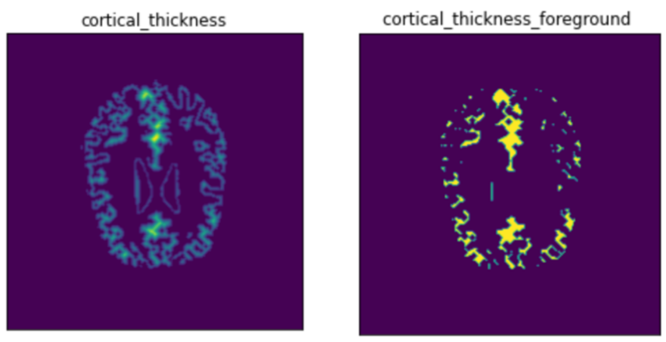
Method one: we judge whether a non-zero pixel should change its value by judging whether there are zero pixels around it. If there are zero pixels around, it means that the thickness here is not enough, and here will be dyed light. On the contrary, it will be dyed dark. Then we merge the two cortical images (the dark part is set to be translucent (alpha=0.5)).

```python
#Method 1
cort = np.array(cortical)

def distance_coclor(cortical1):
    cortical_x, cortical_y = np.nonzero(cortical1+1000)  # get whirt part
    for (x,y) in zip(cortical_x, cortical_y):
        if(x+5 <len(cortical1) and y+5 <len(cortical1) and x-5 >0 and y-5 >0):
            if((cortical1[x+5,y] != -1000 or cortical1[x-5,y] != -1000)
              and (cortical1[x,y+5] != -1000 or cortical1[x,y-5] != -1000)):
                cortical1[x,y] = 1000
            else:
                cortical1[x,y] = -2000
    return cortical1


dis_cortical = distance_coclor(cort)
x_x, y_y = np.nonzero(dis_cortical)
for (x,y) in zip(x_x, y_y):
    if dis_cortical[x,y] == 1000:
        dis_cortical[x,y] = -1000
    else:
        dis_cortical[x,y] = 1000

plt.figure()
plt.imshow(dis_cortical , 'flag', interpolation='none' )
plt.imshow(cortical, 'copper', interpolation='none', alpha=0.5 )
plt.axis('off')
plt.xticks([])
plt.yticks([])
plt.savefig('./cortical1_down_thickness_map', bbox_inches = 'tight',pad_inches = 0)
```

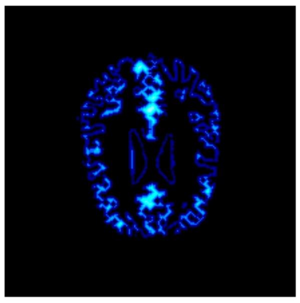


Method two: we can just use cv2.distanceTransform function to show the thickness. The calculation result of this function reflects the distance relationship between each pixel and the background (pixels with a value of 0). We can also use cv2.threshold to get the foreground of the thickness map. Then, we convert the distance map into a color map. Then we merge the two cortical images.

```python
#Method 2
img = cv2.imread('./cortical.png', 0)

img = cv2.fastNlMeansDenoising(img,None,10,7,21)#Denoising
#Calculate distance display thickness map
cortical_thickness = cv2.distanceTransform(img, cv2.DIST_L1, cv2.DIST_MASK_3)
#foreground of thickness map
ret, img_fore = cv2.threshold(cortical_thickness, 0.5 * cortical_thickness .max(), 255, cv2.THRESH_BINARY)
print('cortical_thickness.max()', cortical_thickness.max())

plt.imshow(cortical_thickness)
plt.title("cortical_thickness"), plt.xticks([]), plt.yticks([])
plt.show()


plt.imshow(img_fore)
plt.title("cortical_thickness_foreground "), plt.xticks([]), plt.yticks([])
plt.show()
```



```
cortical_thickness1 = cv2.convertScaleAbs(cortical_thickness )
cortical_thickness2 = cv2.normalize(cortical_thickness1, None, 255,0, cv2.NORM_MINMAX, cv2.CV_8UC1)
#heatmap
heat_cortical_thickness = cv2.applyColorMap(cortical_thickness2, cv2.COLORMAP_HOT)
heat_cortical_thickness = cv2.convertScaleAbs(heat_cortical_thickness, alpha=2, beta=0)
plt.imshow(heat_cortical_thickness)

plt.title("cortical_thickness"), plt.xticks([]), plt.yticks([])
plt.show()
```

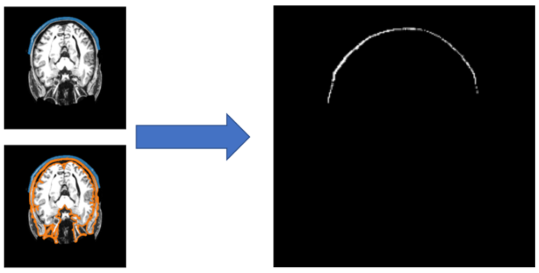


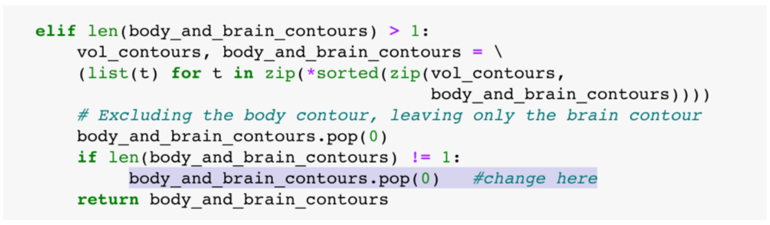
Challenges and Problems

The biggest problem of this project is the screening of contour. This project has 256 slices (raw_t1_subject_02.nii.gz) because each slice is different, the size of the contour is also different. In order to exclude the skull and other non-cortical contours, we can find out the rule of contour size and exclude the non-cortical contours, but this rule does not apply to all slices. Therefore, we need to modify the filter contour function manually according to different slices (find_cerebral () function).

This causes some slices to be blank or show skull outline if we traverse all slices. At this time, we need to modify the filtering cortex function for a specific slice. It's not very convenient, but you can segment the cortex you want.

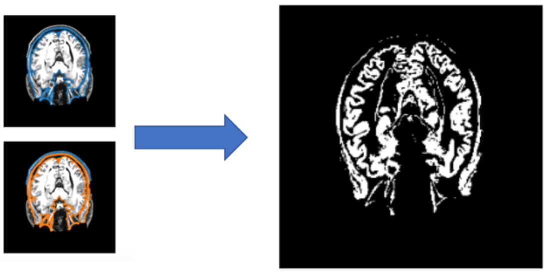
For example, the default delete interference contour function is in looking_ down_ 144 not applicable. We need to modify the interference contour function

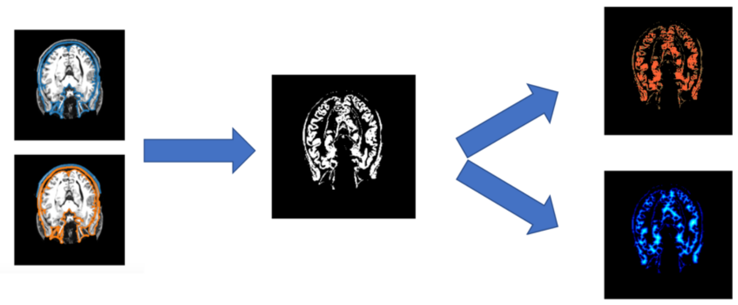




```
elif len(body_and_brain_contours) > 1:
    vol_contours, body_and_brain_contours = \
    (list(t) for t in zip(*sorted(zip(vol_contours,
                                      body_and_brain_contours))))
    # Excluding the body contour, leaving only the brain contour
    body_and_brain_contours.pop(0)
    if len(body_and_brain_contours) != 1:
        body_and_brain_contours.pop(0)    #change here
    return body_and_brain_contours
```

After modifying the function, we extracted the contour correctly, and we have the thickness map too.





In looking_ front_ 176 there are similar problems. For details, please refer to looking_ front_ 176, looking_ down_ 144 code, and corresponding BW file and MAP file pictures.

There is another solution. After we segment other unclosed contours, we will not continue to filter. This will keep the contour of the skull. The advantage is that the outline of the cerebral cortex will not be deleted by bad filter rule, but the picture has interference items, which are inconvenient to observe. This may apply to traversing all slices. If you only need one slice of the cerebral cortex, it is recommended to manually modify the contour filter function (find_cerebral () function).
Second challenge of this project is to find the cerebral cortex through different gray levels. We need to determine the gray scale of the cerebral cortex. If the gray value range is wrong, an accurate cortical thickness image cannot be obtained.

Another problem is the staining of cortical thickness. Method 1 is not very accurate. Method 2 is not very obvious.

The first method only judges whether it is close to the edge but does not calculate the distance. This is because if you want to calculate the distance, you will traverse the distance between all pixels and all non-zero pixels. If the picture is large, it will take a long time. There might be a better way.

Method 2 is very convenient, but the area with medium thickness is not obvious. This can be made more obvious by setting the foreground.

Last question about the ratio of the CT image to the real cerebral. How many cortical pixels are stacked together is appropriately called thick cortical?

Summary

In summary, in this project, the cerebral cortex can be divided and stained according to the thickness For better effect, the cerebral cortex can be segmented once according to different slices (using looking_down script or looking_front script). If you want to traverse all slices, you can relax the contour constraints. This will interfere with the contour, but the cerebral cortex will not be deleted by inappropriate screening rules.

Reference

OpenCV: OpenCV modules. (n.d.). OpenCV. Retrieved July 24, 2022, from https://docs.opencv.org/4.x/index.html

Baba, Y. (2021, September 30). *Windowing (CT)*. Radiology Reference Article | Radiopaedia.Org. https://radiopaedia.org/articles/windowing-ct

Guy-Evans, O. (2021, May 19). *Cerebral Cortex Location & Functions | Simply Psychology*. Olivia Guy-Evans. Retrieved July 24, 2022, from https://www.simplypsychology.org/what-is-the-cerebral-cortex.html