Bishop's University

M.Sc. Computer Science

CS590 Masters Project

Cortical Thickness Estimation

Submitted to: Dr. Russell Butler

Submitted By:

Jagjot Singh (002311630)

Jagpreet Singh (002309033)

Objective of the Challenge

The challenge is to develop an algorithm to estimate the cortical thickness map from a RAW T1 weighted image. Cortical thickness map is the thickness of the grey matter of the brain at every point (distance between the white matter and the pill surface).

Data provided

raw_t1_subject_01.nii.gz

raw_t1_subject_02.nii.gz

raw_t1_subject_02.nii.gz'

Steps we performed To Build this project

Step 1: Importing the necessary libraries

Following libraries were used to implement this challenge

```
import time
import numpy as np
import matplotlib.pyplot as plt
import nibabel as nib
from math import sqrt
from skimage import filters, morphology, segmentation
```

Step 2: Reading given data files

Load the **raw t1** file as a numpy (256,256,256) array using 'nibabel' python library.

```
INPUT_FILE = "./raw_t1_subject_02.nii.gz"

image = nib.load(INPUT_FILE).get_fdata()
```

Step 3: Created an empty numpy (256,256,256) to store the final result

```
final_image = np.full((256,256,256),0, dtype = float)
```

Step 3: Divide the loaded RAW image in 256 2-D slices

Performing the segmentation of each of 256 2d slices and then aggregating the segmented planes to form a 3D thickness

```
for z in range(256):
    sst = time.time()
    im00 = image[:,z,:]
```

We are estimate the cortical thickness of each 256 slice.

Store each estimated 2-D thickness map for z = 0 to 255 in the empty numpy (256,256,256) array created in the 2nd step.

After storing the all the 256 estimation 2-D slices, save the result as a nii.gz file after the completion of algorithm used

Step 4: Generating Binary Mask to extract brain from the image

```
    # Generating mask to remove skull
    im01 = morphology.convex_hull_image(np.logical_and(im00 > 100, im00 < 111))
    # plt.imshow(im00)
    # plt.show()
    im02 = segmentation.expand_labels(im01, distance= 6)
```

We are using ‘convex_hull_image()’ method from the ‘morphology’ package of the ‘skimage’. Convex_hull_image() gives us the initial shape of the brain based on the white matter intensity. Usually, this shape is smaller than the image size, so after taking the initial mask shape, I’m using ‘expand_labels()’ method from the ‘segmentation’ package of the ‘skimage’ with the distance value of 6, which give me the required binary mask required to extract the brain from the skull.

Step 5: Segmenting White matter

After extracting the brain from the skull, we are segmenting the white matter from the image using some value of intensity threshold.

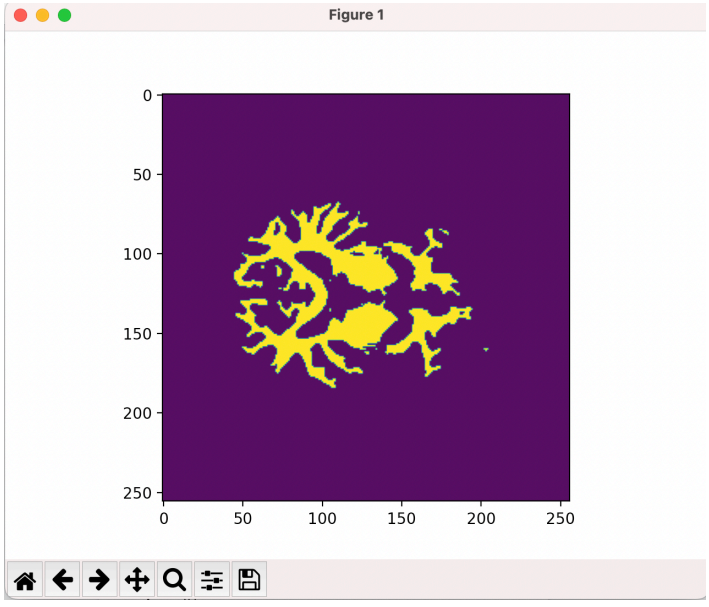
```
im04 = im03 > 87 #White Matter
```

After this we moved on to segment the non-required matter, In the code we are calling it dark matter ()

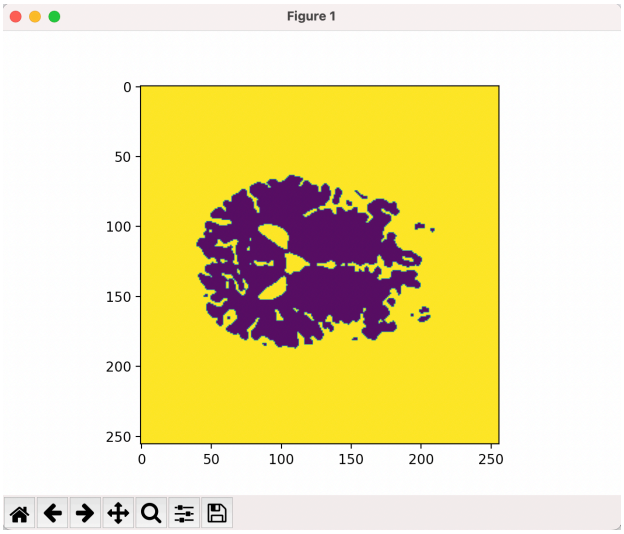
```
im05 = im03 < 51 #Dark Matter
```

Step 6: Using **median filter** for dark matter segmentation and then used **binary_dilation** on both the segmented outputs

```
im04 = im03 > 87 #White Matter
im05 = im03 < 51 #Dark Matter
im06 = filters.median(im04)
im07 = morphology.binary_dilation(im06)
im08 = filters.median(im05)
im09 = morphology.binary_dilation(im08)
im10 = im07 ^ im06 #Whitematter boundary
im11 = im09 ^ im08 #Darkmatter boundary
```



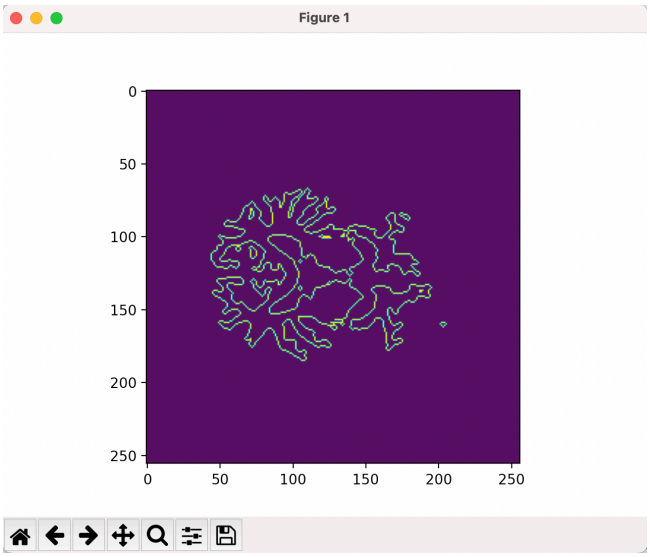
White Matter



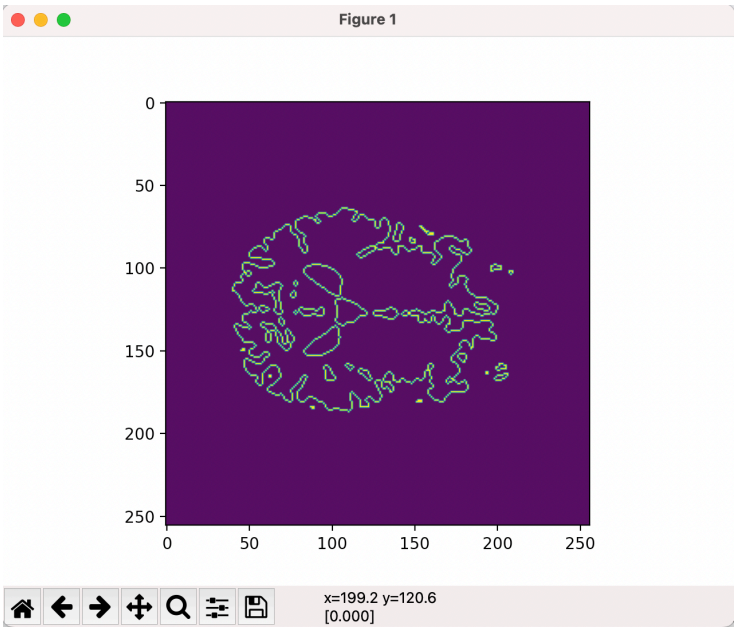
Dark Matter

Step 7: Masking out the segmented image form the dilated image
Using the White matter and Dark Matter boundary performing the below task

```
im10 = im07 ^ im06 #Whitematter boundary
im11 = im09 ^ im08 #Darkmatter  boundary
```



White Matter Boundary



Dark Matter Boundary

Step 8: Building the Algorithm

8.1 After completing the above explained steps we has boundaries of both White matter and Dark Matter, Now we are using the distance formula to calculate the distance between the pixels.

$$d=\sqrt{((x\_2-x\_1)^2+(y\_2-y\_1)^2)}$$

We are iterating the loop for each pixel on White Matter boundary and calculating its distance from every pixel on Dark Matter boundary. After this we are storing the shortest distance value inside the White Matter Pixel

```
## Filling the White Matter boundary points with the closest distance of Dark matter(Grey Matter) point
im15 = np.full((256,256),0,dtype=float)
for i1 in range(256):
    for j1 in range(256):
        x = im10[i1][j1]
        if x:
            min_dis = float('inf')
            # px = py = -1
            for i2 in range(256):
                for j2 in range(256):
                    y = im11[i2][j2]
                    if y:
                        dis = sqrt((i2 - i1)**2 + (j2-j1)**2)
                        if dis < min_dis:
                            min_dis = dis

            im15[i1][j1] = min_dis
```

```
im12 = np.logical_not(im06) #Inverse White Matter(Median Filtered)
im13 = np.logical_not(im08) #Inverse Dark(Grey Matter) Matter(Median Filtered)
im14 = filters.median(im12 & im13) # Cortical Map(Filtered)(Gray Matter)
```

8.2 Calculating the thickness values and filling it in the Cortical Map

Finally, we are masking out the white and dark matter from the brain image to get the cortical mask and then we are filling every pixel on the cortical mask with the value stored at the nearest white

```
# Filling the thickness value in the Cortical Map
# im16 = np.full((256,256),0,dtype=float)
for i1 in range(256):
    for j1 in range(256):
        x = im14[i1][j1]
        if x:
            min_dis = float('inf')
            val = 0 # Closest value of White matter boundary point
            for i2 in range(256):
                for j2 in range(256):
                    y = im15[i2][j2]
                    if y > 0:
                        dis = sqrt((i2 - i1)**2 + (j2-j1)**2)
                        if dis < min_dis:
                            min_dis = dis
                            val = y
            # im16[i1][j1] = val if val <= 5 else 0
            final_image[i1][z][j1] = val if val <= 5 else 0 #Ignoring the error values
```

Step 9: Saving the final image

```
final_image = nib.Nifti1Image(final_image, None)
nib.save(final_image, "./final_thickness_map_fast.nii.gz")
```

Step 10: Final Outputs Obtained After Rendering the Result

Image 1: Rendered Slice 85 on z axis(depth) Thickness value obtained on arbitrary selected point 3.61

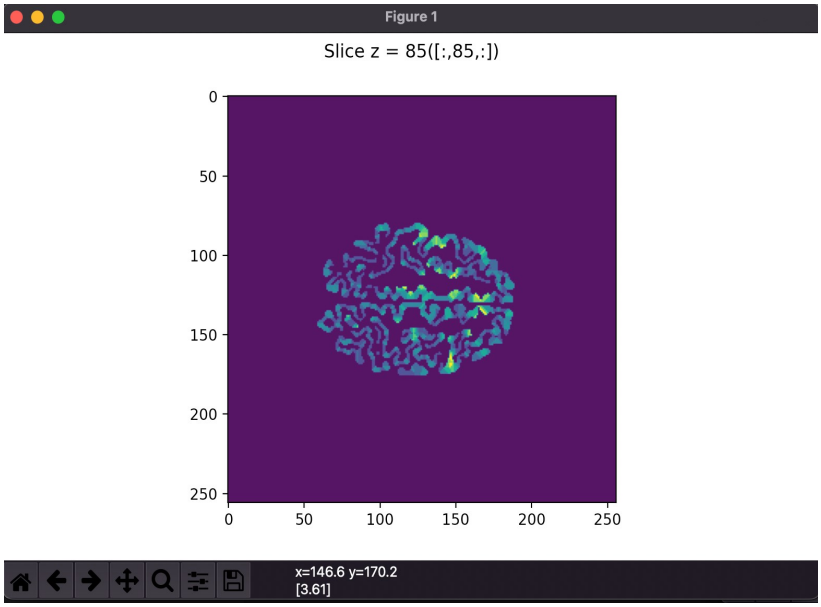


Image 1

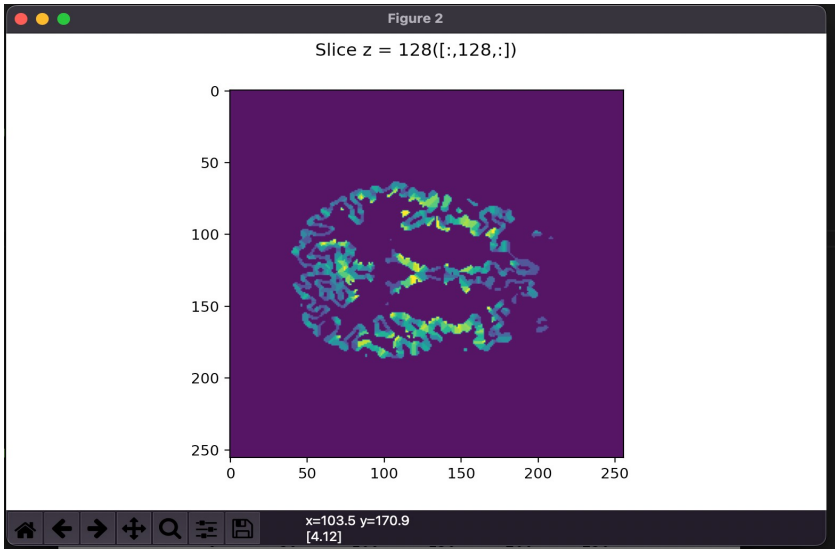


Image 2

Image 2: Rendered slice 128 Thickness value obtained from arbitrary selected point ß

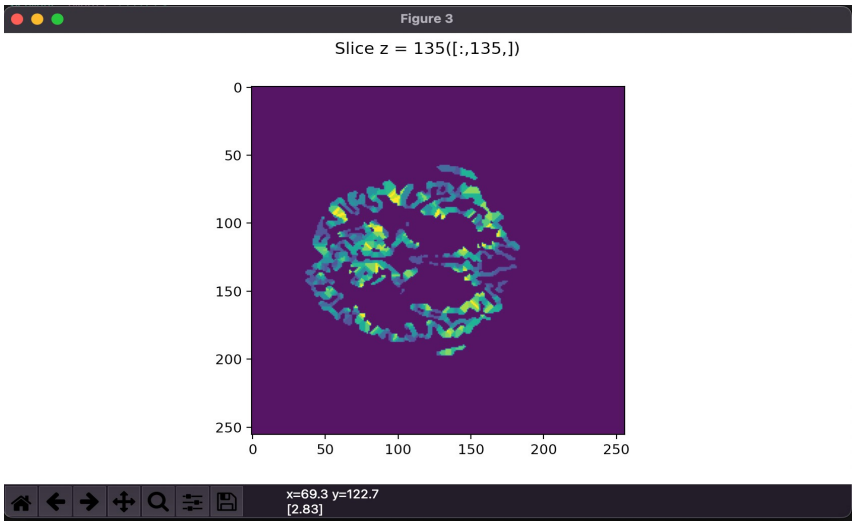


Image 3

ß

Image 3: Rendered slice 135, Thickness value obtained from arbitrary selected point 2.83ß

Challenges faced while completing the project

The biggest hurdle we faced while developing the proper solution to the problem was to separate the skull from the image and extract the brain. We struggled for many days on the same issue. After a lot of searching, we finally understood the process of extracting brain from the image using Convex_hull_image and expand_labels.

Another Challenge was developing the shortest point algorithms to calculate the thickness value, Which we overcome after and understanding different ways of implementation of distance formula from pixel to pixel.