Cortical Thickness Estimation
CS 590

Masters Project

Submission deadline: 08/08/2022

Submission date: 08/07/2022

Mahesh Karnakoti – 002306208

Vamshi Shalapaati – 002279326

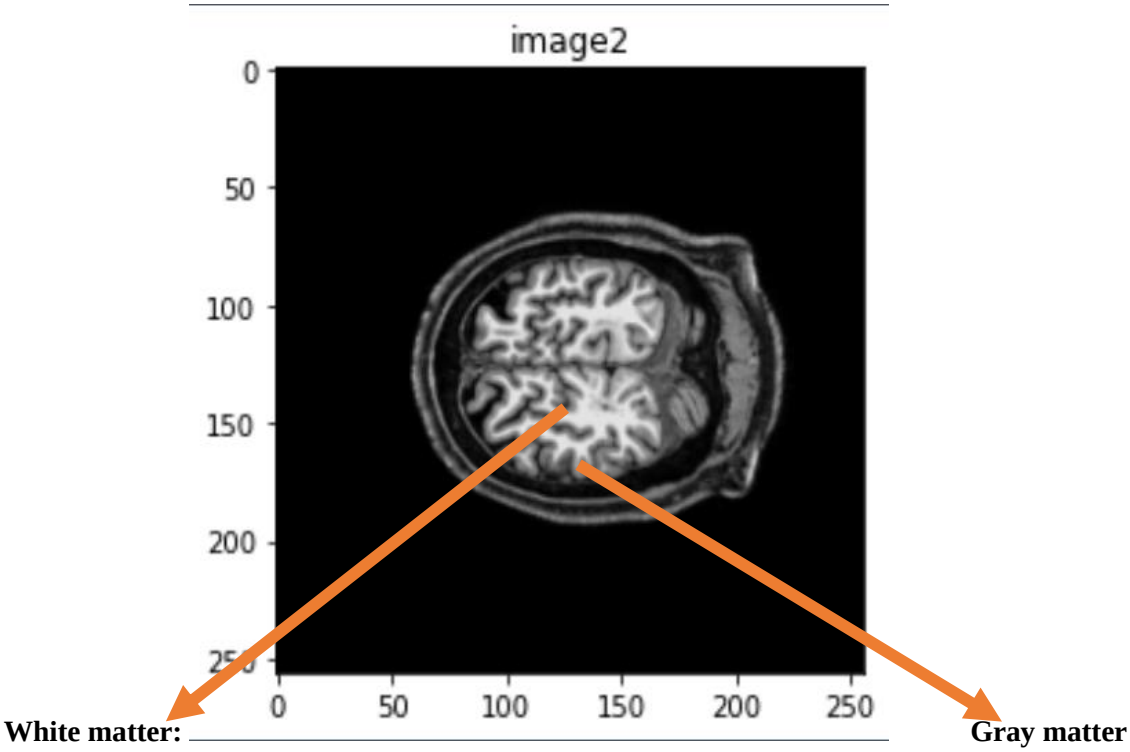
Introduction

In this project, we will calculate thickness of every part of cortical in nii brain image.

Main purpose of this project is to train image processing such as segmentation using pixel value, erosion and dilation. First of all, we will extract white matter using information of pixel difference. Using dilation, we will expand white matter and extract gray matter using subtraction and filtering and then, using erosion we will calculate thickness of cortical.

Detailed steps are as follows:

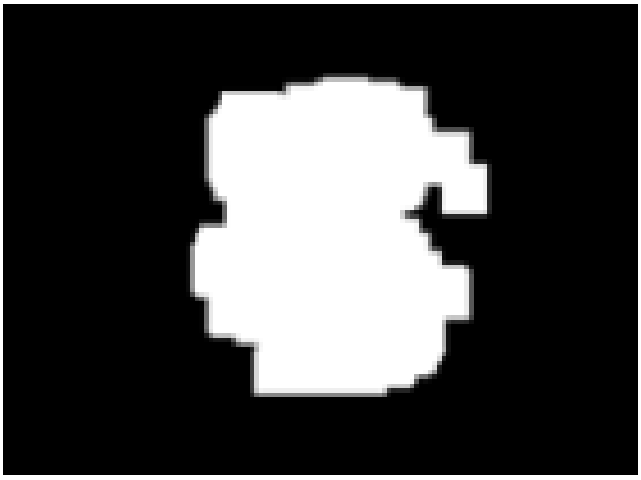
Variable explanation:



Brain image consists of white matter, gray matter(cortical) and bones. We don't have to consider bones. Only we should extract white matter and gray Matter. Gray matter means Cortical. But sometimes we call it Cortical Gray matter to distinguish white matter.

Cortical_brain matter: The part that Gray matter and White matter combined.

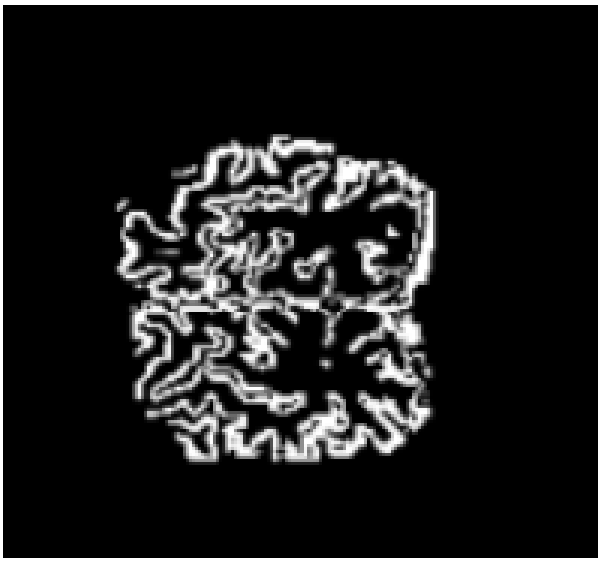
We get this using Dilation of white matter. So, there is noise. We should remove noise.



This is Cortical_brain matter.

Cortical_Thick: This is the gray matter (Cortical) that has noise removed.

We can remove noise using threshold because the pixels of gray matter are in 80~140.



This is Cortical_thick.

Thickness_combined: Finally result of cortical (assigned thickness as pixel value)

This is thickness_combined. (This will be explained in Main steps)

Main Steps

1. **Import .nii image file**

```
image2 = nib.load('raw_t1_subject_02.nii').get_fdata()
```

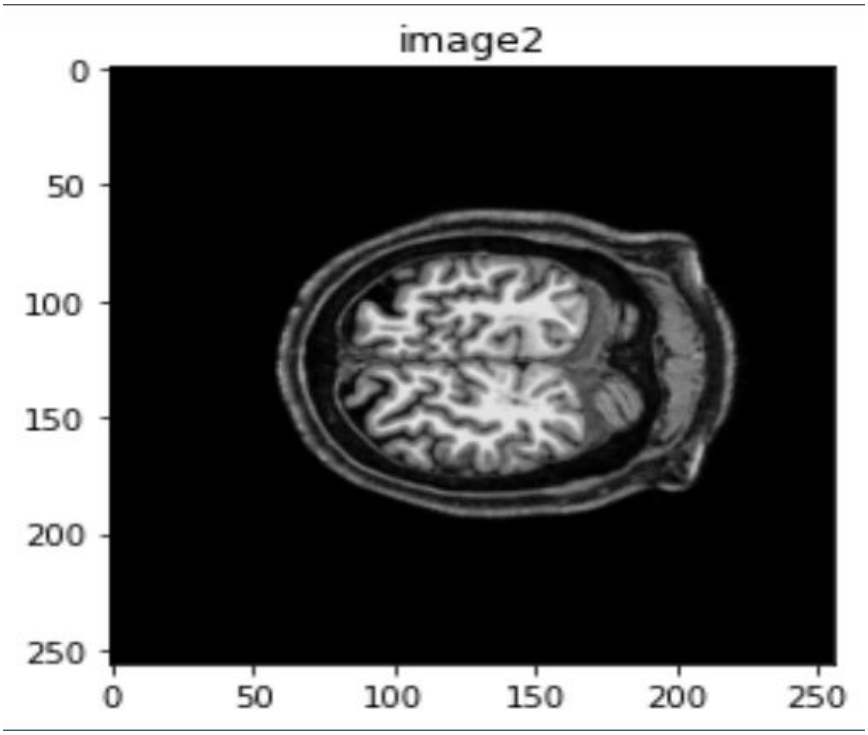


Fig 1. Original image

2. **Extract white matter using pixel information.**

To wide pixel value difference between white matter and other part, we normalized the image to (0, 255).

Classify pixel value into 2 parts – pixel of white matter and pixel of normal part.

The pixel value of white matter is greater than 140. Using this we can extract white matter.

```
def normalize(t):

    ma = np.max(t)
    if ma>0:
        for i in range(len(t)):
            for j in range(len(t[i])):
                t[i,j] = t[i,j]*255/ma
```

```
normalize(test2)
whitematter = np.zeros((256,256))
for i in range(len(test2)):
    for j in range(len(test2[i])):
        if test2[i,j]<140:
            whitematter[i,j] = 0
        else:
            whitematter[i,j] = 1

newwhite = np.zeros((256,256))
for i in range(len(whitematter)):
    for j in range(len(whitematter[i])):
       newwhite[i,j] = whitematter[i,j]
e = np.array((256,256))
```

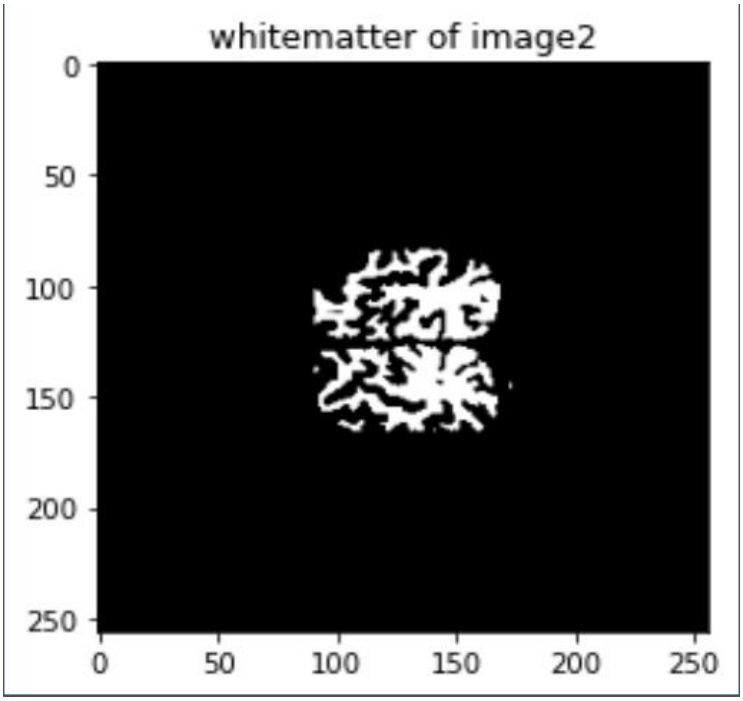
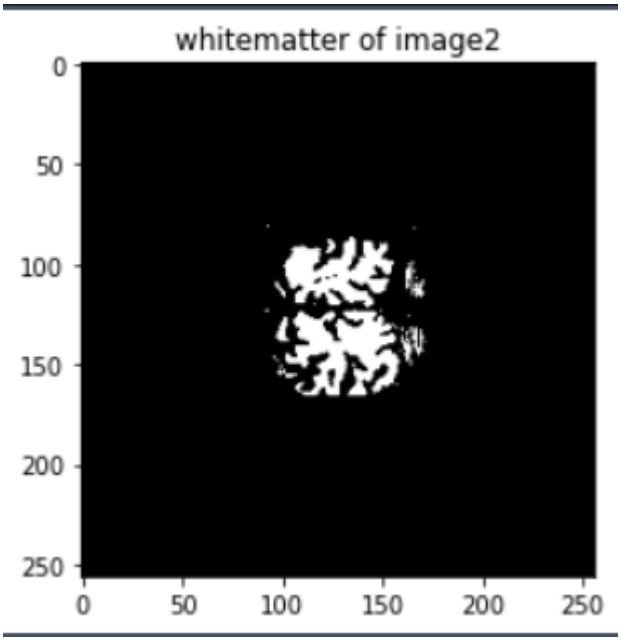


Fig2. Extracted white matter

3. Extraction Gray matter based on white matter.

Already white matter was extracted like this:



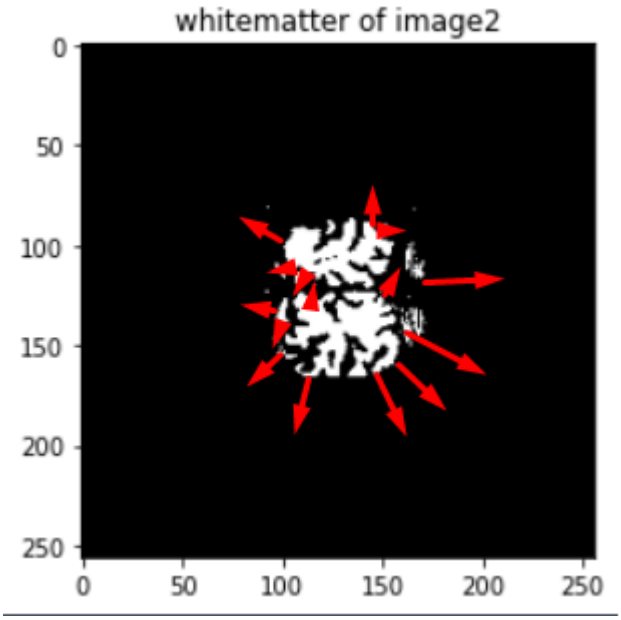


Fig3. Casting vector outwards

We should cast orthogonal vector outwards to detect gray matter to do this, I used well known vector casting function – dilation. By dilation, all elements are expanded outwards in direction of orthogonal vector.

Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image.

We used [3,3] kernel in Dilation as you can see.

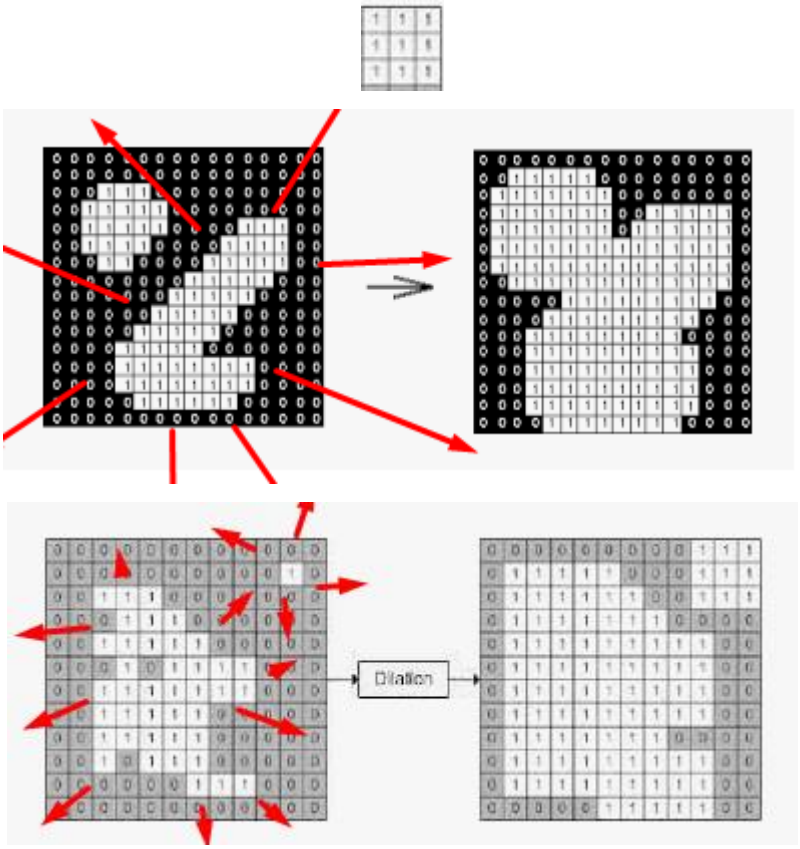


Fig 4. Image Dilation sample

The code of dilation is as follows:

```
def dilation(array):
  out = np.zeros((256,256))
  for i in range(1, array.shape[0] - 1):
    for j in range(2, array.shape[1] - 1):
      if array[i, j] >0:
        for m in [0, 1, -1]:
          for n in [0, 1, -1]:
            out[i + m, j + n] = array[i, j]
      else:
        continue

  return out
```

We applied this function to white matter several times and remove origin white matter.

```
Cortial_whitematter = erosion(whitematter)
Cortial_whitematter = dilation(Cortial_whitematter)
Cortial_whitematter = dilation(Cortial_whitematter)
Cortial_whitematter = dilation(Cortial_whitematter)
Cortial_whitematter = dilation(Cortial_whitematter)
Cortial_whitematter = dilation(Cortial_whitematter)
Cortial_whitematter = dilation(Cortial_whitematter)
whitematter = erosion(whitematter)
```

Fig 5. Extraction gray matter (Cortical) with noise

In result, there is so much noise in gray Matter so, we used filter. This filter will remove noise using pixel value difference between gray matter and noise

```
for i in range(len(whitematter)):
    for j in range(len(whitematter[i])):
        Cortial[i,j] =Cortial_whitematter[i,j]-whitematter[i,j]
```

```
for i in range(len(whitematter)):
    for j in range(len(whitematter[i])):
        if Cortial[i,j]>0 and test2[i,j]<140 and test2[i,j]>80:
            cortial_thick[i,j] = 1
```

We get this result

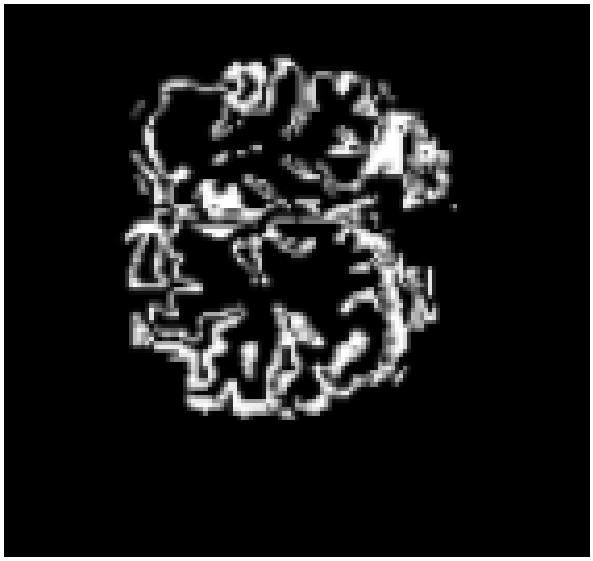


Fig 6. Filtered Cortical (noise removed)

4. We should calculate thickness of cortical.

So, I used erosion to calculate thickness. We found thickness by peeling it layer by layer in the direction of Vector using image morphing. Erosion removes pixels on object boundaries. In other words, it shrinks the foreground objects and enlarge foreground holes. Like in Image Processing Kernels, a larger size of the structure element, the greater the effect of erosion increase.

We used [3,3] kernel in erosion

```
def erosion(array):
  out = np.zeros((256,256))

  for i in range(1, array.shape[0] - 1):
    for j in range(2, array.shape[1] - 1):
      out[i,j] = array[i,j]

  for i in range(1, array.shape[0] - 1):
    for j in range(2, array.shape[1] - 1):
      if array[i, j] ==0:
        for m in [0, 1, -1]:
          for n in [0, 1, -1]:
            out[i + m, j + n] = 0
      else:
        continue

  return out
```

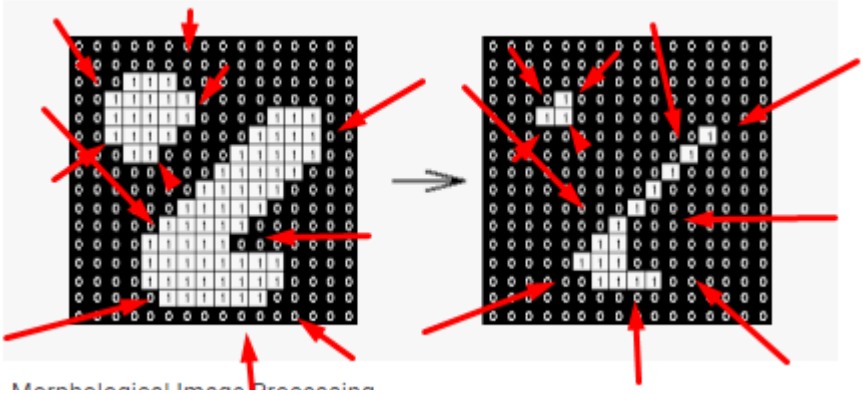


Fig 7. Image Erosion sample

“How many times did you perform erosion “means “How thick is this part of cortical”

When we perform erosion once, the outmost layer will be removed and the layer that has thickness 2 will remain.

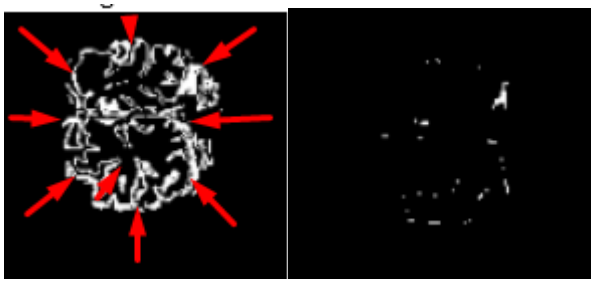


Fig 8. Peeling one layer to calculate thickness

This is one layer removed result (this means that this image illustrate layer that thickness >2). Using this method, we can calculate all thickness of cortical and then totalize all layers:

```
for i in range(len(whitematter)):
    for j in range(len(whitematter[i])):
        Thinkness_combined[i,j] = ersd_cortial[i,j]+cortial_thick[i,j]
```

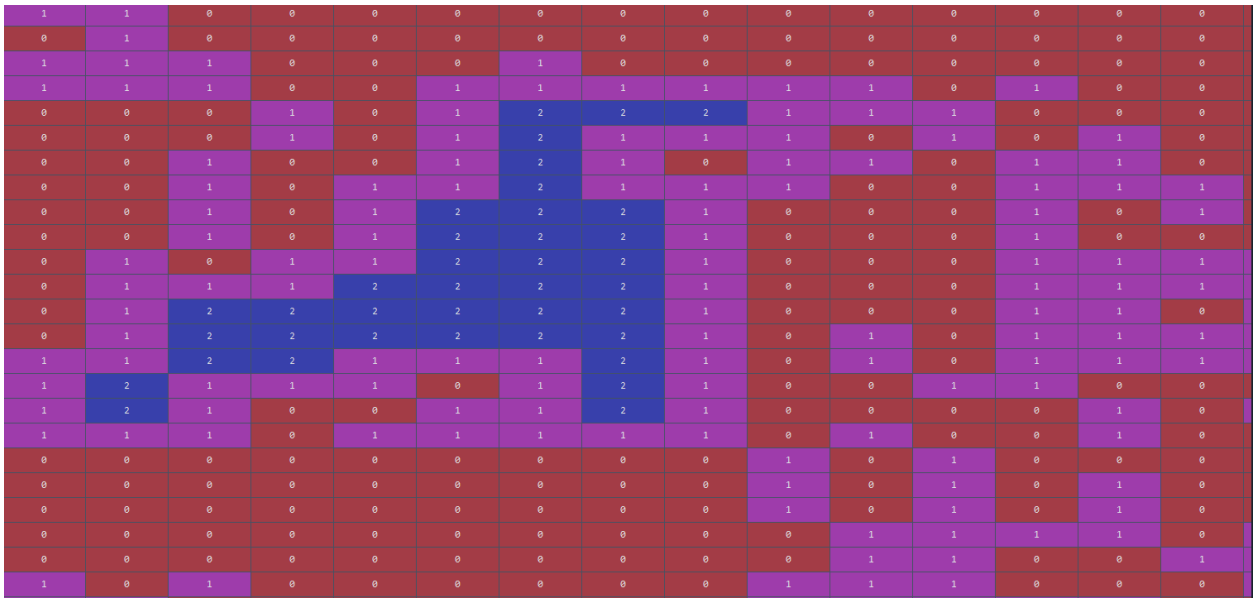


Fig 9. Pixel value of cortical

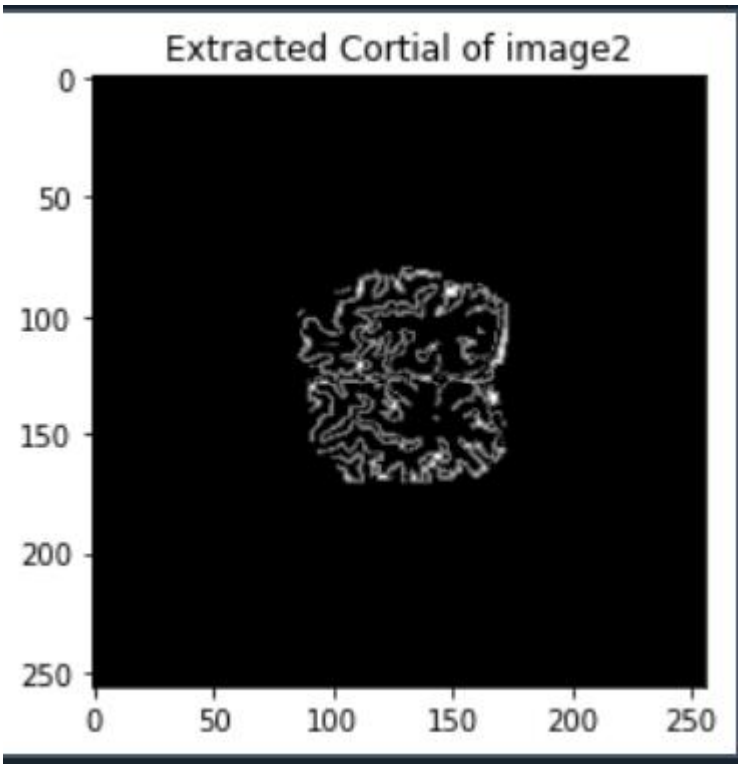


Fig 10. Cortical that assigned thickness (cortical thickness)

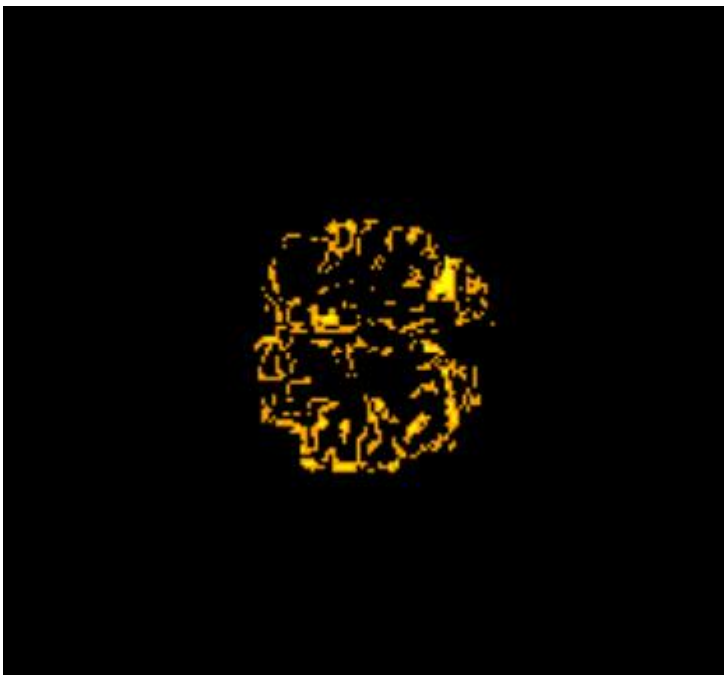
5. Below image shows the thickness map for image2 after segmentation

Fig 11. Thickness map for image2

Then, we assign color to thickness. Thick: Red, Thin: Yellow/Orange

6. Overlayed image

We assign new color to Cortical and overlay it to original image



We combined this with original image.

```
cmap = ListedColormap(["black", "orange", "yellow"])
plt.imshow(Thinkness_combined,cmap = cmap)
plt.imsave("Thickness_map_image2.png",Thinkness_combined)
plt.title('Extracted Cortial of image2(Orange and yellow)')
```

```
plt.figure(9)
plt.imshow(test2, 'gray')
for i  in range( 256):
    for j in range( 256):
        if Thinkness_combined[i,j]==1:
            plt.scatter(j,i,s =0.4, color = 'red')
        if Thinkness_combined[i,j] == 2:
            plt.scatter(j, i, s = 0.8,color = 'orange')


plt.title("Overlayed")
```

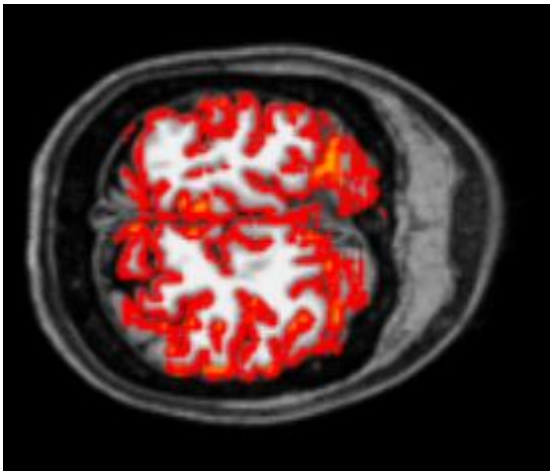


Fig 10. Overlayed image

Conclusion

In this paper, we used segmentation, dilation to cast vector outwards and erosion to calculate thickness. We reached at least 85 % accuracy. In the future, we will increase this step by step.