CS516 Assignment4 report

Group member: Yukuan Hao, Liang Gu

Part1

We use Non local means and bilateral filter to do this part

Non local means Code:

```
def NLmeansfilter(Image, h, templateWindowSize,searchWindowSize):
    f = int(templateWindowSize/2)
    t = int(searchWindowSize/2)
    kernel = np.zeros((2*f+1, 2*f+1))
    for d in range(1, f+1):
        kernel[f-d:f+d+1, f-d:f+d+1] += (1.0/((2*d+1)**2))
    kernel=kernel/kernel.sum()
    PaddedImg = np.pad(Image,t+f,'symmetric')
    padlength=t+f
    DenoisedImg=np.zeros((Image.shape[0],Image.shape[1]))
    height, width = Image.shape[:2]
    Image_wd = PaddedImg[padlength-f:padlength+f+height, padlength-f:padlength+f+width]
    average = np.zeros(Image.shape)
    sweight = np.zeros(Image.shape)
    wmax =  np.zeros(Image.shape)
    for i in range(-t, t+1):
            for j in range(-t, t+1):
                if i==0 and j==0:
                    continue
                PaddedImg1 = PaddedImg[padlength+i-f:padlength+i+f+height,\
                                       padlength+j-f:padlength+j+f+width]
    #           w = np.exp(-cv2.filter2D((PaddedImg1 - T1_wd)**2, -1,\
    #                           kreturn)/h**2)[f:f+height, f:f+width]
                b=(PaddedImg1 - Image_wd)**2
                c=kernel
                dist=signal.convolve2d(b,c,boundary='symm',mode='same')
                w = np.exp(-dist/h**2)[f:f+height, f:f+width]
                sweight += w
                wmax = np.maximum(wmax, w)
                average += (w*PaddedImg1[f:f+height, f:f+width])
    DenoisedImg=(average+wmax*Image)/(sweight+wmax)
    return DenoisedImg
```

SNR code:

```
def snr(A, B):

    return np.mean(B.astype(np.float))/np.std(A.astype(np.float))
#psnr 10*np.log(255*255.0/(((A.astype(np.float)-B)**2).mean()))/np.log(10)
'''
```

We use a 10*10 windows to get noise patch and signal patch, here is our code:

```
snr_T1=snr(T1[320:330,62:72],T1[290:300,70:80])
snr_TFlair=snr(Flair[440:460,30:50],Flair[405:425,90:110])
snr_T1v2=snr(T1_v2[212:222,22:32],T1_v2[196:206,40:50])
snr_T1v3=snr(T1_v3[210:220,60:70],T1_v3[190:200,100:110])
snr_T2=snr(T2[302:312,50:60],T2[260:270,70:80])


#snr of denoised non local means
snr_T1d=snr(T1_den[320:330,62:72],T1_den[290:300,70:80])
snr_TFlaird=snr(Flair_den[440:460,30:50],Flair_den[405:425,90:110])
snr_T1v2d=snr(T1v2_den[212:222,22:32],T1v2_den[196:206,40:50])
snr_T1v3d=snr(T1v3_den[210:220,60:70],T1v3_den[190:200,100:110])
snr_T2d=snr(T2_den[302:312,50:60],T2_den[260:270,70:80])
#bilateral filter
snr_T1d2=snr(T1_den2[320:330,62:72],T1_den2[290:300,70:80])
snr_TFlaird2=snr(Flair_den2[440:460,30:50],Flair_den2[405:425,90:110])
snr_T1v2d2=snr(T1v2_den2[212:222,22:32],T1v2_den2[196:206,40:50])
snr_T1v3d2=snr(T1v3_den2[210:220,60:70],T1v3_den2[190:200,100:110])
snr_T2d2=snr(T2_den2[302:312,50:60],T2_den2[260:270,70:80])
```

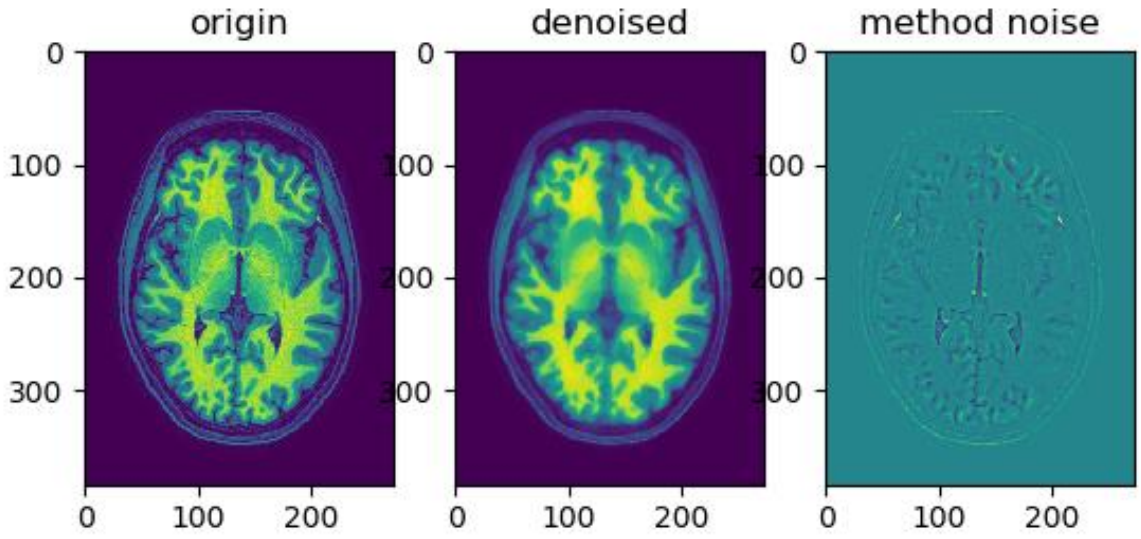
Bilateral filter code:

```
def bilateral_filter(img,kernel_size=5):
    img=img.astype(int)
    output=np.zeros(img.shape)
    d =int ((kernel_size-1)/2)
    row=img.shape[0]
    col=img.shape[1]
    sigmaspace=(kernel_size/2-1)*0.3+0.8
    pix_avg=np.mean(img)
    pix_max=np.max(img)
    sigmacolorsquare=np.sum((img-pix_avg)*(img-pix_avg))/(row*col)
    for i in range(d,row-d):
        for j in range(d,col-d):
            weightsum=0
            filtervalue=0
            for k in range (-d,d+1):
                for l in range (-d,d+1):
                    distance_square=k*k+l*l
                    value_square=(img[i][j]-img[i+k][j+l])*\
                        (img[i][j]-img[i+k][j+l])
                    weight=np.exp(-1 * (distance_square / \
                                        (2 * sigmaspace*sigmaspace)+ \
                                            value_square / (2 * sigmacolorsquare)))
                    weightsum += weight
                    filtervalue += (weight*img[i+k][j+l])
            output[i][j]=filtervalue/weightsum
    return output
```

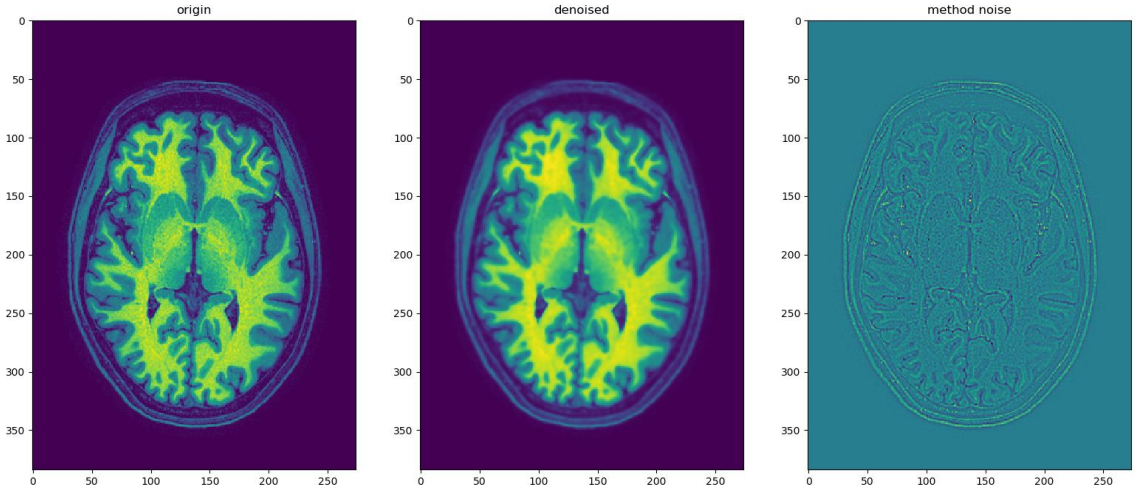
Result:

1: t1.png:

Non local means filter result:



Bilateral filter result:



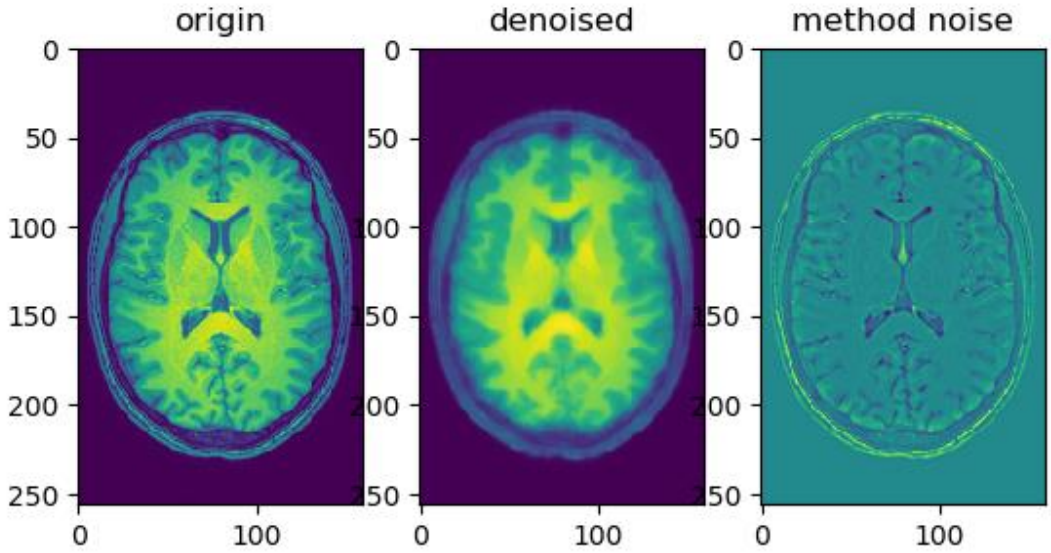
Origin SNR=29.31

Non local means result SNR=56.98

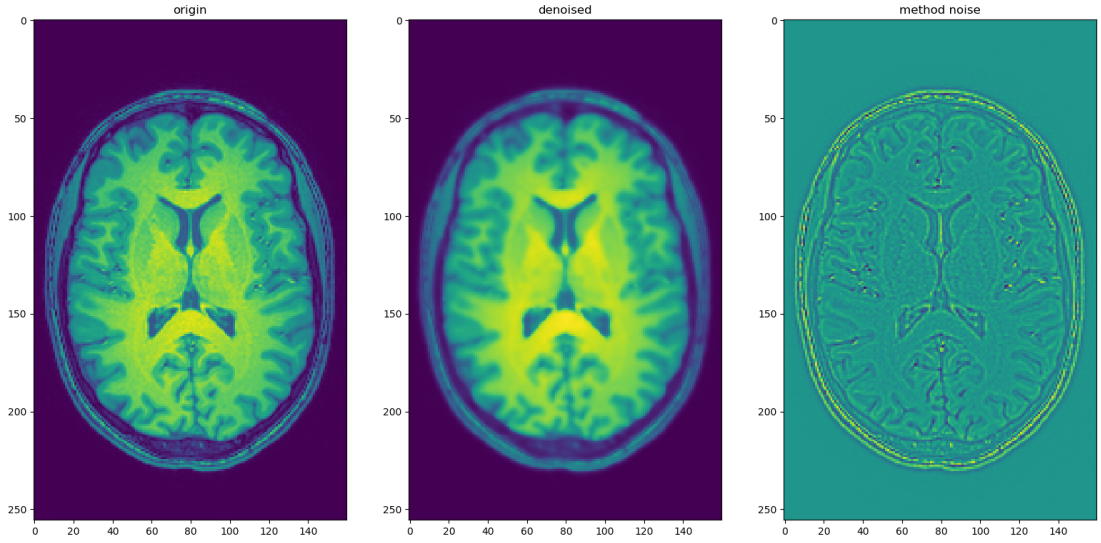
Bilateral filter result SNR=39.98

2: t1_v2.png :

Non local means filter result:



Bilateral filter result:



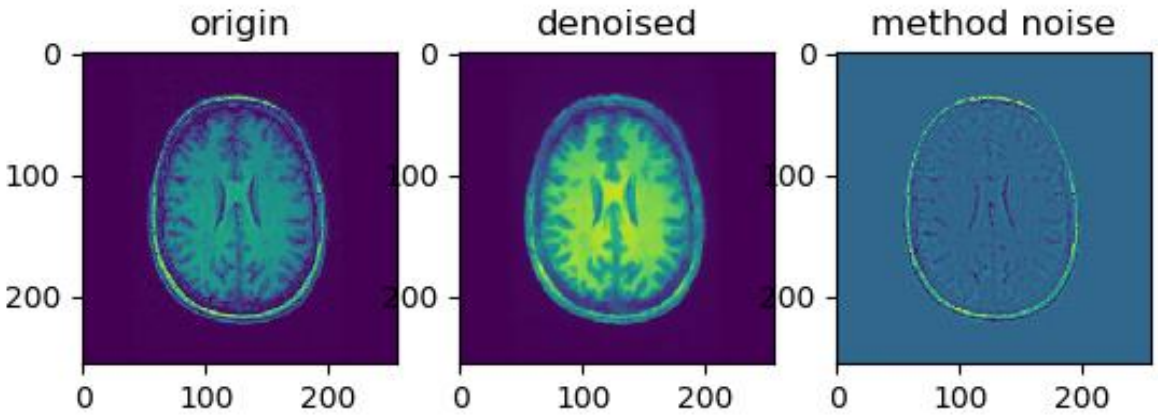
Origin SNR=160.36

Non local means result SNR =221.79

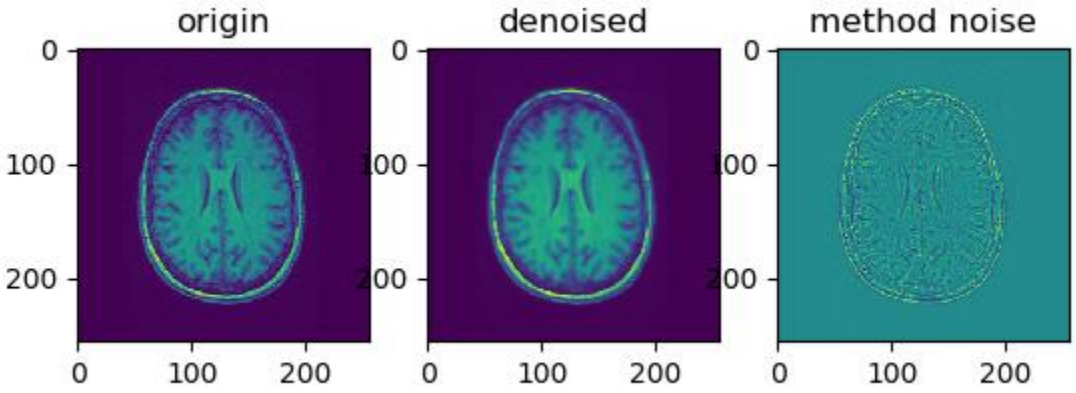
Bilateral filter result SNR=147.91

3: t1_v3.png

Non local means filter result:



Bilateral filter result:



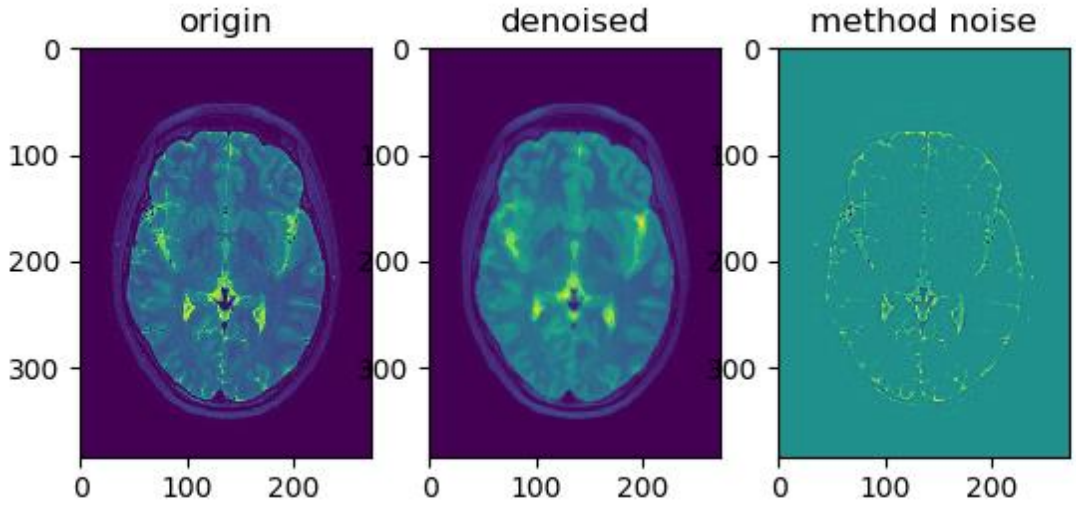
Origin SNR=76.59

Non local means result SNR =487.61

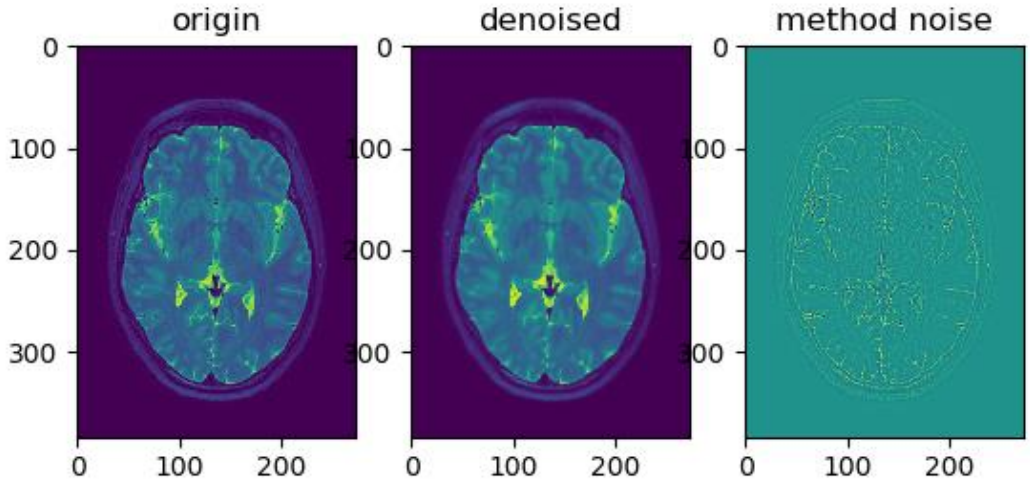
Bilateral filter result SNR=276.35

4: t2.png

Non local means filter result:



Bilateral filter result:



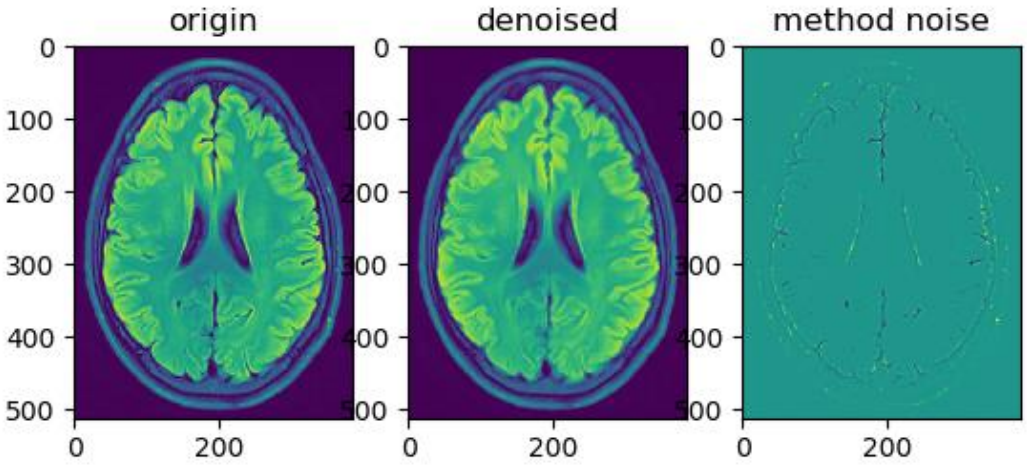
Origin SNR=57.23

Non local means result SNR =80.37

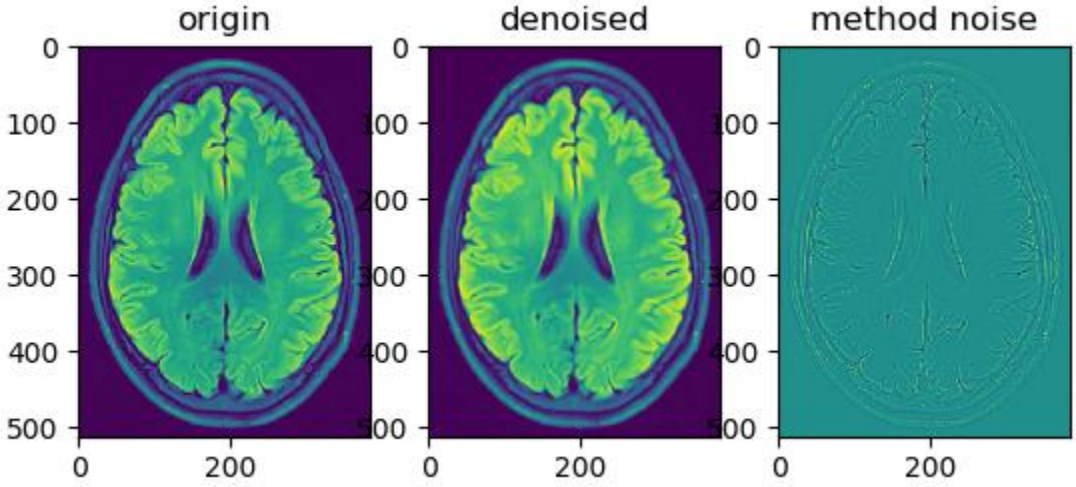
Bilateral filter result SNR=53.77

5: flair.png:

Non local means filter result:



Bilateral filter result:



Origin SNR=101.63

Non local means result SNR =274.91

Bilateral filter result SNR=171.93

Part2:

We choose Otsu's method to segment the gray matter in those 5 images.

Otsu's method is to divide the image into foreground and background, in order to find the gray matter in image, we need to determine whether the gray matter is in the foreground or the background. If the gray matter in foreground with other part like white matter, we should use Otsu's method again to separate the foreground into new foreground and new background and determine whether the gray matter is in the new foreground or the new background, and we will do this step as a loop until we got the satisfied result, here is our codes and results:

Code segments of functions:

```
def otsu(img):
    x=img.shape[0]
    y=img.shape[1]
    histogram=np.zeros(256)
    for i in range (0,x):
        for j in range (0,y):
            histogram[int(img[i][j])]+=1
    size=x*y
    u=float(0)
    for i in range (0,256):
        histogram[i]=histogram[i]/size
        u+=i*histogram[i]
    threshold=0
    maxvariance=float(0)
    w0=float(0)
    avgvalue=0
    for i in range (0,256):
        w0+=histogram[i]
        avgvalue+=i * histogram[i]
        t=float(avgvalue/w0-u)
        variance = float(t * t * w0 /(1 - w0))
        if variance>maxvariance:
            maxvariance=variance
            threshold=i
    return threshold
```

```python
def otsu_d(img,threshold1,threshold2):#threshold1<threshold2
    if threshold1>threshold2:
        temp=threshold1
        threshold1=threshold2
        threshold2=threshold1
    x=img.shape[0]
    y=img.shape[1]
    histogram=np.zeros(256)
    for i in range (0,x):
        for j in range (0,y):
            histogram[int(img[i][j])]+=1
    u=float(0)
    histogram1=histogram[threshold1:threshold2]
    size=np.sum(histogram1)
    for i in range (0,threshold2-threshold1):
        histogram1[i]=histogram1[i]/size
        u+=i*histogram1[i]
    threshold=0
    maxvariance=float(0)
    w0=float(0)
    avgvalue=0
    for i in range (0,threshold2-threshold1):
        w0+=histogram1[i]
        avgvalue+=i * histogram1[i]
        t=float(avgvalue/w0-u)
        variance = float(t * t * w0 /(1 - w0))
        if variance>maxvariance:
            maxvariance=variance
            threshold=i
    return threshold+threshold1
```

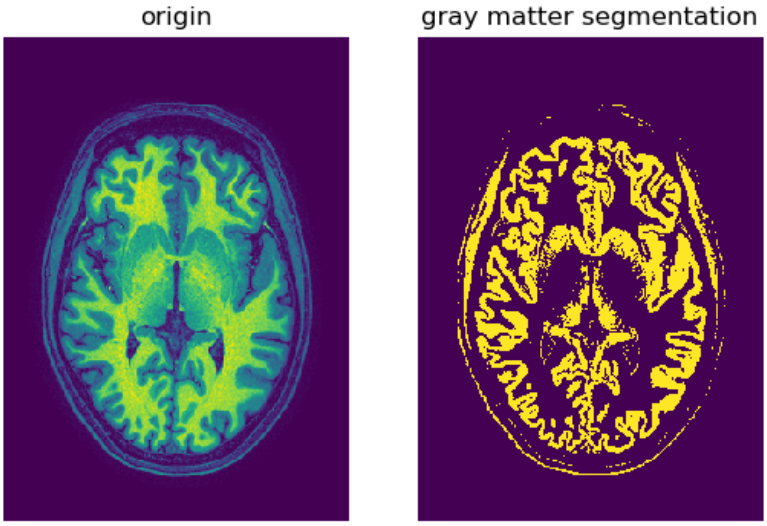
```python
def create_binary(img,threshold):
    binary=np.zeros(img.shape)
    binary[:]=img[:]
    binary[np.where(binary<threshold)]=0
    binary[np.where(binary>=threshold)]=1
    return binary
def create_binary_2(img,t1,t2):
    binary=np.zeros(img.shape)
    binary[:]=img[:]
    binary[np.where(binary<t1)]=0
    binary[np.where(binary>t2)]=0
    binary[np.where(binary>=t1)]=1
```

Code segments of creating binary image(use T1 as example):

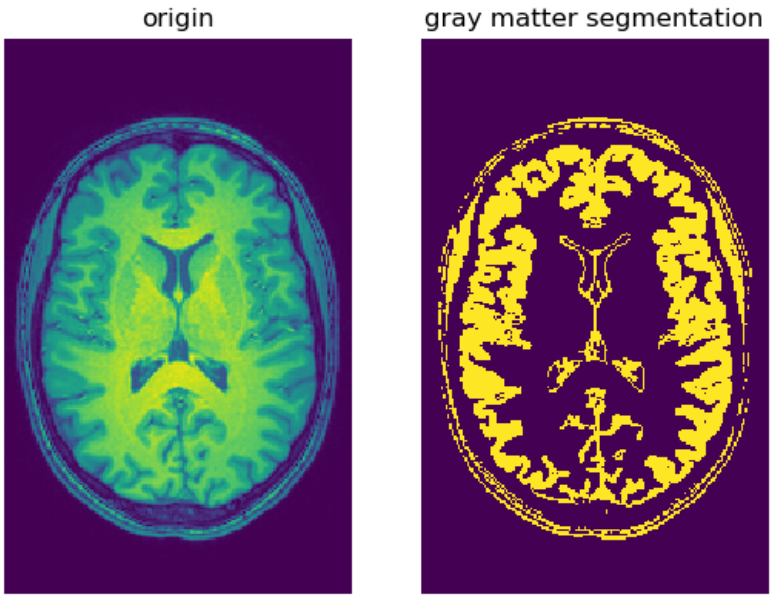
```
tT1=otsu(T1)
tOTSUT1=otsu_d(T1,tT1,256)
OTSUT1=create_binary_2(T1,tT1,tOTSUT1)
plt.figure()
ax1=plt.subplot(121)
plt.axis('off')
ax1.set_title('origin')
ax2=plt.subplot(122)
plt.axis('off')
ax1.imshow(T1)
ax2.set_title('gray matter segmentation')
```

Results:

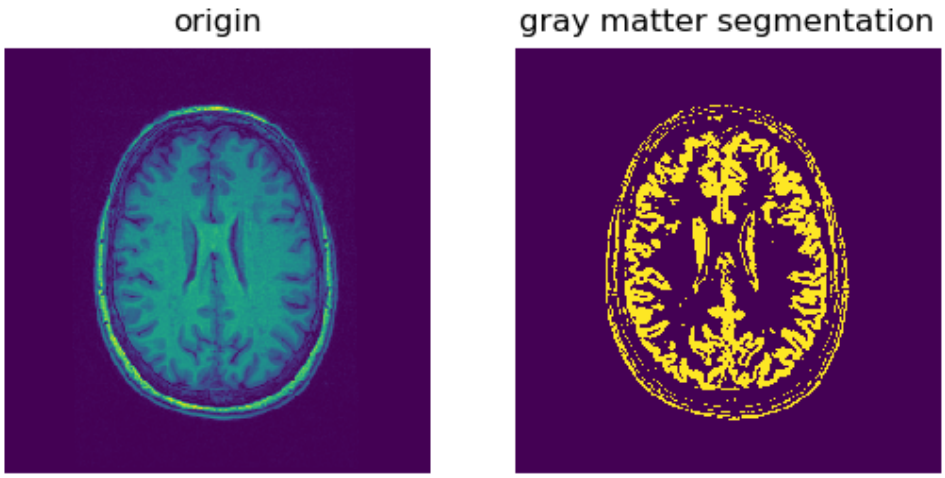
(1)t1.png:



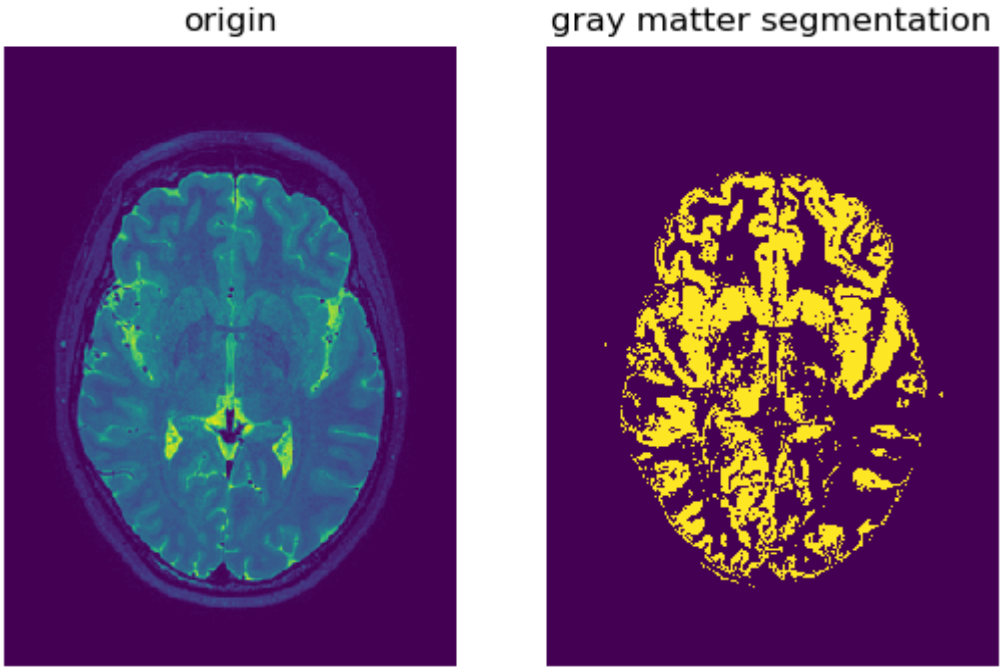
(2) t1_v2.png:



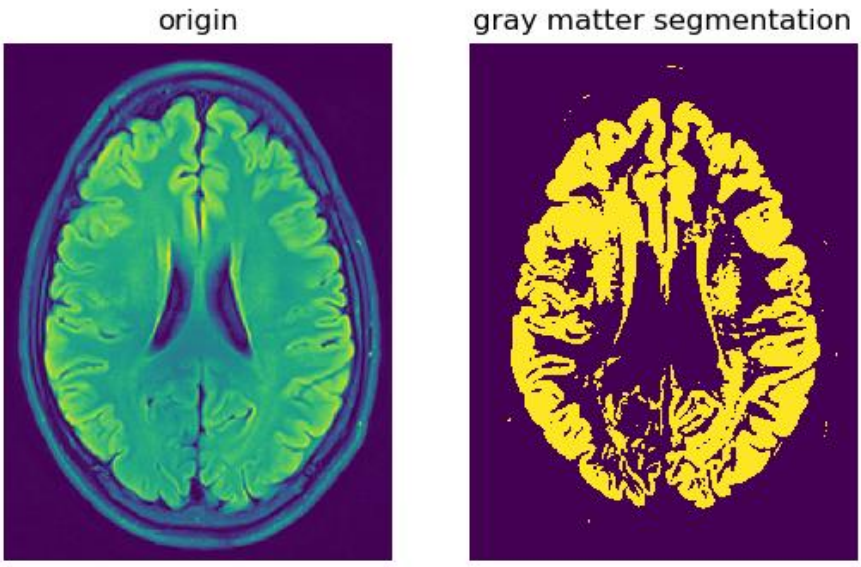
(3)t1_v3.png:



(4) t2.png



(5)flair.png



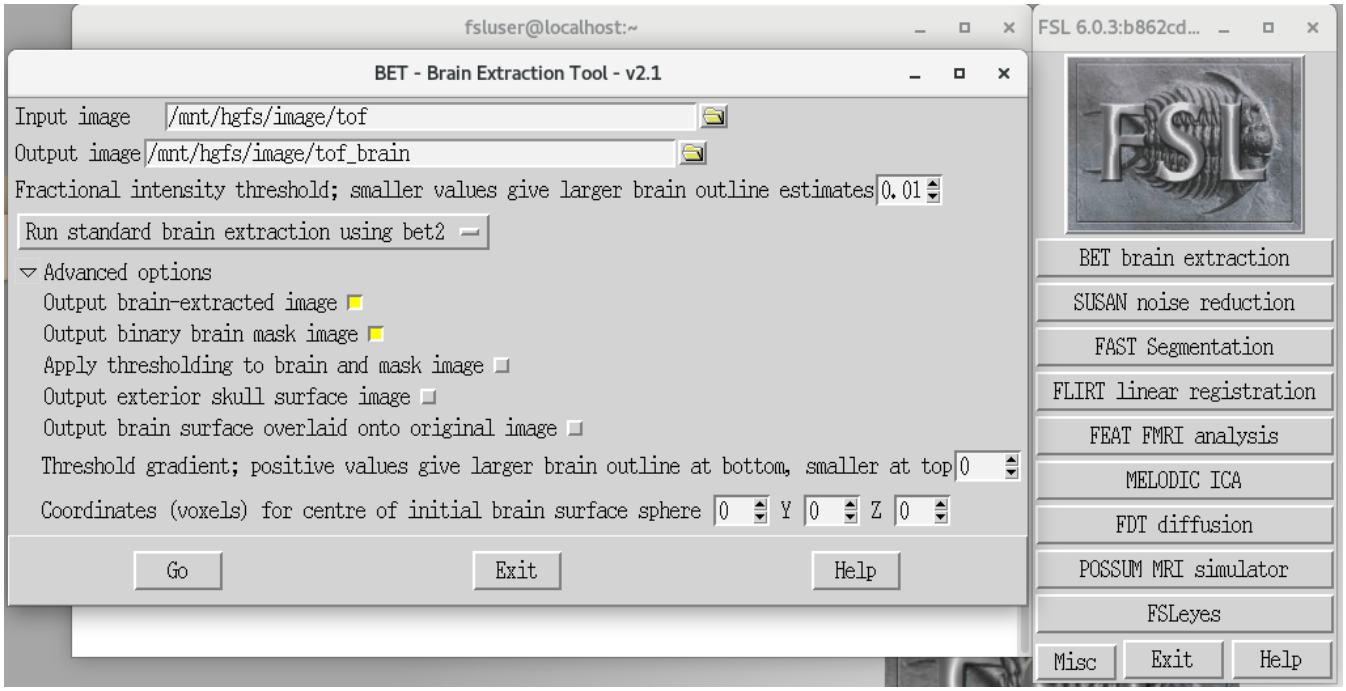
Conclusion:

From those five result images, we can see some images contain skulls, some images contain parts of white matters, it is because there are part of skull and white matter pixel values are very similar to gray matter pixel values. Only use a threshold to segment gray matter is not enough. For example, if gray matter's pixel value is around 80-120, skull's pixel value is around 60-100, and white matter's value is around 110-200, whatever threshold we select, our result will be the part of gray matters or gray matters with skull or gray matter with white matters.

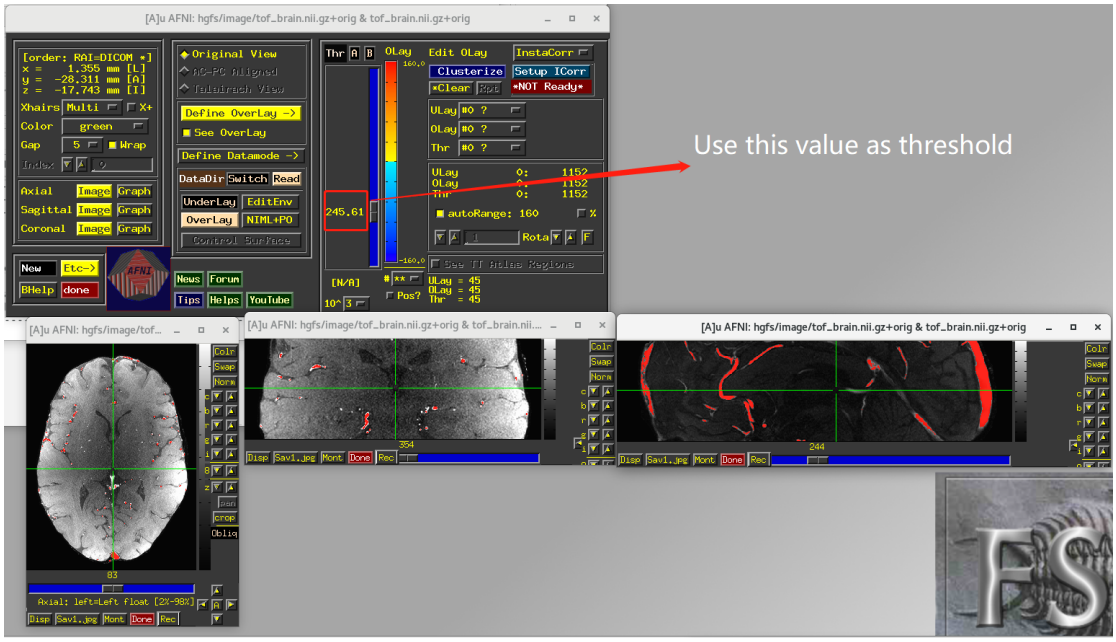
Part3:

segment the arteries from the tof

first, we need to remove skull from the image, we use Brain Extraction Tool from FSL



then, we use afni to see our result and select a threshold

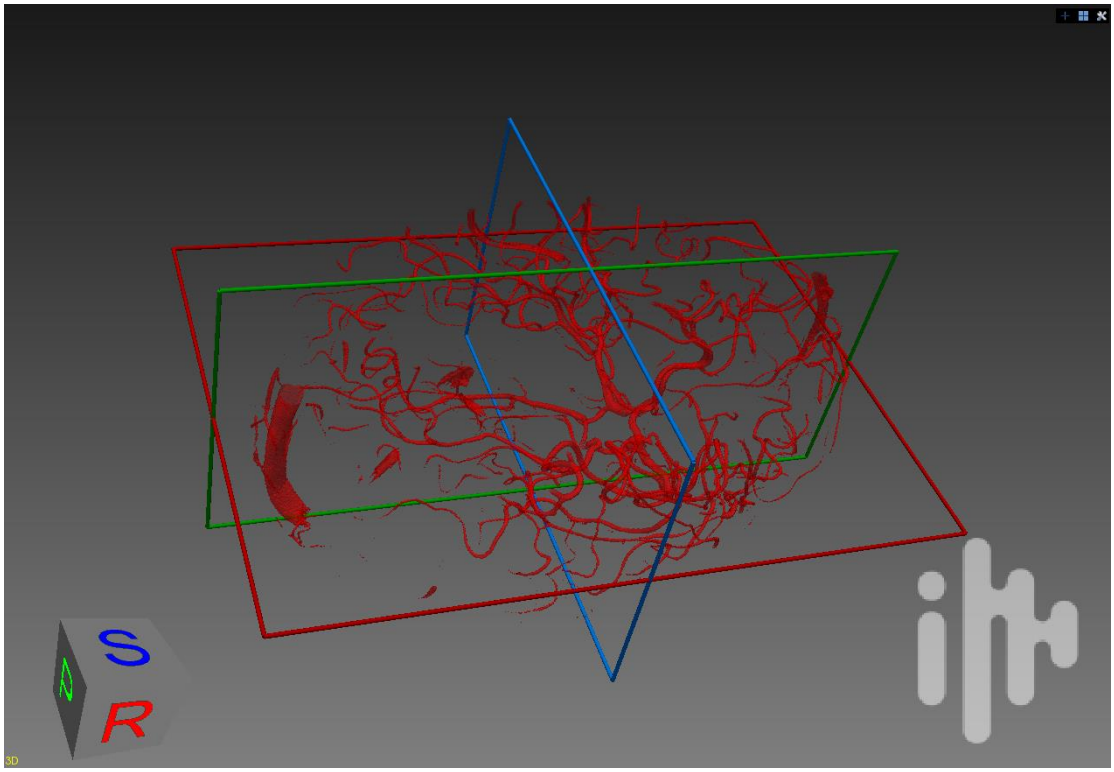


we use threshold to create a binary image

code segment:

```
import nibabel
import numpy as np
import matplotlib.pyplot as plt
from skimage.filters import frangi, hessian
img_data=nibabel.loadsave.load('I:/image/tof_brain.nii')
img=img_data.get_fdata()
#img2=frangi(img,black_ridges=False)
img[np.where(img<245)]=0
img[np.where(img>=245)]=1
save_file=nibabel.Nifti1Image(img, img_data.affine)
nibabel.save(save_file,'I:/image/tofseg_final.nii')
```

visualize the image in MIBRAIN

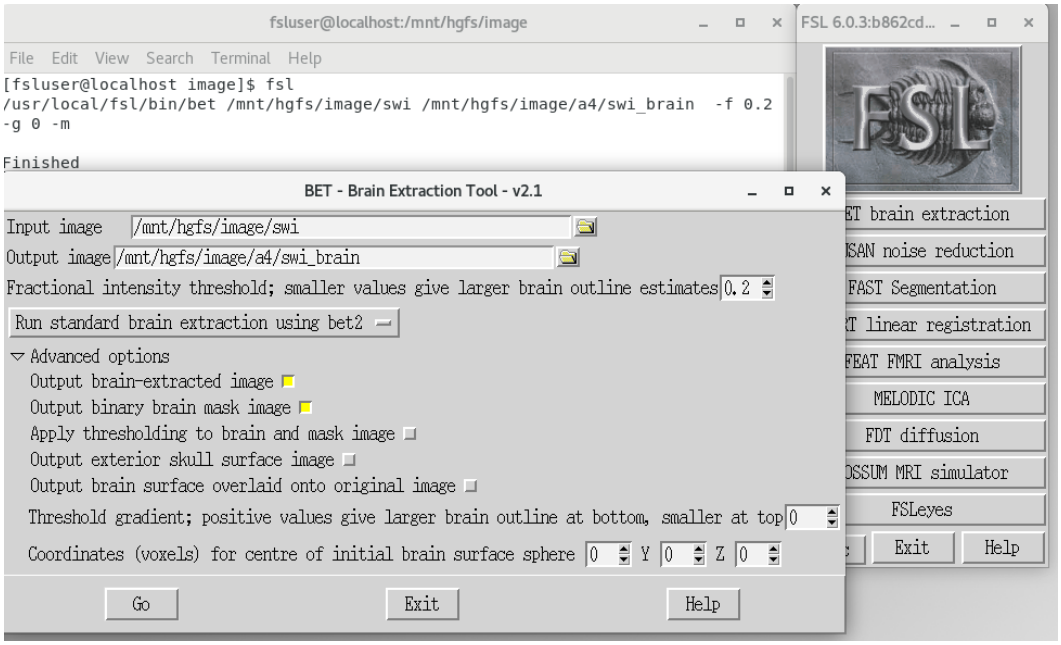


The link to tofseg_final.nii:
https://drive.google.com/file/d/1ztsHoCEuhlXDBJ53fvbMEGARpdou0mSL/view?usp=sharing

The link to tofseg_final.vtp:
https://drive.google.com/file/d/1cdvcNAIQ4FEchYE015P3cswex9WFZ7xY/view?usp=sharing

segment the veins from the swi

first, we need to remove skull from the image, we use Brain Extraction Tool from FSL

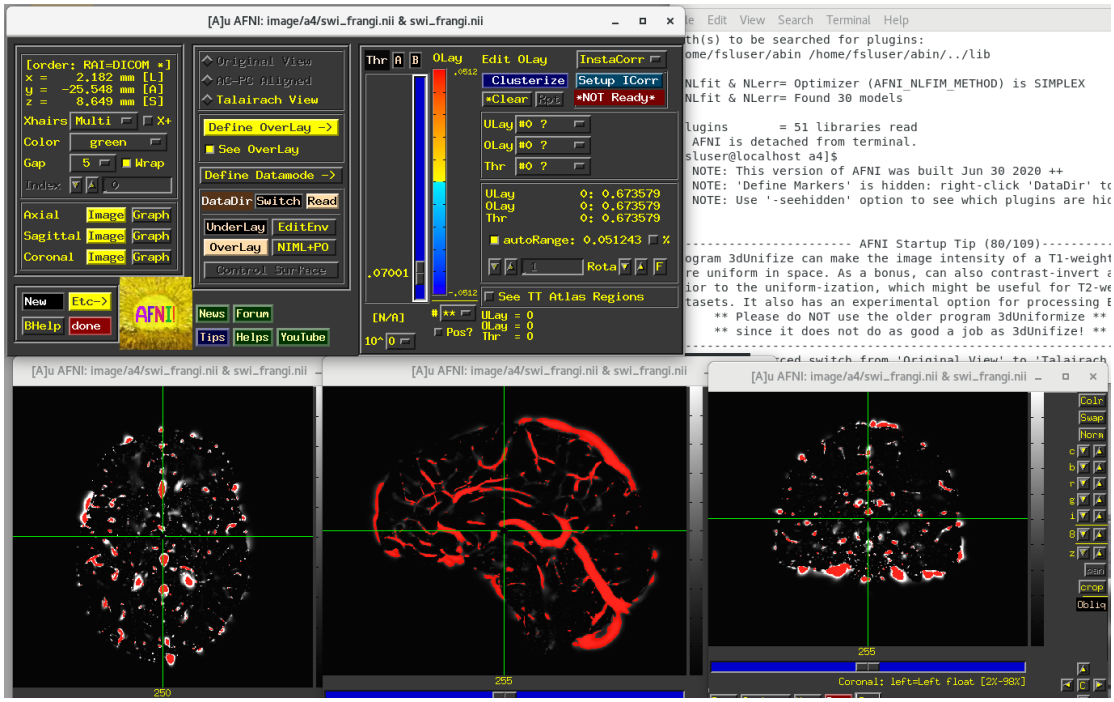


then, we apply frangi filter to preprocess the image,

code segment:

```
from skimage.filters import frangi, hessian
img_data=nibabel.loadsave.load('I:/image/a4/swi_brain.nii')
img=img_data.get_fdata()
img=frangi(img)
save_file=nibabel.Nifti1Image(img, img_data.affine)
nibabel.save(save_file,'I:/image/a4/swi_frangi.nii')
```

then, we use afni to see our result and select a threshold

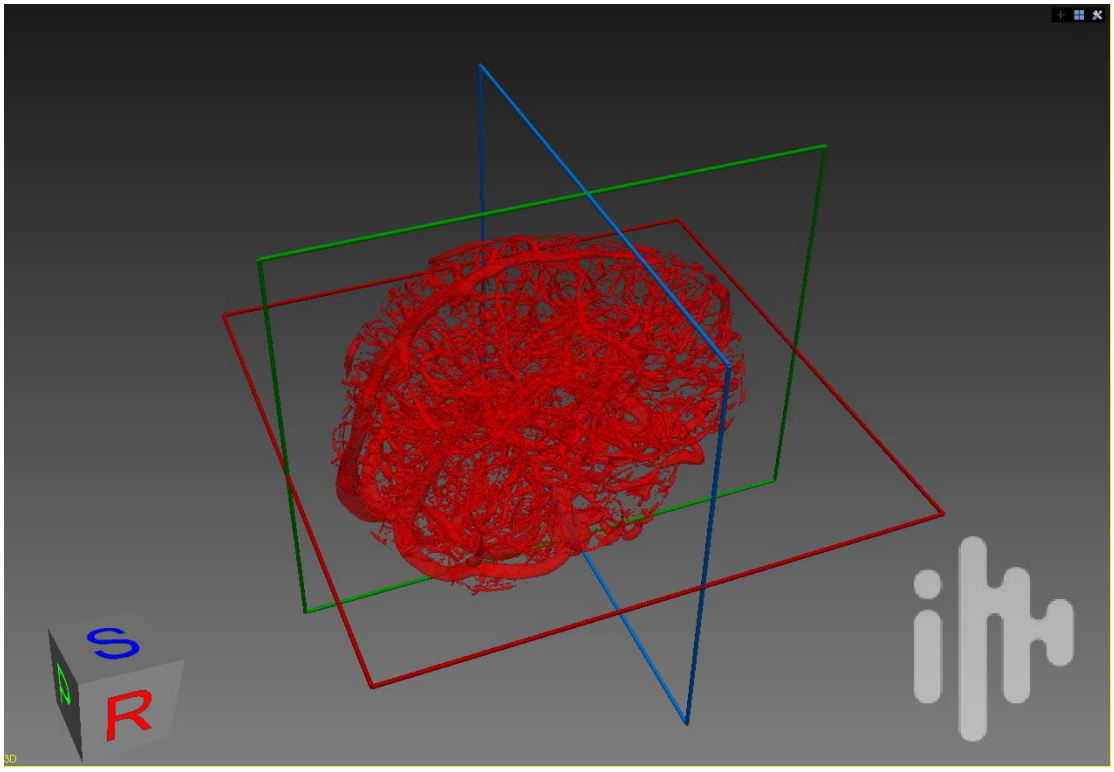


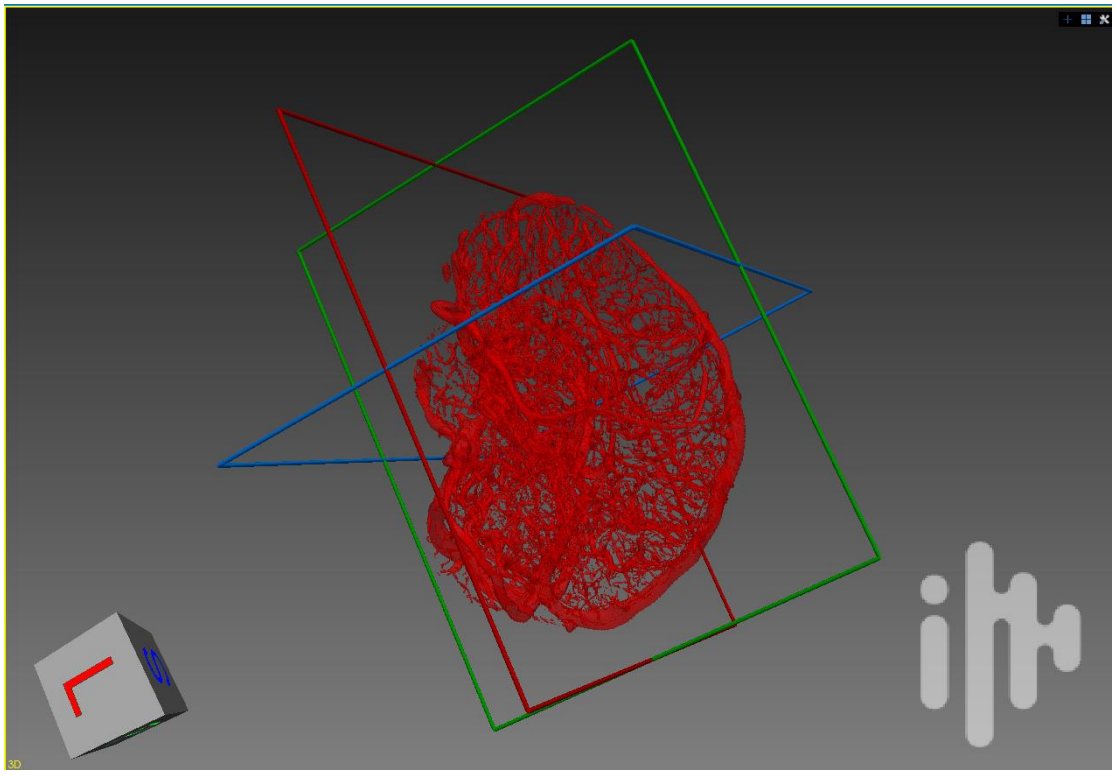
we use threshold to create a binary image

code segment:

```
img[np.where(img<0.07)]=0
img[np.where(img>=0.07)]=1
save_file=nibabel.Nifti1Image(img, img_data.affine)
nibabel.save(save_file,'I:/image/a4/swibrainseg_final.nii')
```

visualize the image in MIBRAIN





The link to swibrainseg_final.nii:

https://drive.google.com/file/d/1c8XRTue-8jDcnPrmnXJQAnDxOV0dUQrt/view?usp=sharing

The link to swibrainseg_final.vtp:

https://drive.google.com/file/d/1utoiduFOgZAHMCiST1gWrlMOfhWD4ccd/view?usp=sharing