Math 320 Research Project

Topic: Extraction of Brain Tumors from MRI scans

Name: Varun S Rajagopal

Student ID: 73342019

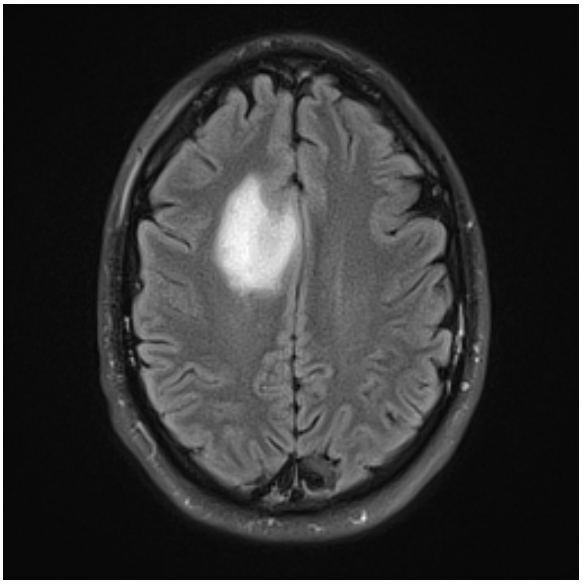
Date: November 27th, 2016

MATH 320 Fall 2016

Introduction

Brain tumors refer to the uncontrolled growth of cells in tissues of the brain that can prove detrimental and ultimately have serious consequences on an individual's well being. As a result, the ability to detect such tumors is of immense importance to saving lives and reducing suffering. Fortunately, developments in technology, in particular, through MRI (magnetic resonance imaging) scans help us to detect these growths at an early stage. MRI scans produce high quality images that display the brain along with these tumors (white regions) as the following sample images in figures 1 and 2 illustrate. By contrast, figure 3 shows an MRI of a healthy brain without any white regions.

Figures 1 & 2 – Brain Tumor MRI scans



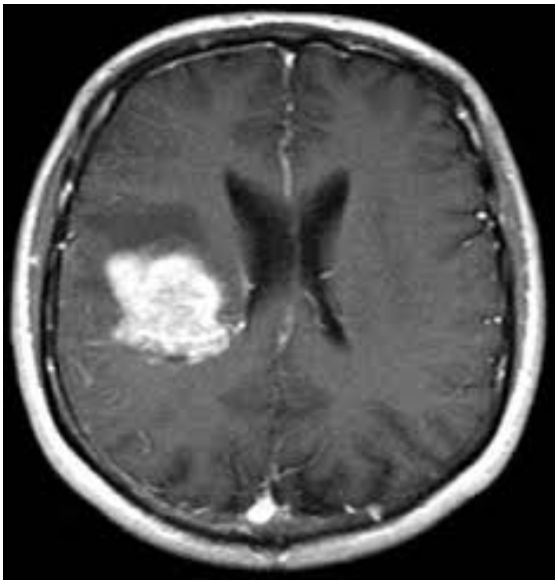
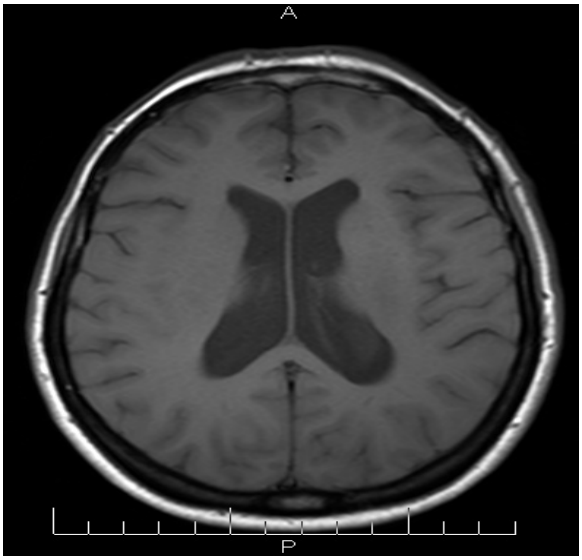


Figure 3 – MRI Scan of A Healthy Brain



The potential implications and applications of these MRI scans are enormous. Firstly, by detecting the tumors in these scans, doctors are able to diagnose people with brain tumors. However, more importantly, for those already diagnosed, the ability to track the growth/reduction in size of the tumors is important in prescribing treatment and in analyzing how effective treatment measures have been. That is, these images provide a gauge of the progress of an individual's condition. The ability to extract the tumors from these images is therefore important as it allows for a clear and easy comparison and analysis of the tumors over time. Using MATLAB, it is possible to use image-processing techniques to extract tumors from the images. They can then be used to analyze the growths by layering them to compare the way that these growths change. In this project, a method to extract these tumors from MRI scans was developed.

Method

In using image processing with MATLAB to analyze MRI scans, the following sequence of steps was implemented. Each of these steps is explained in greater detail in the 'Results and Steps' section, along with the execution results in MATLAB.

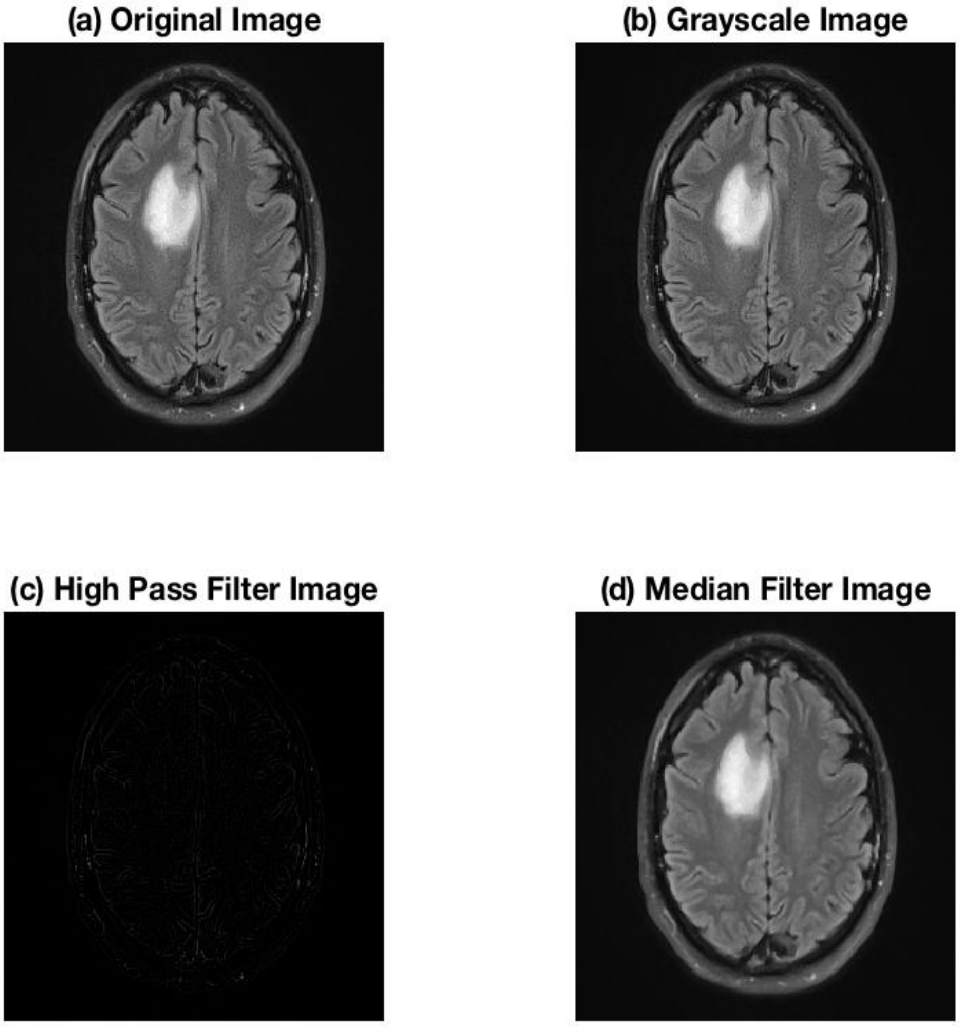
1. First, the MRI image was converted to a gray scale image.
2. Next, a high pass filter was applied to remove noise in the image.
3. A median filter was then used to enhance the image.
4. Following this, threshold segmentation while converting the image to a binary one helped extract the tumors from the rest of the image.
5. Watershed segmentation was implemented and compared to the binary image from the threshold segmentation.
6. Edge detection was performed on this image to get a clear conception of the boundary of the tumor.
7. Morphological operations that sharpened the image and eliminated noise were conducted that finally resulted in the extraction of the tumor from the image.

The exciting part about the application of the program is that the above methodology works for any MRI scan, and pictures of these scans can be obtained through a simple Google search for images corresponding to 'brain tumor MRI scans'.

Results and Steps

All of these steps are illustrated first with the image from figure 1, following which results for the other two images are displayed as well.

Figure 4 – Gray scale and Filtering



Step 1 – Gray scale Imaging

Using the function rgb2gray, it is possible to convert the image to gray scale. A gray scale image is preferred as it is easier to process. By creating a grid of black dots over a white background (or vice-versa), the size of each of the dots gives us the intensity/lightness of the gray in that area and therefore makes for an easy comparison of regions in the photo that are of 'brighter' and 'darker' shades. In terms of how the computer performs this process, every colored image has red, green, and blue components, and these components determine the color of an individual pixel. Each component is a number between 0 and 255 (or in binary between 00000000 and 11111111). In an RGB gray scale image, for every pixel, the values of these components are converted using the equation $x = 0.299R + 0.587G + 0.114B$ that gives us a weighted average of these values given R=G=B. This gives us an easy way to represent the 'brightness' of

each pixel. We can gleam the lightness of each pixel from the gray scale image as a value between R=G=B = 0 or 00000000 to R=G=B = 255 or 11111111. As these numbers increase, the shades of gray increase in brightness and a value of 0 corresponds to black, while a value of 255 represents white.

Code

The following code converts the image to gray scale and the results are seen in figure 4(b):

```
Im = imread('im1.jpg'); %Reads the image
figure(1);
subplot(2,2,1);
imshow(Im), title('(a) Original Image')

GrayIm = rgb2gray(Im); %converts the image to grayscale
subplot(2,2,2);
imshow(GrayIm),title('(b) Grayscale Image');
```

Step 2 – High Pass filter

A high pass filter involves transforming the image by a matrix and serves to sharpen the image. It does so by enhancing the contrast between areas in proximity to one another with little variation in brightness. The matrix used to do so is a kernel that is designed to retain high frequency information by increasing the brightness of the center pixel relative to pixels that surround it. This kernel matrix typically consists of a single positive value at the center surrounded by negative values all around it. For the purpose of the application, the following 3x3 matrix was used and the results can be seen in figure 4(c).

$$\text{Kernel} = \begin{matrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{8}{9} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{matrix}$$

While it is hard to make out the image in figure 4(c), on MATLAB it is clearer to see the outline that was created. Median filtering in the next step will make the purpose of this clearer.

Code

The following is the code for the high pass filter:

```
kernel = [-1/9 -1/9 -1/9;-1/9 8/9 -1/9;-1/9 -1/9 -1/9];
filteredImage = imfilter(GrayIm, kernel); %High pass filter eliminates noise
subplot(2,2,3);
imshow(filteredImage), title('(c) High Pass Filter Image');
```

Step 3 – Median Filter

Like the high pass filter, the median filter is a commonly used method of eliminating noise in images. However, it is used far more often because it can preserve the edges and boundaries of objects in the image while eliminating noise. It functions by running through the image matrix entry by entry and replacing each of the entries with the median of the values of the entries surrounding it. In doing so, the filter is able to enhance the image. The MATLAB function *medfilt2* conducts the median filtering process and the result, which is a clearer image with less noise than before, can be seen in figure 4(d).

Code

Code for median filtering:

```
MedianIm= medfilt2(filteredImage + GrayIm); %medfilt2 conducts median
filtering on the image
figure(1);
subplot(2,2,4);
imshow(MedianIm),title('(d) Median Filter Image');
```

Step 4 – Threshold Segmentation

Thresholding is a method whereby a gray scale image is converted to a binary image through image segmentation. In the process, a threshold level is determined and this fixed constant allows us to segment the pixels in the image. In converting the image to a binary image, we are essentially segmenting pixels based on certain shared characteristics that make them either black or white. However, while other image segmentation methods can group these pixels according to more characteristics, threshold segmentation basis it solely on whether or not the intensity of the pixel falls below or above the selected level. Therefore, while the method does segment the image based on 'shared characteristics', there is a huge range of values for which these characteristics are considered shared.

Threshold segmentation is more about creating an image where differences between two groups of pixels are exaggerated, rather than on creating multiple different segmentations. The procedure creates a binary image since any pixel with intensity lower than the chosen level is assigned a value of 0 (black), and any pixel with a value above the level is assigned a value of 1(white). (Intensity values lie between 0 and 1 and for this application the level selected was 0.6).

Traditionally, there are mathematical procedures used to determine this level. However, after obtaining unsatisfactory results using Otsu's method (that aims to minimize inter-class variance between two binary groups), through trial and error I found that a value of 0.6 was appropriate. For the final draft, it would be worthwhile to design and implement a more rigorous function that determines the appropriate level based on the input image as opposed to a pre-selected level.

The results of this process can be seen in figure 5(a) where we have come significantly closer to

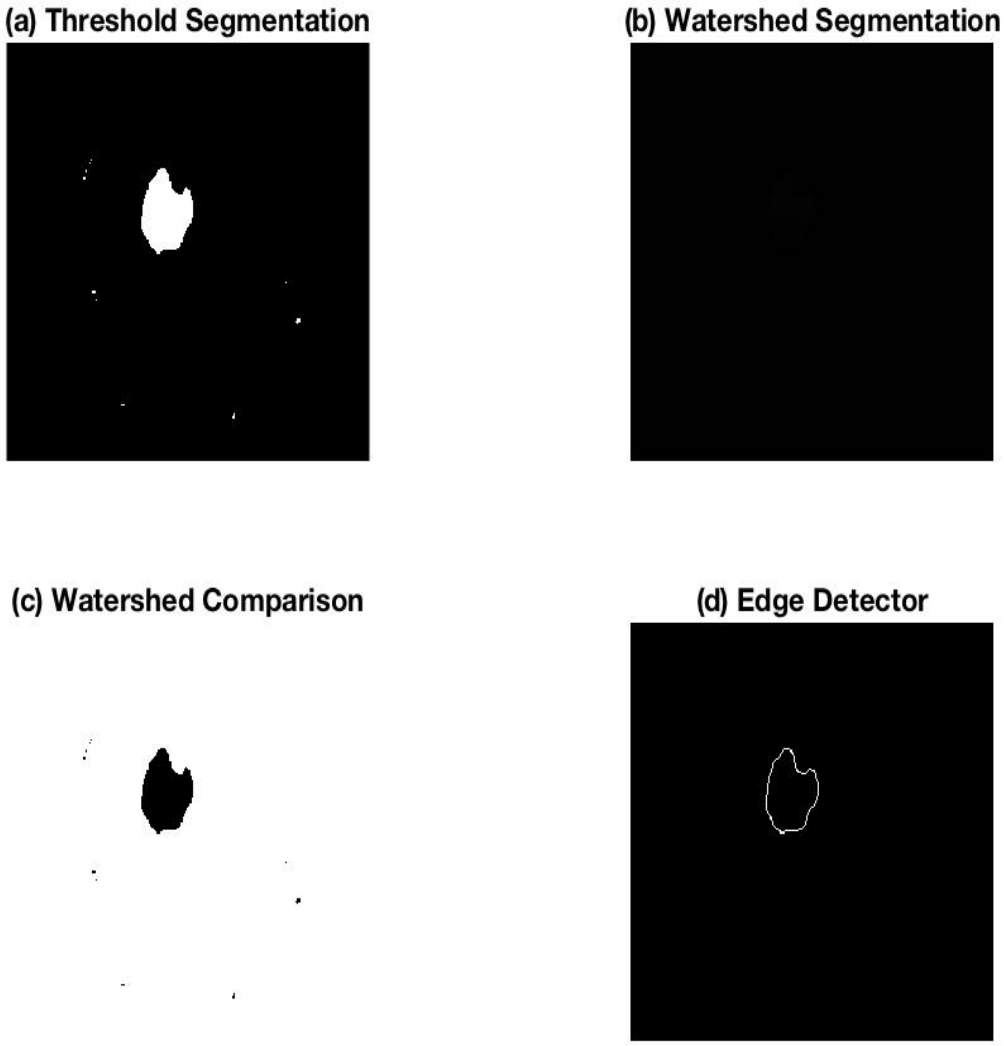
extracting the tumor from the image.

Code

```
level = graythresh(MedianIm); %Otsu's method to obtain a threshold value

BWIm = imbinarize(MedianIm, 0.6); %converts the image to black and white. In
place of 0.6, the value of level from the previous line is normally used.
figure(2);
subplot(2,2,1);
imshow(BWIm),title('(a) Threshold Segmentation');
```

Figure 5 – Segmentation and Edge Detection



Step 6 – Watershed Segmentation

Another image segmentation method that leads to cleaner results is the watershed segmentation. This is applied to the *BWIm* obtained after conducting threshold segmentation. To understand the watershed segmentation, think of the image as a surface. White areas are then considered to be high, while dark areas are considered to be low. These 'low' areas correspond to catchment basins that we want to identify, while the higher areas correspond to watershed lines (the terminology is confusing, but the idea remains simple). Intuitively, a drop of water falling on such a surface would flow down into the catchment basin to local minima, and these local minima are what we want to identify and segment in the image.

In computing the watershed transform, I first computed the distance transform of the image's complement, which gives the distance from every pixel to the nearest non-zero value pixel in the form of a matrix. Using the complement gives better results, but I had to negate the transform to turn bright areas into catchment basins. Then, applying the watershed function results in a matrix of positive integers corresponding to the locations of the catchment basins. Using the zero-valued elements in the matrix that are on the watershed lines, it is possible to separate the objects. These results are seen in figure 5(c).

Code

Below is the code for the watershed segmentation:

```
D = ~BWIm; %The following steps are part of computing the watershed
segmentation
E = -bwdist(D);
E(D) = -inf;
WaterIm = watershed(-D);
figure(2);
subplot(2,2,2);
imshow(WaterIm),title('(b) Watershed Segmentation');
F = BWIm;
F(WaterIm==0)=0;
subplot(2,2,3);
imshow(F),title('(c) Watershed Comparison');
```

Step 6 – Edge Detection

Simply put, edge detection is a method by which a new image, that emphasizes the edges and boundaries of objects in the original image, is obtained. This is clearly visible in figure 5(d) where the edge corresponds to the outer lining of the tumor. Edge detection makes it possible to accurately determine the size and shape of the tumor.

In implementing edge detection, there are several possible methods that can be used. For the purpose of my project, I chose the Sobel function that takes an image and a threshold value and returns an image with the detected edges. Since this is a fairly common image processing

technique, there already exists an inbuilt function in MATLAB to conduct the process and the code show below illustrates how this technique was used.

The Sobel operator works by using two 3x3 kernels that are convolved with the original image, following which approximations of the derivatives in the horizontal and vertical directions are calculated. These kernel filters are thereby estimating the gradient in the *x* and *y* directions and the magnitude of the gradient is obtained as the sum of squares of the two gradients as illustrated below.

$$Gx\ (x\ filter) = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad Gy\ (y\ filter) = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

$$G = \sqrt{(Gx)^2 + (Gy)^2}$$

Using these values of the gradient and computing its direction using an *arctan* function, the edge of the image is obtained as illustrated in figure 5(d).

Code

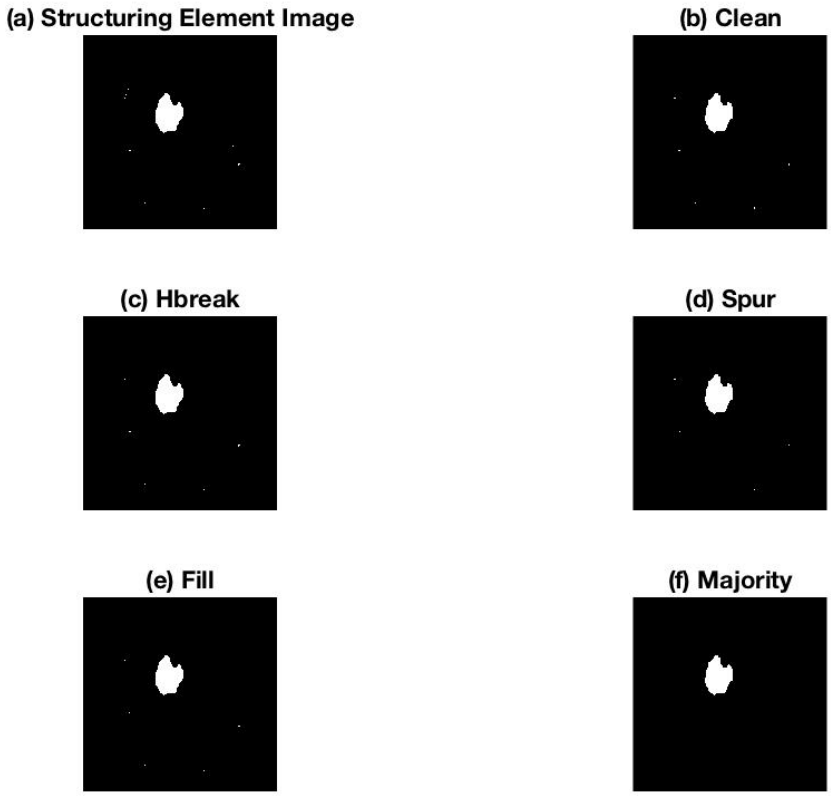
```
EdgeIm = edge(MorphIm4, 'sobel'); %Edge detection using Sobel's method
figure(2)
subplot(2,2,4);
imshow(EdgeIm),title('(d) Edge Detector');
```

Step 7 – Morphological Operations

Morphological operations on binary images involve using a structuring element (a shape) that is positioned at all possible points in the picture and is compared with the corresponding neighborhood of pixels. If it is determined to 'fit', then it creates a new binary value in that position/pixel which sharpens the image. For the purpose of my project, all morphological operations were conducted on the binary image resulting from the threshold segmentation. It is, however, possible to conduct these operations on gray scale images as well.

For part (a), the 'structuring disk' was used to conduct the morphological operation. In part (b), the operation 'clean' removes isolated pixels that are individual 1's surrounded by zeros. The 'hbreak' operation in part (c) removed H-connected pixels so the 1 in the middle row becomes a zero. In part (d), the 'spur' operator removes spur pixels. The 'fill' operator in part (e) fills isolated interior pixels (these are 0's that are surrounded by 1's). While the 'majority' operator in part (f) sets a pixel to 1 if five or more pixels in its 3x3 neighborhood are 1's. If not, it sets the pixel to 0.

Figure 6 – Morphological Operations



Code

The following is the code for the various morphological operations:

```
%Morphological Operations
StrucIm = imtophat(BWIm, strel('disk', 30));
figure(3)
subplot(3,2,1);
imshow(StrucIm),title('(a) Structuring Element Image');

MorphIm = bwmorph(StrucIm, 'clean');
subplot(3,2,2);
imshow(MorphIm), title('(b) Clean');

MorphIm1 = bwmorph(MorphIm, 'hbreak'); %majority, fill, hbreak
subplot(3,2,3);
imshow(MorphIm1), title('(c) Hbreak');
```

```
MorphIm2 = bwmorph(MorphIm1,'spur');
subplot(3,2,4);
imshow(MorphIm2), title('(d) Spur');

MorphIm3 = bwmorph(MorphIm2, 'fill');
subplot(3,2,5);
imshow(MorphIm3), title('(e) Fill');

MorphIm4 = bwmorph(MorphIm3, 'majority');
subplot(3,2,6);
imshow(MorphIm4), title('(f) Majority');
```

Discussion

Figure 7 - Final comparison

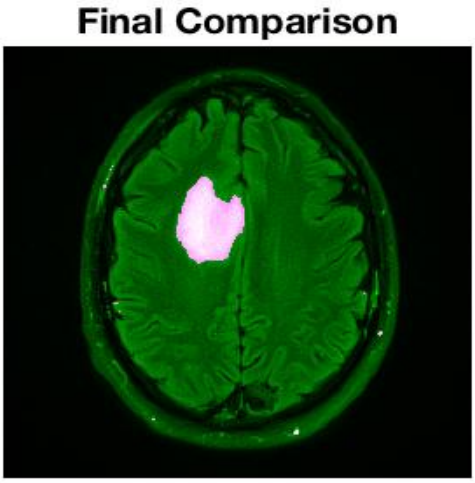


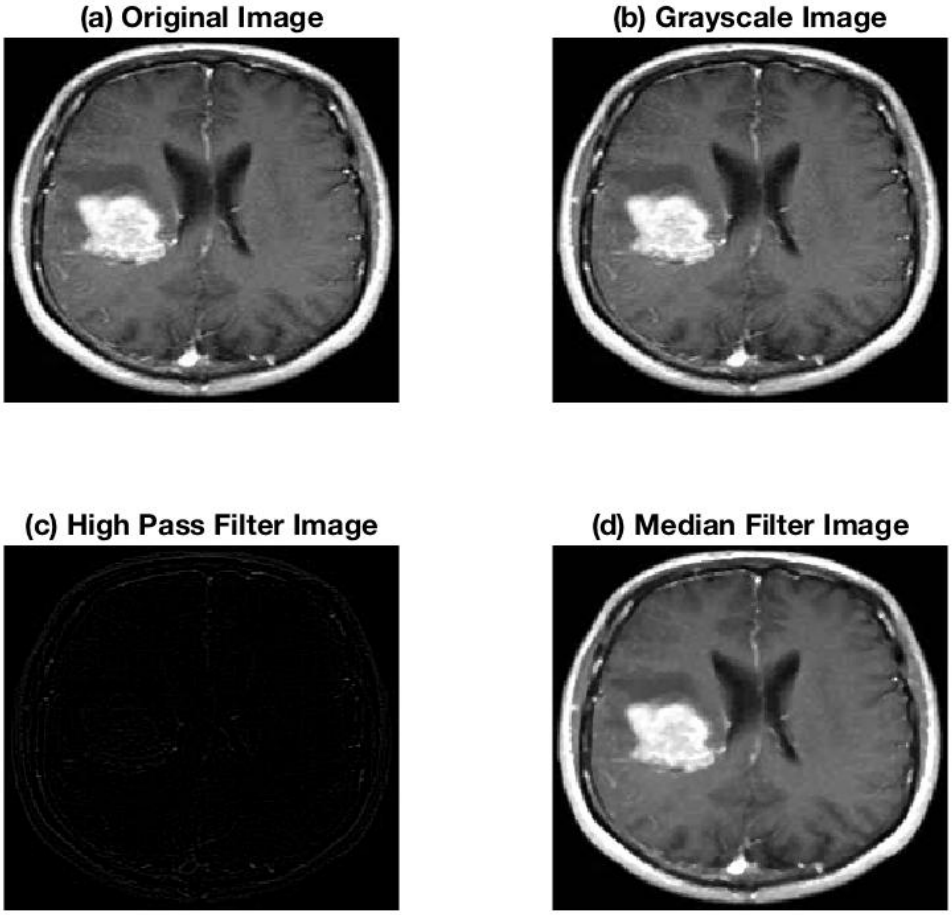
Figure 7 provides an overlay of the final extracted image (after the morphological operations removed the isolated entries of the threshold segmentation) on to the original image. Using the function *imfuse*, it is clear to see the extracted tumor, which is highlighted and presented in a different color to the rest of the image. The following code resulted in this comparison:

```
figure(4);
FinalIm = imfuse(GrayIm,BWIm);
imshow(FinalIm), title('Final Comparison'); %Compares the tumor we extracted
through segmentation by laying it over the image
```

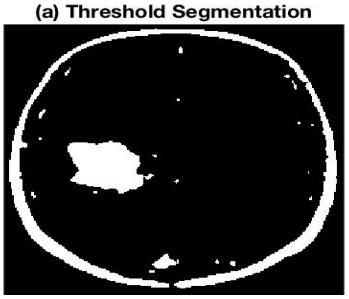
Through this result it is observable that along with extracting the tumor from the rest of the image, using the methodical procedures outlined previously, a lot of the noise in the image was removed as well.

Testing the ability of the program on other images would highlight its ability in extracting tumors, and so the outputs for figures 2 and 3 from the introduction were computed as well.

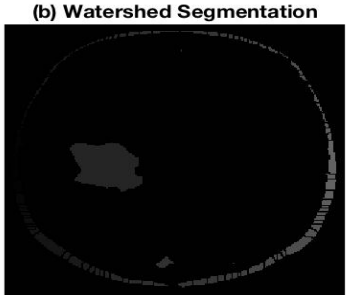
Figures 8,9,10,11 – Outputs for the second image



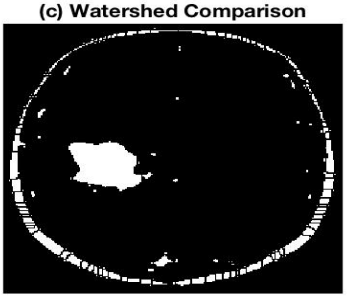
(a) Threshold Segmentation



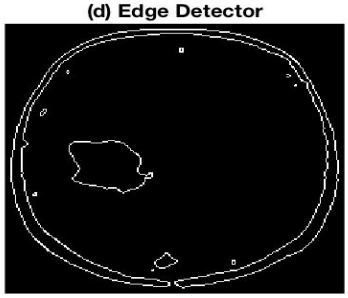
(b) Watershed Segmentation



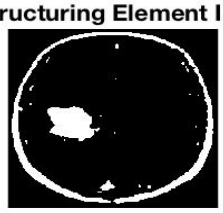
(c) Watershed Comparison



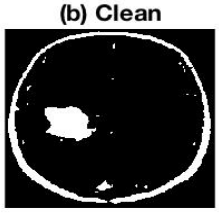
(d) Edge Detector



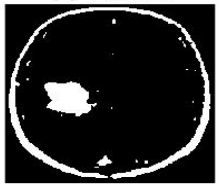
(a) Structuring Element Image



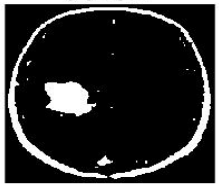
(b) Clean



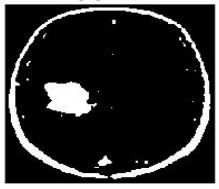
(c) Hbreak



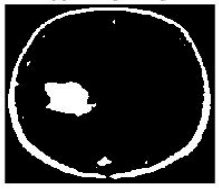
(d) Spur

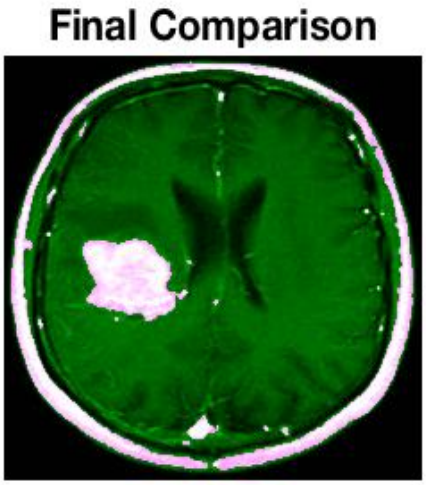


(e) Fill



(f) Majority

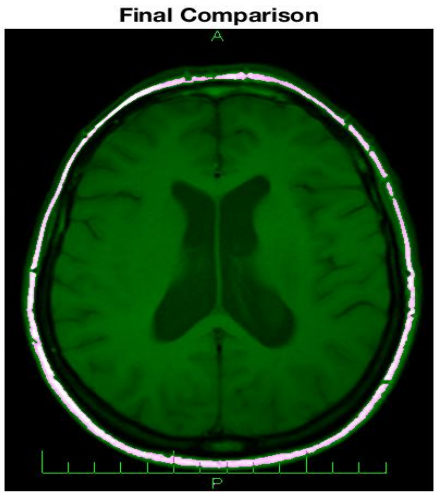




Through all of the above pictures and the final comparison image, it is clear that the code works effectively for other images. However, unlike in the previous image, the MRI of the brain in figure 2 had a very bright outer ring corresponding to the skull that was highlighted in the process. Despite this, the tumor was still extracted effectively and the image obtained from the code can be used for the practical applications of comparing the size of the tumor over time.

Finally, to observe that the code is effective in picking up only tumors and other white regions that may be highlighted in the MRI, the program was run on an image of a healthy brain and the final output is illustrated below.

Figure 12 – Final comparison for healthy brain



From the image, we see that no region other than the outer ring was highlighted and we therefore do not obtain any 'false regions' corresponding to tumors when the application is run on a healthy brain. (The rest of the output is not necessary to analyze since none of the steps outlined in the method would have led to the segmentation and extraction of anything within the ring). The results are encouraging as it is evident that the program effectively extracts regions of the brain that correspond to tumors.

Future Work

For the final draft, I'll explain the morphological operations in greater depth using matrices as examples. Additionally, I ran into trouble with the image quality upon moving it from MATLAB to word that I was trying to rectify. Finally, I could implement a zooming function (through a method such as bilinear interpolation) so that the outer ring of the skull from the MRI isn't highlighted. However, either way, when there is a tumor in the image, it is still extracted efficiently.

NAME: VARUN S RAJAGOPAL
STUDENT ID: 73342019

Appendix (Code In Full)

```
Im = imread('im1.jpg'); %Reads the image

GrayIm = rgb2gray(Im); %converts the image to grayscale
figure(1);
subplot(2,2,1);
imshow(GrayIm),title('Grayscale Image');

kernel = [-1/9 -1/9 -1/9;-1/9 8/9 -1/9;-1/9 -1/9 -1/9];
filteredImage = imfilter(GrayIm, kernel); %High pass filter
eliminates noise
subplot(2,2,2);
imshow(filteredImage), title('High Pass Filter Image');

MedianIm= medfilt2(filteredImage + GrayIm); %medfilt2 conducts
median filtering on the image
figure(1);
subplot(2,2,3);
imshow(MedianIm),title('Median Filter Image');

level = graythresh(MedianIm); %Otsu's method to obtain a
threshold value
BWIm = imbinarize(MedianIm, 0.6); %converts the image to black
and white
figure(2);
subplot(2,2,1);
imshow(BWIm),title('Threshold Segmentation');

D = ~BWIm; %The following steps are part of computing the
watershed segmentation
E = -bwdist(D);
E(D) = -inf;
WaterIm = watershed(E);
figure(2);
```

```
subplot(2,2,2);
imshow(WaterIm),title('Watershed Segmentation');
F = D; %or BWIm
F(WaterIm==0)=0;
subplot(2,2,3);
imshow(F),title('Watershed Comparison');

%Morphological Operations
StrucIm = imtophat(BWIm, strel('disk', 30));
figure(3)
subplot(3,2,1);
imshow(StrucIm),title('(a) Structuring Element Image');

MorphIm = bwmorph(StrucIm, 'clean');
subplot(3,2,2);
imshow(MorphIm), title('(b) Clean');

MorphIm1 = bwmorph(MorphIm, 'hbreak'); %majority, fill, hbreak
subplot(3,2,3);
imshow(MorphIm1), title('(c) Hbreak');

MorphIm2 = bwmorph(MorphIm1,'spur');
subplot(3,2,4);
imshow(MorphIm2), title('(d) Spur');

MorphIm3 = bwmorph(MorphIm2, 'fill');
subplot(3,2,5);
imshow(MorphIm3), title('(e) Fill');

MorphIm4 = bwmorph(MorphIm3, 'majority');
subplot(3,2,6);
imshow(MorphIm4), title('(f) majority');

EdgeIm = edge(MorphIm4, 'sobel'); %Edge detection using Sobel's
method %WaterIm %M
figure(2)
subplot(2,2,4);
imshow(EdgeIm),title('Edge Detector');

figure(4);
FinalIm = imfuse(GrayIm,BWIm);
imshow(FinalIm); %Compares the tumor we extracted through
```

```
segmentation by laying it over the image
```