Math 320 Research Project

Topic: Extraction of Brain Tumors from MRI scans

Name: Varun S Rajagopal

Student ID: 73342019

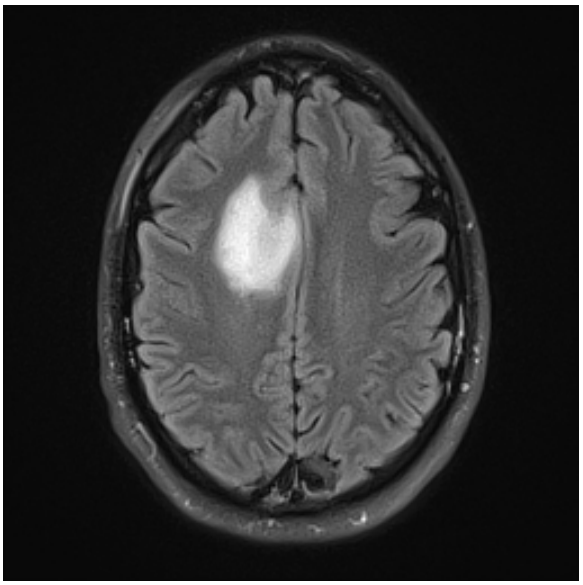
Date: December 14th, 2016

MATH 320 Fall 2016

Introduction

Brain tumors refer to the uncontrolled growth of cells in tissues of the brain that can ultimately prove life-threatening. As a result, the ability to detect such tumors is of immense importance in reducing patient suffering and saving lives. Developments in technology, through MRI (magnetic resonance imaging) scans have helped us detect these growths at an early stage. MRI scans produce high quality images that display the tumors as white regions in the brain. This is illustrated in figures 1 and 2 below. By contrast, figure 3 shows an MRI of a healthy brain without any white regions.

Figures 1 & 2 – Brain Tumor MRI scans



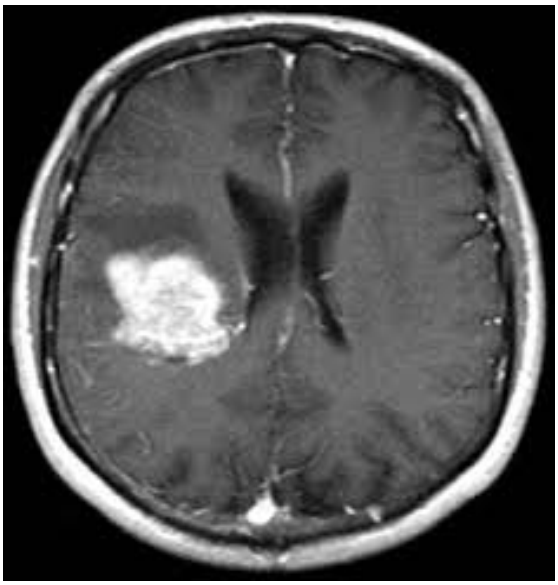
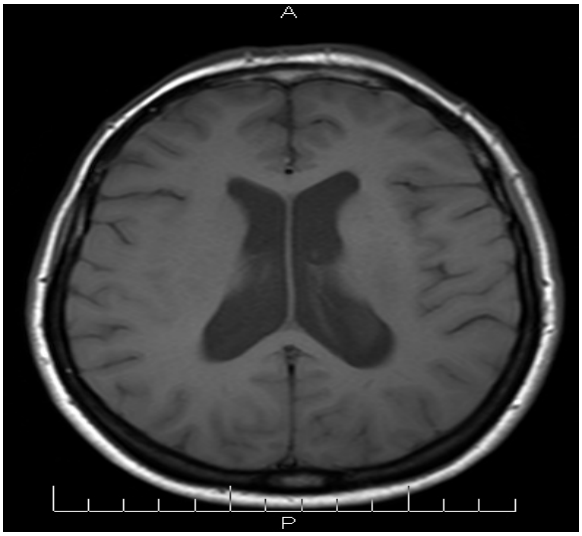


Figure 3 – MRI Scan of A Healthy Brain



The potential implications and applications of MRI scans are enormous. Firstly, by detecting the tumors in these scans, doctors are able to diagnose and prescribe treatments for people with brain tumors. For those already diagnosed, doctors can use MRI scans to track the progress of an individual's condition and to analyze the effectiveness of treatment measures. A growth in the tumor size corresponds to deteriorating health, while a reduction represents improvement. The ability to extract the tumors from these images is therefore important as it allows for a clear, easy comparison and analysis of the tumors over time. By layering the images, they can be used to compare the way the tumors change. Using MATLAB, I improved, explained and explored a method[1] to extract these tumors from MRI scans through image-processing techniques.

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 enhance edges in the image.
3. A median filter was then used to remove noise in the image.
4. Following this, threshold segmentation and converting the image to binary helped distinguish the tumors from the rest of the image.
5. (Optional) Watershed segmentation was applied and compared to the binary image from the threshold segmentation.
6. Morphological operations that sharpened the image and eliminated noise were performed to facilitate the extraction of the tumor from the image.
7. Edge detection was then implemented to get a clear conception of the boundary of the tumor.

All images can be represented as a matrix of pixels. As such, most image processing techniques involve operations on the image matrix. These operations have been explained using 3 by 3 matrices centered on a particular pixel (that is to say, we select a pixel and then convert it and its neighboring pixels into a 3 by 3 matrix). By going through the image and performing calculations, the value of our chosen pixel (the center pixel at position [2,2]) is updated and we then continue till the matrix transformations have been applied to all 3 by 3 neighborhoods in the image matrix, and all the pixel values have been updated.

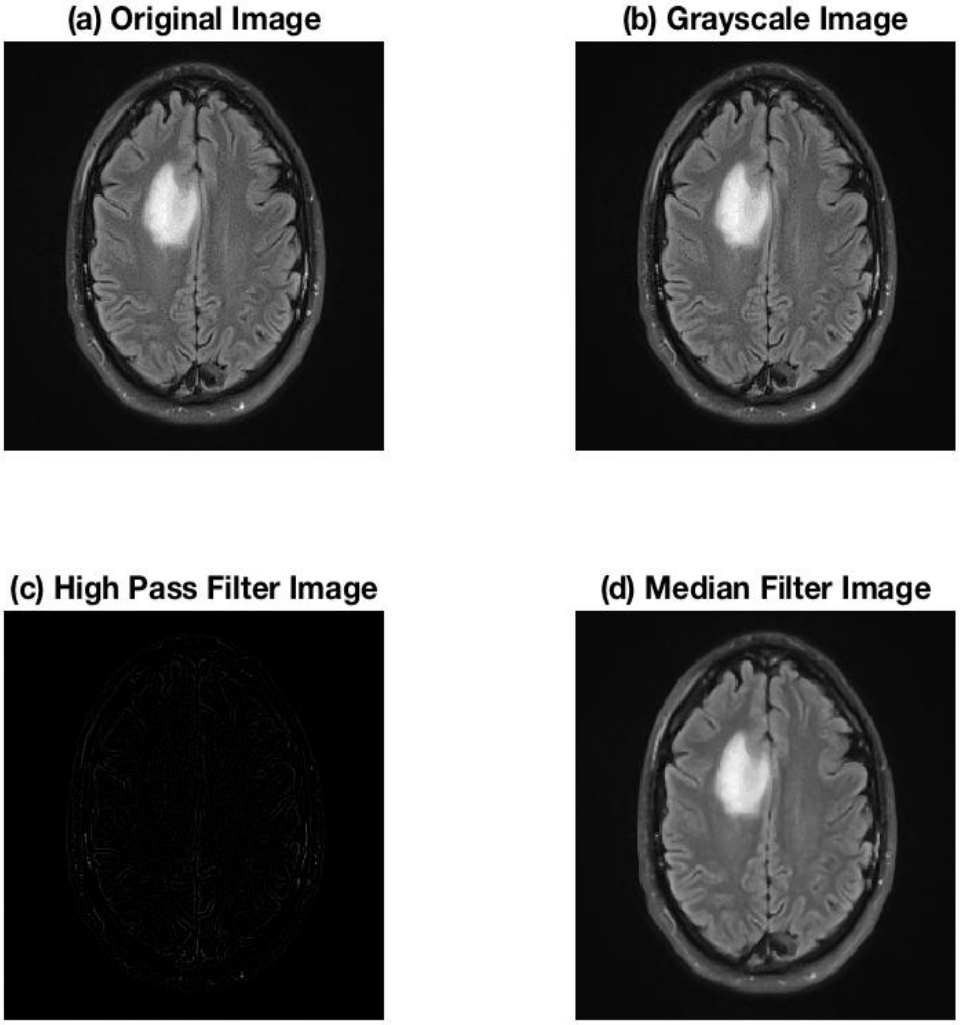
The exciting part about the application of the program is that the above methodology works for any MRI scan, and pictures of these scans can be obtained through a simple Google search for images corresponding to 'brain tumor MRI scans'.

[1] Patil, Rajesh C., and A.S. Balchandra. "Brain Tumour Extraction from MRI Images." *IJECSCSE*, vol. 2, no. 1,

Results and Steps

All of these steps are first illustrated 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 – Grayscale Imaging

Using the function `rgb2gray`, the image was converted to gray scale. A gray scale image is preferred in image processing, as less information needs to be provided for each pixel. This reduces complexity and makes the image easier to process since all the colors are shades of gray. 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. In a gray

scale image, the red, green and blue components are all assigned equal values ($R = G = B$). The value they are given is determined such that it fulfills the equation $Intensity = 0.299R + 0.587G + 0.114B$ that calculates a weighted average intensity for the original color pixel.

By converting to grayscale, it becomes sufficient to specify a single intensity value for each pixel as opposed to the three intensity values required for a pixel in a color image. The grayscale intensity value is stored as an 8-bit integer and each component is a number between 0 and 255 (or in binary between 00000000 and 11111111)[2]. In this scale, a value of 0 corresponds to black, a value of 255 represents white, and all the numbers in between correspond to different levels of 'brightness' (intensity). 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 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.

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 is used to enhance borders in the image and is helpful in edge detection as it retains high frequency information while reducing low frequency information. It does so by convolving the image matrix by a kernel matrix that typically consists of a single positive value at the center surrounded by negative values all around it[3]. For the purpose of the application, the following 3x3 matrix was used and the results can be seen in figure 4(c).

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

[2] "What Is Grayscale?" TechTarget, whatis.techtarget.com/definition/grayscale.

[3] "Filtering an Image." *Filtering an Image*, IDL, 16 June 2005, northstar-www.dartmouth.edu/doc/idl/html_6.2/Filtering_an_Imagehvr.html.

To conduct the convolution[4], we first take a 3 by 3 matrix centered on pixel e in the image. Then, the kernel matrix is rotated by 180 degrees (which is the same as flipping all the rows and columns). In the case of the kernel chosen, a matrix rotation of 180 degrees leads to the same matrix since the center term stays where it is and all the surrounding values are equal. The rotated kernel and the image matrix are then multiplied as follows:

$$\text{Convolution} = \begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix} * \begin{matrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{8}{9} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{matrix}$$

The multiplication in a convolution is not the same as ordinary matrix multiplication. Instead, each term in the first matrix is multiplied with the corresponding term from the second matrix and the values are then summed. This leads to the following scalar value:

$$new\ e = \left(-\frac{1}{9} * a\right) + \left(-\frac{1}{9} * b\right) + \left(-\frac{1}{9} * c\right) + \left(-\frac{1}{9} * d\right) + \left(\frac{8}{9} * e\right) + \left(-\frac{1}{9} * f\right) + \left(-\frac{1}{9} * g\right) + \left(-\frac{1}{9} * h\right) + \left(-\frac{1}{9} * i\right)$$

The initial pixel chosen as the center of the 3 by 3 matrix was pixel e. Therefore, the original value of pixel e is replaced with the new e value calculated in the convolution above.

In high pass filtering, if the center pixel is of a higher intensity than the surrounding pixels, the matrix convolution emphasizes that value and therefore increases the intensity of the pixel. This is because the center pixel is multiplied by a factor of 8× what the surrounding values are multiplied by. So a pixel with a significantly higher intensity than the neighboring pixels appears white, while a pixel of similar intensity to its neighbors appears black since the convolution would lead to a value close to 0.

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');
```

[4] "Convolution." Cornell, 27 Feb. 2013, www.cs.cornell.edu/courses/cs1114/2013sp/sections/s06_convolution.pdf.

Step 3 – Median Filter

The median filter is a commonly used method of eliminating noise in images and is used because it can preserve the edges and boundaries of objects in the image. It functions by running through the image matrix entry by entry and taking the neighboring 3 by 3 matrix around each pixel. The values of the matrix are then sorted in ascending order and the median value at the center pixel position (2,2) replaces the original pixel value.[5] The following matrices illustrate the process:

$$\begin{matrix} 0 & 3 & 5 \\ 17 & 2 & 6 \\ 1 & 4 & 7 \end{matrix} \quad after\ sorting\ becomes \quad \begin{matrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 17 \end{matrix}$$

In the above sorting, the initial pixel we chose had a value of 2. After sorting the matrix, the median value in the center of the second matrix resulted in a value of 4. As a result, we replace the 2 by the 4 in the new image matrix. The following code illustrates how the median filtering process is done on MATLAB:

Median Filtering Function

```
%Add zeros to side of the matrix
modifyGrayIm=zeros(size(GrayIm)+2);
MedianIm=zeros(size(GrayIm));

%copy the original matrix onto the new matrix with zeros on the edges
        for x=1:size(GrayIm,1)
            for y=1:size(GrayIm,2)
                modifyGrayIm(x+1,y+1)=GrayIm(x,y);
            end
        end

      %Create an array of values for the 3 by 3 matrix

for i= 1:size(modifyGrayIm,1)-2
    for j=1:size(modifyGrayIm,2)-2
        arrayform=zeros(9,1);
        value=1;
        for x=1:3
            for y=1:3
                arrayform(value)=modifyGrayIm(i+x-1,j+y-1);
                value=value+1;
            end
        end

       %Sort the array to find the median element
        median=sort(arrayform);
       %Places median element in the median filtered matrix
        MedianIm(i,j)=median(5);
```

[5] Angel, Aaron. "MATLAB PROGRAM : 2D MEDIAN FILTERING FOR SALT AND PEPPER NOISE WITHOUT USING medfilt2 FUNCTION." angeljohnsy.blogspot.com/2011/03/2d-median-filtering-for-salt-and-pepper.html.

```
        end
    end
    MedianIm=uint8(MedianIm);
```

The idea behind the pseudocode is simple. We first pad our matrix with zeros so we can obtain median values at the boundary of our original image as well. We then go through this matrix pixel by pixel using for loops, and for each 3 by 3 neighborhood of the chosen pixel we create an array of the matrix values. The array is then sorted using the *sort* function in ascending order and the median value of the array replaces the pixel in the original image.

By replacing each of the entries with the median of the values of the entries surrounding it, the filter is able to remove noise since values that are large outliers get eliminated in the new image. The MATLAB function *medfilt2* conducts the median filtering process and the result, which is an image with less noise than before (though less sharp due to the removal of large and small values), 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[6]

Thresholding is a method whereby a grayscale 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

[6] "Binarize Image by Thresholding - MATLAB Imbinarize." *Binarize Image by Thresholding - MATLAB Imbinarize*, Mathworks, www.mathworks.com/help/images/ref/imbinarize.html.

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[7] (that aims to minimize in-class variance between two binary groups), through trial and error I initially found that a value of 0.6 was appropriate. After returning to the problem and trying to implement further changes by throwing away the black pixels, I was unable to design a function that determined a good thresholding level based on a given input image.

The results of threshold segmenting can be seen in figure 5(a) where we have come significantly closer to extracting the tumor from the image. Aside from the tumor region (that we desire) and a few white particles that are sporadically spread through the image, most of the healthy regions of the brain have been converted to black.

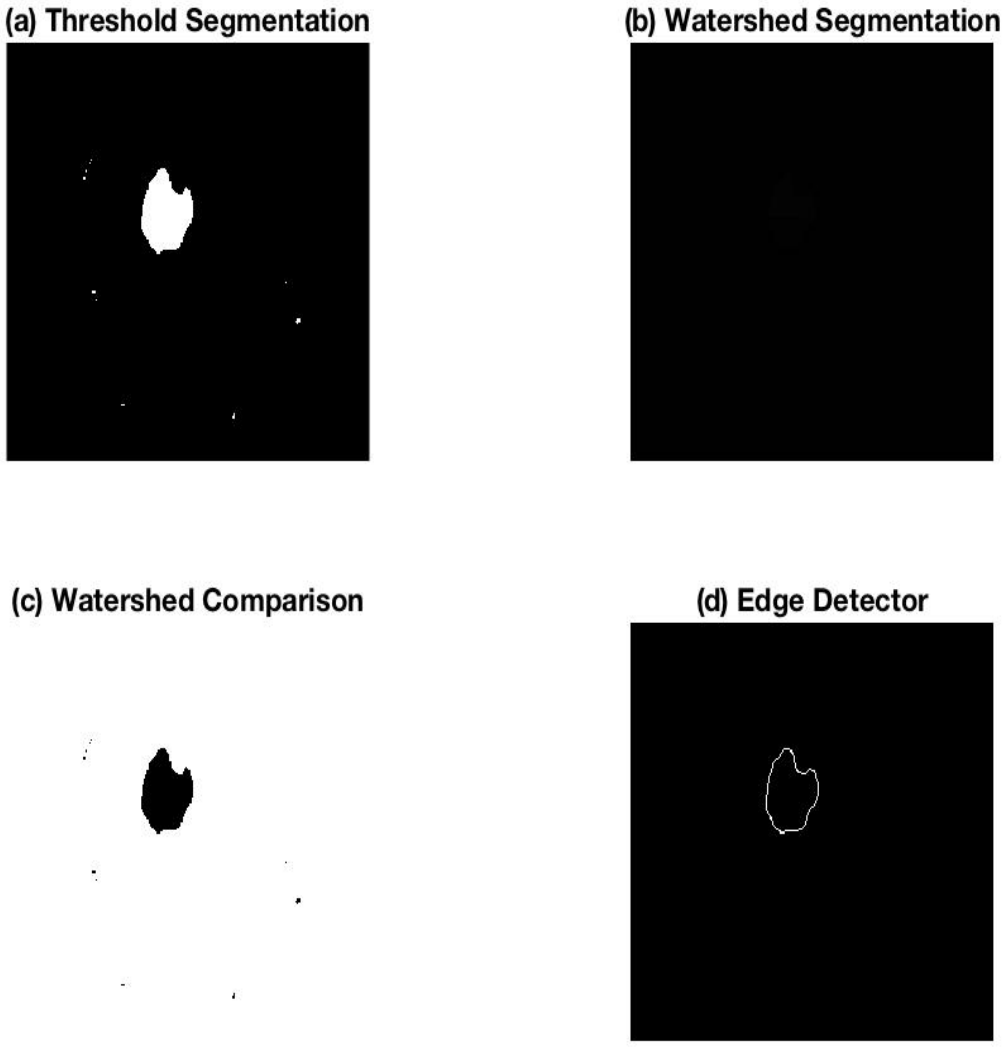
Code

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

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');
```

[7] "Global Image Threshold Using Otsu's Method - MATLAB Graythresh." Mathworks, www.mathworks.com/help/images/ref/graythresh.html.

Figure 5 – Segmentation and Edge Detection



Step 5 (Optional) – Watershed Segmentation

Another image segmentation method is the watershed segmentation[8]. This is applied to the `BWIm` obtained after conducting threshold segmentation. To understand the watershed segmentation,

[8] Eddins, Steve. "The Watershed Transform: Strategies for Image Segmentation." Mathworks, 2002, www.mathworks.com/company/newsletters/articles/the-watershed-transform-strategies-for-image-segmentation.html.

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. Essentially, this provides a greater check on what pixels we label white, improving the threshold segmentation that was done earlier. 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 figures 5(b) (it is difficult to see the shape unless you look closely) and 5(c).

This step is optional and offers an alternative that could provide better results as it identifies fewer white areas in the image. This is seen in the watershed transform of figure 9(c) that creates a better segmentation than the threshold transformation. This is because the entirety of the white outer ring in the threshold-segmented image isn't identified as a catchment basin in the watershed segmentation. In comparison, in the threshold segmentation they are assigned values of 1 just for being over the intensity level that was specified. However, a potential drawback of the image is that for the same reason that the white outer ring wasn't identified completely, in some cases the entire tumor may not be identified either. By comparing the two images, an individual can determine which would offer a better final result.

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 – 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.[9] It is, however, possible to conduct these operations on grayscale images as well.

For part (a), the '`structuring disk`' was used to conduct the morphological operation. In this process, a morphological opening is computed and then subtracted from the original binary matrix. To compute the morphological opening, we first use an erosion and then perform a dilation on the eroded matrix[10]. For erosion, the structuring disk is placed on a pixel and if there are any 0's in the region encompassed by the disk, we assign a value of 0 to the pixel the disk was placed on. Dilation then assigns a 1 to all the pixels for which the structuring disk region contains a 1. This results in the morphological opening that is then subtracted from the original binary matrix resulting in figure 6(a).

In part (b), the operation '`clean`' removes isolated pixels that are individual 1's surrounded by zeros. For example,

$$\begin{matrix}0&0&0\\0&1&0\\0&0&0\end{matrix} \quad \textit{becomes} \quad \begin{matrix}0&0&0\\0&0&0\\0&0&0\end{matrix}$$

The '`hbreak`' operation in part (c) removes H-connected pixels so a 1 in the (2,2) position becomes a zero. Therefore,

$$\begin{matrix}1&1&1\\0&1&0\\1&1&1\end{matrix} \quad \textit{becomes} \quad \begin{matrix}1&1&1\\0&0&0\\1&1&1\end{matrix}$$

In part (d), the '`spur`' operator removes spur pixels. In practice,

$$\begin{matrix}0&0&1\\0&1&0\\1&1&0\end{matrix} \quad \textit{becomes} \quad \begin{matrix}0&0&0\\0&1&0\\1&1&0\end{matrix}$$

[9] "*Morphological Operations on Binary Images - MATLAB Bwmorph*," Mathworks, www.mathworks.com/help/images/ref/bwmorph.html.

[10] "Morphological Image Processing." www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm.

The '`fill`' operator in part (e) fills isolated interior pixels (these are 0's that are surrounded by 1's). Unlike the other operations that aim to reduce noise, this makes the tumor region more pronounced. For example,

$$\begin{matrix}1 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 1\end{matrix} \quad becomes \quad \begin{matrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{matrix}$$

The '`majority`' operator in part (f) sets a pixel to 1 if five or more pixels in its 3 by 3 neighborhood are 1's. If not, it sets the pixel to 0. Therefore,

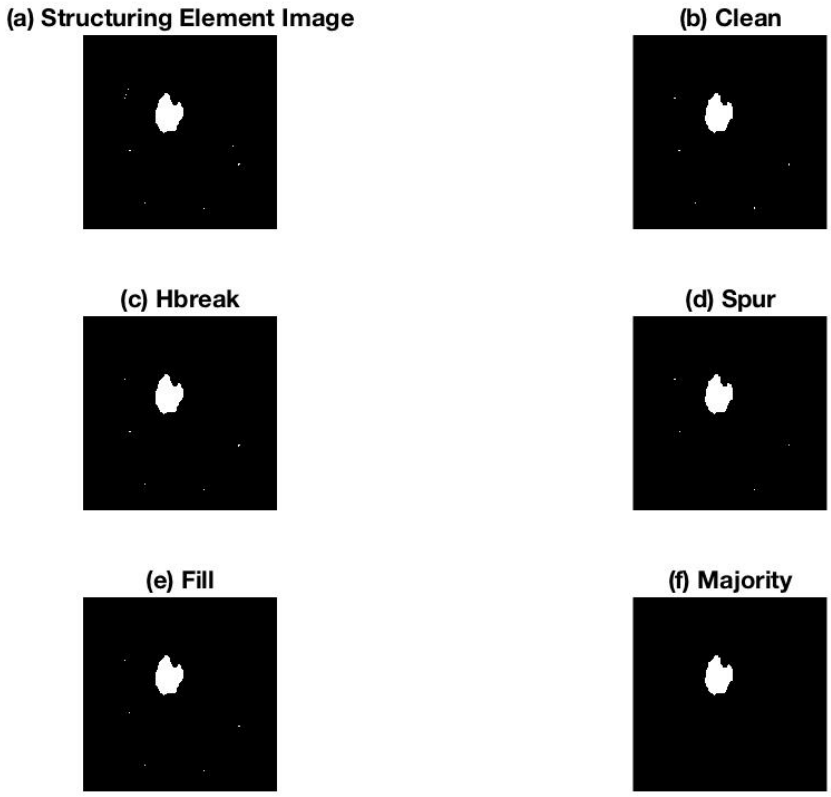
$$\begin{matrix}1 & 1 & 1\\ 1 & 0 & 0\\ 1 & 1 & 0\end{matrix} \quad becomes \quad \begin{matrix}1 & 1 & 1\\ 1 & 1 & 0\\ 1 & 1 & 0\end{matrix}$$

While

$$\begin{matrix}1 & 0 & 0\\ 1 & 1 & 0\\ 1 & 0 & 0\end{matrix} \quad becomes \quad \begin{matrix}1 & 0 & 0\\ 1 & 0 & 0\\ 1 & 0 & 0\end{matrix}$$

From Figure 6, it is clear to see that most of the noise reduction occurred in 6(a) and 6(f). These images correspond to the '`structuring disk`' and '`majority`' operators respectively. The reason that they were the most effective is because they create changes more often than the other operators do. The other operators require more specific 3 by 3 orderings of the pixel matrix, which from a simple probabilistic standpoint, occurs infrequently. On the other hand, the two other operators were more general and did not require a specific ordering of the pixels in the matrix. They also change the image matrix more frequently than the other operators as they have outcomes for failure of the condition they are testing as well. For instance, in the '`majority`' operator, if the condition that more than five 1's are present isn't satisfied, the operator does not leave the center pixel unchanged, but instead gives it a value of zero.

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');
```

Step 7 – 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 shown below illustrates how this was performed.

The Sobel operator[11] works by taking two 3x3 kernels that are convolved with the original image to approximate the derivatives in the horizontal and vertical directions. The convolution process occurs in the same way that convolution took place in the 'High Pass Filtering' section. These kernel filters are thereby estimating the gradient in the x and y directions, and the magnitude of the gradient is the sum of squares of the two gradients as illustrated below.

$$x\ filter = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} \qquad Rotated\ x\ filter = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

$$y\ filter = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \qquad Rotated\ y\ filter = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

[11] Fisher, R et al. "Sobel Edge Detector." *Feature Detectors - Sobel Edge Detector*, 2003, homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm.

NAME: VARUN S RAJAGOPAL
STUDENT ID: 73342019

$$Image\ matrix = \begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix}$$

$$Gx = Rotated\ x\ filter * I = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} * \begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix}$$

$$Gy = Rotated\ y\ filter * I = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} * \begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix}$$

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

As illustrated above, the convolution leads to scalar values for Gx and Gy from which the value of G is calculated. The sobel operator then takes the values of G that lie above the sobel operator's threshold value (this amounts to only a few large values of G), and creates a white boundary with those points as illustrated in figure 5(d). The white regions correspond to points where the value of G was greater than the threshold sobel value.

Code

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

Discussion

Figure 7 - Final Comparison

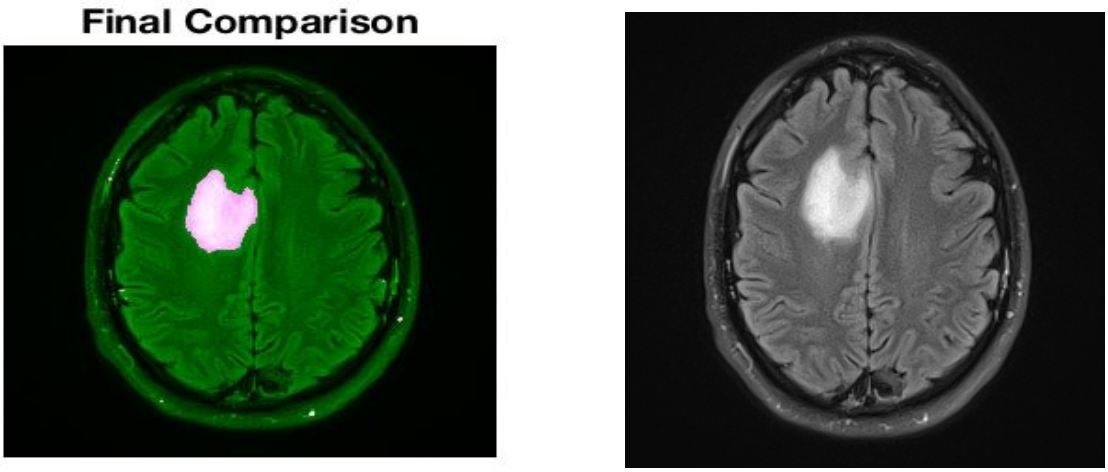


Figure 7 provides an overlay of the final extracted image on to the original image. Using the function ***imfuse***, we can very clearly see the extracted tumor, which is highlighted and presented in a different color to the rest of the image. The tumor region is now significantly better defined in comparison to the original 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.

To determine whether or not the program I implemented is well constructed, testing the ability of the program on other images is important in highlighting how well each part of the program fulfills the steps it is intended to. To do this comparison, I computed the outputs for figures 2 and 3 from the introduction as well.

Figure 8 – Output for the Second Image

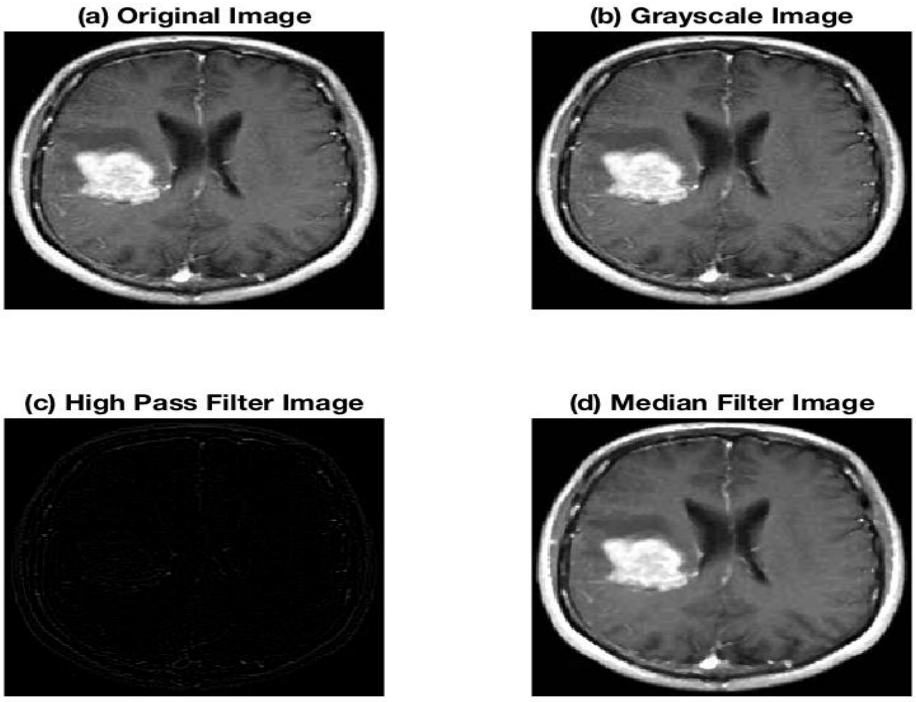


Figure 9 – Output for the Second Image

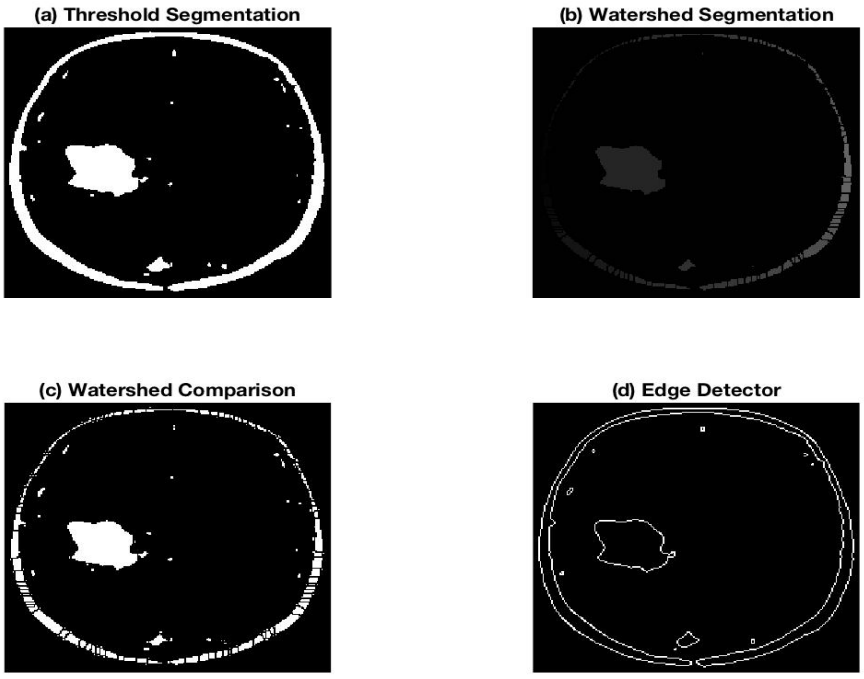


Figure 10 – Output for the Second Image

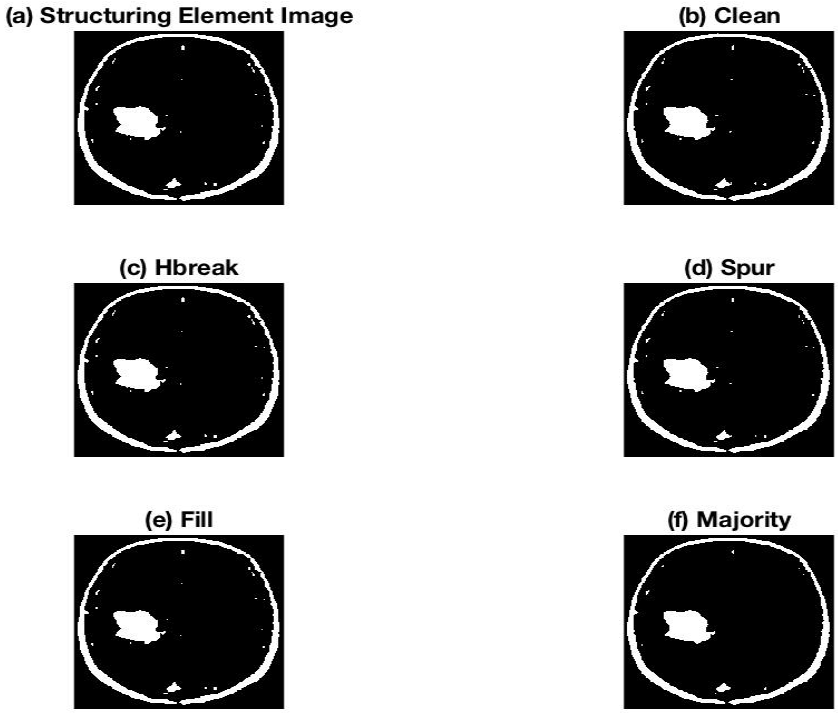
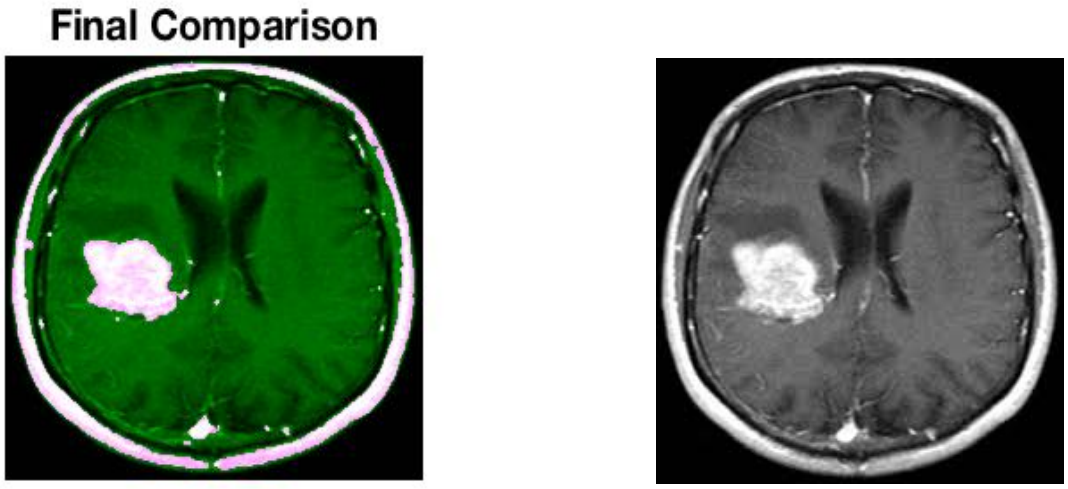


Figure 11 – Final Comparison of the Second Image

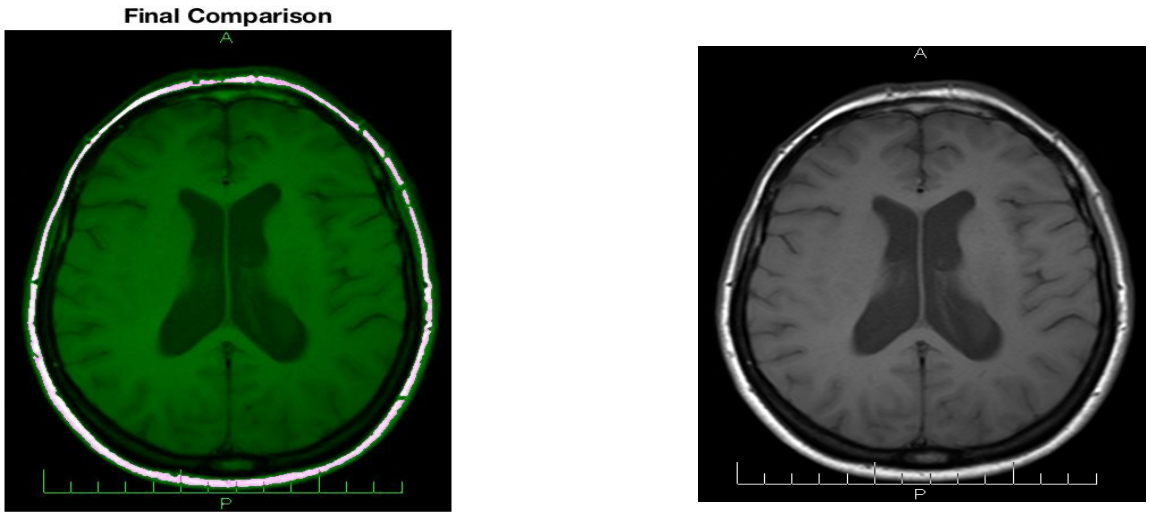


Through all of the above pictures and the final comparison image in figure 11, it is clear that the code works effectively for other images as well. However, unlike in figure 1, the MRI of the

brain in figure 2 has a very bright outer ring corresponding to the skull that is highlighted in the process. Despite this, the tumor was still extracted effectively and the image obtained from the code can still be used for the practical application of comparing tumors 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 figure 12, it is clear see that no region other than the outer ring of the brain was highlighted. Therefore, we 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 incredibly encouraging and satisfying as it is evident that the program effectively extracts regions of the brain corresponding to tumors. For future work, it would be fruitful to come up with a new threshold segmentation method that determines the level to be used based on the input image. Adding such a feature to the existing program would only serve to bolster the immense applications that the program currently offers in the field of medicine.

Works Cited

1. Angel, Aaron. "MATLAB PROGRAM : 2D MEDIAN FILTERING FOR SALT AND PEPPER NOISE WITHOUT USING medfilt2 FUNCTION." angeljohnsy.blogspot.com/2011/03/2d-median-filtering-for-salt-and-pepper.html.

2. "Binarize Image by Thresholding - MATLAB Imbinarize." *Binarize Image by Thresholding - MATLAB Imbinarize*, Mathworks, www.mathworks.com/help/images/ref/imbinarize.html.

3. "Convolution." Cornell, 27 Feb. 2013, www.cs.cornell.edu/courses/cs1114/2013sp/sections/s06_convolution.pdf.

4. Eddins, Steve. "The Watershed Transform: Strategies for Image Segmentation." Mathworks, 2002, www.mathworks.com/company/newsletters/articles/the-watershed-transform-strategies-for-image-segmentation.html.

5. "Filtering an Image." *Filtering an Image*, IDL, 16 June 2005, northstar-www.dartmouth.edu/doc/idl/html_6.2/Filtering_an_Imagehvr.html.

6. Fisher, R et al. "Sobel Edge Detector." *Feature Detectors - Sobel Edge Detector*, 2003, homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm.

7. "Global Image Threshold Using Otsu's Method - MATLAB Graythresh." Mathworks, www.mathworks.com/help/images/ref/graythresh.html.

8. "Morphological Image Processing." www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm.

9. "*Morphological Operations on Binary Images - MATLAB Bwmorph*," Mathworks, www.mathworks.com/help/images/ref/bwmorph.html.

10. Patil, Rajesh C., and A.S. Balchandra. "Brain Tumour Extraction from MRI Images Using MATLAB." *IJECSCSE*, vol. 2, no. 1, www.ijecscse.org/papers/apr2012/Brain-Tumour-Extraction-from-MRI-Images-Using-MATLAB.pdf.

11. "What Is Grayscale?" TechTarget, whatis.techtarget.com/definition/grayscale.

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
```