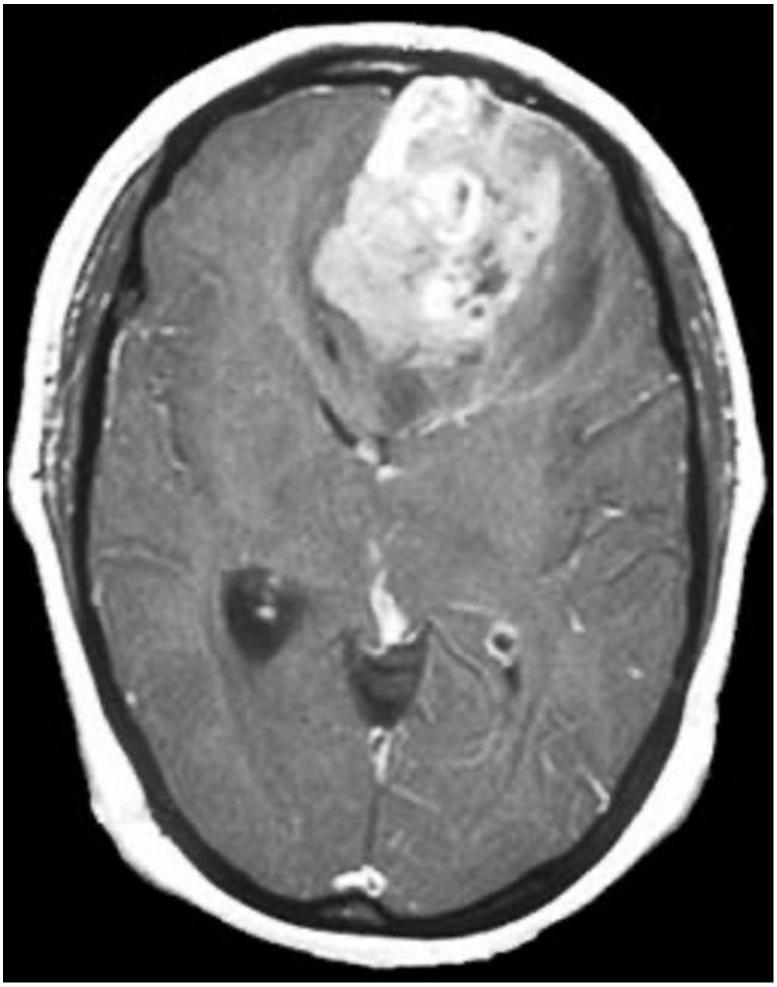
Brain Tumor Segmentation from Static T1 MRIs

CSE 554: Geometric Computing for Biomedicine Final Project



Ritika Tejwani

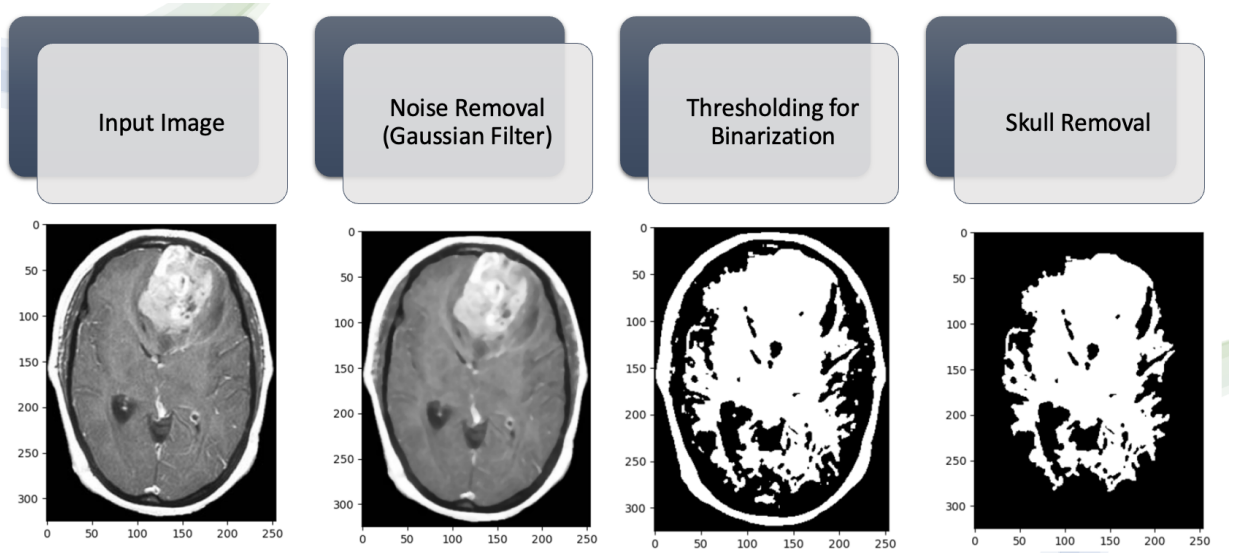
Introduction

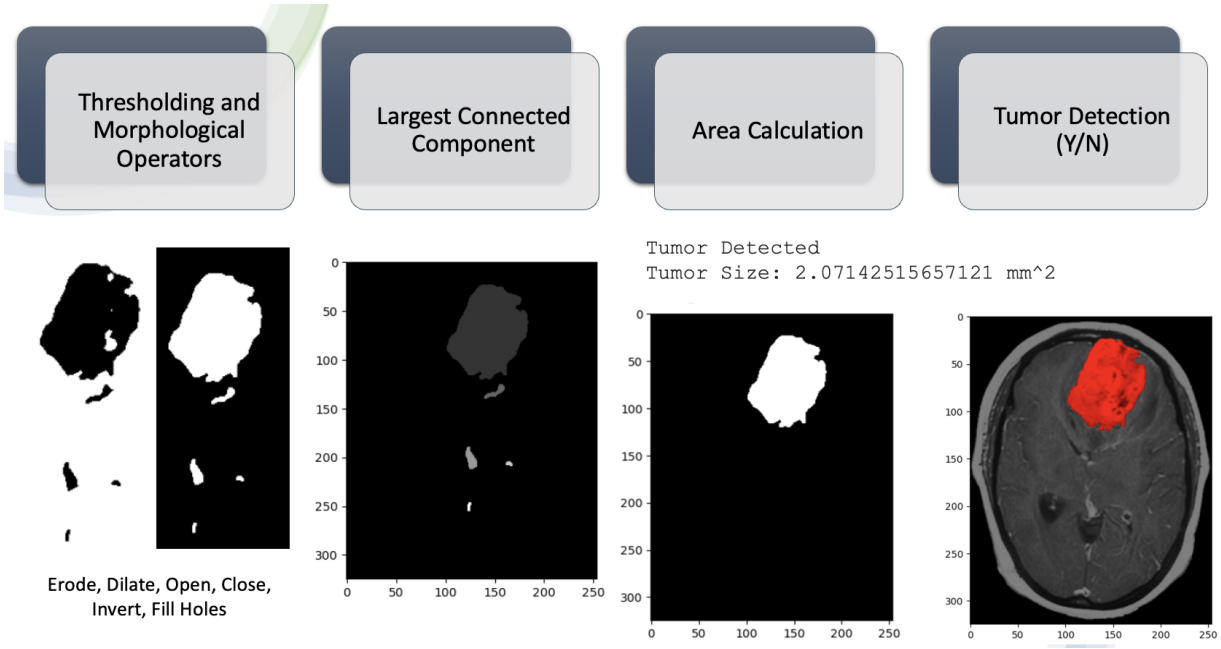
For the final project, I opted for Option 2: solving a real-world problem in biomedicine. One area of research that uses geometric computing is tumor segmentation from brain MRI scans. Brain tumors, both benign and malignant, are abnormal masses of tissue in which cells grow and multiply uncontrollably. Currently, computationally driven tumor segmentation techniques lack accuracy and reliability but are needed in the medical field to improve the efficiency of clinical diagnosis. I developed a tool to scan static TI Magnetic Resonance Images (MRIs) for abnormalities, with the prospect of detecting tumor growths. Various quantitative characteristics of tumors in the brain, such as the volume/area, density, and location may help improve the general understanding of strokes, brain cancer, and other life-threatening diseases and their treatments.

Accomplished Features

The basic requirement for the program was to be able to identify the largest brain tumor given an area of an MRI scan. The tool accomplished this requirement but also went beyond it as I didn't have to prompt the user to outline the area in question; my tool was able to identify the largest white matter tumor given the full static MRI. In addition to the basic requirement, the program also had a feature to measure the area of the tumor from the image. Area measurement and location determination is important for understanding the characteristics of the tumor, the medical implications it imposes, and treatment planning. I used an open source dataset from Kaggle to test the program.

Methodology





Core Algorithms and Significant Coding Components

From the initial input image, I applied a Gaussian Filter using Opencv library. A Gaussian Filter is a linear filtering method that reduces noise on the image by blurring it and reducing the contrast. I then converted the image to a grayscale image by extracting the red, green, and blue RGB components. The noise removal and grayscale conversion allow for higher accuracy in the next algorithm, which is skull stripping. In this algorithm, I attempted to identify the skull using thresholding and component labeling. The component labeling algorithm consists of a labeling function from an Open CV library and a get largest component algorithm I created. Together, the program labels every object pixel of the binary picture by the index of its containing connected component, and extracts the largest connected components from the image, while changing the rest of the pixels to background values (0).

Initially, at a low threshold, every non-zero pixel will be labeled as foreground and ignoring noise, there will likely be one connected component. I started with this very low threshold value and continued increasing the threshold until there were at least 10 connected components in the resulting binary image. I chose the value of 10 connected components in order to take noise into consideration, which would add to the number of connected components in the image even though it's not part of the actual brain in the scan. Once the threshold value reaches this point, I used a labeling function from an Open CV library and created a get largest component algorithm to extract the largest connected component from the image. Theoretically, because pixels representing the skull usually have a higher greyscale value than the inside of the brain, this would separate the brain from the skull, leaving the brain as the largest connected component.

From there, I used my thresholding algorithm again, at the user inputted threshold value, on the brain component. Because the tumor is generally a white mass with a high density, the remaining picture will contain an outline/filled component highlighting the tumor area. But sometimes, the area is filled with holes in the binary picture, so I created and used a combination of inversion, erosion, dilation, opening, and closing algorithms on the image to fill the holes, remove noise, smooth the component, and create a more accurate binary image.

After the primary components are shaped, I again used labeling and the get largest component algorithm to extract the largest connected component from the image. I calculated the area of the component (abnormality) by adding the number of white pixels and dividing by the number of total pixels. I then used a conversion rate (one millimeter squared is equivalent to 8.045 pixels) to get the area in a standard measurement. If the area of the tumor was greater than 0.5 millimeter squared, the program classifies it as a tumor and outputs the corresponding information. If not, it will state that there was no tumor detected in the screening.

The significant coding components consisted of the morphological operators, get largest components, area measurement and calculation, and thresholding algorithms.

GUI Development

To create a Graphic User Interface (GUI), I used Python's ipywidget library and embedded several widgets in the notebook. A widget is a GUI element that allows the user to indirectly interact with the code is a user friendly manner by responding to events and invoking certain handlers. This allows the tool to easily collect user input, set the threshold for tumor segmentation, and see the impact changes in thresholding have on the data/results without manipulating the code. I created buttons, an interactive slider, and an Upload File field. First, the Upload File widget allows the user to input the MRI that will be scanned. Once a file is uploaded, the user can press the "Set Threshold" button, which by using a on-button-clicked function that calls another widget, an interactive slider will pop up on the screen. Once the user sets the slider at the desired value, there is another button, "Find Tumor," whose on-button-clicked calls the core algorithms. From there, the code outputs the processed MRI images and whether or not a tumor was detected or not (and if so, the measured area of the suspected tumor in millimeters squared).

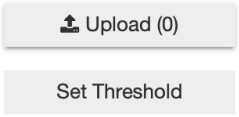
Future Work

One of the major bugs of my program is that when there's a lot of noise or the skull is part of the same connected component as the tumor, my algorithm is not able to correctly identify and remove the skull from the image. In this situation, the program misidentifies the skull as the tumor. Looking forward, I would like to explore more advanced skull stripping algorithms. I suspect I could increase accuracy by using a dynamic MRI, where I could create an algorithm to analyze the density of white matter on the scan, which would then allow me to then differentiate between skull and tumor masses as they are known to have different densities. In addition, I would also like to learn more about the water concentration in the brain and how that affects imaging. Generally, masses with higher water content appear darker on an MRI scan. In such a case, my algorithm wouldn't work because it depends heavily on greyscale and thresholding values to determine if and where there is a tumor. I would like to explore the other factors involved in diagnosing a tumor and reading an MRI scan rather than just looking at the greyscale value. I would also want to explore how this algorithm would apply or vary on different types of brain MRI scans, such as T2 scans.

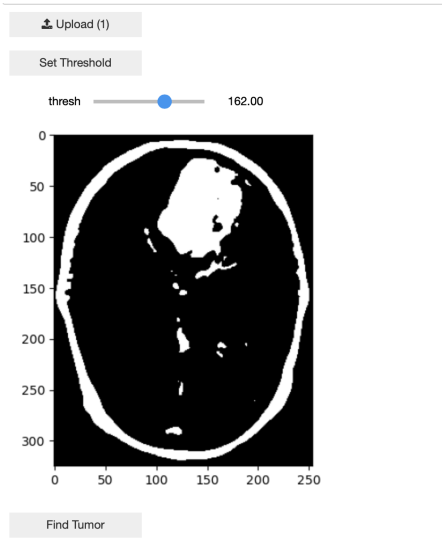
Readme

I made the tool in Jupyter Notebook but it can be used in any Python IDE. After running all the cells, navigate to the last code cell. There, you will be able to interact with widgets and use the tool I created. First, using the first widget, upload an MRI image of your choice.

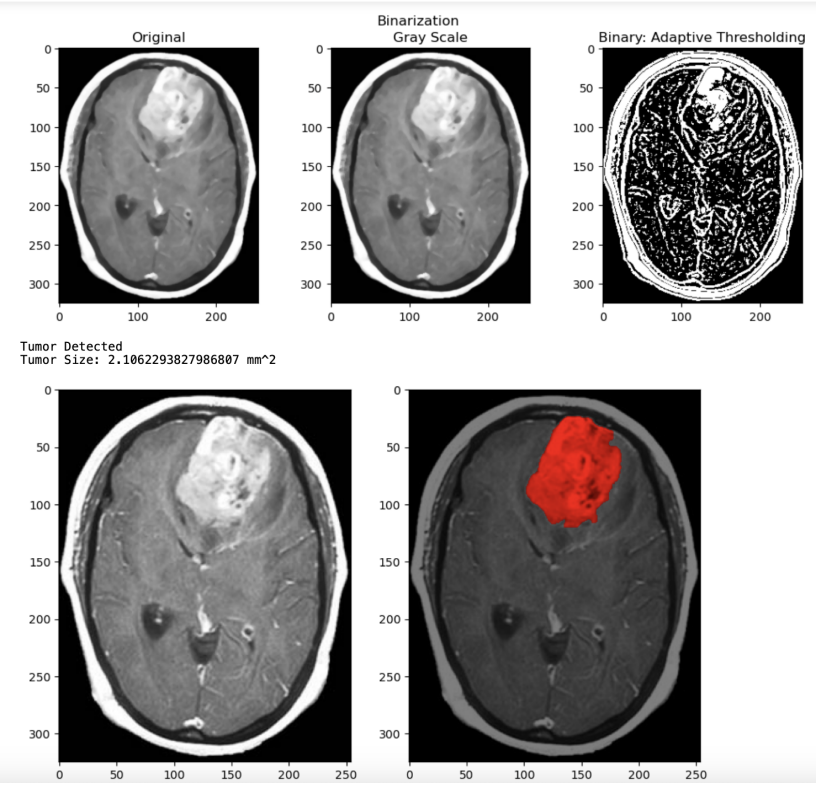
```
MRI_image = widgets.FileUpload(multiple=False)
button = widgets.Button(description='Set Threshold',disabled=False)
button.on_click(on_button1_clicked)
display(MRI_image, button)
```



Once uploaded, select the "Set Threshold" button. This prompts an interactive slider, where you can choose what value to set the threshold at, assigns the pixel values of the image into one of two different groups, background (black) or foreground (white), based on the greyscale value (intensity) of the pixel and a user selected threshold value. The segmentation of the tumor heavily depends on the threshold value selected. Ideally, you would select a threshold value that highlights a segmented region of the scan.



After selecting the desired threshold value, press the “Find Tumor” button. This will call the detection algorithm, which will print the binarization of the image for display purposes and then will output if there was a tumor detected or not and if so, the size and the location of the tumor displayed as a red overlay on the original image.



Re-run the same cell and repeat this process to test this algorithm on other inputs.

References

- **Dataset:** https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection/data
- https://www.ijser.org/researchpaper/Efficient-Brain-Tumor-Detection-Using-Image-Processing-Techniques.pdf
- https://ieeexplore.ieee.org/document/9719773/references#references
- https://ieeexplore.ieee.org/document/8282559
- https://ieeexplore.ieee.org/document/6141398
- https://ieeexplore.ieee.org/document/5674887
- https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0b9d799aa69e9e0b1e35ba5662d46fd8de9d7ffe
- https://ieeexplore.ieee.org/document/7002427
- https://www.irjet.net/archives/V7/i4/IRJET-V7I4620.pdf