"Brain Tumor Detection: A MATLAB-Based Approach":

Introduction

The project leverages **MATLAB**'s robust image processing capabilities to develop a graphical user interface (GUI) that automates brain tumor detection from MRI scans. This approach addresses the limitations of manual tumor detection, which is often time-consuming, labor-intensive, and prone to human error. The proposed system is designed to enhance diagnostic accuracy while simplifying the workflow for medical professionals.

Problem Statement

Manual detection of brain tumors from MRI scans is error-prone and inefficient, requiring significant expertise and time. This project aims to create a solution that automates this process using MATLAB, thereby reducing the reliance on manual intervention and improving diagnostic precision.

Proposed Solution

The project involves the development of a MATLAB-based application featuring a **user-friendly GUI** to streamline the tumor detection process. The application integrates key image processing techniques, from noise reduction to tumor segmentation and visualization, ensuring high accuracy and reliability in detection.

Methodology

The tumor detection workflow is broken down into the following stages:

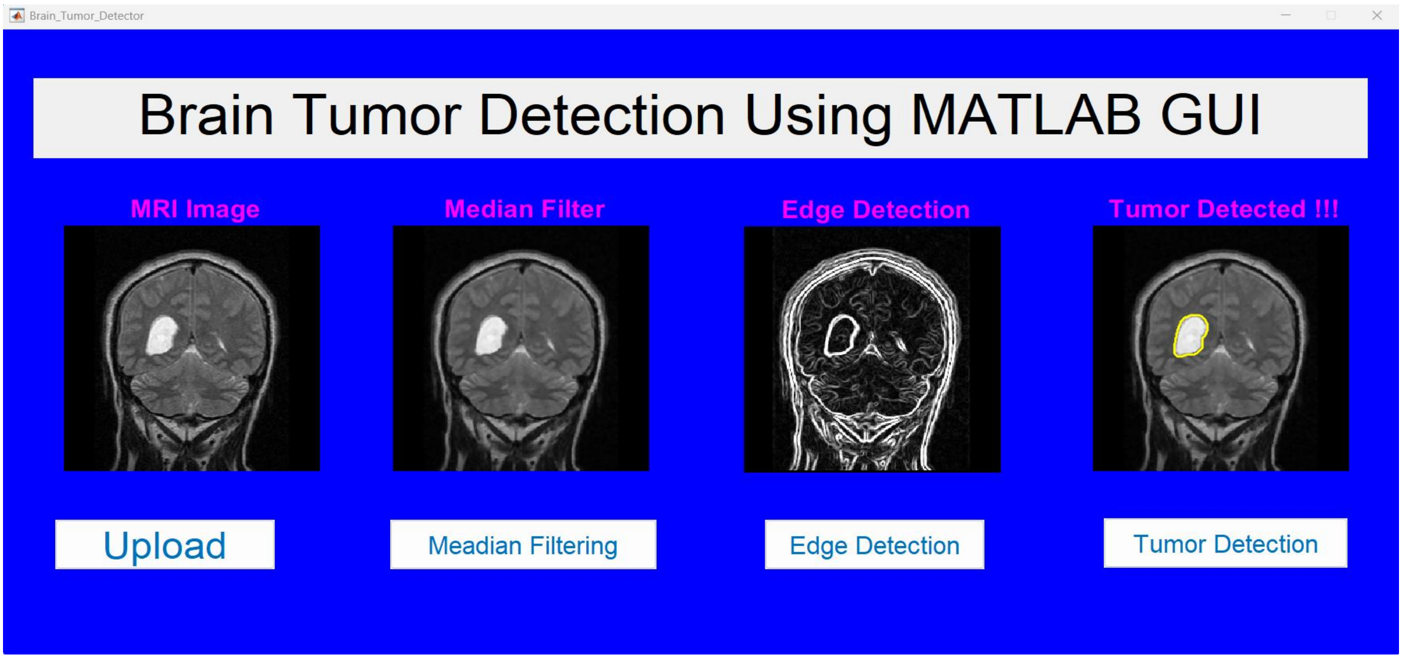
1. **Image Acquisition**:
   - Users select MRI images via MATLAB's imgetfile function.
   - The selected image is displayed in the GUI for further processing.
2. **Preprocessing**:

- **Noise Reduction**: Noise in the MRI images is minimized using **median filtering** through the medfilt2 function.
- **Gray Conversion**: RGB images are converted to grayscale, as grayscale simplifies edge detection and segmentation.

3. **Edge Detection**:
   - **Sobel Operator** is used to calculate image gradients, identifying the intensity variations.
   - Edge directions (Gx, Gy) are computed to generate a binary edge map, visualizing the boundaries of potential tumor regions.
4. **Tumor Detection**:
   - **Segmentation**: Thresholding isolates the tumor from the rest of the brain structure, while dilation enhances its visibility.
   - **Region Analysis**: Tumor properties like solidity and area are calculated using MATLAB's regionprops function.
   - **Overlay**: The tumor boundaries are highlighted and overlaid on the original MRI image for visual verification.
5. **Results Visualization**:
   - The GUI displays the processed image with the detected tumor region, providing an intuitive visualization of results.

<u>Graphical User Interface (GUI) Features</u>

- **Singleton Design**: Ensures only one instance of the GUI operates at a time, preventing resource conflicts.
- **GUIDE Tool**: MATLAB's GUIDE environment was used to design the GUI, allowing users to interact easily with the application for image loading, processing, and results display.



MATLAB Code

```
function varargout = Brain_Tumor_Detector(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Brain_Tumor_Detector_OpeningFcn, ...
                   'gui_OutputFcn',  @Brain_Tumor_Detector_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Brain_Tumor_Detector is made visible.
function Brain_Tumor_Detector_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for Brain_Tumor_Detector
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = Brain_Tumor_Detector_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in select_mage.
function select_mage_Callback(hObject, eventdata, handles)
% Allow user to select an MRI image
global img1 img2
[path, nofile] = imgetfile();

if nofile
    msgbox('Image not found!!!', 'Error', 'Warning');
    return;
end

img1 = imread(path);
img1 = im2double(img1);
img2 = img1;

axes(handles.axes1);
imshow(img1);
title('\fontsize{20}\color[rgb]{1,0,1} MRI Image');

% --- Executes on button press in meadian_filtering.
function meadian_filtering_Callback(hObject, eventdata, handles)
% Apply median filtering to remove noise
global img1

if size(img1,3) == 3
    img1 = rgb2gray(img1);
end

K = medfilt2(img1);
axes(handles.axes2);
imshow(K);
```

```
title('\fontsize{20}\color[rgb]{1,0,1} Median Filter');

% --- Executes on button press in edge_detection.
function edge_detection_Callback(hObject, eventdata, handles)
% Perform edge detection using Sobel operator
global img1

if size(img1,3) == 3
    img1 = rgb2gray(img1);
end

K = medfilt2(img1);
C = double(K);
B = zeros(size(C));

for i = 1:size(C,1)-2
    for j = 1:size(C,2)-2
        % Sobel mask for X-direction
        Gx = ((2*C(i+2,j+1) + C(i+2,j) + C(i+2,j+2)) - (2*C(i,j+1) + C(i,j) + C(i,j+2)));
        % Sobel mask for Y-direction
        Gy = ((2*C(i+1,j+2) + C(i,j+2) + C(i+2,j+2)) - (2*C(i+1,j) + C(i,j) + C(i+2,j)));
        % The Gradient of The Image
        B(i,j) = sqrt(Gx.^2 + Gy.^2);
    end
end

axes(handles.axes3);
imshow(B);
title('\fontsize{20}\color[rgb]{1,0,1} Edge Detection');

% --- Executes on button press in tumor_detection.
function tumor_detection_Callback(hObject, eventdata, handles)
% Detect the tumor region in the MRI image
global img1

K = medfilt2(img1);
bw = imbinarize(K, 0.7);
label = bwlabel(bw);
stats = regionprops(label, 'Solidity', 'Area');

density = [stats.Solidity];
area = [stats.Area];
high_dense_area = density > 0.5;
max_area = max(area(high_dense_area));
tumor_label = find(area == max_area);
tumor = ismember(label, tumor_label);

se = strel('square', 5);
tumor = imdilate(tumor, se);

Bound = bwboundaries(tumor, 'noholes');

axes(handles.axes4);
imshow(K);
hold on;
for i = 1:length(Bound)
    plot(Bound{i}(:,2), Bound{i}(:,1), 'y', 'linewidth', 1.75);
end
title('\fontsize{20}\color[rgb]{1,0,1} Tumor Detected !!!');
hold off;
```

Results

- The system demonstrates **high accuracy** in detecting brain tumors across test datasets, validating the effectiveness of the image processing techniques used.
- The GUI simplifies the workflow, making the tool accessible to medical professionals with limited technical expertise in MATLAB.

Future Scope

1. **Integration of Machine Learning**:
   - Incorporate advanced machine learning or deep learning algorithms to enhance tumor detection accuracy further.
   - Enable classification of tumor types (e.g., benign vs. malignant).
2. **Enhanced Usability**:
   - Improve the GUI's design and interactivity to make the application more user-friendly.
   - Add features for real-time processing and detailed reporting.
3. **Broader Applications**:
   - Adapt the application to analyze other medical imaging modalities, such as CT scans or PET scans, for broader applicability.
   - Expand detection capabilities to other medical conditions beyond brain tumors.

Conclusion

This MATLAB-based approach demonstrates the potential to revolutionize brain tumor detection through automation. By combining advanced image processing techniques with a user-centric design, the project paves the way for improved diagnostic workflows and accuracy. Future enhancements with machine learning and expanded usability will further solidify its value in medical imaging.

SUBMITTED BY: -

PRAJWAL B R – ENG22CS0121

LIKITH S J – ENG22CS0089

NITIN P HEGDE – ENG22CS0113

KEERTHANA N M – ENG22CS0079