Final Project: Brain Tumor Segmentation App

Submitted To:	Mr Rizwan Shah
Submitted By:	Hania Rasheed
Registration No:	B23S0983DS009
Department:	IT & CS
Course Code:	COMP 342 - L
Date:	25th April, 2025

Introduction:

This project implements a classical image processing pipeline for segmenting brain tumors from MRI scans. Built using Streamlit, it offers users interactive controls to test various preprocessing and segmentation methods, evaluate textural features using GLCM, and detect tumor contours. Unlike deep learning models, this app uses transparent, explainable techniques.

Technology Used:

Component	Library/Tool
UI/UX	Streamlit
Image Processing	OpenCV
Image Handling	PIL
Texture Features	scikit-image
Clustering	Scikit-learn, Scikit-fuzzy
Programming Language	Python

Streamlit (st): This library is used for creating interactive web applications. It allows you to easily create UI components (sliders, checkboxes, etc.) and display images, charts, and text.

OpenCV (cv2): A library for computer vision tasks, such as reading images, applying image processing operations, and detecting contours.

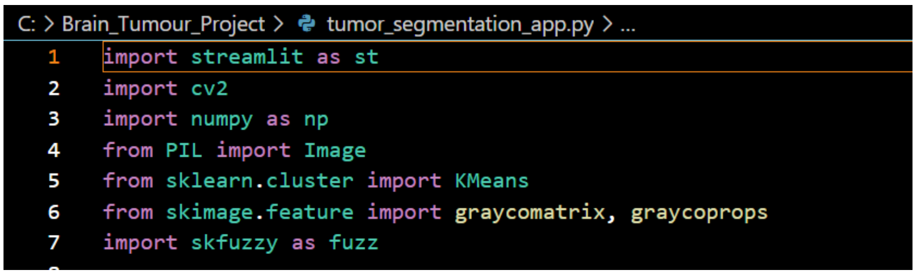
NumPy (np): Essential for working with arrays, particularly in mathematical operations and image processing where the image is represented as a matrix.

Pillow (PIL.Image): Used for opening and converting images to grayscale, which is important for processing medical images like MRIs.

KMeans from Scikit-Learn: Implements the K-Means clustering algorithm, which groups image pixels into a number of clusters (used for segmentation).

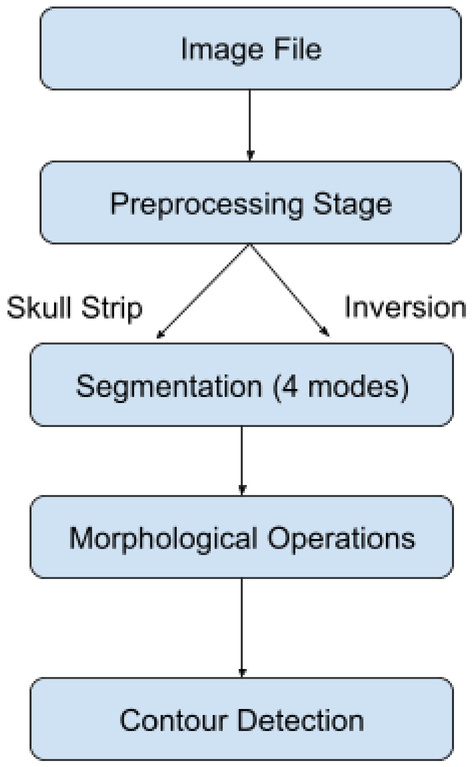
skimage.feature (graycomatrix, graycoprops): Used for extracting GLCM (Gray-Level Co-occurrence Matrix) features like contrast, correlation, energy, and homogeneity.

scikit-fuzzy (fuzz): Provides the implementation of fuzzy C-Means clustering, which is used for soft clustering of image pixels.


C: > Brain_Tumour_Project > tumor_segmentation_app.py > ...

```python
import streamlit as st
import cv2
import numpy as np
from PIL import Image
from sklearn.cluster import KMeans
from skimage.feature import graycomatrix, graycoprops
import skfuzzy as fuzz
```

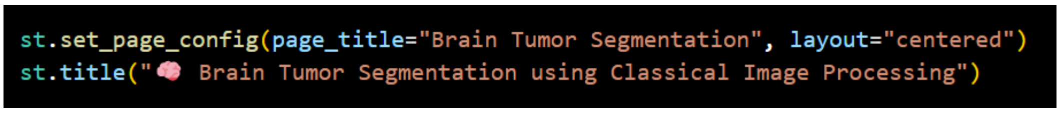
Working of the App:



Code Chunks:

Streamlit Page Configuration and Header:

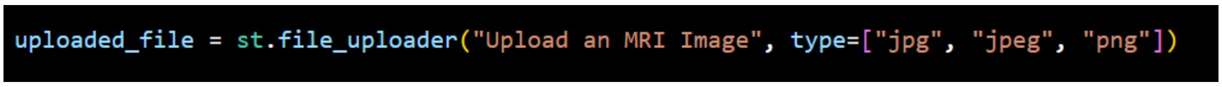
This section sets the page configuration for Streamlit (title and layout). The title of the app is displayed in the center.



```
st.set_page_config(page_title="Brain Tumor Segmentation", layout="centered")
st.title("🧠 Brain Tumor Segmentation using Classical Image Processing")
```

File Uploader for MRI Image:

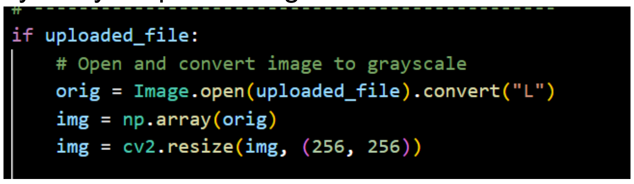
Allows users to upload MRI images. Accepts .png, .jpg, .jpeg. The uploaded image will then be processed by the app.



```
uploaded_file = st.file_uploader("Upload an MRI Image", type=["jpg", "jpeg", "png"])
```

Preprocessing: Grayscale + Resize:

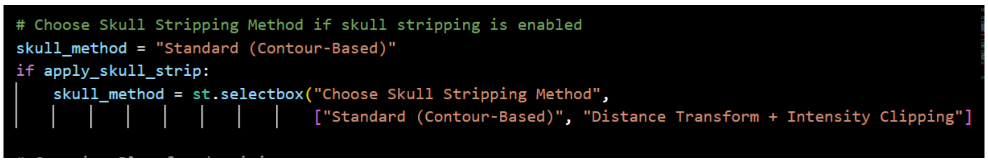
- Converts image to grayscale (L mode).
- Resizes it to 256x256 pixels.
- Converts it into a NumPy array for processing.



```
# ---------------------------------------------------------
if uploaded_file:
    # Open and convert image to grayscale
    orig = Image.open(uploaded_file).convert("L")
    img = np.array(orig)
    img = cv2.resize(img, (256, 256))
```

Skull Stripping - Two Methods

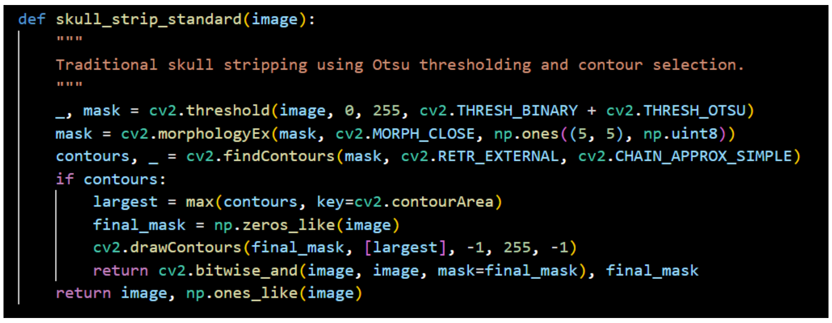
User picks whether and how to strip the skull from the image.This function removes the skull portion from the MRI image to focus on the brain tissue. It uses Otsu's thresholding to create a binary mask and morphological operations to clean the mask.



```python
# Choose Skull Stripping Method if skull stripping is enabled
skull_method = "Standard (Contour-Based)"
if apply_skull_strip:
    skull_method = st.selectbox("Choose Skull Stripping Method",
                                ["Standard (Contour-Based)", "Distance Transform + Intensity Clipping"]
```

Option 1: Standard (Otsu + Largest Contour):

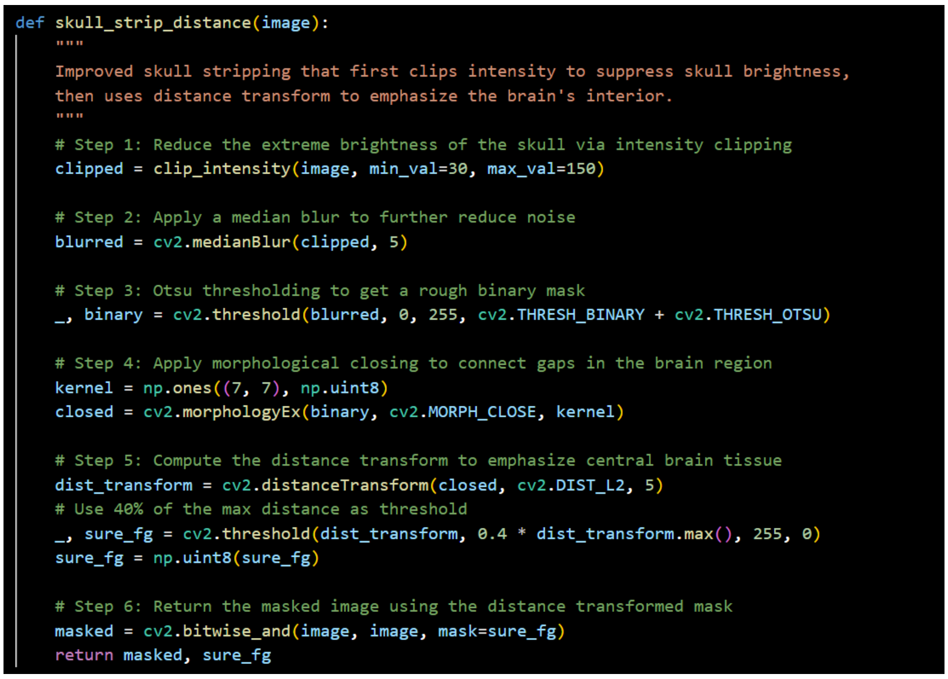
- Uses Otsu thresholding.
- Finds the largest external contour (likely the brain).
- Masks everything else out.



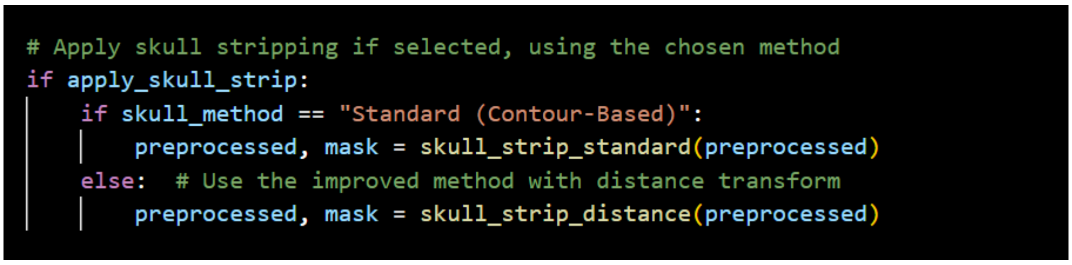
```python
def skull_strip_standard(image):
    """
    Traditional skull stripping using Otsu thresholding and contour selection.
    """
    _, mask = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, np.ones((5, 5), np.uint8))
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        largest = max(contours, key=cv2.contourArea)
        final_mask = np.zeros_like(image)
        cv2.drawContours(final_mask, [largest], -1, 255, -1)
        return cv2.bitwise_and(image, image, mask=final_mask), final_mask
    return image, np.ones_like(image)
```

Option 2: Distance Transform Based:

- Applies Otsu, then morphological closing.
- Applies distance transform to identify central regions of the brain.
- Thresholds distance map and masks it.

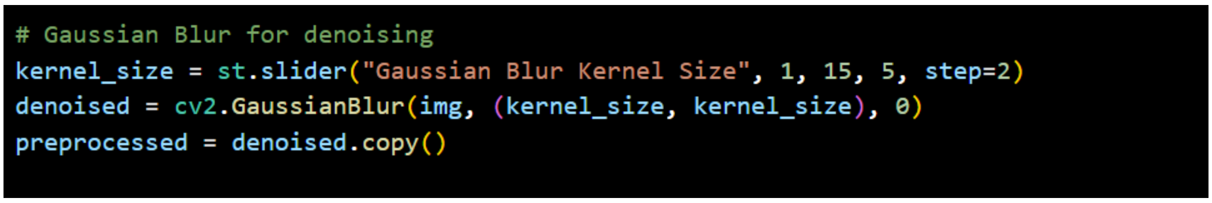


```python
def skull_strip_distance(image):
    """
    Improved skull stripping that first clips intensity to suppress skull brightness,
    then uses distance transform to emphasize the brain's interior.
    """
    # Step 1: Reduce the extreme brightness of the skull via intensity clipping
    clipped = clip_intensity(image, min_val=30, max_val=150)

    # Step 2: Apply a median blur to further reduce noise
    blurred = cv2.medianBlur(clipped, 5)

    # Step 3: Otsu thresholding to get a rough binary mask
    _, binary = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Step 4: Apply morphological closing to connect gaps in the brain region
    kernel = np.ones((7, 7), np.uint8)
    closed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)

    # Step 5: Compute the distance transform to emphasize central brain tissue
    dist_transform = cv2.distanceTransform(closed, cv2.DIST_L2, 5)
    # Use 40% of the max distance as threshold
    _, sure_fg = cv2.threshold(dist_transform, 0.4 * dist_transform.max(), 255, 0)
    sure_fg = np.uint8(sure_fg)

    # Step 6: Return the masked image using the distance transformed mask
    masked = cv2.bitwise_and(image, image, mask=sure_fg)
    return masked, sure_fg
```



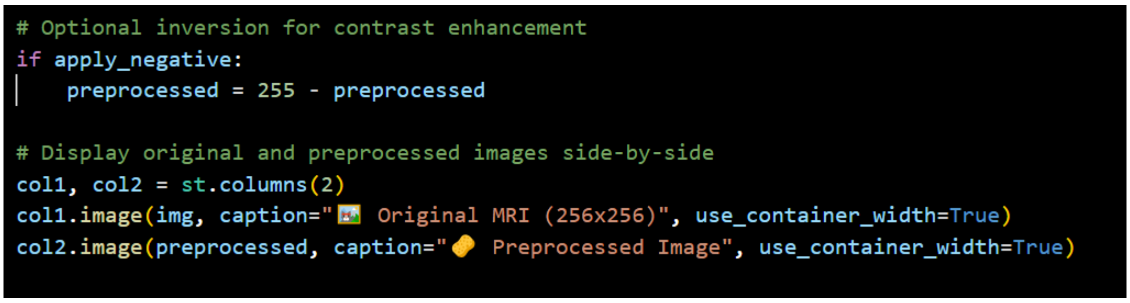
```python
# Apply skull stripping if selected, using the chosen method
if apply_skull_strip:
    if skull_method == "Standard (Contour-Based)":
        preprocessed, mask = skull_strip_standard(preprocessed)
    else:  # Use the improved method with distance transform
        preprocessed, mask = skull_strip_distance(preprocessed)
```

Gaussian Blur: Applies Gaussian blur with user-chosen odd-sized kernel.



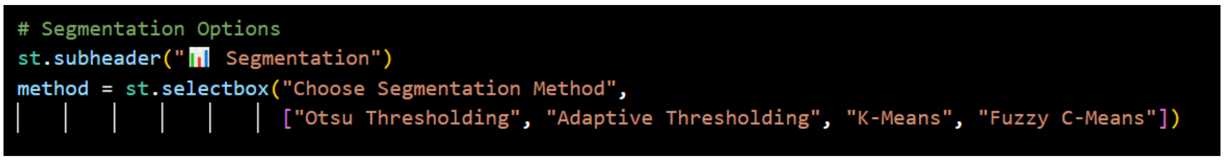
```python
# Gaussian Blur for denoising
kernel_size = st.slider("Gaussian Blur Kernel Size", 1, 15, 5, step=2)
denoised = cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
preprocessed = denoised.copy()
```

Invert Image (Optional): Simple inversion to improve contrast when needed.



```python
# Optional inversion for contrast enhancement
if apply_negative:
    preprocessed = 255 - preprocessed

# Display original and preprocessed images side-by-side
col1, col2 = st.columns(2)
col1.image(img, caption="🖼️ Original MRI (256x256)", use_container_width=True)
col2.image(preprocessed, caption="🧽 Preprocessed Image", use_container_width=True)
```

Segmentation Methods:



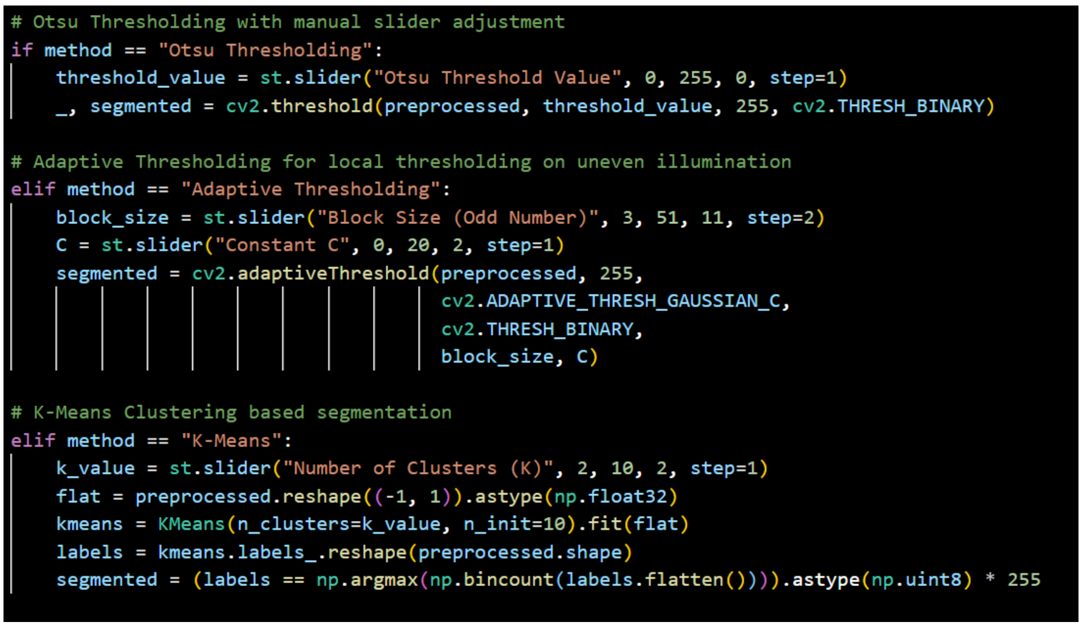
```python
# Segmentation Options
st.subheader("📊 Segmentation")
method = st.selectbox("Choose Segmentation Method",
                      ["Otsu Thresholding", "Adaptive Thresholding", "K-Means", "Fuzzy C-Means"])
```

1. **Otsu Thresholding**
    - Finds a global intensity cutoff.
    - Manual override via slider.
    - Best for bimodal histograms.

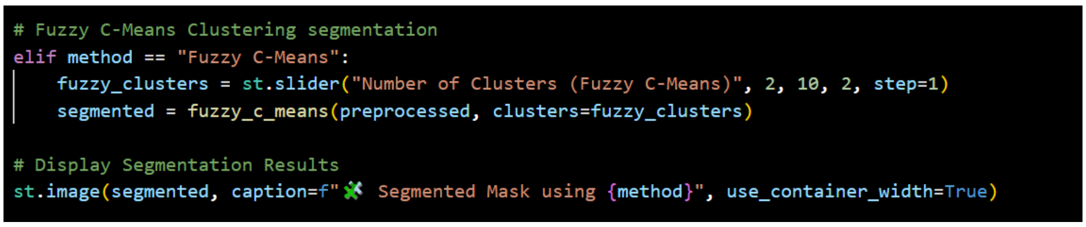
2. **Adaptive Thresholding**
    - Local thresholding for uneven lighting.
    - Parameters: block size, constant C.

3. **K-Means Clustering**
    - Groups pixels into K clusters.
    - Returns the most common cluster as tumor region.
    - Sensitive to brightness overlap.

4. **Fuzzy C-Means**
    - Soft clustering with membership values.
    - Often smoother than K-means.
    - Needs more time but is robust for mixed regions.

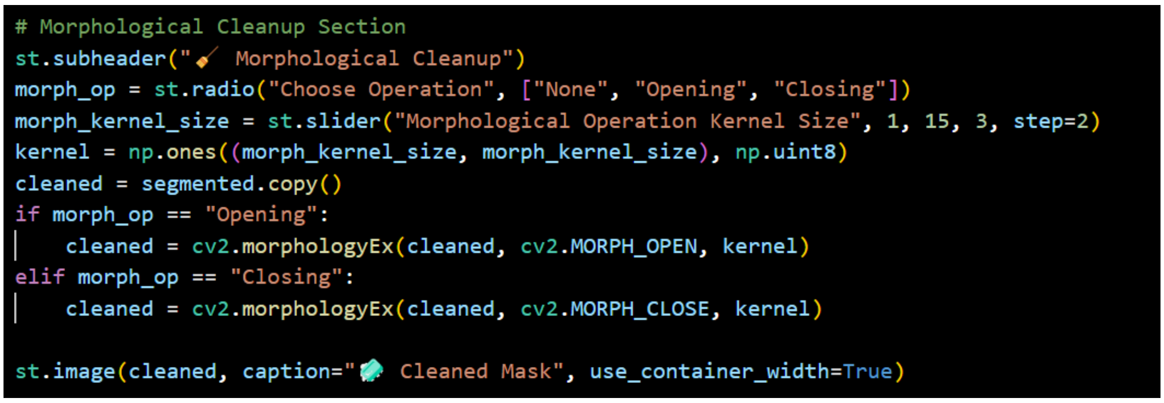


```python
# Otsu Thresholding with manual slider adjustment
if method == "Otsu Thresholding":
    threshold_value = st.slider("Otsu Threshold Value", 0, 255, 0, step=1)
    _, segmented = cv2.threshold(preprocessed, threshold_value, 255, cv2.THRESH_BINARY)

# Adaptive Thresholding for local thresholding on uneven illumination
elif method == "Adaptive Thresholding":
    block_size = st.slider("Block Size (Odd Number)", 3, 51, 11, step=2)
    C = st.slider("Constant C", 0, 20, 2, step=1)
    segmented = cv2.adaptiveThreshold(preprocessed, 255,
                                      cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                      cv2.THRESH_BINARY,
                                      block_size, C)

# K-Means Clustering based segmentation
elif method == "K-Means":
    k_value = st.slider("Number of Clusters (K)", 2, 10, 2, step=1)
    flat = preprocessed.reshape((-1, 1)).astype(np.float32)
    kmeans = KMeans(n_clusters=k_value, n_init=10).fit(flat)
    labels = kmeans.labels_.reshape(preprocessed.shape)
    segmented = (labels == np.argmax(np.bincount(labels.flatten()))).astype(np.uint8) * 255
```



```python
# Fuzzy C-Means Clustering segmentation
elif method == "Fuzzy C-Means":
    fuzzy_clusters = st.slider("Number of Clusters (Fuzzy C-Means)", 2, 10, 2, step=1)
    segmented = fuzzy_c_means(preprocessed, clusters=fuzzy_clusters)

# Display Segmentation Results
st.image(segmented, caption=f"🧩 Segmented Mask using {method}", use_container_width=True)
```

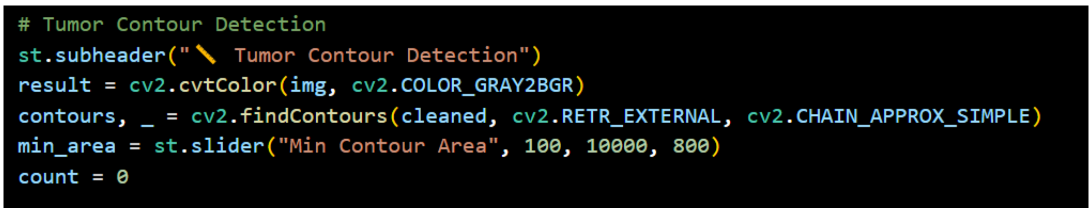
Morphological Operations: Picks morph operation type and kernel. Applies the selected operation or leaves unchanged.



```python
# Morphological Cleanup Section
st.subheader("🧹 Morphological Cleanup")
morph_op = st.radio("Choose Operation", ["None", "Opening", "Closing"])
morph_kernel_size = st.slider("Morphological Operation Kernel Size", 1, 15, 3, step=2)
kernel = np.ones((morph_kernel_size, morph_kernel_size), np.uint8)
cleaned = segmented.copy()
if morph_op == "Opening":
    cleaned = cv2.morphologyEx(cleaned, cv2.MORPH_OPEN, kernel)
elif morph_op == "Closing":
    cleaned = cv2.morphologyEx(cleaned, cv2.MORPH_CLOSE, kernel)

st.image(cleaned, caption="🧼 Cleaned Mask", use_container_width=True)
```

Tumor Detection via Contours:

- Filters by contour area and aspect ratio to highlight plausible tumor regions.
- Draws green contours on the grayscale MRI converted to color.



```python
# Tumor Contour Detection
st.subheader("📏 Tumor Contour Detection")
result = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
contours, _ = cv2.findContours(cleaned, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
min_area = st.slider("Min Contour Area", 100, 10000, 800)
count = 0
```

Function:

- **`cv2.findContours`**`(image, mode, method)`
- `cleaned`: Binary image (tumor mask after morphological ops).
- **`cv2.RETR_EXTERNAL`**:
    - Only retrieves outermost contours.
    - Useful for ignoring internal artifacts.
- **`cv2.CHAIN_APPROX_SIMPLE`:**
    - Compresses horizontal, vertical, and diagonal segments into endpoints.
    - Saves memory.
- `contours`: A list of NumPy arrays. Each array is a contour (boundary point set).
- Convert the grayscale MRI (`img`) to a 3-channel BGR image so we can draw colored contours (e.g., green).
- **`cv2.COLOR_GRAY2BGR`:** Converts from grayscale to color with 3 equal channels.
- Loops through every contour found.
- Each `contour` is an array of `(x, y)` points that make up a boundary.
- Computes the area (in pixels) enclosed by the contour.
- Used to filter out very small (noise) or very large (non-tumor) regions.

Keeps only contours with area:

➢ greater than 800 pixels → removes noise and small artifacts.

➢ less than 20,000 pixels → avoids selecting the whole brain or large non-tumor blobs.

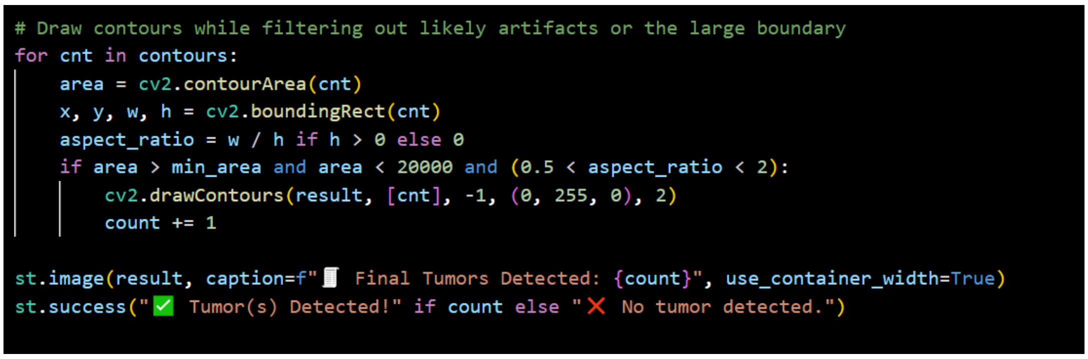
- **`cv2.boundingRect`**`(contour)`:
    - Computes the smallest upright rectangle that fully encloses the contour.
    - Returns:
        - `x, y`: Top-left corner.
        - `w, h`: Width and height.

Aspect Ratio:

- `aspect_ratio = w / h`: Measures the "shape" of the region.
- Tumors are often blob-like, so this helps filter elongated shapes (like vessels).

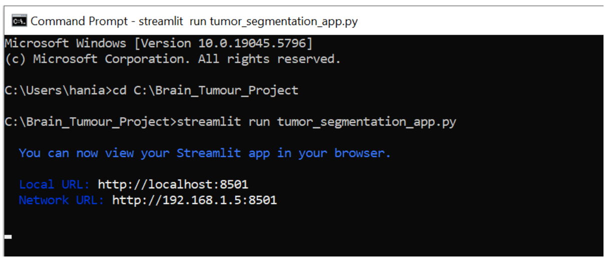
Keeps roughly square or round regions. Removes very narrow or oddly shaped segments:

- `aspect_ratio < 0.5`: Very tall.
- `aspect_ratio > 2.0`: Very wide.
- Draws the selected contour on the `img_color`.



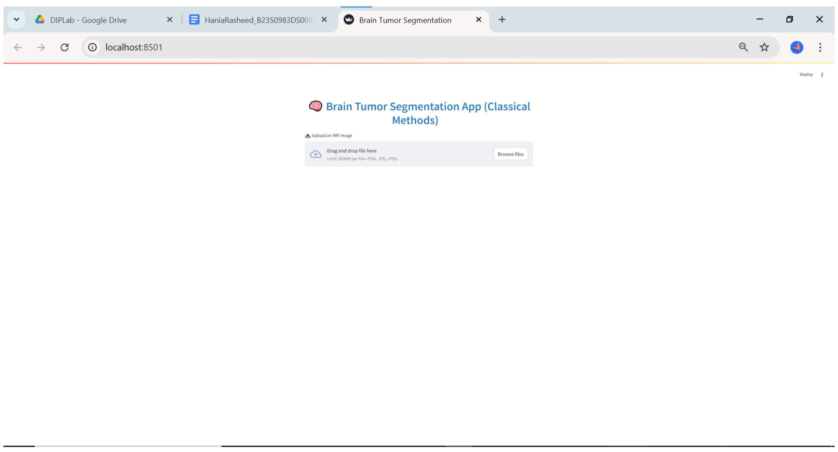
```python
# Draw contours while filtering out likely artifacts or the large boundary
for cnt in contours:
    area = cv2.contourArea(cnt)
    x, y, w, h = cv2.boundingRect(cnt)
    aspect_ratio = w / h if h > 0 else 0
    if area > min_area and area < 20000 and (0.5 < aspect_ratio < 2):
        cv2.drawContours(result, [cnt], -1, (0, 255, 0), 2)
        count += 1

st.image(result, caption=f"🧾 Final Tumors Detected: {count}", use_container_width=True)
st.success("✅ Tumor(s) Detected!" if count else "❌ No tumor detected.")
```

User Interface: To run the app, we go to CMD and run: Streamlit run app_name.py



Command Prompt - streamlit run tumor_segmentation_app.py

```
Microsoft Windows [Version 10.0.19045.5796]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hania>cd C:\Brain_Tumour_Project

C:\Brain_Tumour_Project>streamlit run tumor_segmentation_app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.5:8501
```



Brain Tumor Segmentation App (Classical Methods)

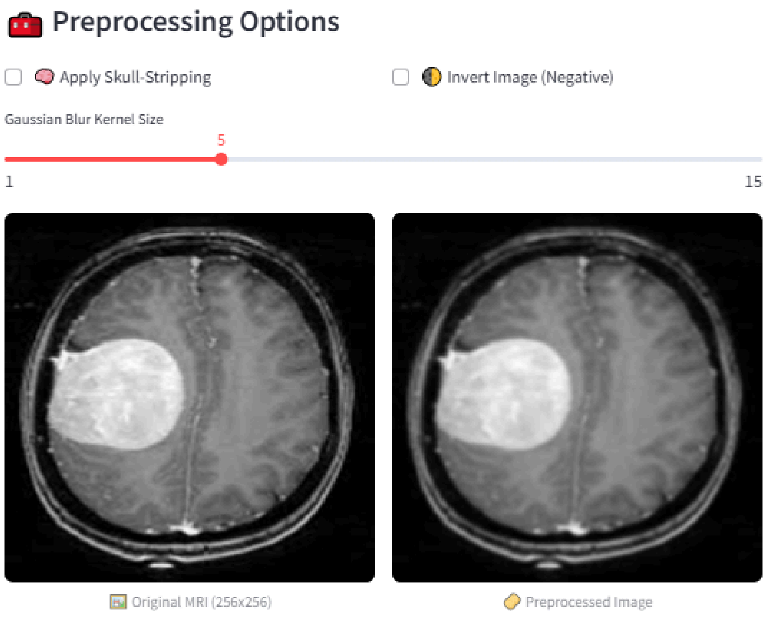
Upload an MRI Image

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

Preprocessing:

Gaussian Blur

- Reduces salt-and-pepper noise using a kernel of user-chosen size.



Preprocessing Options

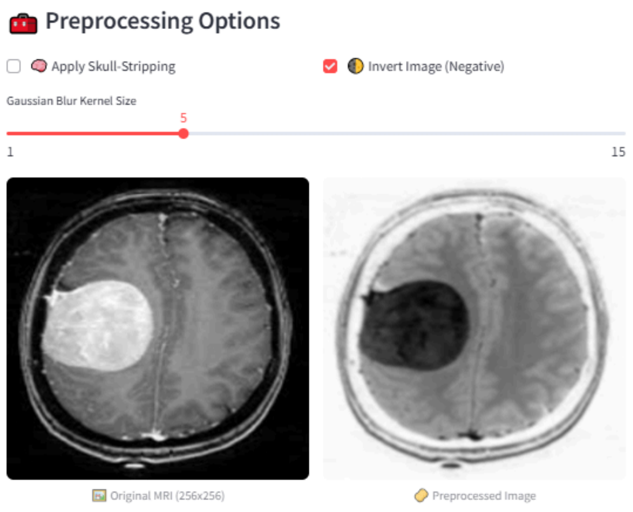
Apply Skull-Stripping   Invert Image (Negative)

Gaussian Blur Kernel Size: 5 (range 1–15)

Original MRI (256x256)   Preprocessed Image

Image Inversion (Optional)

- 255 - pixel_value improves contrast in some cases.



Preprocessing Options

☐ Apply Skull-Stripping ☑ Invert Image (Negative)

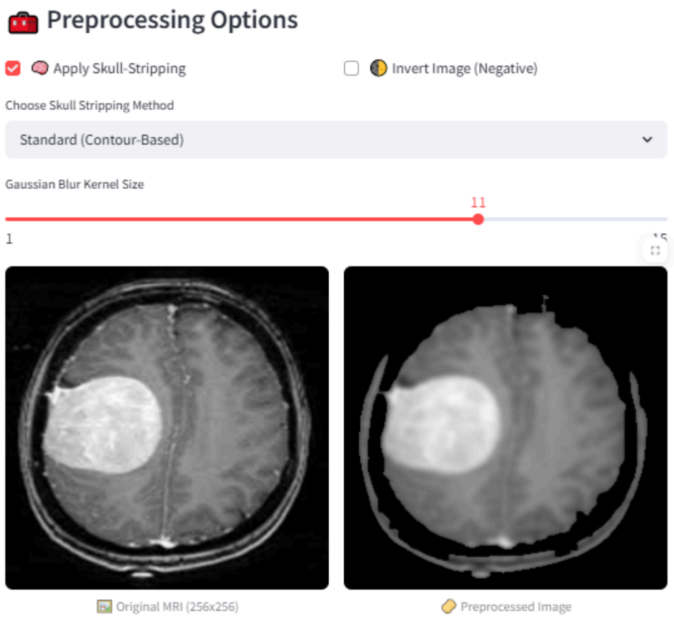
Gaussian Blur Kernel Size: 5 (1–15)

Original MRI (256x256)

Preprocessed Image

Skull Stripping (Optional)

- **Standard (Contour-Based)**:
    - Otsu threshold → largest contour → mask brain area
- **Distance Transform (Improved)**:
    - Clips skull brightness → median blur → Otsu mask → morphological closing → distance transform → central foreground emphasis



Preprocessing Options

☑ Apply Skull-Stripping ☐ Invert Image (Negative)

Choose Skull Stripping Method: Standard (Contour-Based)

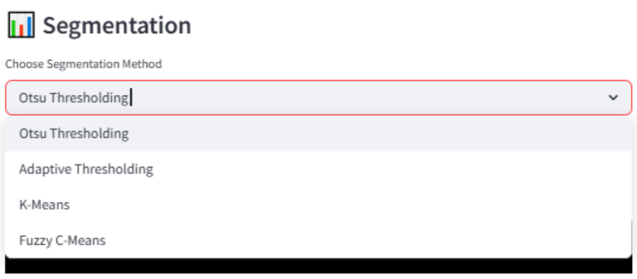
Gaussian Blur Kernel Size: 11 (1–15)

Original MRI (256x256)

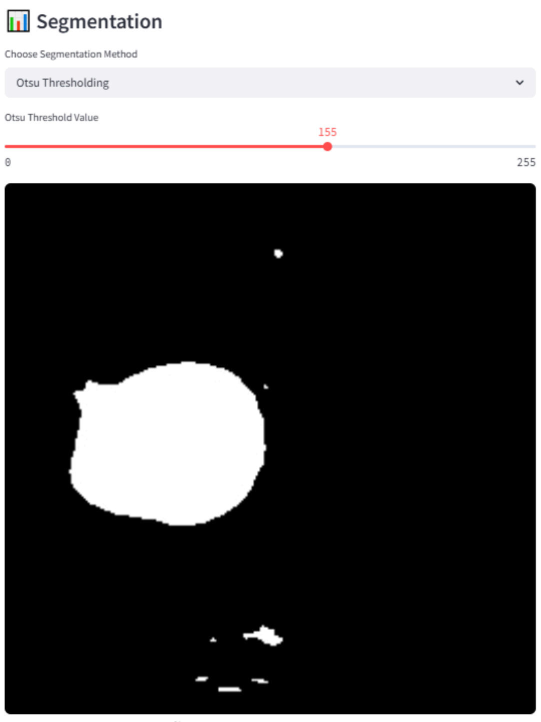
Preprocessed Image

Variation: Standard works faster; Distance Transform better at suppressing noise and skull borders.

Segmentation Methods: Users can choose from four segmentation methods:



- Otsu Thresholding: Automatically determines an optimal threshold value to separate foreground (tumor) from the background.



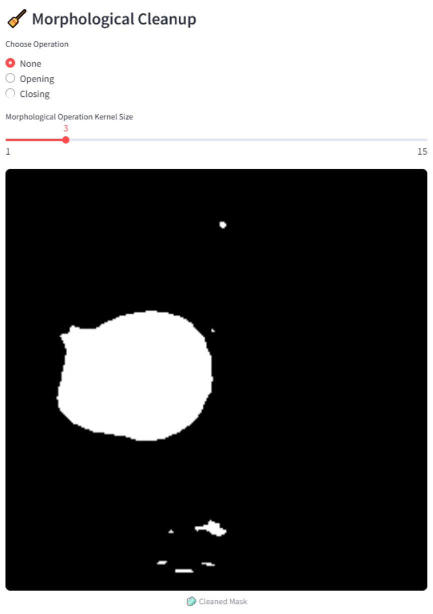
Segmented Mask using Otsu Thresholding

- K-Means Clustering: Partitions the image into a predefined number of clusters and assigns each pixel to the nearest cluster.

- Fuzzy C-Means: Soft clustering method that allows pixels to belong to multiple clusters with varying degrees of membership.

Morphological Operations: After segmentation, the user can apply morphological operations like Opening (removes noise) or Closing (fills small holes) to clean the segmented mask.

- **Opening** (Erosion → Dilation): Removes small white noise.
- **Closing** (Dilation → Erosion): Fills small black holes.
- User selects kernel size and operation.



Morphological Cleanup

Choose Operation

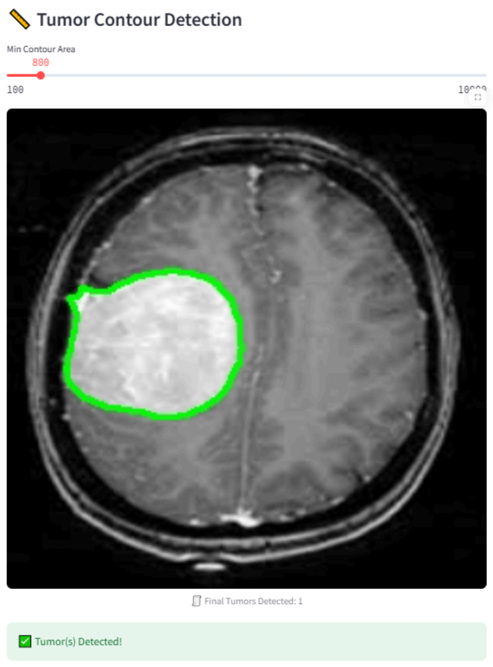
[✓ None]
Opening
Closing

Morphological Operation Kernel Size: 3 (range 1 – 15)

Cleaned Mask

8. Tumor Detection Result

Tumor Detection: Finally, the app checks how many tumors were detected and displays the result accordingly.



Tumor Contour Detection

Min Contour Area: 800 (range 100 – 10000)

Final Tumors Detected: 1

✓ Tumor(s) Detected!

Limitations:

- Not as accurate as deep learning in complex cases.
- Struggles with very low contrast or overlapping intensity distributions.
- Relies heavily on parameter tuning.
- Skull Intensity Values being similar to tumour pixel values causes issues in segmentation.

Conclusion:

This Streamlit app demonstrates that classical image processing, when well-combined and tuned, can yield surprisingly effective tumor segmentation results. It is lightweight, transparent, interpretable, and excellent for educational and research exploration.

Reference:

https://figshare.com/articles/journal_contribution/Identification_of_Brain_Tumor_from_MRI_Images_using_Segmentation/12287375?file=22644926