

data loading

```
from google.colab import drive
drive.mount('/content/drive')

import os
import shutil
import random
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from google.colab import drive

try:
    drive.mount('/content/drive')
    print("Google Drive mounted successfully!")
except:
    print("Running locally or Drive already mounted")

lgg_dataset_path = '/content/drive/MyDrive/ddd'
output_dir = '/content/drive/MyDrive/brain_tumor_dataset'

os.makedirs(os.path.join(output_dir, 'images'), exist_ok=True)
os.makedirs(os.path.join(output_dir, 'masks'), exist_ok=True)

def prepare_dataset(num_samples=10, random_selection=True):

    case_dirs = [d for d in os.listdir(lgg_dataset_path)
                  if os.path.isdir(os.path.join(lgg_dataset_path, d))]

    print(f"Found {len(case_dirs)} case directories")

    if random_selection:

        selected_cases = random.sample(case_dirs, min(len(case_dirs), num_samples))
    else:

        selected_cases = case_dirs[:min(len(case_dirs), num_samples)]

    print(f"Selected {len(selected_cases)} cases")

    processed_count = 0

    for case_id in selected_cases:
        case_dir = os.path.join(lgg_dataset_path, case_id)

        files = os.listdir(case_dir)

        image_files = [f for f in files if f.endswith('.tif') and '_mask' not in f]

        for img_file in image_files:

            mask_file = img_file.replace('.tif', '_mask.tif')

            if mask_file in files:

                img_path = os.path.join(case_dir, img_file)
                mask_path = os.path.join(case_dir, mask_file)

                img = cv2.imread(img_path)
                mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

                if img is not None and mask is not None:

                    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

                    flair = img[:, :, 1]
```

```

out_img_name = f"brain_tumor_{processed_count:03d}.png"
out_mask_name = f"brain_tumor_{processed_count:03d}_mask.png"

out_img_path = os.path.join(output_dir, 'images', out_img_name)
out_mask_path = os.path.join(output_dir, 'masks', out_mask_name)

cv2.imwrite(out_img_path, flair)
cv2.imwrite(out_mask_path, mask)

processed_count += 1
print(f"Processed {processed_count}/{num_samples}: {out_img_name}")

if processed_count >= num_samples:
    return

def visualize_dataset(dataset_path, num_samples=5):

    images_dir = os.path.join(dataset_path, 'images')
    masks_dir = os.path.join(dataset_path, 'masks')

    image_files = sorted(os.listdir(images_dir))

    image_files = image_files[:min(len(image_files), num_samples)]

    plt.figure(figsize=(12, 4 * len(image_files)))

    for i, img_file in enumerate(image_files):

        if img_file.replace('.png', '_mask.png') in os.listdir(masks_dir):
            mask_file = img_file.replace('.png', '_mask.png')
        else:

            mask_file = next((m for m in os.listdir(masks_dir) if m.startswith(img_file.split('.')[0])), None)

        if mask_file:

            img_path = os.path.join(images_dir, img_file)
            mask_path = os.path.join(masks_dir, mask_file)

            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE) # Read as grayscale
            mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

            plt.subplot(len(image_files), 2, i * 2 + 1)
            plt.imshow(img, cmap='gray')
            plt.title(f"MRI Image: {img_file}")
            plt.axis('off')

            plt.subplot(len(image_files), 2, i * 2 + 2)
            plt.imshow(mask, cmap='gray')
            plt.title(f"Tumor Mask: {mask_file}")
            plt.axis('off')

    plt.tight_layout()
    plt.show()

print("Preparing dataset...")
prepare_dataset(num_samples=10, random_selection=True)
print("Dataset preparation completed!")

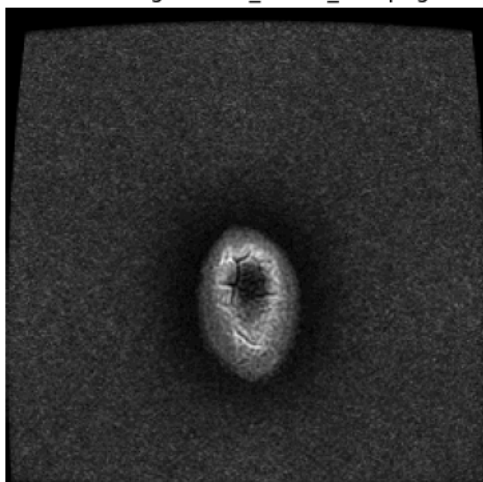
print("Visualizing prepared dataset...")
visualize_dataset(output_dir, num_samples=5)

print("\nDataset is ready to use with the brain tumor segmentation code!")
print(f"Images directory: {os.path.join(output_dir, 'images')}")
print(f"Masks directory: {os.path.join(output_dir, 'masks')}")
print("\nUpdate these paths in the main code:")
print("base_dir = '', output_dir, ''")
print("images_dir = os.path.join(base_dir, 'images')")
print("masks_dir = os.path.join(base_dir, 'masks')")

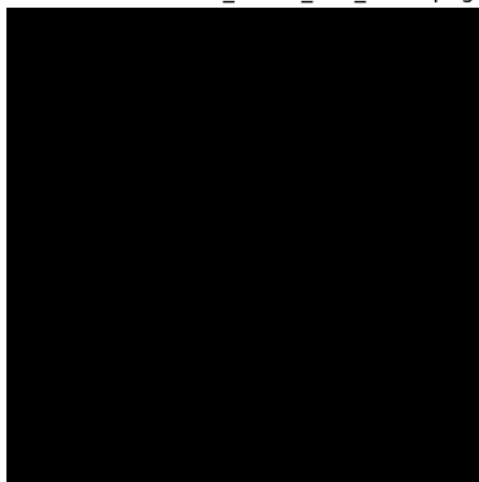
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Google Drive mounted successfully!
Preparing dataset...
Found 1 case directories
Selected 1 cases
Processed 1/10: brain_tumor_000.png
Processed 2/10: brain_tumor_001.png
Processed 3/10: brain_tumor_002.png
Processed 4/10: brain_tumor_003.png
Processed 5/10: brain_tumor_004.png
Processed 6/10: brain_tumor_005.png
Processed 7/10: brain_tumor_006.png
Processed 8/10: brain_tumor_007.png
Processed 9/10: brain_tumor_008.png
Processed 10/10: brain_tumor_009.png
Dataset preparation completed!
Visualizing prepared dataset...
```

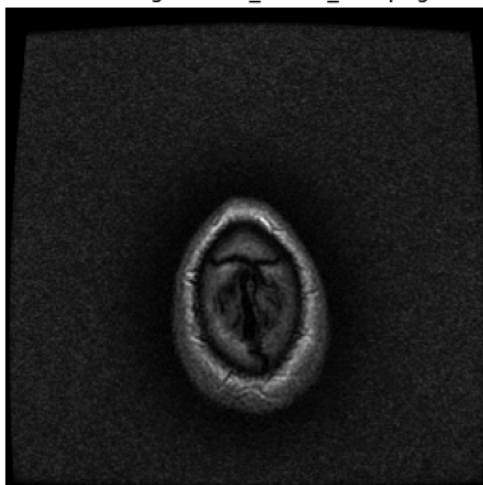
MRI Image: brain_tumor_000.png



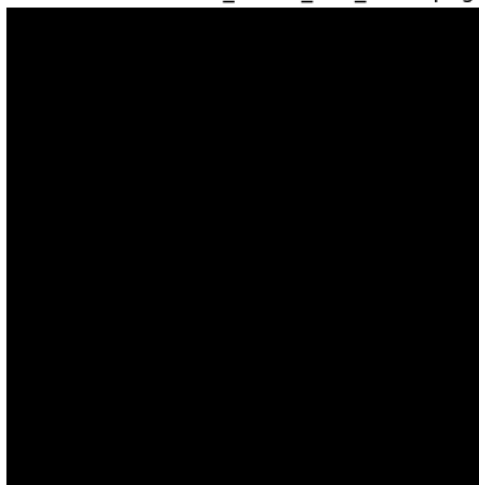
Tumor Mask: brain_tumor_000_mask.png



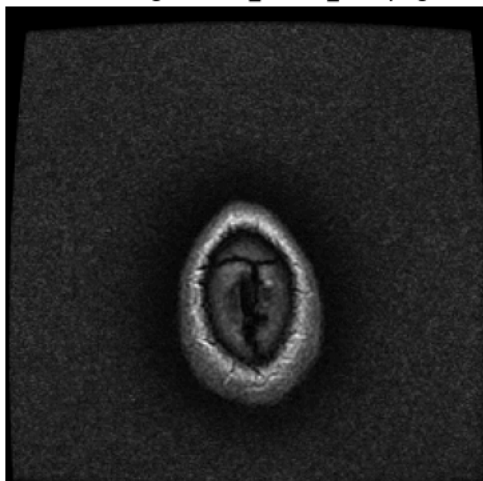
MRI Image: brain_tumor_001.png



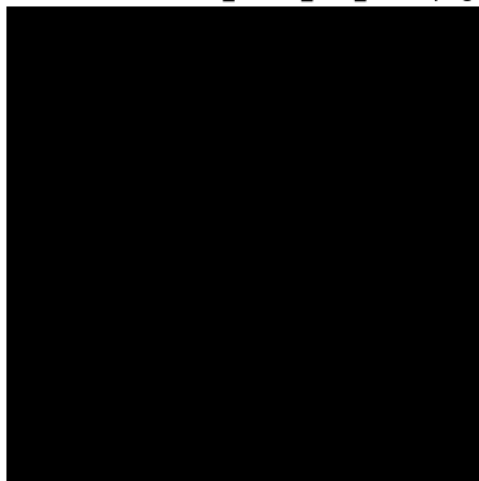
Tumor Mask: brain_tumor_001_mask.png



MRI Image: brain_tumor_002.png



Tumor Mask: brain_tumor_002_mask.png

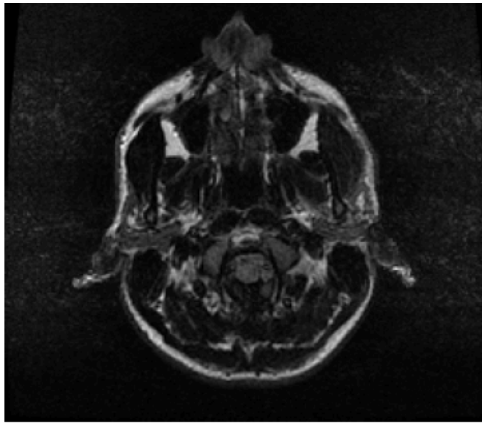


MRI Image: brain_tumor_003.png



Tumor Mask: brain_tumor_003_mask.png

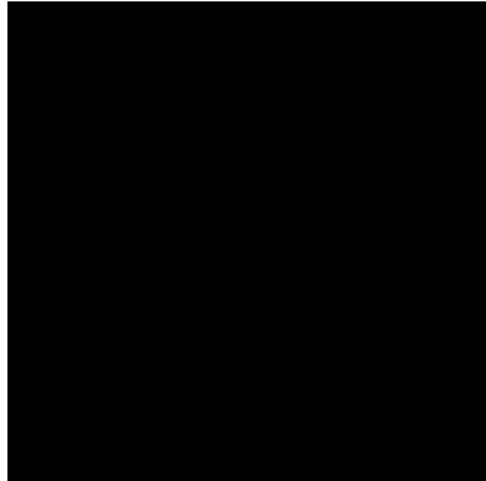
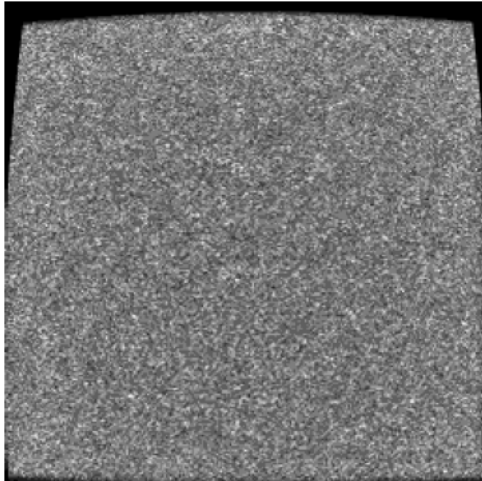




MRI Image: brain_tumor_004.png



Tumor Mask: brain_tumor_004_mask.png



Dataset is ready to use with the brain tumor segmentation code!
Images directory: /content/drive/MyDrive/brain_tumor_dataset/images
Masks directory: /content/drive/MyDrive/brain_tumor_dataset/masks

Update these paths in the main code:
base_dir = ' /content/drive/MyDrive/brain_tumor_dataset '
images_dir = os.path.join(base_dir, 'images')

```

import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, jaccard_score, f1_score, precision_score, recall_score

try:
    drive.mount('/content/drive')
    print("Google Drive mounted successfully!")
except:
    print("Running locally or Drive already mounted")

class BrainTumorSegmentation:
    def __init__(self, base_dir=None):

        self.base_dir = base_dir
        self.images = []
        self.masks = []
        self.processed_images = []
        self.segmented_masks = []
        self.metrics = {}

    def load_dataset(self, images_dir, masks_dir, max_samples=None):

        print("Loading dataset...")

        image_files = sorted(os.listdir(images_dir))

        loaded_count = 0

        for img_file in image_files:
            if not img_file.endswith((''.jpg', '.png', '.jpeg', '.tif')):
                continue

            if '_mask' not in img_file:
                mask_file = img_file.replace('.png', '_mask.png')
                mask_file = mask_file.replace('.tif', '_mask.tif')
                mask_file = mask_file.replace('.jpg', '_mask.jpg')

                image_path = os.path.join(images_dir, img_file)
                mask_path = os.path.join(masks_dir, mask_file)

                if not os.path.exists(mask_path):
                    print(f"Warning: No mask found for {img_file}")
                    continue

                image = cv2.imread(image_path)
                mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

                if image is not None and mask is not None:

                    if len(image.shape) == 3 and image.shape[2] == 3:

                        gray_image = image[:, :, 1]
                    else:
                        gray_image = image.copy()

                    if len(gray_image.shape) == 3:
                        gray_image = cv2.cvtColor(gray_image, cv2.COLOR_BGR2GRAY)

                    _, binary_mask = cv2.threshold(mask, 127, 255, cv2.THRESH_BINARY)

                    self.images.append(gray_image)
                    self.masks.append(binary_mask)

                    loaded_count += 1
                    if max_samples is not None and loaded_count >= max_samples:
                        break

```

```

print(f"Loaded {len(self.images)} images and {len(self.masks)} masks.")

def preprocess_images(self):

    print("Preprocessing images...")
    self.processed_images = []

    for image in self.images:

        if len(image.shape) > 2:
            gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        else:
            gray_image = image.copy()

        if gray_image.max() > 0:
            normalized = ((gray_image - gray_image.min()) /
                           (gray_image.max() - gray_image.min()) * 255).astype(np.uint8)
        else:
            normalized = gray_image

        clahe = cv2.createCLAHE(cliplimit=2.0, tileGridSize=(8, 8))
        enhanced = clahe.apply(normalized)

        blurred = cv2.GaussianBlur(enhanced, (5, 5), 0)

        self.processed_images.append(blurred)

    print(f"Preprocessed {len(self.processed_images)} images.")

def segment_tumors(self, method='watershed'):

    print(f"Segmenting tumors using {method} method...")
    self.segmented_masks = []

    for image in self.processed_images:
        if method == 'threshold':

            _, segmented = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

        elif method == 'watershed':

            _, thresholded = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

            kernel = np.ones((3, 3), np.uint8)
            opening = cv2.morphologyEx(thresholded, cv2.MORPH_OPEN, kernel, iterations=2)

            sure_bg = cv2.dilate(opening, kernel, iterations=3)

            dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
            _, sure_fg = cv2.threshold(dist_transform, 0.7*dist_transform.max(), 255, 0)

            sure_fg = np.uint8(sure_fg)
            unknown = cv2.subtract(sure_bg, sure_fg)

            _, markers = cv2.connectedComponents(sure_fg)

            markers = markers + 1

            markers[unknown == 255] = 0

            markers = cv2.watershed(cv2.cvtColor(image, cv2.COLOR_GRAY2BGR), markers)
            segmented = np.zeros_like(image)
            segmented[markers > 1] = 255

        elif method == 'kmeans':

            image_data = image.reshape((-1, 1))
            image_data = np.float32(image_data)

            criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)

```

```

k = 3
_, labels, centers = cv2.kmeans(image_data, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

centers = np.uint8(centers)
brightest_cluster = np.argmax(centers)

segmented = np.zeros_like(image)
segmented[labels.reshape(image.shape) == brightest_cluster] = 255

else:
    raise ValueError(f"Unknown segmentation method: {method}")

segmented = self.post_process_mask(segmented)
self.segmented_masks.append(segmented)

print(f"Segmented {len(self.segmented_masks)} images.")

def post_process_mask(self, mask):

    if mask.dtype != np.uint8:
        mask = mask.astype(np.uint8)

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    filled_mask = np.zeros_like(mask)
    cv2.drawContours(filled_mask, contours, -1, 255, -1)

    nb_components, output, stats, _ = cv2.connectedComponentsWithStats(filled_mask, connectivity=8)
    sizes = stats[1:, -1]
    min_size = 100

    processed_mask = np.zeros_like(filled_mask)
    for i in range(1, nb_components):
        if sizes[i - 1] >= min_size:
            processed_mask[output == i] = 255

    return processed_mask

def evaluate_segmentation(self, ground_truth_masks=None):

    if ground_truth_masks is None:
        ground_truth_masks = self.masks

    if len(ground_truth_masks) != len(self.segmented_masks):
        raise ValueError("Mismatch between number of ground truth masks and segmented masks")

    dice_scores = []
    jaccard_scores = []
    precision_scores = []
    recall_scores = []

    for gt_mask, pred_mask in zip(ground_truth_masks, self.segmented_masks):

        gt_binary = np.where(gt_mask > 0, 1, 0).flatten()
        pred_binary = np.where(pred_mask > 0, 1, 0).flatten()

        dice = f1_score(gt_binary, pred_binary, zero_division=1)
        dice_scores.append(dice)

        iou = jaccard_score(gt_binary, pred_binary, zero_division=1)
        jaccard_scores.append(iou)

        precision = precision_score(gt_binary, pred_binary, zero_division=1)
        recall = recall_score(gt_binary, pred_binary, zero_division=1)

        precision_scores.append(precision)
        recall_scores.append(recall)

    self.metrics = {
        'dice_coefficient': np.mean(dice_scores),

```

```

        'jaccard_index': np.mean(jaccard_scores),
        'precision': np.mean(precision_scores),
        'recall': np.mean(recall_scores)
    }

    print("Segmentation Evaluation Metrics:")
    print(f" Dice Coefficient (F1-Score): {self.metrics['dice_coefficient']:.4f}")
    print(f" Jaccard Index (IoU): {self.metrics['jaccard_index']:.4f}")
    print(f" Precision: {self.metrics['precision']:.4f}")
    print(f" Recall: {self.metrics['recall']:.4f}")

    return self.metrics

def visualize_results(self, num_samples=5):
    num_samples : int
        Number of samples to visualize

    plt.subplot(num_samples, 3, i * 3 + 1)
    plt.imshow(self.images[i], cmap='gray')
    plt.title(f"Original Image {i+1}")
    plt.axis('off')

    plt.subplot(num_samples, 3, i * 3 + 2)
    plt.imshow(self.masks[i], cmap='gray')
    plt.title(f"Ground Truth Mask {i+1}")
    plt.axis('off')

    plt.subplot(num_samples, 3, i * 3 + 3)
    plt.imshow(self.segmented_masks[i], cmap='gray')
    plt.title(f"Segmented Mask {i+1}")
    plt.axis('off')

    plt.tight_layout()
    plt.show()

def overlay_results(self, num_samples=5):
    num_samples = min(num_samples, len(self.images))

    plt.figure(figsize=(12, 4 * num_samples))

    for i in range(num_samples):
        display_img = cv2.cvtColor(self.images[i], cv2.COLOR_GRAY2RGB)

        plt.subplot(num_samples, 2, i * 2 + 1)

        overlay = display_img.copy()

        green_mask = np.zeros_like(overlay)
        green_mask[:, :, 1] = self.masks[i]

        alpha = 0.5
        cv2.addWeighted(green_mask, alpha, overlay, 1 - alpha, 0, overlay)

        plt.imshow(overlay)
        plt.title(f"Original + Ground Truth {i+1}")
        plt.axis('off')

        plt.subplot(num_samples, 2, i * 2 + 2)

        overlay = display_img.copy()

        red_mask = np.zeros_like(overlay)
        red_mask[:, :, 0] = self.segmented_masks[i] # Red channel

        cv2.addWeighted(red_mask, alpha, overlay, 1 - alpha, 0, overlay)

        plt.imshow(overlay)
        plt.title(f"Original + Segmentation {i+1}")
        plt.axis('off')

```



```
plt.tight_layout()
plt.show()

def run_full_pipeline(self, images_dir, masks_dir, max_samples=None, segmentation_method='watershed'):

    self.load_dataset(images_dir, masks_dir, max_samples)

    self.preprocess_images()

    self.segment_tumors(method=segmentation_method)

    self.evaluate_segmentation()

    self.visualize_results()
    self.overlay_results()

    return self.metrics

if __name__ == "__main__":

    base_dir = '/content/drive/MyDrive/brain_tumor_dataset'
    images_dir = os.path.join(base_dir, 'images')
    masks_dir = os.path.join(base_dir, 'masks')

    tumor_segmentation = BrainTumorSegmentation(base_dir)

    methods = ['threshold', 'watershed', 'kmeans']
    results = {}

    for method in methods:
        print(f"\n{'-'*50}")
        print(f"Running segmentation with {method.upper()} method")
        print(f"{'-'*50}")

    metrics = tumor_segmentation.run_full_pipeline(
```