COMPUTER VISION ASSIGNMENT

Raghuveer Bhat Rajaprakash (ID: 2358410)

I. AIM

This project aims to segment out five tissue layers on ten consecutive 'axial' cross-sections of publicly available MRI of a single human subject so that it can be further analyzed. So we are required to develop 2D as well as 3D segmentation algorithm that will segment out the following regions: Air (class 0), Skin/Scalp (class 1), Skull (class 2), CSF (class 3), Gray Matter (class 4), White Matter (class 5). Also, we need to evaluate the results obtained from 2D and 3D algorithms and conclude with our observation based on the results.

II. METHODOLOGY

A. 2D Segmentation

To solve the 2D segmentation problem, we first need to analyze and identify broadly different regions present in the image. Once this is done, we can go ahead to generate masks to extract these regions. In our problem, we can see three regions that stand out: skin/scalp, skull, and brain matter region. So now we can plan to generate masks for these regions. We have experimented with three methods to get the masks for these regions.

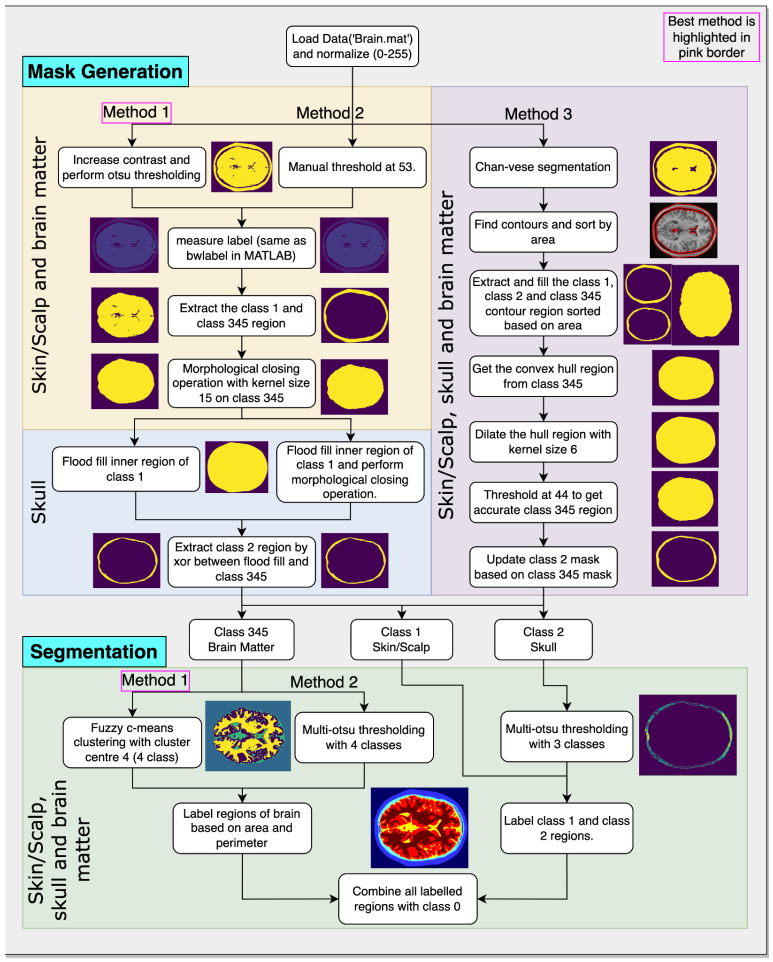
Best method is highlighted in pink border
Load Data('Brain.mat') and normalize (0-255)
Mask Generation
Method 1
Method 2
Method 3
Skin/Scalp and brain matter
Increase contrast and perform otsu thresholding
Manual threshold at 53.
measure label (same as bwlabel in MATLAB)
Extract the class 1 and class 345 region
Morphological closing operation with kernel size 15 on class 345
Skull
Flood fill inner region of class 1
Flood fill inner region of class 1 and perform morphological closing operation.
Extract class 2 region by xor between flood fill and class 345
Skin/Scalp, skull and brain matter
Chan-vese segmentation
Find contours and sort by area
Extract and fill the class 1, class 2 and class 345 contour region sorted based on area
Get the convex hull region from class 345
Dilate the hull region with kernel size 6
Threshold at 44 to get accurate class 345 region
Update class 2 mask based on class 345 mask
Class 345 Brain Matter
Class 1 Skin/Scalp
Class 2 Skull
Segmentation
Method 1
Method 2
Fuzzy c-means clustering with cluster centre 4 (4 class)
Multi-otsu thresholding with 4 classes
Multi-otsu thresholding with 3 classes
Label regions of brain based on area and perimeter
Label class 1 and class 2 regions.
Combine all labelled regions with class 0
Skin/Scalp, skull and brain matter

Figure 1: Flowchart of the 2D segmentation algorithm.

Method 1 and method 2 are similar and rely on thresholding and then labeling connected regions of the image (measure label). Method 3 is different as we use the chan-vese algorithm to get the initial binary mask for the regions and find contours to generate the required masks. Figure 1 'Mask Generation' section gives more details in this regard. Once the masks are generated, we can go to the next stage to segment the regions within the masks. For this, we have used two methods. Method 1 uses Fuzzy c-means clustering, and method 2 uses Multi-otsu thresholding as shown in figure 1 'Segmentation' section. 2D segmentation algorithm runs slice by slice at a time. For simplicity, class 1 indicates skin/scalp mask, class 2 indicates skull mask and class 345 indicates brain matter and CSF mask.

➢ *Mask Generation*

- <u>Method 1 (Best accuracy)</u>
  - At first, we increase the image contrast by a factor of 2.8 and perform otsu thresholding. By increasing the contrast, we get well-defined edges of the region. We also perform the morphological opening operation to remove small bright regions using a disk-shaped kernel of size 1.
  - Then we label all the connected regions using the measure label function. Measure label checks neighbor pixels to tell if the current pixel is connected. If they are connected, they are assigned to the same label; otherwise, a different label is assigned.
  - Extract the class 345 and class 1 region and perform morphological closing operation on the class 345 mask using a disk-shaped kernel of size 2.
  - Next, we take the class 1 mask and flood-fill the region inside the skin/scalp to get the filled mask. Flood-fill function will fill the region defined by their boundaries with the same values essentially filling the hole. Next, take out the class 2 mask by performing xor operation on the fully filled mask and class 345 mask.
- <u>Method 2</u>
  - Method 2 is very similar to method 1, except we use manual thresholding in the first step based on the histogram of pixel intensities. In this work, we have used 53 as the thresholding value after experimentation.
  - Similar to method 1, we do a flood fill operation, but soon after that, we do an additional closing operation with a disk-shaped kernel of size 4 to the fully-filled mask and class 1 mask to plug in small holes in the masks.
- <u>Method 3</u>
  - We first perform the chan-vese algorithm to get the class 1 mask and class 345 mask.
  - Once we get masks, we find contours to the generated mask and sort them based on the area. We will have three contours that we will need class 1 outer and inner contours and class 345 contour. From these contours, extract class 1, class 2, and class 345 region and create individual masks by filling in between contours.
  - For the class 345 (brain matter region) we run into some problems.
    1. The outer edges are not smooth and round.
    2. We seem to miss out on important regions related to CSF since our initial chan-vese couldn't identify those regions due to those regions being on the darker scale. If we tune it to identify those regions, then we risk detecting unwanted portions which might ruin the separateness of the brain matter region.

  To remedy this we have first find the convex hull points of the region which will smoothen out points, thus creating a smooth and almost circular-shaped outer region. Second, we dilate the generated convex hull mask and threshold manually at 44 to perfectly get the class 345 mask.

- We then update the class 2 mask based on the updated class 345 mask, by identifying the intersection of class 2 and class 345 masks and subtracting it from class 2 mask.

By this stage we will have successfully generated all the masks for all the unique regions in the image as shown in Figure 2.

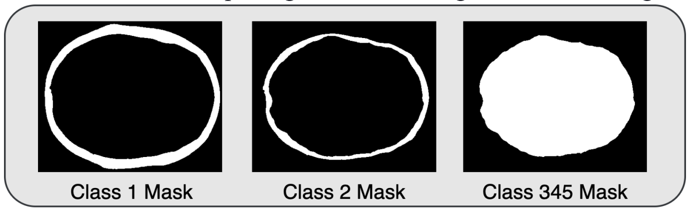

Figure 2: Masks generated for class 1, 2 and 345 region

➢ *Segmentation*
- Method 1 (Best accuracy)
  - We will use the masks generated for class 345 to get the brain matter region of the image and then run fuzzy c-means clustering algorithm with number of centres as four representing three different classes within brain matter region and background class.
  - Once we get all the segmented regions in the brain matter region, we then identify and label all the different regions of the class 345 based on the area and perimeter.
  - For class 2, we use multi-otsu thresholding with three classes to get the skin/scalp region inside class 2.
  - Finally, by combining all the different labelled regions we get the segmented image of the brain as shown in Figure 3.

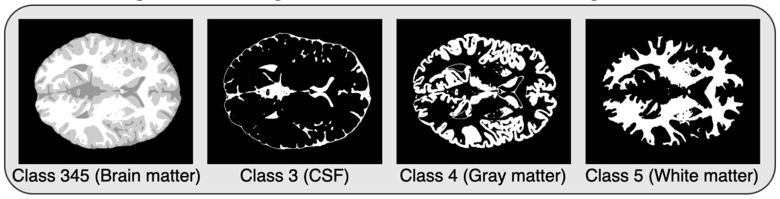

Figure 3: Segmentation result of the brain matter region (class 345)

- Method 2
  - Every step in method 2 is the same as method 1, but instead fuzzy c-means clustering algorithm we use thresholding multi-otsu with number of classes as four.

B. 3D Segmentation

Some major disadvantages of 2D segmentation algorithms are that they do not make utilize the depth aspect of the images, and since they operate slice by slice, sometimes they might be slower. For this reason we have designed a 3D segmentation algorithm which operates on all the slices simultaneously. Figure 4 illustrates the steps in the 3D segmentation algorithm. 3D segmentation algorithm flow is very much similar to the 2D segmentation algorithm.

➢ *Mask Generation*
- Method 1
  - Increase the contrast of the 3D image by a factor of 2.7 and then perform otsu thresholding to separate the background mask and foreground mask.
  - Then perform morphological opening operation to remove small bright marks from the 3D image. For this, we use a custom-built ball-shaped 3D kernel of size 1 where we take the normal ball shaped 3D kernel and elongate it to cover the depth. This will clear most of the bright marks and improve the accuracy. Then we call the measure label to label all the connected regions (in 3D it will consider the depth aspect while checking for neighbour pixel connectivity).

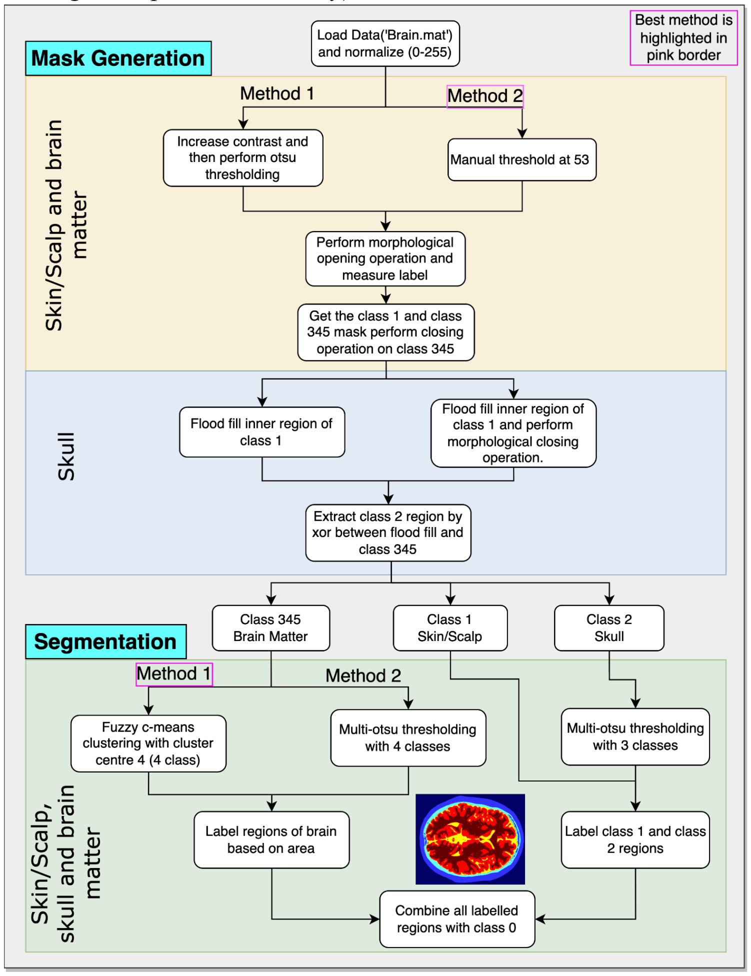
Best method is highlighted in pink border
Mask Generation
Load Data('Brain.mat') and normalize (0-255)
Method 1
Method 2
Skin/Scalp and brain matter
Increase contrast and then perform otsu thresholding
Manual threshold at 53
Perform morphological opening operation and measure label
Get the class 1 and class 345 mask perform closing operation on class 345
Skull
Flood fill inner region of class 1
Flood fill inner region of class 1 and perform morphological closing operation.
Extract class 2 region by xor between flood fill and class 345
Class 345 Brain Matter
Class 1 Skin/Scalp
Class 2 Skull
Segmentation
Method 1
Method 2
Skin/Scalp, skull and brain matter
Fuzzy c-means clustering with cluster centre 4 (4 class)
Multi-otsu thresholding with 4 classes
Multi-otsu thresholding with 3 classes
Label regions of brain based on area
Label class 1 and class 2 regions
Combine all labelled regions with class 0

Figure 4: Flowchart of the 2D segmentation algorithm.

  - Get the class 1 and class 345 regions of the image and perform morphological closing operation on class 345 with ball shaped 3D kernel of size 13 to cover all the holes and cracks.
  - Flood fill the inner region of the class 1 mask which is same as 2D segmentation algorithm except that filling will include depth.
  - Extract the class 2 region by just performing xor operation of flood fill mask and class 345 mask.
- Method 2 (Best accuracy)
  - This is same as method 1 except in the first step we use manual thresholding at 53 to get the foreground/background mask.
  - Also after flood fill operation, we do closing operation with 3D ball-shaped kernel of size 5 on class 1 mask.

➢ *Segmentation*
- Method 1 (Best accuracy)
  - This is same as what we did in 2D segmentation algorithm (i.e use fuzzy c-means clustering with 4 centers) except this is done on all the slices simultaneously. Labelling the region is done only by sorting based on the area for class 345.
- Method 2
  - This is same as what we did in 2D segmentation algorithm (i.e use multi-otsu thresholding with 4 class) except this is done on all the slices simultaneously. Labelling the region is done only by sorting based on the area for class 345.

III. EVALUATION AND COMPARISON

A. Evaluation for 2D and 3D segmentation algorithms

For evaluation we have used three types of metic namely:

- Pixel wise accuracy: It is one of the easiest metric to understand. It indicates the percentage of pixels that are correctly classified. But if there is class imbalance then it is not at all accurate in giving the accuracy for the segmentation. So this metric is ignored in this work.
- Intersection-over-union(IOU): This is one of the most commonly used evaluation criteria for the segmentation tasks. IOU is the area of the overlap between predicted and ground truth segmentation divided by the union of the predicted and ground truth segmentation. The reason why it's so effective is that it tells us how well the two masks are overlapping with each other. Also handles class imbalance problem.
- Structural similarity(SSIM): This metric takes into account the structural as well as perceptual phenomena which include luminance and contrast. In our work, we only use it to get the overall structural accuracy.

Figure 5 and 6 shows the scores for various combination of methodologies. From the table, it is apparent that the fuzzy c-means (segmentation) with otsu thresholding (mask) has the best overall accuracy. In the case of 3D, it is clear that both fuzzy c-means (segmentation) with otsu/manual thresholding (mask) have the best overall accuracy.

		C0	C1	C2	C3	C4	C5	Mean
Mask Method 1 + Segmentation Method 1	Avg. IOU	**0.99**	**0.939**	**0.944**	0.764	**0.91**	0.967	**0.92**
	Avg. SSIM	**0.984**	0.933	**0.985**	0.912	0.903	0.964	**0.947**
Mask Method 2 + Segmentation Method 1	Avg. IOU	0.985	0.923	0.943	0.741	0.907	0.967	0.911
	Avg. SSIM	0.977	0.924	**0.985**	0.905	0.901	0.964	0.943
Mask Method 3 + Segmentation Method 1	Avg. IOU	0.973	0.92	0.928	0.775	0.898	0.957	0.909
	Avg. SSIM	0.963	**0.939**	0.982	0.915	0.898	0.954	0.941
Mask Method 1 + Segmentation Method 2	Avg. IOU	0.99	0.939	**0.944**	0.731	0.907	**0.981**	0.915
	Avg. SSIM	**0.984**	0.933	**0.985**	0.898	**0.904**	0.979	**0.947**
Mask Method 2 + Segmentation Method 2	Avg. IOU	0.985	0.923	0.943	0.708	0.904	**0.981**	0.907
	Avg. SSIM	0.977	0.924	**0.985**	0.89	0.901	**0.98**	0.943
Mask Method 3 + Segmentation Method 2	Avg. IOU	0.973	0.92	0.928	**0.789**	0.872	0.922	0.901
	Avg. SSIM	0.963	**0.939**	0.982	**0.921**	0.868	0.92	0.932

Figure 5: Accuracy scores for various 2D segmentation algorithms

		C0	C1	C2	C3	C4	C5	Mean
Mask Method 1 + Segmentation Method 1	Avg. IOU	0.98	0.92	0.892	0.758	**0.9**	**0.943**	0.898
	Avg. SSIM	**0.976**	0.932	0.975	**0.916**	**0.893**	**0.942**	**0.939**
Mask Method 2 + Segmentation Method 1	Avg. IOU	**0.984**	**0.929**	**0.916**	**0.759**	0.894	**0.943**	**0.904**
	Avg. SSIM	**0.976**	**0.933**	**0.98**	0.912	0.887	**0.942**	0.938
Mask Method 1 + Segmentation Method 2	Avg. IOU	0.984	0.923	0.892	0.727	0.886	**0.943**	0.8925
	Avg. SSIM	0.976	0.932	0.975	0.901	0.88	**0.942**	0.934
Mask Method 2 + Segmentation Method 2	Avg. IOU	0.984	0.929	0.916	0.727	0.879	**0.943**	0.896
	Avg. SSIM	0.976	**0.933**	**0.98**	0.897	0.875	**0.942**	0.933

Figure 6: Accuracy scores for various 3D segmentation algorithms

We can observe overall class 3 accuracy is lower than all other accuracies. This can be attributed to the fact that the number of pixels are less. Furthermore, these pixels are spread out and are very thin in the outer ring region. So a small change in the structure will lead to big accuracy difference. Additionally, fuzzy c-means clustering performs better for class 4 and 5, whereas multi-otsu has better accuracy for class 3. Overall for class 345, fuzzy c-means beats multi-otsu. This is due to the fact that fuzzy c-means assigns soft probability for the cluster memberships so it handles outliers well whereas multi-otsu thresholding might not handle the boundary cases that well. Moreover, one observation we could find with respect to class 3 near the back region of the skull the ground truth mask annotation seems to be little off which might also drastically reduces accuracy of our class 3.

B. Comparison of 2D and 3D segmentation algorithms

Between 2D and 3D segmentation results, we can observe that 2D segmentation outperforms 3D segmentation by a very small margin. This difference arise due to the fact that, in 3D segmentation algorithms, when measure label is called, the neighbour connectivity considers surrounding pixels in depth whereas in 2D it doesn't consider depth. Also we use 3D shaped kernel in morphological operations in case of 3D segmentation which results in slightly varied structure of the mask. But results are fairly comparable. Final segmentation output for both 3D and 2D segmentation is shown in Figure 7.

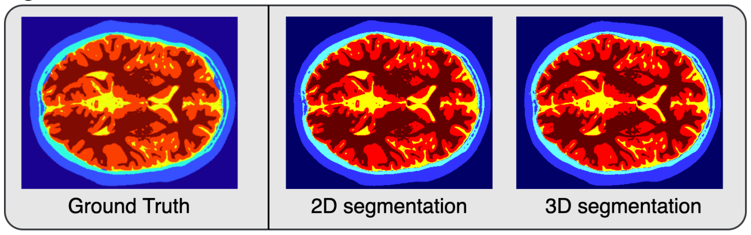

Figure 7: Final segmentation result of 2D and 3D algorithms (Slice0)

IV. CONCLUSION

In this work, we have experimented using a combination of various methodologies. From the best 2D and 3D segmentation algorithms, we can observe that for all the classes except class 3, average IOU scores over all the slices are above 90%. For class 3 however due to the fact that the number of pixels are less and also pixels are very thin in the outer ring region, the accuracy is in the range of 75% to 80%. Also, there is not much difference in the accuracies of 2D and 3D segmentation and if we manually inspect it the results are comparable. Hence we can conclude that the best performing algorithm for both 2D and 3D is otsu thresholding for getting mask and fuzzy c-means clustering for segmentation with average IOU accuracy of **92%** and average SSIM score of **95%** for 2D and **90%** and **94%** in case of 3D. In future work, we can try out other segmentation algorithms like watershed, region growing etc. We can also try out other evaluation metics like f1 score, precision score, and recall score, time taken by the algorithm.

V. APPENDIX

A. Threshold (multi-)otsu

Otsu thresholding is a type of automatic thresholding method where threshold is selected based on minimizing intra-class intensity variance i.e. maximizing inter-class intensity variance iteratively. The steps in calculating the threshold values in terms of class probabilities w, class mean μ and class variance σ are given below:

- Compute the histogram and probabilities at each intensity level.
- Initialize $w(0)$ and $\mu(0)$

- Iterate over all possible threshold values i.e t = 1 to max intensity
    - Update w and μ.
    - Compute σ^2
- Choose the highest σ^2 as the threshold value

Multi-otsu is an extension of otsu thresholding where it can detect several threshold and this is given by the desired number of classes. Some related formulas and implementation can be found under this link.

B. *Fuzzy c-means clustering*

Fuzzy c-means clustering is a soft clustering technique where the datapoint in the dataset belongs to every cluster with some probability. If a data point is closer to the cluster center then it will have a higher membership grade to that cluster. For a far-away cluster, it will have a low membership value to that cluster.

Steps in fuzzy c-means clustering are as follows:

- Randomly initialise cluster center and assign each data point random membership grade for each cluster.
- Iteratively move the cluster center to the right spot and keep updating the membership grade for each data point
- Objective is to minimize the distance between any given data point and cluster center weighted by the membership value of the data point in that cluster.

Algorithm and the objective function is given in this link.

C. *Chan-vese algorithm*

Chan-vese for active contours is a powerful method to segment images and it is capable of segmenting difficult segmentation problems. It is based on Mumford-Shah functional for segmentation. It is basically an energy minimization problem that can be solved using level set formulation. So this energy function is given by the weighted values which corresponds to the sum of differences intensity from average value of intensity outside segmented region, the sum of differences from the average value inside segmented region and a term dependent on the perimeter of the boundary of segmented region. More information about the actual energy equation and additional information is given in this paper.

VI. CODE

Complete working code with implementation and running details is uploaded to github. Please find the complete code in this git link.

Segmentation Code - 2D and 3D

```python
'''
File                : main3d.py
Application         : 2D and 3D brain MRI segmentation (6 classes and 10 slices)
Author              : Raghuveer Bhat R
Last Modified Date  : 13/05/2022
description         : This file is the main entry point for 2D as well as 3D segmentation. Before running this file create "Output1" folder in the current directory.
                     This file requires "Brain.mat" to run it.
Libraries required  : skimage, numpy, matplotlib, scipy, skfuzzy, cv2(opencv), sklearn
Usage               : python3 main3d.py [3D/2D] [fuzzyc/multiotsu (brain segmentation method)] [default/otsu (foreground background segmentation)] [write/show]
'''
import matplotlib.pyplot as plt
import scipy.io
import numpy as np
import cv2
from skimage.filters import threshold_multiotsu
from skimage import measure
import skfuzzy as fuzz
import sys
from skimage.morphology import (erosion, dilation, opening, closing, white_tophat, cube, octahedron, ball, disk)
from brain_regions import BrainRegions3D
from evaluation import *
from utils import *

'''
   Load the "Brain.mat" file and normalize it in the range(0-255)
'''
def load_dataset(file):
   mat = scipy.io.loadmat(file)
   image3d = mat['T1']
   for i in range(0,10):
       image = image3d[:,:,i]
       im = cv2.normalize(image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
       im = np.uint8(im*255)
       image3d[:,:,i] = im
   labels = mat['label']
   return image3d, labels
'''
   Run 3D segmentation algorithm on all the slices simultaneuosly
'''
def segmentation_3D(image3d, labels, segmentation_method="default", segment_brain_method="fuzzyc", write_or_show="show"):
   print("3D segmentation in progress ...")
   # Get all the masks for performing segmentation
   b = BrainRegions3D(image3d,method=segmentation_method)
   print("Regions extracted.")
   # Class1 in class 2 region
   # Applying multi-Otsu threshold for the default value, generating three classes.
```

```python
    thresholds_or2 = threshold_multiotsu(image3d,classes=3)
    # Using the threshold values, we generate the three regions.
    regions_or2 = np.digitize(image3d, bins=thresholds_or2)
    t = b.or2 > 44
    print("Brain region segmentation in progress ...")
    regions_brain = None
    if segment_brain_method == "fuzzyc":
        vec_br = b.br.reshape((1,-1))
        ncenters = 4
        cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        vec_br, ncenters, 2.0, error=0.005, maxiter=1000, init=None)
        cluster_membership = np.argmax(u, axis=0)
        regions_brain = cluster_membership.reshape(b.br.shape).astype('uint8')
    else:
        # Applying multi-Otsu threshold for the default value, generating four classes.
        thresholds_or2 = threshold_multiotsu(b.br,classes=4)
        # Using the threshold values, we generate the three regions.
        regions_brain = np.digitize(b.br, bins=thresholds_or2)
    print("Processing completed.")
    # Class 3, 4 and 5
    region_brain_final = label_regions_brain_3d(regions_brain)
    # Class 0
    final_segmentation = np.zeros(image3d.shape, dtype=np.uint8)
    # Class 1
    final_segmentation[np.where(b.or1_mask == 255)] = 1
    # Class 2
    final_segmentation[np.where(b.or2_mask == 255)] = 2
    # Class 1 inside class 2
    final_segmentation[np.where(t == 1)] = 1
    final_segmentation = final_segmentation | region_brain_final
    # Scores
    iou_scores3d = getIOUScores(labels,final_segmentation)
    mssim_3d = getSSIMScores(labels,final_segmentation,channel_axis=2)
    msqe_3d = getMeanSqError(labels,final_segmentation)
    print("IOU SCORES: ",iou_scores3d)
    print("SSIM: ",mssim_3d)
    show_3D(final_segmentation)

'''
    Run 2D segmentation algorithm on each slice one by one.
'''
def segmentation_2D(image3d, labels, segmentation_method="default", segment_brain_method="fuzzyc", write_or_show="show"):
    iou_scores_2d = []
    mssim_2d = []
    msqe_2d = []
    file_names = ["Slice1 ", "Slice2 ", "Slice3 ", "Slice4 ", "Slice5 ", "Slice6 ", "Slice7 ", "Slice8 ", "Slice9 ", "Slice10 "]
    file_names.append("Average ")
    for idx in range(0,image3d.shape[2]):
        print(f'Processing {file_names[idx]} ...')
        # Get all the masks for performing segmentation
        b = BrainRegions3D(image3d[:,:,idx], type="2D", method=segmentation_method)
        vec_br = b.br.reshape((1,-1))
        regions_brain = None
        if segment_brain_method == "multiotsu":
            # Applying multi-Otsu threshold for the default value, generating
            # four classes.
            b.br = b.increaseContrast(b.br, 0.8)
            thresholds_brain = threshold_multiotsu(b.br,classes=4)
            # Using the threshold values, we generate the three regions.
            regions_brain = np.digitize(b.br, bins=thresholds_brain)
        if segment_brain_method == "fuzzyc":
            ncenters = 4
            cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
            vec_br, ncenters, 2.0, error=0.005, maxiter=1000, init=None)
            cluster_membership = np.argmax(u, axis=0)
            regions_brain = cluster_membership.reshape(b.br.shape).astype('uint8')
        # Segmenting class 1 in class 2 region
        # Applying multi-Otsu threshold for the default value, generating three classes.
        b.or2 = b.increaseContrast(b.or2, 0.048)
        thresholds_or2 = threshold_multiotsu(b.or2,classes=3)
        # Using the threshold values, we generate the three regions.
        regions_or2 = np.digitize(b.or2, bins=thresholds_or2)
        brain_final_segement = label_regions_brain(regions_brain)
        or2_class1_segment = np.zeros(b.or2.shape, dtype=np.uint8)
        regions = measure.regionprops(regions_or2)
        regions = sorted(regions, key=sort_by_area, reverse=True)
        #Get the class 1 segment in or2 region
        npc = np.array(regions[-1].coords)
        or2_class1_segment[(npc[:,0],npc[:,1])] = 1

        region_or1_or2 = b.or1_mask
        region_or1_or2[np.where(region_or1_or2 == 255)] = 1
        region_or1_or2 = region_or1_or2 | b.or2_mask
        region_or1_or2[np.where(region_or1_or2 == 255)] = 2
        region_or1_or2[np.where(regions_or2 == 2)] = 1
        final_segmentation = np.uint8(brain_final_segement) | np.uint8(region_or1_or2)

        # Calculate scores (metrics)
        score = getIOUScores(labels[:,:,idx],final_segmentation)
        iou_scores_2d.append(score.tolist())
        mssim = getSSIMScores(labels[:,:,idx],final_segmentation)
        mssim_2d.append(mssim.tolist())
        mse = getMeanSqError(labels[:,:,idx],final_segmentation)
        msqe_2d.append(mse)

        if write_or_show == "write":
            write_segmented_image(final_segmentation, file_names[idx])
        elif write_or_show == "show":
            print("IOU SCORE: ", score)
            print("SSIM: ", mssim)
            show_image(show_seg_regions(final_segmentation))
    iou_scores_2d = np.array(iou_scores_2d)
    mssim_2d = np.array(mssim_2d)
    msqe_2d = np.array(msqe_2d)
    iou_scores_2d = np.vstack((iou_scores_2d, np.average(iou_scores_2d,axis=0)))
    iou_scores_2d = np.hstack((iou_scores_2d, np.array([np.average(iou_scores_2d,axis=1)]).T))
    mssim_2d = np.vstack((mssim_2d, np.average(mssim_2d,axis=0)))
    mssim_2d = np.hstack((mssim_2d, np.array([np.average(mssim_2d,axis=1)]).T))
    msqe_2d = np.append(msqe_2d, np.average(msqe_2d))
    if write_or_show == "write":
        header = np.array(['File names', '  Class0  ', '  Class1  ', '  Class2  ', '  Class3  ', '  Class4  ', '  Class5  ', 'Overall'])
        write_scores(iou_scores_2d,'Output1/iou_scores.txt',np.array(file_names), header)
    else:
         print("Average IOU scores:\n", iou_scores_2d[-1])
         print("Average SSIM: ", mssim_2d[-1])

if __name__ == "__main__":
    args = sys.argv[1:]
    arglen = len(args)

    type = "2D"
    segment_brain_method = "fuzzyc"
    segmentation_method = "otsu"
    write_or_show = "show"

    # Parse arguments
    if arglen == 1:
        if args[0] == "--help":
            print("python3 main3d.py [3D/2D] [fuzzyc/multiotsu (brain segmentation method)] [default/otsu (foreground background segmentation)] [write/show]")
            exit()
        type = args[0]
    elif arglen == 2:
        type = args[0]
        segment_brain_method = args[1]
    elif arglen == 3:
        type = args[0]
        segment_brain_method = args[1]
        segmentation_method = args[2]
    elif arglen == 4:
        type = args[0]
        segment_brain_method = args[1]
        segmentation_method = args[2]
        write_or_show = args[3]
    else:
        print("Running with defaults - 2D, fuzzyc, otsu and show options")

    #Load Brain.mat datatset
    image3d, labels = load_dataset('Brain.mat')

    if type == "3D":
        segmentation_3D(image3d,labels,segmentation_method,segment_brain_method,write_or_show)
    else:
        segmentation_2D(image3d,labels,segmentation_method,segment_brain_method,write_or_show)
```

Mask generation code 2D and 3D

```
'''
File                 : brain_regions.py
Application          : 2D and 3D brain MRI segmentation (6 classes and 10 slices)
Author               : Raghuveer Bhat R
Last Modified Date   : 13/05/2022
description          : This file is used to create brain mask regions for further
segmentation
'''

# Create custom 3D kernel by elongating it.
def elongate_3d_structure(x):
```

```
    idx = int(x.shape[2]/2)
    first = x[:,:,0:idx+1]
    second = x[:,:,idx+1:x.shape[2]]
    f1 = x[:,:,idx]
    stacked = np.dstack((f1,f1))
    y = np.concatenate((first,stacked),axis=2)
    y = np.concatenate((y,second),axis=2)
    return y

'''
    This class is used for getting all the masks (regions) using method 3 i.e chan-vese
algorithm and finding contours.
'''
```

```python
class BrainRegions:
   def __init__(self, image, method="chanvese", convex_hull=True,
dilate_and_threshold=True):
       self.backup_img = deepcopy(image)
       self.ls = None
       self.or1_mask = None
       self.or2_mask = None
       self.br_mask = None
       self.getDifferentRegionsMask(image, method=method, use_convex_hull=convex_hull,
dilate_and_threshold=dilate_and_threshold)
       self.or1br_mask = self.or1_mask | self.br_mask
       self.or2br_mask = self.or2_mask | self.br_mask
       self.or1or2_mask = self.or1_mask | self.or2_mask
       self.or1 = self.imageFromMask(self.or1_mask)
       self.or2 = self.imageFromMask(self.or2_mask)
       self.br = self.imageFromMask(self.br_mask)
       self.or1br = self.imageFromMask(self.or1br_mask)
       self.or2br = self.imageFromMask(self.or2br_mask)
       self.or1or2 = self.imageFromMask(self.or1or2_mask)


   def getDifferentRegionsMask(self, image, method="chanvese", use_convex_hull=False,
dilate_and_threshold=True):
       if method == "chanvese":
           cv = chan_vese(image, mu=0.40, lambda1=1, lambda2=1, tol=1e-5,
              max_num_iter=2000, dt=0.5, init_level_set="checkerboard",
              extended_output=True)
           self.ls = np.uint8(cv[0]*255)
       elif method == "m-chanvese":
           init_ls = checkerboard_level_set(image.shape, 6)
           self.ls = morphological_chan_vese(image, num_iter=35,
init_level_set=init_ls,
                                       smoothing=3)
           self.ls = np.uint8(self.ls*255)
           #check if background class is inverted
           if self.ls[30][30] == 255.0:
               self.ls = cv2.bitwise_not(self.ls)
       contours, hierarchy = self.findContour(self.ls)
       contours = sorted(contours, key=cv2.contourArea, reverse=True)

       ret, im1 = cv2.threshold(image, 30, 255, cv2.THRESH_BINARY)

       contours_class0, hierarchy_class0 = self.findContour(im1)
       contours_class0 = sorted(contours_class0, key=cv2.contourArea, reverse=True)
       contours[0] = contours_class0[0]

       # getOuter1Region
       segcontour = contours[0:2]
       self.or1_mask = np.zeros((362,434), dtype=np.uint8)
       cv2.fillPoly(self.or1_mask,pts=segcontour,color=255)
       # getOuter2Region
       segcontour = contours[1:3]
       self.or2_mask = np.zeros((362,434), dtype=np.uint8)
       cv2.fillPoly(self.or2_mask,pts=segcontour,color=255)
       # getBrainRegion
       self.br_mask = np.zeros((362,434), dtype=np.uint8)
       cv2.drawContours(self.br_mask, contours, 2, color=255, thickness=cv2.FILLED)

       if use_convex_hull is True:
           # Get the brain segmented coordinates
           generators = contours[2].reshape(contours[2].shape[0],2)
           hull = ConvexHull(points=generators)
           hull_vertices = generators[hull.vertices]
           hull_vertices = np.array([hull_vertices])
           self.br_mask = np.zeros((362,434),dtype=np.uint8)
           cv2.fillPoly(self.br_mask, hull_vertices, 255)

       if dilate_and_threshold is True:
           footprint_d = disk(6)
           footprint_o = disk(2)
           footprint_c = disk(4)
           dilated = dilation(self.br_mask,footprint_d)
           image = self.imageFromMask(dilated)
           ret, im = cv2.threshold(image, 44, 255, cv2.THRESH_BINARY)
           opened = opening(im, footprint_o)
           closed = closing(opened, footprint_c)
           contours, hierarchy = self.findContour(np.uint8(opened))
           contours = sorted(contours, key=cv2.contourArea, reverse=True)
           self.br_mask = np.zeros((362,434), dtype=np.uint8)
           cv2.drawContours(self.br_mask, contours, 0, color=255, thickness=cv2.FILLED)
       # Update class 2 mask
       self.or2_mask = (self.or2_mask & self.br_mask) ^ self.or2_mask

   def findContour(self, image):
       ret, im = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
       return cv2.findContours(image, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

   def imageFromMask(self, mask):
       im = np.zeros((362,434))
       for i in range(0,362):
           for j in range(0,434):
               if mask[i][j] == 255.0:
                   im[i][j] = self.backup_img[i][j]
       return im

   def increaseContrast(self, image, val):
       im = image
       for i in range(0, image.shape[0]):
           for j in range(0, image.shape[1]):
               if im[i][j] * val <= 255:
                   im[i][j] = np.uint8(im[i][j] * val)
               else:
                   im[i][j] = 255
       return im


'''
   This class is used for getting all the masks (regions) using method 1 and 2 for both
2D and 3D type.
'''
class BrainRegions3D:
   def __init__(self, image3d, type="3D", method="default"):
       self.backup_img = deepcopy(image3d)
       self.or1_mask = None
       self.or2_mask = None
       self.br_mask = None
       self.getdifferentRegions(image3d, type, method)

   def increaseContrast3D(self, image, val):
       im = deepcopy(image)
       for i in range(0, image.shape[0]):
           for j in range(0, image.shape[1]):
               for k in range(0, image.shape[2]):
                   if im[i][j][k] * val <= 255:
                       im[i][j][k] = np.uint8(im[i][j][k] * val)
                   else:
                       im[i][j] = 255
       return im

   def increaseContrast(self, image, val):
       im = deepcopy(image)
       for i in range(0, image.shape[0]):
           for j in range(0, image.shape[1]):
               if im[i][j] * val <= 255:
                   im[i][j] = np.uint8(im[i][j] * val)
               else:
                   im[i][j] = 255
       return im

   def getdifferentRegions(self, image3d, type="3D", method="default"):
       footprint_2 = None
       footprint_7 = None
       if type == "3D":
           if method == "default":
               footprint_2 = ball(2)
               footprint_3 = elongate_3d_structure(ball(1))
               footprint_13 = ball(13)
           else:
               footprint_2 = ball(5)
               footprint_3 = elongate_3d_structure(ball(1))
               footprint_13 = ball(13)
       else:
           if method == "default":
               footprint_2 = disk(2)
               footprint_13 = disk(13)
               footprint_3 = disk(1)
           else:
               footprint_2 = disk(4)
               footprint_13 = disk(15)
               footprint_3 = disk(1)

       if method != "default":
           if type == "3D":
               # Increase the contrast of the image
               high_contrast_image = self.increaseContrast3D(image3d, 2.7)
           else:
               high_contrast_image = self.increaseContrast(image3d, 2.8)
           # Applying Otsu threshold
           thresholds_or2 = threshold_otsu(high_contrast_image)
           # Using the threshold values, we generate the foreground mask
           imt_53 = high_contrast_image > thresholds_or2
```

```
            imt_53 = opening(imt_53, footprint_3)
        else:
            imt_53 = image3d > 53
            imt_53 = opening(imt_53, footprint_3)

        # Measure label function to label all the connected regions
        l53 = label(imt_53)

        self.full_mask = np.zeros(image3d.shape, dtype=np.uint8)
        self.full_mask[np.where(l53 == 1)] = 255

        self.or1_mask = np.zeros(image3d.shape, dtype=np.uint8)
        self.or1_mask[np.where(l53 == 1)] = 255

        self.br_mask = np.zeros(image3d.shape, dtype=np.uint8)
        self.br_mask[np.where(l53 == 2)] = 255

        if type == "3D":
            seed_point = (np.uint8(image3d.shape[0]/2),np.uint8(image3d.shape[1]/2),0)
        else:
            seed_point = (np.uint8(image3d.shape[0]/2),np.uint8(image3d.shape[1]/2))
        # Flood fill the inner region of class 1 mask
        flood_fill(self.full_mask, seed_point, np.uint8(255), in_place=True)

        if method == "default":
            self.full_mask = closing(self.full_mask, footprint_2)
            self.or1_mask = closing(self.or1_mask, footprint_2)
        self.full = np.uint8(self.full_mask/255 * image3d)
        # Masks
        self.br_mask = closing(self.br_mask, footprint_13)
        self.or1br_mask = self.or1_mask | self.br_mask
        self.or2_mask = self.or1br_mask ^ self.full_mask
        # Regions from the mask
        self.br = np.uint8(self.br_mask * image3d)
        self.or1br = np.uint8(self.or1br_mask * image3d)
        self.or2 = np.uint8(self.or2_mask/255 * image3d)
        self.or1 = np.uint8(self.or1_mask * image3d)
```

Evaluation code

```
'''
File                : evaluation.py
Application         : 2D and 3D brain MRI segmentation (6 classes and 10 slices)
Author              : Raghuveer Bhat R
Last Modified Date  : 13/05/2022
description         : This file is used implement the evaluation metric like IOU
scores, SSIM scores and pixel accuracy score
'''
import sklearn.metrics as metrics
from skimage.metrics import structural_similarity as ssim
from skimage.metrics import mean_squared_error
import numpy as np

def getIOUScores(y_true, y_pred):
    y_pred = y_pred.flatten()
    y_true = y_true.flatten()
    x = metrics.jaccard_score(y_true, y_pred, average=None)
    return x

def getSSIMScores(im1, im2, channel_axis=None, classes=6):
    scores = []
    for i in range(0,classes):
        c_g = np.full(im1.shape, 10,dtype=np.uint8)
        c_r = np.full(im2.shape, 10,dtype=np.uint8)
        c_g[np.where(im1 == i)] = i
        c_r[np.where(im2 == i)] = i
        x = ssim(c_g, c_r, data_range=c_g.max() - c_r.min(), channel_axis=channel_axis)
        scores.append(x)
    # x = ssim(im1, im2, data_range=im2.max() - im2.min(), channel_axis=channel_axis)
    return np.array(scores)

def getMeanSqError(im1, im2):
    x = mean_squared_error(im1, im2)
    return x
```

Utility code

```
# Utility function for helping in labelling regions of the brain (class 345)

def sort_by_perimeter(s):
    return s.perimeter_crofton

def sort_by_area(s):
    return s.area

def label_regions_brain(regions_brain):
    brain_final_segement = np.zeros((362,434))
    regions = measure.regionprops(regions_brain)
    regions = sorted(regions, key=sort_by_area)
    if regions[-1].area>70000:
        # Background class detected so making it all 0s and running regionprops again
        regions_brain[np.where(regions_brain == 0)] = 5
        regions_brain[np.where(regions_brain == regions[-1].label)] = 0
        regions = measure.regionprops(regions_brain)
        regions = sorted(regions, key=sort_by_area)
    regions[1:] = sorted(regions[1:], key=sort_by_perimeter)

    for idx, region in enumerate(regions):
        for i,j in region.coords:
            brain_final_segement[i][j] = idx+1

    brain_final_segement[np.where(brain_final_segement == 3)] = 4
    brain_final_segement[np.where(brain_final_segement == 2)] = 5
    brain_final_segement[np.where(brain_final_segement == 1)] = 3

    return brain_final_segement


def label_regions_brain_3d(regions_brain):
    brain_final_segement = np.zeros(regions_brain.shape, dtype=np.uint8)
    regions = measure.regionprops(regions_brain)
    regions = sorted(regions, key=sort_by_area)
    if regions[-1].area>700000:
        # Background class detected so making it all 0s and running regionprops again
        regions_brain[np.where(regions_brain == 0)] = 8
        regions_brain[np.where(regions_brain == regions[-1].label)] = 0
        regions = measure.regionprops(regions_brain)
        regions = sorted(regions, key=sort_by_area)

    for idx, region in enumerate(regions):
        for i,j,k in region.coords:
            brain_final_segement[i][j][k] = idx+6

    brain_final_segement[np.where(brain_final_segement == 6)] = 3
    brain_final_segement[np.where(brain_final_segement == 7)] = 5
    brain_final_segement[np.where(brain_final_segement == 8)] = 4
    return brain_final_segement
```

—---END---