I. AIM

The objective is to design and develop an 2D and 3D image processing algorithm that will segment a consecutive MRI axial cross section. The classes that are needed to segment are : Air (class0), Skin/Scalp(class1), Skull(class2), CSF(class3), Gray Matter(class4) and White Matter(class5). Analyze the outcome between 2D and 3D, and conclude the observations with appropriate reason.

II. METHODOLOGY

A. 2D Segmentation

Observing 2D slices, in many places the pixel properties across the different classes are overlapping, it's difficult to use one common method (like manual thresholding) to find a complete solution. Along with the pixel value, spatial information should also be considered to get the accurate results. So to solve the problem effciently, here the problem has been broken down into two sections. In stage-I, two methods were tried to separate the class 0-2 from class 3-5 and finally segmenting the class 0,1,2. In stage-II, two methods were tried to segment class 3,4,5. Below flow charts demonstrate the methods that are tried and which method works best for which class.

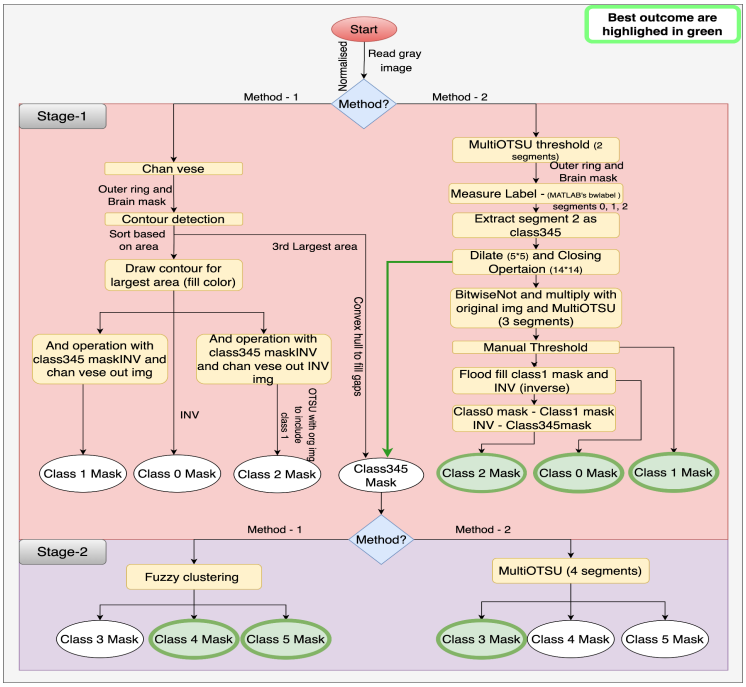


Figure 1: Flow chart demonstrating how the 2D segmentation problem is being handled and the method works best for respective classes (highlighted in green).

Implementation Details:

- *Programing language : Python*
- *Libraries/frameworks : Opencv, scikit, numpy, matplotlib*

Here for easier implementation purposes, the MRI image slices are normalized to value 0-255. As fig 1 describes, the flow is:

→ *In Stage1- method 1*:

Here the aim is to separate class 0,1,2 from class 3,4,5 and accurately segment the class 0,1,2. For that,

- ➢ In loop read input gray image & process through chan vese (active contour - similar to what we learnt in class) method to give segmented outer circle(equivalent to class-1) and inner matter mask (class-345) as output - as shown in fig(2).
- ➢ On the output, we detect contour (opencv) and based on contour area we separate the outer ring from inner matter.
- ➢ With this, now we have separated class 3,4,5 mask and after careful observation, the smooth boundaries(filled) were required to get better accuracy on class 3,4,5. So to get that, a convex hull(opencv) method is used on contours to get a smooth bordered inner matter mask (as shown in fig 2).
- ➢ And for class 0, 1, 2, using inverted inner matter mask, and the mask generated from outer ring (by filling largest contour) and background mask (inverse of outer ring mask), as shown in the flow chart, following the operations we can obtain the segmentation.

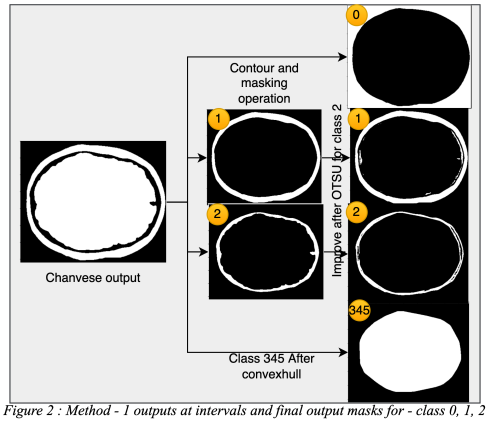


Figure 2 : Method - 1 outputs at intervals and final output masks for - class 0, 1, 2 and 345
(Orange circle represents the class of each mask)

- ➢ Observing fig 2, class 1, 2 could be improved, because in reality within class 2, few areas are occupied by class 1. So to correct that we use OTSU thresholding on the region of class2 mask of original image to get extra parts of class1.

→ *In Stage 1 - method 2*:

With the same aim of seperating class 0, 1, 2 from class 3-5 another method is tried. The reason behind trying this method is, in method 1, in the later stage we had to make some corrections on class 1 and 2 and also, the execution time for chan vese is quite high because of its iterative internal logics. Considering these disadvantages, chan vese - opencv contour's work is kind of replaced by MultiOTSU thresholding, floodfill(similar to Imfill in MATLAB) and MeasureLabel techniques which gives better results than method 1 without any later stage corrections. This method is faster than method 1. Since Measure label and multiOTSU isn't taught in lecture videos, here we understand its working principle at a high level.

** MultiOTSU (for detail theory refer to link):*

Basically the multiOTSU works on the principle of finding histogram and getting threshold, steps are : Assume that we need to segment for n classes. Compute histogram and compute all possible combinations of n-1 thresholds. Iterate over all possible combinations to Compute mean, variance & weight of each class and Compute between class variance. Finally select the set of thresholds that have maximum between class variance.

** MeasureLabel:*

This works based on region growing principle, for instance in 2D, considering 3*3 kernel, center pixel (255) is connective if at least one pixel in the surrounding is connected(255). In 3D, the depth will also be considered (top channel,current and bottom- 26 pixel check).

Steps in stage 1 method 2:

- ➢ Input gray image is processed through the MultiOtsu threshold to give 2 segments (outer ring and inner matter mask) as shown in fig 3.

➢ With this, a measure label algorithm is executed to segment/separate the outer ring from the inner matter mask. The working principle of the algorithm is, based on pixel connectivity, the algorithm creates the segment. In our case, the matter mask is not connected to the outer ring so the segmentation is done efficiently.

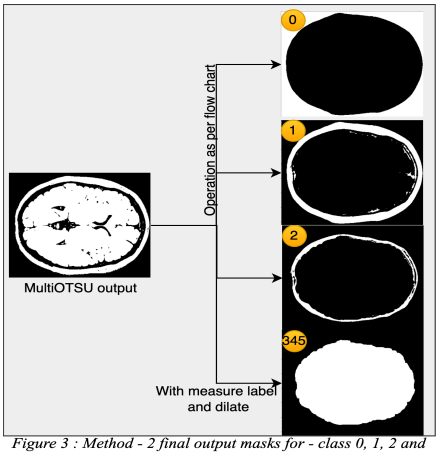


Figure 3 : Method - 2 final output masks for - class 0, 1, 2 and 345
(Orange circle represents the class of each mask)

➢ Once the inner matter mask is separated (class 3-5), we dilate the mask to extend the border and do a closing operation to close inner/border cracks. This is done because few regions of class 3 were missing at the periphery. To cover those pixels, this is an important step which will boost the accuracy of class 3.

➢ And after obtaining the inner matter mask, inverse of that is multiplied with the original image and processed via MultiOTSU to obtain 3 segments. With these segments, following as per flow chart (fig 1) - normal thresholding and masking would give us class 0, 1, 2 masks as shown in fig 3.

Note: Now we have an inner matter mask (class 3,4,5 mask). With this we need to segment class 3, 4, 5 in stage 2. Normal thresholding would also give good results, but considering that manual thresholding might not be the best approach (cause parameters might not work on new data), fuzzy clustering and multi otsu methods are tried.

Fuzzy clustering (C- means) - link :

It is a soft clustering technique similar to k-means, however the membership function is utilized to cluster the pixels here. In this case, one pixel does not belong entirely to one cluster; rather, the likelihood that a pixel is allocated to a certain cluster is determined. Fuzzy is preferred as a practice for most medical image segmentation. The operating idea is that the cluster centers are assigned randomly in the beginning, and fuzzy membership and fuzzy centers are determined repeatedly using corresponding formulas detailed in link.

→ *In Stage 2 - method 1*:

Here we multiply the original image with the inner matter mask and process Fuzzy clustering algorithm to get the output. To get an inner matter mask, for instance we use method 2 of stage 1 here.

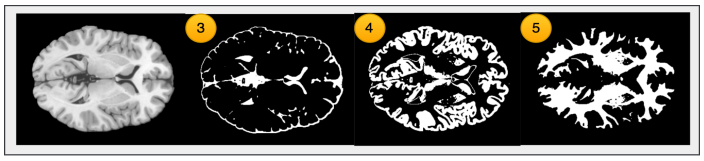


Figure 4 : Fuzzy clustering algorithm output
(Orange circle represents the class of each mask)

→ *In Stage 2 - method 2*:

Here we multiply the original image with the inner matter mask and do the MultiOTSU thresholding to get the output.

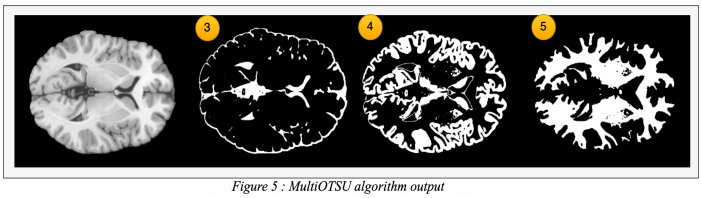


Figure 5 : MultiOTSU algorithm output
(Orange circle represents the class of each mask)

2DSegmentation - Evaluation/Results:

To evaluate the outcome of our experiments, IOU validation is used. Mainly for tuning parameters and select algorithms we use IOU, but also to see structural similarity on the finalized algorithm we did the implementation of SSIM. In IOU, the amount of overlapping between the groundtruth & output is checked. The reason behind using this validation logic is, it considers only foreground pixels and it checks for overlapping, so class imbalance doesn't come into picture while evaluating each mask. We shall check the IOU that we are getting for each stage and method that we did. (100% IOU → target)

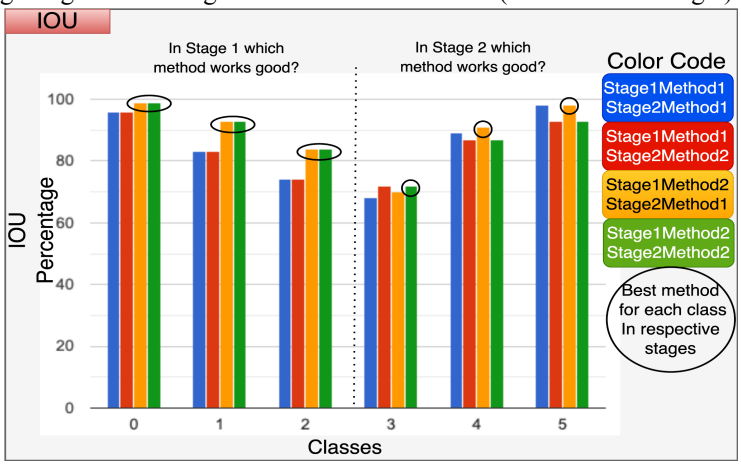


Figure 6 : IOU percentage - Checked on all combinations (all methods and all stages).

From the experimental results we can understand that, For class 0, 1, and 2 the mask we are obtaining from method 2 in stage 1 shows the best results. And for class 3 looks like there is very slightly higher accuracy obtained from MultiOTSU and for class 4 and 5 fuzzy clustering gives better accuracy. This could be because, for class 4 and 5, using fuzzy, the outliers were handled well. Which means, we can see at the center portion of the original image, there are few pixels which have overlapping properties between class 4 and 5. Clustering algorithm after tuned number of iterations it's able to clearly classify between 4 and 5. When it comes to multi otsu, the threshold value which is obtained through histogram might not be working well for those corner cases. And for class 3, both fuzzy and otsu works well, but seems like slightly higher accuracy with multi otsu, might be more pixels are clearly distinguishable from threshold itself.

2D Segmentation - Evaluation conclusion:

From these results, we can combine the techniques to get the best working algorithm to segment the 2D image. That is, we choose method 2 in stage 1 for class 0-2 and method 2 in stage 2 for class 3 and method 1 in stage 2 for class 4,5 (as highlighted in green in fig 1) and call it 2DFuzzyMultiOTSU. Thus, the overall average IOU obtained for 2D across all images is shown in below table and overall accuracy combined considering all images & all classes is 90%. Also, structural similarities between ground truth & output is 93% for 2DFuzzyMultiOTSU, considering all images & all classes combined.

	C0	C1	C2	C3	C4	C5	Overall
IOU%	99.02	93.43	84.50	72.50	92.32	98.88	~90%
SSIM	99.05	94.98	95.89	88.90	90.54	97.23	~93%

Table 1: Representing IOU and SSIM across all images classwise for best 2D algorithm - 2DFuzzyMultiOTSU

B. 3D Segmentation:

Implementation Details:

We shall implement the same logic of 2DFuzzyMultiOTSU to process all channel simultaneously(since functions used can process on multichannel data) and also if we combine stage1 method 2 and stage 2 method 2 (as per fig 1), we get a new 2D algorithm, that we call as 2DMultiOTSU. These two algorithms we implement for 3D data. *Here the aim is to process all slices together to segment* and the functions which are used in 3D will consider multichannel.

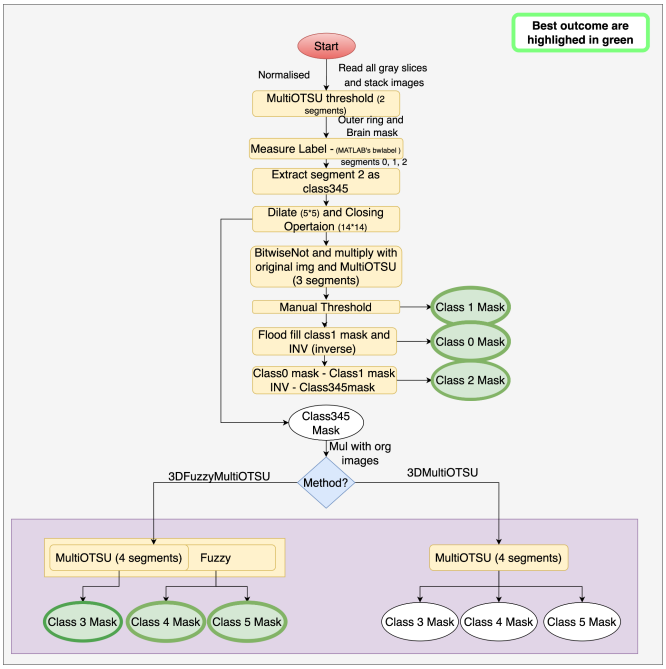


Figure 7: Flow chart demonstrating how the 3D segmentation problem is being handled and the method works best for respective classes (highlighted in green).

Basically, here we tried two methods to segment 3D data. One of them is implementation of the same logic of 2D best algorithm (2DFuzzyMultiotsu). This we call it 3DFuzzyMultiOTSU and the reason for trying this is, as we have seen in 2D, this technique works very well. Also, if we observe in 2DMultiOTSU, the results are not very different. It's just 3% - 4% difference in class 4 and 5 when compared to 2DFuzzyMultiOTSU, so that is why we have experimented with 3DMultiOTSU as the 2nd method. As per the flow chart (fig 7) , all the 2D slices are stacked and processed simultaneously, such that functions like fuzzy, multiotsu will give the generalized output considering all channels. Also one thing to note here is, in closing the depth of kernel (square kernel) will be equivalent to depth of input channel while processing internally.

3D Segmentation - Evaluation/Results:

Here as well to evaluate between two different 3D algorithms we use IOU validation strategy and finally we see SSIM. From table 2 there is a clear pattern as seen in 2D, for the class 3 MultiOTSU works well whereas for the class 4 and 5, fuzzy works well. So with this we can tell that using combined techniques is giving better results rather than using standalone fuzzy or multiOTSU methods.

IOU%	C0	C1	C2	C3	C4	C5
3DFuzzyMultiOtsu	99.15	93.73	83.78	73.01	90.66	96.18
3DMultiOtsu	99.11	93.65	83.98	73.23	87.34	92.82

Table 2: IOU across all images classwise for 3D methods

3D Segmentation - Evaluation conclusion:

We finalize 3DFuzzyMultiOTSU works well for simultaneous processing of all slices. The reason is the same as 2D, that due to overlapping properties of class 4,5 fuzzy works better for class 4 and 5, and for the rest of the classes Multi Otsu seems pretty good. The overall IOU is 89% for 3D across all the classes and all the images. Also, SSIM scores are: class0 - 98%, class1-95%, class2-95%, class3-89%, class4-91%, class5-93%, with overall 93%.

III. COMPARISON BETWEEN 2D & 3D

Knowing the 2D best algorithm - 2DFuzzyMultiOTSU and 3D best algorithm - 3DFuzzyMultiOTSU, comparing between - we cannot see much difference in its overall accuracy ***(IOU, 2D - 90%, 3D - 89%).*** In a few classes like 0 ,1, 2 we use MultiOTSU in both 2D and 3D. Considering the threshold value which is being calculated in multi otsu uses histogram, it is expected to give similar results.

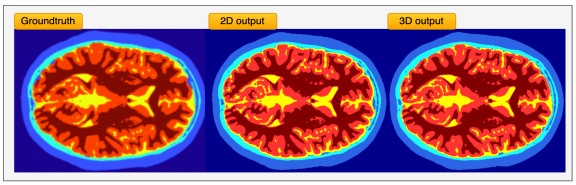


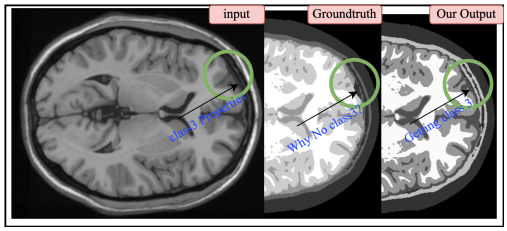
Figure 8: Presenting the sample output of 2D and 3D algorithm

Because slice to slice there is no problem of illumination difference in MRI(pixel value will be almost the same). We can expect to get the same threshold value in both 2D and 3D. And coming to class 3,4,5, the reason for the slight difference in accuracy is that, initially to extract the mask, we use a measure label function to find connectivity. In 2D to consider the pixel is connective, surrounding 8 pixels has to be checked, but in 3D, due to depth, pixels have to be compared across 26 surrounding pixels (with 3*3 kernel - height, width and depth) which leads to a slightly different output mask. From this we miss out/ gain a few pixels in the mask. However, conclusion can be made that both 2D and 3D are working almost similarly, having just 1% overall accuracy difference. In fig 8, a sample colored output is shown(for 3D just one slice is picked for display purpose).

IV. CONCLUSION

Considering both 2D and 3D, very good results are observed in Class 0,1,2,4,5. And Slight lower accuracy for class 3 (~73%). This is because there is a very thin layer of pixels at the periphery and the number of pixels in class 3 is also less. A very little change/off will give a big difference in the accuracy. However, considering the overall result with IOU validation, both 2D and 3D algorithms work really well. Finally we conclude that after trying out different method for 2D, we finalize **2DFuzzyMultiOtsu** method works best with the overall **IOU ~90% and SSIM ~93%** , and **3DFuzzyMultiOtsu** for 3D with **IOU ~89% and SSIM ~93%**.

Note - Additional Information:



This section is just for record purposes for future work as reference. There are few places in specifically class 3 where there is an ambiguity in the ground truth annotations. That is also a definite reason for not getting higher accuracy in class3. Not sure about the ambiguity in the ground truth annotation , required verification before reopening the task for future work.
Also, though we implemented 2 validation methods IOU and SSIM, we used only IOU validation to tune the parameters and get the results(which was sufficient enough). In the future, SSIM, F1-score,dice loss can also be considered to verify and tune the parameters.

V. CODE

Since we are supposed to upload the code within 5 pages, the code in this document has only an optimal 2D algorithm(2DFuzzyMultiOTSU) and optimal 3D algorithm (3DFuzzyMultiOTSU) as Final.py. Also this document has other necessary files like: Finalcluster.py - for performing fuzzy clustering , threshold.py - for performing multiOTSU thresholding ,validation.py - for checking the IOU. Complete code which involves other methods that are worked out, other validation techniques and data, best algorithms output images everything is uploaded in git for detailed reference. Please find the complete code in this git link.

Segmentation Code - 2D and 3D

```
'''
File                            : Final.py
Application                     : 2D and 3D MRI segmentation
Author                          : Santhosh Holla Prakash
Creation Date                   : 13/05/2022
description                     : This .py file will read the input MRI images (which is normalized to 0-255) and process it to provide the segmented image.
Usage                           : python3 Finay.py <input_path> <output_path>
'''

#importing required supporting files and libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
import sys
from Finalcluster import *
from threshold import *
from skimage import measure
from skimage.segmentation import (checkerboard_level_set,chan_vese)
from skimage.filters import threshold_multiotsu

class Main:
        '''
        This class is responsible for processing the 2D and 3D MRI image,based on the method that is configured in constructor and output the segmented image
        '''
        def __init__(self):
                '''
                Constructor to initialize the input path, configure the method to be experimented
                '''
                self.files = sorted(os.listdir(sys.argv[1]))
                self.class12Extraction = "multiotsu_measureLable_dilate" # for stage 1 can specify chanvese_contour, but due to space issues removed. Refer git for full
                self.matterMethod = "fuzzymultiotsu"  #can specify fuzzy or multiotsu for stage 2
                self.type= "3D" #2D or 3D data processing
                if self.type == "2D":
                        self.custom2D()
                else:
                        self.custom3D()

        def getMask2D(self,img):
                '''
                Function to does the operation similar to opencv drawcontour, it with specified color
                Arguments -
                        img      : A image with a closed boundary
                Returns -
                        im_out   : A image with a filled closed boundary with specified color
                '''
                im_floodfill = img.copy()
                h, w = img.shape[:2]
                mask = np.zeros((h+2, w+2), np.uint8)
                cv2.floodFill(im_floodfill, mask, (0,0), 255)
                im_floodfill_inv = cv2.bitwise_not(im_floodfill)
                im_out = (img | im_floodfill_inv)
                return np.asarray(im_out)

        def getMask3D(self,img):
                '''
                Function to does the operation similar to opencv drawcontour, it with specified color
                Arguments -
                        img      : A image with a closed boundary
```

```
                Returns -
                        im_out : A image with a filled closed boundary with specified color
                '''
                im_out = []
                for i in range(0,10):
                        im_floodfill = img[i].copy()
                        h, w = img[i].shape[:2]
                        mask = np.zeros((h+2, w+2), np.uint8)
                        cv2.floodFill(im_floodfill, mask, (0,0), 255)
                        im_floodfill_inv = cv2.bitwise_not(im_floodfill)
                        im_out.append(img[i] | im_floodfill_inv)
                return np.asarray(im_out)

        def segmentInnerPart(self,segmentedImg3,img):
                '''
                This function performs the segmentation of inner matter mask as per the configured method in init (due to space issue removed other methods, refer git)
                Arguments -
                        segmentedImg3      : Mask of the inner matter - inner brain region
                        img                : original grayscale image / image stack if 3D
                Returns -
                        clustImg           : Inner brain region segmented images
                '''
                if self.matterMethod == "fuzzymultiotsu":
                        clust1 = fuzzyCluster(segmentedImg3*img,4)
                        if self.type == "2D":
                                clustImg1 = clust1.cluster()
                        else:
                                clustImg1 = clust1.cluster3D()
                        clust2 = Threshold(np.uint8(segmentedImg3*img)*255)
                        clustImg2 = clust2.multiOtsu()
                        clustImg = clust2.handle(clustImg2,clustImg1)
                        return clustImg

        def otsuMeasureLableDilate(self,img):
                '''
                This function separate the inner matter from the outer ring using multi otsu method and measure label connective detection method and segments class 012
                Arguments -
                        img                     : original grayscale image
                Returns -
                        segmentedImg1           : class 1 mask
                        segmentedImg2           : class 2 mask
                        segmentedImg3           : inner matter mask - class345
                        segmentedImg3ab         : For boosting accuracy of class 3, extra periphery region mask
                '''
                thresholds_brain = threshold_multiotsu(img,classes=2)
                regions_brain = np.digitize(img, bins=thresholds_brain)
                regions_brain = np.uint8(regions_brain*255)
                labeledImage = measure.label(regions_brain)
                class345 = np.zeros(labeledImage.shape)
                class345[np.where(labeledImage==2)]=255
                class345 = np.uint8(class345)
                matterMask = self.getMask2D(class345)
                kernel = np.ones((5,5), np.uint8)
                matterMask = cv2.dilate(matterMask, kernel)
                class012 = cv2.bitwise_not(matterMask)/255
                class012  = class012 * img
                thresholds_brain = threshold_multiotsu(class012,classes=3)
                regions_brain = np.digitize(class012, bins=thresholds_brain)
                regions_brain = np.uint8(regions_brain*255)
                ret,class1 = cv2.threshold(regions_brain,127,255,cv2.THRESH_BINARY)
                class1Inv = cv2.bitwise_not(class1)
                brainMask = self.getMask2D(class1)
                backgroundMask =  cv2.bitwise_not(brainMask)
                segmentedImg3 = matterMask.copy()
                kernel = np.ones((3,3), np.uint8)
                segmentedImg3 = cv2.erode(segmentedImg3, kernel)
                kernel = np.ones((14,14), np.uint8)
                segmentedImg3 = cv2.morphologyEx(segmentedImg3, cv2.MORPH_CLOSE, kernel)
                segmentedImg3temp = cv2.morphologyEx(matterMask, cv2.MORPH_CLOSE, kernel)
                segmentedImg3ab = np.uint8(segmentedImg3temp - segmentedImg3)
                class2 = backgroundMask - class1Inv - (segmentedImg3/255)
                class2 = np.uint8(class2*255)
                return class1,class2,segmentedImg3,segmentedImg3ab

        def otsuMeasureLableDilate3D(self,img):
                '''
                This function separate the inner matter from the outer ring using multi otsu method and measure label connective detection method and segments class012
                Arguments -
                        img                     : original grayscale image stack
                Returns -
                        segmentedImg1           : class 1 mask
                        segmentedImg2           : class 2 mask
                        segmentedImg3           : inner matter mask - class345
                        segmentedImg3ab         : For boosting accuracy of class 3, extra periphery region mask
                '''
                thresholds_brain = threshold_multiotsu(img,classes=2)
                regions_brain = np.digitize(img, bins=thresholds_brain)
```

```
            regions_brain = np.uint8(regions_brain*255)
            labeledImage = measure.label(regions_brain)
            labeledImage = labeledImage.reshape(10,362,434)
            class345 = np.zeros(labeledImage.shape)
            class345[np.where(labeledImage==2)]=255
            class345 = np.uint8(class345)
            matterMask = self.getMask3D(class345)
            kernel = np.ones((5,5), np.uint8)
            matterMask = cv2.dilate(matterMask, kernel)
            class012 = cv2.bitwise_not(matterMask)/255
            class012  = class012 * img
            thresholds_brain = threshold_multiotsu(class012,classes=3)
            regions_brain = np.digitize(class012, bins=thresholds_brain)
            regions_brain = np.uint8(regions_brain*255)
            ret,class1 = cv2.threshold(regions_brain,127,255,cv2.THRESH_BINARY)
            class1Inv = cv2.bitwise_not(class1)
            brainMask = self.getMask3D(class1)
            backgroundMask =  cv2.bitwise_not(brainMask)
            segmentedImg3 = matterMask.copy()
            kernel = np.ones((3,3), np.uint8)
            segmentedImg3 = cv2.erode(segmentedImg3, kernel)
            kernel = np.ones((14,14), np.uint8)
            segmentedImg3 = cv2.morphologyEx(segmentedImg3, cv2.MORPH_CLOSE, kernel)
            segmentedImg3temp = cv2.morphologyEx(matterMask, cv2.MORPH_CLOSE, kernel)
            segmentedImg3ab = np.uint8(segmentedImg3temp - segmentedImg3)
            class2 = backgroundMask - class1Inv - (segmentedImg3/255)
            class2 = np.uint8(class2*255)
            return class1,class2,segmentedImg3,segmentedImg3ab

    def custom2D(self):
            '''
            This function reads 2D gray image and segments by calling appropriate functions and finally writes the segmented image into output path
            Arguments -
                        input path              : specified in init
                        output path             : specified in init
            Returns -
                        None
            '''
            for file in self.files:
                        if file.endswith(".png"):
                                    path = os.path.join(sys.argv[1],file)
                                    img = cv2.imread(path)
                                    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                                    segmentedImg1,segmentedImg2,segmentedImg3,segmentedImg3ab = self.otsuMeasureLableDilate(img)
                                    clustImg2 = self.segmentInnerPart(segmentedImg3,img)
                                    finaltemp = np.zeros(img.shape)
                                    finaltemp[np.where(segmentedImg3ab==255)] = 80
                                    finaltemp[np.where(clustImg2==80)] = 80
                                    kernel = np.ones((3,3), np.uint8)
                                    finaltemp = cv2.morphologyEx(finaltemp, cv2.MORPH_CLOSE, kernel)
                                    finalimage = np.zeros(img.shape)
                                    finalimage[np.where(segmentedImg1==255)] = 50
                                    finalimage[np.where(segmentedImg2==255)] = 190
                                    finalimage[np.where(finaltemp==80)] = 80
                                    finalimage[np.where(clustImg2==150)] = 150
                                    finalimage[np.where(clustImg2==250)] = 250
                                    cv2.imwrite(os.path.join(sys.argv[2],file),finalimage)

    def custom3D(self):
            '''
            This function iteratively reads 2D gray image and stack them and does the segmentation task by calling
            appropriate functions to process all channel simultaneously and finally writes the image slices by slices into output path
            Arguments -
                        input path                          : specified in init
                        output path             : specified in init
            Returns -
                        None
            '''
            images = []
            for file in sorted(self.files):
                        if file.endswith(".png"):
                                    path = os.path.join(sys.argv[1],file)
                                    img = cv2.imread(path)
                                    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                                    images.append(img)
            images = np.asarray(images)
            segmentedImg1,segmentedImg2,segmentedImg3,segmentedImg3ab = self.otsuMeasureLableDilate3D(images)
            clustImg2 = self.segmentInnerPart(segmentedImg3,images)
            finaltemp = np.zeros(images.shape)
            finaltemp[np.where(segmentedImg3ab==255)] = 80
            finaltemp[np.where(clustImg2==80)] = 80
            kernel = np.ones((5,5), np.uint8)
            finaltemp = cv2.morphologyEx(finaltemp, cv2.MORPH_CLOSE, kernel)
            finalimage = np.zeros(images.shape)
            finalimage[np.where(segmentedImg1==255)] = 50
            finalimage[np.where(finaltemp==80)] = 80
            finalimage[np.where(segmentedImg2==255)] = 190
```

```
                        finalimage[np.where(clustImg2==150)] = 150
                        finalimage[np.where(clustImg2==250)] = 250
                        for i in range(0,10):
                                    img = finalimage[i].copy()
                                    file = str(i)+".png"
                                    cv2.imwrite(os.path.join(sys.argv[2],file),img)
# Main Call
if __name__ == '__main__':
            Main()
```

Fuzzy Clustering Code - 2D and 3D

```
'''
File : Finalcluster.py to perform fuzzy clustering
'''
import numpy as np
import cv2
import skfuzzy as fuzz
import os
import sys

class fuzzyCluster:
            '''This class is responsible for clustering the image using fuzzy algorithm'''
            def __init__(self,img,nCluster):
                        self.img = img.copy()
                        self.numCluster = nCluster

            def getRemain(self,valmax,valmin):
                        '''
                        This function is a supporting function for just coloring.
                        '''
                        colors = [0,80,150,250]
                        for color in colors:
                                    if color not in [valmax,valmin]:
                                                remain1 = color
                                                break
                        for color in colors:
                                    if color not in [valmax,valmin,remain1]:
                                                remain2 = color
                                                break
                        return remain1,remain2

            def getMinMax(self,new_img):
                        '''
                        This function is a supporting function for just coloring.
                        '''
                        classes= []
                        colors = [0,80,150,250]
                        for i in range(0,4):
                                    classes.append(len(np.where(new_img==colors[i])[0]))
                        mini = min(classes)
                        maxi = max(classes)
                        for i in range(0,4):
                                    if mini == classes[i]:
                                                valmin = colors[i]
                                                break
                        for i in range(0,4):
                                    if maxi == classes[i]:
                                                valmax = colors[i]
                                                break
                        return valmin,valmax

            def cluster(self):
                        '''
                        This function is responsible for performing fuzzy clustering
algorithm on 2D for class 3, 4, 5
                        Arguments -
                                    img                                     : original
grayscale image (region of class 345 only)
                        Returns -
                                    new_img                     : segmented class 3, 4, 5
image
                        '''
                        dim = self.img.shape
                        self.img = self.img.reshape(1,self.img.shape[0]*self.img.shape[1])
                        cntr, u, u0, d, jm, p, fpc
=fuzz.cluster.cmeans(self.img,self.numCluster,2,0.5,20)
                        new_img= self.changeColorFuzzycmeans(u,cntr)
                        new_img = new_img.reshape(dim)
                        valmin,valmax = self.getMinMax(new_img)
                        new_img1 = np.zeros(new_img.shape)
                        new_img2 = np.zeros(new_img.shape)
                        new_img3 = np.zeros(new_img.shape)
                        new_img4 = np.zeros(new_img.shape)
                        new_img1[np.where(new_img==valmin)] = [255]
                        kernel = np.ones((5,5),np.uint8)
                        new_img1 = cv2.morphologyEx(new_img1, cv2.MORPH_CLOSE, kernel)
                        new_img1[np.where(new_img1==255)] = [80]
                        remain1, remain2 = self.getRemain(valmax,valmin)
                        new_img3[np.where(new_img==remain1)] = [250]
                        new_img4[np.where(new_img==remain2)] = [250]
                        new_img3 = np.uint8(new_img3)
                        new_img4 = np.uint8(new_img4)
                        contours, hierarchy = cv2.findContours(new_img3, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
                        cnt = sorted(contours, key=cv2.contourArea)
                        x,y,w,h = cv2.boundingRect(cnt[-1])
                        area1 = w * h
                        contours, hierarchy = cv2.findContours(new_img4, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
                        cnt = sorted(contours, key=cv2.contourArea)
                        x,y,w,h = cv2.boundingRect(cnt[-1])
                        area2 = w * h
                        if area1 > area2:
                                    new_img3[np.where(new_img==remain1)] = 150
                        else:
                                    new_img4[np.where(new_img==remain2)] = 150
                        new_img = new_img1 + new_img2 + new_img3 + new_img4
                        new_img[np.where(new_img==230)] = [150]
                        return new_img

            def cluster3D(self):
                        '''
                        This function is responsible for performing fuzzy clustering
algorithm on 3D for class 3, 4, 5
                        Arguments -
                                    img                                     : original
grayscale image stack (region of class 345 only)
                        Returns -
                                    new_img                     : segmented class 3, 4, 5
image
                        '''
                        dim = self.img.shape
                        self.img  = self.img.reshape(1,362*434*10)
                        cntr, u, u0, d, jm, p, fpc =fuzz.cluster.cmeans(self.img
,self.numCluster,2,0.05,300)
                        new_img= self.changeColorFuzzycmeans(u,cntr)
                        newImg = new_img.reshape(dim)
                        newImgs = []
                        for i in range(0,10):
                                    new_img = newImg[i].copy()
                                    valmin,valmax = self.getMinMax(new_img)
                                    new_img1 = np.zeros(new_img.shape)
                                    new_img2 = np.zeros(new_img.shape)
                                    new_img3 = np.zeros(new_img.shape)
                                    new_img4 = np.zeros(new_img.shape)
                                    new_img1[np.where(new_img==valmin)] = [255]
                                    kernel = np.ones((5,5),np.uint8)
                                    new_img1 = cv2.morphologyEx(new_img1, cv2.MORPH_CLOSE,
kernel)
                                    new_img1[np.where(new_img1==255)] = [80]
                                    remain1, remain2 = self.getRemain(valmax,valmin)
                                    new_img3[np.where(new_img==remain1)] = [250]
                                    new_img4[np.where(new_img==remain2)] = [250]
                                    new_img3 = np.uint8(new_img3)
                                    new_img4 = np.uint8(new_img4)
                                    contours, hierarchy = cv2.findContours(new_img3,
cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
                                    cnt = sorted(contours, key=cv2.contourArea)
                                    x,y,w,h = cv2.boundingRect(cnt[-1])
                                    area1 = w * h
                                    contours, hierarchy = cv2.findContours(new_img4,
cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
                                    cnt = sorted(contours, key=cv2.contourArea)
                                    x,y,w,h = cv2.boundingRect(cnt[-1])
                                    area2 = w * h
                                    if area1 > area2:
                                                new_img3[np.where(new_img==remain1)] = 150
                                    else:
                                                new_img4[np.where(new_img==remain2)] = 150
                                    new_img = new_img1 + new_img2 + new_img3 + new_img4
                                    new_img[np.where(new_img==230)] = [150]
                                    newImgs.append(new_img)
                        return np.asarray(newImgs)

            def changeColorFuzzycmeans(self,cluster_membership, clusters):
                        '''
                        This function is a supporting function for just coloring.
                        '''
                        color1 = [0,      80,     150,      250] #R
                        img1 = []
                        clusters = list(clusters)
                        clusters.sort()
                        for pix in cluster_membership.T:
                                    if clusters[np.argmax(pix)] == clusters[0]:
                                                img1.append(color1[0])
                                    elif clusters[np.argmax(pix)] == clusters[1]:
                                                img1.append(color1[1])
```

```
                                    elif clusters[np.argmax(pix)] == clusters[2]:
                                                img1.append(color1[2])
                                    elif clusters[np.argmax(pix)] == clusters[3]:
                                                img1.append(color1[3])
```

```
                        return np.asarray(img1)
```

MultiOTSU thresholding Code - 2D and 3D

```
'''
File : threshold.py to perform MultiOTSU
'''
import numpy as np
from skimage.filters import threshold_multiotsu

class Threshold:
            '''
            This class is responsible for doing MultiOTSU thresholding for 2D and 3D data
            '''
            def __init__(self,img):

                        self.img = img.copy()

            def handle(self,img1,img2):
                        '''
                        This function is helping function to merge multiOTSU output to fuzzy
                        Arguments -
                                    img1                                    : MultiOtsu
output
                                    img2                                    : Fuzzy output
                        Returns -
                                    img1                                    : Merged
output
```

```
                        '''
                        img1[np.where(img2==150)] = [150]
                        img1[np.where(img2==250)] = [250]
                        return img1

            def multiOtsu(self):
                        '''
                        This function is responsible for performing multiOTSU on 2D for class
3, 4, 5
                        Arguments -
                                    img                                     : original
grayscale image stack (region of class 345 only)
                        Returns -
                                    new_img                     : segmented class 3, 4, 5
images
                        '''
                        img = self.img.copy()
                        thresholds_brain = threshold_multiotsu(img,classes=4)
                        regions_brain = np.digitize(img, bins=thresholds_brain)
                        regions_brain[np.where(regions_brain==3)] = [250]
                        regions_brain[np.where(regions_brain==2)] = [150]
                        regions_brain[np.where(regions_brain==1)] = [80]
                        return regions_brain
```

IOU validation Code - 2D and 3D

```
'''
File : validation.py to check IOU scores between ground truth and output
Usage : python3 validation.py <input_path> <output_path>
'''
import numpy as np
import cv2
import os
import sys

files = os.listdir(sys.argv[2])

def getMasksforGT(img):
            '''
            This function is responsible for generating the mask for ground truth images
            Input : 3 grayimages
            output : masked images
            '''
            image = []
            colors = [0,51,102,153,204,255]
            for i in range(0,6):
                        tempImg = np.zeros(img.shape)
                        tempImg[np.where(img==colors[i])] = 255
                        image.append(tempImg)
            return image

def getMasksforPTgray(img):
            '''
            This function is responsible for generating the mask for segmented 2D images
            Input : grayimages
            output : masked images
            '''
            image = []
            colors = [0,50,190,80,150,250]
            for i in range(0,6):
                        tempImg = np.zeros(img.shape)
                        tempImg[np.where(img==colors[i])] = 255
                        image.append(tempImg)
            return image

def getMasksforPT(img):
            '''
            This function is responsible for generating the mask for segmented 2D images which
was colored
            Input : grayimages
            output : masked images
            '''
            image = []
            colors = [16,98,184,226,111,38]
            for i in range(0,6):
                        tempImg = np.zeros(img.shape)
                        tempImg[np.where(img==colors[i])] = 255
```

```
                        image.append(tempImg)
            return image

def calculateIOU(target,prediction):
            '''
            This function is responsible for calculating the IOU for given masks
            Input : Segmented mask, groundtruth mask
            output : IOU value
            '''
            intersection = np.logical_and(target, prediction)
            union = np.logical_or(target, prediction)
            iou_score = np.sum(intersection) / np.sum(union)
            return iou_score

class0 = []
class1 = []
class2 = []
class3 = []
class4 = []
class5 = []
colored = False #if passing colored images, then flag has to be true
for file in files:
            if file.endswith(".png"):
                        print(file)
                        groundTruthImg = cv2.imread(os.path.join(sys.argv[1],file),0)
                        predictImg = cv2.imread(os.path.join(sys.argv[2],file))
                        predictImg = cv2.cvtColor(predictImg, cv2.COLOR_BGR2GRAY)
                        Gtimgs = getMasksforGT(groundTruthImg)
                        if colored:
                                    Pimgs = getMasksforPT(predictImg)
                        else:
                                    Pimgs = getMasksforPTgray(predictImg)
                        IOUval = []
                        for i in range(0,6):
                                    val = calculateIOU(Gtimgs[i],Pimgs[i])
                                    IOUval.append(val)
                        print(IOUval)
                        class0.append(IOUval[0])
                        class1.append(IOUval[1])
                        class2.append(IOUval[2])
                        class3.append(IOUval[3])
                        class4.append(IOUval[4])
                        class5.append(IOUval[5])
lis = []
lis.append(np.average(class0))
lis.append(np.average(class1))
lis.append(np.average(class2))
lis.append(np.average(class3))
lis.append(np.average(class4))
lis.append(np.average(class5))
print("\nAverage IOU score ", lis)
print("\n")
```

---END---