COMPUTER VISION MIDTERM ASSIGNMENT

-Vineeth S Subramanyam (2221296)

I. AIM

Given a set of consecutive MRI axial cross sections, develop image processing algorithms in 2D and 3D that are capable of producing segmented masks for the following classes: Air, Skin/Scalp, Skull, CSF, Gray Matter and White Matter. Additionally carry out experiments to empirically validate the developed methods.

II. METHODOLOGY

For the tasks of both 2D and 3D segmentation, a combination of different image processing techniques were tested. The final techniques that were selected involved the use of multi class Otsu Thresholding and Image K means clustering.

Image KMeans Clustering Algorithm Summary

```
1. Set the number of cluster centers k.
2. Initialize array of k cluster centers as k random colors.
3. Initialize k lists to hold points from each cluster.
4. Loop till convergence, or number of iterations:
    a. Iterate through each point in the dataset
        ● Compute Euclidean distance between current point to each cluster center.
        ● Add the current point to the cluster list of the class with lowest euclidean distance from its center.
    b. Compute new cluster centers to be the mean of the list of each cluster points.
```

Multi Otsu Threshold Algorithm Summary

```
1. Set the number of classes to be k.
2. Compute all possible combinations of k-1 thresholds, such that t2>t1, t3>t2, etc. (E.g. (1,2,3,4), (1,2,3,5), (1,2,3,6)….etc)
3. Iterate over each possible combinations:
    a. Compute mean, variance and weight of each class
    b. Compute between class variance given by formula
       $\sigma_B^2 = \sum_{k=1}^{K} P_k(m_k - m_G)^2$, where $P_k = \sum_{i \in C_k} p_i$ and $m_k = \sum_{i \in C_k} ip_i$
4. Select the set of thresholds that have maximum between class variance
```

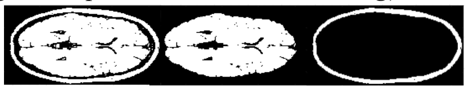
A. 2D Segmentation Algorithm

1) Load dataset (**Brain.mat**)
2) For each (**image, label**) pair:

a) Apply **Multi class Otsu Thresholding** to segment out the outer ring(*Class 1*) , and the inner portion of the scan(*Class 3-5*).

b) Find all connected components in the image mask(**bwlabel**).

c) Sort the components by area, and select the 2 biggest components. (*Biggest component consists of Classes 3-5, and the next biggest component is the outermost ring*)



Background Class

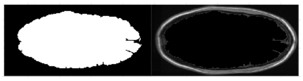
d) To extract the background class, fill (**imfill**) the outermost ring (*obtained from* **c)**), and take the complement of the image obtained.

Classes 1-2

e) Extract the mask corresponding to only the inner portion of the brain(*obtained from step c*), and apply **imfill** to close all holes.

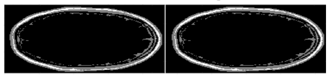
f) Find all pixels in this mask that contain a value=1.

g) Create a copy of the original image and set all the pixels from step **f)** to be 0.



h) Apply one of two methods:

- ***Thresholding Based:*** Apply **Multi class Otsu Thresholding** to segment the image mask into 3 classes (**multithresh** followed by **imquantize**).
- ***Clustering Based:*** Apply **K Means Clustering** to segment the image mask into 3 classes.(**imsegkmeans**)



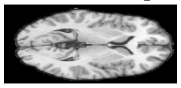
i) Extract the mask corresponding to Class 1, and take its complement. (*Additional pixels inside the ring get filled when creating the masks of classes 3-5, thus reducing error.*)

j) Extract the mask with index corresponding to Class 2 and then subtract the background, and inner matter.

Classes 3-5

k) Erode the mask containing the largest component obtained from step **c)**.

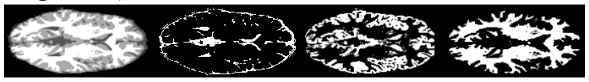
l) Create a mask from the original image containing only pixels inside the largest component after step **k)**.



m) Apply one of two methods:

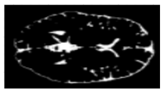
Thresholding Based

- Apply Multi class Otsu Thresholding to segment the image mask into 4 classes(**multithresh+imquantize**)
- Extract the masks corresponding to Class 3,4 and 5 depending on the quantized image indexes. (*e.g. Class 3 mask obtained by using L==3)*

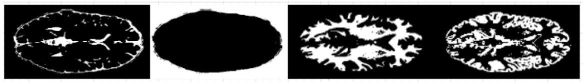


K means Based:

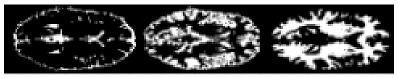
- Apply **K Means Clustering** to segment the mask into 3 classes.
- Since K Means Clustering initializes centers randomly, the class indexes differ from iteration to iteration for each label, therefore an additional check is added to compare similarity of masks to the ground truth to find the correct index.
- **For** i=3:5 (*Class 3-5 of ground truth*)
  - ○ Create a ground truth mask corresponding to class **i** using **label** *(e.g. label==i)*.



  - ○ **For** j=1:4 (*Indexes corresponding to clusters*)
  - ○ Create masks corresponding to class **j** using the clustered image *(e.g. L==j)*.
  - ○ Compute Structural Similarity measure(**SSIM**) between the ground truth mask, and each clustered mask.

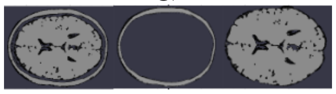


- ○ **If SSIM**>0.8: *(Indicates that the correct index has been obtained)*:
  - ■ Extract predicted mask for the corresponding **i** th ground truth class.



B. 3D Segmentation Algorithm

1. Load dataset (**Brain.mat**)
2. Apply **Multi class Otsu Thresholding** to segment out the outer ring(*Class 1*), and the inner portion of the scan(*Class 3-5*) using the input 3D scan.
3. Find all connected components in the 3D mask(**bwlabeln**).
4. Sort the components by area, and select the 2 biggest components. (*Biggest component consists of Classes 3-5, and the next biggest is the outermost ring*)

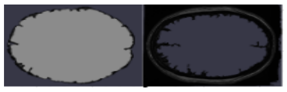


Background Class:

5. To extract the background class, fill the outermost ring (**imfill3**), and take the complement of the 3D mask obtained.

Classes 1-2:

6. Extract the 3D mask corresponding to only the inner portion of the brain(*obtained from step 4*), and apply **imfill3** to close all holes. Find all pixels that contain a value=1.
7. Create a copy of the original 3D input mask and set all the pixels from step **6)** to be 0.

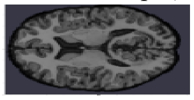


8. Apply one of two methods:
   - ***Thresholding Based:*** Apply **Multi class Otsu Thresholding** to segment the 3D mask into 3 classes (**multithresh** followed by **imquantize**).
   - ***Clustering Based:*** Apply **K Means Clustering** to segment the 3D mask into 3 classes.(**imsegkmeans3**)

9. Extract the mask corresponding to Class 1, and take its complement. (*Additional pixels inside the ring get filled when creating the masks of classes 3-5, thus reducing error.*)
0. Extract the mask with index corresponding to Class 2 by subtracting the background, and inner matter.

Classes 3-5

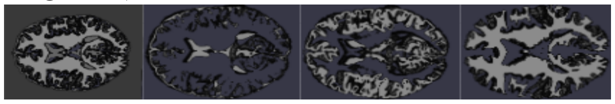
1. Erode the mask containing the largest component obtained from step **4)**.
2. Create a mask from the original image containing only pixels inside the largest component after step **9)**.



3. Apply one of two methods:

   Thresholding Based

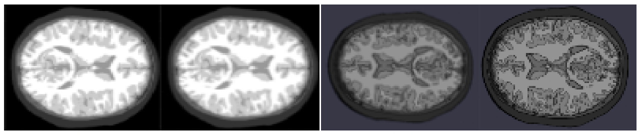
- Apply Multi class Otsu Thresholding to segment the image mask into 4 classes(**multithresh+imquantize**)
- Extract the masks corresponding to Class 3,4 and 5 depending on the quantized image indexes. (*e.g. Class 3 mask obtained by using L==3)*



K means Based:

- Apply **K Means Clustering** to segment the 3D mask into 3 classes.
- Since K Means Clustering initializes centers randomly, the class indexes differ from iteration to iteration for each label, therefore an additional check is added to compare similarity of masks to the ground truth to find the correct index.
- **For** i=3:5 (*Class 3-5 of ground truth*)
  - ○ Create a ground truth mask corresponding to class **i** using **label** *(e.g. label==i)*.
  - ○ **For** j=1:4 (*Indexes corresponding to clusters*)
  - ○ Create masks corresponding to class **j** using the clustered image *(e.g. L==j)*.
  - ○ Compute Structural Similarity measure(**SSIM**) between the ground truth mask, and each clustered mask.
  - ○ **If SSIM**>0.8: *(Indicates that the correct index has been obtained)*:
    - ■ Extract predicted mask for the corresponding **i** th ground truth class.

III. RESULTS



The approaches are tested and validated so as to maximize the following criteria:

- **Pixelwise Accuracy:** The segmentation approach should be capable of classifying individual pixels correctly.
- **Overall Image structure:** The segmentation approach should ensure that each mask segmented has the same overall structure as the ground truth mask. This metric disregards individual pixels, and rather emphasizes on the entire image.
- **Overlap between ground truth and segmented label:** The segmentation approach should create masks that overlap perfectly with the ground truth label. At the same time, it is important to consider that the background class dominates a majority of each class mask, and therefore the selected approach should have a high precision and recall.
- **Inference time:**The approach chosen should be capable of being used in a real time application.

The segmented results are evaluated to maximize the set of criteria as defined above using the following metrics:

- **Intersection over Union (IOU):** It is an estimate of overlap between the ground truth and predicted label.
- **Structural Similarity Measure(SSIM)**: It is an estimate of overall similarity of two images using their luminance, contrast and overall structure.(Reference)
- **Accuracy:** It represents a total of how many pixels were correctly classified.
- **F1 Score**: It is the Harmonic mean of Precision and recall. Allows in selecting a model with a balance between precision and recall.

- **Mean Score**: The final metric used to compare the algorithms is the average of these different scores.

$IOU = \frac{TP}{TP+FP+FN}$	$\text{SSIM}(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2+\mu_y^2+C_1)(\sigma_x^2+\sigma_y^2+C_2)}.$
$Acc = \frac{TP+TN}{TP+TN+FP+FN}$	$F1 = \frac{2*(P*R)}{P+R}$

$$MeanScore = \frac{IOU + SSIM + F1 + ACC}{4}$$

TP: True Positive, FP: False Positive, TN: True Negative, FN: False Negative, P:Precision, R:Recall

K Means 2D algorithm Scores (Overall Mean = 0.942)

Metric	C0	C1	C2	C3	C4	C5
IOU	0.995	0.947	0.896	0.756	0.896	0.943
SSIM	0.988	0.953	0.967	0.883	0.858	0.916
F1 Score	0.997	0.973	0.945	0.861	0.945	0.970
ACC	0.998	0.992	0.993	0.979	0.974	0.987
Mean Score	0.995	0.966	0.950	0.870	0.918	0.954

Thresholding 2D algorithm Scores (Overall Mean = 0.942)

Metric	C0	C1	C2	C3	C4	C5
IOU	0.995	0.947	0.906	0.757	0.890	0.939
SSIM	0.988	0.953	0.971	0.883	0.852	0.911
F1 Score	0.997	0.973	0.951	0.862	0.942	0.968
ACC	0.998	0.991	0.994	0.979	0.973	0.986
Mean Score	0.994	0.966	0.955	0.870	0.914	0.951

K Means 3D algorithm Scores (Overall Mean = 0.942)

Metric	C0	C1	C2	C3	C4	C5
IOU	0.995	0.951	0.893	0.760	0.897	0.944
SSIM	0.987	0.943	0.966	0.855	0.875	0.914
F1 Score	0.997	0.974	0.943	0.864	0.945	0.971
ACC	0.998	0.992	0.993	0.979	0.974	0.987
Mean Score	0.994	0.965	0.949	0.865	0.923	0.954

Thresholding 3D algorithm Scores (Overall Mean = 0.940)

Metric	C0	C1	C2	C3	C4	C5
IOU	0.995	0.951	0.894	0.77	0.887	0.930
SSIM	0.987	0.944	0.966	0.863	0.865	0.892
F1 Score	0.997	0.975	0.944	0.870	0.940	0.963
ACC	0.998	0.992	0.993	0.980	0.972	0.984
Mean Score	0.994	0.965	0.949	0.871	0.916	0.942

Time Taken by each algorithm

Method	Time Taken (seconds)
K Means 2D	3.767
Multi Thresholding 2D	1.998
K Means 3D	4.108
Multi Otsu Thresholding 3D	2.107

IV. CONCLUSION

A. *Final Algorithm Selection*

From the tables in **III**, we can see that the performance of both the **Multi-class Otsu thresholding**, and **K-means clustering** approaches result in similar mean scores for both 2D and 3D. This result is expected since both techniques are operating purely on image intensities, with the same set of image masks. The **K-means clustering** techniques are significantly slower than the thresholding approaches due to the presence of additional checks in clustering required to get the correct mask indexes. Therefore the final chosen algorithm makes use of only **Multi-class Otsu thresholding** which results in a mean score of **0.942** for 2D and 3D.

B. *Result Analysis*

From the tables in **III**, we can infer that the performance of the algorithm suffers for **classes 2 and 3**. This is because of the slight overlap between the two class boundaries, as well as the fact that both classes have a very thin mask where minor pixel errors lead to a large error. The algorithm performs significantly well in **classes 0,1 and 5**, since these classes are easily differentiable by their pixel intensities and have more or less solid masks without complex structures. The algorithm performs moderately well on **class 4** due to its mask having a complex structure with a significant number of holes inside it.

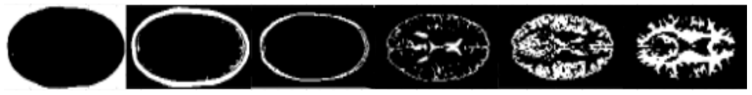
C. *Comparison of 2D and 3D approaches*

When comparing the 3D approaches to their 2D counterparts, we see more or less an identical performance with there only being a slight improvement in the scores of the 3D algorithm in the masks of **class 3.** These minor differences can be attributed to the contour detection algorithm(**bwlabel**) using a connectivity of **8** pixels for 2D, and a connectivity of **26** for 3D, as well as certain minor differences in the implementation of the codes which can cause a small variation in boundaries. Such a similarity in performance is expected since the internal working of the algorithms used does not differ between 2D and 3D. The main difference between 2D and 3D is in the inference time, where the 3D algorithm is slower than its 2D counterpart.

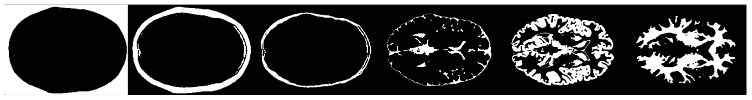
D. *Additional Experiments*

Additionally two algorithms were tested. These techniques are not detailed in the report as they were not taught in the class. Their codes are added to the project Github Repository:

1) Fuzzy C Means: The fuzzy c means clustering approach was implemented to segment the **classes 3-5** which makes use of a distance transform that allows points to belong to multiple clusters with a probability**.** This approach resulted in a mean overall score of **0.947**, with greatly improved performance in **Class 4 and 5**. The implementation of Fuzzy CMeans is referenced from (Reference).



2) Deep Learning: A UNet based semantic segmentation architecture was trained on the dataset of 10 images using **Dice Loss** and **Adam Optimizer**.The model achieved a final mean score of **0.952,** with the model performing significantly better on **Class 3,4 and 5**. Since the dataset contains only 10 images, the model was trained with augmentation techniques, and trained for only a very small number of epochs. The model performance can be improved further by training on a larger dataset.



CODE SUBMISSION

The codes below contains methods for 2D and 3D segmentation along with their supporting scripts. Additionally, the github repository containing all the codes and additional experiments can be found at: Github Repository.

Q1_THRESH.m

```
% Load Dataset
load Brain.mat
% Set seed for random number generator
rng('default');

% Initialize arrays to hold scores
similarity_score = zeros(6,1,'double');
dice_score = zeros(6, 1, 'double');
ssim_array = zeros(6, 1, 'double');
acc_array = zeros(6, 1, 'double');
precision_array = zeros(6, 1, 'double');
recall_array = zeros(6, 1, 'double');
f1_array = zeros(6, 1, 'double');
mean_score = zeros(6, 1, 'double');

% Iterate over all images and labels
for j=1:10
    img = T1(:,:,j);    % Read current image
    lab = label(:,:,j); % Read current label

    % Display each mask from ground truth
    figure(); colormap gray; axis equal; axis off;
    k=0;
    for i=1:2:12
        subplot(6, 2, i);
        imagesc(lab==k)
        k=k+1;
    end

    % Create dummy mask to hold predicted segmentation mask
    label_mask = zeros(size(lab));
    label_idx=0;

    % Threshold original image to form binary image
    thresh = multithresh(img, 1);
    L_outer = imquantize(img,thresh);

    % Extract only white component
    mask = zeros(size(lab));
    threshed_vals = L_outer == 2;
    mask(threshed_vals) = 1;

    % Detect connected components in the binary mask, and sort them
by size
    [L, num] = bwlabel(mask, 8);
    counts = sum(bsxfun(@eq,L(:),1:num));
    [vals, inds] = maxk(counts, 2); % Find two biggest contours
    ind1 = inds(1); % Index corresponding to biggest contour
    ind2 = inds(2); % Index corresponding to second biggest contour

    % Create seperate masks for largest components
    outer_ring_mask = zeros(size(L_outer));
    inner_ring_mask = zeros(size(L_outer));
    outer_vals = L== ind2;
    inner_vals = L == ind1;
    outer_ring_mask(outer_vals) = 1;
    inner_ring_mask(inner_vals) = 1;
    % Create background mask by filling outer ring
    bg_mask = imcomplement(imfill(outer_ring_mask, 'holes'));
    % Fill inner mask to close holes
    filled_inner_ring_mask = imfill(inner_ring_mask, 'holes');

    % Create background mask, and update final predicted segmentation
mask
    subplot(6,2,2);
    label_vals = bg_mask == 1;
    label_mask(label_vals)=label_idx;
    label_idx=label_idx+1;
    imagesc(bg_mask);

    % Create an image mask containing only outer ring
    outer_img_mask = img;
    outer_vals = filled_inner_ring_mask == 1;
    outer_img_mask(outer_vals) = 0;
    % Threshold and quantize the image mask into 2 classes
    thresh = multithresh(outer_img_mask, 2);
    L = imquantize(outer_img_mask, thresh);

    %Extract the mask of Class 1 from the segmented mask, and update
    % final segmented label mask
    outer_ring_mask = zeros((size(lab)));
    outer_ring_vals = L == 1;
    outer_ring_mask(outer_ring_vals) = 1;
    outer_ring_mask = imcomplement(outer_ring_mask);
    outer_ring_label_vals = outer_ring_mask == 1;
    label_mask(outer_ring_label_vals)=label_idx;
    label_idx = label_idx+1;
    subplot(6,2,4);
    imagesc(outer_ring_mask);

    %Extract the mask of Class 2 from the segmented mask, and update
    % final segmented label mask
    inner_ring_mask = zeros((size(lab)));
    inner_ring_vals = L == 2;
    inner_ring_mask(outer_ring_vals) = 1;
    bg_vals = bg_mask == 1;
    inner_ring_mask(bg_vals) = 0;
    inner_vals = filled_inner_ring_mask ==1;
    inner_ring_mask(inner_vals) = 0;
    inner_mask_vals = inner_ring_mask == 1;
    label_mask(inner_mask_vals)=label_idx;
    label_idx = label_idx+1;
    subplot(6,2,6);
    imagesc(L==2);

    % Create an image mask containing only inner components
    se = strel('disk', 5);
    inner_img_mask = img;
    inner_vals = imerode(filled_inner_ring_mask == 0, se);
    inner_img_mask(inner_vals) = 0;
    % Perform Multi Otsu Thresholding to segment inner image mask
    thresh = multithresh(inner_img_mask, 3);
    L_inner = imquantize(inner_img_mask, thresh);

    % Create mask for class 3
    lab4_mask = zeros((size(lab)));
    lab4_vals = L_inner == 2;
    lab4_mask(lab4_vals) = 1;
    label_mask(lab4_vals) = label_idx;
    label_idx = label_idx+1;
    subplot(6,2,8);
    imagesc(lab4_mask);

    % Create mask for class 4
    lab5_mask = zeros((size(lab)));
    lab5_vals = L_inner == 3;
    lab5_mask(lab5_vals) = 1;
    label_mask(lab5_vals) = label_idx;
    label_idx = label_idx+1;
    subplot(6,2,10);
    imagesc(lab5_mask);

    % Create mask for class 5
    lab6_mask = zeros((size(lab)));
    lab6_vals = L_inner == 4;
    lab6_mask(lab6_vals) = 1;
    lab6_vals = lab6_mask == 1;
    label_mask(lab6_vals) = label_idx;
    label_idx = label_idx+1;
    subplot(6,2,12);
    imagesc(lab6_mask);

    % Draw Updated class 1 mask after filling in other class masks
    subplot(6,2,4);
    imagesc(label_mask==1);

    % Calculate metrics to compare ground truth and segmentation mask
    similarity = jaccard(categorical(lab), categorical(label_mask));
    similarity_score = similarity_score + similarity;
    dice_val = dice(categorical(lab), categorical(label_mask));
    dice_score = dice_score + dice_val;
    ssim_score = ssim_scores(lab, label_mask);
    ssim_array = ssim_array + ssim_score;
    [precision, recall, f1_score, acc] = pr(lab, label_mask);
    precision_array = precision_array + precision;
    recall_array = recall_array + recall;
    f1_array = f1_array + f1_score;
    acc_array = acc_array + acc;
end

% Compute mean score
similarity_score = similarity_score / 10;
dice_score = dice_score / 10;
ssim_array = ssim_array / 10;
precision_array = precision_array / 10;
recall_array = recall_array / 10;
f1_array = f1_array / 10;
acc_array = acc_array / 10;
mean_score = (similarity_score + f1_array + ssim_array + acc_array) /
4;
meanval = mean(mean_score);

% Plot final ground truth and segmented result
figure();colormap gray; axis equal; axis off;
subplot(1,2,1);
imagesc(lab);
subplot(1,2,2);
imagesc(label_mask);
```

Q3_THRESH.m

```
% Load Dataset
load Brain.mat
% Set seed for random number generator
rng('default');

% Read Images and Labels
img = T1;
lab = label;

% Create dummy mask to hold final segmented labels
label_mask = zeros(size(lab));
label_idx = 0;

% Threshold original image to form binary image
thresh = multithresh(img, 1);
L_outer = imquantize(img, thresh);
% Extract only white portion
mask = zeros(size(L_outer));
threshed_vals = L_outer == 2;
mask(threshed_vals) = 1;
% Find connected components in 3D
L = bwlabeln(mask);

% Extract masks for two largest components
outer_ring_mask = zeros(size(L_outer));
inner_ring_mask = zeros(size(L_outer));
outer_vals = L== 1;
inner_vals = L == 3;
outer_ring_mask(outer_vals) = 1;
inner_ring_mask(inner_vals) = 1;
% Fill the masks to close holes in 3D
filled_outer_ring_mask = imfill3(outer_ring_mask);
filled_inner_ring_mask = imfill3(inner_ring_mask);

% Extract background mask
bg_mask = zeros((size(lab)));
bg_vals = filled_outer_ring_mask == 0;
bg_mask(bg_vals) = 1;
label_mask(bg_vals) = label_idx;
label_idx = label_idx+1;

% Create a 3D mask containing only outer ring
outer_img_mask = img;
outer_vals = filled_inner_ring_mask == 1;
outer_img_mask(outer_vals) = 0;
% Threshold and quantize the 3D mask into 3 classes
thresh = multithresh(outer_img_mask, 2);
L = imquantize(outer_img_mask, thresh);

%Extract the mask of Class 1 from the segmented mask, and update
% final segmented label mask
outer_ring_mask = zeros((size(lab)));
outer_ring_vals = L == 1;
outer_ring_mask(outer_ring_vals) = 1;
outer_ring_mask = imcomplement(outer_ring_mask);
outer_ring_label_vals = outer_ring_mask == 1;
label_mask(outer_ring_label_vals)=label_idx;
label_idx = label_idx+1;

%Extract the mask of Class 2 from the segmented mask, and update
% final segmented label mask
inner_ring_mask = zeros((size(lab)));
inner_ring_vals = L == 2;
inner_ring_mask(outer_ring_vals) = 1;
bg_vals = bg_mask == 1;
inner_ring_mask(bg_vals) = 0;
inner_vals = filled_inner_ring_mask ==1;
inner_ring_mask(inner_vals) = 0;
inner_mask_vals = inner_ring_mask == 1;
label_mask(inner_mask_vals)=label_idx;
label_idx = label_idx+1;

% Create a 3D mask containing only inner components
se = strel('disk', 5);
inner_img_mask = img;
inner_vals = imdilate(filled_inner_ring_mask, se)==0;
inner_img_mask(inner_vals) = 0;
% Perform Multi Otsu Thresholding to segment inner matter mask
thresh = multithresh(inner_img_mask, 3);
L_inner = imquantize(inner_img_mask, thresh);

% Create mask for class 3
lab4_mask = zeros((size(lab)));
lab4_vals = L_inner == 2;
lab4_mask(lab4_vals) = 1;
label_mask(lab4_vals) = label_idx;
label_idx = label_idx+1;

% Create mask for class 4
lab5_mask = zeros((size(lab)));
lab5_vals = L_inner == 3;
lab5_mask(lab5_vals) = 1;
label_mask(lab5_vals) = label_idx;
label_idx = label_idx+1;

% Create mask for class 5
lab6_mask = zeros((size(lab)));
lab6_vals = L_inner == 4;
lab6_mask(lab6_vals) = 1;
lab6_vals = lab6_mask == 1;
label_mask(lab6_vals) = label_idx;
label_idx = label_idx+1;

% Calculate metrics to compare ground truth and segmentation mask
similarity = jaccard(categorical(lab), categorical(label_mask));
dice_score = dice(categorical(lab), categorical(label_mask));
ssim_score = ssim_scores(lab, label_mask);
[precision, recall, f1_score, acc] = pr(lab, label_mask);
mean_score = (similarity + f1_score + ssim_score + acc) / 4;
mean_val = mean(mean_score);

% Show final ground truth and segmented result
```

```
figure();
volshow(label_mask);
figure();
volshow(lab);
```

pr.m

```
function [precision, recall, f1_score, acc] = pr(gt, label)
    %{
    Description:
        Function to calculate scores for precision, recall, f1 score
and accuracy
    Arguments:
        gt: Ground Truth Segmentation Mask
        label: Predicted Label Mask
    Returns:
        precision: Classwise Precision score (P = TP / (TP + FP))
        recall: Classwise Recall score (R = TP / (TP + FN))
        f1_score: Classwise F1 score (2 * (P * R) / (R + P))
        acc: Classwise Accuracy score(Acc = (TP + TN)/ (TP + TN + FP
+ FN))
    %}
    % Create dummy arrays to hold scores
    precision = zeros(6,1,'double');
    recall = zeros(6,1,'double');
    f1_score = zeros(6,1,'double');
    acc = zeros(6,1,'double');
    for i=0:5   % Iterate over all classes
        % Create ground truth binary mask
        gt_mask = zeros((size(gt)));
        gt_vals = gt == i;
        gt_mask(gt_vals) = 1;

        % Create prediction binary mask
        pred_mask = zeros((size(gt)));
        vals = label == i;
        pred_mask(vals) = 1;

        %Calculate scores
        TP = length(find((gt_mask == 1) & (pred_mask == 1)));
        TN = length(find((gt_mask == 0) & (pred_mask == 0)));
        FP = length(find((gt_mask == 0) & (pred_mask == 1)));
        FN = length(find((gt_mask == 1) & (pred_mask == 0)));
        precision_val = TP / (TP + FP);
        recall_val = TP / (TP + FN);
        acc(i+1) = (TP + TN)/ (TP + TN + FP + FN);
        precision(i+1) = precision_val;
        recall(i+1) = recall_val;
        f1_score(i+1) = 2 * (precision_val * recall_val) /
(recall_val + precision_val);
    end
end
```

imfill.m

```
function mtx=imfill3(mtx)
    %{
    Description:
        Function to fill scan along three dimensions
    Arguments:
        mtx: Input 3D array(W,H,N)
    Returns:
        mtx: Output Matrix filled along 3 dimensions
    %}
    for i=1:3   % Iterate over each dimension
        for j=1:size(mtx,3) % Iterate over each scan
            mtx(:,:,j)=imfill(mtx(:,:,j),'holes');
        end
    end
    for i=1:3   % Iterate over each dimension
        for j=1:size(mtx,3) % Iterate over each scan
            mtx(:,:,j)=imfill(mtx(:,:,j),'holes');
        end
```

```
    end
end
```

ssim.m

```
function score=ssim_scores(gt, label)
    %{
    Description:
        Function to calculate Structural Similarity score
    Arguments:
        gt: Ground Truth Segmentation Mask
        label: Predicted Label Mask
    Returns:
        ssim_scores: Classwise SSIM score
    %}
    % Create dummy arrays to hold scores
    score = zeros(6,1,'double');
    for i=0:5
        % Create ground truth binary mask
        gt_mask = zeros((size(gt)));
        gt_vals = gt == i;
        gt_mask(gt_vals) = 1;

        % Create prediction binary mask
        mask = zeros((size(gt)));
        vals = label == i;
        mask(vals) = 1;

        %Calculate scores
        [ssimval, ~] = ssim(gt_mask, mask);
        score(i+1) = score(i+1) + ssimval;
    end
end
```

Q1_KMeans.m

```
% Load Dataset
load Brain.mat
% Set seed for random number generator
rng('default');

% Initialize arrays to hold scores
similarity_score = zeros(6,1,'double');
dice_score = zeros(6, 1, 'double');
ssim_array = zeros(6, 1, 'double');
acc_array = zeros(6, 1, 'double');
precision_array = zeros(6, 1, 'double');
recall_array = zeros(6, 1, 'double');
f1_array = zeros(6, 1, 'double');
mean_score = zeros(6, 1, 'double');

% Iterate over all images and labels
for j=1:10
    img = T1(:,:,j);    % Read current image
    lab = label(:,:,j); % Read current label

    % Display each mask from ground truth
    figure(); colormap gray; axis equal; axis off;
    k=0;
    for i=1:2:12
        subplot(6, 2, i);
        imagesc(lab==k)
        k=k+1;
    end

    % Create dummy mask to hold predicted segmentation mask
    label_mask = zeros(size(lab));
    label_idx=0;

    % Threshold original image to form binary image
    thresh = multithresh(img, 1);
    L_outer = imquantize(img,thresh);
    % Extract only white component
    mask = zeros(size(lab));
    threshed_vals = L_outer == 2;
    mask(threshed_vals) = 1;

    % Detect connected components in the binary mask, and sort them
by size
    [L, num] = bwlabel(mask, 8);
    counts = sum(bsxfun(@eq,L(:),1:num));
    [vals, inds] = maxk(counts, 2); % Find two biggest contours
    ind1 = inds(1); % Index corresponding to biggest contour
    ind2 = inds(2); % Index corresponding to second biggest contour

    % Create seperate masks for largest components
    outer_ring_mask = zeros(size(L_outer));
    inner_ring_mask = zeros(size(L_outer));
    outer_vals = L== ind2;
    inner_vals = L == ind1;
    outer_ring_mask(outer_vals) = 1;
    inner_ring_mask(inner_vals) = 1;
    % Create background mask by filling outer ring
    bg_mask = imcomplement(imfill(outer_ring_mask, 'holes'));
    % Fill inner mask to close holes
    filled_inner_ring_mask = imfill(inner_ring_mask, 'holes');

    % Create background mask, and update final predicted segmentation
mask
    label_vals = bg_mask == 1;
    label_mask(label_vals)=label_idx;
    label_idx=label_idx+1;
    subplot(6,2,2);
    imagesc(bg_mask);

    % Create an image mask containing only outer ring
    outer_img_mask = img;
    outer_vals = filled_inner_ring_mask == 1;
    outer_img_mask(outer_vals) = 0;
    % Perform K means segmentation on the image mask
    L = imsegkmeans(outer_img_mask, 3);

    %Extract the mask of Class 1 from the segmented mask, and update
    % final segmented label mask
    outer_ring_mask = zeros((size(lab)));
    outer_ring_vals = L == 1;
    outer_ring_mask(outer_ring_vals) = 1;
    outer_ring_mask = imcomplement(outer_ring_mask);
    outer_vals = filled_inner_ring_mask ==1;
    outer_ring_mask(outer_vals) = 0;
    outer_ring_label_vals = outer_ring_mask == 1;
    label_mask(outer_ring_label_vals)=label_idx;
    label_idx = label_idx+1;
    subplot(6,2,4);
    imagesc(outer_ring_mask);

    %Extract the mask of Class 2 from the segmented mask, and update
    % final segmented label mask
    inner_ring_mask = zeros((size(lab)));
    inner_ring_vals = L == 2;
    inner_ring_mask(outer_ring_vals) = 1;
    bg_vals = bg_mask == 1;
    inner_ring_mask(bg_vals) = 0;
    inner_vals = filled_inner_ring_mask ==1;
    inner_ring_mask(inner_vals) = 0;
    inner_mask_vals = inner_ring_mask == 1;
    label_mask(inner_mask_vals)=label_idx;
    label_idx = label_idx+1;
    subplot(6,2,6);
    imagesc(inner_ring_mask);

    % Create an image mask containing only inner components
    se = strel('disk', 5);
    inner_img_mask = img;
    inner_vals = imerode(filled_inner_ring_mask == 0, se);
    inner_img_mask(inner_vals) = 0;
    % Perform K Means segmentation to segment inner image mask
    L_inner = imsegkmeans(inner_img_mask, 4);
```

```
   subplot_idx=8;   % Initialize subplot index to display
    for i=3:5       % Iterate through each of classes 3-5
        % Create ground truth mask for ith class
        gt_mask = zeros((size(lab)));
        gt_vals = lab == i;
        gt_mask(gt_vals) = 1;
        for j=1:4   % Iterate through each k means segmentation class
            % Extract mask corresponding to jth k means class
            mask = zeros((size(lab)));
            vals = L_inner == j;
            mask(vals) = 1;
            % Compute SSIM between GT mask and predicted k means mask
            [ssimval,ssimmap] = ssim(gt_mask, mask);
            if ssimval>0.8     % Check if match is greater than 80%
                % If match is greater than 80%, then create predicted
mask
                label_mask(vals) = label_idx;
                label_idx = label_idx+1;
                subplot(6,2,subplot_idx);
                imagesc(mask);
                subplot_idx = subplot_idx+2;
                break;
            end
        end
    end
    % Draw Updated class 1 mask after filling in other class masks
    subplot(6,2,4);
    imagesc(label_mask==1);

    % Calculate metrics to compare ground truth and segmentation mask
    similarity = jaccard(categorical(lab), categorical(label_mask));
    similarity_score = similarity_score + similarity;
    dice_val = dice(categorical(lab), categorical(label_mask));
    dice_score = dice_score + dice_val;
    ssim_score = ssim_scores(lab, label_mask);
    ssim_array = ssim_array + ssim_score;
    [precision, recall, f1_score, acc] = pr(lab, label_mask);
    precision_array = precision_array + precision;
    recall_array = recall_array + recall;
    f1_array = f1_array + f1_score;
    acc_array = acc_array + acc;
end

% Compute mean score
similarity_score = similarity_score / 10;
dice_score = dice_score / 10;
ssim_array = ssim_array / 10;
precision_array = precision_array / 10;
recall_array = recall_array / 10;
f1_array = f1_array / 10;
acc_array = acc_array / 10;
mean_score = (similarity_score + f1_array + ssim_array + acc_array) /
4;
meanval = mean(mean_score);


% Plot final ground truth and segmented result
figure();colormap gray; axis equal; axis off;
subplot(1,2,1);
imagesc(lab);
subplot(1,2,2);
imagesc(label_mask);
```

Q3_Kmeans.m

```
% Load Dataset
load Brain.mat
% Set seed for random number generator
rng('default');

% Read Images and Labels
img = T1;
```

```
lab = label;

% Create dummy mask to hold final segmented labels
label_mask = zeros(size(lab));
label_idx = 0;

% Threshold original image to form binary image
thresh = multithresh(img, 1);
L_outer = imquantize(img, thresh);
% Extract only white portion
mask = zeros(size(L_outer));
threshed_vals = L_outer == 2;
mask(threshed_vals) = 1;
% Find connected components in 3D
L = bwlabeln(mask);

% Extract masks for two largest components
outer_ring_mask = zeros(size(L_outer));
inner_ring_mask = zeros(size(L_outer));
outer_vals = L== 1;
inner_vals = L == 3;
outer_ring_mask(outer_vals) = 1;
inner_ring_mask(inner_vals) = 1;

% Fill the masks to close holes in 3D
filled_outer_ring_mask = imfill3(outer_ring_mask);
filled_inner_ring_mask = imfill3(inner_ring_mask);

% Extract background mask
bg_mask = zeros((size(lab)));
bg_vals = filled_outer_ring_mask == 0;
bg_mask(bg_vals) = 1;
label_mask(bg_vals) = label_idx;
label_idx = label_idx+1;

% Create an 3D mask containing only outer ring
outer_img_mask = img;
outer_vals = filled_inner_ring_mask == 1;
outer_img_mask(outer_vals) = 0;
% Segment the 3D mask into 3 classes
L = imsegkmeans3(outer_img_mask, 3);

%Extract the mask of Class 1 from the segmented mask, and
update
% final segmented label mask
t_outer_mask = L==1 | imcomplement(bg_mask);
outer_ring_mask = zeros((size(lab)));
outer_ring_vals = L == 1;
outer_ring_mask(outer_ring_vals) = 1;
outer_ring_mask = imcomplement(outer_ring_mask);
outer_ring_label_vals = outer_ring_mask == 1;
label_mask(outer_ring_label_vals)=label_idx;
label_idx = label_idx+1;

%Extract the mask of Class 2 from the segmented mask, and
update
% final segmented label mask
inner_ring_mask = zeros((size(lab)));
inner_ring_vals = L == 2;
inner_ring_mask(outer_ring_vals) = 1;
bg_vals = bg_mask == 1;
inner_ring_mask(bg_vals) = 0;
inner_vals = filled_inner_ring_mask ==1;
inner_ring_mask(inner_vals) = 0;
inner_mask_vals = inner_ring_mask == 1;
label_mask(inner_mask_vals)=label_idx;
```

```
label_idx = label_idx+1;

% Create a 3D mask containing only inner components
se = strel('disk', 5);
inner_img_mask = img;
inner_vals = imdilate(filled_inner_ring_mask, se)==0;
inner_img_mask(inner_vals) = 0;
% Perform 3D K Means segmentation on inner matter mask
L_inner = imsegkmeans3(inner_img_mask, 4);

% Iterate through each of classes 3-5
for i=3:5
    % Create ground truth mask for ith class
    gt_mask = zeros((size(lab)));
    gt_vals = lab == i;
    gt_mask(gt_vals) = 1;
    for j=1:4   % Iterate through each k means segmentation
class
        % Extract mask corresponding to jth k means class
        mask = zeros((size(lab)));
        vals = L_inner == j;
        mask(vals) = 1;
        % Compute SSIM between GT mask and predicted k means
mask
        [ssimval,ssimmap] = ssim(gt_mask, mask);
        if ssimval>0.8  % Check if match is greater than 80%
            % If match is greater than 80%, then create
predicted mask
            label_mask(vals) = label_idx;
            label_idx = label_idx+1;
            break;
        end
    end
end

% Calculate metrics to compare ground truth and segmentation
mask
similarity = jaccard(categorical(lab),
categorical(label_mask));
dice_score = dice(categorical(lab),
categorical(label_mask));
ssim_score = ssim_scores(lab, label_mask);
[precision, recall, f1_score, acc] = pr(lab, label_mask);
mean_score = (similarity + f1_score + ssim_score + acc) / 4;
mean_val = mean(mean_score);

% Show final ground truth and segmented result
figure();
volshow(label_mask);
figure();
volshow(lab);
```