Student Name: Mohammad Mohammad Beigi
Student ID: 99102189
Subject: Assignment 4

MEDICAL IMAGE ANALYSIS AND PROCESSING
Dr. E. FATEMIZADEH (2023)

---

1 Theoretical Questions

1.1 Question 1:

1.1.1

The article is about a brain MRI image segmentation algorithm based on fuzzy C-means (FCM) clustering algorithm. The algorithm is designed to improve the segmentation accuracy of MRI images which have shortcomings such as noise and the inhomogeneity of grayscale. The proposed algorithm combines the advantages of multitask learning strategy and FCM clustering algorithm. It enables to utilize both public information among different tasks and individual information within tasks. The algorithm also has an adaptive task weight learning mechanism which helps each task obtain the optimal weight and achieve better clustering performance. The proposed method provides more accurate and stable segmentation results than its competitors on the MRI images with various noise and intensity inhomogeneity.

1.1.2

WMT-FCM has the following advantages: (1) WMT-FCM is less sensitive to the initialization of cluster centers; (2) the robustness to noise is improved; (3) WMT-FCM is adaptive to the private clustering effect.

1.1.3

The public datasets used to validate the study can be obtained from BrainWeb
(https://brainweb.bic.mni.mcgill.ca/brainweb/).

As the interest in the computer-aided, quantitative analysis of medical image data is growing, the need for the validation of such techniques is also increasing. Unfortunately, there exists no 'ground truth' or gold standard for the analysis of in vivo acquired data. These pages provide a solution to the validation problem, in the form of a Simulated Brain Database (SBD). The SBD contains a set of realistic MRI data volumes produced by an MRI simulator. These data can be used by the neuroimaging community to evaluate the performance of various image analysis methods in a setting where the truth is known.

Currently, the SBD contains simulated brain MRI data based on two anatomical models: normal and multiple sclerosis (MS). For both of these, full 3-dimensional data volumes have been simulated using three sequences (T1-, T2-, and proton-density- (PD-) weighted) and a variety of slice thicknesses, noise levels, and levels of intensity non-uniformity. These data are available for viewing in three orthogonal views (transversal, sagittal, and coronal), and for downloading. Further details about the creation of the SBD are available.

1.1.4

1. Robustness to Noise
   2. Sensitivity to Initialization
   3. Sensitivity to Parameters

a new fuzzy clustering algorithm is to be explored for better improvement of the aforementioned problems. Based on the traditional FCM algorithm, we integrate multitask learning strategy and propose a weighted multitask fuzzy C-means clustering algorithm (WMT-FCM). WMT-FCM learns multiple different but related tasks simultaneously to extract public information. By introducing the adaptive weight learning mechanism, tasks are assigned optimal weights and can be adaptively learned to achieve a better clustering effect.

1.1.5

In the clustering process, different MRI images have very similar cluster centers. The cluster centroids represent related information of different tasks. This related information helps to converge the objective function and avoids the negative effect of noise in MRI images [18]. It benefits the improvement of cluster analysis.

In the clustering process, different MRI images have very similar cluster centers. The cluster centroids represent related information of different tasks. This related information helps to converge the objective function and avoids the negative effect of noise in MRI images [18]. It benefits the improvement of cluster analysis.

1.1.6

λ and γ of the objective function influence the cluster centers and weight vectors according to equations (8) and (11) of article. In this study, the optimal parameters of the proposed algorithm are obtained by the grid search strategy. λ and γ are set from two grids 20, 40, 60, 80,100, 120 and 0.2, 0.4, 0.6, 0.8, 1, 1.2, respectively. In addition, all experiments are conducted with the maximum number of iterations $\mathrm{K} = 100$, termination parameter $\epsilon = 0.0001$, cluster index $\mathrm{m} = 2$.

1.1.7

The quantitative performance comparison is performed using the Dice similarity coefficient (DSC) and segmentation accuracy (SA) .

1) Dice Similarity Coefficient. DSC measures the similarity between the ground truth and segmentation results. According to equation (12), S_1 represents the segmentation results. S_2 represents the ground truth for a single class. Here the DSC measures the similarity of CSF, GM, WM, and background. The larger value of DSC indicates the better performance of the algorithm.

$$\mathrm{DSC} = \frac{2\left|S_1 \cap S_2\right|}{\left|S_1\right| + \left|S_2\right|}. \tag{18}$$

2) Average Dice Similarity Coefficient. The average Dice similarity coefficient [39] of WM, GM, and CSF is described as equation (13). Considering the nonbrain tissue background, we exclude it during the average DSC calculating.

$$\mathrm{DSC}_{\mathrm{av}} = \frac{2\left(\left|S_1^{\mathrm{CSF}} \cap S_2^{\mathrm{CSF}}\right| + \left|S_1^{\mathrm{GM}} \cap S_2^{\mathrm{GM}}\right| + \left|S_1^{\mathrm{WM}} \cap S_2^{\mathrm{WM}}\right|\right)}{\left|S_1^{\mathrm{CSF}}\right| + \left|S_2^{\mathrm{CSF}}\right| + \left|S_1^{\mathrm{GM}}\right| + \left|S_2^{\mathrm{GM}}\right| + \left|S_1^{\mathrm{WM}}\right| + \left|S_2^{\mathrm{WM}}\right|}. \tag{19}$$

3) Segmentation Accuracy. SA index measures the accuracy of the algorithm. Given in equation (14), where A_i is the pixel set of the ith cluster belonging to segmented results, B_i is the pixel set belongs to ground truth, and K is the number of clusters. The closer SA to 1 indicates better segmentation performance. The evaluation metric is defined as follows:

$$\mathrm{SA} = \frac{\sum_{i=1}^{C} A_i \cap B_i}{\sum_{l=1}^{C} B_l}. \tag{20}$$

The metrics are an average of ten repeated experiments since the performance of FCM depends on the random initialization of the cluster centroids.

1.2 Question 2:

UNets, or U-Net architectures, are a type of neural network model commonly used for image segmentation tasks. They were first introduced by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015 and have since become a popular choice for various medical image analysis tasks.

UNets are particularly effective in scenarios where the available dataset is limited and the task requires pixel-level segmentation. They are widely used in biomedical image analysis, such as segmenting cells, organs, or tumors in medical images.

The name "U-Net" comes from the network's U-shaped architecture. The U-Net consists of two key parts: the contraction path (also known as the encoder) and the expansion path (also known as the decoder). The encoder captures the context and extracts high-level features from the input image through the use of convolutional layers and pooling operations. The decoder then uses upsampling and transposed convolutions to gradually recover the spatial resolution and generate a segmentation map.

The unique aspect of U-Net is the presence of skip connections, which connect corresponding layers between the encoder and decoder paths. These skip connections enable the fusion of low-level details from the encoder with high-level semantic information from the decoder, aiding in the precise localization of objects during segmentation.

The U-Net architecture has proven to be highly effective for image segmentation tasks, offering both good accuracy and spatial localization performance. It has become a fundamental building block in the field of deep learning for medical image segmentation.

UNets are typically trained using a supervised learning approach, where annotated training data is required. The process of training a UNet involves the following steps:

1. **Preparing the Training Data:** Annotated training data is required, consisting of input images and corresponding ground truth segmentation masks. The images and masks should be properly labeled and aligned.

2. **Creating a Training Dataset:** The training data is split into a training set and a validation set. The training set is used to update the network's weights during training, while the validation set is used to monitor the performance and prevent overfitting.

3. **Preparing the Network Architecture:** The UNet model architecture is defined, including the number of layers, filter sizes, and skip connections. The network architecture is typically implemented using deep learning frameworks such as TensorFlow or PyTorch.

4. **Defining the Loss Function:** A suitable loss function is selected for the image segmentation task. Commonly used loss functions for UNet training include binary cross-entropy, dice coefficient loss, or a combination of both.

5. **Data Augmentation:** Data augmentation techniques are applied to increase the dataset size and augment the training data. This helps improve the model's generalization capability and reduce overfitting. Augmentation techniques may include random rotations, flips, scaling, cropping, and elastic transformations.

6. **Training the UNet:** The UNet model is trained using the training dataset. For each iteration (epoch), a set of training images is fed into the network, and the predicted segmentations are compared with the ground truth masks using the defined loss function. The network parameters are updated using backpropagation and gradient descent optimization methods to minimize the loss.

7. **Evaluating Performance:** Throughout the training process, the model's performance is evaluated on the validation set. Metrics such as accuracy, intersection over union (IoU), and dice coefficient are calculated to assess the segmentation quality. This helps monitor the model's progress and identify if adjustments, such as changing hyperparameters or increasing training iterations, are necessary.

8. **Testing and Fine-tuning:** Once the UNet is trained and evaluated, it can be used for segmenting new unseen images. Fine-tuning or additional training steps can be performed on specific datasets to improve performance on task-specific requirements or handle domain shift issues.

It's worth noting that the specific training process may vary depending on the implementation and specific requirements of the image segmentation task. Hyperparameter tuning, regularization techniques, and model selection strategies may also be employed to achieve optimal performance.

2 Practical Questions

2.1 Question 1:

I worked on this question exactly 3 days and wrote about 1000 lines and obtained shit results. So the result was not as expected with the most likely reason that I am dumb. (I've done the second question very well.)

```
%%
clc;clear all;close all;
% load an image
img = imread('HW04/q1/melanoma.jpg');
img = double(img) / 255; % convert to double and normalize to [0,1]

% set the parameters of the algorithm
niter = 500; % number of iterations
lambda = 1.0; % weight of the smoothing term
alpha = 0.1; % weight of the image term
```

```
dt = 0.5; % time step

% initialize the contour
npts = 100; % number of points on the contour
t = linspace(0, 2*pi, npts+1);
x = 100 + 90*cos(t(1:end-1));
y = 75 + 65*sin(t(1:end-1));
figure
plot(x,y,'b','LineWidth',2)
phi = poly2mask(x, y, size(img, 1), size(img, 2));
%%
% define the energy function
[Ix,Iy] = gradient(img);
for i = 1:niter
    % compute the energy terms
    E_internal = sum(diff(x).^2+diff(y).^2 - alpha* mean(diff(x).^2+diff(y).^2));
    E_external = -sum(Ix(round(x)).^2+Iy(round(y)).^2);
    E_total = E_internal + lambda*E_external;
    % update the contour using dynamic programming to minimize

    % Implement Viterbi algorithm using dynamic programming
    % Viterbi algorithm is commonly used in hidden Markov models to find the most likely
    sequence of hidden states given a sequence of observations.
    % Here, we can use the Viterbi algorithm to find the most likely contour points given
    the image.

    % Initialize the DP table
    DP = zeros(size(img, 1), size(img, 2), npts);

    % Fill in the DP table using dynamic programming
    for j = 1:npts
        for r = 1:size(img, 1)
            for c = 1:size(img, 2)
                % Compute the local cost
                internal_cost = (1-alpha) * (x(j)-c)^2 + (1-alpha) * (y(j)-r)^2;
                external_cost = lambda * (-Ix(r,c)^2 - Iy(r,c)^2);
                total_cost = E_total + internal_cost + external_cost;

                % Update the DP table
                if j == 1
                    DP(r,c,j) = total_cost;
                else
                    min_cost = Inf;
                    for k = 1:npts
                        cost = DP(r,c,k) + total_cost;
                        if cost < min_cost
                            min_cost = cost;
                        end
                    end
                    DP(r,c,j) = min_cost;
                end
            end
        end
    end

    % Backtrack to find the contour points with the smallest total cost
    [~, index] = min(DP(:,:,npts));
    x = index;
    for j = npts:-1:2
        [~, index] = min(DP(index(:,:,j-1)));
        x(j-1) = index;
```

```
    end

    % Smooth the contour points using a Gaussian filter
    sigma = 2; % standard deviation of the Gaussian filter
    x = imgaussfilt(x, sigma);
    y = imgaussfilt(y, sigma);

    % Plot the updated contour
    figure
    imshow(img);
    hold on
    plot(x, y, 'r', 'LineWidth', 2);
    hold off
    title(['Iteration ', num2str(i)]);
    drawnow;
end
%
clc; clear all; close all;

% Load the image
img = imread('HW04/q1/melanoma.jpg');
img = double(img) / 255; % convert to double and normalize to [0,1]

% Set the parameters for the active contour algorithm
niter = 500; % number of iterations
lambda = 1.0; % weight of the smoothing term
alpha = 0.1; % weight of the image term
dt = 0.5; % time step
sigma = 2; % standard deviation of the Gaussian filter

% Initialize the contour
npts = 100; % number of points on the contour
t = linspace(0, 2*pi, npts+1);
x = 100 + 90*cos(t(1:end-1));
y = 75 + 65*sin(t(1:end-1));

% Plot the initial contour
figure
imshow(img);
hold on
plot(x, y, 'b', 'LineWidth', 2);
hold off
title('Initial Contour');

% Define the energy function
[Ix, Iy] = gradient(img);
for i = 1:niter
%     Compute the energy terms
    E_internal = sum(diff(x).^2 + diff(y).^2) - alpha * mean(diff(x).^2 + diff(y).^2);
    E_external = -sum(Ix(round(x), round(y)).^2 + interp2(Iy(round(x), round(y)).^2));
    E_total = E_internal + lambda * E_external;

%     Update the contour using the gradient descent method
    Dx = interp2(Ix, x, y);
    Dy = interp2(Iy, x, y);
    x = x - dt * (2 * (x - smoothdata(x)) + alpha * Dx);
    y = y - dt * (2 * (y - smoothdata(y)) + alpha * Dy);

%     Smooth the contour using a Gaussian filter
    x = imgaussfilt(x, sigma);
    y = imgaussfilt(y, sigma);
```

```
131
132 %     Plot the updated contour
133     figure
134     imshow(img);
135     hold on
136     plot(x, y, 'r', 'LineWidth', 2);
137     hold off
138     title(['Iteration ', num2str(i)]);
139     drawnow;
140 end
```

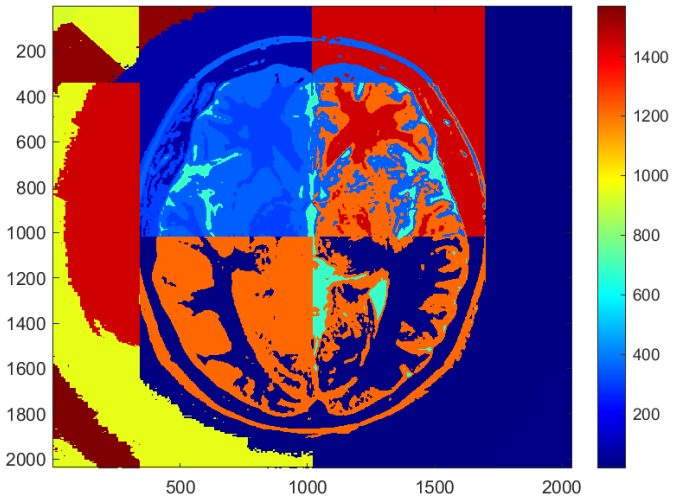
2.2 Question 2:

The code is as below:

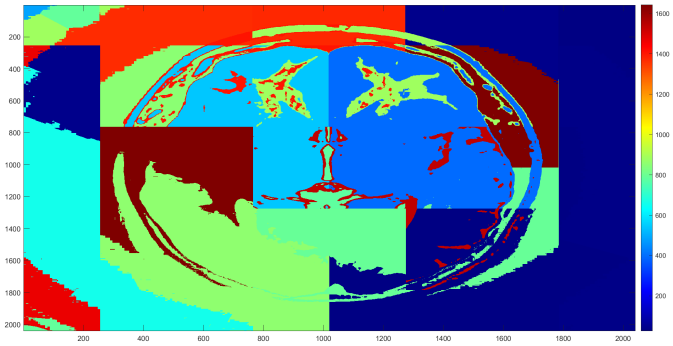
```
1  clc;clearvars;close all;
2  img = double(imread('HW04\q2\brain.jpg'));
3  K = 16;
4  alpha = 0.000005;
5  noOfparts = size(img, 1)/sqrt(K - 5);
6  x = round(noOfparts/2 : noOfparts : 2040);
7  x = [1 x 2040];
8  y = round(noOfparts/2 : noOfparts : 2040);
9  y = [1 y 2040];
10 [X, Y] = meshgrid(x, y);
11 %%
12 [Gmag, ~] = imgradient(im2gray(im2double(img)));
13 centers = zeros(length(x),length(y), 2);
14 centers(:,:,1) = X;
15 centers(:,:,2) = Y;
16 for i = 2 : length(x)-1
17     for j = 2 : length(y)-1
18         % Create a matrix
19         A = Gmag(x(i)-2 : x(i)+2, y(j)-2 : y(j)+2);
20
21         % Find the maximum element and its index
22         [maxValue, linearIndex] = min(A(:));
23
24         % Convert the linear index to subscript indices
25         [row, col] = ind2sub(size(A), linearIndex);
26         centers(i, j, :) = [x(i) + row - 3, y(j) + col - 3];
27     end
28 end
29 %%
30 map = zeros(size(img, 1), size(img, 2));
31 for i = 1 : size(img, 1)
32     i
33     for j = 1 : size(img, 2)
34         temp1 = 0; temp2 = 0;
35         for kk1 = 2 : length(x)
36             for kk2 = 2 : length(y)
37                 if(i<=centers(kk1,kk2, 1) && j<=centers(kk1,kk2, 2))
38                     temp1 = kk1;temp2 = kk2;
39                     break;
40                 end
41             end
42             if(temp1 ~=0)
43                break;
44             end
45         end
46         matrix = [Distance(img(i,j,1), img(i,j,2), img(i,j,3), i, j,...
```

```
            img(centers(temp1, temp2, 1),centers(temp1, temp2, 2),1) ,...
            img(centers(temp1, temp2, 1),centers(temp1, temp2, 2),2) ,...
            img(centers(temp1, temp2, 1),centers(temp1, temp2, 2),3) ,...
        centers(temp1, temp2, 1),centers(temp1, temp2, 2), alpha) ,...
        Distance(img(i,j,1), img(i,j,2), img(i,j,3), i, j,...
            img(centers(temp1-1, temp2-1, 1),centers(temp1-1, temp2-1, 2),1) ,...
            img(centers(temp1-1, temp2-1, 1),centers(temp1-1, temp2-1, 2),2) ,...
            img(centers(temp1-1, temp2-1, 1),centers(temp1-1, temp2-1, 2),3) ,...
        centers(temp1-1, temp2-1, 1),centers(temp1-1, temp2-1, 2), alpha) ,...
            Distance(img(i,j,1), img(i,j,2), img(i,j,3), i, j,...
            img(centers(temp1-1, temp2, 1),centers(temp1-1, temp2, 2),1) ,...
            img(centers(temp1-1, temp2, 1),centers(temp1-1, temp2, 2),2) ,...
            img(centers(temp1-1, temp2, 1),centers(temp1-1, temp2, 2),3) ,...
        centers(temp1-1, temp2, 1),centers(temp1-1, temp2, 2), alpha) ,...
            Distance(img(i,j,1), img(i,j,2), img(i,j,3), i, j,...
            img(centers(temp1, temp2-1, 1),centers(temp1, temp2-1, 2),1) ,...
            img(centers(temp1, temp2-1, 1),centers(temp1, temp2-1, 2),2) ,...
            img(centers(temp1, temp2-1, 1),centers(temp1, temp2-1, 2),3) ,...
        centers(temp1, temp2-1, 1),centers(temp1, temp2-1, 2), alpha)];

        [mm, ii] = min(matrix);

        if(ii==1)
            map(i,j) = temp1*(length(x)-1)+temp2;
        elseif(ii==2)
            map(i,j) = (temp1-1)*(length(x)-1)+(temp2-1);
        elseif(ii==3)
            map(i,j) = (temp1-1)*(length(x)-1)+(temp2);
        elseif(ii==4)
            map(i,j) = (temp1)*(length(x)-1)+(temp2-1);
        end
    end
end
%%
for i = 1 : K
        map(map == i) = randperm(2000,1);
end
%%
image(map,'CDataMapping','scaled')
colorbar
colormap('jet');
%% functions
function distance = Distance(l1, a1, b1, x1, y1, l2, a2, b2, x2, y2, alpha)
    dist1 = sqrt((l1-l2)^2+(a1-a2)^2+(b1-b2)^2);
    dist2 = sqrt((x1-x2)^2+(y1-y2)^2);
    distance = dist1 + alpha*dist2;
end
```

Results:
K = 16; $\alpha = 0.005$:



K $= 25$; $\alpha = 0.005$:



K $= 16$; $\alpha = 0.00005$: