Instructions

You have 1h 45 minutes. You have to submit your files by 10.00AM.
Any late submission will result in a penalty of -5 points (total is 100 points) for every minute.

Notify one of the assistants before leaving the room.

Once you are done, make sure that your files are successfully uploaded to the system by checking with one of the assistants.

Access to Internet resources is forbidden.

There are two exercises, and they do not depend on each other. If you get stuck in one of them, you can proceed to the other. In all, you are asked to implement three functions:

1. ***computeTask1AFeatures.m***, for Task 1A,
2. ***computeTask1BFeatures.m***, for Task 1B,
3. ***region_growing.m***, for Task 2A.

We give instructions as follows.

Exercise 1

In this exercise you have to detect synapses in biomedical microscopy image. A sample image is given below with the ground truth annotation in which white pixels indicate synapses while the black pixels indicate background.

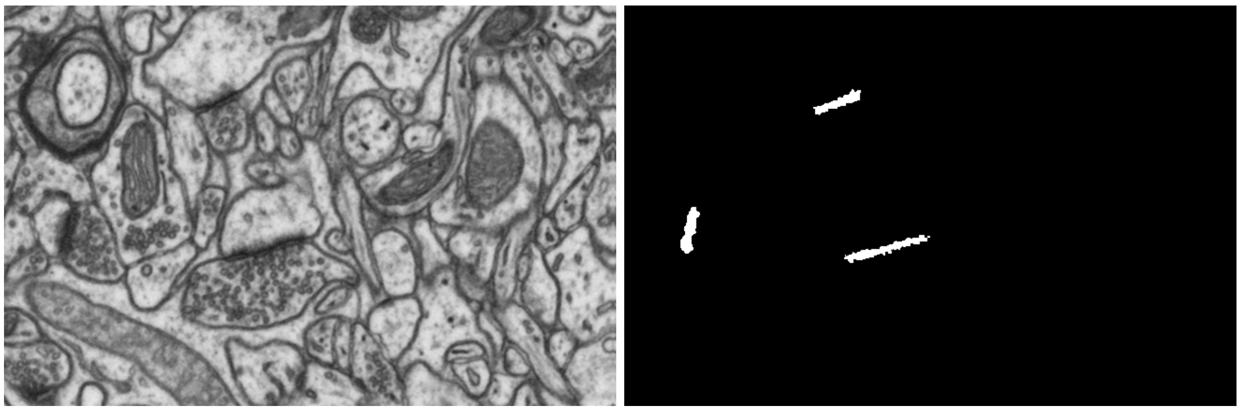


Image Ground truth

First, download the file named *graded_ex_1.zip* and decompress it. Then, open MATLAB and run the file called *main_exercise.m*. It will automatically load two images. One of them will be used to *train* the algorithm which you will develop and the second one is for *testing*.

The task is to determine whether a pixel belongs to a synapse (positive label) or not (negative label). You will notice that synapses are dark structures that look like dark line segments in the image.

We will use a machine learning algorithm named *AdaBoost* to train classifier to detect the synapses. You are asked to implement the features that the classifier will use to predict the label of a pixel.

Task 1A

You are first asked to try three different measures to detect them, applying the following filters:

1. Gaussian smoothing
2. Smoothed gradient magnitude
3. Laplacian of Gaussian

You can use the *fspecial()* MATLAB function to construct the necessary intermediate filters to compute the output image for each case.

You are asked to compute the output of each feature of different *sigma* values, namely

sigma = [1.0, 2.0, 4.0]

Therefore, after implementing the code for task 1A you should have computed 3 * 3 = 9 different outputs, generated from the original image provided.

You have to implement the required functionality in the file ***<u>computeTask1AFeatures.m</u>***, which takes an image as its input and should produce an output matrix **M** of size Nx9, where N is the number of pixels and,

M(K,1) = value of gaussian smoothed image sigma=1.0 at pixel K
M(K,2) = value of gaussian smoothed image sigma=2.0 at pixel K
M(K,3) = value of gaussian smoothed image sigma=4.0 at pixel K
M(K,4) = value of smoothed gradient magnitude, sigma=1.0 at pixel K
M(K,5) = value of smoothed gradient magnitude, sigma=2.0 at pixel K
M(K,6) = value of smoothed gradient magnitude, sigma=4.0 at pixel K
M(K,7) = value of laplacian of gaussian, sigma=1.0 at pixel K
M(K,8) = value of laplacian of gaussian, sigma=2.0 at pixel K
M(K,9) = value of laplacian of gaussian, sigma=4.0 at pixel K

Task 1B

You are asked to implement a new feature to measure the *ridgeness* of an image pixel and its neighborhood, which could be helpful to detect the synapses. The computation of this feature will be based on the eigenvalues of the Hessian matrix:

```
H = [ Ixx Ixy; Ixy Iyy ];
```

where Ixx and Iyy are the second order image derivatives along the x and the y axis, respectively.

For dark structures on a bright background, a frequently used ridgeness measure is defined as the absolute value of the maximum eigenvalue. In this exercise, we will feed the two eigenvalues to *AdaBoost*, which are defined as:

```
lamda_1 = 0.5 * ( Ixx + Iyy - sqrt( (Ixx - Iyy).^2 + 4*Ixy.^2 ) );
lamda_2 = 0.5 * ( Ixx + Iyy + sqrt( (Ixx - Iyy).^2 + 4*Ixy.^2 ) );
```

Prior to computing the image derivatives, we will smooth the image with a Gaussian filter of certain standard deviation (sigma) in order to get rid of the background noise and obtain a smooth response on the elongated structures.

You have to implement the required functionality in the file ***computeTask1BFeatures.m***, which takes an image as its input and should produce an output matrix **M** of size Nx6, where N is the number of pixels and

M(K,1) = value of lambda_1 with sigma=1.0 at pixel K
M(K,2) = value of lambda_2 with sigma=1.0 at pixel K
M(K,3) = value of lambda_1 with sigma=2.0 at pixel K
M(K,4) = value of lambda_2 with sigma=2.0 at pixel K
M(K,5) = value of lambda_1 with sigma=4.0 at pixel K
M(K,6) = value of lambda_2 with sigma=4.0 at pixel K

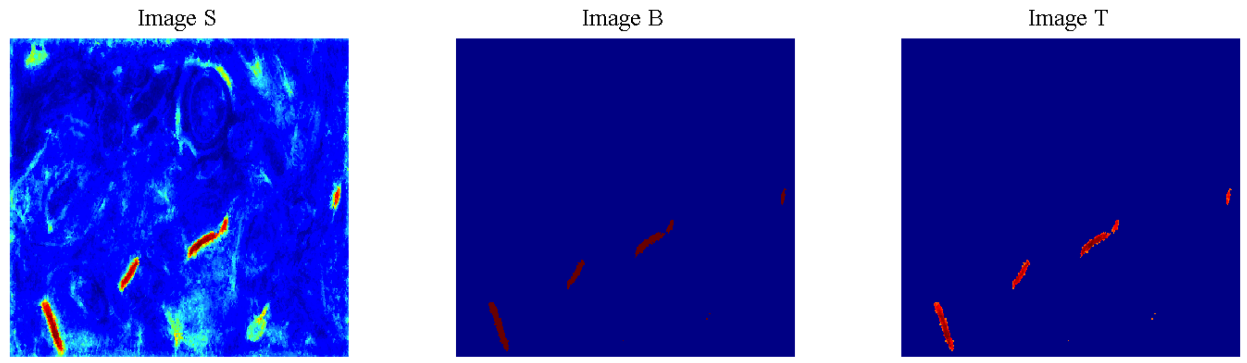
Exercise 2

Once we have implemented the features as described in Task 1, we use them as input and apply *AdaBoost* algorithm to learn a classifier, which predicts if a pixel is part of the synapse or not. The classifier returns a score for each pixel being the part of the synapse, and the higher the score is, the more likely the pixel belongs to the synapse. Thus we obtain an image, which we call the score map S, whose values of pixels indicate the likelihood of being part of the synapse.

Note that, you have already been given the score map S, i.e., the *img1_scores.png* in the *./test* folder, which comes from a reliable machine learning algorithm.

Task 2A: Counting the Number of Synapses

In this part, we implement a function to count the number of detected synapses. In order to obtain the number of synapses, we should segment each synapse. In practice, we use non-maxima suppression on the thresholded score map T. More specifically, given the original score map S, we threshold on each pixel and obtain a binary image B, whose elements comprise of 1s and 0s, where 1 indicates the pixel value is above the threshold, and 0 otherwise. We conduct an element-wise multiplication of S and B and get the thresholded score map T, i.e., T = S .* B. Example of S, B and T can be visualized as follows.



Note that, the above images are generated using the *imagesc* function in matlab. In Image B, red pixels are of value 1, while blue are of value 0.

Then, we conduct non-maxima suppression on T to count the synapses. The steps are as follow:

1. Find the pixel (x,y) with the highest value on T.
2. Use (x,y) as an initial seed, apply the ***Region Growing Algorithm*** to find the region/segment of the synapse, wherein the seed (x,y) lies.
3. Mark the obtained region of the synapse, update T by setting the values of pixels inside the obtained region to be 0. If the region is large enough, we save it as a detected synapse; otherwise it is considered as noise.
4. Repeat steps 1 to 3, until the highest score of T is 0.

The function *nms.m* deals with non-maxima suppression, which is already given to you. You _don’t_ need to modify any code in it.

Your job is to complete the function ***region_growing.m***, which is about the region growing algorithm for segmenting synapses. The goal of the region growing algorithm is to find the segments of the synapses given a seed pixel. This approach examines neighboring pixels of the initial seed and determines whether the pixel neighbors should be added to the region. In our scenario, we grow the regions using the binary mask B. The workflow is as follows:

1. Given the seed pixel (x,y), do the following initialization:
    - a) a variable *neg_list*, to store the neighbors of the seed pixels, it should be initialized as empty;
    - b) a variable *seedVal* to store the value of pixel (x,y), since we are dealing with binary image B, set *seedVal* to be 1;
    - c) the output variable *region_mask*, it should be of the same size of B.
2. Loop while BOTH of the following two conditions are true:
    - a) the number of checked pixels is smaller than the total number of pixels in the image;
    - b) the *seedVal* is greater than 0.

--- LOOP BEGINS --

3. Get the 4 neighbors (up, down, left, right) pixels of the seed pixel (x,y), check whether these neighbor pixels are already in the *neg_list.* If no, add them in the neighbor list *neg_list*.
4. In the neighbor list *neg_list*, find the pixel with the highest value. Set it to be the seed pixel, then remove it from the *neg_list*. Also mark this pixel on the *region_mask.*

--- LOOP ENDS --

5. End of Loop, output the *region_mask*.

The output *region_mask* should contains 1s inside the region and 0s outside.

You should interpret the workflow described above into matlab code in the function ***region_growing.m***. Note that, you should NOT modify the parameters and the code in *nms.m*.