ECSE-415 Introduction to Computer Vision

Assignment #4: Image Segmentation

Due: Wednesday, March 30, 2016, 11:59pm

Please submit your assignment solutions electronically via the myCourses assignment dropbox. The solutions should be in PDF format. Attempt all parts of this assignment. Associated code files should be commented and submitted, so that they can be run for testing.

This assignment is out of a total of **100 points**. It is designed to familiarize the students with different clustering algorithms for image segmentation. The assignment requires the students to use Matlab MEX interface with OpenCV to apply different clustering methods on a set of given image.

The assignment has three parts. In the first part, the student is required to use the K-means clustering method to cluster a given image. In the second part, the student is asked to fit a Gaussian Mixture Model (GMM) to do the segmentation on the same image. In the third part, the student will perform segmentation on the image using Graph Cuts. The outputs of the three methods will be compared against a given ground truth segmentation.

The student is expected to write his/her own code. To submit your code, submit all the required .m and .cpp files. Assignments received up to 24 hours late will be penalized by 30%. Assignments received more than 24 hours late will not be marked.

Overview

In this assignment, the student is asked to perform foreground/background segmentation on a set of given images. You will explore using K-means, GMMs and Graph Cuts methods for image segmentation. To implement the methods, we use the Matlab's MEX interface to use OpenCV's functionalities directly in Matlab. You will write three MEX file, one for each method, which obtain their inputs from Matlab, perform the segmentation in C++, and return the output as a foreground mask back to Matlab. After obtaining the results, the performances of the methods are compared against ground truth. We will also explore using different feature spaces such as intensity and color for segmentation.

The Code Template

Please use the code template in Table 1 for writing your code. The main function loads the input and the ground truth segmentation image, displays the input image and prompt the user to draw a rectangle over the foreground object. The code passes the input image filename and the object's rectangle to the clustering function. In this assignment, we will use k-means, GMMs, and the graph cuts method to do segmentation on the input image. We will use the Matlab's MEX interface to use OpenCV's functionalities in implementing the three clustering methods. You will write three MEX file, one for each method, which take the image path and the user rectangle as the input and return a foreground mask as the output. Since we are dealing with MEX files, we need to compile them before using them on Matlab. You are required to write the necessary code to compile the three MEX files, in the Compile.m file. After obtaining the foreground masks from the three methods, the code proceeds by displaying and comparing the segmentation results for the three methods with the ground truth segmentation image on a subplot.

Table 1: Code template for Assignment 4

```
%% Compile the mex files
Compile;

%% Set the image name
imgName = '100_0109.png';
gtName  = '100_0109_groundtruth.png';

%% Load the input image
img = imread(imgName);

%% Load the ground truth image
gtMask = imread(gtName);

%% Prompt the user to draw a rectangle over the object
imshow(img);
rect = getrect;

%% Clustering using K-means
kmeansMask = kmeansMex(imgName,rect);

%% Clustering using GMM
gmmMask = GmmMex(imgName,rect);

%% Clustering using GraphCut
graphcutMask  = GraphCutMex(imgName,rect);

%% Display the segmentation results
```

1 kmeansMex (30 pts: 20 pts for the code, 10 pts for answers to the questions)

To implement K-means segmentation, you should use OpenCV's `kmeans` function, called in a MEX-file. Follow these steps in writing the `kmeansMex` MEX file:

- Create and store `kmeansMex.cpp` file in Matlab
- Add Matlab MEX API header, "mex.h", and the OpenCV header
- Create the gateway function
- Check to make sure that the function has two inputs and one output
  - Use nlhs and nlhs
- Check to make sure that the first input is a character string
  - Use mxIsChar function
- Check to make sure that the second input is 1 by 4 array of doubles

    - Use mxIsDouble, mxGetM, and mxGetN functions
- Use the first input to load the image
    - Use mxArrayToString function
- Use the second input to create a corresponding cv::Rect object
    - Use mxGetPr function
    - Note that indices start from 1 in Matlab, so you need to subtract 1 from the starting point of the input rectangle
- Create the feature matrix, as an HW by D `Mat` object
    - For the RGB color space, simply use the three color channels to represents the feature vector of each pixel as $M = \begin{bmatrix} r_{11} & g_{11} & b_{11} \\ r_{21} & g_{21} & b_{21} \\ & \vdots & \\ r_{ji} & g_{ji} & b_{ji} \\ & \vdots & \\ r_{HW} & g_{HW} & b_{HW} \end{bmatrix}_{HW \times 3}$, where r_{ji}, g_{ji} and b_{ji} denote the red, green and blue channels of the pixel in the jth row and in the ith column and H and W denote the height and the width of the image
    - Make sure that the feature matrix if of `CV_32F` type
- Create the corresponding initial label as an HW by 1 `Mat` object
    - Use the input rectangle and assign the initial labels of the points inside and outside of the rectangle to 1 and 0, respectively.
    - Make sure that the label matrix is of integer type
- Run the `kmeans` function

After convergence, use the labels returned by the `kmeans` function to create a foreground mask, as a H by W `Mat` object of `CV_8UC1` type, with the foreground pixels assigned to 1 (or 255) and the background pixels assigned to 0. The function should now create and allocate the output matrix and fill its data using the foreground matrix as follows:

- Create a 2-D H by W matrix of mxUINT8_CLASS and mxREAL type using the mxCreateNumericArray function
- Get a `char *` pointer to the first element of the matrix using the mxGetData function
    - Note that you should manually cast the returned `void *` pointer to a `char *` pointer
- Using a nested for loop, copy the value each pixel from the foreground `Mat` object to the `char *` pointer
    - Using the `char *` pointer, we can access all the output pixels in a column-wise order. E.g. Assuming the `char *` pointer is named outMatrix, outMatrix[0] is the pixel in the first row and first column, outMatrix[1] is the pixel in the second row and first column and so on.

For this part of the assignment, answer the following questions on your report:

i. Display the K-means segmentation results with two clusters for each input image (use the returned foreground matrix to set all the background pixels in the original RGB image to 0).

ii. Display the K-means segmentation results with two clusters with the feature matrix set to intensity space for each input image ($M = \begin{bmatrix} I_{11} \\ I_{21} \\ \vdots \\ I_{ji} \\ \vdots \\ I_{HW} \end{bmatrix}_{HW\times 1}$, where I_{ji} denotes the intensity of the pixel in the jth row and in the ith column). Based on your observations, which feature space (intensity or RGB) do you think is best to cluster the input image?

2 GmmMex (30 pts: 20 pts for the code, 10 pts for answers to the questions)

To implement GMM-based segmentation, you should use an object from the OpenCV `EM` class, in a MEX-file. Follow these steps in writing the `GmmMex` MEX file:

- Create and store `GmmMex.cpp` file in Matlab
- Add Matlab MEX API header, "mex.h", and the OpenCV header
- Create the gateway function
- Check to make sure that the function has two inputs and one output
- Check to make sure that the first input is a character string
- Check to make sure that the second input is 1 by 4 array of doubles
- Use the first input to load the image
- Use the second input to create a corresponding cv::Rect object
- Create an object from the OpenCV `EM` class (you can set the number of clusters, the type of the covariance matrix and the stop criteria using the class constructor)
- Create the feature matrix, as an HW by D `Mat` object of `CV_64F` type
- Create initial means, as an $numClusters$ by D `Mat` object of `CV_64F` type
    - Use the input rectangle to compute the mean of all background and foreground pixels, assuming the pixels inside the rectangle to be foreground.
    - You can use the OpenCV `calcCovarMatrix` function, with its flag set as CV_COVAR_NORMAL | CV_COVAR_ROWS.
- Create initial covariance matrices as a `vector<Mat>` object, each `CV_64F Mat` object representing the D by D covariance matrix of each cluster.
    - Use the input rectangle to compute the covariance of all background and foreground pixels, assuming the pixels inside the rectangle to be foreground.
    - You can use the OpenCV `calcCovarMatrix` function, with its flag set as CV_COVAR_NORMAL | CV_COVAR_ROWS.
- Create initial weight matrix of mixture component as a 1 by $numClusters$ floating-point `Mat` object

    - Use the input rectangle to compute the ratio of number of pixels inside and outside of the rectangle to the total number of pixels as initial mixture components.
- Use the `trainE` method of the `EM` object to estimate the GMM parameters and the pixel labels.

After convergence, use the returned labels to create a foreground mask, as a H by W `Mat` object of `CV_8UC1` type, with the foreground pixels assigned to 1 (or 255) and the background pixels assigned to 0. The function should now create and allocate the output matrix and fill its data using the foreground matrix as follows:

- Create a 2-D H by W matrix of mxUINT8_CLASS and mxREAL type using the mxCreateNumericArray function
- Get a `char *` pointer to the first element of the matrix using the mxGetData function
- Using a nested for loop, copy the value each pixel from the foreground `Mat` object to the `char *` pointer

For this part of the assignment, answer the following questions on your report:

iii. Display the GMM segmentation results with two clusters for each input image.
iv. Display the GMM segmentation results with two clusters with the feature matrix set to intensity space for each input image. Based on your observation, which feature space (intensity or RGB) do you think is best to cluster the input image?

3 GraphCutMex (40 pts: 20 pts for the code, 20 pts for answers to the questions)

To implement Graph Cuts segmentation, you should use the OpenCV `grabCut` function, in a MEX-file. Follow these steps in writing the `GraphCutMex` MEX file:

- Create and store `GmmMex.cpp` file in Matlab
- Add Matlab MEX API header, "mex.h", and the OpenCV header
- Create the gateway function
- Check to make sure that the function has two inputs and one output
- Check to make sure that the first input is a character string
- Check to make sure that the second input is 1 by 4 array of doubles
- Use the first input to load the image
- Use the second input to create a corresponding cv::Rect object
- Run the grabCut function, with the GC_INIT_WITH_RECT flag set
    - You need to create three empty `Mat` objects (mask, bgdModel, fgdModel) and pass them to the function. Their values are set within the function.

Note that grabCut function is an implementation of the Graph Cut segmentation which uses a GMM to assign the weights of edges connecting pixels to source node/end node by the probability of a pixel being foreground/background and the weights between the pixels are defined by the edge information or pixel similarity (if there is a large difference in pixel color, the edge between them will get a low weight).

After convergence, use the returned mask `Mat`, and create a foreground mask, as a H by W `Mat` object of `CV_8UC1` type, with its pixels set (1 or 255) wherever the corresponding pixel in the mask matrix equals to GC_FGD (you can use logical and operation between a `Mat` and a scalar to do this). The function should now create and allocate the output matrix and fill its data using the foreground matrix as follows:

- Create a 2-D H by W matrix of mxUINT8_CLASS and mxREAL type using the mxCreateNumericArray function
- Get a `char *` pointer to the first element of the matrix using the mxGetData function
- Using a nested for loop, copy the value each pixel from the foreground `Mat` object to the `char *` pointer

For this part of the assignment, answer the following questions on your report:

v. Display the Graph Cuts segmentation results for each input image.

vi. Display the Graph Cuts segmentation results for a gray-scaled version of the input image. Based on your observation, which feature space (intensity or RGB) do you think is best to cluster the input image?

vii. Display and compare the results of the three segmentation methods, against the ground truth segmentation image for each input image and each feature type. Which segmentation method/feature type is performing the best?