16-720 Computer Vision, Fall 2008 – Assignment 6
Clustering and Segmentation

Due: End of semester – Monday, December 15th, 2008 at midnight
absolutely no extension

1 Introduction

In computer vision, image segmentation is often used as a preprocessing step for higher level tasks such as object detection and object recognition. Therefore, one objective of image segmentation is to divide an image into a set of meaningful regions which are good enough for subsequent tasks. In the first part of this assignment, you will be implementing the Mean Shift algorithm for clustering and then apply it to the problem of color image segmentation. In the second part of the assignment, you will have to generate reasonably good segmentation results for a set of provided images. To achieve this goal, you can extend your implementation from the first part or you can implement a totally new segmentation algorithm.

2 Mean Shift (70 points)

2.1 Basic Implementation

The Mean Shift algorithm clusters a d-dimensional data set by associating each point to a peak of the data set's probability density function. For each point, Mean Shift computes its associated peak by first defining a spherical window at the data point of radius r and computing the mean of the points that lie within the window. Note that in the mean shift paper a kernel which assigns distance decaying weights to all other points is used, while we simply use a spherical window which gives all points within a distance r uniform weights and 0 to all other points. The algorithm then shifts the window to the mean and repeats until convergence ($\epsilon = .01$ works well). Within each iteration, the window will shift to a more densely populated portion of the data set until a peak is reached. You will implement this process as the function:

```
function peak = findpeak(data,idx,r)
```

where data is the d-dimensional data set ($d \times n$ matrix), idx is the column index of the data point for which we wish to compute its associated density peak and r is the search window radius. The algorithm's dependence on r will become apparent from the experiments performed below. Implement the mean shift function, which calls `findpeak` for each point and then assigns a label to each point according to its peak. This function should have the following prototype:

```
function [labels, peaks] = meanshift(data, r)
```

where `labels` are the peak labels (a vector of length n with each entry $\in [1, K]$ where K is the number of distinct peaks) and `peaks` is a $d \times K$ matrix storing the density peaks found using meanshift as its columns. Note that Mean Shift requires that peaks are compared after each call to `findpeak` and for similar peaks to be merged. For our implementation of Mean Shift, we will consider two peaks to be the same if the distance between them is $\leq \frac{r}{2}$. Also, if the peak of a data point is found to already exist in peaks then for simplicity its computed peak is discarded and it is given the label of the associated peak in `peaks`.

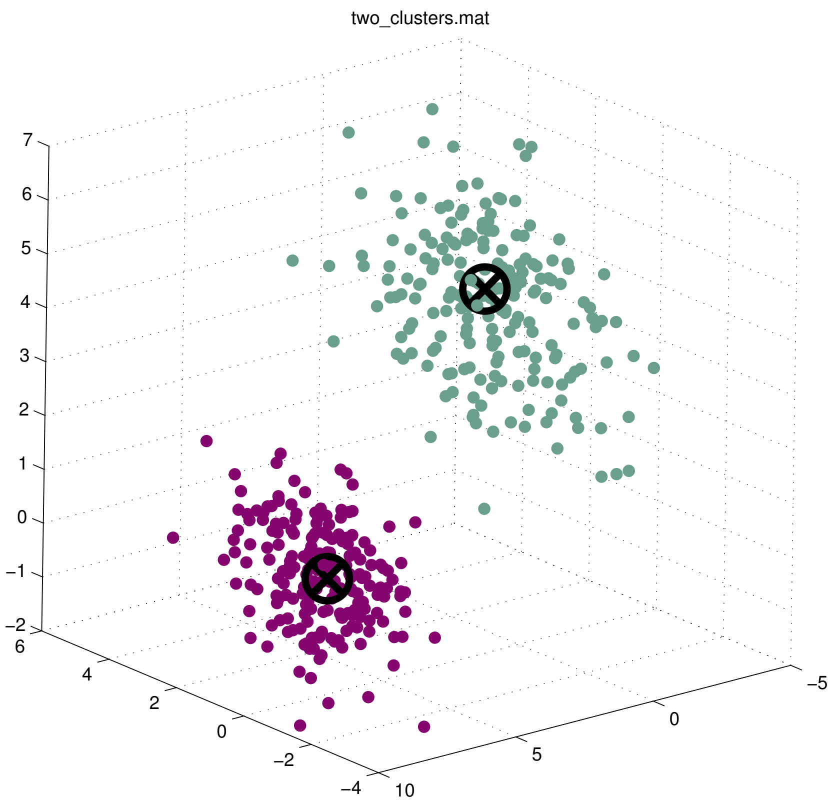

Figure 1: Mean Shift Clustering Result

Debug your algorithm using `two_clusters.mat` with $r = 2$ (this should give two clusters). Plot your result using the `plot3dclusters` function. The result of running `meanshift` on `two_clusters.mat` is depicted in Figure 1. If you are having doubts about your program working correctly, you can generate more synthetic data using the function `generatedata`.

2.2 Optimizations

Unfortunately, the Mean Shift algorithm you just implemented is too slow to be realized for image segmentation. We will therefore incorporate several speedups into our implementation.

2.2.1 MATLAB vectorization (avoid loops!)

First of all, you will definitely want to exploit MATLAB's ability to handle matrix operations efficiently. For example, if you want to find all data points with label equal to 1 and compute their mean, write the following:

```
currdata = data(:,labels == 1);
currmean = mean(currdata,2);
```

You will also be computing Euclidean distances from some **x** to all of your data very often, and it is crucial that you avoid writing `for` loops and computing distances one by one. We are providing you with a MATLAB function `ml_distSqr` that given two matrices of points (where points are the columns in the matrices) as input, will return a matrix of squared Euclidean distances between each of the elements in the input.

2.2.2 Basin and Trajectory of Attraction

The first non-MATLAB specific speedup will be to associate each data point that is at a distance $\leq r$ from the peak with the cluster denoted by that peak. This speedup is known as basin of attraction and is based on the intuition that points that are within one window size distance from the peak will with high probability converge to that peak.

The second speedup is based on a similar principle, where points that are within a distance of $\frac{r}{c}$ of the search path are associated with the converged peak, where c is some constant value. Incorporate the above speedups into your Mean Shift implementation by modifying your implementation from before. The resulting modified function should have the following prototypes:

```
function [labels, peaks] = meanshift_opt(data,r,c)
function [peak, cpts] = findpeak_opt(data,idx,r,c)
```

where `cpts` is a n dimensional boolean vector storing a 1 for each point that is within a distance of $\frac{r}{c}$ from the trajectory or within r from the final peak, 0 otherwise. Your implementation of `meanshift_opt` should produce the same result on `two_clusters.mat` with $r = 2$ and $c = 4$ as `meanshift` (the two clusters will start to mix if you let $c = .5$). You might want to generate some data using `generatedata` and try to cluster it. You will have the ground truth labels and can see how well your algorithm is doing. An example of such a clustering can be seen in Figure 2.

2.3 Image Segmentation

In this section you will build upon your optimized Mean Shift implementation to perform image segmentation. To do so, implement the function

```
function [segmIm,peakIm] = segment_meanshift(I,r,c)
```

In this function, I is a color input image, r is the radius used for our window, and c is the trajectory of attraction parameter. `segIm` is an image created by reshaping the `labels` into the size of the input image. `peakIm` is an image the same dimensions as I, but with each pixel colored by the RGB value of that pixel's associated peak. This function is constructed by reshaping the image into RGB vectors, converting them to Luv color-space representation, and then clustering the resulting color data using the optimized Mean Shift implemented earlier. Mean Shift clusters using the Euclidean distance metrics, and since Euclidean distance in RGB space does not correlate well to the perceived change in color we will use Luv color space. In `segment_meanshift`, convert to Luv color space by using the provided MATLAB function `rgb2luv`. Then convert the resulting cluster centers back to RGB using the function `luv2rgb`.

2.4 Results

First, you will need to cluster the 3D point clouds found in the files `testdata1.mat`, `testdata2.mat`, and `testdata3.mat`. You will need to somehow choose a good value for r and you should use $c = 4$. Produce three result files `results1.mat`, `results2.mat`, and `results3.mat` such that each file loads the variables `labels` and `peaks` (the output of `meanshift_opt`).

You will also have to submit some results on real images. Note that Mean Shift is still computationally expensive, even with these optimizations. Furthermore, the computational time depends on the parameters r and c. For example, with $r = 20, c = 1$, the starfish image in Figure 3 took 3 seconds to to segment; and it took 60 seconds to segment if $r = 10, c = 2$. In general, Mean Shift will take longer if r is small and c

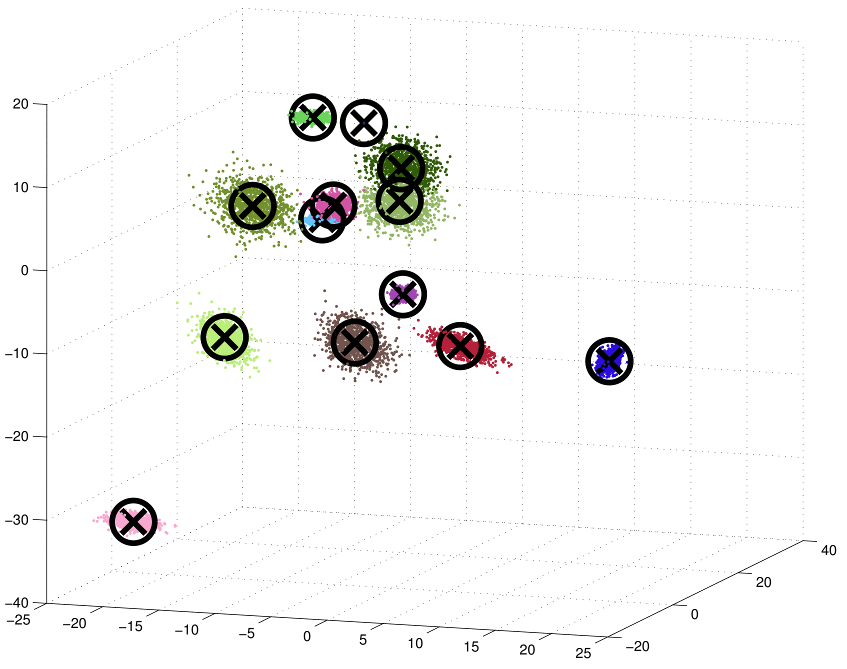

Figure 2: Mean Shift Segmentation of Point Cloud generated from generatedata function.

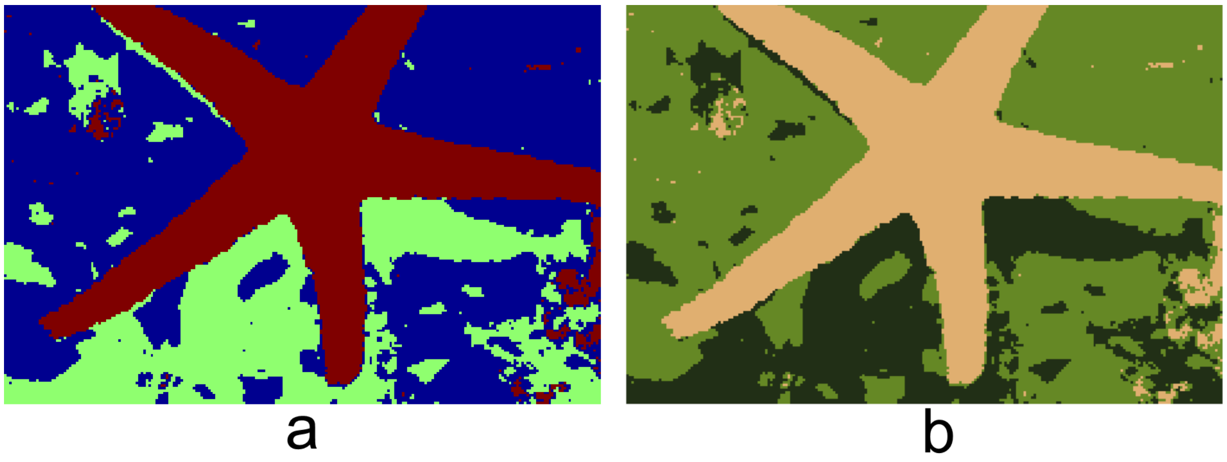

Figure 3: Mean Shift Clustering Result on Starfish Image. a) label image; b) peak image. This result is generated with $r = 20$ and $c = 1$.

is big. In this case, Mean Shift will generate lot of segments which might not be what we want. You will need to play around with different values of r and c. With relatively big r and small c, your algorithm should not take more than a few minutes to run. If it does, you are doing something wrong, or you need to stop using punchcard-based computers.

Segment the four `.png` images (hat,man,star,sheep) using your image segmentation algorithm. Experiment with different values of r and c so that you are able to segment these four images. Consider $r \in \{5, 10, 20\}$ and $c \in \{1, 2, 4\}$. What effect do c and r have on the run time of your algorithm? How about the visual quality of the peaks image? (You might want to down sample these images when debugging)

3 Segmentation for objects of interest (30 points)

In many computer vision applications, you might only be interested in a specific class(es) of objects (e.g. horses, faces). Therefore, your image segmentation algorithm needs to be tuned to work for that specific class. In this part of the homework, you will have to provide segmentation results for a set of provided images.

On the course web page, you can download the dataset that we are going to use for this homework. This dataset consists of 40 images; each image contains a single horse which is considered as the foreground object. Figure 4 displays some of these images. You will soon figure out that using your Mean Shift implementation with color features is not good enough for these images.

Your task is to extend your current Mean Shift implementation to achieve reasonably good results for the provided images. There are many ways you can extend your implementation, here are some:

- Using different features such as color, edges, contours, texture, steerable filters, pixel locations, etc. To achieve good results, you might consider combining different types of features.
- Adding post-processing steps to your algorithm such as grouping segments, dividing segments, tracing contours.
- Adding prior knowledge: shape prior, location prior.

Your extension is not limited to the above list. Moreover, you are not restricted to use the Mean Shift segmentation algorithm from the first part. You are free to propose your own or implement another existing image segmentation algorithm.

Hints: You should not use too many features. Irrelevant features might hurt the performance of your segmentation algorithm. Redundant features increase computational cost; Mean Shift (or Matlab) might explode if you use too many features.

Perfect segmentation is extremely hard and we do not expect the results of your segmentation algorithm to be close to the groundtruth.

3.1 Algorithm design

Q1: List the features used by your algorithm, explain why you think they are suitable for this task.

Q2: If you extend the Mean Shift algorithm, briefly explain the extension. If implement a different image segmentation algorithm, briefly describe it. You need to highlight the intuitions/ideas/theories underlying it. Your answer to Q2 should exclude things in the answer of Q1.

3.2 Implementation

Q3. Write the following function:

```
imSegment(input_image_list, path_to_output_directory)
```

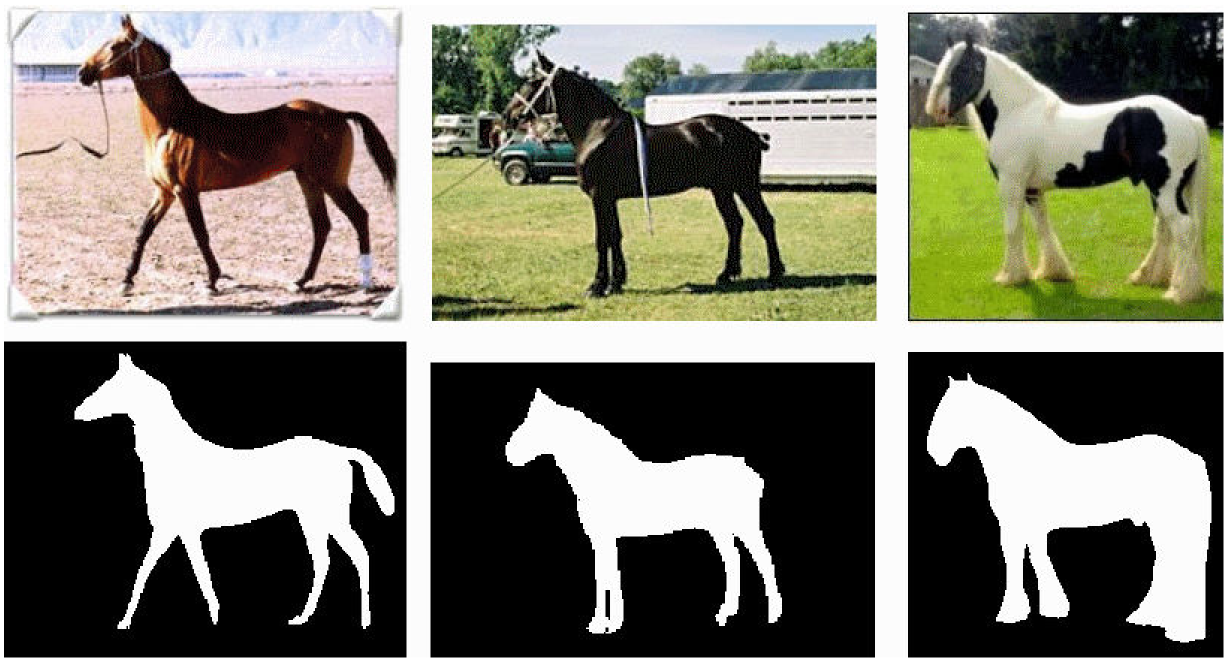

Figure 4: First row: example of horse images. Second row: the groundtruth foreground masks for the corresponding images in the first row; ideally, the segmentation results should look like the groundtruth masks. However, perfect or close-to-perfect segmentation is SUPER hard. There is no perfect solution to this and we do not expect things close to the ground truth.

Running this function will automatically perform image segmentation on a set of images and output the results to a specified directory. `input_image_list` is a cell array of strings that specify the full path of images that one wants to perform segmentation. The other input argument, `path_to_output_directory`, is the output directory to store segmentation results. For each input image, your function needs to generate an output segmentation image. The output image must clearly show the segmentation results. Each segment must be associated with one unique color; different segments must be given different colors. The name of the output file should only differ from the input file by the suffix *_seg*. For example, if the input image is *image-032.png*, the output must be named *image-032_seg.png* and placed in the directory `path_to_output_directory`.

If your code requires pre-computed data (e.g. parameters, classifiers) that cannot be conveniently stored in the code, you can store them in *.mat* files in the same directory with your code. When called, your function must automatically load any necessary data.

Hint: The Matlab command 'saveas' and 'gcf' are quite handy for automatically saving displayed figures.

3.3 Experiments and tuning

Q4. Run your algorithm on the images listed in **data/horses/**, and submit the segmentation results together with your code. The segmentation images should be placed in **results/horses/**

Q5. As with most computer vision applications, your implementation might not work right away. You are expected to spend a substantially amount of time tuning your function (e.g. size of filters, threshold). To answer this question, you will have to describe how you tune your algorithm.

You are free to used all the images for tuning. You can also use the foreground masks that come with the homework (the second row of Figure 4 shows some examples of foreground masks). One way to use these foreground masks is to design a quantitative evaluation and use it for automatic tuning. However, you are not required to do so. Yon can tune your method by qualitative evaluation. Whatever method you choose, you have to document it when answering this question.

3.4 Face segmentation challenge (20 bonus points)

Suppose we are interested in detecting the outer contours of the faces. One idea is to combine the output of a face detector and the an image segmentation algorithm. Given an image, the first step is to run a face detector to find the rectangle region that roughly correspond to the face location. Because the output of the face detector is usually smaller than the face, we enlarge the face detector output so that the enlarged box covers the whole face. The second step is to run an image segmentation algorithm on the enlarged face detector region. The third step is to use the segmented image to find the face contours.

For this part of the homework, you will have a chance to experiment with this idea and gain some bonus points. The first step of this algorithm has been done for you; the enlarged face detector output regions have been provided as images in the directory **data/faces/**. Your task is to achieve good segmentation results on these images. We are mostly interested in the face. Here, beard is considered a part of the face but head hairs, neck, and shoulders are not. You do not need to do the third step.

Q6. Describe what you did and generate results for the images in **data/faces**. Place the result files in the directory **results/faces/**.

3.5 Hints

- Perfect foreground segmentation is NOT required. In fact, it is an extremely hard problem. We are only looking for implementations that provide sensible segmentation results. However, part of your grade for this homework will reflect the relative performance of your algorithm.
- Computational time is not the primary consideration in this homework. However, it should be fast enough for you to carry out repeatable experiments so that you have time to tune the program. Try to start or tune your algorithm with smaller subset of images first.
- Avoid loops in Matlab, practice more with vectorization techniques.
- The deadline for this homework is right before the grade due date. Therefore, we will not grant any extension. Your segmentation algorithm might take several minutes to segment one image. You will need to start the homework early so that you have time to generate the results for 40 horse images.

4 Submission

- All source files and any ***.mat*** files your functions require. Comments are greatly appreciated.
- `results1.mat`, `results2.mat`, and `results3.mat`
- An image of labels and an image of peaks for each of the four `.png` images for a total of 8 resulting images. Call the images `sheep_labels.png`, `sheep_peaks.png`, `hat_peaks.png`, etc.
- A file ***writeup.xxx***, where ***xxx*** is the extension of the writeup file. The only acceptable file formats are *pdf*, *doc*, and *txt*. This file should contains the answers to all the questions and anything you want us to know about your algorithm.
- The results of your image segmentation algorithm on the set of provided images. The results should be placed in a directory called ***results/horses/*** and ***results/faces/***. Furthermore, make sure that this directory ONLY contain the results that we asked; no extra junk please. If you think your algorithm is impressive and you would like to show us other results (e.g. for different types of images, using alternative parameters), please put them in a separate directory called ***results/extra/***. If you do include extra results, please document them in the write-up.
- DO NOT put any non-result (segmentation images) that we do not ask for in the ***results/horses/*** directory.
- DO NOT submit the images that we provide.
- DO NOT include your code in the write-up.

References

The theory behind mean shift clustering is described in the following paper:

`‘‘Mean shift: A robust approach toward feature space analysis’’ by D. Comaniciu and P. Meer, IEEE Trans. Pattern Analysis and Machine Intelligence 24, 2002, 603-619.`

Note that there are some errors in the equations and you should look at the `meanshift_errata.pdf` document. You will not need to understand all of the mathematical details of this paper as specifics of the algorithm that you will be implementing are outlined below. Of interest is section 4 of the paper, which discusses how a suitable feature space can be defined for image segmentation purposes.