Mean Shift Segmentation

CSE 573 : Computer Vision and Image Processing

Adarsh Prakash
UBIT Name: adarshpr
Person #: 5020 8760

December 16, 2016

1 Introduction

Parametric segmentation techniques such as Optimal Thresholding are faced with mainly two problems - (1) assumption that gray levels follow Gaussian distribution and (2) the need to compute a number of parameters of a pre-determined probability density function. Mean Shift Segmentation is an elegant technique which overcomes these problems by avoiding estimation of probability density function altogether. It is a non-parametric technique which can analyze complex multi-modal feature space and works by iterative shifts of each data point towards it's local mean.

This document briefly describes an implementation of Mean Shift Segmentation algorithm by employing techniques proposed by various resources. Density gradient estimation or estimation of mean shift vector is a adopted from [3], [1] and [6]. In specific, estimation using Epanechnikov and Gaussian Kernels has been adopted. A simpler version of limiting the search window in spatial domain has been adopted from [4]. Apart from these, basic image processing techniques such as region merging, edge detection and utilization of image properties ([6]) have been employed in this implementation.

2 Implementation

2.1 Algorithm

A multi-dimensional vector of image points is computed for each of each pixel by combining both the spatial and intensity information of that pixel. This is done prior to performing any further processing.

DiscontinuityPreservingFilter:
Set **Regions** = empty
for each unprocessed image point vector:
pixel_window = **GetPixelWindow**()
SeekCentroid(pixel_window)
Add pixel_window to **regions**
return **Regions**

MergeRegions(Regions)
Set **FinalRegions** = empty
Set **IntermediateRegions** = empty
Get spatial and range centroid for each region
for each unprocessed region from **Regions**:
region_window = **GetRegionWindow**()
Compute new range centroid for all image points in region_window
Merge all image points in window to new_region with newly computed centroid
Add new_region to **IntermediateRegions**
. for each unprocessed region in **IntermediateRegions**:
if region is smaller than **p_threshold**, merge it to last region on **FinalRegions**
else, add region to **FinalRegions**
return **FinalRegions**

SeekCentroid(pixel_window):
Until convergence or **Max_iterations**:
Perform Mean Shift using **Epanechnikov** or **Gaussian** Kernel

2.2 Description

For performing the discontinuity preserving filter as described previously, both Epanechnikov and Gaussian kernels have been implemented. Choice of kernel is deferred to runtime, supplied by the user, and default choice is Gaussian.

$$Gaussian\,Kernel, K_G(\mathbf{x}) = c\ exp(-\frac{1}{2}||\mathbf{x}||^2)$$

where value of c used is, $c = 1$

$$Epanechnikov\,Kernel, K_E(\mathbf{x}) = c\ (1 - ||\mathbf{x}||^2),\ if||x|| \leq 1 \qquad and\ 0,\ otherwise$$

where value of c used is, $c = 0.75$

Before computing the centroid for the window as such, to limit the number of computations, a procedure, **GetPixelWindow**, has been used. Instead of looking through all the image points in the vector space, this procedure projects the pixel coordinates onto a numbered grid and finds image points within the range threshold h_r by searching only in a bounded box of width and height equal to the spatial threshold h_s. This technique is adopted from [4] where much complex strategy is used to further leverage the search window. An illustration of limiting the search window for an arbitrary numbered pixel has been shown below:

```
[ 0  1  2  3  4  5  6  7  8]
[ 9 10 11 12 13 14 15 16 17]
[18 19 20 21 22 23 24 25 26]
[27 28 29 30 31 32 33 34 35]
[36 37 38 39 40 41 42 43 44]
[45 46 47 48 49 50 51 52 53]
[54 55 56 57 58 59 60 61 62]
[63 64 65 66 67 68 69 70 71]
[72 73 74 75 76 77 78 79 80]
```

Figure 1: Limiting the search window to a 5x5 bounding box where $h_s = 2$

The same technique is applied again to find neighboring regions in the procedure call **GetRegionWindow**. This technique alone brought down the computation time from 20 minutes to less than 30 seconds on the test machine for an image of dimensions 512x512.

After performing the filtering operation, a merging is performed to reduce the number of regions. Any two regions within the thresholds h_s and h_r are merged together with the help of spatial and range centroids (convergence points) from the filtering step. After the regions are merged, new centroids are computed. Before returning the so computed regions, any regions smaller than a certain threshold are merged to neighboring regions. This size is set to 5 by default and the user can supply a different threshold by supplying the `min_region_size` parameter.

Regions so obtained from the previous steps have both spatial and intensity information of image pixels that constitute them. From this information, an image is constructed to obtain the segmented image. A canny edge detection is performed on the segmented image to extract the boundary between regions. An overlay of these boundaries on segmented image gives a much better of idea of the results. The resulting image without boundaries and one with boundaries are both returned back to the user at the end of the operation.

2.3 Software Used

Language: Python v2.7
Modules/Libraries: numpy, cv2, matplotlib
IDE: Jupyter Notebook

3 Results

3.1 Evaluation on Test Images

The program was run on three test images provided. After a certain number of test runs the following challenges were encountered:

1. Longer running times (over 20 minutes)
2. Horizontal distortion
3. Oversegmentation

The first challenge was overcome by the technique as described section 2.2. With that solution, running time was significantly brought down to 15 seconds. The second challenge, after iterations of trials and tests, was discovered to be due to ordered scanning of regions from the list of regions. This introduced horizontal distortions in the image. The solution was to process the regions randomly (in a uniform fashion). This solution was adopted from the random sampling of search window, a technique used by Comaniciu et al in [4]. This has been illustrated as below:

(a) Horizontal distortion

(b) Result after random sampling of search windows

The third challenge however could not be solved. The cause of the problem, however, was discovered to be inefficiency of region merging procedure used. Several attempts were made to fix this problem, one such was computing transitive closure. This involves computing a RxR matrix where R is the number of regions. Each cell R_{ij} represents either 1 or 0. Here, 1 means the spatial distance between centroid of region i and j fall within h_s, the spatial threshold. A transitive closure using Warshall's algorithm on such matrix and bitwise operation on range distance will be more effective in grouping regions that fall below both the thresholds. However, this attempt was not successful for images above 128x128 because of limited memory (performing difference, square, sum, square root, transitive closure followed by bitwise AND operation on two 20,000 x 20,000 matrices!). But, it is possible to improve upon this attempt and solve this problem.

3.2 Test Image Results

(a) Original Image

(b) Segmented Image: $h_s = 3$ and $h_r = 20$

(c) Original Image

(d) Segmented Image: $h_s = 6$ and $h_r = 18$

(e) Original Image

(f) Segmented Image: $h_s = 2$ and $h_r = 30$

4 Program and Usage

To use the program, you will need the file **mean_segment.py**. Please find a snippet of sample usage below:

```
import cv2
from mean_shift import MeanShift
import matplotlib.pyplot as plt

im = cv2.cvtColor(cv2.imread('hill_house.jpg'), cv2.COLOR_BGR2RGB)

ms_object = MeanShift(2,30)
result, result_with_borders = ms_object.segment(im)

#display or use the 'result' or 'result_with_borders' objects
```

Samples of this test code have been provided along with **mean_segment.py**

5 Conclusion

5.1 Summary

Mean Shift Segmentation is a simple yet effective technique image segmentation with a wide array of applications in extracting low level image features [5] including real world applications such as object tracking from sequence of images and so on. This project involves applying a combination of different image processing techniques to find the segmented results which are the necessary pieces of input for image understanding and other steps of Computer Vision.

5.2 Lessons Learnt in this course

- Any complex image transformations are only a set of low level image processing techniques applied effectively. (So, I'll never underestimate the basics again!)
- I have often found programming homeworks in this course to clear up many doubts about the theoretical understanding of concepts. (So, hands on is good! Thanks to Prof. Chang and TAs)

References

[1] Lecture Slides, Chapter 6, CSE 573, Prof. Chang Wen Chen.

[2] Project Notes, CSE 574, Radhakrishna Dasari and Shuang Ma.

[3] Y. Cheng, "Mean Shift, Mode Seeking, and Clustering", IEEE Tans. Pattern Analysis and Machine Intelligence, Vol. 17, No. 8, pp. 790-799, August 1995.

[4] D. Comaniciu and P. Meer, "Robust analysis of feature spaces: color image segmentation," Proc. IEEE Computer Vision and Pattern Recognition, pp. 750-755, June 1997.

[5] D. Comaniciu and P. Meer, "Mean Shift Analysis and Applications".

[6] M. Sonka, V. Hlavac and R. Boyle, "Image Processing, Analysis, and Machine Vision", Third Edition, 2008.