CS676A Introduction to Computer Vision

Assignment 1

Mean Shift Image Segmentation

Saurav Kumar, 12641
Shreyash Pandey, 12683

Image segmentation

The problem statement for our assignment is to segment a color image into homogeneous tiles based on “similarity” in pixel values. For this purpose, we employ the mean-shift technique which treats pixels (in image feature space) as points generated from an underlying distribution, and estimates the location of modes leading to clustering of all pixels having the same mode (called the basin of attraction of that mode). The final step involves pruning of multiple local stationary points (modes) which ensures that only local maxima is preserved and the corresponding clusters are merged. As such, the task at hand includes estimating the distribution and defining “locality” for merging purposes. For the former, we employ a kernel based estimation technique (Parzen window technique) using Gaussian and Flat kernels to formulate our results. For the latter, we vary the bandwidth of the kernel, with modes lying within the bandwidth window undergoing pruning.

As an additional (optional) step of Mean Shift Image Segmentation algorithm, we define a parameter M which ensures that spatial regions having less M pixels are eliminated. This prevents the creation of unreasonably small segments.

It is to be noted that our approach involves applying mean shift technique to the **joint domain representation** of image feature space. That is, pixel vectors include both spatial (x,y) and chromaticity (RGB / LUV) coordinates, leading to a 5-dimensional feature space.

Implementation

We started off by preparing the code in Python, our objective being of not using any library to compute mean shift, to simplify the process. But steps such as pdf estimation and mean shift filtering involved iterating over all the image pixels, stored in a 2D list, leading to $O(w^2h^2)$ time complexity. Naïve implementation in a slow language like Python having default data structure as lists rendered our implementation very slow, which is undesirable as running time went close to an hour for a 481 x 321 sample image.

The alternatives to this approach include using numpy library to do matrix operations on the image and optimize the computation wherever possible. Another alternative is to use faster language such as C++ with painstaking effort. But for our assignment, after our naïve implementation, we decided to use a library that in fact uses both numpy and C++ (for OpenCV) to vastly speed up the process. We have also turned in our own naïve implementation in Python of the paper. But for all the experiments and comparisons, we have used the cited library[1]. However, for swift experimentation, we had to tweak the function signature of the library to include the type of kernel as a parameter specified by the user at run time[2].

Experiments

Following are the parameters that we vary and the results we expect from such variations:

1. Effect of spatial bandwidth (sr) : Only features with large spatial support are present in the filtered image when sr increases.
2. Effect of range domain bandwidth (rr): Only features with large color contrast survive when rr increases.
3. Effect of M : Spatial regions containing less than M pixels are eliminated, leading to lesser number of segments (num_seg) as M increases.
4. Effect of kernel type: Smooth Trajectory property of Gaussian kernel ensures that it almost always outperforms Flat kernel in terms of quality of segmentation. It is however slower to converge than the Flat kernel.

Pseudo Code

```
L = {} # set of basin of attraction with modes as keys

C = {} # set of colors of modes


# Find basins of attraction

for i in range(0, height):

    for j in range(0, width):

        x = get_vector(i, j)

        for epoch in range(0, MAX_EPOCH):

            x = next_mean(x)

        if new mode is found:

            Add this mode as key to L

            Add (i, j) to its value array

            Add color information of this mode to C

        else:

            Add (i, j) to value array of the key


# Merge basin of attractions

for each basin in basinList:

    for every other basin in basinList:

        if canMerge(basin, other_basin):

            merge(basin, other_basin)


def canMerge(b1, b2):

    return ((spcDist(b1, b2) < spcBw

          or rngDist(b1, b2) < rngBw))
```

```
def merge(b1, b2):

    for each pixel in b2:

        append pixel in value array of b1

    b1.key = key(argmax(pdf(b1), pdf(b2)))


def next_mean(ci, cj):

    x   = get_vector(ci, cj)

    for i in range(max(0, ci-bw), min(ht, ci+bw)):

        for j in range(max(0, cj-bw), min(wd, cj+bw)):

            x_i     = get_vector(i, j)

            del_x   = (x - x_i) / bw

            g_val   = g_flat(sq_norm(del_x))

            num    += x_i * g_val

            den    += g_val

    return num/den


def pdf (x):

    for i in range(0, height):

        for j in range(0, width):

            x_i     = get_vector(i, j)

            del_x   = (x - x_i) / bw

            est    += epan_kernel(sq_norm(del_x))

    return est
```

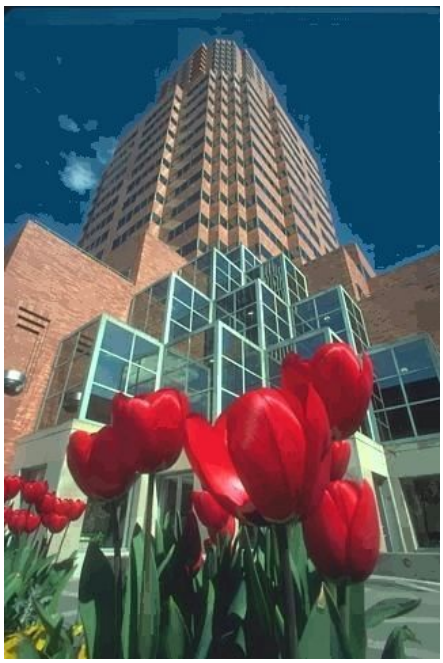
```
import pymeanshift as ms

original_image = cv2.read(imagePath)
(segmented_image, labels_image, number_regions) = ms.segment(original_image, spatial_radius=sr,
                                                             range_radius=rr, min_density=m,
                                                             skernel=sk, rkernel=rk)

cv2.imwrite(newImagePath, segmented_image)
print “Number of segments: ”, number_regions
```

Results

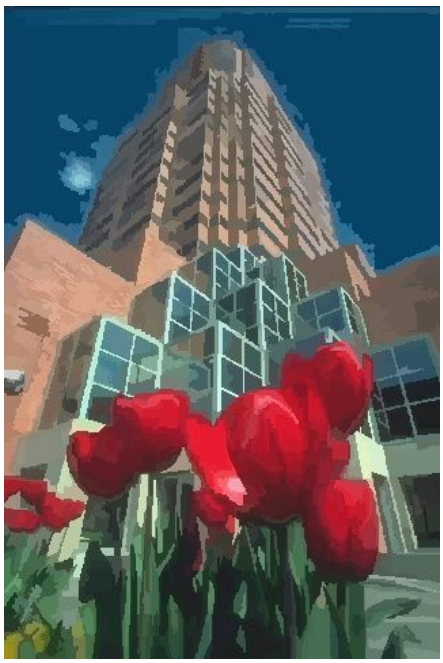
For the first image, '86000.jpg', we can see the effect of spatial bandwidth by comparing images numbered 2 and 12. Segments of the image 2 with small spatial support such as the sky (close to the building) and windows get lost in 12 as sr is increased from 2 to 18. Effect of range domain bandwidth is clear in comparing images numbered 2 and 3. Segments of image 2 with low contrast such as the sky disappear in 3 as rr is increased from 2 to 6. Effect of parameter M is clear in comparing images numbered 8 and 9. As can be seen, there is a significant decrease in num_seg for both the cases. This leads to large blurring effect in 9 as compared to 8. Lastly, Effect of kernel type was visible while running the code - flat kernel converged faster than gaussian kernel. But as can be seen from the images shown, gaussian kernel gives a superior performance when it comes to quality of segmentation.

Legend : (sr, rr, M sk, rk, num_seg)

Original Image: 86000.jpg



1. (2 2 1 G G 53708)



2. (2 2 46 G G 1089)

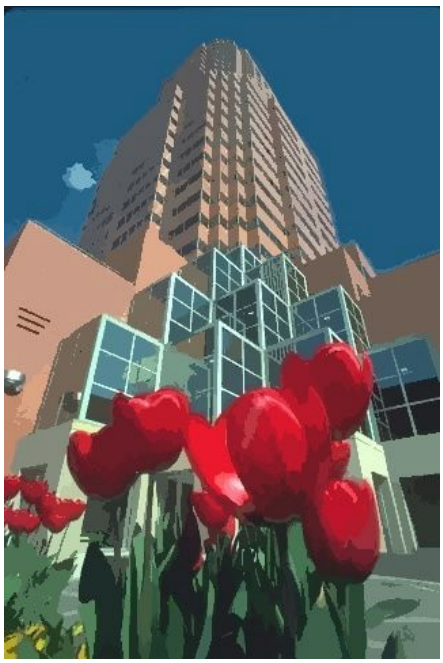
3. (2 6 46 G G 746)

4. (6 6 46 G G 623)

5. (6 10 46 G G 314)

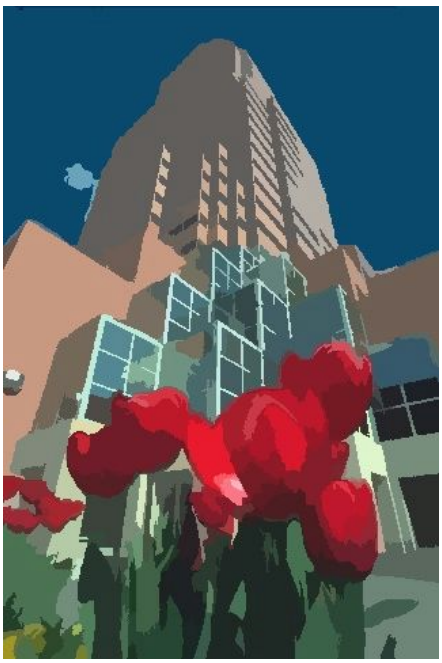
6. (6 14 46 G G 173)

7. (6 18 46 G G 86)



8. (10 6 1 G G 9121)

9. (10 6 46 G G 502)



10. (10 10 46 G G 265)

11. (14 14 46 G G 109)

12. (18 2 46 G G 989)

13. (18 10 46 G G 186)

14. (10 1 10 U U 1703)

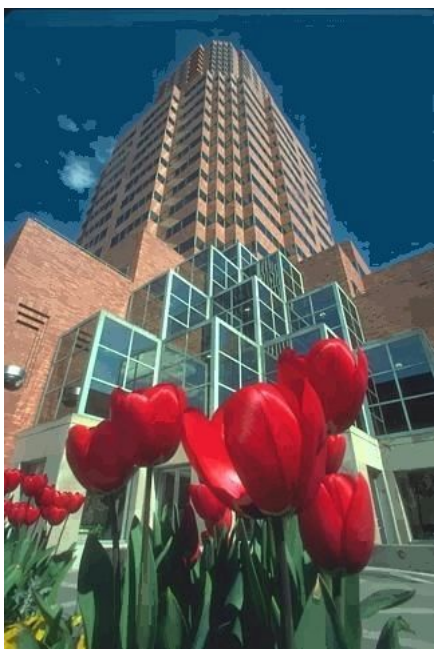
15. (10 1 40 U U 466)

16. (20 1 10 G U 1693)

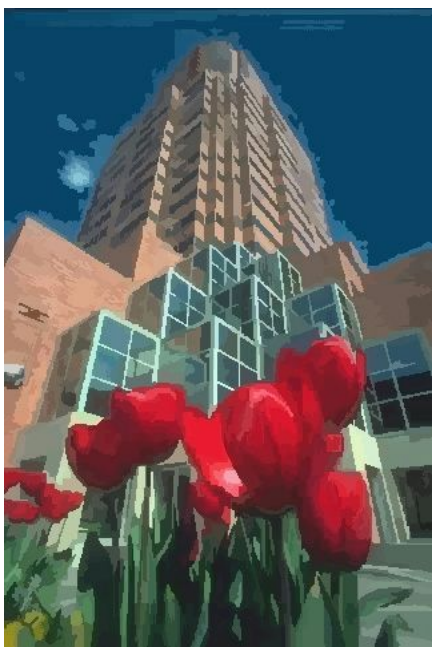
17. (10 1 10 G U 1703)

18. (10 1 40 G U 466)

19. (30 1 10 G U 1682)



20. (2 2 1 U G 53708)

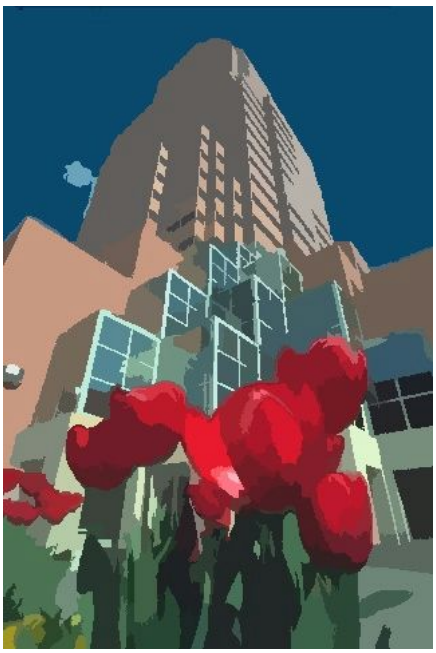


21. (2 2 41 U G 1215)

22. (10 2 1 U G 51851)

23. (10 2 11 U G 4034)

24. (10 2 41 U G 1191)



25. (10 10 41 U G 281)

26. (18 2 1 U G 50643)

27. (18 10 41 U G 193)

For the image '42049.jpg', the results are as shown below. Again, the effect of increase of rr (range domain bandwidth) is visible in the transition from image 3 to image 4 - only features with large color contrast survive in the latter. Also, as M is increased, the number of segments decreases as is visible from image 1 and image 2.

Original Image: 42049.jpg

1. (10 1 40 G G 441)

2. (10 1 10 G G 1834)

3. (10 5 10 G G 511)

4. (10 10 10 G G 94)

5. (10 1 10 U U 592)

6. (10 5 10 U G 511)

Similar results follow for image '101085.jpg'. A smaller value of rr leads to better segmentation results as the image itself is composed of largely similar color intensity levels. For rr=10 (image 4), the statues almost constitute a single segment because of lack of color contrast.

Original Image : 101085.jpg

1. (10 1 40 G G 1578)

2. (10 1 10 G G 6537)

3. (10 5 10 G G 1868)

4. (10 10 10 G G 296)

5. (10 1 10 U U 2672)

6. (10 5 10 U G 1868)

For the image ‘119082.jpg’, there is a significant variation in color intensity values which results in a higher value of rr leading to better segmentation, as can be seen from image 2 and image 3. Increasing M leads to lesser number of segments without compromising the segmentation quality (visible from 1 and 2).

Original Image: [119082.jpg]

1. (10 1 40 G G 1365)

2. (10 1 10 G G 5318)

3. (10 5 10 G G 2123)

4. (10 10 10 G G 646)

	
5. (10 1 10 U U 2309)	6. (10 5 10 U G 2109)

Among all the images assigned to us, image '170057.jpg' with the soldiers in camouflage outfits has the most similar color intensity values. As such, a smaller value of rr (=1) completely outperforms larger values such rr=5 or 10 (visible from images 2,3 and 4). Also, increasing M leads to lesser number of segments without compromising segmentation quality. Also, Gaussian kernel leads to better results.

Original Image : 170057.jpg

	
1. (10 1 40 G G 1412)	2. (10 1 10 G G 5469)

3. (10 5 10 G G 599)

4. (10 10 10 G G 103)

5. (10 1 10 U U 494)

6. (10 5 10 U G 596)

Additionally, following are some results from our own implementation of image segmentation using mean shift algorithm in python. As the runtime for large images was close to an hour, we tested our implementation on resized/cropped parts of images assigned. For a 41*61 image, low values of hr and rr lead to 'sharper' segmentation, as expected. Also, as can be seen from the resized eagle image, gaussian kernel outperforms flat kernel in terms of segmentation quality.

Original | Result

Original | Gaussian | Flat

References

1. Mean Shift Library in Python github.com/fjean/pymeanshift
2. Fork of [1] to specify Kernels as API parameter github.com/2020saurav/pymeanshift
3. Python OpenCV
4. Comaniciu, Dorin; Peter Meer (May 2002). "Mean Shift: A Robust Approach Toward Feature Space Analysis". IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE) 24 (5): 603--619