First Assignment (Segmentation)

TUTORS: MIRELA POPA

DEADLINE: 17/05/2021 AT 11.59 PM

DESCRIPTION

In this assignment, you will develop your own image segmentation application, using the Mean-shift algorithm.

The Mean-shift Algorithm

The Mean-shift algorithm clusters an *n*-dimensional data set by associating each point to a peak of the data set's probability density. 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. The algorithm then shifts the window to the mean and repeats until convergence, i.e. the shift is under some threshold *t* (for example, $t = 0.01$). With each iteration the window will shift to a more densely populated portion of the data set until a peak is reached, where the data is equally distributed in the window. Implement the peak searching processes as the function:

(Matlab) function peak = findpeak(data; idx; r)

(Python) def findpeak (data, idx, r):

where *data* is the *n*-dimensional dataset consisting of *p* points, *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 syntax:

(Matlab) function [labels; peaks] = meanshift(data;r)

(Python) def meanshift(data, r):

…. return labels, peaks

where labels is a vector containing the label for each data point and peaks is a matrix storing the density peaks found using *meanshift* as its columns. Note the *meanshift* algorithm requires that peaks are compared after each call to the *findpeak function* and for similar peaks to be merged. For our implementation of *meanshift*, we will consider two peaks to be the same if the distance between them is smaller than ***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.

Debug your algorithm using the data set (*pts.mat* which stores a set of 3D points belonging to two 3D Gaussians) provided with the assignment zip folder. With r = 2 (this should give two clusters). Plot your result using the plot3dclusters function, also available in the zip archive.

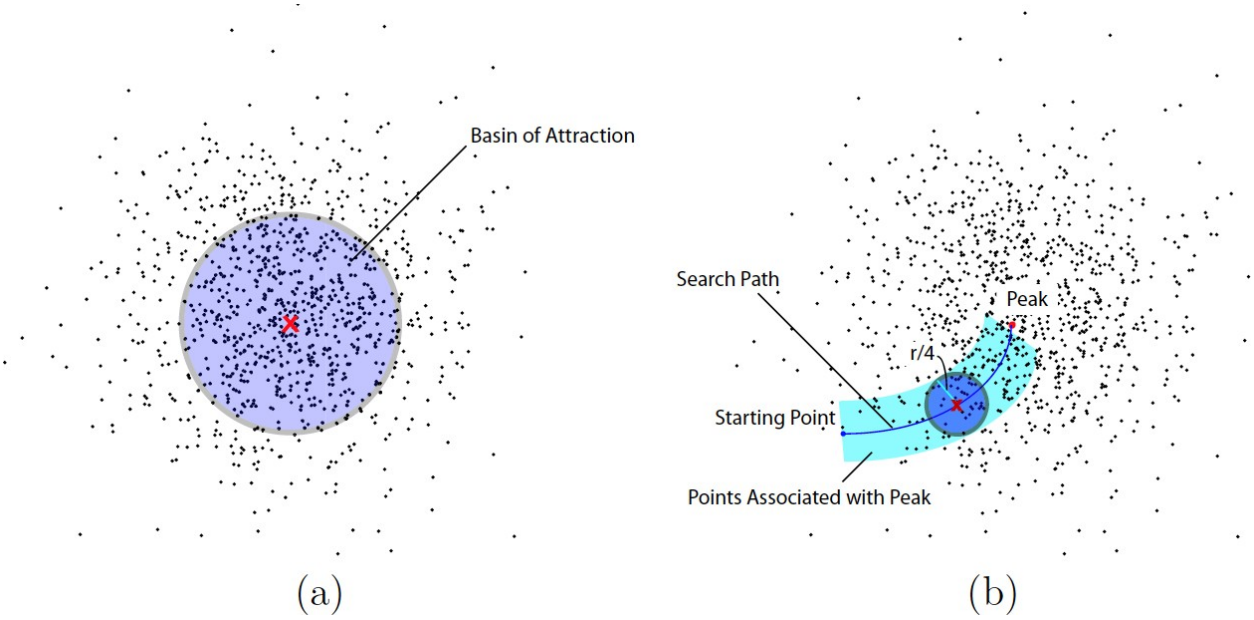

Figure 1. Speedups incorporated into the mean-shift algorithm: (a) basin of attraction, (b) points along

As described above, the *meanshift* algorithm is too slow. Therefore, we will incorporate two speed-ups: Upon finding a peak, the **first speedup** will be to associate each data point that is at a $distance \leq r$ from the peak with the cluster defined 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. This is shown in Figure 1(a). The **second speedup** is based on a similar principle, where points that are within a distance of *r/c* of the search path are associated with the converged peak, where *c* is some constant value (check Figure 1(b)). We will choose *c* = 4 for this problem but you are also asked to check other values. Incorporate the above speedups into your mean-shift implementation by modifying your implementation from the previous part. The resulting modified function should have the following syntax:

(Matlab) function [labels; peaks] = meanshift_opt(data;r)

(Python) def meanshift_opt (data, r, c):

To realize the second speedup, you will need to modify *findpeak* as follows:

(Matlab) function [peak; cpts] = findpeak_opt(data; idx;r)

(Python) find_peak_opt(data, idx, r, threshold, c=4):

where the output ***cpts*** is a vector storing a *1* for each point that is a distance of *r/*4 from the path and *0* otherwise.

For more information about the algorithm, you could check the paper by Dorin Comaniciu and Peter Meer (https://courses.csail.mit.edu/6.869/handouts/PAMIMeanshift.pdf) "Mean shift: A robust approach toward feature space analysis" published in PAMI, 2002.

Implementation Hints:

Pre-processing: vectorize image features prior to applying the Mean-shift algorithm:

- E.g., create *3-by-p* matrix in case you use color features, where *p* is the number of pixels in the input image.
- If we want to include spatial position information as well, you can define the feature vector as a *5D* vector specifying the color channels and *x, y* coordinates of each pixel.
- Use Matlab/Python functions to convert between color spaces
- **Important**: in order to optimize your code, try to **avoid using loops** in your functions, whenever possible. Instead, use matrix manipulation. For example, if you want to select all points that have been labeled "1," instead of writing a for loop you could write:
  (Matlab) found = find(labels == 1);
  curr_data = data(:, found);
  (Python) found = np.argwhere(labels) or labels[labels>0]

  You can use the same expression, to find points with a smaller or bigger threshold.
  Another Hint: use a function to find distances between a data point and all other data:
  (Matlab) *distances = pdist2(data, data_point, 'euclidean');*
  (Python) distances = cdist(data_point, data, metric='euclidian')

  For the actual segmentation, implement the function:
  (Matlab) function segmIm = imSegment (im, r)
  (Python) def segmIm(im, r):
  where *im* is the input color RGB image and *r* the radius. Since *meanshift* algorithm uses Euclidean distance, unfortunately, Euclidean distance in RGB color space does not correlate well to perceived difference in color by people. Work using the CIELAB color space in your code, where Euclidean distances in this space correlate better with color changes perceived by the human eye. In **imSegment** cluster the image data in CIELAB color space by first converting the RGB color vectors to CIELAB using in Matlab the function rgb2lab and in Python color.rgb2lab(im) or cv2.cvtColor(image, cv2.COLOR_RGB2LAB). Then convert the

resulting cluster centers back to RGB using in Matlab the function lab2rgb and in Python color.lab2rgb or cv2.cvtColor(image, cv2.COLOR_LAB2RGB).

Experiments

During the experiments and evaluation, use small pictures, for example images containing frontal face images. Make use, apart from color features, of also (x,y) coordinates (thus, 5D features).
In addition, in your report, provide details about the effect of speeding approaches. Test your algorithm for various values of *r* and *c* and show the results, while also report results in time gain. For each value of r, run your algorithm using **(1)** just the **CIELAB color** values (i.e., a 3D feature vector), and **(2) CIELAB+position** values (i.e., a 5D feature vector). Report on the results.

Preprocessing

Can you suggest other processing steps in order to improve the segmentation results? Explain your steps and the reasons behind them.
In your report, use the following images from the Berkeley Segmentation Dataset:

https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/dataset/images/color/181091.html
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/dataset/images/color/55075.html
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/dataset/images/color/368078.html

And do not forget to attach your results in the report.
Note: although more efficient implementations exist, the above simplified implementation of mean-shift may take some time to run for the provided input images, but not more than a few minutes depending on your PC specifications.

STEPS

1. Read the image and apply the preprocessing steps suggested above.
2. Implement the Mean-shift algorithm, considering the suggested optimizations.
3. Apply the algorithm on the image features, transform the result back to an image and visualize the obtained segmentation.
4. Test different parameters, such as r, c, feature types.

DELIVERABLES:

1. A report of 800-1500 words with plots and figures. Do not forget to explain and display also intermediary steps. Provide the aforementioned information in the implementation steps. For example, describe your observations for the different parameters involved.
2. Well documented code in Python or Matlab, with a function in Matlab:

function [segIm, labels, peaks] segmIm = imSegment(im, r, feature_type) or a script in Python imageSegmentation.py –image -r -feature_type and the related functions/classes. In case your structure is more complex, please also provide a short readme file. Your code will be tested with images having different characteristics.

3. Upload everything on Canvas as a zip file, using the following format: *Name_Surname_First_Assignment_CV2021.zip.*

GRADING (2.5P)

A) CODE – 1.5 POINTS (the code should be running, should display the segmented image and should be documented)
B) REPORT– 1 POINT (should provide the essential steps and a discussion of the experiments, using different parameters, images)