Computer Vision Lab Assignment 4 Report

lijiaj@student.ethz.ch

November 2023

In this assignment, I'm required to implement the mean-shift algorithm for the image segmentation task. Specifically, I need to fill in four placeholder functions in the given source file and further experiment the algorithm with different `bandwidth` values. Therefore, I'll first walk through the four functions and then present the results and findings on the `bandwidth` parameter.

1 The distance Function

The `distance` function is used to compute the (Euclidean) distances between a given point `x` and all points (including itself) `X`. In the code, I first used the `diff = X - x` to get the vector differences between `x` and all points in `X` and then `sq_diff = np.sum(diff ** 2, aix=1)` gave us the squared (Euclidean) distances. The desired distances were obtained by further taking square root with the code `dist = np.sqrt(sq_diff)`.

2 The gaussian Function

The `gaussian` function is to calculate the weights of all points based on kernel function with `bandwidth` and the distances returned by the above `distance` function. Specifically, `weights = (1 / (bandwidth * np.sqrt(2*np.pi))) * np.exp(- (dist ** 2) / (2 * bandwidth ** 2))` computed the weights properly according to the kernel function.

3 The update_point Function

The `update_point` function is designed to update the position of a point using the weights computed by the above `gaussian` function. In terms of the implementation details, `weighted_sum = np.sum(weight.reshape(-1,1) * X, axis=0)` calculated the weighted sum of all points and the total sum of weights was obtained with the help of `total_weight = np.sum(weight)`. Then, `new_position = weighted_sum / total_weight` gave the weighted mean, namely the new position of the point.

4 The meanshift_step Function

The `meanshift_step` function performs one iteration of the meanshift algorithm procedure on all points. In particular, it looped over all points in the `X` and in the loop it called `distance` function, `gaussian` function, and `update_point` function in turn to update the position for each point.

5 Experiments and Results

There is one hyperparameter that we can tune in this meanshift algorithm implementation: `bandwidth`. In our meanshift algorithm, the `bandwidth` parameter determines the range of neighboring points considered when updating points. In simple words, the larger the `bandwidth` value, the wider the range considered. Using a smaller `bandwidth` value means that each point is updated based only on nearby points, which can result in numerous centroids/clusters. Conversely, a larger `bandwidth` value causes each point to consider a wider range of points when updating, thus reducing the final number of centroids/clusters. These theories match the segmentation results/observations in Figure 1.

(a) `bandwidth` of 2.5

(b) `bandwidth` of 3

(c) `bandwidth` of 5

(d) `bandwidth` of 7

Figure 1: Segmentation results with different `bandwidth` values

Among recommended `bandwidth` values (1,3,5,7) to experiment with, our code could not run with the `bandwidth` of 1 because there were only 24 colors defined in the `colors.npz` file while the `bandwidth` of 1 produced more than this number of clusters, making the assignment of different colors to different clusters impossible. On the other hand, the code could run properly with `bandwidth` set to the default value (2.5) and other values in the recommended list, e.g. 3, 5, and 7, as shown in Figure 1. I personally think there is no absolutely "best" bandwidth value and we should choose the bandwidth value based on our specific need, i.e. fine-grained segmentation, or coarse-grained segmentation.

There are several possible solutions to the problem mentioned above: one is to increase the number of available colors; the other is to make the 25th and following clusters share existing colors, which can be achieved with `labels[labels >= len(colors)] = len(colors) - 1`, although this may reduce the quality of the segmentation.