Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision
Autumn 2023

Image Segmentation

Homework-4

November 23, 2023

Alpay Ozkan

aoezkan@student.ethz.ch

By submitting this work, I verify that it is my own. That is both implementation and report are my own work.

Contents

1 Mean Shift Algorithm

1.1 Implementation

For a given point, I calculate L2 (euclidian) distance from each point in the dataset. Then this distance is passed to a gaussian kernel/function which computes the normalized weights of the points.

$$\alpha_n = \frac{K^{'}(\|\frac{x-x_n}{\sigma}\|^2)}{\sum_{n'=1}^{N} K^{'}(\|\frac{x-x_{n'}}{\sigma}\|)^2} \tag{1}$$

By construction weights are normalized.

$$\sum_{n=1}^{N} \alpha_n = 1 \tag{2}$$

where

$$K^{'}(\|\frac{x-x_n}{\sigma}\|^2) = \exp(-0.5 * \frac{\|x-x_n\|^2}{\sigma^2}) \tag{3}$$

For numerical stability and to avoid potential NaN values, I shifted the numerator and denominator by the maximum element in the distance.

Update point just takes the linear combination of the points with the corresponding calculated weights.

$$x_{center} = \sum_{n=1}^{N} \alpha_n * x_n \tag{4}$$

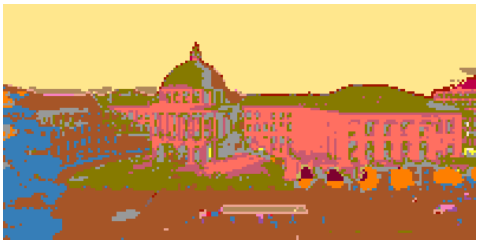
In meanshift-step, I calculate new point centers for each point in the feature-space. The whole procedure is given as follows:

```
def meanshift_step(X, bandwidth=2.5):
    clist = [] # point center list
    for x in X:
        dist = distance(x,X) # calculate L2 distance btw point x (1,3) and all points X (N,3), giving
        w = gaussian(dist, bandwidth=bandwidth) # calculates weights for each point for the given cen
        w = w.reshape(-1,1) # for broadcasting
        c = update_point(w, X) # linear combination of weights and points, (1,3)
        clist.append(c) # store all centers
    X = np.array(clist) # np array of all point centers
    return X
```

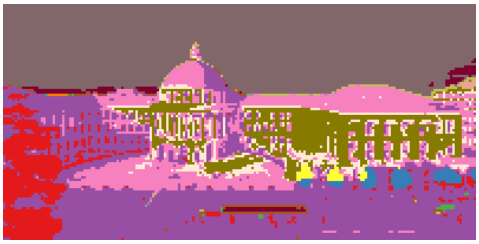
In meanshift algorithm we repeat meanshift-step several times, so that point centers converge to a center.

```
def meanshift(X):
    for _ in range(20):
        X = meanshift_step(X)
    return X
```

1.2 Results



((a)) bandwidth=2.5



((b)) bandwidth=3

((c)) bandwidth=5

((d)) bandwidth=7

Figure 1: MeanShift Segmentation Results with different bandwidths

I have tried my implementation with several bandwidths. Bandwidths smaller than 2.5 in my case 0.1 and 1 did not work and threw error. The reason is that the number of colors specified is limited to 24 and with smaller bandwidths I get more cluster centers. As also can be observed from Figure 1 as bandwidths increases we have less points.

When the bandwidth increases the gaussian gets smoother ie. more close to uniform as a result most of the points receive the same signal, ie. ignoring the locality, but when the bandwidth is smaller then it is more like a delta function which is perfectly local yielding point specific centers resulting more number of point centers.

I think the algorithm mostly is able to separate background and foreground the ETH building assigned to different classes. For smaller bandwidths, it is able to capture the roof, the ground, and the windows mostly to different classes. The rectangle positioned in the bottom is probably due to significant contrast difference as it is brighter that corresponding the metal tram station roof assigned to a different cluster center. For bandwidth 9 it is only able to distinguish between sky and the foreground.

How to solve the un-runnable cases? One solution would be to add more colors which will correspond to more point centers, but this won't be very tractable since we might have thousands of centers. Alternative solution would be to reapply the algorithm recursively, so that point centers are regrouped depending on their closeness. Easiest solution would be to find a suitable or larger bandwidth which reduces the amount of point centers as in our case for bandwidths > 2.5 worked.

2 SegNet

No report needed.