Lab Assignment Report - Mean-Shift

Yaqi Qin: 21-946-819

November 18, 2022

1 Algorithm Implementation

Implement the distance Function. In the distance function, given a point vector x, the distance from it to point i in the matrix X is just $\|X_i - x\|_2$. So the l^2 distance to all other points can be calculated using `torch.norm()` on matrix $X - x$ along dimension 1 (the column).

Implement the Gaussian Function. To compute the weights of points, the Gaussian kernel is used on the distance, i.e. the weight of point i to the given point x with bandwidth l is:

$$w_i = \exp\left(-\frac{(X_i - x)^2}{2l^2}\right) = \exp\left(-\frac{dist_i^2}{2l^2}\right)$$

The element-wise computation is directly applied on the vector $dist \in R^n$ to get all the point weights.

Implement the update point Function. In the `update_point(weight, X)` function, the weight vector $w \in R^n$ is first divided by the sum, denoted as w', to ensure all the entries sum up to 1. Then the weighted average is computed as matrix multiplication, i.e. $w'^T X \in R^3$.

Then at each Mean-Shift step, loop over each point X_i to compute the distance vector from X_i to X, get the weights based on Gaussian kernel, and then use an auxiliary matrix $X\_$ to store the updated weighted average values to ensure the the update is out of place. After all the points are updated in one step, let $X = X\_$ to update X iteratively (20 iterations are set).

Accelerating the Naive Implementation. Instead of looping over each point and do the update, the implementation is optimized by updating $Bs = 25$ points at a time. For every Bs points $X[i : i+Bs] \in R^{Bs\times 3}$, compute the distance of all pairs $\in R^{Bs\times n}$ using the `torch.cdist(x, X, p=2)` function. After getting the weights by applying the Gaussian kernel, the weight matrix are normalized for each row: $W \in R^{Bs\times n}$. The weighted average values for all batch points are again computed by matrix multiplication $WX \in R^{Bs\times 3}$.

2 Result

Comparison of different bandwidths. As shown in Figure 1, as bandwidth decreases, the number of centroids increases. The reason might be that with small bandwidths, the correlation of nearby pixels are much higher than far away ones, so that local neighborhood has higher weight for computing the mean at each iteration, resulting in more varied detected centroids. On the contrary, when the bandwidth increases, the correlation of all pixels are smoothed, so that the variance of all centroids is diminished, resulting in less detected labels.

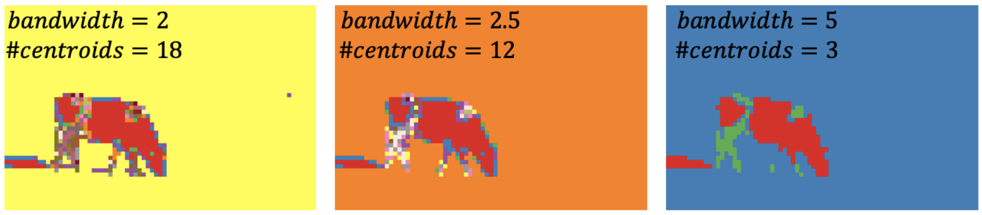

Figure 1: Segmentation results of different bandwidths.

Run time analysis. To ensure fair run time comparison, the algorithms are run on CPU of the same Mac, the naive implementation takes 36.02s, whereas the vectorized implementation takes 1.59s with $Bs = 25$, and 2.34s with $Bs = len(X)$. Clearly, with vectorized computation, the run time can be effectively reduced by roughly a factor of 20.