Exercise 5: Image Segmentation

Hardik Shah

1 Mean-Shift Algorithm

The task was to implement the mean-shift algorithm and validate it on an image. Mean-Shift is a popular non-parametric clustering algorithm commonly used for image segmentation. The algorithm aims to group pixels with similar color characteristics into segments, thereby highlighting different objects or regions in an image.

2 Implementation and Algorithm Outline

We first load an image and convert it to the CIELAB color space, which better represents color perception. The image is then flattened into a 2D array for processing `(h*w) x 3 = num_pixels x 3`. The Mean-Shift algorithm is applied iteratively for 20 steps, where each step involves updating each point's feature values(pixel intensities) based on the weighted average of its neighbors. The bandwidth parameter controls the influence of neighboring points in the update process. After segmentation, the resulting labels are mapped to color values, which gives us the segmented image.

Here is a step-by-step overview of the implementation of mean-shift:

1. **Initialization:**
   For each pixel in the image, initialize its location as the starting point.

2. **Iteration:** `meanshift_step(X, bandwidth=2.5)`
   For each point(pixel) in the image

   (a) **Compute Distances:** `distance(x, X)`
   Calculate the Euclidean distances from the current point to all other points (including itself). The distance is typically measured in the feature space (i.e. 3-channel pixel intensities in CIELAB space).

   (b) **Compute Weights:** `gaussian(dist, bandwidth)`
   Calculate the weight for each point using a Gaussian kernel function of distance. The Gaussian kernel function is defined as $K(x) = \exp(-\frac{x^2}{2h^2})$, where h is the bandwidth, and in this case it's set to 2.5. The weight of a point is determined by its distance to the current point. Closer points will have higher weights, and farther points will have lower weights.

   (c) **Compute Weighted Mean:** `update_point(weight, X)`
   Calculate the weighted mean of all points (including the current point) based

on the computed weights. Weighted_Mean $= \frac{\sum_i w_i.x_i}{\sum_i w_i}$ where w_i is the weight of point i and x_i are the pixel intensity values for point i.

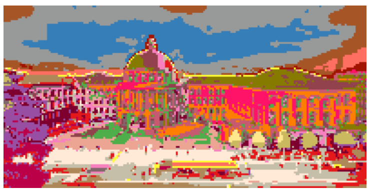
(d) **Update Current Point:**
Update the current point's position with the calculated weighted mean.

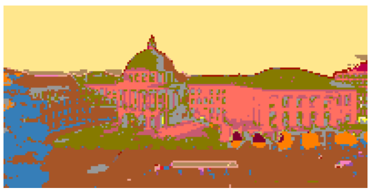
3. **Convergence:** `meanshift(X, bandwidth)`
Repeat the iteration steps until the algorithm converges, i.e., for 20 iterations here.

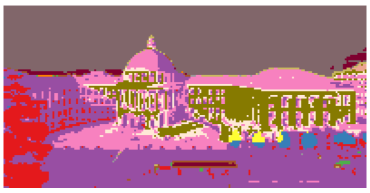
The mean shift algorithm effectively shifts points towards the mode (peak) of the underlying data distribution, leading to the formation of clusters. In the context of image segmentation, pixels that converge to the same mode are considered to belong to the same segment or cluster. The algorithm is capable of handling irregularly shaped clusters and doesn't require specifying the number of clusters beforehand.

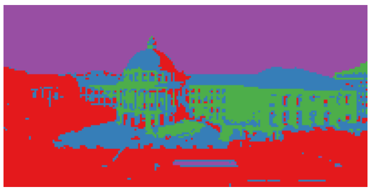
2.1 Results and Observations

We experiment with different bandwidth values $[1, 3, 5, 7]$. A small bandwidth may lead to a very narrow Gaussian kernel, making the algorithm sensitive to small variations in the data. This could result in slow convergence or premature termination. The current implementation fails for `bandwidth=1`, for the simple reason that the number of clusters at the end of mean-shift is greater than the number of colors we have (i.e. 24 which basically represents maximum possible clusters). An increasing bandwidth leads to faster convergence, and hence, the current implementation works for the bandwidths $[2.5, 3, 5, 7]$. For `bandwidth=1`, a possible fix would be to further cluster the points using a simple KMeans clustering, and choose the number of clusters K as the maximum possible clusters (here 24). We could reduce the number of clusters than 24 by visually inspecting the image and approximating the number of segments possible for the given image. Figure 1 below shows the results we obtain for different bandwiths.


(a)


(b)


(c)


(d)

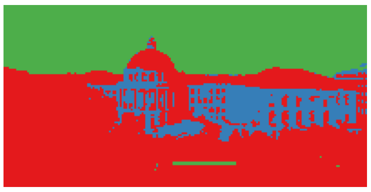

(e)

Figure 1: Comparing segmentation results for different values of the `bandwidth` parameter. **(a)** `bandwidth=1`, along with KMeans with `k=24` **(b)** `bandwidth=2.5` **(c)** `bandwidth=3` **(d)** `bandwidth=5` **(e)** `bandwidth=7`

***Note:** In the `meanshift_step(X, bandwidth=2.5)` function, the way we update each point in the image affects the results we get. Particularly, if we update each point in a copy of the input image array X_{copy} (which is initialized as X), and after updating each point in the image, we update X as $X = X_{\text{copy}}$, we converge more slowly and the code does not work for `bandwidth < 4`. However, updating each point in place in the input image array X, we get a successful segmentation for all bandwidth values in $[2.5, 3, 5, 7]$, only for `bandwidth=1`, the code fails to run. Although, intuitively it makes sense to update in a copy, otherwise we factor in updated points for all points $< i$, when we calculate distances for updating point i. However, since we get a successful segmentation for the given default value of `bandwidth=2.5` in the skeleton code, the implementation of updating in place has been used.