Segmentation lab report

Ambroise AIGUEPERSE

1 Mean Shift algorithm

1.1 distance function

The `distance` function is defined to accept two arguments: the reference point `x` and a matrix of points `X`. The reference point is a one-dimensional NumPy array representing the feature vector of a single data point, while `X` is a two-dimensional NumPy array containing the feature vectors of a set of data points.

This function calculates the Euclidean distance from the reference point to each point in the matrix. It does so by executing a vectorized computation (to not use loops and be faster), that consists of the following steps: it subtracts the reference point from each point in the matrix, squares the result, sums these squares across the feature dimensions, and then applies the square root to each sum.

The output is a one-dimensional NumPy array where each element corresponds to the distance between the reference point and a point in the input matrix.

1.2 gaussian function

The `gaussian` function is tasked with computing the weight of each point in a set based on the Gaussian kernel. It is structured to receive two parameters: a one-dimensional array `dist` containing the computed distances, and a scalar `bandwidth` representing the bandwidth parameter of the kernel.

The function applies the Gaussian kernel to the distance values, transforming them into weights. The operation involves the exponentiation of the negative half of the square of the ratio of `dist` to `bandwidth`, which is then divided by the product of `bandwidth` and the square root of 2π.

The result of this function is a one-dimensional NumPy array with each element representing the Gaussian weight corresponding to the distance value from the `dist` array. These weights will be used in the mean-shift step to adjust the influence of each point during the computation of the new point positions.

1.3 Update Point Function

The `update_point` function is designed to compute the new position of a point in the feature space as a weighted average of all points in a given neighborhood. The function accepts two arguments: a one-dimensional array `weight` containing the weights of each point and a two-dimensional array `X` representing the set of points.

Within the function, the new position is obtained by performing a weighted sum of the points in `X`, where the weights are provided by the `weight` array. This operation involves multiplying each point in `X` by its corresponding weight and summing the results. The sum is then normalized by the total sum of the weights to yield the average.

The return value of this function is a one-dimensional NumPy array representing the updated feature vector of the point.

1.4 Mean-Shift Step Function

The `meanshift_step` function conducts a single iteration of the Mean-Shift algorithm on the entire dataset. It takes as input a two-dimensional array `X`, where each row corresponds to a data point in the feature space, and a scalar `bandwidth` that defines the kernel's bandwidth.

For each data point in `X`, the function calculates the Euclidean distance to all other points, derives the Gaussian weights, and computes the updated position based on these weights. The function iterates over the points, utilizing the previously defined `distance`, `gaussian`, and `update_point` functions to perform these operations.

The output is a new two-dimensional NumPy array of the same shape as `X`, with each row representing the updated position of the corresponding point in the feature space. This step is repeated for a predefined number of iterations in the overall Mean-Shift algorithm.

1.5 results

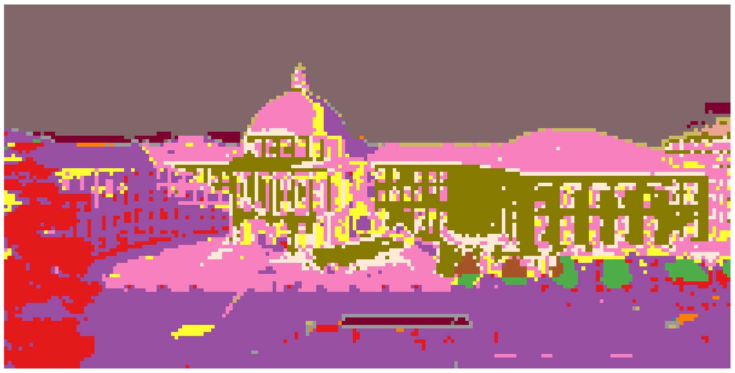
Figure 1: Mean shift with bandwidth = 3

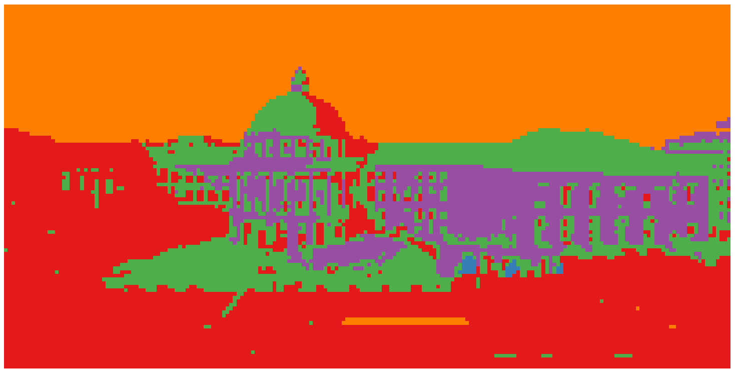
Figure 2: Mean shift with bandwidth = 5

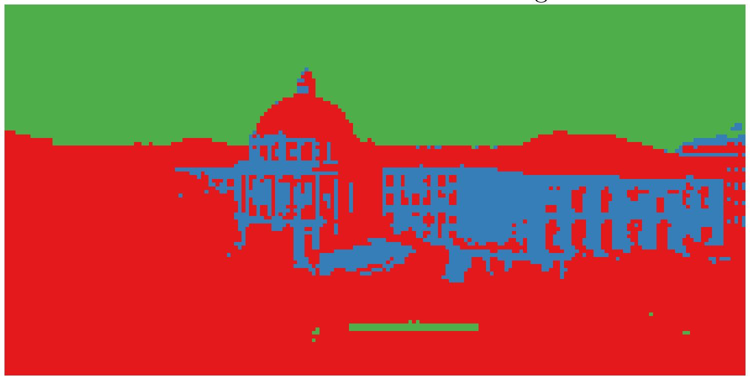
Figure 3: Mean shift with bandwidth = 7

In the application of the Mean-Shift algorithm to image segmentation, the choice of bandwidth is determines the granularity of the resulting segmentation A series of experiments with bandwidths set to 1, 3, 5, and 7 were conducted to observe the impact on the segmentation output.

Bandwidth = 3: The segmentation with a bandwidth of 3 exhibited moderately sized regions, with a lot of details but an acceptable region homogeneity. The segmented image displayed distinct color regions corresponding to various features within the image.

Bandwidth = 5: Increasing the bandwidth to 5 led to larger, more homogenous regions, with finer details beginning to merge. The reduction in the number of distinct regions is indicative of the algorithm's tendency to converge towards broader clusters.

Bandwidth = 7: At a bandwidth of 7, the algorithm produced even coarser segmentation. This result was characterized by a further reduction in the number of regions, with only the most prominent features being delineated.

Bandwidth = 1: Conversely, a bandwidth of 1 resulted in an error due to an out-of-bounds index when attempting to reshape the label array. This error suggests that the segmentation produced a number of clusters exceeding the available labels in the color mapping array. A potential solution to this issue involves either increasing the size of the color mapping array to accommodate a greater number of clusters or adjusting the algorithm to merge closely positioned centroids to reduce the total count of clusters. One way of furhter merging clusters without modifying the algorithm would be to simply increase the number of iterations of the `meanshift_step` function. For example, having 30 iterations works for bandwidth =2.5 but 20 iterations do not work (we get again more than 24 clusters). However, increasing too much the iterations could also result in an unnecessary computational burden without significant improvement in segmentation quality. To solve this issue, we could combine this increased iterations with an additional stopping criterion (for example if the clusters have not moved much or we have reached a small enough number of clusters).

In the end, we observe that lower bandwidth values preserve detail but may lead to over-segmentation, as the influence of nearby points is more pronounced compared to points further away causing the algorithm to detect more clusters. Conversely, higher bandwidth values smooth over finer details, potentially under-segmenting the image, since the increased influence (weights) of points further away results in more points being averaged together, thus merging smaller clusters into larger, more homogeneous ones.