Lab 5: Image Segmentation with Mean Shift

Orion Junkins

November 24, 2023

Mean Shift Implementation

This lab implements an iterative mean shift segmentation algorithm with a Gaussian weighting function. The algorithm takes in bandwidth as a parameter, corresponding to the σ value in the Gaussian weighting.

To compare pixel values, the image is converted to the CIELAB space, where numerical changes in color value correspond proportionally to the perceived difference between the two colors. This allows the distance between two colors to be a visually meaningful concept.

The algorithm takes a 1D numpy array of pixel values for the flattened CIELAB image as input and runs for 20 iterations. Each iteration creates a new shifted version of the image, where pixel values are gradually shifted into clusters in color space. Final clusters are mapped to 24 colors to show results.

Every iteration operates on every pixel in the image, calculating each pixel's new location in color space. Calculating a pixel's new location involves 3 steps: distance calculation, gaussian weight calculation, and point updating.

Distance Calculation

Distance calculation takes in a single reference pixel, *x,* and a vector of all other pixels, *X* of shape *(n,3)*, where *n* is the number of pixels in the original image. The result is a vector, *d,* of shape *(n, 1)*, containing the Euclidean distance between the pixel *x* and every other pixel in *X* and is calculated as follows.

$$d_x = ||X - x||_2$$

In Python:

$$d_x = np.linalg.norm(X - x, axis = 1)$$

Gaussian Weight Calculation

Gaussian weight calculation takes this distance vector *d* and some specified bandwidth σ. A new *(n, 1)* weights vector ω is calculated as follows, containing the Gaussian weights of the distances.

$$\omega = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(d)^2}{2\sigma^2})$$

In Python:

$$\omega = \frac{1}{np.sqrt(2 * np.pi * np.power(bandwidth, 2)} * np.exp(-\frac{(dist**2)}{2 * np.power(bandwidth, 2)})$$

Note that this vector ω is not normalized - normalization is handled in the next step.

Point Updating

Point updating calculates the new value, x', for the current pixel, x, as the weighted average of X with the ω vector found above. x' is defined as follows.

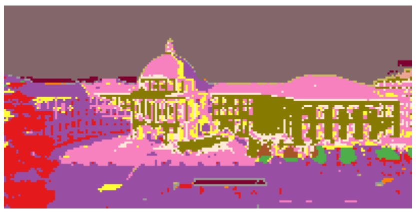
$$x' = \frac{\sum_{i=1}^{n} X_i * \omega_i}{\sum_{i=1}^{n} \omega_i}$$

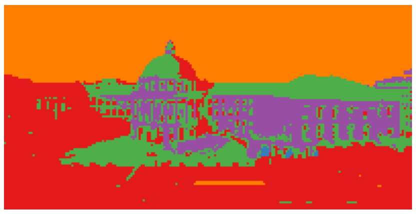
In Python:

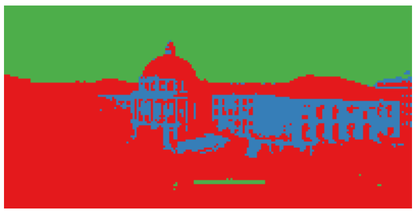
$$x' = np.average(X,\ axis = 0,\ weights = weights)$$

Results

This lab experiments with bandwidth values of 1, 2, 2.5, 3, 5, and 7. Values below 3 (1, 2, 2.5) converged to 282, 44, and 25 classes respectively. The provided code maps results to 24 distinct color values, so these trials all resulted in failures. Trials with bandwidths of 3, 5, and 7 succeeded, converging to 15, 5, and 3 classes, respectively. The results are shown below.


$\sigma = 3$


$\sigma = 5$


$\sigma = 7$

Given the infinite window size used, all values will theoretically converge to a single class given enough iterations regardless of bandwidth size (in practice, this may not happen for very small bandwidths due to rounding precision). Thus, the bandwidth parameter does not affect the degree of convergence possible but rather the speed at which convergence occurs.

Smaller bandwidths cause pixel updates to primarily consider their local region in color space, resulting in small, incremental update steps. Larger bandwidths cause pixels to more heavily consider pixels further away, yielding larger update steps. Stopping this process at 20 iterations gives the results seen above. Smaller bandwidths converge to many classes with noisy but very precise segmentation. The larger bandwidths converge further, with smoother, albeit less precise segmentation.

These conclusions also explain why small bandwidth values fail. Given their slow convergence speed, 20 iterations were insufficient for these small bandwidth trials to converge to <24 points.

This failure could be addressed by increasing the number of iterations to allow further convergence. Alternatively, the image could be downsampled to a lower resolution before running the mean shift, or the outputs could be binned such that nearby values are assigned the same color. However, downsampling or binning would result in a loss of output quality. Increasing the number of iterations and/or using a larger bandwidth value to increase convergence speed are likely the best solutions for most use cases.