1 Mean-Shift Algorithm

1.1 *Implementation Details*

Since the image was already in CIELAB space, I used this as the feature space to work on. The *distance* function I implemented computes the 2-norm (*distance*) of the difference between the pixel feature-pairs for the whole image. These distances are computed for every pixel pair as the radius to consider is infinity (in the batched version *torch's cdist* function has been used).

The weights are computed as $e^{-d^2/2h^2}$; where d is the distance (the 2-norm mentioned above)and h is the bandwidth. Using these weights, for every pixel a weighted mean of the feature vectors (of all the pixels) is calculated. This mean is the new feature vector that would be assigned to the pixel being processed. Again, in batched-version of the *update_point* function, the mean vector calculation for all the pixels is done efficiently by grouping the weights and pixel feature-vectors appropriately in matrices. Once new values have been calculated and assigned for every pixel, one step of the mean shift is complete.

With each step of the mean-shift, the vectors corresponding to every pixel shift towards the closest mode. As multiple steps of mean-shift are run, the feature vectors of the pixels converge to a finite number of unique vectors (which are *modes* or *attraction basins*). Each of these unique modes can be thought of different classes, and different colours are used for these classes to generate a segmentation map (*result.png*). With *bandwith* = 2.5, a total of 12 such modes were computed after 20 mean-shift steps.

1.2 *2.4 Accelerating the Naive Implementation*

For the batched version, I leveraged the inbuilt pytorch functions for performing operations on all the pixels instead of looping over them. This gave a speed-up gain of 15.24 . (All the experiments were run on *11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz*)
Average execution time (without batching) : 35.05 s
Average execution time (with batching) : 2.30 s