

Problem Set 4

CMSC 426

Assigned Tuesday, October 25, Due Tuesday, November 1

The goal of this assignment is for you to implement the K Means algorithm to cluster pixels in an image by their intensity. This algorithm works iteratively. We are given an image and a number k , which indicates the number of clusters we are to produce. First, we guess some reasonable values for the central intensity of each cluster. Then we alternate between performing the following two steps. 1) Every pixel is assigned to a cluster. A pixel is assigned to the cluster whose central intensity is closest to the pixel's own intensity. 2) For every cluster, we recompute the central intensity as the mean intensity of all pixels that have been assigned to that cluster. We stop this iteration when the centers of the cluster stop changing.

Here's an example of how this might work. Suppose the "image" consists of three pixels with intensities: 0, 100, 255, and we want to form these into two clusters. We might start out by guessing random intensities between 0 and 255 for the two cluster centers. For example, we might guess the values 30 and 140. Then we would assign the pixel with intensity 0 to the cluster centered at 30, and we would assign pixels with intensities 100 and 255 to the cluster centered at 140. Next, we would assign the first cluster a center of 0, and the second cluster a center of 177.5. In the next iteration, none of the cluster assignments would change, so none of the cluster centers would change, and we would stop.

1. **25 points** Write a function that will take, as input, a vector of cluster centers, and an image, and that will return a representation showing which pixels are assigned to which clusters. For example, given input of cluster centers = [17 23] and an image = [3 19 22 4 17 44 5] the output might look like: [1 1 2 1 1 2 1]. You do not need to represent your input or output in this fashion, however; just document your representation (in my code, I find it easier to use a 3D matrix to represent cluster membership). If a pixel intensity is equidistant from two cluster centers, you can handle this in any way you choose. Of course, although this example is 1D, your code must work for a 2D image. **Hint:** Be sure to convert images from uint8 to double before performing any arithmetic operations.

Turn in your documented code, and a run showing the output of your code on the example given in the previous paragraph.

2. **25 points** Write a function to display your results. This function should take as input the cluster centers, and a representation that shows which cluster each pixel is assigned to. Use these to generate a new image, in which each pixel intensity is replaced by the value of the cluster center it has been assigned to. For example, with input like: [17 23] for the cluster centers, and [1 1 2 1 1 2 1] for the cluster assignments, this function will create an image: [17 17 23 17 17 23 17].

Test this function using the swanbw.jpg image. Use your function in problem 1 to assign each pixel to one of the centers 90, 150 or 230. Then use these assignments to generate an image in which each pixel is either 90, 150 or 230. Display this image using imshow (keep in mind, that to use imshow, your image should be converted back to type uint8. There is a function called uint8 that will convert a matrix to this type). Turn in your documented code, and the image you generate. The image our code produces is shown in swanbw_KMeans_90_150_230.jpg.

3. **25 points** Write a function that will use the pixel assignments computed by the function you produced for Problem 1, along with the image, to compute the centers of these clusters. For example, this might take as input [1 1 2 1 1 2 1] and [3 19 22 4 17 44 5] and return: [9.6 33].

Turn in your documented code, and a run showing the output of your code on the example given in the previous paragraph.

4. **25 points** Now complete your implementation of Kmeans clustering. Test your implementation by running on the image [0 100 240], forming it into two clusters. Run the algorithm 10 times on this input, and comment on which answers you get, and how often you get them. Turn in your documented code and printouts of the results.

Now run your code on the swan image, forming 4 clusters. Display the resulting image in a figure, and turn this in. Run your code five times, and comment on whether you notice any differences in the output on different runs.

5. **Challenge Problem 10 points:** Extend your K Means program to work on color images. To do this, you'll have to define a distance between two RGB values for a pixel. A simple and reasonable thing to do is to just use the sum of squared differences between the red, green and blue channels of the image. Turn in your documented code, and some examples of the results it produces on some color images.
6. **Extra Challenge Problem 20 points:** Implement EM for color images, and compare the results to K Means. This will require some research on your part, because I haven't really explained EM in any detail. You may find it helpful to look at Yair Weiss' EM tutorial. There's a link to this on our web page.