UNIVERSITY OF ROCHESTER
CSC 249, MACHINE VISION
SPRING 2017

Homework 3

Nathan Conroy

March 16, 2017

1 K-MEANS CLUSTERING IMPLEMENTATION

For this assignment, we were required to implement the K-means clustering algorithm in order to perform segmentation of an image provided to us (seen in Figure 1.1). The algorithm works by initially determining a number of clusters, or segmentations, for the parts of the image to be classified. Each cluster is then given an initial mean based on the value of the feature vector of the image. The next step, which is called the M-step, involves cycling through the image and assigning each pixel to the closest cluster centroid. The next step, which is called the E-step, involves recalculating the cluster means to be the center values of all of the pixels currently classified within that cluster. The M and E steps are then repeated a number of times until variations between iterations have decreased below a certain threshold or an iteration cap has been reached. Depending on the quality of the choices for the number of clusters and initialization values, the result will be a successfully segmented image.

Figure 1.1: original image

In my implementation, I allow the user to change the number of K-clusters by setting a variable, K. I also allow the user to choose the color space by changing the variable color_space to one of 3 constants, RGB, LST, or HSV. If the user selects LST or HSV, the original image is converted to the specified feature vectors before the algorithm is run.

In the M-step of my implementation, I find the distance between any given pixel and the clusters by calculating the Euclidean distance between the pixel and each K mean and storing it in an array of size K. The closest cluster label is then found by finding the minimum value in the array and taking its index. The label is assigned to a 2D array equal in size to the original image at the corresponding pixel, representing the result of the algorithm at the end of the iteration. I then iterate the value in an array called KCounts at the same index to keep track of the number of pixels within that cluster. I also add the values of each vector attribute to the corresponding spot in the means array as a temporary place holder to calculate the average in the E step.

In the E-step of my implementation, I update the K-means by simply dividing the sum that I have accumulated for each vector attribute in the means array by the number stored in the KCounts array at the corresponding index. I also check to see if there should not be another iteration by checking if the current iteration is the 30th (which I've set to be the cap), and if the variation within means for each cluster between the current iteration and the previous is less than a threshold T, which I've set to .01. If either of these conditions are met, the loop is broken and the result array is displayed using imagesc and saved.

2 RGB Color Space

For each of the color spaces that I tested, I tried to use a variety of different K values and initializations. For the RGB color space, I tested 4 K values: 2, 3, 4, and 5. For each of these K values I used a set of initializations that was evenly spaced between 25 and 229 for all vector attributes (R, G, B), as well as a set of initializations that started at 10 and increased by 10 for each cluster.

For example for K = 3, I tested a set that had R,G,B values of 25 for cluster 1, 127 for cluster 2, and 229 for cluster 3. I also tested a set that had R,G,B values of 10 for cluster 1, 20 for cluster 2, and 30 for cluster 3.

The results of all of my tests are seen below.

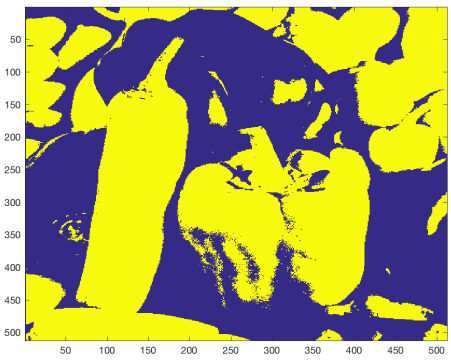

Figure 2.1: K = 2, inits: 25, 229

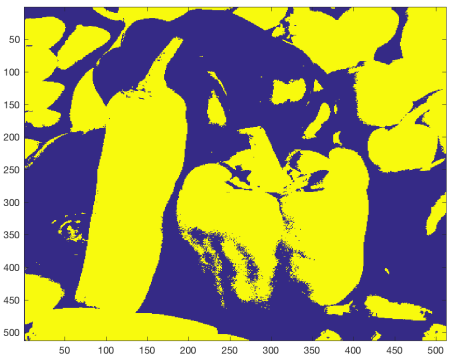

Figure 2.2: K = 2, inits: 10, 20

As you can see, the initialization differences didn't seem to affect too many of the outcomes of the segmentations. For K = 2, the resulting image is the exact same for both tests. The algorithm seemed to segment the green peppers with some of the specular highlights (yellow), and the red peppers with the background and shade (blue).

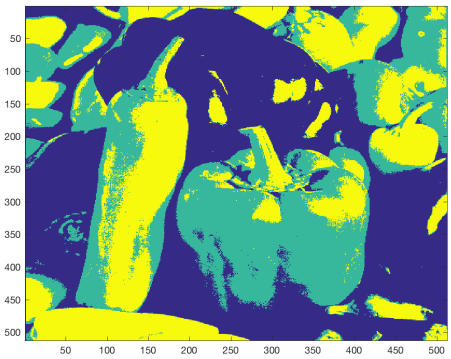

Figure 2.3: K = 3, inits: 25, 127, 229

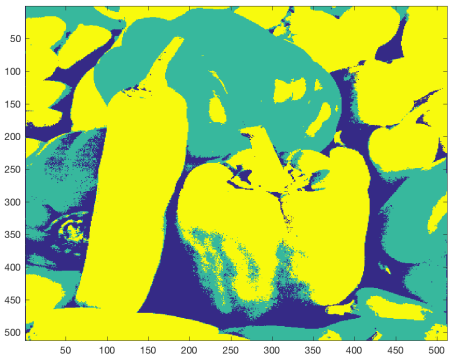

Figure 2.4: K = 3, inits: 10, 20, 30

K = 3 seemed to show the biggest difference between tests. The test with the evenly spread initialization values segmented the red peppers in with the background (blue), the green peppers (green), and the specular highlights got their own segmentation (yellow). Meanwhile the lower set of initializations seemed to segment the background correctly (blue), as well as the red peppers (green) and the green peppers (yellow), and the specular highlights were ignored. This makes sense because in the first test, the initial mean of 229 was very close to the color of the specular highlights, while in the second test even the highest initial cluster mean was below the values of most pixels in the image.

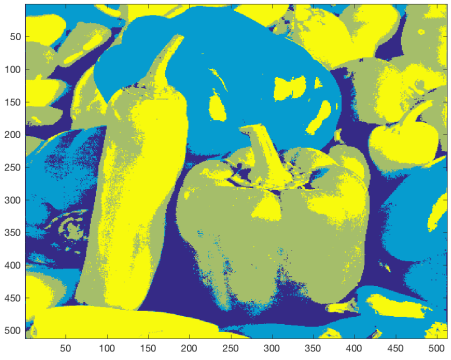

Figure 2.5: K = 4, inits: 25, 95, 168, 229

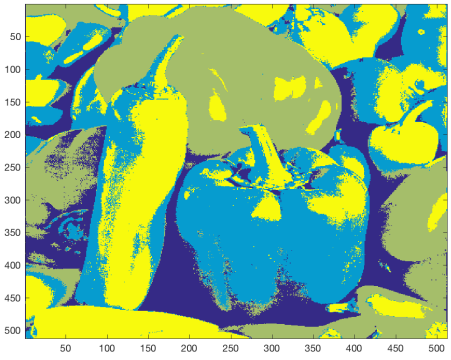

Figure 2.6: K = 4, inits: 10, 20, 30, 40

K = 4 seemed to react similarly to what was seen in K = 2, the segmentations were nearly identical but the labels were slightly different which is why the image sections appear differently colored when displayed using imagesc. The algorithm seemed to segment the green peppers, the red peppers, the background and shade, and the specular highlights into separate clusters.

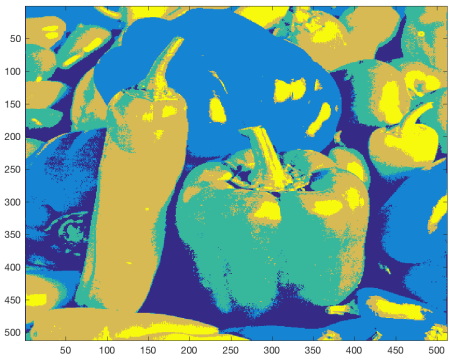

Figure 2.7: K = 5, inits: 25, 76, 127, 178, 229

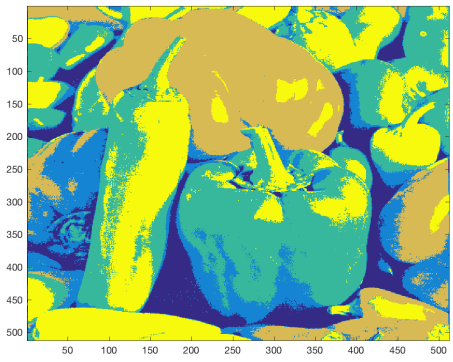

Figure 2.8: K = 5, inits: 10, 20, 30, 40, 50

K = 5 seemed to show minimal changes between the two tests as well. This run through seemed to segment regions even more specifically. The first test (evenly spread initializations) seemed to segment the image into background, red peppers, green peppers, slight specular highlight, and extreme specular highlight. Meanwhile the lower set of initializations segmented the image into the background, shade on the peppers, red peppers, green peppers, and specular highlight.

3 Lst Color Space

For the Lst color space, I tested the same 4 K values used in testing RGB: 2, 3, 4, and 5. For each of these K values I used a set of initializations that was evenly spaced between 0 and 100 for all vector attributes (L, s, t), as well as a set of initializations that started at 10 and increased by 10 for each cluster.

The results of all of my tests are seen below.

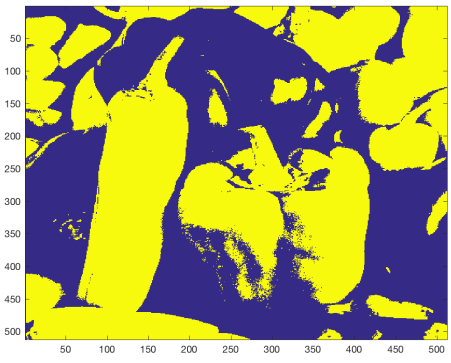

Figure 3.1: K = 2, inits: 0, 100

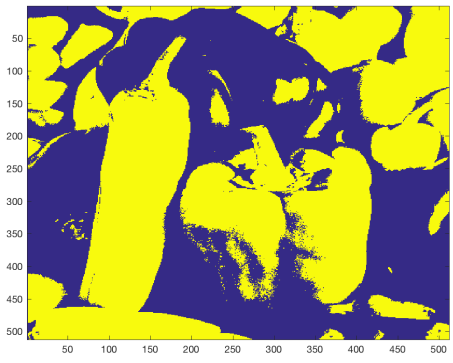

Figure 3.2: K = 2, inits: 10, 20

As you can see, the initialization differences for this color space seemed to show even less variability in the outcomes of the segmentations than RGB. For K = 2, the resulting image is the exact same for both tests, just as it was in RGB. The algorithm seemed to segment the green peppers with some of the specular highlights (yellow), and the red peppers with the background and shade (blue).

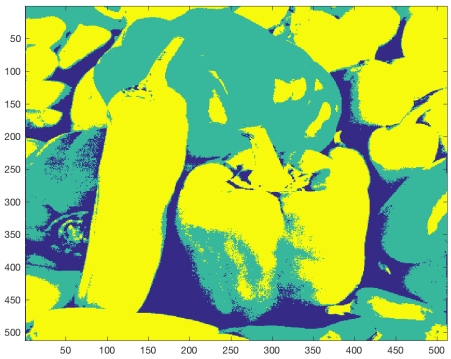

Figure 3.3: K = 3, inits: 0, 50, 100

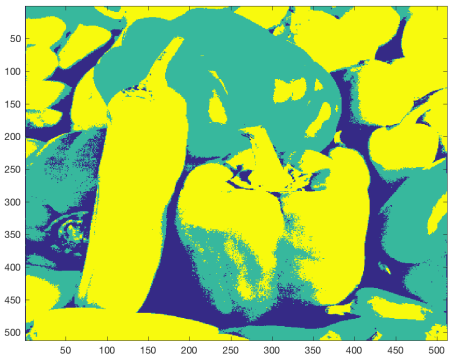

Figure 3.4: K = 3, inits: 10, 20, 30

Just like with K = 2, for K = 3 the resulting image is the exact same for both tests. The image is essentially the exact same segmentation as the second RGB test. See analysis above.

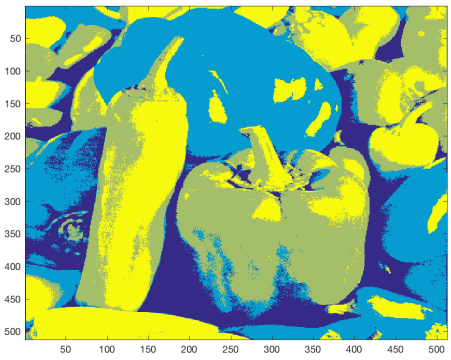

Figure 3.5: K = 4, inits: 0, 33, 66, 100

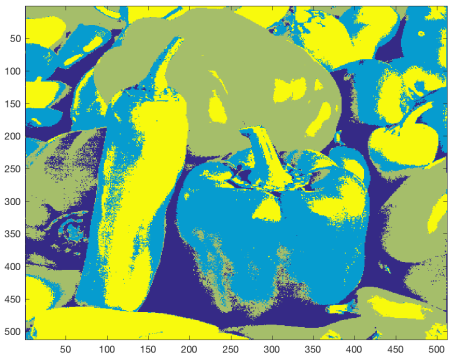

Figure 3.6: K = 4, inits: 10, 20, 30, 40

K = 4 also produced the same segmentation result for both tests, and are nearly identical to those seen with RGB. See above.

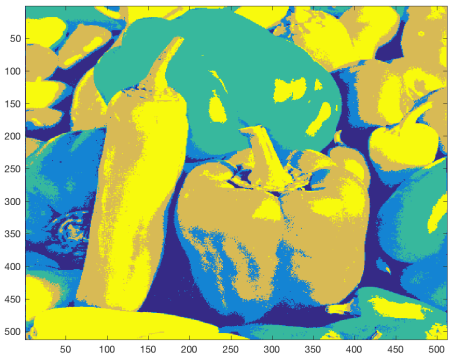

Figure 3.7: K = 5, inits: 0, 25, 50, 75, 100

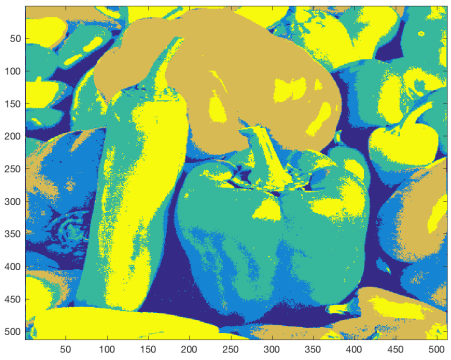

Figure 3.8: K = 5, inits: 10, 20, 30, 40, 50

K = 5 also produced the same segmentation result for both tests, and are nearly identical to those seen with RGB. See above.

4 HSV Color Space

For the HSV color space, I tested the same 4 K values used in testing RGB and Lst: 2, 3, 4, and 5. For each of these K values I used a set of initializations that was spread across the full range for all vector attributes (H, S, V), as well as a set of initializations that started at .1 and increased by .1 for each cluster.

The results of all of my tests are seen below.

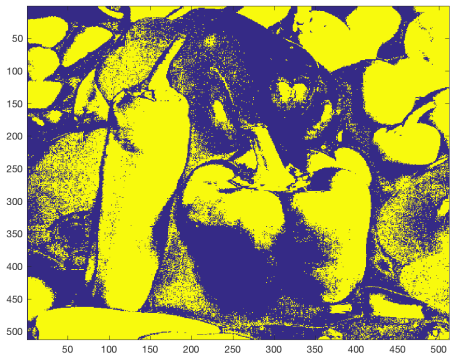

Figure 4.1: K = 2, inits: .25, .7

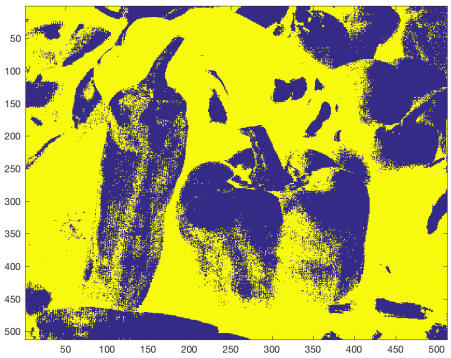

Figure 4.2: K = 2, inits: .1, .2

The results from the HSV color space were significantly different from those seen in the RGB and Lst segmentations. For K = 2, the resulting segmentations seemed much more chaotic than those seen before. In the first test (spread out initializations) the first segmented cluster seemed to be the background, the red peppers, and shade on the green peppers, while the second cluster seemed to be the green peppers, and the specular highlights. In the second test (lower value initializations), the segmented regions seemed to be the green peppers, and everything else.

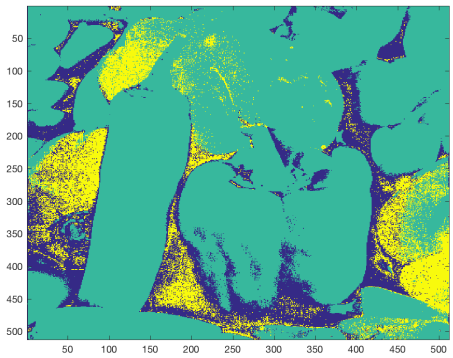

Figure 4.3: K = 3, inits: .3, .5, .7

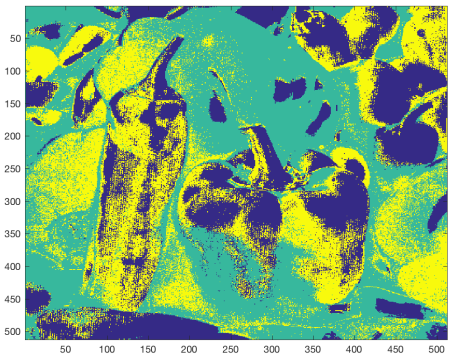

Figure 4.4: K = 3, inits: .1, .2, .3

For the first test using K = 3, the middle cluster swallowed up most of the image (all of the green peppers and some of the red peppers). The background and some slight shading made up one of the other clusters, and some bits of red pepper made up the last. For the second test, the result was even more chaotic, leaving some peppers with a large amount of all 3 clusters within them. One cluster seemed to swallow the background, the shading, as well as the outlines of some of the peppers. Another seemed to segment most of the peppers together, while the last segmented the specular highlight. It seems that this test produced the worst segmentation out of all of the tests run so far.

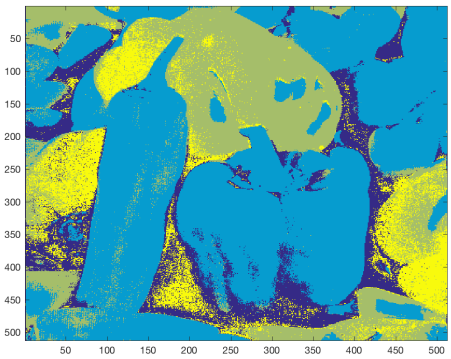

Figure 4.5: K = 4, inits: .3, .43, .57, .7

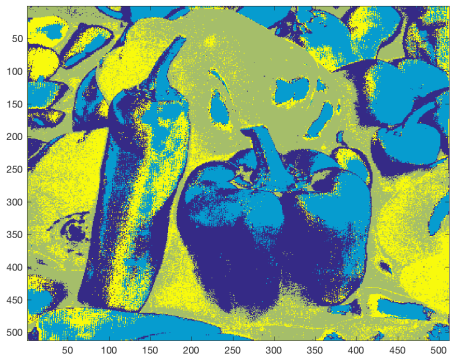

Figure 4.6: K = 4, inits: .1, .2, .3, .4

K = 4 seemed to do a much better job at clustering than K = 3 for the HSV color space. In the first test, most of the green peppers were clustered into the same segmentation (light blue), while the red peppers were mostly clustered into 2 other segments (yellow and green). The background made up the last segment (dark blue). The second test, like in the other examples, seemed to be very chaotic. The background was not well distinguished from the red peppers, and therefore they were clustered together. The specular highlights also were more of a factor in this example, and were present on most of the peppers.

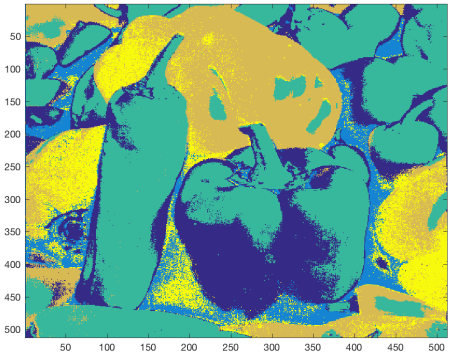

Figure 4.7: K = 5, inits: .3, .4, .5, .6, .7

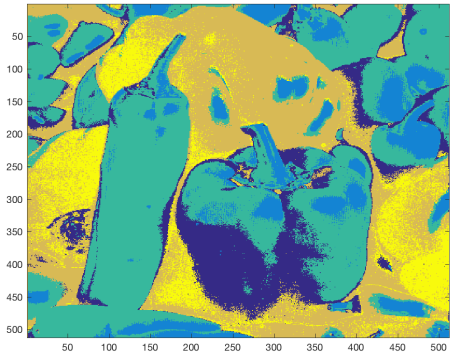

Figure 4.8: K = 5, inits: .1, .2, .3, .4, .5

K = 5 seemed to do a worse job than K = 4, but a better job than K = 3 in this color space. in the first test, the algorithm seemed to cluster the background, the green peppers, shade on the peppers, and the red peppers all separately. The shade was a bigger factor than one would want when attempting to segment components in this example. Like all of the others, the second test did not handle the darker shades very well. While the green peppers seemed to be clustered fairly, the red peppers morphed with the background, making the components almost undistinguishable.

5 Conclusion

Overall, I would say that using RGB or Lst as the color spaces seemed to work much better than using HSV in my implementation of K-means clustering. The segmentations seemed to fit the image much more accurately and they reacted much better to shading and specular highlights. Using a high K value seems as if it would work best in an image with many different colors present, but the image used in this example really did not have much variation in color. Using a K value of 2 often didn't seem to do very much aside from distinguishing the background from the foreground, but using a K value of 5 seemed to be too much in most

tests, segmenting the same pepper into many different clusters based on shade and brightness. I would say that from my findings, a K value of 3 or 4 seemed to fit very well with this image, separating the different colored peppers very nicely. To my surprise, setting the initializations to all be low values seemed to perform better or the same in some cases as evenly spreading them across the range of values. The only exception where this did not hold was in the HSV color space, which performed much worse overall. I would say that one of the best segmentations can be seen in Figure 2.4, which used a K value of 3 and initial means of 10, 20, and 30 for R, G, and B. In this example, specular highlights did not seem to distort the clustering, and the background was nicely defined.

In order to handle specular highlights, I think that one possible solution could be to add an extra cluster than one would normally use for the given image, and evenly space the initial cluster means across the range of values. Then at the end, the specular highlighting could be eliminated by removing the pixels with the highest label (the brightest values) and blending them with surrounding label values. This would work well in an example similar to the clustering I obtained in Figure 2.7 which used the RGB color space, a K value of 5, and initial means of 25, 76, 127, 178, and 229. If one were to remove the yellow regions of this result and blend them with the surrounding values, the result would be a near perfect segmentation.