COSC450: k-means Segmentation

Ashley Manson

October 5, 2015

Department of Computer Science
University of Otago

1 My Code

The code is compiled with the included Makefile, if the setPath.sh has run sucessfully on the lab machines.

My program takes 7 command line arguments in this order:

1. The input image to perform the segmentation on.
2. An output image to save the result.
3. k for the number of clusters.
4. Cluster type which is a number between 0 and 2.
   a. 0 is for random selection.
   b. 1 is for random clustering.
   c. 2 is for k-means++.
5. Distance type is a number between 0 and 1.
   a. 0 is for comparing RGB values.
   b. 1 is for comparing HSV values.
6. Termination type is a number between 0 and 2.
   a. 0 is for giving the maximum iterations before terminating.
   b. 1 is for terminating once the clusters have stopped making changes. A value at the top of the file CLUSTER_MIN_CHANGE can augment this.
   c. 2 is for terminating once the labels have stopped making changes. A value at the top of the file LABEL_MIN_CHANGE can augment this.
7. Max iterations is for giving a maximum number of iterations before the program will terminate. Will only be used if 0 is given for the end type.

Once the first image has been displayed, a key press will begin the k-means segmentation. Output will be printed to stdout of the progress so far. Once the k-means algorithm completes, the segmented image will be displayed. Sometimes it displays behind the original, so moving the original image will reveal the segmented image. Another key press once the segmented image is displayed will close both windows. The segmented image will be saved to the given command line argument for viewing later.

2 Implementation

2.1 Initialisation Methods

I have implemented the three initialisation methods that were proposed in the Assignment 2 specifications; random selection, random clustering, and k-means++.

2.1.1 Random Selection

This method of initialisation is the fastest. For each k cluster point, it will get a random x and y coordinate that corresponds to a pixel in the image. This is then added to a vector of Vec3b for computation later.

2.1.2 Random Clustering

This initialisation method is slower than selection as it goes through all the pixels in the image. Firstly, for each pixel in the image, it is assigned a random k cluster. Once all pixels from the image have been assigned a k cluster, I go through each cluster, calculating the average colour across all the pixels. Once the average has been calculated, the colour average is added to the clusters vector for computation later.

2.1.3 k-Means++

This method of initialisation takes the longest due to the random distribution that contributes to the selection of clusters. Firstly, the first cluster is selected at random from the image. This serves as the first cluster point. The first loop then goes over the rest of the clusters that need to be selected. The first inner loop then goes over the image calculating the distances from the selected cluster, storing the result in its corresponding index in a vector and adding to a count. The second inner loop then goes on until another cluster is added to the cluster vector. Here there is a chance for one of the distances in the vector to be selected based on a random distribution of all distances calculated. Once a new k cluster has been found, it is added to the clusters vector, and this loop exits. All of this is repeated until all k clusters are found.

2.2 Distance Metrics

I have implemented two different ways of calculating the distance of two pixels, one being the raw RGB values, and the second being the HSV of a pixel, ignoring the value. For each pixel in the image, I go through and calculate the distance from it and each of the k cluster points. If the distance is less than the previous distance, that pixel label gets updated and it gets a new minimum distance set for it. For RGB, I simply calculate the difference between the corresponding RGB values. HSV is the same, except I ignore the value when calculating the distance.

2.3 Termination Conditions

I have implemented three different ways for the k-means program to terminate; you can pass in a parameter to set the maximum number of iteration that the algorithm will run for, it can be set to check the differences in cluster changes, or it can be set to check how many labels are changing between iterations.

2.3.1 Maximum Iterations

The algorithm can be set to run for a set number of iterations, no matter if it has stabilized or still has yet to stabilize. I have found that when k is set to 10, that 50 iterations results in a pretty good outcome with the segmentation of the image.

2.3.2 Cluster Changes

The program can be set to check for changes in cluster points. With every iteration, the difference between cluster changes is calculated, and if it is less than a prespecified minimum change, then the algorithm will terminate. I have found that when k is set to 10, on a picture with lots of varying colour, clusters will stabilize around 50 iterations, however on an image that is made up of mostly one colour, white for example, the clusters take longer to stabilize and the iteration count approaches or exceeds 100.

2.3.3 Label Changes

The program can be set to check for changes in labels for each pixel. With every iteration, the number of changed labels is summed up, and if it is less than a prespecified minimum change, the the algorithm will terminate. This method of termination has a similar result as cluster changes, with images made up of lots of colours stabilizing faster than an image made up of mostly one colour.

3 Experiments

I have experimented around with Random Selection and k-Means++, as well as testing each initialisation method with both RGB and HSV. I have ran each with k clusters of; 4, 6, 8, and 10. I have taken the average of each to find out which is faster to segment the image, and have included a resulting image for each.

4 Strengths and Weaknesses

Random Selection seems to perform worst than the others even though its initialization takes the least amount of time. Overall the resulting images typically miss colours that were in the original image, some colours like brown seem turn into a darker colour like black. This is a result of how the selection of clusters happens as if there is one missing colour from all the colours selected as

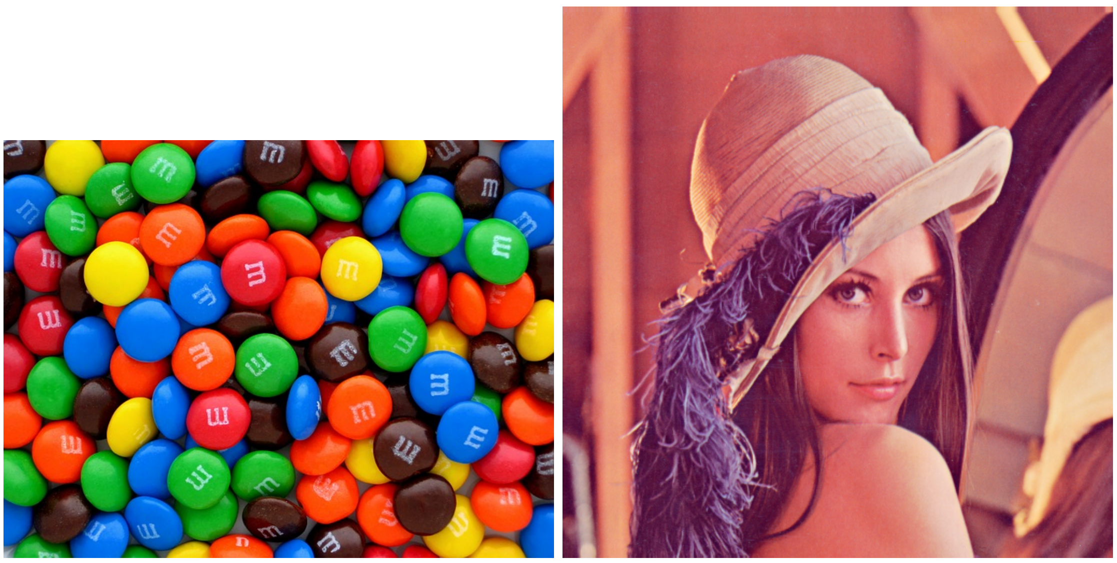

Figure 1: Images used for testing, mms.jpg and lena.png.

the starting clusters, then when the overall colours start to average, they will move away from these colours and result in a different result.

Random Clustering is by far superior than Random Selection as before the k-means algorithm starts, the cluster colours are already averaged out over the entire image. It allows for a more wide range of colours to be initialized as the cluster colours. I have found with my testing that it helps with getting out more colours from the image, which allows for when the segmentation to occur that a lot of the colours appear in the final segmentation of the image. k-means++ I have found is as good or better than Random Clustering. When k-means++ is ran on a image it is good at getting more colours as the initial clusters. I have found that when the colours are selected that there is more smoothness around the edges of colours, such as when there are light spots or shadows. Another thing I have noticed is that when there is a lot of colour in the image white tends to be able to pop out and be properly segmented, whether in the other two methods of initialization make the white blend in with other colours.

In my implementation of RGB and HSV distance metric, RGB tends to perform better than HSV, especially when there is a lot of colour in an image. The resulting segmented images that

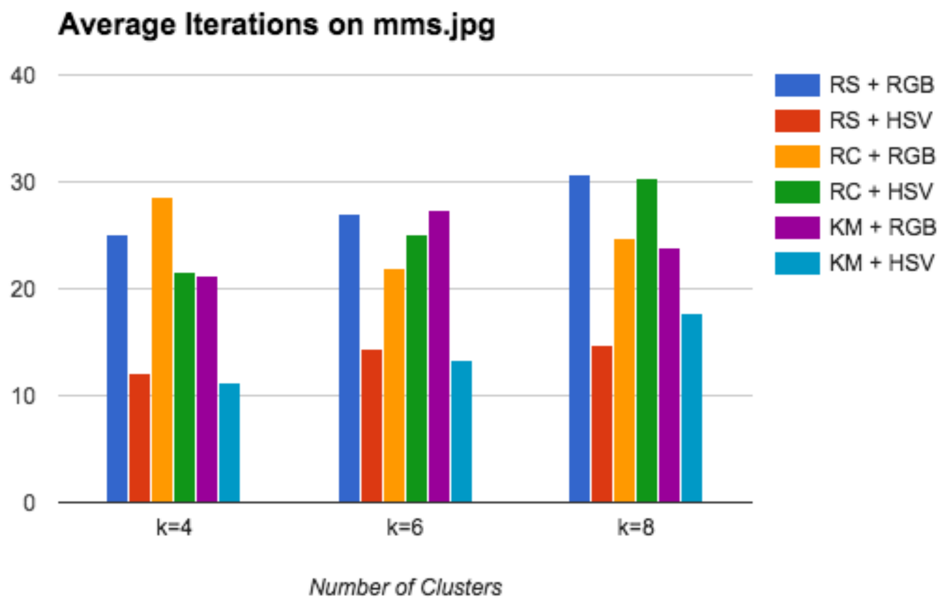

Figure 2: Average iterations on mms.jpg.

RGB returns have a better image quality than HSV. However, on images that are made up of predominantly one colour, HSV correctly segments the image very well, compared to RGB.

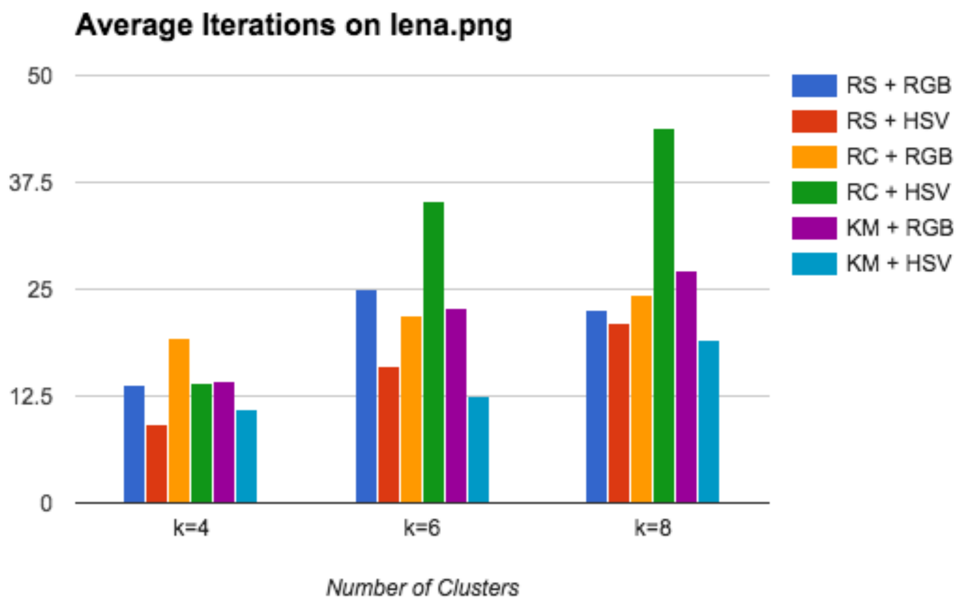

Figure 3: Average iterations on lena.png.