CSE 6740 - HW 1: Image Compression Programming Report

Manvitha Kalicheti
gtID: 903838438

September 21, 2022

1

For the kmedoids algorithm, ideally, the cluster centroid would be the point with the least sum of distances from all other points in the cluster. This is where I started from. When I used simple for loops to implement this, I realised it takes way too long even for small K values due to the very large size of the data set (pixels in our image). I vectorised this code to use matrix methods, then I ran into memory allocation errors due to the huge matrix size requirements (n*n to store pairwise distances).

I designed a modified method to handle this problem. After initialising cluster centers randomly, I found J (I took J = 10000) points closest to the each center. Then, I computed the distortion function for each of these J points (sum of euclidean distances between point j and all points in cluster). I picked the point with the least distortion value to be the next center for the cluster and repeated this for each cluster.

The entire process is performed iteratively till the cluster centers stop moving. The centers will stop moving when the distortion value can no longer be reduced by changing the cluster centers.

I tried the euclidean and manhattan distance metrics using the scipy function spatial.distance.cdist. I noticed better results with euclidean with only a slight increase in running time. So I decided to stick with euclidean.

If an empty cluster occurs, I call the kmedoids function again, but this time with the next lower value of K. This is done iteratively till we find a value of K for which we do not get empty clusters, and kmedoids is run with this K.

2

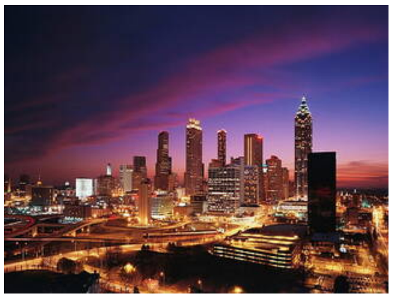
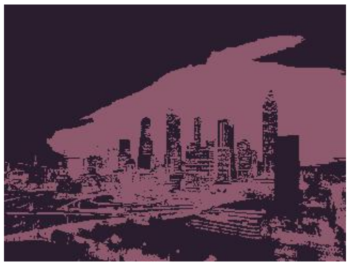
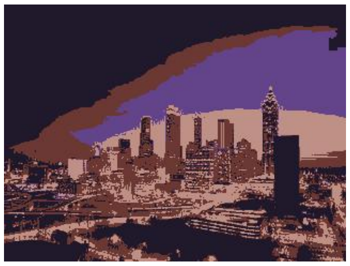


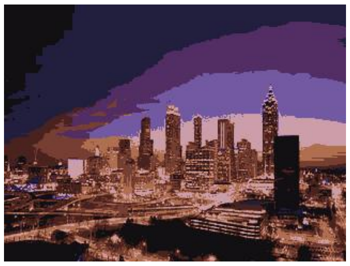
Figure 1: My choice of image, dimensions: 320 x 240

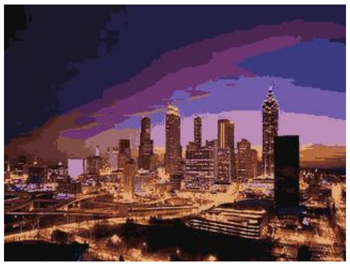
3

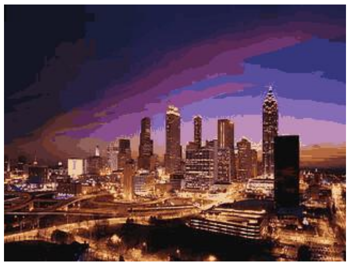
Kmedoids

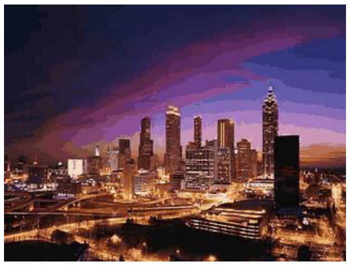

(a) k = 2

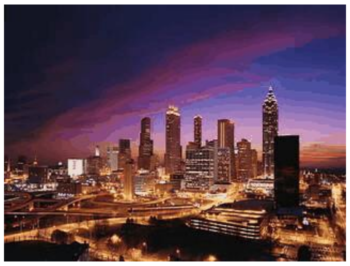

(b) k = 4


(c) k = 8


(d) k = 16


(e) k = 32


(f) k = 64


(g) k = 128

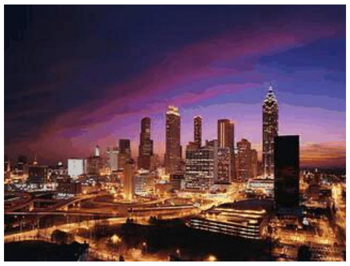

(h) k = 256

Figure 2: Kmedoids Image Compression

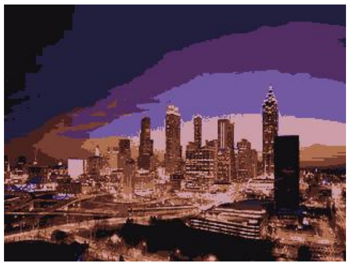
Given K	**Used K**	**No. of Iterations till Convergence**	**Running time (s)**
2	2	8	444.7471
4	4	15	383.2027
8	8	15	182.4632
16	16	15	55.4409
32	32	24	84.0324
64	64	28	112.4823
128	128	13	15.6823
256	172	2	2.5785

In the above table, the 'Used K' field is the final K value (one which does not give rise to empty clusters) used in the kmedoids program. We see that for a high value of K=256, we use K=172 as that is the closest value to 256 that does not give us an empty cluster.

Not much can be said about the relationship between K and required iterations for convergence (or running times). But it seems like as we increase K, due to lesser number of points in each cluster, the running time is following an almost decreasing trend. The compressed image also resembles the original image more and more as the K value increases.

4

Kmedoids Center Initialisation


(a) Randomly initialised centers

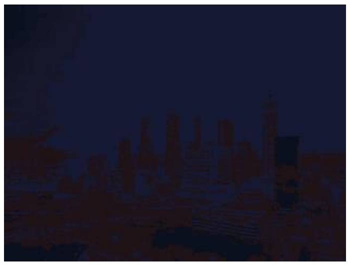

(b) Centers initialised from only the first 1000 pixels of the image

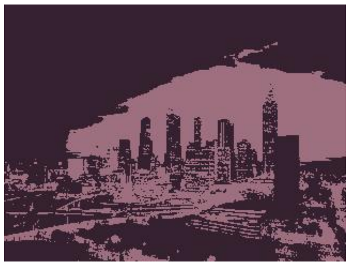
Figure 3: Kmedoids Image Compression with K = 8, center initialisation dependence

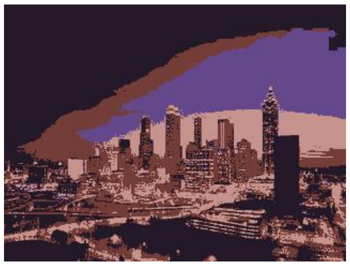
Initialisation	**K**	**No. of Iterations till Convergence**	**Running time (s)**
Random	8	15	182.4632
First 1000 pixels	8	2	2.3838

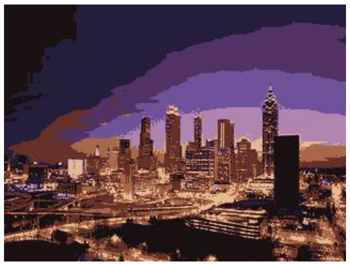
Different center initialisations lead to different final results. In figure 3(b), I forced my kmedoids algorithm to pick all the centers from within the first 1000 pixels of the image. As we can see this led to a vastly different, and much worse image.

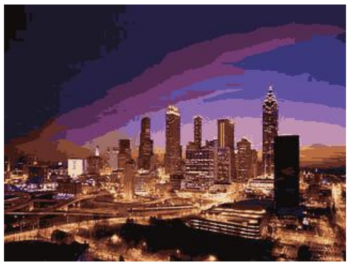
5

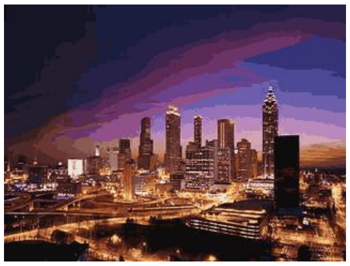
Kmeans

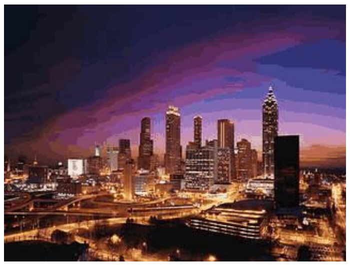

(a) k = 2

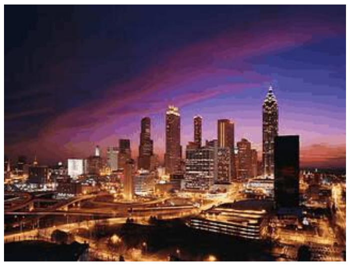

(b) k = 4


(c) k = 8


(d) k = 16


(e) k = 32


(f) k = 64


(g) k = 128

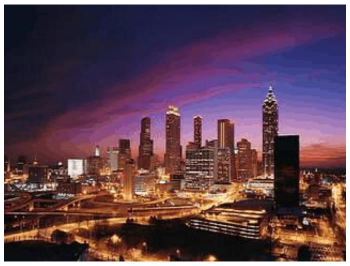

(h) k = 256

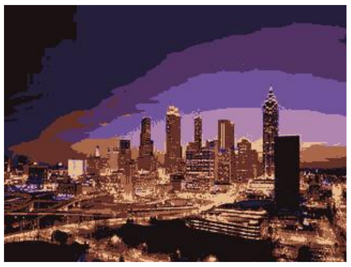
Figure 4: Kmeans Image Compression

Given K	**Used K**	**No. of Iterations till Convergence**	**Running time (s)**
2	2	18	12.7988
4	4	34	23.4731
8	8	43	29.7732
16	16	118	89.4325
32	32	194	188.5183
64	64	345	342.9209
128	128	325	377.8028
256	172	283	409.8322

In the above table, the 'Used K' field is the final K value (one which does not give rise to empty clusters) used in the kmeans program. We see that for a high value of K=256, we use K=172 as that is the closest value to 256 that does not give us an empty cluster.

We see that as the number of K increases, the number of iterations and the running times also increase. The compressed image also resembles the original image more and more as the K value increases.

Kmeans Center Initialisation


(a) Randomly initialised centers

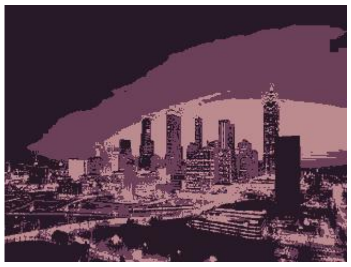

(b) Centers initialised from only the first 10 pixels of the image

Figure 5: Kmeans Image Compression with K = 8, center initialisation dependence

Initialisation	**K**	**No. of Iterations till Convergence**	**Running time (s)**
Random	8	43	29.7732
First 10 pixels	8	30	22.4709

Different center assignments led to slightly different final results. I forced the initial centers to be picked from the first 10 pixels of the image. This led to a worse final result as we can see from fig. 5.

Although kmedoids is a robust method than kmeans in general, the compromises I made due to the computationally expensive nature of kmedoids (using J closest points instead of all points in the cluster to choose the centroid) led to kmeans being a more robust method in this case.

The output quality is quite close for both methods. For small Ks, kmedoids shows longer running times than kmeans. But for larger Ks, it's the other way around.