Image compression using clustering

In this programming assignment, you are going to apply clustering algorithms for image compression. Your task is implementing K-means for this purpose. It is required you implementing the algorithms yourself rather than calling k-means from a package. However, it is ok to use standard packages such as file i/o, linear algebra, and visualization.

Formatting instruction

Input

- pixels: the input image representation. Each row contains one data point (pixel). For image dataset, it contains 3 columns, each column corresponding to Red, Green, and Blue component. Each component has an integer value between 0 and 255.
- k: the number of desired clusters. Too high value of K may result in empty cluster error. Then, you need to reduce it.

Output

- class: cluster assignment of each data point in pixels. The assignment should be 1, 2, 3, etc. For k = 5, for example, each cell of class should be either 1, 2, 3, 4, or 5. The output should be a column vector with size (pixels, 1) elements.
- centroid: location of k centroids (or representatives) in your result. With images, each centroid corresponds to the representative color of each cluster. The output should be a matrix with K rows and 3 columns. The range of values should be [0, 255], possibly floating-point numbers.

Note:

Due to randomness, the results may be different in every run.

- **Solution (Part 1):**

1) Use k-means with **squared-`2 norm** as a metric, for GeorgiaTech.bmp and football.bmp and also choose a third picture of your own to work on. We recommend size of 320 _ 240 or smaller.
   Run your k-means implementation with these pictures, with several different k = 2; 4; 8; 16. How long does it take to converge for each k (report the number of iterations, as well as actual running time)? Please write in your report, and also include the resulted compressed pictures for each k.

SOLUTION:

For all the images in this question, we are using RGB (Red, blue, green) color components of the pixels in an image as the feature to perform clustering, and in-turn compressing the image. For the first part of the question, I have used random initialization of cluster centers to start with for KMeans clustering. We are implementing KMeans algorithm using squared L2 norm as distance metric. The centroid adjustments are done by calculating the average of all the data points within the cluster of that iteration.

Code Parameters and usage:

usage: KMeans.py [-h] pixel k method
sample command: python KMeans.py GeorgiaTech.bmp 3 mean

positional arguments:

pixel	Input image to be compressed. Can be football.bmp, GeorgiaTech.bmp, flowers1.jpg. Do not give any path for the file
k	Number of desired clusters
method	L1 norm/ L2 norm clustering. Can take values mean/median

K-means on football.bmp (k = 2,4,8,16)

The image football.bmp original image has the below structure.

```
Shape of the matrix obtained by reading the image
(412, 620, 3)
```

The image has 412*620 = 255440 pixels with 3 components (RGB) representing the color of each pixel.

Implementing k-means algorithm using the squared-l2 norm as the distance metric for the input image football.bmp, below were the results that were obtained.

K-means with k = 2 clusters

When we execute the clustering algorithm, for k = 2 clusters, we would expect to see the 2 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py football.bmp 2 mean

Results of Execution:

Iterations:

Implementation for **k = 2 took 18 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  7045006712.0
Iteration  2 Within Cluster sum of squares =  1284863032.8947613
Iteration  3 Within Cluster sum of squares =  1242193364.8309097
Iteration  4 Within Cluster sum of squares =  1241528388.6520429
Iteration  5 Within Cluster sum of squares =  1241303551.3393075
Iteration  6 Within Cluster sum of squares =  1241224277.6881294
Iteration  7 Within Cluster sum of squares =  1241199958.2404902
Iteration  8 Within Cluster sum of squares =  1241188071.4259043
Iteration  9 Within Cluster sum of squares =  1241178783.9464931
Iteration  10 Within Cluster sum of squares =  1241176125.5317256
Iteration  11 Within Cluster sum of squares =  1241175384.9428434
Iteration  12 Within Cluster sum of squares =  1241175092.293822
Iteration  13 Within Cluster sum of squares =  1241175030.8904538
Iteration  14 Within Cluster sum of squares =  1241175025.7225301
Iteration  15 Within Cluster sum of squares =  1241175023.7280698
Iteration  16 Within Cluster sum of squares =  1241175018.9387138
Iteration  17 Within Cluster sum of squares =  1241175018.4095407
Iteration  18 Within Cluster sum of squares =  1241175017.828861
```

Execution Time:

Time for the algorithm to execute is **3.3572351932525635 secs**

```
Total run time (in seconds):  3.3572351932525635
```

Centroid and Class Labels:

Since we have run the algorithm for k=2 cluster, below are the classes / labels that are generated for the data points. There are 2 classes (1,2). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2}
[2 2 2 ... 2 2 2]

The final cluster centers(Output: centroid) are:
[[189.91441097 182.64826808 172.45584686]
 [ 74.59407487  76.04869875  66.44037077]]
```

Image Before and After Clustering:

Below are the results before and after k = 2 clustering. As we can see, the compressed final image is represented in 2 color categories, mostly black and white.

Original Image

Compressed Image

K-means with k = 4 clusters

When we execute the clustering algorithm, for k = 4 clusters, we would expect to see the 4 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py football.bmp 4 mean

Results of Execution:

Iterations:

Implementation for **k = 4 took 42 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  806227661.0
Iteration  2 Within Cluster sum of squares =  557201587.7594811
Iteration  3 Within Cluster sum of squares =  538086929.3774841
Iteration  4 Within Cluster sum of squares =  523908411.5410171
Iteration  5 Within Cluster sum of squares =  515499600.2774221
Iteration  6 Within Cluster sum of squares =  512083307.90475065
Iteration  7 Within Cluster sum of squares =  510856668.5237409
Iteration  8 Within Cluster sum of squares =  510447296.5080953
Iteration  9 Within Cluster sum of squares =  510303510.4157002
Iteration  10 Within Cluster sum of squares =  510249615.74345195
Iteration  11 Within Cluster sum of squares =  510228783.9632736
Iteration  12 Within Cluster sum of squares =  510220603.5752249
Iteration  13 Within Cluster sum of squares =  510217561.13789463
Iteration  14 Within Cluster sum of squares =  510216253.81493384
Iteration  15 Within Cluster sum of squares =  510215535.6546092
Iteration  16 Within Cluster sum of squares =  510215212.34871125
Iteration  17 Within Cluster sum of squares =  510214751.16501004
Iteration  18 Within Cluster sum of squares =  510213981.012445
Iteration  19 Within Cluster sum of squares =  510213314.4725531
Iteration  20 Within Cluster sum of squares =  510212748.15965164
Iteration  21 Within Cluster sum of squares =  510212436.7210597
Iteration  22 Within Cluster sum of squares =  510212234.05455595
Iteration  23 Within Cluster sum of squares =  510212156.68080986
Iteration  24 Within Cluster sum of squares =  510212013.42015123
Iteration  25 Within Cluster sum of squares =  510211848.50759035
Iteration  26 Within Cluster sum of squares =  510211655.28019804
Iteration  27 Within Cluster sum of squares =  510211465.3566404
Iteration  28 Within Cluster sum of squares =  510211390.4433248
Iteration  29 Within Cluster sum of squares =  510211363.2268644
Iteration  30 Within Cluster sum of squares =  510211357.8018035
Iteration  31 Within Cluster sum of squares =  510211352.42081076
Iteration  32 Within Cluster sum of squares =  510211343.3910273
Iteration  33 Within Cluster sum of squares =  510211331.9110832
Iteration  34 Within Cluster sum of squares =  510211325.1519596
Iteration  35 Within Cluster sum of squares =  510211323.36011434
Iteration  36 Within Cluster sum of squares =  510211310.371336
Iteration  37 Within Cluster sum of squares =  510211300.23759097
Iteration  38 Within Cluster sum of squares =  510211250.07062465
Iteration  39 Within Cluster sum of squares =  510211216.88150114
Iteration  40 Within Cluster sum of squares =  510211161.0810514
Iteration  41 Within Cluster sum of squares =  510211140.5961986
Iteration  42 Within Cluster sum of squares =  510211138.4546041
```

Execution Time:

Time for the algorithm to execute is **9.701082229614258 secs**

```
Total run time (in seconds):  9.701082229614258
```

Centroid and Class Labels:

Since we have run the algorithm for k=4 cluster, below are the classes / labels that are generated for the data points. There are 4 classes (1,2,3,4). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4}
[4 4 4 ... 2 2 2]

The final cluster centers(Output: centroid) are:
[[ 29.42684065  31.12502731  29.74104749]
 [149.64960438 135.97083705 109.83086442]
 [205.07508275 203.37931034 199.14604545]
 [ 72.7184953   83.89400344  84.86657903]]
```

Image Before and After Clustering:

Below are the results before and after k = 4 clustering. As we can see, there is more distinction in terms of color components from the previous k=2 clusters. We can see shades of red color and some variant of green color in the final image.

Original Image

Compressed Image

K-means with k = 8 clusters

When we execute the clustering algorithm, for k = 8 clusters, we would expect to see the 8 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py football.bmp 8 mean

Results of Execution:

Iterations:

Implementation for **k = 8 took 58 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  398343181.0
Iteration  2 Within Cluster sum of squares =  310792286.06362474
Iteration  3 Within Cluster sum of squares =  288385968.61542964
Iteration  4 Within Cluster sum of squares =  277351290.6142659
Iteration  5 Within Cluster sum of squares =  273599305.8201106
Iteration  6 Within Cluster sum of squares =  272280535.1468128
Iteration  7 Within Cluster sum of squares =  271681078.92761284
Iteration  8 Within Cluster sum of squares =  271231951.88331723
Iteration  9 Within Cluster sum of squares =  270635509.7100531
Iteration  10 Within Cluster sum of squares =  269479334.33199024
Iteration  11 Within Cluster sum of squares =  266978884.30506545
Iteration  12 Within Cluster sum of squares =  263182193.1212538
Iteration  13 Within Cluster sum of squares =  259161067.5761551
Iteration  14 Within Cluster sum of squares =  253793298.57604635
Iteration  15 Within Cluster sum of squares =  245127001.20849577
Iteration  16 Within Cluster sum of squares =  237368490.0781354
Iteration  17 Within Cluster sum of squares =  234412001.0806477
Iteration  18 Within Cluster sum of squares =  233569938.9471304
Iteration  19 Within Cluster sum of squares =  233339027.19303498
Iteration  20 Within Cluster sum of squares =  233258758.7260748
Iteration  21 Within Cluster sum of squares =  233219016.1640799
Iteration  22 Within Cluster sum of squares =  233190894.8173495
Iteration  23 Within Cluster sum of squares =  233170399.42778057
Iteration  24 Within Cluster sum of squares =  233154790.73219565
Iteration  25 Within Cluster sum of squares =  233141237.3301206
Iteration  26 Within Cluster sum of squares =  233130020.49526426
Iteration  27 Within Cluster sum of squares =  233122046.41177765
Iteration  28 Within Cluster sum of squares =  233117120.78472298
Iteration  29 Within Cluster sum of squares =  233113283.17729068
Iteration  30 Within Cluster sum of squares =  233110634.95513767
Iteration  31 Within Cluster sum of squares =  233108774.94936404
Iteration  32 Within Cluster sum of squares =  233107499.81757903
Iteration  33 Within Cluster sum of squares =  233106353.16672024
Iteration  34 Within Cluster sum of squares =  233105461.17871296
Iteration  35 Within Cluster sum of squares =  233104546.39870316
Iteration  36 Within Cluster sum of squares =  233101910.05301774
Iteration  37 Within Cluster sum of squares =  233098742.2860729
Iteration  38 Within Cluster sum of squares =  233096801.00137275
Iteration  39 Within Cluster sum of squares =  233095167.06250727
Iteration  40 Within Cluster sum of squares =  233093865.21675512
Iteration  41 Within Cluster sum of squares =  233092763.38601005
Iteration  42 Within Cluster sum of squares =  233091611.77045617
```

```
Iteration  43 Within Cluster sum of squares =  233090503.55829927
Iteration  44 Within Cluster sum of squares =  233089455.89221153
Iteration  45 Within Cluster sum of squares =  233088835.8964866
Iteration  46 Within Cluster sum of squares =  233088431.2819231
Iteration  47 Within Cluster sum of squares =  233088155.30780762
Iteration  48 Within Cluster sum of squares =  233088042.79208404
Iteration  49 Within Cluster sum of squares =  233087961.7429818
Iteration  50 Within Cluster sum of squares =  233087928.43137342
Iteration  51 Within Cluster sum of squares =  233087908.25494167
Iteration  52 Within Cluster sum of squares =  233087897.79738423
Iteration  53 Within Cluster sum of squares =  233087895.14231378
Iteration  54 Within Cluster sum of squares =  233087894.10589835
Iteration  55 Within Cluster sum of squares =  233087890.52344927
Iteration  56 Within Cluster sum of squares =  233087883.42209667
Iteration  57 Within Cluster sum of squares =  233087883.23351824
Iteration  58 Within Cluster sum of squares =  233087883.1349202
```

Execution Time:

Time for the algorithm to execute is **18.828979015350342 secs**

```
Total run time (in seconds):  18.828979015350342
```

Centroid and Class Labels:

Since we have run the algorithm for k=8 cluster, below are the classes / labels that are generated for the data points. There are 8 classes (1,2,3,4,5,6,7,8). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8}
[2 2 2 ... 3 3 3]

The final cluster centers(Output: centroid) are:
[[175.85166293 156.6471282  135.00993631]
 [ 74.93488098  72.55625465  55.21237292]
 [112.68284507 129.59646845 101.44047591]
 [ 24.32026526  25.23880787  24.25547538]
 [192.1925072  185.76384044 175.86173214]
 [ 24.70134499  64.33786147 117.15682582]
 [157.6238293  108.0678394   84.24322013]
 [217.70114943 222.87928189 226.55815409]]
```

Image Before and After Clustering:

Below are the results before and after k = 8 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. We can see shades of blue and more distinct green variants in image.

Original Image

Compressed Image

K-means with k = 16 clusters

When we execute the clustering algorithm, for k = 16 clusters, we would expect to see the 16 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py football.bmp 16 mean

Results of Execution:

Iterations:

Implementation for **k = 16 took 87 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

Execution Time:

Time for the algorithm to execute is **43.696757555007935 secs**

Centroid and Class Labels:

Since we have run the algorithm for k=16 cluster, below are the classes / labels that are generated for the data points. There are 16 classes (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
[14 14 14 ...  7  7  7]

The final cluster centers(Output: centroid) are:
[[208.34128601 213.90442246 217.42527398]
 [217.9217554  200.80268753 130.80966151]
 [ 49.71545798  45.25667395  29.83513614]
 [189.36169557 168.16106171 153.453429  ]
 [185.15226267 188.00283543 189.81365544]
 [130.30723214 142.59625    153.530625  ]
 [120.88729687 141.90673598  90.3370173 ]
 [ 92.00850895 106.12397348 120.0581775 ]
 [ 24.20825228  68.79522883 127.36101157]
 [160.08315075 105.71960669  83.26585796]
 [ 15.66200601  39.14409522  73.37820661]
 [ 16.04938111  16.40487319  16.44525232]
 [175.5899103  144.2752393  117.76629944]
 [ 73.62461957  72.62298913  56.85923913]
 [229.90647105 234.16262772 237.27095887]
 [104.87180506 101.59561647  72.46053503]]
```

Image Before and After Clustering:

Below are the results before and after k = 16 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. This image is very similar to the original image with slight differences in colors.

K-means on GeorgiaTech.bmp (k = 2,4,8,16)

The image GeorgiaTech.bmp original image has the below structure.

```
Shape of the matrix obtained by reading the image
(400, 400, 3)
```

The image has 400*400 = 160000 pixels with 3 components (RGB) representing the color of each pixel.

Implementing k-means algorithm using the squared-l2 norm as the distance metric for the input image GeorgiaTech.bmp, below were the results that were obtained.

K-means with k = 2 clusters

When we execute the clustering algorithm, for k = 2 clusters, we would expect to see the 2 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py GeorgiaTech.bmp 2 mean

Results of Execution:

Iterations:

Implementation for **k = 2 took 12 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  2801768078.0
Iteration  2 Within Cluster sum of squares =  921615166.383718
Iteration  3 Within Cluster sum of squares =  719340849.1505876
Iteration  4 Within Cluster sum of squares =  716308156.8580381
Iteration  5 Within Cluster sum of squares =  716041268.4704661
Iteration  6 Within Cluster sum of squares =  716011312.2857323
Iteration  7 Within Cluster sum of squares =  716007538.5174598
Iteration  8 Within Cluster sum of squares =  716006725.9485544
Iteration  9 Within Cluster sum of squares =  716006607.6069508
Iteration  10 Within Cluster sum of squares =  716006604.3184276
Iteration  11 Within Cluster sum of squares =  716006604.0216565
Iteration  12 Within Cluster sum of squares =  716006603.1182337
```

Execution Time:

Time for the algorithm to execute is **1.6998546123504639 secs**

```
Total run time (in seconds):  1.6998546123504639
```

Centroid and Class Labels:

Since we have run the algorithm for k=2 cluster, below are the classes / labels that are generated for the data points. There are 2 classes (1,2). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2}
[2 2 2 ... 1 1 1]

The final cluster centers(Output: centroid) are:
[[ 60.27504566  56.21564111  45.89597803]
 [172.03466717 161.22584305 153.36686159]]
```

Image Before and After Clustering:

Below are the results before and after k = 2 clustering. As we can see, the compressed final image is represented in 2 color categories, mostly black and white.

Original Image

Compressed Image

K-means with k = 4 clusters

When we execute the clustering algorithm, for k = 4 clusters, we would expect to see the 4 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py GeorgiaTech.bmp 4 mean

Results of Execution:

Iterations:

Implementation for **k = 4 took 23 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  1046058117.0
Iteration  2 Within Cluster sum of squares =  466693819.5062073
Iteration  3 Within Cluster sum of squares =  427864176.44419783
Iteration  4 Within Cluster sum of squares =  416306666.20227957
Iteration  5 Within Cluster sum of squares =  406597632.5353191
Iteration  6 Within Cluster sum of squares =  397644413.37518334
Iteration  7 Within Cluster sum of squares =  390143619.45062214
Iteration  8 Within Cluster sum of squares =  383500160.20376927
Iteration  9 Within Cluster sum of squares =  379640415.1669483
Iteration  10 Within Cluster sum of squares =  377956301.7062986
Iteration  11 Within Cluster sum of squares =  377374549.5594049
Iteration  12 Within Cluster sum of squares =  377183232.2622708
Iteration  13 Within Cluster sum of squares =  377124343.2496418
Iteration  14 Within Cluster sum of squares =  377101815.194256
Iteration  15 Within Cluster sum of squares =  377096528.24240327
Iteration  16 Within Cluster sum of squares =  377094427.5479851
Iteration  17 Within Cluster sum of squares =  377093292.4848576
Iteration  18 Within Cluster sum of squares =  377092976.58556277
Iteration  19 Within Cluster sum of squares =  377092811.742649
Iteration  20 Within Cluster sum of squares =  377092781.0157288
Iteration  21 Within Cluster sum of squares =  377092769.2629793
Iteration  22 Within Cluster sum of squares =  377092764.4393811
Iteration  23 Within Cluster sum of squares =  377092763.4756299
```

Execution Time:

Time for the algorithm to execute is **4.080775499343872 secs**

```
Total run time (in seconds):  4.080775499343872
```

Centroid and Class Labels:

Since we have run the algorithm for k=4 cluster, below are the classes / labels that are generated for the data points. There are 4 classes (1,2,3,4). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

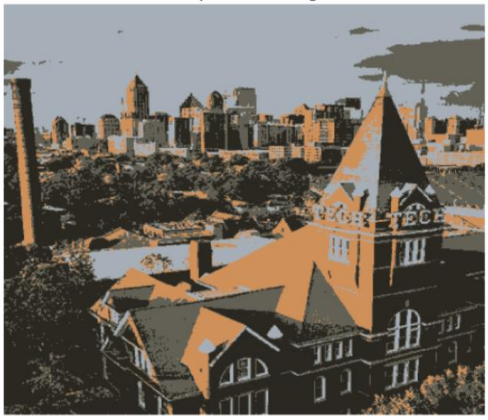
```
The final class labels (Output: class) are  {1, 2, 3, 4}
[1 1 1 ... 4 4 4]

The final cluster centers(Output: centroid) are:
[[219.06576192 193.71971649 175.20372906]
 [120.59033057 148.65117736 170.45716238]
 [164.26524847 117.024551    72.26810811]
 [ 49.51622485  47.40357563  40.51560836]]
```

Image Before and After Clustering:

Below are the results before and after k = 4 clustering. As we can see, there is more distinction in terms of color components from the previous k=2 clusters. We can see shades of red color and some variant of green color in the final image.

Original Image

Compressed Image



K-means with k = 8 clusters

When we execute the clustering algorithm, for k = 8 clusters, we would expect to see the 8 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py GeorgiaTech.bmp 8 mean

Results of Execution:

Iterations:

Implementation for **k = 8 took 87 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  360156076.0
Iteration  2 Within Cluster sum of squares =  312446484.1786216
Iteration  3 Within Cluster sum of squares =  279972992.57198226
Iteration  4 Within Cluster sum of squares =  243371410.9627437
Iteration  5 Within Cluster sum of squares =  216479611.13117573
Iteration  6 Within Cluster sum of squares =  205670069.42984965
Iteration  7 Within Cluster sum of squares =  201611099.81151572
Iteration  8 Within Cluster sum of squares =  199670841.23310283
Iteration  9 Within Cluster sum of squares =  198407190.05927378
Iteration  10 Within Cluster sum of squares =  197320486.80539286
Iteration  11 Within Cluster sum of squares =  196160981.92019475
Iteration  12 Within Cluster sum of squares =  194808771.73867387
Iteration  13 Within Cluster sum of squares =  193361882.6959274
Iteration  14 Within Cluster sum of squares =  192016046.24170476
Iteration  15 Within Cluster sum of squares =  191024677.987106
Iteration  16 Within Cluster sum of squares =  190343161.3379268
Iteration  17 Within Cluster sum of squares =  189803229.2851602
Iteration  18 Within Cluster sum of squares =  189357510.807434
Iteration  19 Within Cluster sum of squares =  189018706.5818736
Iteration  20 Within Cluster sum of squares =  188758178.35543704
Iteration  21 Within Cluster sum of squares =  188552560.91040343
Iteration  22 Within Cluster sum of squares =  188376225.07513955
Iteration  23 Within Cluster sum of squares =  188207314.17887092
Iteration  24 Within Cluster sum of squares =  188026925.67502537
Iteration  25 Within Cluster sum of squares =  187787932.45728806
Iteration  26 Within Cluster sum of squares =  187440591.76898313
Iteration  27 Within Cluster sum of squares =  186855746.15959147
Iteration  28 Within Cluster sum of squares =  185834403.44115937
Iteration  29 Within Cluster sum of squares =  184629609.3676568
Iteration  30 Within Cluster sum of squares =  183516323.73336303
Iteration  31 Within Cluster sum of squares =  182740433.7227237
Iteration  32 Within Cluster sum of squares =  182177662.55876657
Iteration  33 Within Cluster sum of squares =  181742605.05162314
Iteration  34 Within Cluster sum of squares =  181340933.7102582
Iteration  35 Within Cluster sum of squares =  180950401.34878716
Iteration  36 Within Cluster sum of squares =  180580355.5514688
Iteration  37 Within Cluster sum of squares =  180252327.33823308
Iteration  38 Within Cluster sum of squares =  180009223.5222422
Iteration  39 Within Cluster sum of squares =  179804596.07353088
Iteration  40 Within Cluster sum of squares =  179593560.81707123
Iteration  41 Within Cluster sum of squares =  179336274.01495153
Iteration  42 Within Cluster sum of squares =  178998631.76731983
Iteration  43 Within Cluster sum of squares =  178623089.4425571
Iteration  44 Within Cluster sum of squares =  178286604.49583617
Iteration  45 Within Cluster sum of squares =  178010957.21786714
Iteration  46 Within Cluster sum of squares =  177774201.34442264
Iteration  47 Within Cluster sum of squares =  177581482.55134234
Iteration  48 Within Cluster sum of squares =  177393953.87255234
Iteration  49 Within Cluster sum of squares =  177198471.6271456
Iteration  50 Within Cluster sum of squares =  176965715.99623135
Iteration  51 Within Cluster sum of squares =  176651643.48905975
Iteration  52 Within Cluster sum of squares =  176281923.98309556
Iteration  53 Within Cluster sum of squares =  175885714.18893328
Iteration  54 Within Cluster sum of squares =  175443452.60257968
Iteration  55 Within Cluster sum of squares =  174976287.43274003
Iteration  56 Within Cluster sum of squares =  174565517.59687662
Iteration  57 Within Cluster sum of squares =  174221401.4069815
Iteration  58 Within Cluster sum of squares =  173941854.1556172
Iteration  59 Within Cluster sum of squares =  173734329.21141732
Iteration  60 Within Cluster sum of squares =  173602822.66891468
Iteration  61 Within Cluster sum of squares =  173519672.6292914
Iteration  62 Within Cluster sum of squares =  173465034.1829063
Iteration  63 Within Cluster sum of squares =  173431318.26894757
Iteration  64 Within Cluster sum of squares =  173411791.86449626
Iteration  65 Within Cluster sum of squares =  173400864.01860988
Iteration  66 Within Cluster sum of squares =  173392318.25258884
Iteration  67 Within Cluster sum of squares =  173385877.01999736
Iteration  68 Within Cluster sum of squares =  173381412.38666654
Iteration  69 Within Cluster sum of squares =  173377508.7455848
Iteration  70 Within Cluster sum of squares =  173374823.01325685
Iteration  71 Within Cluster sum of squares =  173372995.38760355
Iteration  72 Within Cluster sum of squares =  173372107.17204207
Iteration  73 Within Cluster sum of squares =  173371258.17690876
Iteration  74 Within Cluster sum of squares =  173370775.0345644
Iteration  75 Within Cluster sum of squares =  173370430.21605754
Iteration  76 Within Cluster sum of squares =  173370106.19872594
Iteration  77 Within Cluster sum of squares =  173369853.91634348
Iteration  78 Within Cluster sum of squares =  173369691.85262957
Iteration  79 Within Cluster sum of squares =  173369544.62132546
Iteration  80 Within Cluster sum of squares =  173369324.55246148
Iteration  81 Within Cluster sum of squares =  173369231.80792072
Iteration  82 Within Cluster sum of squares =  173369206.105059
Iteration  83 Within Cluster sum of squares =  173369153.850646
Iteration  84 Within Cluster sum of squares =  173369129.7726076
Iteration  85 Within Cluster sum of squares =  173369101.70054042
Iteration  86 Within Cluster sum of squares =  173369098.8960586
Iteration  87 Within Cluster sum of squares =  173369098.56808522
```

Execution Time:

Time for the algorithm to execute is **19.67169189453125 secs**

```
Total run time (in seconds):  19.67169189453125
```

Centroid and Class Labels:

Since we have run the algorithm for k=8 cluster, below are the classes / labels that are generated for the data points. There are 8 classes (1,2,3,4,5,6,7,8). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8}
[3 3 3 ... 7 6 6]

The final cluster centers(Output: centroid) are:
[[200.99468085 176.87472138 162.41013171]
 [ 58.93183182  78.63985385  90.72852702]
 [250.39753621 241.52771766 232.10687531]
 [133.43882238 107.1877088   78.52401169]
 [217.79942389 140.21080958  79.805943  ]
 [ 37.28699541  32.5428428   28.55906716]
 [ 90.24118368  74.15247244  32.64006189]
 [122.07456412 152.29040039 175.67586658]]
```

Image Before and After Clustering:

Below are the results before and after k = 8 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. We can see shades of more distinct green and red variants in image.

Original Image

Compressed Image

K-means with k = 16 clusters

When we execute the clustering algorithm, for k = 16 clusters, we would expect to see the 16 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py GeorgiaTech.bmp 16 mean

Results of Execution:

Iterations:

Implementation for **k = 16 took 80 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  174634445.0
Iteration  2 Within Cluster sum of squares =  128686558.57932968
Iteration  3 Within Cluster sum of squares =  117301571.90766364
Iteration  4 Within Cluster sum of squares =  111881332.53906164
Iteration  5 Within Cluster sum of squares =  108782901.9657307
Iteration  6 Within Cluster sum of squares =  106581705.25545375
Iteration  7 Within Cluster sum of squares =  104694581.72893669
Iteration  8 Within Cluster sum of squares =  103132598.71064164
Iteration  9 Within Cluster sum of squares =  101868258.81127961
Iteration  10 Within Cluster sum of squares =  100649018.18842553
Iteration  11 Within Cluster sum of squares =  99517581.2387387
Iteration  12 Within Cluster sum of squares =  98645553.65331
Iteration  13 Within Cluster sum of squares =  97868896.10294786
Iteration  14 Within Cluster sum of squares =  97110599.07981059
Iteration  15 Within Cluster sum of squares =  96347078.87536529
Iteration  16 Within Cluster sum of squares =  95576547.6865732
Iteration  17 Within Cluster sum of squares =  94806224.50272219
Iteration  18 Within Cluster sum of squares =  94008667.53178625
Iteration  19 Within Cluster sum of squares =  93247006.72397165
Iteration  20 Within Cluster sum of squares =  92496829.69012906
Iteration  21 Within Cluster sum of squares =  91860150.63869658
Iteration  22 Within Cluster sum of squares =  91352432.85440013
Iteration  23 Within Cluster sum of squares =  90935466.49060243
Iteration  24 Within Cluster sum of squares =  90565512.65886804
Iteration  25 Within Cluster sum of squares =  90207678.37804179
Iteration  26 Within Cluster sum of squares =  89801259.1725466
Iteration  27 Within Cluster sum of squares =  89389934.32582287
Iteration  28 Within Cluster sum of squares =  89106065.7007307
Iteration  29 Within Cluster sum of squares =  88921555.22998855
Iteration  30 Within Cluster sum of squares =  88756080.96831942
Iteration  31 Within Cluster sum of squares =  88588389.76588194
Iteration  32 Within Cluster sum of squares =  88430909.15859184
Iteration  33 Within Cluster sum of squares =  88284875.31647015
Iteration  34 Within Cluster sum of squares =  88111195.7768983
Iteration  35 Within Cluster sum of squares =  87894687.6956706
Iteration  36 Within Cluster sum of squares =  87621596.17227507
Iteration  37 Within Cluster sum of squares =  87282898.32667126
Iteration  38 Within Cluster sum of squares =  86908256.48429403
Iteration  39 Within Cluster sum of squares =  86444091.5078275
Iteration  40 Within Cluster sum of squares =  85848651.20127021
Iteration  41 Within Cluster sum of squares =  85318267.58125351
Iteration  42 Within Cluster sum of squares =  84969442.2151114
Iteration  43 Within Cluster sum of squares =  84725731.62337142
Iteration  44 Within Cluster sum of squares =  84540837.44048816
Iteration  45 Within Cluster sum of squares =  84351853.24487808
Iteration  46 Within Cluster sum of squares =  84169234.29347962
Iteration  47 Within Cluster sum of squares =  83988100.39555572
Iteration  48 Within Cluster sum of squares =  83807510.02695478
Iteration  49 Within Cluster sum of squares =  83651169.14723997
Iteration  50 Within Cluster sum of squares =  83543766.23934615
Iteration  51 Within Cluster sum of squares =  83478428.8544522
Iteration  52 Within Cluster sum of squares =  83435179.20322055
Iteration  53 Within Cluster sum of squares =  83411826.917367
Iteration  54 Within Cluster sum of squares =  83398846.27832022
Iteration  55 Within Cluster sum of squares =  83390420.31192547
Iteration  56 Within Cluster sum of squares =  83384192.27168842
Iteration  57 Within Cluster sum of squares =  83380285.00850104
Iteration  58 Within Cluster sum of squares =  83377683.16070409
Iteration  59 Within Cluster sum of squares =  83375463.253391
Iteration  60 Within Cluster sum of squares =  83373603.91337426
Iteration  61 Within Cluster sum of squares =  83372931.10440856
Iteration  62 Within Cluster sum of squares =  83372098.45530069
Iteration  63 Within Cluster sum of squares =  83370992.8237646
Iteration  64 Within Cluster sum of squares =  83370481.10130256
Iteration  65 Within Cluster sum of squares =  83370156.51258437
Iteration  66 Within Cluster sum of squares =  83370059.46045753
Iteration  67 Within Cluster sum of squares =  83369967.11351992
Iteration  68 Within Cluster sum of squares =  83369909.98810251
Iteration  69 Within Cluster sum of squares =  83369892.54194404
Iteration  70 Within Cluster sum of squares =  83369878.5377807
Iteration  71 Within Cluster sum of squares =  83369869.0892925
Iteration  72 Within Cluster sum of squares =  83369856.73957404
Iteration  73 Within Cluster sum of squares =  83369846.67078449
Iteration  74 Within Cluster sum of squares =  83369841.0089766
Iteration  75 Within Cluster sum of squares =  83369837.64291373
Iteration  76 Within Cluster sum of squares =  83369833.01831041
Iteration  77 Within Cluster sum of squares =  83369825.20774734
Iteration  78 Within Cluster sum of squares =  83369820.14250037
Iteration  79 Within Cluster sum of squares =  83369818.12384053
Iteration  80 Within Cluster sum of squares =  83369817.98987743
```

Execution Time:

Time for the algorithm to execute is **33.376046657562256 secs**

```
Total run time (in seconds):  33.376046657562256
```

Centroid and Class Labels:

Since we have run the algorithm for k=16 cluster, below are the classes / labels that are generated for the data points. There are 16 classes (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
[11 11 11 ...  4  1  1]

The final cluster centers(Output: centroid) are:
[[ 49.17005436  37.44238951  28.75933585]
 [ 33.92764738  56.95823323  69.57355387]
 [163.57070423 132.08037559 106.66065728]
 [ 76.03131177  63.57151774  31.08301259]
 [111.94403092  90.52791297  33.23804752]
 [ 27.88716876  24.6404463   22.16118085]
 [212.48603053 161.16995448 114.14888869]
 [223.67542666 195.32582976 176.51879384]
 [159.90572917 115.21067708  42.62838542]
 [116.26463878  98.38922687  89.05741445]
 [253.78016838 251.5191768  246.86973807]
 [159.77987724 165.34233963 177.97665182]
 [239.0180472  119.20800555  45.53632578]
 [ 74.08579088  77.92675603  83.17394102]
 [ 80.47078576 119.93673606 145.1003358 ]
 [113.22703833 152.99087108 180.41414634]]
```

Image Before and After Clustering:

Below are the results before and after k = 16 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. This image is very similar to the original image with slight differences in colors.

Original Image

Compressed Image

K-means on a third image (k = 2,4,8,16)

The image football.bmp original image has the below structure.

```
Shape of the matrix obtained by reading the image
(300, 200, 3)
```

The image has 300*300 = 60000 pixels with 3 components (RGB) representing the color of each pixel.

Implementing k-means algorithm using the squared-l2 norm as the distance metric for the input image football.bmp, below were the results that were obtained.

K-means with k = 2 clusters

When we execute the clustering algorithm, for k = 2 clusters, we would expect to see the 2 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py flowers1.jpg 2 mean

Results of Execution:

Iterations:

Implementation for **k = 2 took 13 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  819686346.0
Iteration  2 Within Cluster sum of squares =  421951351.4550818
Iteration  3 Within Cluster sum of squares =  414343278.2524041
Iteration  4 Within Cluster sum of squares =  413162202.1697939
Iteration  5 Within Cluster sum of squares =  412780345.19205
Iteration  6 Within Cluster sum of squares =  412679006.99776024
Iteration  7 Within Cluster sum of squares =  412642164.03480095
Iteration  8 Within Cluster sum of squares =  412627958.6451678
Iteration  9 Within Cluster sum of squares =  412623945.1818459
Iteration  10 Within Cluster sum of squares =  412622769.96440977
Iteration  11 Within Cluster sum of squares =  412622392.2166256
Iteration  12 Within Cluster sum of squares =  412622304.85931987
Iteration  13 Within Cluster sum of squares =  412622290.85191417
```

Execution Time:

Time for the algorithm to execute is **0.5743868350982666 secs**

```
Total run time (in seconds):  0.5743868350982666
```

Centroid and Class Labels:

Since we have run the algorithm for k=2 cluster, below are the classes / labels that are generated for the data points. There are 2 classes (1,2). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2}
[2 2 2 ... 1 1 1]

The final cluster centers(Output: centroid) are:
[[145.67978073  76.36909724  81.88791184]
 [167.98197096 189.25461906 231.72358604]]
```

Image Before and After Clustering:

Below are the results before and after k = 2 clustering. As we can see, the compressed final image is represented in 2 color categories, mostly blue and dark pink.

Original Image

Compressed Image

K-means with k = 4 clusters

When we execute the clustering algorithm, for k = 4 clusters, we would expect to see the 4 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:
python KMeans.py flowers1.jpg 4 mean

Results of Execution:

Iterations:
Implementation for **k = 4 took 26 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  676158193.0
Iteration  2 Within Cluster sum of squares =  288358443.91163045
Iteration  3 Within Cluster sum of squares =  272015108.14677346
Iteration  4 Within Cluster sum of squares =  257534956.8577979
Iteration  5 Within Cluster sum of squares =  257158970.94274575
Iteration  6 Within Cluster sum of squares =  256933927.23132914
Iteration  7 Within Cluster sum of squares =  256725745.795181
Iteration  8 Within Cluster sum of squares =  256505414.44339517
Iteration  9 Within Cluster sum of squares =  256257805.21846184
Iteration  10 Within Cluster sum of squares =  255867166.3514034
Iteration  11 Within Cluster sum of squares =  255062160.68337524
Iteration  12 Within Cluster sum of squares =  251973418.38903442
Iteration  13 Within Cluster sum of squares =  234463042.89243203
Iteration  14 Within Cluster sum of squares =  199529380.22129753
Iteration  15 Within Cluster sum of squares =  180109839.33679858
Iteration  16 Within Cluster sum of squares =  174063590.92543143
Iteration  17 Within Cluster sum of squares =  172567208.9867112
Iteration  18 Within Cluster sum of squares =  172249997.5600715
Iteration  19 Within Cluster sum of squares =  172189257.51783478
Iteration  20 Within Cluster sum of squares =  172175810.92881924
Iteration  21 Within Cluster sum of squares =  172172510.6035791
Iteration  22 Within Cluster sum of squares =  172171552.2010221
Iteration  23 Within Cluster sum of squares =  172171325.91411567
Iteration  24 Within Cluster sum of squares =  172171281.03185916
Iteration  25 Within Cluster sum of squares =  172171276.45571125
Iteration  26 Within Cluster sum of squares =  172171275.10767788
```

Execution Time:
Time for the algorithm to execute is **1.0299651622772217 secs**

```
Total run time (in seconds):  1.0299651622772217
```

Centroid and Class Labels:
Since we have run the algorithm for k=4 cluster, below are the classes / labels that are generated for the data points. There are 4 classes (1,2,3,4). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4}
[3 1 1 ... 2 2 2]

The final cluster centers(Output: centroid) are:
[[ 96.52300384 179.05622438 237.85937398]
 [113.28505629  84.7813886   47.19199932]
 [204.15026441 210.24351864 232.32963584]
 [220.5788601   76.96632124 172.75896373]]
```

Image Before and After Clustering:
Below are the results before and after k = 4 clustering. As we can see, there is more distinction in terms of color components from the previous k=2 clusters. We can see shades of white, pink and dark pink now.

K-means with k = 8 clusters

When we execute the clustering algorithm, for k = 8 clusters, we would expect to see the 8 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py flowers1.jpg 8 mean

Results of Execution:

Iterations:

Implementation for **k = 8 took 47 iterations to converge**. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  255320080.0
Iteration  2 Within Cluster sum of squares =  146979024.9594035
Iteration  3 Within Cluster sum of squares =  106321355.72656971
Iteration  4 Within Cluster sum of squares =  99086356.58157729
Iteration  5 Within Cluster sum of squares =  97345794.18652996
Iteration  6 Within Cluster sum of squares =  96635328.8915899
Iteration  7 Within Cluster sum of squares =  96076445.89546233
Iteration  8 Within Cluster sum of squares =  95470293.94698471
Iteration  9 Within Cluster sum of squares =  94408353.72975475
Iteration  10 Within Cluster sum of squares =  91638040.93657345
Iteration  11 Within Cluster sum of squares =  88557769.8718238
Iteration  12 Within Cluster sum of squares =  86876038.19991803
Iteration  13 Within Cluster sum of squares =  86031542.6504666
Iteration  14 Within Cluster sum of squares =  85595345.4859736
Iteration  15 Within Cluster sum of squares =  85286667.30740106
Iteration  16 Within Cluster sum of squares =  85041790.5901266
Iteration  17 Within Cluster sum of squares =  84861978.4764564
Iteration  18 Within Cluster sum of squares =  84715128.4197408
Iteration  19 Within Cluster sum of squares =  84595498.74241997
Iteration  20 Within Cluster sum of squares =  84491461.66444193
Iteration  21 Within Cluster sum of squares =  84405655.80047558
Iteration  22 Within Cluster sum of squares =  84341250.50630738
Iteration  23 Within Cluster sum of squares =  84291532.38994052
Iteration  24 Within Cluster sum of squares =  84256516.65544023
Iteration  25 Within Cluster sum of squares =  84223461.16705328
Iteration  26 Within Cluster sum of squares =  84197369.64190972
Iteration  27 Within Cluster sum of squares =  84179462.75377959
Iteration  28 Within Cluster sum of squares =  84163088.75697537
Iteration  29 Within Cluster sum of squares =  84151674.63764638
Iteration  30 Within Cluster sum of squares =  84144677.34067972
Iteration  31 Within Cluster sum of squares =  84141652.64781147
Iteration  32 Within Cluster sum of squares =  84138966.19979313
Iteration  33 Within Cluster sum of squares =  84131152.58089449
Iteration  34 Within Cluster sum of squares =  84121048.85334957
Iteration  35 Within Cluster sum of squares =  84117104.90368707
Iteration  36 Within Cluster sum of squares =  84114668.99163713
Iteration  37 Within Cluster sum of squares =  84113457.12141973
Iteration  38 Within Cluster sum of squares =  84112374.29854335
Iteration  39 Within Cluster sum of squares =  84110697.98097095
Iteration  40 Within Cluster sum of squares =  84108633.22749394
Iteration  41 Within Cluster sum of squares =  84107040.33748597
Iteration  42 Within Cluster sum of squares =  84106626.90418194
Iteration  43 Within Cluster sum of squares =  84105613.5471088
Iteration  44 Within Cluster sum of squares =  84105457.18902119
Iteration  45 Within Cluster sum of squares =  84105343.43907143
Iteration  46 Within Cluster sum of squares =  84105331.17116392
Iteration  47 Within Cluster sum of squares =  84105326.26670966
```

Execution Time:

Time for the algorithm to execute is **2.7002718448638916 secs**

```
Total run time (in seconds):  2.7002718448638916
```

Centroid and Class Labels:

Since we have run the algorithm for k=8 cluster, below are the classes / labels that are generated for the data points. There are 8 classes (1,2,3,4,5,6,7,8). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

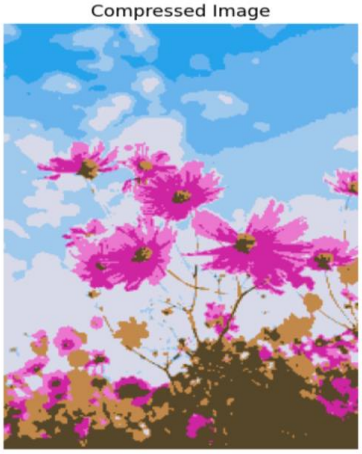
```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8}
[8 8 8 ... 4 4 4]

The final cluster centers(Output: centroid) are:
[[235.88698899 119.33687107 207.60416667]
 [207.58831602  37.4750049  162.05626348]
 [ 38.86007106 163.72642799 235.40694179]
 [ 86.20021262  71.84384597  42.80510276]
 [195.22766571 137.5389049   74.30747063]
 [106.08279824 180.71623296 238.46062859]
 [217.16527296 218.84703243 233.3438711 ]
 [160.66597143 201.16644071 238.64751278]]
```

Image Before and After Clustering:

Below are the results before and after k = 8 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. We can see shades more distinction of while and the flower colors.

Original Image

Compressed Image



K-means with k = 16 clusters

When we execute the clustering algorithm, for k = 16 clusters, we would expect to see the 16 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py flowers1.jpg 16 mean

Results of Execution:

Iterations:

Implementation for **k = 16 took 56 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  127296372.0
Iteration  2 Within Cluster sum of squares =  70568052.29312098
Iteration  3 Within Cluster sum of squares =  62531622.68524346
Iteration  4 Within Cluster sum of squares =  57142995.08096236
Iteration  5 Within Cluster sum of squares =  53831883.49315212
Iteration  6 Within Cluster sum of squares =  51731293.47706267
Iteration  7 Within Cluster sum of squares =  50114272.003517725
Iteration  8 Within Cluster sum of squares =  48746318.12627953
Iteration  9 Within Cluster sum of squares =  47592781.715694785
Iteration  10 Within Cluster sum of squares =  46432986.48406422
Iteration  11 Within Cluster sum of squares =  45276676.64085573
Iteration  12 Within Cluster sum of squares =  44211593.071958765
Iteration  13 Within Cluster sum of squares =  43368965.754614495
Iteration  14 Within Cluster sum of squares =  42684851.744121894
Iteration  15 Within Cluster sum of squares =  42116598.921331696
Iteration  16 Within Cluster sum of squares =  41629654.77445796
Iteration  17 Within Cluster sum of squares =  41269685.53473812
Iteration  18 Within Cluster sum of squares =  40996154.8527938
Iteration  19 Within Cluster sum of squares =  40791104.44959717
Iteration  20 Within Cluster sum of squares =  40657595.946486525
Iteration  21 Within Cluster sum of squares =  40561246.66105212
Iteration  22 Within Cluster sum of squares =  40498310.5891997
Iteration  23 Within Cluster sum of squares =  40458942.97948964
Iteration  24 Within Cluster sum of squares =  40425715.000270694
Iteration  25 Within Cluster sum of squares =  40401243.606996
Iteration  26 Within Cluster sum of squares =  40385451.79284613
Iteration  27 Within Cluster sum of squares =  40370879.01864892
Iteration  28 Within Cluster sum of squares =  40356310.84541128
Iteration  29 Within Cluster sum of squares =  40341220.201973796
Iteration  30 Within Cluster sum of squares =  40326391.1492269
Iteration  31 Within Cluster sum of squares =  40314057.004717246
Iteration  32 Within Cluster sum of squares =  40302927.802189104
Iteration  33 Within Cluster sum of squares =  40293733.3492066
Iteration  34 Within Cluster sum of squares =  40288401.538143136
Iteration  35 Within Cluster sum of squares =  40283806.6032133
Iteration  36 Within Cluster sum of squares =  40278522.45249644
Iteration  37 Within Cluster sum of squares =  40273809.257687755
Iteration  38 Within Cluster sum of squares =  40270571.7225855
Iteration  39 Within Cluster sum of squares =  40267510.366402134
Iteration  40 Within Cluster sum of squares =  40264339.27795547
Iteration  41 Within Cluster sum of squares =  40262126.255629286
Iteration  42 Within Cluster sum of squares =  40260166.10095089
Iteration  43 Within Cluster sum of squares =  40258830.60365461
Iteration  44 Within Cluster sum of squares =  40257615.231979504
Iteration  45 Within Cluster sum of squares =  40256554.334515944
Iteration  46 Within Cluster sum of squares =  40255972.11678409
Iteration  47 Within Cluster sum of squares =  40255667.331541225
Iteration  48 Within Cluster sum of squares =  40255533.85653571
Iteration  49 Within Cluster sum of squares =  40255458.77637509
Iteration  50 Within Cluster sum of squares =  40255420.764847934
Iteration  51 Within Cluster sum of squares =  40255401.659162015
Iteration  52 Within Cluster sum of squares =  40255396.271288894
Iteration  53 Within Cluster sum of squares =  40255386.975879505
Iteration  54 Within Cluster sum of squares =  40255370.49669532
Iteration  55 Within Cluster sum of squares =  40255354.046945766
Iteration  56 Within Cluster sum of squares =  40255343.23233042
```

Execution Time:

Time for the algorithm to execute is **8.793003797531128 secs**

```
Total run time (in seconds):  8.793003797531128
```

Centroid and Class Labels:

Since we have run the algorithm for k=16 cluster, below are the classes / labels that are generated for the data points. There are 16 classes (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
[ 7  7 11 ...  4  4  4]

The final cluster centers(Output: centroid) are:
[[145.58941799  35.78888889  96.79259259]
 [ 83.02352941 173.27656396 237.40690943]
 [243.85098522 175.         230.09482759]
 [ 64.63767222  64.9868394   31.73596545]
 [224.20812111 227.08121109 235.6598755 ]
 [178.51598402 122.17282717 112.31968032]
 [162.68941482 202.97568488 241.40217215]
 [198.63903334 217.70097889 239.00137657]
 [215.93341656  22.05617978 163.07407407]
 [231.33973129 143.15547025  32.27255278]
 [125.52147779 187.80633338 239.60196452]
 [229.4994467   65.10106972 197.71523423]
 [207.95059952 183.37985612 159.12853717]
 [ 24.21511378 161.25676256 234.98969515]
 [236.06970954 117.44896266 210.33526971]
 [118.58117367 101.31784331  50.43699732]]
```

Image Before and After Clustering:

Below are the results before and after k = 16 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. This image is very similar to the original image with slight differences in colors.

➢ Solution (Part 2):

1. (20 points) Run your k-means implementation (with squared-`2 norm) with different initialization centroids. Please test two initialization strategies, compare the results (output image, running time, iterations) and report: (i) random initialization. Please try multiple time and report the best one (in terms of the image quality). (ii) poor initialization. Please design your own strategy, explain why it qualities as a poor initialization, try multiple times, and report the results.
How does this affect your final result? (We usually randomize initial location of centroids in general.)
Please also explain in the report how you initialize the centroid.

In my original method (KMeans algorithm), I used random initialization to defined the first centroids for the KMeans algorithm. In this, I would randomly choose a data point that lies within the range of data points in the input image and initialize centroids depending on the value of k provided.
On analyzing, one poor initialization method is randomly assigning centroids manually with some components of the data points that are outside the range of the input data dimensions. This results in incorrect clustering and the resulting image does not have the expected outcome. For this question, I am comparing random initialization within the image range and random initialization outside the image data range as two methods to compare the results.

I am comparing the results for the two initialization methods for the same images that I worked on in Problem Part1.

Poor Initialization Vs Random Initialization (GeorgiaTech.bmp)

(i) Poor Initialization (k =10)

For this, I am running K-Means for k =10 clusters on GeorgiaTech.bmp. For initialization, I have run the below code.

a) Open the code “KMeans.py”, and modify the below line(Line 200) to manually initialize the centroids to below.

Current Code:

```
#Implementing KMeans clustering on Input Image
Class, centroid = kMeans(img_reshaped, k, method=method, first_center=None)
```

Modify to:

Class, centroid = kMeans(img_reshaped, k, method=method, first_center=np.array([[255., 255., 255.],[65., 67., 27.],[100., 200., 220.],[46., 162., 109.],[10., 0., 25.],[33., 44., 55.],[99., 100., 123.],[105., 110., 150.],[1909., 5099., 8932.],[9900., 9900., 9900.]]))

In the above initialization, there are 10 centroids given. Among these, the last 2 centroids [1909., 5099., 8932.],[9900., 9900., 9900.] are points outside the RGB range which is within 0 and 255.

b) Run the code using the below command after modifying
python KMeans.py GeorgiaTech.bmp 10 mean

Expected Output:
We would expect to see 10 clusters after this.

Actual Results:

```
Total run time (in seconds):  15.537895679473877

Warning: There are empty clusters, try giving a smaller value of k
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8}
[1 1 1 ... 5 5 5]

The final cluster centers(Output: centroid) are:
[[250.96979807 243.19139596 234.51150132]
 [114.08747133  90.55085605  47.05397621]
 [130.67465623 157.60407962 179.68734331]
 [ 93.23621103 113.61046163 127.85139388]
 [ 39.39436554  32.75744404  25.56786221]
 [ 52.82632013  64.99571988  71.47658828]
 [208.51782191 136.87800868  80.3978035 ]
 [206.76482181 179.08550939 160.86051349]]
```

We can see from the results that only 8 clusters are formed, and the remaining two clusters are empty. As a result, below is the resulting image. With 10 clusters, and good initialization, we might expect a better image with more distinction of colors, however this is not the case.

```
Images of the original and compressed images are shown below
```

Original Image

Compressed Image

(ii) Random Initialization (k = 10)
Below are the results of random initialization, with centroids within the range of input data.

Execute the below command without any modification to the code
python KMeans.py GeorgiaTech.bmp 10 mean

```
Total run time (in seconds):  24.05726647377014

The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
[2 2 2 ... 9 6 6]

The final cluster centers(Output: centroid) are:
[[122.31699118 102.59553159  85.80634543]
 [253.26594595 249.50097297 243.46789189]
 [ 50.69693549  66.07748744  75.67430442]
 [ 86.9476374  131.36394102 159.03610925]
 [214.37057713 188.66755779 173.4687844 ]
 [ 37.80182905  31.55841579  25.99450802]
 [223.91596057 118.65363998  44.04789901]
 [204.46098277 155.73647378 111.34584995]
 [ 96.89356924  79.2008166   31.16536237]
 [141.41982729 160.39260921 179.39201659]]
```

Original Image

Compressed Image

Above are the results and images after running the KMeans clustering. We can see that 10 clusters are created and image is represented as expected.

Poor Initialization Vs Random Initialization (football.bmp)

(i) Poor Initialization (k =5)

For this, I am running K-Means for k =5 clusters on football.bmp. For initialization, I have run the below code.

a) Open the code “KMeans.py”, and modify the below line(Line 200) to manually initialize the centroids to below.

Current Code:

#Implementing KMeans clustering on Input Image

Class, centroid = kMeans(img_reshaped, k, method=method, first_center=None)

Modify to:

Class, centroid = kMeans(img_reshaped, k, method=method, first_center=np.array([[255., 255., 255.],[65., 67., 27.],[1000., 2000., 2200.],[4600., 1620., 1090.],[1022., 0., 2512.]]))

In the above initialization, there are 5 centroids given. Among these, the last 3 centroids [1000., 2000., 2200.],[4600., 1620., 1090.],[1022., 0., 2512.] are points outside the RGB range which is within 0 and 255.

b) Run the code using the below command after modifying
python KMeans.py football.bmp 5 mean

Expected Output:
We would expect to see 5 clusters after this.

Actual Results:
We can see from the results that only 2 clusters are formed, and the remaining 3 clusters are empty. As a result, below is the resulting image. With 5 clusters, and good initialization, we can expect a better image with more distinction of colors, however this is not the case.

```
Total run time (in seconds):  1.2700440883636475

Warning: There are empty clusters, try giving a smaller value of k
The final class labels (Output: class) are  {1, 2}
[2 2 2 ... 2 2 2]

The final cluster centers(Output: centroid) are:
[[189.91441097 182.64826808 172.45584686]
 [ 74.59407487  76.04869875  66.44037077]]
```

(ii) Random Initialization (k = 5)
Below are the results of random initialization, with centroids within the range of input data.

Execute the below command without any modification to the code
python KMeans.py football.bmp 5 mean


```
Total run time (in seconds):  3.6025595664978027

The final class labels (Output: class) are  {1, 2, 3, 4, 5}
[3 3 3 ... 1 1 1]

The final cluster centers(Output: centroid) are:
[[132.30906889 123.07173898  95.11312959]
 [213.40453832 218.33917992 221.68489333]
 [ 59.1690565   73.45339834  82.8659067 ]
 [184.31018342 169.637439   152.17032239]
 [ 28.60494856  28.9483428   26.02401895]]
```

Above are the results and images after running the KMeans clustering. We can see that 5 clusters are created and image is represented as expected.

Poor Initialization Vs Random Initialization (football.bmp) Part 2

(i) Poor Initialization (k = 4)

For this, I am running K-Means for k =4 clusters on football.bmp. For initialization, I have run the below code.

a) Open the code "KMeans.py", and modify the below line(Line 200) to manually initialize the centroids to below.

Current Code:
#Implementing KMeans clustering on Input Image
Class, centroid = kMeans(img_reshaped, k, method=method, first_center=None)

Modify to:
Class, centroid = kMeans(img_reshaped, k, method=method, first_center=np.array([[255., 255., 255.],[6533., 6733., 2733.],[1000., 2000., 2200.],[4600., 1620., 1090.]]))

In the above initialization, there are 5 centroids given. Among these, the last 3 centroids .],[6533., 6733., 2733.],[1000., 2000., 2200.],[4600., 1620., 1090.] are points outside the RGB range which is within 0 and 255.

b) Run the code using the below command after modifying
python KMeans.py football.bmp 4 mean

Expected Output:

We would expect to see 4 clusters after this.

Actual Results:

We can see from the results that only 1 cluster are formed, and the remaining 3 clusters are empty. This is because only one cluster center is within the range of the input data image. As a result, below is the resulting image. With 4 clusters, and good initialization, we can expect a better image with more distinction of colors, however this is not the case.

```
Total run time (in seconds):  0.11384963989257812

Warning: There are empty clusters, try giving a smaller value of k
The final class labels (Output: class) are  {1}
[1 1 1 ... 1 1 1]

The final cluster centers(Output: centroid) are:
[[126.59747886 124.11949577 114.24777247]]
```

(ii) Random Initialization(k=4)

Below are the results of random initialization, with centroids within the range of input data.

Execute the below command without any modification to the code
python KMeans.py football.bmp 4 mean

Below are the results and images after running the KMeans clustering. We can see that 4 clusters are created and image is represented as expected

```
Total run time (in seconds):  3.050076484680176

The final class labels (Output: class) are  {1, 2, 3, 4}
[1 1 4 ... 1 1 1]

The final cluster centers(Output: centroid) are:
[[121.11760406 115.964445    93.0774777 ]
 [212.76599581 217.42816254 220.33402448]
 [182.28367015 166.69982352 148.67129213]
 [ 35.63656913  42.20839938  45.50039282]]
```

Solution (Part 3):

3) (20 points) Now try your k-means with the Manhattan distance (or `1 distance) and repeat the same steps in Part (1). Please note that the assignment of data point should be based on the Manhattan distance, and the cluster centroid (by minimizing the sum of deviance (as a result of using the Manhattan distance) will be taken as the \median" of each cluster. Comment on the difference of image compression results using the two methods.

SOLUTION:

For all the images in this question, we are using RGB (Red, blue, green) color components of the pixels in an image as the feature to perform clustering, and in-turn compressing the image. For the first part of the question, I have used random initialization of cluster centers to start with for KMeans clustering. We are implementing KMeans algorithm using L1 norm(Manhattan distance) as distance metric. The centroid adjustments are done by calculating the median of all the data points within the cluster of that iteration.

Code Parameters and usage:

usage: KMeans.py [-h] pixel k method
sample command: python KMeans.py GeorgiaTech.bmp 3 median

positional arguments:
pixel Input image to be compressed. Can be football.bmp, GeorgiaTech.bmp, flowers1.jpg. Do not give any path for the file

K-means on football.bmp (k = 2,4,8,16)

The image football.bmp original image has the below structure.

```
Shape of the matrix obtained by reading the image
(412, 620, 3)
```

The image has 412*620 = 255440 pixels with 3 components (RGB) representing the color of each pixel.

Implementing k-means algorithm using the L1 norm(Manhattan distance) as the distance metric for the input image football.bmp, below were the results that were obtained.

K-means with k = 2 clusters

When we execute the clustering algorithm, for k = 2 clusters, we would expect to see the 2 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:
python KMeans.py football.bmp 2 median

Results of Execution:

Iterations:
Implementation for **k = 2 took 14 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares to identify how closely the cluster is formed. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  49788171.0
Iteration  2 Within Cluster sum of squares =  26056627.0
Iteration  3 Within Cluster sum of squares =  25859435.0
Iteration  4 Within Cluster sum of squares =  25805413.0
Iteration  5 Within Cluster sum of squares =  25752379.0
Iteration  6 Within Cluster sum of squares =  25716281.0
Iteration  7 Within Cluster sum of squares =  25687514.0
Iteration  8 Within Cluster sum of squares =  25666814.0
Iteration  9 Within Cluster sum of squares =  25654147.0
Iteration  10 Within Cluster sum of squares =  25646372.0
Iteration  11 Within Cluster sum of squares =  25641044.0
Iteration  12 Within Cluster sum of squares =  25639751.0
Iteration  13 Within Cluster sum of squares =  25639104.0
Iteration  14 Within Cluster sum of squares =  25638839.0
```

Execution Time:
Time for the algorithm to execute is **1.6617114543914795 secs**

```
Total run time (in seconds):  1.6617114543914795
```

Centroid and Class Labels:
Since we have run the algorithm for k=2 cluster, below are the classes / labels that are generated for the data points. There are 2 classes (1,2). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2}
[2 2 2 ... 2 2 2]

The final cluster centers(Output: centroid) are:
[[192. 181. 168.]
 [ 73.  77.  71.]]
```

Image Before and After Clustering:
Below are the results before and after k = 2 clustering. As we can see, the compressed final image is represented in 2 color categories, mostly black and white.

Original Image

Compressed Image

K-means with k = 4 clusters

When we execute the clustering algorithm, for k = 4 clusters, we would expect to see the 4 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py football.bmp 4 median

Results of Execution:

Iterations:

Implementation for **k = 4 took 33 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  19703619.0
Iteration  2 Within Cluster sum of squares =  17349879.0
Iteration  3 Within Cluster sum of squares =  16650842.0
Iteration  4 Within Cluster sum of squares =  16121573.0
Iteration  5 Within Cluster sum of squares =  15995211.0
Iteration  6 Within Cluster sum of squares =  15969966.0
Iteration  7 Within Cluster sum of squares =  15966059.0
Iteration  8 Within Cluster sum of squares =  15964676.0
Iteration  9 Within Cluster sum of squares =  15962147.0
Iteration  10 Within Cluster sum of squares =  15959359.0
Iteration  11 Within Cluster sum of squares =  15958846.0
Iteration  12 Within Cluster sum of squares =  15956622.0
Iteration  13 Within Cluster sum of squares =  15954329.0
Iteration  14 Within Cluster sum of squares =  15948290.0
Iteration  15 Within Cluster sum of squares =  15942349.0
Iteration  16 Within Cluster sum of squares =  15936268.0
Iteration  17 Within Cluster sum of squares =  15928569.0
Iteration  18 Within Cluster sum of squares =  15918353.0
Iteration  19 Within Cluster sum of squares =  15904362.0
Iteration  20 Within Cluster sum of squares =  15880911.0
Iteration  21 Within Cluster sum of squares =  15833043.0
Iteration  22 Within Cluster sum of squares =  15742316.0
Iteration  23 Within Cluster sum of squares =  15590812.0
Iteration  24 Within Cluster sum of squares =  15368491.0
Iteration  25 Within Cluster sum of squares =  15132631.0
Iteration  26 Within Cluster sum of squares =  14930638.0
Iteration  27 Within Cluster sum of squares =  14809101.0
Iteration  28 Within Cluster sum of squares =  14750099.0
Iteration  29 Within Cluster sum of squares =  14727818.0
Iteration  30 Within Cluster sum of squares =  14721036.0
Iteration  31 Within Cluster sum of squares =  14716183.0
Iteration  32 Within Cluster sum of squares =  14715038.0
Iteration  33 Within Cluster sum of squares =  14714923.0
```

Execution Time:

Time for the algorithm to execute is **4.177151918411255 secs**

```
Total run time (in seconds):  4.177151918411255
```

Centroid and Class Labels:

Since we have run the algorithm for k=4 cluster, below are the classes / labels that are generated for the data points. There are 4 classes (1,2,3,4). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4}
[2 2 2 ... 2 2 2]

The final cluster centers(Output: centroid) are:
[[217. 220. 222.]
 [117. 113.  87.]
 [185. 167. 151.]
 [ 31.  41.  35.]]
```

Image Before and After Clustering:

Below are the results before and after k = 4 clustering. As we can see, there is more distinction in terms of color components from the previous k=2 clusters. We can see shades of red color and some variant of green color in the final image.

Original Image

Compressed Image

K-means with k = 8 clusters

When we execute the clustering algorithm, for k = 8 clusters, we would expect to see the 8 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py football.bmp 8 median

Results of Execution:

Iterations:

Implementation for **k = 8 took 16 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  14921924.0
Iteration  2 Within Cluster sum of squares =  11327808.0
Iteration  3 Within Cluster sum of squares =  10611130.0
Iteration  4 Within Cluster sum of squares =  10530830.0
Iteration  5 Within Cluster sum of squares =  10506113.0
Iteration  6 Within Cluster sum of squares =  10484255.0
Iteration  7 Within Cluster sum of squares =  10469133.0
Iteration  8 Within Cluster sum of squares =  10455101.0
Iteration  9 Within Cluster sum of squares =  10444984.0
Iteration  10 Within Cluster sum of squares =  10437506.0
Iteration  11 Within Cluster sum of squares =  10434546.0
Iteration  12 Within Cluster sum of squares =  10433830.0
Iteration  13 Within Cluster sum of squares =  10433517.0
Iteration  14 Within Cluster sum of squares =  10432888.0
Iteration  15 Within Cluster sum of squares =  10431994.0
Iteration  16 Within Cluster sum of squares =  10431112.0
```

Execution Time:

Time for the algorithm to execute is **2.181169033050537 secs**

```
Total run time (in seconds):  2.181169033050537
```

Centroid and Class Labels:

Since we have run the algorithm for k=8 cluster, below are the classes / labels that are generated for the data points. There are 8 classes (1,2,3,4,5,6,7,8). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8}
[6 6 6 ... 3 3 3]

The final cluster centers(Output: centroid) are:
[[188. 172. 156.]
 [ 17.  20.  18.]
 [120. 137.  97.]
 [165. 112.  89.]
 [ 20.  63. 116.]
 [ 92.  92.  73.]
 [217. 221. 223.]
 [ 57.  55.  40.]]
```

Image Before and After Clustering:

Below are the results before and after k = 8 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. We can see shades of blue and more distinct green variants in image.

Original Image

Compressed Image

K-means with k = 16 clusters

When we execute the clustering algorithm, for k = 16 clusters, we would expect to see the 16 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py football.bmp 16 median

Results of Execution:

Iterations:

Implementation for **k = 16 took 40 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  9836309.0
Iteration  2 Within Cluster sum of squares =  8615094.0
Iteration  3 Within Cluster sum of squares =  8318433.0
Iteration  4 Within Cluster sum of squares =  8137102.0
Iteration  5 Within Cluster sum of squares =  8031792.0
Iteration  6 Within Cluster sum of squares =  7959409.0
Iteration  7 Within Cluster sum of squares =  7875853.0
Iteration  8 Within Cluster sum of squares =  7807724.0
Iteration  9 Within Cluster sum of squares =  7755342.0
Iteration  10 Within Cluster sum of squares =  7703823.0
Iteration  11 Within Cluster sum of squares =  7642740.0
Iteration  12 Within Cluster sum of squares =  7572873.0
Iteration  13 Within Cluster sum of squares =  7540885.0
Iteration  14 Within Cluster sum of squares =  7524527.0
Iteration  15 Within Cluster sum of squares =  7513524.0
Iteration  16 Within Cluster sum of squares =  7505756.0
Iteration  17 Within Cluster sum of squares =  7496531.0
Iteration  18 Within Cluster sum of squares =  7487485.0
Iteration  19 Within Cluster sum of squares =  7480757.0
Iteration  20 Within Cluster sum of squares =  7476815.0
Iteration  21 Within Cluster sum of squares =  7472759.0
Iteration  22 Within Cluster sum of squares =  7470034.0
Iteration  23 Within Cluster sum of squares =  7467727.0
Iteration  24 Within Cluster sum of squares =  7465054.0
Iteration  25 Within Cluster sum of squares =  7462752.0
Iteration  26 Within Cluster sum of squares =  7460618.0
Iteration  27 Within Cluster sum of squares =  7458297.0
Iteration  28 Within Cluster sum of squares =  7456552.0
Iteration  29 Within Cluster sum of squares =  7454596.0
Iteration  30 Within Cluster sum of squares =  7453652.0
Iteration  31 Within Cluster sum of squares =  7452904.0
Iteration  32 Within Cluster sum of squares =  7452643.0
Iteration  33 Within Cluster sum of squares =  7452481.0
Iteration  34 Within Cluster sum of squares =  7451937.0
Iteration  35 Within Cluster sum of squares =  7451773.0
Iteration  36 Within Cluster sum of squares =  7451543.0
Iteration  37 Within Cluster sum of squares =  7451389.0
Iteration  38 Within Cluster sum of squares =  7451293.0
Iteration  39 Within Cluster sum of squares =  7450967.0
Iteration  40 Within Cluster sum of squares =  7450425.0
```

Execution Time:

Time for the algorithm to execute is **10.185413122177124 secs**

```
Total run time (in seconds):  10.185413122177124
```

Centroid and Class Labels:

Since we have run the algorithm for k=16 cluster, below are the classes / labels that are generated for the data points. There are 16 classes (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
[ 9  9 12 ...  2  2  2]

The final cluster centers(Output: centroid) are:
[[140. 149. 151.]
 [119. 141.  88.]
 [225. 229. 230.]
 [158. 104.  82.]
 [101. 114. 126.]
 [194. 176. 164.]
 [ 17.  62. 115.]
 [207. 212. 214.]
 [ 94.  92.  71.]
 [ 13.  17.  15.]
 [173. 135. 112.]
 [ 67.  66.  49.]
 [ 42.  41.  30.]
 [187. 159. 141.]
 [217. 203. 126.]
 [186. 192. 193.]]
```

Image Before and After Clustering:

Below are the results before and after k = 16 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. This image is very similar to the original image with slight differences in colors.

Original Image

Compressed Image

K-means on GeorgiaTech.bmp (k = 2,4,8,16)

The image GeorgiaTech.bmp original image has the below structure.

```
Shape of the matrix obtained by reading the image
(400, 400, 3)
```

The image has 400*400 = 160000 pixels with 3 components (RGB) representing the color of each pixel.

Implementing k-means algorithm using the L1 norm(Manhattan distance) as the distance metric for the input image GeorgiaTech.bmp, below were the results that were obtained.

K-means with k = 2 clusters

When we execute the clustering algorithm, for k = 2 clusters, we would expect to see the 2 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py GeorgiaTech.bmp 2 median

Results of Execution:

Iterations:

Implementation for **k = 2 took 6 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every

iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  16216256.0
Iteration  2 Within Cluster sum of squares =  14431181.0
Iteration  3 Within Cluster sum of squares =  14335968.0
Iteration  4 Within Cluster sum of squares =  14325642.0
Iteration  5 Within Cluster sum of squares =  14324732.0
Iteration  6 Within Cluster sum of squares =  14324108.0
```

Execution Time:

Time for the algorithm to execute is **0.2800023555755615 secs**

```
Total run time (in seconds):  0.2800023555755615
```

Centroid and Class Labels:

Since we have run the algorithm for k=2 cluster, below are the classes / labels that are generated for the data points. There are 2 classes (1,2). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2}
[2 2 2 ... 1 1 1]

The final cluster centers(Output: centroid) are:
[[ 51.  52.  34.]
 [166. 159. 166.]]
```

Image Before and After Clustering:

Below are the results before and after k = 2 clustering. As we can see, the compressed final image is represented in 2 color categories, mostly black and white.

Original Image

Compressed Image

K-means with k = 4 clusters

When we execute the clustering algorithm, for k = 4 clusters, we would expect to see the 4 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py GeorgiaTech.bmp 4 median

Results of Execution:

Iterations:

Implementation for **k = 4 took 36 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  18133606.0
Iteration  2 Within Cluster sum of squares =  13063648.0
Iteration  3 Within Cluster sum of squares =  11992760.0
Iteration  4 Within Cluster sum of squares =  11461801.0
Iteration  5 Within Cluster sum of squares =  11195080.0
Iteration  6 Within Cluster sum of squares =  11035733.0
Iteration  7 Within Cluster sum of squares =  10915793.0
Iteration  8 Within Cluster sum of squares =  10824560.0
Iteration  9 Within Cluster sum of squares =  10761722.0
Iteration  10 Within Cluster sum of squares =  10717799.0
Iteration  11 Within Cluster sum of squares =  10686083.0
Iteration  12 Within Cluster sum of squares =  10663655.0
Iteration  13 Within Cluster sum of squares =  10646100.0
Iteration  14 Within Cluster sum of squares =  10636057.0
Iteration  15 Within Cluster sum of squares =  10626408.0
Iteration  16 Within Cluster sum of squares =  10618570.0
Iteration  17 Within Cluster sum of squares =  10610760.0
Iteration  18 Within Cluster sum of squares =  10605221.0
Iteration  19 Within Cluster sum of squares =  10596324.0
Iteration  20 Within Cluster sum of squares =  10585600.0
Iteration  21 Within Cluster sum of squares =  10570974.0
Iteration  22 Within Cluster sum of squares =  10550643.0
Iteration  23 Within Cluster sum of squares =  10524707.0
Iteration  24 Within Cluster sum of squares =  10494874.0
Iteration  25 Within Cluster sum of squares =  10459170.0
Iteration  26 Within Cluster sum of squares =  10402394.0
Iteration  27 Within Cluster sum of squares =  10294121.0
Iteration  28 Within Cluster sum of squares =  10183294.0
Iteration  29 Within Cluster sum of squares =  10129872.0
Iteration  30 Within Cluster sum of squares =  10109693.0
Iteration  31 Within Cluster sum of squares =  10101206.0
Iteration  32 Within Cluster sum of squares =  10098673.0
Iteration  33 Within Cluster sum of squares =  10098143.0
Iteration  34 Within Cluster sum of squares =  10097988.0
Iteration  35 Within Cluster sum of squares =  10097795.0
Iteration  36 Within Cluster sum of squares =  10097400.0
```

Execution Time:

Time for the algorithm to execute is **2.8326354026794434 secs**

```
Total run time (in seconds):  2.8326354026794434
```

Centroid and Class Labels:

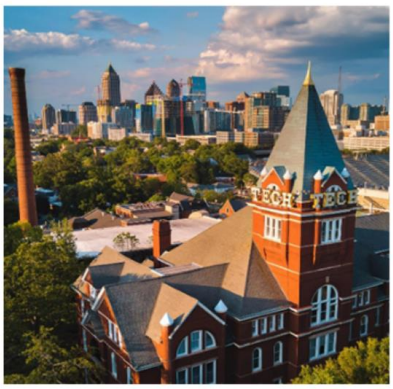
Since we have run the algorithm for k=4 cluster, below are the classes / labels that are generated for the data points. There are 4 classes (1,2,3,4). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4}
[4 4 4 ... 3 3 3]

The final cluster centers(Output: centroid) are:
[[211. 151. 104.]
 [ 91.  87.  76.]
 [ 38.  33.  26.]
 [148. 164. 179.]]
```

Image Before and After Clustering:

Below are the results before and after k = 4 clustering. As we can see, there is more distinction in terms of color components from the previous k=2 clusters. We can see shades of red color and some variant of green color in the final image.

Original Image



Compressed Image

K-means with k = 8 clusters

When we execute the clustering algorithm, for k = 8 clusters, we would expect to see the 8 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py GeorgiaTech.bmp 8 median

Results of Execution:

Iterations:

Implementation for **k = 8 took 44 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  9463611.0
Iteration  2 Within Cluster sum of squares =  7974321.0
Iteration  3 Within Cluster sum of squares =  7773842.0
Iteration  4 Within Cluster sum of squares =  7637898.0
Iteration  5 Within Cluster sum of squares =  7530648.0
Iteration  6 Within Cluster sum of squares =  7452718.0
Iteration  7 Within Cluster sum of squares =  7396009.0
Iteration  8 Within Cluster sum of squares =  7357279.0
Iteration  9 Within Cluster sum of squares =  7328489.0
Iteration  10 Within Cluster sum of squares =  7307138.0
Iteration  11 Within Cluster sum of squares =  7293262.0
Iteration  12 Within Cluster sum of squares =  7281426.0
Iteration  13 Within Cluster sum of squares =  7273158.0
Iteration  14 Within Cluster sum of squares =  7265069.0
Iteration  15 Within Cluster sum of squares =  7256461.5
Iteration  16 Within Cluster sum of squares =  7249814.0
Iteration  17 Within Cluster sum of squares =  7241616.0
Iteration  18 Within Cluster sum of squares =  7235768.0
Iteration  19 Within Cluster sum of squares =  7228767.0
Iteration  20 Within Cluster sum of squares =  7222338.0
Iteration  21 Within Cluster sum of squares =  7217415.0
Iteration  22 Within Cluster sum of squares =  7211743.0
Iteration  23 Within Cluster sum of squares =  7205748.0
Iteration  24 Within Cluster sum of squares =  7200871.0
Iteration  25 Within Cluster sum of squares =  7197513.0
Iteration  26 Within Cluster sum of squares =  7193252.0
Iteration  27 Within Cluster sum of squares =  7190425.0
Iteration  28 Within Cluster sum of squares =  7187392.0
Iteration  29 Within Cluster sum of squares =  7185336.0
Iteration  30 Within Cluster sum of squares =  7175887.0
Iteration  31 Within Cluster sum of squares =  7163277.0
Iteration  32 Within Cluster sum of squares =  7140983.0
Iteration  33 Within Cluster sum of squares =  7116558.0
Iteration  34 Within Cluster sum of squares =  7105537.0
Iteration  35 Within Cluster sum of squares =  7100795.0
Iteration  36 Within Cluster sum of squares =  7098267.0
Iteration  37 Within Cluster sum of squares =  7097299.0
Iteration  38 Within Cluster sum of squares =  7096177.0
Iteration  39 Within Cluster sum of squares =  7095440.0
Iteration  40 Within Cluster sum of squares =  7094851.0
Iteration  41 Within Cluster sum of squares =  7094513.0
Iteration  42 Within Cluster sum of squares =  7094114.0
Iteration  43 Within Cluster sum of squares =  7093675.0
Iteration  44 Within Cluster sum of squares =  7093492.0
```

Execution Time:

Time for the algorithm to execute is **5.703251123428345 secs**

```
Total run time (in seconds):  5.703251123428345
```

Centroid and Class Labels:

Since we have run the algorithm for k=8 cluster, below are the classes / labels that are generated for the data points. There are 8 classes (1,2,3,4,5,6,7,8). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8}
[3 3 3 ... 2 1 1]

The final cluster centers(Output: centroid) are:
[[ 35.  28.  24.]
 [ 85.  71.  31.]
 [240. 209. 192.]
 [163. 115.  46.]
 [137. 159. 176.]
 [ 94.  99. 100.]
 [ 47.  58.  67.]
 [208. 159. 116.]]
```

Image Before and After Clustering:

Below are the results before and after k = 8 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. We can see shades of more distinct green and red variants in image.

Original Image

Compressed Image

K-means with k = 16 clusters

When we execute the clustering algorithm, for k = 16 clusters, we would expect to see the 16 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py GeorgiaTech.bmp 16 median

Results of Execution:

Iterations:

Implementation for **k = 16 took 27 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  7232385.0
Iteration  2 Within Cluster sum of squares =  5718954.0
Iteration  3 Within Cluster sum of squares =  5457491.0
Iteration  4 Within Cluster sum of squares =  5290012.5
Iteration  5 Within Cluster sum of squares =  5128750.0
Iteration  6 Within Cluster sum of squares =  5049124.0
Iteration  7 Within Cluster sum of squares =  4986434.0
Iteration  8 Within Cluster sum of squares =  4923861.0
Iteration  9 Within Cluster sum of squares =  4864879.0
Iteration  10 Within Cluster sum of squares =  4804771.5
Iteration  11 Within Cluster sum of squares =  4755035.0
Iteration  12 Within Cluster sum of squares =  4734029.0
Iteration  13 Within Cluster sum of squares =  4726073.0
Iteration  14 Within Cluster sum of squares =  4719526.0
Iteration  15 Within Cluster sum of squares =  4715871.0
Iteration  16 Within Cluster sum of squares =  4713364.0
Iteration  17 Within Cluster sum of squares =  4711593.0
Iteration  18 Within Cluster sum of squares =  4710262.0
Iteration  19 Within Cluster sum of squares =  4709278.0
Iteration  20 Within Cluster sum of squares =  4707998.0
Iteration  21 Within Cluster sum of squares =  4707370.0
Iteration  22 Within Cluster sum of squares =  4706734.0
Iteration  23 Within Cluster sum of squares =  4706174.0
Iteration  24 Within Cluster sum of squares =  4705230.0
Iteration  25 Within Cluster sum of squares =  4704898.0
Iteration  26 Within Cluster sum of squares =  4704276.0
Iteration  27 Within Cluster sum of squares =  4704015.0
```

Execution Time:

Time for the algorithm to execute is **3.638381004333496 secs**

```
Total run time (in seconds):  3.638381004333496
```

Centroid and Class Labels:

Since we have run the algorithm for k=16 cluster, below are the classes / labels that are generated for the data points. There are 16 classes (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
[13 13 13 ...  9 11 11]

The final cluster centers(Output: centroid) are:
[[ 85. 124. 146.]
 [ 27.  23.  21.]
 [210. 158. 112.]
 [ 57.  71.  80.]
 [ 35.  49.  58.]
 [177. 169. 174.]
 [133. 150. 170.]
 [119.  93.  35.]
 [ 75.  63.  31.]
 [244. 118.  42.]
 [ 47.  34.  27.]
 [123. 165. 194.]
 [255. 255. 255.]
 [229. 197. 179.]
 [155. 122.  88.]
 [ 99.  92.  90.]]
```

Image Before and After Clustering:

Below are the results before and after k = 16 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. This image is very similar to the original image with slight differences in colors.

Original Image

Compressed Image

K-means on a third image (k = 2,4,8,16)

The image football.bmp original image has the below structure.

```
Shape of the matrix obtained by reading the image
(300, 200, 3)
```

The image has 300*300 = 60000 pixels with 3 components (RGB) representing the color of each pixel.

Implementing k-means algorithm using the L1 norm(Manhattan) as the distance metric for the input image football.bmp, below were the results that were obtained.

K-means with k = 2 clusters

When we execute the clustering algorithm, for k = 2 clusters, we would expect to see the 2 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py flowers1.jpg 2 median

Results of Execution:

Iterations:

Implementation for **k = 2 took 7 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  8790194.0
Iteration  2 Within Cluster sum of squares =  7730210.0
Iteration  3 Within Cluster sum of squares =  7657189.0
Iteration  4 Within Cluster sum of squares =  7639732.0
Iteration  5 Within Cluster sum of squares =  7636466.0
Iteration  6 Within Cluster sum of squares =  7635808.0
Iteration  7 Within Cluster sum of squares =  7635460.0
```

Execution Time:

Time for the algorithm to execute is **0.16253876686096191 secs**

```
Total run time (in seconds):  0.16253876686096191
```

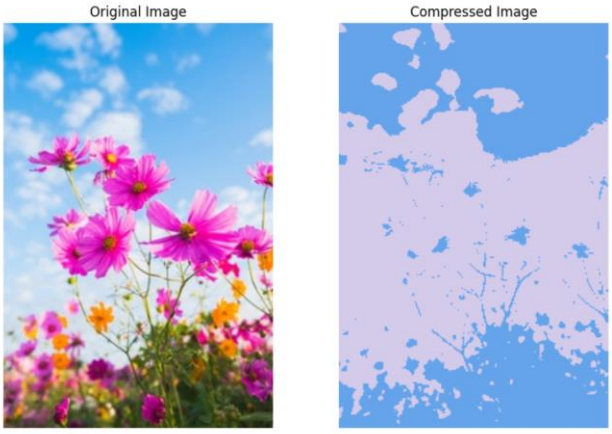
Centroid and Class Labels:

Since we have run the algorithm for k=2 cluster, below are the classes / labels that are generated for the data points. There are 2 classes (1,2). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2}
[2 2 1 ... 1 1 1]

The final cluster centers(Output: centroid) are:
[[101. 164. 235.]
 [212. 203. 235.]]
```

Image Before and After Clustering:

Below are the results before and after k = 2 clustering. As we can see, the compressed final image is represented in 2 color categories, mostly blue and pink.



K-means with k = 4 clusters

When we execute the clustering algorithm, for k = 4 clusters, we would expect to see the 4 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:
python KMeans.py flowers1.jpg 4 median

Results of Execution:

Iterations:
Implementation for **k = 4 took 9 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  6100794.0
Iteration  2 Within Cluster sum of squares =  4435092.0
Iteration  3 Within Cluster sum of squares =  4050236.0
Iteration  4 Within Cluster sum of squares =  3894654.0
Iteration  5 Within Cluster sum of squares =  3860847.0
Iteration  6 Within Cluster sum of squares =  3854191.0
Iteration  7 Within Cluster sum of squares =  3852814.0
Iteration  8 Within Cluster sum of squares =  3852319.0
Iteration  9 Within Cluster sum of squares =  3852109.0
```

Execution Time:
Time for the algorithm to execute is **0.2620682716369629 secs**

```
Total run time (in seconds):  0.2620682716369629
```

Centroid and Class Labels:
Since we have run the algorithm for k=4 cluster, below are the classes / labels that are generated for the data points. There are 4 classes (1,2,3,4). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  {1, 2, 3, 4}
[1 1 1 ... 2 2 2]

The final cluster centers(Output: centroid) are:
[[111. 181. 239.]
 [ 97.  77.  43.]
 [229.  88. 178.]
 [209. 217. 240.]]
```

Image Before and After Clustering:
Below are the results before and after k = 4 clustering. As we can see, there is more distinction in terms of color components from the previous k=2 clusters. We can see shades of white, pink,blue and dark pink now.

K-means with k = 8 clusters

When we execute the clustering algorithm, for k = 8 clusters, we would expect to see the 8 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py flowers1.jpg 8 median

Results of Execution:

Iterations:

Implementation for **k = 8 took 22 iterations to converge**. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  3671061.0
Iteration  2 Within Cluster sum of squares =  3056392.0
Iteration  3 Within Cluster sum of squares =  2812399.0
Iteration  4 Within Cluster sum of squares =  2756109.0
Iteration  5 Within Cluster sum of squares =  2730216.0
Iteration  6 Within Cluster sum of squares =  2711724.0
Iteration  7 Within Cluster sum of squares =  2699571.0
Iteration  8 Within Cluster sum of squares =  2688748.0
Iteration  9 Within Cluster sum of squares =  2681145.0
Iteration  10 Within Cluster sum of squares =  2673130.0
Iteration  11 Within Cluster sum of squares =  2668004.0
Iteration  12 Within Cluster sum of squares =  2665173.0
Iteration  13 Within Cluster sum of squares =  2662950.0
Iteration  14 Within Cluster sum of squares =  2662334.0
Iteration  15 Within Cluster sum of squares =  2662236.0
Iteration  16 Within Cluster sum of squares =  2662114.0
Iteration  17 Within Cluster sum of squares =  2662092.0
Iteration  18 Within Cluster sum of squares =  2661951.0
Iteration  19 Within Cluster sum of squares =  2661879.0
Iteration  20 Within Cluster sum of squares =  2661818.0
Iteration  21 Within Cluster sum of squares =  2661733.0
Iteration  22 Within Cluster sum of squares =  2661716.0
```

Execution Time:

Time for the algorithm to execute is **0.7954628467559814 secs**

```
Total run time (in seconds):  0.7954628467559814
```

Centroid and Class Labels:

Since we have run the algorithm for k=8 cluster, below are the classes / labels that are generated for the data points. There are 8 classes (1,2,3,4,5,6,7,8). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

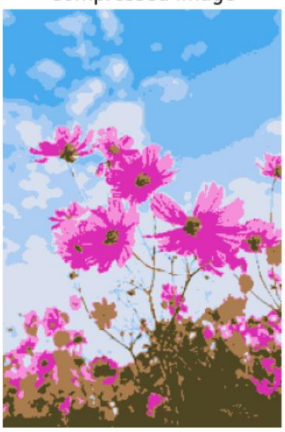
```
The final class labels (Output: class) are  {1, 2, 3, 4, 5, 6, 7, 8}
[5 3 3 ... 6 6 6]

The final cluster centers(Output: centroid) are:
[[246. 143. 217.]
 [ 73. 170. 237.]
 [128. 189. 240.]
 [220.  44. 179.]
 [173. 206. 242.]
 [ 79.  71.  36.]
 [216. 224. 240.]
 [171. 124.  78.]]
```

Image Before and After Clustering:

Below are the results before and after k = 8 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. We can see shades more distinction of while and the flower colors.

Original Image

Compressed Image



K-means with k = 16 clusters

When we execute the clustering algorithm, for k = 16 clusters, we would expect to see the 16 classes in the image, clustered by their RGB components, which would represent the colors in the image.

Code Execution Command:

python KMeans.py flowers1.jpg 16 median

Results of Execution:

Iterations:

Implementation for **k = 16 took 29 iterations to converge**. Below are the iteration results. For each iteration, I have computed the Within cluster sum of squares. We can observe that this sum keeps reducing for every iteration until the local optimum is reached. This means that for every iteration, the clusters get more closely packed.

```
Iteration  1 Within Cluster sum of squares =  4976650.0
Iteration  2 Within Cluster sum of squares =  2466411.0
Iteration  3 Within Cluster sum of squares =  2247679.0
Iteration  4 Within Cluster sum of squares =  2195981.0
Iteration  5 Within Cluster sum of squares =  2166642.0
Iteration  6 Within Cluster sum of squares =  2138892.0
Iteration  7 Within Cluster sum of squares =  2104617.0
Iteration  8 Within Cluster sum of squares =  2076419.0
Iteration  9 Within Cluster sum of squares =  2052342.0
Iteration  10 Within Cluster sum of squares =  2030009.5
Iteration  11 Within Cluster sum of squares =  2006536.5
Iteration  12 Within Cluster sum of squares =  1981654.0
Iteration  13 Within Cluster sum of squares =  1956833.0
Iteration  14 Within Cluster sum of squares =  1936217.0
Iteration  15 Within Cluster sum of squares =  1920691.0
Iteration  16 Within Cluster sum of squares =  1909035.0
Iteration  17 Within Cluster sum of squares =  1901035.0
Iteration  18 Within Cluster sum of squares =  1893826.0
Iteration  19 Within Cluster sum of squares =  1886347.0
Iteration  20 Within Cluster sum of squares =  1880590.0
Iteration  21 Within Cluster sum of squares =  1875752.0
Iteration  22 Within Cluster sum of squares =  1872969.0
Iteration  23 Within Cluster sum of squares =  1870520.0
Iteration  24 Within Cluster sum of squares =  1868194.0
Iteration  25 Within Cluster sum of squares =  1867292.0
Iteration  26 Within Cluster sum of squares =  1866915.0
Iteration  27 Within Cluster sum of squares =  1866646.0
Iteration  28 Within Cluster sum of squares =  1866558.0
Iteration  29 Within Cluster sum of squares =  1866547.0
```

Execution Time:

Time for the algorithm to execute is **1.5211224555969238 secs**

```
Total run time (in seconds):  1.5211224555969238
```

Centroid and Class Labels:

Since we have run the algorithm for k=16 cluster, below are the classes / labels that are generated for the data points. There are 16 classes (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16). Each data point is assigned to either one of the clusters. The centroids for each cluster centers are also shown below.

```
The final class labels (Output: class) are  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
[14 14  2 ...  5  5  5]

The final cluster centers(Output: centroid) are:
[[188.  214.  243. ]
 [127.  189.  240. ]
 [220.   19.  173. ]
 [232.   76.  202. ]
 [ 65.   66.   30. ]
 [248.  131.  219. ]
 [210.  224.  239. ]
 [158.   52.  107. ]
 [116.  100.   55. ]
 [ 31.5 163.  235. ]
 [ 88.  175.  238. ]
 [196.  162.  139. ]
 [226.  231.  240. ]
 [160.  202.  242. ]
 [240.  140.   24. ]
 [250.  185.  231. ]]
```

Image Before and After Clustering:

Below are the results before and after k = 16 clustering. As we can see, there is more distinction in terms of color components from the previous clusters. This image is very similar to the original image with slight differences in colors.


Original Image
Compressed Image