ISYE6740-HW1

Prashant Kubsad, pkubsad@gatech.edu

January 2021

1 K-means clustering [60 points]

Given m data points x^i, $i = 1, \dots, m$, K-means clustering algorithm groups them into k clusters by minimizing the distortion function over $\{r^{ij}, \mu^j\}$

$$J = \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} \|\mathrm{x}^i - \mu^j\|^2, \tag{1}$$

where $r^{ij} = 1$ if x^i belongs to the j-th cluster and $r^{ij} = 0$ otherwise.

1. (10 points) Derive mathematically that using the squared Euclidean distance $\|\mathrm{x}^i - \mu^j\|^2$ as the dissimilarity function, the centroid that minimizes the distortion function J for given assignments r^{ij} are given by

$$\mu^j = \frac{\sum_i r^{ij} \mathrm{x}^i}{\sum_i r^{ij}}.$$

That is, μ^j is the center of j-th cluster.
Hint: You may start by taking the partial derivative of J with respect to μ^j, with r^{ij} fixed.

Answer: As mentioned in the hint, to determine the minimization of distortion function, we can do a partial derivative of function J with respect to μ^j.

$$\frac{\partial J}{\partial \mu^j} = 0. \Rightarrow \frac{\partial \left(\sum_{i=1}^{m} r^{ij} \|\mathrm{x}^i - \mu^j\|^2\right)}{\partial \mu^j} = 0.$$

Expanding the p2 norm:

$$\begin{aligned} \|x^i - \mu^j\|_2^2 &= (x^i - \mu^j)^T (x^i - \mu^j) \\ &= (x^{iT} x^i - x^{iT} \mu^j - \mu^{jT} x^i + \mu^{jT} \mu^j) \\ &= (x^{iT} x^i - 2x^{iT} \mu^j + \mu^{jT} \mu^j) \end{aligned}$$

$$\frac{\partial J}{\partial \mu^j} = \frac{\partial \left(\sum_{i=1}^{m} r^{ij} (x^{iT} x^i - 2x^{iT} \mu^j + \mu^{jT} \mu^j)\right)}{\partial \mu^j}$$

Running a gradient wrt to μ^j

$$\frac{\partial J}{\partial \mu^j} = \sum_{i=1}^{m} r^{ij} (-2x^i + 2\mu^j) = 0$$

Rearranging the equation

$$= 2\sum_{i=1}^{m} r^{ij}\left(x^i - \mu^j\right) = 0$$

$$\sum_{i=1}^{m} r^{ij} x^i - \mu^j \sum_{i=1}^{m} r^{ij} = 0$$

$$\boxed{\mu^j = \frac{\sum_{i=1}^{m} r^{ij} x^i}{\sum_{i=1}^{m} r^{ij}}}$$

reference: Example: Linear algebra and derivatives - Prof X
reference: Pattern Recognition And Machine Learning - Section 9.4 - C m Bishop

2. (10 points) Derive mathematically what should be the assignment variables r^{ij} be to minimize the distortion function J, when the centroids μ^j are fixed.

   Answer: This is the Expectation part of the EM algorithm of K-means clustering. A given data point can be assigned to only one cluster(centroid). If the the centroids are fixed, the incoming datapoint will be assigned to the centroid that it is closest to. That implies, the centroid which will have minimum cost function for that data point will get that data-point.r^{ij} is binary field with value equal to 1 for the cluster it is assigned to and 0 for every other cluster. Mathematically we can represent that as:

$$r^{ij} = \left\{ \begin{matrix} 1 \ \ \text{i}fj = argmin^j \|x^i - \mu^j\|^2 \\ \hfill 0 \ \ otherwise \end{matrix} \right\}$$

   Example: If there are 3 centroids μ^1, μ^2, μ^3, and a new data-point x^1 is to be clustered to one of these centroids,x^1 will be assigned to the centroid it is closest to. In other words, the centroid which will result in least cost function, $\|\mathrm{x}^i - \mu^j\|2_2$. If the distance (cost function) of x^1 with centroids is as below:

$$\|x^1 - \mu^1\|_2^2 = 4$$

$$\|x^1 - \mu^2\|_2^2 = 9$$

$$\|x^1 - \mu^3\|_2^2 = 16$$

   Then, μ^1 is the centroid with least distance and hence x^1 will be assigned to μ^1, hence r^{11}=1 and r^{12}=0, r^{13}=0

3. (10 points) Write down a pseudocode for K-means algorithm here, based on your derived results.

   Answer:
   The approach K means follows is 2 step process similar to a Expectation-Maximization model [EM]. In the 'Expectation' step, we assign every data point to a nearest cluster. In the 'Maximization' step, we determine new centroid for a given cluster. This process is repeated until the centroids are not changing. At this point, the model has converged to an optimal point beyond which there are no gains.

   algorithm: K-means clustering **is**
   input:

X = {$x^1, x^2, x^3, \ldots, x^m$} - Data points that are to be clustered
k = Number of clusters.

output:

C = $\{\mu^1, \mu^2, \mu^3, \ldots, \mu^j\}$ - Centroids of clusters
$\pi = \pi(j)$ | j $\epsilon 1, 2, 3, \ldots, k$ - set of clusters containing datapoints in that cluster

```
Initialize j cluster centers randomly
C[j] ←{$\mu^1, \mu^2, \mu^3, ..., \mu^j$}
$isConverged \leftarrow false$


while !($isConverged$) do
    $C_{previous} \leftarrow C$ //placeholder to be used later to compare with new set of centroids

    for all ($x^i$ in X) , ($\mu^j$ in C) do
        assign $x^i$ to nearest cluster $\pi(j)$ determined by $argmin^j \|x^i - \mu^j\|^2$
        $\pi[j]$ ←x$^i$
    end for

    for all $\pi^j$ in $\pi$ do
        determine new center of cluster
        $\mu^j \leftarrow$ average( x present in the $j^{th}$ cluster )
        C[j] ←$\mu^j$
    end for

    if C == $C_{previous}$ then
        $isConverged \leftarrow true$
    end if
end while
```

4. (10 points) Explain why *K*-means algorithm converges to a local optimum in finite steps.

   Answer: The K-Means clustering is a kind of heuristic algorithm, that will give us a very good result faster. This result might not be the best result, but, it will be closer to the best result. The objective function for K-Means clustering:

$$\min_{c,\pi} \frac{1}{m} \sum_{i=1}^{m} \|x^i - c^{\pi(i)}\|^2$$

$$c = setofclustercenters, m = numberofdatapoints, \pi(i) = clustercontainingx^i$$

   This function is non-polynomial hard,that is, if we plot the value of the objective function [Figure 1], we end up with a shape like below. The overall optimal point of the system is the global optimum point as marked. There will be valleys which will not be as small as global minimum, but they are very small compared to their neighbouring points. These points are called local minimums.Because of the non-convexity of the objective function, we might not be able to find global minimum. Based on the different start points, we might arrive at different local minimums. Given all these conditions, we try to solve to local minimums in K means clustering instead of global minimums.

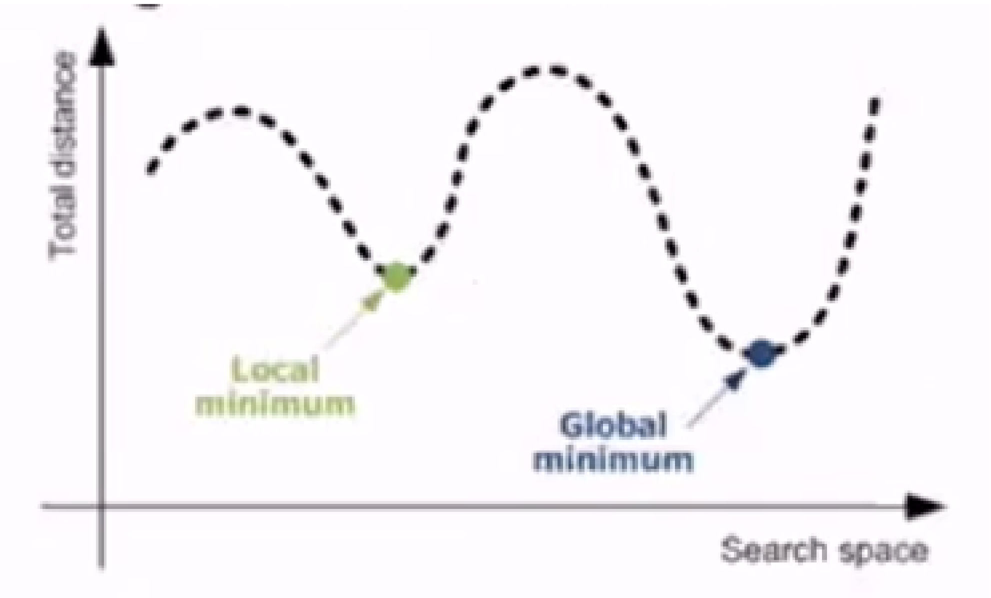


Figure 1: src: slide 18 from Lecture 2 video

When Solving for this local minimums, the value of objective function does not oscillate. It will be monotonically decreasing. The 2 steps of K-means clustering try to reduce the the value of objective function. Each iteration is done only if there is a reduction in the value of the objective function. If there is a increase in the value of objective function, we have found our local minimum and we stop the iteration. The objective function is sum of squares function and hence it cannot be negative. So the smallest possible value of the objective function can be 0. That means, as we are iterating the steps of K-means, the algorithm is converging towards a zero. The worst case scenario for this convergence is K^m steps. This number can be large but is still finite. Based on this information, we can conclude that the K - means algorithm converges to a local optimum in finite steps.

5. (20 points) Calculate k-means by hands using Euclidean distance, i.e., the set up in Equation (1). Given 5 data points configuration in Figure 1. Assume $k = 2$. Assuming the initialization of centroid as shown.

   (a) (15 points) Complete the following table for all iterations until the algorithm converges.

   Answer: Lets start with recording the co-ordinates of the datapoints and centroids given in the question.

Co-ordinates.	μ^1	μ^2	1	2	3	4	5
	(-3,-1)	(2,1)	(2,2)	(-1,1)	(3,1)	(0,-1)	(-2,-2)

Distance Calculation: Using the formula to calculate the Euclidean distance $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
<u>Iteration 1:</u> Using the pre-defined centroids
Distance From Point1 to Centroid 1: $D^{11} = \sqrt{(2-(-3))^2 + (2-(-1))^2} = \sqrt{5^2 + 3^2} = \sqrt{34} = 5.83$
$D^{21} = \sqrt{(-1-(-3))^2 + (1-(-1))^2} = \sqrt{2^2 + 2^2} = \sqrt{8} = 2.83$
$D^{31} = \sqrt{(3-(-3))^2 + (1-(-1))^2} = \sqrt{6^2 + 2^2} = \sqrt{40} = 6.32$
$D^{41} = \sqrt{(0-(-3))^2 + (-1-(-1))^2} = \sqrt{3^2 + 0} = \sqrt{9} = 3$

$D^{51} = \sqrt{(-2-(-3))^2 + (-2-(-1))^2} = \sqrt{1^2+1^2} = \sqrt{2} = 1.41$

Similarly If we calculate distances for data points with 2nd centroid $D^{12} = \sqrt{(2-2)^2+(2-1)^2} = \sqrt{0+1^2} = \sqrt{1} = 1$
$D^{22} = \sqrt{(-1-2)^2+(1-1)^2} = \sqrt{3^2+0} = \sqrt{9} = 3$
$D^{32} = \sqrt{(3-2)^2+(1-1)^2} = \sqrt{1^2+0} = \sqrt{1} = 1$
$D^{42} = \sqrt{(0-2)^2+(-1-1)^2} = \sqrt{2^2+2^2} = \sqrt{8} = 2.83$
$D^{52} = \sqrt{(-2-2)^2+(-2-1)^2} = \sqrt{4^2+3^2} = \sqrt{25} = 5$

Distances from all points to each centroid, and $argmin^j \|x^i - \mu^j\|$ circled.

Distances.	pt^1	pt^2	pt^3	pt^4	pt^5
μ^1	5.83	$\textcircled{2.83}$	6.32	3	$\textcircled{1.41}$
μ^2	$\textcircled{1}$	3	$\textcircled{1}$	$\textcircled{2.83}$	5

At the end of iteration 1, following will be the clusters:

Iteration No.	μ^1	μ^2	r^{11}	r^{21}	r^{31}	r^{41}	r^{51}
1.	(-3,-1)	(2,1)	0	1	0	0	1

<u>Iteration 2.:</u> Calculating new centroids.
μ^1=avg(pt2,pt5) = $(\frac{-1+(-2)}{2}, \frac{1+(-2)}{2})$ = $(\frac{-3}{2}, \frac{-1}{2})$ = (-1.5,0.5)

$\mu^2 = avg(pt1, pt3, pt4) = (\frac{2+3+0}{3}, \frac{2+1-1}{3}) = (\frac{5}{3}, \frac{2}{3}) = (1.67, 0.67)$

Calculating distance like in Iteration1, we get the following distances:

Distances.	pt^1	pt^2	pt^3	pt^4	pt^5
μ^1	4.3	$\textcircled{1.58}$	4.74	$\textcircled{1.58}$	$\textcircled{1.58}$
μ^2	$\textcircled{1.37}$	2.69	$\textcircled{1.37}$	2.36	5

We can observe that datapoint 4 has moved from cluster 1 to cluster 2 in this iteration.:

Iteration No.	μ^1	μ^2	r^{11}	r^{21}	r^{31}	r^{41}	r^{51}
1.	(-3,-1)	(2,1)	0	1	0	0	1
2.	(-1.5,-0.5)	(1.67,0.67)	0	1	0	1	1

<u>Iteration 3.:</u> Calculating new centroids.
μ^1=avg(pt2,pt4,pt5) = $(\frac{-3}{3}, \frac{-2}{3})$ = (-1,-0.67)

$\mu^2 = avg(pt1, pt3) = (\frac{5}{2}, \frac{3}{2}) = (2.5, 1.5)$

Calculating distance like in Iteration1, we get the following distances:

Distances.	pt^1	pt^2	pt^3	pt^4	pt^5
μ^1	4.01	[✓ 1.67]	4.33	[✓ 1.05]	[✓ 1.66]
μ^2	[✓ 0.71]	3.54	[✓ 0.71]	3.54	5.7

Cluster allocation after the end of iteration 3:

Iteration No.	μ^1	μ^2	r^{11}	r^{21}	r^{31}	r^{41}	r^{51}
1.	(-3,-1)	(2,1)	0	1	0	0	1
2.	(-1.5,-0.5)	(1.67,0.67)	0	1	0	1	1
3.	(-1,-0.67)	(2.5,1.5)	0	1	0	1	1

(b) (5) points How many iterations it takes for k-means to converge?

Answer: If we calculate centroids in iteration 4, they will be same as in iteration 3. Even though the cluster allocations between iteration 2 and 3 are same, we will still proceed further until the centroids are not changing.

Iteration No.	μ^1	μ^2	r^{11}	r^{21}	r^{31}	r^{41}	r^{51}
1.	(-3,-1)	(2,1)	0	1	0	0	1
2.	(-1.5,-0.5)	(1.67,0.67)	0	1	0	1	1
3.	(-1,-0.67)	(2.5,1.5)	0	1	0	1	1
4.	(-1,-0.67)	(2.5,1.5)	0	1	0	1	1

We can conclude that the algorithm has converged after **3 iterations.**

2.1. (30 points) Compress pictures using k-means, for `beach.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.

Answer: I have attached the **pkubsad_hw1_question2.ipynb** notebook with the implementation of the python code. The output of the python file is attached as a separate pdf file (pkubsad_hw1_question2.pdf) and also the contents are added to the end of this document.

Image	K	iterations	time-taken (s)
football.bmp	2	23	73.56
football.bmp	4	67	200.6
football.bmp	8	151	456.8
football.bmp	16	138	417
———	—-	————	————-
beach.bmp	2	20	14.5
beach.bmp	4	19	14
beach.bmp	8	103	204.3
beach.bmp	16	177	306

Image	K	iterations	time-taken (s)
grass.bmp	2	10	9.05
grass.bmp	4	25	24.84
grass.bmp	8	54	53.82
grass.bmp	16	73	77

All the resulted compressed pictures are in the attached zip file, under output folder.

Observation:

(a) The value of K determines the number of colors that will be in the reduced image. For.ex. when K=2, the resulting compressed image only has 2 colors that can best represent the picture.

(b) The execution times and number of iterations for convergence are generally faster for lesser values of K. This is not thumb rule but more of a general behavior that lesser K values are resolved quickly.

(c) The execution time also depends on the centroids that are initialized. As part of the next question, I have more deliberate bad choices for centroids.

2.2. (10 points) Run your *k*-means implementation with different initialization centroids. How does this it affect your final result? (We usually randomize initial location of centroids in general. To answer this question, an intentional poor assignment may be useful.) Please write in your report.

Answer:

	Centroids	Iterations	Time-Taken(s)
1	[0,0,0],[255,255,255], [100,100,100],[200,200,200]	27	75.86
2	[109,109,109],[110,110,110], [111,111,111],[112,112,112]	37	105.15
3	[0,1,2],[2,1,0], [244,0,0],[255,0,0]	30	84.51
4	[1,2,3],[4,5,6], [7,8,9],[10,11,12]	75	221.35

- a. centroids spaced almost equally and spread from 0 to 255
- b. All 4 centroids very close to each other but in the middle of the spectrum (109-114).
- c. 2 centroid close to each other at the beginnin of spectrum and other 2 at the end of the spectrum.
- d. All 4 centroids very close to each other but at the beginning of the spectrum (1-12).

Observations:

- Irrespective of centroids I pick, the resulting convergence and clustered picture seems to be same.
- The first centroid that has 4 pts spread accross the spectrum converges faster. In around 27 iterations
- The second centroid set that has all 4 points very close to each other, in the middle of the spectrum, takes longer to converge. In around 37 iterations.
- The third set has 2 centroids as at the beginning and 2 centroids at the end of the spectrum, takes longer than 1st set of centroid but faster than 2nd set of centroids
- The 4th set which has all centroids close to each other at one end of spectrum takes longest.

Based on the observations, if the initial centroids are not close to each other and spread across the spectrum, it will result in convergence faster. This is observation is purely based on the data above. I have run this program many times and consistently got similar results.

Untitled

January 25, 2021

K-Means Clustering: I have code and the test code to run through the algorithm of K-means. The first section is for importing all the libraries we will be using in the code.

[6]:
```
# Importing all the necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from matplotlib.pyplot import imshow
import time
```

The following block of code implements the k_means_clustering logic. As demonstrated in the answer for the question 3,the code below follows the steps of the algorithm: - Use the centers passed in the input, if none, then initialize random centers - Initialize cluster assignment to 0 for all datapoints. - Calculate p2 norm distance for each datapoint with all the centers. -using numpy's linalg package and norm function to calculate this distance - For each datapoint, assign cluster which has lowest distance. - Recalculate a new centroid within a cluster by calculating mean of all datapoints within the cluster. - Check for convergence = are the centroids in the current iteration same as previous iteration.

[5]:
```
def k_means_clustering(inputVector, k,
           initialClusterCenters=None):


    noOfDataPoints=len(inputVector)
    hasConverged = False

# use the initial cluster centroids if passed in the input
# If not initialize random centroids

    if initialClusterCenters is None:
        index = np.random.choice(noOfDataPoints, k, replace=False)
        centroids = inputVector[index]
    else:
        centroids = initialClusterCenters

# initializing array of 0s for clusters
    clusters = np.zeros(noOfDataPoints)
```

```
# initializing temp variable to hold old centers.
# This will be used later to check the convergence
    centersIMinusOne = centroids
    iteration = 1

# Loop through 2 steps as discussed in the lecture:
#     1. assign data points to a nearest cluster
#     2. recalculate the new centroid of the cluster

    while (not hasConverged):
        centersIMinusOne = centroids

#calculating the euclidean distance using norm function from
# numpy's linear algebra functions
# store the distance from each data point to each centroid.

        p2Distances=np.empty((noOfDataPoints,k))
        for i in range(noOfDataPoints):
            p2Distances[i,:]=np.linalg.norm(inputVector[i,:
 ↪]-centroids,ord=2,axis=1)**2

# For each data point, assign to cluster which has lowest distance
        clusters = np.argmin(p2Distances, axis=1)

#calculate new centroid by calculating mean of all the points within a cluster ␣
 ↪
        centroids = np.empty(centroids.shape)
        for j in range(k):

#Ignoring empty clusters if there are any at higher values of K
            if((inputVector[clusters==j]).size ==0):
                print("ignoring empty cluster")
                continue

            centroids[j,:]=np.mean(inputVector[clusters==j,:],axis=0)

#compare the new centroids with previous set of centroids to see if they have␣
 ↪changed
        hasConverged = np.array_equal(centersIMinusOne,centroids)
        iteration += 1
    print("number of iterations for k = ", k ," is :", iteration)

#converting the return parameters to datatypes expected in the question.
    return np.array([clusters]).T,np.asmatrix(centroids)
```

The following code is the helper code that runs the kmeans code written above. It also has helped methods that convert image to an array and also array back to image.

[8]:

```
%matplotlib inline
# Testing function block:
#     Uses MatPlot lib's image function to convert image into array.
#     Each entry of the array will have the RGB values of each pixel
#     Reference: Refered to my code in Assignment 14 from CSE 6040 which I␣
 ↪undertook in Fall`20.

def convertImageToArray(path):
    img = Image.open(path)
    imgArray = np.array(img, dtype='int32')
    img.close()
    return imgArray

def convertArrayToImageAndDisplay(arr):
    arr = arr.astype(dtype='uint8')
    img = Image.fromarray(arr, 'RGB')
    imshow(np.asarray(img))
    plt.show()

# calls k means method, records time for execution and prints clustered image.

def run_k_means(k,imgMatrix,centers=None):
    imgIntoSingleArray = imgMatrix.reshape(-1, imgMatrix.shape[-1])
    r, c, l = imgMatrix.shape
    print("running for ",k," centers")
    start=time.time()
    labels,centroids=k_means_clustering(imgIntoSingleArray, k,centers)
    end = time.time()
    print(f"Runtime of the program for ", k," centers:", {end - start})
    imgClusteredArray = np.array([centroids[label] for label in labels])
    imgClusteredMatrix = np.reshape(imgClusteredArray, (r, c, l), order="C")
    print("printing the ",k," clustered image")
    convertArrayToImageAndDisplay(imgClusteredMatrix)
    return labels,centroids
```

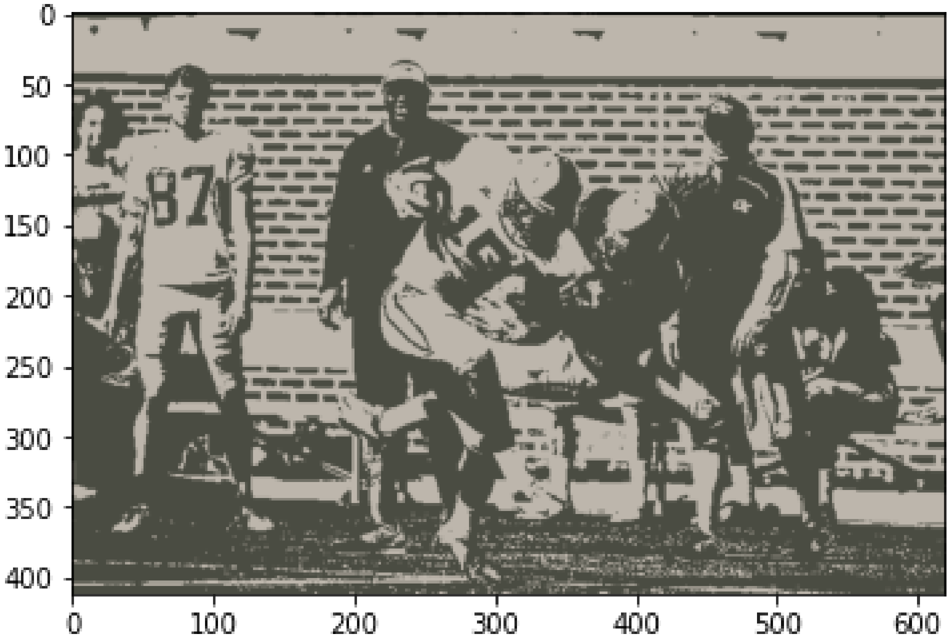
The following is a quick test code that will test the k_means_clustering logic with one file and with 2 centers.Validating the response contains two parameters: - clusters - a column array that will show cluster assignment of a datapoint - cetroids - a matrix of (k,3) shape that will give the co-ordinates of the final centroids.

[9]:

```
# Test Code to verify the code
# Verifying the response has 2 parameters:
#     1. Clusters : Column vector represnting cluster arrangement of each data␣
 ↪point
#     2. Centroids: Matrix of K rows and 3 columns representing the centroids.
```

```
footballImage = "./data/football.bmp"
img_arr=convertImageToArray(footballImage)
clusters,centroids=run_k_means(2,img_arr)
print("clusters:")

print(type(clusters))
print(clusters.shape)
print(clusters)



print("centroids:")
print(type(centroids))
print(centroids.shape)
display(centroids)
```

```
running for  2  centers
number of iterations for k =  2  is : 31
Runtime of the program for  2  centers: {93.95737099647522}
printing the  2  clustered image
```



```
matrix([[189.91441097, 182.64826808, 172.45584686],
        [ 74.59407487,  76.04869875,  66.44037077]])
```

```
clusters:
```

```
<class 'numpy.ndarray'>
(255440, 1)
[[1]
 [1]
 [1]
 …
 [1]
 [1]
 [1]]
centroids:
<class 'numpy.matrix'>
(2, 3)
```

0.0.1 Question 2.1:

Compress pictures using k-means, for beach.bmp ,football.bmp and custom image with k=2,4,6,8. Record the number of iterations it takes and time taken for each K.

0.0.2 Answer:

I have printed the iterations for each K value and time taken for convergence. I have also printed the resulting images in the output.

Image	K	iterations	time-taken (s)
football.bmp	2	23	73.56
football.bmp	4	67	200.6
football.bmp	8	151	456.8
football.bmp	16	138	417
————	—–	————	—————–
beach.bmp	2	20	14.5
beach.bmp	4	19	14
beach.bmp	8	103	204.3
beach.bmp	16	177	306
————	—–	————	—————–
grass.bmp	2	10	9.05
grass.bmp	4	25	24.84
grass.bmp	8	54	53.82
grass.bmp	16	73	77
————	—–	————	—————–

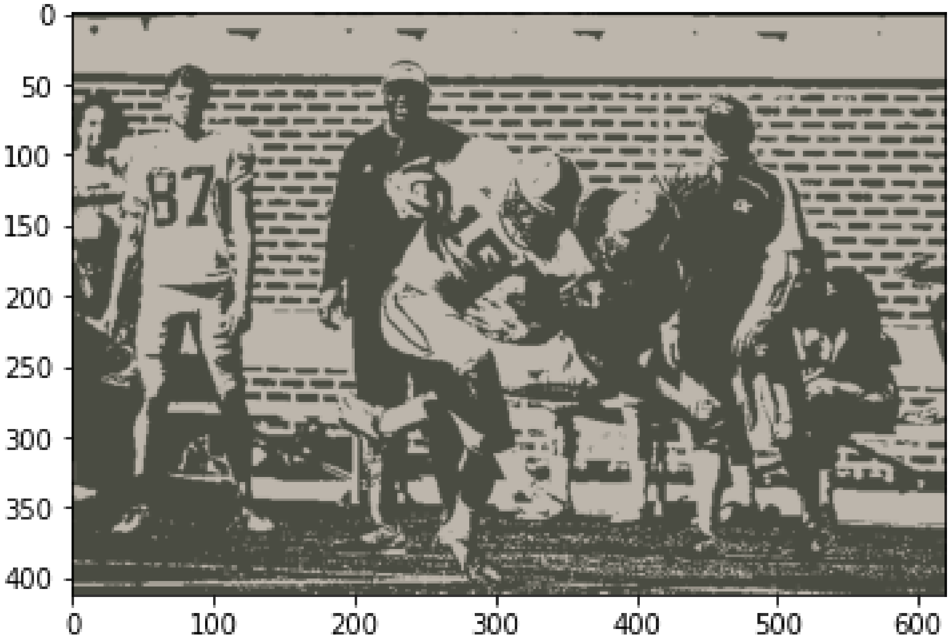
[402]:

```python
## Question 2.1: Compress pictures using k-means, for beach.bmp ,football.bmp and custom image
# with k=2,4,6,8. Record the number of iterations it takes and time taken for each K
images=[]
footballImage = "./data/football.bmp"
beachImage = "./data/beach.bmp"
```

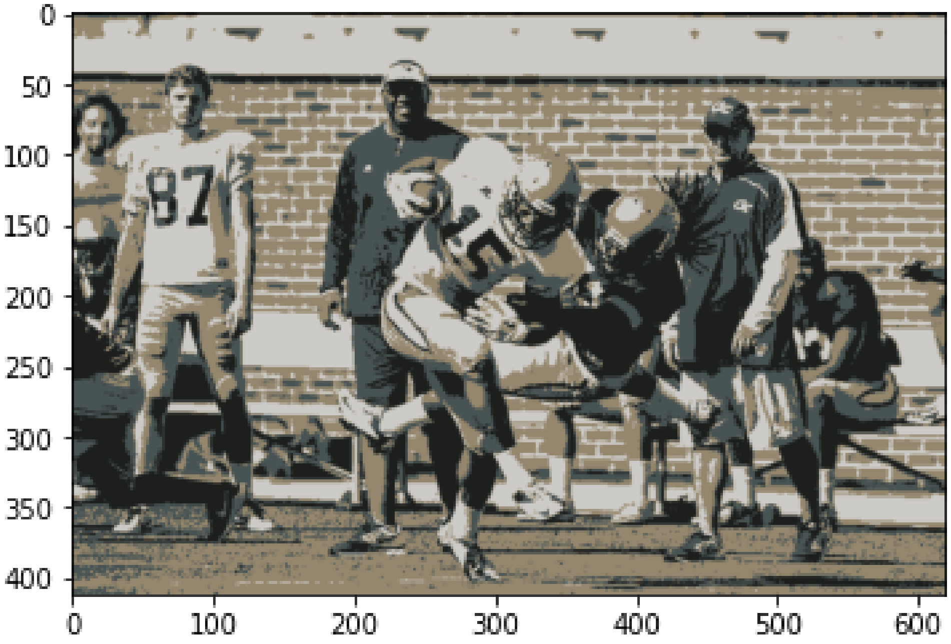
```
grassImage = "./data/grass.bmp"
images.append(footballImage)
images.append(beachImage)
images.append(grassImage)

for image in images:
    imgArray = convertImageToArray(image)
    print("printing the original image")
    convertArrayToImageAndDisplay(imgArray)
    for k in(2,4,8,16):
        run_k_means(k,imgArray)
```

```
printing the original image
```

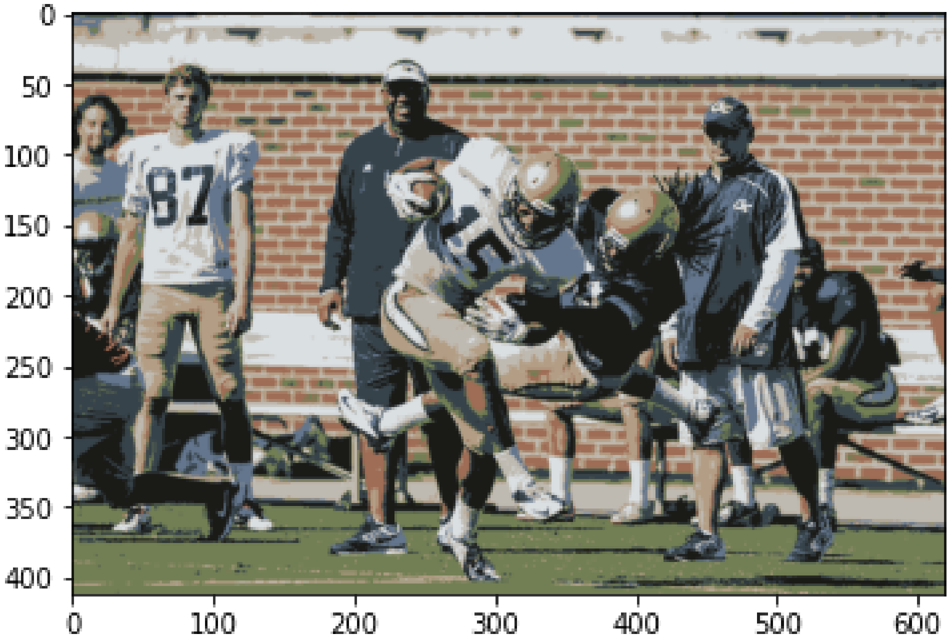
```
running for  2  centers
number of iterations for k =  2  is : 23
Runtime of the program for  2  centers: {73.56592011451721}
printing the  2  clustered image
```



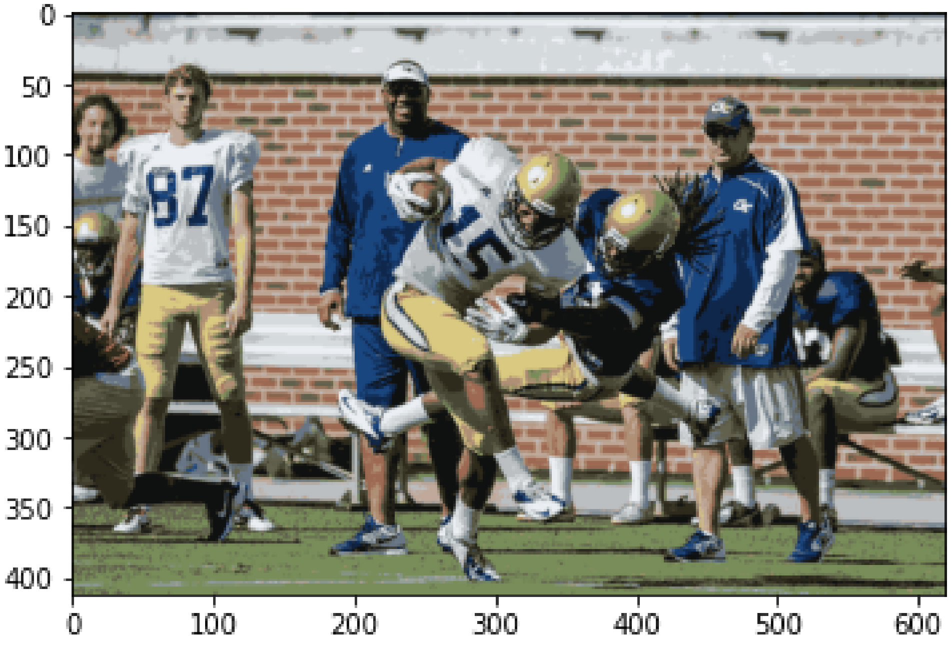
```
running for  4  centers
number of iterations for k =  4  is : 67
Runtime of the program for  4  centers: {200.61285710334778}
printing the  4  clustered image
```



```
running for  8  centers
number of iterations for k =  8  is : 151
Runtime of the program for  8  centers: {456.8332450389862}
printing the  8  clustered image
```



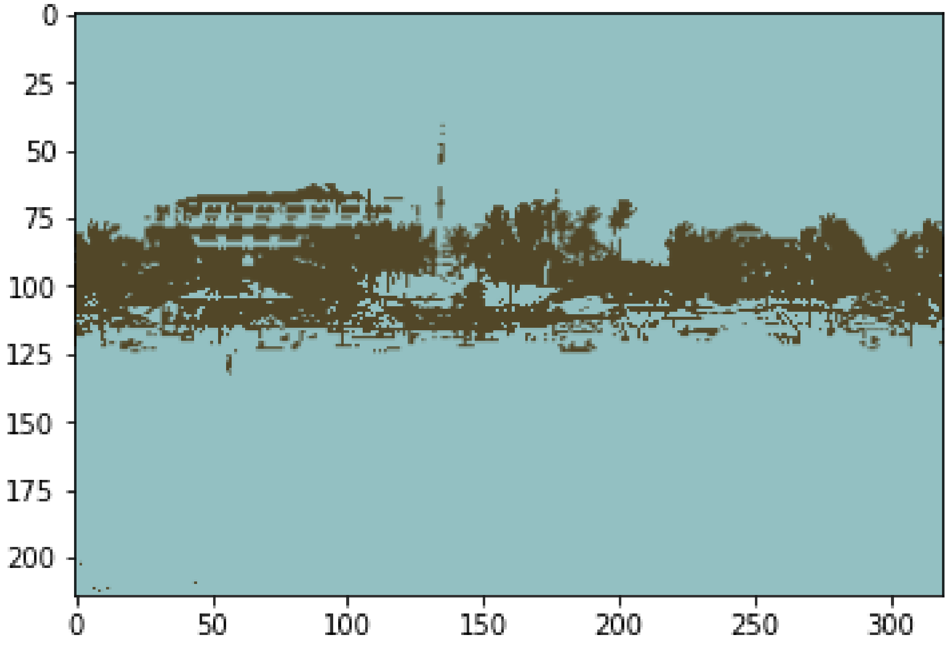
```
running for  16  centers
number of iterations for k =  16  is : 138
Runtime of the program for  16  centers: {417.2805349826813}
printing the  16  clustered image
```



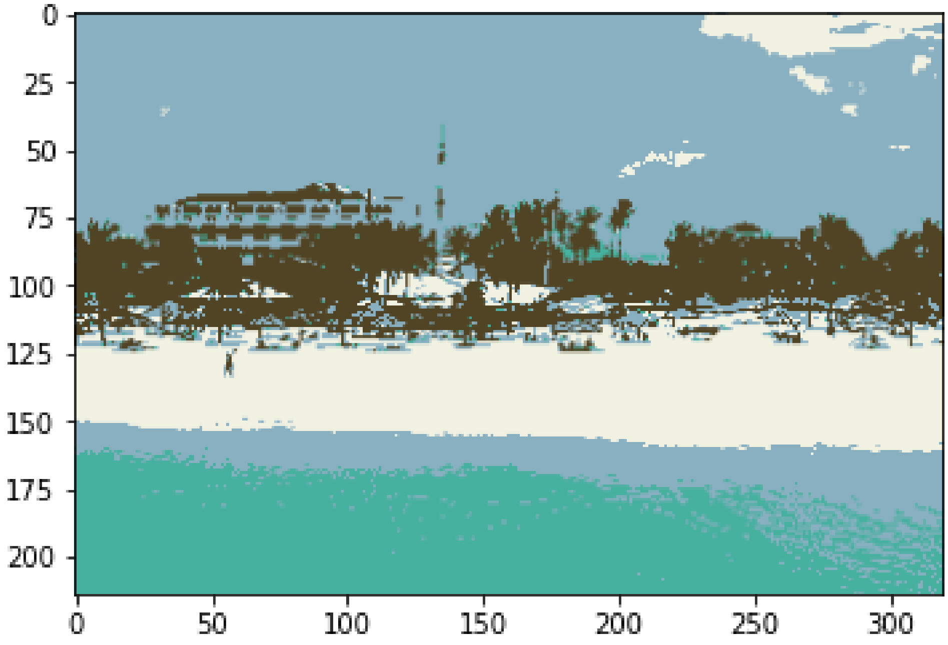
```
printing the original image
```

```
running for  2  centers
number of iterations for k =  2  is : 20
```

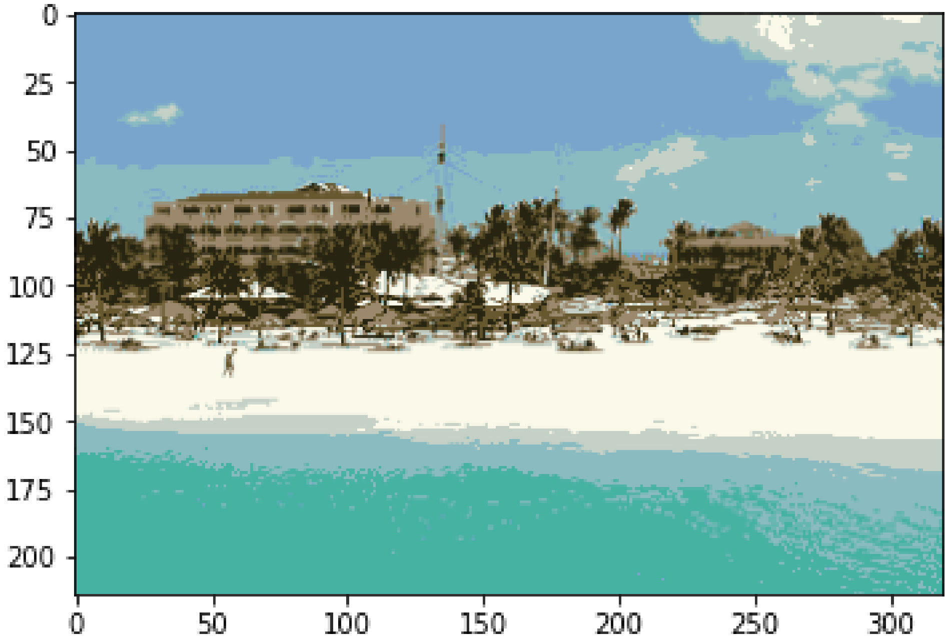
```
Runtime of the program for  2  centers: {14.50742220878601}
printing the  2  clustered image
```



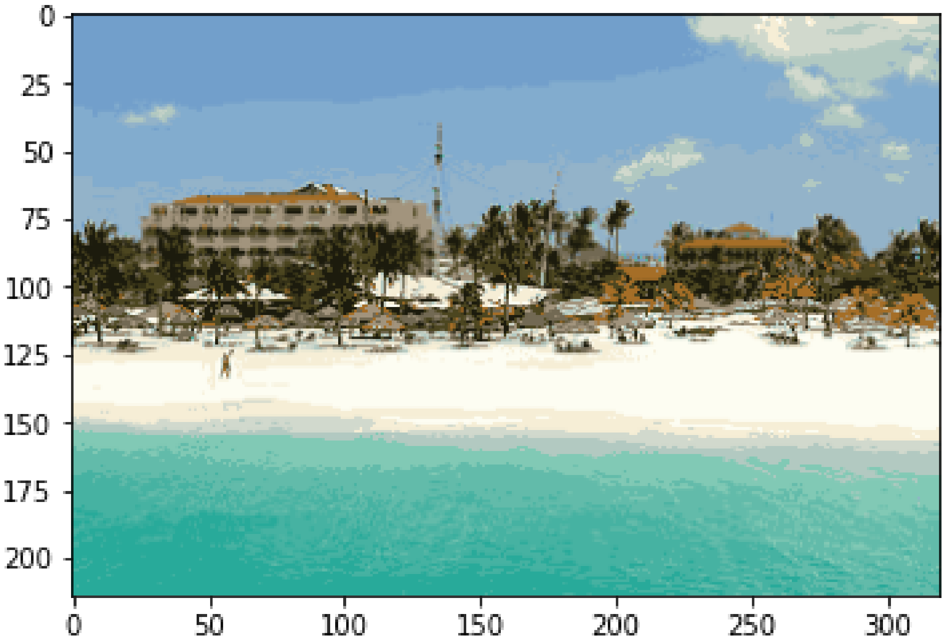
```
running for  4  centers
number of iterations for k =  4  is : 19
Runtime of the program for  4  centers: {14.014214754104614}
printing the  2  clustered image
```



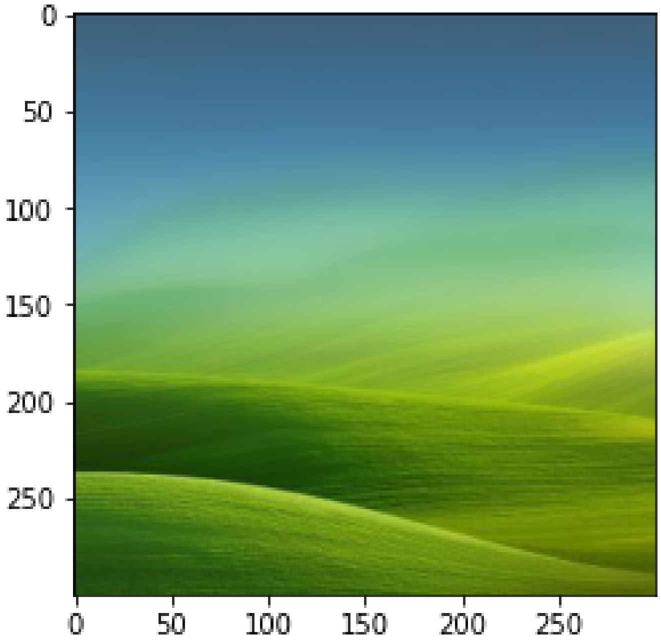
```
running for  8  centers
number of iterations for k =  8  is : 103
Runtime of the program for  8  centers: {2047.3852989673615}
printing the  8  clustered image
```



```
running for  16  centers
number of iterations for k =  16  is : 177
Runtime of the program for  16  centers: {306.1026077270508}
printing the  16  clustered image
```

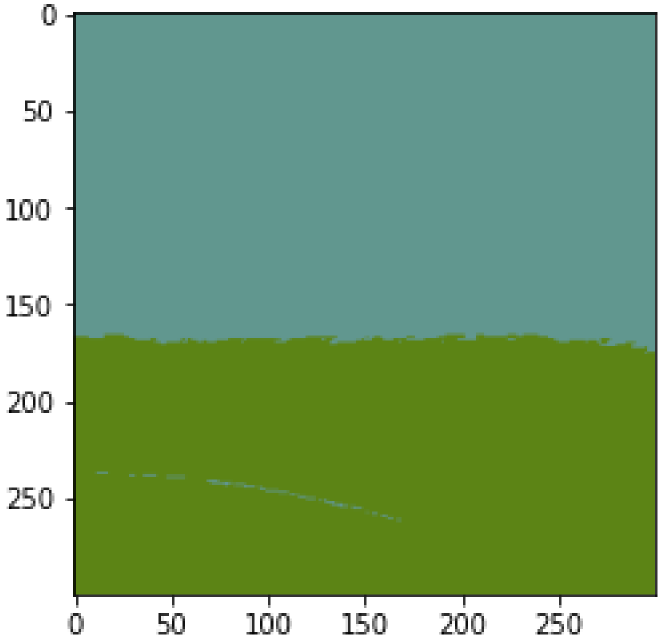


```
printing the original image
```

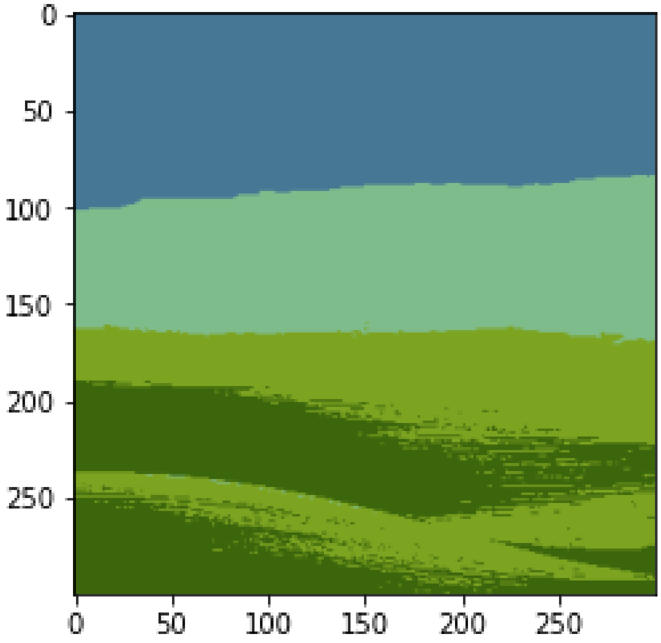


```
running for  2  centers
number of iterations for k =  2  is : 10
```

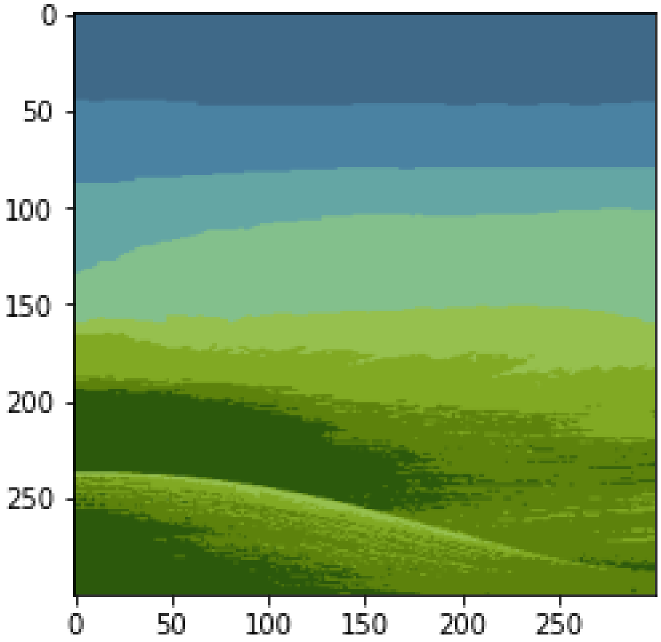
```
Runtime of the program for  2  centers: {9.050542831420898}
printing the  2  clustered image
```



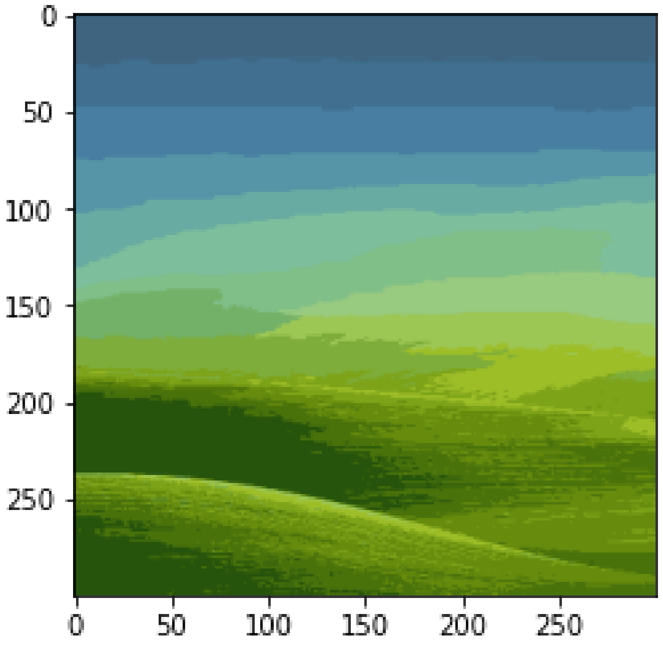
```
running for  4  centers
number of iterations for k =  4  is : 25
Runtime of the program for  4  centers: {24.84894609451294}
printing the  4  clustered image
```



```
running for  8  centers
number of iterations for k =  8  is : 54
Runtime of the program for  8  centers: {53.792957067489624}
printing the  8  clustered image
```



```
running for  16  centers
number of iterations for k =  16  is : 73
Runtime of the program for  16  centers: {77.03693890571594}
printing the  16  clustered image
```



0.0.3 Question 2.2: Run your k-means implementation with different initialization centroids.

0.0.4 Answer:

I have run the logic wiht 4 different centroid points for K=4. I have picked points across the spread:

	Centroids	Iterations	Time-Taken(s)
1	[0,0,0],[255,255,255], [100,100,100],[200,200,200]	27	75.86
2	[109,109,109],[110,110,110], [111,111,111],[112,112,112]	37	105.15
3	[0,1,2],[2,1,0], [244,0,0],[255,0,0]	30	84.51
4	[1,2,3],[4,5,6], [7,8,9],[10,11,12]	75	221.35

- 1. centroids spaced almost equally and spread from 0 to 255
- 2. All 4 centroids very close to each other but in the middle of the spectrum (109-114).
- 3. 2 centroid close to each other at the beginnin of spectrum and other 2 at the end of the spectrum.
- 4. All 4 centroids very close to each other but at the beginning of the spectrum (1-12).

Observations:

- Irrespective of centroids I pick, the resulting convergence and hence the picture seems to be same.
- The first centroid that has 4 pts spread accross the spectrum converges faster. In around 27 iterations

- The second centroid set that has all 4 points very close to each other, in the middle of the spectrum, takes longer to converge. In around 37 iterations.
- The third set has 2 centroids as at the beginning and 2 centroids at the end of the spectrum, takes longer than 1st set of centroid but faster than 2nd set of centroids
- The 4th set which has all centroids close to each other at one end of spectrum takes longest.

Based on the observations, if the initial centroids are not close to each other and spread across the spectrum, it will result in convergence faster. This is observation is purely based on the data above. I have run this program many times and consistently got similar results.

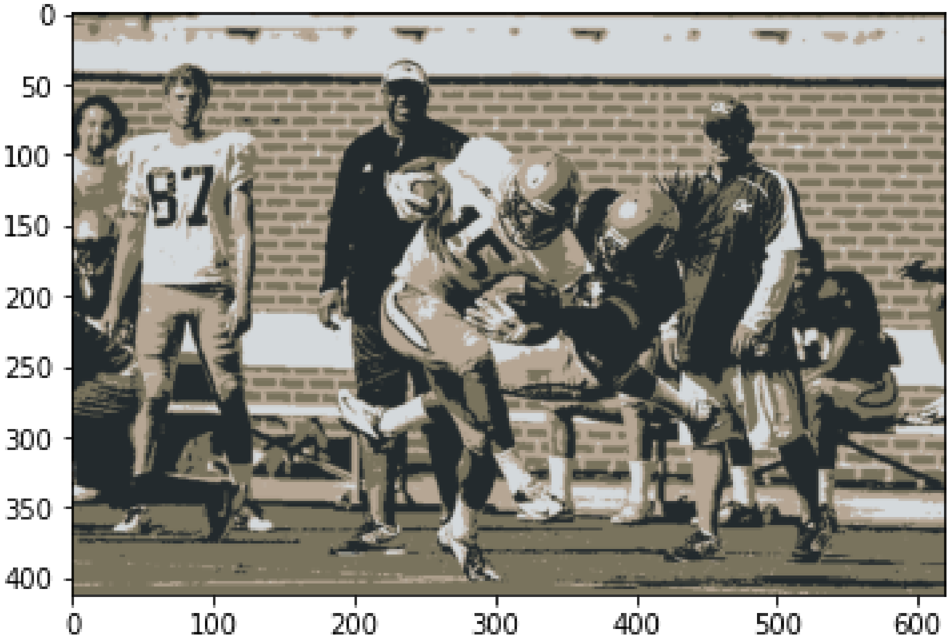
[403]:

```python
## Question 2.2: Run your k-means implementation with different initialization centroids.

img_arr2=read_img("./data/football.bmp")
centers=[]
centers.append(np.array([[0,0,0],[255,255,255],[100,100,100],[200,200,200]]))
centers.append(np.array([[109,109,109],[110,110,110],[111,111,111],[112,112,112]]))
centers.append(np.array([[0,1,2],[2,1,0],[244,0,0],[255,0,0]]))
centers.append(np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]]))
for center in centers:
    print("starting with center:")
    display(center)
    run_k_means(4,img_arr2,center)
```

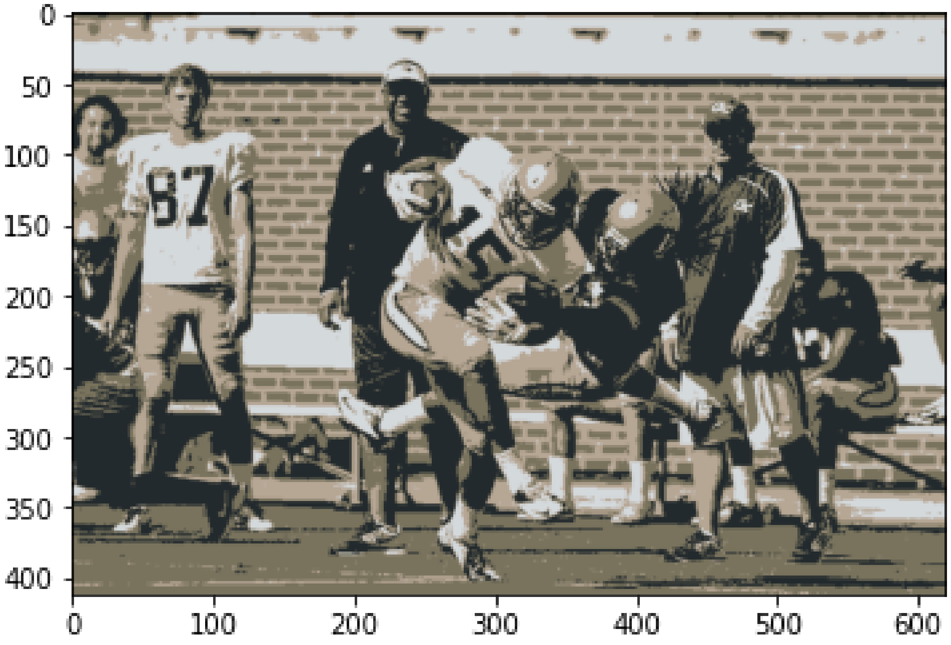
```
starting with center:
```

```
array([[  0,   0,   0],
       [255, 255, 255],
       [100, 100, 100],
       [200, 200, 200]])
```

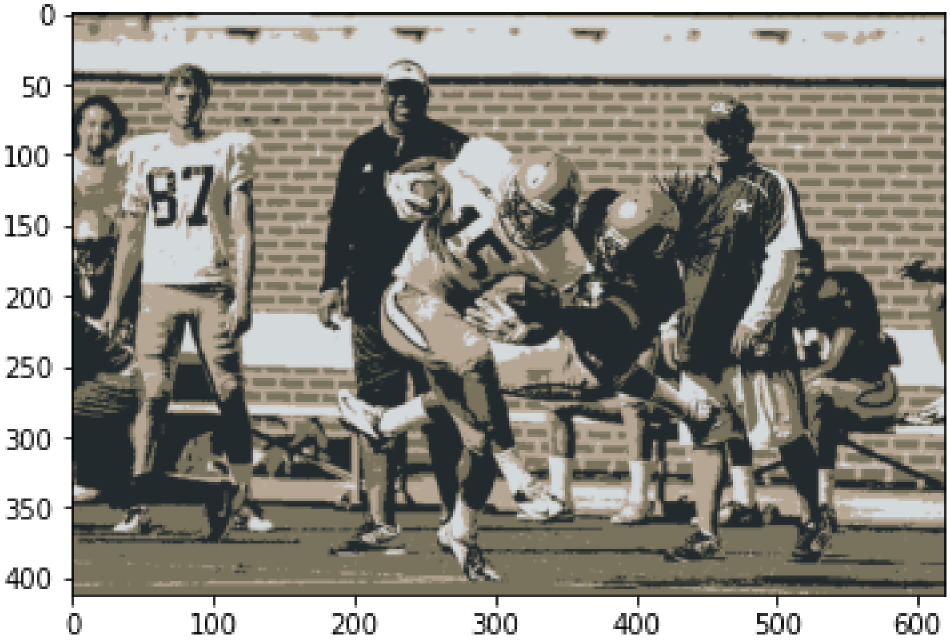
```
running for  4  centers
number of iterations for k =  4  is : 27
Runtime of the program for  4  centers: {75.86233496665955}
printing the  4  clustered image
```



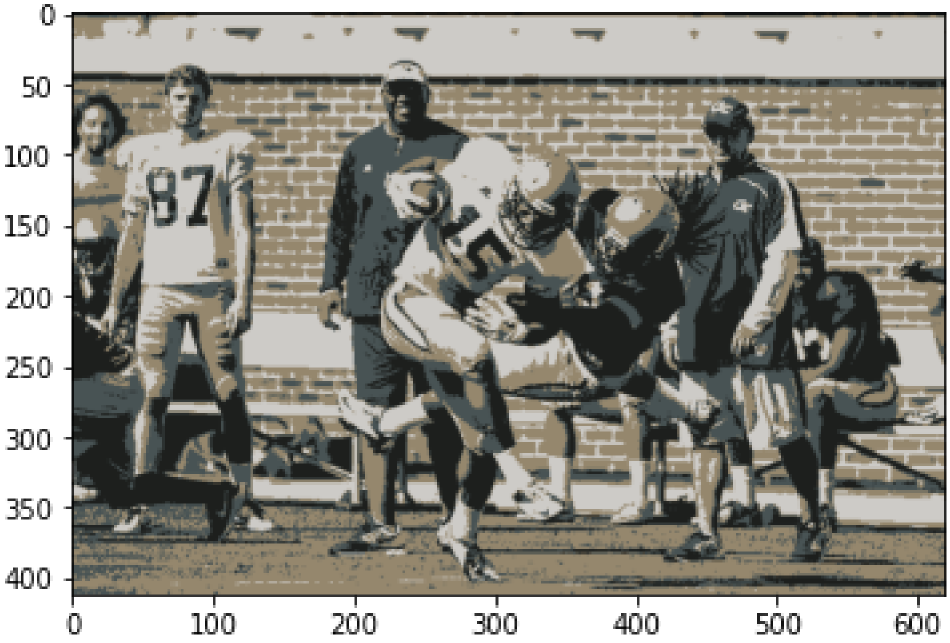
```
starting with center:

array([[109, 109, 109],
       [110, 110, 110],
       [111, 111, 111],
       [112, 112, 112]])


running for  4  centers
number of iterations for k =  4  is : 37
Runtime of the program for  4  centers: {105.15727591514587}
printing the  4  clustered image
```



```
starting with center:

array([[  0,   1,   2],
       [  2,   1,   0],
       [244,   0,   0],
       [255,   0,   0]])


running for  4  centers
number of iterations for k =  4  is : 30
Runtime of the program for  4  centers: {84.51232600212097}
printing the  4  clustered image
```



```
starting with center:

array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])


running for  4  centers
number of iterations for k =  4  is : 75
Runtime of the program for  4  centers: {221.35459089279175}
printing the  4  clustered image
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600