5. Programming: Image compression:

1. K-medoids algorithm explanation:

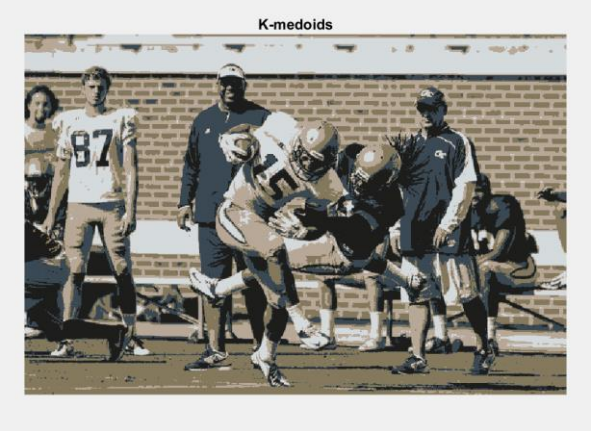
a. Cluster initialization: Instead of generating k 1x3 vectors with random values ranging from 0 to 255 like I did for k-means, I generated k random integers ranging from 1 to the length of the *pixels* vector. I then indexed *pixels* with those random values to guarantee that the initial cluster centers were actual points from *pixels*.

b. Cluster center recalculation: Similarly to what I did for k-means, after assigning each point to its closest cluster center, I calculated the average of all points in each cluster. However, instead of simply assigning the averages as the values of the cluster centers, I looped through each cluster to find which existing point was closest to that cluster's average. I then assigned those points as the new cluster centers. I kept repeating the process of assigning points to clusters and re-determining the cluster centers until no cluster centers changed (this was achieved using a while loop).

c. Distance measure choice: Initially, I used the same distance measure used for k-means (Euclidian or norm 2). I then tried using the norm 1 distance measure, also known as the Manhattan distance. This measure did not result in any considerable difference in speed or quality of generated images. Thus, I decided to go back to using the Euclidian distance measure.

Here are the outputs of test runs with each distance measure using the *football.bmp* image and k=5:

Euclidian Distance:

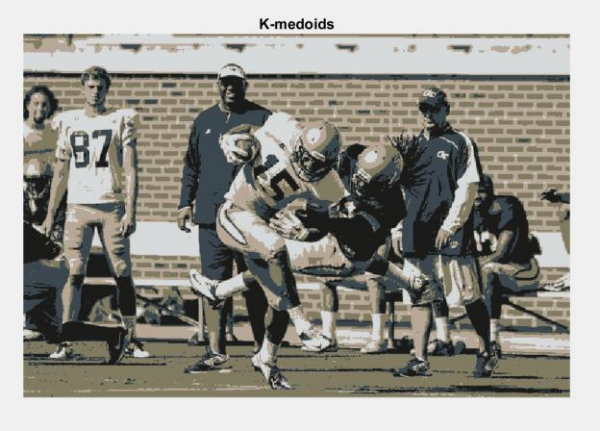


Running time: 15.6534

# of iterations: 9

Time per iteration: 1.7392

Manhattan Distance:



Running time: 21.9661

# of iterations: 14

Time per iteration: 1.5690

2. Custom picture:

For the custom picture, I chose a 320X240 jpg picture showing the congress building in Brasilia, the capital of Brazil and city where I grew up.

3/5. K-means and K-medoids test runs:

I decided to make test runs using both the *football.bmp* image and my custom image *congresso.jpg*. I chose the k-values of 2, 3, 5, 10, 20 and 30. Here are the results:

- k-means for *football.bmp*:

	k = 2	**k = 3**	**k = 5**	**k = 10**	**k = 20**	**k = 30**
Running Time (s):	27.7268	29.2702	49.2549	149.9746	221.078	468.067
# of Iterations:	20	20	33	93	120	228
Time per Iteration:	1.3863	1.4635	1.4925	1.6126	1.8423	2.053

- k-medoids for *football.jpg*:

	k = 2	**k = 3**	**k = 5**	**k = 10**	**k = 20**	**k = 30**
Running Time (s):	15.102	13.2906	24.7218	56.7587	46.6615	58.3808
# of Iterations:	9	8	14	31	23	26
Time per Iteration:	1.6779	1.6613	1.7658	1.8309	2.0287	2.245

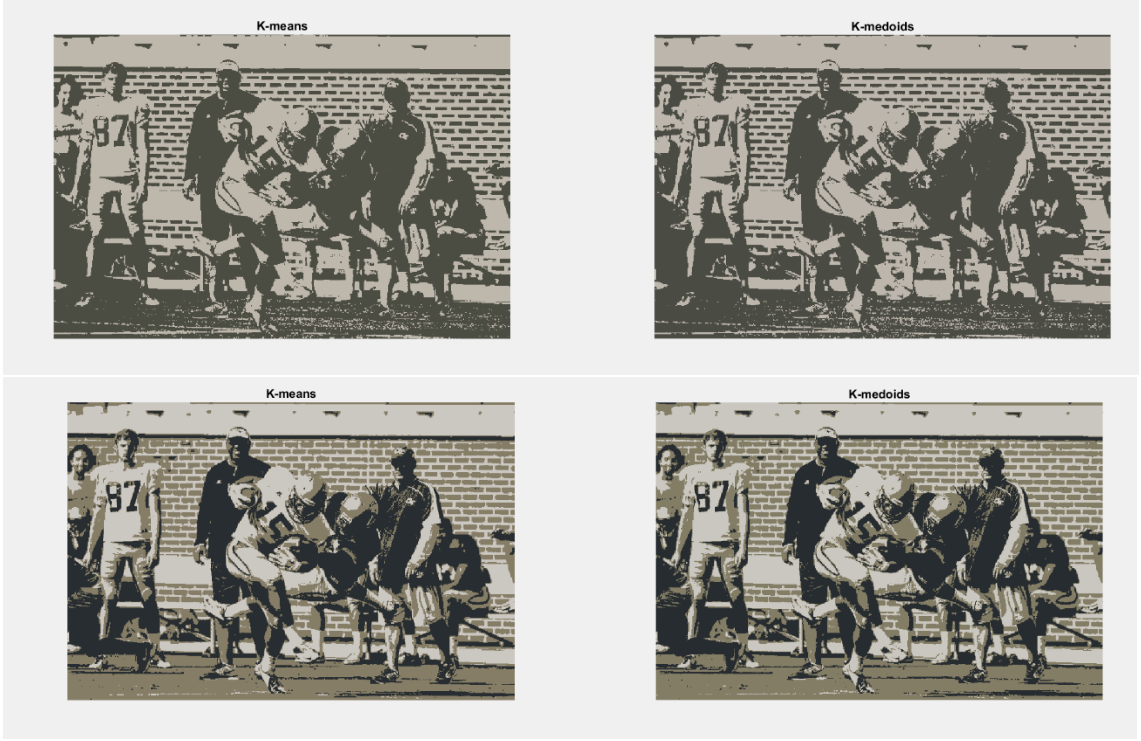
- k-means for *congresso.bmp*:

	k = 2	**k = 3**	**k = 5**	**k = 10**	**k = 20**	**k = 30**
Running Time (s):	3.899	4.6601	5.7614	40.0338	48.1153	50.0478
# of Iterations:	9	11	13	85	90	84
Time per Iteration:	0.4333	0.4326	0.4432	0.471	0.5346	0.5958

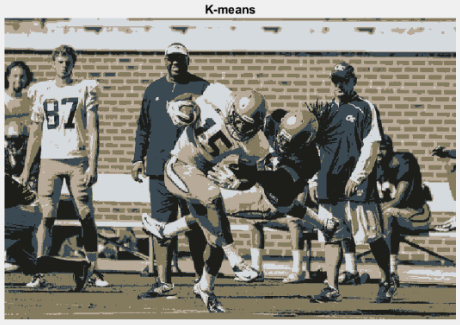
- k-medoids for *congresso.jpg*:

	k = 2	k = 3	k = 5	k = 10	k = 20	k = 30
Running Time (s):	2.6436	5.4095	4.6924	6.981	13.2278	11.3847
# of Iterations:	5	11	9	13	22	17
Time per Iteration:	0.5286	0.4917	0.5213	0.537	0.6012	0.6697

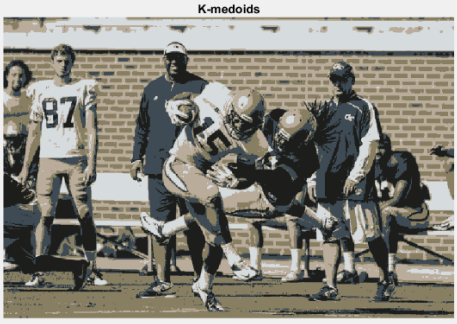
- *football.bmp* output from k = 2 through k = 30



K-means

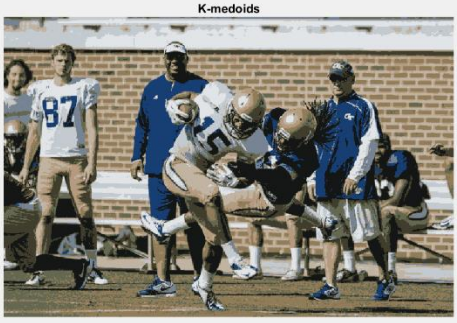


K-medoids



K-means

K-medoids



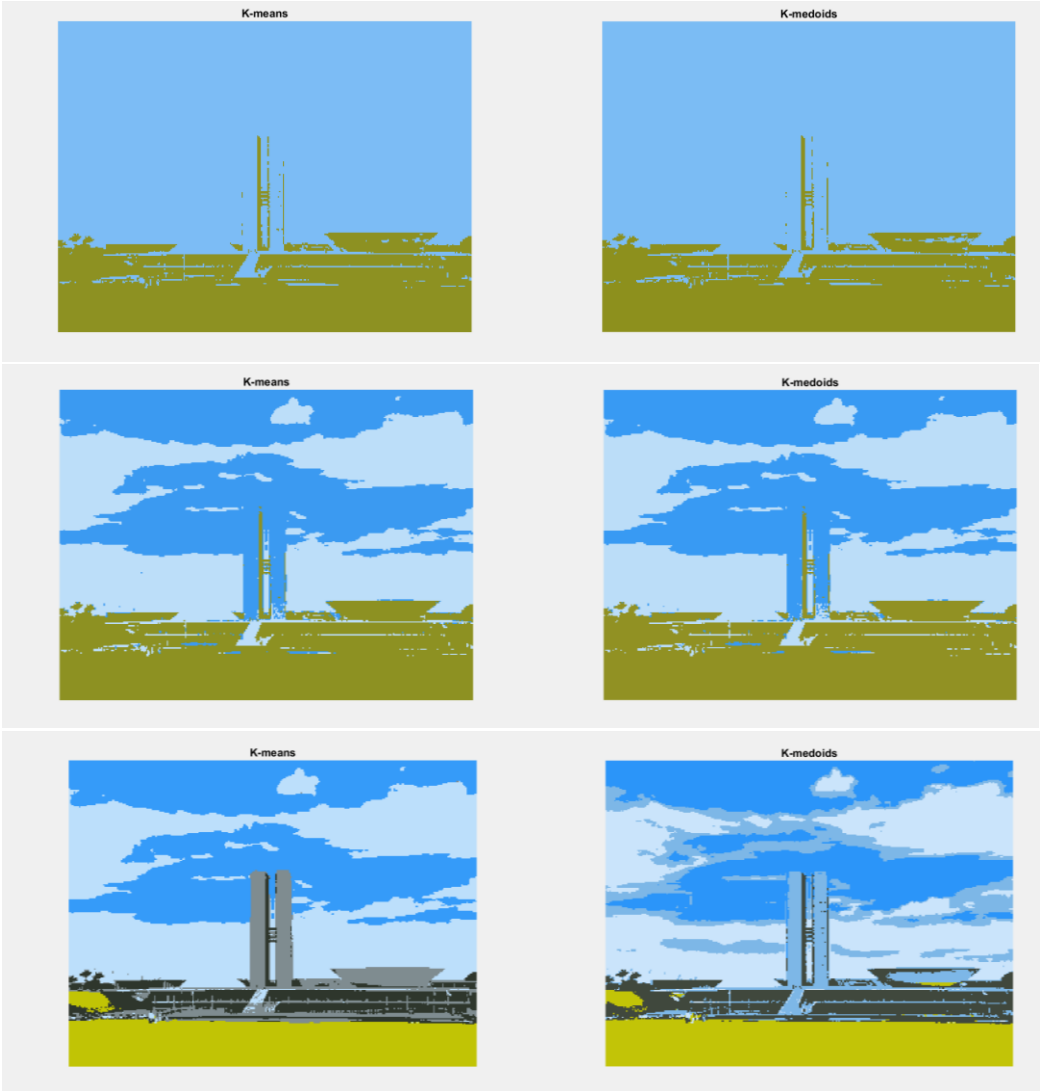
K-means

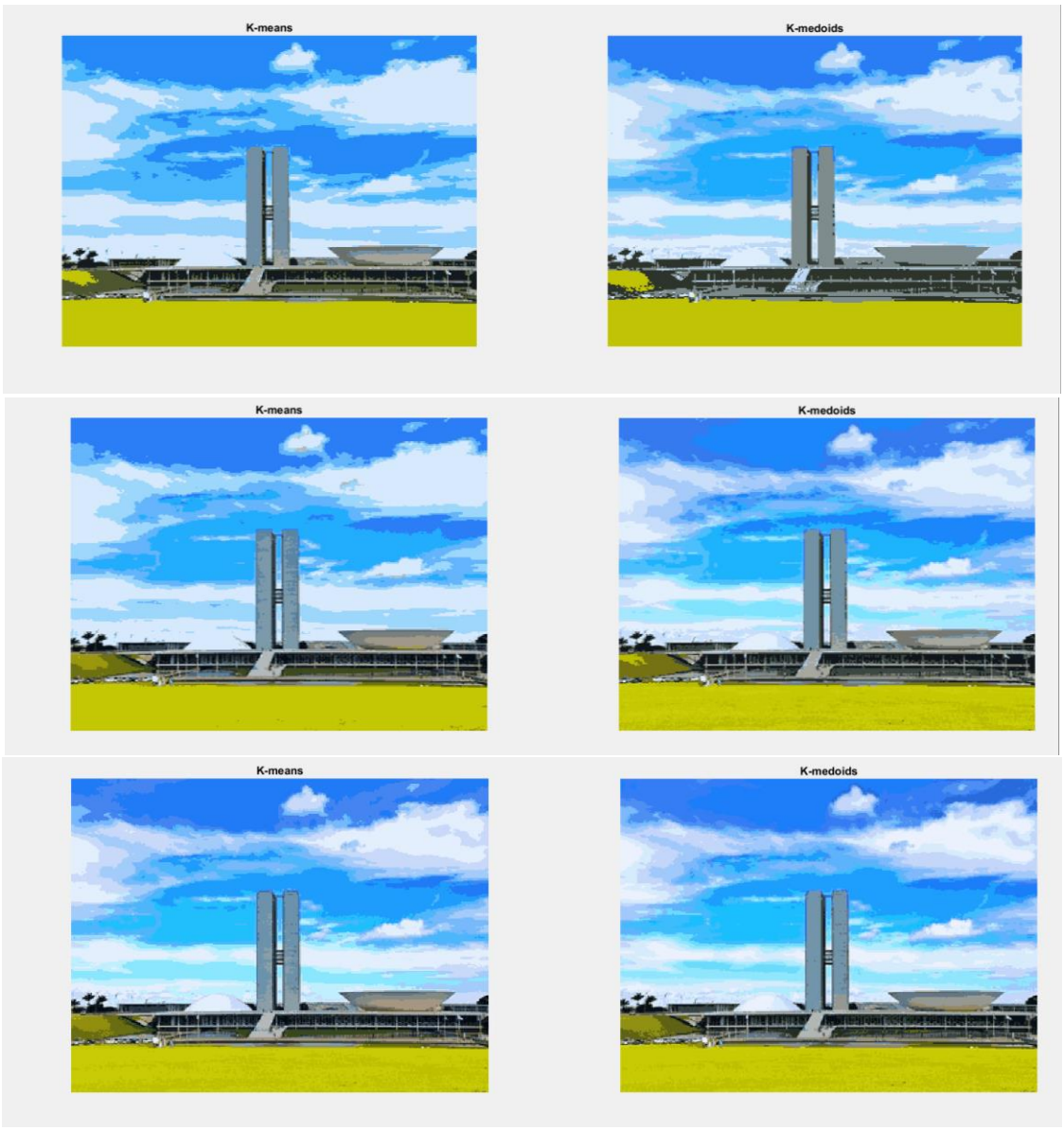
K-medoids

K-means

K-medoids

- *congresso.jpg* output from k = 2 through k = 30





Regardless of the variance between trials containing the same value of k due to random initialization, it is evident that both the convergence time and total number of iterations increases with higher numbers of K. The average time per iteration also increases, but at a slower pace. It was also very evident that the k-medoids algorithm took considerably less iterations to converge than the k-means one. However, the k-medoids took slightly more time per iteration when compared to k-means.

4. Different initial centroids:

The initial random initialization of centroids makes a huge difference in the performance of the k-medoids algorithm. The difference is in the total number of iterations, and, consequently, the total running time. When intentionally choosing poor initial centroids, it was evident that the algorithm needed considerably more time to converge.