Lab 4: Image Segmentation through K-M eans

Submission Deadline: Wednesday (5 October 2022) at 11:59 pm

The goal of image segmentation is to partition an image into regions, each of which have a reasonably homogenous visual appearance or which corresponds to similar objects or parts of an object. Each pixel in an image is a point in a 3-dimensional space comprising of the intensities of the red, blue, and green channels of color. This segmentation algorithm simply treats each pixel in the image as a separate data point, and we calculate some set of means – or centroids – from the distribution of points contained in the image. We then assign each data point to its closest centroid, and redraw the image using only the color vectors from the centroids.

We will illustrate the result of running K-means for many values of K, by re-drawing the image replacing each pixel with the (R,G,B) triplet of the centroid that pixel has been assigned to. Results for various values of K based on example elephant.jpg are shown as follow.

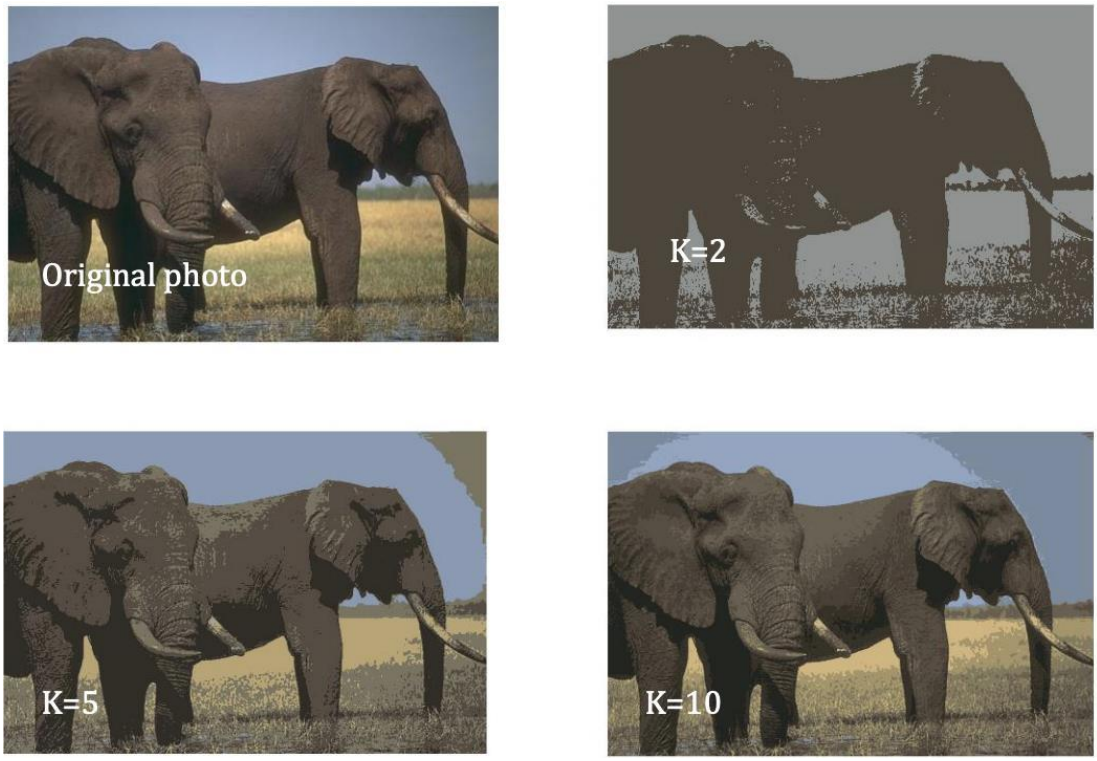

Figure 1. Segmentation of the image into different segments based on K-Means algorithm

Data

You have two images — elephant.jpg and eiffel.jpg, on which you will be running your K-Means code.

Implementing K-Means

To perform K-Means, we want to find K "centroid" vectors, and will redraw the image using these vectors.

1. Create a KMeans function `kmeans( image, k, max_iter)` to implement all of these steps. This function should output 1) The final set of coordinates of the K centroids, *and* 2) the final segmented image based on the k centroids.
2. Vectorize the image based on RGB components
3. Initialize the centroids. Randomly select RGB points from the image as initial centroids.
4. Assign all data points to the closest centroid. Use the L1 norm of the difference between centroid and pixel (R,G,B) values as your distance metric.
5. Re-compute the centroid vectors as the mean of all pixels assigned to each respective centroid.
6. Repeat steps 4 and 5 until the stop criteria is reached.
7. Stop Criteria: The centroids do not change in step 5 (or) the maximum number of iterations have been reached
8. Re-segment the input image. Replace each pixel value with the value of the closest centroid.

Validation on Test Images

9. Load elephant.jpg and eiffel.jpg images.
10. Run K-Means with K = 2, K = 5, and K = 10 on both of the images.
11. Generate two figures, one for each image, like Figure 1. Each figure should have four subplots, one with the original image and also the three segmented images. The value of K should be indicated somewhere in the figure.

Comparison with Existing K-Means

Now use an existing K-Means function and compare the output for elephant.jpg.

12. Run an existing K-Means function with K=5.
13. Generate one figure with the original image, segmented image from your implementation with K=5, and the segmented image from the previous step with K=5

Code Optimization (optional)

14. Using elephant.jpg, print the times it takes your code to run K-Means and the Existing implementation to run with K=5.
15. Try optimizing your K-Means code to generate the segmented image within the order of a few seconds. Matrix operations are your friend.

Deliverables:

If you use Python, use the provided Jupyter Notebook template for all problems, which is also available at following link:

https://colab.research.google.com/drive/1n3QW690QA4jGiT5vmpg2uY2ddDIFTPcb?usp=sharing

MATLAB: A completed lab_5.m or lab_5.mlx source file, and the data files you used combined in a zip file named Lab5_FirstnameLastnaem_JHID.zip

Python: Lab5_FirstnameLastname_JHID.ipynb source file, and the data files you used combined in a zip file named Lab5_FirstnameLastnaem_JHID.zip