Short Answers:

1. The result is invariant to orientations. Images can contain multiple texture patterns with various intensities due to differences in lighting and various scales and orientations due to different perspectives; hence, presenting a texture method being invariant to texture scale and orientation. Texture feature vectors are extracted using Gabor filters, which transforms the feature vectors into a scale and orientation invariant texture space while comparing the texture similarities within the texture space and refining texture rankings via the original Gabor feature vectors.
2. The dots on this graph appear to be uniformly distributed, which means within a circle, the Euclidean distance for each point is the same. Since k=2, the outcome would likely just be a line that cut through the focal point of these two circles. The algorithm may produce different results on each iteration (hence, unexpected result with each iteration), but the results would practically be the same with different orientations. By human eyes, we might categorize the dots into two groups as two circles with different radius but the same focal point (or Gestalt Principles - connecting the dots into two circles); however, this form of categorization is practically drawing a circle between these two circles to separate them. K-means clustering is limited to drawing straight lines separating the max distance between clusters, but since the dots are uniformly distributed, a straight line drawing algorithm may not be able to classify the dots as humans do.
3. The graph cut method would be the most appropriate to use, because it has a feature extraction technique used for image analysis. Objects with a certain class of shapes are extracted by a voting procedure, which uses the parameter space from candidates with objects obtained by locating the local maximas, as known as the accumulator space.
4. **define** ***find_center_of_mass***(blob):
   for each pixel in *blo*b:
   Position_sum = pixel.position()
   CoM = Position_sum/blob.size()
   return CoM

   define double_circularity(blob):
   Mean_radius = blob.size()
   Blob_boundary = blob.boundary()
   for each pixel in *Blob_boundary*:
   sq_D_sum = CoM - pixel.position()
   Circularity = sq_D_sm/blob.size()
   return Circularity

   define k_mean_cluster (Circularity[], k):
   return kmeans = Circularity

   Variables:

   - Position_sum - $\sum_{pixels} pixel.position()$

- CoM - center of mass for the blobs
- sq_D_sum - sum of squared distance, $\sum_{pixels} pixel.position() - CoM.position()$
- Blob_boundary - pixels at the boundary of the blob
- Circularity - the feature which is invariant to radius and shape, because sq_D_sum is normalized with the total number of Blob_boundary

Algorithm:

- Find of the center of mass for the blobs
- Extract the circularity
- Cluster the blobs based on their circularities
- Fitting circles to the blobs (circular regression?? If that's a thing), and score circles by how well they fit the blobs.

Programming:

Part 1:

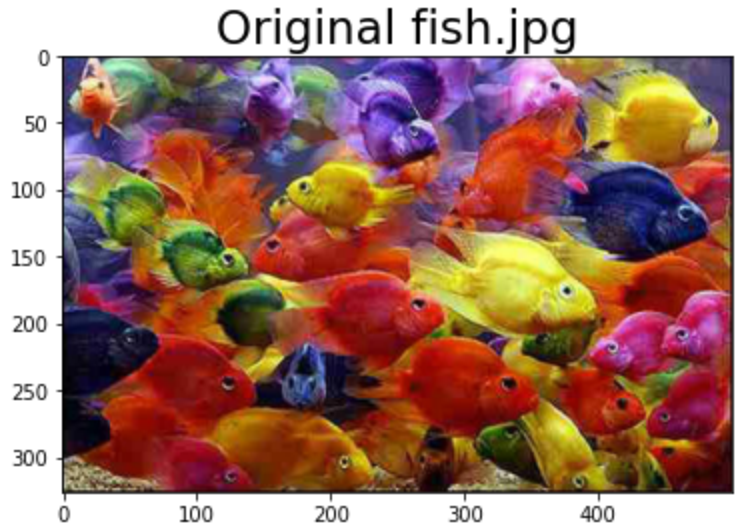


Fig. 1: the original image, aka fish.jpg

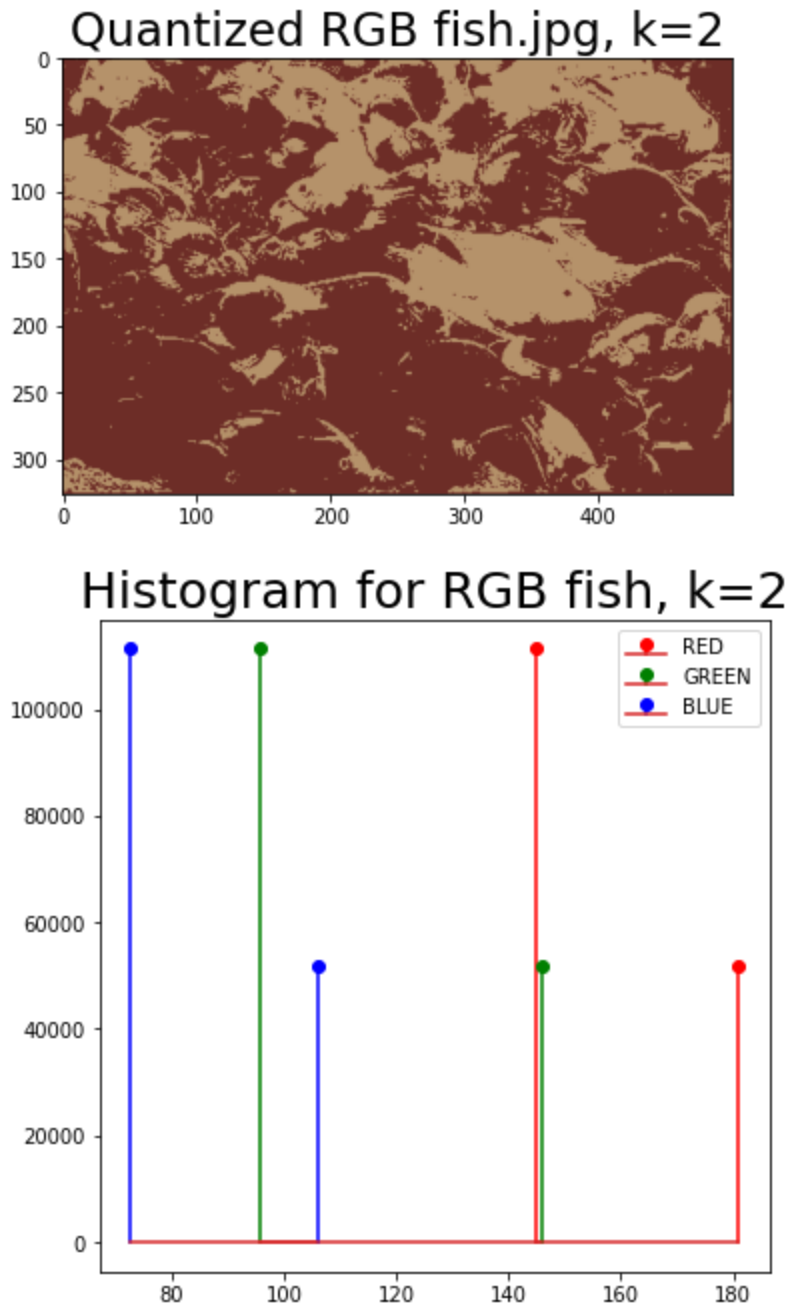


Fig. 2: quantized RGB fish with k = 2, SSD error is 108.45, with corresponding histogram

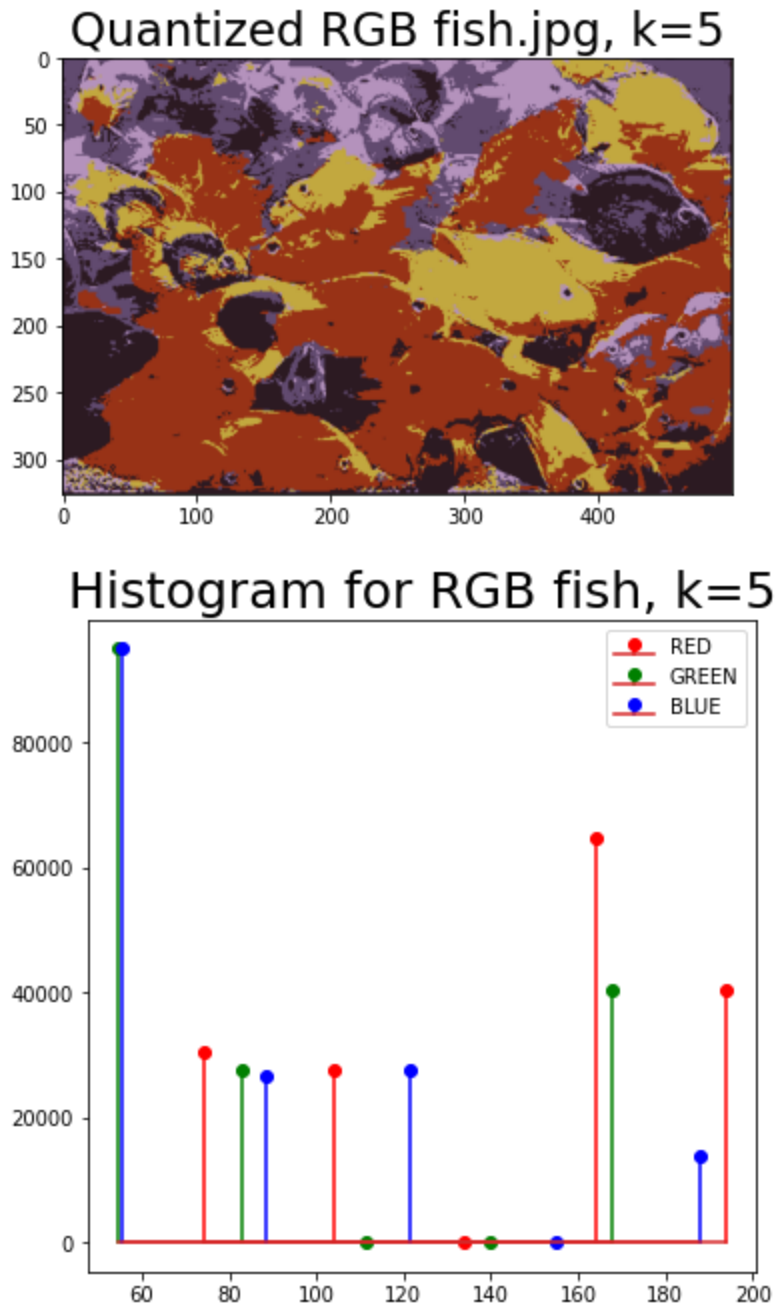


Fig. 3: quantized RGB fish with k = 5, SSD error is 102.398, with corresponding histogram

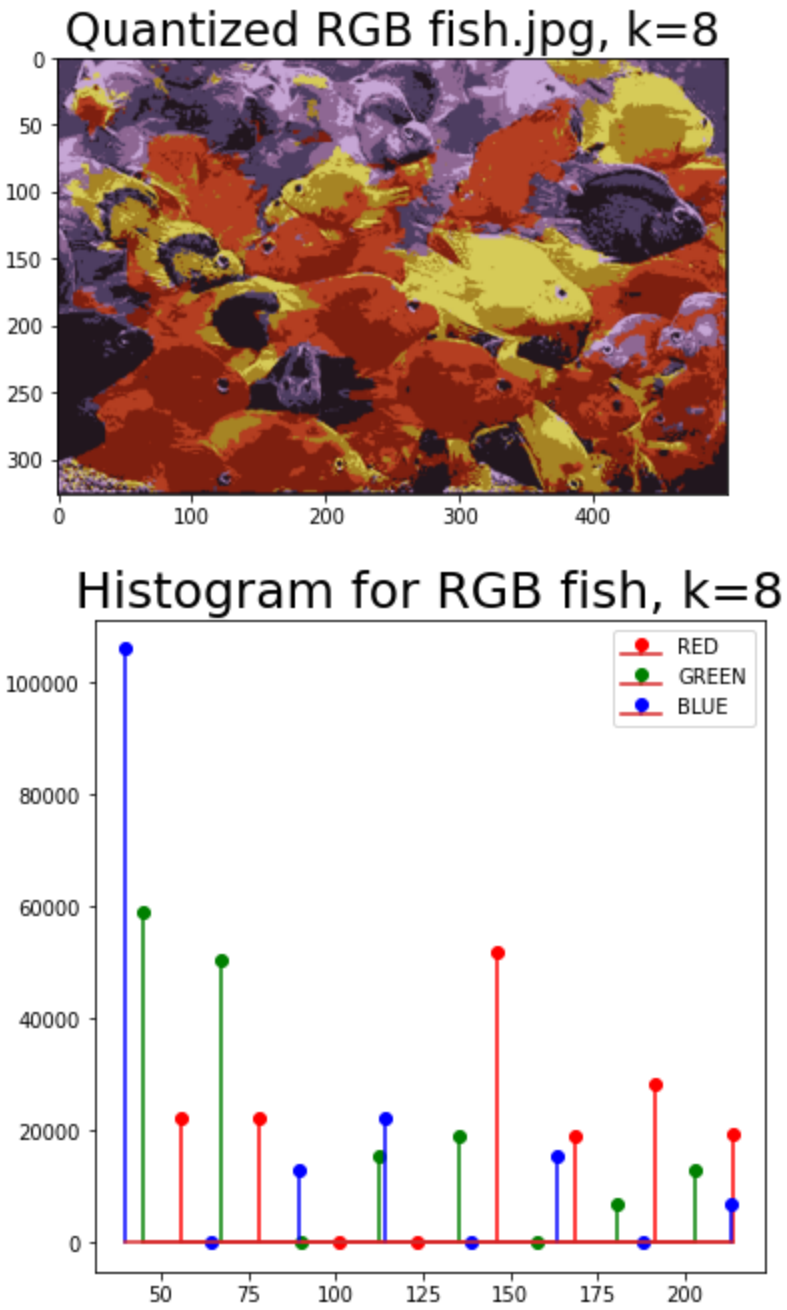


Fig. 4: quantized RGB fish with k = 5, SSD error is 97.14, with corresponding histogram

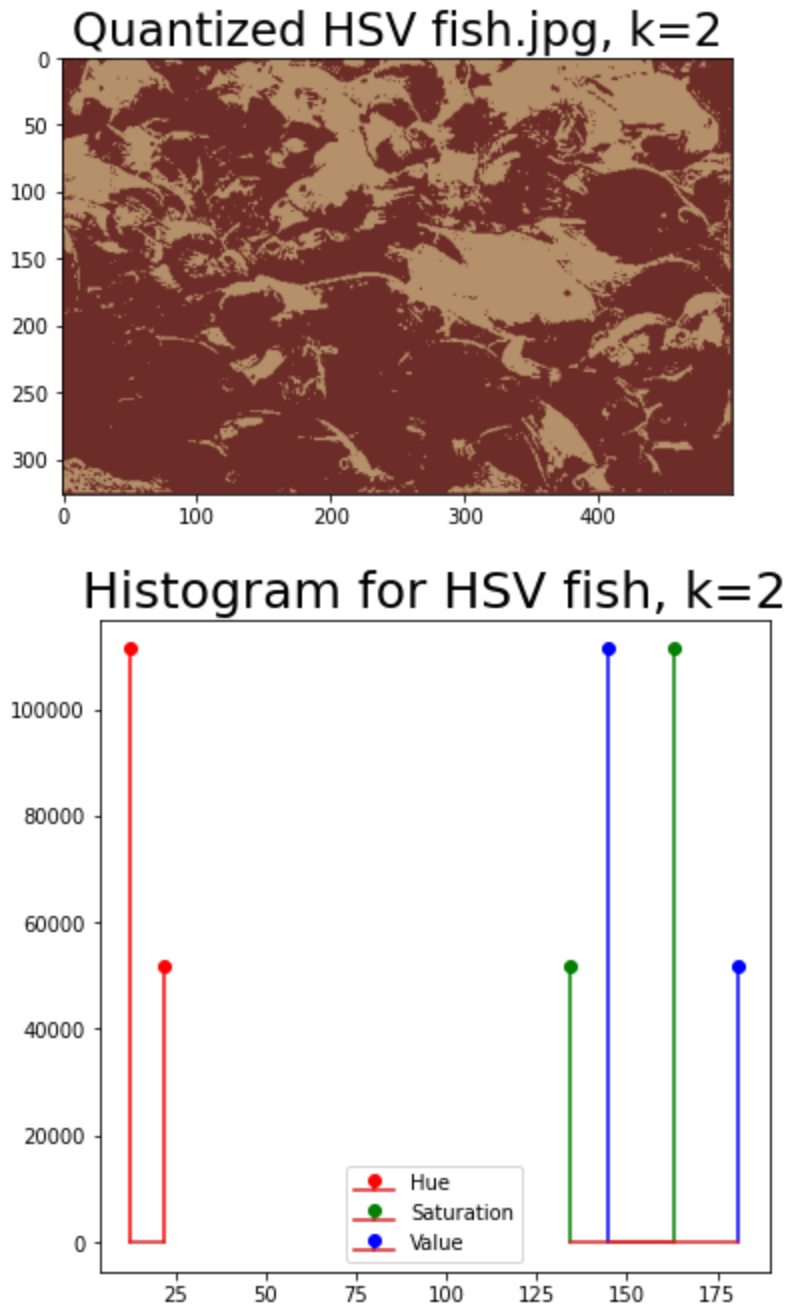


Fig. 5: quantized HSV fish with k = 2, SSD error is 0, with corresponding histogram

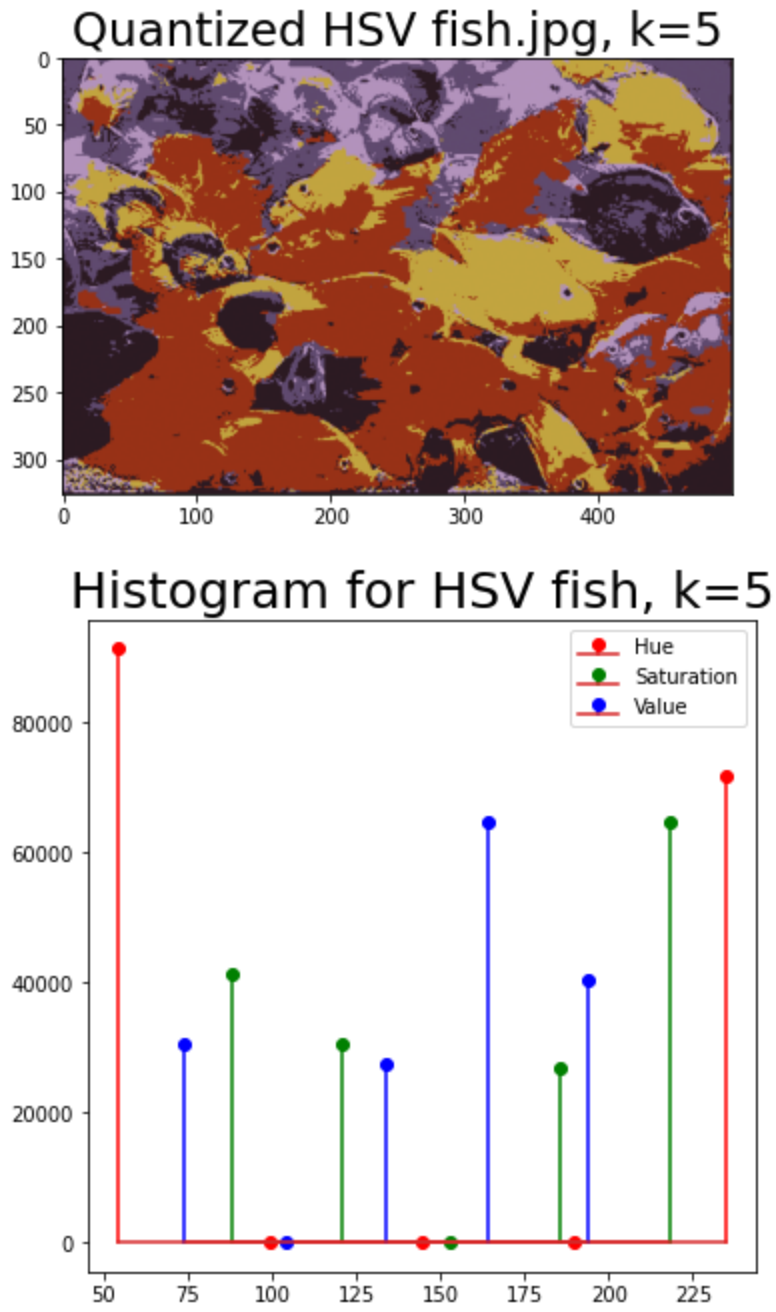


Fig. 6: quantized HSV fish with k = 5, SSD error is 0.201, with corresponding histogram

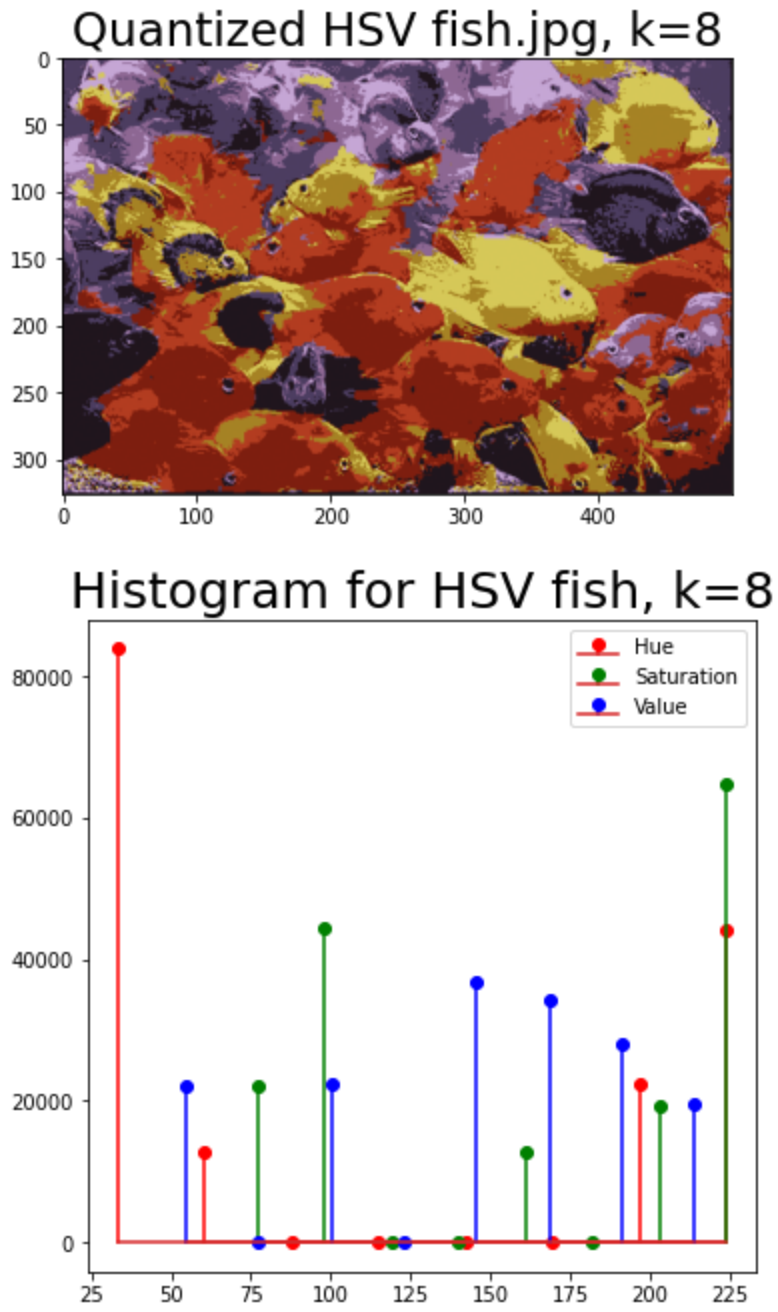


Fig. 7: quantized HSV fish with k = 5, SSD error is 0, with corresponding histogram

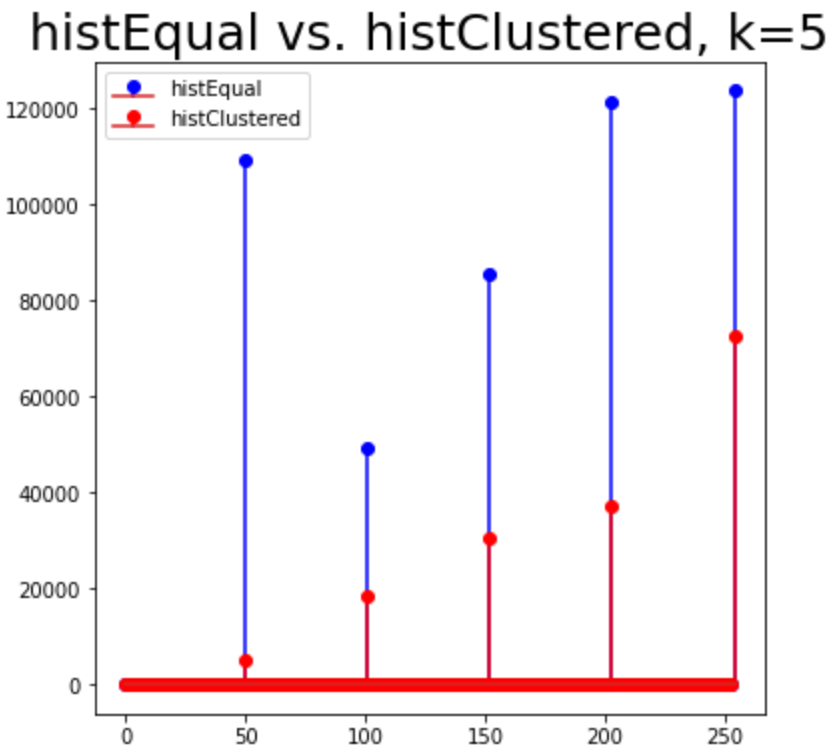


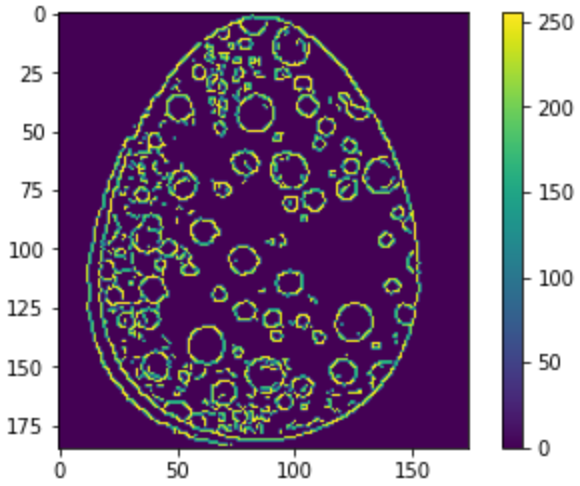
Fig. 8: histogram showing histEqual vs. histClustered

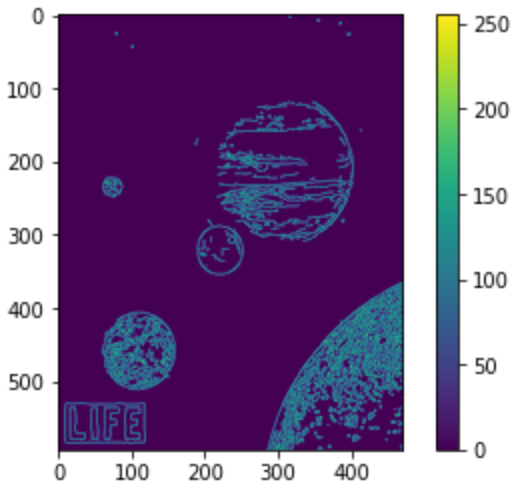
Color quantization is a technique that enables us to represent the pixels with fewer bits to store images in smaller memory maps. For instance, 28 different colors can be mapped to n different colors. In this way each pixel can be represented with log2n bits, instead of 8. Although quantization saves memory space, it also introduces color reproduction error since each pixel will be represented by the centroid color of the cluster which it is assigned. As it can be assessed from the table, the error reduces as $n \rightarrow 255$. Thus, the trade-off between memory and the error should be tuned for each run by choosing the right number of clusters. The effect of n can be shown by the figures above. Along with k, the initialization has a significant effect on the overall performance of k-means algorithm. The initial step of k-means is to choose n random points inside the data set and start assigning the data to the closest cluster. For that reason, starting point both has an effect on the total number of iteration and the points that clusters converge.The difference between color spaces also changes the overall performance of the algorithm. For instance, as can be seen in the figures, the results of quantization of HSV and RGB color spaces display great differences. Even though n is the same for the both instances, the results greatly vary. The reason why that difference occurs is the algorithm is not used for each channel in the HSV image, the scales of mappings are different. This difference leads to better image quality on the HSV quantization and fewer artifacts. Moreover, since less computation is done in the HSV quantization process, it is expected to run faster than RGB clustering.

Part2:

a) - Data Reduction: converting image to grayscale to reduce the amount of data processed.
- Gradient calculation: For later use, calculate the gradient of the image.
- Calculate the array of cosine and sine (in case of not using gradients).
- Initialize Hough space (H(m∗n)) with same dimension of the input image. (a and b resolution is 1)
- Initialize the center and vote arrays in which the most voted centers and their votes will be stored.
- Detect edges with the cv2 Canny algorithm. I used Canny because of the fact that it guarantees that edges will be 1 pixel wide and I can tune the output by optimizing σ and thresholds. By doing so, I reduce the total number of data will be processed
- Edge points are stored in the vector to eliminate one for loop in the Hough transform
- Calculate a and b, delete the results out of the image plane. Since cosine and sine values are stored in the vector, the formulate turns into vector operation. The resulting a and b are also vector
- Increment cells in the hough space; when gradient angle is used, 2 cells are incremented, otherwise the number of affected cells in the hough space equals to cosine and sine vector length.
- Sort the Hough space with corresponding indices.
- Sort the n-strongest circle centers

b) After spending hours constructing my detectCircles, the function failed to produce the right result. It is currently 2:57 a.m 02/08/21. It could take hours more, or days more, to debug my function. Thus, keeping delaying the deadline would not benefit my grade anymore. The function is still included in the .ipynb file, and there are codes showing I made an effort trying to make the correct plots. Please give some credits for making the efforts. I am also attaching pots for the edges of Jupiter and egg.





c) Successfully constructing the Hough space is crucial for the Hough transformation. I suspect my function has a bug when constructing for the Hough space. The Hough space is closely correlated to the edge image. Edge detector must be correctly tuned; otherwise, the result would be heavily affected by noises. Hough transformation is robust for missing and misleading information thanks to the accumulated voting system. To successfully fit a circle with the given radius, each edge point votes all the possible circles to which it belongs with given radius. As the voting process continues, it can be inferable that the peaks of hough space relates to the circle centers.

d) Since my function was not successful, I can't really comment on this problem. I would do more research on how to construct the Hough Space. I speculate I missed some steps.

e) The bin size is a parameter which has a deep impact on the performance of the fitting. By reducing the bin size, the accuracy for locating the center would drop proportionally. .