CS 6476: Computer Vision
Fall 2019 PS2
HUILI HUANG

QUESTION1:

1. The result will not be sensitive to orientation. Since the filter bank contains two scales in six orientation, when we use it to processing the image, the vector of each pixel would contain the information of each filter. Thus, there would be a 12 dimensional matrix for the whole image. Thus, we would find the texture no matter its orientation.
2. When we see Figure 2 , It is clear that the circle inside is one cluster and the bigger one is another cluster. However, it will be hard for K-means to figure it out. In fact, it will return 2 cluster that is symmetric to the center of the circle like Figure 1.

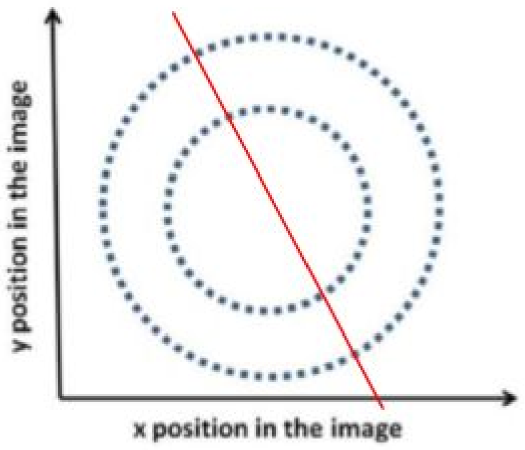

Figure 1: Keans result

3. **Kmeans** may not be a suitable choice for this task, since the input of this algorithm is data sets and label while the hough transform want to find the place of the the maximum vote.
   Mean-shift may help us because it will find multiple nodes that has local maxima of density in the space, which can be useful when we find the maximum votes.
   Graph-cuts divide the image into segments by delete the easiest breaklinks that cross between segments, which is also not helpful for us to collect votes.
4. There are several blobs we need to group. First, we find the center_mass of each blob and then we use Kmeans operator to divide the center_mass into different group. At the end we give the blobs in the same cluster the same color and plot the image. The pseudocode as follows:

```
#compute the center of mass
def find_center_mass(blob):
  sum=0
  for pixel in blob:
     sum=position(pixel)+sum
     center_mass=sum/(size of blob)
  return center_mass
def main():
  for blob in blob_set:
     center[]=find_center_mass(blob)
  kmeans(center[],k);
  if the label are the same:
       plot the area in the same color.
```

QUESTION2:
Color quantization with k-means:
(e):We use k=3,8,13 to run all the function in question (a) to (d). The results as follows:

1. k=3

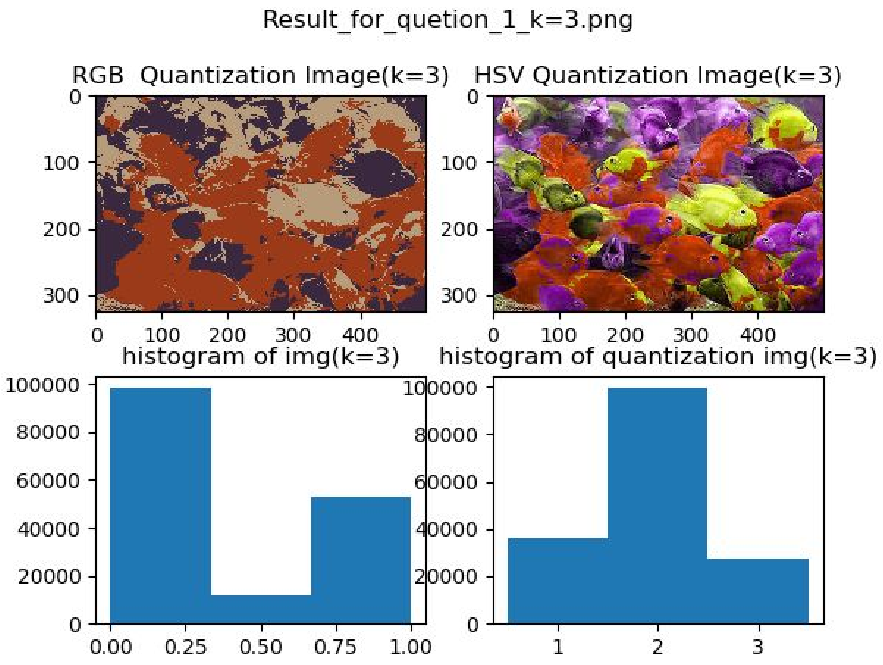

Figure 1. result image when k=3

2. k=8

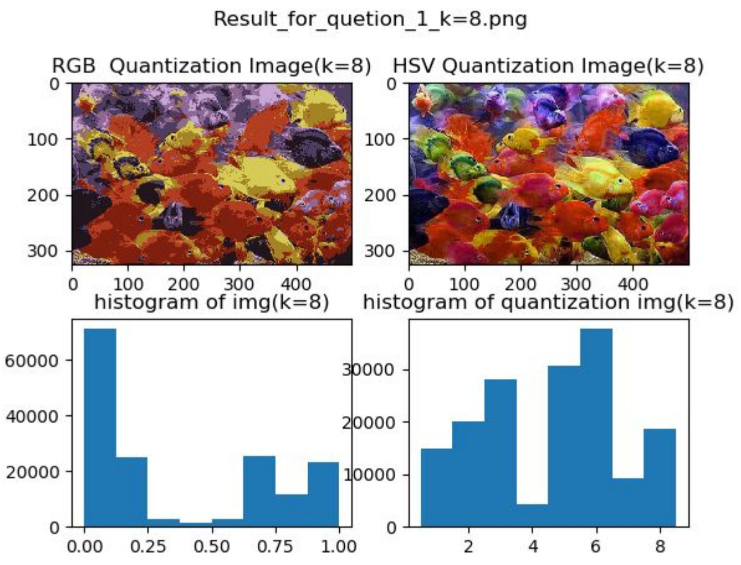

Figure 2. result image when k=8

3. k=13

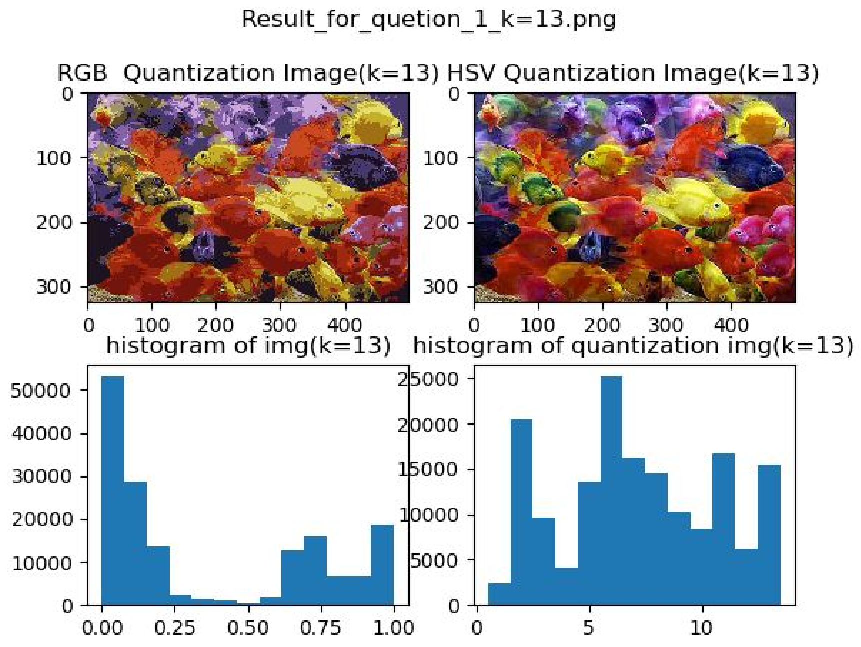

Figure 3. result image when k=13

4. Computer the SSD

Table 1. SSD of RGB image and quantized image

K	3	8	13
SSD	721354729	318285924	218306760

Table 2. SSD of HSV image and quantized image

K	3	8	13
SSD	273944044	53885635	31513401

5. Use the same K to check whether the results are the same when we compute SSD

Table 3. SSD of RGB when k =3

K	3	3	3	3	3
SSD	721357265	721354729	721354729	721357513	721357513

(f)

Conclusion:

As table1 and table 2 shows, with the increase of K, the SSD is decreased. This is because when we quantize the image, although its memory space is reduced, it also

lead to the color reproduction error. In other words, each pixel would be presented by the color of cluster center. Thus, the smaller K is, the bigger the value of SSD.
When we compared the data in table 1 and table 2, we can see : the error value of RGB is always bigger than HSV. It is because we do the kmeans clustering operation on R,G,B channel of the image but only change hue channel in HSV quantization function. We can easily come to the conclusion that smaller error would look more similar to the original image.
When we run the histogram function, we found that the histogram of the quantization image would be different. Also, in table3, when we run the computeQuantizationError function with the same k, we would find the values are different .This is because the initialization has effect on the result we want to plot. If we choose an ideal set of initial points, the time consumption of the code would reduce.
What is more, compared with two histogram, we can recognize that the bin that contain smaller value increase and the distribution of hue in this image become more stable. This is because the original fish image mainly contains red, yellow and purple. After kmeans clustering, the hue of the image becomes softer, which leads to the change of the histogram.
From figure 1 to 3, we can clearly realize the differences between RGB and HSV color space. For example, when k=8, the HSV quantization image is more similar to the original image, which means it reserve more information of the original one. Even when the value of k increases, we could still see the difference between the image produced by the RGB quantization algorithm and the original image. The reason is that the HSV algorithm does not use each channel to do the quantize operation. In addition, because the HSV algorithm only processes one channel, it also run faster than the RGB quantization algorithm.

Circle detection with the Hough Transform

(a)

Step1:Convert the image to a grayscale image
Step2: Compute the gradient of the image and get the value of Θ (When you compute the division of dx and dy, you need to consider the case where the denominator equals zero. If you do not use the gradient method, you can skip this step)
Step3: Run the canny function to get the edge information of the image. (You should pay attention to the value of Sigma, it will have a huge influence on the result.)
Step4: Get the coordinates of X and Y in Hough space, which is (d , Θ),add it to the accumulate array H(d, Θ) (if you think the value of H(d,Θ) is not obvious at the center of the circle, you can augment it by adding 1 to all the pixel surround (d,Θ).)
Step5:Find the values of (d , Θ) where H[d, Θ] is maximum (Since there may be several circle in a image, you need to consider thresholds for finding points that meet the requirements. What is more, if there are several points in a close area, you have to choose one and remove others. The area can be control by parameter min_distance, which means the minimum distance between pixel (d,Θ) and the pixel in the center set.)
Step6:Get the coordinates of d and Θ in image Space, plot all the point. (you can also use plt.circle to help you do it)

(b)

For jupiter.jpg, we use radius=51 and radius =100 to find the circle.The result as follows:

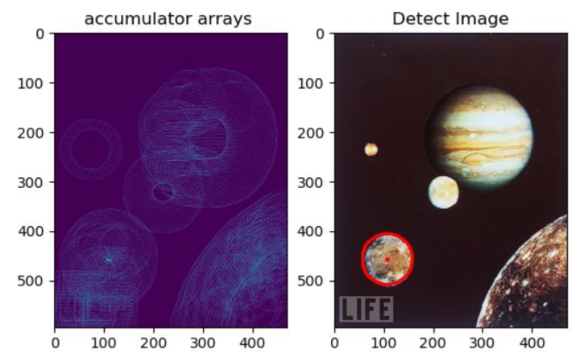

Figure 4. result image of jupiter.jpg when radius=51, useGradient =0

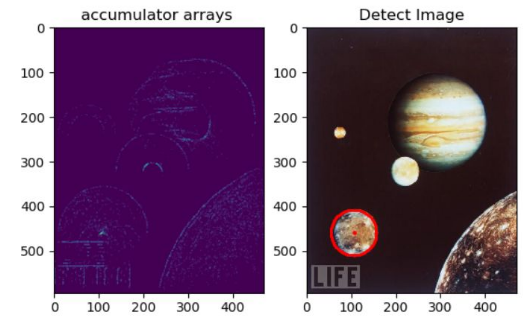

Figure 5. result image of jupiter.jpg when radius=51, useGradient =1

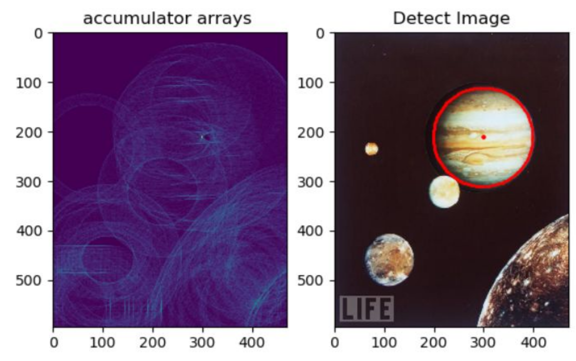

Figure 6. result image of jupiter.jpg when radius=100,useGradient =0

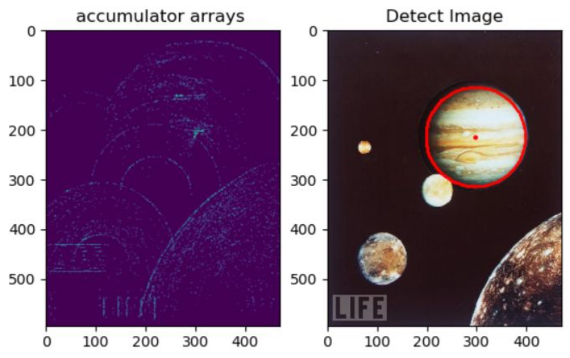

Figure 7. result image of jupiter.jpg when radius=100,useGradient =1

For egg.jpg, we use radius=51 and radius =100 to find the circle.The result as follows:

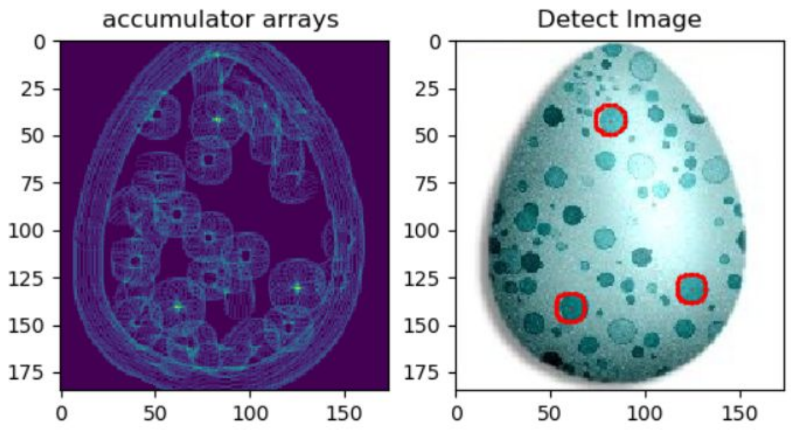

Figure 8. result image of egg.jpg when radius=8,useGradient =0

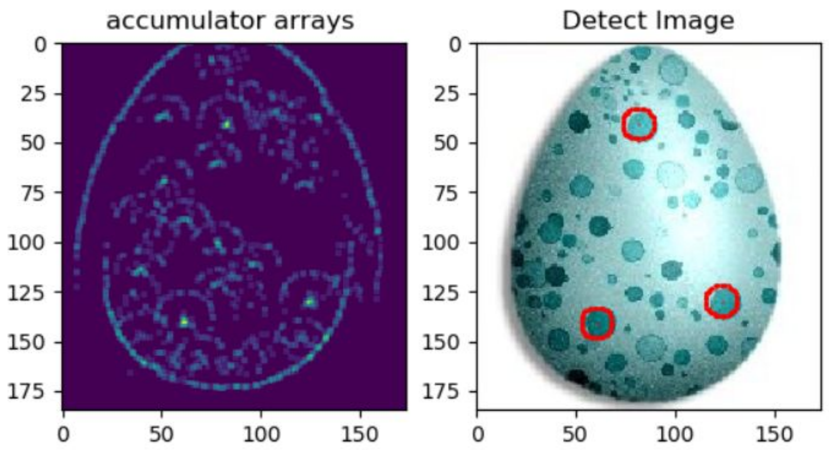

Figure 9. result image of egg.jpg when radius=8,useGradient =1

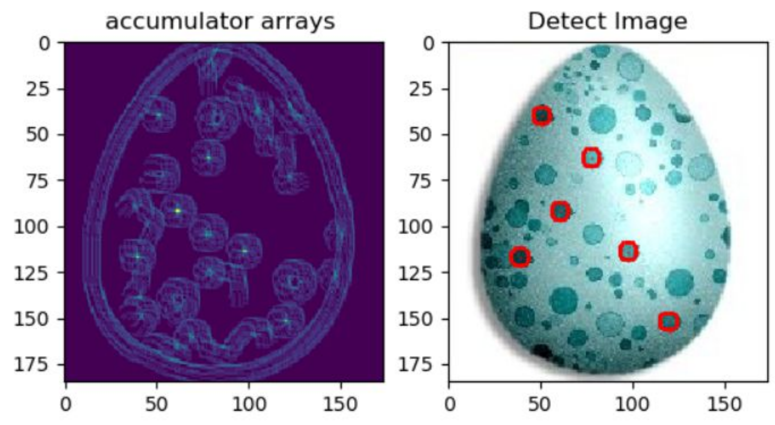

Figure10. result image of egg.jpg when radius=5,useGradient =0

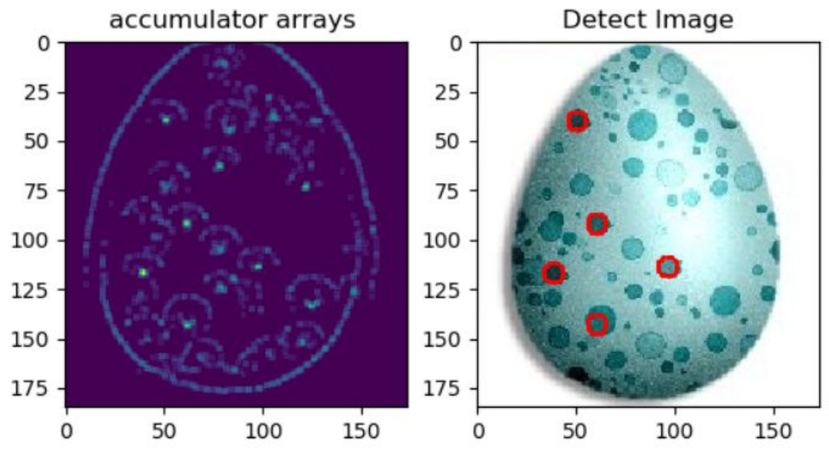

Figure11. result image of egg.jpg when radius=4.5,useGradient =1

(c) As we can see in Figure 5,7,9,11, the hough transform has a good performance on circle detection. By transforming the coordinate into Hough Space, we can realize that the center of each circle contains more information than other spaces (it looks lighter) . This is because different points on the edge of a circle in image space would all intersect with the center of the circle. Thus,by using hough transform, we can detect the circle easily.
However, the image processing before hough transform is also of vital importance. In order to reduce the noise in the image and find a clear curve of each edge, we use canny algorithm. In the research, we suggest that you can use sigma=3 to do process the image. If the value of sigma is too low, the noise can not be eliminated. On the other hand, if the value of sigma is too high, the edge of the circle we want to detect would diminish.
If we compare the performance between useGradient=0 and useGradient=1, you can find it perform well on this two images. Though the detected circle is of slightly different between two methods, we can still find that the light point in both accumulator array image is nearly the same, which means the result is still receptacle. All we have to do further is to adjust the value of threshold and min_distance.
As for the time consumption, since we remove a for loop to calculate the value of theta, the gradient method would run faster.

(d):All you need is to find the size of the center set. In the detectCircles.py, we have filtered the point above the threshold and get the maximum point in the specified area by the the value of min_distance. Thus, you can use the center.shape[0] to find how many circles are in the array. We create a dictionary to save the index information and the value information of each element in the accumulator array. After that, we sort the whole dictionary, remove the

element whose value is smaller than the threshold. Then we check whether the element contain the biggest value of a certain area or not.
(e):When we reduce the size of the bins, we will get a less precise center points and the result may not be good. However, by reducing the size of bins, the program would run faster. For example, following is the egg image when we set radius=8 and undergradient =0, and the size of bins is 3 times smaller than what we set in the questions above. you can easily see that the accumulator array become less precise when we choose the center point.(Since I use VScode to run this question, the color of the background is different)

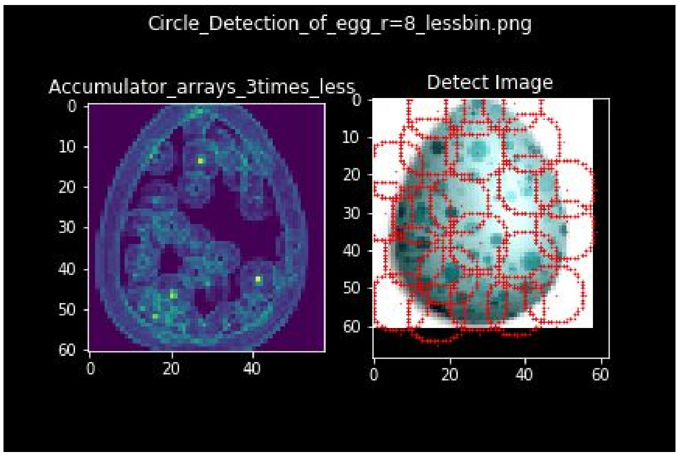

Figure12. When the bin size When bin size shrinks three times, the accuracy decrease (r=8).

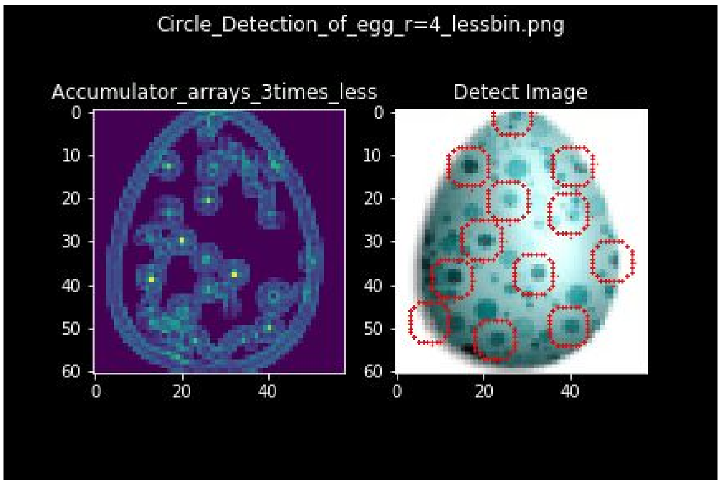

Figure 8.When the bin size When bin size shrinks three times, the accuracy decrease.(r=4)