Kavin Krishnan
PS2 Report

I. Short answer problems

1. The resulting representation is not sensitive to orientation and is invariant to it. We can see that the filter bank has sextons of various orientations. As a result, a sample can be detected regardless of it alignment.

2. K means will take two initial centers, then find all the points closest to them and create two clusters (since k=2 in this case). It will then find the average point amongst the samples in these clusters and then form into two new clusters. The process repeats until this algorithm converges. K means will try to maximize the largest grouping of closest points per main point. Therefore, k means will most likely converge in this example to have two clusters which contain the semi-circles or half circles of each of the circles.

3. The mean-shift algorithm would be most appropriate. K means and graph cuts are not designed to determine the values of parameters such as maximum vote which we desire. On the other hand, mean-shift is designed to find the maxima of a density function. This can therefore be taken advantage of to find the maximum vote.

4. // Blobs represents the list of blobs
   K = input of specified number of groups

   For each blob in Blobs:

   Find centroid (cx, cy) of blob

   // Can be done using through OpenCV through cv2.findContours

   For each pixel in Image: //Image is the binary image

   Calculate the circularity and store in an array called circ[][]

   //This can be done using our centroid and the Circularity

   //equations on Slide 99 of Lecture 4: Edges_Binary.

   //You basically find the circularity from the mean / variance

   // of radial distances.

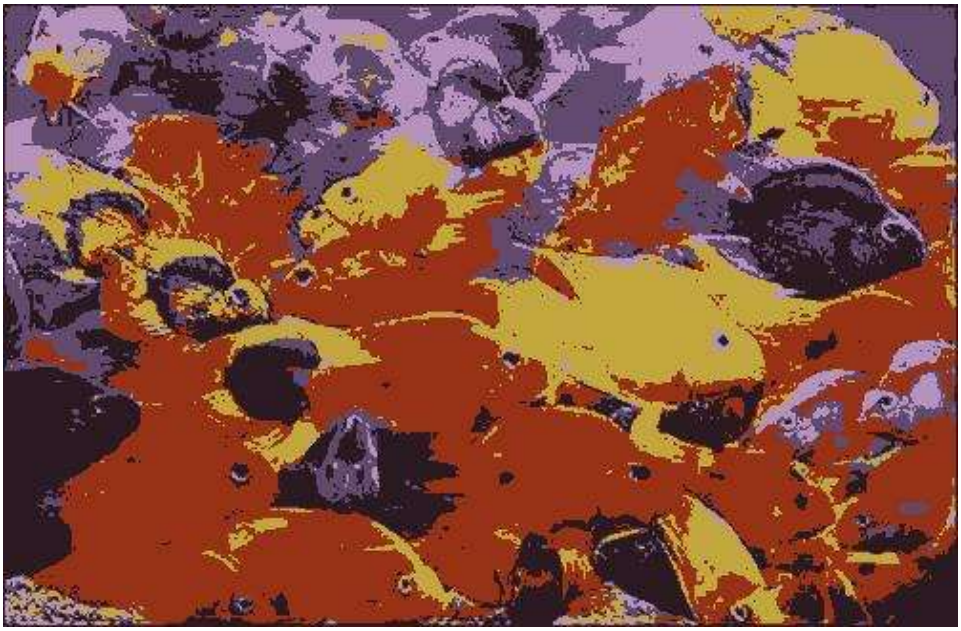
   // Circularity is a simple feature that can help identify a blob

   Run k-means w/ circularities in circ[][] and determine K blob regions

II. Programming

1. Color quantization with k-means on Fish.jpg (part e and f):

RGB Quantized Results with K=5

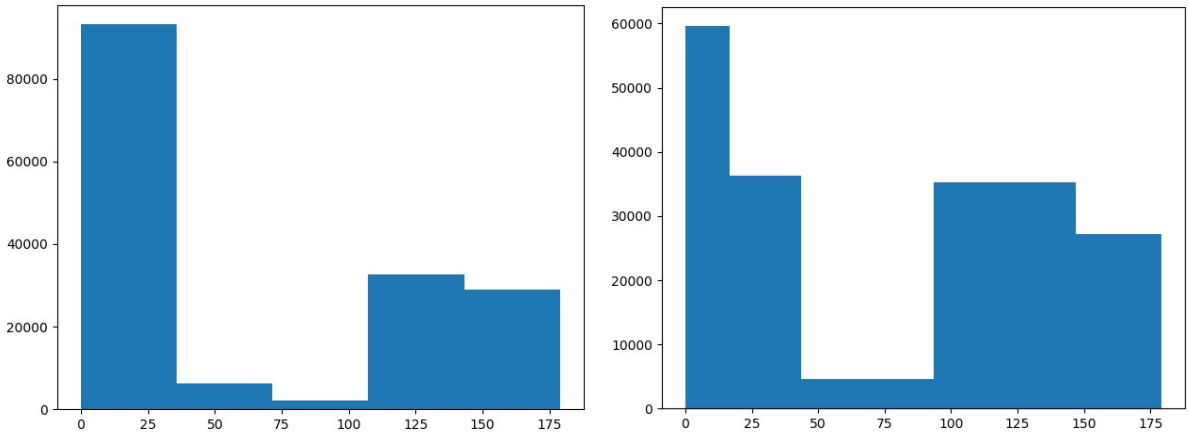


SSE Error: 437785265

HSV Quantized Results with K=5

SSE Error: 69384967

Equal Bins & Clustered Bins Histograms:



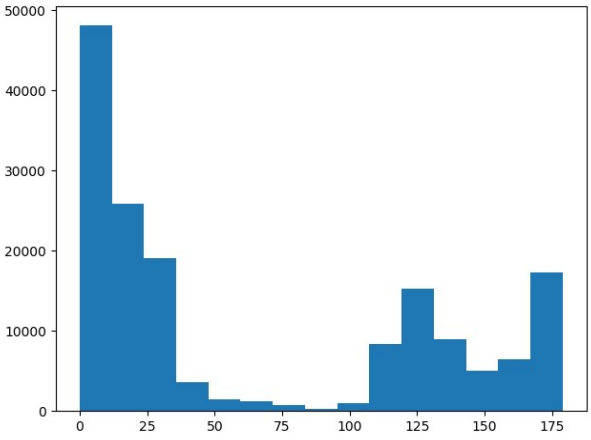
RGB Quantized Results with K=15

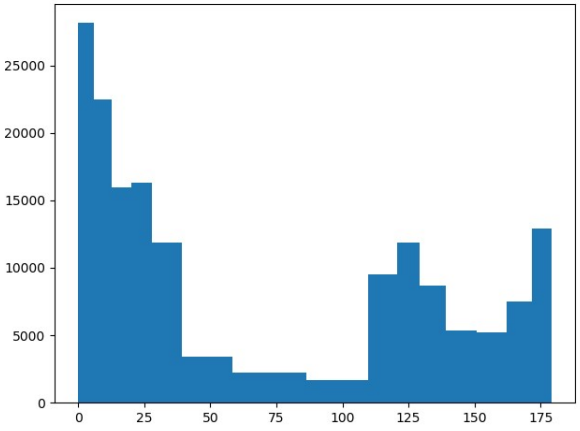
SSE Error: 163459167

HSV Quantized Results with K=15

SSE Error: 17233424

Equal Bins & Clustered Bins Histograms:





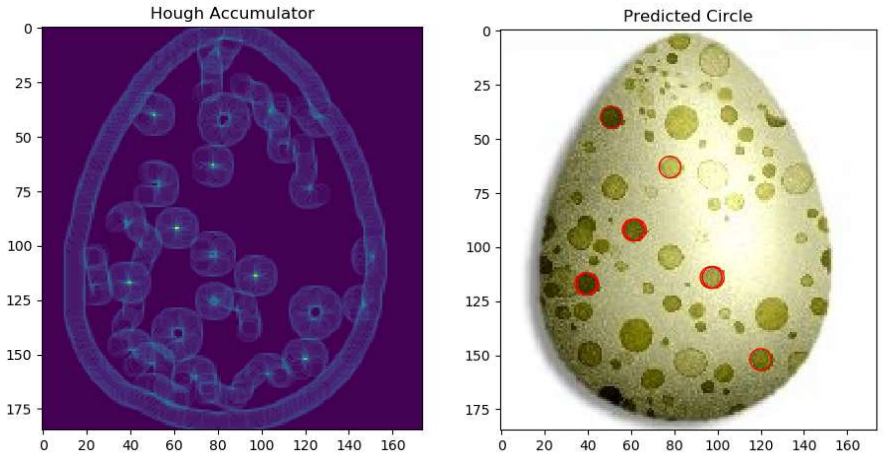
In the first quantized RGB image with K=5, the cluster has the choice of 5 RGB values to select as centers. It can be seen through the result of this experiment that this method of using RGB values does not create a relatively accurate representation of the actual image. The resulting picture consists of clusters of simple colors. On the other hand, the quantized HSV image with K=5 seems to produce a more accurate representation of the original image. This is this case because HSV provides more information to describe the state of a pixel by utilizing color intensity (over simply the RGB values). The quantized HSV image with K=5 had an 84.1509% smaller SSE value

than the quantized RGB image. The quantized RGB image with K=15 allows for 10 more additional RGB values to select as centers than the quantized RGB image with K=5. This enables the K=15 image to have a 62.6623% smaller SSE value than the image with K=5. Although this is the case, this method again shows that it is more difficult to accurate represent the original image. The quantized HSV image with K=15 allows for 10 more additional hue values to cluster around and this creates an image that nearly resembles the original image. This quantized HSV image with K=15 saw a 75.1626% decrease in SSE error over the quantized HSV image with K=5. It also had an 89.457% decrease in SSE error over the quantized RGB image with K=15. With regards to the histograms, the histogram for the equally spaced bins and clustered bins are actually not that different for both K=5 and K=15. This could be as a result of the clustering algorithm selecting centers with the different concentrated neighborhoods of hue values. However, by not having to equally spaced bins, the histogram for clustering seems to more accurate represent the distribution of hue values in the image. The aforementioned results seem to hold true across the 20-30 runs completed using Fish.jpg.

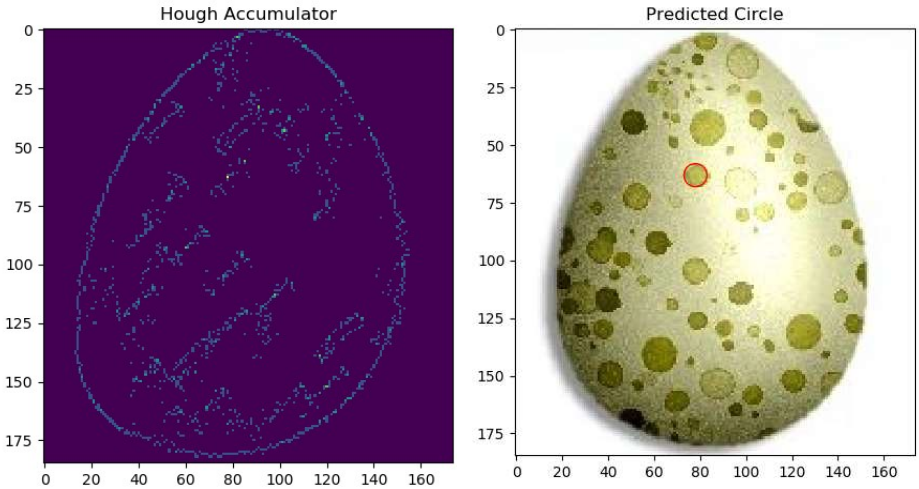
2. Circle detection with the Hough Transform
   (a) First, the input image is converted to an array of greyscale values. This array is then processed by a canny edge detector to produce an array where elements signify the presence of an edge. A hough accumulator is created with the same shape as the image. This accumulator will be used by edges detected by canny operator to vote on circle centers. Each element in this accumulator represents a potential center. Now, the edges detected by the canny operator are used to vote on center points. The array produced from canny edge detection is iterated over and if an element represents no edge it is skipped over. On other hand, when an element represents an edge point, we continue with the following steps. We take a direction (theta) from this edge point to find a center. If using a gradient, we simply take $\tan^{-1}(dI/dy/dIdx)$ to find theta. If we are not using the gradient, we simply iterate from 0 to 360 degrees. We then transform the equation for a circle $(x-a)^2 + (y-b)^2 = r^2$ (where (a, b) is the center and r is the radius) to utilize the information we already know. We get the equations a = x – r * cos(theta), b= y + r * sin(theta). The value at element in the hough accumulator representing center (a,b) is then incremented by 1. After going through all points in the array of the canny edge detector, the elements of the accumulator with the highest votes are our predicted centers.

(b)

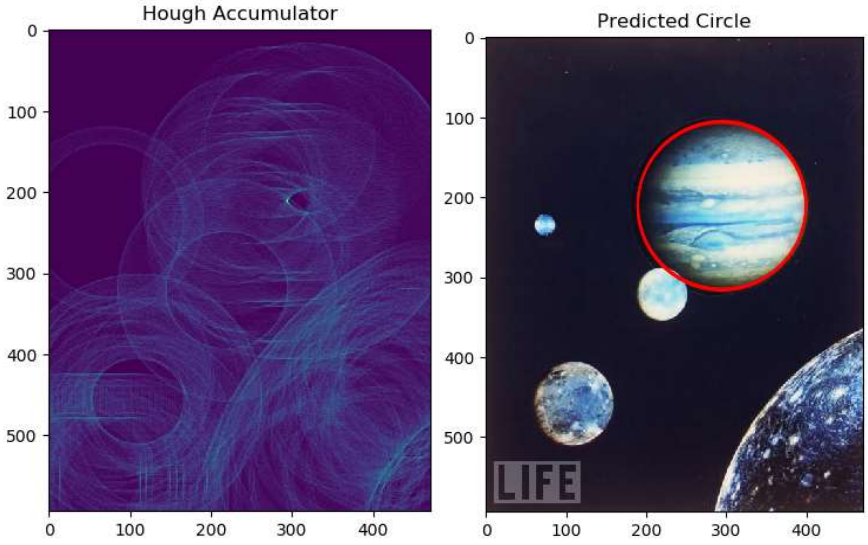
Radius = 5, Gradient = 0



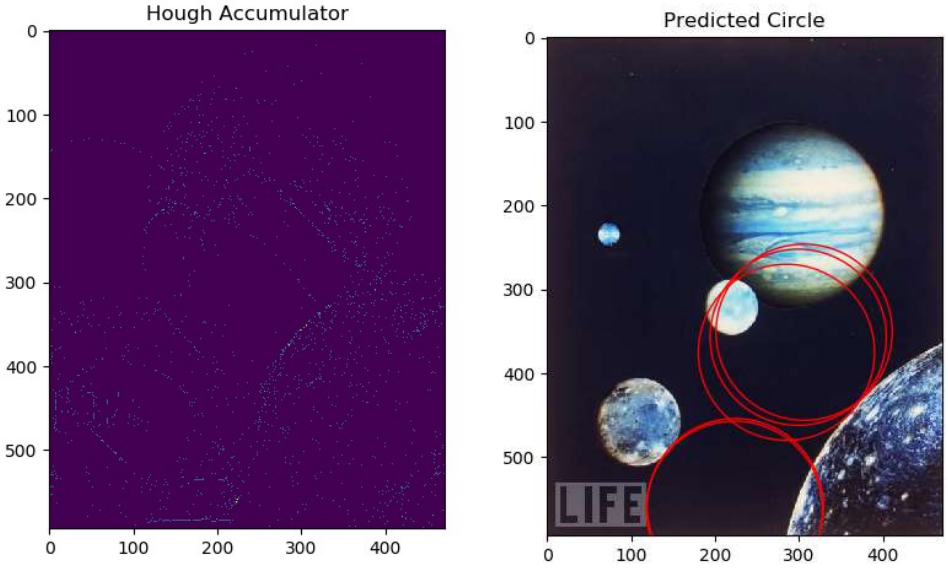
Radius = 5, Gradient = 1



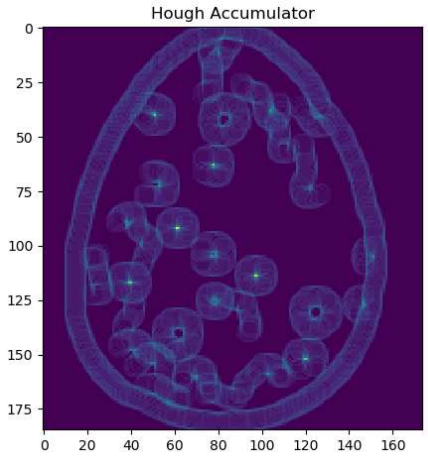
Radius = 105, Gradient = 0



Radius = 105, Gradient = 1



(c)

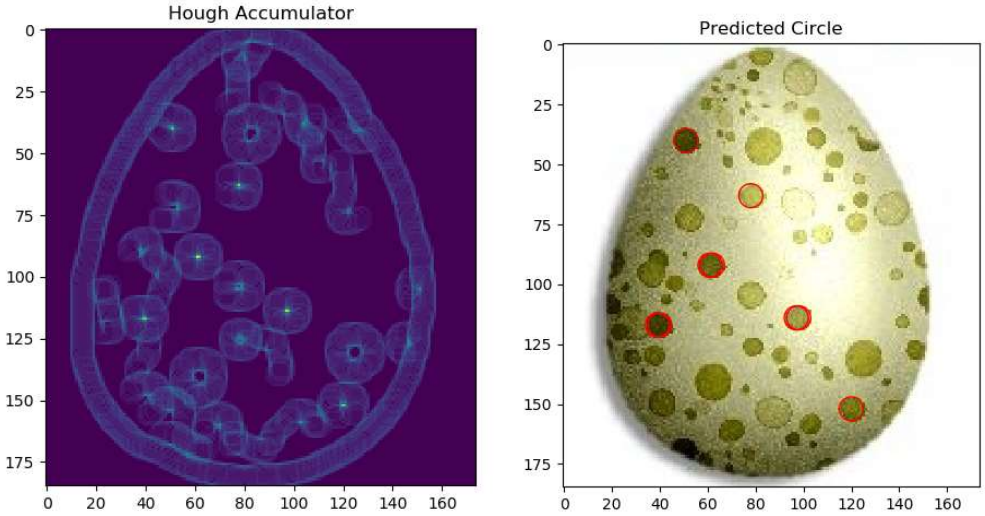


Radius = 5, Gradient = 0

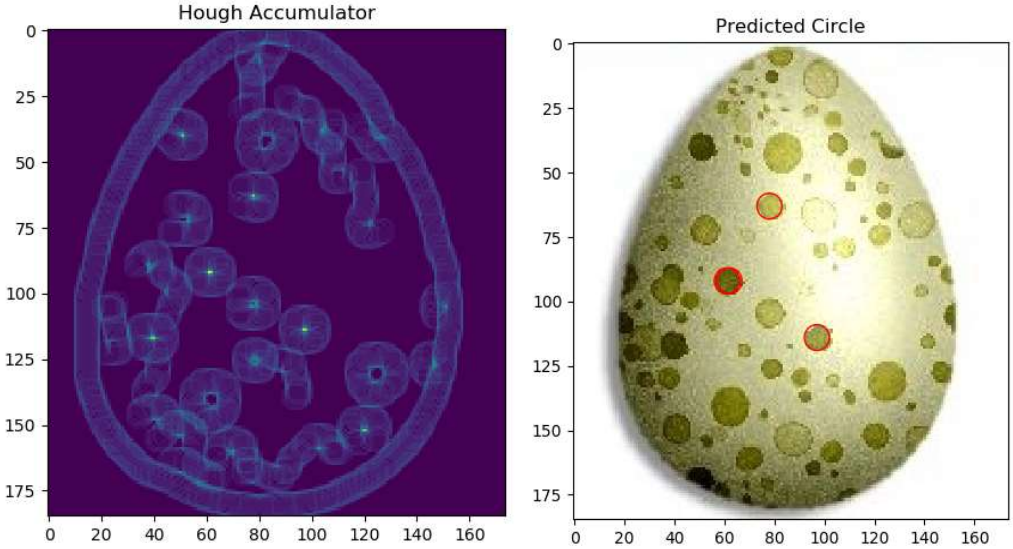
For egg.jpg, we can see that the spots (or circles) on the eggs have relatively easily distinguishable edges compared to Jupiter.jpg. In this run without using gradients, we can see our algorithm iterating along these edges through the voting in the accumulator. We can see high concentrations of votes inside and outside of these spots as seen by the various donut shapes in the displayed hough accumulator. We can see the max votes by the bright blue circles, which is where our algorithm predicted centers.

(d) After conducting a vote from all edges on where centers are, I set a threshold based on the maximum vote in the accumulator to select centers. This is because all circles of the input radius may necessarily have the same amount of votes. We need to extract essentially where there are relative max's to do this. A trivial way to do this was to selects centers where a pixels votes were greater than a certain percentage of the maximum. Here are the results as follows.

Egg.jpg Radius=5, Use_gradient=0
Threshold: centers with votes >= 70% * Max_Vote



Egg.jpg Radius=5, Use_gradient=0
Threshold: centers with votes >= 80% * Max_Vote

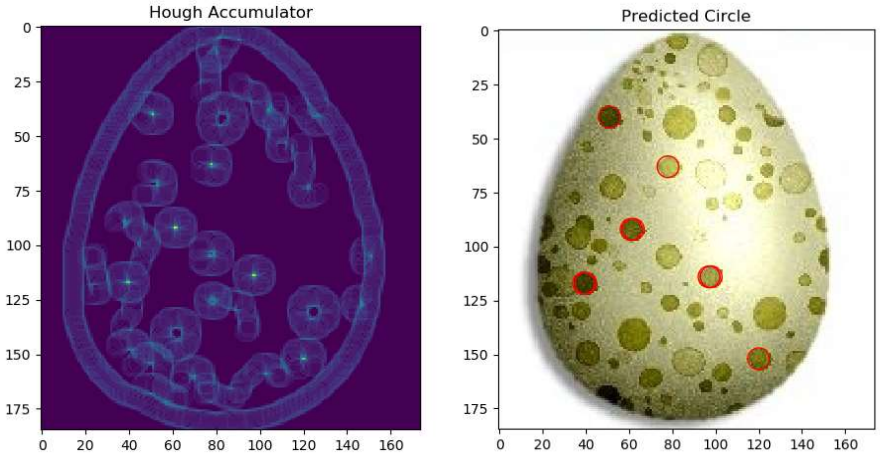


I found that the most optimal threshold was 70 percent as it allowed the algorithm to detect the most circles in egg.jpg without predicting incorrect circles.It was essential to make sure the threshold was not too low as otherwise, non-centers would be more likely to be detected as centers.

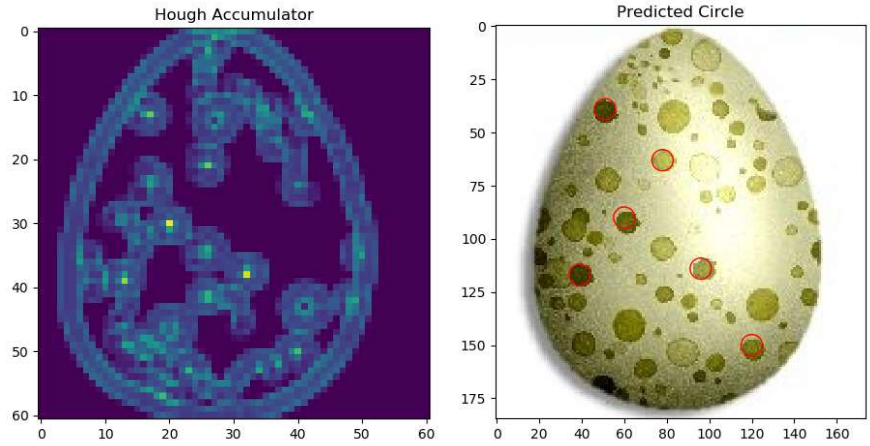
(e)

Egg.jpg Radius=5, Use_gradient=0

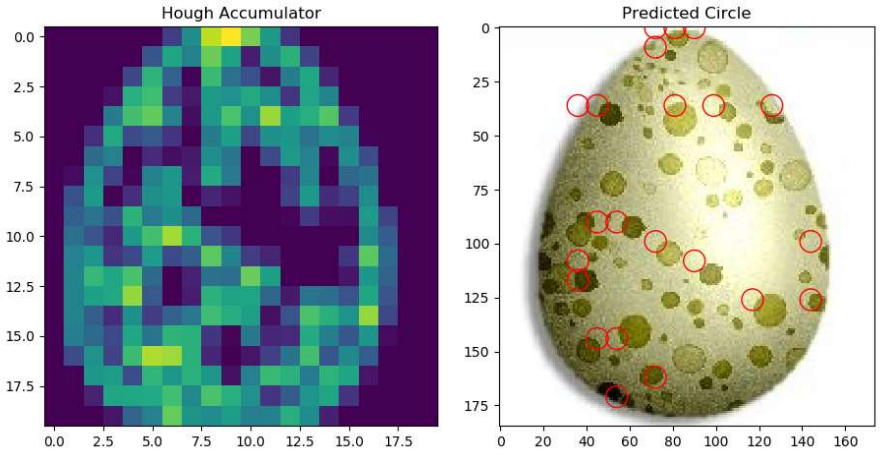
Bin Size = 1



Bin Size = 3



Bin Size = 9



It can clearly be seen that the larger the bin size is for detecting circles in Egg.jpg, the less optimal this algorithm functions (as seen by the off centered circles). Having a larger bin size may have the benefit of trivializing the accumulator array by not having to store as many elements, however it sacrifices precisely predicting centers for small circles. A larger bin size can help eliminate some redundant information in accumulator and double counting the center of a circle. An example of this can be seen when the bin size = 1 and some circles have detected two circles. This is because our algorithm created an accumulator where two centers were detected near each other. In reality, both of these detected centers correspond to only one center. When the bin size = 3, we can see that although circles are off centered, only one circle is accounted for. A larger Bin size might help when dealing with images where the exact curves of circles are harder to detect and simply mapping an accumulator element to each pixel might not make sense as curves might be incomplete or not shaped well. Therefore in this case it makes more sense to set an accumulator that maps elements from a neighborhood of pixels to approximate where the actual center of circle is. However in the case of egg.jpg, the circumference can easily be detected and having smaller bin_size of is more optimal.