1 Short answer problems

1. The representation using the filter bank in the question will be invariant to orientation. This is because it contains rotated variants of the filters.

2. The kmeans randomly initializes two points in the feature space. Due to this and the nearest distance metric of clustering data points in kmeans, it will result in splitting of the points into two halves rather than intuitive clustering in concentric circles.

3. Kmeans algorithm will require the number of clusters first to cluster which we may not have in the beginning. Secondly, kmeans finds the centroid whereas we are looking for a local maxima, which might not coincide with the centroid. Also, kmeans is sensitive to outliers which may result in significant shifting of the centers in Hough space.
   Graph cut algorithm cuts the feature space into two graphs based on weakest edges which will not be useful for hough transform.
   Mean shift algorithm looks for mode and converges to center of mass. Therefore, this will be helpful in finding local maximas in Hough vote space.

4.

```
def calc_centroid(blob):
    sum = 0
    for pixel in blob:
        sum += position(pixel)
    return sum/size(blob)

def calc_distance_from_centroid(blob, centroid):
    sq_dist = 0
    for pixel in blob:
        sq_dist += (centroid - position(pixel))**2
    return sq_dist/size(blob)

def find_similar_shapes(blobs):
    avg_sq_dist = []
    for blob in blobs:
        centroid = calc_centroid(blob)
        avg_sq_dist.append(calc_distance_from_centroid(blob, centroid))
    labels, centers = kmeans(avg_sq_dist, k)
    return blobs grouped by labels
```

Variables

- size(blob) - number of pixels in blob
- position(blob) - (x,y) coordinates of blob
- kmeans - kmeans algorithm
- k - number of clusters

Algorihtm in words

(a) Calculate centroid for each blob.

(b) Calculate average squared distance of each point in blob from centroid to be used as a size invariant shape feature.

(c) Cluster blobs based on average squared distance.

(d) Group blobs by label.

2 Programming problem

1. **Color quantization with k-means RGB Quantization**

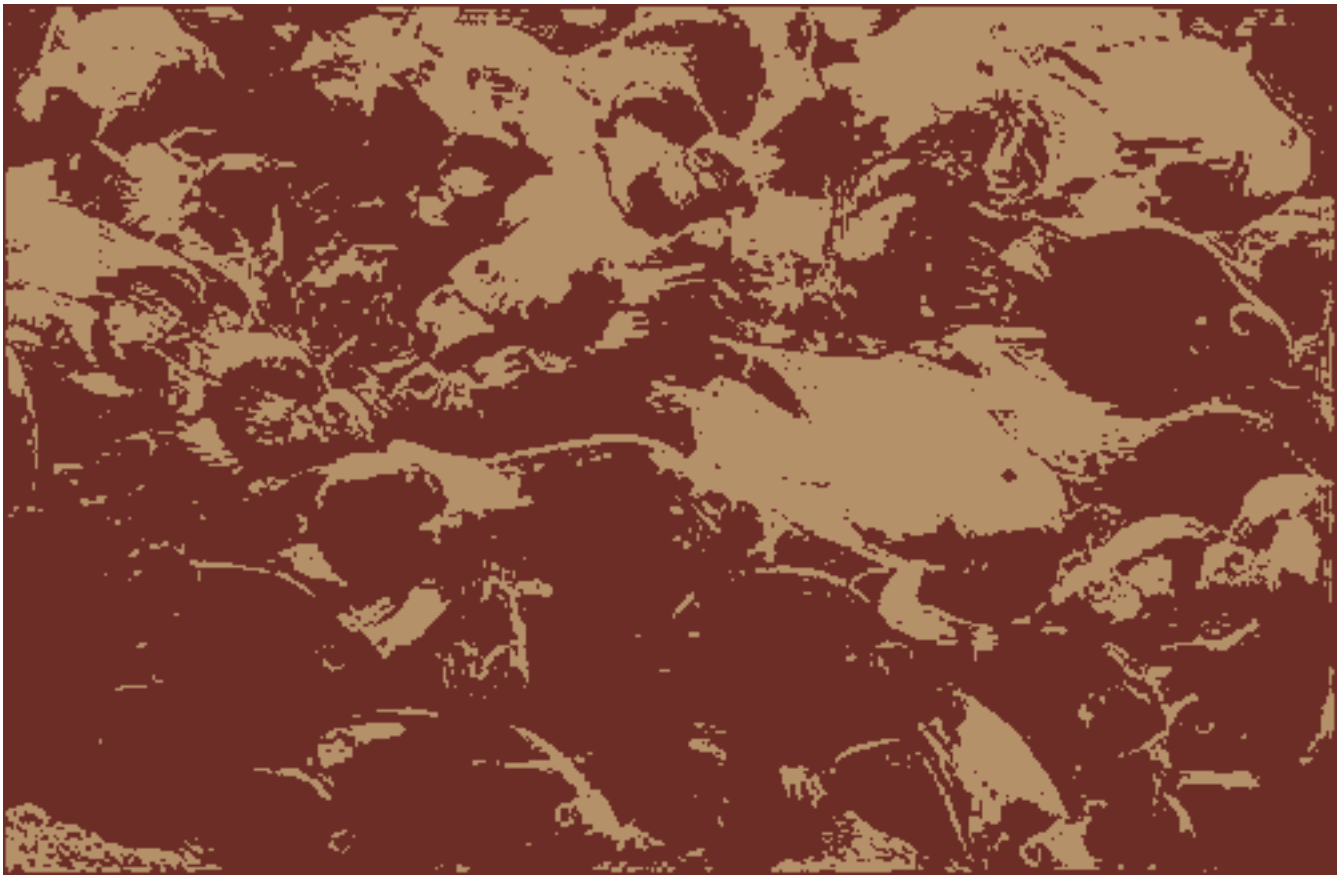

Figure 1: Clusters: 2

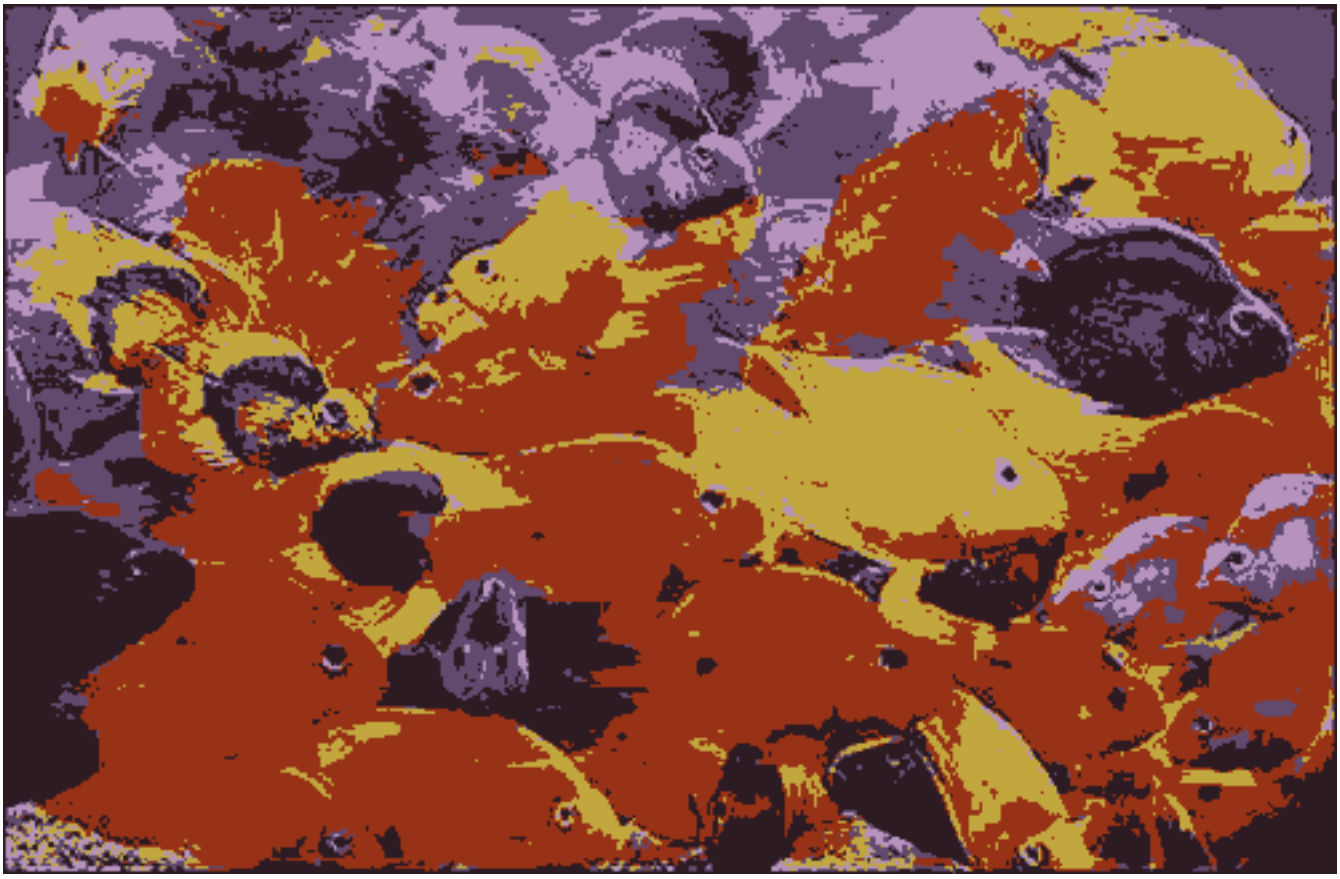

Figure 2: Clusters: 5

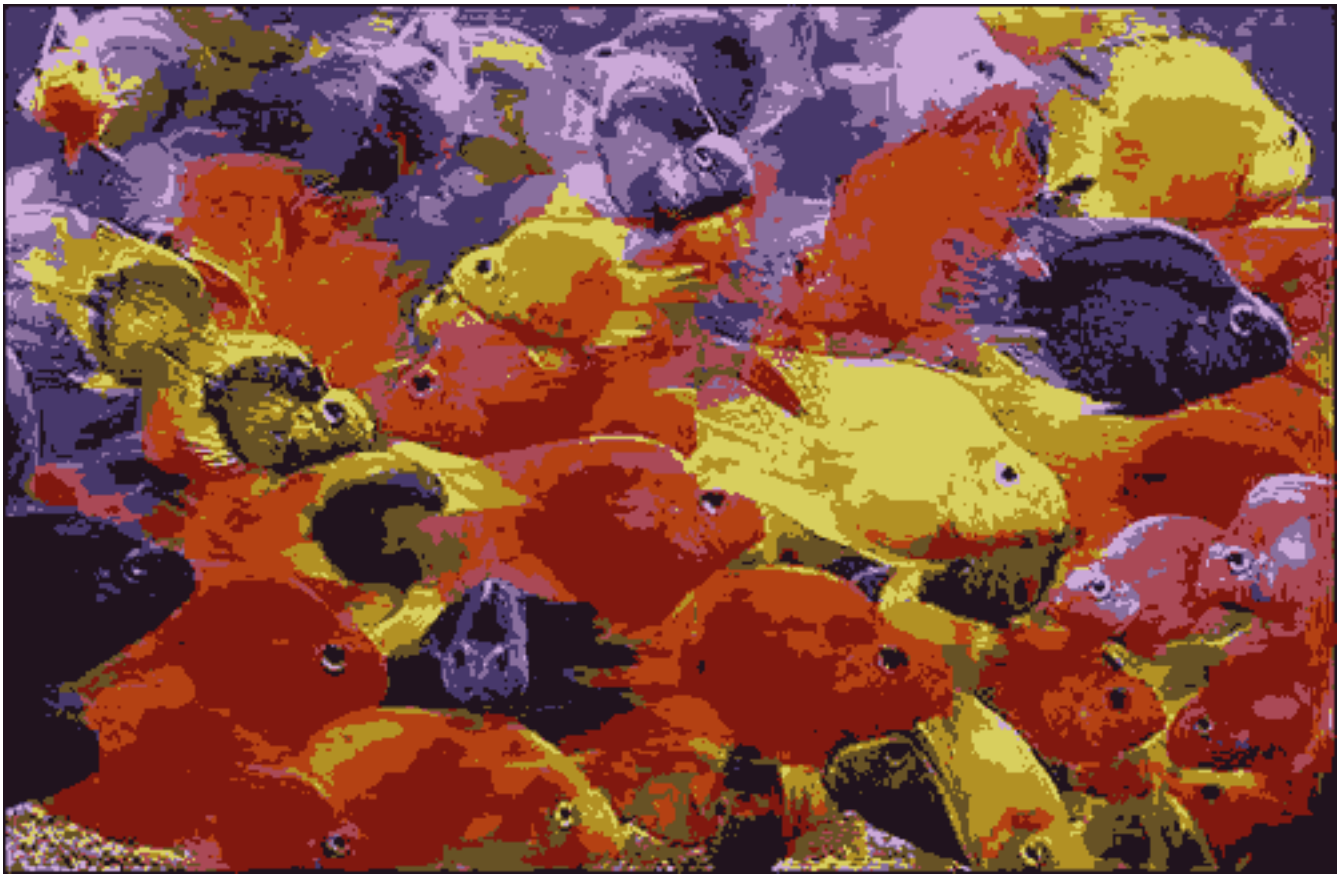

Figure 3: Clusters: 10

Figure 4: Clusters: 20

HSV Quantization

Figure 5: Clusters: 2

Figure 6: Clusters: 5

Figure 7: Clusters: 10

Figure 8: Clusters: 20

How do the two forms of histogram differ?

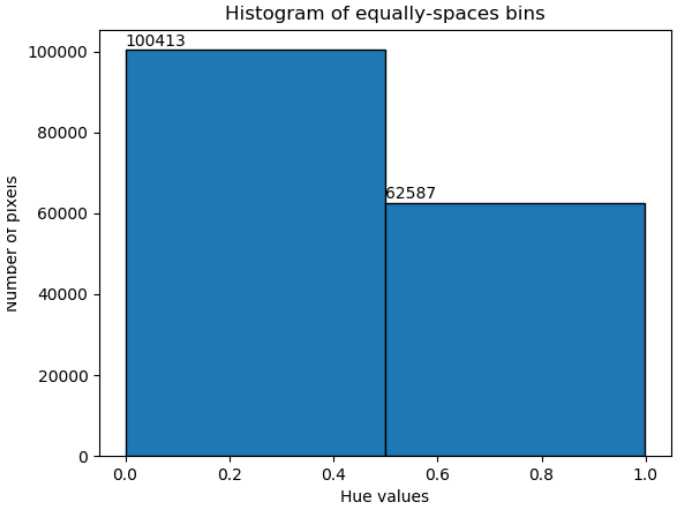

Figure 9: Clusters: 2

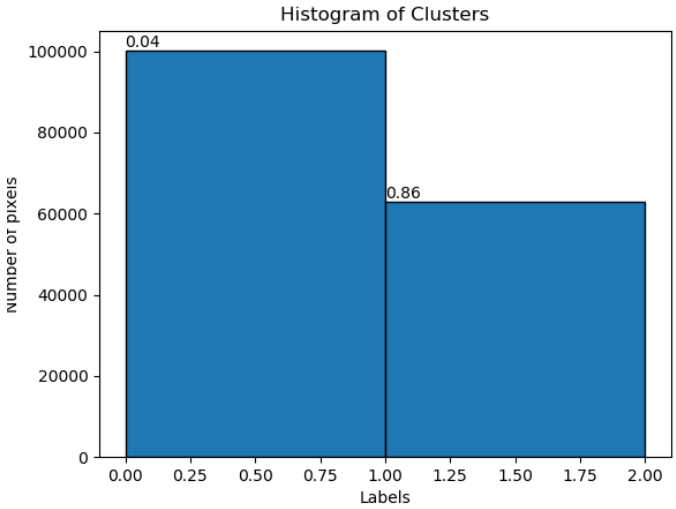

Figure 10: Clusters: 2

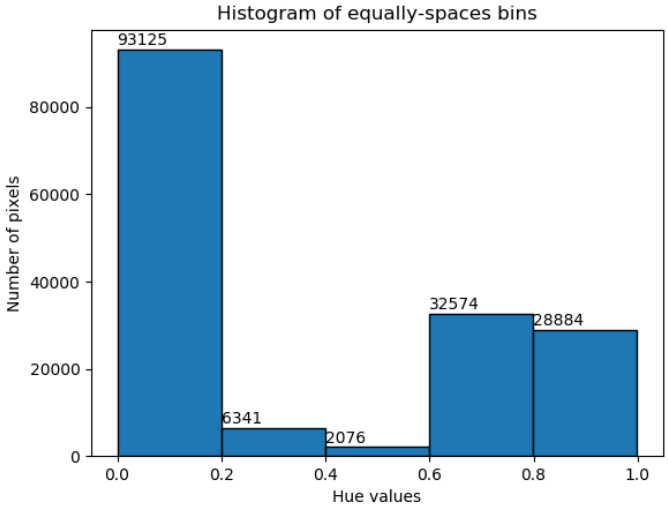

Figure 11: Clusters: 5

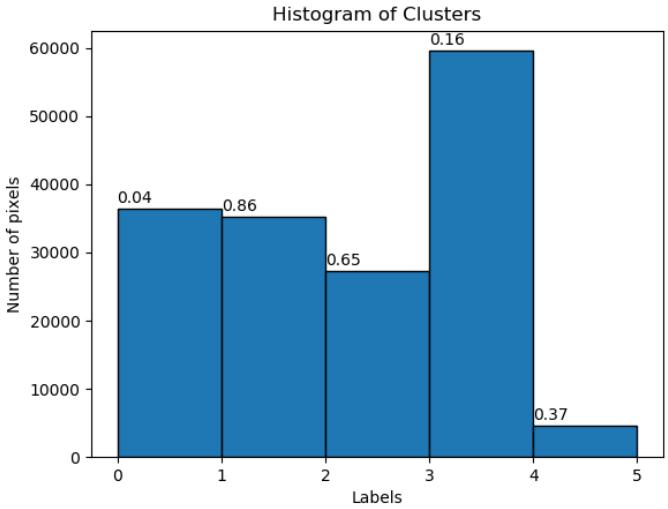

Figure 12: Clusters: 5

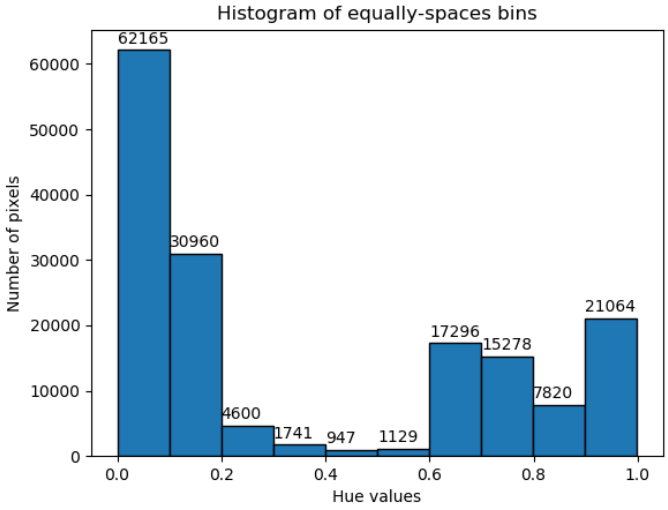

Figure 13: Clusters: 10

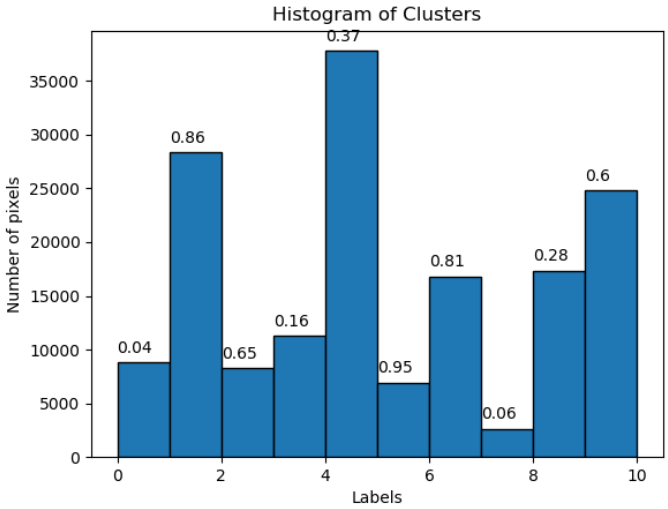

Figure 14: Clusters: 10

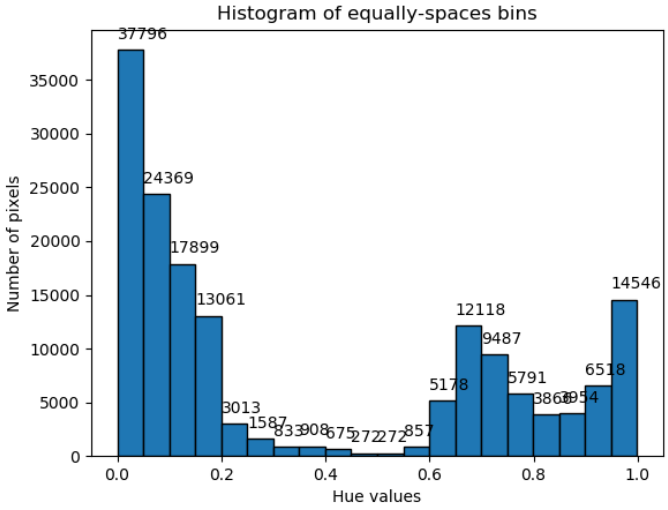

Figure 15: Clusters: 20

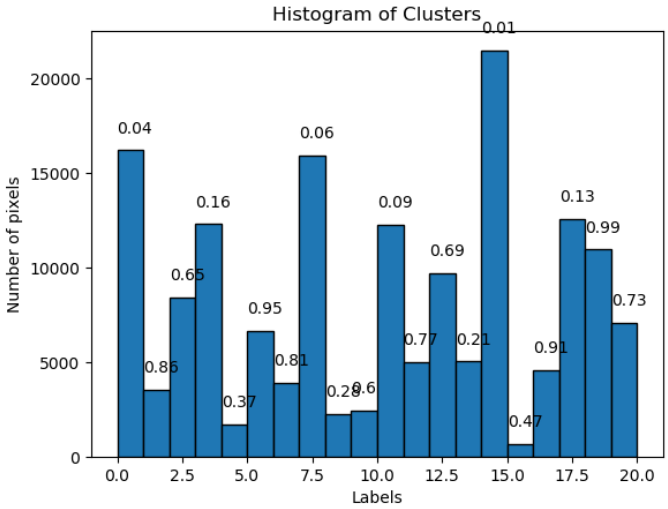

Figure 16: Clusters: 20

Number of clusters	RGB error	HSV error
2	1018917112	400789365
5	462449416	70755116
10	262564303	23981467
20	155398212	6427051

How and why do results vary depending on the color space?
Quantization results in RGB and HSV color spaces vary. Hue based quantization yields much better results even for lower number of clusters compared to RGB quantization. This can be understood as since the clustering for HSV based quantization is only done on the basis of hue values and other planes are left untouched. On the contrary, all the RGB values are set to cluster centers in RGB quantization. This affects how much the image is affected by quantization.

The value of k?
As can be seen clearly from the images, with the increase in value of k, the quality of image increases as expected. We also see a corresponding decrease in the SSD error.

Across different runs?
Since the k centers are initialized randomly. The error in clustering varies according to the cluster initialization. These are the computed errors for RGB quantization with k = 10 for 5 different runs. 262583173, 262562441, 262609418, 262579893, 262563946

2. **Circle detection with the Hough Transform**

   (a) Steps for detecting circles

   - Convert the image to gray image for gradient calculation
   - Get the edges image using Canny edge detector.
   - Calculate the gradient direction at each pixel using sobel filter for calculating magnitude in x and y direction.
   - Initialize Hough Space accumulator array with dimensions as the dimensions of grayscale image.
   - Compute the center of circle for only edge pixels.
   - For useGradient = 1, use the θ at that pixel and calculate a and b.
   - For useGradient = 0, loop through all the possible θ between $[-\pi, \pi]$ with a given θ resolution.
   - For centers, use all points who's vote is greater than equal to the 0.8 times the maximum votes for a pixel.

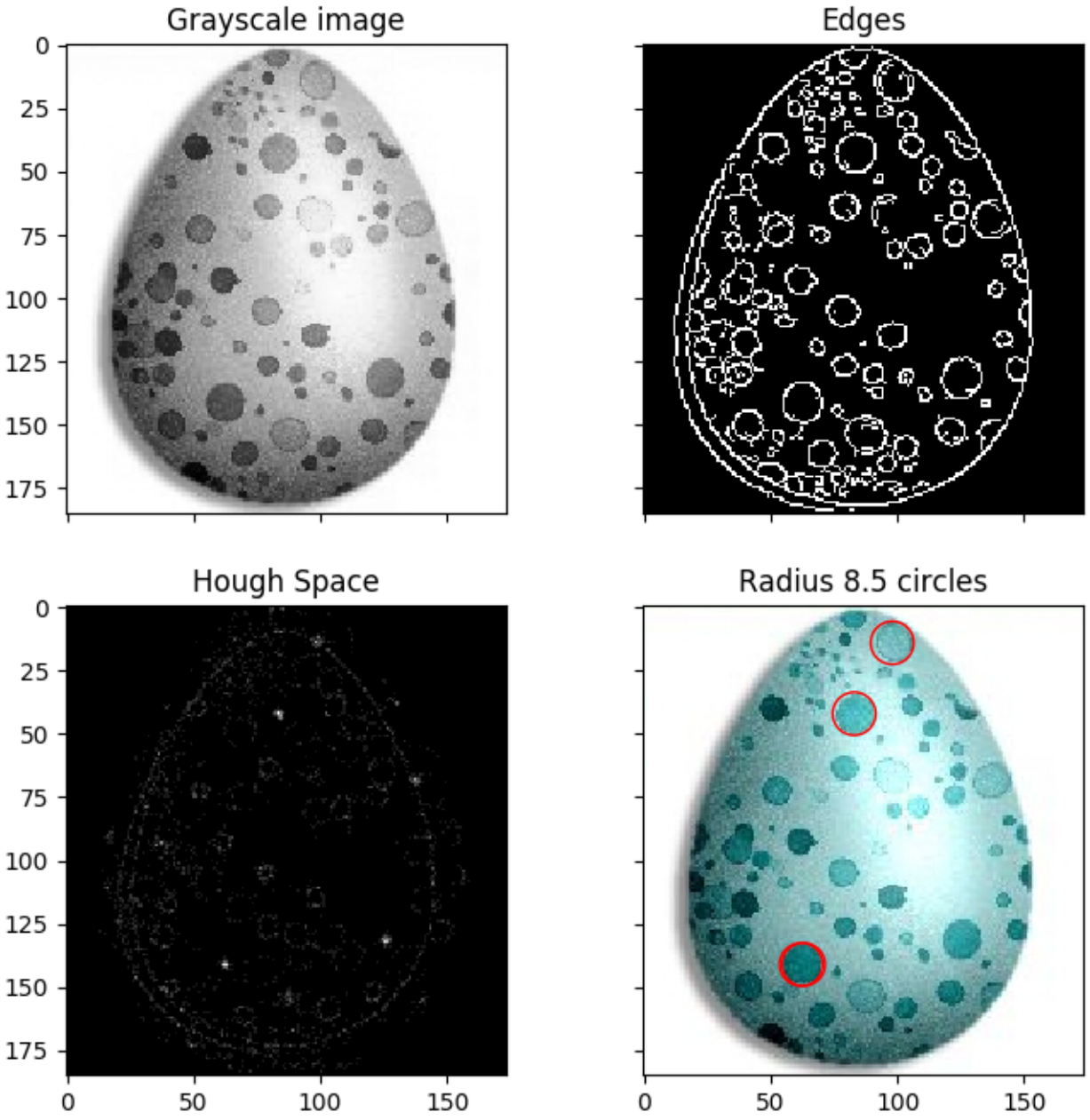

Figure 17: useGradient = 1

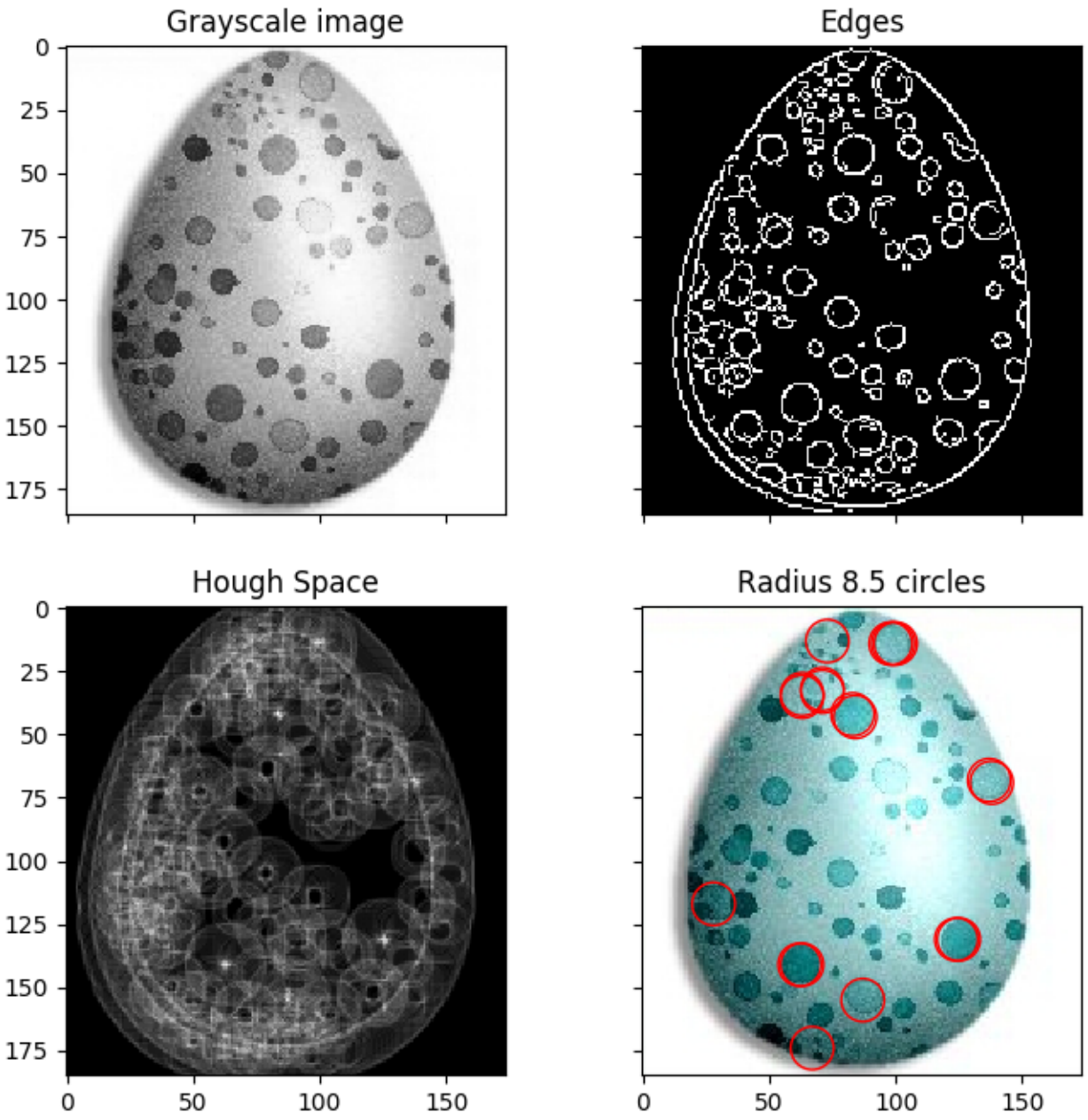

Figure 18: useGradient = 0, θ bin size = 1 degree

(b) **Hough Space** Using the information of gradient direction gives much better circles than trying out all angles for egg.jpg. For useGradient = 0 the spaces between nearby circles are also detected as circles, because the nearby circles edges contribute to the vote and since there is no gradient direction information, the algorithm doesn't know which θ is valid and which is not. Therefore, spaces between nearby circles are also

detected as circles. Hough space for useGradient = 1 has clear peaks while that with useGradient = 0 is hazy.

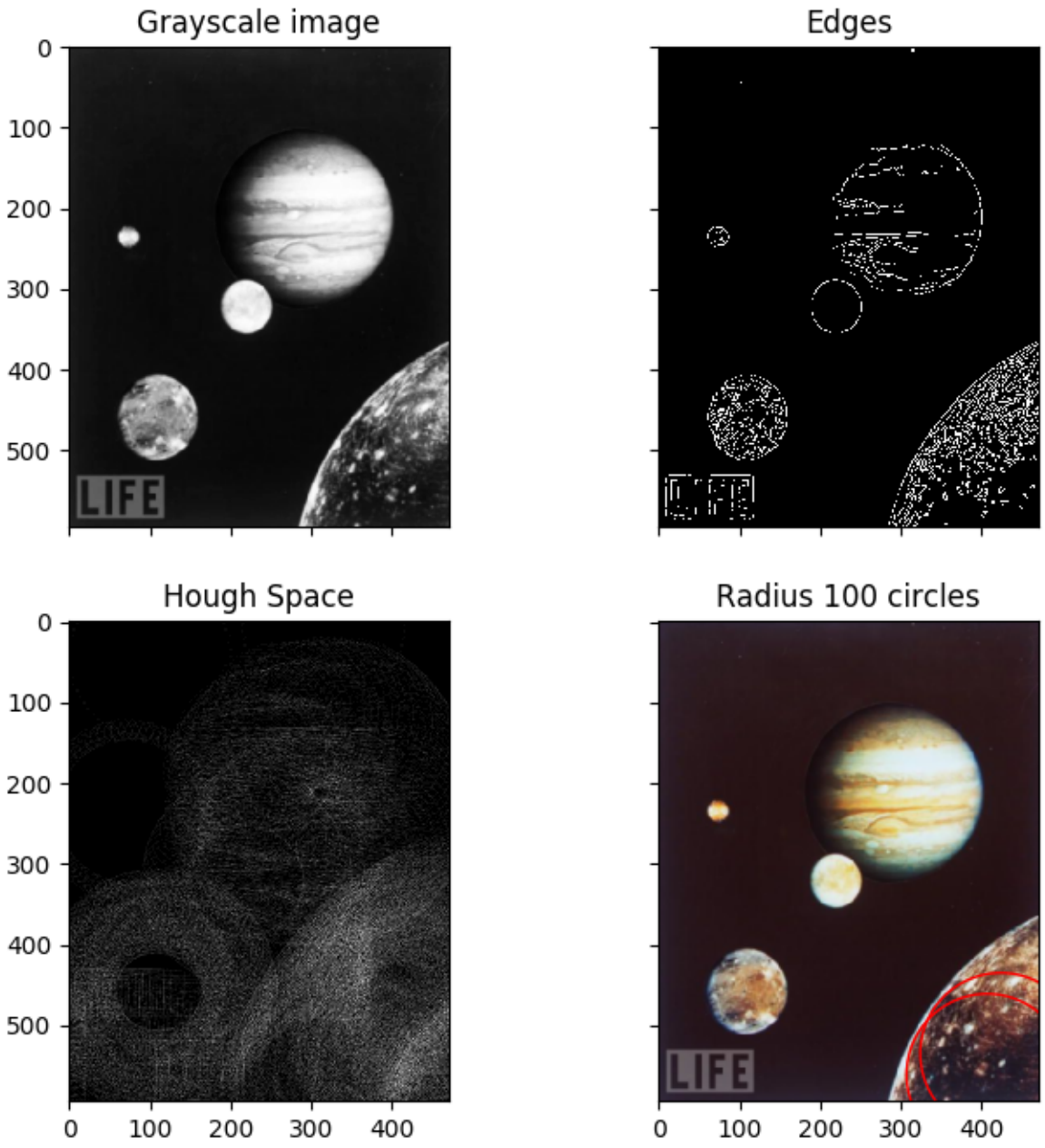

Figure 19: useGradient = 0, θ bin size = 5 degree

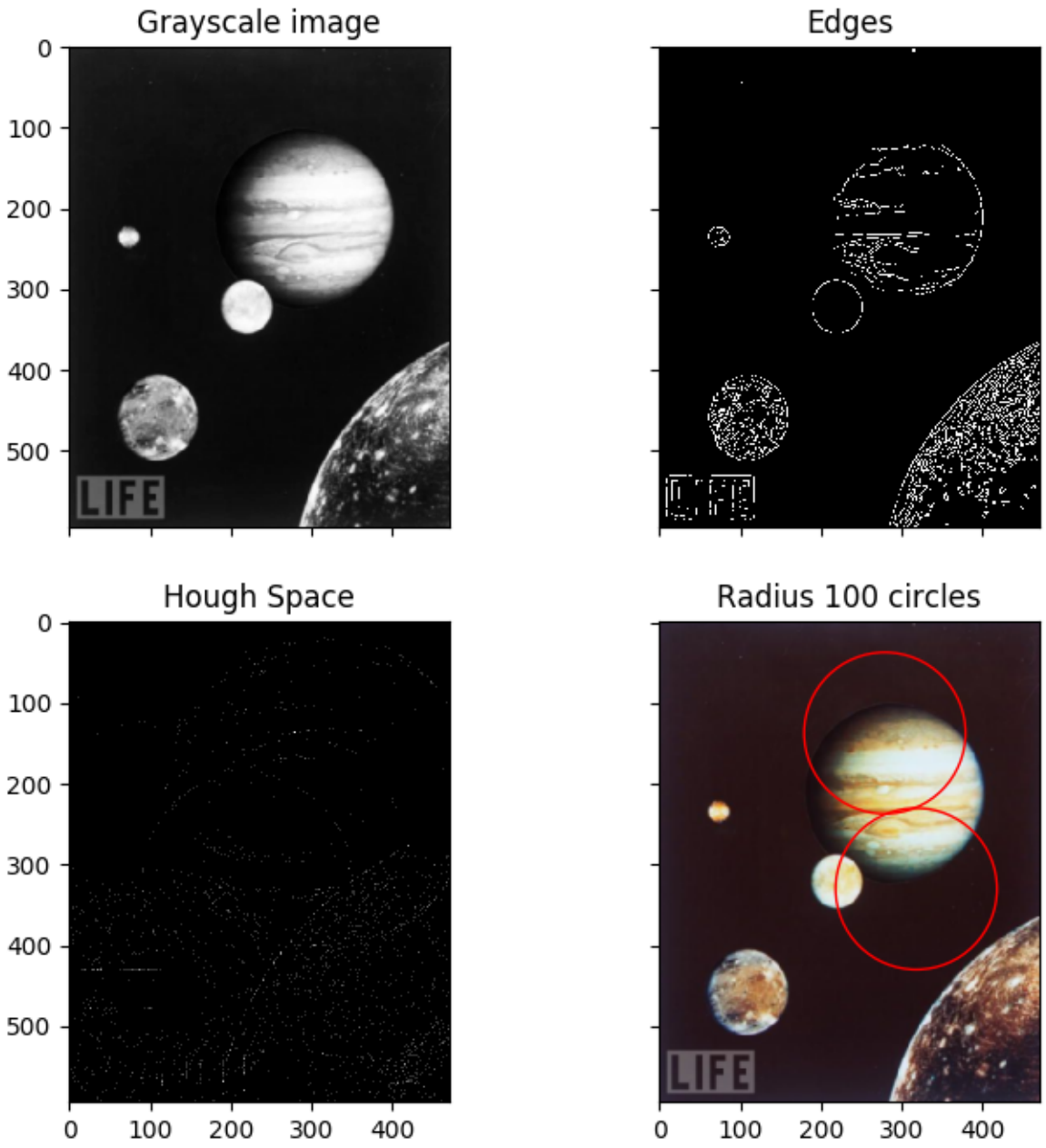

Figure 20: useGradient = 1

Hough Space The jupiter.jpg image's edge image has a lot of noise, especially within the circles. These noisy edges contribute to the vote and result in circles on the bottom right where the noise is highest. In this too, the results for useGradient = 1 are comparitively better than useGradient = 0, but not accurate. In this too the Hough space of useGradient = 0 is noisy.

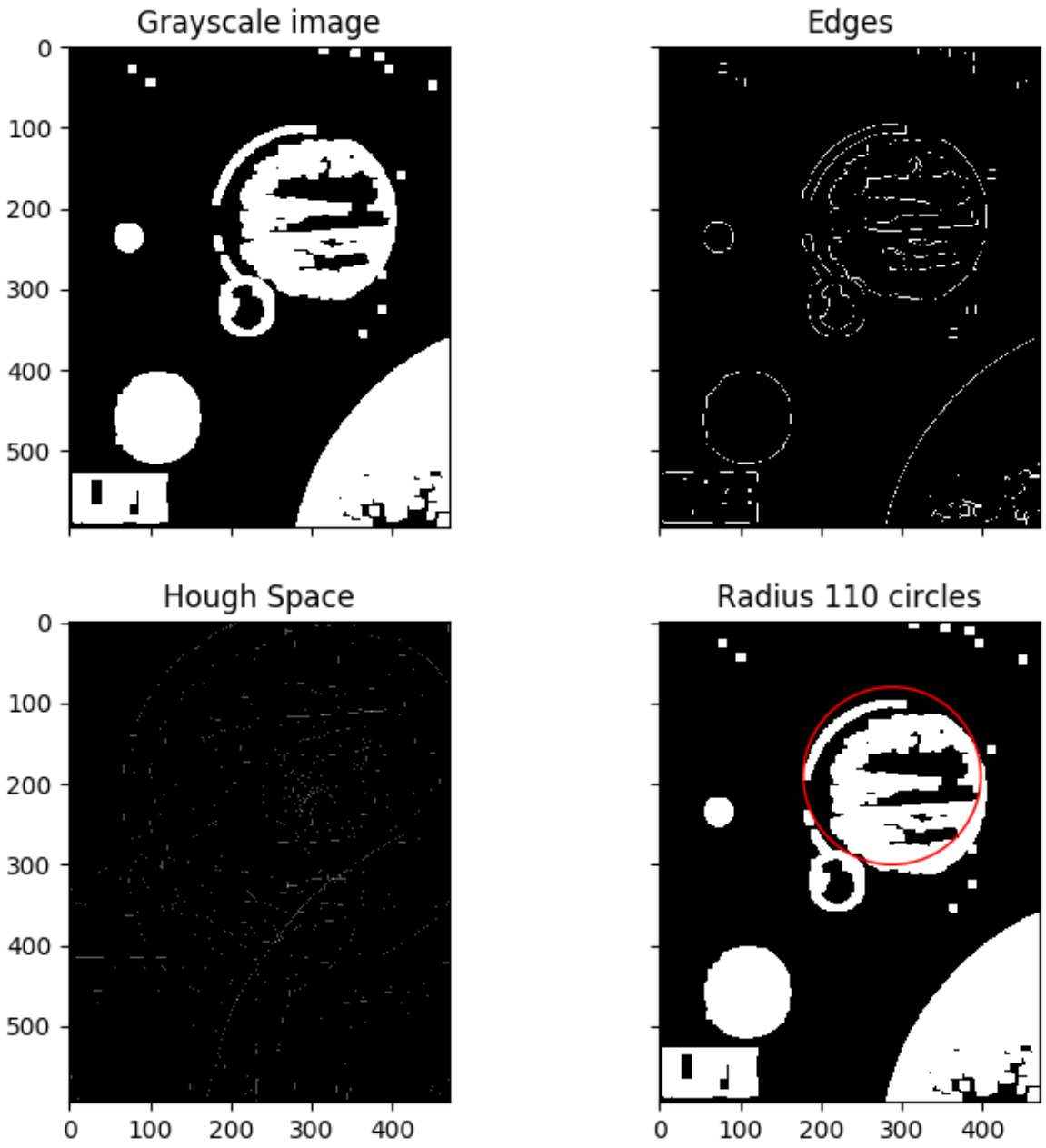

Figure 21: With dilation, useGradient = 1

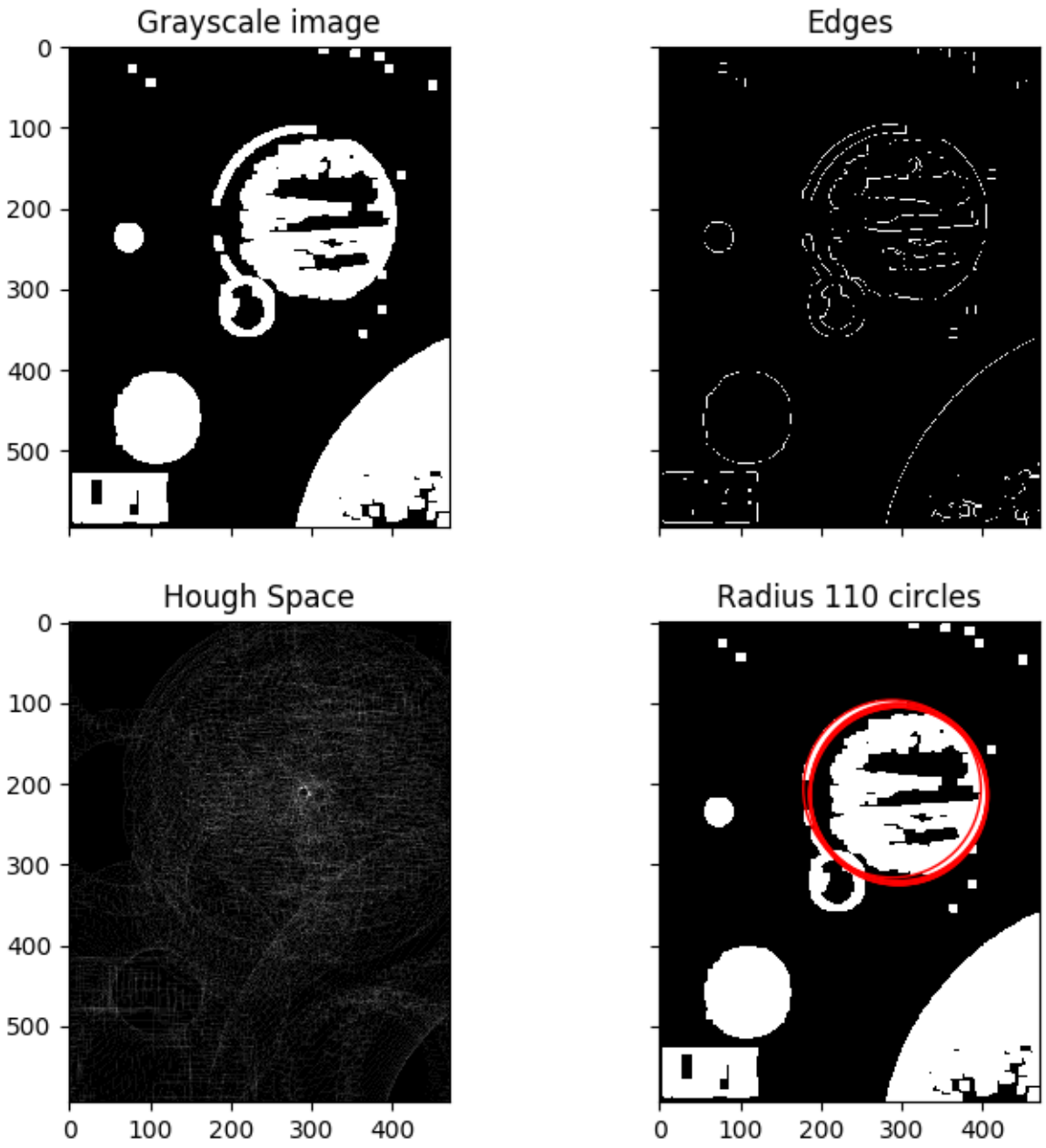

Figure 22: With dilation, useGradient = 0

Post processing after accummulator For dealing with noise, I have applied the morphological operator on the binary image for dilating circles. Dilation helps in filling the circles and therby giving cleaner edges. In this useGradient = 0, image has many circles overlapped because of the lack of gradient direction information whereas useGradient = 1 gives good results.

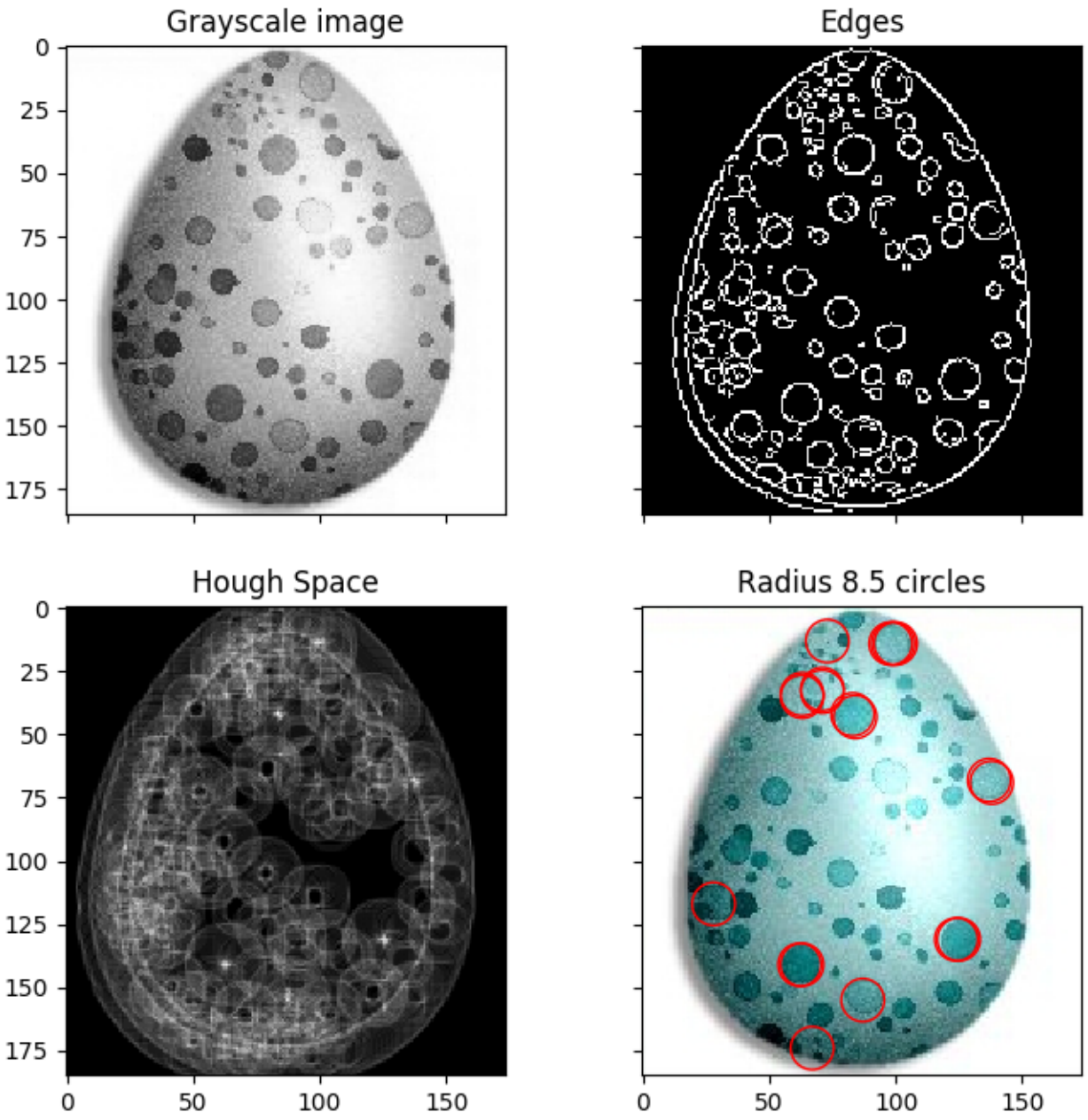

Figure 23: useGradient = 0, θ bin size = 1 degree

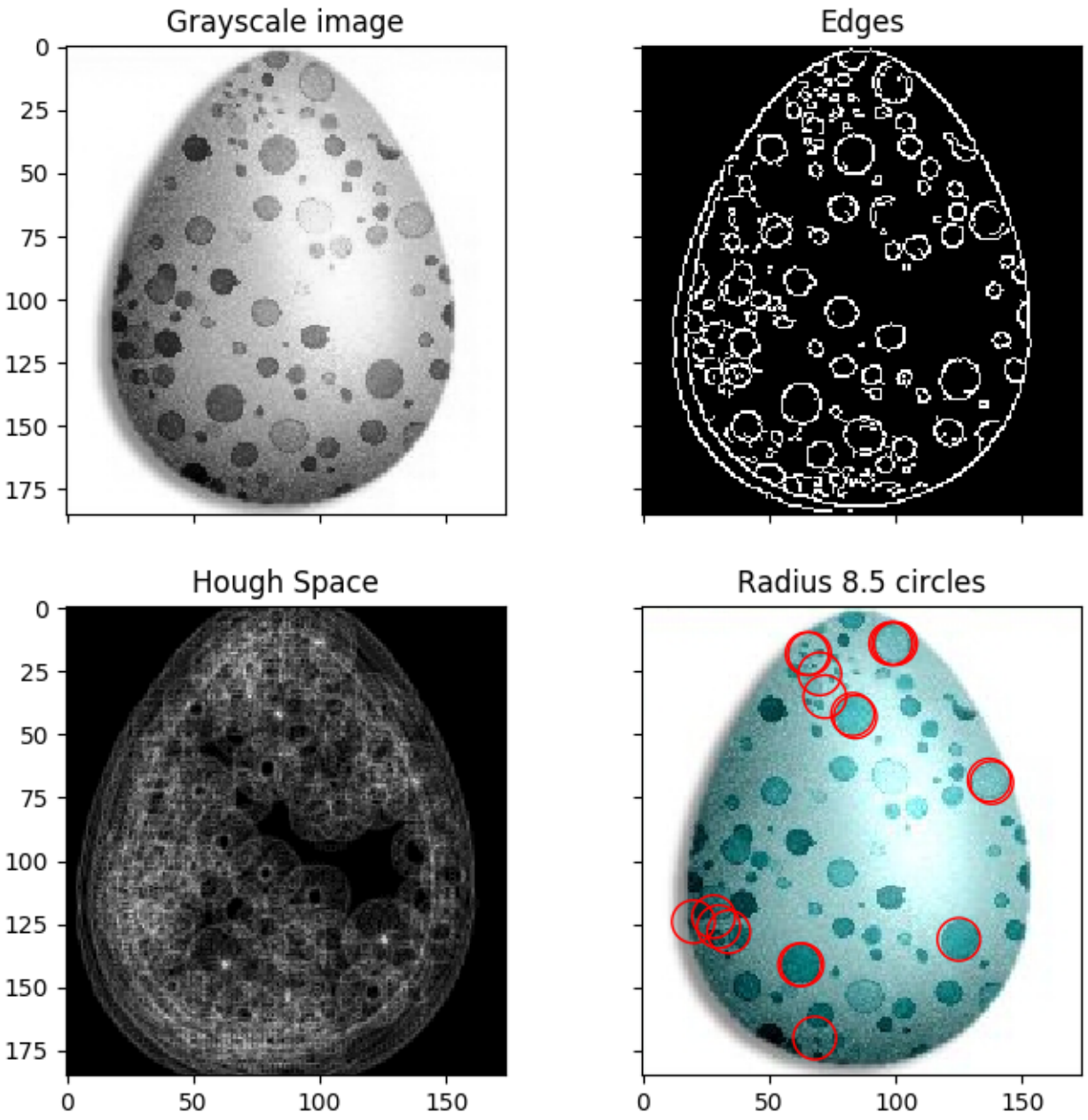

Figure 24: useGradient = 0, θ bin size = 10 degree

Impact of bin size A larger bin size of θ results in spreading out of centers of circles compared to smaller θ.

3. **Extra credits**

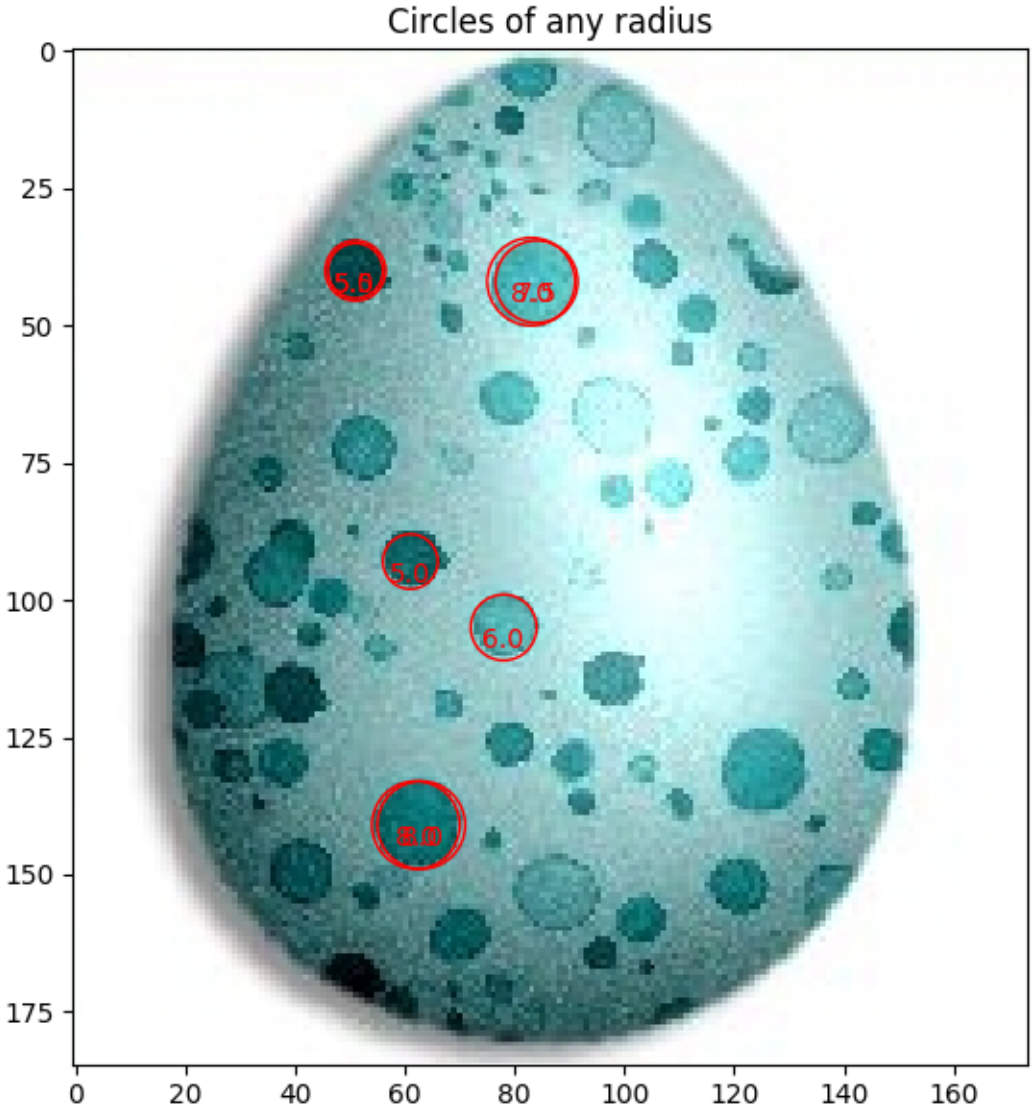

Figure 25: Circles of radius ranging from np.arange(5, 30, 0.5), useGradient = 1