CS 6476 - Computer Vision

Problem Set 2

Harsh Bhate (903424029)

Problem 1

1.

The filter bank comprises of two scales and 6 orientations. Consider a texture and a rotated version of the same texture. The features in the original texture and the rotated texture will point to the same point. This is because the filter bank covers almost all possible orientation and thus will generate similar response. Thus, the resulting representation is invariant to rotation.

2.

Initially, the k-means algorithm will randomly initialize the cluster to two random centers. Subsequently, each point "close" to a particular center will be assigned to that cluster. Note that in this case, "closeness" refers to a Euclidean distance (SSD Error). Itervatively, the cluster centre will converge to split the edge points along the vertical boundary. This is counterintuitive as human perception treats the clusters as two circles of different radii. However, the Euclidean distance heuristics creates cluster centers as shown in image.

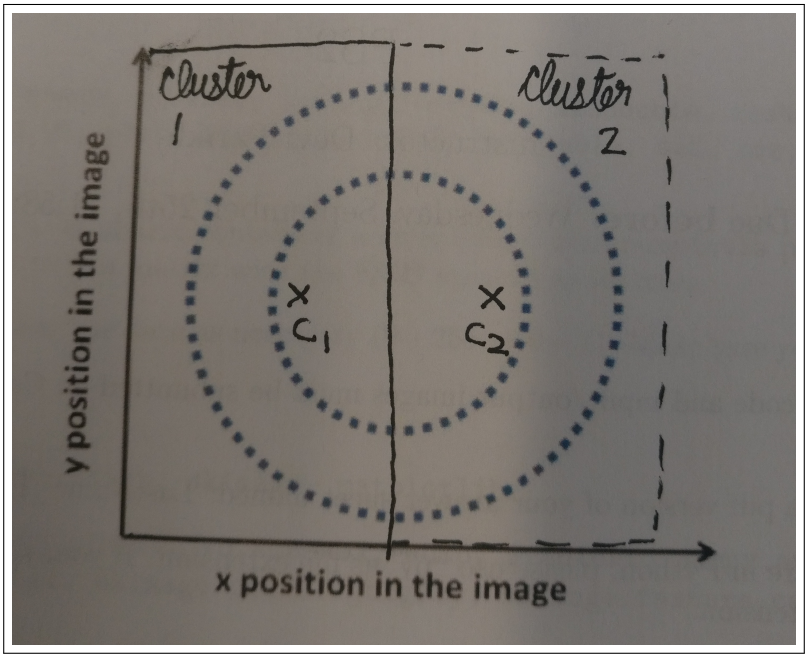

Figure 1: K-means Cluster

3.

Hough Transform in the discrete parameter space works by finding the bin with maximum number of votes. In case of continous vote space would employ finding the point with the largest intensity. To recover the model parameter hypothesis, graph algorithm would be a poor choice because it involves breaking weakest bonds. Clustering algorithm on the other hand relies heavily on spatial distance heuristics such as SSD and Euclidean distance. Such methods do not bode well with the ultimate aim for finding density. Mean-shift algorithm presents a viable option as mean-shift iteratively converges to the "sink" or the place of maximum density.

4.

The problem in question essentially boils down to clustering shapes. In the specific problem, we are given binary images and the objects (blobs) has consistent color property compared to background. Thus, the first trick is to generate a set of features that help give more information to clustering algorithms.

For feature generation, computing rough "centre" of the blob is helpful, it can be computed as:

```
def findCentre(blobPatch):
    """Find centre of blob
    Parameter
    ---------
    blobPatch : array_like
        Patch of blob
    Returns
    -------
    x : float
        Normalized centroid x-coordinate
    y : float
        Normalized Centroid y-coordinate
    """
    boundaryX = list()
    boundaryY = list()
    for every point in blobPatch:
        if point is boundary:
            boundaryX.append(point.x)
            boundaryY.append(point.y)
    midX = (max(boundaryX) - min(boundaryX))/2
    midY = (max(boundaryY) - min(boundaryY))/2
    return (midX/length of patch, midY/length of patch)

centerBank = [findCentre(blob) for blob in allBlob]

a = zeros(100,100)

for center in centerBank:
    a(center*100) = 1

def findShape(a):
    return kmeans2(a, nosOfShapeTypes)
```

Problem 2 (Using Python)

1. Color Quantization with k-means

(d).

Figure 2: Test image (RGB)

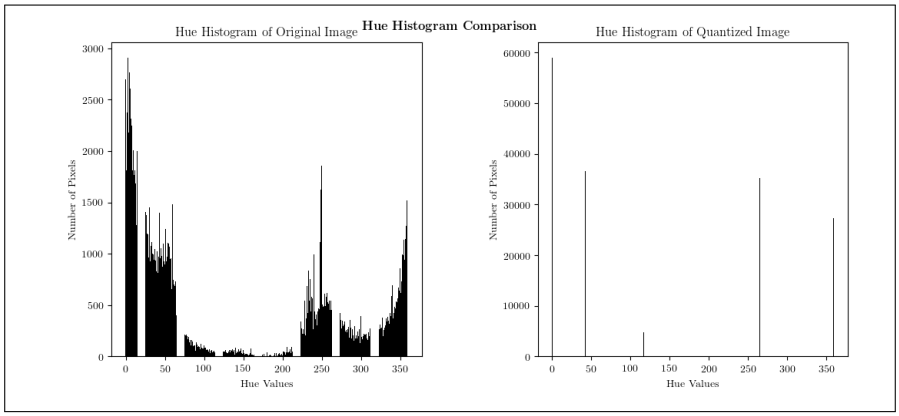

Figure 3: Hue historgram for original and Quantized image with $k = 5$

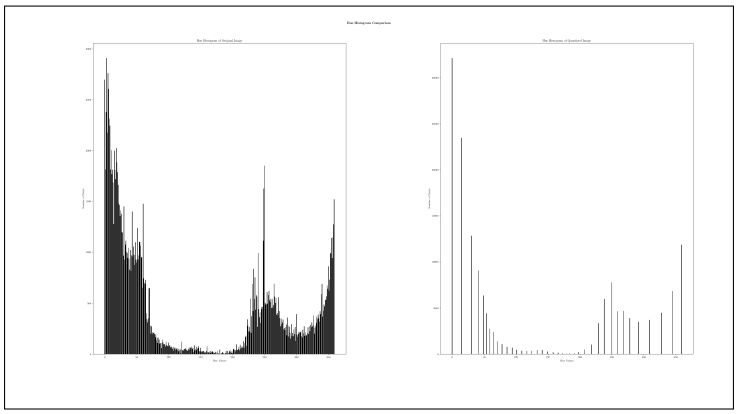

Figure 4: Hue historgram for original and Quantized image with $k = 5$

(e).

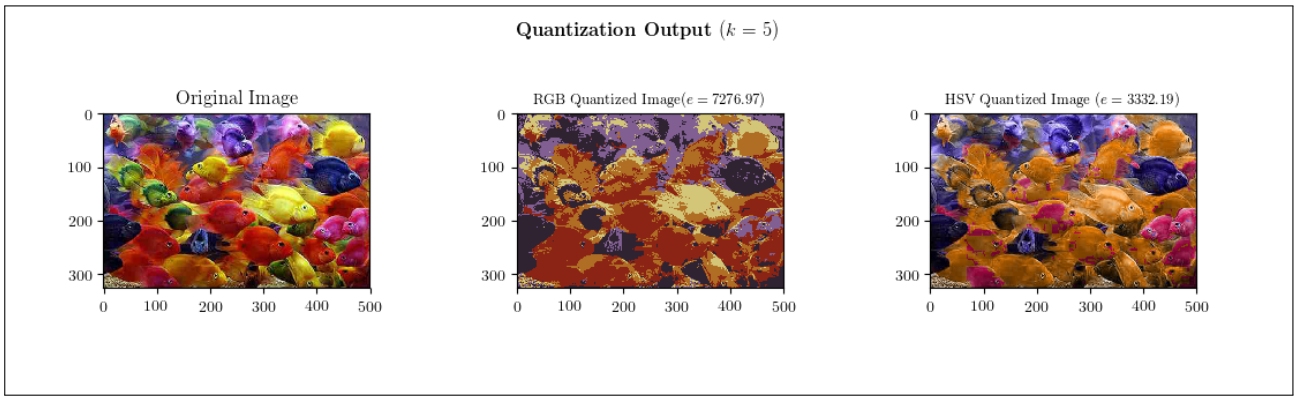

Figure 5: Color Quantization for test image with $k = 5$

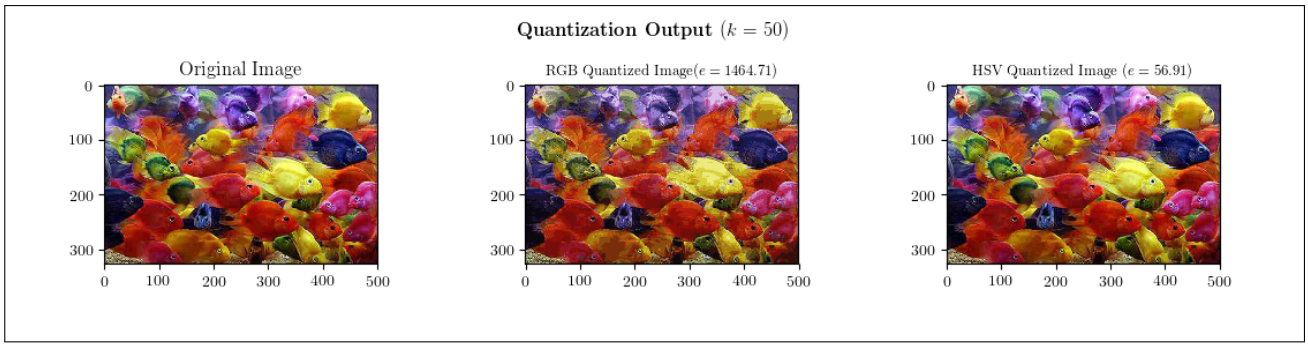

Figure 6: Color Quantization for test image with $k = 50$

(f).

Histogram

According to color theory, the Hue space is expressed in degrees from 0 to 360 with:

- Red between 0 and 60
- Yellow between 61 and 120
- Green between 121-180
- Cyan between 181-240
- Blue between 241-300
- Magenta between 301-360

A visual inspection of the image shows that red, yellow and blue from a majority of the image with some traces of green. The histogram **histEqual** shows a good estimate of the color distribution across the image. The left side values correspond to red while the rightmost correspond to magenta. The **histEqual** shows in good detail the presence of finer colors.
However, the histogram **histClustered** shows only the most critical colors in the images mirroring the visual examination. Especially the **histClustered** for k=5. As the number of cluster centre increase, the histogram shows more resolution in color distribution.

Quantization and Colorspace

For a lower number of clusters, the difference between RGB and HSV based quantization is immense. This is because RGB colorspace does not provide a smooth change in color values and the quantization process is hugely dependent on the heuristics of SSD. This results in Balkanization of colors creating uneven boundaries. HSV color space, on the other hand, provides a far superior performance because of relative proximity of colors resulting in a better contrast. However, this proximity leads to inverted color in patches of seemingly uniform color space.

Quantization and Cluster Points

As evident from a visual evaluation, a higher number of cluster points corresponds to a betweer visual image. This is reiterated by the lower error value. A simple answer for that is that the precense of higher number of color distributions creates a finer information map resulting in better reconstruction.

Quantization and Cluster Points

As evident from a visual evaluation, a higher number of cluster points corresponds to a betweer visual image. This is reiterated by the lower error value. A simple answer for that is that the precense of higher number of color distributions creates a finer information map resulting in better reconstruction.

Quantization and iteration

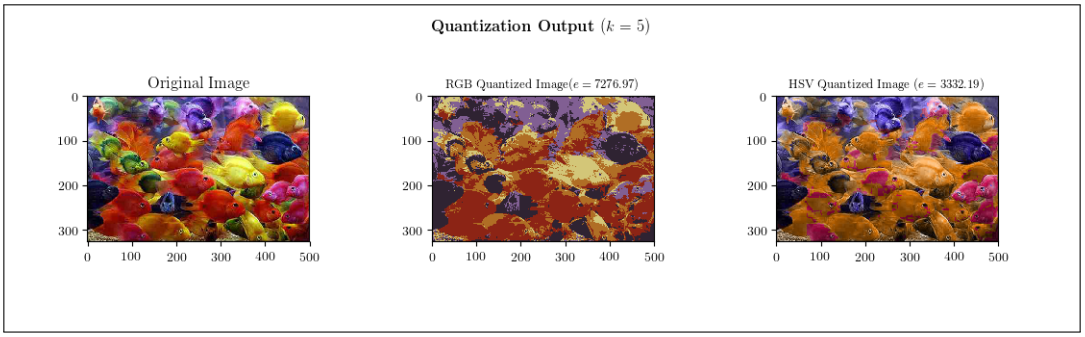

Figure 7: Quantized image with 10 iterations

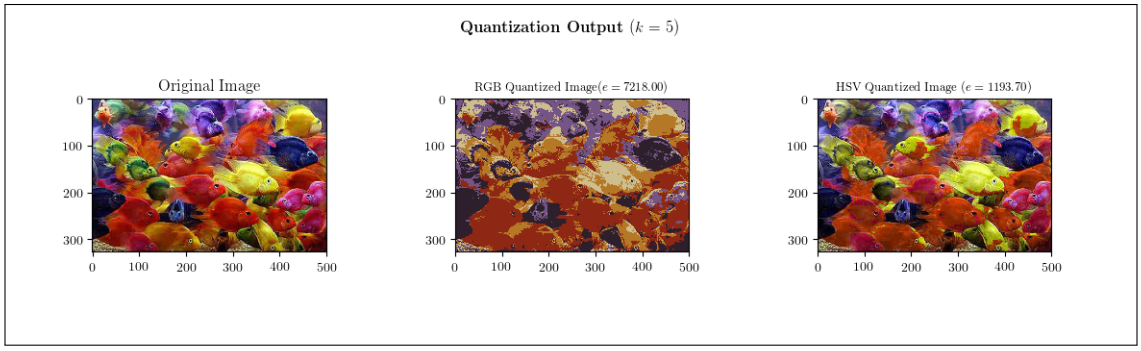

Figure 8: Quantized image with 25 iterations

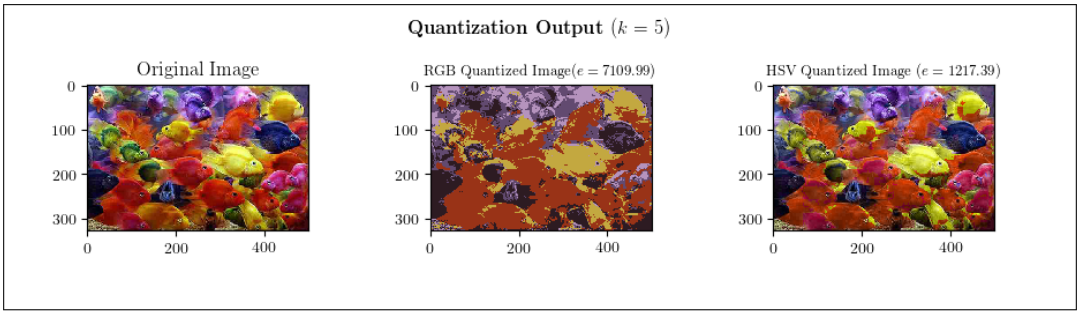

Figure 9: Quantized image with 100 iterations

Computationally, a SSD error decreased in the RGB clustering with number of iterations. This is because RGB is a cubic colorspace and a Euclidean/SSD heuristics works well on convergence. However, visually, the image quality degenerates as the edge information is lost. The HSV is a cylindrical colorspace and thus does not respond well to SSD heuristics despite increasing the number of iterations. However, the quality of color selection is far better with higher iteration because of the proximity of color aiding in better color localization.

2. Circle Detection with Hough Transform

(a).

- Read the input image and convert to grayscale
- Perform edge detection to determine edge points
- Perform hysteresis to imporve connectivity between the edge points
- Threshold the edge matrix to create true edge map
- Extract the spatial values of edge points
- Initialize an accumulator matrix with size slightly larger than the original image (or according to a preferred binning system)
- For every edge point

  To use gradient, compute the gradient from the edge matrix and extract the angle of gradient

  Otherwise, use a bank of angles

  Compute the vote index by using the hough transform equation for a given angle. Subsequently record and update the vote.
- Finally, locate the positions in the accumulator with maximum vote. Declare the indices of said points to be the centre of the detected candidate circle of radius r.

(b).

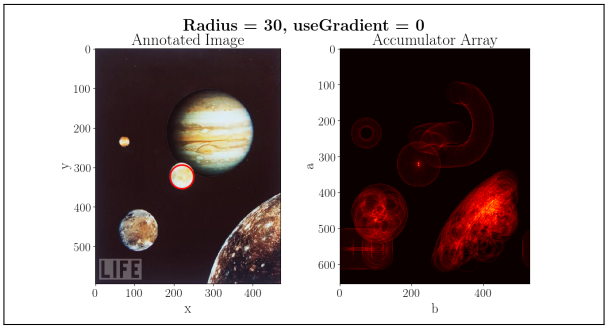

Figure 10: Test Image 1 without using gradient

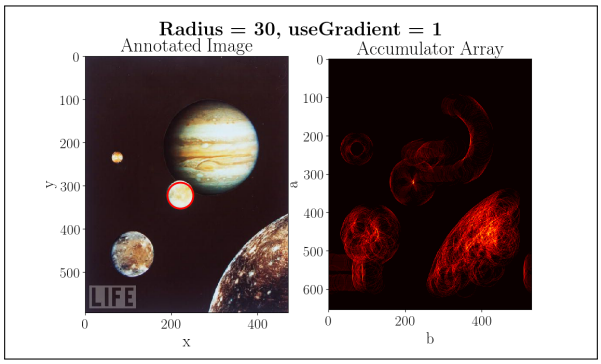

Figure 11: Test Image 1 using gradient

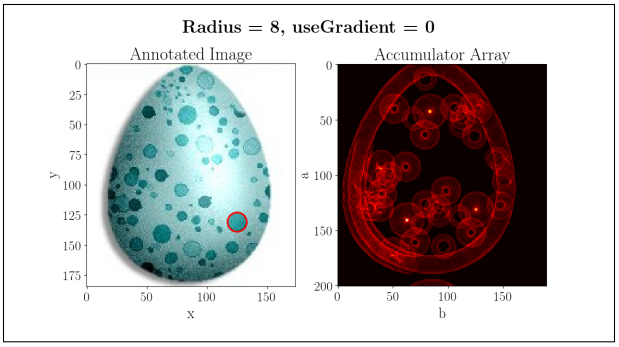

Figure 12: Test Image 2 without using gradient

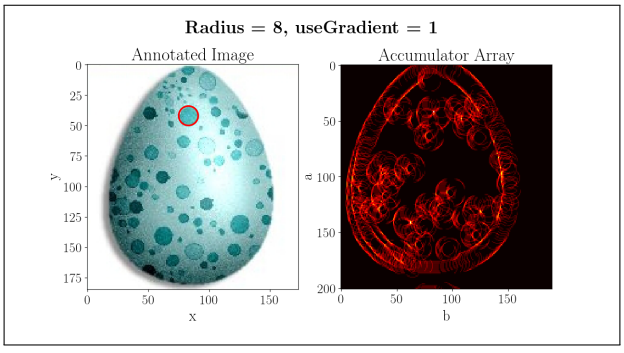

Figure 13: Test Image 2 using gradient

(c).

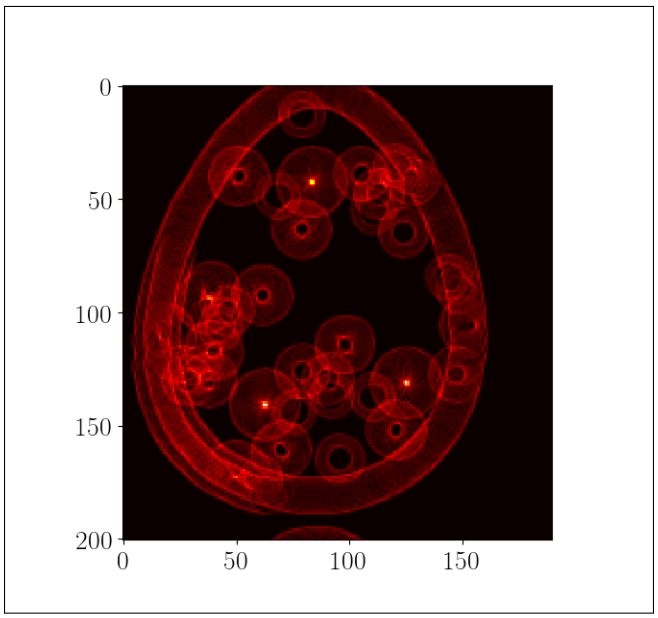

Figure 14: Hough Space Accumulator Array for Test image 2 computed without gradient

A point on the hough space corresponds to a line in image space. Also, geometrically, a circle is a collection of infinite lines. Thus, a circle on the hough domain is represented by a bright dot surrounding a "halo" of points. Another important thing to notice is the presence of edge effects when the object and foreground interact. This is because of the drastic change in gradient that registers in the edge map.

(d).

To reduce the impact of edge effect, gaussian filter was implemented to accumulator.

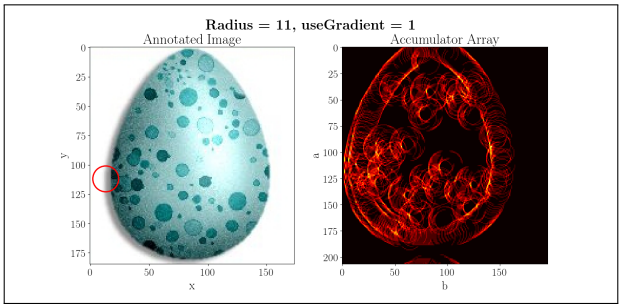

Figure 15: Output without post processing

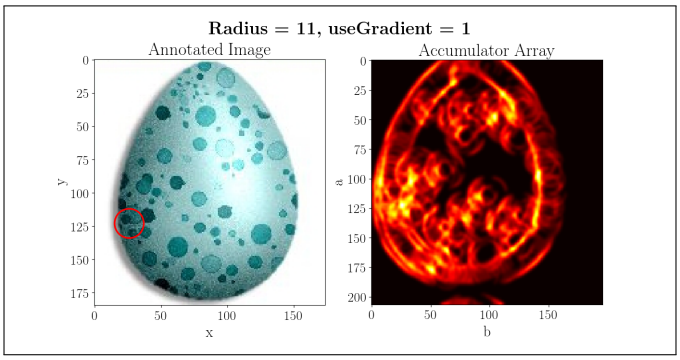

Figure 16: Output using post processing (Gaussian blur on accumulator)

To increase the number of candidate circle, the threshold to determine a circle was relaxed.

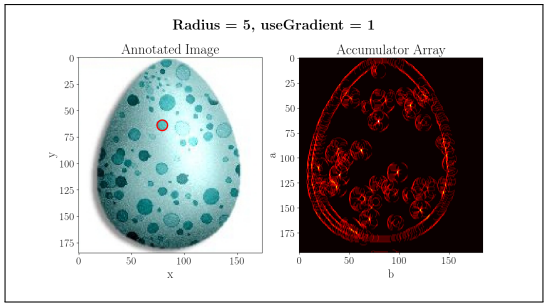

Figure 17: Output without post processing

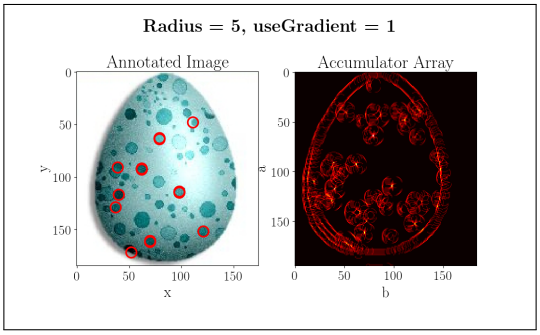

Figure 18: Output using post processing (Lower Thresholding)

(e).

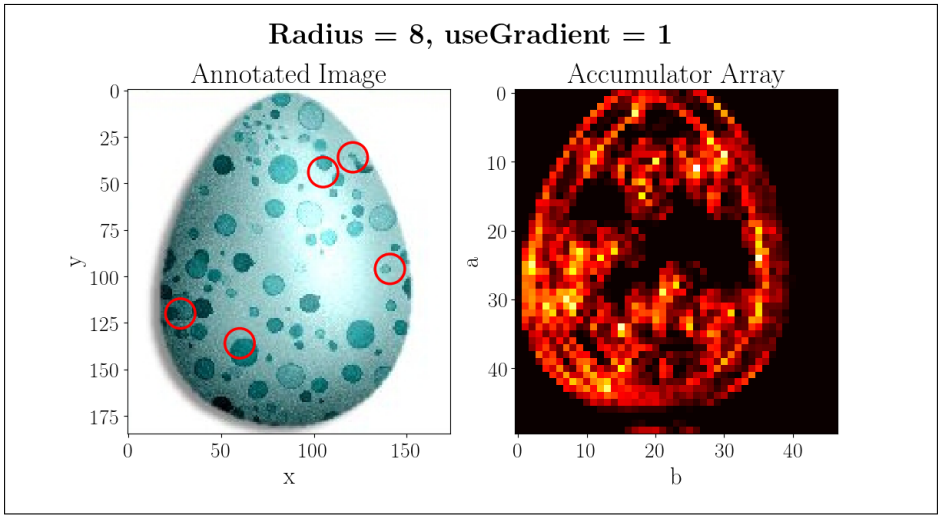

Figure 19: Low Vote Space Quantization

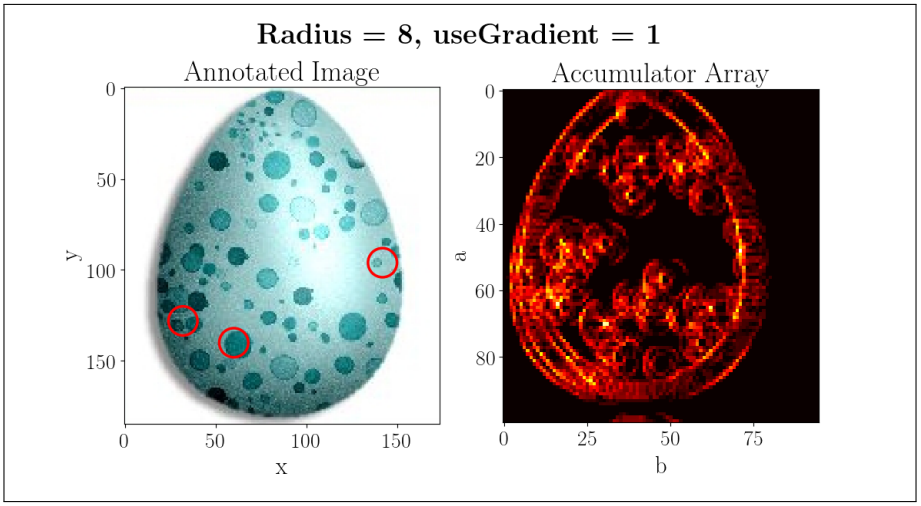

Figure 20: Moderate Vote Quantization

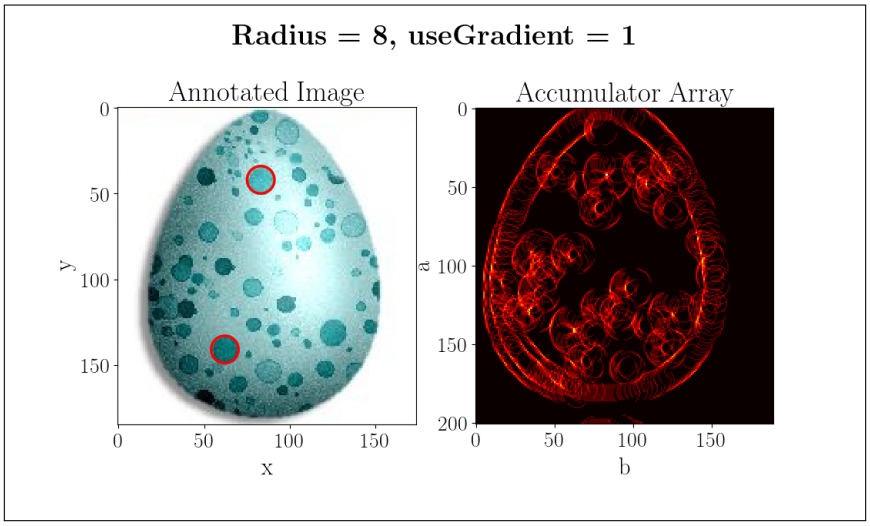

Figure 21: High Vote Quantization

Problem 3 (Bonus)

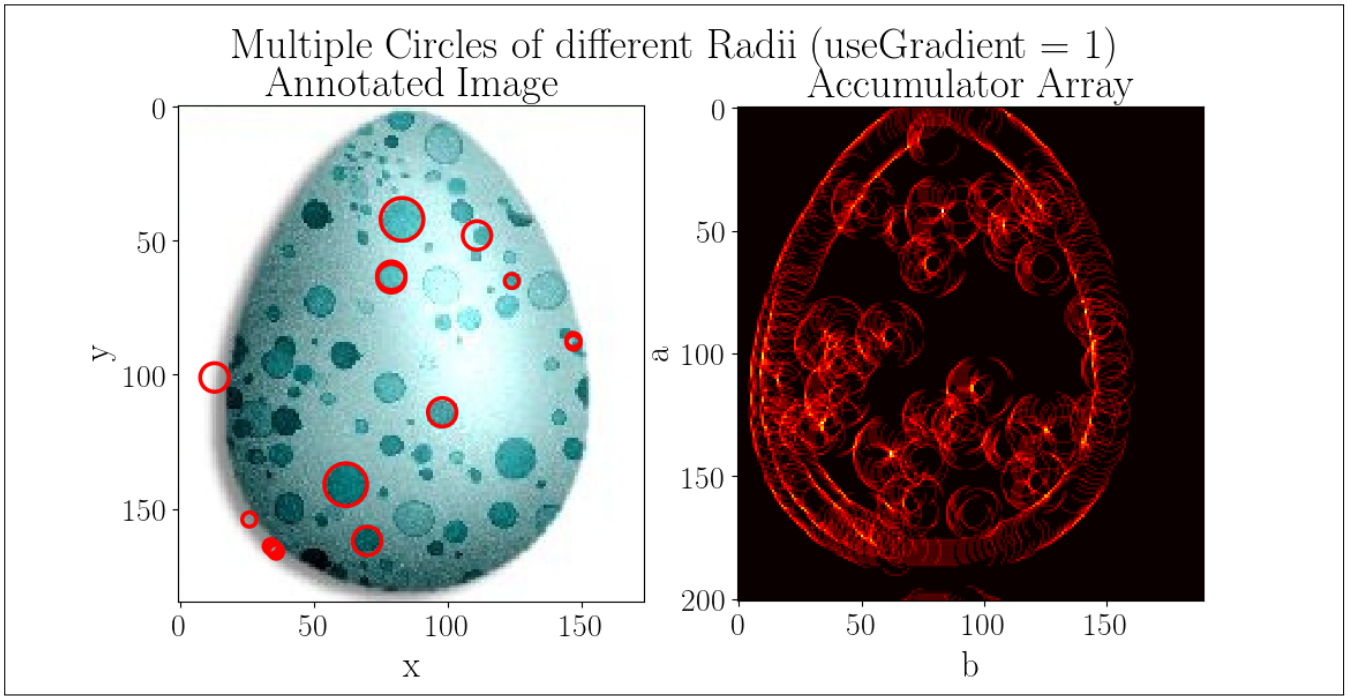

Figure 22: Detecting circles of multiple radius