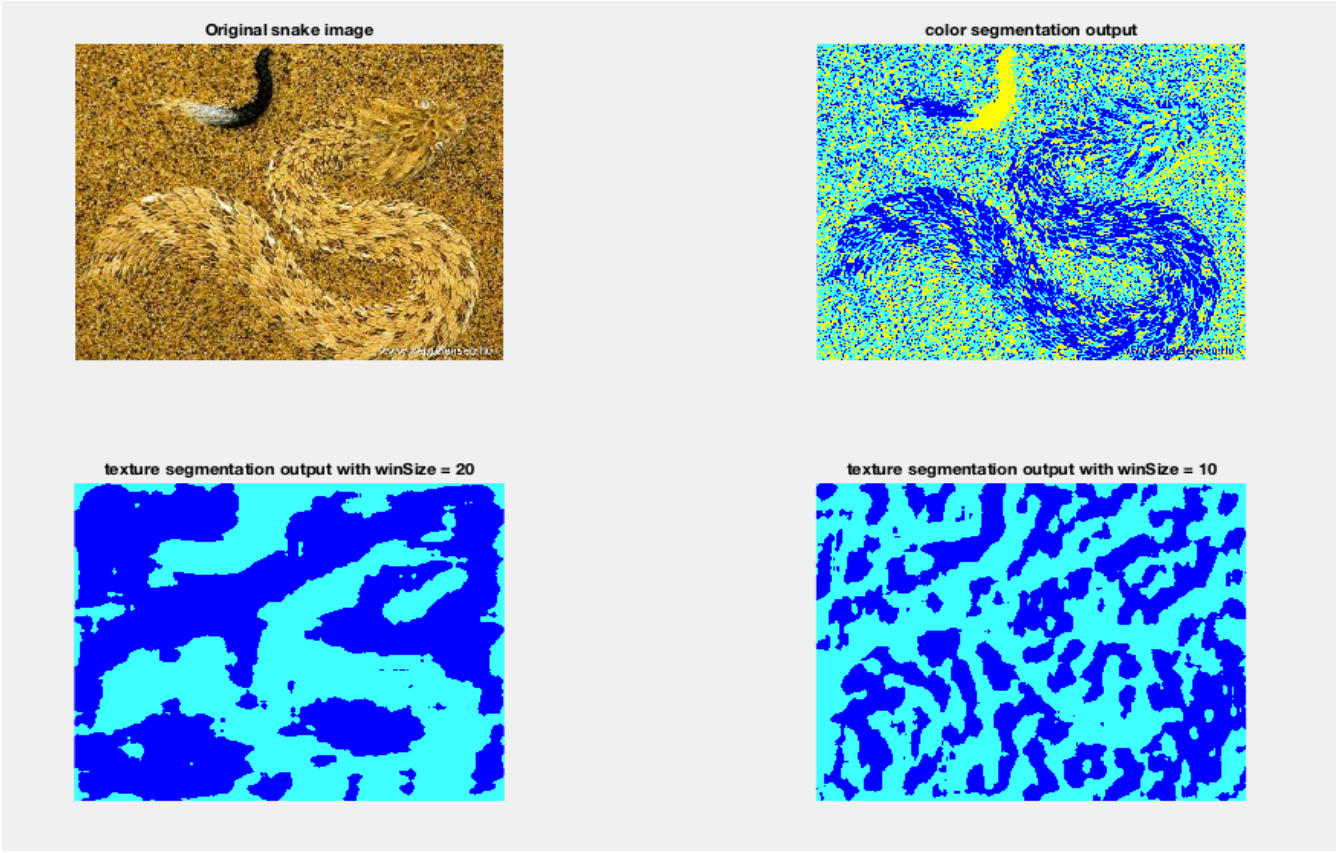
A2 Output Report

I. Image segmentation with k-means

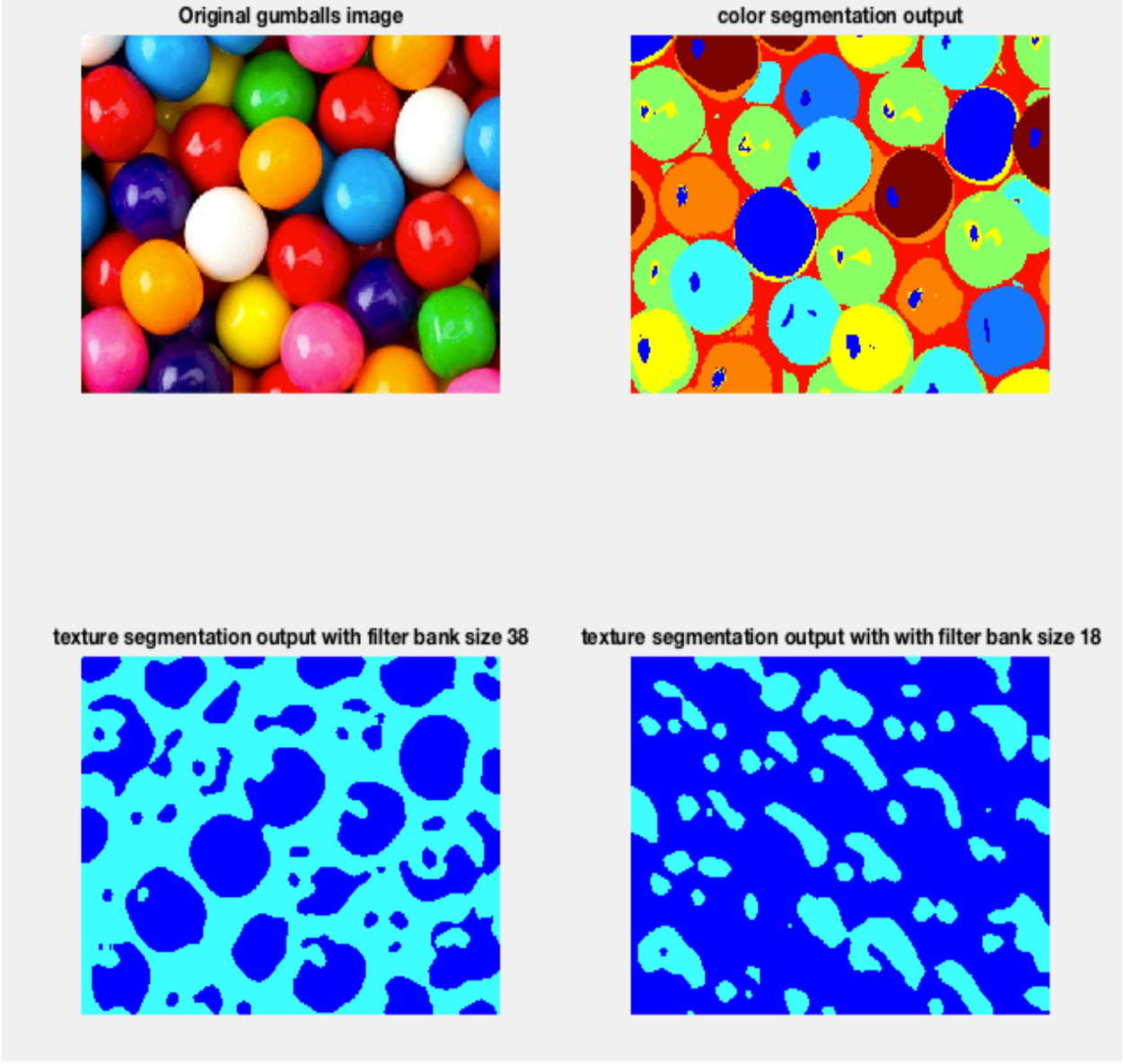
```
numColorRegions = 3; Mainly 3 colors in original image.
numTextureRegions = 2; Set as 2 to identify snake and background.
```



variant: window size

As can be told from the images above, the output with window size 20 has a better output than window size 10. That's because the window size 10 is too little for the size of the snake body in image. The texture segmentation result tends to be to separated. While with window size equals 20, the output shows the snake body well, but the neck and the tail parts of snake sticks together because of the larger window size.

```
numColorRegions = 8; Since there are mainly 8 colors.
numTextureRegions = 2; Since there are mainly 2 kinds of textures.
winSize = 10; Relatively small to separate each gumball.
```



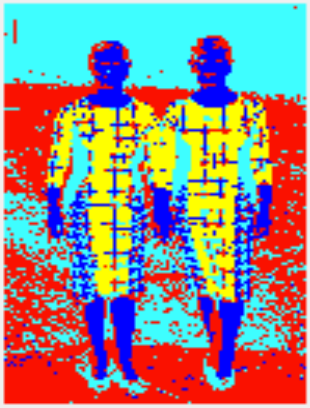
variant: filter bank size

I used all the 38 given filters for the third output but only the first 18 filters for the forth output. The first 18 filters are derivatives of Gaussian filters with different scales and orientations. After comparison, we can tell the full filter bank gives a better output since the full filter bank also provide second derivative of Gaussian filters with different scales and orientations, the output is more concrete.

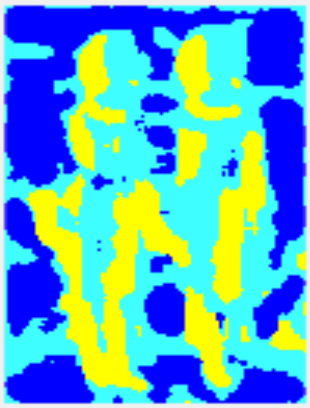
```
numColorRegions = 4; Since there are mainly 4 colors.
numTextureRegions = 3; To identify skin, hair and background.
winSize = 13; Relatively small to identify human skin.
```

Original twins image

color segmentation output



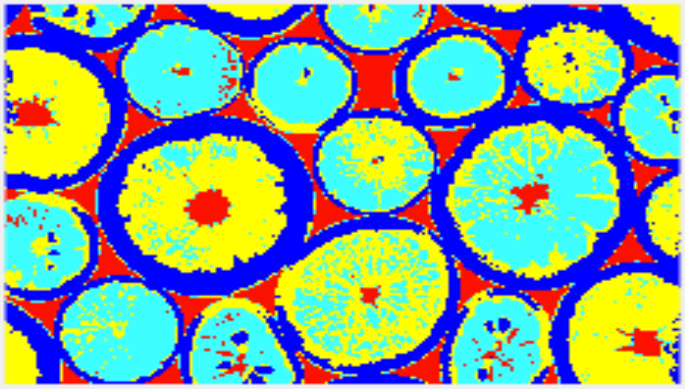
texture segmentation output



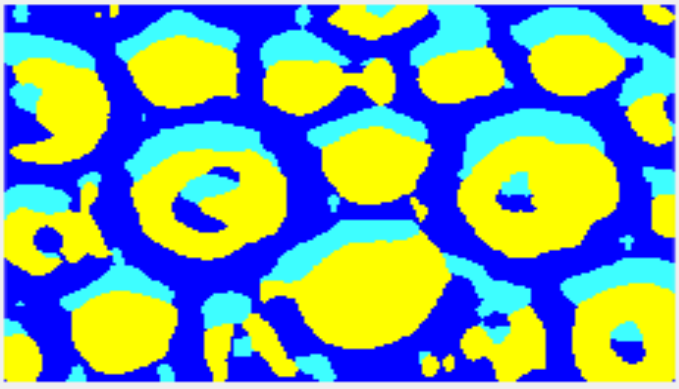
```
numColorRegions = 4; Since there are mainly 4 colors.
numTextureRegions = 3; Since mainly three textures.
winSize = 13; Relatively small to identify each lemon.
```

Original chosen image

color segmentation output



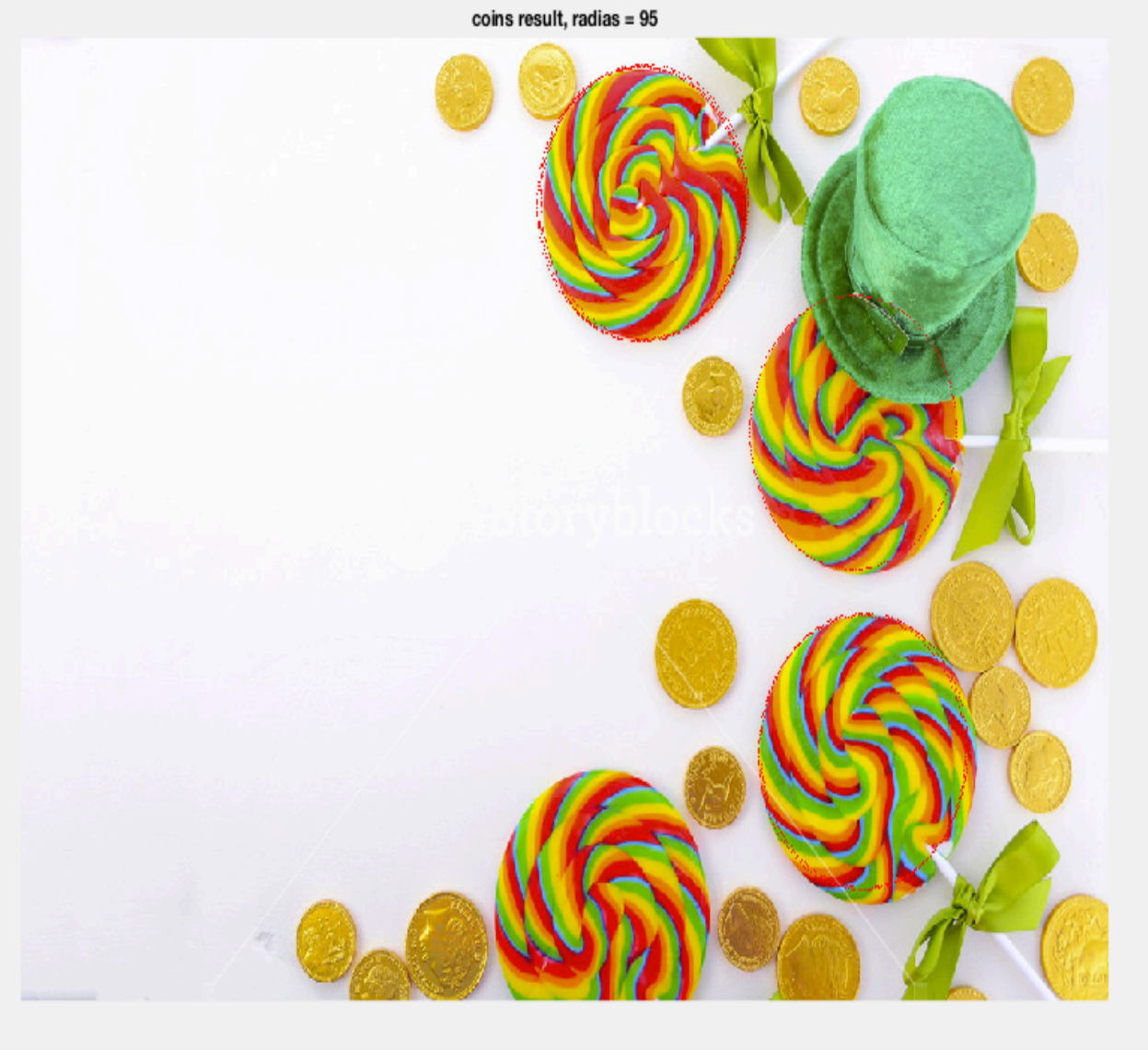
texture segmentation output



2. Circle detection with the Hough Transform

(a)
First of all, use canny to do edge detection. Then for each pixel value not equal to zero, iterate through each angel. Get x and y coordinates and covert them to binned coordinate values. Count the votes in binned accumulator matrix. After counting, find the coordinates in binned accumulator matrix with vote value greater than threshold, convert coordinates values to the real values in original image. Write out the real coordinates.

(b)
Coins output
```
edge(grayscale, 'canny', 0.33)
```



Planets output

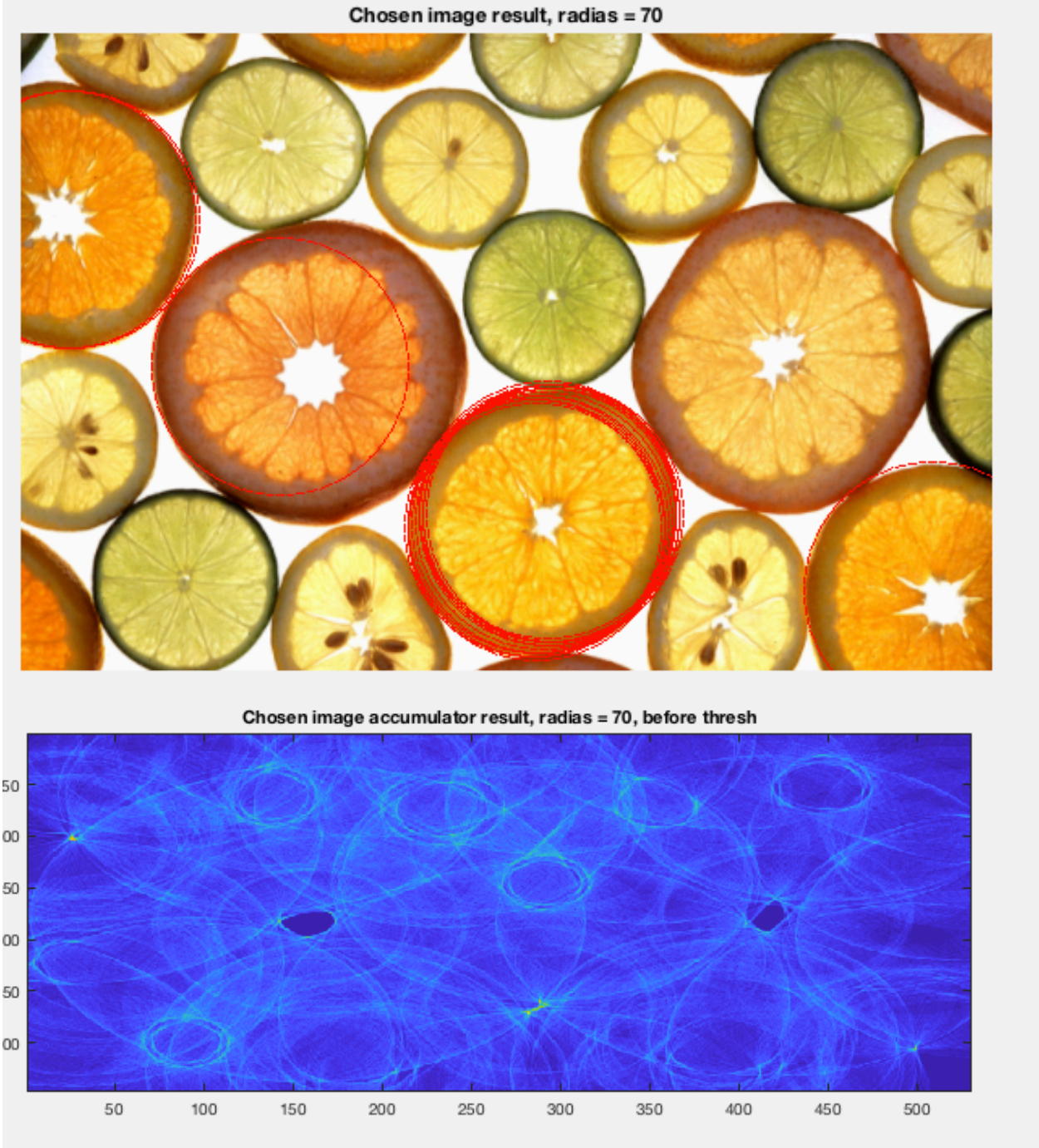
```
edge(grayscale, 'canny', 0.7)
```

Chosen image output

```
edge(grayscale, 'canny', 0.7)
```

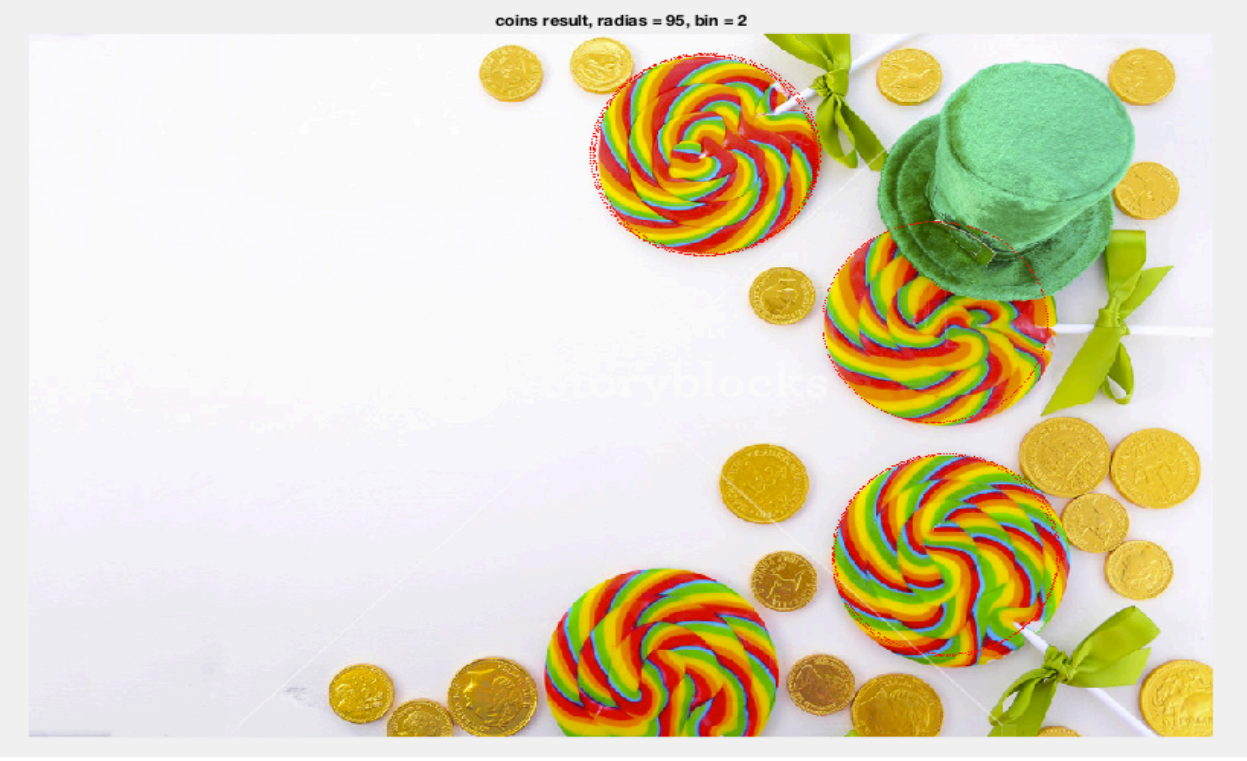
By testing I found that using the value of maxVote *2/3 may not give optimal result but always give a decent output. The function find the max vote value in accumulator automatically and calculate the dynamic threshold value.

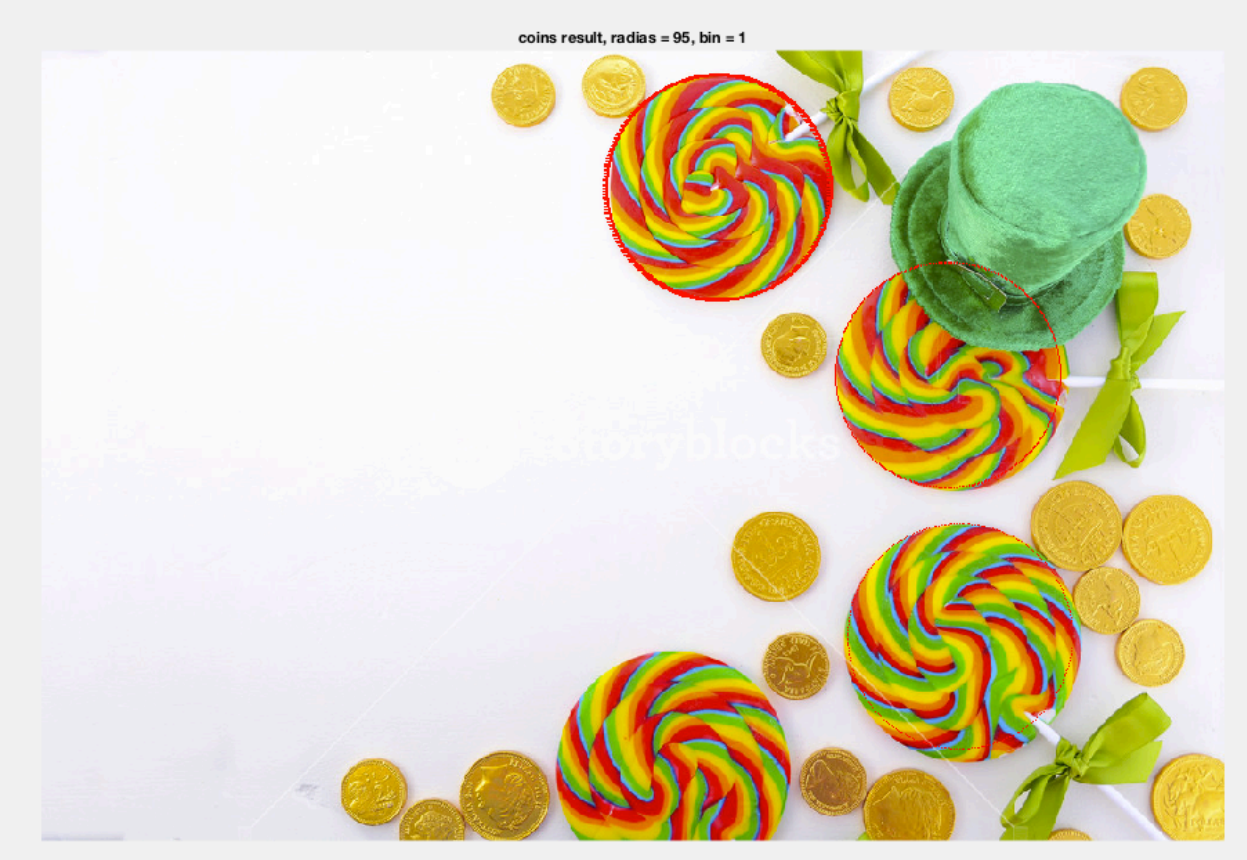
(c)



We can tell from this accumulator matrix result that the peak parts of it are the potential centers of circles. After thresholding, most of them will be gone and the centers we really want will be left.

(d)





As can be told from images above. The bin size of 2 has less duplicate circles on the top coin than the one with bin size 1. That's because the binning combine some votes to extremely close centers to together, which makes the result more accurate.

Extra credit

Extend your Hough circle detector implementation to detect circles of any radius. Demonstrate the method applied to the test images.

function `detectAnyCircle` in `detectAnyCircle.m`

Voting recorded in a 3d matrix. Iterate each possible radius, angel and store the votes in 3d matrix. Get the points with large votes as centers. At last draw the circles out. Output of search circles in image ‘test.png’.

Original image

Detect any circles output