CS 6476: Computer Vision, PS2

Trace Tidwell

September 27, 2019

1

1. The resulting representation will be invariant to orientation, because the filters capture the texture at 6 different orientations. So rotating an image by 45 degrees (or some multiple thereof) would capture the same representation, just in a different order than before.

2. Assuming we are using Euclidean distance as the metric, the clusters are most likely to split the two circles diametrically, with roughly half the points from each circle in one cluster and the other half in the other. This is simply due to the nature of using Euclidean distance. It prioritizes small distances and penalizes large ones. Often this is what is desired, but in this case, there is a clear structure, and a different method should be used.

3. Mean-shift is a non-parametric model, meaning it implicitly models the distribution, so it would not be useful in trying to recover the underlying distribution. k-means (and mixtures of Gaussians) is a parametric model of the density function. As such, we could sample from the model to recover, or perhaps recreate, a representative parameter hypothesis space. I do not see how normalized cuts could be used in this manner.

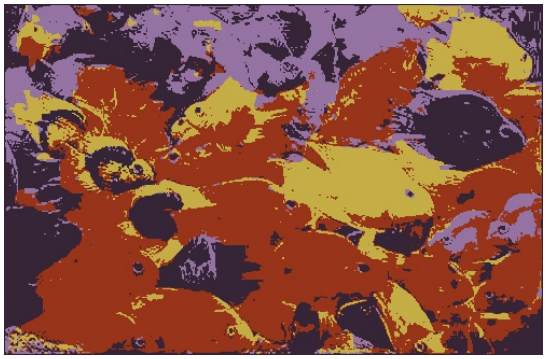
4.

2

1. Figure 1 below shows the original fish image along with the RGB image quantized across all three color channels and an HSV image quantized in just the hue space for $k = [4, 12, 20]$. It becomes immediately apparent that the results of segmenting the image in the hue space are much different than those of segmenting the image in RGB space. For $k = 4$, the hue segmented image much more closely resembles the original than the RGB segmented image. This makes some intuitive sense, because, even though the hue somewhat represents the basic RGB color, the saturation and value still play a major role in how the color is displayed. In RGB space, the hue, saturation, and value are captured in the RGB pixel value. So when we cluster RGB values, we are essentially choosing k colors, each with a set saturation and value. But when we cluster hue values, we are still choosing k colors, but we are allowing for the full range of saturations and values, so a wide range of those k colors can still be displayed. This is really demonstrated in Figure 1(d), where $k = 12$. At this point, the hue segmented image is nearly indistinguishable from the original, whereas with RGB segmentation, it is impossible to recreate the vibrant, multi-colored original with only 12 colors. At $k = 20$, the RGB segmented image starts to resemble the hue segmented image for $k = 4$.

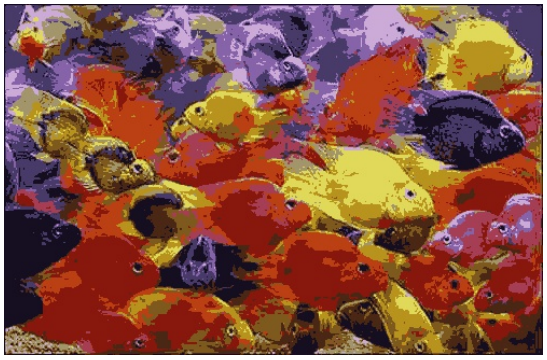
(a) Original Image

(b) Hue Segmentation, k=4



(c) RGB Segmentation, k=4

(d) Hue Segmentation, k=12



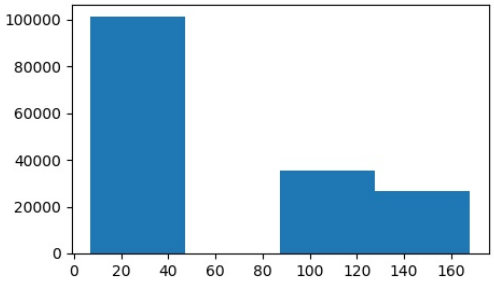
(e) RGB Segmentation, k=12

(f) Hue Segmentation, k=20

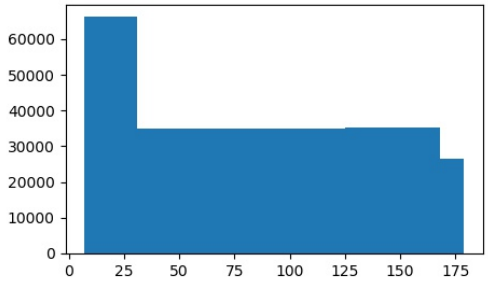
(g) RGB Segmentation, k=20

Figure 1: Hue and RGB Segmentations for Various k

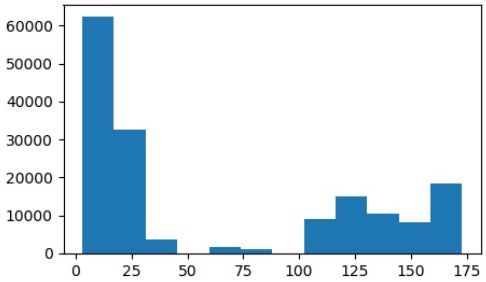
Figure 2 below shows histograms for different values of k. In each pair of images, the left histogram was created using k bins of equal width, while the right histogram was created using the cluster centers found in hue segmentation to define the bins. Though the general shapes match for each pair, there are some distinct differences. For each value of k, the equal binning method has empty bins, or bins to which no pixel values are assigned. When using the cluster centers, there are some bins with relatively few instances, but none of them are empty. We can also see that, for all three pairs, the first bin in the equal binning histogram is two distinct bins in the cluster center binning histogram. From the histograms, we can also somewhat confirm our statement from above that the hue segmented image with $k = 12$ looks very similar to the original. The main difference in Figures 2(d) and 2(f) come between hue values of approximately 40-105. When $k = 12$, this range accounts for 4 bins. When $k = 20$, this range accounts for 10 bins. So, when adding 8 more bins, 6 of them were added here. However, these points only account for 5% of the total points when $k = 12$ and 6% of the total points when $k20$. These means that, even though the clusters are getting more granular, not many points are being clustered much differently than before with fewer clusters. The results is that we should have a difficult time discerning differences when viewing the segmented images, which is true in this case.



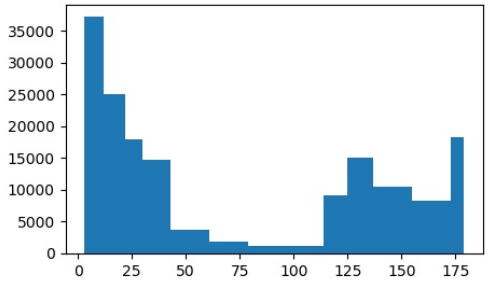
(a) Equal Bins, k=4



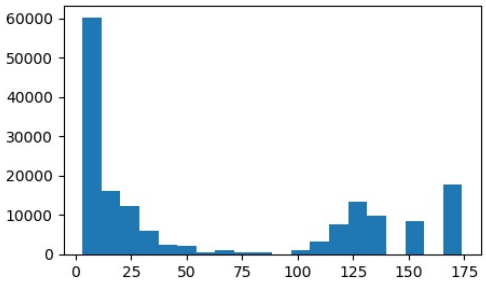
(b) Cluster Center Bins, k=4



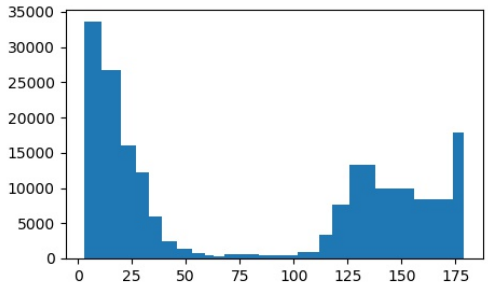
(c) Equal Bins, k=12



(d) Cluster Center Bins, k=12



(e) Equal Bins, k=20



(f) Cluster Center Bins, k=20

Figure 2: Histograms with Equal and Cluster-Center Bins for Various k

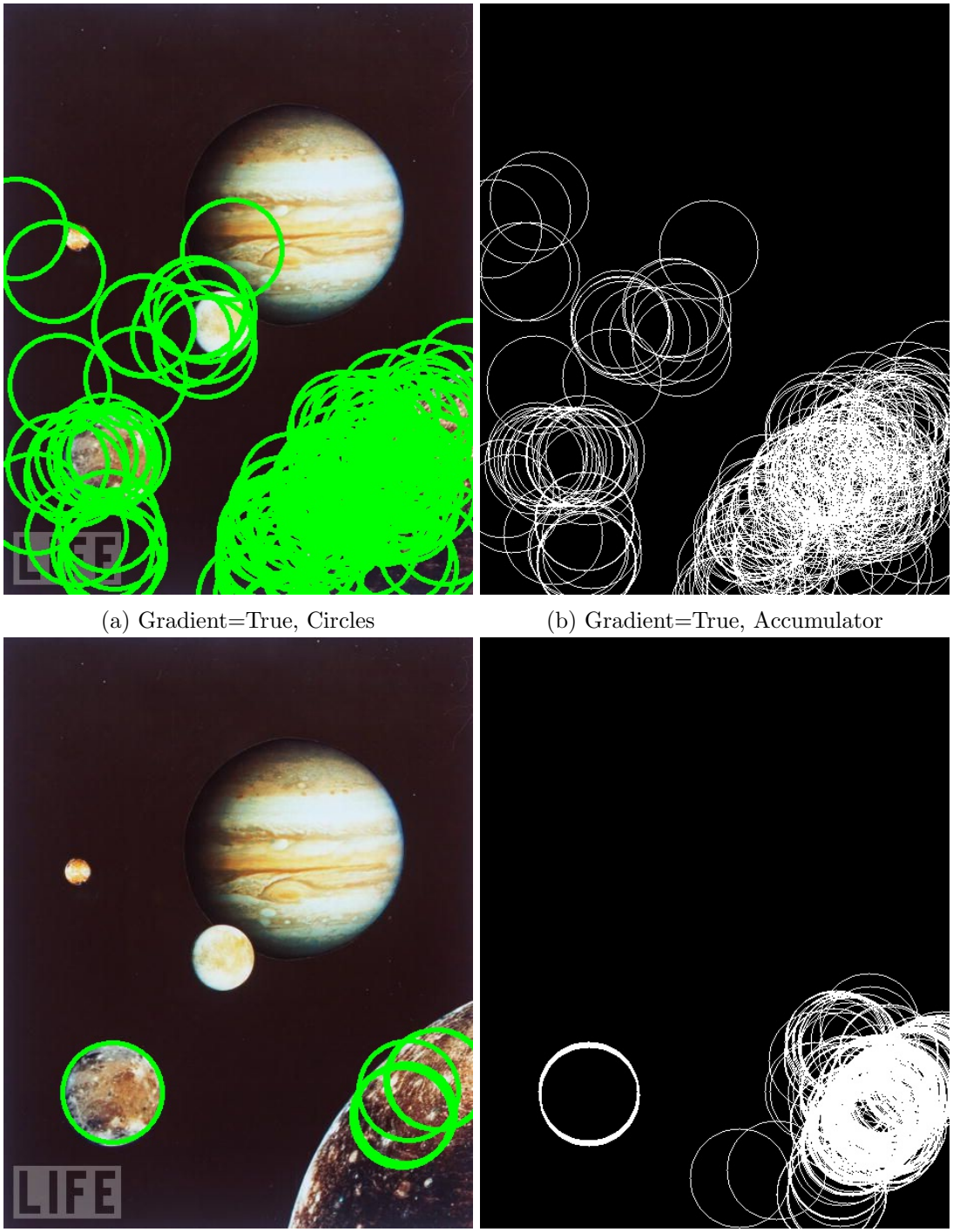
2. (a) The method takes as arguments a color image, the radius to be tested, a flag indicating whether or not to use the gradient of the image, bin sizes for theta and circle centers, and min and max values to be used in the Canny Edge detector. First, the accumulator array and a tracker array, which tracks which points voted for which center, are initialized. If the image passed to the method was color, it is converted to grayscale. Then the Canny Edge detector is run to detect edges. From the results of the edge detector, all remaining pixels whose values are 255 are edges, and they are kept. If the gradient flag is set to True, a convolution in the x- and y-direction is performed to get the gradient. Next, we iterate through every pixel kept after edge detection. If the gradient flag is set to True, we calculate theta from the x and y gradients for that pixel. If not, we iterate through values of theta as specified by the theta bin size argument. Once we have theta, we can calculate a and b, which represent the center of the circle. We then round a and b based on the circle centers bin size argument and "vote," or log the value of the center and radius. If the unrounded center values lie within a certain percentage of the center of the bin, it will be allowed to "vote" in both directions. We also track which centers were voted for by which points for postprocessing later on if needed. After all the edge points have been evaluated, the accumulator and tracker arrays are returned.

   (b) The first image we applied our Hough Transform function to is the image of Jupiter shown in Figure 3 below. As can be seen, there are 4 distinct circles of varying radii, along with another object in the bottom right corner. To demonstrate the effectiveness of the Hough Transform function, we experimented with different hyperparameters and visually inspected the results until finding some that seemed reasonable. We chose to find all circles with a radius, $r = 50$. We set the min_val and max_val arguments of the Canny Edge detector to 300 and 500, respectively.

Figure 3: Jupiter

Figure 4 below shows the results of running the Hough Transform with the hyperparameters outlined above twice: once using the gradients calculated from the image itself to calculate the

direction of the gradient, and once cycling through angles at a predefined step sizes. When using the gradient, we kept all centers that were within 3 votes of the top vote-getter for display and all centers within 10 votes of the top vote-getter to show in the accumulator array. When not using the gradient, we kept all centers that were within 2 votes of the top vote-getter for display and all centers within 3 votes of the top vote-getter to show in the accumulator array. We can see very clearly that the results look much better when not using the gradients derived from the image itself. These seems somewhat counterintuitive at first, but as we dig a bit deeper, we start to see why this occurs. When using the gradients of the image itself, the most votes any center got was 6, whereas when not using the gradients, the most votes any center got was 27. Even though the number of centers was approximately the same between the two methods, when using the gradients, they all received between 1-6 votes. When not using the gradients, the votes are more spread out, which allowed certain centers to clearly be better than others. We can see this with how many fewer circles there are but also how much more concentrated they are in the accumulator array. My hunch is that noise causes the gradients to be off by just enough to weaken the signal, causing us to be unable to find the true center.



(a) Gradient=True, Circles

(b) Gradient=True, Accumulator

(c) Gradient=False, Circles

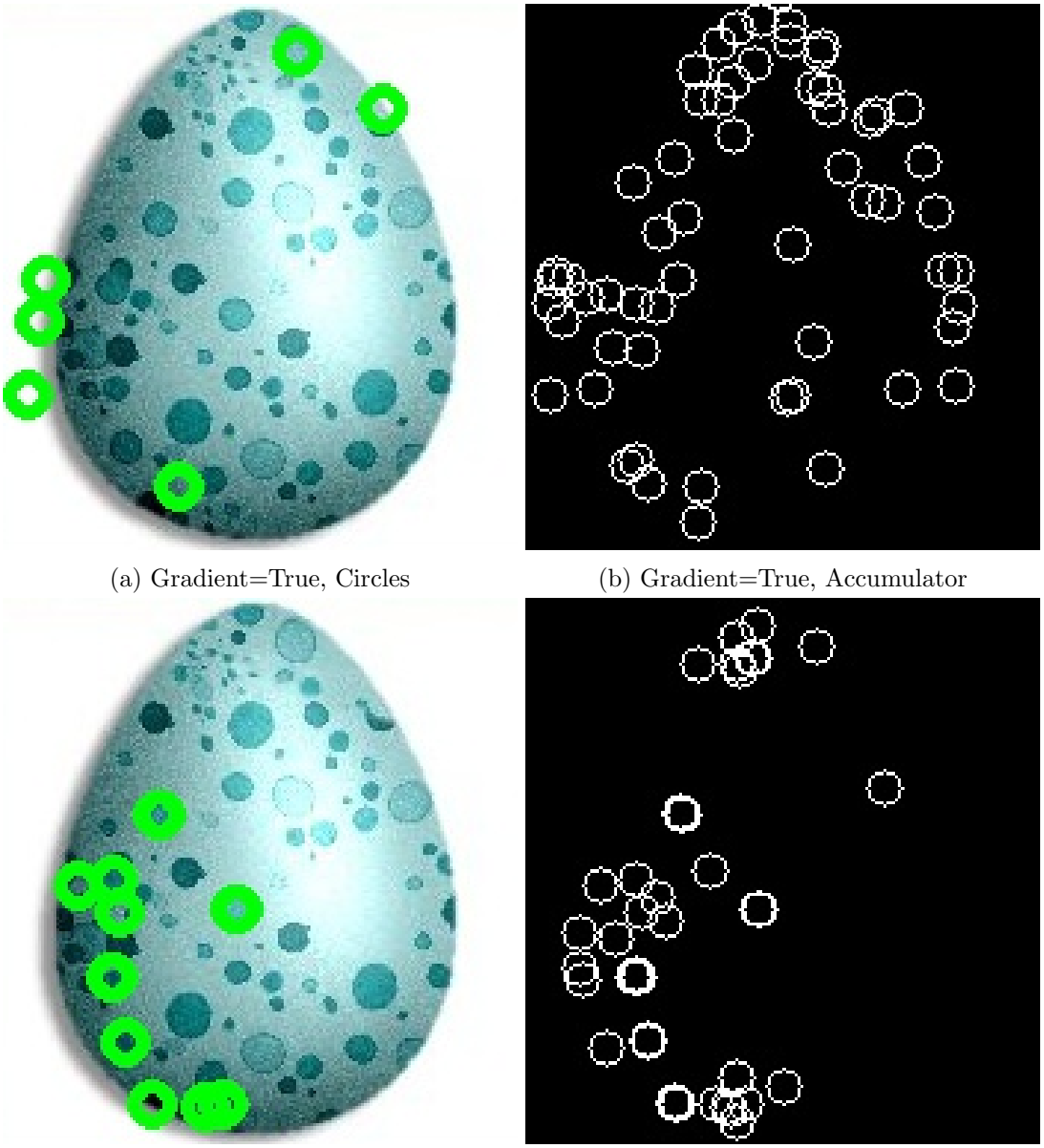
(d) Gradient=False, Accumulator

Figure 4: Results of Hough Transform on Jupiter, r=50

The second image we ran the Hough Transform function on was of the egg shown in Figure 5 below. In this image there are multiple smaller circles of varying radii. Once again, multiple runs were made with different hyperparameters until some were found that made for interesting analysis. In this case, we found all circles with a radius, $r = 6$. The min_val and max_val arguments to the Canny Edge detector were set to 100 and 200, respectively.

Figure 5: Egg

As before, we ran the Hough Transform function both using and not using the image gradients to find the centers. When using the gradient, we kept all centers that were within 20 votes of the top vote-getter for display and all centers within 25 votes of the top vote-getter to show in the accumulator array. When not using the gradient, we kept all centers that were within 3 votes of the top vote-getter for display and all centers within 5 votes of the top vote-getter to show in the accumulator array. In this example, the difference is even more pronounced. The top vote-getter when using gradients received 10 votes, while the top vote-getter when not using gradients received 71 votes. The circles placed on Figure 6(a) do not even really line up with the circles on the egg, and the accumulator array in Figure 6(b) is very spread out with very little overlap. However, in Figure 6(c) we can see that most of the circles line up well with circles on the egg, and the accumulator array in Figure 6(d) clearly shows areas that received multiple votes.



(a) Gradient=True, Circles

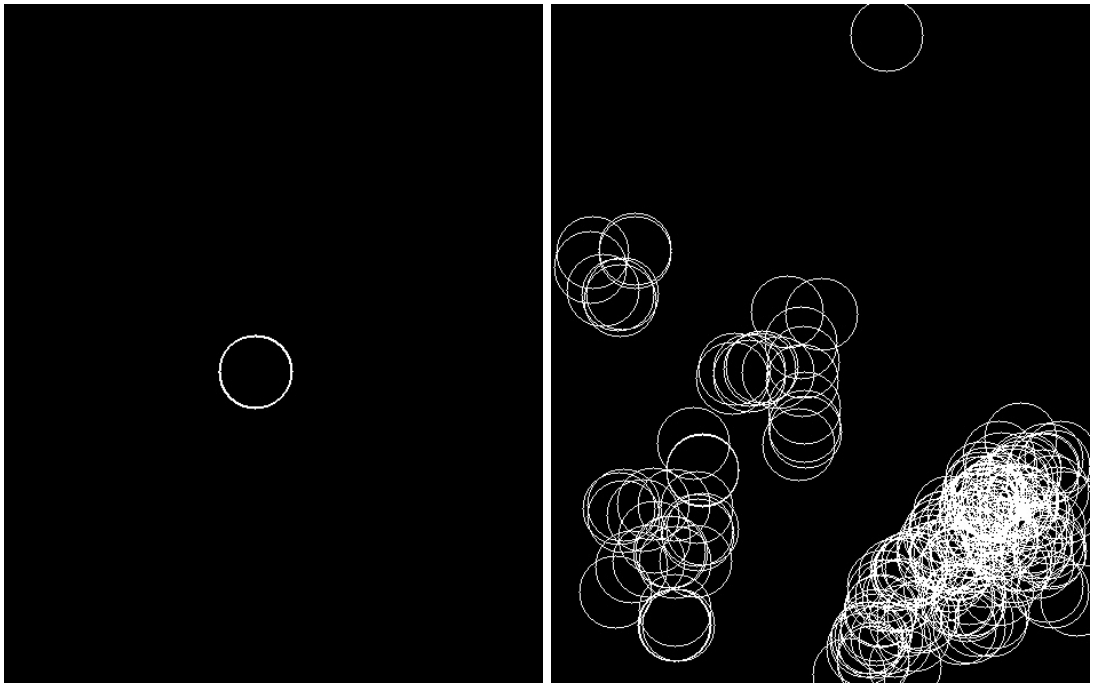
(b) Gradient=True, Accumulator

(c) Gradient=False, Circles

(d) Gradient=False, Accumulator

Figure 6: Results of Hough Transform on Egg, r=6

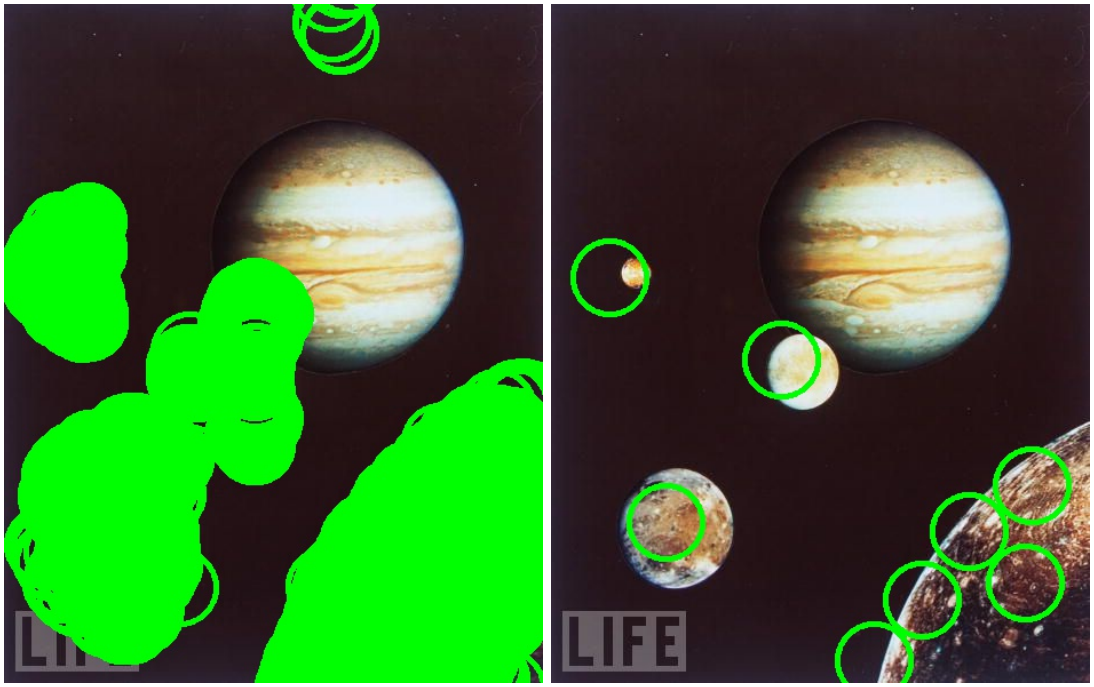
(c) Figure 7 below shows the results of running the Hough Transform function on the Jupiter image with radius $r = 32$. Figure 7(a) does not use the image gradients and keeps all centers within 9 votes of the top vote-getter, while Figure 7(b) uses the image gradients and keeps all centers within 3 votes of the top vote-getter. When not using gradients, the top vote-getter received 47 votes, and when using gradients the top vote-getter received 7 votes. In Figure 7(a), all centers that received between 38-47 votes are displayed, and they all have about the same center. In Figure 7(b), all centers that received between 4-7 votes are displayed, and they are all over the place. This just seemed like a good example of unhelpful the "true" gradients of an image can be when used in downstream tasks.



(a) Gradient=False, Accumulator (b) Gradient=True, Accumulator

Figure 7: Results of Hough Transform on Jupiter, r=32

(d) In part 2(b) above, a very simple method was used to determine how many circles were present. An argument is passed to the function which tells it to keep all centers that are within n votes of the top vote-getter. By viewing the images, it is rather easy to tweak the number and determine how many circles should be present. For this part, we experiment with a bit more complexity that allows us to consider many more possible centers initially. As before, we take all centers that are within n votes of the top vote-getter. However, now we perform K-means clustering on these centers to see if we group them together. This still involves some visual inspection, because we are employing the Elbow Method to determine the correct number clusters, but it allows us to make use of more of the centers without things getting too messy. What is most interesting about this method is that it helps to clean up some of the issues experienced above when using the gradients to find the centers. The example we will use is the Jupiter image with radius $r = 32$, $min_val = 300$, $max_val = 500$, and $n = 25$. When using the gradient, this results in using all centers returned from the accumulator, totalling 4556 (since the highest vote-getter has 6 votes). When not using the gradient, we 941 centers to consider. In Figure 8(a), we see all 4556 centers plotted, and it is not a very useful plot. In Figure 8(b), we see the result of clustering those centers into 8 clusters. The results are still not as good as when not using the gradient, but these are at least getting close to finding the desired circles. In Figure 8(c), we sell the top 941 centers found by not using the gradient. The desired object is circled, but there is a great deal of noise in the bottom right. After again clustering the centers into 8 clusters, we have a much cleaner image, but we were able to make use of much more information from the Hough Transform process.



(a) Gradient=True, No Clustering

(b) Gradient=True, Clustering

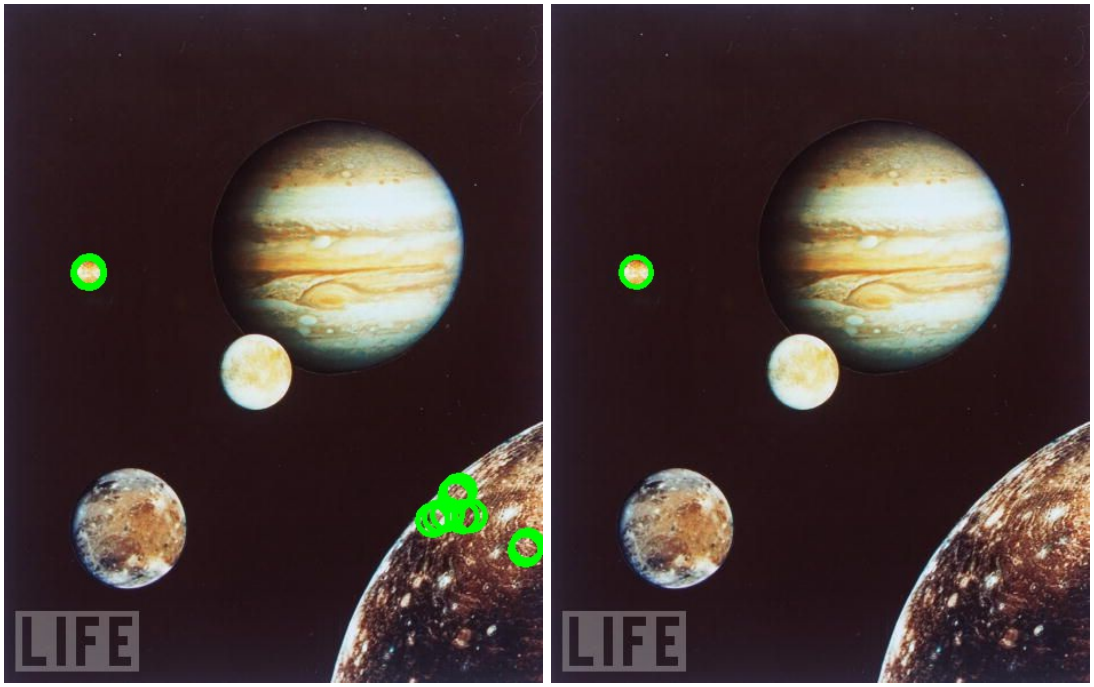
(c) Gradient=False, No Clustering

(d) Gradient=False, Clustering

Figure 8: Comparison of Accumulator Postprocessing, Jupiter, r=32

(e) In the final section, we investigated the effect of vote space quantization, or bin size of the values of a and b that represent the center of the circle. Here we used the Jupiter image, radius $r = 13$, $min_val = 300$, $max_val = 500$, and we kept all centers within 20 votes of the top vote-getter. Figure 9 does an excellent job of demonstrating just how important this hyperparameter is. In Figure 9(a), the bin size is set to 1. This means that all values of a and b are integers. We are

trying to capture the object in the center left of the image, which we do. However, as has often been the case, we also pick up some noise from the object in the bottom right. When we increase the bin size to 2, we only get the desired object. Keep in mind, we are keeping all centers within 20 votes of the top vote-getter. This means all the top votes are going toward the desired center, which is quite impressive. When we further increase the bin size to 3, we completely lose the the desired object and only pick up the noise. Clearly, vote space quantization is critical to selecting the desired objects.



(a) Bin Size = 1 (b) Bin Size = 2

(c) Bin Size = 3

Figure 9: Comparison of Bin Sizes, Jupiter, r=13