

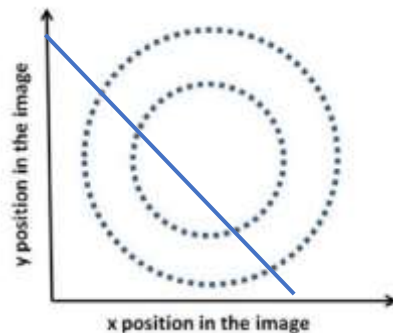
Jorge Betancourt

10/25/18

PS3

Short Answers

1. The following filter bank is sensitive to orientation. The filters' activation is dependent on whether or not an edge is aligned to the filter's orientation, not if the image simply has an edge. An example of a filter bank creating an orientation invariant result could be a circular filter, where the filter will activate regardless of orientation.
2. The clustering would look like something like this depending where the cluster means are initialized:



This is because the clustering doesn't depend on the associations our brain assumes of the image (like that these points belong to 2 separate circles). The clustering is done numerically based on the features fed into the model, so it will try to group points which are close together, whether or not they belong to the same perceived circle.

3. You would want to group the votes using mean shift. This way, all the votes that are close together in the continuous space would converge to a local maximum using gradient ascent. With k-means, you'd have to assign a k without knowing how many groupings there should be. K-means clusters are also heavily affected by outliers when

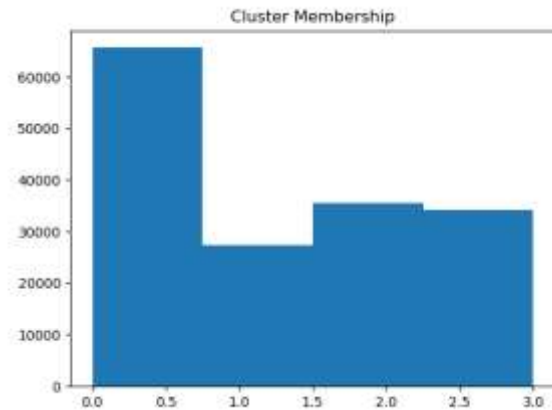
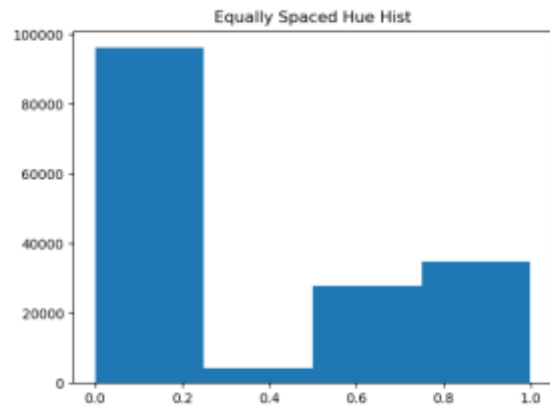
trying to converge on a cluster center. Graph splitting doesn't work as well because you'd need an infinitely large graph to represent all the continuous votes.

4. Psuedocode

- Take the blobs and find bounding boxes around the individual blobs.
- Make a new array that contains the pixel data for each bounding box.
- Take all permutations and pair them up. Scale the smaller box so it's boundaries match the larger blob's matrix (With something like OpenCV's `resize()`).
- For each pair, calculate the SSD error between the two blobs. Then rotate blob A and calculate the SSD again. Continue this for all discrete orientations. Keep track of the minimum SSD (the best orientation fit).
- If the pair reaches a low enough SSD at any of these points, you can consider the blobs to similar enough.
- This implementation is orientation invariant and scale invariant, because it tests against all orientations and naively finds the optimal orientation, and because it scales the shapes based on their bounding boxes.

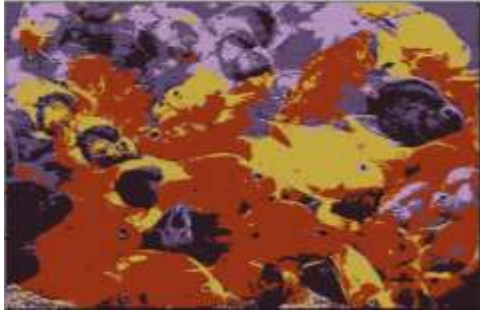





Programming Part 1

d)



Space Left Intentionally Blank

e)

K	Quantized RGB	Quantized HSV
5	<div><p>Quantized RGB</p><p>SSD RGB: 33,860,823,927</p></div>	<div><p>Quantized HSV</p><p>SSD HSV: 5,596,833,656</p></div>
3	<div><p>Quantized RGB</p><p>SSD: 32,945,815,353</p></div>	<div><p>Quantized HSV</p><p>SSD: 5,948,175,270</p></div>
20	<div><p>Quantized RGB</p><p>SSD: 31,532,489,521</p></div>	<div><p>Quantized HSV</p><p>SSD: 4,773,961,942</p></div>

f)

The first histogram shows the distribution of hue values in the image while the second distribution shows the count for each label after having the hues clustered into k groups. The equally spaced bins in the first histogram show a form of naïve clustering where the means are evenly spaced through the possible values from 0 to 1. In this clustering, we see that the second cluster/bin doesn't have a lot of values in it. This is not a good representation of a 4-mean cluster. The second histogram shows the results after using k -means on the hue values. The distribution of hues across the bin has become more uniform as the means are intelligently placed depending on the hue values given to the model.

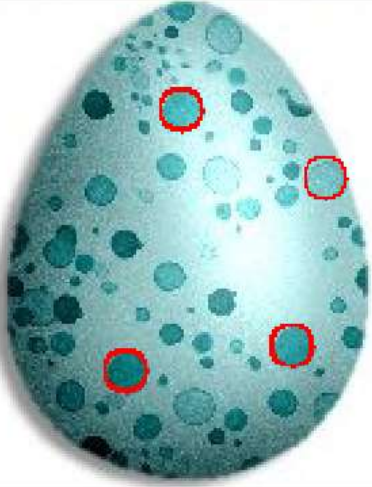
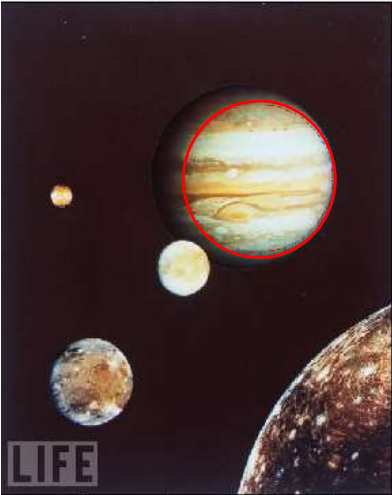
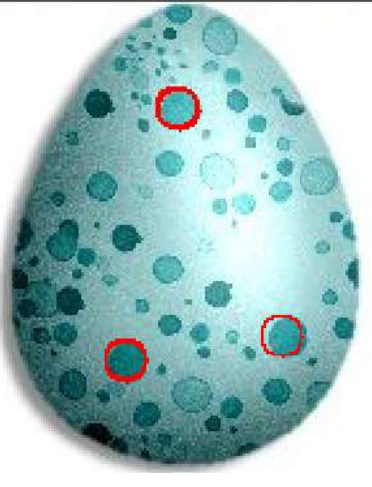

When quantizing across the RGB values and HSV values, we can see different results depending on what color space and what k we choose. We see starker contrasts in the RGB quantization because more of the colors properties are preserved when quantizing in the HSV space. This is because we only affect the hue and not the saturation nor the value. In the smaller k quantization for both color spaces, we can see colors that aren't necessarily that prominent in the original image. That is because the mean values being selected are trying to encapsulate many points in their cluster as possible. When we increase k just a little (from 3 to 5) we can see that the means chosen are more reflective of the original image's color palette.

Another thing to note is that the larger the number of clusters, the smaller the SSD error. This is reflected in the fact that the colors in the higher k clusters are closer to the true color values. The error is also smaller in the HSV clustered images, probably because the only value we are changing in the hue.

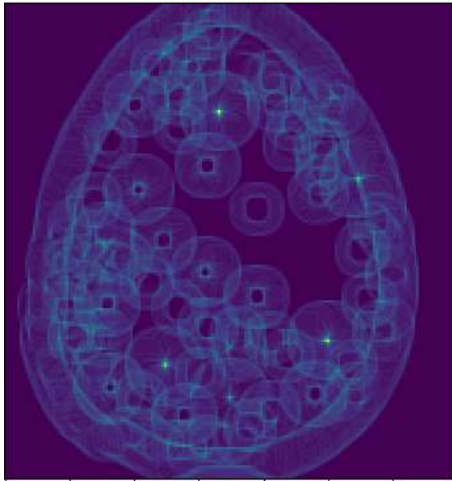
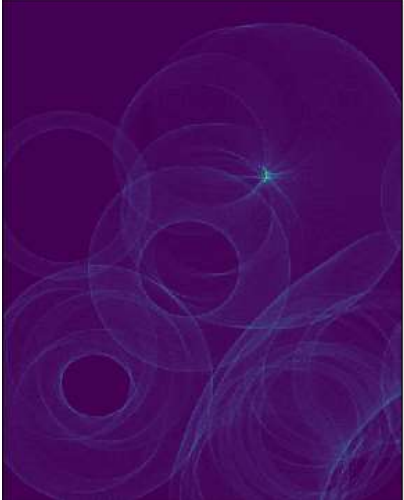
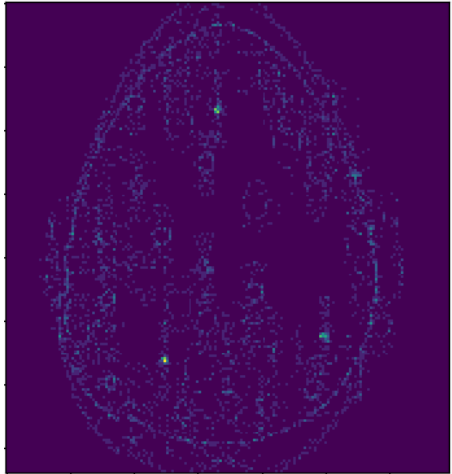

Programming Part 2

- a) The function first grayscales the image. Then it finds Canny edges using a Canny edge detector. It also calculates gradients for the case where the user passes in `useGradient = true`. It generates the vote space by making an all zero array of the original image's dimensions. Then for every pixel representing part of an edge in the edge graph, it increases the votes for every pixel that is a certain distance r away from it. If the user decided to use gradient information, it would only make a 2 votes in the direction of that edge's gradient and the opposite direction (in case the gradient is pointing the wrong way). Then analyze the vote space and find locations where the vote space meets a certain threshold. If none meet the threshold, take the highest voted space.

b) Outputs

Gradient	Egg	Jupiter
False	<p>Output: $r=8$</p> 	<p>Output: $r=89$</p>  <p>(circle added for clarity)</p>
True	<p>Output: $r=8$</p> 	<p>Output: $r=95$</p>  <p>(circle added for clarity)</p>

Accumulators

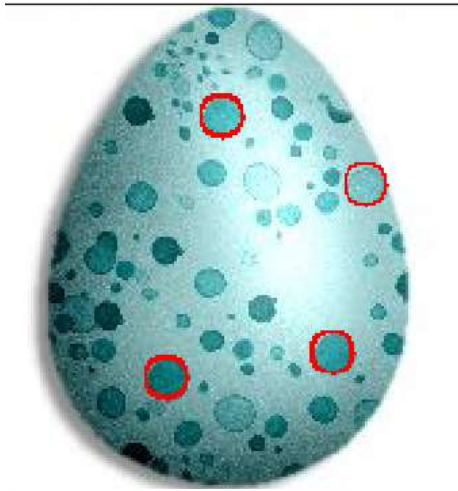
Gradient	Egg	Jupiter
False	<p>Accumulator Array</p> 	<p>Accumulator Array</p> 
True	<p>Accumulator Array</p> 	<p>Accumulator Array</p> 

- c) Looking at the output generated by the egg image without using the gradient, we can see a lot more points/votes than in the one that uses the gradient. We see pixels of high intensity where there are a lot of votes cast. This is because with the given radius, there are a lot of circles drawn on the edges of these circles that intersect with the center of the original circle. The ranges for these high intensity points are in the high 200's because for

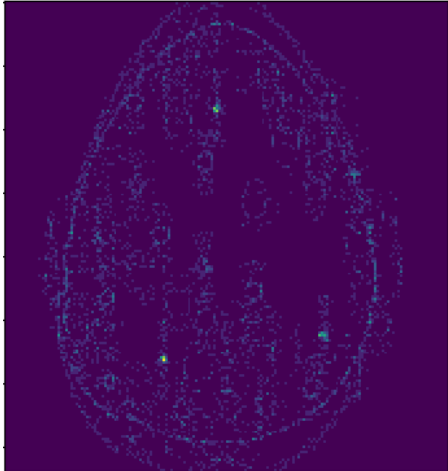
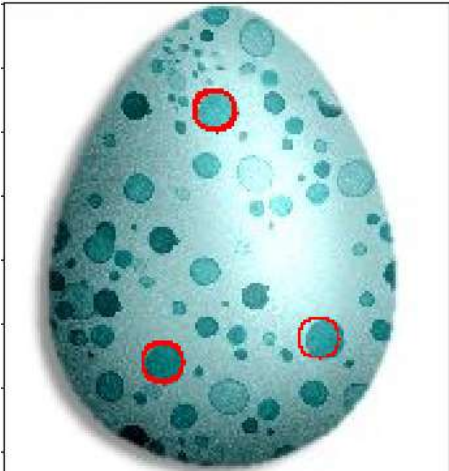
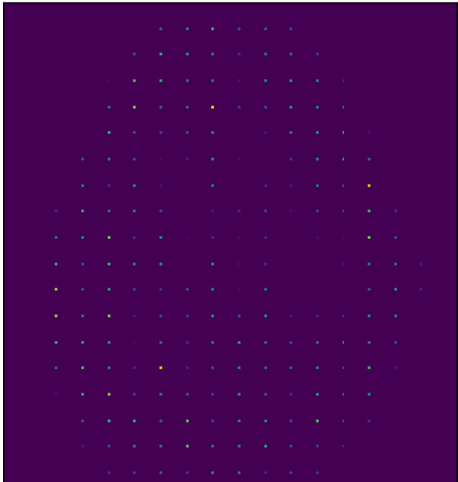
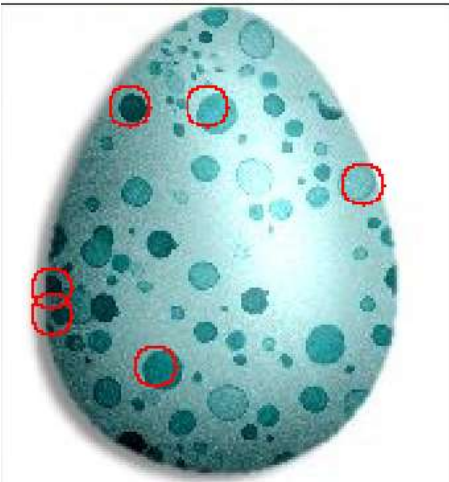
every edge pixel, I cast 360 votes surrounding the edge pixel at a certain radius. This means that if extremely accurate, the center of a circle can reach a vote space of 360 given only the edges of the original circle. Due to rounding errors, having a vote space per cell of >250 is a good indicator of if there is a circle in that location.

- d) You can use mean-shift to post process the vote space and find local maximums for votes that are extremely close together but aren't counted as the same due to rounding error or circle imperfections. For every vote in a location, add that location as a point in a feature list (so these locations are weighted more). Then run mean shift on the data and see how many labels are returned. There will be one label per circle. This is a similar approach to the answer in question 3 in the short answers.

Output: $r=8$



e) Bin Manipulation (Egg Gradient)

Bins	Accumulator	Out
Complete (per pixel)	Accumulator Array 	Output: $r=8$ 
Reduced	Bin Reduction 	Output: $r=8$ 

We can see that more circles are detected because votes are concentrated into larger bins, thereby grouping votes with a wider net. While it finds more circles, the locations are slightly off the true center due to the space in between the bins.