Generalized Hough Transform

Emma Hughson

301356242

Notes:

- All code is broken up into its corresponding animal and orientation.
- The numerical representation of the accumulator arrays is in the appendix
    - I hope it is okay that I took chunks of the array where the peak values were located instead of printing out the entire accumulator array.
- To run each code, you just need to compile and run the MATLAB code. There are no function parameters.
- My Apologies for not putting circles on the accumulator arrays, I was trying through MATLAB but since I don’t have enough memory on my laptop it kept glitching and not putting the circles on.

Generalized Hough Transform

Generalized Hough Transform (GHT) is an extension of Hough Transform. This extension adds the ability of template matching, which allows the algorithm to identify more abstract shapes instead of simple objects such as circles and lines [1]. Template matching involves finding small parts of an image and matching it to a template image. When using GHT, the problem is not a matter of finding an object but to map a template image and locate its position using transformation parameters [2]. However, redundancy is still a major problem in GHT because the abundance of points in the image voting independently [1]. This can be avoided by using a lower resolution, such as using 180 instead of 360 when choosing the size for the r table. The steps for GHT are edge detection, R-table generation, and object detection.

The literature emphasizes the use of either Canny Edge Operator or Sobel for finding edge points in the template and original image [1][3]. Initially, canny edge detection was applied on the animal template and letter template images. After extracting an edge map, the gradient of the image is calculated using a standard filter. A standard gradient filter is a popular way of finding directional information and is also frequently used throughout the literature. Using two filters, that is a 2x1 matrix for dy and a 1x2 matrix for dx, a 1-dimensional convolution operation is implemented. After implementing the convolution operation, we get the gradient direction by using arctan(dy/dx) [3]. Using this gradient reduces the number of potential false positives [1]. Once the gradient direction has been obtained, the r-table can be created. For each index value of the r-table, the reference point information and alpha are stored in a function with phi being the index value.

After creating the r-table, an edge map of the original image is created to implement the accumulator array. This component of GHT aims to find the position of the center of the template image in the original image. The center can be found by finding the peak value(s) amongst the accumulator array. For each edge point in the original image, the gradient direction is computed again. Using the r-table, the counter in our accumulator array is increased for each candidate center point [1]. This part is considered the voting component of Generalized Hough Transform. After voting is complete, the maximum value(s), or peak, is extracted, and the array indices associated with this maximum value are used to identify where the object is in the image. Note that when rotation or scaling is being implemented, the candidate center points are calculated using:

$$X = x - (x'\cos(\text{theta}) - y'\sin(\text{theta})) *s$$
$$Y = y - (x'\sin(\text{theta}) - y'\cos(\text{theta})) *s,$$

Where theta is the angle of rotation of the template image compared to its orientation in the original image [2].

1.1. Edge Map

As already mentioned, Sobel and Canny are used throughout the literature as useful edge map operators when using in conjunction with GHT. More often than not however, Canny is the popular choice amongst researchers. Figure 1 shows when Sobel is used on the template image versus when Canny is used. There are several breaks along the path around the edge of the elephant, which could result in missed gradient information. While in figure 2, Canny shines and shows a completed edge that shows all of the available gradient information. This example illustrates the power of using Canny to garner an edge map.

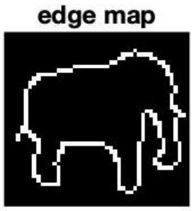


Figure 1. Canny Edge Map for Elephant template image.

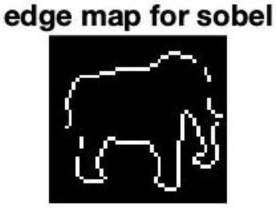


Figure 2. Sobel Edge Map for Elephant template image.

However, when looking at the original image in figure 3 and figure 4, Canny does not show all the possible edges within the image while Sobel outperforms and clearly defines all the images clearly. This indicates that Sobel, when used with a filter beforehand, can do just as well as Canny given a proper threshold value. Furthermore, in figure 5 canny is presented when using the operator on the Letter K template image and in figure 6, Sobel is presented when using it on the Letter K template image. As is apparent, using the canny edge operator considered the shadow under the letter to be part of the object. The shadow does not have the same effect when using Sobel, therefore resulting in an accurate match while Canny resulted in inaccuracy. This indicates that Canny is very sensitive to parameter tuning as well as illustrating gaussian smoothing's ability to influence edges to be off by a bit.

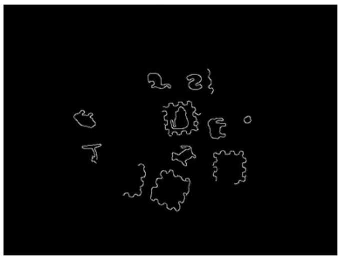


Figure 3. Canny Edge Map for Elephant and Bear Original image.

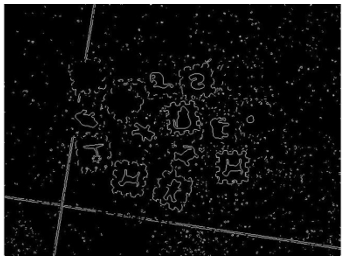


Figure 4. Sobel Edge Map for Elephant and Bear Original image.

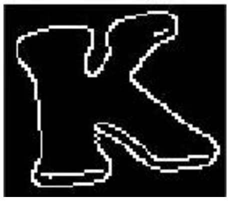


Figure 5. Canny Edge Map for Letter K template image.

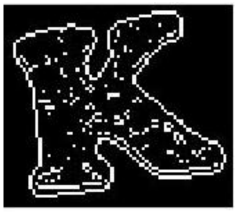


Figure 6. Sobel Edge Map for Letter K template image.

Using the results of the few examples illustrated, using Canny or Sobel to get edge maps are both good operators. However, given the complexity of the image and the amount of noise, one operator will outperform the other. This requires testing several parameter values, or hyperparameter tuning, for each operator until the perfect tuning is discovered. This can take time however, making the process of finding the perfect edge detection method to be time consuming. This also foreshadows GHT's sensitivity to edge detection methods and how slight mistakes in edge identification can result in inaccurate matches.

2.1. The Accumulator Array

After creating the r-table using the values found from the gradient information of the edge maps, the voting process can take place. The voting process involves every pixel voting for its corresponding reference point. The dimensions of the accumulator array are the dimensions of the image so that the accumulator image illustrates the correlation between clusters and the matching. Figure 7 demonstrates how the accumulator array works. In this simple implementation, the peak values cluster around the center of the matched object, with brighter pixels indicating more votes. Around the areas where the voting is little to none the pixel values will be darker, indicating that the template image is not in this location.

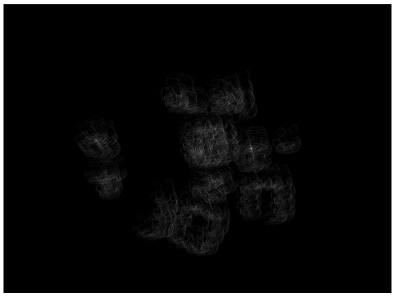


Figure 7. Accumulator array for Elephant image.

A very crucial step to implementing GHT is the edge operator that is chosen. Therefore, hyperparameter tuning is also a very important component. For instance, playing around with the threshold for Sobel edge detector creates varying edge maps and altering gradient information, and thus changing the results of the accumulator array. Figure 8 shows when the threshold for Sobel is 0.09 and Figure 9 show when the threshold is 0.02. The more defined the cluster is the more likely that GHT will be able to find the template image within the original image. In addition to Sobel, the result using Canny Edge detector with a threshold of 0.45 resulted in the best identifiable peak value in the accumulator array for finding the Elephant (figure 10). The use of Canny and a threshold of 0.45 has been used in the literature as useful for detecting coins and other objects in noisy, deformed images [6].

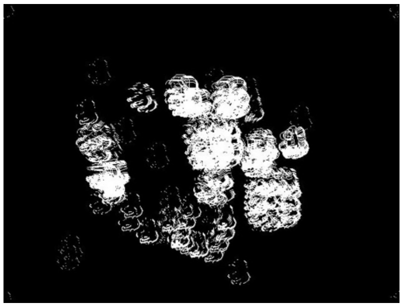


Figure 8. Accumulator array when Sobel threshold is 0.09 for Elephant image.

Figure 9. Accumulator array when Sobel threshold is 0.02 for Elephant image.

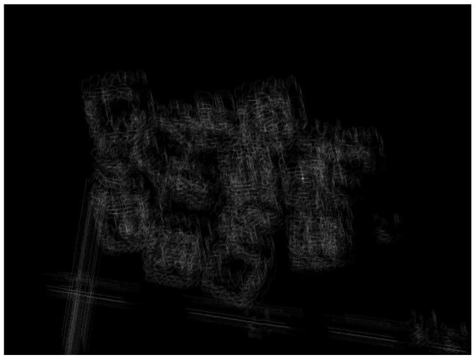


Figure 10. Accumulator array when Canny threshold is 0.45 for Elephant image.

For the template image for the letter K, using various Sobel values also affect the accuracy of the GHT model. Figure 11 shows when Sobel is between 0.05 and 0.09 and figure 12 shows when Sobel is 0.02. The higher threshold once again helped GHT accurately locate the template images for K. Taking a look at the image there are faint bright dots indicating the locations of the template image within the original image. These bright pixels are the location where the peak value is for the location of the K's that are not rotated or scaled. For the rotated images, the peak values are located in a different place, as indicated in figure 13.

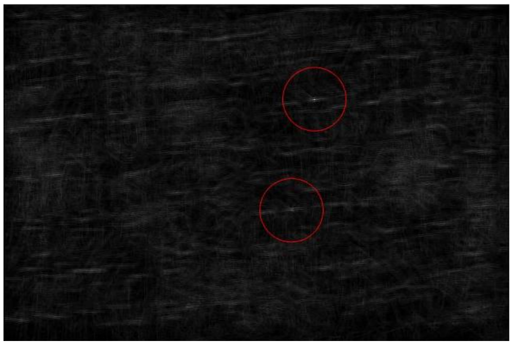


Figure 11. Accumulator array when Sobel threshold is 0.05-0.09 for Letter K image for K's that do not involve rotation or scaling
.

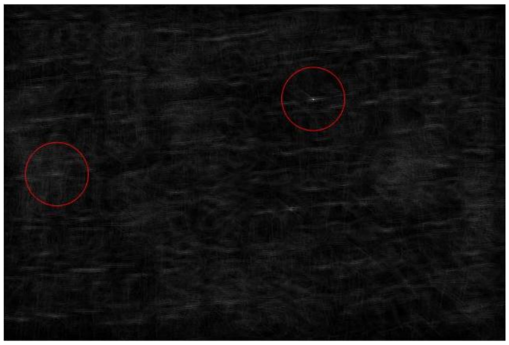


Figure 12. Accumulator array when Sobel threshold is 0.02 for the Letter K image for K's that do not involve rotation or scaling
.

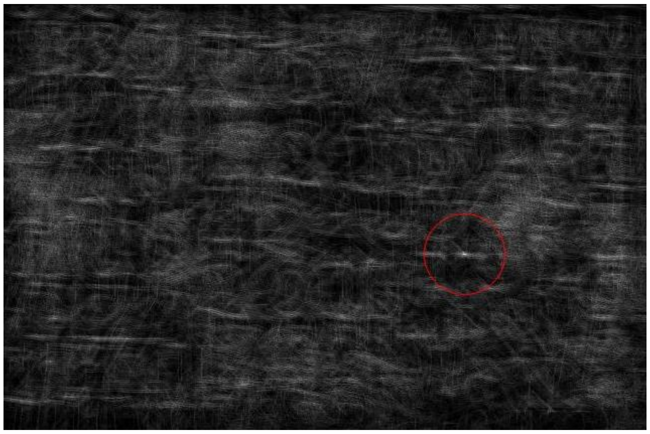
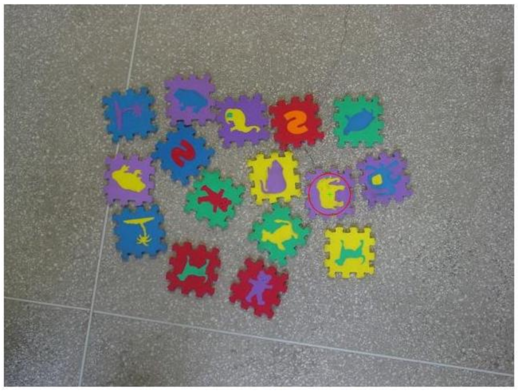


Figure 13. Accumulator array when Sobel threshold is 0.05-0.09 for Letter K image for K's that do involve rotation or scaling
.

Locating the peak value is strongly dependent on the edge operator used. If the gradients of the edges cannot be calculated accurately then the peak values cannot be found in the right location. This is because if voting takes place for gradient values that are in the incorrect location due to fuzzy, poor edges in the edge map then the peak is calculated elsewhere. Furthermore, edge maps play an important role in determining concavity, radii, and rotation angle for calculating the peak value. Varying values of the Sobel operator may result in inaccurate location detection but sometimes this can be protected against due to rotation and radii values removing votes that are not indicative of the matching object [4]. This also eludes to the susceptibility of GHT to noise in images. Although the original images did not contain much noise, the similarity between all the objects in the image and the template object(s) can be considered complex and can be used as an example of what would happen if a lot of noise existed in the original image. Therefore, incorrect parameters for edge maps, noise or high similarity amongst objects can result in imprecise object positioning resulting in a wrong peak value [5].

3.1. Finding the Template Image

For finding the elephant and the bear, the algorithm was self-explanatory. Each image contained some form of rotation, so the candidate center points were calculated accordingly. The need for using the formula for template images that do not involve rotation or scaling was not needed and therefore, the rotation formula was used. Finding the bear required finding the indices for the two peaks in the accumulator array and overlaying them both on the original image. For both the elephant and the bear, the best results occurred when Canny Edge Operator was used on the template image and Sobel was used on the original image to get the edge values (Figure 14 and Figure 15). However, using Canny or Sobel on the original image for the first bear (i.e., purple bear on red squares) worked fine but using canny on the second bear (i.e., red bear on green square) did not work. For the second bear, the threshold for Sobel also played an important factor in finding the bear. The threshold had to be around 0.4 in order for GHT to work correctly and correctly find a match. Further indicating how sensitive GHT is to edge operators. If the edge operator is not exact, then GHT will not work. When attempting to find K and Q in the letter image, a similar sensitivity occurred with the edge operator used. Thus, further emphasizing the importance of using the correct edge operator when performing GHT because of GHT's sensitivity to smoothed images.



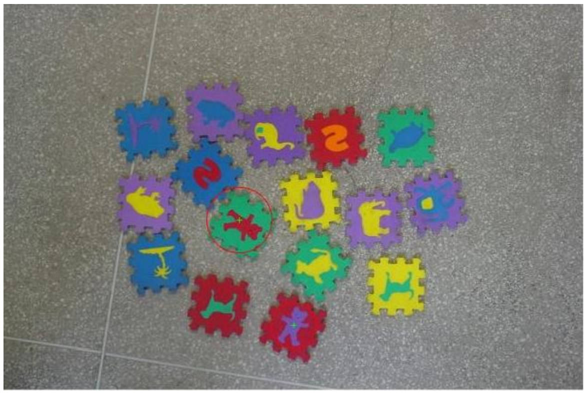


Figure 14. Resulting image for locating the Elephant template in the image.
.

Figure 15. Resulting image for locating the two bears in the image.
.

In addition to edge operator, GHT was sensitive to rotation and scaling factors. If the scale factor or rotation angle was not exact, or near exact, then GHT failed to find the correct matchings. Furthermore, using the correct distance between the reference point and the value of the point around the edge also influenced the correctness of the algorithm. In the letter images, there were some K's or Q's which involved no rotation or scale and so using the distance alone was needed to calculate candidate reference points. However, for finding the fourth K, which is located in the bottom right corner of the letter image, could only be found when finding the difference between the edge point in the original image and rho. This may be due to the K being in a corner and only half of the letter is showing. As such, this formula that was implemented was able to capture the peak values for the K that was located on the edge.

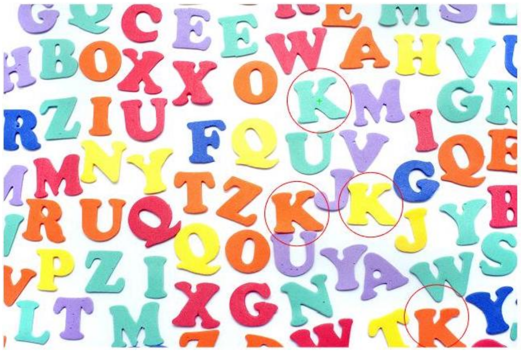


Figure 15. Resulting image for locating the K's in the image.
.

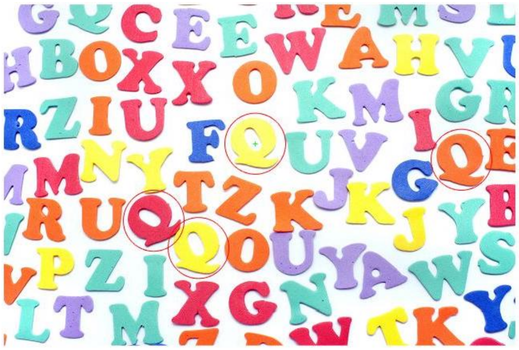


Figure 16. Resulting image for locating the Q's in the image.
.

In conclusion, GHT was found to be sensitive to orientation of the object, smooth images, and distance calculation of the edge point and the center. Given that smoothing images can make edge points difficult to find and can provide less gradient information, the susceptibility of GHT to smoothed images is to be expected. As well, the choice of edge map operator and gradient information were very crucial to setting up the success of the GHT algorithm. If the edge operator chosen was not sufficient enough to garner the correct information GHT failed. Choosing the correct edge operator was also strongly dependent on hyperparameter tuning which was timely to conduct but resulted in correct match detection. Furthermore, reducing the resolution of the r table to 180 angles seemed to mitigate the inefficiency of the algorithm without losing information regarding orientation.

Refrences

[1] M. A. Larhmam, S. Mahmoudi and M. Benjelloun, "Semi-automatic detection of cervical vertebrae in X-ray images using generalized hough transform," *2012 3rd International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Istanbul, 2012, pp. 396-401.

[2]"Generalised Hough transform", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Generalised_Hough_transform. [Accessed: 15- Feb- 2020].

[3] P. Rerkngamsanga, M. Tummala, J. Scrofani and J. McEachen, "Generalized hough transform for object classification in the maritime domain," *2016 11th System of Systems Engineering Conference (SoSE)*, Kongsberg, 2016, pp. 1-6.

[4]D. Tsai, "An improved generalized Hough transform for the recognition of overlapping objects", *Image and Vision Computing*, vol. 15, no. 12, pp. 877-888, 1997. Available: 10.1016/s0262-8856(97)00033-4.

[5] Y. Ji, L. Mao, Q. Huang and Y. Gao, "Research on Object Shape Detection from Image with High-Level Noise Based on Fuzzy Generalized Hough Transform," *2011 International Conference on Multimedia and Signal Processing*, Guilin, Guangxi, 2011, pp. 209-212.

[6]] M. Roushdy, "Detecting Coins with Different Radii based on Hough Transform in Noisy and Deformed Image", *GVIP Journal*, vol. 7, no. 1, 2007. [Accessed 18 February 2020].

Appendix

Accumulator Array Numerical Information:

For Elephant #1:
Note: I used `xdatatemp = accArray(:, [220:250 400:420]);` To take a chunck of the array where I know the peak value is located and I copied the Columns 24 through 46 chunk and removed the large chunks of zero to focus on the area where I know the elephant is located.

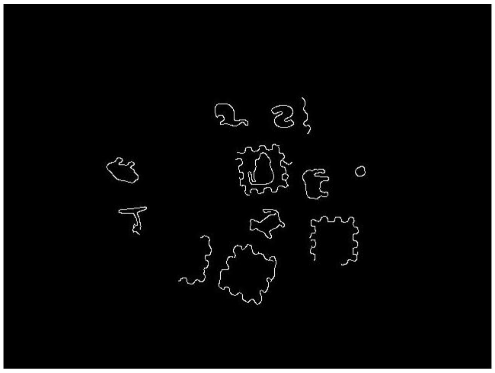
```
   0    0    0    0    0    0    0    0    3    4    3    4    3    6    9    5    2    2    2    1    1    2
2
   0    0    0    0    0    0    0    0    4    4    4    5    5    7   12   10    5    3    2    1    1    1
2
   0    0    0    0    0    0    0    0    5    4    3    1    0    0    6    7    4    2    3    2    1    1
1
   0    0    0    0    0    0    0    0    4    1    0    0    0    0    4    7    6    3    3    3    1    0
1
   0    0    0    0    0    0    0    0    5    1    1    2    1    1    4    6    6    1    1    2    2    2
2
   0    0    0    0    0    0    0    0    4    1    3    5    5    5    5    5   10    2    2    2    2    1
1
   0    0    0    0    0    0    0    0   13   10    9    7    5    5    6   10   19   10    7    6    5
3    2
   0    0    0    0    0    0    0    0    6    3    4    4    4    3    3    2    8    5    3    5    5    4
4
   0    0    0    0    0    0    0    0    3    0    2    2    2    2    1    2    8    5    3    6    5    4
2
   0    0    0    0    0    0    0    0    4    1    0    1    1    2    0    0    4    9    5    6    4    2
3
   0    0    0    0    0    0    0    0    8    7    7    6    7    6    6    8   12   14   10    7    3
4    4
   0    0    0    0    0    0    0    0    9    7    7    8   10   10   11   11   10   16   15   13   12
9    7
   0    0    0    0    0    0    0    0    5    4    6    6    5    3    3    3    5   11    7    1    1    4
5
   0    0    0    0    0    0    0    0    7    5    6    4    3    3    3    2    3   10    9    1    5    2
2
   0    0    0    0    0    0    0    0    4    3    3    3    2    2    2    1    3   11   10    7    4    4
5
   0    0    0    0    0    0    0    0    4    5    7    9   10   10    9    6    4   14   21   11    3
5    5
   0    0    0    0    0    0    0    0    3    5    5    5    4    6    7   10   12   24   38   15   12
11   11
```

```
    0   0   0   0   0   0   0   0   3   3   7   4   2   3   6   5   7  13  18   6   6   6
5
    0   0   0   0   0   0   0   0   4   4   5   1   4   6   8   5   5   7  15   4   3   5
4
    0   0   0   0   0   0   0   0   1   4   3   3   3   5   4   4   4   6  14   7   5   6
4
    0   0   0   0   0   0   0   0   2   4   2   4   6   6   5   4   5   6   9   9   7   6
3
    0   0   0   0   0   0   0   0   3   4   2   6   7   5   5   3   4   6   5  14  12
12  11
    0   0   0   0   0   0   0   0   3   4   3   4   3   3   3   3   5   6   5  11   6   4
3
    0   0   0   0   0   0   0   0   2   3   1   4   3   3   4   3   5   6   6   9   5   4
4
    0   0   0   0   0   0   0   0   2   2   1   2   2   2   3   2   4   4   6   5   4   2
5
    0   0   0   0   0   0   0   0   1   0   0   2   1   3   2   1   3   3   5   5   6   2
4
    0   0   0   0   0   0   0   0   2   1   2   3   1   2   1   1   2   4   4   3   5   3
4
    0   0   0   0   0   0   0   0   0   1   1   1   3   3   2   4   5   5   3   4   4   4
5
    0   0   0   0   0   0   0   0   1   2   1   1   2   2   4   5   3   5   4   4   2   5
5
    0   0   0   0   0   0   0   0   1   2   2   3   2   6   6   5   2   2   2   4   3   4
2
    0   0   0   0   0   0   0   0   1   1   1   3   5  10   7   7   3   3   3   2   2   1
1
    0   0   0   0   0   0   0   0   0   0   0   1   4   6   7   7   6   5   4   5   7   6
3
    0   0   0   0   0   0   0   0   0   0   0   0   4   3   3   2   4   5   4   5   6   4
3
    0   0   0   0   0   0   0   0   0   0   0   0   3   3   3   2   4   2   4   3   3   3
1
    0   0   0   0   0   0   0   0   1   0   0   0   3   3   2   4   3   2   3   4   3   3
1
    0   0   0   0   0   0   0   0   2   1   1   1   3   3   3   5   6   5   5   4   5   4
3
    0   0   0   0   0   0   0   0   2   2   1   1   6   5   6   8   8   5   4   7   9   9
7
    0   0   0   0   0   0   0   0   4   1   1   3   6   4   4   4   6   6   6   8   7   7
5
    0   0   0   0   0   0   0   0   3   2   4   6   4   4   2   3   1   2   1   1   3   1
1
    0   0   0   0   0   0   0   0   3   1   1   3   1   2   1   3   1   2   1   0   1   1
1
```

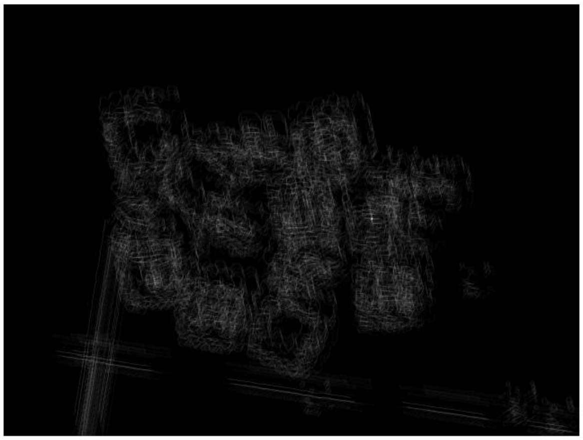
```
   0  0  0  0  0  0  0  0  6  3  2  2  3  1  2  2  1  2  1  1  2  2
2
   0  0  0  0  0  0  0  0  3  1  0  0  4  2  3  5  6  4  2  3  2  2
5
   0  0  0  0  0  0  0  0  1  1  1  1  4  6  5  5  4  4  8  7  8  10
8
   0  0  0  0  0  0  0  0  1  2  1  2  3  4  6  5  5  3  4  3  3  4
3
   0  0  0  0  0  0  0  0  5  3  3  1  2  2  3  4  4  3  3  0  1  3
5
   0  0  0  0  0  0  0  0  5  7  8  8  6  4  3  2  4  3  2  2  0  0
6
   0  0  0  0  0  0  0  0  1  3  3  2  3  3  4  4  4  3  2  1  1  2
8
   0  0  0  0  0  0  0  0  2  4  2  2  3  4  3  2  4  5  2  3  5  4
5
   0  0  0  0  0  0  0  0  2  4  5  4  3  2  2  2  2  6  6  3  3  2
2
   0  0  0  0  0  0  0  0  3  6  5  3  2  0  1  2  3  6  5  2  2  1
0
   0  0  0  0  0  0  0  0  5  6  2  3  1  2  1  3  1  2  5  0  2  0
1
   0  0  0  0  0  0  0  0  7  4  1  1  1  3  0  0  1  1  6  2  3  3
2
   0  0  0  0  0  0  0  0  4  3  1  0  2  2  2  2  2  2  4  2  2  2
4
   0  0  0  0  0  0  0  0  1  4  3  2  1  2  1  1  1  1  4  1  4  2
4
   0  0  0  0  0  0  0  0  0  1  2  4  2  2  2  3  1  1  1  3  5  2
4
   0  0  0  0  0  0  0  0  1  0  0  2  1  1  0  0  0  1  1  4  4  2
2
   0  0  0  0  0  0  0  0  1  1  1  0  0  1  1  1  0  1  2  4  1  0
1
   0  0  0  0  0  0  0  0  1  1  2  1  1  0  0  0  1  1  2  2  0  0
0
   0  0  0  0  0  0  0  0  1  1  2  1  2  1  1  0  1  1  3  1  0  0
0
   0  0  0  0  0  0  0  0  0  0  1  1  2  1  1  1  2  1  2  0  0  0
0
   0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  2  1  1  0  0  0
0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  1
1
   0  0  0  0  0  0  0  0  1  0  0  0  0  1  1  1  1  0  0  1  1  0
0
```

```
     0     0     0     0     0     0     0     0     1     1     1     2     2     0     0     1     2     4     6     4     2     1
0
     0     0     0     0     0     0     0     0     0     0     1     1     1     1     2     2     2     1     2     1     2     2
2
```

Elephant #1 Edge Maps:



Elephant #1 Accumulator Array:

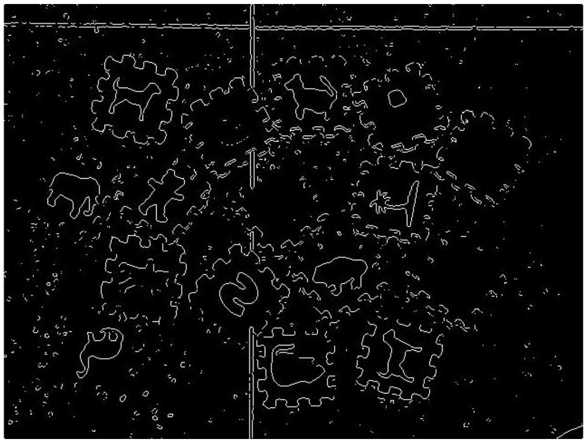


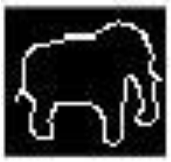
Elephant #1 Final Result:

Elephant #2:

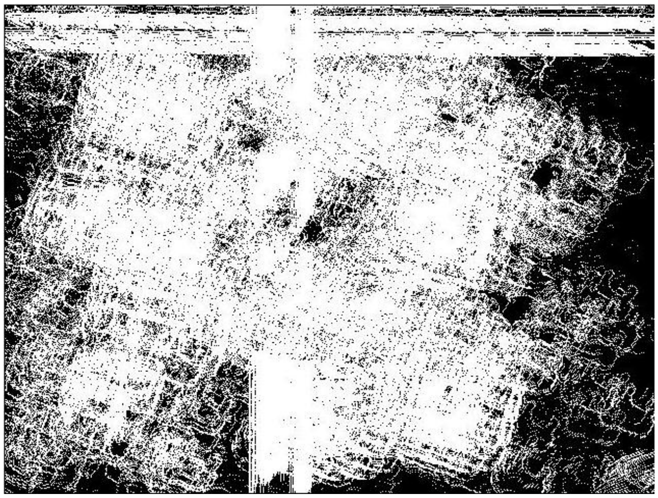
5	1	3	4	2	5	6	1	3	3	4	2	3	2	4	5	6	3	2	2	2	3
5	4	1	2	3	6	6	3	5	7	5	4	2	3	0	7	3	3	2	3	4	5
5	6	3	2	1	4	3	4	3	5	5	4	3	2	2	8	3	4	3	3	6	4
5	2	5	4	3	5	3	4	6	7	4	4	3	0	3	6	4	4	1	2	4	3
3	3	2	2	5	4	2	5	3	5	2	3	6	4	3	5	6	2	0	6	2	2
1	0	6	5	5	8	8	3	5	6	4	3	4	4	2	7	6	3	2	4	3	1
2	3	0	6	7	3	2	2	1	4	5	4	6	2	4	8	5	7	7	5	2	1
2	1	1	7	5	4	3	5	2	3	3	5	5	4	9	7	5	5	3	1	3	4
1	1	5	4	8	1	2	2	1	2	3	6	6	5	8	9	7	12	0	3	4	0
6	3	4	6	3	3	1	2	1	3	5	6	7	4	7	12	9	5	2	4	3	1
2	5	6	3	1	4	0	4	3	4	1	4	6	6	10	10	9	2	1	6	3	4
6	1	5	1	0	2	0	1	3	4	4	7	10	7	11	7	6	6	4	5	4	2
3	3	2	3	1	0	1	1	3	0	3	8	13	16	14	11	8	3	4	10	5	3
4	3	1	4	0	0	0	2	5	0	3	13	15	19	19	13	5	7	3	9	3	9
3	3	6	0	1	1	2	5	3	1	1	16	19	20	15	7	10	5	10	5	6	6
1	7	3	3	1	0	3	1	4	4	1	18	22	23	15	7	4	5	7	8	12	4
3	5	3	0	2	5	2	2	4	1	1	14	18	20	15	8	6	11	8	8	12	4
1	2	3	0	3	3	2	5	1	2	2	7	15	16	2	4	7	8	13	10	8	5
2	3	4	4	1	4	5	0	4	0	1	8	16	10	4	1	5	9	11	9	13	6
3	2	6	2	1	4	3	1	0	2	1	8	12	6	2	1	5	5	2	10	10	1
3	3	2	2	2	2	1	0	2	1	1	9	8	3	2	0	7	10	2	6	8	4
3	5	4	3	1	3	3	1	2	1	2	3	7	3	1	0	9	9	7	6	6	4
2	3	1	4	4	2	0	3	2	2	2	2	3	4	2	1	8	4	6	1	5	1
3	2	2	3	0	0	1	0	2	3	3	2	3	4	1	3	8	4	2	2	5	2
2	2	2	0	3	2	1	1	2	1	0	5	1	4	3	6	4	5	2	3	4	2
6	1	3	2	2	1	1	0	1	0	2	1	3	2	2	5	6	2	1	0	2	4
4	4	3	1	3	1	3	6	5	1	0	1	1	1	4	3	3	2	0	2	5	1
5	1	0	2	3	2	3	1	1	2	3	3	1	3	2	8	2	3	1	2	2	2
5	2	2	0	3	2	1	2	1	1	2	1	2	2	5	4	3	0	0	4	1	0

Elephant #2 Edge Maps:

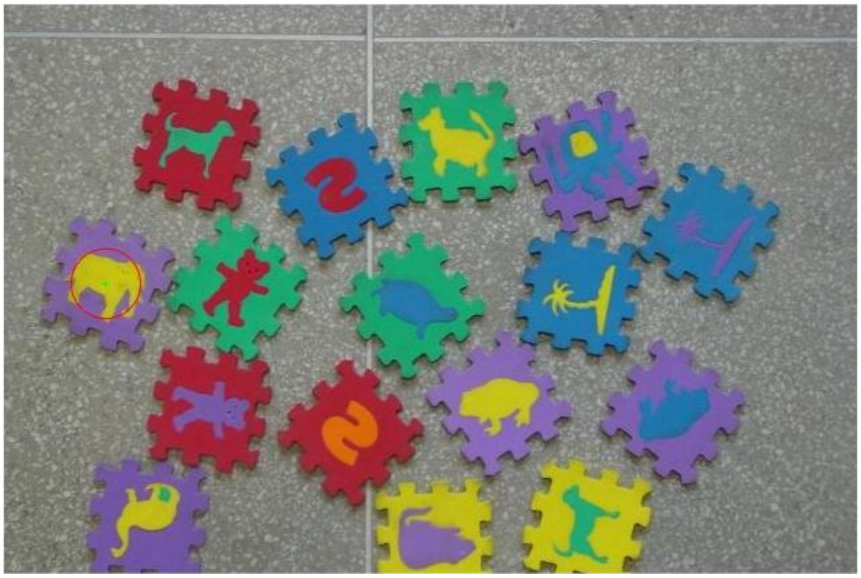




Elephant #2 Accumulator Array:



Elephant #2 Final Result:



Bear #1

Accumulator #1:

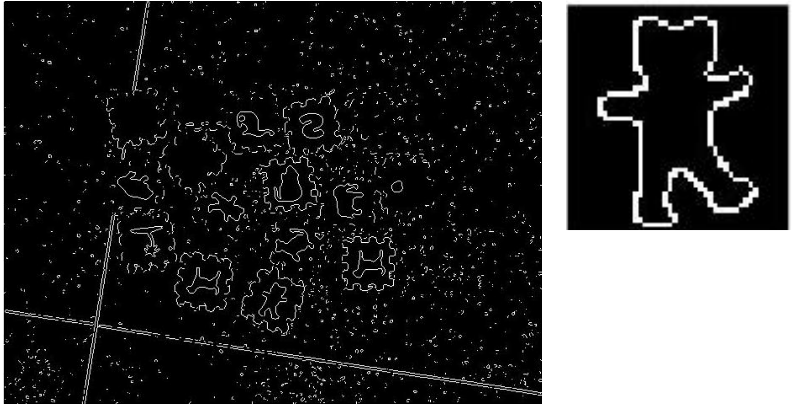
8	10	13	5	9	7	8	8	2	7	8	6	7	7	7	3	10
12	11	14	12	6	7	9	9	7	5	7	9	9	12	8	10	12
11	14	10	10	9	7	8	5	6	5	6	8	6	15	9	18	9
19	12	14	12	10	6	7	4	6	8	5	5	8	9	9	11	8
15	11	11	12	14	11	7	5	5	7	7	5	5	12	15	11	8
16	7	9	9	16	11	5	4	7	7	8	12	13	20	19	12	6
11	8	8	9	12	9	7	8	9	10	8	13	19	39	39	18	9
9	10	11	9	14	6	9	6	9	7	8	6	17	31	25	12	9
17	13	10	9	10	6	2	3	8	6	7	11	12	18	15	9	6
9	13	11	9	13	11	5	6	5	4	9	12	10	7	13	10	5
9	12	9	9	19	13	10	9	7	7	6	11	13	11	9	9	7
11	10	13	8	16	14	16	11	10	7	2	4	7	7	6	3	9
10	11	9	9	11	14	13	11	12	6	5	7	4	3	3	6	4
4	7	6	12	11	14	8	11	15	12	11	7	6	3	6	7	9

Accumulator #2:

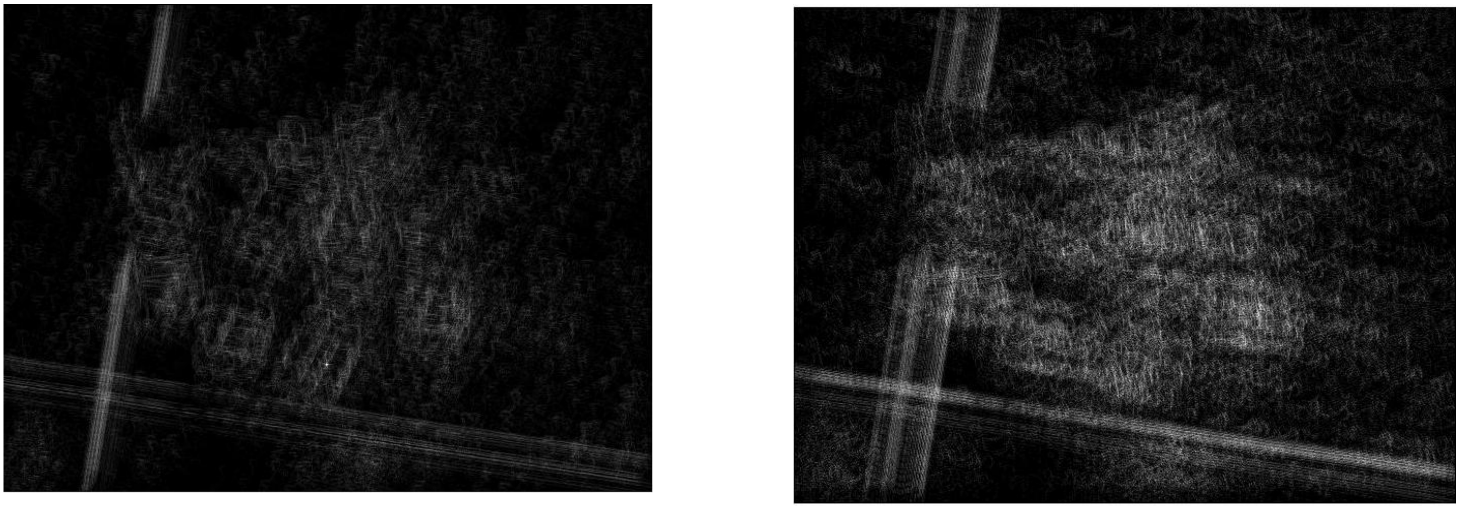
2	5	3	2	3	1	1	2	4	3	3	1	2	1	5	5	2	2	1	2	2	5
2	3	2	1	1	0	0	2	7	5	3	3	3	1	3	6	3	3	1	4	4	10
1	4	4	1	1	2	4	4	8	6	4	5	4	1	3	3	6	7	5	4	4	9
1	3	4	3	1	1	2	5	5	4	1	2	5	2	4	6	9	10	5	4	1	4
2	2	0	4	3	4	2	2	3	4	3	6	3	4	3	7	7	8	4	5	1	4
1	1	2	0	1	2	1	2	3	6	2	5	4	2	6	2	9	13	7	6	2	4
2	4	4	3	3	2	2	5	6	6	4	4	4	4	6	4	6	8	7	4	3	3
5	4	3	4	0	6	3	2	3	7	3	4	5	1	7	5	9	8	4	6	3	7
3	1	4	2	1	3	2	3	3	3	4	7	4	3	3	6	5	16	8	4	4	10
2	4	2	4	4	2	3	6	2	3	2	7	6	3	4	7	4	12	14	5	6	8
1	0	1	4	1	2	1	3	3	3	2	6	7	5	1	5	4	6	8	5	4	5
2	3	0	3	3	0	0	3	1	4	0	4	7	4	7	9	4	4	7	4	1	4
1	1	1	2	4	3	1	2	0	0	2	5	4	6	3	8	5	2	7	8	2	3

2 1 1 4 1 1 1 1 2 1 2 4 5 3 3 5 7 4 4 3 2 4

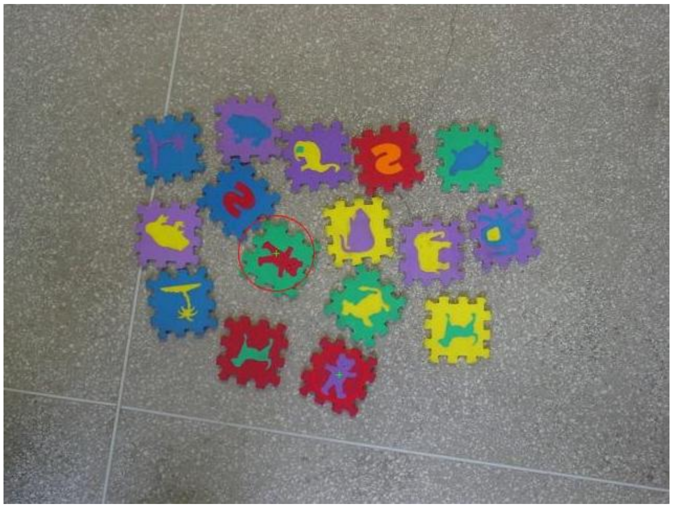
Bear #1 Edge Maps:



Bear #1 Accumulator Arrays:



Bear #1 Final result:



Bear #2:

Accumulator #1:

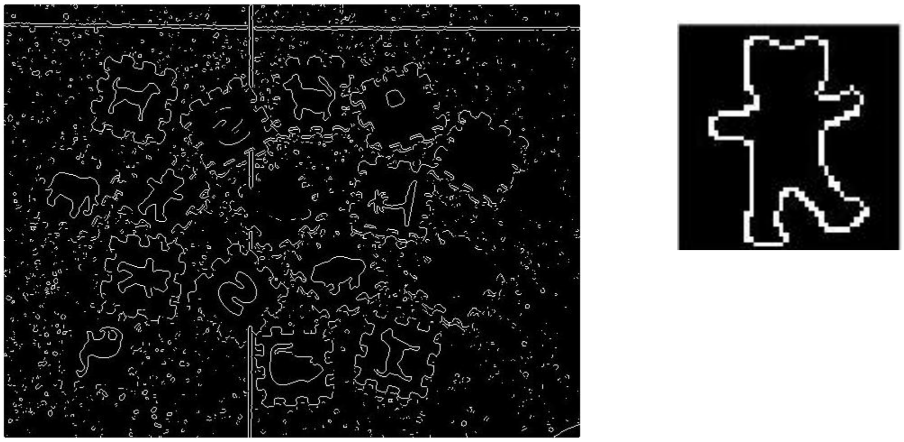
5	5	5	5	4	5	3	2	4	5	4	2	2	3	6	8	7	6	5	3	2	2
5	3	2	2	8	6	6	6	4	3	2	5	3	3	7	5	6	6	5	0	1	2
5	4	6	5	6	6	6	10	7	8	7	4	4	5	8	8	9	7	3	2	1	3
3	2	4	5	1	6	5	4	7	2	6	5	2	6	6	8	11	8	5	3	2	2
4	3	5	6	5	7	10	2	3	2	3	2	1	3	8	5	10	6	5	6	5	5
4	2	6	5	5	6	6	6	4	3	0	1	2	3	4	8	8	5	3	5	4	5
3	4	5	6	9	5	3	3	3	3	3	3	2	6	6	5	10	6	5	4	3	4
4	6	5	5	4	3	5	5	4	2	1	4	1	7	4	6	8	4	2	3	5	1
3	3	7	3	5	2	4	4	2	2	1	12	9	6	8	7	11	4	3	4	3	3
3	4	1	6	5	5	5	2	3	1	3	9	12	12	13	14	15	7	2	3	1	2
5	6	6	2	1	1	4	1	3	1	0	6	6	7	10	15	18	12	9	4	3	1
6	6	6	3	5	2	1	2	1	1	0	6	5	5	4	13	15	9	6	4	3	3
4	6	4	6	4	5	3	6	1	1	1	5	5	4	7	8	10	4	4	3	2	1
7	5	2	2	1	5	2	2	2	3	4	3	6	8	8	13	7	3	4	3	2	2
8	5	4	1	1	2	2	1	2	1	1	5	8	2	5	6	7	3	7	5	3	2
7	4	3	2	1	3	1	1	2	1	2	7	6	6	9	7	5	2	3	3	1	2
6	5	2	3	1	2	1	2	1	3	2	4	3	4	4	6	6	2	2	2	2	2

Accumulator #2:

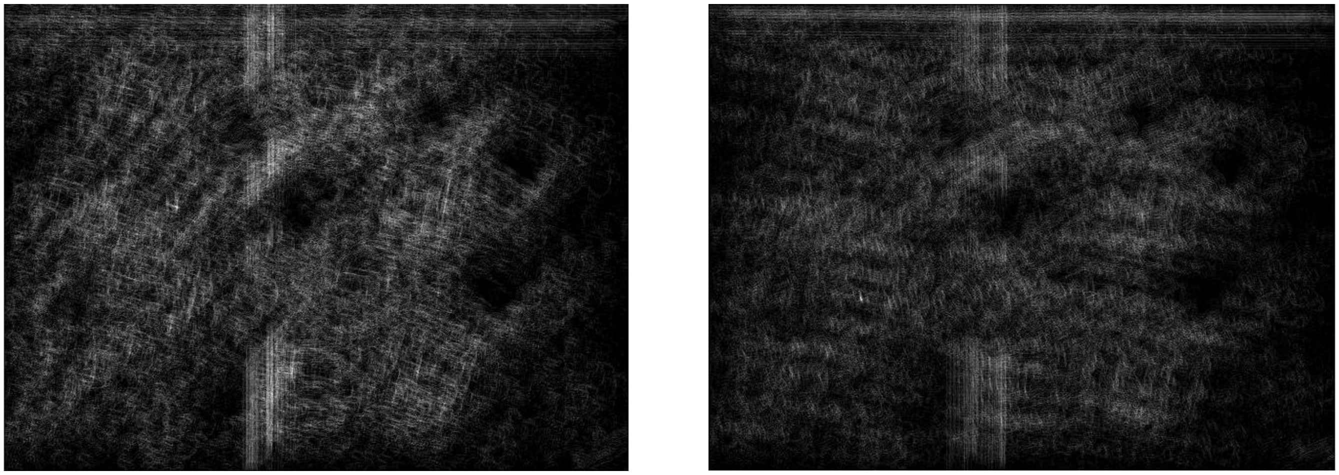
8	3	3	5	3	4	2	3	11	9	7	3	4	2	5	4	7	8	2	1	7	6
4	4	7	5	4	3	2	4	10	4	14	3	7	4	3	2	7	10	3	2	3	2
3	5	7	4	4	4	4	4	5	8	9	4	5	1	2	5	8	10	4	1	2	1
7	4	1	6	3	4	3	3	6	10	10	2	2	1	0	3	7	12	5	1	2	5
3	3	2	6	9	4	5	4	4	5	8	7	7	4	3	8	6	13	8	4	2	2
3	1	3	5	10	8	4	6	4	8	12	5	3	4	1	2	9	17	9	3	1	3
1	4	6	3	5	7	3	3	3	9	11	3	6	4	4	3	8	23	11	6	3	3

6	3	3	1	7	5	5	6	11	8	5	4	5	0	4	4	8	18	15	10	8	6
3	4	2	3	5	6	5	6	4	8	7	4	7	1	3	6	5	19	25	9	3	5
1	1	3	3	1	5	5	3	5	6	11	4	5	6	5	8	9	8	12	9	4	6
2	3	2	6	2	7	3	4	7	8	5	2	3	6	5	5	1	7	9	10	4	2
3	6	4	4	4	8	2	5	8	4	9	2	3	4	3	4	2	6	7	5	5	3
3	2	0	3	1	8	3	4	5	5	9	3	4	4	3	8	4	4	3	5	1	2
2	3	2	2	1	7	5	5	7	7	5	2	2	5	7	3	2	2	4	3	3	0
5	4	3	3	4	6	10	7	1	9	4	1	4	5	2	3	3	4	3	3	2	5

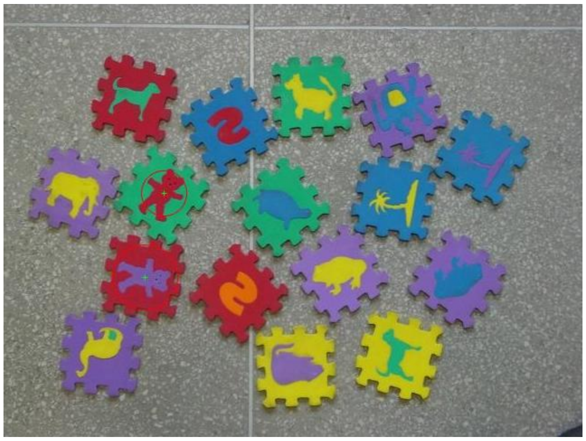
Bear #2 Edge Maps:



Bear #2 Accumulator Arrays:



Bear #2 Final Result:



Letter K:

Accumulator #1:

3	3	4	3	3	4	3	0	1	1	1	4	6	7	5	5	5	4	2	2	5	3
3	6	3	2	3	4	1	1	2	2	2	6	7	6	8	8	2	2	3	1	1	5
3	4	2	2	1	2	0	1	1	1	1	9	5	6	4	1	6	1	2	2	4	3
2	3	1	0	0	1	1	1	1	1	1	5	9	4	3	2	2	2	3	5	1	4
4	4	4	3	1	0	0	0	0	0	0	7	6	10	3	5	6	3	3	4	1	1
3	0	1	1	1	1	0	0	0	0	0	16	12	5	11	12	9	11	13	7	7	10
1	0	1	0	1	1	1	0	0	0	0	15	26	24	26	22	21	11	6	4	5	5
2	3	3	1	1	1	1	0	0	0	3	18	14	23	24	3	20	18	9	10	9	11
0	1	2	2	2	2	1	1	0	3	1	22	31	32	37	83	26	23	28	17	13	9
0	1	1	1	1	2	1	1	2	1	1	17	22	21	13	7	19	14	8	13	9	6
1	1	3	1	2	2	2	2	1	2	4	13	11	17	30	22	17	22	20	8	13	14
0	2	2	1	1	2	2	3	3	1	0	8	12	11	8	7	11	5	11	18	7	9
2	4	1	3	1	2	4	2	2	1	2	5	4	4	0	1	2	8	4	6	16	8
1	3	2	4	1	3	5	4	2	3	1	5	2	0	2	0	1	1	7	5	6	12

Second K of Accumulator 1:

1	2	6	4	6	4	4	4	4	7	6	8	6	8	6	6	4	1	6	4	3	2
1	4	2	3	4	5	5	2	4	3	1	8	6	5	3	3	5	5	10	4	5	7
2	3	4	6	5	4	4	5	6	5	4	7	7	5	5	5	3	4	4	5	4	3
1	3	2	2	2	2	2	2	2	2	3	17	8	8	8	5	6	5	2	5	2	3
1	3	2	2	2	3	4	4	4	4	2	9	15	10	8	21	12	10	11	11	14	12
1	3	3	4	3	2	2	2	2	3	4	20	19	24	23	12	16	18	11	5	2	1
0	2	1	0	0	0	0	0	0	2	6	12	14	9	21	25	11	9	11	4	7	12
3	3	1	1	2	2	2	4	6	4	7	16	19	27	16	28	34	22	18	19	14	5

```
3    3    3    1    0    2    5    2    3    6   10    9    8   12   21   17    9   18    8    4    8   13
4    5    0    2    5    2    2    4    7   12    7   12   11    8   12   14   10    7   19   15    8 10
8    9    6    1    1    2    6    8    9    7    4    7    8   11    7    7   10    7    2    8    9    9
10    8   10    6    4    8    9    5    6    1    2    6   10    5    4    2    1    6    5    4    9    8
11    9    4   10   11    7    1    4    1    3    0    4    7    1    6    3    4    2    5    6    3    7
4    9   10    6    9    9    7    3    1    0    0    3    3   10    3    5    7    3    6    3    5    5
```

Third K from Accumulator 2:

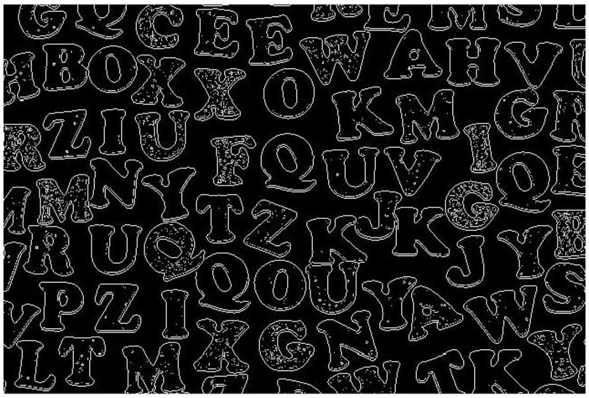
```
8    5    3    4    9    8   12    7    6    5    7    8    8    4    3    5    6    4    3    5    2    2
4    6    4    6    5    2    5    7    9    8    6    3    3    5    4    5    1    7    8    1    1    4
4    3    3    3    2    6    3    3    6    3    8   13    6    5    5    2    6    1    5    3    3    3
2    2    3    4    3    4    4    2    6    5    3    4   11    5    5    7    3    4    1    7    3    2
4    4    3    3    5    5    4    4    4    2    2    7    7   12    7    6    8    2    8    6    1    4
4    5    4    6    6    4    5    5    4    3    4   11   12   13   18    7   12   11    9   10    6    6
4    3    6    4    5    6    4    4    6    5    3    8   13   13   15   29   20   14   17    8    6    1
6    7    6    7    8    7    6    7    5    3    5   11   14   20   20   24   21   14   20   19   19 10
4    5    5    3    6    5    3    4    4    4    6   13   11   18   24   25   43   42   23   11    6   6
5    3    1    2    5    8    5    4    7    3    4   10   15   18   18   23   12   20   24   21  26  16
6    6    4    1    3    4    4    2    3    4    4   12    8    7   12   18   23   18   19   12    6   16
5    3    4    6    5    2    2    2    4    3    2    5    7    7    7    7    5   11    9   20   15   12
5    6    5    4    4    4    4    2    2    2    3    5    5    3    1    3    2    4    7    4   17   10
3    3    5    5    6    3    3    3    4    2    0    1    0    3    1    1    1    1    3    8    4   14
```

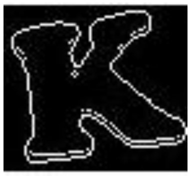
Fourth K from Accumulator 3:

```
2    8   12   16    2    4    3    5    7    4    6    0    0    0    3    3    0    1    0    3    0    2
10    3    9   13   16    6    1    3    4    5    2   10    5    0    0    1    1    0    3    0    2    0
6    6    1    6   16   15    3    5    6    4    7   44    7    8    0    0    3    3    0    2    0    2
5    9   13    3   11   16   16    7    5    3    3    0   45    6    7    0    0    2    2    0    2    0
0    5    8    9    2    9   16   11   11    3    3    0    0   47    7    5    0    0    2    2    0    3
2    0    3   14    6    2   16   17   10    9    2    0    0    0   53    7    4    0    0    1    1    5
5    2    0    4   10   10    1   12   14   12    7    0    0    0    0   52    5    2    0    0    0    0
0    2    1    0    2    7    6    0   11    9   13    0    0    0    0    0   53    6    3    0    0    0
1    0    3    0    0    2    8    6    0   10   16    2    0    0    0    0    0   51    7    1    0    0
0    0    0    2    0    0    1    8    5    0    7    3    0    0    0    0    0    0   51    6    2    0
```

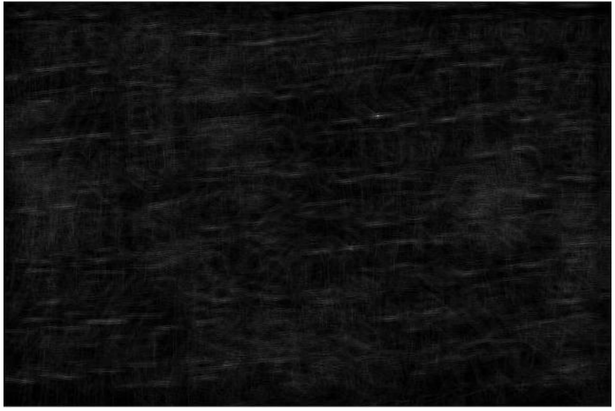
0	0	0	0	2	0	0	0	3	4	0	4	4	1	0	0	0	0	0	48	6	3
0	2	0	0	0	1	0	0	0	10	6	2	2	2	0	0	0	0	0	0	46	7
0	0	1	0	0	0	0	0	0	0	5	0	4	4	4	2	0	0	0	0	0	42
5	0	0	1	0	0	0	0	0	0	0	2	0	2	2	2	1	0	0	0	0	0
2	3	0	0	3	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0	0

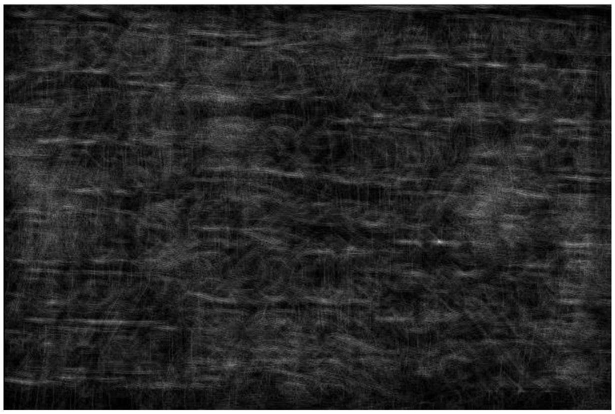
Letter K Edge Maps:



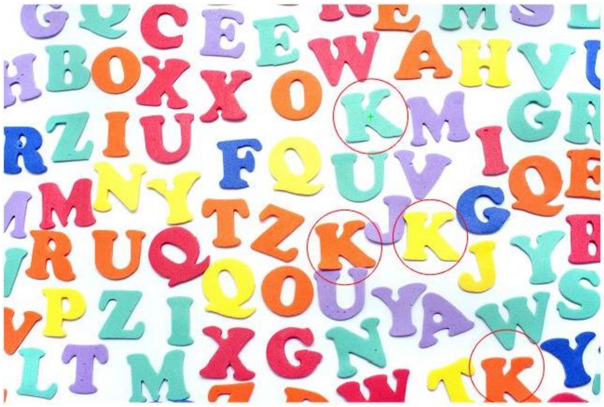


Letter K Accumulator Arrays:





Letter K Final Result:



Letter Q:

First Q from Accumulator #1:

```
15  15  13  17  12  16  13  19  24  20  13   5   9  13  16  12   9  15  12  16  8  12
17  16  12  10  14  18  22  22  23  18  13   9   3  11  18   9  16  11  18  12  13  13
12  18  14  15  18  18  23  17  26  16  17  11  13   8  21  16  14  25  17  14  13  11
16  12  18  19  16  20  22  20  15  20  15  10  10  12  13  23  29  24  14  16  12  16
16  23  19  22  15  20  19  20  18  21  15  11  11  14  16  46  22  25  21  14  16  16
18  18  22  14  15  17  23  14  22  19  20   9  12  14  35  15  33  29  37  24  18  11
16  19  18  17  16  18  20  19  18  14  10  11  24  30  18  18  69  35  22  14  19  13
16  17  17  18  15  18   7  20  14  16  10  18  27  25  19  55  71  27  10  21  13   9
15  20  16  17  16  15  22  16  16  13  10  18   9  23  44  32  41  31  29  25  14  11
17  17  16  14  17  11  17  11  15   6  10  15  21  28  25  21  16  41  23  12  12  10
14  16  14  14  16  13  10   9  11  12  11  20  26  20  22  24  43  30  21  11  12  11
13  15  16  11  14  12   7  18   9  12  10  19  14  11  13  22  15  14  16  12   9   9
17  16  13  12  19  13  14  16  12   6  12  11  10  15  14  12  13  10  18  10  11   8
16  15  12  14  11  11  11  10   8  10   8   7  15  13   5  15   9   9  14  12   6   7
```

Second Q from Accumulator #1:

13	16	15	14	11	14	17	16	16	10
14	9	7	10	15	15	20	11	22	13
11	7	13	5	11	15	20	14	22	18
7	10	10	4	6	11	23	25	23	17
5	11	12	7	16	13	25	34	34	16
14	6	5	12	20	14	26	62	26	16
8	9	6	9	15	30	35	39	40	18
7	2	13	5	21	36	37	19	23	23
5	5	12	14	23	22	16	27	32	31
5	5	10	22	12	12	11	26	29	16
6	6	17	15	11	13	27	27	20	17
9	12	20	15	13	20	13	11	11	15
12	13	13	15	17	6	11	9	13	15
6	12	14	18	5	7	12	12	11	14

Third Q from Accumulator #1:

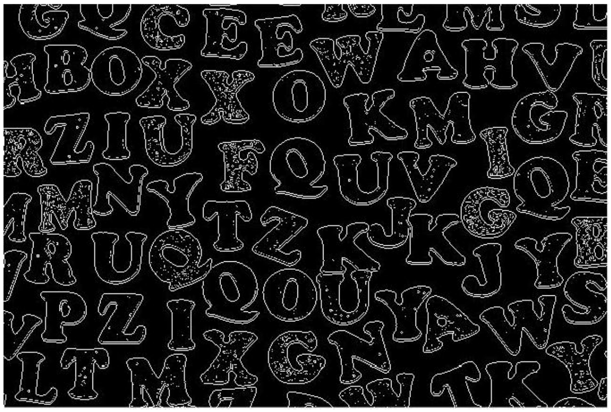
11	12	15	14	10	15	6	10	12	7	6	8
12	13	9	15	15	16	11	10	15	8	11	9
9	10	8	15	15	14	9	9	15	10	10	9
15	9	8	16	17	13	12	11	17	11	9	12
14	13	8	11	13	11	15	17	16	11	18	15
9	3	12	10	16	15	18	13	17	18	17	16
11	11	7	18	14	13	23	22	25	26	13	14
14	13	11	10	13	10	19	38	30	25	22	17
18	10	5	10	13	12	40	31	37	26	25	15
8	12	10	15	15	10	26	40	56	28	19	22
5	5	10	10	15	15	28	55	45	27	11	17
5	9	11	13	13	13	41	33	43	25	31	15
4	4	7	12	14	20	36	29	29	40	21	12
8	7	9	11	10	15	18	20	35	21	17	9
2	4	8	12	13	17	30	26	19	19	7	9

1	5	8	7	11	18	17	13	6	17	14	12

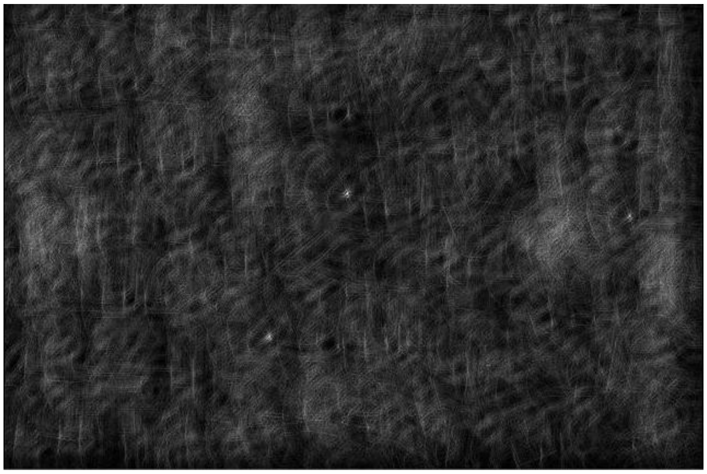
Fourth Q from Accumulator #2:

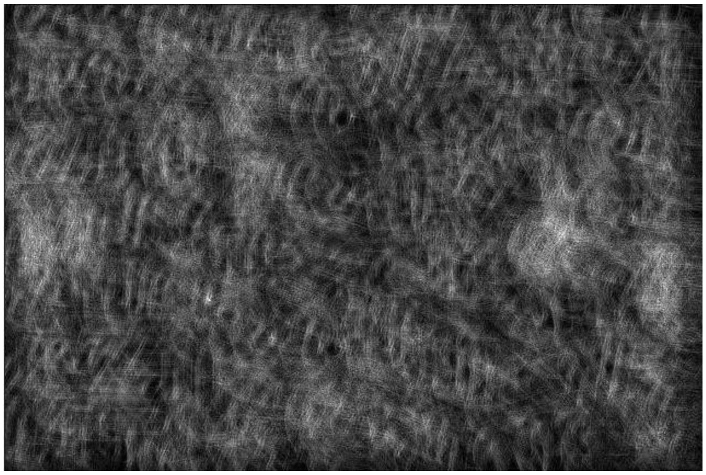
16	17	19	14	18	19	29	27	25	17	23
19	16	12	19	18	15	24	30	20	21	25
16	12	15	19	11	22	34	31	13	27	30
13	15	12	14	15	22	38	32	17	34	27
13	19	23	17	10	21	41	35	19	30	15
7	16	16	14	17	31	36	36	45	25	14
14	21	14	10	19	26	28	36	45	38	18
14	13	10	14	18	35	23	26	24	20	23
14	10	7	7	12	32	25	25	17	20	17
10	11	11	17	15	22	20	29	24	22	17
15	9	12	15	12	19	14	19	17	18	19
11	19	15	10	10	15	15	17	19	13	14
9	7	15	17	18	16	21	5	13	13	14
9	7	13	15	15	16	14	10	13	17	9

Letter Q Edge Maps:



Letter Q Accumulator Arrays:





Final Results for Letter Q: