EE 569: Homework #2

Issue: 1/20/2019 Due: 11:59PM 2/12/2019

Name: Wenjun Li
USC ID: 8372-7611-20
Email: wenjunli@usc.edu
Submission Date: 2/11/2019

Problem 1: Edge Detection (50%)

1. Motivation

In order to use compute to help us classify images, understand images, process images and so on, we need to let computer first understand images, i.e. what are there in images. Since image is 2D, so edge is one of the most important properties that objects can have on a flat pictures. Normally, people use discontinuities in brightness to detect edges. With time going on, people implement more advanced and complex techniques to detect edges in images. With edge detection, people can reduce unnecessary information but preserve important objects' structures: such as lines, corners, curves, etc.

2. Approach

- Sobel Edge Detector

In 1968, Irwin Sobel and Gary Feldman proposed a discrete difference operator to detect edges in digital images. As we know, the intensity values of pixel along the edge and on the edge difference a lot, i.e. they are very close but have apparent difference in intensity. If we take the derivative of the intensity function along horizontal direction (or vertical direction), we will observe a gradient decrease and a gradient increase. Based on this property, Sobel operator track the gradient variation along horizontal direction and vertical direction.

Below are the equations of how to compute gradient, magnitude and orientation.

$$\text{Gradient: } \nabla f = \begin{bmatrix} gx \\ gy \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \qquad \text{Magnitute: } |\nabla f| = \sqrt{g_x^2 + g_y^2} \qquad \text{Orientation: } \theta = tan^{-1}\left[\frac{g_y}{g_x}\right]$$

Since gradient is continuous in real life, we need to make the gradient operator discrete so that it can be applied on computer. Here comes the 3*3 Sobel mask (shown as below). We compute the gradient value of the center pixel using its 8-neighbor pixels and we repeat this operation to all the image pixels. The Gx Sobel mask calculate the horizontal gradients and the Gy Sobel mask calculate the vertical gradients.

$$Gx = \begin{vmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{vmatrix}$$

$$Gy = \begin{vmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}$$

After we obtain the x-gradient and y-gradient values, we need to normalize them, because the value might be out of the range of 0-255. We take the absolute value and map all the values to 0-255. Then we have the normalized x-gradient map and y-gradient map (Results are shown in Result part).

Next, we need to combine Gx and Gy and get G, i.e. $|\nabla f| = \sqrt{g_x^2 + g_y^2}$. In order to obtain a best edge map, we still need to remove those less-probable edges and keep those strong-edges. To achieve this, here we set threshold (in terms of percentage) to filter G and binarize G.

$$G = \begin{cases} 0 & G \le T \\ 255 & T < G \end{cases}$$

(Note, in my code I set intensity value as Threshold, since it is more easily to manipulate. I have converted the intensity value threshold into percentage threshold in *Result* part.)

- Canny Edge Detector

As we can see from the *Result* part, the Sobel detector is not that good. So, researchers proposed a new method to enhance the performance of edge detection, which is called Canny Edge Detector. Canny detector applies intensity gradients and non-maximum suppression with double thresholding. The main steps of Canny detection are below:

Filter image with Gaussian filter;
Find magnitude and direction of gradient using Sobel mask;
Non-maximum suppression (NMS);
Double thresholding (Hysteresis thresholding);

The first and second step are easy to understand and implement. Here I introduce non-maximum suppression and double thresholding.

So, what is NMS and what does NMS do? Basically, NMS suppresses and removes those pixels are not considered to be part of an edge. Therefore, NMS will return a result only with thin lines and edges and no blurring or rough edge region. NMS is trying to suppress all the gradient values to "0" except the local maximum. In the direction of gradient, we compute C1 and C2 (intersection potion with Sobel mask edge along the direction of θ) using interpolation. If C is larger than C1 and C2, i.e. local maximum, we keep it; if C is smaller than C1 and C2, we suppress C to "0" (remove it).

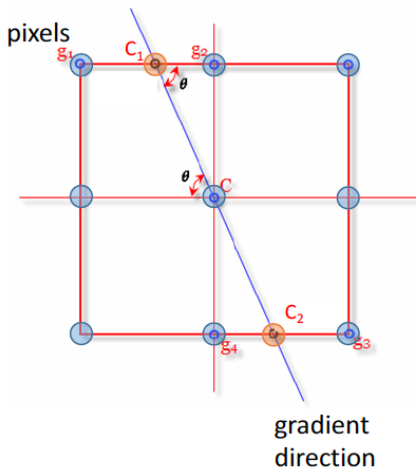


Figure 1 NMS Algorithm

After NMS is applied, we will have thinner edges and for the final step, we implement double thresholding. In order to do double thresholding, we need to set an upper threshold and a lower threshold and then do following comparisons:

(a) If a pixel gradient value is higher than the upper threshold, the pixel is accepted as an edge;
(b) If a pixel gradient value is lower than the lower threshold, the pixel is rejected;
(c) If a pixel gradient value is between the 2 thresholds, then it will be accepted only if it is connected to a pixel that is above the upper threshold.

Note: usually, the ratio of upper/lower is between 2:1 and 3:1.

After we repeat all the above steps to all the pixels within the image, Canny edge detection is done.

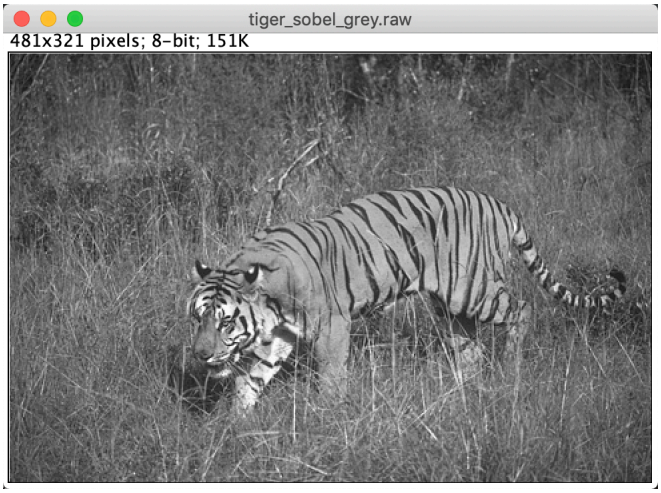
- Structured Edge Detector

The summary of SE detection, flow chart and random forest description are in the discussion part. Please refer to Page.13.

3. Result

- Sobel Edge Detector Result

Tiger results (Note: the last number of the screenshot title is intensity threshold; the percentage in the table is percentage threshold)

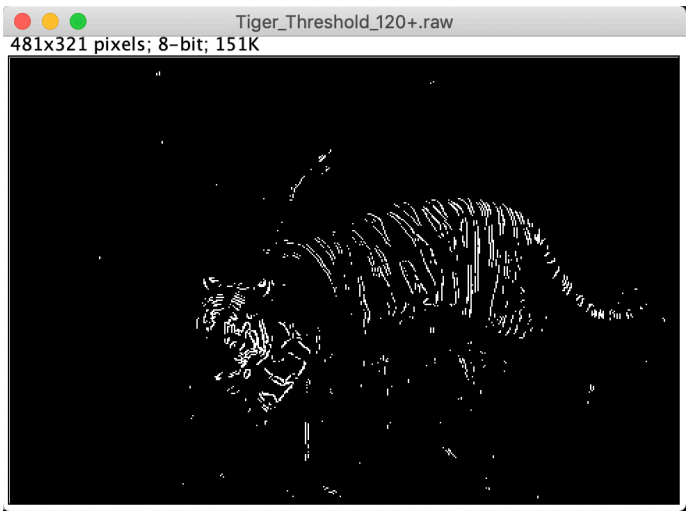
Tiger.raw	Tiger_grey.raw
	
Tiger_Gx.raw	Tiger_Gy.raw
	
Tiger_Gx.raw	Tiger_sobel_threshold_39%.raw

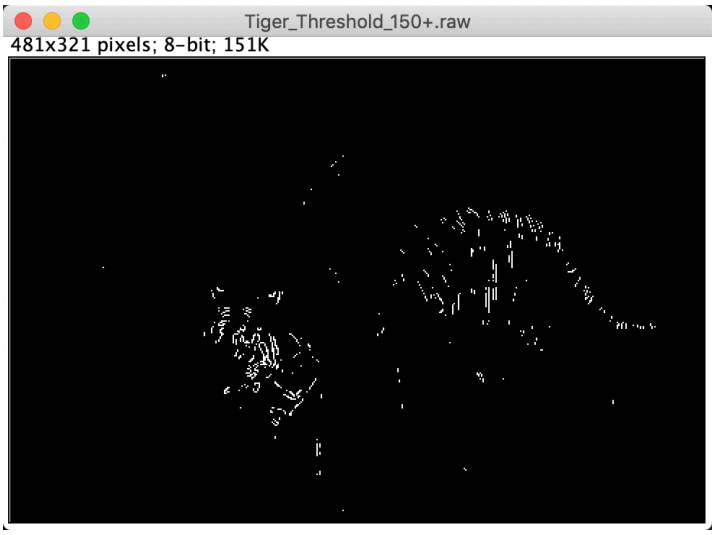
Tiger_sobel_threshold_43%.raw

Tiger_sobel_threshold_45%.raw

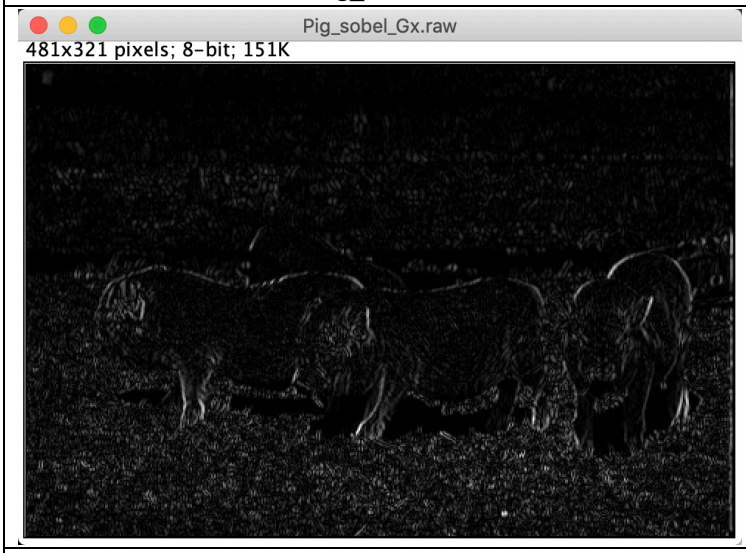
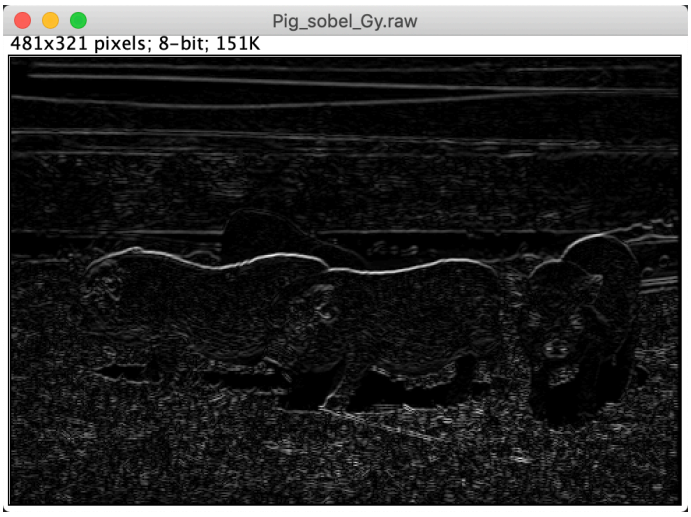
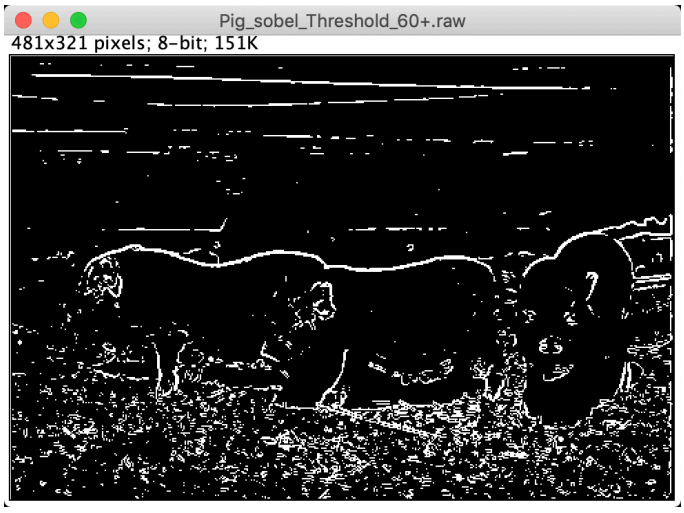
Tiger_sobel_threshold_47%.raw

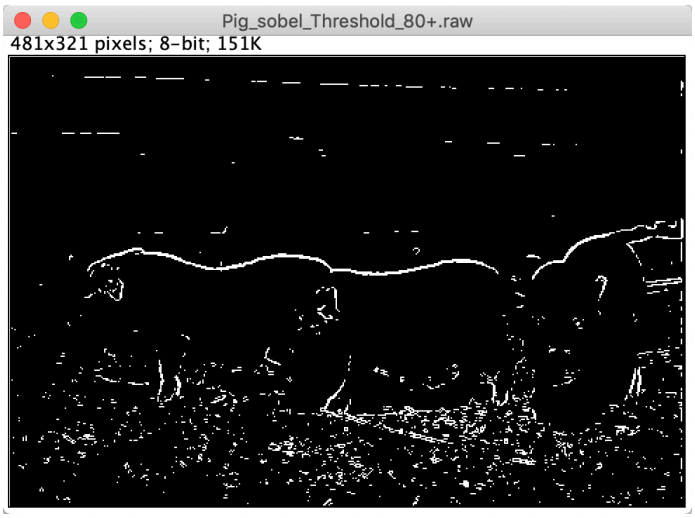
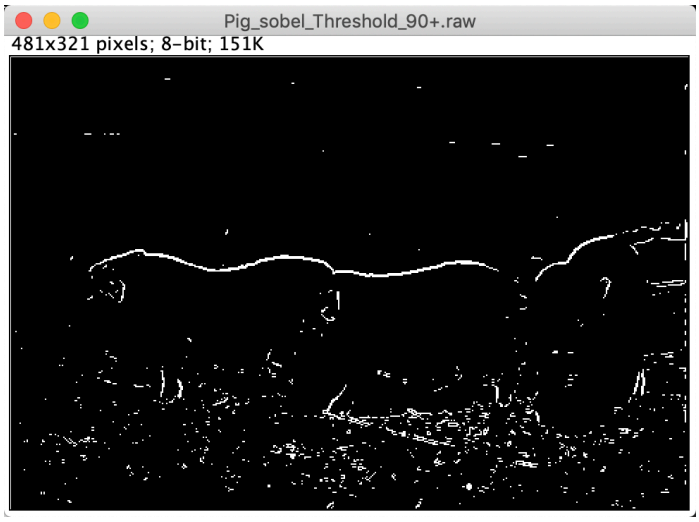
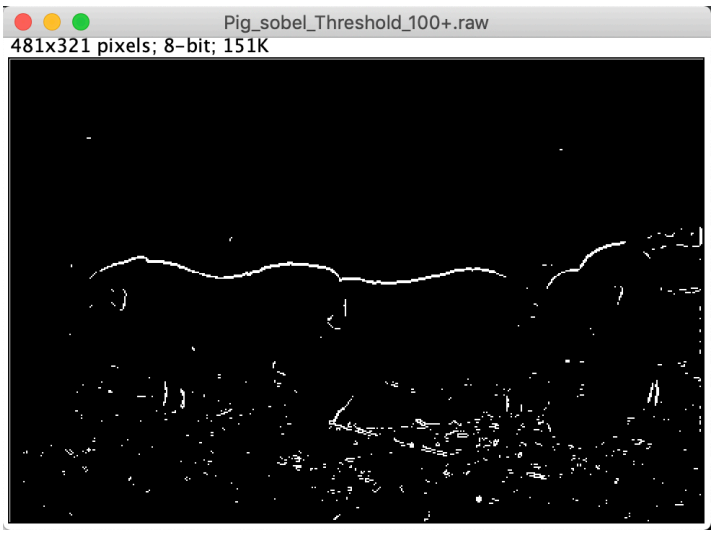
Tiger_sobel_threshold_59%.raw





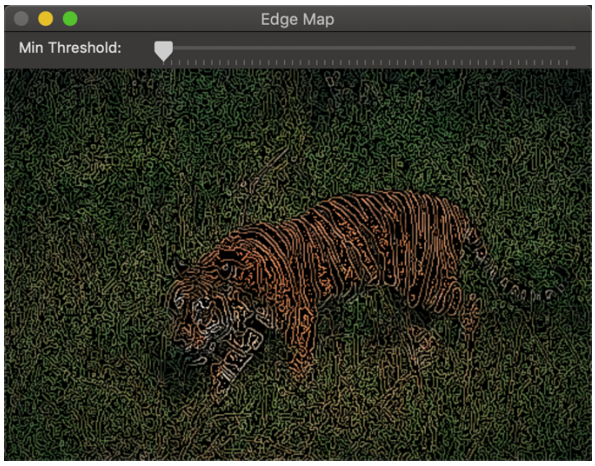
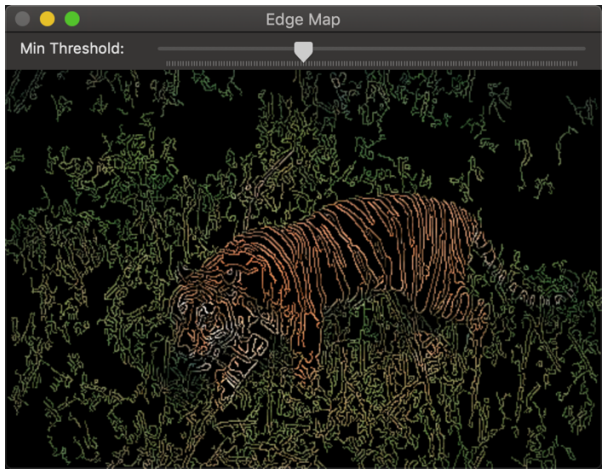
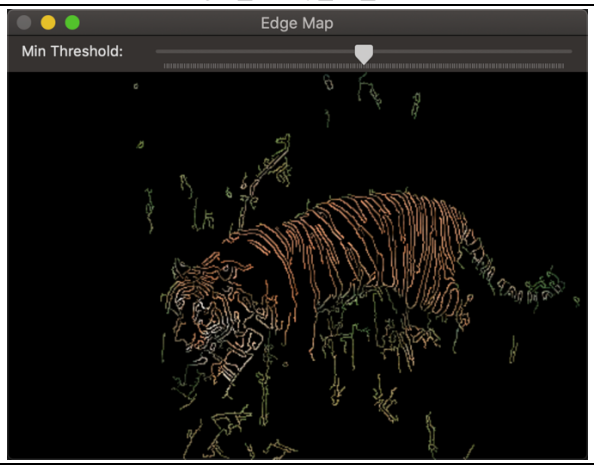
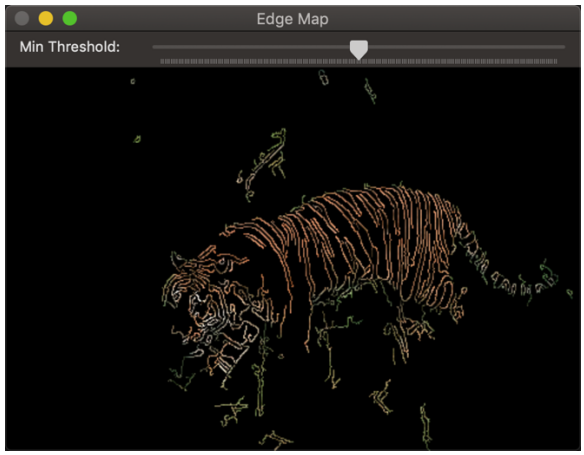
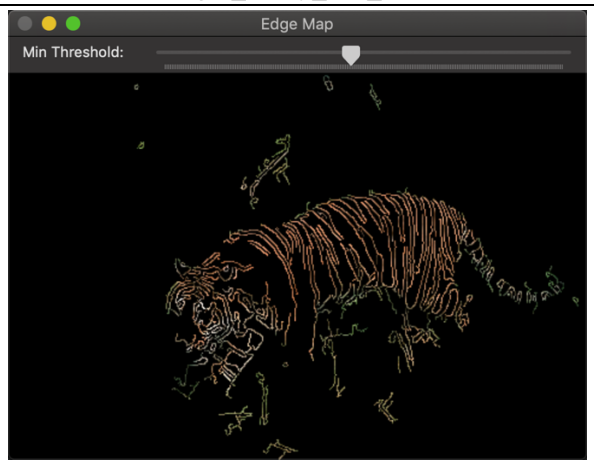
Pig results(Note: the last number of the screenshot title is intensity threshold; the percentage in the table is percentage threshold)

Pig.raw	Pig_grey.raw
	
Pig_Gx.raw	Pig_Gy.raw
	
Pig_G.raw	Pig_sobel_threshold_23%.raw
	

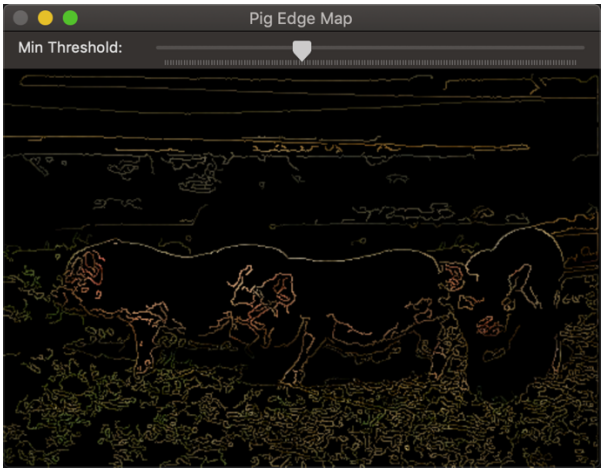
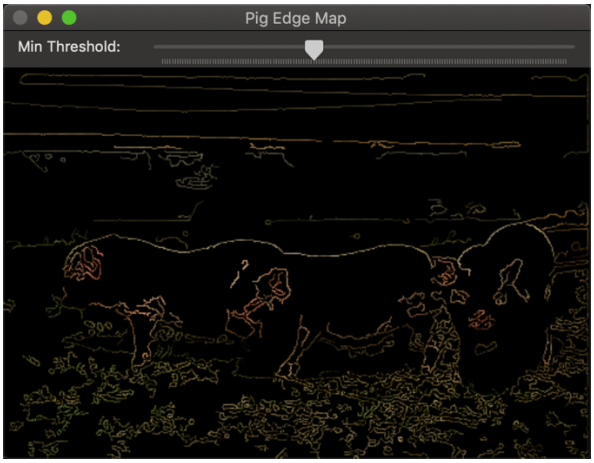
Pig_sobel_threshold_27%.raw	Pig_sobel_threshold_31%.raw
	
Pig_sobel_threshold_35%%.raw	Tiger_sobel_threshold_39%.raw
	

- Canny Edge Detector Result

Tiger results (the 1st number is lower threshold; the 2nd number is upper threshold)

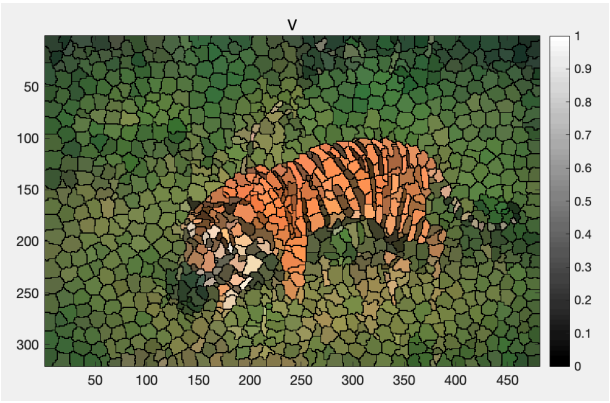
Tiger_canny_0_50	Tiger_canny_50_150
	
Tiger_canny_90_180	Tiger_canny_100_200
	
Tiger_canny_110_220	Tiger_canny_50_200
	

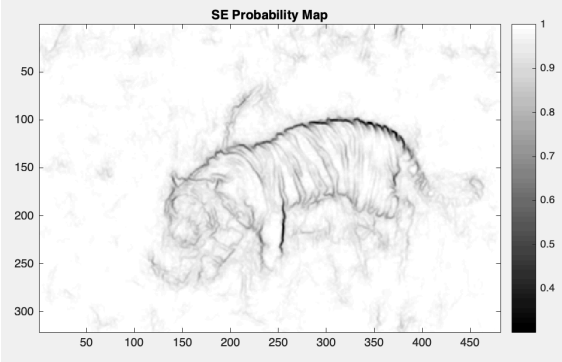
Pig results (the 1st number is lower threshold; the 2nd number is upper threshold)

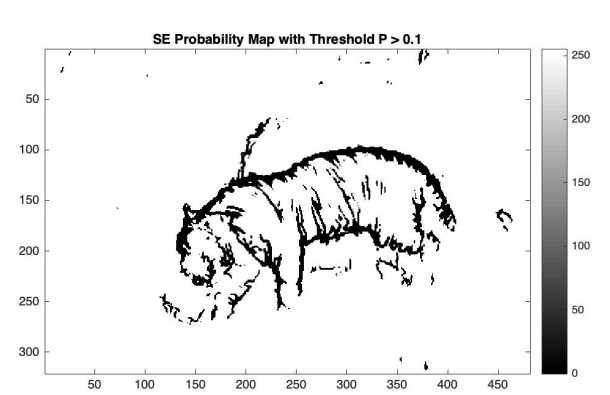
Pig_canny_50_150	Pig_canny_60_160
	
Pig_canny_80_180	Pig_canny_100_200
	

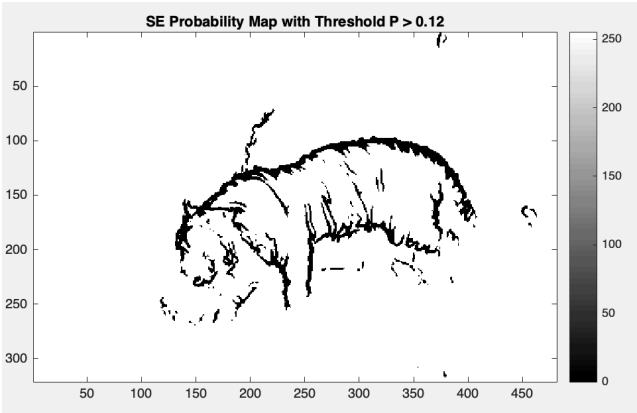
- SE Detector Result

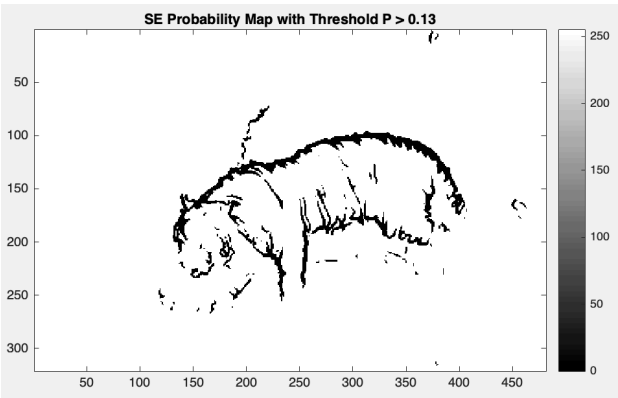
Tiger Results (Note: Threshold with P > (·) is shown in title)

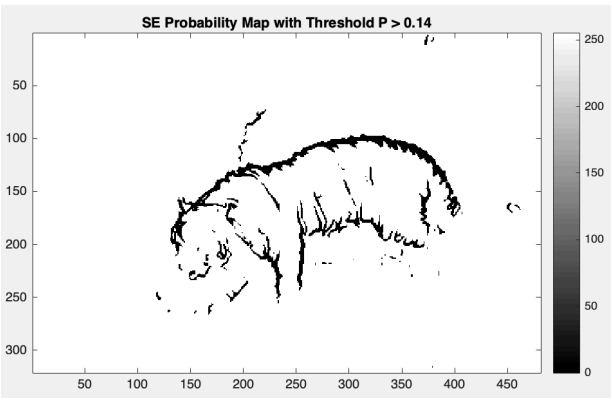


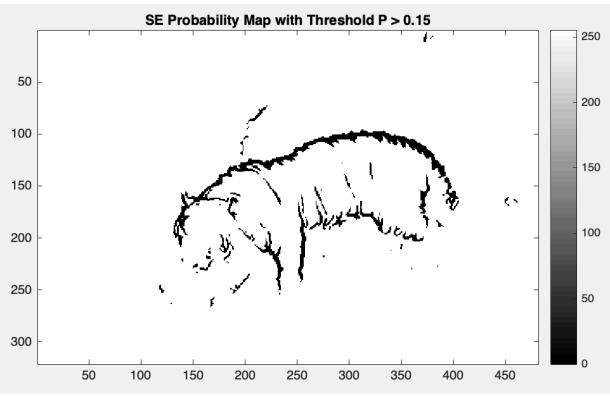


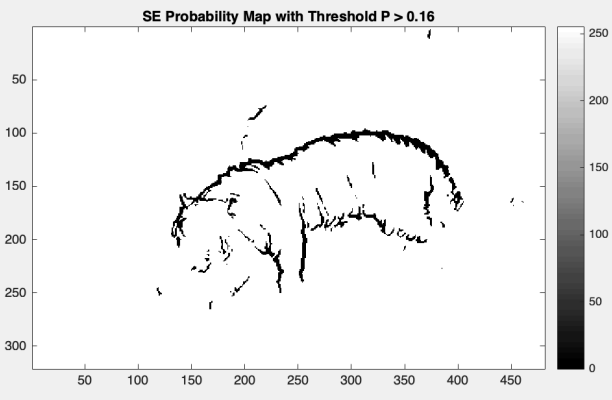


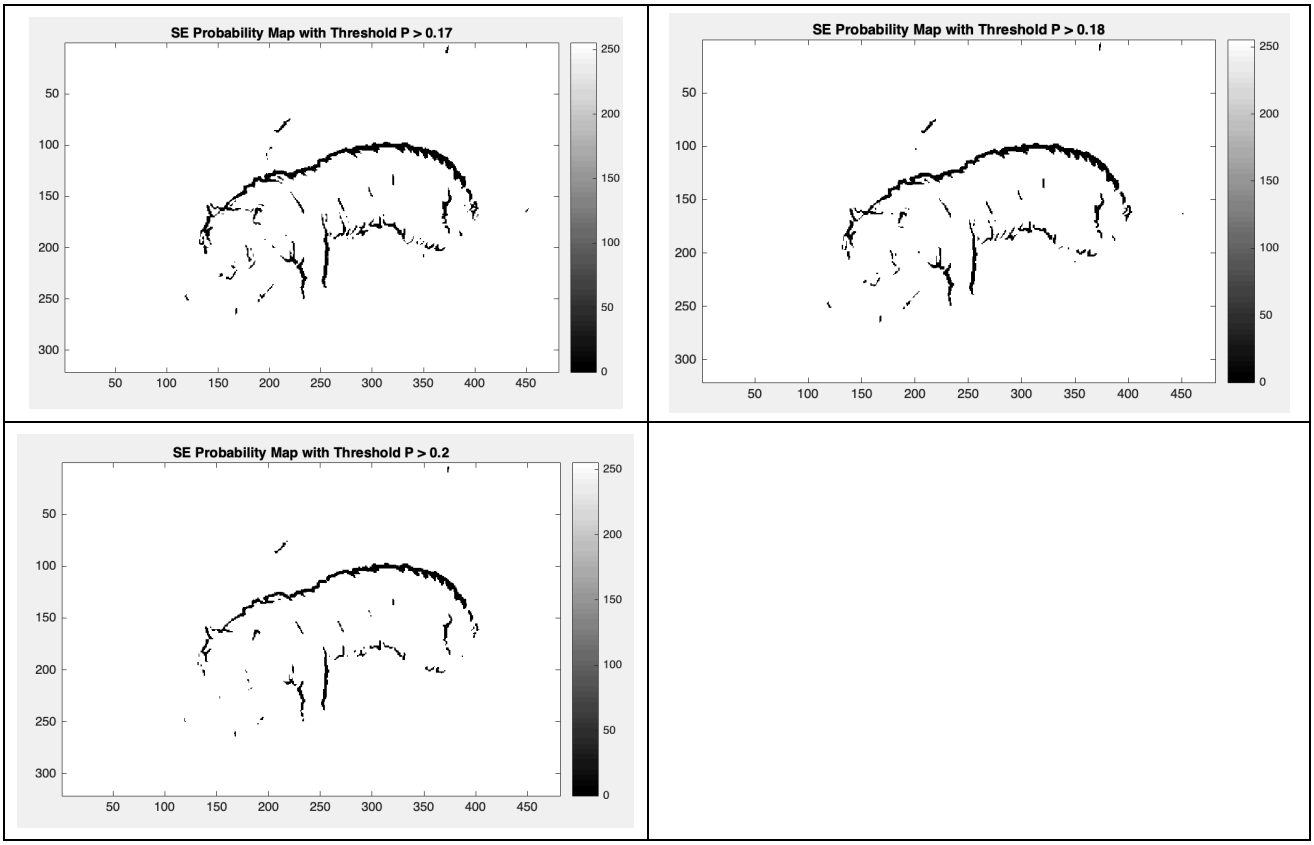




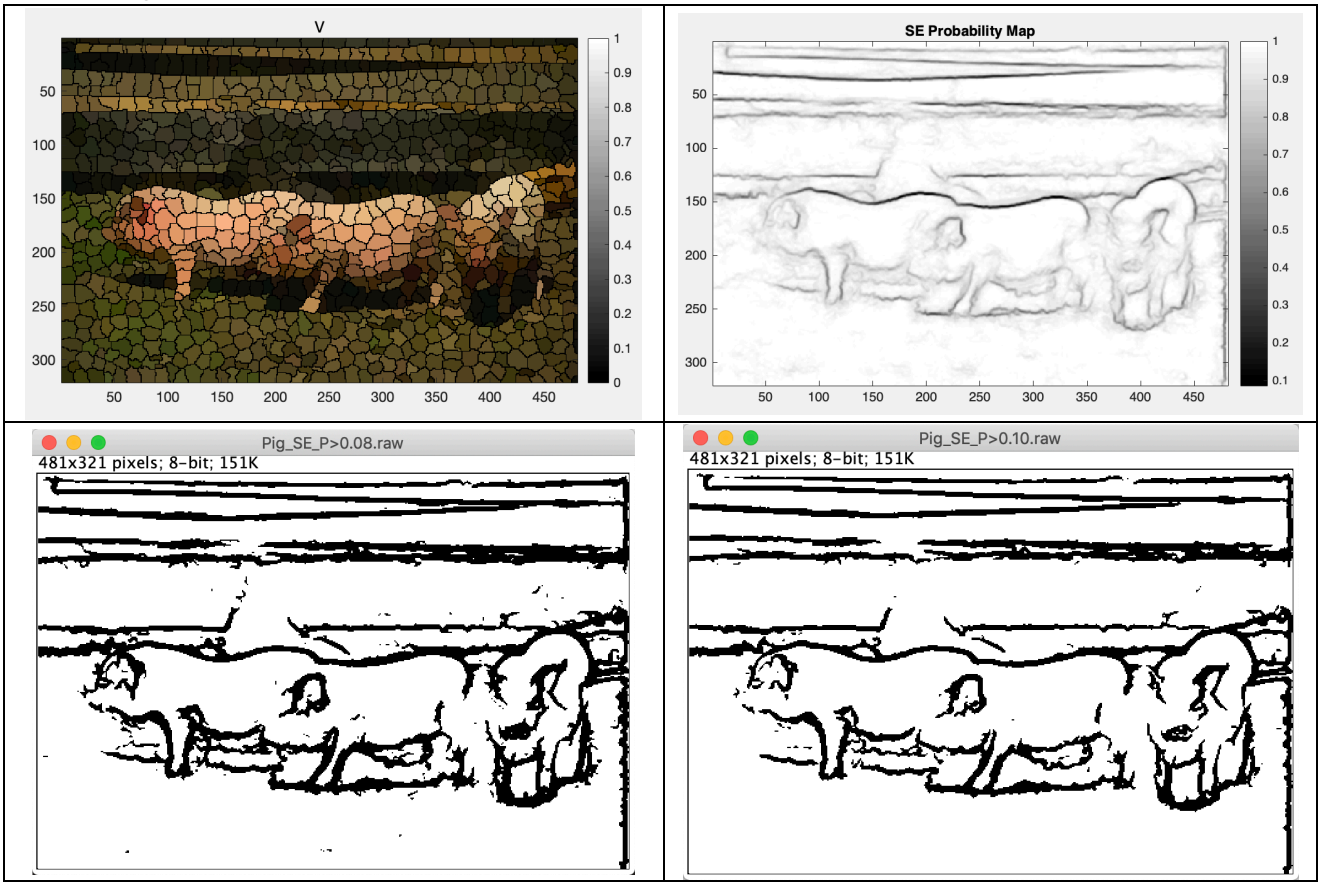


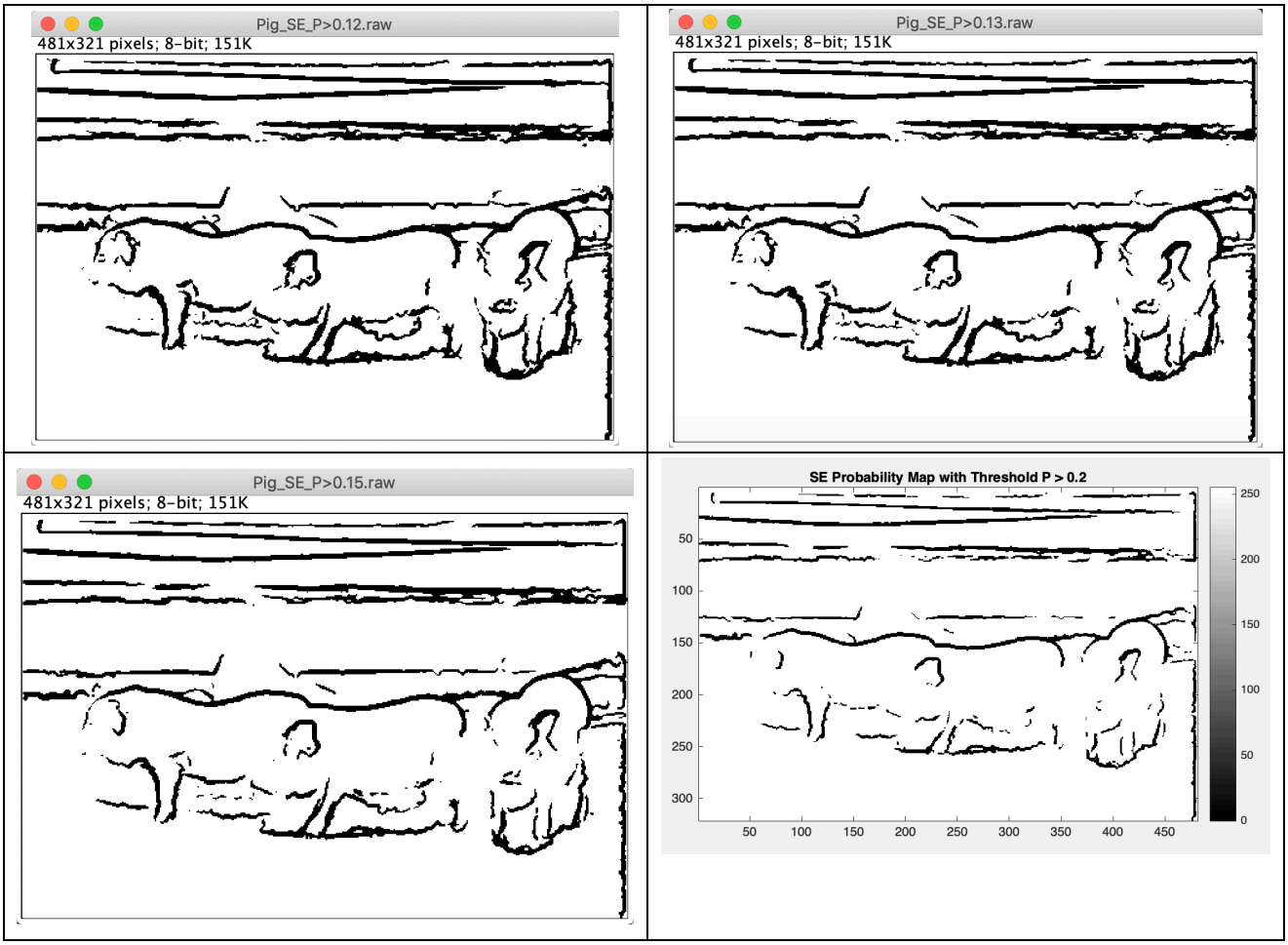






Pig Results (Note: Threshold with P > (·) is shown in title)




Pig_SE_P>0.12.raw
481x321 pixels; 8-bit; 151K
Pig_SE_P>0.13.raw
481x321 pixels; 8-bit; 151K
Pig_SE_P>0.15.raw
481x321 pixels; 8-bit; 151K
SE Probability Map with Threshold P > 0.2
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
250
200
150
100
50
0

4. Discussion

(1) Summary of SE explanation and a flow chart

With technology develops, nowadays, people applied data driven detection to find edges in images. And structured edge detection is one of data driven methods. For data driven methods, image data will be split into 2 group: one is training data and another one is testing data. In training process, computer will "self-learn" features and properties of edges and then computer will apply what is has learned to test data and check its learning result.

Structured edge detection means we split an image into many small patches, and try to find inherent edge properties in those small patches. Why this idea would be feasible? Because edges within a small region (i.e. a small patch) will be highly independent with other factors that might affect edge detection (such as blurring, noise, etc.), and they will display more general edge properties, such as straight line, Y-junction, T-junction, parallel lines, and etc. Here we call these inherent edge properties as local structures.

After finding those edge properties, we then attach a label to each of these properties, such as string line, curve, T-junction, etc. When all the features have a corresponding label, we apply a generalized structured learning approach to train this SE detection forest. SE detection formulate this problem of edge detection as predicting local segmentation masks give input image patches. By using structured labels to determine the splitting function at each branch in the tree, the decision forest of SE detection can predict a patch of edge pixel labels and aggregate them to form the final edge map.

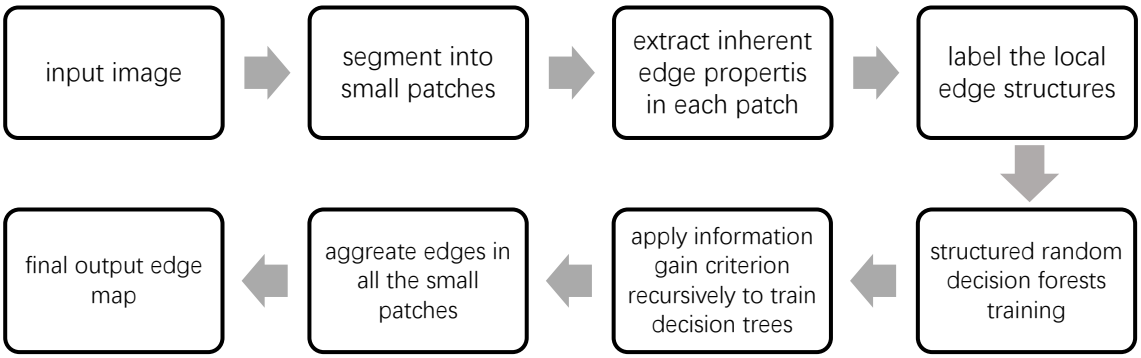


Figure 2 Flow Chart of Structured Edge Detection

(2) Decision Tree Construction and Principle of Random Forest (RF) classifier

SE assumes that the output space is structured but operate on a standard input space, i.e. the image is structured during processing. Here SE divides the whole image into a lot of small image patches. First, SE detection algorithm labels the patches with "edge" or "no edge" label. After preparing these training data and labels, SE detection algorithm builds a mathematical way to convert these edge features to vectors (or matrices).

Next SE detector continues to train its random decision forest. In this training process, each tree is trained independently in a recursive way. A decision tree $f_t(x)$ classifies a sample $x \in X$ by

branching left or right down the tree until a leaf node is reached, i.e. each node j in the tree is a binary decision:

$$h(x, \theta_j) \in \{0, 1\}$$

where θ_j is the parameter to make the decision. The goal of training is to find a parameters θ_j of the binary decision function $h(x, \theta_j)$ that can reach a "good split" of the data for each node j and training set *X*. Here we define an information grain criterion (below) for above purpose:

$$I_j = I(S_j, S_j^L, S_j^R)$$

where $S_j^L = \{ (x,y) \in S_j \mid h(x, \theta_j)=0 \}$, $S_j^R = S_j / S_j^L$. The θ_j is trained to maximize the information gain I_j. The training process will stop when θ_j achieves a maximal depth or become smaller than the previously-set threshold. For multiclass classification, the standard definition of information gain is:

$$I_h = H(S_j) - \sum_{k \in \{L,R\}} \frac{|S_j^k|}{|S_j|} H(S_j^k)$$

where $H(S) = \sum_y p_y log(p_y)$ is the Shannon entropy and p_y is the fraction of elements in S with label y.

Generally, individual decision trees are more likely to overfit and have high variance. To conquer this, SE introduces more decision trees and combine their outputs to form a random decision forest. To obtain random decision trees, SE detector randomly sampling the training data or sampling the features obtained before training individual decision trees. In fact, here is a trade-off between accuracy of each decision tree and the diversity ensemble.

(3) Compare and comment on the visual results of the Canny detector and the SE detector.

As you can see from the result images of SE detector and Canny detector, the result of SE detector is better than that of Canny. Canny detector can draw a thin edge line, but the edge line needs a good corresponding threshold. If the threshold is not chosen properly, the detection result will be bad. For SE detector, it removes more non-edge information and irrelevant information.

(d) Performance Evaluation (Advanced: 15%)

1. Motivation

For different people, each may have different opinions about what is an edge and what is not. In order to enhance our processing algorithm, parameters and result image, we need to set a standard, based on which we can quantitatively improve the detection result. To solve this, researchers proposed performance evaluation.

2. Approach

To do performance evaluation, first we need to claim what is an edge. In order to make this claim fair, we take 5 people's results, which is called ground truth. Take Figure 3 as an example: we take the mean of the sum of the 5 ground truth and produce a final ground truth, based on which we will continue to do performance evaluation.

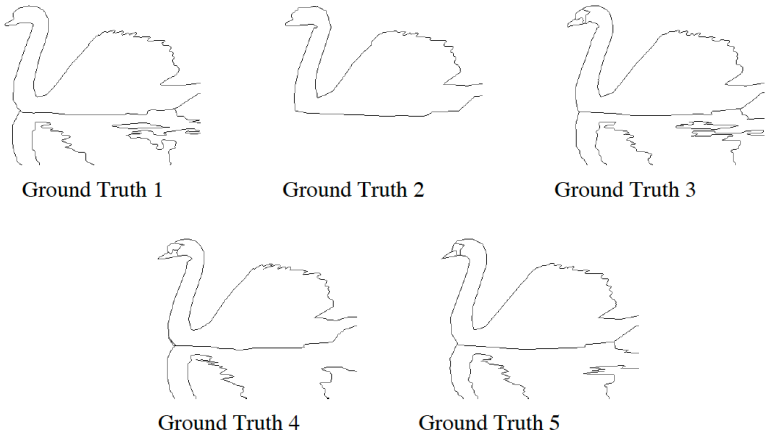


Figure 3 Ground Truth Example

There are 4 types of possible detection result, which are:

1. True positive (Edge pixels in the edge map coincide with edge pixels in the ground truth. These are edge pixels the algorithm successfully identifies).

2. True negative (Non-edge pixels in the edge map coincide with non-edge pixels in the ground truth. These are non-edge pixels the algorithm successfully identifies).

3. False positive (Edge pixels in the edge map correspond to the non-edge pixels in the ground truth. These are fake edge pixels the algorithm wrongly identifies).

4. False negative (Non-edge pixels in the edge map correspond to the true edge pixels in the ground truth. These are edge pixels the algorithm misses).

In this case, (1) and (2) are correct and (3) and (4) are wrong. We define correct detection and incorrect detection as Precision and Recall respectively:

$$\text{Precision: } P = \frac{\#True\ Positive}{\#True\ Positive + \#False\ Positive}$$

$$\text{Recall: } R = \frac{\#True\ Positive}{\#True\ Positive\ +\ \#False\ Negative}$$

And use P and R, we can calculate the F measurement, which is the evaluation parameter for result quality:

$$F = 2 * \frac{P * R}{P + R}$$

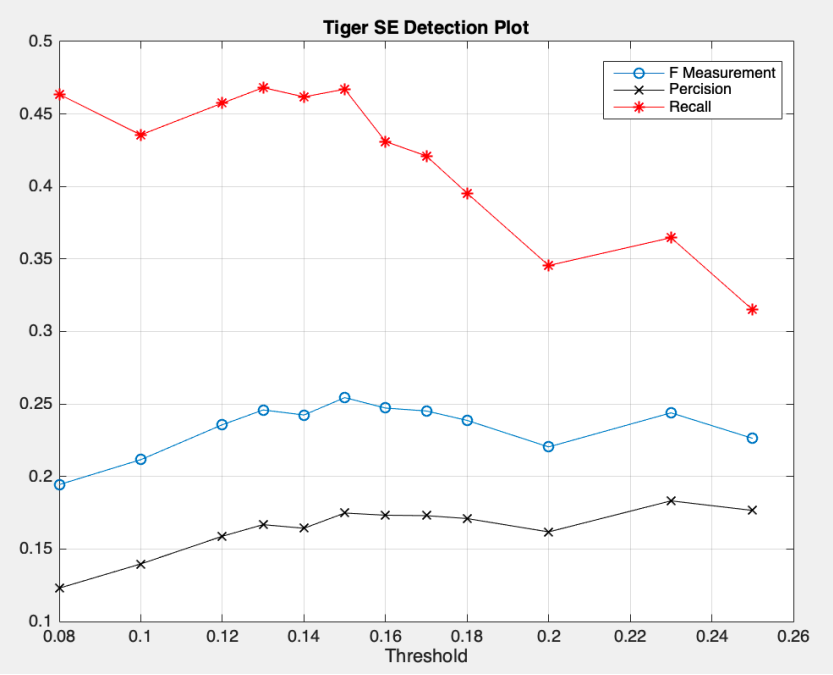
When choosing parameters, we need to take both precision and recall into consideration, since they both have influence on F measurement. Generally, a higher F measurement represents a better edge detector.

3. Result

- **Performance Evaluation for Structured Edge Detection**

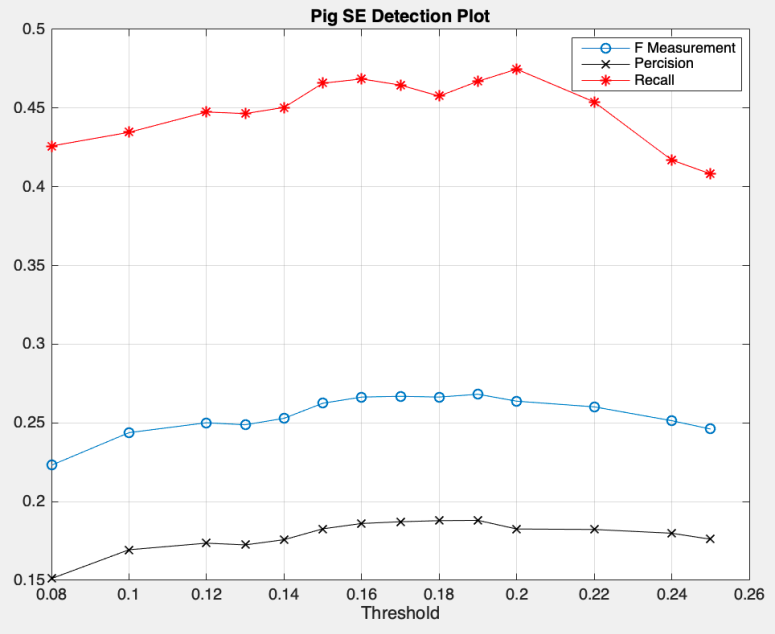
Note: the precision and recall here is already the mean precision and mean recall

Tiger SE	F measurement	Precision	Recall
Threshold=0.08	0.1944	0.1230	0.4636
Threshold=0.10	0.2116	0.1397	0.4356
Threshold=0.12	0.2357	0.1588	0.4575
Threshold=0.13	0.2459	0.1688	0.4682
Threshold=0.14	0.2424	0.1643	0.4618
Threshold=0.15	0.2544	0.1748	0.4671
Threshold=0.16	0.2472	0.1733	0.4310
Threshold=0.17	0.2452	0.1730	0.4211
Threshold=0.18	0.2387	0.1710	0.3954
Threshold=0.20	0.2204	0.1618	0.3455
Threshold=0.23	0.2439	0.1832	0.3647
Threshold=0.25	0.2263	0.1766	0.3150



Pig SE Pig	F measurement	Precision	Recall
Threshold=0.08	0.2231	0.1511	0.4259
Threshold=0.10	0.2438	0.1694	0.4346
Threshold=0.12	0.2501	0.1736	0.4475
Threshold=0.13	0.2488	0.1725	0.4465
Threshold=0.14	0.2529	0.1758	0.4504
Threshold=0.15	0.2625	0.1827	0.4658
Threshold=0.16	0.2664	0.1861	0.4685
Threshold=0.17	0.2669	0.1872	0.4646

Threshold=0.18	0.2664	0.1879	0.4576
Threshold=0.19	0.2682	0.1881	0.4670
Threshold=0.20	0.2638	0.1826	0.4747
Threshold=0.22	0.2602	0.1823	0.4539
Threshold=0.24	0.2514	0.1799	0.4170
Threshold=0.25	0.2461	0.1761	0.4082



- **Performance Evaluation for Sobel Edge Detection**

Tiger Sobel	F measurement	Precision	Recall
Threshold=100	0.2202	0.1326	0.6475
Threshold=110	0.2307	0.1418	0.6186
Threshold=115	0.2289	0.1427	0.5774
Threshold=120	0.2261	0.1433	0.5349
Threshold=150	0.2523	0.1814	0.4140

Pig Sobel	F measurement	Precision	Recall
Threshold=60	0.1622	0.0925	0.6611
Threshold=70	0.1705	0.0988	0.6216
Threshold=80	0.1866	0.1110	0.5862
Threshold=120	0.1801	0.1116	0.4659
Threshold=150	0.1779	0.1155	0.3876

As you can see from the above 2 tables, the Recall of Sobel is higher than that of SE, but the Precision is lower than that of SE. And overall, the F measure is lower than SE, which implies worse detection results.

- **Performance Evaluation for Canny Edge Detection**

The F measure for Canny is better than Sobel, and worse than SE, i.e. the quality of detection

results is between Sobel and SE.

4. Discussion

(1) Comment on the performance of different edge detectors (i.e. their pros and cons)

Sobel:

- Pros: easy to apply;
  has a quick computing speed;
  high Recall;
- Cons: detection result is worse than Canny and SE;
  low precision;
  can only return rough edges;
  sensitive to non-edge information (easy to be affected by noise, edge-like patterns, etc.);
  only has vertical and horizontal detection direction, so, not sensitive to other directional edge;

Canny:

- Pros: easy to apply;
  relatively quick detection speed;
  good detection results;
  not easily to be affected by noise;
  has 45° detection direction, can detect multiple directional edges;
- Cons:
  Need more computation than Sobel;
  cannot draw a good contour;

SE:

- Pros: best edge detection result;
  can draw a good contour;

- Cons: highly computationally intensive, needs a lot computation and runtime;
  Not suitable for simple edge detection, only suitable for complex edge & contour detection (i.e. no need to implement SE to detect those image with only simple edges)

(2) Which image is easier to get a high F measure, Tiger or Pig? Please provide an intuitive explanation to your answer?

Pig image is more easily to get a high F measure. Because there are a lot of stripes on tiger's body in the Tiger image and some people will not consider these stripes as a part of contour or edge of a tiger. Therefore, ground truth differs a lot for Tiger image and this will influence the final F measure. In contrast, people will more likely to reach a consensus on what is the pig's contour or edge and this lead a more compatible ground truth for the Pig image, which have good effect on calculating the F measure for Pig image.

(3) Discuss the rationale behind the F measure definition. Is it possible to get a high F if precision is significantly higher than recall, or vice versa? If the sum of precision and recall is a constant, show that F measure reaches the maximum when precision is each to recall.

No, it is not possible to get a high F if one of precision and recall is significantly large and the other is small.

Proof:

$$F = 2\cdot\frac{P\cdot R}{P+R} = 2\left(\frac{1}{P}+\frac{1}{R}\right)$$

let $P+R=C$ (C is a constant).

then $R = C-P$.

$$\therefore F = 2\cdot\frac{P(C-P)}{P+C-P} = 2\cdot\frac{PC-P^2}{C} = 2P+\frac{-2P^2}{C}$$

$$\frac{dF}{dP} = 2+\frac{-4P}{C} \quad \text{let } \frac{dF}{dP}=0$$

$$2+\frac{-4P}{C}=0 \Rightarrow P=\frac{C}{2}$$

$\therefore$ when $P=\frac{C}{2}=R$, F reaches the maximum.

Problem 2: Digital Half-toning (50%)

(a) Dithering (Basic: 15%)

1. Motivation

Normally, digital photos and pictures have a very brand color range (for example 256 grey level for grey images), which is unlikely to be printed by a simple printing machine, since simple printing machines only have 2 kinds of ink: white and black. So, here comes the question, how to print a color image or a grey image with only several color inks. For grey image halftoning, we want to express "grey level" using only black dots and white dots. Thanks to the human visual property, we can achieve this by mixing the black dots and white dots in a certain percentage.

There are several ways to do halftoning, and I will introduce their methods below.

2. Approach

- Constant Thresholding

  Constant thresholding is the easiest way to do halftoning. In most cases, we set the threshold as 127 for images with grey level 0-255. If the pixel value is less than 127, we set it as "0", i.e. the white dot; if the pixel value is greater than 127, we set it as 255, i.e. the black dot.

- Random Thresholding

First, we generate a random number in the range of (0,255) for each pixel F(i, j) and set this T(i, j) as threshold. Then we compare T(i, j) with the pixel value F(i, j). If T(i, j) is greater, then map the pixel value to 0; otherwise, map the pixel value to 255. i.e. (F is the input pixel, G is the output pixel)

$$G(i,j) = \begin{cases} 0 & if\ 0 \le F(i,j) < rand(i,j) \\ 255 & if\ rand(i,j) \le F(i,j) < 256 \end{cases}$$

In the output image, we display G(i ,j).

- Dither Matrix

Unlike random thresholding, in this dithering matrix method, we set the dithering matrix manually and use this dithering matrix to obtain the threshold T(i, j). For example, a 2*2 dithering matrix is given by:

$$I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

and larger dithering matrix follows this formula:

$$I_{2n}(i,j) = \begin{bmatrix} 4 \times I_n(i,j) + 1 & 4 \times I_n(i,j) + 2 \\ 4 \times I_n(i,j) + 3 & 4 \times I_n(i,j) \end{bmatrix}$$

We can transform this index matrix into a threshold matrix T for any input gray image by the following equation:

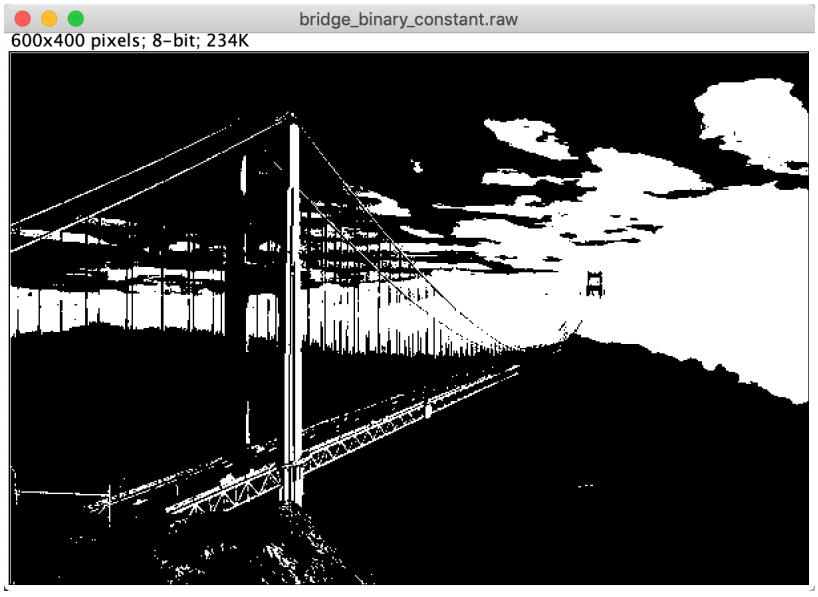
$$T(i,j) = \frac{I(i,j) + 0.5}{N^2} \times 255$$

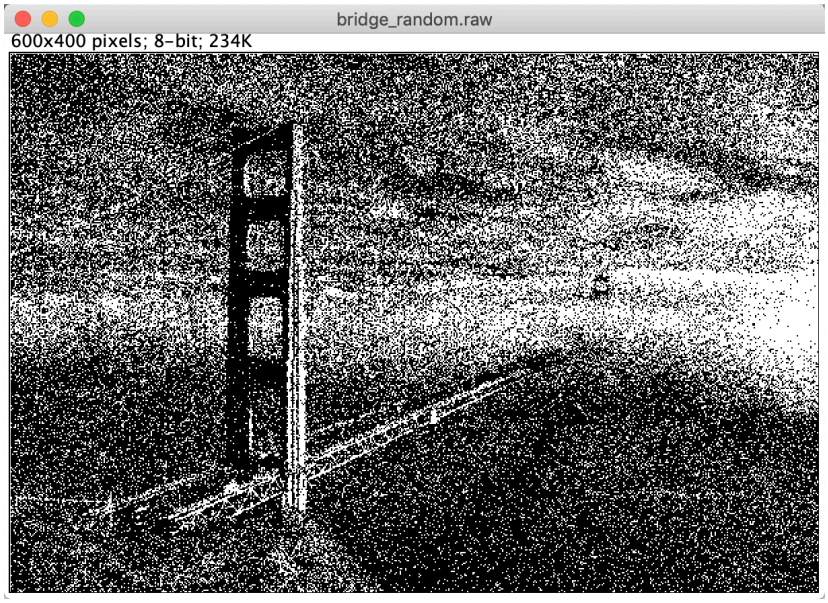
Where N^2 is the pixel quantity within the matrix. Now, we can apply the thresholding equation to get the output image:

$$G(i,j) = \begin{cases} 0 & if\ 0 \le F(i,j) < T(i\ modN, j\ modN) \\ 255 & if\ T(i\ modN, j\ modN) \le F(i,j) < 256 \end{cases}$$

3. Result

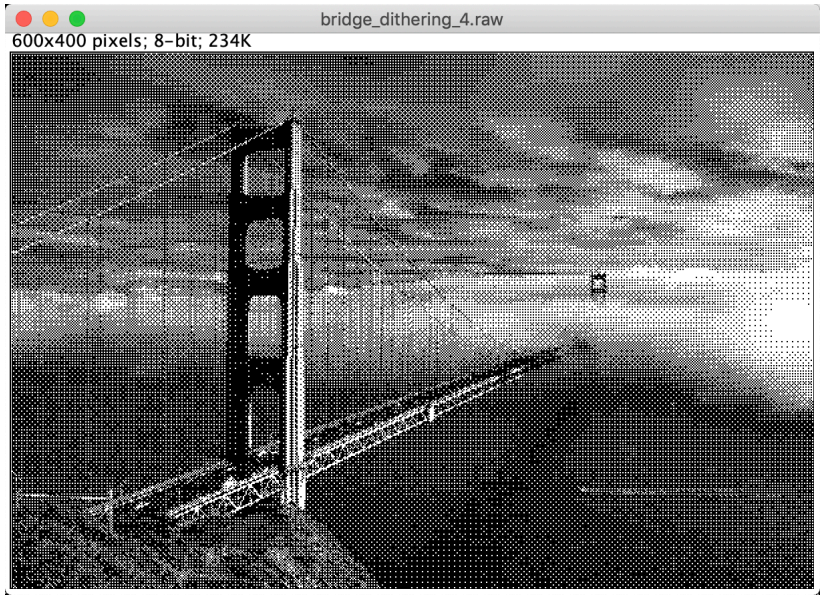
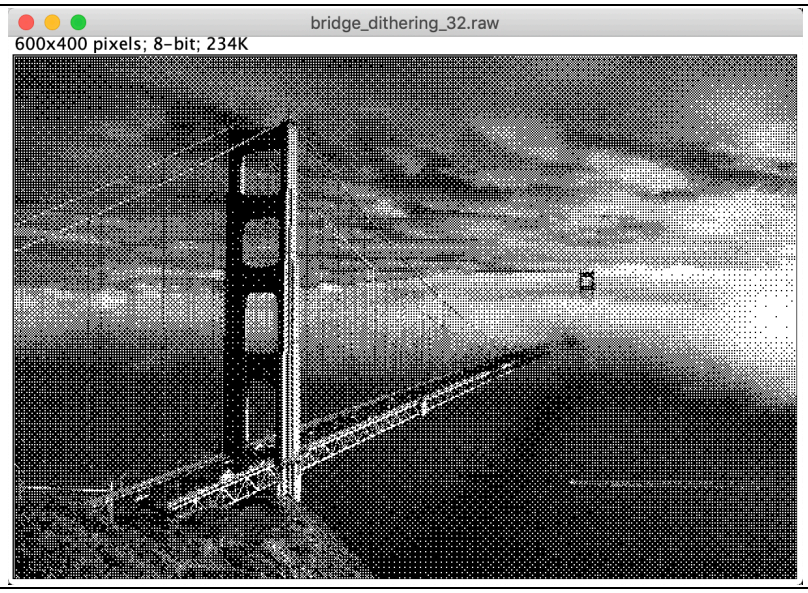
Result images of constant thresholding and random thresholding.

Constant Thresholding	

Random Thresholding	

Result images of dithering matrix thresholding with different matrix size.

Dithering matrix: 2*2	

Dithering matrix: 4*4	
Dithering matrix: 8*8	
Dithering matrix: 32*32	

4. Discussion

As we can see from the above result images, we can rank the result of this 3 halftoning methods from best to worst as: dithering matrix thresholding > random thresholding > constant thresholding.

For constant image thresholding, the "grey level" is badly expressed. The black dots and white dots disconnected and they display a "black area" and "white area" separately. Without "grey level" effect, the result image is pretty bad.

For random thresholding, because of the random threshold, there are usually white dots inside a black area and black dots inside a white area, causing a "grey level" effect. So, in this image, we can see the shape of golden gate bridge and the see. At the same time, because of the randomly set thresholding, we also see a "noisy" effect, i.e., the sharp edges are not clear and small details are ruined.

To make the random thresholding have a better result, we need to set the threshold manually, i.e., we need to make the threshold less "random". In this dithering matrix thresholding, we set the threshold T(x, y) differently for different matrix position. Also, we use "mod" operation here to determine which threshold should be applied to pixel F(i, j). Since the image size is much larger than the matrix size, we can achieve a evenly distributed threshold for pixels at different position. As we can see from the 4 result images, their quality is much better than constant thresholding and random thresholding. Also, when we compare the 4 result images of dithering matrix thresholding, we can conclude that a larger dithering matrix size lead to a better halftoning result. This is because that when we have a large dithering matrix, we will have more threshold value determine a pixel to be white or black and this is will return us more precise decision result; while in a 2*2 dithering matrix, we only 4 threshold value to determine a pixel value to be white or black, which will bring blurring and broad edge effect.

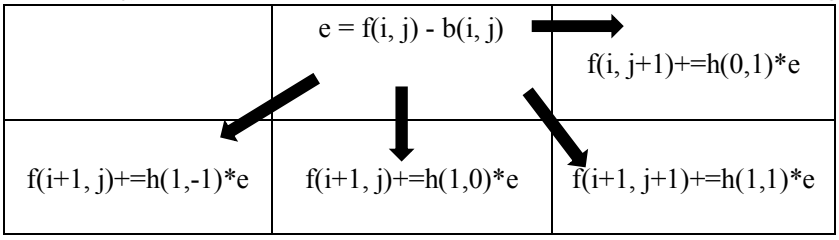
(b) Error Diffusion (Basic: 15%)

1. Motivation

The motivation to develop error diffusion halftoning is to conquer the shortcomings of the above discussed methods. Error diffusion can reach a best halftoning result and is widely used nowadays.

2. Approach

Error diffusion is easy to implement. First, we normalize the original pixel value to 0-1 and then use 0.5 as the threshold to determine the pixel to be white or black, i.e. to be "0" or "255". Here, we denote the original pixel value as F(i, j), the normalized pixel value as f(i, j), binary result pixel value as b(i, j), the threshold as T (T=0.5), error as "e" and the diffusion weight as h(x, y).



As is shown in the above demonstration, after we calculate the error e = f(i, j) - b(i, j), we diffuse this error to its neighbor pixels according to a diffusion matrix, which assigns different weight to neighbor pixels at different position. Note that, here we use the pixel value after diffusion (in other words, the updated pixel value) to determine whether this pixel should be "0" or "255".

One thing we need to pay attention to is that we do the error diffusion in serpentine scanning direction. For example, when we do error diffusion from left to right to the current row, then we need to do error diffusion from right to left to the next row. The reason for this is that we do not want accumulate errors from the beginning until the end in a same direction, or the errors would add up and have bad influence to the next pixel.

Here are 3 kind of commonly used diffusion matrices. See below.

- Floyd-Steinberg

$$\text{FS error diffusion matrix: } \frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

- JJN

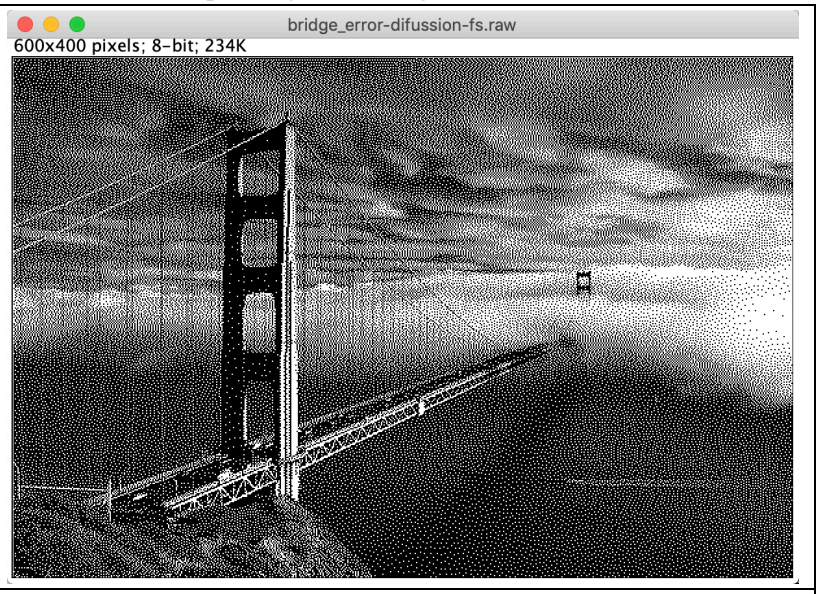
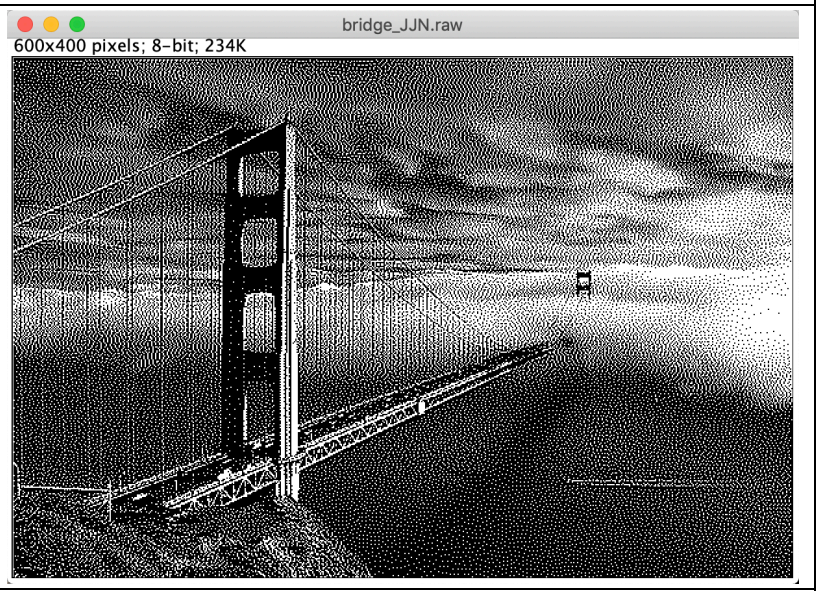
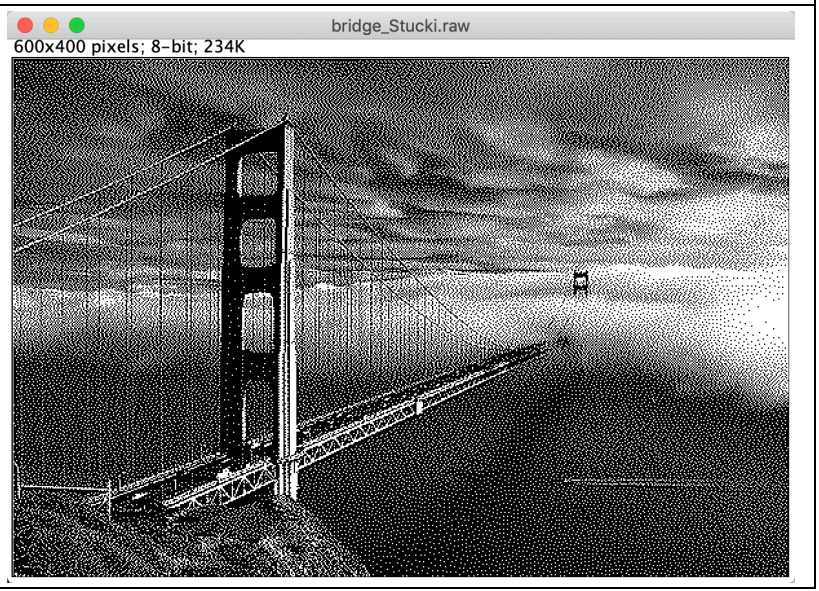
$$\text{JJN error diffusion matrix: } \frac{1}{48}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

- Stucki

$$\text{Stucki error diffusion matrix: } \frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

3. Result

Error diffusion methods and their corresponding result images.

Floyd-Steinberg	
JJN	
Stucki	

4. Discussion

(1) Compare these results with dithering matrix. Which method do you prefer? Why?

I would definitely prefer this error diffusion thresholding method. The error diffusion result images are smoother and more nature than those of dithering matrix. Below shows the result image of JJN (above one) and dithering matrix of size-8 (bottom one). As you can see, the variation of the shadow region down the golden gate bridge in JJN is smoother and more nature. And we can see some disconnection and artifacts from the result image of dithering matrix.

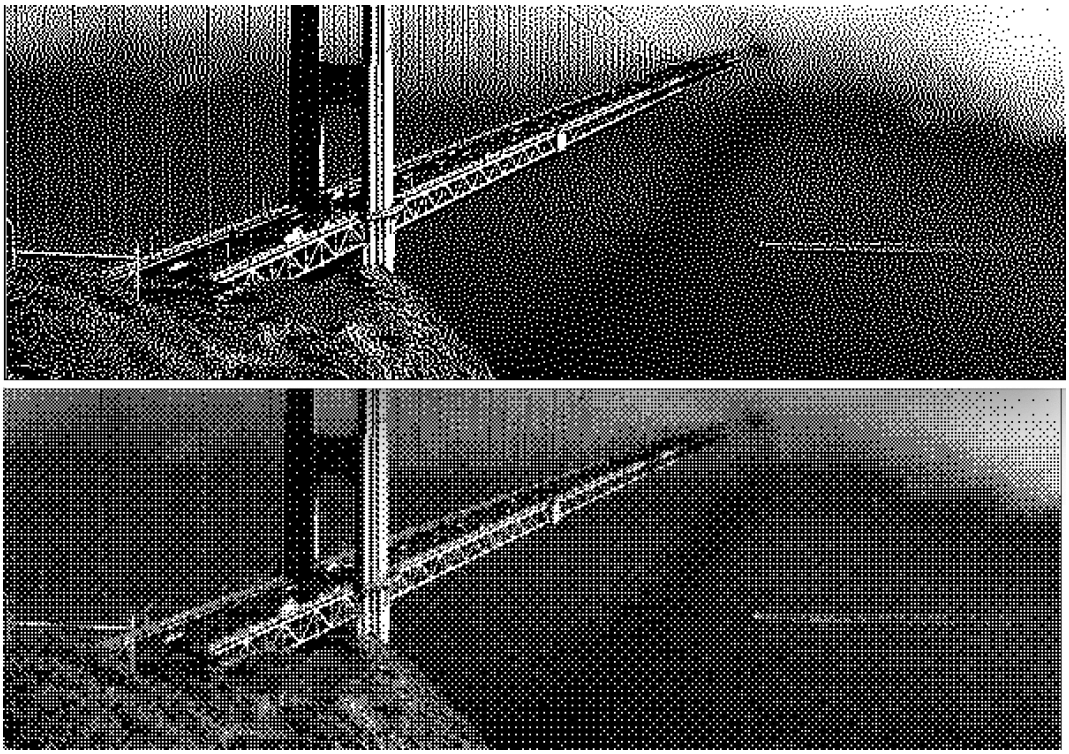


Figure 4 Result images of JJN error diffusion (above) and dithering matrix size-8 (bottom)

(2) Describe your own idea to get better results. There is no need to implement it if you do not have time. However, please explain why your proposed method will lead to better results.

I listed the original bridge image and the result image of JJN error diffusion. Although, the result image quality of JJN error is very good, if we observe closely and look into those details, we can some pepper & salt noise in those all-while and all-black region. For example, bridge pillar region in the original image is all-black, but, the same region in the JJN result image has some salt noise. This is because error diffusion still will accumulate errors from previous pixel and release the error at one time (in this context, make the black dot to be white dot, and cause salt noise).

To solve this problem, after finish error diffusion I propose to search for those salt & pepper noise in the all-white and all-black region, and replace them with their neighbor pixel.

Also, I propose to remove those discontinuous pixels in those continuous regions. For

example, the iron cables are continuous pixel region in the original image, but, the error diffusion breaks this continuity in some extend.

To remove this discontinuity, I propose to re-connect those regions with similar pixel value. Those discontinuous regions should satisfy some properties: in the same row/column; within a region with same pixel value; close to each other, etc. Take the iron cables as an example, my method would remove those "noise" pixels within the iron cable regions and "connect" them together.

In conclusion, my proposed method would improve the result image quality.

(c) Color Half-toning with Error Diffusion (Advanced: 20%)

1. Motivation

For grey images, we want to use only black ink to print it (black dot and white dot) in order to be economic. Similarly, for color images, we also want to use less color inks to print a color image. For grey images, there is only 1 channel. For RGB images, there are 3 channels: R, G and B. So, the algorithm to do color image halftoning is more complex.

2. Approach

(1) Separable Error Diffusion

Before we start, we need to know something about color space. Normally, we display images on our computer screens or TV screens with RGB color space. However, when we print images, due the difference of omitting and reflecting, we use CMY color space. CMY color space is jus the opposite color space of RGB color space (CMY is reflecting color space, RGB is omitting color space).

So, the first step to do color image halftoning is to convert the image from RGB color space to CMY color space. The equations to be used is shown below.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 255 \\ 255 \\ 255 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 255 \\ 255 \\ 255 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

After we convert the image from RGB color to CMY color space, we do error diffusion to C, M, Y channel separately. Here, I implemented Floyd-Steinberg method. To display images on computer, I convert the image from CMY to RGB finally. When we finish all the steps and get the result image, we have 8 kind of pixel value in the result image. So, we only need 3 kind of inks to print a halftoned "RGB" image.

$\mathrm{w} = (0, 0, 0)$ Y = (0, 0, 1), $\mathrm{C} = (0, 1, 0)$, M = (1, 0, 0)
$\mathrm{G} = (0, 1, 1)$ R = (1, 0, 1), $\mathrm{B} = (1, 1, 0)$, K = (1, 1, 1)

(2) MBVQ-based Error Diffusion

Based on human eye visual system property, Shaked et al. [1] proposed Minimum Brightness Variation Criterion (MBVC). Human eyes are actually more sensitive to brightness variation than to color variation. So, when we want to assign the original pixel value with a new value, we need to consider more about brightness variation and make it minimum, so that human eyes would less likely to tell the difference between the original pixel value and the new pixel value. Therefore, the halftoned images based on MBVC is closer (i.e. more similar) to the original images.

Figure 2 shows the MBVQ quadruples and there are 6 quadruples in total, which cover all the color cube. When we assign new pixel values within the quadruples, the brightness variation is minimal, which satisfies MBVC. Next, we assign the current pixel with the value of its closest vertex point value. Then, we use new pixel value and original pixel value to compute error and diffuse it to future pixels. Repeat above steps, we can obtain the final result.

We can conclude the MBVQ-based half-toing method as below:
For each pixel at (i, j) in the image do:

- ✧ Determine MBVQ (RGB(i, j))
- ✧ Find the vertex v ∈MBVQ which is closest to (RGB(i, j) + e(i, j)
- ✧ Compute the quantization error (RGB(i, j) +e(i, j) – v
- ✧ Distribute the error to future pixels

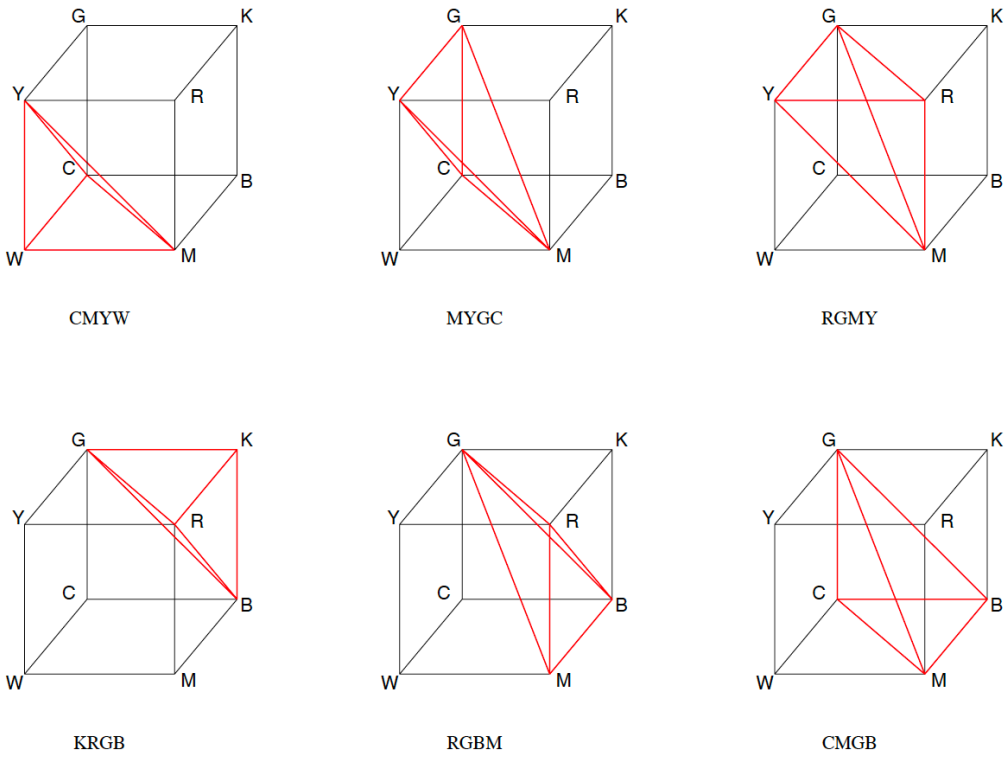


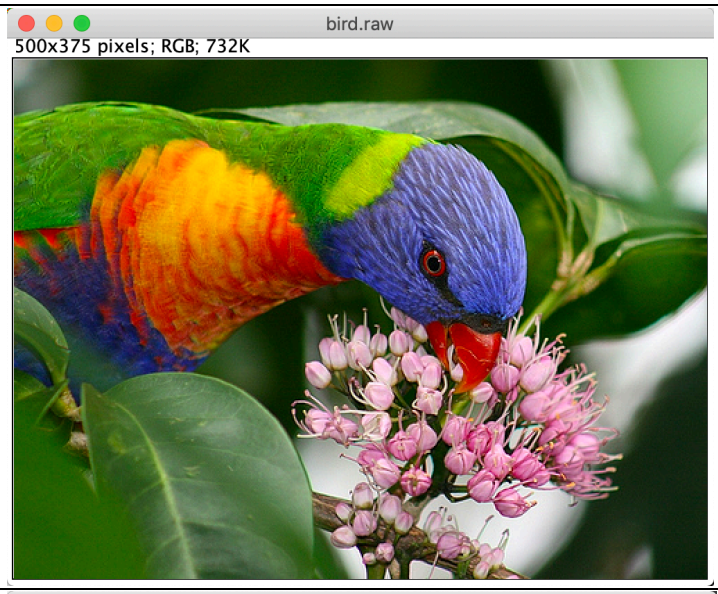
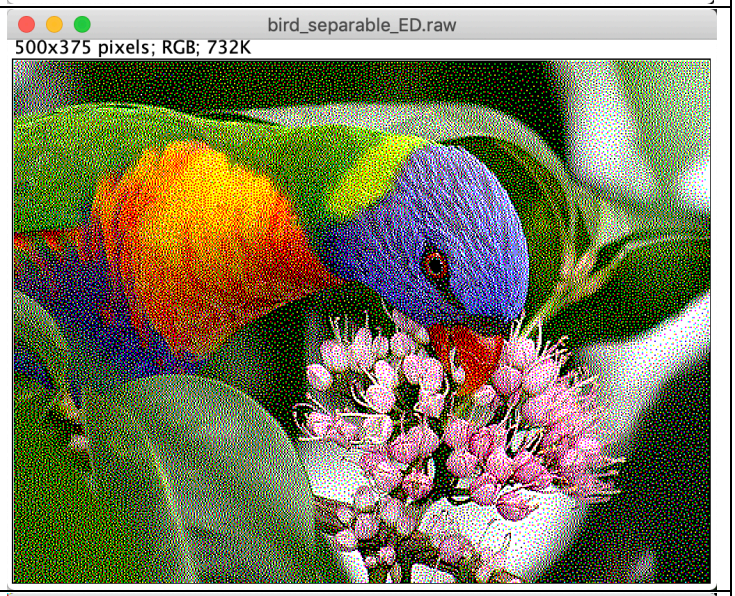
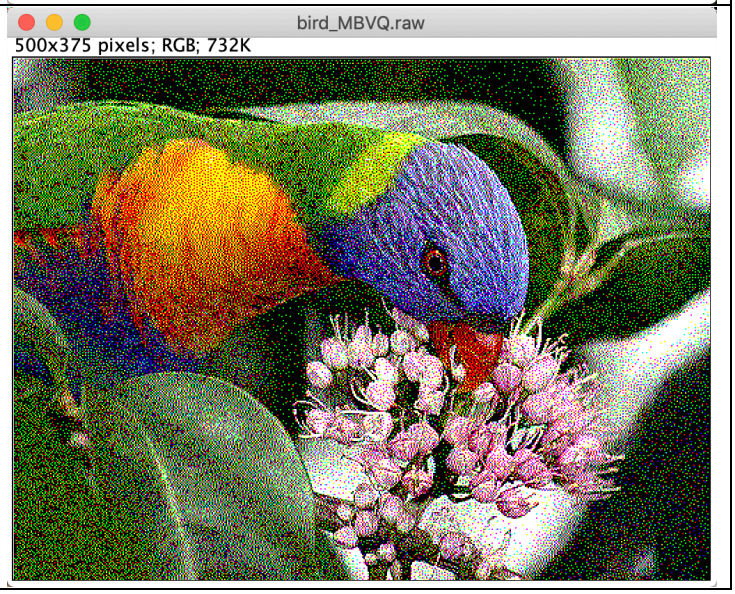
Figure 5 Minimum Brightness Variation Quadruples (MBVQ)

Determine MBVQ

```
pyramid MBVQ(BYTE R, BYTE G, BYTE B)
{
  if((R+G) > 255)
    if((G+B) > 255)
      if((R+G+B) > 510)     return CMYW;
      else                  return MYGC;
    else                    return RGMY;
  else
    if(!((G+B) > 255))
      if(!((R+G+B) > 255)) return KRGB;
      else                  return RGBM;
    else                    return CMGB;
}
```

Figure 6 Determine MBVQ

3. Result

Original Image	
Separable Error Diffusion	
MBVQ-based Error Diffusion	

4. Discussion

(1) Describe the key ideas on which the MBVQ-based Error diffusion method is established and give reasons why this method can overcome the shortcoming of the separable error diffusion method.

In the "Approach" part, I've discussed on which idea MBVQ-based error diffusion is established. Here, I continue to discuss about how MBVQ-based method overcomes the shortcomings of separable error diffusion method.

Since human eyes are more sensitive to brightness variation than to color variation. So, when we make a variation to brightness and color respectively, human eyes are more likely to distinguish the brightness change. Inside the MBVQ quadruples, brightness variation is minimal, and therefore human eyes will consider the result image more alike the original image, i.e. reach a better quality and result compared to separable error diffusion method.

(2) Implement the algorithm using a standard error diffusion process (e.g. the FS error diffusion) and apply it to Fig. 5. Compare the output with that obtained by the separable error diffusion method.

Result images of separable error diffusion and MBVQ-based error diffusion are shown above in the "Result" part.

As we can see from Figure 4, the result image of MBVQ-based method maintains more brightness similarity with regard to the original image. Observe the bird head region carefully, we can tell that the left image expresses the bright blue stripes on the bird head more obviously. And this is because the minimum brightness variation effect. For the right image, i.e. the result image of separable error diffusion, it loses some brightness properties and the image seems like to be too smooth to distinguish colorful patterns.

Figure 7 Bird Head Region (left: MBVQ-based error diffusion; middle: original; right: separable error diffusion)

Reference

[1] D. Shaked, N. Arad, A. Fitzhugh, I. Sobel, "Color Diffusion: Error-Diffusion for Color Halftones", HP Labs Technical Report, HPL-96-128R1, 1996.

[2] P. Dollár and C. L. Zitnick, “Structured forests for fast edge detection,” in Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 1841–1848.