HOMEWORK 2

Jiazhi Li

jiazhil@usc.edu

February 11, 2019

1 Edge Detection

1.1 Abstract and Motivation

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. It includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges[1].

1.2 Approach and Procedures

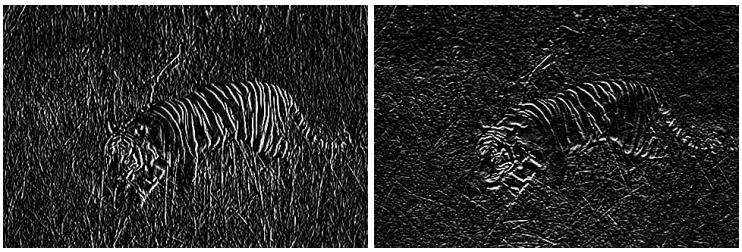
In ths report, there are three kinds of edge detector to introduce:

- Sobel Edge Detector
- Canny Edge Detector
- Structured Edge

1.3 Sobel Edge Detector

1.3.1 Experimental Results

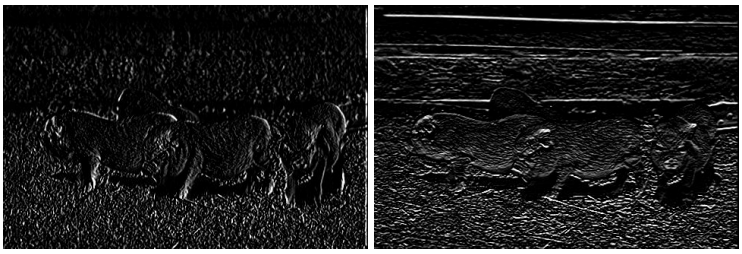
After applying the Sobel edge detection to Tiger, the x-gradient and y-gradient values of tiger are shown in **Figure** 1.



(a) x-gradient value (b) y-gradient value

Figure 1: Tiger after Sobel Edge detector

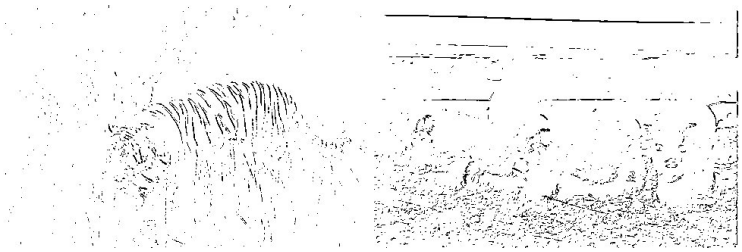
After applying the Sobel edge detection to Pig, the x-gradient and y-gradient values of pig are shown in **Figure** 2.



(a) x-gradient value (b) y-gradient value

Figure 2: Pig after Sobel Edge detector

After that, I combine the x-gradient and y-gradient to the edge map. Having compared the F-measure and visual quality, I choose 98% for tiger and 95% for pig. By this way, the best edge maps are shown in **Figure** 3



(a) Tiger (Threshold is 98%) (b) Pig (Threshold is 95%)

Figure 3: Best edge map

1.3.2 Discussion

In order to tune the thresholds to obtain the best edge map, I calculate the F measure for some thresholds of image, which is shown in Table 1. Since there is only a small difference in threshold, it is difficult to compare the visual quality of each image. Thus, maybe the F measure could be one reasonable indicator to find the best map. By this way, I just present the best map according to the F measure.

Table 1: F measure for different thresholds after Sobel Edge detection

	Threshold	F measure
Tiger	95%	0.3961
	96%	0.4238
	97%	0.4740
	98%	0.5045
	99%	0.5041
Pig	95%	0.3305
	96%	0.2803
	97%	0.2742
	98%	0.2709
	99%	0.2635

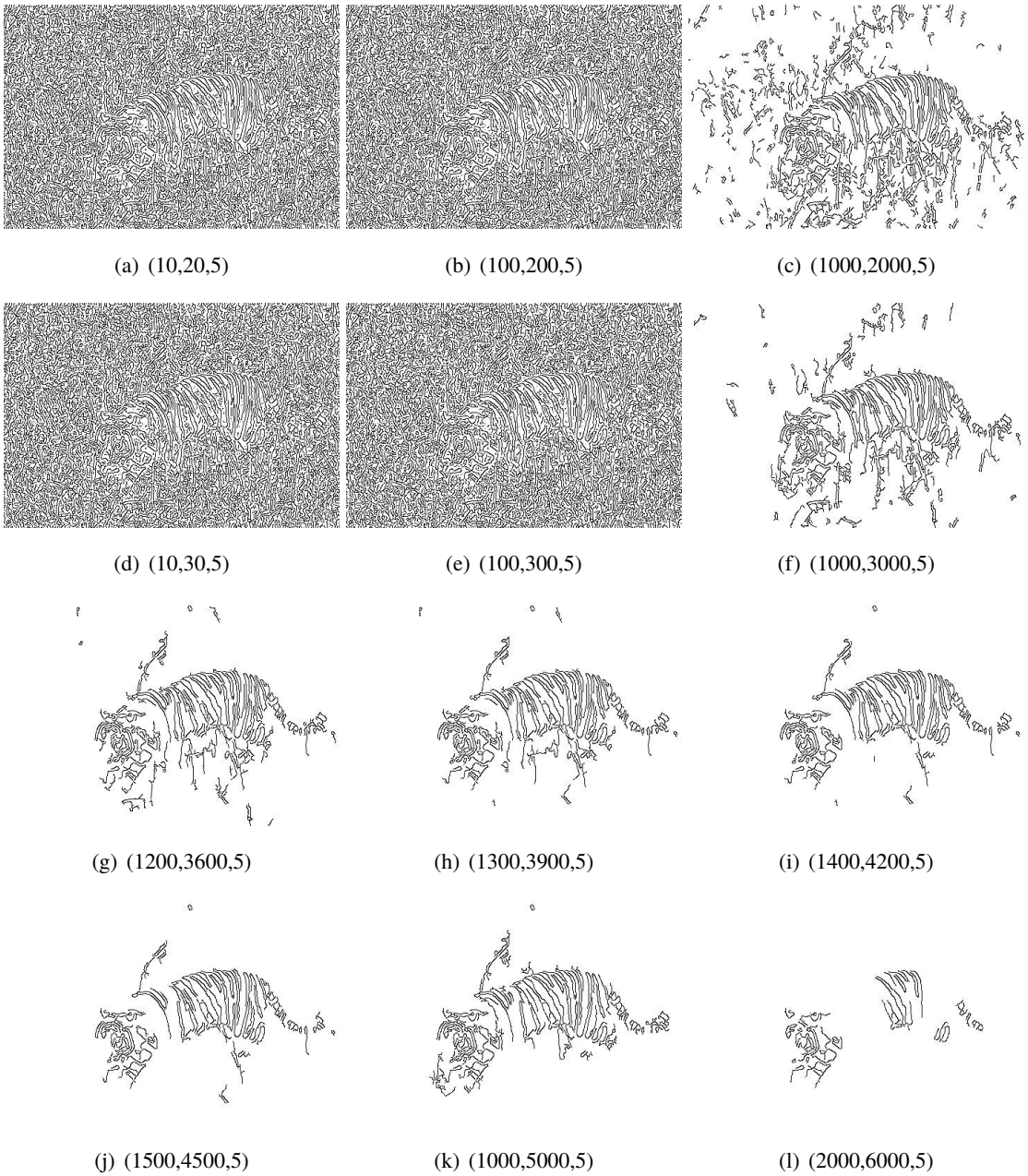
- Computational complexity

Since the Sobel edge detection use the convolution operation, the time complexity of algorithm is $O(n^4)$. To be more specific, it is $O(m * n * k^2)$, where n is size of image and k is the size of filter (for Sobel, k is 3).

1.4 Canny Edge Detector

1.4.1 Experimental Results

The result is shown in **Figure** 4. The notation is represented by (Low threshold, High threshold, kernel size).



(a) (10,20,5) (b) (100,200,5) (c) (1000,2000,5)

(d) (10,30,5) (e) (100,300,5) (f) (1000,3000,5)

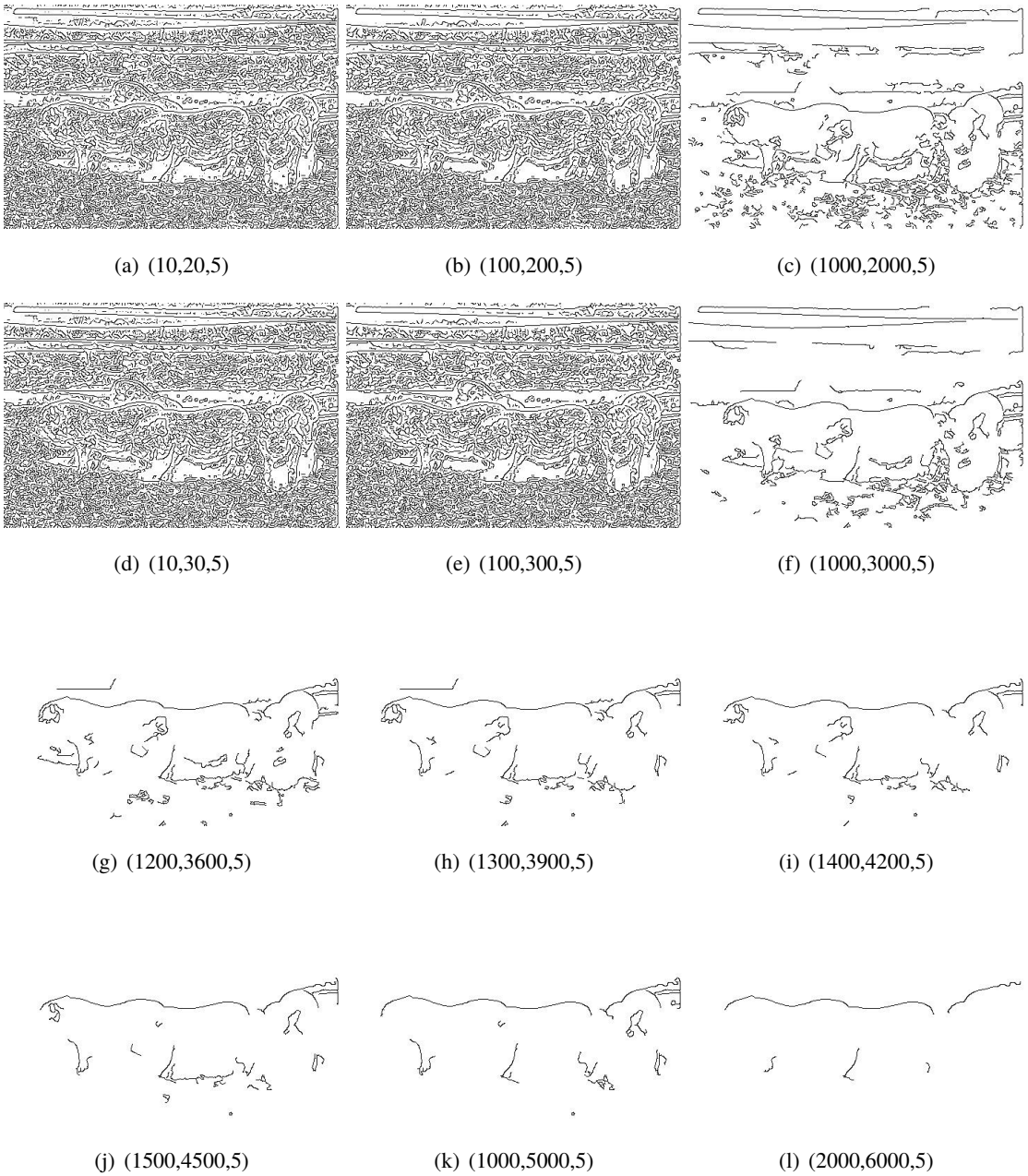
(g) (1200,3600,5) (h) (1300,3900,5) (i) (1400,4200,5)

(j) (1500,4500,5) (k) (1000,5000,5) (l) (2000,6000,5)

Figure 4: Tiger after Canny Edge detector

Having compared the F measure and visual quality of different threshold selections, the best parameter is (1300,3900,5) for tiger.

The result is shown in **Figure** 5. The notation is represented by (Low threshold, High threshold, kernel size).



(a) (10,20,5) (b) (100,200,5) (c) (1000,2000,5)

(d) (10,30,5) (e) (100,300,5) (f) (1000,3000,5)

(g) (1200,3600,5) (h) (1300,3900,5) (i) (1400,4200,5)

(j) (1500,4500,5) (k) (1000,5000,5) (l) (2000,6000,5)

Figure 5: Pig after Canny Edge detector

Having compared the F measure and visual quality of different threshold selection, the best parameter is (1000,3000,5) for pig.

1.4.2 Discussion

- Non-maximum suppression

In order to find the "real" edge and ignore the "semi-real" edge, the non-maximum suppression is introduced. The technique is to define the "semi-real" edge. In canny edge detector, compared with Sobel edge detector,

we not only record the gradient value of each pixel but also record the gradient directions.

After applying gradient calculation, the edge according to the gradient value is obtained. At this time, we need to double check which edge is the real edge by non-maximum suppression. The method is that for each edge pixel, we compare it with the value of the pixel in the positive and negative gradient direction. And, if the value of this edge pixel is the largest one with other neighbor pixel in the gradient direction, this one will be preserved. Otherwise, it will be a "semi-real" edge which is ignored.

I just use the following **Figure** 6 to illustrate the normal case. If the gradient direction is 135 degree for pixel C, we need to check the value of northwest g1 and southeast g2. The same technique can be used for 0 degree, 90 degree and 45 degree.

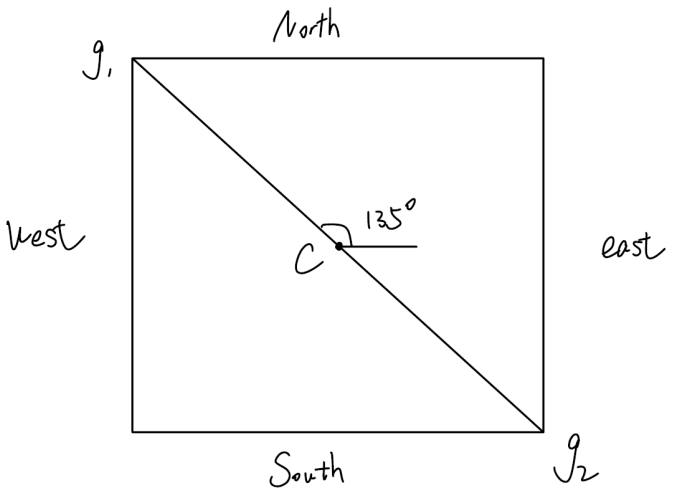


Figure 6: Normal case

For other direction, it may be a little bit complicated. I just present the next figure to illustrate it. If we need to check the values of pixel C1 and pixel C2. How can we get the gradient value of these pixels? This time, we apply the interpolation. The figure is shown in **Figure** 7.

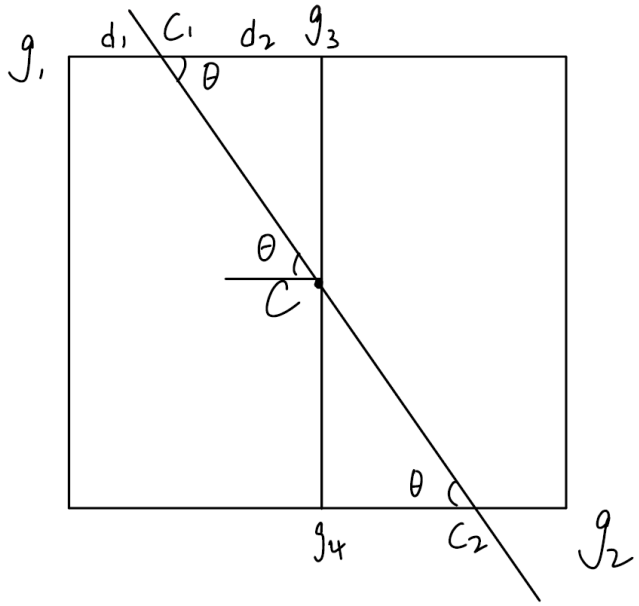


Figure 7: Interpolation case

For example, the gradient values of C_1 can be got by this formula,

$$C_1 = \frac{d_1}{d_1 + d_2} * g_1 + \frac{d_2}{d_1 + d_2} * g_2$$

where d_1 and d_2 represent the distance and g_1 and g_2 represent the gradient values of these two points.

- Double thresholding

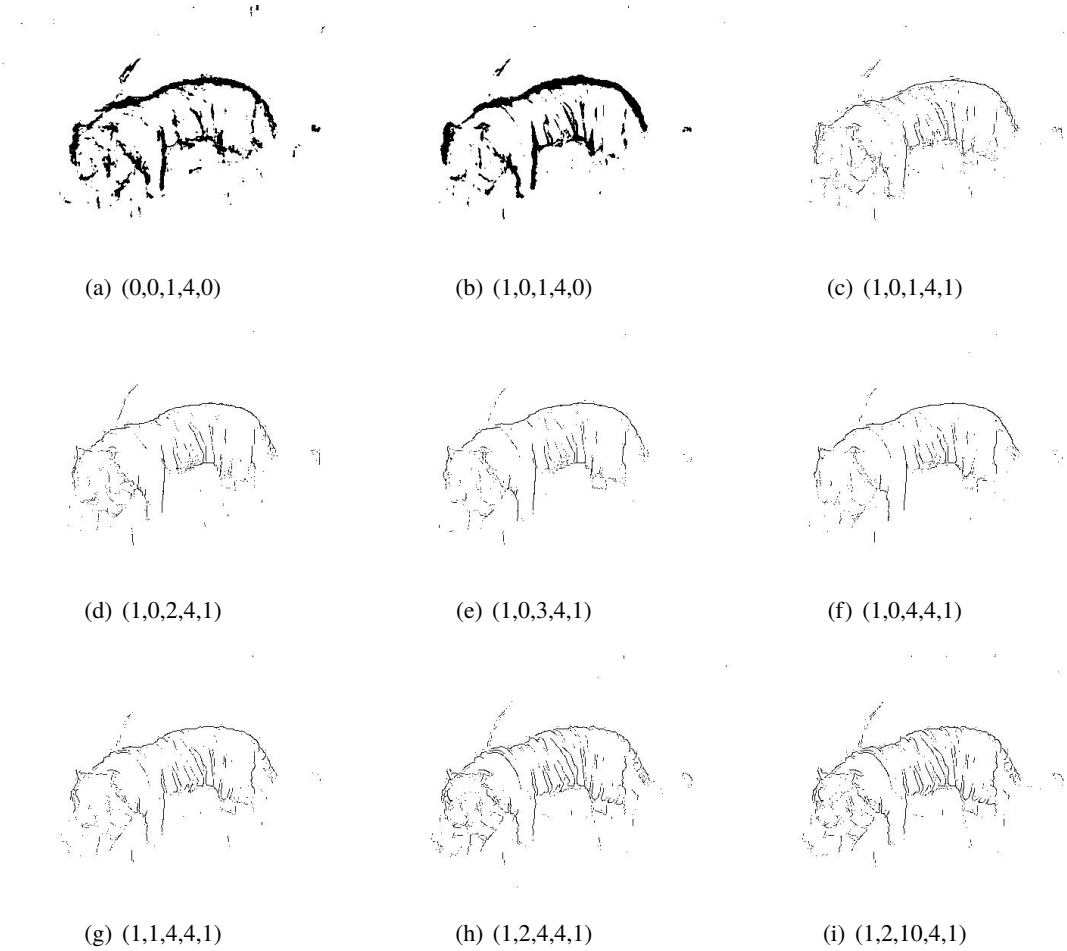
There are two thresholds for canny detector. The higher threshold is used to find the "strong" edge, which means that if a pixel gradient is higher than the upper threshold, the pixel is accepted as an edge. And the lower threshold is used to reject the non-edge pixel, which means that if a pixel gradient is lower than the lower threshold, it will be ignored. What's more, for the pixel gradient whose gradient value is between the two thresholds, it needs to be double checked for the connectivity. If it is connected to a pixel that is above the upper threshold, it will be also accepted as edge.

By this way, it is a little bit complicated than one thresholding so that it will help to find the real edge and increase the connectivity of edge.

1.5 Structured Edge

1.5.1 Experimental Results

The result is shown in **Figure** 8. The notation is represented by (multiscale,sharpen,nTreesEval,nThreads,nms).



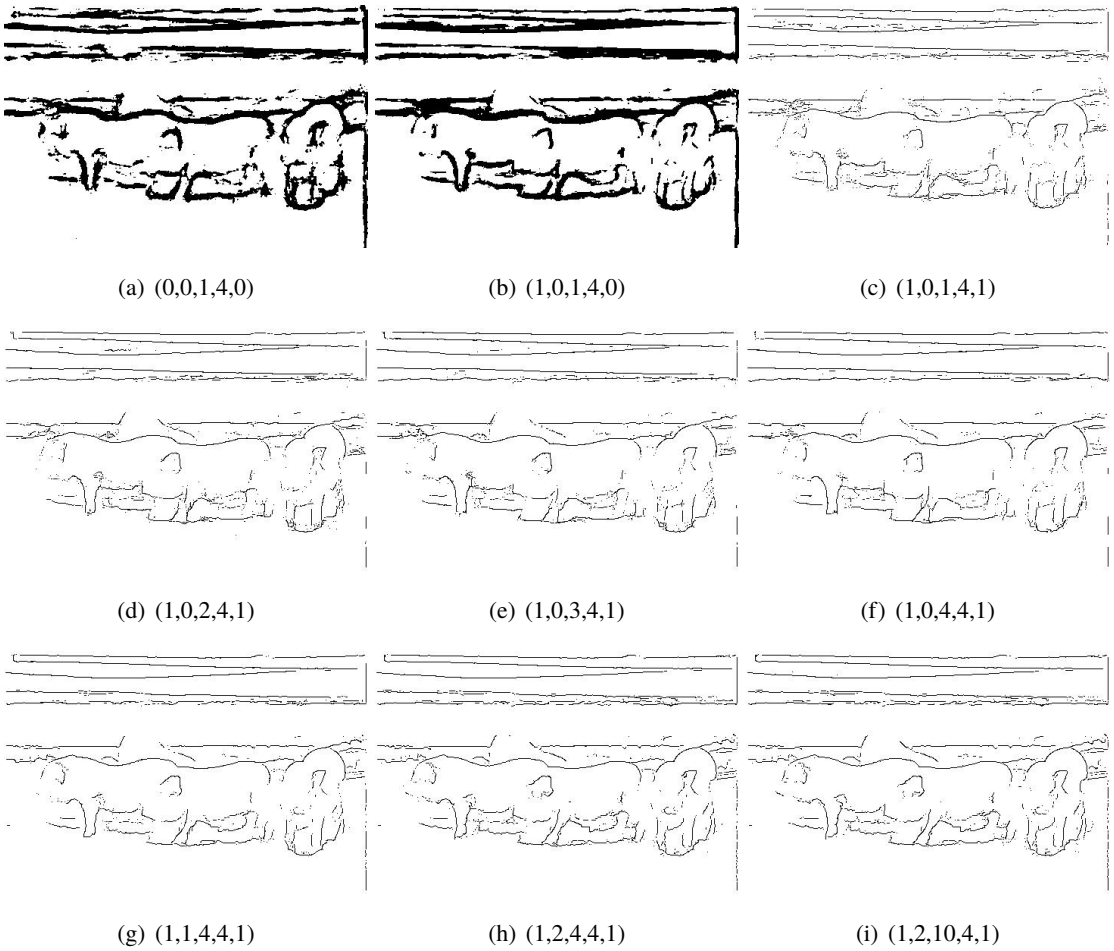
(a) (0,0,1,4,0) (b) (1,0,1,4,0) (c) (1,0,1,4,1)

(d) (1,0,2,4,1) (e) (1,0,3,4,1) (f) (1,0,4,4,1)

(g) (1,1,4,4,1) (h) (1,2,4,4,1) (i) (1,2,10,4,1)

Figure 8: Tiger after Structure Edge detector

The result is shown in **Figure** 9. The notation is represented by (multiscale,sharpen,nTreesEval,nThreads,nms).



(a) (0,0,1,4,0) (b) (1,0,1,4,0) (c) (1,0,1,4,1)

(d) (1,0,2,4,1) (e) (1,0,3,4,1) (f) (1,0,4,4,1)

(g) (1,1,4,4,1) (h) (1,2,4,4,1) (i) (1,2,10,4,1)

Figure 9: Pig after Structure Edge detector

1.5.2 Discussion

- Structure edge detection algorithm [2]

In this part, I will not introduce the details about the structure of decision tree and random forest, which will be presented in next section, though it is very important part of the structured edge detection. But for any machine learning algorithms, data set, feature and label are necessary components. Thus, in this part, I will try my best to introduce this part.

Firstly, data set and feature. The whole flow diagram is shown in **Figure** 10.

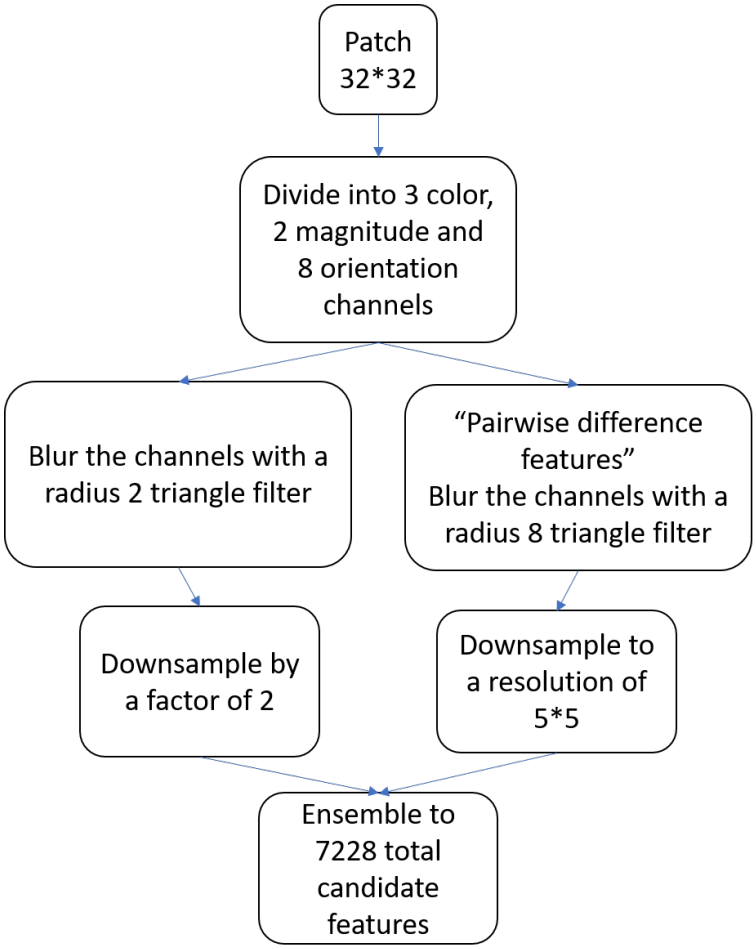


Figure 10: Finding features

All in all, for structured edge detection, we use 32*32 image patch as data set. And according to the description in the flow diagram, we get 7228 total candidate features.

Secondly, labels. For edge detection, the labels are 16*16 segmentation masks and they define a intermediate mapping to map masks to be a long binary vector that encodes whether every pair of pixels belong to the same or different segments. But for edge detection there are 32640 unique pixel pairs in a 16*16 segmentation mask, which is very expensive to compute and store. Thus, in order to decrease the complexity of algorithm, they use a new notation to record the same informaton. Instead of recording the specific binary values of pixels, they just encodes the equality for every unique pair of different indices. By this way, the complexity decreases from 2^{256} to 32640.

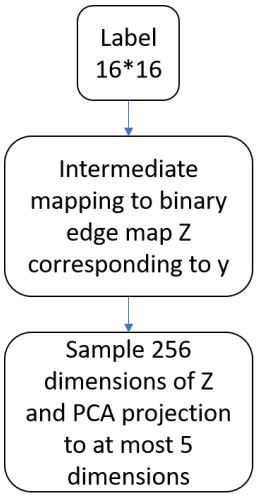


Figure 11: Preprocessing for labels

Having defined the features and obtained useful label, the decision tree can be construct.

- Decision tree construction [3]

Decision tree is a very popular machine learning algorithm for classification problems. In decision tree, just like a tree, there are three kinds of node such as root node, inner node and leaf node. Each inner node represents one kind of characteristics test. For example, when the whole sample meets the first node, the sample will be divided into several classes according to their conditions in this characteristic. After that, the several new samples will meet the different characteristics node one by one until it comes to the final leaf node. By this way, leaf node represents the decision result. The structure of decision tree is shown in the **Figure** 12.

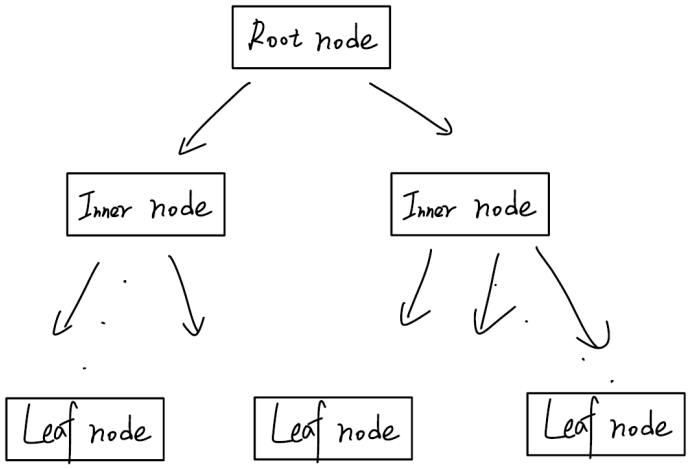


Figure 12: Decision tree

Thus, the key of decision tree constructing is to find the best characteristic combination for the sample. There are two important indicators for finding the best characteristic, information entropy and information gain.

Information entropy:

$$Ent(D) = -\sum_{k=1}^{y} p_k \log_2 p_k$$

Information gain:

$$Gain(D, a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v)$$

The strategy is to use some regular patterns of sample to decrease the uncertainty of the sample. The higher information entropy is, the higher uncertainty of sample is. And, the information gain means that by using this characteristic, the decreasing degree of certainty of sample. By this way, at very beginning, for the choice of first characteristic, we need to find the characteristic which can decrease the uncertainty of sample mostly. Also, for the following characteristics, we need to follow this rule to find the characteristic which is the best characteristic to decrease the uncertainty at that time.

After that, the basic decision tree is construct by the regular pattern of training set. In order to avoid overfitting, we need to do some extra operations for this decision tree. The operations are prepruning and postpruning. For both two kinds of pruning, we tend to consider the generalization ability of this decision tree. Prepruning is that in the generation process of decision tree, if the division of current node cannot increase the generalization ability, we stop dividing and let the current node be leaf node. Postpruning is that after the whole decision tree is construct, we evaluate the all inner node bottom-up. If replacing the subtree to be a leaf node can increase the generalization ability, then we replace the subtree to be leaf node. By this way, the decision tree is construct.

- Random forest [4]

Before we talk about random forest, the Bagging need to be introduced first. Bagging is a kind of ensemble learning. Based on bootstrap sampling, given a sample set containing m samples, we randomly put a sample into sample set with replacement in order that next time this sample can be still selected again. By this way, after that, we can get T sample sets containing m samples. For each sample set, followed the procedure of constructing decision tree, one decision tree is trained. When we ensemble predictable results, majority voting is used for classification and averaging is used for regression problem. Random forest is based on Bagging ensemble learning which selects decision tree as basic learning method. In random forest, for each node of sub decision tree, we don't select the best characteristic directly anymore. Instead, we randomly select a subset of the feature for this node firstly and find the best characteristic in this subset of the feature. Thus, for random forest classifier, the "random" mean that a random subset of the features is selected and the "forest" means that it is an ensemble learning method of many decision trees as sub learning method.

- Parameter chosen [2]

`Multiscale`

When the parameter of multiscale detection is set as 1, the program will run structured edge detector on the origin, half, and double resolution version of and average the result of the three edge maps after resizing to the original image dimensions.

`Non-maximal suppression`

Sometimes, the predicted edge maps from the structured edge detector are diffused. In the last section, I have already introduced the effect of non-maximal suppression which can effectively detect the edge peaks and get the strong and isolated edges. When this parameter is set as 1, the program will activate the part of non-maximal suppression.

`Sharpen`

Although non-maximal suppression can detect edge peaks, for weak edges that receive few votes, no clear peak may emerge. Specifically, each overlapping prediction may be shifted by a few pixels from the true edge location. By this way, the sharpen is used to align overlapping masks to the underlying image data implicitly so that we can get sharper, better localized edge responses. The program supports sharpening of 2 pixels at most. Thus, we can set this parameter as 0, 1 and 2.

`nTreesEval`

This is about the number of decision trees to consist the random forest. Having tried some different number of decision tree, we can see some improvement in edge detection. However, the larger the number is, the longer time it takes. By this way, I will find an enough number of decision tree. It is really a trade-off between performance and time.

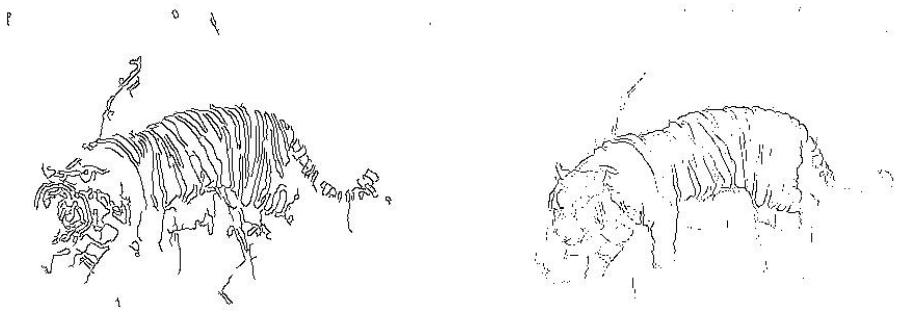
`nThreads`

This is an important parameter for time expense. It represents the number of computer threads for compute and evaluate. This time, I just give a fixed number for this parameter.

Having observed the result of different parameter, the (1,2,10,4,1) is the best edge detected parameter for both tiger and pig. We use 10 decision trees to consist random forest, which is calculated with efficiency corresponding to performance improvement. Also, multiscale, sharpen and non-maximal suppression are quite useful to improve the performance so that I use the version of SE+MS+SH.

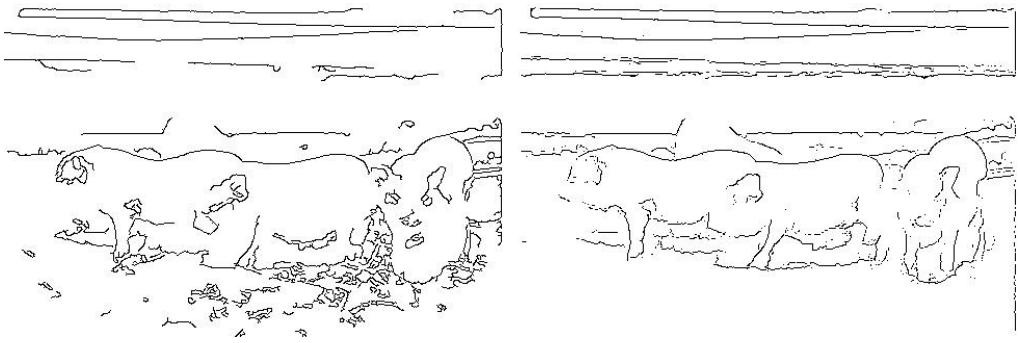
- Comparison

The comparison of tiger for Canny detector and the SE detector are shown in **Figure** 13. The comparison of pig for Canny detector and the SE detector are shown in **Figure** 14.



(a) Best Canny result (b) Best Structured edge result

Figure 13: Comparison for tiger



(a) Best Canny result (b) Best Structured edge result

Figure 14: Comparison for pig

From the figure, structured edge detector has better performance in edge detection and it can detect approximate shape of tiger and pig. What's more, Structured edge detection has cleaner result. In contrast, the Canny edge detector is more sensitive to noise and have some mis judgement in background. However, after binarizing the voting result of SE, the connectivity of edge is declined, but Canny have better connectivity of edge by double thresholding.

1.6 Performance Evaluation

1.6.1 Approach

In this part for the calculation of F measure, instead of writing code by myself according to the pseudo code in **Figure** 15, I just use the function of released MATLAB toolbox(https://github.com/pdollar/edges). And, the function is shown in **Figure** 16.

```
for each GT g:
    for each threshold t:
        compute Precision(g,t) and Recall(g,t)

for each GT g:
    compute mean_Precision_over_thresholds(g), mean_Recall_over_thresholds(g) #"mean precision, recall for each ground truth"
compute mean_F = 2 * mean(mean_Precision_over_thresholds) * mean(mean_Recall_over_thresholds) /
                     (mean(mean_Precision_over_thresholds) + mean(mean_Recall_over_thresholds))

for each thredhold t:
    compute mean_Precision_over_GTs(t), mean_Recall_over_GTs(t)
    compute F(t) = 2 * mean_Precision_over_GTs(t) * mean_Recall_over_GTs(t) / (mean_Precision_over_GTs(t) + mean_Recall_over_GTs(t))
plot F #x axis would be threshold values
find the best F score max(F)
```

Figure 15: Pseudo code

```
%Image path
E = 'G:\569\Homework\HW2\Image\Enhanced\12441.jpg';
%Ground truth path
[thrs, cntR, sumR, cntP, sumP, V] = edgesEvalImg( E, "Tiger.mat" );
%compute recall precision F-measure
[R, P, F] = computeRPF(cntR, sumR, cntP, sumP);
%find the best recall precision F-measure
[bstR, bstP, bstF, bstT] = findBestRPF(thrs, R, P);
```

Figure 16: MATLAB function

1.6.2 Experimental Results

Table 2: Best Sobel detector result(Threshold is 98%) of Tiger for different ground truth

	Precision	Recall	F measure
Tiger1	0.0661	0.9452	0.1235
Tiger2	0.0708	0.9551	0.1319
Tiger3	0.0784	0.9415	0.1448
Tiger4	0.3130	0.9524	0.4712
Tiger5	0.0809	0.8374	0.1476
Final	0.3482	0.9144	0.5045

Table 3: Best Sobel detector result(Threshold is 95%) of Pig for different ground truth

	Precision	Recall	F measure
Pig1	0.0761	0.8492	0.1397
Pig2	0.0793	0.8181	0.1445
Pig3	0.1152	0.7423	0.1994
Pig4	0.1463	0.7503	0.2449
Pig5	0.1130	0.7461	0.1963
Final	0.2101	0.7784	0.3305

Table 4: Best Canny detector result(1300,3900,5) of Tiger for different ground truth

	Precision	Recall	F measure
Tiger1	0.0753	0.9313	0.1393
Tiger2	0.0793	0.9257	0.1462
Tiger3	0.0893	0.9276	0.1629
Tiger4	0.3696	0.9740	0.5358
Tiger5	0.0877	0.7857	0.1578
Final	0.4184	0.9300	0.5772

Table 5: Best Canny detector result(1000,3000,5) of Pig for different ground truth

Pig1	0.0888	0.8159	0.1601
Pig2	0.0947	0.8046	0.1695
Pig3	0.1403	0.7443	0.2360
Pig4	0.1804	0.7645	0.2919
Pig5	0.1354	0.7392	0.2289
Final	0.2538	0.8573	0.3916

Table 6: Best Structured edge detector result(1,2,10,4,1) of Tiger for different ground truth

	Precision	Recall	F measure
Tiger1	0.1509	0.3446	0.2099
Tiger2	0.1298	0.6067	0.2139
Tiger3	0.1843	0.3533	0.2422
Tiger4	0.4544	0.8409	0.5900
Tiger5	0.1309	0.4698	0.2048
Final	0.4976	0.8248	0.6207

Table 7: Best Structured edge detector result(1,2,10,4,1) of Pig for different ground truth

Pig1	0.2010	0.6399	0.3060
Pig2	0.2193	0.6455	0.3273
Pig3	0.3037	0.5838	0.3996
Pig4	0.4426	0.6475	0.5258
Pig5	0.3120	0.5876	0.4076
Final	0.5112	0.6310	0.5649

1.6.3 Discussion

- Comparison for the performance of different edge detector

`Sobel`

The Sobel operator is relatively easy to implement in space, which not only produces better edge detection, but also introduces local averaging and is less affected by noise. If a larger neighborhood is used, the noise immunity will be better, but the amount of calculation will also increase, and the resulting edges will be coarser. The Sobel operator is a more common edge detection algorithm when the accuracy requirement is not very high. The advantage of Sobel edge detector is its simple calculation and fast speed. However, since the simplicity of calculation direction, it is difficult for Sobel to handle a little bit complex texture and edge. And the edge judgement method according to threshold directly is too simple without reasonable explanation so that it will create some noise and false detection.

`Canny`

The Canny method is not susceptible to noise and can detect true weak edges. The advantage is that the strong edge and the weak edge are detected separately using two different thresholds, and the weak edge is included in the output image when the weak edge is connected to the strong edge.But it is easy to misjudge noise as a boundary. In order to get better edge detection results, it usually needs to use a larger filtering scale, so that it is easy to lose some details. The double threshold of the Canny operator needs to be artificially selected and cannot be adaptive.

`Structured edge`

From the F measure and visual quality, we can find that the performance of SE is much better than Sobel and Canny. After applying sharpen and multiscale version, the improvement is more obvious. Although it is a little bit time consuming, corresponding to the excellent improvement and robustness to noise, the expense can be accepted. What's more, the fast and direct inference procedure is ideal for applications requiring computational efficiency. One disadvantage is that there are many parameters to set, which is not quite adaptive. And sometimes for some image, the effect of sharpen version is not obvious. Also, with increasing in the scope of training data set, the time spent increases quickly.

- High F measure

Tiger is easier to a get high F measure because there are too many decorative patterns in the body of tiger and there is nothing in the body of pig, which make there are more edges to detect for tiger than for pig. By this way, there are more pixels labeled as true positive for tiger than pig, which will increase the recall and precision. Also, the detected image of tiger will get the high F measure.

- Rationale behind the F measure definition

Although we want to make both recall and precision as high as possible, in some extreme case, the recall is very different from precision. For example, the recall is significantly higher than precision, or vice versa. This time, evaluating the result by recall or precision separately is not rigorous. In order to reconcile this contradiction, F measure is designed to consider recall and precision comprehensively.

We cannot get a high F measure if precision is significantly higher than recall and opposite is also not true because the F measure is more sensitive to the lower value of precision and recall instead of higher value between them. By formula,

$$\frac{2}{F} = \frac{1}{R} + \frac{1}{P}$$

For example, we select P as 1 and R as 0.01. Then, the reciprocal of R is very large, which make the right-hand side large. However, the F-measure will be very small. By this way, it is not possible to get a high F measure if precision is significantly higher than recall, or vice versa.

We know the definition of F measure is,

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

Then if the sum of recall and precision is a constant, the formula comes to,

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

This time, R and P just like the height and width of rectangle. We know if we want to make the area maximum, the shape of rectangle should be square where height is equal to width. By this way, recall is equal to precision.

2 Digital Half-toning

2.1 Abstract and Motivation

Halftone is the reprographic technique that simulates continuous tone imagery through the use of dots, varying either in size or in spacing, thus generating a gradient-like effect. "Halftone" can also be used to refer specifically to the image that is produced by this process.

Where continuous tone imagery contains an infinite range of colors or greys, the halftone process reduces visual reproductions to an image that is printed with only one color of ink, in dots of differing size (pulse-width modulation) or spacing (frequency modulation). This reproduction relies on a basic optical illusion: the tiny halftone dots are blended into smooth tones by the human eye [5].

2.2 Approach and Procedures

There are several methods for halftoning:

- Random dithering
- Ordered dithering
- Error diffusion

2.3 Dithering

2.3.1 Experimental Results

First, the result for Random thresholding is shown in **Figure** 17.

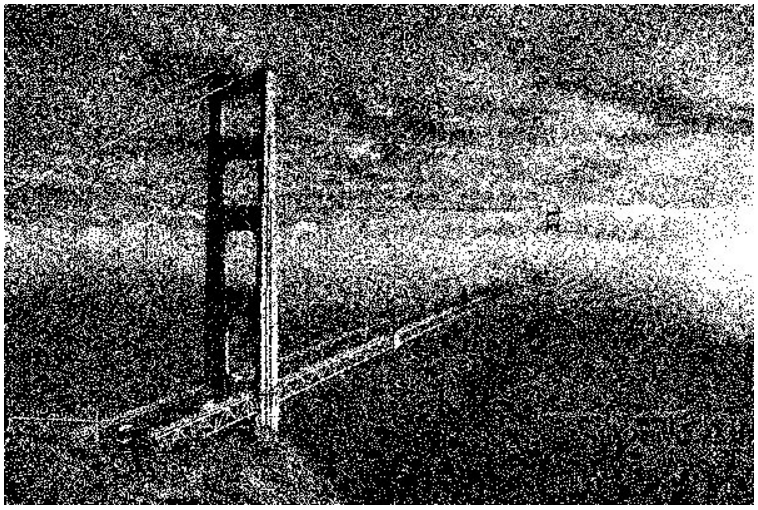
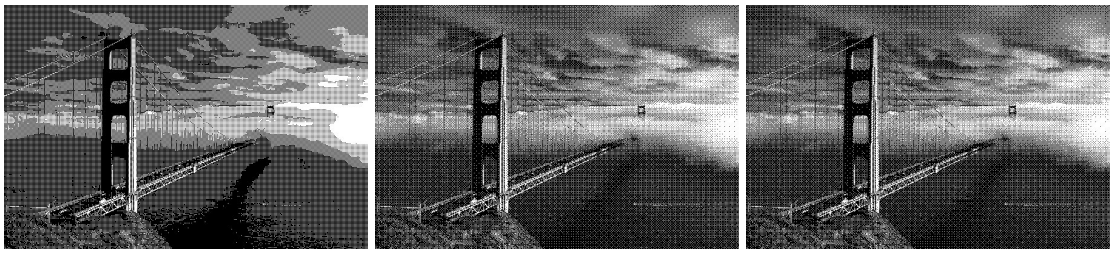


Figure 17: Bridge by Random thresholding

Second, the result for Dithering matrix is shown in **Figure** 18.



(a) I2 (b) I8 (c) I32

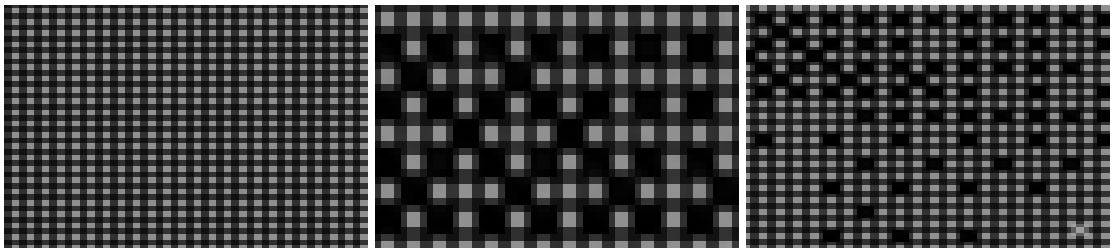
Figure 18: Bridge by Dithering matrix

2.3.2 Discussion

The random dither matrix is generated completely randomly, so the image quality after halftone is often unsatisfactory with lots of random noises, and is basically no longer used in practice.

Although the ordered dithering algorithm is relatively simple and has good halftone image quality, it also has a fatal disadvantage that it contains obvious periodic artificial texture. Even if the dither matrix design is perfect, the halftone image of the output still has flaws, and its halftone image quality is not as good as the halftone image obtained by the error diffusion algorithm. Having observed the zooming result in details, we can easily find the periodic artificial texture for different size of dithering matrix.

The details for periodic artificial texture is shown in **Figure** 19.



(a) I2 (b) I8 (c) I32

Figure 19: Periodic artificial texture for different dithering matrix

- Computational complexity

Since the Random thresholding is just that comparison with random value in range [0,255], the time complexity of algorithm is $O(n^2)$. To be more specific, it is $O(m * n)$, where n is size of image.

For dithering matrix, the time complexity of matrix constructing is just a constant. By this way, the time complexity of algorithm is $O(n^2)$. To be more specific, it is $O(m * n)$, where n is size of image.

2.4 Error Diffusion

2.4.1 Experimental Results

The result for error diffusion is shown in **Figure** 20.

(a) Floyd-Steinberg (b) JJN (c) Stucki

Figure 20: Bridge by error diffusion

2.4.2 Discussion

The output produced by applying JJN and Stucki error diffusion are much better than that of Floyd Steinberg because there are the bigger error diffusion matrices for JJN and Stucki. By this way, the error from the anchor is diffused broader than 3*3 error diffusion matrix for Floyd. Also, there are only four pixels distributed for Floyd but for JJN and Stucki, there are 12 pixels distributed, which make the output smoother. For most images, there are minimal difference between the output of Stucki and JJN algorithms. However, there are some differences in computation complexity. Since in Stucki error diffusion matrix, there are some multiple kernel values such as 1/42, 2/42, 4/42 and 8/42, which make it a little bit faster while calculating the value of

kernel. Thus, compared with JJN algorithms, Stucki is other used because of its slight speed increasing.
The main problem for dithering algorithms is that there are some grainy looking images in uniform values area caused by the distracting bank of speckles. To avoid this problem on areas with uniform values, we can diffuse only a fraction of the error instead of the full amount. By only propagating part of the error, speckling is reduced.
All in all, Floyd–Steinberg (FS) dithering only diffuses the error to neighboring pixels. This results in very fine-grained dithering. Minimized average error dithering by Jarvis, Judice, and Ninke diffuses the error also to pixels one step further away. The dithering is coarser but has fewer visual artifacts. However, it is slower than Floyd–Steinberg dithering, because it distributes errors among 12 nearby pixels instead of 4 nearby pixels for Floyd–Steinberg. Stucki dithering is slightly faster and its output tends to be clean and sharp. By this way, I think Stucki is much better.

- Improvment

In order to get the half-toned image without "stream-like" random pattern, we can design a new matrix with larger size. Maybe it can be 7*7 and the error will be distributed to 17 nearby pixels. Also, we can change the scanning order such that Hilbert curve in **Figure** 21.

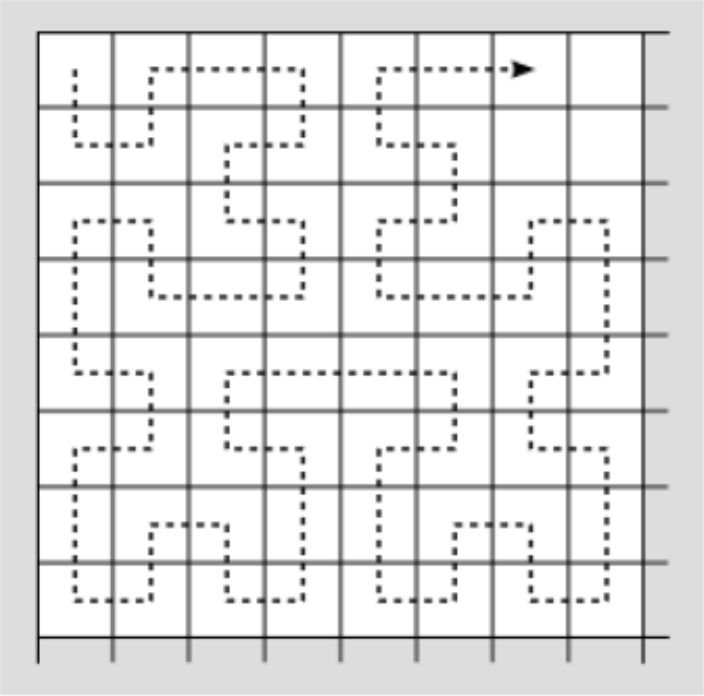


Figure 21: Hilbert curve

- Computational complexity

Since Error Diffusion use the convolution operation, the time complexity of algorithm is $O(n^4)$. To be more specific, it is $O(m * n * k^2)$, where n is size of image and k is the size of filter (for Floyd, k is 3 and for JJN and Stucki, k is 5). And, since in Stucki error diffusion matrix, there are some multiple kernel values such as 1/42,

2/42, 4/42 and 8/42, which make it a little bit faster while calculating the value of kernel. Thus, compared with JJN algorithms, Stucki is other used because of its slight speed increasing.

2.5 Color Halftoning with Error Diffusion

2.5.1 Experimental Results

The result for separable error diffusion is shown in **Figure** 22.

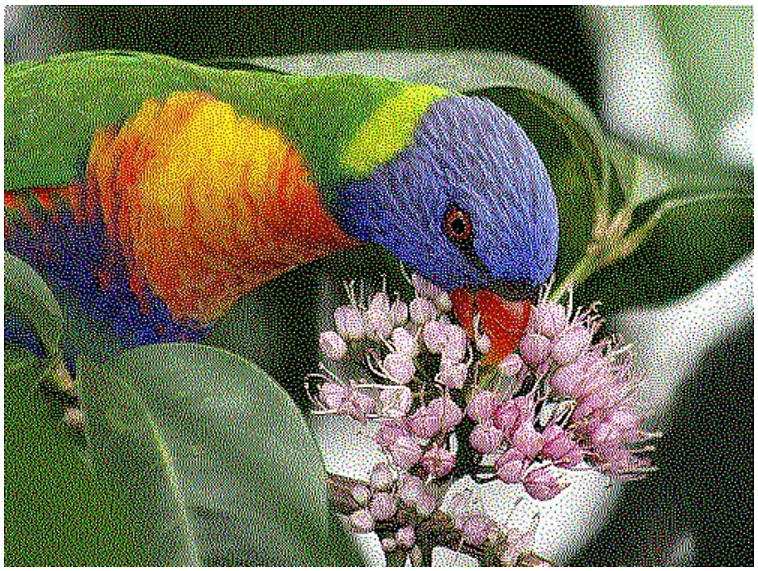


Figure 22: Bird by Separable Error Diffusion

The result for MBVQ is shown in **Figure** 23.

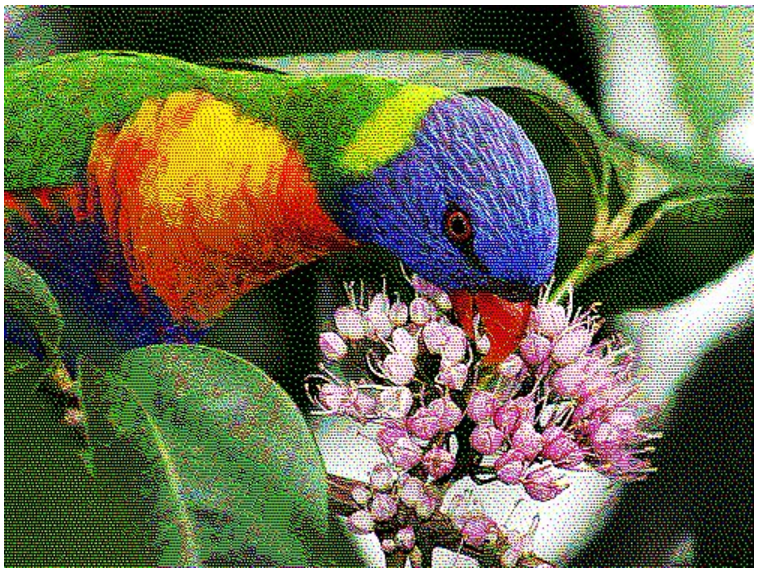


Figure 23: Bird by MBVQ-based Error diffusion

2.5.2 Discussion

- Main shortcoming of Separable Error Diffusion

The results obtained from separate error diffusion have some obvious artifacts such as color distortion. And the halftoning image is a little bit tendency to white than origin image, which is the appearance of color distortion.

Based on some characteristics of the human visual system, the rule of color imaging is a little bit complex. As we know, digital halftone takes the use of low-pass-filter characteristic of the human eye. When we view the image at a certain distance, the human eye regards a spatially close part of the image. With this characteristic, the local average gray scale of the halftone image observed by the human eye approximates the local average gray value of the original image, thereby forming a continuous tone effect as a whole.

When presented with high frequency patterns, the human visual system "applies" a low-pass filter and perceives only their average. The human visual system is more sensitive to changes in brightness than to changes in the chrominance, which filter much lower frequencies. And for the color image, the high frequency component is image changing instead of the representation of the specific color. However, the separable error diffusion doesn't think about this important characteristic. It just applies the error diffusion for CMYB channel separately, which will cause the low color reduction compared with origin image.

Having observed the detail in **Figure** 24, we can easily find the color distortion, which is the main shortcoming of this approach. Also, there is the variation in the brightness of the dots.

(a) Distortion 1 (b) Distortion 2

Figure 24: Details for bird after separable error diffusion

- Key ideas of MBVQ-based Error diffusion method [6]

MBVQ is based on minimal brightness variation criterion. To reduce halftone noise, select from all halftone sets by which the desired color may be rendered, the one whose brightness variation is minimal. Just like the characteristics of human visual system, the high frequency component is more important. MBVC considers colors in a high frequency pattern. By this way, chrominance differences between participating colors play the role.

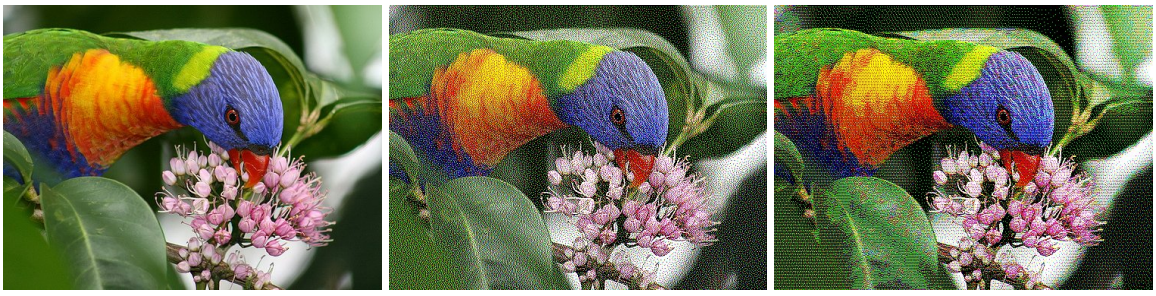
According to the MBVC, the colors have to be rendered using the halftone set whose brightness variation is minimal. By Ink-Relocation-type, neighboring halftone couples are transformed to minimize their brightness variation, while preserving their average color. Thus, the whole color space is divided into six halftone

quadruples. Through the different cases of six quadruples, given an RGB triplet, one may compute its MBVQ by point location. This is the first improvement of MBVQ-based Error diffusion method.

The second key of this method is color diffusion. The algorithm looks for the closest vertex in the MBVQ of the color, as opposed to the closest of the eight vertices of the cube in separable Error Diffusion. This is the biggest difference between separable error diffusion method and MBVQ error diffusion method, which help it to overcome the shortcoming of separable error diffusion. When applying separable error diffusion each component of the RGB value is compared to the threshold value, 127, and a tessellation of dimension 3 with respect to the eight vertices is formed. However, the closer look reveals that this is not necessary. The same does not hold when a tessellation relative to a proper subset of the eight vertices is called for: When restricted to the RGB cube itself, any given quadruple gives rise to the same tessellation regardless of the norm used, however outside the cube these tessellations may differ. Thus, the rule of diffusion is not "closest to vertices" anymore. Instead, determining the closest vertex to a given point entails traversing a decision tree of depth for different six quadruples.

- Comparison

Origin image and result image are shown in **Figure** 25.

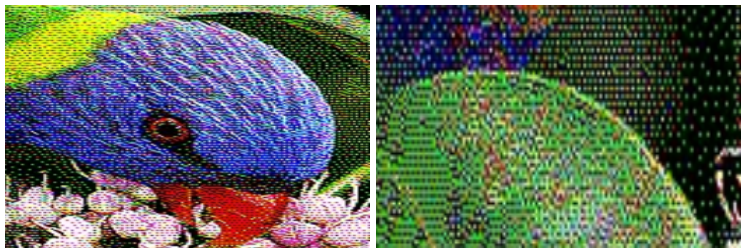


(a) Origin image (b) Separable error diffusion (c) MBVQ

Figure 25: Comparison

Compared with the result of separable error diffusion, application of MBVQ reduces brightness variation and rendering is still done with all eight halftones.

Having observed the details in **Figure** 26, we can easily see that with MBVQ the feather of bird and leaves have better visual quality. The color looks like more abundant by MBVQ. Although both of them are obtained by halftoning, the result from MBVQ is closer to origin image with more brilliance.



(a) Improvement 1 (b) Improvement 2

Figure 26: Details for bird after MBVQ

- Computational complexity

Since Error Diffusion use the convolution operation, the time complexity of algorithm is $O(n^4)$. To be more specific, it is $O(4 * m * n * k^2)$, where n is size of image and k is the size of filter (for Floyd, k is 3).

Reference

[1] Lindeberg T. Edge Detection and Ridge Detection with Automatic Scale Selection[J]. International Journal of Computer Vision, 1998, 30(2):117-156.

[2] Piotr Dollár, Zitnick C L . Structured Forests for Fast Edge Detection[C]// 2013 IEEE International Conference on Computer Vision. IEEE, 2014.

[3] Quinlan J R. Induction on decision tree[J]. Machine Learning, 1986, 1(1):81-106.

[4] Vladimir Svetnik, Andy Liaw, Christopher Tong, et al. Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling[J]. Journal of Chemical Information and Computer Sciences, 2003, 43(6):1947.

[5] Zhi Z, Arce G R, Crescenzo G D. Halftone visual cryptography[J]. IEEE Transactions on Image Processing A Publication of the IEEE Signal Processing Society, 2006, 15(8):2441-53.

[6] Shaked D. Color diffusion: error diffusion for color halftones[J]. Proceedings of SPIE - The International Society for Optical Engineering, 1999, 3648(96).

Appendix

In this part, I want to introduce the structure of program in details. There are three classes for the homework, Imagedata, kernel and algorithm.

Class Imagedata

In class Imagedata, there are six kinds of function such as basic function, Image operation, edge detection, dithering, error diffusion and color halftoning with error diffusion.

Basic function

```
//basic function
Imagedata(int h, int w, int p); //constructor
~Imagedata(); //destructor
void set_doubledata(); //initialize double data for shot noise
void initialize(int d); //initialize all data with value d
void read(unsigned char* buff); //read data from buff
void load(string path); //load data from path
unsigned char* write(); //write data to buff
void save(string path); //save data to path
Imagedata get_RGB(int i); //get single channel data
int convert(int h, int w, int p);
```

Image operation

```
//image operation
Imagedata Boundaryextension(int ex); //Boundary extension
Imagedata Crop(int N); //crop operation
Imagedata Convolution(Kernel k); //convolution operation with kernel
Imagedata Boundaryextension_double(int ex); //Boundary extension for double data
Imagedata DN_Gaussion_double(int N, double sigma); // Gaussian filter for double data
Imagedata Convolution_double(Kernel k); // convolution for double data
void double2usignedc(); //change double data to unsigned char data
Imagedata Crop_double(int N); // crop operation for double data
Imagedata Save_double(string path); // save double data to path
double* write_double(); // write double data to buff
```

Edge detection

```
//Edge detection
Imagedata RGB2GRAY(); //convert RGB image to gray-level image
Imagedata Sobel_X(); //Get the x-gradient values
Imagedata Sobel_Y(); //Get the y-gradient values
Imagedata Sobel_merge(Imagedata Sobel_Y); //Combine x and y gradient into whole gradient image
Imagedata Tune(int threshold); //Get the edge map according to threshold
```

Dithering

```
//Digital halftoning
//Dithering
Imagedata Random_threholding(); //Random thresholding
Imagedata Dithering(int index); //Generate dithering matrix
```

Error diffusion

```
//Error diffusion
Imagedata Error_diffusion(Kernel k); //Error diffusion with kernel k
Imagedata Floyd(); //Floyd-Steinberg's error diffusion
Imagedata JJN(); //JJN error diffusion
Imagedata Stucki(); //Stucki error diffusion
```

Color halftoning with error diffusion

```
//Color Halftoning with error diffusion
Imagedata RGB2CMY(); //convert RGB image to CMY image
Imagedata CMY2RGB(); //convert CMY image to RGB image
Imagedata Color_Floyd(); //Separable error diffusion
void Color_merge(Imagedata R_component, Imagedata G_component, Imagedata B_component); //Combine three separable
Imagedata Color_MBVQ(); //MBVQ-based error diffusion

void set_color_data(int i, int j, vector<unsigned char> c); //set color data
vector<unsigned char> get_color_data(int i, int j); //get color data
```

Class Kernel

```
Kernel(int h, int w); //constructor
~Kernel(); //destructor
void set_data(double d, int i, int j); //set elements(i,j) as value d
double sum(); //get sum
vector<vector<double>> get_wholedata(); //return a two dimension vector as kernel
double get_data(int i, int j); //get the value of element(i,j)
int get_height();
int get_width();
Kernel EQ(); //equalization


Kernel Dithering_matric(int index); //Generate dithering matrix
void resize(int h, int w); //Resize the kernel
```

Class Algorithm

```
int RangedRandDemo(int range_min, int range_max, int n); //Generate random number in range[min,max]
int find_closest(int a); //Find the close binary value to 0 or 255
int correction(double a); //Correct the gray-level in range[0,255]

string getNearestVertex(vector<unsigned char> old); //Find the nearest vertex according to MBVQ
vector<unsigned char> string2int_MBVQ(string vertex); //Convert the string to integer

cv::Mat Canny_edge_detection(cv::Mat src, int lowthreshold, int highthreshold, int kernel_size); //Apply Canny_edge_detection
```