EE 569 Homework 2

Zheng Wen

Zwen1423@usc.edu

Problem 1: Edge Detection

(a) Sobel Edge Detector

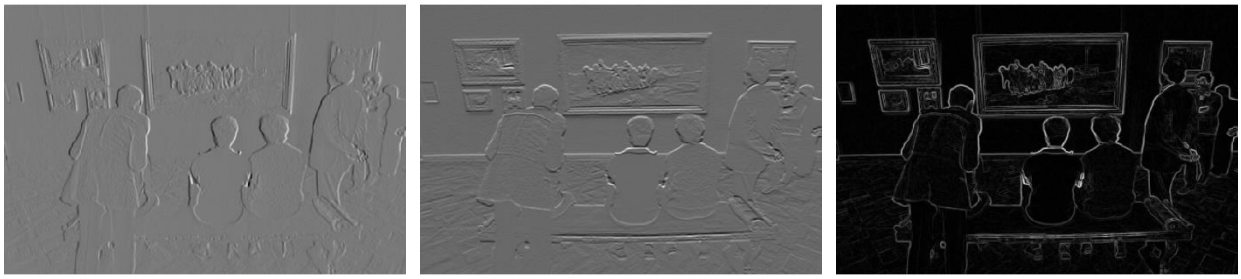
Sobel edge detection is an easy edge detection method, this method obtain edge by detect the change of pixels, calculate the gradient of pixels. to get the x-gradient map and y-gradient map, the RGB image is transferred into grayscale image by

$$\mathrm{Y} = 0.2989 * \mathrm{R} + 0.5870 * \mathrm{G} + 0.1140 * \mathrm{B}$$

Then, the grayscale image is convolved with x-direction Sobel kernel and y-direction Sobel kernel to get x-gradient map and y-gradient map respectively, and the gradient magnitude map could be obtained using $G = \sqrt{G_x^2 + G_y^2}$. To visualize gradient map and gradient magnitude map, the pixels are normalized to 0-255.

(a) Gradient on x (b) Gradient on y (c)Gradient map

Fig. 1 Sobel Edge Detector on Dogs.raw



(a) Gradient on x (b) Gradient on y (c)Gradient map

Fig. 2 Sobel Edge Detector on Gallery.raw

To get the final edge map, all pixels on Gradient map are proceeded by [3]. Threshold is a certain value below which there are a certain percentage of pixels of the original gradient map is set to zero, the rest of them are set to 255. Through experiment, to get the best visualization of edge map, the threshold of Dogs.raw is set to 96%, i.e. if the pixels are arranged in a sequence by pixel values from small to large, and if the pixel lies in the first (1-96%), it will be set to 255, otherwise

it will be set to 0, and that of Gallery.raw is set to 95%, for the items there could be shown clearly with few detail, noise and texture.

$$Final(i,j) = \begin{cases} 255 & if\ G(i,j) \le Threshold \\ 0 & if\ G(i,j) > Threshold \end{cases}$$

(a) Final Edge Map of Dogs.raw
Threshold = 96%

(b) Final Edge Map of Gallery.raw
Threshold = 95%

Fig. 3 Final Edge Map

(b) Canny Edge Detector

Canny edge detector is an advanced geometrical edge detector, to optimize the performance of edge detection, non-maximum suppression and double thresholds are used in this algorithm.
This part is implement with the help of edge() function in MATLAB.
(1) Ideally, edge map should contain only the thin lines standing for edges. However, the images proceeded by detection kernel are always thick, i.e. edges are described by several lines repeatedly, instead of a single line after preliminary processing. Non-maximum suppression is a method which could make the edge thinner by cutting off non-maximum pixels on edges, leaving the point with maximum value of pixels to stand for edges. In this algorithm, the point is remained if its pixel value is larger than the neighboring points along gradient direction and negative gradient direction of this point, or the point will be set to 0. Gradient direction could be decided by $tan\angle G = \frac{\partial G_x}{\partial G_y}$.

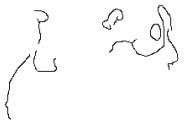
(2) Because images after proceeding processing contain something that does not belong to edges, such as noise and texture. The points with pixel value above high threshold are regarded as strong edges, those below low threshold are not regarded as edges, those between high threshold and low threshold are regarded as weak edges. Strong edges will be shown on final edge map for certain, and set to 255, those under low threshold will be set to 0, the weak edges are used to connect strong edges, to keep connection of edges, they will be shown on final edge map if there are strong edges in their neighborhood, otherwise they will be set to 0.

(3) High threshold could decide the boundary of foreground and background, if the high threshold is too high, the edge map will lose a lot of information of object in the image, if it is too

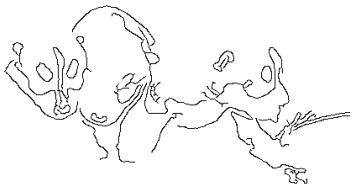
low, too much noise or texture in the background will be regarded as strong edges. Low threshold could decide the continuity of edges, if it is too low, the edge map will include unnecessary noise and textures, if it is too high, the edge map will not be continuous enough to show the edges. Some results are shown in Fig. 4 and Fig. 5.

(a) High threshold is too high, containing too much texture.
High=0.3, Low=0.15

(b) High threshold and low threshold are proper.
High=0.5, Low=0.15



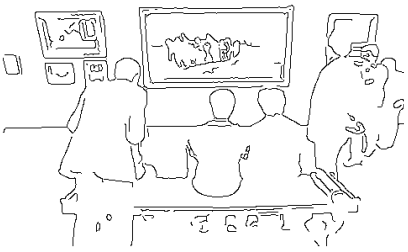
(c) High threshold is too high, dogs couldn't be recognized.
High=0.8, Low=0.35



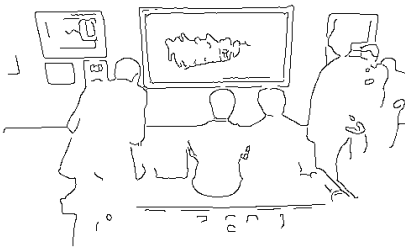
(d) High threshold is proper while low threshold too low, containing some texture information.
High=0.6, Low=0.1

(e) High threshold is proper while low threshold too high, some edges are not continuous.
High=0.6, Low=0.4

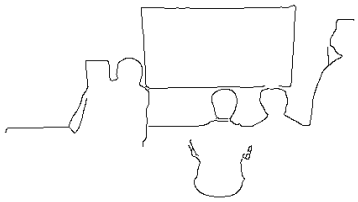
Fig. 4 Canny on Dogs.jpg



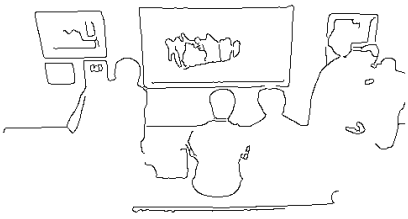
(a)
High=0.2, Low=0.15



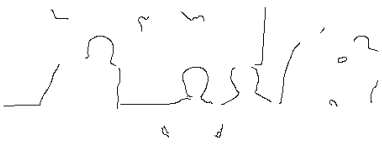
(b) Proper one,
High=0.3, Low=0.2



(c)
High=0.7, Low=0.15



(d)
High=0.5, Low=0.1

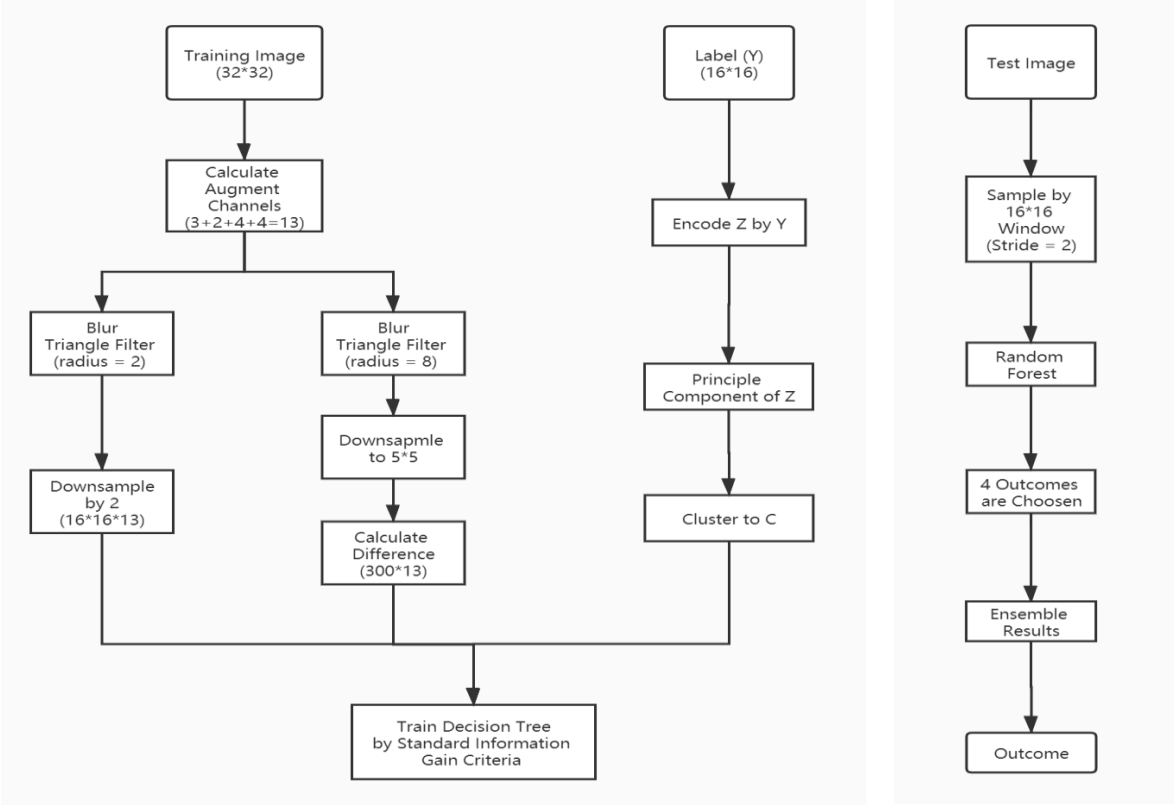


(e)
High=0.6, Low=0.4

Fig. 4 Canny on Gallery.jpg

(c) Structured Edge

Structure edge is a machine learning algorithm to detect edge. In the beginning, a random forest containing several structure decision trees is trained by augmented feature from training set, using the trained random forest, every pixel in test set is decided several times to get the probability whether it is an edge or not, techniques are illustrated below.



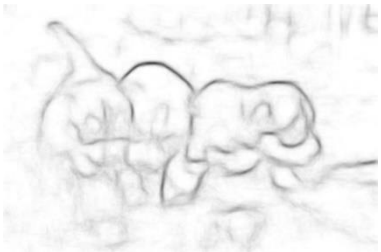
(a) Train Process (b) Test Process

Fig. 5 Flow Chat of SE Detection

(1) To train a decision tree, the input image is cut to 32*32 patches, each patch is augmented by calculating 13 channels in total, 3 LUV channels, 2 magnitude channels and 8 orientation channels. These channels are blurred by triangle filter with radius 2 and then downsampled by 2, 32*32*13/4 = 3328 features are obtained. The channels are also blurred by triangle filter with radius 5 and downsampled to 5*5 patches, the difference between pixels are calculated, another $13 * C_2^{5*5} = 13 * 300 = 3900$ features are calculated. In total, 7228 features in training set X are used to train. The structured labels Y are 16*16 segmentation masks or edge maps. A mapping from Y to Z is used to make estimating dissimilarity of Y by calculating Euclidean distance in Z is possible, which also avoid the problem that Y lies in high dimension by mapping Y with similar Z into the same class C. Z is encoded by checking whether y(j_1) and y(j_2) are in the same segment. Then a subset of Z is chosen by dimensional reduction method like Principle Component Analysis,

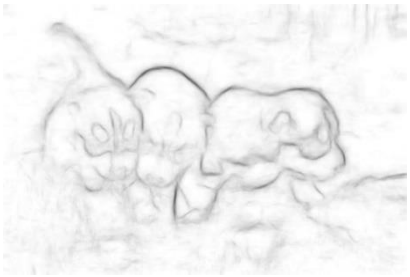
which also contribute to randomness in learning process, and then mapping to class set C by top $\log_2 k$ PCA dimensions, where k=2 by experiment. With preprocessing to X and Y, the decision tree could be train by standard information gain criteria based on Shannon entropy or Gini impurity. 2T decision trees are trained so that the random forest is set up, where T trees are regarded as an alternating set of T trees and are used to decorrelated predictions at each adjacent location. To combine the output of all decision trees together without overlapping leading to soft edge response, the corresponding edge map is also stored, so the outcome could be obtained by simply averaging the outcomes of each decision tree.

The random forest estimates each pixel 64T times for the test image is segmented by a 16*16 window, with a stride of 2 pixels, which make each point appear in 16*16/4 patches. In the outcome of total 2T decision trees, T of them are used to give the final result.

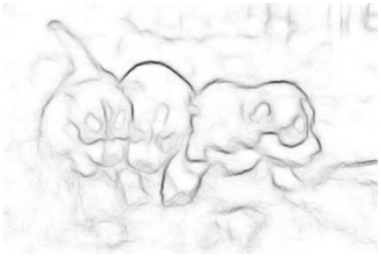


(a) multiscale=0, sharpen=0, nTreesEval=4, nms=0

(e) multiscale=1, sharpen=0, nTreesEval=4, nms=0



(i) multiscale=1, sharpen=0, nTreesEval=1, nms=0



(b) multiscale=0, sharpen=1, nTreesEval=4, nms=0

(f) multiscale=1, sharpen=1, nTreesEval=4, nms=0

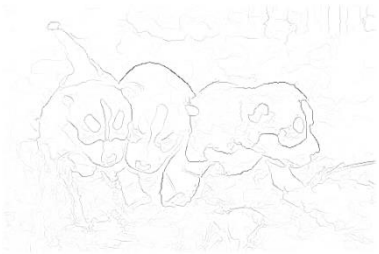
(j) multiscale=1, sharpen=2, nTreesEval=1, nms=0

(c) multiscale=0, sharpen=2, nTreesEval=4, nms=0

(g) multiscale=1, sharpen=2, nTreesEval=4, nms=0

(k) multiscale=1, sharpen=2, nTreesEval=8, nms=0

(d) multiscale=0, sharpen=2, nTreesEval=4, nms=1



(h) multiscale=1, sharpen=2, nTreesEval=4, nms=1

(l) multiscale=1, sharpen=2, nTreesEval=16, nms=1, binarized, Threshold=0.85

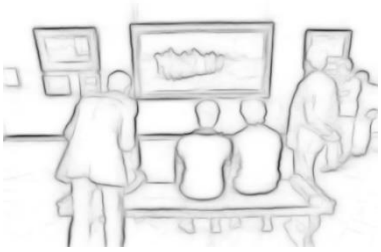
Fig. 6 Structure Edge Algorithm for Dogs.jpg

(a) multiscale=0, sharpen=0, nTreesEval=4, nms=0

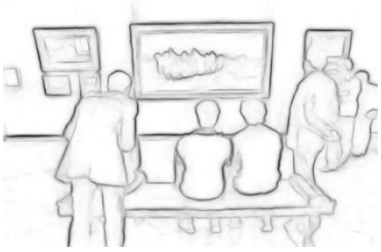
(e) multiscale=1, sharpen=0, nTreesEval=4, nms=0

(i) multiscale=1, sharpen=0, nTreesEval=1, nms=0

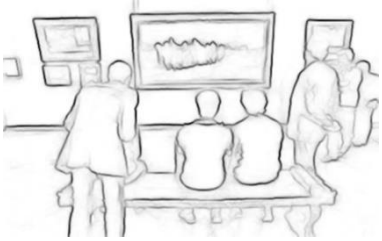
(b) multiscale=0, sharpen=1, nTreesEval=4, nms=0



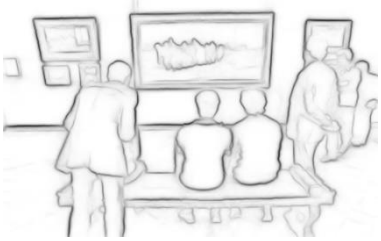
(f) multiscale=1, sharpen=1, nTreesEval=4, nms=0



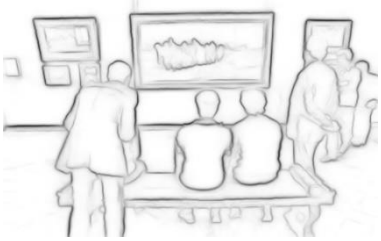
(j) multiscale=1, sharpen=2, nTreesEval=1, nms=0



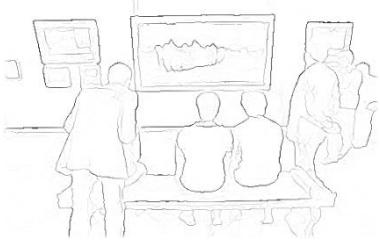
(c) multiscale=0, sharpen=2, nTreesEval=4, nms=0



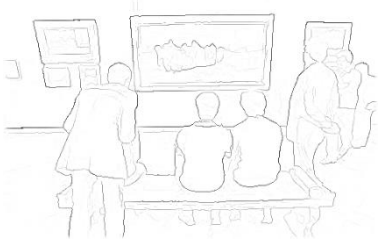
(g) multiscale=1, sharpen=2, nTreesEval=4, nms=0



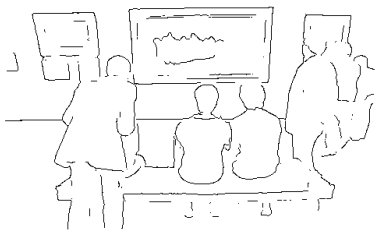
(k) multiscale=1, sharpen=2, nTreesEval=8, nms=0



(d) multiscale=0, sharpen=2, nTreesEval=4, nms=1



(h) multiscale=1, sharpen=2, nTreesEval=4, nms=1



(l) multiscale=1, sharpen=2, nTreesEval=4, nms=1, binarized, Threshold=0.75

Fig. 7 Structure Edge Algorithm for Gallery.jpg

(2) How the decision trees are constructed are illustrated in (1). For using a single decision tree to give the prediction could be affected by noise in training set heavily, so random forest model is carried out. Random forest contains several uncorrelated decision trees, each tree in random forest is trained by random features, then every tree will give input a prediction, they are averaged to get the final result.

(3) From the result, we could see that from (a) to (c), or (e) to (g), sharpen is set from 0 to 2, which increases contrast of different edges, leaving the width of edges unchanged. Parameter multiscale could make edge detection more accurately, from the first two columns, the foreground becomes darker and the background are blurred. nTreesEval could make more decision trees to make decision, which make results accurate, from the last column. Nms is the parameter which makes edges on the probability maps slimmer, leaving the maximum pixels in edges and shade off other pixels, from (d), (h) of Fig. 6 and Fig. 7. The edge map could be obtained by binarize the probability map with a given threshold.

(d) Performance Evaluation

In this part, results from different algorithms are compared with ground truth labeled by people, the algorithm with a higher F-score performs better.

Ground Truth	Sobel			Canny			Structure Edge		
	P	R	F	P	R	F	P	R	F
1	0.18	0.24	0.21	0.13	0.45	0.20	0.15	0.44	0.22
2	0.18	0.54	0.27	0.18	0.31	0.23	0.19	0.51	0.27
3	0.28	0.15	0.19	0.16	0.39	0.23	0.17	0.43	0.24
4	0.17	0.27	0.21	0.10	0.43	0.16	0.13	0.47	0.20
5	0.29	0.43	0.35	0.29	0.53	0.37	0.23	0.69	0.35
Overall	0.26	0.63	**0.37**	0.40	0.42	**0.41**	0.33	0.82	**0.47**

Table 1: Precision, Recall and F-Measure for Dogs.raw

Ground Truth	Sobel			Canny			Structure Edge		
	P	R	F	P	R	F	P	R	F
1	0.24	0.63	0.35	0.22	0.40	0.29	0.22	0.45	0.24
2	0.22	0.61	0.33	0.20	0.41	0.27	0.19	0.79	0.31
3	0.23	0.72	0.35	0.22	0.42	0.29	0.21	0.80	0.33
4	0.22	0.74	0.34	0.24	0.50	0.32	0.19	0.85	0.31
5	0.34	0.67	0.45	0.31	0.42	0.36	0.28	0.79	0.42
Overall	0.40	0.71	**0.51**	0.37	0.43	**0.40**	0.36	0.83	**0.51**

Table 2: Precision, Recall and F-Measure for Gallery.raw

(1) For Dogs.raw is more difficult to do edge detection, so the following discussion is mainly derived from table 1. From table 1, we could see that the effect of Sobel Algorithm is the worse

in all of the three algorithm, this algorithm has only one threshold to considerate, and the procedure is the simplest, however, if the threshold is high, there will be a lot of boundaries set to 0, which will lead to high false negative rate, so recall will be low, and if the threshold is low, there will be noise classified as boundaries, so the false positive rate will be low, so the F score will be low. The advantage of this algorithm is its low complexity and simple implementation, but from table 2, it obtains almost the same result as Structure Edge algorithm, which shows that Sobel algorithm may be perform better in sample images. Canny algorithm is the middle one in the three algorithms from table 1, the probability map is generated by the average of hundreds of outputs by MATLAB function edge(), for Dogs.raw, low threshold range from 0.1 to 0.5, high threshold range from 0.4 to 0.7, for Gallery.raw, low threshold range from 0.1 to 0.3, high threshold range from 0.4 to 0.6. There are two thresholds there, suitable threshold could improve the performance, with suitable low threshold, noise and texture could be filtered out, and edges could keep continuous, with suitable high threshold, edges could be detected thoroughly. However, how to choose threshold is a problem and Canny algorithm could get wide edges just as Sobel algorithm, and the noise which is higher than low threshold could not be eliminated. So, the precision, recall and F-Measure score is not too high. Structure Edge Algorithm is the best one in the three algorithms, with the design of random forest and preprocessing of input image, the information of images could be used relatively completely. However, the algorithm is very complex, need data to train and how to choose parameters are still problems.

(2) Gallery.raw gets a higher F-score, for the edges in Gallery.raw is more distinct and there is no texture information like the grass and fur of dogs in dogs.raw there. Besides, Gallery.raw is less complex for the pixel on the same item, such as people, chair and wall, vary less than that of dogs.raw for the spots on dogs.

(3) Precision could test how many pixels labeled as edge are really edges, and recall could test how many edge pixels are labeled as pixels. If only precision is considered, and we only label one point which stands for edge as edge, the precision could reach 100%, or, if only recall is considered, and we label all points as edge, recall could reach 100%, however, both of the situations are not desirable. F-Score could balance between recall and precision. Only if recall and precision are high, we could get a high F-score. In the situations above, one is 100%, but the other is a very small number if the number of edges is large, so F-score in the situation is very small. Suppose$P + R = C$, where C is a constant. So $R = C - P$, so $F = 2\frac{P(C-P)}{C} = 2\frac{PC-P^2}{C}$, so $\frac{\partial F}{\partial P} = 2\frac{C-2P}{C}$, so when $\frac{\partial F}{\partial P} = 0$, i.e. $R = P = \frac{1}{2}C$, F-score reaches its maximum value.

Problem 2: Digital Half-toning

(a) Dithering

Dithering is a half-toning method that could convert continuous image to discrete image with black and white, there are several dithering methods, the basic idea of them is to compare the pixel value to some threshold, below are some different dithering method.

1. Fixed Thresholding

For fixed thresholding, every pixel in image is traversed, and if the value is larger than the threshold, it will be set to 1, or set to 0. The result is shown in Fig.8 we could see that the feature in the picture, like cloud, light tower or stones on the beach, could not be displayed completely, no matter what the value of thresholds is, but the result with proper threshold, like threshold=125, is kind of stylish, it could tell where the objects are with some shade, but the sky is not displayed well.

2. Random Thresholding

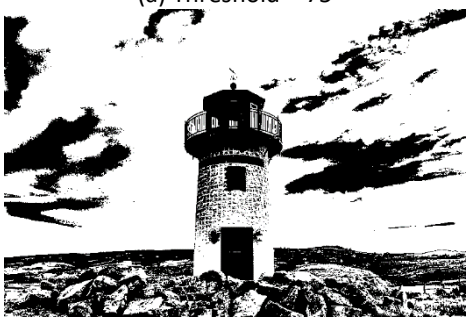
For random thresholding, the threshold is generated when each traversal of pixel. The result is shown in Fig.9. The outlook of the picture is preserved better than fixed thresholding, we could see the difference even on the door, the cloud, however, the whole picture is so messy, just like added noise.

3. Dithering Matrix

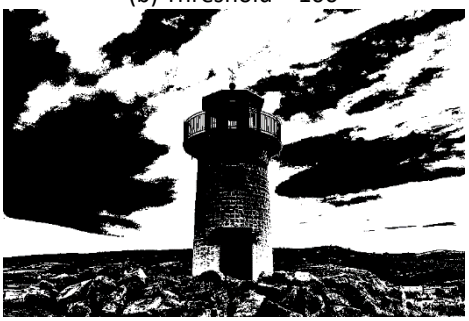
For dithering matrix, the threshold matrix is generated, pixels are binarized by the same pattern periodically. The result is shown in Fig.10. The detail of picture is preserved better, with the dithering matrix becomes larger, the picture become finer. We could see that the outcome of I2 is finer than fixed thresholding, and less noisy in comparison with random threshold. I8 and I32 look similar. These pictures are more suitable to print. However, there are artificial patterns on the output image, they are regularly distributed dots, as if there are girds on the image, and this algorithm needs extra storage space on devices, requiring more hardware support.

(a) Threshold = 75

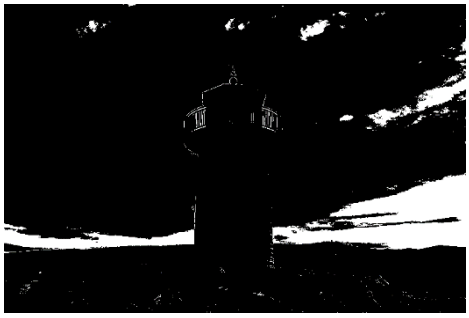
(b) Threshold = 100



(c) Threshold = 125



(d) Threshold = 150

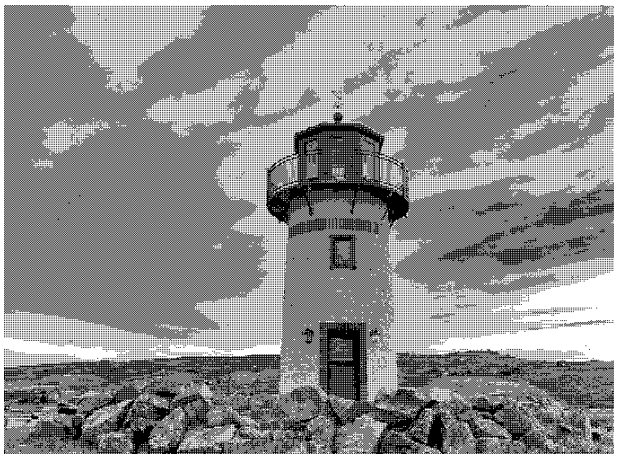


(f) Threshold = 175

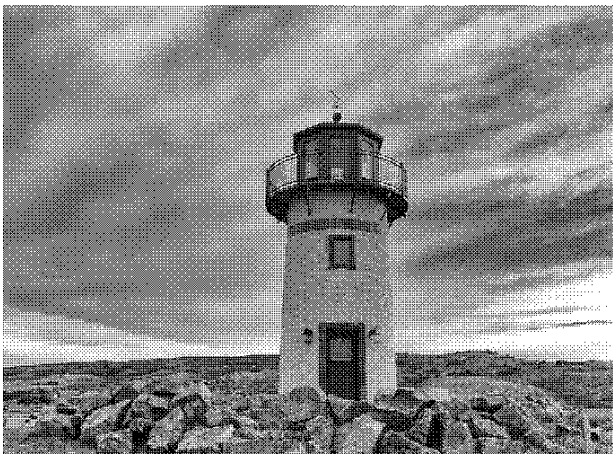
(g) Threshold = 200

Fig. 8 Fixed Thresholding Dithering

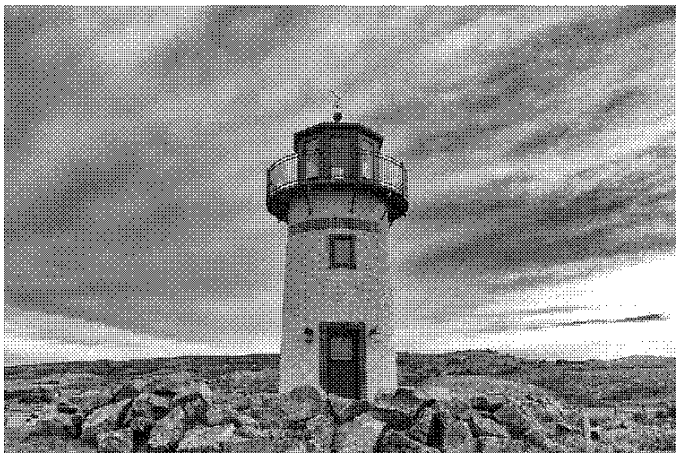
Fig. 9 Random Thresholding Dithering



(a) Dithering by I2



(b) Dithering by I8



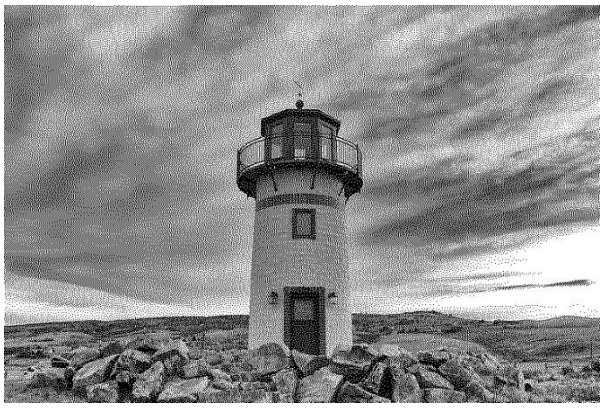
(c) Dithering by I32

Fig. 10 Dithering Matrix

(b) Error Diffusion

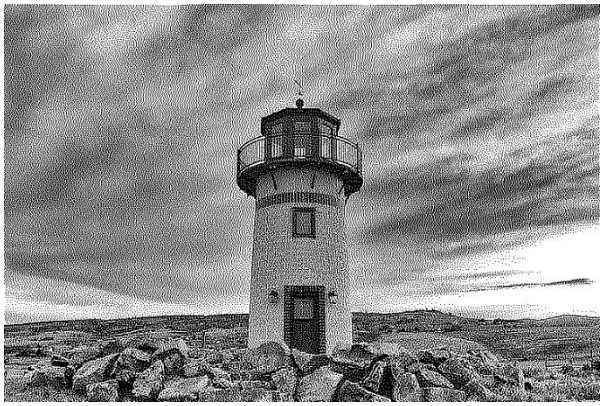
Error diffusion is another kind of half-toning method. Differently from dithering, error diffusion method could generate related noise, making the image looks more real-like.

Given the error diffusion matrix, procedure begins with the top left pixel, for every pixel, if the value is larger than 0.5, it will be set to 1, or set to 0, then the difference between the new binarized value and the original continuous value is compared, the difference is spread to other following points using error diffusion matrix. For the next pixel, before thresholding, it has already been added with error value from the former pixels, then decide its binary result. As is shown in Fig. 11, the output of these three error diffusion methods looks similar, all of them looks more like the original image than the methods in part (a). The difference derives from the fact that noise in part (b) relate with pixels on images, but part (a) do not use noise, or use random noise which has nothing to do with the pixel values, or use fixed dithering. So, this could keep

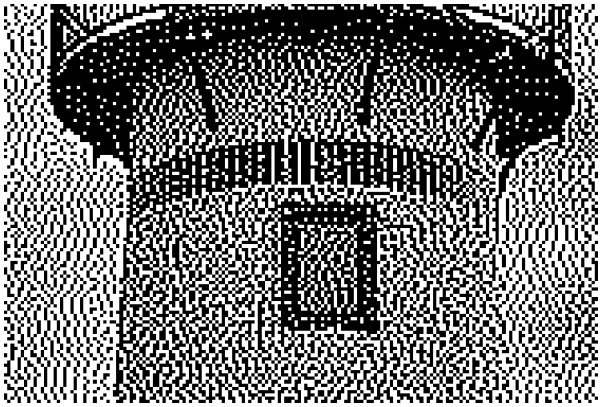


(a) Floyd-Steinberg Error Diffusion

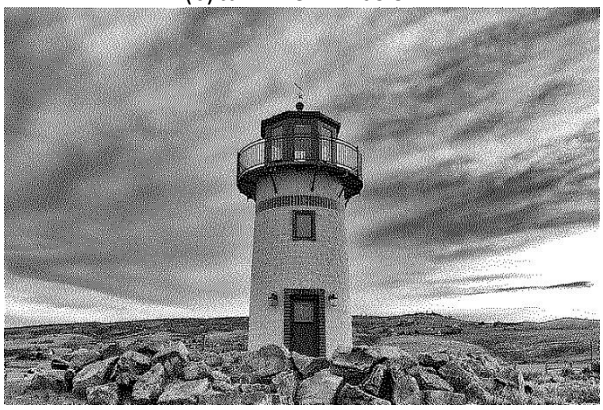
(d) Detail of (a)



(b) JJN Error Diffusion



(d) Detail of (b)



(c) Stucki Error Diffusion

(d) Detail of (c)

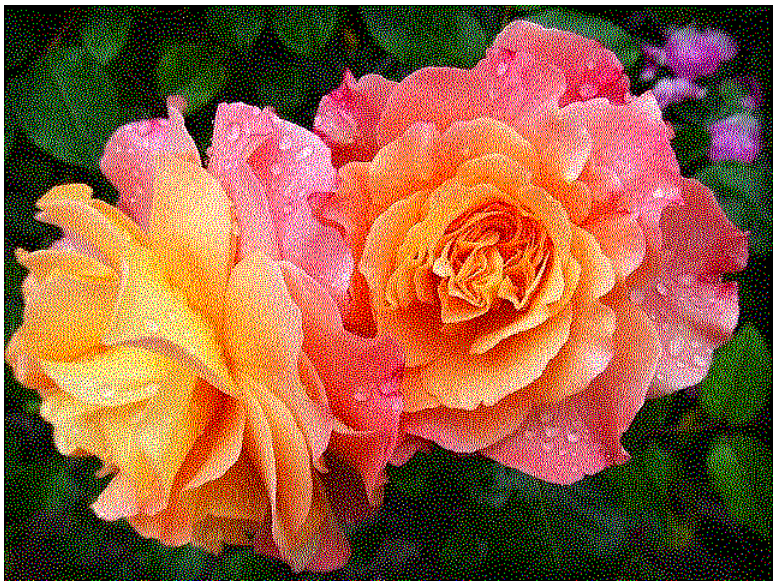
Fig. 11 Error Diffusion

more information of the original image. From the detail of these images, we could see that the distribution of black and white pixels is slightly different, Floyd- Steinberg's method tend to keep less detail and rougher than the other two method, the edge of the light house is not very sharp, but the shape of the edge is smooth, JJN's method makes the edge become sharper, making the object more distinct, but there are some discontinuity alone the edges. Stucki's method could make the edge a little sharper, and the edge is also smooth. In common, there are artificial patterns like stripe, or curve, making the image kind of messy. One way to overcome this shortcoming is to change the sequence of scanning, not in line, but in a 2D direction. Before scanning, we could also add some noise which could disturb the pixels, only changing the original image slightly.

(c) Color Halftoning with Error Diffusion

Error diffusion method could be used to process 3-channels color images. The most intuitive method is to apply error diffusion to three channels separately, called separable error diffusion. This method could binarize image, but do not take the correlation between channels into account. To overcome this shortcoming, MBVQ-Based error diffusion is carried out, which looks for the less brightness variance when deciding the projection pixels, making image less noisy.

(1) Separable Error Diffusion



(a) Separable error diffusion result

(b) Detail

Fig. 12 Separable Error Diffusion

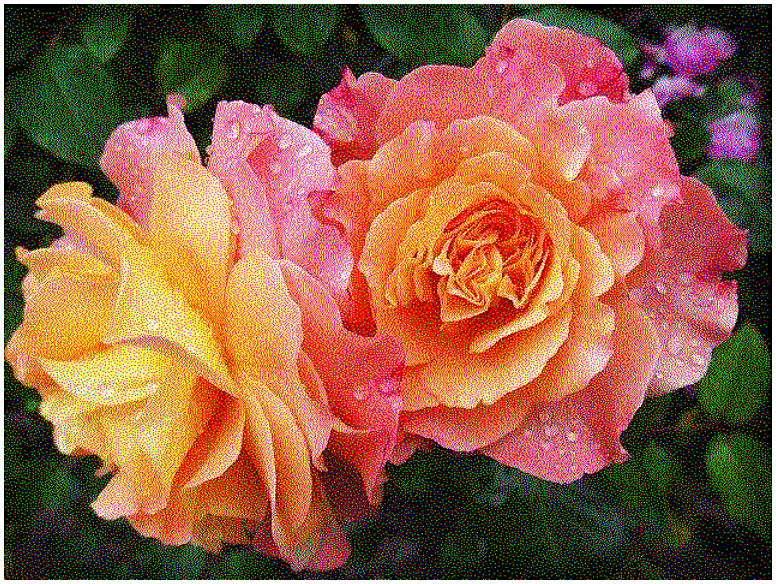
For separable error diffusion, the image is first separate into three channels, and they are processed as a grayscale image by the algorithm in (b), then the three channels are combined together again. The result is shown in Fig. 12, the main shortcoming of separable error diffusion is that this algorithm could incur artificial pattern in the image. Because this algorithm only takes every channel into account, leaving the relationship among them alone. So, there are a lot of undesired dots in the image, for example, on the top left corner and the bottom right corner of the image, there are two part of black background, in Fig.12, these regions are not “dark” enough, making the areas looks kind of abrupt. Also, because this algorithm only takes every single channel into account, so the strips stated in part (b) also exist. From the detail on a part of yellow flower, there are a lot of black or blue dots, which should have not appeared on the yellow flower, even through there are error spread from other pixels. These undesirable pixels make the image noisy.

(2) MBVQ-based Error diffusion

1) The key idea of MBVQ is that we human are more sensitive to brightness of color rather than color tone. To make the change of colors between neighborhood smoothly, for a certain pixel, this algorithm decides a tetrahedra first, which could minimize the change of brightness, then in this tetrahedra, the pixel is projected into the nearest vertex (this procedure is completed with the help of the opensource file getNearestVertex.m), to keep the color tone changes most slightly. After the vertex is found, the value of the corresponding point could be set, then error diffusion method could be implemented as previously stated. The algorithm is better than separable error diffusion, for it takes human visual into account, the outcome is more smooth than separable error diffusion.

2) As is shown in Fig.13, the image is smoother than Fig.12, for example, in the black region, MBVQ tends to use more blue and less green to represent it, while there are more red and green in the black region in Fig.12. In the water drop on flowers, where there are more white, Fig.13 tend to use more cyan and yellow, while Fig.12 has no preference about color tone, there are still

red pixels. From the detail of the same part with Fig.12, we could see that there are no black dots, only some blue dots, making the image smoother.



(a) MBVQ-Based error diffusion result

(b) Detail

Fig. 13 MBVQ-Based Error Diffusion

Appendix

Codes for Structure Edge: codes in edgesDemo.m are slighted changed.

Parameters are adjusted below:

```
%% set detection parameters (can set after training)
model.opts.multiscale=1;          % for top accuracy set multiscale=1
model.opts.sharpen=2;             % for top speed set sharpen=0
model.opts.nTreesEval=4;          % for top speed set nTreesEval=1
model.opts.nThreads=4;            % max number threads for evaluation
model.opts.nms=1;                 % set to true to enable nms
```

Following codes are slightly adjusted so that the outcome is desirable.

```
%% detect edge and visualize results
I = imread('Dogs.jpg');
tic, E=edgesDetect(I,model); toc
figure(1); im(I); figure(2); im(1-E);
imwrite(double(1-E), 'Dogs_SE.tif');
```

Codes for Performance Evaluation:

Using codes in P1_4.m, with the help of edgesEvalImg(). P1_4.m is provided in source code.
In edgesEvalImg.m, following codes are changed to get results regarding different ground truth:

```
G=load(G); G=G.groundTruth; n=length(G);  %to get the overall result
G=load(G); G=G.groundTruth(i); n=length(G);  %to get the ith ground truth result
```