EE 569 HOMEWORK # 2

1.1 Abstract and Motivation

Human vision perceives appearance and shape(edges). Even if the image has better PSNR, it may not be pleasing for human perception. Edge detection is an important step in image analysis, pattern recognition, computer vision techniques. It is often difficult to detect an edge for the following reasons:
1. Denoising may affect the sharpness of the edge.
2. Localization is poor
3. Gradient: Choice of parameters(thresholds)
4. To differentiate edges and texture (high frequency pattern)

Edge detection depends on neighbourhood size and template matching. Different methodologies like Prewitt, Sobel, Laplacian of Gaussian, Canny edge detector, Probability of boundary, Sketch Token and Structured Edge are used. Evaluation of the performance is essential for quantitative comparison of different edge detection methods.

1.2 Approach

1.2.a Sobel Edge Detector

Edge can be identified due to sudden/sharp change of intensities/luminosity resulting in positive, maximum and negative first order derivative gradient. Since it is difficult to implement the continuous domain approach, discrete finite difference is adopted.
To find the edges in an image given the dimensions, the following operations are performed:
Step 1: Rec.ITU-R BT.601-7 calculates gray scale from RGB scale using the following formula:
0.299 * R + 0.587 * G + 0.114 * B
Step 2: Boundary of the image is extended according to the filter size (Sobel matrix size)
Step 3: Compute the finite difference Gr(Gx) and Gc(Gy) at the centre pixel by convolution

$$Gr = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad Gc = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Step 4: Gradient map is obtained by performing the following operation at each pixel
$G=\sqrt{Gr^2 + Gc^2}$
Step 5: Cumulative Histogram of normalized Gradient map is used to set threshold for obtaining edge map depending on the percentage of edges required to be displayed on the Edge map
Step 6: Gr, Gc, are normalized to 0~255 for output

1.2.b Canny Edge Detector

Canny edge detection technique is performed by the following operations:
1. Convolution with derivative of Gaussian: For suppression of noise.
2. Magnitude and orientation of gradient is computed.
3. Non-Maximum Suppression: Search for an edge in a particular direction that will result in finding a maximum point in the edge since it is orthonormal, and the remaining points are suppressed in contrast to the case of Sobel.
4. Hysteresis Thresholding: Contours which are blur due to background, cannot be connected. Thresholding allows smaller gradient to be considered as an edge depending on neighbouring pixel. Two thresholds "high" and "low" divides the gradient map into regions of edge, non-edge and ambiguous region. The edges in the ambiguous region are more probable to be an edge, if the neighbouring pixels are edge in contrast to neighbouring pixels which are non-edge.

Generating edge maps for different thresholds are performed using the following operation:
Step 1: Rec.ITU-R BT.601-7 calculates gray scale from RGB scale using the following formula:
0.299 * R + 0.587 * G + 0.114 * B
Step 2: Gray scale image is binarized with threshold 127.
Step 3: Parse gray scale image and threshold vector to edge() function in Matlab R2018a selecting 'canny'
Step 4: Binary image obtained from the function output has background as black and edge as white. For the sake of generalization, the binary image is negated, and the resulting image is displayed.

1.2.c Structured Edge

Structured Edge is a data driven approach for detecting an edge. Each edge have a structure. Random forest classifier is used to identify whether a pixel is an edge or not. Random forest consists of many decision trees which are trained using random sample of data and feature set. A 16 x 16 patch segmentation mask is used to predict whether a pixel is an edge or not. Alternative 0's and 1's can be used to detect a contour.
Following procedure was adopted to obtain Structured edge output:
Step 1: Initailize few parameters to train the decision tree/forest with Training data using Structured edge Toolbox.
Step 2: Train structured edge detector using edgeTrain() function
Step 3: Set detection parameters like multiscale, sharpen, nThreads, nTreesEval, nms(non maximum suppression)
Step 4: Load the image for which edges have to be detected and run edgesDetect() function with required parameters, with output as probability edge map
Step 5: Probability edge map can be converted to Binary edge map using a threshold

1.2.d Performance Evaluation

To perform quantitative comparison between different edge maps obtained by different edge detectors. The ultimate goal of edge detection is to enable the machine to generate contours of priority to human being. For this reason, we need the edge map provided by human (called the ground truth) to evaluate the quality of a machine-generated edge map. However, different people may have different opinions about important edge in an image. To handle the opinion diversity, it is typical to take the mean of a certain performance measure with respect to each ground truth, e.g. the mean precision, the mean recall, etc.

Following procedure was adopted to obtain F score, precision, recall etc

Step 1: The Edge map along with the groundtruth have to be parsed to the function edgesEvalImg() function.

Step 2: The output arguments [thrs,cntR,sumR,cntP,sumP,V] are made use to calculate the precision and recall.

Step 3: For a given threshold, precision(P) is calculated as cntP/sump and recall(R) is calculated as cntR/sumR for each groundtruth image.

Step 4: F score is calculated for each groundtruth image, mean precision and mean recall against different ground truth are evaluated and F measure is calculated for the same using

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

1.3 Results:

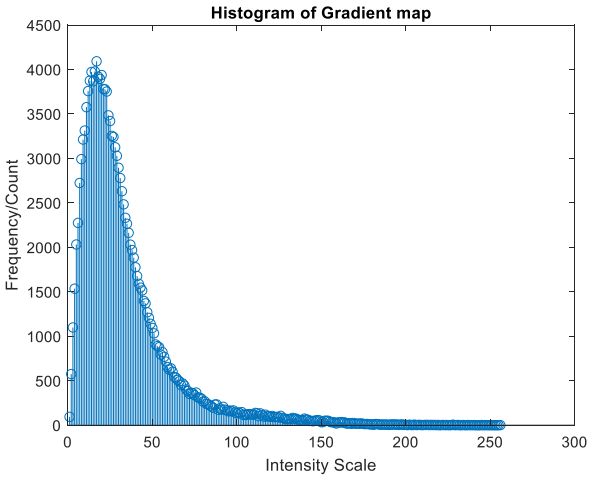
1.3.1 Sobel Edge Detector

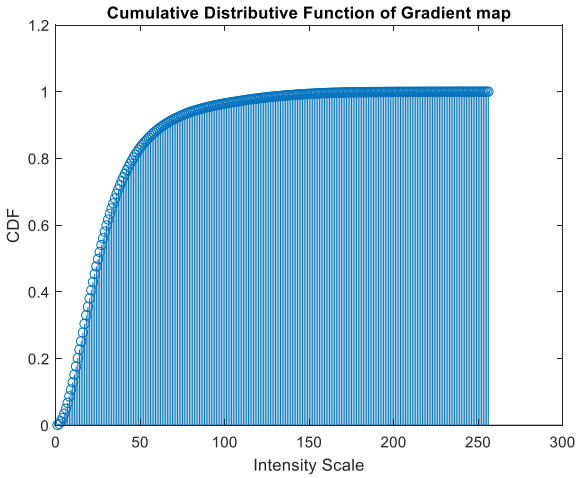
Original Image

Normalized Gr : x gradient

Normalized Gc : y gradient

Normalized Gradient map



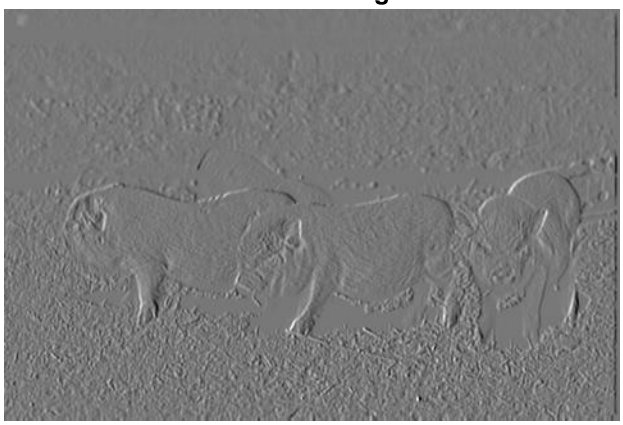


Edge Map with 95% edge required to be displayed

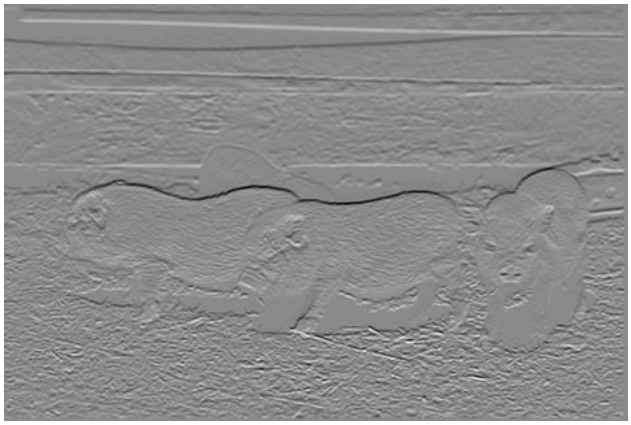
Edge Map with 90% edge required to be displayed

Original Image

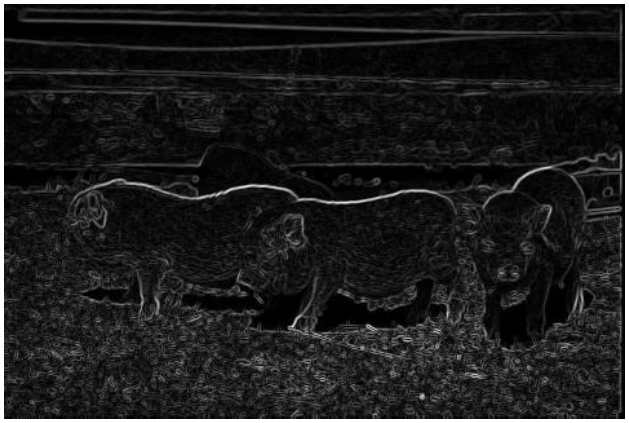
Normalized Gr : x gradient

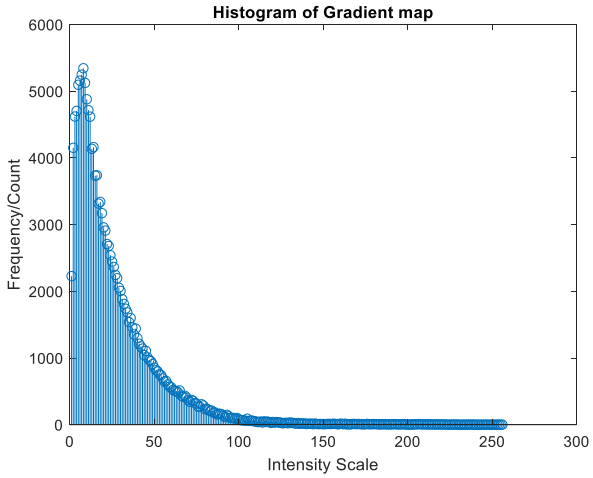


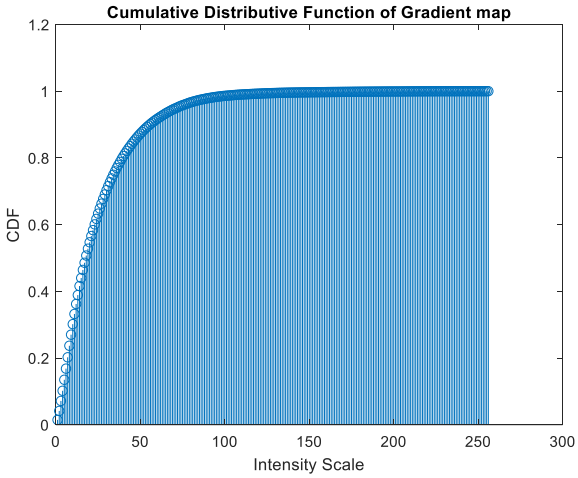
Normalized Gc : y gradient



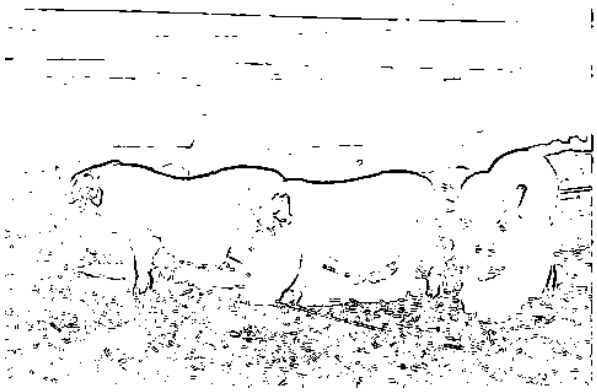
Normalized Gradient map



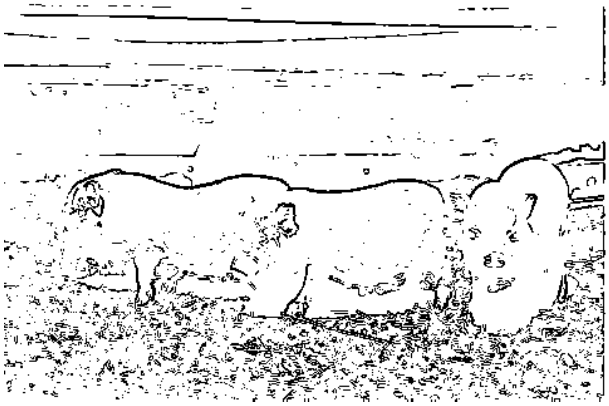




Edge Map with 95% edge required to be displayed

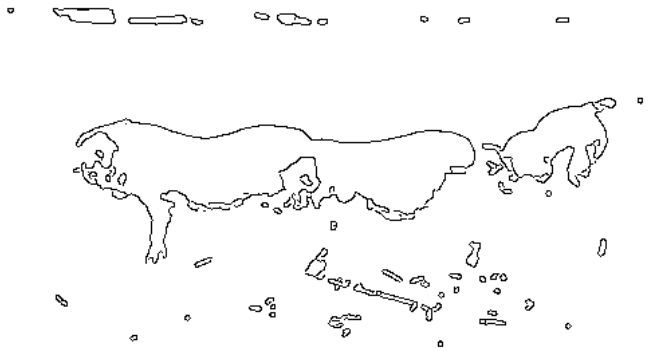


Edge Map with 90% edge required to be displayed

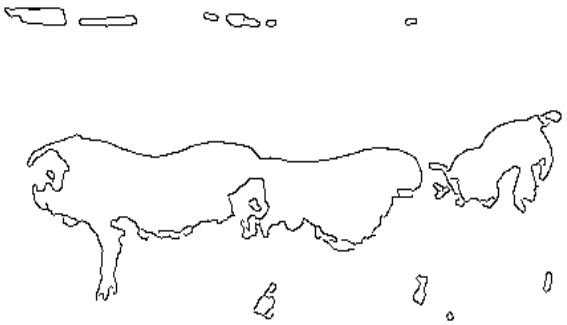


1.3.b Canny Edge Detector Results:

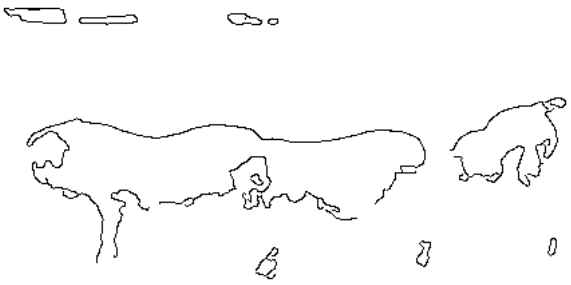
Edge Map with Low Theshold = 15 and High Threshold = 35



Edge Map with Low Threshold = 15 and High Threshold = 70

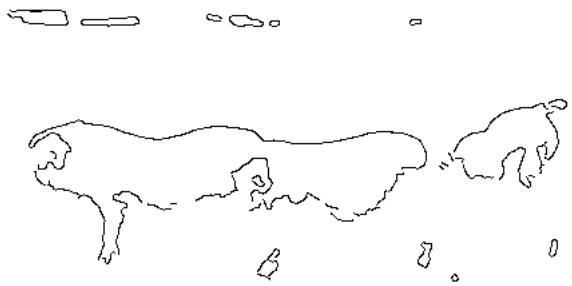


Edge Map with Low Threshold = 15 and High Threshold = 90

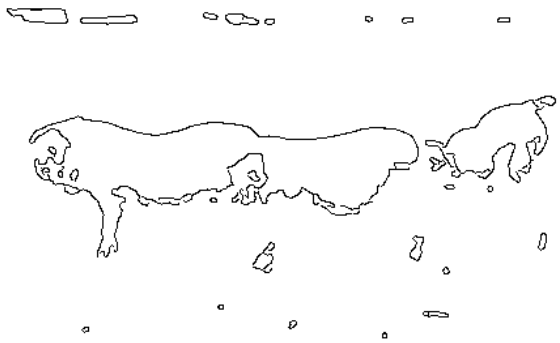


Edge Map with Low Threshold = 80 and High Threshold = 90

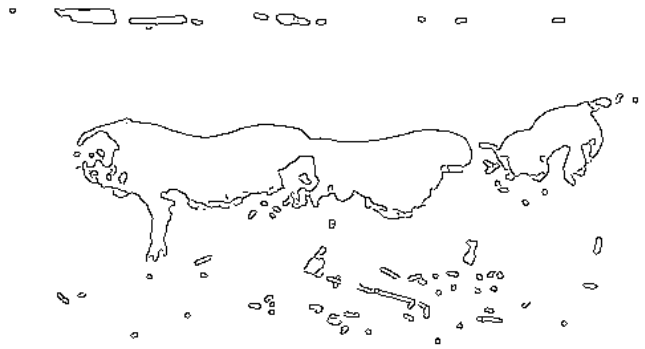
Edge Map with Low Threshold = 50 and High Threshold = 70



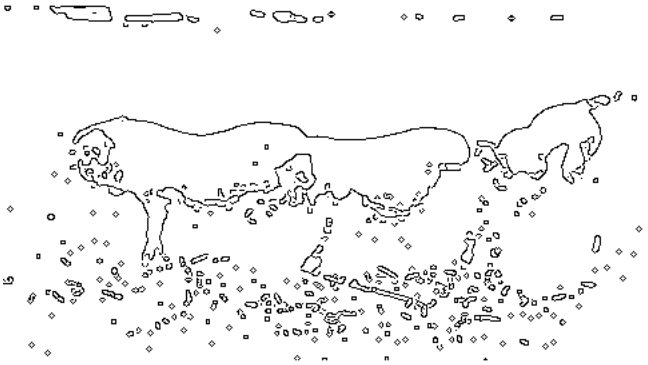
Edge Map with Low Threshold = 20 and High Threshold = 50



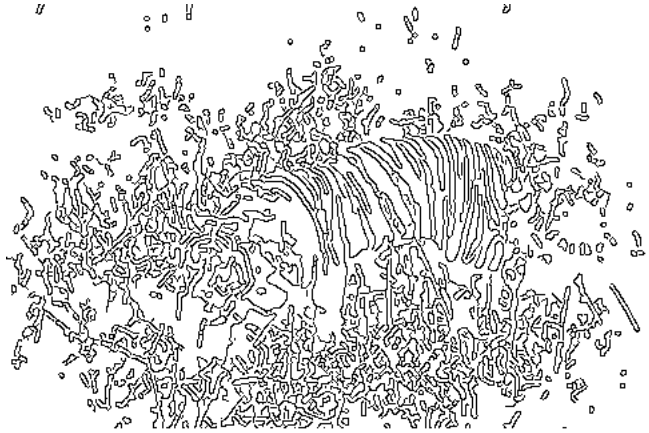
Edge Map with Low Threshold = 20 and High Threshold = 30



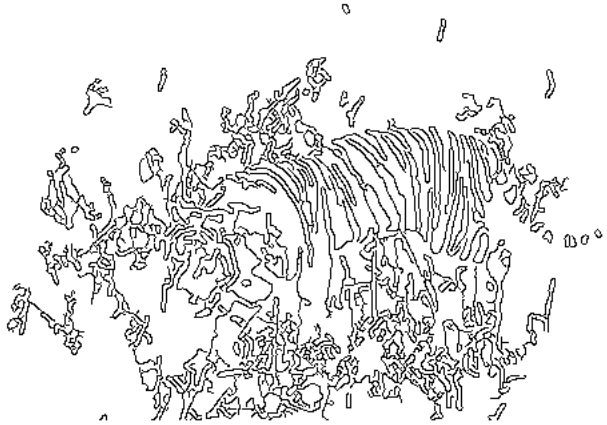
Edge Map with Low Threshold = 5 and High Threshold = 10



Edge Map with Low Threshold = 15 and High Threshold = 35

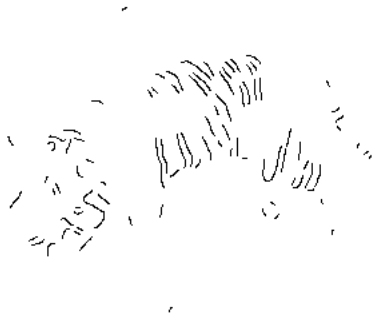


Edge Map with Low Threshold = 15 and High Threshold = 70



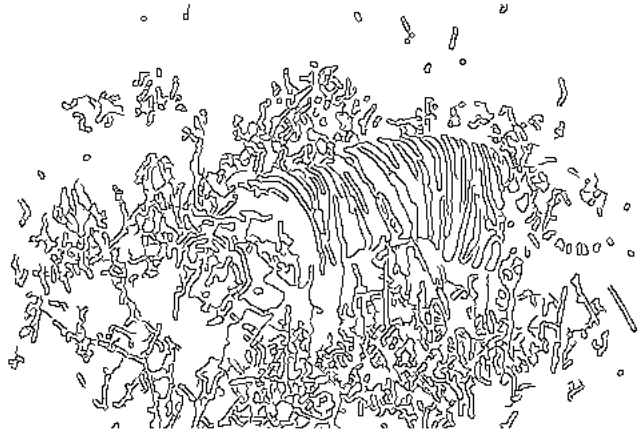
Edge Map with Low Threshold = 15 and High Threshold = 90

Edge Map with Low Threshold = 80 and High Threshold = 90

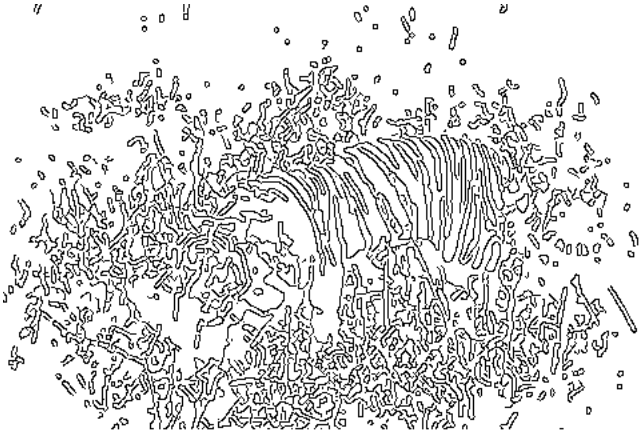


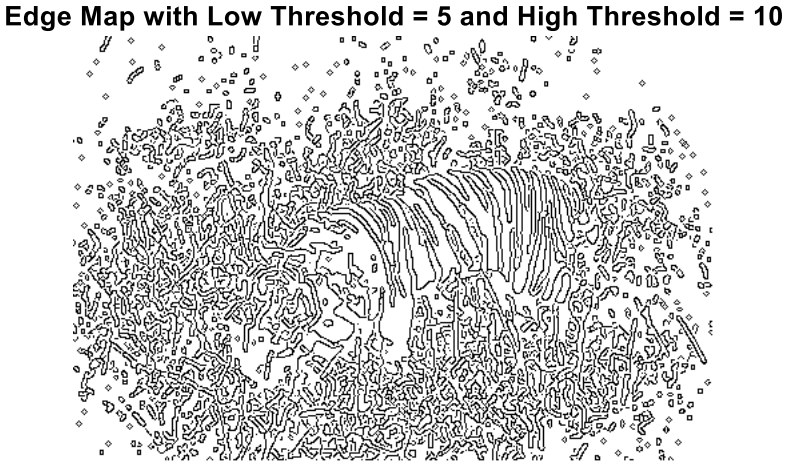
Edge Map with Low Threshold = 50 and High Threshold = 70

Edge Map with Low Threshold = 20 and High Threshold = 50

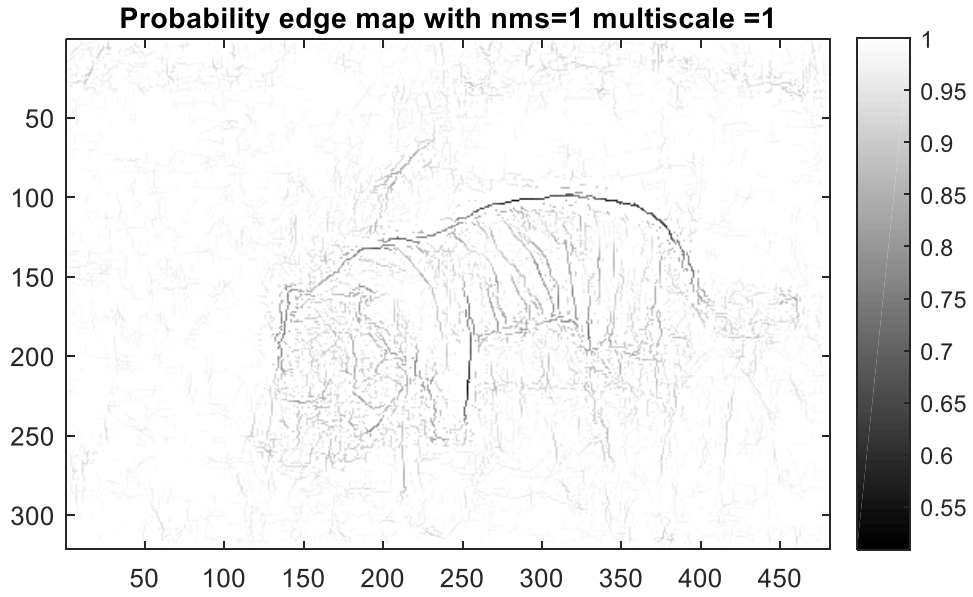


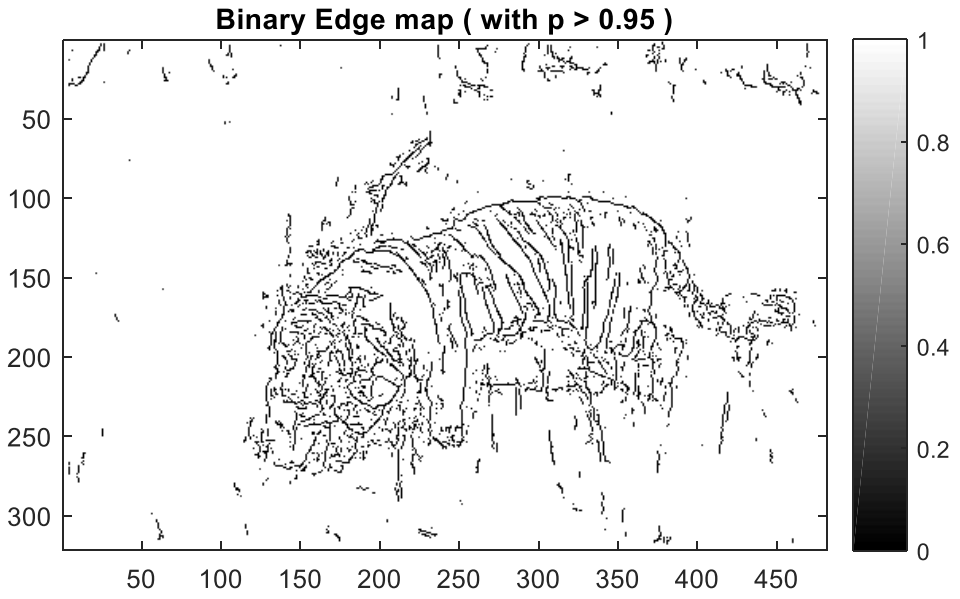
Edge Map with Low Threshold = 20 and High Threshold = 30

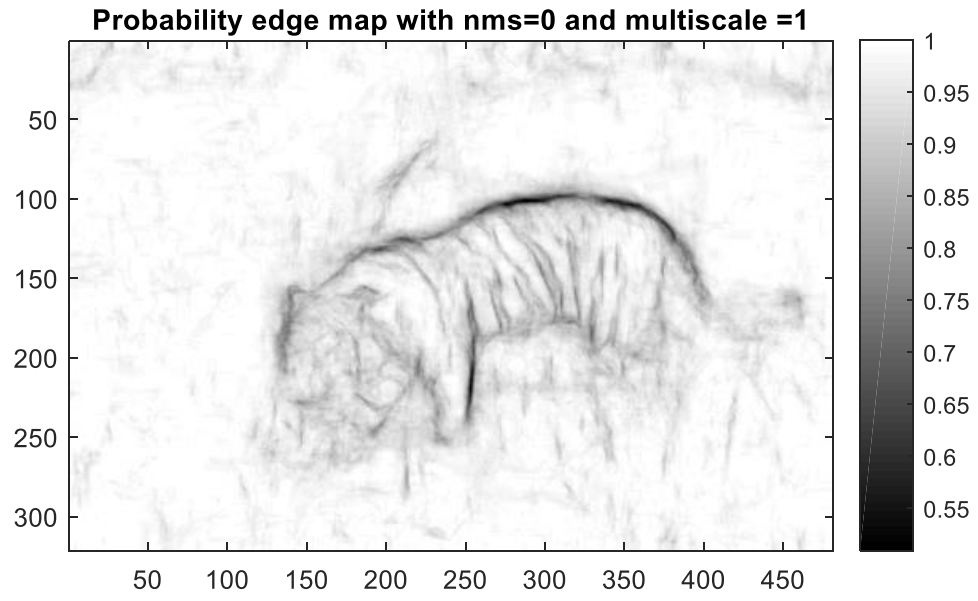


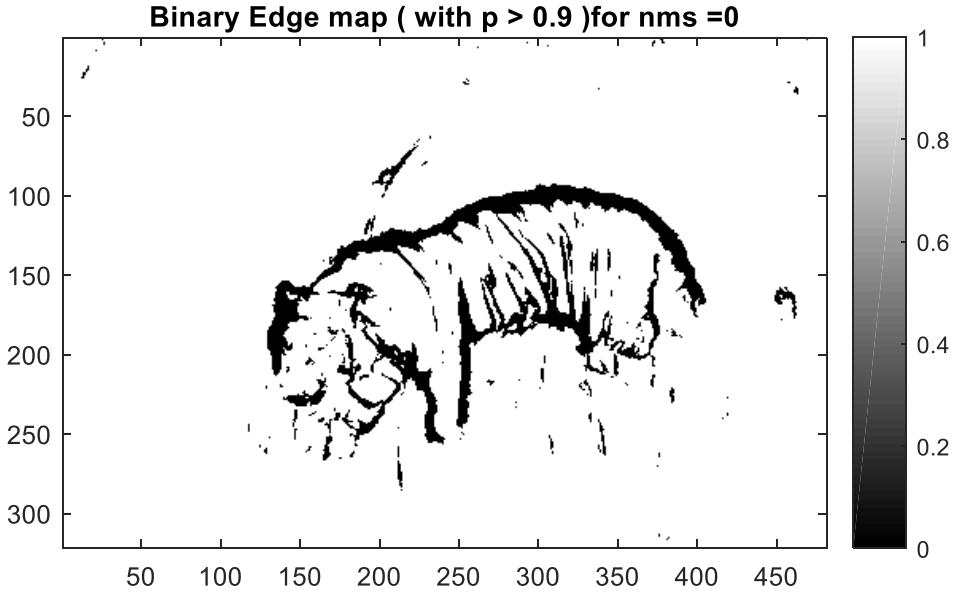


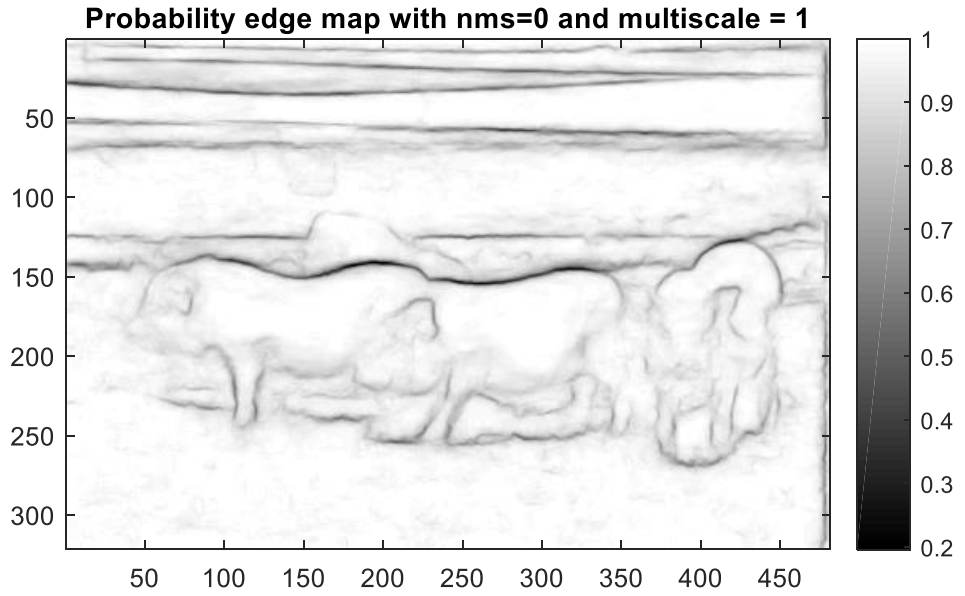
1.3.c Structured Edge

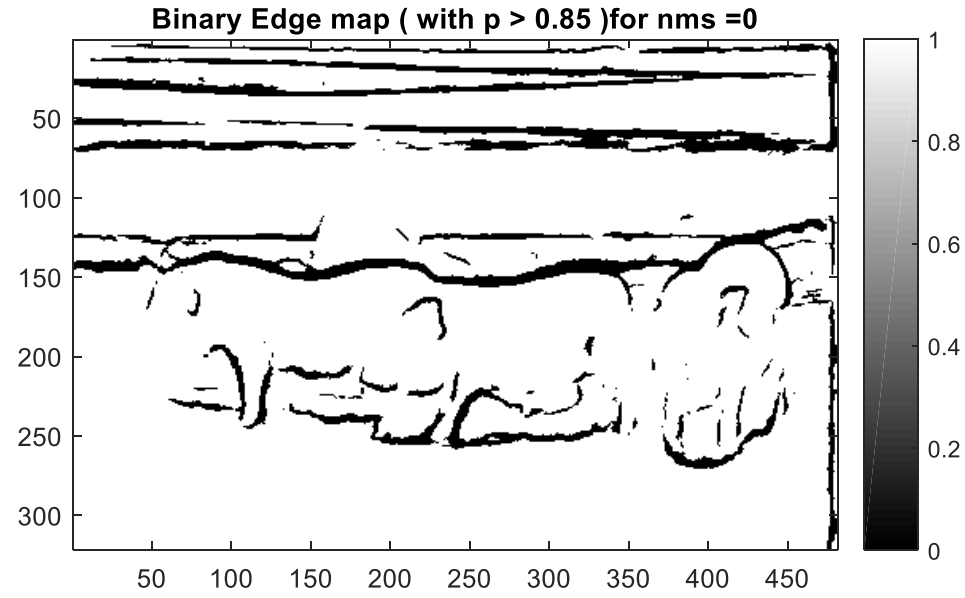


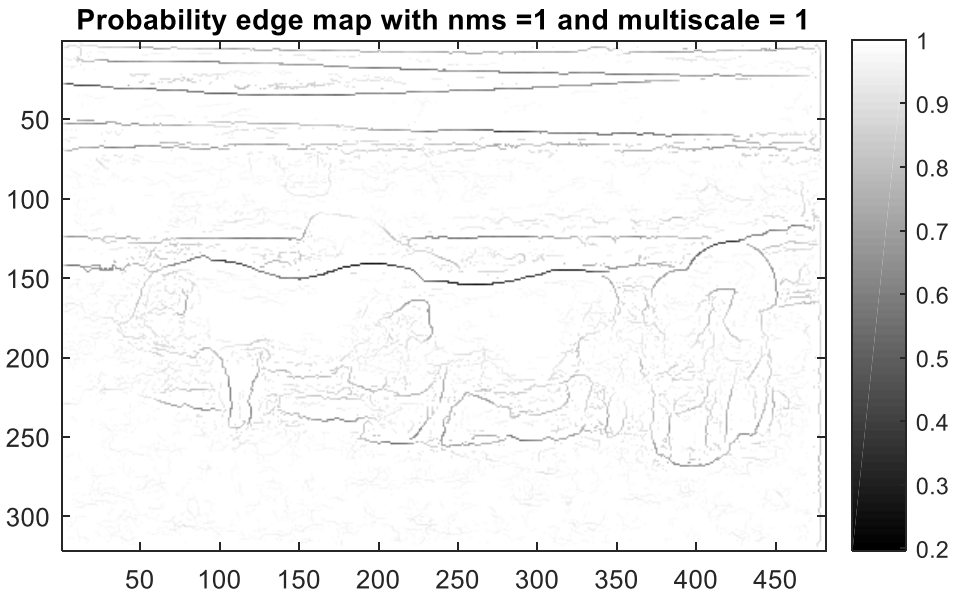

Binary Edge map (with p > 0.95)
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0
Binary Edge map (with p > 0.9)
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

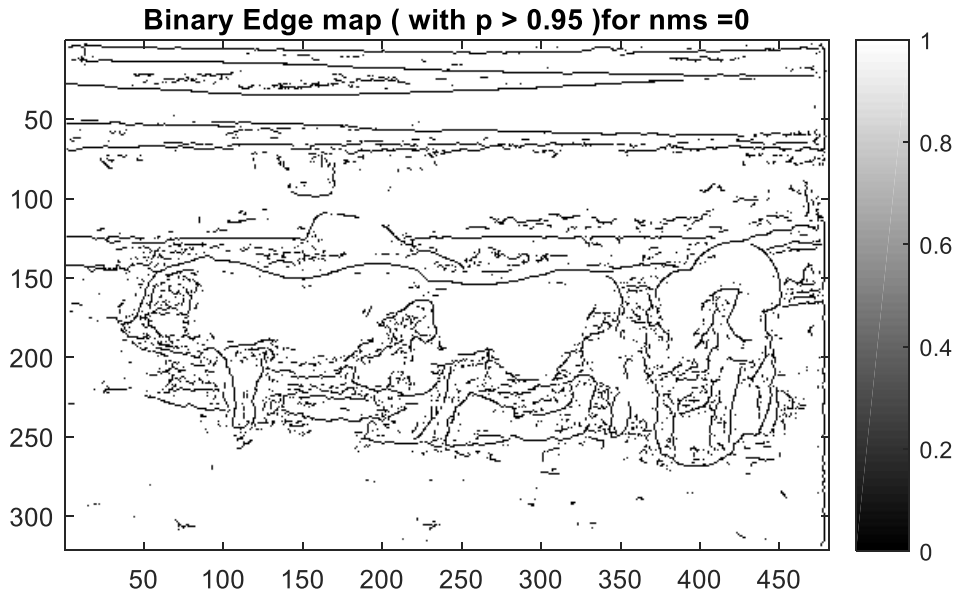

Probability edge map with nms=0 and multiscale =1
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55

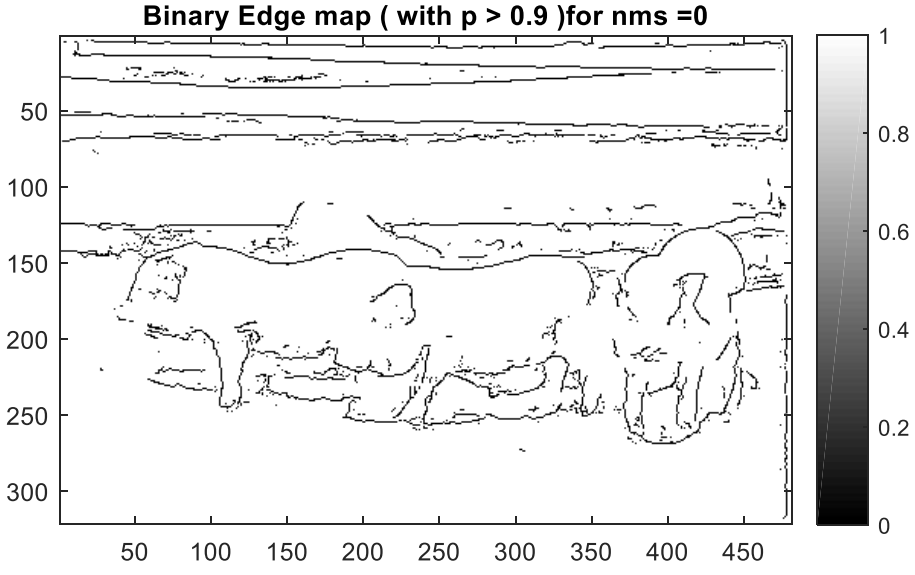

Binary Edge map (with p > 0.9)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0


Probability edge map with nms=0 and multiscale = 1
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2


Binary Edge map (with p > 0.85)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0
Binary Edge map (with p > 0.9)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0


Probability edge map with nms =1 and multiscale = 1
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

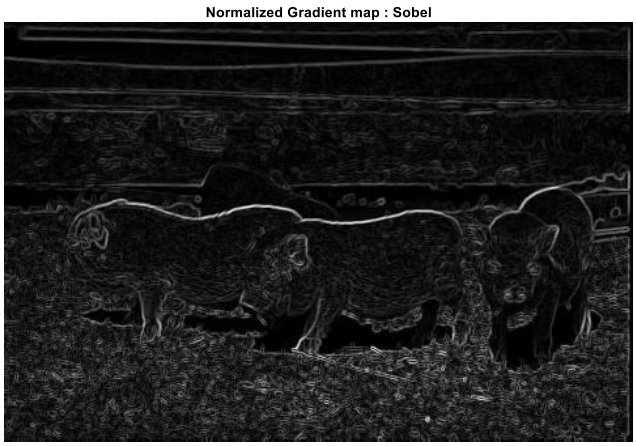

Binary Edge map (with p > 0.95)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

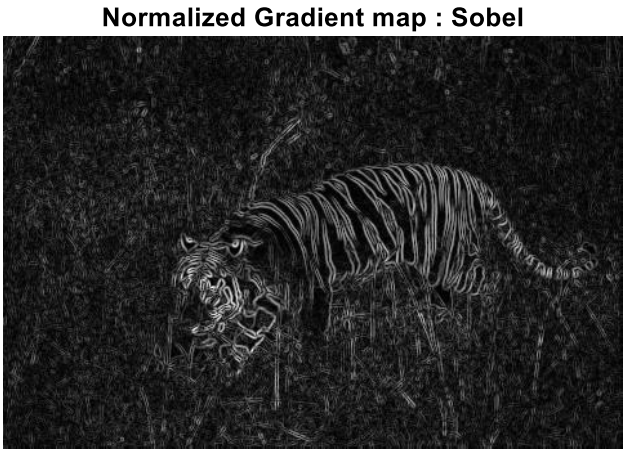

Binary Edge map (with p > 0.9)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

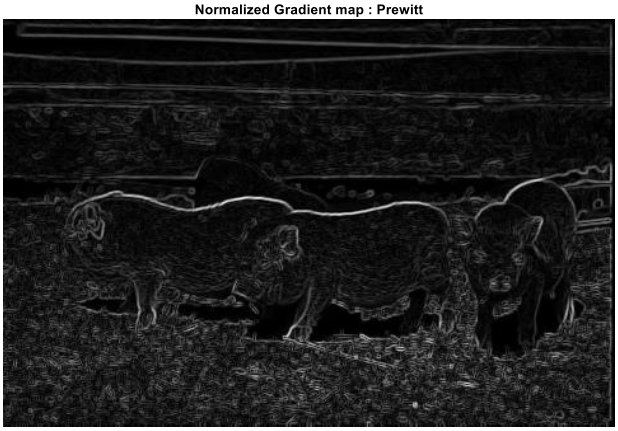
1.4 Discussion

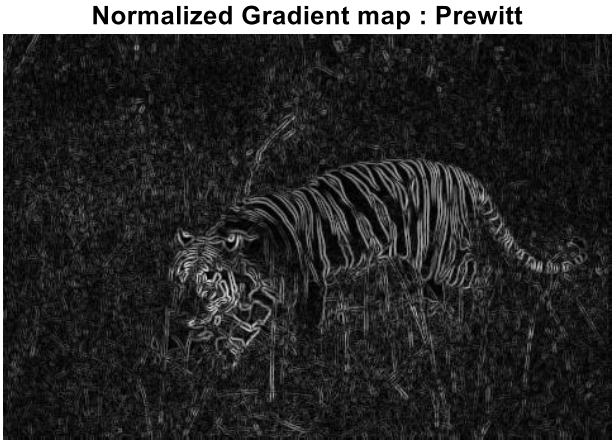
1.4.a Sobel Edge Detector

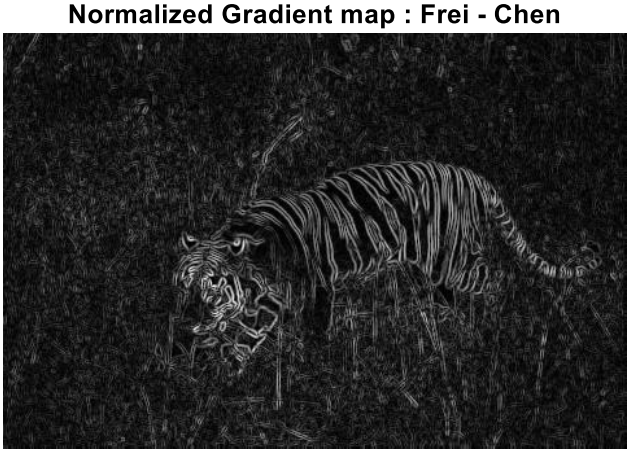
*** Following are the Normalized Gradient map for Sobel, Prewitt and Frei – Chen masks of edge detectors:**







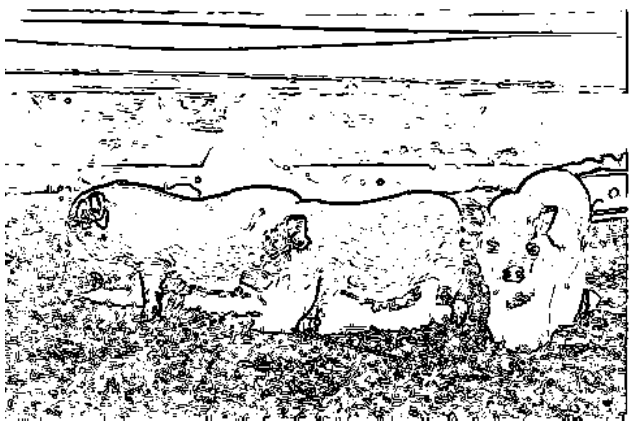




*** Following are the edge maps for different thresholds:**

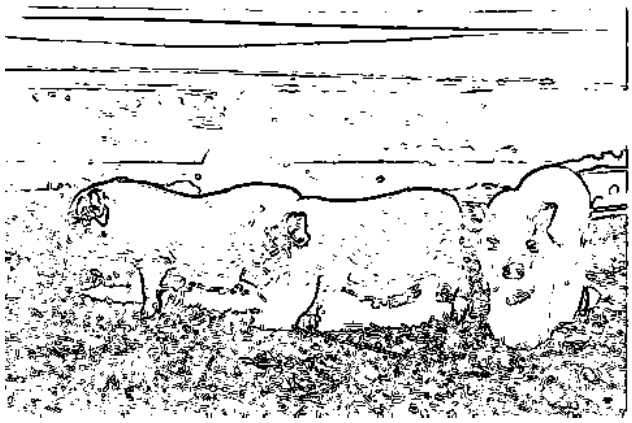
Threshold = 80%

Threshold = 80%



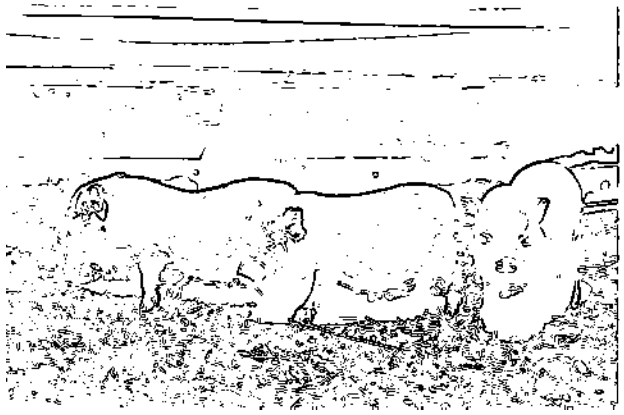
Threshold = 85%

Threshold = 85%



Threshold = 90%

Threshold = 90%



Threshold = 95%

Threshold = 95%

Threshold = 97%

Threshold = 97%

Threshold = 98%

Threshold = 98%

1. First order gradient is not good for edge detection since the threshold will change the output drastically.
2. Sensitive to noise.
3. Capable of detecting edges but poor localization of edges (contour).
4. Computational complexity is low as it is simple to implement.
5. Pixel wise estimation is performed based on first order gradient. It does not make use of other features like channel, brightness, luminance.
5. Threshold ranging from 80% to 98% shows a decrease in the density of the edges displayed.
7. Setting too low threshold will incorporate noise while setting too high threshold may not be suffice to distinguish the important edges. Thus, F- score is low.

6. The threshold set depending on top % edge to be displayed, is not unique. Each image will have different foreground and background. The point of interest or focus of edge detection varies from applications from identifying the main object (Tiger, pig) to identifying all objects. The threshold of 97 % is sufficient to detect the tiger while the same threshold when used for pig image fails since the boundary/edge of the pig is clear when threshold is 95 %.

1.4.b Canny Edge Detector

1. Non-Maximum Suppression: Search for an edge in a particular direction which will result in finding a maximum point of an edge since they are orthonormal, and the remaining points are suppressed. Unlike Sobel, the edges are thin. This feature accounts for better localization of an edge. Non- maximum suppression helps to suppress all the gradient values except the local maxima, which has the sharpest intensity change. The algorithm for determining the local maxima is:

a. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.
b. If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction (i.e., the pixel that is pointing in the y-direction, it will be compared to the pixel above and below it in the vertical axis), the value will be preserved. Otherwise, the value will be suppressed.

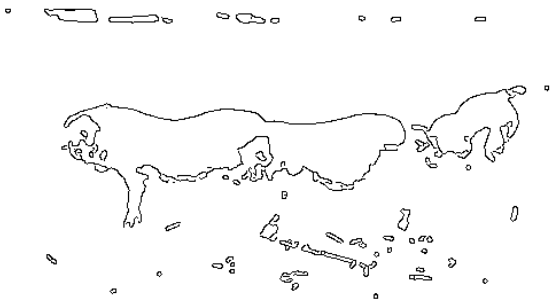
2. Hysteresis Thresholding: Contours which are blur due to background, cannot be connected. Thresholding allows smaller gradient to be considered as an edge depending on neighbouring pixel. Two thresholds “high” and “low” divides the gradient map into regions of edge, non-edge and ambiguous region. The edges in the ambiguous region are more probable to be an edge, if the neighbouring pixels are ‘edge’ or above the high threshold in contrast to neighbouring pixels which are below the low threshold.

Discussion:

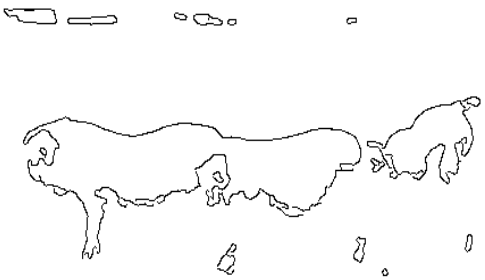
1. Improved PSNR and good detection of edges
2. Well localized
3. Second order statistics is used and is more stable
4. Computational complexity is high and time consuming
5. Derivative of gaussian filter smooths both noise and edge. In order to have better accuracy, smoothing should be performed to noise instead of edge.
6. Two threshold comparison:
   a. In Case 1 (mentioned below), for or the same lower threshold and for different higher threshold values, it is evident that the number of edges is more in case of low higher threshold in both Pig and Tiger edge map.
   b. In Case 2 (mentioned below), it can be observed that the edges are connected (contour) in a better manner when there is large difference between the two thresholds.
   c. In Case 3 (mentioned below), setting a very low value to lower threshold and setting higher value for high threshold may result in low recall. Both the thresholds have to be chosen carefully.
   d. Circumstances where the edge detection is necessary for segmentation on an object, the two-threshold play a vital role.
7. Robust method to determine dual – threshold value is necessary due to the fall-backs as stated above.

Case 1:

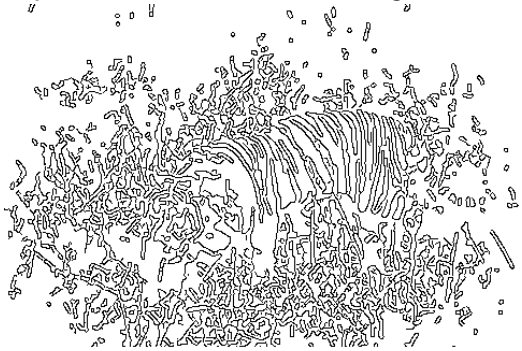
Edge Map with Low Theshold = 15 and High Threshold = 35



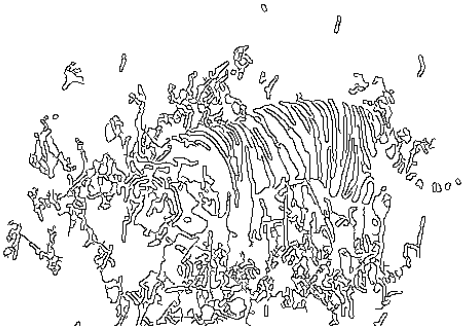
Edge Map with Low Threshold = 15 and High Threshold = 70



Edge Map with Low Threshold = 15 and High Threshold = 35

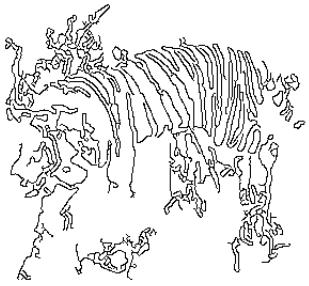


Edge Map with Low Threshold = 15 and High Threshold = 70



Case 2:

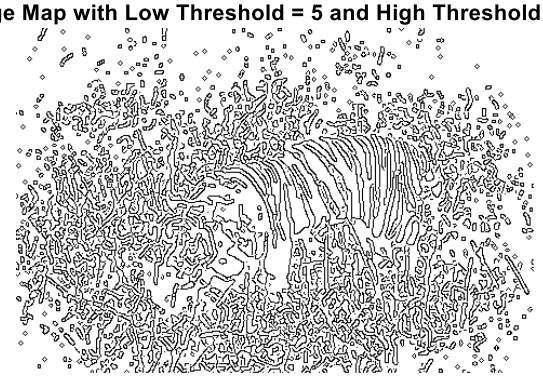
Edge Map with Low Threshold = 15 and High Threshold = 90



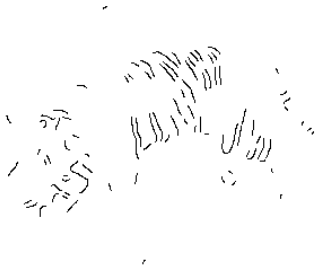
Edge Map with Low Threshold = 50 and High Threshold = 70

Case 3:

Edge Map with Low Threshold = 5 and High Threshold = 10

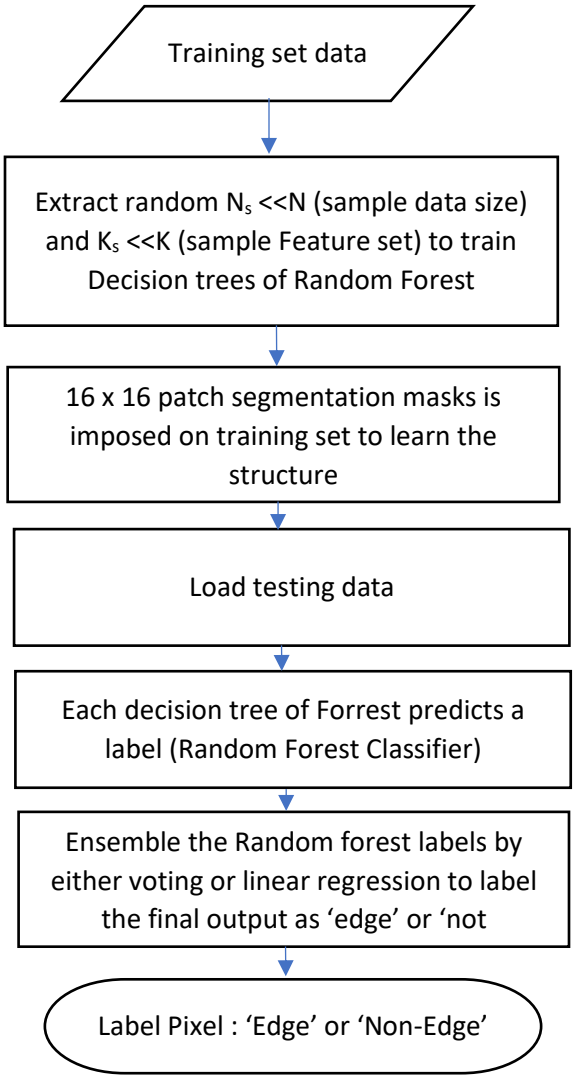


Edge Map with Low Threshold = 80 and High Threshold = 90



1.4.c Structured Edge

1.4.c.1 SE Detection Algorithm



Load the training set data, choose N_s << N : N is sample data size and K_s << K (feature set) to train a decision tree randomly so that the system is not overfitted by the classifier. Relevant features are extracted so that the Gini impurity or purity is high, so that classification are distinct. Each decision tree is trained similarly. Random

forest classifier is a set of all the decision trees. 16 x 16 patch is the segmentation mask which is trained over. Once the test data passes through each decision tree, it is assigned a label. Random forest classifier classifies the pixel as with label ‘edge’ or ‘not edge’ depending on voting(maximum decision tree label) or by linear regression.

1.4.c.2

The Random Forest (RF) classifier is used in the SE detector. The RF classifier consists of multiple decision trees and integrate the results of these decision trees into one final probability function. Construction of decision tree is as follows:

1. A decision tree ft(x) classifies a sample $x \epsilon X$ by recursively branching left or right down the tree until a leaf node is reached.
2. If $h(x, \theta j) = 0$ node j sends x left, otherwise right, with the process terminating at a leaf node.
3. There exists many features that can be chosen, but the feature which divides the branch with highest purity is considered.
4. The θ parameter of x data is compared against the standard Υ (theshold). If it is greater than the threshold, x data is classified to left node else the right node.
5. The classification continues until a maximum depth is reached or if the information gain or training set size fall below a fixed threshold.
6. Information gain is given by

   $Ij = H(Sj) - \sum_{k \epsilon \{L,R\}} \frac{|Sjk|}{|Sjl|} H(S_j^k)$

where $H(S_j) = - \sum_y p(1-\log(p))$ alternatively Gini Impurity is $H(S_j) = - \sum_y p(1-p)$

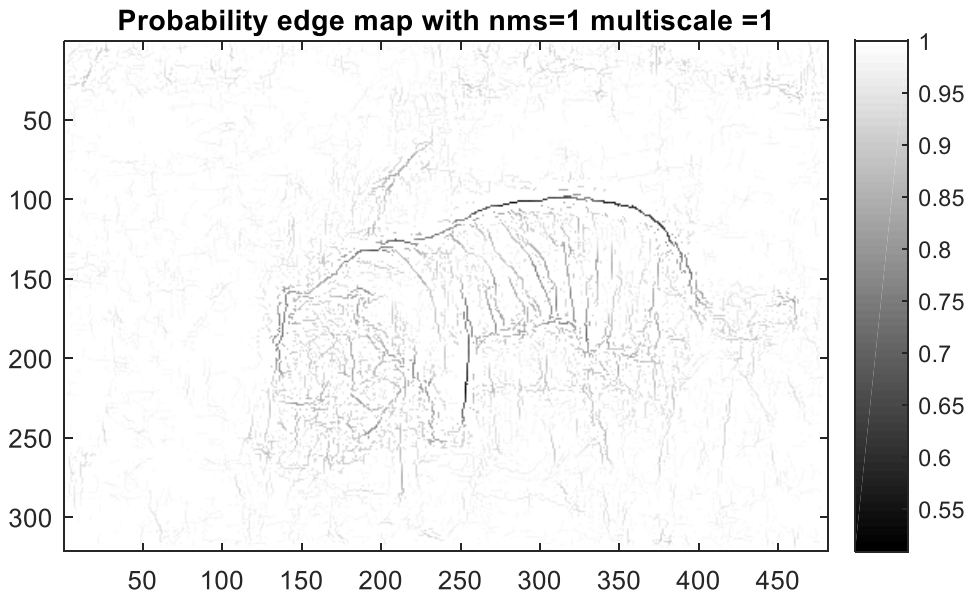
7. The output of the tree is the prediction stored at the leaf node with a target label $\acute{y} \epsilon \Upsilon$ for input x.
8. Each decision tree is trained by a random subset of data ($N_s <<< N$: Number of data samples and $\theta_s << \theta$: number of features). Thus randomness is inhibited to the system.

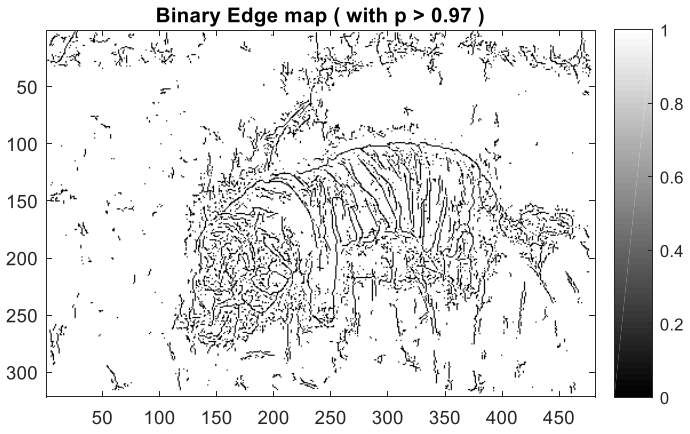
Prinicple of RF Classifier:

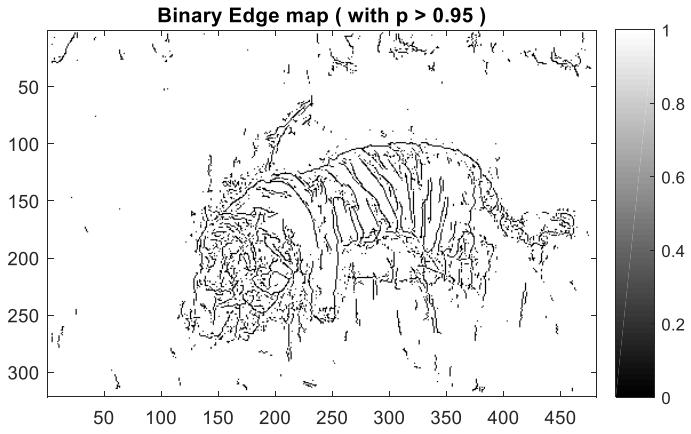
1. A decision forest is an ensemble of T independent trees ft.
2. Given a sample x, the predictions ft(x) from the set of trees are combined using an ensemble model into a single output.
3. Choice of ensemble model is problem specific and depends on Υ, common choices include majority voting for classification and averaging for regression, although more sophisticated ensemble models may be employed

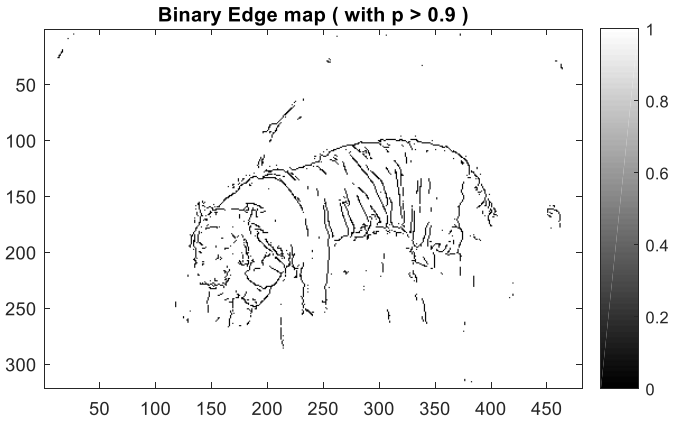
1.4.c.3

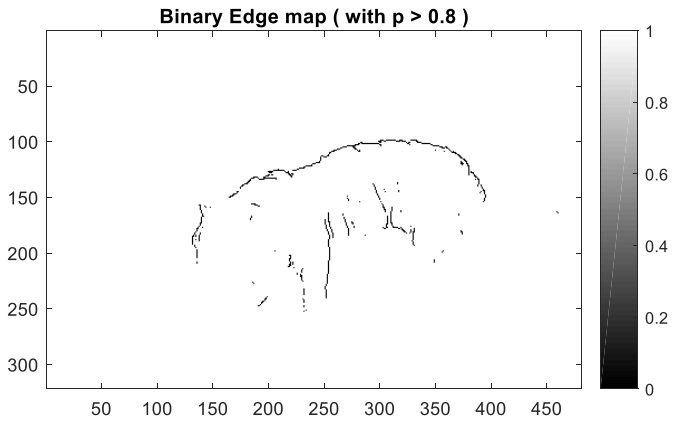
Following are the results when SE detector is applied to Tiger and Pig images by setting parameters like nms(non-maximum suppression) and threshold (convert probability edge map to binary map)

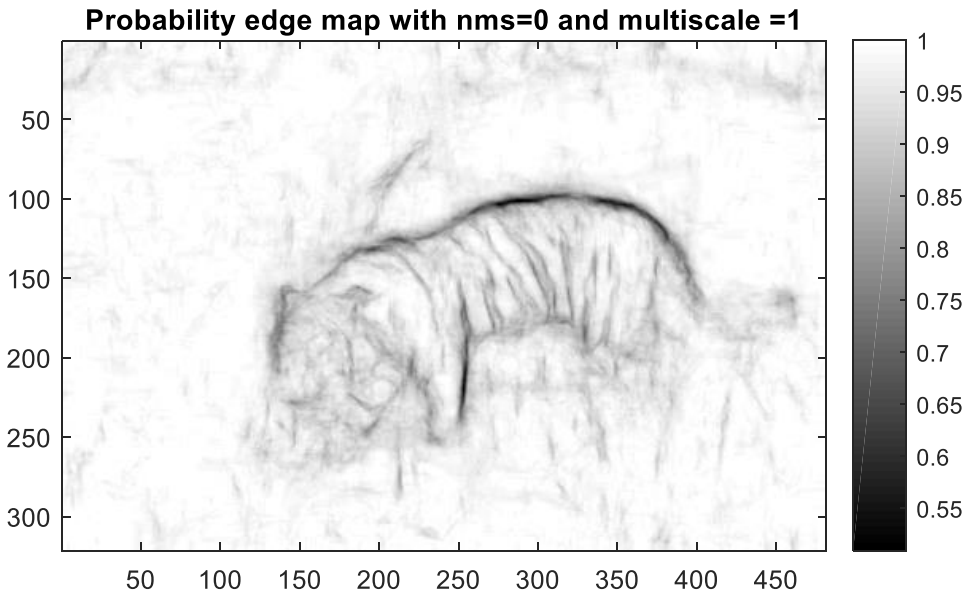

Probability edge map with nms=1 multiscale =1
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55

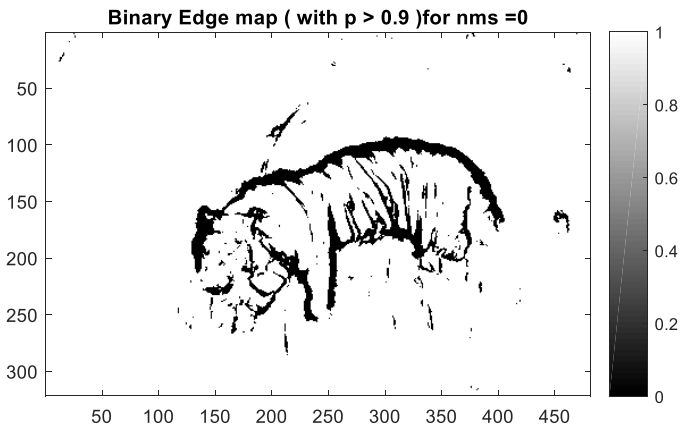

Binary Edge map (with p > 0.97)
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

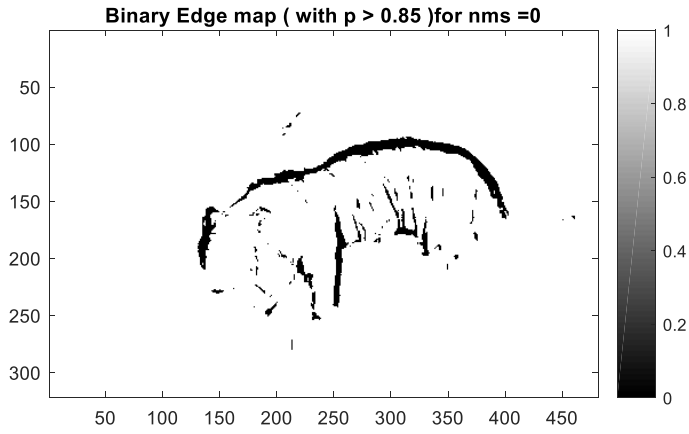

Binary Edge map (with p > 0.95)
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

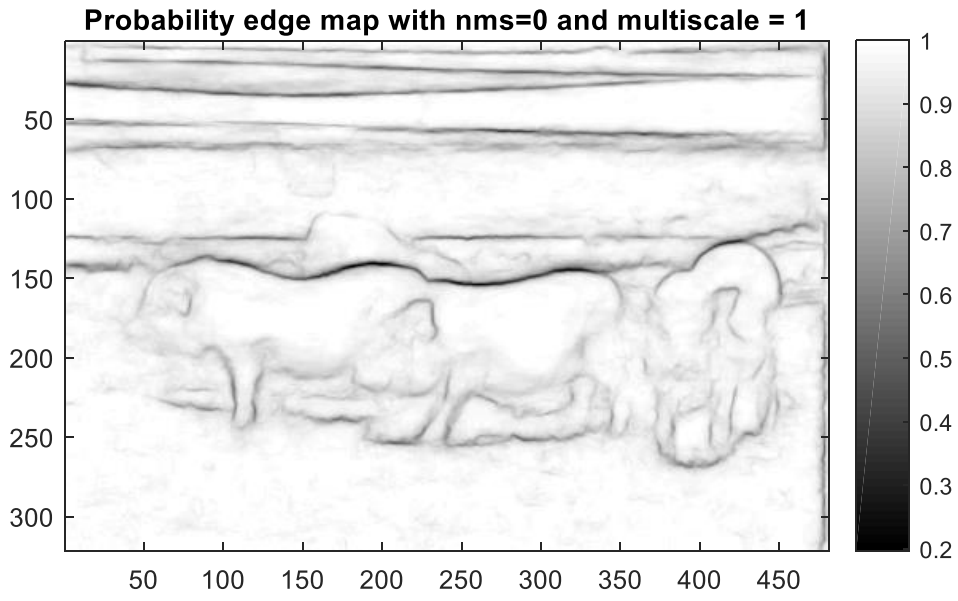

Binary Edge map (with p > 0.9)
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

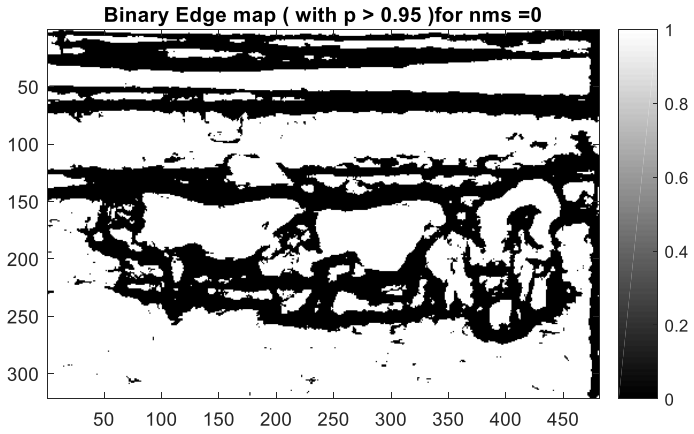

Binary Edge map (with p > 0.8)
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

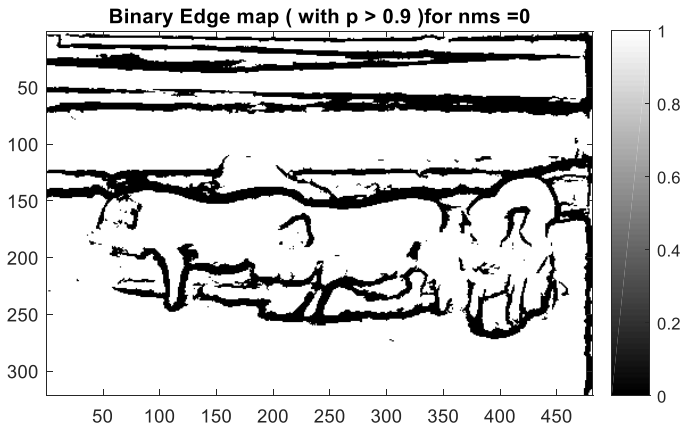

Probability edge map with nms=0 and multiscale =1
50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55

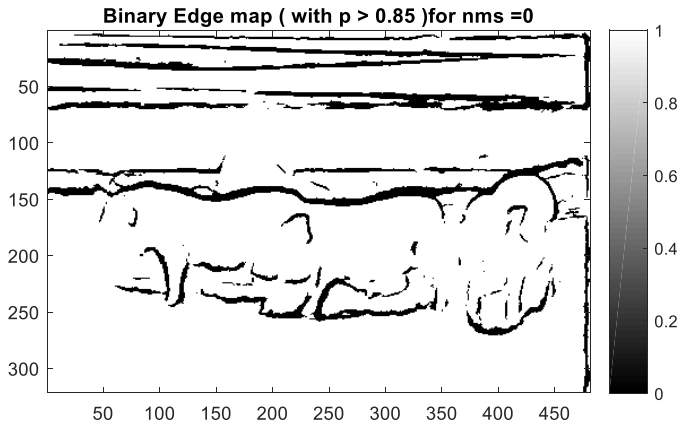

Binary Edge map (with p > 0.9)for nms =0
50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.8
0.6
0.4
0.2
0
Binary Edge map (with p > 0.95)for nms =0
50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.8
0.6
0.4
0.2
0

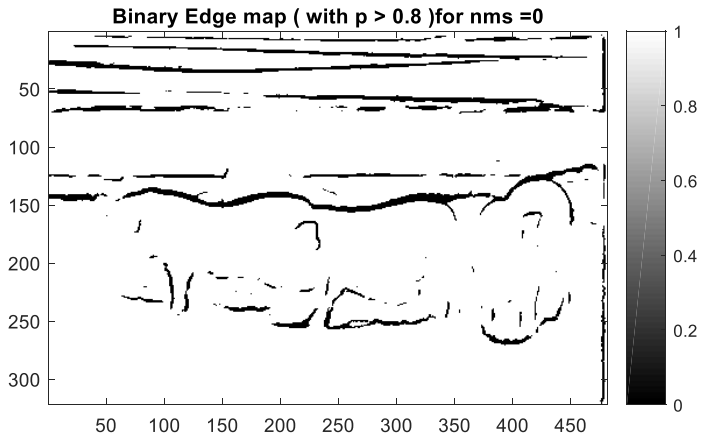

Binary Edge map (with p > 0.85)for nms =0
50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.8
0.6
0.4
0.2
0
Binary Edge map (with p > 0.85)for nms =0 with sharpen =1
50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.8
0.6
0.4
0.2
0

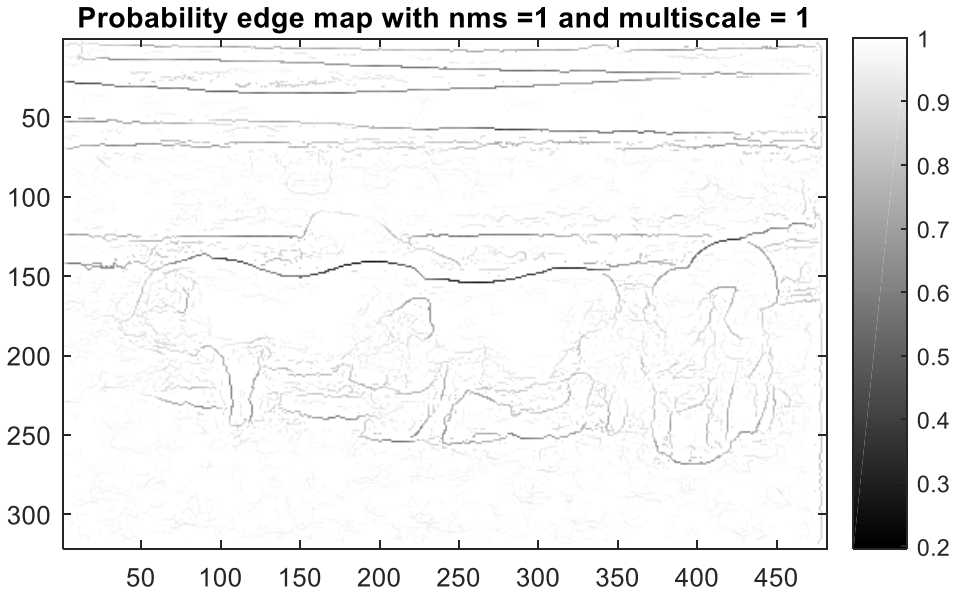

Probability edge map with nms=0 and multiscale = 1
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

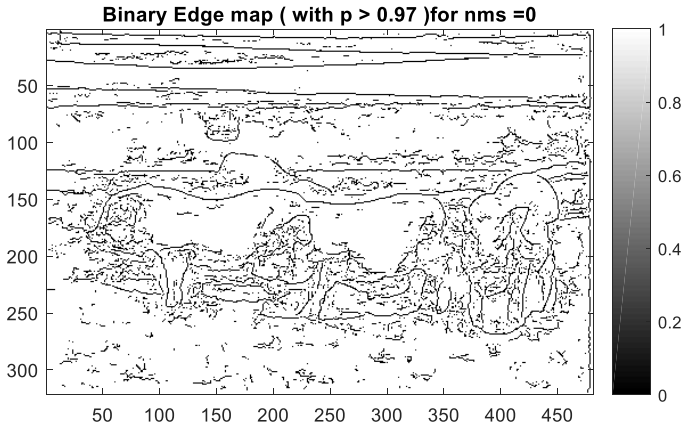

Binary Edge map (with p > 0.95)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

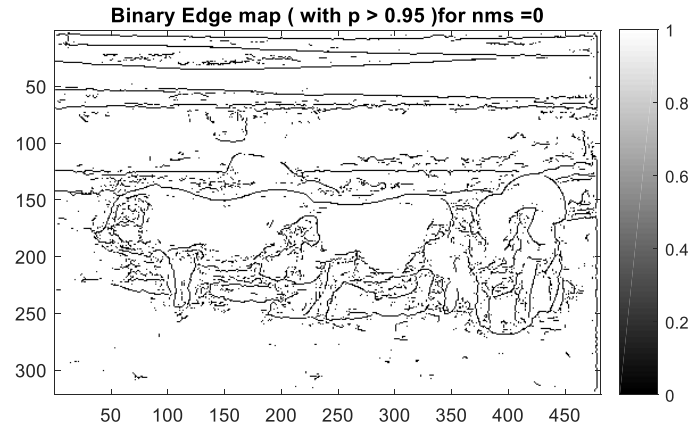

Binary Edge map (with p > 0.9)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

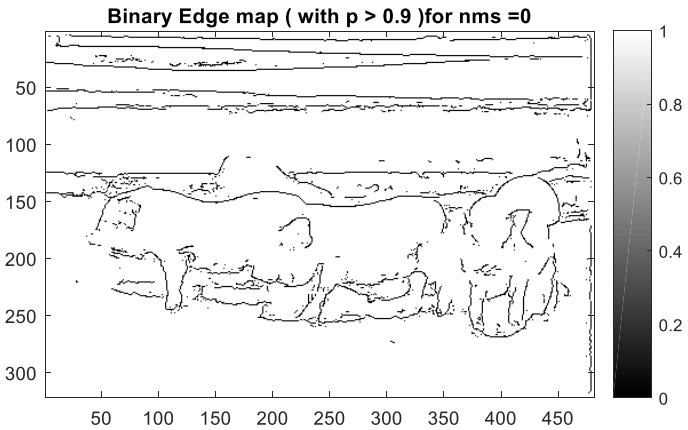

Binary Edge map (with p > 0.85)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0

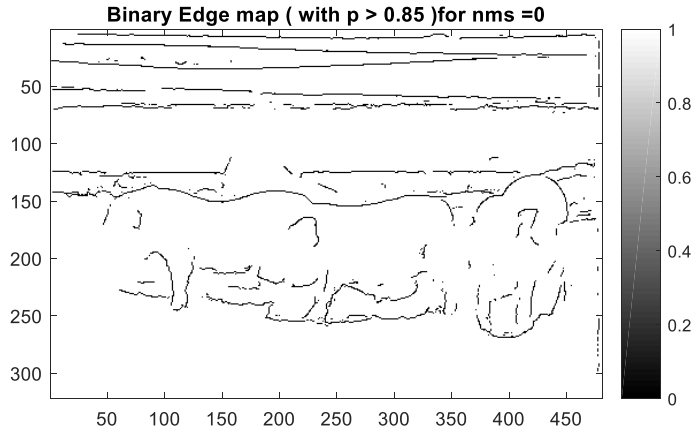

Binary Edge map (with p > 0.8)for nms =0
50
100
150
200
250
300
50
100
150
200
250
300
350
400
450
1
0.8
0.6
0.4
0.2
0











Edge detection parameters which are set for edgesEval() function are:

1.Multiscale: used for top accuracy test, if set to 1 : edgesDetect() function runs multiple times to output the edges.

2.Sharpen: Sharpens the image if set to 1

3.nTreesEval: used for top speed set, if set to 1: more number of threads are used for evaluation to speed up the process

4.nms: non maximum suppression is adopted if set to 1 for edge detection

5.Another parameter used to convert probability edge map to binary edge is threshold.

My choice/ selection of parameters are:
1. multiscale=1, so that the accuracy is increased.
2. sharpen =1, so that the edges are made sharp even if nms=0;
3. nTreesEval=1, to speed up the process
4. nms= 1, performs non-maximum suppression after the edge is detected, resulting in thin edges (at the local maximum point)
5. Threshold: Section 1.4.c.3 has detailed plots of images with different threshold conditions.
a. If nms=1 for tiger image, threshold of 0.95 is a better choice since it outlies the important edges.
b. If nms=0 for tiger image, threshold of 0.9 is a better choice since it outlies the important edges.
c. If nms=1 for pig image, threshold of 0.95 is a better choice since it outlies the important edges.
d. If nms=0 for pig image, threshold of 0.9 is a better choice since it outlies the important edges.

* In general, nms=0 has thicker edges compared to nms=1 for all the set of cases shown in section 1.4.c.3. nms =1 has thin edges and it performs better for edge detection.
* The threshold set varies from image to image depending on the end outcome, to what extent an edge has to be detected (main object only or inclusive of surrounding background)

Results of SE Detector and Canny detector
1. SE detector has finer output compared to Canny.
2. SE detector makes use of many features to arrive at a decision of 'edge' or not, while Canny uses only derivative of Gaussian to detect the edge crossing.
3. Canny detector uses two thresholds for the edge(contour) detection which has a trade off for compromising on too many edges or very less edge (kindly refer section 1.4.b case1,2 and 3 which results clumsy)
4. SE detector is more capable of identifying important object's edge in contrast to canny.
5. SE output can be parallelized while Canny approach cannot be.
6. SE performs better in contrast to Canny, since derivative of gaussian smooths the edges as well.

1.4.d

1.4.d.1

1.4.d.2

1.4.d.3

F measure is necessary to evaluate the edge detector performance in contrast to edge map provided by human (ground truth).

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

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

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

The two parameters precision and recall play a vital role.

No, it is not possible to get a high F measure if precision is significantly higher than recall or vice versa because making the precision higher by decreasing the threshold in deriving the binary edge map will result in a lower recall or versa. Thus, the multiplying factor is low, making F-measure low.

If Precision + Recall = constant

F= $\frac{2*P*(\text{constant - P})}{\text{constant}}$: the function maximizes when the equation is differentiated and set to 0.

P=constant-P=R

Thus P=R

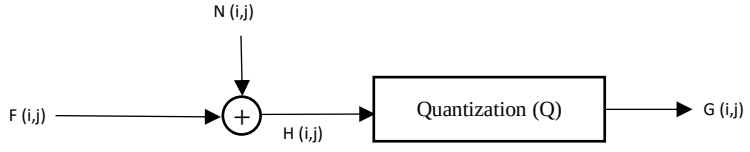
2.1 Abstract and Motivation

Digital Halftoning is a representation of an image with only black and white tones (2 tonnes). Halftoning is used for creating illusion of continuous tone output with a binary device, by making use of quantization. Human eye senses/performs spatial averaging. It is necessary for printing documents like pdf files, newspapers, books. Poor printing hardware capabilities require better halftoning algorithm. Hence the approximation of gray scale to black and white is essential. To convert a gray scale image to halftoned image, three methods are adopted: Random thresholding, Dithering and Error diffusion. To convert a color image to halftoned image, two techniques are used: Separable Error diffusion and Minimum Brightness Variation Quadruple.

2.2 Approach and Procedure

2.2.a. Dithering

Input image: F(i,j) Halftoned output image: G(i,j)



Block Diagram 1: Dithering Theoretical Model

If F(i,j) is fed directly to Q block with constant threshold, then the detailing of the image is missed even if the mean squared error is minimized. Instead we add noise to the F(i,j) before Quantization, to reduce monotonicity in the local varying region. Pink noise (low frequency noise), Blue noise are potential noises to be used as N(i,j). The noise acts like variance so that closer values to threshold are not reduced to 0(black).

2.2.a.1 Random thresholding

In Block Diagram 1, N(i,j) is a random number generator from uniform distribution in case of Random thresholding.
Step 1: Random noise (number) from uniform distribution is generated for each pixel.
Step 2: If the intensity of the pixel is greater than or equal to the random noise generated and less than 256, the pixel intensity is scaled to 255 else 0.

2.2.a.2 Dithering Matrix

In practise, we do not add noise to the image, but adopt adaptive threshold value based on the index matrix called dithering marix, used as a threshold for quantization (Q in Block diagram 1).

Step 1: Generation of nxn Index matrix (I) recursively using the below formula where n=1,2,3..

$$I_{2n}(i,j) = \begin{bmatrix} 4 * In(i,j) + 1 & 4 * In(i,j) + 2 \\ 4 * In(i,j) + 3 & 4 * In(i,j) \end{bmatrix} \quad \text{where} \quad I_0 = 0$$

Step 2: Generation of T (Threshold matrix) based on Index matrix, if the input image is not normalized:

$$T(x,y) = \frac{I2n(x,y) + 0.5}{n^2} * 255$$

If the input image is normalized, then the T (Threshold matrix) transforms as:

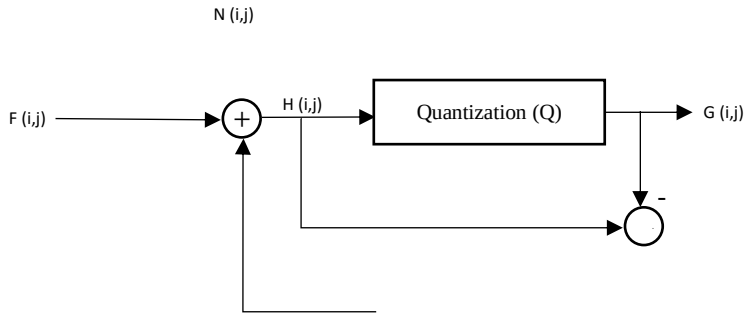
$$T(x,y) = \frac{I2n(x,y) + 0.5}{n^2}$$

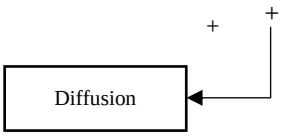
Step 3: Each pixel of an image F(i,j) are compared against the threshold value and scaled to 0 or 255, G(i,j) is halftoned matrix of image F(i,j)

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

2.2.b Error Diffusion

Input image: F(i,j) Halftoned output image: G(i,j)



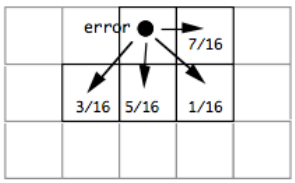


Block Diagram 2: Error Diffusion Model

Error diffusion model as shown in Block Diagram 2, inputs F(i,j) and outputs halftoned output G(i,j). Each time G(i,j) is computed, the error between the original F(i,j) and G(i,j) is evaluated and is diffused to neighbouring pixel. The mean square error is reduced, thus improving PSNR. The error can be diffused to neighbouring pixel in many ways, following are a few error diffusion matrix:

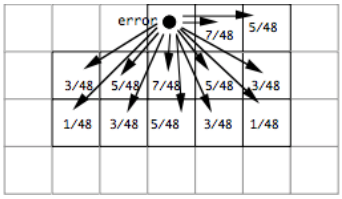
a. Floyd- Steinberg

$$\frac{1}{16}\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 7\\ 3 & 5 & 1\end{bmatrix}$$



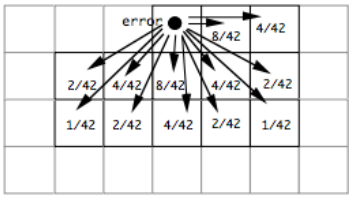
b. Jarvis, Judice, and Ninke (JJN)

$$\frac{1}{48}\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 7 & 5\\ 3 & 5 & 7 & 5 & 3\\ 1 & 3 & 5 & 3 & 1\end{bmatrix}$$



c. Stucki

$$\frac{1}{42}\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 8 & 4\\ 2 & 4 & 8 & 4 & 2\\ 1 & 2 & 4 & 2 & 1\end{bmatrix}$$



Algorithm for error diffusion using serpentine scanning is as follows:

Step 1: Let F(i,j) be the pixel intensity at i^{th} row and j^{th} column. For every (i,j), if H(i,j) is greater than a fixed threshold T (usually 127), G(i,j) is equal to 255 else 0.

Step 2: Computation of error is performed by evaluating H(i,j)- G(i,j)

Step 3: Depending on the scanning order, the diffusion matrix are mirrored accordingly.

Step 4: Serpentine parsing is adopted in this context. For every odd i^{th} row, the scanning is performed from left to right (j=1:1: width) by convolving the error diffusion matrix with zero padded boundary extended image or by alternating if-else conditions for j=1, 2, width-1, width column conditions and i=1, height row conditions for updating the neighbour pixel value with current fraction of error computed. Error spelled after the boundary of the image are omitted (not considered).

Step 5: For every even i^{th} row, the scanning is performed from right to left (j=width:-1:1)by convolving the mirrored error diffusion matrix with zero padded boundary extended image or by alternating if-else conditions for j=width, width -1, 2, 1 column conditions and i=1, height row conditions for updating the neighbour pixel value with current fraction of error computed. Error spelled after the boundary of the image are omitted (not considered).

Step 6: G(i,j) is the halftoned image using error diffusion. The same procedure is repeated for different error diffusion matrix.

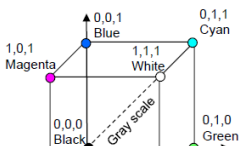
2.2.c Color Halftoning with Error Diffusion

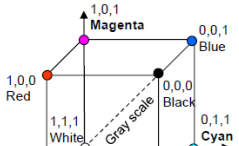
2.2.c.1 Separable Error Diffusion

To achieve color halftoning is to separate an image into CMY channels and apply the Floyd-Steinberg error diffusion algorithm to quantize each channel separately since CMY is used for printing.

Following algorithm is used:

Step 1: RGB channels of an original image is transformed into CMY channel using:





$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$ ⟶

RGB Cube CMY Cube

Step 2: Error diffusion model as shown in Block Diagram 2, inputs F(i,j) and outputs halftoned output G(i,j). Each time G(i,j) is computed, the error between the original F(i,j) and G(i,j) is evaluated and is diffused to neighbouring pixel. The error can be diffused to neighbouring pixel in many ways, following method is used for each individual channel of CMY separately and in serpentine parsing order.

a. Floyd- Steinberg

$$\frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

Step 3: Let F(i,j) be the pixel intensity at i^{th} row and j^{th} column of C channel of CMY model. For every (i,j), if H(i,j) is greater than a fixed threshold T (usually 127), G(i,j) is equal to 255 else 0.

Step 4: Computation of error is performed by evaluating H(i,j)- G(i,j)

Step 5: Depending on the scanning order, the diffusion matrix is mirrored accordingly.

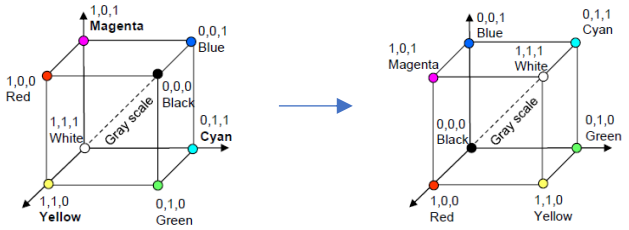
Step 6: Serpentine parsing is adopted in this context. For every odd i^{th} row, the scanning is performed from left to right (j=1:1: width) by convolving the error diffusion matrix with zero padded boundary extended image or by alternating if-else conditions for j=1, 2, width-1, width column conditions and i=1, height row conditions for updating the neighbour pixel value with current fraction of error computed. Error spelled after the boundary of the image are omitted (not considered).

Step 7: For every even i^{th} row, the scanning is performed from right to left (j=width:-1:1)by convolving the mirrored error diffusion matrix with zero padded boundary extended image or by alternating if-else conditions for j=width, width -1, 2, 1 column conditions and i=1, height row conditions for updating the neighbour pixel value with current fraction of error computed. Error spelled after the boundary of the image are omitted (not considered).

Step 8: G(i,j) is the halftoned image using error diffusion for a single C channel. The same procedure is repeated for M, Y channel separately from Step 3 to 8.

Step 9: Error diffused C, M, Y channel are transformed back to R, G, B channel respectively as

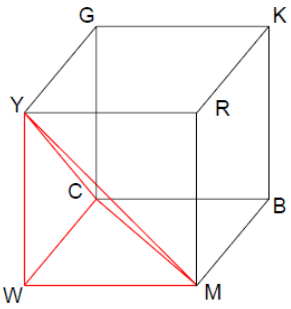
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$



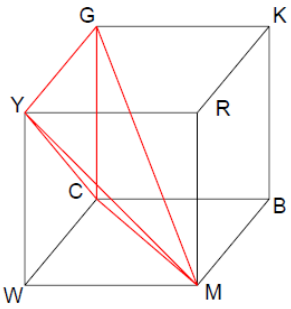
CMY Cube RGB Cube

2.2.c.2 MBVQ – based Error Diffusion

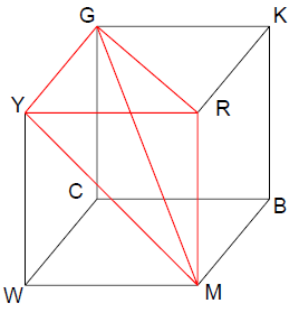
Vector Color error diffusion method partitions the CMYK color space into 6 Minimum Brightness Variation Quadruples as shown below.



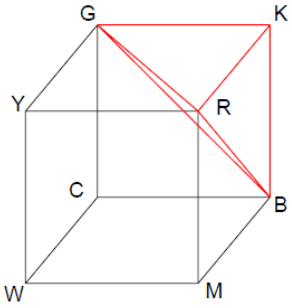
CMYW



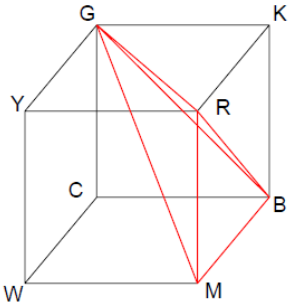
MYGC



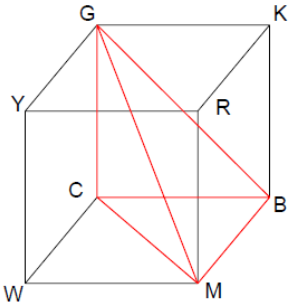
RGMY



KRGB



RGBM



CMGB

Algorithm for implementing MBVQ – based error diffusion:

Step 1: Initialize the quantization error of all pixel to zero.

Step 2: Error diffusion model as shown in Block Diagram 2, inputs RGB_F(i,j) and outputs halftoned output V(i,j) (Vector based on MBVQ Quantization). Each time V(i,j) is computed, the error between the RGB_H(i,j) and V(i,j) is evaluated and is diffused to neighbouring pixel. The error can be diffused to neighbouring pixel in many ways, following method is used in serpentine parsing order.

a. Floyd- Steinberg

$$\frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

Step 3: Find MBVQ tetrahedron that is closest to RGB(i,j), which limits the pixel into one quadrant among ‘CMYW’, ‘MYGC’, ‘RGMY’, ‘KRGB’, ‘RGBM’, ‘CMGB’ by performing:

```
if ((R(i,j)+G(i,j)) > 255)
   if ((G(i,j)+B(i,j)) > 255)
      if ((R(i,j)+G(i,j)+B(i,j)) > 510)
        return CMYW;
      else
        return MYGC;
   else
        return RGMY;
else
   if(!((G(i,j)+B(i,j)) > 255))
       if(!((R(i,j)+G(i,j)+B(i,j)) > 255))
            return KRGB;
       else
```

```
            return RGBM;
        else
            return CMGB;
```

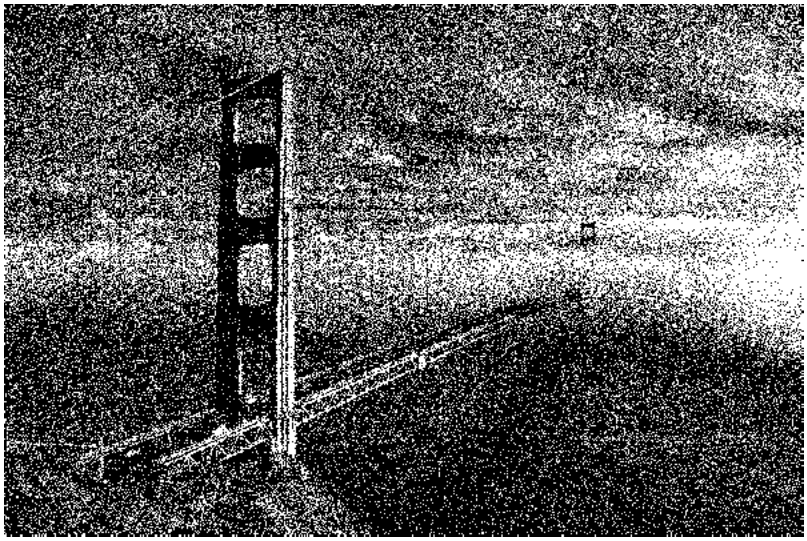
Step 4: Find Vertex, V, of MBVQ tetrahedron that is closest to RGB_H(i,j), which limits the pixel into one color among 8 colors ‘White’, ‘Black’, ‘Red’, ‘Green’, ‘Blue’, ‘Yellow’, ‘Cyan’, ‘Magenta’ using getNearestVertex() function.
Step 5: Compute the error RGB_H(i,j) – V(i,j)
Step 8: Depending on the scanning order, the diffusion matrix (a) is mirrored accordingly.
Step 6: Serpentine parsing is adopted in this context. For every odd i^{th} row, the scanning is performed from left to right (j=1:1: width) by convolving the error diffusion matrix with zero padded boundary extended image or by alternating if-else conditions for j=1, 2, width-1, width column conditions and i=1, height row conditions for updating the neighbour pixel value with current fraction of error computed. Error spelled after the boundary of the image are omitted (not considered).
Step 7: For every even i^{th} row, the scanning is performed from right to left (j=width:-1:1)by convolving the mirrored error diffusion matrix with zero padded boundary extended image or by alternating if-else conditions for j=width, width -1, 2, 1 column conditions and i=1, height row conditions for updating the neighbour pixel value with current fraction of error computed. Error spelled after the boundary of the image are omitted (not considered).
Step 8: V(i,j) is the halftoned image using MBVQ – based error diffusion.

2.3 Result

2.3.a Dithering

2.3.a.1 Random Thresholding

Dithering - Random Thresholding

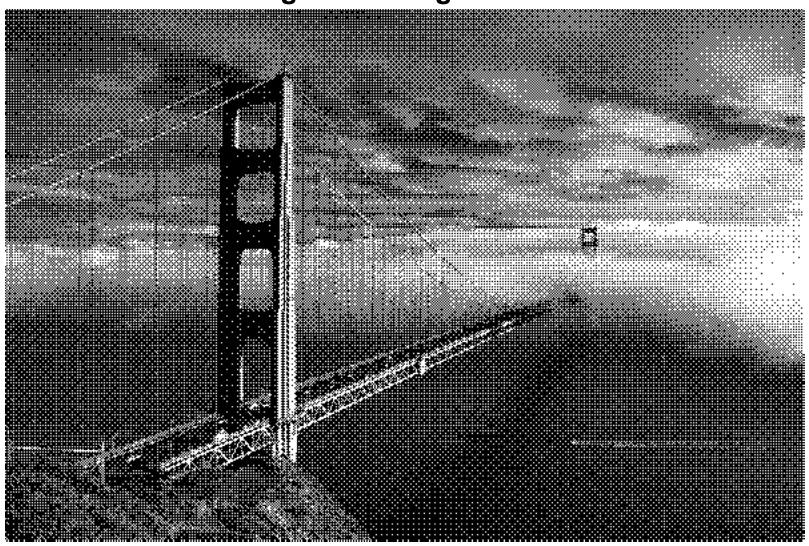


2.3.a.2 Dithering Matrix

Dithering - Dithering matrix 2x2

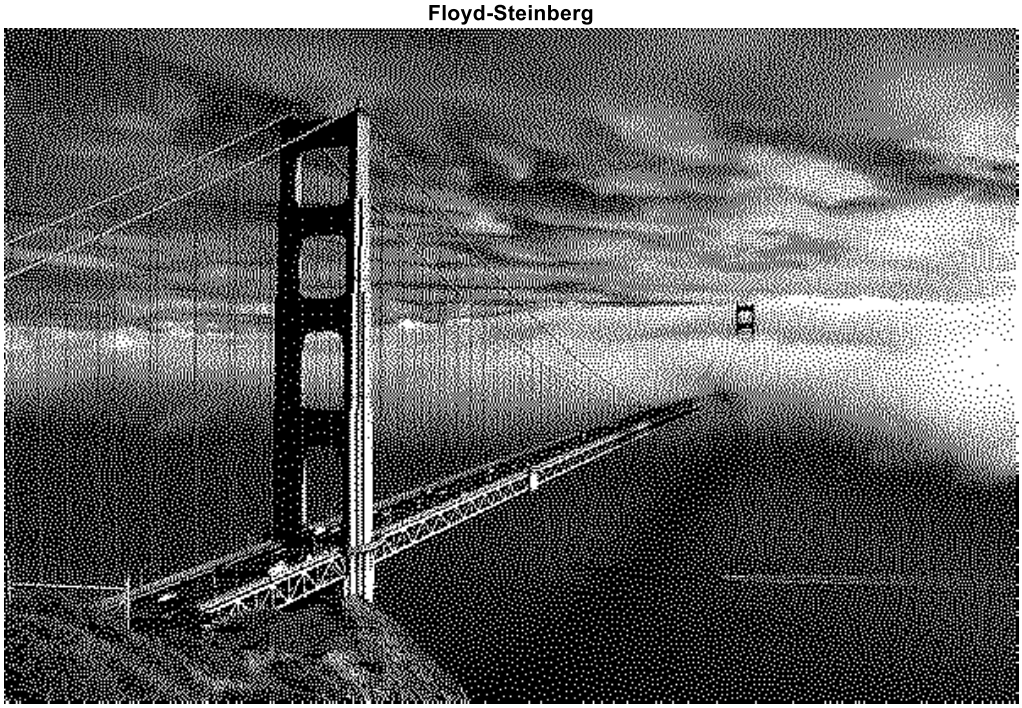
Dithering - Dithering matrix 8x8

Dithering - Dithering matrix 32x32



2.3.b Error Diffusion

Floyd-Steinberg



JJN

Stucki

2.3.c Color Halftoning with Error Diffusion

Original Image

Seperable Error Diffusion : CMY

Seperable Error Diffusion

MBVQ

Seperable Error Diffusion : Zoomed

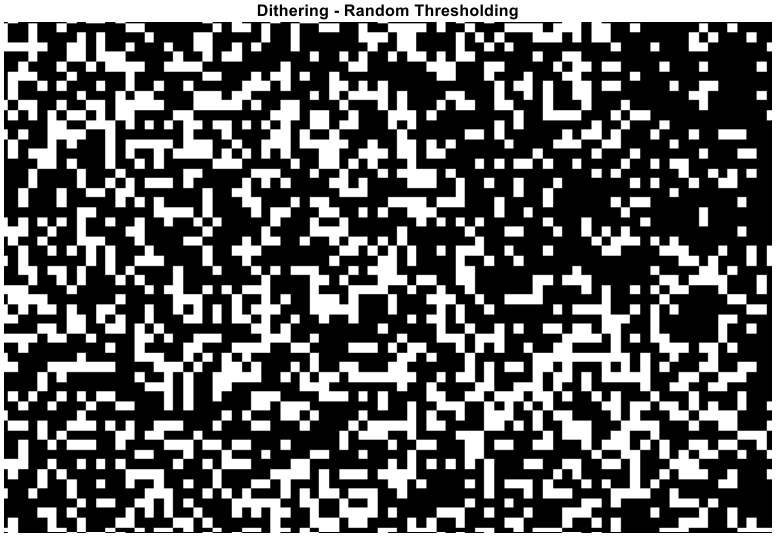
MBVQ : Zoomed

2.4. Discussion

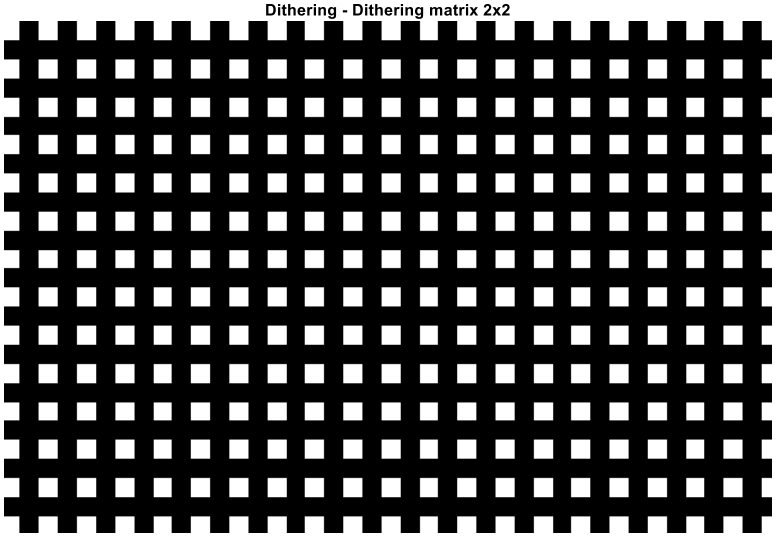
2.4.a, b Random Thresholding and Dithering Matrix

Artifacts/Patterns

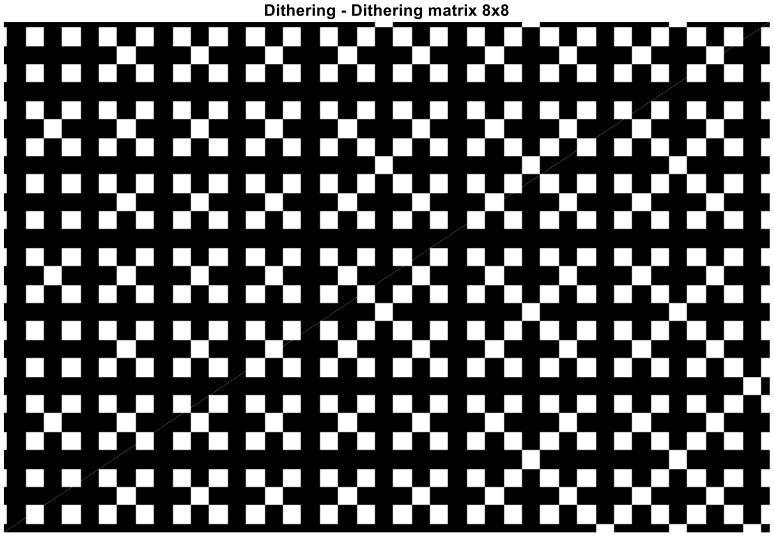
Dithering - Random Thresholding

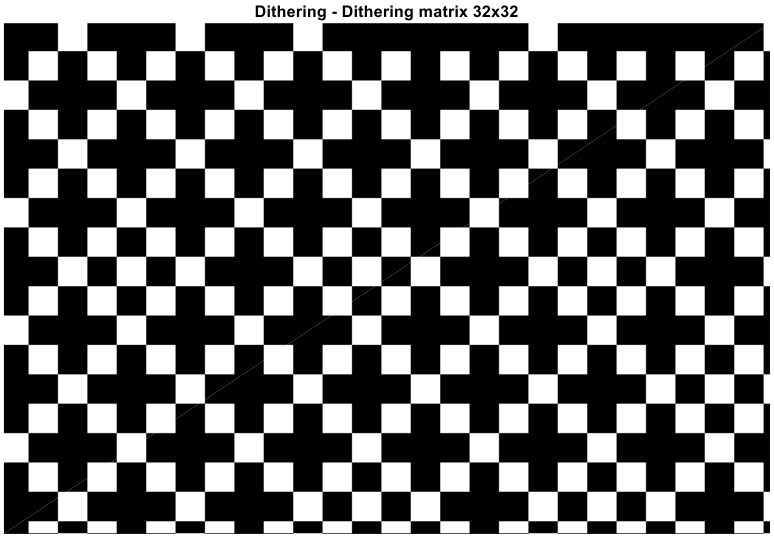


Dithering - Dithering matrix 2x2



Dithering - Dithering matrix 8x8





1. Random thresholding has an unpredictable output since random noise is the threshold. Dithering using index matrix render better quality than random thresholding but suffers from artifacts.
2. I_{2x2} , I_{8x8} , I_{32x32} results are in section 2.3.1. In case of I_{2x2} , the pattern keeps repeating more frequently compared to I_{8x8} , I_{32x32}. I_{8x8} has its pattern more frequently repeating compared to I_{32x32}.
3. PSNR is computed for random thresholding and all dithering matrices. It is observed that I_{32x32} has better performance comparatively.

Dithering method	PSNR (dB)
Random Thresholding	7.52073500141305
I_{2x2}	7.41080771827855
I_{8x8}	7.52849177450331
I_{32x32}	7.54287246545667

Table 1: Dithering method vs PSNR(dB)

4. Since the threshold matrix are large as n increases, the distance/gap between repeating pattern increases which can be observed as below, the highlighted pixel is more probable to be turned/switched on.

Threshold matrix for I_{2x2} :

95.625	159.375
223.125	31.875

Threshold matrix for I_{8x8}

85.66	149.41	101.60	165.35	89.65	153.40	105.59	169.34
213.16	21.91	229.10	37.85	217.15	25.90	233.09	41.84
117.54	181.29	69.73	133.48	121.52	185.27	73.71	137.46
245.04	53.79	197.23	5.98	249.02	57.77	201.21	9.96
93.63	157.38	109.57	173.32	81.68	145.43	97.62	161.37
221.13	29.88	237.07	45.82	209.18	17.93	225.12	33.87
125.51	189.26	77.70	141.45	113.55	177.30	65.74	129.49
253.01	61.76	205.20	13.95	241.05	49.80	193.24	1.99

Random Thresholding:
1. Introduces random noise which is better than a fixed/constant threshold
2. Includes more details of an image.

Dithering:
1. The method is computationally inexpensive since the threshold matrix obtained from the index matrix (dithered matrix) is simply compared against the corresponding pixel intensities.
2. The performance is poor since it generates regular/artificial pattern (as shown above).
4. Detailing of an image over a smaller area is retained.

2.4.b Error Diffusion

1. PSNR is computed for error diffusion using Floyd-Steinberg, Jarvis, Judice, and Ninke (JJN) and Stucki error diffusion matrix indicating performance is best for JNN, followed by Stucki and Floyd-Steinberg.

Error Diffusion method	PSNR (dB) for T=127
Floyd-Steinberg	7.63023861922587
Jarvis, Judice, and Ninke (JJN)	7.75386393510364
Stucki	7.72888762636471

Table 2: Error Diffusion method vs PSNR(dB)

2. Computational complexity of JNN and Stucki is higher than Floyd-Steinberg since the latter diffuses to only one neighbouring column and row instead of two columns and rows or 4 neighbouring pixel vs 12 neighbouring pixel.
3. More quantization error is placed along the diagonal due to serpentine scanning.
4. The constant threshold T, does not have a huge impact on the halftoned image (visually), since error keeps on diffusing to neighbouring pixels. Following is the table for comparison of PSNR of error diffusion matrices for different values of T.

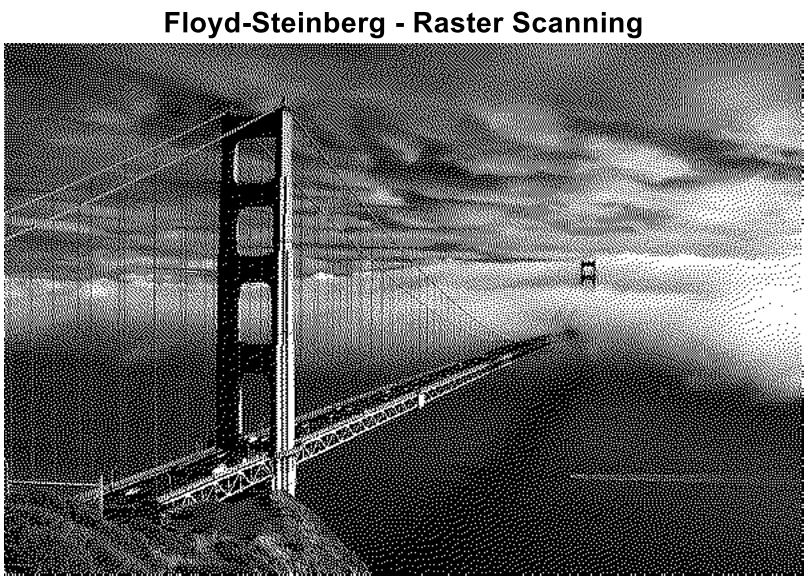
Error Diffusion method	PSNR (dB) for T=64	PSNR (dB) for T=192
Floyd-Steinberg	7.62483794287174	7.63291349884798
Jarvis, Judice, and Ninke (JJN)	7.74257738434762	7.76706833276362
Stucki	7.71830056335961	7.73660500713641

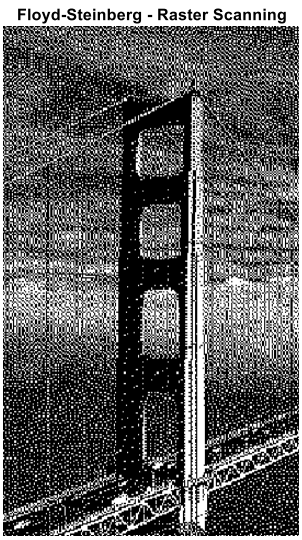
Table 3: Error Diffusion method vs PSNR(dB) for different T

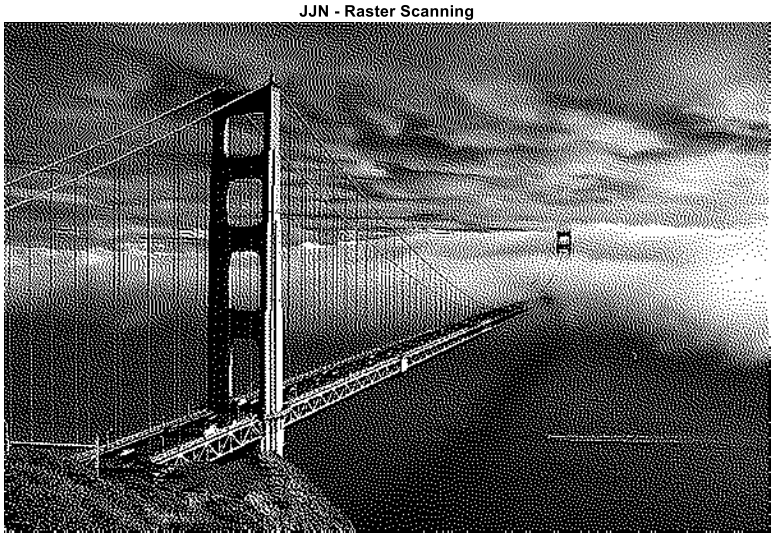
5. Scanning order is crucial to the results. Like wise following are the PSNR values for different error diffusion methods using Raster scanning.

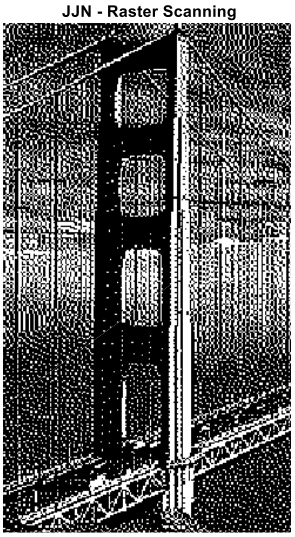
Error Diffusion method	PSNR (dB) for T=127
Floyd-Steinberg	7.62941834580996
Jarvis, Judice, and Ninke (JJN)	7.75168140695646
Stucki	7.72842283217553

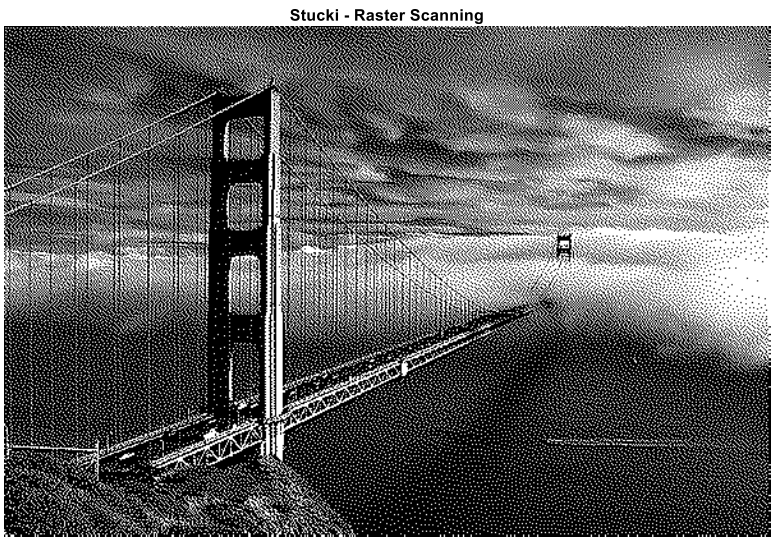
Table 4: Error Diffusion method vs PSNR(dB) for Raster Scanning

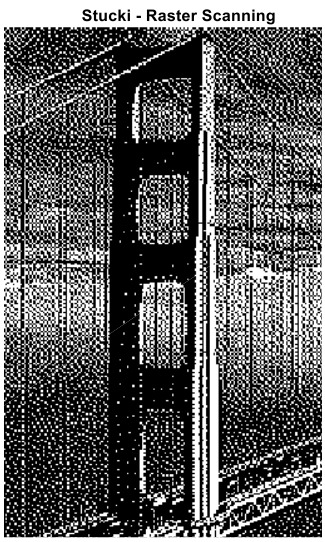












Clearly from the zoomed view, the error diffused towards right end of the boundary is visible for all the methods using raster scanning.
Serpentine parsing order prevents the apparition of artifacts in error diffusion algorithms which uses raster parsing order.

Comparison of Error Diffusion with dithering matrix:
1. Dithering matrix have regular patterns visible.
2. PSNR of Dithering is low.
3. Quality of halftoned image are better in case of error diffusion.
4. Computational complexity is high in case of error diffusion method.
5. Error diffusion leads to non-linear distortion, sharpening, additive noise.
6. Spatial correlation is made use in error diffusion while dithering uses intensity resolution.

I would prefer error diffusion in comparison with dithering because of the following reasons:
1. PSNR is high.
2. Artifacts/pattern are avoided.
3. Quality of the image rendered in more important than the complexity involved.
4. Detailing of an image is retained (thin lines of hanging bridge).
5. Spatial correlation is made use in error diffusion

To get better results: Proposed Idea -
Irrespective of serpentine/ raster parsing order, more error is diffused along the diagonal of any pixel, causing artifacts in both scanning orders with respect to the error diffusion matrix like Floyd, JJN, Stucki as highlighted below.

a. Floyd- Steinberg

$$\frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ \mathbf{3} & 5 & \mathbf{1} \end{bmatrix}$$

b. Jarvis, Judice, and Ninke (JJN)

$$\frac{1}{48}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & \mathbf{5} & 7 & \mathbf{5} & 3 \\ \mathbf{1} & 3 & 5 & 3 & \mathbf{1} \end{bmatrix}$$

c. Stucki

$$\frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & \mathbf{4} & 8 & \mathbf{4} & 2 \\ \mathbf{1} & 2 & 4 & 2 & \mathbf{1} \end{bmatrix}$$

Instead less weightage has to be given towards the disbursement of error diagonally. This may eventually create less artifacts along the diagonal.

2.4.c Color Halftoning with Error Diffusion

2.4.c.1 Separable Error Diffusion

Shortcoming of this approach are as follows:
1. Image quality is poor (visible artifacts)
2. Computational complexity is more
3. Additional memory is required
4. Run time is high
5. Color dots are not equally bright
6. Does not make use of spatial domain correlation among three planes (RGB)
7. Halftone noise is high

2.4.c.2 MBVQ based Error Diffusion

Shortcoming of Separable Error diffusion is overcome by:

1. Visible artifacts are due to variation in the brightness of dots, MBVQ minimizes the variation in brightness of the dots.
2. Color dots are made equally bright by having minimum brightness variation.
3. Quality of the image is high.
4. Halftone noise is minimized since only 4 colors are used (any color can be rendered using no more than 4 colors).
5. Spatial domain correlation is made use, because all red, green and blue pixel intensities at the same / different location are considered together to determine MBVQ quadruple and vertex.
6. Run time is less
7. No additional memory required

Comparison of Separable Error Diffusion and MBVQ – based Error Diffusion

1. Visible artifacts of Seperable error diffusion is minimized in MBVQ
2. Quality of image in case of MBVQ is better than Seperable Error Diffusion.
3. Results look similar, but MBVQ preserves color and brightness.
4. The texture near the head of the bird has less difference in brightness in case of MBVQ, while Seperable error diffusion method has significant difference due to black dots in the blue region.
5. The placement is visually unnoticeable in MBVQ and the local average color is the desired color, while it is false for Seperable error diffusion method (Individual plane images are included to differentiate the placement pattern)

Seperable Error Diffusion : Red plane

MBVQ - Red plane

Seperable Error Diffusion : Green plane

MBVQ - Green plane

Seperable Error Diffusion : Blue plane

MBVQ - Blue plane

Seperable Error Diffusion : Zoomed

MBVQ : Zoomed

Reference: Wikipedia:canny