EE569: Introduction to Digital Image Processing

ASSIGNMENT 2

Edge Detection
Digital Half-toning

Suchismita Sahu | 12th Feb 2019 | USCID: 7688176370

PROBLEM 1: EDGE DETECTION

ABSTRACT AND MOTIVATION

Edge detection and contour detection are fundamental operations in Image Processing pipeline. They are considered low-level Computer Vision operations. Edges are pixel-based features that is characterized by sudden change in intensity of the image pixels and hence the frequency domain of the Image. Edges are high frequency features. We can use various filtering techniques, as well as data driven approaches for implementing edge detection.

Here, we implement Sobel Edge Detector, Canny Edge Detector and Structured Edge Detector and compare their performance evaluations.

SOBEL EDGE DETECTOR

Sobel Mask is one of the basic edge detection masks. It performs a 2D gradient measurement of the spatial domain of an image. Masking operation with Sobel mask emphasizes the regions of high spatial frequency in the Image, thus detecting the edges. It finds the absolute gradient magnitude of each pixel of an input grayscale image.

Sobel operators are a pair of 3*3 kernels, the other rotated by 90 degrees.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 1 Sobel Kernels for X-gradient and Y-gradient

The X-gradient detects the frequency increase in the horizontal direction whereas Y-gradient detects the same in the vertical direction. After finding the X- and Y-

gradients respectively, we combine the gradient values to find the absolute gradient magnitude at each pixel as given by the following equation:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

The Angle of the absolute gradient is given by:

$$\theta = \arctan(Gy/Gx)$$

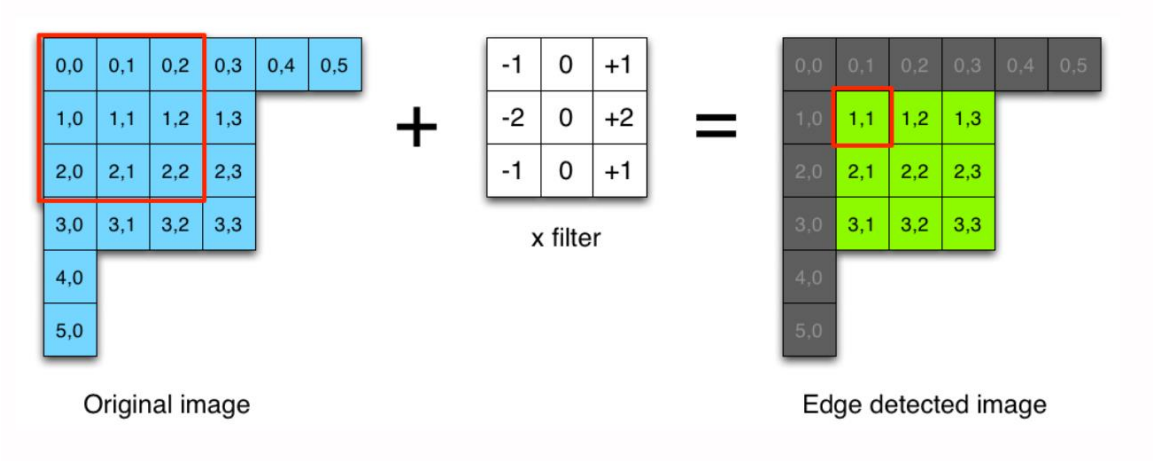


Figure 2 Overview of Sobel Edge Detection Operation

CANNY EDGE DETECTOR

This technique uses the intensity gradients of the image and implements non-maximum suppression with double thresholding or hysteresis thresholding. It can model the edges to be sharp changes in pixel brightness when moving from one pixel to another. It adds an additional layer of intelligence on the basic Sobel detector. Canny edge detection finds all the edges at minimal error rate that means the detection should accurately catch as many edges shown in image as possible. This algorithm localizes edge points such that the distance between the edges and center of the true edge is minimum. Thus, Canny Edge detection involves simple computations and provides more reliability.

The main steps of Canny Edge Detection are as follows:

1. Apply a Gaussian filter to smoothen the image and remove noise.
2. Calculate the intensity gradients of the image Gx and Gy.
3. Apply non-maximum suppression (NMS) to get rid of non-edges
4. Apply double threshold to determine potential edges

5. Track edge by hysteresis: Finalize the edges by suppressing all the other edges that are weak and not connected to neighboring strong edges.

NON-MAXIMUM SUPPRESSION:

It is an edge thinning technique that is applied to find the largest or longest edge in an image.

Basically, NMS suppresses all the gradient values (i.e. sets them to zero) except the local maxima that eventually indicate the regions with sharpest change in intensity values.

The algorithm of NMS can be described as:

1. Compare edge strengths of a current pixel with pixels in its positive and negative gradient directions
2. If edge strength of the current pixel is largest in one direction, it is saved, otherwise it is suppressed.

For implementation, Linear interpolation is used between the two adjacent pixels in the same direction. The gradient magnitude at center pixel must be the highest among both gradient directions to be marked as an edge. So, only thin lines remain as edges.

LOW AND HIGH THRESHOLDING or DOUBLE(HYSTERESIS) THRESHOLDING:

After NMS, the edges in the image are somewhat accurate but some spurious edges remain that are caused by noise and color variations or image texture. Hence, we need to filter out weak gradient edge pixels.

This is done by selecting two thresholds, high and low. Recommended 2:1 or 3:1.

Thresholding is done as:

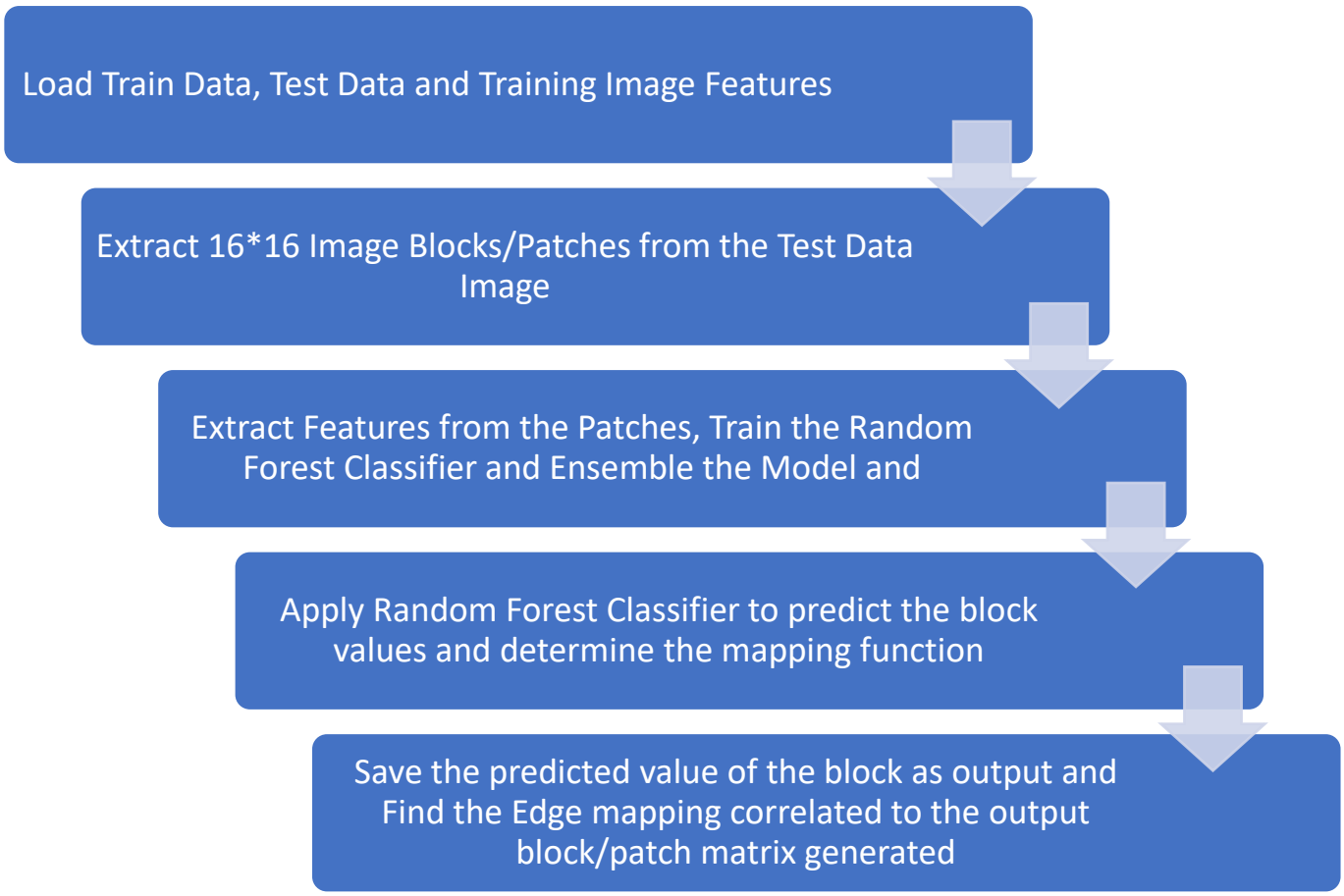
1. If an edge pixel's gradient value is greater than high threshold – Strong edge pixel
2. If an edge pixel's gradient value is less than low threshold – Weak edge pixel
3. If an edge pixel's gradient value is between high and low thresholds – Termed as weak or strong pixel depending on it's neighboring pixels' gradient strengths i.e. accepted as strong pixel if connected to a strong pixel.

STRUCTURED EDGE

Since Edge detection is an integral part of Image Processing pipeline and hence computer vision techniques, we need better edge detectors for application in applications like object recognition, etc. Structured edge is a data driven state of the art technique for edge detection that outperforms the traditional LoG, Sobel and Canny Detectors. It is used for finely distinguishing between object and background.

The method is based blocks of image, rather than pixel by pixel edge detection. For every block/patch of the image, the structured edge algorithm takes into consideration, the structural advantages i.e. the likelihood of pixels present in the block being an edge pixel or not to learn about the overall behavior of the image and hence detect the edges accurately. Further, each of these blocks are classified into sketch tokens implementing Random Forest Classifier.

Structured Edge Algorithm can be explained as follows:



PRINCIPLE OF RANDOM FOREST CLASSIFIER AND DECISION TREE CONSTRUCTION

A random forest classifier works on the principles of Divide and Rule. This is what improves their performance. The central idea here is to ensemble a set of weak learners to constituently for a strong learner. The basic element of a random forest is a decision tree which is a weak learner.

A decision tree has complex input and output spaces i.e. histograms and sketch maps. It classifies an input belonging to space A as an output that belongs to space B. Firstly, the input enters the root node and then traverses down the tree in a recursive manner till it reaches the leaf node. Every node in a decision tree is composed of a binary split function (which can be complex as well) that is defined by some parameters that help decide whether the input should progress towards the right of the node or towards the left of the node i.e. right child node or left child node.

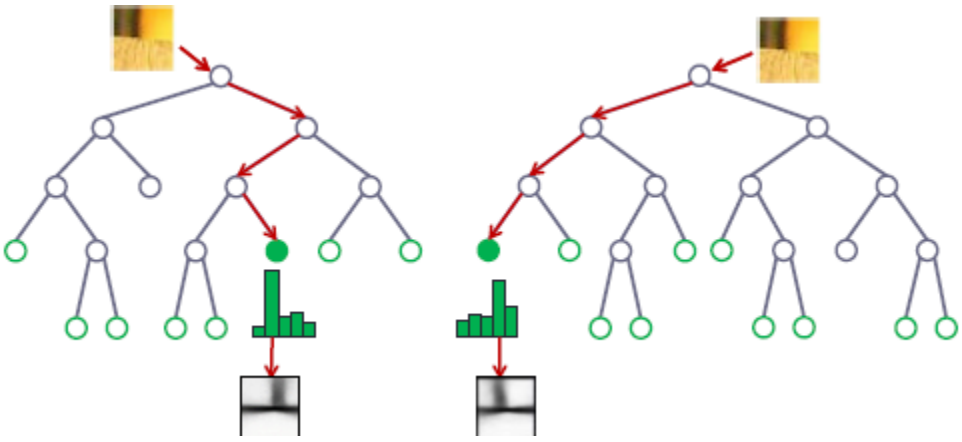


Figure 3 Depiction of Decision Trees

A large set of individual decision trees are trained independently to evaluate the parameters of complex split function to ensure good splitting of data to maximize information gain of the classifier.

Random Forest has a high ability to select effective features and is not sensitive to the feature normalization. We need to have good enough diversity of trees to ensure no-overfitting. Random Forests are white box model and mirror human decision making very closely.

PERFORMANCE EVALUATION

We use Precision, Recall and F-measure metrics to evaluate the accuracy and performance of the different edge detectors with respect to the ground truth images that are human labeled.

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

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

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}.$$

Where,

True Positive = Edge pixels in the edge output that coincide with the ground truth pixels.
False Positive = Pixels in the edge output that coincide with the non-edge pixels ground truth pixels.
True Negative = Non-Edge pixels in the edge output that coincide with the non-edge pixels ground truth pixels.
False Negative = Non-Edge pixels in the edge output that coincide with the edge pixels ground truth

APPROACH AND PROCEDURE

Algorithm Implemented for Sobel Edge Detector:

1. Read Input Image and convert Image to gray-level from RGB.
2. Initialize the two Sobel Kernels Gx and Gy.
3. Calculate the X-gradient and Y-gradient of the Image by implementing correlation with Gx Sobel Kernel and Gy Sobel Kernel respectively and Normalize Gx and Gy to value between (0-255).
4. Find the absolute gradient magnitude mapping from the X- and Y- gradients.
5. Find the CDF histogram of the magnitude pixels to find threshold and using this threshold, binarize the end output edge-detected image.

6. Save the interpolated data into 3D new image data created.

<u>Algorithm Implemented for Canny Edge Detector:</u> (in Matlab) – Done Two ways

1. Using Predefined Matlab function "edge".
    a. Read the Image and do rgb2gray to convert it into gray scale image.
    b. Use function out = edge(Image, 'canny', 0.1) where 0.1 is the high threshold. Did this for many thresholds to find best edge map.
    c. Matlab function Edge calculates the lower threshold as 0.4*threshold by itself and gives the edge detection Output.
2. Coding a Matlab Function for Canny Edge Detection following all steps of the algorithm:

    1. Apply a Gaussian filter to smoothen the image and remove noise.
    2. Calculate the intensity gradients of the image Gx and Gy.
    3. Apply non-maximum suppression (NMS) to get rid of non-edges
    4. Apply double threshold to determine potential edges

    5. Track edge by hysteresis: Finalize the edges by suppressing all the other edges that are weak and not connected to neighboring strong edges.

    In this method, we can find edge maps for varying high and low thresholds combinations and incorporated code for calculating probability edge map for use in performance evaluation as well.

    Have pasted outputs for canny detection using the first method and have used the output from second method for evaluation.

<u>Structured Edge Algorithm</u>:

Online available code is used for performing structured edge detection as asked in the question from https://github.com/pdollar/edges.

From the list of files, edgesDemo.m file was used to perform Structured Edge Detection.

For Performance Evaluation:

Implemented pDollars toolbox in Matlab. Function used :

[thrs,cntR,sumR,cntP,sump,V] = edgesEvalImg(BinaryMap, ground_truths, varagin);

Where,

thrs = threshold

Precision and Recall are calculated as:

Precision = cntP/sump

Recall = cntR/sumR

F-score = 2PR/P+R

EXPERIMENTAL RESULTS

SOBEL EDGE DETECTOR

Figure 4 Original Tiger Image

Figure 5 RGB to Gray Scale Converted Tiger Image

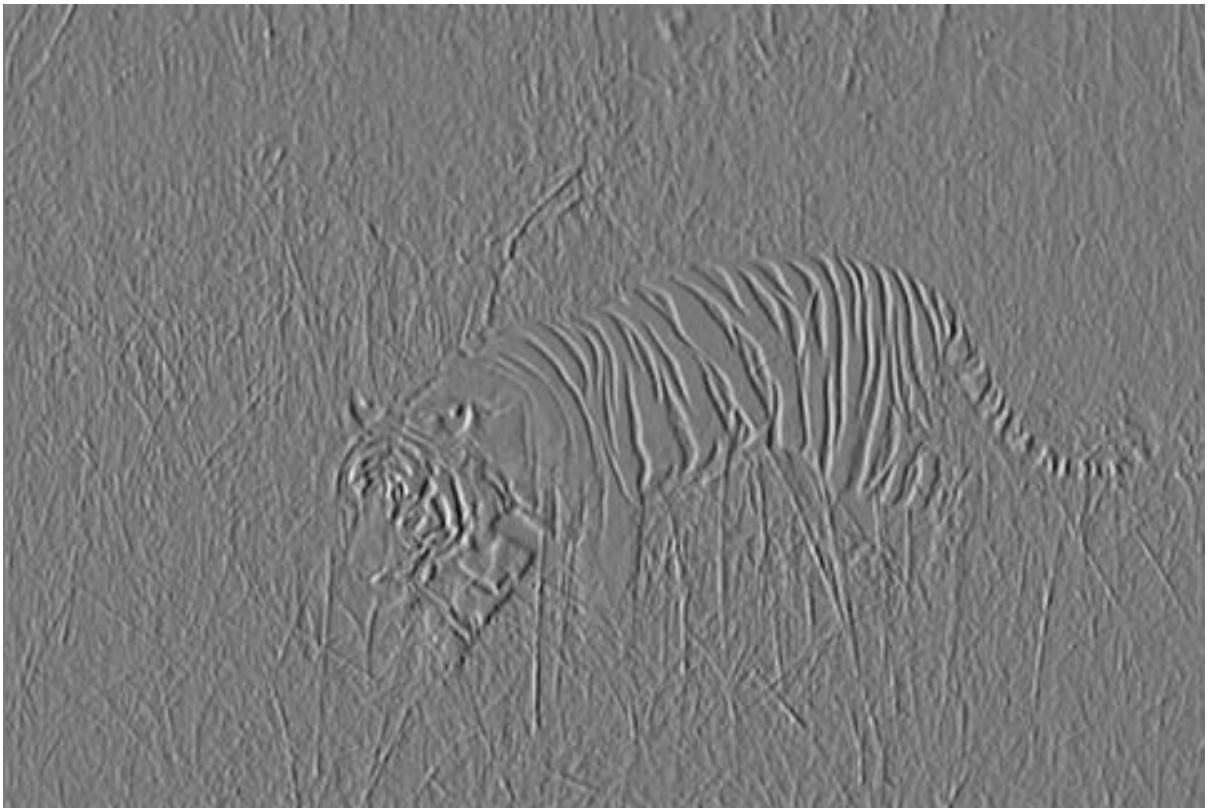


Figure 6 Tiger Image Normalized X-Gradient Using Sobel X-Mask

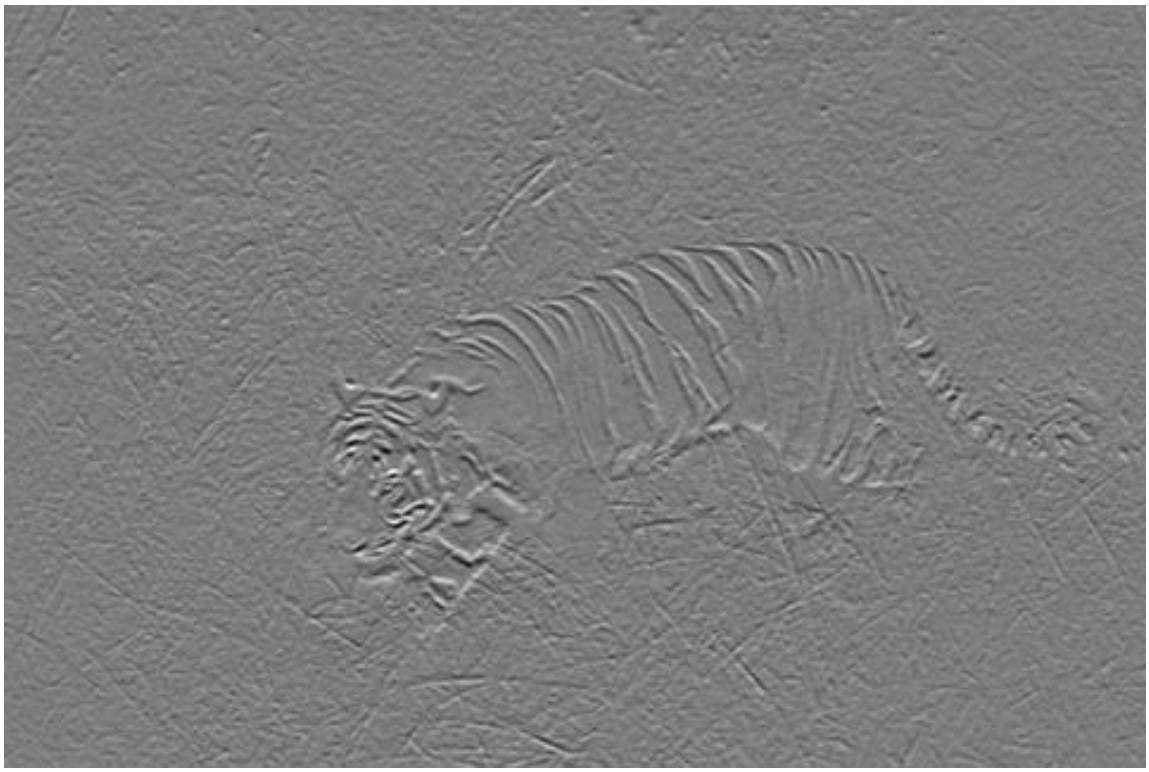


Figure 7 Tiger Image Normalized Y-Gradient Image Using Sobel Y-Mask

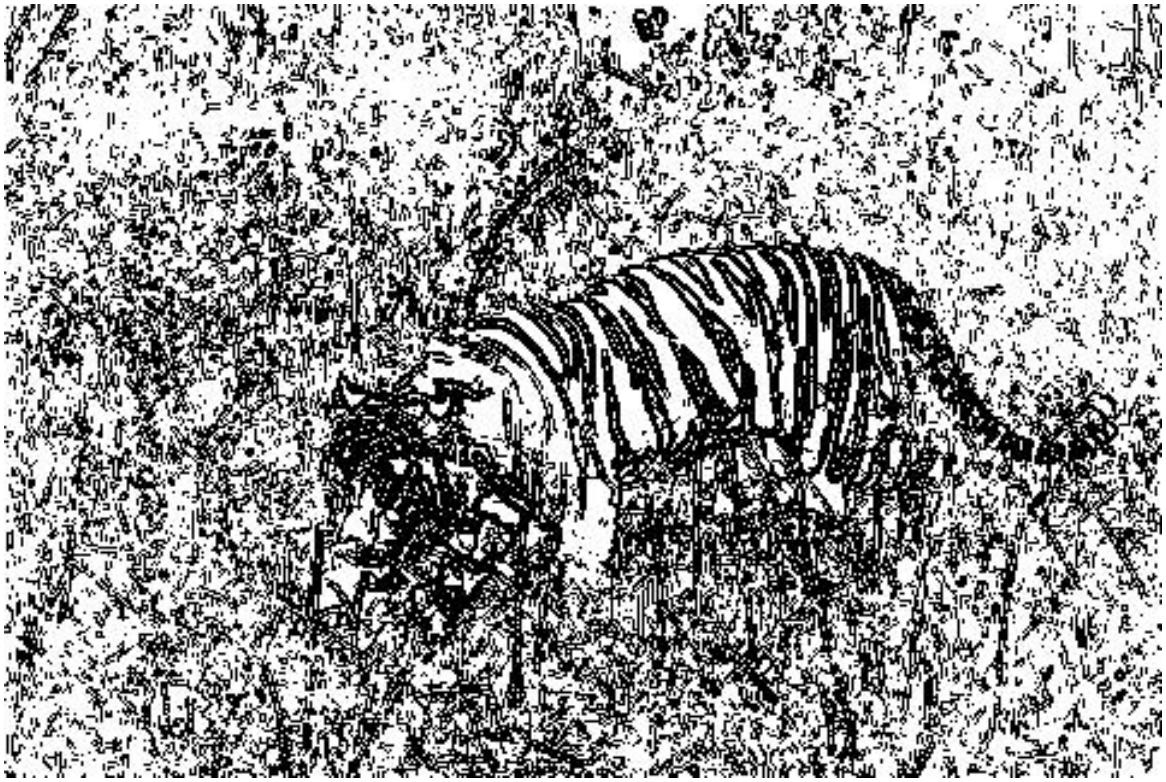


Figure 8 Tiger Edge Output for 60% Threshold

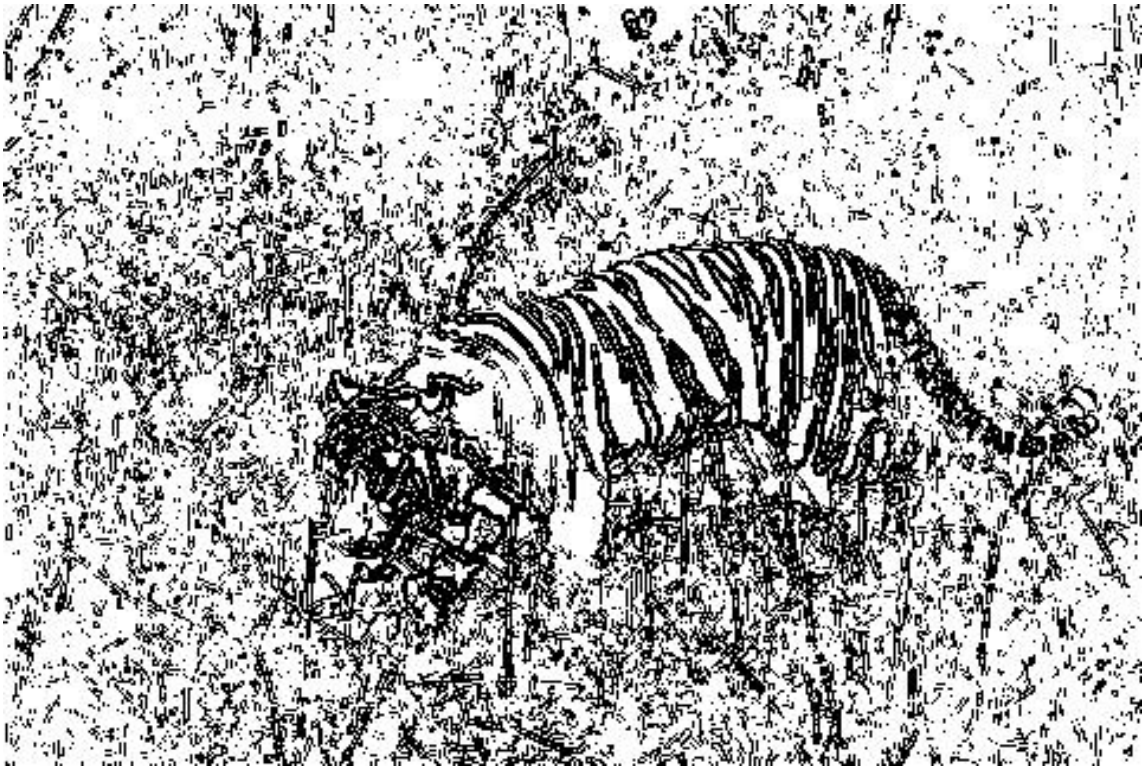


Figure 9 Tiger Edge Output for 70% Threshold

Figure 10 Tiger Edge Output for 85% Threshold

Figure 11 Tiger Edge Output for 90% Threshold

Figure 12 Original Pig Image

Figure 13 RGB to Gray Scale Converted Pig Image

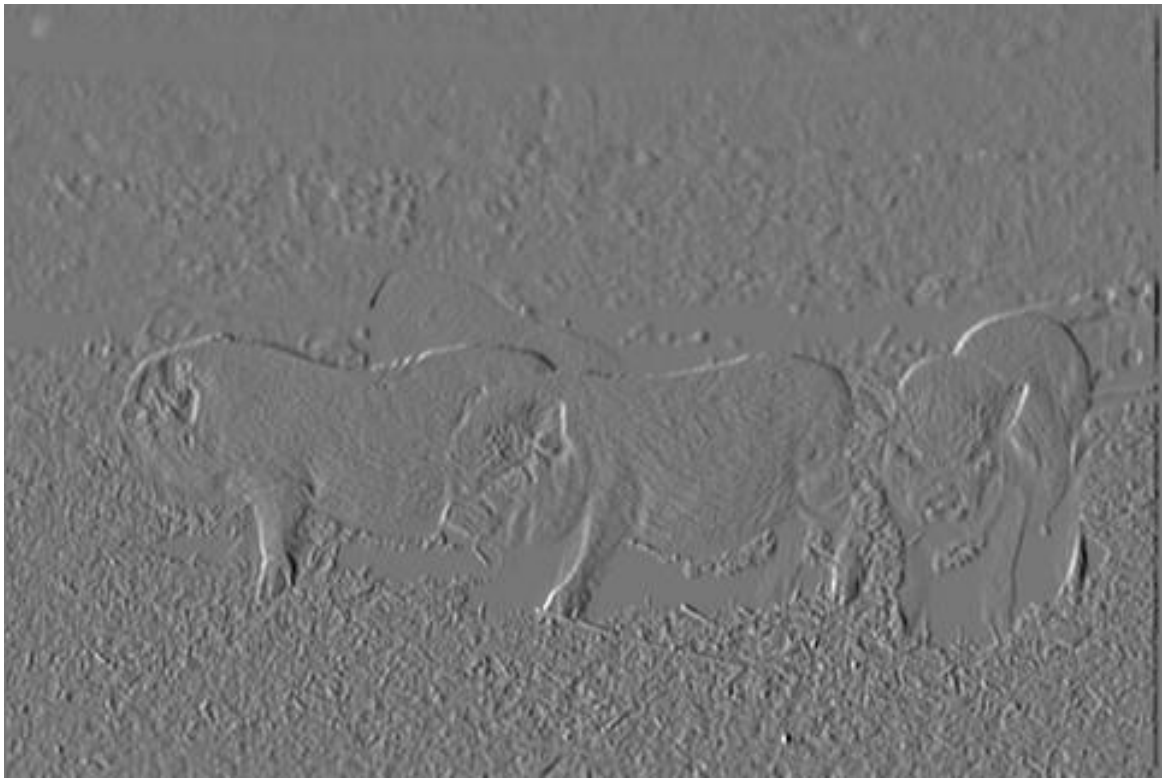


Figure 14 Pig Image Normalized X-Gradient Using Sobel X-Mask

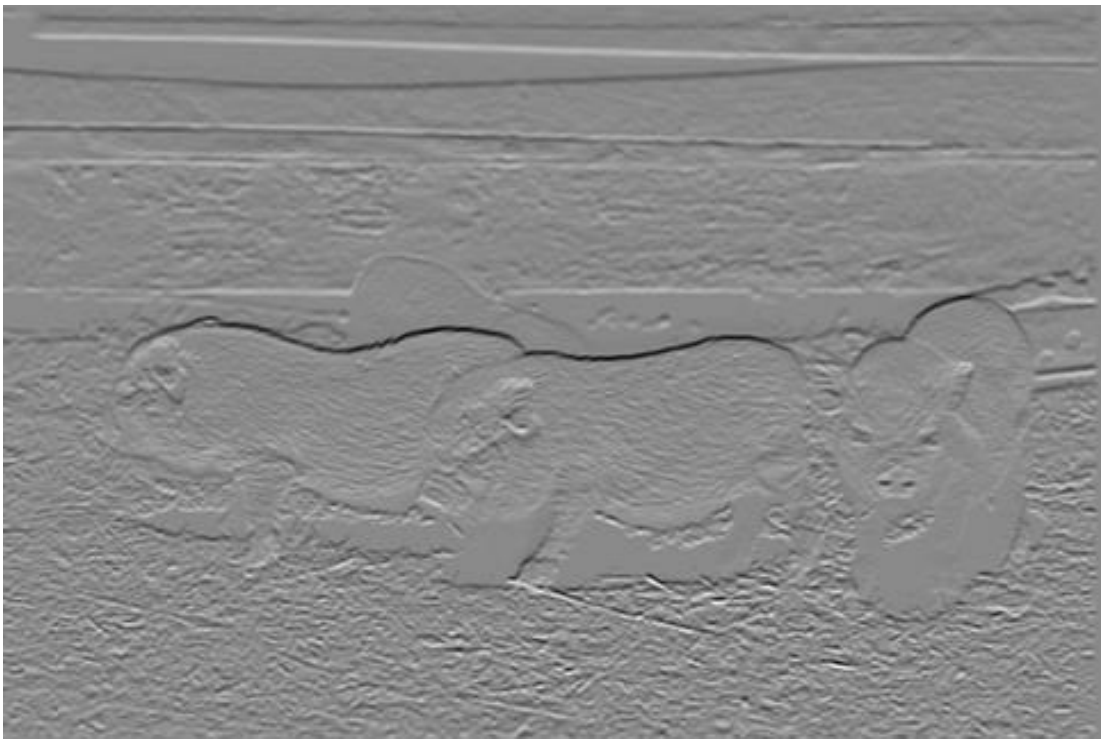


Figure 15 Pig Image Normalized Y-Gradient Using Sobel Y-Mask

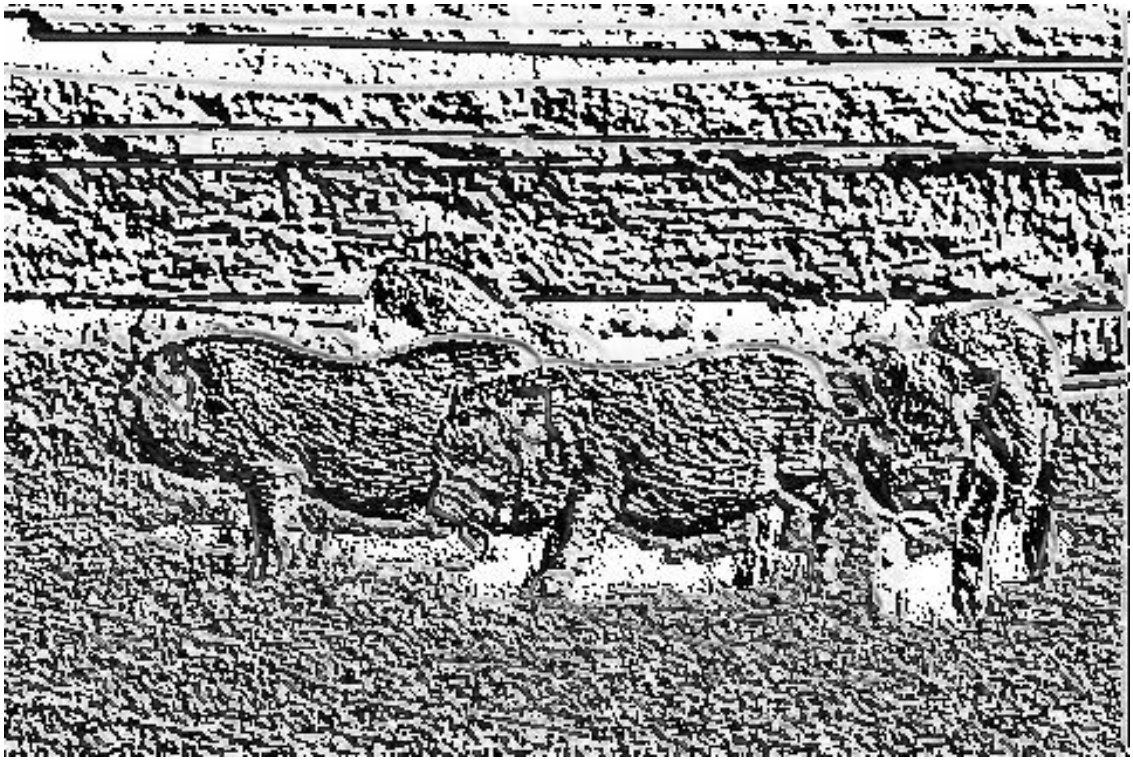


Figure 16 Pig Image Gradient Map

Figure 17 Pig Edge Output for 60% Threshold

Figure 18 Pig Edge Output for 70% Threshold

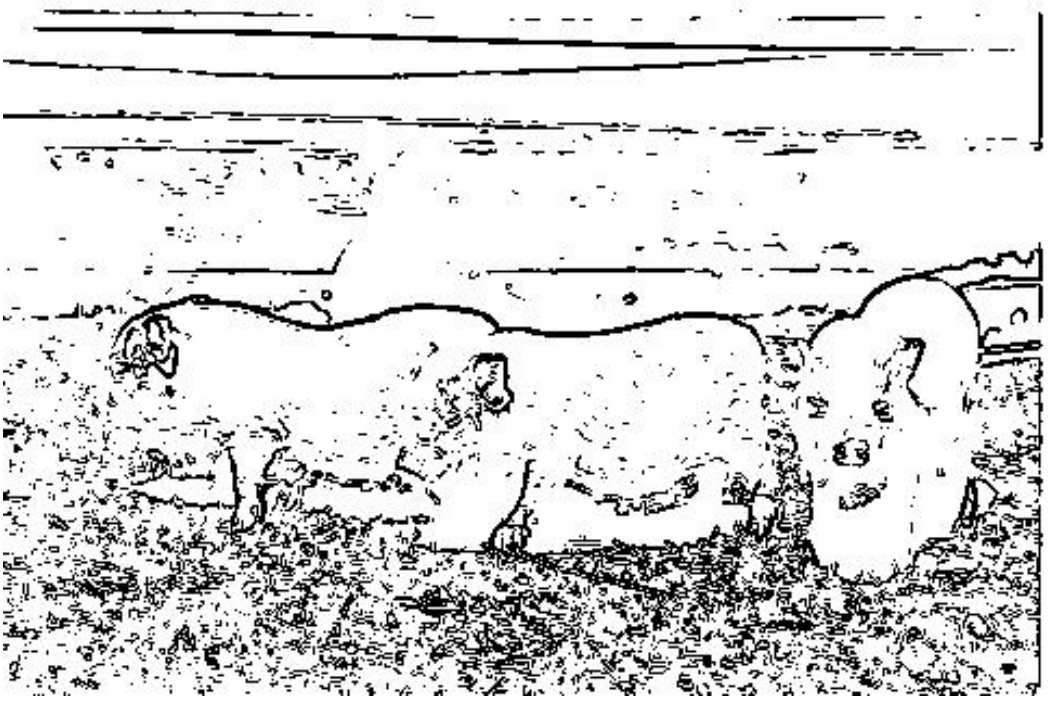


Figure 19 Pig Edge Output for 85% Threshold

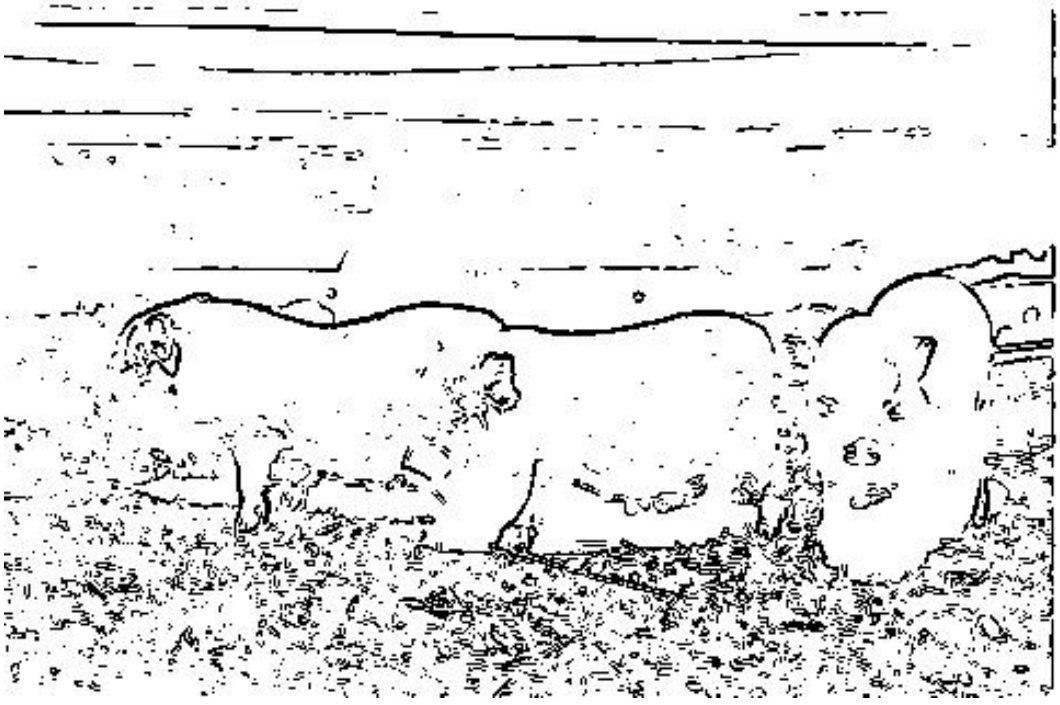
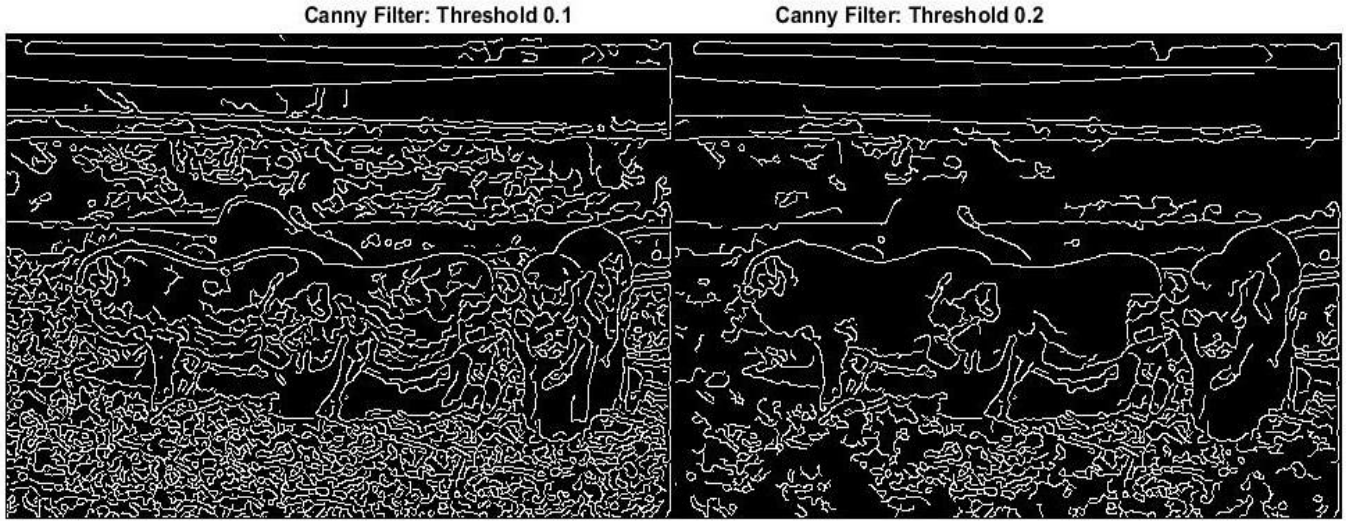
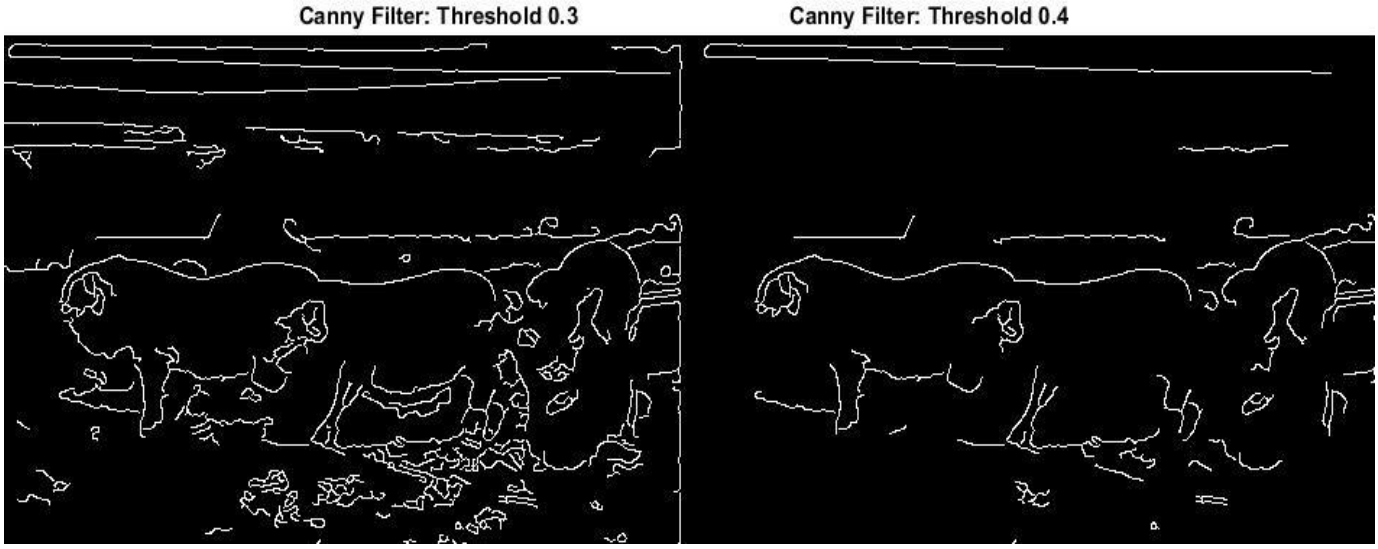
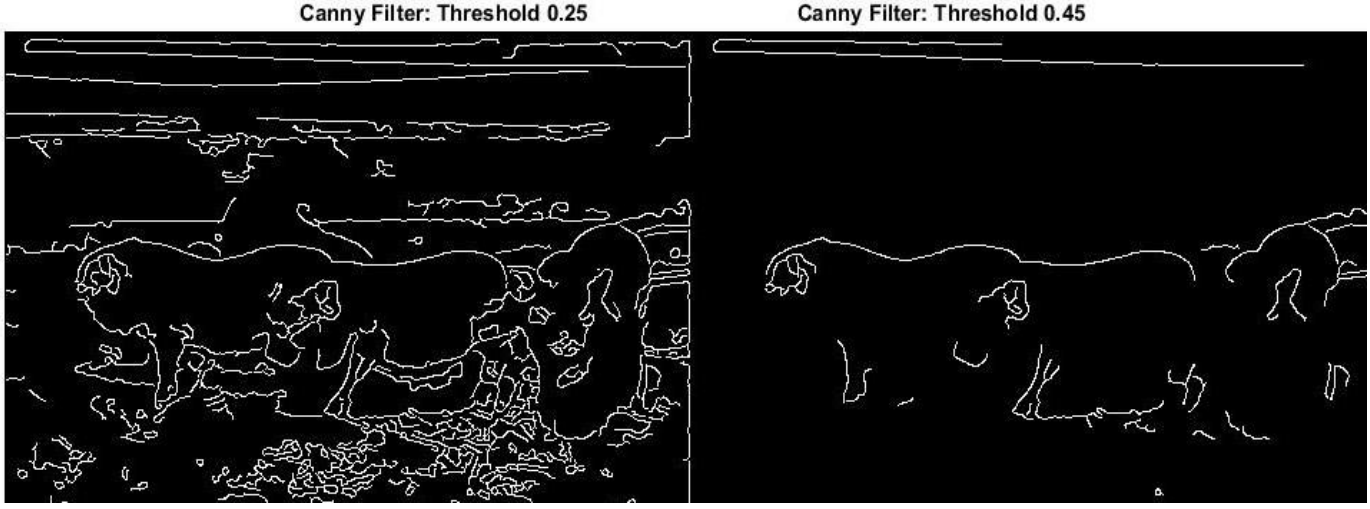


Figure 20 Pig Edge Output for 90% Threshold

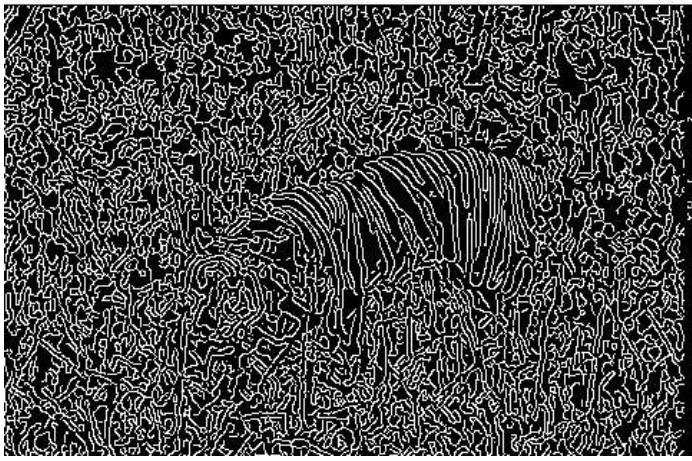
CANNY EDGE DETECTOR



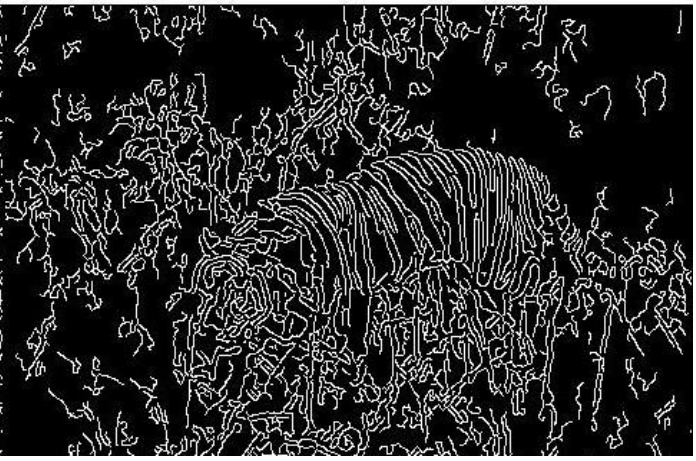




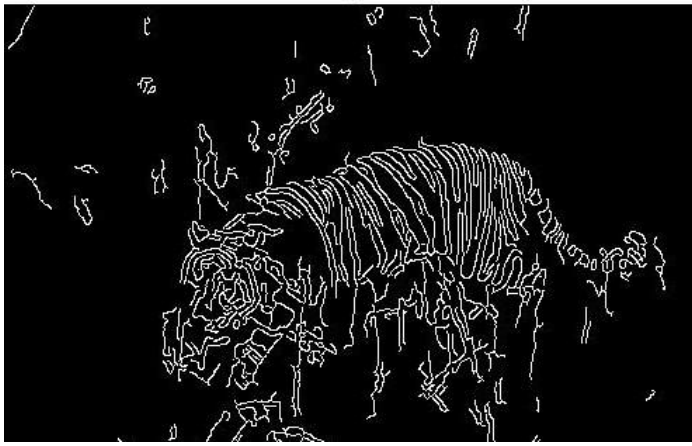
Canny Filter: Threshold 0.1



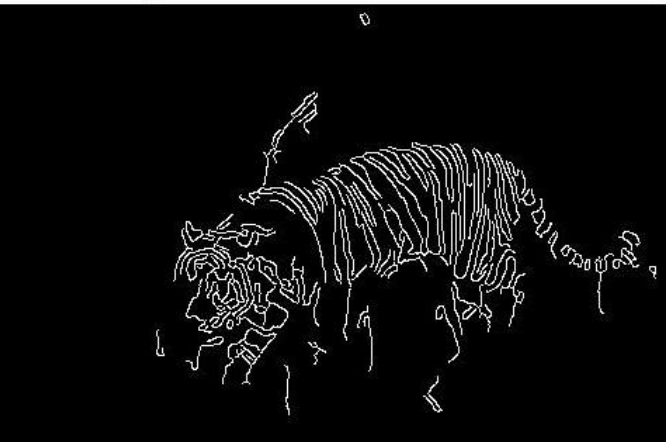
Canny Filter: Threshold 0.2



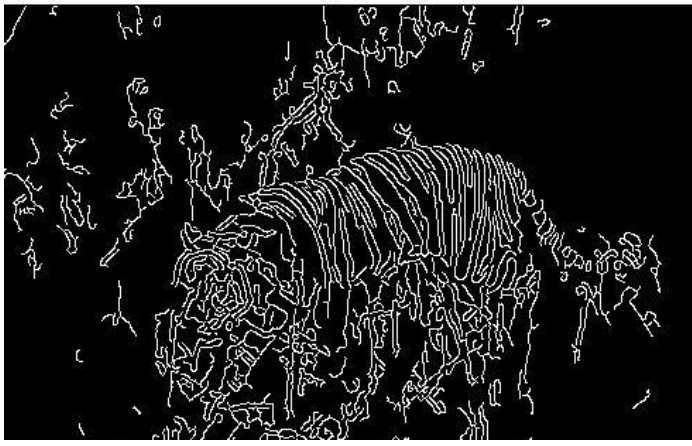
Canny Filter: Threshold 0.3



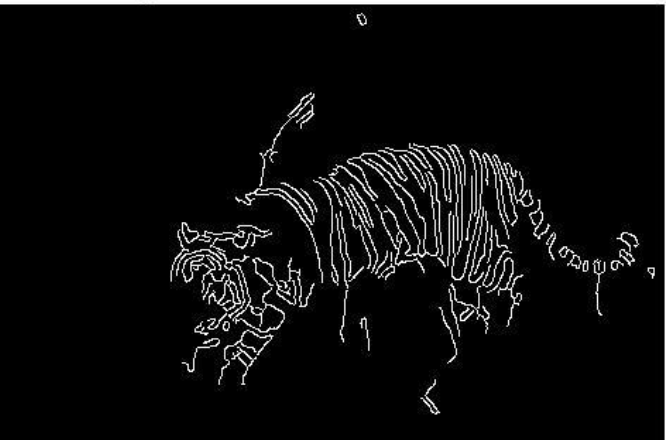
Canny Filter: Threshold 0.4



Canny Filter: Threshold 0.25



Canny Filter: Threshold 0.45



STRUCTURED EDGE

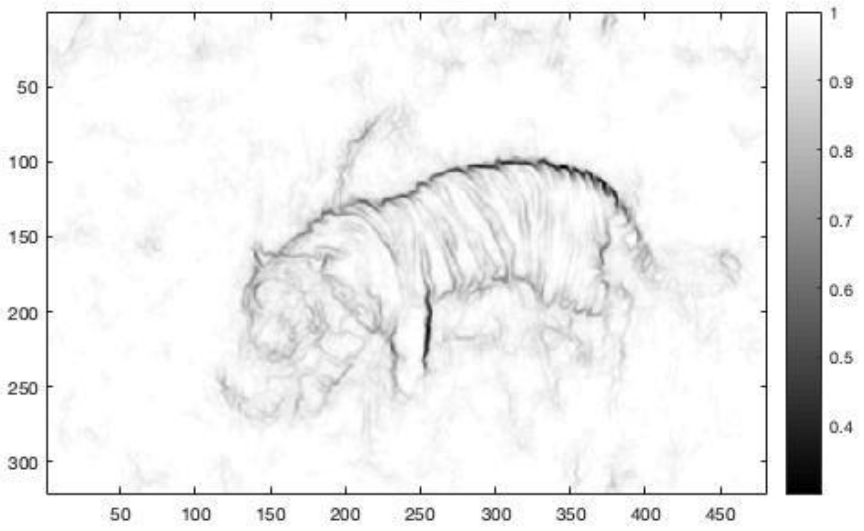


Figure 21 Structured Edge Tiger Probability Edge Map

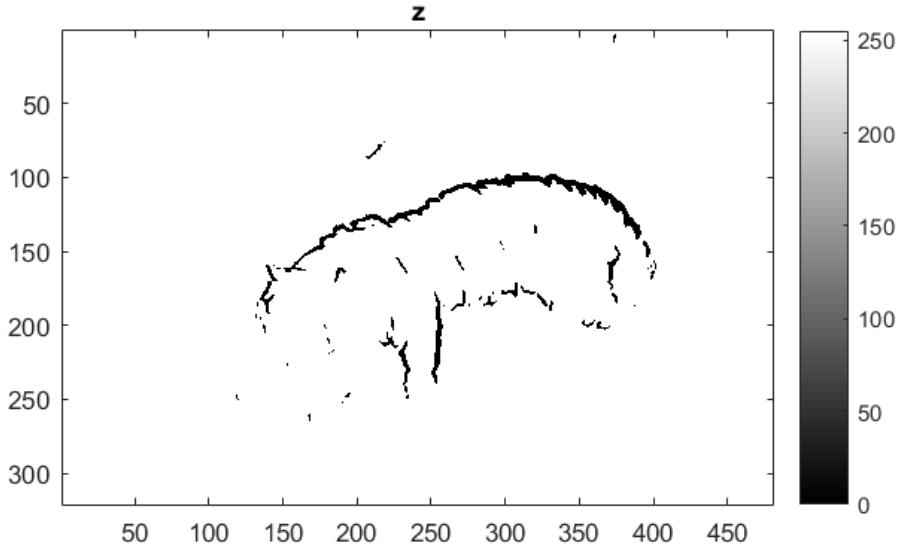


Figure 22 Structured Edge Tiger Edge Map

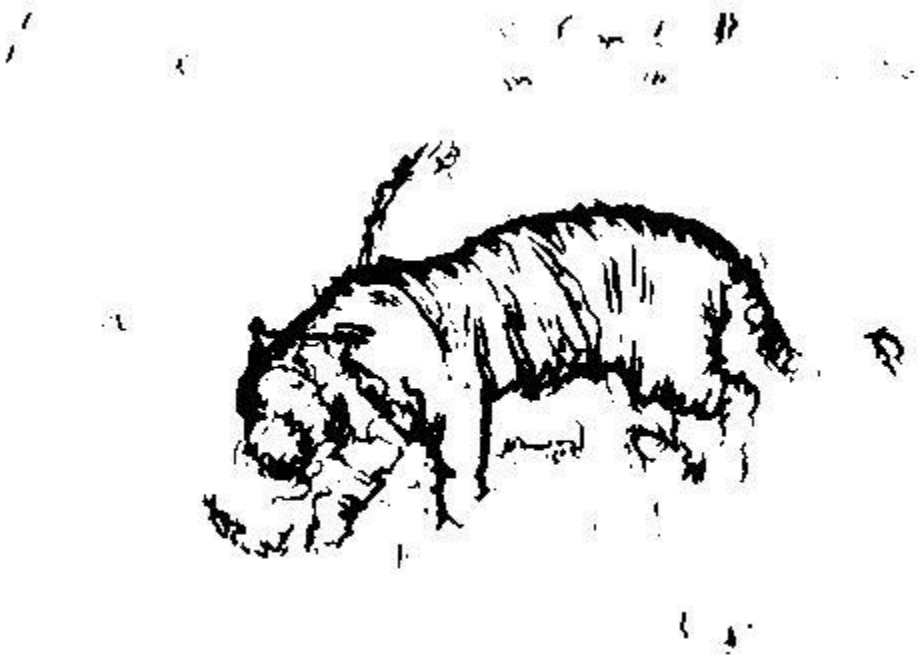


Figure 23 Structured Edge Tiger Binary Edge Map p<0.92 or p>0.08

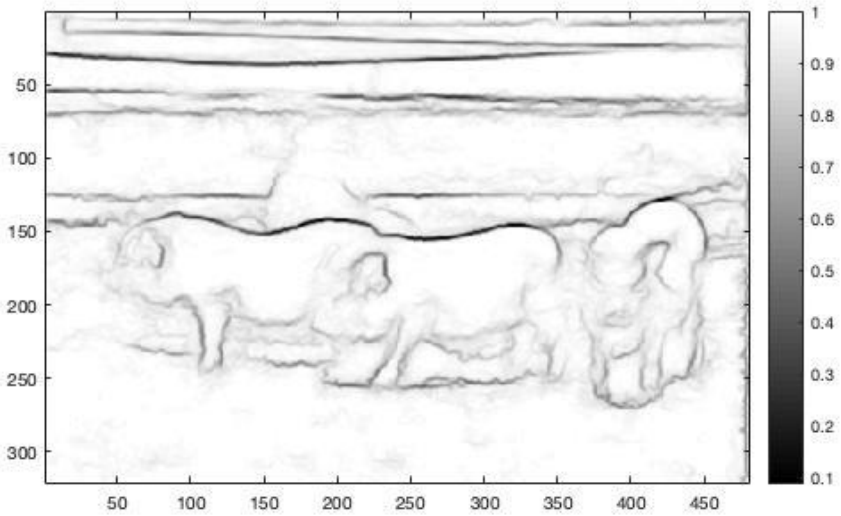


Figure 24 Structured Edge Pig probability Edge Map

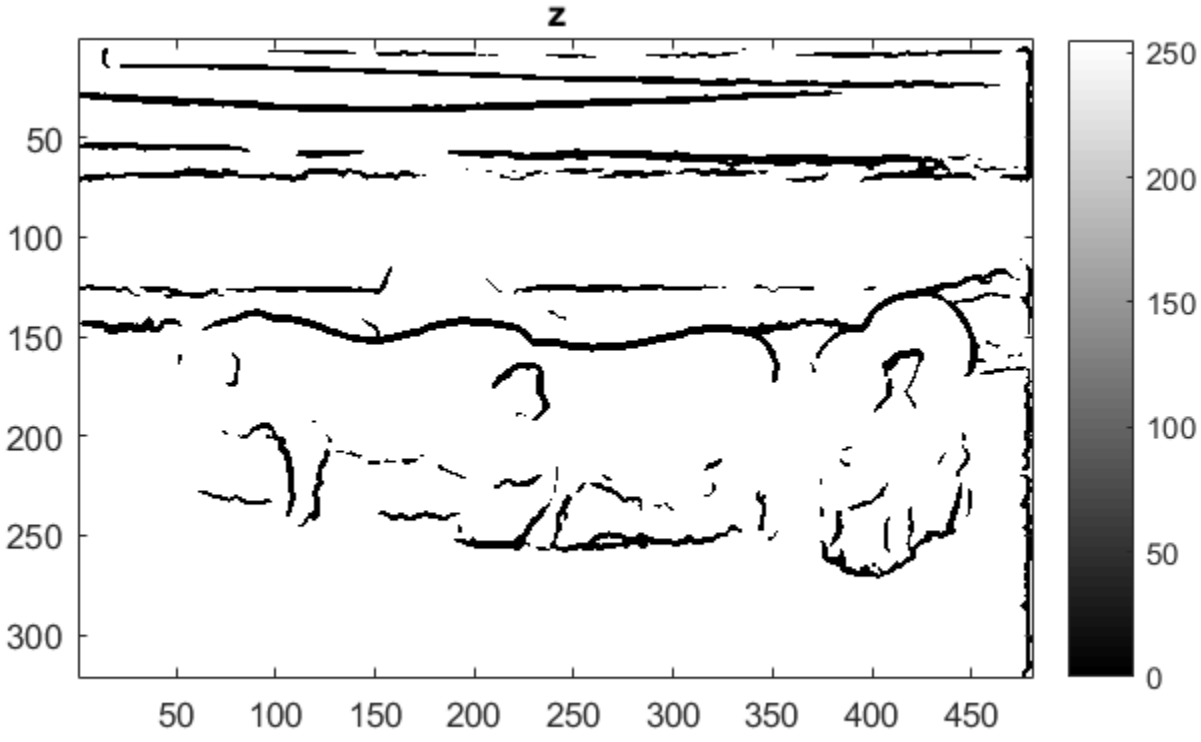


Figure 25 Structured Edge Pig Binary Edge Map

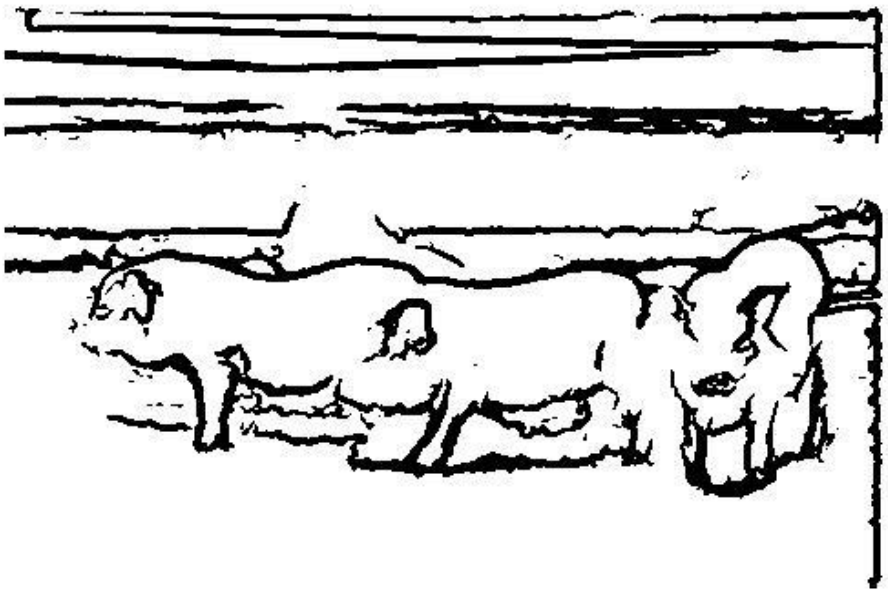


Figure 26 Structured Edge Pig Binary Edge Map p<0.88, or p>0.12

DISCUSSION

1. SOBEL EDGE DETECTION

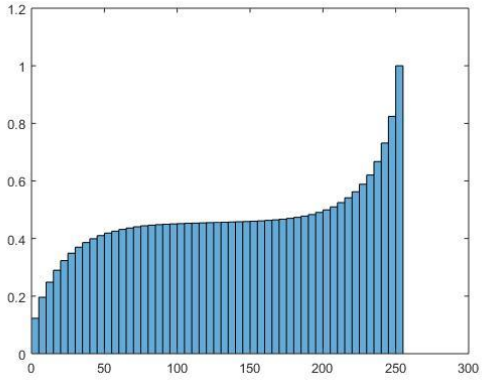


Figure 27 Cumulative Histogram Plot of Pig_Gradient

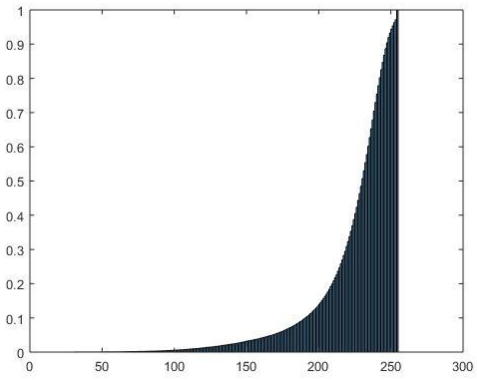


Figure 28 Cumulative Histogram Plot Tiger_Gradient

From the above images, we observe that:

- For Pig, best edge detection is seen for 70% threshold whereas for Tiger, The best edge map output is obtained for about 85% threshold. (About ~251 pixel intensity)
- When threshold is too low (eg 60%), a lot of edges are detected which are object textures and other details and are unwanted, whereas when threshold is too high (eg 90%), a few important edges are not detected, causing discontinuity in object edge contours.
- Because of bright texture of the pig's body, we see loss of edges around the body outline of middle pig. i.e. whenever there is a too bright texture in the image, the edges across that region have spurious edge detection
- This method is computationally fast and less complex.
- It can be further improved using Canny Edge Detection.

2. CANNY EDGE DETECTION

Implemented Canny Edge detector for various thresholds for both tiger and pig images.

From the results, we can observe,

- Best edge output for pig is obtained for threshold 0.25 and for Tiger, the edge output comes best for threshold 0.4.

- Canny Edge Detector gives better localization and better Signal to Noise Ratio.
- As threshold is increased than 0.4, the connectivity of the edges breaks and the edge map is distorted.
- For very low threshold value like 0.1, we see a lot of spurious edge detections which are not edges, but are detected as edge because of noise, texture of image and low thresholding.
- For the second code, I got best output for low threshold 0.1 and high threshold 0.3. i.e. 3:1 high-low thresholding.
- Wherever there is a too bright texture in the image, the edges across that region have bad edge detection.
- As the value of sigma increases, the poorer is edge detection of intricate regions, but since the noise is removed more by the high gaussian sigma, we get a cleaner and thinner edge detection as sigma increases. i.e.
    - Large sigma detects large scale images
    - Small sigma detects fine features
- Canny Edge detector fails to differentiate if the edge is of the object or just it's texture, hence a lot of spurious edges are seen which cannot be handled by changing the values of high and low thresholds. Thus, Canny mostly ends up detecting edges inside objects. Like for example, we can't detect the middle pig outline correctly and tiger face outline edge correctly.
- Sigma 0.33 gives a good edge map for both Pig and Tiger images.
- Canny edge detection is complex to implement and time consuming.

3. STRUCTURED EDGE DETECTION

I have explained about Structured edge Detection along with flowchart in the Abstract and Motivation Section.

I have explained the process of decision tree construction and principles of RF classifier also in the Abstract and Motivation Section.

We can observe from the above results that:

- Structured Edge gives the best output and performance for edges than Sobel and Canny edge detectors.
- It doesn't detect texture unlike canny edge detector. So, texture parts can be ruled out.

- SE is suitable for every kind of local structure because it has well developed local learning-based contour detection.
- For tiger and pig images, the threshold set to probability map was set to0.92 and 0.88 respectively for getting a binary map. i.e. p< 0.92 and p < 0.88.
- The parameters used for the model are:

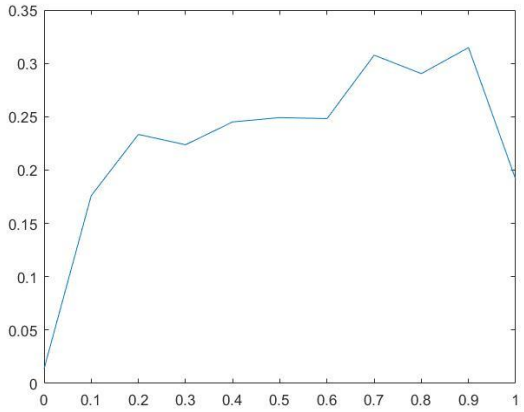
```
%% set detection parameters (can set after training)
model.opts.multiscale=1;          % for top accuracy set multiscale=1
model.opts.sharpen=2;             % for top speed set sharpen=0
model.opts.nTreesEval=1;          % for top speed set nTreesEval=1
model.opts.nThreads=4;           % max number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

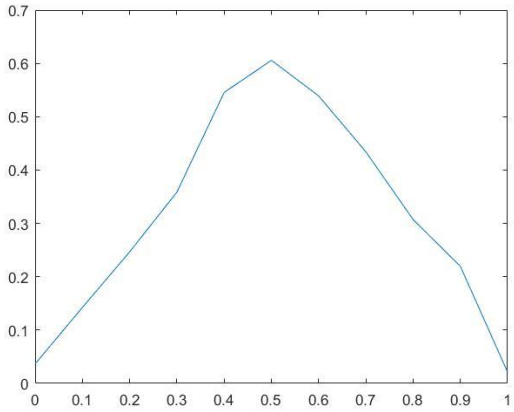
Figure 29 Parameter Selection for edgesDemo.m

- Unlike Canny Edge Detector, Structured edge detector has an intelligent layer that allows it to filter out and display only the important edges hence reducing noise and unwanted edges.
- It is a **fast-paced algorithm** unlike other algorithms which are not intelligent. Other algorithms also depend on gradient, orientation and other parameters that increase complexity like for Sobel and Canny.
- As sharpen parameter decreases, edge map gets diffused edges. Hence, I have kept sharpen parameter = 2 to get sharp edges.
- I have set multiscale parameter = 1 for top accuracy.
- I have set Tree evaluation parameter = 1 so that the algorithm will select one tree from available 2. (Binary Split).
- Binary Edge map of SE is very thick which results in coverage of less details of the edges as compared to Canny that gives a very fine and detailed edge detection.
- Also, for SE, we can differentiate between objects based on the probability edge map which won't be possible for a canny edge detector output.

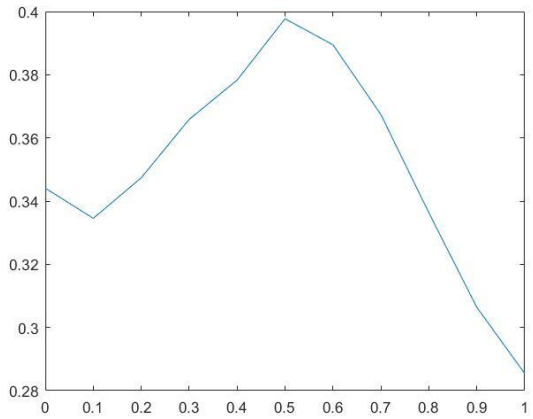
4. PERFORMANCE EVALUATION

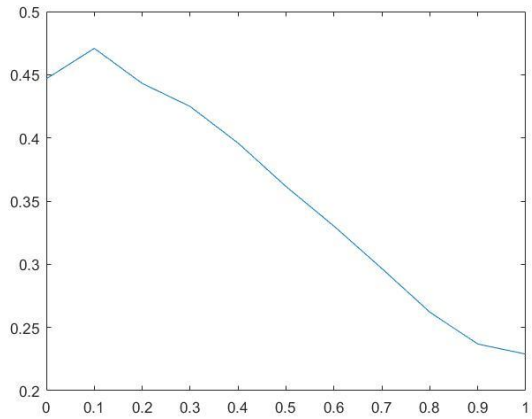
For performance evaluation I found probability edge maps for Tiger and Pig images for Canny and Sobel Edge detectors.



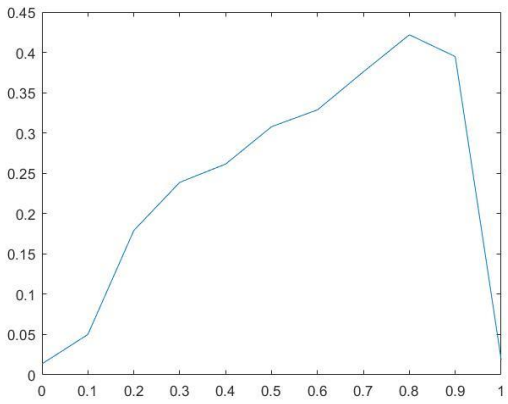


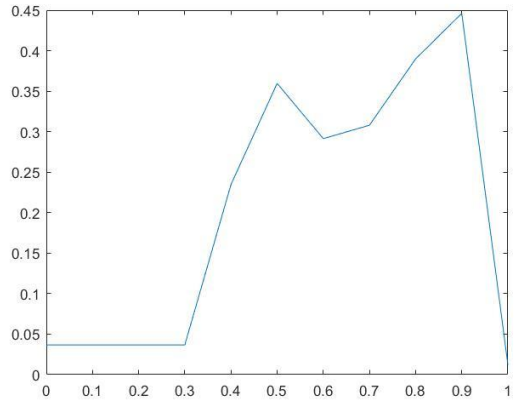
Sobel Edge Detector Pig (Best F-measure 0.3149) and Tiger F-measure (Best F-measure 0.6059) vs Threshold Plot





Canny Edge Detector Pig (Best F-measure 0.3969) and Tiger F-measure (Best F-measure- 0.4710) vs Threshold Plot





Structured Edge Detector Pig (Best F-measure 0.4733) and Tiger F-measure (Best F-measure 0.57690) vs Threshold Plot

TABULATED RESULTS:

STRUCTURED EDGE:

Pig: - Best F-measure = 0.4733

Ground Truths	Mean Recall	Mean Precision	Mean F-score
GT1	0.0097	0.1242	0.0179
GT2	0.1552	0.2557	0.1931
GT3	0.2089	0.1789	0.1840
GT4	0.2203	0.1799	0.1961
GT5	0.4430	0.2184	0.2935

Tiger: - Best F-measure = 0.5769

Ground Truths	Mean Recall	Mean Precision	Mean F-score
GT1	0.0067	0.1267	0.0239
GT2	0.1534	0.1723	0.1868
GT3	0.2086	0.2343	0.1578
GT4	0.1386	0.4789	0.1366
GT5	0.1401	0.1067	0.1406

CANNY EDGE:

Pig: -

Ground Truths	Mean Recall	Mean Precision	Mean F-score
GT1	0.1716	0.0625	0.2436
GT2	0.1752	0.657	0.1561
GT3	0.2089	0.1789	0.2250
GT4	0.2203	0.1799	0.2415
GT5	0.4430	0.2184	0.2181

Tiger: -

Ground Truths	Mean Recall	Mean Precision	Mean F-score
GT1	0.0067	0.1267	0.0916
GT2	0.1534	0.1723	0.0957
GT3	0.2086	0.2343	0.1036
GT4	0.1386	0.4789	0.2728
GT5	0.1401	0.1067	0.1383

SOBEL EDGE:

Pig: -

Ground Truths	Mean Recall	Mean Precision	Mean F-score
GT1	0.1716	0.0625	0.1278
GT2	0.1752	0.657	0.1462
GT3	0.2089	0.1789	0.1493
GT4	0.2203	0.1799	0.4164
GT5	0.4430	0.2184	0.1561

Tiger: -

Ground Truths	Mean Recall	Mean Precision	Mean F-score
GT1	0.4532	0.1340	0.2069
GT2	0.4470	0.1396	0.2127
GT3	0.3792	0.1696	0.2343
GT4	0.3372	0.1720	0.2268
GT5	0.3584	0.1552	0.2166

Best F-measure scores:

Sobel Pig = 0.3149

Sobel Tiger = 0.6059

Canny Pig = 0.3969

Canny Tiger = 0.4710

- F-score gives a measure of evaluation of performance of edge detection. Better F-score means better performance. From above results, we can infer that:
- SE has highest F-score hence best performance. So, better than Canny and Sobel.
- F-score is dependent on input image.
- From intuition, it is easier to get better or higher F-score for Pig than Tiger. Pig has better defined edges than Tiger. Tiger has a lot of inside body texture edges which are just texture and not desired contour edge. Also, it is simpler to segment the pigs from their background rather than Tiger from it's grassy background since the grasses have a lot of texture components and also, the tiger is very close to source, so more difficult to segment.
- F-score depends on both R and P and there is always a tradeoff between them. We generally don't prefer too much difference between the R and P values, we prefer them to be on similar levels.
- Assume P + R = C (constant)

  F = 2 * P * R / (P + R)

  Or, F = 2 *(R – C) * R/C

  Taking derivative with respect to R, we get,

  4*R = 2*C

  Which implies, **P = R.**

  Thus, **F measure reaches maximum value when precision equals recall.**

PROBLEM 2: DIGITAL HALF-TONING

ABSTRACT AND MOTIVATION

Halftoning is a technique used in the printing as well as publishing industry to simulate shades of gray by varying the size and density of black dots that are arranged in a regular pattern. It helps reduce cost by interpolating all colors from the smaller subset of colors present. For example, if we inspect a newspaper, we can notice that the pictures are composed of fine black dots, even though from a distance, it appears to have gray pixels because of the spatial low-pass filtering by our eyes that blend fine details and record the overall intensity. Half-toning is a method for creating the illusion of continuous tone output with a binary device.

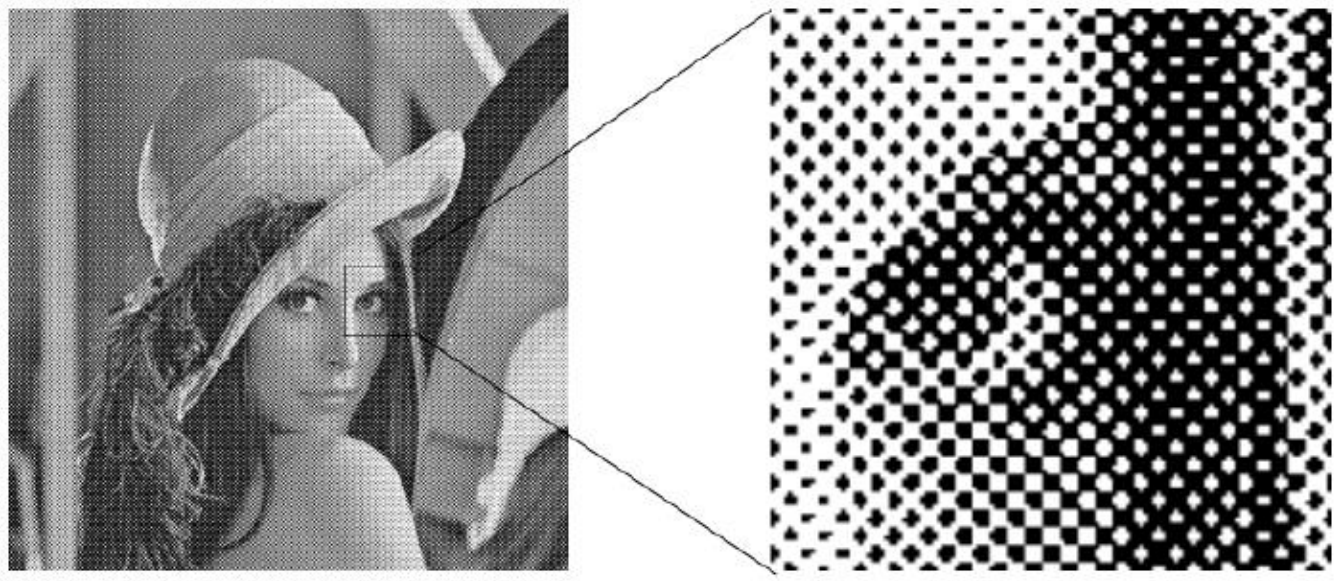


Figure 30 Digital Halftoning in Principle

We decompose the image into a grid of halftone cells. Higher the density of black dots, darker the cell appears. In halftoned Image, the density of black dots in the half-toned image approximates the averaged gray level intensity of the original Image.

There are three ways of generating Digital Halftoning Images:

1. Patterning
2. Dithering
3. Error Diffusion

Here, we implement Dithering and Error Diffusion Methods and compare their performances.

DITHERING USING RANDOM THRESHOLDING

Here, instead of thresholding with a constant value, we threshold the Image with respect to a randomly generated threshold value.

The algorithm can be described as:

• For each pixel, generate a random number in the range $0 \sim 255$, $rand(i,j)$

• Compare the pixel value with $rand(i, j)$. If it is greater, then map it to 255; otherwise, map it to 0,

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

i.e.

ORDERED DITHERING

In Ordered Dithering, we create an output image having the same number of dots as the number of pixels in the input image using a dithering matrix. It works like thresholding the input image with a dither matrix that is laid repeatedly over the input image.

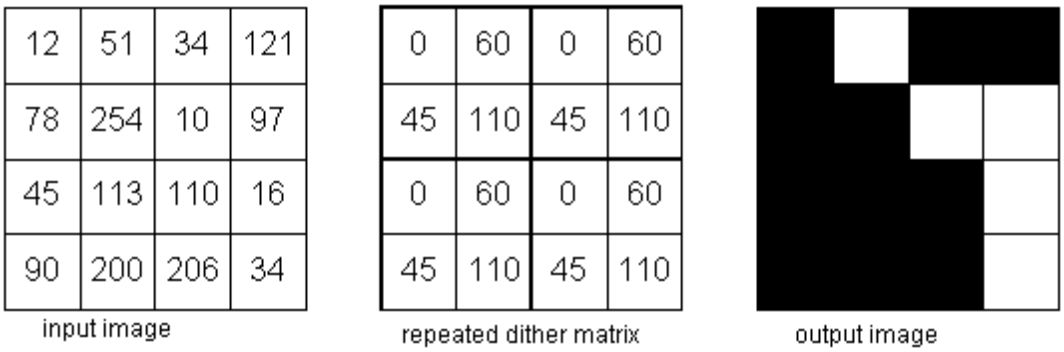


Figure 31 Dithering Workflow

In principle, we intend to apply noise to the image to randomize the quantization error.

Here, we use Bayer's dithering matrix. The dithering parameters here are defined by an index matrix where the values in the index matrix indicate the likelihood of a dot being turned on. Basic Index matrix is defined as below, where 0 indicates the pixel most likely to be turned on, and 3 is the least likely one.

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

The Bayer index matrices are defined recursively using the formula:

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

Figure 32 Formula for Generating Bayer's Dithering Matrices

$$I_2 = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} \qquad I_4 = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} \qquad I_8 = \begin{bmatrix} 0 & 32 & 8 & 40 & 2 & 34 & 10 & 42 \\ 48 & 16 & 56 & 24 & 50 & 18 & 58 & 26 \\ 12 & 44 & 4 & 36 & 14 & 46 & 6 & 38 \\ 60 & 28 & 52 & 20 & 62 & 30 & 54 & 22 \\ 3 & 35 & 11 & 43 & 1 & 33 & 9 & 41 \\ 51 & 19 & 59 & 27 & 49 & 17 & 57 & 25 \\ 15 & 47 & 7 & 39 & 13 & 45 & 5 & 37 \\ 63 & 31 & 55 & 23 & 61 & 29 & 53 & 21 \end{bmatrix}$$

Figure 33 Ordered Dithering Index Matrices

(Source https https://en.wikipedia.org/wiki/Ordered_dithering)

The index matrix can then be transformed into a threshold matrix T for an input gray-level image with normalized pixel values (i.e. with its dynamic range between 0 and 255) by the following formula:

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

where N denotes the number of pixels in the matrix. Since the image is usually much larger than the threshold matrix, the matrix is repeated periodically across the full image using Mod function. This is done by using the following formula:

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

APPROACH AND PROCEDURE

Algorithm used for Dithering using Random Thresholding:

1. For every pixel in the image threshold the image using random number generated in each iteration. As the algorithm is defined in the Abstract and Motivation Part.

Algorithm used for Ordered Dithering:

1. Implement Recursive Function for creating the Dithering matrix.
2. Use the Dithering Matrices for I=2, I=4, I=8and I=32 and get the corresponding thresholding matrices by using the Threshold formula.
3. To get halftone output, for each pixel in the image, compare the pixel with the corresponding pixel in the threshold matrix and set it to 0(low) or 255(high) based on the thresholding values of the threshold matrix.

```
C:\Users\AbsurdFantasy\DIP_Dithering_Matrix>main bridge.raw dither2.raw
1 2
3 0
```

Figure 34 Dithering using I2 Matrix Created

```
C:\Users\AbsurdFantasy\DIP_Dithering_Matrix>main bridge.raw dither4.raw
5 9 6 10
13 1 14 2
7 11 4 8
15 3 12 0
```

Figure 35 Dithering using I4 Matrix Created

```
C:\Users\AbsurdFantasy\DIP_Dithering_Matrix>g++ main.cpp -o main

C:\Users\AbsurdFantasy\DIP_Dithering_Matrix>main bridge.raw dither8.raw
21 37 25 41 22 38 26 42
53 5 57 9 54 6 58 10
29 45 17 33 30 46 18 34
61 13 49 1 62 14 50 2
23 39 27 43 20 36 24 40
55 7 59 11 52 4 56 8
31 47 19 35 28 44 16 32
63 15 51 3 60 12 48 0
```

Figure 36 Dithering using I32 Matrix Created

EXPERIMENTAL RESULTS

Figure 37 Original Bridge Image

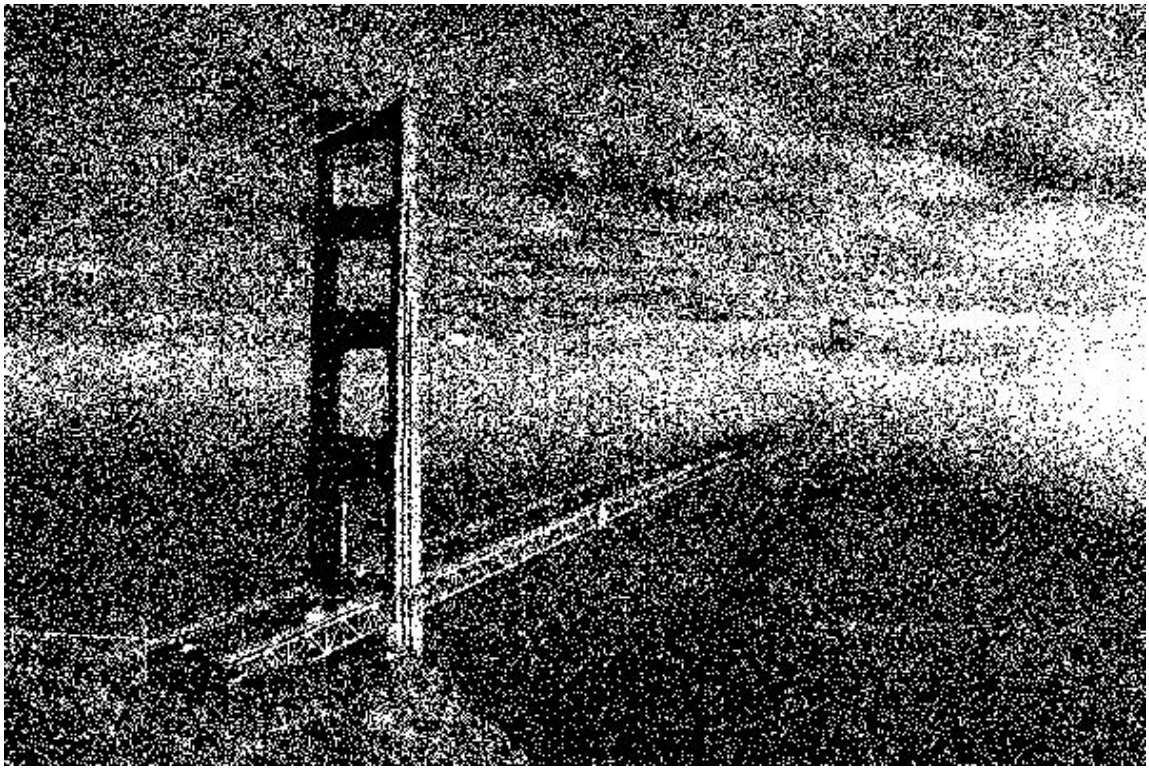


Figure 38 Halftoned Output Image Using Random Thresholding

Figure 39 Original Bridge Image

Figure 40 Halftoned Output Image Using Ordered I2 Dither Matrix

Figure 41 Original Bridge Image

Figure 42 Halftoned Output Image Using Ordered I4 Dither Matrix

Figure 43 Original Bridge Image

Figure 44 Halftoned Output Image Using Ordered I8 Dither Matrix

Figure 45 Original Bridge Image

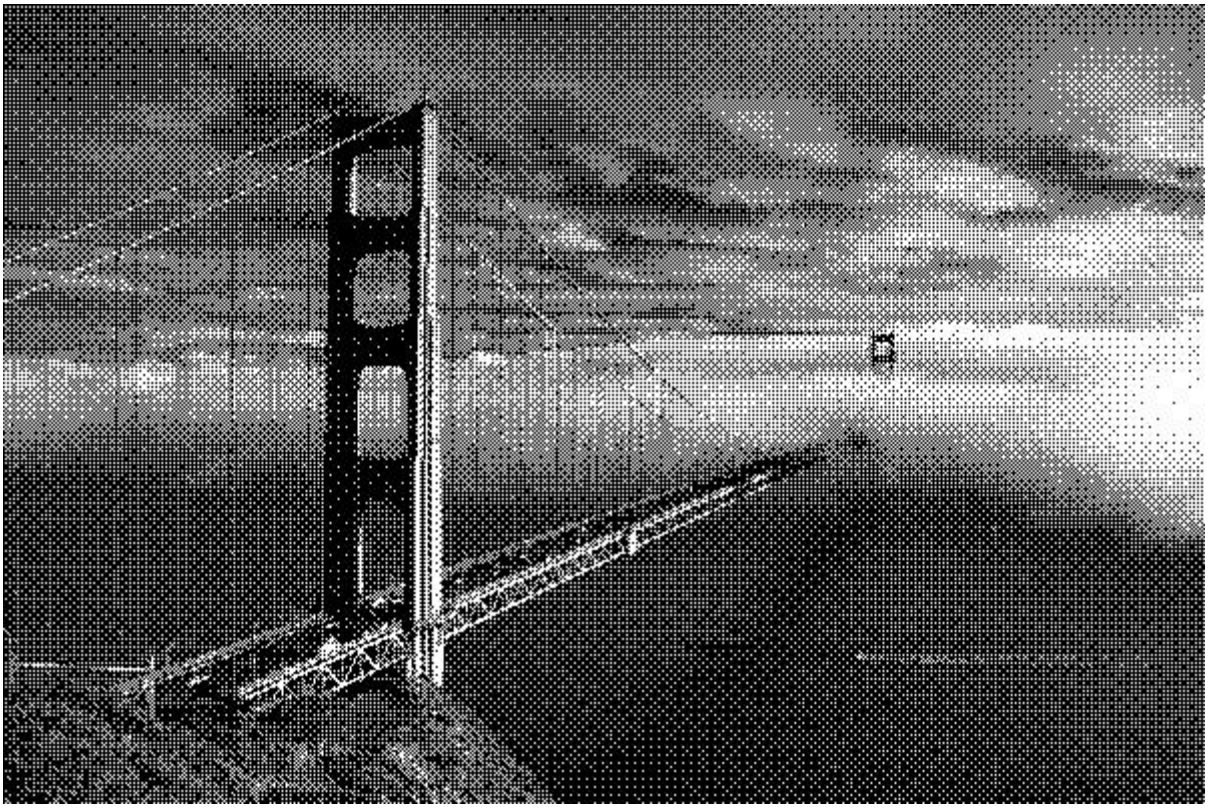


Figure 46 Halftoned Output Image Using Ordered I32 Dither Matrix

DISCUSSION

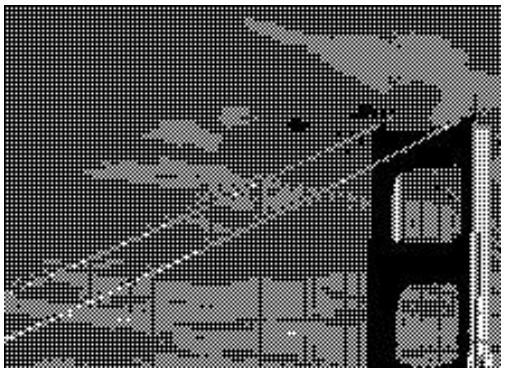


Figure 47 Enlarged I2 Dithered Halftone Output

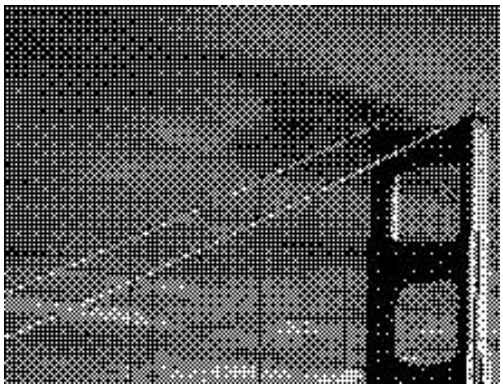


Figure 48 Enlarged I8 Dithered Halftone Output

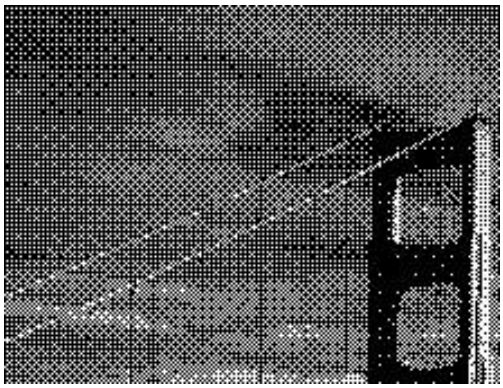


Figure 49 Enlarged I32 Dithered Halftone Output

From the above results, we observe that:

1. In dithering using Random thresholding, we see the introduction of **random noise**, giving a distorted halftoned output i.e. with poor visual fidelity.
2. Further, random thresholding gives a **False Contour Effect** and gives a **Staticky Image** output, in which the noise swamps the details of the image.

3. Halftoning with Bayer matrix gives better output i.e. output image with better visual fidelity than random thresholding because of the **distribution of the quantization error** of each pixel.
4. Dithering gives finer amplitude quantization over larger area.
5. Even though this method retains good retail rendition of the image over smaller areas, we can see the introduction of a periodic pattern or visual artifact in each output halftoned Image i.e. undesired patterns that might closely relate to distortion. But, the false contour effect is reduced.
6. Random Thresholding is faster method consuming less memory.
7. As we increase the size or the order of the Bayer Dithering matrix, the density of the dots keeps on increasing, giving a better Halftoned Output. (More theoretically than is visually apparent, unless viewed from a distance).
8. The I32 halftone output overall gives a halftoned Image that resembles the grayscale image if seen from a distance.
9. I2 and I8 halftoned Output Images look filled with dots, as the density of dots is comparatively lesser than I32.
10. For a continuous gradient part of an image, dithering causes holed block like structures because of the varying thresholds in the matrix for example, the water body in the bridge image.
11. I8 inherently exhibits plus sign patterns that smoothens the output image. Hence, in practice, I8 matrix is used popularly.

ERROR DIFFUSION

ABSTRACT AND MOTIVATION

In this halftoning method, we diffuse the quantization error into the neighboring pixels in 2 dimensions. For example, we can diffuse the error to one or more pixels on the right, left, up or down. Mostly, we diffuse the error into the forward pixels that have not yet been processed.

This is an area-based algorithm since the diffusion of error at one area influences the diffusion to happen at the other locations. The distribution of error is different when we use different algorithms.

Basic Error Diffusion technique can be depicted as:

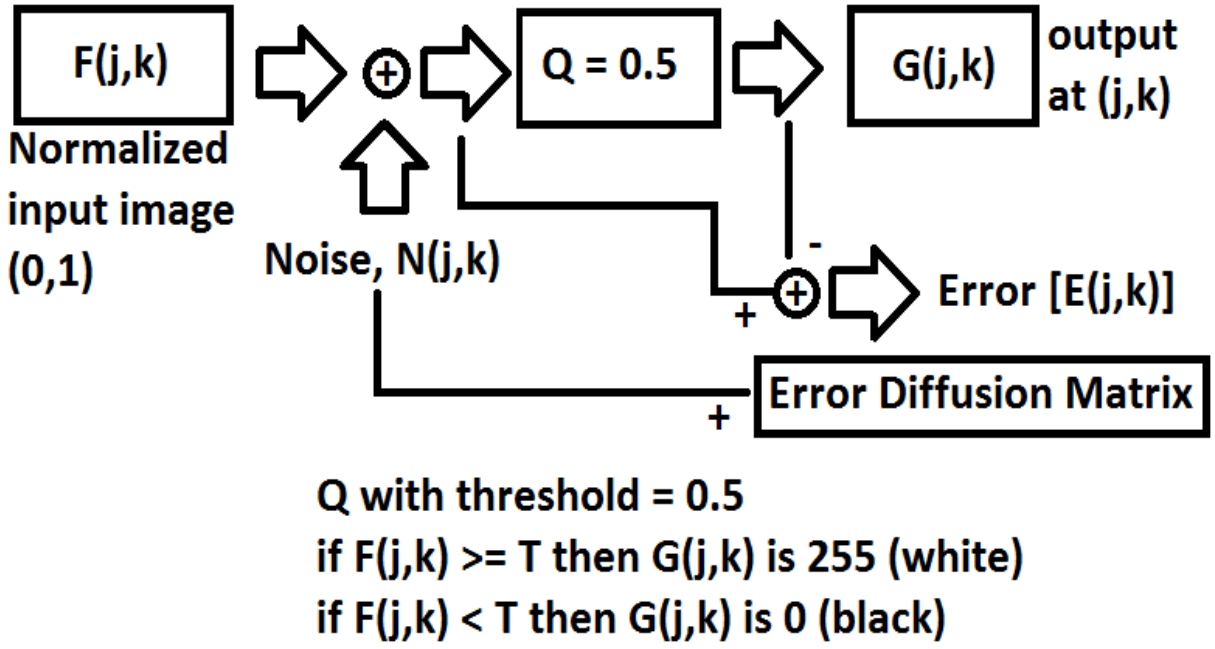


Figure 50 Error Diffusion Method

In terms of equations, we can write the algorithm as:

$$b(i,j) = \begin{cases} 255, & if\ \tilde{f}(i,j) > T \\ 0, & otherwise \end{cases}$$

$$e(i,j) = \tilde{f}(i,j) - b(i,j)$$

$$\tilde{f}(i,j) = f(i,j) + \sum_{k,l} h(k,l)e(i-k,j-l)$$

The algorithm pushes (adds) the error of a pixel onto its neighboring pixels, spreading the debt throughout the Image by scanning it pixel by pixel. The quantization error is diffused without affecting already quantized pixels. Scanning can be done in different ways, for example Raster Scanning, Serpentine Scanning, etc.

APPROACH AND PROCEDURE

The Main algorithms for Error Diffusion are given by the following Error Diffusion Matrices. The sum of all coefficients is 1 for each matrix. Since we have to use the Serpentine Scanning, we use different diffusion locations for Left->Right scanning and Right->Left Scanning respectively as shown below.

For Odd rows: Left-Right:

A. Floyd Steinberg's error diffusion matrix,

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

B. Jarvis, Judice and Ninke (JJN) error diffusion matrix,

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

C. Stucki's error diffusion matrix,

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

The matrices are correlated with the corresponding indexes of the Image to give the Error values to be diffused.

For Even Rows: (Right to Left)

A. Floyd Steinberg's error diffusion matrix,

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

B. Jarvis, Judice and Ninke (JJN) error diffusion matrix,

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

C. Stucki's error diffusion matrix,

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

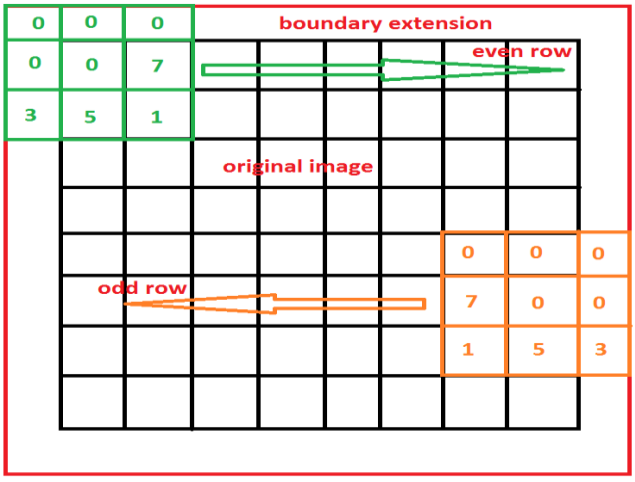


Figure 51 Floyd Steinberg Error Diffusion using Serpentine Method (Matrix Values are divided by 16)

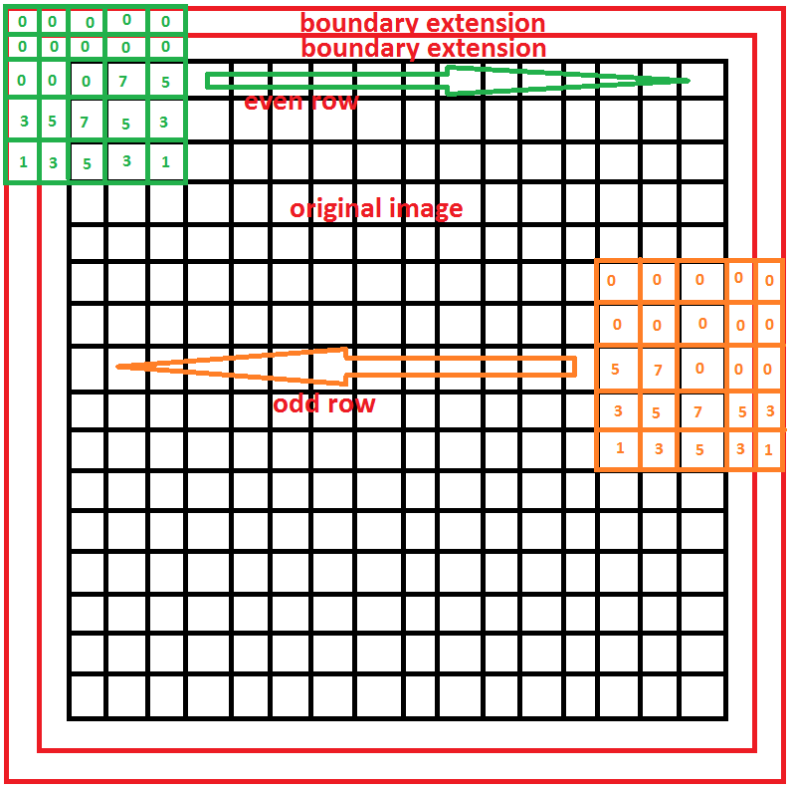


Figure 52 JJN Error Diffusion using Serpentine Method (Matrix Values are divided by 48)

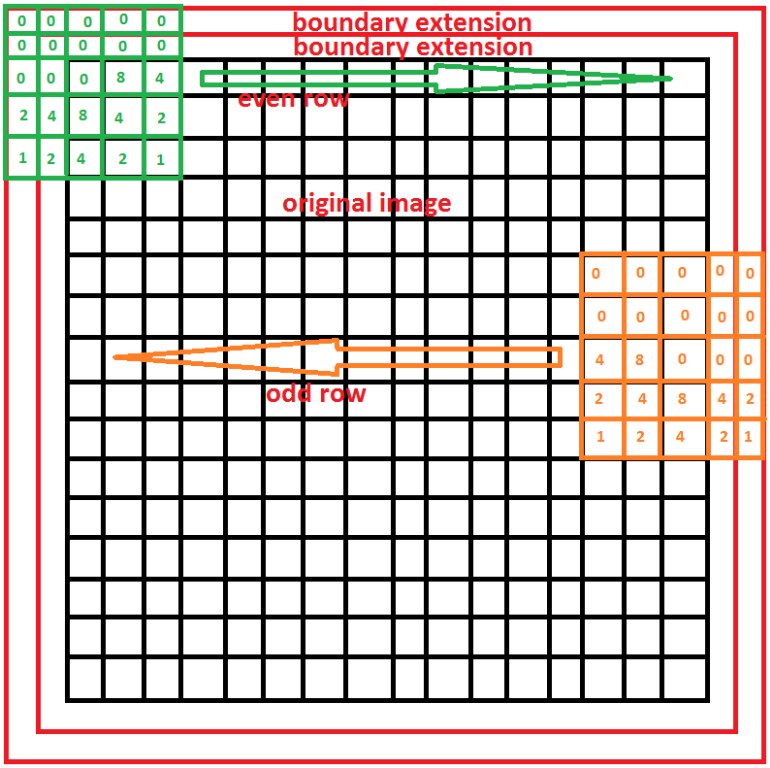


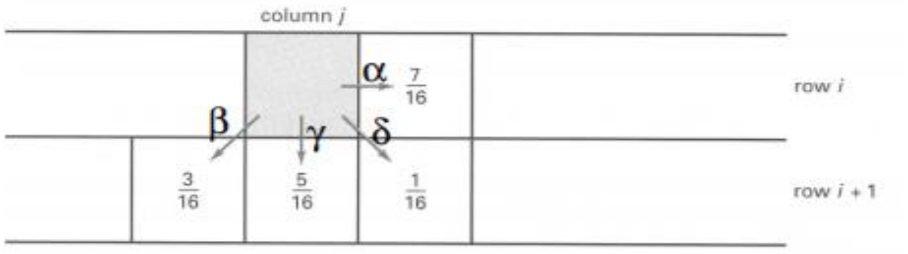
Figure 53 Stucki Error Diffusion using Serpentine Method (Matrix Values are divided by 42)

ERROR DIFFUSION ALGORITHM FOR ALL THREE MATRICES

1. Read the input Image and implement Boundary Extension on the image using Reflection Technique.
2. Create a New Output Image array or buffer Image.
3. For each pixel in the boundary extended image,
    a. binarize the pixel based on threshold value 127 and store the binarized pixel in Output data array.

$$b(i,j) = \begin{cases} 255, & if\ \tilde{f}(i,j) > T \\ 0, & otherwise \end{cases}$$ where, b(i,j) is binary output and T = 127.

    b. Calculate the quantization error by subtracting the output pixel from the corresponding boundary extended image pixel.
    c. Diffuse the error in the unprocessed pixels in the boundary extended image by correlating with the corresponding error diffusion matrix (Floyd, JJN, Stucki)
4. Save the Ouput data array and write into Ouput Image.



$$\alpha + \beta + \gamma + \delta = 1.0$$

EXPERIMENTAL RESULTS

Figure 54 Original Bridge Image

Figure 55 Halftoned Output Image Using Floyd Steinberg Error Diffusion

Figure 56 Original Bridge Image

Figure 57 Halftoned Output Image Using JJN Error Diffusion

Figure 58 Original Bridge Image

Figure 59 Halftoned Output Image Using Stucki Error Diffusion

DISCUSSION

From the above results, we observe that:

1. Blobs and Patterns are introduced in the halftoned Image when the Quantization Error in dithering is not diffused properly.
2. All Error Diffusion techniques produce better local average intensity and gives enhanced edges, retaining the detailed rendition of the image.
3. Floyd-Steinberg's error diffusion matrix gives a fine grained image, which looks well dithered from a distance. This is a faster algorithm as the error is diffused to nearby one and next pixels, unlike in JJN and Stucki in which they are diffused to more pixels, hence are slower, with more computational complexity.
4. Error Diffusion takes care of the noisy block like patches that occurred in ordered dithering at regions of very less or no gradient change, like constant water body in the bridge.
5. As JJN and Stucki diffuse the error further up than Floyd, they have a coarser output having lesser visual artifacts and better contour details in the halftoned image output than Floyd. But both are slower than Floyd-Steinberg algorithm. Stucki is slightly faster than JJN, and gives a cleaner and sharper output.
6. Floyd-Steinberg gives lowest contrast performance than JJN and Stucki. Whereas JJN and Stucki give more detailed image than Floyd.
7. The best method out of all three types implemented above (Random and Ordered Dithering) is Error Diffusion because we get a **smoother output halftoned image** and we **don't see any visual artifacts** or patterns in the output image.
8. Unlike dithering, error diffusion **doesn't cause square-box effects** (it shows a worm-like diffused pattern) and has **better contrast performance**.

Error Diffusion Method	PSNR
Floyd Steinberg Method	6.98
JJN Method	7.16
Stucki Method	7.12

OWN IDEA TO GIVE BETTER RESULTS

From literature review, I believe there are more specific Error Diffusion matrices which can give better results than what we saw above.

For Example :

1. Gradient Based Error Diffusion Dithering : It diffuses error locally, by considering the gradient of surrounding pixels hence, it will not introduce boxy effect in regions of constant gradient and diffuse error according to the need for visual fidelity .
    a. It removes the structural artifacts produced by Floyd-Steinberg by modulated randomization.
    b. It also enhances the structures by modulating error diffusion accordingly.
2. Taking a different threshold than 127 or 128 depending on the input image might help get better results.
3. Implementing Multi-Scale Error Diffusion: This is make the halftoned output more smooth and having better visual fidelity.

Which method do you prefer? Why?

Ans – I prefer Error Diffusion over Ordered Dithering because of the reasons as explained above. Overall Better output that preserves details.

And out of the Three Error Diffusion matrices, I'll prefer Stucki for it's detailed and clean output.

COLOR HALF-TONING WITH ERROR DIFFUSION

ABSTRACT AND MOTIVATION

In color-halftoning, we intend to convert or quantize the RGB colors of 256*3 types to 8 colors (2^3, binarizing each channel). Here, we implement two methods of color-halftoning.

1. **Separable Error Diffusion:** Here, we diffuse each of the three color planes independently i.e. we separate the image into different channels and then use basic error diffusion matrix like the Floyd Steinberg matrix to diffuse the error in the forward direction for each individual channel.

$$W = (0,0,0),\ Y = (0,0,1),\ C = (0,1,0),\ M = (1,0,0),$$
$$G = (0,1,1),\ R = (1,0,1),\ B = (1,1,0),\ K = (1,1,1)$$

In this method, we see the above 4 complimentary pairs of colors and use these to quantize the color image to get halftone output.

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

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

Figure 60 Relation between RGB and CMY

2. MBVQ-based Error diffusion : Key ideas behind MBVQ :

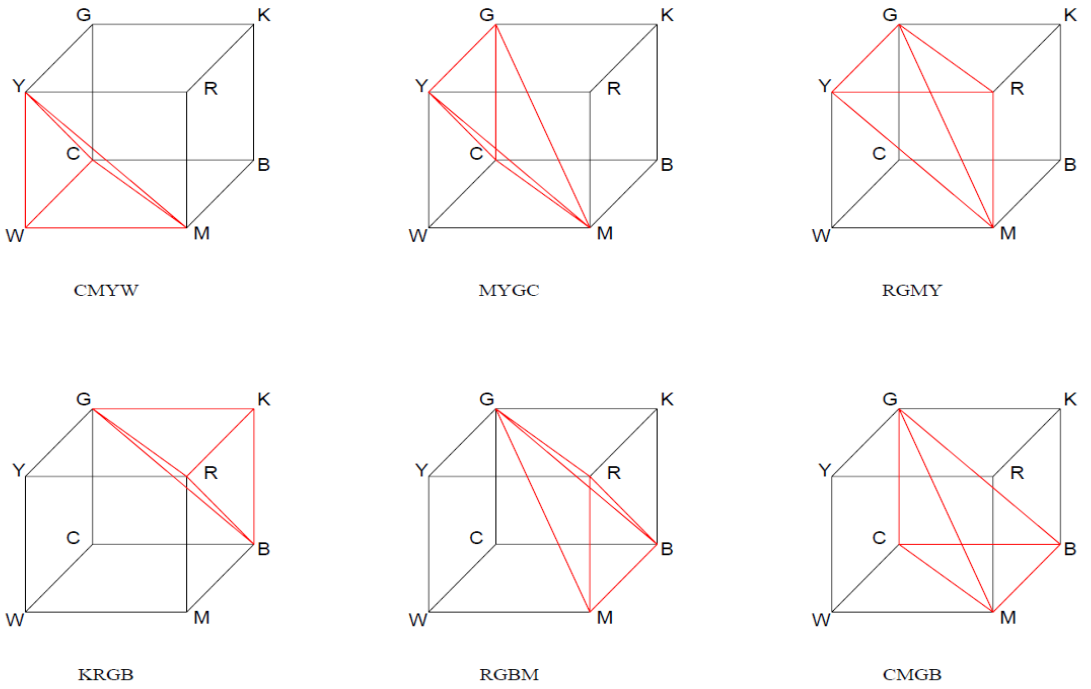


Figure 61 RGB Partition into six tetrahedral volumes

Key Ideas:

1. It follows the minimal brightness variation criterion design rule: The Minimal brightness variation criterion explains “To reduce halftone noise, select from within all halftone sets by which the desired color may be rendered, the one whose brightness variation is minimal”.
2. So, here, we separate the RGB color space into Minimum Brightness Variation Quadruples (MBVQs) i.e. we render each input color using one of the six complimentary quadruples each with minimal brightness variation: RGBK, WCMY, MYGC, RGMY, RGBM and CMGB.
3. So, here we assume that the RGB cube can be rendered using 8 basic colors. MBVC follows the idea that human visual system acts as a low pass filter when presented with high frequency patterns and is more sensitive to change in brightness than to changes in chrominance.

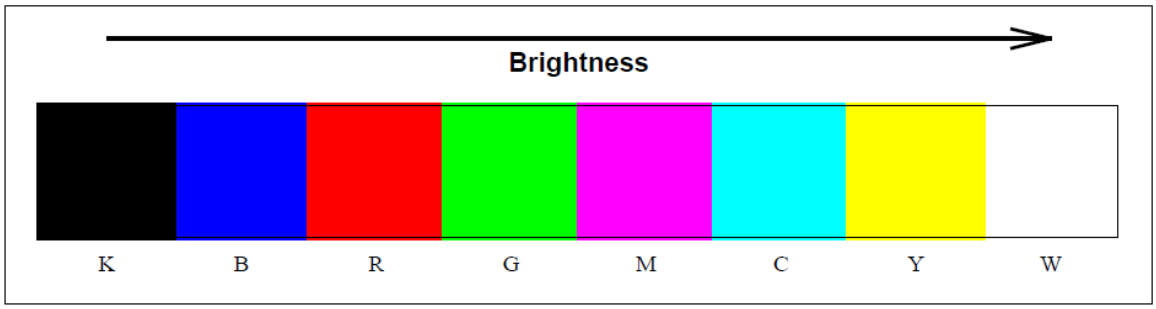


Figure 62 Brightness scale of colors

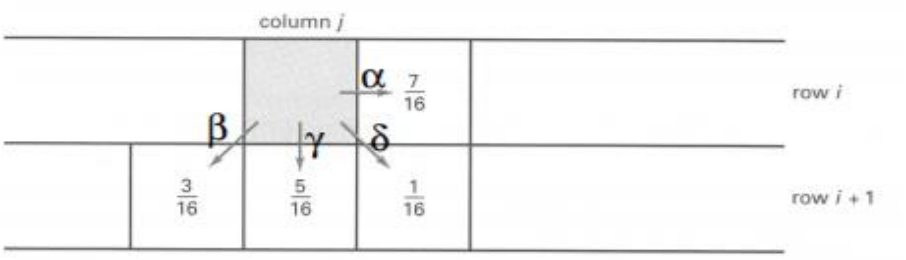
APPROACH AND PROCEDURE

ALGORITHM FOR SEPARABLE ERROR DIFFUSION:

1. Read the input Image and implement Boundary Extension on the image using Reflection Technique.
2. Create a New Output Image array or buffer Image.
3. Convert the RGB image into CMY using their relationship as explained before.
4. For **each pixel of each channel**, in the boundary extended CMY image,
    a. binarize the pixel based on threshold value 127 and store the binarized pixel in Output data array.

$$b(i,j) = \begin{cases} 255, & if\ \tilde{f}(i,j) > T \\ 0, & otherwise \end{cases}$$ where, b(i,j) is binary output and T = 127.

b. Calculate the quantization error by subtracting the output pixel from the corresponding boundary extended image pixel.
c. Diffuse the error in the unprocessed pixels in the boundary extended image by correlating with the corresponding Floyd error diffusion matrix.



$$\alpha + \beta + \gamma + \delta = 1.0$$

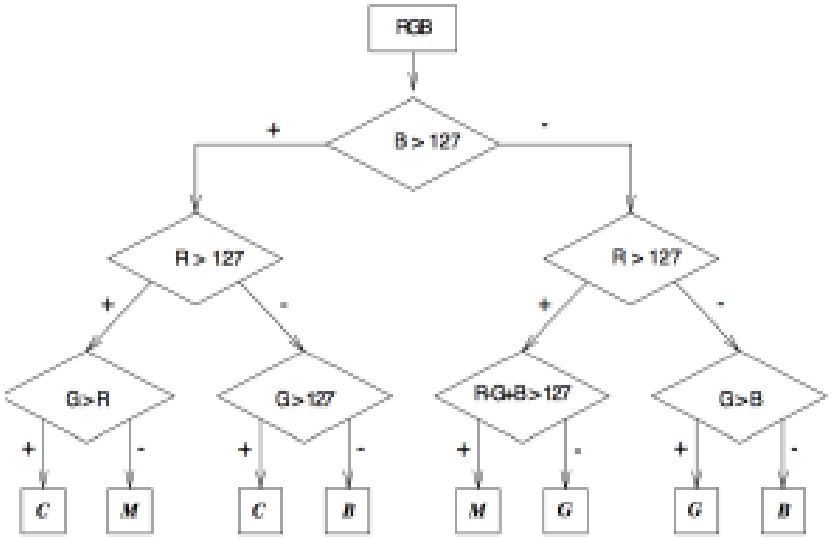
5. Convert the Image back from CMY to RGB using their relationship.
6. Save the Output data array and write into Output Image.

ALGORITHM FOR MBVQ-BASED ERROR DIFFUSION:

- For each pixel in the Image:
    a. Find the quadrant of input pixel using MBVQ function i.e. MBVQ(RGB(i, j)). Pseudocode :

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

b. Find the vertex, v belonging to MBVQ which is closest to RGB(i, j) + e(i, j). Following Pseudocode e.g. for CMGB Quadruple,



c. Compute the quantization error RGB(i, j) + e(i, j) – v
d. Distribute the error to future pixels using Floyd Steinberg Error diffusion matrix.

EXPERIMENTAL RESULTS

Figure 63 Original Bird Image

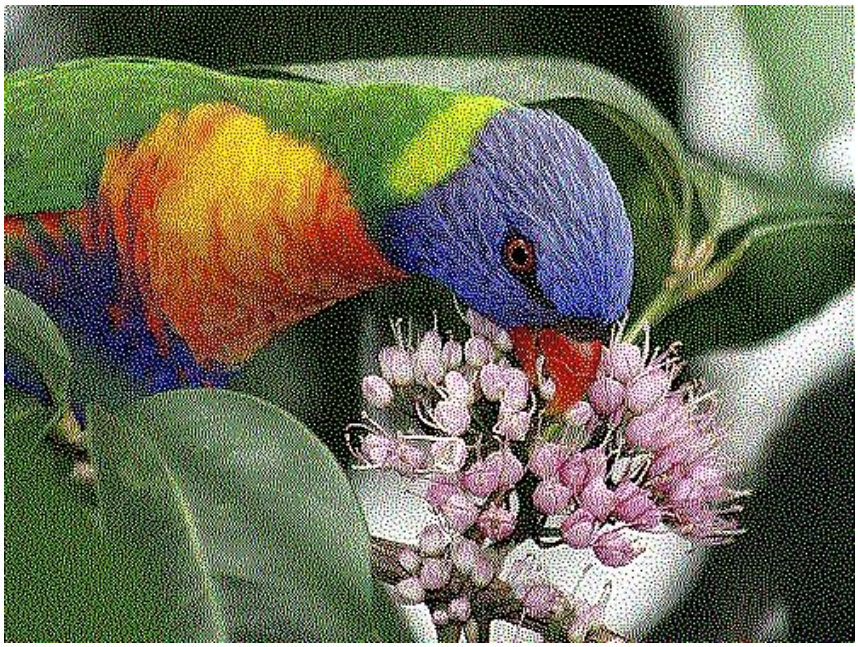


Figure 64 Color-Halftoned Output Image Using Separable Error Diffusion

Figure 65 Original Bird Image

Figure 66 Color-Halftoned Output Image Using MVBQ Diffusion Method

DISCUSSION

I have described the Key Ideas of MBVQ in <u>Approach and Motivation</u> Section Above.

Figure 67 Separable Enlarged Output

Figure 68 MBVQ Enlarged Output

From the above results of color half-toning we can observe that:

1. The color placement pattern is noticeable more Separable Error Diffusion output than in MBVQ output.
2. The local average color obtained is the final desired color in both the outputs.
3. The colors used should reduce the noticeability of the pattern but the color patch pattern in more visible in Separable Error Diffusion output.

Method	PSNR
Separable Error Diffusion	2.7514
MBVQ	2.7616

Why is MBVQ method better than Separable Error Diffusion method:

Ans - Because in MBVQ method, the chosen approximation color is the color among Black, White, Red, Green, Blue, Cyan, Magenta and Yellow that has the least difference in brightness. With LPF of human eye involved, this causes **reduction in noticeability of patterns** hence giving good halftoned image output.

What is the main shortcoming of this approach?

- Separable Error Diffusion **doesn't exploit the inter-color correlation**, hence leading to color artifacts and poor color visual rendition.
- Further, for high sharpness and low artifacts, error diffusion doesn't converge. It shows patchy and blurry output and there is noticeability of the pattern.

- Generalization of monochrome overlooks that the colored pixels are not equally bright.

MBVQ overcomes these disadvantages.

At the end, MBVQ is preferred over Separable method because it reduces the halftone noise of the Image.

MBVQ is computationally more intensive algorithm than separable diffusion method because it retains local averaging and reduces noise.

REFERENCES:

1. http://www.tannerhelland.com/4660/dithering-eleven-algorithms-source-code/
2. http://www.ece.ubc.ca/~irenek/techpaps/introip/manual04.html
3. https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-2-error-diffusion/
4. https://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm
5. http://www.ece.ubc.ca/~irenek/techpaps/introip/manual04.html
6. https://engineering.purdue.edu/~bouman/ece637/notes/pdf/Halftoning.pdf
7. https://en.wikipedia.org/wiki/Dither
8. https://engineering.purdue.edu/~bouman/ece637/notes/pdf/Halftoning.pdf