CHARU SINGH

USCID:2773417243

charusin@usc.edu

Submission date: 2/16/2020

Homework2

Sobel Edge Detector:

Motivation:

In order to determine edges in an image we use various algorithms. We exploit various properties such as the gradients in order to determine edges present.

Approach:

Whenever we want to detect an edge in an image, we look for discontinuities and derivatives can be used to detect them

Sobel Operator:

In order to detect edges in an image we use a 2D derivative mask which is known as Sobel operator. It is a first order derivative-based edge detector.

Following masks are used in x and y direction:

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

Vertical

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

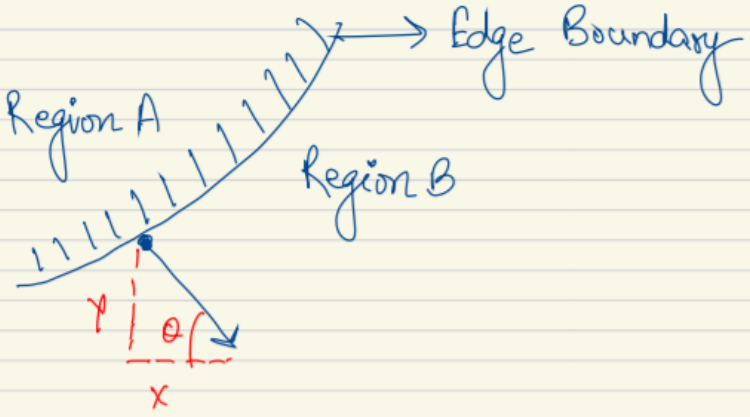
Horizontal

Limitation:

- Localization is poor: there can be number of edges in place of only one edge.
- Can miss some of the edges which are neither vertical nor horizontal.

Procedure:

- First, we will create both vertical and horizontal masks.
- We will pass masks as argument to function called masked element in order to perform convolution in both directions that is calculating derivative in both directions.
- Next combine the derivative in both x and y directions by using below formula:



GRADIENT

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

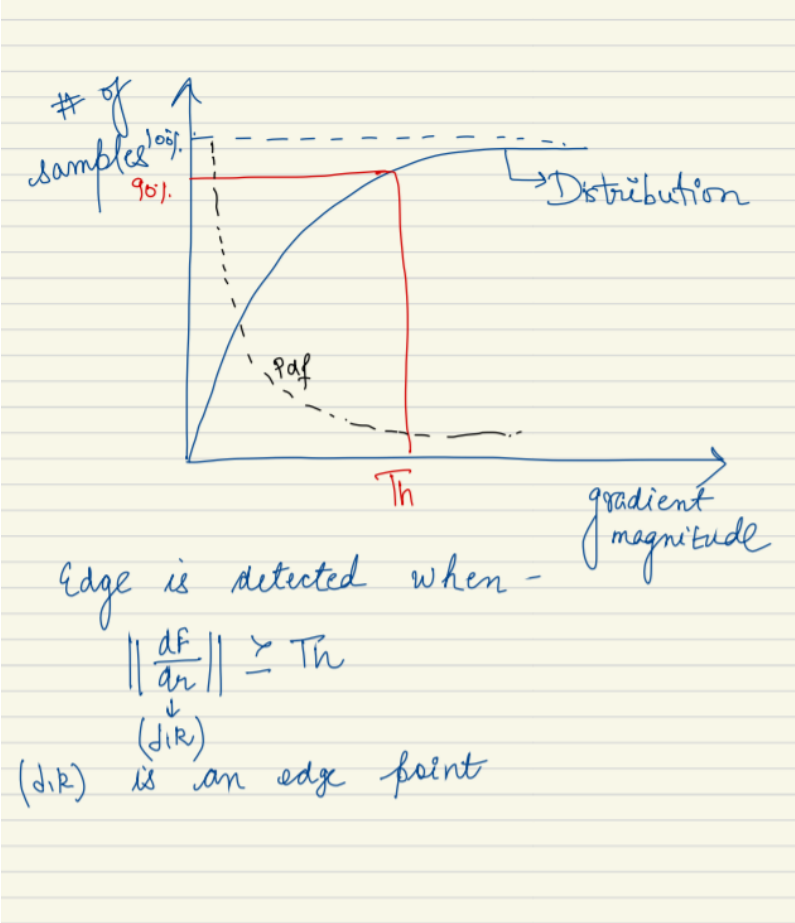
Magnitude

$$\nabla f = \sqrt{g_y^2 + g_x^2}$$

Orientation

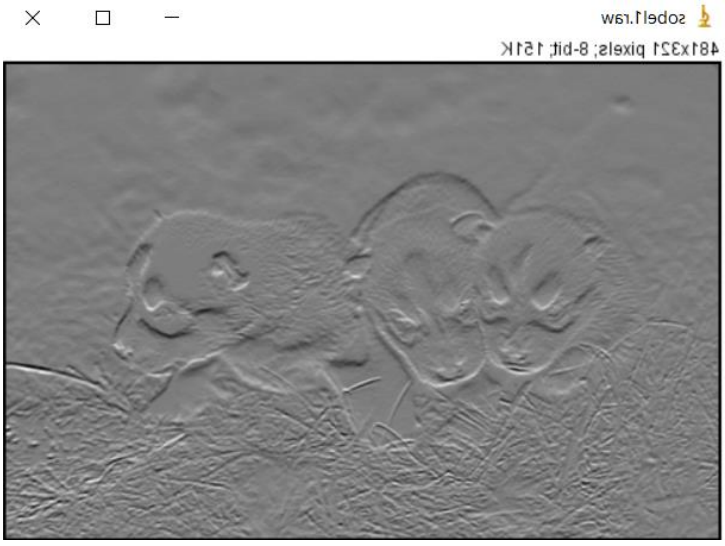
$$\theta = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

How to choose thresholding:



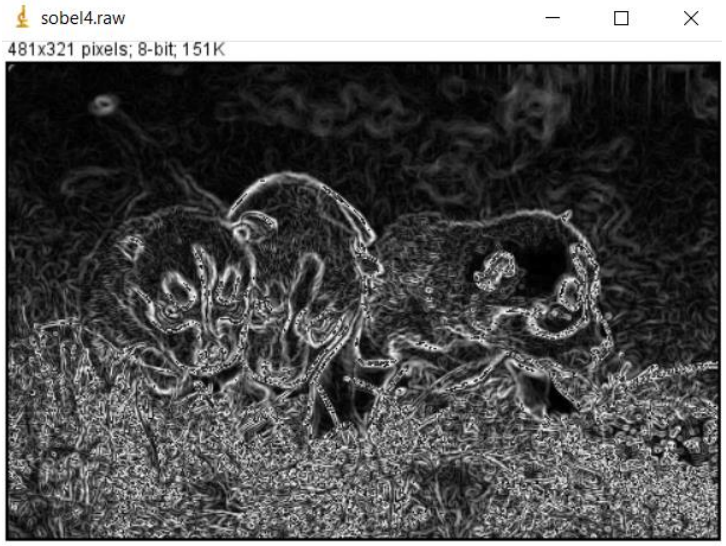
RESULTS:

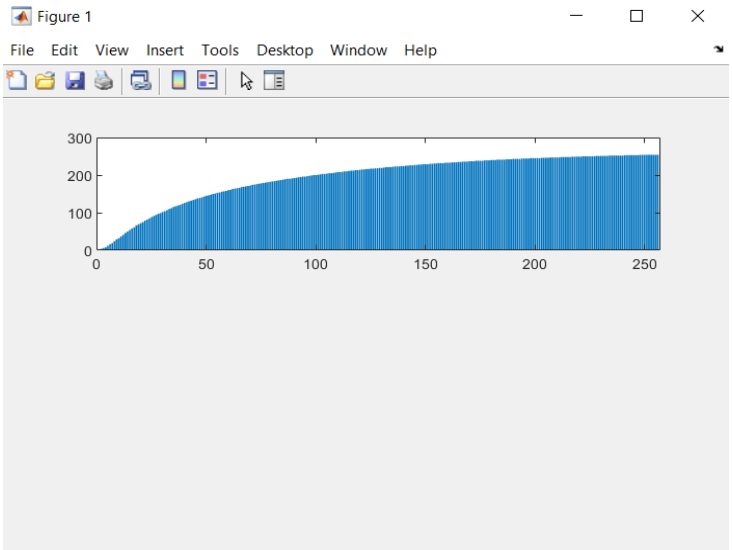
Derivative in x direction:



Derivative in y direction:

Normalized gradient magnitude map:



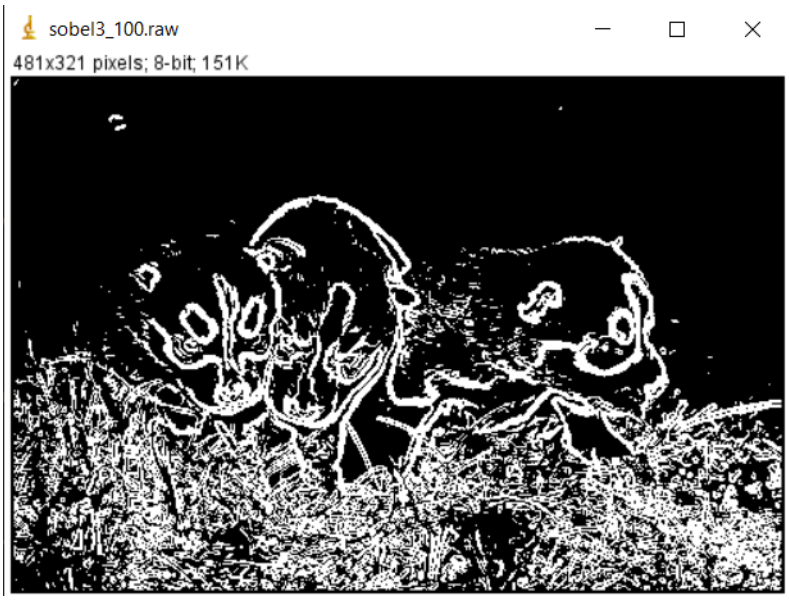


Above figure is cdf of the normalized gradient. We will take around 80% of the maximum value to get the best output.

Edge Map: Threshold:200

Edge Map: Threshold:150

Edge Map: Threshold:100



Derivative in X direction:

Derivative in x direction:

Normalized gradient magnitude map:

Edge Map: Threshold:200

Edge Map: Threshold:150

Edge Map: Threshold:100

Conclusion:

We observe that edge detection gives better performance in gallery image as compared to dogs because the dog image has lot of texture, so the algorithm considers the non-edges as true edges**.**

Also, we observe that the best threshold is around 100, value too high such as 200 misses some edges.

(b). Canny Edge Detector-

Motivation:

Thinning of edges by removing pixels that are not part of an edge.

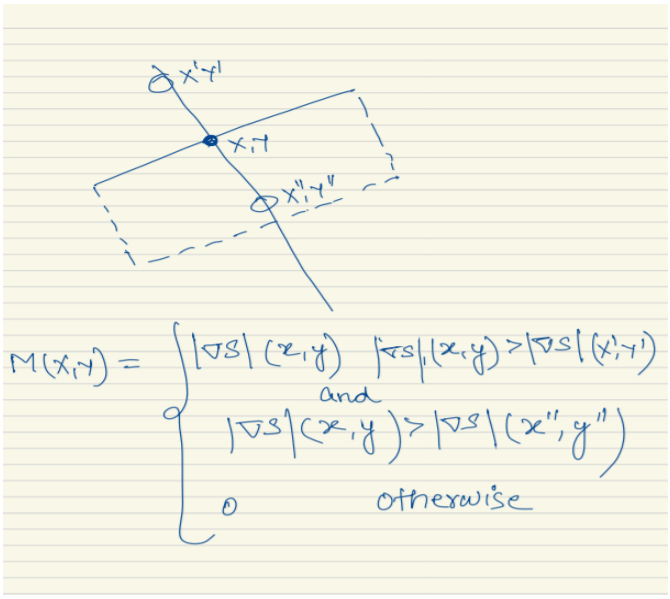
Utilize post processing to refine edge maps. Instead of isolating each point we try to look neighbor of each point and see whether you can connect them together.

If you cannot them together it means, it's a noise.

You can connect to other points using some information, orientation etc.

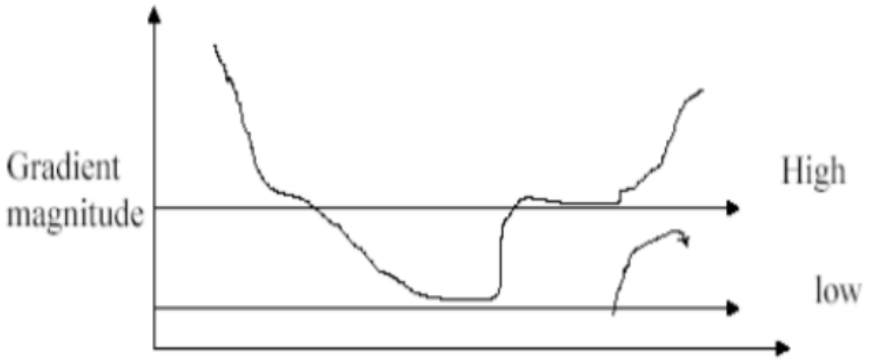
Steps:

1. *Convolution with derivative of Gaussian*:
   This is dome in order to remove high frequency noise.
2. *Non maximum suppression*
   Suppresses the pixels in gradient magnitude image which are not local maximum. Edge band is being converted into edge line by the process of thinning. If an edge point is detected and more edge point need to be located, there is a need to lower the threshold so that instead of getting isolated point, a band of edge is obtained containing x,y , x'y' and x''y'' in figure shown below: Consider the normal direction and find the maximal. Our goal is to find out the gradient with largest magnitude and remove the surrounding ones.



3. *Hysteresis thresholding*:
   Choose two thresholds: high and low Region above high threshold is defined as edge, below threshold is not an edge and between high and low we must determine whether it connects to edge or not.
   Drawbacks:
   Sensitive to texture regions.

Canny recommended upper: lower ratio as 2:1 and 3:1



(3) Edge maps:

Procedure:

We use MATLAB inbuilt function edge detector to perform canny edge detection.

```
BW = edge(I,method,threshold)
```

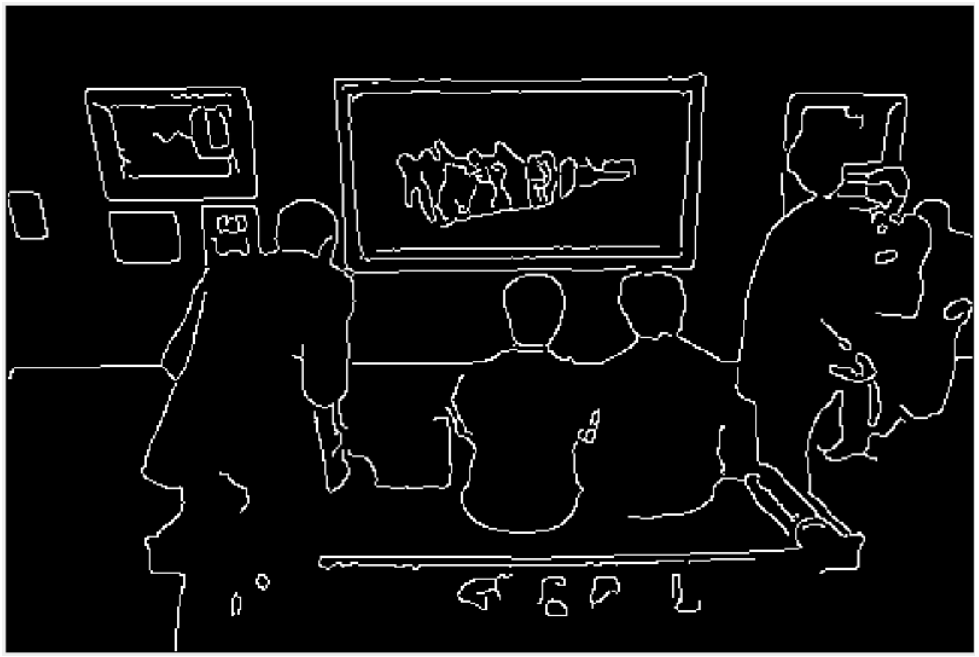
- It looks for local minima for an image I to find the edges. the edge function determines the derivative of a gaussian filter which uses 2 threshold, high and low in order to detect strong and weak edges. Because it uses two thresholds in order to determine there are less chances that it determines noise as an edge and can detect true weak edges efficiently.
- NOTE: Here the threshold parameter is a 2-element vector.

RESULTS:

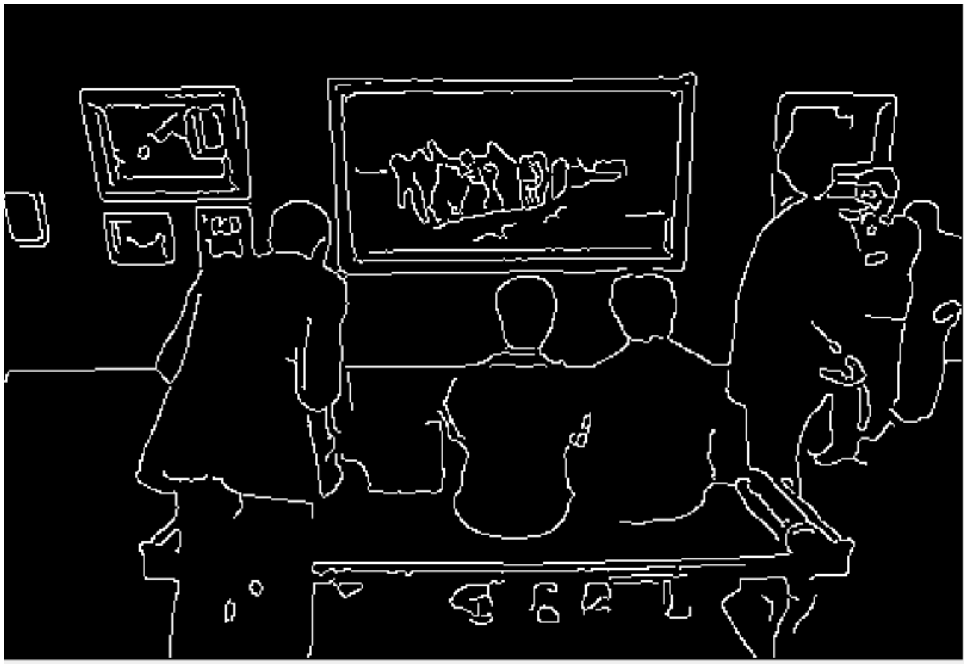
We have chosen 3 combinations of low and high threshold that is:

Upper: lower- 3:1 and 2:1.

1. Low threshold=0.1 High threshold=0.3



2. *Low threshold=0.1 High threshold=0.2*



3. Low threshold=0.05 High threshold=0.1

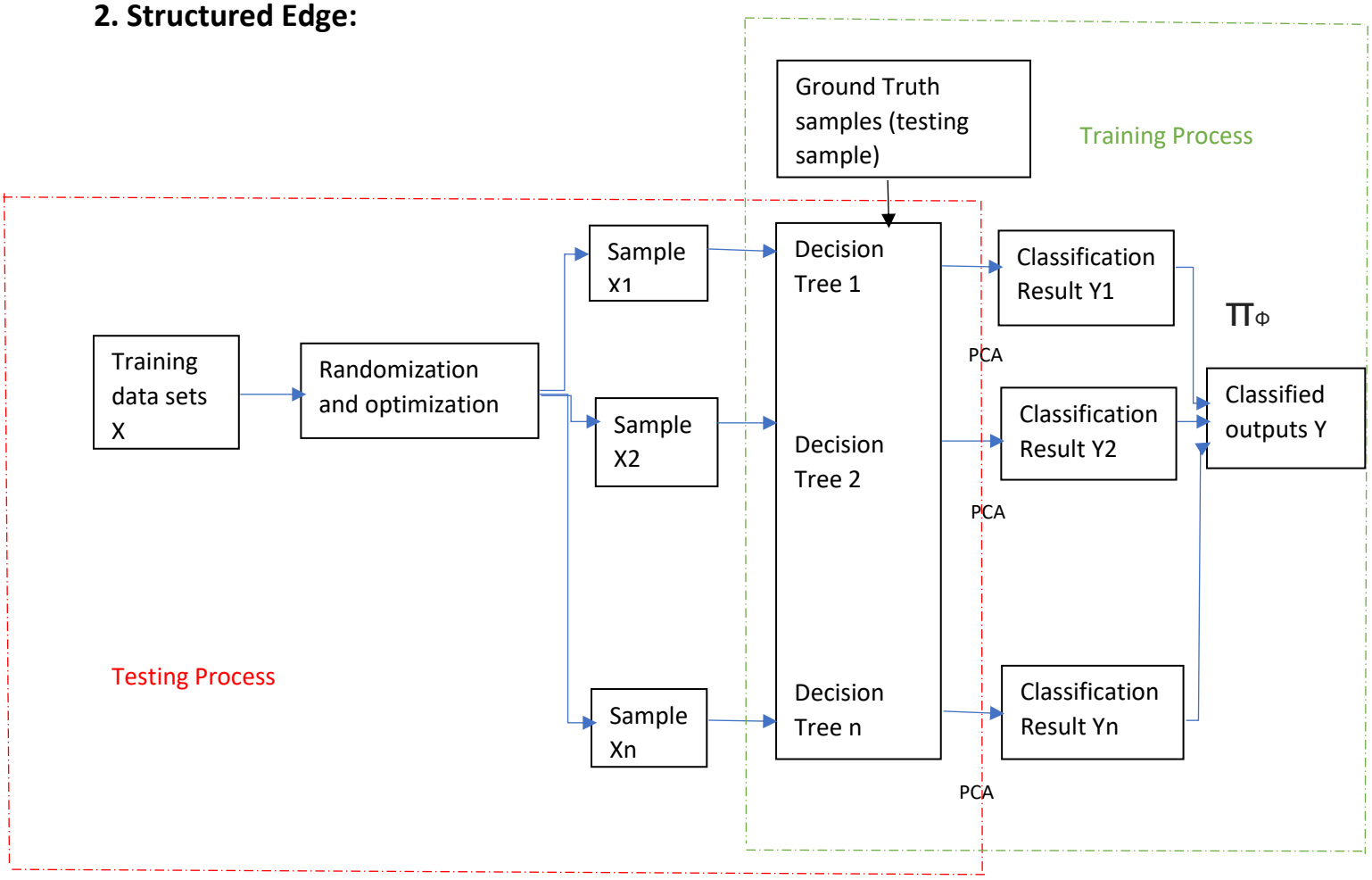
Conclusion:

We can see that results are better that is more fine edges are defined when we chose the ratio of upper: lower as 2:1.

We can also see that Gallery image produces more better results as compared to Dogs because the Dogs image contains texture pattern that is the grass which are classified as edges but in real, they are not.

Also lowering the thresholds values gives better result that is more edges are defined correctly (refer figure above).

2. Structured Edge:



Our goal is to label each pixel either 0 or 1 indicating that it contains an edge or not.

Learning approach:

It predicts a structured 16*16 segmentation mask from 32*32 image patch. We augment each image patch with information of all channels resulting in a feature vector $x \in R^{32*32*K}$

It yields 2^256 structured output that is binary decision for every point. Because 16*16 edge is structured we really cannot get the edges at every position 0.1 there is a random pattern not necessarily be an edge or contour pattern. Therefore, mapping from Y->Z is being done where Z encodes the information

whether every pair pattern in Y belong to the same segment or not (C(256,2))=32460 dimension. 256 dimensions in Z then reduces to 5 dimensions by PCA. K mean clustering is used to classify into two classes for each node.

Classified label is selected by majority voting among all the classified labels.

Decision tree:

Train:

We build the tree based on features of training data and measure the information gain to decide feature for each node.

Test:

Once we have the tree finalized, we just pass samples to traverse the tree to get final prediction.

2.1 Random Decision Forest:

- ***Decision tree*** is denoted by $f_t(x)$ which classifies samples x belongs to X by branching down the tree left/right recursively until leaf node is reached.
- Each node we select one feature as a splitting criterion and at each node we have final decision that is labels of each node of data set. Each node in tree is represented by j which is associated with a **binary split function.**

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

if it is 0 then x is sent to left else right. Input sample x is given to the decision tree and predicted output is stored at the leaf, $y \in Y$ which are called labels

Advantages:

1. Fast classification
2. Speed of classifier is directly proportional to depth of tree which is generally not very big.
   - Decision Forest is formed by ensemble $f_t(x)$ independent trees into a single output by using ensemble model. the output depends on Y that is mainly decided by majority vote of labels for classification.

2.1 Training Decision Trees:

Classification of trees (training decision):

Our goal is to find parameter θ_j of the split function for a given node j and training set ($S_j \subset X*Y$).

To decide which feature criterion is used at each node in training:

1. Measure the information gain (purity/entropy) given by:

$$I_j = H(\mathcal{S}_j) - \sum_{k \in \{L,R\}} \frac{|S_j^k|}{|S_j|} H(S_j^k)$$

   Where

$$H(\mathcal{S}) = \sum_y p_y(1-p_y)$$

   H(S) denotes Shannon entropy and p_y represents fraction of data sets in S with data labeled as y.
2. Choose one that gives maximum information gain.

As a result, it will maximize information gain and minimize entropy

Each tree in a recursive manner is trained independently

- *Randomness and optimality*:
  Individual decision trees exhibit high variance and tend to overfit. Decision trees compensates this by training multiple uncorrelated trees and then combining their output, but the accuracy of individual is sacrificed.
- *Structured Learning:*
  Motivation:
  In traditional classification approach the data samples are assigned to a single label without considering the dependencies among the classifiers. Therefore, this problem is addressed by making the classifier aware of local topological structure of the output label space.

2.3 Structured Random Forests:

Here we will consider $x \in X$ which will represent an image patch and $y \in Y$ will encode corresponding local image segmentation mask.

Challenges:

1. Output spaces are highly complex and high dimensional.
2. Information gain might not be well defined on structured labels.

Goal:

All similar structured labels y are to be assigned to same discrete levels so we have to map all structured label $y \in Y$ at a given node to discrete set of labels $c \in C$ (C=1…..k).

Next information gain is calculated over C, given discrete labels C so there is no need to calculate information gain over structured labels Y.

2.4 Intermediate mapping:

Motivation:
Measurement of information gain relies on similarity of Y but in structured output spaces measurement of similarity over Y is not well defined.
Therefore, we use an intermediate stage where we map Y to Z in which the distance will be measured.
It's a 2-stage approach: Y->Z then Z->C.

Procedure:

- Compute dissimilarity over $y \in Y$ by finding Euclidean distance in Z.
  Note: here labels $y \in Y$ are 16*16 segmentation masks.
- We define z as:

$$z = \Pi(y)$$

  z is a binary vector that tells us that whether every pair in y belongs to the same segment or not. In Z space distance is easily measured.
- As dimension of Z is high dimensional so there is a computationally complexity. For example, there are C(256,2)=32640 pixels pairs in 16*16 segmentation mask. To ameliorate this, we will reduce the dimension of Z. we will use Principal Component Analysis (PCA) to reduce from 256 to at most 5 dimensions.
  HOW TO OBTAIN C GIVEN Z:

There are two approaches to find discrete label set C:

1. k means clustering: cluster z into k clusters.
2. Quantize z based on the top PCA dimensions.

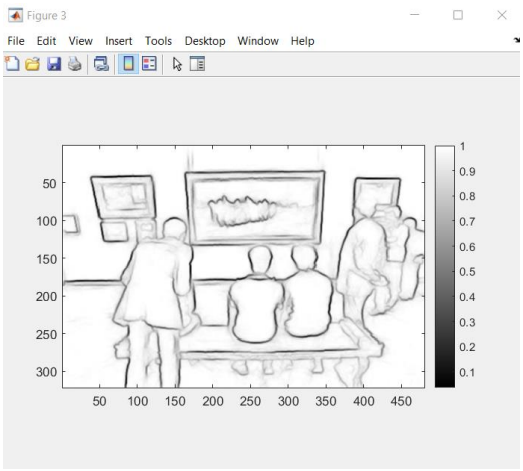
2.5 Ensemble Model:

Our goal is to combine n labels y1…. y_n into a single prediction for training(to set leaf labels) as well as testing(to merge predictions). This model is unable to synthesize new labels so any prediction $y \in Y$ should be observed at time of training. So, we will go for domain specific models. Hence for each image patch obtain edge maps independently and then merge overlapping predictions by performing averaging.

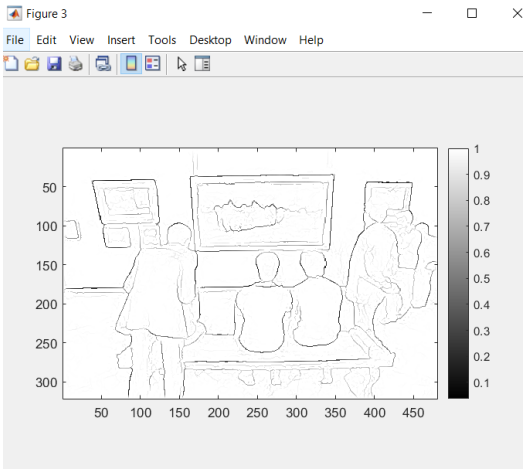
In structured edge they average among the overlapping region among different patches. For each patch we have predicted structured patch or binary patch. The result is relatively smooth because we average among the predictions of overlapping regions. and the textures which were visible in sobel detection is not there.

RESULTS:

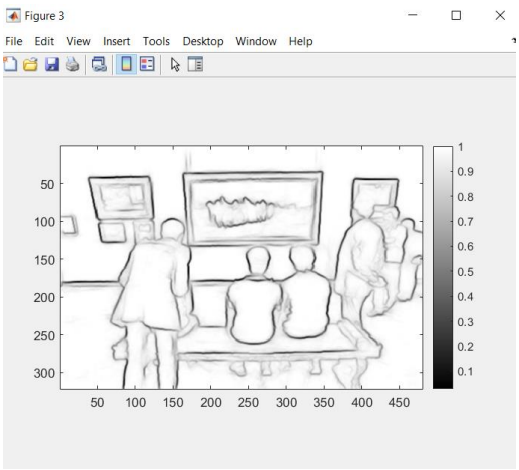
1. Multiscale=0, Sharpen=2, nTreesEval=4, nThreads=4, nms=0/1



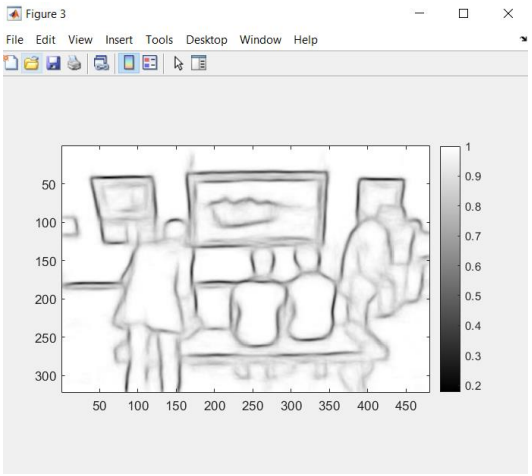
Multiscale=0, Sharpen=10, nTreesEval=10, nThreads=4, nms=0



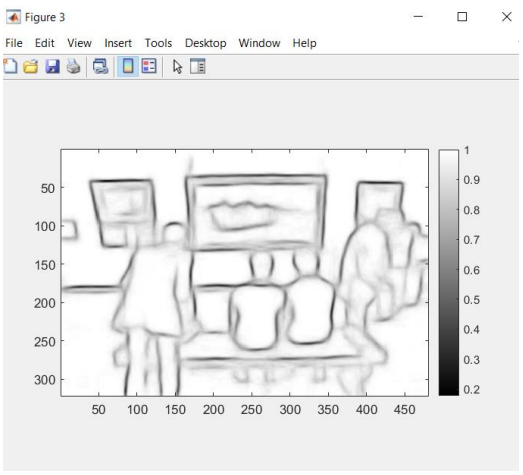
Multiscale=0, Sharpen=0.5, nTreesEval=10, nThreads=4, nms=0



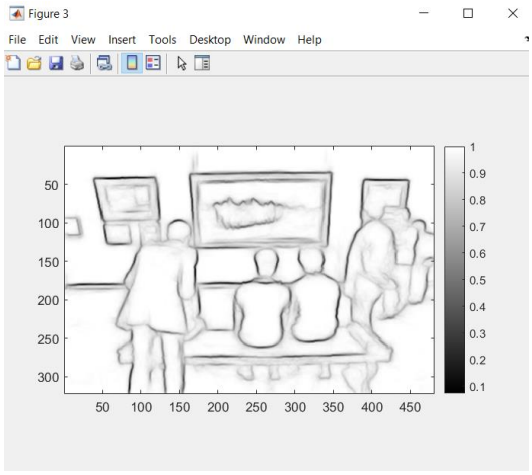
Multiscale=0, Sharpen=0.1, nTreesEval=10, nThreads=4, nms=0



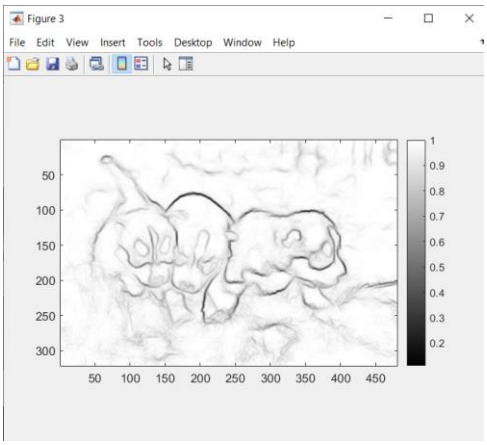
Multiscale=0, Sharpen=1.5, nTreesEval=10, nThreads=4, nms=0



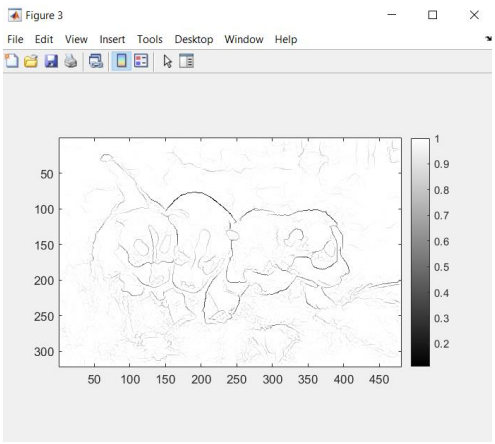
Multiscale=0, Sharpen=2, nTreesEval=4, nThreads=4, nms=0



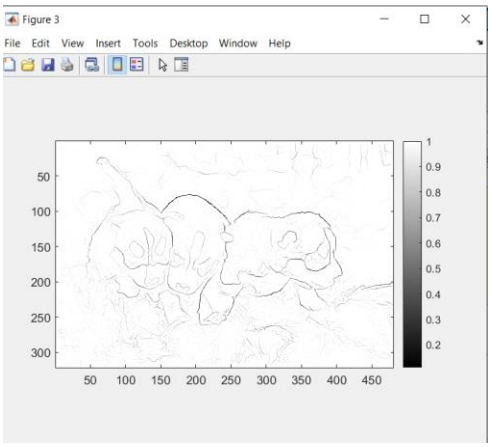
Multiscale=0, Sharpen=2, nTreesEval=4, nThreads=4, nms=1

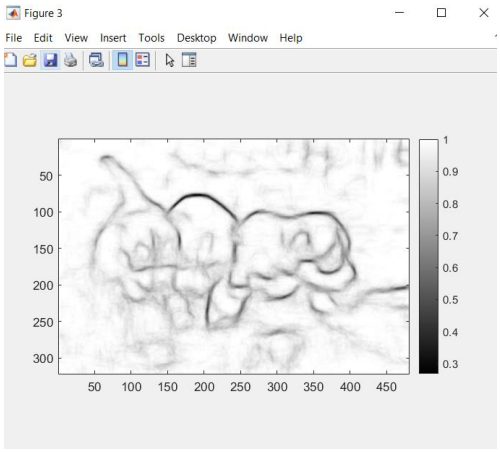


Multiscale=0, Sharpen=100, nTreesEval=4, nThreads=4, nms=1



Multiscale=0, Sharpen=0, nTreesEval=4, nThreads=4, nms=0





Conclusion:

Chosen parameters are as follows:

Sharpening,nTreesEval, nThreads, nms

where nTreesEval: number of trees participating in the decision making, nms: non maximal suppression.

Its been observed that when enabling nms it results in thinning of edges, finer image.

Changing nTreesEval does not make any large impact on the output. It just increases the processing time.

Sharpening gives best result at 2 if we increase it still remain same, saturation occurs but we make it around 0.5 we are getting blur image.

Comparison between Canny and Structured edge:

Structured edge has low computational cost and high ability to select effective features. Enough diversity of trees could avoid the overfit problem.

In canny edge we apply gaussian filter, so it smoothens image therefore the sharpening effect is not observed whereas in structured edge it provides sharp edge this method analyzes the inherent structure of edge patches and moreover it is computationally efficient.

Performance Evaluation:

To evaluate the performance of a generates edge map we must find out the error. Each pixel in an edge map is being mapped to one of the four classes:

i. True Positive: correctly classified edge pixels

ii. True Negative: correctly classified non edge pixels
iii. False Positive: The algorithm falsely identifies pixels as edge, but they are not.
iv. False Negative: pixels that are edge are being missed

Using F measure the performance of edge detection algorithm can be measured:

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

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

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

2. Gallery image will provide better results than Dogs because we can see that there are lot of textures in dog image so the greater number of pixels will fall under false positive thereby decreasing the precision value and decrement in the F measure.

RESULTS:

Algorithm	Image	Ground truth	Precision	Recall	F measure
Sobel	Dogs	1	0.0829	0.7267	0.1488
Threshold=70		2	0.0836	0.7269	0.1500
		3	0.0831	0.7261	0.1492
		4	0.0828	0.7263	0.1486
		5	0.0835	0.7266	0.1497
Algorithm	Image	Ground truth	Precision	Recall	F measure
Sobel	Gallery	1	0.1492	0.7895	0.2509
		2	0.1490	0.7896	0.2507
		3	0.1491	0.7896	0.2508
		4	0.1490	0.7895	0.2507
		5	0.1497	0.7896	0.2516

Algorithm	Image	Ground truth	Precision	Recall	F measure
Canny	Dogs	1	0.5309	0.7718	0.6290
Thresh=0.1		2	0.5304	0.7720	0.6288

Thresh=0.3		3	0.1863	0.6928	0.2936
		4	0.1855	0.6931	0.2927
		5	0.1873	0.6942	0.2951
	Gallery	1	0.5362	0.7703	0.6323
		2	0.5358	0.7706	0.6321
		3	0.5359	0.7703	0.6321
		4	0.5359	0.7707	0.6322
		5	0.5383	0.7706	0.6338

Algorithm	Image	Ground truth	Precision	Recall	F measure
Structured Edge	Dogs	1	0.7048	0.2584	0.3782
		2	0.7057	0.2581	0.3780
		3	0.7049	0.2570	0.3767
		4	0.7046	0.2597	0.3784
		5	0.7056	0.2852	0.3781
	Gallery	1	0.5362	0.7703	0.5564
		2	0.5358	0.7706	0.5563
		3	0.5359	0.7703	0.5566
		4	0.5359	0.7707	0.5565
		5	0.5383	0.7706	0.5566

3.The machine generated edge map in different edge detection algorithms is being compared to the ground truth edge map provided by human.

Both precision and recall are inversely proportional that mean if you increase either of them other will decrease therefore it will lower the F value. So, if Precision is higher than recall it is not possible to get high F measure.

According to the formula below:

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

When precision is equal to recall then F value is 1.

Problem2. Digital Halftoning-

Motivation:

To measure performance of different algorithms generally good hardware gives high dpi therefore performance gap between good and poor halftoning algorithm is small but hardware is expensive on the other hand poor hardware gives low dpi so algorithm plays an important role

Goal: Rendering the illusion of gray level images on two tone display devices. For example: Printers

A darker region represents denser black pixels and white region represent sparser black pixels.

The term half tone means there are two levels 0(dot) or 1(no dot).

Fixed Thresholding: It is a two-level bit quantization that is 0 or 1, it converts gray scale to binary image.

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

Here f(i,j) is a gray scale image and b(i,j) is a binary image output.

Random Thresholding:

To each input pixel of the gray scale image we add random number ranging between 0-255 and perform quantization according to below equation**.**

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

There are two commonly used methods:

1. Dithering:
   Motivation:
   Human visual system doesn't sense the individual pixel rather it averages a region around the pixel.

Abstract:
Our goal is to create an illusion of using many gray levels but in real using only two 0 or 1. In order to reduce the dependence between original image and quantization error we will be using a variable threshold.

Noise N(j,k) is needed because generally we are not able to see variations in a particular region we need noise to break the monotonicity . Here noise will behave like a variance

Procedure:

- Every pixel in the input image is quantized according to the threshold of the dither pattern shown below.
- Turn on the pixel in a specified order for a constant gray level patch this creates a perception of continuous variation of gray.
- We consider a $2^n * 2^n$ window so that for every constant value you will get a part of it as '0' or '1'.
- Index matrix:
  A N*N matrix determines the order to use

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

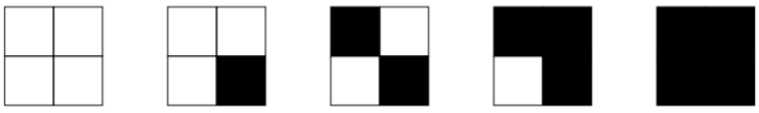
  Lower the index lower will be the threshold

  0,1……..4^n-1

  If n=1 0,1…..3

  If n=2 0,1……15

  Below shown is the order of how the pixels are turned on-



  0 1 2 3 4

- Implementation of dithering via thresholding:

  The index matrix is converted to a threshold matrix with pixel values being normalized between 0 and 255 using below formula:

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

$N^{2=}$ total number of pixels in threshold matrix
i,j= location in matrix

- As we know the that an image is much larger than the threshold matrix, the threshold matrix is repeated periodically all over the image by using formula below:

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

where T(i mod N,j mod N) is a periodic replication.

- Bayer's index matrix can be used recursively by using formula below:

$$I_{2n} = \begin{bmatrix} 4*I_n+1 & 4*I_n+2 \\ 4*I_n+3 & 4*I_n \end{bmatrix}$$

This matrix will give us dispersed dot screens.
For example:

1	2
3	0

2×2

5	9	6	10
13	1	14	2
7	11	4	8
15	3	12	0

4×4

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

8×8

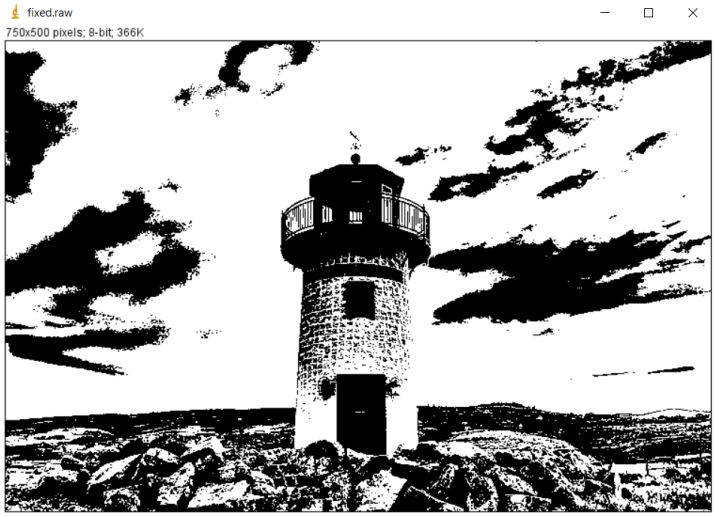
Pros:

- Over larger area this method yields finer amplitude quantization.
- Within smaller area it retains good detail interpretation.
- There is no tradeoff between number of gray levels and resolution.
- In region having K dots, the K thresholds are distributed uniformly.
- Textures which represent variations in gray level have low visibility.

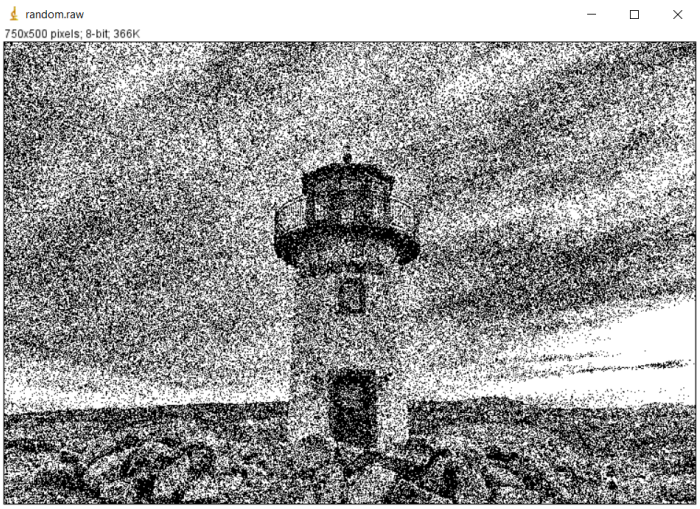
Shortcomings:
In output there should be a constant pattern but now we have a texture like visual pattern.

RESULTS:

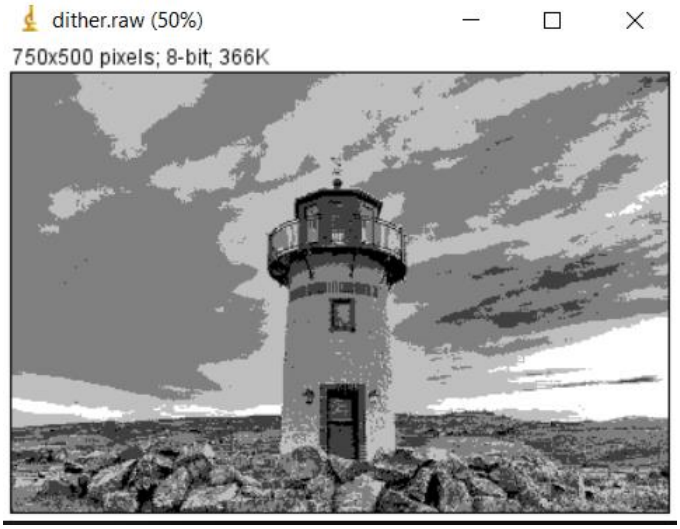
1. **Fixed Thresholding:**



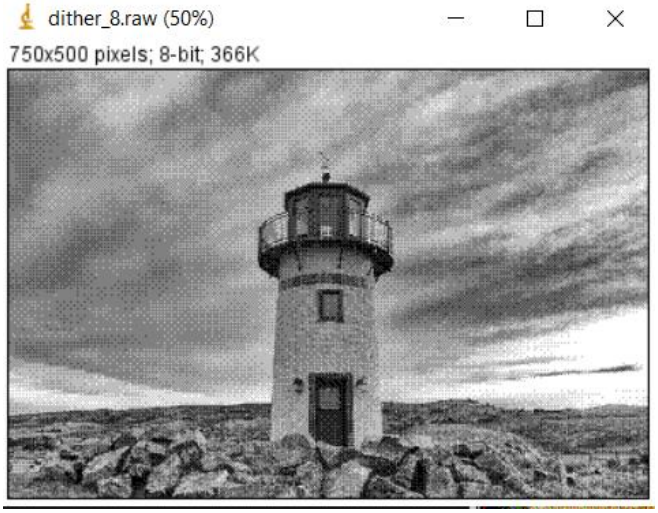
2. **Random Thresholding:**



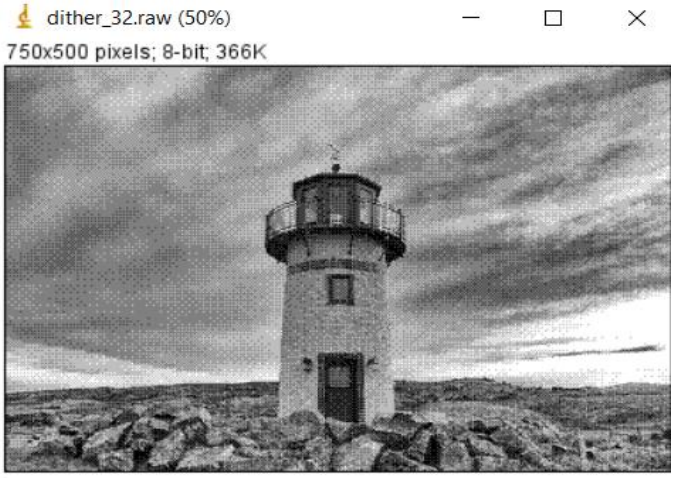
3. **Dithering _2:**



4. **Dithering_8:**



5. **Dithering_32:**



Conclusion:

There is an artifact known as false contouring that is binary image is not shaded properly.

Here we are quantizing only one bit that gives us low bit rate. Quantization error depends highly on an input image.

In order to reduce the dependency of quantization on input we introduce some uniformly distributed noise before quantization. Visually we can see that the false contouring has been reduced though there is some additive white noise. It gives an impression of having many gray levels with blurring effect when you look it from a distance.

Dithering gives us the detail rendition, but the drawback is the square grid pattern can be seen in the output image. Also, it reduces the great amount of noise as compared to above methods.

Visually we can see that

Using the root mean square error (RMSE) formula we have calculated RMSE for I2, I8 and I32 in code itself using:

$$\mathrm{RMSE} = \sqrt{\frac{1}{NM}\sum_{i,j}\{f(i,j) - b(i,j)\}^2}$$

We observed that RMSE is highest in case of I32 dithering followed by I8 then I2.

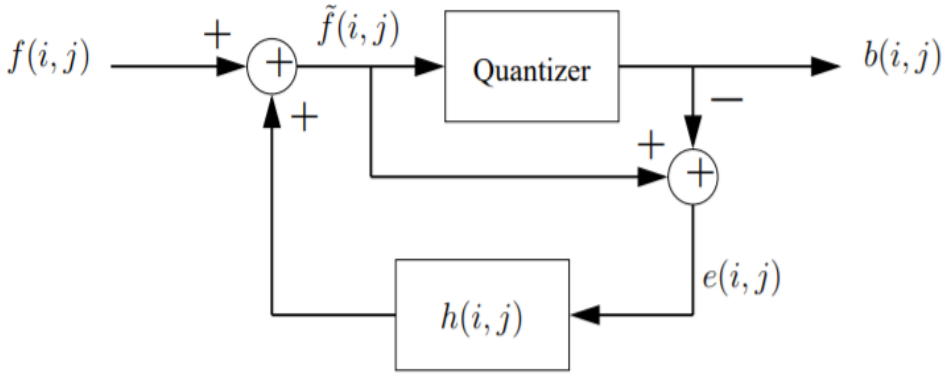
RMSE I32=117.502

RMSE I8=117.574

RMSE I2=117.187

b. Error diffusion:

- It's a nonlinear feedback system which quantize into higher frequencies.
- Noise is generates using correlation among neighborhood rather than generating independent random noise that means instead of point wise operation it quantizes each pixel using its neighborhood.
- If you do error diffusion row by row so the error will be all cumulative to right hand side, so the right-hand side does not look very natural. Therefore, the error diffusion will have a snake type scanning across the rows that is first from right to left then left to right so on. This type of scanning is also known as Serpentine scanning.
  We do serpentine scanning because along the diagonal it places more quantization error than raster.
  Hence by quantizing the result it pushes the error forward.



Above shown is the block diagram of error diffusion.

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

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

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

h(k,l) is an error diffusion matrix .

Algorithm:

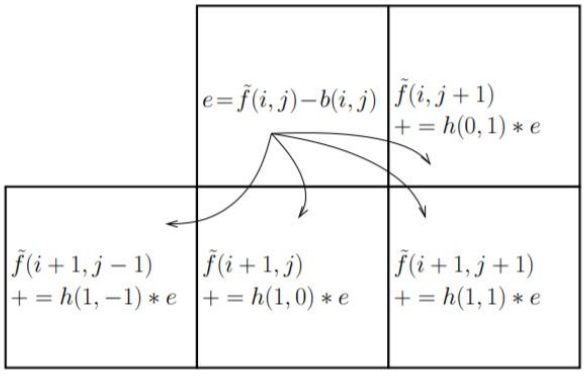
1. Initialize:

$$\tilde{f}(i,j) \leftarrow f(i,j)$$

2. Compute:

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

3. Diffusing the error forward:



4. Displaying the binary image b(i,j)

5. There are two commonly used error diffusion mask
   a. Floyd and Steinberg:

		7/16
3/16	5/16	1/16

   b. Jarvis et al:

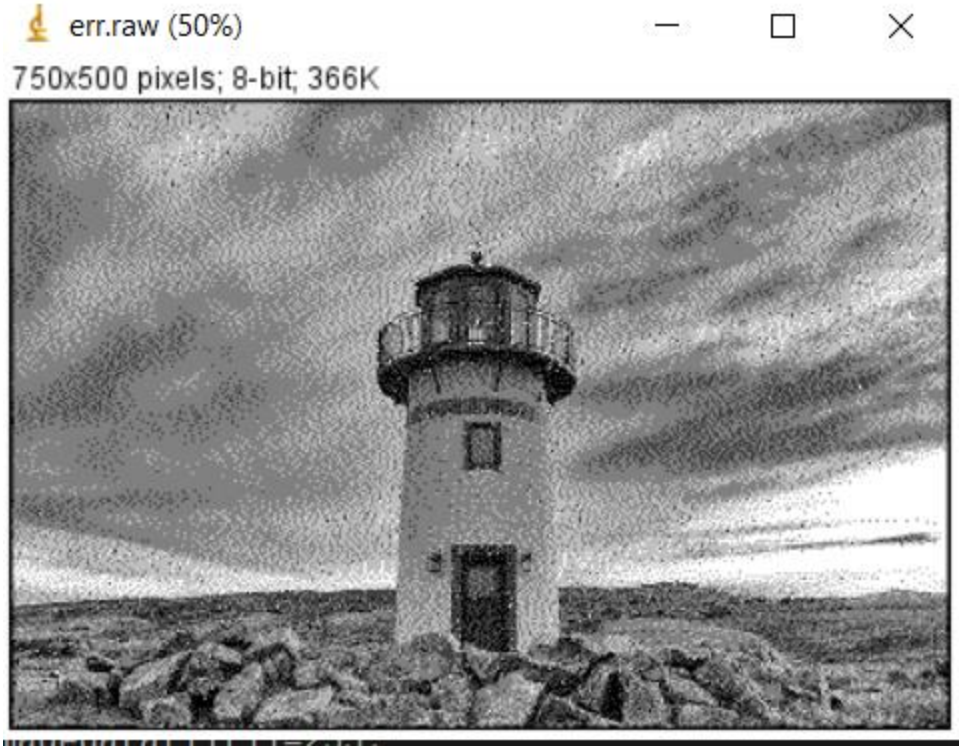
			7/48	5/48
3/48	5/48	7/48	5/48	3/48
1/48	3/48	5/48	3/48	1/48

   c. Stucki:

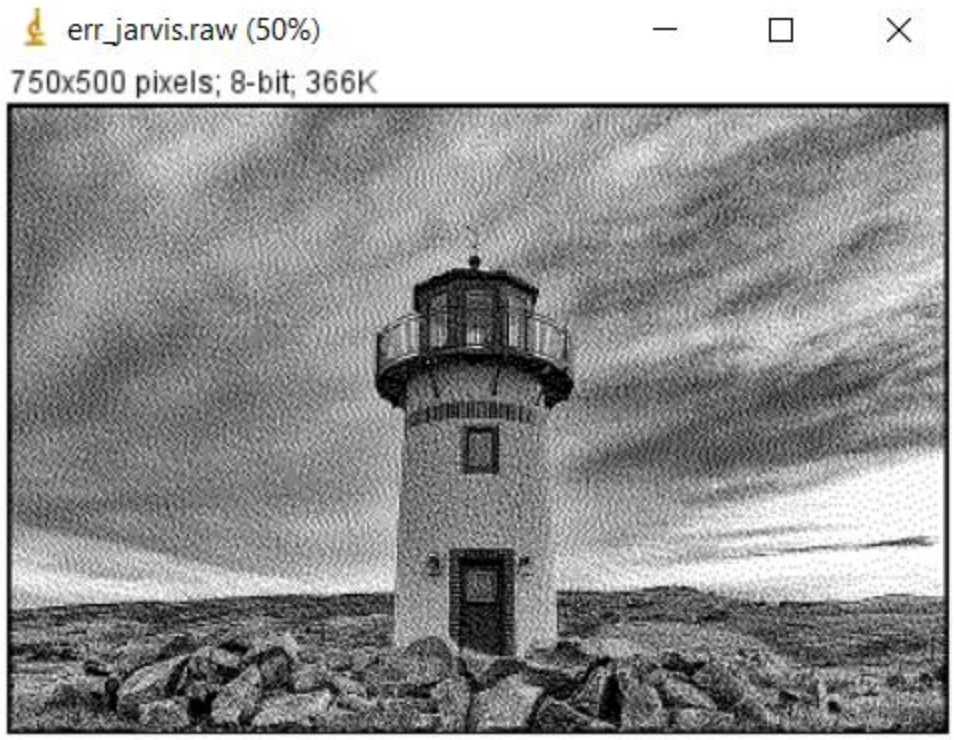
$$\frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

Results:

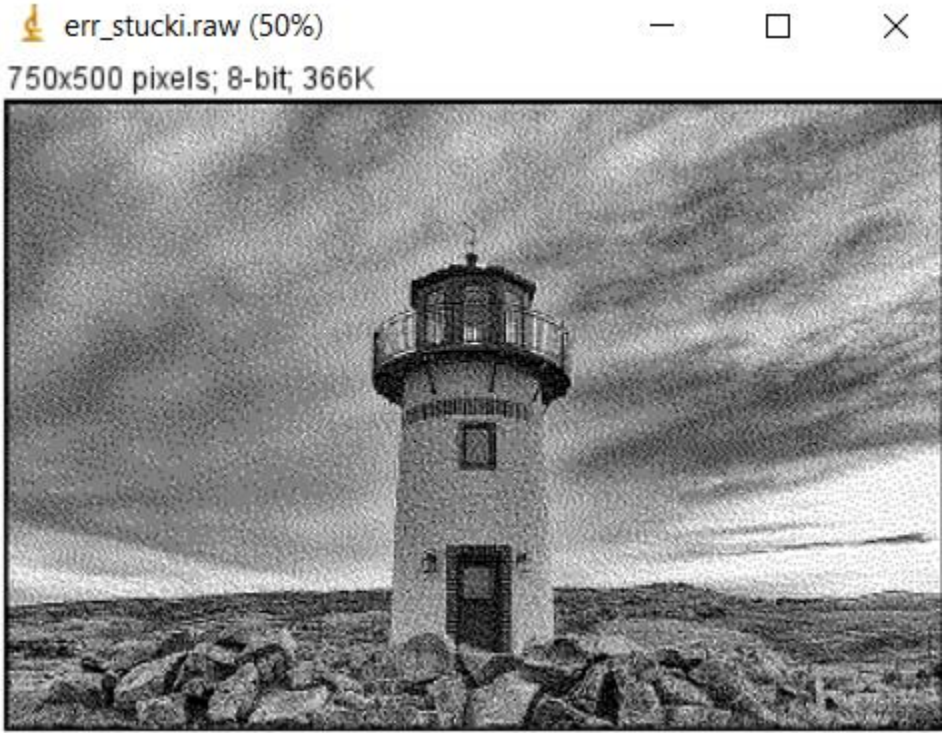
1. **Floyd-Steinberg:**



2.Jarvis:


err_jarvis.raw (50%)
750x500 pixels; 8-bit; 366K

3.Stucki:


err_stucki.raw (50%)
750x500 pixels; 8-bit; 366K

We can see that sharpening is directly proportional to correlation between error output image and input image.

- As we can observe the order of sharpness from above figures is:

  Stucki>Jarvis>Floyd

- PSNR: Peak Signal to noise ratio tells us how much noise is present in proportion to an image, the higher the better.
  Visually we can conclude that the order of PSNR is:

  Floyd>Jarvis>Stucki

In order to improve the error diffusion technique, we will use non causal error diffusion technique known as ***Multiscale error diffusion***. An image quadtree is used to represent difference between input gray image and output halftone image. The algorithm search for the brighter region (which requires more white dots) in an image in order to assign dots and then diffuses that quantization error non casually that is distribute all over the image via maximum intensity guidance. Therefore, it achieves the distribution of quantization more globally. It doesn't let the quantization error gets accumulated over range of resolutions. This method does not require any preprocessing as it preserves contrast of an original image and it doesn't over smooth the image. Diffusion filter is being used by which we can control amount of sharpness. Based on hierarchical intensity distribution we can measure quality of halftone images. This method provides very good result visually and intensity quality measure.

Color Halftoning:

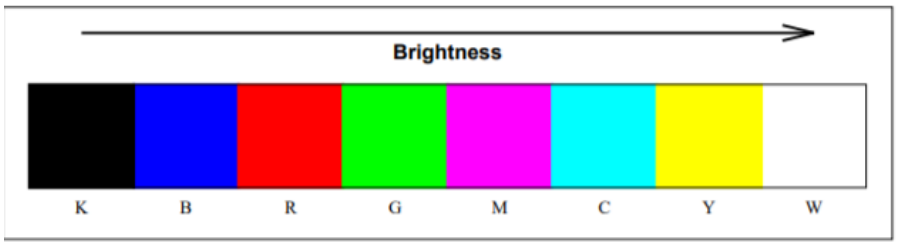
Motivation:

In order to reduce the ability of humans to notice the pattern because human eyes are more sensitive to change in brightness than to change in chrominance which performs averaging at lower frequencies.

Extension of error diffusion to color images by diffusing the error to all the three colored planes.

To reduce the visible artifacts such as the variation in the brightness of the dots.

Below shown is the order of brightness:



Minimal Brightness Variation Criterion (MBVC):

Goal:

Choosing uniform color spaces (visually) and therefore color difference is measured for color patches. In MBVC we use saturated dots such as Red, Blue, Green, Cyan, Magenta, Yellow instead of Black or White because their variation in brightness is minimum.

Separable Error Diffusion:

Separate the image into CMY channels and then apply Floyd=Steinberg error diffusion to each of the 3 channels separately.

To render any solid color patch, we use all the 8 colors. As a result, Black and white will be used to render any solid color patch because its variation in brightness is maximum.

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

2.1) Minimal Brightness Variation Quadruple (MBVQ):

This method can overcome problems in separable error diffusion as it uses ink relocation transformation of neighboring halftone couples in order to minimize variation in brightness and at the same time it preserves their average color.

Following process is done until no black or white halftones remains:

I. Transformation of white and black halftones to green and magenta.
KW ->MG

II. Transformation of Black and primary color halftones to secondary colors as follows:

KY->RG, KC->BG, KM->BR

Now our MBVQ will become RGBK, if black halftone is still there.

III. Transformation of Black and primary color halftones to secondary colors as follows:

WB ->CM, WR->YM, WG->YC

Now our MBVQ will become RGBK, if white halftone is still there.

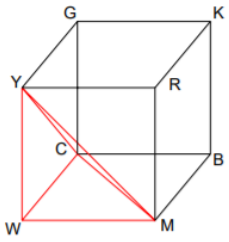
IV. If neither black nor white is there then we are left with 6 half tone colors, so now the following transformations will be made to eliminate last two half tone colors:
BY->MG; RC->MG

Now we have 4 possible MBVQ:

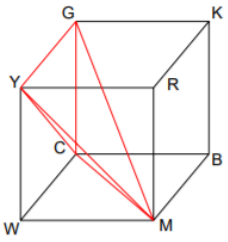
MYGC, RGMY, RGBM, CMGB

Every input color can be rendered using 6 half tone quadruples: RGBK, WCMY, MYGC, RGMY, RGBM, CMGB.

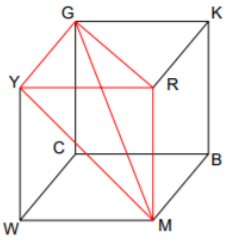
Given RGB triplet we compute MBVQ by location of a point:



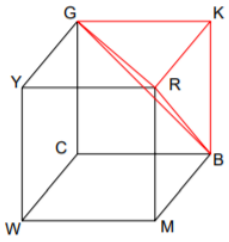
CMYW



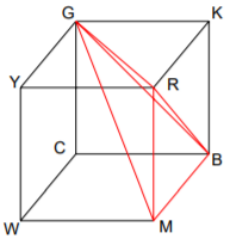
MYGC



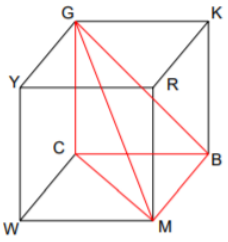
RGMY



KRGB



RGBM



CMGB

Color Diffusion Algorithm:

Algorithm:

1. For each pixel in image do:
   Determine MBVQ (RGB(i,j)) using above shown algorithm.
2. Find the vertex v ∈ MBVQ which is closest to RGB(i,j)+ e(i,j). here e is the error at pixel (i,j).
3. Compute the quantization error RGB(i,j) + e(i,j) -v.
4. Distribute the error to future pixels using Floyd-Steinberg error diffusion.

RESULTS:

1. Separable error diffusion:

2. MBVQ based error diffusion:

Conclusion:

In MBVQ artifacts are minimized. Variation of green pixels value will decrease in MBVQ and noise is also reduced in MBVQ as compared to separable error diffusion. And in MBVQ variation in brightness will be less and in separable

error diffusion, to render any solid color patch, we use all the 8 colors. As a result, Black and white will be used to render any solid color patch because its variation in brightness is maximum.

References:

1. J. Canny, “A computational approach to edge detection,” IEEE Transactions on pattern analysis and machine intelligence, no. 6, pp. 679–698, 1986
2. http://mcl.usc.edu/wp-content/uploads/2014/01/1997-03-A-multiscale-error-diffusion-technique-for-digital-Halftoning.pdf
3. http://www.adeveloperdiary.com/data-science/computer-vision/how-to-implement-sobel- edge-detection-using-python-from-scratch/
4. D. Shaked, N. Arad, A. Fitzhugh, I. Sobel, "Color Diffusion: Error-Diffusion for Color Halftones", HP Labs Technical Report, HPL-96-128R1, 1996.
5. https://engineering.purdue.edu/~bouman/ece637/notes/pdf/Halftoning.pdf
6. https://engineering.purdue.edu/~bouman/grad-labs/Image-Halftoning/pdf/lab.pdf