HOMEWORK #2

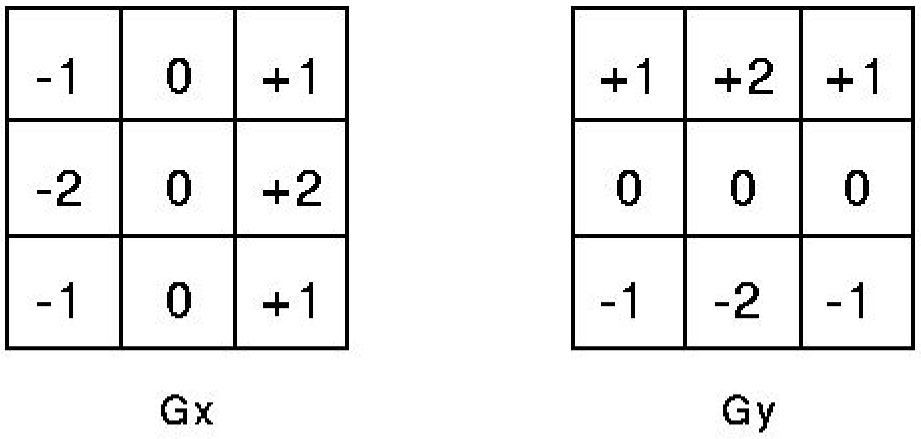
Issued: 01/29/2020 **Due: 02/16/2020**

Problem 1: Edge Detection

I. Abstract and Motivation

Edge Detection: Edges are the important features for analyzing and processing the image. Edge detection serves as a building block for high-level computer vision tasks like Object detection, localisation and recognition. Furthermore, edge detection serves as an important feature extraction step for scene analysis. Edge detection is basically finding the sudden change in pixel values in an image. We implement and evaluate most popular edge detection algorithms in this section.

1) **Sobel Edge Detector:** One of the most common and widely used edge detection techniques is Sobel edge detection. Sobel operator is the approximation of first order derivative of an image. We calculate the gradient (derivative) of an image in X and Y direction and find the gradient magnitude and orientation. We set a threshold for gradient magnitude and obtain binary edge maps based on the threshold. The 3*3 kernel to calculate gradient in X and Y direction is shown in Figure 1. (Image are taken from [4]).



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

Gx

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

Gy

Figure 1. 3*3 Kernel for calculating gradient in x and y direction

The entire operation is summarized in Figure 2.

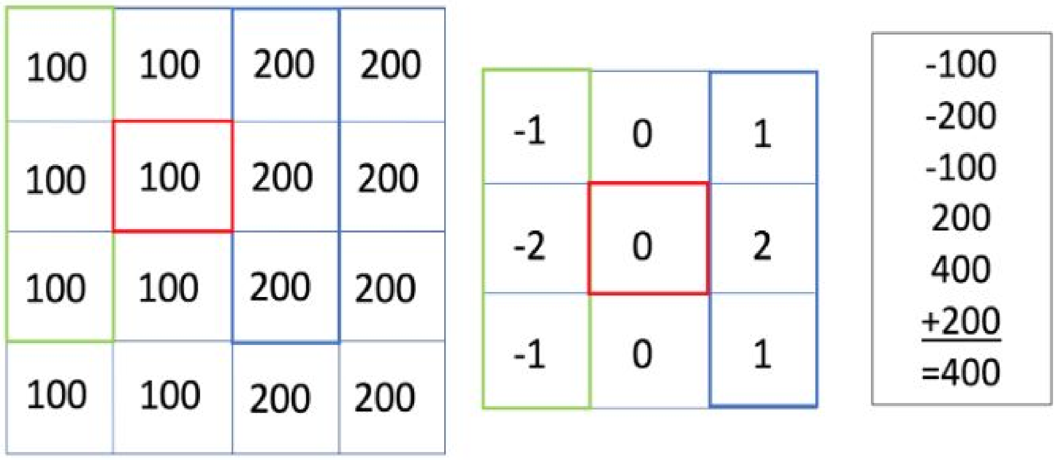


Figure 2. Kernel Convolution

Gradient magnitude at the center pixel is calculated using the below equation.

$$G' = \sqrt{G_x^{\,2} + G_y^{\,2}}$$

$$G_\theta = atan(\frac{Gy}{Gx})$$

We then normalize the gradient magnitude using min-max normalization. To generate edge maps, we take CDF of the gradient magnitude and set a threshold (usually 90% of the maximum gradient).

2) Canny Edge Detector: Canny edge detector is an improvised multi-stage algorithm to enhance the edges obtained from Sobel edge detector. The algorithm was proposed by John Canny in 1986 [1]. This method takes a gray scale or binary image as input and outputs a binary image with bright values in edge pixels.

The flowchart of the canny edge detector is shown in Figure 3.

Basically canny edge detectors are improvisation on Sobel edge detectors. Initially we convolve the original image with derivative of the Gaussian i.e., Gaussian smoothing followed by 1st order derivative. Gaussian smoothing is mainly to remove noise and remove the sensitivity of the edge detector to noise. We calculate gradient in both X and Y direction and calculate the gradient magnitude and gradient orientation. After this, we utilize post-processing to refine edge maps. This involves two steps namely, **Non-Maximum suppression** and **Hysteresis thresholding**

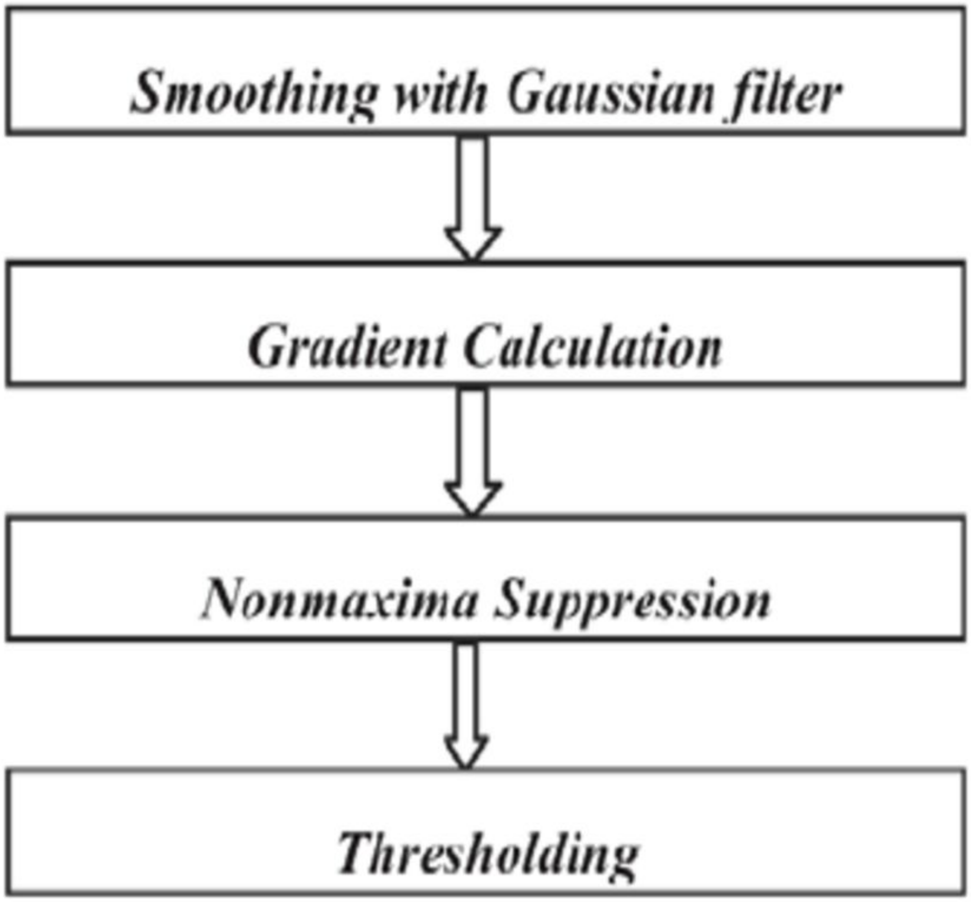


Figure 3. Flowchart of Canny Edge Detector.

Non-Maximum Suppression: After calculating the gradient magnitude, we often tend to have thick edges which are visually unpleasant and difficult to comprehend while using for mid-level operations like object detection and localisation. Thick edges don't really add any value because all we care about is whether there is an edge or not. So, non maximum suppression is a process of thinning the multiple-pixel wide ridges (Edge-band) to single pixel-wide ridge (Edge-line).

Non-maximum suppression considers the maximum valued pixel in the edge. If that pixel value is greater than the pixels in the gradient direction, then we keep that pixel else we set the pixel value to zero. Figure 4. Shows the pictorial representation of Non maximum suppression [2].

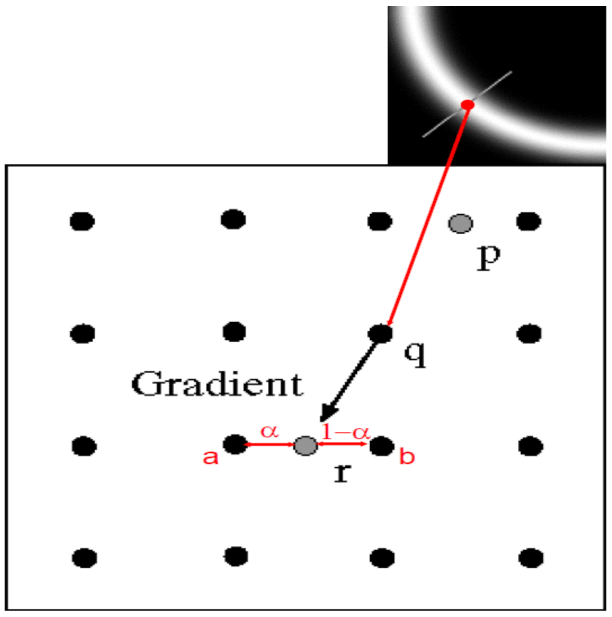


Figure 4. Non Maximum Suppression

In the above figure, consider the pixel q which has maximum value in the edge. Let p and r be the pixels in the gradient direction of q. If both p and r pixels are lesser than q, then we preserve q else we set it to 0. This way reduces the edge band from the edge line.

Dog Image (0.0563 0.1406) Gallery Image (0.0375 0.0938)

Hysteresis Thresholding or Double thresholding: The noise in the images give us hard time in edge detection because noisy pixel values are also detected as edges. Also, textures in the images are actually not edges, but instead they are detected as edge because of the strong gradient magnitude value. To overcome this, we use hysteresis thresholding or double thresholding. In this method we choose two thresholds - **low threshold and high threshold.**

We normalize the pixel values in between 0-1 after non-maximum suppression. Any pixel value,

1. Above high threshold is considered as a strong edge.
2. Below the low threshold is a weak edge.
3. Between two threshold values, we will have to explicitly determine whether it is an edge or not.

After detecting strong and weak edges, we determine whether the pixels in between the threshold values are real edges or not. For that we use edge tracking algorithm. In this, the edges which are connected to strong edges are considered as edge pixels and which are not connected are removed.

3) Structured Edge: Until mid 2010, traditional approaches for edge detection were prevalent. Canny edge detectors are still favorite for many edge detection tasks. After knowing the significance of machine learning and pattern classification, researchers explored faster and efficient ways for edge detection, which they call as - **Data driven approach.** Structured edge is an improvisation on **Sketch token** algorithm. Before diving into the intricacies of Structured edge, let us skim the Sketch token algorithm.

- Sketch tokens collect hand drawn contours that represent a wide variety of edge structures (35*35 Image patch).
- K-means clustering algorithm to categorize the tokens into 150 classes.
- Collect features from original images like channel features and self similarity.
- Train the random forest classifier such that it should classify any input patch to one of the token classes.
- So, the **output space dimensionality** in Sketch tokens in **151** (150 classes for edges and 1 class for no edge).

In structured edge approach, instead of using predefined labels we directly use edge structures (16*16 structure patch). The **output space**

dimensionality is 2^{256}**,** i.e, each pixel will have two decisions- whether it's an edge pixel or not, similar to semantic image labelling.

Our main goal is to train the random forest decision trees in a recursive way so that we increase the information gain by finding the optimum parameter for split function $h(x, \theta_j)$ that gives the best split of the given image patch.

Major hindrance in the training decision tree is, the output space dimensionality is very high which makes it hard to calculate Gini impurity at each node i.e., to train the decision trees. To overcome this, we Z encode the information and check whether every pair in the output belong to same segment or not which brings down the dimension from 2^{256} **to 32640**. We further reduce the output dimension to 5 using **Principal Component Analysis (PCA) and K-means clustering**. To predict the contour structure, we take the mean of the responses from each decision tree to obtain an edge map. Thex flowchart shown in Figure 5 summarizes the above steps.

Figure 5 Shows the flowchart of the structured edge algorithm.

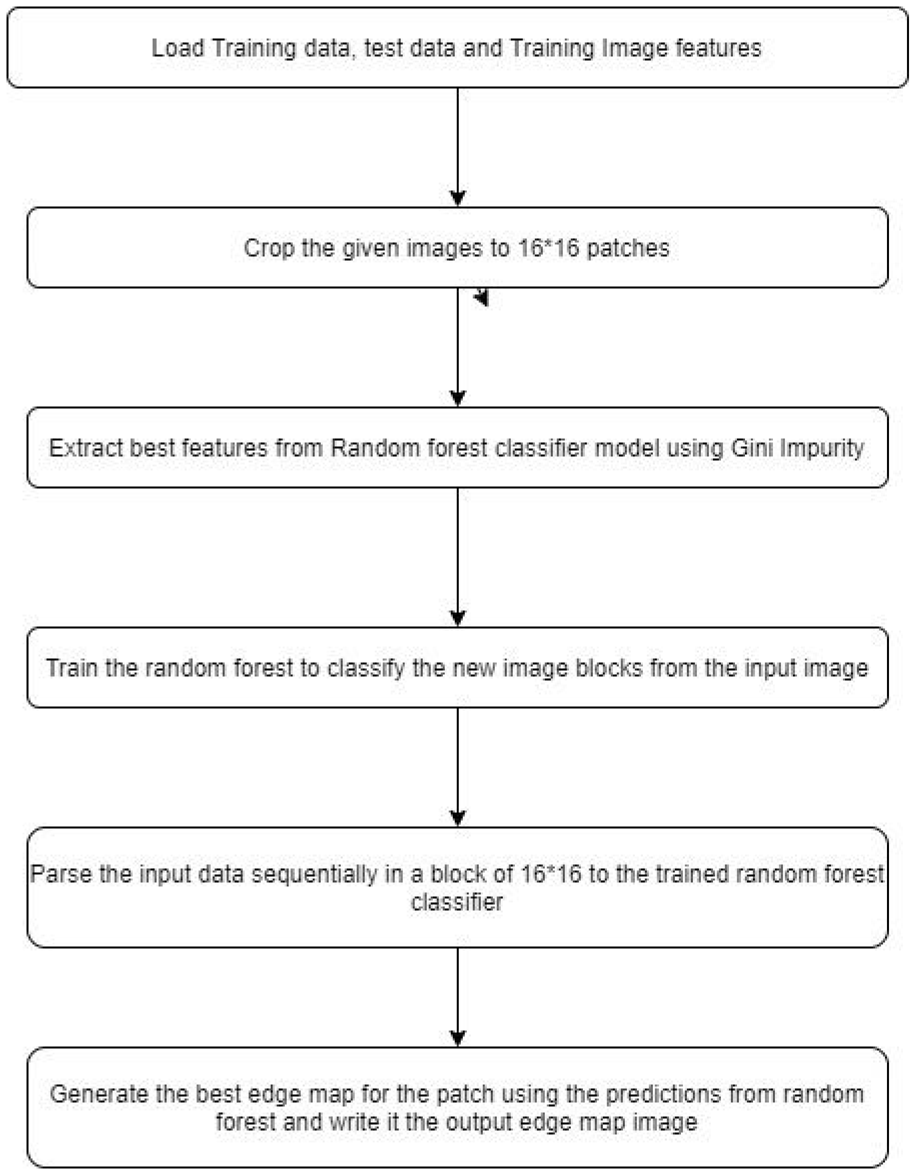

Load Training data, test data and Training Image features
Crop the given images to 16*16 patches
Extract best features from Random forest classifier model using Gini Impurity
Train the random forest to classify the new image blocks from the input image
Parse the input data sequentially in a block of 16*16 to the trained random forest classifier
Generate the best edge map for the patch using the predictions from random forest and write it the output edge map image

Figure 5: Flow Diagram of Structured edge algorithm

Random Forests: Random forest classifier is a supervised machine learning algorithm which uses decision trees to accurately classify given data. Decision trees alone fail to produce accurate results due to the homogeneity of the training data. As a result, they perform well only on training data but fail to perform on test/validation data. In random forests, we use the power of statistics to overcome this inaccuracy. To understand the Random forest algorithm, we need to comprehend the idea of decision trees.

Decision trees are tree-like models used to classify a given set of data. For training a decision tree, we need a labelled training set. While training a decision tree, the first question comes to our mind is - what feature should be placed at which node, so that the node splits the data accurately. To tackle this, we use **Gini Impurity.** Gini impurity is a probabilistic measure to find which feature produces maximum information gain i.e., by using which features we can classify the data accurately.

$$Gini\ Impurity\ =\ 1-(Probability\ of\ yes)^2-(Probability\ of\ No)^2$$

We calculate Gini impurity for all the features and the feature with lowest Gini impurity (or highest information gain if we consider entropy) will be used in the particular node.

The straightforward approach of building a decision tree and using it on new data won't produce accurate results. Individual decision trees have high inaccuracy because of overfitting problems. So, we use Random forests classifier to tackle this. Random forests are a bunch of decision trees aggregated together. Below is the step by step approach.

1. **Create a bootstrap dataset:** Pick samples randomly from the given labelled dataset. Here we are allowed to pick a sample more than once.
2. **Create a decision tree using bootstrap dataset**: Pick only a subset of features from the bootstrap data set and build a decision tree.
3. **Repeat step 1 and 2 multiple times.**
4. The variety is what makes random forests more effective than individual decision trees.
5. **Using Random forest:** Use the first decision tree from the random forest and keep the count of decisions.
6. Repeat the same process using all the decision trees in the random forest.
7. Pick the decision with the highest count.

   Note: Bootstrapping the data plus using the aggregate to make decision is called "**Bagging**"

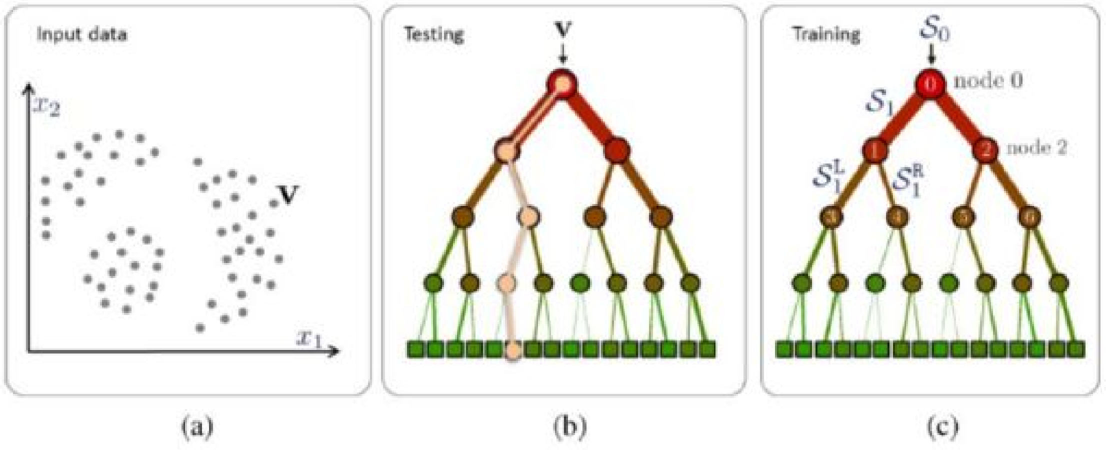


Figure 6: Random Forest - Decision tree

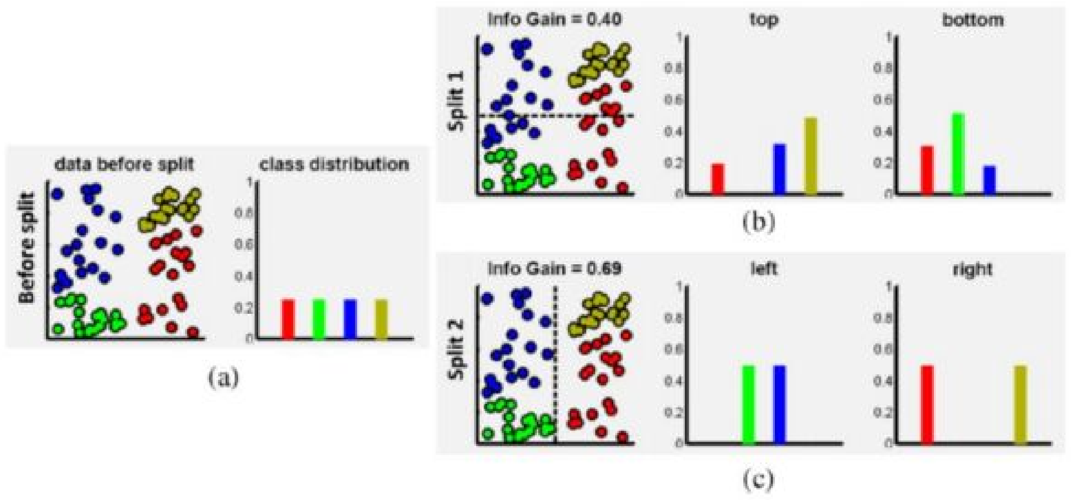


Figure 7: Gini Impurity calculation and segregation

4) Performance Evaluation: In order to evaluate the performance of different edge detection algorithms, we use a quantitative measure to evaluate different algorithms. Edges are a very important feature in object detection. Nevertheless, different people will have different opinions about the edge boundaries. So, we create ground truth edge maps for each image from a handful of people. To handle this diversified opinion about edges we take mean performance measure with respect to each ground truth data. So every pixel of the edge (probability) map will belong to either of the following groups:

1. **True Positive:** Pixels of the edge map intersects with the ground truth (successfully identified).
2. **True Negative:** Non edge pixel of the edge map correctly interacts with the non edge pixel of the ground truth.
3. **False Positive:** This is also called False Alarm. In this the edge detection algorithm inaccurately detected a non edge pixel.
4. **False Negative:** This corresponds to the missing edge pixel in the edge map compared to ground truth image.

Figure * depicts the above points concisely

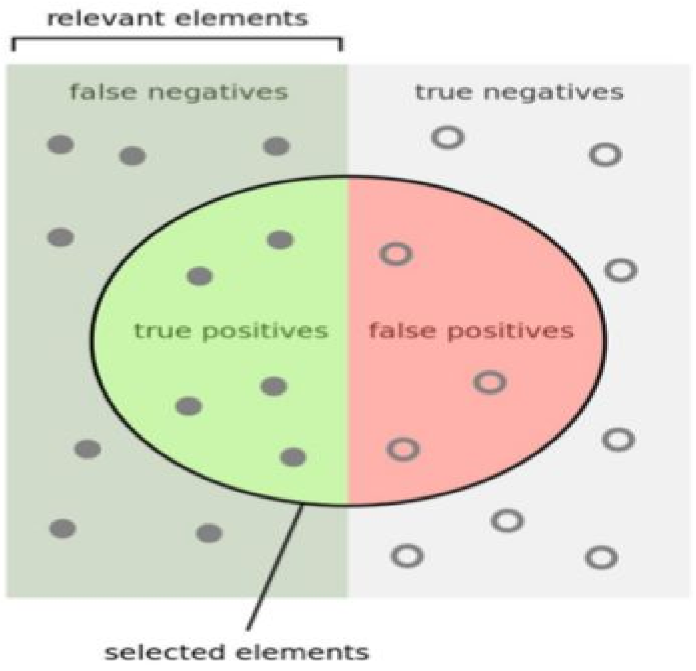


Figure * Representation of Performance measures

The performance of an edge detection algorithm is quantified based on Precision and Recall and F Measure. The formulas are given below

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

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

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

II. Approach and Implementation

1) **Sobel Edge Detector:**
   a) Input image is converted to gray scale image.
   b) Convoluted the image with X Filter and Y Filter.
   c) Calculated Gradient Magnitude (without normalization).
   d) For Visualization purpose Min-Max normalization is used for both X-Y Gradient and Gradient Magnitude.
   e) Cumulative distribution is calculated on normalized gradient magnitude.
   f) Tried with threshold values and obtained the better edge map.
2) **Canny Edge Detector:**
   a) Used matlab inbuilt function to implement canny edge detection (https://www.mathworks.com/help/images/edge-detection.html)
   b) Obtained the image with default threshold.
   c) Change the threshold values and see for which threshold value we get better output.
3) **Structured Edge:**
   a) The instructions given in the readme.txt file are carefully followed and compiled the source code.
   b) Downloaded the necessary plugins and toolboxes.
   c) EdgeDemo.m function is used to obtain edge maps for both Gallery and Dogs Image with different parameters.

4) **Performance Evaluation:**
   a) edgeEvalImg.m function is executed to obtain the precision,recall for different values of threshold.
   b) Calculated F-measure of each threshold and plotted it.

c) Also, F-measure for each ground truth image is calculated to obtain mean F-measure.

III. Results and Discussions:

1) **Sobel Edge Detector:** All the edge detectors are implemented on two images shown in Figure 8 and Figure 9.

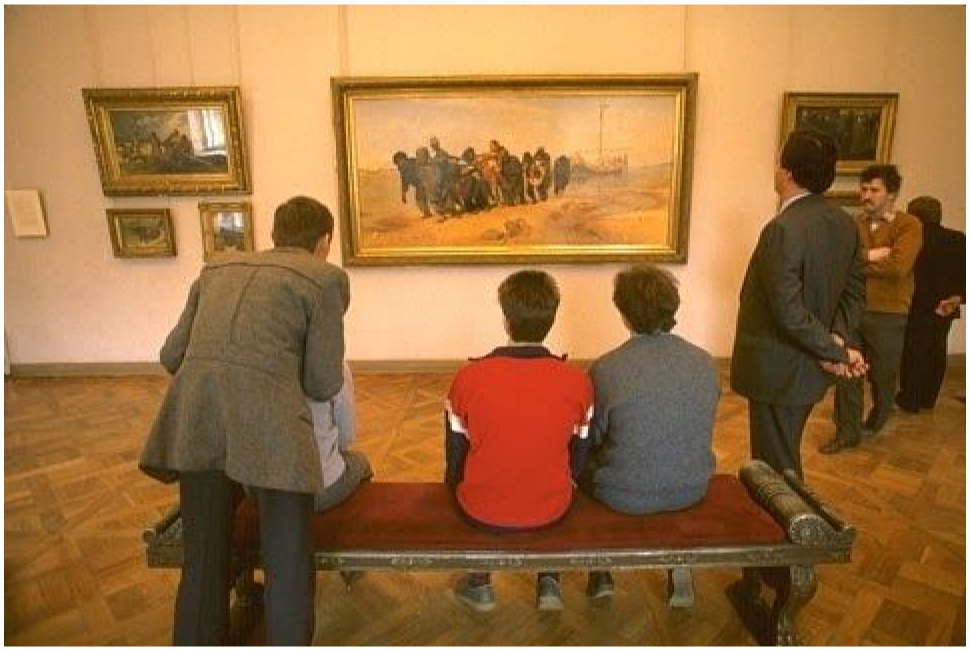


Figure 8: Gallery Image (Input Image)

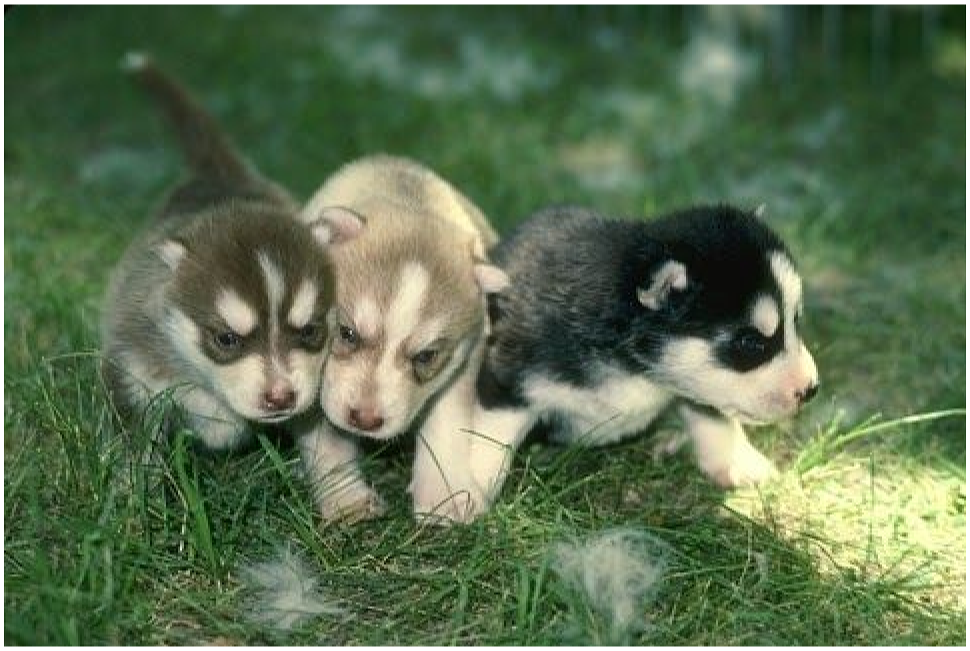


Figure 9: Dogs Image (Input Image)

The result of the Sobel edge detector is shown in the Figures below. Figure 10-17 shows the result on Gallery Image.

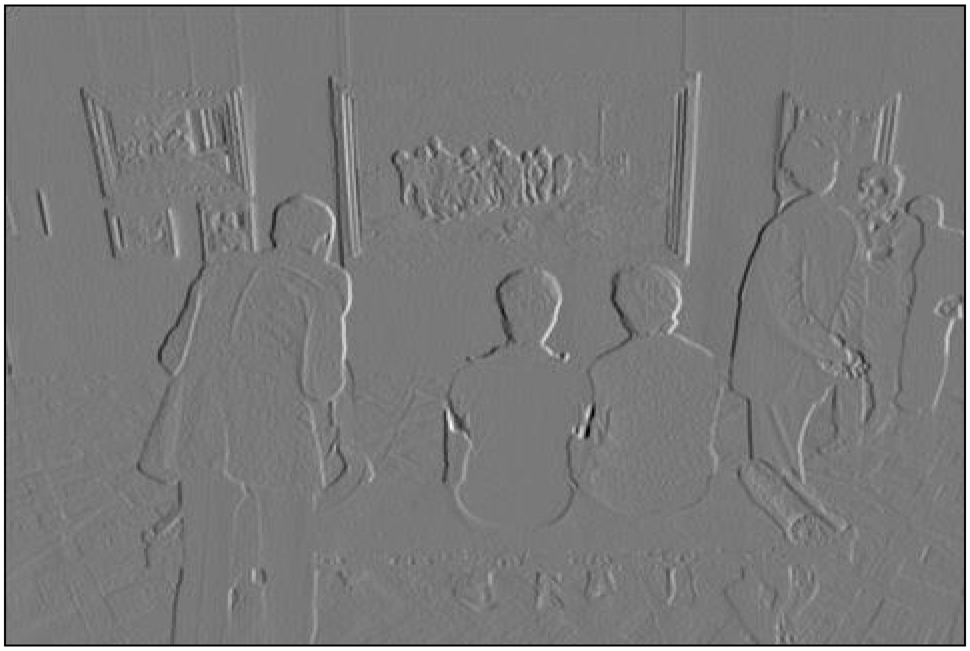


Figure 10: X Gradient of Gallery Image

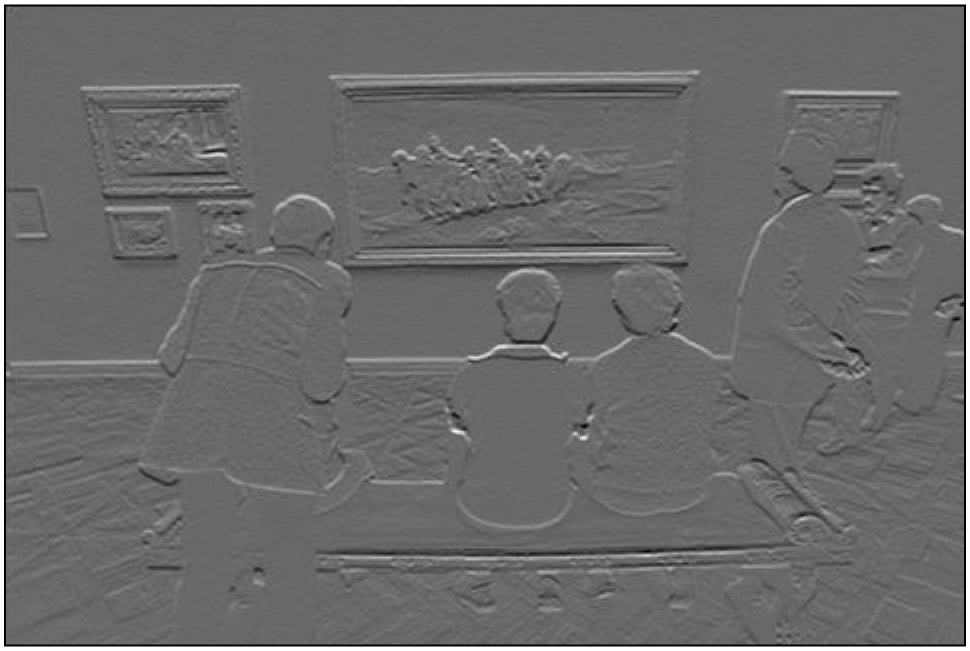


Figure 11: Y Gradient of Gallery Image

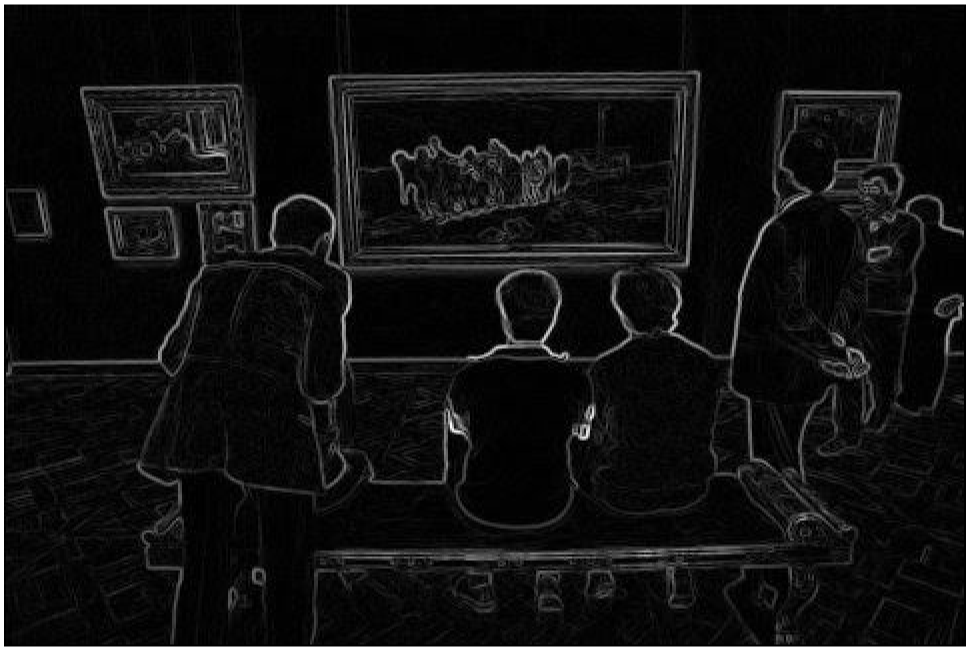


Figure 12: Gradient Magnitude of Gallery Image

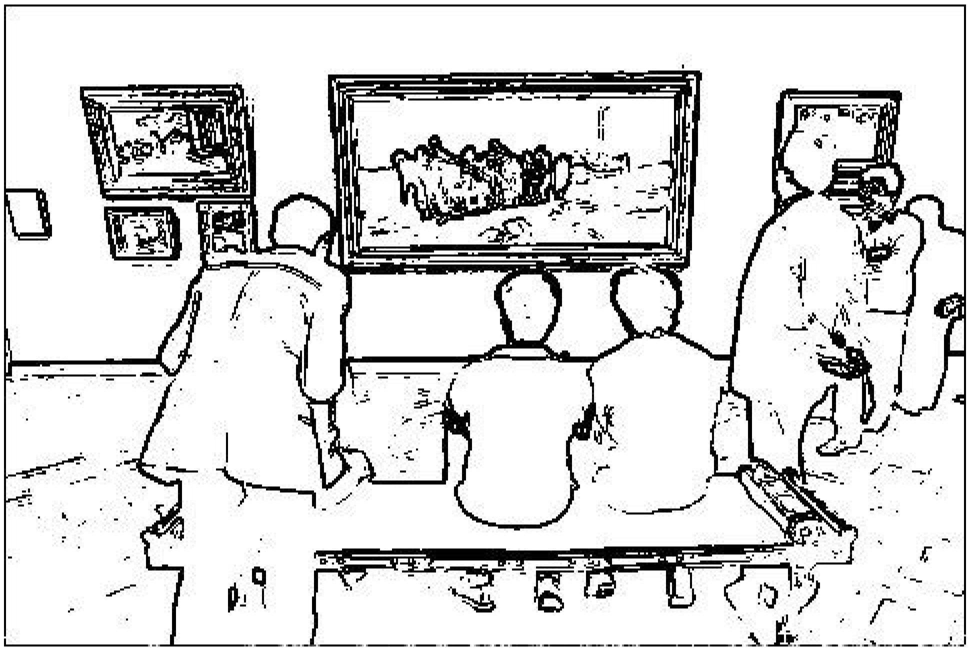


Figure 13: Edge Map with Threshold 85% of Gallery Image

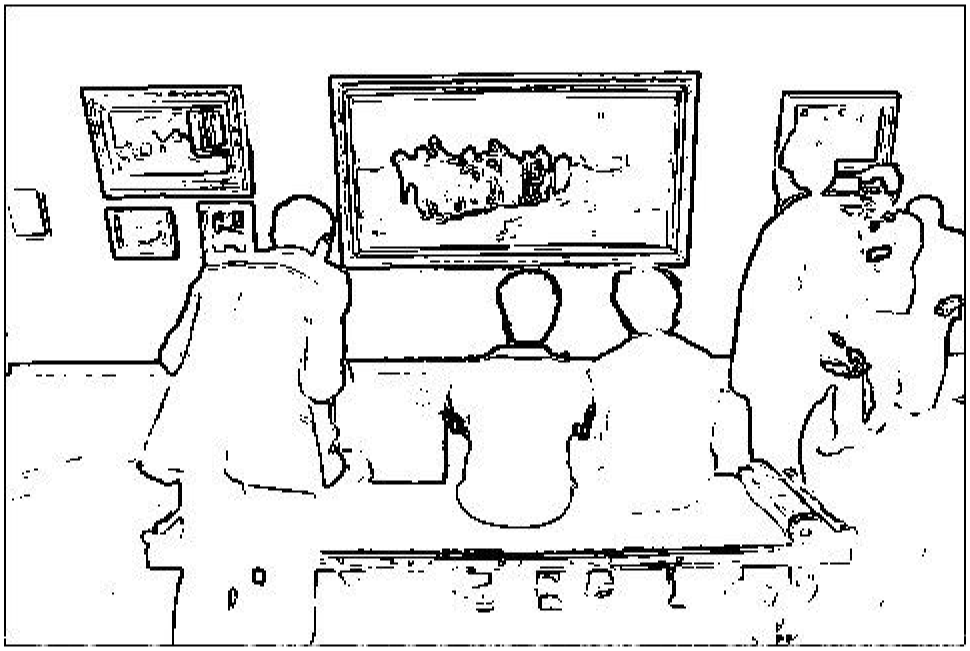


Figure 14: Edge Map with Threshold 90% of Gallery Image

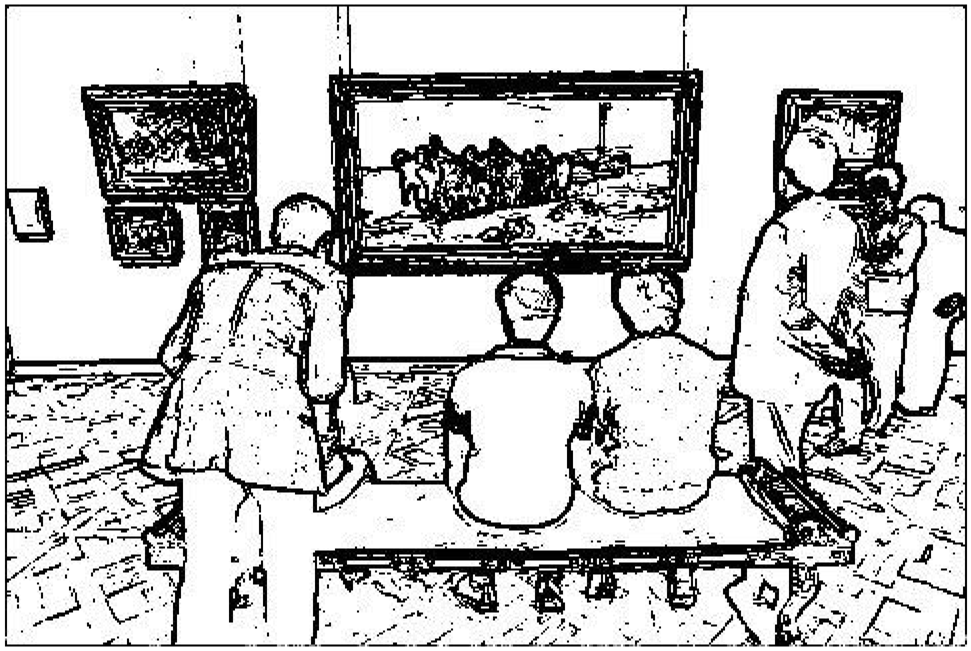


Figure 15: Edge Map with Threshold 70% of Gallery Image

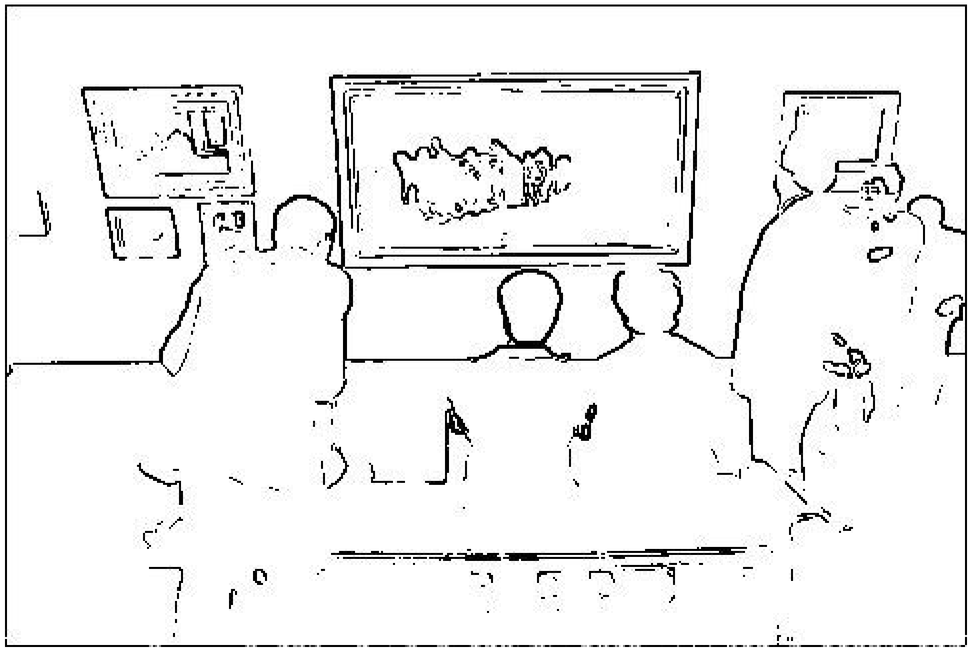


Figure 16: Edge Map with Threshold 95% of Gallery Image

Binary edge maps for different thresholds are tried. We can see that with 70% threshold value there will be thick edges and textures are highlighted. Also in 95% threshold value, actual edge boundaries are eliminated. So, the pragmatic choice of threshold should be in between 85% to 90%. I have chosen 90% threshold because it resulted in the best output.

Figure 17 shows the cumulative distribution of the Gradient Magnitude of Gallery image.

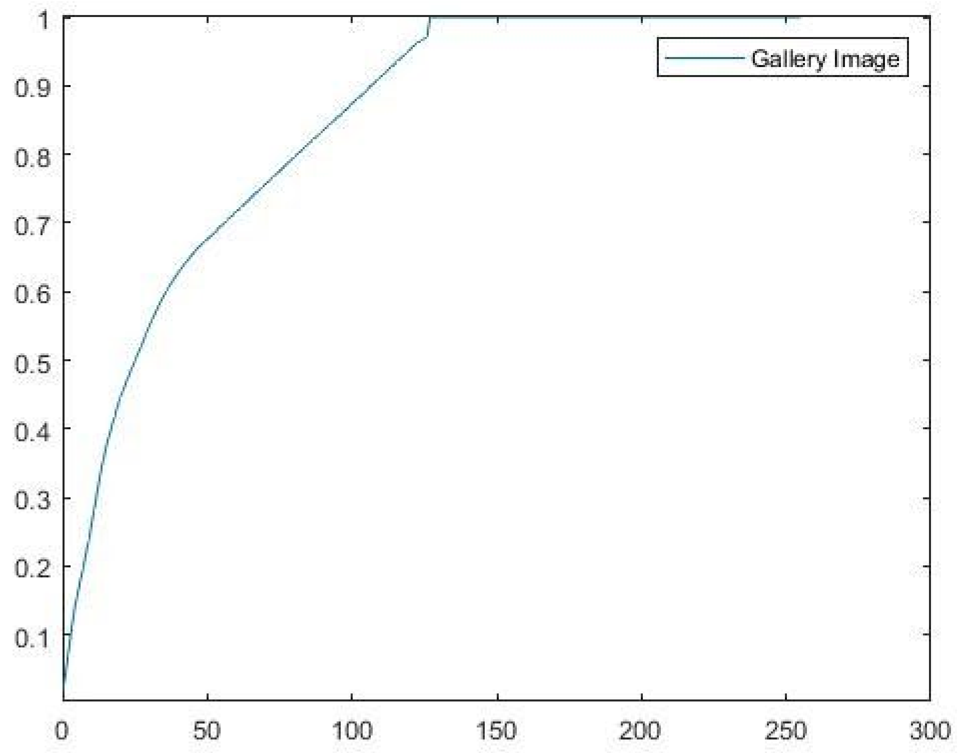


Figure 17: Cumulative Distribution of Gallery Image (Threshold of 90% is chosen)

Figure 18-25 shows the result on Dogs Image.

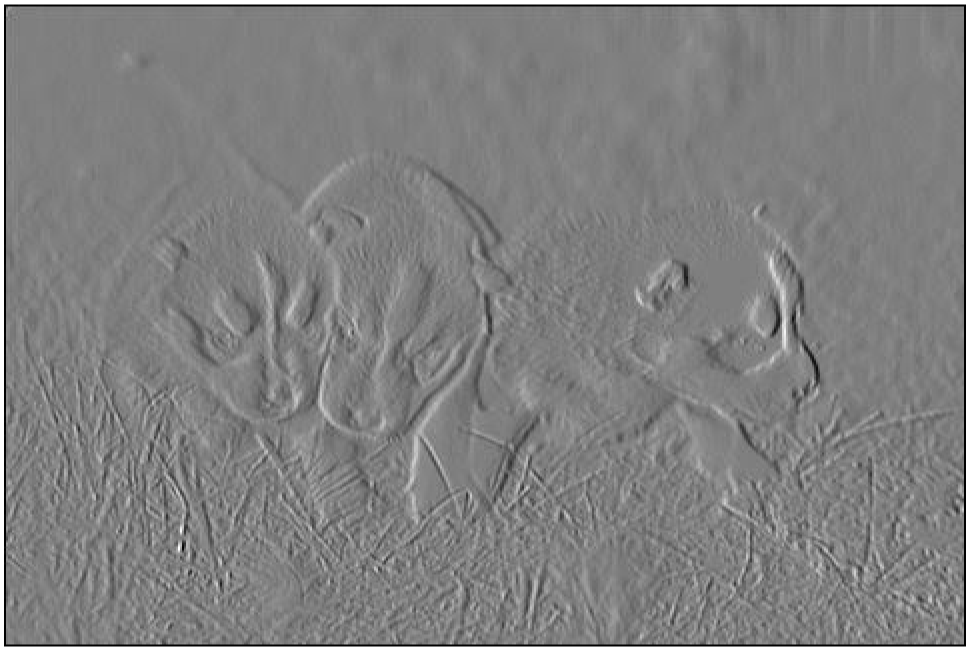


Figure 18: X Gradient of Dogs Image

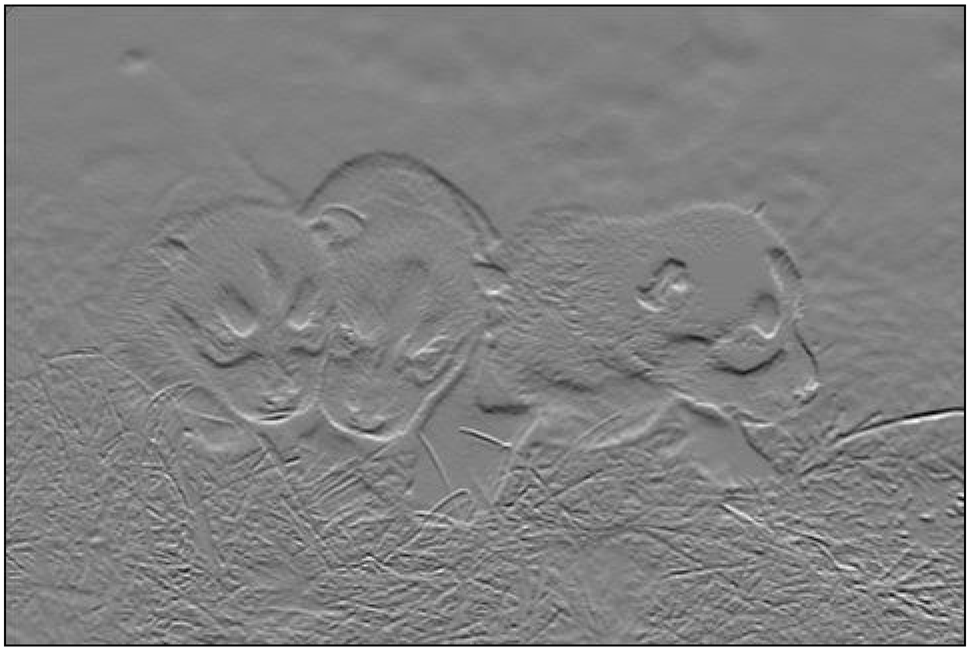


Figure 19: Y Gradient of Dogs Image

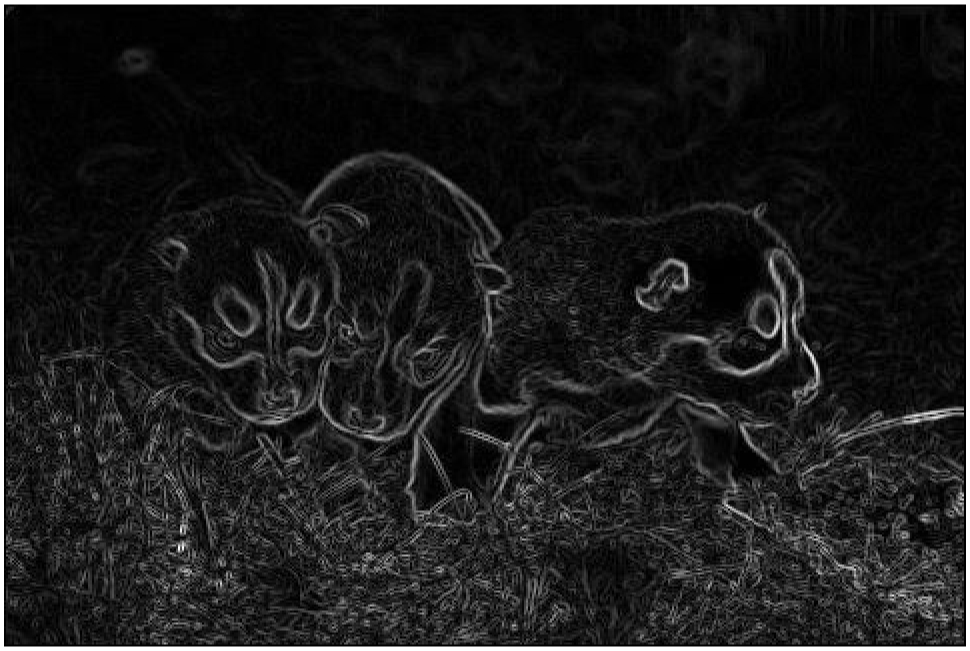


Figure 20: Gradient Magnitude of Dogs Image

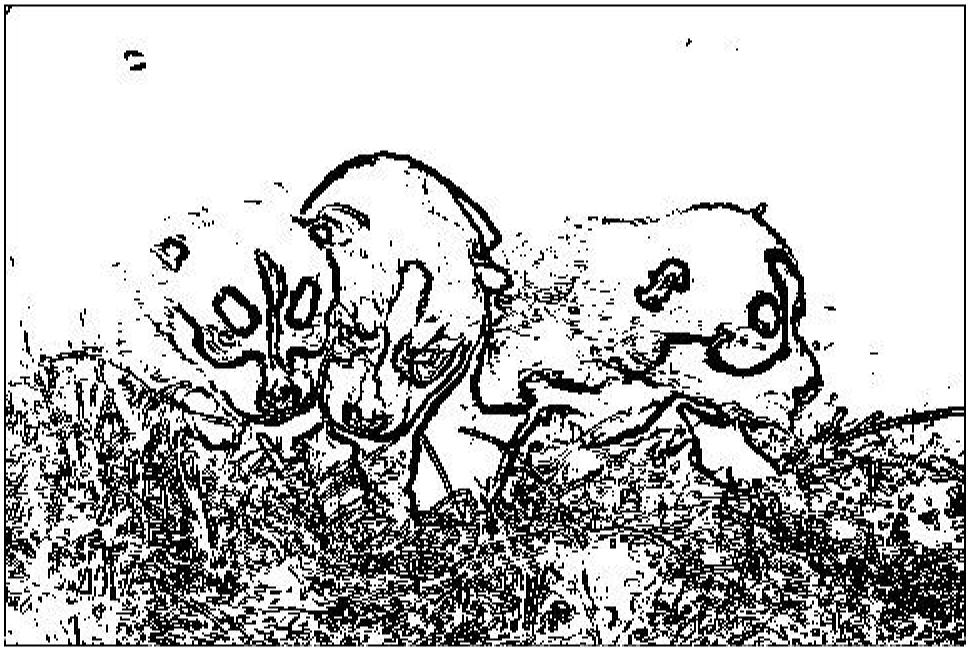


Figure 21: Edge map with Threshold 75% of Dogs Image

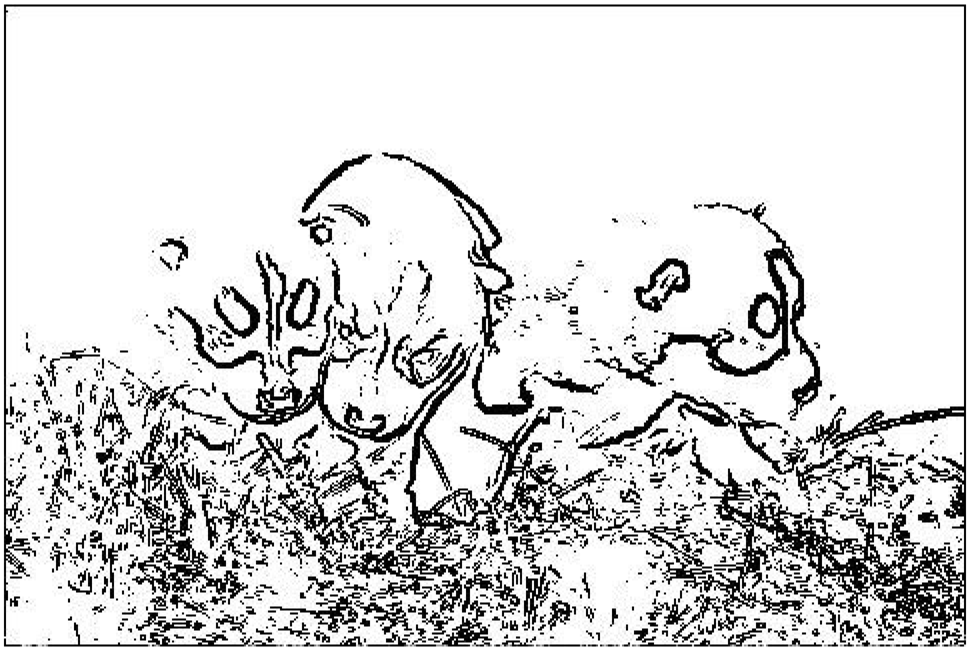


Figure 22: Edge map with Threshold 85% of Dogs Image

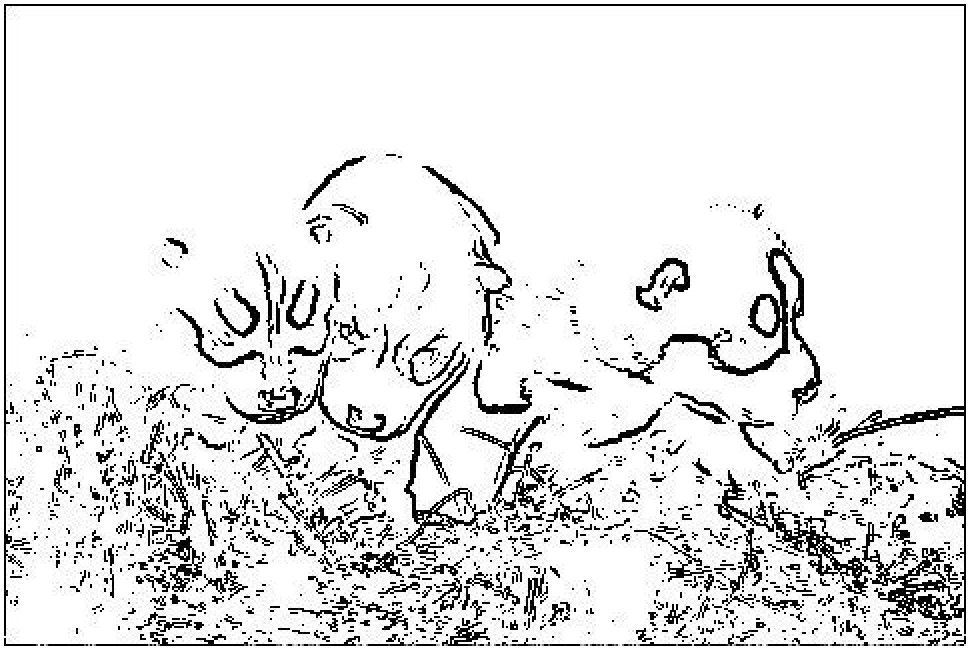


Figure 23: Edge map with Threshold 90% of Dogs Image

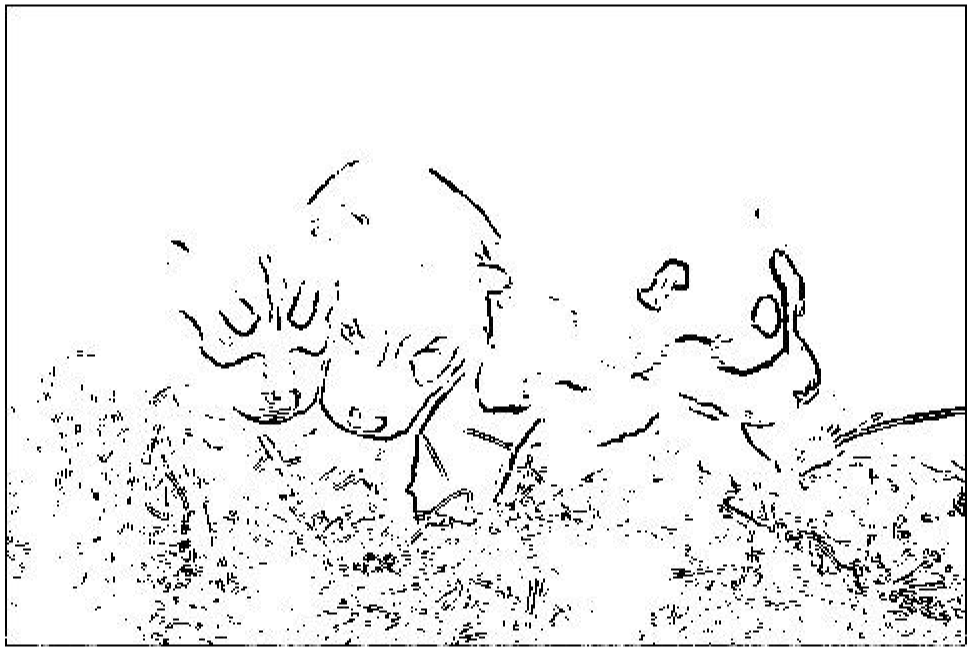


Figure 24: Edge map with Threshold 95% of Dogs Image

Binary edge maps for different thresholds were tried. We can see that with 70% threshold value there will be thick edges and textures in the background like grass are highlighted. With 95% threshold value while the background texture is being eliminated we can see that the actual edge boundaries are also eliminated. So, the pragmatic choice of threshold should be in between 85% to 90%. I have chosen 90% threshold because it resulted in the best output.

Figure 25 shows the cumulative distribution of the Gradient Magnitude of Dogs image.

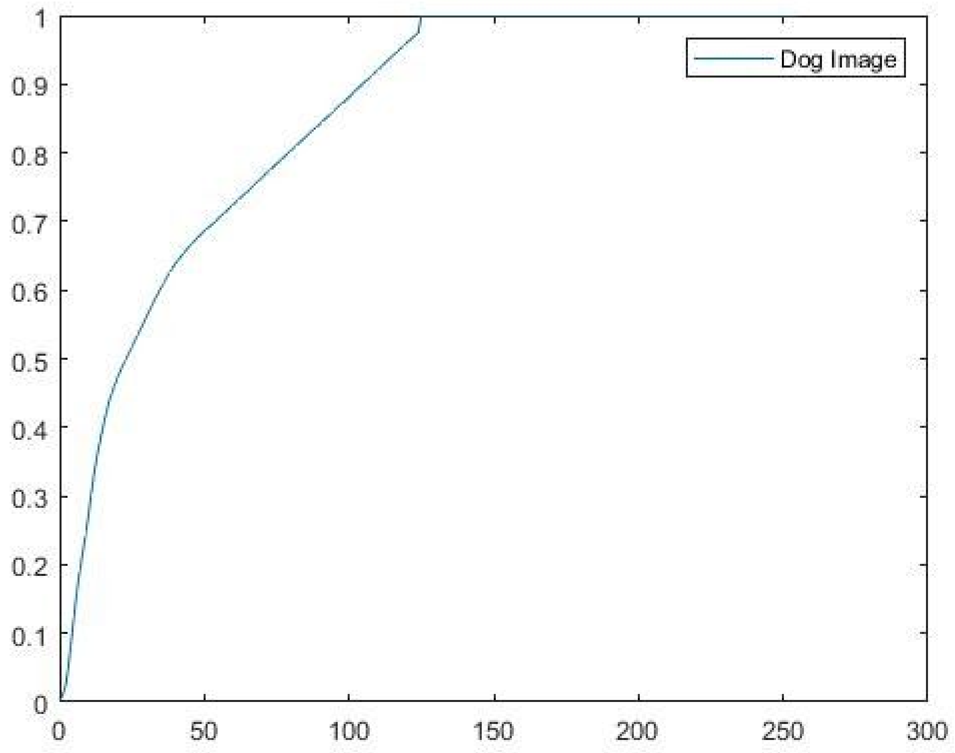


Figure 25: Cumulative Distribution of Dogs Image (Threshold of 90% is chosen)

Observations:

A. We got a better edge map for Gallery image using Sobel Edge detector. However, the actual downside of the Sobel edge detector can be evaluated on Dogs Image.
B. In the Dogs Image, we can see that the **background noise**-In this case it is grass are detected as edges.
C. Sobel edge detectors are comparatively **faster and easy to implement.**
D. Also, it preserves the overall structure of the image.
E. We can also observe from the Gradient magnitude image that the **edges are thick.**
F. As discussed in B, Sobel filters are **sensitive to noise**.
G. They provide **weak localization.**

2) Canny Edge Detector:

Canny Edge detector is applied to both *Gallery and Dogs Images* and results are compared for different values of low and high threshold values. Figures 26-29 shows the result of Canny Edge detector for different values of thresholds on Gallery Image.

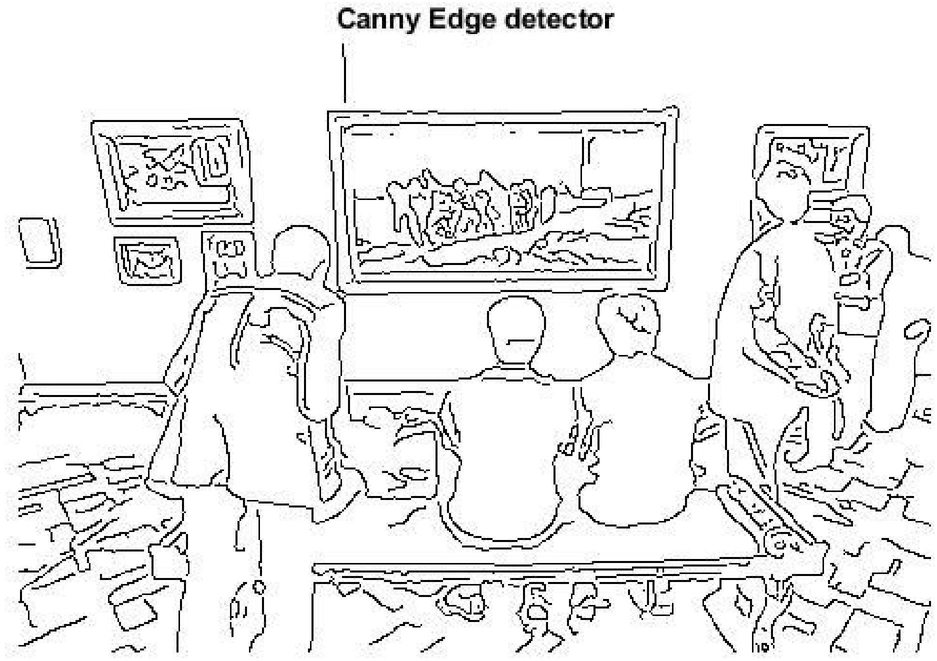


Figure 26: Low: 0.0375 - High: 0.0938 (default value in matlab)

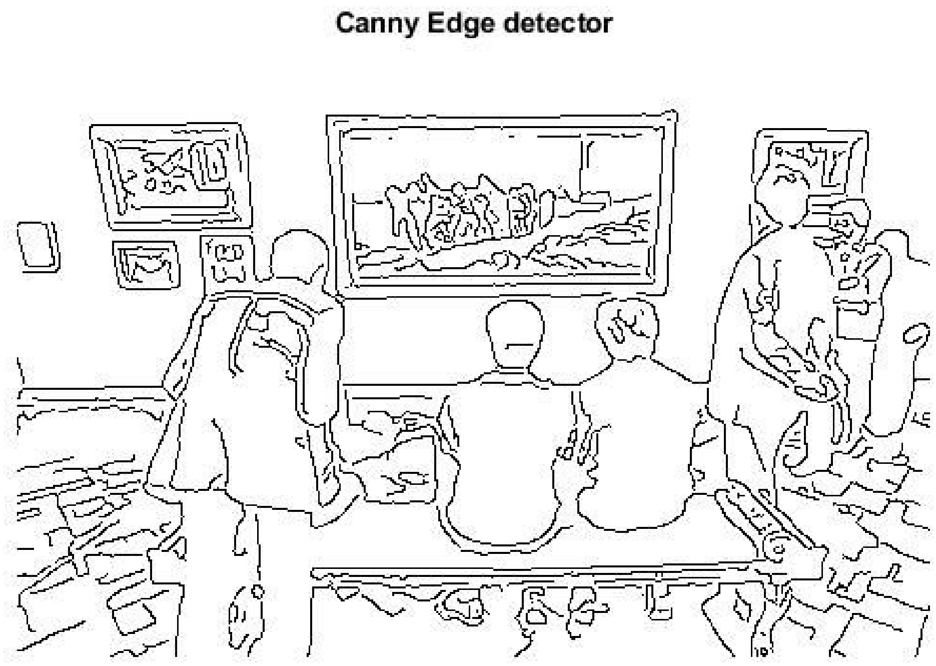


Figure 27: Low: 0.02 - High: 0.1

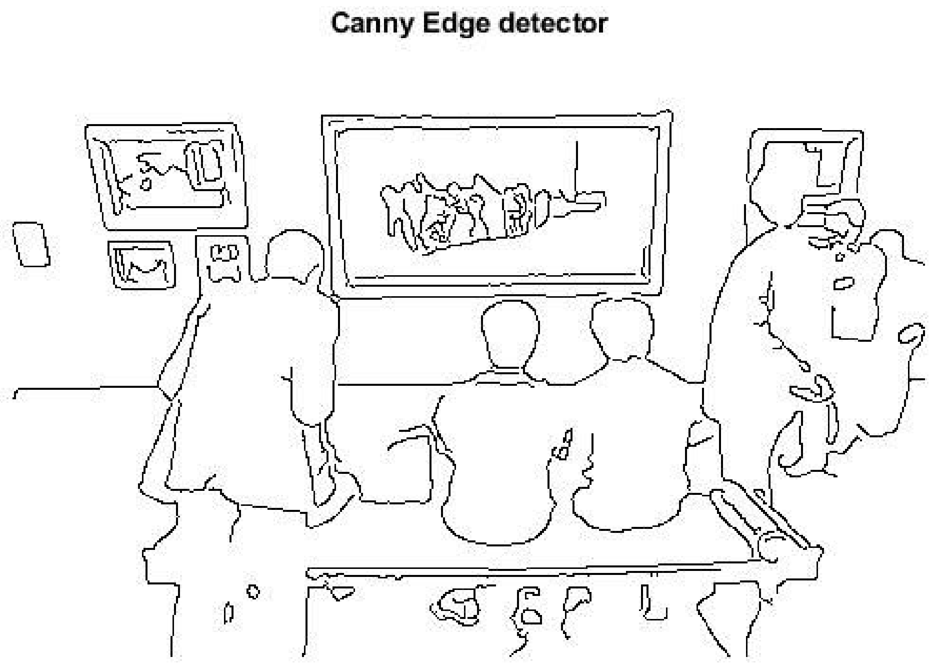


Figure 28: Low: 0.045 - High: 0.25

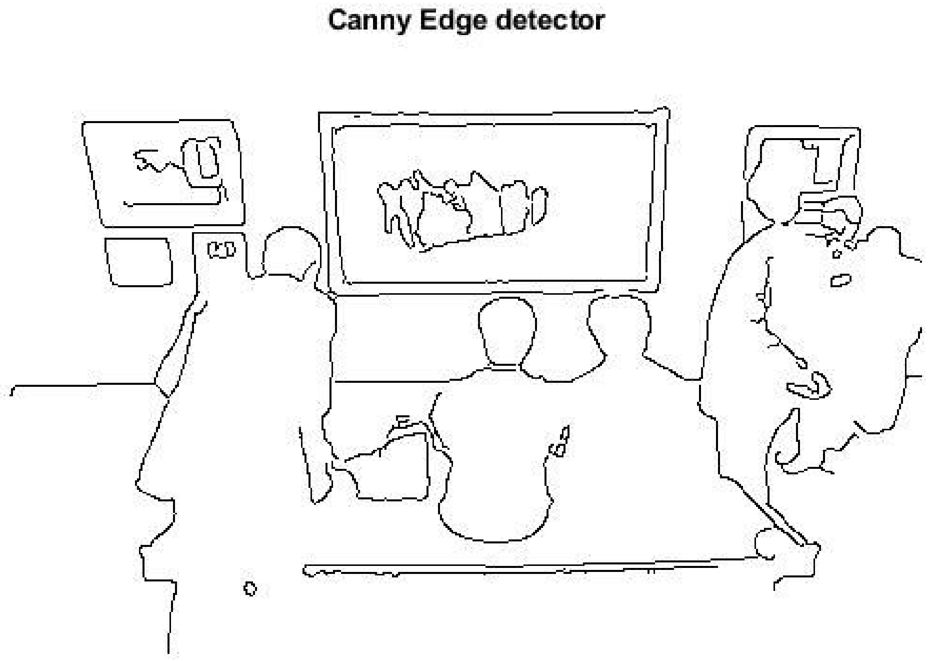


Figure 29: Low: 0.025 - High: 0.4

As we can see from the above Figures, the edge map of the canny edge detector can be efficiently using low and high threshold values. Initially in Figure 27 we didn't vary the thresholds much so the output was almost the same. But later when we increased the high threshold value we started to get clear edges. The textures in the ground got eliminated considerably. **Figure 28 with low threshold value 0.045 and high threshold value 0.25 is the best and optimal result**.

Figures 30-33 shows the output of canny edge detector on *Dogs Image.*

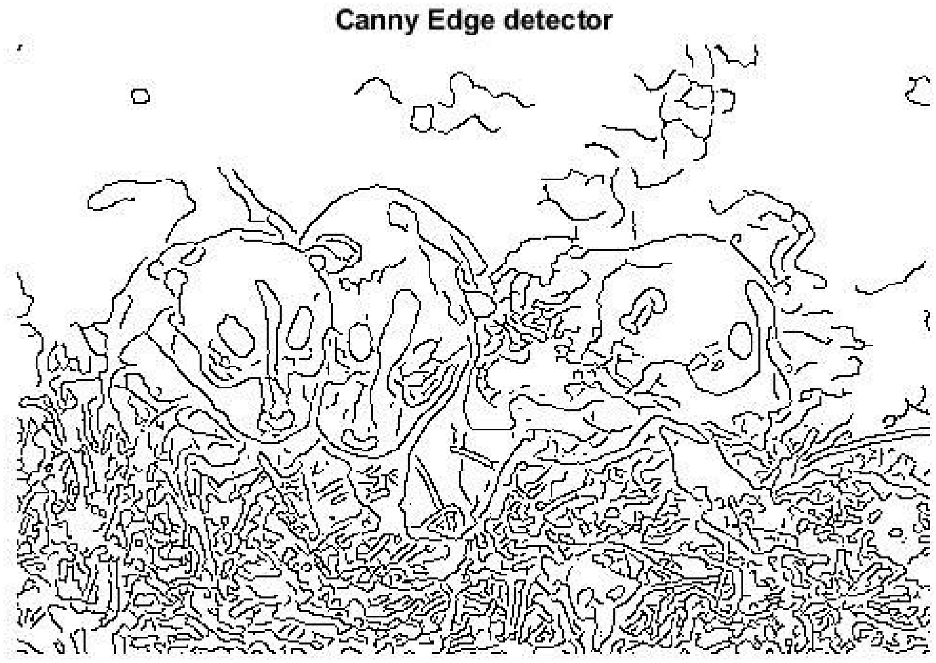


Figure 30: Low: 0.0563 - High: 0.1406 (Default)

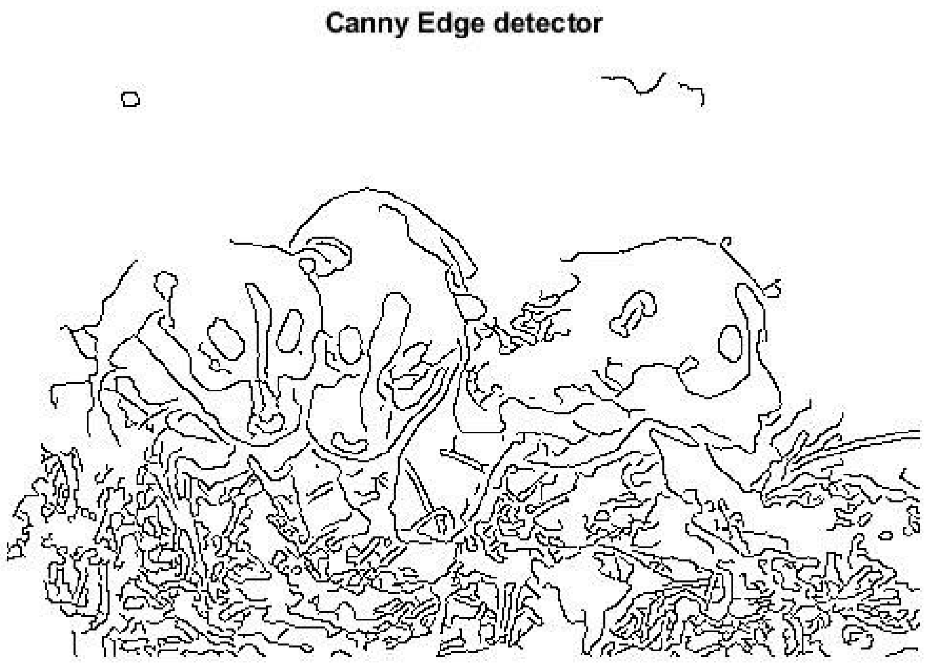


Figure 31: Low: 0.025 - High: 0.25

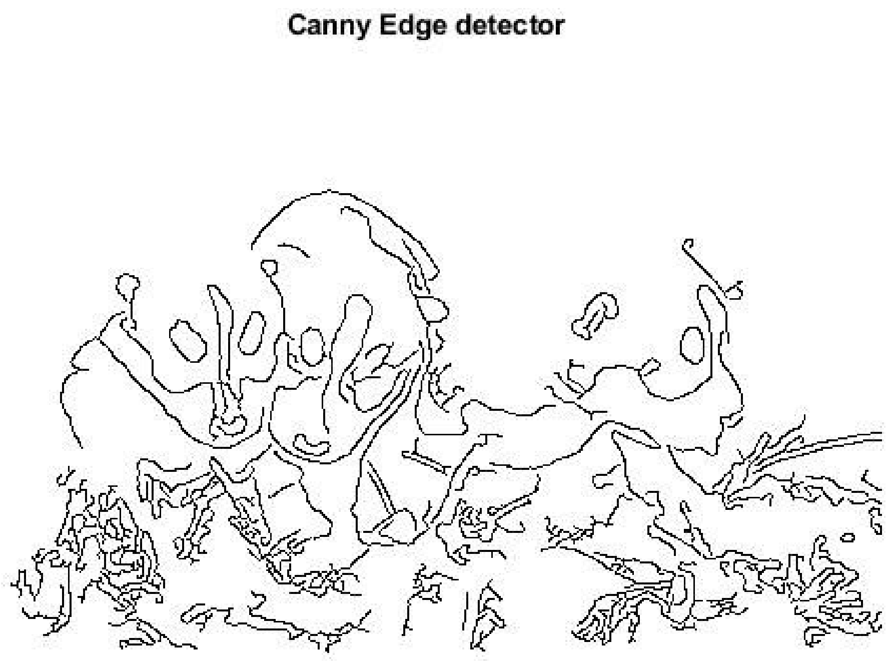


Figure 32: High: 0.0300 - Low: 0.4000

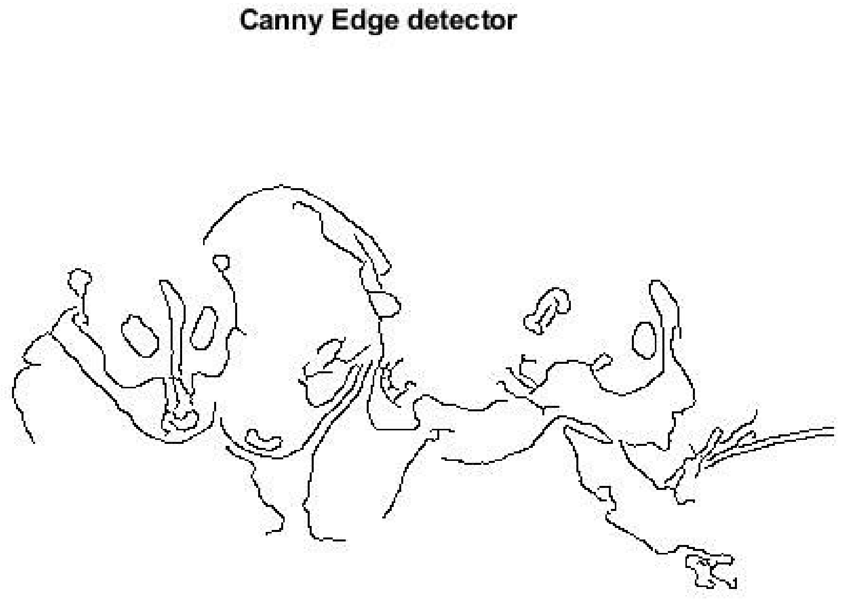


Figure 33: Low: 0.0200 - High: 0.6000

As we can see from the above Figures, the edge map of the canny edge detector can be efficiently using low and high threshold values. Initially in Figure 30 we didn’t vary the thresholds much so the output was almost the same. But later when we increased the high threshold value we started to get clear edges. The textures in the ground got eliminated considerably. **Figure 32 with low threshold value 0.0300 and high threshold value 0.400 is the best and optimal result**.

Observations:

A. As canny employs denoising prior to the edge detection results are better compared to Sobel.
B. For images which have weak edges and have more noise, canny edge can be preferred.
C. Significantly increases Signal to Noise Ratio.
D. Provides **better localisation** compared to sobel.
E. Non max suppression **provides thinner edges**.

F. We get continuous and **connected edges** because of hysteresis thresholding.
G. However, canny edge detectors are **sensitive to textured regions.**
H. Not enough for image interpretation because **Gradient on Luminous only**.

Note: Non Maximal suppression and Hysteresis thresholding is explained briefly in the <u>Abstract and Motivation section</u>

3) **Structure Edge:**

[Note: Structured Edge detection algorithm (1) and Random Forest Classifier (2) is explained with flow chart in Abstract and Motivation section]

The online source code (https://github.com/pdollar/edges) is implemented In matlab using the pretrained parameters on *Gallery Image and Dogs Image.* The model parameters are varied and results are compared. The description of model parameters are discussed below,

1. Multiscale: For top accuracy we need to set it to 1.
2. Sharpen: For top speed we need to set this to 0.
3. nTreesEval: For top speed we need to set nTreesEval to 1
4. nThreads: max number threads for evaluation
5. Nms: set to true to enable Non Max suppression.

Default Model Parameters:
Multiscale : 0
Sharpen: 2
nTreesEval:4
nThreads: 4
Nms: 0

Figure 34-35 shows the Structured Edge result of Gallery and Dog Images using Default Parameters.

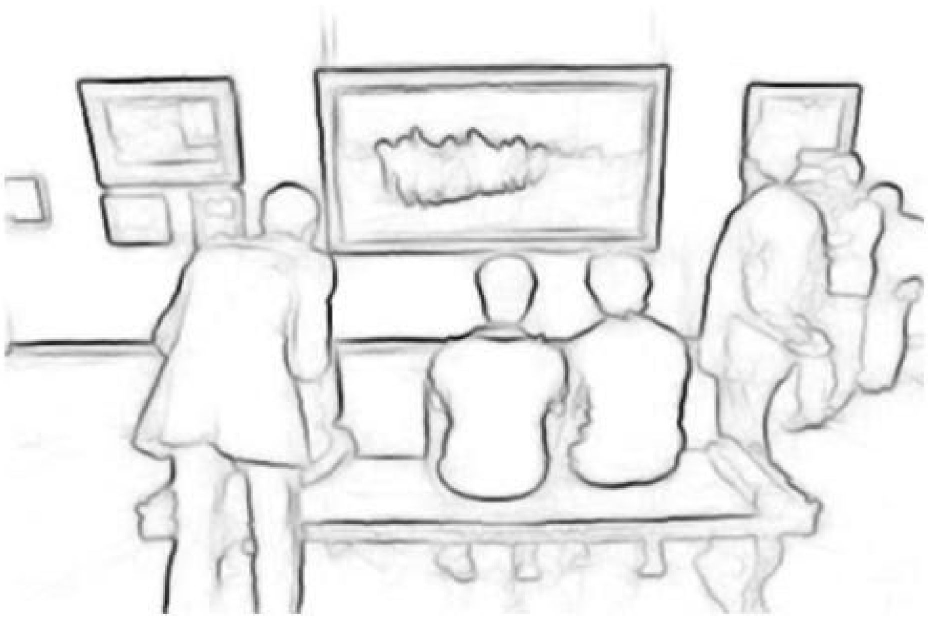


Figure 34: Gallery with Default Parameters (Structured Edge)

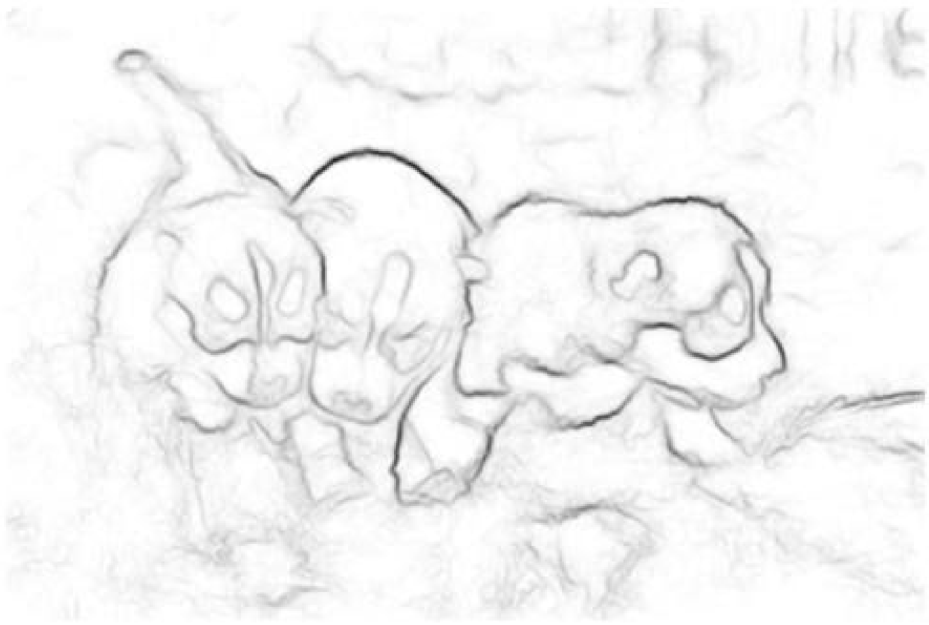


Figure 35: Dogs with Default Parameters (Structured Edge)

Upon changing the parameters with following values, we get better edge maps

1:

Multiscale : 1 (To get the better edge map)

Sharpen: 0

nTreesEval:2

nThreads: 5

Nms: 1 (Enabling Non max suppression)

Figure 36-37 shows the result of parameters (**1)** on Gallery and Dogs Image.

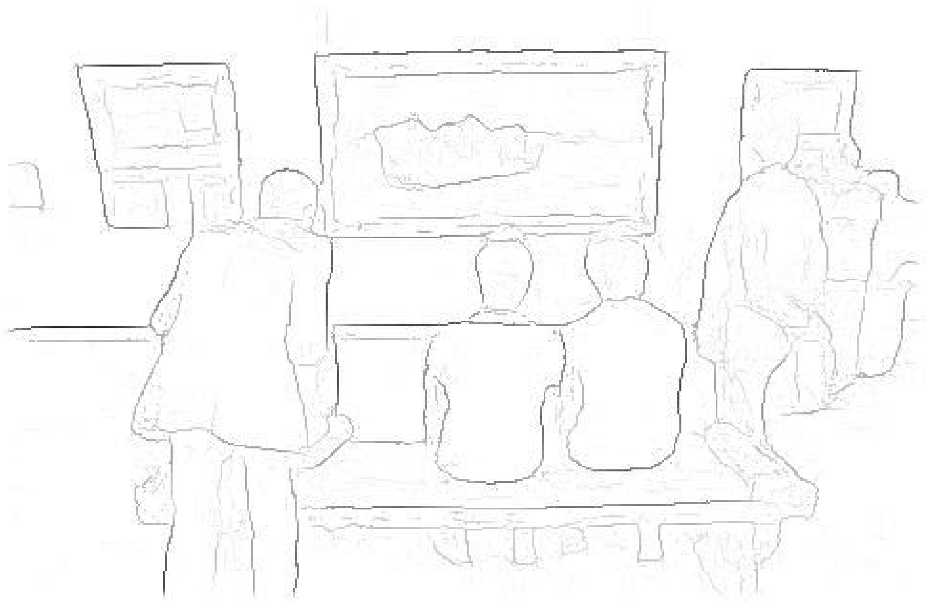


Figure 36: Gallery with (1) Parameters (Structured Edge)

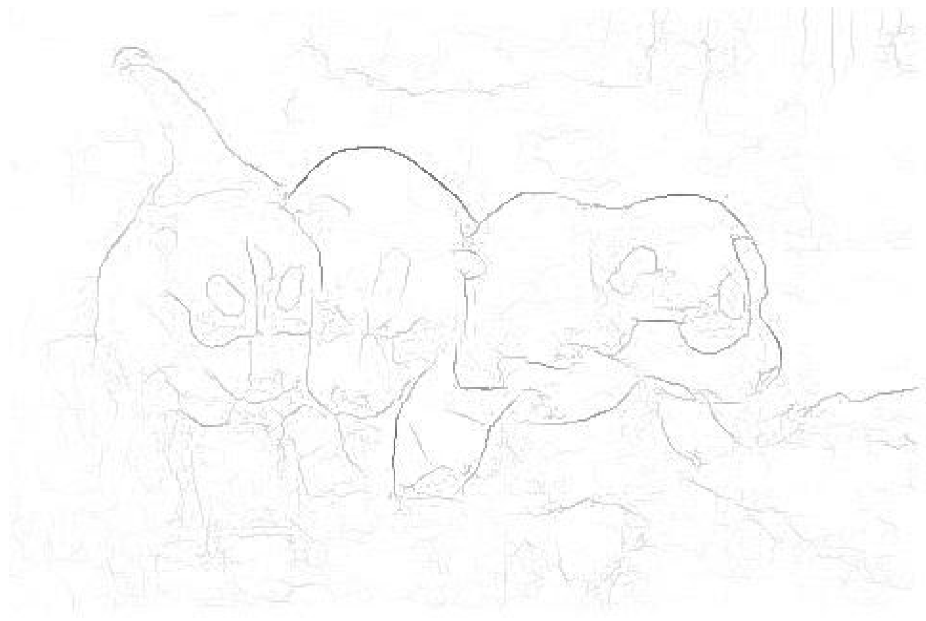


Figure 37: Dogs with (1) Parameters (Structured Edge)

We can clearly see the improved result with thinner edges as we enable Non Max suppression. Also, the accuracy is more compared to the default parameters and speed of execution is faster.

Let us try with new set of parameters labelled as (2)

2:

Multiscale : 0.5

Sharpen: 1

nTreesEval:1

nThreads: 5

Nms: 1

Figure 38-39 shows the result of Structured edge using (2) parameters on Gallery and Dogs Images.

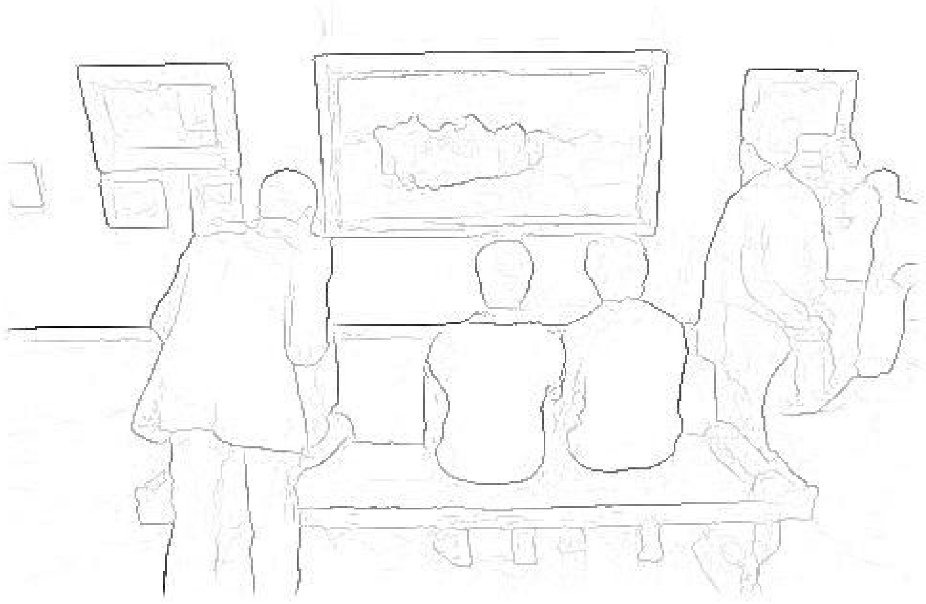


Figure 38: Gallery with (2) Parameters (Structured Edge)

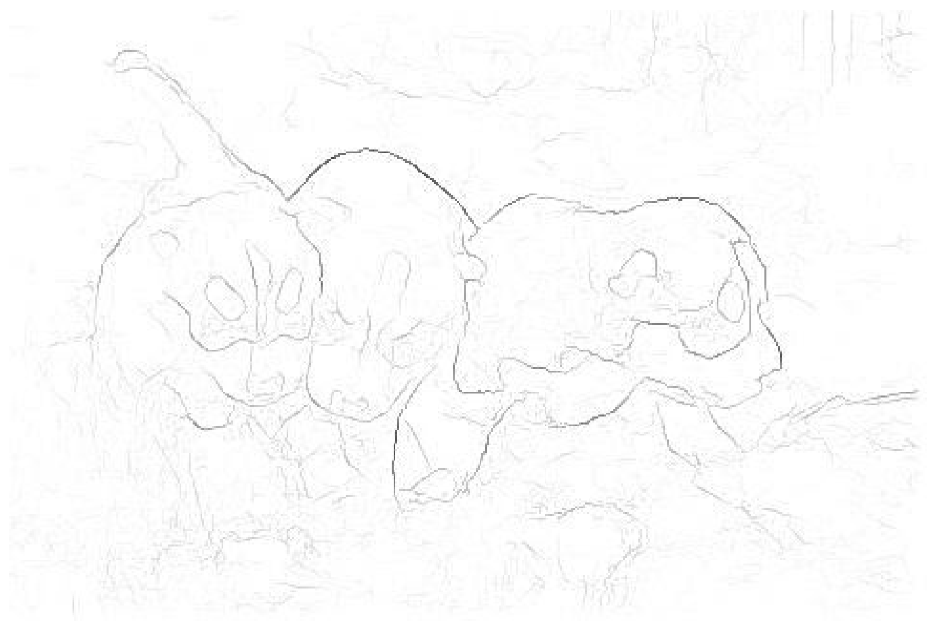


Figure 39: Dogs with (2) Parameters (Structured Edge)

It is apparent that the result with Model parameters (1) is better, I will chose the Model parameters as,

Multiscale : 1 (To get the better edge map)
Sharpen: 0
nTreesEval:2
nThreads: 5
Nms: 1 (Enabling Non max suppression)

<u>Observations:</u>

A. Since this is a supervised learning algorithm the edge map is very clear and mimics the handwritten edge maps.
B. High quality edges are obtained with high efficiency.
C. Completely **eliminates the noise and textures.**
D. Due to high quality edge maps, structured edges can be used in high level vision tasks like object recognition.

E. Very fast in computation and provides better results than competing edge detection techniques like Sketch Tokens.
F. Diversity of trees will **avoid the problem of overfitting.**
G. Downside of Structured edge includes to **Non sensitivity to feature normalization, high output feature dimension (2^{256}), hard to calculate Information gain.**

Comparison between canny edge detectors and Structured edge detectors:

01. As we can see the dog output of the canny edge detector, we can say it's performance is poor against noise. Also varying low and high thresholds may eliminate edge components. This is completely nullified in Structure Edges
02. Unlike Canny, Structure edge is a data driven approach in which labels are generated from human sketches. Hence Structured edges are more accurate than canny edge detectors.
03. Provides better localisation than canny.

4) Performance Evaluation:

a) Structured Edge:

Recall and Precision for each ground truth image in calculated for different threshold values (0.1 - 1 in steps of 0.1)

Note: Images with Model Parameters (1) are used.

Gallery Image:

Ground Truth	Recall	Precision	F-measure
1	0.95026	0.818	0.8801
2	0.7725	0.94	0.8480
3	0.7262	0.9727	0.8315
4	0.6683	0.9851	0.7963
5	0.6824	0.9978	0.81049

Mean Precision = 0.759932
Mean Recall = 0.94272
Mean F-measure = 0.84151

Dogs Image:

Ground Truths	Recall	Precision	F-measure
1	0.75131	0.694754	0.72
2	0.4007	0.9155	0.648
3	0.6733	0.9467	0.78
4	0.8777	1	.88
5	0.7899	1	.78

Mean Precision = 0.911
Mean Recall = 0.6984

Mean F-measure = 0.7906

Also the F-Measure is calculated for different threshold values and plotted. Figure 40 shows the plot for Gallery.

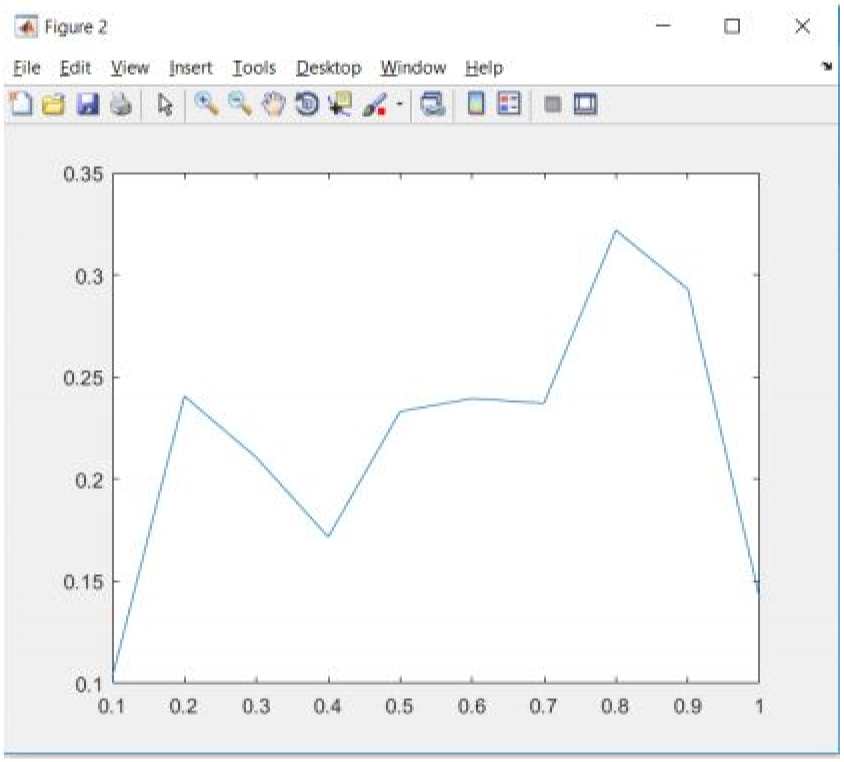


Figure 40: F-measure for Gallery Image for 10 Threshold values

We can see that the peak value of F-measure is obtained for Threshold 0.33.

a) From the above discussion we can conclude that the performance of Structured Edge > Canny > Sobel. The Performance, Pros and Cons of each edge detection algorithm is discussed in their respective sections.
b) We can clearly see that the F-measure value is image dependent based on the previous calculations. By the Mean F-measure values for Gallery and Dogs Image we can conclude that **Gallery Image** is easier to get high precision value because it has no background noise like Dogs image (grass). Also, In Dogs' images there are a lot of textures like hairs on Dogs which reduces the F-measure. So, Gallery Image is easier to get higher F-measure compared to Dogs.
c) The Rationale behind F-measure is explained in the previous section (Abstract and Motivation). It is **not possible** to get a higher F measure if the precision is higher because higher precision is obtained by decreasing the threshold value. However, this reduces the recall. So we have to consider both Precision and recall to get higher F-measure. Vice Versa is also true.

i) If Precision + Recall = constant
So, F = (2*P*(constant-P))/constant
Therefore, if we differentiate this and equate to 0 we get maximum value.
So, P = constant-P
P=R

Another intuitive explanation is,

F= (2*P*P)/(P+P) - If P=R
F = 1 which is the maximum value.

Problem 2: Digital Half-toning

I. Abstract and Motivation:

Digital half-toning is a process of transforming a gray scale/color image to a halftone image i.e., binary representation of a pixel. But this seems complicated because, for gray level values in the image it is very difficult to represent the pixels with binary values. Nevertheless, when images meet printers we need to represent the image with a fixed level of values. So, the main agenda of the digital halftoning algorithms is to create a perception of continuous scale pixel values using a handful of discrete values (Quantization) which is capable enough to fool our visual system [5]. Figure 5 shows the Haltoning output of a continuous gray scale image.

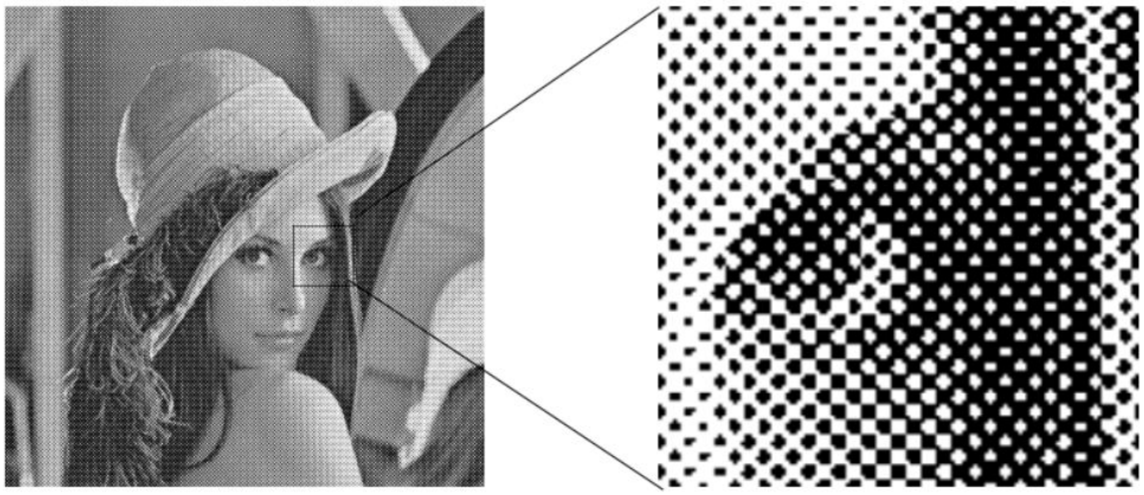


Figure 5. Example of Digital Halftoning.

There are different Halftoning algorithms and are listed as follows,

1. Dithering (Point Process)
2. Error Diffusion (Neighborhood algorithm)
3. Iterative methods.

In this section we broadly discuss and analyze the results of Dithering and Error diffusion algorithms. At the end, we also discuss a few color halftoning algorithms and evaluate their performances.

1) **Fixed Thresholding:** One of the Naive approaches of digital halftoning is to set a fixed threshold value (conventionally 128) and set the value to 1 if it is greater than 128 else set it to 0. Below equation shows the operation of fixed thresholding.

$$b(i,j) = 255 \; if \; X(i,j) > T(i,j) \; else \; 0$$

where $b(i,j)$ is the mapped image, $X(i,j)$ is the input image.

2) **Random Thresholding:** Fixed Thresholding creates monotones in the image which often look patchy in one location and plain in other locations. Inorder to avoid this we generate random threshold value for each pixel and if the pixel value of the image is greater than the random number we set the pixel value to 1 else 0.

Usually we choose Uniform random variable to generate a random number. Below equation shows the operation of Random Thresholding.

$$b(i,j) = 255 \; if \; rand(i,j) \le X(i,j) < 256, \; 0 \; if \; 0 \le X(i,j) < rand(i,j)$$

3) **Ordered Dithering:** Ordered Dithering is a halftoning technique commonly categorized as point process, where we turn a pixel on or off in a certain pattern which gives the perception of a continuous color tone. We generate an index matrix (Dithering matrix) and corresponding threshold matrix to implement ordered Dithering. Dithering matrix and generation of corresponding threshold matrix is shown in below equations.

$$I2 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$I2$ is the base Index matrix using which we can generate higher order Index (Dithering) matrix. The equation is shown below.

$$I2n = \begin{pmatrix} 4*In+1 & 4*In+2 \\ 4*In+3 & 4*In \end{pmatrix}$$

$$I4 = \begin{pmatrix} 5 & 9 & 6 & 10 \\ 13 & 1 & 14 & 2 \\ 7 & 11 & 4 & 8 \\ 15 & 3 & 12 & 0 \end{pmatrix}$$

Index matrix should be translated to a respective threshold matrix where we can apply to the image patches.

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

The output of the Image can be represented as given in the equation below.

$$G(i,j) = 0 \; if \; F(i,j) \leq T(i \; mod \; N, \; j \; mod \; N) \; else \; 255$$

In the similar way we construct 4*4, 8*8, 16*16 and 32*32 Threshold matrices and compare the result.

4) **Error Diffusion:** The main shortcoming of Dithering is that we often see texture like visual patterns. Inorder to avoid this, we add noise before quantizing the pixel values because without noise we can't able to see the variation. So, noise adds randomness to the image (which is 0 mean) which helps us to provide variations. However, the pursuit of noise didn't get any breakthrough over the years. But Error diffusion gave promising results through quantization and propagating the errors. The noise added to the pixel value was succinctly satisfying the requirement by adding randomness to get the desired variations.

In Error Diffusion, pixel values are quantized in a specific pattern like raster or serpentine pattern. After quantizing the error is propagated to the local pixel values so that the average intensity of the image will be tied closer to the original image [7]. Figure 6 shows the block diagram of the approach.

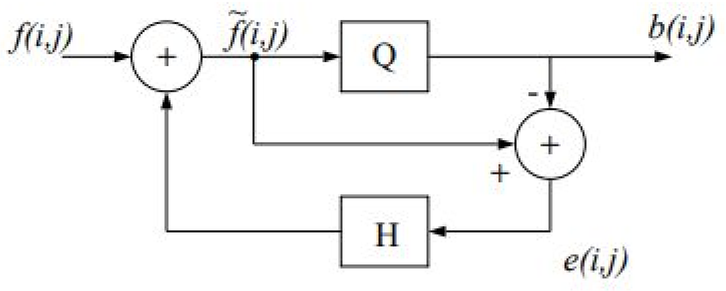


Figure 6. Block diagram of error diffusion method

The error generated by the quantizing the pixel $e(i,j)$ is propagated to the future pixel values using a 2D filter called **Diffusion filter.** The equation of the operation is given below.

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

Figure 7 shows how quantization error is diffused to the future pixels.

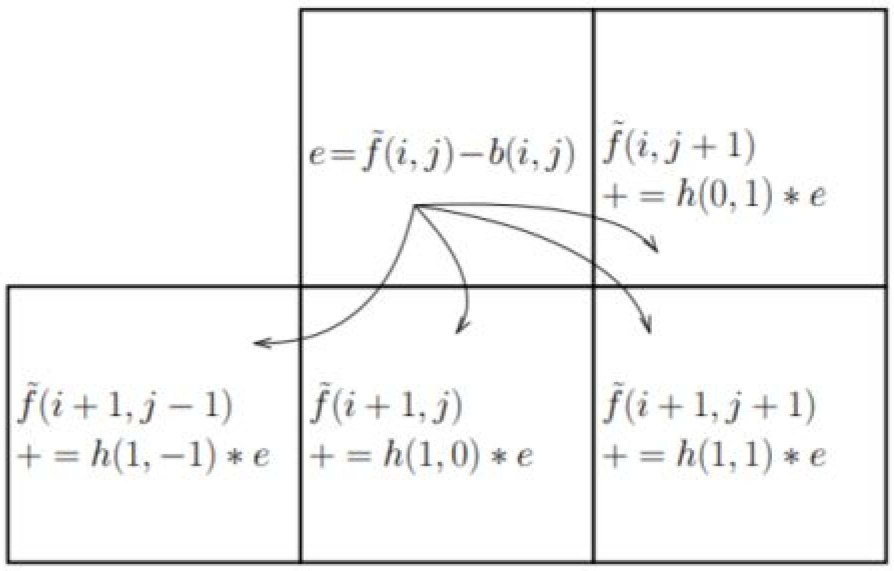

$e = \tilde{f}(i,j) - b(i,j)$
$\tilde{f}(i,j+1)$ $+= h(0,1) * e$
$\tilde{f}(i+1,j-1)$ $+= h(1,-1) * e$
$\tilde{f}(i+1,j)$ $+= h(1,0) * e$
$\tilde{f}(i+1,j+1)$ $+= h(1,1) * e$

Figure 7. Propagation of Quantization error to the future pixels.

In this assignment, we follow the serpentine pattern for quantization of the pixels. The schematic of the pattern is shown in Figure 8.

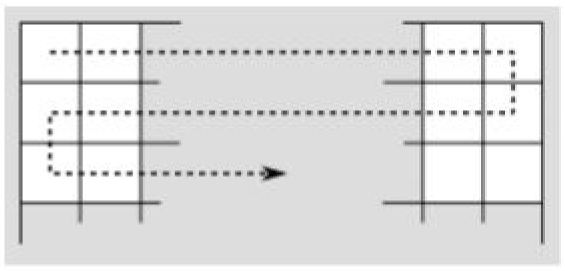


Figure 8. Serpentine Pattern for Quantization

In the Serpentine pattern, the error is diffused while traversing from left to right and right to left alternatively. While traversing from right to left we need to mirror the diffusion filter. There are popular diffusion filters proposed by many experts. In this assignment we discuss and implement a few of them and compare our results.

01. **Floyd-Strinberg Error Diffusion:** The very popular error diffusion technique was proposed by Floyd and Stienberg. In this method,

the error is diffused to more than one neighboring pixels using following filter coefficients shown in Figure 9.

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

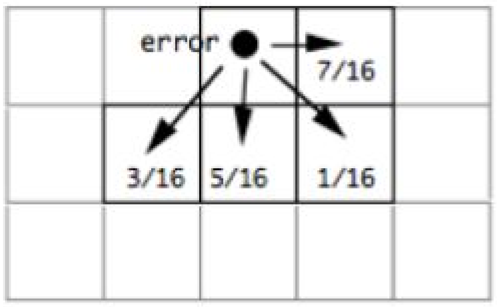


Figure 9. Floyd-Steinberg Algorithm for Error Diffusion

The algorithm gives better results compared to ordered dithering.

02. **Jarvis, Judice and Ninke Error Diffusion:** This algorithm is very similar to the Floyd-Stienberg algorithm. Here, the quantization error is propagated to the pixels more than 3*3 local neighbors i.e., 5*5 neighbors. The filter coefficients are shown in Figure 10.

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

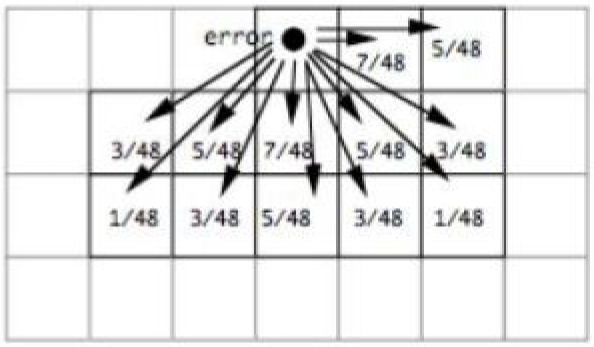


Figure 10. JJN Error Diffusion Filter

03. **Stucki Error Diffusion Matrix:** This is a slight variation of JJN Error diffusion matrix. Figure 11 shows the Diffusion matrix proposed by Stucki.

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

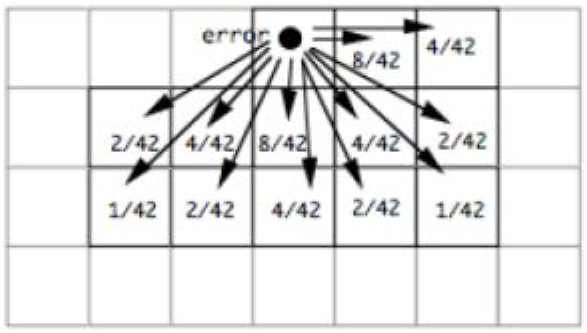
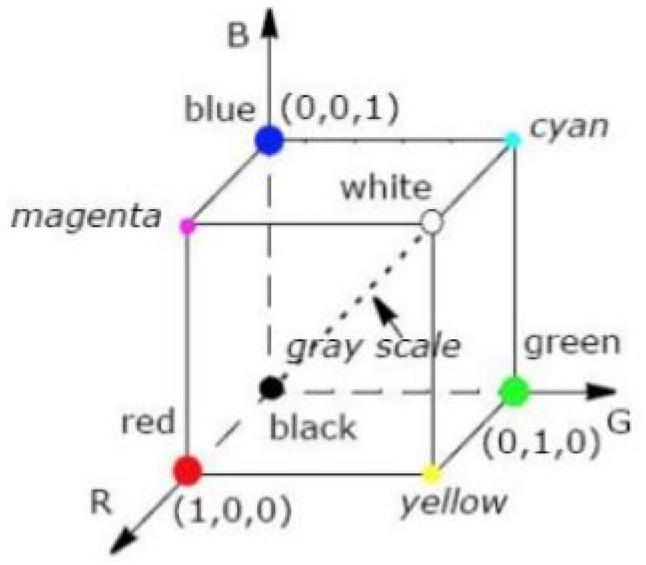


Figure 11. Stucki Error Diffusion Filter

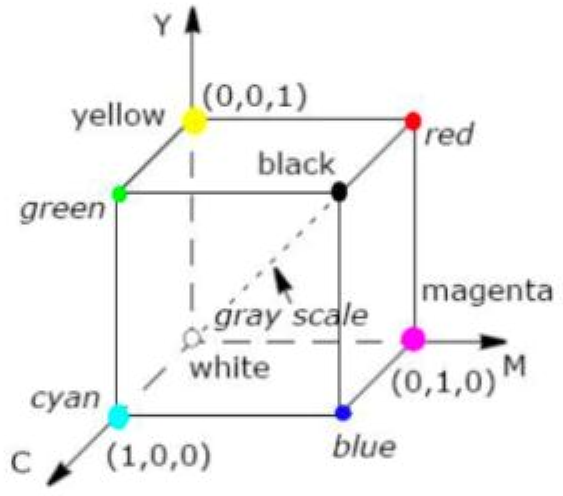
4) Color Halftoning with Error Diffusion: Error diffusion approach was originally developed for Gray scale images. But it was extended to color images by diffusing the quantization error in each channel separately. Just like quantizing the gray level to values to either 255 or 0 based on thresholding, we quantize the image to one of the 8 color values (cartesian coordinates). Figure 12. shows the representation of RGB color cube.



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

Figure 12. RGB color cube representation.

Similarly Figure 13 shows the CMY Cube representation.



1. **Figure 13. CMY color cube representation**

There are many state-of-the-art algorithms for Color halftoning. In this assignment we implement two popular color halftoning and discuss the results.

A. **Separable Color diffusion:** In separable error diffusion, we seperate each RGB channel and diffuse the quantizing error to the future pixels. But we use the CMY color model to do this because CMYK color model is the popular color model for printing. So, we convert RGB channels to CMY channels and each channel is quantized based on a Threshold value (conventionally 128) and the quantization error is pushed to the neighbor pixels using Floyd-Stienberg diffusion matrix. After this we convert the image back to RGB. The flow diagram of separable error diffusion is shown in Figure 14.

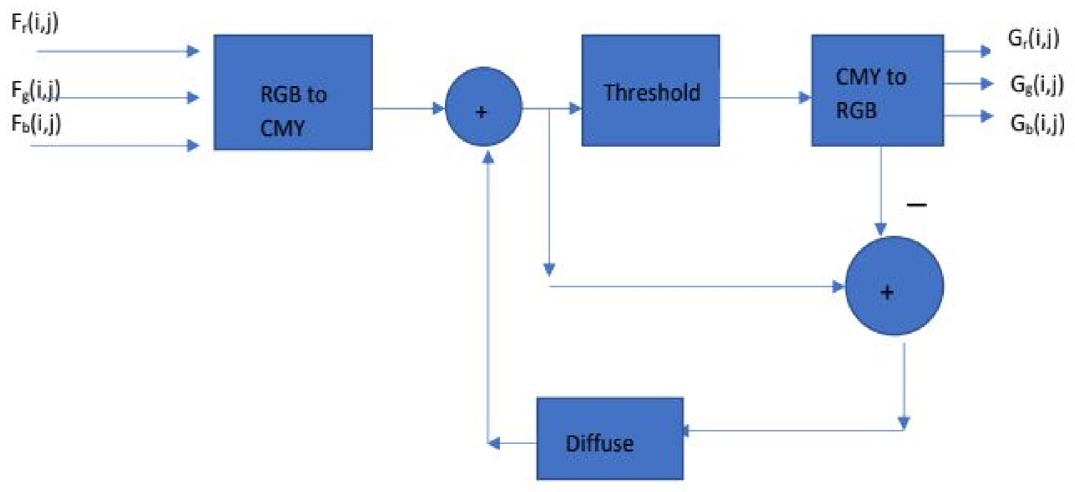


Figure 14. Flow diagram of Separable Error diffusion.

B. **Minimal Brightness Variation Quadruples (MBVQ):** Doron et.al proposed a novel color halftoning algorithm which is basically an extension of the separable error diffusion by exploiting the characteristics of human color perception [8].The artifacts in the monochrome halftone algorithms are mainly due to the **variation in the brightness.** In separable error diffusion, we quantize a pixel to one of the 8 basic colors shown in Figure 12. But, any color can be rendered efficiently using only 4 colors. The main idea behind the rise of new algorithms is “When presented with high frequency patterns, the human visual system applies a low pass filter and perceives only their average”[8]. So, from this premise we can say that the human visual system is more sensitive to variation in brightness than chrominance. Leading to this concept they proposed MBVC algorithm (Minimal brightness variation criterion) in which respective colors were rendered using the set of halftones which have minimum variation in brightness.

The main agenda of MBVQ is to preserve the average color of the pixel along with reducing the brightness variation. This reduces the number of required halftones to 4. The region where the pixel belongs is called

Minimal Brightness Variation Quadruple and it’s one among 6 tetrahedrons-**RGBK, WCMY, MYGC, RGMY, RGBM, CMGB.** Thus given R, G, B pixel values we find the MBVQ region which is one among 6 tetrahedrons. Figure 15 shows the 6 quadruples.

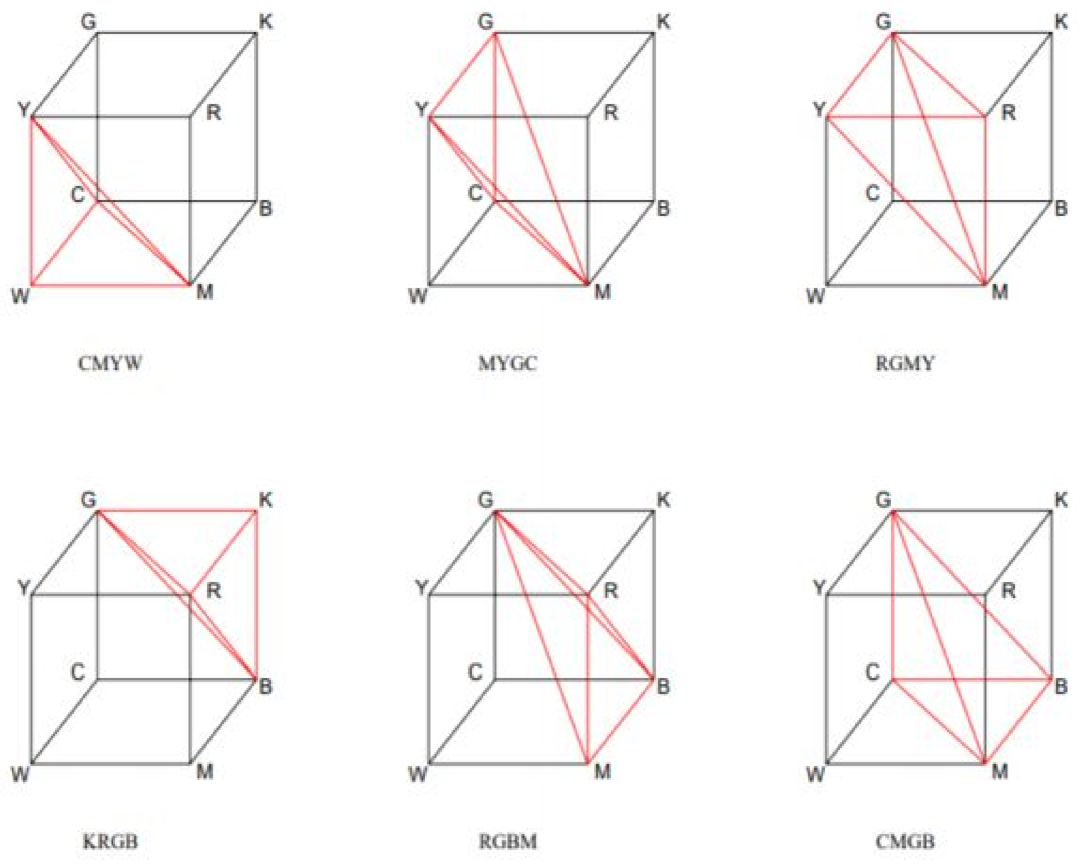


Figure 15. The partition of RGB values to six tetrahedrons

The algorithm is described in the following steps,

01. Arrange 8 colors in the increasing order of brightness. This is shown in Figure 16.

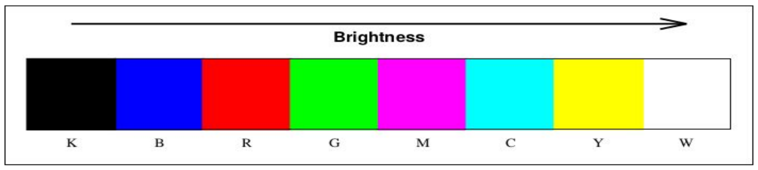


Figure 16. Brightness scale of RGB CMYK.

02. Transform black and white halftones to green and magenta.
03. If (02) eliminated white, we transform black and primary combination to corresponding secondary color halftones.
04. If black halftones are intact then MBVQ is RGBK.
05. If (02) removes the black halftones we follow (03) but with pairs with white.
06. If black halftones are still intact, MBVQ is WCMY.
07. This leaves 4 possibilities of quadruples.

The algorithm is summarized by the algorithm shown in Figure 17.

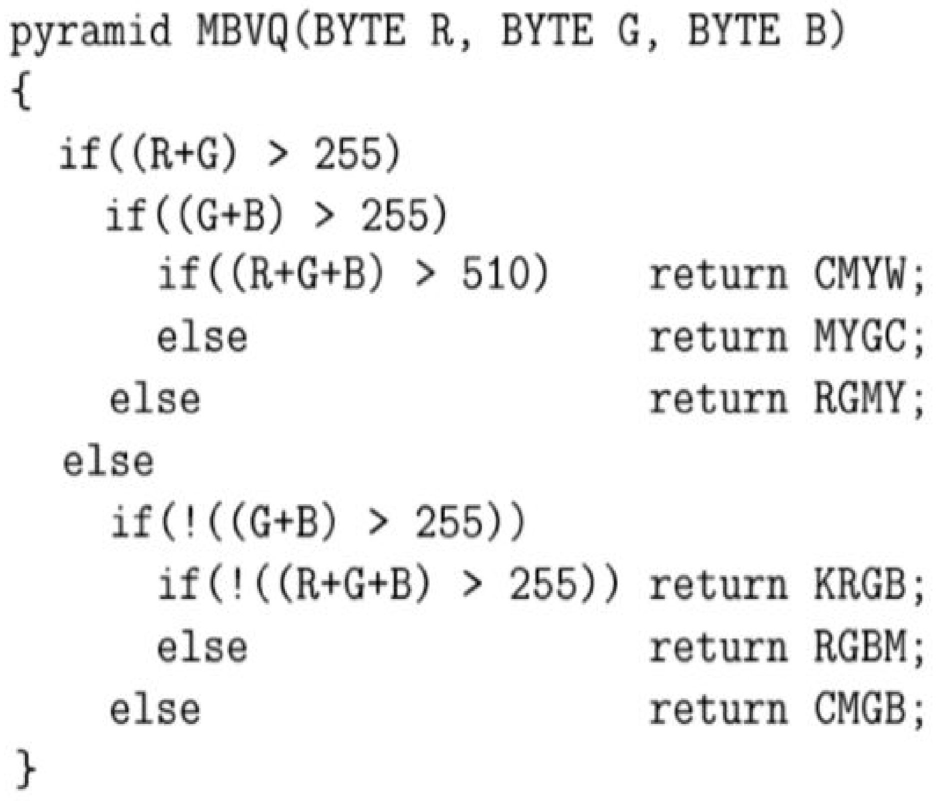

```
pyramid MBVQ(BYTE R, BYTE G, BYTE B)
{
  if((R+G) > 255)
    if((G+B) > 255)
      if((R+G+B) > 510)    return CMYW;
      else                 return MYGC;
    else                   return RGMY;
  else
    if(!((G+B) > 255))
      if(!((R+G+B) > 255)) return KRGB;
      else                 return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

Figure 17. Algorithm to classify a pixel to one of 6 quadruples

Now the problem is reduced to finding the nearest pixel among 4 pixels in the tetrahedron. The output pixel is assigned with the color values of the vertices it belongs and the quantization error is found. The error is further diffused to the local neighborhood using Floyd-Stienberg diffusion matrix. The algorithm for finding the nearest vertices in a given tetrahedron is tricky and varies for different tetrahedrons. One of the algorithms is shown in Figure 18.

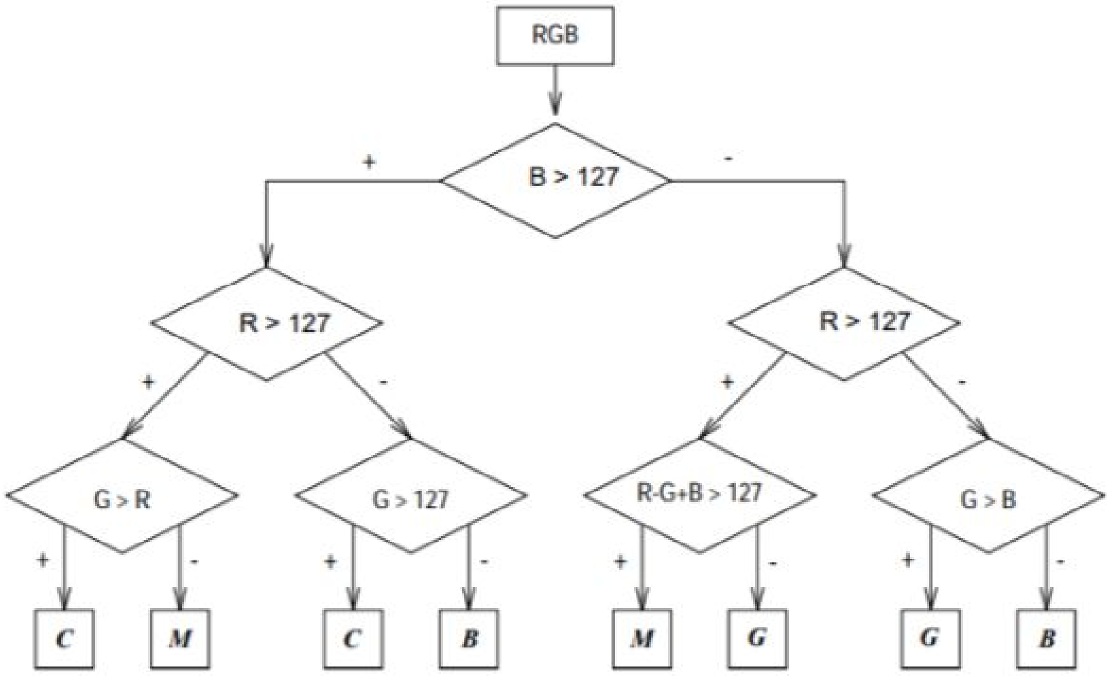


Figure 18. Decision tree for finding nearest vertex of CMGB

II. Approach and Procedure:

1) **Fixed Thresholding:**
    a) Used a fixed threshold of 128 and implemented the algorithm.
    b) Results were noted and analyzed.

2) **Random thresholding:**
    a) Used built-in function in c++ to generate uniform noise in the range of 0-255.
    b) Generated the random threshold for each pixel and output image is obtained.
    c) Analyzed and compared the result with Fixed thresholding.
3) **Dithering:**
    a) Designed a recursive algorithm to generate I2, I4, I8, I16, I32 Index matrices.
    b) Obtained the respective threshold matrices using the equation given above.
    c) Compared the threshold for each block with the pixel values to generate the Dithering Image.
    d) Results were analyzed and compared.
4) **Error Diffusion:**
    a) Wrote an iterative algorithm to generate serpentine pattern i.e, left to right in even rows and right to left in odd rows.
    b) Stored Floys-Stienberg, JJN and Stucki diffusion filters in a 2D array.
    c) Iterate through the image and threshold based on 128.
    d) Calculate the quantization error for each pixel and diffuse the error to future pixels using the above methods.
    e) Stored the image and results of all three methods were compared.
5) **Color Halftoning:**
    a) First implemented the separable error diffusion using Floyd-Stienberg diffusion matrix.
    b) Convert the image to CMY color model.
    c) Perform error diffusion using FS algorithm on each channel separately.
    d) Convert back to RGB and write the image and results were observed.
    e) For MB-VQ, we calculate the one of 6 tetrahedron to which a pixel belongs to.
    f) Then we find among which 8 vertices the pixel belongs to and assign the pixel value of that vertex.

g) Compute the quantization error in each channel and diffuse the error to future pixels.

III. Results and Discussions:

1) **Dithering:** The algorithm is tested on the light house picture shown in Figure 19.

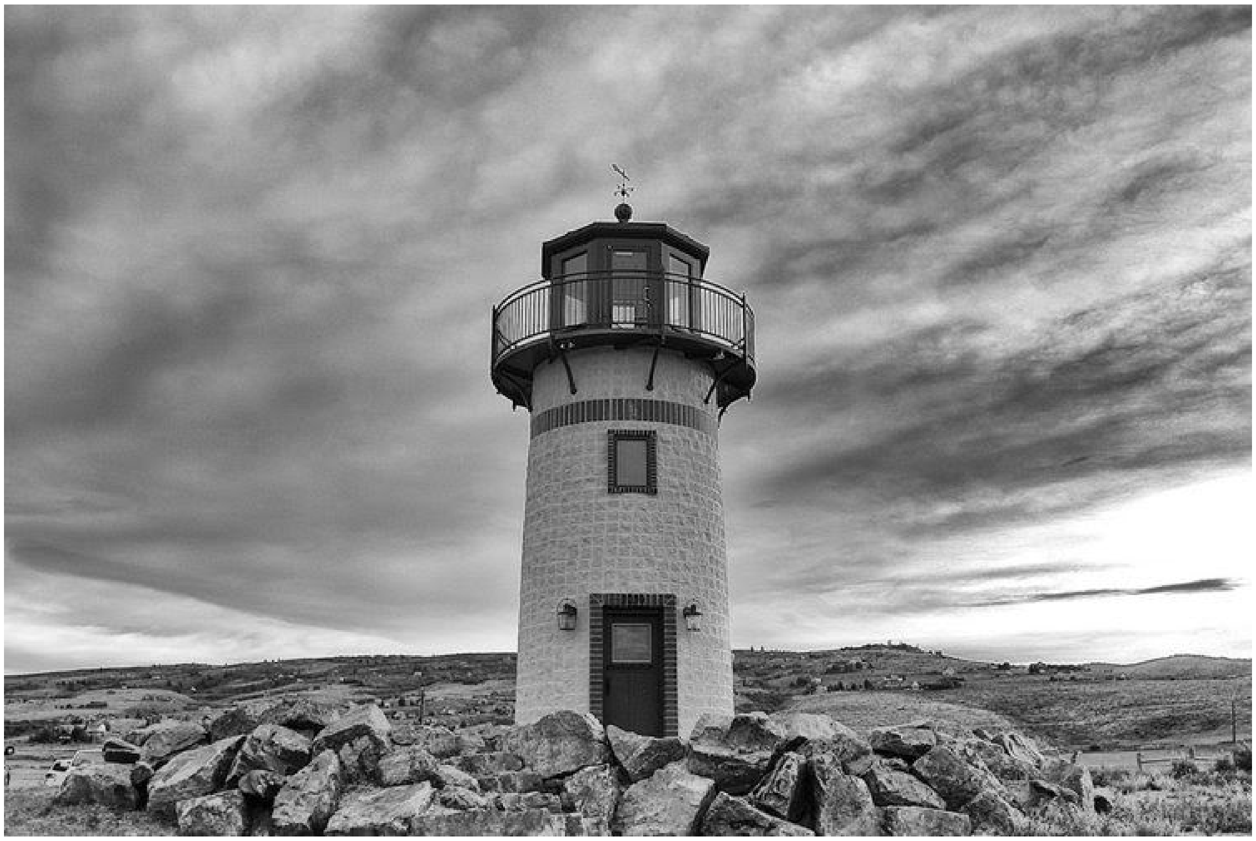


Figure 19: Lighthouse Image

a) ***Fixed Thresholding:*** The result of halftone image obtained by fixed thresholding using threshold value 128 is shown in Figure 20.

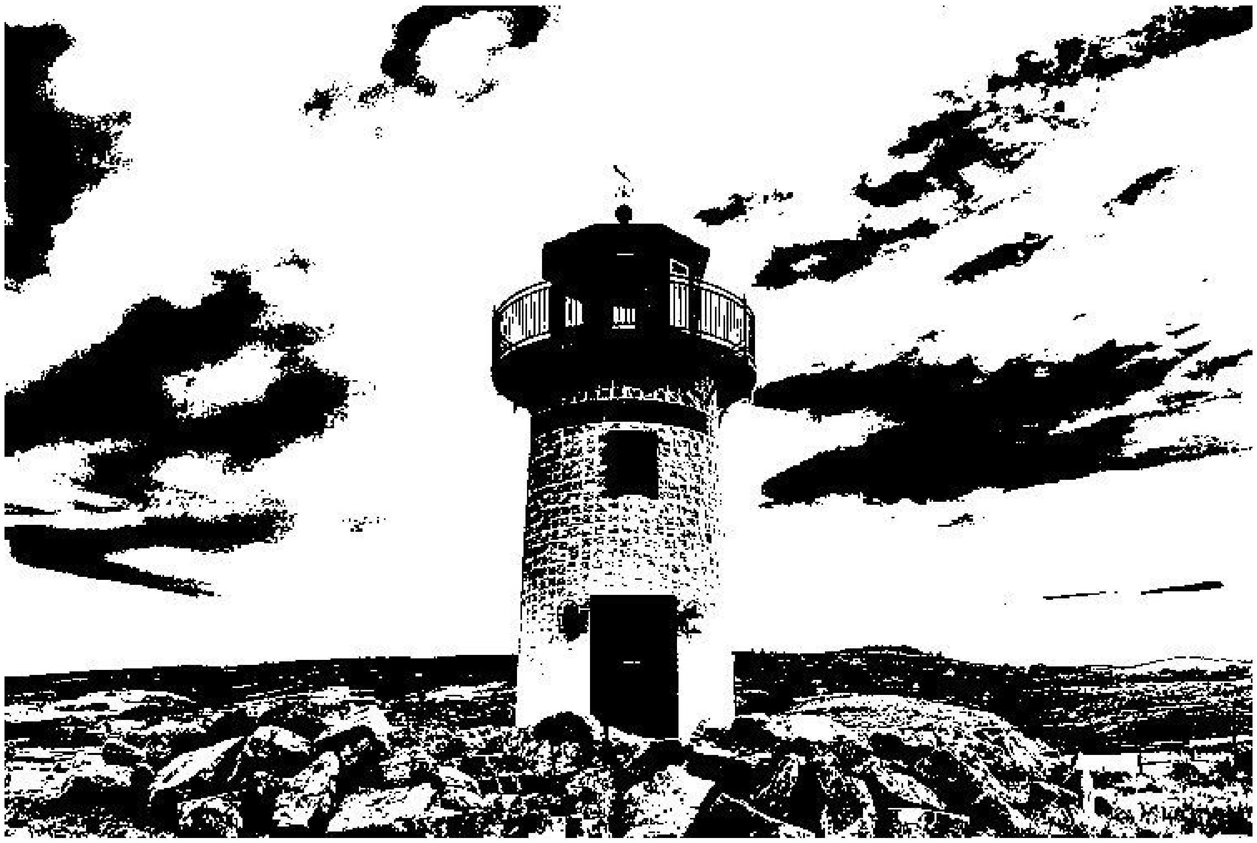


Figure 20. Result of Fixed thresholding (128)

Observations:

A. This is the simplest method to quantize the input pixel value.
B. We can observe that the image is not properly shaded i.e., textures are not properly represented.
C. This results in the artifacts in the haltoned image known as “False Contouring.”
D. False contouring usually occurs because we are quantizing at a lower bit rate. In this case which is 1 bit. Also, the quantization error is strongly correlated with input image pixels which result is poor image rendering.

b) ***Random Thresholding:*** In this method, we generate random threshold for each pixel using a uniform random variable. The result is shown in Figure 21.

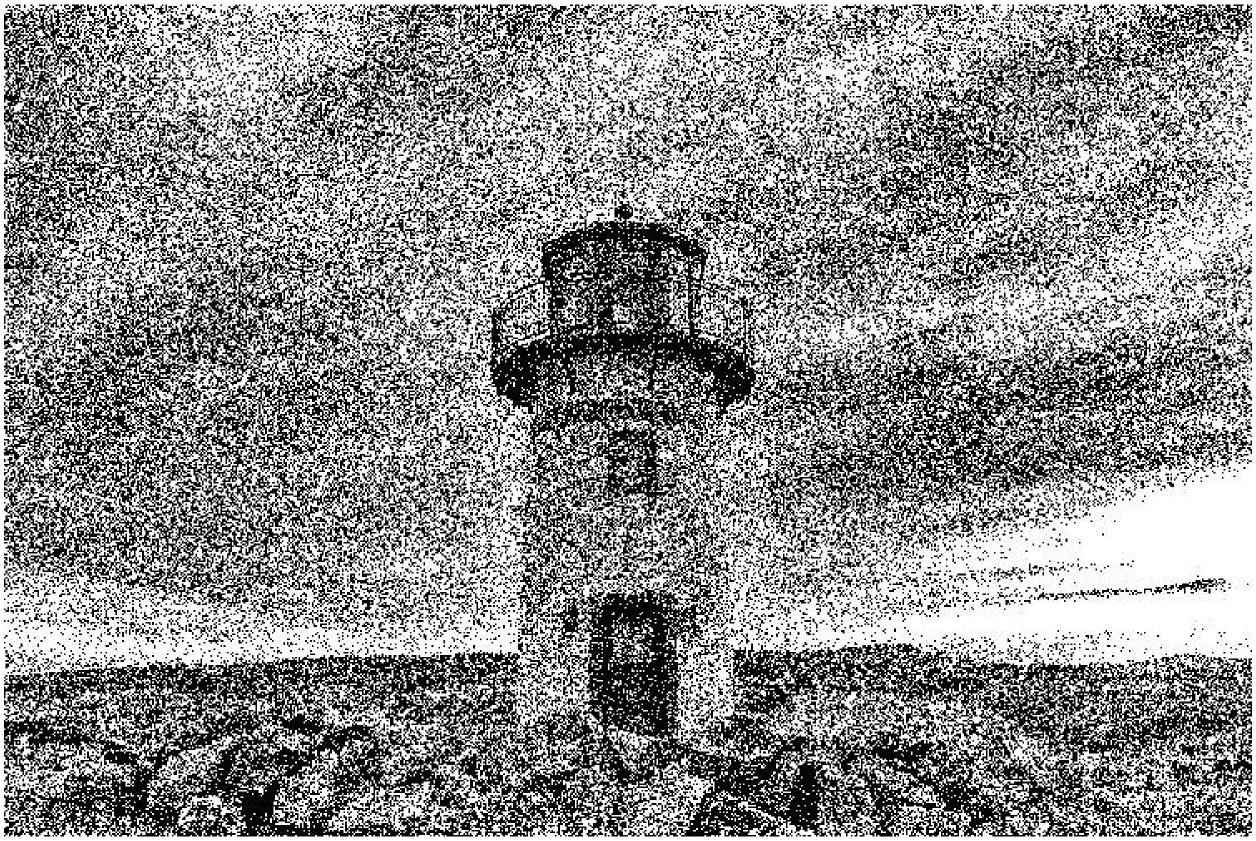


Figure 21: Result of Random Thresholding (Uniform RV)

Observations:

A. This approach reduces the correlation between input pixels and error by adding white noise independently to each pixel.
B. This method breaks the **monotones** of the result of fixed thresholding
C. Even though the resulting image is noisy, the **false contouring** has been reduced drastically compared to fixed thresholding
D. With sufficient blurring, the result gives the impression of having some gray levels compared to fixed thresholding.
E. Although this method is faster with lesser memory, the visual fidelity is very poor and SNR is very low.

F. However, the quantization error is not taken care of and the root mean square error between input and output image is high.

c) ***Dithering:*** We have generated 2*2, 8*8 and 32*32 thresholding matrix using base index matrix recursively. The results are shown in Figure 22,23 and 24 respectively.

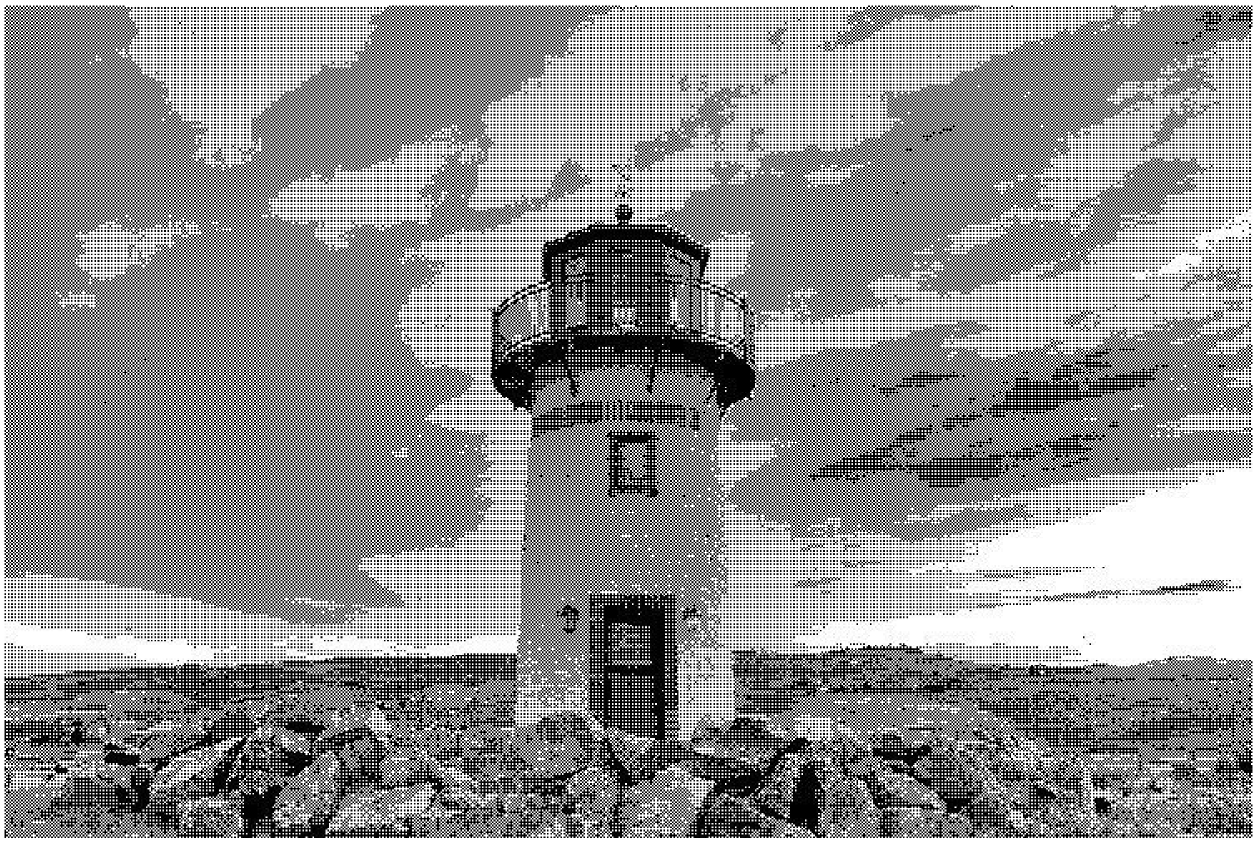


Figure 22: Ordered Dithering using 2*2 Threshold matrix

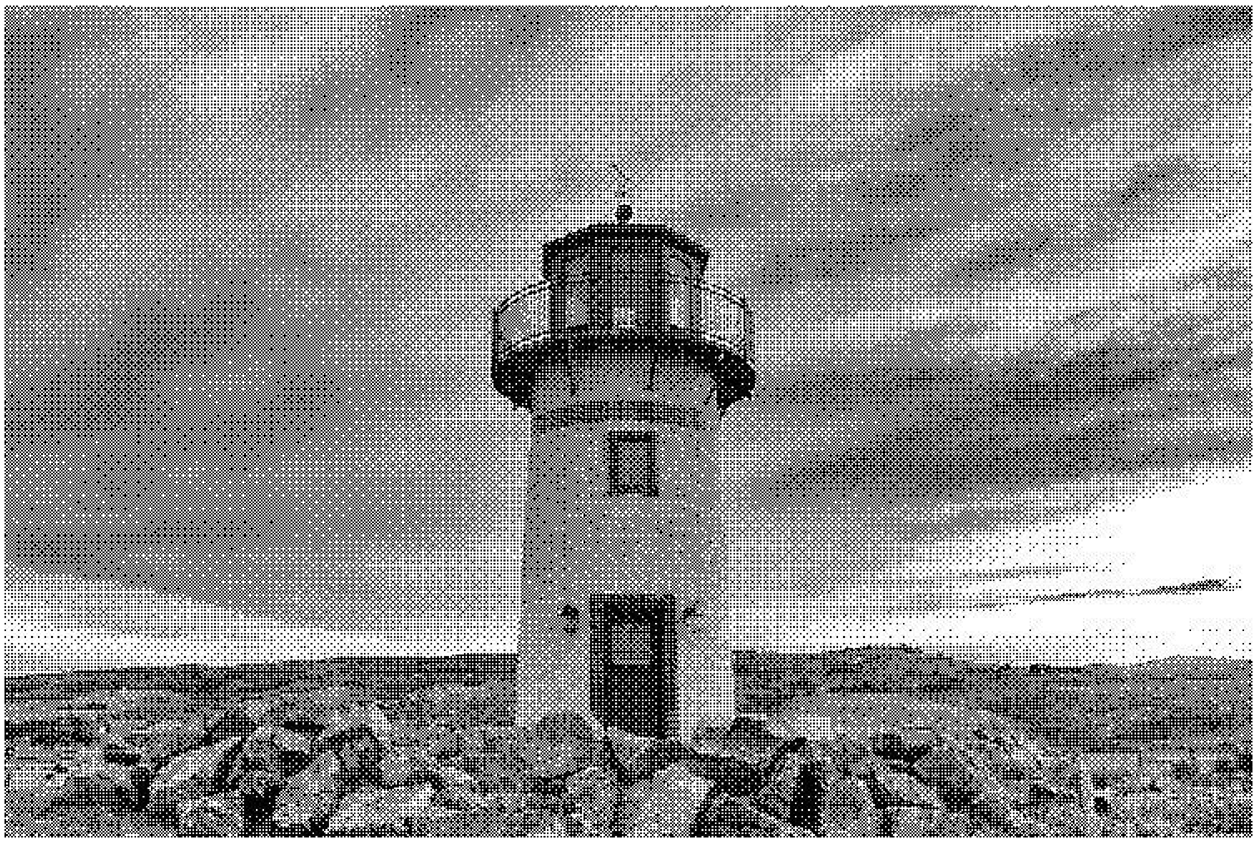


Figure 23: Ordered Dithering using 8*8 Threshold matrix

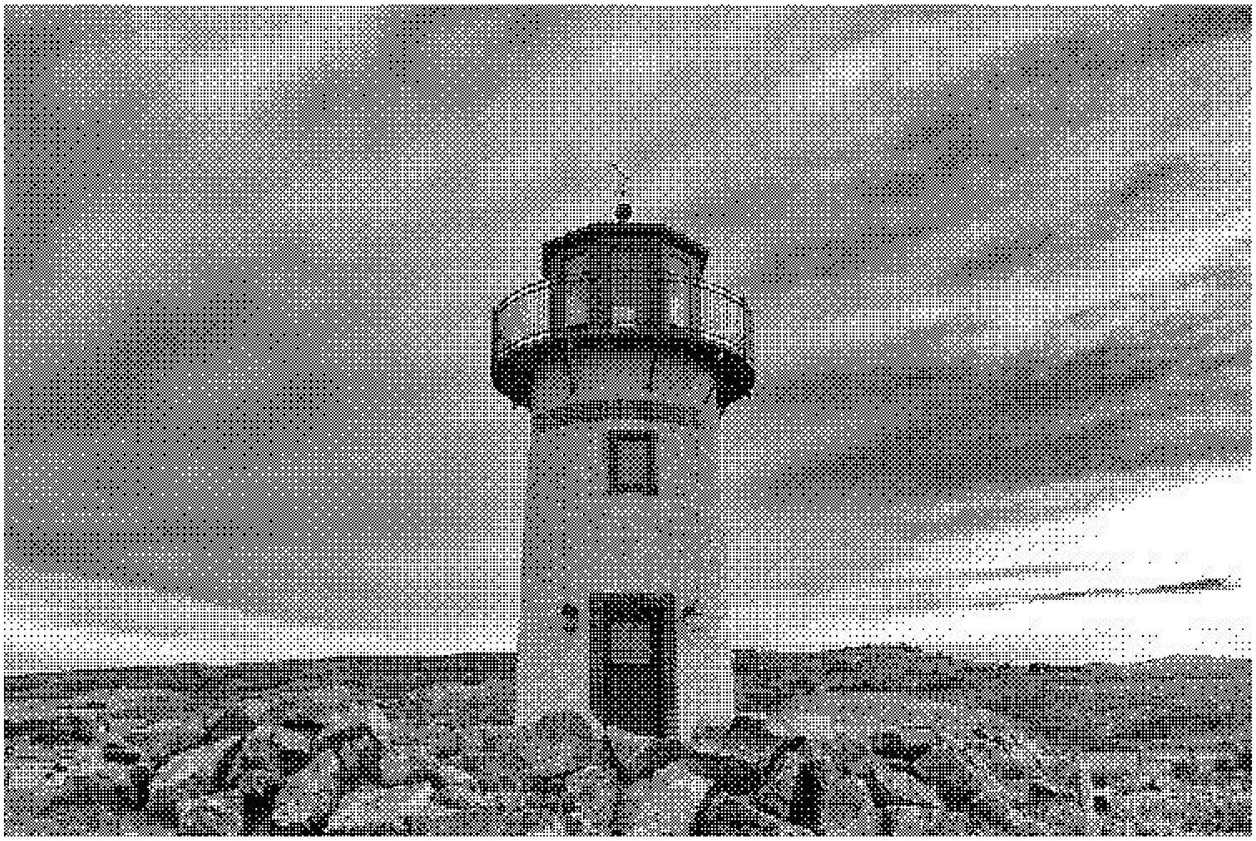


Figure 24: Ordered Dithering using 32*32 Threshold matrix

Observations:

A. The main idea behind dithering is to generate black and white pixels in a specific pattern so that the human visual system perceives as a continuous gray level values. This is because the human visual system perceives the image by averaging the patch of region instead of each pixel. This creates an illusion.
B. In the previous observation we discussed the correlation between input pixels and quantisation error. This effect is reduced considerably in ordered dithering because we are generating variable thresholds.
C. Since we repeatedly use the threshold matrix across the image, we can see “**tiled**” patterns in the image.
D. False contours are considerably reduced compared to Random thresholding.
E. The fidelity of the halftone image is better with 8*8 and 32*32 compared to 2*2. As we can observe, 2*2 gives few gray

scale impression than 8*8, while the best halftone image is obtained using 32*32 threshold matrix.

F. Also in 2*2 we can see **grid like patterns** which are not evident in 8*8 and 32*32 because of the reason stated in C.
G. This method retains very good rendition over a small patch of the image.
H. The quantization error effect has been reduced compared to Random thresholding.
I. But the disadvantage of dithering is that square grid patterns are visible in the halftone image.

d) Error Diffusion: The error diffusion method is applied on the lighthouse image using Floyd-Steinberg's error diffusion filter, Jarvis, Judice and Ninke error diffusion filter and Stucki's error diffusion filter. The results are compared and shown in Figure 25, 26 and 27 respectively.

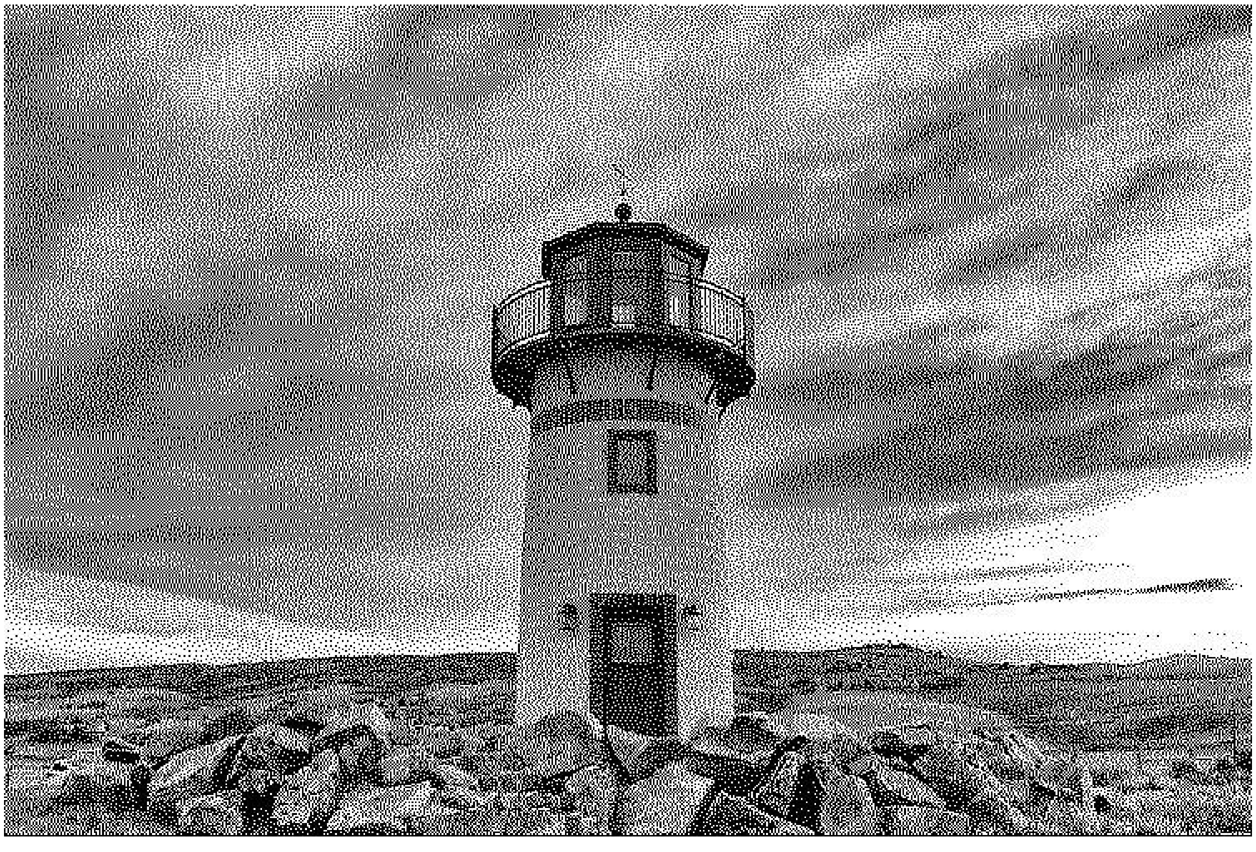


Figure 25: Output for Floyd-Steinberg diffusion with threshold 128 using serpentine pattern

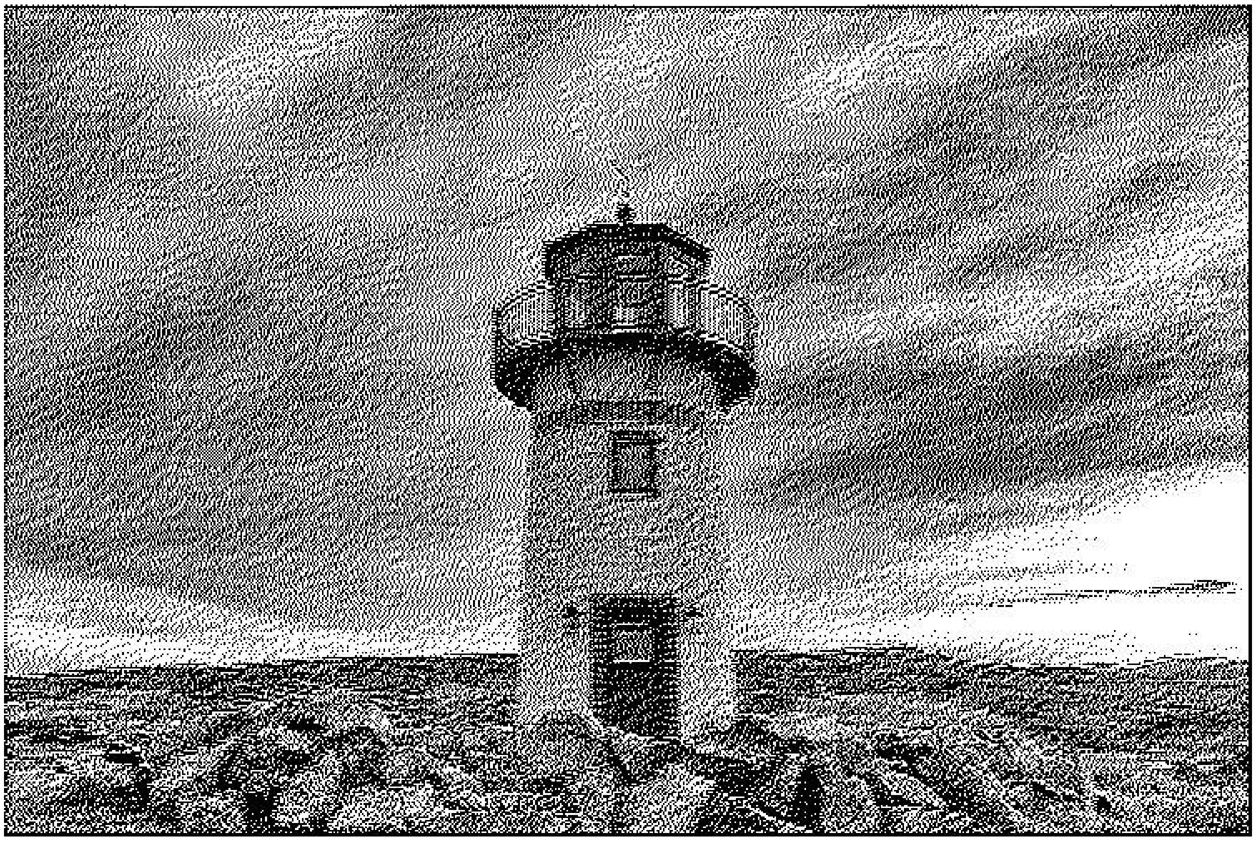


Figure 26: Output for JJN diffusion filter with threshold 128 using serpentine pattern

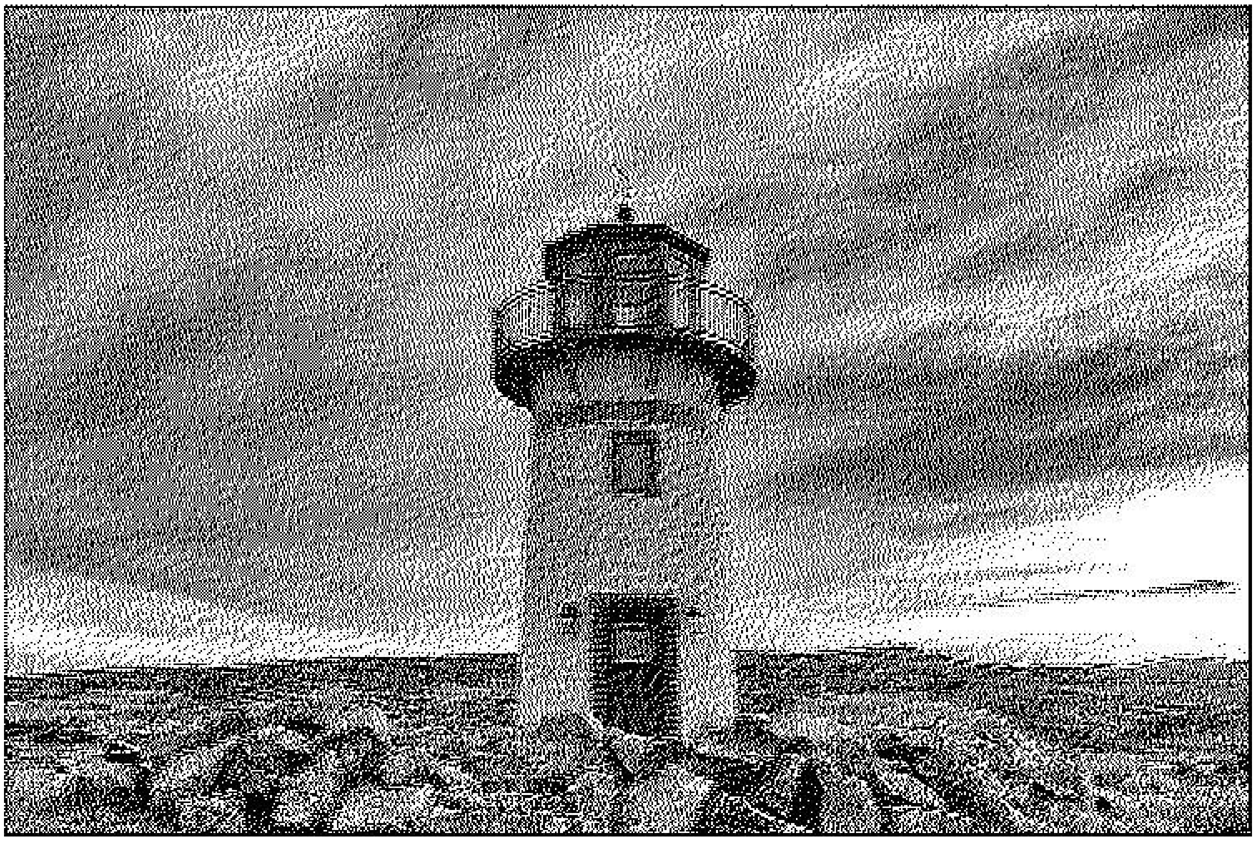


Figure 27: Output for Stucki diffusion filter with threshold 128 using serpentine pattern

Observations:

A. The quantization error effect has been nullified to a greater extent because the quantization error has been propagated to future pixels.
B. Serpentine parsing gives better rendition of image than Raster parsing.
C. The result of the Error diffusion algorithms are impressive compared to the best result of ordered dithering.
D. All the algorithms are tested with different threshold values and there is a contrast variance if we reduce the threshold. So, 128 would be the nominal choice for the threshold value.
E. Compared to Dithering, Error diffusion gives better contrast performance.
F. But the downside of Error diffusion is we observe few streaking artifacts which are commonly called as **“worm patterns**”. This can be seen in the Floyd output near the clouds.

Comparison between the output of three Error diffusion filters:

A. All the three methods perform considerably well compared to dithering and all three methods are fast.
B. However, Floyd produces little false contours (can be observed near clouds) and grid patterns. But this can't be seen in both JJN and Stucki.
C. Performance wise output of Stucki is better than JJN which is better than Floyd ($Stucki > JJN > Floyd$).
D. JJN is comparatively slower than Floyd because it used 5*5 neighbors. But Stucki is faster than JJN.
E. The Root mean squared error for Stucki (98.34) is less compared to JJN (103.49) and Floyd (108.61).

Which method do you prefer (Dithering vs Error Diffusion)? Why?

- I'd prefer **Stucki's error diffusion** method over other error diffusion algorithms and Dithering because it is fast with least mean squared error and produces better performance without false contours and gives the gray level impression perfectly.

Describe your idea to improve the results.

- We can effectively improve the result by incorporating better parsing methods than Serpentine and Raster. The best choice would be the **Hilbert curve.** This helps to propagate the quantization error in a unique pattern and thereby avoiding "worm patterns"
- Also, we can use **Variable Coefficients Error diffusion** instead of fixed coefficients. This will certainly give better results because the diffusion filter values vary according to the input pixel value.

2) Color Halftoning with Error Diffusion:

a) **Separable Error Diffusion:** Separable error diffusion is performed on Rose image shown in Figure 28.

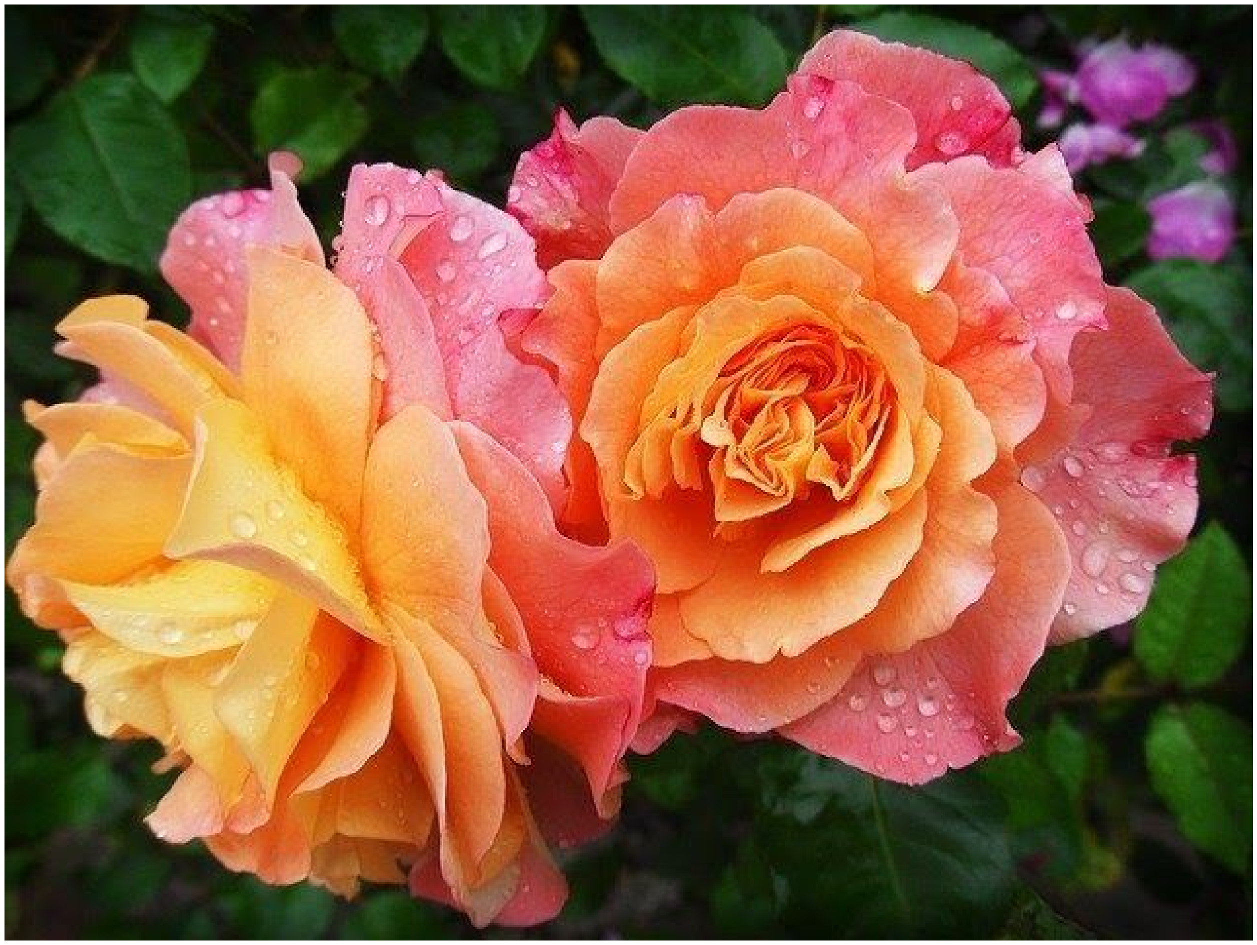


Figure 28: Input Rose Image

The result of Separable error diffusion is shown in Figure 29.

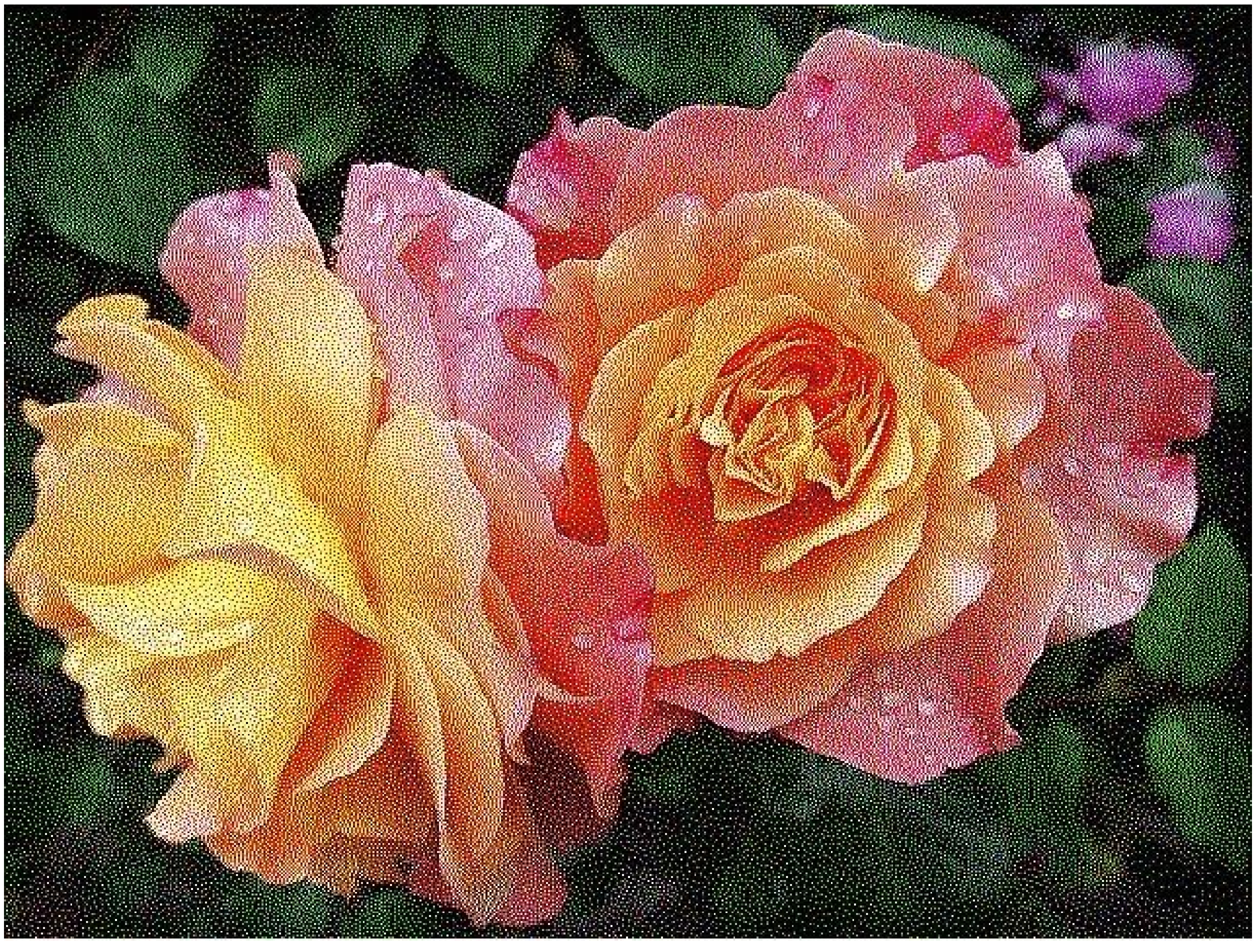


Figure 29: Color Halftone image using Separable Error Diffusion

Observations:

A. We can see some visual artifacts in the resulting image (corner black patches are a bit noisy because of green dots). This is due to the variation in brightness of the dots (Discussed in Shortcomings).
B. The placement pattern of the colored dots is almost unnoticeable.
C. Also, we can observe that the local average is the desired color.
D. However, the colors used don't reduce the noticeability of the patterns in Separable error diffusion.

Shortcoming of Separable error diffusion:

A. This method is the generalization of monochrome algorithm so it tends to overlook the fact that colored dots are not equally bright. As a result we can see colored patches.

B. Overlooks the characteristics of the human visual system (which is more sensitive to changes in brightness than chrominance) and doesn't consider variation of brightness into account.

b) Minimal Brightness Variation Quadruple (MBVQ): Figure 30 shows the result of MBVQ on Rose image.

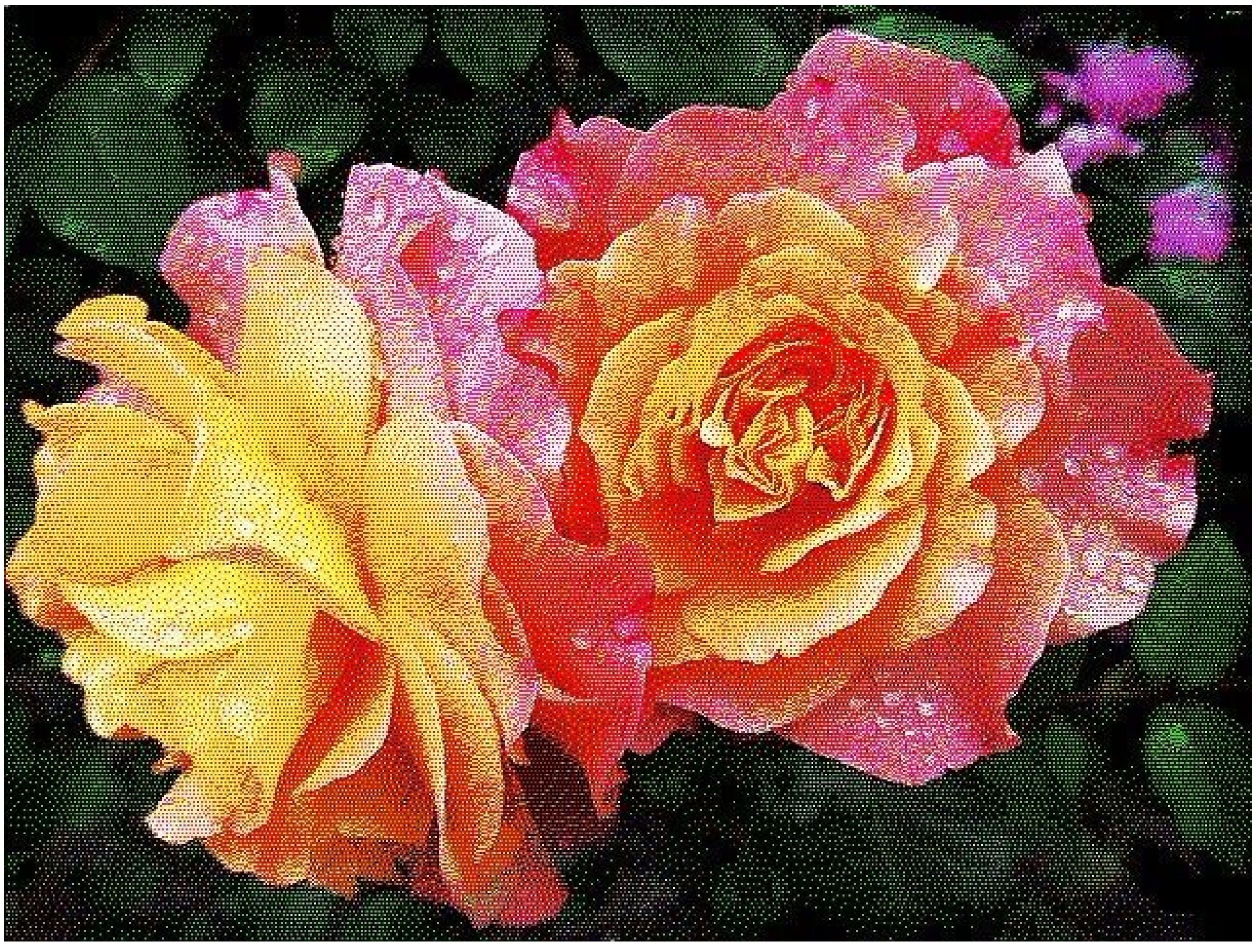


Figure 30: Color Halftone image using MBVQ

<u>Observations (Comparison with Separable error diffusion):</u>

A. As compared to Separable error diffusion, the result of MBVQ is improved a lot. We can see the clarity of the image in petals, especially the water droplets.

B. The purple flowers in the background are more clear compared to Separable error diffusion.
C. Also, the black patches in the corner are not distorted with green dots like it happened in separable error diffusion.
D. Clear distinction between the colors can be observed in MBVQ compared to Separable error diffusion.

Key Ideas of MBVQ approach:

(Explained in detail in Abstract and Motivation Section)

A. In separable error diffusion, we quantize a pixel to one of the 8 basic colors. But, any color can be rendered efficiently using only 4 colors. The main idea behind the rise of] MBVQ algorithm is "When presented with high frequency patterns, the human visual system applies a low pass filter and perceives only their average".
B. The main agenda of MBVQ is to preserve the average color of the pixel along with reducing the brightness variation. This reduces the number of required halftones to 4. The region where the pixel belongs is called Minimal Brightness Variation Quadruple and it's one among 6 tetrahedrons-**RGBK, WCMY, MYGC, RGMY, RGBM, CMGB.**
C. By Exploiting the characteristics of the human visual system, MBVQ considers the variation in brightness and reduces the noticeability of the color patterns is considerably compared to Separable error diffusion.

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://justin-liang.com/tutorials/canny/
3) https://github.com/pdollar/edges
4) https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900
5) https://ieeexplore.ieee.org/document/1407718

6) https://engineering.purdue.edu/~bouman/ece637/notes/pdf/Halftoning.pdf.
7) https://engineering.purdue.edu/~bouman/grad-labs/Image-Halftoning/pdf/lab.pdf
8) Doron Shaked, Nur Arad, Andrew E. Fitzhugh, and Irwin E. Sobel "Color diffusion: error diffusion for color halftones", Proc. SPIE 3648, Color Imaging: Device-Independent Color, Color Hardcopy, and Graphic Arts IV, (22 December 1998)