EE569 Digital Image processing: Homework #2

Name: Nazim N Shaikh
USC Id: 8711456229
Email Id: nshaikh@usc.edu
Date: 02/12/2019

Problem 1: Edge Detection

I. Abstract and Motivation

Edge detection, as the name suggests, is a process of identifying object edges in an image. An edge in an image is a significant local change in the image intensity, usually associated with a discontinuity in the image intensity. These discontinuities in image brightness usually result from
Surface color discontinuity, depth discontinuity, changes in material properties or variation in scene illumination
The purpose of edge detection is to capture significant features of object in a scene such as corners, lines and curves while significantly reducing the amount of data to be processed. It helps in reducing unnecessary information in the image while preserving important structural properties of the image. It is highly useful in object recognition, image pattern recognition, image segmentation, etc.

II. Approach and Procedures

There are many different edge detection methods, the majority of which can be grouped into two categories: Gradient based and Laplacian. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. The Laplacian method searches for zero crossings in the second derivative of the image. Here, we will discuss three commonly used edge detection methods.

a) Sobel Edge Detector

In this, Sobel filter is used for edge detection. It works by calculating the gradient of image intensity at each pixel within the image. It finds the direction of the largest increase from light to dark and the rate of change in that direction.
The Sobel filter uses two 3 x 3 kernels. One for changes in the horizontal direction, and one for changes in the vertical direction as shown below

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

Gx

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

Gy

3-by-3 Sobel mask

Steps involved:

1) Apply the masks separately to the input image, to produce separate measurements of the gradient component in each orientation (call these g_x and g_y).

2) Combine them to find the absolute magnitude of the gradient at each point and the orientation of that gradient as given below

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

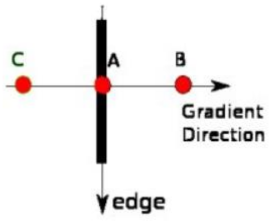
$$\theta = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

3) Normalize the gradient magnitude between (0~255).
4) The output of step 3 is a probability edge map, so we apply thresholding to get final binary edge map.

b) <u>Canny Edge Detector</u>
It is used to detect edges while at the same time suppress noise. It is a multi-stage algorithm implemented as below:

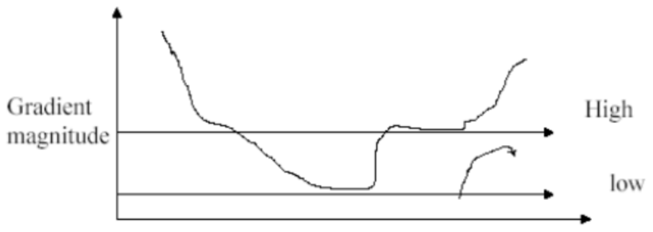
1. Apply Derivative of Gaussian filter to smooth the image in order to remove the noise
2. Find magnitude and orientation gradient of the image by applying 2 Sobel masks.
3. Apply **Non-maximum suppression (NMS)** to remove any unwanted pixels which may not constitute the edge as well as to give out thing edges.

   For this, at every pixel location, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. If yes, the gradient magnitude of that pixel is preserved, otherwise it is suppressed to 0. Check the image below:



   Pixel A is on the edge (in vertical direction). Gradient direction is normal to the edge. Pixel B and C are in gradient directions. So, pixel A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, else, it is suppressed to 0.

4. Apply **Double thresholding** to filter out edge pixels with a weak gradient value and preserve edge pixels with a high gradient value.



   This is done by selecting high and low threshold values. If an edge pixel's gradient value is higher than the high threshold, it is marked as a strong edge pixel, else, it is suppressed. However, if an edge pixel's gradient value is smaller than the high threshold and larger than the low threshold, it is marked as a strong edge pixel only if it is connected to a pixel that is above upper threshold

c) Structured Edge Detection

Traditional Edge Detectors are not able to detect visually silent edges such as texture edges, illusory contours, etc. These features correspond to variety of visual information. Hence, the need for state-of-the-art edge detectors which can use multiple features such as brightness, color, texture, depth, etc. for edge detection. This brings to Structured Edge Detection.

It is a machine learning based technique, which takes an image patch (possible edge structure) as input and computes the likelihood that the patch contains an edge. In this method, random forest algorithm is used to capture structured information. A random forest is trained with structured labels. These structured labels are used to determine random subset of features to be selected at each candidate split while going down the decision trees. Each forest then predicts a patch of edge pixel labels. Prediction of all the forests are then aggregated across the image to compute final edge map.

d) Performance Evaluation

To evaluate the performance of an edge map, we need to identify the error. All pixels in an edge map belong to one of the following four classes:
(1) True positive: Edge pixels in the edge map coincide with edge pixels in the ground truth. These are edge pixels the algorithm successfully identifies.
(2) True negative: Non-edge pixels in the edge map coincide with non-edge pixels in the ground truth. These are non-edge pixels the algorithm successfully identifies.
(3) False positive: Edge pixels in the edge map correspond to the non-edge pixels in the ground truth. These are fake edge pixels the algorithm wrongly identifies.
(4) False negative: Non-edge pixels in the edge map correspond to the true edge pixels in the ground truth. These are edge pixels the algorithm misses.
Clearly, pixels in (1) and (2) are correct ones while those in (3) and (4) are error pixels of two different types to be evaluated. The performance of an edge detection algorithm can be measured using the F measure, which is a function of the precision and the recall.

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

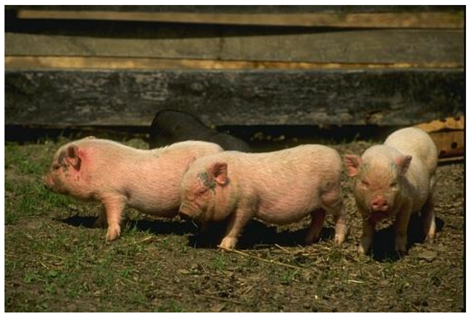
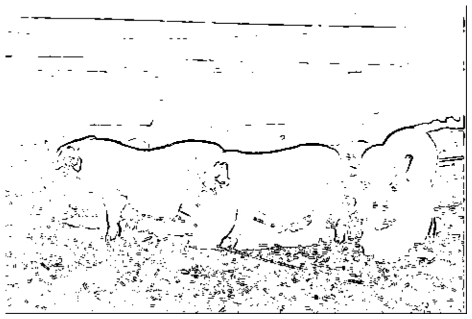
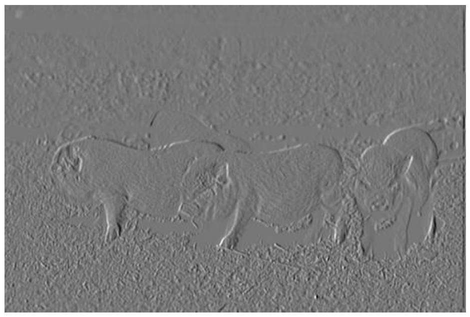
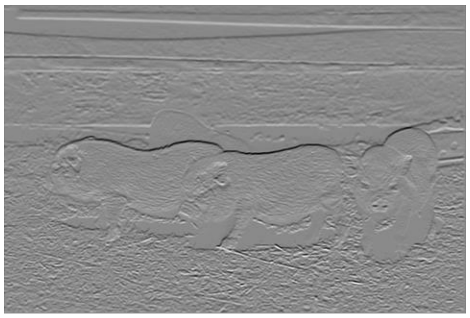
$$\text{Recall} : R = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}}$$

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

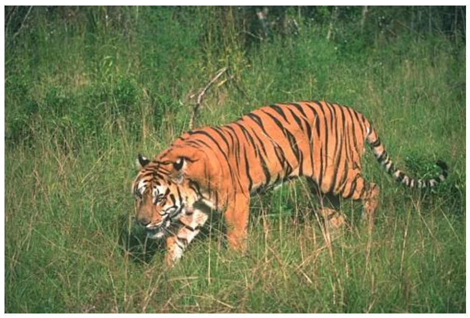
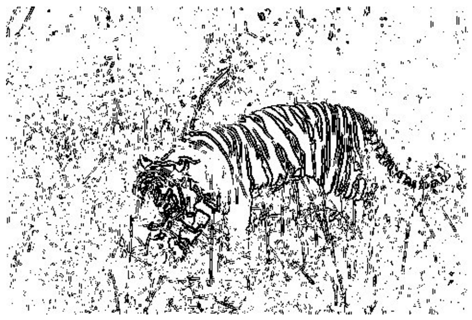
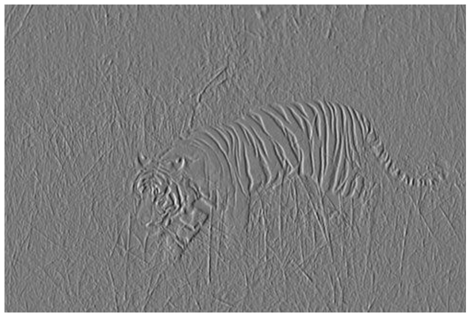
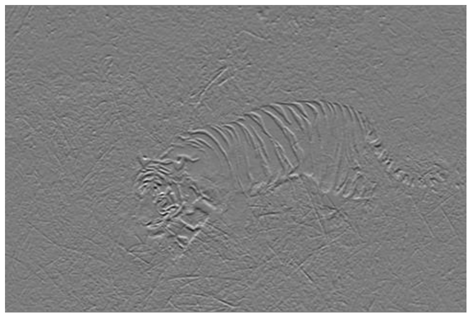
III. Experimental Results

1) Sobel

o/p for pig:

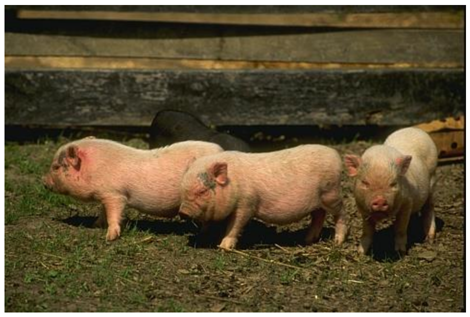
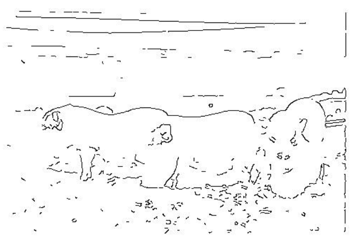
 *Figure 1 Original Image*	 *Figure 2 Sobel Edge Map*
 *Figure 3 x-gradient map*	 *Figure 4 y-gradient map*

o/p for tiger:

 *Figure 5 Original Image*	 *Figure 6 Sobel Edge Map*
 *Figure 7 x-gradient map*	 *Figure 8 y-gradient map*

2) <u>Canny</u>

o/p for pig:

 *Figure 9 Original Image*	 *Figure 10 Canny Edge Map*

o/p for tiger:

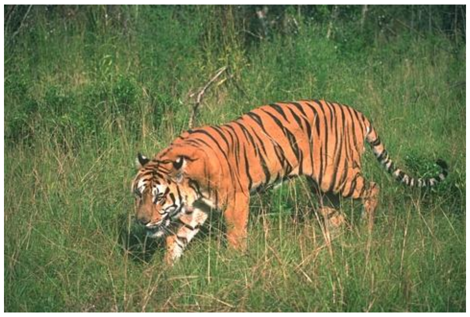


Figure 11 Original Image

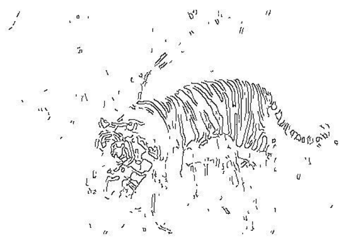


Figure 12 Canny Edge map

3) <u>Structure Edge</u>

o/p for tiger:

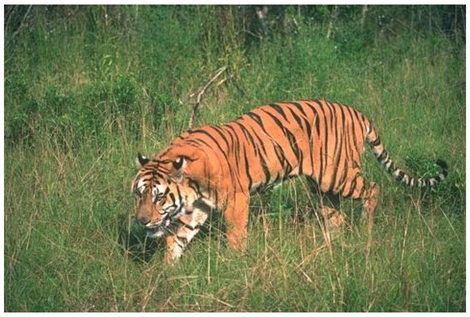


Figure 13 Original Image

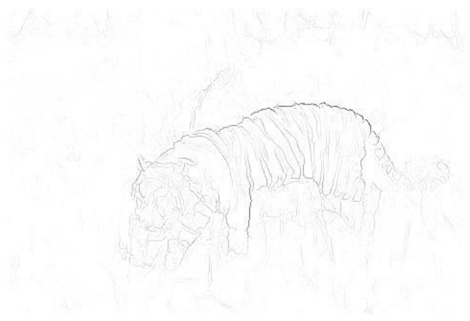


Figure 14 Probability Edge map

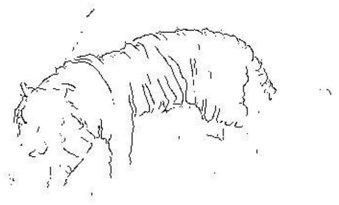


Figure 15 Binary Edge Map

o/p for pig:

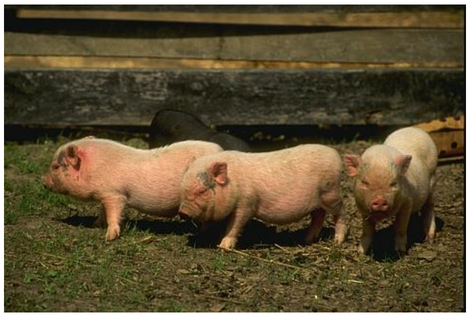


Figure 16 Original Image

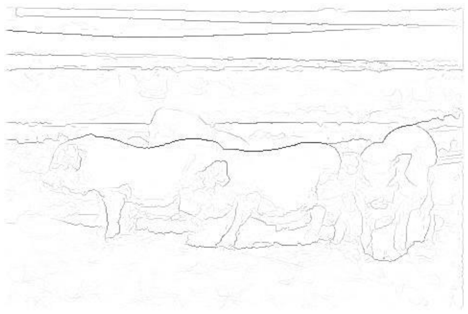


Figure 17 Probability Edge Map

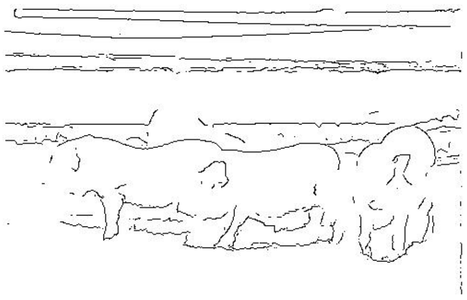


Figure 18 Binary Edge Map

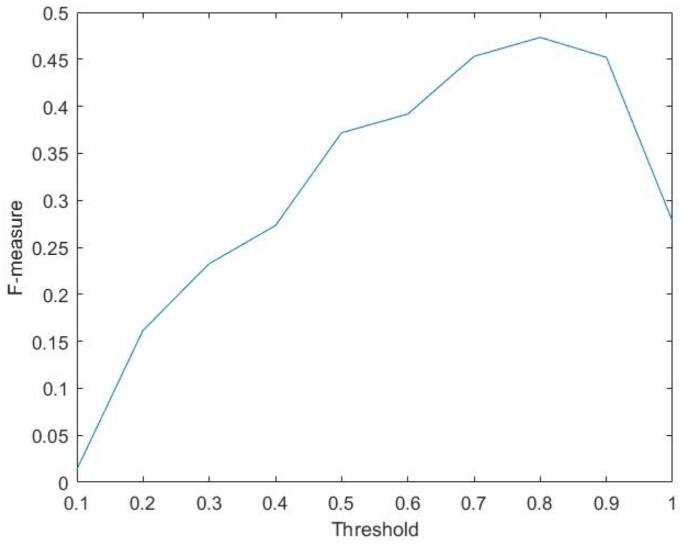
4) Performance Evaluation
   a) Structured Edge:

Pig:

Ground Truths	Mean Recall	Mean Precision	Mean F-Measure
Ground Truth 1	0.0097	0.1242	0.0179
Ground Truth 2	.1551	.2557	0.1931
Ground Truth 3	.2099	.1781	0.1890
Ground Truth 4	.2203	.1799	.1981
Ground Truth 5	.4430	.2184	.2925

F-Plot:

Best F: 0.4733

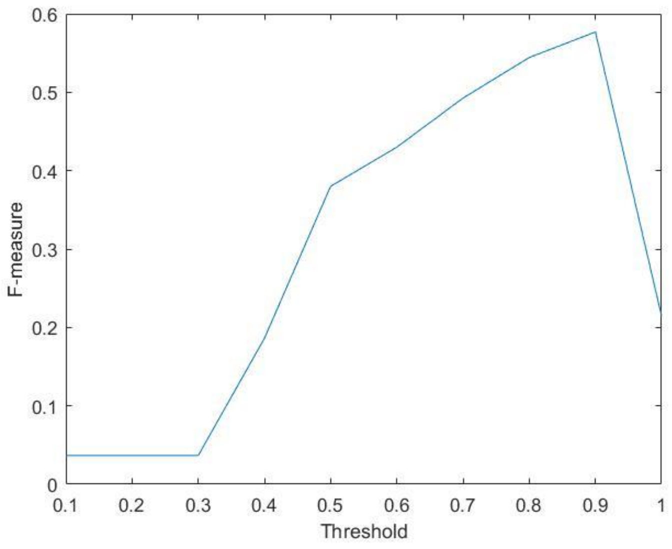


Tiger:

Ground Truths	Mean Recall	Mean Precision	Mean F-Measure
Ground Truth 1	0	0	0
Ground Truth 2	0	0	0
Ground Truth 3	0	0	0
Ground Truth 4	0.1366	0.4789	0.1366
Ground Truth 5	0.1401	0.1077	0.1406

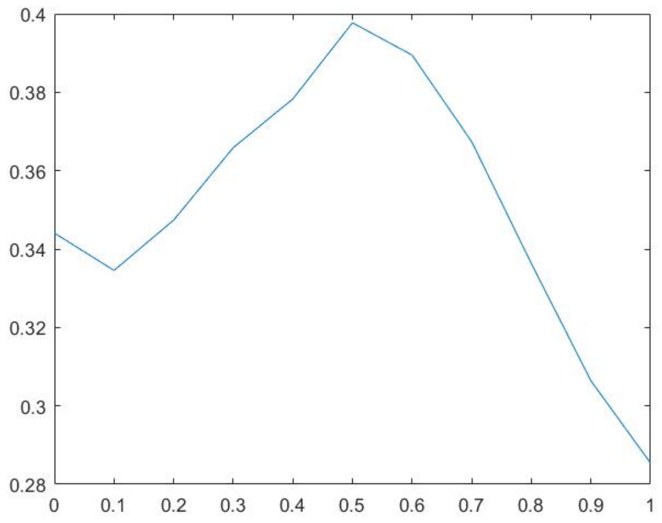
F -plot:

Best-F: 0.5769



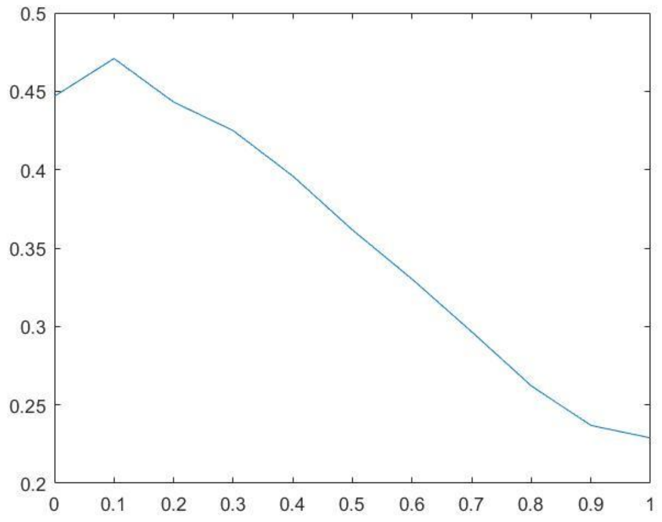
b) Canny

Pig F plot: (x-axis -> Threshold and y-axis-> F-measure)
Best F-measure: 0.3969



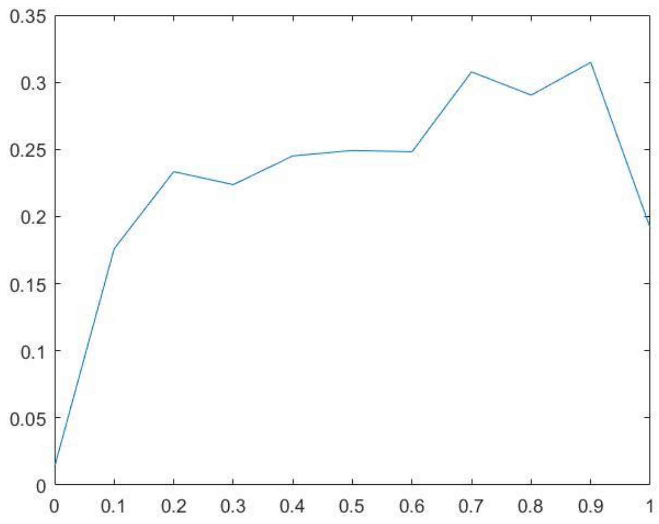
Tiger F plot: (x-axis -> Threshold and y-axis-> F-measure)

Best F-measure: 0.4010



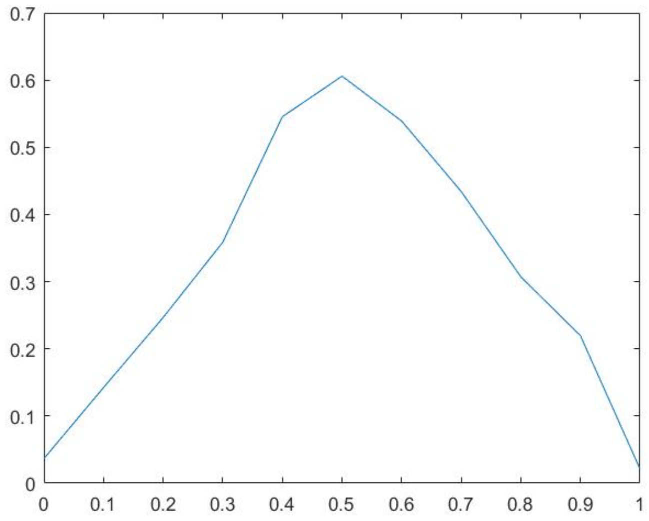
c) Sobel
Pig F plot: (x-axis -> Threshold and y-axis-> F-measure)

Best F-measure: 0.3149



Tiger F plot: (x-axis -> Threshold and y-axis-> F-measure)

Best F-measure: 0.6059



IV. Discussion

1) Sobel

➔ The output of sobel edge detector for pig and tiger is shown from fig 1 – 8
The corresponding x-gradient map and y-gradient map is show.
From the output, we see that, Sobel gives out adequate edges, with not much noise.
The gaussian filter helps reducing noise by smoothing, but this also affects accuracy for edge detection.

Although edges detected are not thin, present with lot of textures, Sobel is faster in computation compared to canny

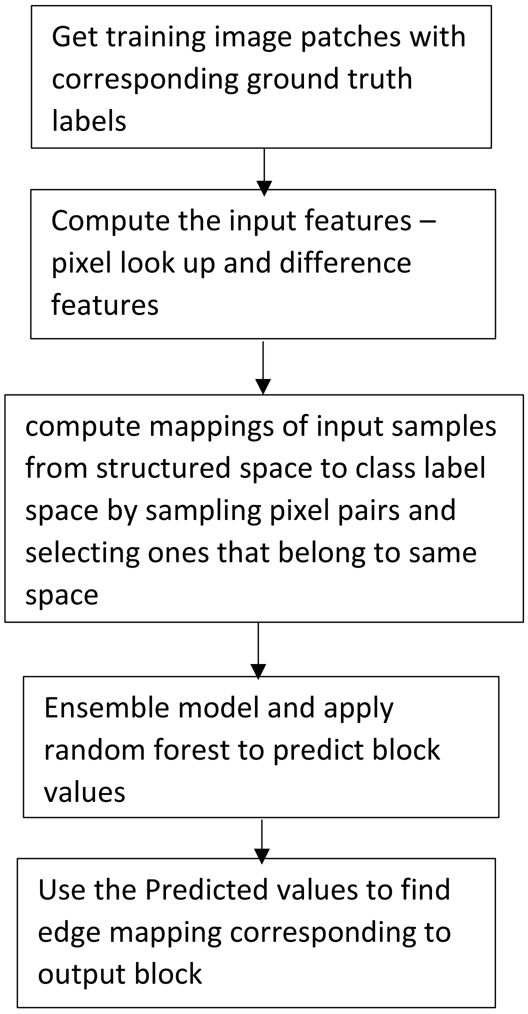
2) <u>Canny</u>

➔ The output of canny edge for pig and tiger is shown from fig 9 – 10. Compared to Sobel we observe that canny output has more fine and connected edges. This is due to non-maximum suppression and double thresholding which helps in thinning and selecting strong edges. However, there are still lot of textures present. To overcome this structured edge detection is applied

3) <u>Structured Edge</u>

➔ It gives the best output. As seen from fig 13 -18, only fine edges are being detected. Other visually salient features such as textures are fairly removed.

➔ The structured edge algorithm has been briefly explained in the approach section. Here we look at the flowchart:



Get training image patches with corresponding ground truth labels

Compute the input features – pixel look up and difference features

compute mappings of input samples from structured space to class label space by sampling pixel pairs and selecting ones that belong to same space

Ensemble model and apply random forest to predict block values

Use the Predicted values to find edge mapping corresponding to output block

➔ Random Forest Classifier and Decision Trees: It is an ensemble machine learning algorithm. It creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.

Decision tree literally has tree-chart structure. Each node represents 'test' feature, branch represents outcome of test and each leaf node represents class label.
Construction of decision tree:

- Start with root node which contains all samples. (right and wrong)
- Split them best on best attribute and assign corresponding labels.
- Repeat the steps until decision for each feature has been made.
-

The decision tree training starts with bootstrap aggregating or bagging.

- For a given training set and response, random sample of training set is selected repeatedly
- For every selected samples, a decision tree is fit
- Train decision tree on those selected samples

The final predictions are then made by taking majority vote.

In Random Forest, at each split in the training process explained, random subset of features is selected. This is done to avoid correlation between the trees.

Advantages:

- Good Model Performance as bootstrapping decreases variance without increasing bias by de-correlating the tree by showing different training sets.
- Not sensitive to feature normalization
- Low computation cost
- it can be used for both classification and regression problems,
- enough diversity of trees avoid overfitting

Disadvantages:

- For data with categorical variables with more classes, information gain can be biased
- For many linked outcomes, calculations can be complex.

4) <u>Performance Evaluation</u>

- Precision and recall for each ground truth for structured edge detection is shown in the experimental section along F-plot and best-F values for various edge detectors.
- We see that structured edge gives out the best f-measure in case of all the detectors.
- From Image perspective, pig f-measure should be better than tiger as I believe this is due to less texture features present in pig as compared to tiger and as well as more color variation compared to tiger, thereby helping in more efficient edge detection.

- Best F measure is when precision and recall are equal. One can make the precision higher by decreasing the threshold in deriving the binary edge map. However, this will result in a lower recall. Generally, we need to consider both precision and recall at the same time and a metric called the F measure is developed for this purpose. A higher F measure implies a better edge detector.

Problem 2: Digital Halftoning

I. Abstract and Motivation

Digital Halftoning is process of representing original digital image with black and white pixels only. Most Image rendering devices are not able to reproduce different shadows of gray. Halftoning helps in creating the illusion of continuous tone output for such devices.

It is mainly used in image display devices capable of reproducing two-level outputs such as scientific workstations, laser printers, and digital typesetters. Effective digital halftoning can also substantially improve the quality of rendered images at minimal cost.

II. Approach and Procedures

There are various approaches to digital halftoning. Here we will discuss digital halftoning for gray-scale as well as color images and look it different methods to achieve the same.

1) Direct Binarization

It is the simplest way to convert image from grayscale to binary. It approaches attempt to minimize weighted least squares criterion directly. There are 2 ways to achieve this.

a) Constant Thresholding

Also known as fixed-quantization, it compares the input pixel value with a fixed gray value. If it is higher, we quantize it to 1, if it is less, we quantize it to 0. The algorithm can be described as follows:

1. Compare each pixel value with a fixed value T (usually middle gray value). If it is less than T, map it to 0; otherwise, map it to 255.

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

Where T = 127

b) Random Thresholding

In order to avoid information loss, we have another method called 'random' threshold, in which pixel is compared against random threshold. The algorithm can be described as below:

2. For each pixel, generate a random number in the range 0 – 255. Let's call it *rand(i,j)*
3. Compare the pixel value with *rand(i,j).* If it is less, map it to 0; otherwise, map it to 255.

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

Although, this method adds more details to the image, it results in a spatial accumulation of quantization error.

2) <u>Dithering</u>

The dithering technique was invented to overcome above disadvantage of direct binarization. In this, the quantization error is removed by adding uniformly distributed white noise before quantization of image. The amount of noise to be added is determined by the order of pixel. Dithering uses a "dither matrix". For every pixel in the image the value of the pattern at the corresponding location is used as a threshold.

The implementation can be described as follows:

1. Start with Initial Index Matrix as below:

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

   Here, 0 indicates pixel most likely to be turned on and 3 is the least likely one.

2. As per the required pattern size, generate higher order index matrix recursively using below formula:

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

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

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

   Where N^2 represents the number of pixels in the matrix.

4. The N × N Threshold matrix is then “tiled” over the image using periodic replication.

   T(i modN, j modN)

5. The ordered dither algorithm is then applied via thresholding as below to get final image

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

3) <u>Error Diffusion</u>

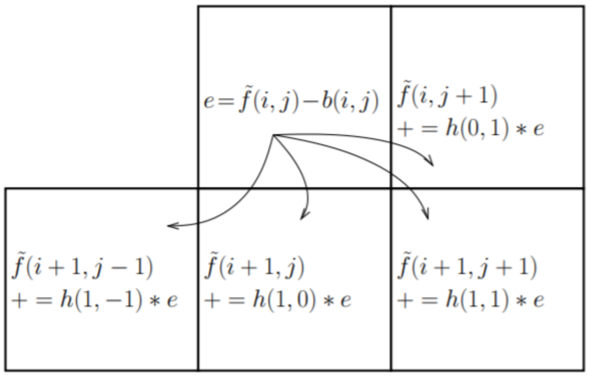
Error diffusion is another technique used for generating digital halftoned images. Error diffusion sequentially traverses each pixel of the source image. Each pixel is compared to a threshold. If the pixel value is higher than the threshold, a 255 is outputted; otherwise, a 0 is outputted. The error - the difference between the input pixel value and the output value - is dispersed to nearby neighbors. This can be interpreted as a way of keeping the local average intensity of the printed image as close to that of the original image as possible.

The detailed procedure is described as below:

1. Initialize $\tilde{f}(\mathrm{i,j}) \leftarrow \mathrm{f(i, j)}$
2. For each pixel in the image
    i. Pull compute binary output as below:

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

    ii. Compute pixel's error and diffuse the error forward using below scheme



    iii. Display final binary image

'h' in the above scheme is the diffusion weight of the neighboring pixels to be affected by the quantization error of a pixel at a given location of the input image. Which pixels to be affected is given by any of below error diffusion matrices:

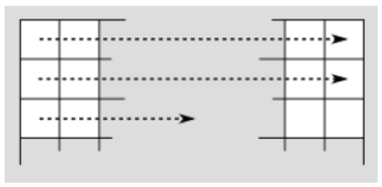
- **Floyd-Steinberg** $\frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$

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

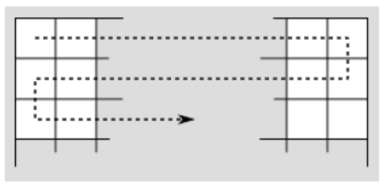
- **Stucki** $\frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$

The most commonly used is Floyd-Steinberg. As seen from the filter matrices, error tends to be diffusing to the right boundary of the image. This makes scanning order very crucial to result. Various scanning method are shown below, however for this homework, serpentine scanning is employed as it handles diffusion of error on the right boundary. While going from left to right original matrix is used and its mirrored reflection is used when the direction is from right to left.

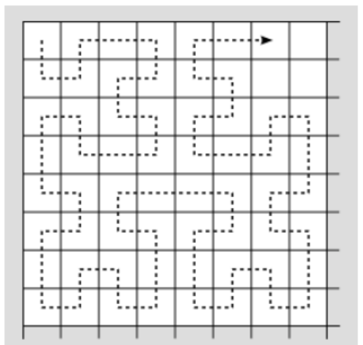
- **Raster parsing**



- **Serpentine parsing**



- **Hilbert curve**



Error diffusion is a neighborhood operation since it operates not only on the input pixel, but also its neighbors. Generally, neighborhood operations produce higher quality results than point operations. Error diffusion, when compared to dithering, does not generate those artifacts introduced by fix thresholding matrices. However, since error diffusion requires neighborhood operations, it is very computationally intensive.

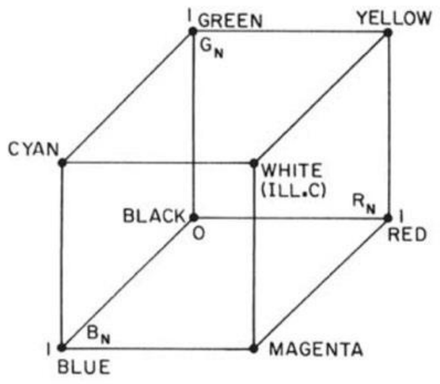
4) Color halftoning

It is the process of transforming continuous-tone color images into images with a limited number of colors. The importance of this process arises from the fact that many color imaging systems use output devices such as color printers and low-bit depth displays that are bilevel or multilevel with a few levels. Here we discuss 2 methods to achieve color halftoning

a) Separable Error Diffusion:

The original grayscale error diffusion is extended to color images in which quantization errors are diffused to future pixels separately on each channel. The final color is rendered using one of the 8 basic colors located at the vertices of the cube as shown below

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)



The algorithm is implemented as below:

1. Convert the RGB into CMY color space
2. Separate the CMY image into individual channels
3. Apply the Floyd-Steinberg error diffusion algorithm to quantize each channel separately.

b) MBVQ-based Error Diffusion:

MBVQ – Minimum Brightness Variation Quadruples. The method proceeds by separating the RGB color space into minimum brightness variation quadruples (MBVQs)

In this method, every input color is rendered using one of six complementary quadruples: RGBK, WCMY, MYGC, RGMY, RGBM, or CMGB, each with minimal brightness variation.

The algorithm is implemented as follows:

For each pixel in the image,

1. Determine MBVQ(RGB(i,j)), limiting the pixel to one quadrant.

2. Find the vertex belonging to MBVQ which is closest to RGB(i,j), again limiting the pixel to the closest vertex.
3. Compute the quantization error as RGB(i,j) – v
4. Distribute the error to future pixels
5. Display final image

III. Experimental Results

1) Dithering:

a) Thresholding

b)

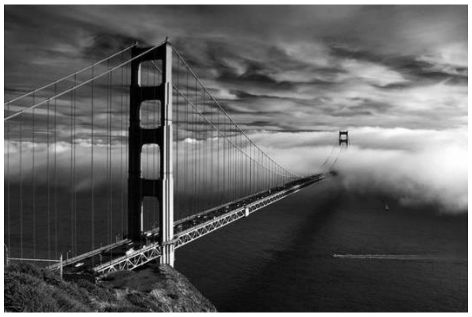


Figure 19 Original Image

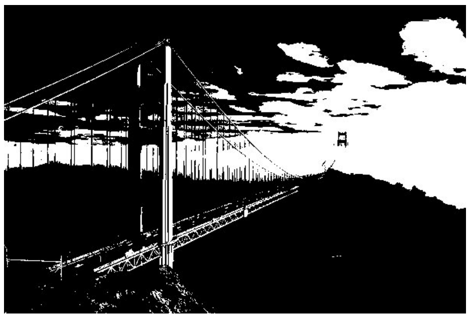


Figure 20 Fixed Thresholding

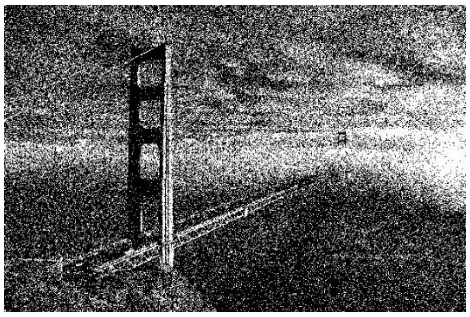


Figure 21 Random Thresholding

c) Dithering Matrix:

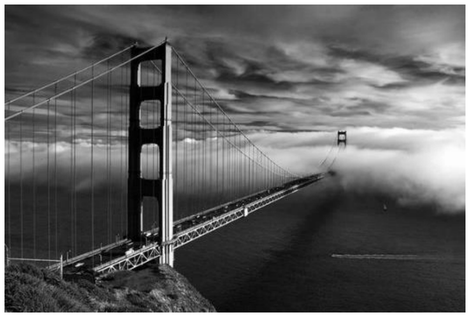


Figure 22 Original Image

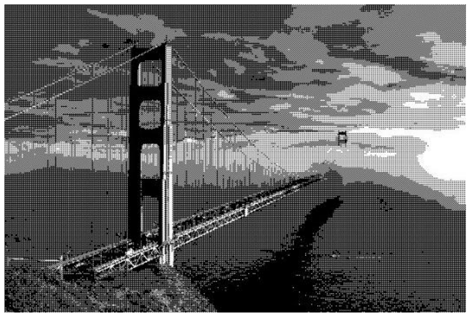


Figure 23 Dithering o/p with n = 2

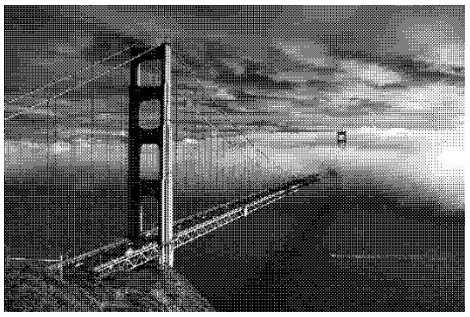


Figure 24 Dithering o/p with n = 4

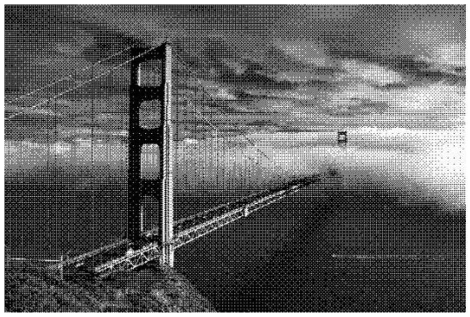


Figure 25 Dithering o/p with n = 8

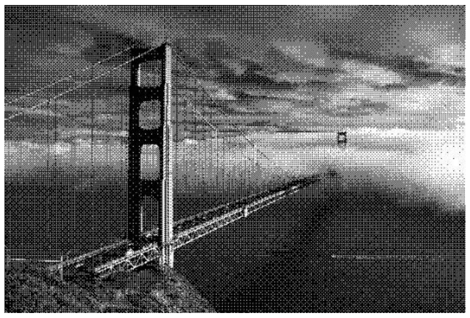
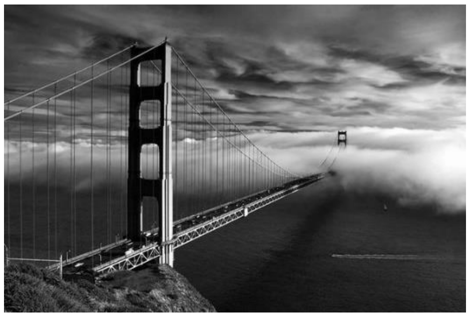
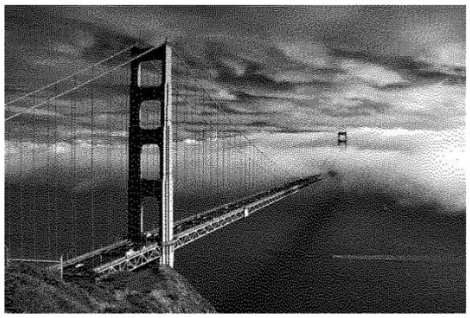
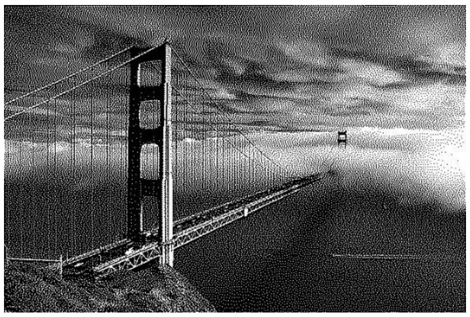
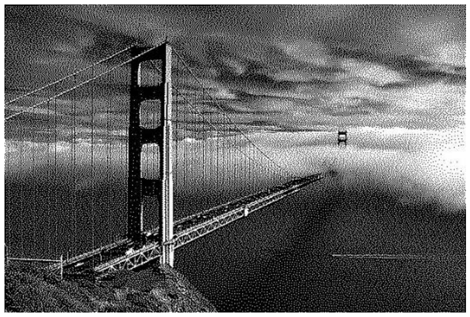


Figure 26 Dithering o/p with n = 32

d) Error Diffusion:

 Figure 27 Original Image	 Figure 28 Floyd Steinberg Error Diffusion
 Figure 29 JNN Error Diffusion	 Figure 30 Stucki Error Diffusion

2) Color Halftoning with Error Diffusion:

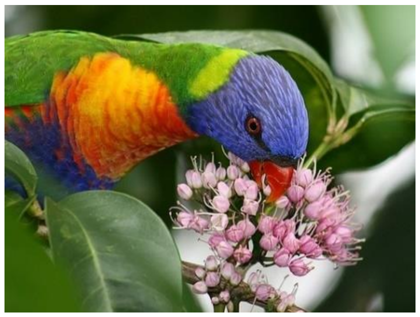


Figure 31 Original Image

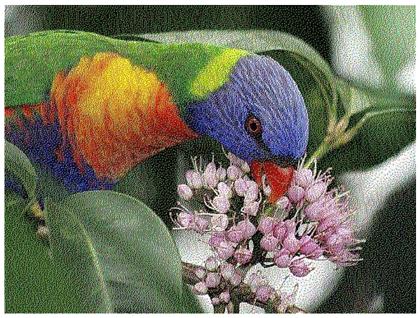


Figure 32 Separable ED

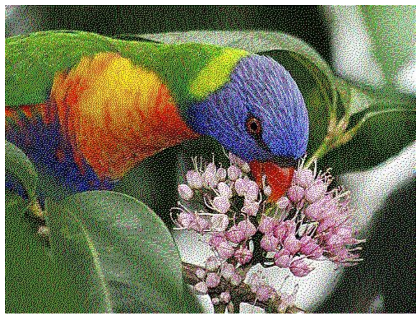


Figure 33 MBVQ

IV. Discussion

1) Dithering

a. Thresholding

As seen from the output, constant thresholding output is pure black and white with immense loss of details and contouring.

- computationally fast but huge loss in detail

On applying random thresholding above drawback seems to be avoided, but the noise tends to swamp the detail of the image.

- more image detailing but with irregular noise resulting from quantization

b. Dithering matrix

The dithering is applied for different thresholding matrices ranging from n = 2 to 32.

- As seen from the output (fig 5-8), we see that quantization error resulting from random thresholding is reduced as we introduce a uniform noise. Also, as n increases the image quality gets better but the image intensity seems to be corrupted.
- Though simple to implement, this dithering algorithm is not easily changed to work with free-form, arbitrary palettes. It results in an easily identifiable cross-hatch pattern artifact.

c. Error Diffusion

Error diffusion is applied using 3 different filters - Floyd Steinberg, JNN and Stucki.
The output is shown from fig 10-12.
FS results in very fine-grained dithering. With JNN, dithering is coarser, but has fewer artifacts. However, JNN is slower than FS.
Stucki output is clean and sharp as error is being diffused to a greater number of neighboring pixels. Also, it is slightly faster than remaining 2.

Comparing the output with Dithering matrix, we see that image intensity is being preserved as because of error diffusion to neighboring pixels, local average intensity of the image remains close to the original image. Also, this helped reduced the cross-shaped artifact seen in dithering method but not without giving a slightly worm-like artifact along horizontal direction. This is mainly due to modulated randomization

Error Diffusion is preferred over dithering as it is computationally faster and gives much cleaner and sharper output with less artifacts.

<u>Improvement:</u>

It is possible to remove worm like artifacts by randomizing weight distribution during diffusion. However, there are also structural artifacts in low gradient regions. To improve this, an efficient method is proposed in the paper- "Simple gradient-based error-diffusion method (by Xiangyu Hu, Technische Universität München)". Here, the input image is separated into the flatten and detailed areas based on a gradient-based measure and randomization is applied to remove the structural artifact. Then, a gradient based diffusion is applied to remove structural artifact. This algorithm is proven to be easy to implement and computationally efficient outperforming previous methods.

2) Color Halftoning

a. Separable Error Diffusion

- The output is shown in figure 14. As seen the output is fine grained and almost resembles original image. However, there are noticeable patterns. This is mainly due to variation in brightness of the dots.
- This disadvantage is overcome with the use MBVQ.

b. MBVQ

- It is based on the fact human vision better perceives small differences of lightness in small local areas, then similar differences of hue in the same area, and even more than similar differences of saturation on the same area.
- Because, better visual results may be obtained by first converting the color channels into a perceptive color model that will separate lightness, hue and saturation channels, so that a higher weight for error diffusion will be given to the lightness channel, than to the hue channel.

- As seen from the output in figure 15, Using this method, the artifacts are significantly reduced. The output exhibits much smoother color with significantly reduced objectionable color variation.

References:

- Wikipedia
- Homework 2 Q pdf
- Class reading material
- Fast Edge Detection paper by Piotr
- Color Diffusion paper by Shaked