Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

Problem 1: Edge Detection

1. Abstract and Motivation:

The early stages of vision processing identify features in images that are relevant to estimating the structure and properties of objects in a scene. Edges are one such feature. Edges are significant local changes in the image and are important features for analyzing images. Edges typically occur on the boundary between two different regions in an image. Edge detection is frequently the first step in recovering information from images. Due to its importance, edge detection continues to be an active research area.

The main idea is to calculate gradients in each direction of the image pixels and find the intensity variations with the help of slope and magnitude of gradients.

(a) Sobel Edge Detector

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically, it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

In theory at least, the operator consists of a pair of 3×3 convolution kernels as shown in Figure 1. One kernel is simply the other rotated by 90°.

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

Gx

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

Gy

Figure 1: Kernel matrices for x and y gradient

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

separately to the input image, to produce separate measurements of the gradient component in each orientation (call these *Gx* and *Gy*). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

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

The magnitude value is then normalized and a cumulative histogram is obtained to choose a threshold. Based on the threshold, the pixel values are turned on or off as edges.

(b) Canny edge detector

Canny edge detector has following steps:

(i) Preprocessing: Edge detectors are prone to noise. A bit of smoothing with a Gaussian blur helps.

(ii) Calculating gradients: Next, gradient magnitudes and directions are calculated at every single point in the image. The magnitude of the gradient at a point determines if it possibly lies on an edge or not. A high gradient magnitude means the colors are changing rapidly - implying an edge. A low gradient implies no substantial changes. So it's not an edge. The direction of the gradient shows how the edge is oriented. The gradients can be calculated in the similar way like Sobel detector.

(iii) Non maximal suppression: In this step, the pixels that are actually not edges but give high values for gradient magnitudes are suppressed with a technique called non maximal suppression. In this technique, the neighbors of the pixels which have high gradient magnitude values along with the direction of gradient at that pixel is chosen. It is suppressed if there is substantial zero crossing from its neighbors otherwise retained.

(iv) Hysteresis thresholding: This step involves using double threshold values to classify any given pixel into definitely edge, definitely not edge and probably edges. All the gradient magnitudes that are above given higher threshold is definitely an edge. All the gradient magnitudes below given lower threshold are classified as definitely not an edge. Of those that lie between the higher and lower threshold values, classify it as definitely edge if its contours are definitely edge otherwise classify not definitely edge.

(c) Structured Edge

The flow diagram and method of Structured edge is written in the discussion part for Structured edge.

(d) Performance Evaluation

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

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

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

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

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

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.

2. Approach and Procedure

(a) Sobel Edge detector

- The given input raw image is read byte by byte into an input imagedata of three dimension with three channels.
- The given imagedata was then converted to grayscale image by using weighted average of each channels r, g and b.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

- The gray image was then boundary extracted by one row and column on each side.
- The extended boundary image was convolved with sobel x gradient and y gradient kernals to get x gradient and y gradient values.
- The gradient values in each direction was normalized and written as a raw file each.
- The magnitude of the gradient was found using formula described above in the abstract part.
- The histogram was found for the intensity values in the magnitude array. Further, normalized cumulative histogram was generated and plotted. The intensity value corresponding to 80% of cumulative histogram was noted.
- All the pixels in magnitude array having above the threshold intensity was classified as an edge (black) and the rest were classified to be non edges (white).
- The output image was then stored and written to a raw file. Outputs were observed and analyzed.

(b) Canny Edge detector

- The installation procedure for opencv using C++ was carefully followed and implemented.
- The input jpg image was read by opencv imread function.
- The image was converted from color to gray using cv functions.
- The gray scale image was denoised with gaussian blur.
- Then, canny was implemented on the denoised image by giving lower and higher threshold values.
- The output obtained was observed and analyzed.

(c) Structured edge

- The procedure and instructions of the given github source was followed carefully.
- The edgesDemo.m file was run by varying the model parameters and observed the outputs.
- The results obtained was observed and analyzed.

(d) Performance Evaluation

- The probability edge map for each of tiger and pig images using sobel, canny and structured edge was read in matlab.
- Edgesdemo.m and EdgesevalImg.m was run to evaluate performance of each detectors in terms of F measure.
- Plots and Values for F measure for each of the ground truth given was compared in a tabular form.

3. Discussions

(a) Sobel Edge Detector

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

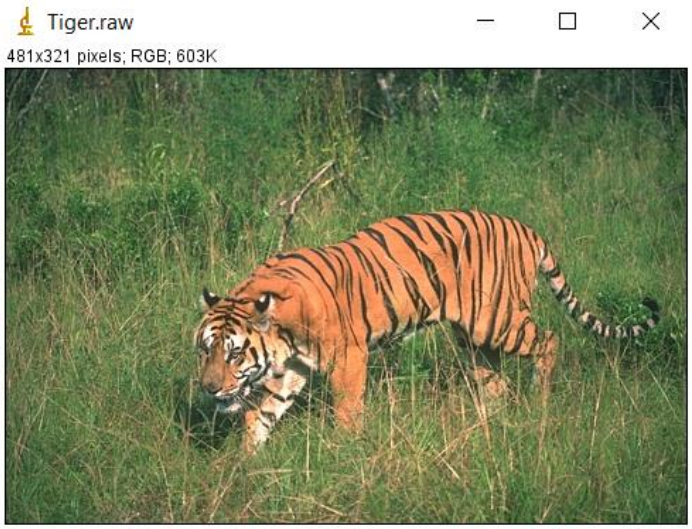


Figure 2: Original Tiger Image

Figure 3: Original Pig Image

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

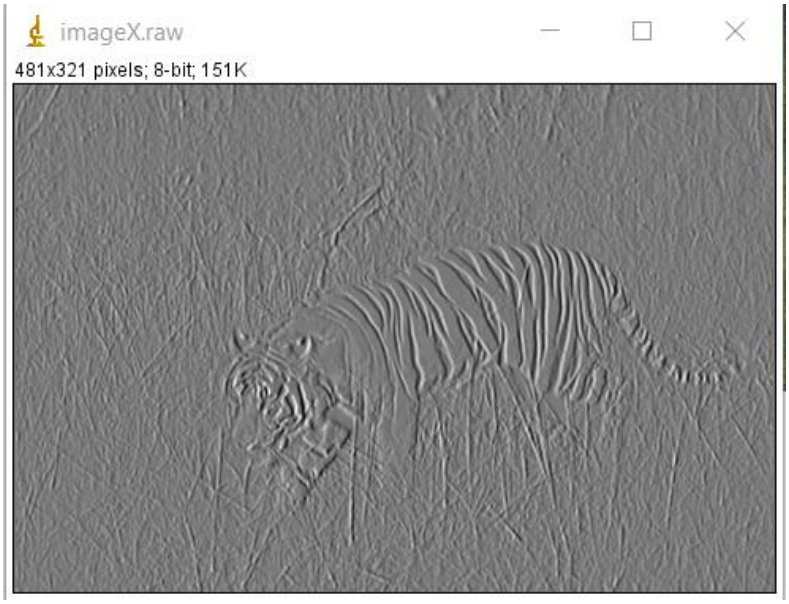


Figure 4: X gradient for Tiger image

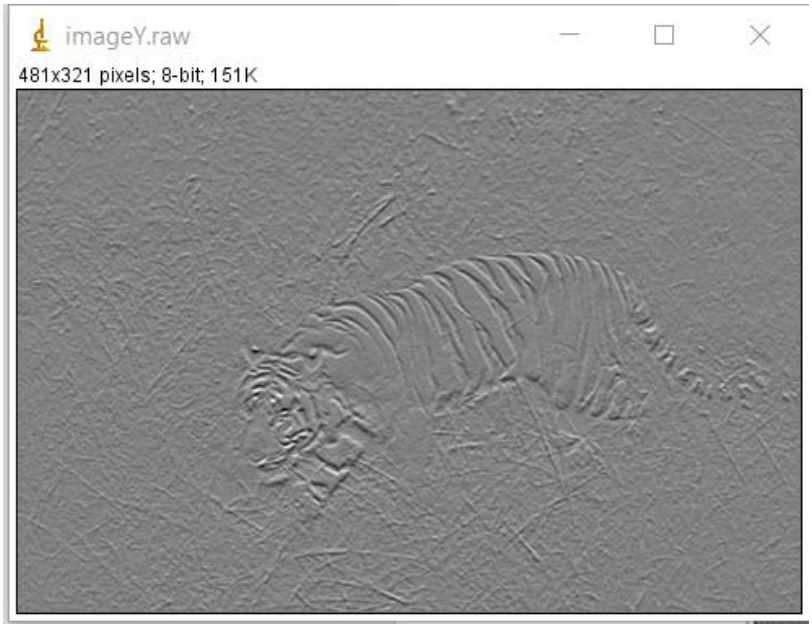


Figure 5: Y gradient for Tiger Image

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

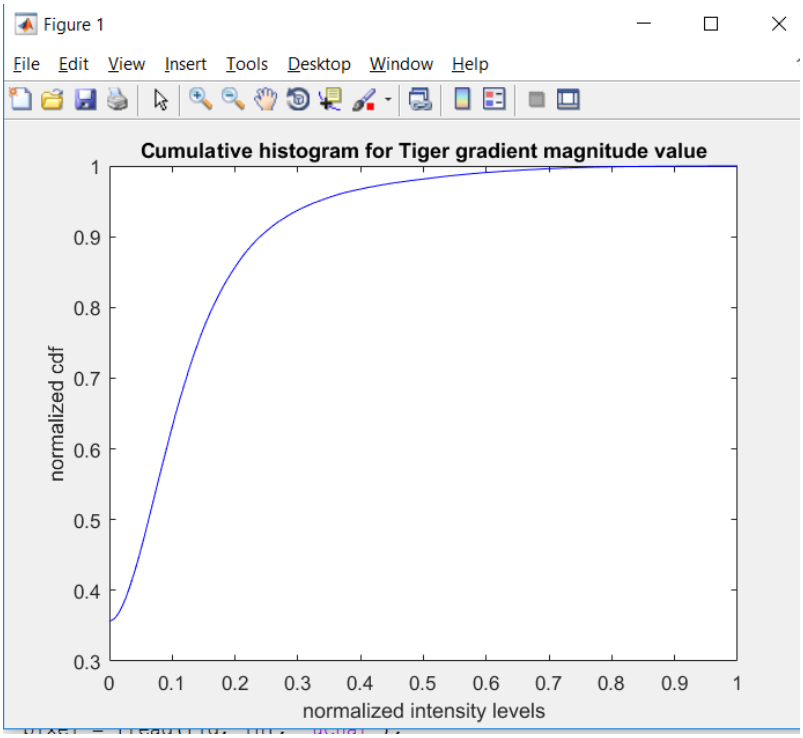


Figure 6: Cumulative histogram map for Tiger image. Threshold of 0.9 was chosen.

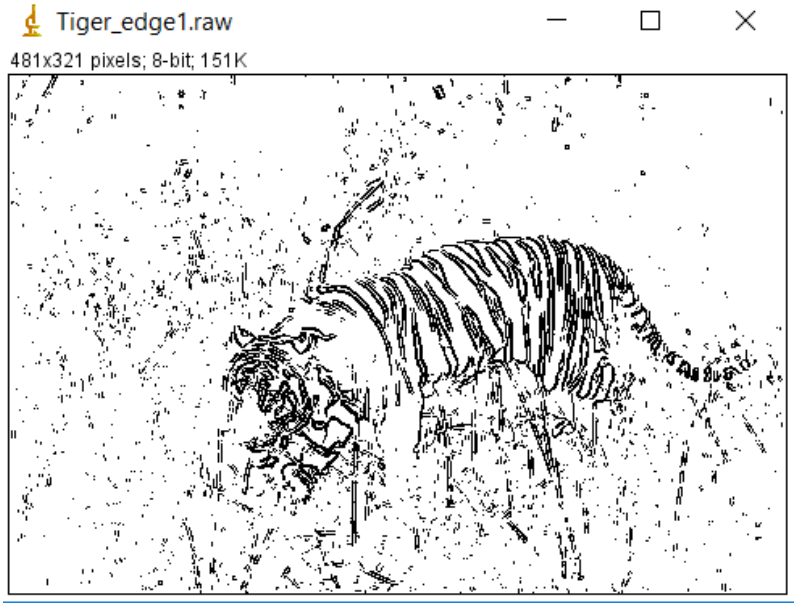


Figure 7: Output for Sobel using 90% threshold on cumulative histogram of gradient magnitudes

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

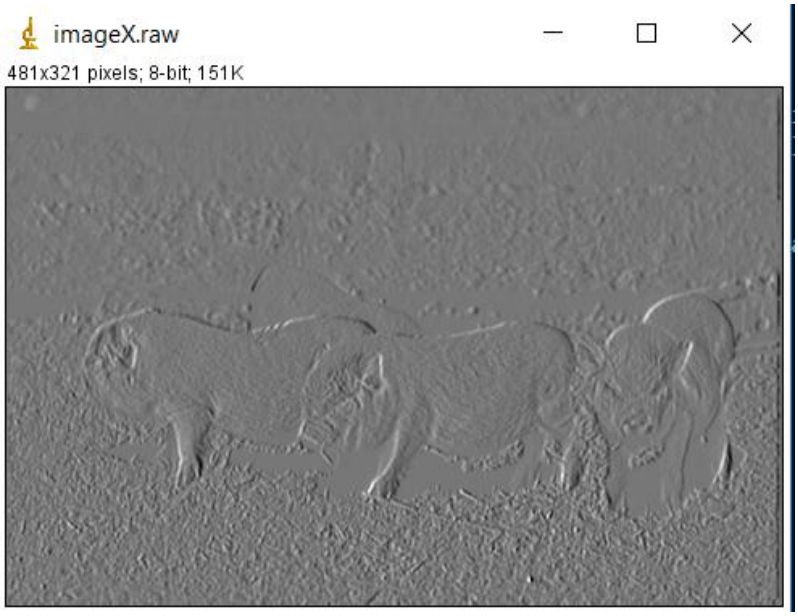


Figure 8: X gradient for Pig image

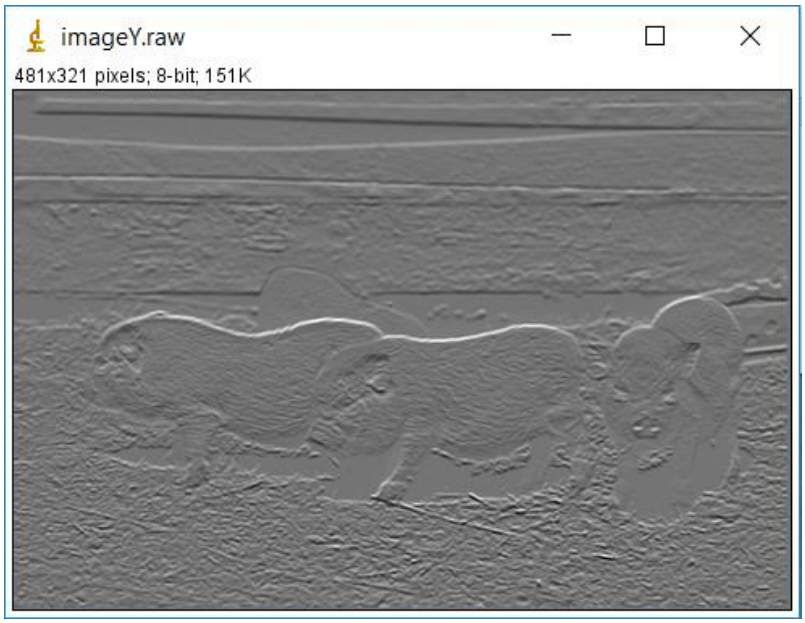


Figure9: Y gradient for Pig image

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

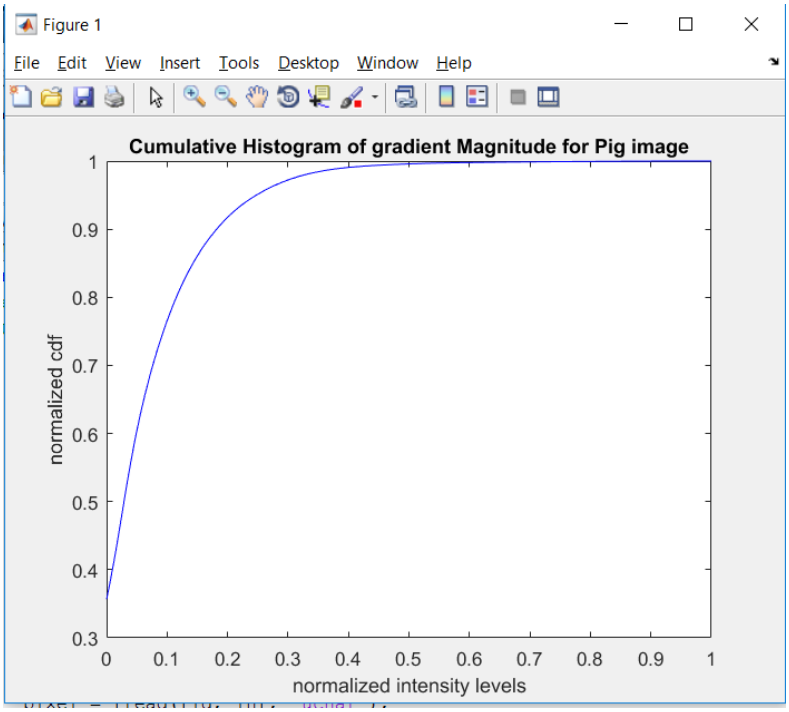


Figure 9: Cumulative histogram map for Pig image. Threshold of 0.9 was chosen.

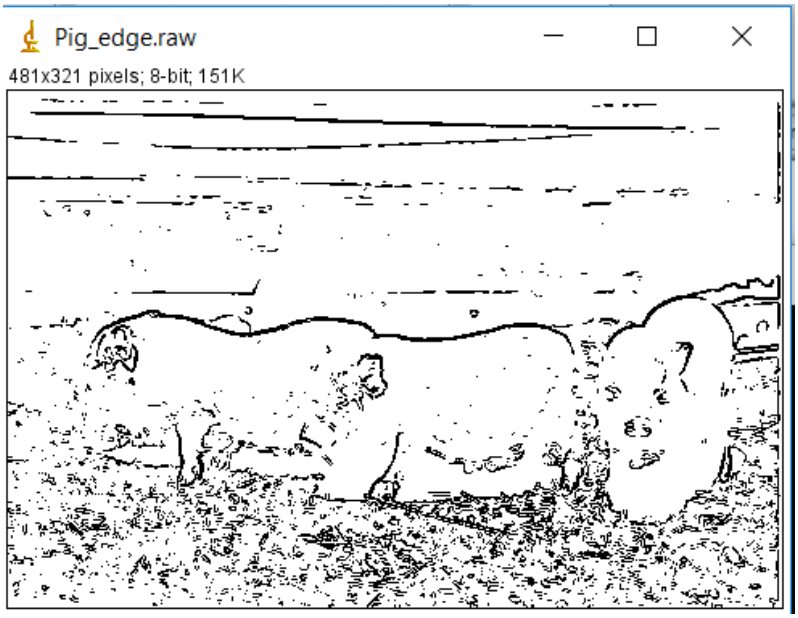


Figure 10: Output for Sobel using 90% threshold on cumulative histogram of gradient magnitudes

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

Discussion:

- Faster and easier to implement.
- Preserves the structure very well.
- Edges are not thin,
- Sensitive to noise.
- Weak localization
- Pixelwise noise can be seen.

(b) Canny Edge Detector

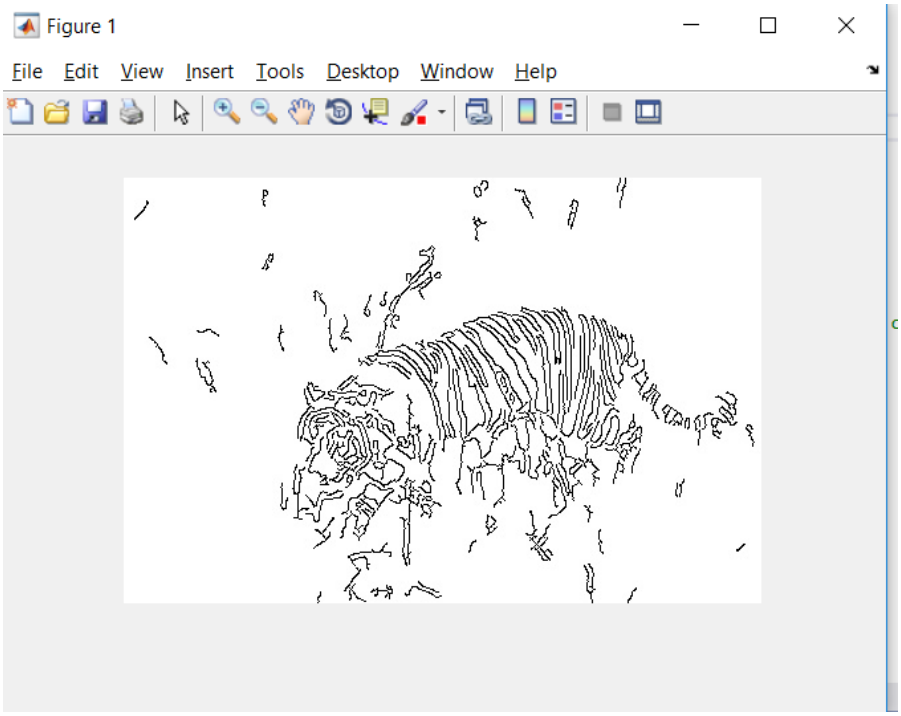


Figure 12: Canny result for Tiger image with lower threshold 85, upper threshold 135

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

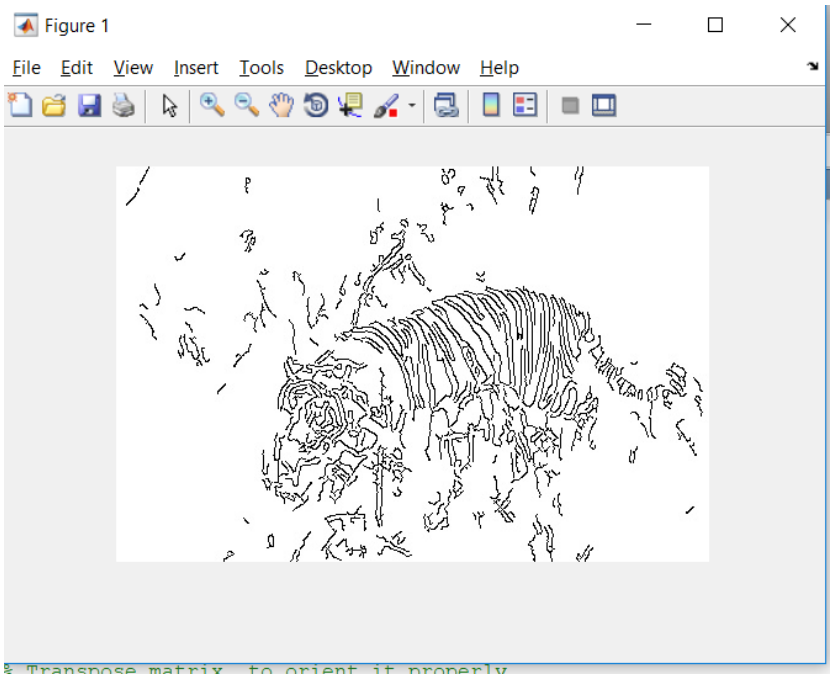


Figure 13: Canny result for Tiger image with lower threshold 65, upper threshold 135

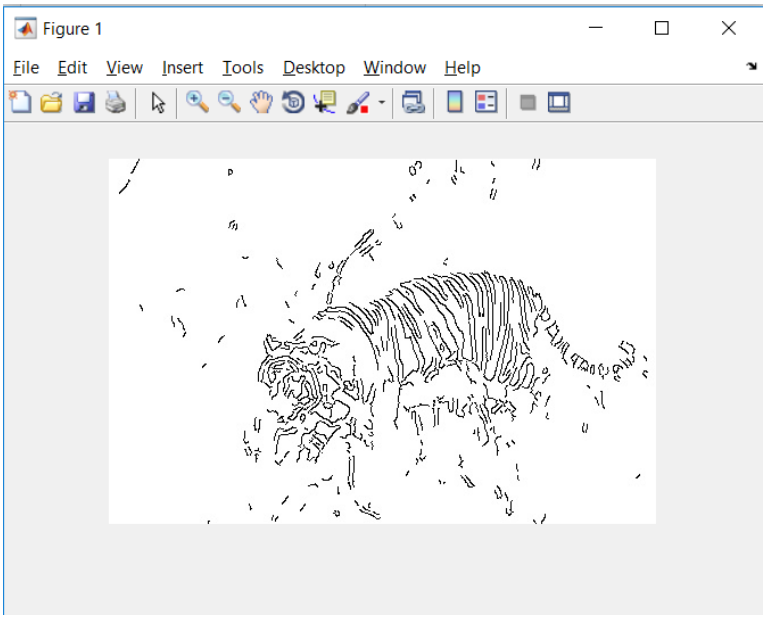


Figure 14: Canny result for Tiger image with lower threshold 105, upper threshold 145

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

Figure 15: Canny result for Pig image with lower threshold 65, upper threshold 165

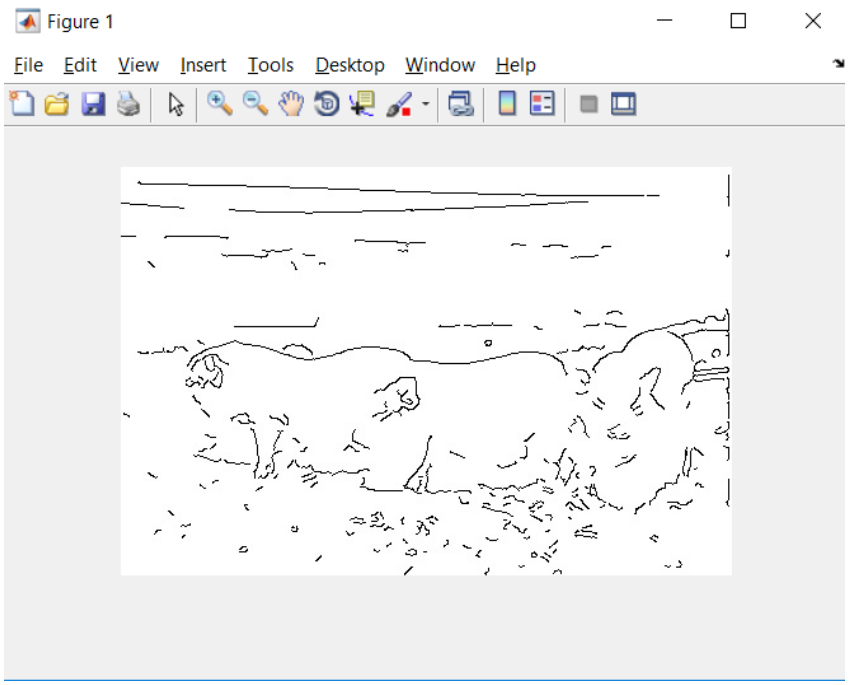


Figure 17: Canny result for Pig image with lower threshold 105, upper threshold 145

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

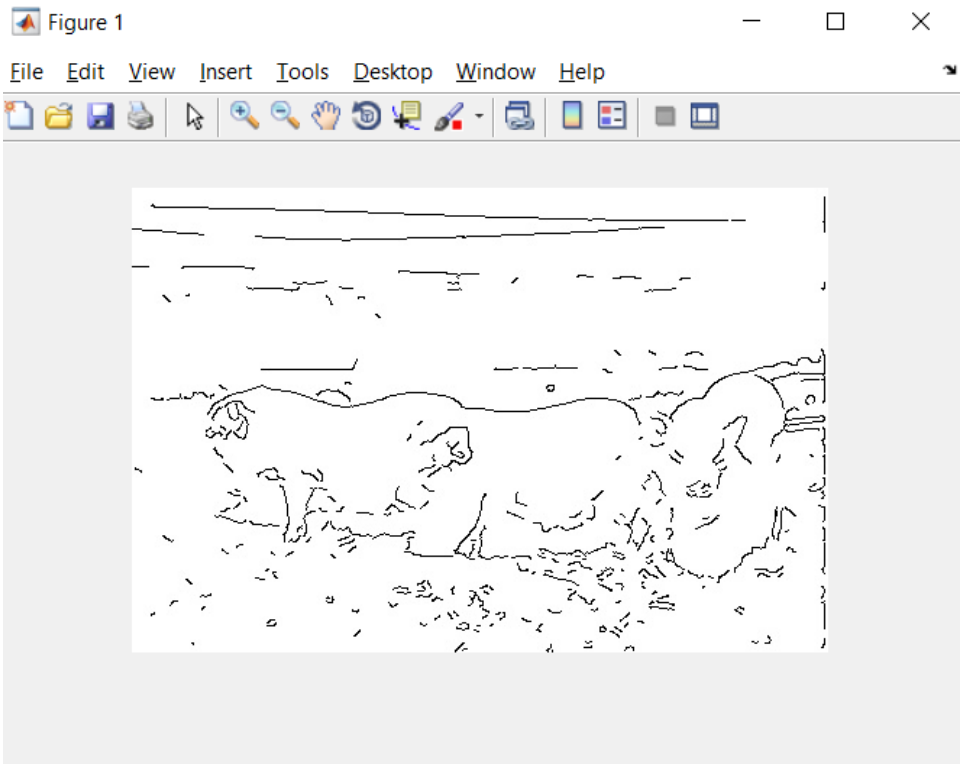


Figure 18: Canny result for Pig image with lower threshold 105, upper threshold 135

Discussion:

- Better edges compared to Sobel as it involves denoising in the first step.
- Preferred over Sobel for weak edges and higher noises.
- Enhances Signal to noise ratio.
- Better localization as opposed to Sobel.
- Connected edges due to double thresholding.
- Difficult to implement and slower.
- Sensitive to texture regions.
- Not enough algorithm for image interpretation as can be seen in Tiger and Pig images.
- Variable threshold values might be required for complex images.

Non maximal Suppression: In this step, the pixels that are actually not edges but give high values for gradient magnitudes are suppressed with a technique called non maximal suppression. In this technique, the neighbors of the pixels which have high gradient magnitude values along with the direction of gradient at that pixel is chosen. It is suppressed if there is substantial zero crossing from its neighbors otherwise retained.

Double thresholding: This step involves using double threshold values to classify any given pixel into definitely edge, definitely not edge and probably edges. All the gradient magnitudes that are above

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

given higher threshold is definitely an edge. All the gradient magnitudes below given lower threshold are classified as definitely not an edge. Of those that lie between the higher and lower threshold values, classify it as definitely edge if its contours are definitely edge otherwise classify not definitely edge.

(c) Structured Edge

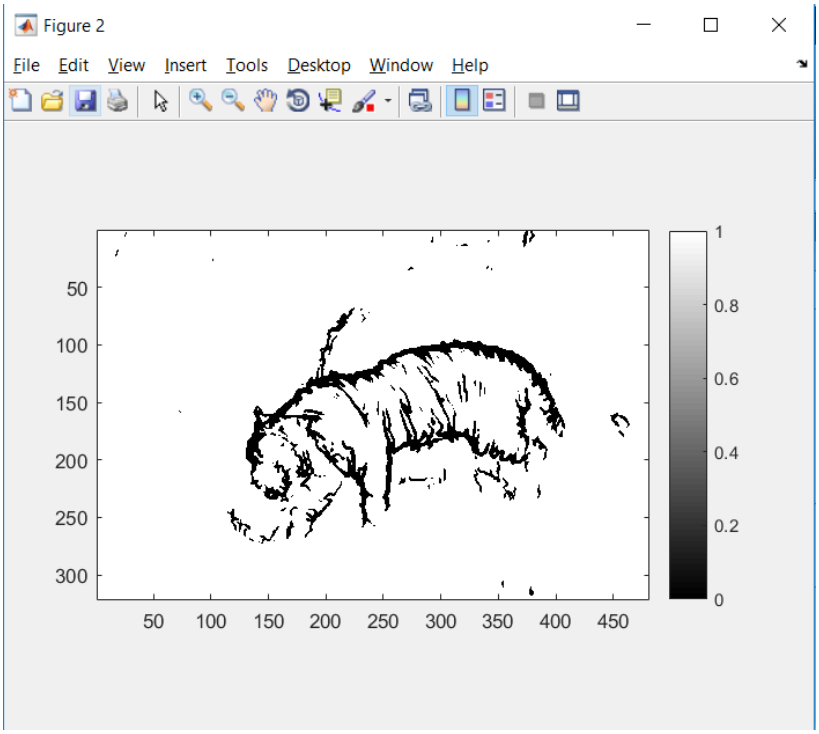


Figure 19: Output for Tiger Image with Structured using threshold 0.9

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

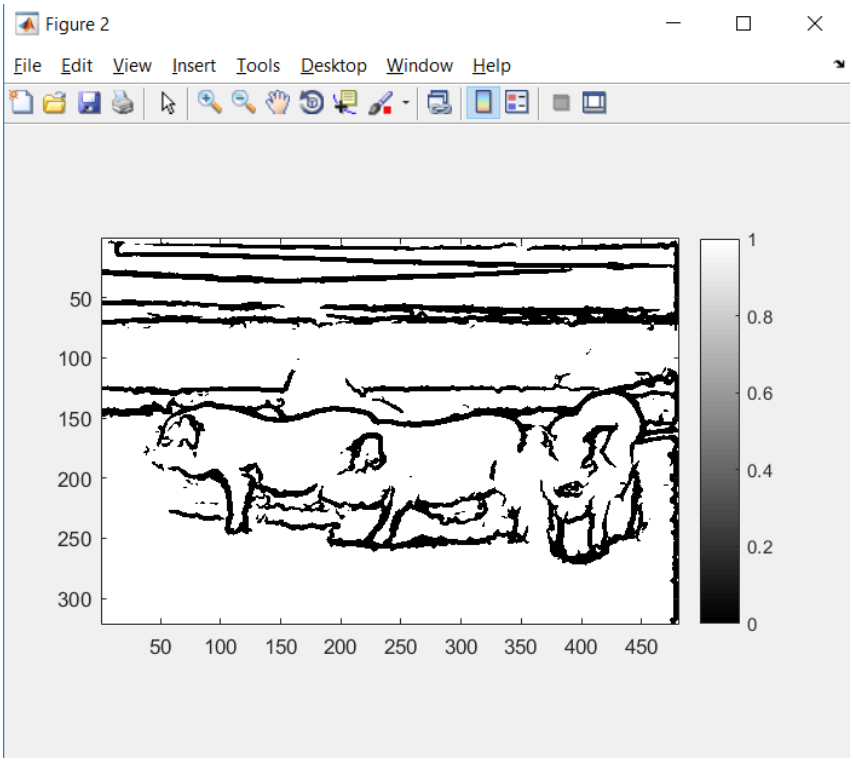
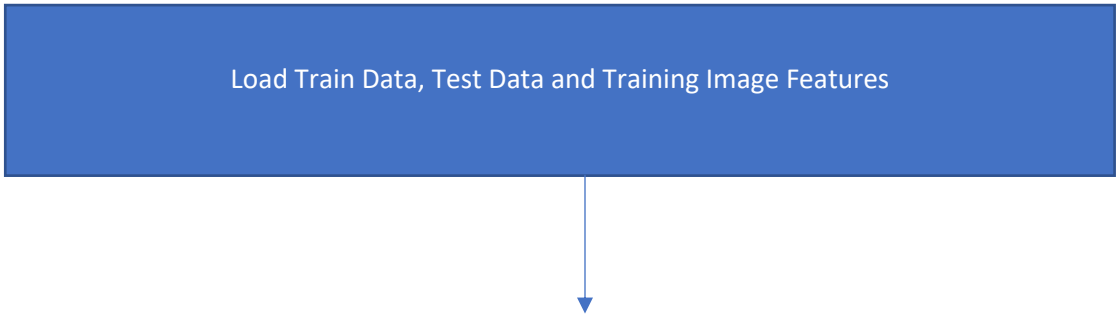


Figure 20: Output for Pig Image using Structured edge with threshold 0.9

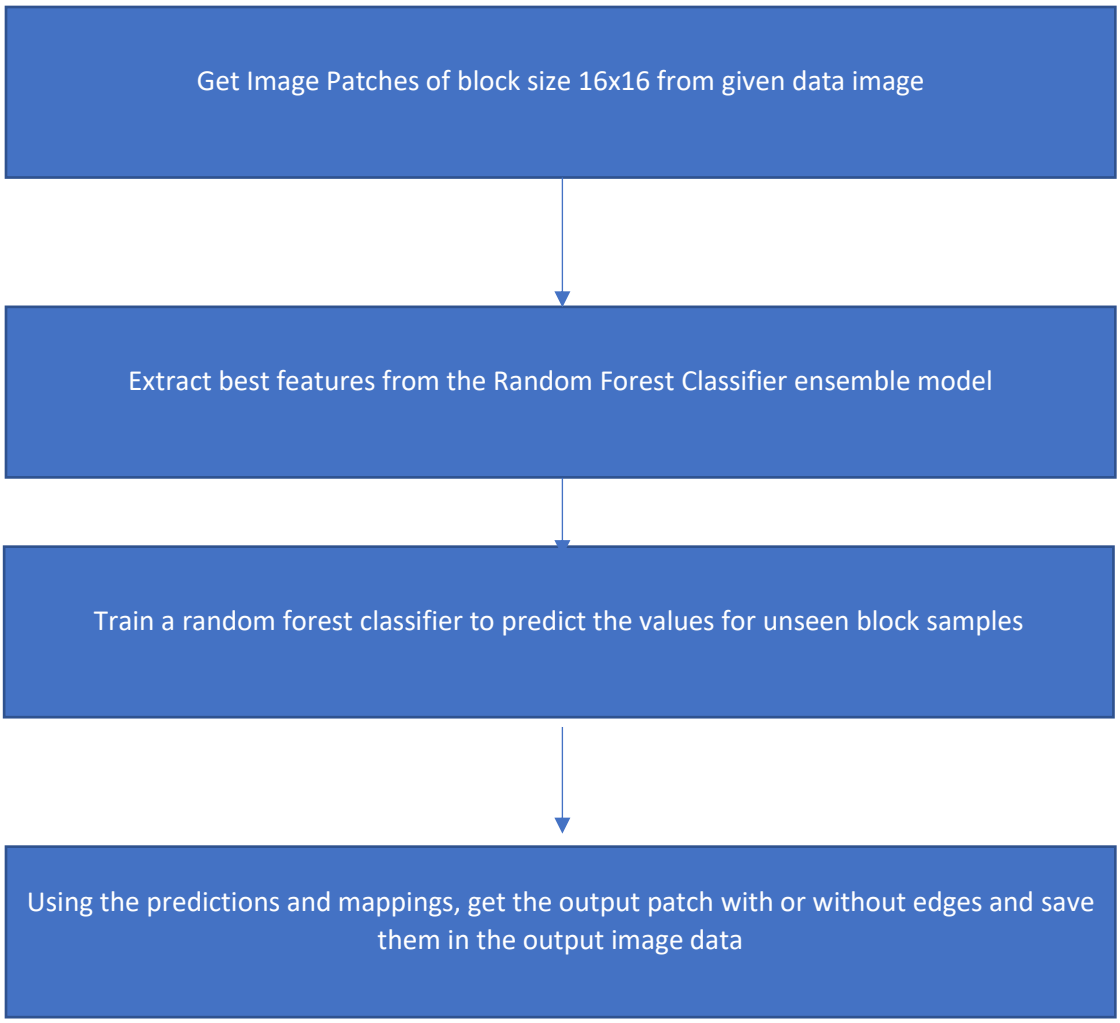
Discussion:

- Very fast computation and more accurate than techniques like Sketch Tones.
- High quality edge detection with clean result and efficiency.
- Very good application in time sensitive object recognition tasks.
- Since many images contain structured data, it is good to use structured random forests.
- Low computation cost in using random forest classifier with high ability to select effective features.
- Not sensitive to feature normalization.
- Sufficient diversity of trees could avoid overfit problem.
- Challenging part is high output feature dimension (2^256)
- Hard to calculate information gain.

Flowchart of algorithm:



Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019



Step1&2: Collect a set of segmented imges for training purpose in which boundaries between segments correspond to contours. Patches of 16x16 is selected and trained using the classifier to know about the features.

Step3: Features extraction is done choosing pairwise features among all the features present. The feature dimension is nor (2^256). These features are downsampled to lesser features per pixel channel using a transfer domain function from input Y space to output Z space.

Step4: Random Forest Classifier is trained on these features. The detailed explanation of Random forest is explained below.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

Step 5: Similarity between structures is observed and predicted by Random Forest classifier. The predicted decisions from each forest is averaged to get final prediction.

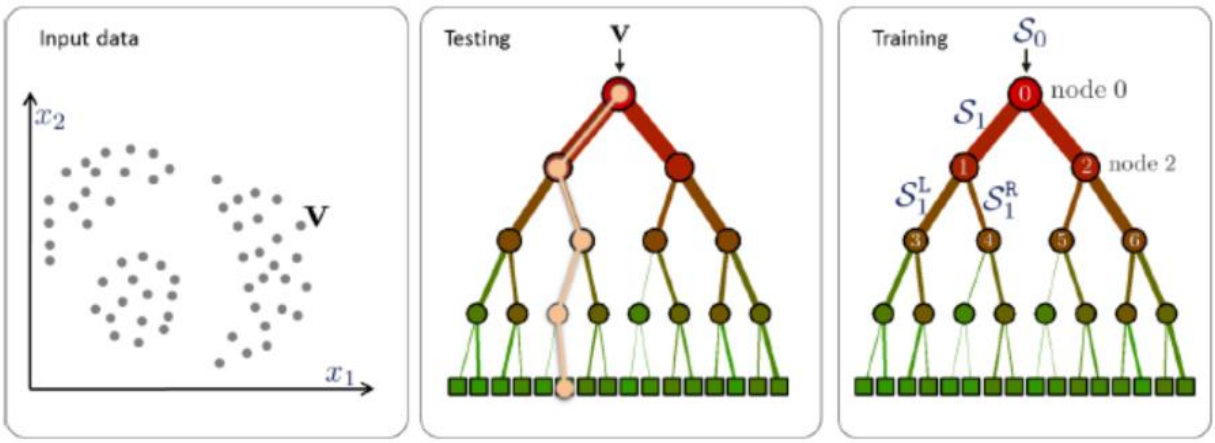
Random Forest Classifier

Random forest classifier 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.

Suppose training set is given as [x1,x2,x3] with labels [y1,y2,y3], Random Forest creates decision trees as follows:

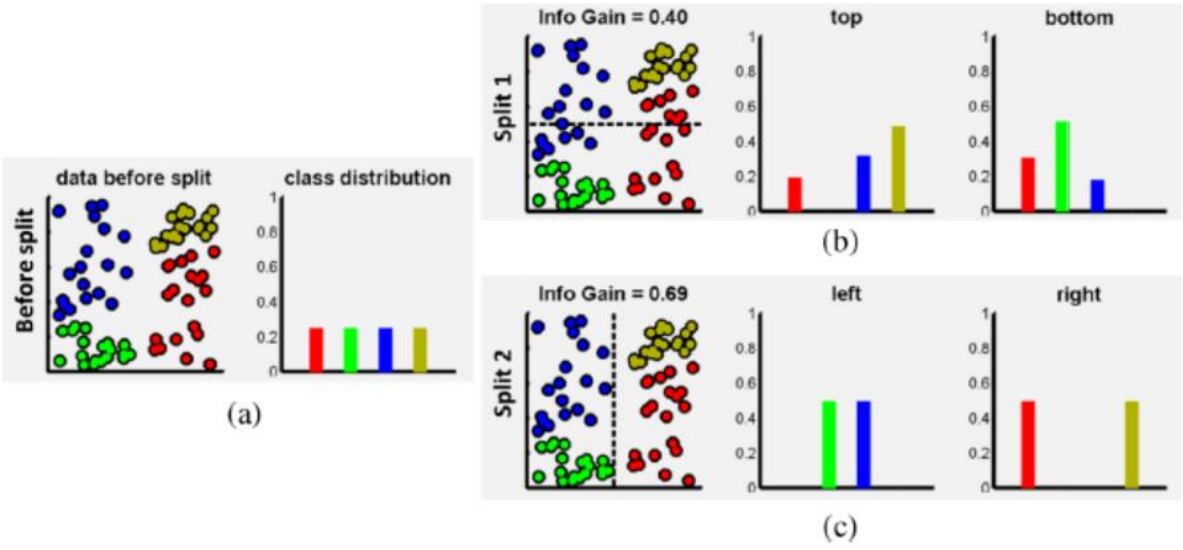
(i) [x1,x2]
(ii) [x2,x3]
(iii) [x1,x3]

Each tree makes a decision based on the features and branching. The split at each node depends on a constraint function that minimizes the error in the left and right split further. The final result is decided by taking majority vote or average of decisions of each trees depending on classification or regression respectively. The structure is shown below:



(a) (b) (c)

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019



(i) Structured Edge

(a) For threshold values of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 the edgesEvalImg was run for five ground truth of Tiger.mat was run. The values of R, P and F ae shown below. The plot is also shown:

R =	P =	F =
0.0561	0.6537	0.1034
0.1696	0.4136	0.2406
0.1662	0.2874	0.2106
0.1462	0.2077	0.1716
0.2151	0.2544	0.2331
0.2640	0.2191	0.2394
0.3095	0.1922	0.2371
0.4377	0.2545	0.3218
0.4007	0.2309	0.2930
0.1261	0.1614	0.1416

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

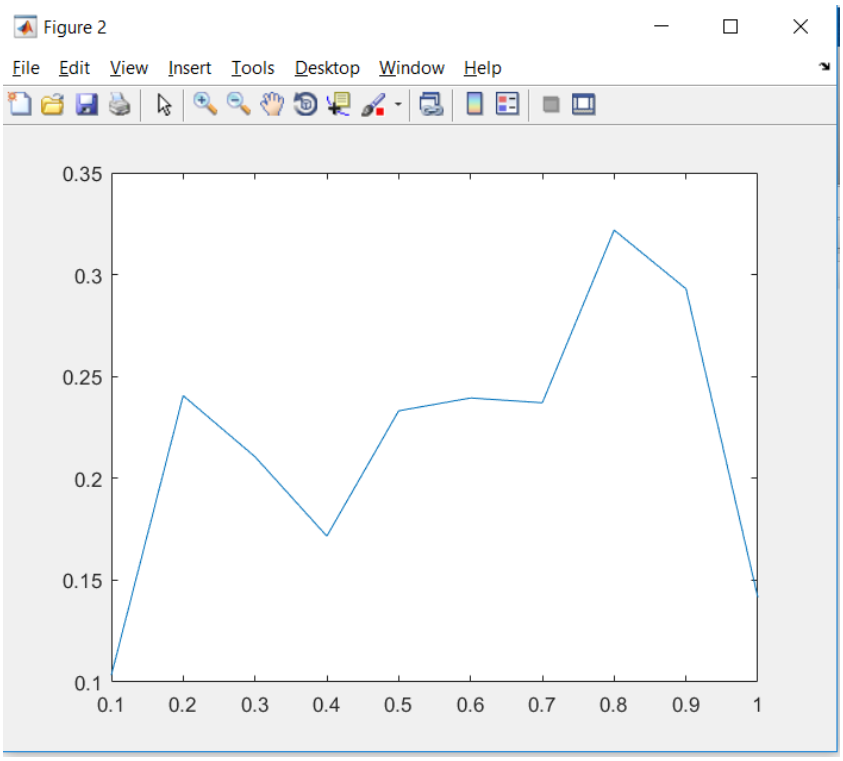


Figure 20: F measure for Tiger Image using Structured Edge with ten thresholds

For Pig Image,

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

For threshold values of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 the edgesEvalImg was run for five ground truth of Pig.mat was run. The values of R, P and F ae shown below. The plot is also shown:

R =	P =	F =
0.0283	0.2148	0.0501
0.1270	0.3037	0.1791
0.1713	0.3947	0.2389
0.1855	0.4421	0.2613
0.2544	0.3903	0.3080
0.3183	0.3393	0.3285
0.3919	0.3622	0.3765
0.4803	0.3762	0.4219
0.4419	0.3561	0.3943
0.0123	0.0395	0.0188

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

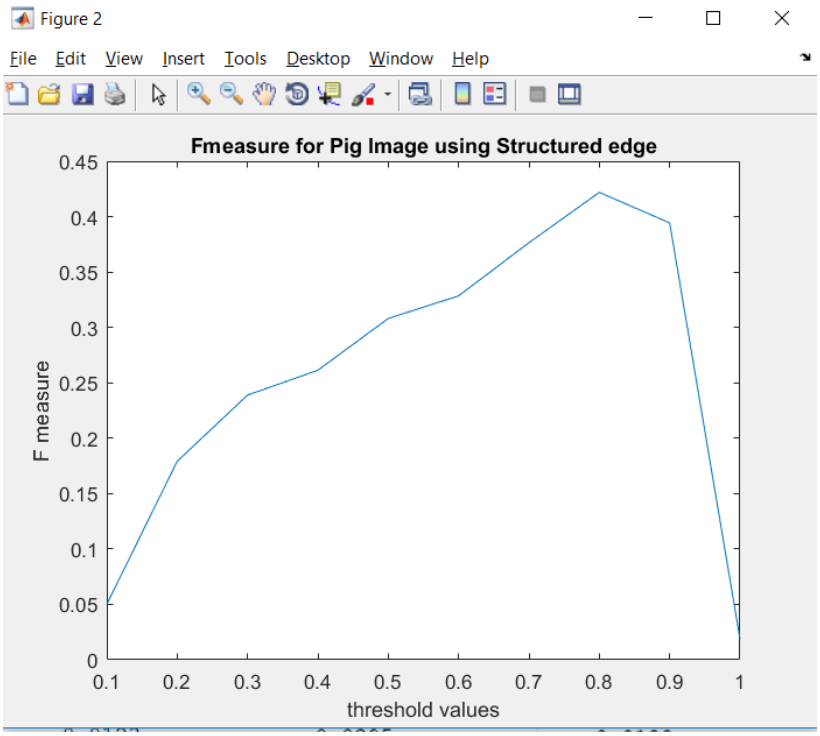


Figure 20: F measure for Tiger Image using Structured Edge with ten thresholds

(b) For each ground truth table, mean P, mean R and mean F was taken for all ten threshold of Tiger Image. This is shown below:

	Ground Truth 1	Ground Truth 2	Ground Truth 3	Ground Truth 4	Ground Truth 5
Mean R	0.1716	0.1739	0.1721	0.1795	0.1545
Mean P	0.0625	0.660	0.0741	0.5680	0.1252
Mean F	0.0916	0.0957	0.1036	0.2728	0.1383

For each ground truth table, mean P, mean R and mean F was taken for all ten threshold of Pig Image. This is shown below:

	Ground Truth 1	Ground Truth 2	Ground Truth 3	Ground Truth 4	Ground Truth 5
Mean R	0.2245	0.2253	0.2544	0.2442	0.2436
Mean P	0.1121	0.1194	0.2017	0.2389	0.1974
Mean F	0.1495	0.1561	0.2250	0.2415	0.2181

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

(ii) Sobel

(a) For threshold values of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 the edgesEvalImg was run for five ground truth of Tiger.mat was run. The values of R, P and F ae shown below. The plot is also shown:

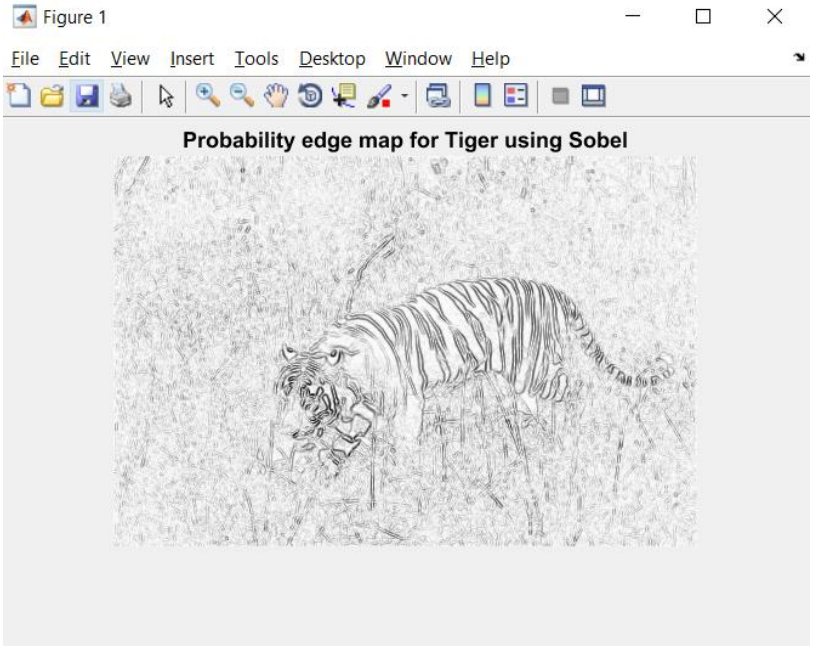


Figure 21: Probability edge map for Tiger using Sobel

R =	P =	F =
0.0938	0.5432	0.1600
0.1691	0.4807	0.2502
0.2819	0.4897	0.3578
0.5893	0.5030	0.5428
0.8210	0.4807	0.6064
0.9287	0.3783	0.5376
0.9775	0.2781	0.4330
0.9825	0.1801	0.3044
0.9205	0.1246	0.2195
0.0068	0.2185	0.0132

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

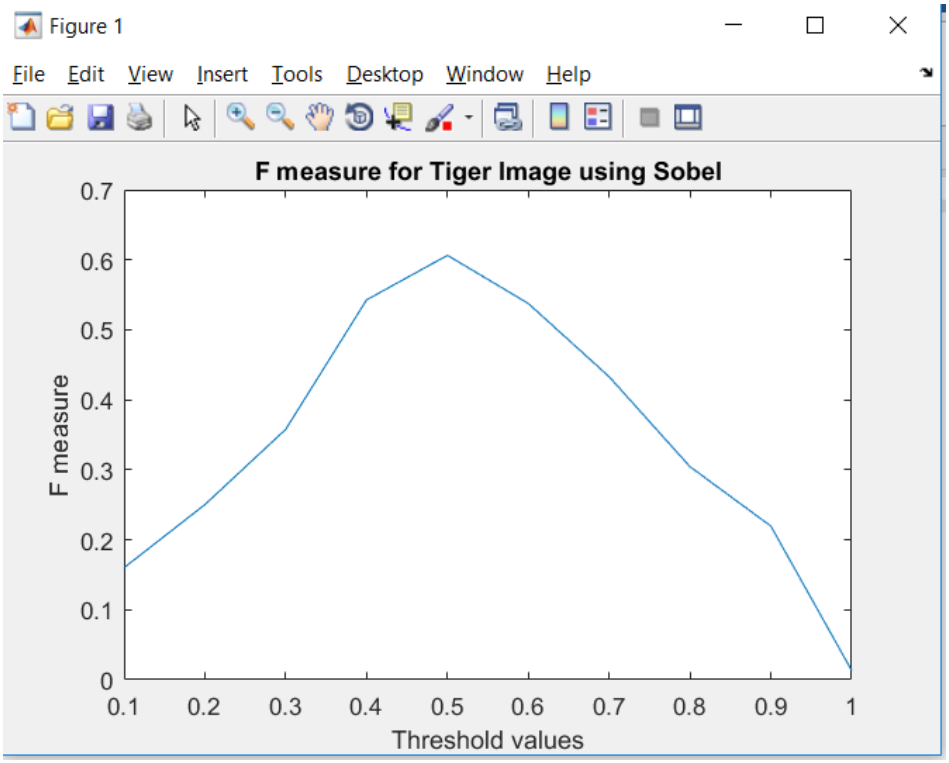


Figure 22: F measure for Tiger Image using Sobel Detector with ten thresholds

For Pig Image,

) For threshold values of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 the edgesEvalImg was run for five ground truth of .mat was run. The values of R, P and F ae shown below. The plot is also shown:

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

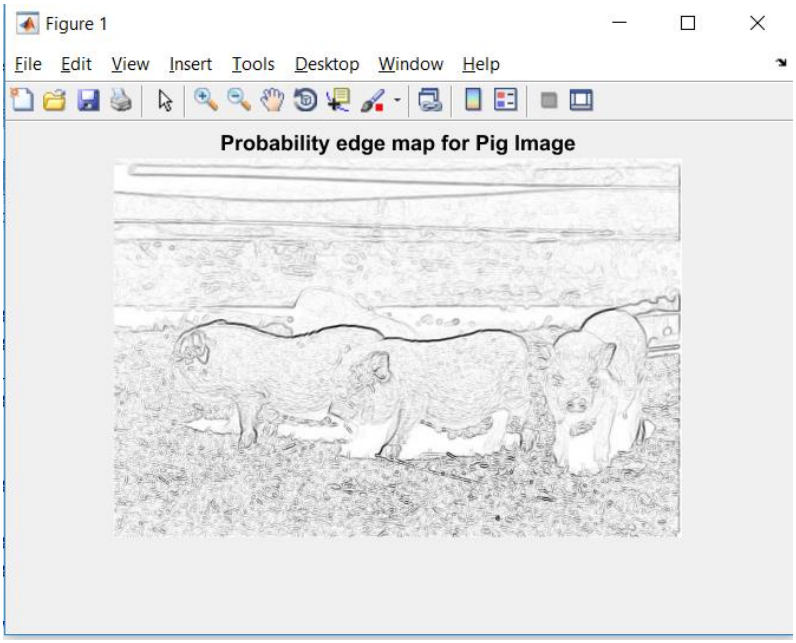


Figure 23: Probability edge map for Pig using Sobel

```
R =

    0.1093
    0.3528
    0.3292
    0.2563
    0.2927
    0.3895
    0.5535
    0.7345
    0.9396
    0.0925
```

```
P =

    0.4878
    0.4641
    0.3548
    0.2378
    0.1958
    0.1789
    0.1557
    0.1590
    0.1851
    0.2441
```

```
F =

    0.1786
    0.4009
    0.3415
    0.2467
    0.2347
    0.2452
    0.2431
    0.2614
    0.3093
    0.1341
```

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

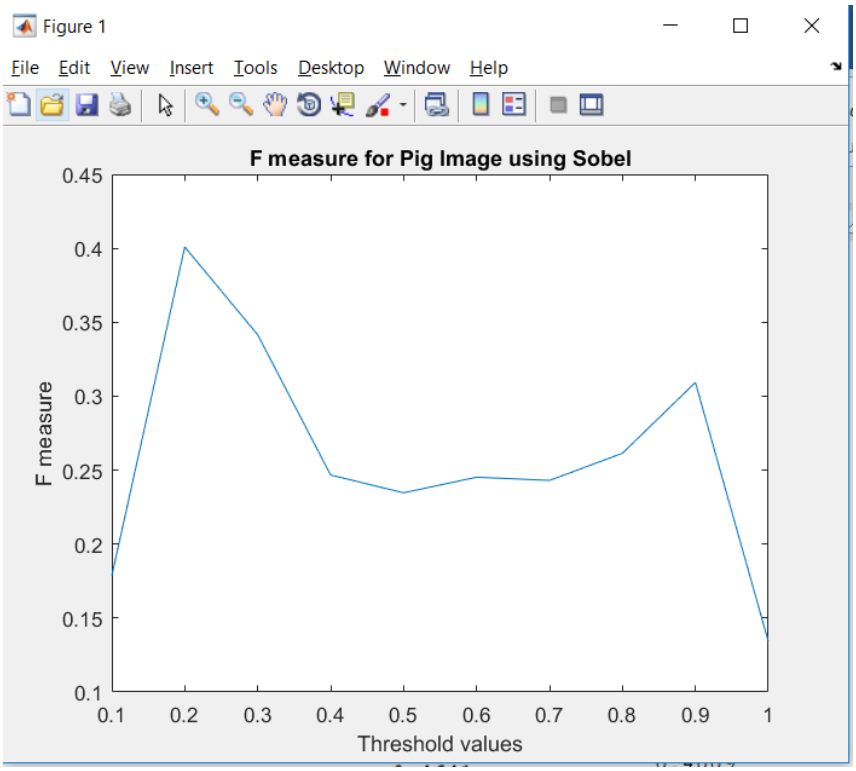


Figure 24: F measure for Pig Image using Sobel with ten thresholds

(c) For each ground truth table, mean P, mean R and mean F was taken for all ten threshold of Tiger Image. This is shown below:

	Ground Truth 1	Ground Truth 2	Ground Truth 3	Ground Truth 4	Ground Truth 5
Mean R	0.5935	0.6056	0.5876	0.5785	0.5291
Mean P	0.0716	0.0831	0.0855	0.3252	0.0915
Mean F	0.1278	0.1462	0.1493	0.4164	0.1561

For each ground truth table, mean P, mean R and mean F was taken for all ten threshold of Pig Image. This is shown below:

	Ground Truth 1	Ground Truth 2	Ground Truth 3	Ground Truth 4	Ground Truth 5
Mean R	0.4532	0.4470	0.3792	0.3327	0.3584
Mean P	0.1340	0.1396	0.1696	0.1720	0.1552
Mean F	0.2069	0.2127	0.2343	0.2268	0.2166

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

Discussion:

(1) As can be seen from the figures and results above, the higher precsion is seen for Sobel than Structured edges for given ground truths.
(2) F measure is image dependent. Of the images Tiger.jpg and Pidg.jpg, Pig image is easier to get a higher F measure. It is because of more texture in the Tiger images near the tiger and harder to predict the edges because of the environment.
(3) Rationale behind F measure is that it is averaged in inverse principle because of the power ability:

$1/F = ½ *(1/R + 1/P)$
$F = 2* P * R / (P+R)$

If $P >> R$,
$1/F$ is nearly equal to $1/R$ and hence F is nearly equal to R. Hence, its not possible to achieve high F measure and vice versa.

If $P + R$ is constant,
F depends on $P*R$.
Now the problem reduces to maximizing the area of rectangle with length P and breadth R which is a square with $P = R$.

Problem 2: Digital Half-toning

4. Abstract and Motivation:

A rendering device such as printer makes use of only two levels of intensities to print a gray scale image: black or white. Yet, the image printed can be visualized as different tones due to the continuous-toned property of images. The process of generating patterns of pixels with limited number of colors, when seen by the human eye, is called **digital halftoning.** The eye acts as a spatial low pass filter that blurs the rendered pixel pattern, so that it is perceived as a continuous-tone image. There are several techniques that achieve digital half-toning. The basic idea is to implement an algorithm to find a threshold which acts as a barrier between colors to be chosen. The model efficiency depends on choosing a threshold and manipulating rest of the pixel values in some way so as to get maximum quality in the rendered output images. The digital half-toning techniques also help in data compression. In this way, a large image with varied intensity levels in given number of channels can as well be transferred using smaller bandwidths where only the half=toned values are transmitted. The general structure for half-toning and examples are shown in Figure 1 and Figure 2 respectively.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

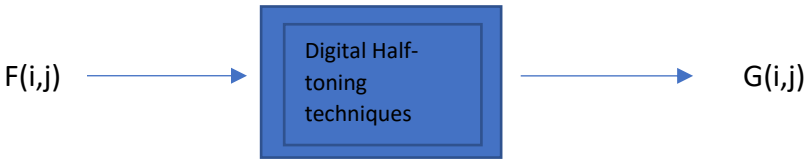


Figure 1: Digital-halftoning structure

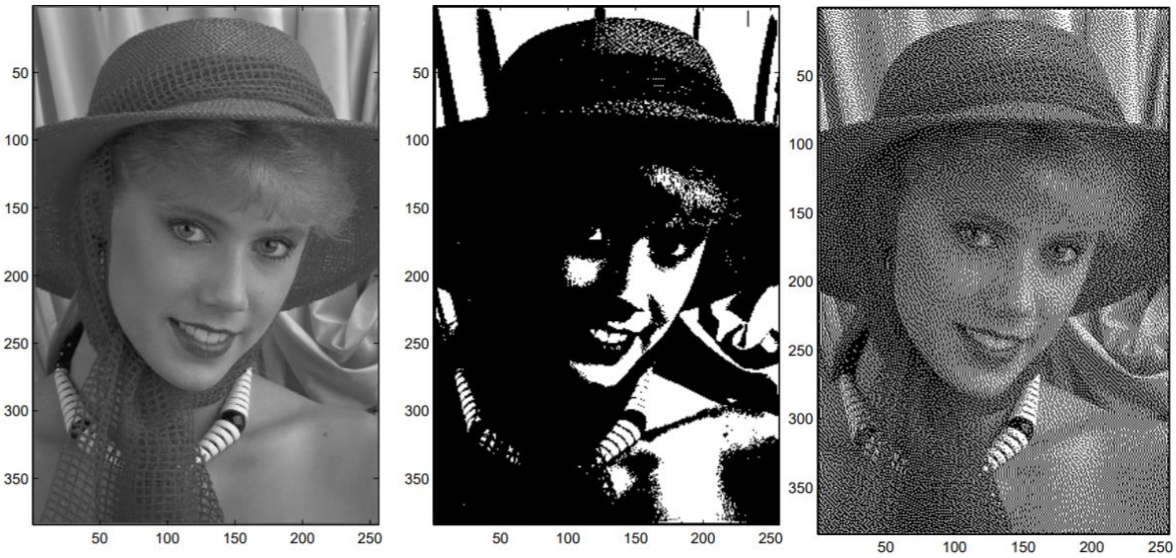


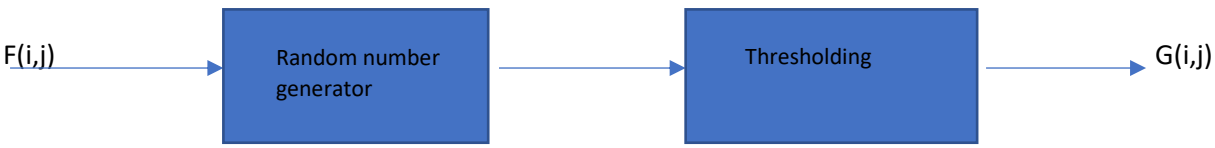
Figure 2: (a) Original Image (b) Thresholded Image (c) Half-toned images

(a) Dithering

Dithering is a technique used to implement digital half-toning by finding a threshold which is used to decide the intensity level of output pixel values. Since we're dealing with gray scale images in this part, we quantize every pixel to two levels: black or white depending on the threshold. There are different ways to obtain the thresholds depending on several techniques described below:

(i) Random thresholding

The simplest of techniques is dithering using random thresholding. The idea is to randomly generate uniform samples from 0-255 and use it as a threshold for every pixel. A random number generator is employed to obtain a random threshold. If the current pixel value exceeds the threshold the output pixel for that location is made white (255) otherwise black (0). The structure of random thresholding is as shown in Figure 3.



Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

Figure 3: Structure for random thresholding

(ii) Dithering matrix

It is another pointwise operation technique in which a dithering matrix is made use of in accounting with the threshold. For a region with k dots, k thresholds should be distributed uniformly so as to eliminate trade-off between the number of gray levels and resolution. Hence, Bayer came up with a matrix called dithering matrix as shown below:

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

The matrix above is a 2x2 dithering matrix used as thresholding for input pixels. The idea is that the values in the matrix describe the pixels most or least likely to be turned on. 0 indicates most likely to be turned on and 3 indicates least likely to be turned on. The pixels are turned on in the following order:

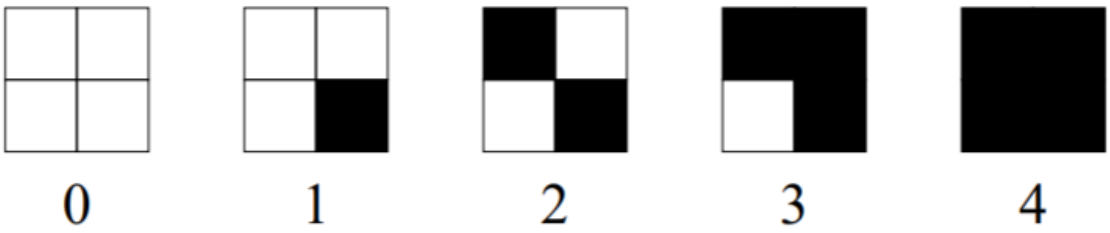


Figure 4: Pixel turning on based on the dithering matrix

The interesting thing about Bayer's matrix is that it is recursive for any multiples of 2 order matrices as shown below:

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

A threshold matrix is found using the dithering matrix which is compared with corresponding pixels in the input to determine the output values. The output pixel is turned on if input pixel is greater than corresponding threshold otherwise it is turned off. The calculation of threshold values at each location is shown as below:

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

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

where x,y represent the locations in the dithering matrix and N^2 is the sum of all the values in the dithering matrix.

(b) Error Diffusion

While dithering matrix is a point based operation that deal with only a particular location of the input pixel, the error diffusion is a spatial neighborhood based operation. In this technique, each pixel is quantized to a level and then error between the quantized level and the updated pixel value is obtained. This error is diffused to its neighborhood to update the pixel values of the neighbors based on several diffusion matrices proposed. The filter view of error diffusion looks like as shown in the Figure 5 below:

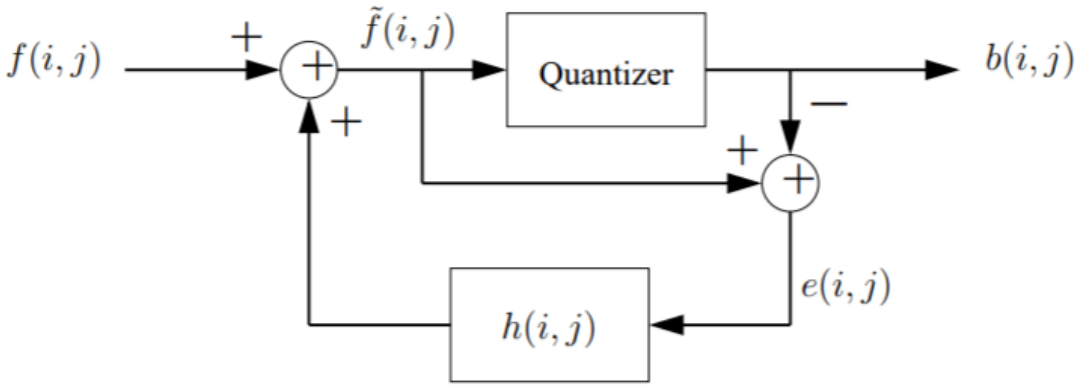


Figure 5: Filter view of error diffusion

A threshold T is chosen and compared with every input pixel value. Based on this threshold, the binary output for gray scale image is given as:

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

The error is pushed forward in the 2D spatial domain using several diffusion techniques. The error is diffused among the local neighborhood as shown in Figure 6.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

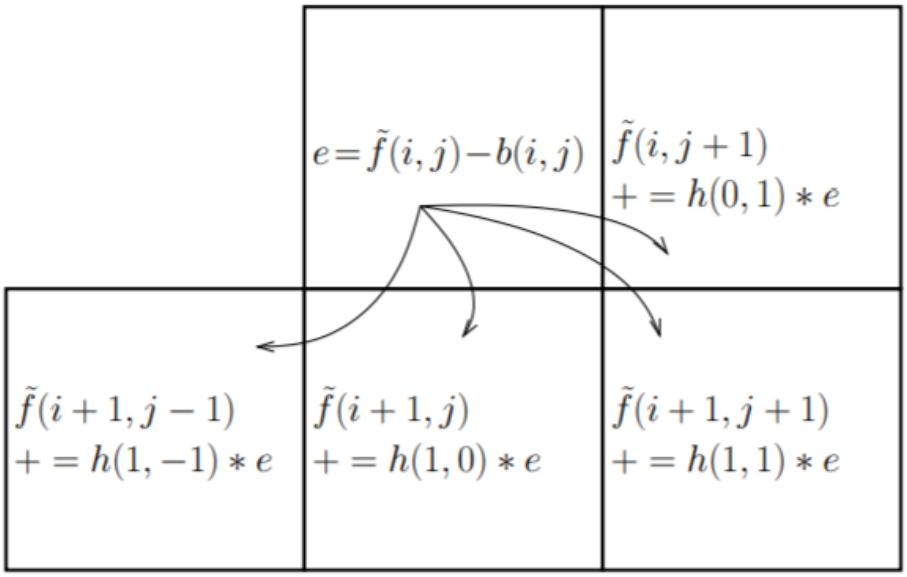


Figure 6: Error diffusion among neighborhood locations

As shown in the Figure 6, the error at each pixel location is diffused in particular proportion h(i,j) among the neighbors. There are several matrices propped for proportion of error to be diffused is described in the next section.

Each of the technique below is carried out based on serpentine scanning of the images. This means that the first row is scanned from left to right and error is diffused based on the error diffusion matrix while the next line is scanned left to right in which error is diffused based on the transpose of the diffusion matrix. The serpentine parsing is as shown below:

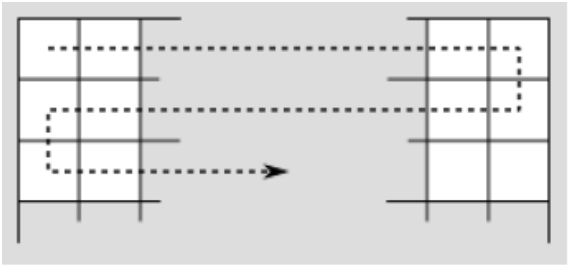


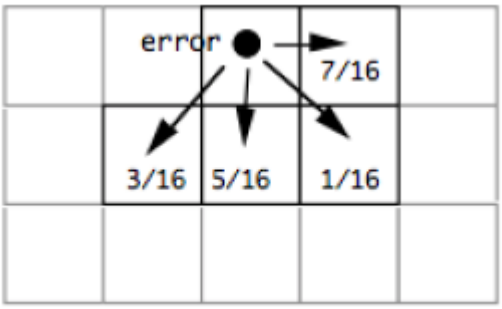
Figure 7: Serpentine parsing of pixels

(i) Floyd-Steinberg's error diffusion matrix:

The Floyd-Steinberg's error diffusion matrix and error diffusion is shown below:

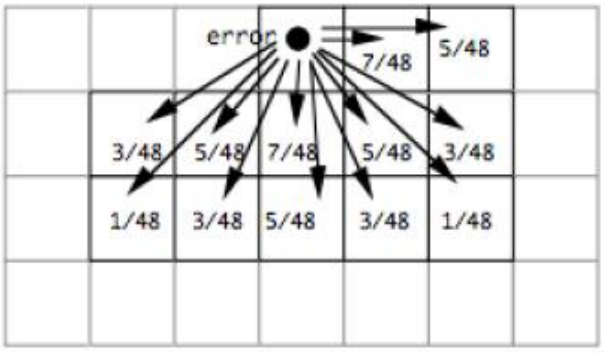
Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

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



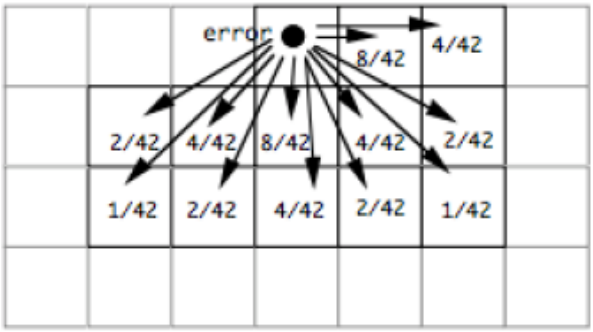
(ii) Jarvis, Judice and Ninke (JJN) error diffusion matrix:
The JJN error diffusion matrix and error diffusion is shown below:

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



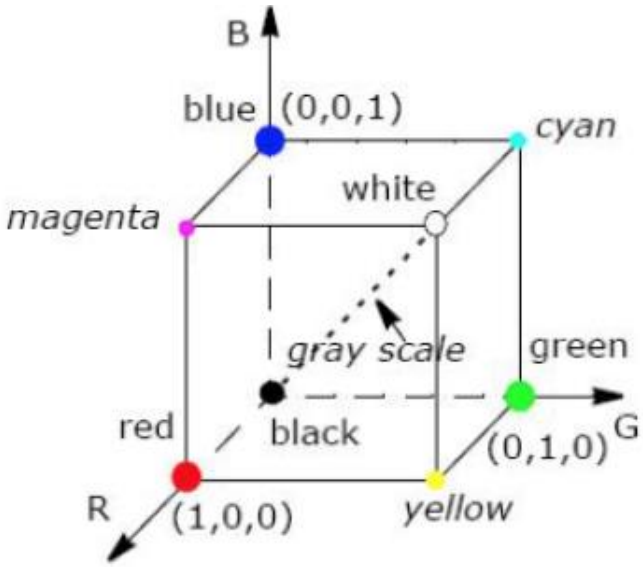
(iii) Stucki error diffusion matrix:
The Stucki error diffusion matrix and error diffusion is shown below:

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



(c) Color Halftoning with error diffusion
Just like its gray scale counterpart, halftoning in color images is carried in each of the three channels. The input image is now quantized into total of 8 values. These 8 values form a cube called CMY cube in which each of the vertex of a unit cube contains a color. The cube representation is shown in Figure 8.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

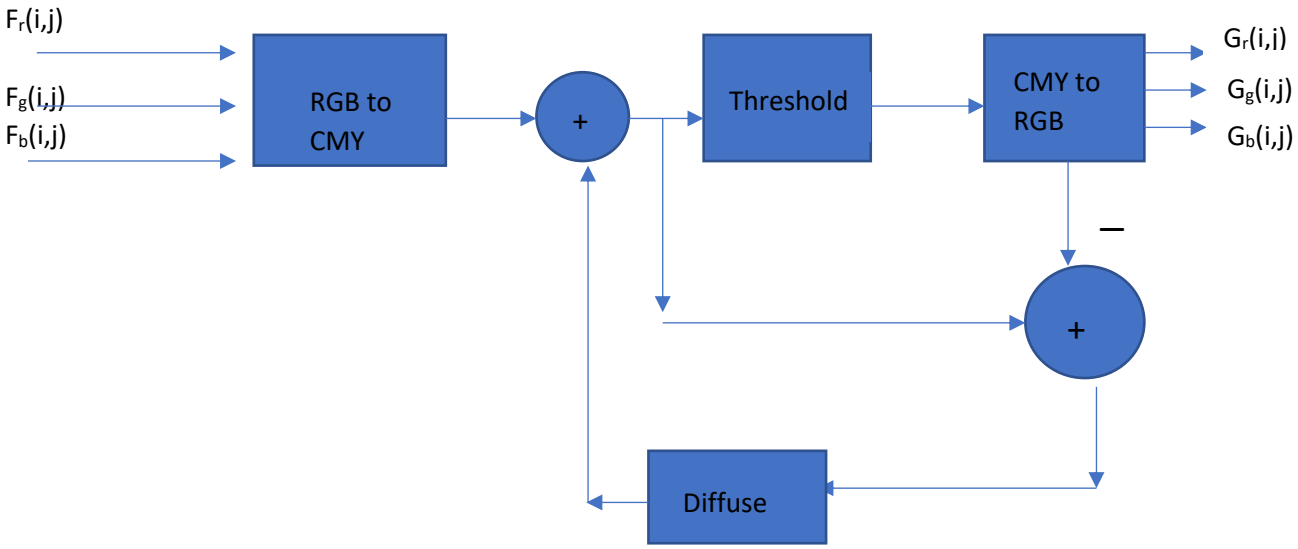


K = (0,0,0)
B = (0,0,1)
G = (0,1,0)
R = (1,0,0)
C = (0,1,1)
M = (1,0,1)
Y = (1,1,0)
W = (1,1,1)

Figure 8: Cube representation of 8 colors

(i) Separable error diffusion

In this technique, the basic error diffusion of the gray scale is extended to three channels of the color image. The r,g,b based color image is first converted to c,m,y channels by using the complimentary values in each channel. Each of the c, m and y channel is now compared with a threshold T to determine output value at each channel. The error based on output pixel value and input updated value is obtained for each channel. This error is pushed forward using Floyd-Steinberg diffusion matrix in each channel. The scanning of each rows follows serpentine parsing method illustrated above. The final output image is again converted to r,g,b channels by complimenting pixel values at each channel again. The structure of Separable error diffusion is as shown below:



Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

(ii) MBVQ based error diffusion method:

A given color in RGB cube may be rendered using 8 basic colors shown in the cube in Figure 8. But, for a matter of fact, any color may be rendered using no more than 4 colors, different colors requiring different quadruples. When presented with high frequency patterns, the human visual system applies a low-pass filter and perceives only their average. For this reason, it is more sensitive to changes in brightness than to changes in chrominance, which average at lower frequencies. Thus, we make use of Minimum Brightness Variation Criterion (MBVC) for halftoning of color pixels.

MBVC says that to reduce halftone noise, select from within all halftone sets by which the desired color may be rendered, the one whose brightness variation is minimal. To consider brightness variation of color sets, the basic eight colors can be arranged on a brightness scale as shown in the Figure 9. Through this method, colors like K or W are used less as compared to the rest of 6 colors.

To reduce the brightness variation, it is proven that the number of participating halftone colors is 4. This results in a quadruple region where any input color can be said to belong to. This region is called Minimal Brightness Variation Quadruple (MBVQ). Hence, any input color can be said to belong to one of six quadruples: RGBK, WCMY, MYGC, RGMY, RGBM or CMGB each of which have minimal brightness variations. These six regions form six tetrahedron in the color cube as shown in Figure 9. Thus, given R, G, B values for a pixel location, we find out MBVQ based on the location of the pixel.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

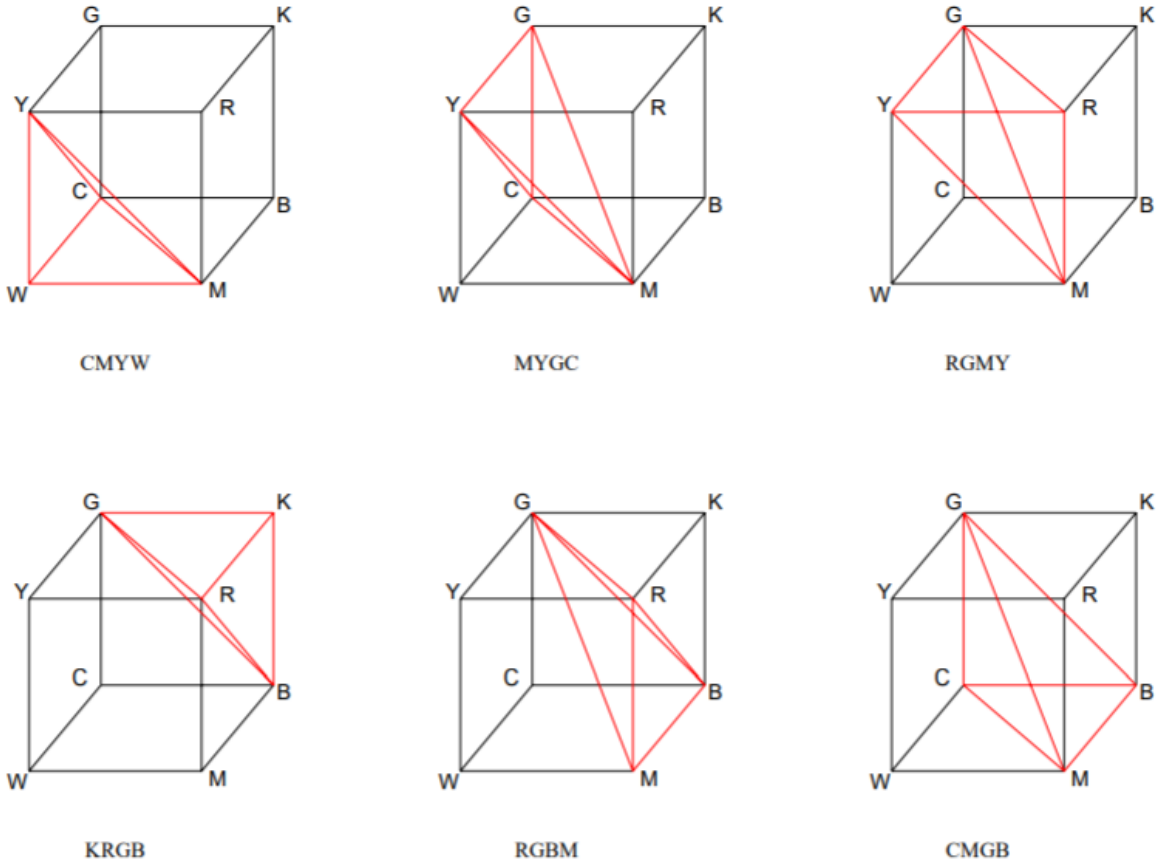


Figure 9: Six tetrahedrons based on MBVQ

Now, the problem is made easier. After the quadruple in which given pixel is said to belong is determined, the vertex in the quadruple closest to the given pixel is found out. The output pixel for the given location is then given the value of the vertex just found out. The error between the vertex and the given input pixel is then pushed forward using Floyd-Steinberg's diffusion matrix. The process is repeated for rest of the pixel values in the same fashion. The procedure to find nearest vertex based on the quadruple is dependent on the location in the cube. An example of finding the nearest vertex in CMGB quadruple is shown in Figure 10.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

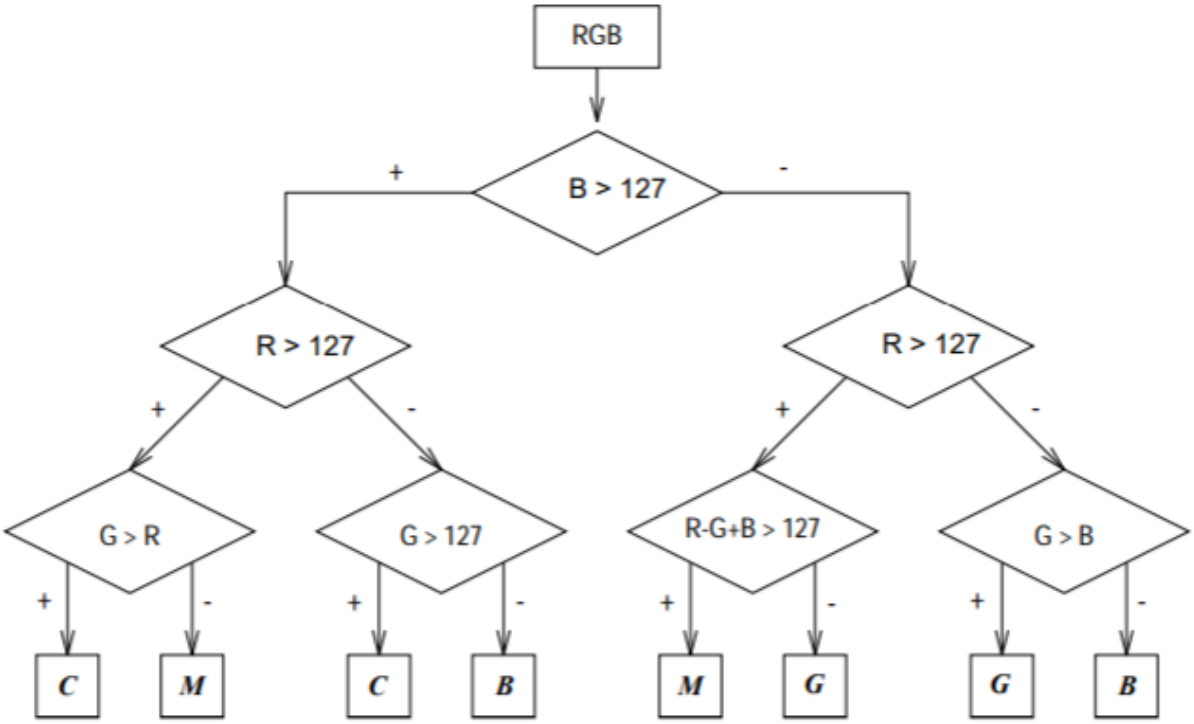


Figure 110: Flowchart for finding the Nearest Vertex in CMGB quadruple

5. Approach and Procedure

(a) Dithering

(i) Random Thresholding:

- The given input image was read byte by byte into an array of three dimension having single channel for color.
- A random number generator was initialized with appropriate seed value.
- Iterating through the input pixel values, each pixel value was compared with the threshold which is a random number generated from 0 – 255.
- The output pixel was set 255 if greater than the threshold otherwise 0.
- The output three-dimensional array was written to a raw file. The results were observed and analyzed.

(ii) Dithering matrix

- The given input image was read byte by byte into an array of three dimension having single channel for color.
- A function getDither was written which takes the arguments as the matrix indices and returns the value of the Bayer matrix at that index.
- Iterating through the input imagedata, the dithering matrix was found for each of 2x2, 8x8 and 32x32 Bayer matrices.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

- Threshold matrix was computed using the equation specified in the abstract above for all three cases.
- Iterating through the image data, each pixel was compared with the corresponding threshold value in the threshold matrix as explained above and the output pixel was turned on or off according to the threshold.
- The output three-dimensional array was written to a raw file. The results were observed and analyzed.

(b) Error diffusion

- The given input image was read byte by byte into an array of three dimension having single channel for color.
- A threshold of 128 was chosen for each pixel value to be compared to. An extended boundary for input image data was created and declared as float variable.
- Iterating through the extended boundary image, each pixel was compared with the threshold and consequently the output pixel was turned on or off.
- The error between output image pixel and the original pixel was calculated as floating value. This error was diffused among the neighbors using each of Floyd-Steinberg, J J N and Stucky error diffusion matrices in serpentine fashion.
- The output three-dimensional array for each error diffusion method was written to a raw file. The results were observed and analyzed.

(c) Color halftoning with error diffusion

(i) Separable error diffusion

- The given input image was read byte by byte into an array of three dimension having three channels for color.
- A threshold of 128 was chosen for each pixel value of each channel to be compared to. An extended boundary for input image data of three dimensions was created and declared as float variable.
- Iterating through the extended boundary image, each pixel of each channel was compared with the threshold and consequently the output pixel was turned on or off. This gives total possibility of eight color values per output pixel location.
- The error between output image pixel and the original pixel was calculated as floating value. This error was diffused among the neighbors using each of Floyd-Steinberg error diffusion matrix in serpentine fashion.
- The output three-dimensional array was written to a raw file. The results were observed and analyzed.

(ii) MBVQ based error diffusion

- The given input image was read byte by byte into an array of three dimension having three channels for color.
- An extended boundary for input image data was created and declared as float variable.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

- A function for finding the mbvq was written which takes in arguments of input image r, g, b values and returns the quadrant in which the pixel location lies.
- A function for finding nearest vertex among the vertices of the quadrant returned by the mbvq function was written.
- Iterating through the extended boundary image and original input image, each input image pixel was given as arguments to mbvq function.
- Further, the extended boundary pixel of this location along with returned mbvq was given as arguments to nearest vertex function and the nearest vertex was found.
- The output image pixel was assigned the value of this vertex and error was diffused into the neighbors of extended boundary image using Floyd Steinberg's error diffusion matrix in serpentine fashion.
- The output three-dimensional array was written to a raw file. The results were observed and analyzed.

6. Discussions

(a) Dithering

The original bridge image is shown in Fig 11.

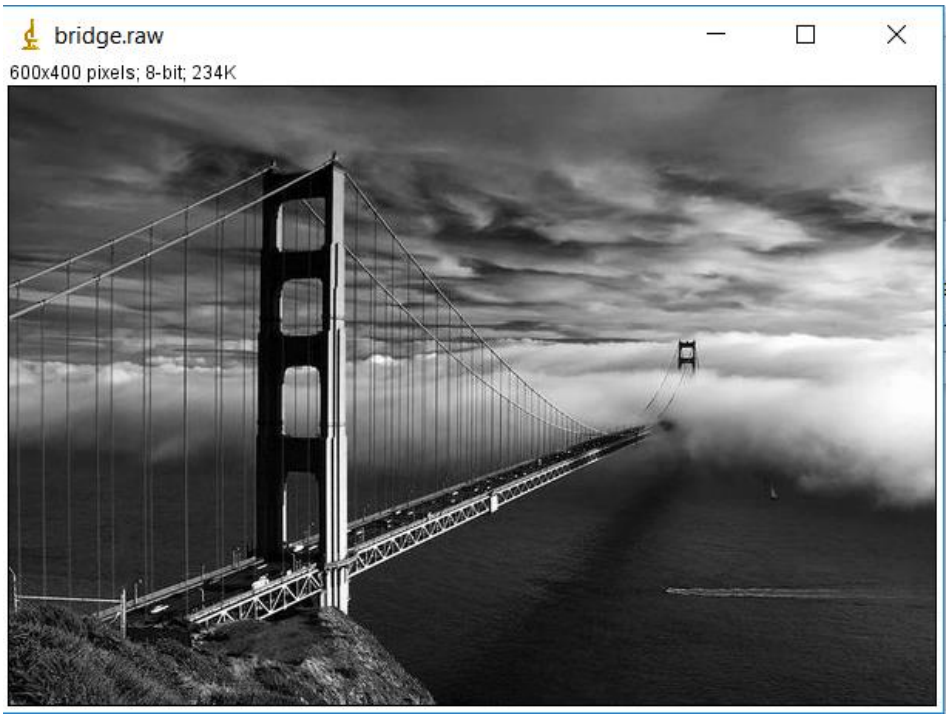


Figure 11: Original bridge.raw image

(i) Random Thresholding

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

Figure 12: Halftoning using Random Thresholding

- The image somewhat gives the impression of having few gray scale levels.
- Since random threshold value was used in any given location, it doesn't address any other properties of the image in thresholding.
- Random noise have been introduced because of random thresholding.
- The visual fidelity of binary image obtained is very poor.
- False contouring effects can be seen which means that the image is not shaded properly.
- The quantization error has not been taken care of. This results in a very low value of SNR in the output image.
- It is faster method with less memory.
- The root mean squared error between final and original image is 106.8634

(iii) Dithering

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

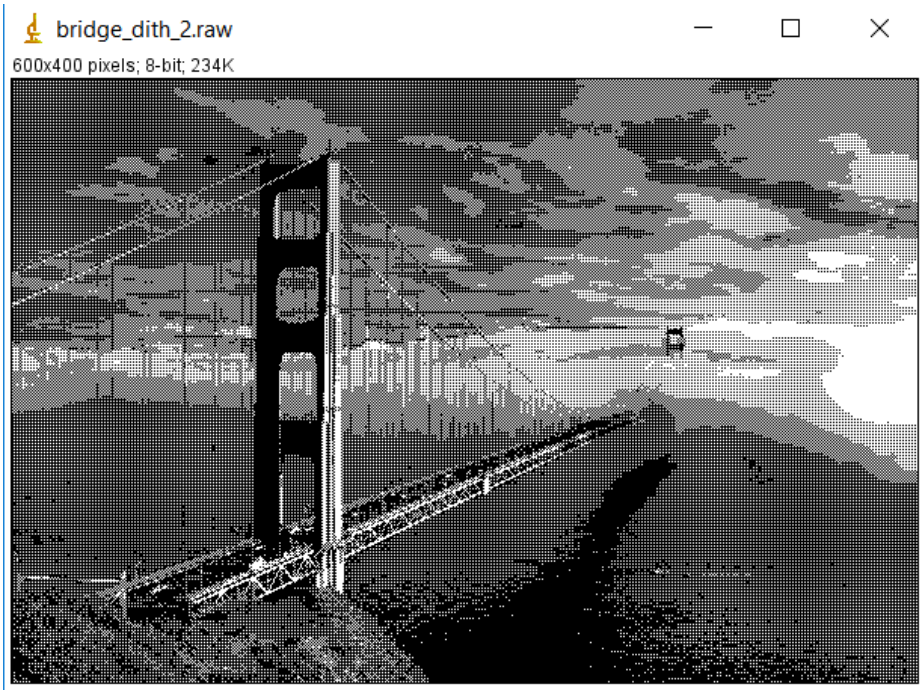


Figure 12: Output for dithering using I_2

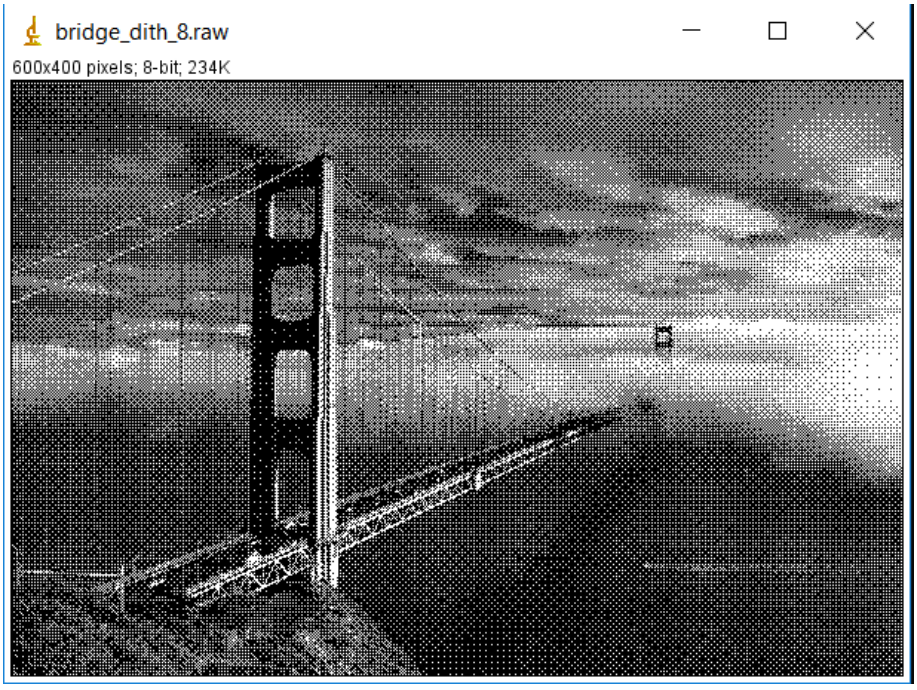


Figure 13: Output for dithering using I_8

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

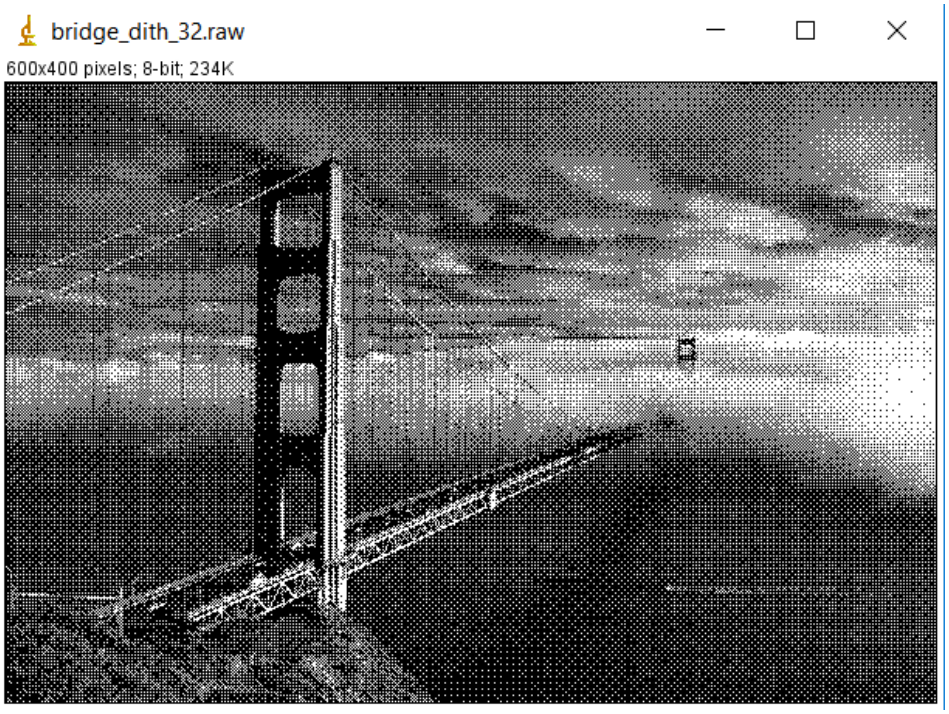


Figure 14: Output for dithering using I_{32}

- The visual fidelity is better in using I_8 and I_{32} than in using I_2 dithering matrix.
- I_2 gives fewer gray level impressions than I_8 while I_{32} has the best gray level impressions.
- False contour effect is reduced as compared to random thresholding because of variable threshold levels in each block decreasing the quantization error effect.
- Faster method
- This method yields finer amplitude quantization over larger area
- It retains good detail rendition over smaller area.
- For a continuous solid patch, this method creates image with noise holes in the centre because of variable thresholding in blocks.
- I_2 gives lot of square effect of the blocks compared to I_8 and I_{32}
- Transition between different gray level pixels are more visible.
- The root mean square error for I_2 is 108.6425, for I_8 is 107.1805 and I_{32} is 107.0031

(b) Error diffusion

(i) Floyd Steinberg

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

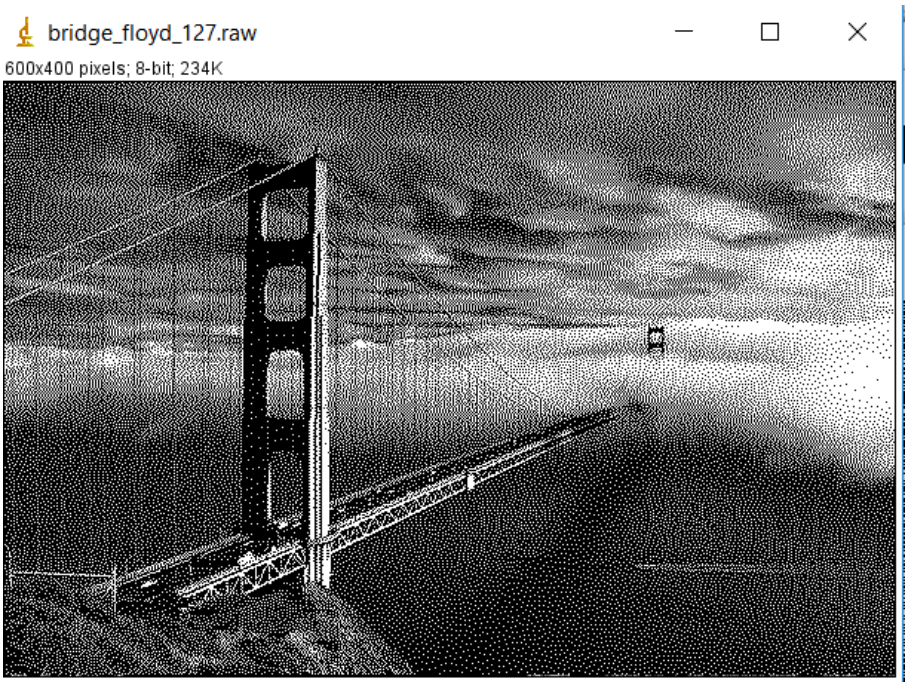


Figure 15: Output for error diffusion using Floyd Steinberg in serpentine parsing with threshold 127

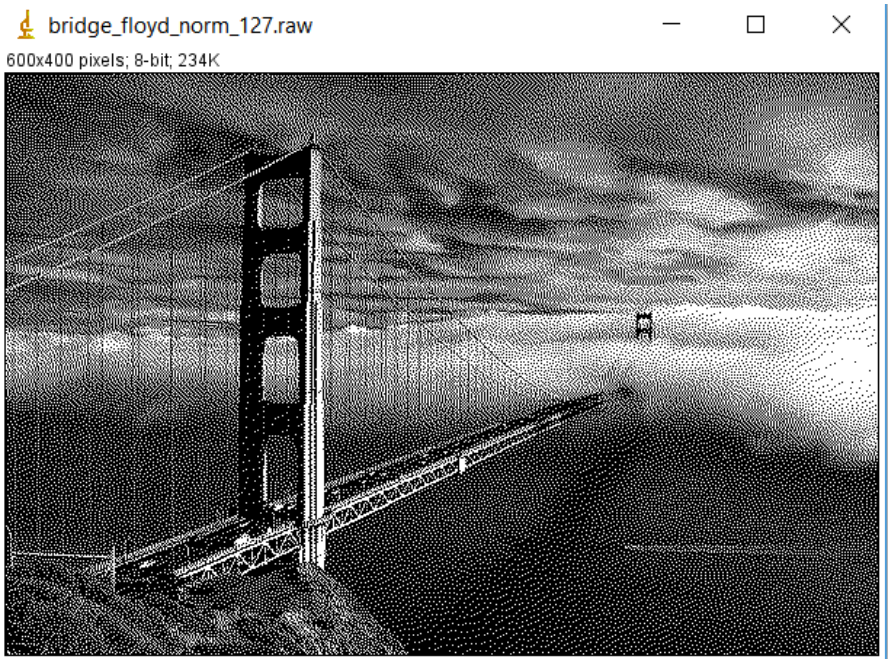


Figure 16: Output for error diffusion using Floyd Steinberg in normal parsing with threshold 127

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

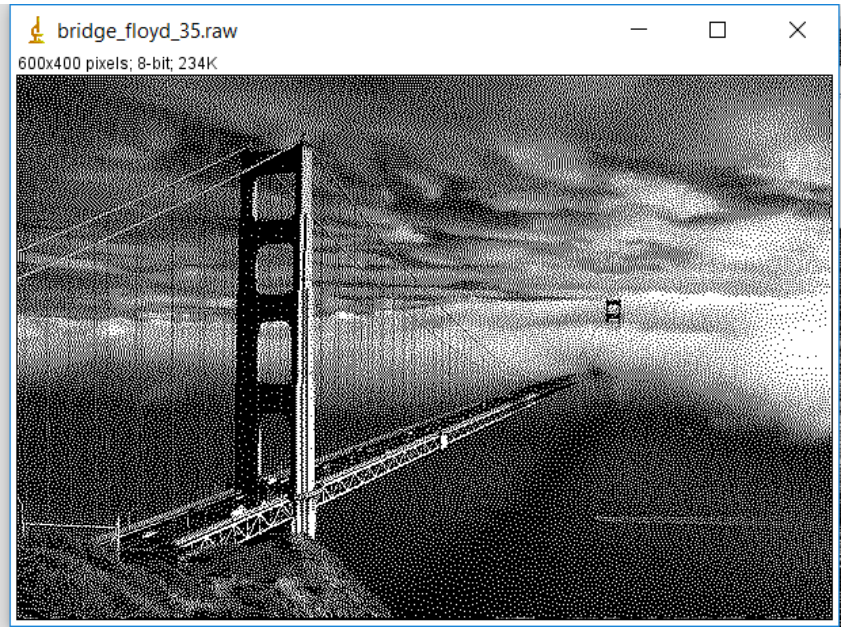


Figure 15: Output for error diffusion using Floyd Steinberg in serpentine parsing with threshold 45

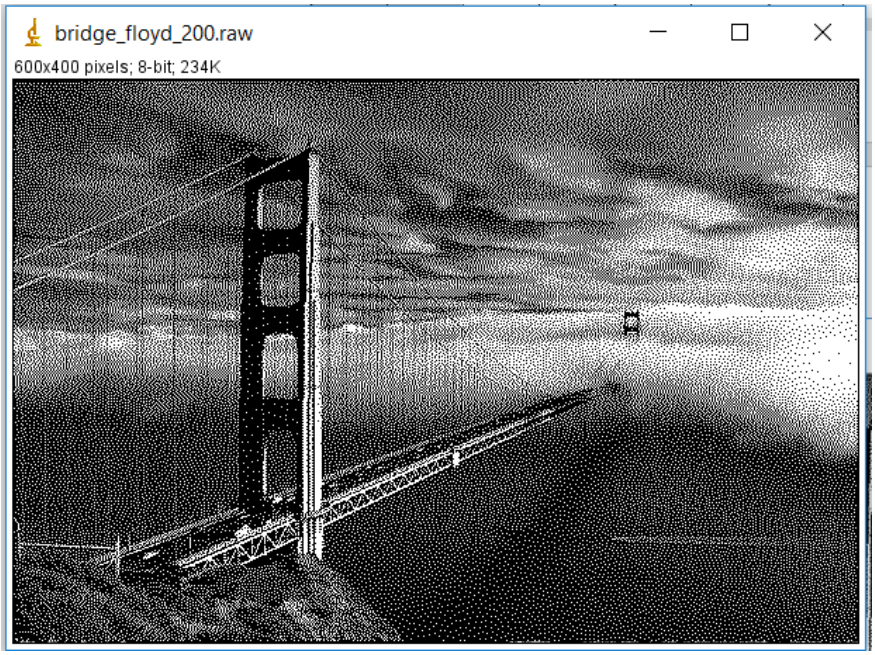


Figure 15: Output for error diffusion using Floyd Steinberg in serpentine parsing with threshold 205

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

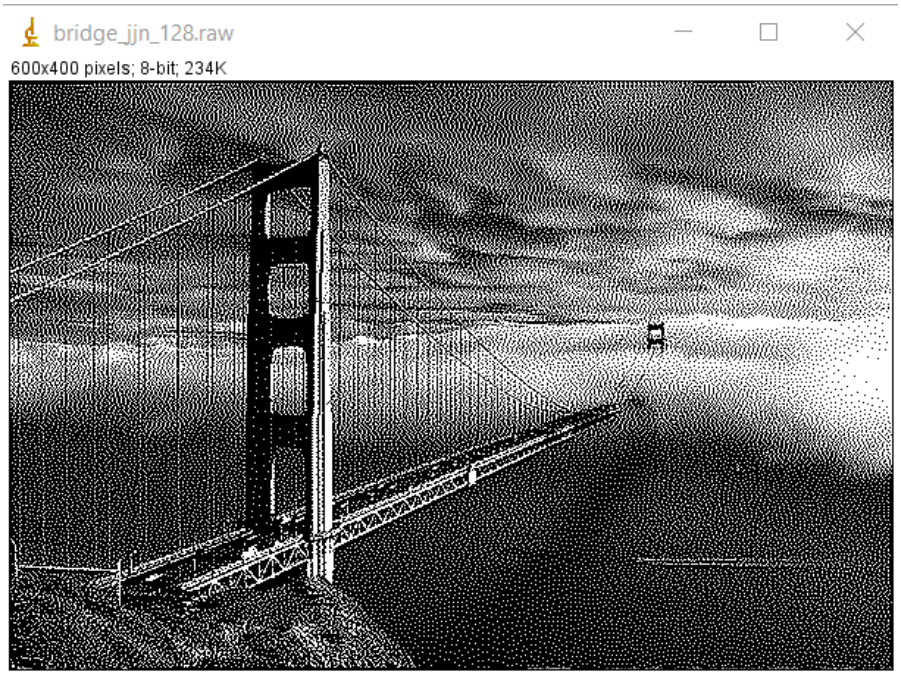


Figure 17: Output for error diffusion using JJN in serpentine parsing with threshold 128

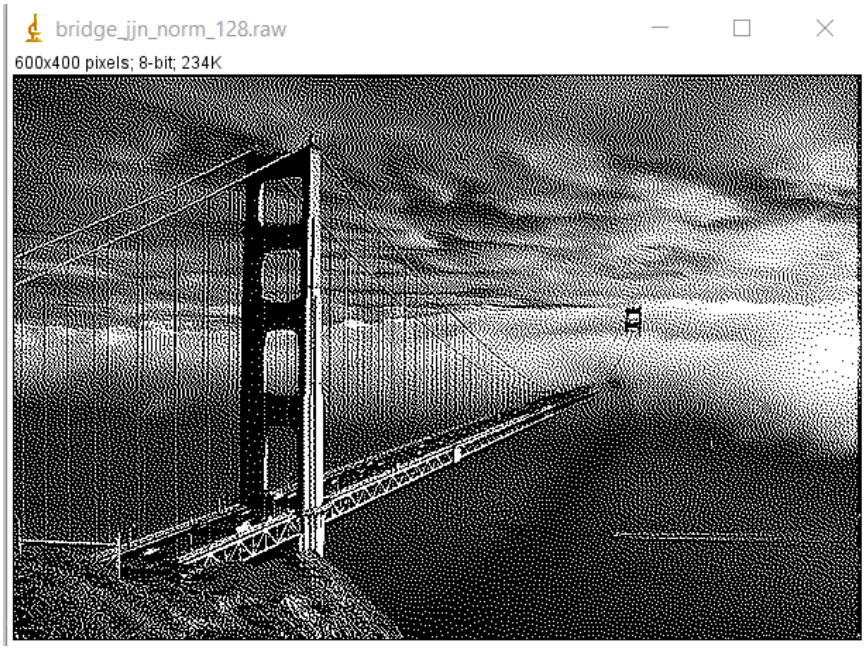


Figure 18: Output for error diffusion using Floyd Steinberg in normal parsing with threshold 127

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

Figure 17: Output for error diffusion using JJN in serpentine parsing with threshold 45

Figure 17: Output for error diffusion using JJN in serpentine parsing with threshold 205

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

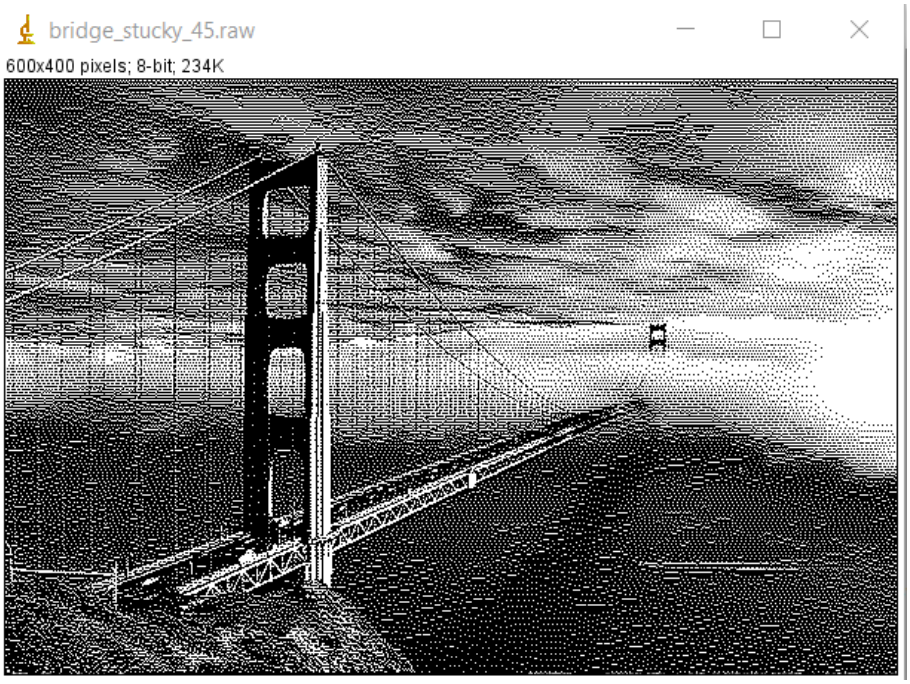


Figure 17: Output for error diffusion using Stucky in serpentine parsing with threshold 45

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

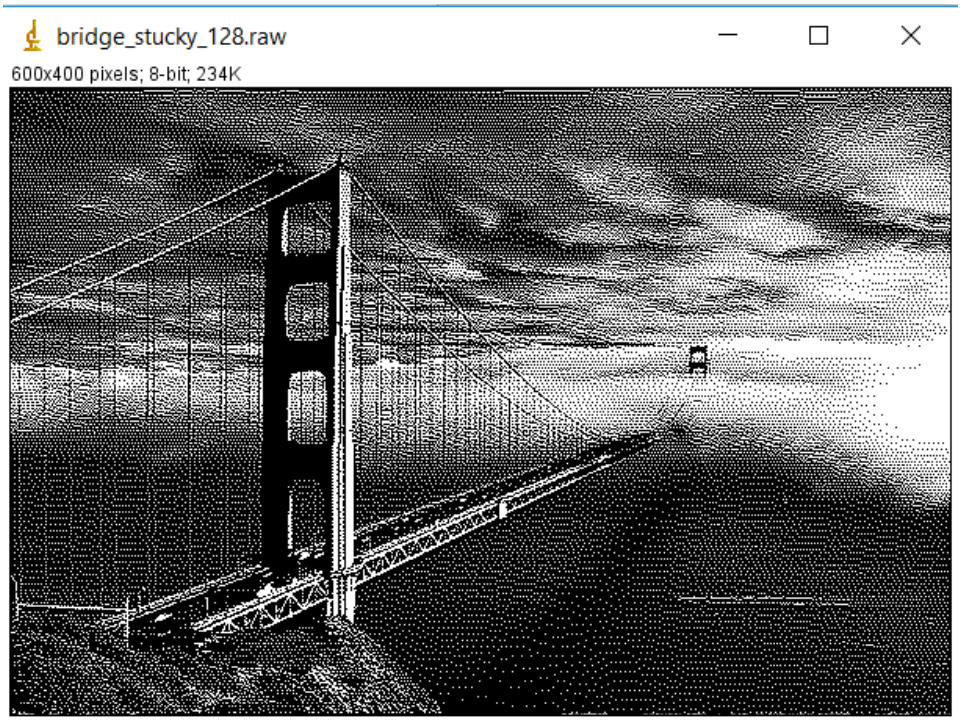


Figure 17: Output for error diffusion using Stucky in serpentine parsing with threshold 128

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

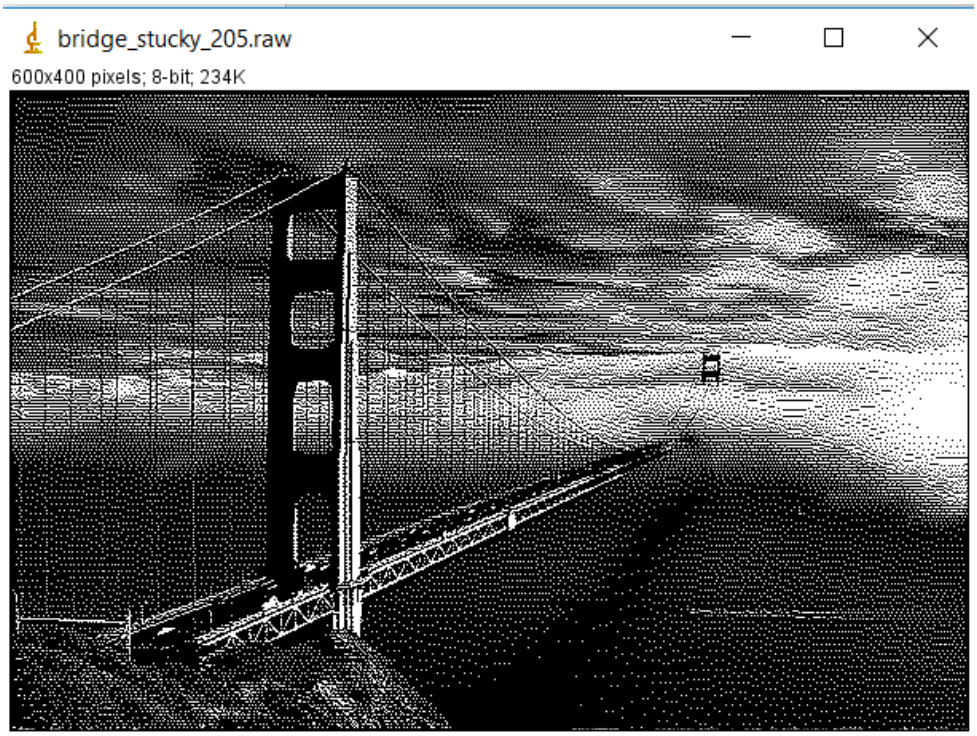


Figure 17: Output for error diffusion using Stucky in serpentine parsing with threshold 205

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

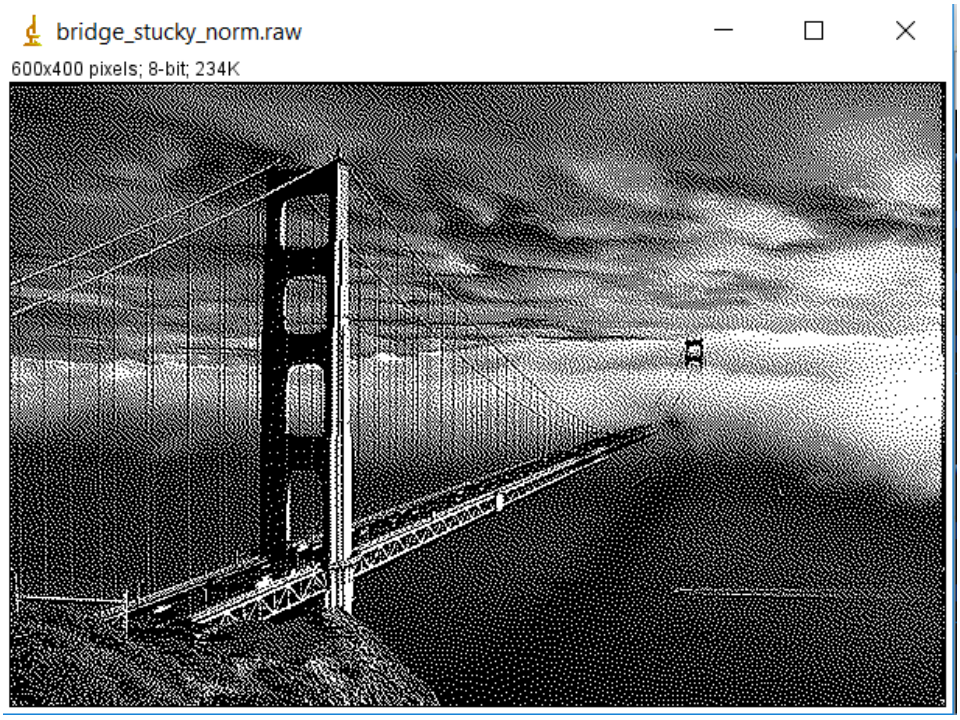


Figure 18: Output for error diffusion using Stucky in normal parsing with threshold 127

Discussion about outputs using three diffusion methods:

- Faster algorithm.
- As can be seen, serpentine parsing gives better result than normal parsing.
- Creates an impression of more gray scale levels for eyes.
- Because of error diffusion, the solid patch which could have been represented as complete black has few white dots in between (as seen on the bridge pillar).
- For thresholds of 45, 128, 205 there is a contrast variance in representation of pixels. As can be seen, 128 works well for all three methods.

Comparison between the three diffusion methods:

- All three method gives decent impression of gray scale levels.
- There is subtle difference in false contouring. JJN and Stucky has better contours for bridge ropes than Floyd Steinberg method
- Floyd Steinberg gives fine drained dithering.
- Performance wise output of Stucky is cleaner and sharper compared to JJN which is better than Floyd Steinberg.
- JJN is slower than Floyd Steinberg but Stucky is slightly faster than JJN.
- The RMSE value for Floyd is 105.9032, JJN is 104.4392, Stucky is 98.3284 for threshold value of 128

Comparison between error diffusion and dithering:

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

- Both are faster.
- Unlike dithering, error diffusion doesn't produce **square box** effects.
- Error diffusion has better **contrast performance** than dithering.
- Error diffusion produces **worm effects** which is not seen in dithering.

Own idea:

- Can create adaptive error diffusion with localizing a particular block patches and use different error diffusion values depending on high frequency/ low frequency regions.

(c) Color halftoning

(i) Separable error diffusion

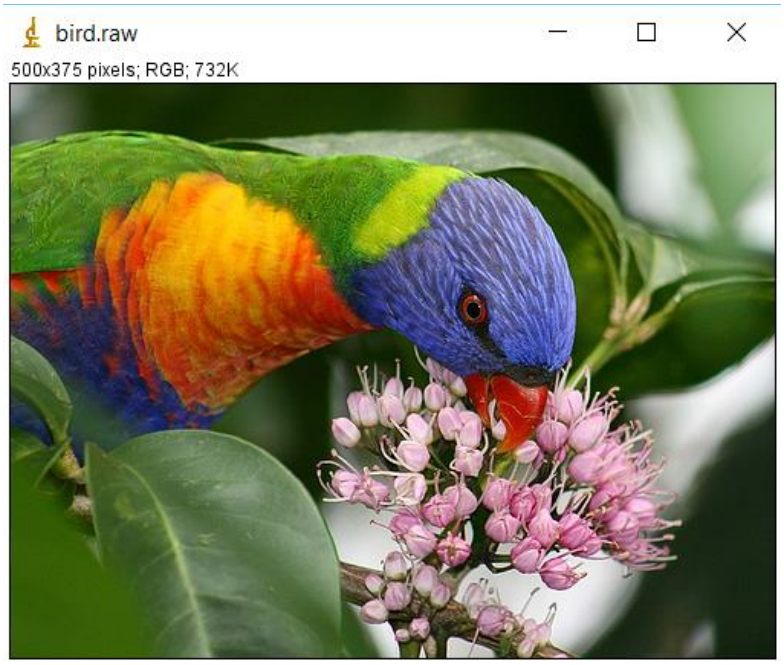


Figure 12: Original bird.raw image

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

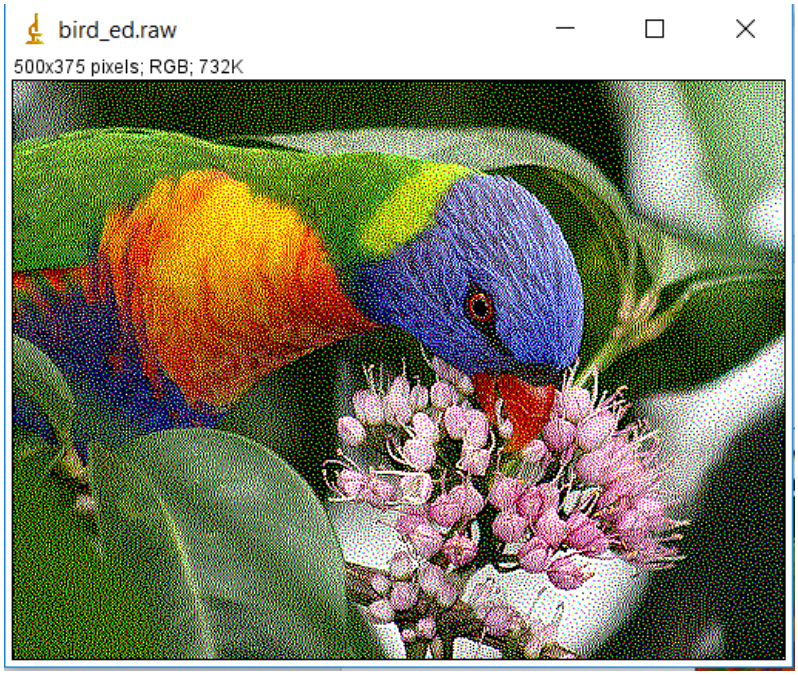


Figure 13: Output for Separable error diffusion

Discussion:

- The placement pattern in visually unnoticeable.
- As can be seen, the local average color is the desired color.
- The generalization of monochrome algorithm overlooks the fact that colored dots are not equally bright.
- The main drawback is the colors used do not reduce the notice-ability of the pattern.

(ii) MBVQ

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

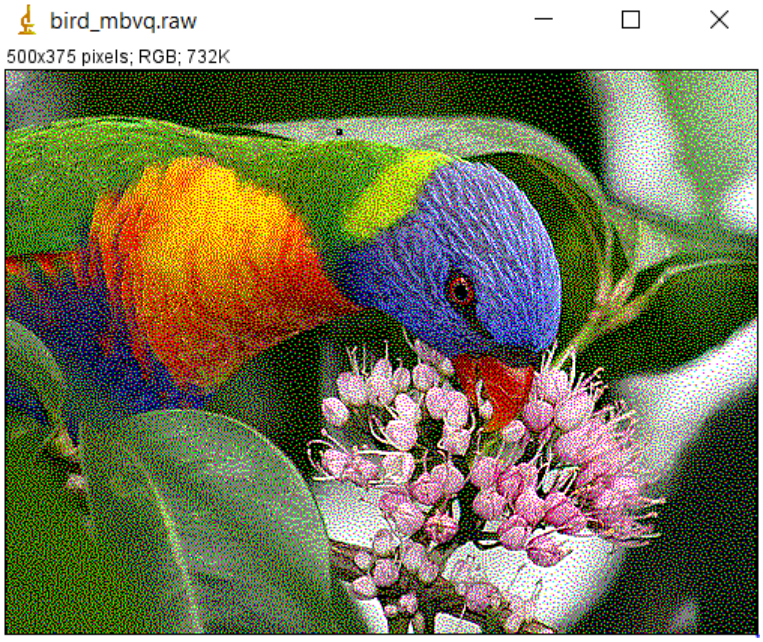


Figure 14: Output for MBVQ

- The placement pattern in visually unnoticeable.
- As can be seen, the local average color is the desired color.
- The drawback of Separable Error Diffusion of reducing the notice-ability of the pattern is met in MBVQ method.

The main idea and algorithm of MBVQ is explained in abstract and motivation part of MBVQ.

Akash Mohan Das
USC Id: 7247503828
USC email: amohanda@usc.edu
Submission Date: 02/12/2019

References:

1. Wikipedia
2. Lecture Notes
3. Research Papers
4. Tutorial notes on Halftoning techniques on web