COGS 260: Assignment 1

Ankur Jain, A53097130, anj022@ucsd.edu

***Abstract*—Edge detection has been fundamental tool in image processing and computer vision. The purpose of detecting sharp changes in image brightness is to capture important events and changes in properties of the world. In this paper, we present a comparative study between a variety of available edge detection methods. This paper also touches on comparison of pre-existing image smoothing and noise removal methods and effect of filter size.**

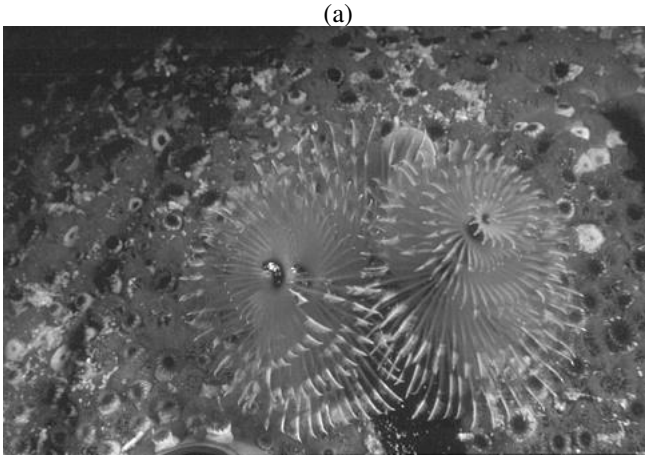
***Index Terms*—De-noising, Edge Detection, Histogram Equalization, Gaussian Filter, Image Processing**

I. Basic Image Operation

A. Image Read/write

We used the inbuilt read function in MATLAB to read the image 12084.jpg, and the inbuilt function was used to convert the image from RGB to grayscale. The results are shown in fig 1. Corresponding code can be found in listing 2

(a)



(b)

Figure 1. (a) Original Image (b) Corresponding grayscale image

B. Image Smoothing

Image smoothing is useful technique for removing noise. It actually removes high frequency content (e.g: noise, edges) from the image resulting in edges being blurred when this is filter is applied. In this paper we study gaussian and average smoothing.

1) Gaussian Smoothing: The Gaussian blur is a type of image-blurring filter that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. The equation of a Gaussian function in two dimension is describe by eq 1.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

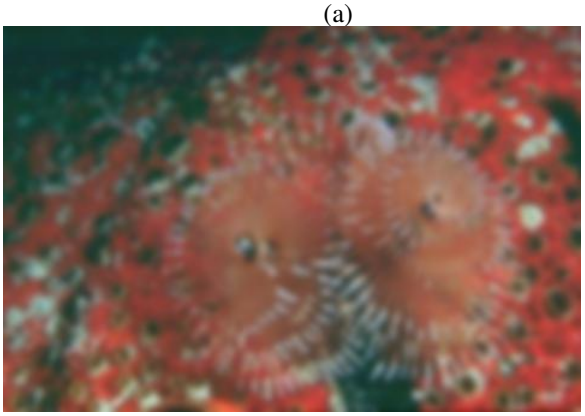
where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution. When applied in two dimensions, this formula produces a surface whose contours are concentric circles with a Gaussian distribution from the center point. Values from this distribution are used to build a convolution matrix which is applied to the original image. Each pixel's new value is set to a weighted average of that pixel's neighborhood. The original pixel's value receives the heaviest weight (having the highest Gaussian value) and neighboring pixels receive smaller weights as their distance to the original pixel increases. This results in a blur that preserves boundaries and edges better than other, more uniform blurring filters.

We conducted experiments where kernel width was varied from 3 to 7. The results are shown in fig 2, 4, 6

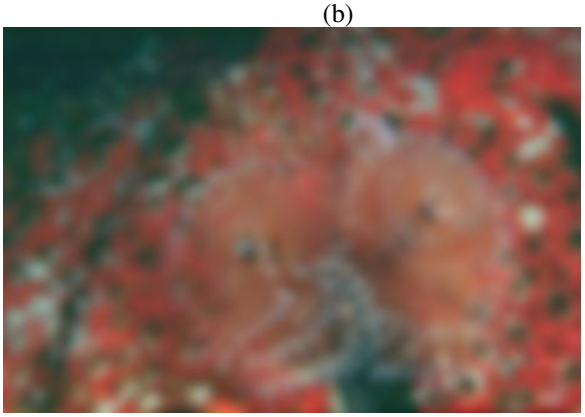
2) Average Smoothing: This method replaces each pixel with a weighted average of neighborhood. In our case we have assumed all weights to be equal. We conducted experiments where kernel width was varied from 3 to 7. The results are shown in fig 3, 5, 7

The results for gaussian smoothing and average smoothing are placed side-by-side for direct visual comparison and it shows that the degree of smoothing increases with the size of the kernel due to higher area of averaging of a pixel with its neighboring pixels. The results of gaussian smoothing are more realistic and approximate more closely the real-world effect of shooting a subject with an out-of-focus lens, whereas average smoothing gives a poor smoothing result for a smaller window size. If we repeat average smoothing 3-4 times, the results seems to be comparable to gaussian smoothing. (The results for later experiments are not shown in this paper). Code can be found in listing 3

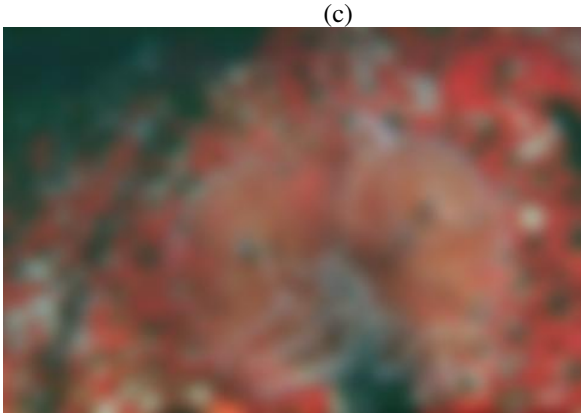
(a)



(b)



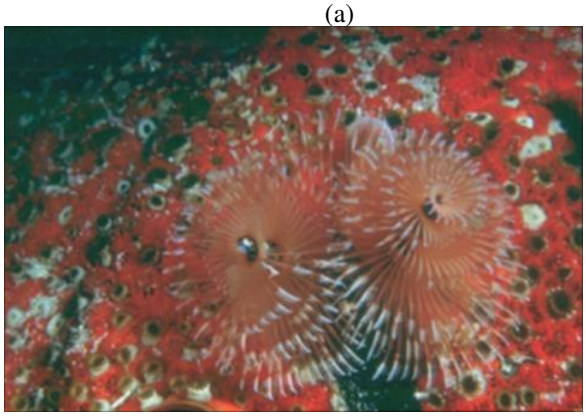
(c)



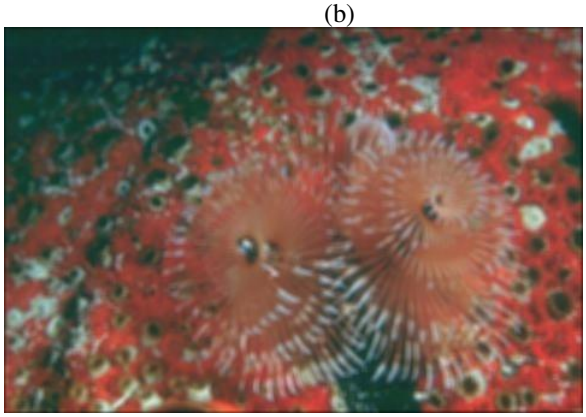
(d)

Figure 2. Smoothing (a) Original Image (b) Gaussian smoothed image(Filter Size=3) (c) Gaussian smoothed image(Filter Size=5) (d) Gaussian smoothed image(Filter Size=7)

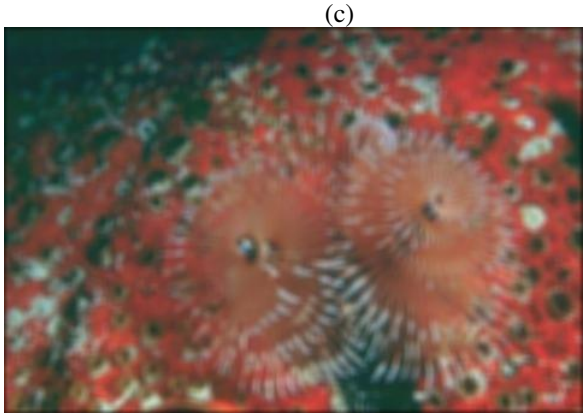
(a)



(b)



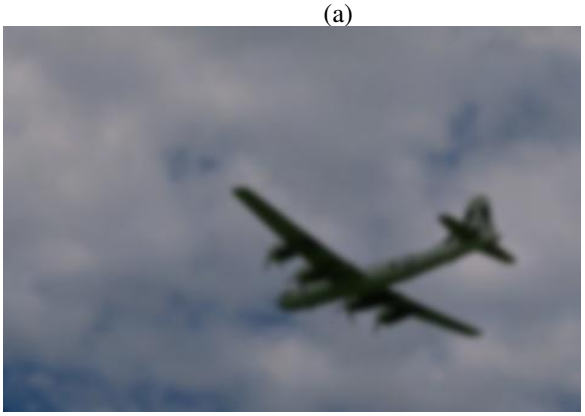
(c)



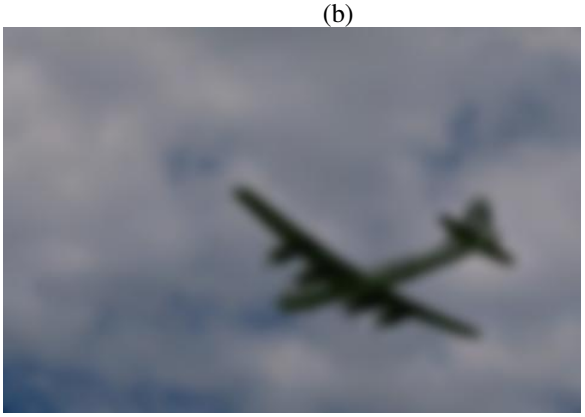
(d)

Figure 3. Smoothing (a) Original Image (b) Average smoothed image(Filter Size=3) (c) Average smoothed image(Filter Size=5) (d) Average smoothed image(Filter Size=7)

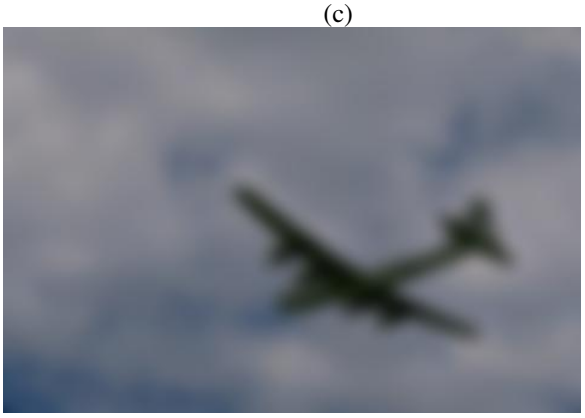
(a)



(b)



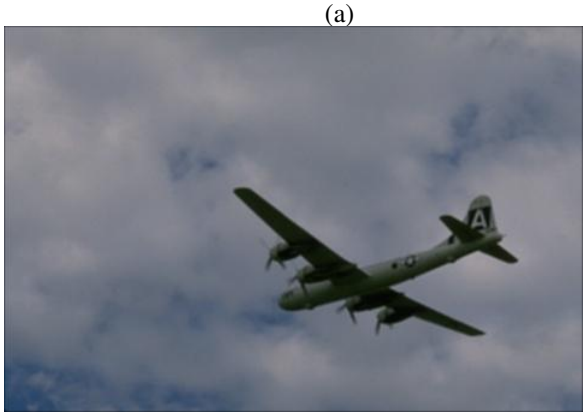
(c)



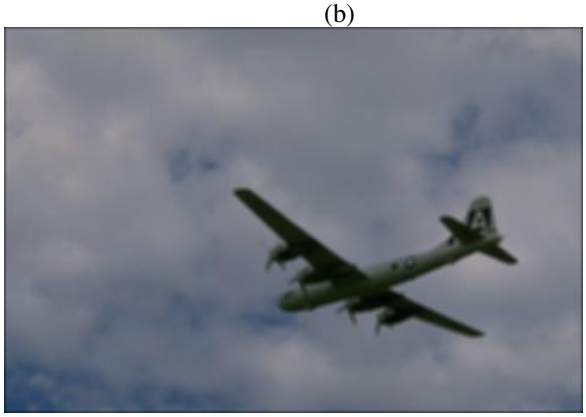
(d)

Figure 4. Smoothing (a) Original Image (b) Gaussian smoothed image(Filter Size=3) (c) Gaussian smoothed image(Filter Size=5) (d) Gaussian smoothed image(Filter Size=7)

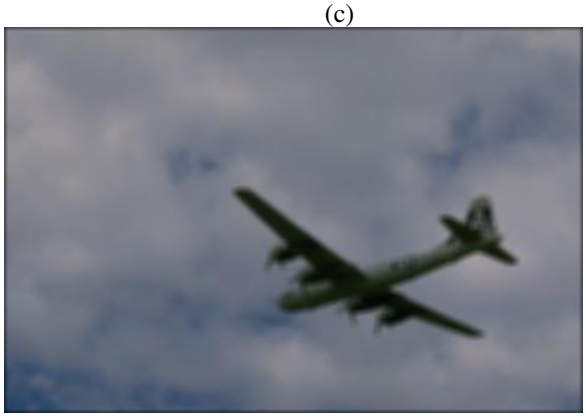
(a)



(b)



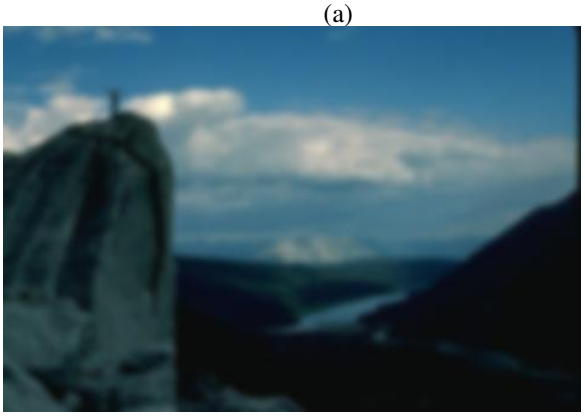
(c)



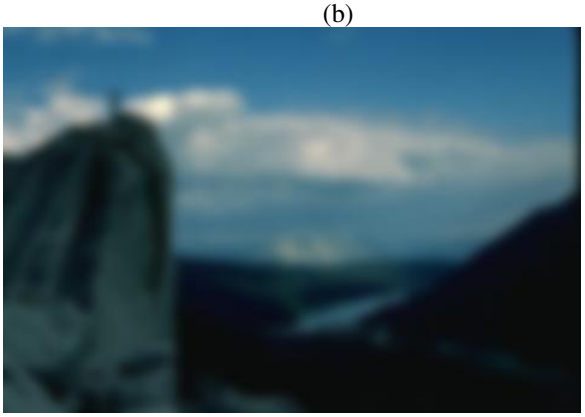
(d)

Figure 5. Smoothing (a) Original Image (b) Average smoothed image(Filter Size=3) (c) Average smoothed image(Filter Size=5) (d) Average smoothed image(Filter Size=7)

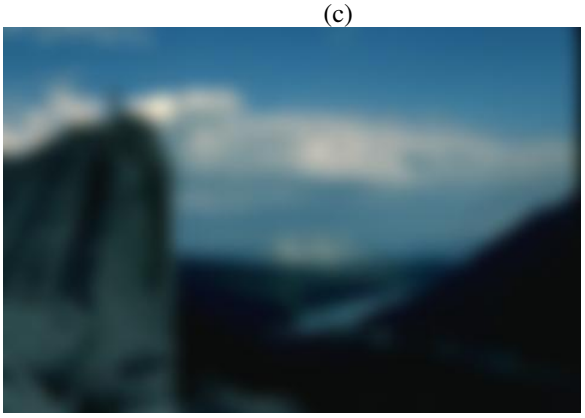
(a)



(b)



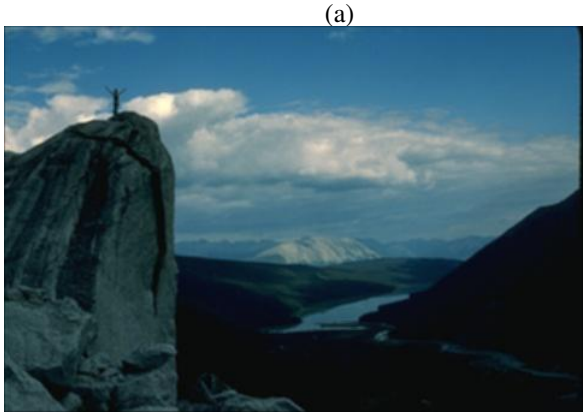
(c)



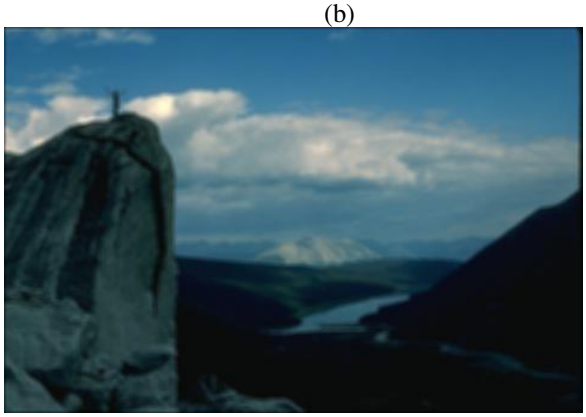
(d)

Figure 6. Smoothing (a) Original Image (b) Gaussian smoothed image(Filter Size=3) (c) Gaussian smoothed image(Filter Size=5) (d) Gaussian smoothed image(Filter Size=7)

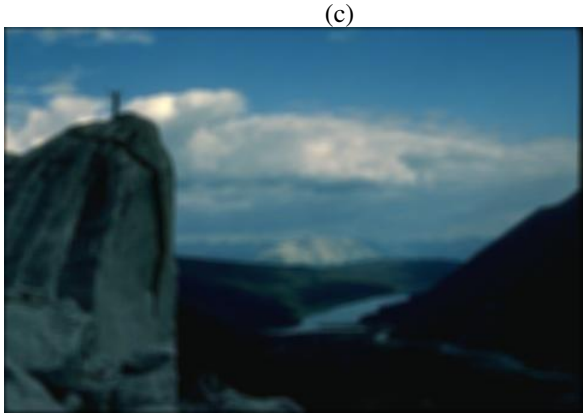
(a)



(b)



(c)



(d)

Figure 7. Smoothing (a) Original Image (b) Average smoothed image(Filter Size=3) (c) Average smoothed image(Filter Size=5) (d) Average smoothed image(Filter Size=7)

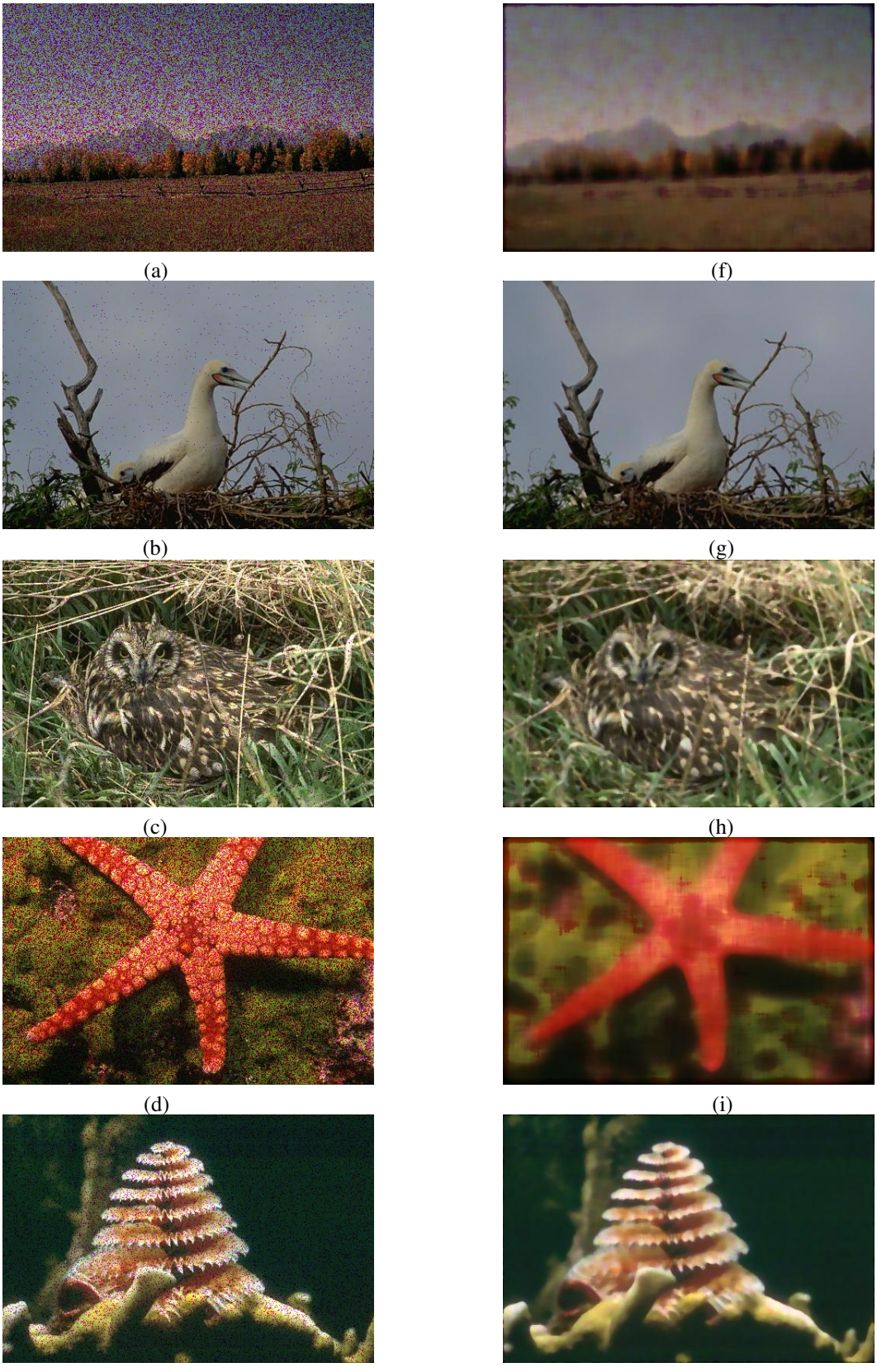


Figure 8. Median Filtering (a-e) Original Image with salt and pepper noise (f-j) Median Filtered Images with varying filter size, 14*14, 3*3, 5*5, 20*20, 7*7 in-order

C. Median Filtering

Each output pixel contains the median value in the M-by-N neighborhood around the corresponding pixel in the input image. This is highly effective in removing salt-and-pepper noise. In the Gaussian and average filters, the filtered value for the central element can be a value which may not exist in the original image. However this is not the case in median filtering, since the central element is always replaced by some pixel value in the image. This reduces the noise effectively. The kernel size must be a positive odd integer. The results of median filtering are shown in fig 8 (a-e shows original image and f-j shows the filtered images). Note that filter size was different for each image due to fact that amount of noise in each image varied significantly. It should also be noted that due to a very high noise in 8 (a) and 8 (d) the recovered image is not very clear. Code can be found in listing 4

II. Image Enhancement

A. Histogram

The plot for histogram of each channel of 3 images is shown in fig 10, 11, 12.

In fig 10 it can be seen that the image is primarily red in color and histogram for red channel dominates over all other color. Similarly in fig 11 the primary color of the image seems to be white, hence the histogram of red, green and blue channels seems equally distributed. In fig 12 we can see that the image has a lot of dark component in the bottom half and a visible blue sky in the upper half and we can see blue dominating in the histogram as well. The results are shown in fig 10, 10, 12 where (a) shows the original image and (b-d) show the histogram for each of the channel. Code can be found in listing 5

B. Histogram Equalization

Consider an image whose pixel values are confined to some specific range of values only. For eg, brighter image will have all pixels confined to high values. But a better image will have pixels from all regions of the image. So we need to stretch this histogram to either ends which is referred to as Histogram Equalization. This normally improves the contrast of the image.

Histogram Equalization calculation is based in cumulative distribution function corresponding to the probability of an occurrence of a pixel of level i in the image.

$$p_x(i) = p(x = i) = \frac{n_i}{n}, 0 \leq i < L \tag{2}$$

L being the total number of gray levels in the image (typically 256), n being the total number of pixels in the image. In order to map the values back into their original range, the following simple transformation needs to be applied on the result:

$$y' = y.(max\{x\} - min\{x\}) + min\{x\} \tag{3}$$

The above describes histogram equalization on a grayscale image. However it can also be used on color images by applying the same method separately to the Red, Green and Blue components of the RGB color values of the image. Note: however using this method may yield dramatic changes in the image's color balance since the relative distributions of the color channels change as a result of applying the algorithm. Fig 9 shows the transformation of original condensed histogram to a relatively uniformly distributed histogram.

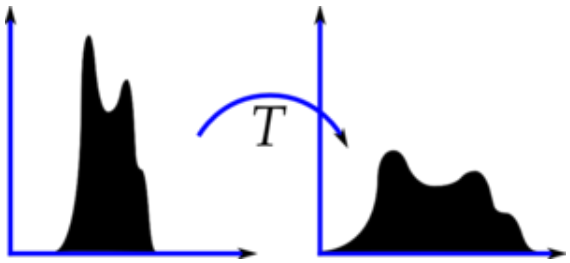


Figure 9. Histogram Equalization [2]

We have used inbuilt MATLAB function to produce histogram equalization for each channel. It can be seen that now all the values in histogram are approximately uniformly distributed.

The results are shown in fig 13, 14, 15 where (a) shows the original image, (b) shows the histogram equalized image and (c-e) show the histogram for each of the channel. Code can be found in listing 6

C. Adaptive Histogram Equalization (CLAHE)

Adaptive Histogram Equalization (CLAHE) is very similar to the normal histogram equalization. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image. In our experiments we divided the image into 16*16 tiles, limited the contrast to 0.005 and set the histogram distribution to uniform.

The results are shown in fig 16, 17, 18 where (a) shows the original image, (b) shows the adaptive histogram equalized image and (c-e) show the histogram for each of the channel. Code can be found in listing 7

III. Edge Detection

A. Sobel Edge Detection

The Sobel operator [4] is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. The directional filter G_x and G_y are as below

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 2 \end{bmatrix}$$

and

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

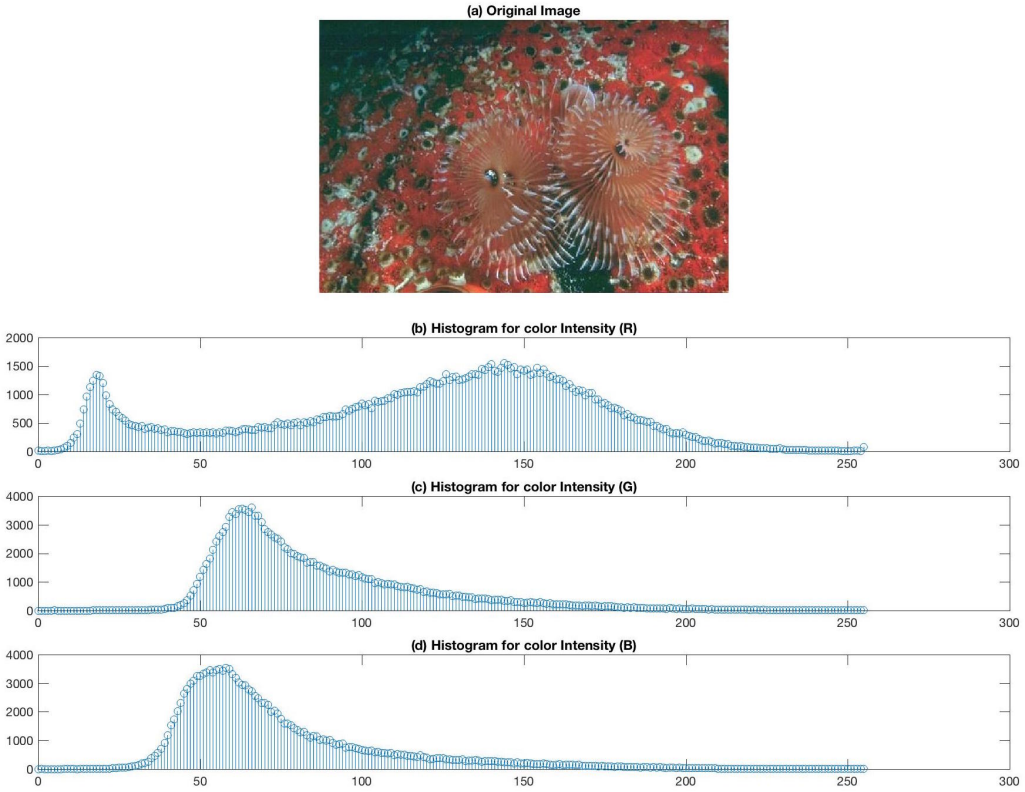


Figure 10. (a) Original Image (b) Histogram for R channel (c) Histogram for G channel (d) Histogram for B channel

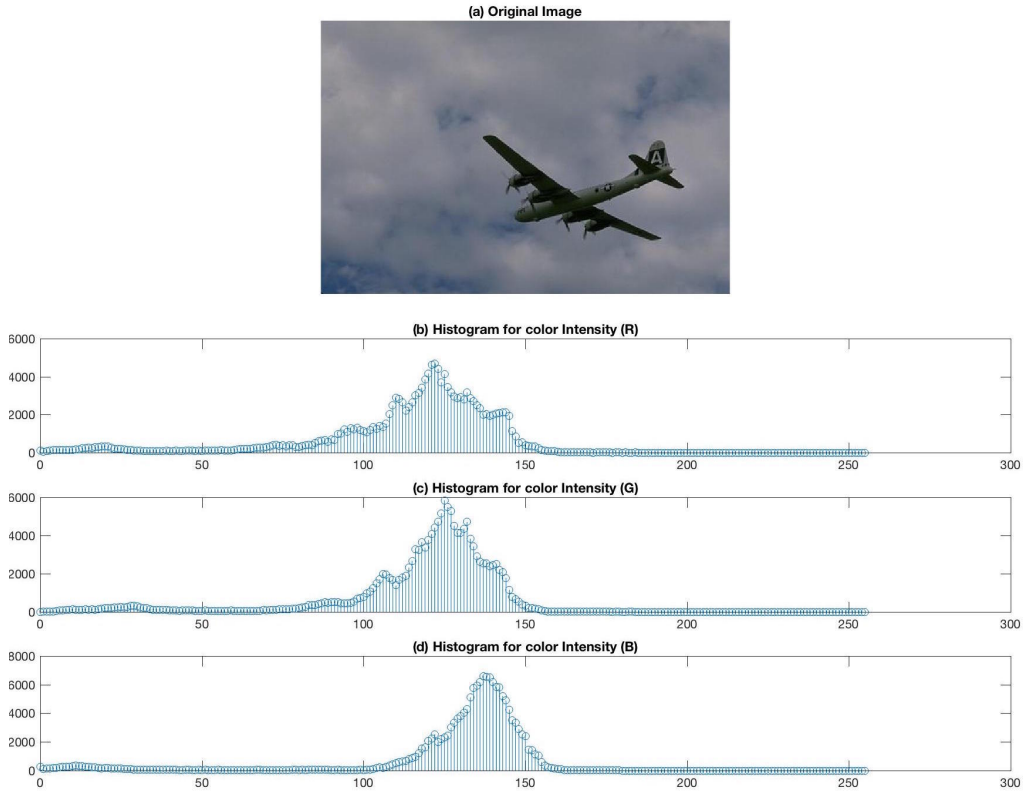


Figure 11. (a) Original Image (b) Histogram for R channel (b) Histogram for G channel (c) Histogram for B channel

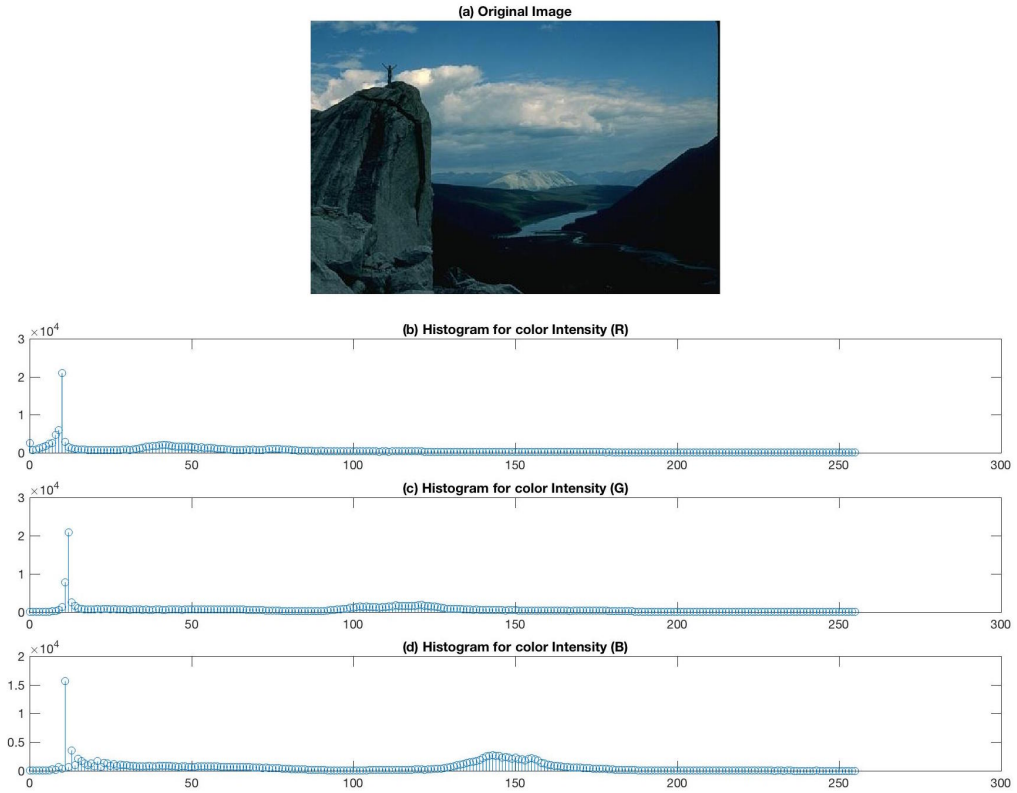


Figure 12. (a) Original Image (b) Histogram for R channel (b) Histogram for G channel (c) Histogram for B channel

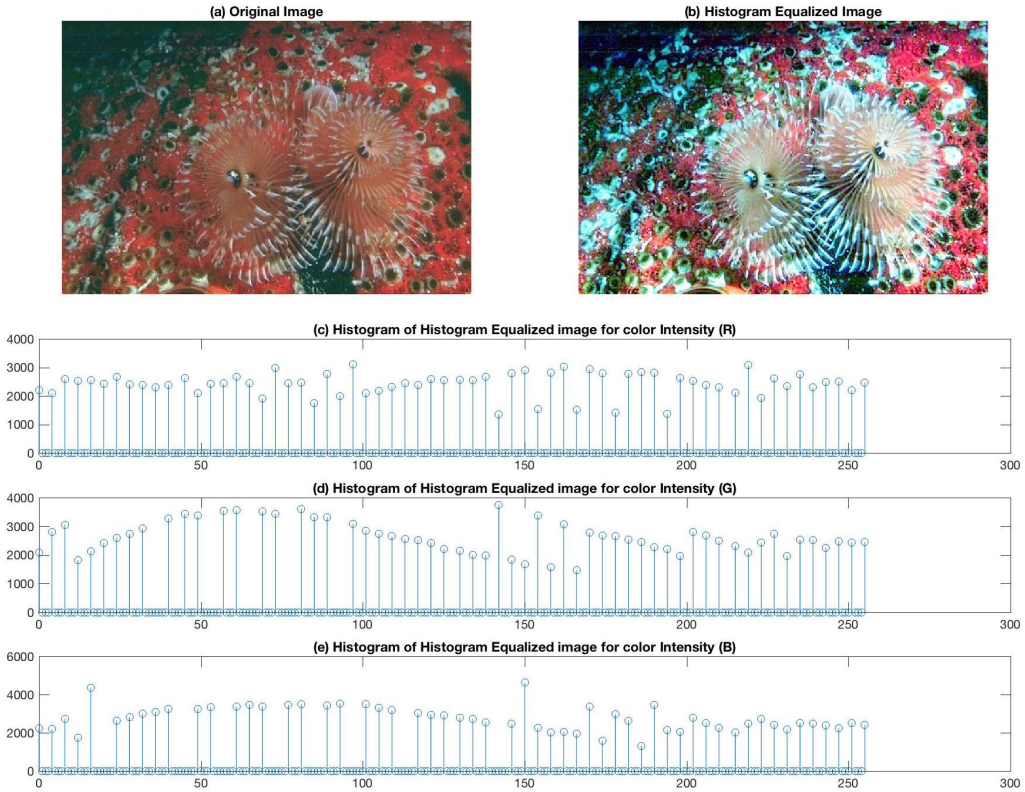


Figure 13. (a) Original Image (b) Histogram Equalized Image (c) Histogram for R channel (d) Histogram for G channel (e) Histogram for B channel

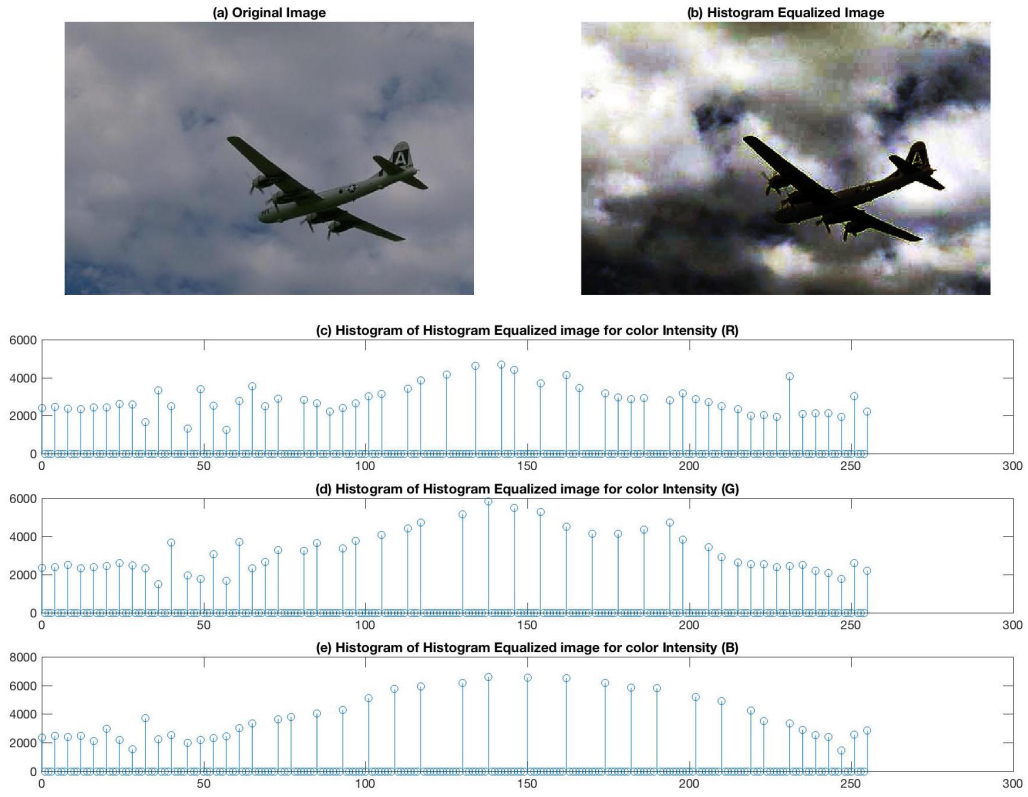


Figure 14. (a) Original Image (b) Histogram Equalized Image (c) Histogram for R channel (d) Histogram for G channel (e) Histogram for B channel

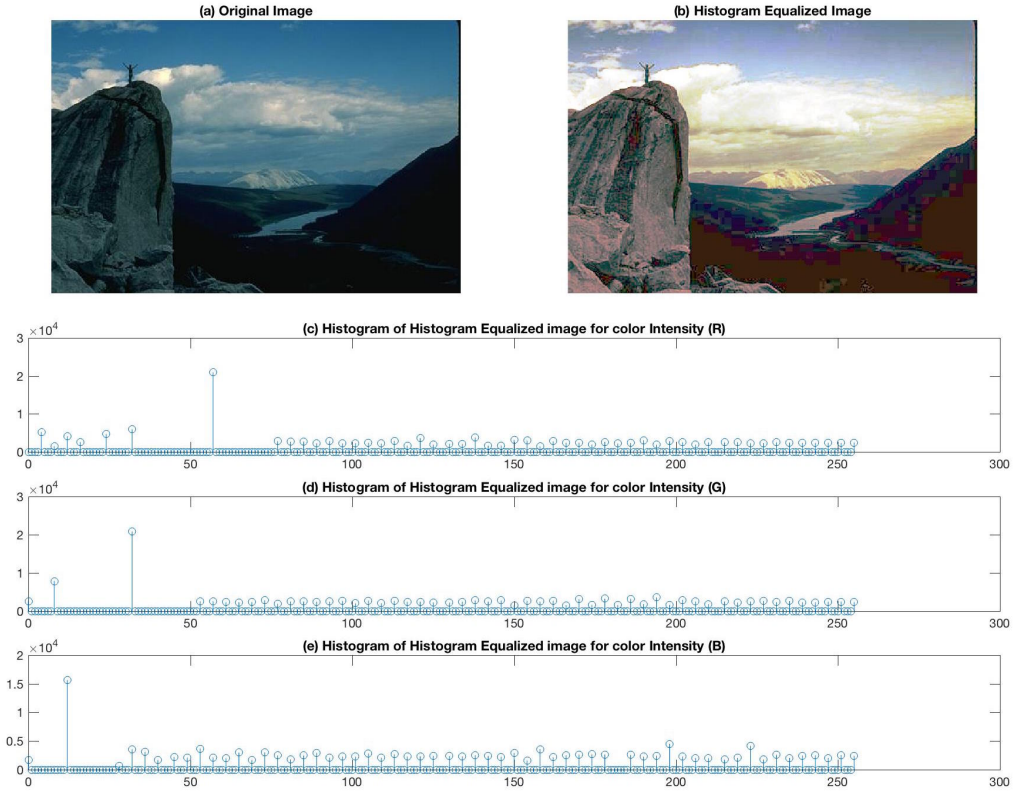


Figure 15. (a) Original Image (b) Histogram Equalized Image (c) Histogram for R channel (d) Histogram for G channel (e) Histogram for B channel

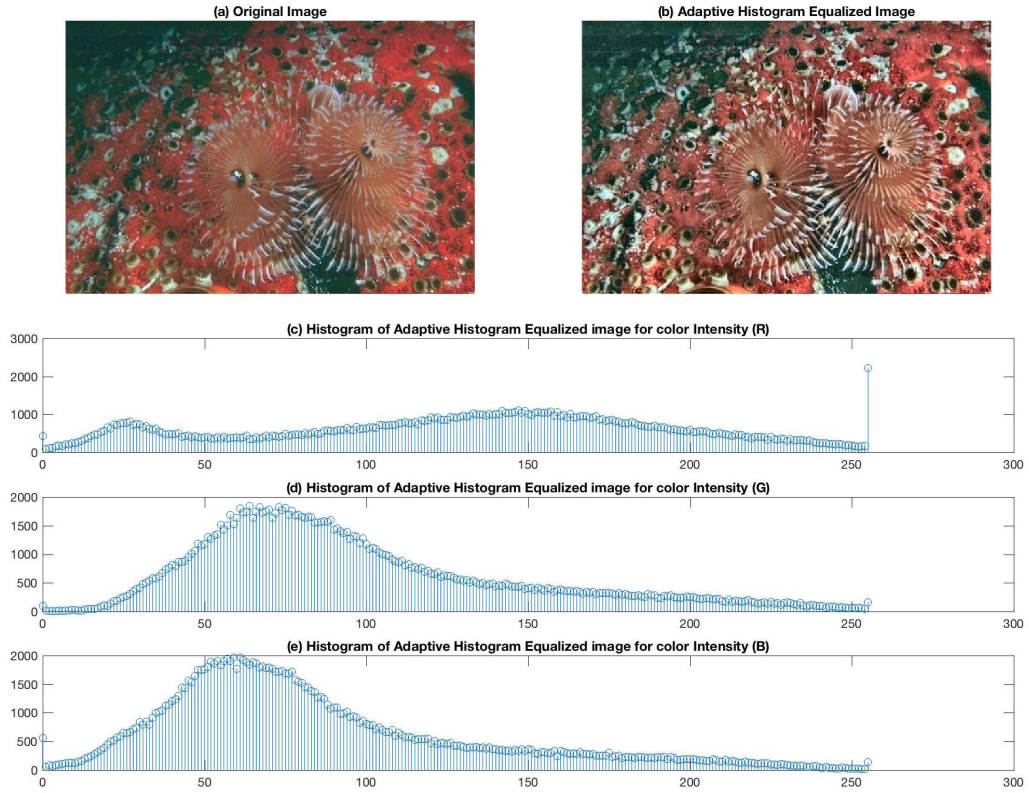


Figure 16. (a) Original Image (b) Adaptive Histogram Equalized Image (c) Histogram for R channel (d) Histogram for G channel (e) Histogram for B channel

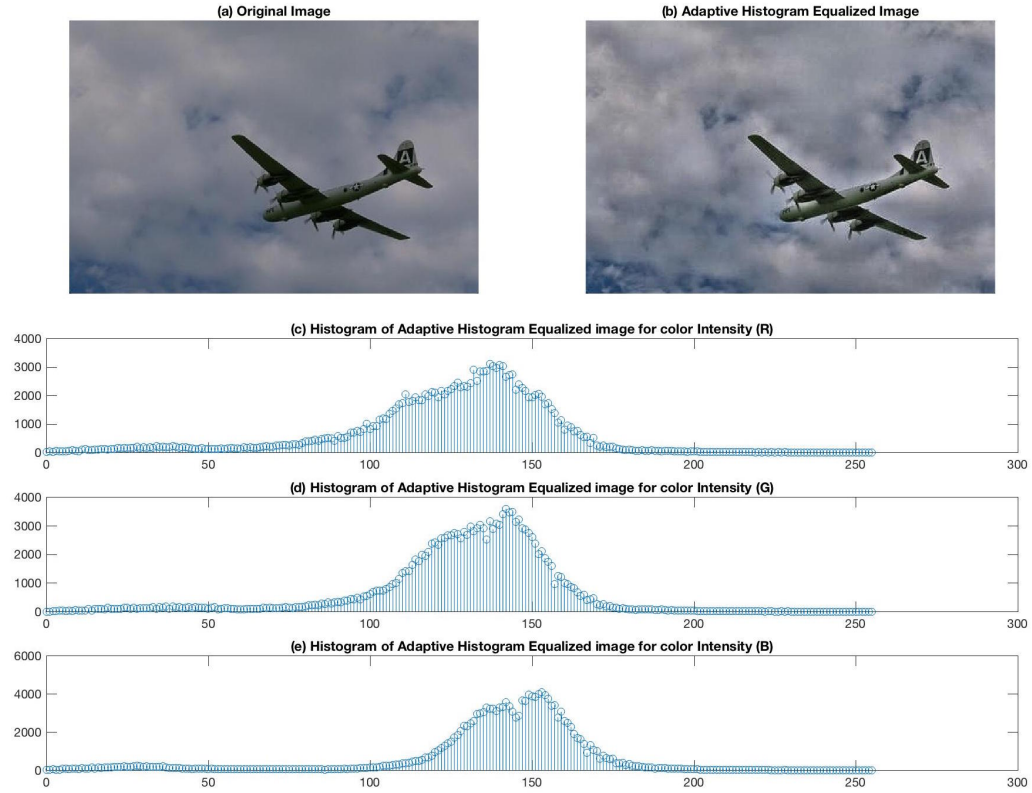


Figure 17. (a) Original Image (b) Adaptive Histogram Equalized Image (c) Histogram for R channel (d) Histogram for G channel (e) Histogram for B channel

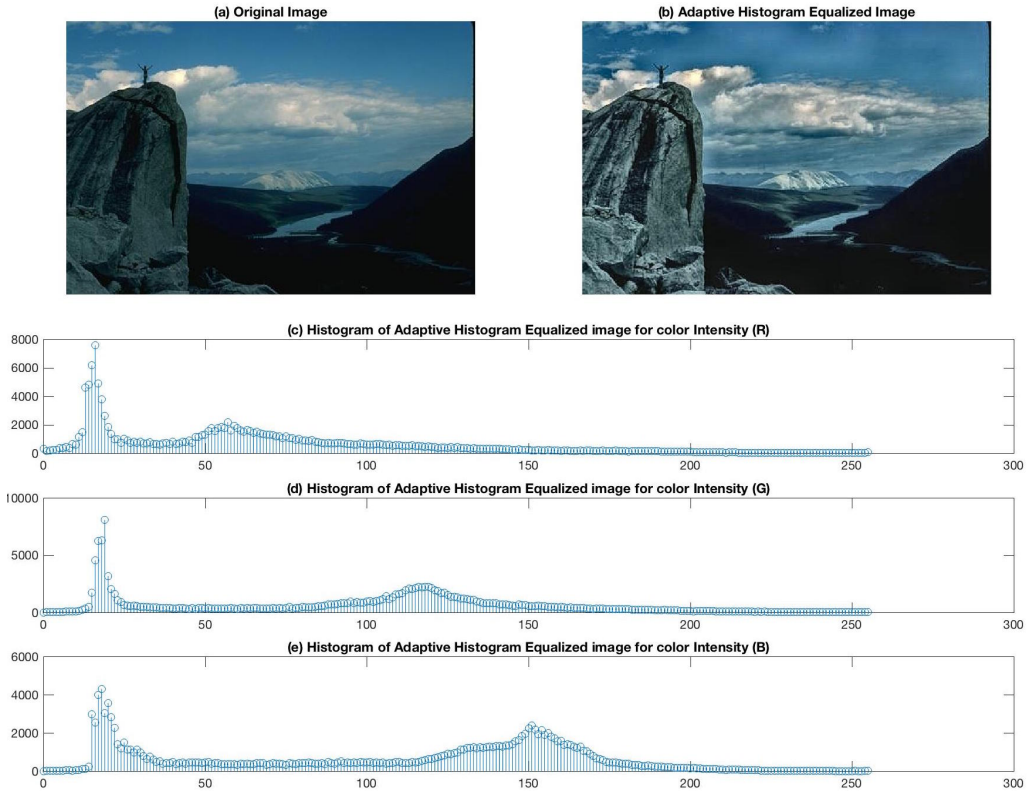


Figure 18. (a) Original Image (b) Adaptive Histogram Equalized Image (c) Histogram for R channel (d) Histogram for G channel (e) Histogram for B channel

Directional images can be computed as $I_x = G_x.conv.A$ $I_y = G_y.conv.A$ where conv represent the convolution and A represents the input image. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using

$$I = \sqrt{I_x^2 + I_y^2}$$

B. Canny Edge Detection

The Process of Canny edge detection algorithm can be broken down to 5 different steps: [1, 3]

1) Apply Gaussian filter to smooth the image in order to remove the noise
2) Find the intensity gradients of the image
3) Apply non-maximum suppression to get rid of spurious response to edge detection
4) Apply double threshold to determine potential edges
5) Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

C. Prewitt Edge Detection

Prewitt operator [5] calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how "abruptly" or "smoothly" the image changes at that point, and therefore how likely it is that part of the image represents an edge, as well as how that edge is likely to be oriented. In practice, the magnitude (likelihood of an edge) calculation is more reliable and easier to interpret than the direction calculation.

The directional filter G_x and G_y are as below

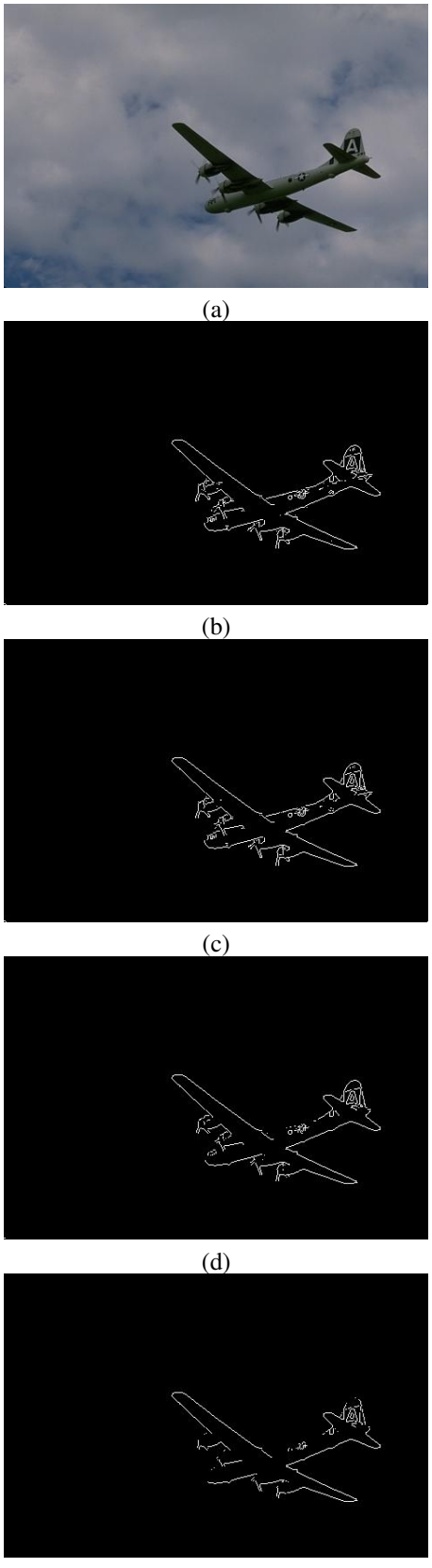
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

and

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Directional images can be computed as $I_x = G_x.conv.A$ $I_y = G_y.conv.A$ where conv represent the convolution and A represents the input image. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using

$$I = \sqrt{I_x^2 + I_y^2}$$



(a)

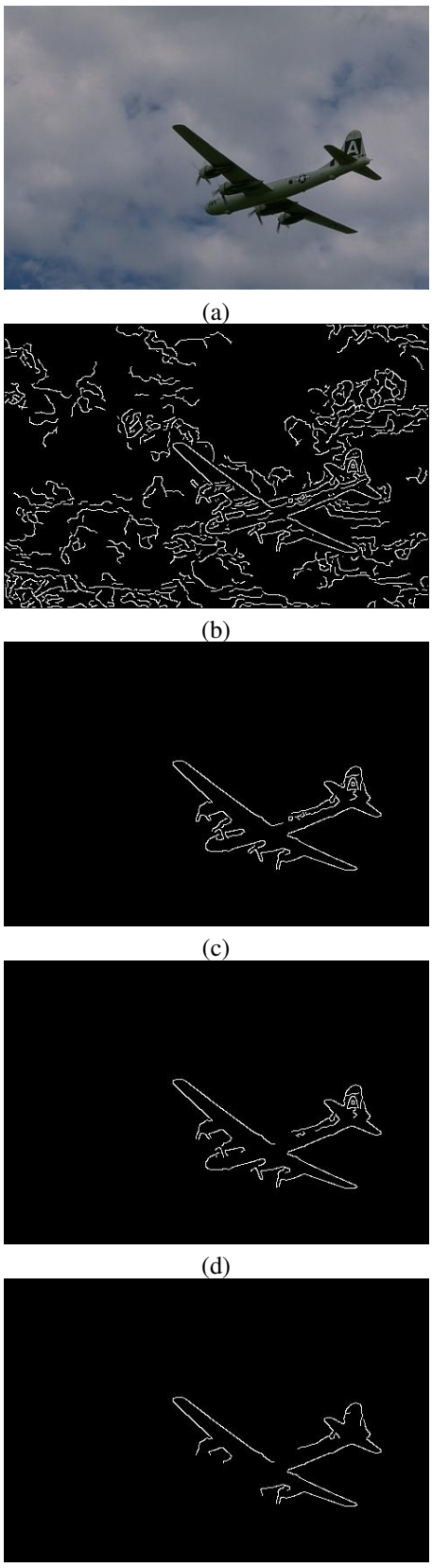
(b)

(c)

(d)

(e)

Figure 19. (a) Original Image (b-e) Sobel Edge detected, (b) threshold=default, (c) threshold=0.05 (d) threshold=0.08 (e) threshold=0.12



(a)

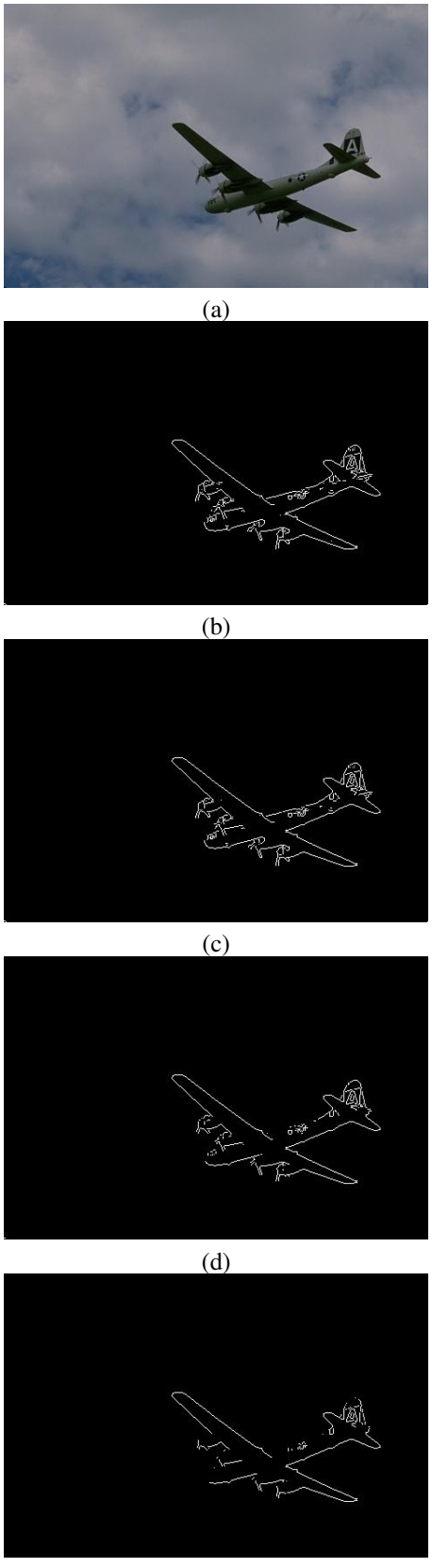
(b)

(c)

(d)

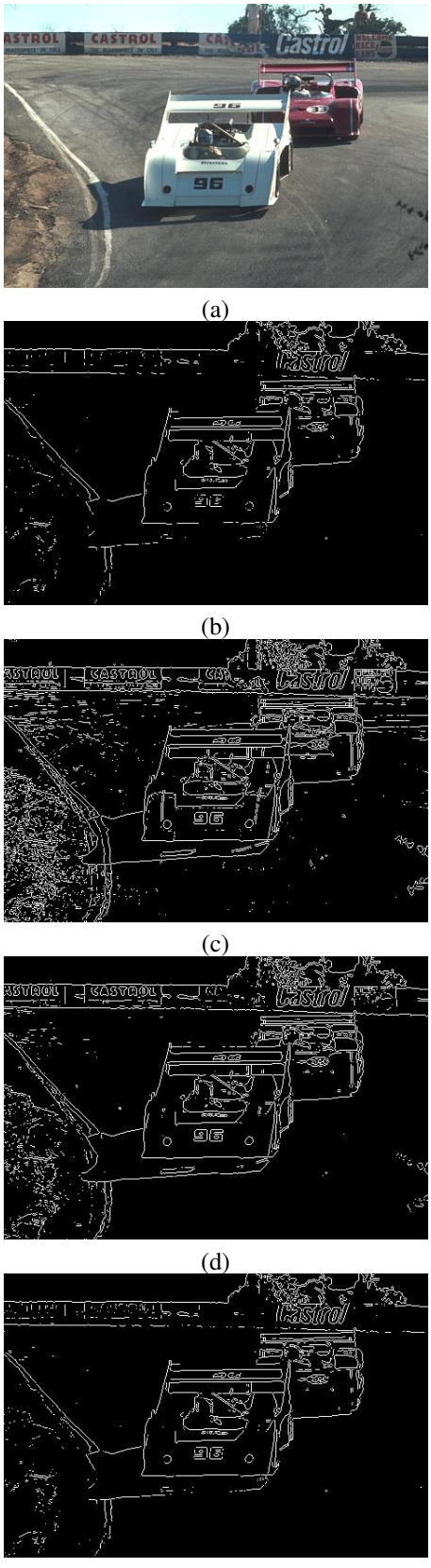
(e)

Figure 20. (a) Original Image (b-e) Canny Edge detected, (b) threshold=default, (c) threshold=0.4 (d) threshold=0.5 (e) threshold=0.7



(a) (b) (c) (d) (e)

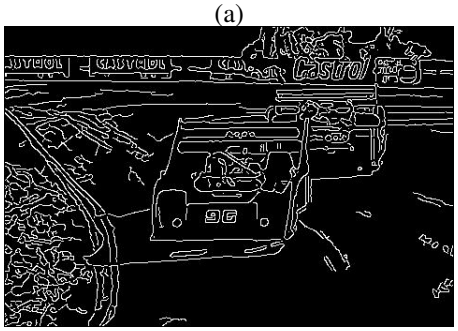
Figure 21. (a) Original Image (b-e) Prewitt Edge detected, (b) threshold=default, (c) threshold=0.05 (d) threshold=0.08 (e) threshold=0.12



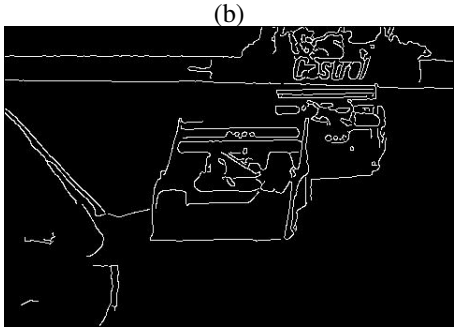
(a) (b) (c) (d) (e)

Figure 22. (a) Original Image (b-e) Sobel Edge detected, (b) threshold=default, (c) threshold=0.05 (d) threshold=0.08 (e) threshold=0.12

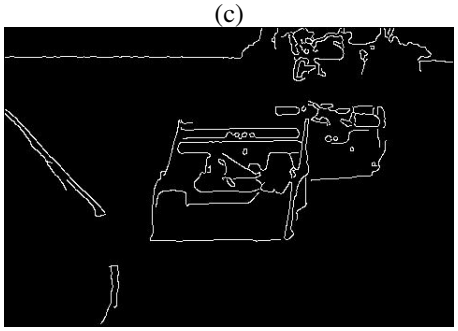
(a)



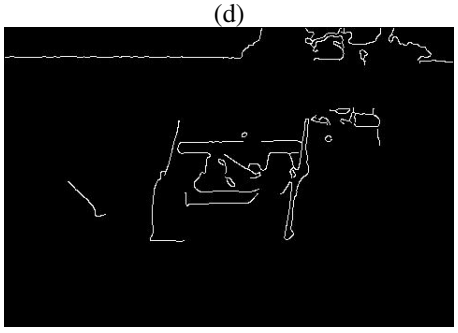
(b)



(c)



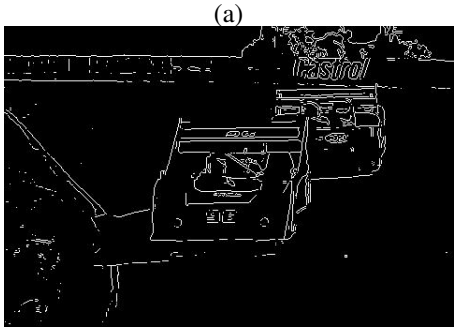
(d)



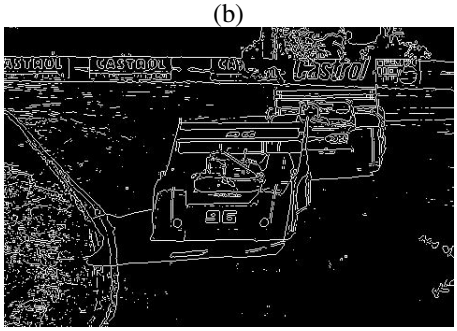
(e)

Figure 23. (a) Original Image (b-e) Canny Edge detected, (b) threshold=default, (c) threshold=0.4 (d) threshold=0.5 (e) threshold=0.7

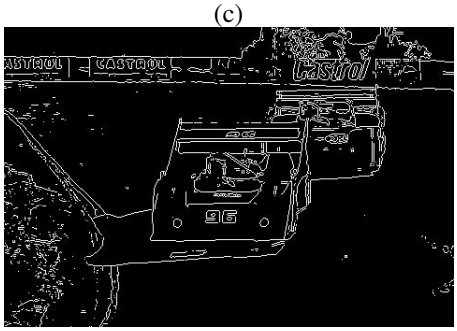
(a)



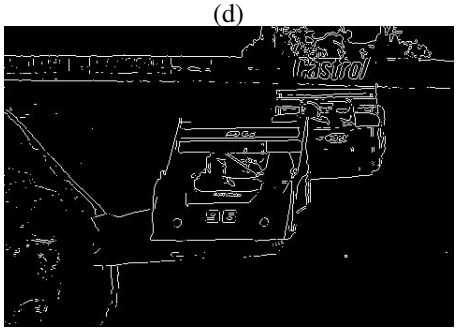
(b)



(c)



(d)



(e)

Figure 24. (a) Original Image (b-e) Prewitt Edge detected, (b) threshold=default, (c) threshold=0.05 (d) threshold=0.08 (e) threshold=0.12

The rest of images are attached separately.

$Method$	$Sobel$				$Canny$				$Prewitt$			
	$Threshold$				$Threshold$				$Threhsold$			
	$Default$	0.05	0.08	0.12	$Default$	0.4	0.5	0.7	$Default$	0.05	0.08	0.12
$3096.jpg$	0.8457	0.8465	0.8482	0.8478	0.5494	0.8473	0.84663	0.8411	0.8459	0.8468	0.8487	0.84698
$8023.jpg$	0.7697	0.5771	0.7481	0.8131	0.2630	0.7913	0.8304	0.8387	0.7685	0.588	0.75331	0.8162

*Detailed results are attached separately.

Table I
AVERAGE ACCURACY OF DEFAULT PERFORMANCE OF DIFFERENT EDGE DETECTORS

$Method$	$Sobel$	$Canny$	$Prewitt$
$Average\ Accuracy$	0.6667416144	0.2529561078	0.6723453346

Table II
AVERAGE ACCURACY OF DEFAULT PERFORMANCE OF DIFFERENT EDGE DETECTORS

D. Dataset

We referred to the complete BSDS dataset [6]. The dataset contains 500 natural images that have been manually segmented. The human annotations serve as ground truth. The specific 10 images filtered for the class experiment have also been included in this. The ground truth was available as a mat file and we wrote a small program to extract the different versions of ground truth available for each image. We noticed that some images had up to 9 different values of ground truth. We accepted the first value for each image for the purpose of comparison.

E. Experiment

We applied the default Sobel, Canny and Prewitt filter on all 500 images. The results of which have been attached separately along with the submission. The average values are shown in table 2.

For a few selected images we applied the Sobel filter with different threshold values and noted the results in table 1. Similarly, we experimented with Canny and Prewitt filter with different threshold values. The extended results is available in results.txt file attached with the submission. The default threshold value for each filter is determined per image. In all cases, the default threshold is chosen heuristically, depending on the input data.

The results of Sobel, Canny and Prewitt filter are shown in fig 19, 20, 21, 22, 23, 24. Code can be found in listing 8

F. Discussion

The results in Table 2 summaries the effectiveness of each of the edge detection methods. A few observations:

1) Performance of canny and prewitt filter is almost comparable. Under the hood, both filters work exactly the same, except for the difference in values of directional filter.
2) Sobel and Prewitt seems to be very aggressive and looses a lot of edges with a very small threshold whereas Canny is less aggressive.
3) On average Prewitt performs the best out of three with default values but Canny outperforms the other two when the parameters are tuned for each image. This is due to the fact that canny removes the noise using a gaussian filter first.
4) Since Canny filter captures a lot of details, it is prone to resulting in a lot of nomadic non-zero pixels when used with image which has a lot of minute details spread across the image, which reduces the performance of the filter.

IV. REFERENCES

[1] J. Canny. "A computational approach to edge detection". PAMI, 1986.
[2] http://goo.gl/pRj2Qq
[3] https://en.wikipedia.org/wiki/Canny_edge_detector
[4] https://en.wikipedia.org/wiki/Sobel_operator
[5] https://en.wikipedia.org/wiki/Prewitt_operator
[6] Contour Detection and Hierarchical Image Segmentation P. Arbelaez, M. Maire, C. Fowlkes and J. Malik. IEEE TPAMI, Vol. 33, No. 5, pp. 898-916, May 2011.

APPENDIX

```
clc;
close all;

ans1a;
ans1b;
ans1c;
ans2a;
ans2b;
ans2c;
ans3;

close all;
```

Listing 1. Main

```
clc;
close all;

input = imread('../Data/img/12084.jpg');
image1gray  = rgb2gray(input);
imwrite(input,'../Results/ans1a1.jpg');
imwrite(image1gray,'../Results/ans1a1_gray.jpg
    ');
```

Listing 2. Answer1a

```
clc;
close all;

input1 = imread('../Data/img/12084.jpg');
input2 = imread('../Data/img/3096.jpg');
input3 = imread('../Data/img/14037.jpg');
```

```
arr = cat(3, input1, input2, input3);
filterSize = [3,5,7];

for i=1:3    %image counter
    im = arr(:,:,(i-1)*3+1:i*3);
    for j=1:3      %filter size
        res = imgaussfilt(im, filterSize(j));
        dest = sprintf('../Results/ans1b%d_%
            d_gaussion.jpg', i, j);
        imwrite(res, dest);
    end

    for j=1:3
        h = fspecial('average', filterSize(j))
            ;
        res = imfilter(im, h);
        dest = sprintf('../Results/ans1b%d_%
            d_average.jpg', i, j);
        imwrite(res, dest);
    end
end
```

Listing 3. Answer1b

```
clc;
close all;

input1 = imread('../Data/snp/1.jpg');
input2 = imread('../Data/snp/2.jpg');
input3 = imread('../Data/snp/3.jpg');
input4 = imread('../Data/snp/4.jpg');
input5 = imread('../Data/snp/5.jpg');

arr = cat(3, input1, input2, input3, input4,
    input5);
filterSize = [14 3 5 20 7];

for i=1:5    %image counter
    imR  = arr(:,:,(i-1)*3+1);
    imG  = arr(:,:,(i-1)*3+2);
    imB  = arr(:,:,(i-1)*3+3);

    resR = medfilt2(imR, [filterSize(i)
        filterSize(i)]);
    resG = medfilt2(imG, [filterSize(i)
        filterSize(i)]);
    resB = medfilt2(imB, [filterSize(i)
        filterSize(i)]);

    res = cat(3,resR, resG, resB);
    dest = sprintf('../Results/ans1c_image%
        d_median_filterSize3x3.jpg', i);
    imwrite(res, dest);
end
```

Listing 4. Answer1c

```
clc;
close all;
warning('off', 'Images:initSize:adjustingMag')
    ;

input1 = imread('../Data/img/12084.jpg');
input2 = imread('../Data/img/3096.jpg');
input3 = imread('../Data/img/14037.jpg');

arr = cat(3, input1, input2, input3);

for i=1:3    %image counter
    imR  = arr(:,:,(i-1)*3+1);
    imG  = arr(:,:,(i-1)*3+2);
    imB  = arr(:,:,(i-1)*3+3);

    fg   = figure;
    set(fg, 'Color', [1 1 1], 'Position', [1 1
         1024 768], 'Visible', 'on', 'menubar
        ', 'none');
    [countR,r] = imhist(imR);
    [countG,g] = imhist(imG);
    [countB,b] = imhist(imB);
    subplot(5,2,[1,2,3,4]);
    imshow(arr(:,:,(i-1)*3+1:i*3));
    title('(a) Original Image');
    subplot(5,2,[5,6]);
    stem(r,countR);
    title('(b) Histogram for color Intensity (
        R)');

    subplot(5,2,[7,8]);
    stem(g,countG);
    title('(c) Histogram for color Intensity (
        G)');

    subplot(5,2,[9,10]);
    stem(b,countB);
    title('(d) Histogram for color Intensity (
        B)');

    dest = sprintf('../Results/ans2a_%d.jpg',
        i);
    saveas(fg, dest);
end

close all;
```

Listing 5. Answer2a

```
clc;
close all;
warning('off', 'Images:initSize:adjustingMag')
    ;

input1 = imread('../Data/img/12084.jpg');
input2 = imread('../Data/img/3096.jpg');
input3 = imread('../Data/img/14037.jpg');

arr = cat(3, input1, input2, input3);

for i=1:3    %image counter
    imR  = arr(:,:,(i-1)*3+1);
    imG  = arr(:,:,(i-1)*3+2);
    imB  = arr(:,:,(i-1)*3+3);

    [countR,r] = imhist(imR);
    [countG,g] = imhist(imG);
    [countB,b] = imhist(imB);

    imR_new = histeq(imR);
    imG_new = histeq(imG);
    imB_new = histeq(imB);

    fg   = figure;
    set(fg, 'Color', [1 1 1], 'Position', [1 1
         1024 786], 'Visible', 'on', 'menubar
        ', 'none');
    [countR_new,r_new] = imhist(imR_new);
    [countG_new,g_new] = imhist(imG_new);
```

```
    [countB_new,b_new] = imhist(imB_new);

    subplot(5,4,[1,2,5,6]);
    imshow(cat(3, imR, imG, imB), '
        InitialMagnification', 1000, 'border',
         'tight' );
    title('(a) Original Image');

    subplot(5,4,[3,4,7,8]);
    imshow(cat(3, imR_new, imG_new, imB_new),
        'InitialMagnification', 1000, 'border
        ', 'tight' );
    title('(b) Histogram Equalized Image');

    subplot(5,4,[9,10,11,12]);
    stem(r_new,countR_new);
    title('(c) Histogram of Histogram
        Equalized image for color Intensity (R
        )');

    subplot(5,4,[13,14,15,16]);
    stem(g_new,countG_new);
    title('(d) Histogram of Histogram
        Equalized image for color Intensity (G
        )');

    subplot(5,4,[17,18,19,20]);
    stem(b_new,countB_new);
    title('(e) Histogram of Histogram
        Equalized image for color Intensity (B
        )');

    dest = sprintf('../Results/ans2b_%d_new.
        jpg', i);
    saveas(fg, dest);

end
```

Listing 6. Answer2b

```
clc;
close all;
warning('off', 'Images:initSize:adjustingMag')
    ;

input1 = imread('../Data/img/12084.jpg');
input2 = imread('../Data/img/3096.jpg');
input3 = imread('../Data/img/14037.jpg');

arr = cat(3, input1, input2, input3);

for i=1:3    %image counter
    im  = arr(:,:,(i-1)*3+1:i*3);

    cform2lab = makecform('srgb2lab');
    LAB = applycform(im, cform2lab);
    L = LAB(:,:,1);
    LAB(:,:,1) = adapthisteq(L,'NumTiles',...
                          [16 16],'ClipLimit
                              ',0.005);

    cform2srgb = makecform('lab2srgb');
    J = applycform(LAB, cform2srgb);

    imR = J(:,:,1);
    imG = J(:,:,2);
    imB = J(:,:,3);

    fg   = figure;
    set(fg, 'Color', [1 1 1], 'Position', [1 1
         1024 786], 'Visible', 'on', 'menubar
        ', 'none');
    [countR,r] = imhist(imR);
    [countG,g] = imhist(imG);
    [countB,b] = imhist(imB);

    subplot(5,4,[1,2,5,6]);
    imshow(im);
    title('(a) Original Image');

    subplot(5,4,[3,4,7,8]);
    imshow(cat(3, imR, imG, imB), '
        InitialMagnification', 1000, 'border',
         'tight' );
    title('(b) Adaptive Histogram Equalized
        Image');

    subplot(5,4,[9,10,11,12]);
    stem(r,countR);
    title('(c) Histogram of Adaptive Histogram
         Equalized image for color Intensity (
        R)');

    subplot(5,4,[13,14,15,16]);
    stem(g,countG);
    title('(d) Histogram of Adaptive Histogram
         Equalized image for color Intensity (
        G)');

    subplot(5,4,[17,18,19,20]);
    stem(b,countB);
    title('(e) Histogram of Adaptive Histogram
         Equalized image for color Intensity (
        B)');

    dest = sprintf('../Results/ans2c_%d.jpg',
        i);
    saveas(fg, dest);

end
```

Listing 7. Answer2c

```
clc;
close all;

imagefiles = dir('../Data/img/*.jpg');
nfiles = length(imagefiles);    % Number of
    files found
pattern = '.jpg';
replacement = '';

for ii=1:nfiles
    currentfilename = imagefiles(ii).name;
    currentfilename_stripped = regexprep(
        currentfilename, pattern, replacement)
        ;
    currentimage = imread(sprintf('%s%s','../
        Data/img/',currentfilename));

    gray = rgb2gray(currentimage);

    sobel = edge(gray, 'Sobel');
    dest = sprintf('../Results/ans3_%
        s_sobel_th_default.jpg',
        currentfilename_stripped);
    imwrite(sobel, dest);
```

```
    threshold = [5,8,12];
    for j=1:3
        sobel = edge(gray, 'Sobel', threshold(
            j)/100);
        dest = sprintf('../Results/ans3_%
            s_sobel_th_0_%d.jpg',
            currentfilename_stripped,
            threshold(j));
        imwrite(sobel, dest);
    end

    threshold = [4,5,7];
    canny = edge(gray, 'Canny');
    dest = sprintf('../Results/ans3_%
        s_canny_th_default.jpg',
        currentfilename_stripped);
    imwrite(canny, dest);

    for j=1:3
        canny = edge(gray, 'Canny', threshold(
            j)/10);
        dest = sprintf('../Results/ans3_%
            s_canny_th_0_%d.jpg',
            currentfilename_stripped,
            threshold(j));
        imwrite(canny, dest);
    end

    prewitt = edge(gray, 'Prewitt');
    dest = sprintf('../Results/ans3_%
        s_prewitt_th_default.jpg',
        currentfilename_stripped);
    imwrite(prewitt, dest);
    threshold = [5,8,12];
    for j=1:3
        prewitt = edge(gray, 'Prewitt',
            threshold(j)/100);
        dest = sprintf('../Results/ans3_%
            s_prewitt_th_0_%d.jpg',
            currentfilename_stripped,
            threshold(j));
        imwrite(prewitt, dest);
    end
end
```

Listing 8. Answer3

```
clc;
close all;

matfiles = dir('groundTruth/*.mat');
nfiles = length(matfiles);    % Number of
    files found
pattern = '.mat';
replacement = '';

for ii=1:nfiles
    currentfilename = matfiles(ii).name;
    currentfilename_stripped = regexprep(
        currentfilename, pattern, replacement)
        ;
    load(sprintf('%s%s','groundTruth/',
        currentfilename));
    for j=1:length(groundTruth)
        im1 = groundTruth{1,j}.Boundaries;
        dest = sprintf('groundTruthIm%d/%s.jpg
            ', j, currentfilename_stripped);
        imwrite(im1, dest);
    end
end
```

Listing 9. Convert gound truth to image