1. (20 points) Generate the histogram of the image you are using, and then perform a number of histogram operations (at least including contrast enhancement, thresholding, and equalization) to make the image visually better for either viewing or processing (10 points).
   Done (see blow)
   If it is a color image, please first turn it into an intensity image and then generate its histogram. Try to display your histograms of the original and the processed images (5 points),
   Done
   and make some observations of the images based on their histograms (5 points).
   See comments below
   What are the general distributions of the intensity values of each histogram? How many major peaks and valleys does your histogram have, and how do they behave?
   See comments below
   How could you use the histograms to understand, analyze or segment the image?
   See comments below

```
%================================================
% Computer Vision Programming Assignment 2
% @Zhigang Zhu, 2003-2009
% City College of New York
%================================================

% Ismail Akram 8834

InputImage = 'IDPicture.bmp';
%OutputImage1 = 'IDPicture_bw.bmp';

C1 = imread(InputImage);
[ROWS COLS CHANNELS] = size(C1); % 250 * 250 *3 (RGB)

% Splitting bands for RGB
CR1 =uint8(zeros(ROWS, COLS, CHANNELS));
for band = 1 : CHANNELS,
    CR1(:,:,band) = (C1(:,:,1));
end

CG1 =uint8(zeros(ROWS, COLS, CHANNELS));
for band = 1 : CHANNELS,
    CG1(:,:,band) = (C1(:,:,2));
end

CB1 =uint8(zeros(ROWS, COLS, CHANNELS));
for band = 1 : CHANNELS,
    CB1(:,:,band) = (C1(:,:,3));
end

%I = (0.299 * CR1) + (0.587 * CG1) + (0.114 * CB1); % Gray scale
I = uint8(round(sum(C1,3)/3)); % Intensity

% Color mapping b/w
MAP = zeros(256, 3);

for i = 1 : 256,  % a comma means pause
    for band = 1:CHANNELS,
        MAP(i,band) = (i-1)/255; % scaling 0-255 into 0 to 1
    end
end

No1 = figure;
image(I);
title('Intensity Grayscale Image');
colormap(MAP)
```

Source Code: imread image and turning it into a grayscale ala Assignment 1

```
%% Question 1
% Histogram
IntensityHistogram = zeros(256, 1);
for i = 1:250
    for j = 1:250
        IntensityHistogram(I(i,j) + 1) = ...
            IntensityHistogram(I(i,j)+1)+1; % increment by 1
    end
end

No2 = figure;
bar(IntensityHistogram, 0.8); % histogram
title('Intensity Histogram');

% Contrast enhancement

% Linear Stretch
% K = 255; % max value for K (gray scale value);
% minI = 8;
% maxI = 254; % desired gray scale range [0, K]
% LinStretch = (I*(K/(maxI - minI))) - ((K/(maxI - minI))*minI);

LinStretch = imadjust(I);

No3 = figure;
image(LinStretch);
title('Contrast Enhancement')
colormap(MAP)

No4 = figure;
bar(LinStretch, 0.8);
% imhist(I);
title('Contrast Enhancement')
% colormap(MAP)

% Thresholding
I_thres = graythresh(I); % level
BW_thres = imbinarize(I, I_thres);

No5 = figure;
imshowpair(I, BW_thres, 'montage')
title('Threshold')

% Equalization
% i = MAX[0, round{CH(j)/Np}-1]
% Np = M*N/G = 250*250/256
I_eq = histeq(I, 64);

No6 = figure;
imhist(I_eq);
title('Equalization')

No7 = figure;
imshow(I_eq);
title('Equalization')
```

Source Code: Question 1; Manual histogram via for loop, then applying contrast (linear stretch), threshold function, and equalization.

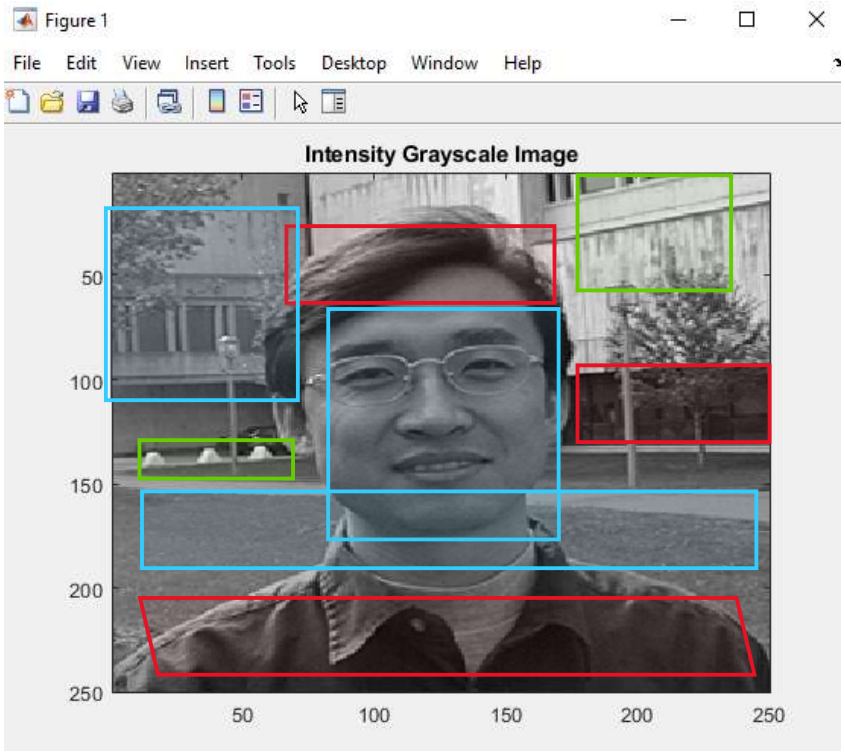


Fig 1. Grayscale intensity image.

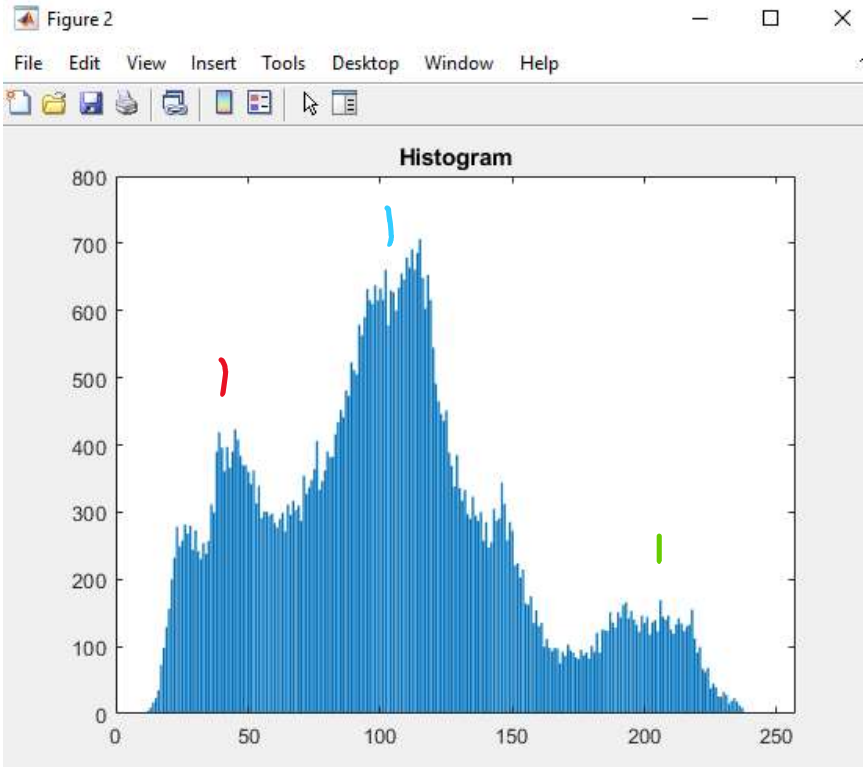


Fig 2. Manual Histogram via for loop

- Peak at grey level ~100 for grey color throughout the image, namely the face and grass in the background.
- Peak at grey level ~45 for darker values, namely the shirt, hair and underneath the building.
- Peak at ~200 for is lighter grey values, corresponding to the building.

Overall, the histogram's distribution is biased to the left, so it has generally darker grey values.
Total of three peaks (~45, ~100, ~200) and two valleys (~60, ~175).

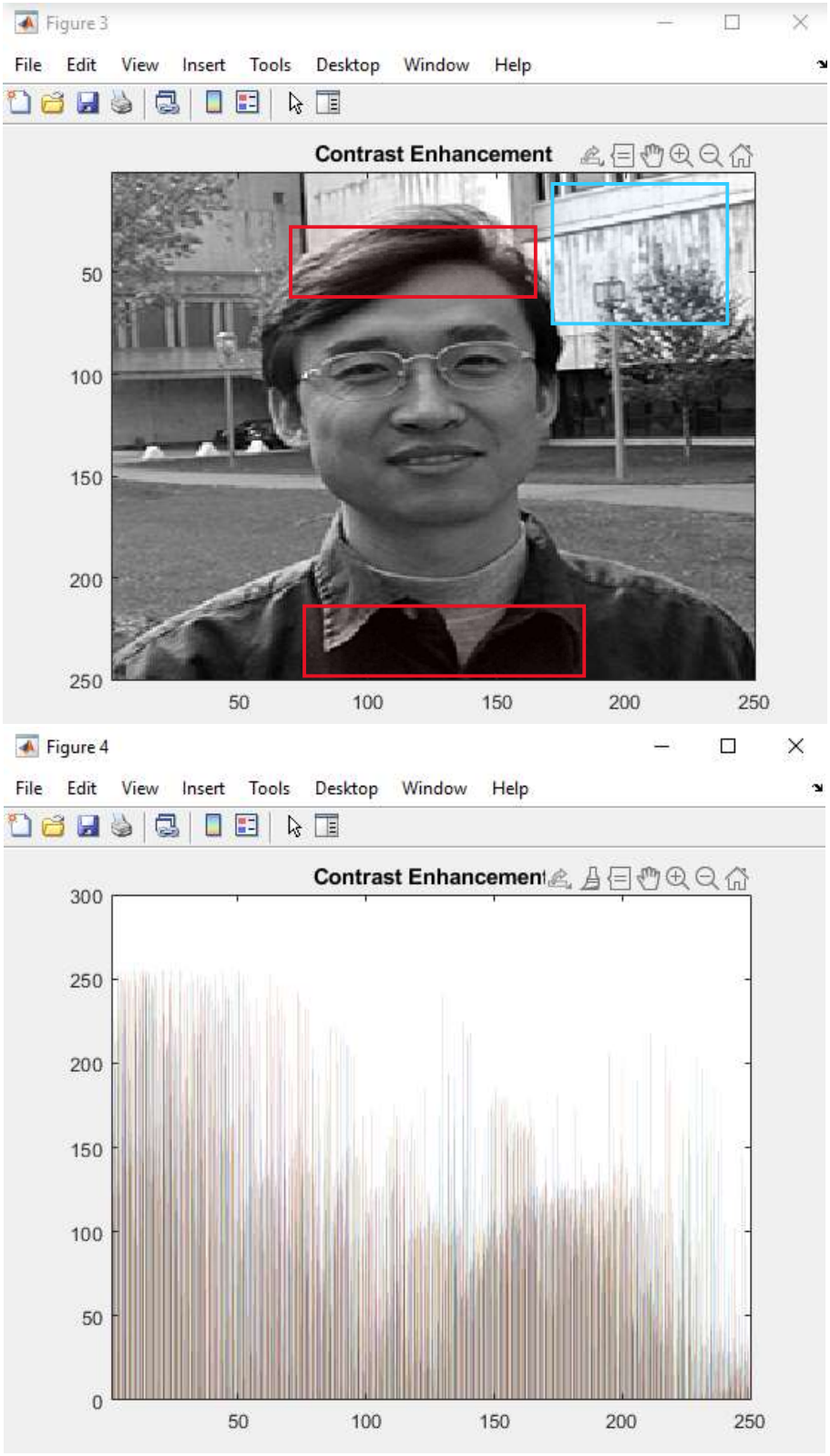


Fig 3 & 4. Contrast enhancement via Linear Stretch and imadjust function.

- Note the shift in darker value (0-50 peak) corresponding to the hair and shirt.
- Note the peak at grey level ~175 corresponding to the lighter building.
- Histogram has been expanded to grey levels 0-255. Contrast is very subtle due to Linear Stretch.

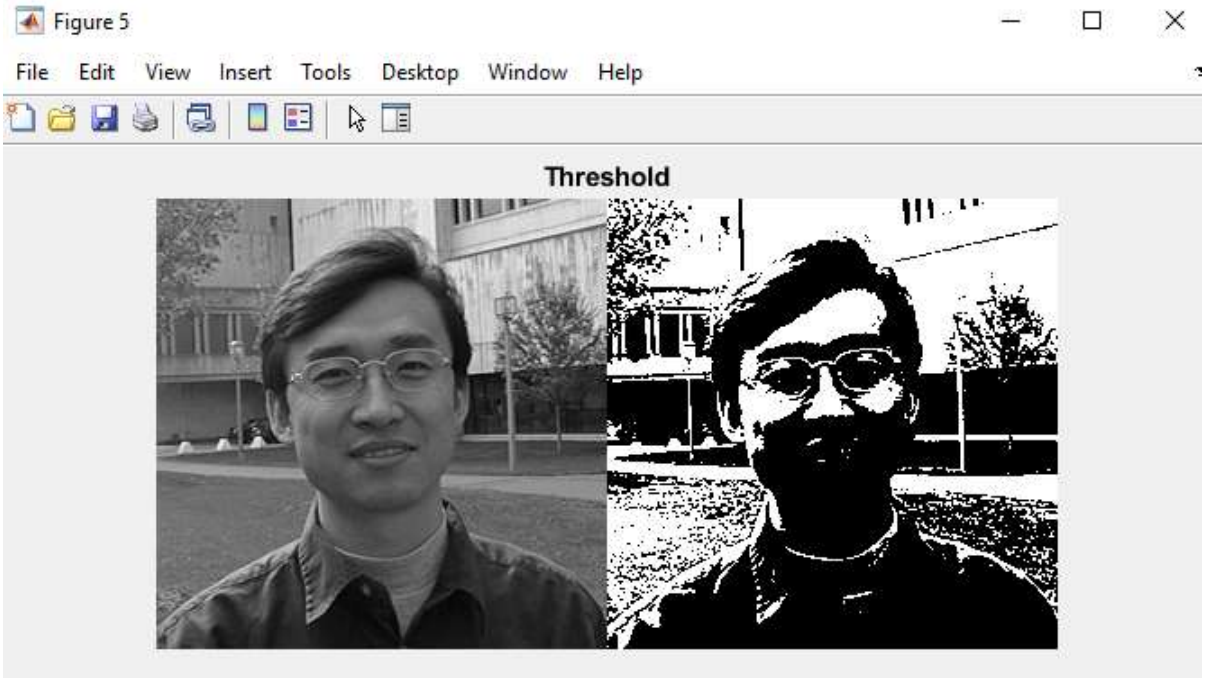


Fig 5. Threshold using graythres(). We then binarize into two gray level (0 and 255)

We're essentially applying this condition to generate the resulting binarized image:

$$I'(x,y) = \begin{cases} I_{max} \text{ if } I(x,y) > t \\ I_{min} \text{ if } I(x,y) \leq t \end{cases}$$

From Slide I-3: 18

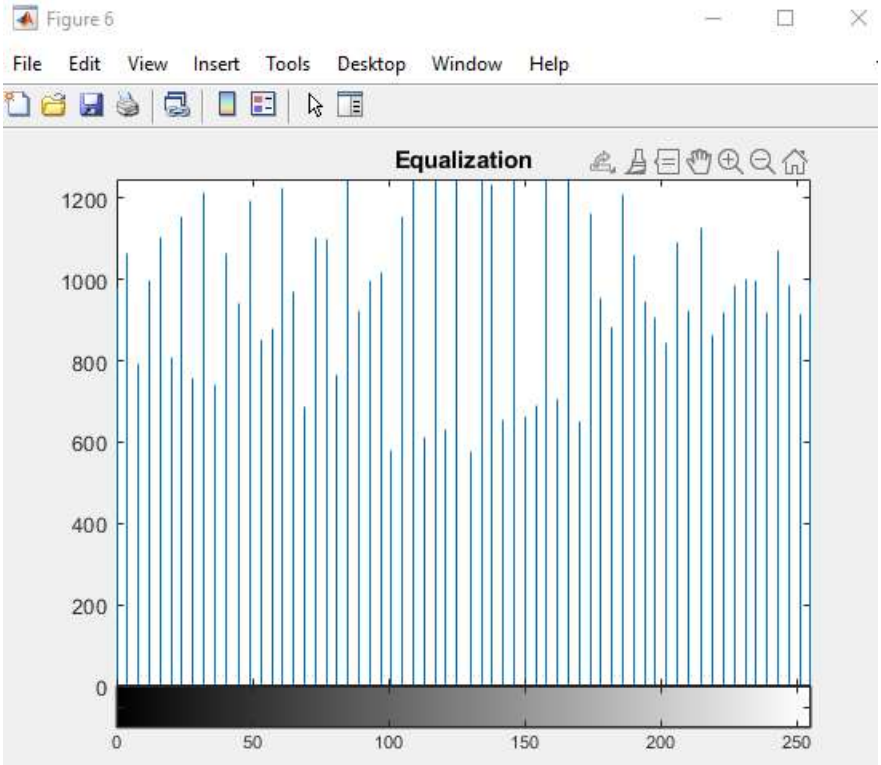


Fig 6 & 7. Equalization (using histeq function).

We're applying this logic (see Source Code 2) to net the evened out image:

$$N_p = \frac{M*N}{G}$$

$$i = MAX\left[\ 0,\ round\left\{\frac{CH(j)}{N_p}\right\} - 1\ \right]$$

$$CH(j) = \sum_{i=0}^{j} H(i)$$

From Slide I-3: 32

Histogram has an even gray level distribution.
Although some noise introduced (due to the oscillation of gray values).

2. (20 points) Apply BOTH the 1×2 operator and Sobel operator to your image and analyze and compare the results of the gradient magnitude images (including vertical gradients, horizontal gradients, and the combined) (10 points).

   Done (see below)

   Please don't forget to normalize your gradient images, noting that the original vertical and horizontal gradients have both positive and negative values. I would recommend you to display the absolute values of the horizontal and vertical gradient images.

   Done

   Does the Sobel operator have any clear visual and processing advantages over the 1×2 operator? Any disadvantages (5 points)? If you subtract the normalized 1×2 gradient image from the normalized Sobel gradient image, are there any residuals?

   See comments below

   You might use two different types of images: one ideal man-made image, and one image of a real scene with more details (5 points).

   (**Note: don't forget to normalize your results as shown in the slides of feature extraction lecture: part 2**)

```
%% Question 2
% Converting to double
input_image = double(I);

% Pre-allocating the gradient_image matric with zeros
gradient_image = zeros(size(input_image));

% 1x2 operator mask
gradient_image_1x2 = gradient_image;

Mx_1x2 = [-1, 1]; % 1x2 x-axis
% My_1x2 = [-1; 1]; % 2x1 y-axis

for i = 1:size(input_image, 1) - 2
    for j = 1:size(input_image, 2) - 2

        % Gradient approximations
        Gx = sum(sum(Mx_1x2.*input_image(i:i+1, j:j+1)));
        % Gy = sum(sum(My_1x2.*input_image(i:i+1, j:j+1)));

        % Calculate magnitude of vector
        gradient_image_1x2(i+1, j+1) = sqrt(Gx.^2);
    end
end

% Displaying Gradient Image
% gradient_image_1x2 = normalize(gradient_image_1x2);
gradient_image_1x2 = (uint8(gradient_image_1x2))/4; % normalizing |-1| + |1| + |-1| + |1|

No8 = figure;
imshow(gradient_image_1x2);
title('Gradient Image (1x2)');

% 2x1 operator mask
gradient_image_2x1 = gradient_image;

% Mx_2x1 = [-1, 1]; % 1x2 x-axis
My_2x1 = [-1; 1]; % 2x1 y-axis

for i = 1:size(input_image, 1) - 2
    for j = 1:size(input_image, 2) - 2

        % Gradient approximations
        % Gx = sum(sum(Mx_2x1.*input_image(i:i+1, j:j+1)));
        Gy = sum(sum(My_2x1.*input_image(i:i+1, j:j+1)));

        % Calculate magnitude of vector
        gradient_image_2x1(i+1, j+1) = sqrt(Gy.^2);
    end
end

% Displaying Gradient Image
% gradient_image_2x1 = normalize(gradient_image_2x1);
gradient_image_2x1 = (uint8(gradient_image_2x1))/4;
No10 = figure;
imshow(gradient_image_2x1);
title('Gradient Image (2x1)');
```

```
% Combined 2x2 operator mask
gradient_image_2x2 = gradient_image;

Mx_2x2 = [-1, 1]; % 1x2 x-axis
My_2x2 = [-1; 1]; % 2x1 y-axis

for i = 1:size(input_image, 1) - 2
    for j = 1:size(input_image, 2) - 2

        % Gradient approximations
        Gx = sum(sum(Mx_2x2.*input_image(i:i+1, j:j+1)));
        Gy = sum(sum(My_2x2.*input_image(i:i+1, j:j+1)));

        % Calculate magnitude of vector
        gradient_image_2x2(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

    end
end

% Displaying Gradient Image
% gradient_image_2x2 = normalize(gradient_image_2x2);
gradient_image_2x2 = uint8(gradient_image_2x2/4);
No12 = figure;
imshow(gradient_image_2x2);
title('Gradient Image (2x2)');

% Sobel Operator Mask
gradient_image_S = gradient_image;

Mx_S = [-1 0 1; -2 0 2; -1 0 1];
My_S = [-1 -2 -1; 0 0 0; 1 2 1];

% Edge Detection Process
% When i = 1 and j = 1, then gradient_image pixel
% position will be gradient_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to gradient_image(size(input_image, 1) - 2
%, size(input_image, 2) - 2)
% Thus we are not considering the borders.

for i = 1:size(input_image, 1) - 2
    for j = 1:size(input_image, 2) - 2

        % Gradient approximations
        Gx = sum(sum(Mx_S.*input_image(i:i+2, j:j+2)));
        Gy = sum(sum(My_S.*input_image(i:i+2, j:j+2)));

        % Calculate magnitude of vector
        gradient_image_S(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);
    end
end

% Displaying Gradient Image
% gradient_image_S = normalize(gradient_image_S);
gradient_image_S = uint8(gradient_image_S/8);

No14 = figure;
imshow(gradient_image_S);
title('Gradient Image (Sobel 3x3)');
```

Source code: 1x2 and SOBEL operator for gradient mapping

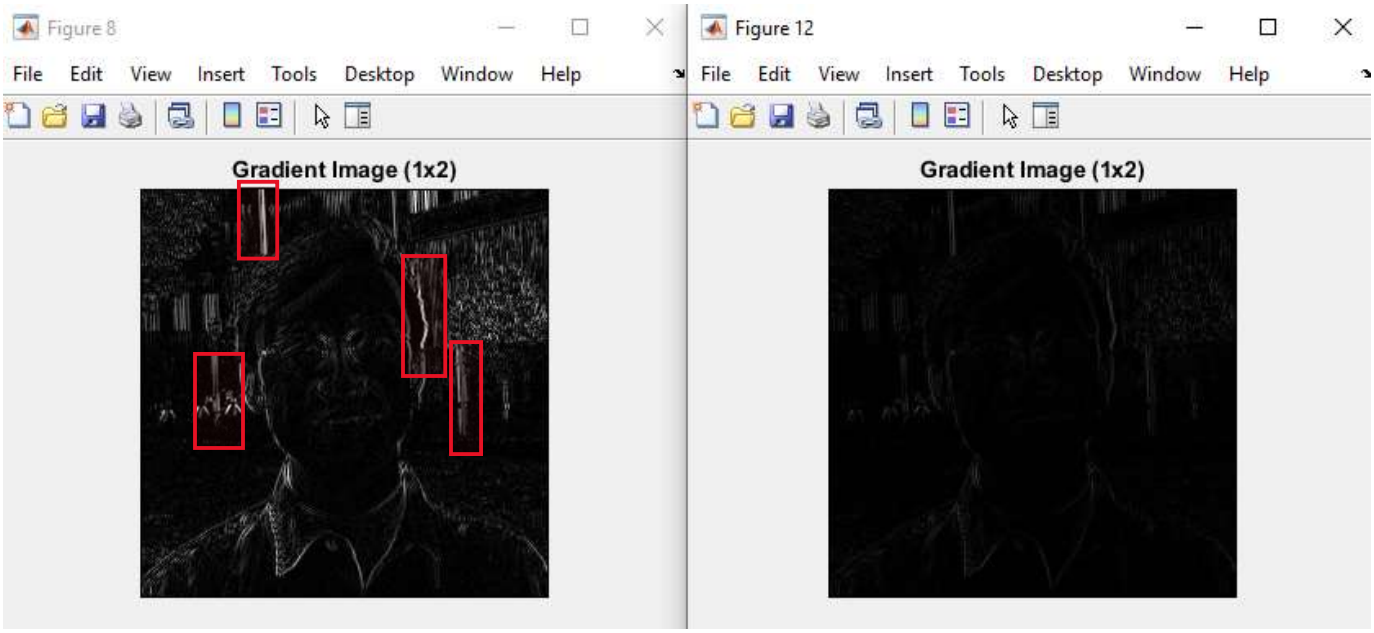


Fig 8. Gradient image and normalized (right) of 1x2 operator. It made the image dimmer.

We can see some strong vertical edges in the building, tree trunks, and the right side of the hair.

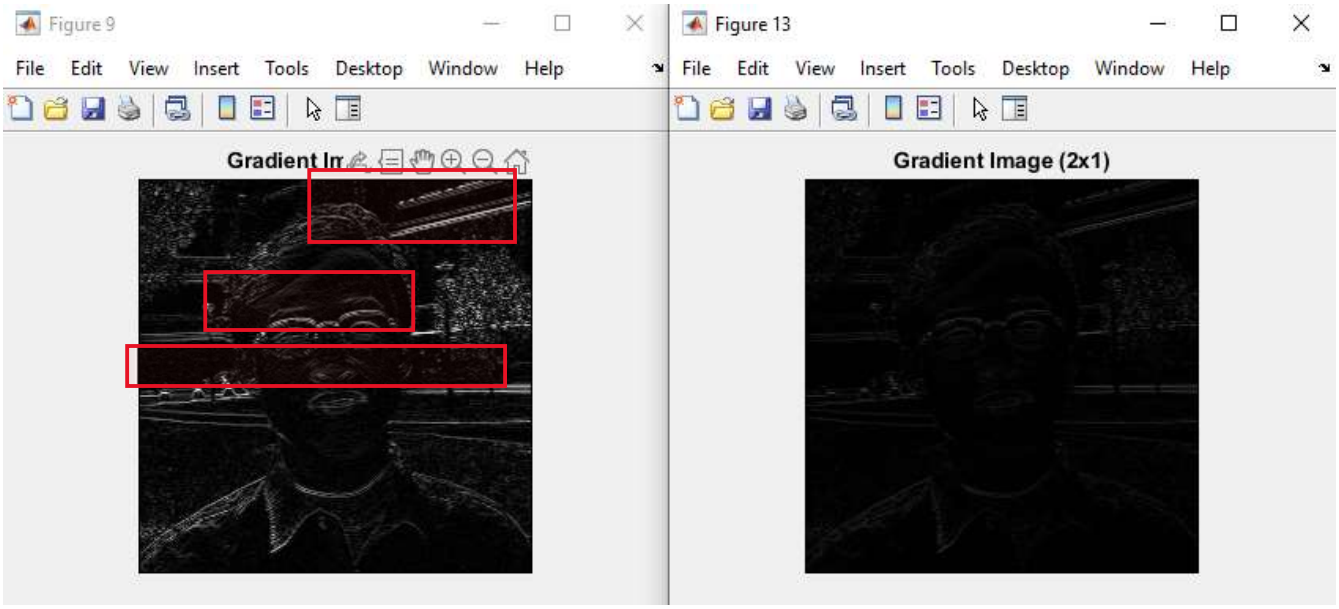


Fig 9. Gradient image and normalized (right) of the 2x1 operator.

We can see some strong horizontal edges along the glasses frame, field background, and the building.

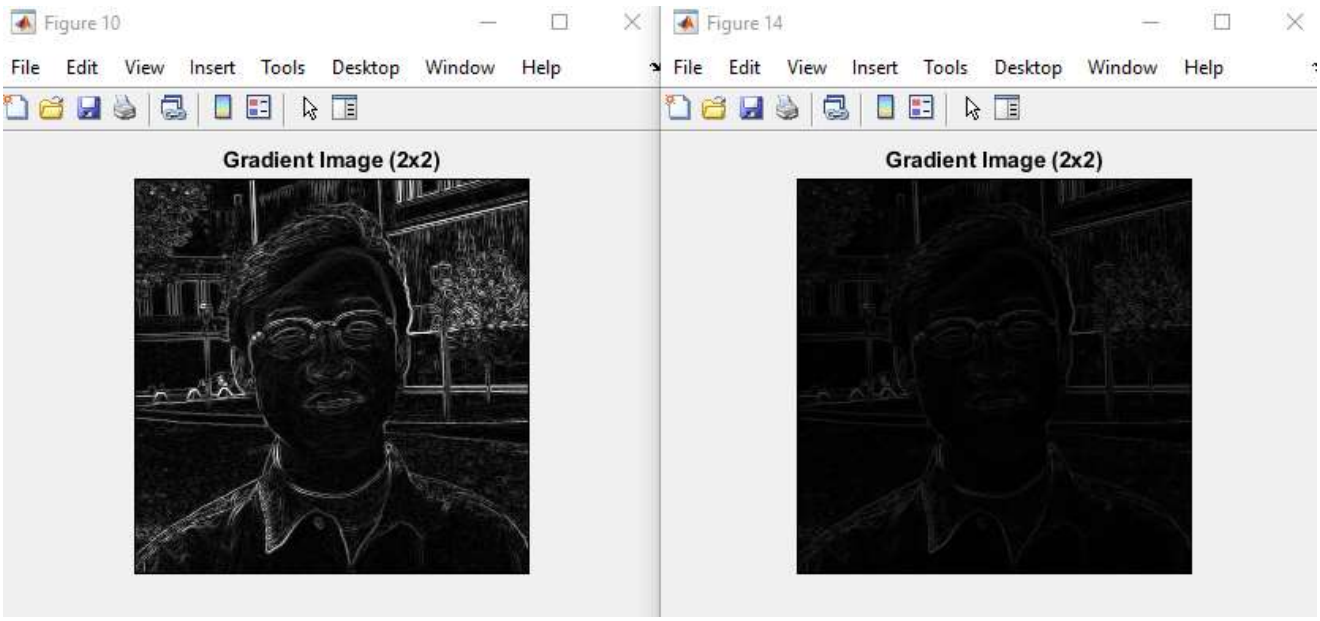


Fig 10. Gradient image and normalized (right) of the 2x2 operator.

Combined gradient output of 1x2 and 2x1 to net the 2x2 image.

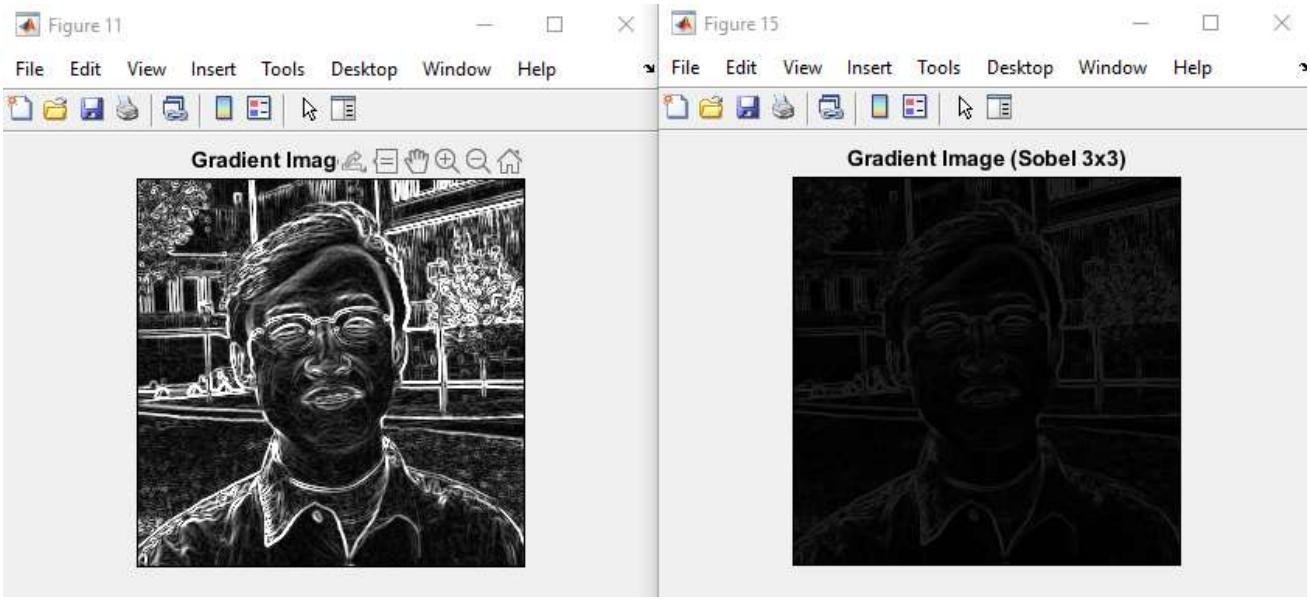


Fig 11. Gradient image and normalized (right) of the Sobel operator.

Much higher visual clarity from the 3x3 matrix. But the processing might be more expensive, but the results are more prominent than either of 1x2 and 2x1 and 2x2. Note the prominent outline.

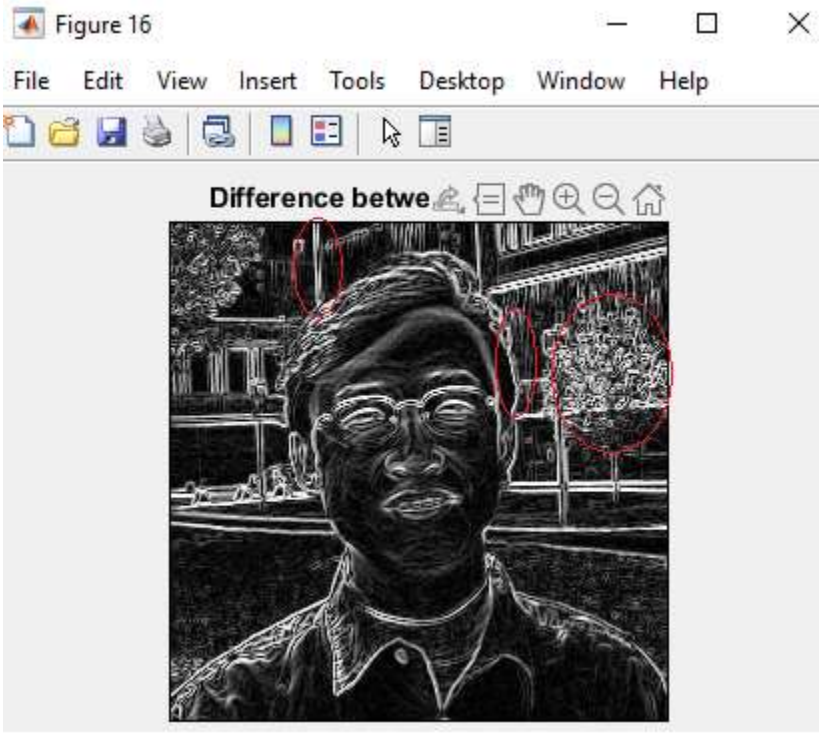


Fig 16. Difference between Sobel and 1x2

Removal of edges prominent in 1x2 from Sobel output. Note the slightly thinner edge on the right side of the hair, thinner edge on the building and tree is slightly dimmer.

3. (20 points) Generate edge maps of the above two combined gradient maps (10 points). An edge image should be a binary image with 1s as edge points and 0s as non-edge points. You may first generate a histogram of each combined gradient map, and only keep certain percentage of pixels (e.g. 5% of the pixels with the highest gradient values) as edge pixels (edgels) .
   Done
   Please study what is the best percentage for a specific image, and why. Use the varying percentage to automatically find a corresponding threshold for the gradient magnitudes, and then pick up the one having the best visual performance.
   In your report, please write up the description and probably equations for finding the threshold and discuss what percentage is a good value (5 points).
   Done

```
%% Question 3
% % % No14 = figure;
% % % imhist(gradient_image_2x2); NEEDED?
% % % title('threshold-test')

% You may first generate a histogram of each combined gradient map,
% and only keep certain percentage of pixels
% (e.g.  5% of the pixels with the highest gradient  values)
% as edge pixels (edgels)

% Get the histogram values position in original picture.
% Take that position, divide it by total possible pixels.
% if equal .95, set that value equal to threhsold.
threshold = 0;
position = zeros(256,1);
var = ((250 * 250) * (1 - 0.95));
result = 0;
for i = 1:256
     count = IntensityHistogram(i);
     result = result + count;
     if (result >= var)
         threshold = i;
         break;
     end
end

% disp(threshold);

% 1x2 Edge Map
output_image_1x2 = max(gradient_image_1x2, threshold);
output_image_1x2(output_image_1x2 == round(threshold)) = 0;

% Displaying Output Image
%output_image_1x2 = uint8(output_image_1x2);
output_image_1x2 = im2bw(output_image_1x2);
No13 = figure;
imshow(output_image_1x2);
title('Edge Detected Image (1x2)');

% 2x1 Edge Map
output_image_2x1 = max(gradient_image_2x1, threshold);
output_image_2x1(output_image_2x1 == round(threshold)) = 0;

% Displaying Output Image
output_image_2x1 = im2bw(output_image_2x1);
No14 = figure;
imshow(output_image_2x1);
title('Edge Detected Image (2x1)');

% 2x2 Edge Map
output_image_2x2 = max(gradient_image_2x2, threshold);
output_image_2x2(output_image_2x2 == round(threshold)) = 0;

% Displaying Output Image
output_image_2x2 = im2bw(output_image_2x2); % im2bw
No15 = figure;
imshow(output_image_2x2);
title('Edge Detected Image (2x2)');

% Sobel Edge Map
output_image_S = max(gradient_image_S, threshold);
output_image_S(output_image_S == round(threshold)) = 0;

% Displaying Output Image
output_image_S = im2bw(output_image_S); % im2bw
No16 = figure;
imshow(output_image_S);
title('Edge Detected Image (Sobel 3x3)');
```

Source code: determining the 5% pixels with the highest pixel values)

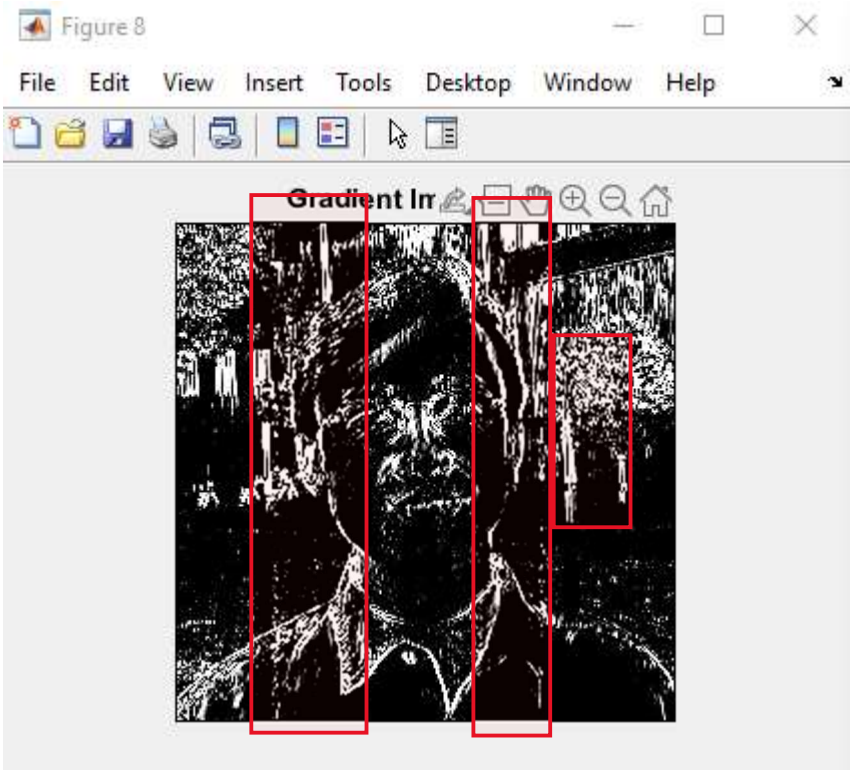


Fig 17. Edge map of 1x2

We can see all the prominent vertical edges here.

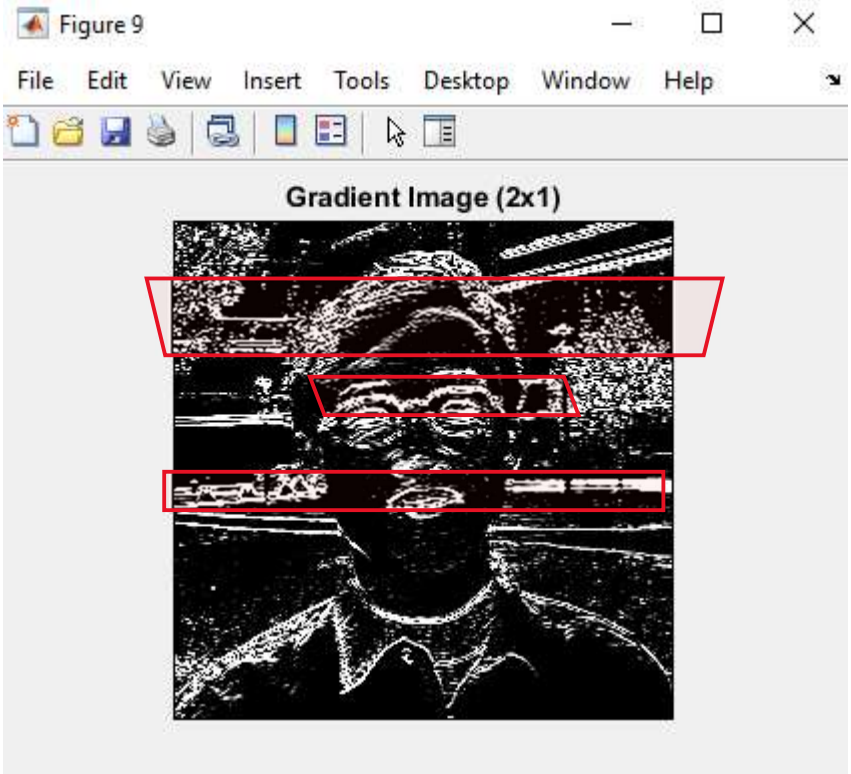


Fig 18. Edge map of 2x1

We can see the prominent horizontal edges here

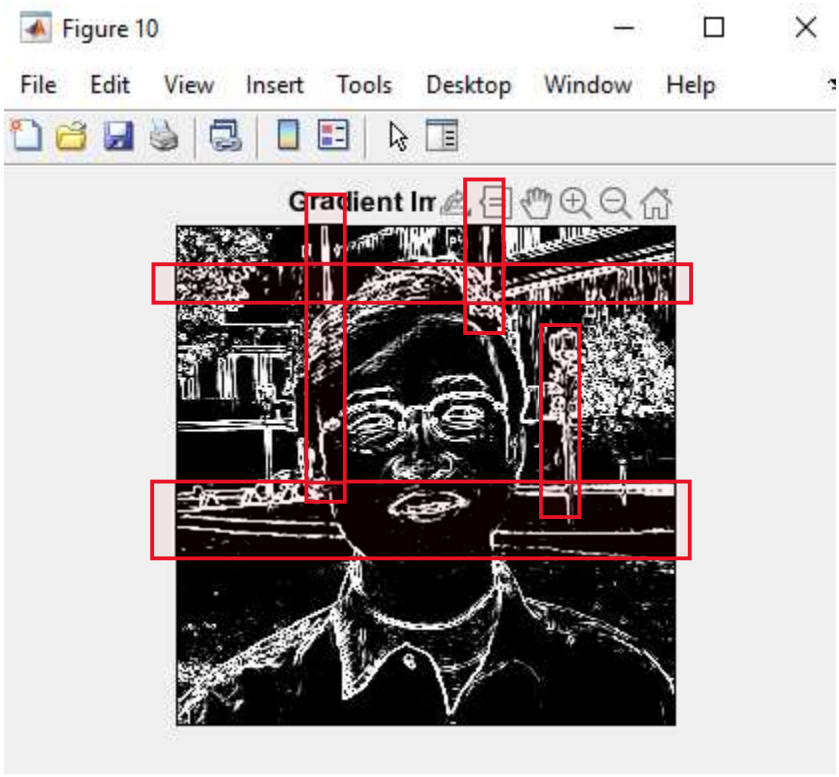


Fig 19. 2x2 Combined edges.

We can see the combination of vertical and horizontal edges.

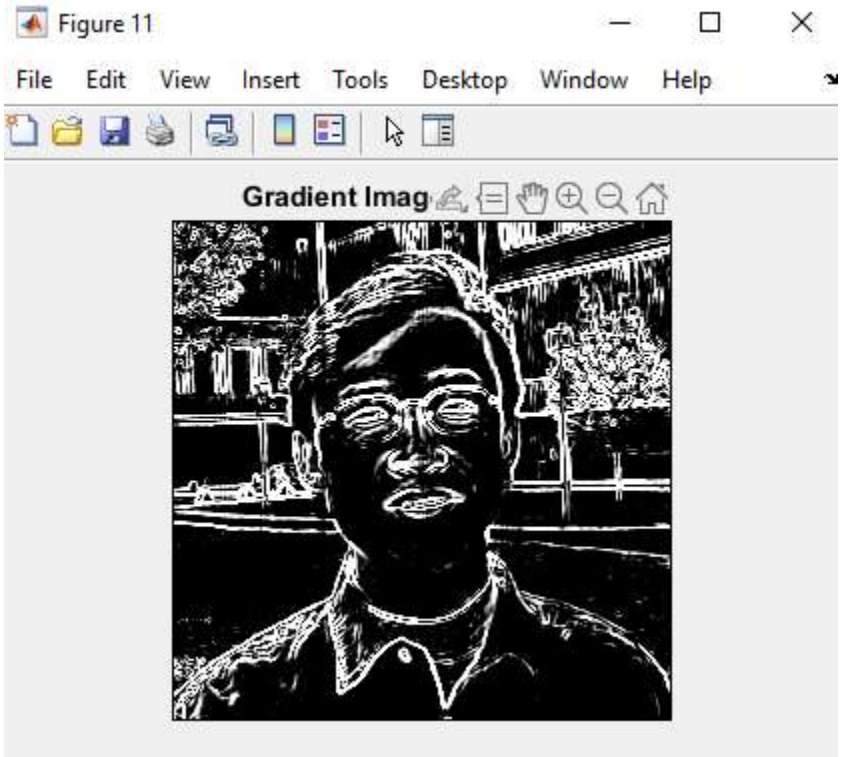


Fig 20. Sobel edge map

And with Sobel we have a much cleaner edge map.

You may also consider to use local, adaptive thresholds to different portions of the image so that all major edges will be shown up nicely (5 points). In the end, please try to generate a sketch of an image, such as the ID image of Prof. Zhu.

Done (see above Sobel image)

4. (20 points) What happens when you increase the size of the edge detection kernel from 1×2 to 3×3 and then to 5×5 , or 7×7? Discuss (1) computational cost (in terms of members of operations, and the real machine running times – 5 points); (2) edge detection results (5 points) and (3) sensitivity to noise, etc. (5 points). Note that your larger kernel should still be an edge detector.

Edge detection for 1x2 and 3x3 are already done in Questions 2 & 3.

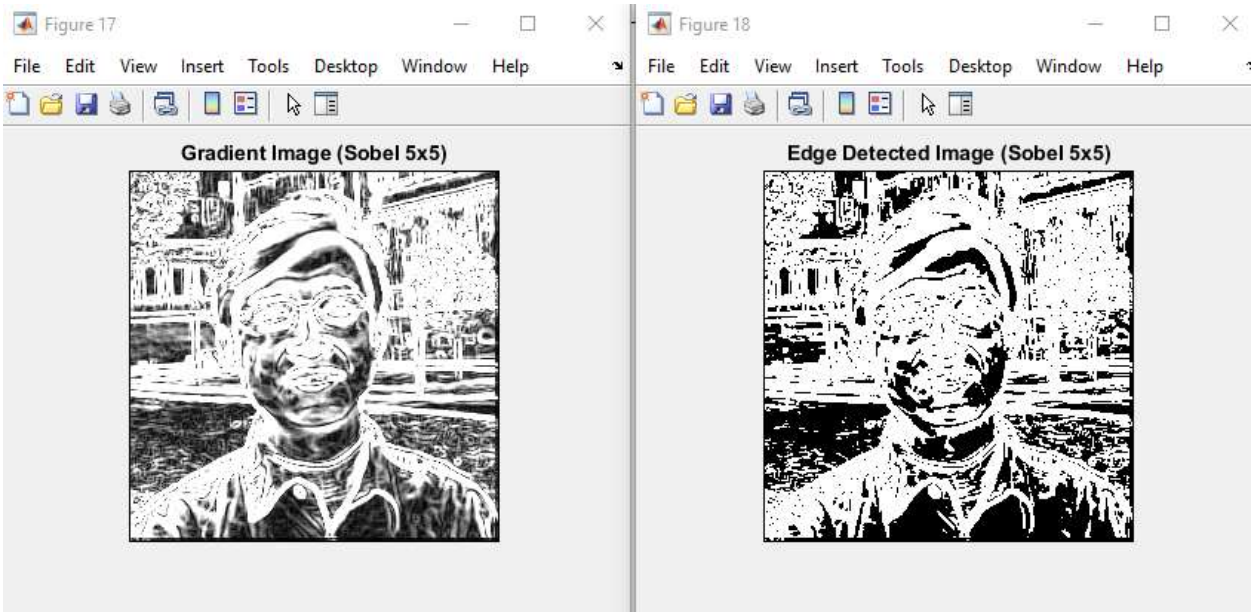


Fig 17 & 18 Gradient and edge image of Sobel 5x5

Since the kernel map is wider and catching more difference, we see more “thick/blurry” edges being picked up. Certain edges, such as the crease lines on the cheeks have been picked up now. Pre-existing edges have been amplified. A lot of noise introduced. Computation cost is higher due to using a 5x5 array (see below in red box).

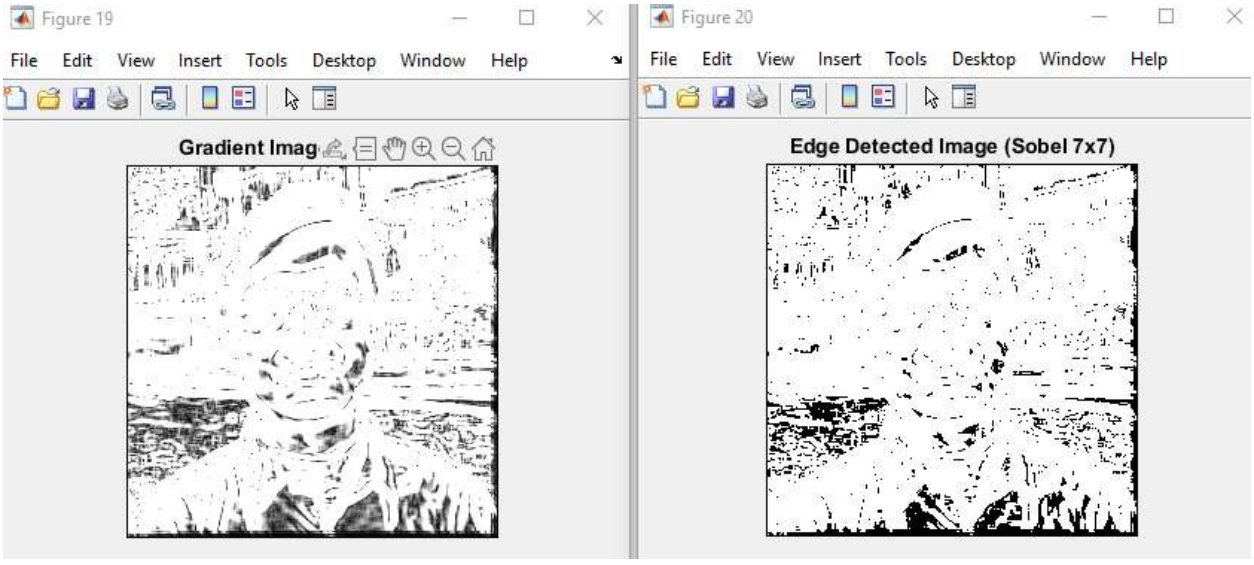
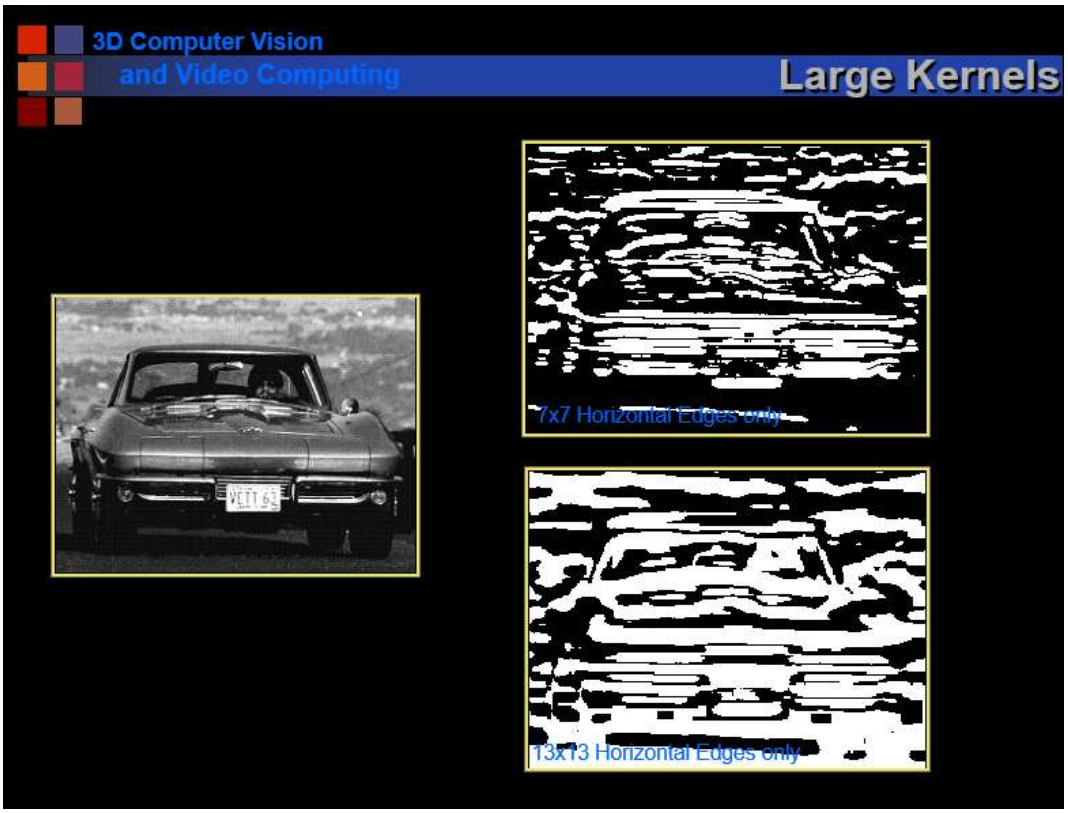


Fig 19 & 20 Gradient and edge image of Sobel 7x7

The previous issues in 5x5 kernal map have been greatly amplified in this 7x7 kernel. To the point we can barely recognize the image. While this mapping is great for catching extremely tiny details,

it catches too much to the point it's obsolete. A lot of noise has also been introduced. Cost is significantly higher due to using a 7x7 array (see below in red box).

It reminds me of this slide from the lectures:



Please list your kernels as matrices in your report, and tell us what they are good for (5 points).

```
%% Question 4
% Sobel 5x5 Operator Mask
Mx_S5x5 = [+2 +2 +4 +2 +2; +1 +1 +2 +1 +1; 0 0 0 0 0; -1 -1 -2 -1 -1; -2 -2 -4 -2 -2];
My_S5x5 = [+2 +1 0 -1 -2; +2 +1 0 -1 -2; +4 +2 0 -2 -4; +2 +1 0 -1 -2; +2 +1 0 -1 -2];

% Edge Detection Process
% When i = 1 and j = 1, then gradient_image pixel
% position will be gradient_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to gradient_image(size(input_image, 1) - 2
%, size(input_image, 2) - 2)
% Thus we are not considering the borders.

for i = 1:size(input_image, 1) - 4
    for j = 1:size(input_image, 2) - 4

        % Gradient approximations
        Gx = sum(sum(Mx_S5x5.*input_image(i:i+4, j:j+4)));
        Gy = sum(sum(My_S5x5.*input_image(i:i+4, j:j+4)));

        % Calculate magnitude of vector
        gradient_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

    end
end

% Displaying Gradient Image
gradient_image_S5x5 = uint8(gradient_image);
No16 = figure;
imshow(gradient_image_S5x5);
title('Gradient Image (Sobel 5x5)');

output_image_S5x5 = max(gradient_image_S5x5, thresholdValue);
output_image_S5x5(output_image_S == round(thresholdValue)) = 0;

% Displaying Output Image
output_image_S5x5 = im2bw(output_image_S5x5); % im2bw
No17 = figure;
imshow(output_image_S5x5);
title('Edge Detected Image (Sobel 5x5)');

% Sobel 7x7 Operator Mask
Mx_S7x7 = [+3 +3 +3 +6 +3 +3 +3; +2 +2 +2 +4 +2 +2 +2; +1 +1 +1 +2 +1 +1 +1; 0 0 0 0 0 0 0; -1 -1 -1 -2 -1 -1 -1; -2 -2 -2 -4 -2 -2 -2; -3 -3 -3 -6 -3 -3 -3];
My_S7x7 = [+3 +2 +1 0 -1 -2 -3; +3 +2 +1 0 -1 -2 -3; +3 +2 +1 0 -1 -2 -3; +3 +4 +2 0 -2 -4 -6; +3 +2 +1 0 -1 -2 -3; +3 +2 +1 0 -1 -2 -3; +3 +2 +1 0 -1 -2 -3];

% Edge Detection Process
% When i = 1 and j = 1, then gradient_image pixel
% position will be gradient_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to gradient_image(size(input_image, 1) - 2
%, size(input_image, 2) - 2)
% Thus we are not considering the borders.

for i = 1:size(input_image, 1) - 6
    for j = 1:size(input_image, 2) - 6

        % Gradient approximations
        Gx = sum(sum(Mx_S7x7.*input_image(i:i+6, j:j+6)));
        Gy = sum(sum(My_S7x7.*input_image(i:i+6, j:j+6)));

        % Calculate magnitude of vector
        gradient_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

    end
end

% Displaying Gradient Image
gradient_image_S7x7 = uint8(gradient_image);
No18 = figure;
imshow(gradient_image_S7x7);
title('Gradient Image (Sobel 7x7)');

output_image_S7x7 = max(gradient_image_S7x7, thresholdValue);
output_image_S7x7(output_image_S == round(thresholdValue)) = 0;

% Displaying Output Image
output_image_S7x7 = im2bw(output_image_S7x7); % im2bw
No19 = figure;
imshow(output_image_S7x7);
title('Edge Detected Image (Sobel 7x7)');
```

Source Code: Sobel operator 5x5 and 7x7. Respective matrices highlighted in red.

5. (20 points) Suppose you apply the Sobel operator to each of the RGB color bands of a color image. How might you combine these results into a color edge detector (5 points)?

   Done (see end of code below; I used concatenation)

   Do the resulting edge differ from the gray scale results? How and why (5 points)?

   Yes, the edge images have significantly less noise. Due to less/no oscillation in these bands histograms. That and these bands have less information that the full color to work with.

   You may compare the edge maps of the intensity image (of the color image), the gray-scale edge map that are the combination of the three edge maps from three color bands, or a real color edge map that edge points have colors (5 points). Please discuss their similarities and differences, and how each of them can be used for image enhancement or feature extraction (5 points). Note that you want to first generate gradient maps and then using thresholding to generate edge maps. In the end, please try to generate a color sketch of an image, such as the ID image of Prof. Zhu. You may also consider local, adaptive thresholding in generating a color edge map.

   Done (see below)

```
%% Question 5
% Converting to double

% CRBand = uint8(CR1);
% RBand_input_image = double(CRBand); % red band

% Red band
RBand_input_image = C1(:, :, 1);
RBand_input_image = double(RBand_input_image);

% Pre-allocating the gradient_image matric with zeros
gradient_image = zeros(size(RBand_input_image));

% Sobel Operator Mask
Mx_S = [-1 0 1; -2 0 2; -1 0 1];
My_S = [-1 -2 -1; 0 0 0; 1 2 1];

% Edge Detection Process
% When i = 1 and j = 1, then gradient_image pixel
% position will be gradient_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to gradient_image(size(input_image, 1) - 2
%, size(input_image, 2) - 2)
% Thus we are not considering the borders.

for i = 1:size(RBand_input_image, 1) - 2
    for j = 1:size(RBand_input_image, 2) - 2

        % Gradient approximations
        Gx = sum(sum(Mx_S.*RBand_input_image(i:i+2, j:j+2)));
        Gy = sum(sum(My_S.*RBand_input_image(i:i+2, j:j+2)));

        % Calculate magnitude of vector
        gradient_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

    end
end

% Displaying Gradient Image
gradient_image_RBand = uint8(gradient_image);
No20 = figure;
imshow(gradient_image_RBand);
title('Gradient Image Red Band (Sobel 3x3)');

output_image_RB = max(gradient_image_RBand, thresholdValue);
output_image_RB(output_image_RB == round(thresholdValue)) = 0;

% Displaying Output Image
output_image_RB = im2bw(output_image_RB); % im2bw
No21 = figure;
imshow(output_image_RB);
title('Edge Detected Image Red Band (Sobel 3x3)');

% Green band
GBand_input_image = C1(:, :, 1);
GBand_input_image = double(GBand_input_image);

% Pre-allocating the gradient_image matric with zeros
gradient_image = zeros(size(GBand_input_image));

% Sobel Operator Mask
Mx_S = [-1 0 1; -2 0 2; -1 0 1];
My_S = [-1 -2 -1; 0 0 0; 1 2 1];

% Edge Detection Process
% When i = 1 and j = 1, then gradient_image pixel
% position will be gradient_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to gradient_image(size(input_image, 1) - 2
%, size(input_image, 2) - 2)
% Thus we are not considering the borders.

for i = 1:size(GBand_input_image, 1) - 2
    for j = 1:size(GBand_input_image, 2) - 2

        % Gradient approximations
        Gx = sum(sum(Mx_S.*GBand_input_image(i:i+2, j:j+2)));
        Gy = sum(sum(My_S.*GBand_input_image(i:i+2, j:j+2)));

        % Calculate magnitude of vector
        gradient_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

    end
end
```

```
% Displaying Gradient Image
gradient_image_GBand = uint8(gradient_image);
No22 = figure;
imshow(gradient_image_GBand);
title('Gradient Image Green Band (Sobel 3x3)');

output_image_GB = max(gradient_image_GBand, thresholdValue);
output_image_GB(output_image_GB == round(thresholdValue)) = 0;

% Displaying Output Image
output_image_GB = im2bw(output_image_GB); % im2bw
No23 = figure;
imshow(output_image_GB);
title('Edge Detected Image Green Band (Sobel 3x3)');

% Blue band
BBand_input_image = C1(:, :, 1);
BBand_input_image = double(BBand_input_image);

% Pre-allocating the gradient_image matric with zeros
gradient_image = zeros(size(BBand_input_image));

% Sobel Operator Mask
Mx_S = [-1 0 1; -2 0 2; -1 0 1];
My_S = [-1 -2 -1; 0 0 0; 1 2 1];

% Edge Detection Process
% When i = 1 and j = 1, then gradient_image pixel
% position will be gradient_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to gradient_image(size(input_image, 1) - 2
%, size(input_image, 2) - 2)
% Thus we are not considering the borders.

for i = 1:size(BBand_input_image, 1) - 2
    for j = 1:size(BBand_input_image, 2) - 2

        % Gradient approximations
        Gx = sum(sum(Mx_S.*BBand_input_image(i:i+2, j:j+2)));
        Gy = sum(sum(My_S.*BBand_input_image(i:i+2, j:j+2)));

        % Calculate magnitude of vector
        gradient_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

    end
end

% Displaying Gradient Image
gradient_image_BBand = uint8(gradient_image);
No24 = figure;
imshow(gradient_image_BBand);
title('Gradient Image Blue Band (Sobel 3x3)');

output_image_BB = max(gradient_image_BBand, thresholdValue);
output_image_BB(output_image_BB == round(thresholdValue)) = 0;

% Displaying Output Image
output_image_BB = im2bw(output_image_BB); % im2bw
No25 = figure;
imshow(output_image_BB);
title('Edge Detected Image Blue Band (Sobel 3x3)');

Full_color_band_gradient = gradient_image_RBand + gradient_image_GBand + gradient_image_BBand;
No27 = figure;
imshow(Full_color_band_gradient);
title('Gradient Image Full Band (Sobel 3x3)');

Full_color_band_edge = output_image_RB + output_image_RB + output_image_BB;
No28 = figure;
imshow(Full_color_band_edge);
title('Edge Detected Image Full Band (Sobel 3x3)');
```

Source code: Sobel operator on the three color bands.

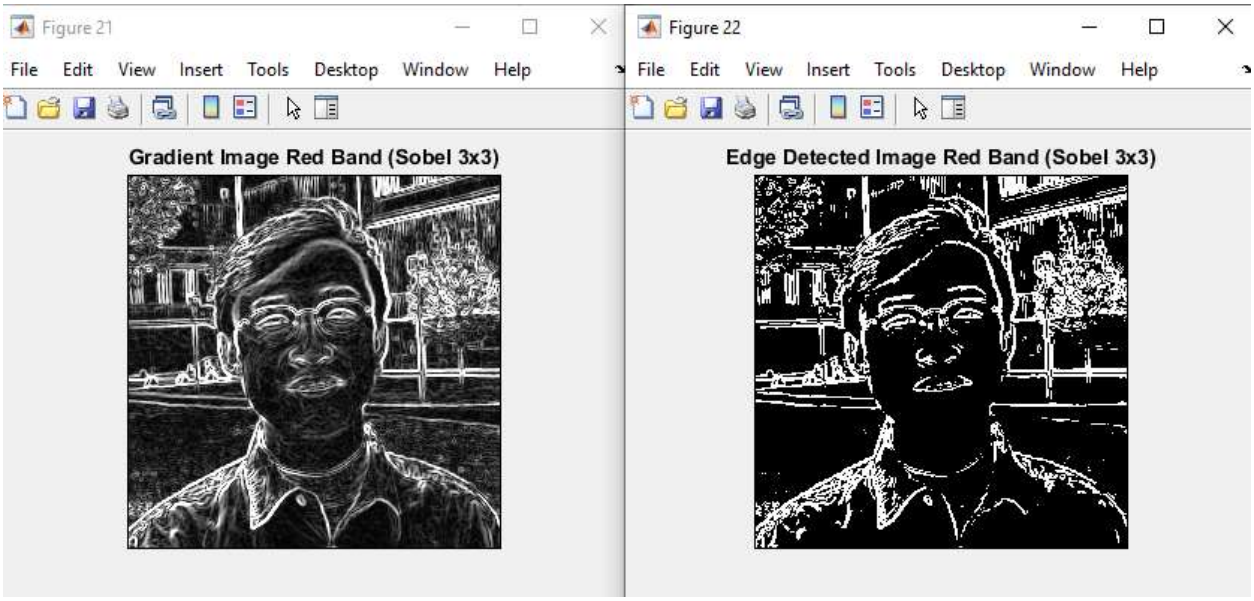


Fig 21 & 22. Gradient and edge image of red band.

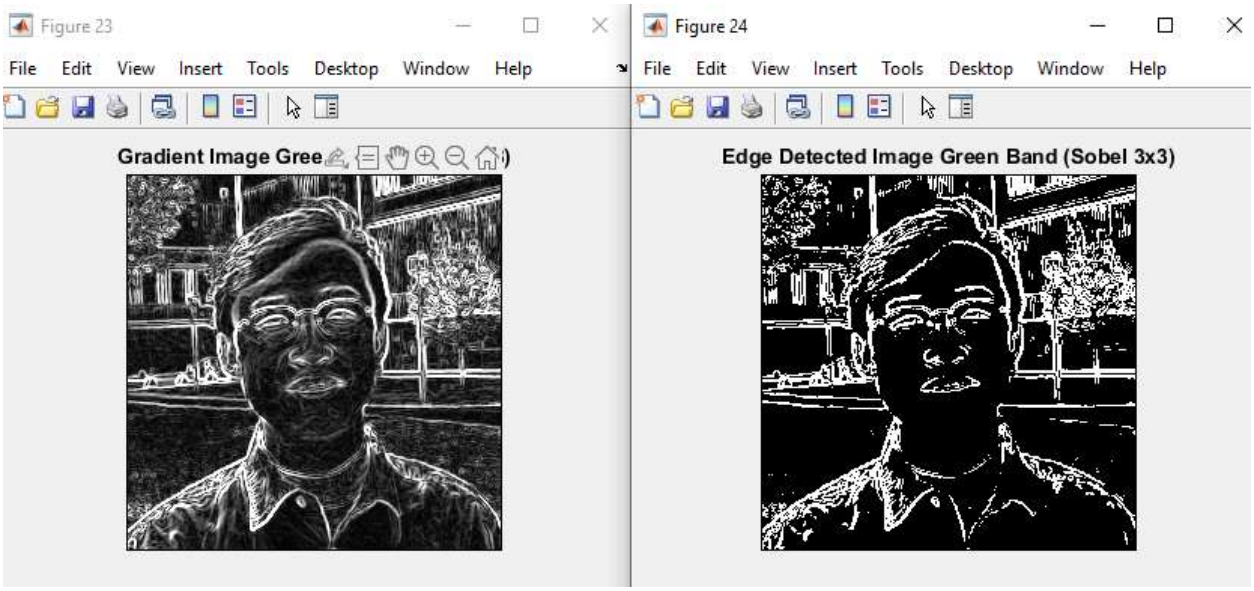


Fig 23 & 24. Gradient and edge image of green band.

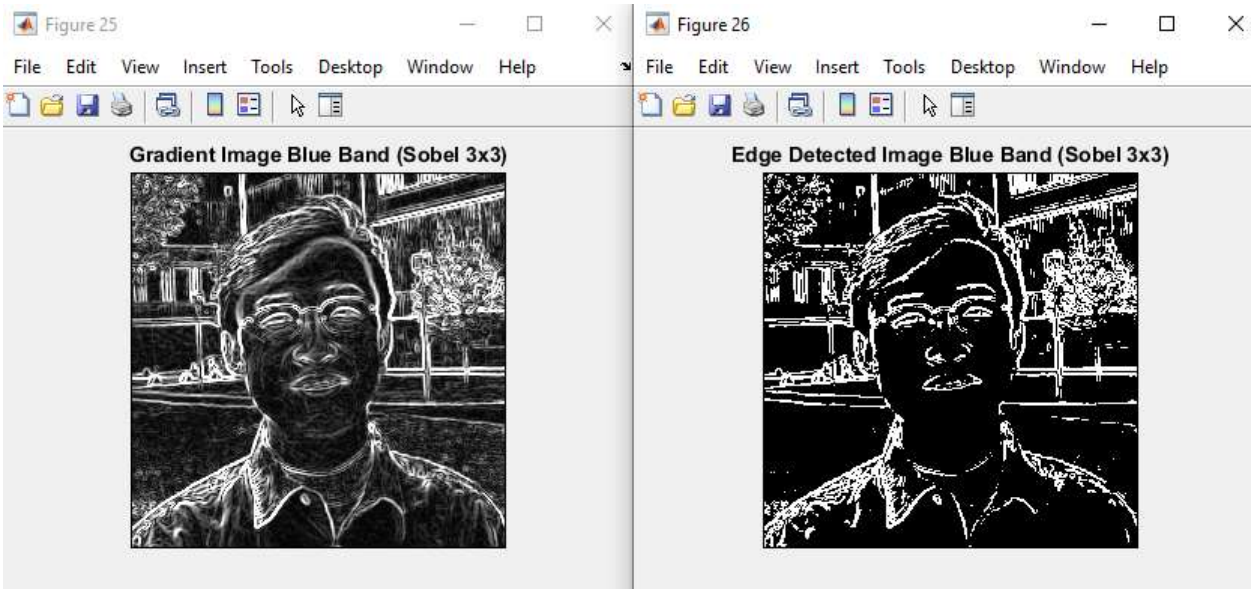


Fig 25 & 26. Gradient and edge image of blue band.

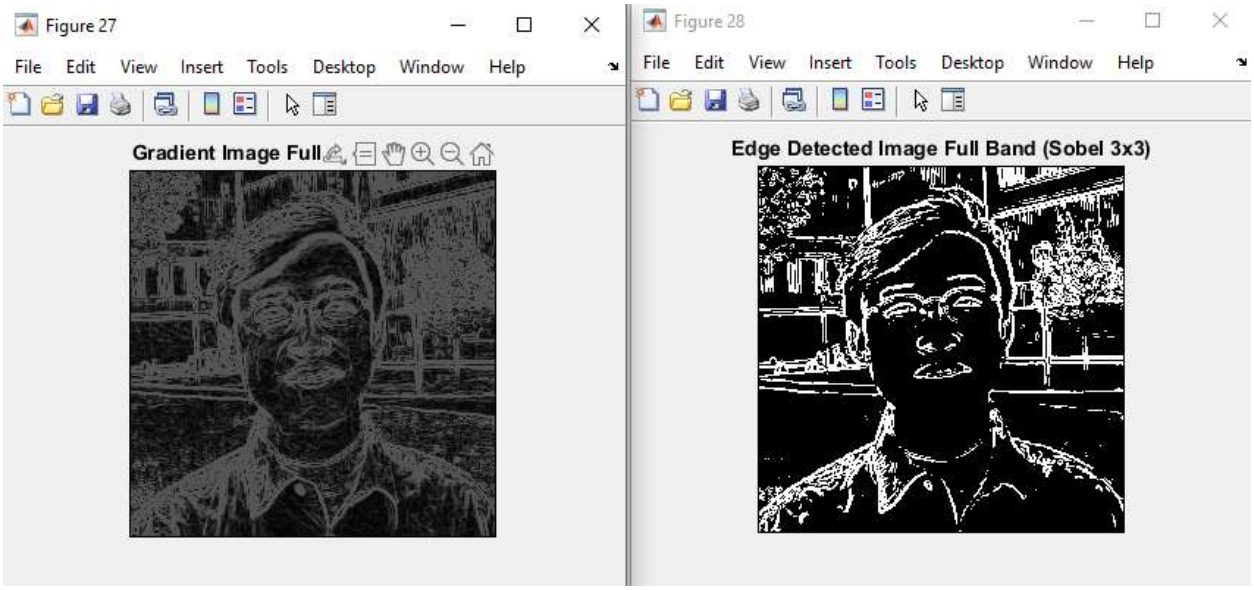


Fig 27 & 28. Combining the three.

Each of the respective RGB bands have a similar grayscale gradient which isn't surprising.

Adding up each of the edges netted a sketch outline.

I honestly struggled a lot in this assignment. Sorry. If there's any extra credit options, I'd love to take them.