CMSC 733: Assignment #2

Due on Thursday, October 15, 2015

Aloimonos, Yiannis 11:00am

Kanishka Ganguly

10/15/2015

Contents

Problem 1

(a)

Rotation of Image

Listing 1: Rotation

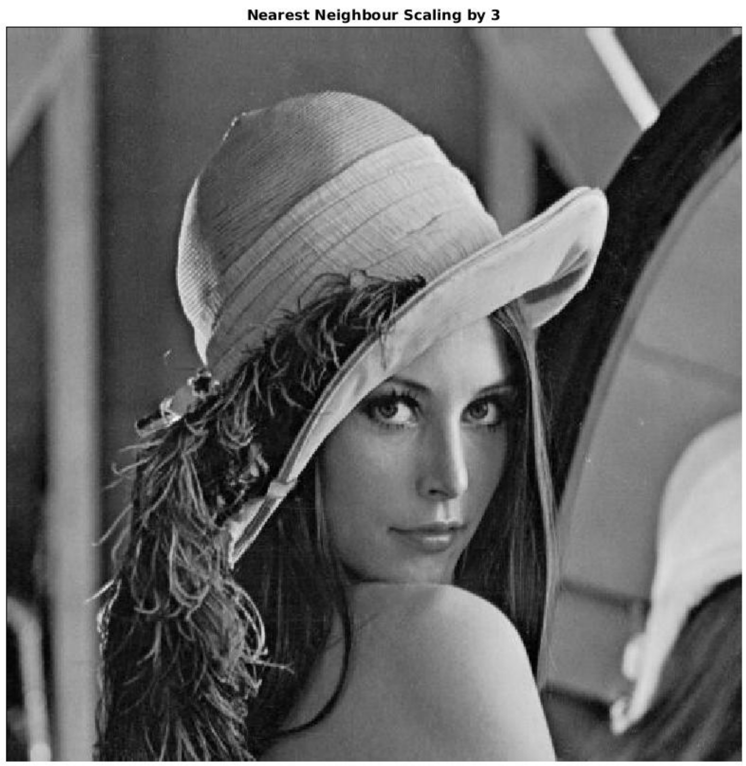
```
clear all
close all
clc

img = imread('lena.jpg');
figure
imshow(img);
title('Original Image');

[rows, cols] = size(img);

center_x = rows/2;
center_y = cols/2;

% Rotation Nearest Neighbour %
new_img = zeros(rows);
angle = 15;
for i = 1:rows
    for j = 1:cols
        x = cosd(angle) * (i-center_x) - sind(angle)*(j-center_y) + center_x;
        y = sind(angle) * (i-center_x) + cosd(angle)*(j-center_y) + center_y;
        x_2 = round(x);
        y_2 = round(y);
```

Problem 1 [(a)] continued on next page...

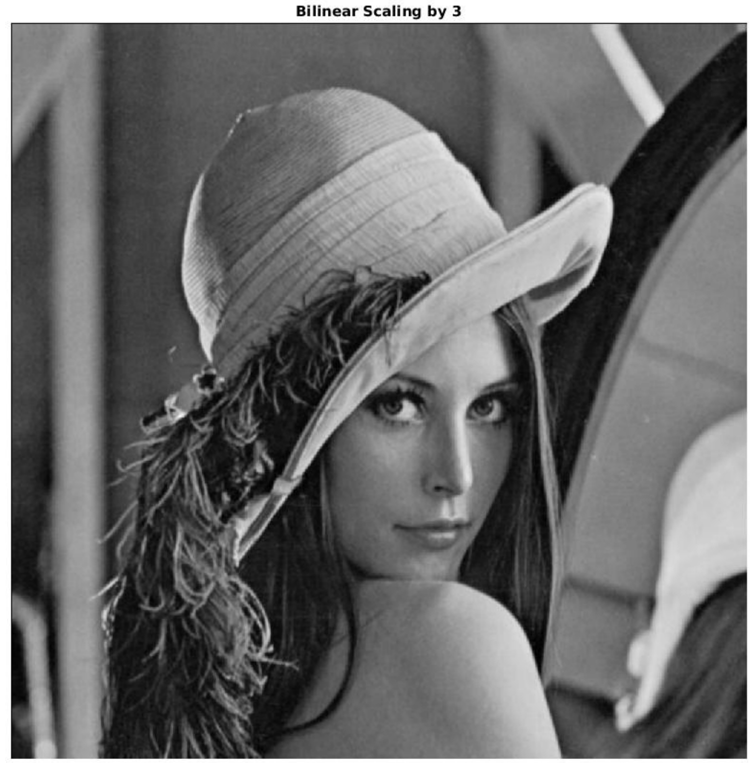
```

        if (x_2 > cols || x_2 < 1)
            continue
        elseif(y_2 > rows || y_2 < 1)
            continue
        else
            new_img(i,j) = img(x_2,y_2);
        end
    end
end
rot_img = uint8(new_img);

% Rotation Bilinear %
angle = 15;
new_img2 = zeros(rows);
for i = 1:rows
    for j = 1:cols
        x = cosd(angle) * (i-center_x) - sind(angle)*(j-center_y) + center_x;
        y = sind(angle) * (i-center_x) + cosd(angle)*(j-center_y) + center_y;

        x1 = floor(x);
        y1 = floor(y);
        x2 = ceil(x);
        y2 = ceil(y);

        if(x1 > cols || x1 < 1)
            continue
        elseif(x2 > cols || x2 < 1)
            continue
        elseif(y1 > cols || y1 < 1)
            continue
        elseif(y2 > cols || y2 < 1)
            continue
        else
            I1 = (x-x1)*img(x2,y1);
            I2 = (x2-x)*img(x1,y1);
            I1_ = I1 + I2;

            I1 = (x-x1)*img(x2,y2);
            I2 = (x2-x)*img(x1,y2);
            I2_ = I1 + I2;

            I = (y-y1)*I2_ + (y2-y)*I1_;
            new_img2(i,j) = I;
        end
    end
end
rot_img2 = uint8(new_img2);

figure
subplot(1,2,1)
imshow(rot_img);
title('Nearest Neighbour Rotation');
```

```
subplot(1,2,2)
imshow(rot_img2);
title('Bilinear Rotation');
```

(b)



Nearest Neighbour Scaling of Image



Bilinear Scaling of Image

Listing 2: Scaling

```
clear all
close all
clc

img = imread('lena.jpg');

[rows, cols] = size(img);

scale = 3;
scale_mat = ([scale 0; 0 scale]^-1);
rows = round(rows * scale);
cols = round(cols * scale);

% Scaling - Nearest Neighbour %
new_img = zeros(rows,cols);

for i = 1:rows
    for j = 1:cols
        coord = scale_mat*[i;j];
        x_2 = round(coord(1));
        y_2 = round(coord(2));

        if(x_2 > 0 && y_2 > 0)
```

```
            new_img(i,j) = img(x_2,y_2);
        end
    end
end
scale_img = uint8(new_img);

% Scaling Bilinear %
new_img2 = zeros(rows,cols);

for i = 1:rows
    for j = 1:cols
        coord = scale_mat*[i;j];
        x = coord(1);
        y = coord(2);

        x1 = floor(x);
        y1 = floor(y);
        x2 = ceil(x);
        y2 = ceil(y);

        if(x1 > 0 && x2 > 0 && y1 > 0 && y2 > 0)
                I1 = (x-x1)*img(x2,y1);
                I2 = (x2-x)*img(x1,y1);
                I1_ = I1 + I2;

                I1 = (x-x1)*img(x2,y2);
                I2 = (x2-x)*img(x1,y2);
                I2_ = I1 + I2;

                I = (y-y1)*I2_ + (y2-y)*I1_;
                new_img2(i,j) = I;
        end
    end
end
scale_img2 = uint8(new_img2);

figure
imshow(scale_img);
title('Nearest Neighbour Scaling by 3');

figure
imshow(scale_img2);
title('Bilinear Scaling by 3');
```

(c)

Listing 3: Skewing

```
clear all
close all
clc

img = imread('lena.jpg');
```

```
figure
imshow(img);
title('Original Image');

[rows, cols] = size(img);

skew = 5;

% Skewing Nearest Neighbour %
new_img = zeros(rows, cols);
for i = 1:rows
    for j = 1:cols
        x = i + j*tand(skew);
        y = j;
        x_2 = round(x);
        y_2 = round(y);

        if (x_2 > cols || x_2 < 1)
            continue
        elseif(y_2 > rows || y_2 < 1)
            continue
        else
            new_img(i,j) = img(x_2,y_2);
        end
    end
end
skew_img = uint8(new_img);
figure
imshow(skew_img);
title('Nearest Neighbour Skewing');

% Skewing Bilinear %
new_img2 = zeros(rows, cols);
for i = 1:rows
    for j = 1:cols
        x = i + j*tand(skew);
        y = j;

        x1 = floor(x);
        y1 = floor(y);
        x2 = ceil(x);
        y2 = ceil(y);

        if(x1 > cols || x1 < 1)
            continue
        elseif(x2 > cols || x2 < 1)
            continue
        elseif(y1 > cols || y1 < 1)
            continue
        elseif(y2 > cols || y2 < 1)
            continue
        else
            I1 = (x-x1)*img(x2,y1);
```

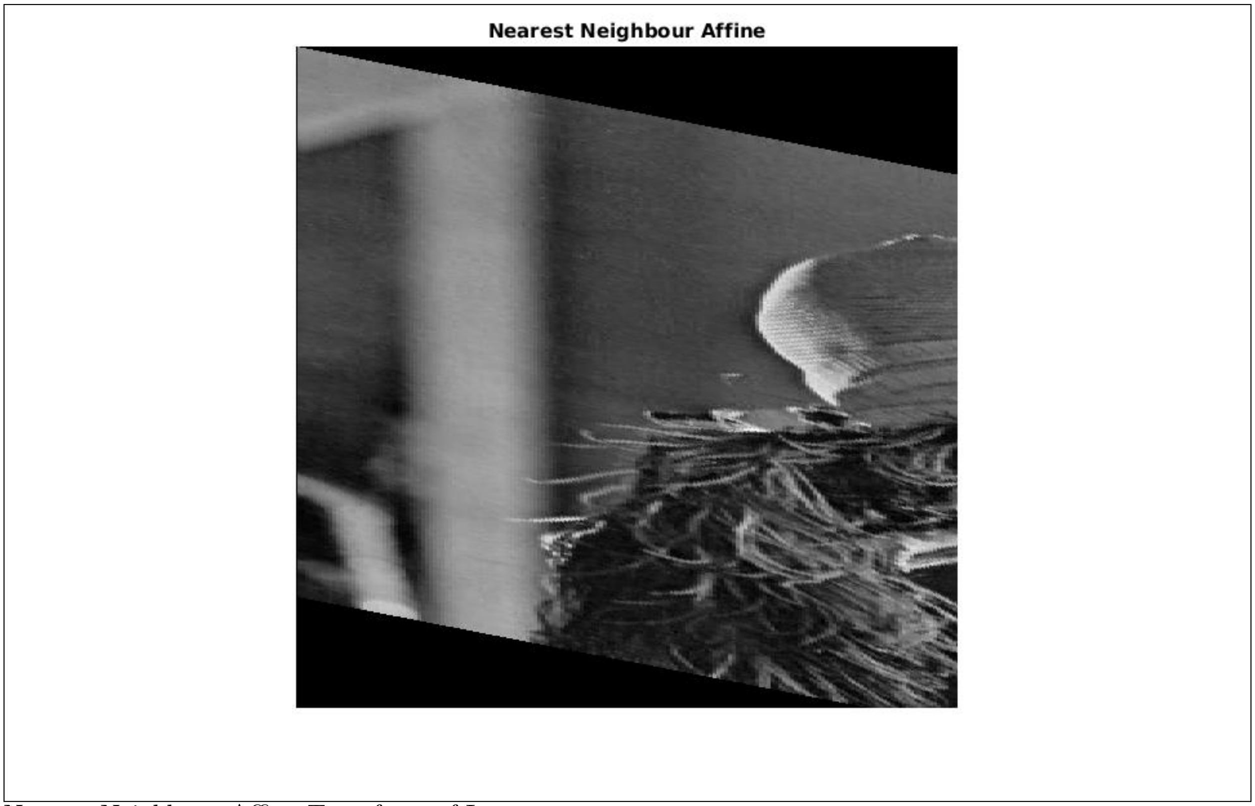
```
            I2 = (x2-x)*img(x1,y1);
            I1_ = I1 + I2;

            I1 = (x-x1)*img(x2,y2);
            I2 = (x2-x)*img(x1,y2);
            I2_ = I1 + I2;

            I = (y-y1)*I2_ + (y2-y)*I1_;
            new_img2(i,j) = I;
        end
    end
end
skew_img2 = uint8(new_img2);
figure
imshow(skew_img2);
title('Bilinear Skewing');
```

(d)

Listing 4: Affine Transformation

```
clear all
close all
clc

img = imread('lena.jpg');
figure
imshow(img);
title('Original Image');

[rows, cols] = size(img);

T = [1.9 -0.5 0; 0 0 1; 1 1 1] * [1 -1 0; 0 0 3; 1 1 1]^-1

% Affine - Nearest Neighbour %
new_img = zeros(rows,cols);

for i = 1:rows
    for j = 1:cols
        coord = T(1:2,1:2)*[i;j];
        x_2 = round(coord(1));
        y_2 = round(coord(2));

        if(x_2 > 0 && x_2 < rows && y_2 > 0 && y_2 < cols)
            new_img(i,j) = img(x_2,y_2);
        end
    end
end
affine_img = uint8(new_img);
figure
imshow(affine_img);
title('Nearest Neighbour Affine');
```

```

% Affine - Bilinear %
new_img2 = zeros(rows,cols);

for i = 1:rows
    for j = 1:cols
        coord = T(1:2,1:2)*[i;j];
        x = coord(1);
        y = coord(2);

        x1 = floor(x);
        y1 = floor(y);
        x2 = ceil(x);
        y2 = ceil(y);

        if(x1 > 0 && x2 > 0 && y1 > 0 && y2 > 0 && x1 < rows && y1 < cols && x2 < rows && y2 < cols)
                I1 = (x-x1)*img(x2,y1);
                I2 = (x2-x)*img(x1,y1);
                I1_ = I1 + I2;

                I1 = (x-x1)*img(x2,y2);
                I2 = (x2-x)*img(x1,y2);
                I2_ = I1 + I2;

                I = (y-y1)*I2_ + (y2-y)*I1_;
                new_img2(i,j) = I;
        end
    end
end
affine_img2 = uint8(new_img2);
figure
imshow(affine_img2);
title('Bilinear Affine');
```



Nearest Neighbour Affine Transform of Image

Bilinear Affine Transform of Image

Problem 2

Given Image Patch as

$$\begin{bmatrix} 9 & 10 & 9 & 4 & 3 \\ 5 & 7 & 8 & 9 & 3 \\ 4 & 5 & 6 & 8 & 5 \\ 3 & 4 & 5 & 6 & 8 \\ 2 & 3 & 4 & 5 & 6 \end{bmatrix} \tag{1}$$

(a)

Listing 5: 3x3 Gaussian Filter

```matlab
clear all;
close all;
clc;

img = [9 10 9 4 3; 5 7 8 9 3; 4 5 6 8 5; 3 4 5 6 8; 2 3 4 5 6]

% Gaussian Filter %
gaussian = (1/16)*[1 2 1; 2 4 2; 1 2 1];

center_img = img(2:4,2:4);

for i= 1:3
    for j = 1:3
        center_img(i,j) = center_img(i,j) * gaussian(i,j);
    end
end
filter_val = 0;
filter_val = filter_val + sum(center_img(:,1)) + sum(center_img(:,2)) +
    sum(center_img(:,3));
gauss_img = img;
gauss_img(3,3) = filter_val;
gauss_img
```

Listing 6: 3x3 Box Filter

```matlab
clear all;
close all;
clc;

img = [9 10 9 4 3; 5 7 8 9 3; 4 5 6 8 5; 3 4 5 6 8; 2 3 4 5 6]

% Box Filter %
center_img = img(2:4,2:4);
avg = (sum(center_img(:,1)) + sum(center_img(:,2)) + sum(center_img(:,3)))/9;
box_img = img;
box_img(3,3) = avg;
box_img
```

Problem 2 continued on next page. . .

(b)

Listing 7: Sobel Edge Detector

```
clear all;
close all;
clc;

img = [9 10 9 4 3; 5 7 8 9 3; 4 5 6 8 5; 3 4 5 6 8; 2 3 4 5 6];
sobel_x = (1/8)*[-1 0 1; -2 0 2; -1 0 1];
sobel_y = (1/8)*[1 2 1; 0 0 0; -1 -2 -1];

center_img = img(2:4,2:4);

for i= 1:3
    for j = 1:3
        center_img(i,j) = center_img(i,j) * sobel_x(i,j);
    end
end
sobel_x_val = sum(center_img(:,1)) + sum(center_img(:,2)) + sum(center_img(:,3));

center_img = img(2:4,2:4);
for i= 1:3
    for j = 1:3
        center_img(i,j) = center_img(i,j) * sobel_y(i,j);
    end
end
sobel_y_val = sum(center_img(:,1)) + sum(center_img(:,2)) + sum(center_img(:,3));

edge_direction = atand(sobel_y_val/sobel_x_val)
edge_strength = sqrt(sobel_x_val^2 + sobel_y_val^2)
```

We get the following output:

```
edge_direction = 50.1944
edge_strength = 1.9526
```

(c)

Listing 8: Median Filter

```
clear all;
close all;
clc;

img = [9 10 9 4 3; 5 7 8 9 3; 4 5 6 8 5; 3 4 5 6 8; 2 3 4 5 6]

% Median Filter %
center_img = img(2:4,2:4);
arr = center_img(:);
median = median(arr);
median_img = img;
median_img(3,3) = median;
median_img
```

Problem 2 [(c)] continued on next page...

Median filtering has been shown to give best results for salt-and-pepper noise. Median filtering has the added advantage (over mean filters) of removing noise while preserving edges in an image (under certain conditions).

- Since the median is a more robust averaging technique than the mean, a single unrepresentative pixel in a neighbourhood will not affect the median value significantly. Thus, it is robust to the presence of outliers.
- Also, since the median value must actually be the value of one of the pixels in the neighborhood, the median filter does not create new unrealistic pixel values when the filter straddles an edge. For this reason the median filter is much better at preserving sharp edges than the mean filter.

(d)

Why the Gaussian is a good smoothing filter

- A Gaussian filter is a better smoothing filter (as compared to the mean filter) because the Gaussian filter outputs a 'weighted average' of each pixel's neighbourhood, where the average is weighted based on the distance each neighbouring pixel is away from the central pixel. So, pixels directly neighbouring the central pixel is weighted more than the ones farther away from the central pixel. This gives a much 'gentler' smoothing and preserves edges better, as compared to a mean filter's uniformly weighted averaging technique.
- Also, the Gaussian filter has a better *frequency response* than the box (mean) filter. While both filters remove high spatial frequency components from an image, the Gaussian filter shows little to no oscillations in its frequency response as compared to the mean filter, which shows several oscillations.

To perform Gaussian filtering fast

- Gaussian filter has the property that it is separable, i.e. we can express the 2-D convolution as two 1-D convolutions. Thus, two 1-D convolutions can be performed in lesser time than one 2-D convolutions.
- If the filter is large, we can use the fact that a convolution in the spatial domain is equivalent to a multiplication in the frequency (Fourier) domain. This works well for multiple images, since we need to take the Fourier Transform of the Gaussian filter only once. Also, the set of Gaussian functions is closed under Fourier transforms, which means that the Fourier transform of a Gaussian is another Gaussian.

(e)

If the Gaussian kernel is larger than the width of the line, then after smoothing, the width of the line is reduced, i.e. the line becomes 'thinner'. Also, at the edges of the line, we can see a 'tapering' effect.

(f)

A box filter attenuates the noise because on taking the mean value of neighbouring pixels, the resulting output reduces the 'variance' of the noise.
That is, the variance of noise in the mean is smaller than variance of the pixel noise.

Problem 3

We have a 1-D image as follows:

$$I(x) = cos(x/100) \tag{2}$$

When we apply gradient smoothing to $I(x)$, there is no effect on the intensity, which remains unchanged. Thus, $I_{smooth} = cos(x/100)$.

On this smoothed image, when we calculate the derivatives, we use the mask $[-101]$. We see that the maximum absolute value (magnitude) of the derivative is obtained at $x = 100[\pi/2 + n\pi]$. This is because the derivative of $cos(x)$ is $-sin(x)$, which is maximum at $\pi/2 + n\pi$. We can thus find the edges for the image at the zero-crossings of $cos(x/100)$.

Now, we see that the effect of increasing the σ of the Gaussian filter doesn't affect the image since the intensities remain unchanged even after smoothing.
Also, an increase in the threshold value doesn't have any effect on the local maxima of the derivative since the maximum absolute value is still obtained at $x = 100[\pi/2 + n\pi]$.

Problem 4

Let the given image I be

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & \mathbf{7} & 8 & 9 \\ 7 & 8 & 9 & 10 & 11 \\ 9 & 10 & 11 & 12 & 13 \end{bmatrix} \tag{3}$$

(a)

For the gradient in the x-direction, we design a filter $K = \frac{1}{2}[-1 \quad 0 \quad 1]$. Upon convolution with $f(x, y)$ we get $\frac{1}{2}[f(x+1, y) - f(x-1, y)]$ which is the gradient along the x-direction.

$$\therefore gradient_x = \frac{8-6}{2} = 1 \tag{4}$$

Similarly, to compute the gradient along the y-direction, we choose the filter $M = [-1 \quad 0 \quad 1]^T$.

$$\therefore gradient_y = \frac{9-5}{2} = 2 \tag{5}$$

The resultant gradient is given by:

$$gradient_total = \sqrt{gradient_x^2 + gradient_y^2} = \sqrt{5} \tag{6}$$

To calculate the direction, we have $tan^{-1}(2)^{\circ}$ with x-axis or 63.44°.

(b)

It is given that gradient at point $(3, 7)$ is $(3, -2)$ and the intensity at that point is 17.
We know that the gradient is defined as the change of intensity per unit distance, i.e. per pixel.
Thus we have

$$I(3.1, 7.3) = I(3, 7) + G_x(0.1) + G_y(0.3) \tag{7}$$

$$\implies 17 + 0.3 - 0.6 = 16.7 \tag{8}$$

(c)

It is given that $I(x, y) = (x-5)^2 + (y-1)^2$
At (x, y),

$$gradient_x = \frac{1}{2}[I(x+1, y) - I(x-1, y)] = -4 \tag{9}$$

$$gradient_y = \frac{1}{2}[I(x, y+1) - I(x, y-1)] = 2 \tag{10}$$

$$\tag{11}$$

Thus we have gradient at $(3, 2)$ as $(-4, 2)$.

Problem 5

NOT ATTEMPTED

Problem 6

Listing 9: Main Routine

```
clear all;
close all;
clc;

myDir = 'hist_img/';
ext_img = '*.jpg';
a = dir([myDir ext_img]);
nfile = max(size(a));
for i=1:nfile
    my_img1(i).img = imread([myDir a(i).name]);
    my_img2(i).img = imread([myDir a(i).name]);
end

for i=1:6
    for j=(i+1):6
        h1 = Q6_myColorHist(my_img1(i).img,0);
        h2 = Q6_myColorHist(my_img2(j).img,0);
        ssd = Q6_histDist(h1, h2);
        ssd_red_table(i,j) = ssd(1,:);
        ssd_green_table(i,j) = ssd(2,:);
        ssd_blue_table(i,j) = ssd(3,:);
        ssd_red_table(j,i) = ssd(1,:);
        ssd_green_table(j,i) = ssd(2,:);
        ssd_blue_table(j,i) = ssd(3,:);
    end
end

% Semantic Similarity In Percentages Between Images %
similarity = Q6_similarity(ssd_red_table,ssd_green_table, ssd_blue_table);

images_list = {'desert1', 'desert2', 'desert3', 'forest1', 'forest2', 'forest3'};
array2table(similarity, 'VariableNames', images_list, 'RowNames', images_list)
```

Listing 10: Function *myColorHist(I)*

```
function [ hist ] = Q6_myColorHist( img, show_hist )
[rows, cols] = size(img);

img_red = img(:,:,1);
img_green = img(:,:,2);
img_blue = img(:,:,3);

bin_red = zeros(1,8);
bin_green = zeros(1,8);
bin_blue = zeros(1,8);
```

```

for(i=1:rows)
    for(j=1:cols/3)
        if(img_red(i,j) >= 1 && img_red(i,j) <= 32)
            bin_red(1) = bin_red(1) + 1;
        elseif(img_red(i,j) >= 33 && img_red(i,j) <= 64)
            bin_red(2) = bin_red(2) + 1;
        elseif(img_red(i,j) >= 65 && img_red(i,j) <= 96)
            bin_red(3) = bin_red(3) + 1;
        elseif(img_red(i,j) >= 97 && img_red(i,j) <= 128)
            bin_red(4) = bin_red(4) + 1;
        elseif(img_red(i,j) >= 129 && img_red(i,j) <= 160)
            bin_red(5) = bin_red(5) + 1;
        elseif(img_red(i,j) >= 161 && img_red(i,j) <= 192)
            bin_red(6) = bin_red(6) + 1;
        elseif(img_red(i,j) >= 193 && img_red(i,j) <= 224)
            bin_red(7) = bin_red(7) + 1;
        elseif(img_red(i,j) >= 225 && img_red(i,j) <= 255)
            bin_red(8) = bin_red(8) + 1;
        end
    end
end

for(i=1:rows)
    for(j=1:cols/3)
        if(img_green(i,j) >= 1 && img_green(i,j) <= 32)
            bin_green(1) = bin_green(1) + 1;
        elseif(img_green(i,j) >= 33 && img_green(i,j) <= 64)
            bin_green(2) = bin_green(2) + 1;
        elseif(img_green(i,j) >= 65 && img_green(i,j) <= 96)
            bin_green(3) = bin_green(3) + 1;
        elseif(img_green(i,j) >= 97 && img_green(i,j) <= 128)
            bin_green(4) = bin_green(4) + 1;
        elseif(img_green(i,j) >= 129 && img_green(i,j) <= 160)
            bin_green(5) = bin_green(5) + 1;
        elseif(img_green(i,j) >= 161 && img_green(i,j) <= 192)
            bin_green(6) = bin_green(6) + 1;
        elseif(img_green(i,j) >= 193 && img_green(i,j) <= 224)
            bin_green(7) = bin_green(7) + 1;
        elseif(img_green(i,j) >= 225 && img_green(i,j) <= 255)
            bin_green(8) = bin_green(8) + 1;
        end
    end
end

for(i=1:rows)
    for(j=1:cols/3)
        if(img_blue(i,j) >= 1 && img_blue(i,j) <= 32)
            bin_blue(1) = bin_blue(1) + 1;
        elseif(img_blue(i,j) >= 33 && img_blue(i,j) <= 64)
            bin_blue(2) = bin_blue(2) + 1;
        elseif(img_blue(i,j) >= 65 && img_blue(i,j) <= 96)
            bin_blue(3) = bin_blue(3) + 1;
```

```
        elseif(img_blue(i,j) >= 97 && img_blue(i,j) <= 128)
            bin_blue(4) = bin_blue(4) + 1;
        elseif(img_blue(i,j) >= 129 && img_blue(i,j) <= 160)
            bin_blue(5) = bin_blue(5) + 1;
        elseif(img_blue(i,j) >= 161 && img_blue(i,j) <= 192)
            bin_blue(6) = bin_blue(6) + 1;
        elseif(img_blue(i,j) >= 193 && img_blue(i,j) <= 224)
            bin_blue(7) = bin_blue(7) + 1;
        elseif(img_blue(i,j) >= 225 && img_blue(i,j) <= 255)
            bin_blue(8) = bin_blue(8) + 1;
        end
    end
end

% Normalization Stage %
sum_red = sum(bin_red);
bin_red = bin_red/sum_red;
sum_green = sum(bin_green);
bin_green = bin_green/sum_green;
sum_blue = sum(bin_blue);
bin_blue = bin_blue/sum_blue;
hist = [bin_red;bin_green;bin_blue];

% Histogram Visualization %
if show_hist == 1
    hist_plot=figure('Position', [100, 100, 3000, 3000]);
    subplot(1,3,1)
    bar(bin_red)
    set(gca,'XTickLabel',{'1-32', '33-64', '65-96', '97-128','129-160',
        '161-192', '193-224', '225-255'},'FontSize',8)
    title('Red Channel Histogram')
    xlabel('Intensity')
    ylabel('Pixel Count')

    subplot(1,3,2)
    bar(bin_green)
    set(gca,'XTickLabel',{'1-32', '33-64', '65-96', '97-128','129-160',
        '161-192', '193-224', '225-255'},'FontSize',8)
    title('Green Channel Histogram')
    xlabel('Intensity')
    ylabel('Pixel Count')

    subplot(1,3,3)
    bar(bin_blue)
    set(gca,'XTickLabel',{'1-32', '33-64', '65-96', '97-128','129-160',
        '161-192', '193-224', '225-255'},'FontSize',8)
    title('Blue Channel Histogram')
    xlabel('Intensity')
    ylabel('Pixel Count')
end
end
```

Listing 11: Function *histDist(h1, h2)*

Problem 6 continued on next page. . .

```
function [ ssd ] = Q6_histDist( h1, h2 )
ssd_red = sum((h1(1,:)-h2(1,:)).^2);
ssd_green = sum((h1(2,:)-h2(2,:)).^2);
ssd_blue = sum((h1(3,:)-h2(3,:)).^2);
ssd = [ssd_red;ssd_green;ssd_blue];
end
```

For extra credit, I have developed a function that returns a percentage similarity between images that can be used to determine semantic similarity between scenes.

Listing 12: Function *similarity(ssdR, ssdG, ssdB)*

```
function [ similarity ] = Q6_similarity( ssd_red, ssd_green, ssd_blue )

ssd_red_percent = 1 - ssd_red;
ssd_green_percent = 1 - ssd_green;
ssd_blue_percent = 1 - ssd_blue;

similarity = bsxfun(@times, ssd_blue_percent, bsxfun(@times, ssd_red_percent,
    ssd_green_percent));
similarity = similarity * 100;
end
```

ans =

	desert1	desert2	desert3	forest1	forest2	forest3
desert1	100	66.418	89.814	47.336	71.69	64.536
desert2	66.418	100	56.619	24.911	49.945	46.145
desert3	89.814	56.619	100	52.115	65.898	62.29
forest1	47.336	24.911	52.115	100	57.396	76.786
forest2	71.69	49.945	65.898	57.396	100	71.908
forest3	64.536	46.145	62.29	76.786	71.908	100

Percentage Matrix for Semantic Similarity Determination Between Scenes

From the individual histograms of the Red, Green and Blue channels of two images, it may be determined whether two images are 'similar' in which channel. This, combined with a precomputed set of 'generic' values for each type of scene can be used to determine and differentiate between two scenes, such as a desert and a forest.

However, a more intelligent and adaptive algorithm than the one presented here is required, since it is evident from the values that in some cases, the presences of certain 'unnecessary' features can dominate a histogram output.

For example, in a desert and a forest scene, if the 'amount' of blue skies that are present dominate the 'actual' desert and forest portions, then there would be a higher similarity being returned than what is expected.

Thus, along with color histograms, it might be necessary and useful to include other classification algorithms to detect 'features' such as sand and trees in desert and forest scenes respectively.

Problem 7

NOT ATTEMPTED

Problem 8

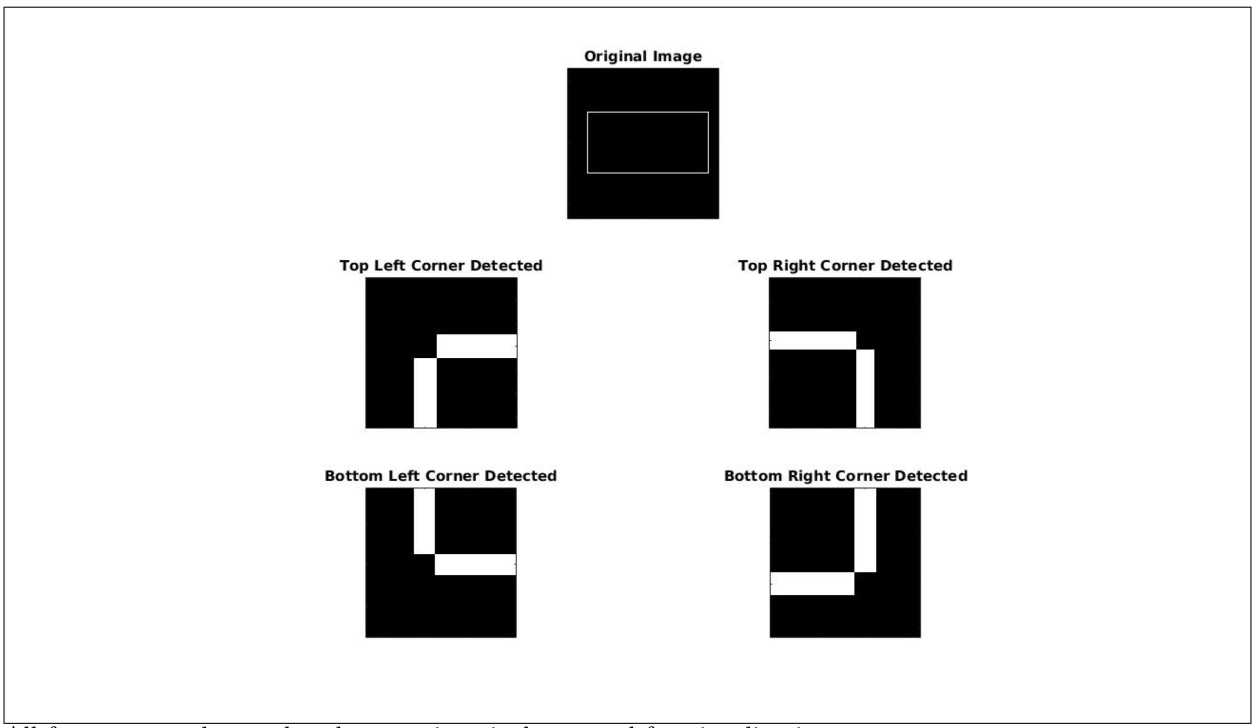
Listing 13: Corner Detection

```
clear all;
close all;
clc;

img = zeros(500, 500);
img(100:200,100) = 255;
img(100:200,300) = 255;
img(100,100:300) = 255;
img(200,100:300) = 255;
img_show = uint8(img);

figure
subplot(3,2,[1,2])
subimage(img_show)
set(gca,'XtickLabel',[],'YtickLabel',[]);
title('Original Image');

top_left_mask = [0 0 0 0 0; 0 0 0 0 0; 0 0 1 1 1; 0 0 1 0 0; 0 0 1 0 0];
top_right_mask = [0 0 0 0 0; 0 0 0 0 0; 1 1 1 0 0; 0 0 1 0 0; 0 0 1 0 0];
bottom_left_mask = [0 0 1 0 0; 0 0 1 0 0; 0 0 1 1 1; 0 0 0 0 0; 0 0 0 0 0];
bottom_right_mask = [0 0 1 0 0; 0 0 1 0 0; 1 1 1 0 0; 0 0 0 0 0; 0 0 0 0 0];

[rows, cols] = size(img);

baseline = 0;

for i=1:rows
    for j=1:cols
        if(i == 1 && j == 1)
            edge = bsxfun(@times,img(i:i+4,j:j+4),top_left_mask);
            baseline = edge(3,3);
        end
        if i+4 < rows && j+4 < cols
            edge = bsxfun(@times,img(i:i+4,j:j+4),top_left_mask);
            if edge(3,3) == edge(3,4)
                if edge(3,4) == edge(3,5)
                    if edge(3,5) == edge(4,3)
                        if edge(4,3) == edge(5,3)
                            if(edge(3,3) ~= baseline)
                                img(i+2,j+2) = 0;
                                break;
                            else
                                continue;
                            end
                        end
```

```
                    end
                end
            end
        end
    end
end
img_show = uint8(img);
subplot(3,2,3)
subimage(img_show)
set(gca,'XtickLabel',[],'YtickLabel',[]);
title('Top Left Corner Detected');

img = zeros(500, 500);
img(100:200,100) = 255;
img(100:200,300) = 255;
img(100,100:300) = 255;
img(200,100:300) = 255;

for i=1:rows
    for j=1:cols
        if(i == 1 && j == 1)
            edge = bsxfun(@times,img(i:i+4,j:j+4),top_right_mask);
            baseline = edge(3,3);
        end
        if i+4 < rows && j+4 < cols
            edge = bsxfun(@times,img(i:i+4,j:j+4),top_right_mask);
            if edge(3,3) == edge(3,2)
                if edge(3,2) == edge(3,1)
                    if edge(3,1) == edge(4,3)
                        if edge(4,3) == edge(5,3)
                            if(edge(3,3) ~= baseline)
                                img(i+2,j+2) = 0;
                                break;
                            else
                                continue;
                            end
                        end
                    end
                end
            end
        end
    end
end
img_show = uint8(img);
subplot(3,2,4)
subimage(img_show)
set(gca,'XtickLabel',[],'YtickLabel',[]);
title('Top Right Corner Detected');

img = zeros(500, 500);
img(100:200,100) = 255;
img(100:200,300) = 255;
img(100,100:300) = 255;
```

```
img(200,100:300) = 255;

for i=1:rows
    for j=1:cols
        if(i == 1 && j == 1)
            edge = bsxfun(@times,img(i:i+4,j:j+4),top_right_mask);
            baseline = edge(3,3);
        end
        if i+4 < rows && j+4 < cols
            edge = bsxfun(@times,img(i:i+4,j:j+4),bottom_left_mask);
            if edge(3,3) == edge(2,3)
                if edge(2,3) == edge(1,3)
                    if edge(1,3) == edge(3,4)
                        if edge(3,4) == edge(3,5)
                            if(edge(3,3) ~= baseline)
                                img(i+2,j+2) = 0;
                                break;
                            else
                                continue;
                            end
                        end
                    end
                end
            end
        end
    end
end
img_show = uint8(img);
subplot(3,2,5)
subimage(img_show)
set(gca,'XtickLabel',[],'YtickLabel',[]);
title('Bottom Left Corner Detected');
figure
imshow(img_show)

img = zeros(500, 500);
img(100:200,100) = 255;
img(100:200,300) = 255;
img(100,100:300) = 255;
img(200,100:300) = 255;

for i=1:rows
    for j=1:cols
        if(i == 1 && j == 1)
            edge = bsxfun(@times,img(i:i+4,j:j+4),bottom_right_mask);
            baseline = edge(3,3);
        end
        if i+4 < rows && j+4 < cols
            edge = bsxfun(@times,img(i:i+4,j:j+4),bottom_right_mask);
            if edge(3,3) == edge(3,2)
                if edge(3,2) == edge(3,1)
                    if edge(3,1) == edge(2,3)
                        if edge(2,3) == edge(1,3)
```

```
                            if(edge(3,3) ~= baseline)
                                img(i+2,j+2) = 0;
                                break;
                            else
                                continue;
                            end
                        end
                    end
                end
            end
        end
    end
end
img_show = uint8(img);
subplot(3,2,6)
subimage(img_show)
set(gca,'XtickLabel',[],'YtickLabel',[]);
title('Bottom Right Corner Detected');
figure
imshow(img_show)
```



All four corners detected and respective pixel removed for visualization

The algorithm proposed and demonstrated above uses the following masks:

$$\text{Top Left Detector} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{12}$$

$$\text{Top Right Detector} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{13}$$

$$\text{Bottom Left Detector} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{14}$$

$$\text{Bottom Right Detector} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{15}$$

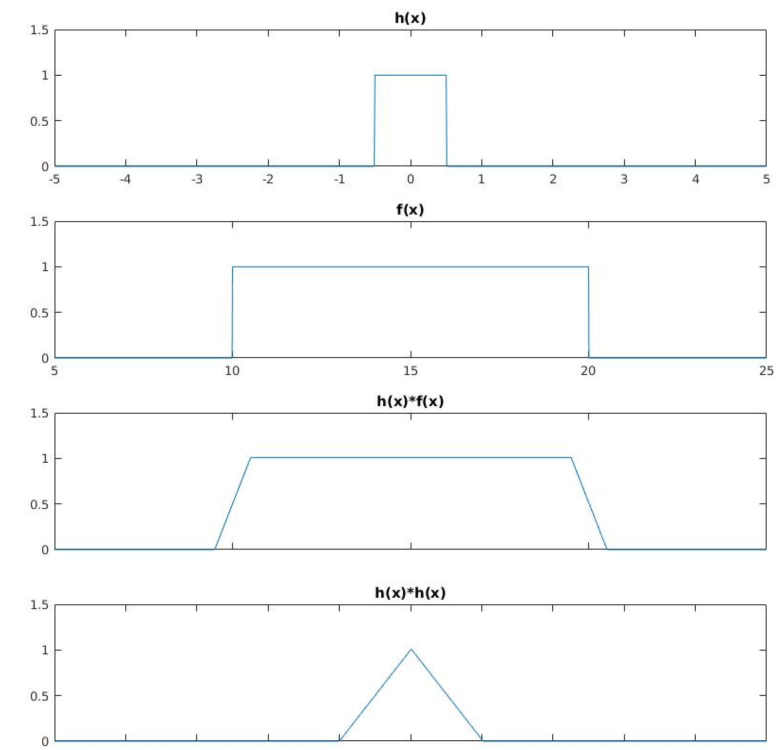
$$\tag{16}$$

The algorithm works as follows:

1. Pad the image to make **ROWS** and **COLS** multiples of 5
2. Place respective mask over image, starting from top-left corner
   (a) Multiply mask element with image patch
   (b) Calculate value for element $(3, 3)$ and store as **BASELINE**
3. Traverse mask over image from **ROWS** to **COLS**
   (a) Multiply mask element with image patch
   (b) Check if all non-zero elements of matrix are equal.
   (c) IF EQUAL, compare value of non-zero element with **BASELINE**.
   (d) IF EQUAL, corner is found. Break loop.
4. ELSE, REPEAT.

This algorithm makes the assumption that the rectangle is made of lines which are 1 pixel thick and that the background is of uniform intensity.
It also assumes that the rectangle is anywhere within the bounds of the image and not touching any of the boundaries of the image.
This is a fairly robust algorithm since it can detect corners of multiple rectangles in the same image. Also, the entire algorithm can detect all four corners of a rectangle in one iteration, by placing all four masks in the same loop routine.
The main drawback of said algorithm is with boundary cases where any one edge of the rectangle is touching the edge of the image.

Problem 9



Figures 1. $h(x)$ 2. $f(x)$ 3. $f(x) * h(x)$ 4. $h(x) * h(x)$

Problem 10

(a)

Most images taken in a real-world situation (using normal cameras) have an inherent noise. Also, these images have several features that contain edges.
Taking derivates using finite differences is an operation that reduces the noise in the image. Also, using a smoothing filter such as a Gaussian causes these edges to blur, thus solving the problem of inhibiting the local edges that are unimportant to the analysis of the image, while retaining the more prominent, useful edges.

(b)

In the former implementation, i.e. convolving with Gaussian first and then taking the Laplacian, we first apply the Gaussian filter to the image using a discrete convolution and then apply the Laplacian operator. This requires two computations to the image, which decreases efficiency since the Gaussian filter needs to be applied to each image that we may have.
However, in the latter implementation, i.e. Laplacian of Gaussian, we can precompute the effect of the Laplacian on the Gaussian filter and then store and apply the resulting filter to the image. This reduces the computation time to just one major calculation, irrespective of the number of images we may have.

(c)

We can show that the Laplacian of the Gaussian is $G_1 * I - G_2 * I$ by taking

$$LoG(x, y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2 + y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{17}$$

This equation can be approximated by using the difference of two Gaussian kernels with suitable values for σ. This ratio turns out to be $1 : 1.6$ for the best results.

$$DoG = G_{\sigma_1} - G_{\sigma_2} \tag{18}$$

$$DoG = \frac{1}{2\pi}\left(\frac{1}{\sigma_1}e^{-\frac{x^2+y^2}{2\sigma_2^1}} - \frac{1}{\sigma_2}e^{-\frac{x^2+y^2}{2\sigma_2^2}}\right) \tag{19}$$

(d)

Difference of Gaussians (DoG) is more efficient than Laplacian of Gaussian (LoG) because one 2-D Gaussian operation is separable into two 1-D Gaussian operations, which is much more computationally efficient. Also, when transformed to the frequency domain, a Gaussian convolution operation becomes a multiplication operation. This also makes it more computationally efficient to calculate (DoG) than (LoG).

Problem 11

Listing 14: Unsharp Sharpening

```
clear all;
close all;
clc;

img = imread('lena.jpg');
scaling_constant = 0.7;

% Standard Kernel Size %
box = imboxfilt(img,3);
gauss = imgaussfilt(img,2);

img_box_subtract = img - box;
img_gauss_subtract = img - gauss;

img_box_sharpen = img + (scaling_constant * img_box_subtract);
img_gauss_sharpen = img + (scaling_constant * img_gauss_subtract);

figure
subplot(1,2,1)
imshow(img)
title('Original Image')
subplot(1,2,2)
imshow(img_box_sharpen)
title('Box Sharpened Image')

figure
subplot(1,2,1)
imshow(img)
title('Original Image')
subplot(1,2,2)
imshow(img_gauss_sharpen)
title('Gaussian Sharpened Image')

clear all;
close all;
clc;
img = imread('lena.jpg');
scaling_constant = 0.7;

% Increased Kernel Size %
box = imboxfilt(img,7);
gauss = imgaussfilt(img,4);

img_box_subtract = img - box;
img_gauss_subtract = img - gauss;

img_box_sharpen = img + (scaling_constant * img_box_subtract);
img_gauss_sharpen = img + (scaling_constant * img_gauss_subtract);

figure
```

```
subplot(1,2,1)
imshow(img)
title('Original Image')
subplot(1,2,2)
imshow(img_box_sharpen)
title('Box Sharpened Image')

figure
subplot(1,2,1)
imshow(img)
title('Original Image')
subplot(1,2,2)
imshow(img_gauss_sharpen)
title('Gaussian Sharpened Image')
```

Unsharp Sharpening Using 3x3 Box Filter

By increasing kernel size, we get

Unsharp Sharpening Using Gaussian Filter with $\sigma = 4$

Unsharp Sharpening Using 7x7 Box Filter