EECE-5626 (IP&PR) : Homework-5

Due on November 12, 2021 by 11:59 pm via submission portal

NAME: McKean, Tyler

Instructions

1. You are required to complete this assignment using Live Editor.
2. Enter your MATLAB script in the spaces provided. If it contains a plot, the plot will be displayed after the script.
3. All your plots must be properly labeled and should have appropriate titles to get full credit.
4. Use the equation editor to typeset mathematical material such as variables, equations, etc.
5. After completeing this assignment, export this Live script to PDF and submit the PDF file through the provided submission portal.
6. You will have two attempts to submit your assignment. However, make every effort to submit the correct and completed PDF file the first time. If you use the second attempt, then that submission will be graded.
7. Your submission of problem solutions must be in the given order, i.e., P1, P2, P3, etc. Do not submit in a random order.
8. Please submit your homework before the due date/time. A late submission after midnight of the due date will result in loss of points at a rate of 10% per hour until 8 am the following day, at which time the solutions will be published.

Reading Assignment:

- Chapters 8 (except 8.12) and 9 from DIP4E.
- Chapters 9 (except 9.6) and 10 from DIPUM3E

Table of Contents

Problem-5.1: Data and Image Quantization

(a) DIP4E Problem 8.3 (Page 688)

Note: Quantize the given data using truncation and not rounding.

Solution:

For this problem we are using a 1D array of 8-pixel values that are {108,139,135,244,172,173,56,99} and will be uniformly quantized with 4-bit accuracy.

```
f = [108,139,135,244,172,173,56,99];
```

The error $e(x, y)$ can be calculated between the original pixel values $f(x, y)$ and the approximated 4-bit pixel values $\hat{f}(x, y)$ through the equation

$$e(x, y) = \hat{f}(x, y) - f(x, y)$$

Since the new pixel values will be 4-bit, we divide the array by 2^4 or 16 to represent the new level of intensity values and then truncate the results. The resulting truncated value will then be multiplied by 16 to represent the new approximated intensity level value.

```
f_hat = zeros(1,8);
for i = 1:length(f)
    f_hat(i) = floor(f(i)/16) * 16;
end
f_hat
```

```
f_hat = 1×8
    96   128   128   240   160   160    48    96
```

```
e = double(f_hat) - double(f);
```

we then use these new values for the approximated pixel values $f(\hat{x}, y)$ into the root-mean-squared error equation, e_{rms}

$$e_{rms} = \sqrt{\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y) - f(x,y)\right]^2}$$

```
[M,N] = size(e);
e_rms = sqrt(sum(e(:).^2)/(M*N))
```

```
e_rms = 9.4604
```

the mean-squared signal-to-noise ratio, SNR_{ms} is then computed by

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y) - f(x,y)\right]^2}$$

```
num = sum(f_hat(:).^2);
denom = sum((f_hat(:) - f(:)).^2);
SNRms = num/denom
```

```
SNRms = 226.6816
```

(b) Image Quantization

Repeat part (a) using the **`boy.tif`** image which is an 8-bit image. Display the original and quantized images side by side.

Solution:

```
clc; close all; clear;

f = imread('boy.tif');
[M,N] = size(f);
topMask = uint8(240);
f_hat = bitand(f, topMask);
e = double(f_hat) - double(f);
[M,N] = size(e);
e_rms = sqrt(average(e(:).^2))
```

```
e_rms = 8.8312
```

```
num = sum(f_hat(:).^2);
denom = sum((double(f_hat(:)) - double(f(:))).^2);
SNRms = num/denom
```

```
SNRms = 3.0984
```

```
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(f); title('8-bit Boy Image');
```

```
subplot(1,2,2); imshow(f_hat); title('Quantized Boy Image');
```

Problem-5.2: Huffman Coding

This problem has two independent parts.

(a) Huffman Coding on Image Fragment

Consider the 4×8 size 8-bit image from Slide-13 of Topic-10 Powerpoint, which is given below.

$$\begin{matrix} 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \end{matrix}$$

(i) Compression

Compress the image using Huffman coding, that is, provide an encoded bit pattern (ignore that we may need end-of-line markers after every row). Provide either a row-wise bit arrangement or a column-wise bit arrangement.

Solution:

Taking the probabilites of the 4 different intensity values results in an arbitrary symbol table of these intensity values with their respective probabilites

$$a_{21} \rightarrow \frac{3}{8},\ a_{243} \rightarrow \frac{3}{8},\ a_{95} \rightarrow \frac{1}{8}\ ,\text{and}\ a_{169} \rightarrow \frac{1}{8}$$

After following the steps of the Huffman coding procedure the following symbols with have the Huffman codes:

$$a_{21} \rightarrow 1,\ a_{243} \rightarrow 00,\ a_{95} \rightarrow 010\ ,\text{and}\ a_{169} \rightarrow 011$$

Then, using these codes to encode the intensity images in a row-wise arrangement, we would get:

111010011000000

(ii) Unique Codes

How many unique Huffman codes are possible for this four-symbol source?

Solution:

Referencing the Huffman codes I used above, the number of possible unique Huffman codes for these 4 intensity values would be 4 codes = {1, 00, 010, 011}

(iii) Compression Ratio

Compute the compression ratio achieved and discuss the effectiveness of the Huffman coding on this image.

Solution:

To determine the compression ratio, we need to solve for the average number of bits required to represent each pixel in the Huffman coded image, so

$$L_{\text{avg}} = \left(\frac{3}{8}\right)(1) + \left(\frac{3}{8}\right)(2) + \left(\frac{1}{8}\right)(3) + \left(\frac{1}{8}\right)(3) = 1.875 \text{ bits/pixel}$$

the compression ratio would then be

$$C = \frac{4 \times 8 \times 8}{4 \times 8 \times 1.875} = \frac{256}{60} = 4.2667 \text{ or } 4.27$$

To display the effectiveness of the Huffman coding, we first calculate the entropy of the original image

$$\widetilde{H} = -\sum_{k=0}^{255} p_r(r_k) log_2 p_r(r_k)$$

$$= \frac{3}{8} log_2 \frac{3}{8} + \frac{1}{8} log_2 \frac{1}{8} + \frac{1}{8} log_2 \frac{1}{8} + \frac{3}{8} log_2 \frac{3}{8} = -[-0.5306 - 0.375 - 0.375 - 0.5306] = 1.811 bits/pixel$$

the theoretical compression would then be

$C = \frac{4 \times 8 \times 8}{4 \times 8 \times 1.811} = \frac{8}{1.811} = 4.4417$ then comparing the theoretical compression to the compression achieved by the Huffman coding

we see that $\frac{4.27}{4.417} = 0.9667$ or about 97% of the maximum comporession is possible through removing redundancies using Huffman coding.

(b) Huffman Decoding

Using the Huffman code in Fig. 8.8 (in DIP4E), decode the encoded string

`0100110000010101010111`.

Show all of your work to get full credit.

Solution:

Referencing the Fig 8.8 in DIP4E, the following symbols have their assigned Huffman codes

$$a_2 \rightarrow 1,\ a_6 \rightarrow 00,\ a_1 \rightarrow 011,\ a_4 \rightarrow 0100,\ a_3 \rightarrow 01010,\ \text{and}\ a_5 \rightarrow 01011$$

Then examining the provided Huffman code, the decode sequence of symbols would be:

$$a_4a_2a_2a_6a_6a_3a_2a_5a_2 \rightarrow 0100\,|\,1\,|\,1\,|\,00\,|\,00\,|\,01010\,|\,1\,|\,01011\,|\,1\,|$$

Problem-5.3: DIP4E MATLAB Project 8.3 (Page 691)

(a) MATLAB function `compare4e`

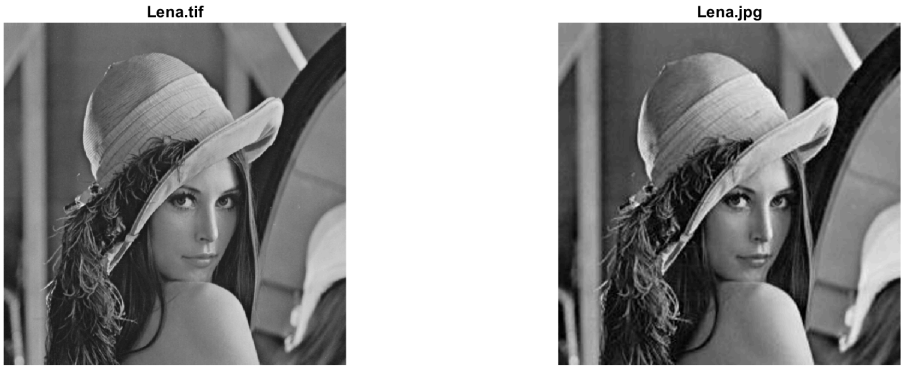
Solution: Insert the code of your function below after the comments.

```
function rmse = compare4e(f1,f2)
%function rmse = compare4e(f1,f2)
%   RMSE = COMPARE4E(F, FA) returns the root-mean-square error
%   between inputs F and FA.

e = double(f1) - double(f2);
[M,N] = size(e);
rmse = sqrt(sum(e(:).^2)/(M*N));
end
```

(b) RMSE between Uncoded and Coded Images

Solution:

```
clc; close all; clear;
f1 = imread('lena.tif');
f2 = imread('lena.jpg');
f2 = imresize(f2,2);
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(f1); title('Lena.tif');
subplot(1,2,2); imshow(f2); title('Lena.jpg');
```




```
rmse = compare4e(f1,f2)
```

```
rmse = 13.0281
```

Problem-5.4: DIP4E MATLAB Project 8.7 (Page 691) - Modified

JPEG Compression

The function **`c = im2jpeg4e(f,quality,bits)`**, to approximate the baseline JPEG image encoding process, is available as a solution to part (a) in Student MATLAB Project Solutions.

(a) Compression of `lena.tif` Image

Using this function with the quality factor equal to 2, compress the image **`lena.tif`**. From the resulting encoded structure, determine the number of bits in the encoded image.

Solution:

```
clc; close all; clear;
f = imread('lena.tif');
f_jpeg = im2jpeg4e(f,2);
fhuffman = f_jpeg.huffman;
whos('fhuffman')
```

```
  Name          Size            Bytes  Class     Attributes

  fhuffman      1x1             14774  struct
```

```
fcode = f_jpeg.huffman.code;
whos('fcode')
```

```
  Name       Size              Bytes  Class     Attributes

  fcode      1x6991            13982  uint16
```

```
dec2bin(double(fcode));
[r,numBytes] = size(fcode);
numBits = numBytes * 16
```

```
numBits = 111856
```

So the number of bits in the encoded image is 111,856 bits

(b) Computation of Compression Ratio

Compute the compression ratio using the **`compressionRatio4e`** function, which is also available as a solution to Problem 8.2 in Student MATLAB Project Solutions.

Solution:

```
c = compressionRatio4e(f,fcode)
```

```
c = 18.7487
```

(c) Part (c) of Project 8.7

Solution: Insert the code of your **`jpeg2im4e`** function below after the comments.

```
function x = jpeg2im4e(y)
% FUNCTION JPEG2IM4E decodes the compressed image Y, and generates an approximate reconstruction of image X. Y

narginchk(1,1); % check input argument is a struct
```

```
m =  [16  11  10  16  24  40  51  61          % JPEG normalizing array and zig-zag reordering pattern
      12  12  14  19  26  58  60  55
      14  13  16  24  40  57  69  56
      14  17  22  29  51  87  80  62
      18  22  37  56  68  109 103 77
      24  35  55  64  81  104 113 92
      49  64  78  87  103 121 120 101
      72  92  95  98  112 100 103 99];
order = [1 9  2  3  10 17 25 18 11  4  5 12 19 26 33 ...
        41 34 27 20 13 6  7  14 21 28 35 42 49 57 50 ...
        43 36 29 22 15 8  16 23 30 37 44 51 58 59 52 ...
        45 38 31 24 32 39 46 53 60 61 54 47 40 48 55 ...
        62 63 56 64];
rev = order;  % Compute inverse ordering
for k = 1:length(order)
    rev(k) = find(order == k);
end
m = double(y.quality)/100*m % encoding quality
xb = double(y.numblocks); % x blocks
sz = double(y.size);
xm = sz(1); % x rows
xn = sz(2); % x cols
x = huff2mat(y.huffman); % Huffman decode
eob = max(x(:)); % EOB symbol

z = zeros(64,xb);
k = 1;
for j = 1:xb
  for i = 1:64
    if x(k) == eob
        k = k + 1;
        break;
    else
        z(i,j) = x(k);
        k = k + 1;
       end
    end
 end

z = z(rev,:); % restore order
x = col2im(z,[8 8], [xm xn],'distinct'); % Form matrix blocks

fun = @(block_struct)denorm(block_struct.data,m);
x = blockproc(x,[8 8], fun); % denormalize DCT
t = dctmtx(8); % 8 x 8 DCT matrix
fun = @(block_struct)iblkdct(block_struct.data,t);
x = blockproc(x,[8 8],fun); % compute block DCT-1
x = x + double(2^(y.bits - 1)); % level shift
if y.bits <= 8
   x = uint8(x);
else
   x = uint16(x);
end
end

% Inverse DCT matrix multiplications
function y = iblkdct(x,a)
```

```
y = a'*x*a;
end

%Denormalize DCT
function y = denorm(x,m)
y = x.*m
end
```

Decoding of Compressed Image

```
f_decode = jpeg2im4e(f_jpeg);
figure, imshow(f_decode), title('Decoded JPEG Image'); pause(1)
```

(d) Part (d) of Project 8.7

Use function **`compare4e`** from Project 8.3 to compute the root-mean-squared error between image **`lena.tif`** and the decompressed image from part (c).

Solution:

```
rms = compare4e(f,f_decode)
```

```
rms = 4.9346
```

Problem-5.5: DIPUM3E MATLAB Project 9.1 (Page 580)

Relative Data Redundancy (RDR)

(c) Function `redundancy`

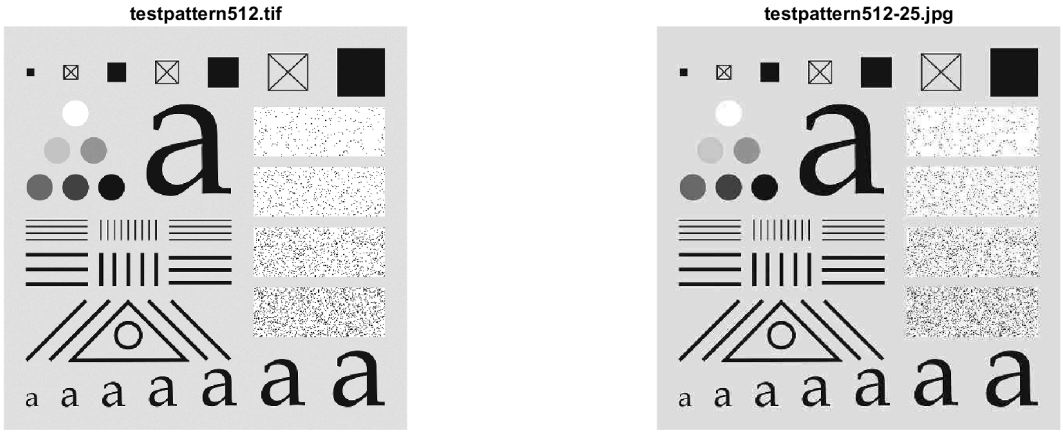
Solution: Insert the code of your function below after the comments.

```
function [r] = redundancy(I1,I2)
%REDUNDANcY returns the relative data redundancy of I1 with respect to I2.
% Since R = (i1–i2)/i1, we can write R = (i1/i1) – (i2/i1) = 1 – i2/i1.
% Denoting cr = il/i2, we have R = 1 – 1/cr. Note that in these
% equations, i1 and i2 represent the number of bytes in input images I1
% and I2, respectively.

cr = imratio(I1,I2); % Compute compression ratio of bytes between i1 and i2
r = 1 - 1/cr;
end
```

(d) Computation of RDR

Solution: MATLAB script.

```
clc; close all; clear;
f = imread('testpattern512.tif');
imwrite(f,'testpattern512-25.jpg','quality',25);
f1 = imread('testpattern512-25.jpg');
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(f); title('testpattern512.tif');
subplot(1,2,2); imshow(f1); title('testpattern512-25.jpg');
```




```
r = redundancy(imread('testpattern512.tif'),'testpattern512-25.jpg')
```

```
r = 0.9015
```

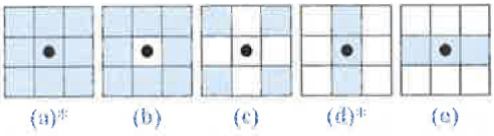
The redundancy of the testpattern512.tif file with respect to the testpattern512-25.jpg file is about 90%.

Problem-5.6: Erosion and Dilation

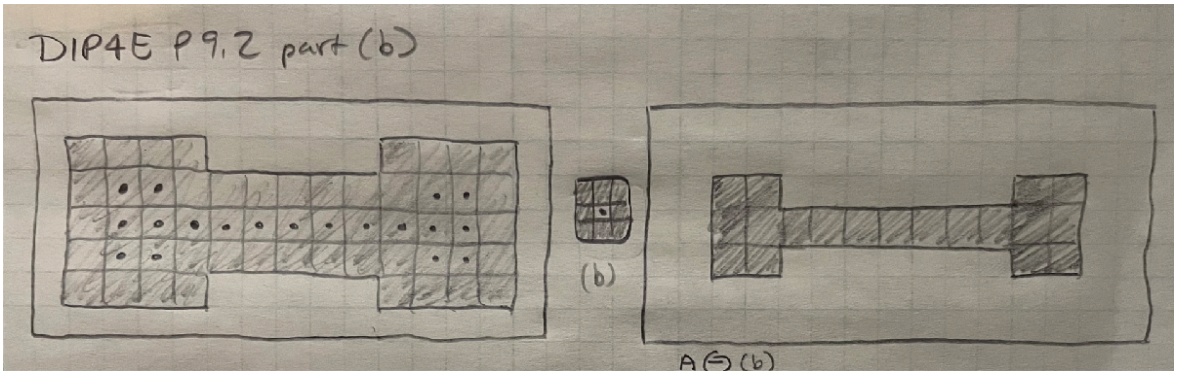
In the following parts, perform the stated operations by hand first and then verify using MATLAB.

(a) DIP4E Problem 9.2 (b) only (Page-750).

Sketch the results of eroding Fig. 9.3(a) with the structuring element (b) below. Assume a boundary of zeros.



Hand Calculated Solution:



MATLAB script:

```
clc; close all; clear;
A = zeros(7,15);
A(2,2:5) = 1; A(2,11:14) = 1;
A(3:5,2:14) = 1;
A(6,2:5) = 1; A(6,11:14) = 1;
A
```

```
A = 7×15
     0     0     0     0     0     0     0     0     0     0     0     0     0 ···
     0     1     1     1     1     0     0     0     0     0     1     1     1
     0     1     1     1     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     0     0     0     0     0     1     1     1
     0     0     0     0     0     0     0     0     0     0     0     0     0
```

```
se = strel('rectangle',[3,3]); arr = se.Neighborhood
```

```
arr = 3×3 logical array
   1   1   1
   1   1   1
   1   1   1
```

```
arr(2,2) = 0
```

```
arr = 3×3 logical array
   1   1   1
   1   0   1
   1   1   1
```

```
imerode(A,arr)
```

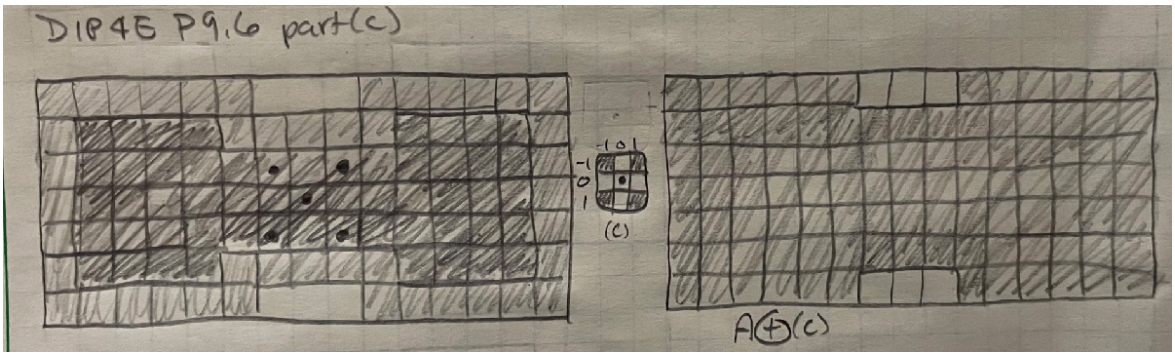
```
ans = 7×15
     0     0     0     0     0     0     0     0     0     0     0     0     0 ···
     0     0     0     0     0     0     0     0     0     0     0     0     0
```

```
     0     0     1     1     0     0     0     0     0     0     0     1     1
     0     0     1     1     1     1     1     1     1     1     1     1     1
     0     0     1     1     0     0     0     0     0     0     0     1     1
     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0
```

(b) DIP4E Problem 9.6 (c) only (Page-750).

Dilate Fig. 9.3(a) using the structuring element in figure (c) of Problem 9.2.

Hand Calculated Solution:



MATLAB script:

```
A
```

```
A = 7×15
     0     0     0     0     0     0     0     0     0     0     0     0     0 ···
     0     1     1     1     1     0     0     0     0     0     1     1     1
     0     1     1     1     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     0     0     0     0     0     1     1     1
     0     0     0     0     0     0     0     0     0     0     0     0     0
```

```
arr(2,:) = 0; arr(:,2) = 0;
arr
```

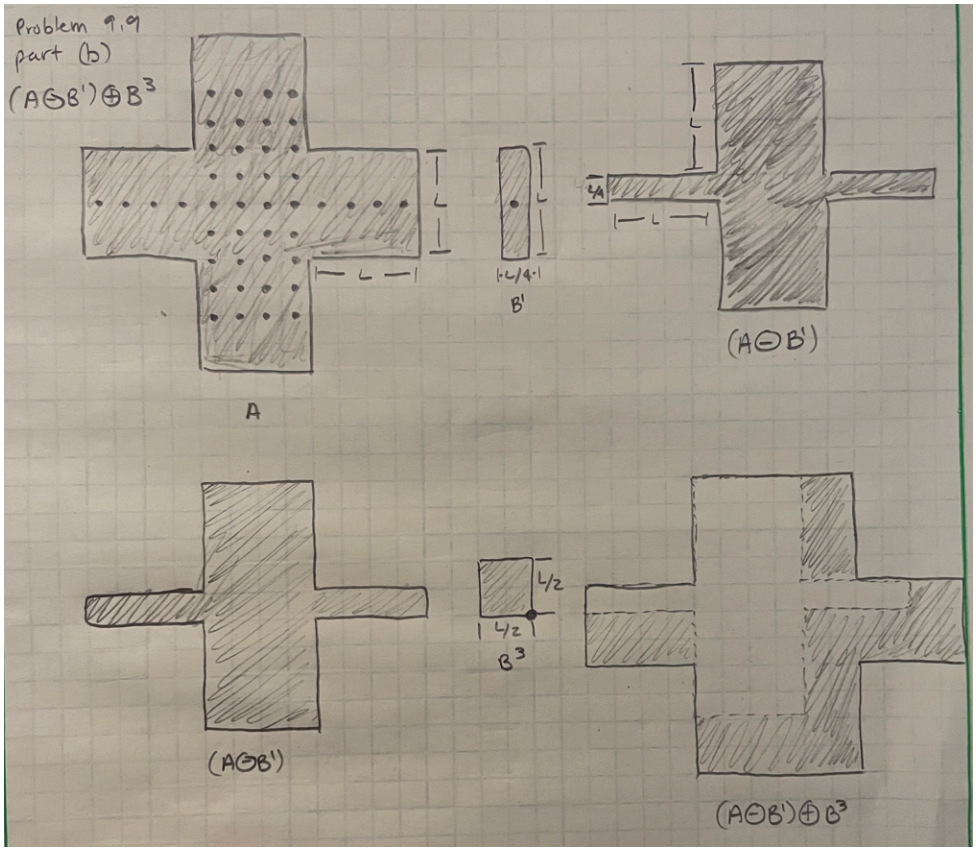
```
arr = 3×3 logical array
   1   0   1
   0   0   0
   1   0   1
```

```
imdilate(A,arr)
```

```
ans = 7×15
     1     1     1     1     1     1     0     0     0     1     1     1     1 ···
     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     0     0     0     1     1     1     1
```

Problem-5.7: DIP4E 9.9 (b) only (Page 751)

Solution:



Problem-5.8 Grayscale Erosion and Dilation

Let A be the matrix

$$A = \begin{bmatrix} 35 & 1 & 6 & 26 & 19 & 24 \\ 3 & 32 & 7 & 21 & 23 & 25 \\ 31 & 9 & 2 & 22 & 27 & 20 \\ 8 & 28 & 33 & 17 & 10 & 15 \\ 30 & 5 & 34 & 12 & 14 & 16 \\ 4 & 36 & 29 & 13 & 18 & 11 \end{bmatrix}$$

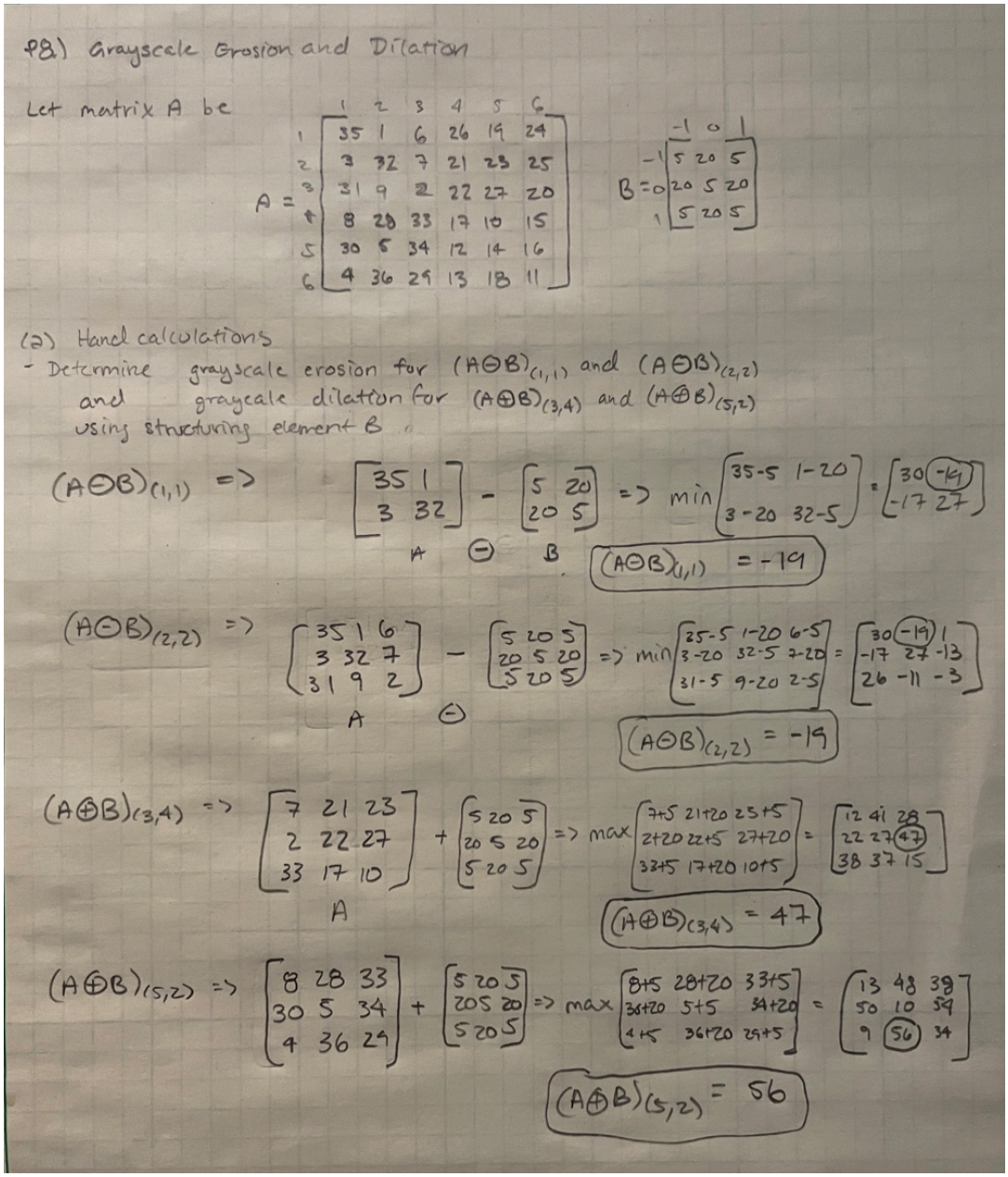
(a) Hand Calculations

Using hand calculations, determine the grayscale erosion for $(A \ominus B)_{(1,1)}$ and $(A \ominus B)_{(2,2)}$ elements and dilation for $(A \oplus B)_{(3,4)}$ and $(A \oplus B)_{(5,2)}$ elements with the structuring element B given as

$$B = \begin{bmatrix} 5 & 20 & 5 \\ 20 & 5 & 20 \\ 5 & 20 & 5 \end{bmatrix}.$$

All details must be shown to get full credit.

Solution:



(b) MATLAB Verification

Determine $A \ominus B$ and $A \oplus B$ using MATLAB and verify your calculations in (a).

```
clc; close all; clear;
B = strel('arbitrary',ones(3),[5,20,5;20,5,20;5,20,5])
```

```
B =
strel is a arbitrary shaped structuring element with properties:

      Neighborhood: [3×3 logical]
    Dimensionality: 2
```

```
A = [35,1,6,26,19,24; 3,32,7,21,23,25; 31,9,2,22,27,20; 8,28,33,17,10,15; 30,5,34,12,14,16; 4,3
```

```
AerodeB = imerode(A,B)
```

```
AerodeB = 6×6
   -19   -14   -19   -14     3    -1
    -4   -19   -18   -13    -1     0
   -17   -18   -13   -18   -10    -5
     0   -15   -18   -10    -6   -10
   -16    -1   -15    -7   -10    -9
    -1   -16    -7    -8    -9    -4
```

```
AdilateB = imdilate(A,B)
```

```
AdilateB = 6×6
    40    55    46    41    46    45
    55    40    52    46    47    44
    37    52    53    47    43    47
    51    53    54    53    47    40
    41    56    53    54    38    35
    56    49    56    49    34    38
```

Problem-5.9 DIP4E MATLAB Project 9.5 (a) & (c) only (Page 757)

Boundary Extraction

(a) morphoBoundary4e Function

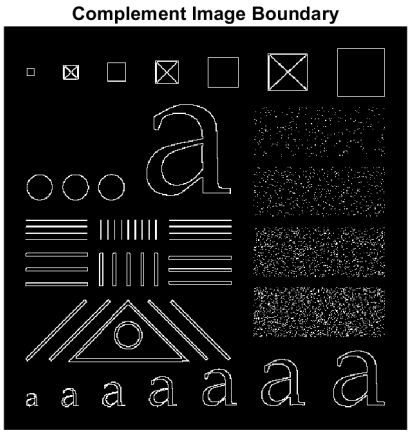
Solution: Insert the code of your function below after the comments.

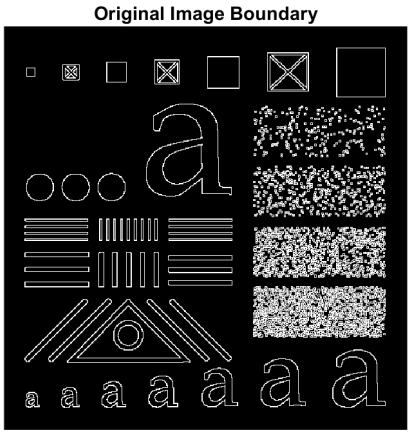
```
function BD = morphoBoundary4e(I,B)
%MORPHOBOUNDARY4E Boundary of objects in a binary image.
% BD = MORPHOBOUNDARY4E(I,B) computes the boundaries of all
% foreground pixels in binary image I using structuring element B.
% If B is not inc1uded in the input argument, it defaults to a
% 3-by—3 array of 1's. If it is, it must be a m—by-n array (m and n
% odd) of 1's.
%

if nargin < 2
    B = strel('square',3);
end
[M,N] = size(B.Neighborhood)
if ~isodd(M)
    error('M is even value and must be odd!')
end
if ~isodd(N)
    error('N is even value and must be odd!')
end

IerodeB = imerode(I,B);
BD = I - IerodeB;

end
```

(c) Processing of testpattern512-binary.tif Image

MATLAB script:

```
clc; close ALL; clear;
B = strel('square',3);
f = imread('testpattern512-binary.tif'); fc = not(f);
BDc = morphoBoundary4e(fc,B); BD = morphoBoundary4e(f,B);
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(BDc); title('Complement Image Boundary');
subplot(1,2,2); imshow(BD); title('Original Image Boundary');
```

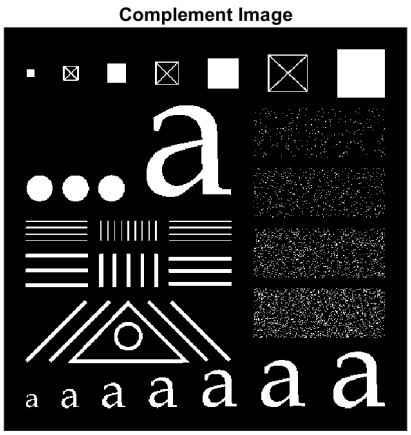


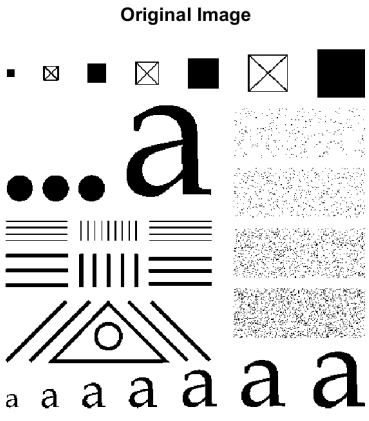


Comment on the difference between the result of (b) and the result in this part.

Answer:

```
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(fc); title('Complement Image');
subplot(1,2,2); imshow(f); title('Original Image');
```





The differences from the boundary extraction of the original testpattern image and it's complement are explained by which pixels are in the foreground and which were in the background.

The original binary image contains mostly foreground white pixels (1s), with the shapes, lettering, and noise patterns are made up of background pixel values (0s). The opposite is true of the complement binary image,

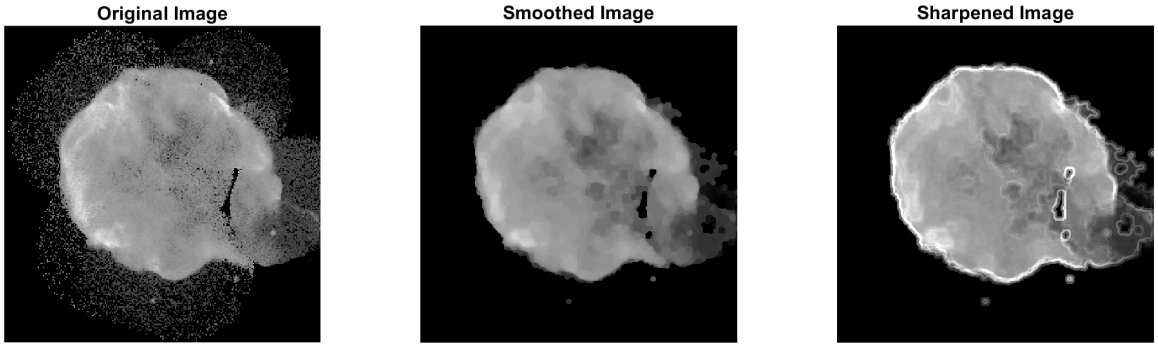
whereas now, the forground pixels are lettering, shapes, and noise patterns and the rest of the image are composed of background pixel values. When performing the erosion process with the structuring element, B, followed by the subtraction of this erosion result from the input image, the complementary boundary had less foreground pixels in the noise patterns then did the original boundary extraction image. This is because the original image had its noise pattern as background pixels so when taking the boundary the white pixel unaffected by noise were extracted from the boundary process. The complementary image has less boundaries in this region, because its forground images became the complement to these noise pattern pixels. The original boundary image also has more shapes that appear hollow, with a boundary line from the inside and outside of the triangle and line shapes. The complement boundaries do not have as much of a gap of background pixels, like the original boundary image does.

Problem-5.10 DIPUM3E Project 10.8 (Page 632)

(b) Morphological Smoothing/Sharpening of `cygnusloop.tif` Image

MATLAB script:

```
clc; close all; clear;
f = imread('cygnusloop.tif');
w = fspecial('laplacian',0);
w_se = strel('arbitrary',ones(3),w);
se = strel('disk',5);
fo = imopen(f,se);
foc = imclose(fo,se);
fsharp = foc + (imdilate(foc,se) - imerode(foc,se));

figure('Units','inches','Position',[0,0,12,4]);
subplot(1,3,1); imshow(f); title('Original Image');
subplot(1,3,2); imshow(foc); title('Smoothed Image');
subplot(1,3,3); imshow(fsharp); title('Sharpened Image');
```



```
function rmse = compare4e(f1,f2)
%function rmse = compare4e(f1,f2)
%   RMSE = COMPARE4E(F, FA) returns the root-mean-square error
%   between inputs F and FA.

e = double(f1) - double(f2);
[M,N] = size(e);
rmse = sqrt(sum(e(:).^2)/(M*N));
end

function x = jpeg2im4e(y)
% FUNCTION JPEG2IM4E decodes the compressed image Y, and generates an approximate reconstructio

narginchk(1,1); % check input argument is a struct

m =  [16  11  10  16  24  40  51  61            % JPEG normalizing array and zig-zag reordering
      12  12  14  19  26  58  60  55
      14  13  16  24  40  57  69  56
      14  17  22  29  51  87  80  62
      18  22  37  56  68  109 103 77
      24  35  55  64  81  104 113 92
      49  64  78  87  103 121 120 101
      72  92  95  98  112 100 103 99];
order = [1 9  2  3  10 17 25 18 11  4  5 12 19 26 33 ...
        41 34 27 20 13 6  7  14 21 28 35 42 49 57 50 ...
        43 36 29 22 15 8  16 23 30 37 44 51 58 59 52 ...
        45 38 31 24 32 39 46 53 60 61 54 47 40 48 55 ...
        62 63 56 64];
rev = order;  % Compute inverse ordering
for k = 1:length(order)
    rev(k) = find(order == k);
end
m = double(y.quality)/100*m; % encoding quality
xb = double(y.numblocks); % x blocks
sz = double(y.size);
xm = sz(1); % x rows
xn = sz(2); % x cols
x = huff2mat(y.huffman); % Huffman decode
eob = max(x(:)); % EOB symbol

z = zeros(64,xb);
k = 1;
for j = 1:xb
    for i = 1:64
        if x(k) == eob
        k = k + 1;
        break;
    else
        z(i,j) = x(k);
        k = k + 1;
       end
    end
end
```

```
z = z(rev,:); % restore order
x = col2im(z,[8 8], [xm xn],'distinct'); % Form matrix blocks

fun = @(block_struct)denorm(block_struct.data,m);
x = blockproc(x,[8 8], fun); % denormalize DCT
t = dctmtx(8); % 8 x 8 DCT matrix
fun = @(block_struct)iblkdct(block_struct.data,t);
x = blockproc(x,[8 8],fun); % compute block DCT-1
x = x + double(2^(y.bits - 1)); % level shift
if y.bits <= 8
   x = uint8(x);
else
   x = uint16(x);
end
end
% Inverse DCT matrix multiplications
function y = iblkdct(x,a)
y = a'*x*a;
end

%Denormalize DCT
function y = denorm(x,m)
y = x.*m;
end

function [r] = redundancy(I1,I2)
%REDUNDANcY returns the relative data redundancy of I1 with respect to I2.
% Since R = (i1–i2)/i1, we can write R = (i1/i1) – (i2/i1) = 1 – i2/i1.
% Denoting cr = il/i2, we have R = 1 – 1/cr. Note that in these
% equations, i1 and i2 represent the number of bytes in input images I1
% and I2, respectively.

cr = imratio(I1,I2); % Compute compression ratio of bytes between i1 and i2
r = 1 - 1/cr;
end

function BD = morphoBoundary4e(I,B)
%MORPHOBOUNDARY4E Boundary of objects in a binary image.
% BD = MORPHOBOUNDARY4E(I,B) computes the boundaries of all
% foreground pixels in binary image I using structuring element B.
% If B is not inc1uded in the input argument, it defaults to a
% 3-by–3 array of 1's. If it is, it must be a m–by-n array (m and n
% odd) of 1's.
%

if nargin < 2
    B = strel('square',3);
end
[M,N] = size(B.Neighborhood);
if ~isodd(M)
    error('M is even value and must be odd!')
end
if ~isodd(N)
```

```
    error('N is even value and must be odd!')
end

IerodeB = imerode(I,B);
BD = I - IerodeB;

end
```