ECE568 – Digital Image Processing, Winter 2019
Assignment 1

Due: 8:30 am Thursday 1/24/2019, before the lecture begins.

<u>Note: Please follow the homework submission guidelines on the class webpage.</u>

1) Image processing and commercial tools **(15 points)**

   a. Capture a digital color image of yourself and enlarge it by a factor of 2.5 in both horizontal and vertical dimensions using an image editing tool. Put the original and enlarged images in your report.
   b. Adjust **1_1.jpg** (e.g. brightness, contrast …) until you find it most pleasant. Put the adjusted image in your report.

   HINT: You could use any software you have (e.g. Adobe Photoshop, Paint/Photos in Windows).

2) Image I/O and data types. **(15 points)**
   a. Load the Lena image **1_4.bmp**, using MATALB: imread() or Python: cv2.imread(), and show it using MATLAB: imshow() or Python: cv2.imshow().
   b. Get the type of the loaded image data (Use MATLAB function class(), or Python/Numpy array_name.dtype), and get the maximum and the minimum data values for this image (Use MATLAB function max() and min(); or Numpy np.amax() and np.amin()).
   c. Convert the data to the "double" type (use MATLAB function double() or Numpy astype(float)), can you show the double-typed image using MATLAB: imshow() or Python: cv2.imshow()?
   d. If not, given an image which has been converted to the "double" type, how do you show the image?

   HINT: MATLAB has an image value range for the default uint8 type in [0 255]. For imshow(), if the data type is double, it should be in the range [0 1]. Double type data can be converted to uint8 data, or data can be normalized to be in [0 1] for imshow().
   To use Python, make sure numpy and opencv are installed, then import those two modules:

```
import numpy as np

import cv2
```

   To display image using Python opencv, use:

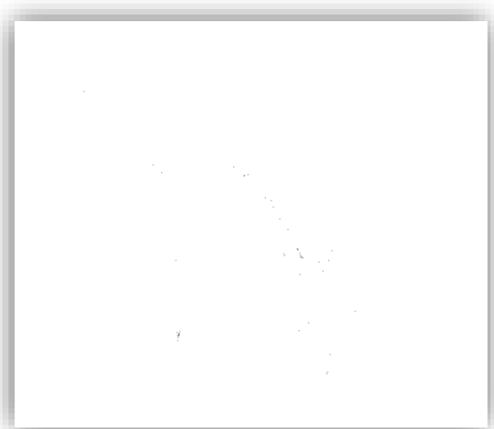
```
cv2.imshow('a_string', im)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

   Sol:

MATLAB Code:

```
clear all;  % clear all variables in workspace
close all;  % close all figure windows
clc;        % clear the command window

% Read lena
f = imread('1_4.bmp');

% show it
figure, imshow(f);

% get the type
class(f)

% get minimum and maximum value
min(f(:))
max(f(:))

% convert to double
g = double(f);

% show it
figure, imshow(g);

% that is not correct -
% correction 1.
figure, imshow(g,[]);
% correction 2. Convert back to uint8 type
g2 = uint8(g); figure, imshow(g2);
% correction 3. Convert to [0 1]
g3 = g / 255; figure, imshow(g3);
```

When you type class(f), it will show: uint8, which means the image data is stored in the memory as unsigned-int-8 type. Double() will convert the data into double-precision-floating type. When you show the double-precision-floating type, this is what you will get:



You only see several black dots on the almost white picture! This is because when Matlab is displaying or writing a double-precision-floating image, it will treat value 0 as black, value 1 as white, and any other value out of this range clipped.

Keep this in mind when you work with double-type data!

3) Matlab/Python basics: Matlab/Python commands. **(20 points)**

Write a script to do the following.

a. Read **1_2.tif** and its associated colormap into variables named “X” and “map”. Use “X” and “map” to convert the image to a 256-level grayscale image “Y”.
b. Rotate “Y” 120 degrees clockwise to generate image “Z0”.
c. Rotate “Y” 10 degrees 12 times to generate image “Z1”.
d. Can you observe any differences between image “Z0” and “Z1”?
e. Submit images Y, Z0 and Z1, and the script you wrote.

HINT:
Use the Matlab commands: [X,map]=imread(‘1_2.tif', 'tif') , imshow(X, map) , ind2gray, imrotate.
For Python/Opencv, X=cv2.imread(‘1_2.tif’), to rotate an image, use the function below:

```
def imRotate(X, degrees):
  '''
  Args:
       X: numpy array of shape [height, width, channels]
       degrees: rotation angle, positive means counter clockwise rotation
  Output: rotated image
  '''
  height, width, _ = X.shape
  M = cv2.getRotationMatrix2D((height/2.0,width/2.0), degrees, 1)
  return cv2.warpAffine(X, M, (height, width))
```

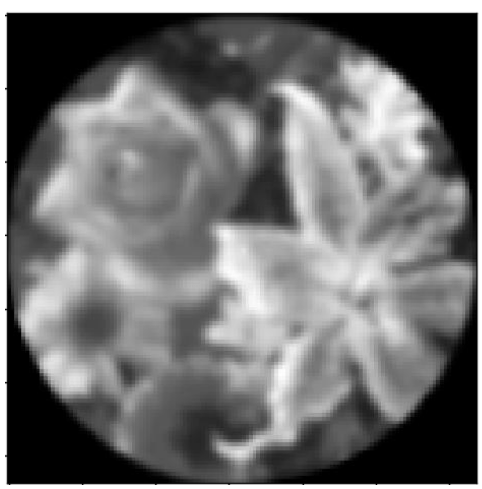
<u>Note: Write your own codes for the following problems.</u>

Sol:

a)

b)

c)



Note: you may get different image if you use different rotation functions

d)

image rotated by 12 times is blurry

 # Python Code

```
import numpy as np

import matplotlib.pyplot as plt

import cv2

# uncomment below if you use jupyter notebook

# %matplotlib notebook


X = cv2.cvtColor(cv2.imread('1_2.tif'), cv2.COLOR_BGR2GRAY)

plt.figure()

plt.imshow(X, cmap='gray')


def imRotate(X, degrees):

   '''

   Args: X: numpy array of shape [height, width, channels]

        degrees: rotation angle, positive means counter clockwise rotation

   '''

   height, width= X.shape

   M = cv2.getRotationMatrix2D((height/2.0,width/2.0), degrees, 1)

   return cv2.warpAffine(X, M, (height, width))

X_rotated120 = imRotate(X, 120)

plt.figure()

plt.imshow(X_rotated120, cmap='gray')


X_rotated_12_times = X.copy()

for i in range(12):

   X_rotated_12_times = imRotate(X_rotated_12_times, 10)

plt.figure()

plt.imshow(X_rotated_12_times, cmap='gray')
```

4) Image Resolution. **(50 points)**

a. Reduce the resolution of **1_3.asc** by a factor of 4 in both horizontal and vertical dimensions (e.g., if the original image is 400 by 400, then result shall be 100 by 100) to create a decimated image using two different methods:

   HINT: To read in an ".asc", use MATLAB: X=load('**1_3.asc**') or Python: X=np.loadtxt('1_3.asc'). For the double type, 'imshow' only works for images with values between 0 and 1. To display the .asc image, you may scale the pixel value from [0, 255] to [0, 1] range.

   i. Keep one pixel out of every 4x4 pixel area. Display the resulting image Y1. ***HINT:*** This can be done with only one line of code. You do not need to use for loops to accomplish this. Consider what the command MATLAB: A=B(1:3:20, 1:3:30) or Python: A=B[0:20:3,0:20:3] does to an image B.
   ii. Replace every 4x4 pixel area in **1_3.asc** by the average value of the pixel values in that region. Display the resulting image Y2.

b. Enlarge Image Y1 by a factor of 4 in both horizontal and vertical dimensions (e.g., from 100 by 100 to 400 by 400) using:

   i. Pixel repeating. (Since each pixel is blown up to a 4x4 block, the image looks "blocky".)
   ii. Bilinear interpolation (do not use built-in interpolation function, *use your own code*).
   iii. Keep the resulting images from (b.i) and (b.ii) the same size as **1_3.asc** and compare the images.

[Sol]
a)

Y1

Y2

MATLAB CODE

```
%a)
clear all;
% Read in image 3
X=load('1_3.asc');

% Show the original image 3
figure;
imshow(X/256);

% Keep one pixel out of every four pixels and store image as Y1
Y1=X(1:4:end,1:4:end);

% Display the image Y1
figure;
imshow(Y1/256);

% Make another empty image Y2 with the same dimensions as Y1
Y2=Y1*0;

% Now use the average value for every 4x4 region of the original
image
for I=1:size(X,1)/4
    for J=1:size(X,2)/4
        Y2(I,J)=mean(mean(X((I-1)*4+1:I*4,(J-1)*4+1:J*4)));
    end
end

% Display the image Y2
figure;
imshow(Y2/256);
```

b)

Pixel Repeating

Bilinear Interpolation

MATLAB CODE

```
% b)
% Enlarge image Y1 using two separate methods

clear all;
% Read in image 3
X=load('1_3.asc');

% Keep one pixel out of every four pixels and store image as Y1
Y1=X(1:4:end,1:4:end);

% Make two empty images that are 4 times as large as Y1
Y3=zeros(size(Y1)*4);
Y4=Y3;

% Enlarge image Y1 using pixel repeating
for I=0:size(Y1,1)-1
for J=0:size(Y1,2)-1
Y3(I*4+1:I*4+4,J*4+1:J*4+4)=Y1(I+1,J+1);
end
end

% Display enlarged image with pixel repeating
figure;
imshow(Y3/256);

% Enlarge image Y1 using bilinear interpolation
% first do column linear interpolation
[r, c] = size(Y1);
for I=1:r
for J=1:(c-1)

% Calculate the column interpolations
P1=Y1(I,J);
P5=Y1(I,J+1);
P2=(3/4)*P1 + (1/4)*P5;
P3=(1/2)*P1 + (1/2)*P5;
P4=(1/4)*P1 + (3/4)*P5;

% Place the results in an intermediary matrix
YTEMP(I,(4*J-3):(4*J)) = [P1 P2 P3 P4];
end
%Fill in the last column
YTEMP(I,(4*c-3):4*c) = Y1(I,c);
end
```

```
% Second do row linear interpolation on the intermediary matrix
[m, n] = size(YTEMP);
for J=1:n
    for I=1:(m-1)

        % Calculate the row interpolations
        P1=YTEMP(I,J);
        P5=YTEMP(I+1,J);
        P2=(3/4)*P1 + (1/4)*P5;
        P3=(1/2)*P1 + (1/2)*P5;
        P4=(1/4)*P1 + (3/4)*P5;

        % Place the results in the final bilinear interpolated
image
        Y4((4*I-3):4*I, J) = [P1 P2 P3 P4]';
    end
    %Fill in the last row
    Y4((4*m-3):4*m, J) = YTEMP(m,J);
end

% Display the enlarged image with bilinear interpolation
figure;
imshow(Y4/256);
```