DS601: Digital Image Processing - Assignment
Teacher: Dr. Nitin Khanna (nitin@iitbhilai.ac.in)

Teacher: Dr. Nitin Khanna (nitin@iitbhilai.ac.in)

Guidelines:

- There should be only one .zip file (containing all your .m or .py files, figures/images corresponding to results and a .pdf and corresponding .tex/.doc file for the summary report). Filename shall be in the format as follows: DIP-A-groupno.zip (For e.g. DIP-A-G1.zip). Only one member of every group needs to submit this zip file on Google classroom.
- There shall be one report as .tex or .doc(x) file (with corresponding .pdf file) showing the input image, output image, discussion about results etc. corresponding to different questions. Also include the description of parameters chosen and any specific observations made. Filename for the report shall again be in format as follows: DIP-A-report-groupno.tex and DIP-A-report-groupno.pdf ex. DIP-A-report-G1.tex and DIP-A report-G1.pdf.
- Include a readme.txt file mentioning different steps for running your program and any other comments from your side.
- Make suitable assumptions as needed and clearly mention them at appropriate places in your code, readme file and report.
- In writing code for the following problems, you can use Matlab/Python's in-built function (from any existing toolbox/package/library, etc.) for image reading and writing (such as imread and imwrite or their equivalents), apart from these, please do not use any other function from Matlab/Python's toolboxes related to image processing (except the functions mentioned as allowed function under a particular question). Write the code from scratch, using basic constructs like loops etc. Please confirm with me by posting a query in Google classroom if you think any other in-built function is unavoidable to use. *Note: Allowed in-built Matlab/Python functions/constructs (these or their equivalents): nargin, rand, find, imshow, imread, imwrite.*
- Matlab/Python code should be very well documented. The evaluation will also give consideration to your writ ing efficient code/algorithms, using dynamic and not-static declarations etc., code having proper indentation and self-explanatory comments etc.
- Unless you state otherwise, it is implicitly assumed that you wrote the code yourself without any help from your friends/online-resources. If you take some help, you must mention that explicitly in your report as well as in your Matlab/Python code comments. Plagiarism related polices as per Institute rules will be applicable and strictly enforced.

Note: Images given in the questions are just shown for reference purpose, they are not upto scale. Feel free to let me know if you need any further clarification.

1. (Digital Paths)

   Write a generic Matlab/Python function

   $$[paths\_info] = find\_paths(I, x1, y1, x2, y2, V, path\_type), \quad (1)$$

   which will find all 4, 8- and m-paths between two points (x1,y1) and (x2,y2) in an image.

   Inputs:
   I: An image as 2D matrix,
   x1,y1,x2,y2: coordinates of the two points

   V: set V as an array,
   path_type: type of path (= 4, 8 or 10 (10 for m path))

Output: array of structures containing all the paths of desired type, their lengths and the shortest path (paths will be stored as cell array)

Test your code on the image segment shown in Figure 1 (write a separate test script to do so), for V = *{4,2}* and find all the 4, and m-paths between p and q.

1	0	3	2	4
4	3	4	0	2 (q)
2	2	1	3	0
2 (p)	4	0	2	3
3	2	4	1	0

Figure 1: Sample Image

2. Write a Matlab/Python function [*cc*] = *Conncomp*(*I, connectivity, V*) to find all the connected components in an Image using Sequential algorithm. The parameter *connectivity* may be either 4 or 8, while the parameter *V* contains the set of intensity values to define connectivity. Compare your results with Matlab/Python's inbuilt function bwconncomp.

3. (Digital Image Creation - Circles) Write a generic Matlab/Python function

$$[I] = create\ discs(M, N, border, n, r_1, r_2, V_f, V_b), \quad (2)$$

which will create a 8-bit grayscale image similar to Figure 2. This image is of size $M \times N$, additionally it has a black border around it of thickness *border*. Thus, the size of complete image is (M+2×*border*)×(N +2×*border*). This image contains n non-overlapping discs of radius uniformly distributed in [r_1, r_2]. Centers of each of these discs are obtained from a uniform distribution over all possible pixel locations in the image. The selected centers should be such that the discs are non-overlapping. For randomly selecting the centers of these non-overlapping discs, you might need to increase the size beyond $M \times N$. One possible way to handle this situation is that if for selecting n centers, you have tried $2 \times n$ random points and still they don't satisfy the desired constraint of non-overlapping discs, then you can increase the image size to $2M \times 2N$ and recursively do this till you get n centers with required properties. The parameters V_f and V_b are optional. If they are not provided then disc will be of black color (0) and background will be of white color (255). If they are provided, then the intensity values inside the disc should be uniformly distributed over all intensity values present in the array V_f and the intensity values outside the disc (background) should be uniformly distributed over all intensity values present in the array V_b. For example, we may specify V_f = [0 : 1 : 128] and the background intensities in the range V_f = [129 : 2 : 255]. Write a Matlab/Python script test.m which will call the earlier defined function with couple of settings of input parameters and save the output image after displaying it.

Note: Allowed in-built Matlab/Python functions: rand, imshow, imwrite.

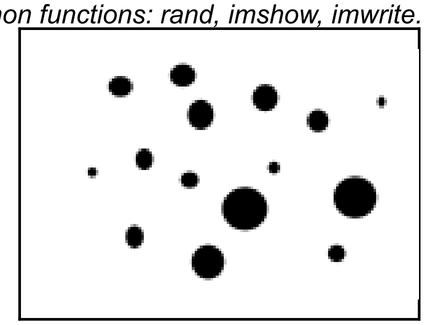

Figure 2: Sample Disc Image

Page 2

4. (Digital Image Creation - Rectangles) Write a generic Matlab/Python function

$$[I] = create\_rectangles(M, N, border, n, w_1, w_2, alpha, orientation, V_f, V_b), \quad (3)$$

which will create a 8-bit grayscale image similar to Figure 3. This image is of size $M \times N$, additionally it has a black border around it of thickness *border*. Thus, the size of complete image is $(M{+}2{\times}border){\times}(N{+}2{\times}border)$. This image contains n non-overlapping rectangles with height to width ratio fixed as *alpha*, with width uniformly distributed in $[w_1, w_2]$. One of the corners of each of these rectangles are obtained from a uniform distribution over all possible pixel locations in the image. The selected corners should be such that the rectangles are non-overlapping. For randomly selecting the corners of these non-overlapping rectangles, you might need to increase the size beyond $M \times N$. One possible way to handle this situation is that if for selecting n corners, you have tried $2 \times n$ random points and still they don't satisfy the desired constraint of non-overlapping rectangles, then you can increase the image size to $2M \times 2N$ and recursively do this till you get n corners with required properties. The parameter *orientation* will be subset of $\{1,2\}$ indicating the two possible orientations of these rectangles. The orientations of the rectangles should be uniformly distributed over the orientations listed in the parameter *orientation*. The parameters V_f and V_b are optional. If they are not provided then rectangles will be of black color (0) and background will be of white color (255). If they are provided, then the intensity values inside the rectangles should be uniformly distributed over all intensity values present in the array V_f and the intensity values outside the rectangles (background) should be uniformly distributed over all intensity values present in the array V_b. For example, we may specify $V_f = [0 : 1 : 128]$ and the background intensities in the range $V_f = [129 : 2 : 255]$. Write a Matlab/Python script test.m which will call the earlier defined function with couple of settings of input parameters and save the output image after displaying it.

Note: Allowed in-built Matlab/Python functions: rand, imshow, imwrite.

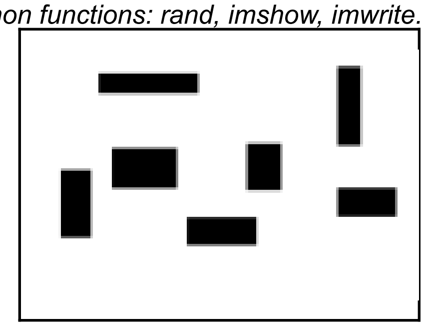

Figure 3: Sample Rectangles Image

5. (Arithmetic Coding) Write a function and corresponding test script for performing Arithmetic Coding. Input to the function should be: 1) either a matrix with two columns (first column is symbol number and second column is its probability and each row indicating a symbol) or a single channel image, and 2) a number 'N' which indicates the number of symbols which are to be coded together (if the total length of the message to be coded is not a multiple of N, then last message sequence will contain fewer than N symbols), and 3) (optional argument) a 1-D array listing symbols/numbers in the message to be coded (if this argument is missing then the function should return the result of applying Arithmetic Coding on the input single channel image.)

   Output of the function should be the result of applying Arithmetic Coding on the specified data, specified as a column vector of appropriate size and the probability table used for encoding. For each sub-message ('N' symbols which are to be coded together), the function should give the decimal number with the least number of digits needed to represent that sequence of symbols of length 'N'.

   Your test script should apply this on a sample image and save the output as a .mat file (or equivalent for Python).