CS 663 : Digital Image Processing : Assignment 1

Instructor : Suyash P. Awate

Note: The input data / image(s) for a question is / are present in the corresponding data/ subfolder.

5 points are reserved for submission in the described format.

For all questions, display images using a colormap that uses at least 200 colors/intensities. Also, display the colormap/colorbar alongside each image. Don't round the resulting images to integers; use a floating-point data type for the pixel value.

1. (40 points) Image Subsampling and Interpolation.

   (a) (2 points) Image Shrinking.

   Input image: `data/interp/suit.png`

   Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes.

   Shrink the image size by a factor of d along each dimension using image subsampling by sampling/selecting every d-th pixel along the rows and columns.

   • Write a function `myImageShrink()` to implement this.

   • Display the original and subsampled images, with the correct aspect ratio, for $d = 2$ and $d = 3$ appropriately to clearly show the Moire effects. Display the pixel units along each axis and the colorbar.

   (b) (6 points) Image Enlargement using Nearest-Neighbor Interpolation.

   Input image: `data/interp/random.png`

   Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes. Consider this image as the data. Consider the number of rows as M and the number of columns as N.

   Resize the image to have the number of rows $= 300(M - 1) + 1$ and the number of columns $= 300(N - 1) + 1$, such that the first and last rows, and the first and last columns, in the original and resized images represent the same data.

   Use nearest-neighbor interpolation for resizing.

   • Write a function `myNearestNeighborInterpolation()` to implement this.

   • Display the original and resized images. Display the pixel units along each axis and the colorbar. Use the "jet" colormap.

   (c) (6 points) Image Enlargement using Bilinear Interpolation.

   Input image: `data/interp/random.png`

   Redo the previous problem using bilinear interpolation.

   • Write a function `myBilinearInterpolation()` to implement this, using exactly the algorithm described in the lecture slides.

   • Display the original and resized images. Use the "jet" colormap.

   (d) (6 points) Image Enlargement using Bicubic Interpolation.

   Input image: `data/interp/random.png`

   Redo the previous problem using bicubic interpolation, using exactly the algorithm described in the lecture slides.

   • Write a function `myBicubicInterpolation()` to implement this.

   • Display the original and resized images. Use the "jet" colormap.

   (e) (10 points) Image Rotation using Bilinear Interpolation.

   Input image: `data/interp/main.png`

Rotate the entire image (all objects in the image) such that the lamp post in the front of the building becomes vertical (instead of slanted). Keep the size of the rotated image same as the size of the input image.

• Write a function `myImageRotationUsingBilinearInterp()` to implement the rotation using the bilinear interpolation function that you coded before.

• Write a function `myImageRotationUsingNearestNeighborInterp()` to implement the rotation using the nearest-neighbor interpolation function that you coded before.

• Display the original and rotated images using both kinds of interpolation.

(f) (10 points) Image Downsampling and Upsampling.

Input images: `data/interp/ct.mat`

The file contains two real-valued human chest-CT 2D images, i.e., "original" and "subsampled".

Enlarge the subsampled image using all 3 kinds of interpolation (nearest neighbor, bilinear, bicubic) to make the enlarged image the same size as the original image. Ensure that the first and last rows in the subsampled image map, respectively, *exactly* to the first and last rows in the enlarged image. Similarly, ensure that the first and last columns in the subsampled image map, respectively, *exactly* to the first and last columns in the enlarged image.

Display (i) the original image, (ii) the 3 enlarged images comparing them to the original side by side, and (iii) 3 difference images (i.e., original - enlarged). Use the "jet" colormap to display all images.

Show all enlarged images using exactly the same limits on the colormap; what do you infer regarding the quality of the interpolation schemes ?

Show all difference images using exactly the same limits on the colormap; what do you infer regarding the quality of the interpolation chemes ?

Report the root mean square errors (RMSE) for the 3 enlarged images with respect to the original image as the ground truth.

2. (20 points) Thresholding.

   Input image 1: `data/thresh/receipt.png`

   Input image 2: `data/thresh/blackboard.png`

   Input image 3: `data/thresh/lilavati.tif`

   Input image 4: `data/thresh/qr.png`

   (a) (2 points)

   Write a function for **manual thresholding** with the aim of binarizing the given images such that the text/writing/illustrations are black and the background is white.

   Tune the parameters as well as possible.

   Display the original and thresholded images side by side.

   Explain why it should work well, or it shouldn't work well on the given images.

   (b) (6 points)

   Write a function for **Otsu thresholding** with the same aim as before.

   Tune the parameters as well as possible.

   Display the original and thresholded images side by side.

   Explain why it should work well, or it shouldn't work well on the given images.

   (c) (12 points)

   Write a function for **local/adaptive thresholding** with the same aim as before.

   Tune the parameters as well as possible.

   Display the original and thresholded images side by side.

   Display the image of per-pixel threshold values used in the local/adaptive thresholding.

   Explain why it should work well, or it shouldn't work well on the given images.

3. (20 points) Contrast Enhancement.

   (a) (2 points)

   Input image: `data/hist/leh.png`

   • Write code for a function `myLinearContrastStretch()` to perform linear contrast stretching on the luminance component.

   • Show the original and contrast-enhanced images side by side. Show their histograms.

   (b) (6 points)

   Input image: `data/hist/leh.png`

   • Write code for a function `myHistEqualize()` to perform nonlinear contrast stretching using histogram equalization (HE) on the luminance component.

   • Show the original and contrast-enhanced images side by side. Show their histograms.

   (c) (6 points)

   Input image 1: `data/hist/canyon.png`

   Input image 2: `data/hist/retina.png`

   • Write code for a function `myCLAHE()` to perform nonlinear contrast stretching using contrast-limited adaptive histogram equalization (CLAHE) on the luminance component. For canyon.png, the goal is to clarify the dark regions in the image resulting from shadows, e.g., regions in front of the boulder on the bottom right.

   Manually tune the number-of-bins parameter, the window-size parameter, and the histogram-threshold parameter $\in [0, 1]$ to appropriate values in order to perform contrast enhancement for objects in the image, while preventing excessive noise amplification. For pixels close to the boundary, where the window falls outside the image, use only the pixels that are within the window and within the image (i.e, effectively cropping the window to restrict it within the image boundaries).

   • Redo CLAHE with neighborhood sizes chosen to be (i) significantly larger and (ii) significantly smaller than that chosen for the previous result in order to clearly show the effects of (i) low contrast improvement and (ii) excessive noise amplification.

   • Redo CLAHE with (i) the same window size as before and (ii) the histogram-threshold parameter set to a value that is half of the value tuned before.

   Show the original and contrast-enhanced images, under various parameter settings for all questions above, side by side. Show their histograms.

   (d) (6 points) Histogram Matching.

   Input image 1: `data/hist/retina.png`

   Input image 2: `data/hist/retinaRef.png`

   • Write code for a function `myHistMatch()` to perform histogram matching on the first image (retina) to modify its luminance and chroma components to match

those of the second image (retinaRef). Ignore the black background in both images; thus, the histogram matching transformation should be computed and applied only to the foreground.

Manually tune the number-of-bins parameter for histogram construction as best as you can. What happens if the number of bins is too small ? What happens if it is too large ?

• Show the original, reference, and histogram-matched images, under various parameter settings for all questions above, side by side. Show their histograms.