ENGN1610&2605 Image Understanding

Lab02: Filtering and Convolution, Neighborhood Operations, Deblurring

The goal of this lab is to

- Understand how convolution works
- Implement various filters, and understand their effects on the image
- Understand various types of image noise
- Techniques to restore severely degraded images

Part I: 2D convolution

Problem 1. Implement a 2D convolution
As you have learned in the class, convolving filters with images enables smoothing or enhancing the image data. Equation 1 shown below represents a discrete 2D convolution formula you need to implement in this lab.

$$y[m,n] = x[m,n] * h[m,n] = \sum_{j=-\omega}^{\omega} \sum_{i=-\omega}^{\omega} x[i,j]h[m-i,n-j] \tag{1}$$

Write a MATLAB function that does a 2D convolution with a filter and an image as inputs and the filtered image as the output.

```
function [output]=my_conv(image,filter)
```

- You will need to do a raster scan over the input image, and for each pixel, apply the filter, this will require four loops
- In cases when your filter falls outside the boundary of the image, pad your image data with zeros.
- Set the output to an empty matrix with same size as the input image.

It is very **important** that your convolution works correctly as this function will be used for future problems. To verify whether the correctness, compare your results to MATLAB's `imfilter` function. The results should be equal.

```
image=magic(10);
filter=ones(3,3);
matlab_output=imfilter(image,filter);
my_output=my_conv(image,filter);
```

Problem 2. Play with the filters
Using the code you have written in problem 1, implement the following filters on the test images. (Remember that the output of your convolution function using all the following filters can be verified by MATLAB's `imfilter` function.)

- Smoothing Filters: (1) Box Filter (2) Weighted Box Filter (3) Horizontal Filter (4) Vertical Filter (5) Circular-Shape Filter (6) 5×5 Box Filter (7) 5×5 Circular Shape Filter, Try this filters out on the images below in Figure 1

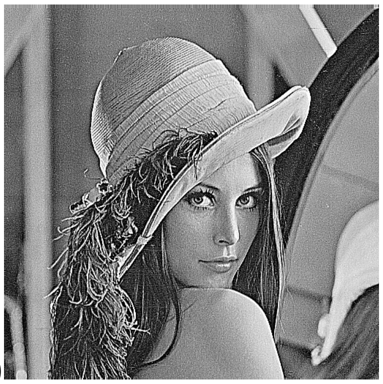
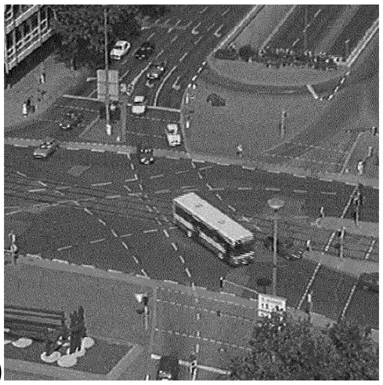
(a) (b)

Figure 1: Images to Test on. (a) A very sharp image. (b) An image degraded by noise.

$$(1)\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \quad (2)\ \frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} \quad (3)\ \frac{1}{3}\begin{bmatrix}0&0&0\\1&1&1\\0&0&0\end{bmatrix} \quad (4)\ \frac{1}{3}\begin{bmatrix}0&1&0\\0&1&0\\0&1&0\end{bmatrix}$$

$$(5)\ \frac{1}{5}\begin{bmatrix}0&1&0\\1&1&1\\0&1&0\end{bmatrix} \quad (6)\ \frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix} \quad (7)\ \frac{1}{25}\begin{bmatrix}0&1&1&1&0\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\0&1&1&1&0\end{bmatrix}$$

- Enhancment Filters: (1) 4×4 Laplacian (2) 8×8 Laplacian. Apply the filters to the images in Figure 2.

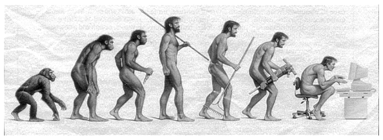
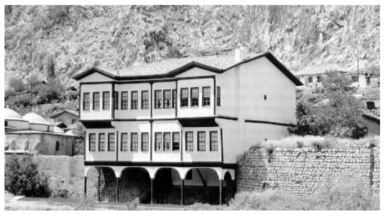
 

Figure 2: Images to test on for enhancement and edge filters.

$$(1)\ \begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix} \quad (2)\ \begin{bmatrix}1&1&1\\1&-8&1\\1&1&1\end{bmatrix}$$

- Edge Filters: (1) Prewitt X (2) Prewitt Y (3) Sobel X (4) Sobel Y. Apply these filters to the images in Figure 2

$$(1)\begin{bmatrix}1 & 0 & -1\\1 & 0 & -1\\1 & 0 & -1\end{bmatrix}\quad(2)\begin{bmatrix}1 & 1 & 1\\0 & 0 & 0\\-1 & -1 & -1\end{bmatrix}\quad(3)\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix}\quad(4)\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}$$

Please answer the following questions:

- Smoothing Filters: Give some general comments on each filter. What do you observe? Which filter performs the best on each image? What is the difference between the box filter, and the weighted box filter?
- Enhancement Filters: Give some general comments on what this filter does to the image. Which filter does the best on each image?
- Edge Filters: Which filter performs the best on each image? What effect does weighting the center pixel have on the performance? What difference do you notice between the enhancement filters and edge filters?

Problem 3. Gaussian Smoothing

This problem aims to investigate how a kernel can be described as a Gaussian function. You will have to sample a zero mean symmetric Gaussian, Equation 2.

$$f(x,y) = e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)} \tag{2}$$

Implement the function

```
function [filter]=gauss_kernel(filter_size,sigma);
```

- You will have to sample the Gaussian on a grid size equivalent to your window. For example, if the window size is 3×3, you will evaluate your Gaussian at x values -1, 0, 1 and y values -1, 0, 1. You are allowed to use `meshgrid` function to help with this.
- Remember to normalize your filter by the sum of all the elements in the filter.
- You can verify your answer using MATLAB's function, *i.e.*, `H = fspecial('gaussian',HSIZE,SIGMA`
- Report your results with these combinations on the images in Figure 1
  1. Window Size of 3, $\sigma = 0.5$, 1, and 2
  2. Window Size of 5, $\sigma = 0.5$, 1, and 2
  3. Window Size of 7, $\sigma = 0.5$, 1, and 2
- Discuss what the effect of increasing window size is, what the effect of increasing sigma is, and compare the Gaussian kernel with other smoothing filters you tried.

Problem 4. Non-Linear Filters
Using the same code you implemented for a 2D convolution in problem 1, implement some non-linear filters. Here we are dealing with order-statistic filters. In each region, you will return either the median, the max, or the min, rather than summing the values within the window of the filter. Apply the non-linear filters to the image in Figure 3 to get rid of the noise.

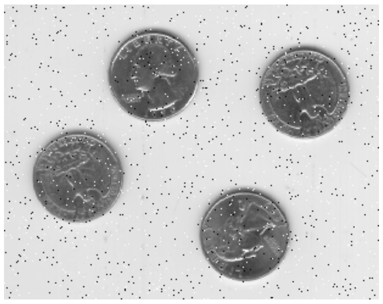


Figure 3: Noisy Coins

- Min Filter: Use window sizes of 3×3, 5×5, and 7×7, output the pixel value as the minimum value in the window.
- Max Filter: Use window sizes of 3×3, 5×5, and 7×7, output the pixel value as the maximum value in the window.
- Median Filter: Use window sizes of 3×3, 5×5, and 7×7, output the pixel value as the median value in the window.

Report the best setting for each type of filter. Out of the three types of the filters, which filter performs the best.

Problem 5. Salt and Pepper vs White Noise Two images you have used so far, Figure 4, are degraded by noise. Each has a different type of noise: the coin image is contaminated by **salt and pepper noise**, and the traffic image is degraded by **white noise**.

- Use the Gaussian smoothing you have implemented in problem 2 (pick your own window size, and sigma), and also pick one of the box filters. Apply them to the noisy coin image.
- Apply the median, max, and min filters (pick the best window size) you have implemented in problem 4 on the noisy traffic image.
- Which type of filter is best for white noise, and why? Which type of filter is best for salt and pepper noise, and why?

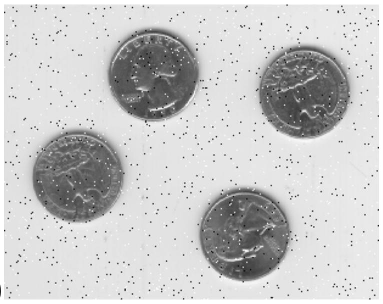
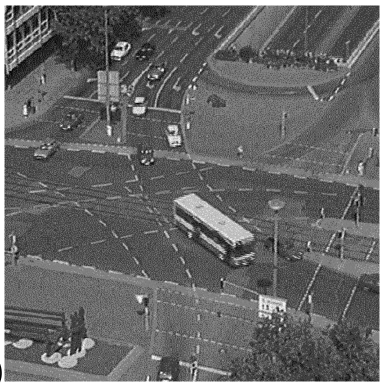
(a)  (b) 

Figure 4: Images displaying different types of noise. (a) Salt and pepper noise. (b) White noise.

Problem 6. Separable Convolution 2D
Certain filters have a special property that they are **separable**, *i.e.*, the filter represented by a $M \times N$ matrix can be decomposed into $M \times 1$ and $1 \times N$ matrices where the outer product returns the original. A 3×3 averaging filter, Equation 3, for example, is a separable filter. Gaussian smoothing is another well-known separable matrix.

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \begin{bmatrix}\frac{1}{3}\\ \frac{1}{3}\\ \frac{1}{3}\end{bmatrix}\begin{bmatrix}\frac{1}{3} & \frac{1}{3} & \frac{1}{3}\end{bmatrix} \tag{3}$$

The separable property of a filter is beneficial in terms of computation efficiency. Specifically, convolve the image with $M \times 1$ filter in the vertical direction, then convolve the result with a $1 \times N$ filter in the horizontal direction. The first vertical 1D convolution requires M multiplications while the horizontal 1D convolution needs N multiplications; thus only $M + N$ products is necessary.

However, the separable 2D convolution requires additional storage (buffer) to keep the intermediate computations. That is, if you do the vertical 1D convolution first, you must preserve the results in a temporary buffer in order to use them for the horizontal convolution subsequently.

Notice that convolution is associative: the result is same, even if the order of convolution is changed. Thus, convolving the image in the horizontal direction first then the vertical direction later gives you the same result.

Implement the function

```
function [output]=conv_separable(image,filter)
```

The convolution of an $N \times N$ image with an $M \times M$ filter kernel requires a time proportional to N^2M^2. In other words, each pixel in the output image depends on all the pixels in the filter kernel. In comparison, convolution by separability only requires the time proportional to N^2M. For filter kernels that are hundreds of pixels wide, this technique reduces the execution time by a factor of hundreds. Use the MATLAB timer `tic, toc` command to show and compare the required computation time.

```
tic
function [output]=conv_separable(image,filter)
toc
```

Answer the following questions:

- Which filters in problem 2 are separable? Describe a way to test whether a filter is separable.
- Compare the computational time of convolving the image with the following filters using the separable and the brute force approach. Use the image in Figure 1 (a).

  1. 5×5 Box Filter

$$\frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{4}$$

  2. 5×5 circular shape filter

$$\frac{1}{25}\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{5}$$

  3. 7×7 Gaussian with sigma of 0.5
  4. 17×17 Gaussian with sigma of 0.5
  5. 35×35 Gaussian with sigma of 2

Part II: Image Restoration

A model for image degradation can be seen in Equation 6

$$g = h \otimes f + n \tag{6}$$

where f is the original undistorted image, g is the distorted noisy image, h is the point spread function (PSF) of the system, $\otimes$ is the convolution operator, and n is the corrupting noise.

One method to recover the image is by using the Lucy-Richardson algorithm. This iterative algorithm can be succinctly expressed as

$$\hat{f}_{k+1} = \hat{f}_k \left(h * \frac{g}{h \otimes \hat{f}_k} \right) \tag{7}$$

where $\hat{f}_k$ is the estimate of f after k iterations, $*$ is the correlation operator. The image $h \otimes \hat{f}_k$ is referred to as the reblurred image.

Problem 7. Deblurring - Lucy Richardson
Implement this algorithm

```
function [restored_image]=lucy_richardson(degraded_image,psf);
```

Your results might vary, but an example of what you should get is shown:

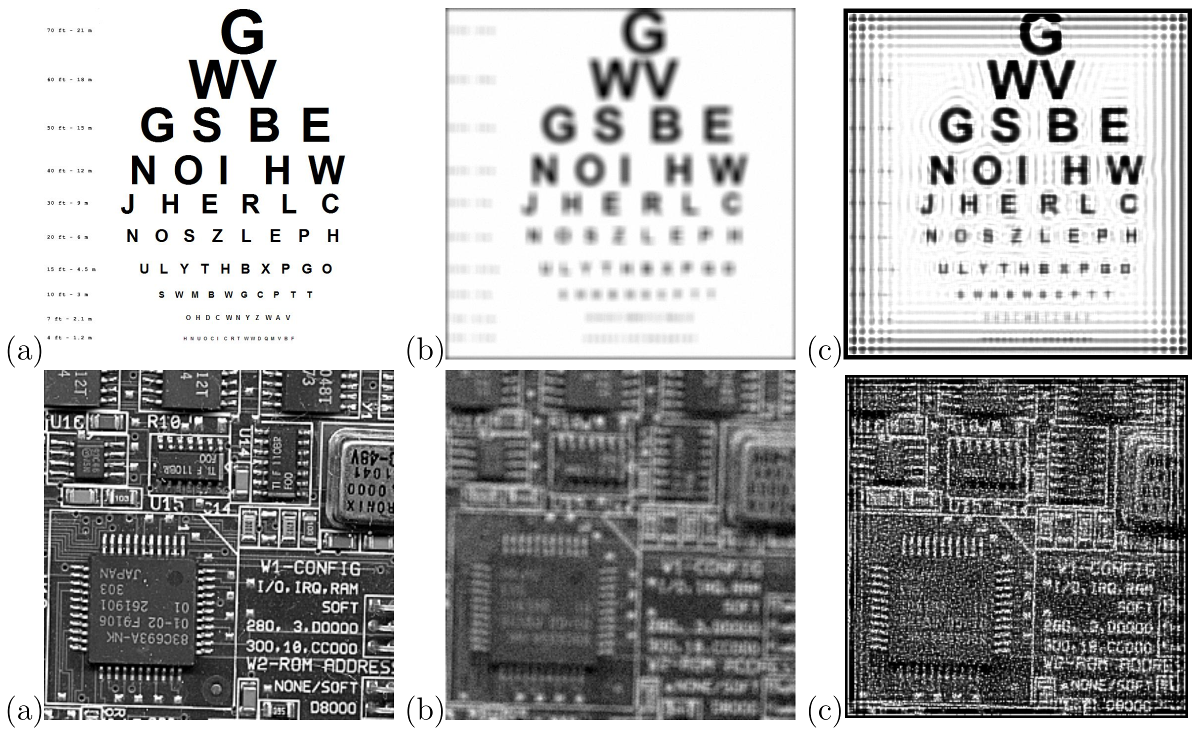


Figure 5: (a) Original image. (b) Degraded image. (c) Restored image using Lucy-Richardson algorithm.

- For the first iteration $\hat{f}_1$, set it to the degraded image g, or you can try your own initialization.
- For the convolution operation in Equation 7 you can use the MATLAB's function `conv2`.
- For the correlation operation you can use MATLAB's `filter2` function.

- Use `load(<MAT_FIlE>)` to load the `.mat` file of PSF.
- Try various number of iterations. In general, more iterations lead to better results.

Compare your results against MATLAB's Lucy-Richardson implementation; you can check it by putting `help deconvlucy` in the MATLAB command window. Show images from both your implementation and the MATLAB's.

Now try restoring the two degraded images using the median filter you wrote earlier with various window size. Compare those results to the Lucy-Richardson algorithm. Also try any of the sharpening filters you implemented and see what you get.

Challenge You might have noticed that the restored images contains some ringing. What causes this? What can be done to deal with this? Submit your results with your solution.