Lecture 2

(Introduction to Digital Image Analysis)

Outlines:

1. Pre-processing
2. Region of Interest (ROI) Image Geometry
3. Arithmetic and Logic Operations
4. Spatial Filters
5. Binary Image Analysis

References:

1. Chapter 3, Scott E Umbaugh, 2018, "Digital Image Processing and Analysis Applications with MATLAB and CVIPtools", 3th Ed, Taylor & Francis.

1. Digital Image Analysis System Model

The image analysis process, illustrated in figure 39, can be broken down into three primary stages:

1. Preprocessing.
2. Data Reduction.
3. Features Analysis.

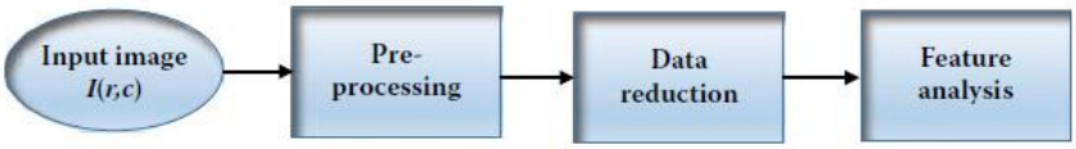

Figure 39. Image Analysis.

To analyze the image useful higher level information must be extracted from the raw pixel data.

1.1 Preprocessing

The preprocessing algorithm, techniques, and operators are used to perform initial processing that makes the primary data reduction and analysis task easier (see figure 40). It is a stage where the requirements are typically obvious and simple, includes operations related to:

- Gray level, or spatial quantization (reducing the number of bits per pixel or the image size).
- Is used to remove noise and eliminate irrelevant, visually unnecessary information, (Noise is unwanted information that can result from the image acquisition process)
- Extracting regions of interest for further processing.
- Performing a basic mathematical operation on the images.
- Simple enhancing specific image features.
- Color space transforms
- Reducing data in both the resolution and brightness.

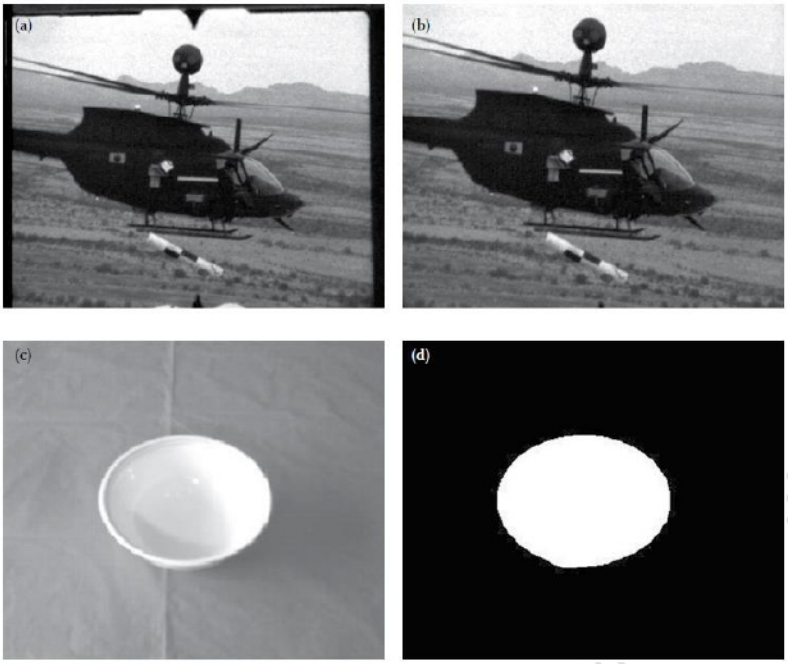

Figure 40. Preprocessing examples. (a) an image needing to border removal, (b) the image after the border is removed, (c) an image where only shape information is necessary, and (d) the image after unnecessary information removal which only the object shape.

1.1.1 Region of Interest (ROI) Image Geometry

Investigation of a specific area within the image, called an ROI, is required in image analysis. We need operations that modify the spatial coordinates of the image, and these are categorized as image geometry operations. The image geometry operations include crop, zoom, enlarge, shrink, translate, and rotate. The image crop process is the process of selecting a portion of the image, a sub-image, and cutting it away from the rest of the image, that is how the border was removed in figure 2b. Once a sub-image is cropped from the original image, we can zoom in on it by enlarging it. Image enlargement may allow visible recognition of image degradation, helping in the selection of a restoration model.

Zoom process: It can be performed in numerous ways:

- Zero-order hold
- First-order hold
- Convolution

1. **Zero-order hold** (**Nearest neighbor interpolation**): is performed by **repeating previous pixel values**, thus creating a **blocky effect** (checkerboard effect). The implementation of the zero-order hold is **straightforward**.

Pixel replication (a special case of nearest neighbor interpolation) is used to increase the size of an image an integer number of times. The example below shows 8-bit image zooming by 2x (2 times) (2N×2N) using nearest neighbor interpolation.

$$\begin{bmatrix}69 & 50 & 80\\ 45 & 60 & 66\\ 30 & 55 & 80\end{bmatrix} = \begin{bmatrix}69 & 69 & 50 & 50 & 80 & 80\\ 45 & 45 & 60 & 60 & 66 & 66\\ 30 & 30 & 55 & 55 & 80 & 80\end{bmatrix} = \begin{bmatrix}69 & 69 & 50 & 50 & 80 & 80\\ 69 & 69 & 50 & 50 & 80 & 80\\ 45 & 45 & 60 & 60 & 66 & 66\\ 45 & 45 & 60 & 60 & 66 & 66\\ 30 & 30 & 55 & 55 & 80 & 80\\ 30 & 30 & 55 & 55 & 80 & 80\end{bmatrix}$$

Original image | image with rows expanded | image with rows and columns expanded

2. **First-order hold (Bilinear interpolation):** is performed between **adjacent pixels,** thus creating a **blurring effect**. A comparison of the images resulting from these two methods is shown in figure 5. The first-order hold is **more complicated**, the easiest way to do this is to find the average value between two pixels and use that as the pixel value between those two; this can be done for the rows first, as follows:

Original image array

$$\begin{bmatrix}8 & 4 & 8\\ 4 & 8 & 4\\ 8 & 2 & 8\end{bmatrix}$$

Image with rows expanded

$$\begin{bmatrix}8 & 6 & 4 & 6 & 8\\ 4 & 6 & 8 & 6 & 4\\ 8 & 5 & 2 & 5 & 8\end{bmatrix}$$

The first two pixels in the first row are averaged, (8 + 4)/2 = 6, and this number is inserted in between those two pixels. This is done for every pixel pair in each row. Next, take that result and expand the columns in the same way, as follows:

Image with rows and columns expanded

$$\begin{bmatrix}8 & 6 & 4 & 6 & 8\\ 6 & 6 & 6 & 6 & 6\\ 4 & 6 & 8 & 6 & 4\\ 6 & 5.5 & 5 & 5.5 & 6\\ 8 & 5 & 2 & 5 & 8\end{bmatrix}$$

This method will allow us to enlarge an (N×N) sized image to a size of ((2N–1)×(2N–1)), and can be repeated as desired. The bilinear interpolated images are smoother than those resulting from nearest-neighbor interpolation.

Figure 41. Zooming Methods (a) Original image. The ape's face will be zoomed by a factor of 5, (b) image enlarged by zero-order hold, notice the blocky effect, (c) image enlarged by first-order hold, note the smoother effect.

3. Convolution: requires a **mathematical process to enlarge an image**. This method required two steps:

1. Extend the image by adding rows and columns of zeros between the existing rows and columns.
2. Perform the convolution.

The image is extended as follows:

Original image array

$$\begin{bmatrix} 3 & 5 & 7 \\ 2 & 7 & 6 \\ 3 & 4 & 9 \end{bmatrix}$$

Image extended with zeros

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 5 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 7 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 4 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Next, we use what is called a convolution mask which is slid across the extended image and a simple arithmetic operation is performed at each pixel location.

Convolution mask for the first-order hold

$$\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}$$

The convolution process requires us to overlay the mask on the image, multiply the coincident values, and sum all these results. This is equivalent to finding the vector inner product of the mask with the underlying sub-image. For example, if we put the mask over the upper left corner of the image, we obtain (from right to left and top to bottom):

$$1/4(0)+1/2(0)+1/4(0)+1/2(0)+1(3)+1/2(0)+1/4(0)+1/2(0)+1/4(0)=3$$

The next step is to slide the mask over by one pixel and repeat the process, as follows:

$$1/4(0)+1/2(0)+1/4(0)+1/2(3)+1(0)+1/2(5)+1/4(0)+1/2(0)+1/4(0)=4$$

This process continues until we get to the end of the row, each time placing the result of the operation in the location corresponding to the center of the mask. Once the end of the row is reached, the mask is moved down one row and the process is repeated row by row until this procedure has been performed on the entire image; the process of sliding, multiplying, and summing is called convolution (see figure 42). Note that the output image must be put in a separate image array, called a buffer. If we designate the convolution mask as M(r,c) and the image as I(r,c), then the convolution equation is given by:

$$\sum_{x=-\infty}^{+\infty}\sum_{y=-\infty}^{+\infty} I(r-x,c-y)M(x,y)$$

The zero-order hold can also be achieved by extending the image with zeros and using the following convolution mask:

Zero-order hold convolution mask

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

For this mask, the result will be put in the pixel location corresponding to the lower right corner, as there is no center pixel. These methods will only allow the enlargement of an image by a factor of (2N−1).

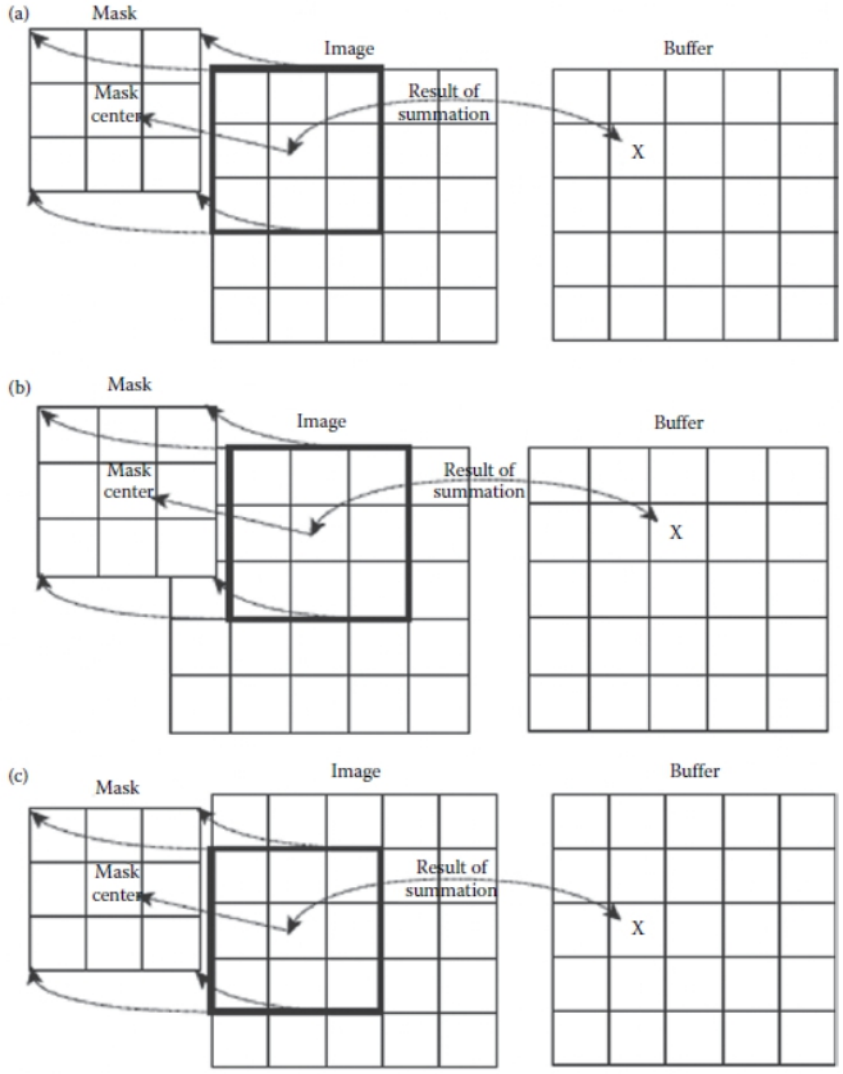

Figure 42. The convolution process.

(a) Overlay the convolution mask in the upper left corner of the image. Multiply coincident terms, sum, put the result into the image buffer at the location that corresponds to the mask's current center, which is (r, c)=(1,1), (b) Move the mask one pixel to the right, multiply coincident terms, sum, and place the new result into the buffer at the location that corresponds to the new center location of the convolution mask, now at (r, c)=(1,2). Continue to the end of the row, (c) Move the mask down one row and repeat the process until the mask is convolved with the entire image.

Note that we **lose** the outer row(s) and column(s).

Translation and rotation processes

Two other operations of interest for the ROI image geometry are translation and rotation. These processes may be performed for many application, for example, to align an image with a known template in a pattern matching process or to make certain image details easier to see. The translation process can be done with

the following equations:

$$r' = r + r_0$$
$$c' = c + c_0$$

Where 'r and 'care the new coordinates, r and c are the original coordinates and r_0 and c_0 are the distances to move, or translate, the image. The rotation process requires the use of these equations:

$$\hat{r} = r(\cos\theta) + c(\sin\theta)$$
$$\hat{c} = -r(\sin\theta) + c(\cos\theta)$$

Where ˆr and ˆc are the new coordinates, r and c are the original coordinates, and θ is the angle to rotate the image. θ is defined in a clockwise direction from the horizontal axis at the image origin in the upper left corner. The rotation and translation process can be combined into one set of equations:

$$\hat{r}' = (r + r_0)(\cos\theta) + (c + c_0)(\sin\theta)$$
$$\hat{c}' = -(r + r_0)(\sin\theta) + (c + c_0)(\cos\theta)$$

Where ˆ'r and ˆc' are the new coordinates and r, c, r_0,c_0, and θ are previously defined.

There are some practical difficulties with the direct application of these equations. When translating, what is done with the "leftover" space? If we move everything one row down, what do we put in the top row? There are two basic options: fill the top row with a constant value, typically black (0) or white (255), or wrap-around by shifting the bottom row to the top, as shown in figure 7. Rotation also creates some practical difficulties. figure 44 (a) illustrates, the image may be rotated off the image plane. Although this can be fixed by a translation back to the center (Figure 44 (b and c)), leftover space appears in the corners. This space can be filled with a constant or we can extract the central, rectangular portion of the image and enlarge it to the original image size

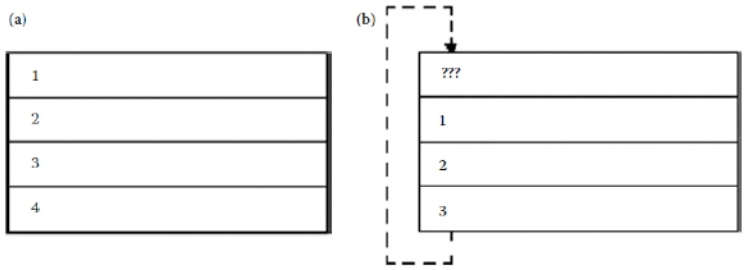

Figure 43. Image Translation. Involves movement within the image by modifying the row and column coordinates.(a) a four row image before translation down by one row, $r0 = 1$, (b) after translation the row that was translated out of the image space is either inserted into the blank space (wrap-around), or discarded and the empty space is filled with a constant

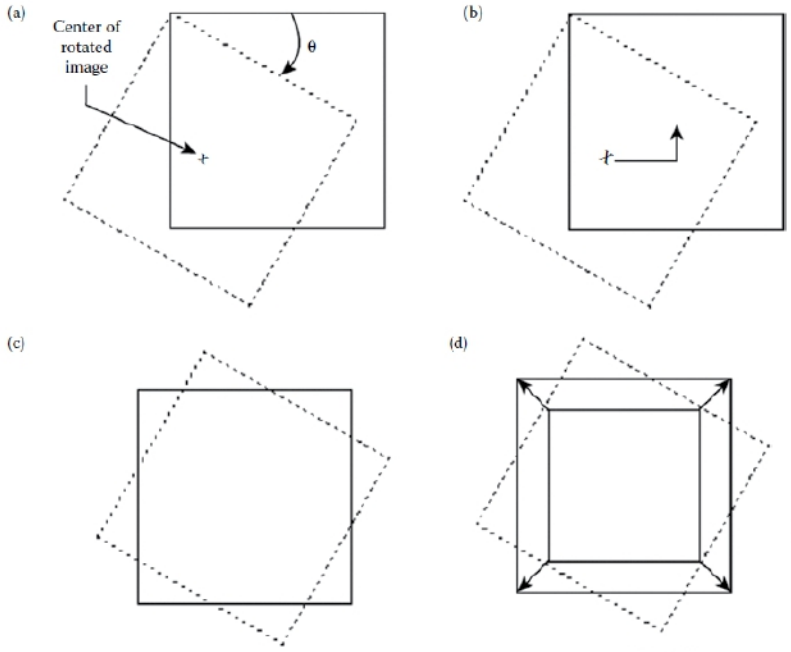

Figure 44. Image Rotation. (a) Image is rotated off the screen, (b) Fix by translating toward center, (c) Translation complete, and (d) Crop and enlarge if desired

1.1.2 Arithmetic and Logic Operations

Four arithmetic operations:

$$s(x, y) = f(x, y) + g(x, y)$$

$$d(x, y) = f(x, y) - g(x, y)$$

$$p(x, y) = f(x, y) \times g(x, y)$$

$$v(x, y) = f(x, y) / g(x, y)$$

Where, x = 0,1,2,…,M-1, y = 0,1,2,….N-1. All images are of size M (rows) × N (columns)

Arithmetic and logic operations are often applied as preprocessing steps in image analysis to combine images in various ways. Addition, subtraction, division, and multiplication comprise the arithmetic operations, whereas AND, OR, and NOT make up the logic operations. These operations are performed on two images, except the NOT logic operation which requires only one image, and are **done on a pixel by pixel basis**.

To apply the arithmetic operations to two images, we simply **operate on corresponding pixel values**. For example, to add images I_1 and I_2 to create I_3:

$$I_1(r,c)+I_2(r,c)=I_3(r,c)$$

$$I_1=\begin{bmatrix}3 & 4 & 7\\ 3 & 4 & 5\\ 2 & 4 & 6\end{bmatrix}\quad I_2=\begin{bmatrix}6 & 6 & 6\\ 4 & 2 & 6\\ 3 & 5 & 5\end{bmatrix}\quad I_3=\begin{bmatrix}3+6 & 4+6 & 7+6\\ 3+4 & 4+2 & 5+6\\ 2+3 & 4+5 & 6+5\end{bmatrix}=\begin{bmatrix}9 & 10 & 13\\ 7 & 6 & 11\\ 5 & 9 & 11\end{bmatrix}$$

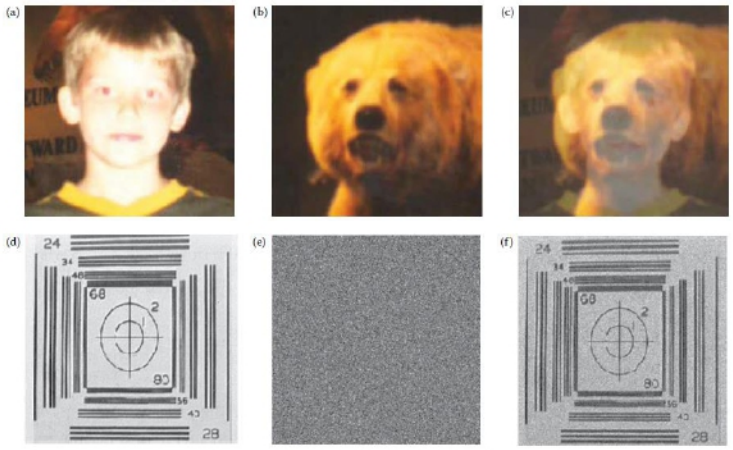

Figure 45. Image Addition Examples. This example shows one step in the image morphing process where an increasing percentage of the one image is slowly added to another image, and a geometric transformation is usually required to align the images. (a) first original, (b) second original, (c) addition of 50% of (a) and 100% of (b). The next example shows adding noise to an image which is often useful for developing image restoration models, (d) original image, (e) Gaussian noise, variance = 400, mean = 0, and (f) addition of images (d) and (e).

Addition is used to combine the information in two images. Applications include the development of image restoration algorithms for modeling additive noise, and for special effects, such as image morphing, in motion pictures (figure 45).

Subtraction of two images is often used to **detect motion**. Consider the case where nothing has changed in a scene; the image resulting from the subtraction of two sequential images is filled with zeros a black image. If something has moved in the scene, subtraction produces a nonzero result at the location of movement, enabling detection of both the motion and the direction. If the time between image acquisitions is known, the moving object's speed can also be calculated. Figure 10 illustrates the use of subtraction for motion detection. Here we can learn two things:

- We must threshold the result and,
- The process is imperfect and will require some further processing.

Multiplication and division are used to adjust the brightness of an image. This is done on a pixel by pixel basis and the options are to multiply or divide an image by a constant value, or by another image. Multiplication of the pixel

values by a value greater than one will brighten the image (or division by a value <1), and division by a factor greater than one will darken the image (or multiplication by a value < 1). Brightness adjustment by a constant is often used as a preprocessing step in image enhancement and is shown in figure 48.

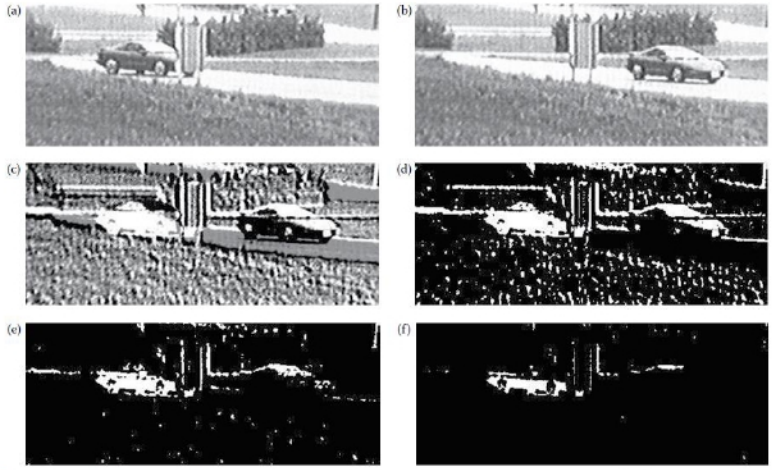

Figure 46. Image Subtraction (a) Original scene, (b) same scene later, (c) enhanced version of scene a image subtracted from scene b, (d) the subtracted image with a threshold of 150, (e) the subtracted image with a threshold of 175, and (f) the subtracted image with a threshold of 200.

Applying multiplication or division to two images can be done to model a multiplicative noise process, or to combine two images in unique ways for special effects. In figure 48, the results of multiplying two images together are shown. The first set of images superimposes an x-ray of a hand onto another image, and the second set shows how multiplication can be used to add texture to a computer generated image. In both cases, the output image has been remapped to byte data range (0–255) for display purposes. Note that multiplication and division of images can also be used for image filtering in the spectral domain.

Figure 47. Image Division. (a) original image, (b) image divided by a value < 1 to brighten, and (c) image divided a value > 1 to darken.

Figure 48. Image Multiplication. (a) original image, (b) x-ray image of a hand, (c) images (a) and (b) multiplied together which superimposes the hand onto the original, (d) the output image (c) after image enhancement, (e) a computer generated image of a hand, (f) Gaussian noise image, (g) the result of multiplying image (d) and image (e), this operation adds texture to a computer generated image, and (h) image enhanced version of the hand image with texture added by multiplication.

logic operations

The logic operations AND, OR, and NOT form a complete set, meaning that any other logic operation (XOR, NOR, and NAND) can be created by a combination of these basic operations. They operate in a bit-wise fashion on pixel data.

Q/ We are performing a logic AND on two images. Two corresponding pixel values are 111_{10} in one image and 88_{10} in the second image. The corresponding bit strings are:

$$111_{10} = 01101111_2 \quad 88 = 01011000_2.$$

$$\begin{array}{ll} & 01101111_2 \\ \text{AND} & \underline{01011000_2} \\ & 01001000_2 \end{array}$$

The logic operations AND and OR are used to combine the information in two images. The useful application for image analysis by this operation is to **perform a masking operation**. AND and OR can be used as a simple method to extract an ROI from an image. For example, a white mask ANDed with an image will allow only the portion of the image coincident with the mask to appear in the output image, with the background turned black; and a black mask OR with an image will allow only the part of the image corresponding to the black mask to appear in the output image, but will turn the rest of the image white. This process is called image masking and Figure 13 illustrates the results of these operations. **Masking operation is a simple method to extract a region of interest (ROI) processing from an image.**

The NOT operation creates a negative of the original image, by inverting each bit within each pixel value, and is shown in Figure 50. For color images, each individual red, green, and blue band is processed separately and then reassembled.

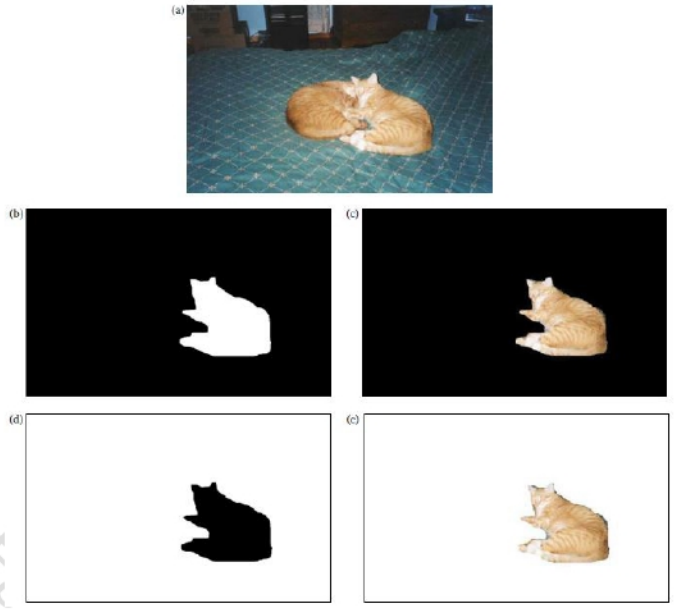

Figure 49. **Image Masking**. (a) Original image of two cats, (b) image mask for AND operation to extract one cat only, (c) Resulting image from (a) AND (b), resulting in one cat on black background, (d) image mask for OR operation, created by performing a NOT on mask (b), and (e) Resulting image from (a) OR (d), resulting in one cat on white background

Figure 50. **Complement Image- NOT Operation**. (a) monochrome image, (b) NOT operator applied, (c) color image, and (d) NOT operator applied

Table 1. Truth table defining the logical operators AND(∩), OR(∪), and NOT(~).

a	b	a AND b	a OR b	NOT(a)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

1.2 Spatial Filters

Spatial filtering is typically applied for noise mitigation or to perform some type of image enhancement. These operators are called spatial filters as they operate on the raw image data in the (r,c) space, the spatial domain. The spatial filters operate on the image data by considering small neighborhoods in an image, such as 3×3, 5×5, 7×7 and so on, and returning a result based on a linear or nonlinear operation, moving sequentially across and down the entire image.

The three types of filters discussed here include:

(1) mean,

(2) median, and

(3) enhancement filters.

The first two are used primarily to deal with noise in images, although they may also be used for special applications. For instance, a mean filter adds a softer look to an image, as in figure 15. The enhancement filters highlight edges and details within the image.

Many spatial filters are implemented with convolution masks. As a convolution mask operation provides a result that is a weighted (جمع وقسمه ع عدد المجاورات وكنه ضرب كل عناصر القناع بقيمه) sum of the values of a pixel and its neighbors, it is called a **linear filter**. One interesting aspect of convolution masks is that the overall effect can be predicted based on their general pattern. For example, if the coefficients of the mask sum to a positive number, the average brightness of the image will be retained. If the sum of the coefficients to zero, the average brightness will be lost and will return a dark image. A negative of the image will result if they sum to a negative number. Furthermore, if the coefficients are both positive and negative, the mask is a filter that will sharpen or enhance details in an image; if the coefficients are all positive, it is a filter that will blur the image.

Figure 51. Mean Filter. (a) Original image, (b) mean filtered image, 3×3 kernel. Note the softer appearance.

The mean filters are essentially averaging filters. They operate on local groups of pixels called neighborhoods and replace the center pixel with an average of the pixels in this neighborhood. This replacement is done with a convolution mask such as the following 3×3 mask:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

The result is normalized by multiplying by 1/9, so overall mask coefficients sum to 1. It is more computationally efficient to perform the integer operations and only multiply by the 1/9 after the image has been processed.

The median filter is a nonlinear filter. A nonlinear filter has a result that cannot be found by a weighted sum of the neighborhood pixels, such as is done with a convolution mask. This filter does operate on a local neighborhood, after the size of the local neighborhood is defined, the center pixel is replaced with the median, or middle, the value present among its neighbors, rather than by their average.

Q/ Given the following 3×3 neighborhood:

$$\begin{bmatrix} 5 & 5 & 6 \\ 3 & 4 & 5 \\ 3 & 4 & 7 \end{bmatrix}$$

We first sort the values in order of size: (3,3,4,4,**5**,5,6,7,5), then we select the middle value, in this case, it is 5. This 5 is then placed in the center location.

A median filter can use a neighborhood of any size, but 3×3, 5×5, and 7×7 are typical. Note that the output image must be written to a separate image (a buffer), so that the results are not corrupted as this process is performed. Figure 16, illustrates the use of a median filter for noise removal.

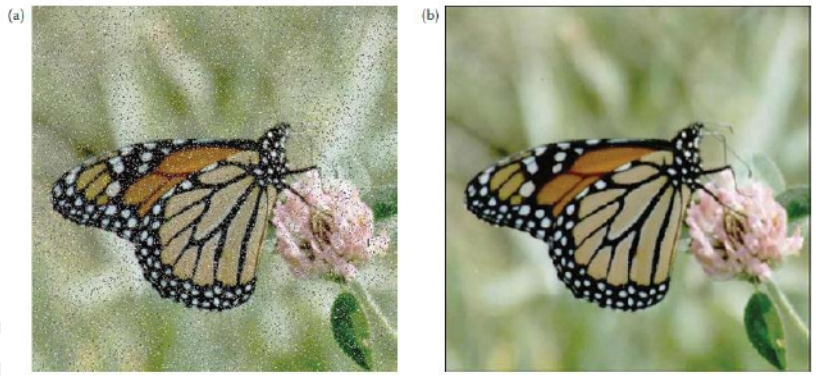

Figure 52. **Median Filter**. (a) Original image with added salt and pepper noise, (b) median filtered image using a 3×3 mask

The enhancement filters are linear, implemented with convolution masks having alternating positive and negative coefficients, so they will enhance image details. Many enhancement filters can be defined, including Laplacian-type and difference filters. Three 3×3 convolution masks for the Laplacian-type filters are:

Filter 1

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Filter 2

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Filter 3

$$\begin{bmatrix} -2 & 1 & -2 \\ 1 & 5 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

Theoretically, Laplacian-type filters are rotationally invariant, or isotropic, which means they tend to enhance details in all directions equally. With these examples, only Filter 2 is truly isotropic; Filter 1 is biased for horizontal and vertical edges, and Filter 3 for diagonal edges.

The difference filters will enhance details in the direction specific to the mask selected. There are four primary difference filter convolution masks, corresponding to edges in the vertical, horizontal, and two diagonal directions:

Vertical	Horizontal	Diagonal 1	Diagonal 2
$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$

Note that these are all simply rotated versions of the first mask. By completing the rotation we have four more difference filter masks:

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The results of applying the Laplacian-type and difference filters are shown in figure 53.

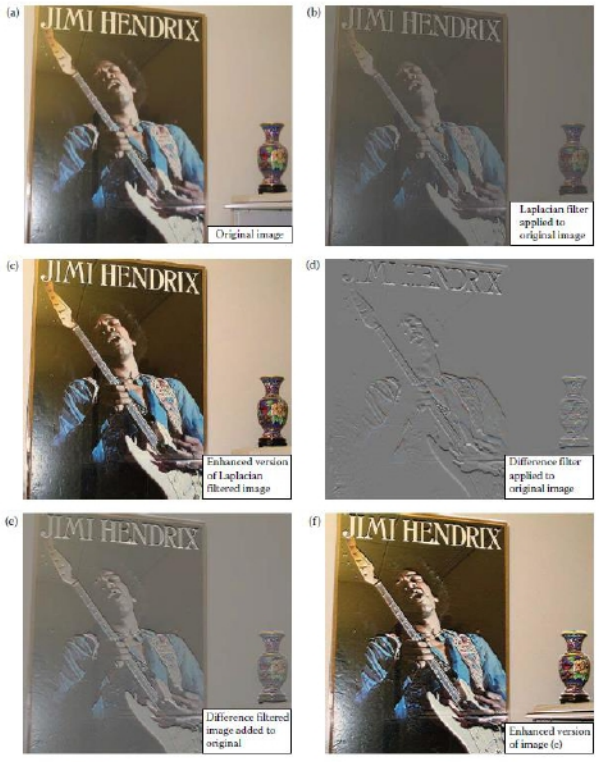

Figure 53. **Enhancement Filters.** (a) Original image, (b) image after Laplacian filter, (c) contrast enhanced version of Laplacian filtered image, compare with (a) and note the improvement in fine detail information, (d) result of a difference (emboss) filter applied to image (a), (e) difference filtered image added to the original, (f) contrast enhanced version of image (e).

1.3 Binary Image Analysis

1.3.1 Basic Image Thresholding

In order to create a binary image from a gray-level image, a threshold operation must be performed. This is done by specifying a threshold value and will set all values above the specified gray level to "1" and everything below or equal to the specified value to "0." Although the actual values for the "0" and "1" can be anything, typically 255 is used for "1" and 0 is used for the "0" value. The "1" value will appear white and the "0" value will appear black.
In many applications, the threshold value will be determined experimentally and is highly dependent on lighting conditions and object to background contrast. It will be much easier to find an acceptable threshold value with proper lighting and good contrast between the object and the background. Figure 54 (a and b) show an example of good lighting and high object to background contrast, whereas Figure 54 (c and d) illustrates a poor example. Imagine trying to identify the object based on the poor example compared to a good example.

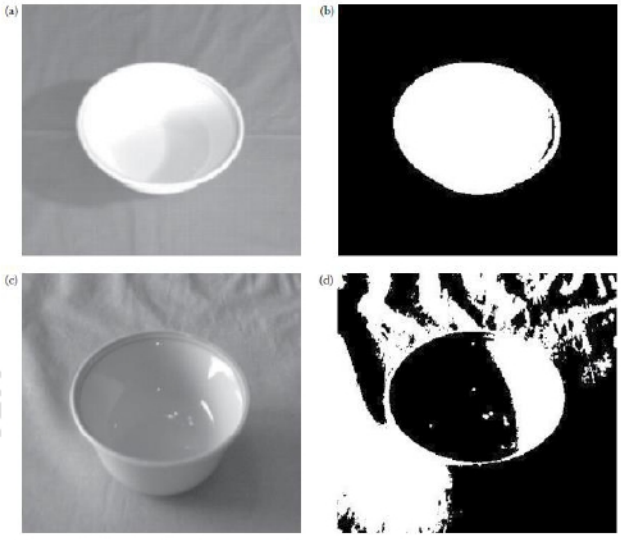

Figure 54. Effects of Lighting and Object to Background Contrast on Thresholding. (a) An image of a bowl with high object to background contrast and good lighting, (b) result of thresholding image (a), (c) an image of a bowl with poor object to background contrast and poor lighting, and (d) result of thresholding image (c).

To select the proper threshold value, the histogram is examined. The histogram of an image is a plot of gray level versus the number of pixels in the image at each gray level. Figure 55 shows the two bowl images and their corresponding histograms. The peaks and valleys in the histogram are examined and a threshold is experimentally selected that will best separate the object from

the background. Notice the peak in Figure 55 b on the far right; this corresponds to the maximum gray-level value and has the highest number of pixels at that value. This peak and the two small peaks to its left represent the bowl. Although many suitable valleys can be seen in the histogram for the poor example (Figure 55 d), none will separate the object from the background successfully, which serves to illustrate the vital importance of proper lighting.

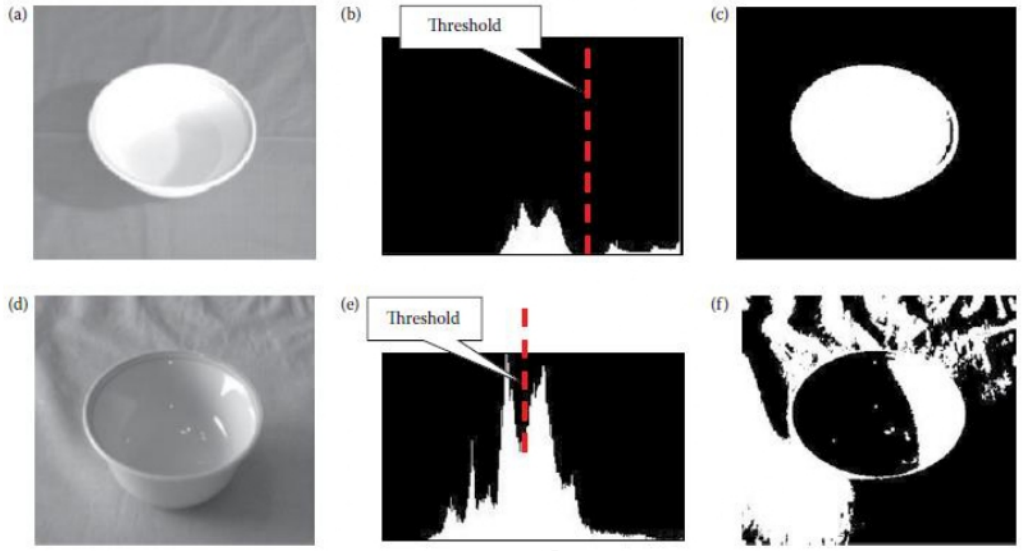

Figure 55. Histograms. (a) An image of a bowl with high object to background contrast and good lighting, (b) the histogram of image (a), showing the threshold that separates object and background, (c) the result after the threshold, (d) an image of a bowl with poor object to background contrast and poor lighting, (e) the histogram of image (d), showing what appears to be a good threshold, but it does not successfully separate object and background, and (f) the result after the threshold.

The basic method of automatically finding a threshold is an iterative process, easily implemented via a computer program. It proceeds as follows:

1. Select an initial value for the threshold, T; typically the average gray-level value for the image.
2. Apply the selected threshold value, T. This will separate the image into two groups of gray levels, those greater than the threshold and those less than or equal to the threshold.
3. Find the average (mean) values for each of these two groups of pixels.

$$\text{Mean for Group 1} = m_1 = \frac{1}{\#\text{pixels} > T} \sum_{I(r,c)>T} I(r,c)$$

$$\text{Mean for Group 2} = m_2 = \frac{1}{\#\text{pixels} \le T} \sum_{I(r,c)\le T} I(r,c)$$

4. Calculate a new threshold by finding the average of the two mean values:

1. $$T_{\text{new}} = (m_1 + m_2)/2$$

5. If the change in threshold value from one iteration to the next is smaller than a previously specified limit, then we are done.

$$|T_{\text{old}} - T_{\text{new}}| < \text{Limit?} \rightarrow \text{Done!}$$

If the change is still greater than the specified limit, go to Step 2 and use T_{new}.

Selection of the value used as a limit in Step 5 will also help determine how long the algorithm will take by limiting the number of iterations, but will also affect the resulting image (see Figure 56).

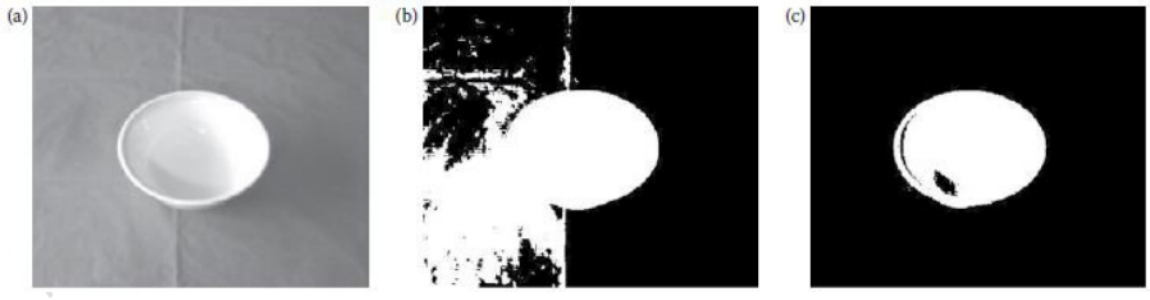

Figure 56. The Limit Parameter with the Automatic Thresholding Algorithm. (a) An image of a bowl with high object to background contrast and good lighting, (b) result of using the automatic thresholding algorithm with a limit = 10, and (c) result of using the automatic thresholding algorithm with a limit = 4. Although using a higher value for the limit will require fewer iterations and is faster, the results may be undesirable.