Convolution and Blurring

We have seen that the convolution operation is a key aspect of edge detection methods. The basic approach we used does not work so well for noisy images, but we will see that the situation can be improved using other convolution kernels. In the process, we will discover how to mathematically describe blurring of an image, which is a necessary step in understanding how image restoration (deblurring) methods work.

1 Problems with Edge Detecting

In the previous notes, the kernels denoted as K_1 and K_2 are called *sobel* kernels. They are particularly useful in detecting, respectively, vertical and horizontal edges in an image. The kernel K_3 is called a *Laplacian*, is direction invariant, and, in general, can detect edges in any direction. However, because it is detecting sharp changes in intensity, it is very sensitive to noise.

Noise can be thought of as small random variations in the pixel values of the original image. These random variations can have different probability distributions; the most common are normal (some times called Gaussian white noise) and Poisson. We can generate a noisy image in MATLAB as follows:

```
I = imread('logo.tif');
In = imnoise(I, 'gaussian');
```

The statement `imnoise(I, 'gaussian')` adds random entries to the pixel values of `I`. The random entries are normally distributed with mean 0 and variance 0.01. We can change these values. For example, if we want to use mean 0 and variance 0.2, then we can use the statement `imnoise(I, 'gaussian', 0, 0.2)`. Increasing the variance like this is equivalent to increasing the level of noise in the image.

Problem 1 *Create a* MATLAB *script m-file with the following statements:*

```
I = imread('logo.tif');
In = imnoise(I, 'gaussian', 0, 0.1);
K3 = [0 -1 0;-1 4 -1;0 -1 0];
O3 = conv2(I, K3, 'same');
level3 = graythresh(O3);
O3 = im2bw(O3,level3);
On3 = conv2(In, K3, 'same');
level3n = graythresh(On3);
On3 = im2bw(On3,level3n);
subplot(2,2,1), imshow(I,[]),  title('Original Image')
subplot(2,2,2), imshow(In,[]), title('Noisy Image')
subplot(2,2,3), imshow(O3,[]), title('Edges of original image')
subplot(2,2,4), imshow(On3,[]), title('Edges of noisy image')
```

Run the script, and observe what happens to the performance of the Laplacian when used on the noisy image. Reduce the variance from 0.1 to 0.01, 0.001, etc., and determine the point at which the edges can be detected accurately, without additional objects appearing in the image.

As we can see from the previous problem, it is difficult to detect edges in noisy images! To get around this, we might first try to remove the noise. However, this can be a difficult problem, since we may not know the source or the precise level of the noise. Another way to get around this is to reduce the importance of the noise by "smoothing" it out. Smoothing can be done by averaging neighboring pixels, which is implemented by convolving the noisy image with certain kernels. For example, one could use any of the kernels:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix},$$

In some sense these kernel are too local for very noisy images, and a larger kernel is often needed. The most common averaging scheme is to use a Gaussian kernel of the form:

$$\exp\left(\frac{-r^2}{2\sigma^2}\right),$$

where $r^2 = x^2+y^2$. This kernel can be constructed in MATLAB using the following statement:

```
K = fspecial('gaussian', [n, n], sigma);
```

where `[n, n]` is the dimension of the desired kernel, and `sigma` is the standard deviation of the Gaussian.

Problem 2 *Try the following statements in* MATLAB*:*

```
K = fspecial('gaussian', [3,3], 0.5)
sum(K(:))
K = fspecial('gaussian', [5,5], 0.5)
sum(K(:))
K = fspecial('gaussian', [32,32], 2);
sum(K(:))
mesh(K)
```

Notice that each time you sum the entries in the kernel you get the same result. Why? In the last case, the kernel is too large to display all of the entries on the screen, so we use `mesh` *to plot it.*

The Gaussian kernel can be used in detecting edges of noisy images by inserting the following statements into your script file:

```
K = fspecial('gaussian', [32, 32], 2);
In = conv2(In, K, 'same');
```

You may have to play around with the value of `sigma` for the various noise levels. We also should note that there are many other approaches that can be used for edge detection, which can be found in books on image processing.

2 Blurring Kernels

The purpose of these notes is not to find a good edge detection method, but to observe what happens when an image is convolved with a Gaussian kernel.

Problem 3 *In* MATLAB, *try the following:*

```
I = imread('logo.tif');
K1 = fspecial('gaussian', [32,32], 1);
K2 = fspecial('gaussian', [32,32], 2);
K3 = fspecial('gaussian', [32,32], 3);
K4 = fspecial('gaussian', [32,32], 4);
O1 = conv2(I, K1, 'same');
O2 = conv2(I, K2, 'same');
O3 = conv2(I, K3, 'same');
O4 = conv2(I, K4, 'same');
figure(1), clf
subplot(2, 2, 1), imshow(O1, [])
subplot(2, 2, 2), imshow(O2, [])
subplot(2, 2, 3), imshow(O3, [])
subplot(2, 2, 4), imshow(O4, [])
figure(2), clf
subplot(2, 2, 1), mesh(K1)
subplot(2, 2, 2), mesh(K2)
subplot(2, 2, 3), mesh(K3)
subplot(2, 2, 4), mesh(K4)
```

How does the value of `sigma` *affect the* `mesh` *plot of the Gaussian kernel? How does it affect the convolved image?*

The point of the previous problem is to notice that by convolving an image with certain kernels, the result is a blurred image. An important problem in image processing is restoring images that are degraded by blurring; that is, we want to undo the convolution operation that caused the blurring. The difficulty is, given a picture, how do we know the convolution kernel that caused the blur?

3 Point Spread Function

Image restoration (sometimes called deblurring or deconvolution) problems arise in many applications. For example, ground based telescopes used by astronomers and the military record images that are blurred when the light travels through the atmosphere. Another example occurs when hardware limitations cause microscopic images to be out of focus. In these cases, computational methods are used to remove the blur. The first problem, though, is to determine the convolution kernel that mathematically describes the particular blur.

In some rare situations a mathematical formula may be known for the convolution kernel, but in most cases it must be determined experimentally using the imaging system. The most common approach is to find how a single point of light is blurred by the imaging system. The single point of light is called a *point source*, and the blurred image of the point source is called the *point spread function.*

Problem 4 *You should convince yourself that the point spread function is the convolution kernel. You can do this with a small example. Suppose the point source image is denoted by I, and the convolution kernel by K; that is,*

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad K = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} \\ k_{21} & k_{22} & k_{23} & k_{24} & k_{25} \\ k_{31} & k_{32} & k_{33} & k_{34} & k_{35} \\ k_{41} & k_{42} & k_{43} & k_{44} & k_{45} \\ k_{51} & k_{52} & k_{53} & k_{54} & k_{55} \end{bmatrix}$$

Using zero boundary conditions, show that the convolution of K and I produces K.

So now, to get the point spread function (i.e., convolution kernel), we need to generate an image of a point source. What constitutes a point source depends on the application. For example, when using a telescope to collect images in space, the point source can be a single bright star. In microscopy, though, the point source is typically a fluorescent microsphere having a diameter which is about half the diffraction limit of the lens. From now on we will assume that a point spread function is given with every blurred image.

The next big question is: Given the point spread function and the blurred image, how do we "undo" the convolution, and hence remove the blur. As will be seen later, this is a very difficult problem, and is the subject of our research.