University of Washington

Applications of Convolution in Image Processing with MATLAB

Author:
Sung Kim

Instructor:
Riley Casper

August 20, 2013

1 Abstract

The article presents a short introduction to image processing and image filtering techniques. The mathematical concepts of convolution and the kernel matrix are used to apply filters to signals, to perform functions such as extracting edges and reducing unwanted noise. The Sobel operator and Gaussian smoothing filter are implemented in MATLAB to achieve the functions previously mentioned, and are evaluated on test images. The effects of adding Gaussian and 'salt and pepper' noise before filtering are then presented as an approximation to signals that occur in real applications. Pre-processing images before applying other filters is shown to produce improved results when extracting edges from images with noise.

2 Introduction

Since the beginning of information theory and computing in the early 20th century, digital signal processing (DSP) has played an important role in fields such as communications, graphic arts, medical imaging, and remote sensing, to name a few. From collision-detection systems in vehicles to computer-aided photo editing, DSP is a cornerstone for much of the technology we use today

Signals can take on many forms, and the physical nature of a signal varies drastically between fields and applications. In the general sense, a signal is simply information that has been collected or quantified in some way, such as an electrical current or temperature measurement. For instance, the value of a company's stock as a function of time is a simple signal. As another example, take a regular bathroom scale; the signal in this case is the person's weight displayed by the scale, as a function of the person's mass. As the reader can imagine, signals are everywhere and can exist in many different forms. However, the underlying theory that applies to signals of varying kinds is largely the same. That is, the mathematical theory of signal processing can be used to model most signals, regardless of the form or measurement. While the mathematics behind DSP theory is entirely general, people are usually only interested in analyzing and processing very specific types of signals, such as those useful for communications or image processing. For this reason, this article will focus primarily on several practical applications of DSP such as image filtering and noise reduction. However, the reader should understand that the mathematics presented can model a much wider variety of signals and systems.

Probably the most important and well known applications for DSP are related to

enhancing or extracting information from signals that have been corrupted in some way. For instance, communication between a pilot in the cockpit and a control tower is inevitably exposed to unwanted noise, from a variety of sources such as other wireless transmitters and electromagnetic noise from the atmosphere. In such cases, DSP techniques can remove noise from the signal and restore parts of the signal that were distorted during transmission. Similarly, DSP techniques can be applied to the general area of image enhancement and restoration. For instance, images of space from satellites or telescopes are often incomplete or degraded, due to the limitations of the imaging hardware and communication equipment. Even in modern imaging systems such as the Hubble Space Telescope, a large fraction of the pixels in any given picture of space are actually interpolated with DSP techniques and never existed in the original image. Another application of DSP techniques is with the extraction or enhancement of certain features within an image signal, such as edges.

As the previous paragraphs mention, images are a type of signal for which many DSP applications exist. The following sections will elaborate on the ideas of DSP when applied to images, known collectively as image processing, and will introduce the concepts of convolution as a means to apply DSP techniques and simplify calculations. In order to understand how image filters use convolution, the idea of a kernel matrix, also known as a mask, will also be explained briefly. Since many DSP techniques are computation heavy and/or applied with computers, the MATLAB programming language, with full code examples, will used to implement and visualize the image processing techniques that are presented.

3 Background

As the earlier section states, image processing is an important subset of DSP that involves analyzing the characteristics of image signals or modifying an image in some way to enhance or remove certain features. Many different approaches exist to achieve these ends; one approach involves the application of what is called a filter. In analog, image processing and filtering is achieved through electrical means. For instance, in CRT displays (e.g., old TVs and computer monitors), image characteristics such as brightness are controlled using varying voltage levels, with a continuous range of values. In contrast, digital image processing and filtering is done on computers, with numerical representations of signals to which mathematical operations can be applied. A key difference between analog and digital image processing is that digital signals are quantized in both length and level, that is, the different values a digital signal can take are a finite, as is the length of the signal. In contrast, analog signals

are continuous with respect to level and length. Since the mathematics of certain image processing ideas is very similar, concepts such as convolution will be introduced in both continuous and discrete form. Digital representations of images and the concept of convolution kernels will also be presented.

3.1 Digital Images

On modern computer systems, images are usually displayed on a screen or monitor with discrete image pixels, which create colors using different ratios of red, green, and blue (RGB). When computing with RGB, each pixel has a parameter to specify the level of each color, but for simplicity this article will only deal with black and white pixels (grayscale). While many different formats exist, UTF-8 (UCS Transformation Format-8-bit) is one of the most common on modern systems and will be used in this article. UTF-8 represents an image by storing the position and intensity of any given pixel in the form of a 2-D array, where each array element has an integer value between 0 and 255 (the range of values representable by an 8-bit number), and maps to a pixel in the target image. In UTF-8, the value 255 is white and 0 is black, with discrete shades of grey in between. While there is much more to say on the topic of image formats and encoding schemes, the depth covered here is sufficient for understanding image filters in grayscale.

3.2 Kernel Matrices

In image processing, many filter operations are applied to an image by performing a special operation called convolution with a matrix called a kernel. Kernels are typically 3x3 square matrices, although kernels of size 2x2, 4x4, and 5x5 are sometimes used. The values stored in the kernel directly relate to the results of applying the filter, and filters are characterized solely by their kernel matrix. For instance, the following kernels are used for detecting the vertical and horizontal edges in an image, and when applied result in the image shown in Figure 1.

$$K_h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad K_v = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

While there are many books are articles devoted to developing different kernels, we will only introduce several types as a means to filter images. These will be presented later, after the reader has developed some sense of how filtering is performed.

Figure 1: Original image and combined results of vertical and horizontal edge-detection kernels

3.3 Convolution

Convolution can be intuitively described as a function that is the integral or summation of two component functions, and that measures the amount of overlap as one function is shifted over the other. An easy way to think of convolution with respect to one variable is to picture a square pulse sliding across the x-axis towards a second square pulse. The convolution at a point is the product of the two functions that occurs when the leading edge of the moving pulse is at that point. When actually taking the convolution of two functions, one function is flipped with respect to the independent variable before shifting, and a change of variables from t to $\tau$is used to facilitate the shifting operation. In one dimension, the mathematical definitions of convolution in discrete and continuous time are indicated by the "$*$" operator:

If f and g are functions in t, then the convolution of f and g over an infinite interval is an integral given by:

$$f * g \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{1}$$

If the convolution is performed over a finite range [0, t], then the convolution is:

$$[f * g](t) \equiv \int_0^t f(\tau)g(t-\tau)d\tau \tag{2}$$

To understand these equations, we can make some simple observations. Notice that because of the change-of-variables, f and g are functions of τ under the integral, but $f*g$ is still a function in t. Since the reflection of function $p(x)$ is given by $p(-x)$, it is clear that $g(t-\tau)$ is a reflection of $g(\tau)$, shifted by an amount t on the τ axis. Hence, as the integral with respect to τ is evaluated, the amount that the function $g(t-\tau)$ changes, rather, the function g "slides" from $-\infty$ to ∞ (or from 0 to t).

If $f[n]$ and $g[n]$ are functions with respect to a single discrete n such as with digital signals, then convolution takes the following form:

$$A[n_a] * B[n_b] = C[n_c] = \sum_{\tau=0}^{n_a-1} A[\tau]B[n_c - \tau] \tag{3}$$

Where $0 \leq n_c < n_a + n_b - 1$

The one-dimensional versions are given here as they are the most simple presentation of the convolution operation. However, since images are intrinsically two-dimensional, the 2-D extension of discrete convolution is required to perform convolution with images. The two-dimensional and one-dimensional versions are very similar in that the two are identical save for an additional set of indices:

$$A[i_a, j_a] * B[i_b, j_b] = C[i_c, j_c] = \sum_{\tau_1=0}^{i_a-1} \sum_{\tau_2=0}^{j_a-1} A[\tau_1, \tau_2]B[i - \tau_1, j - \tau_2] \tag{4}$$

The reader can imagine the two-dimensional case as one matrix "sliding" over the other one unit at a time, with the sum of the element-wise products of the two matrices as the result. Figure 2 shows the convolution of a matrix and a kernel at a single coordinate; the complete convolution is found by repeating the process until the kernel has passed over every possible pixel of the source matrix. In the case where the two matrices are a source image and a filter kernel, the result of convolution is a filtered version of the source image.

It is expected that the concept of convolution and a kernel matrix may not be entirely lucid to the reader. If this is the case, it is recommended that the reader refer

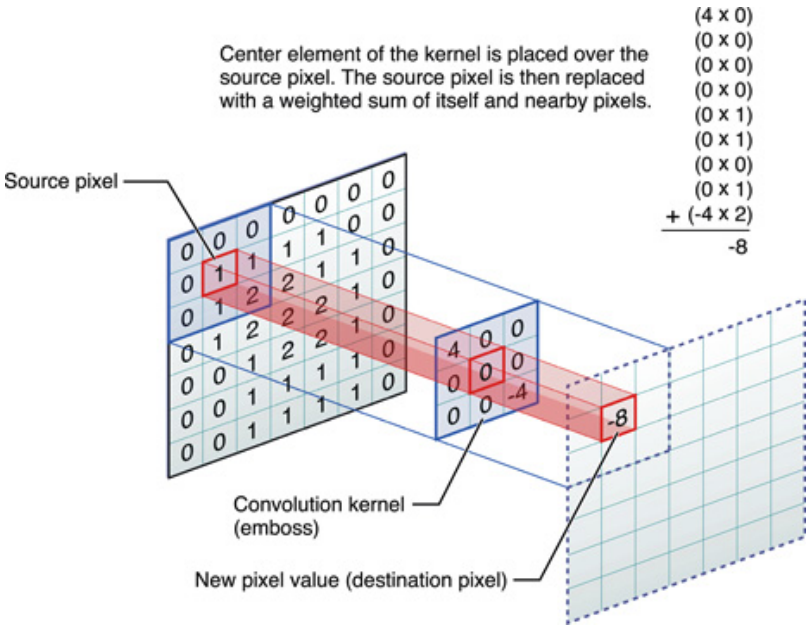


Figure 2: A single location in a 2-D convolution. Source: [7]

to the references or other resources for practice problems and in-depth explanations. Step-by-step video lectures for basic problems can also be found online, and are highly recommended.

4 Image Filters

Now that the reader has an idea of some of the mathematics behind image filters, we will introduce various types of filters and their applications, as well as real implementations using MATLAB. First, filters for image enhancement and edge-extraction will be presented. Gaussian and low-pass filters will then demonstrated as effective ways to reduce noise in signals, and improve the quality of images.

4.1 Sobel-Edge Detectors

Many applications in engineering and science require the correct identification of edges. For instance, many tools for automated manufacturing processes are equipped with cameras to detect markers (such as a thick black line) that might designate special reference points or physical locations. Another application relates to pattern

recognition. In robotics, a common feature is for a robot to behave a certain way in response to visual cues or special markers, such as a driverless car braking at a stop sign. An edge-detector can process and extract relevant features in a set of images before they are fed into a pattern recognition algorithm, which can result in superior performance.

The following are two such kernels for detecting horizontal and vertical edges, together called the Sobel operator. By inspection, we notice that the kernels are vertically and horizontally symmetric, and that the sum of the kernel elements is zero. This means that as the kernel passes over the image, vertically or horizontally aligned pixels that differ in intensity value from their neighbors will be multiplied with the non-zero parts of the kernel, which results in a non-zero pixel in the result. Since the sum of the kernel elements is zero, as soon as the kernel enters a region with uniform pixel values the sum of the element-wise products becomes close to zero. Since we are using UTF-8 which represents black as 0, edges which are non-zero should appear white and non-edges close to black.

$$K_h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad K_v = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{5}$$

As sections 3.3 and 3.2 mention, a filter can be applied to an image by convolving the image with a kernel. Using MATLAB, we will implement the Sobel edge-detector and test it on several images:

```
% We can first read in the source image using the imread()
% function. Since we are only operating on grayscale images,
% we will take the gray transformation using rgb2gray().
% Finally, we want to define our Sobel matrices, and use
% the built in convolution function conv() to convolve the
% source image with the Sobel kernels. For the sake of code
% reuse with multiple images, we will incorporate these
% operations into a function:

function Sobel(im)

% Read in the image and convert to gray
orig = imread(im);
grayscale = rgb2gray(orig);

```

```
% Display the original and gray image
figure(1);
imshow(grayscale);
figure(2);
imshow(orig);

% Define the Sobel kernels
k_v = [-1 0 1; -2 0 2; -1 0 1];
k_h = [1 2 1; 0 0 0; -1 -2 -1];

% Convolve the gray image with Sobel kernels, store result in
    M1 and M2
M1 = conv2(double(grayscale), double(k_v));
M2 = conv2(double(grayscale), double(k_h));

% Display the horizontal edges and vertical edges separately
figure(3);
imshow(abs(M1), []);
figure(4);
imshow(abs(M2), []);

% Display the normalized vertical and horizontal edges
    combined
figure(5);
imshow((M1.^2+M2.^2).^0.5, []);

end
```

Now that our function Sobel() is defined, we can easily apply the edge-detection filter to different images. Figures 3, 4, 5, and 6 show the results of the Sobel filter on several test images.

```
% Call the Sobel() function with the image path as the
    argument.
Sobel('Darkly.jpg');
Sobel('water.jpg');
```

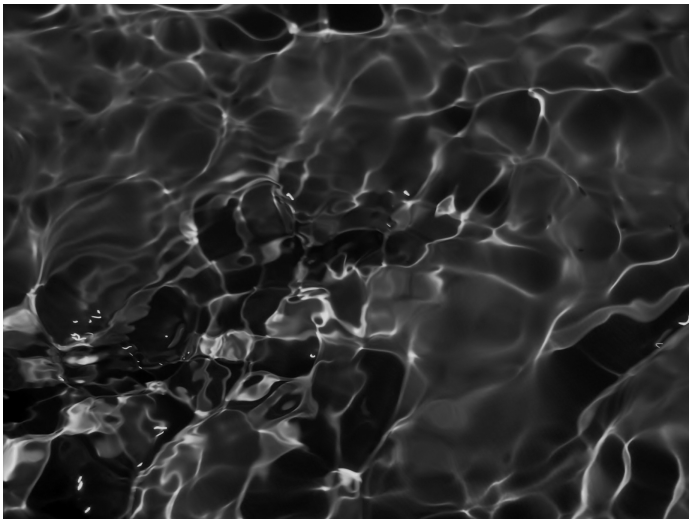


Figure 3: Original image, before convolution. Source: Associated Press

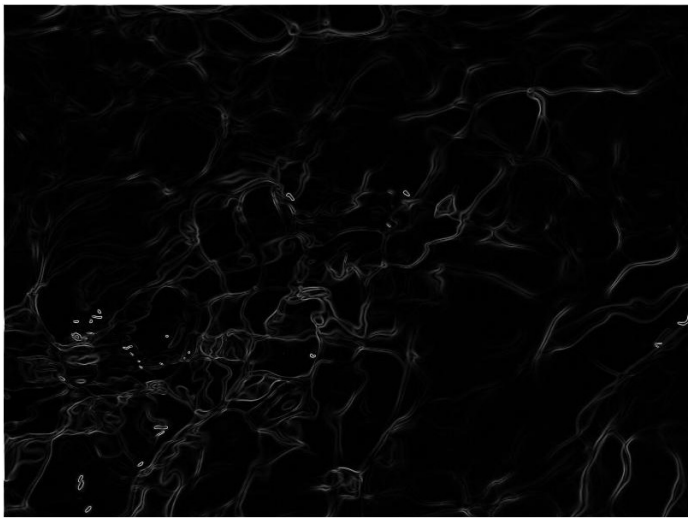


Figure 4: After convolution, horizontal and vertical edges combined

4.2 Noise Reduction and Gaussian Filters

In addition to applications such as feature extraction, filters can be used for denoising signals and images. Many different filters can achieve this purpose and the optimal filter often depends on the particular requirements of the application. One such filter is called a Gaussian, so named because the filter's kernel is a discrete

Figure 5: Original image, before convolution. Source:*A Scanner Darkly (2006).* No copy-write infringment intended.

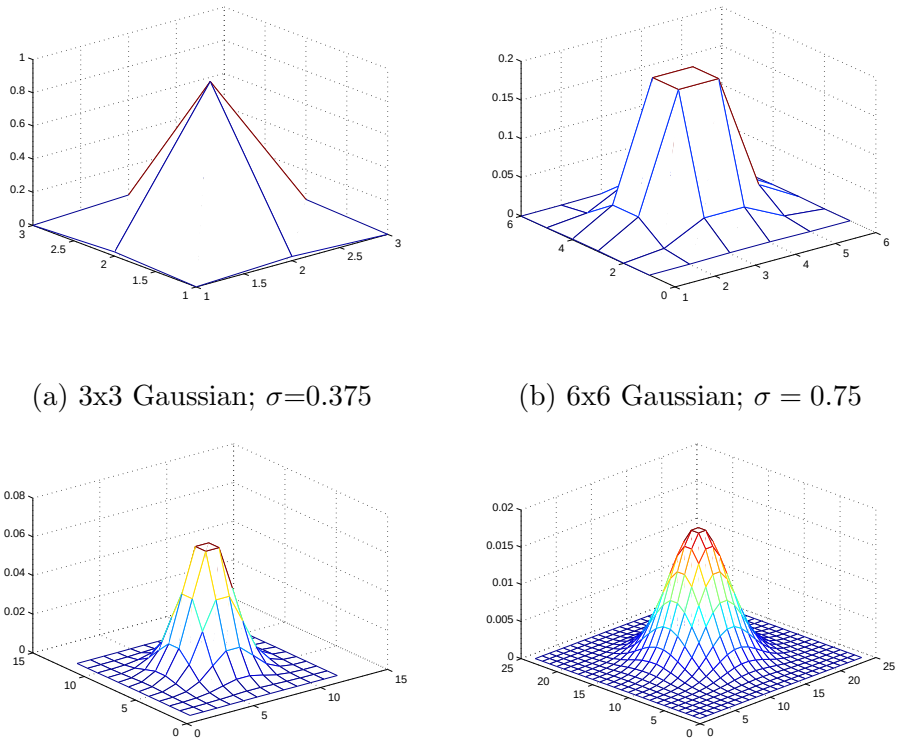
Figure 6: After convolution, horizontal and vertical edges combined

approximation of the Gaussian (normal) distribution. The Gaussian filter is known as a 'smoothing' operator, as its convolution with an image averages the pixels in the image, affectively decreasing the difference in value between neighboring pixels.

$$g[i,j] = e^{-\frac{i^2+j^2}{2\sigma^2}} \tag{6}$$

The σ parameter in equation 6 is equal to the standard deviation of the Gaussian, and can be adjusted according to the desired distribution. It should be noted that as the σ parameter varies the size of the kernel must be adapted, else the kernel might

exclude a sufficient number of elements to contain the 'bell shape' of the distribution. Figure 7 shows the associated kernels for several σ values.



(a) 3x3 Gaussian; σ=0.375

(b) 6x6 Gaussian; $\sigma = 0.75$

(c) 12x12 Gaussian; $\sigma = 1.5$

(d) 24x24 Gaussian; $\sigma = 3.0$

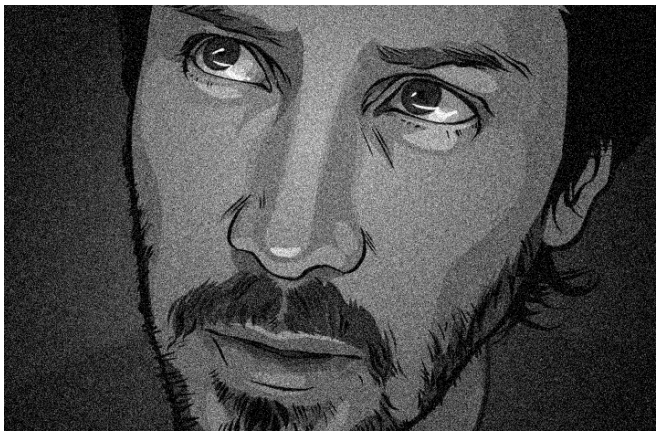
Figure 7: Comparison of various Gaussian kernels

We will now proceed to implement the Gaussian filter in MATLAB with several σ values, and apply the generated kernels on several images. To illustrate the effect of Gaussian filters on images with noise, we will incorporate several types of noise into the test image before applying the filter.

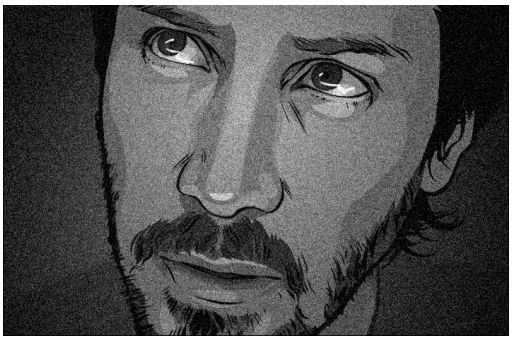
```
% While we could manually define the equation for the Gaussian
    ,
% MATLAB provides the convenient function fspecial(), which
% accepts arguments for the distribution type and associated
% paramaters. We will store the output of the fspecial() in a
% variable, which we can convolve with the source image using
% the conv() function. As before, we will define the filtering
% operation as a function so that we can easily apply the
% filter to multiple images.

% The MATLAB standard library also includes the function
% imnoise(), which we will use to add various types of
% noise to the test image.

function Gaussian(im, size, sigma, noiseType)

% Display the original and gray image
original = imread(im);
grayscale = rgb2gray(original);
figure(1);
imshow(original);
figure(2);
imshow(grayscale);

% Add noise to the grayscale image and display
noisyImage = imnoise(grayscale, noiseType);
figure(4);
imshow(noisyImage);

% Generate Gaussian matrix
h = fspecial('gaussian', size, sigma);

% Convolve the noised image with the Gaussian kernel
M = conv2(double(grayscale), double(h));

% Display the result
figure(3);
imshow((M.^2).^0.5, []);

end
```

As before, we can then apply the filter to test images. We will use the σ values shown in figure 7. While real noise is inherently random, MATLAB and other computational tools can simulate noise mathematically using statistical distributions. We will use 'salt and pepper' noise, which introduces "coarse" black and white distortions uniformly distributed over the signal, and 'gaussian' noise, which is finer than salt and pepper, more densely distributed, and can take on a range of values between black and white.

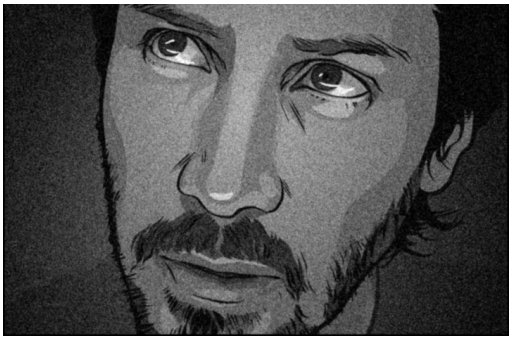
```
% We will call our Gaussian() function first with Gaussian
% noise, then with 'salt and pepper' noise.

% Size 3x3, sigma = 0.375
Gaussian('darkly.jpg', [3 3], 0.375, 'gaussian');

% Size 6x6, sigma = 0.75
Gaussian('darkly.jpg', [6 6], 0.75, 'gaussian');

% Size 12x12, sigma = 1.5
Gaussian('darkly.jpg', [12 12], 1.5, 'gaussian');

% Size 24x24, sigma = 3
Gaussian('darkly.jpg', [24 24], 3, 'gaussian');

% Size 3x3, sigma = 0.375
Gaussian('darkly.jpg', [3 3], 0.375, 'salt & pepper');

% Size 6x6, sigma = 0.75
Gaussian('darkly.jpg', [6 6], 0.75, 'salt & pepper');

% Size 12x12, sigma = 1.5
Gaussian('darkly.jpg', [12 12], 1.5, 'salt & pepper');

% Size 24x24, sigma = 3
Gaussian('darkly.jpg', [24 24], 3, 'salt & pepper');
```



(a) Image with Gaussian noise added



(b) Filter params: Size 3x3, $\sigma = 0.375$

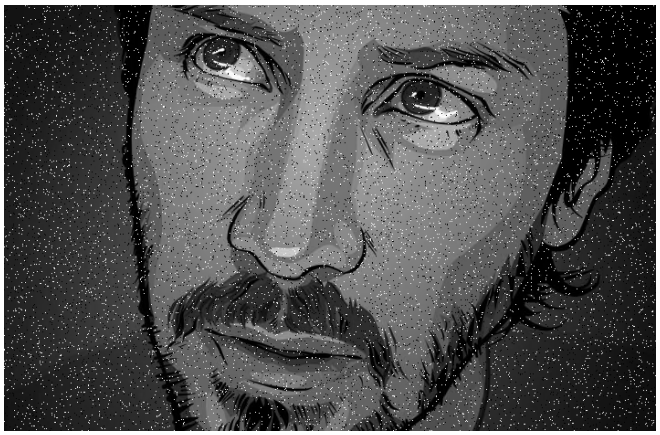


(c) Filter params: Size 6x6, $\sigma = 0.75$

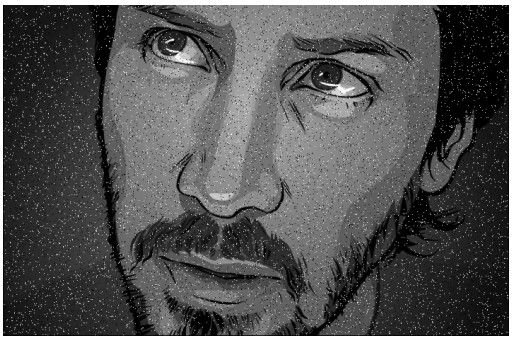
(d) Filter params: Size 12x12, $\sigma = 1.5$

(e) Filter params: Size 24x24, $\sigma = 3.0$

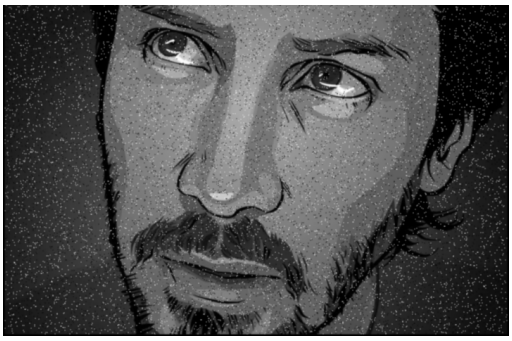
Figure 8: Results of smoothing a noised (Gaussian) image with different Gaussian kernels



(a) Image with 'salt and pepper' noise added



(b) Filter params: Size 3x3, $\sigma = 0.375$



(c) Filter params: Size 6x6, $\sigma = 0.75$

(d) Filter params: Size 12x12, $\sigma = 1.5$

(e) Filter params: Size 24x24, $\sigma = 3.0$

Figure 9: Results of smoothing a noised (salt and pepper) image with different Gaussian kernels

As figures 8 and 9 show, the Gaussian filters perform varying levels of 'smoothing', depending on the parameters used to design the kernel. While the filtering operation itself may or may not improve the subjective quality of the image to the eye, the filtering can drastically improve the results of other filtering operations, such as feature extraction. In section 4.1 we observed how Sobel kernels could extract edges from *noiseless* images. How does the Sobel operator perform when there is noise, as occurs in real life? The following section demonstrates the effects of noise on feature extraction, as well as how pre-processing with other filters can improve results significantly.

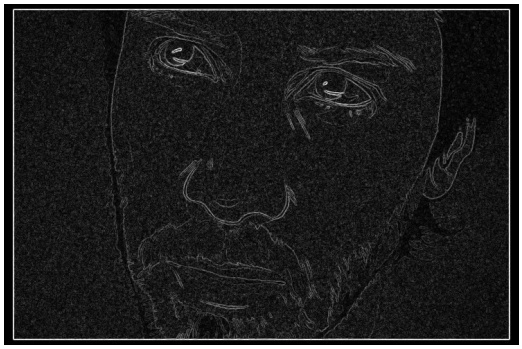
4.3 Extracting Features from Noisy Images

Section 4.1 mentioned several important applications for edge-detection and showed how the Sobel kernels process noiseless images. Figures 10 and 11 show the results of the Sobel operator on the noisy images from 4.1. The affectiveness of the Sobel operator is reduced significantly in the presence of noise, but pre-processing with the Gaussian filter markedly improves the results of the Sobel.

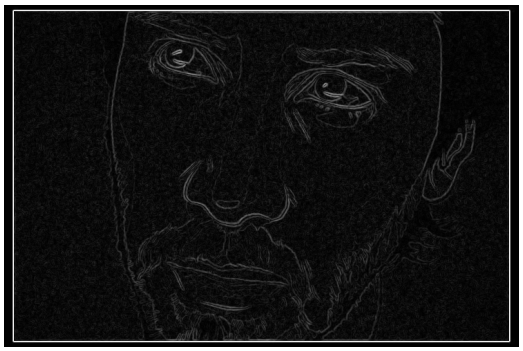
5 Future Study

Convolution and kernel matrices were discussed in this article mostly as a means for filtering images. The reader may find the theory and practice of kernel design interesting, or how convolution is interpreted in N-dimensions. Other mathematical topics related to DSP and signal processing include the Fourier, Laplace, and Z transforms, which can be used to model signal systems, as well as more advanced filtering operations such as non-linear and adaptive filters. In addition, DSP is closely linked with a variety of other fields such as machine learning, statistics, and computer science, all of which are firmly rooted in mathematics. For more information about the implementation of DSP techniques, many textbooks related to hardware optimized for processing signals can be found, as well as free texts online related to all of the topics mentioned above.

(a) Noisy image (Gaussian) after edge-extraction, no Gaussian filter



(b) Filter params: Size 3x3, $\sigma = 0.375$



(c) Filter params: Size 6x6, $\sigma = 0.75$

(d) Filter params: Size 12x12, $\sigma = 1.5$

(e) Filter params: Size 24x24, $\sigma = 3.0$

Figure 10: Results of Sobel edge-detection with and without Gaussian filtering on image with Gaussian noise

(a) Noisy image ('salt and pepper') after edge-extraction, no Gaussian filter

(b) Filter params: Size 3x3, $\sigma = 0.375$

(c) Filter params: Size 6x6, $\sigma = 0.75$

(d) Filter params: Size 12x12, $\sigma = 1.5$

(e) Filter params: Size 24x24, $\sigma = 3.0$

Figure 11: Results of Sobel edge-detection with and without Gaussian filtering on image with salt and pepper noise

6 Conclusion

In a world saturated with technology and information, digital signals are literally everywhere. Much of the technology that exists today would not be possible without a means of extracting information and manipulating digital signals. One such method for processing digital images, called filtering, can be used to reduce unwanted signal information (noise) or extract information such as edges in the image. As sections 3.1 and 4 show, filtering images can be achieved within a mathematical framework using convolution and the kernel matrix. In particular, variations of the Sobel edge-detector and Gaussian smoothing filter were implemented in sections 4.1 and 4.2, using the MATLAB language. Finally, 4.3 demonstrates how multiple filters can be used in conjunction to produce superior results in approximations of real conditions. In a general sense, this article gives just a glimpse of how mathematical techniques can be used in real life, to analyze and extract the information we want from our environment.

References

[1] A. V. Oppenheim, "Linear Time-Invariant Systems", in *Signals and Systems*, 2 Ed. Prentice Hall, 1996. Ch.2.

[2] W. Fred, *Digital Image Filtering.* pp. 1-39.

[3] J. Ramesh and R. Kasturi, "Image Filtering," in *Machine Vision.* McGraw-Hill, 1995. Ch.4.

[4] K. M. M. Rao, "Overview of Image Processing," *Readings in Image Processing*, NRSA, Hyderabad-500 037.

[5] V. Podlozhnyuk, "Image Convolution with CUDA," NVIDIA, 2007.

[6] P. C Rossin. (2011). *Image Filtering & Convolution* [Online], Available: http://www.cse.lehigh.edu/ spletzer/rip_f06/lectures.

[7] Mac Developer Library. "Performing Convolution Operations," [Online], Available: https://developer.apple.com/library/mac/documentation/

[8] E. W. Weisstein. "Convolution." [Online], Available:http://mathworld.wolfram.com/Convolution.html.

[9] R. Fisher and S.Perkins. (2003)."Gaussian Smoothing." [Online], Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm