Chapter 3
Intensity Transformations and Spatial Filtering

Objectives:
We will learn about different transformations, some seen as enhancement techniques.

3.1 Background

3.2 Intensity Transformation Functions

3.2.1 Function imadjust
3.2.2 Logarithmic and Contrast-Stretching Transformations
3.2.3 Some Utility M-Functions for Intensity Transformations

3.3 Histogram Processing and Function Plotting

3.3.1 Generating and Plotting Image Histograms
3.3.2 Histogram Equalization
3.3.3 Histogram Matching (Specification)
Chapter 3
Intensity Transformations and Spatial Filtering

The term spatial domain refers to the image plane itself, and methods in this category are based on direct manipulation of pixels in an image. There are two main important categories of spatial domain processing: 1) intensity (gray level) transformation and spatial filtering. In general the spatial processing is denoted as: $g(x,y) = T[f(x,y)]$, where $f(x,y)$ is the input image, $g(x,y)$ is the output (processed) image, and $T$ is an operator on $f$ defined over a specific neighborhood about point $(x,y)$. 

3.4 Spatial Filtering

3.4.1 Linear Spatial Filtering

3.4.2 Nonlinear Spatial Filtering

3.5 Image Processing Toolbox Standard Spatial Filters

3.5.1 Linear Spatial Filters

3.5.2 Nonlinear Spatial Filters

Summary
The principal approach for defining spatial neighborhoods about a point \((x,y)\) is to use a square or rectangular region centered at \((x,y)\). This is shown on Figure 3.1

The idea is to move the center of this region pixel to pixel starting at one corner and continue with this procedure until the entire image is covered.
Chapter 3
Intensity Transformations Functions

The simplest form of the transform function $T$ is when the neighborhood square is of size $1 \times 1$ (a single pixel). In this case, the value of $g$ at $(x,y)$ depends only on the intensity of $f$ that pixel and $T$ becomes an intensity or gray-level transformation function. Since only intensity values play a role not $(x,y)$, the transformation function might be written as:

$$s = T(r)$$

where $r$ denotes the intensity of $f$ and $s$ the intensity of $g$, both at any corresponding point $(x,y)$ in the image.

Example:
Consider the following matrix that represent the image data for a 4X4 image. Apply a 2X2 average transformation. I.e. a pixel will be represented by the average of 4 pixels around it. Start from top left.

\[
\begin{bmatrix}
2 & 4 & 10 & 4 \\
2 & 4 & 10 & 4 \\
16 & 8 & 4 & 6 \\
14 & 2 & 2 & 8 \\
4 & 0 & 2 & 2 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
24 & 24 & 22 & 24 \\
4 & 4 & 4 & 4 \\
40 & 26 & 26 & 22 \\
4 & 4 & 4 & 4 \\
36 & 24 & 16 & 18 \\
4 & 4 & 4 & 4 \\
22 & 8 & 6 & 14 \\
4 & 4 & 4 & 8 \\
\end{bmatrix}
\]
Chapter 3
Intensity Transformations Functions

The \textit{imadjust} function is the basic IPT tool for intensity transformations of gray scale images.

\[ g = \text{imadjust}(f, [\text{low\_in, high\_in}], [\text{low\_out, high\_out}], \text{gamma}) \]

The above command will map the intensity values in image \( f \) to new values in \( g \) such that values between low\_in and high\_in map to values between low\_out and high\_out.

Notes on \textit{imadjust}:
- Using ([ ]) for low\_in, high\_in, low\_out, and high\_out results in the default values [0 1].
- If high value is lower than low value, then the intensity is reversed.
- The input image can be of class \textit{unit8}, \textit{unit16}, or \textit{double} and the output image has the same class as the input.
- Parameter \textit{gamma} specifies the shape of the curve that maps the intensity values \( f \) to create \( g \).
- If gamma is less than 1, the mapping is weighted toward higher (brighter) output.
This is a bit confusing. Let’s go through an example hoping to clear things up.

```matlab
>> f = uint8([0 100 156 222 255 12])
0 100 156 222 255 12
```

Which means this values are in the `uint8` class, i.e. numbers between 0 to 255.

I will use:

```matlab
>> g = imadjust(f, [0.0; 0.2], [0.5 1], 1)
```

Which means (this is the tricky part):

Map the values between 0 to 0.2*255 i.e. [0 to 51] to values from 0.5*255 = 128 and 255, i.e [128 255],

results:

```
128 255 255 255 255 158
```

Repeat the same example:

```matlab
>> f = uint8([0 100 156 222 255 12])
0 100 156 222 255 12
```

But, this time I will use:

```matlab
>> g = imadjust(f, [0.2; 0.5], [0.5 1], 1)
```

This maps the values between 0.2*255 to 0.5*255: i.e. [51, 128] to values from 0.5*255 = 128 and 255, i.e [128 255],

results:

```
128 209 255 255 255 128
```
Practice:
Suppose:  >> f = uint8([0 100 156 222 255 12])
What would you get for:
>> g = imadjust(f, [0.4; 0.5], [0.5 1],1)

128 128 255 255 255 128

What would you get for:
>> g = imadjust(f, [0.4; 0.5], [1 0.5],1)

255 255 128 128 128 255

Practice:
Suppose:  >> f = uint8([0 100 156 222 255 12])
What would you get for:
>> g = imadjust(f, [0.4; 0.5], [0.5 1],2)

128 128 255 255 255 128

What would you get for:
>> g = imadjust(f, [0.4; 0.5], [1 0.5],2)

255 255 128 128 128 255
Logarithmic and Contrast-Stretching Transformations

Use widely for dynamic range manipulation. Logarithmic transformation is implemented using:

\[ g = c \log(1 + \text{double}(f)) \]

where \( c \) is a constant. What kind of behavior does this model show?
When performing a logarithmic transformation, it is often desirable to bring the resulting compressed values back to the full range of the display.

In MATLAB, for 8 bits, we can use:

```matlab
>> gs = im2unit8(mat2gray(g));
```

Use of `mat2gray` brings the values to the range [0, 1], and `im2unit8` brings them to the range [0, 255].
Example:

\[ \text{img} = \begin{bmatrix} 1 & 23 & 212 & 254 \\ 12 & 78 & 255 & 87 \\ 65 & 212 & 43 & 21 \\ 112 & 254 & 3 & 7 \end{bmatrix} \]

\[ \gg \text{c} = 2; \]
\[ \gg \text{g} = \text{c} \ast \log(1 + \text{double(img)}) \]

\[ \text{g} = \]


\[ \gg \text{g1} = \text{mat2gray(g)} \]

\[ \text{g1} = \]

\[ \begin{array}{cccc} 0 & 0.5121 & 0.9621 & 0.9992 \\ 0.3858 & 0.7577 & 1.0000 & 0.7799 \\ 0.7206 & 0.9621 & 0.6371 & 0.4942 \\ 0.8315 & 0.9992 & 0.1429 & 0.2857 \end{array} \]

\[ \gg \text{gs} = \text{im2uint8(g1)} \]

\[ \text{gs} = \]

\[ \begin{array}{cccc} 0 & 131 & 245 & 255 \\ 98 & 193 & 255 & 199 \\ 184 & 245 & 162 & 126 \\ 212 & 255 & 36 & 73 \end{array} \]
**Contrast-Stretching Transformations**

It compresses the input levels lower than \( m \) into a narrow range of dark levels in the output image, and compresses the values above \( m \) into a narrow band of light levels in the output.

\[
s = T(r)
\]

Limiting case:

**Thresholding**
Contrast-Stretching Transformations

The function corresponding to the diagram on the left is:

\[ s = T(r) = \frac{1}{1 + (m/r)^E} \]

Where \( r \) represents the intensities of the input image, \( s \) denotes the intensity in the output image, and \( E \) controls the slope of the function.

In MATLAB, this can be written as:

\[
g = \frac{1}{1 + (m/(double(f) + eps))^E}
\]

Original:

\[
\begin{bmatrix}
1 & 23 & 212 & 254 \\
12 & 78 & 255 & 87 \\
65 & 212 & 43 & 21 \\
112 & 254 & 3 & 7 \\
\end{bmatrix}
\]

\[
>> g = 1./(1 + (127./(double(img) + eps)).^2)
\]

\[
g =
\begin{bmatrix}
0.0001 & 0.0318 & 0.7359 & 0.8000 \\
0.0088 & 0.2739 & 0.8013 & 0.3194 \\
0.2076 & 0.7359 & 0.1028 & 0.0266 \\
0.4375 & 0.8000 & 0.0006 & 0.0030 \\
\end{bmatrix}
\]
Histogram Processing and Function Plotting

Plots and histograms of the image date can be used in enhancement of images.

A histogram of a digital image with $L$ total possible intensity levels in the range of $[0, G]$ is defined as the discrete function: $h(r_k) = n_k$

Where $r_k$ is the $k$th intensity level in the interval $[0, G]$ and $n_k$ is the number of pixels in the image whose intensity level is $r_k$. The value of $G$ depends on the data class type. For $\text{uni8}$, it is 255, for $\text{unit16}$ it is 65536, and so on.

Note that $G = L - 1$ for images of class $\text{unit8}$ and $\text{unit16}$. 
**Histogram Processing and Function Plotting**

Often, it is useful to work with the *normalized* histograms which is obtained simply by dividing all elements of $h(r_k)$ by the total number of pixels in the image:

$$
p(r_k) = \frac{h(r_k)}{n} = \frac{n_k}{n} \quad \text{For } k = 1, 2, 3, \ldots, L.
$$

From basic probability, we know that $p(r_k)$ is an estimate of the probability of occurrence of intensity level $r_k$.

In MATLAB, the core function to compute image histogram is:

$$h = \text{imhist}(f, b)$$

Where $f$ is the input image, $h$ is its histogram, $h(r_k)$ and $b$ is the number of bins used in formatting the histogram.

The normalized histogram can be obtained using:

$$h = \text{imhist}(f, b)/\text{numel}(f)$$

Where `numel(f)` will give the number of elements in array $f$.

There are several different ways to plot a histogram. The most common one is the bar graph.

```matlab
bar(horz, v, width)
```

Where $v$ is a row vector containing the points to be plotted, $horz$ is a vector of the same dimension as $v$ that contains the increments of the horizontal scale, and $width$ is a number between 0 to 1.
Example:
\[ f = \text{uint8}([2 \ 30 \ 255 \ 40 \ 20 \ 70 \ 80 \ 80 \ 90 \ 200 \ 30 \ 255 \ 255 \ 60 \ 50 \ 70 \ 255 \ 2 \ 3 \ 40]) \]
\[ h = \text{imhist}(f, 20); \]

Produces:  
\[ h = \begin{bmatrix} 3 & 1 & 2 & 2 & 2 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix} \]

What does this mean?

Width = 255 – 2 = 253  
Number of bins = 20

\[ \Delta = \frac{253}{20} = 12.65 \]

Numbers are between:

\[ 2 \ 2+12.65 \ 2+2\times12.65 \ 2+3\times12.65 \ \ldots \ 2+20\times12.65 \]

\[ >> h1 = (1:20) \]
\[ >> \text{bar}(h1, f) \]

What does this mean?

Bins corresponding to \( f \)
Histogram Equalization

Assuming $p_s(r_j)$ where $j = 1, 2, ..., L$ denotes the probability of observing an intensity $j$ in an image with $n$ pixels.

$$p_s(r_j) = \frac{n_j}{n} \text{ same as PDF}$$

In general, the histogram of the processed image will not be uniform. In order to make a create a more uniform distribution, we use a histogram equalization which is based on the CDF.

The Cumulative Distribution Function (CDF) can be written as:

$$S_L = \sum_{j=1}^{L} p_j(r_j) \text{ That is sum of all probabilities}$$

The equalization transformation is defined as:

$$s_k = T(r_k) = \sum_{j=1}^{k} p_s(r_j) = \sum_{j=1}^{k} \frac{n_j}{n}$$

For $k = 1, 2, ..., L$, where $s_k$ is the intensity value in the output (processed) image corresponding to value $r_k$ in the input image.
Example:
\[ f = [2 2 4 5 2 5 4 5 2 3 1 6] \]

<table>
<thead>
<tr>
<th>r</th>
<th>p(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/12</td>
</tr>
<tr>
<td>2</td>
<td>4/12</td>
</tr>
<tr>
<td>3</td>
<td>1/12</td>
</tr>
<tr>
<td>4</td>
<td>2/12</td>
</tr>
<tr>
<td>5</td>
<td>3/12</td>
</tr>
<tr>
<td>6</td>
<td>1/12</td>
</tr>
</tbody>
</table>

Cumulative Probability

<table>
<thead>
<tr>
<th>r_k</th>
<th>s_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/12</td>
</tr>
<tr>
<td>2</td>
<td>5/12</td>
</tr>
<tr>
<td>3</td>
<td>6/12</td>
</tr>
<tr>
<td>4</td>
<td>8/12</td>
</tr>
<tr>
<td>5</td>
<td>11/12</td>
</tr>
<tr>
<td>6</td>
<td>12/12</td>
</tr>
</tbody>
</table>
### Example of Histogram Equalization

Histogram equalization is a technique used to improve the contrast in images. The formula for histogram equalization is given by:

$$\text{Hist} \_ \text{eq} = \text{histeq}(r, 15)$$

#### Table of Histogram Equalization

<table>
<thead>
<tr>
<th>r</th>
<th>prob</th>
<th>cum prob</th>
<th>Hist eq</th>
<th>p dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.125</td>
<td>0</td>
<td>1/15=0.066667</td>
</tr>
<tr>
<td>1</td>
<td>0.125</td>
<td>0.125</td>
<td>0</td>
<td>1/15=0.066667</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>0.125</td>
<td>0</td>
<td>1/15=0.066667</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>0.3125</td>
<td>51</td>
<td>1/15=0.066667</td>
</tr>
<tr>
<td>3</td>
<td>0.1875</td>
<td>0.3125</td>
<td>51</td>
<td>1/15=0.066667</td>
</tr>
<tr>
<td>4</td>
<td>51</td>
<td>0.4375</td>
<td>85</td>
<td>2/15=0.133333</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>0.4375</td>
<td>85</td>
<td>2/15=0.133333</td>
</tr>
<tr>
<td>5</td>
<td>119</td>
<td>0.125</td>
<td>119</td>
<td>2/15=0.133333</td>
</tr>
<tr>
<td>5</td>
<td>0.125</td>
<td>0.5625</td>
<td>119</td>
<td>2/15=0.133333</td>
</tr>
<tr>
<td>6</td>
<td>153</td>
<td>0.625</td>
<td>153</td>
<td>3/15=0.2</td>
</tr>
<tr>
<td>6</td>
<td>0.0625</td>
<td>0.625</td>
<td>153</td>
<td>3/15=0.2</td>
</tr>
<tr>
<td>7</td>
<td>170</td>
<td>0.125</td>
<td>170</td>
<td>3/15=0.2</td>
</tr>
<tr>
<td>7</td>
<td>0.125</td>
<td>0.75</td>
<td>170</td>
<td>3/15=0.2</td>
</tr>
<tr>
<td>8</td>
<td>221</td>
<td>0.125</td>
<td>221</td>
<td>4/15=0.266667</td>
</tr>
<tr>
<td>8</td>
<td>0.1875</td>
<td>0.9375</td>
<td>221</td>
<td>4/15=0.266667</td>
</tr>
<tr>
<td>9</td>
<td>221</td>
<td>1.0</td>
<td>255</td>
<td>5/15=0.333333</td>
</tr>
<tr>
<td>9</td>
<td>0.0625</td>
<td>1.0</td>
<td>255</td>
<td>5/15=0.333333</td>
</tr>
<tr>
<td>10</td>
<td>190</td>
<td>1.0</td>
<td>255</td>
<td>5/15=0.333333</td>
</tr>
<tr>
<td>10</td>
<td>0.125</td>
<td>1.0</td>
<td>255</td>
<td>5/15=0.333333</td>
</tr>
<tr>
<td>11</td>
<td>221</td>
<td>1.0</td>
<td>255</td>
<td>5/15=0.333333</td>
</tr>
<tr>
<td>11</td>
<td>0.1875</td>
<td>1.0</td>
<td>255</td>
<td>5/15=0.333333</td>
</tr>
<tr>
<td>12</td>
<td>255</td>
<td>1.0</td>
<td>255</td>
<td>5/15=0.333333</td>
</tr>
<tr>
<td>12</td>
<td>0.0625</td>
<td>1.0</td>
<td>255</td>
<td>5/15=0.333333</td>
</tr>
</tbody>
</table>

The histogram equalization process involves transforming the pixel values of an image to make the histogram of the output image more uniform. The cumulative probability values are calculated and mapped to the new pixel values.
Histogram Matching

Histogram equalization can achieve enhancement by spreading the levels. However, since the transformation function is based on the histogram of the image it does not change unless the histogram of the image changes. The result is not always successful.

Sometimes, we wish the histogram of the image to look like a given histogram. The method used to make the histogram of processed image looks like a given histogram is called histogram matching or histogram specification.
The input levels have probability density function $p_r(r)$ and the output levels have the specified probability density function $p_z(z)$.

From histogram equalization, we learned that:

Results in intensity levels, $s$, have a uniform probability density functions $p_s(s)$. Suppose now we define a variable $z$ with the property $H(z)$:

We are trying to find an image with intensity levels $z$, which have specified density $p_z(z)$, from these two equations, we have:

$$Z = H^{-1}(s) = H^{-1}[T(r)]$$

We can find $T(r)$ from the input image. Then we need to find the transformed level $z$ whose PDF is the specified $p_z(z)$, as long as we can find $H^{-1}$.

MATLAB: $g = 	ext{histeq}(f, 	ext{hspec})$ where $f$ is input, hspec is the given histogram.
**Spatial Filtering**

The neighborhood processing consists of:

1) Defining a center point, \((x,y)\);
2) Performing an operation that involves only pixels in a predefined neighborhood about that center point;
3) Letting the result of that operation be the “response” of the process at that point;
4) Repeating the process for every point in the image.

The two principal terms used to identify this operation are *neighborhood processing* and *spatial filtering*, with the second term being more prevalent.

If the operation is linear, then it is called *linear spatial filtering* (*spatial convolution*), otherwise, it is called *nonlinear spatial filtering*.

---

**Linear Spatial Filtering (LSF)**

This filtering has its root in the use of the Fourier transform for signal processing in the frequency domain. The idea is to multiply each pixel in the neighborhood by a corresponding coefficient and summing the results to obtain the response at each point \((x,y)\). If the neighborhood is of size \(m\text{-by-}n\), then \(mn\) coefficients are required. These coefficients are arranged as a matrix called a *filter*, *mask*, *filter mask*, *kernel*, *template*, or *window*. The first three terms are most prevalent.

It is not required but is more intuitive to use odd-size masks because they have one unique center point.
There are two closely related concepts in LSF:

1) *correlation*,

2) *convolution*

Correlation is the process of passing the mask $w$ by the image array $f$.

Mechanically, convolution is the same process, except that $w$ is rotated by 180° prior to passing it by $f$.

In both cases we will compute the sum of products of participating values and will place it at the desired position.
**Correlation**

<table>
<thead>
<tr>
<th>Original $f$</th>
<th>$w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 0 0 0 0</td>
<td>1 2 3 2 0</td>
</tr>
</tbody>
</table>

**Step 0**

<table>
<thead>
<tr>
<th>0 0 0 1 0 0 0 0</th>
<th>Zero padding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 2 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 0 0 0 0 0 0 1 0 0 0 0</th>
<th>Zero padding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 2 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0</th>
<th>Zero padding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 2 0</td>
<td></td>
</tr>
</tbody>
</table>

**Correlation-cont.**

<table>
<thead>
<tr>
<th>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</th>
<th>After Shift 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 2 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</th>
<th>After Shift 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 2 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0</th>
<th>Last Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 2 0</td>
<td></td>
</tr>
</tbody>
</table>

`'full' correlation result` 0 0 0 2 3 2 1 0 0 0 0

`'same' correlation result` 0 0 2 3 2 1 0 0 0 0
**Convolution**

Original $f$: 0 0 0 1 0 0 0 0  

$w$: 0 2 3 2 1

Step 0: 0 0 0 1 0 0 0 0 0 2 3 2 1

Zero padding:

$0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 2 1$

**Convolution-cont.**

After Shift 1: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 2 1

After Shift 4: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 2 1

Last Shift: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 2 1

'full' convolution result: 0 0 0 1 2 3 2 0 0 0 0 0 0

'same' convolution result: 0 1 2 3 2 0 0 0 0
A 2-D Correlation Example

original $f(x, y)$

\[
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

$w(x, y) = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

Initial position for $w$

\[
\begin{bmatrix}
1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 5 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\
7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

First Pixel on Original $f$

Initial position for $w$

\[
\begin{bmatrix}
1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 5 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\
7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

'full' correlation result

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

'same' correlation result

\[
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
### A 2-D Convolution Example

**original**

\[
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Original:

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{bmatrix}
\]

\[w(x, y) = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\]

\[w_{\text{rotated}}^{180^\circ} = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}\]

**Initial position for \(w\)**

\[
\begin{bmatrix}
9 & 8 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 5 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

**‘full’ convolution result**

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 0 \\
0 & 0 & 0 & 4 & 5 & 6 & 0 & 0 & 0 \\
0 & 0 & 0 & 7 & 8 & 9 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

**‘same’ convolution result**

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
Correlation Example

Given the original matrix:

\[
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

\[w(x, y) = \begin{bmatrix} 1 & 2 \\ 4 & -1 \end{bmatrix}\]

First, we will replace one in the circle.

\[
\begin{bmatrix}
1 & 2 \\
4 & -1
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

Moving across top 1 row produces:

\[
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]
### Example of Pixel Operations

**One Pixel:**

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\( (1) \times (-1) = -1 \)

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

**Two Pixels:**

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\( 4 \times 1 = 4 \times (-1) = 5 \)

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

**Two Pixels:**

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\( 4 \times (-1) = -4 \)

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
How does MATLAB do these?

\[ g = \text{imfilter}(f, w, \text{filtering
d_mode}, \text{boundary_options}, \text{size_options}) \]

Where \( f \) is the input image, \( w \) is the filter mask, \( g \) is the filtered result, and the other parameters are summarized below.

\begin{verbatim}
>> f = [0 0 0 1 0 0 0 0];
>> w = [1 2 3 2 0]
>> g = imfilter(f, w, 'corr', 0, 'full')
g = 0 0 0 0 2 3 2 1 0 0 0 0

>> g = imfilter(f, w, 'corr', 0, 'same')
g = 0 0 2 3 2 1 0 0
\end{verbatim}

Write the convolution version.
Nonlinear Spatial Filtering

Nonlinear spatial filtering is based on neighborhood operations also by the mechanics of defining \( m \)-by-\( n \) neighborhoods by sliding the center points through an image. However, unlike linear spatial filtering that is based on computing the sum of products; nonlinear spatial filtering is based on nonlinear operations involving the pixels of a neighborhood.

For example: letting the response at each center point be equal to the maximum pixel value in its neighborhood.

Also, the concept of a mask is not as prevalent in nonlinear processing.

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Filtering Mode</strong></td>
<td>Filtering is done using correlation (see Figs. 3.13 and 3.14). This is the default.</td>
</tr>
<tr>
<td>'corr'</td>
<td>Filtering is done using convolution (see Figs. 3.13 and 3.14).</td>
</tr>
<tr>
<td><strong>Boundary Options</strong></td>
<td>The boundaries of the input image are extended by padding with a value, ( P ) (written without quotes). This is the default, with value 0.</td>
</tr>
<tr>
<td>'replicate'</td>
<td>The size of the image is extended by replicating the values in its outer border.</td>
</tr>
<tr>
<td>'symmetric'</td>
<td>The size of the image is extended by mirror-reflecting it across its border.</td>
</tr>
<tr>
<td>'circular'</td>
<td>The size of the image is extended by treating the image as one period a 2-D periodic function.</td>
</tr>
<tr>
<td><strong>Size Options</strong></td>
<td>The output is of the same size as the extended (padded) image (see Figs. 3.13 and 3.14).</td>
</tr>
<tr>
<td>'full'</td>
<td>The output is of the same size as the input. This is achieved by limiting the excursions of the center of the filter mask to points contained in the original image (see Figs. 3.13 and 3.14). This is the default.</td>
</tr>
</tbody>
</table>
Nonlinear Spatial Filtering in MATLAB
MATLAB provides two functions for performing general nonlinear filtering: `nlfilter` and `colfilt`.

`nlfilter` performs operations directly in 2-D while `colfilt` organizes the data in the form of columns. `colfilt` requires more memory but runs much faster than `nlfilter`.

\[ g = colfilt(f, [m, n], 'sliding', @fun, parameters) \]

Where \( m \) and \( n \) are the dimensions of the filter region, ‘sliding’ indicates that the process is one of sliding the \( m \)-by-\( n \) mask from pixel to pixel on image \( f \). The \( @fun \) references an arbitrary function with its parameters defined by \( parameters \).

Nonlinear Spatial Filtering in MATLAB
When using `colfilt` the input image must be padded explicitly before filtering. For this we need to define a new function, `padarray`.

\[ fp = padarray(f, [r c], method, direction) \]

Where \( f \) is the input image, \( fp \) is the padded image, \([r c]\) gives the number of additional rows and columns to pad \( f \).

The `method` and `direction` are defined in Table 3.3.
The **method** and **direction**

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method</strong></td>
<td></td>
</tr>
<tr>
<td>'symmetric'</td>
<td>The size of the image is extended by mirror-reflecting it across its border.</td>
</tr>
<tr>
<td>'replicate'</td>
<td>The size of the image is extended by replicating the values in its outer border.</td>
</tr>
<tr>
<td>'circular'</td>
<td>The size of the image is extended by treating the image as one period of a 2-D periodic function.</td>
</tr>
<tr>
<td><strong>Direction</strong></td>
<td></td>
</tr>
<tr>
<td>'pre'</td>
<td>Pad before the first element of each dimension.</td>
</tr>
<tr>
<td>'post'</td>
<td>Pad after the last element of each dimension.</td>
</tr>
<tr>
<td>'both'</td>
<td>Pad before the first element and after the last element of each dimension. This is the default.</td>
</tr>
</tbody>
</table>

Example:

```matlab
>> fp = padarray(f, [3 2], 'replicate', 'post')
fp =

original

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Padded using replication

Example:

```matlab
>> fp = padarray(f, [3 2], 'symmetric', 'post')
```

fp =

original

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Padded using symmetric
Example:
We want to implement a nonlinear filter whose response at any point is the geometric mean of the intensity values of pixels in the neighborhood centered at that point. The geometric mean of a mask of $mn$ size is the product of intensity values to the power of $1/mn$.

First we implement the nonlinear filter function call $gmean$

```matlab
function v = gmean(A)
    mn = size(A, 1); % the length of the columns of A
    v = prod(A, 1).^(1/mn);
end
```

To reduce border effects, we pad the input image

```matlab
>> f = padarray(f, [m n], 'replicate');
>> g = colfilt(f, [m n], 'sliding', @gmean);
```

```matlab
f = [1 2 3 4;
     5 6 7 8;
     1 2 3 4;
     5 6 7 8]
>> f = padarray(f, [3 3], 'replicate','both')
1 1 1 1 2 3 4 4 4 4
1 1 1 1 2 3 4 4 4 4
1 1 1 1 2 3 4 4 4 4
1 1 1 1 2 3 4 4 4 4
5 5 5 5 6 7 8 8 8 8
1 1 1 1 2 3 4 4 4 4
5 5 5 5 6 7 8 8 8 8
5 5 5 5 6 7 8 8 8 8
5 5 5 5 6 7 8 8 8 8
5 5 5 5 6 7 8 8 8 8
```
\[
g = \text{colfilt}(f, [3 \ 3], 'sliding', @gmean)
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1.0000 & 1.0000 & 1.2599 & 1.8171 & 2.8845 & 3.6342 & 4.0000 \\
0 & 1.0000 & 1.0000 & 1.2599 & 1.8171 & 2.8845 & 3.6342 & 4.0000 \\
0 & 1.7100 & 1.7100 & 2.0356 & 2.6974 & 3.8674 & 4.6580 & 5.0397 \\
0 & 1.7100 & 1.7100 & 2.0356 & 2.6974 & 3.8674 & 4.6580 & 5.0397 \\
0 & 2.9240 & 2.9240 & 3.2887 & 4.0041 & 5.1852 & 5.9700 & 6.3496 \\
0 & 2.9240 & 2.9240 & 3.2887 & 4.0041 & 5.1852 & 5.9700 & 6.3496 \\
0 & 5.0000 & 5.0000 & 5.3133 & 5.9439 & 6.9521 & 7.6517 & 8.0000 \\
0 & 5.0000 & 5.0000 & 5.3133 & 5.9439 & 6.9521 & 7.6517 & 8.0000 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

The last two columns

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5.0397 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5.0397 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6.3496 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6.3496 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\text{Original} \\
f = [1 \ 2 \ 3 \ 4; \ 5 \ 6 \ 7 \ 8; \ 1 \ 2 \ 3 \ 4; \ 5 \ 6 \ 7 \ 8]
\]

\[
g(4:7, 4:7) \\
2.0356 \ 2.6974 \ 3.8674 \ 4.6580 \\
2.0356 \ 2.6974 \ 3.8674 \ 4.6580 \\
3.2887 \ 4.0041 \ 5.1852 \ 5.9700 \\
3.2887 \ 4.0041 \ 5.1852 \ 5.9700
\]
**Linear Spatial Filter**

The MATLAB Toolbox supports a number of predefined 2-D linear filters. The `fspecial` generates a filter mask \( w \)

\[ w = \text{fspecial('type', parameters)} \]

Where `type` specifies the filter type, and `parameters` further define the specified filter.

Possible values for `TYPE` are:

- `'average'`  averaging filter
- `'disk'` circular averaging filter
- `'gaussian'` Gaussian lowpass filter
- `'laplacian'` filter approximating the 2-D Laplacian operator
- `'log'` Laplacian of Gaussian filter
- `'motion'` motion filter
- `'prewitt'` Prewitt horizontal edge-emphasizing filter
- `'sobel'` Sobel horizontal edge-emphasizing filter
- `'unsharp'` unsharp contrast enhancement filter

---

**A quick Review– First and Second Order derivative**

Suppose we are working in 1-D dimension – \( x \) axis for now. The first order derivative of \( f \) with respect to \( x \) at a point \( x_i \) in digital form can be defined as:

\[
\frac{df}{dx} = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}
\]

\( f(x_{i-1}) \)  \( f(x_i) \)  \( f(x_{i+1}) \)

\( x_{i-1} \)  \( x_i \)  \( x_{i+1} \)
Second Order derivative

The second order derivative of $f$ with respect to $x$ at a point $x_i$ in digital form can be defined as:

$$\frac{d^2(f)}{dx^2} = \frac{d}{dx} \left( \frac{df}{dx} \right) = \frac{d}{dx} \left( \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \right)$$

$$= f(x_{i+1}) - 2f(x_i) + f(x_{i-1})$$

Second Order derivative with respect to $x$ and $y$

The second order derivative of $f$ with respect to $x$ at a point $x_i$ in digital form can be defined as:

$$\frac{d^2(f(x,y))}{dx^2} = f(x_{i+1}, y_i) - 2f(x_i, y_i) + f(x_{i-1}, y_i)$$

$$\frac{d^2(f(x,y))}{dy^2} = f(x_i, y_{i+1}) - 2f(x_i, y_i) + f(x_i, y_{i-1})$$
Laplacian Filter
The Laplacian filter of an image \( f(x,y) \) defined as:

\[
\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}
\]

Using the digital approximation of the 2\(^{nd}\) derivative:

\[
\begin{align*}
\frac{\partial^2 f}{\partial x^2} &= f(x+1,y) + f(x-1,y) - 2f(x,y) \\
\frac{\partial^2 f}{\partial y^2} &= f(x,y+1) + f(x,y-1) - 2f(x,y)
\end{align*}
\]

\[
\nabla^2 f = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)
\]

Enhancement is done using:

\[
g(x,y) = f(x,y) + c[\nabla^2 f(x,y)]
\]

Where: \( c = 1 \) if the center coefficient of the mask is positive, otherwise it is -1.

Laplacian Filter
The Laplacian filter of an image \( f(x,y) \) defined as:

Laplacian Mask:

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}
\]

Also shown considering The diagonal elements As:

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & -8 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]
What do all these means anyway?

The first mask, considers a center pixel and the four pixels at its top, bottom, left, and right to compute the resulting value. The Laplacian transform process is as follow:

Top + bottom + left + right - 4*center = resulting intensity

On the other hand the second mask also considers the diagonal pixels. It is done as follow:

Top + bottom + left + right + top-left-corner + top-right-corner + bottom-left-corner + bottom-right-corner - 8*center = resulting intensity

Example:
Apply both methods on matrix: \( f = \begin{bmatrix} 2 & 6 \\ 4 & 2 \end{bmatrix} \)
In MATLAB, function `fspecial` generates a filter mask, \( w \), using:
\[
w = \text{fspecial}('\text{type}', \text{parameters})
\]

Where `type` specifies the filter type, and `parameters` further define the specified filter. However the mask is defined as:
\[
\begin{bmatrix}
\alpha & 1-\alpha & \alpha \\
1+\alpha & 1+\alpha & 1+\alpha \\
1-\alpha & -4 & 1-\alpha \\
\alpha & 1+\alpha & 1+\alpha \\
1+\alpha & 1-\alpha & \alpha \\
\end{bmatrix}
\]

Where \( \alpha \) is used for fine tuning. If you replace it with 0, you will get the same Mask as the one we had before.

Table 3.4

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax and Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>'average'</td>
<td><code>fspecial('average', [r c])</code>. A rectangular averaging filter of size ( r \times c ). The default is ( 3 \times 3 ). A single number instead of ( [r \ c] ) specifies a square filter.</td>
</tr>
<tr>
<td>'disk'</td>
<td><code>fspecial('disk', r)</code>. A circular averaging filter (within a square of size ( 2r + 1 )) with radius ( r ). The default radius is 5.</td>
</tr>
<tr>
<td>'gaussian'</td>
<td><code>fspecial('gaussian', [r c], sig)</code>. A Gaussian lowpass filter of size ( r \times c ) and standard deviation ( \text{sig} ) (positive). The defaults are ( 3 \times 3 ) and 0.5. A single number instead of ( [r \ c] ) specifies a square filter.</td>
</tr>
<tr>
<td>'laplacian'</td>
<td><code>fspecial('laplacian', alpha)</code>. A ( 3 \times 3 ) Laplacian filter whose shape is specified by ( \text{alpha} ), a number in the range ([0, 1]). The default value for ( \text{alpha} ) is 0.5.</td>
</tr>
<tr>
<td>'log'</td>
<td><code>fspecial('log', [r c], sig)</code>. Laplacian of a Gaussian (LoG) filter of size ( r \times c ) and standard deviation ( \text{sig} ) (positive). The defaults are ( 5 \times 5 ) and 0.5. A single number instead of ( [r \ c] ) specifies a square filter.</td>
</tr>
<tr>
<td>'motion'</td>
<td><code>fspecial('motion', len, theta)</code>. Outputs a filter that, when convolved with an image, approximates linear motion of a camera with respect to the image) of ( \text{len} ) pixels. The direction of motion is ( \text{theta} ), measured in degrees, counterclockwise from the horizontal. The defaults are ( 9 ) and ( 0 ), which represents a motion of ( 9 ) pixels in the horizontal direction.</td>
</tr>
<tr>
<td>'prewitt'</td>
<td><code>fspecial('prewitt')</code>. Outputs a ( 3 \times 3 ) Prewitt mask, ( w ), that approximates a vertical gradient. A mask for the horizontal gradient is obtained by transposing the result: ( \text{wh} = w' ).</td>
</tr>
<tr>
<td>'sobel'</td>
<td><code>fspecial('sobel')</code>. Outputs a ( 3 \times 3 ) Sobel mask, ( s ), that approximates a vertical gradient. A mask for the horizontal gradient is obtained by transposing the result: ( \text{sh} = \text{s}’ ).</td>
</tr>
<tr>
<td>'unsharp'</td>
<td><code>fspecial('unsharp', alpha)</code>. Outputs a ( 3 \times 3 ) unsharp filter. Parameter ( \text{alpha} ) controls the shape; it must be greater than 0 and less than or equal to 1.0; the default is 0.2.</td>
</tr>
</tbody>
</table>
Nonlinear Spatial Filters

In MATLAB, the function `ordfilt2` is used to generate order-statistic filters (also called rank filters). These contain nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in an image neighborhood and then replacing the value of the center pixel in the neighborhood with the value determined by the ranking result.

The syntax for this type of filter is:

\[ g = \text{ordfilt2}(f, \text{order, domain}) \]