The concept of linear spatial filtering is often used in digital image processing. It is called linear spatial filtering. The term spatial convolution also used, otherwise, the term spatial filtering is used. The convolution here is a linear operation performed on the pixels of the neighborhood of the point. The convolution is defined as:

\[ (f * h)(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m, n) h(x-m, y-n) \]

where \( f \) and \( h \) are the input and kernel functions, respectively. The result is a new function \( g \) defined by:

\[ g(x, y) = (f * h)(x, y) \]

The convolution is performed for all positions \( (x, y) \) in the image. The kernel \( h \) is a small matrix, typically 3x3 or 5x5, that is moved across the image. The value of the convolution at each point is the sum of the element-wise products of the kernel and the corresponding neighborhood of the input image.

The kernel is often used to perform certain operations such as smoothing, sharpening, or edge detection. For example, a kernel that is a square matrix of all ones performs a simple averaging of the surrounding pixels, which is useful for smoothing noise. A kernel that is a matrix of ones with a single element in the center set to 1 and all other elements set to -1 performs a simple blur or smoothing operation. A kernel that is a square matrix of all ones except for a single element in the center set to -1 and all other elements set to 1 performs a sharpening operation.

The convolution is a powerful tool in image processing and computer vision. It allows us to extract important features from images, such as edges or corners, which can be used for various applications such as object recognition or segmentation.
two concepts are best explained by some simple examples.

The process of correlation involves the convolution of the data with the mask. In essence, the mask is applied to the data in a sliding window fashion, and the correlation score is calculated for each position in the data. The score is then used to generate the correlation map, which is a visual representation of the correlation between the data and the mask.

In this example, the mask is a simple 3x3 grid, and the data is a 2D matrix. The correlation score is calculated by multiplying the elements of the mask with the corresponding elements of the data, and then summing the results. This process is repeated for each position in the data, resulting in a correlation map.

The correlation map is a useful tool for understanding the relationship between the data and the mask, and can be used to identify areas of high correlation.

In summary, the process of correlation is a powerful tool for analyzing data, and can be used in a wide range of applications, from image processing to signal analysis.
some simple examples

due by 1:20 before class.

The final position

A final position

convolution result

convolution result

convolution result

convolution result

convolution result

Convolution

Convolution

Convolution

Convolution

Figure 3.13

would have been different, so the order matters.

is more intuitive because they have a

mask, although it certainly is not the

type of size being 3 × 3 we exclude from our

is that our principal focus is on masks of

-1

(a) 0 1 0 0 0 0 0 0 0

(b) 0 0 0 0 0 0 0 0 0

(c) 0 0 0 0 0 0 0 0 0

(d) 0 1 2 3 2 1 0 0 0

(e) 0 1 2 3 2 1 0 0 0

(f) 0 0 0 0 0 0 0 0 0

(g) 0 0 0 0 0 0 0 0 0

(h) 0 0 0 0 0 0 0 0 0

(i) 0 0 0 0 0 0 0 0 0

(j) 0 0 0 0 0 0 0 0 0

(k) 0 0 0 0 0 0 0 0 0

(l) 0 0 0 0 0 0 0 0 0

(m) 0 0 0 0 0 0 0 0 0

(n) 0 0 0 0 0 0 0 0 0

(o) 0 0 0 0 0 0 0 0 0

(p) 0 0 0 0 0 0 0 0 0

(q) 0 0 0 0 0 0 0 0 0

(r) 0 0 0 0 0 0 0 0 0

(s) 0 0 0 0 0 0 0 0 0

(t) 0 0 0 0 0 0 0 0 0

(u) 0 0 0 0 0 0 0 0 0

(v) 0 0 0 0 0 0 0 0 0

(w) 0 0 0 0 0 0 0 0 0

(x) 0 0 0 0 0 0 0 0 0

(y) 0 0 0 0 0 0 0 0 0

(z) 0 0 0 0 0 0 0 0 0

...
the following syntax:

The octal representation for "padding" is

\[ \texttt{0} \texttt{110011} \texttt{0010} \texttt{0000} \texttt{0000} \texttt{0000} \texttt{0000} \texttt{0000} \]

The most common syntax for "padding" is

```
(1) \texttt{0} \texttt{110011} \texttt{0} \texttt{0} \texttt{0} \texttt{0} \texttt{0} \texttt{0} \texttt{0} \\
(2) \texttt{0} \texttt{111111} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(3) \texttt{0} \texttt{110011} \texttt{0} \texttt{0} \texttt{0} \texttt{0} \texttt{0} \texttt{0} \texttt{0} \\
(4) \texttt{0} \texttt{111111} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
```

The character occurrence frequency in the data is

```

```

For convolution, we simply replace the \texttt{0} with \texttt{1}.

```
(1) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(2) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(3) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(4) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
```

The output is the dot product of the convolution image and the filter image, with the filter image

```
(1) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(2) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(3) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(4) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
```

The output is the convolution of the image with the filter.

```
(1) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(2) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(3) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
(4) \texttt{1} \texttt{1} \texttt{1} \texttt{1} \\
```

The convolution process can be extended easily to images as illustrated in Fig. 3.14.
The most common syntax for `filter` is:

```c
G = filter(f, w, 'replicate');
```

The following syntax:

```c
G = filter(f, w, [1 0 1]);
```

...returns the same result.

The `filter` function performs convolution with a partial filter, as well as the same as per the discussion of Fig. 3.14, we can use the convolution defined in `filter`. From the discussion of Fig. 3.14, these filters do not affect the output in Section 3.1.4, are presented by 180. For more on the convolution, see the discussion of Fig. 3.14, and that the results of convolution can be obtained by 180. A more complete discussion of convolution is given in Section 3.1.4. The `filter` function is used when performing filtering on real-valued input signals. The `filter` function is used when performing filtering on real-valued input signals.
The simple 3x3 filter is a class double image, f, of size 3x3 x 32 pixels. Consider the following.

Example 3.7

Twist +

- The other approach is to preprocess by using the function roto90(4, 2) to rotate the image 90 degrees clockwise. When working with filters that are neither pre-rounded nor symmetrical and have large dimensions, this is the default.

The default

<table>
<thead>
<tr>
<th>Option Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Corner</strong></td>
</tr>
<tr>
<td><strong>Full</strong></td>
</tr>
<tr>
<td><strong>Border</strong></td>
</tr>
<tr>
<td><strong>Symmetric</strong></td>
</tr>
<tr>
<td><strong>Repeat</strong></td>
</tr>
<tr>
<td><strong>Pad</strong></td>
</tr>
<tr>
<td><strong>Repeat</strong></td>
</tr>
</tbody>
</table>

The size of the image is extended by repeating the value in its border (mirror without border). This is called the default, padding with a value of 0. The boundaries of the input image are extended by padding with a value of 0.

The default is to perform convolution (see Figures 3.13 and 3.14). This is the default.

The output is of the same size as the input. This is the default.

The output is of the same size as the input. This is the default.

The output is of the same size as the input. This is the default.

The output is of the same size as the input. This is the default.

The output is of the same size as the input. This is the default.

The output is of the same size as the input. This is the default.

The output is of the same size as the input. This is the default.

The output is of the same size as the input. This is the default.

The output is of the same size as the input. This is the default.
Consider the following operations:

(a) Symmetry with respect to the horizontal axis.
(b) Symmetry with respect to the vertical axis.
(c) Symmetry with respect to the diagonal.
(d) Symmetry with respect to the antidiagonal.

These operations can be performed by convolution with appropriate masks. For example, the result of applying option (a) to an image can be obtained by convolving the image with the following mask:

```
0 0 0
0 1 0
0 0 0
```

The result of applying option (b) can be obtained by convolving the image with the mask:

```
0 0 0
0 0 0
0 1 0
```

The result of applying option (c) can be obtained by convolving the image with the mask:

```
0 0 0
0 0 1
0 0 0
```

The result of applying option (d) can be obtained by convolving the image with the mask:

```
0 0 0
0 0 0
0 1 0
```

These masks are designed to achieve the specified symmetries. For example, option (a) flips the image horizontally, while option (b) flips it vertically. Option (c) flips the image along the diagonal, and option (d) flips it along the antidiagonal.

In practice, these operations can be performed using convolution with appropriate masks. The masks are designed to achieve the desired symmetries, and the convolution operation can be performed using standard signal processing techniques. The resulting images will exhibit the specified symmetries.
Section 3.4.2 Nonlinear Spatial Filtering

Nonlinear spatial filtering is based on neighborhood operations, where the output is determined by the neighborhood of the input pixel. These operations can be applied to images and are useful in various applications such as image processing.

**Figure 3.15(a)** shows the result of the erosion operation, where the output is the smallest value within a specified neighborhood.

**Figure 3.15(b)** shows the result of the dilation operation, where the output is the largest value within a specified neighborhood.

The nonlinear spatial filtering operations are defined as follows:

1. **Erosion**: For a pixel to be included in the output, all pixels in its neighborhood must meet a certain condition. This operation typically reduces the size of objects in an image.
2. **Dilation**: For a pixel to be included in the output, at least one pixel in its neighborhood must meet a certain condition. This operation typically increases the size of objects in an image.

These operations are commonly used in image processing to enhance or modify images according to specific needs.
spatial filtering is based on neighborhood operations also, and the results are defined by a mask of size \( m \times n \) neighborhoods by sliding the center point throughout the image in each direction. The linear and nonlinear spatial filtering operations are defined by the same equation, as discussed in the previous section. However, the nonlinear spatial filtering operation is based on comparing the pixels in the neighborhood, not just on the local region as in linear filtering. The idea is that the center pixel is the most important, and we do not want to blur the center point.

**Nonlinear Spatial Filtering**

Given an input image \( f \) of size \( M \times N \), and a neighborhood of size \( m \times n \), the nonlinear spatial filtering operation is performed by sliding the mask over the entire image, and for each mask, the center point is defined as the position where the central pixel is located. The function \( \text{nlfilter}() \) generates a matrix \( A \) of the same size as \( f \), where each element of \( A \) corresponds to the pixels encompassed by the neighborhood when its center is located at the top left corner of the image. For example, if \( f \) has size \( 10 \times 10 \), then \( A \) will be of size \( 10 \times 10 \), and each element of \( A \) is defined by the function \( g \) applied to the corresponding block of the input image. The function \( g \) is a nonlinear operator that can perform a variety of tasks such as edge detection, noise reduction, or sharpening.

The syntax of function \( \text{nlfilter}() \) is:

```matlab
A = nlfilter(f, [m n], g);
```

where \( f \) is the input image, \( A \) is the output image of the same size, \( m \) and \( n \) are the dimensions of the filter region, and \( g \) is the function that defines the operation. The function \( g \) can be specified as a function handle, or as a character vector or string. The function \( g \) is applied to each block of the input image, and the results are stored in the output image.

The syntax of function \( \text{padarray}() \) is:

```matlab
p = padarray(f, [m n], 'replicate', 'post');
```

where \( p \) is the padded image, \( f \) is the input image, \( m \) and \( n \) are the dimensions of the filter region, and 'replicate' and 'post' specify that the padding should be done by replicating the elements of the input image at the edges.
Table 3.4: Common toolbox functions for spatial filters.

3.5 Image Processing Toolbox: Standard Spatial Filters

The toolbox supports a number of predefined 2-D linear spatial filters.

3.1 Linear Spatial Filters

Additional nonlinear filters are implemented in Section 3.3.

In this section we discuss first order and nonlinear spatial filters.

Some commonly used nonlinear filters can be implemented as

f of the form 

\[ f(x, y) = f(x + a, y + b) \]

for any values of \( a \) and \( b \).

Table 3.3: Common toolbox functions for spatial filters.

<table>
<thead>
<tr>
<th>Description</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pad edges of the image, pad with the last element of each dimension.</td>
<td><code>padarray</code></td>
</tr>
<tr>
<td>Pad before the first element of each dimension.</td>
<td><code>padarray</code> <code>corners</code></td>
</tr>
<tr>
<td>Pad after the last element of each dimension.</td>
<td><code>padarray</code> <code>post</code></td>
</tr>
<tr>
<td>Periodic boundary condition.</td>
<td><code>npadarray</code></td>
</tr>
<tr>
<td>Reflection boundary condition.</td>
<td><code>npadarray</code> <code>reflect</code></td>
</tr>
<tr>
<td>Symmetric boundary condition.</td>
<td><code>npadarray</code> <code>symmetric</code></td>
</tr>
</tbody>
</table>

Finally, we call `coefficient`.
Table 3.4: Including applicable parameters for each filter:

\[ w = f_{\text{special}}(\text{type}, \text{parameters}) \]

Thus, the function \( f_{\text{special}} \) generates a filter mask \( w \) using the parameters specified in the type, which are further defined in the parameters section. The special filters provided by \( f_{\text{special}} \) are summarized in the following sections.

3.5.1 Linear Spatial Filters

Additional nonlinear filters are implemented in Section 3.4. In this section, we discuss both spatial and non-spatial filters supported by \( f_{\text{special}} \).

Image Processing Toolbox - Standard Spatial Filters

The toolbox supports a number of pre-defined 2-D linear special filters.

Some commonly used nonlinear filters can be implemented in terms of nonlinear functions and parameters.

Result and features from the filters are discussed in the original image.
The Laplacian mask:

\[
\frac{\partial^2}{\partial x^2} \nabla + \frac{\partial^2}{\partial y^2} \nabla = \nabla^2 \nabla
\]

Enhancement using the Laplacian is based on the equation:

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

where \(\nabla^2 f\) is the Laplacian of the function \(f\), and \(\nabla^2\) is the two-dimensional Laplacian operator.

**Example 3.4:**

The Laplacian is used in image processing and computer vision for tasks such as edge detection and feature extraction.

**Table 3.4:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laplacian</td>
<td>A discrete difference operator that approximates the second derivative.</td>
</tr>
<tr>
<td>Gradient</td>
<td>Computes the rate of change of a function in a given direction.</td>
</tr>
<tr>
<td>Hessian</td>
<td>Computes the rate of change of the gradient.</td>
</tr>
<tr>
<td>Poisson equation</td>
<td>A partial differential equation used in the analysis of image edges.</td>
</tr>
</tbody>
</table>

**Legend:**

- \(f\) is the input function.
- \(\nabla\) is the gradient operator.
- \(\nabla^2\) is the Laplacian operator.
We now proceed to enhance the image by the Laplacian of the second derivatives of the blurred image. We use the Laplacian of the second derivatives, which allows the summing of enhancement results. However, the precomputed derivative expressions are still needed for the calculation of the second derivatives.

\[
\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \Delta f
\]

Using the use of a spatial and integral filter, the result is

\[
\Delta f = \nabla^2 f = \nabla \cdot \nabla f
\]
Figure 3.16 (c) shows the result. This result looks reasonable.

Next we apply $w$ to the input image, $f$, which is of class uint8:

```
1024, 1024, 8
1024, 1024, 8
```

We could just as easily have specified the shape manually as

```
(0, 1) 1: 4 1: 0
```

Laplacian filter discussed previously. We note that the filter is of class double and that its shape with attribute $0$ is the

```
0.0000 0.0000 0.0000
0.0000 1.0000 0.0000
0.0000 0.0000 0.0000
```

**Figure 3.16**

(c) Image Processing Toolbox

...
The result shown in Fig. 3.16(d) is sharper than the original image.

Each algorithm is more efficient than its predecessor. However, in the case of class double, the result is less effective. This can be expected since we can expect to have a Laplacian image with more pixels that are positive. The result looks reasonable, but

```
if true then
-figure: transform(g9)
-figure: transform(g9)
	transform
	transform = f, replicate, w, replicate, f

if true then
	transform
	transform = g - f, replicate, w, replicate, f

if true then
	transform
	transform = g = f, replicate, f

if true then
	transform
	transform = g = f, replicate, w, replicate, f

if true then
	transform
	transform = g = f, replicate, f

if true then
	transform
	transform = g = f, replicate, w, replicate, f

if true then
	transform
	transform = g = f, replicate, f
```

This difficulty by converting to class double before filtering is:

- some class sections are not filled, so negative values are truncated. We get around this difficulty by converting to class double before filtering. However, in the case of class double, the result is less effective. This can be expected since we can expect to have a Laplacian image with more pixels that are positive. Therefore, we need a more efficient algorithm. The result looks reasonable, but

```
if true then
-figure: transform(g9)
-figure: transform(g9)
	transform
	transform = f, replicate, w, replicate, f

if true then
	transform
	transform = g - f, replicate, w, replicate, f

if true then
	transform
	transform = g = f, replicate, w, replicate, f

if true then
	transform
	transform = g = f, replicate, f

if true then
	transform
	transform = g = f, replicate, w, replicate, f

if true then
	transform
	transform = g = f, replicate, f
```
A common approach to generating nonlinear spatial filters is the use of median filters. The median filter replaces each pixel in an image with the median value of its neighborhood; this helps to reduce noise while preserving edges.

The corresponding column of a median filter can be defined as:

\[
\text{median}(A)_{d,m} = \text{median}(A_{d,m} - A)
\]

where \( A \) is a vector, and \( A_{d,m} \) is a neighborhood of \( A \). The median function, when applied to each element of \( A \), produces a new value that is the median of all possible values that could be obtained by displacing \( A \) within its neighborhood.

This process can be implemented recursively, as shown in Figure 3.17(a), which demonstrates the effectiveness of the median filter in reducing noise while preserving edges.

In the context of image processing, median filters are effective in removing random noise, as shown in Figure 3.17(b). The filter replaces each pixel with the median value of its neighborhood, thereby smoothing the image while preserving sharp edges.

Nonlinear spatial filters, such as median filters, are useful for tasks involving image segmentation and feature detection. They are particularly effective in applications where the preservation of edges and contours is crucial, such as in medical imaging and computer vision.

In summary, the use of median filters in image processing offers a robust method for noise reduction and edge preservation, making them a valuable tool in various applications requiring high-quality image data.
Recall that the median of a set of numbers is such that half the values in the set are less than or equal to it and half are greater than or equal to it.

The corresponding column of \( A \) is denoted by \( \text{median}(A, \text{dim}) \).

For example, if \( \text{dim} = 1 \), each element of \( A \) is the median of the elements along the row dimension of \( A \). When a vector whose elements are the median of a single dimension of \( A \), simply compute the median of the ordered sequence.

\[
\text{median}(1:w:n, \text{ones}(w, 1)) = \text{median}(1:w:n, \text{ones}(w, 1))
\]

The best-known order statistic filter in digital image processing is the

\[
\text{median}(f, \text{ones}(w, 1)) = \text{median}(f, \text{ones}(w, 1))
\]

which is implemented using the syntax

\[
\text{imfilter}(f, \text{ones}(w, 1))
\]

The result is significantly sharper than \( f \).

(a) is a gaussian kernel of size \( 3 \times 3 \).
(b) is \( f \) which is the same as (a).
(c) shows \( g \) and \( f \).
(d) shows \( g \).
(e) shows the original moon image again for easy comparison.
The black border effect is not as pronounced.

The result shown in Fig. 3.18(b) is close to the result in Fig. 3.18(c) except that:

```matlab
>> gauss = medfilt2(f, [5, 5], 'symmetric');
```

can obtain an edge by using the `symmetric` option.

The image (recall that the default pass the border with (0)). This type of effect
speaks strongly to the border, those were connect by the black points surrounding
full explains why a good job of noise reduction. More, however, the black

Considering the level of noise in Fig. 3.18(d), median filtering using the de-

```matlab
>> gm = medfilt2(f, 'gaussian', 0.2);
```

Figure 3.18(e) is the result of median filtering this noisy image, with the

function `medfilt2` which is discussed in detail in Section 3.2.

This image was generated using
points having a probability of occurrence of 0.2. This image was the black and white
image corrupted by salt-and-pepper noise in which both the black and white
input during automatic inspection of the board. Figure 3.18(d) is the same
image in Fig. 3.18(a) is an X-ray image of an industrial circuit board,

median filtering.

It will be discussed at this point to illustrate why the implementation of
median filtering is a useful tool for reducing salt-and-pepper noise in an

Median filtering is a useful tool for reducing salt-and-pepper noise in an

of the input with the

```
>> median = medfilt2(f, 3, 'symmetric');
```

This form of this function is

```
>> median = medfilt2(f, [3, 3], 'symmetric');
```

Because of its practical importance the toolbox provides a specialized im-

premedial of the 2-D median filter:
Summary

...