Image Filtering
Readings: Ch 5: 5.4, 5.5, 5.6, 5.7.3, 5.8
(This lecture does not follow the book.)

Images by Pawan Sinha

- formal terminology
- filtering with masks
  - mean filter
  - Gaussian filter
  - general cross-correlation
  - convolution
- median filter
What is an image?

We can think of an **image** as a function, \( f \), from \( \mathbb{R}^2 \) to \( \mathbb{R} \):

- \( f(x, y) \) gives the **intensity** at position \((x, y)\).
- It is a **continuous** function, usually over a rectangle.

A color image is just three functions pasted together. We can write this as a “vector-valued” function:

\[
    f(x, y) = \begin{bmatrix}
        r(x, y) \\
        g(x, y) \\
        b(x, y)
    \end{bmatrix}
\]

![Image processing steps](image.png)

Courtesy of NASA
Images as functions

These are 4 different ways that people use to show gray tones, just for illustration.

The 3D shows the “height” of each pixel.
Digital images

In computer vision we usually operate on digital (discrete) images:

- **Sample** the continuous 2D space on a regular grid.
- **Quantize** each sample by rounding to nearest integer.

The image can now be represented as a matrix of integer values.
Filtering Operations Use Masks

- Masks operate on a neighborhood of pixels.
- A mask of coefficients is centered on a pixel.
- The mask coefficients are multiplied by the pixel values in its neighborhood and the products are summed.
- The result goes into the corresponding pixel position in the output image.

Input Image

```
36 36 36
36 36 45
36 45 45
36 45 54
36 45 54
```

Output Image

```
** ** ** ** **
** ** ** ** **
** ** ** ** **
1/9 1/9 1/9
1/9 1/9 1/9
```

3x3 Mask

```
36 36 45
45 45 45
45 54 45
45 54 54
```
Application: Noise Filtering

Image processing is useful for noise reduction...

Common types of noise:

- **Salt and pepper noise**: contains random occurrences of black and white pixels
- **Impulse noise**: contains random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution
Practical noise reduction

How can we “smooth” away noise in a single image?

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>130</td>
<td>110</td>
<td>120</td>
<td>110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>110</td>
<td>90</td>
<td>100</td>
<td>90</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>130</td>
<td>100</td>
<td>90</td>
<td>130</td>
<td>110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td>100</td>
<td>130</td>
<td>110</td>
<td>120</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>110</td>
<td>80</td>
<td>120</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Mean filtering

\[ F[x, y] \]

\[ G[x, y] \]
Mean filtering

$$F[x, y]$$

Sum the values in a 3x3 nbd. Divide by 9. Replace center.

$$G[x, y]$$

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>60</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>60</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>50</td>
<td>80</td>
<td>80</td>
<td>90</td>
<td>60</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>50</td>
<td>80</td>
<td>80</td>
<td>90</td>
<td>60</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>30</td>
<td>50</td>
<td>50</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Effect of mean filters

- **Gaussian noise**
  - 3x3
  - 5x5
  - 7x7

- **Salt and pepper noise**
  - 3x3
  - 5x5
  - 7x7
Generalization:
Cross-Correlation Filtering

Let’s write this down as an equation. Assume the averaging window is \((2k+1)\times(2k+1)\):

\[
G[i, j] = \frac{1}{(2k + 1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i + u, j + v]
\]

We can generalize this idea by allowing different weights for different neighboring pixels:

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v]
\]

This is called a **cross-correlation** operation and written:

\[
G = H \otimes F
\]

H is called the “filter,” “kernel,” or “mask.”
Mean Kernel

What’s the kernel for a 3x3 mean filter?

In other words, what do you multiply each pixel by when you find the mean?

\[
F[x, y] = \frac{1}{9} \sum_{u=-1}^{1} \sum_{v=-1}^{1} H[u, v] \times F[x+u, y+v]
\]
Mean Kernel

What’s the kernel for a 3x3 mean filter?

In other words, what do you multiply each pixel by when you find the mean?

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 0 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

mean filter \[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
F[x, y]
\]

\[
H[u, v]
\]

1/9
Gaussian Filtering

A Gaussian kernel gives less weight to pixels further from the center of the window.

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 & 0 \\
0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 & 0 \\
0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 & 0 \\
0 & 0 & 90 & 0 & 90 & 90 & 90 & 0 & 0 & 0 \\
0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[\frac{1}{16} \quad H[u, v]\]

This kernel is an approximation of a Gaussian function:

\[h(u, v) = \frac{1}{2\pi\sigma^2}e^{-\frac{u^2 + v^2}{\sigma^2}}\]
Mean vs. Gaussian filtering
Convolution

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v]F[i - u, j - v]
\]

It is written: \( G = H \ast F \)

Suppose H is a Gaussian or mean kernel. How does convolution differ from cross-correlation?

In computer vision, we tend to use symmetric kernels most of the time, and we tend to call them convolution kernels.

In EE, convolution is useful for solving linear systems problems.
**Convolution vs. Correlation**

When do they differ?

**1D Example from Signal Processing**

<table>
<thead>
<tr>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask: ( w = 1 \ 2 \ 3 \ 2 \ 0 )</td>
</tr>
<tr>
<td>Signal: 0 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>Zero padding: 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Apply mask to first position: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( 1 \ 2 \ 3 \ 2 \ 0 )</td>
</tr>
<tr>
<td>Produces a 0 for first out</td>
</tr>
<tr>
<td>Apply mask to 4(^{th}) position: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( 1 \ 2 \ 3 \ 2 \ 0 )</td>
</tr>
<tr>
<td>Produces a 0</td>
</tr>
<tr>
<td>Final result: 0 0 0 2 3 2 1 0 0 0 0</td>
</tr>
<tr>
<td>Remove padding: 0 0 2 3 2 1 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Convolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask: ( w' = 0 \ 2 \ 3 \ 2 \ 1 )</td>
</tr>
<tr>
<td>Signal: 0 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>Zero padding: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Apply mask to first position: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( 0 \ 2 \ 3 \ 2 \ 1 )</td>
</tr>
<tr>
<td>Produces a 0 for first out</td>
</tr>
<tr>
<td>Apply mask to 4(^{th}) position: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( 0 \ 2 \ 3 \ 2 \ 1 )</td>
</tr>
<tr>
<td>Produces a 1</td>
</tr>
<tr>
<td>Final results: 0 0 0 1 2 3 2 0 0 0 0 0</td>
</tr>
<tr>
<td>Remove padding: 0 1 2 3 2 0 0 0 0</td>
</tr>
</tbody>
</table>


**Convolution Examples**

**Simple box blur**

Here's a first and simplest. This convolution kernel has an averaging effect. So you end up with a slight blur. The image convolution kernel is:

\[
\begin{pmatrix}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
\end{pmatrix}
\]

Note that the sum of all elements of this matrix is 1.0. This is important. If the sum is not exactly one, the resultant image will be brighter or darker.

Here's a blur that I got on an image:

![A simple blur done with convolutions](image)

http://www.aishack.in/2010/08/image-convolution-examples/
Convolution Examples

Gaussian blur

Gaussian blur has certain mathematical properties that makes it important for computer vision. And you can approximate it with an image convolution. The image convolution kernel for a Gaussian blur is:

```
0  0  0  5  0  0  0
0  5 18 32 18  5  0
0 18 64 100 64 18  0
 5 32 100 100 32  5
0 18 64 100 64 18  0
0  5 18 32 18  5  0
0  0  0  5  0  0  0
```

Here's a result that I got:
Convolution Examples

**Line detection with image convolutions**

With image convolutions, you can easily detect lines. Here are four convolutions to detect horizontal, vertical and lines at 45 degrees:

- **Horizontal lines**
  
  \[
  \begin{bmatrix}
  -1 & -1 & -1 \\
  2 & 2 & 2 \\
  -1 & -1 & -1 \\
  \end{bmatrix}
  \]

- **Vertical lines**
  
  \[
  \begin{bmatrix}
  -1 & 2 & -1 \\
  -1 & 2 & -1 \\
  -1 & 2 & -1 \\
  \end{bmatrix}
  \]

- **45 degree lines**
  
  \[
  \begin{bmatrix}
  -1 & -1 & 2 \\
  -1 & 2 & -1 \\
  2 & -1 & -1 \\
  \end{bmatrix}
  \]

- **135 degree lines**
  
  \[
  \begin{bmatrix}
  2 & -1 & -1 \\
  -1 & 2 & -1 \\
  -1 & -1 & 2 \\
  \end{bmatrix}
  \]

I looked for horizontal lines on the house image. The result I got for this image convolution was:
Convolution Examples

**Edge detection**

The above kernels are in a way edge detectors. Only thing is that they have separate components for horizontal and vertical lines. A way to "combine" the results is to merge the convolution kernels. The new image convolution kernel looks like this:

<table>
<thead>
<tr>
<th>-1</th>
<th>-1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>8</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Below result I got with edge detection:
Median filters

A Median Filter operates over a window by selecting the median intensity in the window.

What advantage does a median filter have over a mean filter?

Is a median filter a kind of convolution?

No, it’s called a nonlinear filter.
Comparison: salt and pepper noise

Mean  Gaussian  Median

3x3

5x5

7x7
Comparison: Gaussian noise

<table>
<thead>
<tr>
<th>Mean</th>
<th>Gaussian</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="3x3 Mean" /> <img src="image2" alt="Gaussian" /> <img src="image3" alt="Median" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image4" alt="5x5 Mean" /> <img src="image5" alt="Gaussian" /> <img src="image6" alt="Median" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image7" alt="7x7 Mean" /> <img src="image8" alt="Gaussian" /> <img src="image9" alt="Median" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More Comparisons: Mean vs. Median using Matlab

1. Read in the image and display it.

\[
I = \text{imread('eight.tif')};
\text{imshow}(I)
\]

2. Add noise to it.

\[
J = \text{imnoise}(I,'\text{salt \\& pepper}',0.02);
\text{figure, imshow}(J)
\]

3. Filter the noisy image with an averaging filter and display the result.

\[
K = \text{filter2}([\text{fspecial('average')}],J)/255;
\text{figure, imshow}(K)
\]

4. Now use a median filter to filter the noisy image and display the result.

\[
L = \text{medfilt2}(J,[3 3]);
\text{figure, imshow}(L)
\]

Edge Detection

Basic idea: look for a neighborhood with strong signs of change.

Problems:

- neighborhood size
- how to detect change
Differential Operators

Differential operators

- attempt to approximate the gradient at a pixel via masks

- threshold the gradient to select the edge pixels

**What’s a gradient?**

**Def:** the gradient of a scalar function \( f(x_1, x_2, \ldots, x_n) \) is denoted by \( \nabla f \) (del f) and is defined by:

\[
\nabla f = (\partial f/\partial x_1, \partial f/\partial x_2, \ldots, \partial f/\partial x_n)
\]

**What’s a derivative?**
Example: Sobel Operator

\[ S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \]

On a pixel of the image I
- let \( g_x \) be the response to mask \( S_x \)
- let \( g_y \) be the response to mask \( S_y \)

Then the gradient is
\[ \nabla I = [g_x \quad g_y]^T \]

\[ g = (g_x^2 + g_y^2)^{1/2} \] is the gradient magnitude.

\[ \theta = \text{atan2}(g_y, g_x) \] is the gradient direction.
Sobel Operator on Blocks Image

original image  gradient magnitude  thresholded gradient magnitude
Some Well-Known Masks for Computing Gradients

Sobel:

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix}
\quad
\begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\]

Prewitt:

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

Roberts

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