Image filtering
Course announcements

- Office hours for rest of semester:
  - Tiancheng Tuesday 4-6 pm.
  - Yannis Friday 2-4 pm.
  - For this week only, Yannis has extra office hours on Thursday 4-6pm.

- Homework 1 will is available on website and due on September 14th.
  - How many of you have read/started/finished the homework?

- Make sure you are on Piazza (sign up on your own using the link on the course website).
  - How many of you aren’t already on Piazza?

- Everyone still on waitlist will be enrolled.

- Three guest lectures scheduled: Ravi Mullapudi, Suren Jayasuriya, Aswin Sankarayarayanan.
  - Potentially one more to come.

- September 27th lecture will need to be rescheduled for 28th or 29th (sorry).
  - Make sure to take Doodle survey linked on Piazza.
Overview of today’s lecture

• Quick wrap-up of image processing pipeline.
• Types of image transformations.
• Point image processing.
• Linear shift-invariant image filtering.
• Convolution.
• Image gradients.
Most of these slides were adapted directly from:

- Kris Kitani (15-463, Fall 2016).

Inspiration and some examples also came from:

- Fredo Durand (Digital and Computational Photography, MIT).
- Kayvon Fatahalian (15-769, Fall 2016).
Types of image transformations
What is an image?
What is an image?

Each channel is a 2D array of numbers.

color image patch

How many bits are the intensity values?

colorized for visualization

actual intensity values per channel
What is an image?

A (grayscale) image is a 2D function.

What is the range of the image function $f$?

$$f(x)$$

domain $x = \begin{bmatrix} x \\ y \end{bmatrix}$

grayscale image
What types of image transformations can we do?

Filtering

changes pixel values

Warping

changes pixel locations
What types of image transformations can we do?

- **Filtering**
  - \( G(x) = h\{F(x)\} \)
  - Changes the *range* of the image function

- **Warping**
  - \( G(x) = F(h\{x\}) \)
  - Changes the *domain* of the image function
The (in-camera) image processing pipeline

Which of these transformations are filtering and which are warping?
What types of image transformations can we do?

- **Filtering** (lectures 3-6)
  - Changes range of image function
  - Formula: $G(x) = h\{F(x)\}$

- **Warping** (lectures 7-10)
  - Changes domain of image function
  - Formula: $G(x) = F(h\{x\})$
What types of image filtering can we do?

**Point Operation**

**Neighborhood Operation**

point processing

“filtering”
The (in-camera) image processing pipeline

Which of these transformations are (neighborhood) filtering and which are point processing?
Point processing
Examples of point processing

- Original
- Darken
- Lower contrast
- Non-linear lower contrast
- Invert
- Lighten
- Raise contrast
- Non-linear raise contrast
Examples of point processing

- Invert
- Lighten
- Raise contrast
- Non-linear raise contrast

How would you implement these?
Examples of point processing

original  darken  lower contrast  non-linear lower contrast

$x$

$x - 128$

invert  lighten  raise contrast  non-linear raise contrast

How would you implement these?
Examples of point processing

- original
- darken
- lower contrast
- non-linear lower contrast

x

x - 128

x / 2

invert

lighten

raise contrast

non-linear raise contrast

How would you implement these?
Examples of point processing

Original

Darken

Lower contrast

Non-linear lower contrast

\[ x \]

\[ x - 128 \]

\[ \frac{x}{2} \]

\[ \left( \frac{x}{255} \right)^{1/3} \times 255 \]

Invert

Lighten

Raise contrast

Non-linear raise contrast

How would you implement these?
Examples of point processing

- Original
- Darken: $x - 128$
- Lower contrast: $x / 2$
- Non-linear lower contrast: $\left( \frac{x}{255} \right)^{1/3} \times 255$
- Invert: $255 - x$
- Lighten: $x$
- Raise contrast: $x / 2$
- Non-linear raise contrast: $\left( \frac{x}{255} \right)^{1/3} \times 255$
Examples of point processing

How would you implement these?

- original
- darken
- lower contrast
- non-linear lower contrast

- invert
- lighten
- raise contrast
- non-linear raise contrast

- \( x \)
- \( x - 128 \)
- \( \frac{x}{2} \)
- \( \left( \frac{x}{255} \right)^{1/3} \times 255 \)

- \( 255 - x \)
- \( x + 128 \)
Examples of point processing

How would you implement these?

- **Original**
- **Darken**
- **Lower contrast**
- **Non-linear lower contrast**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original</strong></td>
<td>$x$</td>
</tr>
<tr>
<td><strong>Darken</strong></td>
<td>$x - 128$</td>
</tr>
<tr>
<td><strong>Lower contrast</strong></td>
<td>$\frac{x}{2}$</td>
</tr>
<tr>
<td><strong>Non-linear lower contrast</strong></td>
<td>$\left(\frac{x}{255}\right)^{1/3} \times 255$</td>
</tr>
<tr>
<td><strong>Invert</strong></td>
<td>$255 - x$</td>
</tr>
<tr>
<td><strong>Lighten</strong></td>
<td>$x + 128$</td>
</tr>
<tr>
<td><strong>Raise contrast</strong></td>
<td>$x \times 2$</td>
</tr>
</tbody>
</table>
Examples of point processing

- **original**: $x$
- **darken**: $x - 128$
- **lower contrast**: $\frac{x}{2}$
- **non-linear lower contrast**: $\left(\frac{x}{255}\right)^{1/3} \times 255$
- **invert**: $255 - x$
- **lighten**: $x + 128$
- **raise contrast**: $x \times 2$
- **non-linear raise contrast**: $\left(\frac{x}{255}\right)^2 \times 255$

How would you implement these?
A lot more point processing in lecture 12

camera output

image after stylistic tonemapping

[Bae et al., SIGGRAPH 2006]
Linear shift-invariant image filtering
Linear shift-invariant image filtering

• Replace each pixel by a linear combination of its neighbors (and possibly itself).

• The combination is determined by the filter’s kernel.

• The same kernel is shifted to all pixel locations so that all pixels use the same linear combination of their neighbors.
Example: the box filter

• also known as the 2D rect (not rekt) filter
• also known as the square mean filter

\[
g[\cdot, \cdot] = \frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]

• replaces pixel with local average
• has smoothing (blurring) effect
Let’s run the box filter

\[ h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l] \]

note that we assume that the kernel coordinates are centered
Let’s run the box filter

\[ h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l] \]

image \( f[\cdot, \cdot] \)

output \( h[\cdot, \cdot] \)

kernel \( g[\cdot, \cdot] \)
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

output \quad filter \quad image (signal)

shift-invariant: as the pixel shifts, so does the kernel
Let’s run the box filter

\[
\frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

\[
\frac{1}{9} \sum_{k,l} g[k, l] f[m + k, n + l]
\]

image

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

output

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
Let’s run the box filter

$$h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]$$

output \hspace{1cm} filter \hspace{1cm} image (signal)
Let’s run the box filter

\[
h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]
\]

output  \( h[\cdot, \cdot] \)

filter  \( f[\cdot, \cdot] \)

image  \( g[\cdot, \cdot] \)
Let’s run the box filter

\[ h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l] \]
Let’s run the box filter

\[ h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l] \]

output \hspace{1cm} filter \hspace{1cm} image (signal)
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]
Let’s run the box filter

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

output filter image (signal)
Let’s run the box filter

The box filter uses a simple 3x3 kernel:

\[
g[k, l] = \begin{cases} 1 & \text{if } k = l = 0 \\ 0 & \text{otherwise} \end{cases}
\]

The output image \( h[m, n] \) is calculated as:

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

Here, \( f \) is the input image, \( g \) is the kernel, and \( h \) is the output image.

### Image (signal)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

output \quad image (signal)

filter

kernel

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
0 & 0 & 0 & 90 & 0 & 90 & 90 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 90 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Let’s run the box filter

\[
h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]
\]

output \quad filter \quad image (signal)
Let’s run the box filter

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

output \quad \text{filter} \quad \text{image (signal)}
Let’s run the box filter

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

output \quad filter \quad image (signal)
Let’s run the box filter

\[
\begin{array}{c|c|c}
\text{image} & f[\cdot, \cdot] & \text{output} & h[\cdot, \cdot] \\
\hline
0 0 0 0 0 0 0 0 0 & & 0 10 20 30 30 30 20 10 & \\
0 0 0 0 0 0 0 0 0 & & 0 20 40 60 60 60 40 20 & \\
0 0 0 90 90 90 90 0 0 & & 0 & \\
0 0 0 90 90 90 90 0 0 & & & \\
0 0 0 90 90 90 90 0 0 & & & \\
0 0 0 90 90 90 90 0 0 & & & \\
0 0 0 90 90 90 90 0 0 & & & \\
0 0 0 0 0 0 0 0 0 & & & \\
0 0 0 0 0 0 0 0 0 & & & \\
0 0 0 0 0 0 0 0 0 & & & \\
0 0 0 0 0 0 0 0 0 & & & \\
\end{array}
\]

\[h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]\]
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]
Let’s run the box filter

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]
Let’s run the box filter

Let's run the box filter with the following kernel:

\[
g[k, l] = \begin{cases} \frac{1}{9} & \text{if } k = l = 0 \\ 1 & \text{otherwise} \end{cases}
\]

The image is:

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

The output is:

\[
\begin{bmatrix}
0 & 10 & 20 & 30 & 30 & 30 & 20 & 10 \\
0 & 20 & 40 & 60 & 60 & 60 & 40 & 20 \\
0 & 30 & 50 & 80 & 80 & 80 & 60 & 30 \\
0 & 30 & 50 & 80 & 80 & 80 & 60 & 30 \\
0 & 20 & 30 & 50 & 50 & 60 & 40 & 20 \\
0 & 10 & 20 & 30 & 30 & 30 & 20 & 10 \\
10 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
10 & 10 & 10 & 10 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

The formula for the output is:

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]
... and the result is

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

output \quad filter \quad image (signal)
Some more realistic examples
Some more realistic examples
Some more realistic examples
Convolution
Convolution for 1D continuous signals

Definition of filtering as convolution:

\[(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x - y)dy\]
Convolution for 1D continuous signals

Definition of filtering as convolution:

\[(f * g)(x) = \int_{-\infty}^{\infty} f(y) g(x - y) dy\]

Consider the box filter example:

1D continuous box filter

\[f(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & otherwise \end{cases}\]

filtering output is a blurred version of g

\[(f * g)(x) = \int_{-0.5}^{0.5} g(x - y) dy\]
Convolution for 2D discrete signals

Definition of filtering as convolution:

\[(f \ast g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x - i, y - j)\]
Convolution for 2D discrete signals

Definition of filtering as convolution:

\[(f \ast g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x - i, y - j)\]

filtered image \quad \text{filter} \quad \text{input image}

If the filter \(f(i, j)\) is non-zero only within \(-1 \leq i, j \leq 1\), then

\[(f \ast g)(x, y) = \sum_{i,j=-1}^{1} f(i, j)I(x - i, y - j)\]

The kernel we saw earlier is the 3x3 matrix representation of \(f(i, j)\).
Convolution vs correlation

Definition of discrete 2D convolution:

\[
(f \ast g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x - i, y - j)
\]

notice the flip

Definition of discrete 2D correlation:

\[
(f \ast g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x + i, y + j)
\]

notice the lack of a flip

• Most of the time won’t matter, because our kernels will be symmetric.
• Will be important when we discuss frequency-domain filtering (lectures 5-6).
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

Example: box filter

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

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

What is the rank of this filter matrix?
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

Example: box filter

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

= 

\[
\begin{array}{c}
1 \\
1 \\
\end{array}
\]

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

Why is this important?
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

Example: box filter

\[
\begin{pmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{pmatrix} =
\begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix} \ast
\begin{pmatrix}
1 & 1 & 1
\end{pmatrix}
\]

column row

2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).
A 2D filter is separable if it can be written as the product of a “column” and a “row”.

Example: box filter

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
\end{bmatrix} *
\begin{bmatrix}
1 & 1 & 1 \\
\end{bmatrix}
\]

column

row

2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

If the image has \(M \times M\) pixels and the filter kernel has size \(N \times N\):
- What is the cost of convolution with a non-separable filter?
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
= \begin{bmatrix} 1 \\
1 \\
1 \\
\end{bmatrix} \ast \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

example: box filter
column
row

2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

If the image has $M \times M$ pixels and the filter kernel has size $N \times N$:

- What is the cost of convolution with a non-separable filter? $M^2 \times N^2$
- What is the cost of convolution with a separable filter?
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\end{array}
\times
\begin{array}{c}
1 \\
1 \\
1 \\
\end{array}
= 
\begin{array}{c}
1 & 1 & 1 \\
\end{array}
\]

example: box filter

2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

If the image has \( M \times M \) pixels and the filter kernel has size \( N \times N \):
- What is the cost of convolution with a non-separable filter? \( M^2 \times N^2 \)
- What is the cost of convolution with a separable filter? \( 2 \times N \times M^2 \)
A few more filters

Original

3x3 box filter

do you see any problems in this image?
The Gaussian filter

• named (like many other things) after Carl Friedrich Gauss

• kernel values sampled from the 2D Gaussian function:

\[ f(i, j) = \frac{1}{2\pi \sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \]

• weight falls off with distance from center pixel

• theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?
The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss

- kernel values sampled from the 2D Gaussian function:
  \[ f(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \]
  
- weight falls off with distance from center pixel

- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?
- usually at 2-3\( \sigma \)
The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:
  \[ f(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2 + j^2}{2\sigma^2}} \]
- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?
- usually at 2-3\(\sigma\)

Is this a separable filter? Yes!

\[
\begin{array}{ccc}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{array}
\]
Gaussian filtering example
Gaussian vs box filtering

Which blur do you like better?
How would you create a soft shadow effect?
How would you create a soft shadow effect?

Gaussian blur

overlay
Other filters

input

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

output

?
Other filters

input

filter

output

unchanged
Other filters

input

filter
0 0 0
0 1 0
0 0 0

output unchanged

input

filter
0 0 0
0 0 1
0 0 0

output ?
Other filters

input

filter

output

unchanged

shift to left by one
Other filters

<table>
<thead>
<tr>
<th>input</th>
<th>filter</th>
<th>output</th>
</tr>
</thead>
</table>
| ![Image](image_url) | [0 0 0; 0 2 0; 0 0 0] \(- \frac{1}{9}\) | [1 1 1; 1 1 1; 1 1 1] |?
Other filters

- do nothing for flat areas
- stress intensity peaks
Sharpening examples
Sharpening examples
Sharpening examples
Sharpening examples

do you see any problems in this image?
Do not overdo it with sharpening

original

sharpened

oversharpened

What is wrong in this image?
Image gradients
What are image edges?

Very sharp discontinuities in intensity.

domain $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$
Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?
Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?

✓ You take derivatives: derivatives are large at discontinuities.

How do you differentiate a discrete image (or any other discrete signal)?
Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?

✓ You take derivatives: derivatives are large at discontinuities.

How do you differentiate a discrete image (or any other discrete signal)?

✓ You use finite differences.
Finite differences

High-school reminder: definition of a derivative using forward difference

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]
Finite differences

High-school reminder: definition of a derivative using forward difference

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h}$$

For discrete signals: Remove limit and set $h = 2$

$$f'(x) = \frac{f(x + 1) - f(x - 1)}{2}$$

What convolution kernel does this correspond to?
Finite differences

High-school reminder: definition of a derivative using forward difference

\[ f'(x) = \lim_{{h \to 0}} \frac{f(x + h) - f(x)}{h} \]

Alternative: use central difference

\[ f'(x) = \lim_{{h \to 0}} \frac{f(x + 0.5h) - f(x - 0.5h)}{h} \]

For discrete signals: Remove limit and set \( h = 2 \)

\[ f'(x) = \frac{f(x + 1) - f(x - 1)}{2} \]

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Finite differences

High-school reminder: definition of a derivative using forward difference

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

Alternative: use central difference

\[ f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h} \]

For discrete signals: Remove limit and set \( h = 2 \)

\[ f'(x) = \frac{f(x + 1) - f(x - 1)}{2} \]

1D derivative filter

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

What filter is this?

Sobel filter

1D derivative filter
The Sobel filter

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

Sobel filter

\[
\begin{bmatrix}
1 \\
2 \\
1 \\
\end{bmatrix}
\]

Blurring

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

1D derivative filter

In a 2D image, does this filter responses along horizontal or vertical lines?
The Sobel filter

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

\[
\begin{bmatrix}
1 \\
2 \\
1 \\
\end{bmatrix}
\]

Sobel filter \quad Blurring

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

1D derivative filter

Does this filter return large responses on vertical or horizontal lines?
The Sobel filter

Horizontal Sobel filter:

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

\[
\begin{bmatrix}
1 \\
2 \\
1 \\
\end{bmatrix}
\]

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

What does the vertical Sobel filter look like?
The Sobel filter

Horizontal Sobel filter:

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

Vertical Sobel filter:

\[
\begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\begin{bmatrix}
1 \\
0 \\
-1
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 1
\end{bmatrix}
\]
Sobel filter example

original

which Sobel filter?

which Sobel filter?
Sobel filter example

original

horizontal Sobel filter

vertical Sobel filter
Sobel filter example

original

horizontal Sobel filter

vertical Sobel filter
### Several derivative filters

<table>
<thead>
<tr>
<th>Filter</th>
<th>Coefficients</th>
</tr>
</thead>
</table>
| Sobel    | \[
\begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix}
\] |
|          | \[
\begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{bmatrix}
\] |
| Scharr   | \[
\begin{bmatrix}
3 & 0 & -3 \\
10 & 0 & -10 \\
3 & 0 & -3 \\
\end{bmatrix}
\] |
|          | \[
\begin{bmatrix}
3 & 10 & 3 \\
0 & 0 & 0 \\
-3 & -10 & -3 \\
\end{bmatrix}
\] |
| Prewitt  | \[
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{bmatrix}
\] |
|          | \[
\begin{bmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1 \\
\end{bmatrix}
\] |
| Roberts  | \[
\begin{bmatrix}
0 & 1 \\
-1 & 0 \\
1 & 0 \\
\end{bmatrix}
\] |
|          | \[
\begin{bmatrix}
1 & 0 \\
0 & -1 \\
\end{bmatrix}
\] |

- How are the other filters derived and how do they relate to the Sobel filter?
- How would you derive a derivative filter that is larger than 3x3?
Computing image gradients

1. Select your favorite derivative filters.

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

1. Select your favorite derivative filters.

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

2. Convolve with the image to compute derivatives.

\[ \frac{\partial f}{\partial x} = S_x \otimes f \quad \frac{\partial f}{\partial y} = S_y \otimes f \]
Computing image gradients

1. Select your favorite derivative filters.

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

2. Convolve with the image to compute derivatives.

$$\frac{\partial f}{\partial x} = S_x \otimes f \quad \frac{\partial f}{\partial y} = S_y \otimes f$$

3. Form the image gradient, and compute its direction and amplitude.

$$\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

$$\theta = \tan^{-1} \left( \frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \right)$$

$$\|\nabla f\| = \sqrt{\left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2}$$

gradient \quad direction \quad amplitude
Image gradient example

How does the gradient direction relate to these edges?
How do you find the edge of this signal?

intensity plot
How do you find the edge of this signal?

Using a derivative filter:

What’s the problem here?
Differentiation is very sensitive to noise

When using derivative filters, it is critical to blur first!

How much should we blur?
Derivative of Gaussian (DoG) filter

Derivative theorem of convolution: \( \frac{\partial}{\partial x} (h \star f) = (\frac{\partial}{\partial x} h) \star f \)

- How many operations did we save?
- Any other advantages beyond efficiency?
Laplace filter

Basically a second derivative filter.

- We can use finite differences to derive it, as with first derivative filter.

\[
\begin{align*}
\text{first-order finite difference} & \quad f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h} \quad \rightarrow \quad \text{1D derivative filter} \quad 1 \ 0 \ -1 \\
\text{second-order finite difference} & \quad f''(x) = \lim_{h \to 0} \frac{f(x + h) - 2f(x) + f(x - h)}{h^2} \quad \rightarrow \quad \text{Laplace filter} \quad ?
\end{align*}
\]
Laplace filter

Basically a second derivative filter.
• We can use finite differences to derive it, as with first derivative filter.

\[
\text{first-order finite difference} \quad f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h}
\]

\[
\text{second-order finite difference} \quad f''(x) = \lim_{h \to 0} \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}
\]
Laplacian of Gaussian (LoG) filter

As with derivative, we can combine Laplace filtering with Gaussian filtering.

Diagram:
- Input
- Laplacian of Gaussian
- Output
Laplacian of Gaussian (LoG) filter

As with derivative, we can combine Laplace filtering with Gaussian filtering.

input

Laplacian of Gaussian

output

“zero crossings” at edges
Laplace and LoG filtering examples

Laplacian of Gaussian filtering

Laplace filtering
Laplacian of Gaussian vs Derivative of Gaussian

Laplacian of Gaussian filtering  
Derivative of Gaussian filtering
Laplacian of Gaussian vs Derivative of Gaussian

Zero crossings are more accurate at localizing edges (but not very convenient).

Laplacian of Gaussian filtering  Derivative of Gaussian filtering
2D Gaussian filters

\[ h_\sigma(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2 + v^2}{2\sigma^2}} \]

\[ \frac{\partial}{\partial x} h_\sigma(u, v) \]

\[ \nabla^2 h_\sigma(u, v) \]

how does this relate to this lecture’s cover picture?
References

Basic reading:
• Szeliski textbook, Section 3.2