CSE 152: Computer Vision
Hao Su

Lecture 1: Filters and Features
Announcements

• Course website: https://ucsd-cse-152.github.io/

• Linear algebra review
  • TAs will lecture basics of linear algebra in office hours (three times of the same content)
  • Check Piazza for time and location
  • I will give a lecture about more advanced topics next Tuesday

• Friday discussion session: Assignment programming environment
  • Numerical programming in Python/Jupyter lab
Announcements (cont.)

• Homework
  • 4 assignments, 40% (10% per homework)
  • Due: 11:59PM of Tue in the 4, 6, 8, and 10 weeks.
  • Late policy: check the homepage of course website
• Mid-term 20%, 5th week
• Final 40% (TBD)
Filters and Features
Image filtering

- Image filtering:
  - Compute function of local neighborhood at each position

\[ h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l] \]
Image filtering

- Image filtering:
  - Compute function of local neighborhood at each position

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

\[
\begin{bmatrix}
[ ] & [ ] & [ ]
\end{bmatrix}
\]
Example: box filter

\[ f[\cdot, \cdot] \]

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

Slide credit: David Lowe (UBC)
Image filtering

\[ I[\cdot, \cdot] \quad h[\cdot, \cdot] \]

\[ f[\cdot, \cdot] = \frac{1}{9} \]

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

Credit: S. Seitz
Image filtering

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

Credit: S. Seitz
Image filtering

\[ I[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

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

Credit: S. Seitz
Image filtering

\[ I[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

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

Credit: S. Seitz
Image filtering

\[ f[\cdot, \cdot] = \frac{1}{9} \]

\[ I[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

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

Credit: S. Seitz
Image filtering

\( I[\cdot, \cdot] \)

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

\[
h[m, n] = \sum_{k,l} f[k, l] I[m+k, n+l]
\]
Image filtering

\[ I[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

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

Credit: S. Seitz
Image filtering

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

\[ I[\cdot, \cdot] \quad h[\cdot, \cdot] \]

\[ h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l] \]
Box Filter

What does it do?

• Replaces each pixel with an average of its neighborhood

• Achieve smoothing effect (remove sharp features)
What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)
- Why does it sum to one?
Smoothing with box filter
Image filtering

• Image filtering:
  – Compute function of local neighborhood at each position

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

• Really important!
  – Enhance images
    • Denoise, resize, increase contrast, etc.
  – Extract information from images
    • Texture, edges, distinctive points, etc.
  – Detect patterns
    • Template matching
1. Practice with linear filters

Original

Source: D. Lowe
1. Practice with linear filters

Original

Filtered (no change)

Source: D. Lowe
2. Practice with linear filters

Original

Source: D. Lowe
2. Practice with linear filters

Original

Source: D. Lowe
3. Practice with linear filters

Sobel

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

Vertical Edge (absolute value)
3. Practice with linear filters

![Sobel Filter](image)

Sobel

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

Horizontal Edge (absolute value)
4. Practice with linear filters

Original

(Note that filter sums to 1)

Source: D. Lowe
4. Practice with linear filters

**Sharpening filter**
- Accentuates differences with local average

Source: D. Lowe
4. Practice with linear filters

before

after

Source: D. Lowe
Correlation and Convolution

• 2d correlation

\[ h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l] \]
Correlation and Convolution

- 2d correlation

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

Example code
-----------------

```python
import numpy as np
import scipy.signal as ss
import matplotlib.pyplot as plt

input = plt.imread('leo.jpeg')  # input is H*W*3
input = input.mean(axis=-1)    # input now becomes H*W
kernel = np.array([[-1, 0, 1]])
kernel = kernel[::-1, ::-1]    # flip the kernel to make it convolution
out = ss.convolve2d(input, kernel, mode="same")

plt.figure(); plt.imshow(input, cmap="gray")
plt.figure(); plt.imshow(out, cmap="gray")
```

James Hays
Convolution properties

Commutative: $a \ast b = b \ast a$
- Conceptually no difference between filter and signal
- But particular filtering implementations might break this equality, e.g., image edges

Associative: $a \ast (b \ast c) = (a \ast b) \ast c$
- Often apply several filters one after another: $(((a \ast b_1) \ast b_2) \ast b_3)$
- This is equivalent to applying one filter: $a \ast (b_1 \ast b_2 \ast b_3)$

Source: S. Lazebnik
Convolution properties

Commutative: \( a \ast b = b \ast a \)
- Conceptually no difference between filter and signal
- But particular filtering implementations might break this equality, e.g., image edges

Associative: \( a \ast (b \ast c) = (a \ast b) \ast c \)
- Often apply several filters one after another: \(((a \ast b_1) \ast b_2) \ast b_3\)
- This is equivalent to applying one filter: \(a \ast (b_1 \ast b_2 \ast b_3)\)
Convolution properties

• Commutative: \( a \ast b = b \ast a \)
  – Conceptually no difference between filter and signal
  – But particular filtering implementations might break this equality, e.g., image edges

• Associative: \( a \ast (b \ast c) = (a \ast b) \ast c \)
  – Often apply several filters one after another: \(((a * b_1) * b_2) * b_3\)
  – This is equivalent to applying one filter: \(a * (b_1 * b_2 * b_3)\)

• Distributes over addition: \( a \ast (b + c) = (a \ast b) + (a \ast c) \)

• Scalars factor out: \(ka * b = a * kb = k(a * b)\)

• Identity: unit impulse \(e = [0, 0, 1, 0, 0]\), \(a * e = a\)

Source: S. Lazebnik
Important filter: Gaussian

- Weight contributions of neighboring pixels by nearness

\[
G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}
\]

Slide credit: Christopher Rasmussen
Smoothing with Gaussian filter
Smoothing with box filter

James Hays
Why Do We Have Two Eyes?

Depth information is lost in image formation. Binocular (stereo) vision enables depth estimation.
Correspondence estimation

• Motivation: panorama stitching
  – We have two images – how do we combine them?
Correspondence estimation

- Motivation: panorama stitching
  - We have two images – how do we combine them?

Step 1: extract features
Step 2: match features

Slide courtesy: Rick Szeliski
Correspondence estimation

• Motivation: panorama stitching
  – We have two images – how do we combine them?

Step 1: extract features
Step 2: match features
Step 3: align images

Slide courtesy: Rick Szeliski
Image matching

by Diva Sian

by swashford
Harder case

by Diva Sian

by scgbt
Harder still?

NASA Mars Rover images
Answer below (look for tiny colored squares...)

NASA Mars Rover images with SIFT feature matches
Figure by Noah Snavely
Feature descriptors

We know how to detect good points

Next question: **How to match them?**
Feature descriptors

We know how to detect good points
Next question: **How to match them?**

Lots of possibilities (this is a popular research area)
- Simple option: match square windows around the point
- State of the art approach: SIFT

Slide courtesy: Rick Szeliski
Simple matching methods

- SSD (Sum of Squared Differences)
  \[ \sum_{x,y} |W_1(x, y) - W_2(x, y)|^2 \]

- NCC (Normalized Cross Correlation)
  \[ \frac{\sum_{x,y} (W_1(x, y) - \overline{W_1})(W_2(x, y) - \overline{W_2})}{\sigma_{W_1}\sigma_{W_2}} \]

where
  \[ \overline{W_i} = \frac{1}{n} \sum_{x,y} W_i \]
  \[ \sigma_{W_i} = \sqrt{\frac{1}{n} \sum_{x,y} (W_i - \overline{W_i})^2} \]

- What advantages might NCC have over SSD?
Invariance

Suppose we are comparing two images $I_1$ and $I_2$
- $I_2$ may be a transformed version of $I_1$
- What kinds of transformations are we likely to encounter in practice?

We’d like to find the same features regardless of the transformation
- This is called transformational *invariance*
- Most feature methods are designed to be invariant to
  - Translation, 2D rotation, scale
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 30 degrees)
  - Limited illumination or contrast changes
Feature Matching
Desirable property: invariance

Find features that are invariant to transformations

- geometric invariance: translation, rotation, scale
Edges in Natural Images

Source: Photografr.com
What Causes an Edge?

- Depth discontinuity
- Surface orientation discontinuity
- Illumination discontinuity (e.g., shadow)
- Specular reflection of other discontinuity
- Reflectance discontinuity (i.e., change in surface material properties)

Source: Photografr.com
How Can We Find Edges?

• Find regions where magnitude of gradient is large.
Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

Adapted from slide by David Lowe
Properties of SIFT

Extraordinarily robust matching technique

- Can handle changes in viewpoint
  - Up to about 30 degrees out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available

Feature matching

Given a feature in $I_1$, how to find the best match in $I_2$?

1. Define distance function that compares two descriptors

2. Test all the features in $I_2$, find the one with min distance

Slides courtesy: Rick Szeliski
Feature distance

How to define the difference between two features $f_1, f_2$?

- Simple approach is $\text{SSD}(f_1, f_2)$
  - sum of square differences between entries of the two descriptors
  - can give good scores to very ambiguous (bad) matches
Feature distance

How to define the difference between two features \( f_1, f_2 \)?

- Better approach: ratio distance = \( \frac{\text{SSD}(f_1, f_2)}{\text{SSD}(f_1, f_2')} \)
  
  - \( f_2 \) is best SSD match to \( f_1 \) in \( I_2 \)
  
  - \( f_2' \) is 2\(^{nd}\) best SSD match to \( f_1 \) in \( I_2 \)
  
  - gives small values for ambiguous matches
Evaluating the results

How can we measure the performance of a feature matcher?

feature distance
The distance threshold affects performance

- **True positives** = number of detected matches that are correct
  - Suppose we want to maximize these—how to choose threshold?

- **False positives** = number of detected matches that are incorrect
  - Suppose we want to minimize these—how to choose threshold?
Sony Aibo

SIFT usage:

- Recognize charging station
- Communicate with visual cards
- Teach object recognition
Correspondence is a vital 3D cue

We haven’t yet described how such projections are performed