Image Processing

Cosimo Distante

8 - Filtering
Last time: Color

• Measuring color
  – Spectral power distributions
  – Color mixing
  – Color matching experiments
  – Color spaces
    • Uniform color spaces

• Perception of color
  – Human photoreceptors
  – Environmental effects, adaptation

• Using color in machine vision systems
Today: Image Filters

Smooth/Sharpen Images...  Find edges...  Find waldo...
Image neighborhoods

• Q: What happens if we reshuffle all pixels within the images?

• A: Its histogram won’t change. Point-wise processing unaffected.

• Need to measure properties relative to small neighborhoods of pixels
Images as functions

Source: S. Seitz
Images as functions

• 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 \((x, y)\)
  - Realistically, we expect the image only to be defined over a rectangle, with a finite range:
    - \( f: [a,b] \times [c,d] \rightarrow [0, 1.0] \)

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

\[
\begin{bmatrix}
\begin{array}{c}
\mathbf{r}(x, y) \\
\mathbf{g}(x, y) \\
\mathbf{b}(x, y)
\end{array}
\end{bmatrix}
\]

Source: S. Seitz
Digital images

- In computer vision we operate on **digital (discrete)** images:
  - **Sample** the 2D space on a regular grid
  - **Quantize** each sample (round to nearest integer)
- Image thus represented as a matrix of integer values.

![Image](image_url)

<table>
<thead>
<tr>
<th>j</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>79</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>62</td>
</tr>
<tr>
<td>12</td>
<td>78</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>58</td>
<td>197</td>
</tr>
<tr>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>176</td>
<td>135</td>
</tr>
<tr>
<td>5</td>
<td>188</td>
</tr>
<tr>
<td>191</td>
<td>68</td>
</tr>
<tr>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>26</td>
<td>37</td>
</tr>
<tr>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>0</td>
<td>89</td>
</tr>
<tr>
<td>144</td>
<td>147</td>
</tr>
<tr>
<td>187</td>
<td>102</td>
</tr>
<tr>
<td>62</td>
<td>208</td>
</tr>
<tr>
<td>255</td>
<td>252</td>
</tr>
<tr>
<td>0</td>
<td>166</td>
</tr>
<tr>
<td>123</td>
<td>62</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>166</td>
<td>63</td>
</tr>
<tr>
<td>127</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>99</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

Adapted from S. Seitz
Motivation: noise reduction

- We can measure **noise** in multiple images of the same static scene.
- How could we reduce the noise, i.e., give an estimate of the true intensities?
Common types of noise

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

\[ f(x, y) = \hat{f}(x, y) + \eta(x, y) \]

Gaussian i.i.d. ("white") noise:
\[ \eta(x, y) \sim \mathcal{N}(\mu, \sigma) \]

>> noise = randn(size(im)).*sigma;

>> output = im + noise;
Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

\[ \text{sigma}=1 \]
Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

\[ \text{sigma} = 4 \]
Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

\[ \text{sigma=}16 \]
Effect of sigma on Gaussian noise:

This shows the noise values added to the raw intensities of an image.

$\sigma = 1$
Effect of sigma on Gaussian noise

This shows the noise values added to the raw intensities of an image.
Motivation: noise reduction

- How could we reduce the noise, i.e., give an estimate of the true intensities?
- What if there’s only one image?
First attempt at a solution

• Let’s replace each pixel with an average of all the values in its neighborhood

• Assumptions:
  – Expect pixels to be like their neighbors
  – Expect noise processes to be independent from pixel to pixel
First attempt at a solution

- Let’s replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:

Source: S. Marschner
Weighted Moving Average

• Can add weights to our moving average
• *Weights* $[1, 1, 1, 1, 1] / 5$

Source: S. Marschner
Weighted Moving Average

- Non-uniform weights \([1, 4, 6, 4, 1] / 16\)

Source: S. Marschner
Moving Average In 2D

$$F[x, y]$$  $$G[x, y]$$

Source: S. Seitz
Moving Average In 2D

\[ F[ x, y ] \]

\[ G[ x, y ] \]
Moving Average In 2D

\[ F[x, y] \]

\[ G[x, y] \]
Moving Average In 2D

\[ F[x, y] \]

\[ G[x, y] \]
Moving Average In 2D

$$F[x, y]$$

$$G[x, y]$$
Moving Average In 2D

\[ F[x, y] \]

\[ G[x, y] \]

Source: S. Seitz
Correlation filtering

Say the averaging window size is $2k+1 \times 2k+1$:

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

Attribute uniform weight Loop over all pixels in neighborhood around to each pixel image pixel $F[i,j]$.

Now generalize to allow different weights depending on neighboring pixel’s relative position:

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

Non-uniform weights
Correlation filtering

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

This is called cross-correlation, denoted \( G = H \otimes F \)

Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter “kernel” or “mask” \( H[u,v] \) is the prescription for the weights in the linear combination.
Averaging filter

- What values belong in the kernel $H$ for the moving average example?

$$G = H \otimes F$$
Smoothing by averaging

depicts box filter:
white = high value, black = low value
Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?

This kernel is an approximation of a Gaussian function:

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2 + v^2}{\sigma^2}}$$

$$H[u, v]$$
Smoothing with a Gaussian
Gaussian filters

• What parameters matter here?
• **Size** of kernel or mask
  – Note, Gaussian function has infinite support, but discrete filters use finite kernels

\[ \sigma = 5 \text{ with } 10 \times 10 \text{ kernel} \]

\[ \sigma = 5 \text{ with } 30 \times 30 \text{ kernel} \]
Gaussian filters

- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing

\[
\sigma = 2 \text{ with } 30 \times 30 \text{ kernel} \quad \text{vs} \quad \sigma = 5 \text{ with } 30 \times 30 \text{ kernel}
\]
```matlab
>> hsize = 10;
>> sigma = 5;
>> h = fspecial('gaussian', hsize, sigma);

>> mesh(h);

>> imagesc(h);

>> outim = imfilter(im, h);
>> imshow(outim);
```
Wider smoothing kernel

\( \sigma = 0.2 \)

\( \sigma = 1 \) pixel

\( \sigma = 2 \) pixels
Boundary issues

- What is the size of the output?
- MATLAB: `filter2(g, f, shape)`
  - `shape = 'full'`: output size is sum of sizes of f and g
  - `shape = 'same'`: output size is same as f
  - `shape = 'valid'`: output size is difference of sizes of f and g

Source: S. Lazebnik
Boundary issues

• What about near the edge?
  – the filter window falls off the edge of the image
  – need to extrapolate
  – methods:
    • clip filter (black)
    • wrap around
    • copy edge
    • reflect across edge

Source: S. Marschner
Boundary issues

• What about near the edge?
  – the filter window falls off the edge of the image
  – need to extrapolate
  – methods (MATLAB):
    • clip filter (black): \texttt{imfilter(f, g, 0)}
    • wrap around: \texttt{imfilter(f, g, ‘circular’)}
    • copy edge: \texttt{imfilter(f, g, ‘replicate’)}
    • reflect across edge: \texttt{imfilter(f, g, ‘symmetric’)}

Source: S. Marschner
Filtering an impulse signal

What is the result of filtering the impulse signal (image) \( F \) with the arbitrary kernel \( H \)?

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\( H[u, v] \)

\[
\begin{array}{ccc}
a & b & c \\
d & e & f \\
g & h & i \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
& & & & & & & & & & & & \\
& & & & & & & & & & & & \\
& & & & & & & & & & & & \\
& & & & & & & & & & & & \\
& & & & & & & & & & & & \\
& & & & & & & & & & & & \\
& & & & & & & & \otimes & & & & \\
& & & & & & & & & & & & \\
& & & & & & & & & & & & \\
& & & & & & & & & & & & \\
& & & & & & & & & & & & \\
\end{array}
\]

\( F[x, y] \)

\( G[x, y] \)

Grauman
Filtering an impulse signal

What is the result of filtering the impulse signal (image) $F$ with the arbitrary kernel $H$?

$F[x, y] \times H[u, v] \Rightarrow G[x, y]$
Convolution

• Convolution:
  – Flip the filter in both dimensions (bottom to top, right to left)
  – Then apply cross-correlation

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

\[ G = H \ast F \]

Notation for convolution operator
Convolution vs. correlation

**Convolution**

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

\[
G = H \ast F
\]

**Cross-correlation**

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

\[
G = H \odot F
\]

For a Gaussian or box filter, how will the outputs differ?

If the input is an impulse signal, how will the outputs differ?
Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

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

\[ G = H \ast F \]

*Notation for convolution operator*
Smoothing with a Gaussian

Parameter \( \sigma \) is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.

```
for sigma=1:3:10
    h = fspecial('gaussian', fsize, sigma);
    out = imfilter(im, h);
    imshow(out);
    pause;
end
```
Predict the filtered outputs

\[
\begin{array}{c}
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
\end{array}
\times
\begin{array}{c}
\begin{pmatrix}
\ast & & & \\
& & & \\
& & & \\
\end{pmatrix}
\end{array}
= ?
\]

\[
\begin{array}{c}
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
\end{array}
\times
\begin{array}{c}
\begin{pmatrix}
\ast & & & \\
& & & \\
& & & \\
\end{pmatrix}
\end{array}
= ?
\]

\[
\begin{array}{c}
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
\end{array}
\times
\begin{array}{c}
\begin{pmatrix}
\ast & & & \\
& & & \\
& & & \\
\end{pmatrix}
\end{array}
-rac{1}{9}
\begin{array}{c}
\begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{pmatrix}
\end{array}
= ?
\]
Practice with linear filters

Original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

?
Practice with linear filters

Original

Filtered (no change)

Source: D. Lowe
Practice with linear filters

Original

Source: D. Lowe
Practice with linear filters

Original

Shifted left by 1 pixel with correlation

Source: D. Lowe
Practice with linear filters

Original

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

\[
\frac{1}{9}
\]

Source: D. Lowe
Practice with linear filters

Original

Blur (with a box filter)

Source: D. Lowe
Practice with linear filters

Original

$$\begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} - \frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix} = \ ?$$

Source: D. Lowe
Practice with linear filters

Original

Sharpening filter
- Accentuates differences with local average

Source: D. Lowe
Filtering examples: sharpening

before

after
Shift invariant linear system

- **Shift invariant:**
  - Operator behaves the same everywhere, i.e. the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.

- **Linear:**
  - Superposition: \( h \ast (f_1 + f_2) = (h \ast f_1) + (h \ast f_2) \)
  - Scaling: \( h \ast (k \cdot f) = k \cdot (h \ast f) \)
Properties of convolution

• Linear & shift invariant

• Commutative:
  \[ f * g = g * f \]

• Associative
  \[ (f * g) * h = f * (g * h) \]

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

• Differentiation:
  \[ \frac{\partial}{\partial x} (f * g) = \frac{\partial f}{\partial x} * g \]
Separability

• In some cases, filter is separable, and we can factor into two steps:
  – Convolve all rows
  – Convolve all columns
Separability

• In some cases, filter is separable, and we can factor into two steps: e.g.,

\[
f \ast (g \ast h) = (f \ast g) \ast h
\]

What is the computational complexity advantage for a separable filter of size \( k \times k \), in terms of number of operations per output pixel?
Effect of smoothing filters

5x5

Additive Gaussian noise
Salt and pepper noise
Median filter

- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
Median filter

Salt and pepper noise

Median filtered

Plots of a row of the image

Source: M. Hebert
### Median filter

- Median filter is edge preserving

<table>
<thead>
<tr>
<th>INPUT</th>
<th>MEDIAN</th>
<th>MEAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Input Data]</td>
<td>![Median Filter Output]</td>
<td>![Mean Filter Output]</td>
</tr>
</tbody>
</table>
Filters for features

• Previously, thinking of filtering as a way to remove or reduce **noise**
• Now, consider how filters will allow us to abstract higher-level “**features**”.
  – Map raw pixels to an intermediate representation that will be used for subsequent processing
  – Goal: reduce amount of data, discard redundancy, preserve what’s useful
Template matching

• Filters as templates:
  Note that filters look like the effects they are intended to find --- “matched filters”

• Use normalized cross-correlation score to find a given pattern (template) in the image.
  – Szeliski Eq. 8.11

\[
E_{\text{NCC}}(u) = \frac{\sum_i [I_0(x_i) - \bar{I}_0] [I_1(x_i + u) - \bar{I}_1]}{\sqrt{\sum_i [I_0(x_i) - \bar{I}_0]^2} \sqrt{\sum_i [I_1(x_i + u) - \bar{I}_1]^2}},
\]

\[
\bar{I}_0 = \frac{1}{N} \sum_i I_0(x_i) \quad \bar{I}_1 = \frac{1}{N} \sum_i I_1(x_i + u)
\]

• Normalization needed to control for relative brightnesses.
Template matching

A toy example
Template matching

Detected template

Template
Template matching

Detected template

Correlation map
Where’s Waldo?

Scene

Template
Where’s Waldo?

Scene

Template
Where’s Waldo?

Detected template

Correlation map
Template matching

What if the template is not identical to some subimage in the scene?
Template matching

Match can be meaningful, if scale, orientation, and general appearance is right.