Ch3 – Spatial Filtering 1

- Overview
- Correlation and Convolution
- Neighborhood Averaging
- Binomial Filtering
- Gaussian Blurring
- Outlier Removal
- Median Filtering
- k-Nearest Neighbors
- Conclusion
Overview

• Spatial filters make use of a fixed sized neighborhood in an input image to calculate output intensities

• A large number of techniques have been invented to smooth or sharpen images

• We will consider pros/cons of some popular methods
Overview

- **Linear filters** use a weighted sum of pixels in the input image to calculate the output pixel.
- In most cases, the sum of weights is one, so the output brightness = input brightness.
- **Nonlinear filters** cannot be calculated using just a weighted sum.
- Other operations (e.g. sqrt, log, sorting, selection) are involved in the calculation.
We can formalize the phrase “weighted sum of pixels” using correlation and convolution.

\[ g(x) = f(x) \circ w(x) \]

\[ g(x) = \int_{-\infty}^{\infty} f(x+u) \cdot w(u) \, du \]

\[ g(x) = \sum_{-\infty}^{\infty} f(x+u) \cdot w(u) \]

\[ g(x) = \sum_{-N/2}^{N/2} f(x+u) \cdot w(u) \]
Correlation and Convolution

<table>
<thead>
<tr>
<th></th>
<th>f(x)</th>
<th>w(x)</th>
<th>g(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>24</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

![Graph showing f(x) and g(x)](image-url)
# Correlation and Convolution

<table>
<thead>
<tr>
<th>$f(x)$</th>
<th>6</th>
<th>6</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>6</th>
<th>6</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w(x)$</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$g(x)$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>-6</td>
</tr>
</tbody>
</table>

![Graph showing $f(x)$ and $g(x)$]

*Legend:*
- **Blue Line**: $f(x)$
- **Pink Line**: $g(x)$
Correlation and Convolution

Both formulas extend easily to two dimensions

\[ g(x, y) = f(x, y) \ast w(x, y) \]

\[ g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x-u, y-v) \cdot w(u, v) dudv \]

\[ g(x, y) = \sum_{-\infty}^{\infty} \sum_{-\infty}^{\infty} f(x-u, y-v) \cdot w(u, v) \]

\[ g(x, y) = \sum_{-N/2}^{N/2} \sum_{-N/2}^{N/2} f(x-u, y-v) \cdot w(u, v) \]
Correlation and Convolution

\[
g(x,y) = f(x,y) \ast w(x,y)
\]
Correlation and Convolution

• Some important properties of convolution:
  \[ f * g = g * f \]
  \[ f * (g * h) = (f * g) * h \]
  \[ f * (g + h) = f * g + f * h \]
  \[ a(f * g) = (af) * g = f * (ag), \text{ where } a=\text{scalar} \]
  \[ f * \delta = f, \text{ where } \delta=\text{delta function} \]
  \[ \frac{d}{dx}(f * g) = (df/dx) * g = f * (dg/dx) \]
Neighborhood Averaging

• The easiest spatial filter to implement is neighborhood averaging
• Each \((x,y)\) pixel in the image is replaced by the average of the pixels in an \(N \times N\) neighborhood centered at \((x,y)\)
• This will smooth an image and remove noise and small details
Neighborhood Averaging

// Loop over output image
for (int y = N/2; y < Ydim-N/2; y++)
for (int x = N/2; x < Xdim-N/2; x++)
{
    // Loop over NxN window
    int sum = 0;
    for (int dy = y-N/2; dy <= y+N/2; dy++)
    for (int dx = x-N/2; dx <= x+N/2; dx++)
    {
        sum += Input[dy][dx];
    }
    Output[y][x] = sum / N*N;
}

Neighborhood Averaging

- Neighborhood sizes 3x3, 5x5, 9x9, 15x15, 35x35
Neighborhood Averaging

• Several options for dealing with image border
  – Only process points within N/2 pixels of border (produces black border on output image)
  – Assume points outside the image are zero (produces a gradual black border)
  – Use wrap around to find points outside the image (mathematically best results)
  – Reduce neighborhood size near border of image (visually best results)
Binomial Filtering

- Binomial filtering uses Binomial coefficients as weights to give more emphasis to pixels near the center of the NxN neighborhood.

\[
\begin{array}{cccccc}
1 & 2 & 1 & 1 & 3 & 3 & 1 \\
2 & 4 & 2 & 3 & 9 & 9 & 3 \\
1 & 2 & 1 & 3 & 9 & 9 & 3 \\
1 & 3 & 3 & 1 & 4 & 16 & 6 & 4 & 1 \\
2 & 4 & 2 & 3 & 9 & 9 & 3 \\
1 & 2 & 1 & 3 & 9 & 9 & 3 \\
1 & 3 & 3 & 1 & 4 & 16 & 6 & 4 & 1 \\
1 & 4 & 6 & 4 & 1
\end{array}
\]
Binomial Filtering

// Loop over output image
int Weight[N][N] = { {1,2,1}, {2,4,2}, {1,2,1} };  
for (int y = N/2; y < Ydim-N/2; y++)  
for (int x = N/2; x < Xdim-N/2; x++)  
{
    // Loop over NxN window  
    int sum = 0;  
    for (int dy = -N/2; dy <= N/2; dy++)  
    for (int dx = -N/2; dx <= N/2; dx++)  
        sum += Input[dy+y][dx+x] * Weight[dy+N/2][dx+N/2];  
    Output[y][x] = sum / 16;  
}
Binomial Filtering

• Neighborhood sizes 1x1, 3x3, and 5x5
Binomial Filtering

- Neighborhood sizes 7x7, 9x9, and 11x11
Gaussian Blurring

- Gaussian filtering uses the Gaussian function to define the neighborhood weights.
Gaussian Blurring

// Initialize Gaussian weights
int Weight[N][N];
for (int y = -N/2; y <= N/2; y++)
for (int x = -N/2; x <= N/2; x++)
    Weight[y+N/2][x+N/2] =
        exp(-(x*x+y*y) / (2*sigma*sigma));
Gaussian Blurring

- Sigma values: 1, 2, 3, 4, and 5
Outlier Removal

• Weighted neighborhood averaging works for many types of noise, but it does not work well for isolated bright/dark pixels (impulse noise)
• Smoothing these points produces bright/dark smudges in the image
• A better solution would be to locate and remove these noise pixels while leaving the rest of the image unchanged
Outlier Removal

• Algorithm:
  – For each pixel calculate neighborhood average
  – Compare smoothed pixel value to original pixel value
  – If difference greater than threshold use smooth value
  – If difference less than threshold use original value

• Is this technique linear?
  – It uses neighborhood average
  – It also uses pixel comparison
Outlier Removal

// Loop over output image
for (int y = N/2; y < Ydim-N/2; y++)
for (int x = N/2; x < Xdim-N/2; x++)
{
    // Loop over NxN window
    int sum = 0;
    for (int dy = y-N/2; dy <= y+N/2; dy++)
        for (int dx = x-N/2; dx <= x+N/2; dx++)
            sum += Input[dy][dx];
    if (abs(Input[y][x] – sum / N*N) > threshold)
        Output[y][x] = (sum – Input[y][x]) / (N*N – 1);
    else
        Output[y][x] = Input[y][x];
}
Outlier Removal
Outlier Removal

10% Noise

N=5, T=30

N=3, T=50

N=7, T=10
Median Filtering

- The **mean** value of N values is not a robust statistic because an outlier can introduce bias.
- On the other hand, the **median** value of N values is a robust statistic.
- Advantage: We do not need to specifically identify outliers or decide how to replace them.
- Disadvantage: Median calculation is slower than mean calculation because of sorting.
Median Filtering

• **Algorithm:**
  – For each pixel in the input image
  – Copy the NxN neighborhood values into an array
  – Sort the NxN values in ascending or descending order
  – Use the midpoint value (the median) as the output pixel

• **Is this technique linear?**
  – No, sorting is based on pixel comparison
  – Impossible to implement using only weighted averages
## Median Filtering

### Example:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>1.8</td>
<td>2.1</td>
<td>2.3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>2</td>
<td>2.3</td>
<td>2.7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>2.3</td>
<td>2.7</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

**Input image**  **3x3 Average**  **3x3 Median**
Median Filtering

- Comparison of 3x3 average to 3x3 median
k-Nearest Neighbors

- One problem with smoothing is that it blurs the edges between different objects in the image.
- One solution is to average only the pixels that are close in brightness to the central pixel.
- Instead of using a closeness threshold, we select the k-nearest neighbors in intensity.
- This approach requires some comparisons and/or sorting to select the kNN.
k-Nearest Neighbors

• **Algorithm:**
  – For each pixel in the input image
  – Copy the NxN neighborhood values into one array
  – Store the NxN intensity differences in second array
  – Sort both arrays based on intensity differences
  – Average the k intensity values with smallest differences

• **Is this technique linear?**
  – No, sorting is based on pixel comparison
  – Impossible to implement using only weighted averages
**k-Nearest Neighbors**

Example:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input image

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>1.8</td>
<td>2.1</td>
<td>2.3</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>2</td>
<td>2.3</td>
<td>2.7</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>2.3</td>
<td>2.7</td>
<td>3</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3x3 Average

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>1.2</td>
<td>1.8</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>1.5</td>
<td>2</td>
<td>2.5</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3x3 kNN6
k-Nearest Neighbors

original, original+noise, kNN, true noise, original-kNN
Conclusion

- The spatial filtering techniques we have seen so far give us effective ways to smooth images to remove noise or unwanted details.
- The linear methods use different convolution masks that work well for uniform noise.
- The nonlinear methods use selection and/or sorting to remove impulse noise.
- Linear filtering is significantly faster than nonlinear filtering.