Neighborhood operations

- Generate an output pixel on the basis of the pixel and its neighbors
- Often involve the convolution of an image with a filter kernel or mask

\[ g(i, j) = f * h = \sum_m \sum_n f(i-m, j-n)h(m,n) \]

//example: for a 3x3 kernel
for \( r = 1 : H \)
  for \( c = 1 : W \)
    // for each feasible points in the image
    temp=0.0;
    for \( m = -1 : 1 \)
      for \( n = -1 : 1 \)
        temp=temp+f(r-m,c-n)*h(m,n);
    end
    end
    g(r,c)=temp;
  end
  end

Computational cost, order of \( m\times n\times W\times H \) for an image \( W\times H \)

Example: noise suppression

- Assuming (usually correctly) that the noise has a high spatial frequency
- Apply a low-pass spatial filter
- Of course high-frequencies in the image will be degraded after filtering...

\[
101\times\frac{1}{9} + 100\times\frac{1}{9} + 103\times\frac{1}{9} + 110\times\frac{1}{9} + 140\times\frac{1}{9} + 120\times\frac{1}{9} + 134\times\frac{1}{9} + 134\times\frac{1}{9} + 135\times\frac{1}{9}
\]
Gaussian smoothing

- The image is convolved with a Gaussian function

\[ G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2 + y^2}{2\sigma^2}\right)} \]

\( \sigma \) defines the spread of the function
small \( \sigma \) → narrow gaussian
large \( \sigma \) → broad gaussian

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>7</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16</td>
<td>26</td>
<td>16</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>26</td>
<td>41</td>
<td>26</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>26</td>
<td>16</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

from: http://www.cee.hw.ac.uk/hipr/html/gsmooth.html
Why smooth with a Gaussian? (1)

- The Gaussian filter has nice properties in the frequency domain:

\[ \mathcal{F}(\text{rect}(ax)) = \frac{1}{a} \cdot \text{sinc} \left( \frac{\xi}{a} \right) \]

\[ \mathcal{F}(e^{-\alpha x^2}) = \sqrt{\frac{\pi}{\alpha}} \cdot e^{-\frac{(\pi \xi)^2}{\alpha}} \]

Beware of aliasing…
Why smooth with a Gaussian? (2)

• Convolving a Gaussian with a Gaussian, gives a Gaussian:

\[ G_{\sigma_1} \ast G_{\sigma_2} = G_{\sqrt{\sigma_1^2 + \sigma_2^2}} \]

• it is possible to obtain heavily smoothed images by resmoothing smoothed images…
Why smooth with a Gaussian? (3)

- Filtering with a 2D Gaussian can be separated in two convolutions with one dimensional Gaussian function:

\[
G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}
\]
Why smooth with a Gaussian? (3)

- Filtering with a 2D Gaussian can be separated in two convolutions with one dimensional Gaussian function:

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

\[
g (i, j) = f \ast g = \sum_m \sum_n g (m, n) f (i - m, j - n) =
\]

\[
= \sum_m \sum_n e^{-\frac{m^2 + n^2}{2\sigma^2}} f (i - m, j - n) =
\]
\[ e^{-\frac{m^2}{2\sigma^2}} \sum_n e^{-\frac{n^2}{2\sigma^2}} f(i - m, j - n) = \]
\[ = \sum_m e^{-\frac{m^2}{2\sigma^2}} h'(i - m, j) \]
\[ \text{convolve with a vertical 1D Gaussian} \]
\[ \text{convolve the result with an horizontal 1D Gaussian} \]

- Convolving with a separable filter is the same as convolving with two 1D kernels, but faster:
\[ nxHxW + mxHxW \text{ operations, instead of } nxmxHxW \]
Example

- Above: images corrupted by normally distributed additive noise (std 5, 10, 15, 20)
- Below: smoothing with a gaussian filter 10x10 std=3
Median Filter

- Non linear technique, useful for noise suppression
- In one dimension:
  - slide a window of an odd number of pixels
  - replace the center pixel with the median within the window
- The median of a discrete sequence of $N$ (odd) elements is the number so that $(N-1)/2$ elements are smaller or equal in value and $(N-1)/2$ elements are larger or equal in value
  - In practice: sort the pixels and pick the value in the middle...

<table>
<thead>
<tr>
<th>123</th>
<th>125</th>
<th>126</th>
<th>130</th>
<th>140</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>124</td>
<td>126</td>
<td>127</td>
<td>135</td>
</tr>
<tr>
<td>118</td>
<td>120</td>
<td>150</td>
<td>125</td>
<td>134</td>
</tr>
<tr>
<td>119</td>
<td>115</td>
<td>119</td>
<td>123</td>
<td>133</td>
</tr>
<tr>
<td>111</td>
<td>116</td>
<td>110</td>
<td>120</td>
<td>130</td>
</tr>
</tbody>
</table>

Neighbourhood values:

115, 119, 120, 123, 124, 125, 126, 127, 150

Median value: 124
Median Filter

- Non linear technique, useful for noise supression
- In one dimension:
  - slide a window of an odd number of pixels
  - replace the center pixel with the median within the window
- The median of a discrete sequence of N (odd) elements is the number so that (N-1)/2 elements are smaller or equal in value and (N-1)/2 elements are larger or equal in value
  - In practice: sort the pixels and pick the value in the middle…

```
<table>
<thead>
<tr>
<th>123</th>
<th>125</th>
<th>126</th>
<th>130</th>
<th>140</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>124</td>
<td>126</td>
<td>127</td>
<td>135</td>
</tr>
<tr>
<td>118</td>
<td>120</td>
<td>150</td>
<td>125</td>
<td>134</td>
</tr>
<tr>
<td>119</td>
<td>115</td>
<td>119</td>
<td>123</td>
<td>133</td>
</tr>
<tr>
<td>111</td>
<td>116</td>
<td>110</td>
<td>120</td>
<td>130</td>
</tr>
</tbody>
</table>
```

Neighbourhood values: 115, 119, 120, 123, 124, 125, 126, 127, 150

```
<table>
<thead>
<tr>
<th>123</th>
<th>125</th>
<th>126</th>
<th>130</th>
<th>140</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>124</td>
<td>126</td>
<td>127</td>
<td>135</td>
</tr>
<tr>
<td>118</td>
<td>120</td>
<td>125</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>115</td>
<td>123</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>116</td>
<td>120</td>
<td>130</td>
<td></td>
</tr>
</tbody>
</table>
```
**FIGURE 12.6-1.** Examples of median filtering on primitive signals, $L = 5$.

<table>
<thead>
<tr>
<th>ORIGINAL</th>
<th>MEAN FILTERED</th>
<th>MEDIAN FILTERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) STEP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b) RAMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c) SINGLE PULSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d) DOUBLE PULSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e) TRIPLE PULSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f) TRIANGLE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 12.6-3.** Examples of one-dimensional median filtering for images corrupted by impulse noise. (a) Image with impulse noise, 15 errors per line. (b) Median filtering of (a) with $L = 3$. (c) Median filtering of (a) with $L = 5$. (d) Median filtering of (a) with $L = 7$. 

*W.K. Pratt, Digital Image Processing*
Edge detection

- It is an example of *feature extraction*
- Edges correspond to abrupt changes of luminous intensity in the image
- They usually correspond to discontinuities in the visual scene, due to illumination, object surface or material → object boundaries
- Estimate the gradient of the image:

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

- The *rate of change* of the image is maximum along the direction:

\[
\theta = \arctan \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)
\]

- With magnitude:

\[
|\nabla f| = g(x, y) = \sqrt{\left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2}
\]
• Simplest way to estimate the derivatives:

\[
\frac{\partial f}{\partial x} = f(x+1, y) - f(x, y), \quad \frac{\partial f}{\partial y} = f(x, y+1) - f(x, y)
\]

• Another approach (Roberts) compute the derivatives diagonally over a 2x2 region:

\[
g(x, y) \approx R(x, y) = \sqrt{[f(x+1, y+1) - f(x, y)]^2 + [f(x+1, y) - f(x, y+1)]^2}
\]

• First differences are sensitive to noise, a better approach is to combine differencing with local averaging. For example Sobel operator:

\[
S_y = [f(x-1, y+1) + 2f(x, y+1) + f(x+1, y+1)]
- [f(x-1, y-1) + 2f(x, y-1) + f(x+1, y-1)]
\]

\[
S_x = [f(x+1, y-1) + 2f(x+1, y) + f(x+1, y+1)]
- [f(x-1, y-1) + 2f(x-1, y) + f(x-1, y+1)]
\]

\[
g(x, y) \approx \sqrt{S_x^2 + S_y^2}
\]

or: \[g(x, y) = |S_x| + |S_y|\]
Edge operators can be represented as convolution kernels:

a) Roberts
b) Prewitt
c) Sobel

<table>
<thead>
<tr>
<th>$\Delta_1$</th>
<th>$\Delta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>-1 0</td>
<td>0 -1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\Delta_1$</th>
<th>$\Delta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 0 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>-1 0 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>-1 0 1</td>
<td>-1 -1 -1</td>
</tr>
</tbody>
</table>

\[
\frac{\partial I}{\partial x} = K_x \ast I
\]

\[
\frac{\partial (S \ast I)}{\partial x} = (K_x \ast S) \ast I
\]

\[
\frac{\partial (S \ast I)}{\partial x} = \frac{\partial S}{\partial x} \ast I
\]

If S is Gaussian then the kernel is a derivative of a G.
• Once the gradient magnitude has been estimated, decide if an edge is present or not based on a threshold:

\[
edge(x, y) = \begin{cases} 
0 & \text{if } g(x, y) < t \\
1 & \text{otherwise}
\end{cases}
\]

```matlab
I = imread('rice.tif');
eRob = edge(I, 'roberts');
ePre = edge(I,'prewitt');
eSob = edge(I,'sobel');
figure(1), imshow(eRob)
figure(2), imshow(ePre)
figure(3), imshow(eSob)
```
Laplacian

- Alternative method, computes the Laplacian:

\[ \nabla^2 f(x, y) = \left( \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \right) \]

- Approximation:

1d: \( f'(x+1) = f(x+1) - f(x) \), \( f'(x) = f(x) - f(x-1) \) \( \Rightarrow f''(x) = f'(x+1) - f'(x) \)

2d: convolve with kernels:

<table>
<thead>
<tr>
<th>0</th>
<th>-1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

or

<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>

- Zero response to linear ramps (gradual changes in intensity), respond to either sides of edges (+/-) \( \Rightarrow \) detect edges as zero crossing
- Drawback \( \Rightarrow \) strong response to noise
- First convolve with a Gaussian (Marr and Hildreth)
The Laplacian operator has some nice properties (and is a linear kernel):
\[ \nabla^2 \{G(x, y) * I(x, y)\} = \nabla^2 G(x, y) * I(x, y) \]

\[ \nabla^2 G(x, y) \] is also called LoG (Laplacian of Gaussian)

To recap the Laplacian is:
- zero distant from the edge
- > 0 just before the edge
- zero in between the edge
- < 0 after the edge
More on the LoG (useful things to know)

- The two dimensional convolution can be separated into four one-dimensional convolution (cost efficiency):

\[
\nabla^2 \{I(x, y) * G(x, y)\} = G(x) * \left\{ I(x, y) * \frac{\partial^2}{\partial^2 y} G(y) \right\} + G(y) * \left\{ I(x, y) * \frac{\partial^2}{\partial^2 x} G(x) \right\}
\]

- The LoG can be approximated as (again, easy to compute):

\[
g_1(x, y) - g_2(x, y) = G_{\sigma_1} * f(x, y) - G_{\sigma_2} * f(x, y) = \left( G_{\sigma_1} - G_{\sigma_2} \right) * f(x, y) = DoG * f(x, y)
\]

\[\sigma_1 > \sigma_2\]
Canny Edge Detector (1986)

- Problem of the edge detectors: produce thick edges, sometimes edges are not connected because of noise

- Smooth with a Gaussian, then apply Sobel
- Thin edges, *non-maxima suppression*:
  Edge is found if
  a) response exceeds a given threshold *and*
  b) it is not dominated by responses at neighboring points in a direction normal to the candidate edge → the edge must be higher than the edge magnitude of the pixels on either side
Canny Edge (2)

- Extends weak edges: Hysteresis controlled by two thresholds $T_1 > T_2$; all pixels above $T_1$ are marked as edge; then all pixels connected to these edges whose value is above $T_2$ will be selected as edge (avoid ‘dashed edges’).
While there are points with high gradient that have not been visited

Find a start point that is a local maximum in the direction perpendicular to the gradient erasing points that have been checked

while possible, expand a chain through the current point by:

1) predicting a set of next points, using the direction perpendicular to the gradient

2) finding which (if any) is a local maximum in the gradient direction

3) testing if the gradient magnitude at the maximum is sufficiently large

4) leaving a record that the point and neighbours have been visited

record the next point, which becomes the current point

end