5. Neighborhood Operations
Neighborhood operations

- Linear filtering
- Smoothing, edge detection, pattern matching
- Canny edge detection
- Distance operations
- Morphological operations
- Skeletonization
- Performance implications
Neighborhood vs. Window

- These are related terms
  - The neighborhood of a pixel are the pixels that are connected to it
    - 4-neighborhood
    - 8-neighborhood
  - The window is a rectangular subtile of the image centered around the pixel
    - A 13x7 window
      - We’ll take this as 13 rows and 7 columns
      - 6 rows above, 6 rows below, 3 cols left, 3 cols right
    - Typical to consider only odd-numbered window dimensions
- Neighborhood is used for morphological (shape-based) operations
  - Window typically used for filtering operations
Reducing noise

- Imagine that you have spatial grids that are noisy
  - Instrument artifacts, etc. that are stochastic in nature
- You measure the same thing many times
  - Can then treat each pixel as an independent observation
  - Reduce noise by replacing each pixel value by the average value at that pixel over time
    - Just standard statistics
    - The mean is the maximum likelihood estimator for many probability distributions, especially the Normal distribution
- But what if you don’t have multiple observations of a spatial grid?
Stationarity/Markov

- Make an assumption:
  - The neighbors of a pixel are drawn from the same probability distribution as this one
  - So, find maximum-likelihood-estimate of this pixel’s value
  - Replace pixel by MLE
    - Note that this replacement can not be carried out in-place!
Boxcar Filter

- Can smooth an image by averaging in neighborhood:

```
 1 1 1
1 1 1
1 1 1
```

- But the “boxcar” filter is subject to ringing artifacts

Log(population)  After 5x5 boxcar
Handling boundaries and missing data

- What do you do at boundaries of image?
  - Can not use all the coefficients if your center point is near boundary
  - Several options:
    - Zero out the output image near the boundaries
    - Zero pad the input image before convolving
      - Crazy side effects possible
    - Use only the coefficients that are within image
      - And scale denominator appropriately
      - Doable for smoothing; less so for other operations such as edges
- Dealing with missing data within image similarly has varying tradeoffs
  - Do not use it in computing weighted average
  - Treat it as zero
Convolution

- Compute a weighted sum of values within a pixel’s window
  - Create a copy of the image where each pixel’s value is replaced by the weighted sum of values within that pixel’s window in the original image
A better smoothing filter is the Gaussian
  - Highest weight to center, lower weights towards edges
  - Make sure that coefficients sum up to one
    - Otherwise, “amplification”
  - Common to truncate Gaussian coefficients at 3*sigma

Log(population)  After 5x5 boxcar  After 11x11 Gaussian (sigma = 11/6)
Bandwidth of Gaussian

- In practice, the size of the window of a Gaussian filter has to be 3x the sigma of the Gaussian (why?)

- So, for similar levels of smoothing, the Gaussian has to be larger than a boxcar.
Boxcar
Gauss
Why the shifts and the ringing?

- When you smooth an image you get rid of fast variations in that image
  - Also results in displacement of slow-moving features
  - The larger the filter the more the shift

- The Fourier transform of a Gaussian is also a Gaussian
  - So, the frequency components you keep are also slowly decaying

- The Fourier transform of a boxcar filter is a sinc function
  - Interference pattern when you pass light through a pinhole …
  - Results in adjacent high frequencies being treated differently
Doing convolution in spatial domain:
- NxN image
- PxP filter

At each pixel of original image, have to carry out PxP multiplications
- $O(\text{NxNxPxP})$

The larger the filter, the slower it is
- So, might want to look for filters that are smaller even if they come with a little ringing …

Will look at transform methods that provide speed up …
- But a spatial optimization exists
Separability

- If the 2D function can be written as a product of two 1D functions
  - \( G(x,y) = G(x) \cdot G(y) \)

- Then, instead of convolving in 2D neighborhood, can convolve all rows by 1D row filter, then convolve columns of result by 1D column filter

- This works for a Gaussian filter, for example
  - Can cut operations from \( N^2 \) to \( 2N \), so helpful for large neighborhoods
Pattern Matching

- Can think of convolution as filtering with a “matched” filter
  - If filter and pixels within window are normalized i.e. add to 1, then maximum response when pixels in window align with filter
Example pattern match

- What would this filter match?

\[
\begin{array}{cccccc}
0.167 & 0.167 & 0.167 & 0.000 & 0.000 \\
0.167 & 0.167 & 0.000 & 0.000 & 0.000 \\
0.167 & 0.000 & 0.000 & 0.000 & 0.000 \\
\end{array}
\]
Example of pattern match

(a) Original
(b) 3x3 Northwest edges
(c) 7x7 Northwest edges
(d) 11x11 Northwest edges
Limitation of matched filter

Consider:

```
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.111 0.111 0.111 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.111 0.111 0.111 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.111 0.111 0.111 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.111 0.111 0.111 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.111 0.111 0.111 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.111 0.111 0.111 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

(a) Original
(b) 11x11 isolated areas
(c) 23x23 isolated areas
(d) 29x29 isolated areas
Matched filters are impractical

- Only 23x23 works well
  - How would you know this in advance?
- Problem: have to match size, orientation of shape being searched for
  - Impractical, except in robotics applications
- Alternately … try out different sizes, orientations & choose best
Smooth along the front

- How could you use the matched filter idea to smooth this image?
  - Want to smooth “along the front”
Filter bank

Fig. 5.6 Oriented filters to smooth along particular directions.

\[
\frac{x \cos \theta - y \sin \theta^2}{a} + \frac{x \sin \theta + y \cos \theta^2}{b} \leq 1 \quad (5.1)
\]
Example filter bank

Fig. 5.7 Smoothing a population density image using 3x11 filters oriented in different directions.
Example filter bank

- Final result at any pixel is the maximum response to any of the oriented filters at that pixel (smoothing along best direction)

- Could also choose result from direction with least variance
Better oriented filter

- Ellipse filter is similar to a boxcar filter
  - Ringing artifacts
  - Better to use Gaussian
- Problem: performance
  - An isotropic Gaussian is separable:
    \[ G(x,y) = e^{-\left(\frac{x^2}{2}\right)} = e^{-x^2} e^{-\frac{y^2}{2}} = G(x)G(y) \]
  - But not an oriented filter:
    \[ G(x,y) = e^{-\left(\frac{x^2 \cos \theta - y \sin \theta}{2}\right)} + (x \sin \theta + y \cos \theta)^2 \]
- Recommended reading:
The orientation varies throughout the image

- So use a filter bank of filters at several possible orientations
  - Theta varies from 0 to 180 in increments of 10-deg, for example

\[
f(x, y) = 1 \text{ if } \frac{(x \cos \theta - y \sin \theta)^2}{a^2} + \frac{(x \sin \theta + y \cos \theta)^2}{b^2} \leq 1
\]

- Take maximum of the results of each
- Very, very time consuming
  - Optimization exists:
Sobel Edge Filter

- Compute the horizontal and vertical gradient by convolving the image twice:

\[
\begin{array}{ccc}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1 \\
\end{array}
\quad
\begin{array}{ccc}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 \\
\end{array}
\]

Gx  
Gy

- Then take \(|G_x| + |G_y|\) (or the actual magnitude by \(\sqrt{G_x \cdot G_x + G_y \cdot G_y}\))
- The resulting field is maximum at edges
What if …

- What if you compute the edges on the original data
  - Instead of log-scaled?

- The Sobel filter works if the data values are linearly scaled
  - Otherwise, it merely picks up maxima, not edges
What if …

- What if you smooth the data before finding edges?

  - Edges become thicker!
    - But also find more significant variations

  - Is it possible to keep edges thin while smoothing to remove very high frequencies (noise)
LoG

- Laplacian of a Gaussian (LoG) idea
  - Smooth image before finding edges
  - Define edge as center of the thick edge that is found
- Convolving grid with a Gaussian and then computing gradient
  - Is same as convolving grid with the gradient of a Gaussian
- The LoG filter window equation is:
  \[
  \Delta(x,y) = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} e^{-\frac{x^2 + y^2}{2\sigma^2}}
  \]
  - Filter has to sum up to zero
    - Customary to compute all coefficients except the center one
    - Then set the center one so that sum of coeffs is zero
    - This approach takes care of scaling factors
- Find zero crossing in any of the 4 directions and keep only strong edges
Canny Edge Filter


- Consists of:
  - Convolve image with first derivative of Gaussian
  - Compare gradient magnitude to the gradient in the direction of the gradient (e.g: for horizontal gradients, compare $G_y$ to total gradient)
    - Suppress gradients that are dispersed
  - Follow edges
    - Start at a high gradient magnitude (threshold $T_1$)
    - Connect all gradient pixels with magnitude $> T_2$
      - $T_2$ is typically about $\frac{1}{2}$ of $T_1$
Median Filtering

- One of the best speckle filters
  - Replace each pixel by the median of its neighbors
  - Also smooths the image
    - Tends to preserve edges better than Gaussian

Original; median in 3x3, 7x7 and 11x11 neighborhoods
Median Filtering: Speed improvement

- Not necessary to sort all the pixels
  - “nth_element” algorithms exist that can find median without sorting
  - Look for “quick select” in “Numerical Recipes”

- Sorting a P x P neighborhood would be on the order of P log(P)
  - Finding P x P/2 th element of list of size P x P is on the order of P (on average)
Using histogram for median filtering

Can find median from histogram

The windows at successive pixels overlap, so can update histogram


Works only if data range are limited or can be quantized to fall into a very limited range
Speckle filtering

- Find if a pixel is “noisy” (if it is too different from the median)
- Replace noisy pixels by median value

Not perfect!
Morphological operations

- Can improve many spatial operations by taking advantage of the high correlation between adjacent pixels
Morphological Operations

- Replace a pixel by the maximum in that pixel’s neighborhood
  - What happens?

- Dilation of the shape
  - A morphological operation

- Can be used to fill in holes
  - Size of neighborhood ~ size of holes filled
Dilation

Original; dilate in 3x3, 7x7 and 11x11 neighborhoods

Problem? Blockiness ...
Erosion

- Replace a pixel by the maximum in that pixel’s neighborhood
  - What happens?

- Erosion of the shape
  - A morphological operation

- Can be used to remove speckle
  - Size of neighborhood ~ size of speckle removed
Erosion

Original; erode in 3x3, 7x7 and 11x11 neighborhoods

Problem? Blockiness ...
Dilate + Erode

- Carry out dilation and erosion successively
  - Dilate, then erode result of dilation
    - Will not increase size of areas, just fill in holes

Original; dilate+erode in 3x3 neighborhood 3 times; 7 times; in 5x5 neighborhood 3 times
Thinning thick objects

- The skeleton consists of points that somewhat characterize the shape of a region.

- How can we get at the skeleton?
From distance transform …

- One way to create a skeleton is to approximate by the medial axis
  - Start with a distance transform
    - Distance from “boundary” (skeleton only defined for binary images)
  - Retain only pixels that are the local maxima
    - Consider a 4-neighborhood here
Medial Axis Transform

Above: Original; > 100K/km^2; EDT; skeleton (sensitive to noise!)
Below: first dilate+erode (5x5 neighborhood, 3 times) + erode (5x5)
Another approach is to look in 3x3 neighborhood and remove pixels on the boundary until that pixel becomes essential to maintain continuity.

- Basic idea: remove pixels that are on a “solid” boundary
  - Also consider exceptions such as two-pixel-wide lines
Hilditch Algorithm

- Create a binary image consisting of 1s for pixels within object and 0s outside the object.
- For every pixel p1, define its neighborhood as:
  - A(p1) is the number of zero to one transitions moving clockwise from p2 to p9.
  - B(p1) is the number of non-zero neighbors of p1.
- March through binary image and remove a pixel p1 if all 4 of these are true:
  1. \(2 \leq B(p1) \leq 6\)
  2. \(A(p1) = 1\)
  3. \(p2.p4.p6 = 0\) or \(A(p2) \neq 1\)
  4. \(p2.p4.p8 = 0\) or \(A(p4) \neq 1\)
- Condition 1 is met only for boundary pixels that are neither isolated nor essential.
- Condition 2 is for skeleton continuity.
- Conditions 3 and 4 handle two-pixel-wide boundaries.
- Iterate until thing start to converge.
Other Thinning Algorithms

- Other skeletonization algorithms exist, for example:
- But the MAT and Hilditch algorithms suffice for most practical purposes
Assignment

- Pick a spatial grid (any grid: your choice)
  - A good opportunity to start thinking about your final project
  - Using neighborhood operations (smoothing, filter banks, morphological, edge filters, thinning), isolate features of interest

- Submit a 1 to 2 page report and critical code:
  - Describe your dataset
  - What you wanted to isolate
  - What kind of operations you carried out on your dataset
    - Why you believed those operations would be helpful
  - Successive stages of results

- You will need to present your results on this assignment to class
Frequency Domain Convolution

of the filtering the grid using these window coefficients as $W \otimes I$. Suppose $\mathcal{F}(x)$ represents the Digital Fourier Transform (DFT) of $x$. Then:

$$\mathcal{F}(W \otimes I) = \mathcal{F}(W) \cdot \mathcal{F}(I)$$  \hspace{1cm} (5.7)

In other words, the DFT of the result is equal to the pixel-by-pixel product of the DFTs of the weights and the grid.

One way to speed up the filtering of large grids is to follow this process:

1. Pad the weights array with zeroes so that it is the same size as the grid.
2. Find the DFT of the padded weights array.
3. Find the DFT of the spatial grid.
4. Do a pixel-by-pixel product of the two DFTs.
5. Compute the inverse DFT of the product. This is the result of filtering.

The results of convolving the nationwide population density grid using a 301x301 Gaussian filter is shown in Figure 5.22. The result is the same as what we would have obtained had we carried it out in the spatial domain except that the treatment of data in the boundaries is different. When the smoothing was carried out on a mid-range laptop computer (in 2011), the FFT technique took 11.5 seconds whereas the spatial domain convolution took a whopping 223 seconds (more than 3 minutes). When your windows are large, it pays to carry out filtering in the frequency domain. For smaller windows, the difference in performance is not as dramatic.
Caveats

- Caveats to FFT approach:
  - Have to have non-missing data at every pixel
    - Replace missing data by zero or by the mean over the entire grid or by nearby values
  - The grid size needs to be amenable to Fast Fourier Transform
    - FFTs exist for most multiples of small prime numbers, but the traditional (and fastest/simplest) FFT is for powers of two
    - Pad your image to reach this size
Reading

- Read:

- Will discuss paper in class

- Read up to the end of ch. 5 in the class text book