Announcements

• You'll take your first midterm exam next week!
• Your 1st programming assignment will be also announced next week!

Image Filtering

• **Image filtering**: computes a function of a local neighborhood at each pixel position
• Called “Local operator,” “Neighborhood operator,” or “Window operator”
• \( f \): image \( \rightarrow \) image
• Uses:
  – Enhance images
    • Noise reduction, smooth, resize, increase contrast, recolor, artistic effects, etc.
  – Extract features from images
    • Texture, edges, distinctive points, etc.
  – Detect patterns
    • Template matching, e.g., eye template

Filtering

• The name “filter” is borrowed from frequency domain processing (next week’s topic)
• Accept or reject certain frequency components
• Fourier (1807):
  Periodic functions could be represented as a weighted sum of sines and cosines
Signals

- A signal is composed of low and high frequency components

  low frequency components: smooth / piecewise smooth
  Neighboring pixels have similar brightness values
  You’re within a region

  high frequency components: oscillatory
  Neighboring pixels have different brightness values
  You’re either at the edges or noise points

Signals – Examples

Low/high frequencies vs. fine/coarse-scale details

- Assume image is degraded with an additive model.
- Then,

  Observation = True signal + noise
  Observed image = Actual image + noise

  low-pass filters
  high-pass filters

  smooth the image
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

Motivation: noise reduction

- Make multiple observations of the same static scene
- Take the average
- Even multiple images of the same static scene will not be identical.

Motivation: noise reduction

- Make multiple observations of the same static scene
- Take the average
- Even multiple images of the same static scene will not be identical.
- What if we can’t make multiple observations? **What if there’s only one image?**
Image Filtering

- **Idea**: Use the information coming from the neighboring pixels for processing.
- **Design**: A transformation function of the local neighborhood at each pixel in the image.
  - Function specified by a “filter” or mask saying how to combine values from neighbors.
- **Various uses of filtering**:
  - Enhance an image (denoise, resize, etc)
  - Extract information (texture, edges, etc)
  - Detect patterns (template matching)

Filtering

- Processing done on a function
  - can be executed in continuous form (e.g. analog circuit)
  - but can also be executed using sampled representation
- **Simple example**: smoothing by averaging

Linear filtering

- Filtered value is the linear combination of neighboring pixel values.
- **Key properties**
  - linearity: \( \text{filter}(f + g) = \text{filter}(f) + \text{filter}(g) \)
  - shift invariance: behavior invariant to shifting the input
    - delaying an audio signal
    - sliding an image around
- Can be modeled mathematically by convolution

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 (spatial regularity in images)
  - 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:

Convolution warm-up

• Same moving average operation, expressed mathematically:

\[ b_{\text{smooth}}[i] = \frac{1}{2r + 1} \sum_{j=-r}^{i+r} b[j] \]

Discrete convolution

• Simple averaging:

\[ b_{\text{smooth}}[i] = \frac{1}{2r + 1} \sum_{j=-r}^{i+r} b[j] \]
  
  – every sample gets the same weight
• Convolution: same idea but with weighted average

\[ (a \ast b)[i] = \sum_{j} a[j]b[i - j] \]
  
  – each sample gets its own weight (normally zero far away)
• This is all convolution is: it is a moving weighted average

Filters

• Sequence of weights \( a[j] \) is called a filter
• Filter is nonzero over its region of support
  – usually centered on zero: support radius \( r \)
• Filter is normalized so that it sums to 1.0
  – this makes for a weighted average, not just any old weighted sum
• Most filters are symmetric about 0
  – since for images we usually want to treat left and right the same

Filters

• Sequence of weights \( a[j] \) is called a filter
• Filter is nonzero over its region of support
  – usually centered on zero: support radius \( r \)
• Filter is normalized so that it sums to 1.0
  – this makes for a weighted average, not just any old weighted sum
• Most filters are symmetric about 0
  – since for images we usually want to treat left and right the same
Convolution and filtering

- Can express sliding average as convolution with a box filter
- \(a_{\text{box}} = [\ldots, 0, 1, 1, 1, 1, 1, 0, \ldots]\)

Example: box and step

Convolution and filtering

- Convolution applies with any sequence of weights
- Example: bell curve (gaussian-like) \(\ldots, 1, 4, 6, 4, 1, \ldots) / 16\)

And in pseudocode...

```plaintext
function convolve(sequence a, sequence b, int r, int i)
    s = 0
    for j = -r to r
        s = s + a[j] * b[i - j]
    return s
```
**Key properties**

- **Linearity:** $\text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2)$
- **Shift invariance:** $\text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f))$
  - Same behavior regardless of pixel location, i.e., the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.
  - Theoretical result: any linear shift-invariant operator can be represented as a convolution

**Properties in more detail**

- **Commutative:** $a \ast b = b \ast a$
  - Conceptually no difference between filter and signal

- **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: $a \ast (b_1 \ast b_2 \ast b_3)$

- **Distributes over addition:** $a \ast (b + c) = (a \ast b) + (a \ast c)$

- **Scalars factor out:** $ka \ast b = a \ast kb = k(a \ast b)$

- **Identity:** unit impulse $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$

  - $a \ast e = a$

**A gallery of filters**

- **Box filter**
  - Simple and cheap

- **Tent filter**
  - Linear interpolation

- **Gaussian filter**
  - Very smooth antialiasing filter

**Box filter**

$$q_{\text{box},r}[i] = \begin{cases} 
1/(2r + 1) & |i| \leq r, \\
0 & \text{otherwise}
\end{cases}$$

$$f_{\text{box},r}(x) = \begin{cases} 
1/(2r) & -r \leq x < r, \\
0 & \text{otherwise}
\end{cases}$$
**Tent filter**

\[
 f_{\text{tent}}(x) = \begin{cases} 
 1 - |x| & |x| < 1, \\
 0 & \text{otherwise};
\end{cases}
\]

\[
 f_{\text{tent},r}(x) = \frac{f_{\text{tent}}(x/r)}{r}.
\]

---

**Gaussian filter**

\[
 f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.
\]

---

**Discrete filtering in 2D**

- Same equation, one more index
  
  \[
  (a * b)[i, j] = \sum_{i', j'} a[i', j'][b[i - i', j - j']]
  \]
  
  - now the filter is a rectangle you slide around over a grid of numbers
  
  - Usefulness of associativity
    
    - often apply several filters one after another: \(((a * b_1) * b_2) * b_3\)
    
    - this is equivalent to applying one filter: \(a * (b_1 * b_2 * b_3)\)

---

**And in pseudocode...**

```pseudocode
function convolve2d(filter2d a, filter2d b, int i, int j)
    s = 0
    r = a.radius
    for j' = -r to r do
        for j'' = -r to r do
            s = s + a[i''][j''] * b[i - i'][j - j'']
        end for
    end for
    return s
```

---

Slide credits: S. Marschner
Moving Average In 2D

\[ F[x, y] \]

\[ G[x, y] \]
Image Correlation Filtering

- Center filter $g$ at each pixel in image $f$
- Multiply weights by corresponding pixels
- Set resulting value in output image $h$
- $g$ is called a filter, mask, kernel, or template
- Linear filtering is sum of dot product at each pixel position
- Filtering operation called cross-correlation

Correlation filtering

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

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

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.

Correlation filtering

Cross correlation example

Slide credit: K. Grauman

Slide credit: Fei-Fei Li
Averaging filter

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

$$ F[x, y] \ast H[u, v] = G[x, y] $$

- "box filter"

Smoothing by averaging

- What if the filter size was 5 x 5 instead of 3 x 3?

Boundary issues

- What is the size of the output?
- MATLAB: output size / "shape" options
  - 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

- 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
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’)

Gaussian filter

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

This kernel is an approximation of a 2d Gaussian function:

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

Smoothing with a Gaussian

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

Gaussian filters

- \( \sigma = 5 \) with 10 x 10 kernel
- \( \sigma = 5 \) with 30 x 30 kernel
Gaussian filters

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

Matlab

```matlab
>> hsize = 10;
>> sigma = 5;
>> h = fspecial('gaussian', hsize, sigma);

>> mesh(h);
>> imagesc(h);
>> outim = imfilter(im, h); % correlation
>> imshow(outim);
```

Choosing kernel width

• Rule of thumb: set filter half-width to about $3 \sigma$

Smoothing with a Gaussian

Parameter $\sigma$ is the "scale"/"width"/"spread" of the Gaussian kernel, and controls the amount of smoothing.
**Gaussian Filters**

\[ \sigma = 1 \text{ pixel} \quad \sigma = 5 \text{ pixels} \quad \sigma = 10 \text{ pixels} \quad \sigma = 30 \text{ pixels} \]

**Spatial Resolution and Color**

**Blurring the G Component**

**Blurring the R Component**
**Blurring the B Component**

Original

Processed

Slide credit: C. Dyer

**“Lab” Color Representation**

A transformation of the colors into a color space that is more perceptually meaningful: L: luminance, a: red-green, b: blue-yellow

Slide credit: C. Dyer

**Blurring L**

Original

Processed

Slide credit: C. Dyer

**Blurring a**

Original

Processed

Slide credit: C. Dyer
Blurring b

original  processed

Separability

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

Separability of the Gaussian filter

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

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y.

In this case, the two functions are the (identical) 1D Gaussian filters:

Separability example

2D convolution (center location only):

\[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1
\end{bmatrix}
\times
\begin{bmatrix}
2 & 3 & 3 \\
3 & 5 & 5 \\
4 & 4 & 6
\end{bmatrix}
= \begin{bmatrix}
11 \\
18 \\
10
\end{bmatrix}
\]

The filter factors into a product of 1D filters:

Perform convolution along rows:

\[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1
\end{bmatrix}
\times
\begin{bmatrix}
2 & 3 & 3 \\
3 & 5 & 5 \\
4 & 4 & 6
\end{bmatrix}
= \begin{bmatrix}
11 \\
18 \\
10
\end{bmatrix}
\]

Followed by convolution along the remaining column:
Why is separability useful?

- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?
  - $O(n^2 m^2)$
- What if the kernel is separable?
  - $O(n^2 m)$

Properties of smoothing filters

- **Smoothing**
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter

Filtering an impulse signal

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

Convolution

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

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

\[
G = H * F
\]
**Convolution vs. Correlation**

- **Convolution** is an integral that expresses the amount of overlap of one function as it is shifted over another function.  
  - convolution is a filtering operation
- **Correlation** compares the similarity of two sets of data. Correlation computes a measure of similarity of two input signals as they are shifted by one another. The correlation result reaches a maximum at the time when the two signals match best.  
  - correlation is a measure of relatedness of two signals

**Predict the outputs using correlation filtering**

For a Gaussian or box filter, how will the outputs differ?  
If the input is an impulse signal, how will the outputs differ?

**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

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

\[ G = H \otimes F \]

**Practice with linear filters**

\[ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \]

\[ \begin{bmatrix} 0 & 0 \end{bmatrix} \]

\[ \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \]

\[ \begin{bmatrix} 0 & 0 \end{bmatrix} \]
Practice with linear filters

Original

Filtered (no change)

Practice with linear filters

Original

Shifting left by 1 pixel with correlation

Practice with linear filters

Original

?
Practice with linear filters

Original

Blur (with a box filter)

Practice with linear filters

Original

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

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

Sharpening filter: accentuates differences with local average

Practice with linear filters

Original

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

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

Filtering examples: sharpening

before

after

Slide credit: D. Lowe

Slide credit: D. Lowe

Slide credit: D. Lowe

Slide credit: K. Grauman
Sharpening
• What does blurring take away?

Let's add it back:

Unsharp mask filter

\[ f + \alpha(f - f \ast g) = (1 + \alpha)f - \alpha f \ast g = f \ast ((1 + \alpha)e - g) \]

Sharpening using Unsharp Mask Filter

Original
Filtered result

Unsharp Masking

Original
Filtered result

Slide credits: S. Lazebnik
Slide credits: C. Dyer
Other filters

Sobel

1 0 -1
2 0 -2
1 0 -1

Vertical Edge (absolute value)

Sobel

1 2 1
0 0 0
-1 -2 -1

Horizontal Edge (absolute value)

Median filters

- A **Median Filter** operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?
- Is a median filter a kind of convolution?

Median filter

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

adapted from S. Seitz

Slide credit: J. Hays

Slide credit: J. Hays

Slide credit: K. Grauman
**Median filter**

Salt and pepper noise

Plots of a row of the image

Matlab: `output im = medfilt2(im, [h w]);`

Median filtered

---

**Median filter**

- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers
  - Median filter is edge preserving

Matlab: `output im = medfilt2(im, [h w]);`

Slide credit: M. Hebert