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

Signals

- A signal is composed of low and high frequency components
- Neighboring pixels have similar brightness values
- You're within a region
- Neighboring pixels have different brightness values
- You're either at the edges or noise points

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

Spatial Filtering

- Image courtesy of Technology Review

Image courtesy of Technology Review
**Low/high frequencies vs. fine/coarse-scale details**

Original image  
Low-frequencies (coarse-scale details) boosted  
High-frequencies (fine-scale details) boosted

**Motivation: noise reduction**

- 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

**Signals – Examples**

Slide credit: S. Seitz
Gaussian noise

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

Gaussian i.i.d. (“white”) noise:
\[ \eta(x, y) \sim \mathcal{N}(0, \sigma) \]

\[
\begin{align*}
\text{>> noise = randn(size(im)).*sigma;} \\
\text{>> output = im + noise;}
\end{align*}
\]

What is the impact of the sigma?  

Slide credit: M. Hebert

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.

Adapted from: K. Grauman

Motivation: noise reduction

- What if we can’t make multiple observations?

What if there’s only one image?

Adapted from: K. Grauman

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)

Adapted from: K. Grauman
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
**Convolution warm-up**

- Same moving average operation, expressed mathematically:

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

  - every sample gets the same weight

**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

  \[ (a * b)[i] = \sum_{j} a[j]b[i-j] \]

  - 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

**Convolution and filtering**

- Can express sliding average as convolution with a **box filter**

- \( a_{\text{box}} = [..., 0, 1, 1, 1, 1, 0, ...] \)
Example: box and step

Convolution and filtering

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

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**: \(\text{filter(shift}(f)) = \text{shift(filter}(f))\)
  - 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:** $a * b = b * a$
  - Conceptually no difference between filter and signal

- **Associative:** $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$

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

- **Scalars factor out:** $k a * b = a * k b = k (a * b)$

- **Identity:** unit impulse $e = [..., 0, 1, 0, 0, ...]$, $a * e = a$

---

A gallery of filters

- **Box filter**
  - Simple and cheap

- **Tent filter**
  - Linear interpolation

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

---

**Box filter**

$$a_{box,r}[i] = \begin{cases} \frac{1}{2r+1} & |i| \leq r, \\ 0 & \text{otherwise.} \end{cases}$$

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

---

**Tent filter**

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

$$f_{tent,r}(x) = \frac{f_{tent}(x/r)}{r}.$$
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...

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

Moving Average In 2D
Moving Average In 2D

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

\[ F[x, y] \]

\[ G[x, y] \]

Slide credit: S. Seitz

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 \( 2k+1 \times 2k+1 \):

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

Attribute uniform weight to each pixel
Loop over all pixels in neighborhood around 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

Slide credit: C. Dyer

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.

Slide credit: K. Grauman
Correlation filtering

Template (mask)

Scene

Cross correlation example

Left Right

Scanline

Norm. corr.

Averaging filter

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

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

Slide credit: Fei-Fei Li

Slide credit: K. Grauman
**Smoothing by averaging**

- box filter: white = high value, black = low value

- 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): `imfilter(f, g, 0)`
    - wrap around: `imfilter(f, g, 'circular')`
    - copy edge: `imfilter(f, g, 'replicate')`
    - reflect across edge: `imfilter(f, g, 'symmetric')`
**Gaussian filter**

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

| 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 |
| 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 |

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

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

- Removes high-frequency components from the image ("low-pass filter").

---

**Gaussian filters**

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

- **Variance** of Gaussian: determines extent of smoothing

---

**Smoothing with a Gaussian**

Slide credit: K. Grauman

---

Slide credit: K. Grauman
Choosing kernel width

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

Matlab

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

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

Smoothing with a Gaussian

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

Gaussian Filters

- $\sigma = 1$ pixel
- $\sigma = 5$ pixels
- $\sigma = 10$ pixels
- $\sigma = 30$ pixels
Spatial Resolution and Color

Blurring the G Component

Blurring the R Component

Blurring the B Component
“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

Blurring L

Blurring a

Blurring b
Separability

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

Separability example

2D convolution (center location only)

\[
\begin{array}{ccc}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{array} \ast
\begin{array}{ccc}
2 & 3 & 3 \\
3 & 5 & 5 \\
4 & 4 & 6 \\
\end{array} =
\begin{array}{ccc}
11 & & \\
18 & & \\
\end{array}
\]

The filter factors into a product of 1D filters:

\[
\begin{array}{ccc}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{array} \ast
\begin{array}{ccc}
1 & & \\
1 & & \\
1 & & \\
\end{array} =
\begin{array}{ccc}
11 & & \\
18 & & \\
\end{array}
\]

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?
  - \( O(n^2 m) \)

Separability of the Gaussian filter

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

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.

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 \( \rightarrow \) 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 \( H \)?

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

\[
F[x, y] \times H[u, v] \rightarrow G[x, y]
\]

Convolution

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

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

Convolution vs. Correlation

- A **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
**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**

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

\[ G = H \circledast F \]

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

---

**Predict the outputs using correlation filtering**

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

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

---

**Practice with linear filters**

**Original**

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

**Filtered**

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

(No change)

---

**Practice with linear filters**

**Original**

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

**Filtered**

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

(No change)
Practice with linear filters

Original

? 0 0 0
0 0 1
0 0 0

Slide credit: D. Lowe

Practice with linear filters

Original

Shifted left by 1 pixel with correlation

? 0 0 0
0 0 1
0 0 0

Slide credit: D. Lowe

Practice with linear filters

Original

Blur (with a box filter)

? 1 1 1
1 1 1
1 1 1

Slide credit: D. Lowe

Practice with linear filters

Original

? 1 1 1
1 1 1
1 1 1

Slide credit: D. Lowe
Practice with linear filters

Original

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0
\end{bmatrix}
\quad \frac{1}{9}
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
\quad ?
\]

Slide credit: D. Lowe

Sharpening filter: accentuates differences with local average

Filtering examples: sharpening

before

after

Slide credit: K. Grauman

Sharpening
- What does blurring take away?
- Let's add it back:

original

smoothed (5x5)

detail

original

detail

sharpened

Slide credit: S. Lazebnik
Unsharp mask filter

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

Sharpening using Unsharp Mask Filter

Original

Filtered result

Unsharp Masking

Other filters

Sobel

Vertical Edge (absolute value)
Other filters

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

Median filtered

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

Plots of a row of the image

Salt and pepper noise

Median filtered

Slide credit: J. Hays

Median filter

Median filtered

Plots of a row of the image

Salt and pepper noise

Median filtered

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

Slide credit: K. Grauman

Slide credit: M. Hebert
Median filter

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