TOPICS FOR TODAY'S LECTURE

Image Processing:
1. Noise filtering, spectrum
2. Deconvolution, filtering
3. Edge detection
4. Median filtering

IMAGE PROCESSING [1/2]

Generalize from 1-D to 2-D:
1. Sampling theorem, Fourier transform.
2. LTI, $\delta[n]$, $h[n]$, convolution.
3. DTFT, DFT, FFT, $z$-transform.
4. $H(z)$, frequency response, filters.
5. Noise filtering, deconvolution

IMAGE PROCESSING [2/2]

DON'T Generalize usefully:
1. Laplace transform, initial conditions.
2. Poles and zeros, partial fractions.
3. ARMA difference equations.
4. Lower half of “Atlanta airport.”

Matlab: `A=imread('mandrill.tif','tif');`

IMAGES=2-D SIGNALS

Image=2-D signal $z[m,n]$.
Usually has finite support:
$0 \leq m, n \leq M-1 (M \times M)$.

Sample continuous-space $z(x,y)$:
$z[m,n]=z(x=mT,y=nT); T$ small.
A/D and D/A same as in 1-D.

2-D CONVOLUTION

Impulse: $\delta[m,n]=1$ if $m=n=0$.

PSF: Point-Spread Function:

System: $\delta[m,n] \ast h[m,n] = h[m,n]$.

That is: PSF=2-D impulse response.

Convo: $y[m,n]=h[m,n]**x[m,n]$ means:

olution: $y[m,n]=\sum_i \sum_j h[i,j] x[m-i,n-j]$.

2-D SPECTRUM [1/3]

DSFT: $X(e^{j\omega_1}, e^{j\omega_2})=
\sum_m \sum_n x[m,n] e^{-j(\omega_1 m + \omega_2 n)}$.

DSFT: Discrete Space Fourier Transform.

Note: Periodic in both $\omega_1$ and $\omega_2$.

DFT: $X_{k_1,k_2}=
\sum_m \sum_n x[m,n] e^{-j\frac{2\pi}{N}(k_1 m + k_2 n)}$.

So: $X_{k_1,k_2}=X(e^{j\omega_1}, e^{j\omega_2})|_{\omega_1=2\pi k_1/N, \omega_2=2\pi k_2/N}$.
2-D SPECTRUM [2/3]

2-D DFT: \( FX = \text{fft2}(X, M, N) \);
2-D IDFT: \( FX = \text{ifft2}(X, M, N) \);
Center DC: \( FY = \text{fftshift}(FX) \);
Display: \( \text{imagesc}(X) \), axis off
Gray: \( \text{colormap(gray)} \)
Note: Most images oversampled.

CRUDE 2-D FILTERING [1/2]

EX: \( h[m,n] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \)
Note: Separable: \( h[m,n] = h[m]h[n] \).
Then: 2-D decouples to 1-D; apply 1-D:
So: \( H(e^{j\omega_1}, e^{j\omega_2}) = H(e^{j\omega_1})H(e^{j\omega_2}) = (1 + 2\cos(\omega_1))(1 + 2\cos(\omega_2)) \).
Used as: Crude 2-D lowpass filter (notch?).

DFT 2-D FILTERING [1/2]

But: Image proc. seldom real-time.
So: Just set parts of \( \text{fft2}(X) \) to 0.
Could: Attain brick-wall 2-D LPF!
Don't: Get “ringing” in image.
Do: Window spectrum to zero.

CRUDE 2-D FILTERING [2/2]

EX: Apply above filter to noisy image:
\( Y = X + 600 \cdot \text{rand}(256, 256) \);
\( Z = \text{conv}(Y, \text{ones}(5, 5)) \);

DFT 2-D FILTERING [2/2]

EX: Set high wavenumbers to zero:
\( Y = X + 600 \cdot \text{rand}(256, 256) \);
\( FY = \text{fft2}(Y) ; L = 20 ; \)
\( FY(L:258-L,L:258-L)=0; Z=\text{real}(\text{ifft2}(FY)) \);
**IMAGE DECONVOLUTION [1/3]**

**Given:** \( y[m,n] = h[m,n] \ast x[m,n] \).

**From:** A medical or optical imaging system of some kind.

**where:** \( h[m,n] \) is known PSF.

**since:** Measure PSF \( h[m,n] \) by imaging a small bead.

**Goal:** Compute \( x[m,n] \) from \( y[m,n] \).

---

**IMAGE DECONVOLUTION [2/3]**

**Soln:** If \( y[m,n] \) is \( N \times N \):

Just:

\[
X = \text{real}(\text{ifft2}(\text{fft2}(Y) ./ \text{fft2}(H, N, N)))
\]

That is:

\[
x_{k_1, k_2} = \frac{y_{k_1, k_2}}{H_{k_1, k_2}}.
\]

**Problem:** \( H_{k_1, k_2} = 0 \) for some values of \((k_1, k_2)\)!

**Problem:** \( H_{k_1, k_2} \approx 0 \rightarrow 1/H_{k_1, k_2} \) large.

so: Any noise in \( Y_{k_1, k_2} \) will be amplified!

---

**IMAGE DECONVOLUTION [3/3]**

**Soln:** Use \( X_{k_1, k_2} = \frac{Y_{k_1, k_2}}{H_{k_1, k_2}} \)

unless: \( H_{k_1, k_2} \approx 0 \). Called a **Wiener filter**.

Does: \( \text{MIN} \left( \sum_{k_1, k_2} |Y_{k_1, k_2} - H_{k_1, k_2} X_{k_1, k_2}|^2 + \epsilon^2 |X_{k_1, k_2}|^2 \right) \).

Called: **Tikhonov regularization**.

Means: Add bias to reduce variance.

---

**IMAGE DECONVOLUTION: EX #1**

**PSF:** \( H = 0.8 \ast \text{abs}(-22:22); H = H' \ast H; Y = \text{conv2}(X, H); \)

**EX:** \( Z = \text{real}(\text{ifft2}(\text{fft2}(Y) ./ \text{fft2}(H, 300, 300))) \).

**RECONSTRUCTED IMAGE**

---

**IMAGE DECONVOLUTION: EX #2**

**Noise:** \( Y = Y + 600 \ast \text{rand}(300, 300); F_Y = \text{fft2}(Y); F_H = \text{fft2}(H, 300, 300); \)

**EX:** \( F_Z = F_Y \ast \text{conj}(F_H) ./ (\text{abs}(F_H) \ast 2 + 10); Z = \text{real}(\text{ifft2}(F_Z)); \)

---

**EDGE DETECTION: [1/3]**

**What:** Sharp change in values

**Why?** Segmentation; understanding.

**1-D SIGNAL EDGE**

**Y=X(2:10)-X(1:9)**

**Y=(X(2:10)-X(1:9))*0.6**
**EDGE DETECTION: [2/3]**

2-D: Use directional gradients for directional edge detection:

1. \( \nabla_z[i,j] = (z[i+1,j] - z[i-1,j]) + (z[i+1,j+1] - z[i-1,j+1]) + (z[i+1,j-1] - z[i-1,j-1]) \approx \frac{\partial z}{\partial x} \)

2. If \( |\nabla_z[i,j]| > \eta \), declare \( \exists \) “vertical edge.”

3. Local maxima of \( |\nabla_x(i,j)| \): “edge thinning.”

Matlab: \( \text{edge}(A, 'sobel', n, 'vertical') \)

**MEDIAN FILTERING [1/2]**

For: Impulsive = “salt-and-pepper” = shot noise (arises from bit errors).

while: Preserving image edges.

NOTE: Example of nonlinear filtering.

1-D: \( y[n] = \text{median}(x[n+2], x[n+1], x[n], x[n-1], x[n-2]) \).


2-D: Apply 1-D to rows, then columns.

Matlab: \( \text{medfilt2}(A, 5, 5) \)

**CIRCULAR SYMMETRY [1/2]**

Radial: \( H(e^{j\omega_1}, e^{j\omega_2}) = \begin{cases} 1 & \text{for } \sqrt{\omega_1^2 + \omega_2^2} < B \\ 0 & \text{for } \sqrt{\omega_1^2 + \omega_2^2} > B \\ 0 & |\omega_1|, |\omega_2| \leq \pi \end{cases} \)

Note: \( H(e^{j\omega_1}, e^{j\omega_2}) = \text{“pill-box”} \) (circularly symmetric)

Then: \( h[i, j] = \frac{B}{2\pi} J_1(\beta R) \) (circularly symmetric)

where: \( R = \sqrt{i^2 + j^2} \). (Derivation: Lim p.57).

and: \( J_1(\cdot) \) = Bessel function of 1\(^{st}\) kind of order 1.

**CIRCULAR SYMMETRY [2/2]**

Note: \( H(e^{j\omega_1}, e^{j\omega_2}) = \begin{cases} H(\sqrt{\omega_1^2 + \omega_2^2}) & \text{for } \sqrt{\omega_1^2 + \omega_2^2} < \pi \\ \text{constant} & \text{rest } 0 \leq |\omega_1|, |\omega_2| \leq \pi \end{cases} \)

\( \rightarrow h[i, j] = h(\sqrt{i^2 + j^2}) \) (circularly symmetric)

But: \( h[i, j] = h(\sqrt{i^2 + j^2}) \) does NOT

\( \rightarrow H(e^{j\omega_1}, e^{j\omega_2}) = H(\sqrt{\omega_1^2 + \omega_2^2}) \).

**EDGE DETECTION: [3/3]**

OR: Find zero-crossings of Laplacian since: \( \nabla^2 x[i,j] = \text{gradient extrema.} \)

1. \( \nabla^2 x[i,j] = x[i+1,j] + x[i-1,j] + x[i,j+1] + x[i,j-1] - 4x[i,j] \).

2. Find zero-crossings of \( \nabla^2 x[i,j] \).

3. If \( \sigma^2_{x[i,j]} = \sum_{m=-2}^{m=2} \sum_{n=-2}^{n=2} (x[m,n] - \bar{x}[m,n])^2 > \eta \) then: declare \( \exists \) “edge” at \([i,j]\).

Matlab: \( \text{edge}(A, 'log') \)