Linear Operators

- $D$ is a linear operator iff:
  \[ D(\alpha f_1 + \beta f_2) = \alpha D(f_1) + \beta D(f_2) \]

  Where $f_1$ and $f_2$ are images,
  and $\alpha$ and $\beta$ are scalar multipliers

- Not a linear operator (why?):
  \[ g = D(f) = af + b \]
Kernel Operators

- Kernel \( h \) = “small image”
  - Often 3x3 or 5x5
- Correlated with a “normal” image \( f \)
- Implied correlation (sum of products) makes a kernel an operator. A linear operator.
- Note: This use of correlation is often mislabeled as convolution in the literature.
- Any linear operator applied to an image can be approximated with correlation.

Kernels for Derivatives

- Task: estimate partial spatial derivatives
- Solution: numerical approximation
  - \( \left[ f(x + 1) - f(x) \right]/1 \)
    - Really Bad choice: not even symmetric
  - \( \left[ f(x + 1) - f(x - 1) \right]/2 \)
    - Still a bad choice: very sensitive to noise
- We need to blur away the noise (only blur orthogonal to the direction of each partial):

\[
\frac{\partial f}{\partial x} = \frac{1}{6} \left( \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \otimes f \right) \quad \text{or} \quad \frac{\partial f}{\partial x} = \frac{1}{8} \left( \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \end{bmatrix} \otimes f \right)
\]

The Sobel kernel is center-weighted.
Derivative Estimation #2: Use Function Fitting

- Think of the image as a surface
  - The gradient then fully specifies the orientation of the tangent planes at every point, and vice-versa.
- So, fit a plane to the neighborhood around a point
  - Then the plane gives you the gradient
- The concept of fitting occurs frequently in machine vision.
  Ex:
  - Gray values
  - Surfaces
  - Lines
  - Curves
  - Etc.

Derivative Estimation: Derive a 3x3 Kernel by Fitting a Plane

- If you fit by minimizing squared error, and you use symbolic notation to generalize, you get:
  - A headache
  - The kernel that we intuitively guessed earlier:

\[
\begin{bmatrix}
1 & -1 & 0 & 1 \\
6 & -1 & 0 & 1 \\
-1 & 0 & 1 & -1 \\
\end{bmatrix}
\]
Vector Representations of Images

- Also called lexicographic representations
- Linearize the image
  - Pixels have a single index (that starts at 0)

0 is the Lexicographic index

Change of coordinates

Vector listing of pixel values

Vector Representations of Kernels

- Can also linearize a kernel
- Linearization is unique for each pixel coordinate and for each image size.
  - For pixel coordinate (1,2) (i.e., pixel $F_d$) in our image:

Can combine the kernel vectors for each of the pixels into a single lexicographic kernel matrix ($H$)
- $H$ is circulant (columns are rotations of one another). Why?
Convolution in Lexicographic Representations

- Convolution becomes matrix multiplication!
- Great conceptual tool for proving theorems
- $H$ is almost never computed or written out

Basis Vectors for (Sub)Images

- Carefully choose a set of basis vectors (image patches) on which to project a sub-image (window) of size $(x,y)$
  - Is this lexicographic?
- The basis vectors with the largest coefficients are the most like this sub-image.
- If we choose meaningful basis vectors, this tells us something about the sub-image

Cartesian Basis Vectors

$\mathbf{u}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
$\mathbf{u}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
$\vdots$
$\mathbf{u}_9 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

Frei-Chen Basis Vectors

$\mathbf{u}_3 = \begin{bmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{bmatrix}$
$\mathbf{u}_4 = \begin{bmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{bmatrix}$
$\mathbf{u}_5 = \begin{bmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{bmatrix}$
$\mathbf{u}_6 = \begin{bmatrix} -1 & 0 & -1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$
$\mathbf{u}_7 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$
$\mathbf{u}_8 = \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & -1 & 2 \end{bmatrix}$
$\mathbf{u}_9 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
Edge Detection (VERY IMPORTANT)

- Image areas where:
  - Brightness changes suddenly =
  - Some derivative has a large magnitude

- Often occur at object boundaries!

- Find by:
  - Estimating partial derivatives with kernels
  - Calculating magnitude and direction from partials

\[
\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}^T = \begin{bmatrix} G_x \\ G_y \end{bmatrix}^T
\]

\[
|\nabla f| = \sqrt{G_x^2 + G_y^2} = \text{Edge Strength}
\]

\[
\angle \nabla f = \arctan \left( \frac{G_x}{G_y} \right)
\]

Then threshold the gradient magnitude image

Easy to Find
- Positive step edge
- Negative step edge
- Positive roof edge
- Negative roof edge
- Positive ramp edges
- Negative ramp edges
Noisy Positive Edge
Noisy Negative Edge

Harder To Find

Detected edges are:
- Too thick in places
- Missing in places
- Extraneous in places

Diatom image (left) and its gradient magnitude (right).

(http://bigwww.epfl.ch/theve/raz/differentials/)
Convolving w/ Fourier

- Sometimes, the fastest way to convolve is to multiply in the frequency domain.
- Multiplication is fast. Fourier transforms are not.
- The Fast Fourier Transform (FFT) helps.
- Pratt (Snyder ref. 5.33) figured out the details
  - Complex tradeoff depending on both the size of the kernel and the size of the image

*For almost all image sizes

For kernels \( \leq 7\times7 \), normal (spatial domain) convolution is fastest*.

For kernels \( \geq 13\times13 \), the Fourier method is fastest*.

---

Image Pyramids

- A series of representations of the same image
- Each is a 2:1 subsampling of the image at the next “lower level.”
  - Subsampling = averaging = down sampling
  - The subsampling happens across all dimensions!
  - For a 2D image, 4 pixels in one layer correspond to 1 pixel in the next layer.
- To make a Gaussian pyramid:
  1. Blur with Gaussian
  2. Down sample by 2:1 in each dimension
  3. Go to step 1

Increasing Scale
Scale Space

- Multiple levels like a pyramid
- Blur like a pyramid
- But don’t subsample
  - All layers have the same size
- Instead:
  - Convolve each layer with a Gaussian of variance $\sigma$.
  - $\sigma$ is the “scale parameter”
  - Only large features are visible at high scale (large $\sigma$).

Quad/Oc Trees

- Represent an image
- Homogeneous blocks
- Inefficient for storage
  - Too much overhead
- Not stable across small changes
- But: Useful for representing scale space.
Gaussian Scale Space

- Large scale = only large objects are visible
  - Increasing $\sigma \rightarrow$ coarser representations
- **Scale space causality**
  - Increasing $\sigma \rightarrow$ # extrema should not increase
  - Allows you to find “important” edges first at high scale.
- How features vary with scale tells us something about the image
- Non-integral steps in scale can be used
- Useful for representing:
  - Brightness
  - Texture
  - PDF (scale space implements clustering)

How do People Do It?

- Receptive fields
- Representable by Gabor functions
  - 2D Gaussian +
  - A plane wave
- The plane wave tends to propagate along the short axis of the Gaussian
- But also representable by Difference of offset Gaussians
  - Only 3 extrema
Canny Edge Detector

1. Use kernels to find at every point:
   - Gradient magnitude
   - Gradient direction
2. Perform Nonmaximum suppression (NMS) on the magnitude image
   - This thins edges that are too thick
   - Only preserve gradient magnitudes that are maximum compared to their 2 neighbors in the direction of the gradient

Canny Edge Detector, contd.

- Edges are now properly located and 1 pixel wide
- But noise leads to false edges, and noise+blur lead to missing edges.
  - Help this with 2 thresholds
  - A high threshold does not get many false edges, and a low threshold does not miss many edges.
  - Do a “flood fill” on the low threshold result, seeded by the high-threshold result
    - Only flood fill along isophotes