Recall: Applying Linear Filters: Convolution

For each image position $I(u,v)$:

1. Move filter matrix $H$ over image such that $H(0,0)$ coincides with current image position $(u,v)$

2. Multiply all filter coefficients $H(i,j)$ with corresponding pixel $l(u + i, v + j)$

3. Sum up results and store sum in corresponding position in new image $I'(u, v)$

Stated formally:

$\sum_{(i,j) \in R_H} I(u + i, v + j) \cdot H(i, j)$

$R_H$ is set of all pixels Covered by filter. For 3x3 filter, this is:

$I'(u, v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u + i, v + j) \cdot H(i, j)$
Recall: Mathematical Properties of Convolution

- Applying a filter as described called **linear convolution**
- For discrete 2D signal, convolution defined as:

\[
I'(u, v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i, j)
\]

Formal definition:
Sum to \( \pm \infty \)

\[
I' = I \ast H
\]
Recall: Properties of Convolution

- **Commutativity**
  \[ I * H = H * I \]

- **Linearity**
  \[ (s \cdot I) * H = I * (s \cdot H) = s \cdot (I * H) \]
  \[ (I_1 + I_2) * H = (I_1 * H) + (I_2 * H) \]

  *(notice)*
  \[ (b + I) * H \neq b + (I * H) \]

- **Associativity**
  \[ A * (B * C) = (A * B) * C \]

  *Same result if we convolve image with filter or vice versa*
  *If image multiplied by scalar Result multiplied by same scalar*
  *If 2 images added and convolve result with a kernel \( H \), Same result if we each image is convolved individually + added*
  *Order of filter application irrelevant Any order, same result*
Properties of Convolution

- **Separability**

\[ H = H_1 \ast H_2 \ast \ldots \ast H_n \]

\[ I \ast H = I \ast (H_1 \ast H_2 \ast \ldots \ast H_n) = \left((I \ast H_1) \ast H_2 \right) \ast \ldots \ast H_n \]

- If a kernel \( H \) can be separated into multiple smaller kernels

  Applying smaller kernels \( H_1, H_2 \ldots, H_n \) one by one computationally cheaper than apply 1 large kernel \( H \)

\[ H = H_1 \ast H_2 \ast \ldots \ast H_n \]

- Computationally
  - More expensive
  - Cheaper
Separability in x and y

- Sometimes we can separate a kernel into “vertical” and “horizontal” components.
- Consider the kernels

\[ H_x = [1 \ 1 \ 1 \ 1 \ 1], \quad \text{and} \quad H_y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]

Then

\[ H = H_x * H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \]
Complexity of $x/y$ Separable Kernels

- What is the number of operations for $3 \times 5$ kernel $H$?
  \[ H = H_x \ast H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \]
  \text{Ans: } 15wh

- What is the number of operations for $H_x$ followed by $H_y$?
  \text{Ans: } 3wh + 5wh = 8wh

\[ H_x = [1 \ 1 \ 1 \ 1 \ 1], \quad \text{and} \quad H_y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]
Complexity of $x/y$ Separable Kernels

- What is the number of operations for $3 \times 5$ kernel $H$?
  \textit{Ans:} $15wh$

- What is the number of operations for $H_x$ followed by $H_y$?
  \textit{Ans:} $3wh + 5wh = 8wh$

- What about $M \times M$ kernel?
  $O(M^2)$ – no separability ($M^2wh$ operations, grows quadratically!)
  $O(M^2)$ – with separability ($2Mwh$ operations, grows linearly!)
Gaussian Kernel

- **1D**

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

- **2D**

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

- 2D gaussian is just product of 1D gaussians:

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

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

\[
= g_\sigma(x) \cdot g_\sigma(y)
\]

Separable!
Separability of 2D Gaussian

- Consequently, convolution with a gaussian is separable

\[ I \ast G = I \ast G_x \ast G_y. \]

- Where \( G \) is the 2D discrete gaussian kernel;
- \( G_x \) is “horizontal” and \( G_y \) is “vertical” 1D discrete Gaussian kernels
Impulse (or Dirac) Function

- In discrete 2D case, impulse function defined as:

\[ \delta(u, v) = \begin{cases} 
1 & \text{for } u = v = 0 \\
0 & \text{otherwise.} 
\end{cases} \]

- Impulse function on image?
  - A white pixel at origin, on black background
Impulse (or Dirac) Function

- Impulse function neutral under convolution (no effect)
- Convolving an image using impulse function as filter = image
Impulse (or Dirac) Function

- Reverse case? Apply filter $H$ to impulse function
- Using fact that convolution is commutative

\[ H \ast \delta = \delta \ast H = H \]

- Result is the filter $H$
Noise

- While taking picture (during capture), noise may occur
- Noise? Errors, degradations in pixel values
- Examples of causes:
  - Focus blurring
  - Blurring due to camera motion
- Additive model for noise: $H \ast I + \text{Noise}$

- Removing noise called **Image Restoration**
- Image restoration can be done in:
  - Spatial domain, or
  - Frequency domain
Types of Noise

- Type of noise determines best types of filters for removing it!!
- **Salt and pepper noise**: Randomly scattered black + white pixels
- Also called **impulse noise, shot noise or binary noise**
- Caused by sudden sharp disturbance

Courtesy
Allasdair McAndrews

(a) Original image  (b) With added salt & pepper noise
Types of Noise

- **Gaussian Noise**: idealized form of white noise added to image, normally distributed
  \[ I + \text{Noise} \]

- **Speckle Noise**: pixel values multiplied by random noise
  \[ I(1 + \text{Noise}) \]
Types of Noise

- **Periodic Noise**: caused by disturbances of a periodic nature

- Salt and pepper, gaussian and speckle noise can be cleaned using spatial filters

- Periodic noise can be cleaned using frequency domain filtering (later)

Figure 5.3: The twins image corrupted by periodic noise

Courtesy
Allasdair McAndrews
Non-Linear Filters

- Linear filters blurs all image structures points, edges and lines, reduction of image quality (bad!)
- Linear filters thus not used a lot for removing noise
Using Linear Filter to Remove Noise?

- **Example:** Using linear filter to clean salt and pepper noise just causes smearing (not clean removal)
- Try non-linear filters?

![Example Images](https://example.com/example_images.png)

(a) 3 × 3 averaging  
(b) 7 × 7 averaging

*Courtesy Allasdair McAndrews*
Non-Linear Filters

- Pixels in filter range combined by some non-linear function
- Simplest examples of nonlinear filters: Min and Max filters

\[ I'(u, v) \leftarrow \min \{ I(u+i, v+j) \mid (i, j) \in R \} \]
\[ I'(u, v) \leftarrow \max \{ I(u+i, v+j) \mid (i, j) \in R \} \]

Before filtering

After filtering

Effect of Minimum filter

- Step Edge (shifted to right)
- Narrow Pulse (removed)
- Linear Ramp (shifted to right)
Non-Linear Filters

Original Image with Salt-and-pepper noise

Minimum filter removes bright spots (maxima) and widens dark image structures

Maximum filter (opposite effect): Removes dark spots (minima) and widens bright image structures
Median Filter

- Much better at removing noise and keeping the structures

\[ I'(u, v) \leftarrow \text{median}\{I(u+i, v+j) \mid (i, j) \in R\} \]
Illustration: Effects of Median Filter

**Isolated pixels are eliminated**

(a) Original image

(b) Median filter applied

**Thin lines are eliminated**

(c) Original image

(d) Median filter applied

**A step edge is unchanged**

**A corner is rounded off**
Effects of Median Filter

Original Image with Salt-and-pepper noise

Linear filter removes some of the noise, but not completely. Smears noise

Median filter salt-and-pepper noise and keeps image structures largely intact. But also creates small spots of flat intensity, that affect sharpness
Median Filter ImageJ Plugin

```java
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import java.util.Arrays;

public class Filter_Median3x3 implements PlugInFilter {
    final int K = 4; // filter size

    public void run(ImageProcessor orig) {
        int w = orig.getWidth();
        int h = orig.getHeight();
        ImageProcessor copy = orig.duplicate();

        // vector to hold pixels from 3x3 neighborhood
        int[] P = new int[2*K+1];

        for (int v = 1; v <= h-2; v++) {
            for (int u = 1; u <= w-2; u++) {
                // fill the pixel vector P for filter position u, v
                int k = 0;
                for (int j = -1; j <= 1; j++) {
                    for (int i = -1; i <= 1; i++) {
                        P[k] = copy.getPixel(u+i, v+j);
                        k++;
                    }
                }
                // sort pixel vector and take the center element
                Arrays.sort(P);
                orig.putPixel(u, v, P[K]);
            }
        } // end of class Filter_Median3x3
```
Weighted Median Filter

- Color assigned by median filter determined by colors of “the majority” of pixels within the filter region
- Considered robust since single high or low value cannot influence result (unlike linear average)
- Median filter assigns weights (number of “votes”) to filter positions

\[
W(i,j) = \begin{bmatrix}
1 & 2 & 1 \\
2 & 3 & 2 \\
1 & 2 & 1 \\
\end{bmatrix}
\]

- To compute result, each pixel value within filter region is inserted \(W(i,j)\) times to create extended pixel vector
- Extended pixel vector then sorted and median returned
Weighted Median Filter

Pixels within filter region

Insert each pixel within filter region \( W(i,j) \) times into extended pixel vector

Sort extended pixel vector and return median

Note: assigning weight to center pixel larger than sum of all other pixel weights inhibits any filter effect (center pixel always carries majority)!!
Weighted Median Filter

- More formally, **extended pixel vector** defined as

\[ Q = (p_0, \ldots, p_{L-1}) \] of length \[ L = \sum_{(i,j) \in R} W(i, j) \]

- For example, following weight matrix yields extended pixel vector of length 15 (sum of weights)

\[
W(i, j) = \begin{bmatrix}
1 & 2 & 1 \\
2 & 3 & 2 \\
1 & 2 & 1 \\
\end{bmatrix}
\]

- Weighting can be applied to non-rectangular filters

- Example: *cross-shaped* median filter may have weights

\[
W^+(i, j) = \begin{bmatrix}
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0 \\
\end{bmatrix}
\]
An Outlier Method of Filtering

- Algorithm by Pratt, Ref: Alasdair McAndrew, Page 116
- Median filter does sorting per pixel (computationally expensive)
- Alternate method for removing salt-and-pepper noise
  - Define noisy pixels as outliers (different from neighboring pixels by an amount > D)

- Algorithm:
  1. Choose threshold value D
  2. For given pixel, compare its value $p$ to mean $m$ of 8 neighboring pixels
  3. If $|p - m| > D$, classify pixel as noise, otherwise not
  4. If pixel is noise, replace its value with $m$; Otherwise leave its value unchanged

- Method not automatic. Generate multiple images with different values of $D$, choose the best looking one
Outlier Method Example

- Effects of choosing different values of $D$

(a) $D = 0.2$

$D$ value too small: removes noise from dark regions

(b) $D = 0.4$

$D$ value too large: removes noise from light regions

- $D$ value of 0.3 performs best
- Overall outlier method not as good as median filter

Courtesy Allasdair McAndrews
Other Non-Linear Filters

- Any filter operation that is not linear (summation), is considered linear
- Min, max and median are simple examples
- More examples later:
  - Morphological filters (Chapter 10)
  - Corner detection filters (Chapter 8)
- Also, filtering shall be discussed in frequency domain
Extending Image Along Borders

**Pad:** Set pixels outside border to a constant

**Mirror:** Pixels around image border

**Extend:** Pixels outside border take on value of closest border pixel

**Wrap:** Repeat pixels periodically along coordinate axes
Filter Operations in ImageJ

- Linear filters implemented by ImageJ plugin class \texttt{ij.plugin.filter.Convolver}
- Has several methods in addition to \texttt{run( )}

```java
import ij.plugin.filter.Convolver;
...
public void run(ImageProcessor I) {
    float[] H = { // filter array is one-dimensional!
        0.075f, 0.125f, 0.075f,
        0.125f, 0.200f, 0.125f,
        0.075f, 0.125f, 0.075f
    };
    Convolver cv = new Convolver();
    cv.setNormalize(false); // do not use filter normalization
    cv.convolve(I, H, 3, 3); // apply the filter H to I
}
```

$H(i,j) = \begin{bmatrix}
0.075 & 0.125 & 0.075 \\
0.125 & 0.2 & 0.125 \\
0.075 & 0.125 & 0.075
\end{bmatrix}$
Gaussian Filters

- `ij.plugin.filter.GaussianBlur` implements gaussian filter with radius ($\sigma$)
- Uses separable 1d gaussians

```java
import ij.plugin.filter.GaussianBlur;
...
public void run(ImageProcessor ip) {
    GaussianBlur gb = new GaussianBlur();
    double radius = 2.5;
    gb.blur(ip, radius);
}
```

Create new instance of GaussianBlur class
Blur image ip with gaussian filter of radius r
Non-Linear Filters

- A few non-linear filters (minimum, maximum and median filters implemented in `ij.plugin.filter.RankFilters`)
- Filter region is approximately circular with variable radius
- Example usage:

```java
import ij.plugin.filter.RankFilters;
...
public void run(ImageProcessor ip) {
    RankFilters rf = new RankFilters();
    double radius = 3.5;
    rf.rank(ip, radius, RankFilters.MIN); // minimum filter
    rf.rank(ip, radius, RankFilters.MAX); // maximum filter
    rf.rank(ip, radius, RankFilters.MEDIAN); // median filter
}
```
Recall: Linear Filters: Convolution

\[ I'(u, v) \leftarrow \sum_{(i, j) \in R_H} I(u + i, v + j) \cdot H(i, j) \]

\[ I'(u, v) \leftarrow \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(u + i, v + j) \cdot H(i, j) \]
Convolution as a Dot Product

- Applying a filter at a given pixel is done by taking dot-product between the image and some vector.
- Convolving an image with a filter equal to:
  - Filter each image window (moves through image)
What is an Edge?

- Edge? sharp change in brightness (discontinuities)
- Where do edges occur?
  - **Actual edges:** Boundaries between objects
  - Sharp change in brightness can also occur within object
    - Reflectance changes
    - Change in surface orientation
    - Illumination changes. E.g. Cast shadow boundary
Edge Detection

- Image processing task that finds edges and contours in images
- Edges so important that human vision can reconstruct edge lines
Characteristics of an Edge

- Edge: A sharp change in brightness
- Ideal edge is a step function in some direction
Characteristics of an Edge

- Real (non-ideal) edge is a slightly blurred step function
- Edges can be characterized by high value first derivative

\[ f'(x) = \frac{df}{dx}(x) \]
Characteristics of an Edge

- Ideal edge is a step function in certain direction.
- First derivative of $I(x)$ has a **peak** at the edge.
- Second derivative of $I(x)$ has a **zero crossing** at edge.

![Graphs showing ideal and real edges with their first and second derivatives.](image)
Slopes of Discrete Functions

- Left and right slope may not be the same
- Solution? Take average of left and right slope
Computing Derivative of Discrete Function

\[ \frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0.5 \cdot (f(u+1) - f(u-1)) \]
Finite Differences

- Forward difference (right slope)
  \[ \Delta_+ f(x) = f(x + 1) - f(x) \]

- Backward difference (left slope)
  \[ \Delta_- f(x) = f(x) - f(x - 1) \]

- Central Difference (average slope)
  \[ \Delta f(x) = \frac{1}{2} (f(x + 1) - f(x - 1)) \]
Definition: Function Gradient

- Let $f(x,y)$ be a 2D function
- **Gradient**: Vector whose direction is in direction of maximum rate of change of $f$ and whose magnitude is maximum rate of change of $f$
- Gradient is perpendicular to edge contour

$$\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]^T$$

- magnitude = $\left( \left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2 \right)^{1/2}$
- direction = $\tan^{-1} \left( \frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \right)$
Image Gradient

- Image is 2D discrete function
- Image derivatives in horizontal and vertical directions

\[
\frac{\partial I}{\partial u}(u, v) \quad \text{and} \quad \frac{\partial I}{\partial v}(u, v)
\]

- Image gradient at location \((u,v)\)

\[
\nabla I(u, v) = \begin{bmatrix}
\frac{\partial I}{\partial u}(u, v) \\
\frac{\partial I}{\partial v}(u, v)
\end{bmatrix}
\]

- Gradient magnitude

\[
|\nabla I|(u, v) = \sqrt{\left(\frac{\partial I}{\partial u}(u, v)\right)^2 + \left(\frac{\partial I}{\partial v}(u, v)\right)^2}
\]

- Magnitude is invariant under image rotation, used in edge detection
Derivative Filters

- Recall that we can compute derivative of discrete function as
  \[
  \frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0.5 \cdot (f(u+1) - f(u-1))
  \]

- Can we make linear filter that computes central differences
  \[
  H_x^D = [-0.5 \  0 \  0.5] = 0.5 \cdot [-1 \  0 \  1]
  \]
Finite Differences as Convolutions

- Forward difference
  \[ \Delta_+ f(x) = f(x + 1) - f(x) \]

- Take a convolution kernel
  \[ H = [0 \quad -1 \quad 1] \]

\[ \Delta_+ f = f * H \]
Finite Differences as Convolutions

- Central difference
  \[ \Delta f(x) = \frac{1}{2} (f(x + 1) - f(x - 1)) \]

- Convolution kernel is:
  \[ H = \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} \]
  \[ \Delta f(x) = f \ast H \]

- Notice: Derivative kernels sum to zero
$x$-Derivative of Image using Central Difference

$\ast \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} =$
y-Derivative of Image using Central Difference
Derivative Filters

Gradient slope in horizontal direction

\[ H_x^D = \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \]

Gradient slope in vertical direction

\[ H_y^D = \begin{bmatrix} -0.5 \\
0 \\
0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\
0 \\
1 \end{bmatrix} \]

Magnitude of gradient

A synthetic image

(a) (b) (c) (d)
Edge Operators

- Approximating local gradients in image is basis of many classical edge-detection operators
- Main differences?
  - Type of filter used to estimate gradient components
  - How gradient components are combined
- We are typically interested in
  - Local edge direction
  - Local edge magnitude
Partial Image Derivatives

- Partial derivatives of images replaced by finite differences
  \[ \Delta_x f = f(x, y) - f(x - 1, y) \]
  \[ \Delta_y f = f(x, y) - f(x, y - 1) \]

- Alternatives are:
  \[ \Delta_{2x} f = f(x + 1, y) - f(x - 1, y) \]
  \[ \Delta_{2y} f = f(x, y + 1) - f(x, y - 1) \]

- Robert’s gradient
  \[ \Delta_+ f = f(x + 1, y + 1) - f(x, y) \]
  \[ \Delta_- f = f(x, y + 1) - f(x + 1, y) \]
Using Averaging with Derivatives

- Finite difference operator is sensitive to noise
- Derivatives more robust if derivative computations are averaged in a neighborhood
- Prewitt operator: derivative in $x$, then average in $y$

$$
H^P_x = \frac{1}{3} \left[ \begin{array}{c} 1 \\ 1 \\ 1 \end{array} \right] \ast \left[ \begin{array}{ccc} 0.5 & 0 & -0.5 \end{array} \right] = \frac{1}{6} \left[ \begin{array}{ccc} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{array} \right]
$$

- $y$-derivative kernel, $H^P_y$ defined similarly

Note: Filter kernel is flipped in convolution
Sobel Operator

- Similar to Prewitt, but averaging kernel is higher in middle

\[ H_x^S = \frac{1}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \ast \begin{bmatrix} 0.5 & 0 & -0.5 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \]

\[ H_y^S = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \ast \begin{bmatrix} 0.5 \\ 0 \\ -0.5 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \]

Average in x direction

Derivative in y direction

Note: Filter kernel is flipped in convolution
Prewitt and Sobel Edge Operators

- **Prewitt Operator**

  \[ H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \]

  Written in separable form

  \[ H_x^P = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \ast \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^P = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \ast \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \]

- **Sobel Operator**

  \[ H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \]
Improved Sobel Filter

- Original Sobel filter relatively inaccurate
- Improved versions proposed by Jahne

\[
H_x^{S'} = \frac{1}{32} \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \quad \text{and} \quad H_y^{S'} = \frac{1}{32} \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}
\]
Prewitt and Sobel Edge Operators
Scaling Edge Components

- Estimates of local gradient components obtained from filter results by appropriate scaling

\[ \nabla I(u, v) \approx \frac{1}{6} \cdot \begin{bmatrix} (I \ast H_x^P)(u, v) \\ (I \ast H_y^P)(u, v) \end{bmatrix} \]

Scaling factor for Prewitt operator

\[ \nabla I(u, v) \approx \frac{1}{8} \cdot \begin{bmatrix} (I \ast H_x^S)(u, v) \\ (I \ast H_y^S)(u, v) \end{bmatrix} \]

Scaling factor for Sobel operator
Gradient-Based Edge Detection

- Compute image derivatives by convolution
  \[ D_x(u, v) = H_x * I \quad \text{and} \quad D_y(u, v) = H_y * I \]

- Compute edge gradient magnitude
  \[ E(u, v) = \sqrt{(D_x(u, v))^2 + (D_y(u, v))^2} \]

- Compute edge gradient direction
  \[ \Phi(u, v) = \tan^{-1}\left(\frac{D_y(u, v)}{D_x(u, v)}\right) = \text{ArcTan}(D_x(u, v), D_y(u, v)) \]

![Diagram of Gradient-based edge detection process]

Scaled Filter results

Typical process of Gradient based edge detection
Gradient-Based Edge Detection

- After computing gradient magnitude and orientation then what?
- Mark points where gradient magnitude is large wrt neighbors
Non-Maxima Suppression

- Retain a point as an edge point if:
  - Its gradient magnitude is higher than a threshold
  - Its gradient magnitude is a local maxima in gradient direction

Simple thresholding will compute thick edges
Non-Maxima Suppression

- A maxima occurs at $q$, if its magnitude is larger than those at $p$ and $r$
Roberts Edge Operators

- Estimates directional gradient along 2 image diagonals
- Edge strength $E(u,v)$: length of vector obtained by adding 2 orthogonal gradient components $D_1(u,v)$ and $D_2(u,v)$

![Diagram showing Roberts Edge Operators]

- Filters for edge components

$$H_1^R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{and} \quad H_2^R = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$
Roberts Edge Operators

- Diagonal gradient components produced by 2 Robert filters

\[ D_1 = I \ast H_1^R \]
\[ D_2 = I \ast H_2^R \]
Compass Operators

• Linear edge filters involve trade-off

Sensitivity to Edge magnitude \[\uparrow\] \[=\] \[\downarrow\] Sensitivity to orientation

• Example: Prewitt and Sobel operators detect edge magnitudes but use only 2 directions (insensitive to orientation)

• Solution? Use many filters, each sensitive to narrow range of orientations (compass operators)
Compass Operators

- Edge operators proposed by Kirsh uses 8 filters with orientations spaced at 45 degrees.

\[
\begin{align*}
H_0^K &= \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} & H_4^K &= \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \\
H_1^K &= \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} & H_5^K &= \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \\
H_2^K &= \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} & H_6^K &= \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \\
H_3^K &= \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} & H_7^K &= \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}
\end{align*}
\]

Need only to compute 4 filters since \(H_4 = -H_0\), etc.
Compass Operators

- Edge strength $E^K$ at position $(u,v)$ is max of the 8 filters

\[ E^K(u, v) \triangleq \max(D_0(u, v), D_1(u, v), \ldots D_7(u, v)) \]
\[ = \max(|D_0(u, v)|, |D_1(u, v)|, |D_2(u, v)|, |D_3(u, v)|) \]

- Strongest-responding filter also determines edge orientation at a position $(u,v)$

\[ \Phi^K(u, v) \triangleq \frac{\pi}{4} \quad \text{with} \quad j = \arg\max_{0 \leq i \leq 7} D_i(u, v) \]
Edge operators in ImageJ

- ImageJ implements Sobel operator
- Can be invoked via menu `Process -> Find Edges`
- Also available through method `void findEdges()` for objects of type `ImageProcessor`
References

- University of Utah, CS 4640: Image Processing Basics, Spring 2012
- Rutgers University, CS 334, Introduction to Imaging and Multimedia, Fall 2012