Recall: Histogram Equalization

- Adjust 2 different images to make their histograms (intensity distributions) similar
- Apply a point operation that changes histogram of modified image into uniform distribution
Recall: Equalization Transformation Function
Linear Histogram Equalization

- Histogram cannot be made exactly flat – peaks cannot be increased or decreased by point operations.
- Following point operation makes histogram as flat as possible:
  (assuming $M \times N$ image and pixels in range $[0, K - 1]$)

Point operation that returns Linear equalized value of $a$

$$f_{eq}(a) = \left[ H(a) \cdot \frac{K - 1}{MN} \right]$$

Cumulative Histogram: $\Sigma$ how many times intensity $a$ occurs
Effects of Linear Histogram Equalization

Original Image \( I \)

Image \( I' \) after Linear Equalization

Original histogram

Histogram after Linear Equalization

Cumulative Histogram

Cumulative Histogram After Linear Equalization
Sample Linear Equalization Code

```java
public void run(ImageProcessor ip) {
    int w = ip.getWidth();
    int h = ip.getHeight();
    int M = w * h; // total number of image pixels
    int K = 256; // number of intensity values

    // compute the cumulative histogram:
    int[] H = ip.getHistogram();
    for (int j = 1; j < H.length; j++) {
        H[j] = H[j-1] + H[j];
    }

    // equalize the image:
    for (int v = 0; v < h; v++) {
        for (int u = 0; u < w; u++) {
            int a = ip.get(u, v);
            int b = H[a] * (K-1) / M;
            ip.set(u, v, b);
        }
    }
}
```

Obtain histogram of image \( ip \)
Compute cumulative histogram in place
Get intensity value at \((u,v)\)
Equalize pixel intensity

Cumulative Histogram: \( \sum \) how many times intensity \( a \) occurs
Histogram Specification

- Real images never show uniform distribution (unnatural)
- Most real images, distribution of pixel intensities is gaussian
- **Histogram specification**
  - modifies an image’s histogram into an arbitrary intensity distribution (may not be uniform)
- Image 1’s histogram can also be used as target for image 2
  - Why? Makes images taken by 2 different cameras to appear as if taken by same camera
Images and Probability

Histograms can be interpreted as probabilities.

**Question:** If I pick a pixel from an image at random, what is the probability that the pixel has intensity $i$?

**Answer:**
$$P(I(u, v) = i) = \frac{\text{# pixels with value } i}{\text{total # pixels}}$$

Or, in terms of the histogram $h$:
$$P(I(u, v) = i) = \frac{h(i)}{wh}$$
Histogram Specification

- Find a mapping such that distribution of $a$ matches some reference distribution. i.e.

$$a' = f_{hs}(a)$$

Mapping function: maps distribution on right to equivalent point (same height) On distribution on left

To convert original image $I_A$ into $I_{A'}$ such that

$$P_{A'}(i) \approx P_R(i) \quad \text{for} \quad 0 \leq i < K$$

i.e. $a$ and $a'$ have same height ($b$) on different CDF distributions

$$f_{hs}(a) = a' = P_R^{-1}(P_A(a))$$
Adjusting Linear Distribution Piecewise

- In practice, reference distribution may be specified as a *piecewise linear* function

\[ \mathcal{L} = [\langle a_0, q_0 \rangle, \langle a_1, q_1 \rangle, \ldots \langle a_k, q_k \rangle, \ldots \langle a_N, q_N \rangle] \]

- 2 endpoints are fixed: \( \langle 0, q_0 \rangle \) and \( \langle K - 1, 1 \rangle \)
Adjusting Linear Distribution Piecewise

For each segment, linearly interpolate to find any value

\[ P_L(i) = \begin{cases} 
q_m + (i-a_m) \cdot \frac{(q_{m+1} - q_m)}{(a_{m+1} - a_m)} & \text{for } 0 \leq i < K-1 \\
1 & \text{for } i = K-1 
\end{cases} \]

We also need the inverse mapping

\[ P_L^{-1}(b) = \begin{cases} 
0 & \text{for } 0 \leq b < P_L(0) \\
 a_n + (b-q_n) \cdot \frac{(a_{n+1} - a_n)}{(q_{n+1} - q_n)} & \text{for } P_L(0) \leq b < 1 \\
K-1 & \text{for } b \geq 1 
\end{cases} \]

\[ n = \max\{j \in \{0, \ldots N-1\} \mid q_j \leq b\} \]
Adjusting Linear Distribution Piecewise

1: \textbf{PiecewiseLinearHistogram}(h_A, \mathcal{L}_R)
\hspace{1em} h_A: \text{histogram of the original image.}
\hspace{1em} \mathcal{L}_R: \text{reference distribution function, given as a sequence of } N + 1
\hspace{1em} \text{control points } \mathcal{L}_R = [(a_0, q_0), (a_1, q_1), \ldots (a_N, q_N)], \text{ with } 0 \leq a_k < K
\hspace{1em} \text{and } 0 \leq q_k \leq 1.

2: \hspace{1em} \text{Let } K \leftarrow \text{Size}(h_A)
3: \hspace{1em} \text{Let } P_A \leftarrow \text{CDF}(h_A) \hspace{1em} \triangleright \text{cdf for } h_A \text{ (Alg. 5.1)}
4: \hspace{1em} \text{Create a table } f_{hs}[\cdot] \text{ of size } K \hspace{1em} \triangleright \text{mapping function } f_{hs}
5: \hspace{1em} \text{for } a \leftarrow 0 \ldots (K-1) \text{ do}
6: \hspace{1em} \hspace{1em} \hspace{1em} b \leftarrow P_A(a)
7: \hspace{1em} \hspace{1em} \hspace{1em} \text{if } (b \leq q_0) \text{ then}
8: \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} a' \leftarrow 0
9: \hspace{1em} \hspace{1em} \hspace{1em} \text{else if } (b \geq 1) \text{ then}
10: \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} a' \leftarrow K-1
11: \hspace{1em} \hspace{1em} \hspace{1em} \text{else}
12: \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} n \leftarrow N-1
13: \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \text{while } (n \geq 0) \land (q_n > b) \text{ do} \hspace{1em} \triangleright \text{find line segment in } \mathcal{L}_R
14: \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} n \leftarrow n - 1
15: \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} a' \leftarrow a_n + (b - q_n) \cdot \frac{(a_{n+1} - a_n)}{(q_{n+1} - q_n)} \hspace{1em} \triangleright \text{see Eqn. (5.23)}
16: \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} f_{hs}[a] \leftarrow a'
17: \hspace{1em} \hspace{1em} \hspace{1em} \text{return } f_{hs}.
Adjusting Linear Histogram Piecewise

Original Image

Modified Image

Reference Distribution (piecewise linear)

(a) $I_A$

(b) $I_{A'}$

(c) $h_R$

(d) $h_A$

(e) $h_{A'}$

(f) $P_R$

(g) $P_A$

(h) $P_{A'}$
Histogram Matching

- Prior method needed reference distribution to be invertible

\[ f_{hs}(a) = a' = P_R^{-1}(P_A(a)) \]

- What if reference histogram is not invertible?
- For example not invertible if histogram has some intensities that occur with probability 0? i.e. \( p(k) = 0 \)
- Use different method called **histogram matching**
Histogram Matching

- Given two images $I_A$ and $I_B$, we want to make their intensity profiles look as similar as possible.
- How?
  - “Match” their cumulative histograms $H_A$ and $H_B$
- Works well for images with similar content.
- Looks bad for images with different content.
Adjusting to a Given Histogram

\[ f_{hs}(a) = a' = \min\{ j \mid (0 \leq j < K) \land (P_A(a) \leq P_R(j)) \} \]
Adjusting to a Given Histogram

1: MATCHHISTOGRAMS(h_A, h_R)
   h_A: histogram of the target image
   h_R: reference histogram (of same size as h_A)

2: Let K ← Size(h_A)
3: Let P_A ← CDF(h_A) \( \triangleright \) cdf for h_A (Alg. 5.1)
4: Let P_R ← CDF(h_R) \( \triangleright \) cdf for h_R (Alg. 5.1)
5: Create a table \( f_{hs}[\_] \) of size K \( \triangleright \) pixel mapping function \( f_{hs} \)
6: for \( a \leftarrow 0 \ldots (K-1) \) do
7:     \( j \leftarrow K-1 \)
8:     repeat
9:         \( f_{hs}[a] \leftarrow j \)
10:        \( j \leftarrow j - 1 \)
11:    while \((j \geq 0) \land (P_A(a) \leq P_R(j)) \)
12: return \( f_{hs} \).
Adjusting to a Given Histogram

Original Image

Gaussian ($\sigma = 50$)

Gaussian ($\sigma = 100$)

Reference Histogram

$H(x)$

Cumulative Reference Histogram

$H^r(x)$

Original histogram

CDF of original histogram

Original histogram after matching

CDF of original histogram after matching
Adjusting to a Given Histogram

Target Image

Reference Image

Modified Image

(a) $I_A$

(b) $I_R$

(c) $I_{A'}$

(d) $h_A$

(e) $h_R$

(f) $h_{A'}$

(g) $H_A$

(h) $H_R$

(i) $H_{A'}$
Gamma Correction

- Different camera sensors
  - Have different responses to light intensity
  - Produce different electrical signals for same input

- How do we ensure there is consistency in:
  a) Images recorded by different cameras for given light input
  b) Light emitted by different display devices for same image?
Gamma Correction

- What is the relation between:
  - **Camera**: Light on sensor vs. "intensity" of corresponding pixel
  - **Display**: Pixel intensity vs. light from that pixel
- Relation between pixel value and corresponding physical quantity is usually complex, nonlinear
- An approximation?
What is Gamma?

- Originates from analog photography
- **Exposure function:** relationship between:
  - logarithmic light intensity vs. resulting film density.
- **Gamma:** slope of linear range of the curve
- The same in TV broadcasting

![Diagram showing the relationship between log of light intensity and film density](image)
What is Gamma?

- **Gamma function**: a good approximation of exposure curve
- Inverse of a Gamma function is another gamma function with
  \[ \tilde{\gamma} = 1/\gamma \]
- Gamma of CRT and LCD monitors:
- 1.8-2.8 (typically 2.4)

\[ b = s^{1/\gamma_c} = \left(B^{\gamma_c}\right)^{1/\gamma_c} = B^{(\gamma_c \frac{1}{\gamma_c})} = B^{1} \]

Output signal Raised by gamma

Correct output signal By dividing by 1/ gamma (called Gamma correction)
Gamma Correction

- Obtain a measurement $b$ proportional to original light intensity $B$ by applying inverse gamma function.
- Gamma correction is important to achieve a device independent representation.

$$b = s^{{\bar{\gamma}_c}} \approx B$$

$\gamma_c$: Gamma value for the camera.

$\gamma_c'$: Gamma value for the corrected signal.

$f_{gc}(s, \bar{\gamma}_c)$: Gamma correction function.

$B$: Original light intensity.

$s$: Signal value.

$\bar{\gamma}_c$: Corrected gamma value.
Gamma Correction

\[ \bar{\gamma}_c = \frac{1}{1.3} \]

\[ \bar{\gamma}_m = \frac{1}{2.6} \]

\[ \bar{\gamma}_p = \frac{1}{3.0} \]

\[ \gamma_c = 1.3 \]

\[ \gamma_m = 2.6 \]

\[ \gamma_p = 3.0 \]
Gamma Correction Code

```java
public void run(ImageProcessor ip) {
    // works for 8-bit images only
    int K = 256;
    int aMax = K - 1;
    double GAMMA = 2.8;

    // create a lookup table for the mapping function
    int[] Fgc = new int[K];

    for (int a = 0; a < K; a++) {
        double aa = (double) a / aMax; // scale to [0,1]
        double bb = Math.pow(aa, GAMMA); // gamma function
        // scale back to [0,255]:
        int b = (int) Math.round(bb * aMax);
        Fgc[a] = b;
    }
    ip.applyTable(Fgc); // modify the image ip
}
```

Compute corrected intensity and store in lookup table `fgc`
## Point Operations in ImageJ

<table>
<thead>
<tr>
<th>Function</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void abs()</code></td>
<td>$I'(u,v) \leftarrow</td>
</tr>
<tr>
<td><code>void add(int p)</code></td>
<td>$I'(u,v) \leftarrow I(u,v) + p$</td>
</tr>
<tr>
<td><code>void gamma(double g)</code></td>
<td>$I'(u,v) \leftarrow (I(u,v)/255)^g \cdot 255$</td>
</tr>
<tr>
<td><code>void invert(int p)</code></td>
<td>$I'(u,v) \leftarrow 255 - I(u,v)$</td>
</tr>
<tr>
<td><code>void log()</code></td>
<td>$I'(u,v) \leftarrow \log_{10}(I(u,v))$</td>
</tr>
<tr>
<td><code>void max(double s)</code></td>
<td>$I'(u,v) \leftarrow \max(I(u,v), s)$</td>
</tr>
<tr>
<td><code>void min(double s)</code></td>
<td>$I'(u,v) \leftarrow \min(I(u,v), s)$</td>
</tr>
<tr>
<td><code>void multiply(double s)</code></td>
<td>$I'(u,v) \leftarrow \text{round}(I(u,v) \cdot s)$</td>
</tr>
<tr>
<td><code>void sqr()</code></td>
<td>$I'(u,v) \leftarrow I(u,v)^2$</td>
</tr>
<tr>
<td><code>void sqrt()</code></td>
<td>$I'(u,v) \leftarrow \sqrt{I(u,v)}$</td>
</tr>
</tbody>
</table>
# ImageJ Operations involving 2 images

<table>
<thead>
<tr>
<th>Operation</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>$ip1 \leftarrow ip1 + ip2$</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>$ip1 \leftarrow (ip1 + ip2) / 2$</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>$ip1 \leftarrow</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>$ip1 \leftarrow ip1 / ip2$</td>
</tr>
<tr>
<td>MAX</td>
<td>$ip1 \leftarrow \max(ip1, ip2)$</td>
</tr>
<tr>
<td>MIN</td>
<td>$ip1 \leftarrow \min(ip1, ip2)$</td>
</tr>
<tr>
<td>MULTIPLY</td>
<td>$ip1 \leftarrow ip1 \cdot ip2$</td>
</tr>
<tr>
<td>SUBTRACT</td>
<td>$ip1 \leftarrow ip1 - ip2$</td>
</tr>
</tbody>
</table>
Example: Alpha Blending

\[ I'(u, v) \leftarrow \alpha \cdot I_{BG}(u, v) + (1 - \alpha) \cdot I_{FG}(u, v) \]
import ij.IJ;
import ij.ImagePlus;
import ij.WindowManager;
import ij.gui.GenericDialog;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;

public class Alpha_Blending implements PlugInFilter {

    static double alpha = 0.5; // transparency of foreground image
    ImagePlus fgIm = null; // foreground image

    public int setup(String arg, ImagePlus imp) {
        return DOES_8G;
    }

    public void run(ImageProcessor bgIp) { // background image
        if (runDialog()) {
            ImageProcessor fgIp = fgIm.getProcessor().convertToByte(false);
            fgIp = fgIp.duplicate();
            fgIp.multiply(1-alpha);
            bgIp.multiply(alpha);
            bgIp.copyBits(fgIp, 0, 0, Blitter.ADD);
        }
    }

    // continued ...
What Is Image Enhancement?

Image enhancement makes images more useful by:

- Highlighting interesting detail in images
- Removing noise from images
- Making images more visually appealing
Image Enhancement Examples
Image Enhancement Examples (cont...)

Image Enhancement Examples (cont...)
Image Enhancement Examples (cont...)
Spatial & Frequency Domains

There are two broad categories of image enhancement techniques

- Spatial domain techniques
  - Direct manipulation of image pixels (intensity values)
- Frequency domain techniques
  - Manipulation of Fourier transform or wavelet transform of an image

First spatial domain techniques
Later: frequency domain techniques
What is a Filter?

- Capabilities of point operations are limited
- **Filters**: combine *pixel’s value + values of neighbors*
- **E.g blurring**: Compute average intensity of block of pixels

- Combining multiple pixels needed for certain operations:
  - Blurring, Smoothing
  - Sharpening
What Point Operations Can’t Do

- Example: sharpening
What Point Operations Can’t Do

- Other cool artistic patterns by combining pixels
Definition: Spatial Filter

- An image operation that combines each pixel’s intensity $I(u, v)$ with that of neighboring pixels.
- E.g.: average/weighted average of group of pixels.
Example: Average (Mean) of 3x3 Neighborhood

Blurring: Replace each pixel with AVERAGE Intensity of pixel + neighbors
Smoothing an Image by Averaging

- Replace each pixel by average of pixel + neighbors
- For 3x3 neighborhood:

\[ I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9} \]
Smoothing an Image by Averaging

\[
I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}
\]

\[
I'(u, v) \leftarrow \frac{1}{9} \cdot [I(u-1, v-1) + I(u, v-1) + I(u+1, v-1) + I(u-1, v) + I(u, v) + I(u+1, v) + I(u-1, v+1) + I(u, v+1) + I(u+1, v+1)]
\]

\[
I'(u, v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^{1} \sum_{i=-1}^{1} I(u + i, v + j)
\]
Example: Smoothing Spatial Filtering

The above is repeated for every pixel in the original image to generate the smoothed image.
Smoothing an Image by Averaging

- Many possible filter parameters (size, weights, function, etc)
- **Filter size** (size of neighborhood): 3x3, 5x5, 7x7, ..., 21x21, ...
- **Filter shape**: not necessarily square. Can be rectangle, circle, etc
- **Filter weights**: May apply unequal weighting to different pixels
- **Filters function**: can be linear (a weighted summation) or nonlinear

Previous example: Filter size: 3x3
The Filter Matrix

$I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$

$I'(u, v) \leftarrow \frac{1}{9} \cdot [I(u-1, v-1) + I(u, v-1) + I(u+1, v-1) + I(u-1, v) + I(u, v) + I(u+1, v) + I(u-1, v+1) + I(u, v+1) + I(u+1, v+1)]$

$H(i, j) = \begin{bmatrix}
\frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9}
\end{bmatrix} = \frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}$

Filter operation can be expressed as a matrix

Example: averaging filter
Example: What does this Filter Do?

Identity function (leaves image alone)
What Does this Filter Do?

Mean (averages neighborhood)
Mean Filters: Effect of Filter Size

Original

7 × 7

15 × 15

41 × 41
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 \( I(u + i, v + j) \)

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

Stated formally:

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

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

\[
I'(u, v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u + i, v + j) \cdot H(i, j)
\]
Computing Filter Operation

- Filter matrix $H$ moves over each pixel in original image $I$ to compute corresponding pixel in new image $I'$
- Cannot overwrite new pixel value in original image $I$ Why?

Version A

1. Copy original image $I$
2. Filter $I$ to get intermediate image
3. Store results $I'$ in intermediate image, then copy back to replace $I$

Version B

1. Copy original image $I$
2. Filter $I$ to get intermediate image
3. Copy intermediate image, use it as source, then store results $I'$ to replace original image
Simple 3x3 Averaging Filter ("Box" Filter)

```java
1 import ij.*;
2 import ij.plugin.filter.PlugInFilter;
3 import ij.process.*;
4
5 public class Filter_Average3x3 implements PlugInFilter {
6     ...
7     public void run(ImageProcessor orig) {
8         int w = orig.getWidth();
9         int h = orig.getHeight();
10        ImageProcessor copy = orig.duplicate();
11
12         for (int v = 1; v <= h-2; v++) {
13             for (int u = 1; u <= w-2; u++) {
14                 //compute filter result for position (u,v)
15                 int sum = 0;
16                 for (int j = -1; j <= 1; j++) {
17                     for (int i = -1; i <= 1; i++) {
18                         int p = copy.getPixel(u+i, v+j);
19                         sum = sum + p;
20                     }
21                 }
22                 int q = (int) Math.round(sum/9.0);
23                 orig.putPixel(u, v, q);
24             }
25         }
26     }
27 } // end of class Filter_Average3x3
```

No explicit filter matrix since all coefficients are the same (1/9)

No clamping required

Make copy of original image to use as source

Loop over all pixels in image

Filter computation by adding current pixel's neighbors

Store result back in original image
Weighted Smoothing Filters

- More effective smoothing filters can be generated by allowing different pixels in the neighbourhood different weights in the averaging function.
  - Pixels closer to central pixel more important
  - Often referred to as a weighted averaging

<table>
<thead>
<tr>
<th></th>
<th>1/16</th>
<th>2/16</th>
<th>1/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/16</td>
<td>4/16</td>
<td>2/16</td>
<td></td>
</tr>
<tr>
<td>1/16</td>
<td>2/16</td>
<td>1/16</td>
<td></td>
</tr>
</tbody>
</table>

Weighted averaging filter
Another Smoothing Filter

```java
public void run(ImageProcessor orig) {
    int w = orig.getWidth();
    int h = orig.getHeight();
    // 3 x 3 filter matrix
    double[][] filter = {
        {0.075, 0.125, 0.075},
        {0.125, 0.200, 0.125},
        {0.075, 0.125, 0.075}
    };
    ImageProcessor copy = orig.duplicate();
    for (int v = 1; v <= h-2; v++) {
        for (int u = 1; u <= w-2; u++) {
            // compute filter result for position (u, v)
            double sum = 0;
            for (int j = -1; j <= 1; j++) {
                for (int i = -1; i <= 1; i++) {
                    int p = copy.getPixel(u+i, v+j);
                    // get the corresponding filter coefficient:
                    double c = filter[j+1][i+1];
                    sum = sum + c * p;
                }
            }
            int q = (int) Math.round(sum);
            orig.putPixel(u, v, q);
        }
    }
}
```

Use real filter matrix with coefficients
Apply bell-shaped function $H(i,j)$

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

Bell-shaped function $H(i,j)$?
- More weight applied to center

Apply filter
Store result back in original
Integer Coefficients

- Instead of floating point coefficients, more efficient, simpler to use:

\[ H(i, j) = \begin{bmatrix}
0.075 & 0.125 & 0.075 \\
0.125 & 0.200 & 0.125 \\
0.075 & 0.125 & 0.075 \\
\end{bmatrix} = \frac{1}{40} \begin{bmatrix}
3 & 5 & 3 \\
5 & 8 & 5 \\
3 & 5 & 3 \\
\end{bmatrix} \]

\[ H(i, j) = s \cdot H'(i, j) \]
Example: 5x5 Filter in Adobe Photoshop

\[ I'(u, v) \leftarrow \text{Offset} + \frac{1}{\text{Scale}} \sum_{j=-2}^{j=2} \sum_{i=-2}^{i=2} I(u+i, v+j) \cdot H(i, j) \]

- Integer filter coefficients
- If resulting pixel value is negative, offset shifts it into visible range
- Scaling factor for coefficients
Computation Range

- For a filter of size \((2K+1) \times (2L+1)\), if image size is \(M \times N\), filter is computed over the range:

\[
K \leq u' \leq (M - K - 1) \quad \text{and} \quad L \leq v' \leq (N - L - 1)
\]

Filter can only be applied at image locations \((u, v)\) where filter matrix \(H\) is fully contained in the image.
public void run(ImageProcessor orig) {
    int M = orig.getWidth();
    int N = orig.getHeight();

    // filter matrix of size \((2K+1) \times (2L+1)\)
    int[][] filter = {
        {0,0,1,1,1,0,0},
        {0,1,1,1,1,1,0},
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,0},
        {0,1,1,1,1,1,0},
        {0,0,1,1,1,0,0}
    };
    double s = 1.0/23; // sum of filter coefficients is 23

    int K = filter[0].length/2;
    int L = filter.length/2;

    ImageProcessor copy = orig.duplicate();

    for (int v = L; v <= N-L-1; v++) {
        for (int u = K; u <= M-K-1; u++) {
            // compute filter result for position \(u,v\)
            int sum = 0;
            for (int j = -L; j <= L; j++) {
                for (int i = -K; i <= K; i++) {
                    int p = copy.getPixel(u+i, v+j);
                    int c = filter[j+L][i+K];
                    sum = sum + c * p;
                }
            }
            int q = (int) Math.round(s * sum);
            if (q < 0) q = 0;
            if (q > 255) q = 255;
            orig.putPixel(u, v, q);
        }
    }
}
What to do at image boundaries?

a) Crop
What to do at image boundaries?

a) Crop
b) Pad
What to do at image boundaries?

a) Crop
b) Pad
c) Extend
What to do at image boundaries?

a) Crop
b) Pad
c) Extend
d) Wrap
Linear Filters: Smoothing Filters

- 2 main classes of linear filters:
  - **Smoothing**: +ve coefficients (weighted average). E.g. box, gaussian
  - **Difference** filters: +ve and –ve weights. E.g. Laplacian

(a) Box
(b) Gaussian
(c) Laplacian
Gaussian Filter

\[ G_\sigma(r) = e^{-\frac{r^2}{2\sigma^2}} \quad \text{or} \quad G_\sigma(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \]

- **where**
  - \( \sigma \) is width (standard deviation)
  - \( r \) is distance from center
Difference Filters

- **Coefficients**: some +ve, some negative
- Example: Laplacian filter
- Computation is difference

\[ \sum (+ve\ coefficients) - \sum (-ve\ coefficients) \]

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

Laplacian filter
Mathematical Properties of Convolution

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

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

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

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

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

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

**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 each image is convolved individually + added

**Order of filter application irrelevant**
Any order, same result
References

  - Histograms (Ch 4)
  - Point operations (Ch 5)
  - Filters (Ch 6)
- University of Utah, CS 4640: Image Processing Basics, Spring 2012
- Rutgers University, CS 334, Introduction to Imaging and Multimedia, Fall 2012