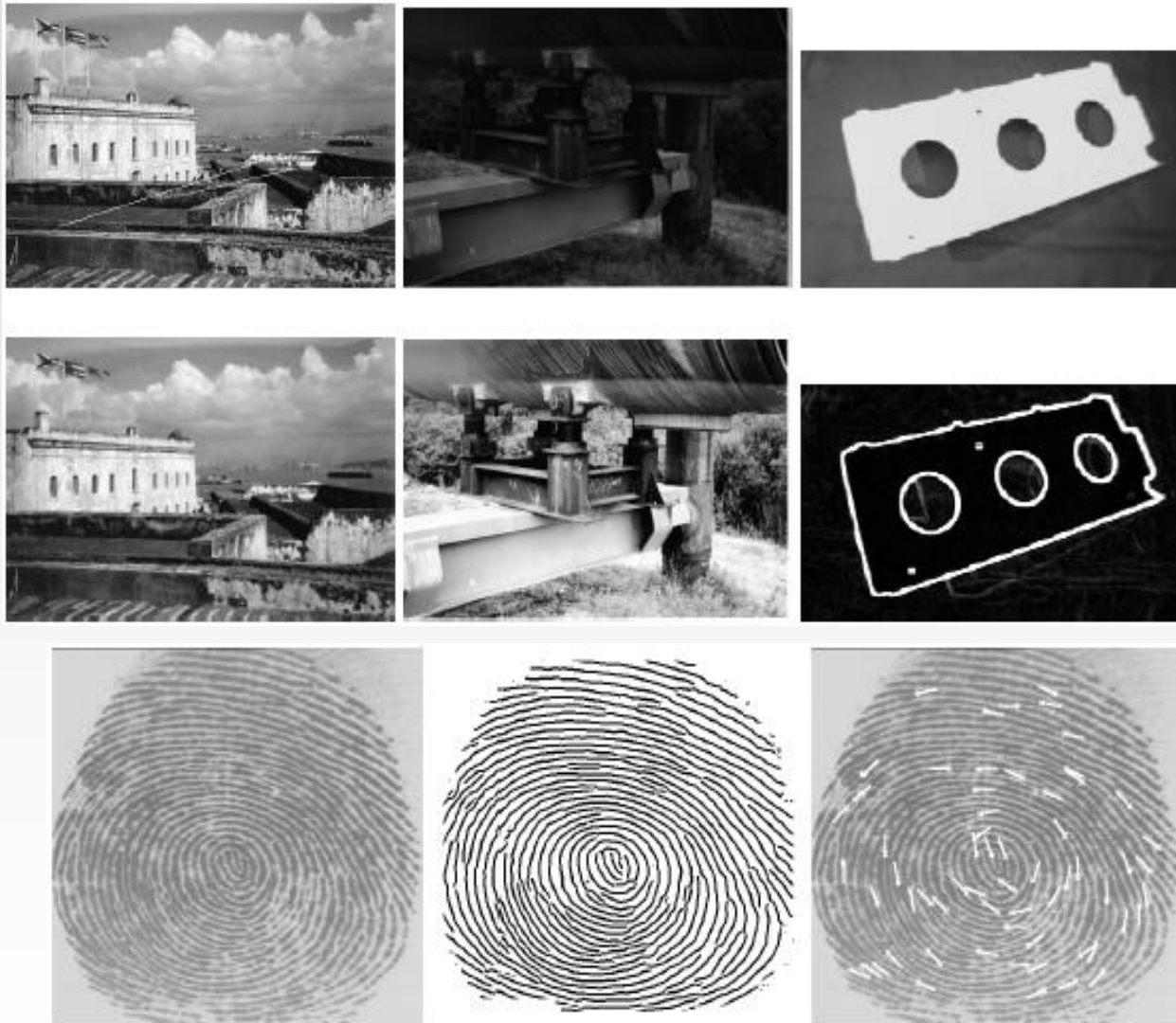


Filtros espaciais **(suavização)**

Prof. Luiz Otavio Murta Jr.
Informática Biomédica

Depto. de Física e Matemática (FFCLRP/USP)

Necessidade de pré-processamento



- **Operadores de suavização**

- os elementos da máscara são positivos e somam um, de modo a que a saída é igual à entrada em regiões de constante intensidade
- A quantidade de suavização e remoção de ruído é proporcional à dimensão da máscara
- Transições abruptas (step edges) são tanto mais espalhadas (blurred) quanto maior for a dimensão da máscara

- **Operadores diferenciais**

- as coordenadas das máscaras tem sinais opostos para que se obtenha uma resposta máxima quando existem transições de intensidade (contraste)
- A soma dos valores é zero para que a resposta seja zero quando a região é constante
- As máscaras de primeira derivada produzem valores absolutos elevados em pontos de grande contraste
- As máscaras de segunda derivada produzem cruzamentos por zero em pontos de grande contraste

Suavização de imagem

- Suavização (filtragem passa-baixas) de imagem

$$O(r,c) = \left(\sum_{i=-N}^N \sum_{j=-N}^N I(r+i, c+j) \right) / N^2$$

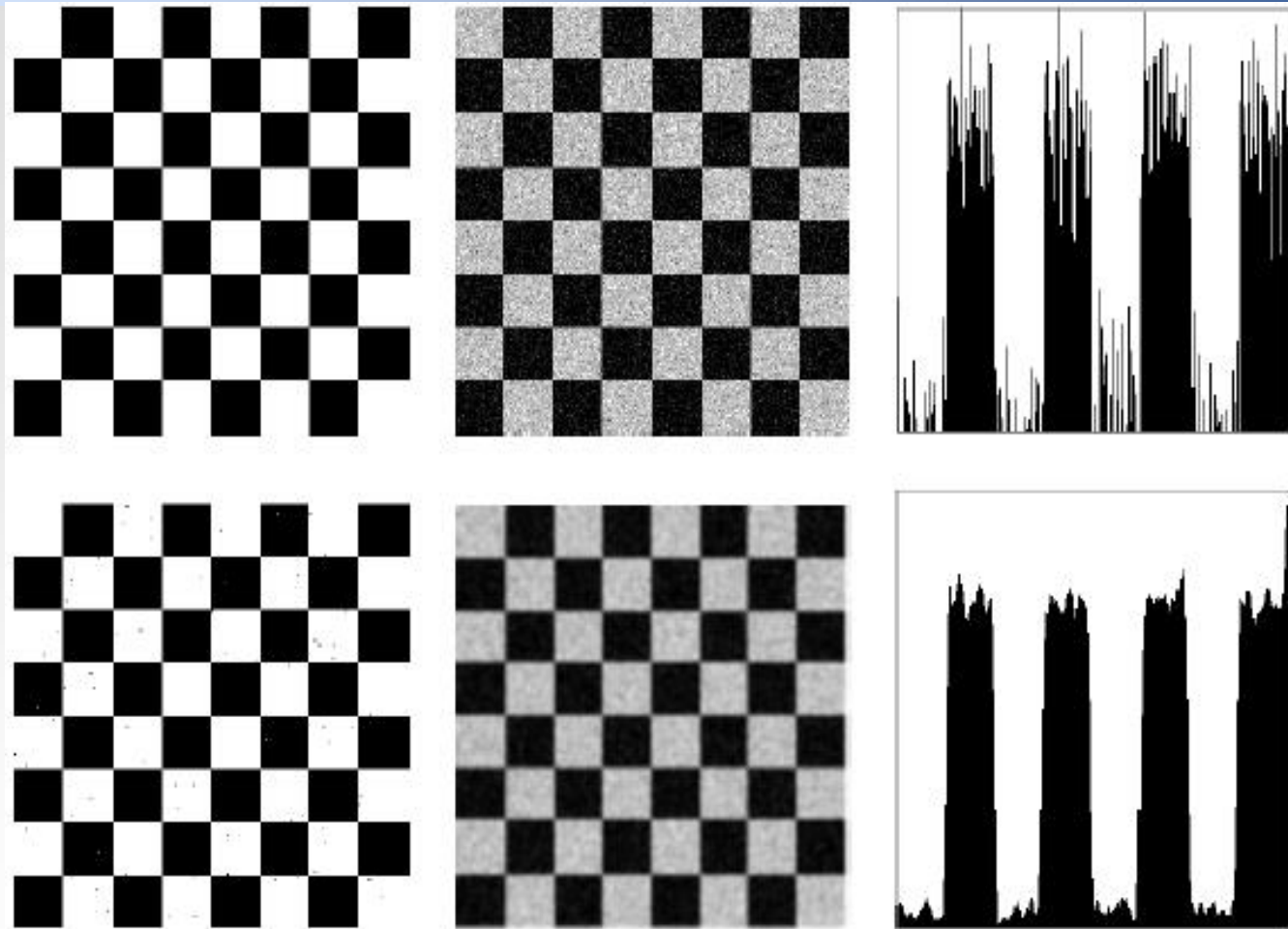
- filtro de média (*box filter*)

$$O(r,c) = \sum_{i=-N}^N \sum_{j=-N}^N g(i,j) I(r+i, c+j)$$

- filtro gaussiano $g(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{d^2}{2\sigma^2}}$

$$d = \sqrt{(x - x_c)^2 + (y - y_c)^2}$$

Necessidade de operação de suavização



Filtros espaciais

- Suavização (borramento)



Filtros espaciais

- Suavização (borramento)



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

Filtros espaciais

- Suavização (borramento)



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

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

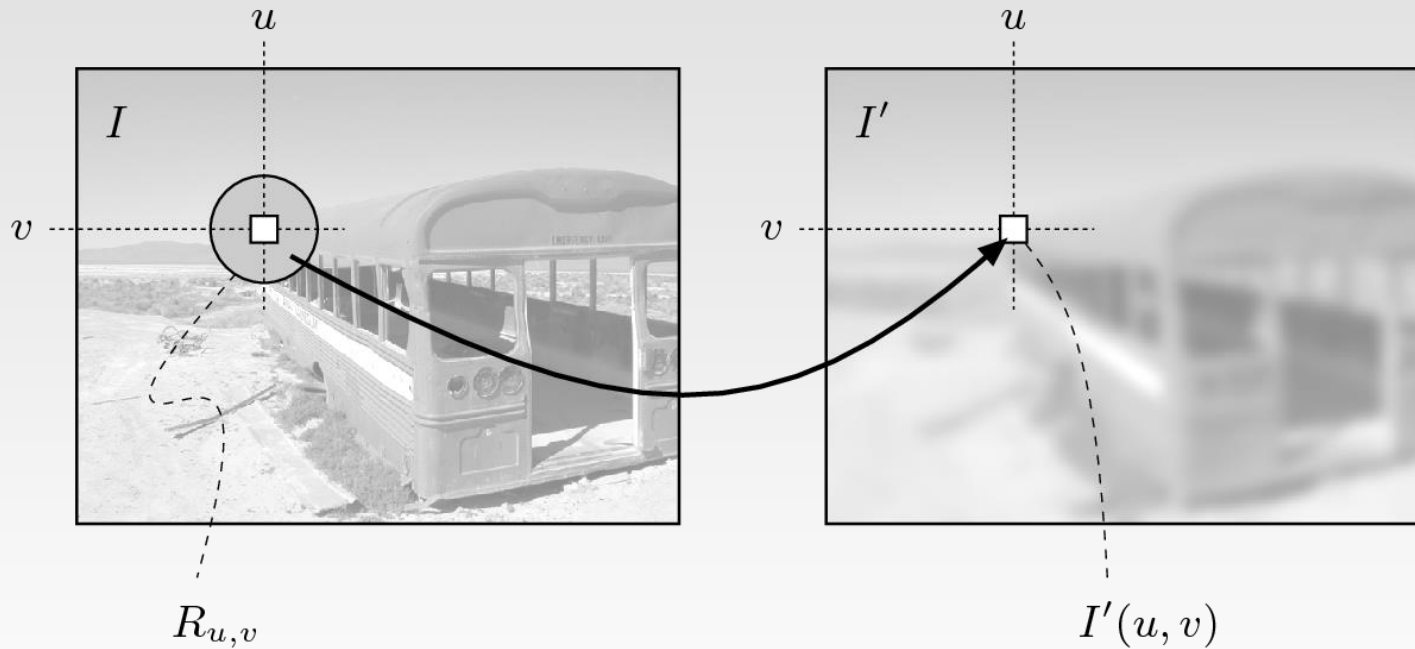
- Suavização (borramento)



$$I'(u, v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^1 \sum_{i=-1}^1 I(u+i, v+j)$$

Filtros espaciais

- Suavização (borramento)



$$I'(u, v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^1 \sum_{i=-1}^1 I(u + i, v + j)$$

Filtros espaciais

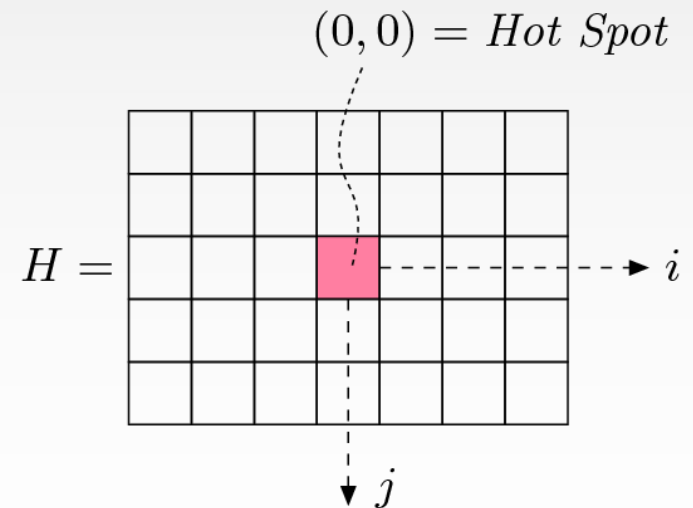
- Convolução 2D (bidimensional)

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

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

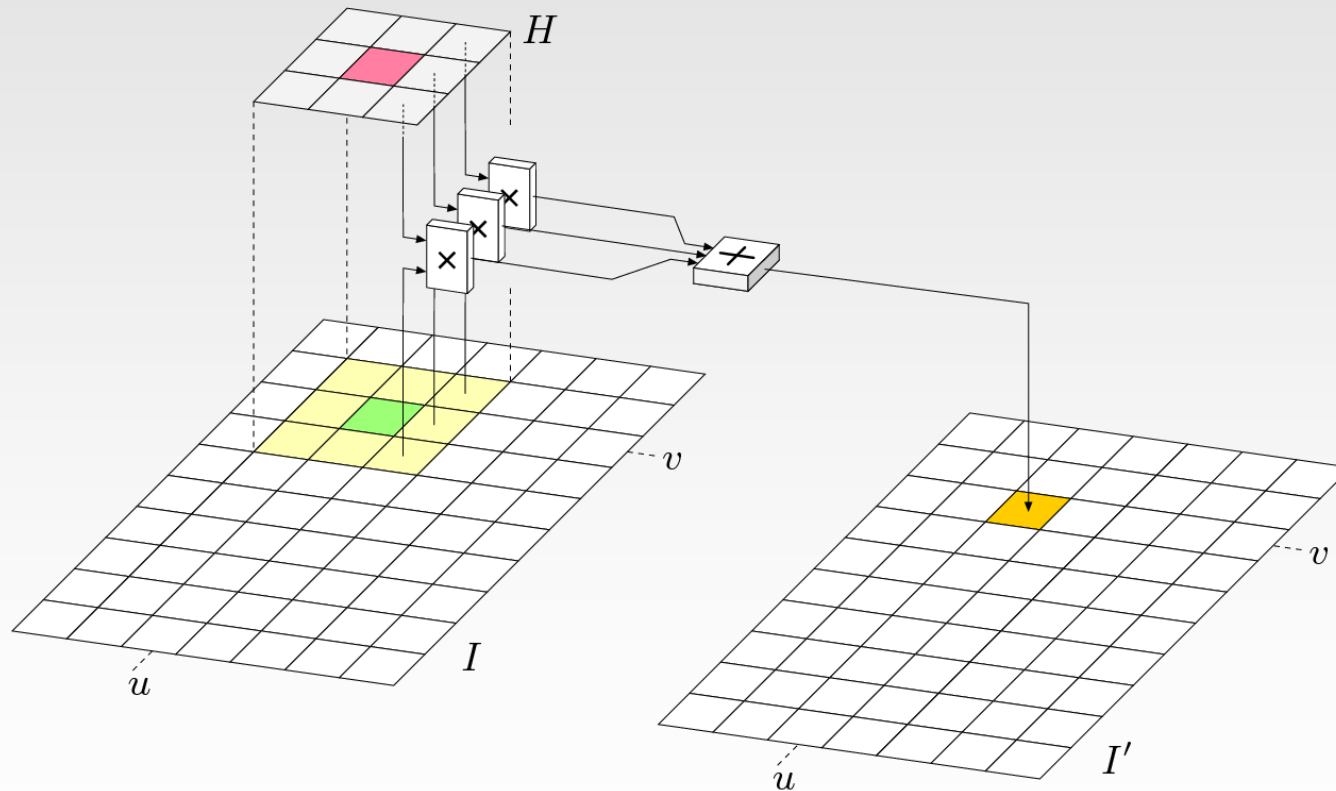
ou

$$I'(u, v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u + i, v + j) \cdot H(i, j)$$

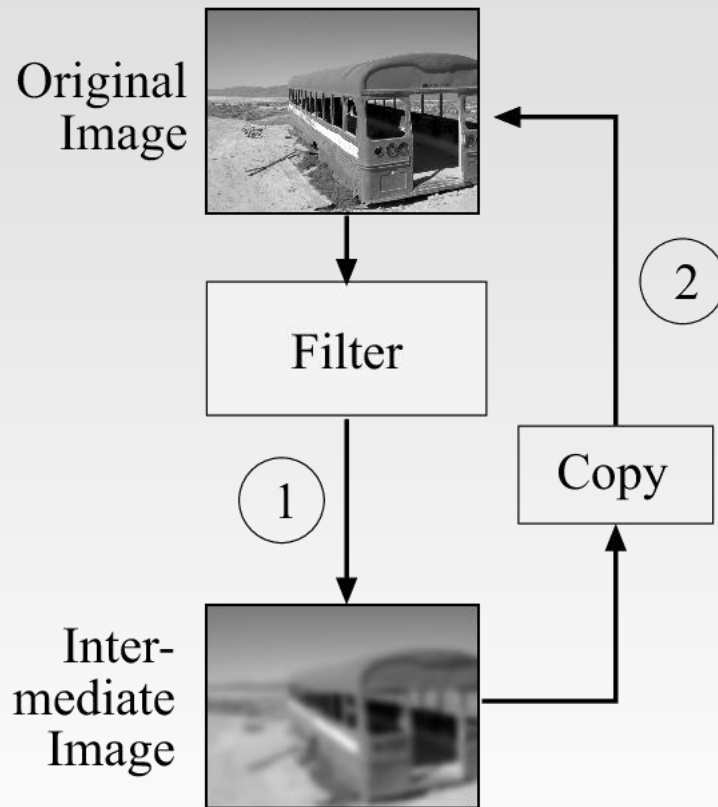


Filtros espaciais

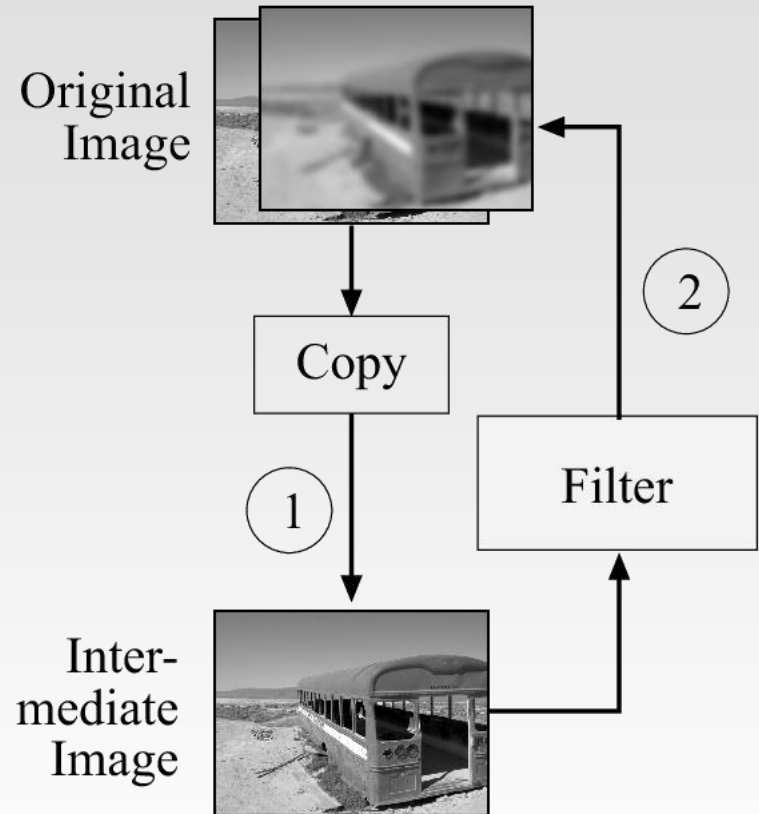
- Convolução 2D (bidimensional)



Filtros espaciais



Version A



Version B

Filtros espaciais

- Convolução 2D (bidimensional)

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

$$H(i, j) = s \cdot H'(i, j) \quad \text{sendo} \quad s = \frac{1}{\sum_{i,j} H'(i, j)}$$

Por exemplo:

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

Filtros espaciais

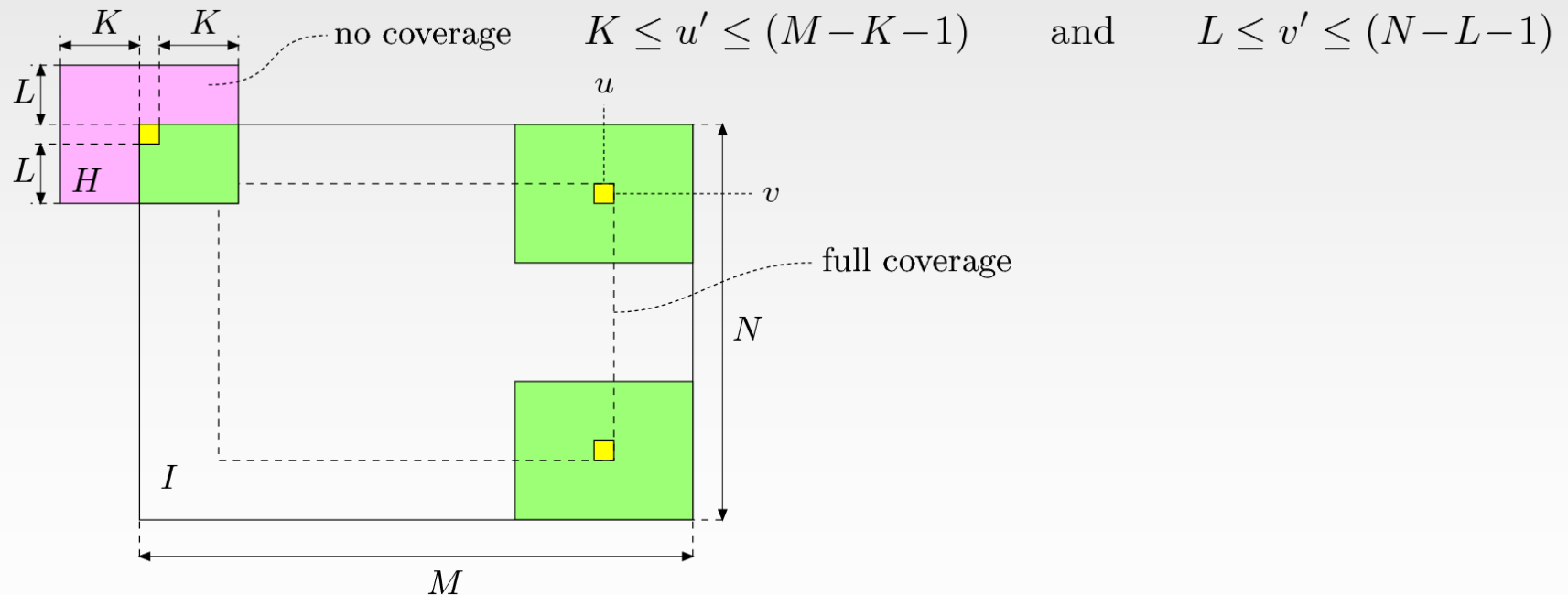
```
1 import ij.*;
2 import ij.plugin.filter.PlugInFilter;
3 import ij.process.*;
4
5 public class Filter_Average3x3 implements PlugInFilter {
6     ...
7     public void run(ImageProcessor orig) {
8         int w = orig.getWidth();
9         int h = orig.getHeight();
10        ImageProcessor copy = 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
```

Filtros espaciais

Tratamento de fronteiras

$$I'(u, v) \leftarrow \text{Offset} + \frac{1}{\text{Scale}} \sum_{j=-2}^{j=2} \sum_{i=-2}^{i=2} I(u+i, v+j) \cdot H(i, j) \quad , \text{ caso geral:}$$

$$I(u, v) \quad \text{with} \quad 0 \leq u < M \quad \text{and} \quad 0 \leq v < N$$



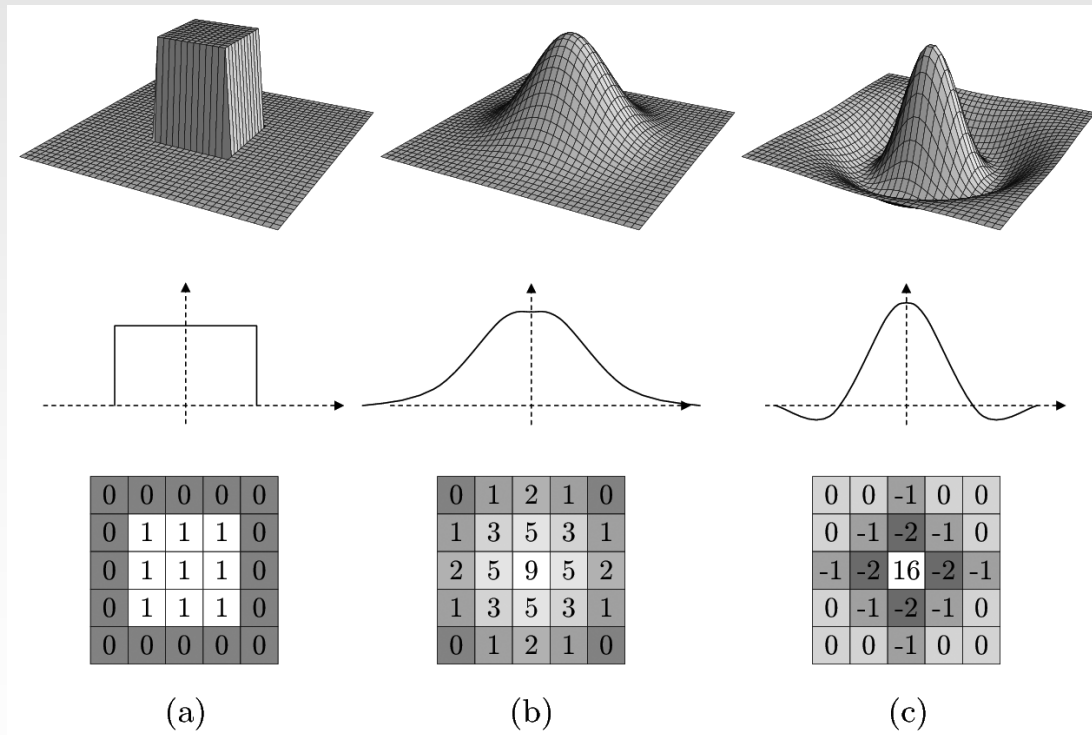
Filtros espaciais

```
1  public void run(ImageProcessor orig) {
2      int w = orig.getWidth();
3      int h = orig.getHeight();
4      // 3 x 3 filter matrix
5      double[][] filter = {
6          {0.075, 0.125, 0.075},
7          {0.125, 0.200, 0.125},
8          {0.075, 0.125, 0.075}
9      };
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             double 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                     // get the corresponding filter coefficient:
20                     double c = filter[j+1][i+1];
21                     sum = sum + c * p;
22                 }
23             }
24             int q = (int) Math.round(sum);
25             orig.putPixel(u, v, q);
26         }
27     }
28 }
```

Filtros espaciais

Mascaras (H):

a) quadrada; b) gaussiana; c) LoG



Filtros espaciais

Mascaras (H):

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

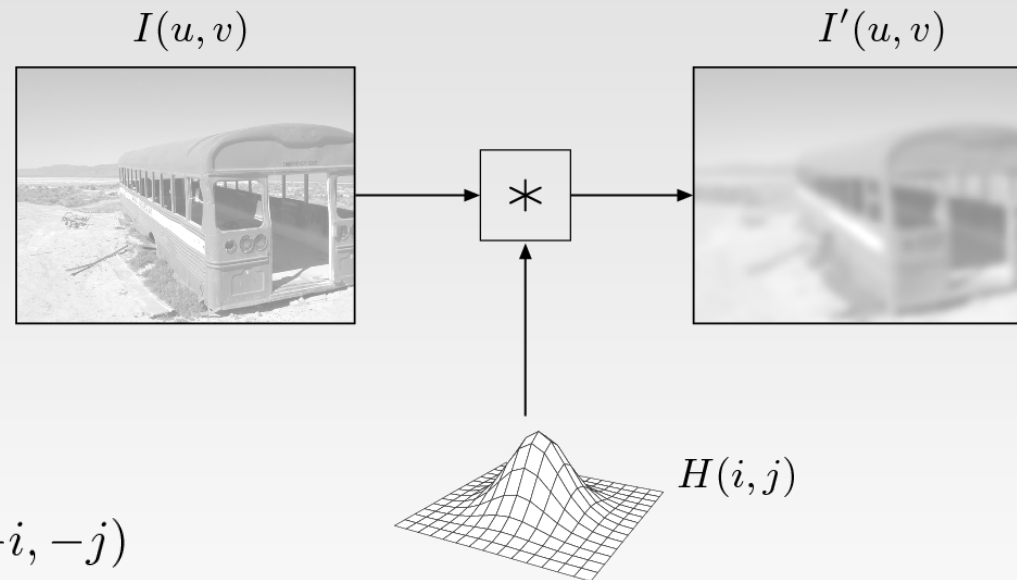
De fato, a equação pode assumir a forma geral:

$$I'(u, v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i, j) \quad ; \quad I' = I * H$$

$$I'(u, v) = \sum_{(i,j) \in R_H} I(u-i, v-j) \cdot H(i, j) \\ = \sum_{(i,j) \in R_H} I(u+i, v+j) \cdot H(-i, -j) \\ = \sum_{(i,j) \in R_H} I(u+i, v+j) \cdot H^*(i, j).$$

Filtros espaciais

Propriedades da convolução 2D



$$H^*(i, j) = H(-i, -j)$$

$$I * H = H * I$$

$$(s \cdot I) * H = I * (s \cdot H) = s \cdot (I * H)$$

Propriedades da convolução 2D

$$H^*(i, j) = H(-i, -j)$$

$$I * H = H * I$$

$$(s \cdot I) * H = I * (s \cdot H) = s \cdot (I * H)$$

$$(I_1 + I_2) * H = (I_1 * H) + (I_2 * H)$$

$$A * (B * C) = (A * B) * C$$

$$\begin{aligned} I * H &= I * (H_1 * H_2 * \dots * H_n) \\ &= (\dots ((I * H_1) * H_2) * \dots * H_n) \end{aligned}$$

Separabilidade da convolução 2D

Sejam:

$$H_1 = [11111] \quad \text{e} \quad H_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

dada a propriedade:

$$H = H_1 * H_2 * \dots * H_n$$

Podemos considerar:

$$I' \leftarrow (I * H_x) * H_y = I * \underbrace{(H_x * H_y)}_{H_{xy}}$$

Separabilidade da convolução 2D

Sejam:

$$H_1 = [11111] \quad \text{e} \quad H_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Podemos considerar:

$$I' \leftarrow (I * H_x) * H_y = I * \underbrace{(H_x * H_y)}_{H_{xy}}$$

Então:

$$H_{xy} = H_x * H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & \mathbf{1} & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Filtros gaussianos

Caso 1D:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

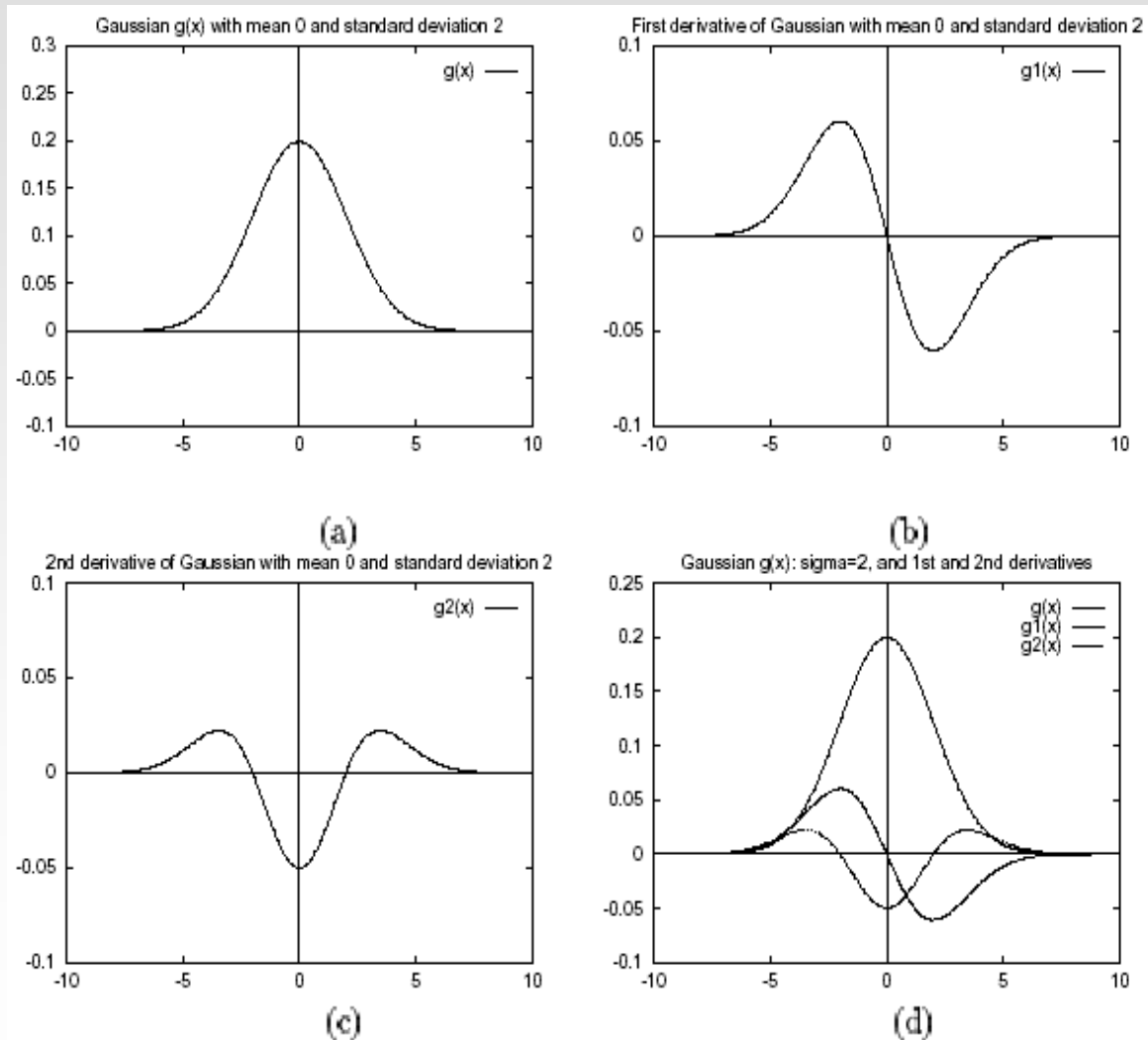
$$g'(x) = \frac{-x}{\sigma^2} g(x)$$

$$g''(x) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} \right) g(x)$$

Caso 2D:

$$h(x, y) = g(r)$$

$$r = \sqrt{x^2 + y^2}$$



Filtros espaciais

Separabilidade da convolução 2D (H Gaussiano)

Seja:

$$H_{x,y}(i, j) = (H_x \otimes H_y)(i, j) = H_x(i) \cdot H_y(j)$$

Podemos considerar:

$$G_\sigma(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}} = e^{-\frac{x^2}{2\sigma^2}} \cdot e^{-\frac{y^2}{2\sigma^2}} = g_\sigma(x) \cdot g_\sigma(y)$$

Então:

$$I' \leftarrow I * H^{G,\sigma} = I * H_x^{G,\sigma} * H_y^{G,\sigma}$$

Filtros espaciais

Kernel Gaussiano (H Gaussiano)

```
1  float[] makeGaussKernel1d(double sigma) {
2
3      // create the kernel
4      int center = (int) (3.0*sigma);
5      float[] kernel = new float[2*center+1]; // odd size
6
7      // fill the kernel
8      double sigma2 = sigma * sigma;           //  $\sigma^2$ 
9      for (int i=0; i<kernel.length; i++) {
10         double r = center - i;
11         kernel[i] = (float) Math.exp(-0.5 * (r*r) / sigma2);
12     }
13
14     return kernel;
15 }
```

Filtros espaciais

Convolução 2D (Função Delta)

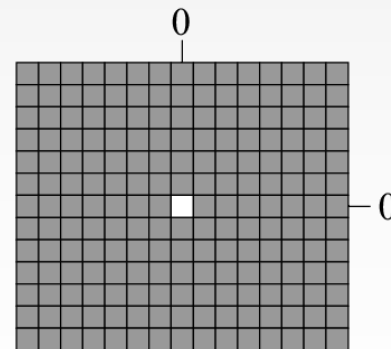
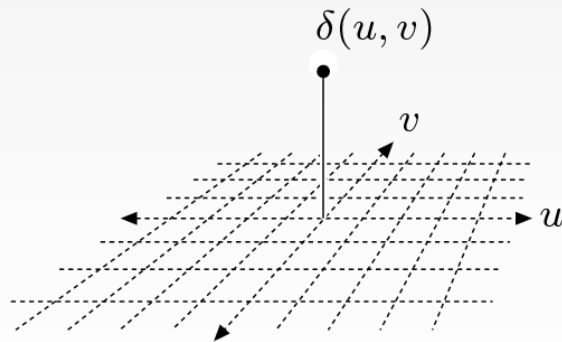
Seja:

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

Sabemos que:

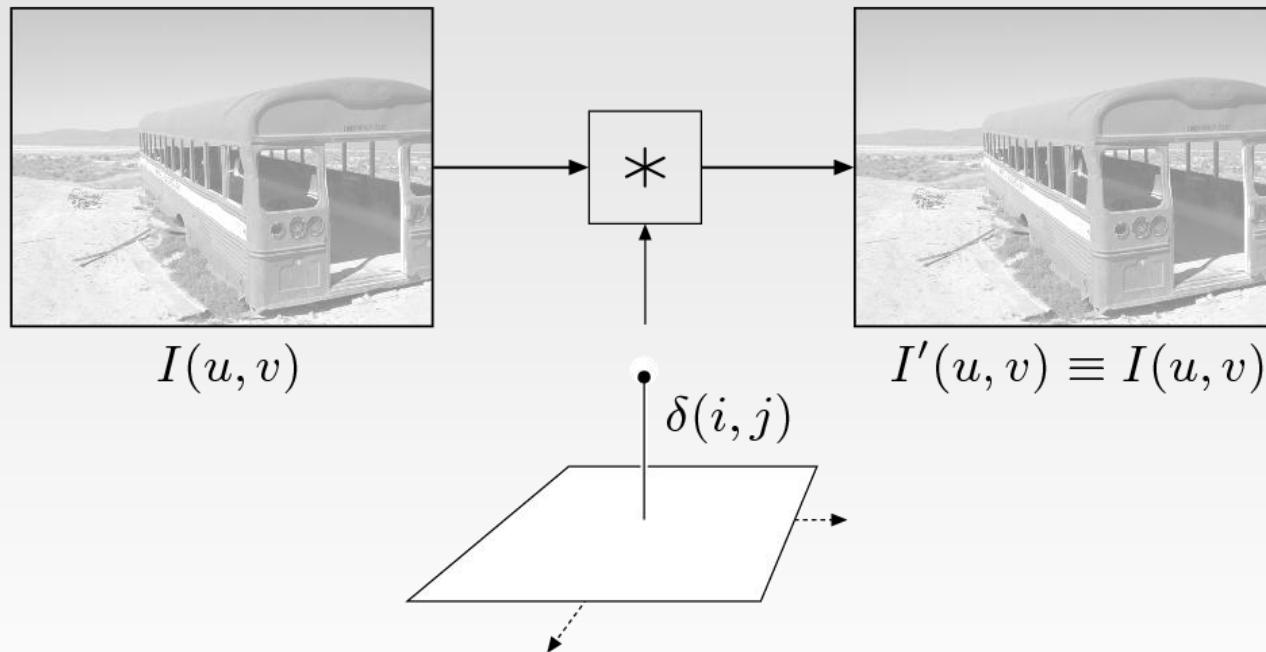
$$I * \delta = I$$

$$H * \delta = \delta * H = H$$



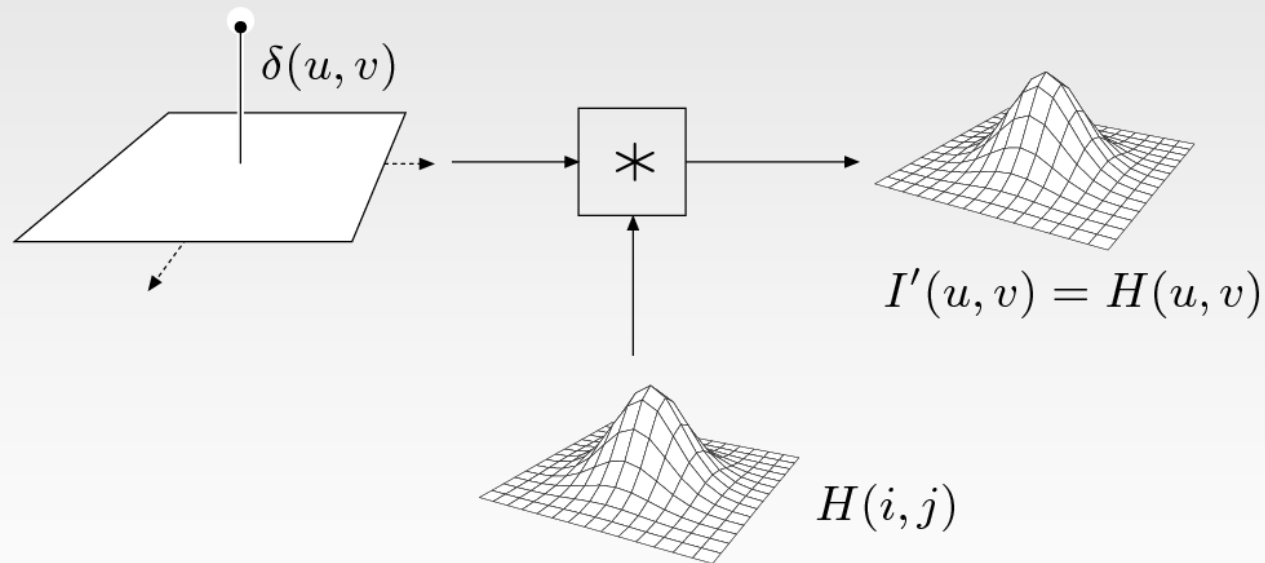
Filtros espaciais

Convolução 2D (Função Delta)



Filtros espaciais

Convolução 2D (Função Delta e Kernel Gaussiano)

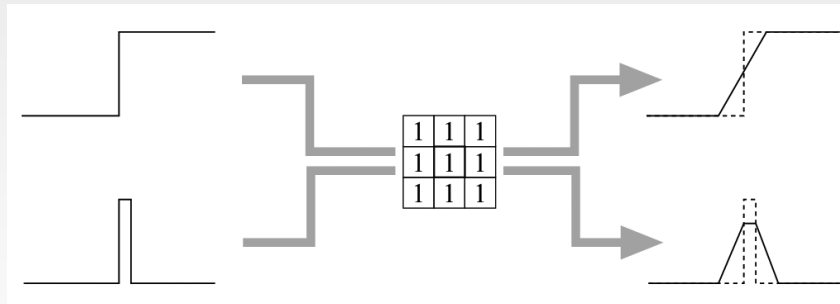


Filtros espaciais

Convolução 2D (Função Delta e Kernel Gaussiano)

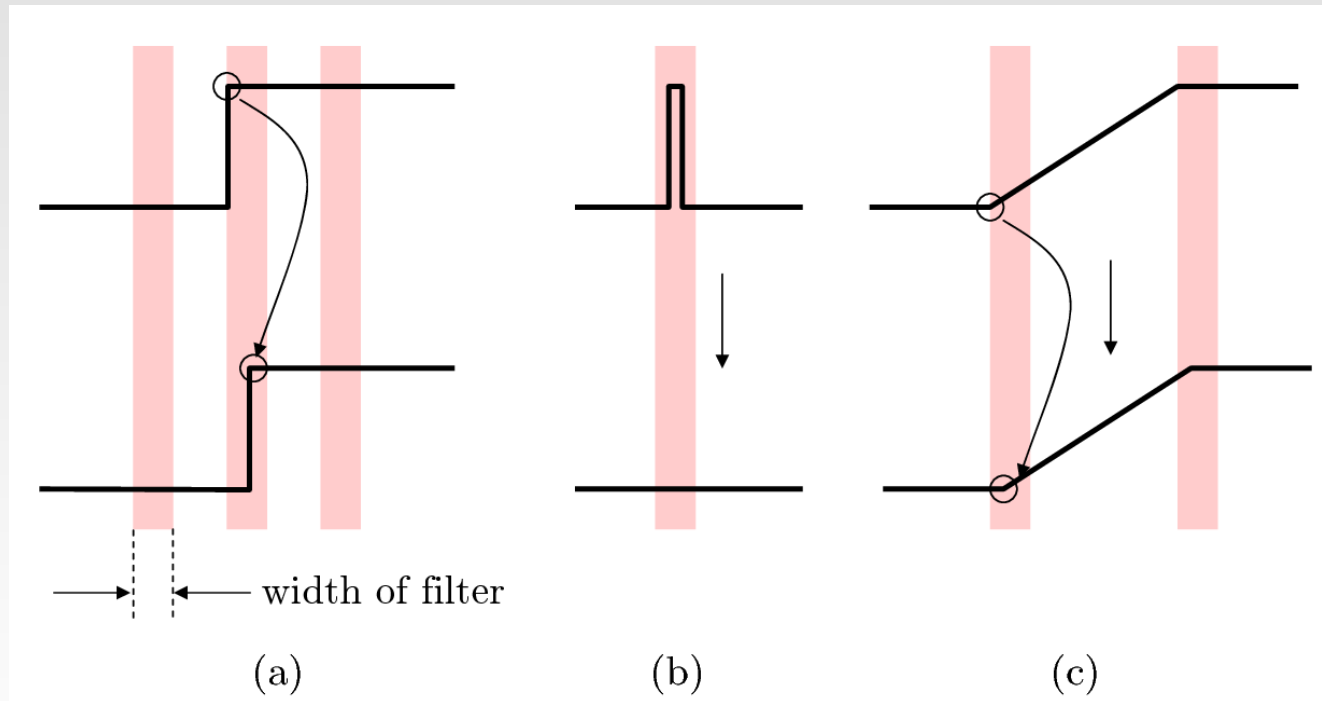
$$I'(u, v) \leftarrow \min \{I(u+i, v+j) \mid (i, j) \in R\}$$

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



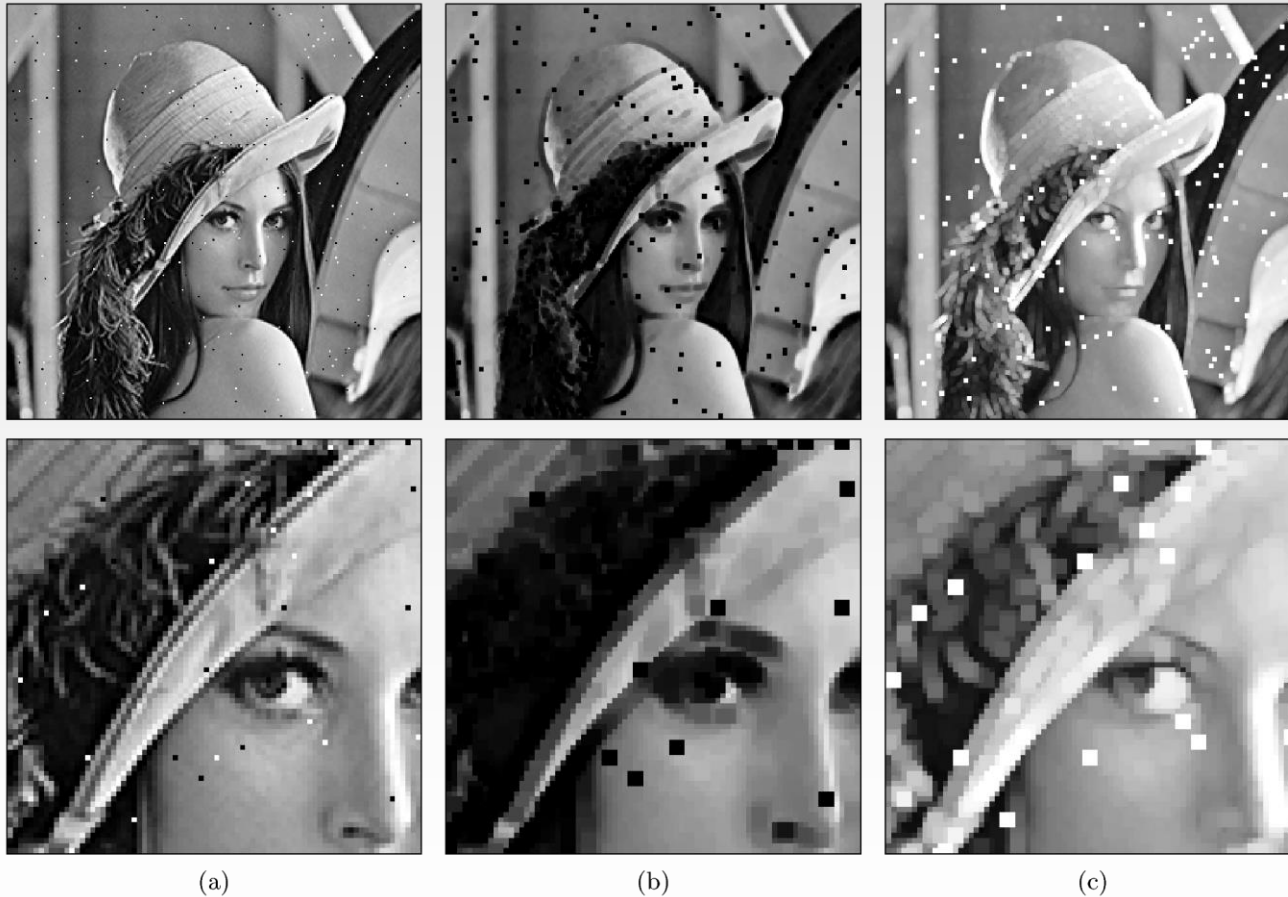
Filtros espaciais

Efeitos do tamanho do Kernel H



Filtros espaciais

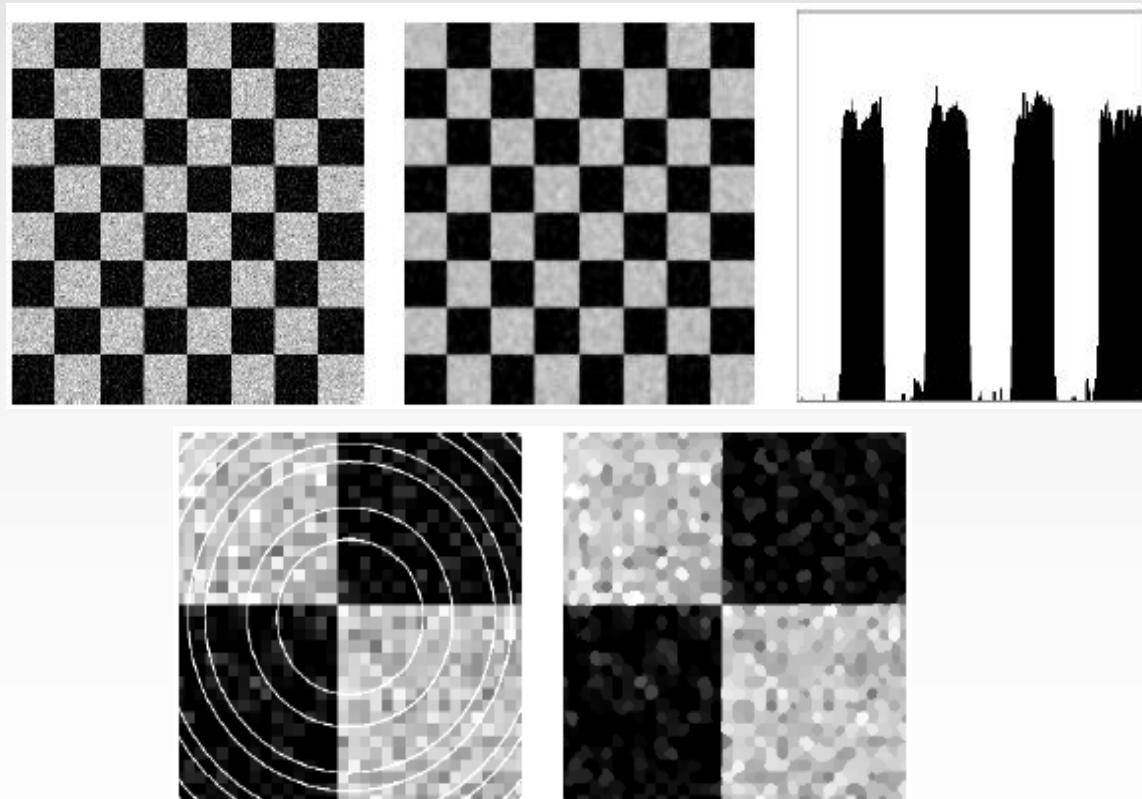
Filtro de mediana



Filtragem de mediana

- Seja $A[i]_{i=0,\dots,n-1}$ uma lista ordenada de números reais. A mediana do conjunto A é o valor $A[(n-1)/2]$

- Exemplos



Filtro de mediana

$$\text{median}(p_0, p_1, \dots, p_K, \dots, p_{2K}) \triangleq p_K$$

$$\text{median}(p_0, \dots, p_{K-1}, p_K, \dots, p_{2K-1}) \triangleq (p_{K-1} + p_K) / 2$$

$$I'(u, v) \leftarrow \text{median} \{I(u+i, v+j) \mid (i, j) \in R\}$$

Filtro de mediana

$$\text{median}(p_0, p_1, \dots, p_K, \dots, p_{2K}) \triangleq p_K$$

$$\text{median}(p_0, \dots, p_{K-1}, p_K, \dots, p_{2K-1}) \triangleq (p_{K-1} + p_K) / 2$$

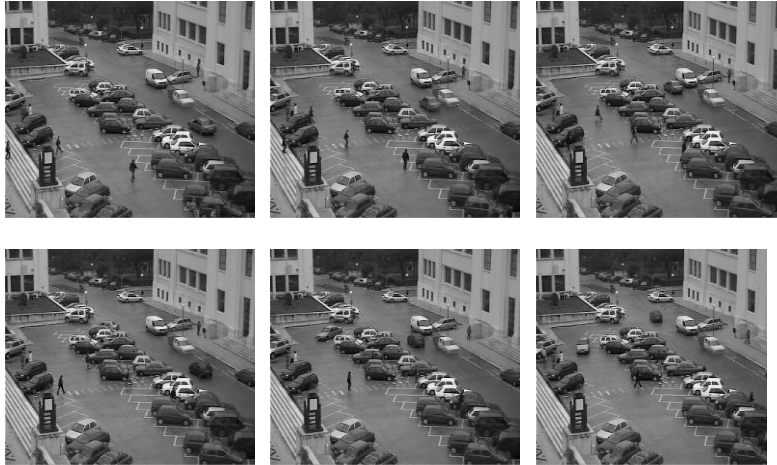
$$I'(u, v) \leftarrow \text{median} \{I(u+i, v+j) \mid (i, j) \in R\}$$

Filtros espaciais

Filtro de mediana



Filtragem temporal com filtro de mediana



Filtragem de Mediana da Sequência



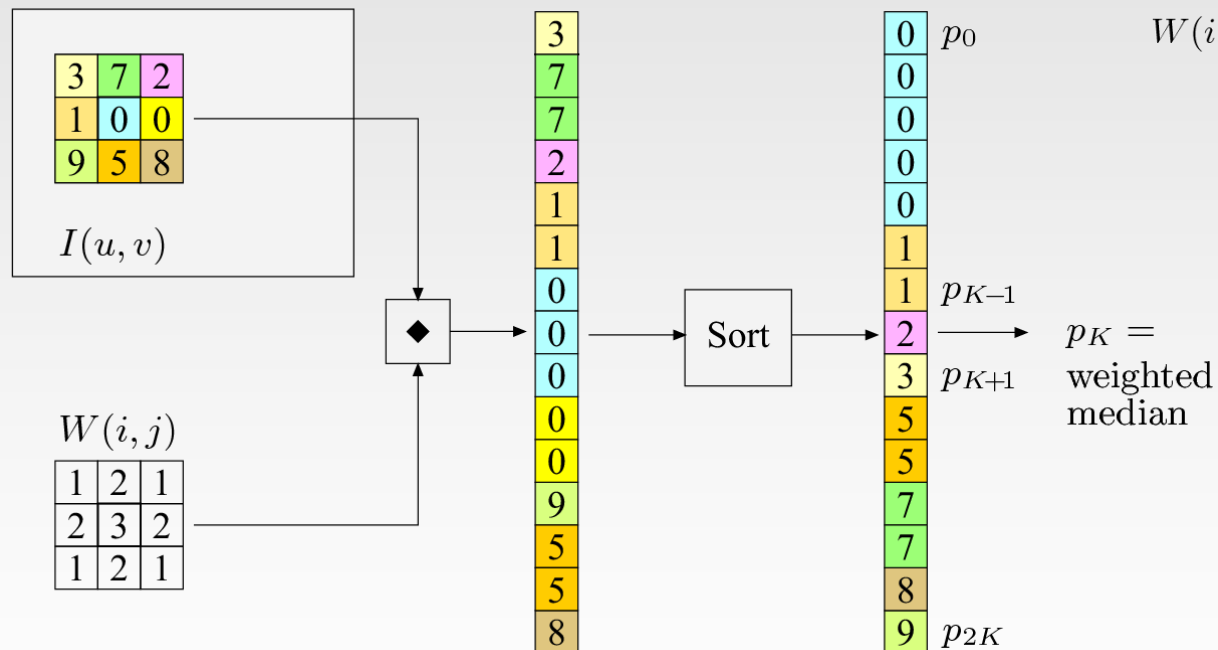
Filtros espaciais

Filtro de mediana

```
1 import ij.*;
2 import ij.plugin.filter.PlugInFilter;
3 import ij.process.*;
4 import java.util.Arrays;
5
6 public class Filter_Median3x3 implements PlugInFilter {
7     final int K = 4; // filter size
8
9     public void run(ImageProcessor orig) {
10         int w = orig.getWidth();
11         int h = orig.getHeight();
12         ImageProcessor copy = orig.duplicate();
13
14         // vector to hold pixels from 3x3 neighborhood
15         int[] P = new int[2*K+1];
16
17         for (int v = 1; v <= h-2; v++) {
18             for (int u = 1; u <= w-2; u++) {
19                 // fill the pixel vector P for filter position u, v
20                 int k = 0;
21                 for (int j = -1; j <= 1; j++) {
22                     for (int i = -1; i <= 1; i++) {
23                         P[k] = copy.getPixel(u+i, v+j);
24                         k++;
25                     }
26                 }
27                 // sort pixel vector and take the center element
28                 Arrays.sort(P);
29                 orig.putPixel(u, v, P[K]);
30             }
31         }
32     }
33
34 } // end of class Filter_Median3x3
```

Filtros espaciais

Filtro de mediana ponderada



$$W^+(i, j) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

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

Filtros espaciais

Tratamento de fronteiras



(a)



(b)



(c)



(d)

Filtros espaciais

Implementação dos filtros

```
1 import ij.plugin.filter.GaussianBlur;
2 ...
3 public void run(ImageProcessor ip) {
4     GaussianBlur gb = new GaussianBlur();
5     double radius = 2.5;
6     gb.blur(ip, radius);
7 }
```

```
1 import ij.plugin.filter.Convolver;
2 ...
3 public void run(ImageProcessor I) {
4     float[] H = { // filter array is one-dimensional!
5         0.075f, 0.125f, 0.075f,
6         0.125f, 0.200f, 0.125f,
7         0.075f, 0.125f, 0.075f };
8     Convolver cv = new Convolver();
9     cv.setNormalize(false); // do not use filter normalization
10    cv.convolve(I, H, 3, 3); // apply the filter H to I
11 }
```

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

Tratamento de fronteiras

```
1 import ij.plugin.filter.GaussianBlur;
2 ...
3 public void run(ImageProcessor ip) {
4     GaussianBlur gb = new GaussianBlur();
5     double radius = 2.5;
6     gb.blur(ip, radius);
7 }
```

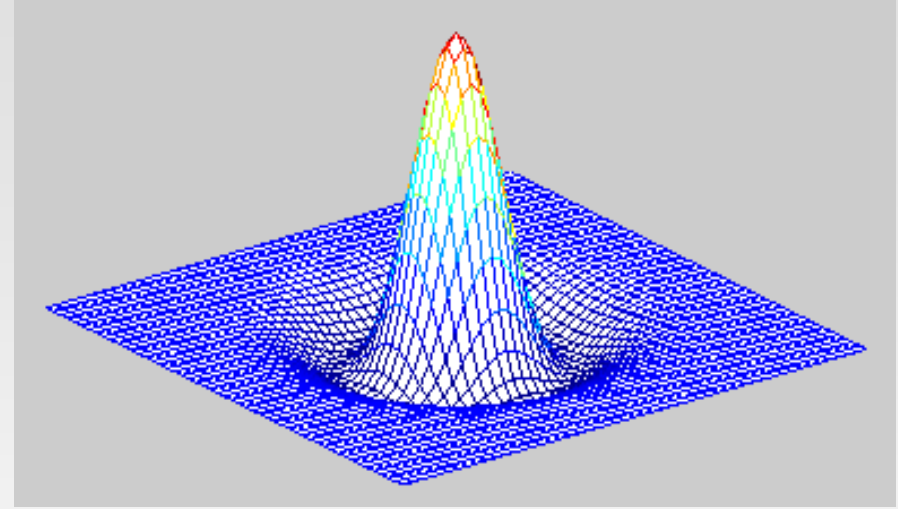
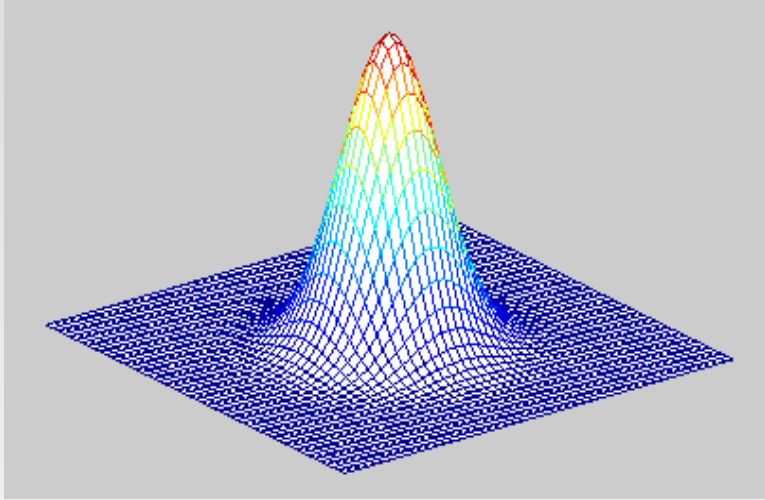
```
1 import ij.plugin.filter.Convolver;
2 ...
3 public void run(ImageProcessor I) {
4     float[] H = {                // filter array is one-dimensional!
5         0.075f, 0.125f, 0.075f,
6         0.125f, 0.200f, 0.125f,
7         0.075f, 0.125f, 0.075f };
8     Convolver cv = new Convolver();
9     cv.setNormalize(false); // do not use filter normalization
10    cv.convolve(I, H, 3, 3); // apply the filter H to I
11 }
```

Filtro LoG:

$$\text{LoG}_{\sigma}(x, y) = -\left(\frac{x^2 + y^2 - \sigma^2}{\sigma^4}\right) \cdot e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

Detector de bordas baseado na função Laplaciana – filtro LOG

$g(x,y)$



0	0	0	-1	-1	-2	-1	-1	0	0	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	-2	-7	-15	-22	-23	-22	15	-7	-2	0
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-2	-9	-23	-1	103	178	103	-1	-23	-9	-2
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
0	-2	-7	-15	-22	-23	-22	15	-7	-2	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	0	0	-1	-1	-2	-1	-1	0	0	0

$-L(x,y)$

→ Chapéu mexicano

← Máscara 11x11 ($\sigma^2=2$)

$$L(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}$$