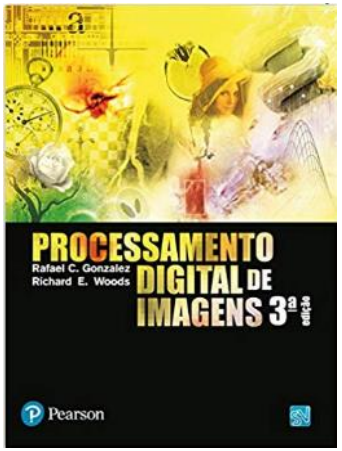
VISÃO COMPUTACIONAL

TÉCNICAS DE FILTRAGEM

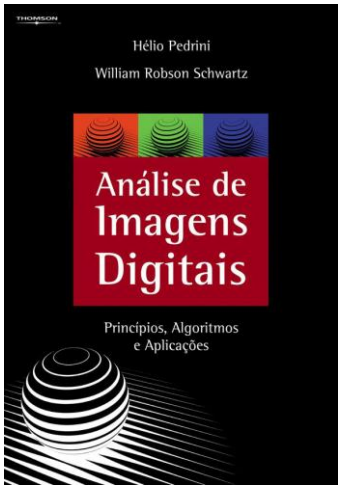
Prof. Msc. Giovanni Lucca França da Silva
E-mail: giovanni-lucca@live.com

SOBRE A DISCIPLINA

- Bibliografia principal:
  - GONZALEZ, Rafael C.; WOODS, Richard C. **Processamento digital de imagens**. Pearson, 2011.



- Bibliografia complementar:
  - PEDRINI, Hélio; SCHWARTZ, William Robson. **Análise de imagens digitais: princípios, algoritmos e aplicações**. Thomson Learning, 2008.



NA AULA PASSADA...

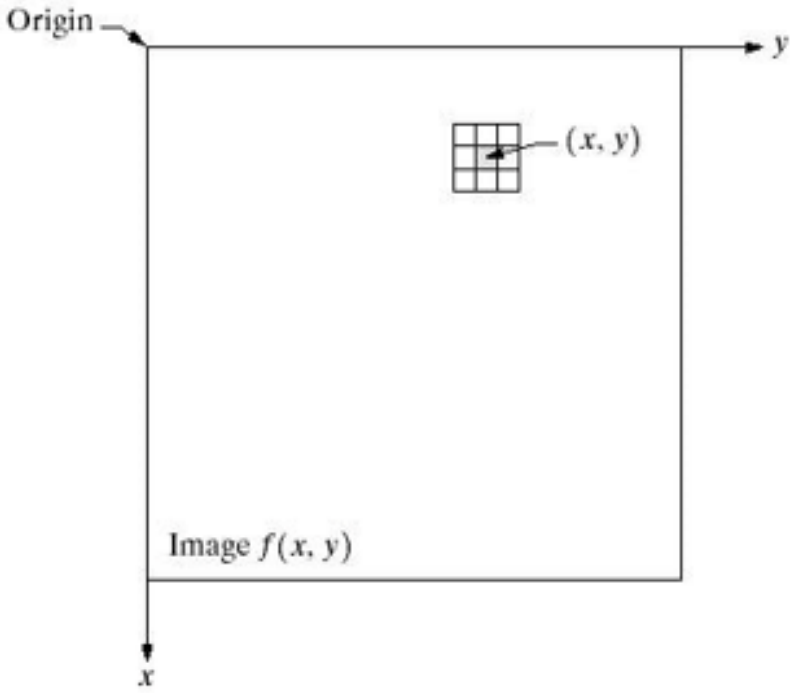
- Fundamentos de cores.
- Modelos de cores.

ROTEIRO

- Introdução.
- Filtros.

INTRODUÇÃO

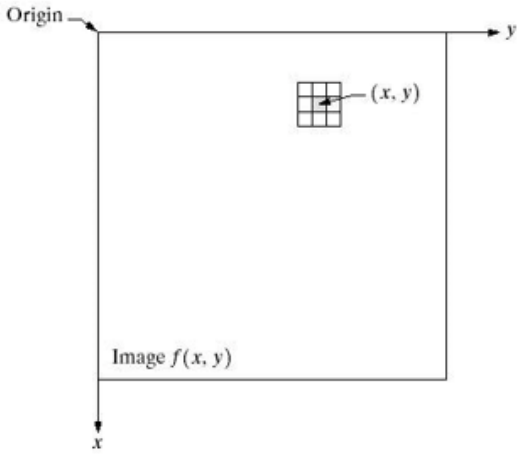
- Convolução.



INTRODUÇÃO

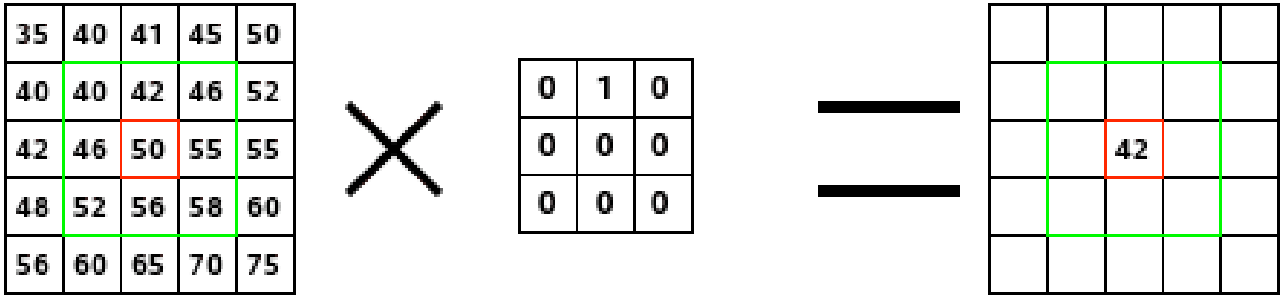
- Convolução.
  - É um operador linear que, a partir de duas funções dadas, resulta numa terceira que mede a soma do produto dessas funções ao longo da região subentendida pela superposição delas em função do deslocamento existente entre elas.

$$g(x,y) = \sum_{i=1}^{k} w_i \cdot f(x,y)$$



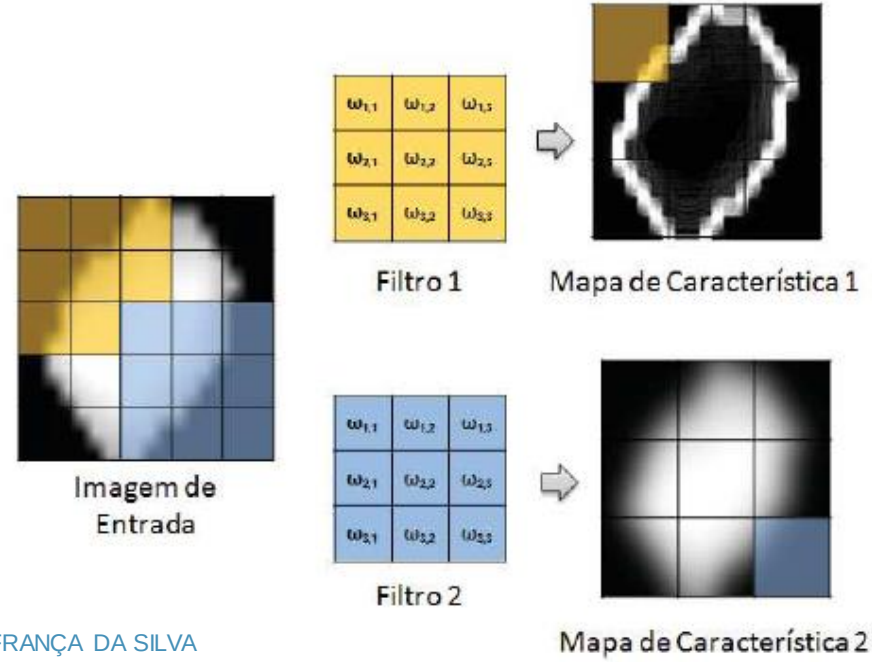
INTRODUÇÃO

- Convolução.
  - Convolução é o tratamento de uma matriz por outra chamada “núcleo” (kernel).



INTRODUÇÃO

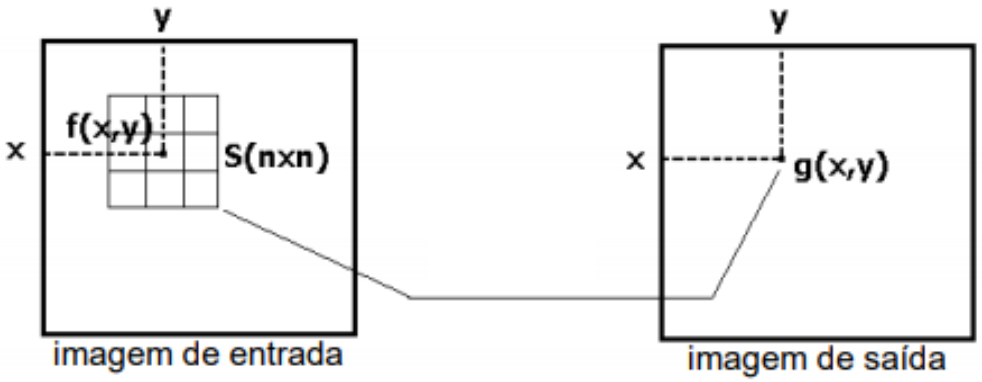
- Convolução.
  - Exemplo: redes neurais convolucionais.



FILTRAGEM

- Operações locais que visam extrair informações importantes da imagem.

- Combinar a intensidade de um certo número de pixels, para gerar a intensidade da imagem de saída.



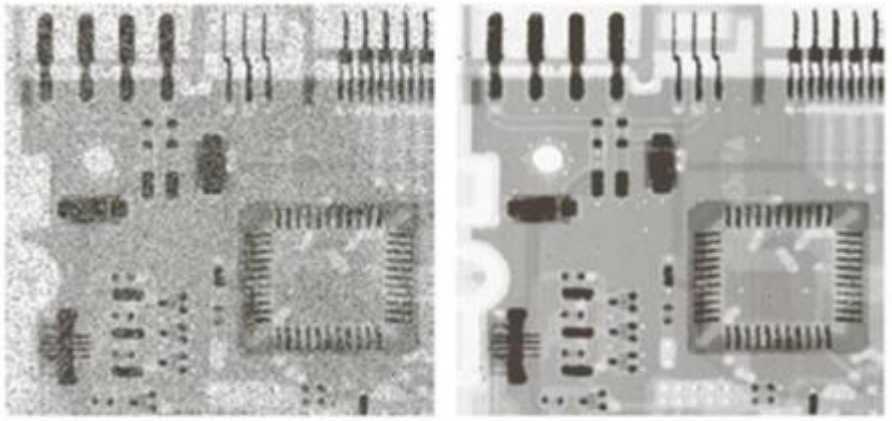
FILTRAGEM

- Classificação.
  - Domínio: frequência ou espacial.
  - Tipo de frequência:
    - Passa ou elimina baixas frequências.
    - Passa ou elimina altas frequências.
    - Passa ou elimina faixas de frequências.
  - Linearidade: lineares ou não lineares.

FILTRAGEM

- Uma grande variedade de filtros digitais podem ser implementados através da convolução no domínio espacial.

- São os operadores locais mais utilizados em processamento de imagens, com diversas aplicações:
  - Pré-processamento:
    - Eliminação de ruídos.
    - Suavização.
  - Segmentação.



FILTRAGEM

- Filtragem espacial refere-se ao plano da imagem.
  - Envolve a manipulação direta dos pixels da imagem utilizando uma máscara espacial.
  - Valores das máscaras são chamados de coeficientes.
  - O processo de filtragem é similar a uma operação de convolução.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

FILTRAGEM

- Processo de filtragem:
  - Cada elemento da máscara é multiplicado pelo valor do pixel correspondente na imagem f.
  - A soma desses resultados é o novo valor do nível de cinza na nova imagem g.
  - Exemplo: w é uma máscara de n x n = k pixels. O processo de filtragem para cada pixel na imagem g(x, y) será dado por:

$$g(x,y) = \sum_{i=1}^{k} w_i . f(x,y)$$

FILTRAGEM

- O tamanho da máscara e os valores de seus coeficientes definem o tipo de filtragem produzido:
  - Passa baixa e média espacial (suavização).
  - Filtragem da mediana (eliminação de ruídos).
  - Passa alta (realce).
  - Gradientes (detectores de borda).

FILTRAGEM

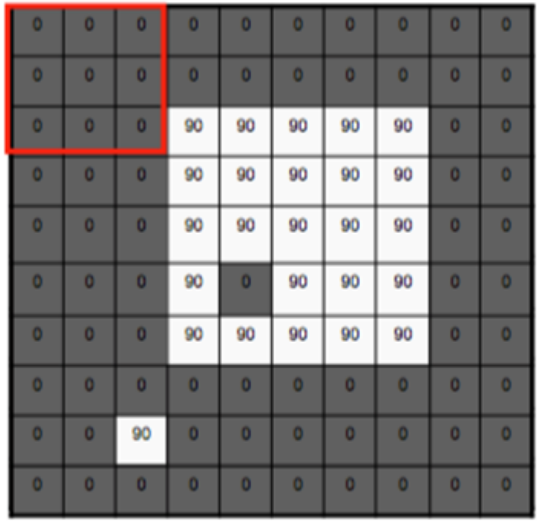
- Filtros de suavização (filtros passa-baixa):
  - Utiliza uma máscara que realiza a média da vizinhança.
  - Numa máscara de média, os coeficientes são positivos e a soma deles é igual a 1.
  - Quanto maior a máscara, maior o efeito de borramento (redução de ruídos).

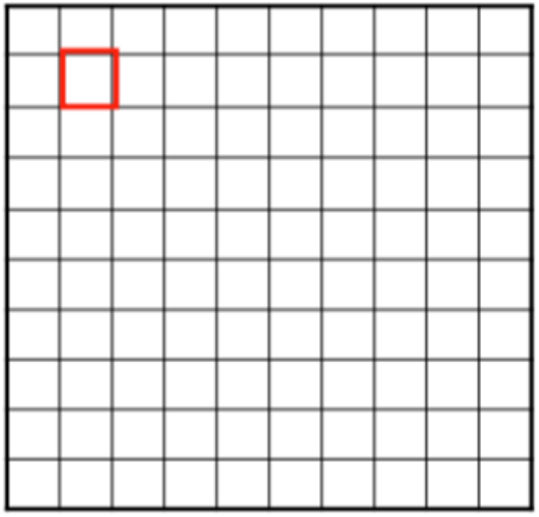
$$\frac{1}{5}\begin{bmatrix}0 & 1 & 0\\ 1 & 1 & 1\\ 0 & 1 & 0\end{bmatrix} \quad \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \quad \frac{1}{32}\begin{bmatrix}1 & 3 & 1\\ 3 & 16 & 3\\ 1 & 3 & 1\end{bmatrix} \quad \frac{1}{8}\begin{bmatrix}0 & 1 & 0\\ 1 & 4 & 1\\ 0 & 1 & 0\end{bmatrix}$$

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.

$F[x, y]$ $G[x, y]$



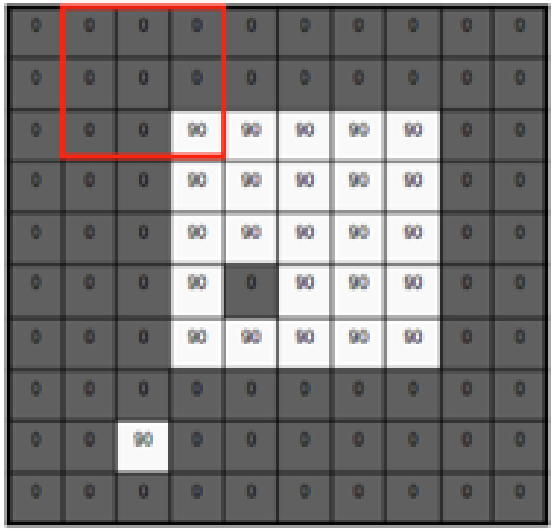


$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

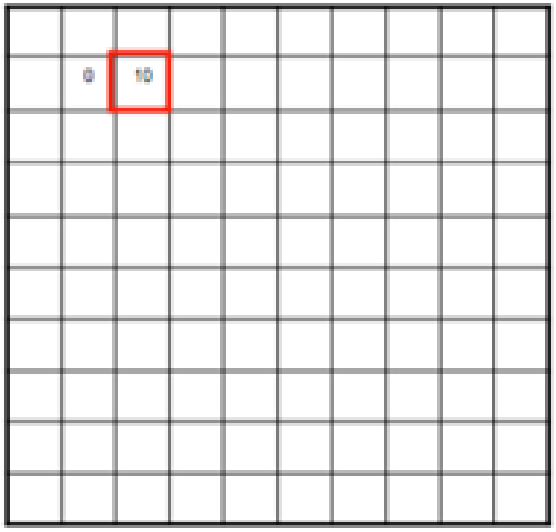
FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.

$F[x, y]$



$G[x, y]$

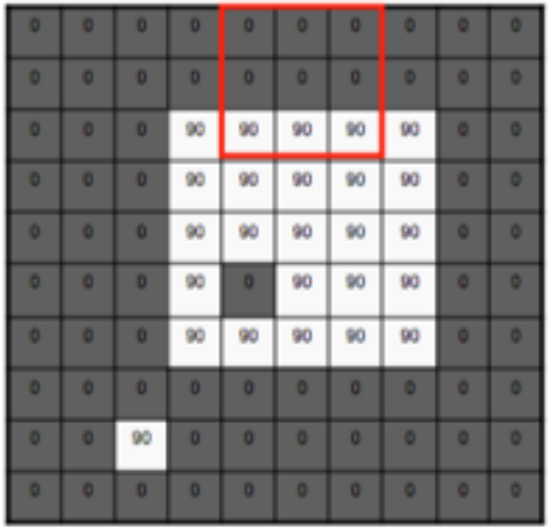


$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

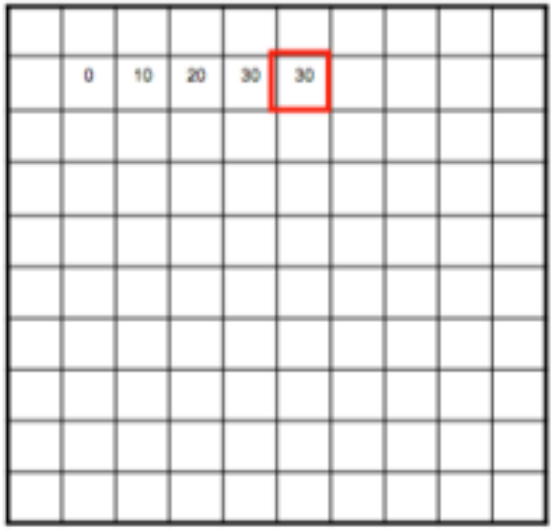
FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.

$F[x, y]$



$G[x, y]$

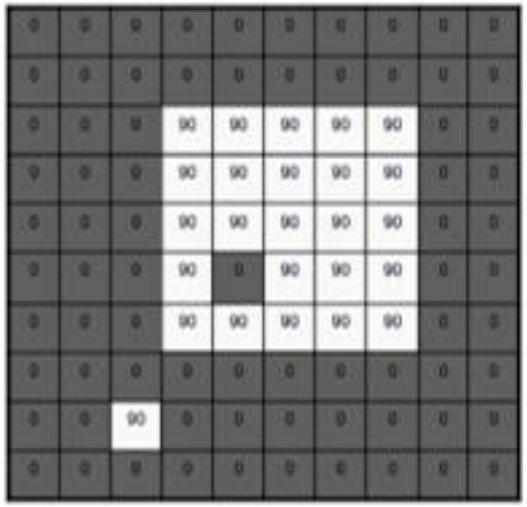


$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

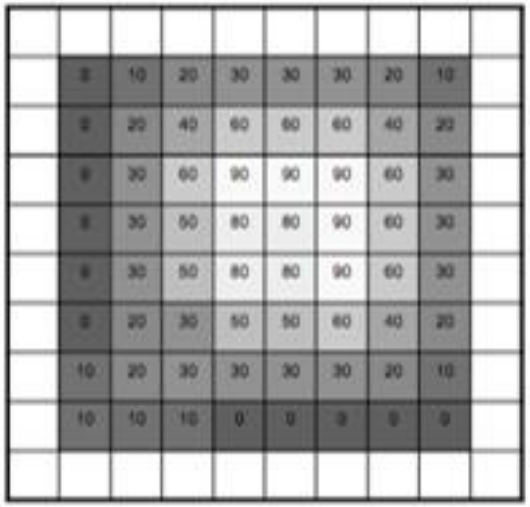
FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.

$F[x, y]$



$G[x, y]$



FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.

5x5

7x7

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.
    - Código no OpenCV:

C++: void **blur**(InputArray **src**, OutputArray **dst**, Size **ksize**, Point **anchor**=Point(-1,-1), int **borderType**=BORDER_DEFAULT)

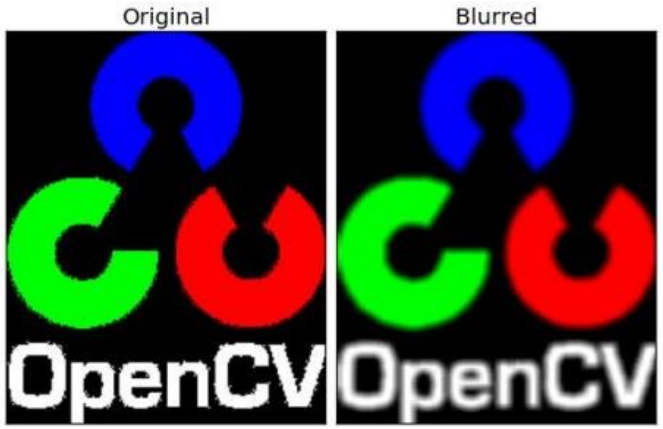
Python: `cv2.blur`(src, ksize[, dst[, anchor[, borderType]]]) → dst

Parameters:
- **src** – input image; it can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – blurring kernel size.
- **anchor** – anchor point; default value `Point(-1,-1)` means that the anchor is at the kernel center.
- **borderType** – border mode used to extrapolate pixels outside of the image.

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.
    - Código no OpenCV:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('opencv_logo.png')

blur = cv2.blur(img,(5,5))

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```



FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da média.

```
int value = 0;
int edge = windowSize / 2;

for(int i = edge; i < width-edge; i++){
    for(int j = edge; j < height-edge; j++){

        for(int x = 0; x < windowSize; x++){
            for(int y = 0; y < windowSize; y++){

                ImageType::IndexType pixelIndex={{i-edge+x,j-edge+y}};
                value += image->GetPixel(pixelIndex);

            }
        }
        ImageType::IndexType pixelIndex={{i,j}};
        int newPixel = round((value * 1.0)/(windowSize * windowSize));
        filterImage->SetPixel(pixelIndex, newPixel);
        value = 0;
    }
}
```

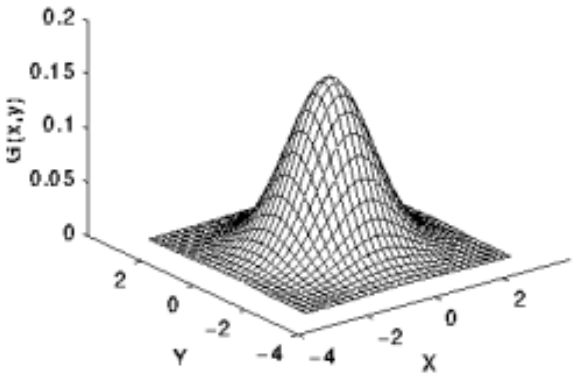
FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro Gaussiano.
    - Utiliza a função gaussiana para o cálculo dos coeficientes da máscara.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

1/16 x

1	2	1
2	4	2
1	2	1



Máscara (sigma = 1)

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro Gaussiano.
    - Geração da máscara do filtro Gaussiano.

```
% Função gaussiana 2D com média
% zero e desvio sigma
function g = GaussXY(x,y,sigma)

  fator = (1/sqrt(2*pi*(sigma^2)));

  g = fator*exp( -(x.^2 + y.^2) ...
              /(2*(sigma.^2)));

end
```

```
% Máscara 5x5
result = zeros(5,5);
a = 2; b = 2;

sigma = 1;

for x=-a:a
  for y=-b:b
    result(x+a+1,y+b+1) ...
      = GaussXY(x,y,sigma);
  end
end

%normalizando o resultado
result = result./sum(result(:));
```

result ~=

$10^{-3} *$

23	34	38	34	23
34	49	56	49	34
38	56	63	56	38
34	49	56	49	34
23	34	38	34	23

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro Gaussiano.

5x5

11x11

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro Gaussiano.
    - Código no OpenCV:

C++: void **GaussianBlur**(InputArray **src**, OutputArray **dst**, Size **ksize**, double **sigmaX**, double **sigmaY**=0, int **borderType**=BORDER_DEFAULT)

Python: `cv2.GaussianBlur`(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst

Parameters:
- **src** – input image; the image can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – Gaussian kernel size. `ksize.width` and `ksize.height` can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from `sigma*` .
- **sigmaX** – Gaussian kernel standard deviation in X direction.
- **sigmaY** – Gaussian kernel standard deviation in Y direction; if `sigmaY` is zero, it is set to be equal to `sigmaX`, if both sigmas are zeros, they are computed from `ksize.width` and `ksize.height` , respectively (see `getGaussianKernel()` for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of `ksize`, `sigmaX`, and `sigmaY`.
- **borderType** – pixel extrapolation method (see `borderInterpolate()` for details).

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro Gaussiano.
    - Código no OpenCV:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('opencv_logo.png')

blur = cv2.GaussianBlur(img,(5,5),0)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

FILTRAGEM

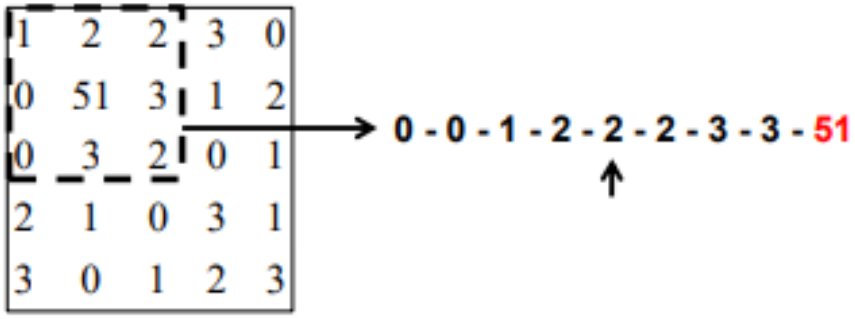
- Filtros de suavização (filtros passa-baixa):
  - Filtro Gaussiano:

```
int mask[9] = {1,2,1,2,4,2,1,2,1};
int edge = windowSize / 2;
vector <int> neighbors;
```

```
for(int i = edge; i < width-edge; i++){
    for(int j = edge; j < height-edge; j++){

        for(int x = 0; x < windowSize; x++){
            for(int y = 0; y < windowSize; y++){

            ImageType::IndexType pixelIndex={{i-edge+x,j-edge+y}};
            neighbors.push_back(image->GetPixel(pixelIndex));

            }
        }

        for(int k = 0; k <neighbors.size(); k++){
            value += neighbors.at(k) * mask[k];
        }

        result = (value/16);
        int newPixel = round(result);
        if(newPixel < 0) newPixel = 0;
        if(newPixel > 255) newPixel = 255;

        ImageType::IndexType pixelIndex={{i,j}};
        filterImage->SetPixel(pixelIndex, newPixel);

        neighbors.clear();
        value = 0;
    }
}
```

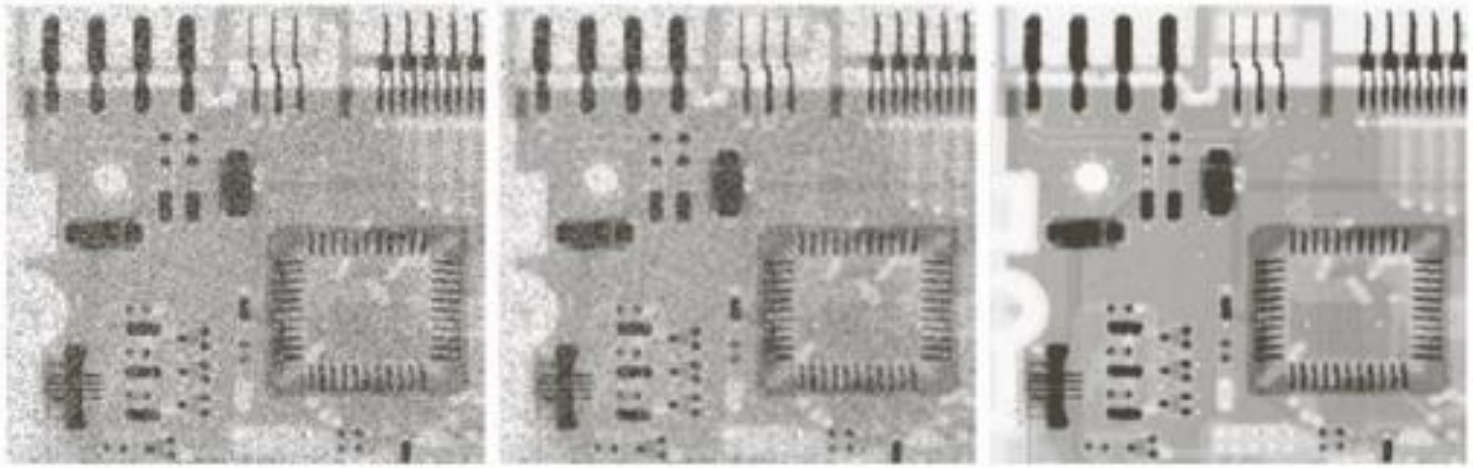
FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da mediana:
    - Valor que ocupa a posição central de um conjunto ordenado de forma crescente.
    - Trata-se de um filtro não linear, pois não é feita a convolução de uma máscara.
    - Aplicado para redução de ruídos.



FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da mediana:



FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da mediana:
    - Código no OpenCV:

C++: void **medianBlur**(InputArray **src**, OutputArray **dst**, int **ksize**)

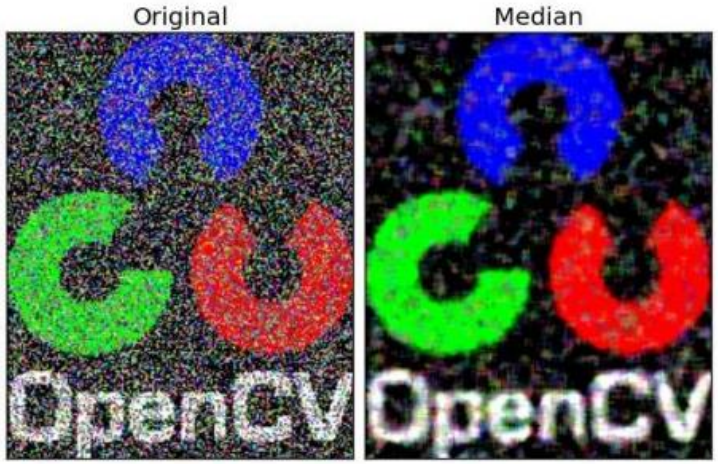
Python: cv2.**medianBlur**(src, ksize[, dst]) → dst

Parameters:
- **src** – input 1-, 3-, or 4-channel image; when `ksize` is 3 or 5, the image depth should be `CV_8U`, `CV_16U`, or `CV_32F`, for larger aperture sizes, it can only be `CV_8U`.
- **dst** – destination array of the same size and type as `src`.
- **ksize** – aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...

FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da mediana:
    - Código no OpenCV:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('opencv_logo.png')

median = cv2.medianBlur(img,5)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```



FILTRAGEM

- Filtros de suavização (filtros passa-baixa):
  - Filtro da mediana:

```
int edge = windowSize / 2;
vector <int> neighbors;

for(int i = edge; i < width-edge; i++){
    for(int j = edge; j < height-edge; j++){

        for(int x = 0; x < windowSize; x++){
            for(int y = 0; y < windowSize; y++){

            ImageType::IndexType pixelIndex={{i-edge+x,j-edge+y}};
            neighbors.push_back(image->GetPixel(pixelIndex));

            }
        }

        sort(neighbors.begin(), neighbors.end());

        int newPixel = (neighbors.at((neighbors.size()/2)));

        ImageType::IndexType pixelIndex={{i,j}};

        filterImage->SetPixel(pixelIndex, newPixel);

        neighbors.clear();
    }
}
```

REFERÊNCIAS

- GONZALEZ, Rafael C.; WOODS, Richard C. **Processamento digital de imagens**. Pearson, 2011.

- PEDRINI, Hélio; SCHWARTZ, William Robson. **Análise de imagens digitais: princípios, algoritmos e aplicações**. Thomson Learning, 2008.

- SILVA, Aristófanes. **Notas de aula da disciplina Processamento de Imagens da Universidade Federal do Maranhão**. 2018.

- BRAZ Jr, Geraldo. **Notas de aula da disciplina Visão Computacional da Universidade Federal do Maranhão**. 2018.