Processamento de Imagens

Prof. MSc. Daniel Menin Tortelli

e-mail: danielmenintortelli@gmail.com
Skype: daniel.menin.tortelli

Site: http://sites.google.com/site/danielmenintortelli/home

Aplicação de Limiar

Aplicação de Limiar

- Outra transformação pontual básica é a aplicação de limiar (*thresholding*), também referido como **limiarização**.

- Esta transformação produz uma imagem binária a partir de uma imagem em escala de cinza: ***valores de pixel são limitados a 1 ou 0 dependendo se estão acima ou abaixo do valor de limiar***.

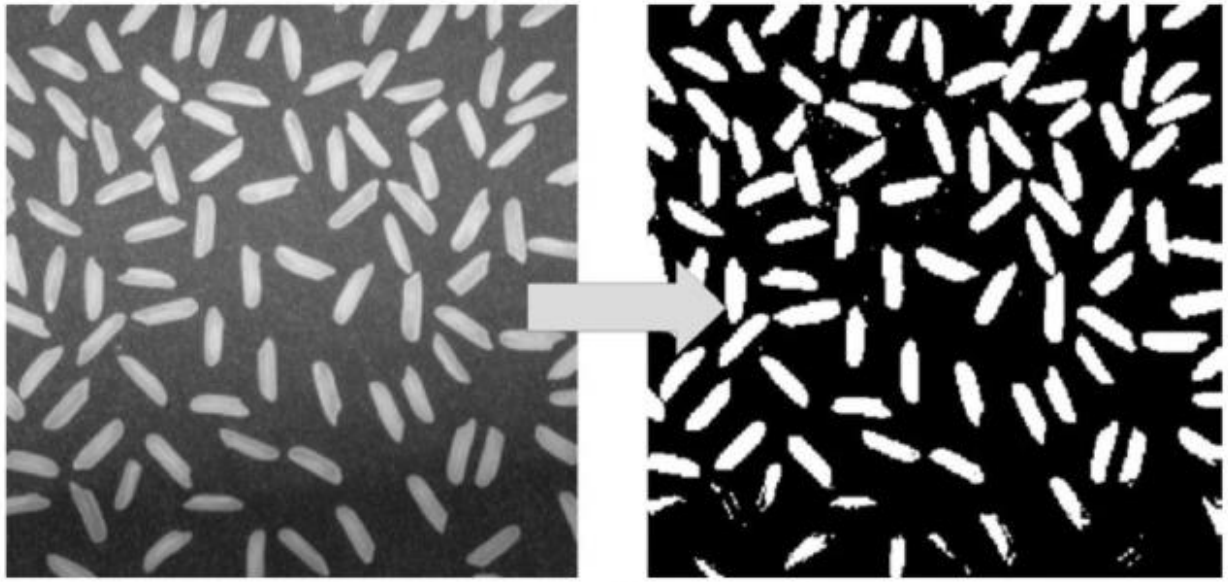
- Isso é comumente usado para separar ou segmentar uma região ou objeto em uma imagem com base nos valores de pixel.

Aplicação de Limiar

- Em sua forma básica, a aplicação de limiar funciona da seguinte forma:

$$G(x,y) = \begin{cases} I(x,y) \geq T & 1 \\ I(x,y) < T & 0 \end{cases}$$

```
cada pixel I(i,j) na imagem I
    if I(i, j) > limiar
        I(i,j) = 1
    else
        I(i,j) = 0
    end
end
```

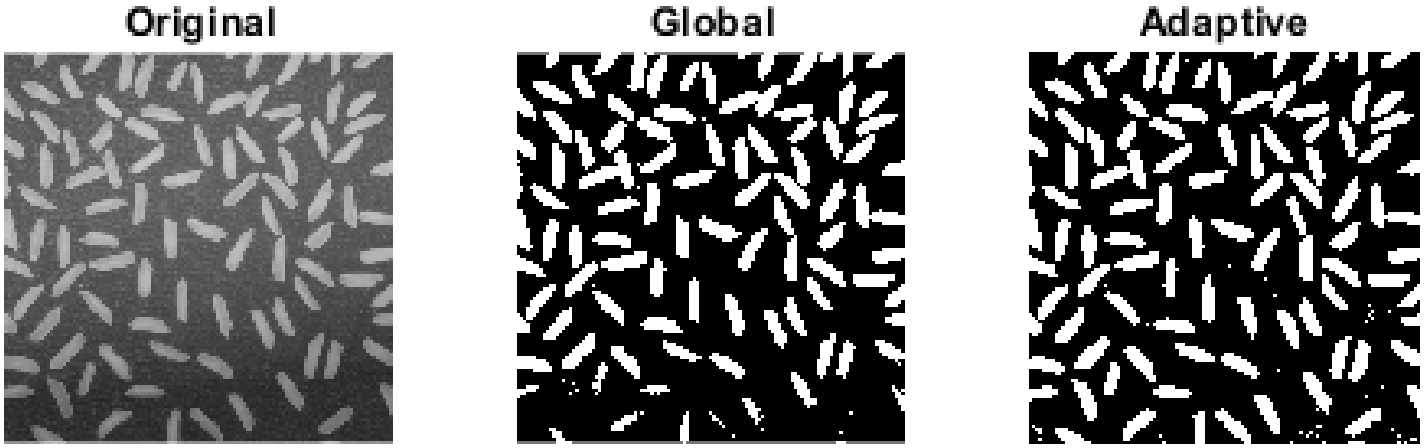


Aplicação de limiar para identificação de objetos.

Aplicação de Limiar (imbinarize)

- No Matlab, essa operação pode ser efetuada com a função **imbinarize**:

```
I = imread('rice.png');
I_global = imbinarize(I, 'global');
I_adaptative = imbinarize(I, "adaptive");
subplot(1,3,1), imshow(I); title('Original');
subplot(1,3,2), imshow(I_global); title('Global');
subplot(1,3,3), imshow(I_adaptative); title('Adaptive');
```



Aplicação de Limiar (graythresh)

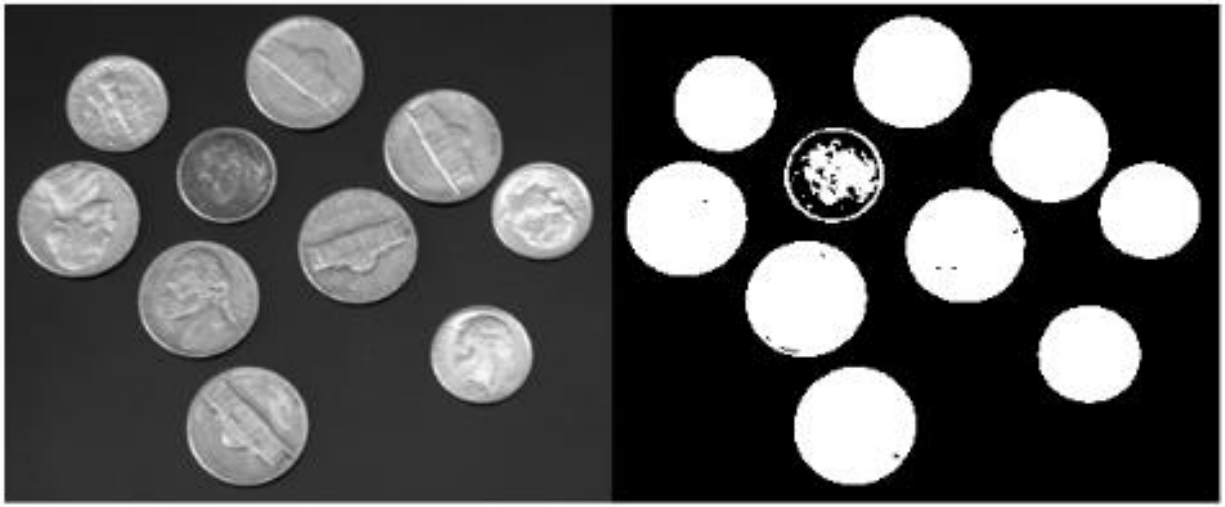
- A função **graythresh** calcula o limiar global **T** de uma imagem em escala de cinza **I**, usando o método de **Otsu**.

- O método de Otsu é um algoritmo de limiarização, proposto por Nobuyuki Otsu.

- Seu objetivo é, a partir de uma imagem em tons de cinza, determinar o valor ideal de um *threshold* que separe os elementos do fundo e da frente da imagem em dois clusters, atribuindo a cor branca ou preta para cada um deles.

Aplicação de Limiar (graythresh)

```
% Lê uma imagem em escala de cinza
I = imread('coins.png');

% Calcula o threshold usando graythresh.
level = graythresh(I);  % level = 0.4941

% Converte a imagem em uma imagem binária, usando o threshold
BW = imbinarize(I,level);

% Mostra a imagem original e a imagem resultante da limiarização
imshowpair(I,BW,'montage');
```



Técnicas de Melhoramento de Imagens Digitais

Melhoramento de Imagens Digitais

- Em alguns casos, uma imagem possui informações inúteis ou insuficientes para extrair objetos devido a diferentes defeitos.

- Portanto, a imagem deve ser processada usando diferentes técnicas de processamento de imagens digitais para remover os defeitos ou artefatos.

- Algumas técnicas podem ser usadas para aumentar a visibilidade e diminuir os defeitos.

Brilho

- O brilho de uma imagem pode ser ajustado pela adição ou subtração de um certo valor de intensidade de um pixel.

- A equação abaixo define o ajuste de brilho em uma imagem digital:

$$G(i,j) = F(i,j) + b \qquad \begin{matrix} b > 0 \; Brightness \; incerease \\ b < 0 \; Brightness \; decrease \end{matrix}$$

Brilho

- **imadjust** – função do Matlab que ajusta o valor de intensidade de um pixel.

```
I = imread('pout.tif');
I_adjustAUTO = imadjust(I);
I_adjustMANUAL = imadjust(I, [0.2; 0.5], [0.0; 1.0]);
subplot(3,1,1); imshow(I); title('Imagem Original');
subplot(3,1,2); imshow(I_adjustAUTO); title('Brilho Automático');
subplot(3,1,3); imshow(I_adjustMANUAL); title('Brilho Manual');
```

Imagem Original

Brilho Automático

Brilho Manual

$$I\_adjust = imadjust(I, [low\_in; hig\_in], [low\_out; high\_out])$$

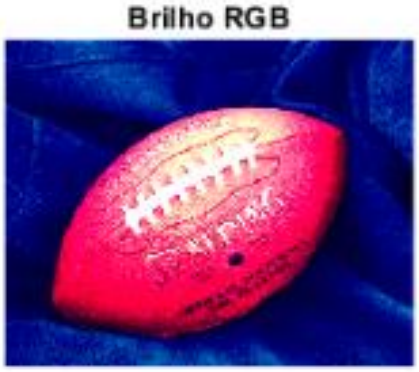
- Mapeia os valores de intensidade dos pixels da imagem **I** para novos valores em **I_adjustMANUAL**, tal que os valores entre ***low_in*** e ***high_in*** são substituídos pelos valores entre ***low_out*** and ***high_out***.

Brilho

- É possível também mudar o valor de intensidade de brilho de imagens RGB, modificando a intensidade de cada um dos canais de cor do pixel.

O ajuste abaixo:

```
I = imread('football.jpg');
I_adjustRGB = imadjust(I, [0.2 0.3 0.0; 0.4 0.5 0.7], []);
subplot(3,1,1); imshow(I); title('Imagem Original');
subplot(3,1,2); imshow(I_adjustRGB); title('Brilho RGB');
```



é equivalente a:

```
I = imread('football.jpg');
I_adjustRGB = imadjust(I, [0.2 0.3 0.0; 0.4 0.5 0.7], [0.0 0.0 0.0; 1.0 1.0 1.0]);
subplot(3,1,1); imshow(I); title('Imagem Original');
subplot(3,1,2); imshow(I_adjustRGB); title('Brilho RGB');
```

Standard Deviation Based Image Stretching

```
% Lê uma imagem em escala de cinza
I = imread('pout.tif');

% Calcula o desvio padrão e a média da imagem
n = 2;
Idouble = im2double(I); % converte os valores dos pixels para double (0 = 0.0 até 255 = 1.0)
avg = mean2(Idouble); % calcula a média dos pixels da matriz
sigma = std2(Idouble); % calcula o desvio padrão dos pixels da matriz

% Ajusta o contraste da imagem, baseado no desvio padrão
J = imadjust(I, [avg-n*sigma avg+n*sigma], []);

subplot(1,2,1); imshow(I); title('Original Image');
subplot(1,2,2); imshow(J); title('Standard Deviation Based Image Stretching');
```

A média (M_e) é calculada somando-se todos os valores de um conjunto de dados e dividindo-se pelo número de elementos deste conjunto. Sendo,

$$M_e = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n}$$

M_e: média

$x_1, x_2, x_3, \ldots, x_n$: valores dos dados

n: número de elementos do conjunto de dados

O desvio padrão (DP) é calculado usando-se a seguinte fórmula:

$$DP = \sqrt{\frac{\sum_{i=1}^{n}(x_i - M_A)^2}{n}}$$

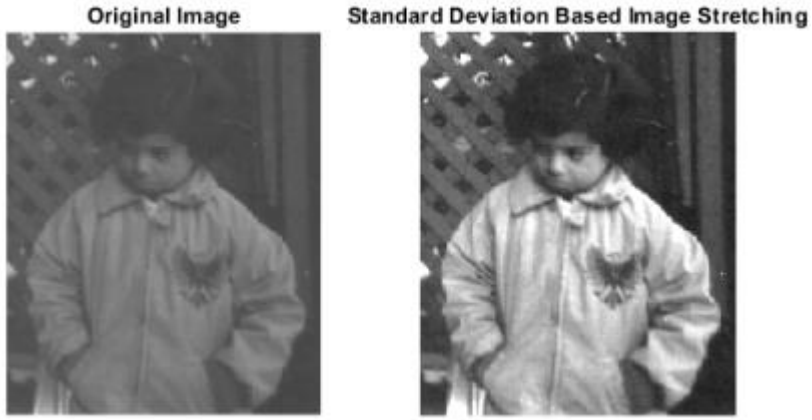
Sendo,

$\sum$: símbolo de somatório. Indica que temos que somar todos os termos, desde a primeira posição (i=1) até a posição n

x_i: valor na posição **i** no conjunto de dados

M_A: média aritmética dos dados

n: quantidade de dados



Contraste

- O contraste em uma imagem pode ser ajustado pela multiplicação de todos os pixels de por um certo valor.

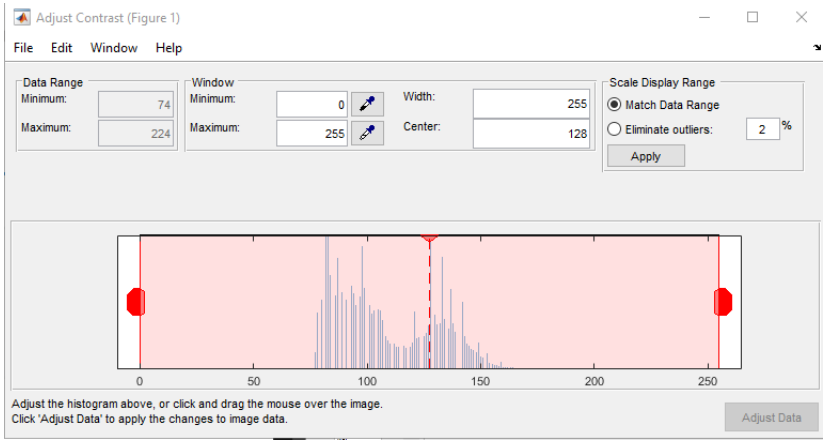
- A equação abaixo define o ajuste de contraste em uma imagem digital:

$$G(i,j) = c * F(i,j) \begin{matrix} c > 0 \, contrast \, increase \\ c < 0 \, contrast \, decrease \end{matrix}$$

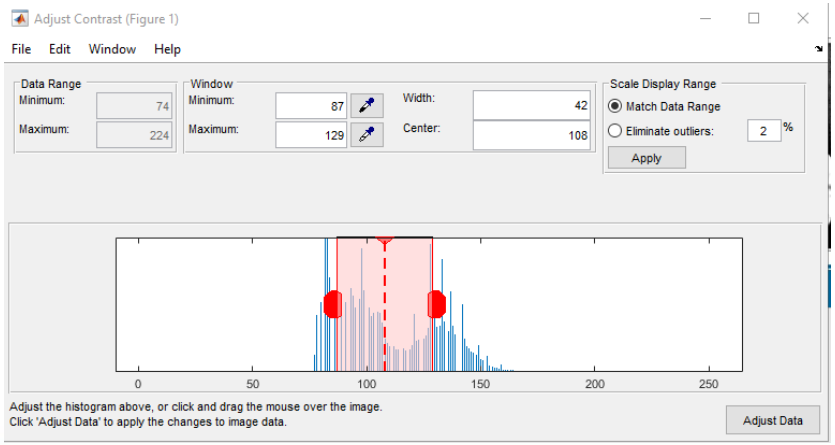
Contraste

- **imcontrast** – função usada para modificar o contraste de uma imagem, através do histograma.

```
Command Window
>> clear
I = imread('pout.tif');
imshow(I);
imcontrast();
>>
```



- Observação:
  - Os códigos devem ser feitos via *Command Window...*



Negativo

- Os valores de intensidade dos pixels são invertidos linearmente para formar uma imagem negativa.

$$G(x,y) = 255 - F(x,y)$$

```
I = imread("pout.tif");
I_neg = imcomplement(I);
subplot(1,2,1); imshow(I); title('Original Image');
subplot(1,2,2); imshow(I_neg); title('Negative Image');
```

Negativo

- Os valores de intensidade dos pixels são invertidos linearmente para formar uma imagem negativa.

Método 1:

```
a = imread('DuasCaras1.tif');
d(:,:,3) = 255 - a(:,:,3);
d(:,:,2) = 255 - a(:,:,2);
d(:,:,1) = 255 - a(:,:,1);
d = fliplr(d);
imshow([a,d]);
```

Método 3:

```
a = imread('DuasCaras1.tif');
d=a;
for row=1:size(a,1)
    for col=1:size(a,2)
        d(row,col,:)=255-a(row,col,:);
    end
end
d = fliplr(d);
imshow([a,d]);
```

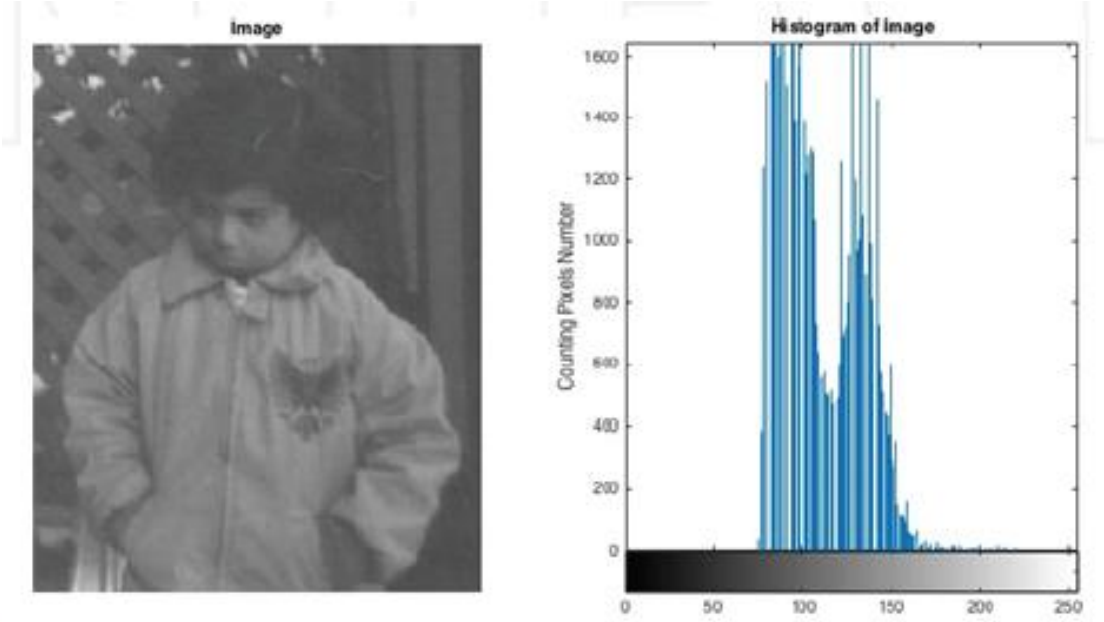
Método 2:

```
a = imread('DuasCaras1.tif');
d=255-a;
d = fliplr(d);
imshow([a,d]);
```

Histograma

Histograma

- Um histograma apresenta a contagem dos valores dos pixels e sua distribuição entre a escala de valores possíveis de intensidade da imagem. Cada pixel, em uma imagem de escala de cinza de 8bits, pode ter valores entre 0 e 255.



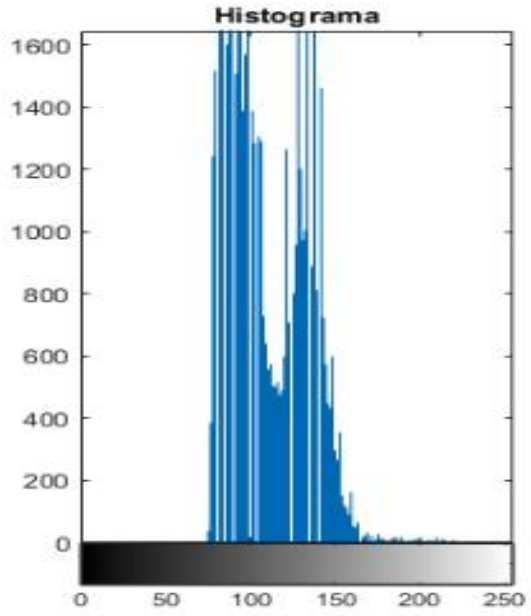
Gray value	Counting of pixel number
75	2
76	38
77	0
78	389
79	1245
80	0
81	1518

Histogram of Specific Gray Values.

Histograma

- No Matlab, o histograma de uma imagem pode ser calculado usando a função **imhist(*image*)**:

```
I = imread('pout.tif');
I_hist = imhist(I);
subplot(1,2,1); imshow(I); title('Imagem');
subplot(1,2,2); imhist(I); title('Histograma');
```



Variables - I_hi

I_hist

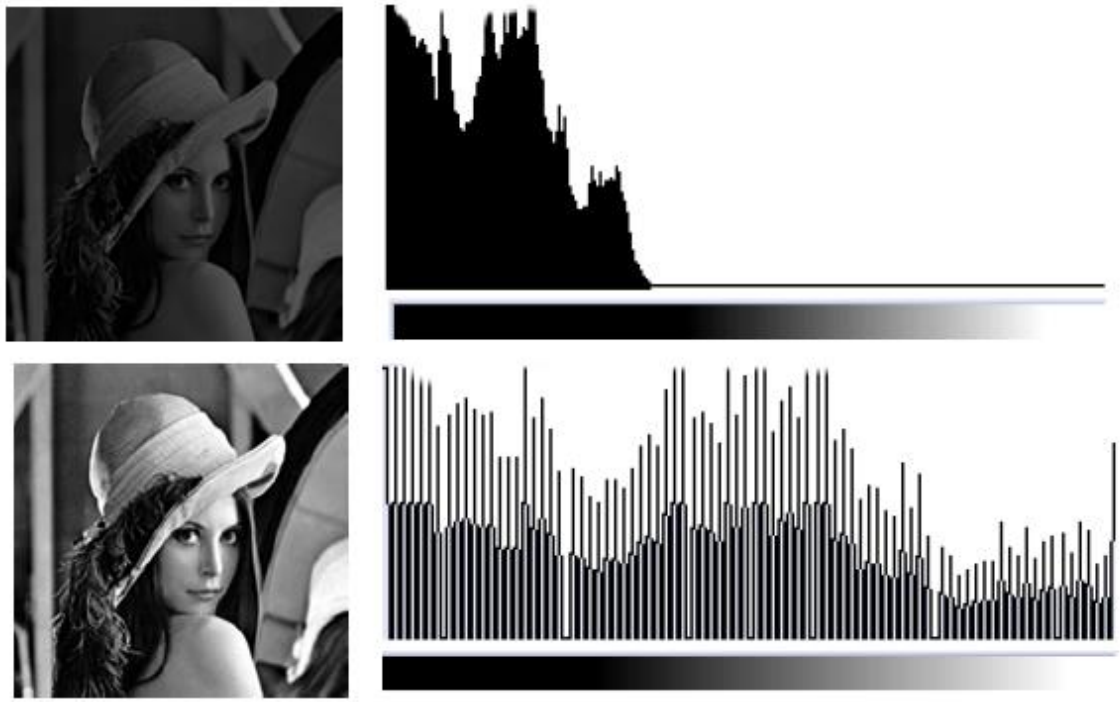
256x1 double

	1
76	38
77	0
78	389
79	1245
80	0
81	1518
82	0
83	3628
84	3146
85	2041
86	0
87	1601
88	2415
89	0
90	1684
91	0
92	1506
93	0
94	1813
95	1660
96	1390
97	0

Equalização de Histograma

- O processo de *equalização de histograma* visa o aumento da uniformidade da distribuição de níveis de cinza de uma imagem.

- Usualmente empregado para realçar diferenças de tonalidade na imagem e resultando, em diversas aplicações, em um aumento significativo no nível de detalhes perceptíveis.



Equalização de Histograma

- A equalização de histograma é uma técnica para ajuste de contraste de uma imagem, usando todos os valores dos pixels.

- Assumindo a imagem 5x5 abaixo, com os valores dos pixels variando entre 0 a 7:

1	1	1	1	1
3	3	0	0	2
3	3	2	2	2
6	4	4	2	6
6	7	7	5	5

Original Image

- <u>Passo 1</u>: Calcular o histograma da imagem:

I	0	1	2	3	4	5	6	7
f(I)	2	5	5	4	2	2	3	2

Equalização de Histograma

- Passo 2: Calcular a Distribuição Cumulativa de Frequências:

I	0 1	2	3	4	5	6	7
f(I)	2 5	5	4	2	2	3	2
CFD	2 2+5=7	7+5=12	12+4=16	16+2=18	18+2=20	20+3=23	23+2=25

- Passo 3: Calcular a nova cor de cada pixel da imagem com a equação:

$$h(v) = floor\left(\frac{CFD(v) - CFD_{min}}{(MxN) - CFD_{min}} x(L-1) \right)$$

Onde:

h é o novo valor do pixel

v é o número do pixel

MxN são o número de linhas e colunas da imagem

L é o valor máximo que um valor de nível de cinza pode assumir (no caso do exemplo L = 8)

Equalização de Histograma

- Por exemplo, se for calculado para o pixel com valor 4, teremos:

$$h(4) = floor((16-2)/((5x5)-2)x(8-1)) \cong floor(4,26) \cong 4$$

1	1	1	1	1
3	3	0	0	2
3	3	2	2	2
6	4	4	2	6
6	7	7	5	5

Original Image

1	1	1	1	1
4	4	0	0	3
4	4	3	3	3
6	4	4	3	6
6	7	7	5	5

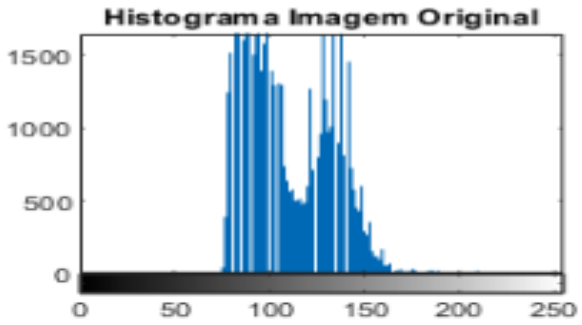
New Image

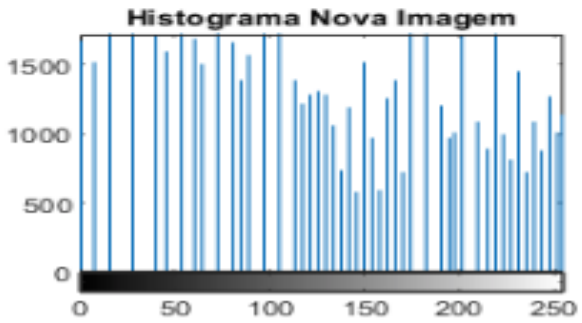
- Depois que todos os pixels da imagem forem recalculados, o resultado terá a distribuição...

I	0	1	2	3	4	5	6	7
f(I)	2	5	5	4	2	2	3	2
CFD	2	7	12	16	18	20	23	25
h	0	1	3	4	4	5	6	7

Equalização de Histograma

```
I = imread('pout.tif');
I_histeq = histeq(I);
subplot(2,2,1); imshow(I); title('Imagem Original');
subplot(2,2,2); imhist(I); title('Histograma Imagem Original');
subplot(2,2,3); imshow(I_histeq); title('Nova Imagem');
subplot(2,2,4); imhist(I_histeq); title('Histograma Nova Imagem');
```





Realce de Imagens

Realce de Imagens

- O objetivo básico da aplicação de realce é processar a imagem de modo que possamos ver e avaliar a informação visual nela contida com maior clareza.

- O realce de imagens é, portanto, muito subjetivo, pois depende fortemente da informação específica que o observador espera extrair da imagem.

- O principal objetivo de realce de imagens é o processamento de uma imagem de forma a torná-la visualmente mais aceitável ou agradável.

Realce por meio da filtragem de imagens

- A remoção de ruído, realce de bordas da imagem e o efeito de ‘foco suave’ (*blurring*), são exemplos de técnicas de realce de imagens.

- *Essas e outras operações de realce podem ser implementadas pelo processo de* ***filtragem no domínio espacial***.

- A filtragem no domínio espacial indica que o processo de filtragem ocorre diretamente nos pixels de uma imagem.

Realce por meio da filtragem de imagens

- Filtros atuam sobre uma imagem para alterar os valores de pixel de alguma forma especificada e, em geral, são classificados em dois tipos: lineares e não lineares.

- Os pixels em uma imagem − o pixel em consideração em um dado momento é referido como pixel-alvo − são tratados sucessivamente.

- O valor do pixel-alvo é, então, substituído por um novo valor que depende apenas dos valores de pixels em uma dada vizinhança em torno do pixel-alvo.

Vizinhança

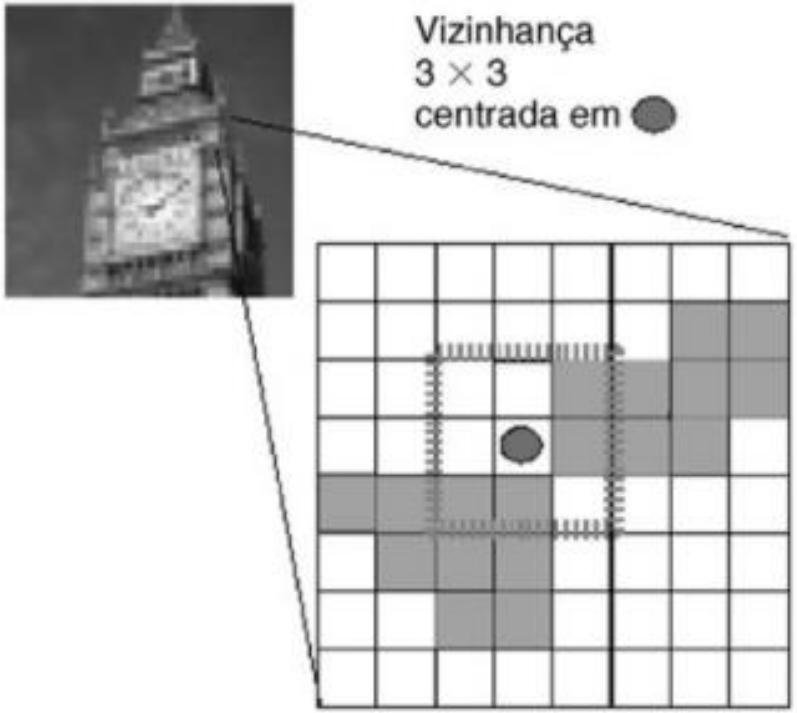
- Uma importante medida em imagens é o conceito de ***conectividade***.

- No processamento de imagens, muitas operações usam o conceito de ***vizinhança*** de imagem local para definir uma área local de influência, relevância ou interesse.

- Central à questão de definição de vizinhança local é a noção de conectividade de pixels, ou seja, a determinação de quais pixels estão conectados a outros.

Vizinhança

NO	N	NE
O	(i,j)	L
SO	S	SE



Conectividade de vizinhança (à esquerda) e um exemplo de uma vizinhança 3 × 3 centrada em uma posição específica de pixel da imagem.

Vizinhança

- Um *pixel* ***p***, de coordenadas ***(x,y)***, tem 4 vizinhos horizontais e verticais, cujas coordenadas são:

$$(x+1, y), (x-1, y), (x, y+1) \text{ e } (x, y-1).$$

Estes *pixels* formam a chamada "**4-vizinhança**" de ***p***, que será designada $N_4(\boldsymbol{p})$.
Uma conectividade 4 implica que somente os pixels que se encontram ao N, O, L, S do pixel em questão estão conectados.

- Os quatro vizinhos diagonais de ***p*** são os *pixels* de coordenadas

$$(x-1, y-1), (x-1, y+1), (x+1, y-1) \text{ e } (x+1, y+1),$$

que constituem o conjunto $N_d(\boldsymbol{p})$.

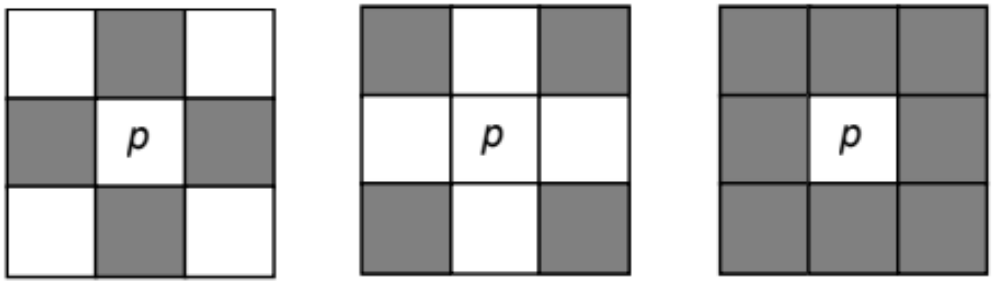
- Se os pixels nas diagonais também estiverem conectados, teremos conectividade 8 (ou seja, pixels ao N, NO, O, NE, SE, L, SO, S estão conectados.

A "8-vizinhança" de **p** é definida como:

$$N_8(p) = N_4(p) \cup N_d(p)$$

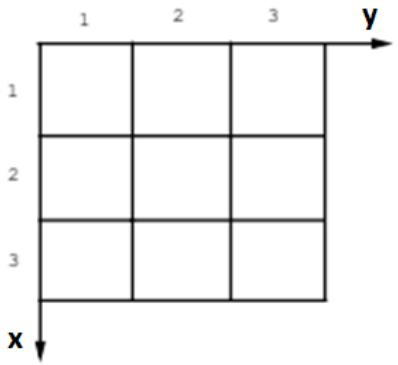
Vizinhança

- Os vários tipos de vizinhança estão ilustrados na figura abaixo:



Conceitos de 4-vizinhança, vizinhança diagonal e 8-vizinhança.

- Operações efetuadas localmente em imagens, como filtragem e detecção de borda, consideram uma dada posição de pixel *(i, j)* em termos de sua vizinhança local de pixels indexada como um deslocamento *(i ± x, j ± y)*



f(x - 1, y - 1)	f(x - 1, y)	f(x - 1, y + 1)
f(x, y - 1)	p	f(x, y + 1)
f(x + 1, y - 1)	f(x + 1, y)	f(x + 1, y + 1)

Aplicando filtros na imagem

- A maioria das técnicas de processamento de imagens agora usa conectividade-8 por *default*;
- Operações de filtragem em toda uma imagem são, em geral, efetuadas como uma série de operações em vizinhanças locais com o emprego do principio de *janela deslizante.*
- Ou seja, cada pixel na imagem é processado com base em uma operação realizada em sua vizinhança local de N x N pixels (região de influência).

Aplicando filtros na imagem

```
A = imread('cameraman.tif'); % Lê a imagem original
subplot(2,2,1), imshow(A); title('Imagem Original'); % Exibe a imagem

func1 = @(x) max(x(:)); % Especifica o filtro a ser aplicado
B = nlfilter(A,[3 3],func1); % Aplica o filtro na vizinhança 3x3
subplot(2,2,2), imshow(B); title('MAX'); % Exibe a imagem resultante

func2 = @(x) min(x(:));
C = nlfilter(A,[3 3],func2);
subplot(2,2,3), imshow(C); title('MIN');

func3 = @(x) uint8(mean(x(:)));
D = nlfilter(A,[3 3],func3);
subplot(2,2,4), imshow(D); title('MEAN');
```

- Neste exemplo, especificamos a função ***func()*** como a função de filtro ***max()*** a ser aplicada a cada vizinhança 3x3 da imagem.
- Isso substitui, na imagem de saída, todos os pixels de entrada pelo máximo valor de pixel da vizinhança de pixels de entrada.
- Também foram criadas outras funções para o ***min()*** e ***mean()***

Min_Max_Mean

```
% Reinicia Workspace
clear

% Carrega uma imagem grayscale
A = imread('cameraman.png');
A_Min = A;
A_Max = A;
A_Mean = A;

% Varre a imagem e recalcula a intensidade dos pixels
for i = 2:size(A)-2
    for j = 2:size(A)-2
        mask = [
            A(i-1,j-1)  A(i-1,j)  A(i-1,j+1)
            A(i,j-1)    A(i,j)    A(i,j+1)
            A(i+1,j-1)  A(i+1,j)  A(i+1,j+1)
        ];

        % Encontra MENOR valor no kernel
        minVal = min(mask,[],'all');
        A_Min(i,j) = minVal;

        % Encontra MAIOR valor no kernel
        maxVal = max(mask,[],'all');
        A_Max(i,j) = maxVal;

        % Calcula MÉDIA dos valores no kernel
        meanVal = mean(mask,'all');
        A_Mean(i,j) = meanVal;

    end
end

subplot(2,2,1), imshow(A); title("Imagem Original");
subplot(2,2,2), imshow(A_Min); title("Filtro Valor MIN");
subplot(2,2,3), imshow(A_Max); title("Filtro Valor MAX");
subplot(2,2,4), imshow(A_Mean); title("Filtro Valor MEAN");
```

Convolação Digital

Convolução Digital

- Convolução é geralmente utilizada para modificar as características espaciais de uma imagem.
- No processo de convolução, o novo valor de um pixel na imagem é dado pela média ponderada de seus pixels vizinhos.
- O valor dos pixels vizinhos do pixel em análise é pesado por uma matriz de coeficientes denominada **máscara de convolução** (*mask*, *kernel*).
- Dependendo da aplicação, a máscara de convolução pode ser de diferentes tamanhos, tais como: 3x3, 5x5, 7x7.

Convolução Digital

- A definição matemática da convolução é a equação:

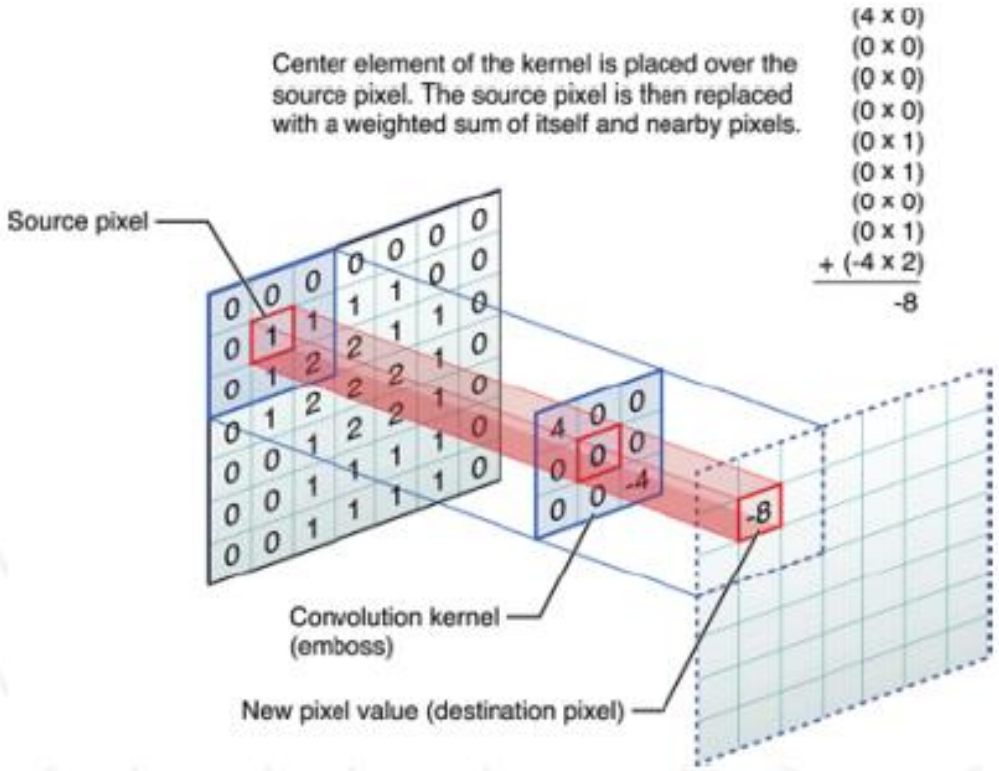
$$f(x,y) = \sum_{i=I_{min}}^{I_{máx}} \sum_{j=J_{min}}^{J_{máx}} w(i,j)I(x+i,y+j)$$

onde:

w é a máscara de convolução

I é a imagem a ser processada

i e **j** são linhas e colunas da imagem.



Núcleos de filtros e a mecânica de filtragem linear (Convolução)

- Em filtros espaciais lineares, o valor novo ou filtrado do pixel-alvo é determinado por meio de uma combinação linear dos valores de pixel em sua vizinhança.

- Qualquer outro tipo de filtro é, por definição, não linear.

- A especificada combinação linear dos pixels vizinhos é determinada pelo núcleo (*kernel*) do filtro (também chamado de máscara).

- Isso é simplesmente um conjunto/subimagem com exatamente as mesmas dimensões da vizinhança que contem os pesos a serem alocados aos correspondentes pixels na vizinhança do pixel-alvo.

Núcleos de filtros e a mecânica de filtragem linear (Convolução)

- Na filtragem, a localização do núcleo é variada sucessivamente, de modo que a posição de seu pixel central coincida com a de cada pixel-alvo, e o valor filtrado seja calculado segundo a escolhida combinação ponderada dos pixels vizinhos ao de alvo.

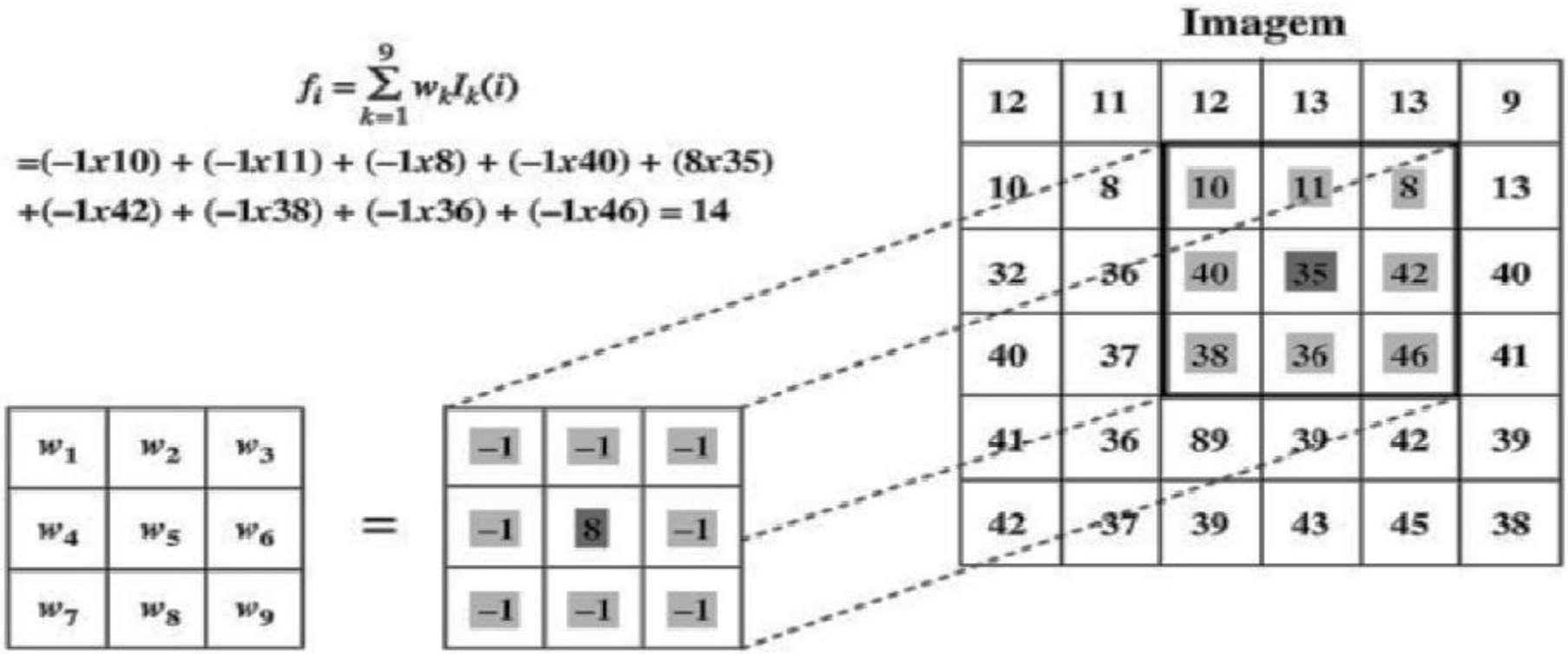
- Esse procedimento de filtragem pode ser visualizado como:
  1. o deslocamento do núcleo (*kernel*, máscara) sobre todas as posições ***(i, j)*** de interesse na imagem original,
  2. a multiplicação dos pixels sob o núcleo pelos correspondentes pesos ***w***,
  3. o cálculo dos novos valores pela soma ponderada e,
  4. a cópia desses valores nas mesmas posições de uma nova imagem (filtrada) ***f***.

Convolução Digital

Mecânica da filtragem de uma imagem com um filtro de núcleo N x N = 3 x 3.

$$f_i = \sum_{k=1}^{9} w_k I_k(i)$$

$$=(-1x10) + (-1x11) + (-1x8) + (-1x40) + (8x35)$$
$$+(-1x42) + (-1x38) + (-1x36) + (-1x46) = 14$$



Convolução discreta. O pixel central do núcleo e o pixel-alvo na imagem são indicados pela sombra em cinza-escuro. O núcleo é 'colocado' na imagem, de modo que o pixel central e o pixel-alvo coincidam. O valor filtrado do pixel-alvo é, então, dado por uma combinação linear dos pixels vizinhos, cujos pesos específicos são determinados pelos valores do núcleo. Neste exemplo, o pixel-alvo, de valor original 35, passou a ter um valor filtrado de 14.

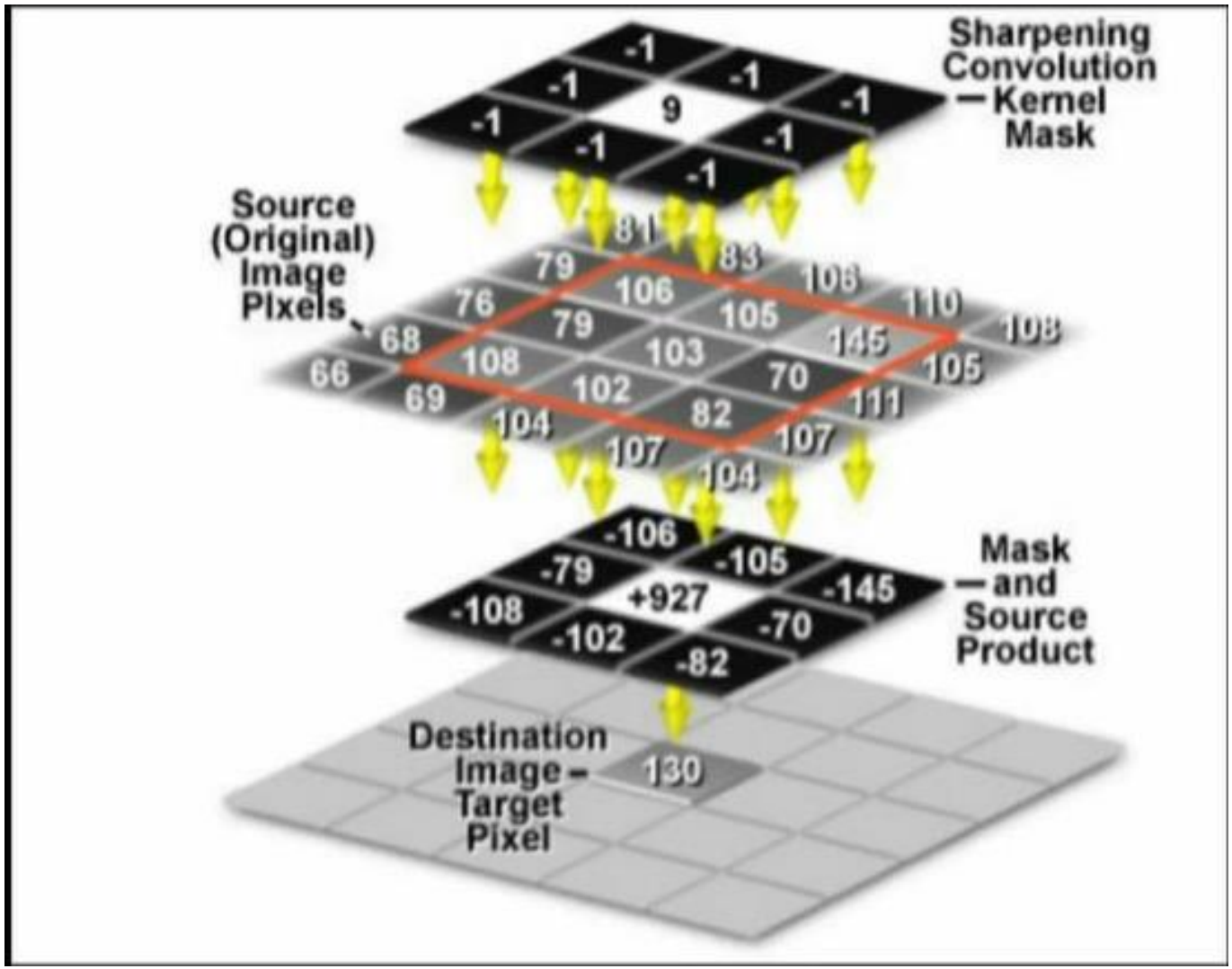
Os passos na filtragem linear (convolução)

Os passos na filtragem linear (convolução) podem ser resumidos da seguinte forma:

1. Definir o núcleo (*kernel*, máscara) do filtro.
2. Deslizar o núcleo sobre a imagem de modo que o pixel central do núcleo coincida com cada pixel-alvo na imagem.
3. Multiplicar os pixels sob o núcleo pelos correspondentes valores (pesos) no núcleo e somar os resultados.
4. Para cada pixel-alvo, copiar o valor resultante na mesma posição de uma nova imagem (filtrada).

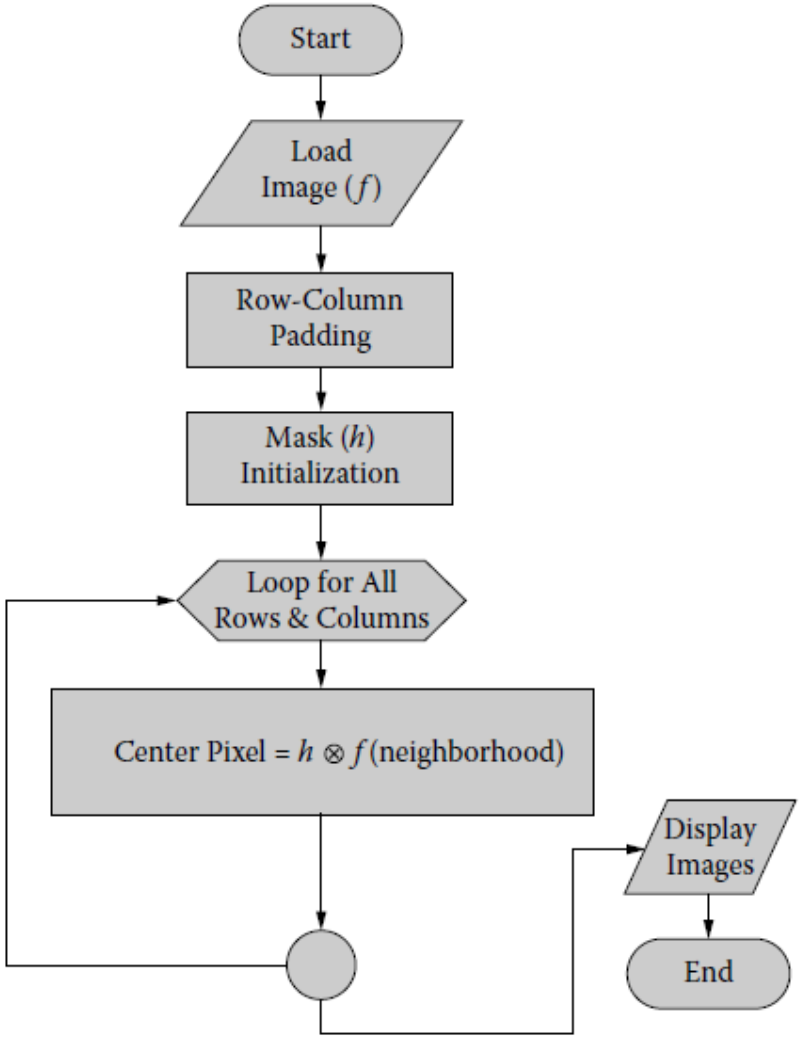
Em certas situações, podemos aplicar um filtro linear a apenas uma dada região da imagem, em vez de aplicá-lo à imagem completa; a isto nos referimos como **filtragem de região**.

Os passos na filtragem linear (convolução)



- Efeitos de processamento de imagem podem ser alcançados com máscaras com diferentes configurações.
- O valor final do pixel alvo (pixel central) vai depender do tamanho da máscara (*kernel*, núcleo) e dos valores dos pesos que a compõem.

Passos na filtragem espacial (algoritmo)



Kernel-based spatial filtering algorithm.

Filtragem linear nas fronteiras da imagem

A aplicação de filtragem às fronteiras da imagem representa um desafio.

Cabe perguntar o que devemos fazer quando os pixels-alvo estão próximos das fronteiras da imagem e o núcleo de convolução se sobrepõe à fronteira.

Em geral, algumas soluções são adotadas para tratar tal situação:

1. Simplesmente deixar inalterados os pixels-alvo que estejam na região de fronteira.
2. Efetuar a filtragem apenas nos pixels que estejam na fronteira (e ajustar a operação de filtragem de modo correspondente).
3. ‘Preencher’ os pixels que faltam na operação de filtragem com cópias dos valores sobre a fronteira.
4. ‘Cortar’ a imagem − ou seja, extrair uma imagem de dimensões reduzidas, da qual são removidos todos os pixels de borda que não tenham sido filtrados adequadamente.

Convolução Digital

- Diferentes tipos de filtros presentes no Matlab *Image Processing Toolbox*:

Value	Description
average	Averaging filter
disk	Circular averaging filter (pillbox)
gaussian	Gaussian low-pass filter
laplacian	Approximates the two-dimensional Laplacian operator
log	Laplacian of Gaussian filter
motion	Approximates the linear motion of a camera
prewitt	Prewitt horizontal edge-emphasizing filter
sobel	Sobel horizontal edge-emphasizing filter

Convolução Digital

```
% Carrega uma imagem
I = imread('onion.png');

% Cria um filtro de média circular (pillbox) dentro
% de uma matriz quadrada de tamanho 2*radius+1
H = fspecial('disk', 10);

% Realiza a filtragem de uma imagem 2D (I) com um filtro
% específico (H) e retorna a imagem resultante.
Blurred = imfilter(I, H, 'replicate');

% Filtro de desfoque de movimento que simula o movimento linear de uma câmera.
% O segundo parâmetro (len) especifica a duração do movimento
% O terceiro parâmetro (theta) especifica o ângulo do movimento em graus no sentido anti-horário.
% O filtro se torna um vetor para movimentos horizontais e verticais.
% O len padrão é 9 e o theta padrão é 0, o que corresponde a um movimento horizontal de nove pixels.
H = fspecial('motion', 20, 45);
MotionBlur = imfilter(I, H, 'replicate');

% Filtro de Sobel
% Retorna um filtro 3x3 que enfatiza as bordas horizontais usando o efeito de
% suavização aproximando um gradiente vertical.
% Para enfatizar arestas verticais, transponha o filtro h'.
%[ 1  2  1
%  0  0  0
% -1 -2 -1 ]
H = fspecial('sobel');
Sobel = imfilter(I, H, 'replicate');

% Exibe os resultados
subplot(2,2,1); imshow(I); title("Imagem Original");
subplot(2,2,2); imshow(Blurred); title("Blur Kernel");
subplot(2,2,3); imshow(MotionBlur); title("Motion Blur Kernel");
subplot(2,2,4); imshow(Sobel); title("Sobel Kernel");
```

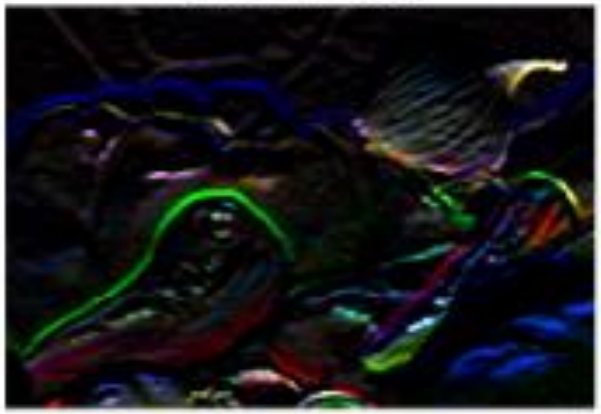
Convolução Digital

Imagem Original

Blur Kernel

Motion Blur Kernel

Sobel Kernel



Convolução Digital

```
% Carrega uma imagem
I = imread('onion.png');

% Cria um filtro de média
% O segundo parâmetro (hsize) especifica o tamanho do filtro. Ex. 3x3
H = fspecial('average', 3);
Average = imfilter(I, H, 'replicate');

% Filtro passa baixo de desfoque gaussiano.
% O segundo parâmetro (hsize) especifica o tamanho do filtro. Ex. 3x3
% O terceiro parâmetro (sigma) especifica desvio padrão da distribuição gaussiana
H = fspecial('gaussian', 3, 0.5);
Gaussian = imfilter(I, H, 'replicate');

% Filtro Laplassiano
% O segundo parâmetro (alpha) [0.0 - 1.0] especifica a forma do Laplaciano
H = fspecial('laplacian', 0.2);
Laplacian = imfilter(I, H, 'replicate');

% Exibe os resultados
subplot(2,2,1); imshow(I); title("Imagem Original");
subplot(2,2,2); imshow(Average); title("Average Kernel");
subplot(2,2,3); imshow(Gaussian); title("Gaussian Kernel");
subplot(2,2,4); imshow(Laplacian); title("Laplacian Kernel");
```

Convolução Digital

Imagem Original

Average Kernel

Gaussian Kernel

Laplacian Kernel

Filtragem para remoção de ruído

Filtragem para remoção de ruído

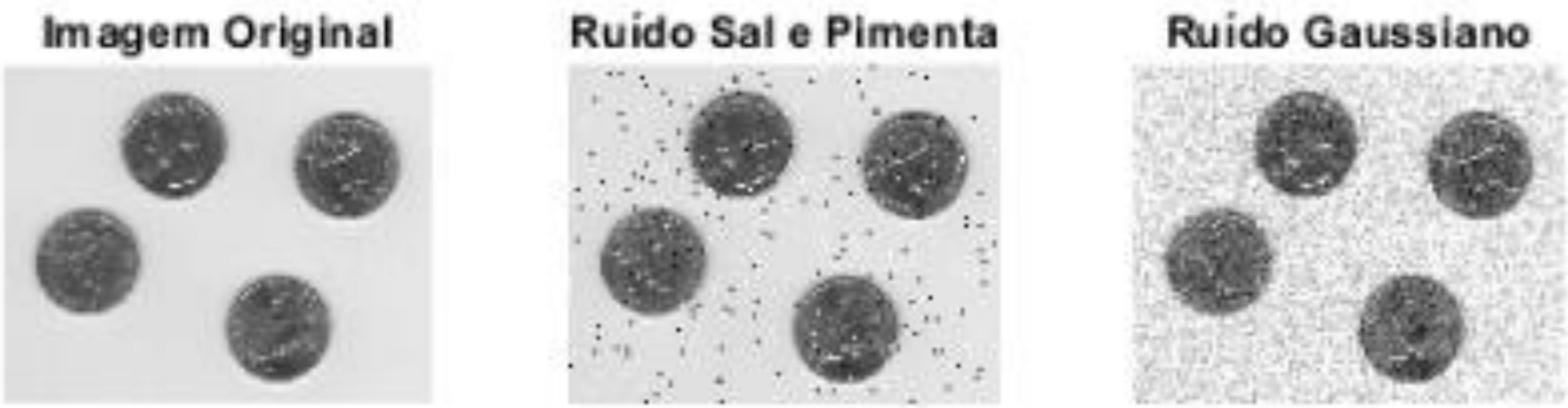
- Um dos principais usos de filtragem linear e não linear em processamento de imagens é na remoção de ruído.

- A seguir, exploraremos a aplicação de alguns filtros diferentes para a remoção de ruído típico, como ruído aditivo ‘sal e pimenta’ e ruído gaussiano.

- No Matlab, a função **imnoise** pode ser usada para adicionar ruídos específicos em uma imagem.

Adicionando ruído na imagem

```
I = imread('eight.tif'); % Lê a imagem
subplot(1,3,1), imshow(I); title('Imagem Original'); % Exibe a imagem

Isp = imnoise(I, 'salt & pepper', 0.03); % Adiciona 3% (0,03) de ruído sal e pimenta
subplot(1,3,2), imshow(Isp); title('Ruído Sal e Pimenta');  % Exibe a imagem resultante
imwrite(Isp, 'eightSaltPepper.tif',"tiff"); % Salva em disco a imagem resultante

Ig = imnoise(I, 'gaussian', 0.02);  % Adiciona ruído gaussiano (com variância 0,02)
subplot(1,3,3), imshow(Ig);  title('Ruído Gaussiano'); % Exibe a imagem resultante
imwrite(Ig, 'eightGaussian.tif',"tiff"); % Salva em disco a imagem resultante
```



Filtragem pela Média

- O filtro de média aplica pesos iguais $\mathbf{w_k}$ a todos os pixels na vizinhança.

- Um peso $\mathbf{W_k = 1/(NM)}$ é usado para uma vizinhança N x M e tem o efeito de suavizar a imagem, dando a cada pixel na imagem de saída o valor médio de sua vizinhança N x M.

$$\frac{1}{9}$$

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

- Filtros de média podem ser usados como um método de suprimir ruído em uma imagem.

- Outro uso comum do filtro de média é como um passo preliminar de processamento para suavizar uma imagem, de modo que operações subsequentes sejam mais eficientes.

Filtragem pela Média

```
k = ones(3,3)/9; % Define o filtro da Média

I = imread('eight.tif');
subplot(2,3,1), imshow(I); title('Imagem Original'); % Exibe a imagem
I_m = imfilter(I, k); % Aplica o filtro da Média na imagem original
subplot(2,3,4), imshow(I_m); title('Original->Media'); % Exibe a imagem

Isp = imread('eightSaltPepper.tif'); % Lê a imagem com ruído Sal e Pimenta
subplot(2,3,2), imshow(Isp); title('Ruído Sal e Pimenta');  % Exibe a imagem ruído Sal e Pimenta
Isp_m = imfilter(Isp, k); % Aplica o filtro da Média na imagem com ruído Sal e Pimenta
subplot(2,3,5), imshow(Isp_m); title('SalPimenta->Media'); % Exibe a imagem

Ig = imread('eightGaussian.tif'); % Lê a imagem com ruído Gaussiano
subplot(2,3,3), imshow(Ig);  title('Ruído Gaussiano'); % Exibe a imagem com ruído Gaussiano
Ig_m = imfilter(Ig, k); % aplica o filtro da Média na imagem com ruído Gaussiano
subplot(2,3,6), imshow(Ig_m); title('Gaussiano->Media'); % Exibe a imagem
```

Por exemplo vejamos a janela 3x3 abaixo :

$$\begin{bmatrix} 244 & 247 & 245 \\ 252 & 12 & 238 \\ 244 & 245 & 250 \end{bmatrix}$$

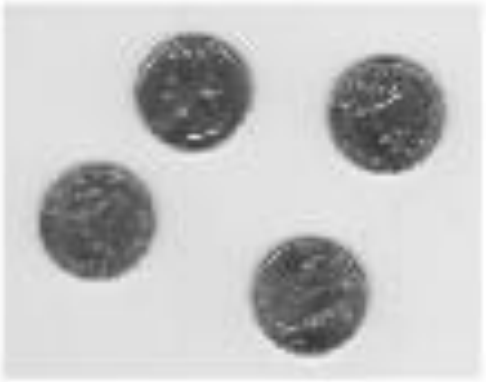
A media dos valores será :
$(12 + 238 + 244 + 244 + 245 + 245 + 247 + 250 + 252)/9 = \mathbf{219}$

Assim o valor desse pixel que era 12 será de 219

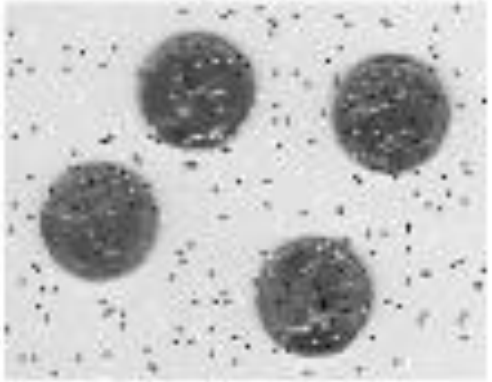
A função **imfilter** do Matlab aplica um filtro definido em uma imagem.

Filtragem pela Média

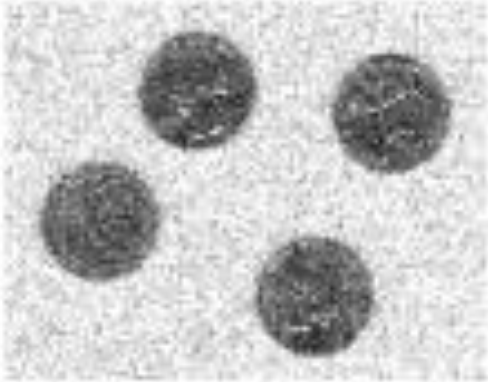
Imagem Original



Ruido Sal e Pimenta

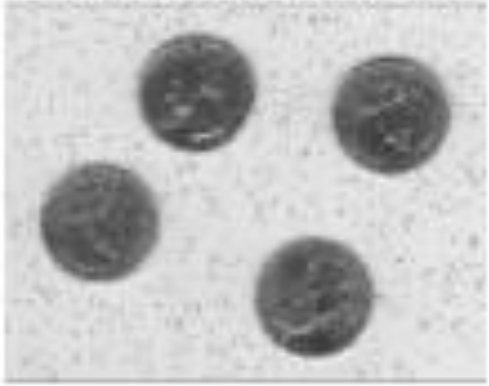


Ruído Gaussiano



Original->Media

SalPimenta->Media



Gaussiano->Media

Filtragem pela Média (considerações)

- A filtragem pela média é razoavelmente eficaz na remoção do ruído gaussiano, às custas de uma perda de detalhes de alta frequência da imagem (bordas).

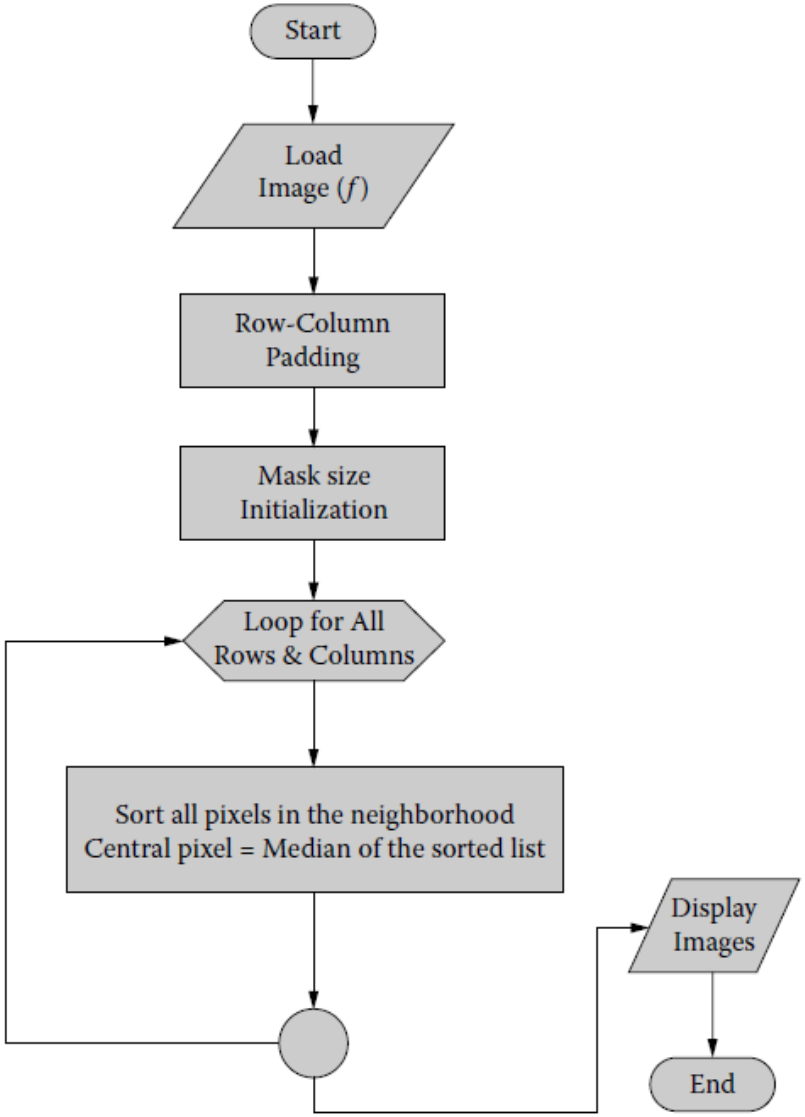
- Em resumo, as principais desvantagens da filtragem pela média são:

  a) não é robusta em relação a grandes distribuições de ruído na imagem (pixels estranhos) tipo sal e pimenta;

  b) causa desfocalização (*blurring*) nas bordas da imagem.

- Devido ao ultimo efeito, um filtro de média pode também ser usado como um filtro passa-baixas genérico.

Filtragem pela Média (algoritmo)



Filtragem pela Mediana

- A filtragem pela mediana supera as principais limitações da filtragem pela média, às custas de maior gasto computacional.
- O valor de cada pixel-alvo é substituído pela **mediana estatística** dos valores dos N x M pixels vizinhos e não pela média.
- A mediana ***m*** de um conjunto de números é o número que divide o conjunto em dois grupos iguais, de modo que metade dos números seja menor do que ***m*** e a outra metade, maior;
- A mediana é o ponto central de uma distribuição ordenada de valores.
- Por ser um valor de pixel obtido da própria vizinhança do pixel-alvo, a mediana é mais robusta em relação a pixels estranhos e não cria valores irreais de pixel. Isso ajuda a prevenir desfocalização de bordas e perda de detalhes da imagem.
- Por definição, o operador mediana requer ordenação dos valores na vizinhança do pixel-alvo, para cada posição de pixel. Isso aumenta as exigências computacionais do operador mediana.

Filtragem pela Mediana

```
I = imread('eight.tif');
subplot(2,3,1), imshow(I); title('Imagem Original'); % Exibe a imagem
I_m = medfilt2(I, [3 3]); % Aplica o filtro da Mediana na imagem original
subplot(2,3,4), imshow(I_m); title('Original->Mediana'); % Exibe a imagem

Isp = imread('eightSaltPepper.tif'); % Lê a imagem com ruído Sal e Pimenta
subplot(2,3,2), imshow(Isp); title('Ruído Sal e Pimenta');  % Exibe a imagem ruído Sal e Pimenta
Isp_m = medfilt2(Isp, [3 3]); % Aplica o filtro da Mediana na imagem com ruído Sal e Pimenta
subplot(2,3,5), imshow(Isp_m); title('SalPimenta->Mediana'); % Exibe a imagem

Ig = imread('eightGaussian.tif'); % Lê a imagem com ruído Gaussiano
subplot(2,3,3), imshow(Ig);  title('Ruído Gaussiano'); % Exibe a imagem com ruído Gaussiano
Ig_m = medfilt2(Ig, [3 3]); % aplica o filtro da Mediana na imagem com ruído Gaussiano
subplot(2,3,6), imshow(Ig_m); title('Gaussiano->Mediana'); % Exibe a imagem
```

Dada a seguinte janela 3x3

$$\begin{bmatrix} 244 & 247 & 245 \\ 252 & 12 & 238 \\ 244 & 245 & 250 \end{bmatrix}$$

Ordenando os valores teremos :

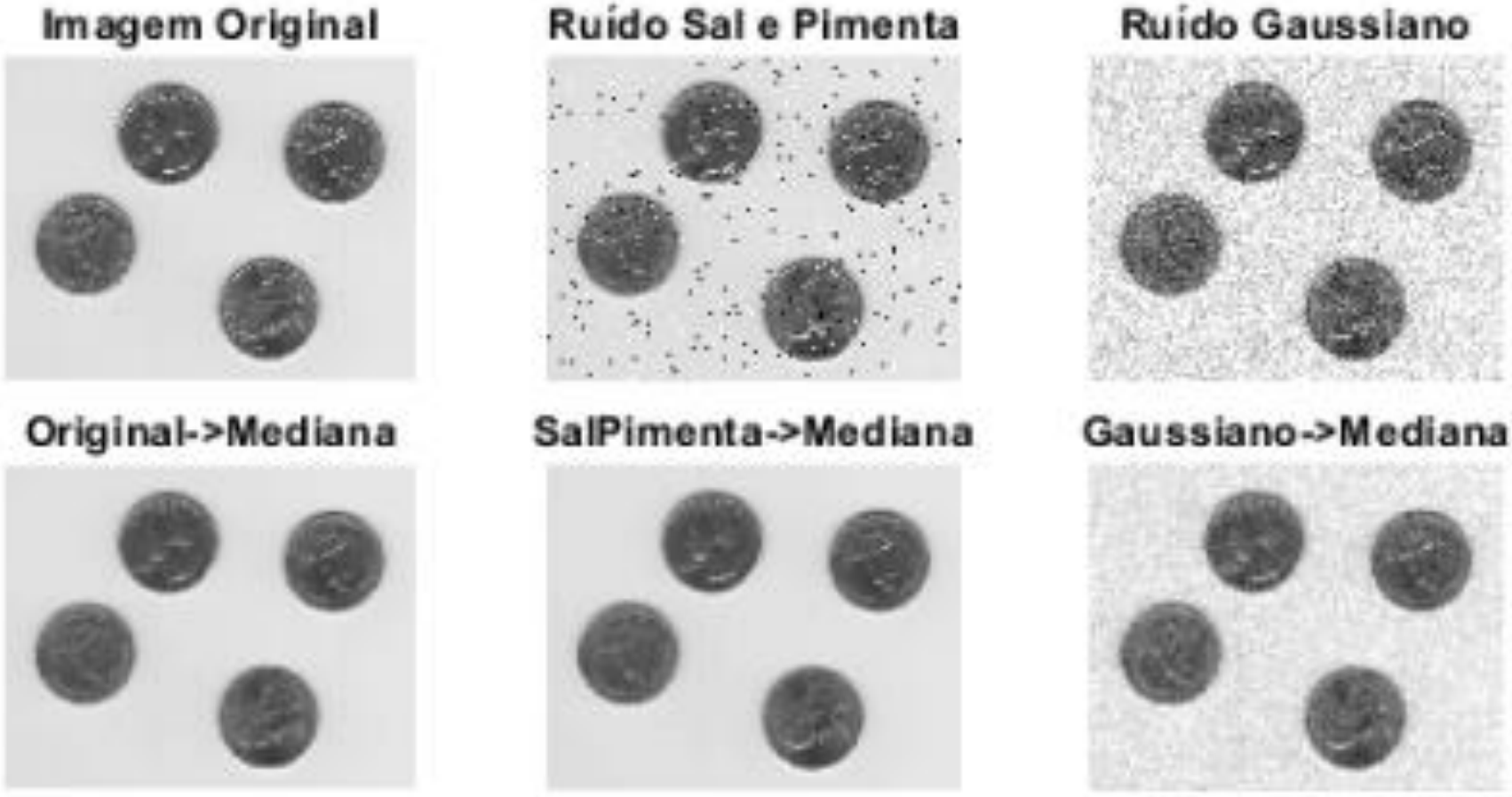
12 238 244 244 **245** 245 247 250 252

O quinto valor será a mediana ou seja **245**

A função **medfilt2** do Matlab configura e aplica o Filtro da Mediana em uma imagem.

Filtragem pela Mediana

- Filtro de Mediana 3x3 aplicado às imagens originais:



Filtragem por Ordem

- Similar ao filtro da mediana, o filtro de ordem é um filtro não linear que consiste nos seguintes passos:

  1. Definição da vizinhança do pixel-alvo (N x N).
  2. Ordenação dos valores de pixel em ordem crescente (o primeiro é o de menor valor e o (N x N)-ésimo, o de maior valor).
  3. Escolha da ordem do filtro (de 1 a N).
  4. Especificação do valor do filtro igual ao valor de pixel da ordem escolhida.

- Filtros de ordem que selecionam o valor máximo ou mínimo na vizinhança especificada são denominados, respectivamente, filtros de máximo e de mínimo.

Filtragem por Ordem

A função **ordfilt2** do Matlab configura e aplica o filtro de ordem em uma imagem.

```
I = imread('eight.tif');
subplot(2,3,1), imshow(I); title('Imagem Original'); % Exibe a imagem
I_m = ordfilt2(I, 25, ones(5,5)); % Aplica o filtro de máximo na imagem original
subplot(2,3,4), imshow(I_m); title('Original->Ordem'); % Exibe a imagem

Isp = imread('eightSaltPepper.tif'); % Lê a imagem com ruído Sal e Pimenta
subplot(2,3,2), imshow(Isp); title('Ruído Sal e Pimenta');  % Exibe a imagem ruído Sal e Pimenta
Isp_m = ordfilt2(Isp, 25, ones(5,5)); % Aplica o filtro de máximo na imagem com ruído Sal e Pimenta
subplot(2,3,5), imshow(Isp_m); title('SalPimenta->Ordem'); % Exibe a imagem

Ig = imread('eightGaussian.tif'); % Lê a imagem com ruído Gaussiano
subplot(2,3,3), imshow(Ig);  title('Ruído Gaussiano'); % Exibe a imagem com ruído Gaussiano
Ig_m = ordfilt2(Ig, 25, ones(5,5)); % aplica o filtro de máximo na imagem com ruído Gaussiano
subplot(2,3,6), imshow(Ig_m); title('Gaussiano->Ordem'); % Exibe a imagem
```

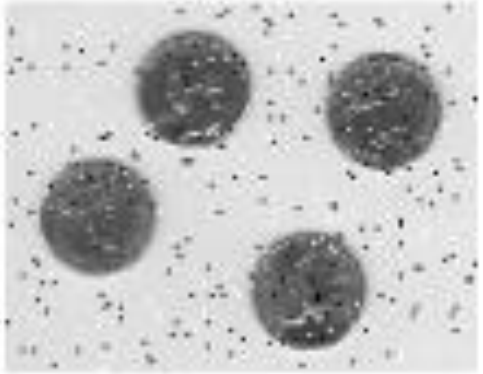
Median filter	`B = ordfilt2(A,5,ones(3,3))`	1 1 1 1 1 1 1 1 1	88 16 56 5 3 **30** 21 63 42
Minimum filter	`B = ordfilt2(A,1,ones(3,3))`	1 1 1 1 1 1 1 1 1	88 16 56 5 **3** 30 21 63 42

Maximum filter	`B = ordfilt2(A,9,ones(3,3))`	1 1 1 1 1 1 1 1 1	**88** 16 56 5 3 30 21 63 42
Minimum of north, east, south, and west neighbors	`B = ordfilt2(A,1,[0 1 0; 1 0 1; 0 1 0])`	0 1 0 1 0 1 0 1 0	88 16 56 **5** 3 30 21 63 42

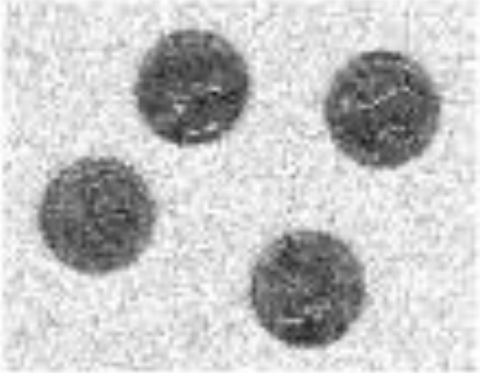
Filtragem por Ordem

Imagem Original

Ruído Sal e Pimenta

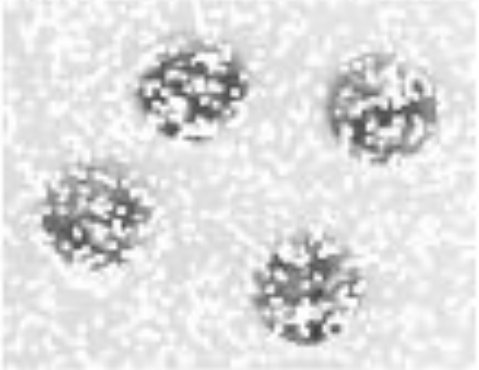


Ruído Gaussiano



Original->Ordem

SalPimenta->Ordem



Gaussiano->Ordem

Filtragem de Suavização Conservativa

- Esse tipo de suavização é conservadora em termos de não modificar os pixels cegamente.
- Como tal, existem duas condições a serem testadas antes da substituição ser efetuada.
- Primeiro, uma vizinhança é selecionada ao redor do pixel-alvo. Normalmente, uma região de oito vizinhos é selecionada (ou seja, todos os oito pixels ao redor do pixel subjacente).

Filtragem de Suavização Conservativa

- Os valores mínimo e máximo desta vizinhança são encontrados e o valor do pixel central é comparado contra esses valores limitantes.

- Se o valor do pixel central estiver acima do máximo, é definido como o valor máximo;

- Se for menor que o mínimo, é definido como o valor mínimo; caso contrário, é deixado como está.

Filtragem de Suavização Conservativa

- Devido a essa estratégia de seletividade de substituição, as chances de remover os valores extremos e, consequentemente, o ruído de salpicos e sal e pimenta, pode ser eliminado.

- Os efeitos podem ser aperfeiçoados escolhendo uma vizinhança maior.

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

Valor central: 150

Vizinhança:

115, 119, 120, 123, 124, 125, 126, 127

Max.: 127

Min.: 115

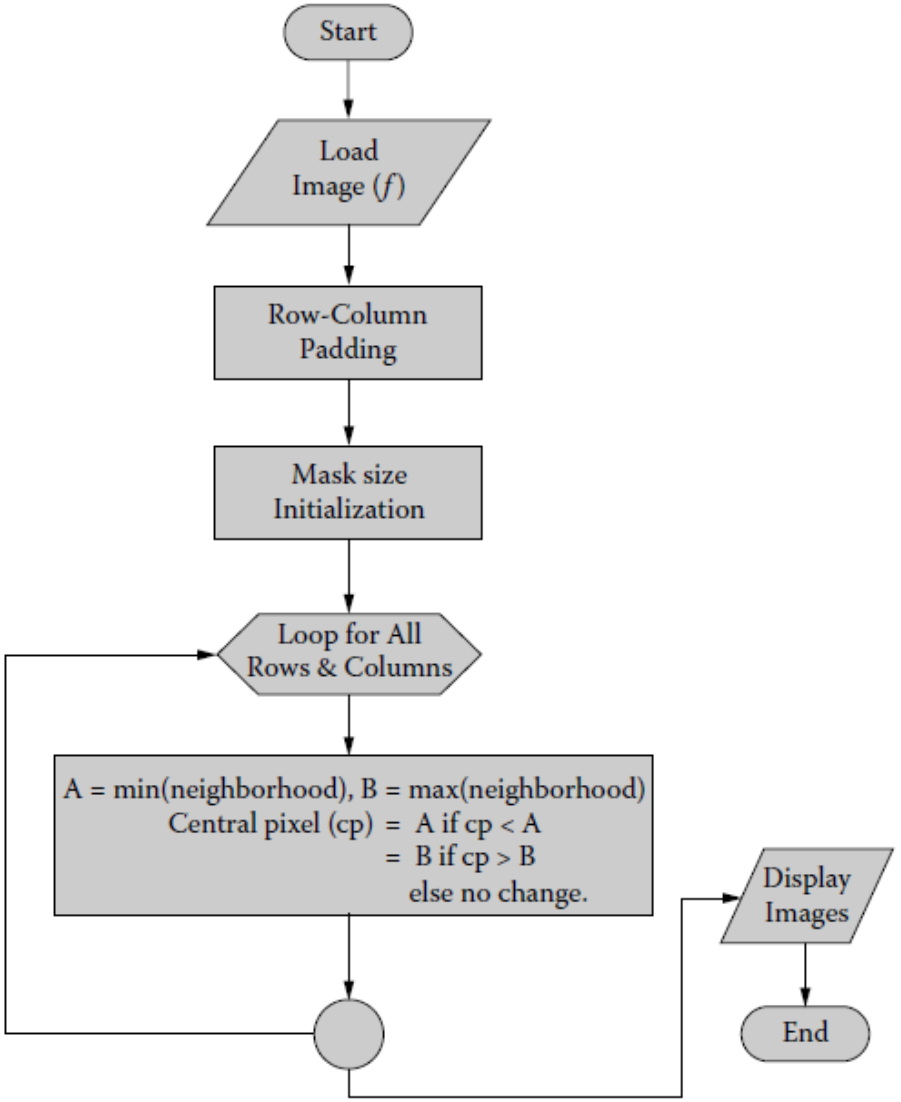
Resultado: **150 é substituído por 127**

Antes

Depois

Filtragem de Suavização Conservativa (algoritmo)

```
function yl = conserv(xng)
[r,c] = size(xng);
xl = zeros(r+2,c+2);
xl(2:r+1,2:c+1,:) = xng(:,:);
[r,c] = size(xl);
yl = xl;

for i = 2 : r-1
    for j = 2 : c-1
        nh = [xl(i-1,j-1) xl(i-1,j) xl(i-1,j+1)
              xl(i,j-1)   xl(i,j)   xl(i,j+1)
              xl(i+1,j-1) xl(i+1,j) xl(i+1,j+1)];

        cp = xl(i,j);
        mx = max(nh);
        mn = min(nh);

        if (cp > mx)
            cp = mx;
        else
            if (cp < mn)
                cp = mn;
            end
        end

        yl(i,j) = cp;
    end
end
return
```



Filtragem Gaussiana

- A aplicação de um filtro gaussiano tem o efeito de suavizar a imagem, removendo detalhes e ruídos.

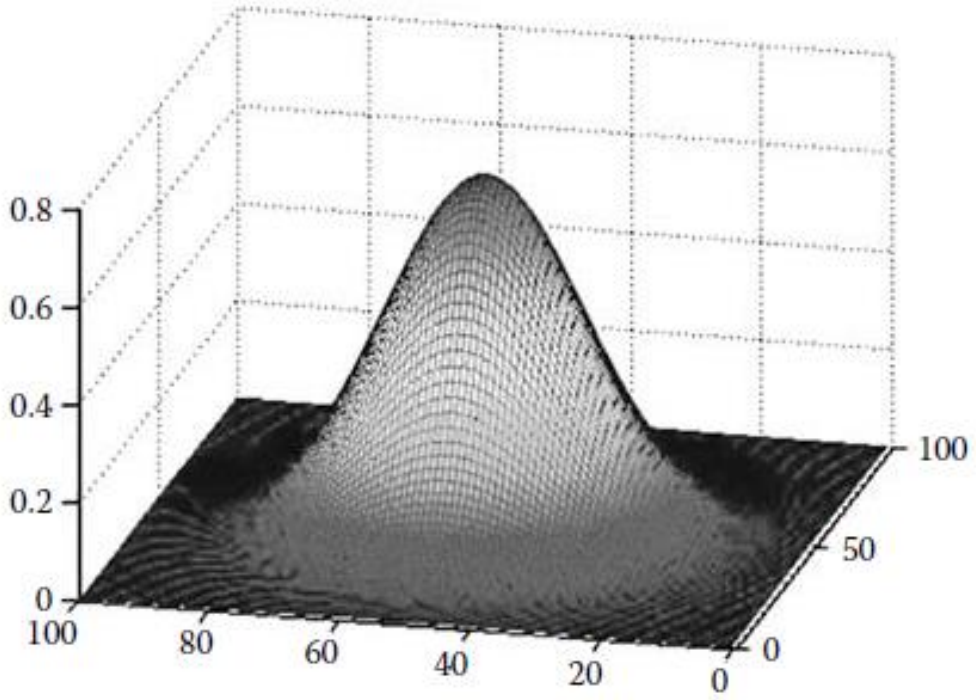
- O grau de suavização é controlado pela escolha do parâmetro de desvio-padrão **σ** e não pelo valor absoluto das dimensões do núcleo (como no caso do filtro de média).

- A imagem é filtrada usando um núcleo discreto derivado de uma forma radialmente simétrica da função gaussiana contínua bidimensional, definida como:

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

Filtragem Gaussiana

- A ideia da suavização gaussiana é usar a distribuição como uma função de espalhamento de pontos, realizando a convolução da máscara com a imagem processada.



A distribuição gaussiana 2D com média (50%, 50%) e desvio padrão 1

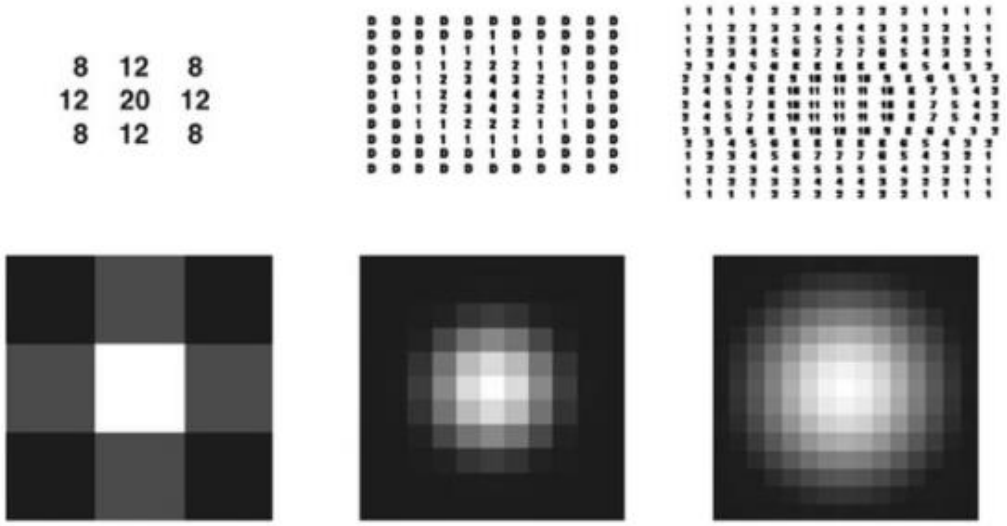
$$\frac{1}{273}$$

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

A versão matricial da distribuição gaussiana ao lado.

Filtragem Gaussiana

- Aproximações discretas a essa função contínua são especificadas com o uso de dois parâmetros livres:
  1. As desejadas dimensões do núcleo (como uma máscara N x N);
  2. O valor de **σ**, o desvio-padrão da função gaussiana.



Núcleos de filtros gaussianos $3 \times 3\ \sigma = 1$, $11 \times 11\ \sigma = 2$ e $21 \times 21\ \sigma = 4$ (os valores numéricos são mostrados normalizados).

Criar Filtro Gaussiano

1°) Determinar o tamanho do filtro. Isso determinará a largura e a altura do *kernel* usado para convolução. O tamanho deve ser um número ímpar, como 3, 5, 7, etc., para ter um centro bem definido.

Criar Filtro Gaussiano

2º) Calcular os valores do *kernel* gaussiano. Os valores do *kernel* são baseados na distribuição gaussiana. A fórmula para a distribuição Gaussiana 2D é dada por:

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

Onde **x** e **y** são as coordenadas no *kernel* e (**σ**) ***sigma*** é o desvio padrão da distribuição gaussiana.

Valores maiores de **(σ)** ***sigma*** resultam em uma distribuição mais ampla e suave.

Criar Filtro Gaussiano

Valores para **x** e **y** são as coordenadas no *kernel*

	Y				
X	(-2,-2)	(-2,-1)	(-2,0)	(-2,1)	(-2,2)
	(-1,-2)	(-1,-1)	(-1,0)	(-1,1)	(-1,2)
	(0,-2)	(0,-1)	(0,0)	(0,1)	(0,2)
	(1,-2)	(1,-1)	(1,0)	(1,1)	(1,2)
	(2,-2)	(2,-1)	(2,0)	(2,1)	(2,2)

Criar Filtro Gaussiano

3°) Normalizar o *kernel*. Para garantir que a soma de todos os valores do *kernel* seja igual a **1.0**, é preciso dividir cada valor do *kernel* pela soma de todos os valores dentro do *kernel*.

4°) Aplicar o filtro a uma imagem usando convolução. Iterar sobre cada pixel da imagem e executar a convolução com o *kernel* gaussiano.

A convolução envolve tomar a soma ponderada do pixel e seus vizinhos de acordo com os valores do *kernel*.

Filtragem Gaussiana

- Uma função gaussiana com grande valor de **σ** também é um exemplo do chamado filtro passa-baixas, que suprime o conteúdo de alta frequência de uma imagem (por exemplo, características agudas de bordas).

- *A suavização (ou filtragem) gaussiana, em geral, representa o primeiro estágio de algoritmos de detecção de bordas (por exemplo, detector de bordas de Canny) em que é usada como meio de supressão de ruído.*

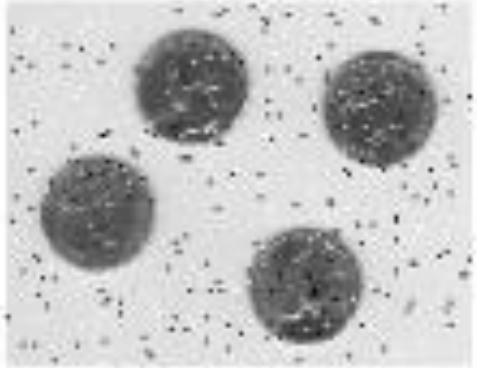
Filtragem Gaussiana

```
% Define o filtro Gaussiano 5x5 com desvio padrão 2
k = fspecial('gaussian', [5 5], 2);

I = imread('eight.tif');
subplot(2,3,1), imshow(I); title('Imagem Original'); % Exibe a imagem
I_g = imfilter(I, k); % Aplica o filtro gaussiano na imagem original
subplot(2,3,4), imshow(I_g); title('Original->Gaussiano'); % Exibe a imagem

Isp = imread('eightSaltPepper.tif'); % Lê a imagem com ruído Sal e Pimenta
subplot(2,3,2), imshow(Isp); title('Ruído Sal e Pimenta');  % Exibe a imagem ruído Sal e Pimenta
Isp_g = imfilter(Isp, k); % Aplica o filtro gaussiano na imagem com ruído Sal e Pimenta
subplot(2,3,5), imshow(Isp_g); title('SalPimenta->Gaussiano'); % Exibe a imagem

Ig = imread('eightGaussian.tif'); % Lê a imagem com ruído Gaussiano
subplot(2,3,3), imshow(Ig);  title('Ruído Gaussiano'); % Exibe a imagem com ruído Gaussiano
Ig_g = imfilter(Ig, k); % Aplica o filtro gaussiano na imagem com ruído Gaussiano
subplot(2,3,6), imshow(Ig_g); title('Gaussiano->Gaussiano'); % Exibe a imagem
```

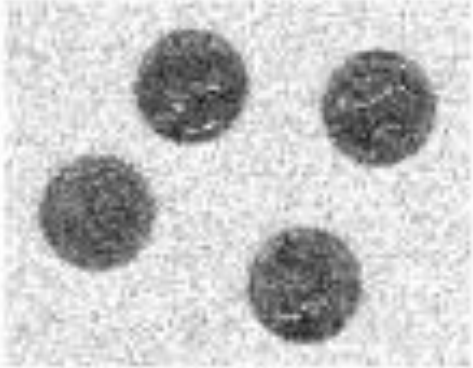
Filtragem Gaussiana

Imagem Original

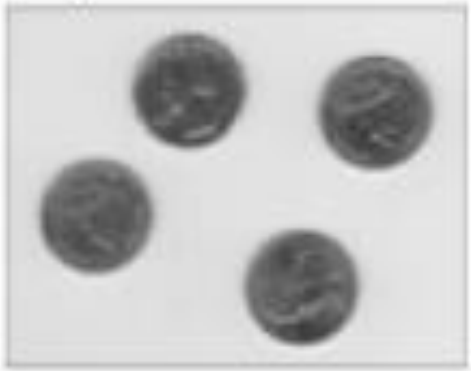
Ruído Sal e Pimenta



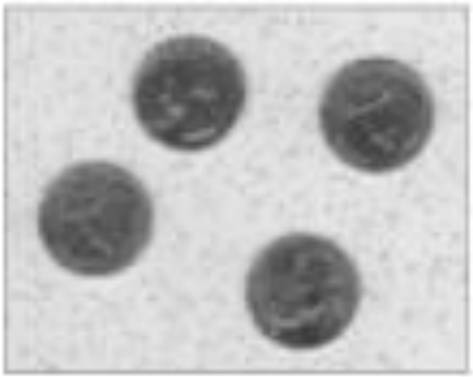
Ruído Gaussiano



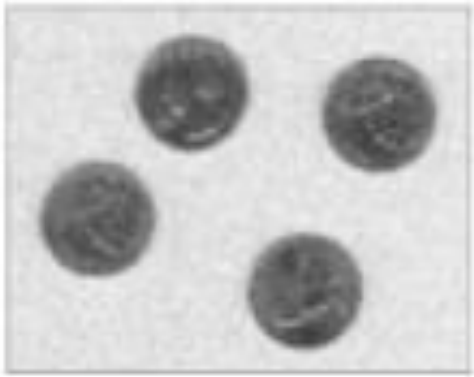
Original->Gaussiano



SalPimenta->Gaussiano



Gaussiano->Gaussiano



Remoção de Ruído Gaussiano

```
h1 = ones(3)/9; % Filtro Média
h2 = fspecial('gaussian',5,5); % Filtro Gaussiano

x = imread('Lenna.jpg'); % Lê imagem
x = rgb2gray(x); % Converte imagem RGB para Gray
figure; imshow(x); title('Original'); % Exibe imagem original

xng = imnoise(x,'gaussian',0,0.1); % Aplica ruído gaussiano na imagem
figure; imshow(xng); title('Gaussian Noise'); % Exibe imagem com ruído gaussiano

% Aplica filtragem por média
g1 = imfilter(xng,h1);
figure; imshow(g1); title('Media Filter');

% Aplica filtragem gaussiana
g2 = imfilter(xng,h2);
figure; imshow(g2); title('Gaussian Filter');

% Aplica filtragem por mediana
g3 = medfilt2(xng);
figure; imshow(g3); title('Median Filter');

% Aplica filtragem consevativa
g4 = conserv(xng);
figure; imshow(g4*0.003); title('Conservative Filter');
```

Remoção de Ruído Gaussiano

Original

Gaussian Noise

Media Filter

Gaussian Filter

Median Filter

Conservative Filter

Remoção de Ruído Sal e Pimenta

```
h1 = ones(3)/9; % Filtro Média
h2 = fspecial('gaussian',5,5); % Filtro Gaussiano

x = imread('Lenna.jpg'); % Lê imagem
x = rgb2gray(x); % Converte imagem RGB para Gray
figure; imshow(x); title('Original'); % Exibe imagem original

% Salt & Pepper Noise
xns = imnoise(x, 'salt & pepper',0.1); % Aplica ruído sal e pimenta na imagem
figure; imshow(xns); title('Salt & Pepper Noise'); % Exibe imagem com ruído sal e pimenta

g5 = imfilter(xns,h1); % Aplica filtragem por média
figure; imshow(g5); title('Media Filter');

g6 = imfilter(xns,h2); % Aplica filtragem gaussiana
figure; imshow(g6); title('Gaussian Filter');

g7 = medfilt2(xns); % Aplica filtragem por mediana
figure; imshow(g7); title('Median Filter');

g8 = conserv(xns); % Aplica filtragem consevativa
figure; imshow(g8*0.003); title('Conservative Filter');
```

Remoção de Ruído Sal e Pimenta

Original

Salt & Pepper Noise

Media Filter

Gaussian Filter

Median Filter

Conservative Filter

Lista de Exercícios

Exercício 1

Explore o uso da função ***imnoise()*** de Matlab para a adição de diferentes níveis de ruídos ‘sal e pimenta’ e gaussiano às imagens.

Use as imagens '*onion.png*' e '*eight.tif*' como exemplos de imagens em cores e em escala de cinza para construir uma série de variáveis de imagens em Matlab com diferentes níveis e tipos de ruído (investigue, também, outros tipos de ruído disponíveis com essa função).

Com base nos tópicos de filtragem apresentados, investigue a utilidade das filtragens pela média, pela mediana e gaussiana na remoção de diferentes tipos e níveis de ruído em imagens.

Exercício 2

A partir da definição de vizinhança local de pixel, implemente uma função em Matlab para extrair a vizinhança N x N de uma dada posição de pixel (x, y) e copie os valores de pixel em uma imagem nova e menor N x N.

Explore a sintaxe da estrutura iterativa **FOR** de Matlab.

Combine seu programa de extração com a função ***imresize()*** de Matlab para criar um programa para extração e amplificação de regiões.

Exercício 3

A função ***colfilt()*** de Matlab utiliza as altamente eficientes operações matriciais de Matlab para efetuar operações de filtragem em vizinhanças de pixels de imagens.

Usando as funções ***tic(), toc()*** de Matlab para medida de tempo, meça o tempo de computação da operação efetuada com e sem o uso desse parâmetro de otimização.

Varie as dimensões da vizinhança em que a operação é efetuada (assim como a operação ***min()***, ***max()***, ***mean()***).

O que você observa?

Há consistência entre a melhora de desempenho e vizinhanças muito pequenas ou muito grandes?

Exercício 4

Explore o uso de diferentes dimensões de vizinhanças e o resultante efeito sobre a imagem de saída.

Use as funções de Matlab para desenho de gráficos (função ***plot()***) e as funções ***tic()*** e ***toc()*** de Matlab para medida de tempo para criar um gráfico da dimensão N da vizinhança versus tempo de computação na aplicação das funções ***min()***, ***max()*** a uma imagem.

O tempo de computação aumenta linearmente com N ou não? Por quê?

Exercício 5

Com base na filtragem pela média em uma dada imagem, registre o tempo de execução desta operação (usando as funções ***tic()***/***toc()*** de Matlab para medida de tempo) para diferentes dimensões da vizinhança na faixa 0−25.

Use as facilidades de Matlab para o desenho de gráficos e apresente seus resultados na forma de um gráfico. O que você nota?

Como o tempo de execução varia com o aumento nas dimensões da vizinhança? (Sugestão: Para automatizar esta tarefa, considere o emprego da estrutura iterativa for de Matlab.)

Como uma extensão, repita o exercício para diferentes tamanhos de imagens (ou uma faixa de tamanhos de imagem obtida com ***imresize()***) e faça um gráfico dos resultados. Que tendência observa?

Exercício 6

Repita a primeira parte do Exercício 5 e compare as diferenças entre filtragem pela média e filtragem pela mediana.

Como as tendências se comparam?

Como explica quaisquer diferenças observadas?

Exercício 7

Uma região de interesse (RDI)* em uma imagem é uma sub-região da imagem (em geral, de natureza retangular) na qual operações localizadas de processamento de imagem podem ser efetuadas.

Em Matlab, uma RDI pode ser selecionada de modo iterativo exibindo, primeiro, uma imagem (via ***imshow()***) e, depois, usando a função ***roipoly()*** para retornar uma sub-região da imagem definida como uma imagem binária do mesmo tamanho que a original (zero fora da RDI e um no interior da RDI).

Explore o uso da função ***roifilt()*** para aplicação seletiva de filtragem gaussiana e filtragem pela média a uma dada região de interesse em uma das imagens de exemplo disponíveis em Matlab.

Explore, também, a combinação da função de seleção de RDI com sua resposta ao Exercício 2 para extrair uma dada RDI para equalização de histograma ou processamento de detecção de bordas separadamente do resto da imagem.

Exercício 8

Implemente uma função em Matlab para efetuar a operação de **filtragem por média**. Não utilize a função ***imfilter()***, mas sim, implemente a teoria por trás da filtragem por média manualmente. (Sugestão: Explore a estrutura iterativa FOR de Matlab.)

Teste essa operação de filtragem com diferentes tipos de ruídos.

Compare os resultados da sua função de filtragem por média contra a função do Matlab. Os resultados foram similares?

Exercício 9

Implemente uma função em Matlab para efetuar a operação de **filtragem por mediana**. Não utilize a função ***medfilt2()***, mas sim, implemente a teoria por trás da filtragem por mediana manualmente. (Sugestão: Explore a estrutura iterativa FOR de Matlab.)

Teste essa operação de filtragem com diferentes tipos de ruídos.

Compare os resultados da sua função de filtragem por mediana contra a função do Matlab. Os resultados foram similares?