PRE-PROCESSAMENTO

SEGMENTAÇÃO

Marta Bez

Pré-processamento

- *Pode ser usado para retirar informações* irrelevantes da imagem.

- Tem como objetivo principal preparar uma imagem para algum processamento, alterando seu brilho (*gray-scale,* filtragem, *detectores de bordas, etc…).*

Transformação Grayscale

- *Modifica o brilho de cada pixel, dependendo de suas propriedades*.
- Leva em conta seu brilho, mas não sua posição na imagem.
- Usada para tornar mais fácil a interpretação devido ao realce e contraste.

Transformação Grayscale

Útil para etapas posteriores de processamento, pois o trabalho de análise e processamento da imagem será feito em relação a um só canal de cor uniforme e não a três, que seria o caso se trabalhar com imagens RGB.


```
namespace DoisDWeb.Projetos.DigitalImage.Metodos
{
   public class TonsCinza : MetodoFiltroBase
   {
      public override ImageBytes Executar(ImageBytes image)
      {
         int width = image.Width;
         int height = image.Height;

         for (int x = 0; x < width; x++)
         {
            for (int y = 0; y < height; y++)
            {
               Color color = image.GetPixel(x, y);
               byte media = Convert.ToByte((color.R + color.G + color.B) / 3);
               color.R = media;
               color.G = media;
               color.B = media;

               image.SetPixel(x, y, color);
            }
         }
         return image;
      }
   }
}
```

Código Fonte em C#

```
// ============================================================
// GrayscaleBT709.cs
// .NET Image Tools
// ============================================================
// Copyright (c) .NET Image Tools Development Group.
// All rights reserved.
// ============================================================

namespace DoisDWeb.Projetos.DigitalImage.Metodos
{
    public sealed class GrayscaleBT709 : Grayscale
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="GrayscaleBT709"/> class.
        /// </summary>
        public GrayscaleBT709() : base(0.2125, 0.7154, 0.0721) { }
    }
}
```

Código Fonte em C#

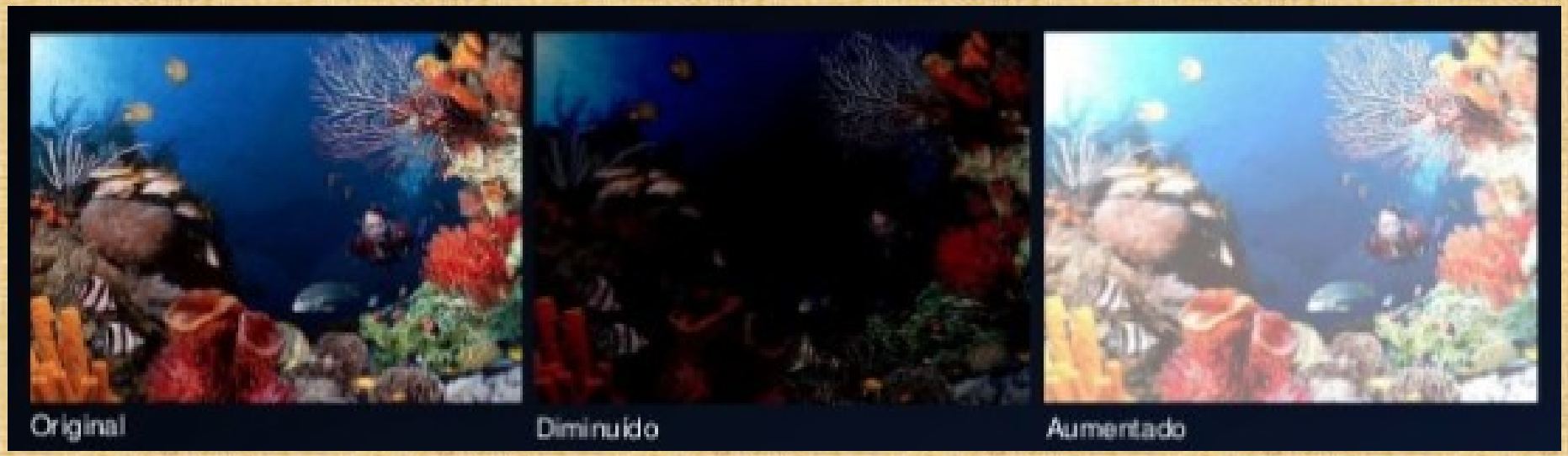
```
// ============================================
// GrayscaleRMY.cs
// .NET Image Tools
// ============================================
// Copyright (c) .NET Image Tools Development Group.
// All rights reserved.
// ============================================
using System.Diagnostics.CodeAnalysis;


namespace DoisDWeb.Projetos.DigitalImage.Metodos
{
   public sealed class GrayscaleRMY : Grayscale
   {
      public GrayscaleRMY()
         : base(0.5, 0.419, 0.081)
      {
      }
   }
}
```

Brilho

1. A iluminação não uniforme em uma imagem causa vários problemas na identificação objetiva de objetos desta imagem.
2. Grandes variações de iluminação podem causar baixo contraste em regiões de uma imagem.
3. Baixo contraste tende a ocultar ou tornar não inteligíveis detalhes na imagem.
4. Quando o contraste é muito baixo, a pessoa que vê a imagem pode nem sempre perceber alguns objetos ou detalhes de objetos na imagem.

Brilho



Brilho

O ajuste de brilho é uma transformação linear regida pela fórmula

$$g(x,y) = a * f(x,y) + b$$

Onde

$g(x,y)$ vo valor do pixel

$f(x,y)$ antigo valor do pixel

O ajuste de brilho é feito alterando ***b*** para um valor maior ou menor que 0.

Se o valor for mantido em 0, a imagem não é alterada.

Assim, adiciona-se intensidade ao canal do pixel em questão.

Brilho


UNIVERSIDADE
FEEVALE


UNIVERSIDADE
FEEVALE


UNIVERSIDADE
FEEVALE


UNIVERSIDADE
FEEVALE


UNIVERSIDADE
FEEVALE

Brilho

Código Fonte em C#

```
using System;
using System.Collections.Generic;
using System.Text;

namespace DoisDWeb.Projetos.DigitalImage.Metodos
{
   public partial class BrilhoContrasteMenosBrilho : BrilhoContraste
   {
      public BrilhoContrasteMenosBrilho()
         : base()
      {
         this.Contraste = 1;
         this.Brilho = -10;
      }
   }
}
```

```
private byte aplicaEfeito(byte b)
{
   double n  = Contraste * b + Brilho;
   if (n > 255)
   {
      n = 255;
   }
   else if (n < 0)
   {
      n = 0;
   }
   return Convert.ToByte(n);
}
```

Contraste

O ajuste de contraste é uma transformação regida pela mesma fórmula do brilho.

O contraste se volta para a alteração do ganho, no caso, o parâmetro **a**.

Se o ganho for mantido em 1, a imagem permanece inalterada.

Com valores entre 0 e 1, a imagem tem seu contraste reduzido e, para valores maiores, o contraste é aumentado.

$$g(x,y) = a * f(x,y) + b$$

Contraste

Contraste


UNIVERSIDADE
FEEVALE


UNIVERSIDADE
FEEVALE

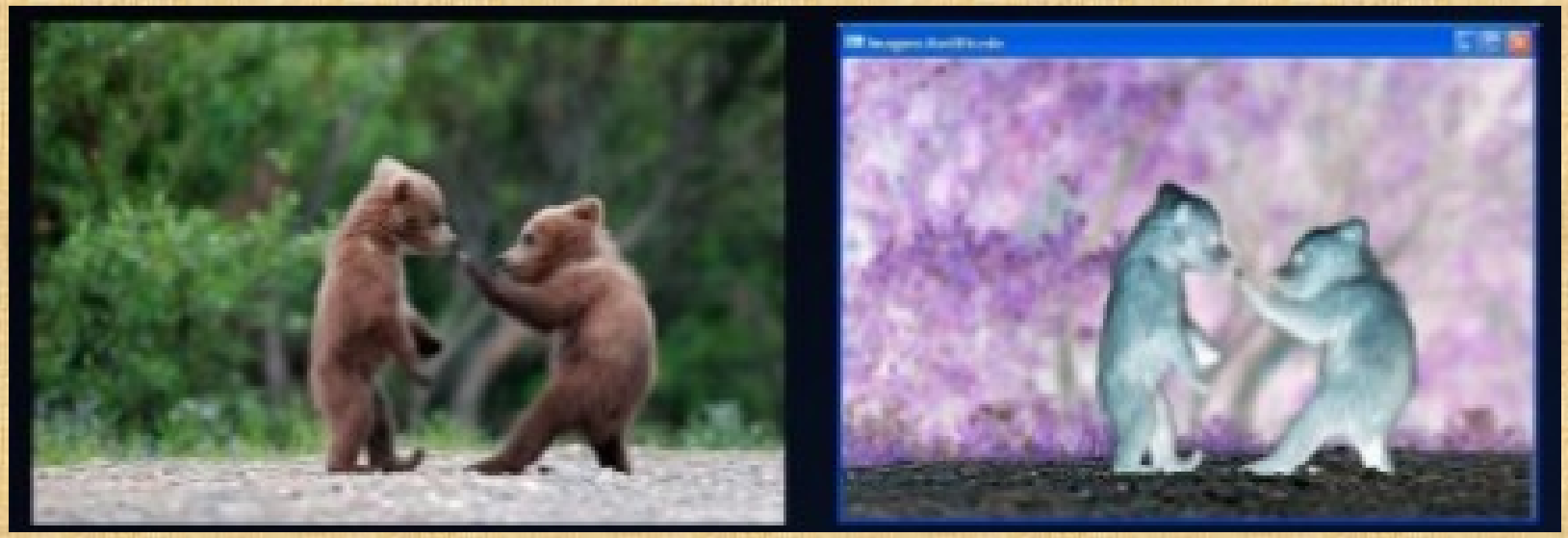

UNIVERSIDADE
FEEVALE


UNIVERSIDADE
FEEVALE


UNIVERSIDADE
FEEVALE

Negativo da Imagem

Obtém-se o negativo da imagem subtraindo o valor máximo permitido (255) do valor do pixel.



Novo_pixel = 255 – pixel_original

Filtragem

As técnicas de filtragem são transformações da imagem "pixel" a "pixel", que dependem do nível de cinza de um determinado "pixel" e do valor dos níveis de cinza dos "pixels" vizinhos, na imagem original, ou seja, o pixel "filtrado” tem um valor dependente do contexto em que ele se encontra na imagem original.

Filtragem

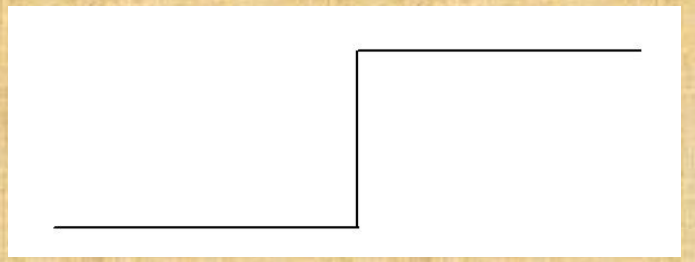
O objetivo principal da acentuação de contornos é enfatizar detalhes dos limites de objetos de modo a permitir sua identificação e análise posterior.

- **Contorno**: identificado por mudanças locais de intensidade significativas na imagem, ocorrendo tipicamente na separação de duas regiões diferentes.

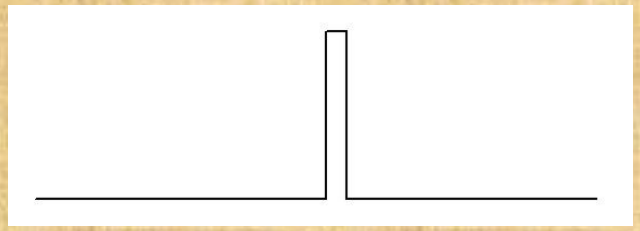
- **Regiões ou objetos**: identificados por patamares mais ou menos constantes de tons e cores.

- **Fronteira**: ocorre onde a função de intensidade da imagem, *f(xi,yi),* varia bruscamente, consistindo em limites de regiões cujos valores de cor apresentam grandes diferenças.

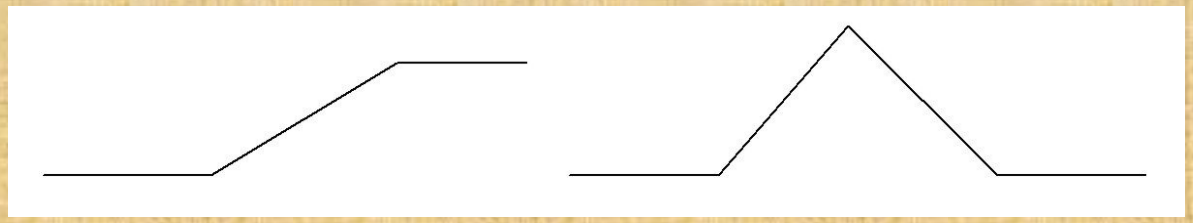
Descontinuidades



Descontinuidade do tipo degrau.

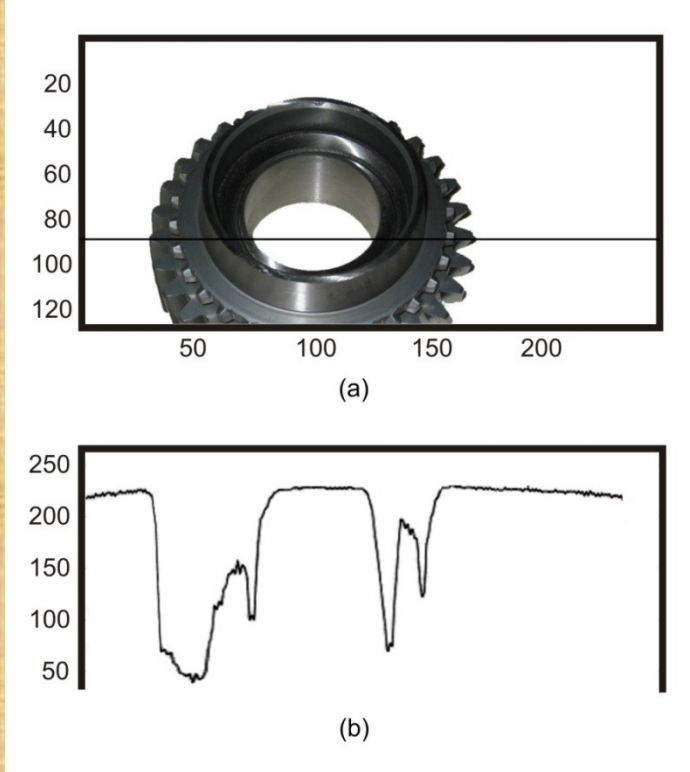


Descontinuidade do tipo pico.



Modelos de descontinuidade em rampa.

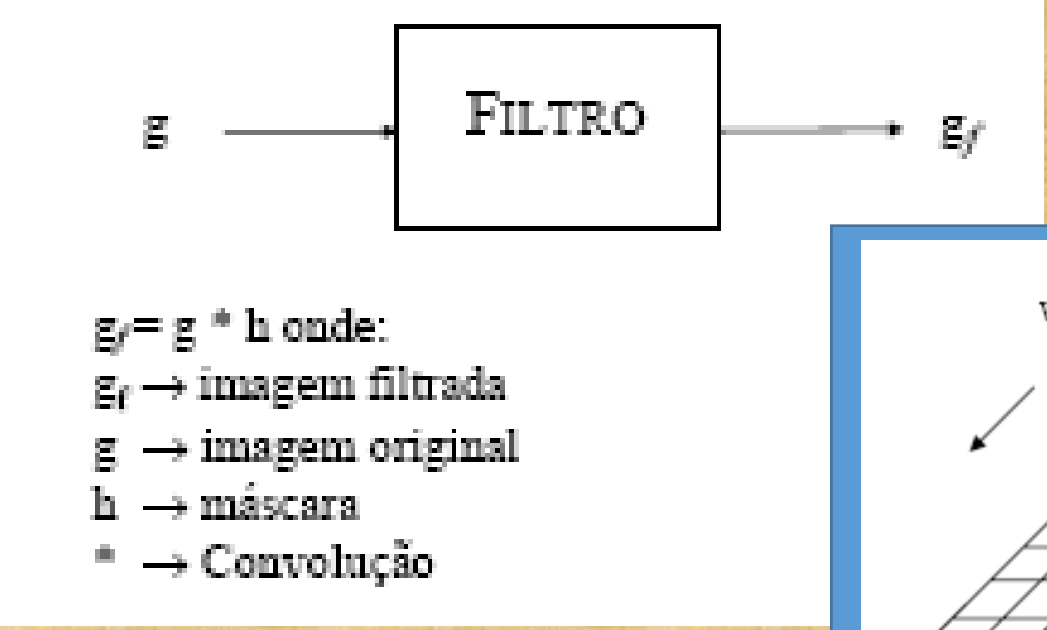
Descontinuidades



Exemplo de variação de intensidade ao longo da linha 80 da imagem Engrenagem.

Filtragem por convolução

A Convolução discreta é descrita pelo seguinte esquema:



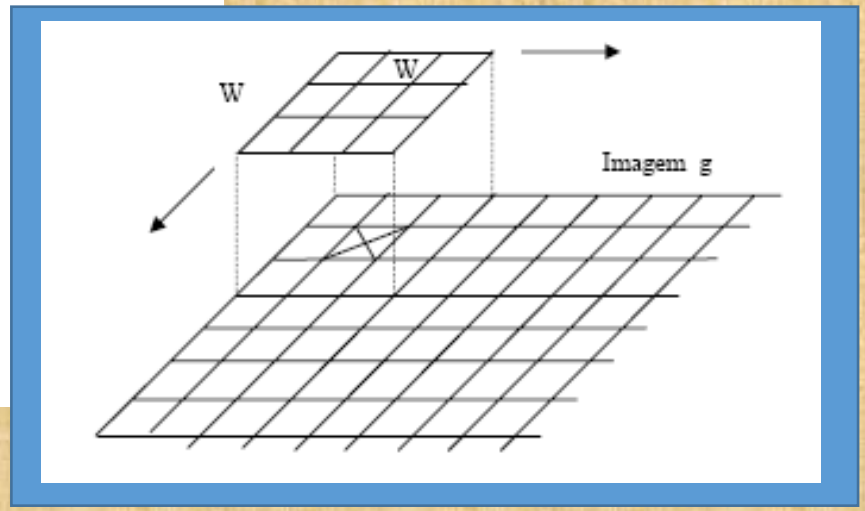
$g_f = g * h$ onde:

$g_f \rightarrow$ imagem filtrada

$g \rightarrow$ imagem original

$h \rightarrow$ máscara

$* \rightarrow$ Convolução



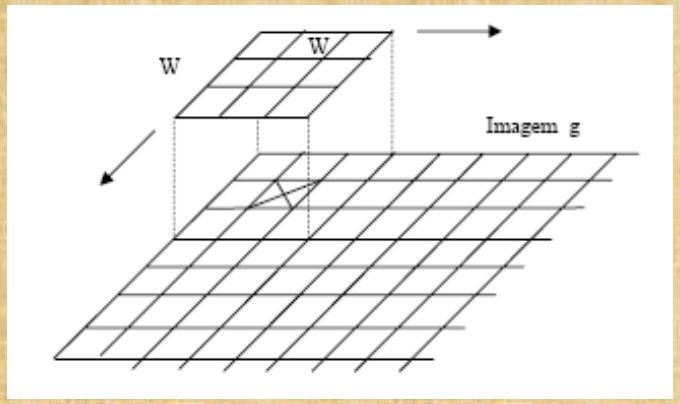
Convolution kernel

Filtragem por convolução

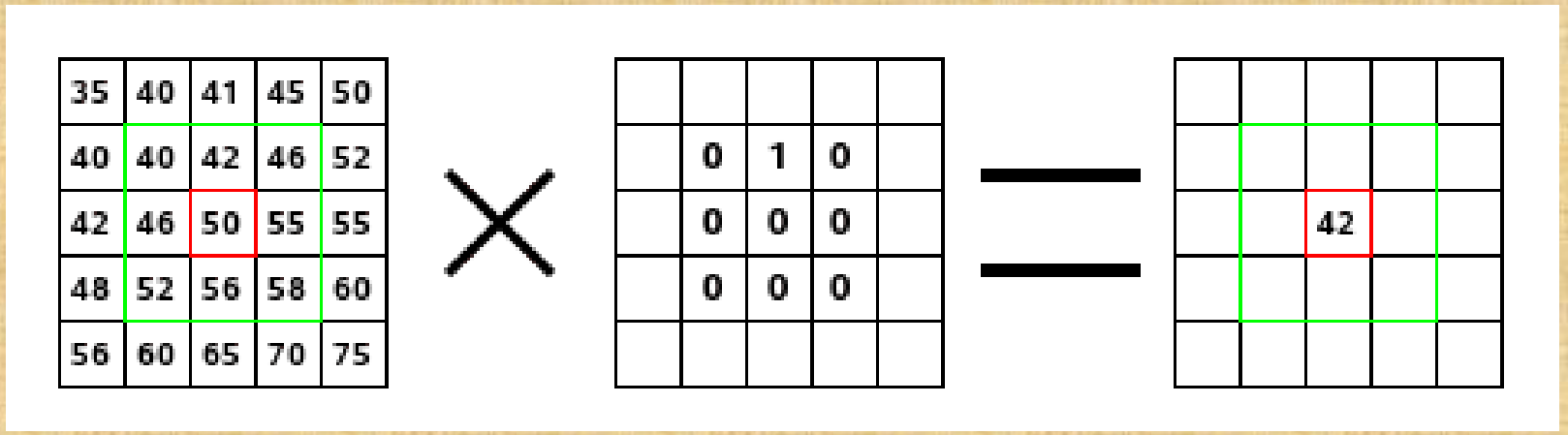
- Com esta operação o pixel central (marcado com x) terá um novo valor que depende dele e dos vizinhos.

- Os filtros espaciais são implementados através de máscaras (matrizes) com dimensões ímpares.

- Os tipos de filtros são Passa Baixas, Passa Altas, Direcionais, Passa Banda.



Filtragem por convolução



O pixel inicial que era 50 passou a ser 42

(40*0)+(42*1)+(46*0) +
(46*0)+(50*0)+(55*0) +
(52*0)+(56*0)+(58*0) = 42.

https://elo7.dev/convolucao

(o filtro não trabalha diretamente sobre

Filtragem por convolução

6	9	2	7	8	6	3	2	8	7
1	5	2	7	3	6	7	1	4	7
8	2	2	3	8	8	4	0	4	4
5	1	2	0	5	2	9	9	1	5
4	3	9	1	7	8	8	8	1	2
7	5	7	7	5	3	7	8	4	1
6	9	1	1	8	3	0	2	1	8
6	10	7	5	2	10	10	1	1	10
3	8	1	3	1	10	3	3	3	4
3	1	5	7	6	8	9	4	8	9

*

1	2	3
4	5	6
7	8	9

=

137	146	202	188	231	212	158	163	192	151
156	161	163	211	277	269	171	152	173	133
118	132	89	155	189	264	242	205	190	101
112	188	131	204	215	305	315	232	173	64
152	250	222	250	219	274	316	257	172	61
211	255	201	212	208	204	182	144	168	97
251	298	258	200	229	263	224	138	185	137
225	237	210	132	235	246	226	130	178	124
138	179	204	193	285	311	270	227	257	181
51	69	90	101	141	148	126	122	131	88

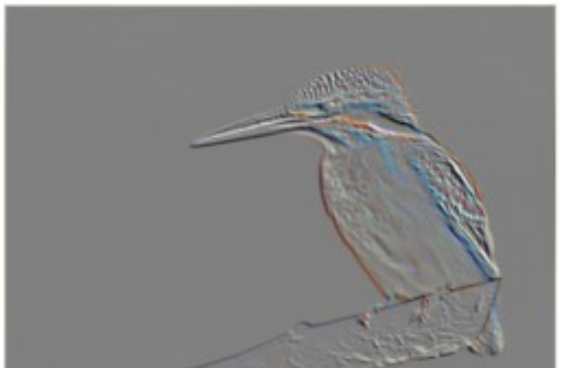
161 = 1*6 + 2*9 + 3*2 + 4*1 + 5*5 + 6*2 + 7*8 + 8*2 + 9*2

168 = 1*8 + 2*1 + 3*2 + 4*8 + 5*4 + 6*1 + 7*2 + 8*1 + 9*8

123 = 1*1 + 2*1 + 3*8 + 4*7 + 5*5 + 6*2 + 7*1 + 8*3 + 9*1

https://elo7.dev/convolucao

Filtragem por convolução

Tarefa	Kernel	Imagem Original	Resultado
Borrar uma imagem	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$		
Detectar bordas de objeto	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$		
Detectar objetos	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		

https://elo7.dev/convolucao/

Filtragem por convolução

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

Aguçar

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Desfocar

https://docs.gimp.org/2.8/pt_BR/plug-in-convmatrix.html

Filtragem por convolução

	0	0	0	
	-1	1	0	
	0	0	0	

Realçar Bordas

	0	1	0	
	1	-4	1	
	0	1	0	

Detectar Bordas

https://docs.gimp.org/2.8/pt_BR/plug-in-

Problemas com as bordas das imagens

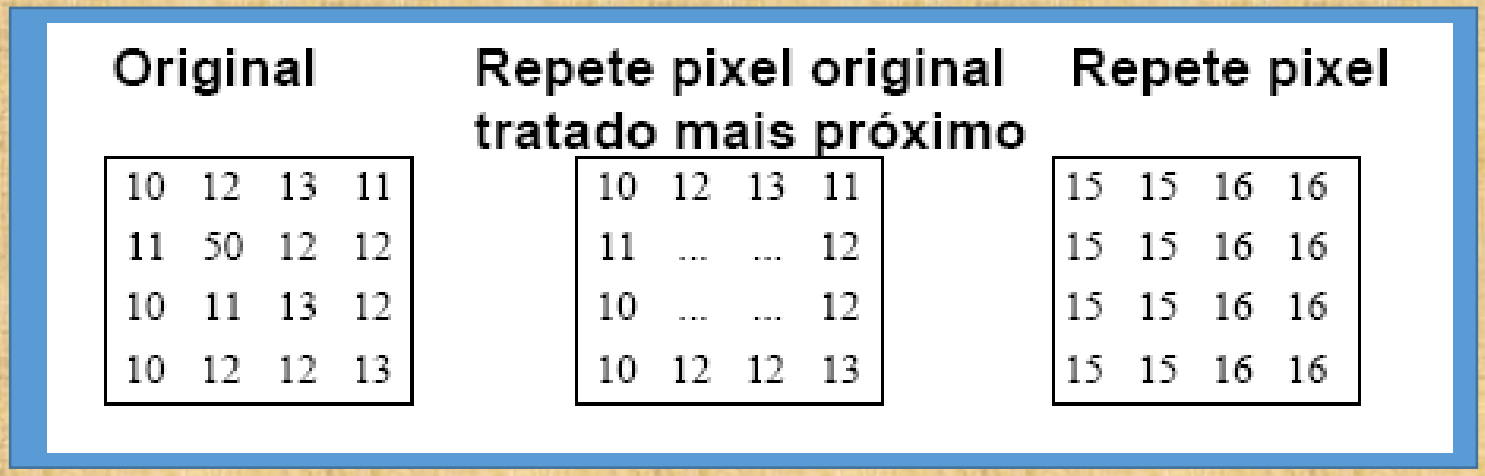
- O problema da borda está em como resolver o problema da primeira linha, da última linha, da primeira coluna e da última coluna de pixels. Nestes locais as máscaras não correspondem diretamente aos pixels da imagem.

- Várias estratégias podem ser implementadas para preencher estes pixels:

  - É possível repetir para estes locais os valores dos pixels originais, ou repetir o valor do pixel tratado mais próximo.

  - É possível considerar a Convolução apenas dos pixels que intersectam o filtro (a máscara).

Problemas com as bordas



A **Filtragem por Convolução** é uma operação pontual que pode tomar bastante tempo de processamento. Os resultados da aplicação destes filtros resultam em imagens com baixo brilho e baixo contraste.

Filtros **Passa-Baixas**

- Eliminam altas frequências, sendo usados para eliminar ruídos em imagens.

- O ruído é uma fonte de alta frequência.

- O efeito produzido é uma desfocalização caracterizada por uma imagem borrada. Esta desfocalização depende das dimensões do filtro, quanto maior a dimensão do filtro, maior será a desfocalização.

Filtros **Passa-Baixas**

- O efeito visual é a suavização da imagem pela redução das variações nos níveis de cinza que dão a aparência de serrilhado nos patamares de intensidade.

- Faz a imagem perder nitidez, causando o efeito borrado na imagem.

- Elimina ruídos.

Filtros **Passa-Baixas**

Imagem

Com Ruído

Filtro da Média

Filtro Gaussiano

Filtro da Média

O pixel central é a média aritmética dos pixels dentro da área da janela.

Máscara de convolução *n* x *n* com todos seus coeficientes iguais a 1 e depois dividindo-se o valor obtido pelo número de *pixels* da máscara (*n2*)

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

3 x 3

$$Z = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

5 x 5

$$Z = \frac{1}{49}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

7 x 7

Filtro da Média

1) **Filtro da Média**: pixel central é a média aritmética dos pixels dentro da área da janela.

Filtro da Média

```
using System;
using System.Collections.Generic;
using System.Text;
using DoisDWeb.Lib;
using DoisDWeb.Projetos.DigitalImage.Tools;
using System.Windows.Media;

namespace DoisDWeb.Projetos.DigitalImage.Metodos
{
   public partial class FiltroMedia : MetodoFiltroBase
   {
      private static int[,] mascara = new int[,]
     {
       { 1,  1,  1 },
       { 1,  1,  1 },
       { 1,  1,  1 }
     };

      public override ImageBytes Executar(ImageBytes image)
      {
         int width = image.Width;
         int height = image.Height;
         int widthM1 = width - 1;
         int heightM1 = height - 1;
         int i, j;
         double v, z = 0;
         byte novoValor;
         ImageBytes imageOrigem = image.Clone();
         for (int y = 1; y < heightM1; y++)
         {
            for (int x = 1; x < widthM1; x++)
            {
               z = 0;
               for (i = 0; i < 3; i++)
               {
                  for (j = 0; j < 3; j++)
                  {
                     Color color = imageOrigem.GetPixel(x + (i - 1), y + (j - 1));
                     z += color.R * mascara[i, j];
                  }
               }
               novoValor = Convert.ToByte( z / 9);
               image.SetPixel(x, y, novoValor, novoValor, novoValor, 255);
            }
         }
         return image;
      }
   }
}
```

Filtro da Moda

O nível de cinza do pixel central é o nível de cinza mais populoso dentro da janela de dimensão do filtro.

Máscaras:

Original		
10	12	13
11	50	12
10	10	13

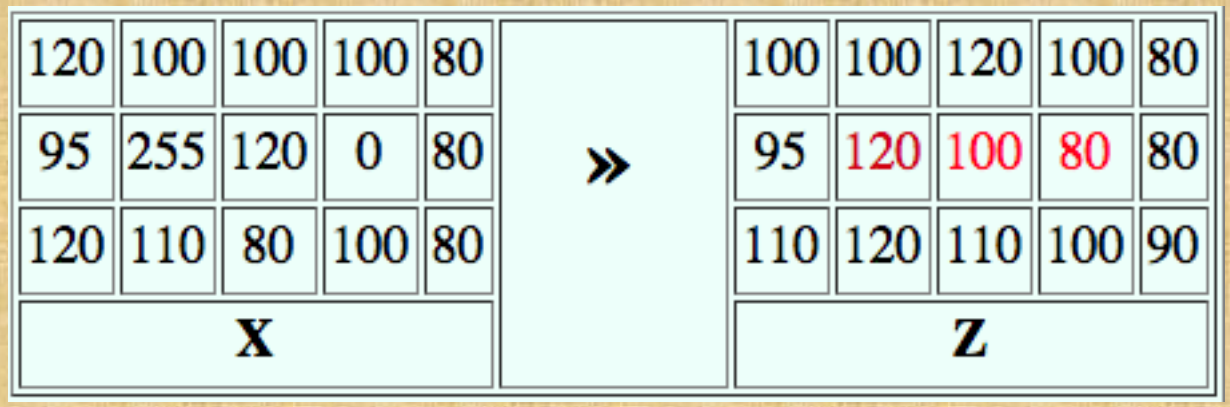
Filtrado		
10	12	13
11	10	...
...	...	...

MODA = 10

Este filtro é usado para homogeneizar imagens temáticas, ou para reduzir ruídos mantendo o máximo de informação na imagem.

Filtro da Moda

Considere a imagem X apresentada abaixo. Aplicando-se à imagem X uma filtragem de moda, considerando-se vizinhança 8, obteremos a imagem Z de saída apresentada à direita.



O filtro da moda garante que o conjunto de valores digitais da imagem de saída é um subconjunto do domínio de valores da imagem de entrada. Assim, não são criados níveis digitais diferentes daqueles presentes na imagem de entrada.

Filtros **da Mediana**

O nível de cinza do pixel central é o nível de cinza intermediário do conjunto ordenado de níveis de cinza dentro da janela da máscara.

Máscaras:

Original		
28	32	29
28	30	27
28	32	29

Filtrada		
27	27	...
27	27	...
...	...	...

Ordenação :

24 26 22 28 30 27 28 32 29

22 24 26 |27 28| 29 30 32

NC do pixel central = 27 (ou 28)

Este é um filtro complexo por envolver ordenação. Mas sua aplicação suaviza a imagem preservando a informação de bordas na imagem.

Filtros **da Mediana**

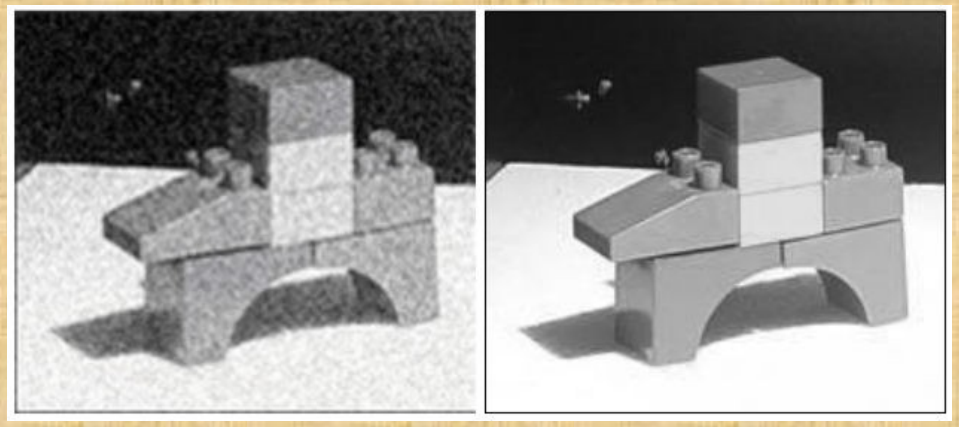
Considere a imagem **X** apresentada abaixo. Aplicando-se à imagem X uma filtragem de mediana, considerando-se vizinhança 9, obteremos imagem **Z** de saída apresentada à direita.

100	100	120	100	80		100	100	120	100	80
95	255	120	0	115	»	95	110	110	110	115
110	120	110	110	80		110	120	110	100	80
X						**Z**				

Filtro da Mediana

As principais aplicações são:

- ruído é impulsivo (do tipo não contínuo, consistindo em pulsos irregulares de grandes amplitudes),
- ruído do tipo *sal e pimenta* (representando descontinuidades abruptas e isoladas na imagem).

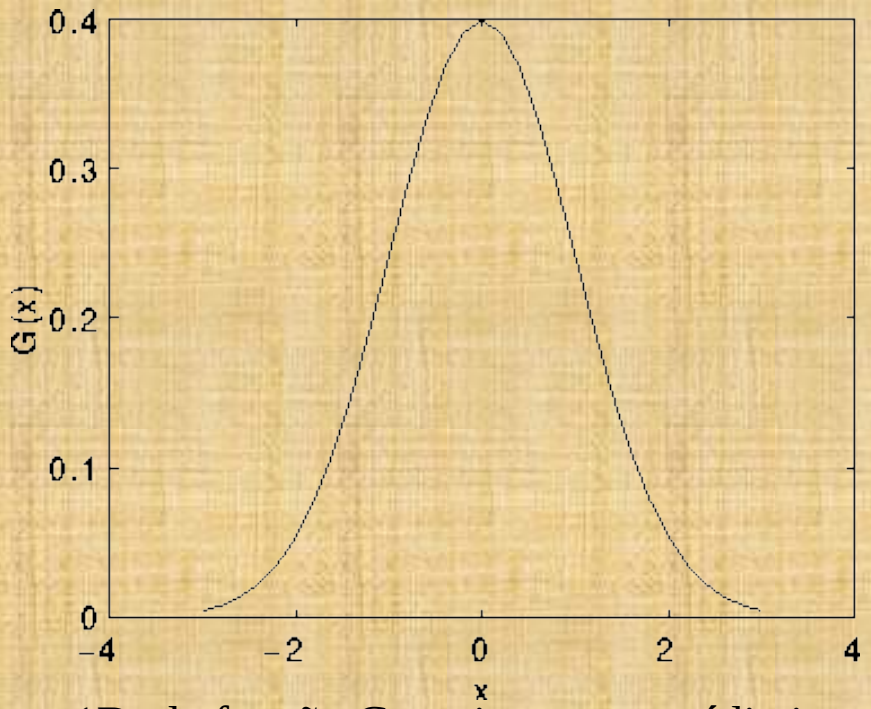


Resultado da aplicação do filtro de mediana (b) na imagem Blocos degradada com ruído impulsivo (a) (Silva, 2004).

Filtro Gaussiano

É baseado em uma aproximação digital da função gaussiana. O Filtro Gaussiano em 1-D é descrito por:

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

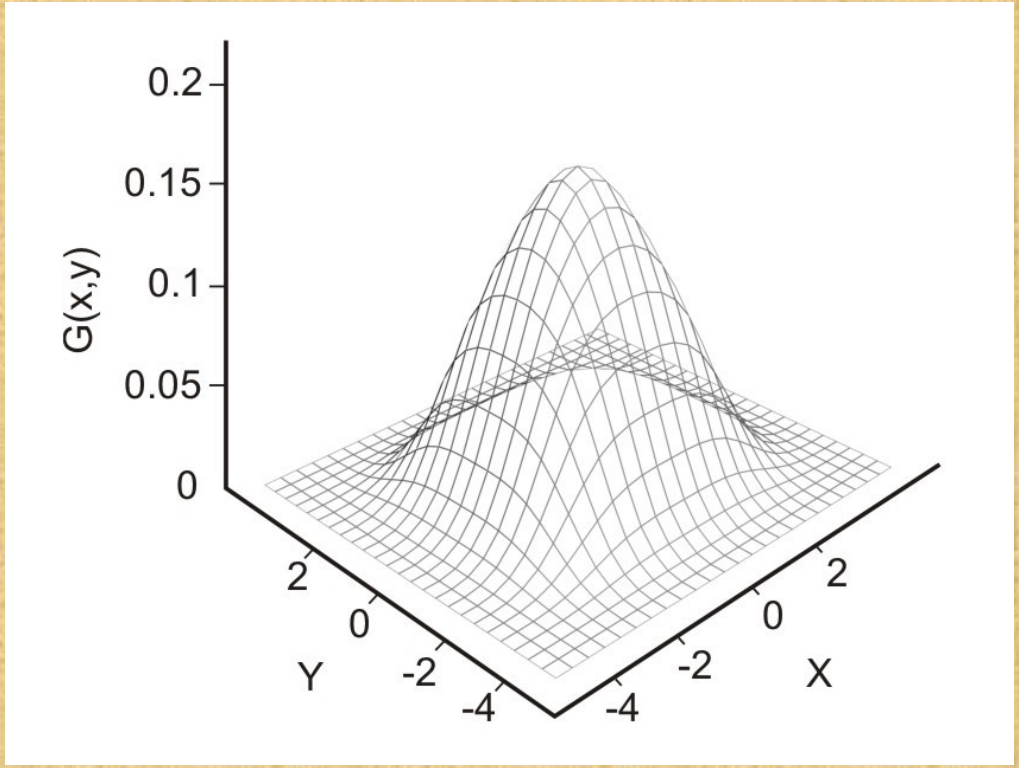


Forma 1D da função Gaussiana com média igual a zero e desvio padrão igual a um.

Filtro Gaussiano

O Filtro Gaussiano em 2-D é descrito por:

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



Representação da função Gaussiana em 2D com média em (0,0) e desvio padrão $\sigma = 1$.

Filtro Gaussiano

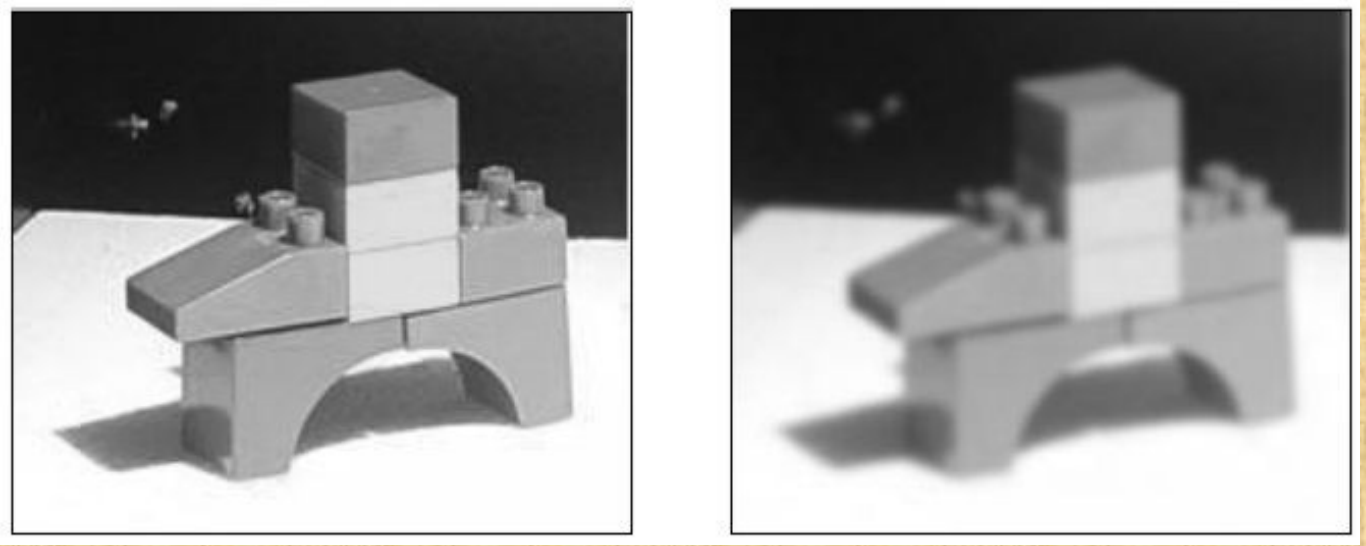
Forma aproximada da Gaussiana para $\sigma = 1{,}0$ no kernel 5x5

$$Z = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Forma discreta 3x3 aproximada da função Gaussiana

$$Z = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Filtro Gaussiano



(a) (b)

Resultado da aplicação do filtro Gaussiano (b) à imagem Blocos (a) (Silva, 2004).

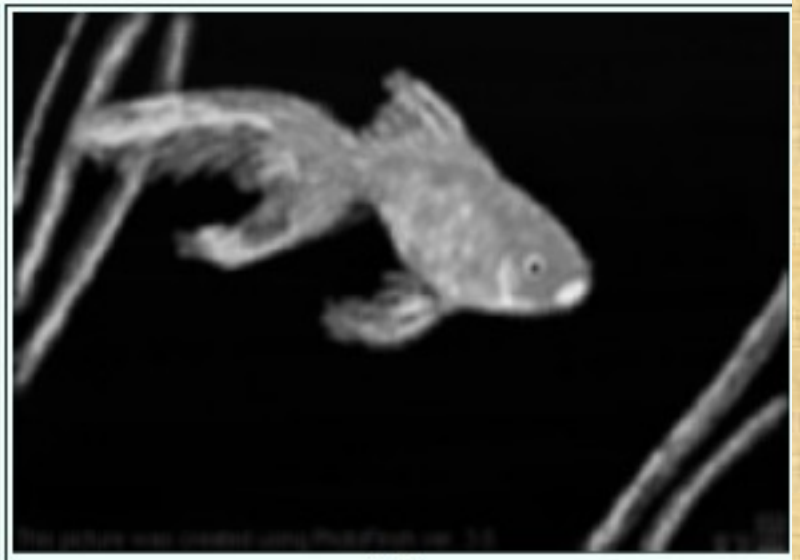
Filtro Gaussiano

Matriz 3X3

(a)

Matriz 5X5

(b)



(c)

Filtro Gaussiano

Filtro Gaussiano

```
using System;
using System.Collections.Generic;
using System.Text;
using DoisDWeb.Lib;
using DoisDWeb.Projetos.DigitalImage.Tools;
using System.Windows.Media;

namespace DoisDWeb.Projetos.DigitalImage.Metodos
{
   public partial class FiltroGaussiano : MetodoFiltroBase
   {
      private static int[,] mascara = new int[,]
                {
                        { 1,  2,  1 },
                        { 2,  4,  2 },
                        { 1,  2,  1 }
                };
public override ImageBytes Executar(ImageBytes image)
      {
         ImageBytes imageOrigem = image.Clone();
         int width = image.Width;
         int height = image.Height;
         int widthM1 = width - 1;
         int heightM1 = height - 1;
         int i, j;
         double v, z = 0;
         byte novoValor;
            for (int y = 1; y < heightM1; y++)
            {
               for (int x = 1; x < widthM1; x++)
               {
                  z = 0;
                  for (i = 0; i < 3; i++)
                  {
                     for (j = 0; j < 3; j++)
                     {
                        Color color =
imageOrigem.GetPixel(x + (i - 1), y + (j - 1));
                        z += color.R * mascara[i, j];
                     }
                  }
                  novoValor = Convert.ToByte( z / 16);
                  image.SetPixel(x, y, novoValor, novoValor,
novoValor, 255);
               }
            }
            return image;
      }
   }
}
```

Limiarização - (Thresholding)

Um objeto pode ser entendido como uma região formada por pixels contíguos que tenham em comum uma faixa de intensidades.

- A limiarização usa a intensidade dos pixels para distinguí-los.
  - O processo se baseia na análise do histograma da imagem.

- Este é um processo muito veloz e em geral realizado em um passo. É sempre o primeiro método a se tentar e muitas vezes fornece resultados bons em situações “impossíveis”.

- *Thresholding* não funciona bem em imagens com iluminação não uniforme e com baixo contraste entre as diversas regiões.

- Existem processos de *thresholding* manual e automático.

Limiarização - (Thresholding)

Analisamos a similaridade dos níveis de cinza da imagem extraindo os objetos de interesse através da seleção de um limiar *T* que separa os agrupamentos de níveis de cinza.

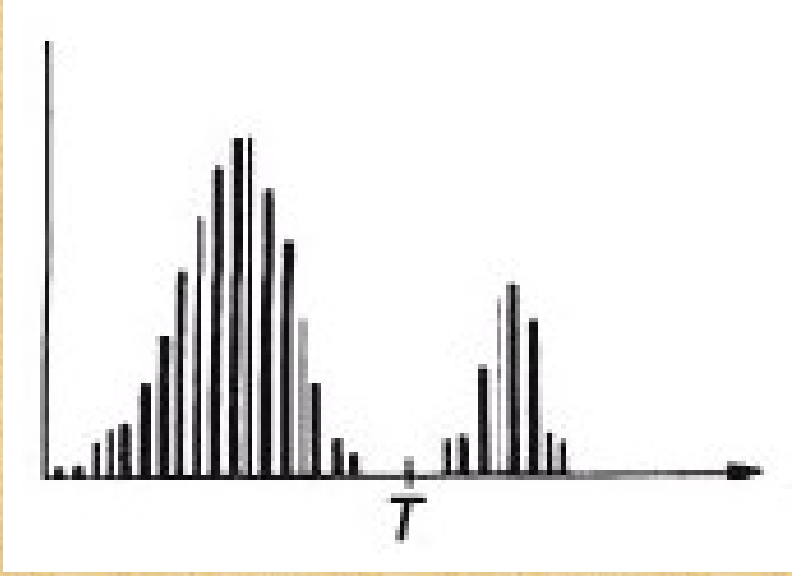
Uma imagem limiarizada *g(x,y)* é definida como:

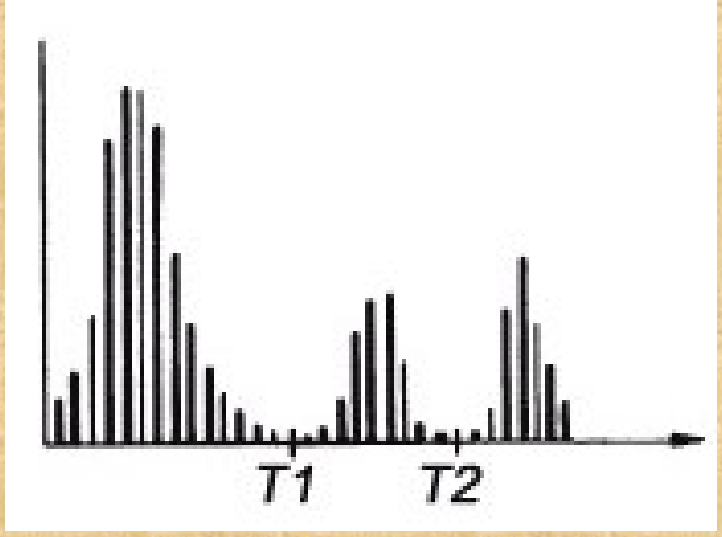
$$g(x,y) = \begin{cases} 1 \text{ se } f(x,y) \geq T \\ 0 \text{ se } f(x,y) \leq T \end{cases}$$

Onde *f(x,y)* corresponde ao nível de cinza do ponto, os pixels rotulados com 1 correspondem aos objetos e os pixels rotulados com 0 correspondem ao fundo e ***T*** é um valor de tom de cinza predefinido denominado limiar.

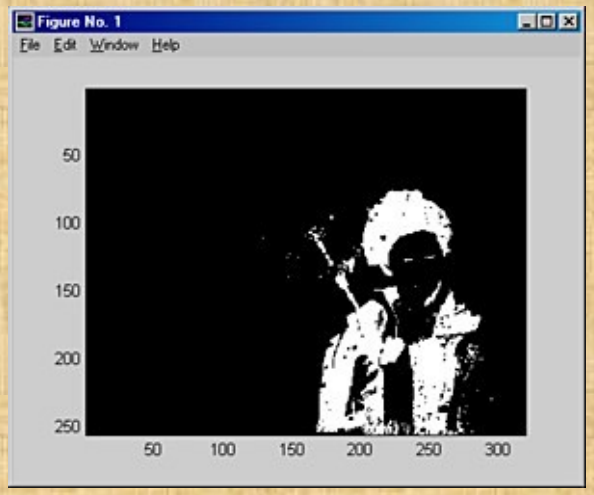
Limiarização - (Thresholding)

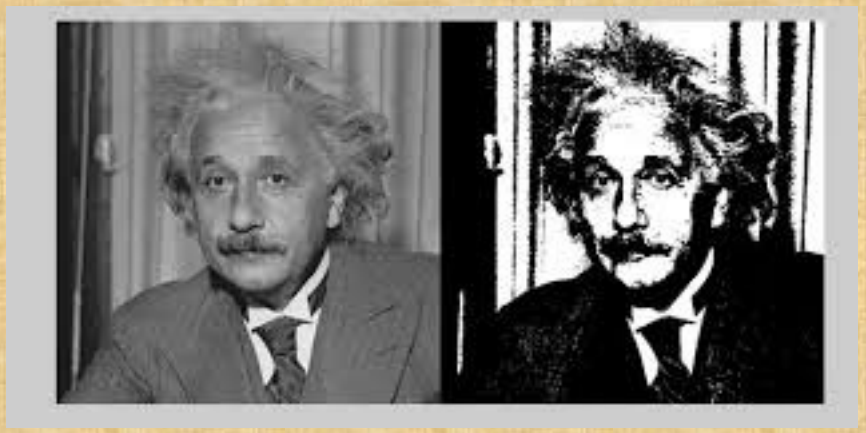
Varre-se a imagem, pixel por pixel, rotulando-se cada pixel como sendo do objeto ou do fundo, dependendo se o nível de cinza daquele pixel for maior ou menor que *T. O su*cesso desse método depende inteiramente de quão bem o histograma pode ser particionado.





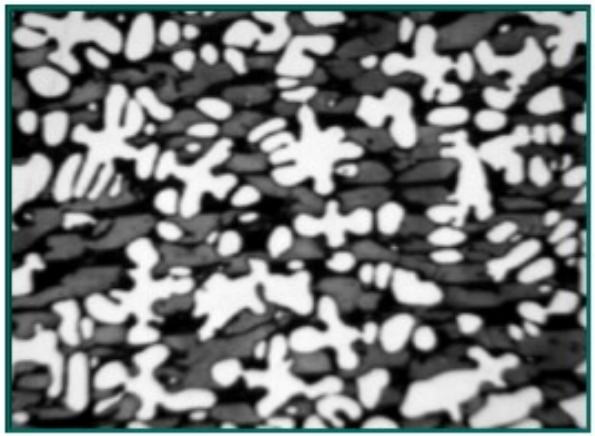
Limiarização - (Thresholding)

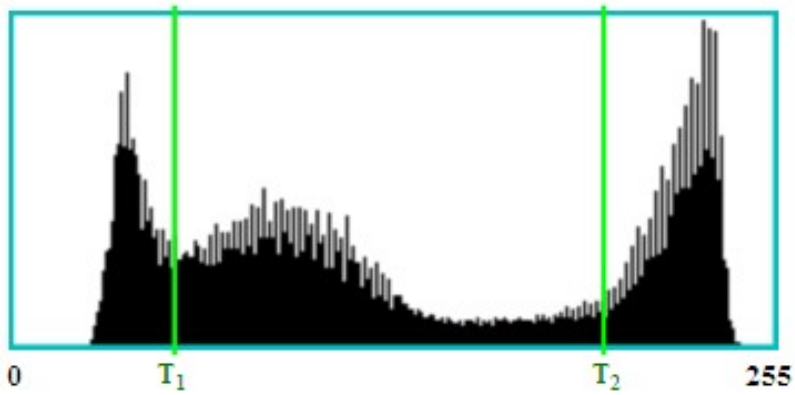




Limiarização - (Thresholding)

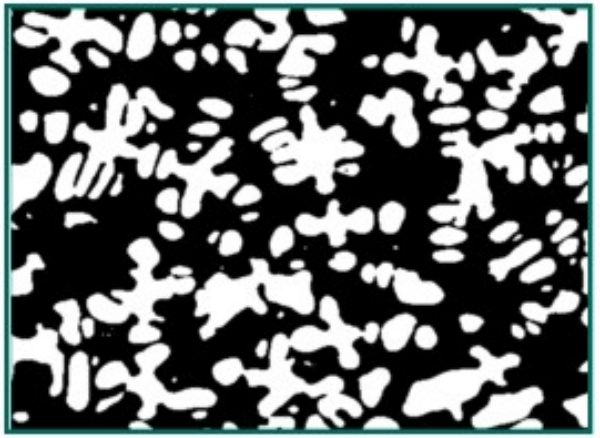
Limiarização - (Thresholding)





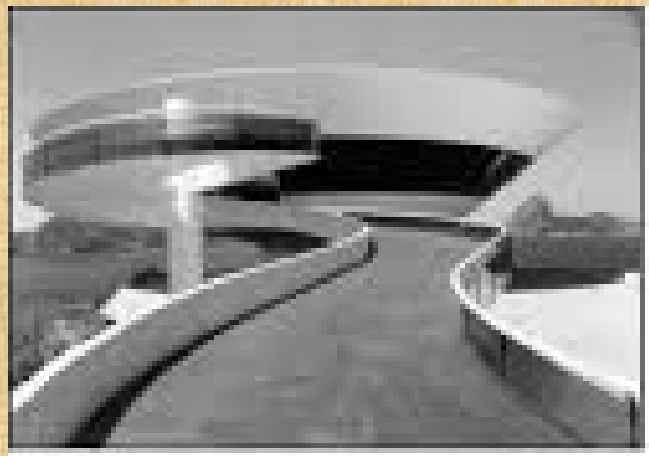
Fase escura ($t<T_1$)

Fase média ($T_1<t<T_2$)

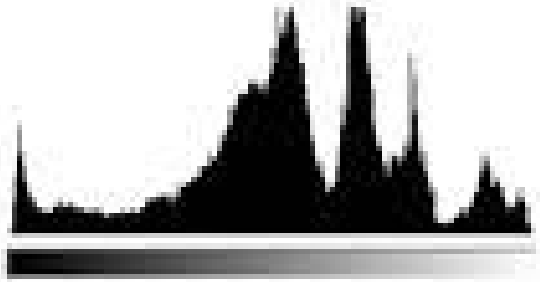


Fase clara ($t>T_2$)

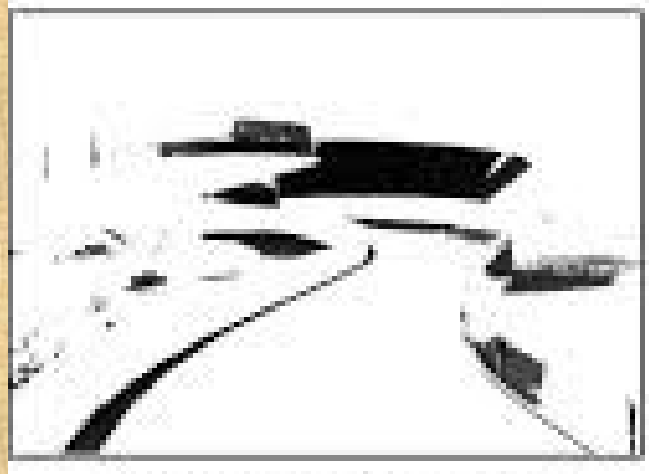
Limiarização - (Thresholding)



(a) Imagem com 256 tons de cinza



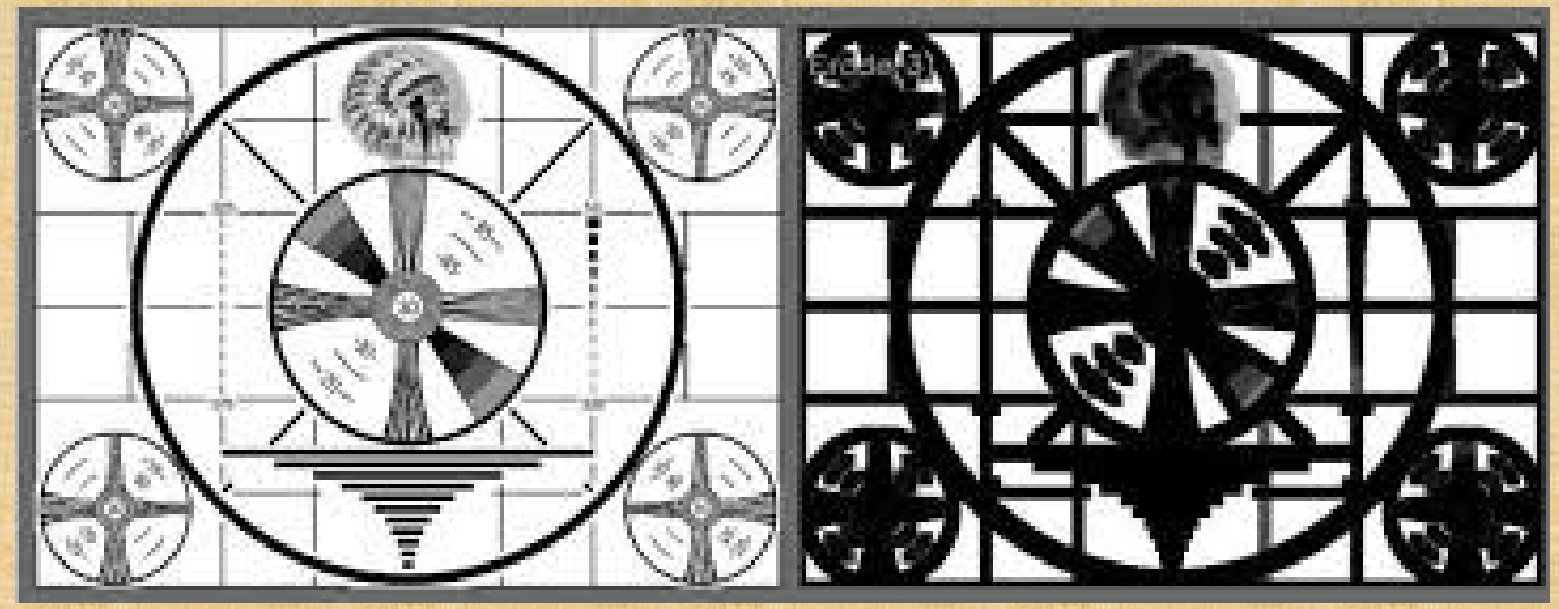
(b) Histograma

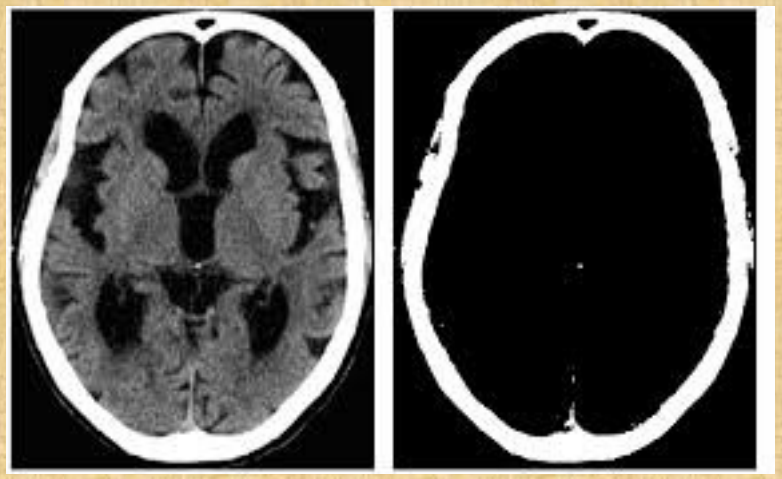


(c) Limiarização com valor 80

(d) Limiarização com valor 150

Limiarização - (Thresholding)





Limiarização - (Thresholding)

Threshold = 50

Threshold = 100

Threshold = 150

Threshold = 200

Threshold = 250

Bibliografia

- [1] GONZALEZ R.; WOODS, R. Processamento de Imagens Digitais, 2000.
- [2] MATARREDONA, E. O Uso do Processamento de Imagens Aplicadas na Radiologia Médica, 1994.
- [3] SAMPAIO, R; CATALDO, E.; RIQUELME, R. Introdução à Análise e ao Processamento de Sinais Usando o MATLAB, 1998.
- [4] Software Digital Image. Disponível em http://digitalimage.2dweb.com.br/ com usr: visitante e senha: visitante.