IAP

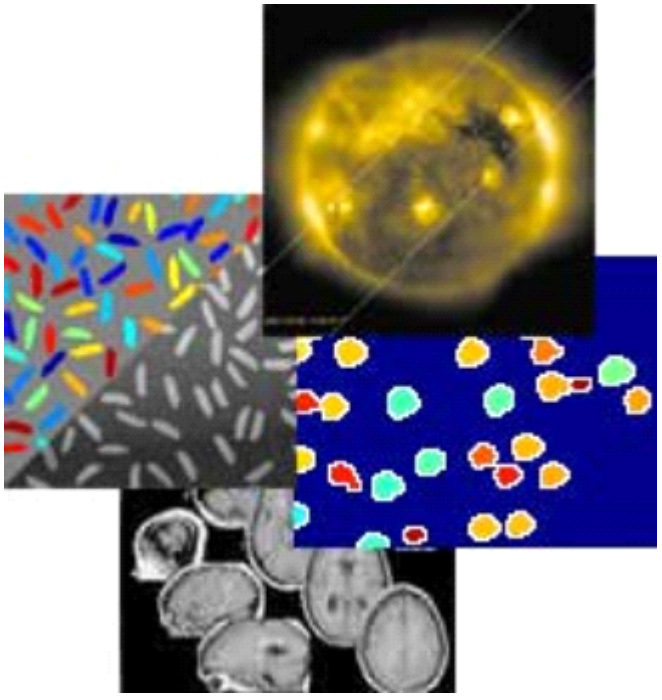
Processamento e Analise de Imagem

sábado, Janeiro 14, 2006

Gustavo Arranhado Nº6688

Sumário e Objectivos

Processar e analisar uma determinada imagem é muito importante nos dias de hoje, pois permite descodificar certos aspectos existentes na informação de modo a satisfazer as nossas necessidades. Podemos encontrar diversos tipos de softwares de tratamento de imagem, desde aqueles que nos permitem modificar a apresentação imagem aos que nos permitem extrair determinadas informações ocultas.



Este trabalho tem como objectivo uma aplicação informática capaz de analisar e aplicar diversas operações a uma determinada imagem.

Índice

Índice de Figuras

Índice de Equações e Algoritmos

1. Introdução

Análise e reconhecimento de imagens é uma área de grande relevância por suas inúmeras aplicações.

Um sistema genérico para processamento e análise de imagens de documentos é composto pelos seguintes módulos:

1. **Formação** - A sequência começa com uma imagem (por exemplo uma imagem captado por uma câmara de vídeo ligada a um microscópio).
2. **Digitalização** - O sinal da câmara passa por um processo de *Digitalização* e a imagem digital resultante é armazenada na memória do computador. Esta imagem poderá já ser o resultado final e ser visualizada num ecrã ou imprimida numa impressora.
3. **Pré-Processamento** - No entanto, poderá ser necessário algum processamento para melhorar a qualidade da imagem. Ruído ou pouco contraste são exemplos típicos de problemas que podem ser resolvidos com a fase de *Pré-processamento*.
4. **Segmentação** - Se o objectivo é a extracção de informação quantitativa, então a sequência continua com o passo *Segmentação* no qual são identificados e discriminados objectos na imagem. Este é um passo crítico e muito difícil porque tenta imitar a sofisticada capacidade cognitiva humana.
5. **Pós-Processamento** - Devido a esta complexidade, é por vezes necessário corrigir a segmentação resultante através do passo *Pós-Processamento* onde os chamados operadores morfológicos são aplicados.
6. **Extracção de Atributos** - A imagem resultante do pós-processamento é formado pelos objectos desejados que poderão ser analisados na fase *Extracção de Atributos* para a obtenção de parâmetros quantitativos tais como tamanhos, formas, posições, texturas, etc. Estes dados podem ser apresentados através de gráficos ou serem alvo de analise estatística.

2. Processamento e Analise de Imagem

2.1. Imagem

Uma imagem digital (*ver Figura 1*) é uma matriz de pixels. A sua qualidade depende da resolução (quantidade de pixels) e da quantização (quantidade de cores).

Figura 1 - Imagem digital

2.1.1. Formatos

O byte é a unidade mínima de armazenamento. Por definição um byte é constituído por 8 bits. Cada bit pode representar dois valores (0 ou 1). A combinação de *n* bits permite representar 2^n valores distintos. É sempre ocupado um byte mesmo que não sejam usados os 8 bits que o constituem.

A quantidade de memória ocupada por uma imagem digital depende da resolução, da quantização e do algoritmo de compressão usado.

Determinar tamanho de uma imagem sem compressão

$$Tamanho = Width \times Height \times Bpp$$

BMP

Este formato foi desenvolvido pela Microsoft e é o formato nativo do sistema operativo Windows. Suporta imagens em tons de cinzento de 8 bits e imagens a cores RGB de 24 bits. É um formato sem compressão mas pode usar o método de compressão sem perdas RLE (Run Length Encoding).

Estrutura

Um arquivo de BMP consiste em 3 ou 4 partes (*ver Figura 2*). A primeira parte é um cabeçalho, isto é seguido por uma secção de informação, se a imagem é indexada então a paleta segue, e último de tudo são os dados dos pixels. A posição dos dados de imagem com respeito ao sart do arquivo é contida no cabeçalho. Informação como a largura de imagem e altura, o tipo de compressão, o número de cores é contido no cabeçalho de informação.

Figura 2 – Estrutura de um BMP

Cabeçalho

O cabeçalho consiste nos seguintes campos. Notemos que estamos assumindo int curto de 2 bytes, int de 4 bytes, e int longo de 8 bytes. O cabeçalho é 14 bytes em comprimento.

```
typedef struct {
   unsigned short int type;        /* Magic identifier            */
   unsigned int size;              /* File size in bytes          */
   unsigned short int reserved1, reserved2;
   unsigned int offset;            /* Offset to image data, bytes */
} HEADER;
```

Informação

Os dados de informação da imagem que segue é de 40 bytes em comprimento, é descrito no struct dado abaixo. Os campos de maior interesse é a largura de imagem e altura, o número de bits por pixel (deveria ser 1, 4, 8 ou 24,), o número de camadas (assumido para ser 1 aqui), e o tipo de compressão (assumido para ser 0 aqui).

```
typedef struct {
   unsigned int size;               /* Header size in bytes      */
   int width,height;                /* Width and height of image */
   unsigned short int planes;       /* Number of colour planes   */
   unsigned short int bits;         /* Bits per pixel            */
   unsigned int compression;        /* Compression type          */
   unsigned int imagesize;          /* Image size in bytes       */
   int xresolution,yresolution;     /* Pixels per meter          */
   unsigned int ncolours;           /* Number of colours         */
   unsigned int importantcolours;   /* Important colours         */
} INFOHEADER;
```

Imagem de 24 bits

Os dados mais simples para ler são os das imagens de 24 bits. Neste caso os dados de imagem seguem imediatamente depois do cabeçalho de informação, quer dizer, não há nenhuma paleta de cor. Consiste em três bytes por pixel em b, g, ordem de r. Cada byte dá a saturação aquele componente de cor, 0 para preto e 1 para branco (completamente saturado).

Imagem indexada

Se a imagem é indexada então imediatamente depois do cabeçalho de informação haverá uma mesa de cores de infoheader.ncolours, cada um de 4 bytes. Os primeiros três bytes correspondem a b, g, componentes de r, o último byte está reservado, mas poderia representar o canal alfa obviamente. Para imagens de 8 bits de greyscale este índice de cor será geralmente há pouco uma rampa de greyscale. Se fazer as somas, então o comprimento do cabeçalho mais o comprimento do bloco de informação mais 4 vezes o número de cores de paleta deveria igualar os dados de imagem compensados.

```
typedef struct                /**** Colormap entry structure ****/
{
   unsigned char  rgbBlue;          /* Blue value */
   unsigned char  rgbGreen;         /* Green value */
   unsigned char  rgbRed;           /* Red value */
   unsigned char  rgbReserved;      /* Reserved */
} RGBQUAD;
```

IAP

Este formato foi desenvolvido para este trabalho. É um formato que usa o método de compressão sem perdas Adaptive Huffman.

Algumas aplicações exigem que a compressão dos dados não incorra em perda de informação, como em aplicações médicas, espaciais, etc. A primeira opção de se fazer isto é a remoção somente da redundância de código. O uso de códigos de tamanhos variáveis como o código de Huffman, que produz o menor número possível de símbolos de código por símbolo de entrada.

Princípios

Seja $x_n = x_1x_2 \ldots x_n$ a sequência de valores (escalares ou vectoriais) produzida por uma fonte de informação até ao instante n. As técnicas preditivas baseiam-se na codificação da sequência $r^n = r_1r_2 \ldots r_n$, em vez da sequência x_n, em que:

$$r_k = x_k - x'_k$$

e

$$x'_k = p(x^{k-1})$$

- Designamos os x'_k de estimativas e os valores da sequência r^n de residuais;
- A função p() é o estimador ou preditor;
- Objectivo principal: $H(r^n) < H(x^n)$.

Quanto mais parecido do valor real for o valor estimado, mais eficiente será a codificação. Logo, é natural que a construção dos preditores seja um aspecto bastante importante neste tipo de técnicas. Durante a descodificação o sinal residual é reconstruído, gerando uma sequência r'^n que

pode ou não ser igual a r^n, dependendo se a compressão foi efectuada com ou sem perdas. O descodificador tem que ser capaz de gerar a sequência estimada x'^n (igual ao que foi utilizado pelo codificador), ou seja, o preditor não pode introduzir erro no processo de codificação / descodificação.

Para que o valor estimado no descodificador seja igual ao valor estimado pelo codificador têm que se verificar duas condições:

- O preditor tem que ser causal, para que o descodificador também possa efectuar o cálculo da estimativa;
- No caso de compressão com perdas, devem-se utilizar para predição no codificador os valores afectados pelo erro, isto é, a sequência x'^n, em vez dos valores reais x^n.

Só assim o descodificador é capaz de efectuar uma estimativa igual à do codificador. Em geral, a complexidade do preditor depende de dois factores:

- Número de valores utilizados para efectuar as estimativas (ordem do preditor);
- Distribuição espacial (ou temporal) destes valores.

Este tipo de estimador pode ser facilmente estendido para ordens superiores, usando-se os últimos *k* elementos de imagem processados. Contudo, para ordens superiores a 3 ou 4, a sua eficácia não é alterada substancialmente. Isto acontece porque, de facto, as imagens são bidimensionais, e não sequências unidimensionais de dados.

Uma das vantagens das técnicas preditivas é permitirem, facilmente, realizar codificadores sem perdas. De facto, a maioria dos codificadores sem perdas para áudio e imagem são baseados em técnicas preditivas. Contudo, para que isso seja possível, é necessário que as estimativas geradas pelo preditor sejam independentes da plataforma em que os codificadores / descodificadores forem implementados. Em geral, este requisito obriga a que todos os cálculos efectuados pelo preditor só envolvam aritmética inteira.

Codificação de Huffman

A codificação de Huffman é um método de compressão que usa estatísticas do conjunto de dados a ser comprimido para determinar códigos de tamanho variável. Probabilidade de ocorrência são computadas em todos os possíveis valores na imagem e esses são então ordenados.

O menor código é associado aos dados com maior probabilidade de ocorrer e assim por diante. Desde que o código seja de tamanho variável, a associação de código é feita pela relação da menor probabilidade recursivamente até que uma árvore binária seja gerada com uma raiz de duas probabilidades.

O código de Huffman primeiro cria uma série de reduções de entrada, ordenando pela probabilidade de ocorrência dos símbolos e combinado os dois símbolos de menor probabilidade em um símbolo.

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4 →	0.6
a_6	0.3	0.3	0.3	0.3 →	0.4
a_1	0.1	0.1 →	0.2 →	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06 →	0.1			
a_5	0.04				

Em um segundo passo, os códigos são gerados iniciando-se no fim do primeiro passo, que produz o menor número possível de símbolos de código por símbolo de entrada. No exemplo abaixo, o tamanho médio do código é:

$$L_{AVG} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2 bits / simbolo$$

Original source			Source reduction							
Sym.	Prob.	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00 ←	0.4	1
a_1	0.1	011	0.1	011	0.2	010 ←	0.3	01 ←		
a_4	0.1	0100	0.1	0100 ←	0.1	011 ←				
a_3	0.06	01010 ←	0.1	0101 ←						
a_5	0.04	01011 ←								

O código de Huffman é óptimo a tabela mostra tamanhos médios para os diferentes métodos.

Source symbol	Probability	Binary Code	Huffman	Truncated Huffman	B_2-Code	Binary Shift	Huffman Shift
Block 1							
a_1	0.2	00000	10	11	C00	000	10
a_2	0.1	00001	110	011	C01	001	11
a_3	0.1	00010	111	0000	C10	010	110
a_4	0.06	00011	0101	0101	C11	011	100
a_5	0.05	00100	00000	00010	C00C00	100	101
a_6	0.05	00101	00001	00011	C00C01	101	1110
a_7	0.05	00110	00010	00100	C00C10	110	1111
Block 2							
a_8	0.04	00111	00011	00101	C00C11	111 000	00 10
a_9	0.04	01000	00110	00110	C01C00	111 001	00 11
a_{10}	0.04	01001	00111	00111	C01C01	111 010	00 110
a_{11}	0.04	01010	00100	01000	C01C10	111 011	00 100
a_{12}	0.03	01011	01001	01001	C01C11	111 100	00 101
a_{13}	0.03	01100	01110	100000	C10C00	111 101	00 1110
a_{14}	0.03	01101	01111	100001	C10C01	111 110	00 1111
Block 3							
a_{15}	0.03	01110	01100	100010	C10C10	111 111 000	00 00 10
a_{16}	0.02	01111	010000	100011	C10C11	111 111 001	00 00 11
a_{17}	0.02	10000	010001	100100	C11C00	111 111 010	00 00 110
a_{18}	0.02	10001	001010	100101	C11C01	111 111 011	00 00 100
a_{19}	0.02	10010	001011	100110	C11C10	111 111 100	00 00 101
a_{20}	0.02	10011	011010	100111	C11C11	111 111 101	00 00 1110
a_{21}	0.01	10100	011011	101000	C00C00C00	111 111 110	00 00 1111
Entropy	4.0						
Average length		5.0	4.05	4.24	4.65	4.59	4.13

Exemplo

Temos 9 caracteres diferentes, portanto, se utilizarmos um comprimento fixo, serão necessários 4 bits para representar cada letra.

	a	b	c	d	e	f	g	h	i
Frequência	50	25	15	40	35	25	10	25	25
Comprimento fixo	0000	0001	0010	0011	0100	0101	0110	0111	1000
Comprimento variável	11	0100	0001	011	001	0101	0000	100	101

Conforme esta na tabela, o texto tem 250 caracteres. Se estes estiverem codificados com tamanho fixo de bits, o arquivo ocuparia 250x4 bits = 1000 bits. Porém, se for utilizado o código de Huffman para se fazer uma nova codificação, o arquivo terá (50*2 + 25*4 + 15*4 + 40*3 + 35*3 + 25*4 + 10*4 + 25*3 + 25*3) = 775 bits, uma economia de 32,5%. Este índice não corresponde a economia verdadeira que o código de Huffman pode proporcionar. Deve-se levar em conta que quanto mais equilibrado for as ocorrências de cada caracter, menor será a economia.

Algoritmo Huffman

Para atribuir aos caracteres mais frequentes as codificações menores e, como geralmente é usado código binário, constrói-se uma árvore binária a partir dos caracteres de menor valor de probabilidade onde os nós representam a soma a frequência de ocorrência de todos as letras da sub-árvore correspondente e cujas folhas representam cada um dos caracteres da mensagem a ser codificada e sua frequência. A codificação de cada símbolo será representação do caminho da raiz até ao símbolo.

1) O alfabeto de entrada deve ser organizado em uma coluna em ordem decrescente de frequência com sua probabilidade ou o seu número de ocorrência no texto;
2) Defina um novo nó unindo dois cuja soma das probabilidades seja a menor possível;
3) Calcule a probabilidade deste nó como a soma das probabilidades dos grupos unidos (nós);
4) Para um dos nós filho usado para definir o novo nó atribuo o bit 0 e para o outro o bit 1;
5) Se restar mais de um grupo (sub-árvores) vá para o passo 2, senão passe para o passo 6;
6) Obter o código de cada letra pegando-se a sequência de bits atribuída aos grupos, da esquerda para a direita. Como o processo constitui uma estrutura de árvore binária, o código de cada elemento será a sequência de bits da raiz até a folha que corresponde ao caracter.

O caracter de maior frequência possui o menor comprimento que é exactamente a meta da compressão estatística. Assim acontece com os demais caracteres, em ordem proporcionalmente crescente do número de bits.

O Adaptive Huffman é basicamente o mesmo método, só que em vez de se verificar estatisticamente as frequências ao inicio, vão se verificando a medida que cada elemento entra na arvore, sendo esta arvora construída adaptativamente. Este método é mais lento e mais complexo que o inicial mas consegue obter melhores taxas de compressão.

Método de compressão adaptativo

Definir "modelo"
Para "*cada pixel*" **Fazer**
    Codificar "pixel"
    Actualizar "modelo"
Fim Para

A descompressão funciona da mesma forma. O problema é então actualizar o modelo. Para fazer um Huffman adaptativo, vamos apenas refazer a arvore cada vez que enviarmos um pixel o que pode tornar este algoritmo lento mas mais eficiente.

Algoritmo Adaptive Huffman

A árvore de Huffman é inicializada com um único nó, conhecido como o Not-Yet-Transmitted (NYT) ou código escape. Este código será enviado toda vez que um pixel novo que não está na árvore é encontrado, seguido pela sua codificação. Isto permite que o decompressor destinga entre um código e um pixel novo. O procedimento também cria um nó novo para o pixel e um NYT novo do nó NYT velho.

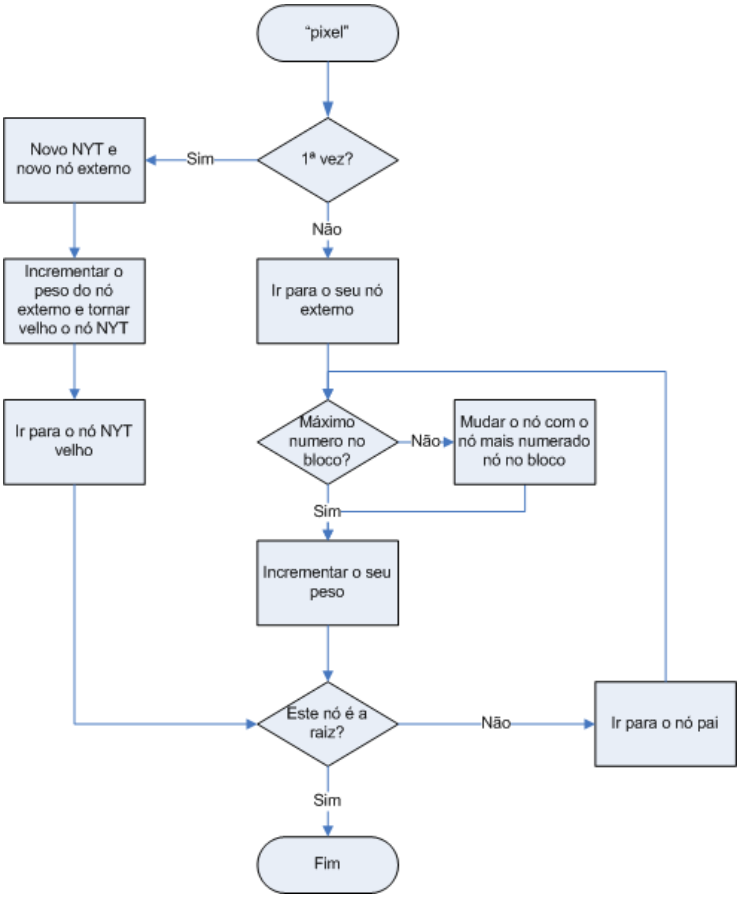
Sempre que um pixel que já está na árvore é encontrado, o código é enviado e o peso é aumentado.

Em ordem para trabalhar, precisamos acrescentar um pouco de informação adicional à árvore de Huffman para este algoritmo. Cada nó tem um peso e será nomeado um número de nó único. Também, todos os nós que têm o mesmo peso são chamados para estar no mesmo bloco.

Estes números de nó serão nomeados de tal um modo que:

1. Um nó com um peso mais alto terá um número de nó mais alto.
2. Um nó pai terá sempre um número de nó mais alto que os seus filhos.

Isto é conhecido como a propriedade de irmão, e o algoritmo de actualização troca nó simplesmente para ter certeza que esta propriedade é apoiada. Obviamente, o nó de raiz terá o número de nó mais alto porque tem o peso mais alto.



Depois que um contador ser incrementado, os movimentos de procedimento de actualização move para cima a árvore e inspecciona os antepassados do nó um de cada vez. Confere para ter certeza que o nó tem o nó mais alto em seu bloco, e se não, troca-o com o número de nó mais alto. Aumenta o peso do nó e vai para nó pai. Continua até que alcança o nó raiz. Como se vê, isto assegura que os nós com o peso mais alto são mais próximos do topo e têm códigos mais curtos.

Exemplo

Numa imagem sem compressão com o tamanho do 257 KB obtemos um ficheiro comprido com 167 KB, ou seja o ficheiro ficou a ~64% do tamanho original.

2.1.2. Histograma

Um histograma é uma representação gráfica da distribuição de frequências. O histograma de uma imagem fornece informação útil para fazer realce e análise da imagem. O histograma de uma imagem revela a distribuição dos níveis da imagem (*ver Figura 3*). É representado por um gráfico que dá o número de pixels na imagem para cada nível.

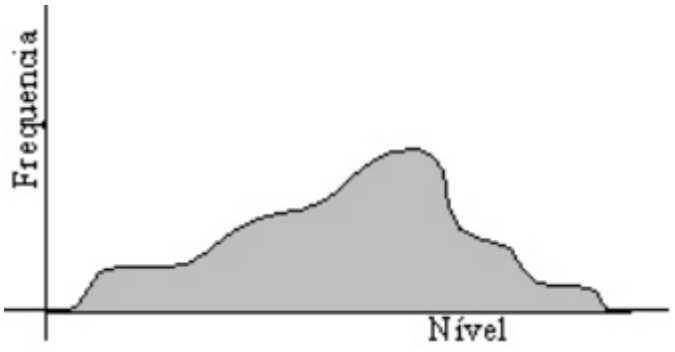


Figura 3 – Histograma de uma imagem

Algoritmo para obter o histograma de uma imagem

Definir $H(256)$
Para "*cada pixel da imagem*" **Fazer**
$H(valor) = H(valor) + 1$
Fim Para

2.1.3. Transformada

A transformada de fourier aplicada às imagens, tem como propósito a aplicação de filtros no domínio das frequências. Para isso é necessário transformar uma imagem para o domínio das frequências, edita-la nesse domínio e voltar a transformar a imagem no seu domínio normal (*ver Figura 4*).

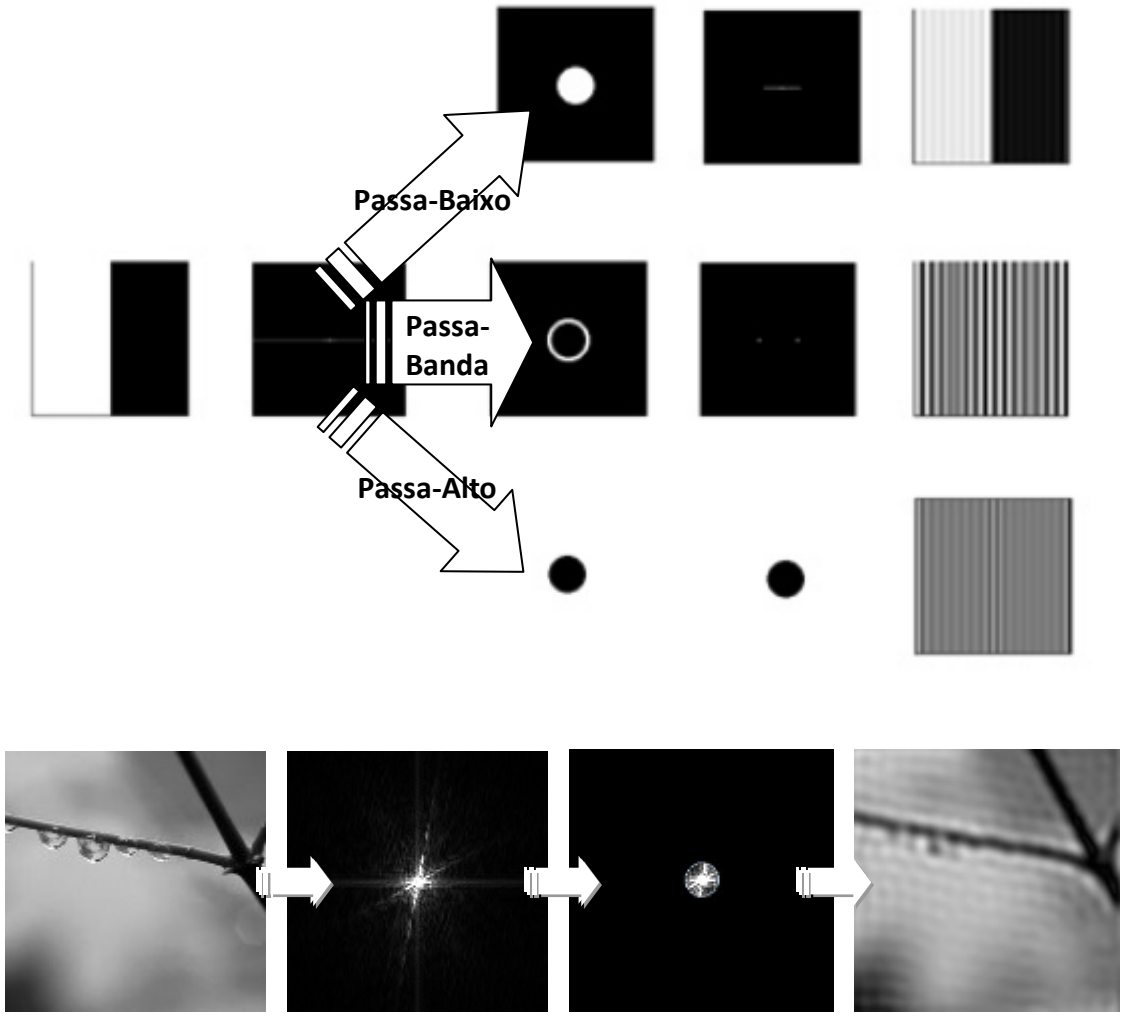


Figura 4 – Aplicação de filtros no domínio das frequências.

Transformada Fourier Discreta a 1D

DFT: $F(u)=\frac{1}{N}\sum_{x=0}^{N-1} f(x)e^{\frac{-j2\pi ux}{N}}, u=0,1...,N-1$

DFT inversa: $f(x)=\sum_{u=0}^{N-1} F(u)e^{\frac{j2\pi ux}{N}}, x=0,1...,N-1$

Transformada Fourier Discreta a 2D

DFT: $F(u,v)=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}, u=0,1\ldots,M-1, v=0,1,\ldots N-1$

DFT inversa: $f(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}F(x,y)e^{2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}, x=0,1\ldots,M-1, y=0,1,\ldots N-1$

Função DFT2

Esta função vai converter os pixels de uma imagem num formato de números complexos. Usando as equações acima mencionadas.

Definir $C(\max(D.Len1, D.Len2))$
Para "*cada i <D.Len1*" **Fazer**
  Para "*cada j <D.Len2 "* **Fazer**
    $C(j)=0$
    $\arg = -inversa \times 2 \times \pi \times j / D.Len1$
    Para "*cada k <D.Len1*" **Fazer**
      $\cos = \cos(k * \arg)$
      $\sin = \sin(k * \arg)$
      $C(j).\mathrm{Re} += D(i,k).\mathrm{Re} \times \cos - D(i,k).\mathrm{Im} \times \sin$
      $C(j).\mathrm{Im} += D(i,k).\mathrm{Re} \times sim + D(i,k).\mathrm{Im} \times \cos$
    Fim Para
  Fim Para
  Se "não inversa" **Fazer**
    Para "*cada j <D.Len1*" **Fazer**
      $D(i,j).\mathrm{Re} = C(j).\mathrm{Re} / D.Len1$
      $D(i,j).\mathrm{Im} = C(j).\mathrm{Im} / D.Len1$
    Fim Para
  Senão
    Para "*cada j <D.Len1*" **Fazer**
      $D(i,j).\mathrm{Re} = C(j).\mathrm{Re}$
      $D(i,j).\mathrm{Im} = C(j).\mathrm{Im}$
    Fim Para
  Fim Se
Fim Para
Para "*cada j <D.Len1*" **Fazer**
  Para "*cada i <D.Len2 "* **Fazer**
    $C(i)=0$
    $\arg = -inversa \times 2 \times \pi \times j / D.Len2$
    Para "*cada k <D.Len2*" **Fazer**

$\cos = \cos(k * \arg)$
$\sin = \sin(k * \arg)$
$C(i).\mathrm{Re} += D(k,j).\mathrm{Re} \times \cos - D(k,j).\mathrm{Im} \times \sin$
$C(i).\mathrm{Im} += D(k,j).\mathrm{Re} \times sim + D(k,j).\mathrm{Im} \times \cos$
Fim Para
Fim Para
Se "não inversa" **Fazer**
Para "*cada i <D.Len2*" **Fazer**
$D(i,j).\mathrm{Re} = C(i).\mathrm{Re} / D.Len1$
$D(i,j).\mathrm{Im} = C(i).\mathrm{Im} / D.Len1$
Fim Para
Senão
Para "*cada i <D.Len2*" **Fazer**
$D(i,j).\mathrm{Re} = C(i).\mathrm{Re}$
$D(i,j).\mathrm{Im} = C(i).\mathrm{Im}$
Fim Para
Fim Se
Fim Para

Função Filter

Esta função vai aplicar determinadas operações (adição, multiplicação, etc) na imagem no domínio das frequências modificando os complexos.

Definir $hw = I.Width^2$
$hh = I.Height^2$
Para "cada pixel da imagem" **Fazer**
$y' = y - hh$
$x' = x - hw$
$d = sqrt(x' \times x' + y' \times y')$
Se $(d > \max)$ ou $(d < \min)$ **Fazer**
$D(x,y).\mathrm{Re} = re$
$D(x,y).\mathrm{Im} = im$
Fim Se
Fim Para

Função Forward

A função forward vai então transformar uma determinada imagem no seu domínio das frequências.

Definir $DI(SI.Width, SI.Height)$
$D(SI.Width, SI.Height)$
Para "cada pixel da nova imagem" **Fazer**
$D(y,x) = SI(x,y)/255$
Fim Para
Para "cada pixel da nova imagem" **Fazer**
Se $((x+y)\,\&\,0x1) \neq 0$ **Fazer**
$D(y,x).\mathrm{Re} = -1 \times D(y,x).\mathrm{Re}$
$D(y,x).\mathrm{Im} = -1 \times D(y,x).\mathrm{Im}$
Fim Se
Fim Para
Call DFT2(false)
$scale = sqrt(DI.Width, DI.Height)$
Para "cada pixel da nova imagem" **Fazer**
$DI(x,y) = \max(\min(D(y,x).\mathrm{Magnitude} \times scale \times 255, 255), 0)$
Fim Para

Função Backward

Vai-se aplicar a transformada inversa para transformar os complexos alterados (ou não) numa nova imagem.

Call DFT2(true)
Para "cada pixel da nova imagem" **Fazer**
Se $((x+y)\,\&\,0x1) \neq 0$ **Fazer**
$D(y,x).\mathrm{Re} = -1 \times D(y,x).\mathrm{Re}$
$D(y,x).\mathrm{Im} = -1 \times D(y,x).\mathrm{Im}$
Fim Se
Fim Para
Para "cada pixel da nova imagem" **Fazer**
$DI(x,y) = \max(\min(D(y,x).\mathrm{Magnitude} \times 255, 255), 0)$
Fim Para

2.2. Operações

Uma operação feita a uma imagem é, transformar uma determinada imagem noutra imagem, esta modificada. As operações normalmente são a nível dos pixels.

2.2.1. Transformação

Aqui vamos transformar uma imagem a nível espacial como por exemplo redimensionar (*ver Figura 5*) uma imagem, entre outros como translações, rotações, etc.

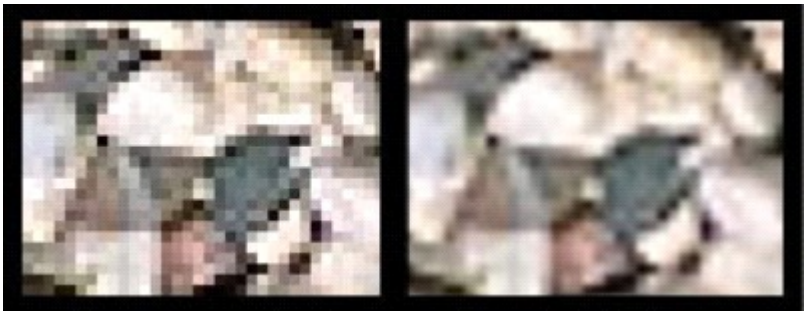


Figura 5 – Imagem redimensionada, sem e com interpolação

Redimensionar uma imagem

Sem Interpolação

Definir $xf = D.Width / S.Width$
$yf = D.Height / S.Height$
Para "cada pixel da nova imagem" **Fazer**
$D(x, y) = S(x * xf, y * yf)$
Fim Para

Com Interpolação

Para "cada pixel na nova imagem" **Fazer**

Definir

$$fx = x / S.Width \times D.Width$$
$$fy = y / S.Height \times D.Height$$
$$x_0 = floor(fx)$$
$$x_1 = ceil(fx)$$
$$y_0 = floor(fy)$$
$$y_1 = ceil(fy)$$
$$fu = fx - x_0$$
$$fv = fy - y_0$$

$$D(x,y) = (1-fu)\times(1-fv)\times S(x_0,y_0) + fu\times(1-fv)\times S(x_1,y_0) + (1-fu)\times fv\times S(x_0,y_1) + fu\times fv\times S(x_1,y_1)$$

Fim Para

Flip de uma imagem

Para "cada pixel da imagem" **Fazer**

$$I(x,y) = I(x, Height - y)$$

Fim Para

Mirror de uma imagem

Para "cada pixel da imagem" **Fazer**

$$I(x,y) = I(Width - x, y)$$

Fim Para

2.2.2. Realce

Frequentemente, não é possível perceber os detalhes espectrais de uma imagem. O realçamento de imagem consiste em mexer nos níveis da imagem (*ver Figura 6*), ou seja, intensidades, brilho, contraste, etc. Isto pode também implicar a utilização da informação existente no histograma de uma imagem.

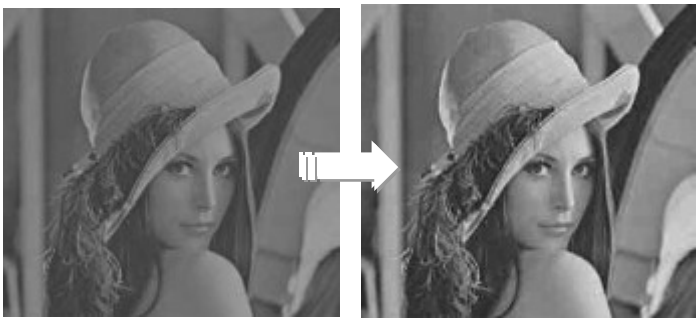


Figura 6 – Normalização de uma imagem

Clamp de uma imagem

Para "cada pixel da imagem" **Fazer**
$$I(x,y) = \max(\min(I(x,y), high), low)$$
Fim Para

Normalização

Com a normalização a informação contida nas extremidades do histograma vai ser realçada.

Definir $\min = 255, \max = 0$
Para "cada pixel da imagem" **Fazer**
$$\min = \min(\min, I(x,y))$$
$$\max = \max(\max, I(x,y))$$
Fim Para
Para "cada pixel da imagem" **Fazer**
$$I(x,y) = (I(x,y) - \min) \times (255/(\max - \min))$$
Fim Para

Equalização

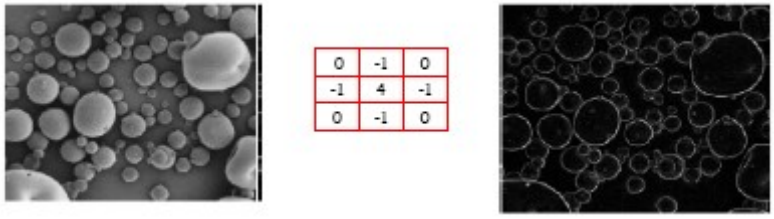
A equalização procura colocar um número igual de pixels para cada nível de cinza. Um aumento de contraste só seria ideal, se todos os possíveis 256 níveis fossem igualmente utilizados. Dessa maneira, todas as barras verticais que compõem o histograma seriam da mesma altura, o que não é possível devido à natureza discreta dos dados digitais de uma imagem. Uma aproximação pode ser conseguida se os picos do histograma da imagem forem espalhados, mantendo-se intactas suas partes mais achatadas.

$lenres = H.Length / (I.Width \times I.Height)$
Definir $LUT(256)$
$LUT(0) = H(0) * lenres$
$prev = H(0)$
Para "cada elemento do LUT + 1" **Fazer**
$prev = prev + H(i)$
$LUT(i) = prev \times lenres$
Fim Para
Para "cada pixel da imagem" **Fazer**
$I(x, y) = \min(LUT(I(x, y)), 255)$
Fim Para

2.2.3. Convolução

Os filtros são no geral usados para suprimir as altas-frequências na imagem (filtros passa baixa que suavizam a imagem), ou suprimir as baixas frequências (filtros passa alto que definem as arestas da imagem).

Os filtros são normalmente aplicados através da operação matemática de convolução. Esta operação permite calcular a intensidade de saída de um pixel em função, não só da intensidade do pixel de entrada correspondente, como também da intensidade dos pixels vizinhos. A operação equivale a uma média ponderada dos pixels da vizinhança. Cada vizinho tem um peso associado que é multiplicado pela sua intensidade. Os pesos são definidos por uma matriz denominada kernel (*ver Figura 7*).



0	-1	0
-1	4	-1
0	-1	0

Imagem

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}

Kernel

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}
K_{31}	K_{32}	K_{33}

$$O_{22} = I_{11}K_{11} + I_{12}K_{12} + I_{13}K_{13} + I_{21}K_{21} + I_{22}K_{22} + I_{23}K_{23} + I_{31}K_{31} + I_{32}K_{32} + I_{33}K_{33}$$

$$O_{23} = I_{12}K_{12} + I_{13}K_{13} + I_{14}K_{14} + I_{22}K_{22} + I_{23}K_{23} + I_{24}K_{24} + I_{32}K_{32} + I_{33}K_{33} + I_{34}K_{34}$$

. . .

Figura 7 – Filtrar imagens por convolução

Filtro Passa-Baixo

O filtro passa baixo (*ver Figura 8*) filtra as variações bruscas de cor da imagem, ou seja, vai atenuar os contornos da imagem. Este filtro atenua os vários picos de cor da imagem tendendo a uniformizar o seu espectro.

No filtro média cada pixel na imagem filtrada é obtido fazendo a média dos pixels que estão na vizinhança desse mesmo pixel na imagem original. Por isso quanto maior for a vizinhança desse a calcular a média, mais uniforme vai ficar a imagem filtrada.

O filtro gaussiano tem uma resposta em frequência também ela uma função gaussiana. Desta forma a sua resposta em frequência é semelhante à resposta em frequência de um filtro passa baixo, no entanto apresenta uma variação mais suave, o que se traduz numa maior uniformização da imagem e com isso a perda dos contornos.

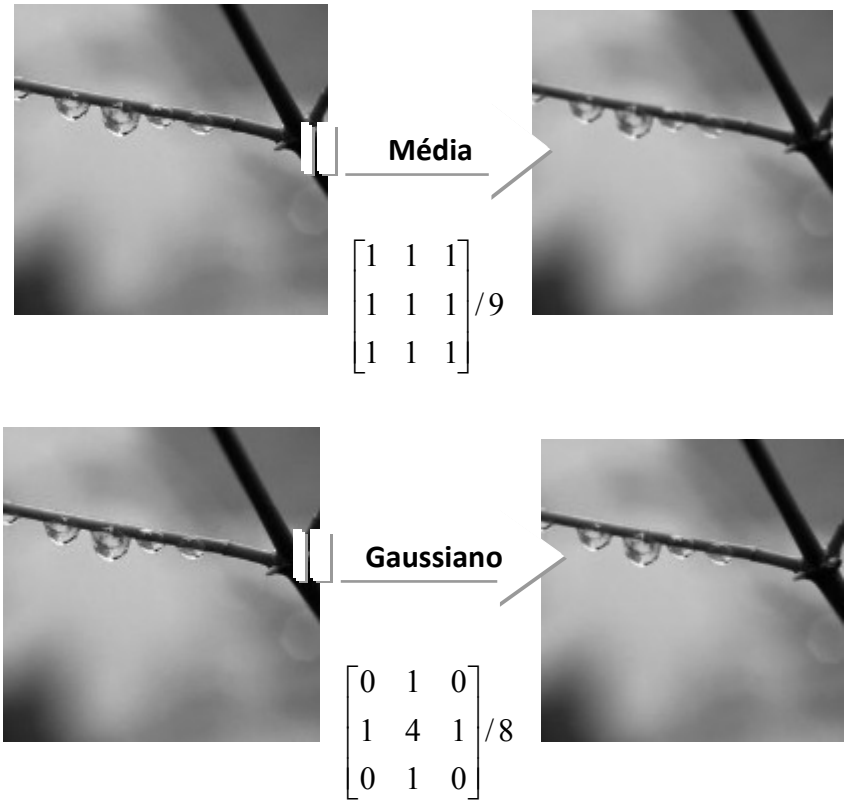


Figura 8 - Filtro passa baixo

Filtro Passa-Alto

Este filtro ao contrário do passa baixo apenas deixa passar as variações bruscas de cor da imagem uniformizando as zonas da imagem onde a cor é constante (*ver Figura 9*).

O filtro laplaciano detecta transições de cor, por isso propício para detectar contornos na imagem, enquanto o filtro unsharp acentua as variações bruscas de cor da imagem, isto é, dá um maior realce aos contornos da imagem.

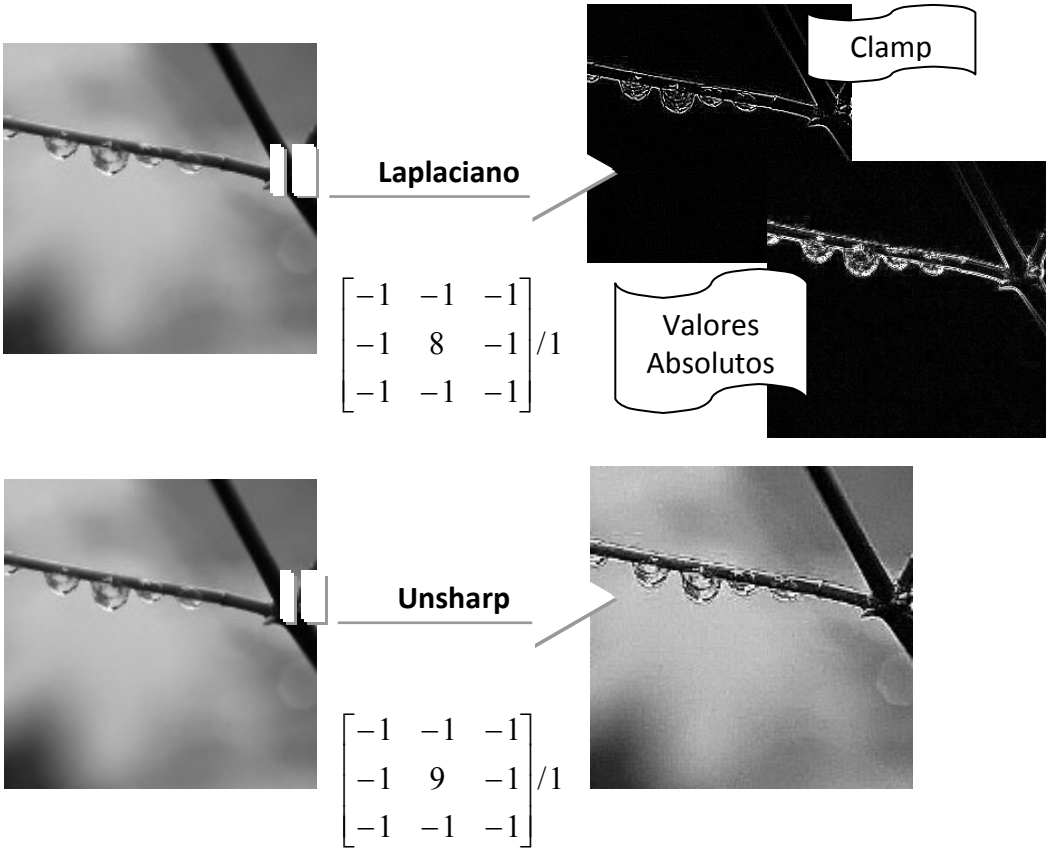


Figura 9 - Filtro passa alto

Filtro Passa-Banda

Este filtro tem um comportamento intermédio entre um filtro passa baixo e um passa alto em que, quer as variações bruscas de cor da imagem quer as zonas de cor mais uniforme são filtradas.

Filtrar uma imagem por convolução

Para "cada pixel da nova imagem" **Fazer**
Definir: $valor = 0$
Para "cada pixel existente no kernel da imagem velha" **Fazer**
$$valor = valor + S(x + xk, y + yk) \times K(xk, yk)$$
Fim Para
$valor = valor / div$
$D(x, y) = clamp(valor)$ ou $D(x, y) = abs(valor)$
Fim Para

2.2.4. Outras

Outras operações aplicadas a imagem como a redução do ruído e inversão da imagem.

Mediana

Para cada vizinhança, ordena os pixels em ordem crescente de intensidade e escolhe como saída o valor mediano -aquele que está no centro da sequência (*ver Figura 10*). Excelente eliminador de ruído localizado com intensidade muito diferente da vizinhança (*ver Figura 11*).

Figura 10 - Mediana

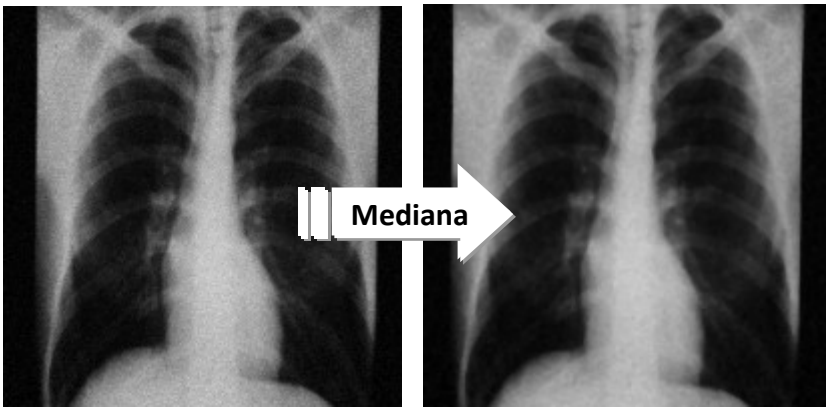


Figura 11 – Redução de ruído com mediana

Para "cada pixel da nova imagem" **Fazer**
Definir: $Lista$
Para "cada pixel existente no mediansize da imagem velha" **Fazer**
"adicionar o valor a lista"
Fim Para
"ordenar a lista"
$D(x,y) = Lista(Lista.Length/2)$
Fim Para

Inversão

Para "cada pixel da imagem" **Fazer**
$I(x,y) = 255 - I(x,y)$
Fim Para

3. Implementação

A implementação está dividida em dois componentes, a biblioteca que contem os tipos de imagens e respectivas operações e a aplicação final que usa como referencia a biblioteca.

Esta implementação foi escrita em C# para o .NET Framework 2.0.

3.1. Core

A biblioteca é a parte fundamental deste software pois contem os seus objectos (*ver Figura 12*) a sua funcionalidade independente da aplicação final que o ira usar, ou seja, qualquer software pode fazer referência a esta biblioteca para aplicar usar e aplicar determinadas operações em imagens. Nela existem diversas classes.

A classe “Image” é uma classe abstracta que só pode ser usada como base para diversos tipos ou formatos de imagem. Esta classe também contem os métodos necessários para podermos configurar a palete, manipular os seus pixels, obter o histograma e mostra-la na aplicação. As classes “IAPImage”, “BmpImage” e “ComplexImage” representam os respectivos formatos de imagem, sejam eles comprimidos ou no domínio das frequências.

A classe “ImageOperations” contem os métodos necessários para podermos aplicar operações e filtros na imagem.

Esta biblioteca contem a implementação da parte teórica falada até aqui neste documento sobre processamento e análise de imagem.

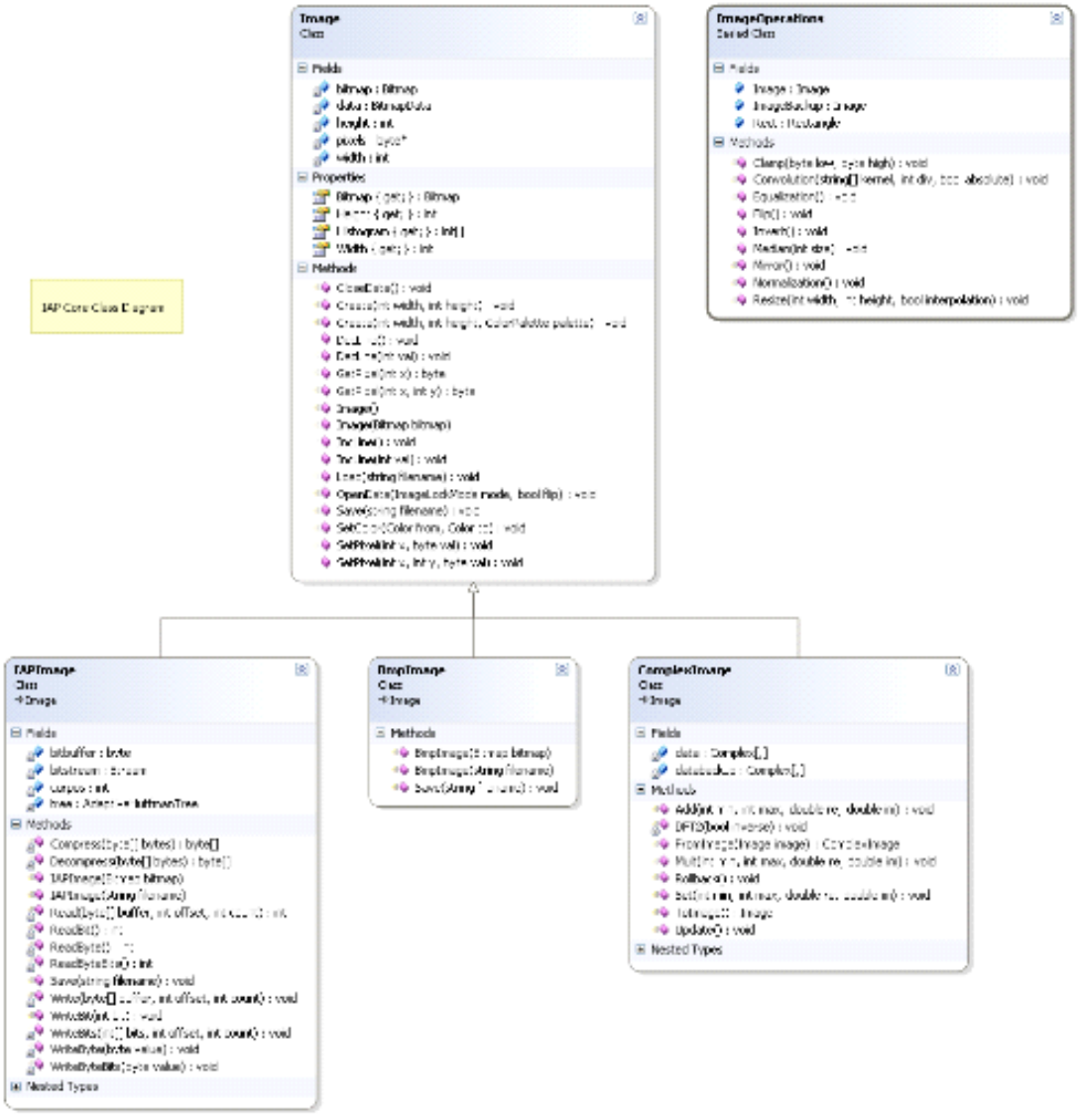


Figura 12 – IAP Core diagrama de classes

Exemplos

```
// Abrir uma imagem
BmpImage imagem = new BmpImage("exemplo.bmp");

// Configurar paleta
imagem.SetColor(Color.Black, Color.White);

// Manipular os pixels
imagem.OpenData(ImageLockMode.ReadWrite, false);

imagem.IncLine(0);

for (int y = 0; y < imagem.Height; y++)
{
    for (int x = 0; x < imagem.Width; x++)
        imagem.SetPixel(x, imagem.GetPixel(x));

    imagem.IncLine();
}

imagem.CloseData();

// Aplicar uma operação
ImageOperations.Image = imagem;
ImageOperations.Rect = new Rectangle(0, 0, imagem.Width, imagem.Height);

ImageOperations.Normalization();
ImageOperations.Convolution(new string[]{"1 1 1",
                                         "1 1 1",
                                         "1 1 1"}, 9, false);

// Converter para o dominio das frequencias
ComplexImage cimagem = ComplexImage.FromImage(imagem);

// Fitrar no dominio das frequencias
cimagem.Add(0, 16, 0, 0);

// Converter para o dominio normal
imagem = cimagem.ToImage();
```

3.2. Aplicação

A aplicação (*ver Figura 14 e Figura 15*) é então a interface que ira permitir o utilizador de processar e analisar imagens usando a biblioteca. Está dividida em duas partes, uma corresponde a imagem e a outra as operações que lhe poderemos aplicar. É simples e intuitivo.

O menu "File" (*ver Figura 13*) permite abrir/guardar o formato IAP como importar/exportar o formato BMP, entre outras opções.

Figura 13 – Menu "File"

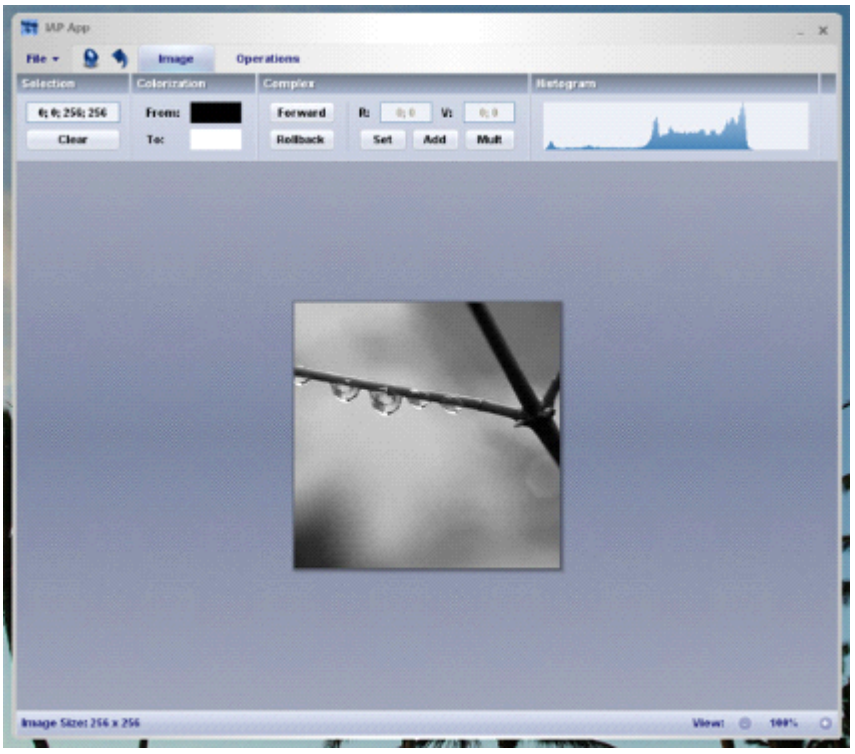


Figura 14 – IAP App Screenshot

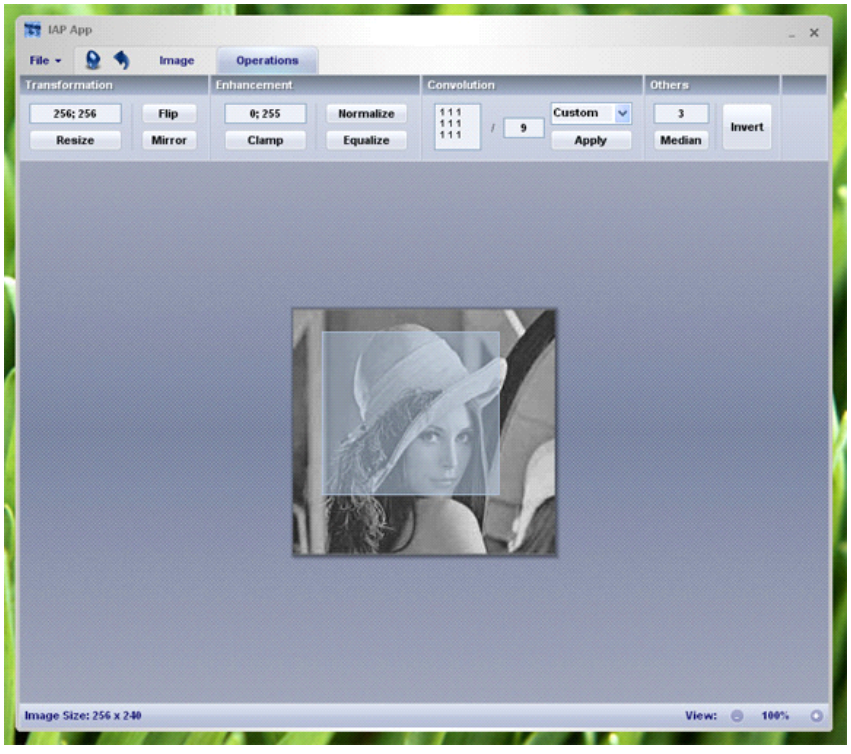


Figura 15 – IAP App Screenshot2

4. Conclusão

Estes tipos de software são das coisas mais complexas de implementar, pois requerem bons conhecimentos de matemática, algoritmia e bons conhecimentos de programação e de optimização. Muitos temas nesta área ainda estão em desenvolvimento, não só para acelerar os processos existentes como também para encontrar novas metodologias e aplicações.

Esta é uma área da engenharia que ainda tem muitos frutos para dar. É em alguns casos uma área em investigação e desenvolvimento. Uma área que também tem evoluído com a evolução tecnológica.

Existem inúmeras aplicações nesta área ao qual tem vindo a satisfazer/ajudar o homem nas suas necessidades/problemas.

Bibliografia

- Aulas do professor;
- Wolfram Research, http://www.wolfram.com/;
- Wikipedia, http://en.wikipedia.org/wiki/Main_Page;
- Internet em geral.

Anexos

No CD anexado está a respectiva implementação e seu código fonte.