Compressão JPG

Ao contrário da compressão de texto ou de programas, quando não é aceitável a perda de qualquer informação, no caso de imagens (e sobretudo no caso de vídeo) é perfeitamente aceitável algum nível de degradação da imagem. Isso ocorre devido a dois fatores:

⋆ Nossos olhos (e ouvidos) não são capazes de distinguir mudanças sutis e principalmente

⋆ Imagens (e vídeos) são objetos muito grandes que praticamente exigem serem comprimidos para um manuseio adequado.

As etapas pelas quais uma imagem passa antes de ser guardada em memória como um objeto JPG são as seguintes:

1. A imagem é dividida em blocos de 8×8 pixels. A razão é diminuir a quantidade de operações aritméticas necessárias durante a compressão (ainda que pagando para isso ter uma compressão ligeiramente ineficiente).

2. Cada bloco de imagem sofre a ação da transformada discreta do cosseno. Este procedimento (reversível) gera outra matriz 8×8, mas com uma propriedade interessante. Apenas os valores mais à esquerda e acima é que têm valores significativos. Embaixo e à direita, o valor mais comum é zero.

3. Os dados são quantizados visando diminuir o comprimento dos valores e aumentar a redundância. Aqui ocorre a degradação na compressão, já que este processo é irreversível.

4. Na compressão, os dados quantizados são percorridos em zig-zag. A seguir processos de compressão, tais como run-length, Huffman, QM (esta última protegida por patente) são aplicados gerando o arquivo final.

É de se notar que a maioria das câmeras digitais de fotografia (ou como dizem os franceses: *photo numérique*) já fazem todo este processamento, a fim de economizar memória.

Ao longo do processo, inúmeras otimizações são feitas para torná-lo mais eficiente. A primeira já citada é trabalhar com blocos de 64 valores ao invés de trabalhar com a imagem inteira. Uma segunda otimização ocorre quando os cossenos (uma função transcedental) são previamente calculados e armazenados, sendo depois apenas consultados. Uma terceira otimização ocorre quando o processo é manuseado algebricamente a fim de que as iterações sejam convertidas em um produto matricial de matrizes. Finalmente, usa-se aritmética de ponto fixo (fazendo os ajustes necessários) já que esta sempre é muito mais eficiente que a aritmética de ponto flutuante.

Transformada discreta do cosseno

Baseada no conceito de localidade de referência espacial (probabilidade alta de dois pixeis vizinhos terem valores próximos ou iguais) esta transformação atua muito eficientemente na preparação de dados de fotografias para compressão. A fórmula da transformada bidimensional é

$$G_{ij} = \frac{1}{\sqrt{2n}} C_i C_j \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} P_{xy} cos\left(\frac{(2y+1)j\pi}{2n}\right) cos\left(\frac{(2x+1)i\pi}{2n}\right)$$

Onde n é igual a 8, $C_f = 1$ quando $f > 0$ e $C_f = \frac{1}{\sqrt{2}}$ quando $f = 0$. G_{ij} é o valor na matriz transformada, P_{xy} é o valor do pixel correspondente na imagem, e i, j, x e m variam entre 0 e 7.

A transformada inversa discreta do cosseno bidirecional (conhecida como $IDCT$) apresenta a seguinte formulação:

$$P_{xy} = \frac{1}{4} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C_i C_j G_{ij} cos\left(\frac{(2x+1)i\pi}{2n}\right) cos\left(\frac{(2y+1)j\pi}{2n}\right)$$

Onde n é igual a 8, $C_f = 1$ quando $f > 0$ e $C_f = \frac{1}{\sqrt{2}}$ quando $f = 0$. P_{xy} é o valor do pixel correspondente na imagem, G_{ij} é o valor na matriz transformada, e i, j, x e m variam entre 0 e 7.

A fórmula da transformada direta pode ser facilmente traduzida a um programa em qualquer linguagem (neste caso em Python)

```
import math
import numpy as np
def c(x):
  if x==0:
    return 1/math.sqrt(2.0)
  else:
    return 1.0
def dct(ima):
  r=np.zeros((8,8),float)
  i=0
  while (i<8):
    j=0
    while(j<8):
      s=0.0
      x=0
      while x<8:
        y=0
        while y<8:
          t=ima[x][y]
          s=s+t*(math.cos((1+2.0*y)*j*math.pi/16.0))*
             (math.cos((1+2.0*x)*i*math.pi/16.0))
          y=y+1
        x=x+1
      t=0.25*(c(i))*(c(j))*s
      r[i][j]=t
      j=j+1
    i=i+1
  return r
im=np.array([[1, 19, 37, 55, 73, 91, 109, 127],
[19, 37, 55, 73, 91, 109, 127, 145],
[37, 55, 73, 91, 109, 127, 145, 163],
[55, 73, 91, 109, 127, 145, 163, 181],
[73, 91, 109, 127, 145, 163, 181, 199],
[91, 109, 127, 145, 163, 181, 199, 217],
[109, 127, 145, 163, 181, 199, 217, 235],
[127, 145, 163, 181, 199, 217, 235, 253]],float)

for linha in dct(im):
    for val in linha:
        print ('%7.1f '% val,end='')
    print ()
```

Exemplo

Suponha um bloco de imagem com os seguintes valores

```
  1   19   37   55   73   91  109  127
 19   37   55   73   91  109  127  145
 37   55   73   91  109  127  145  163
 55   73   91  109  127  145  163  181
 73   91  109  127  145  163  181  199
 91  109  127  145  163  181  199  217
109  127  145  163  181  199  217  235
127  145  163  181  199  217  235  253
```

A transformada discreta do cosseno para este bloco é

```
1016.0 -328.0  0.0 -34.3  0.0 -10.2  0.0 -2.6
-328.0    0.0  0.0   0.0  0.0   0.0  0.0  0.0
   0.0    0.0  0.0   0.0  0.0   0.0  0.0  0.0
 -34.3    0.0  0.0   0.0  0.0   0.0  0.0  0.0
   0.0    0.0  0.0   0.0  0.0   0.0  0.0  0.0
 -10.2    0.0  0.0   0.0  0.0   0.0  0.0  0.0
   0.0    0.0  0.0   0.0  0.0   0.0  0.0  0.0
  -2.6    0.0  0.0   0.0  0.0   0.0  0.0  0.0
```

☞ Para você fazer

Suponha um bloco de 8×8 com os seguintes valores de cinza.

```
 34   46   46   46   46   22   46   58
 34   58   46   58   58   22   34   34
 46   22   46   22   22   34   22   22
 46   58   46   22   22   34   34   46
 34   22   46   22   46   58   46   46
 22   34   46   22   58   46   46   34
 46   46   58   58   46   22   34   34
 22   58   58   34   34   46   34   22
```

Aplique a transformada discreta do cosseno a este bloco e informe:

1.Valor da transformada em [0,0]	Valor da transformada em [0,2]