TP2
Compression JPEG

Romain Raveaux

Objectif

Sensibiliser les étudiants aux algorithmes de compression/décompression d'images utiliser dans les systèmes multimédia.
Coût algorithmique.
Qualité d'image

Compression par DCT : Discrete Cosine Transform

Dans un premier nous allons travailler sur une matrice 8x8. On utilisera la variable globale step=8 pour stocker cette valeur de taille de block. Nous allons considérer f comme notre image de départ.

$$f = \begin{bmatrix} 139 & 144 & 149 & 153 & 155 & 155 & 155 & 155 \\ 144 & 151 & 153 & 156 & 159 & 156 & 156 & 156 \\ 150 & 155 & 160 & 163 & 158 & 156 & 156 & 156 \\ 159 & 161 & 162 & 160 & 160 & 159 & 159 & 159 \\ 159 & 160 & 161 & 162 & 162 & 155 & 155 & 155 \\ 161 & 161 & 161 & 161 & 160 & 157 & 157 & 157 \\ 162 & 162 & 161 & 163 & 162 & 157 & 157 & 157 \\ 162 & 162 & 161 & 161 & 163 & 158 & 158 & 158 \end{bmatrix}$$ (cf wikipédia JPEG)

Voici le code python pour créer cette image :

```
def makedummyimage():
   im = numpy.zeros((8,8))
   im[0,0]=139
   im[0,1]=144
   im[0,2]=149
   im[0,3]=153
   im[0,4]=155
   im[0,5]=155
   im[0,6]=155
   im[0,7]=155
   im[1,0]=144
   im[1,1]=151
   im[1,2]=153
   im[1,3]=156
   im[1,4]=159
   im[1,5]=156
   im[1,6]=156
   im[1,7]=156
```

```
    im[2,0]=150
    im[2,1]=155
    im[2,2]=160
    im[2,3]=163
    im[2,4]=158
    im[2,5]=156
    im[2,6]=156
    im[2,7]=156
    im[3,0]=159
    im[3,1]=161
    im[3,2]=162
    im[3,3]=160
    im[3,4]=160
    im[3,5]=159
    im[3,6]=159
    im[3,7]=159
    im[4,0]=159
    im[4,1]=160
    im[4,2]=161
    im[4,3]=162
    im[4,4]=162
    im[4,5]=155
    im[4,6]=155
    im[4,7]=155
    im[5,0]=161
    im[5,1]=161
    im[5,2]=161
    im[5,3]=161
    im[5,4]=160
    im[5,5]=157
    im[5,6]=157
    im[5,7]=157
    im[6,0]=162
    im[6,1]=162
    im[6,2]=161
    im[6,3]=163
    im[6,4]=162
    im[6,5]=157
    im[6,6]=157
    im[6,7]=157
    im[7,0]=162
    im[7,1]=162
    im[7,2]=161
    im[7,3]=161
    im[7,4]=163
    im[7,5]=158
    im[7,6]=158
    im[7,7]=158
    return numpy.uint8(im)
```

1. Créer une fonction DCT prenant en entrée cette matrice et retourne une matrice avec les coefficients de la DCT.

La fonction DCT pour une cellule de la matrice vaut :

$$\mathrm{DCT}(i,j) = \frac{2}{N} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathrm{pixel}(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{pour } x = 0 \\ 1 & \text{pour } x > 0 \end{cases}$$

Vous devez obtenir les valeurs suivantes (aux arrondis près):

$$F = \begin{bmatrix} 1260 & -1 & -12 & -5 & 2 & -2 & -3 & 1 \\ -23 & -17 & -6 & -3 & -3 & 0 & 0 & -1 \\ -11 & -9 & -2 & 2 & 0 & -1 & -1 & 0 \\ -7 & -2 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 2 & 0 & -1 & 1 & 1 \\ 2 & 0 & 2 & 0 & -1 & 1 & 1 & -1 \\ -1 & 0 & 0 & -1 & 0 & 2 & 1 & -1 \\ -3 & 2 & -4 & -2 & 2 & 1 & -1 & 0 \end{bmatrix}$$

2. Inversion de la DCT
   Créer une fonction (inverseDCT) faisant la DCT inverse prenant en entrées une matrice de coefficients de DCT et retournant une matrice contenant les pixels de l'image.

La DCT inverse se calcule ainsi :

$$\mathrm{pixel}(x,y) = \frac{2}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j)\,\mathrm{DCT}(i,j) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

Vous devez obtenir la matrice f de départ aux arrondis près.

3. Créer une fonction de quantification prenant en entrées une matrice de DCT et une matrice de quantification et retournant une matrice quantifiée.

$$F^*(u,v) = \left\lfloor \frac{F(u,v) + \left\lfloor \frac{Q(u,v)}{2} \right\rfloor}{Q(u,v)} \right\rfloor \cong \text{entier le plus proche} \left(\frac{F(u,v)}{Q(u,v)}\right)$$

Avec : $\lfloor x \rfloor$ entier directement inférieur à x

4. Créer une fonction de déquantification prenant en entrées une matrice de DCT et une matrice de quantification et retournant une matrice déquantifiée.

$$\hat{F}(u,v) = F^*(u,v) \cdot Q(u,v)$$

Vous essayerez les matrices de quantifications suivantes :

Q1=numpy.ones((step, step))

Q2= makeQuantificationMatrix(40)

Avec

```
def makeQuantificationMatrix(level):
  res=numpy.zeros((step, step))
  for i in range(0,step):
    #print(i)
    for j in range(0,step):
```

```
        res[i,j]=(i*j)+level
    return res
```

Compression et Décompression sur une image

A partir de l'image « lena.std.tif » qui se trouve dans votre répertoire « ~/pythonwork » implémenter une compression et décompression de type DCT par bloc 8x8.
Nous travaillerons sur l'image en niveaux de gris plutôt que l'image couleur.
Nous réduirons un peu la taille de l'image de départ avec la commande suivante :
gray = cv2.resize(gray, (64,64))

Les étapes à suivre son les suivantes :
1°) Décomposer l'image en blocks afin d'obtenir une liste de blocks. Faire une fonction prenant en entrée une image et retournant une liste de blocks 8x8
2°) Appliquer la DCT sur chaque block
3°) Afficher dans la console le numéro du block traité
4°) Quantifier Chaque block
5°) DéQuantifier chaque block
6°) Appliquer la DCT inverse sur chaque block
7°) Afficher dans la console le numéro du block traité
8°) Afficher l'image décompressée.

Compression et Décompression rapide sur une image

Vos fonctions DCT et inverseDCT sont bien implémentées mais relativement lentes.
Remplacer les appels à votre fonction DCT par un appel à la fonction DCT d'OpenCV

```
#dct rapide
imgf=numpy.float32(imgblock) #caste de l'image en float
dctblock=cv2.dct(imgf)

#dct inverse rapide
img=cv2.idct(dctblock)
```

5. Faites varier la taille des blocks et conclure sur l'impact de ce paramètre. Illustrer vos résultats. Quantifier vos résultats.
   Pour obtenir des résultats quantitatifs il est possible de faire la différence pixel à pixel entre l'image originale et l'image décompressée et de sommer le carré de ces différences.
   $$\text{ISE}=\left\| I_{x,y}^{originale} - I_{x,y}^{decompréssée} \right\|$$
6. Faites varier le niveau de quantification de 1 à 100 (paramètre level) et conclure sur l'impact de ce paramètre.