AIV1 - TP03
Compression d'image

Suliac Lavenant et Antoine Nollet

28 January 2022

Table des matières

1 Transformée en cosinus discrète

1.1 Transformée 2D directe

```
    def dct2d(image):
        """ Compute the 2D Discrete Cosine Transform of input <image> (numpy array)
        """
        dct1d = np.zeros(image.shape, dtype=np.uint8)
        dct1d = dct(image, type=2, axis=0, norm="ortho")

        dct2d = np.zeros(image.shape, dtype=np.uint8)
        dct2d = dct(dct1d, type=2, axis=1, norm="ortho")

        return dct2d
```

Afin d'effectuer une tranformée en cosinus discrète 2D de l'image de base, nous avons effectué deux tranformées en cosinus discrète 1D à la suite sur l'image de base. Pour cela, nous utilisons la fonction dct. Le paramètre type correspond au calcul effectué, ici il s'agit d'une transformée et non de sa transformée inverse (qui serait possible avec type=3). Le paramètre axis correspond à l'axe sur lequel on effectue la transformée 1D, et ici nous avons d'abord effectuée une transformée sur les lignes, puis sur les colonnes. Enfin, le paramètre norm correspond à "ortho", alors nous multiplirons le résultat de dct par un certain facteur f. Notre objectif est d'appliquer la formule du cours qui est celle-ci :

$$F(u) = \sqrt{\frac{2}{M}} c(u) \sum_{m=0}^{M-1} f(m) \cos(\pi \frac{(2m+1)u}{2M})$$

Avec $c(u)$ défini comme ceci :

$$c(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u \neq 0 \end{cases}$$

Avec le paramètre type à 2 et le paramètre norm à "ortho", d'après la documentation de la fonction scipy.fftpack.dct, voici la formule utilisée :

$$y(k) = 2f(k) \sum_{n=0}^{N-1} x(n) \cos(\frac{\pi(2n+1)k}{2N})$$

avec $f(k)$ défini comme ceci :

$$f(k) = \begin{cases} \sqrt{\frac{1}{4N}} & \text{if } k = 0 \\ \sqrt{\frac{1}{2N}} & \text{if } k \neq 0 \end{cases}$$

En adaptant les noms de variables pour qu'elles correspondent au cours,
— y devient F
— k devient u
— N devient M
— n devient m
— x devient f_c (le f de la formule du cours)
Ainsi, on obtient :

$$F(u) = 2f(u) \sum_{m=0}^{M-1} f_c(m) \cos(\frac{\pi(2m+1)u}{2M})$$

Nous avons donc la formule du cours correspondant à la formule de la documentation, à condition que $2f(u)$ soit égal à $\sqrt{\frac{2}{M}}c(u)$. Prouvons cette égalité avec $u = 0$:

$$\sqrt{\frac{2}{M}}c(u) = 2f(u) \tag{1}$$
$$\sqrt{\frac{2}{M}}\frac{1}{\sqrt{2}} = 2\sqrt{\frac{1}{4M}} \tag{2}$$
$$\frac{\sqrt{2}}{\sqrt{M}}\frac{1}{\sqrt{2}} = 2\frac{\sqrt{1}}{\sqrt{4M}} \tag{3}$$
$$\frac{1}{\sqrt{M}} = 2\frac{1}{\sqrt{4}\sqrt{M}} \tag{4}$$
$$\frac{1}{\sqrt{M}} = 2\frac{1}{2\sqrt{M}} \tag{5}$$
$$\frac{1}{\sqrt{M}} = \frac{1}{\sqrt{M}} \tag{6}$$

Nous avons confirmé l'égalité avec $u = 0$, faisons de même pour $u \neq 0$:

$$\sqrt{\frac{2}{M}}c(u) = 2f(u) \tag{7}$$
$$\sqrt{\frac{2}{M}}1 = 2\sqrt{\frac{1}{2M}} \tag{8}$$
$$\frac{\sqrt{2}}{\sqrt{M}} = \frac{2}{\sqrt{2M}} \tag{9}$$
$$\frac{\sqrt{2}}{\sqrt{M}} = \frac{2}{\sqrt{2}\sqrt{M}} \tag{10}$$
$$\frac{\sqrt{2}}{\sqrt{M}} = \frac{\sqrt{2}}{\sqrt{2}\sqrt{M}\frac{\sqrt{2}}{2}} \tag{11}$$
$$\frac{\sqrt{2}}{\sqrt{M}} = \frac{\sqrt{2}}{\sqrt{M}} \tag{12}$$
$$\tag{13}$$

L'égalité est également confirmée avec $u \neq 0$, nous pouvons aisément utiliser la fonction dct avec le paramètre norm égal à "ortho", qui effectuera le même calcul que vu en cours.

1.2 Validation sur un exemple

```python
def jpeg(img_name, show=False, write=False):
    img = plt.imread(img_name) * 255
    img-=128 #centrage

    img_dct2d = dct2d(img)
    img_dct2d = np.array(img_dct2d, dtype=np.int)

    if show:
        plt.figure()
        plt.imshow(img_dct2d, cmap='gray')
        plt.show()

    if(write):
        plt.imsave("img_dct2d.png", img_dct2d, cmap='gray')

    return img_dct2d
```

Nous allons donc appliquer ce script sur l'image suivante :

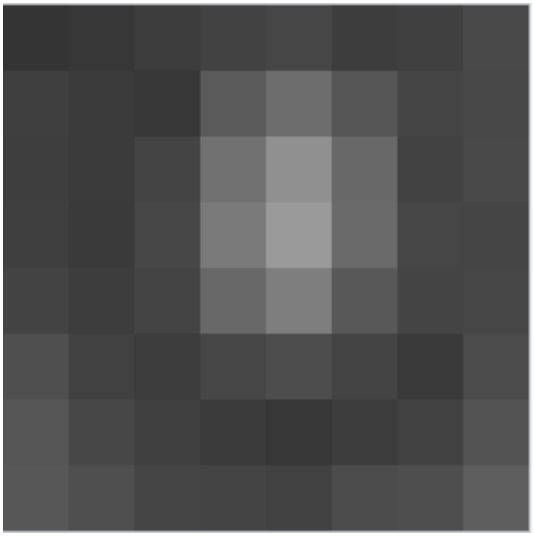

FIGURE 1 – exemple8x8$_capture$

valeur de l'image de base :

52	55	61	66	70	61	64	73
63	59	55	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

Valeur de l'image après centrage des valeurs (-128) :

-76	-73	-67	-62	-58	-67	-64	-55
-65	-69	-73	-38	-19	-43	-59	-56
-66	-69	-60	-15	16	-24	-62	-55
-65	-70	-57	-6	26	-22	-58	-59
-61	-67	-60	-24	-2	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-43	-57	-64	-69	-73	-67	-63	-45
-41	-49	-59	-60	-63	-52	-50	-34

Valeur de l'image après application de dct2d :

-415	-30	-61	27	56	-20	-2	0
4	-21	-60	10	13	-7	-8	4
-46	7	77	-24	-28	9	5	-5
-48	12	34	-14	-10	6	1	1
12	-6	-13	-3	-1	1	-2	3
-7	2	2	-5	-2	0	4	1
-1	0	0	-2	0	-3	4	0
0	0	-1	-4	-1	0	0	1

En comparant avec les valeurs désirées [référence] , on confirme notre calcul de dct2d.

2 Utilisation de la DCT pour la compression JPEG

2.1 Traitement par blocs d'une image

```python
def jpeg(img_name, show=False, write=False):
    img = plt.imread(img_name) * 255
    img-=128 #centrage

    img_dct2d = np.zeros(img.shape)

    for c in range(0, img.shape[1], BLOCK_SIZE):
        for l in range(0, img.shape[0], BLOCK_SIZE):
            tmp = dct2d(img[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c])
            img_dct2d[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c] = tmp


    img_dct2d = np.array(img_dct2d, dtype=np.int)

    if show:
        plt.figure()
        plt.imshow(img_dct2d, cmap='gray')
        plt.show()

    if(write):
        cv2.imwrite("img_dct2d.TIF", img_dct2d)

    return img_dct2d
```

En appliquant ce traitement a l'image lena8bit on obtient le resultat suivant :

FIGURE 2 – exemple8x8$_{c}apture$

Malgré que nous n'ayons que des coefficients DCT, nous reconnaissons quand même l'image de base de Léna. Cela est expliqué car la DCT nous conserve les basses fréquences de l'image.

Interpréter le résultat (image de la DCT) et commenter les coefficients DCT obtenus pour 2 ou 3 blocs caractéristiques de l'image

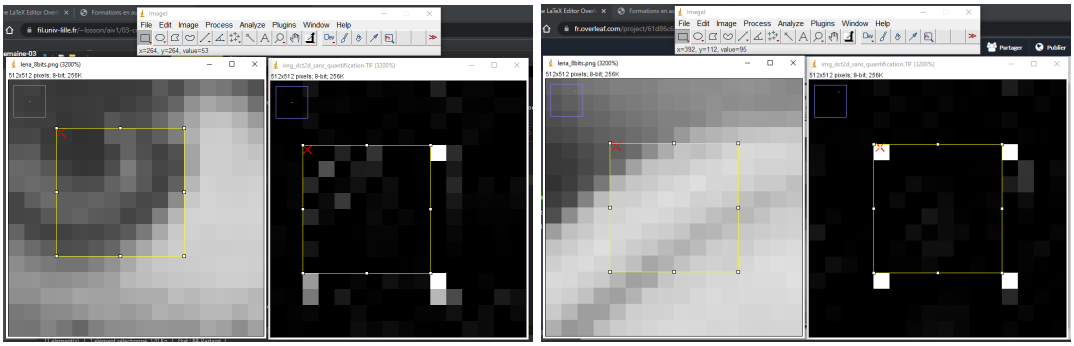

FIGURE 3 – Comparaisons de 2 blocs significatifs (oeil et chapeau) avec leurs coefficients DCT équivalents

Dans la première comparaison, celle de l'oeil, il y a beaucoup de hautes fréquences (beaucoup de changement de valeurs), ainsi, les coefficients élevés se trouveront dans le centre du bloc DCT. Contrairement à la deuxième comparaison, celle du chapeau, où là, il y a beaucoup de basses fréquences, ce qui justifiera la présence du haut coefficient en haut à gauche du bloc DCT (le carré blanc).

2.2 Quantification

Considérons la matrice QY suivante :

$$QY = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Et le code suivant :

```python
def jpeg(img_name, show=False, write=False):
    img = plt.imread(img_name) * 255
    img-=128 #centrage

    img_dct2d = np.zeros(img.shape)

    for c in range(0, img.shape[1], BLOCK_SIZE):
        for l in range(0, img.shape[0], BLOCK_SIZE):
            tmp = dct2d(img[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c])
            tmp = tmp/QY #appliquation matrice quantification
            img_dct2d[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c] = tmp

    img_dct2d = np.round(img_dct2d,0) #arrondi a l'entier

    if show:
        plt.figure()
        plt.imshow(img_dct2d, cmap='gray')
        plt.show()

    if(write):
        plt.imsave("img_dct2d.png", img_dct2d, cmap='gray')
        cv2.imwrite("img_dct2d.TIF", img_dct2d)

    return img_dct2d
```

En reprenant l'image exemple8x8 vu précédemment, on lui applique la quantification par la matrice QY et un arrondi à l'entier pour obtenir le résultat suivant :

-26	-3	-6	2	2	-1	0	0
0	-2	-4	1	1	0	0	0
-3	1	5	-1	-1	0	0	0
-3	1	2	-1	0	0	0	0
1	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

En comparant ces valeurs avec celle-ci (cliquer ici), on confirme notre quantification.

2.3 Décompression

```python
def idct2d(coefficients):
    """ Compute the 2D Inverse Discrete Cosine Transform of given <coefficients> (numpy array)
    """
    idct1d = np.zeros(coefficients.shape)
    idct1d = idct(coefficients, type=2, axis=0, norm="ortho")

    idct2d = np.zeros(coefficients.shape)
    idct2d = idct(idct1d, type=2, axis=1, norm="ortho")

    return idct2d

def jpeg(img_name, q, show=False, write=False):
    Q = getQ(q)
    img = plt.imread(img_name) * 255
    img-=128 #centrage

    img_dct2d = np.zeros(img.shape)

    for c in range(0, img.shape[1], BLOCK_SIZE):
        for l in range(0, img.shape[0], BLOCK_SIZE):
            tmp = dct2d(img[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c])
            tmp = tmp/Q #appliquation matrice quantification
            img_dct2d[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c] = tmp

    img_dct2d = np.round(img_dct2d,0) #arrondi a l'entier
    img_dct2d = np.array(img_dct2d, dtype=np.int)

    ####################decompression
    img = np.zeros(img_dct2d.shape)

    for c in range(0, img_dct2d.shape[1], BLOCK_SIZE):
        for l in range(0, img_dct2d.shape[0], BLOCK_SIZE):
            tmp = img_dct2d[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c]*Q #appliquation matrice quantification
            tmp = idct2d(tmp)
            img[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c] = tmp

    img+=128

    if show:
        plt.figure()
        plt.imshow(img_dct2d, cmap='gray')
        plt.figure()
        plt.imshow(img, cmap='gray')
        plt.show()

    if(write):
        plt.imsave("img_dct2d.png", img_dct2d, cmap='gray')
        cv2.imwrite("img_dct2d.TIF", img_dct2d)
        plt.imsave("img.png", img, cmap='gray')

    return img
```

On applique donc la compression à l'image exemple8x8 pour ensuite la décompresser. On obtient donc le résultat suivant :

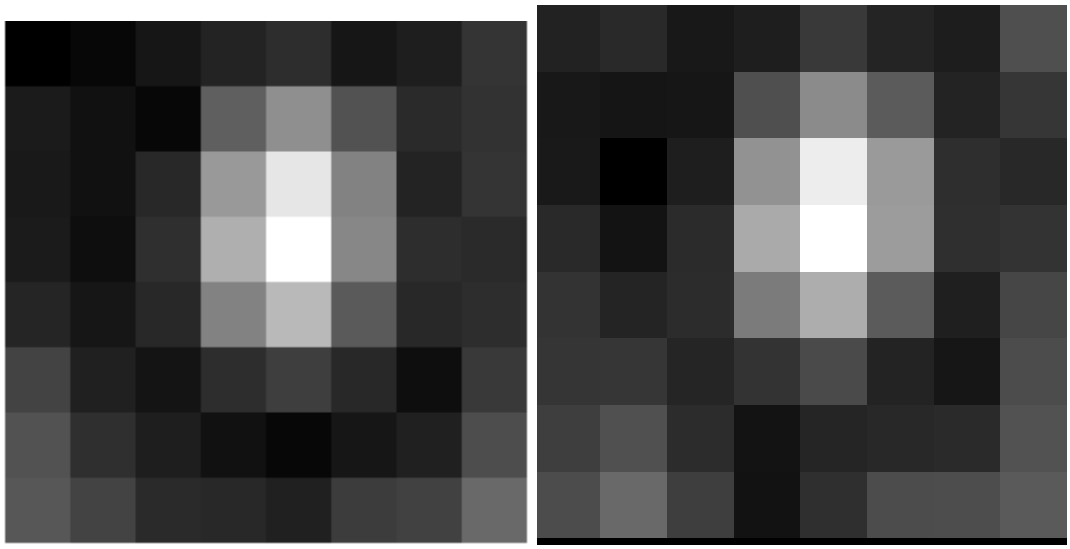

FIGURE 4 – exemple8x8 à gauche et son image compresser puis décompresser à droite
(Les images sont de dimentions differentes car se sont des captures des images. Les images de base sont floutées dans le pdf.)

Après observation, l'image resultante est très similaire à l'originale malgré quelque differences mineurs de niveaux de gris. On pourra notamment mesurer rigoureusement la similitude des images grâce à la prochaine partie.

3 Qualité et performance de la compression JPEG

3.1 Évaluation de la distorsion

FIGURE 5 – lena_8bits à gauche et son image compresser puis décompresser à droite

On peut constater visuellement que ces images sont identiques à quelque détails près (chevelure plus marquée, yeux plus marqués, un noir général plus intense,...)

Nous allons mesurer l'erreur absolue moyenne (MAE), l'erreur quadratique moyenne (MSE) et le rapport signal sur bruit pic-à-àpic (PNSR).

```python
def error_measures(I, Iest,show=False):
    """ Return the MAE, MSE, and PSNR error measures between the original image <I> and its estimate <Iest>
    """
    M = I.shape[0]
    N = I.shape[1]
    MAE = 1/(M*N) * sum([abs(I[m,n] - Iest[m,n]) for m in range(M) for n in range(N)])
    MSE = 1/(M*N) * sum([(I[m,n] - Iest[m,n])**2 for m in range(M) for n in range(N)])
    PSNR = 10*log((255**2)/MSE,10)

    if(show):
        print("MAE : " +str(MAE))
        print("MSE : " +str(MSE))
        print("PSNR : "+str(PSNR))
        print(PSNR)
    return MAE, MSE, PSNR
```

Après avoir appliquer la compression et la décompression à l'aide du code de la partie précédente, nous avons évalué la distorsion à l'aide de la fonction error_measures défini ci-dessus. Voici les résultats obtenues :

- MAE : 2.9475989536399254
- MSE : 16.567380800644184
- PSNR : 35.93826506265026

Avec un PSNR comme celui ci-dessus, on peut conclure que ces images sont presques identiques à l'oeil humain (proche de 40db).

3.2 Influence de la quantification

FIGURE 6 – compression pui decompression de lena avec un q=5

- MAE : 7.858465786262379
- MSE : 113.59381899105082
- PSNR : 27.57725660224519

Sur cette première image, avec un q égale à 5 on voit visuellement que l'image est bien dégradée. Cela est confirmé par le PSNR à 27.5, l'image est ressemblante mais différente.

FIGURE 7 – compression pui decompression de lena avec un q=40

— MAE : 3.150710113876217
— MSE : 19.255212649737544
— PSNR : 35.285320416433784

Puis sur cette deuxième image, avec un q de 40, l'image est visuellement plus propre mais on voit quand même des différences de contraste par rapport à l'originale. Cela est soutenu par le PSNR à 35.2.

FIGURE 8 – compression pui decompression de lena avec un q=75

— MAE : 2.408987372313124
— MSE : 10.475869752989938
— PSNR : 37.92890270686059

Pour finir, sur cette troisième image, avec un q de 75, l'image est visuellement beaucoup plus ressemblante à l'originale. Cela est soutenu par le PSNR élevé de 37.9.

Pour comparer ces trois images, on voit bien la différence entre la première de q=5 et les deux autres, cela est confirmé par les MAE et MSE beaucoup plus grande sur cette image et son PSNR beaucoup plus faible.

Les deux autres images de q=40 et q=75 sont quand à elles beaucoup plus proches que cela soit en MAE, MSE et PSNR (et même visuellement) et on comprend facilement le choix souvent fait d'une compression à q=40 au lieu d'une avec un q=75 du fait de la légère difference de qualité pour une bonne différence de taille.

3.3 Ratio de compression

```python
def jpeg(img_name, q, show=False, write=False):
    ...
    #zigzag
    img_zigzag = []
    for c in range(0, img.shape[1], BLOCK_SIZE):
        for l in range(0, img.shape[0], BLOCK_SIZE):
            img_zigzag += coefsBeforeEOB(img_dct2d[l:BLOCK_SIZE+l,c:BLOCK_SIZE+c])
    for i in range(len(img_zigzag)):
        img_zigzag[i] += 128
    ...
    if(write):
        plt.imsave("img_dct2d.png", img_dct2d, cmap='gray')
        cv2.imwrite("img_dct2d.TIF", img_dct2d)
        plt.imsave("img_decomp.png", img_decomp, cmap='gray')
        plt.imsave("img_zigzag.png", [img_zigzag], cmap='gray')

    return img_decomp

def zigzag(coefficients):
    """ Return the list of <coefficients> in zigzag read order
        Adapted from: http://rosettacode.org/wiki/Zig-zag_matrix#Python:_By_sorting_indices
    """
    def compare(xy):
        x, y = xy
        return x + y, -y if (x + y) % 2 else y
    nrows, ncols = coefficients.shape
    assert nrows == ncols
    ind = range(nrows)  # indices along x and y
    ordered_indices = sorted(((x, y) for x in ind for y in ind), key=compare)
    return [coefficients[index] for index in ordered_indices]


def coefsBeforeEOB(coefficients):
    """ Return the first non-zero values in zigzag read order of <coefficients>
    """
    zig = zigzag(coefficients)
    allCoefsBeforeEOB = []
    i = 0
    while zig[i]!=0:
        allCoefsBeforeEOB.append(zig[i])
        i+=1

    return allCoefsBeforeEOB
```

Le ratio de compression se calcul avec la formule suivante :

$$q = \frac{volume(I)}{volume(I_{compress})}$$

(à ne pas confondre avec le taux de compression $t = \frac{volume(I_{compress})}{volume(I)}$)

En usant du code cité précédemment, avec notre image de base d'une taille de 219.3kb, nous obtenons les résultats suivants :

facteur qualité q	taille fichier généré (en kb)	ratio de compression
5	2.4	91.375
40	21.3	10.296
75	38.3	5.726

Ces résultats rejoignent le précédent constat, plus le ratio de qualité sera élevé, plus le fichier compressé sera de taille importante et donc la ratio de compression sera moindre.