PROJET INFORMATIQUE N°2

Compression d'image JPEG

Sommaire

Bibliographie

Définition Wikipédia du JPEG : https://fr.wikipedia.org/wiki/JPEG

Exemple de code et explication pour la compression JPEG : https://compression.fiches-horaires.net/la-compression-avec-perte-1/le-compression-jpeg/

Sujet du devoir avec explication et les équations : https://drive.google.com/drive/folders/1RAkPGSM4HIEGRwyu4ugy9dA1McIHH5t-

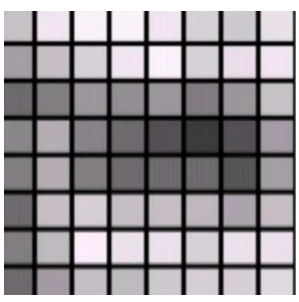
Présentation du sujet et de l'image d'exemple

On cherche à faire une compression sur une image monochrome de 8 x 8 pixels en JPEG. Afin de partir d'un tableau de valeur et finir par une ligne plus courte de valeur plus faible que celle d'origine pour obtenir une compression qui n'occupe presque pas de place de stockage.

Après la compression JPEG, on obtient une image avec une perte de qualité pratiquement invisible à l'œil nu.

Pour ce projet, on va utiliser l'exemple suivant :

188	225	237	237	227	205	199	222
157	186	205	225	237	295	218	225
104	125	149	143	151	123	149	191
132	169	128	104	81	61	81	152
125	169	125	104	104	107	87	158
107	185	201	187	185	179	161	186
125	180	233	223	223	205	223	223
104	152	186	180	196	203	203	212



DCT et DCT Inverse

Le programme de DCT est le suivant :

```
def encoder_dct(ancien_tableau: List[List[int]]) -> List[List[int]]:
    nouveau_tableau: List[List[int]] = deepcopy(ancien_tableau)
    # Copie complète des valeurs du tableau d'origine
    for nouveau_y in range(8):
        for nouveau_x in range(8):
    # Permet le balayage complet de toutes les valeurs du tableau
            nouvelle_valeur = 0
    # On insère une constante pour la boucle
            for ancien_y in range(8):
                for ancien_x in range(8):
                    nouvelle_valeur += (
                            ancien_tableau[ancien_y][ancien_x] *
                            cos(((2 * ancien_y + 1) * nouveau_y * pi) / 16) *
                            cos(((2 * ancien_x + 1) * nouveau_x * pi) / 16)
                    )
    # Utilisation de la formule de DCT
            if nouveau_y == 0:
                nouvelle_valeur /= sqrt(2)
            if nouveau_x == 0:
                nouvelle_valeur /= sqrt(2)
            nouvelle_valeur /= 4
            nouveau_tableau[nouveau_y][nouveau_x] = math.floor(nouvelle_valeur)
    return nouveau_tableau # On redonne les valeurs après la conversion DCT
```

La DCT (en anglais : Discrete Cosinus Transformation) est une fonction mathématique qui transforme notre tableau de 8 x 8 valeurs de notre image en tableau de 8 x 8 nouvelles valeurs de coefficients. C'est un passage du domaine spatial au domaine fréquentiel.

La DCT permet une réorganisation complète des valeurs.

La formule de DCT est :

$$F(u,v) = \frac{A(u) \times A(v)}{4} \times \sum_{i=0}^{7} \sum_{j=0}^{7} \cos\left(\frac{(2 \times i + 1) \times u \times \pi}{16}\right) \times \cos\left(\frac{(2 \times j + 1) \times v \times \pi}{16}\right) \times f(i,j)$$

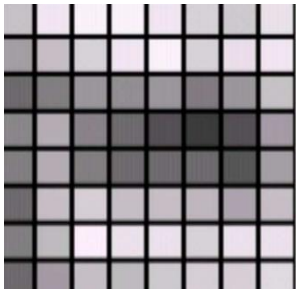
Avec

$$A(\alpha) = \begin{cases} \frac{1}{\sqrt{2}} \; si \; \alpha = 0 \\ 1 \; sinon \end{cases}$$

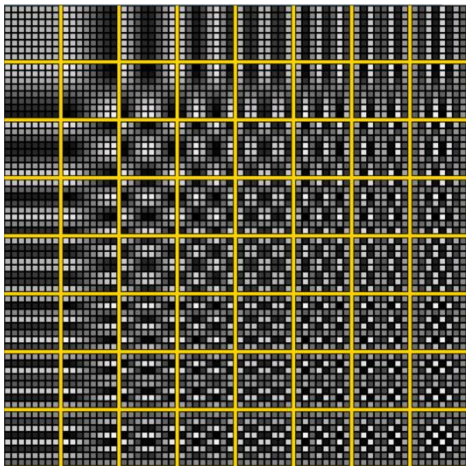
Cette partie de la fonction est pour réduire certaines valeurs pour avoir un résultat cohérent.

En utilisant une DCT, on fait une sorte de décomposition de série de Fourier, on va décomposer notre bloc de pixels en rayures de différentes tailles, des rayures les plus amples aux plus rapprochées. Chaque ensemble de stries aura sa propre intensité, en combinant toutes les stries, on obtient l'image d'origine.

Image d'origine



Décomposition



On a le tableau suivant pour les valeurs d'origine :

188	225	237	237	227	205	199	222
157	186	205	225	237	295	218	225
104	125	149	143	151	123	149	191
132	169	128	104	81	61	81	152
125	169	125	104	104	107	87	158
107	185	201	187	185	179	161	186
125	180	233	223	223	205	223	223
104	152	186	180	196	203	203	212

En temps normal, un pixel possède 4 valeurs : le rouge, le bleu, le vert et l'intensité. Ici, nous avons ignoré l'intensité et fait la moyenne de RGB pour les valeurs d'origine.

Après la conversion DCT par le programme python, on obtient le tableau de valeurs suivant :

1366	-66	-43	-106	0	-66	-11	9
37	39	11	12	41	-7	-14	12
270	-78	-94	21	-5	20	-4	28
102	36	-36	5	-8	-22	-2	-4
-89	57	50	9	9	-27	0	-11
11	24	0	-23	5	4	-5	13
-35	-12	-13	-2	-2	-9	6	3
-16	-18	3	6	6	-1	6	7

En vérifiant avec l'exemple de l'énoncé, les valeurs sont similaires donc la fonction python est correcte pour la conversion DCT.

On a maintenant la DCT Inverse, qui permet de retrouver à peu près les valeurs d'origine, les variations ne seront pas discernables à l'œil nu.

Dans ce programme on a appliqué la fonction suivante qui correspond à la DCT Inverse :

$$f(i,j) = \frac{1}{4} \times \sum_{u=0}^{7} \sum_{v=0}^{7} A(u) \times A(v) \times \cos\left(\frac{(2 \times i + 1) \times u \times \pi}{16}\right) \times \cos\left(\frac{(2 \times j + 1) \times v \times \pi}{16}\right) \times F(u,v)$$

Avec

$$A(\alpha) = \begin{cases} \frac{1}{\sqrt{2}} \; si \; \alpha = 0 \\ 1 \; sinon \end{cases}$$

Cette partie de la fonction est pour réduire certaine valeur pour avoir un résultat cohérent.

Le programme de DCT Inverse est le suivant :

```
def decoder_dct(ancien_tableau: List[List[int]]) -> List[List[int]]:
    nouveau_tableau: List[List[int]] = deepcopy(ancien_tableau)
    # On fait une copie de l'ancien tableau.
    for nouveau_y in range(8):
        for nouveau_x in range(8):
            nouvelle_valeur = 0
            for ancien_y in range(8):
                for ancien_x in range(8):
                    valeur_a_ajouter = (
                            ancien_tableau[ancien_y][ancien_x] *
                            cos(((2 * nouveau_y + 1) * ancien_y * pi) / 16) *
                            cos(((2 * nouveau_x + 1) * ancien_x * pi) / 16)
                    )
                    if ancien_y == 0:
                        valeur_a_ajouter /= sqrt(2)
                    if ancien_x == 0:
                        valeur_a_ajouter /= sqrt(2)
                    nouvelle_valeur += valeur_a_ajouter
            nouvelle_valeur /= 4
            nouveau_tableau[nouveau_y][nouveau_x] = math.floor(nouvelle_valeur)
    return nouveau_tableau # Même déroulement du programme sauf qu'on utilise une formule différente
```

Lorsqu'on applique le programme, on obtient ce tableau :

184	225	236	237	276	204	199	222
157	186	205	224	236	205	217	224
104	125	148	143	150	122	149	190
132	169	127	104	80	60	80	152
123	169	124	103	104	106	87	157
107	185	201	187	184	178	160	186
124	179	232	222	223	205	223	222
103	152	185	179	196	202	202	212

Les valeurs correspondent à ce qu'on veut donc le programme de DCT Inverse est correcte et fonctionne.

Fonction de Seuil et de Seuil Inverse

L'œil étant peu sensible aux fortes variations de couleurs, on peut légèrement lisser ce bloc en négligeant certaines valeurs dans la partie en bas à droite de la DCT obtenue. Donc, on applique une fonction de seuil à nos valeurs qui est l'équivalent d'un filtre passe-bas.

Dans ce programme, on applique un seuil de 4 comme dans l'exemple de l'énoncé. Dans l'écriture de ce programme, on réutilise la même forme de programme que la DCT et DCT Inverse.

Dans le programme de fonction de seuil, on doit appliquer un calcul sur toutes les valeurs du tableau.

Avec les valeurs de N (u et v), on doit calculer F : F = 1 + (u + v + 1) * s.

Puis avec les valeurs de F et de N, on fait une division entière à toutes les valeurs du tableau.

Le programme de fonction de seuil est le suivant :

```
def encoder_seuil(ancien_tableau: List[List[int]]) -> List[List[int]]:
    nouveau_tableau: List[List[int]] = deepcopy(ancien_tableau)
    nouveau_tableau2: List[List[int]] = deepcopy(ancien_tableau)
    s = 4 # On applique un seuil, ici 4 de l'exemple
    for nouveau2_y in range(8):
        for nouveau2_x in range(8):
            for nouveau_y in range(8):
                for nouveau_x in range(8):
                    nouvelle_valeur = 0
                    for ancien_y in range(8):
                        for ancien_x in range(8):
                            nouvelle_valeur += (
                                nouveau_tableau2[nouveau2_y][nouveau2_x] // (1 + (nouveau_y + nouveau_x + 1) * s)
                            ) # On applique le calcul : N // F avec F = 1 + (u + v + 1) * s
                    nouveau_tableau[nouveau_y][nouveau_x] = math.floor(nouvelle_valeur)
    return nouveau_tableau
```

Ce programme donne une réponse différente que sur l'exemple de l'énoncé mais la réponse est cohérente et logique pour la fonction de seuil.

On obtient donc le tableau :

64	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
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	0

On constate que de nombreux zéros sont apparus.

Maintenant, on peut appliquer la fonction de seuil Inverse.

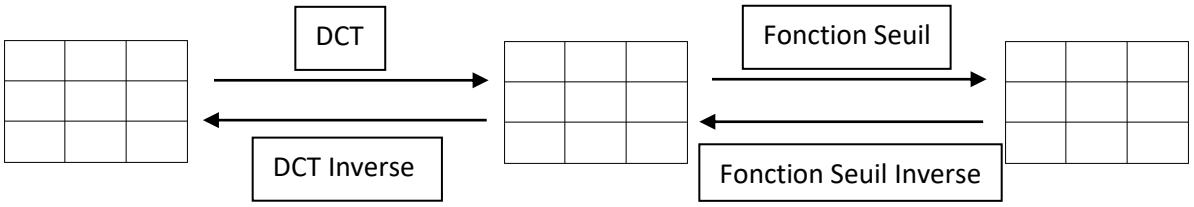
Le programme de fonction de seuil Inverse est le suivant :

```
def decoder_seuil(ancien_tableau: List[List[int]]) -> List[List[int]]:
    nouveau_tableau: List[List[int]] = deepcopy(ancien_tableau)
    nouveau_tableau2: List[List[int]] = deepcopy(ancien_tableau)
    s = 4
    for nouveau2_y in range(8):
        for nouveau2_x in range(8):
            for nouveau_y in range(8):
                for nouveau_x in range(8):
                    nouvelle_valeur = 0
                    for ancien_y in range(8):
                        for ancien_x in range(8):
                            valeur_a_ajouter = (
                                    nouveau_tableau2[nouveau2_y][nouveau2_x] * (1 + (nouveau_y + nouveau_x + 1) * s)
                            )
                            nouvelle_valeur += valeur_a_ajouter
                    nouveau_tableau[nouveau_y][nouveau_x] = math.floor(nouvelle_valeur)
    return nouveau_tableau
```

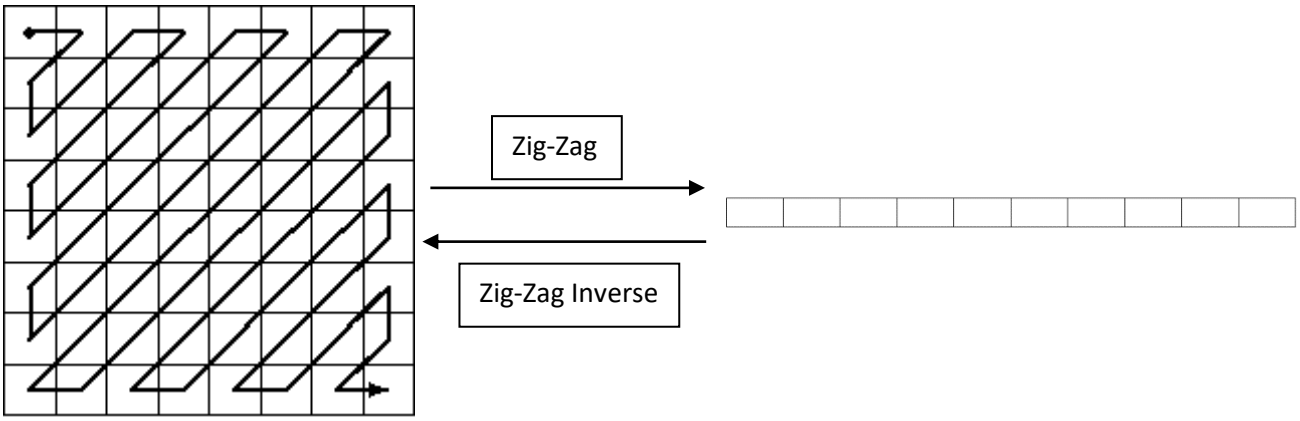
Avec ce programme, on obtient un tableau rempli de 0. Comme préciser dans l'énoncer, certaines valeurs passent à zéro et resteront à zéro lors de la fonction de seuil Inverse.

Fonction de Lecture en Zig-Zag

Dans les fonctions précédentes (DCT et seuil), on est resté dans le domaine de la 2D :



Maintenant, on veut passer de la 2D à la 1D grâce à la lecture en zig-zag :



On obtient les valeurs du tableau sur une seule ligne avec la lecture Zig-Zag.

Lors de l'exécution du programme de Zig-Zag, on doit obtenir une ligne commençant par 64, puis n'avoir que des 0 pour la suite :

64	0	0	…

Lors de l'exécution du programme de Zig-Zag Inverse, on doit obtenir un tableau telle que :

64	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
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	0

Fonction RLE et RLE Inverse

Le codage RLE consiste à remplacer une liste de nombres par une liste contenant le nombre d'occurrences du caractère puis ledit caractère.

Le programme de RLE est le suivant :

```
def encoder_RLE(input_list):
    # On définit les constantes pour évoluer dans le programme
    output = []
    precedent_value = None
    taille_sequence = 0
    # On démarre la boucle :
    for current_value in input_list:
        if current_value != precedent_value and precedent_value != None:
        # Si la valeur n'est pas égale à la valeur précédente et si la
        # précédente valeur n'est pas nul alors :
            # On fait apprendre à output la taille et la valeur.
            output.append(taille_sequence)
            output.append(precedent_value)
            # Définition de la taille de la séquence
            taille_sequence = 1
        else:
        # Sinon on ajoute 1 à la taille de la séquence à chaque fois.
            taille_sequence += 1
        precedent_value = current_value
    # Dans la boucle, on fait apprendre à output la taille
    # puis la valeur à chaque fois
    output.append(taille_sequence)
    output.append(precedent_value)
    return output # On redonne les valeurs
```

La réponse du programme python est :

1	64	63	0

La réponse est cohérente avec les valeurs obtenues avec les fonctions précédentes.

Maintenant, on a la fonction RLE Inverse qui permet de retrouver la liste d'origine de la fonction de lecture en Zig-Zag.

Le programme de RLE Inverse est le suivant :

```
def decoder_RLE(input):
    # On définit les constantes pour évoluer dans le programme
    output = []
    input_size = len(input)
    x = 0
    # On démarre la boucle :
    while x < input_size:
        # Les tailles et les valeurs sont restituer
        taille_sequence = input[x]
        valeur_sequence = input[x + 1]
        # Calcul pour redonner les valeurs dans le bon ordre
        output = output + [valeur_sequence] * taille_sequence
        x = x + 2
    return output # On redonne les valeurs
```

La réponse du programme python est :

64	0	0	…

Avec 63 fois 0.

La réponse est cohérente avec les valeurs obtenues avec les fonctions précédentes.