Université de Versailles – Saint-Quentin-en-Yvelines 2022-2023
Licence 1, IN202

Projet – Compression JPEG

Version du 21 février 2023

Yann Strozecki – Yann.Strozecki@uvsq.fr

Le but de ce projet est de créer des images au format JPEG. Nous implémenterons une version simplifiée, mais très proche de la réalité. Pour vous renseigner sur le format JPEG, regarder par exemple la vidéo `https://www.youtube.com/watch?v=Kv1Hiv3ox8I` et la page wikipedia `https://fr.wikipedia.org/wiki/JPEG`.

Le projet est à faire en Python, par groupe de 2. Vous présenterez votre travail lors d'une soutenance la semaine du 15 mai, sur les horaires de TD habituels. Des exemples d'images compressées sont donnés avec ce document, pour tester si votre code fonctionne correctement.

Le chargement des images vous est donné dans le fichier *projet.ipynb*. Vous obtenez alors un tableau numpy à trois dimension. Il s'agit en fait d'une matrice dont les éléments sont des triplets d'entiers dans $[0, 255]$ qui représentent les intensités en rouge, vert, bleu (RGB). On vous donne également le code qui permet d'afficher une image, qui vous servira à tester vos fonctions. Pour tester vos fonctions, on vous donne également la fonction PSNR qui calcule la proximité de deux images. Deux images identiques ont un PSNR de 100, des images très proches un PSNR d'environ 40.

Votre code doit comporter une section test qui permet de tester toutes les fonctions de votre programme et d'illustrer le fait qu'elles fonctionnent correctement.

1 Manipulation d'image

Nous allons changer la manière de représenter chaque pixel, en transformant l'information RGB en une information YCbCr. Il s'agit de trois valeurs qui représentent pour Y la luminance (luminosité) et Cb et Cr la chrominance (couleur). Les formules qui permettent de passer de RGB à YCbCr se trouvent par exemple sur `https://fr.wikipedia.org/wiki/YCbCr`.

Question 1 : Donner le code qui transforme une image RGB en une image YCbCr. Vous pourrez produire une matrice pour chaque composante, pour pouvoir plus facilement les manipuler indépendamment. Vous pouvez stocker les données YCbCr comme des entiers ou des flottants, mais vous expliquerez votre choix.

Question 2 : Donner le code qui transforme une image YCbCr en une image RGB. Attention, les valeurs des canaux RGB doivent être un entier dans $[0, 255]$ qui pourra être codé sur un octet. Appliquer successivement la transformation RGB vers YCbCr puis YCbCr vers RGB et vérifier que vous obtenez l'image de départ. Vous consulterez la documentation des fonctions de numpy *clip*, *uint8* et *mask* qui pourraient vous être utiles.

Pour pouvoir traiter une image, nous avons besoin que ses dimensions soient des multiples de 8. Pour le garantir, il faut faire du remplissage (padding), c'est à dire qu'on va ajouter des lignes et des colonnes de pixel noirs en bas et à droite de l'image. Si on a une image de dimension 15×21, on obtiendra une image de dimension 16×24 avec une ligne supplémentaire et trois colonnes supplémentaires.

Question 3 : Donner la fonction qui réalise ce padding ainsi que celle qui l'élimine et vérifier que l'application de ces deux transformations laissent l'image inchangée.

L'oeil est moins sensible aux informations de chrominance que de luminance. Pour exploiter cela, nous allons réduire la quantité d'information des canaux Cb et Cr en leur appliquant un sous-échantillonnage. Il s'agit de

remplacer deux pixels adjacents par la moyenne des deux pixels. Par exemple la matrice

$$\begin{pmatrix} 1 & 2 & 3 & 6 \\ 3 & 5 & 10 & 0 \end{pmatrix}$$

sera remplacé par la matrice

$$\begin{pmatrix} 1.5 & 4.5 \\ 4 & 5 \end{pmatrix}$$

dont la deuxième dimension est deux fois plus petite. On appelle ce mode d'échantillonage le $4:2:2$.

Question 4 : Implémenter la fonction qui sous-échantillonne une matrice et renvoie une matrice deux fois plus petite.

Pour retrouver une matrice de la bonne dimension, à partir d'une matrice obtenue par sous-échantillonnage, on répète chaque pixel deux fois.

Question 5 : Implémenter la fonction qui multiplie par deux la deuxième dimension d'une matrice. Tester à la suite le sous-échantillonnage et cette fonction, vous devez retrouver une image presque identique à celle de départ.

2 Blocs et traitement

Dans le format JPEG, l'image est décomposée en une grille de blocs de taille 8×8 qui seront traités indépendamment. C'est ce découpage qui créé des artefacts carrés caractéristiques du jpeg quand la compression est trop forte.

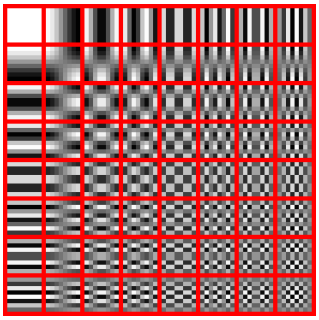

FIGURE 1 – Découpage d'une image en blocs 8×8

Question 6 : Soit une matrice dont les deux dimensions sont divisibles par 8. Donner une fonction qui découpe cette matrice en blocs 8×8 et les stocke dans une liste. L'ordre des blocs correspond à l'ordre de lecture d'une image.

Les blocs sont d'abord transformés en leur représentation par fréquence en utilisant une transformée en cosinus discrète. Dans le fichier source projet.ipynb, les fonctions *dct2(a)* et *idct2(a)* vous sont fournies. La fonction *dct2* donne la représentation en fréquence d'une matrice carrée (qui est une matrice carrée de la même dimension) et la fonction *idct2* calcule la fonction inverse.

Question 7 : Donner une fonction qui applique la transformée à chaque bloc d'une liste.

Vous allez implémenter plusieurs modes de compression. Dans le mode 0, on garde les blocs transformés tels quels.

Dans le mode 1, on impose un seuil aux coefficients. Il s'agit de remplacer tous le coefficients plus petit que ce seuil par 0. Vous choisirez un seuil qui permet de supprimer beaucoup de coefficients sans trop dégrader l'image.

Question 8 : Implémentez le filtrage des coefficients des blocs selon un seuil donné en argument.

Dans le mode 2, en plus du filtrage à seuil, vous appliquerez le sous-échantillonnage à la chrominance (Cb et Cr).

Question 9 : Donner une fonction qui est capable, à partir d'une image RGB, de créer les listes de blocs compressés dans les 3 modes.

3 Écriture dans un fichier

Pour simplifier cette partie, nous allons écrire l'information sous forme de texte. Pour ouvrir un fichier en mode écriture, vous pouvez faire *f = open(path, "w")*. Ensuite pour écrire une ligne vous pouvez faire *f.write("maligne\n")*. Si vous avez une variable numérique k, pour obtenir la chaîne de caractères représentant cette valeur, il suffit de faire *str(k)*.

Les trois questions suivantes détaillent les différentes parties de votre fonction d'écriture dans un fichier. Attention à respecter scrupuleusement les instructions, autrement nous ne pourrons pas lire les fichiers que vous aller produire.

Question 10 : Pour commencer, vous écrirez quatres lignes contenant les informations de votre image. La première ligne contiendra le type du fichier : "SJPG". La deuxième ligne contiendra les dimensions de l'image dans l'ordre hauteur puis largeur, séparées par un espace, par exemple "200 300". La troisième ligne contiendra le mode de compression, par exemple "mode 1". La quatrième ligne contiendra "RLE" si vous utilisez un run length encoding, ou "NORLE" sinon.

Question 11 : Vous écrirez ensuite le contenu des blocs, d'abord ceux de Y, puis ceux de Cb puis ceux de Cr. Chaque bloc est écrit sur une ligne, les valeurs étant des entiers séparés par de espaces.

Pour améliorer la compression, nous allons utiliser le run length encoding (RLE). Il s'agit d'une méthode similaire au code LZW, mais bien plus simple. Quand un caractère est répété plusieurs fois de suite, on va écrire le caractère et son nombre de répétitions. Ici se sont les zéros qui sont répétés de nombreuses fois, donc on appliquera ce code uniquement aux zéros. À la place de k zéros consécutifs, on écrira "♯k" ou k est un entier.

Question 12 : Ajouter une option à votre fonction d'écriture pour qu'elle puisse écrire les blocs en appliquant le codage RLE.

4 Décompression

Il s'agit d'écrire les fonctions qui à partir d'une image compressée par votre code permet de recréer l'image en RGB qu'on pourra ensuite afficher. Quand vous aurez fait cette section, vous pourrez beaucoup plus facilement débuguer le code de la compression (en visualisant le résultat). Si vous n'arrivez pas à faire cette section, nous vous encourageons à faire décompresser les fichiers que vous avez produit par le décodeur d'un de vos camarades pour vérifier qu'ils sont corrects. Vous pourrez appliquer votre fonction de décompression aux quatre fichiers compressés fournis avec le sujet pour vérifier sa correction.

Question 13 : Écrire une fonction de décompression qui prend une liste de blocs pour chaque canal Y,Cb,Cr contenant les coefficients de la DCT et calcule une matrice représentant l'image en RGB.

Question 14 : Écrire une fonction qui lit un fichier SJPG et qui créé les listes de blocs décrites par le fichier.

5 Optimisations

Cette partie est **complètement facultative** et vous propose d'implémenter les méthodes utilisées dans le format JPEG pour mieux compresser.

On propose un mode 3, dans lequel on passe à 0 90% des coefficients. On choisit les coefficients les plus petits globalement, sur tous les blocs.

Question 15 : Implémenter une fonction qui réalise la compression en mode 3. La décompression ne change pas.

On propose un mode 4, dans lequel on fournit une table de quantification de dimension $8 * 8$ donnée par exemple dans `https://fr.wikipedia.org/wiki/JPEG`. Il s'agit de diviser les coefficients des blocs par les coefficients de cette table. Pour décompresser, il faut multiplier par les coefficients de la table.

Question 16 : Implémenter la compression et la décompression par le mode 4. L'écriture dans un fichier ne change pas.

Pour améliorer la compression, on a intérêt à regrouper les 0, ce qui améliore la méthode RLE. Pour cela, il vaut mieux parcourir les blocs 8×8 en zig-zag, comme illustré dans `https://fr.wikipedia.org/wiki/JPEG`. Pour aller plus loin, on peut même faire un codage de Huffman des informations stockées.

Question 17 : Écrire et lire les coefficients dans le fichier en suivant l'ordre en zig-zag.

6 Modalités pratiques

Merci de respecter les consignes suivantes :

- le projet est à faire en binôme ;
- le projet est à rendre sur ecampus, au plus tard le **lundi 15 mai à 9h00** ;
- une soutenance aura lieu la semaine du 15 mai pendant votre séance de TD.
- vous devez déposer un fichier `XY_NOM1_Prenom1-NOM2_Prenom2.zip` qui est le zip du dossier `XY_NOM1_Prenom1-NOM2_Prenom2` contenant votre projet.
  `XY` sont les initiales de votre chargé de TD (`LD`, `CG`, `LG`, `YS`).
- Un outil de détection de plagiat sera utilisé sur vos codes.

Le retard de la remise du projet entraîne 1 point de moins par heure de retard.