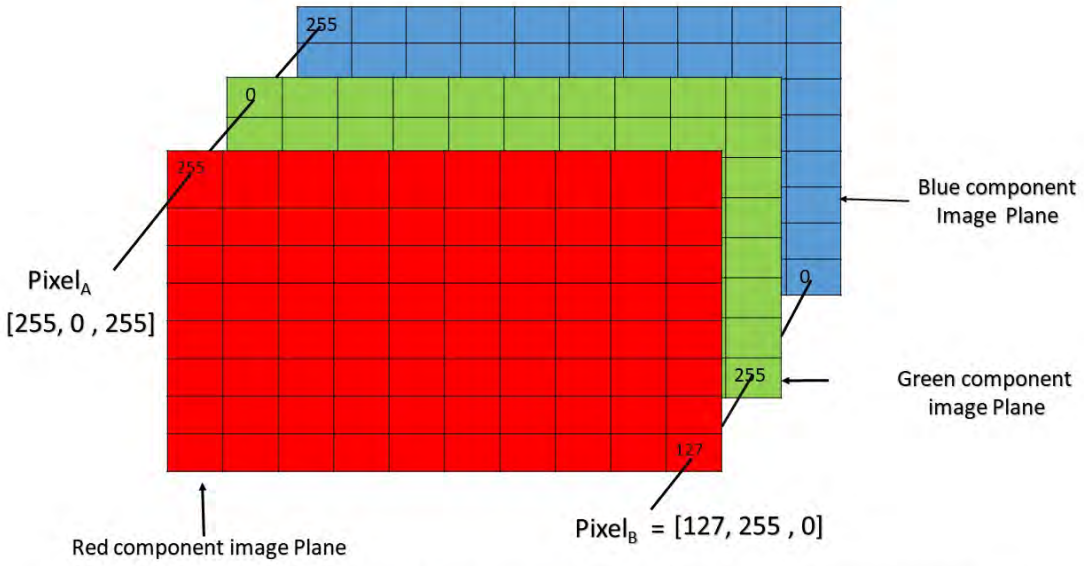
Projet MAM3 - Compression d'images par transformée de Fourier

Didier Auroux & Cédric Boulbe

Décembre 2022 - Janvier 2023

1 Représentation numérique d'une image

Une image numérique peut être représentée sous forme d'un tableau multidimensionnel. Il existe plusieurs formes de représentation, on considère ici la plus simple où l'image est décomposée en trois composantes de couleur : RGB (Red-Green-Blue). Une image bi-dimensionnelle, de $n_x \times n_y$ pixels, se représente alors sous forme d'une matrice tri-dimensionnelle de dimension $n_x \times n_y \times 3$. Les entrées de cette matrice correspondent aux intensités lumineuses de chaque pixel dans chacun des trois canaux de couleur (rouge, vert et bleu).

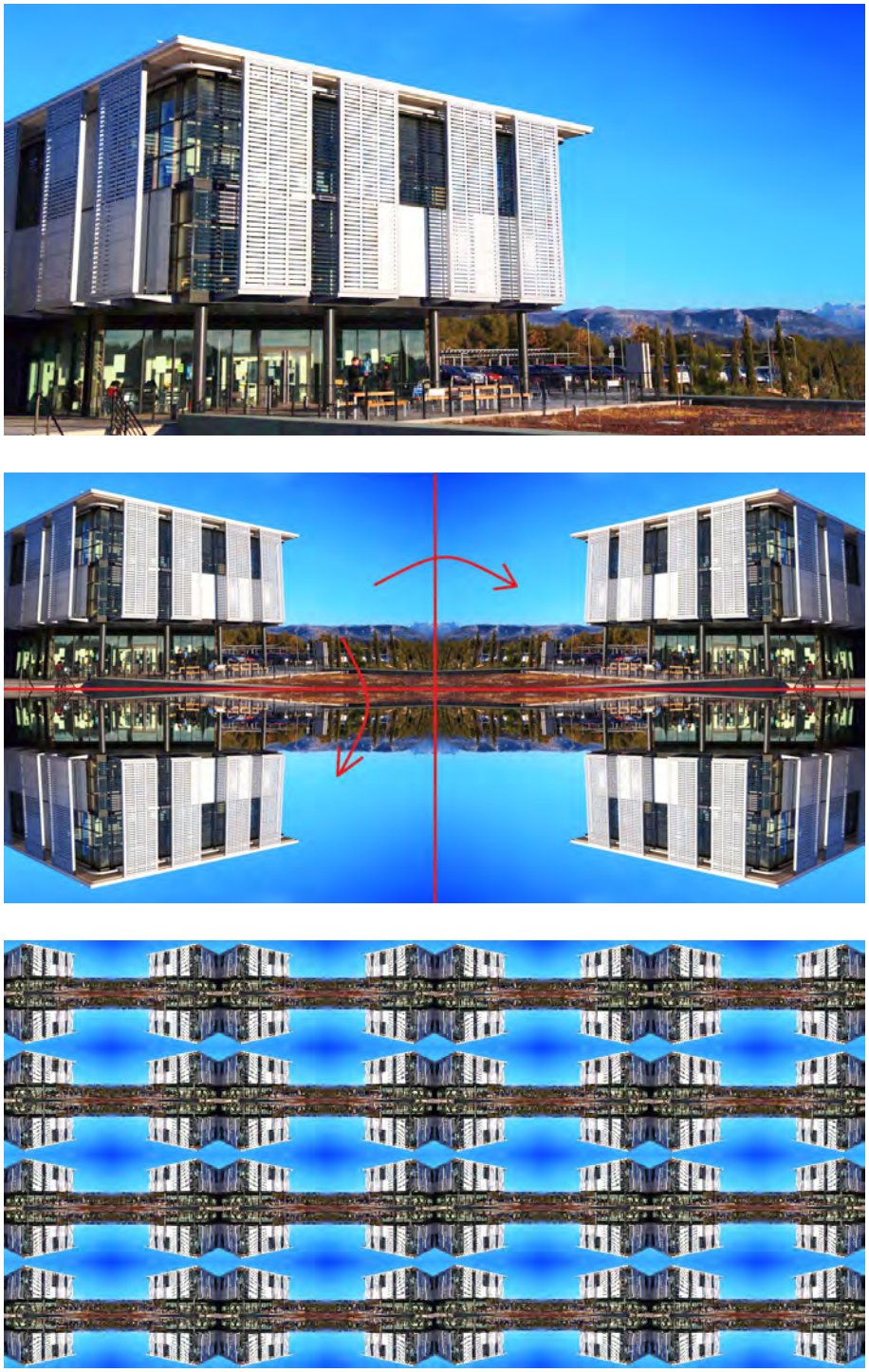


Pixel of an RGB image are formed from the corresponding pixel of the three component images

Suivant les formats d'images, et la façon de charger l'image, ces intensités sont soit des réels compris entre 0 et 1, soit des entiers compris entre 0 et 255. L'image chargée peut aussi avoir un quatrième canal, qui ne contient pas d'information).

On souhaite ici utiliser des transformées de Fourier (élément clé du traitement du signal et des images) pour étudier et compresser l'information contenue dans une image.

Par symétrisation et périodisation d'une image, on peut la rendre *infinie* dans les deux directions, périodique et paire. Toute fonction périodique peut se décomposer dans une base de fonctions cosinus et sinus, et si elle est paire, seulement avec des fonctions cosinus.



Processus de périodisation d'une image

Nous allons donc pouvoir décomposer n'importe quelle image comme une combinaison linéaire de fonctions cosinus, en x et en y. Autrement dit, la DFT (transformée de Fourier discrète) peut se ramener à une transformée de cosinus uniquement, par parité de la fonction (image périodique ainsi construite).

2 Transformée de cosinus discrète (DCT) d'une image

Afin de simplifier les calculs, nous allons travailler sur des blocs 8×8 de l'image. La période, dans chaque direction, sera donc 16 (puisque par symétrie, on ne travaille que sur une demi-période).

Avant d'appliquer la DCT (Discrete Cosine Transform), restriction de la DFT (Discrete Fourier Transform) aux fonctions paires, on va supposer que les valeurs de l'image sont centrées autour de 0 (cela revient à se débarrasser du cœfficient constant dans la transformée).

On soustrait donc 128 aux intensités de chaque pixel, afin de traiter des entiers compris entre -128 et 127.

Si on suppose qu'un bloc 8×8 est représenté par une matrice $M = (M_{i,j})_{0 \leq i,j \leq 7}$ de dimension 8×8 (en supposant qu'on ne traite qu'un seul canal de couleur), la DCT de cette petite image s'écrit alors en dimension 2, pour $0 \leq k, l \leq 7$:

$$D_{k,l} = \frac{1}{4} C_k C_l \sum_{i=0}^{7} \sum_{j=0}^{7} M_{i,j} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cos\left(\frac{(2j+1)l\pi}{16}\right), \tag{1}$$

où $C_0 = \frac{1}{\sqrt{2}}$ et $C_k = 1$ si $k > 0$ (et idem en l).

On obtient alors une matrice D de même taille que la matrice M originale (donc 8×8).

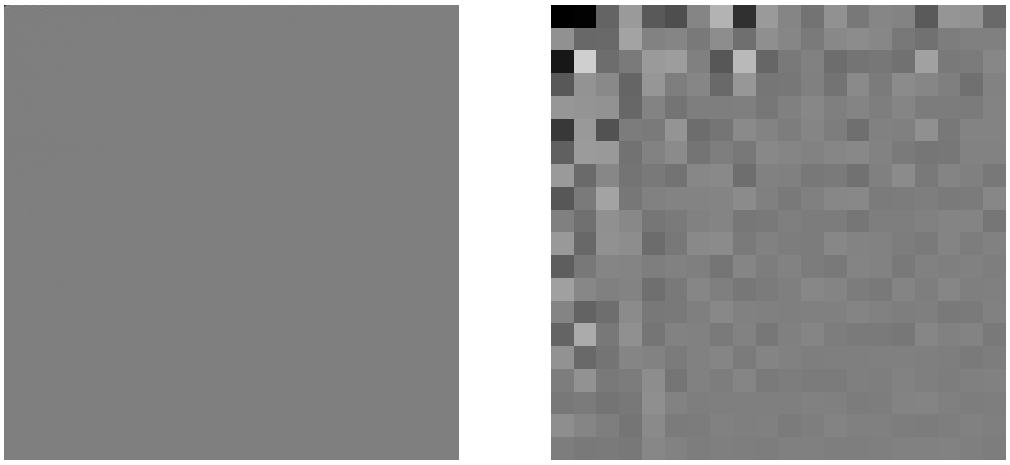
Ce choix particulier de DCT (généralement appellée DCT-II) permet d'obtenir un opérateur orthogonal. La transformation adjointe (transposée) donne alors la transformée inverse (DCT inverse).

La transformation décrite dans la formule (1) s'apparente simplement à un changement de base orthonormée (passage en mode *fréquentiel*), et peut donc se réécrire sous la forme

$$D = PMP^T$$

où P est une matrice orthogonale contenant les cœfficients de la DCT.

Les entrées de la matrice D donnent la fréquence des changements d'intensité lumineuse. Les cœfficients dans le coin supérieur gauche de la matrice D correspondent aux petites fréquences en x et en y (variations lentes), alors que les cœfficients en bas à droite correspondent aux variations d'intensité aux hautes fréquences (variations rapides).



DCT de l'image précédente, et zoom sur les 20 premières fréquences uniquement (et donc sur les 20×20 premiers pixels de la DCT).

3 Compression

La compression passe par une étape de quantification, dans laquelle on ne va garder que les modes les plus importants (qui sont généralement les basses fréquences - les hautes fréquences pouvant être vues comme du bruit)

L'idée est alors de diviser terme à terme la matrice D par une matrice de quantification Q, puis d'arrondir le résultat (à la partie entière).

La matrice de quantification Q dans la norme de compression JPEG est la suivante :

$$Q = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 13 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{2}$$

Les termes en bas à droite de la matrice Q étant élevés, la division par Q va quasiment toujours annuler les termes en bas à droite, correspondant aux hautes fréquence (sauf s'ils étaient très élevés dans la matrice D). A l'inverse, l'essentiel de l'information qui se trouve généralement aux basses fréquences sera conservé, les cœfficients de Q en haut à gauche étant nettement plus petits.

La matrice ainsi obtenue sera très creuse, avec très peu de valeurs non nulles. Sur une image standard, on peut facilement diviser par 10 à 15 le nombre de valeurs non nulles (et ainsi avoir un taux de compression de l'ordre de 85 à 90%).

4 Décompression

À partir de l'image compressée, il suffit d'appliquer les opérations à l'envers. On commence par découper l'image en blocs 8×8, puis on multiplie terme à terme la matrice compressée par Q. On obtient alors l'image décompressée exprimée dans la base des cosinus $\tilde{D}$. On applique alors le changement de base inverse à $\tilde{D}$ pour obtenir l'image exprimée dans la base des intensités lumineuses : $\tilde{M} = P^T \tilde{D} P$.

Il suffit alors de réassembler les blocs $\tilde{M}$ 8×8 que l'on a obtenus pour récupérer l'image *originale*. Comme on peut le constater, elle est visuellement très proche de l'image originale (ici, environ 10% d'erreur relative par rapport à l'image de départ, mais visuellement on ne le voit pas).

Et il suffit d'appliquer la transformation à chaque canal de couleur pour traiter les images en couleur.

Image originale - image compressée (taux de compression de 90%, erreur de 10%)

Image originale - image compressée (taux de compression de 90%, erreur de 10%)

5 Filtrage des hautes fréquences et débruitage

Une façon simple, mais efficace, pour compresser une image (mais aussi pour enlever du bruit sur une image), consiste à simplement tronquer l'information au-delà d'une certaine fréquence. On remplace alors l'étape de quantification (division par Q à la compression, puis remultiplication par Q à la décompression) par une simple troncature. On met à 0 tous les coefficients $D_{l,k}$ de la matrice D dont les indices vérifient $l + k \geq F$ où F est la fréquence de coupure ($F = 6$ par exemple).

Et on peut évidemment combiner les deux étapes : quantification pour comprimer, et troncature pour filtrer les hautes fréquences et débruiter - puis décompression.

C'est moins efficace que la matrice Q de la norme JPEG, mais c'est aussi plus rapide, et on se rend compte de l'importance de l'information stockée dans les toutes premières (basses) fréquences.

Image originale - image tronquée en ne gardant que les fréquences ≤ 6, 4, ou 2. Taux de compression de 80, 90 et 95%, erreur de 10, 18 et 25%.

Cette approche permet aussi à moindre coût de débruiter une image, lorsque le bruit est blanc gaussien (et donc porte principalement sur des hautes fréquences).

Image bruitée et débruitée en tronquant les hautes fréquences.

6 Projet

Implémenter cet algorithme en python, le tester sur différentes images (N&B, couleur, plus ou moins oscillantes/bruitées, ...), essayer d'autres matrices de quantification, étudier les pertes dûes à la compression, les effets du débruitage, comparer avec une DCT implémentée par une librairie toute faite, ...

Recommandation : utiliser Jupyter Notebook sur abel :
`https://abel.polytech.unice.fr/jupyter/hub/login`

⇒ Notebook "Python MAM"

Librairies et commandes python utiles : numpy, math, pyplot (dans matplotlib) ; pyplot.imread, pyplot.imsave ; pour un tableau qui s'appelle data : data.shape, data.astype(int) ; numpy.maximum, numpy.minimum ; math.sqrt, math.pi, math.cos ; numpy.matmul, numpy.transpose, numpy.divide ; numpy.count_nonzero ; numpy.linalg.norm

7 Algorithme

- **Initialisation :**

— Lire l'image
— Tronquer l'image à des multiples de 8 en x et y
— Ne garder qu'une composante (couleur) pour avoir une matrice $2D$ dont les dimensions sont multiples de 8
— Transformer les intensités en entiers entre 0 et 255, puis centrer pour se ramener entre -128 et 127
— Définir la matrice de passage en fréquentiel de la DCT2 P

- **Compression :**

— Pour chaque bloc 8×8 de l'image :
  — Appliquer le changement de base $D = PMP^T$
  — Appliquer la matrice de quantification terme à terme $D./Q$ et prendre la partie entière
  — et/ou Filtrer les hautes fréquences et mettre à 0 les cœfficients sur les dernières lignes et colonnes dans la matrice
— Stocker ces nouveaux blocs 8×8 dans la matrice compressée/débruitée
— Compter le nombre de cœfficients non nuls pour obtenir le taux de compression

- **Décompression**

— Pour chaque bloc 8×8 :
  — Multiplier par la matrice Q terme à terme
  — Appliquer la transformée inverse $P^T \tilde{D} P$
— Réassembler la matrice décompressée/débruitée

- **Post-processing :**

— Comparer cette matrice avec la matrice originale (en norme L^2 relative par exemple)
— Le faire pour chaque canal de couleur si nécessaire
— Re-transformer les valeurs entre -128 et 127 en réels entre 0 et 1 et sauver l'image (pour la comparer visuellement)