

Projet MAM3 - Compression d'images par transformée de Fourier

Yassine Zouhri Lyna Gaiech Sarah Benazouz

6 janvier 2024



Table des matières

1	Introduction	2
2	Objectifs	2
3	Explications théoriques	2
3.1	Représentation d'une image	2
3.2	Transformées de Fourier	2
3.3	Transformée de cosinus discret	3
4	Réalisations pratiques	3
4.1	Initialisation	3
4.2	Calcul de la matrice P	3
4.3	Compression	4
4.4	Filtrage	4
4.5	Decompression	4
4.6	Post-processing	4
5	Résultats	4
6	Conclusion	8
7	Annexe	8
7.1	Plan de travail et carnet de bord	8
7.2	Difficultés rencontrées	8

1 Introduction

Au fil des dernières années, les smartphones ont considérablement évolué dans leur capacité à capturer des images de haute qualité et en grande quantité. Cependant, ces images, si stockées telles quelles, peuvent occuper un espace de stockage important sur nos appareils.

Pour optimiser cet espace limité, les images sont souvent soumises à un processus de compression. Ce processus permet de conserver davantage de photos sur nos téléphones portables. Bien que la compression implique une perte de qualité inévitable, cette perte demeure généralement minime et peu perceptible, surtout lorsqu'on visualise les images sur nos téléphones.

Pour réaliser cette compression, nous cherchons à développer un programme en Python utilisant les transformées de Fourier. Ces transformées sont des outils clés pour effectuer des opérations de compression d'images tout en préservant une qualité visuelle acceptable.

2 Objectifs

Nous envisageons plusieurs objectifs dans le cadre de notre projet. Tout d'abord, nous cherchons à acquérir une compréhension approfondie du processus de compression/décompression des images, ainsi que du processus de filtrage.

Ensuite, notre objectif est de développer un programme de compression qui utilise la transformée de Fourier, puis de procéder à une comparaison des résultats obtenus. Nous souhaitons analyser les performances et les caractéristiques de notre approche par rapport à d'autres méthodes.

Enfin, nous avons l'intention de mener des comparaisons avec d'autres techniques, y compris avec d'autres Q, et d'évaluer notre code en le confrontant à des implémentations déjà existantes en Python.

Pour atteindre ces objectifs, nous avons constitué un plan de travail réaliste et avons tenu un carnet de bord que vous pourrez retrouver en annexe.

3 Explications théoriques

3.1 Représentation d'une image

Pour représenter une image, nous utilisons le modèle de la représentation RGB . Ainsi, l'image peut-être visualisée sous forme d'une matrice tri-dimensionnelle. Cela se traduit par 3 matrices dimension $nx \times ny$, chacune représentant une composante de couleur. Les valeurs des coefficients dans ces matrices peuvent être des nombres réels compris entre 0 et 1 ou des entiers entre 0 et 255, selon le format de l'image et la méthode utilisée pour charger cette image.

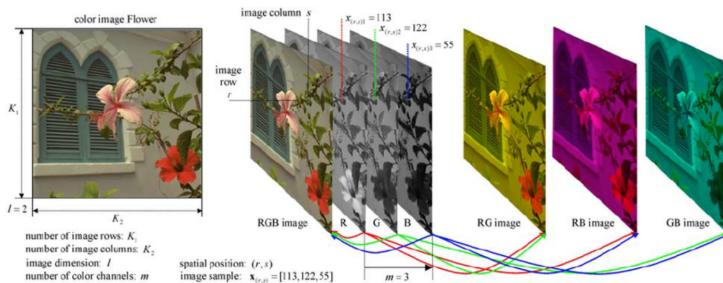


FIGURE 1 – Image représenté en RGB

3.2 Transformées de Fourier

Les transformées de Fourier jouent un rôle essentiel dans le traitement des signaux et des images. Elles sont utilisées pour analyser et compresser les informations contenues dans une image.

En symétrisant et en rendant périodique une image, on la considère infinie dans toutes les directions, périodique, et possédant une symétrie par rapport à un point central. Cette caractéristique de symétrie paire de l'image permet d'utiliser une forme simplifiée de la transformée de Fourier discrète, connue sous le nom de transformée de cosinus discrète (DCT). En exploitant cette propriété, les coefficients nuls de la DCT peuvent être éliminés, ce qui permet d'optimiser l'encodage

de l'image. La DCT est souvent utilisée pour la compression d'images car elle a la capacité de regrouper l'énergie du signal dans un petit nombre de coefficients, ce qui permet de représenter l'image avec moins d'informations sans perdre trop de qualité visuelle.

3.3 Transformée de cosinus discret

La DCT est mathématiquement définie par des formules qui calculent les coefficients de cosinus pour une série de valeurs, et elle peut être appliquée de manière séquentielle sur des blocs d'une image pour compresser ou décompresser des données.

Avant toute chose, nous supposons que les valeurs de l'image sont centrées autour de 0 et on soustrait 128 aux intensités de chaque pixel, afin de traiter des entiers compris entre -128 et 127.

Pour appliquer la DCT, nous utilisons une matrice D tel que ces coefficients sont :

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

Cette matrice D représente la fréquence des changements d'intensité lumineuse. Les coefficients en haut à gauche de la matrice D correspondent aux fréquences basses, alors que les coefficients en bas à droite correspondent aux fréquences hautes. Il peut arriver que l'image soit bruité, pour résoudre ce problème, nous effectuerons un tronquage à partir d'une certaine fréquence.

4 Réalisations pratiques

Pour diminuer la complexité de notre code, nous avons fait le choix de ne pas utiliser de fonctions et d'éviter les boucles. Nous avons favorisé l'utilisation d'expression vectorielles de numpy.

4.1 Initialisation

Nous commençons par importer les bibliothèques numpy, matplotlib.pyplot et matplotlib.image. L'image est lue à partir du chemin spécifié. Si l'image a un canal alpha (transparence), le canal alpha est ignoré. Elle est ensuite transformée en matrice 2D. L'image est tronquée pour avoir des dimensions multiples de 8, ce qui est requis pour la DCT. Les intensités des canaux sont normalisées dans la plage [-128, 127].

4.2 Calcul de la matrice P

Nous avons besoin de définir la matrice P pour l'appliquer à notre code.

On a

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

et

$$D = P M P^T$$

On pose $\forall k, d \in [0, 7]$

$$\begin{aligned} PM &= \sum_{k=0}^7 p_{i,k} m_{k,j} \\ (PM)P^T &= \sum_{d=0}^7 \left(\sum_{k=0}^7 p_{i,k} m_{k,d} \right) p_{d,i} \\ (PM)P^T &= \sum_{d=0}^7 \left(\sum_{k=0}^7 p_{i,k} p_{d,i} m_{k,d} \right) \\ (PM)P^T &= \sum_{j=0}^7 \left(\sum_{i=0}^7 p_{k,i} p_{j,k} m_{i,j} \right) \end{aligned}$$

et on a

$$P P^T = I$$

donc par identification, $\forall i, j \in [0, 7]$

$$P_{i,j} = \frac{1}{2} C_i \cos\left(\frac{(2i+1)j\pi}{16}\right)$$

4.3 Compression

Nous définissons la matrice de quantification (Q) Pour chaque bloc 8x8, la DCT est appliquée en utilisant la matrice de base P. Les résultats sont quantifiés en divisant par la matrice de quantification Q et en arrondissant les valeurs. Les valeurs quantifiées sont clipsées pour rester dans la plage acceptable de valeurs pour les composantes de couleur. Les résultats sont stockés dans les matrices compressées MC, HC et KC, qui sont ensuite utilisées dans le reste de l'algorithme de compression JPEG.

4.4 Filtrage

Nous demandons à l'utilisateur quelle est la fréquence de coupure. Les hautes fréquences au-dessus de cette fréquence de coupure sont alors supprimées, puis nous calculons le taux de compression.

4.5 Decompression

Nous parcourons à l'aide de boucles les blocs 8x8 de l'image et pour chaque composante , la quantification inverse est effectuée en multipliant les coefficients quantifiés par la matrice de quantification inverse. La matrice de base P est ensuite utilisée pour effectuer la transformation inverse DCT.

Nous convertissons les valeurs de retour à la plage originale.

4.6 Post-processing

Nous calculons le taux d'erreur et la nouvelle image est affichée. Nous avons effectué plusieurs tests sur l'image. Vous retrouverez les résultats ci-dessous. Nous avons effectué une exécution du code sans filtre 3, une exécution sans matrice de quantification 4.

Nous avons aussi implementé notre code avec une matrice de quantification pour chrominance 5 et pour luminance 6, correspondant respectivement aux Q suivants :

$$Q1 = \begin{pmatrix} 20 & 23 & 26 & 30 & 35 & 40 & 46 & 53 \\ 23 & 26 & 30 & 35 & 40 & 46 & 53 & 60 \\ 26 & 30 & 35 & 40 & 46 & 53 & 60 & 68 \\ 30 & 35 & 40 & 46 & 53 & 60 & 68 & 76 \\ 35 & 40 & 46 & 53 & 60 & 68 & 76 & 85 \\ 40 & 46 & 53 & 60 & 68 & 76 & 85 & 94 \\ 46 & 53 & 60 & 68 & 76 & 85 & 94 & 104 \\ 53 & 60 & 68 & 76 & 85 & 94 & 104 & 114 \end{pmatrix}$$
$$Q2 = \begin{pmatrix} 10 & 15 & 25 & 37 & 51 & 66 & 82 & 100 \\ 15 & 19 & 28 & 39 & 52 & 67 & 84 & 101 \\ 24 & 28 & 35 & 45 & 58 & 73 & 90 & 107 \\ 37 & 39 & 45 & 51 & 65 & 80 & 97 & 114 \\ 51 & 52 & 58 & 65 & 76 & 91 & 108 & 125 \\ 66 & 67 & 73 & 80 & 91 & 105 & 122 & 139 \\ 82 & 84 & 90 & 97 & 108 & 122 & 140 & 157 \\ 100 & 101 & 107 & 114 & 125 & 139 & 157 & 175 \end{pmatrix}$$

Enfin, nous avons implémenté notre code avec la dct de Python 7. Nous avons compressé une image en noir et blanc 8. Enfin, nous avons fait le test avec une image de grande qualité, ce qui marche aussi très bien.⁹

5 Résultats

Voici les résultats obtenus pour chaque test.

le taux de compression est de l'ordre de 84.93734335839599
 Quel est la fréquence de coupure? 6
 Taux d'erreur total sur l'image en couleur : 0.04631735881051515



Temps d'exécution : 2.812161445617676 secondes

FIGURE 2 – Résultat avec filtrage et matrice de quantification

Nous obtenons un taux de compression de 85, et un taux d'erreur de 4,6, ce qui est cohérent. En revanche, notre temps d'exécution est de quasiment 3 secondes, ce qui paraît un peu long pour une opération qui devrait s'effectuer très rapidement dans nos appareils.

le taux de compression est de l'ordre de 84.93734335839599
 Taux d'erreur total sur l'image en couleur : 0.040580159642963624

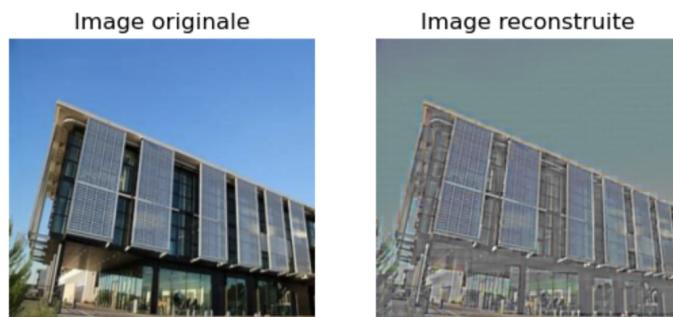


Temps d'exécution : 1.095094919204712 secondes

FIGURE 3 – Résultat sans filtrage et avec matrice de quantification

Le taux de compression est de 56, ce qui n'est pas très bon et nous voyons bien que le résultat n'est pas optimal, les couleurs sont ternes.

le taux de compression est de l'ordre de 56.26879699248121
 Taux d'erreur total sur l'image en couleur : 0.34838343



Temps d'exécution : 1.1858551502227783 secondes

FIGURE 4 – Résultat avec filtrage et sans matrice de quantification

le taux de compression est de l'ordre de 89.53281641604009
 Quel est la fréquence de coupure? 6
 Taux d'erreur total sur l'image en couleur : 0.05581588318187937



FIGURE 5 – Résultat avec filtrage et avec matrice de quantification pour chrominance

le taux de compression est de l'ordre de 88.296914160401
 Quel est la fréquence de coupure? 6
 Taux d'erreur total sur l'image en couleur : 0.0514415092811888

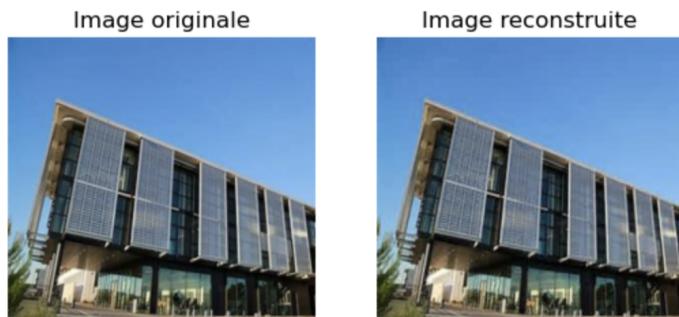


FIGURE 6 – Résultat avec filtrage et avec matrice de quantification pour luminance

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
 Taux d'erreur : 0.0776

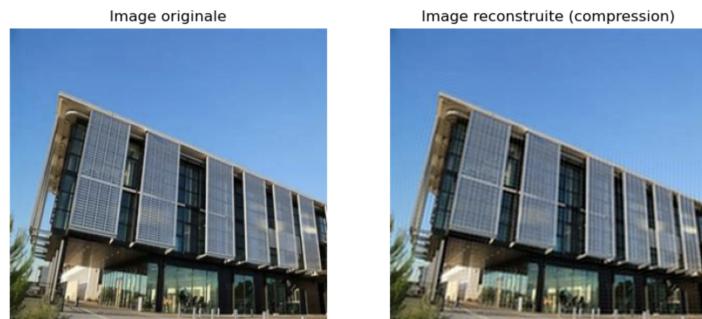


FIGURE 7 – Résultat avec la dct de python
 Nous remarquons que l'image est de bien moins bonne qualité.

le taux de compression est de l'ordre de 90.83916505791507
Taux d'erreur total sur l'image en couleur : 0.03501015845139028



FIGURE 8 – Résultat sur une image en noir et blanc

Nous remarquons que le taux de compression est plus élevé 8. C'est normal car l'image non coloré est plus simple à traiter.

le taux de compression est de l'ordre de 85.0770918367347
Taux d'erreur total sur l'image en couleur : 0.035594104341793165

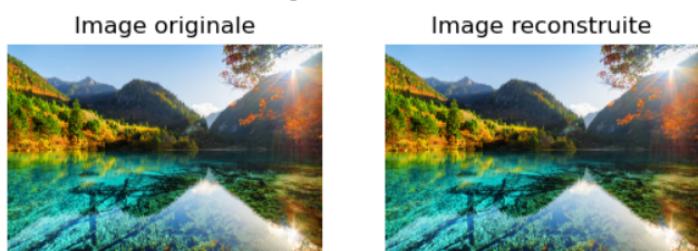


FIGURE 9 – Résultat sur image hd

6 Conclusion

En conclusion, nous pouvons affirmer que la transformée de cosinus discrète est un outil optimal pour la compression d'images. Sans son utilisation, les images ne pourraient être stockées en gardant une qualité tolérable. Mais les transformées de Fourier ne sont pas la seule solution. Nous pouvons aussi utiliser les noyaux de convolution, et ainsi se demander comment réaliser une compression avec un programme python à l'aide de cette méthode.

7 Annexe

7.1 Plan de travail et carnet de bord

Nous avons décidé de créer le code en début de semaine, en rédigeant le rapport en parallèle, afin de consacrer du temps aux tests, et à la préparation à l'oral. Bien que le rapport ne soit qu'à rendre à la rentrée, nous trouvons cela plus judicieux de le rédiger au fur et à mesure afin de garder en tête notre parcours et nos objectifs. Nous avons perfectionné notre code et achevé le rapport pendant les vacances

Lundi matin :

Prise de connaissance du sujet, compréhension et organisation (groupe)

Calcul de la matrice P (Lyna)

Rédaction de l'introduction du rapport et de la présentation (Sarah et Yassine)

Lundi après midi :

Code de la fonction décompression (Yassine)

Code de la fonction DCT et compression (Lyna)

Rédaction du rapport (Sarah)

Mardi :

Assemblage, discussion du code (groupe)

Debugging du code (Yassine et Lyna)

Rédaction du rapport et de la présentation (Sarah)

Mercredi :

Optimisation du code (Lyna et Yassine)

Tests de différents type d'image (groupe)

Jeudi matin :

Préparation à l'oral (groupe)

Jeudi après midi :

Rédaction du rapport (groupe)

Optimisation du code (Yassine et Lyna)

Vendredi matin :

Soutenance

7.2 Difficultés rencontrées

A partir de mardi, nous avons eu du mal à avancer sur le projet car une erreur se trouvait dans notre code, et l'image que nous obtenions n'était pas de même couleur que l'image originale. Après avoir relu plusieurs fois le code, nous avons finalement réalisé que nous multiplions P sur les lignes au lieu des colonnes. Après avoir corrigé cette erreur, notre code marchait et nous pouvions enfin faire les différents tests. Lors de notre soutenance, nous nous sommes rendus compte que la façon dont nous calculions le taux de compression était erronée. Nous avons compris qu'il ne fallait pas soustraire à 1. Nous avons mis quelques jours à comprendre qu'on devait aussi diviser par 3, car nous avons 3 matrices de couleur.