

Rapport du projet TS2I

Traitement du signal et d'image : Compression d'images par transformée de Fourier
13/01/2025 - 17/01/2025

BEN KHALIFA Emna, COSTANTIN Perline, HONAKOKO Giovanni, ZOUARHI Yassmin
MAM 3

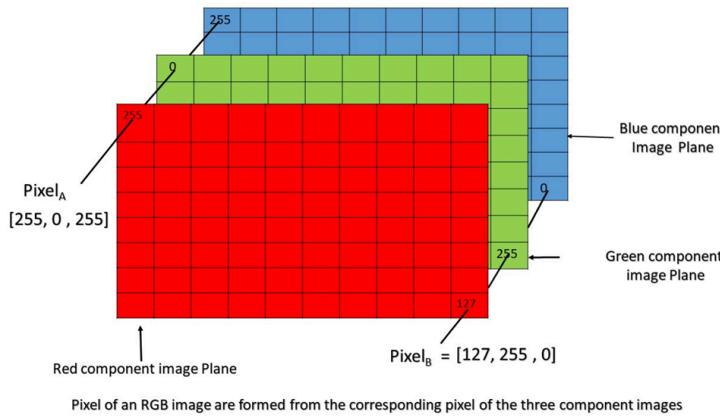
Sommaire :

- I) Introduction
- II) Concepts mathématiques
- III) Algorithme
- IV) Résultats
- V) Difficultés rencontrées
- VI) Conclusion

I) Introduction

Dans le cadre de ce projet “Traitement du signal et de l'image”, nous avons travaillé sur la représentation d'images numériques.

En effet, une image peut être représentée sous forme d'un tableau multidimensionnel: on considère ici la représentation RGB (rouge, vert et bleu) où l'image est décomposée en trois composantes de couleur. Une image de $nx \times ny$ pixels se représente sous forme d'une matrice tri-dimensionnelle de dimension $nx \times ny \times 3$. Les entrées de cette matrice correspondent aux intensités lumineuses de chaque pixel dans chacun des trois canaux de couleur.



Le but du projet est d'implémenter un programme Python afin de compresser puis décompresser des images à l'aide des transformées de Fourier.

Objectifs

1. Comprendre les bases théoriques transformée de cosinus discrète (DCT) d'une image
2. Implémenter un algorithme de compression d'images
 - Segmenter une image en blocs de 8x8 pixels pour simplifier les calculs.
 - Appliquer la transformée de cosinus discrète sur chaque bloc
3. Implémenter un algorithme de décompression
 - Reconstruire l'image compressée
 - Comparer l'image décompressée avec l'image d'origine en termes de qualité et de précision.
4. Évaluer la performance de la méthode

II) Concepts mathématiques

L'objectif est donc ici d'utiliser les transformations de Fourier dans l'analyse et la compression des données contenues dans une image.

Une image peut être étendue de manière infinie dans les deux directions en appliquant un processus de symétrisation et de périodisation. Or une fonction périodique peut être décomposée en une base de fonctions cosinus et sinus ou seulement avec la fonction cosinus si elle est paire. Ce principe permet de décomposer une image en combinaison linéaire de fonctions cosinus en x et en y. Par conséquent, la transformée de Fourier discrète d'une image périodisée peut être représentée uniquement à l'aide de fonctions cosinus.

Ce principe est appliqué via la transformée de Fourier discrète (DFT), qui est restreinte ici à la transformée de cosinus discrète (DCT). La DCT est calculée sur des blocs de 8x8 pixels et s'exprime pour un bloc représenté par une matrice $M=(M_{i,j})$ de dimension 8×8 comme suit :

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

###montrer comment on passe de la formule au dessus à D=PMPT

La matrice D contient les amplitudes des fréquences : les coefficients situés en haut à gauche correspondent aux basses fréquences , tandis que ceux en bas à droite représentent les hautes fréquences. La compression est réalisée en divisant D par une matrice de quantification Q et en arrondissant les résultats, ce qui élimine les hautes fréquences. La reconstruction de l'image est obtenue par la DCT inverse, définie comme $M=PT DP$, où P est une matrice orthogonale contenant les coefficients de la DCT. Ces concepts permettent une compression efficace tout en conservant l'essentiel de l'information visuelle.

Lors de la compression, certaines informations, notamment les hautes fréquences (qui correspondent aux détails fins de l'image), sont souvent supprimées pour réduire la taille du fichier, ce qui peut causer des imperfections comme des blocs visibles ou du bruit. En appliquant des filtre passe bas (supprimer les hautes valeurs de la matrice qui dépasse la valeur de coupure) en peut avoir une image décompressée plus clair et net.

III) Algorithme

1) Initialisation :

Lire l'image :

Tout d'abord, nous commençons par lire une image enregistrée sur notre ordinateur en utilisant la fonction imread de la librairie matplotlib.pyplot. Cela permet de récupérer une matrice avec les données de l'image : nombre de pixels en hauteur, en largeur et nombre de canaux. L'image est ensuite affichée à l'aide de imshow de la librairie matplotlib.pyplot..

Tronquer l'image :

La seconde étape de l'initialisation est de découper l'image en sous-images de 8×8 pixels. Pour cela, nous devons tronquer l'image à des multiples de 8 en x et y. Nous redimensionnons dans un premier temps l'image pour que son nombre de pixels en largeur et en hauteur soit un multiple de 8 en supprimant les pixels en trop. Une fois l'image redimensionnée, nous procédons à la division en sous-images ou blocs de 8×8 .

Transformation des intensités :

En ajoutant une condition, on vérifie si les données de l'image ne sont pas déjà dans le format np.uint8 , c'est à dire si ce sont entiers de 0 à 255. Si ce n'est pas le cas (données en flottants entre 0 et 1), nous les convertissons en un format approprié , avec des valeurs dans la plage [0, 255], en multipliant simplement les données par 255. Sinon, on divise les données par la valeur maximale, puis on multiplie par 255 (image/np.max(image))*255).

Centrage des données :

On retire 128 à toutes les données afin de les centrer entre -128 et 127.

Cela permet une manipulation et un affichage correct, quel que soit le format initial des données.

2) Compression:

Initialisation de la matrice de quantification Q :

Nous utilisons la matrice de quantification pour la norme jpg, c'est à dire

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

Calcul de la matrice de passage P :

Giovanni ajoute la Formule avec Safia stp

Boucles de compression :

Pour chaque bloc 8×8 de l'image, nous appliquons la formule de changement de base ##D = PMPT en utilisant la fonction dot de la librairie numpy pour faire les multiplications matricielles. Pour calculer l'inverse de P, nous utilisons l'instruction P.T. Ensuite, nous appliquons la matrice de quantification Q terme à terme et nous conservons seulement la partie entière avec round.

Tout cela est fait dans trois boucles imbriquées. La boucle extérieure itère sur les trois canaux de couleur (rouge, vert, bleu). Pour chaque canal, deux boucles internes parcouruent l'image en hauteur et en largeur avec un pas de 8 pixels. Chaque bloc 8×8 extrait est ensuite transformé en une représentation fréquentielle en appliquant la DCT selon la formule suivante :

ecrire la formule de compression stp

Calcul du taux de compression :

Ensuite, nous avons calculé le taux de compression d'une image en comptant le nombre de coefficients non nuls dans l'image compressée et en le comparant à la taille totale de l'image, exprimée en pourcentage. Pour cela, on utilise la formule : `100 - ((nb_coeff_non_zero / (taille_image[1]*taille_image[0]*3)) * 100)`

##taper en latex Giovanni stpp

Filtre pour les hautes-fréquences:

Nous implémentons maintenant un filtre afin de filtrer les hautes-fréquences

Grâce à deux boucles, nous itérons sur chaque élément du bloc de taille 8×8 qui représente les coefficients après la transformation et la quantification. Nous appliquons ensuite un seuil: la condition `if k + l >= SEUIL` vérifie si la somme des indices k et l dépasse une valeur. Si cette condition est satisfaite, le coefficient $D_{\tilde{k}, l} = 0$ est mis à zéro, c'est -à -dire que nous supprimons les éléments de la matrice si la somme de l'indice i et de l'indice j est supérieure ou égale à la fréquence de coupure.

3) Décompression:

Boucles de décompression :

Dans la phase de décompression, on recrée l'image à partir des coefficients compressés. Pour chaque canal de couleur (R, G, B), on parcourt l'image par blocs de 8×8 , extrait le bloc correspondant de l'image compressée, puis on applique la matrice de quantification Q et la matrice de passage P pour effectuer la transformation inverse.### ecrire la formule de la décompression stp

Décentralisation des données :

Après avoir effectué la transformation, on ajoute une constante de 128 pour revenir à la plage de valeurs appropriée, puis on arrondit les résultats. Les valeurs sont ensuite normalisées entre 0 et 255 et converties en entier pour obtenir l'image décompressée. Finalement, l'image décompressée est affichée.

4) Post-processing:

Calcul de l'erreur :

Afin de comparer la matrice décompressée avec la matrice originale, on calcule la différence élément par élément entre l'image originale et l'image compressée, puis on calcule la norme de Frobenius de cette différence avec `np.linalg.norm(image - compressed)`. Ce résultat est ensuite divisé par la norme de l'image originale (`np.linalg.norm(image)`), ce qui permet d'obtenir une erreur relative.

La valeur obtenue indique à quel point l'image compressée est différente de l'image d'origine : plus la valeur est proche de 0, plus la signifie une compression conserve l'image d'origine.

Recentralisation des données :

Nous re-transformons les valeurs entre -128 et 127 en réels entre 0 et 1 et nous sauvegardons l'image obtenue

IV) Résultats

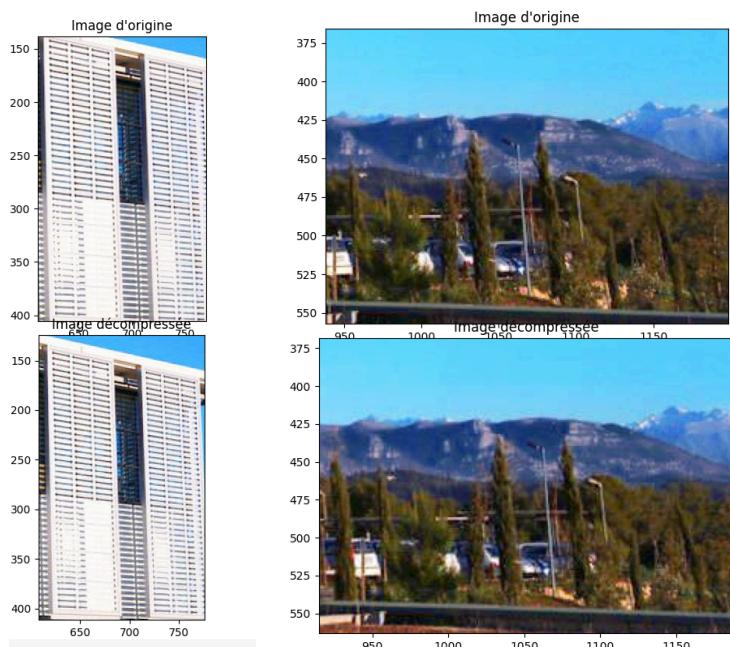
5.1) observation

1. Image Originale vs Décompressée

A première vue, l'image décompressée semble identique à l'image originale :



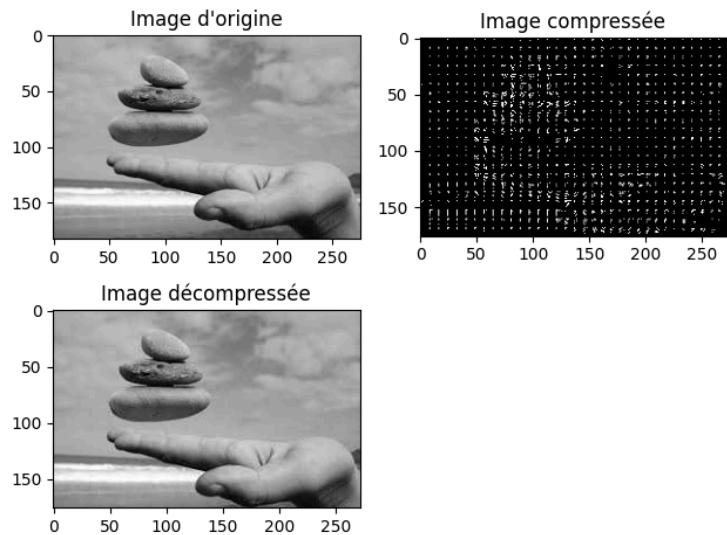
Les différences avec l'image originale sont peu visibles dans l'ensemble, mais en zoomant, nous observons que la compression a perdu de l'information : l'image originale est plus nette, moins pixélisée :



Le taux de compression est en effet de 84%, et l'erreur de 6% donc il y a bien une perte d'information durant le processus de compression.

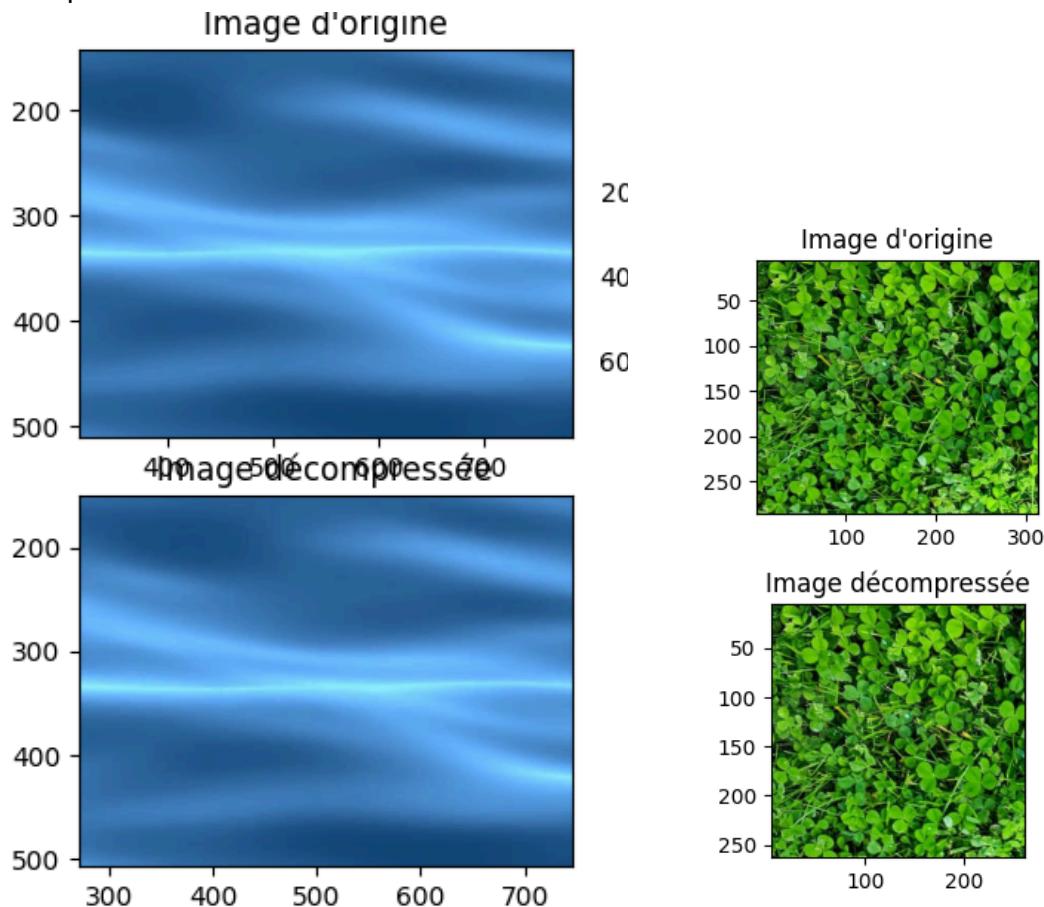
2. image en noir et blanc

Testons maintenant le programme avec une image en noir et blanc : cela fonctionne de la même manière que pour les images avec 3 canaux de couleur car il y a aussi 3 canaux pour ces images. Le taux de compression est de 90 % et l'erreur également de 6%



3. image avec beaucoup de texture vs sans texture

En testant avec des images avec beaucoup de texture et des images moins texturées, nous remarquons une différence dans les résultats pour la matrice compressée puis décompressée : en effet



VAGUES :

taux de compression : 96,5 %
erreur : 1.8008235841989517 %

TREFLES :

taux de compression : 66,2 %
erreur : 13.808168470859528 %

Le taux de compression est plus élevé pour une image avec peu de texture.

4. Image Originale vs Décompressée filtrée

Après avoir implémenté le filtre, nous testons pour différentes valeurs de seuil et différentes images : voici les résultats dans le tableau ci-dessous.

Image	Filtre	Compression	Erreur (%)
Trefles (PNG)	2	95.50	39.78
	6	73.70	20.08
	10	66.28	13.94
Papillon (PNG)	2	95.27	14.19
	6	87.53	4.97
	10	86.79	3.81
Vagues (PNG)	2	96.80	2.12
	6	96.46	1.80
	10	96.46	1.80
Papillon (JPEG)	2	96.61	40.11
	6	89.73	18.08
	10	87.31	11.21

On remarque que plus un filtre est faible plus le nombre de coefficients non nuls est élevé, ce qui se traduit par un taux de compression élevé, mais aussi une erreur importante. C'est le cas du filtre 2. Si on utilise un filtre modéré, par exemple 6, on préserve des détails car l'erreur est plus faible, et on maintient un bon taux de compression.

On observe aussi que les résultats diffèrent selon les caractéristiques de l'image d'origine, notamment avec la complexité et le format:

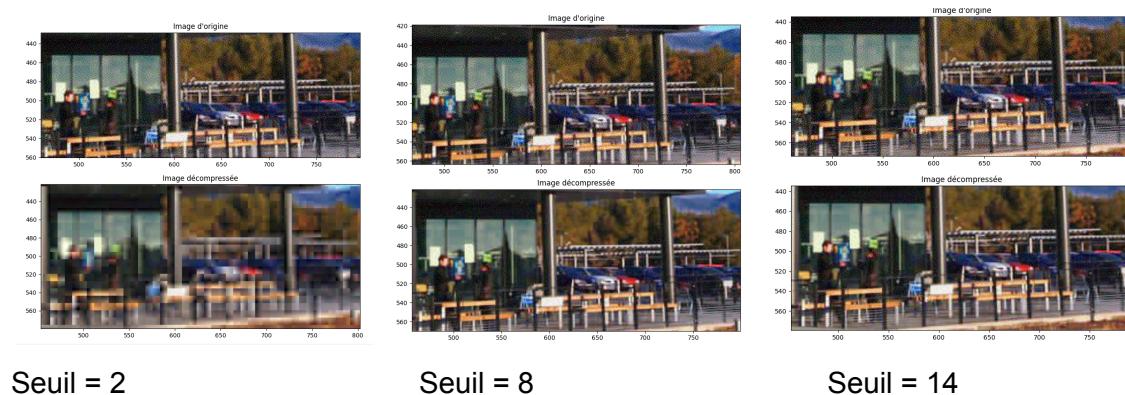
Les images avec des détails fins ou beaucoup de texture, comme les trèfles, présentent une plus grande perte de qualité pour des filtres faibles. Les images avec des motifs simples ou uniformes, et donc pas de hautes fréquences, comme les vagues, maintiennent un faible taux d'erreur même pour des filtres élevés.

Les images PNG préservent mieux la qualité, il vaut mieux utiliser un filtre modéré ou fort. Le format JPEG, déjà compressé, amplifie la perte de qualité pour des filtres faibles. Les filtres forts sont nécessaires.

Pour conclure, les filtres faibles conviennent pour des images simples ou uniformes (comme les vagues) mais provoquent une forte dégradation pour les images complexes (trèfles). Les filtres modérés offrent un bon compromis pour maintenir la qualité tout en assurant une compression significative. Les filtres forts sont nécessaires pour les images avec beaucoup de détails ou des formats déjà compressés, comme JPEG, pour minimiser l'erreur.

5. Image bruitée

Le filtre permet de supprimer les hautes fréquences de bruit. A seuil faible, le bruit reste partiellement visible dans l'image décompressée et à seuil élevé, le bruit est efficacement supprimé, mais la qualité de l'image peut être moins bonne.



Seuil = 2

Seuil = 8

Seuil = 14

6. Format jpeg vs png :

Le code fonctionne pour les deux formats, la seule différence est la manière de stocker les données : dans le format jpg, les données sont comprises entre 0 et 255, en revanche dans le format png, les données sont comprises entre 0 et 1. Nous prenons en compte le format de stockage dans l'initialisation de notre code pour traiter différemment chaque type de format (voir section "Transformation des intensités").

Comme dit précédemment, le choix du filtre dépend aussi du format de l'image

7. Temps d'exécution :

Nous avons rajouté dans le code le nécessaire pour mesurer les temps d'exécution de nos boucles de compression et de décompression : les résultats sont très satisfaisants, le code

s'exécute rapidement. Nous testons le code avec différentes images dans différents formats pour comparer les temps d'exécution :

- image en jpeg : Temps d'exécution de la compression : 0.042 s
- image en png : Temps d'exécution de la compression : 0.11 s
Temps d'exécution de la décompression : 0.048 s
- image avec peu de texture (png) : Temps d'exécution de la compression : 0.662 s
Temps d'exécution de la décompression : 0.292 s
- image avec beaucoup de texture (png) : Temps d'exécution de la compression : 0.810 s
Temps d'exécution de la décompression : 0.259 s
- image en noir et blanc (jpeg) : Temps d'exécution de la compression : 0.032 s
Temps d'exécution de la décompression : 0.017 s

Analyse :

On remarque donc que les JPEG sont plus rapides à traiter que les PNG (pour la compression et la décompression), car les JPEG sont déjà adaptés à une compression avec pertes tandis que les PNG, étant sans perte, contiennent davantage d'informations, ce qui augmente la complexité du traitement.

De plus, les images avec beaucoup de texture prennent légèrement plus de temps à compresser que les images peu texturées.

Enfin, les images en noir et blanc sont les plus rapides à traiter.

On remarque aussi que dans tous ces cas, la décompression est plus rapide que la compression. Cela s'explique par le fait que la compression implique des calculs supplémentaires pour analyser et réduire les données, tandis que la décompression traite moins de données, ce qui la rend plus rapide.

8. Influence de la matrice Q :

On remarque finalement que le taux de compression dépend de la matrice Q : en effet, en multipliant Q par un coefficient k, où k est un entier strictement positif, le taux de compression varie. Nous testons pour k variant de 1 à 5 avec un pas de 1 :

Image utilisée : trèfles

k=1 : taux de compression : 66.15 %
l'erreur est : 13.81 %

k=2 : taux de compression : 76.53 %
l'erreur est : 18.31 %

k=3 : taux de compression : 81.63 %
l'erreur est : 21.01 %

k=4 : taux de compression : 84.95 %

l'erreur est : 23.08 %

k=5 : taux de compression : 87.30 %

l'erreur est : 24.83 %

On déduit de ces résultats que lorsque k augmente :

- le taux de compression augmente : plus k est grand, plus les coefficients de la matrice Q sont élevés, ce qui signifie que les coefficients sont divisés par des valeurs plus grandes. Cela entraîne plus de coefficients égaux à zéro et donc la quantité de données conservées est réduite, c'est pourquoi le taux de compression est élevé.
- L'erreur augmente : plus de détails sont éliminés, ce qui dégrade la qualité de l'image reconstituée.

V) Difficultés rencontrées

Les principales difficultés rencontrées ont été :

- Au départ, nous avions créé une fonction pour découper l'image en blocs de 8x8, et ensuite appliquer la compression et la décompression sur chaque bloc avant de rassembler la matrice. Le problème dans cette technique était que le taux de compression était plutôt faible (autour de 50%). Nous avons donc modifié le code pour faire directement la compression/décompression sur chaque bloc sans avoir à découper toute la matrice et ensuite la reconstituer.
- La gestion des conversions des types données pendant la compression et la décompression d'une image. Nous avions des erreurs dues aux différentes plages de données pour les nombres entiers et flottants.
- Nous n'avions pas créé de matrices distinctes comme compressed, les opérations de transformation (DCT, quantification, suppression des coefficients) se faisaient directement sur la matrice image qui contient les données d'origine. Ainsi, l'image originale était modifiée pendant la compression et nous ne parvenions pas à faire la décompression. Pour éviter cela, nous avons introduit des matrices intermédiaires initialisées avec des zéros.
- Notre image décompréssée s'affichait bien comme l'originale mais avec toutes les couleurs modifiées dans des teintes de rose et jaune. L'erreur venait d'un mauvais type de données et d'une absence de contrainte des valeurs après le décalage de 128. Cela a conduit à des valeurs hors de l'échelle valide, et donc interprétées de manière incorrecte par le logiciel d'affichage.
- Nous avons également rencontré des confusions dans les noms des variables, car chacun de nous ajoutait des parties au code et elles n'étaient pas harmonisées.

Améliorations possibles :

Le programme fonctionne seulement pour les images en format .jpeg ou .png. Une amélioration possible de notre projet serait de trouver une solution pour compresser et décompresser les images avec d'autres formats.

VI) Conclusion

Ce projet nous a donc permis de mettre en évidence que la transformée de Fourier, en particulier la transformée de cosinus discrète, est très utile dans le traitement d'images. Après avoir créé la matrice contenant les données de l'image, nous avons appliqué la transformée de Fourier pour la compresser, puis la transformée de Fourier inverse pour la décompresser.

La compression d'images permet ainsi de stocker l'image de manière plus compacte ou de la transmettre plus rapidement. Nous avons constaté qu'après compression et décompression, l'image obtenue conserve une qualité satisfaisante.