

Rapport de projet

Traitement du signal et d'image : Compression d'images par transformée de Fourier

Ben Khalifa Emna

Costantin Perline

Honakoko Giovanni

Zouarhi Yassmin

14/01/2025

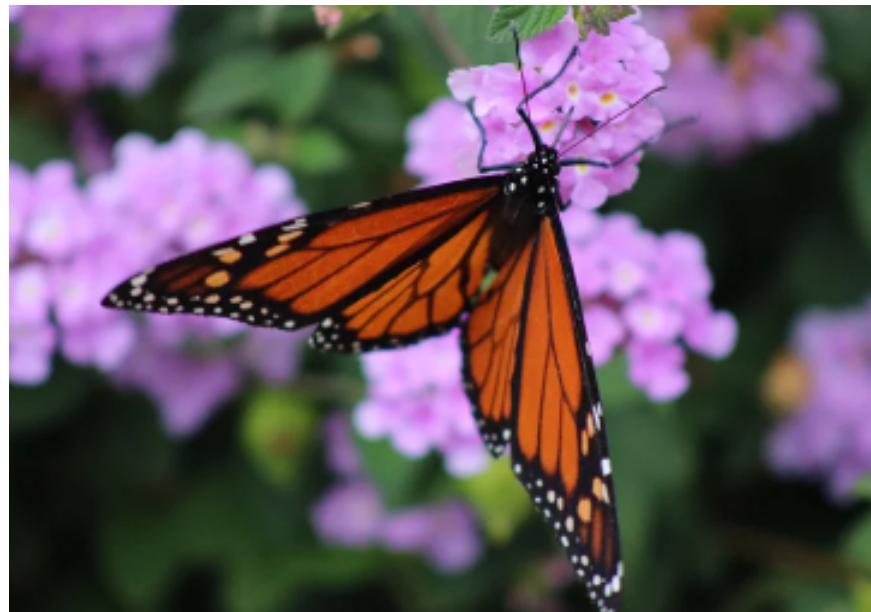
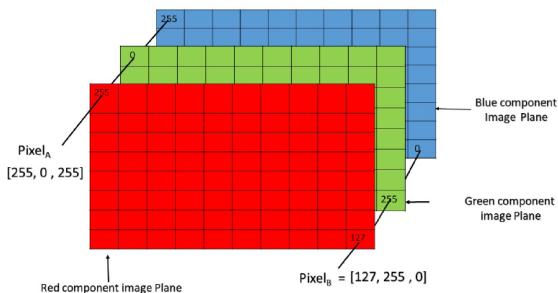


Table des matières

1	Introduction	2
2	Théorie de la Compression d'Image	3
2.1	Changement de base fréquentiel	3
2.2	Calcul de la matrice de passage P	3
2.3	Filtre Passe-Bas	4
3	Algorithm	5
4	Implémentation	5
4.1	Compression	5
4.1.1	Boucles de compression	5
4.2	Filtre Passe-Bas	6
4.3	Décompression :	6
4.3.1	Boucles de décompression	6
4.3.2	Décentralisation des données	6
4.4	Post-processing	6
4.4.1	Calcul de l'erreur	6
4.4.2	Re-centralisation des données	6
4.5	Calcul du taux de compression	6
5	Résultats	7
5.1	Application du Filtre Passe-Bas	9
5.1.1	Application du filtre sur une image bruitée	10
5.2	Temps d'exécution	10
5.3	Influence de la Matrice de Quantification	11
6	Difficultés rencontrées	12
7	Conclusion	12
8	Annexes	13
8.1	Plan de Travail	13
8.2	Carnet de Bord	13

1 Introduction

Dans le cadre de ce projet TRAITEMENT DU SIGNAL ET DE L'IMAGE, nous avons travaillé sur la représentation d'images numériques. On considère ici la représentation **RGB** (*Rouge, Vert, Bleu*) où l'image est décomposée en trois composantes de couleur.



Une image de $n_x \times n_y$ pixels se représente 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.

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

2 Théorie de la Compression d'Image

Le terme d'**image numérique** désigne, dans son sens le plus général, toute image qui a été acquise, traitée et sauvegardée sous une forme codée représentable par des nombres (*valeurs numériques*).

L'objectif est donc ici d'utiliser cet outil 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 par la théorie de Fourier en une combinaison linéaire de fonction sin et cos. Ainsi on peut également décomposer une image dans une base {sin, cos}.

2.1 Changement de base fréquentiel

Ce principe est transcrit par la **Transformée de Fourier Discrète** (DFT). On se restreint à la **Transformée de Cosinus Discrète** (DCT), en faisant en sorte lors du processus de symétrisation que notre image soit paire. La DCT est calculée sur des blocs de 8×8 pixels et chaque bloc est représenté par une matrice $M = M(i, j)_{0 \leq i, j \leq 7}$ de dimension 8×8 , tels que :

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

avec :

$$C_k = \begin{cases} 1, & \forall k \neq 0 \\ \frac{1}{\sqrt{2}}, & \text{sinon} \end{cases}, \quad (\text{resp. } C_l)$$

La formule (1) retranscrit en vérité le changement de base :

$$D = P D^t P \quad (2)$$

où P est orthogonale et contient les coefficients de la DCT.

2.2 Calcul de la matrice de passage P

On a cette relation qui définit le changement de base :

$$\begin{pmatrix} D_{0,0} & \dots & D_{0,7} \\ \vdots & \ddots & \vdots \\ D_{7,0} & \dots & D_{7,7} \end{pmatrix} = \begin{pmatrix} P_{0,0} & \dots & P_{0,7} \\ \vdots & \ddots & \vdots \\ P_{7,0} & \dots & P_{7,7} \end{pmatrix} \begin{pmatrix} M_{0,0} & \dots & M_{0,7} \\ \vdots & \ddots & \vdots \\ M_{7,0} & \dots & M_{7,7} \end{pmatrix} \begin{pmatrix} P_{0,0} & \dots & P_{7,0} \\ \vdots & \ddots & \vdots \\ P_{0,7} & \dots & P_{7,7} \end{pmatrix}$$

Dans l'objectif de reconstruire cette formule, on pose :

$$B = PM \iff B = \begin{pmatrix} P_{0,0} & \dots & P_{0,7} \\ \vdots & \ddots & \vdots \\ P_{7,0} & \dots & P_{7,7} \end{pmatrix} \begin{pmatrix} M_{0,0} & \dots & M_{0,7} \\ \vdots & \ddots & \vdots \\ M_{7,0} & \dots & M_{7,7} \end{pmatrix} \quad (3)$$

Ainsi on a par définition du produit matricielle :

$$\forall (i, j, r) \in \llbracket 0, 7 \rrbracket^3, \quad B_{i,j} = \sum_{r=0}^7 P_{i,r} M_{r,j}$$

En multipliant par la tP à droite de part et d'autre de (2), on a :

$$\begin{pmatrix} D_{0,0} & \dots & D_{0,7} \\ \vdots & \ddots & \vdots \\ D_{7,0} & \dots & D_{7,7} \end{pmatrix} = \begin{pmatrix} B_{0,0} & \dots & B_{0,7} \\ \vdots & \ddots & \vdots \\ B_{7,0} & \dots & B_{7,7} \end{pmatrix} \begin{pmatrix} P_{0,0} & \dots & P_{7,0} \\ \vdots & \ddots & \vdots \\ P_{0,7} & \dots & P_{7,7} \end{pmatrix}$$

On remarque que pour chaque coefficient on a la relation :

$$\forall (i, j, r, q) \in \llbracket 0, 7 \rrbracket^4, \quad D_{i,j} = \sum_{q=0}^7 B_{i,r} P_{j,q}$$

Donc :

$$\begin{aligned} \forall (i, j, r, q) \in \llbracket 0, 7 \rrbracket, \quad D_{i,j} &= \sum_{q=0}^7 \sum_{r=0}^7 P_{i,r} M_{r,q} P_{j,b} \\ D_{k,l} &= \sum_{q=0}^7 \sum_{r=0}^7 P_{k,r} M_{r,q} P_{l,q} \end{aligned}$$

Par identification avec (1), il en résulte :

$$\begin{cases} P_{k,i} &= \frac{1}{2} C_k \cos\left(\frac{(2i+1)k\pi}{16}\right) \\ P_{l,j} &= \frac{1}{2} C_l \cos\left(\frac{(2j+1)l\pi}{16}\right) \end{cases}$$

Finalement on en déduit :

$$P_{i,j} = \frac{1}{2} C_i \cos\left(\frac{(2j+1)i\pi}{16}\right) \quad (4)$$

■

La matrice D contient les amplitudes des fréquences :

- Les coefficients situés en haut à gauche correspondent aux **basses fréquences**.
- Les coefficients en bas à droite représentent les **hautes fréquences**.

Une fois que le changement de base est réalisé On divise terme par terme D par Q , où Q est une **matrice de quantification**. Puis on prends la partie entière pour chaque coefficient, ce qui élimine les hautes fréquences.



Ce protocole permet une compression efficace tout en conservant l'essentiel de l'information visuelle.

LA DÉCOMPRESSION DE L'IMAGE : Il suffit d'appliquer le protocole de DCT inverse.

2.3 Filtre Passe-Bas

Certaines images sont **bruitées**, ce qui signifie qu'elles contiennent trop de haute-fréquence (*souvent inutile*). D'où l'intérêt des filtres, notamment du **filtre passe-bas** qui permet en définissant une fréquence de coupure ω_c , d'annihiler les hautes-fréquences désirées. Ce qui peut avoir pour effet d'éliminer le dit *bruit*.

3 Algorithme

INITIALISATION :

- Lire l'image.
- Tronquer l'image à des multiples de 8 en x et y .
- 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 P (*de la DCT2*).

COMPRESSION :

Pour chaque bloc 8×8 de l'image :

- Appliquer le changement de base $D = PM^tP$.
- 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 coefficients sur les dernières lignes et colonnes dans la matrice.
- Stocker ces nouveaux blocs 8×8 dans la matrice compressée/débruitée.
- Compter le nombre de coefficients non nuls pour obtenir le taux de compression.

DÉCOMPRESSION :

- Multiplier par la matrice Q terme à terme
- Appliquer la transformée inverse tPDP
- Ré-assembler la matrice décompressée/débruitée

POST-PROCESSING :

- Comparer cette matrice avec la matrice originale (*en norme de Frobenius*).
- Re-transformer les valeurs entre -128 et 127 en réels entre 0 et 1 et sauver l'image (*pour la comparer visuellement*).

4 Implémentation

4.1 Compression

4.1.1 Boucles de compression

Pour chaque bloc 8×8 de l'image, nous appliquons la formule de changement de base $D = PM^tP$ en utilisant la fonction `dot` de la librairie `numpy` pour obtenir l'expression de chaque coefficient de (1).

Pour calculer la transposée 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 parcourtent 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 :

4.2 Filtre Passe-Bas

Nous implémentons maintenant un filtre afin de réduire les hautes-fréquences grâce à deux boucles. Nous itérons sur chaque élément du bloc 8×8 .

Nous appliquons ensuite un seuil ω_c .

La condition `if k + 1 >= SEUIL` vérifie si la somme des indices k et l dépasse la valeur de notre fréquence de coupure. Si cette condition est satisfaite, le coefficient `D_tilde[k, l]` 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 ω_c .

4.3 Décompression :

4.3.1 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 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 :

$$M = {}^t P D P$$

4.3.2 Décentralisation des données

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

4.4 Post-processing

4.4.1 Calcul de l'erreur

Afin comparer la matrice décompressée avec la matrice originale, on calcule la différence en norme matricielle avec la norme de Frobenius. Puis en divisant par la norme de l'image originale `np.linalg.norm(image)`, on obtient l'erreur relative.

$$\text{erreur_totale} = (\text{np.linalg.norm}((\text{image} + 128) - \text{decompressed})) / (\text{np.linalg.norm}(\text{image})) * 100$$

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

4.4.2 Re-centralisation des données

Nous re-transformons les valeurs dans $[-128, 127]$ en réels contenu dans $[0, 1]$ et nous sauvegardons l'image obtenue

4.5 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 :

$$\text{taux de compression} = 100 - \left(\frac{\text{nombre de coefficient non-nuls}}{x \times y \times 3} \right) \times 100$$

5 Résultats

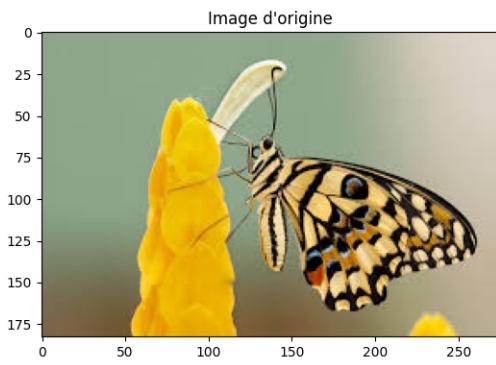


FIGURE 1 – Papillon (*originale*)

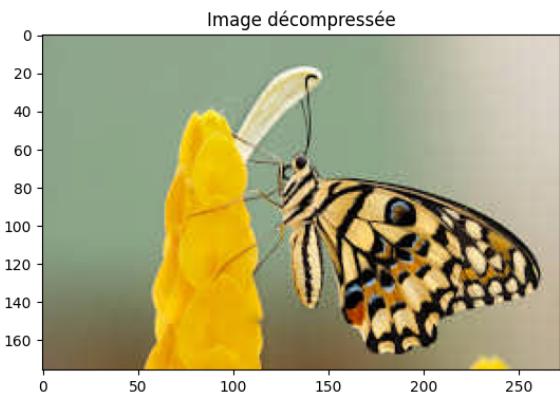


FIGURE 2 – Papillon (*décompressée*)

Les différences avec l'image originale et la décompressée sont peu visibles dans l'ensemble, mais en zoomant on remarque belle et bien une pixellisation sur le format décompressé.

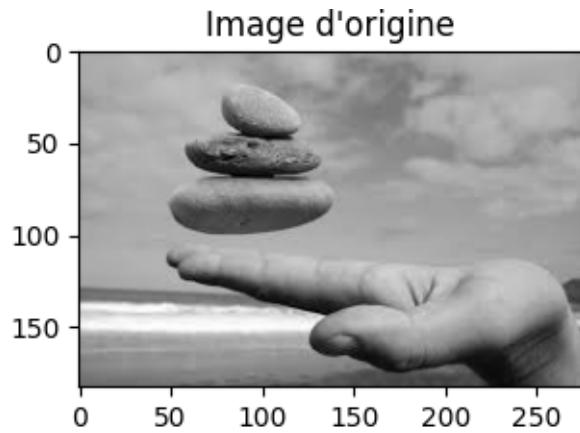


FIGURE 3 – Pierres (*originale*)

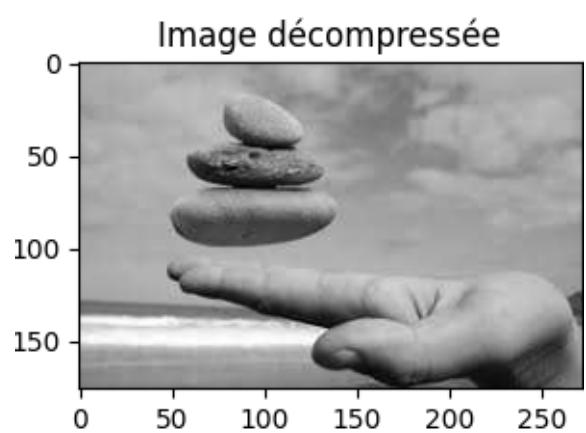


FIGURE 4 – Pierres (*décompressée*)

Là encore le constat est identique. On verra mieux la pixellisation sur les figures 5 et 6.

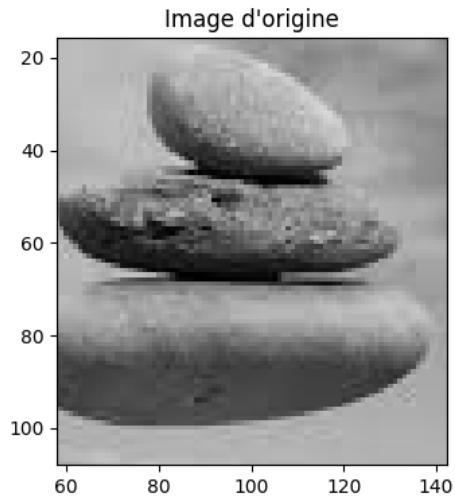


FIGURE 5 – Pierres (*originale zoomée*)

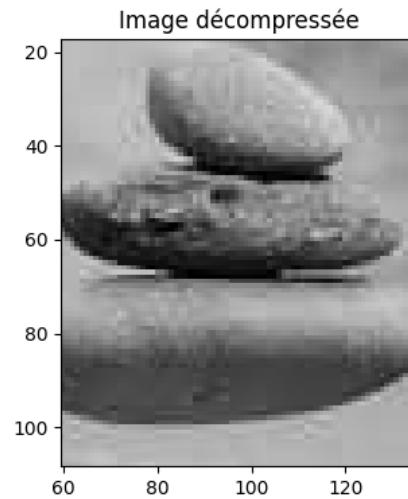


FIGURE 6 – Pierres (*décompressée zoomée*)

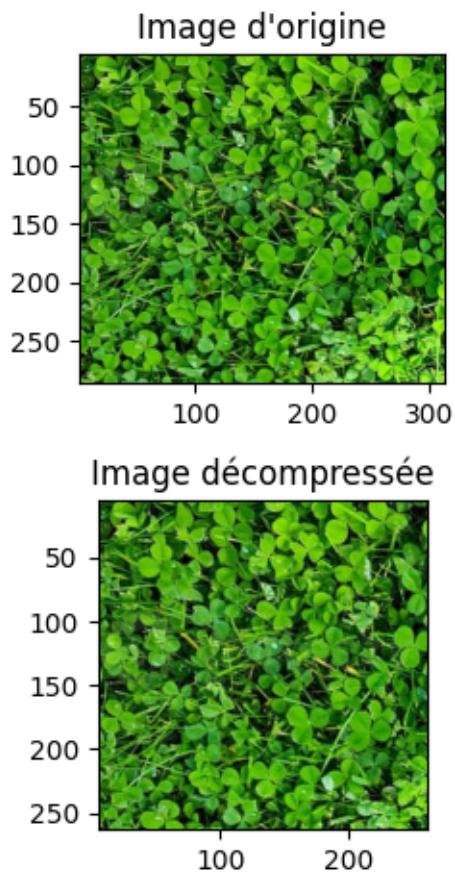


FIGURE 7 – Trèfles

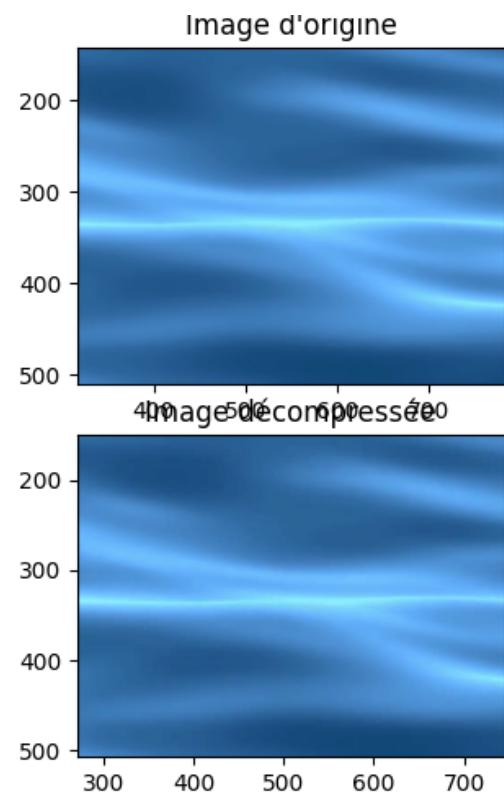


FIGURE 8 – Vagues

En comparant une image très texturée (cf.Fig. 7) à une image avec peu de texture (cf.Fig. 8). On fait les observations suivantes :

TRÈFLES :

- Taux de compression : 66,2%
- Erreur : $\simeq 13.8\%$

VAGUES :

- Taux de compression : 96,5%
- Erreur : $\simeq 1,8\%$



Le taux de compression est plus élevé pour les vagues et l'erreur y est également plus faible. Ce qui est l'exact opposé pour une image très texturée comme les trèfles.

5.1 Application du Filtre Passe-Bas

Après avoir implémenté notre filtre, nous l'avons testé sur différentes images avec différents seuils. Voici le tableau récapitulatif de ces applications :

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

Soit ω_c la **fréquence de coupure du filtre**.

Si un filtre est *faible*, alors le nombre de coefficients non nuls est élevé. Cela se traduit par un taux de compression et une erreur élevée. Par exemple pour $\omega_c = 2$.

Si on utilise un filtre *modéré*, par exemple $\omega_c = 6$.

Alors on conserve les détails de l'image, puisque l'erreur est plus faible tout en maintenant un taux de compression correcte.

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 (cf.Fig 7), présentent une plus grande perte de qualité pour des filtres faibles.
- Les images avec des motifs simples ou uniformes, donc avec peu de haute fréquence (cf.Fig. 8) ; maintiennent un faible taux d'erreur pour tout ω_c .

Les images **PNG** préservent mieux la qualité, il est préférable d'utiliser un filtre modéré, voir fort.

Le format **JPEG**, étant déjà compressé, amplifie la perte de qualité pour des filtres faibles. Les filtres forts sont donc à privilégier.

Pour conclure, les filtres faibles conviennent pour des images simples ou uniformes (cf.Fig. 8) mais provoquent une forte dégradation pour les images complexes (cf.Fig. 7).



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 le JPEG, afin d'en minimiser l'erreur.

5.1.1 Application du filtre sur une image bruitée

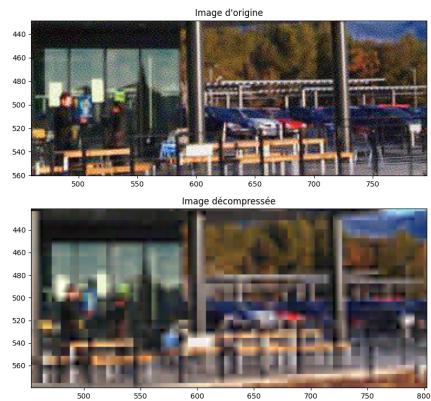


FIGURE 9 – $\omega_c = 2$



FIGURE 10 – $\omega_c = 8$

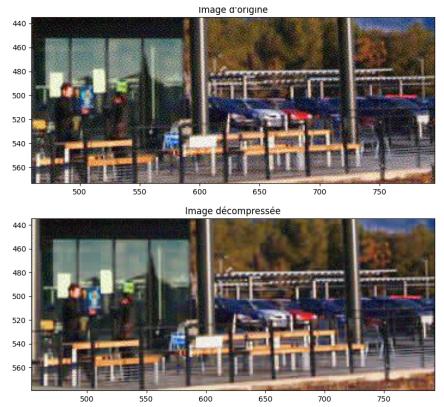


FIGURE 11 – $\omega_c = 14$

Le filtre permet de supprimer les hautes fréquences où se situe le bruit. A seuil faible, le bruit reste partiellement visible dans l'image décompressée.

Pour un seuil élevé, le bruit est efficacement supprimé, mais la qualité de l'image peut en pâtrir.

Le code fonctionne pour les images sous format PNG ou JPEG, la seule différence se trouve dans la manière de stocker les données.

- Pour le format **JPEG**, les données sont comprises entre $[0, 255]$.
- Pour le format **PNG**, les données sont comprises dans $[0, 1]$.

Nous prenons en compte le format de stockage dans l'initialisation de notre code pour traiter différemment chaque type de format (cf. INITIALISATION DE L'ALGORITHME).

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

5.2 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'exécutions :

- **Image en JPEG :**
 - Temps d'exécution de la compression : 0.042 s
 - Temps d'exécution de la décompression : 0.021 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 (ONG) :**
 - 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 images sous format JPEG sont plus rapides à traiter que celles en PNG (*pour la compression et la décompression*), car les JPEG sont déjà adaptés à une compression avec pertes.

Tandis que les images en format PNG, étant sans perte, contiennent davantage d'information 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, là où la décompression traite moins de donnée ce qui la rend plus rapide.

5.3 Influence de la Matrice de Quantification

On remarque finalement que le taux de compression dépend de la matrice Q : en effet, en multipliant Q par un coefficient k , où $k \in \mathbb{N}^*$, le taux de compression varie.

Pour tout $k \in [1, 5]$ sur l'image des trèfles (cf.Fig. 7), on a :

- $k = 1$:
 - Taux de compression : 66.15%
 - Erreur : 13.81%
- $k = 2$:
 - Taux de compression : 76.53 %
 - Erreur : 18.31%
- $k = 3$:
 - Taux de compression : 81.63%
 - Erreur : 21.01%
- $k = 4$:
 - Taux de compression : 84.95%
 - Erreur : 23.08%
- $k = 5$:
 - Taux de compression : 87.30%
 - Erreur : 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 le sont aussi, ce qui signifie que les coefficients sont divisés par des valeurs plus grandes. Cela entraîne plus de coefficients nuls 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.

6 Difficultés rencontrées

Les principales difficultés rencontrées ont été :

- Au départ, nous avions créé une fonction pour découper l'image en blocs de 8×8 , 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 $\simeq 50\%$. 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 contenait les données d'origines. 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 décompressée s'affichait bien comme l'originale mais avec toutes les couleurs modifiées dans des teintes de rose et de 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, ce qui fut interprété 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.

7 Conclusion

Ce projet a permis de mettre en exergue l'utilité de la transformée de Fourier. Plus particulièrement l'accent était mis sur la performance de la DCT, permettant une compression d'image plus optimale. En effet le stockage de l'image s'en retrouve plus compact. Nous avons constaté qu'après compression et décompression, l'image obtenue conserve une qualité satisfaisante. Les différences avec l'image originale sont peu visibles dans l'ensemble, néanmoins, en zoomant on peut remarquer une pixellisation due à la perte de donnée lors de la compression.

8 Annexes

8.1 Plan de Travail

Etapes prévues	LUNDI	MARDI	MERCREDI	JEUDI
Initialisation				
Compression				
Décompression				
Post-processing				
Implémentation				
Résolution des bugs				
Clarification du code				
Tests				
Interprétation des résultats				
Rédaction du rapport				
Préparation de la présentation				

8.2 Carnet de Bord

Carnet de bord : Traitement de signal

Groupe: EGPY

Membres:

- Ben Khalifa Emna
- Costantin Perline
- Honakoko Giovanni
- Zouarhi Yassmin

Jour 1 : 13/01/2025

Ben Khalifa Emna :

- Initialisation
- Une première méthode (division de blocs) compression et décompression pour un canal
- Début deuxième méthode

Costantin Perline :

- Codage de la méthode de compression en divisant les blocs 8x8 dès le départ puis en reconstruisant de la matrice pour un canal
- Calcul du taux de compression

Honakoko Giovanni :

- Définition de la matrice de passage en fréquentielle P (*DCT-II*)
- Participation pour l'implémentation de la compression pour un bloc 8x8
- Participation pour l'implémentation de la décompression pour un bloc 8x8
- Implémentation complète du filtre passe-bas, mais avec erreur
- Implémentation complète du Post-Processing, mais avec erreur

Zouarhi Yassmin :

- Initialisation
- Implémenter des lignes de codes pour la méthode de compression par blocs 8×8 en essayant de ne pas définir des fonctions
- Définir la matrice de passage en fréquentiel P (DCT)

Jour 2 : 14/01/2025

Ben Khalifa Emna :

- Fin deuxième méthode
- Implémentation avec tous les canaux + correction des bugs

Costantin Perline :

- Réécriture du code sans les fonctions
- Amélioration du code pour avoir un meilleur taux de compression
- Correction des bugs

Honakoko Giovanni :

- Correction du Post-Processing et du filtre passe-bas.
- Début du rapport en LateX

Zouarhi Yassmin :

- Compléter l'implémentation en ajoutant la méthode de décompression pour un bloc 8x8
- Essayer d'optimiser le code de façon à diminuer le coût et augmenter le taux de compression

Jour 3 : 15/01/2025

Ben Khalifa Emna :

- Rédaction du rapport
- Clarification du code, ajout de commentaires

Costantin Perline :

- Rédaction du rapport
- Codage de la partie filtrage des hautes fréquences
- Tests du filtre avec différentes images et différentes fréquences

Honakoko Giovanni :

- Codage de la partie filtrage des hautes fréquences
- Rédaction du rapport en LateX.

Zouarhi Yassmin :

- Clarification du code
- Rédaction du rapport
- Présentation (introduction : contexte, intérêt, présentation en pixel)

Jour 4 : 16/01/2025

Ben Khalifa Emna :

- Rédaction du code final avec commentaire
- Création du support pour la soutenance

Costantin Perline :

- Rédaction du rapport
- Tests pour les images bruitées
- Préparation de l'oral

Honakoko Giovanni :

- Rédaction du rapport en Latex

Zouarhi Yassmin :

- Participation à la rédaction du rapport
- Préparation de la présentation avec Emna
- Préparation de l'oral

Jour 5 : 17/01/2025

Soutenance devant le jury