

Compte rendu : Compression et traitement d'images par transformée de Fourier

Matthieu Keruzoret, Petru Piculescu, Camille Veillon, Zineb Ziad

19 janvier 2025

Introduction et objectifs

Nous souhaitons, à travers un programme Python, compresser et décompresser des images à l'aide de la transformée de Fourier. Pour cela, il faut savoir qu'une image numérique peut être représentée par un tableau multidimensionnel contenant les composantes des couleurs RGB. En effet, un pixel est divisé en trois canaux rouge, vert et bleu qui, selon l'intensité des canaux, représenteront une couleur spécifique.

Aussi, l'utilisation de la transformée de Fourier est un outil essentiel aux traitements des images. En effet, en utilisant les propriétés de symétrie et de périodisation d'une image, nous pouvons rendre cette dernière infinie dans les deux directions, périodique et paire. Alors elle pourra être représentée avec une base de fonctions cosinus. Nous utilisons alors une transformée de cosinus discrète (DCT). Dans notre cas, nous allons utiliser la deuxième variante de cette transformée (DCT-2).

L'objectif de ce projet est ainsi de créer un programme capable d'appliquer la transformée de Fourier pour extraire les informations sur les hautes fréquences puis d'appliquer la transformée de Fourier inverse. Ce procédé est la compression/décompression d'une image. Pour cela, nous avons besoin de découper l'image en blocs de 8×8 pour les traiter individuellement. Le but de la compression d'une image est de minimiser les données à stocker pour sauvegarder ou encore partager cette image plus simplement. Il sera également pertinent de connaître le taux de compression de l'image ainsi que le taux d'erreur obtenus après ce processus pour estimer le taux de perte de données car pour certaines images, il est primordial de garder le maximum d'informations sur la photo (comme pour les imageries médicales par exemple).

Table des matières

1 Plan de travail et carnet de bord	3
1.1 Plan de Travail	3
1.2 Carnet de bord	5
2 Algorithme	6
2.1 Initialisation	6
2.2 Transformée de cosinus discrète (DCT-2) d'une image	6
2.3 Compression	7
2.3.1 Compression par quantification	7
2.3.2 Compression par filtre	7
2.4 Décompression	8
2.5 Post-Processing	8
3 Résultats et interprétations	9
3.1 Compression par quantification	10
3.2 Compression par filtre	11
3.3 Essais avec des matrices de quantification différentes et en appliquant un filtre fixé . . .	14
3.4 Essais avec la matrice de quantification Q et en appliquant des filtres différents	16
3.5 Compression avec la matrice de quantification Q et en appliquant des filtres différents pour une image bruitée	17
4 Conclusions	18

Chapitre 1

Plan de travail et carnet de bord

1.1 Plan de Travail

Préparation du projet

Organisation

- Organisation et structuration du groupe.
- Désignation d'un chef de groupe différent tous les jours.

Familiarisation avec les outils

- Explorer les fonctions Python pour manipuler les images grâce à la bibliothèque `matplotlib`.
- Tester des opérations de base sur des tableaux avec `NumPy`.

Implémentation de l'algorithme

Initialisation

1. Chargement et prétraitement de l'image :
 - Charger une image et ajuster ses dimensions pour les rendre multiples de 8.
 - Sélectionner un seul canal (exemple : rouge) pour obtenir une matrice 2D.
 - On vérifie si l'image donnée est en format `.jpg` ou en format `.png` (différence de traitement pour les valeurs RGB).
 - Convertir les intensités lumineuses en entiers entre 0 et 255, puis centrer les valeurs entre -128 et 127.
2. Définir la matrice de transformation DCT :
 - Construire la matrice orthogonale pour effectuer la DCT-II.

Compression

1. Diviser l'image en blocs.
2. Pour chaque bloc :
 - Calculer la DCT.
 - Quantifier les coefficients :
 - Diviser chaque élément par les termes correspondants de la matrice.
 - Arrondir les valeurs obtenues à la partie entière.
 - Filtrer les hautes fréquences en mettant à zéro certains coefficients selon une règle prédéfinie.
3. Stocker les blocs compressés dans une nouvelle matrice et calculer le taux de compression.

Décompression

1. Pour chaque bloc compressé :
 - Multiplier chaque élément par les termes correspondants de la matrice.
 - Appliquer la transformée inverse.
2. Réassembler l'image en regroupant les blocs décompressés.

Post-traitement

- Convertir les valeurs de la matrice finale en réels entre 0 et 1.
- Sauvegarder et visualiser l'image décompressée pour comparaison avec l'originale.

Études et Analyses

Analyse de la compression

- Calculer la perte en norme relative entre l'image originale et l'image compressée.
- Tester différentes matrices de quantification pour analyser leur impact sur la qualité et le taux de compression.

Filtrage des hautes fréquences

- Remplacer la quantification par une simple troncature des hautes fréquences (éléments de D pour lesquels $|D_{ij}| < \varepsilon$) où ε correspond à la valeur du filtre.
- Expérimenter avec plusieurs valeurs de ε et évaluer les résultats.

Comparaison avec des librairies existantes

- Utiliser une librairie comme `scipy.fft` pour effectuer une DCT standard.
- Comparer les performances (qualité des résultats et temps d'exécution).

Comparaison avec une compression combinée avec matrice de quantification et filtre haute fréquence

- Combiner les compressions.
- Adapter la décompression.
- Analyser les résultats.

Documentation et Rendu Final

Rapport

Rédiger un rapport expliquant :

- Les étapes de l'algorithme.
- Les résultats obtenus (visuels et quantitatifs).
- Les comparaisons effectuées.

Code

- Fournir des codes Python bien commentés.

Résultats

- Inclure les visualisations des images compressées et décompressées.

Extensions Possibles

1. Compression adaptative : Adapter la matrice selon les caractéristiques locales de l'image.
2. Compression vidéo : Adapter l'algorithme pour traiter des séquences d'images.

1.2 Carnet de bord

Voici comment nous avons organisé cette semaine :

Lundi

Nous avons réalisé :

- Développement de la partie **Initialisation** (Matthieu) ;
- Définir la matrice de passage P en fréquentiel de la *DCT-2*. (Camille)
- Développement de la partie **Compression**. (Petru, Camille)
- Développement de la partie **Décompression**. (Zineb)
- Élaboration du plan de travail. (Camille)

Mardi

- Création et rédaction du rapport. (Matthieu)
- Création de la présentation PowerPoint. (Zineb)
- Recherches sur les matrices de décompression. (Camille)
- Calcul de l'erreur pour comparer la matrice après décompression avec la matrice originale (en norme L^2 relative par exemple). (Matthieu)
- Optimisation du filtrage haute fréquence. (Camille, Petru)
- Développement de la partie **Décompression**. (Matthieu, Zineb)
- Réalisation des comparaisons sur les tailles mémoires des fichiers avant et après compression. (Petru)
- Essais des algorithmes de compression sur les vidéos. (Petru)

Mercredi

- Rédaction du rapport. (Matthieu)
- Réalisation de la présentation PowerPoint. (Zineb)
- Optimisation des différentes compressions. (Camille, Petru)

Jeudi

- Rédaction du rapport. (Matthieu)
- Réalisation de la présentation PowerPoint. (Zineb)
- Réalisations de la partie **Résultats et interprétations**. (Petru, Camille)

Vendredi

- Présentation orale

Chapitre 2

Algorithm

2.1 Initialisation

Lors de l'initialisation, nous ouvrons une image enregistrée pour tester notre compression et notre décompression, afin de voir les différences entre l'image au début et celle affichée en fin de décompression. Ouvrir ce fichier reviens à obtenir un tableau `numpy` dont les composantes sont des triplets pour les valeurs RGB d'un pixel. Pour cela, nous utilisons la fonction `imread` de la librairie `matplotlib.pyplot`. Nous tronquons ensuite l'image à des multiples de 8 en x et y . Pour redimensionner la matrice, nous effectuons simplement un modulo 8 du nombre de lignes et de colonnes et nous redéfinissons la matrice avec ces nouvelles valeurs. Pour finir, on centre les valeurs de l'image pour chaque canal de couleur (entre -128 et 127). Finalement, nous convertissons toutes ces valeurs flottantes en des valeurs entières.

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

Afin de simplifier les calculs, nous allons travailler sur des blocs 8×8 de l'image. Si un bloc 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-2 de cette image s'écrit alors en dimension 2, pour $0 \leq k, l \leq 7$:

$$D_{k,l} = \frac{1}{4} C_k C_l \sum_{i=0}^7 \sum_{j=0}^7 M_{i,j} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cos\left(\frac{(2j+1)l\pi}{16}\right) \quad (2.1)$$

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-2 permet d'obtenir un opérateur orthogonal. La transformation transposée donne alors la transformée inverse. La transformation décrite dans la formule (1.1) s'apparente simplement à un changement de base orthonormée (passage en mode fréquentiel), et peut donc se réécrire sous la forme :

$$D = P M P^T$$

où P est une matrice orthogonale contenant les coefficients de la DCT-2. Dans notre cas, on crée cette matrice inverse par la formule $P.T$.

Dans notre code, on implémente P à l'aide de cette formule :

$$P_{k,n} = \sqrt{\frac{2}{N}} \cdot \begin{cases} \frac{1}{\sqrt{2}}, & \text{si } k = 0, \\ \cos\left(\frac{\pi(2n+1)k}{2N}\right), & \text{si } k > 0 \end{cases}$$

où N est la taille des blocs, en l'occurrence ici $N = 8$.

Les entrées de la matrice D donnent la fréquence des changements d'intensité lumineuse. Les coefficients dans le coin supérieur gauche de la matrice D correspondent aux petites fréquences en x et en

y (variations lentes), alors que les coefficients en bas à droite correspondent aux variations d'intensité aux hautes fréquences (variations rapides).

2.3 Compression

2.3.1 Compression par quantification

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 :

$$Q = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 13 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \quad (2.2)$$

À noter que nous allons tester notre code avec d'autres matrices Q (nous multiplierons la matrice (2.2) avec un coefficient $k \geq 1$) dans la partie **Résultats et interprétation**.

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 coefficients 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%).

À la fin de la compression, nous calculons le nombre de coefficients non nuls grâce à la fonction de la bibliothèque `numpy nonzero` que nous appliquons à chaque matrice de filtre de couleur. Nous pouvons ainsi calculer le taux de compression en pourcentage grâce à :

$$\text{taux de compression} = 1 - \frac{\text{nombre de coefficients non nuls}}{\text{taille des matrices de couleurs} \times 3} \times 100$$

2.3.2 Compression par filtre

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.

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

2.5 Post-Processing

Dans la partie Post-Processing, on compare la matrice obtenue à l'ouverture du fichier de l'image avec celle obtenue après les étapes de compression et de décompression. On calcule la norme de la différence des deux matrices (calcul de la distance) pour avoir le pourcentage d'erreur lors de la compression à l'aide de la fonction `numpy.linalg.norm` de la bibliothèque `numpy`.

Pour finir, nous pouvons sauvegarder les images obtenues grâce à la fonction `plt.imsave` pour sauvegarder en format .png et la fonction `.save` pour sauvegarder en format .jpg.

On pourra comparer les tailles de fichier afin de voir si la compression réduit la taille du fichier ou non.

Chapitre 3

Résultats et interprétations

Pour comparer nos différentes méthodes, nous allons d'abord utiliser ses différentes images :



FIGURE 3.1 – Trèfles - originale

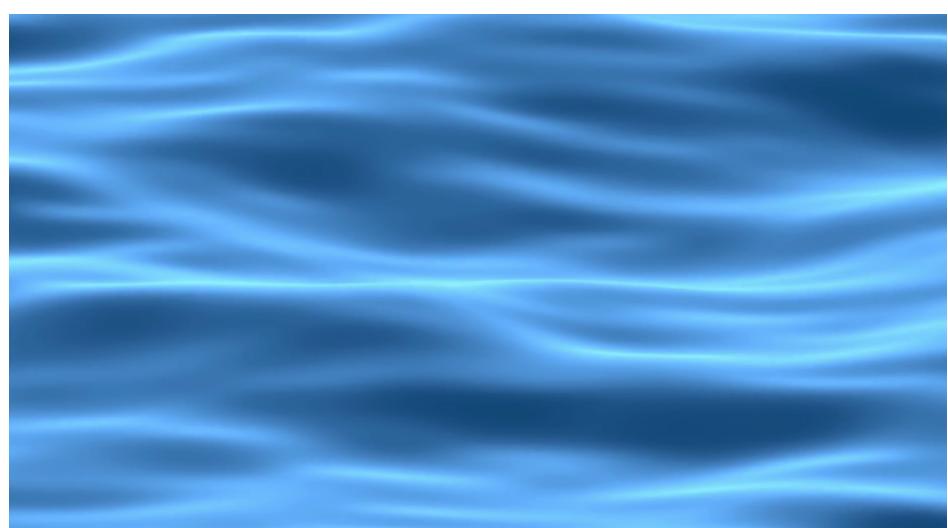


FIGURE 3.2 – Vagues - originale

3.1 Compression par quantification

Nous avons appliqué notre algorithme de compression par quantification (avec la matrice Q (2.2)) et de décompression aux deux images et nous avons obtenu les résultats suivants :

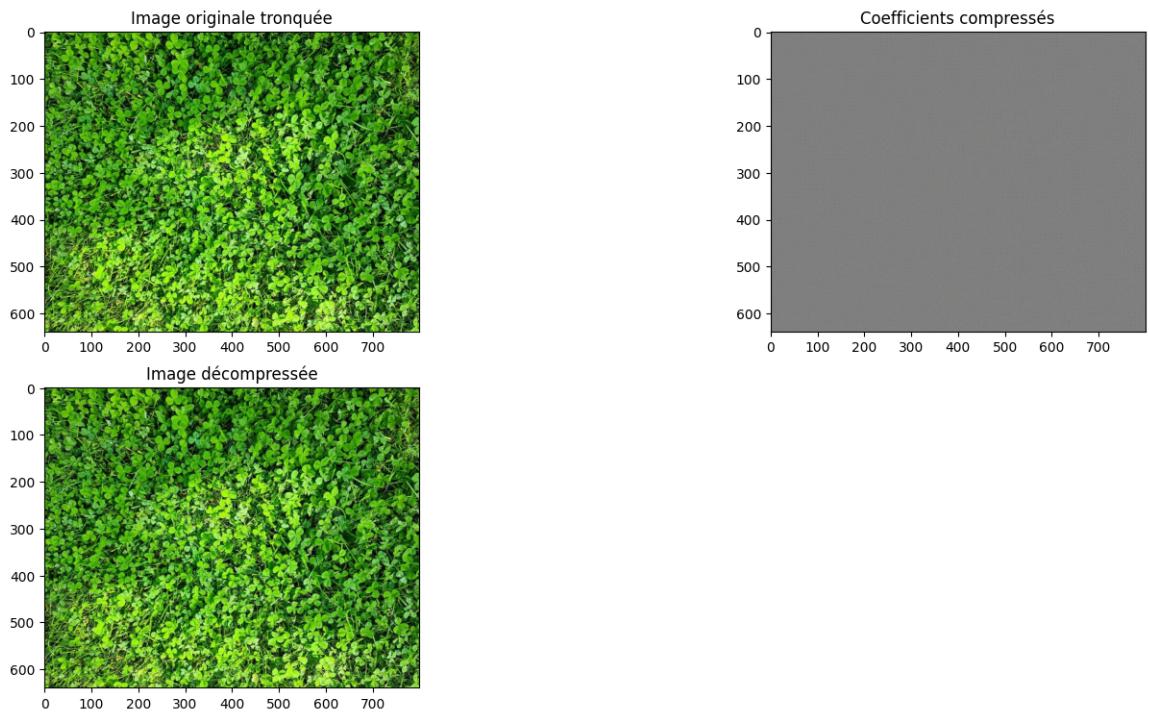


FIGURE 3.3 – Trèfles - image compressée/décompressée

Pour la figure 3.3, il y a un taux de compression de 66 % et un pourcentage d'erreur de 11.09 %.

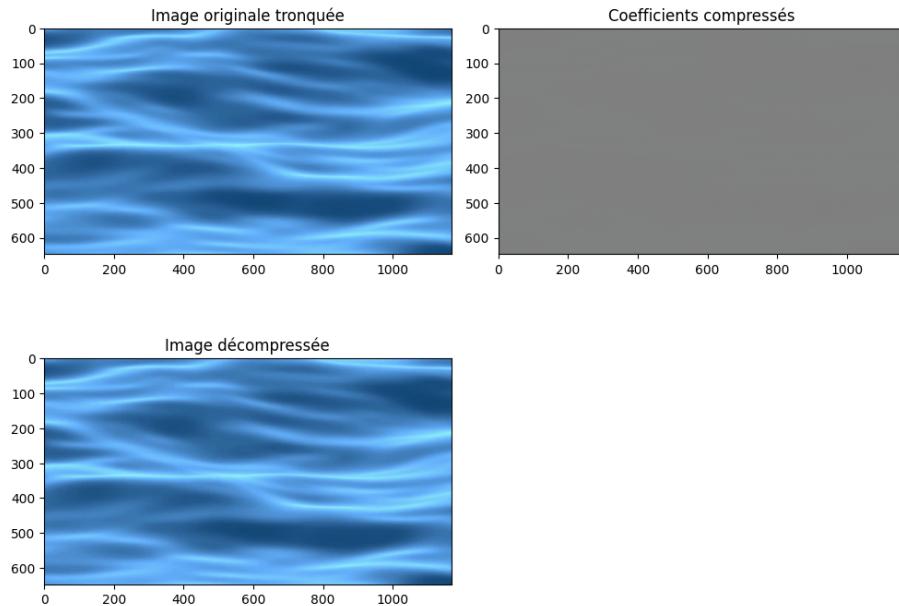


FIGURE 3.4 – Vagues - image compressée/décompressée

Pour la figure 3.4, il y a un taux de compression de 96 % et un pourcentage d'erreur de 0.87 %.

3.2 Compression par filtre

De même, nous avons appliqué notre algorithme de compression par filtre et de décompression aux deux images et nous avons obtenu les résultats suivants pour un filtre égal à 2 :

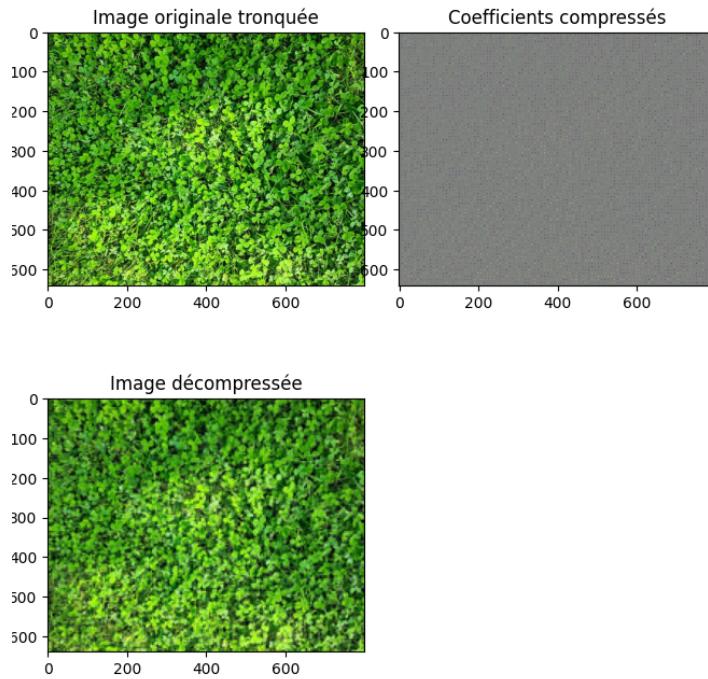


FIGURE 3.5 – Trèfles - image compressée/décompressée - F=2

Pour la figure 3.5, il y a un taux de compression de 95 % et un pourcentage d'erreur de 31.28 %.

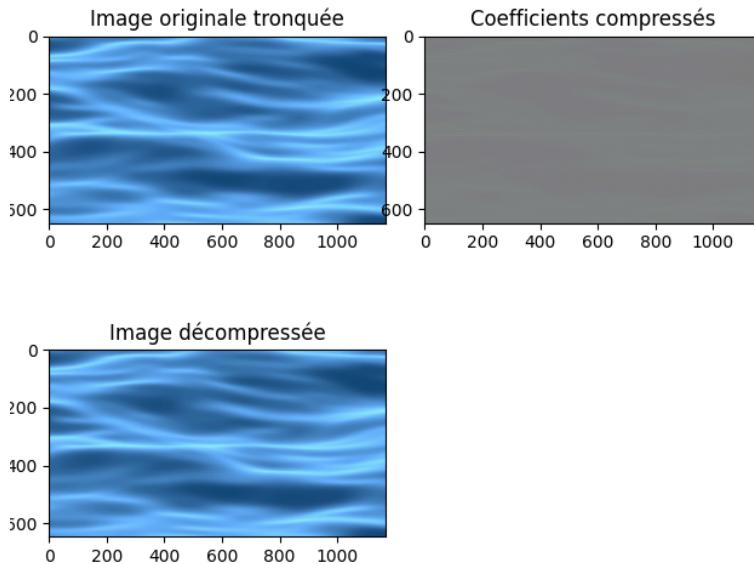


FIGURE 3.6 – Vagues - image compressée/décompressée - F=2

Pour la figure 3.6, il y a un taux de compression de 96 % et un pourcentage d'erreur de 0.83 %.

Maintenant, prenons un filtre égal à 10 :

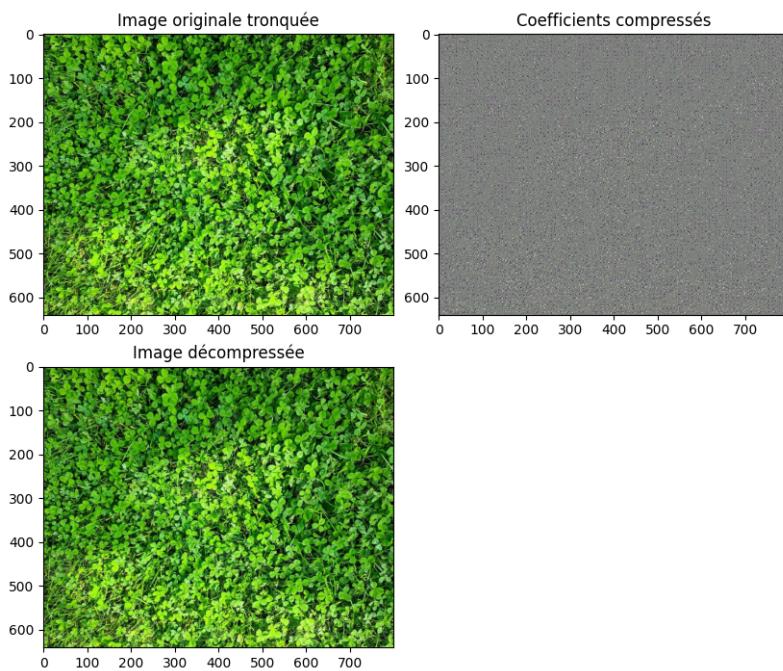


FIGURE 3.7 – Trèfles - image compressée/décompressée - F=10

Pour la figure 3.7, il y a un taux de compression de 32 % et un pourcentage d'erreur de 6.17 %.

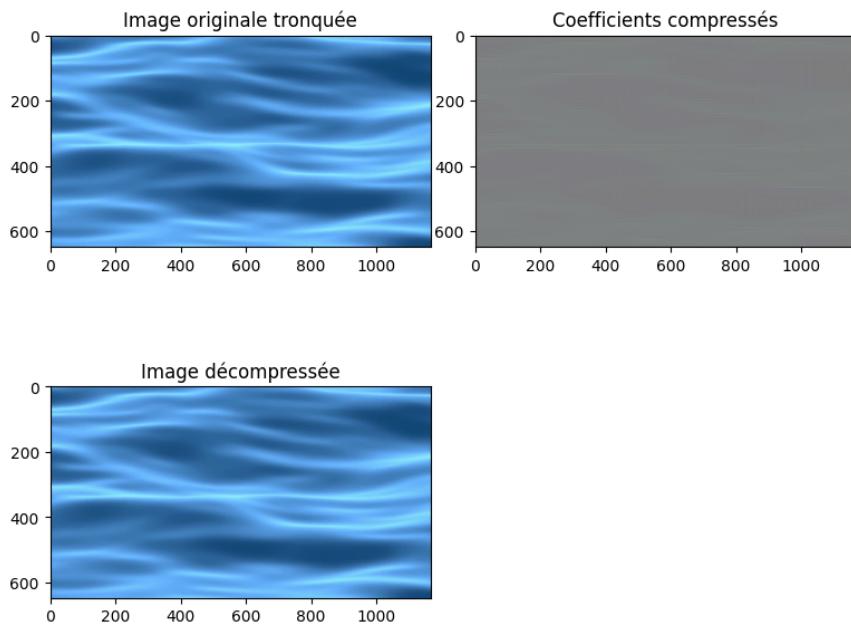


FIGURE 3.8 – Vagues - image compressée/décompressée - F=10

Pour la figure 3.8, il y a un taux de compression de 89 % et un pourcentage d'erreur de 0.52 %.

Cependant, à première vue, on dirait qu'il n'y a pas de différences entre les images originales et celles compressées puis décompressées. Essayons de zoomer un peu.

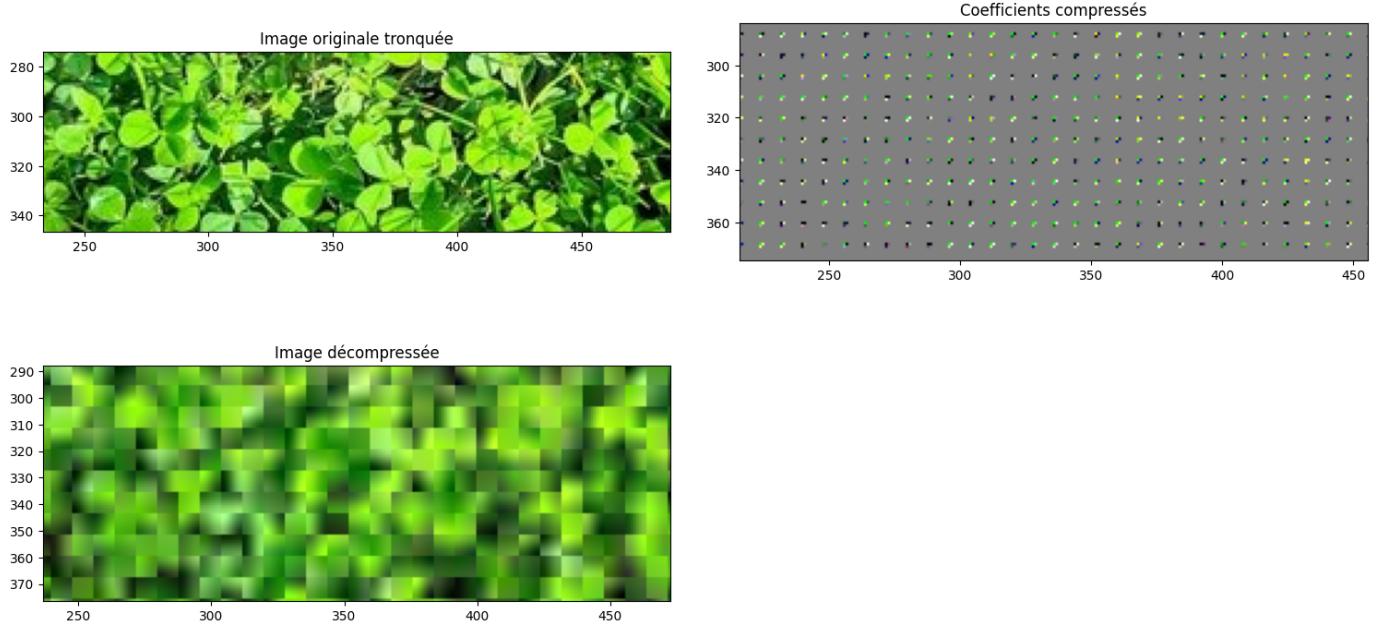


FIGURE 3.9 – Trèfles - image compressée/décompressée - $F=2$

On observe bien que l'image compressée/décompressée pour un filtre égal à 2 est moins détaillée que l'originale.

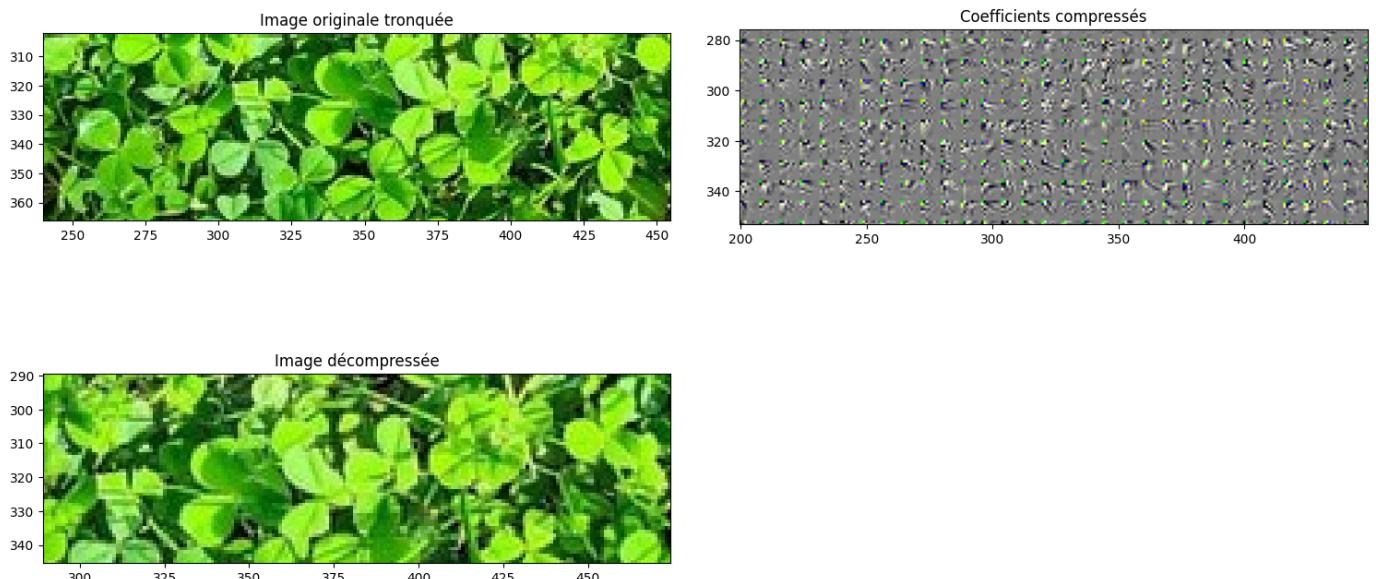


FIGURE 3.10 – Trèfles - image compressée/décompressée - $F=10$

Ici, la différence est moins flagrante car le filtre est égal à 10.

3.3 Essais avec des matrices de quantification différentes et en appliquant un filtre fixé

Nous allons maintenant utiliser cette image très détaillée d'un cerisier :



FIGURE 3.11 – Cerisier

En utilisant un filtre égal à 6, et en prenant un $k \in [1; 4; 8; 15; 40]$, nous faisons cette compression par quantification : $Q \times k$ où Q est la matrice (2.2). On obtient :

Image originale



Compressée Q classique
Filtre : 6



Compressée Q * 4
Filtre : 6



Taux : 93%

Taux : 87%
Compressée Q * 8
Filtre : 6



Taux : 96%

Compressée Q * 15
Filtre : 6



Taux : 98%

Compressée Q * 40
Filtre : 6



Taux : 99%

FIGURE 3.12 – Cerisier - image compressée/décompressée - F=6 - Q varie

On voit que plus les valeurs dans Q sont grandes, plus l'image obtenue à la fin est compressée.

3.4 Essais avec la matrice de quantification Q et en appliquant des filtres différents

De la même manière, en utilisant un filtre $F \in [1; 2; 4; 6; 8]$, et en prenant Q qui est la matrice (2.2), on obtient :

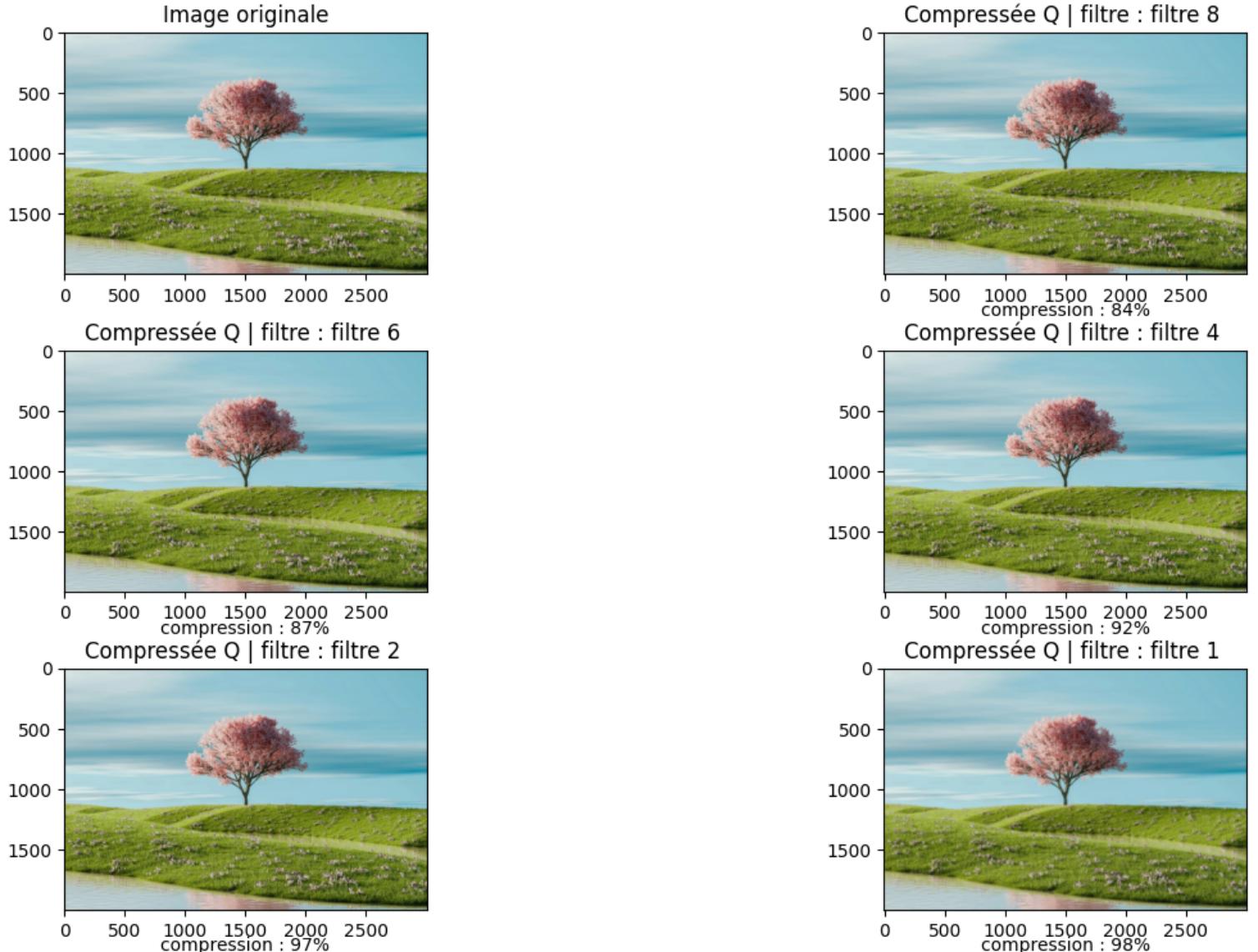


FIGURE 3.13 – Cerisier - image compressée/décompressée - Q fixée - F varie

On observe que les images sont bien compressées. Cependant, les images obtenues sont beaucoup moins compressées que lorsqu'on compresse en faisant varier la matrice de quantification et en prenant un filtre fixé.

3.5 Compression avec la matrice de quantification Q et en appliquant des filtres différents pour une image bruitée

De la même manière, en utilisant un filtre $F \in [1; 2; 4; 6; 8]$, et en prenant Q qui est la matrice (2.2), on obtient :

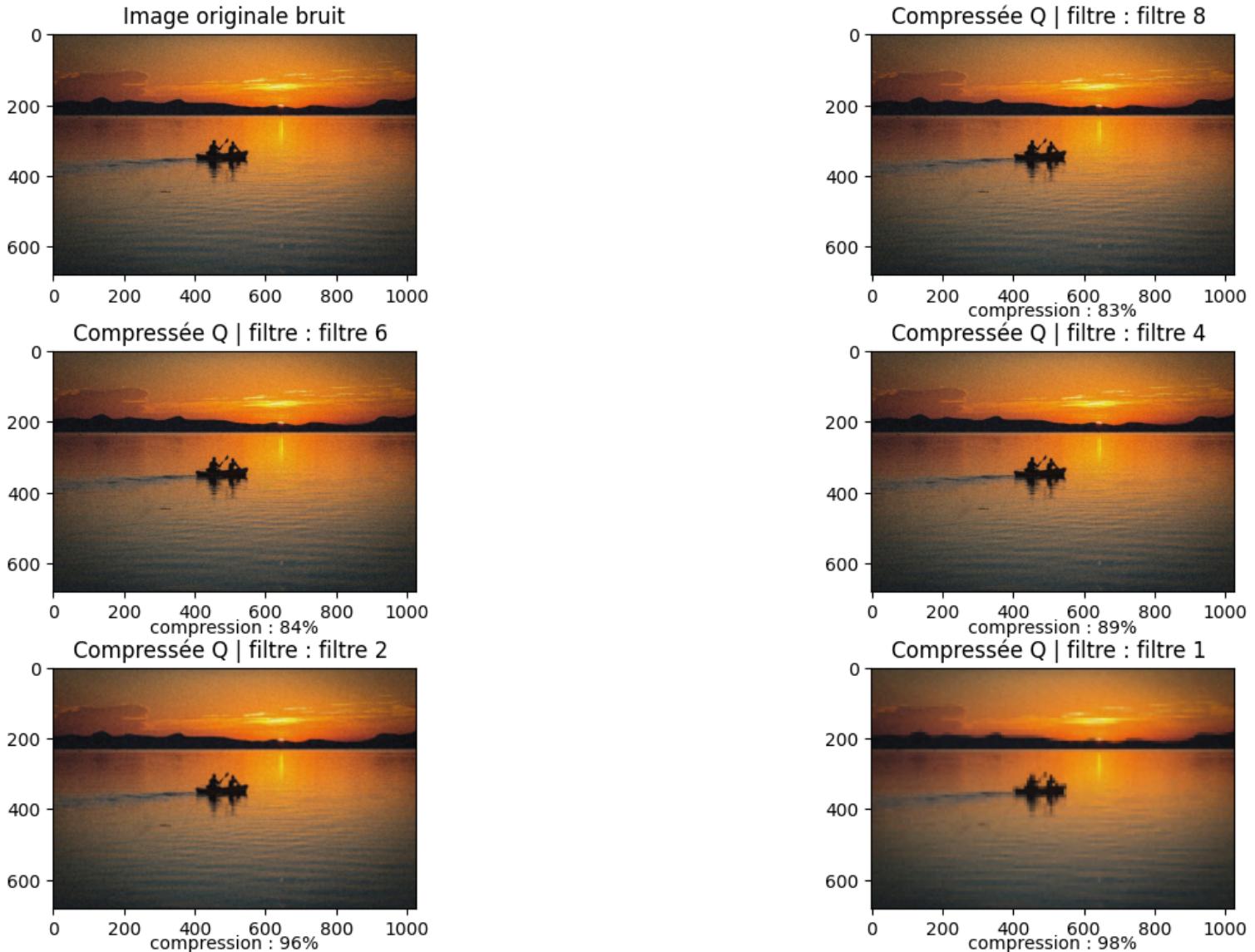


FIGURE 3.14 – Image bruitée - image compressée/décompressée - Q fixée - F varie

On observe que plus le filtre est bas, moins l'image est bruitée. Cependant, les images obtenues subissent la même perte de qualité.

Chapitre 4

Conclusions

Finalement, nous avons pu observer que la transformée de Fourier et plus précisément la transformée de cosinus discrète est un outil efficace pour le traitement des images. En effet, après avoir initialisé la matrice, nous lui avons appliqué la transformée de Fourier pour compresser, puis enlever les hautes fréquences avant d'utiliser la transformée de Fourier inverse. Cela permet de minimiser l'espace de stockage d'une image tout en gardant une image de qualité correcte. Nous observons des petits carrés de même couleur sur l'image modifiée qui correspondent aux blocs de 8×8 pixels mais qui sont visibles seulement lorsque la photo est zoomée. De plus, le temps d'exécution de la compression et de la décompression est relativement rapide pour que notre projet soit utilisé en pratique.

À noter que si on prend une matrice de quantification avec des valeurs très élevées et en prenant un filtre très bas, alors on pourrait obtenir une image très compressée.

On peut s'imaginer également rajouter des détails sur une image de faible qualité, comme par exemple pour la restauration d'images anciennes.

Références

Wikipédia : Transformée en cosinus discrète

Manuel d'utilisation de la bibliothèque `NumPy`

Manuel d'utilisation de la bibliothèque `Matplotlib`

Compression par transformée et Codage JPEG