Rapport du projet TS2I
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

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, 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 La 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 décomposer une image dans une base trigonométrique. La transformée de

Fourier discrète d'une image périodisée peut être représentée uniquement à l'aide de fonctions cosinus, grâce à la parité de la fonction.

2.1 Changement de base

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 8×8 pixels et s'exprime pour un bloc représenté par une matrice $M = M(i,j)$ de dimension 8×8 comme suit :

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

avec :

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

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 :

$$B = PM \iff B = \begin{pmatrix} P_{0,0} & \dots & P_{0,7} \\ \vdots & \ddots & \vdots \\ P_{7,0} & \dots & P_{7,7} \end{pmatrix} \begin{pmatrix} M_{0,0} & \dots & M_{0,7} \\ \vdots & \ddots & \vdots \\ M_{7,0} & \dots & M_{7,7} \end{pmatrix} \tag{2}$$

Ainsi on a par définition du produit matricielle :

$$\forall (i,j,r) \in [\![0,7]\!]^3, \;\; B_{i,j} = \sum_{r=0}^{7} P_{i,r} M_{r,j}$$

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 :

$$\forall (i,j,r,q) \in [\![0,7]\!]^4, \;\; D_{i,j} = \sum_{q=0}^{7} B_{i,r} P_{k,q}$$

Donc :

$$\forall (i,j,r,q) \in [\![0,7]\!], \;\; D_{i,j} = \sum_{q=0}^{7} \sum_{r=0}^{7} P_{i,r} M_{r,q} P_{j,b}$$

$$D_{l,k} = \sum_{q=0}^{7} \sum_{r=0}^{7} P_{l,r} M_{r,q} P_{k,b}$$

Par identification avec (1), il en résulte :

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

Finalement on en déduit :

$$P_{i,j} = \frac{1}{2} C_i \cos\left(\frac{(2j+1)i\pi}{16}\right) \tag{3}$$

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 le changement de base est réalisée. On divise terme par terme D par Q, où Q est une matrice de quantification. Puis on arrondis les résultats, 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 de 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 d'annihiler les hautes-fréquences inutiles. Et donc éliminer le bruit.

3 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 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 *(avec la norme de Frobenius).*
- 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).*

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 les multiplications matricielles.
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 parcourent 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 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]` 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.

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 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 :

$$M = {}^tPDP$$

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, puis on arrondit 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.

$$\texttt{erreur\_totale} = (\texttt{np.linalg.norm}(((\texttt{image} + 128) - \texttt{decompressed}))/(\texttt{np.linalg.norm}(\texttt{image}))) * 100$$

.
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.

4.4.2 Recentralisation 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 :

$$\texttt{taux\_compression} = 100 - ((\texttt{nb\_coeff\_non\_zero}/(\texttt{taille\_image}[1] * \texttt{taille\_image}[0] * 3)) * 100)$$

5 Résultats

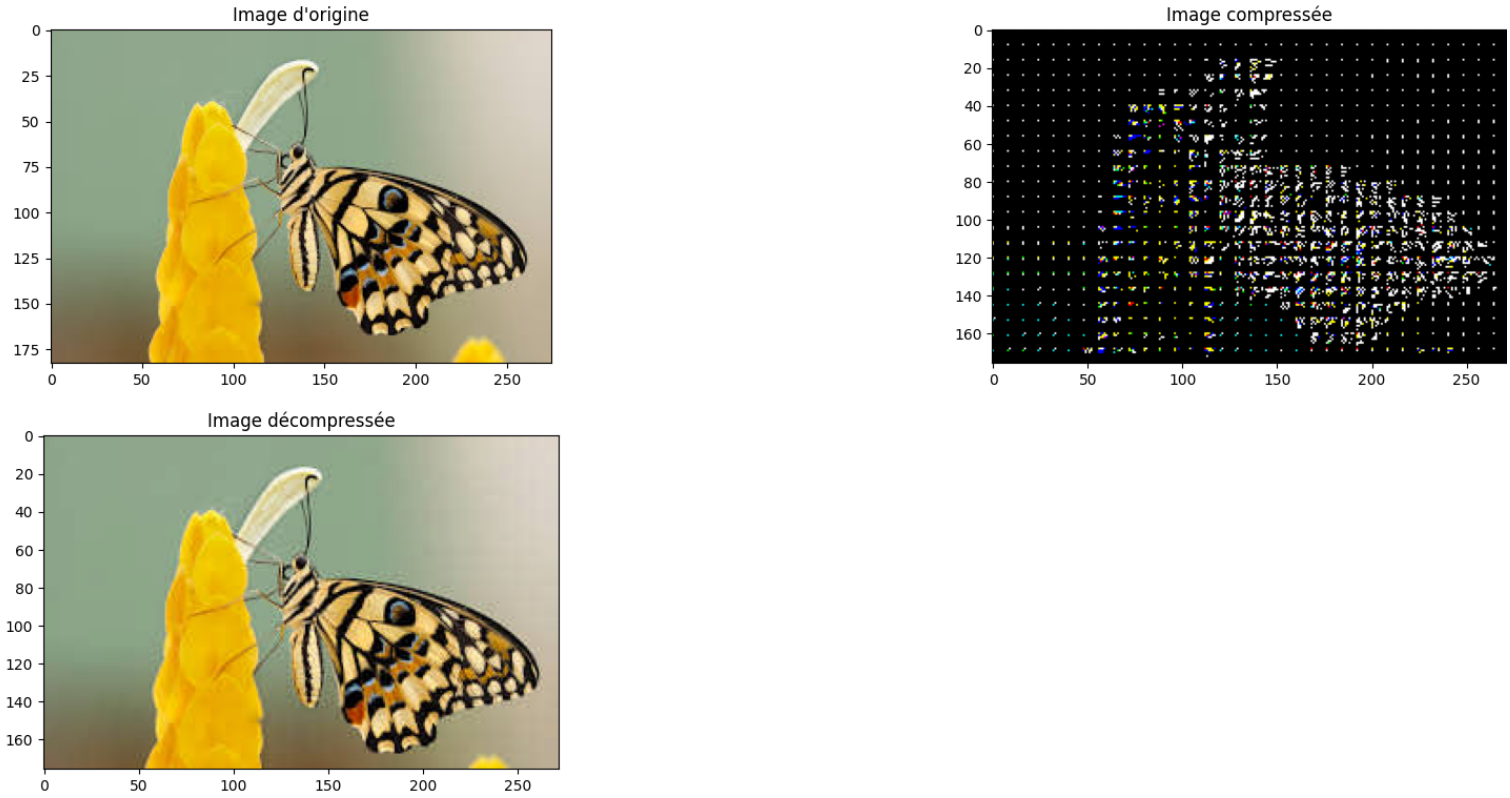

FIGURE 1 – Comparaison entre une image originale et sa version décompressé *(en couleur)*

6 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 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.

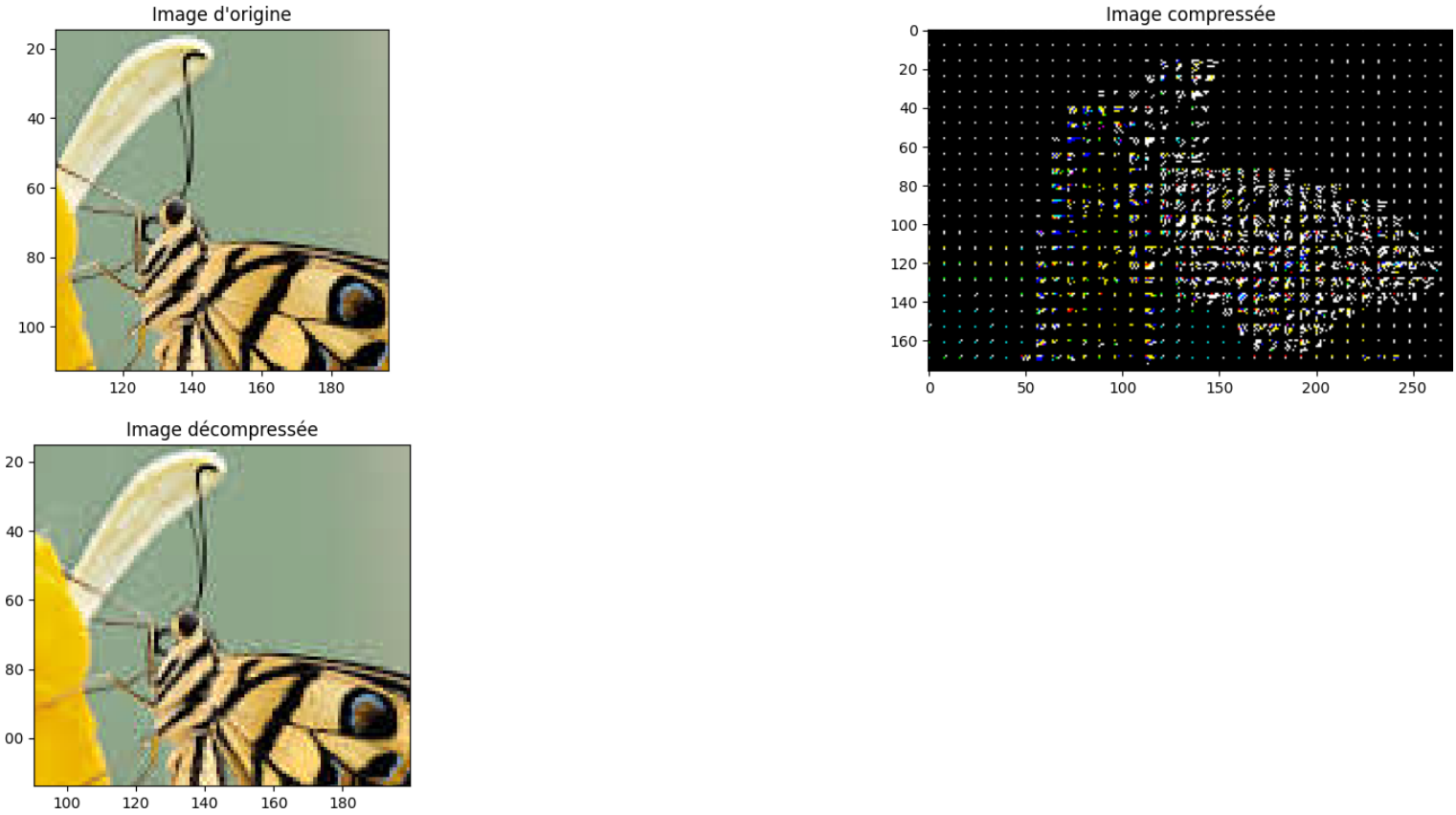

FIGURE 2 – Zoom sur l'image précédente

7 Conclusion

En conclusion, ce projet a 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. Les différences avec l'image originale sont peu visibles dans l'ensemble, mais en zoomant, on peut remarquer une pixellisation.

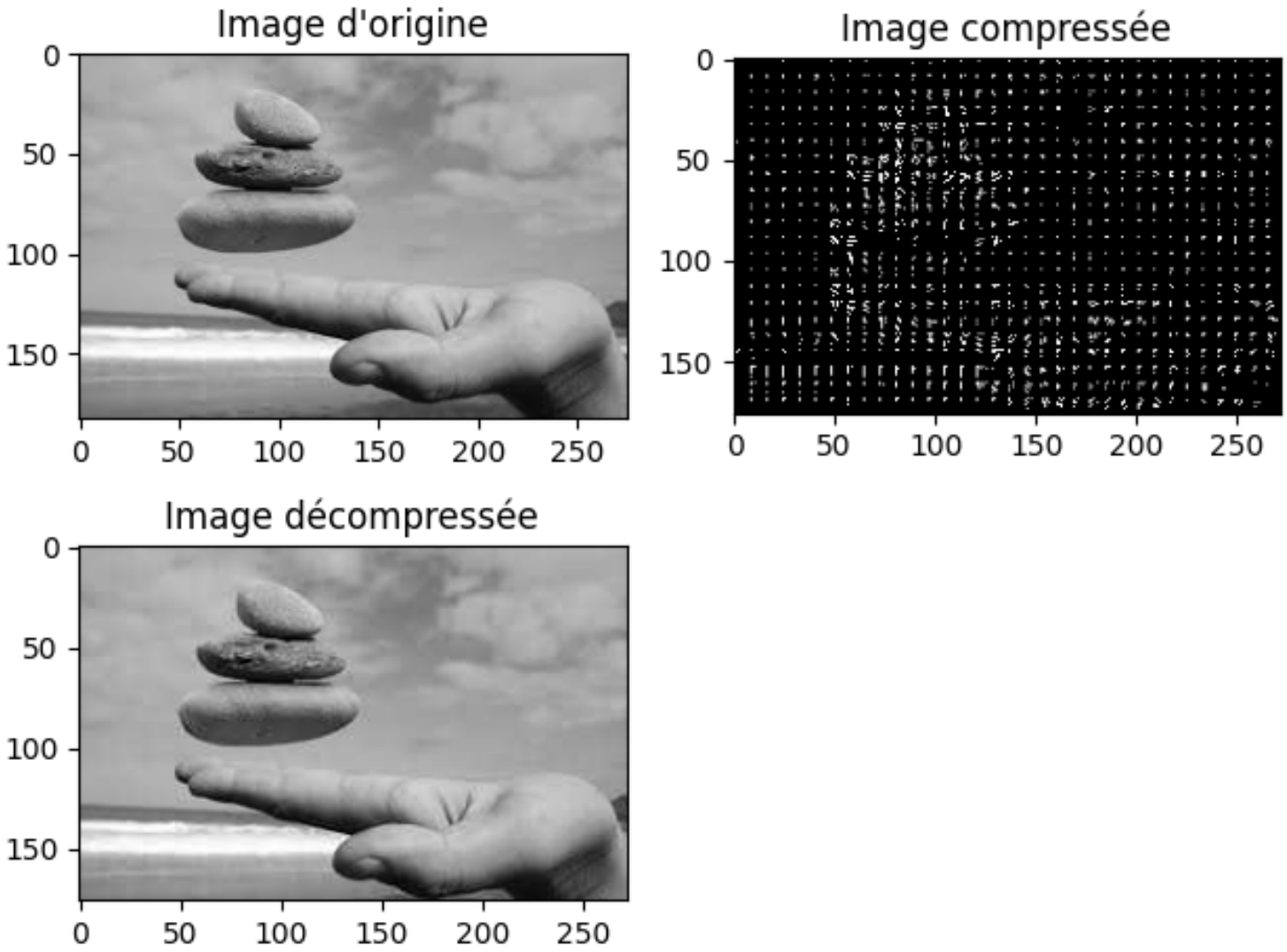

FIGURE 3 – Comparaison entre une image originale et sa version décompressé *(en noir & blanc)*

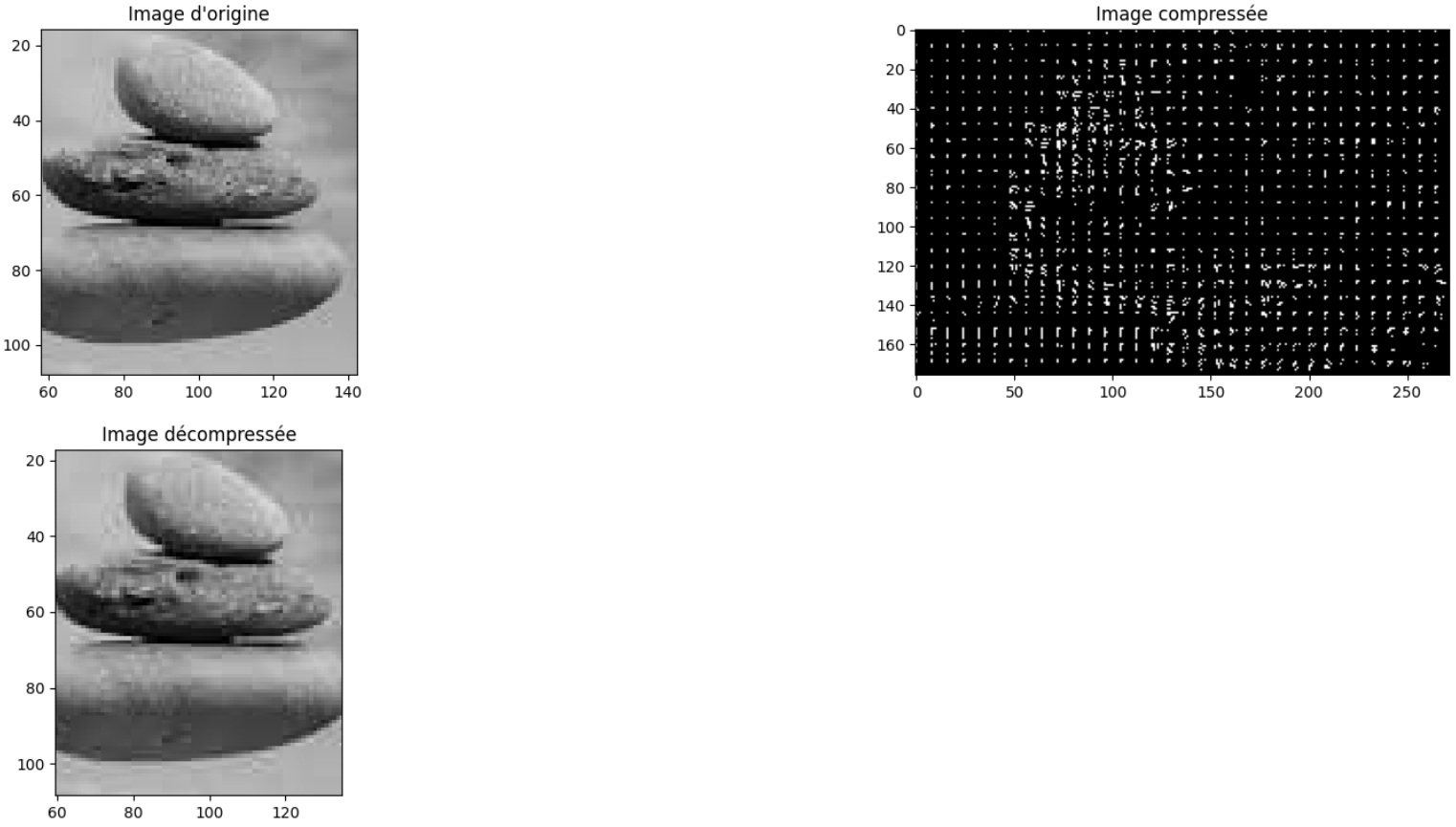

FIGURE 4 – Zoom sur l'image précédente

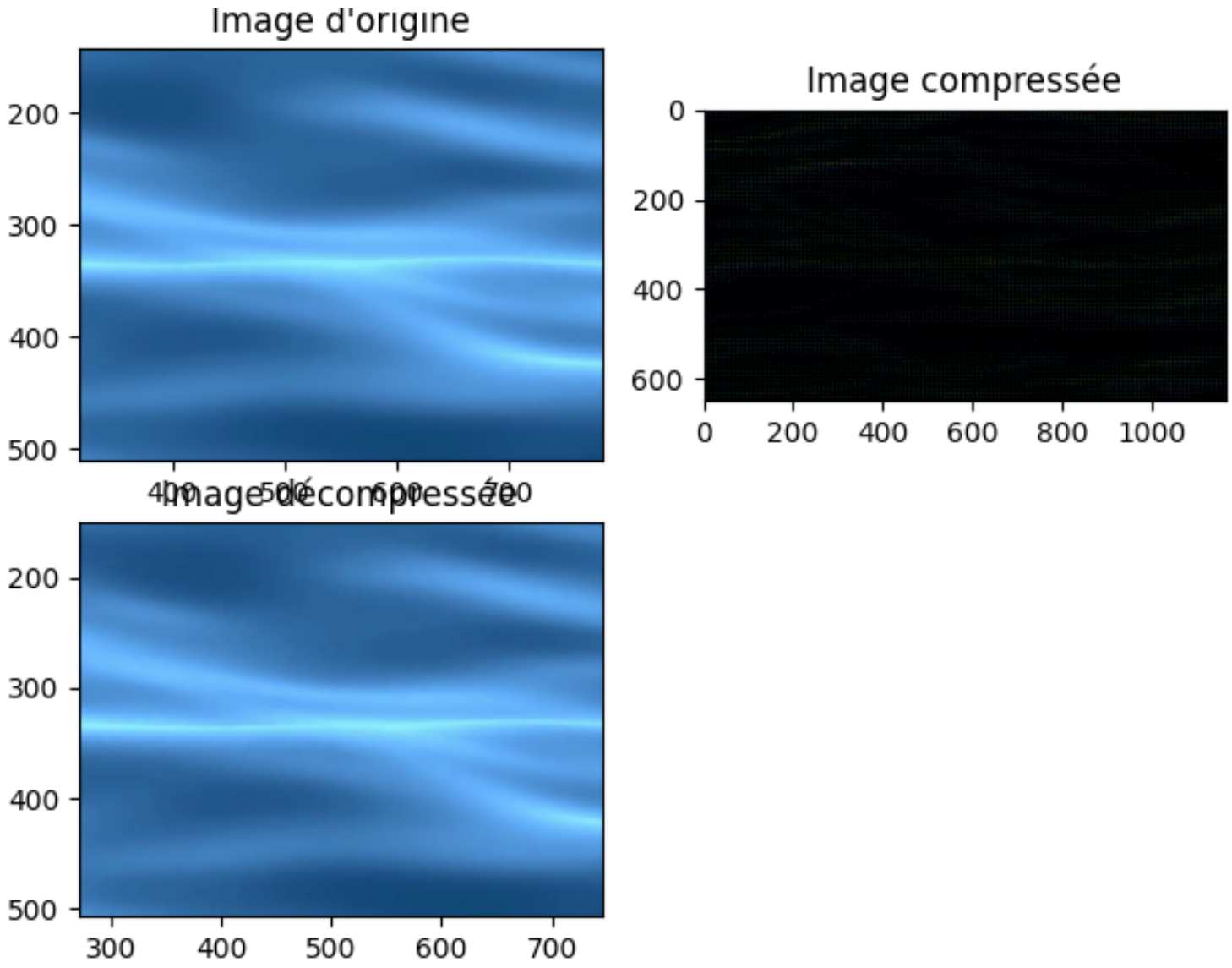

FIGURE 5 – Comparaison entre une image originale avec peu de mouvement et sa version décompressée