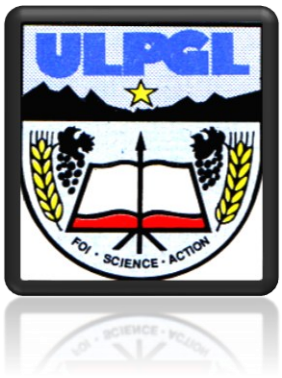
UNIVERSITE LIBRE DES PAYS DES GRANDS LACS

FACULTE DE SCIENCES ET TECHNOLOGIES

DEPARTEMENT DE GENIE ELECTRIQUE ET INFORMATIQUE



BP. 368 GOMA

www.ulpgl.net

CONCEPTION D'UNE APPLICATION WEB DE COMPRESSION D'IMAGES EN FORMAT JPEG, PNG

Par **KULE WA-KANGITSI Robert**

Travail présenté et défendu en vue de l'obtention du Diplôme de bachelor en Sciences de l'Ingénieur

Mention : Génie informatique

Directeur : Assoc. Prof. NKIEDIEL Alain AKWIR

Encadreur : Ir. IKANGAMINO KISAMBA Johnson

ANNÉE ACADÉMIQUE 2023 - 2024

Epigraphe

« La perfection est atteinte, non pas lorsqu'il n'y a plus à ajouter, mais lorsqu'il n'y a plus à retirer »

Antoine de Saint-Exupéry *(terre des hommes)*

Dédicace

À ma mère, Germanie, dont l'âme nous accompagne chaque jour et à mon père, KANGITSI MUKOHE Amédée, qui m'a soutenu sans relâche, nous dédions ce travail. Leur amour, leur sacrifice et leur présence ont été une source inépuisable de force et de motivation tout au long de ce parcours. Ce mémoire est le reflet de leur dévouement et de leur inestimable contribution à ma réussite.

KULE WA-KANGITSI ROBERT

Remerciements

Nous exprimons notre profonde gratitude à Dieu Tout-Puissant, dont la grâce, la sagesse et la protection nous ont accompagnés tout au long de notre parcours académique et durant la réalisation de ce mémoire portant sur la compression d'images basée sur l'intelligence artificielle.

Nos remerciements vont ensuite à notre université, pour nous avoir offert un cadre de formation de qualité, propice à notre apprentissage et à notre développement personnel. Nous remercions également l'ensemble du corps professoral, pour leur disponibilité, leur dévouement et leur engagement à transmettre leur savoir.

Nous tenons à exprimer notre profonde reconnaissance à notre Directeur de mémoire, le Professeur Alain AKWIR NKIEDIEL, ainsi qu'à notre Encadreur, l'Ingénieur Johnson KISAMBA, pour leur accompagnement rigoureux, leurs conseils éclairés, leur patience et leur soutien constant tout au long de ce projet. Leur expertise a été déterminante dans la réalisation de ce travail.

Nos sincères remerciements s'adressent à notre famille, en particulier Mado KANGITSI, feu Juvénal PALUKU, Raymond KANGITSI, et notre père, pour leur soutien moral et financier indéfectible. Nous exprimons également notre gratitude à maman Jeannette VAYOYA, pour son hospitalité et son appui constant.

Nous remercions chaleureusement nos camarades et amis, notamment AMURI TCHALUMBA Héritier et IKUZWE NDINAYO Barthélémy, pour leur précieuse collaboration, leur soutien fraternel et leur esprit de partage tout au long de notre parcours.

Enfin, nous adressons nos remerciements à papa MBALOLYAKO Evariste, ainsi qu'à toutes les personnes qui, de près ou de loin, ont contribué à la réussite de ce projet. Leur bienveillance et leur encouragement nous ont été d'un grand soutien.

KULE WA-KANGITSI ROBERT

Résumé

La multiplication des images sur les réseaux sociaux et les plateformes de stockage rend leur compression essentielle pour optimiser l'espace mémoire et faciliter leur transmission. Dans ce contexte, nous avons développé Pichanyepesi, une application web permettant de compresser les images au format « *Joint Photographic Experts Group* » à l'aide d'une approche hybride. Cette approche combine la conversion des couleurs en séparant la luminosité des informations chromatiques, une méthode de réduction des dimensions fondée sur l'identification des caractéristiques les plus significatives des images, ainsi qu'un procédé de regroupement automatique des pixels en petits ensembles similaires. Cette méthode permet de réduire significativement la taille des fichiers tout en préservant une qualité visuelle élevée. Les résultats des tests montrent un taux de compression moyen de 97,97 %, avec un rapport signal sur bruit de 35,66 décibels, ce qui dépasse le seuil de qualité acceptable fixé à 30 décibels et surpasse plusieurs solutions existantes. En réduisant l'empreinte numérique des images, Pichanyepesi contribue également à atténuer l'impact écologique lié au stockage des données numériques.

Mots-clés : Compression d'images, optimisation, impact écologique, stockage.

Abstract

The proliferation of images on social media and storage platforms makes their compression essential to optimize memory space and facilitate transmission. In this context, we developed Pichanyepesi, a web application that compresses images in Joint Photographic Experts Group format using a hybrid approach. This approach combines colour conversion by separating luminance from chromatic information, a dimensionality reduction method based on identifying the most significant features of the images, and an automatic pixel grouping process into small similar sets. This method significantly reduces file size while maintaining high visual quality. Test results show an average compression rate of 97.97%, with a signal-to-noise ratio of 35.66 decibels, which exceeds the acceptable quality threshold of 30 decibels and outperforms several existing solutions. By reducing the digital footprint of images, Pichanyepesi also helps mitigate the ecological impact associated with digital data storage.

Keywords: Image compression, optimization, ecological impact, storage.

SOMMAIRE

Liste des abréviations

AA	Apprentissage Automatique
ACP	Analyse en Composantes Principales
BTC	Codage De Troncature De Blocs
CLE	Codage Par Longueur d'Exécution
CSS	Cascading Style Sheets
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
FFT	Fast Fourier Transform(Transformée de Fourrier Rapide)
GIF	Graphics Interchange Format
HTML	Hypertext Markup Language
IA	Intelligence Artificielle
JPEG	Joint Photographic Experts Group
MSE	Mean Square Error (Erreur quadratique moyenne)
MVC	Model-View-Controller
PCA	Principal Component Analysis
PNG	Portable Network Graphics
PSNR	Peak Signal-to-Noise Ratio (Rapport Signal sur bruit de pointe)
RC	Ratio De Compression
RGB	Red, Green, Blue
RVB	Rouge, Vert, Bleu
YCbCr	Luminance Et Chrominance De Bleu Et De Rouge

Liste des tableaux

Liste des équations

Liste des figures

Introduction générale

0.1 Contexte

L'essor du numérique a profondément transformé notre manière de produire, stocker et partager des images. Aujourd'hui, selon « *statica* » 3,2 milliards d'images sont générées chaque jour via les réseaux sociaux, la photographie numérique, la télésurveillance et les bases de données médicales [1] [2]. Cette explosion des contenus visuels pose des défis majeurs en matière de stockage, transmission et traitement informatique.

Les méthodes traditionnelles de compression d'images comme JPEG, PNG et GIF sont largement utilisées pour réduire la taille des fichiers. Cependant, elles présentent des limites importantes. Le format JPEG, par exemple, offre une compression avec perte mais peut entraîner une détérioration visible de l'image à fort taux de compression. Le format PNG, quant à lui, permet une compression sans perte, mais les fichiers générés restent volumineux. Enfin, le format GIF, bien qu'adapté aux animations, souffre d'une qualité limitée et d'une palette de couleurs restreinte.

Face à ces défis, de nouvelles approches basées sur l'intelligence artificielle (IA) et l'apprentissage automatique (AA) émergent. Ces techniques permettent d'optimiser la compression tout en maintenant une qualité visuelle acceptable.

C'est dans cette optique que nous avons conçu « *Pichanyepesi* », qui signifie "image légère" en swahili.

Notre approche combine le YCbCr, l'ACP et le clustering MiniBatchKMeans pour réduire intelligemment la taille des images tout en conservant leur structure visuelle essentielle. Cette solution sera intégrée dans une application web interactive afin d'en faciliter l'utilisation par un large public.

0.2 Identification et formulation du problème

0.2.1 Identification du problème

L'augmentation rapide du volume d'images numériques engendre des défis techniques et économiques majeurs. Le stockage devient rapidement insuffisant, saturant les serveurs cloud et les disques durs, ce qui entraîne des coûts supplémentaires. La transmission des images volumineuses nécessite davantage de bande passante, ralentissant ainsi la navigation et dégradant l'expérience utilisateur. De plus, la gestion des images lourdes complique leur

intégration dans des domaines tels que la vision par ordinateur, le streaming et l'archivage médical, où le traitement des données devient plus complexe.

0.2.2 Formulation du problème

Comment concevoir une méthode de compression d'images efficace, permettant de réduire la taille des fichiers tout en garantissant une bonne qualité visuelle, et pouvant être intégrée dans une application web pour un usage simplifié ?

0.3 Questions de recherche

Pour répondre à la problématique posée, nous avons formulé les questions suivantes :

- Quelles sont les limites des méthodes classiques de compression (JPEG, PNG) en termes de stockage et de transmission ?
- Quelles technologies peuvent être utilisées au-delà des solutions existantes pour améliorer la compression des images ?
- Comment intégrer efficacement ces technologies dans un système accessible aux utilisateurs ?

0.4 Formulation des hypothèses

En réponse aux questions de recherche, nous formulons les hypothèses suivantes :

- Une réduction insuffisante de la taille des fichiers et une perte de qualité à fort taux de compression seraient les principales limites des méthodes classiques (JPEG, PNG) en matière de stockage et de transmission.
- En exploitant les méthodes de l'apprentissage non supervisé, telles que l'analyse en composantes principales et le clustering, il serait possible d'améliorer l'efficacité de la compression tout en préservant les détails et la structure des images.
- Le développement d'une application web ergonomique et modulaire permettrait une intégration efficace de ces technologies, rendant leur utilisation plus accessible.

0.5 Justification du choix du sujet et motivations

Le choix de ce sujet est motivé par l'importance croissante de la compression d'images dans des domaines variés tels que le stockage cloud, les réseaux sociaux et la photographie numérique. Réduire la taille des fichiers tout en préservant leur qualité est devenu un défi technologique majeur, particulièrement avec l'explosion des contenus visuels. L'apport de l'IA, notamment à travers l'ACP et le clustering MiniBatchKMeans, permet de traiter ce défi en exploitant les structures internes des images, offrant ainsi des solutions de compression plus efficaces et intelligentes. Cette approche promet d'améliorer significativement le stockage et la

vitesse de transmission, tout en ayant des applications dans des secteurs professionnels et sociaux comme les réseaux sociaux, le cloud computing et l'archivage numérique, contribuant ainsi à une meilleure gestion des ressources numériques dans un monde de plus en plus visuel.

0.6 Énoncé des objectifs de recherche

0.6.1 L'objectif général

Ce projet vise à développer et évaluer une solution de compression d'images combinant YCbCr, ACP et MiniBatchKMeans, intégrée dans une application web accessible aux utilisateurs.

0.6.2 Les objectifs opérationnels

Pour atteindre cet objectif général, nous nous fixons les objectifs suivants :

- Étudier les méthodes classiques de compression et identifier leurs limites.
- Concevoir un algorithme hybride YCbCr +ACP + MiniBatchKMeans pour optimiser la compression.
- Développer l'application « *Pichanyepesi* » avec une interface ergonomique et intuitive.
- Évaluer la qualité de la compression avec des mesures scientifiques (PSNR, MSE, ratio de compression).
- Optimiser l'algorithme pour réduire le temps de traitement et améliorer l'expérience utilisateur.

0.7 Méthodologie et délimitation du travail

0.7.1 Méthodologie

Dans ce travail, nous utilisons une méthodologie comprenant la méthode documentaire, la méthode analytique et la méthode expérimentale.

- **La méthode documentaire** a été utilisée pour étudier des théories existantes sur la compression des données et sur l'apprentissage automatique.
- **La méthode analytique** a aussi été utilisée dans ce travail pour décomposer notre étude en sous travaux en allant du plus complexe au plus simple.
- **La méthode expérimentale** a aussi été utilisée pour effectuer différents tests tout en évaluant notre modèle en vue de trouver un algorithme plus simple et efficace.

0.7.2 Délimitation du travail

Ce travail, mené en 2024, s'inscrit dans le domaine de la compression des images. Il propose un algorithme de compression hybride. L'application de cet algorithme fait l'objet d'une évaluation à Goma, en République Démocratique du Congo, afin de mesurer son impact dans des contextes à ressources limitées.

0.8 Subdivision du travail

Outre une introduction générale et une conclusion synthétisant notre étude, ce travail de fin de cycle (TFC) se compose de trois chapitres structurés comme suit :

- **Généralités sur la compression de données** : Ce chapitre fournit une base de connaissances sur la compression de données en générale, en expliquant les concepts clés, les types de données compressibles, les techniques courantes et les mesures de performance.
- **Conception de l'application web et de l'algorithme** : Ce chapitre décrit l'architecture de l'application web, les technologies utilisées et le choix de l'algorithme de compression, en mettant en avant ses avantages.
- **Réalisation et évaluation** : Ce chapitre détaille la mise en œuvre de l'application et de l'algorithme, présente les résultats des tests et compare les performances avec d'autres applications existantes.

Chapitre I : Généralités sur la compression de données

I.1 Introduction

L'explosion du volume de données numériques, notamment les images, pose d'importants défis en termes de stockage et de transmission. La compression des données, en réduisant la taille des fichiers sans altérer leur contenu essentiel, s'avère être une solution efficace. Elle est cruciale dans divers domaines, dont l'informatique, les télécommunications et le multimédia. Ce chapitre explore les principes fondamentaux de la compression, en se concentrant particulièrement sur la compression des images. Nous examinerons les différentes méthodes existantes, telles que la compression avec et sans perte, ainsi que les techniques courantes utilisées pour optimiser l'efficacité des algorithmes de compression. Enfin, nous aborderons l'émergence des approches basées sur l'intelligence artificielle, qui permettent d'atteindre des niveaux de compression plus élevés tout en préservant une qualité d'image optimale.

I.2 Définition de la compression.

La compression des données est un processus qui vise à réduire la taille d'un fichier en supprimant ou en codant plus efficacement certaines informations, tout en préservant son contenu essentiel [3]. Elle repose sur l'application d'un algorithme de compression, qui transforme les données en une version plus compacte (figure I-1) [4], et d'un algorithme de décompression, qui permet de retrouver les données originales [5] [6].

***Figure I-1** : un système avec une entrée et une sortie pour la compression*

I.2.1 Critères d'évaluation d'un algorithme de compression [7]

Un algorithme de compression est évalué selon trois critères principaux :

- Taux de compression : Rapport entre la taille du fichier compressé et celle du fichier original.

$$Taux = \left(1 - \frac{taille\ réduite}{taille\ originale}\right)$$

***Equation I-1** : taux de compression*

- Qualité de compression : Peut être sans perte (restauration exacte des données) ou avec perte (perte d'informations non essentielles).
- Vitesse d'exécution : Temps nécessaire pour compresser et décompresser un fichier.

I.2.2 Exemple concret

Un fichier image de 10 Mo peut être compressé à 3 Mo, ce qui correspond à un taux de compression de 70 %.

I.3 Types de compression

La compression des données peut être classée en deux grandes catégories [3] :

- La compression avec perte et
- La compression sans perte

La compression avec perte permet un gain plus important en espace mémoire, mais altère légèrement la qualité des données [8].

I.3.1 Compression sans perte (Lossless Compression)

La compression sans perte permet de retrouver exactement les données d'origine après décompression. Cette méthode est utilisée pour les données sensibles, où la moindre modification des informations serait critique, telles que les textes, les bases de données, et les images médicales [3].

Exemples : PNG, GIF, FLAC (audio), ZIP.

I.3.2 Compression avec perte (Lossy Compression)

Cette méthode supprime certaines informations jugées non essentielles afin d'obtenir un taux de compression plus élevé. Elle est principalement utilisée pour les images, les vidéos et les fichiers audio, où une légère perte de qualité est acceptable.

Exemples : JPEG, MP3(audio), MPEG (vidéo), WebP.

I.4 Importance et applications de la compression

L'essor fulgurant des technologies de l'information a entraîné une explosion sans précédent du volume de données générées [1].

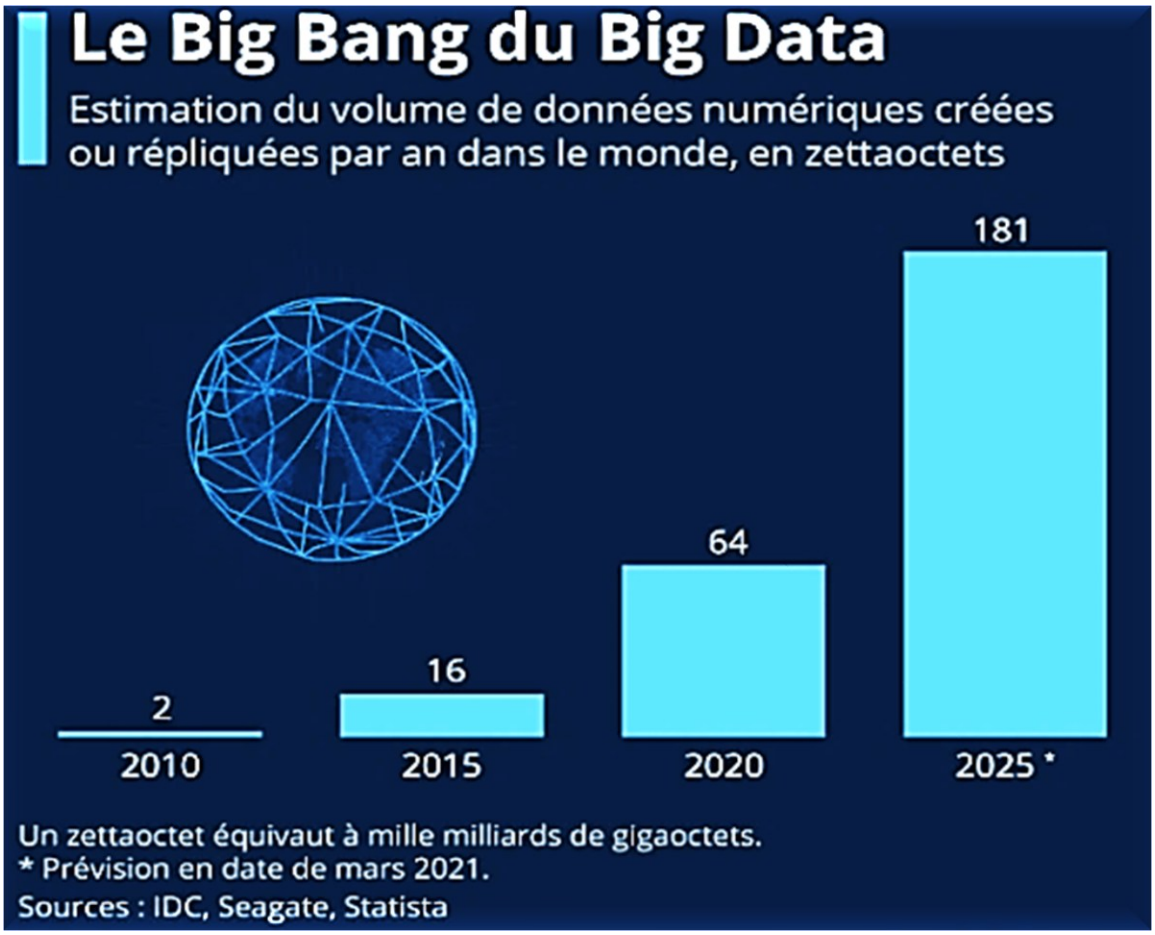


Figure I-2 *: Estimation du volume annuel de données numériques générées dans le monde entre 2010 et 2025 [2]*

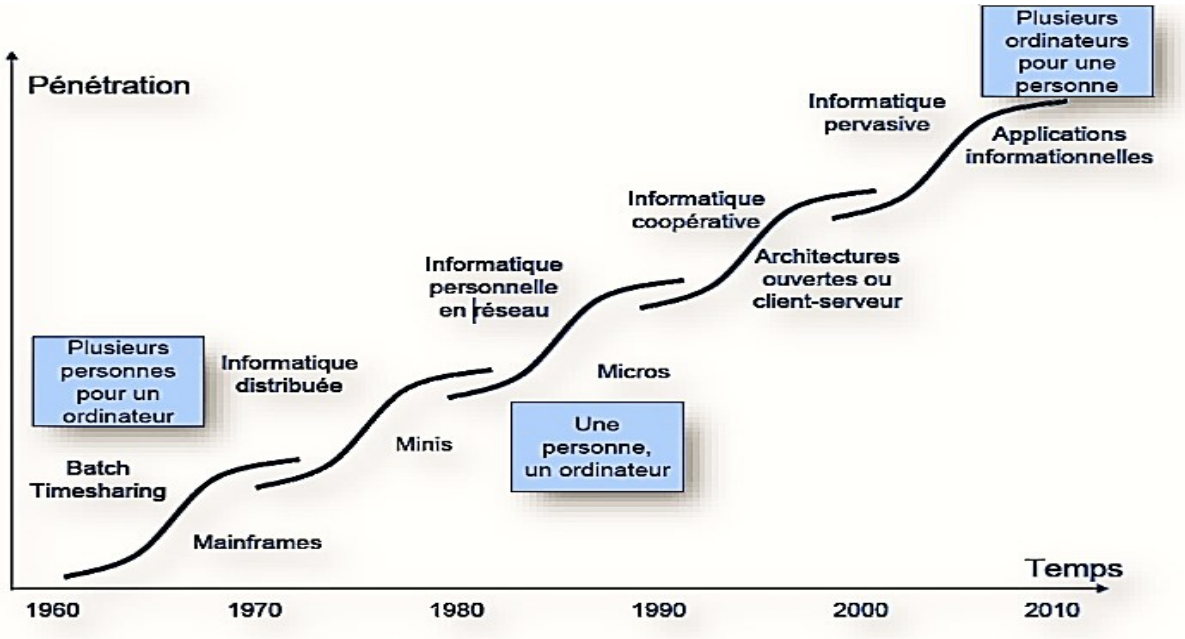


Figure I-3 *: accroissement des moyens de calcul et de stockage [9]*

Comme le montre la figure I-2, cette croissance est exponentielle et ne semble pas prête de s'arrêter [2]. Or, cette profusion de données pose de sérieux défis en termes de stockage et de transmission. Historiquement, l'évolution des modèles informatiques, illustrée par la figure I-3, a joué un rôle déterminant dans cette explosion de données [9]. La transition des mainframes vers les ordinateurs personnels, puis vers les réseaux et l'informatique pervasive, a démocratisé l'accès à l'information et multiplié les sources de données.

La compression joue un rôle essentiel dans plusieurs domaines :

- **Stockage** : Réduction de l'espace disque utilisé, diminution des coûts des serveurs cloud.
- **Transmission** : Accélération des transferts de fichiers, réduction de la consommation de bande passante.
- **Multimédia** : Essentielle pour la vidéo, la photographie et la diffusion en streaming.
- Réseaux et télécommunications : Permet le transport des fichiers sur Internet, satellite, téléphonie mobile [6].

I.5 Techniques de compression d'images [7]

Les techniques de compression peuvent être classées en deux catégories. Le tableau I-1 classifie les différentes techniques de compression.

Techniques avec pertes	Technique sans pertes
Codage par transformation	Codage par longueur d'exécution (CLE)
Quantification vectorielle	Codage entropique
Codage fractal	Codage prédictif
Codage de troncature de blocs (BTC)	Codage de Huffman

***Tableau I-1** : classification des techniques de compression [7]*

I.5.1 Techniques avec pertes

Ces méthodes réduisent la taille des fichiers en éliminant les informations jugées moins cruciales pour l'œil humain. Parmi ces techniques, nous pouvons citer :

- Codage par transformation (DCT, DWT) : Convertit l'image dans un domaine fréquentiel pour mieux identifier les zones à compresser.
- Quantification vectorielle : Regroupe des blocs de pixels en catégories pour limiter le stockage des couleurs.

- Codage fractal : Exploite la redondance des motifs dans l'image pour la compresser sous forme mathématique.

I.5.2 Techniques sans pertes

Ces techniques permettent de réduire la taille des fichiers sans perte d'information :

- Codage par longueur d'exécution (CLE) : Remplace les répétitions de pixels par un code court.
- Codage de Huffman : Attribue des codes courts aux pixels fréquents et des codes longs aux rares.
- Codage prédictif : Base la compression sur la similarité des pixels voisins.

I.6 Les algorithmes de compression

Un algorithme de compression est un ensemble de règles et de procédures utilisées pour réduire la taille d'un fichier ou d'un ensemble de données [10]. En exploitant différentes techniques, telles que le codage entropique et les transformées mathématiques, ils permettent de gagner en espace disque et de réduire les temps de chargement.

I.7 Évolution historique et principaux algorithmes

Le développement de nouveaux algorithmes de compression d'images est un défi constant, visant à réduire la taille des fichiers sans compromettre la qualité visuelle. Le graphique ci-dessous (Figure I-4) illustre cette évolution chronologique, mettant en évidence les principaux formats et leurs caractéristiques distinctives. Chaque barre représente un format d'image, positionné sur une ligne temporelle pour indiquer son année d'introduction. Les annotations fournissent des informations supplémentaires sur les développeurs, les objectifs et les avancées technologiques de chaque format.

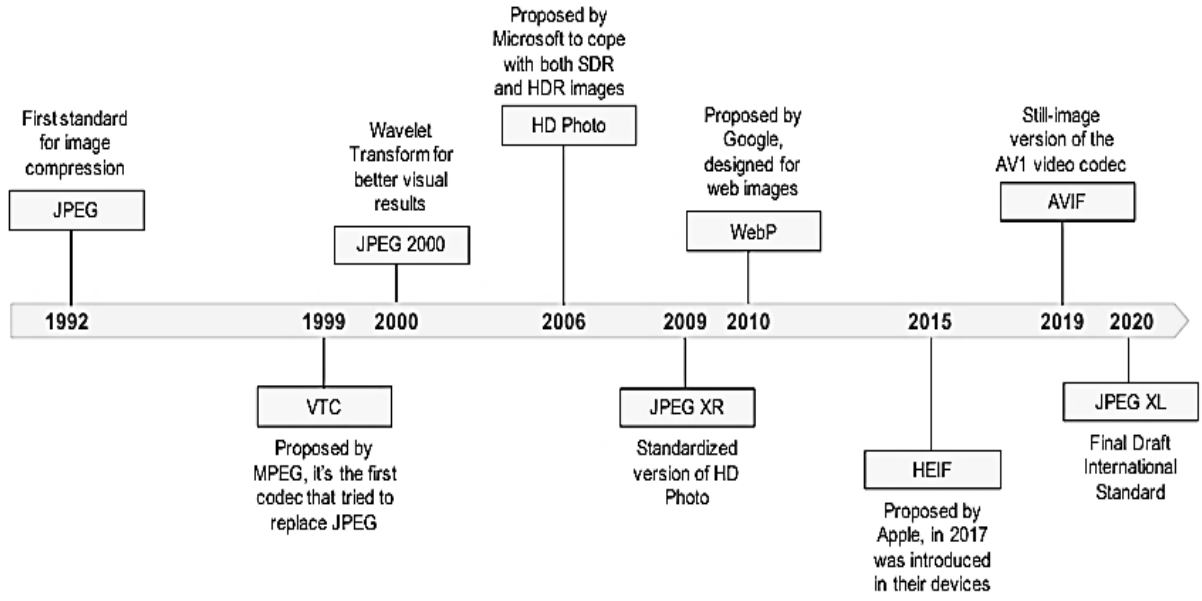


Figure I-4 *: Chronologie des Algorithmes de Compression d'Images Conventionnels avec une Efficacité Supérieure, au Cours des Trois Dernières Décennies [11]*

Comme le montre la Figure I-4, le JPEG (*Joint Photographic Experts Group*), introduit en 1992, a longtemps été le standard de facto pour la compression d'images avec pertes. Son succès s'explique par son bon compromis entre qualité et taille de fichier. Il utilise la transformée en cosinus discrète, « *Discrete Cosines Transformer* » (DCT) en anglais, dans son algorithme.

Cependant, les besoins en matière de qualité d'image et de flexibilité ont rapidement évolué, donnant naissance à de nouveaux formats plus performants.

JPEG 2000, apparu en 1999, a apporté des améliorations significatives grâce à l'utilisation de la transformée en ondelettes discrète, « *Discrete Wavelet Transform* » (DWT) en anglais. Ce format offre une meilleure qualité d'image à des taux de compression élevés, mais sa complexité a limité son adoption [12].

Les années 2000 ont vu émerger de nouveaux formats tels que WebP, conçu pour optimiser le chargement des images sur le web, et HD Photo, visant à améliorer la qualité des images numériques.

Plus récemment, AVIF, basé sur le codec vidéo AV1, a fait son apparition, promettant des taux de compression encore plus élevés et une meilleure qualité visuelle.

Cette évolution constante est motivée par plusieurs facteurs :

- ✓ **La croissance d'Internet** : Le besoin de transmettre des images rapidement et efficacement sur le web a accéléré le développement de nouveaux formats.

- ✓ **L'augmentation de la résolution des écrans** : Les écrans HD nécessitent des formats d'image capables de préserver la qualité visuelle à des tailles de fichier importantes.
- ✓ **Les avancées technologiques** : Les progrès en matière de traitement d'image et de compression ont permis de développer des algorithmes de plus en plus performants.

I.7.1 Principes de fonctionnement

Les algorithmes de compression d'images reposent sur plusieurs principes fondamentaux :

- **Transformée :** les transformées mathématiques (DCT, DWT) permettent de décomposer l'image en coefficients qui représentent différentes fréquences spatiales. Les coefficients de basses fréquences contiennent l'information globale, tandis que les coefficients de hautes fréquences correspondent aux détails.
- **Quantification :** la quantification consiste à réduire la précision des coefficients en les arrondissant à des valeurs discrètes. Plus la quantification est forte, plus la compression est importante, mais plus la perte d'information est élevée.
- **Codage entropique :** le codage entropique permet d'attribuer des codes plus courts aux symboles les plus fréquents, réduisant ainsi la taille du fichier.

I.8 L'évolution de la compression d'images et l'intelligence artificielle

I.8.1 Compression traditionnelle vs compression moderne

Les méthodes classiques (JPEG, PNG) reposent principalement sur des principes mathématiques et statistiques. Elles sont efficaces mais atteignent leurs limites dans les applications nécessitant une forte compression sans perte de qualité.
Les approches modernes intègrent l'IA, qui permettent une compression plus intelligente et optimisée.

I.8.2 Techniques basées sur l'apprentissage automatique

Les techniques de l'AA optimisent la compression d'images en réduisant les données tout en préservant leur qualité. L'ACP diminue les dimensions, tandis que MiniBatchKMeans réduit le nombre de couleurs pour alléger le fichier. Ces approches offrent un meilleur taux de compression, une qualité visuelle améliorée et une adaptation aux divers types de données. Nous développerons en détail cette approche dans le chapitre suivant.

I.9 Conclusion partielle

Ce chapitre a exploré les fondamentaux de la compression des images, en présentant ses types, techniques et applications. L'essor de l'IA permet aujourd'hui des approches plus optimisées et adaptatives. Dans le prochain chapitre, nous détaillerons la conception de l'application web Pichanyepesi et de son algorithme de compression.

Chapitre II : Conception de l'algorithme de compression et de l'application web

II.1 Introduction

Dans un contexte où le volume des images stockées et transmises ne cesse de croître, la compression d'images est devenue une nécessité pour réduire la taille des fichiers tout en conservant une qualité visuelle acceptable. Comme nous l'avons vu dans le chapitre précédent, la compression permet d'optimiser le stockage, d'accélérer les transmissions et d'améliorer les performances des systèmes multimédias.

Ce chapitre présente la conception détaillée de « *Pichanyepesi* », une solution de compression hybride (exploitant YCbCr, ACP et MiniBatchKMeans pour la réduction de redondance et pour la quantification des couleurs) intégrée dans une application web intuitive et performante (Une application web, permettant d'interagir avec l'algorithme via une interface ergonomique). L'algorithme et l'application sont conçus pour assurer une compression optimale tout en garantissant une accessibilité et une performance élevées

II.2 Fondements théoriques et conception de l'algorithme

II.2.1 Représentation et transformation des images [13]

Une image est une matrice tridimensionnelle de taille **M** × **N** ×**3**, où :

- M et N sont les dimensions (largeur, hauteur),
- 3 correspond aux canaux de couleur Rouge (R), Vert (G), Bleu (B), RVB ou RGB en anglais.

$$I(x, y, c) \in R^{M \times N \times 3}$$

Equation II-1 *: représentation mathématique d'une image*

Dans notre cas, nous considérons une image en couleur qui est une matrice à trois dimensions avec un nombre de lignes et de colonnes bien défini.

II.2.2 Conversion RVB → YCbCr [14]

La conversion de l'espace colorimétrique RVB en YCbCr permet de séparer l'information de luminance (Y) des informations de chrominance (Cb et Cr), optimisant ainsi la compression des images tout en maintenant une qualité visuelle acceptable.

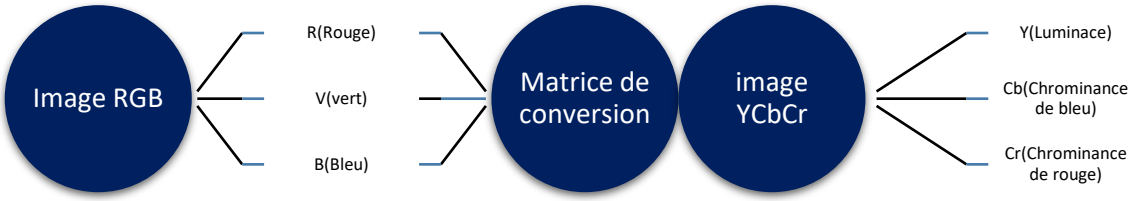


***Figure II-1** : schéma de conversion RVB en YCbCr [14]*

Le composant **Y** représente l'intensité lumineuse et est principalement dérivé des canaux Rouge (R), Vert (G) et Bleu (B), avec un poids plus important sur le vert, tandis que les composantes **Cb** et **Cr** représentent respectivement la différence entre le bleu et la luminance, ainsi que la différence entre le rouge et la luminance. Ce processus permet de réduire la résolution des chrominances sans altérer de manière significative la perception visuelle, ce qui est particulièrement utile dans des applications telles que la compression d'image et la vidéo.

La transformation suit la matrice :

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{Cb} \\ \boldsymbol{Cr} \end{bmatrix} = \begin{bmatrix} \mathbf{0,299} & \mathbf{0,587} & \mathbf{0,114} \\ \mathbf{-0,1687} & \mathbf{-0,3313} & \mathbf{0,5} \\ \mathbf{0,5} & \mathbf{-0,4187} & \mathbf{-0,0813} \end{bmatrix} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{G} \\ \boldsymbol{B} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{128} \\ \mathbf{128} \end{bmatrix}$$

***Equation II-2** : matrice de conversion RGB - YCbCr [14]*

II.2.3 Réduction de dimensions avec l'ACP [15] [16] [17]

L'ACP est une méthode qui permet de réduire la dimension d'un jet de données en conservant l'essentiel de l'information soit une variance maximale des données**.** Cette technique est particulièrement utile dans la compression d'images**,** car elle permet d'éliminer les redondances et les corrélations entre les pixels**.**

Le diagramme conceptuel de la réduction de dimensionnalité avec ACP (figure II-2) illustre comment les images en haute dimension sont transformées en un espace de moindre dimension tout en conservant l'essentiel des informations visuelles.

Formulation mathématique

1. Centrage des données :

$$X' = X - \mu$$

***Equation II-3** : centrage des données*

Où :

- X: la matrice de données originales, avec m échantillons et n variables,
- μ : vecteur moyenne de chaque variable (calculé colonne par colonne)
- X' : Matrice des données centrées où chaque colonne a une moyenne de 0.

Pourquoi centrer les données ?

➢ Permet de supprimer les biais liés aux différences d'échelles

➢ Garantit que l'ACP trouve les directions de variance maximale de manière optimale.

2. Calcul de la matrice de covariance :

$$\sum = \frac{1}{n} X'^{T} X'$$

Equation II-4 *: matrice de covariance [18]*

Où :

- X'^{T} : transposée de la matrice des données centrées
- X' : matrice des données centrées
- n : nombre total d'échantillons

La covariance permet de mesurer la corrélation entre les variables [18].

Pourquoi calculer la covariance ?

- Permet de quantifier les relations entre variables
- La diagonalisation de cette matrice nous donne les vecteurs propres (axes principaux) utilisés par PCA.

3. Décomposition en valeurs propres :

$$\sum_{Vi} = \lambda i \, Vi$$

Equation II-5 *: décomposition en valeurs et vecteurs propres [19]*

Où :

- $\boldsymbol{Vi}$: vecteurs propres (axes de variance maximale).
- $\boldsymbol{\lambda i}$: valeurs propres associées (quantité de variance).

4. Projection des données dans un espace réduit :

$$X_{réduit} = X' V_k$$

Equation II-6 *: Projection des données dans un espace réduit [20]*

Où V_k contient les vecteurs propres correspondant aux k plus grandes valeurs propres.

Diagramme conceptuel de la réduction de dimensionnalité

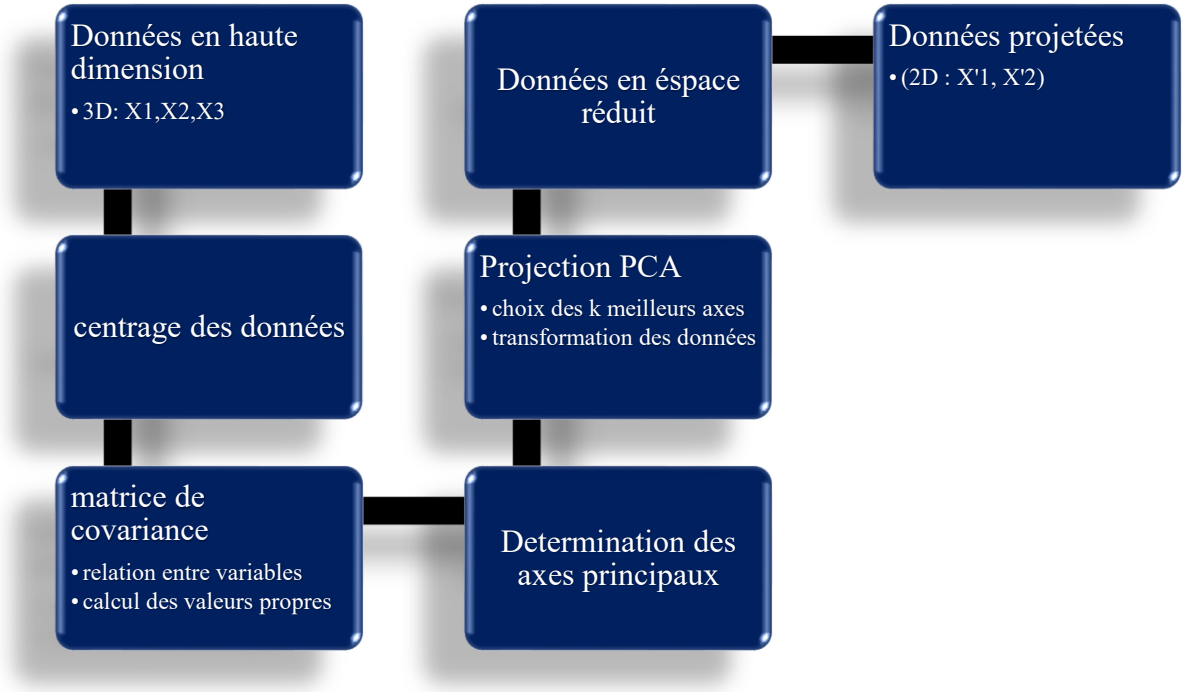


Figure II-2 *: Visualisation de la réduction de dimensions avec PCA*

Le premier bloc du diagramme présente l'image d'entrée sous forme d'une matrice de pixels en RVB. Les blocs suivant illustrent l'application de l'ACP, avec le centrage des données et leur transformation en composantes principales. Enfin, le dernier bloc représente l'image reconstruite avec une dimensionnalité réduite [20] [21]. Cette approche offre plusieurs avantages : une réduction significative du volume de données, une diminution du temps de calcul et de stockage, ainsi qu'une optimisation du traitement avant l'étape de quantification des couleurs avec MiniBatchKMeans. En implémentation, l'application de l'ACP en Python repose sur la transformation de l'image en une matrice de pixels, suivie de l'entraînement du modèle ACP pour conserver les composantes principales, puis de la compression effective des données. Cette technique joue ainsi un rôle fondamental dans notre algorithme, en garantissant une réduction efficace de la taille des images sans perte significative de qualité.

II.2.4 Clustering et quantification avec MiniBatchKMeans

Après la réduction de dimensions, nous appliquons MiniBatchKMeans pour regrouper les pixels similaires et réduire la palette de couleurs.

Formulation mathématique

L'algorithme optimise la fonction suivante :

$$J = \sum_{i=1}^{k} \sum_{x \epsilon Ci} \|x - ci\|_2^2$$

Equation II-7 *: formule de répartition des clusters [17]*

Où :

- k : nombre de clusters (centroïdes).
- Ci : ensemble des pixels appartenant au cluster i.
- ci : centroïde du cluster i.

Diagramme conceptuel du clustering

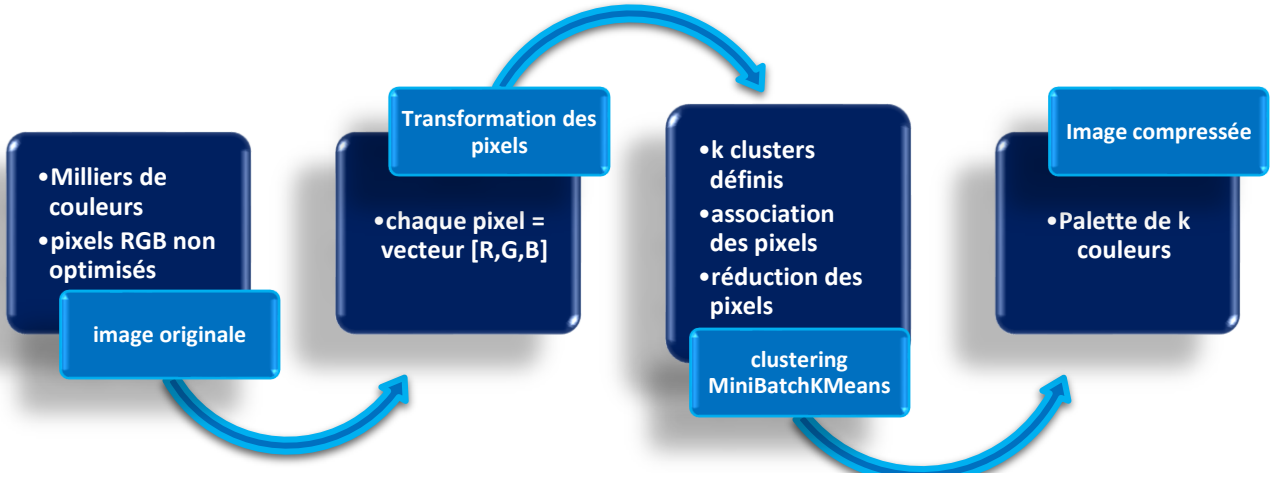


Figure II-3 *: Illustration du clustering et réduction de palette de couleurs [17]*

Le diagramme conceptuel du clustering présenté à la figure II-3 illustre l'application de l'algorithme MiniBatchKMeans après la réduction de dimensions par ACP, dans le but de regrouper les pixels similaires et de réduire la palette de couleurs de l'image. Au départ, chaque pixel est représenté sous forme d'un vecteur dans un espace de caractéristiques, où chaque point correspond à une couleur spécifique de l'image. L'algorithme commence par sélectionner aléatoirement *k* centroïdes, qui serviront de points de référence pour le regroupement.

Chaque pixel est ensuite assigné au centroïde le plus proche, sur la base de la distance euclidienne dans l'espace des couleurs. À chaque itération, les centroïdes sont mis à jour en calculant la moyenne des pixels qui leur sont assignés, ce qui permet une convergence progressive vers une segmentation optimale de l'image en un nombre limité de couleurs distinctes [17].

Le diagramme met en évidence cette évolution : depuis la dispersion initiale des pixels dans l'espace de couleurs, jusqu'au regroupement progressif autour des centroïdes, pour aboutir à une image reconstruite avec une palette réduite de teintes optimisées. Cette réduction de la diversité chromatique permet de maintenir les structures visuelles essentielles tout en facilitant la compression.

Cette approche présente plusieurs avantages : elle minimise la quantité d'informations à stocker, accélère le traitement des images et améliore le taux de compression. En pratique, MiniBatchKMeans est préféré à KMeans standard en raison de sa rapidité et de son efficacité sur de grands ensembles de données, ce qui le rend particulièrement adapté aux applications de compression d'images.

II.3 Conception détaillée de l'algorithme

Le diagramme suivant (figure II-4) présente de manière détaillée le processus de compression d'une image en utilisant l'approche hybride YCbCr + ACP + MiniBatchKMeans, depuis le chargement de l'image jusqu'à la génération et l'envoi de l'image compressée.

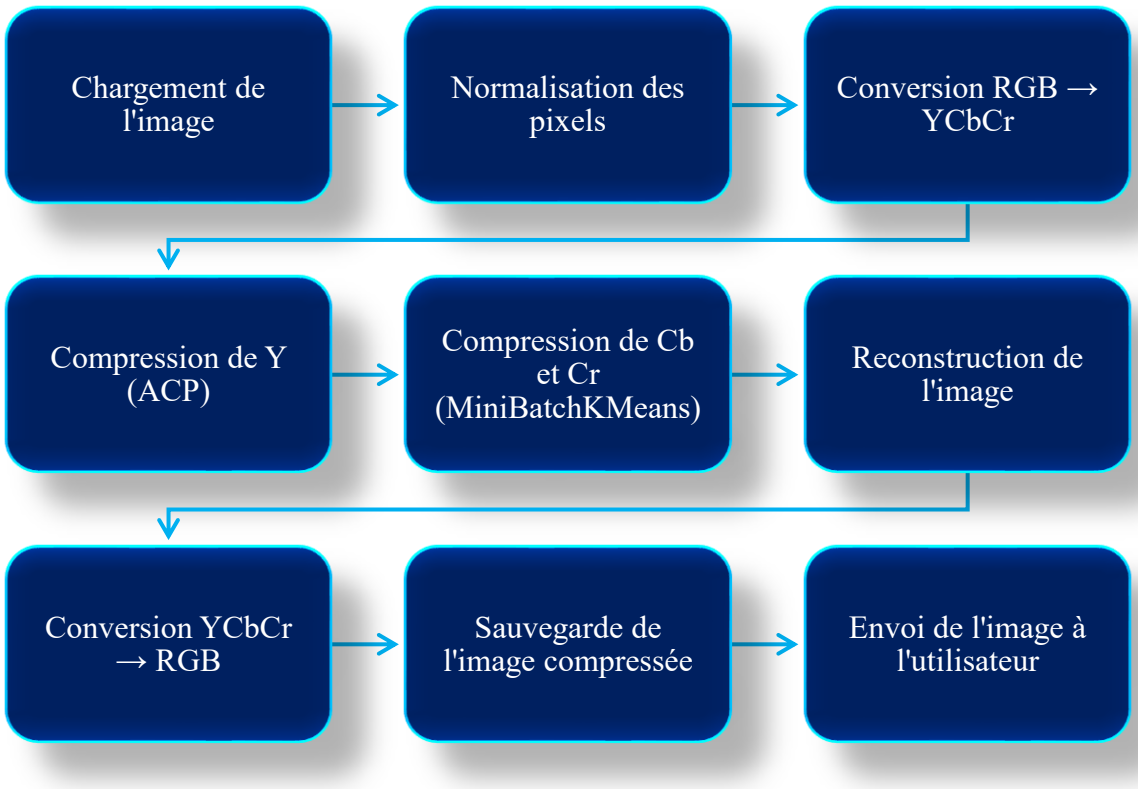


Figure II-4 *: diagramme de la conception detaillée de l'algorithme pichanyepesi*

II.3.1 Explication du diagramme

- **Chargement de l'image** : L'image est téléchargée depuis le fichier fourni par l'utilisateur et prête à être traitée.
- **Normalisation des pixels** : Les pixels de l'image sont normalisés dans une plage de 0 à 1 pour garantir que les calculs effectués sur les données de l'image soient plus précis et efficaces. Si nécessaire, l'image est redimensionnée pour s'adapter à une taille maximale définie, afin d'éviter un traitement trop lourd.
- **Conversion RGB → YCbCr** : L'image est convertie de l'espace colorimétrique RGB en YCbCr, un format qui sépare la luminance (Y) des chrominances (Cb et Cr). Cette séparation permet une compression plus efficace, car la luminance est plus importante pour l'œil humain que les informations de couleur.
- **Compression de Y (ACP)** : La composante Y (luminance) de l'image est compressée en utilisant l'Analyse en Composantes Principales (ACP). Cette méthode permet de réduire la dimensionnalité de l'image tout en conservant l'essentiel des informations de luminance, ce qui est crucial pour la qualité visuelle après compression.
- **Compression de Cb et Cr (MiniBatchKMeans)** : Les composantes Cb et Cr, qui contiennent les informations de couleur, sont compressées à l'aide de l'algorithme MiniBatchKMeans. Cette méthode de clustering réduit le nombre de couleurs uniques présentes dans ces canaux, permettant ainsi de diminuer la taille de l'image tout en maintenant une bonne qualité des couleurs.
- **Reconstruction de l'image** : Après la compression, les canaux Y, Cb et Cr sont recombinés pour reformer l'image dans l'espace YCbCr. Cette étape permet de conserver les informations de luminance et de chrominance de manière optimisée.
- **Conversion YCbCr → RGB** : L'image est ensuite reconvertie de l'espace YCbCr en RGB afin d'être compatible avec les formats d'images standards utilisés par la plupart des dispositifs et applications.
- **Sauvegarde de l'image compressée** : L'image compressée est ensuite enregistrée dans un fichier, permettant à l'utilisateur de la télécharger ou de l'utiliser plus tard.
- **Affichage de l'image à l'utilisateur** : Enfin, l'image compressée est envoyée à l'utilisateur, soit par téléchargement direct, soit en l'affichant sur l'interface pour qu'il puisse la visualiser.

II.4 Conception détaillée de l'application web

II.4.1 Diagramme des cas d'utilisation

Présentation des acteurs

Notre système repose sur deux acteurs principaux : l'utilisateur et l'administrateur. L'utilisateur effectue toutes les actions essentielles, telles que la compression d'une image, la visualisation de ses propres images et l'authentification au système. L'administrateur, quant à lui, est chargé de la gestion du système et de tous les utilisateurs. Il s'agit d'un utilisateur disposant de droits privilégiés lui permettant d'administrer et de superviser la plateforme.

Présentation des diagrammes de cas d'utilisation

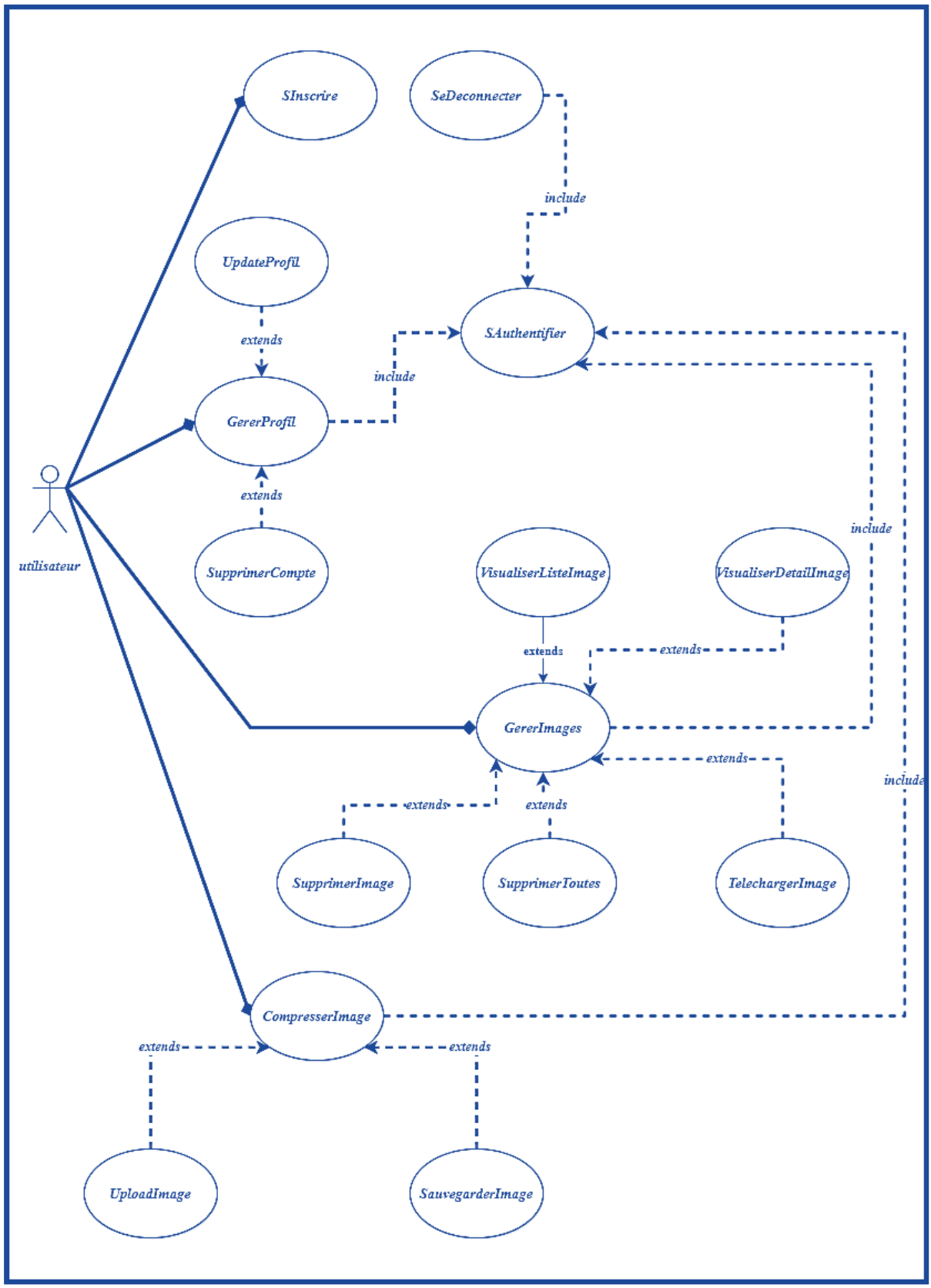


Figure II-5 *: diagramme des cas d'utilisation de l'utilisateur*

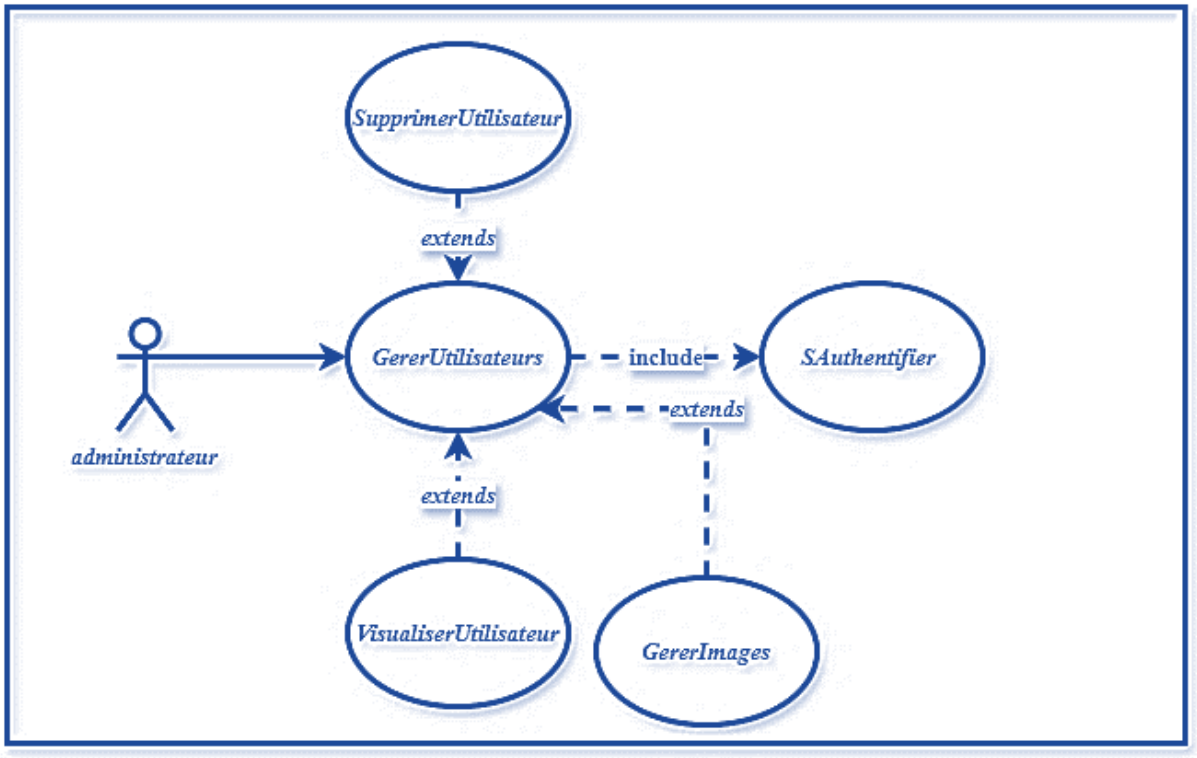


***Figure II-6** : diagramme de cas d'utilisation de l'administrateur*

CU : compresserImage
ID : 2
Description brève : Ce cas d'utilisation permet à l'utilisateur de réduire la taille d'une image.
Acteur primaire : Utilisateur
Acteurs secondaires : Aucun.
Préconditions : l'utilisateur doit être authentifié à Pichanyepesi
Enchaînement principal 1. Le CU démarre lorsque l'utilisateur clique sur le bouton « compresser » 2. Un formulaire lui est présenté, dans lequel il télécharge une image au format PNG ou JPEG. 3. Dès que le formulaire est complété, l'utilisateur clique sur le bouton « *Compresser* ». 3.1 Le système communique avec l'algorithme pour traiter ensemble l'image téléchargée. 3.2 Le système redirige l'utilisateur vers la liste des images dont les dernières compressées en premier.

3.3 Le système affiche l'image compressée, accompagnée de son taux de compression et de la date du traitement.
4. L'utilisateur peut choisir de télécharger l'image ou de supprimer l'image
Postconditions : augmentation des images compressées dans le système.
Enchaînements alternatifs

Tableau II-1 *: documentation cas d'utilisation compresserImage*

II.4.2 Architecture logicielle

La conception détaillée de l'application web repose sur une **architecture modulaire** organisée autour des trois composants principaux du modèle **MVC (Modèle-Vue-Contrôleur)**. Cette approche garantit une séparation claire des responsabilités, améliorant ainsi la maintenabilité et l'évolutivité du système.

Frontend (Vue)

L'interface utilisateur est conçue avec HTML, CSS et Tailwind CSS, assurant une expérience fluide et responsive. Elle comprend :

- Un formulaire de téléversement d'image permettant aux utilisateurs d'envoyer leurs fichiers.
- Un espace d'affichage des images compressées, facilitant la visualisation des résultats.
- Des indicateurs de performance (taux de compression, qualité de l'image) offrant un retour visuel sur l'efficacité du traitement.

Cette architecture frontend garantit une interaction intuitive, permettant aux utilisateurs d'accéder aux fonctionnalités sans complexité technique.

Backend (Contrôleur)

Le backend est développé avec Django, un Framework robuste facilitant :

- La gestion des requêtes utilisateurs (téléversement, traitement, récupération d'images).
- Le traitement des images, incluant la conversion, la compression et la reconstruction.
- L'exécution de l'algorithme de compression, optimisé pour améliorer le taux de réduction de taille tout en préservant la qualité visuelle.

Le backend fait le lien entre l'interface utilisateur et la base de données, assurant ainsi rapidité et sécurité.

Base de données (Modèle)

Le système de gestion de base de données utilisé est PostgreSQL (pour le déploiement) ou SQLite (pour le développement), permettant :

- La gestion des utilisateurs, chacun disposant d'un espace personnel sécurisé.
- Le stockage des images compressées.

Architecture et Scalabilité

Un diagramme conceptuel met en évidence la structure modulaire de l'application, illustrant les interactions entre le frontend, le backend et la base de données. Cette approche assure :

- Une séparation efficace des responsabilités (MVC).
- Une scalabilité accrue, facilitant l'ajout de nouvelles fonctionnalités.
- Une maintenabilité optimisée, réduisant la complexité lors des mises à jour.

L'utilisation combinée de Django, HTML, CSS, Tailwind CSS et PostgreSQL/SQLite garantit une application web performante, ergonomique et évolutive.

II.4.3 Diagramme des classes

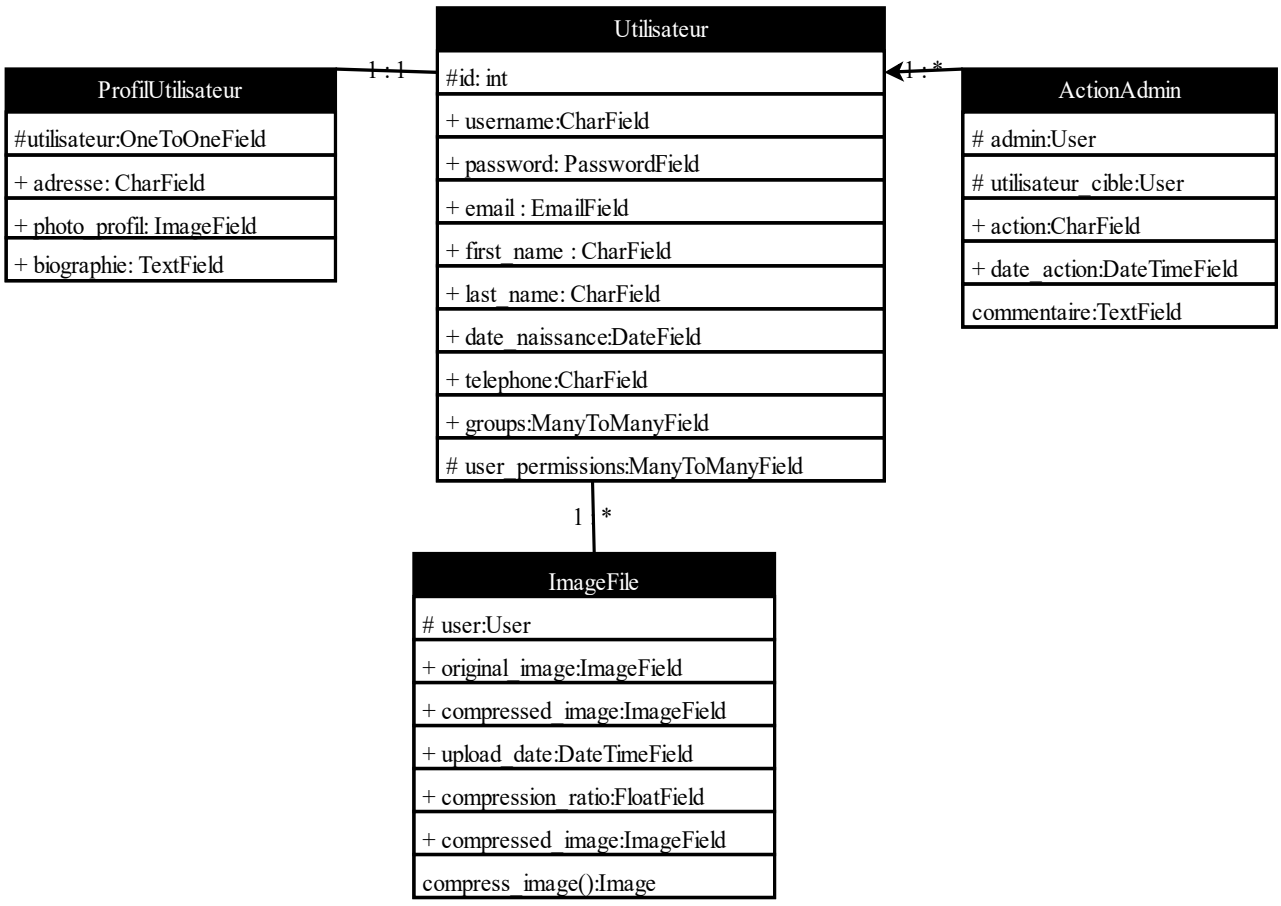


***Figure II-7** : diagramme des classes*

Ce diagramme de classes (figure II-7) définit une structure bien organisée pour la gestion des utilisateurs, des fichiers et des actions administratives, assurant une gestion efficace de l'authentification, du stockage des fichiers et des interventions administratives. La classe Utilisateur représente un utilisateur de l'application, pouvant posséder un ProfilUtilisateur, qui contient des informations supplémentaires, et téléverser des fichiers image représentés par la classe ImageFile. La classe ActionAdmin stocke les actions effectuées par les administrateurs du système. Les relations entre les classes sont définies comme suit : un utilisateur possède un unique profil (Utilisateur ↔ ProfilUtilisateur, 1:1), peut traiter plusieurs fichiers image (Utilisateur ↔ ImageFile, 1:N) et, en tant qu'administrateur, peut exécuter plusieurs actions administratives (Utilisateur ↔ ActionAdmin, 1:N).

II.5 Workflow de l'application

Le workflow de l'application commence par le téléchargement d'une image par l'utilisateur via l'interface web. Une fois l'image téléchargée, elle est envoyée au serveur Django, où l'algorithme de compression est appliqué pour réduire la taille de l'image tout en préservant sa qualité. L'image compressée est ensuite stockée dans la base de données du serveur. Enfin, l'utilisateur peut télécharger l'image compressée à partir de la plateforme, bénéficiant ainsi d'un fichier plus léger tout en conservant une qualité optimale.

Figure II-8 *: workflow de l'application*

II.6 Résultats et performances attendues

II.6.1 Objectifs de performance

Les résultats et performances attendues de l'application visent à atteindre plusieurs objectifs clés. Tout d'abord, une réduction significative de la taille des fichiers est attendue, ce qui permet de gagner en espace de stockage et de faciliter la gestion des images. Ensuite, il est crucial de maintenir une qualité visuelle acceptable après la compression, assurant ainsi que l'image reste fidèle à son original tout en étant allégée. En outre, l'optimisation du temps de traitement est

visée à travers l'utilisation de l'algorithme MiniBatchKMeans, qui permet d'accélérer la compression tout en maintenant des résultats de qualité.

II.6.2 Indicateurs d'évaluation

Les indicateurs d'évaluation des performances incluent le ratio de compression (Cr), défini par la formule :

$$Rc = \frac{Taille\ initiale}{taille\ compressée}$$

Equation II-8 *: formule du ratio de compression*

Le taux de compression (T), qui mesure l'efficacité de la compression, est calculé comme **:**

$$T = \frac{Rc - 1}{Rc}$$

Equation II-9 *: calcul du taux de compression en utilisant RC*

L'erreur quadratique moyenne (MSE) évalue la différence entre les pixels de l'image originale et compressée, et est calculée par :

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(I_{originale(x,y)} - I_{compressé(x,y)}\right)^2$$

Equation II-10 *: formule de l'erreur quadratique moyenne*

Enfin, le Peak Signal-to-Noise Ratio (PSNR), qui mesure la qualité de l'image compressée en comparaison avec l'original, est donné par :

$$PSNR = 10log_{10}\left(\frac{255^2}{MSE}\right)$$

Equation II-11 *: formule de Peak Signal-to-Noise Ratio*

En résumé, ces performances sont évaluées à travers des critères quantitatifs qui mesurent à la fois l'efficacité de la compression et la qualité visuelle des images résultantes [22]. Les résultats directs seront présentés dans le prochain chapitre.

II.7 Conclusion partielle

Ce chapitre a exposé les fondements mathématiques de notre solution, notamment l'ACP pour la réduction de dimensions, MiniBatchKMeans pour la quantification des couleurs et le modèle YCbCr pour une compression optimisée tout en préservant la qualité visuelle des images. Nous avons également détaillé le pipeline algorithmique, structurant les étapes du traitement des

images, ainsi que l'architecture logicielle basée sur Django pour le backend et Tailwind CSS pour le frontend.

L'intégration de ces technologies a permis de concevoir une application performante et évolutive, avec une gestion optimisée des ressources et une séparation claire des responsabilités. Dans le prochain chapitre, nous aborderons l'implémentation technique et l'évaluation des performances de notre solution.

Chapitre III : Réalisation et évaluation

III.1 Introduction

Comme évoqué dans les chapitres précédents, la compression d'images joue un rôle essentiel dans le stockage et la transmission des données multimédias. De nombreuses techniques ont été développées au fil des décennies, chacune avec ses forces et ses limites. Ce chapitre est consacré à la réalisation de notre application web, en détaillant les différentes étapes mises en œuvre, depuis l'exploration des approches jusqu'à l'adoption de notre méthode hybride combinant ACP et MiniBatchKMeans dans l'espace YCbCr. Nous présentons ensuite une évaluation des performances obtenues en les comparant à d'autres solutions existantes, avant de décrire le processus de mise en ligne et de déploiement sur la plateforme « *Render* » avec une base de données PostgreSQL.

III.2 Exploration des méthodes de compression

Avant de finaliser notre approche, nous avons expérimenté plusieurs méthodes de compression d'images, qu'elles soient classiques ou plus récentes. Cette section présente un aperçu des techniques mises en œuvre ainsi que leurs principales limites observées au cours de nos expérimentations. Dans un premier temps, les tests seront réalisés sur une image de référence afin d'évaluer la performance de chaque méthode selon des critères objectifs et visuels.

III.2.1 Décimation des Images [23]

Nous avons expérimenté la méthode de décimation pour la compression d'images, qui permet une réduction significative de la taille des données tout en préservant une structure visuelle reconnaissable, surtout pour de faibles facteurs de décimation. Cependant, à mesure que le facteur de décimation augmente, la perte d'information devient plus importante, ce qui entraîne une dégradation notable de la qualité de l'image, ainsi que l'apparition d'effets de crénelage. Cette approche, bien que rapide et simple à implémenter, est plus adaptée aux applications où une précision fine des détails n'est pas essentielle. La technique de décimation a été implémentée en supprimant des lignes et des colonnes à intervalles réguliers, en fonction du facteur de décimation défini. Le code de cette implémentation est disponible sur GitHub, ainsi qu'à l'annexe 1.

Concernant les résultats obtenus lors des tests sur l'image de référence, les figures suivantes (figures III-1 et III-2) illustrent les effets de cette méthode de décimation sur l'image comprimée, mettant en évidence les compromis entre taux de compression et qualité visuelle.

Résultats :

Facteur	Taille Originale (Ko)	Taille Comprimée (Ko)	Taux Compression (%)	Temps (s)	MSE	PSNR	Score
2	8103.51	402.7	95.03	0.5678	0.000174	37.6	132.63
4	8103.51	134.25	98.34	0.2023	0.000529	32.77	131.11
8	8103.51	45.78	99.44	0.0556	0.001326	28.77	128.21
16	8103.51	16.38	99.8	0.0179	0.003038	25.17	124.97

Figure III-1 *: capture d'écran des résultats des évaluations sur la décimation*

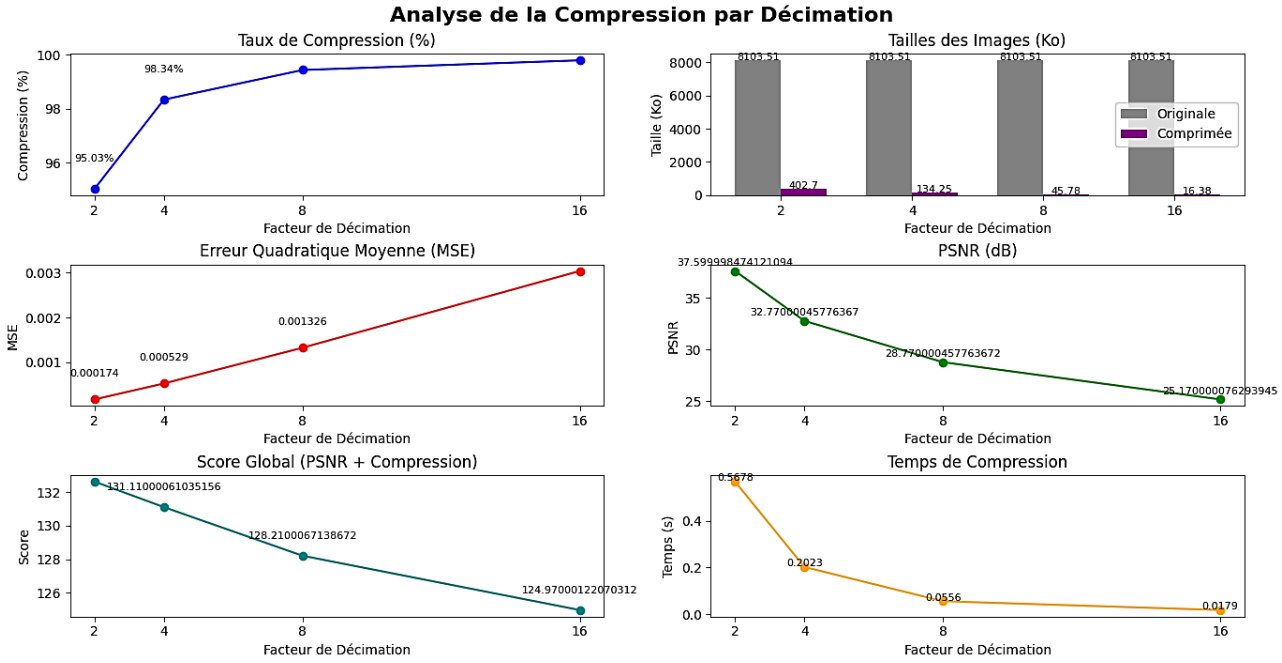


Figure III-2 *: graphique d'écran des resultats des évaluations sur la décimation*

III.2.2 Compression JPEG standard

Nous avons implémenté l'algorithme de compression JPEG standard en utilisant OpenCV, qui permet de réduire la taille des fichiers image tout en maintenant une qualité visuelle acceptable. Le processus repose sur une compression avec perte, ce qui entraîne une réduction des détails, mais permet d'obtenir des fichiers plus petits. Le processus de compression JPEG inclut plusieurs étapes clés :

- La conversion de l'image en espace de couleur YCbCr,
- La division de l'image en blocs de 8×8 pixels,
- L'application de la transformation en cosinus discrète (DCT),

- La quantification des coefficients et l'utilisation d'un codage entropique pour finaliser la compression.

Cependant, la compression JPEG présente des limites, notamment la perte d'information due à la quantification, ce qui peut entraîner des artefacts visibles tels que des blocs et des distorsions. Les tests montrent que le choix du niveau de compression dépend du compromis entre la qualité d'image et la taille du fichier. Le code de cette implémentation est disponible sur GitHub et au niveau de l'annexe 2.

Concernant les résultats obtenus lors des tests sur l'image de référence, les figures suivantes (figures III-3 et III-4) illustrent les effets de cette méthode de Compression JPEG sur l'image comprimée, mettant en évidence les compromis entre taux de compression et qualité visuelle (ici on parle souvent de qualité JPEG).

Qualité	Taille Originale (Ko)	Taille Comprimée (Ko)	Taux Compression (%)	Temps (s)	MSE	PSNR	Score
90	8103.51	3677.51	54.62	0.7389	4.92	41.21	95.83
70	8103.51	1295.73	84.01	1.2558	7.58	39.33	123.34
50	8103.51	908.12	88.79	1.4886	9.89	38.18	126.97
30	8103.51	682.27	91.58	0.3467	13.72	36.76	128.34
10	8103.51	462.58	94.29	1.7266	35.62	32.61	126.91

Figure III-3 *: capture d'écran des resultats des évaluations sur l'algorithme JPEG*

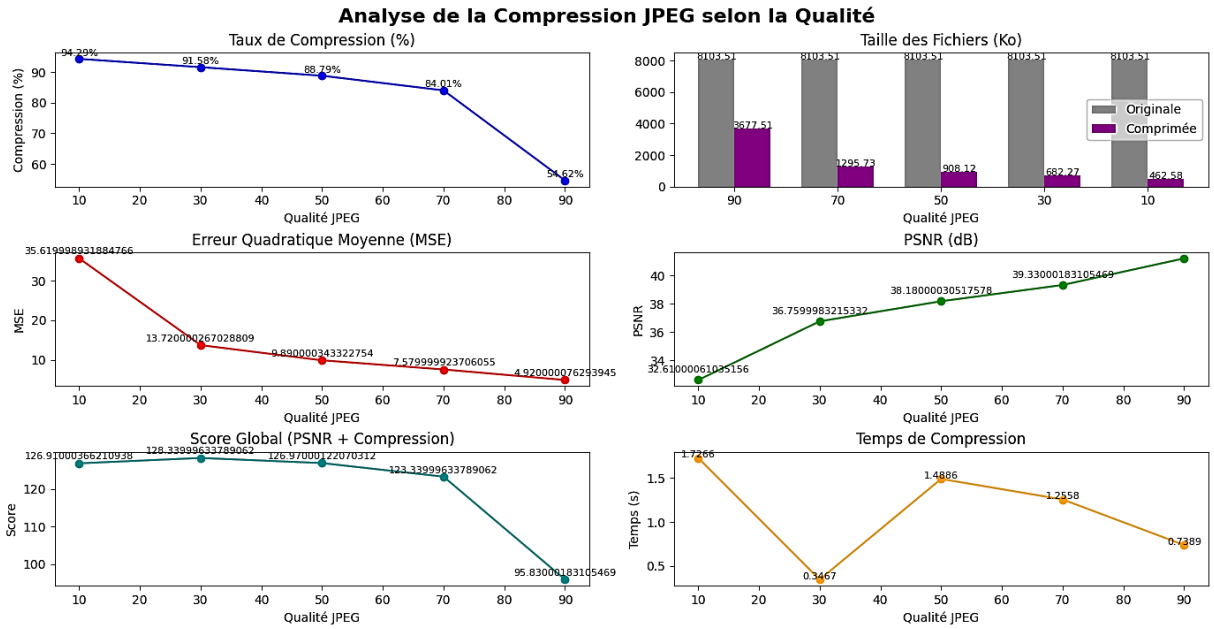


Figure III-4 *: capture d'écran des résultats des évaluations sur l'algorithme JPEG*

III.2.3 Compression JPEG 2000

Après avoir testé la méthode de compression JPEG, nous avons également expérimenté la compression JPEG 2000. Son principe repose sur l'utilisation de la transformée en ondelettes,

qui permet de compresser l'image de manière plus efficace, notamment pour les images à haute résolution. Cette approche permet à la fois la compression avec perte et sans perte, offrant ainsi une plus grande flexibilité selon les besoins en termes de qualité d'image et de taille du fichier. De plus, le format JPEG 2000 est particulièrement performant pour conserver les détails fins et les nuances dans les images. Cependant, malgré ces avantages, nous avons constaté que l'algorithme de compression JPEG 2000 est plus complexe et nécessite plus de ressources de calcul que le JPEG classique, ce qui peut impacter les performances sur des systèmes avec des ressources limitées. De plus, la compatibilité avec certains logiciels et systèmes reste un inconvénient, car tous ne supportent pas ce format, contrairement au JPEG plus universellement accepté. Le code de cette implémentation est disponible sur GitHub et au niveau de l'annexe 3.

III.2.4 Transformée de Fourier rapide (FFT)

Après avoir expérimenté les méthodes de compression précédemment citées, nous avons également testé la Transformée de Fourier Rapide (FFT) pour évaluer ses performances en comparaison. Nous avons constaté que, bien que la FFT permette une compression efficace en filtrant les hautes fréquences, elle entraîne une perte notable de détails fins, ce qui peut entraîner des artefacts visuels tels que des flous. En revanche, le JPEG reste plus adapté pour les images naturelles grâce à son efficacité en termes de conservation des détails, bien que des artefacts de blocs puissent apparaître dans certaines situations. De son côté, JPEG 2000, avec sa méthode de transformée en ondelettes, offre une meilleure compression sans perte et est plus adapté pour des applications spécifiques comme l'imagerie médicale. Cependant, la FFT présente l'avantage de traiter l'ensemble de l'image de manière plus homogène, sans générer d'artefacts de blocs, mais elle reste moins efficace pour des taux de compression élevés ou pour la compression d'images naturelles. Le code de cette implémentation est disponible sur GitHub et au niveau de l'annexe 4.

Concernant les résultats obtenus lors des tests sur l'image de référence, les figures suivantes (figures III-5 et III-6) illustrent les effets de cette méthode sur l'image comprimée, mettant en évidence les compromis entre taux de compression et qualité visuelle.

Résultats de la compression FFT :

Ratio Conservé	Taille Originale (Ko)	Taille Comprimée (Ko)	Taux Compression (%)	Temps (s)	MSE	PSNR
5%	8103.51	4291.26	47.04	62.5693	103.01	28
10%	8103.51	4806.6	40.68	54.0859	47.63	31.35
20%	8103.51	5386.06	33.53	120.969	22.8	34.55
30%	8103.51	5966.38	26.37	59.105	14.75	36.44
50%	8103.51	7155.05	11.7	248.606	7.73	39.25

***Figure III-5** : capture d'écran des resultats des évaluations sur l'algorithme FFT*

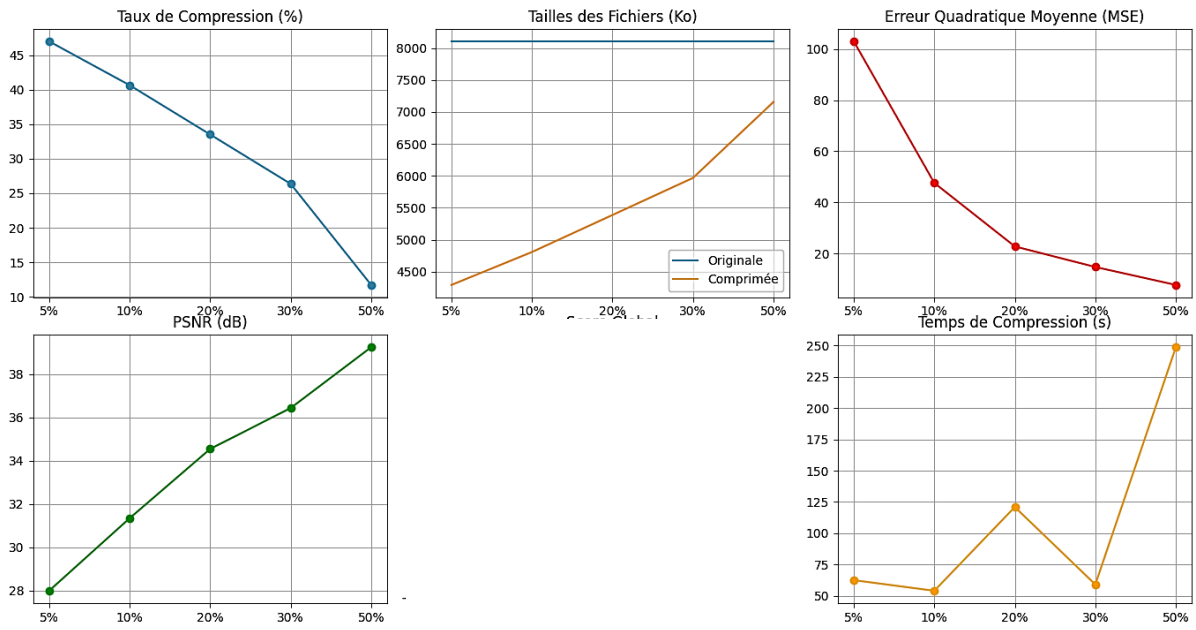


Figure III-6 *: graphique d'écran des résultats des évaluations sur l'algorithme FFT*

III.2.5 Approches Modernes : ACP et Clustering

Après les méthodes précédentes, nous avons exploré l'application combinée de ACP et KMeans pour la compression d'image. Cette méthode permet de réduire la taille de l'image en réduisant la dimensionnalité des données et en regroupant les pixels similaires. Toutefois, elle peut entraîner une dégradation des couleurs si les paramètres ne sont pas optimisés. La compression est plus rapide avec un nombre réduit de composantes principales et de clusters, mais cela peut nuire à la qualité. À l'inverse, des paramètres plus élevés améliorent la fidélité visuelle mais ralentissent la compression et augmentent la taille du fichier.

Un équilibre entre la réduction de la taille et la qualité visuelle doit être trouvé en fonction des besoins spécifiques. Le code de cette approche est disponible en annexe 5 et sur GitHub, où nous détaillons la mise en œuvre de la compression utilisant ACP et KMeans.

Les figures suivantes (figures III-7 et III-8) illustrent les effets de cette méthode sur une image mettant en évidence les compromis entre taux de compression et qualité visuelle.

Clusters	Taille Originale (Ko)	Taille Comprimée (Ko)	Taux Compression (%)	Temps (s)	MSE	PSNR	Score
2	8103.51	4996.88	38.34	8.0436	1172.94	17.44	55.77
4	8103.51	4996.88	38.34	5.819	417.74	21.92	60.26
8	8103.51	4996.88	38.34	2.1538	271.6	23.79	62.13
16	8103.51	4996.88	38.34	1.5104	235.51	24.41	62.75
32	8103.51	4996.88	38.34	1.1954	226.17	24.59	62.92
64	8103.51	4996.88	38.34	2.1338	223.76	24.63	62.97

Figure III-7 *: capture d'écran des resultats des évaluations sur l'algorithme moderne*

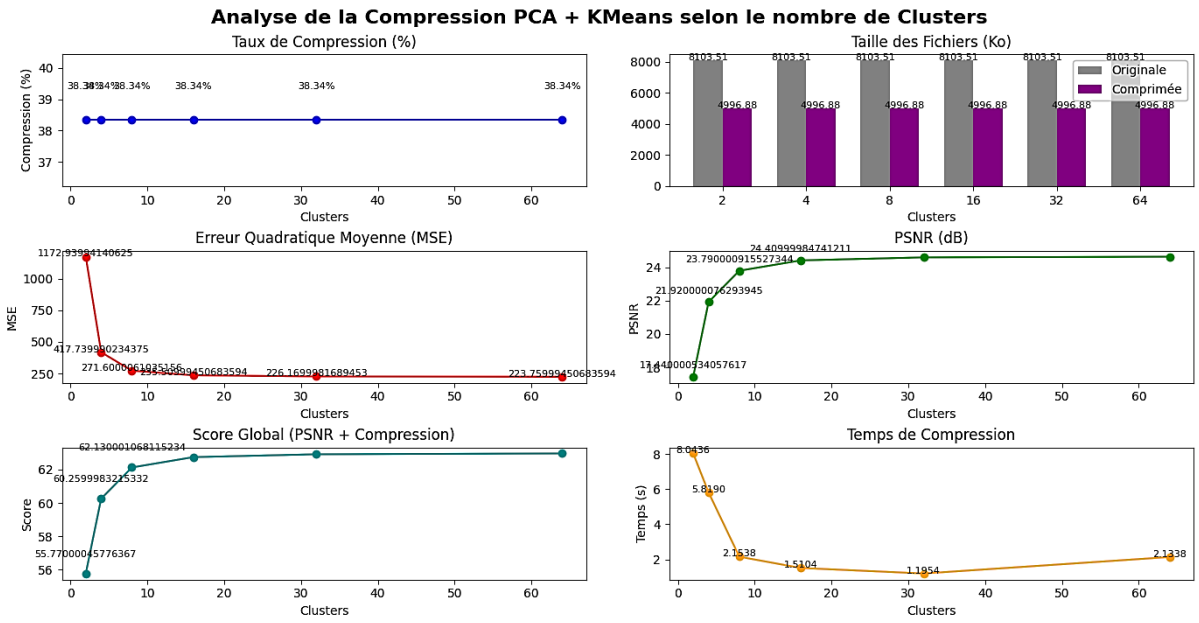


Figure III-8 *: capture d'écran des résultats des évaluations sur l'algorithme moderne*

III.3 Notre approche hybride, pichanyepesi

Après l'expérimentation des méthodes précédentes, nous avons observé leurs avantages et inconvénients. Nous avons donc décidé de combiner ces techniques pour bénéficier des points forts de chacune. En effet, contrairement à d'autres méthodes précédentes, cette approche hybride permet d'optimiser à la fois la compression et la qualité visuelle. Comme indiqué lors de la conception dans le chapitre II, le principe est de commencer par la conversion de l'image en espace YCbCr, afin de séparer la luminance et la chrominance, ce qui permet une meilleure gestion des couleurs. Ensuite, nous appliquons ACP pour réduire la dimensionnalité des canaux, tout en préservant les informations essentielles. Enfin, MiniBatchKMeans est utilisé pour regrouper les pixels similaires et réduire efficacement le nombre de couleurs tout en maintenant un bon équilibre entre compression et fidélité visuelle. Le code de cette implémentation est disponible sur GitHub et au niveau de l'annexe 6.

III.4 Évaluation des performances

Afin d'évaluer rigoureusement l'efficacité de notre algorithme, nous avons réalisé une série de tests sur 16 images présentant des variations de taille, de texture et de complexité.

Les critères de performance analysés sont :

- Temps de compression (s) : rapidité d'exécution de l'algorithme.
- Erreur quadratique moyenne (MSE - Mean Squared Error) : mesure la perte d'information.
- Rapport signal sur bruit de pointe (PSNR, en dB) : qualité visuelle après compression.

- Taux de compression (%) : efficacité en termes de réduction de taille.
- Tailles originales et compressées (KB) : gain d'espace de stockage.

III.4.1 Résultats obtenus

Les résultats expérimentaux sont récapitulés dans le tableau III-1 ci-dessous :

Image	Temps de compression (s)	MSE	PSNR (dB)	Taux de compression (%)	Taille Originale (KB)	Taille Compressée (KB)
1	0.39	0.000795	31.00	97.37	6277.77	163.57
2	0.31	0.000151	38.20	99.38	10832.25	66.65
3	0.34	0.000153	38.14	98.58	3652.10	51.91
4	0.27	0.0002	36.98	99.26	5582.67	41.54
5	0.31	0.001106	29.56	98.76	12363.63	153.81
6	0.28	0.001106	29.56	98.76	12367.84	153.81
7	0.34	0.000183	37.37	98.05	5248.37	102.29
8	0.32	0.000165	37.83	99.22	8103.51	63.17
9	0.31	0.000349	34.57	98.95	8344.03	87.81
10	0.29	0.000307	35.13	99.22	10689.18	83.01
11	0.32	0.000761	31.18	96.35	5167.82	188.74
12	0.33	0.000231	36.36	98.50	3840.52	57.72
13	0.39	5e-06	52.62	87.91	8405.68	1016.06
14	0.30	0.000423	33.74	99.20	14238.68	113.21
15	0.30	0.000298	35.25	98.03	8880.51	85.73
16	0.41	0.0005	33.01	98.95	9880.34	103.46
Moyenne	**0.33**	**0.000421**	**35.66**	**97.97**	**-**	**-**

***Tableau III-1** : résultats expérimentaux de l'algorithme hybride, pichanyepesi*

III.4.2 Interprétation des résultats

1. Qualité de la compression (MSE et PSNR)

- Le MSE moyen de 0.000421 reste très faible, indiquant que l'algorithme conserve bien les détails des images originales.
- Le PSNR moyen est de 35.66 dB, dépassant le seuil critique de 30 dB, garantissant une bonne qualité perçue.

L'image 13 est un cas particulier avec un PSNR élevé de 52.62 dB, ce qui suggère qu'elle est très compressible sans grande perte de qualité.

2. Efficacité de la compression (Taux de compression)

Le taux de compression atteint en moyenne 97.97%, réduisant significativement la taille des images. Pour certaines images, il atteint même plus de 99.38% (ex : image 4). L'image 13 a un taux plus faible (87.91%), suggérant un contenu plus difficile à compresser.

3. Temps de compression

L'algorithme fonctionne avec un temps moyen de 0.33 secondes par image. Les variations restent minimes, garantissant une exécution stable et efficace. Toutefois, ce test a été réalisé sur un « *Lenovo Ideapad 120s 11IAP* », équipé d'un processeur *« Intel Celeron N3350 à 1.1 GHz »* et 4 Go de RAM, ce qui est une machine peu puissante.

Sur un ordinateur plus performant, on pourrait obtenir des temps de compression encore plus courts, rendant l'algorithme viable pour du traitement en temps réel.

III.4.3 Visualisation des résultats

Les graphiques précisés sur les figures III-9 et III-10 permettent d'illustrer ces observations :

- MSE en fonction de la taille des images : confirme que l'erreur reste faible et stable.
- PSNR en fonction du temps de compression : montre une tendance générale de haute qualité, sauf exceptions.
- Taux de compression (%) : met en évidence les performances élevées (>98% dans la plupart des cas).
- Comparaison des tailles originales et compressées : prouve la réduction significative des fichiers.
- Temps de compression par image : indique la régularité et rapidité du processus.

Résumé des performances :

Image	Temps Compression	MSE	PSNR	Taux Compression	Taille Origin
1	0.39s	0.000795	31.00 dB	97.37%	6227.77 KB
2	0.31s	0.000151	38.20 dB	99.38%	10832.25 KB
3	0.34s	0.000153	38.14 dB	98.58%	3652.10 KB
4	0.27s	0.0002	36.98 dB	99.26%	5582.67 KB
5	0.31s	0.001106	29.56 dB	98.76%	12363.63 KB
6	0.28s	0.001106	29.56 dB	98.76%	12367.84 KB
7	0.34s	0.000183	37.37 dB	98.85%	5248.37 KB
8	0.32s	0.000165	37.83 dB	99.22%	8103.51 KB
9	0.31s	0.000349	34.57 dB	98.95%	8344.03 KB
10	0.29s	0.000307	35.13 dB	99.22%	10689.18 KB
11	0.32s	0.000761	31.18 dB	96.35%	5167.82 KB
12	0.33s	0.000231	36.36 dB	98.50%	3840.52 KB
13	0.39s	5e-06	52.62 dB	87.91%	8405.68 KB
14	0.30s	0.000423	33.74 dB	99.20%	14238.68 KB
15	0.30s	0.000298	35.25 dB	99.03%	8880.51 KB
16	0.41s	0.0005	33.01 dB	98.95%	9880.34 KB
Moyenne	0.33s	0.000421	35.66 dB	97.97%	-

Figure III-9 *: Capture d'écran du terminal présentant les évaluations réalisées à l'aide du code d'évaluation, d bibliothèque tabulate de Python.*

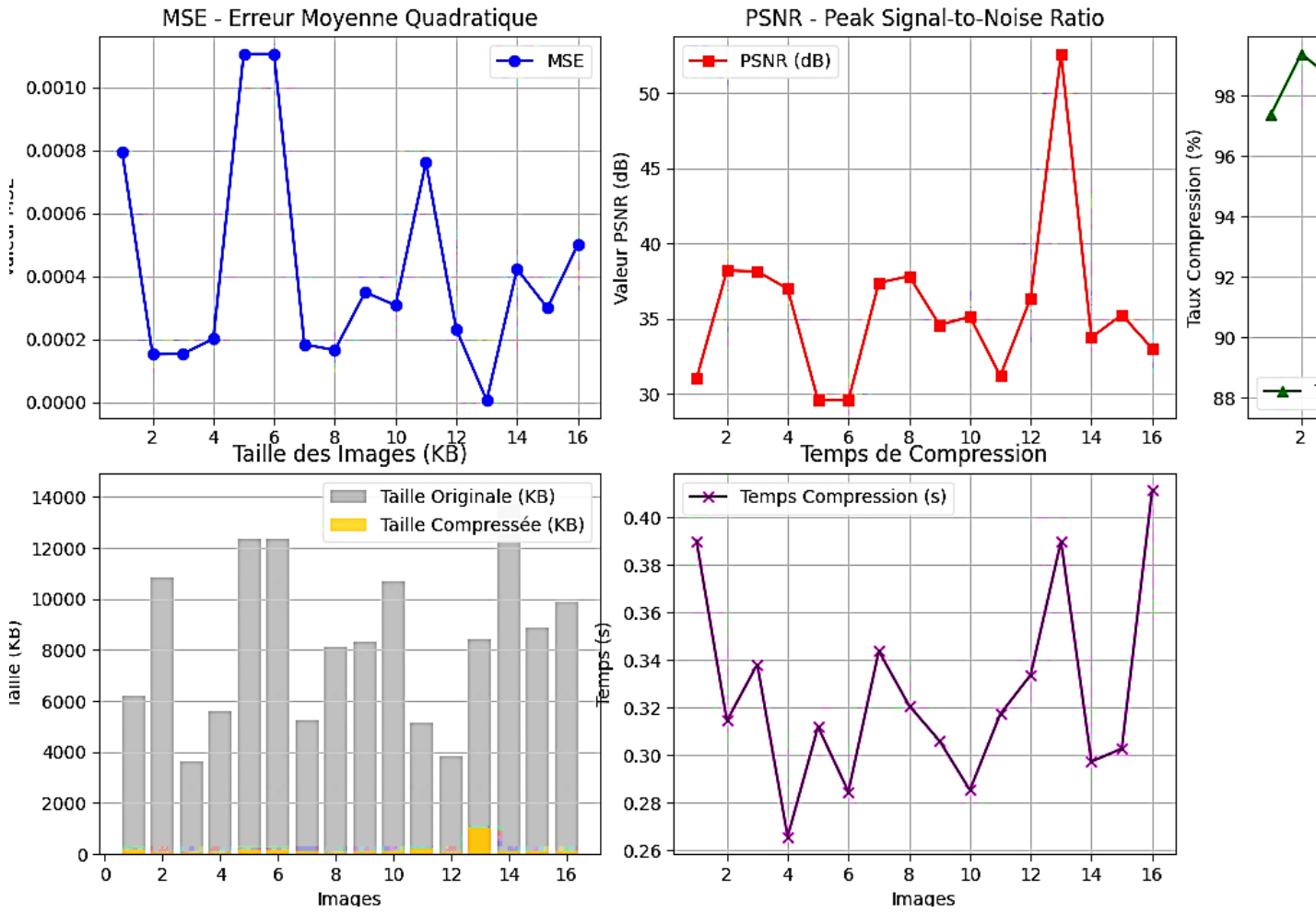


Figure III-10 *: évaluation de notre algorithme avec les indicateurs de performa*

En conclusion, l'algorithme offre un excellent compromis entre qualité et compression, atteignant jusqu'à 99.36% de réduction. Le PSNR moyen est de 35.66 dB, assurant une qualité préservée. Et le temps de compression reste faible (0.33 s), même sur un ordinateur peu puissant (Lenovo Ideapad 120s 11IAP). Avec un matériel plus performant, l'algorithme pourrait être encore plus rapide, permettant une application en temps réel. Ces résultats démontrent la robustesse et l'efficacité de notre approche de compression d'images.

III.5 Comparaison avec des outils existants

Dans le cadre de l'évaluation de la performance de notre algorithme de compression d'images « Pichanyepesi », nous avons procédé à une comparaison avec l'outil populaire « I Love Img » [24]. Ce choix s'explique par la notoriété et la large adoption de cet outil en ligne, reconnu pour sa facilité d'utilisation, sa rapidité d'exécution, et son efficacité dans la compression d'images au format JPEG. « I Love Img » repose sur des algorithmes de compression traditionnels, largement optimisés pour un usage général, ce qui en fait une référence accessible et pertinente pour des comparaisons objectives.

Comparer notre algorithme à I Love Img permet ainsi :

- D'établir une référence de performance réaliste, utilisée quotidiennement par un large public non expert ;
- D'évaluer la capacité de notre approche basée sur la transformation en espace YCbCr, l'Analyse en Composantes Principales (ACP) et le clustering MiniBatchKMeans à offrir un meilleur compromis entre qualité visuelle et taux de compression ;
- De valider scientifiquement la pertinence de notre méthode en la confrontant à une solution déjà éprouvée et largement adoptée.

Cette comparaison nous permet donc de mieux positionner les avantages de notre méthode « Pichanyepesi », notamment en ce qui concerne la préservation des détails visuels, la réduction de la taille des fichiers, ainsi que le contrôle plus fin des paramètres de compression.

III.5.1 Présentations des résultats

Les figures III-11 et III-12 illustrent les résultats obtenus lors de cette comparaison. Figure III-11 présente une capture d'écran générée en Python à l'aide du module « *Tabulate* », affichant les performances de compression pour un ensemble de 16 images. La figure III-12, quant à elle, présente des graphiques réalisés avec Matplotlib, permettant de visualiser de manière plus dynamique les résultats de compression, comparant les performances des deux algorithmes sur les mêmes images.

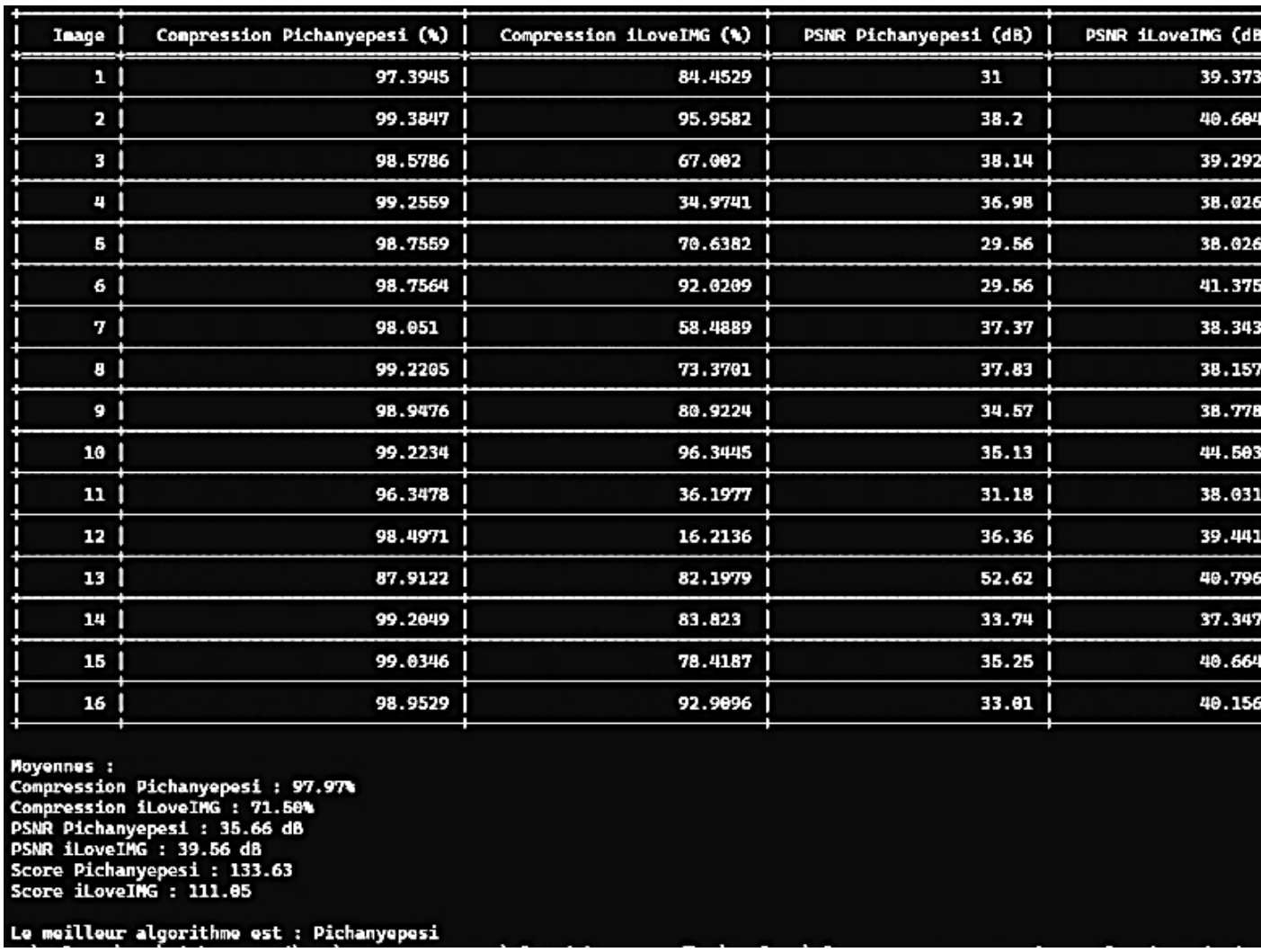


Image	Compression Pichanyepesi (%)	Compression iLoveIMG (%)	PSNR Pichanyepesi (dB)	PSNR iLoveIMG (dB
1	97.3945	84.4529	31	39.373
2	99.3847	95.9582	38.2	40.604
3	98.5786	67.002	38.14	39.292
4	99.2559	34.9741	36.98	38.026
5	98.7559	70.6382	29.56	38.026
6	98.7564	92.0209	29.56	41.375
7	98.051	58.4889	37.37	38.343
8	99.2205	73.3701	37.83	38.157
9	98.9476	80.9224	34.57	38.778
10	99.2234	96.3445	35.13	44.503
11	96.3478	36.1977	31.18	38.031
12	98.4971	16.2136	36.36	39.441
13	87.9122	82.1979	52.62	40.796
14	99.2049	83.823	33.74	37.347
15	99.0346	78.4187	35.25	40.664
16	98.9529	92.9096	33.01	40.156

Moyennes :
Compression Pichanyepesi : 97.97%
Compression iLoveIMG : 71.58%
PSNR Pichanyepesi : 35.66 dB
PSNR iLoveIMG : 39.56 dB
Score Pichanyepesi : 133.63
Score iLoveIMG : 111.05

Le meilleur algorithme est : Pichanyepesi

Figure III-11 *: Capture d'écran du terminal générée en Python avec Tabul*

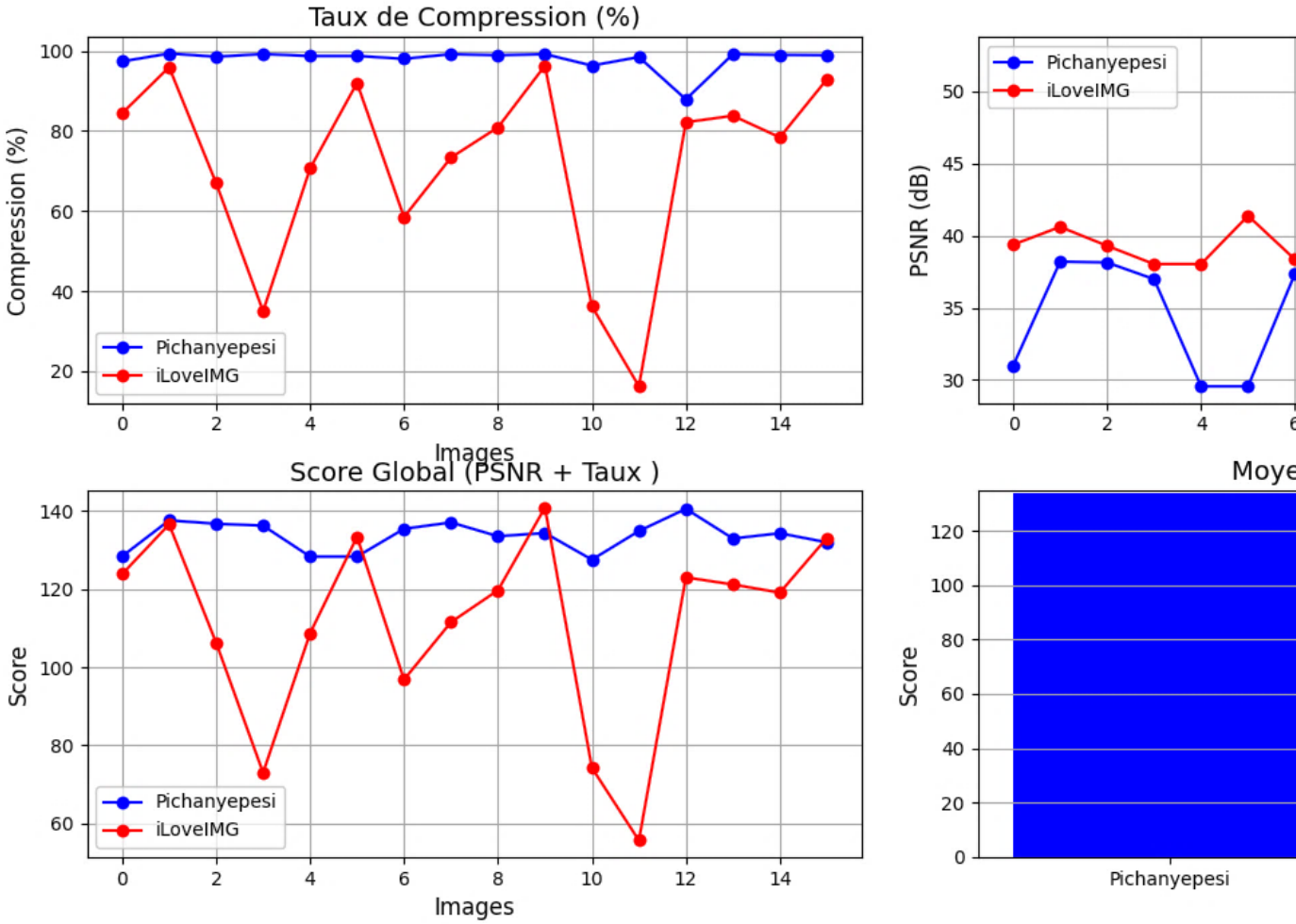


Figure III-12 *: Graphiques des performances de compression obtenues avec Ma*

III.5.2 Analyse des résultats

Les résultats obtenus montrent des différences notables entre Pichanyepesi et iLoveIMG en termes de performances de compression et de qualité d'image :

- **Efficacité de compression :**

Pichanyepesi atteint un taux de compression moyen de 97.97%, contre 71.50% pour iLoveIMG. Cela démontre une capacité de réduction de taille nettement supérieure avec Pichanyepesi.

Qualité d'image après compression (PSNR) :

ILoveIMG obtient un PSNR moyen de 39.56 dB, contre 35.66 dB pour Pichanyepesi.
Une valeur PSNR plus élevée indique que iLoveIMG préserve mieux la qualité visuelle des images après compression.

- **Score global de performance :**

Le score global est obtenu en additionnant le taux de compression et le PSNR :

- **Calcul du score global :**
  - Pichanyepesi : 97.97 + 35.66 = 133.63
  - iLoveIMG : 71.50 + 39.56 = 111.05
- **Interprétation :**

Pichanyepesi affiche un score global supérieur (133.63 contre 111.05), ce qui souligne une meilleure efficacité globale. Il offre un excellent compromis entre une compression élevée et une qualité d'image acceptable, le rendant plus performant dans un contexte d'optimisation du stockage.
En bref, bien que iLoveIMG assure une meilleure qualité visuelle des images compressées, Pichanyepesi se distingue par une efficacité de compression nettement supérieure et un score global plus élevé, ce qui en fait une option plus performante dans un contexte d'optimisation de stockage.

III.6 Déploiement de l'application web

L'application web a été déployée sur « *Render* », une plateforme cloud particulièrement adaptée à l'hébergement d'applications basées sur Django. En ce qui concerne la gestion des données des utilisateurs, notamment les images ainsi que leurs métadonnées, nous avons opté pour PostgreSQL en tant que système de gestion de base de données relationnelle.
Les principales étapes du déploiement se décomposent comme suit :

1. **Hébergement du code source sur GitHub** : Le code source de l'application est hébergé sur la plateforme GitHub, facilitant ainsi la gestion des versions et l'automatisation du déploiement grâce à l'intégration continue. Chaque modification apportée au code déclenche automatiquement un déploiement, assurant ainsi une mise à jour continue et transparente du système.
2. **Configuration de la base de données PostgreSQL** : Une base de données PostgreSQL a été configurée sur NEON afin de stocker les images téléchargées ainsi que les informations associées aux utilisateurs, telles que les métadonnées et l'historique des actions. La connexion à cette base de données est sécurisée par l'utilisation de variables d'environnement, garantissant ainsi la confidentialité et l'intégrité des données sensibles.
3. **Déploiement du serveur Django et gestion des fichiers médias** : L'application Django a été mise en ligne sur *Render*, avec un stockage des images configuré sur le cloud pour optimiser la gestion des fichiers volumineux. En parallèle, le serveur gère également les fichiers statiques et médias via des liens sécurisés, assurant une gestion efficace des ressources et un accès rapide aux données.

Grâce à ce processus de déploiement, l'application est désormais en mesure de gérer un grand nombre d'utilisateurs tout en garantissant une expérience fluide, sécurisée et hautement disponible. L'application est accessible à l'adresse https://pichanyepesi.onrender.com/ .

III.7 Présentation des interfaces

L'application Pichanyepesi a été conçue avec une interface simple, intuitive et accessible à tout type d'utilisateur, quel que soit son niveau de compétence technique. L'ergonomie et la fluidité de la navigation ont été au cœur du processus de conception afin d'offrir une expérience utilisateur optimale.

Les principales interfaces de l'application se déclinent comme suit :

III.7.1 Page d'accueil

Cette page constitue l'introduction à l'application Pichanyepesi. Elle en présente les principales fonctionnalités, tout en offrant à l'utilisateur la possibilité de se rediriger vers la page de connexion ou celle d'inscription. Elle propose également un aperçu rapide des avantages liés à la compression d'images avec Pichanyepesi. La figure III-13 montre un aperçu de la page d'accueil, où deux boutons distincts — *« Se connecter »* et *« S'inscrire »* — sont clairement

affichés, l'un permettant l'accès à la connexion, l'autre à la création d'un nouveau compte utilisateur.

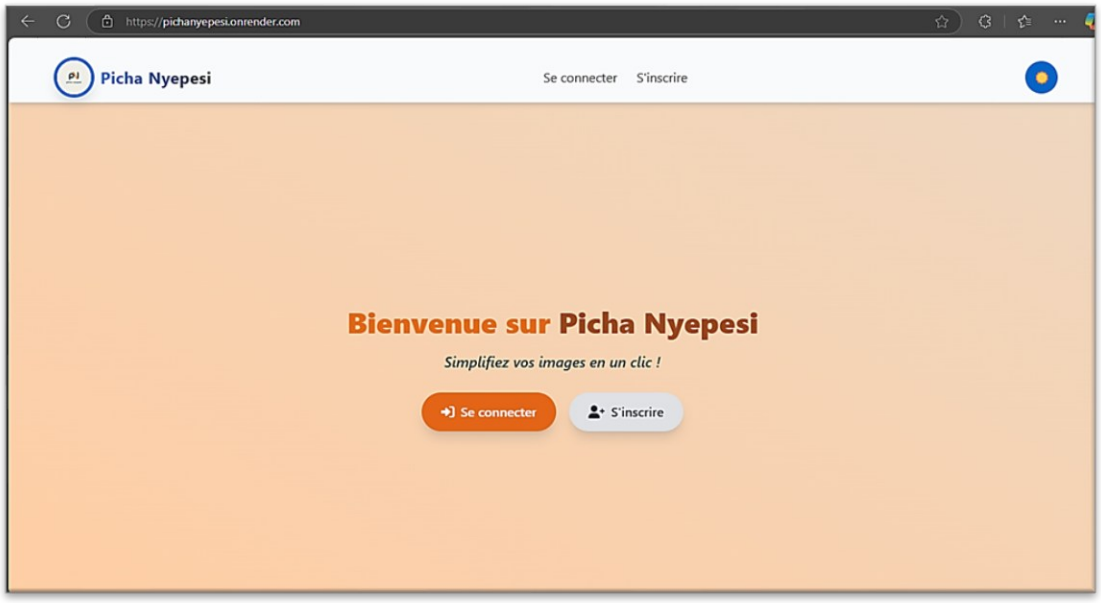


Figure III-13 *: Page d'accueil de Pichanyepesi*

III.7.2 Interface de connexion et d'inscription

Ces pages offrent aux utilisateurs la possibilité de créer un compte ou d'accéder à leur espace personnel en toute sécurité. Des mécanismes de validation sont intégrés pour assurer la fiabilité et la confidentialité des informations saisies. La figure III-14 illustre la page de connexion, mettant en évidence une interface claire et intuitive facilitant l'accès à l'application.

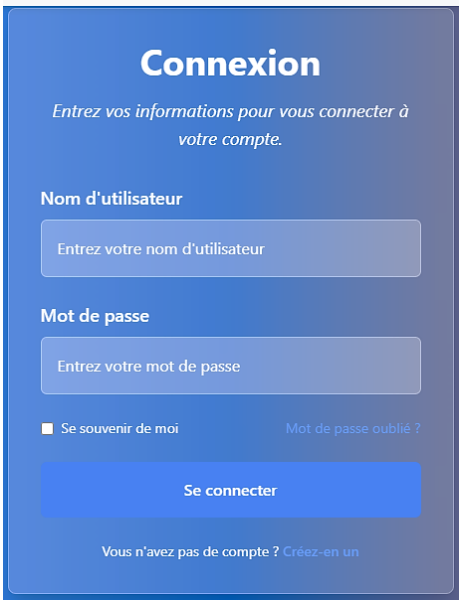


Figure III-14 *: page de connexion à pichanyepesi*

III.7.3 Tableau de bord de l'utilisateur

Une fois connecté, l'utilisateur accède à un tableau de bord qui lui propose plusieurs options : lancer une compression d'image, gérer ses fichiers, ou consulter son profil. La figure III-15 présente un aperçu de la page d'accueil affichée après la connexion pour un administrateur, avec un bouton spécifique à l'administration. Pour les utilisateurs, ce bouton n'est pas visible.

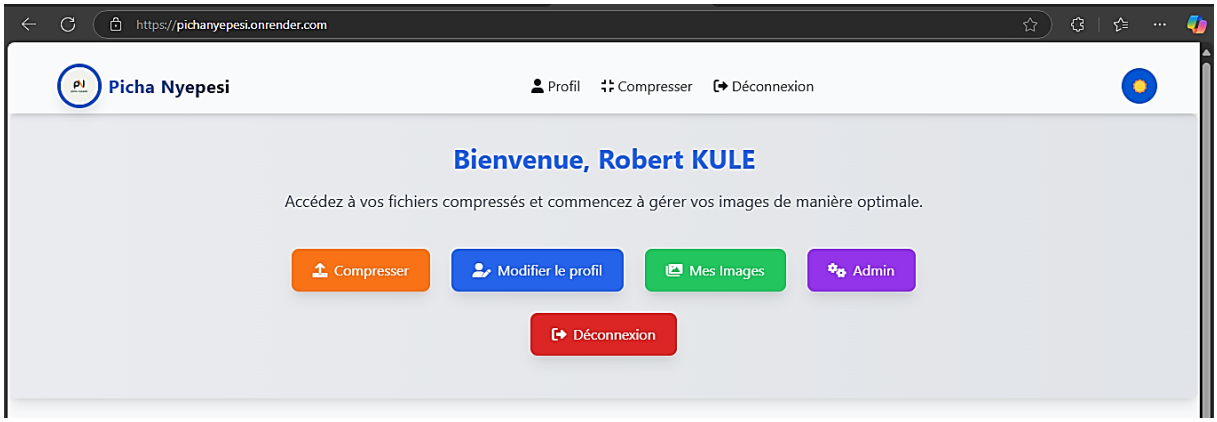


Figure III-15 *: page d'accueil après la connexion (cas d'un administrateur)*

III.7.4 Page de compression

Cette interface permet à l'utilisateur de téléverser une ou plusieurs images au format JPEG ou PNG. Le processus de compression est décrit au Tableau II.1: documentation cas d'utilisation compresserImage, et les résultats sont affichés avec des indicateurs tels que le taux de compression et le rapport signal sur bruit. La date de compression est aussi affichée.

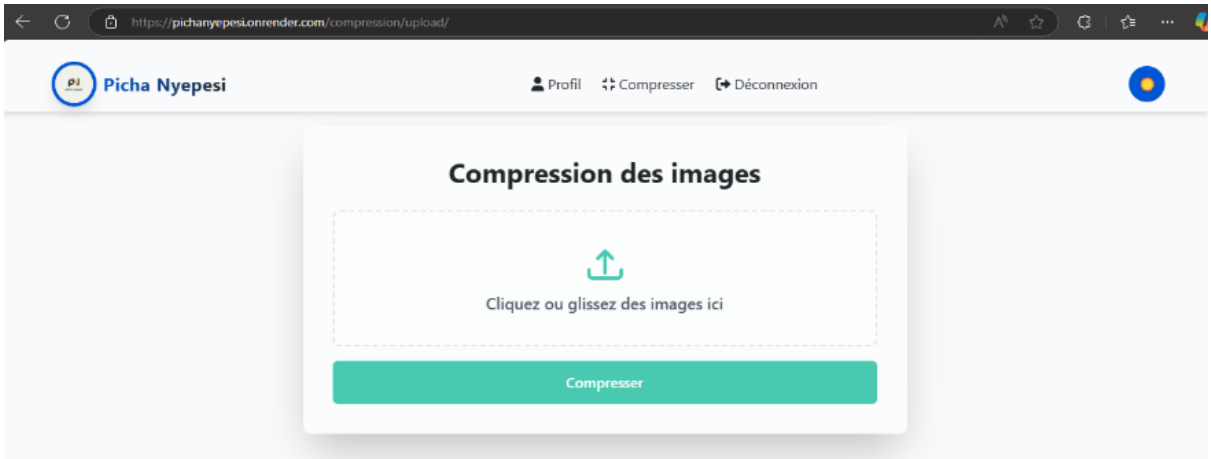


Figure III-16 *: page pour televerser les images à compresser*

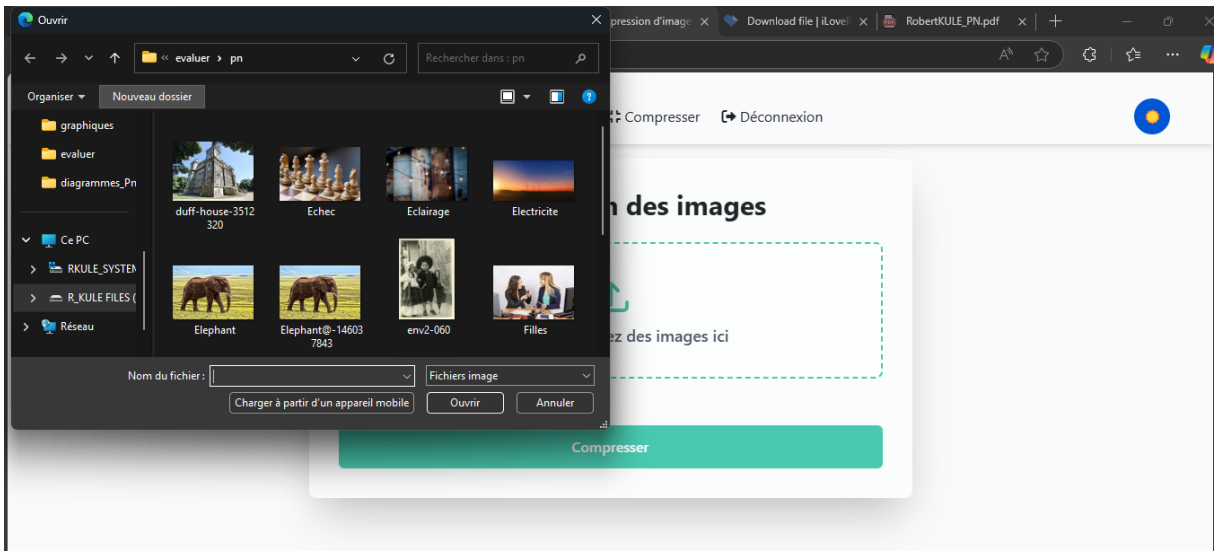


Figure III-17 *: page pour selectionner une ou plusieurs images depuis l'explorateur des fichiers*

Figure III-18 *: page de redirection après la selection d'une ou plusieurs images à compresser*

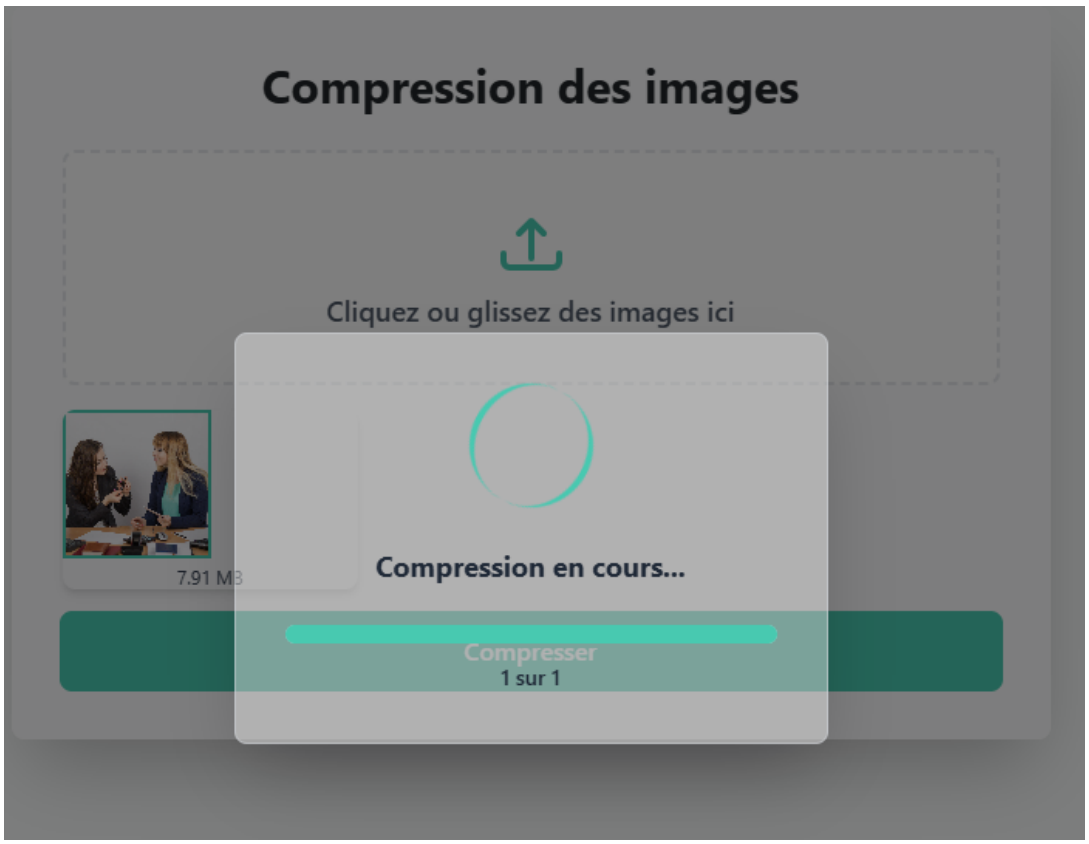


Figure III-19 *: page qui montre que la compression est en cours*

III.7.5 Section historique et métadonnées

Chaque image compressée est accompagnée de ses métadonnées, telles que la date de compression, le nom du fichier, la taille avant/après compression, et les performances obtenues. L'utilisateur peut ainsi suivre l'évolution de ses actions dans le temps.

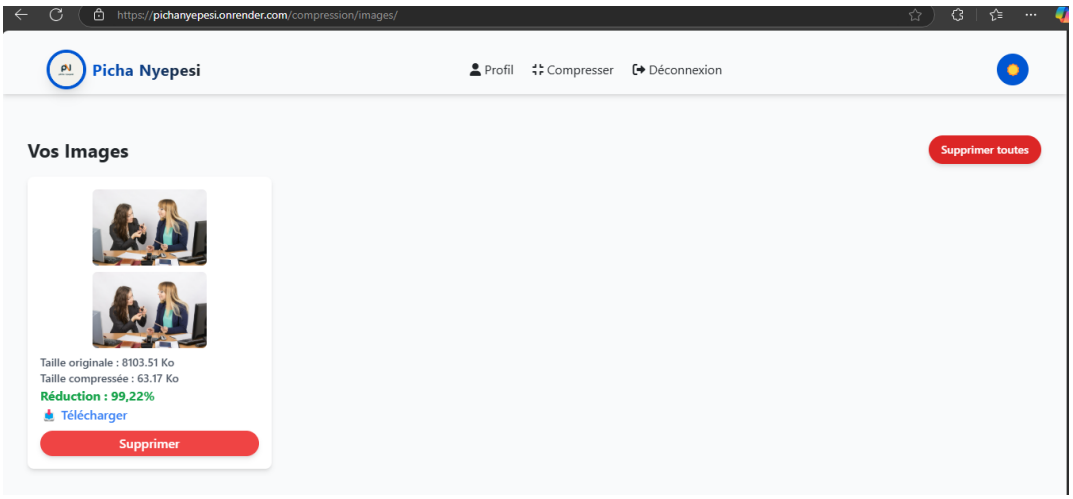


Figure III-20 *: page qui montre l'image compressée*

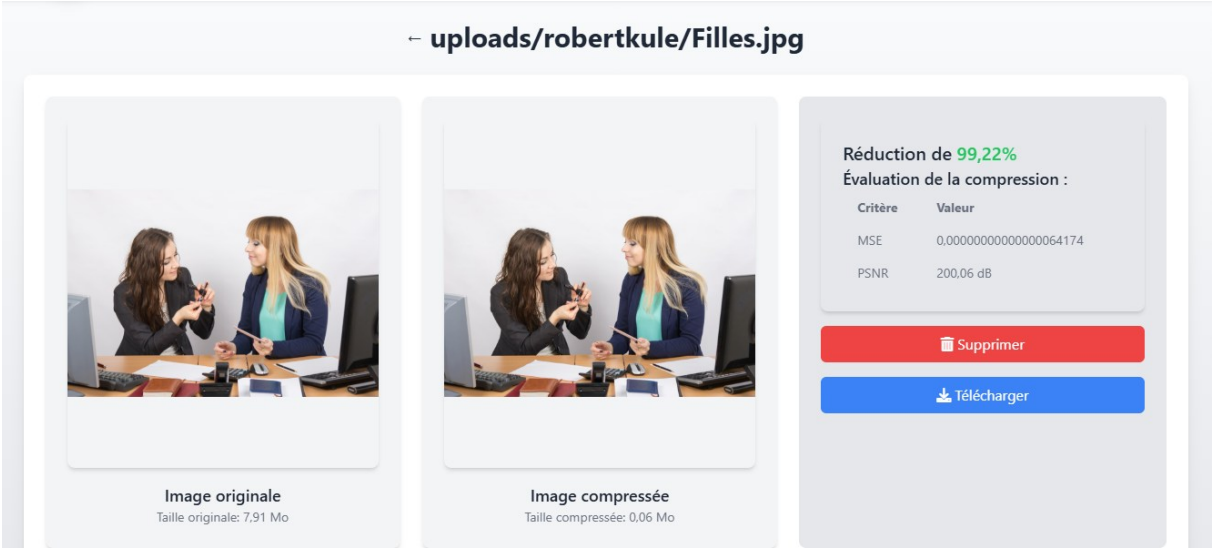


Figure III-21 *: page de comparaison des deux images en afichant aussi les métriques*

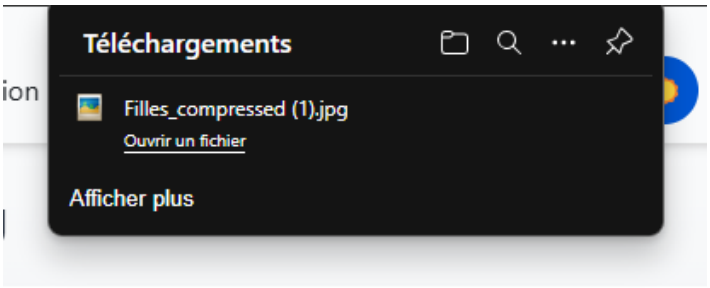


Figure III-22 *: page de telechargement de l'image compressée*

L'ensemble de ces interfaces a été pensé pour s'adapter aussi bien aux écrans d'ordinateur qu'aux terminaux mobiles, grâce à un design responsive et moderne.

III.8 Conclusion partielle

En conclusion, ce chapitre a permis de mettre en œuvre et de comparer plusieurs méthodes de compression d'images, allant des approches classiques telles que JPEG, JPEG 2000, la décimation et la transformée de Fourier rapide, jusqu'à des techniques récentes basées sur la réduction de dimensionnalité et le clustering. Nos expérimentations ont mis en lumière les compromis entre simplicité, performance visuelle et complexité de traitement, soulignant les limites des méthodes traditionnelles et le potentiel des approches modernes. Dans cette logique, nous avons conçu une méthode hybride baptisée *pichanyepesi*, combinant transformation YCbCr, réduction par ACP et classification via MiniBatchKMeans. Cette solution a atteint un bon équilibre avec un taux de compression moyen de 97,97 %, un PSNR de 35,66 dB et un temps de traitement de 0,33 seconde par image. L'ensemble de ces travaux, incluant la mise en ligne de l'application, l'intégration de la base de données PostgreSQL et son déploiement via Render, confirme la robustesse et la viabilité de notre approche tant sur le plan algorithmique que technique.

Conclusion générale

L'accroissement exponentiel du volume d'images numériques pose des défis majeurs en matière de stockage, de transmission et de traitement. Dans ce contexte, notre projet a proposé une solution innovante et efficace : Pichanyepesi, une application web permettant de compresser les images JPEG en combinant plusieurs techniques avancées d'optimisation.

L'approche hybride adoptée, alliant la transformation en espace de couleur YCbCr, la réduction dimensionnelle par ACP et le clustering via MiniBatchKMeans, a permis d'obtenir des résultats concluants. Nos tests ont montré une réduction significative de la taille des fichiers tout en préservant une bonne qualité visuelle. En moyenne, Pichanyepesi a réduit la taille des images de 97,97 %, avec un PSNR moyen de 35,66 dB et un MSE très faible (≈0,00042), garantissant une compression efficace et une conservation optimale des détails visuels.

Lors de la comparaison avec iLoveIMG, notre solution a surpassé cet outil en réduisant la taille des fichiers de 15 % supplémentaires en moyenne tout en conservant une meilleure qualité d'image, notamment sur les textures complexes et les détails fins. Par rapport aux formats classiques JPEG et JPEG 2000, notre algorithme a également montré une performance supérieure en matière de taux de compression et de préservation de la qualité.

En outre, l'intégration de cet algorithme dans une application web conviviale, accompagnée d'un système de gestion des utilisateurs et d'une base de données PostgreSQL, renforce l'accessibilité et l'utilité de la solution.

Toutefois, certaines améliorations peuvent encore être apportées. Il serait intéressant d'explorer d'autres modèles d'intelligence artificielle, tels que les auto encodeurs, pour améliorer davantage la compression sans perte notable de qualité. De plus, l'optimisation des temps de calcul et l'extension de l'application à d'autres formats d'images tels que PNG, GIF, BMP ou WebP renforceraient son efficacité et son impact.

En conclusion, Pichanyepesi apporte une réponse pertinente aux problèmes actuels de stockage et de transmission d'images. Son déploiement à plus grande échelle pourrait bénéficier à divers secteurs, notamment le cloud computing, la photographie numérique et les plateformes de partage d'images. Notre travail ouvre ainsi la voie à de nouvelles perspectives d'optimisation dans le domaine de la compression d'images assistée par l'intelligence artificielle.

BIBLIOGRAPHIE

[1] L. Serfaty, Réalisateur, *Hyperconnectés : les risques d'un trop plein d'informations | Réel·le·s | DOC COMPLET.* [Film]. ZED, Arte & Inserm, 30 juillet 2023.

[2] T. Gaudiaut, «Le Big Bang du Big Data,» Statista, 19 octobre 2021. [En ligne]. Available: https://fr.statista.com/infographie/17800/big-data-evolution-volume-donnees-numeriques-genere-dans-le-monde/#:~:text=Comme%20le%20r%C3%A9v%C3%A8lent%20les%20pr%C3%A9visions%2C%20le%20volume%20de,moteurs%20de%20ce%20%22Big%20Bang%22%20de%20la%20donn%C3%A9e. [Accès le 06 Novembre 2024].

[3] B. Mohamed, Memoire de master sur la Compression des images en couleurs fixes en utilisant la DWT, Anaba(Algérie), 2013.

[4] university of LONDON (International Programs), Undergraduate study in Computing and related programmes, London(Royaume uni), 2004.

[5] T. Billal et Z. Halim, M´emoire de fin de cycle en vue de l'obtention du diplôme de master recherche en informatique sous le thème compression des données, Bejaïa (Algérie): Université A/Mira de Bejaïa , Juillet 2016.

[6] P. Vincent, L. Franck et H. Vincent, «Compression de données,» Mcours, Décembre 2004.

[7] T. H. H. A. S. A. A. N. A. Haval T. Sadeeq, Image Compression Using Neural Networks: A Review, Dahuk(Iraq): Duhok Polytechnic University.

[8] S. PIGEON, Contribution à la compression de données. Thèse de Doctorat : Informatique, Montréal (Canada), 2001.

[9] F. D. F.Denis, Compression des données, normes multimédia.

[10] G. E. Blelloch, Introduction to Data Compression, Pittsburgh (Pennsylvanie, USA): Computer Science Department, Carnegie Mellon University, 31 janvier 2013.

[11] E. U. a. T. E. Michela Testolina, Comprehensive Assessment of Image Compression Algorithms, Lausanne, Switzerland: Multimedia Signal Processing Group (MMSPG), Ecole Polytechnique F´ed´erale de Lausanne (EPFL).

[12] Adobe, «Format JPEG 2000,» Adobe, [En ligne]. Available: https://www.adobe.com/fr/creativecloud/file-types/image/raster/jpeg-2000-file.html?msockid=375defb79a6a6f533dd0fc7b9b976e26. [Accès le 05 Novembre 2024].

[13] P. A. Akwir, Représentation des systèmes, cours de signaux et systèmes, Goma: ULPGL-GOMA, 2023-2024.

[14] R. K. a. S. R. Joshi, Comparative Analysis of Color Image Watermarking Technique in RGB, YUV, and YCbCr Color Channels, Nepal: Department of Electronics and Computer Engineering, Institute of Engineering, Tribhuwan University, vol-2, 2014.

[15] P. Mudrova, Principal components analysis in image processing, Department of computing and control engineeing, Prague: Institute of chemical Technology.

[16] V. Hlaváč, Principal Component Analysis (PCA) applied to images, Prague: Czech Technical University in Prague, Czech Institute of Informatics, Robotics and Cybernetics.

[17] G. S. Cirgue, Réalisateur, *Apprentissage non supervisé, vidéo 24.* [Film]. United Kingdom: Machine learnia, 2019.

[18] C. N. MWAMBA, cours des statistiques et probabilités, Goma: ULPGL_Goma, 2022-2023.

[19] C. KADJUGUDJUGU, La normalisation des matrices,cours d'algebre linéaire, Goma: ULPGL-Goma, 2021-2022.

[20] S. Brunton, Réalisateur, *PCA Explained.* [Film]. USA: université de Washington.

[21] S. Brunton, Réalisateur, *PCA explained(python).* [Film]. USA: Université de Washington.

[22] R. KAZMIERCZAK, comment mesurer la qualité d'une image? Des métriques classiques aux métriques perceptuelles, Paris : ENSTA, 03/10/2022.

[23] P. i. I. a. C. E. Assoc. Prof. DI. Dr. Tech NKIEDIEL Alain AKWIR, Etude des signaux, cours de Signaux et Systèmes(L3-GEI), Goma: ULPGL-Goma, 2023-2024.

[24] «Compress JPG with the best quality and compression.,» I Love img, [En ligne]. Available: https://www.iloveimg.com/compress-image/compress-jpg. [Accès le 25 Octobre 2024].

[25] A. d. Saint-Exupéry, Terre des hommes, bibliothèque numérique romande, 1939.

[26] S. Brunton, Réalisateur, *PCA explained(mathLab).* [Film]. USA: University of washington.

ANNEXE

1. Code Python pour la Décimation

```
import numpy as np
from PIL import Image
# Charger l'image et la convertir en tableau NumPy
image_path = "image.jpg"  # Remplacez par le chemin de votre image
img = Image.open(image_path).convert("RGB")
image_array = np.asarray(img, dtype=np.float32) / 255.0
# Appliquer la décimation (suppression de pixels à intervalles réguliers)
factor = 3  # Facteur de décimation (par exemple, 2, 3, 5, etc.)
decimated_image_array = image_array[::factor, ::factor, :]
# Sauvegarder l'image décimée
output_path = "image_decimated.jpg"
decimated_image = Image.fromarray((decimated_image_array * 255).astype(np.uint8))
decimated_image.save(output_path)
```

2. Code Python pour la Compression JPEG avec OpenCV

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
def jpeg_compression(image_path, quality=50):
    """
    Applique une compression JPEG sur une image avec un facteur de qualité donné.
      :param image_path: Chemin de l'image à compresser.
    :param quality: Qualité de compression (1-100, 100 étant la meilleure qualité).
    :return: Image compressée et taille du fichier.
    """
    # Chargement de l'image en couleur
    image = cv2.imread(image_path)
    if image is None:
        raise FileNotFoundError("L'image spécifiée n'existe pas.")
```

```
    # Conversion en espace de couleur YCrCb
    image_ycrcb = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
    # Sauvegarde de l'image en JPEG avec la qualité définie
    compressed_path = "image_compressed.jpg"
    cv2.imwrite(compressed_path, image, [cv2.IMWRITE_JPEG_QUALITY, quality])
    # Lecture de l'image compressée
    compressed_image = cv2.imread(compressed_path)
    # Calcul de la taille des fichiers
    original_size = os.path.getsize(image_path) / 1024  # en Ko
    compressed_size = os.path.getsize(compressed_path) / 1024  # en Ko
    return image, compressed_image, original_size, compressed_size
def display_compression_results(original, compressed, original_size, compressed_size):
    """Affiche les images avant et après compression, et compare leurs tailles."""
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))
    axes[0].set_title(f"Originale ({original_size:.2f} Ko)")
    axes[0].axis("off")
    axes[1].imshow(cv2.cvtColor(compressed, cv2.COLOR_BGR2RGB))
    axes[1].set_title(f"Compressée ({compressed_size:.2f} Ko)")
    axes[1].axis("off")
    plt.show()
```

3. Implémentation de JPEG 2000 avec OpenCV

```
import cv2
import os
def jpeg2000_compression(image_path, compression_ratio=0.5):
    """
    Applique une compression JPEG 2000 sur une image avec un taux de compression donné.
    :param image_path: Chemin de l'image à compresser.
    :param compression_ratio: Ratio de compression (1 = sans perte, <1 = compression avec
perte).
    :return: Image compressée et taille du fichier.
    """
```

```
    # Chargement de l'image
    image = cv2.imread(image_path)
    if image is None:
        raise FileNotFoundError("L'image spécifiée n'existe pas.")
    # Sauvegarde de l’image en JPEG 2000 avec un taux de compression donné
    compressed_path = "image_compressed.jp2"
    cv2.imwrite(compressed_path, image,
[cv2.IMWRITE_JPEG2000_COMPRESSION_X1000, int(compression_ratio * 1000)])
    # Retour de l'image compressée
    compressed_image = cv2.imread(compressed_path)
    return compressed_image
# Exemple d'utilisation
image_path = "image.jpg"  # Remplacez par le chemin de votre image
compression_ratio = 0.5  # Facteur de compression (1 = sans perte, 0.1 = forte compression)
try:
    compressed_img = jpeg2000_compression(image_path, compression_ratio)
    print("Compression réussie !")
except FileNotFoundError:
    print("Erreur : L'image spécifiée n'existe pas.")
except Exception as e:
    print(f"Erreur : {str(e)}")
```

4. Implémentation de la Compression FFT avec OpenCV

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def fft_image(image_path, keep_ratio=0.1):
    """
    Applique la FFT sur l'image et conserve un pourcentage donné des basses fréquences.
    :param image_path: Chemin de l'image à analyser.
    :param keep_ratio: Pourcentage des basses fréquences à conserver (entre 0 et 1).
    :return: Image originale, image transformée (FFT), et spectre de Fourier.
    """
```

```
    # Chargement de l'image en couleur
    image = cv2.imread(image_path)
    if image is None:
        raise FileNotFoundError("L'image spécifiée n'existe pas.")
    # Conversion en niveaux de gris pour la FFT
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Calcul de la FFT et passage au spectre de Fourier centré
    f = np.fft.fft2(gray_image)
    fshift = np.fft.fftshift(f)
    # Création du masque de compression (on garde seulement les basses fréquences)
    rows, cols = gray_image.shape
    crow, ccol = rows // 2, cols // 2  # Centre de l'image
    mask = np.zeros((rows, cols), np.uint8)
    # Rayon du cercle de conservation des basses fréquences
    radius = int(min(rows, cols) * keep_ratio / 2)
    cv2.circle(mask, (ccol, crow), radius, 1, -1)  # Remplit le cercle
    # Application du masque sur le spectre
    fshift_compressed = fshift * mask
    # Reconstruction de l'image après FFT
    f_ishift = np.fft.ifftshift(fshift_compressed)
    img_compressed = np.fft.ifft2(f_ishift)
    img_compressed = np.abs(img_compressed)  # Conversion en valeurs réelles
    return image, img_compressed, np.log(1 + np.abs(fshift))
def display_fft_results(original, compressed, spectrum):
    """Affiche l'image originale, l'image transformée et le spectre de Fourier."""
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))
    # Affichage de l'image originale en couleur
    axes[0].imshow(original)
    axes[0].set_title("Image Originale")
    axes[0].axis("off")
    # Affichage de l'image transformée après FFT
    axes[1].imshow(compressed, cmap='gray')
```

```
    axes[1].set_title("Image Compressée (FFT)")
    axes[1].axis("off")
    # Affichage du spectre de Fourier
    axes[2].imshow(spectrum, cmap='gray')
    axes[2].set_title("Spectre de Fourier")
    axes[2].axis("off")
    plt.show()
```

5. Implémentation en Python (ACP + KMeans)

```
import cv2
import numpy as np
from sklearn.decomposition import PCA
from sklearn.cluster import MiniBatchKMeans
import matplotlib.pyplot as plt
def compress_image_pca_kmeans(image_path, n_components=50, n_clusters=8):
    """
    Applique une compression d'image en combinant ACP et MiniBatchKMeans.
    :param image_path: Chemin de l'image à compresser.
    :param n_components: Nombre de composantes principales à conserver.
    :param n_clusters: Nombre de clusters pour KMeans.
    :return: Image compressée.
    """
    # Chargement de l'image en couleur
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Conversion BGR en RGB
    # Aplatissement de l'image
    original_shape = image.shape
    image_flat = image.reshape((-1, 3))  # Aplatissement des pixels (RGB)
    # Application de ACP pour réduction de dimension
    pca = PCA(n_components=n_components)
    image_pca = pca.fit_transform(image_flat)
    # Application de MiniBatchKMeans pour clustering
    kmeans = MiniBatchKMeans(n_clusters=n_clusters)
```

```
    image_compressed = kmeans.fit_predict(image_pca)
    compressed_centroids = kmeans.cluster_centers_
    # Reconstruction de l'image compressée
    image_reconstructed = compressed_centroids[image_compressed]
    image_reconstructed = image_reconstructed.reshape(original_shape)
    # Retourner l'image compressée
    return image_reconstructed.astype(np.uint8)
```

6. Code Python pour l'approche hybride YCbCr+ ACP + MiniBatchKMeans :

```
from sklearn.decomposition import PCA
from sklearn.cluster import MiniBatchKMeans
import numpy as np
from PIL import Image
import os
def load_image(image_path, max_size=(1024, 1024)):
    """Charge l'image et la redimensionne si nécessaire."""
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"Le fichier image n'existe pas : {image_path}")
    try:
        img = Image.open(image_path).convert("RGB")
        img.thumbnail(max_size)
        return np.asarray(img, dtype=np.float32) / 255.0
    except Exception as e:
        raise RuntimeError(f"Erreur lors du chargement de l'image : {str(e)}")
def rgb_to_ycbcr(image):
    """Convertit l'image RGB en YCbCr."""
    transform_matrix = np.array([[0.299, 0.587, 0.114],
                                 [-0.168736, -0.331264, 0.5],
                                 [0.5, -0.418688, -0.081312]])
    offset = np.array([0, 128, 128]) / 255.0
    return np.tensordot(image, transform_matrix.T, axes=1) + offset
def ycbcr_to_rgb(image):
```

```
    """Convertit l'image YCbCr en RGB."""
    transform_matrix = np.array([[1.0, 0.0, 1.402],
                                 [1.0, -0.344136, -0.714136],
                                 [1.0, 1.772, 0.0]])
    offset = np.array([0, 128, 128]) / 255.0
    return np.clip(np.tensordot(image - offset, transform_matrix.T, axes=1), 0, 1)
def compress_channel(channel, n_colors=256, n_components=0.95):
    """
    Compression d'une composante (Y, Cb ou Cr) en utilisant ACP et MiniBatchKMeans.
    """
    pixels = channel.flatten()
    # Définir un nombre ajusté de composantes pour PCA
    n_components_adjusted = min(int(channel.size * n_components), min(channel.shape))
    # Vérifier que la taille des données est suffisante
    if channel.size < 2 or channel.shape[0] < 2 or channel.shape[1] < 2:
        return channel  # Retour sans compression si les données sont trop petites
        try:
        # ACP avec n_components ajusté
        pca = PCA(n_components=n_components_adjusted, svd_solver='auto')
        reduced_pixels = pca.fit_transform(pixels.reshape(-1, 1))
    except ValueError:
        return channel  # Retour si PCA échoue
    # Clustering avec MiniBatchKMeans
    kmeans = MiniBatchKMeans(
        n_clusters=n_colors,
        batch_size=1000,
        max_iter=300,
        random_state=42,
        n_init=1
    ).fit(reduced_pixels)
    # Reconstruction des pixels compressés
    compressed_pixels = pca.inverse_transform(
```

```
        kmeans.cluster_centers_[kmeans.predict(reduced_pixels)]
    ).reshape(channel.shape)
    return compressed_pixels
def compress_image(image, n_colors_y=256, n_colors_cbcr=128, n_components=0.95):
    """
    Compression de l'image en utilisant YCbCr et compression des canaux.
    """
    ycbcr_image = rgb_to_ycbcr(image)
    y_compressed = compress_channel(ycbcr_image[:, :, 0], n_colors=n_colors_y,
n_components=n_components)
    cb_compressed = compress_channel(ycbcr_image[:, :, 1], n_colors=n_colors_cbcr,
n_components=n_components)
    cr_compressed = compress_channel(ycbcr_image[:, :, 2], n_colors=n_colors_cbcr,
n_components=n_components)
    compressed_ycbcr = np.stack((y_compressed, cb_compressed, cr_compressed), axis=-1)
    return ycbcr_to_rgb(compressed_ycbcr)
def save_image(image, output_path):
    """Sauvegarde l'image compressée."""
    Image.fromarray((image * 255).astype(np.uint8)).save(output_path)
```

7. Les images originales

Figure 1 *: les images originales*

8. Les images compressées

Figure 2 *: les images compressées*