



Institut Polytechnique de l'Université de Nice  
Departement Mathématiques Appliquées et Modélisation (MAM)

## Analyse et traitement d'image, compression d'image

Amiel Metier  
Etan Benchiha  
Hedi Chennoufi  
Nicolas Muratore

*Supervisor:* Cédric Boulbe, Didier Auroux

Rapport présenté en vue de satisfaire aux exigences de l'Université de Nice  
pour l'obtention du diplôme de  
Master of Science en Mathématiques Appliquées et Modélisation

# Contents

1	Introduction	1
2	Objectifs	1
3	Explications théoriques	2
3.1	Fonctionnement d'une image	2
3.2	Transformée de cosinus discrète (DCT) d'une image	2
3.3	Compression	3
3.4	Décompression	4
3.5	Filtrage des hautes fréquences et débruitage	5
4	Réalisations pratiques	6
4.1	Lecture et normalisation	6
4.2	Définition de la Matrice P	6
4.3	Compression	7
4.4	Décompression et Recomposition	8
4.5	Post-Traitement et affichage des résultats	8
5	Conclusion	9

## 1 Introduction

Les images numériques occupent une place centrale dans notre quotidien, qu'il s'agisse de photographies personnelles, de contenus multimédias ou d'applications industrielles. Toutefois, elles peuvent rapidement devenir **encombrantes** en termes de mémoire et de stockage, notamment lorsque des niveaux de résolution inutiles surpassent les capacités de perception de l'œil humain.

Une gestion inefficace de ces ressources entraîne non seulement une consommation excessive d'espace, mais aussi des impacts significatifs sur les performances des systèmes de traitement.

Face à ces défis, les avancées dans le domaine de la compression d'images jouent un rôle clé. Il y a une trentaine d'années, **Joan Mitchell** a révolutionné ce domaine en contribuant au développement de la norme **JPEG** (Joint Photographic Experts Group), une méthode de compression d'images qui repose sur des outils mathématiques avancés tels que la transformée en cosinus discrète (DCT).

Cette solution que nous allons étudier ici a marqué un tournant en optimisant le stockage des images tout en préservant leur qualité visuelle essentielle.

## 2 Objectifs

Les objectifs principaux de ce projet sont les suivants :

1. **Étudier la représentation numérique des images** et leur **décomposition en fréquences**.
2. **Implémenter la compression d'images** à l'aide de la **transformée en cosinus discrète (DCT)**.
3. **Réduire la taille des fichiers** tout en préservant la qualité visuelle essentielle de l'image.

4. **Tester différentes méthodes de compression** et de **dé-bruitage**, en particulier la quantification et la troncature.
5. **Évaluer les pertes dues à la compression** en termes de qualité visuelle et de **taux de compression**.
6. **Comparer les résultats obtenus** avec des implémentations existantes et des bibliothèques DCT bien établies.
7. **Appliquer ces techniques sur diverses images**, qu'elles soient en niveaux de gris, en couleur ou bruitées.

## 3 Explications théoriques

### 3.1 Fonctionnement d'une image

Une image peut être représentée de différentes manières. Dans le cadre de notre projet, nous adopterons la représentation la plus simple, à savoir celle décomposée en trois composantes de couleur : RGB (Red-Green-Blue).

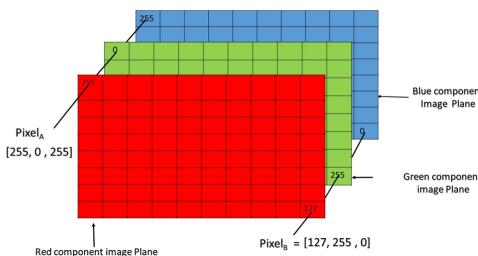


Figure 1: Enter Caption

Cette image sera modélisée sous la forme d'une matrice à trois dimensions : longueur, largeur et composantes de couleur. Chaque couleur sera traitée indépendamment, à l'aide de matrices classiques à deux dimensions. Les éléments de ces matrices représentent les intensités lumineuses des pixels pour chaque couleur. Ces intensités peuvent être exprimées soit sous forme de valeurs réelles comprises entre 0 et 1, soit sous forme d'entiers allant de 0 à 255. Il convient de noter qu'une quatrième composante peut apparaître, mais elle ne contiendra aucune information, car elle est dédiée à la gestion de la transparence.

### 3.2 Transformée de cosinus discrète (DCT) d'une image

Nous allons utiliser des transformées de Fourier pour compresser l'information contenue dans l'image. En symétrisant et périodisant l'image, nous la rendons infinie et périodique. Toute fonction périodique peut être décomposée en une combinaison de cosinus et de sinus, mais si elle est paire, seule la décomposition en cosinus est nécessaire.

Grâce à ce processus, nous pouvons décomposer une image en une combinaison de fonctions cosinus en  $x$  et  $y$ . La **transformée de Fourier discrète** (DFT) peut ainsi être réduite à une transformée de cosinus grâce à la parité de l'image.

Dans ce projet, nous limiterons à des blocs de  $8 \times 8$  de l'image. Bien sûr, nous aurions pu choisir d'autres dimensions, comme  $16 \times 16$  ou  $4 \times 4$ , ce qui aurait influé sur le taux de compression et la qualité de l'image. Avant d'appliquer la DCT, nous allons centrer les valeurs de l'image autour de 0, c'est-à-dire entre  $-127$  et  $128$ , pour plus de simplicité. Pour cela, il suffit de soustraire  $128$  à l'intensité de chaque pixel.

Pour chaque bloc  $8 \times 8$ , que nous notons  $M$  pour la matrice, nous allons appliquer la DCT pour  $0 \leq k, l \leq 7$  :

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

où  $C_0 = \frac{1}{\sqrt{2}}$  et  $C_k = 1$  si  $k \geq 0$  (idem en  $l$ ).

Cette formule est adaptée uniquement pour des blocs de  $8 \times 8$ . Pour d'autres tailles, il faudra l'adapter. Grâce à cela, nous obtenons une matrice  $D$  de la même taille que  $M$ . Ce choix particulier pour la DCT (DCT-II) permet d'obtenir un opérateur orthogonal. La transformation adjointe (transposée) correspond alors à la transformée inverse (DCT inverse).

Notre formule s'apparente à un simple changement de base orthonormée, qui peut s'écrire sous la forme :

$$D = PMP^T$$

où  $P$  est une matrice contenant la coefficient de la DCT.

Comme évoqué précédemment, on va pouvoir calculer la transposé de  $P$  très facilement car c'est aussi son inverse, bien évidemment pour trouver  $P$ , il faudra raisonner par identification et on obtient alors :

$$P = \frac{1}{2} C_k \sum_{i=0}^7 \sum_{j=0}^7 \cos\left(\frac{(2i+1)k\pi}{16}\right)$$

Notre matrice  $D$  représente la fréquence des changements d'intensité lumineuse. Les coefficients situés dans le coin supérieur gauche de cette matrice correspondent aux basses fréquences en  $x$  et en  $y$  (variations lentes), qui sont plus visibles à l'œil nu. En revanche, les coefficients situés dans le coin inférieur droit correspondent aux hautes fréquences (variations rapides), qui sont presque invisibles à l'œil nu.

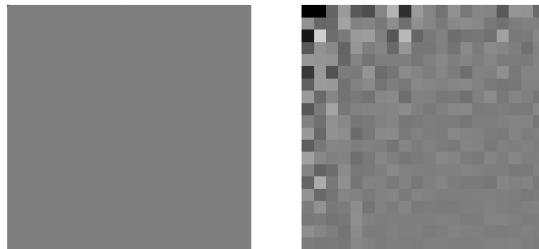


Figure 2: DCT de l'image précédente, et zoom sur les 20 premières fréquences uniquement.

### 3.3 Compression

Pour débuter la compression, nous effectuons une étape de **quantification**, qui consiste à éliminer ou réduire significativement les hautes fréquences. Ces hautes fréquences, souvent assimilées à du bruit, contiennent généralement peu d'informations perceptibles par l'œil humain. (Nous travaillons toujours sur des blocs de  $8 \times 8$ .)

La quantification s'effectue en divisant chaque élément de notre matrice  $D$  par le terme correspondant dans une matrice de quantification  $Q$ . Ensuite, tous les résultats sont arrondis à leur

valeur entière. Cette opération permet d'améliorer le taux de compression mais entraîne une perte d'information.

Dans la norme JPEG, plusieurs matrices de quantification sont disponibles et peuvent être choisies en fonction de la nature de l'image. Par exemple :

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

Cette matrice, appelée **matrice de quantification de luminance**, est utilisée pour préserver les détails et les transitions douces dans les zones lumineuses. En effet, l'œil humain est beaucoup plus sensible aux variations de luminosité qu'aux variations de couleur.

De manière similaire, une **matrice de quantification de chrominance** est utilisée pour les données de couleur. Par exemple :

$$Q_{\text{chrominance}} = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

Cette matrice influe principalement sur les couleurs. Comme l'œil humain est moins sensible aux détails chromatiques qu'aux variations de luminosité, elle contient des valeurs plus élevées, permettant une compression accrue des données de chrominance sans impact significatif sur la qualité perçue de l'image.

Après quantification, la matrice résultante est souvent très creuse (majoritairement composée de zéros). Pour une image dite standard, on peut facilement réduire par un facteur de 10 à 15 le nombre d'éléments non nuls, ce qui permet d'atteindre un taux de compression de l'ordre de 85 à 90%.

### 3.4 Décompression

Pour la décompression, il suffit d'appliquer les mêmes opérations que celles décrites précédemment, mais dans l'ordre inverse, avec quelques nuances.

Tout d'abord, l'image est redécoupée en blocs de  $8 \times 8$ . Chaque élément de la matrice décompressée est ensuite multiplié par son élément correspondant dans la matrice de quantification  $Q$ . Cela nous permet de récupérer une matrice exprimée dans la base des cosinus, notée  $\tilde{D}$ .

Ensuite, pour obtenir l'image finale, exprimée dans la base des intensités lumineuses, nous appliquons un changement de base inverse selon la formule suivante :

$$\tilde{M} = P^T \tilde{D} P$$

où  $P$  représente la matrice de transformation cosinus. Enfin, nous rassemblons tous les blocs  $8 \times 8$  dans le bon ordre pour reconstituer l'image décompressée.

À l'œil nu, l'image compressée semble identique à l'originale. Cependant, en zoomant, des différences deviennent visibles, comme illustré ci-dessous :

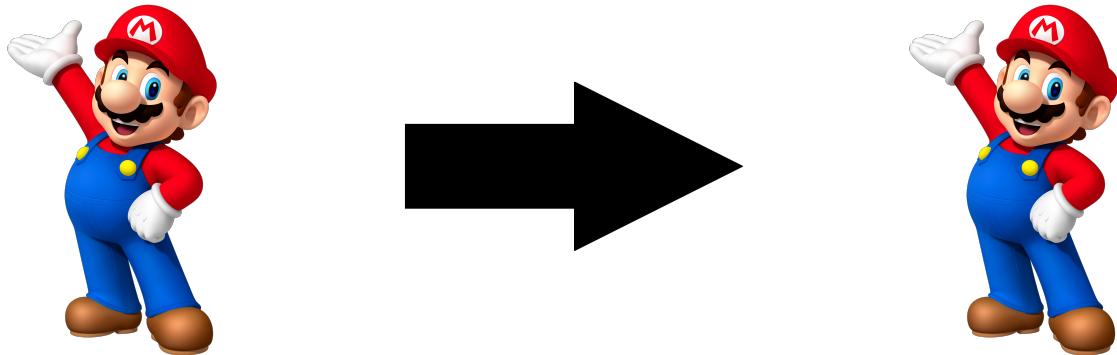


Figure 3: Image compressée (gauche) et originale (droite).

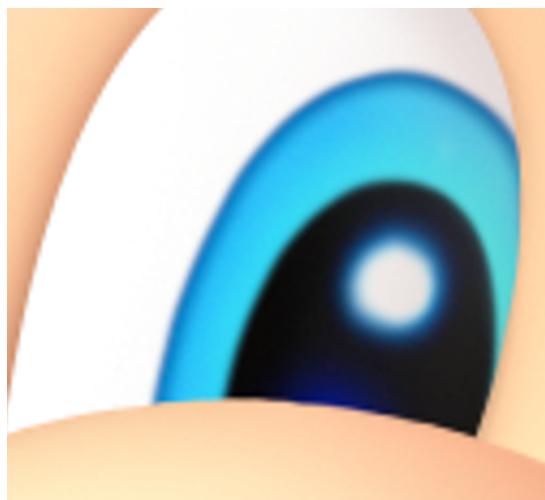


Figure 4: Image originale zoomée.



Figure 5: Image compressée zoomée.

### 3.5 Filtrage des hautes fréquences et débruitage

Une méthode simple et efficace pour compresser une image tout en réduisant le bruit consiste à tronquer l'information au-delà d'une certaine fréquence. Plutôt que d'utiliser uniquement une quantification classique, on choisit ici de supprimer directement certaines informations. Concrètement, cela revient à mettre à zéro les coefficients  $D_{lk}$  de la matrice  $D$  lorsque les indices  $l$  et  $k$  dépassent une fréquence seuil  $F$  (par exemple,  $F = 6$ ).

Cette méthode peut également être combinée avec la quantification : les hautes fréquences sont d'abord supprimées pour éliminer le bruit, puis une quantification est appliquée pour maximiser la compression. Bien que cette approche soit moins performante en termes de compression que la méthode standard JPEG (qui utilise une matrice  $Q$ ), elle est plus rapide et met en évidence que les basses fréquences contiennent l'essentiel de l'information de l'image. Cette méthode permet également de réduire efficacement le bruit, notamment lorsque celui-ci est de type bruit blanc gaussien.



Figure 6: Image bruitée.



Figure 7: Image débruitée (fréquences  $\leq 6$ ).

Illustration du débruitage par suppression des hautes fréquences ( $l, k > 6$ ).

## 4 Réalisations pratiques

Dans cette section, nous décrivons la réalisation en Python des différentes étapes de compression appliquées à l'image, en traitant chacune des 3 couches (RVB) et en calculant les données intéressantes.

### 4.1 Lecture et normalisation

Ce code charge une image nommée "trefles.png", en extrait ses dimensions, et la recadre pour que sa hauteur et sa largeur soient des multiples de 8. Cette étape est nécessaire pour diviser l'image en blocs  $8 \times 8$ , utilisés dans les algorithmes de compression comme JPEG. Enfin, un tableau vide de même taille est créé pour stocker l'image traitée ultérieurement.

```

1 image = plt.imread("trefles.png")
2 taille = np.shape(image)
3 x = taille[0]
4 y = taille[1]
5 multiple_8_x = (x // 8) * 8
6 multiple_8_y = (y // 8) * 8
7 image = image[:multiple_8_x, :multiple_8_y]
8 image_finale = np.zeros((multiple_8_x, multiple_8_y, 3))

```

### 4.2 Définition de la Matrice P

La matrice  $P$  est une matrice  $8 \times 8$  utilisée dans la transformée en cosinus discrète (DCT). Elle est calculée à l'aide des coefficients cosinus selon la formule suivante :

$$P[i, j] = \frac{1}{2} \cos\left(\frac{(2j+1) \cdot i \cdot \pi}{16}\right)$$

pour  $i, j \in \{0, 1, \dots, 7\}$ .

De plus, pour le premier indice  $i = 0$ , une normalisation supplémentaire est appliquée afin de garantir l'orthogonalité de la transformation :

$$P[0, j] = \frac{P[0, j]}{\sqrt{2}} \quad \text{pour tout } j.$$

Cette matrice est utilisée pour effectuer la DCT sur des blocs de  $8 \times 8$  pixels dans l'image, permettant ainsi la décomposition de l'image en fréquences.

```

1 P = np.zeros((8, 8))
2 for i in range(8):
3     for j in range(8):
4         P[i, j] = math.cos(((2 * j + 1) * i * math.pi) / 16) / 2
5     if i == 0:
6         P[i, j] = P[i, j] / math.sqrt(2)

```

### 4.3 Compression

#### Normalisation et préparation de chaque canal de l'image

Dans cette partie du code, chaque canal de l'image (R,V,B) est extrait et converti en type float. Ensuite, le canal est normalisé en fonction de sa valeur maximale pour qu'il ait une amplitude de [0, 255]. Puis, un décalage de 128 est appliqué à chaque pixel pour que les valeurs de l'image se situent dans l'intervalle [128,127].

```

1 if(image.ndim == 3): #Si l'image est en 3D
2     for n in range(3): #Pour chacune des 3 couches
3         canal = image[:, :, n].astype(float)
4         canal = (canal / np.max(canal)) * 255
5         canal = canal - 128 * np.ones((multiple_8_x, multiple_8_y)) #
[-128, 127]

```

#### Application de la transformée en cosinus discrète (DCT) par blocs 8x8

On applique la transformée en cosinus discrète (DCT). Le calcul utilise la matrice  $P$  qui est préalablement définie. La transformation est effectuée sur chaque sous-matrice de  $8 \times 8$  pixels, en multipliant le bloc par  $P$ , puis en le multipliant par la **transposée de  $P$** . Cela correspond à la formule pour la DCT. Après cette opération, chaque bloc est divisé par la matrice  $Q$  (matrice de quantification) et converti en entier. Cela permet de réduire la quantité de données nécessaires pour coder l'image.

```

1 for k in range(0, multiple_8_x, 8):
2     for w in range(0, multiple_8_y, 8):
3         canal[k:k+8, w:w+8] = np.matmul(P, np.matmul(canal[k:k+8, w:w+8], np
        .transpose(P)))
4         canal[k:k+8, w:w+8] = np.divide(canal[k:k+8, w:w+8], Q).astype(int)

```

#### Élimination du bruit (Dé-bruitage)

Afin d'améliorer la qualité de l'image tout en réduisant la taille du fichier sans perdre trop de détails perceptibles. Ici, pour chaque bloc  $8 \times 8$ , si la somme des indices  $z$  et  $t$  est supérieure à un certain seuil, les coefficients correspondants dans le bloc sont mis à zéro. Cela permet de ne conserver que les informations les plus significatives du bloc et de supprimer les détails moins importants.

```

1 for k in range(0, multiple_8_x, 8): # Parcourir par ligne, par blocs de 8
2     for w in range(0, multiple_8_y, 8): # De même pour les colonnes
3         for z in range(0, 8): # Parcourir les lignes relatives au bloc
4             actuel
5                 for t in range(0, 8): # De même pour les colonnes
6                     if z + t >= 16 - SEUIL:
7                         canal[k + z, w + t] = 0

```

## 4.4 Décompression et Recomposition

Les coefficients quantifiés sont décompressés en multipliant par  $Q$  et en appliquant l'inverse de la DCT. L'image est reconstruite bloc par bloc, puis les valeurs sont ramenées dans la plage [0, 255].

```

1 #Decompression et recomposition
2 for k in range(0, multiple_8_x, 8):
3     for w in range(0, multiple_8_y, 8):
4         canal[k:k+8, w:w+8] = canal[k:k+8, w:w+8] * Q
5         canal[k:k+8, w:w+8] = np.matmul(np.transpose(P), np.matmul(canal[k:k+8, w:w+8], P))
6
7 # Post processing
8 canal = canal + 128
9 canal = np.clip(canal, 0, 255) / 255
10 image_finale[:, :, n] = canal

```

## 4.5 Post-Traitemet et affichage des résultats

### Taux de compression et erreur

Le premier calcul est lié au taux de compression, où la proportion de coefficients non nuls est utilisée pour évaluer le taux de compression pour chaque canal. Le second calcul évalue l'erreur de compression pour chaque canal en mesurant la différence entre l'image originale et l'image compressée.

```

1 #Taux de compression par couche
2     taux1 = (100 - (np.count_nonzero(canal)/(x * y)) * 100)
3     taux.append(taux1)
4 #...
5
6 # Erreur par couche
7     error1 = np.linalg.norm(image[:, :, n] - canal)
8     errors.append(error1 / np.linalg.norm(image[:, :, n]))

```

Les résultats finaux, y compris le temps de calcul (CPU), sont affichés.

```

1 em = 0
2 t = 0
3 for to in taux:
4     t += to
5 for error in errors:
6     em += error

```

```

7 print("Taux de compression : ",t/3,"%")
8 print("Erreur moyenne : ", em / 3)
9 print(f"Temps CPU utilis\'e : {end - start} secondes")
10 plt.imsave('image_decompresssee.png', image_finale)
11 plt.imshow(image_finale)

```

Un affichage graphique peut être fait avec la bibliothèque **tkinter** et serait intéressant pour comparer directement l'image avant et après compression.

## 5 Conclusion

Ce projet a permis de développer et d'implémenter une méthode de compression d'images basée sur la Transformée en Cosinus Discrète (DCT). L'objectif principal était de réduire la taille des fichiers tout en minimisant les pertes d'information perceptibles. Le procédé a été appliqué à des blocs de  $8 \times 8$  pixels, permettant une représentation des données en termes de fréquences spatiales. Les coefficients des basses fréquences, contenant la majeure partie de l'information visuelle, ont été conservés, tandis que les hautes fréquences ont été réduites ou supprimées à l'aide d'une matrice de quantification.

Les résultats montrent que la compression réduit la taille des fichiers d'environ 85% à 90%, selon le niveau de quantification choisi. La qualité visuelle des images reste acceptable pour un usage standard, bien que des artefacts apparaissent dans les zones à fortes variations de luminosité. L'évaluation des erreurs a révélé que les différences entre l'image originale et l'image compressée sont localisées principalement dans ces zones, confirmant l'efficacité de la méthode pour préserver les informations globales.

Enfin, des expériences de filtrage des hautes fréquences ont montré que cette approche pouvait également être utilisée pour réduire le bruit dans les images, notamment en présence de bruit blanc gaussien. Ces résultats confirment la pertinence des techniques développées et offrent une base pour des améliorations futures, telles que l'adaptation dynamique des matrices de quantification ou l'intégration de méthodes plus avancées de compression.