UE 455 TP 2 Théorie de l'Information - Codage de Source

SHI Wenli
XU Kaiyuan

29 Avril 2024

Table des matières

1 Objectif

Un codeur d'images tel que JPEG ou JPEG 2000 est composé principalement de trois blocs. Le bloc de transformation, le quantificateur et le codeur entropique. Le bloc de transformation permet de transformer l'image à coder de manière à regrouper en sous-ensembles les informations réellement utiles à la bonne reconstitution d'une image. Dans la norme JPEG, la transformation exploite une transformée en cosinus discrete. Dans la norme JPEG, elle se fait suivant une décomposition pyramidale à l'aide d'une transformée en ondelettes. Le bloc de quantification permet de forcer à zéro un certain nombre de coefficients transformés, ce qui autorisera, grâce au dernier bloc, le codage entropique, d'atteindre de forts taux de compression, tout en conservant une qualité acceptable.

Ce TP permettra d'analyser l'effet des trois blocs formant un codeur d'images et de voir suivant les choix réalisés, quelles performances peuvent être atteintes, tant en termes de qualité d'image que de débit binaire. Vous travaillerez sur des images en niveaux de gris.

2 Manipulations autours de JPEG

Commencez par importer quelques fonctions utiles des bibliothèques `matplotlib`, `numpy`, `scipy`, et du fichier `quant` qui doit être placé dans votre répertoire de travail.

2.1 Lecture et affichage d'une image

Pour des raisons de commodité, les images à compresser sont stockées au format `.pgm`, qui est un format binaire, précédé d'un en-tête en clair indiquant la taille de l'image. Ouvrez par exemple le fichier `lenna.256.pgm` avec un éditeur de texte pour visualiser l'en-tête.

Manipulation: 1 Pour charger et visualiser une image, utiliser les commandes `imread` et `imshow`.

En utilisant les codes suivants, on peut afficher cette image comme la figure 2.1.1.

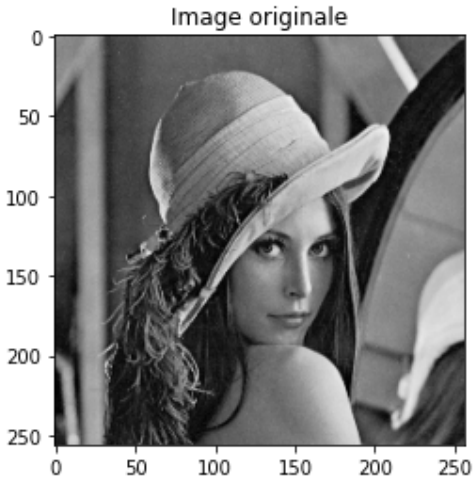

FIGURE 2.1.1 – Lenna 256.

FIGURE 2.1.2 – Lenna 256 modifiee.

Listing 1 – Affichage d'image

```
1 # Manipulation 1
2 I = plt.imread('lenna.256.pgm')
3 plt.imshow(I, cmap='gray')
4 plt.title("Image␣originale")
5 plt.show()
```

Manipulation: 2 Quelle est la taille de l'image chargée (utilisez la proprieté `shape`) ?

En utilisant la proprieté `shape`, on peut savoir que le taille de l'image chargée est de 256×256.

Listing 2 – Output de Console - Taille d'image

```
L'image est de taille: (256, 256)
```

Manipulation: 3 En général, les codeurs d'image travaillent sur une image dont l'intensité moyenne des pixels est nulle. Pour cela, il est nécessaire de calculer la valeur moyenne des pixels de l'image et de l'ôter à chaque pixel de l'image.

En utilisant les codes suivants, on peut modifier une image dont l'intensité moyenne des pixels est nulle. En comparant l'image avec l'image affichée dans la manipulation 1, on peut constater qu'il n'y a pas de différence et cette étape est seulement pour les codeurs suivants.

Listing 3 – Affichage d'image

```
# Manipulation 3
moyenne = np.floor(I.mean())
I = I.astype(np.float32)- moyenne

plt.imshow(I, cmap='gray')
plt.title("Image modifiee")
plt.show()
```

2.2 Transformée en cosinus discrête

La norme de compression JPEG avec perte définit le bloc de 8×8 pixels comme l'unité de traitement élémentaire des images à compresser. A chaque bloc $\{B_{x,y}\}$ de l'image est appliquée une transformation en cosinus discrête bi-dimensionnelle permettant d'obtenir un bloc transformé $\{B_{u,v}\}$ de la manière suivante

$$\mathbf{B}_{u,v} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} \mathbf{b}_{x,y} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

avec

$$C_u = \begin{cases} 1/\sqrt{2} & \texttt{si } u = 0 \\ 1 & \texttt{si } u \neq 0 \end{cases}$$

Cette transformation est inversible

$$\mathbf{b}_{x,y} = \frac{1}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} C_u C_v \mathbf{B}_{u,v} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

Manipulation: 4 Rappelez la définition d'une transformée unitaire et séparable.

Pour une transformée unitaire, elle est une transformée unitaire si elle est la transposée de sa transformée inverse. Pour une matrice de transformée $\mathbf{U}$, elle satisfait :

$$\mathbf{U}^T \mathbf{U} = \mathbf{I} \texttt{ ou } \mathbf{U}^T = \mathbf{U}^{-1}$$

où $\mathbf{I}$ est une matrice d'identité.

Pour une transformée séparable, elle est séparable si elle peut être décomposée en transformées indépendantes pour les lignes et les colonnes, et ces transformées indépendantes peuvent agir sur différentes dimensions des données.

Manipulation: 5 Le calcul de la transformée en cosinus discrête bi-dimensionnelle se fait à l'aide de la commande `dctn` de la bibliothèque `scipy.fftpack` qui permet de calculer des transformée en cosinus discrêtes sur des blocs d'image dont la taille est ajustable. Ainsi le code permet de calculer la transformée de blocs de tailles 8×8. Le résultat est stocké dans une matrice dont chaque colonne contient l'ensemble des coefficients transformés d'un bloc.

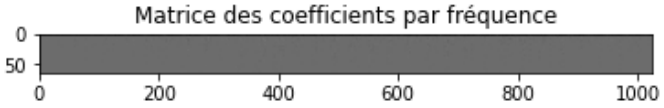

FIGURE 2.2.1 – Matrice des coefficients par fréquence.

Ce code définit la taille du bloc DCT comme 8×8, puis parcourt chaque bloc 8×8 de l'image via une double boucle et utilise la fonction `dctn` pour DCT chaque bloc. Et on stocke les coefficients de chaque bloc dans une matrice `It`. La taille de la matrice `It` est de 64×1024. Cela indique qu'il y a 1024 blocs DCT dans cette image. Le résultat est la figure 2.2.1.

Listing 4 – Affichage d'image

```
# Manipulation 5
B = 8
nb_blocks_rows = int(rows/B)
nb_blocks_cols = int(cols/B)
It = np.zeros((B*B, nb_blocks_rows * nb_blocks_cols), np.float32)
for r in range(nb_blocks_rows):
    for c in range(nb_blocks_cols):
        currentblock = dctn(I[r * B:(r+1) * B, c * B:(c+1) * B], norm='ortho')
        It[:, r * nb_blocks_cols + c] = currentblock.flatten()

plt.imshow(It, cmap='gray')
plt.title("Matrice des coefficients par frequence")
plt.show()
```

Manipulation: 6 Extrayez la première ligne de la matrice `It`. Calculez un histogramme des valeurs des coefficients de cette ligne et affichez-le. Affichez également la première ligne de `It` après l'avoir remise dans des dimensions compatibles avec l'image initiale.
Que constatez-vous ? Quelles informations contient la première ligne de `It`. Quelle distribution de probabilité vous paraît le mieux décrire les valeurs des coefficients de la première ligne ?

En utilisant les codes suivants, on peut calculer et afficher un histogramme de la première ligne. Le résultat est la figure 2.2.2.

On peut constater que des valeurs inférieures se produisent plus fréquemment dans l'histogramme et que l'histogramme présente une variance plus grande. De plus, l'image affichée contient déjà les grandes lignes de l'image originale et le contenu flou de l'image est déjà visible.

La première ligne de la matrice `It` contient les coefficients du coin supérieur gauche des 1024 blocs DCT. C'est-à-dire qu'elle contient les informations de fréquence la plus basse parmi tous les blocs DCT. Ce sont également des informations basse fréquence de l'image originale.

La distribution normale peut mieux décrire la distribution de tous les coefficients de la première ligne de la matrice `It`.

Listing 5 – Histogramme de la première ligne

```
# Manipulation 6
idx = 0
coefs = It[idx, :]

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle("Coefficients a la position {}".format(idx))

ax1.hist(coefs)
ax1.set_xlabel('Histogramme')
ax2.imshow(coefs.reshape((nb_blocks_rows, nb_blocks_cols)), cmap='gray')
plt.show()
```

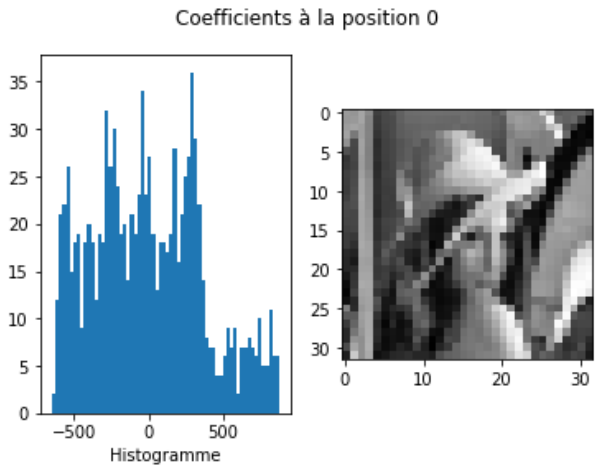

FIGURE 2.2.2 – Histogramme de la première ligne.

Manipulation: 7 Reprenez la question avec la seconde ligne, avec la neuvième ligne et avec deux autres lignes de votre choix.

On a choisit les lignes 2, 9, 21 et 41 pour reprendre la question précédente. Les résultats sont les figures suivantes.

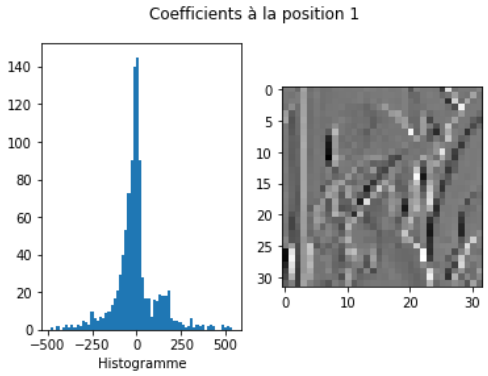

FIGURE 2.2.3 – Histogramme de la 2eme ligne.

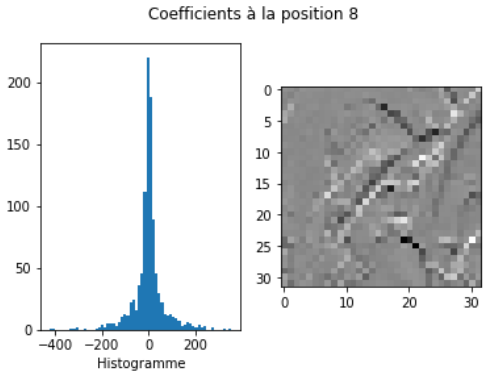

FIGURE 2.2.4 – Histogramme de la 9eme ligne.

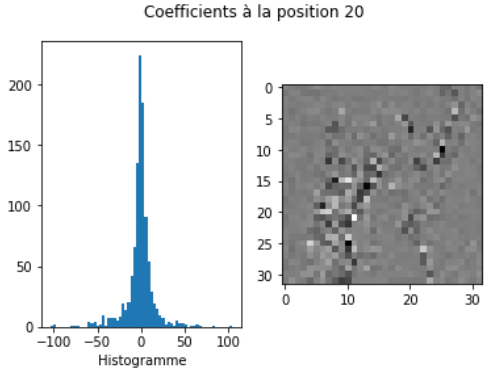

FIGURE 2.2.5 – Histogramme de la 21eme ligne.

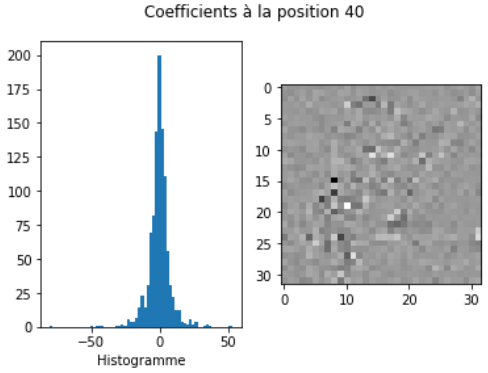

FIGURE 2.2.6 – Histogramme de la 41eme ligne.

À partir des images ci-dessus, on peut observer que l'histogramme des coefficients présente toujours à peu près une distribution normale, mais la variance est bien inférieure à celle de la première ligne.

En observant les image présentées à droite, on peut constater qu'à mesure que le nombre de lignes augmente, le contenu de l'image devient de plus en plus flou, c'est-à-dire qu'il y a de moins en moins d'informations basse fréquence. En revanche, ces images contiennent davantage d'informations à haute fréquence de l'image originale, c'est-à-dire les informations détaillées de l'image originale.

Manipulation: 8 Estimez et représentez en échelle logarithmique la variance des coefficients de chaque ligne de `It`.
Conclusions ?

En utilisant les codes suivantes, on peut estimer et représenter en échelle logarithmique la variance des coefficients. Le résultat est la figure 2.2.7. En observant la courbe obtenue, on peut conclure les points suivants.

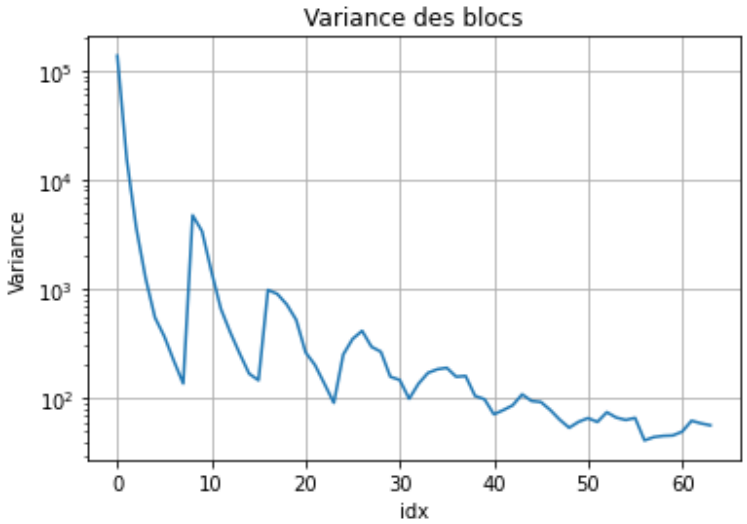

FIGURE 2.2.7 – Echelle logarithmique.

Conclusions :

À partir du tracé de variance des coefficients DCT, on peut constater que les coefficients basse fréquence ont une variance plus élevée, ce qui signifie que l'énergie de l'image est principalement concentrée sur ces coefficients. À mesure que l'indice de ligne augmente, la variance diminue généralement, ce qui indique que les coefficients haute fréquence présentent des changements plus faibles et une énergie plus faible.

La DCT convertit l'image du domaine spatial en domaine fréquentiel, où la composante basse fréquence correspond aux informations de luminosité de l'image et la composante haute fréquence correspond aux informations détaillées.

Listing 6 – Echelle logarithmique

```
1 # Manipulation 8
2 var = np.var(It, axis=1)
3 plt.semilogy(var)
4 plt.grid()
5 plt.title("Variance␣des␣blocs")
6 plt.xlabel('idx')
7 plt.ylabel('Variance')
8 plt.show()
```

Manipulation: 9 On procède ensuite à un balayage zig-zag des coefficients transformés dans chaque bloc, voir la figure 1. Ceci permet d'obtenir une matrice `Itz`, analogue à `It`, mais pour laquelle les colonnes sont rangées différemment. On peut alors recalculer la variance des coefficients de chaque ligne de `Itz`.

Affichez `var_zigzag` sur la même figure que `var`. Quel est l'intéret du balayage zig-zag ?

En utilisant les codes suivantes, on peut réaliser un balayage zig-zag des coefficients transformés dans chaque bloc et le résultat est la figure 2.2.9. En comparant les images, on peut constater que la différence entre les courbes de variance des deux n'est pas très grande.

Le principal avantage du balayage Zig-zag est qu'il peut réorganiser les coefficients en fonction de la distribution d'énergie, de sorte que des coefficients nuls consécutifs puissent être placés ensemble. Cela permet aux données d'être compressées plus efficacement lors des étapes de quantification et de codage entropique.

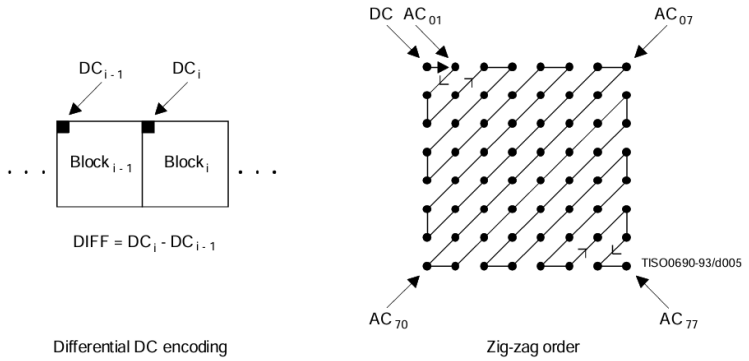

FIGURE 2.2.8 – Balayage zig-zag des coefficients.

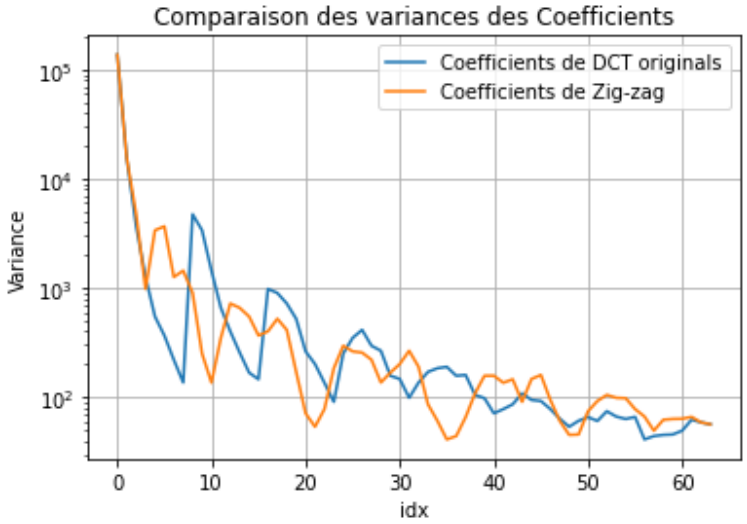

FIGURE 2.2.9 – `var_zigzag` et `var`.

Listing 7 – Balayage Zig-zag

```
# Manipulation 9
Itz = It[Zig, :]
var_zigzag = np.var(Itz, axis=1)

plt.semilogy(var, label='Coefficients de DCT originals')
plt.semilogy(var_zigzag, label='Coefficients de Zig-zag')
plt.grid()
plt.title('Comparaison des variances des Coefficients')
plt.xlabel('idx')
plt.ylabel('Variance')
plt.legend()
plt.show()
```

2.3 Quantification

La quantification des coefficients transformés est faite dans la norme JPEG à l'aide d'une quantification à pas fixe, mais dont la taille du pas de quantification est adaptée à l'indice du coefficient. Ainsi, les coefficients importants sont quantifiés avec un pas faible, les coefficients moins importants sont quantifiés avec un pas plus important. Le niveau de quantification est réglé par un paramètre multiplicatif γ appartenant généralement à l'intervalle $[0.1, 10]$.

Manipulation: 10 Pour réaliser une quantification avec un facteur $\gamma = 1$, on procède de la manière suivante.

En utilisant les codes suivantes, on peut réaliser une quantification avec un facteur $\gamma = 1$. On affiche aussi les 6 premières lignes et 6 premières colonnes de la matrice `It_quant` pour vérifier le résultat.

Listing 8 – Output de Console - Quantification

```
[[ 17.   15.   21.   11. -16. -12.]
 [  0.    1.   -5.   16.   -3.   -0.]
 [ -0.    0.   -1.   -3.    0.   -0.]
 [ -0.   -0.   -0.    0.    0.    0.]
 [  0.   -0.    0.   -0.    0.    0.]
 [ -0.   -0.   -0.    0.    0.    0.]]
```

Listing 9 – Quantification

```
# Manipulation 10
gamma = 1
q = gamma * Quant.flatten()
It_quant = np.fix(It.transpose()/q).transpose()
```

2.4 Codage entropique

Le codage entropique est relativement complexe dans la norme JPEG. Il combine un codage différentiel des coefficients basse fréquence et un codage par Run-Length des autres coefficients. L'entropie des coefficients après quantification permet d'avoir une première idée du débit d'information par bit qu'il sera nécessaire d'employer pour reconstituer l'image.

Manipulation: 11 Cette entropie peut être calculée de la manière suivante en utilisant la fonction `entropy` que vous avez programmée en début de cours.

En utilisant la fonction `entropy`, on peut calculer la valeur de l'entropie. La valeur de l'entropie de la matrice `It_quant` est de 0.976 bits/symbole.

Listing 10 – Output de Console - Entropie de It_quant

```
Entropie : 0.976 bits/symbole
```

Listing 11 – Entropie

```
# Manipulation 11
H = entropy.entropy(It_quant.flatten().tolist())
```

Manipulation: 12 Il peut être utile de réaliser un codage entropique séparé de chaque ligne de la matrice `It_quant`. En effet, les statistiques des différentes lignes de cette matrice sont très dissemblables. Proposer une méthode permettant de calculer l'entropie moyenne en considérant chaque ligne indépendemment. Comparez le résultat obtenu avec celui de la question précédente.

On calcule la valeur d'entropie de chaque ligne en parcourant chaque ligne de la matrice `It_quant`, puis on calcule la moyenne des valeurs d'entropie de toutes les lignes pour obtenir le résultat final. La valeur de l'entropie moyenne de la matrice `It_quant` est de 0.682 bits/symbole.

En observant la valeur de sortie, on peut constater que la valeur d'entropie calculée par cette méthode est plus petite.

Listing 12 – Output de Console - Entropie de It_quant

```
Entropie : 0.976 bits/symbole
Entropie moyenne : 0.682 bits/symbole
```

Listing 13 – Entropie Moyenne

```
# Manipulation 12
entropies = [entropy.entropy(It_quant[i, :].tolist()) for i in range(It_quant.shape
    [0])]

H_moyenne = np.mean(entropies)
```

Manipulation: 13 Enfin, un codage différentiel des éléments quantifiés de la première ligne de `It_quant` peut réduire la quantité d'information à coder. Il peut se faire très simplement par le code.
Recalculez l'entropie obtenue après codage différentiel de la première ligne. Commentaires ?

En utilisant les codes suivantes, on peut calculer l'entropie obtenue après codage différentiel de la première ligne. La valeur de l'entropie après codage différentiel est de 0.958 bits/symbole. Pour comparer les valeurs, on a également affiché la valeur de l'entropie de la première ligne sans codage. Le valeur de l'entropie sans codage est de 6.296 bits/symbole.

Par comparaison, on peut constater qu'après avoir utilisé le codage différentiel, la valeur d'entropie de la première ligne a été considérablement réduite par rapport à avant. Cela illustre également l'efficacité du codage différentiel dans ce problème.

L'effet du codage différentiel dépend principalement de la corrélation entre les coefficients. S'il existe des lignes fortement corrélées dans les données, ce codage peut réduire efficacement l'entropie, réduisant ainsi la quantité d'informations de codage requises pour une compression plus efficace.

Listing 14 – Output de Console - Entropie apres codage

```
Entropie de 1 : 6.296 bits/symbole
Entropie apres codage: 0.958 bits/symbole
```

Listing 15 – Entropie apres codage différentiel

```
# Manipulation 13
It_pred = It_quant.copy()
It_pred[0, 1:] = It_quant[0, 1:] - It_quant[0, 0:-1]
```

2.5 Reconstitution de l'image quantifiée

Pour reconstituer l'image après quantification, il faut d'abord effectuer un codage différentiel inverse, si celui-ci a eu lieu le code.

Ensuite, une quantification inverse est indispensable.

L'opération suivante consiste à réaliser la transformation inverse et à remettre tous les blocs dans le bon ordre. Cela se fait avec la commande `idctn`.

Pour évaluer la qualité de l'image reconstruite, on peut calculer le rapport signal à bruit crête en dB(PSNR)

$$PSNR_{(dB)} = 10 \log_{10} \frac{255 \times 255}{\sigma_D^2}$$

où σ_D^2 est la variance de la différence entre l'image initiale et l'image reconstruite. Pour calculer le rapport signal à bruit, crête vous pouvez utiliser le code.

Manipulation: 14 Etudiez l'influence de γ sur la qualité de l'image reconstruite ainsi que sur le débit binaire. Pour cela, tracez l'évolution du PSNR en (dB) en fonction du débit, pour des PSNR allant de 20 dB à 40 dB.

En changeant la valeur de γ, on peut obtenir des images reconstruites avec différentes valeurs PSNR et également obtenir des débits différents. On peut tracer le PSNR en fonction des débits. Le résultat est la figure 2.5.1.

Dans le même temps, on a également tracé les changements de PSNR et de débits en fonction des valeurs de γ. Les résultats sont les figures 2.5.2 et 2.5.3.

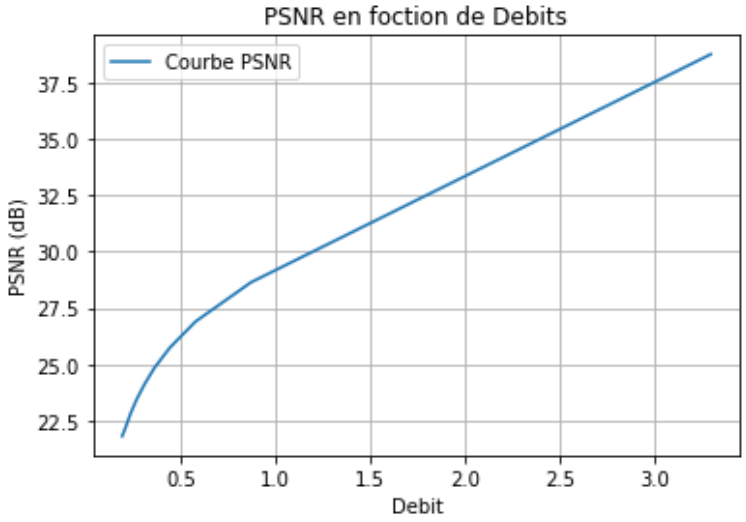

FIGURE 2.5.1 – PSNR en foction de Debits.

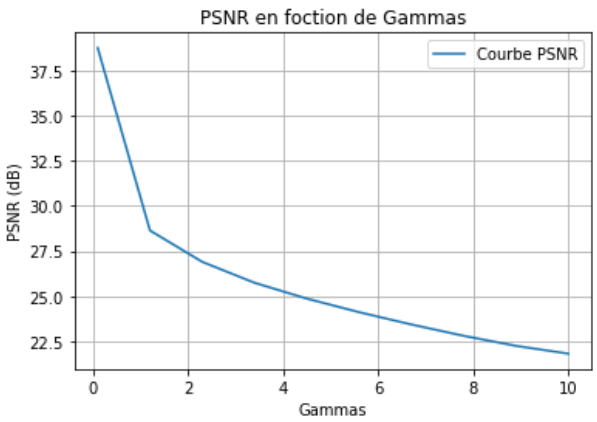

FIGURE 2.5.2 – PSNR en foction de Gammas.

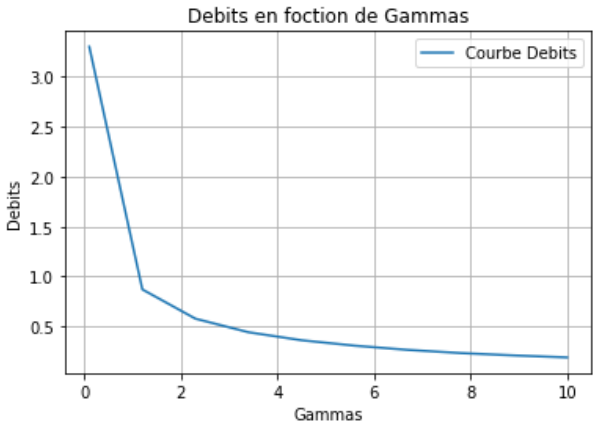

FIGURE 2.5.3 – Debits en foction de Gammas.

Listing 16 – Entropie apres codage différentiel

```
# Manipulation 14, 15
gammas = np.linspace(0.1, 10, 10)

psnrs = []
debits = []

for gamma in gammas:
    q = gamma * Quant.flatten()
    It_quant2 = np.fix(It.transpose()/q).transpose()

    debit = entropy.entropy(It_quant2.flatten().tolist())
    debits.append(debit)

    It_pred2 = It_quant2.copy()
    It_pred2[0, 1:] = It_quant2[0, 1:] - It_quant2[0, 0:-1]

    It_rec2 = It_pred2.copy()
    for i in range(1, nb_blocks_rows * nb_blocks_cols):
        It_rec2[0, i] = It_rec2[0, i] + It_rec2[0, i-1]

    It_rec2 = (It_rec2.transpose()*q).transpose()

    I_hat2 = np.zeros_like(I)
    for r in range(nb_blocks_rows):
        for c in range(nb_blocks_cols):
```

```
            I_hat2[r * B:(r+1) * B, c * B:(c+1) * B] = \
            idctn(It_rec2[:, r*nb_blocks_cols+c].reshape(B, B),norm='ortho')

    sigma2 = np.var((I - I_hat2).flatten())
    psnr = 10 * np.log10(255 * 255 / sigma2)
    psnrs.append(psnr)
```

Manipulation: 15 A partir de quelle valeur de PSNR est-il difficile de distinguer la version compressée et la version originale de l'image.

En sélectionnant différentes valeurs γ, on peut obtenir des images reconstruites sous différents PSNR. Sur la figure, on montre l'image originale, l'image reconstruite avec $PSNR = 38.77$ dB et l'image reconstruite avec $PSNR = 28.64$ dB. En observant ces images, nous pouvons constater que pour $PSNR = 38.77$ dB, la différence entre l'image originale et l'image reconstruite ne peut pas être distinguée à ce moment. Pour $PSNR = 28.64$ dB, on peut également constater quelques différences.

Par conséquent, lorsque la valeur du PSNR est supérieure à 28.64 dB, on ne peut pas distinguer la version compressée et la version originale de l'image.

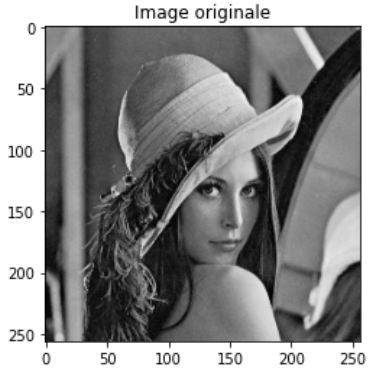

FIGURE 2.5.4 – Lenna 256.

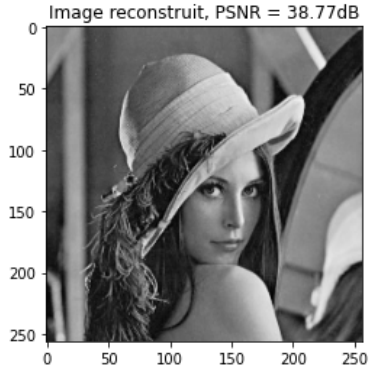

FIGURE 2.5.5 – Lenna $PSNR = 38.77$ dB.

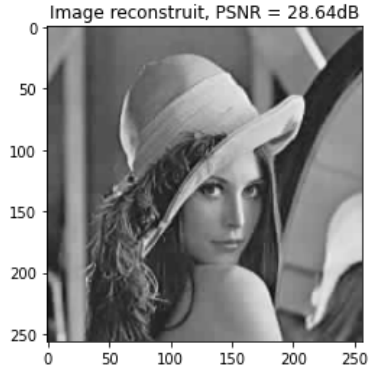

FIGURE 2.5.6 – Lenna $PSNR = 28.64$ dB.

3 Manipulations autour de JPEG 2000

Commencez par importer quelques fonctions utiles des bibliothèques `matplotlib`, `numpy`, `pywt` (c'est `PyWavelets`), et `copy`.

3.1 Lecture et affichage d'une image

Utilisez la même procédure que pour la partie JPEG.

3.2 Décomposition en sous-bandes

La principale différence entre JPEG et JPEG 2000 repose sur la transformée utilisée. JPEG 2000 repose sur une transformée en ondelettes dyadiques. L'image à compresser est filtrée à l'aide d'un filtre passe-bas et d'un filtre passe-haut, verticalement et horizontalement. Les quatre versions filtrées des images obtenues sont alors sous-échantillonnées, voir la figure 2. Ainsi, une image de 512×768 pixels après une étape de décomposition est transformée en 4 images de 256×384.

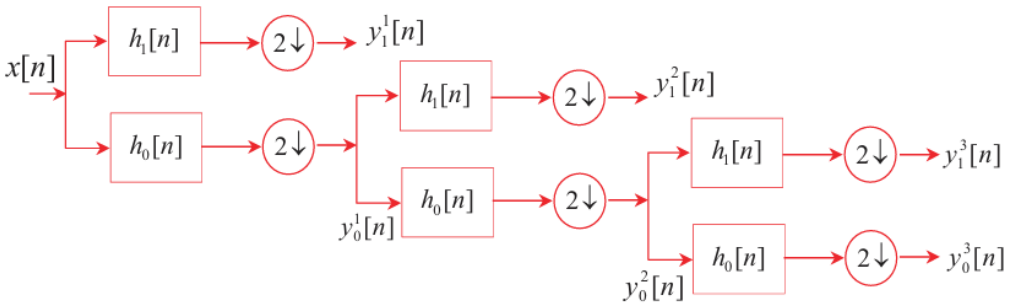

FIGURE 3.2.1 – Banc de filtres pour la décomposition en ondelettes.

La fonction pour réaliser une décomposition en sous-bandes avec un seul niveau de décomposition est `dwt2`.

Listing 17 – dwt

```
1 coeffs2 = pywt.dwt2(I, 'db4')
```

`I` correspond à l'image d'entrée (en niveaux de gris), le second argument au type d'ondelettes utilisées (par exemple `'haar'`, `'db1'`, `'db4'`...).

Manipulation: 1 Regardez la documentation de PyWavelets afin de voir quelles sont les autres ondelettes possibles.

En utilisant le code `pywt.families()`, on peut afficher tous les types des ondelettes possibles.

Listing 18 – Output de Console - Ondelettes

```
1 >> pywt.families()
2 >> ['haar', 'db', 'sym', 'coif', 'bior', 'rbio', 'dmey', 'gaus', 'mexh', 'morl', '
      cgau', 'shan', 'fbsp', 'cmor']
3
4 >> pywt.families(short=False)
5 >> ['Haar', 'Daubechies', 'Symlets', 'Coiflets', 'Biorthogonal', 'Reverse
      biorthogonal', 'Discrete Meyer (FIR Approximation)', 'Gaussian', 'Mexican hat
      wavelet', 'Morlet wavelet', 'Complex Gaussian wavelets', 'Shannon wavelets', '
      Frequency B-Spline wavelets', 'Complex Morlet wavelets']
```

Manipulation: 2 Quelle est la structure de `coeffs2` ?

`coeffs2` est le résultat obtenu après application de la décomposition en ondelettes à l'image. La structure de `coeffs2` est un tuple de quatre éléments, contenant une composante d'approximation (LL) et trois composantes de détail (LH, HL, HH).

Manipulation: 3 Affichez les différentes composantes de `coeffs2`.
Commentez le résultat obtenu. Pourquoi la composante d'approximation s'appelle-t-elle ainsi ?

On a affiché les différentes composantes de `coeffs2`.

Chaque composant est un tableau bidimensionnel représentant une sous-bande de l'image. Pour la sous-bande LL, il s'agit d'un composant approximatif qui contient des informations basse fréquence de l'image.

La sous-bande LH est le composant de détail horizontal, qui contient les informations horizontales haute fréquence de l'image.

La sous-bande HL est le composant de détail vertical, qui contient les informations verticales haute fréquence de l'image.

La sous-bande HH est le composant de détail diagonal, qui contient les informations diagonales haute fréquence de l'image.

Grâce à l'analyse ci-dessus, on peut également savoir pourquoi ces noms sont appelés.

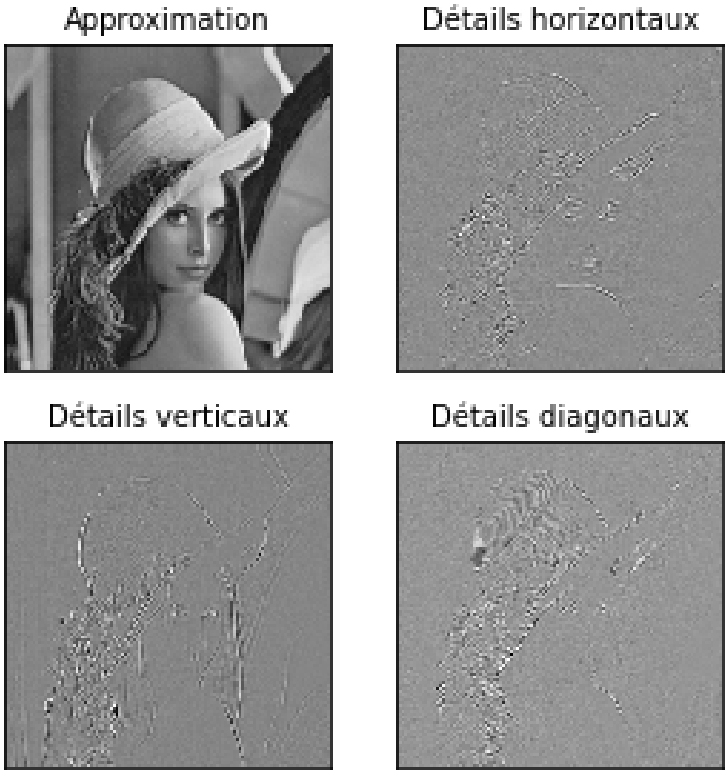

FIGURE 3.2.2 – Composantes de `coeffs2`.

Listing 19 – Affichage des composantes de `coeffs2`

```
1  # Manipulation 3
2  LL, (LH, HL, HH) = coeffs2
3
4  fig = plt.figure(figsize=(4, 4))
5  titles = ['Approximation', 'Details␣horizontaux', 'Details␣verticaux', 'Details␣
       diagonaux']
6
7  for i, a in enumerate([LL, LH, HL, HH]):
8      ax = fig.add_subplot(2, 2, i + 1)
9      ax.imshow(a, interpolation="nearest", cmap='gray')
10     ax.set_title(titles[i], fontsize=10)
11     ax.set_xticks([])
12     ax.set_yticks([])
13
14 fig.tight_layout()
15 plt.show()
```

Pour réaliser une décomposition en ondelettes avec plusieurs niveaux de décomposition, on peut appliquer la décomposition en ondelettes précédante sur la composante d'approximation. Cela se fait à partir de l'image initiale à l'aide de la fonction `wavedecn`.

Listing 20 – wavedecn

```
niveaux = 2
I_dwt = pywt.wavedec2(I, 'db4', level=niveaux)
```

Manipulation: 4 Afin de visualiser correctement les différentes composantes, il est utile de les normaliser. Cela se fait de la manière suivante. La fonction `coeffs_to_array` permet de rassembler les composantes normalisées en un tableau unique.

On a affiché la figure correctement les différentes composantes. Il s'agit d'une décomposition en ondelettes à deux niveaux qui décompose davantage l'image.

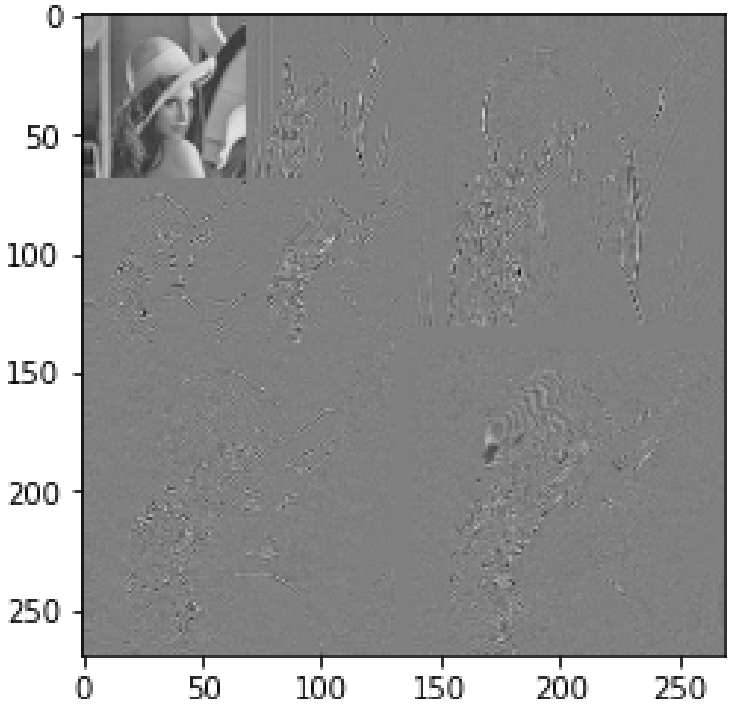

FIGURE 3.2.3 – Composantes (niveaux = 2).

Listing 21 – Manipulation 4

```
# Manipulation 4
I_dwt_norm = copy.deepcopy(I_dwt)

# Normalisation des coefficients pour ameliorer la lisibilite
# Composante d'approximation
I_dwt_norm[0] /= np.abs(I_dwt_norm[0]).max()

# Composantes de detail
for detail_level in range(niveaux):
    I_dwt_norm[detail_level + 1] = [d/np.abs(d).max() for d in I_dwt_norm[
        detail_level + 1]]

# Affichage des coefficients normalises
arr, slices = pywt.coeffs_to_array(I_dwt_norm)
plt.imshow(arr, cmap=plt.cm.gray)
plt.show()
```

Manipulation: 5 Que se passe-t-il lorsqu'on change le nombre de niveaux de décomposition ?

On a mis la valeur de niveau = 3 et on peut afficher la figure. On peut voir qu'à mesure que le niveau augmente, l'image sera décomposée en davantage de composants et les détails seront décomposés plus en détail.

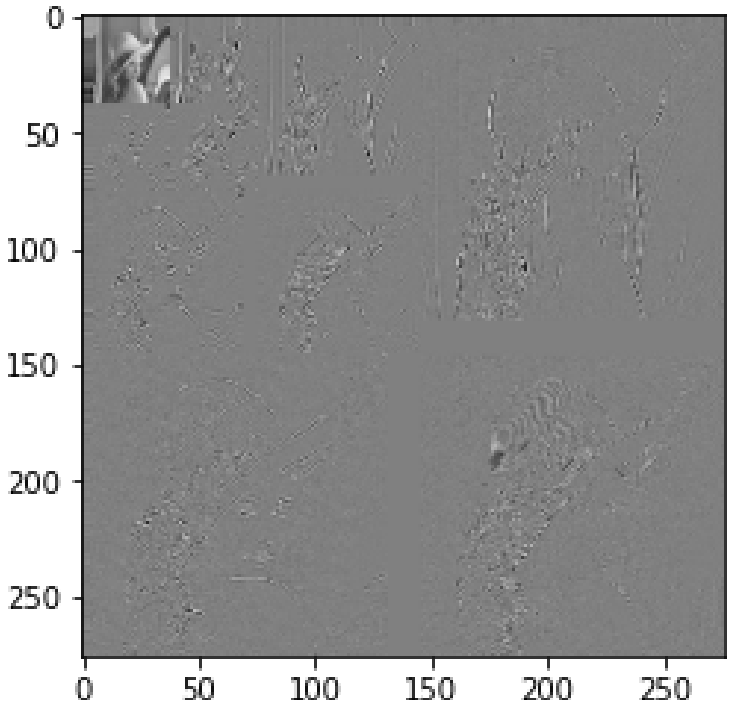

FIGURE 3.2.4 – Composantes (niveaux = 3).

La synthèse se fait à l'aide de `waverecn` pour réaliser une décomposition inverse.

Manipulation: 6 Synthétisez à nouveau l'image originale et vérifiez que les différences entre l'image reconstruite et l'image originale n'est due qu'à des bruits de calcul.

On a affiché l'image originale, l'image reconstruit et aussi les différences entre les 2 images. En observant la différence entre les deux images, on peut observer que la différence est un bruit uniformément réparti, ce qui signifie qu'aucun bruit particulier n'est introduit lors du calcul, mais plutôt une erreur de calcul provoquée par le calcul.

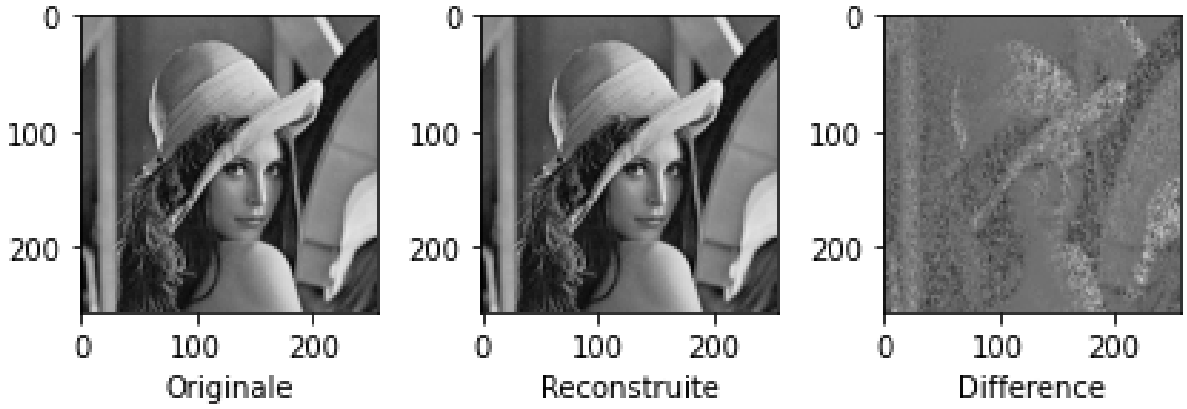

FIGURE 3.2.5 – Comparaison.

3.3 Quantification

Manipulation: 7 En vous inspirant de ce qui a été vu dans la partie sur le codage JPEG, estimez la variance des coefficients transformés dans chaque sous-bande résultant de la décomposition en ondelettes. Tracez également un histogramme des intensités lumineuses pour chaque sous-bande. Quelles sont les sous-bandes les plus importantes ? Conclusion quant à la quantification des sous-bandes pour obtenir les meilleures performances ?

Les figures d'histogrammes sont la figure 3.3.1.
En observant l'histogramme, il peut constater que la variance de l'histogramme de la sous-bande LL sera plus grande, cette sous-bande est donc la plus importante. La sous-bande LL contient les informations basse fréquence de l'image.

La variance de l'histogramme des sous-bandes LH, HL et HH est plus petite et contient des informations plus détaillées sur l'image, qui sont plus nombreuses.

Lorsque l'on considère la quantification de chaque sous-bande, pour les sous-bandes avec une plus grande variance, on peut choisir un pas de quantification plus petit, et pour les sous-bandes avec une variance plus petite, on peut choisir un pas de quantification plus grand pour réduire l'objectif de quantité de données.

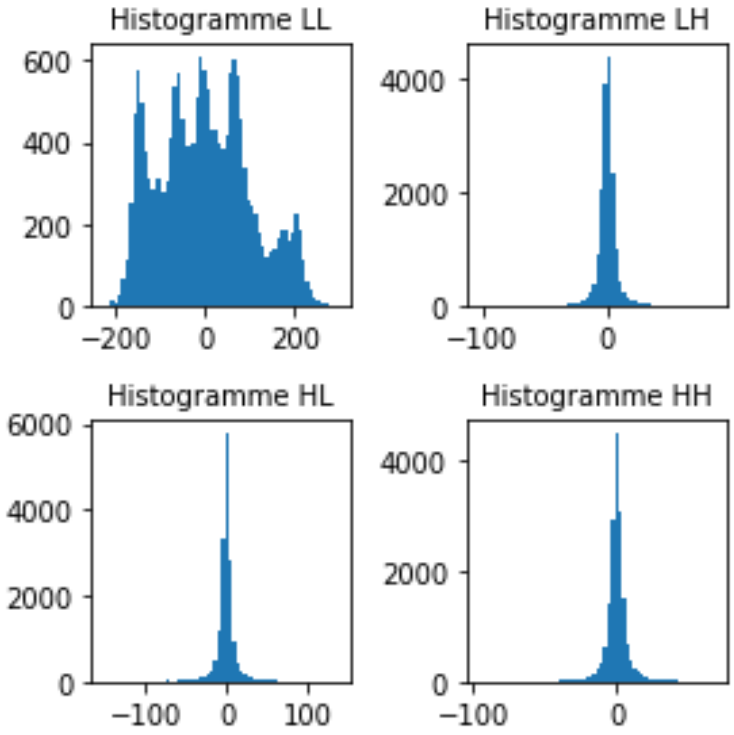

FIGURE 3.3.1 – Histogrammes.

Nous allons commencer par réaliser une quantification des sous-bandes avec le même pas de quantification.

Manipulation: 8 Visualisez dans les effets d'une quantification scalaire uniforme de toutes les sous-bandes avec un pas.

En utilisant les codes suivantes, on peut visualiser les effets d'une quantification scalaire uniforme.

Listing 22 – Manipulation 8

```
# Manipulation 8
q = 32
I_quant = copy.deepcopy(I_dwt)

# Quantification des coefficients d'approximation
I_quant[0] = np.fix(I_dwt[0] / q)
```

```
7
8  # Quantification des coefficients de detail
9  for detail_level in range(niveaux):
10     I_quant[detail_level + 1] = [np.fix(d/q) for d in I_dwt[detail_level + 1]]
11
12 arr2, slices2 = pywt.coeffs_to_array(I_quant)
13 plt.imshow(arr2, cmap=plt.cm.gray)
14 plt.show()
```

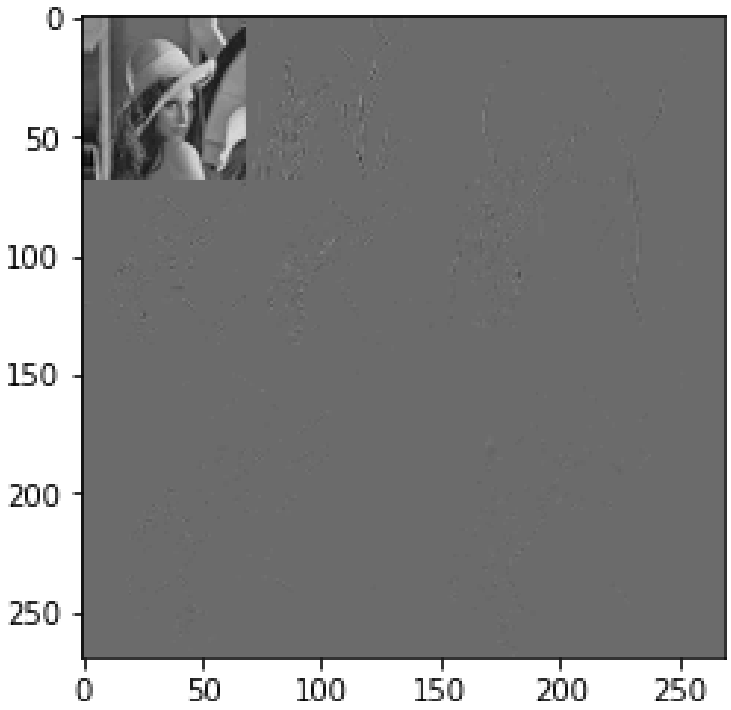

FIGURE 3.3.2 – Quantification scalaire.

Manipulation: 9 Evaluer la distorsion introduite par la quantification et le PSNR (en dB) de l'image reconstruite).

En utilisant la même méthode que dans la partie JPEG, on peut calculer la valeur de distorsion et la valeur de PSNR. L'image reconstruit est dans la figure 3.3.3.

En observant la figure reconstruit, on peut voir que la qualité de reonstruction n'est pas le meilleur avec cette valeur de pas de quantification.

Manipulation: 10 Modifiez le pas de quantification pour faire varier la qualité de reconstruction.

En utilisant la même méthode que dans la partie JPEG, on peut évaluer l'évolution de la distorsion et du PSNR en fonction de la taille du pas de quantification.

En observant ces deux courbes de changement, on peut conclure que lorsque la taille du pas de quantification est plus grande, la distorsion entre l'image reconstruite et l'image originale est plus grande et la valeur PSNR est plus petite, ce qui signifie que la qualité de la reconstruction est moins bonne. Au contraire, plus la taille du pas de quantification est petite, plus la distorsion entre l'image reconstruite et l'image originale est faible, et plus le PSNR est grand. Par conséquent, on doit choisir une taille de pas de quantification appropriée pour obtenir des résultats de reconstruction acceptables.

FIGURE 3.3.3 – Image reconstruit.

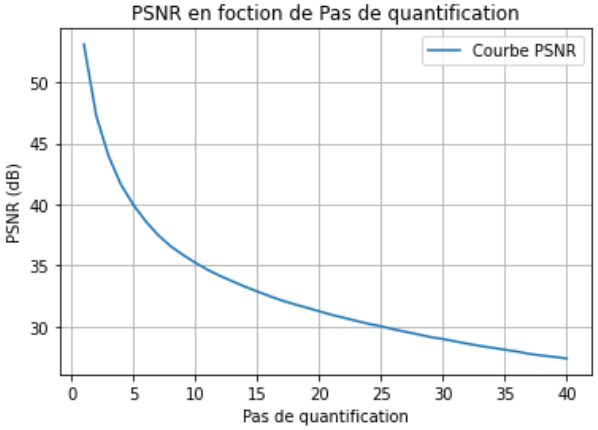

FIGURE 3.3.4 – PSNR en foction de pas.

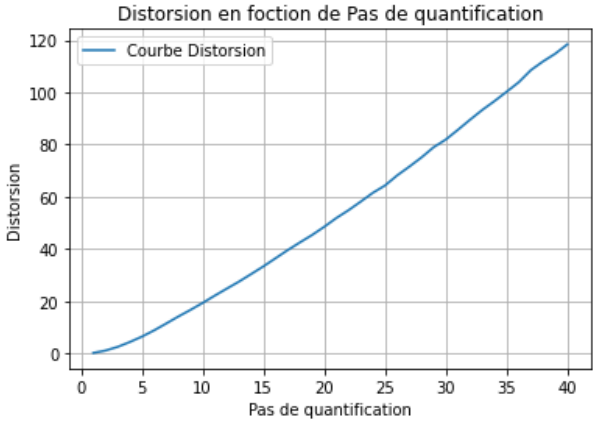

FIGURE 3.3.5 – Distorsions en foction de pas.

3.4 Codage entropique

De la même manière que dans la partie JPEG, on peut avoir une première idée du débit binaire nécessaire en évaluant l'entropie des différentes sous-bandes en sortie de quantificateur.

Manipulation: 11 Proposer une fonction permettant d'évaluer la quantité de bit par pixel à transmettre en moyenne en tenant compte qu'un codeur entropique par sous-bande sera utilisé. Attention, tenez compte du nombre de pixels par sous-bande.

Puisque le nombre de pixels dans chaque sous-bande est différent, il est un peu plus compliqué de calculer la valeur d'entropie. On a construit une fonction pour calculer la valeur d'entropie.

Pn peut obtenir que l'entropie d'image quantifiée est 0.613 bits/symbole

Listing 23 – Output de Console - Entropie

```
Bit par pixel : 0.613 bits/symbole
```

Listing 24 – Manipulation 11

```
# Manipulation 11
def bit_par_pixel(I_quant):

```

```
    H = []
    R = 0;
    size = 0;
    H.append(entropy.entropy(I_quant[0].flatten().tolist()))
    R += H[-1] * I_quant[0].size
    size += I_quant[0].size
    for detail_level in range(niveaux):
        for d in I_quant[detail_level + 1]:
            H.append(entropy.entropy(d.flatten().tolist()))
            R += H[-1] * d.size
            size += d.size
    R = R / size
    return R

bits = bit_par_pixel(I_quant)
```

Manipulation: 12 Tracer le PSNR en fonction du débit.

On peut tracer la courbe de PSNR en fonction du débit. Le résultat est dans la figure 3.4.1. On peut voir que si nous voulons obtenir une valeur PSNR plus élevée, nous avons besoin de plus de débits.

La figure 3.4.1 est un peu différent que la figure 2.5.1. A priori, ces 2 figures soient presque identiques, on choisit la figure 2.5.1 pour les questions suivantes.

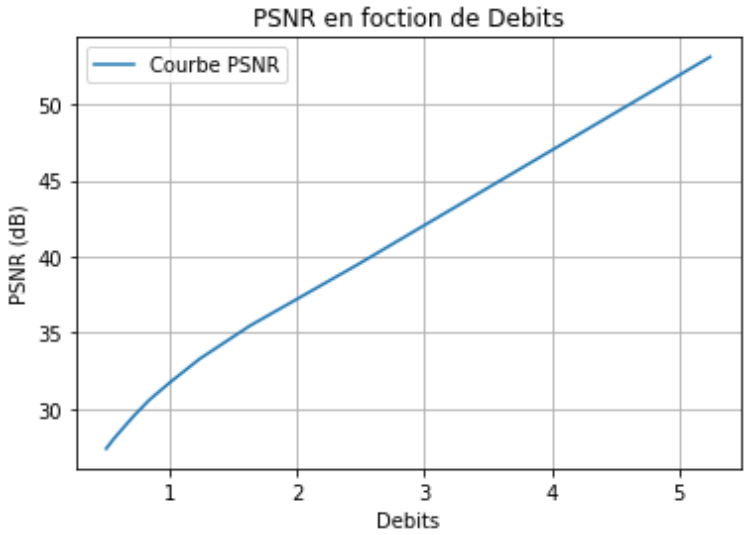

FIGURE 3.4.1 – Trace de PSNR en fonction du débit.

Manipulation: 13 Est-il intéressant de faire varier le pas de quantification pour chaque sous-bande ?

Comme nous l'avons analysé précédemment dans Manipulation 7, on peut choisir différentes tailles de pas de quantification pour différentes sous-bandes.

Pour la sous-bandesLL, on peut choisir un pas de quantification plus petit, et pour les sous-bandes LH, HL et HH, on peut choisir un pas de quantification plus grand pour réduire l'objectif de quantité de données.

Manipulation: 14 Proposez un dispositif permettant de faire une allocation optimale de débit, c'est à dire à débit global fixé, qui optimise la qualité des images à l'aide de courbes débit-distorsion.

3.5 Synthèse

Rassemblez toutes les briques formant un codeur JPEG 2000 et répondez aux questions suivantes.

Manipulation: 15 Quel est le débit minimal qui permette d'obtenir des images comprimées sans perte visible à l'oeil ?

Dans Manipulation 15 de la partie JPEG, on a analysé que lorsque le PSNR est supérieur ou égal à 30 dB, il n'est pas facile de le distinguer à l'œil nu. En utilisant la figure obtenue précédente dans la partie JPEG 2.5.1

Donc, le débit minimal est environ 1.2 bit/symbole.

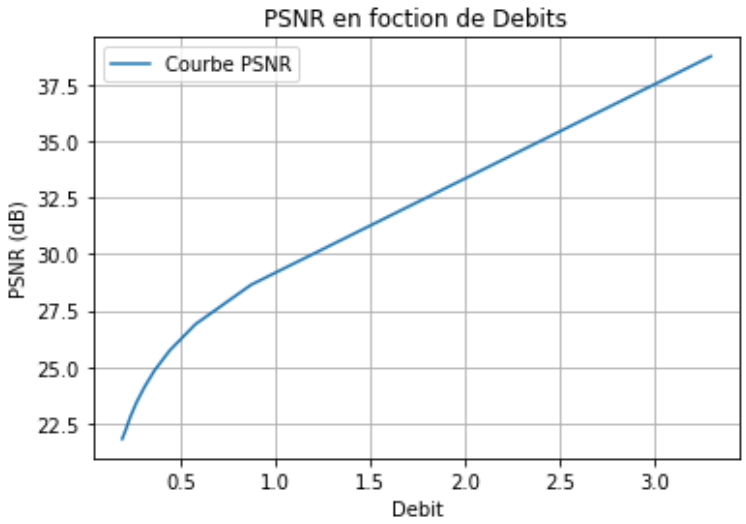

FIGURE 3.5.1 – PSNR en foction de Debits.

Manipulation: 16 Quel est le débit minimal pour avoir une qualité acceptable ?

On a affiché les figures dont PSNR sont 24.85 dB, 24.85 dB et 22.79 dB. On peut voir que lorsque le PSNR est approximativement égal à 26 dB, l'image reconstruite a atteint un niveau acceptable.

De cette manière, le débit minimum est d'environ 0.5 bit/symbole.

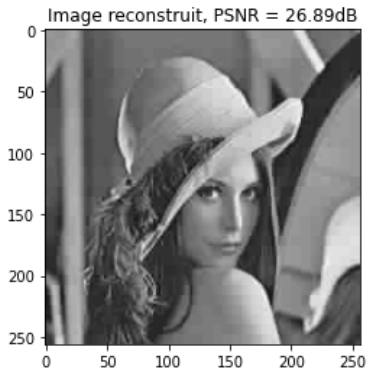

FIGURE 3.5.2 – Lenna $PSNR = 26.89$ dB.

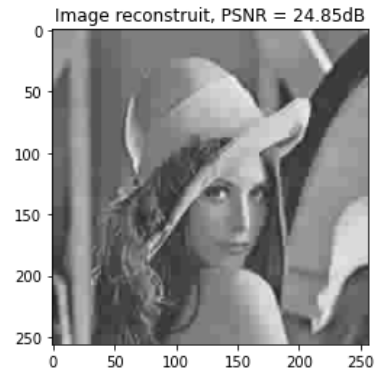

FIGURE 3.5.3 – Lenna $PSNR = 24.85$ dB.

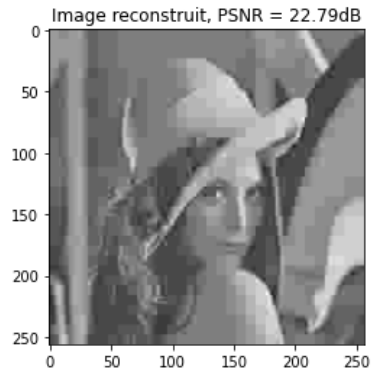

FIGURE 3.5.4 – Lenna $PSNR = 22.79$ dB.

Manipulation: 17 Quelle est l'influence du banc de filtres utilisé ?

Différents filtres de transformation en ondelettes ont des effets différents sur différentes images. Par conséquent, l'utilisation de différents bancs de filtres peut obtenir différents effets de compression, et les valeurs PSNR résultantes seront également différentes.

On peut choisir différents types d'ondelettes pour obtenir le meilleur effet de compression.

Manipulation: 18 Quelle est l'influence du nombre de niveaux de décompositions ?

Lorsque le niveau de transformation en ondelettes est plus grand, les détails haute fréquence de l'image seront décomposés plus en détail. Par conséquent, un meilleur effet de compression peut être obtenu lors de la compression. Mais à mesure que le niveau augmente, l'effet ne s'améliore pas toujours, on doit donc choisir un niveau d'ondelettes appropriée.

4 Conclusion

Pendant ce TP, on a fait
— étudié la méthode de compression JPEG et JPEG2000.
— comparé également ces 2 méthodes et les influences de la méthode d'ondelette.

[1]

Références

[1] M. Kieffer. *UE 455 – Théorie de l'Information - Codage de Source.* Université Paris-Saclay, 2023-2024.