R 5.A.06

2025 – 2026

Sensibilisation à la programmation multimédia

Corrigé du TP n° 2
Images

ANNE Jean-François
D'après le TD de F. DOIDY

Sensibilisation à la programmation multimédia

Le but de ce TP est de se familiariser avec la programmation Python pour les images.

A. Echantillonnage et quantification d'une image

Dans ce TP nous allons analyser l'impact d'une diminution de la quantification ou de l'échantillonnage d'une image. Nous verrons ensuite comment suréchantillonner une image.

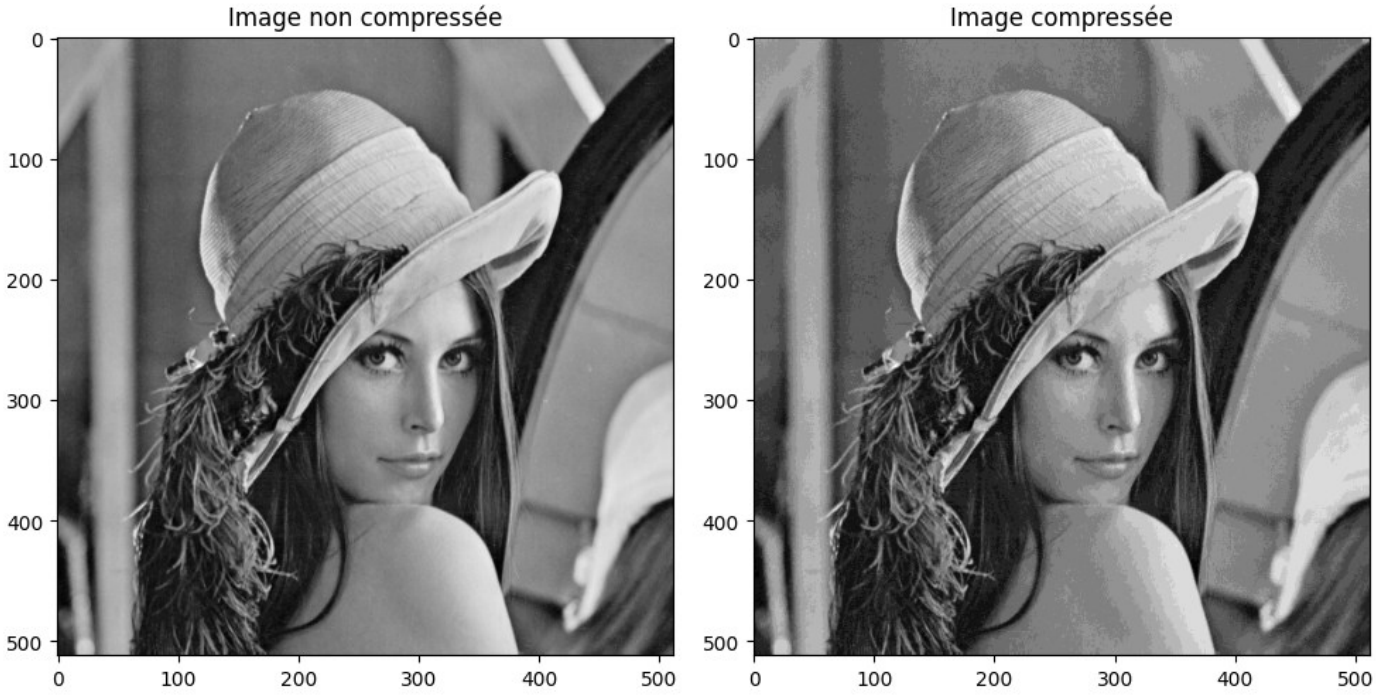
1°) Exercice 1 : Compression par diminution du nombre de bits

Ecrire un programme python qui prend une image qui affiche le nombre de bits dans lequel est codée l'image et qui diminue le nombre de bits de chaque pixel pour compresser l'image. Par exemple passer de 8 bits à 4 bits

- Afficher l'une à côté de l'autre l'image originale et l'image compressée.
- Afficher la taille en octets de l'image originale, de l'image compressée et le taux de compression.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os

# Charger l'image
image_path = "D:\Cours\Electronique\DUT
Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png"
image = Image.open(image_path)

# Conversion de l'image en niveau de gris
image_gray = image.convert("L")

# Convertir l'image en un tableau NumPy
image_array = np.array(image_gray)

# Réduire le nombre de bits
bits = 4
image_compressed = (image_array // (256 // 2 ** bits)) * (256 // 2 ** bits)

# Créer une nouvelle image à partir du tableau compressé
image_compressed = Image.fromarray(image_compressed)

# Afficher les images et calculer le taux de compression
# Taille de l'image non compressée en octets
original_size = os.path.getsize(image_path)
# Taille de l'image compressée en octets
compressed_size = image_compressed.size[0] * \
    image_compressed.size[1] * bits // 8
compression_ratio = original_size / compressed_size

# Affichage des images et de leurs tailles respectives
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image_gray, cmap="gray")
axs[0].set_title("Image non compressée")
axs[1].imshow(image_compressed, cmap="gray")
```

```
axs[1].set_title("Image compressée")

plt.tight_layout()
plt.show()

print(f"Taille de l'image non compressée : {original_size} octets")
print(f"Taille de l'image compressée : {compressed_size} octets")
print(f"Taux de compression : {compression_ratio:.2f}")
```




```
Taille de l'image non compressée : 473831 octets
Taille de l'image compressée : 131072 octets
Taux de compression : 3.62
```

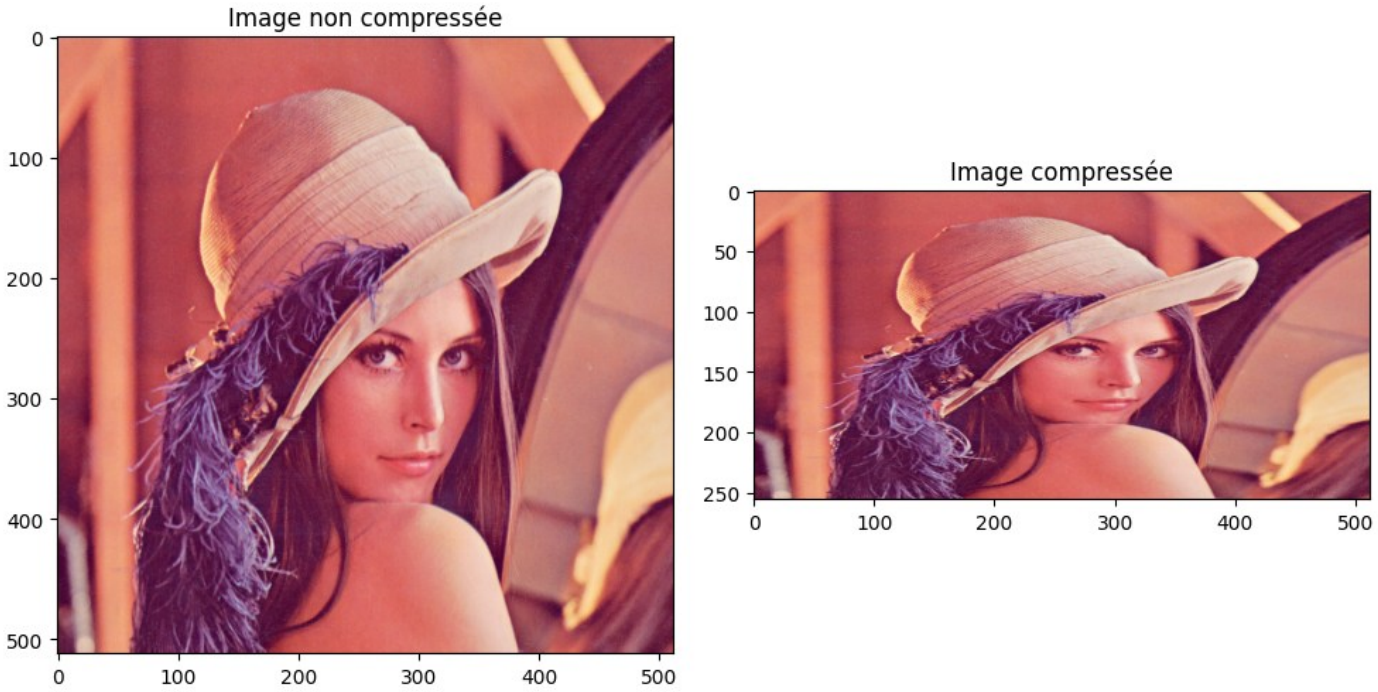
2°) ***Exercice 2 : Compression par diminution du nombre de lignes***

Ecrire un programme python qui prend une image qui réduit la taille de l'image en ne prenant qu'une ligne sur 2

- Afficher l'une à côté de l'autre l'image originale et l'image compressée.
- Afficher la taille en octets de l'image originale, de l'image compressée et le taux de compression.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os

# Charger l'image
image_path = "D:\Cours\Electronique\DUT
Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png"
image = Image.open(image_path)

# Convertir l'image en un tableau NumPy
image_array = np.array(image)

# Compression en ne prenant qu'une ligne sur deux
```

```
compressed_array = image_array[::2, :]

# Créer une nouvelle image à partir du tableau compressé
compressed_image = Image.fromarray(compressed_array)

# Afficher les images et calculer le taux de compression
# Taille de l'image non compressée en octets
original_size = os.path.getsize(image_path)
# Taille de l'image compressée en octets (supposant une image RVB)
compressed_size = compressed_image.size[0] * compressed_image.size[1] * 3
compression_ratio = original_size / compressed_size

# Affichage des images et de leurs tailles respectives
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image)
axs[0].set_title("Image non compressée")
axs[1].imshow(compressed_image)
axs[1].set_title("Image compressée")

plt.tight_layout()
plt.show()

print(f"Taille de l'image non compressée : {original_size} octets")
print(f"Taille de l'image compressée : {compressed_size} octets")
print(f"Taux de compression : {compression_ratio:.2f}")
```




```
Taille de l'image non compressée : 473831 octets
Taille de l'image compressée : 393216 octets
Taux de compression : 1.21
```

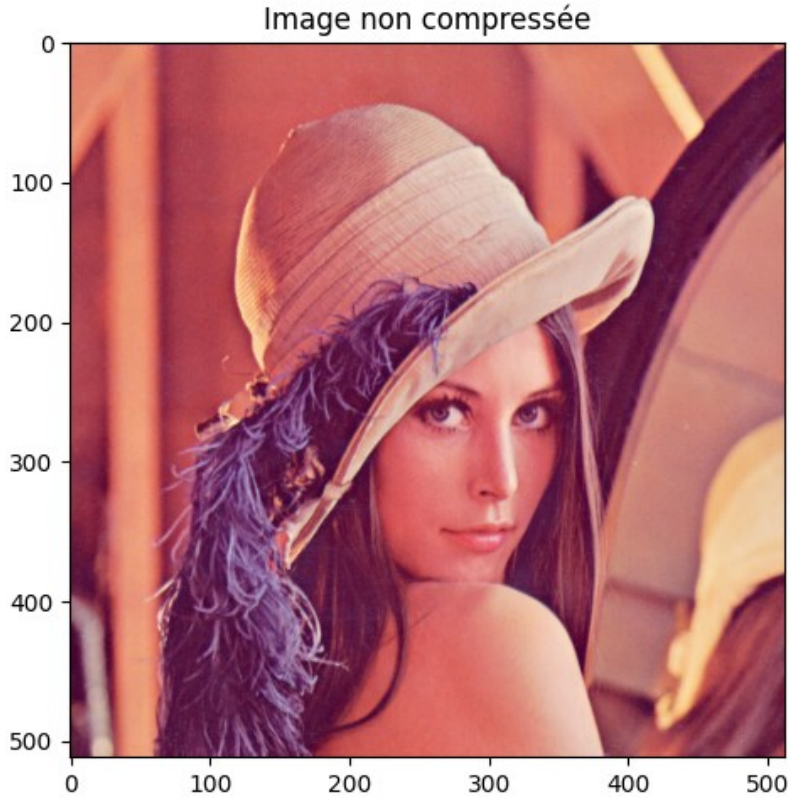
3°) ***Exercice 3 : Compression par diminution du nombre de colonnes***

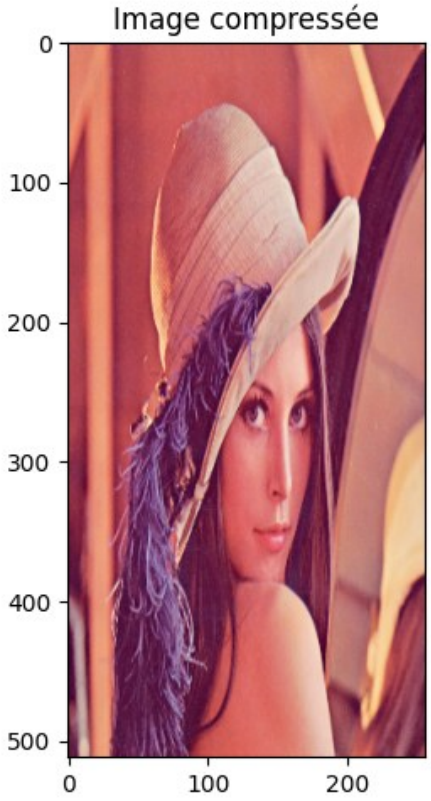
Ecrire un programme python qui prend une image qui réduit la taille de l'image en ne prenant qu'une colonne sur 2

- Afficher l'une à côté de l'autre l'image originale et l'image compressée.

- Afficher la taille en octets de l'image originale, de l'image compressée et le taux de compression.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os

# Charger l'image
image_path = "D:\Cours\Electronique\DUT
Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png"
image = Image.open(image_path)

# Convertir l'image en un tableau NumPy
image_array = np.array(image)

# Compression en ne prenant qu'une colonne sur deux
compressed_array = image_array[:, ::2]

# Créer une nouvelle image à partir du tableau compressé
compressed_image = Image.fromarray(compressed_array)

# Afficher les images et calculer le taux de compression
# Taille de l'image non compressée en octets
original_size = os.path.getsize(image_path)
# Taille de l'image compressée en octets (supposant une image RVB)
compressed_size = compressed_image.size[0] * compressed_image.size[1] * 3
compression_ratio = original_size / compressed_size

# Affichage des images et de leurs tailles respectives
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image)
axs[0].set_title("Image non compressée")
axs[1].imshow(compressed_image)
axs[1].set_title("Image compressée")

plt.tight_layout()
plt.show()

print(f"Taille de l'image non compressée : {original_size} octets")
print(f"Taille de l'image compressée : {compressed_size} octets")
print(f"Taux de compression : {compression_ratio:.2f}")
```






```
Taille de l'image non compressée : 473831 octets
Taille de l'image compressée : 393216 octets
Taux de compression : 1.21
```

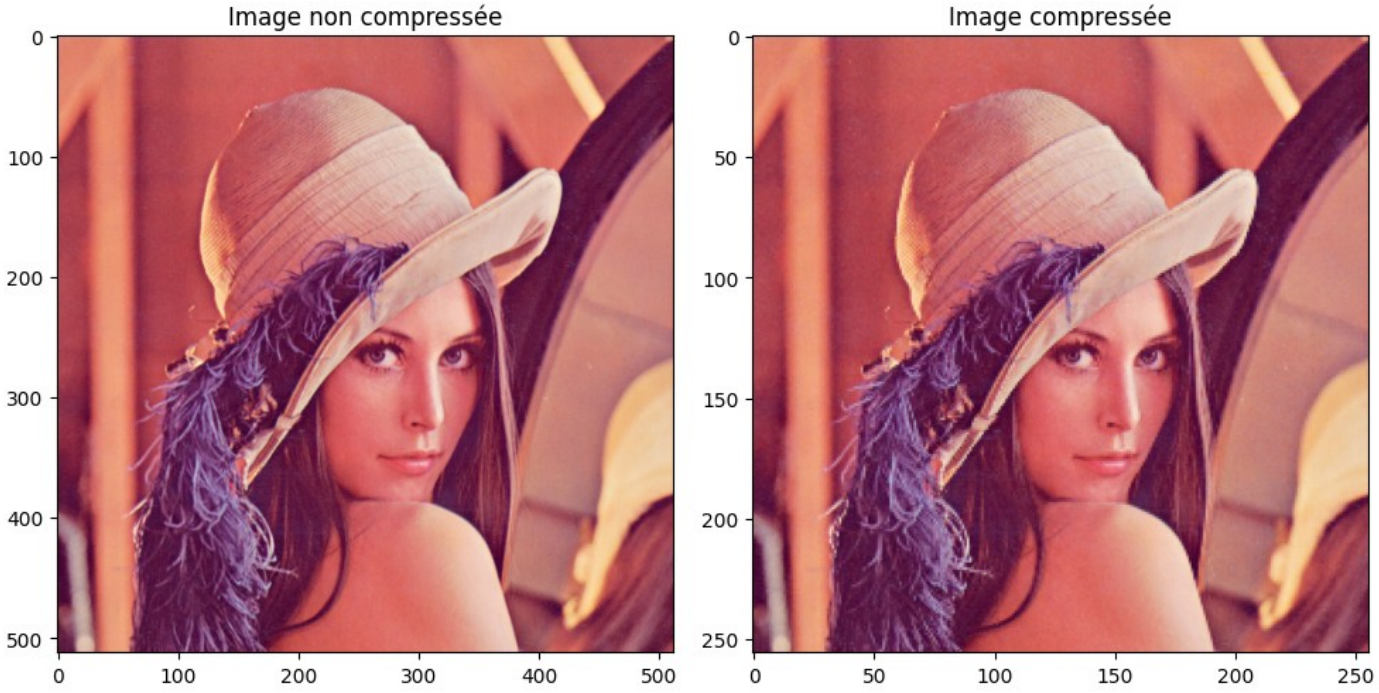
4°) Exercice 4: Compression par diminution du nombre de lignes et de colonnes

Ecrire un programme python qui prend une image qui réduit la taille de l'image en ne prenant qu'une ligne sur 2 et qu'une colonne sur 2

- Afficher l'une à côté de l'autre l'image originale et l'image compressée.
- Afficher la taille en octets de l'image originale, de l'image compressée et le taux de compression.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os

# Charger l'image
image_path = "D:\Cours\Electronique\DUT
Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png"
image = Image.open(image_path)

# Convertir l'image en un tableau NumPy
image_array = np.array(image)

# Compression en ne prenant qu'une colonne sur deux et une ligne sur deux
compressed_array = image_array[::2, ::2]

# Créer une nouvelle image à partir du tableau compressé
compressed_image = Image.fromarray(compressed_array)

# Afficher les images et calculer le taux de compression
```

```
# Taille de l'image non compressée en octets
original_size = os.path.getsize(image_path)
# Taille de l'image compressée en octets (supposant une image RVB)
compressed_size = compressed_image.size[0] * compressed_image.size[1] * 3
compression_ratio = original_size / compressed_size

# Affichage des images et de leurs tailles respectives
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image)
axs[0].set_title("Image non compressée")
axs[1].imshow(compressed_image)
axs[1].set_title("Image compressée")

plt.tight_layout()
plt.show()

print(f"Taille de l'image non compressée : {original_size} octets")
print(f"Taille de l'image compressée : {compressed_size} octets")
print(f"Taux de compression : {compression_ratio:.2f}")
```




```
Taille de l'image non compressée : 473831 octets
Taille de l'image compressée : 196608 octets
Taux de compression : 2.41
```

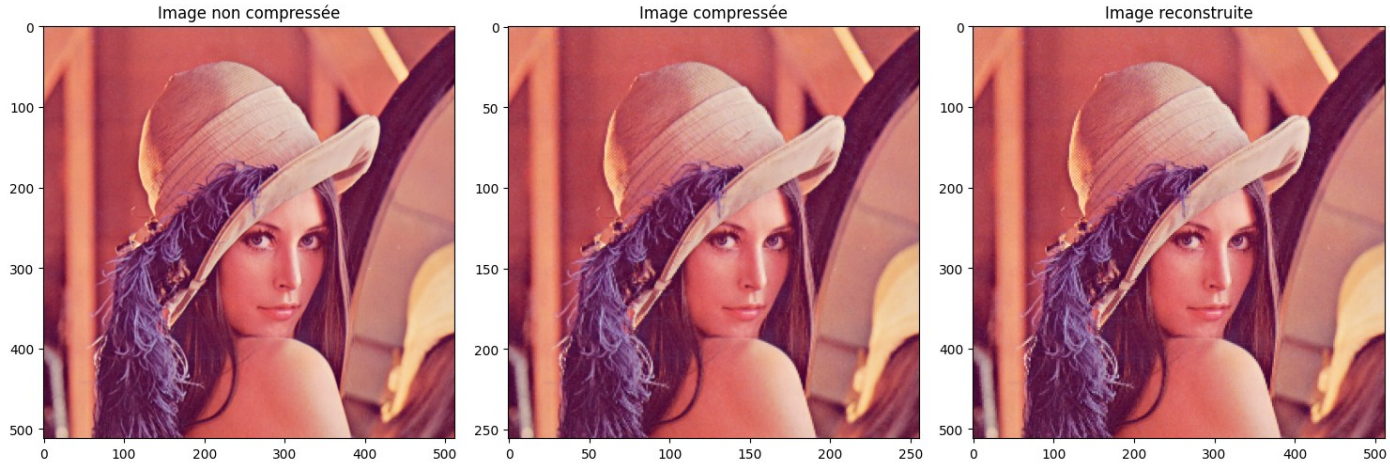
Afficher l'une à côté de l'autre l'image originale et l'image compressée.
Afficher la taille en octets de l'image originale, de l'image compressée et le taux de compression.

5°) ***Exercice 5: Duplication des lignes et des colonnes pour retrouver l'image d'origine***

Ecrire un programme python qui prend une image qui réduit la taille de l'image en ne prenant qu'une ligne sur 2 et qu'une colonne sur 2, puis dupliquant chaque ligne et chaque colonne, pour retrouver une image de la taille égale à celle de l'image originale.

- Afficher l'une à côté de l'autre l'image originale et l'image compressée et l'image décompressée. Mettre un titre à chaque image.
- Afficher la taille en octets de l'image originale, de l'image compressée, de l'image reconstituée et le taux de compression.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os

# Charger l'image
image_path = "D:\Cours\Electronique\DUT
Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png"
image = Image.open(image_path)

# Convertir l'image en un tableau NumPy
image_array = np.array(image)

# Compression en ne prenant qu'une colonne sur deux et une ligne sur deux
compressed_array = image_array[::2, ::2]

# Recréer l'image en recopiant une ligne sur deux et une colonne sur deux
reconstructed_array = np.repeat(
    np.repeat(compressed_array, 2, axis=0), 2, axis=1)

# Créer une nouvelle image à partir du tableau compressé et reconstruit
compressed_image = Image.fromarray(compressed_array)
reconstructed_image = Image.fromarray(reconstructed_array)

# Enregistrer l'image reconstruite sous un nouveau nom de fichier
reconstructed_image_path = "reconstructed_image.jpg"
reconstructed_image.save(reconstructed_image_path)

# Afficher les images et calculer le taux de compression
# Taille de l'image non compressée en octets
original_size = os.path.getsize(image_path)
# Taille de l'image compressée en octets (supposant une image RVB)
compressed_size = compressed_image.size[0] * compressed_image.size[1] * 3
# Taille de l'image reconstruite en octets
reconstructed_size = os.path.getsize(reconstructed_image_path)
compression_ratio = original_size / compressed_size

# Affichage des images et de leurs tailles respectives
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].imshow(image)
axs[0].set_title("Image non compressée")
axs[1].imshow(compressed_image)
axs[1].set_title("Image compressée")
axs[2].imshow(reconstructed_image)
axs[2].set_title("Image reconstruite")

plt.tight_layout()
plt.show()

print(f"Taille de l'image non compressée : {original_size} octets")
print(f"Taille de l'image compressée : {compressed_size} octets")
print(f"Taille de l'image reconstruite : {reconstructed_size} octets")
print(f"Taux de compression : {compression_ratio:.2f}")
```




```
Taille de l'image non compressée : 473831 octets
Taille de l'image compressée : 196608 octets
Taille de l'image reconstruite : 40519 octets
Taux de compression : 2.41
```

6°) ***Exercice 6 : Duplication de la moyenne des pixels des lignes et des colonnes pour retrouver l'image d'origine***

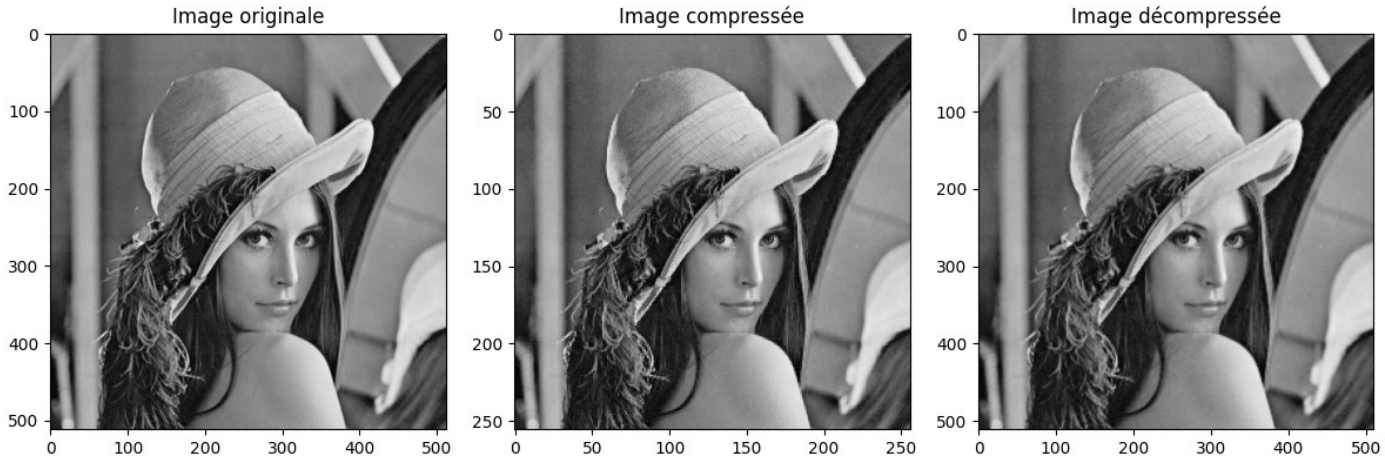
Ecrire un programme python qui prend une image qui réduit la taille de l'image en ne prenant qu'une ligne sur 2 et qu'une colonne sur 2, puis en créant les lignes et les colonnes correspondant à la moyenne de 2 lignes et des colonnes contiguës, pour retrouver une image de la taille égale à celle de l'image originale.

- Afficher l'une à côté de l'autre l'image originale et l'image compressée et l'image décompressée. Mettre un titre à chaque image.
- Afficher la taille en octets de l'image originale, de l'image compressée, de l'image reconstituée et le taux de compression.

L'image finale sera de taille 511 x 511

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Charger l'image
image_path = "D:\Cours\Electronique\DUT
Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png"
image = Image.open(image_path).convert("L")

# Convertir l'image en un tableau NumPy
ima = np.array(image)

# Compression en ne prenant qu'une ligne et une colonne sur deux
ima3 = ima[::2, ::2]

# Créer une nouvelle matrice vide pour l'image décompressée
ima5 = np.empty((ima3.shape[0] * 2 - 1, ima3.shape[1]), dtype=np.uint8)

# Faire la moyenne pour les lignes intermédiaires
for i in range(ima3.shape[0] - 1):
    ima5[2 * i, :] = ima3[i, :]
    ima5[2 * i + 1, :] = (ima3[i, :].astype(float) +
                          ima3[i + 1, :].astype(float)) / 2
```

```
# Ajouter la dernière ligne de l'image originale dans la dernière ligne de l'image
finale
ima5[-1, :] = ima3[-1, :]

# Créer une nouvelle matrice vide pour l'image décompressée après la compression des
colonnes
ima6 = np.empty((ima5.shape[0], ima5.shape[1] * 2 - 1), dtype=np.uint8)

# Faire la moyenne pour les colonnes intermédiaires
for i in range(ima5.shape[1] - 1):
    ima6[:, 2 * i] = ima5[:, i]
    ima6[:, 2 * i + 1] = (ima5[:, i].astype(float) +
                          ima5[:, i + 1].astype(float)) / 2

# Ajouter la dernière colonne de l'image originale dans la dernière colonne de
l'image finale
ima6[:, -1] = ima5[:, -1]

# La matrice que l'on vient de calculer devient l'image décompressée
ima6 = ima6.astype(np.uint8)

# Affichage des images
plt.figure(figsize=(12, 4))
plt.subplot(131)
plt.imshow(ima, cmap="gray")
plt.title("Image originale")

plt.subplot(132)
plt.imshow(ima3, cmap="gray")
plt.title("Image compressée")

plt.subplot(133)
plt.imshow(ima6, cmap="gray")
plt.title("Image décompressée")

plt.tight_layout()
plt.show()
print(f"Taille de l'image non compressée : {original_size} octets")
print(f"Taille de l'image compressée : {compressed_size} octets")
print(f"Taille de l'image reconstruite : {reconstructed_size} octets")
print(f"Taux de compression : {compression_ratio:.2f}")
```




```
Taille de l'image non compressée : 473831 octets
Taille de l'image compressée : 65536 octets
Taille de l'image reconstruite : 40519 octets
Taux de compression : 7.23
```

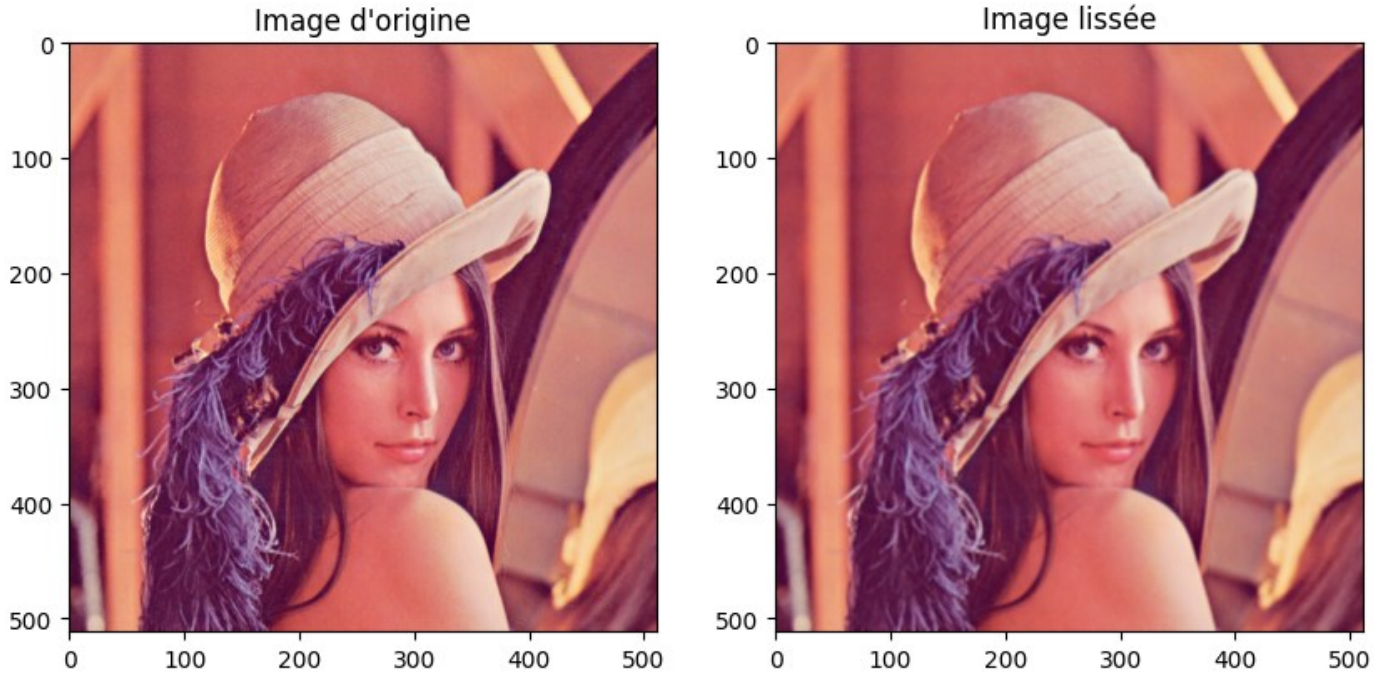
7°) ***Exercice 7 : Lisser une image***

Ecrire un programme python qui prend une image et qui lisse cette image en prenant une matrice 3x3 pour le lissage.

- Afficher l'une à côté de l'autre l'image originale et l'image lissée. Mettre un titre à chaque image.
- Afficher la taille en octets de l'image originale, de l'image lissée, et le taux de compression.

```
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt

# Charger l'image
image_path = "D:\Cours\Electronique\DUT
Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png"
image_originale = Image.open(image_path)

# Créer une copie de l'image d'origine
image_lissee = image_originale.copy()

# Définir une matrice de convolution 3x3 pour le lissage
kernel = [[1, 1, 1], [1, 1, 1], [1, 1, 1]]

# Obtenir la largeur et la hauteur de l'image
largeur, hauteur = image_originale.size

# Parcourir chaque pixel de l'image (à l'exception des bords)
for x in range(1, largeur - 1):
    for y in range(1, hauteur - 1):
        r, g, b = 0, 0, 0
        for i in range(-1, 2):
            for j in range(-1, 2):
                pixel = image_originale.getpixel((x + i, y + j))
                r += kernel[i + 1][j + 1] * pixel[0]
                g += kernel[i + 1][j + 1] * pixel[1]
                b += kernel[i + 1][j + 1] * pixel[2]
        r = int(r / 9)
```

```
            g = int(g / 9)
            b = int(b / 9)
            image_lissee.putpixel((x, y), (r, g, b))

# Afficher les images côte à côte
plt.figure(figsize=(10, 5))  # Définir la taille de la figure
plt.subplot(1, 2, 1)  # Première sous-figure
plt.title('Image d\'origine')
plt.imshow(image_originale)
# Sauvegarder l'image d'origine
image_originale.save('image_originale.jpg')  # Sauvegarder l'image d'origine

plt.subplot(1, 2, 2)  # Deuxième sous-figure
plt.title('Image lissée')
plt.imshow(image_lissee)
# Sauvegarder l'image lissée
image_lissee.save('image_lissee.jpg')  # Sauvegarder l'image lissée

plt.show()

# Titres des images
image_originale.title = "Image d'origine"
image_lissee.title = "Image lissée"
# Obtenir la taille des images
taille_originale = os.path.getsize('image_originale.jpg')
taille_lissee = os.path.getsize('image_lissee.jpg')
# Calculer le taux de compression
taux_compression = taille_lissee / taille_originale

# Afficher la taille et le taux de compression
print(f"Taille de l'image d'origine : {taille_originale} octets")
print(f"Taille de l'image lissée : {taille_lissee} octets")
print(f"Taux de compression : {taux_compression:.2f}")
```



```
Taille de l'image d'origine : 37788 octets
Taille de l'image lissée : 29724 octets
Taux de compression : 0.79
```

B. Pour aller plus loin :

1°) Détections de voitures :

Les images images1png et image2.png représentent deux photographies prises par une caméra située sur l'autoroute Paris-Lyon. Ces photographies ont été prises à quelques minutes d'intervalle. On va chercher à faire ressortir l'affichage des voitures.

Méthode :

❖ Créer une image Somme contenant l'ensemble des véhicules présents sur l'image1.png et sur l'image2.png en faisant la somme des 2 images.

❖ Créer une image Diff1 qui est la soustraction de l'image 1 et l'image 2.

❖ Effectuer l'opération Diff1 = Diff1 * (Diff1 si >40)

❖ Créer une image Diff2 qui est la soustraction de l'image 2 et l'image 1.

❖ Effectuer l'opération Diff2 = Diff2 * (Diff2 si >40)

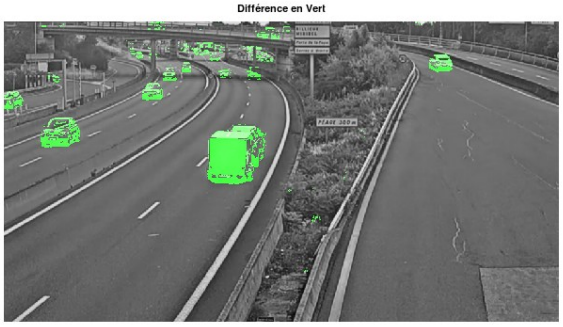
❖ Ajouter les deux images Diff1 et Diff2

❖ Créer une copie de l'image Somme dans laquelle vous modifierez le canal vert en ajoutant 255* diff aux valeurs de somme.

❖ Vous obtenez alors une image dans laquelle les véhicules en circulation seront mis en vert.

❖ Afficher, sur une même figure côte à côte :

- Les 2 images originales
- En dessous l'image contenant tous les véhicules et l'image contenant les véhicules en circulation qui sont en vert.



```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# Charger les images
# Remplacez par le chemin de votre première image
image_path1 = "Images\image1.png"
# Remplacez par le chemin de votre deuxième image
image_path2 = "Images\image2.png"

image1 = np.array(Image.open("Images\image1.png"))
image2 = np.array(Image.open("Images\image2.png"))

# Projection des véhicules sur une image
somme = (image1.astype(np.float32) + image2.astype(np.float32)) / 2
somme = np.uint8(somme)

diff1 = np.zeros_like(image1)
diff2 = np.zeros_like(image1)
diff = np.zeros_like(image1)

diff1 = np.uint8(image1 - image2)
diff1 = diff1 * np.uint8(diff1 > 80)  # 40 correspond à un seuil
diff2 = np.uint8(image2 - image1)
diff2 = diff2 * np.uint8(diff2 > 80)  # 40 correspond à un seuil

diff = diff1 + diff2


# Créer l'image diff avec trois canaux RVB
image_diff = np.zeros((image1.shape[0], image1.shape[1], 3), dtype=np.uint8)
# Affecter les canaux RVB correctement
image_diff[:, :, 0] = np.uint8(somme)
image_diff[:, :, 1] = np.uint8(somme + 255 * diff)
image_diff[:, :, 2] = np.uint8(somme)

# Affichage des images dans un sous-ensemble de graphiques
plt.figure(figsize=(12, 8))

plt.subplot(3, 2, 1)
plt.imshow(image1, cmap='gray')
plt.title('1ère image')

plt.subplot(3, 2, 2)
plt.imshow(image2, cmap='gray')
plt.title('2ème image')

plt.subplot(3, 2, 3)
plt.imshow(somme, cmap='gray')
plt.title('Somme')

plt.subplot(3, 2, 4)
plt.imshow(diff1, cmap='gray')
```

```
plt.title('diff1')

plt.subplot(3, 2, 5)
plt.imshow(diff2, cmap='gray')
plt.title('diff2')

plt.subplot(3, 2, 6)
plt.imshow(image_diff)
plt.title('Différence en Vert')

plt.show()
```

C. Compression d'image (x2)

1°) Compression et décompression « sans perte » :

a) Compression

Le « Run Length Encoding » (RLE) est le plus simple des algorithmes de compression de données. Il remplace les successions de deux ou plusieurs caractères identiques par un nombre représentant la longueur du passage, suivi du caractère d'origine.

Par simplification, dans ce TP, vous coderez de la même manière le fait que le caractère soit isolé ou qu'il existe une succession de caractères identiques.

Algorithme de compression

- Recherche des caractères répétés plus de n fois (vous prendrez n=1).
- Remplacement de l'itération de caractères par :
    1. un caractère spécial identifiant une compression. Exemple : @.
    2. le nombre de fois où le caractère est répété.
    3. le caractère répété

Exemple : AAAAARRRRRROLLLLBBBBBUUTTTTTT.

On choisit comme caractère spécial : @

et comme seuil de répétition : 3.

Après compression : @5A@6RO@4L@5BUU@6T

gain : 11 caractères soit 38%

Il faudrait utiliser cet algorithme sur une image n'ayant pas trop de changement de gris ! (image ayant des bandes de couleurs horizontales de part et d'autre et en format PNG sans compression ou en BMP, et en codant sur 8 bits).

```
def compress_string(s, threshold=3):
    compressed = []  # Liste pour stocker la version compressée de la chaîne
    count = 1  # Compteur pour le nombre de répétitions consécutives
    special_char = '@'  # Caractère spécial pour la compression
    gain = 0  # Initialisation du gain

    for i in range(1, len(s)):
        if s[i] == s[i - 1]:
            count += 1
```

```
            else:
                if count >= threshold:
                    compressed.append(f'{special_char}{count}{s[i - 1]}')
                    gain += count - 1
                else:
                    compressed.extend([s[i - 1]] * count)
                count = 1

        if count >= threshold:
            compressed.append(f'{special_char}{count}{s[-1]}')
            gain += count - 1
        else:
            compressed.extend([s[-1]] * count)

        compressed_string = ''.join(compressed)
        original_length = len(s)
        compressed_length = len(compressed_string)
        compression_ratio = (
            original_length - compressed_length) / original_length * 100

        return compressed_string, gain, compression_ratio


    # Chaîne d'entrée
    input_string = "AAAAARRRRROLLLLBBBBBUUTTTTTT"

    # Compression
    compressed_result, gain, compression_ratio = compress_string(
        input_string, threshold=3)

    # Affichage du résultat
    print("Chaîne d'origine:", input_string)
    print("Chaîne compressée:", compressed_result)
    print(f"Gain obtenu: {gain} caractères soit {compression_ratio:.2f}%")
```

```
Chaîne d'origine: AAAAARRRRROLLLLBBBBBUUTTTTTT
Chaîne compressée: @5A@6RO@4L@5BUU@6T
Gain obtenu: 21 caractères soit 37.93%
```

b) Décompression

Algorithme de décompression :

Durant la lecture du fichier compressé, lorsque le caractère spécial est reconnu, on effectue l'opération inverse de la compression tout en supprimant ce caractère spécial.

Exemple : @5A@6RO@4L@5BUU@6T

On choisit comme caractère spécial : @ et comme seuil de répétition : 3.

Après décompression : AAAAARRRRROLLLLBBBBBUUTTTTTT

gain : 11 caractères soit 38%

```
    def decompress_string(s, special_char='@'):
        decompressed = []  # Liste pour stocker la version décompressée de la chaîne
        i = 0
```

```
    while i < len(s):
        if s[i] == special_char:
            count = int(s[i + 1])  # Récupère le nombre de répétitions
            char_to_repeat = s[i + 2]  # Récupère le caractère répété
            decompressed.extend([char_to_repeat] * count)
            i += 3
        else:
            decompressed.append(s[i])
            i += 1

    return ''.join(decompressed)


# Chaîne compressée
compressed_string = "@5A@6RO@4L@5BUU@6T"

# Décompression
decompressed_result = decompress_string(compressed_string)

# Affichage du résultat
print("Chaîne compressée:", compressed_string)
print("Chaîne décompressée:", decompressed_result)

Chaîne compressée: @5A@6RO@4L@5BUU@6T
Chaîne décompressée: AAAAARRRRRROLLLLBBBBBUUTTTTTT
```

c) Compression d'une image

Appliquer l'algorithme de compression précédent à une image BMP.

d) Décompression d'une image

Appliquer l'algorithme de décompression précédent pour réobtenir l'image BMP.

D. Webographie :

- *http://nsinfo.yo.fr/cours_python/python_10_images.html*
- *http://tvaira.free.fr/dev/python/python-images.html*
- *https://www.cours-gratuit.com/tutoriel-python/tutoriel-python-les-bases-de-traitement-dimages-en-python-bibliothque-numpy*
- *https://www.cours-gratuit.com/tutoriel-python/tutoriel-python-les-bases-de-traitement-dimages-avec-scipy*
- *https://www.cours-gratuit.com/tutoriel-python/tutoriel-python-les-bases-de-traitement-dimages-avec-scikit-image*
- *https://www.cours-gratuit.com/tutoriel-python/tutoriel-python-les-bases-de-traitement-dimages-en-python-opencv*
- *https://ensip.gitlab.io/pages-info/ressources/transverse/tuto_images.html*
- *https://www.codingame.com/playgrounds/53303/apprendre-python-dans-le-secondaire/manipulations-dimages-i*
- *https://compression.fiches-horaires.net/la-compression-avec-perte-1/le-compression-jpeg/*
- *https://scikit-image.org/skimage-tutorials/lectures/three_dimensional_image_processing.html*