



Master SDA

Traitement de données multimédia

TP 1 : Manipulations de base en Traitement d'Images avec Python

Encadré par :

- **Pr. Alioua Nawal**

Réalisée par :




- **BENAGUERRI Safaa**
- **AGUERCHI Saida**



2. Lecture et affichage d'images

2.1 Lecture d'une image entière

1. Téléchargement des 3 images de LenaC, LenaB et LenaT et les placer dans le répertoire de travail.

Nom	Modifié le	Type	Taille
 lenaB	10/03/2025 20:58	Fichier BMP	66 Ko
 lenaC	10/03/2025 20:56	Fichier JPG	4 Ko
 lenaT	10/03/2025 20:56	Fichier TIF	263 Ko

2. Lecture et affichage des 3 images en utilisant OpenCV

Code python :

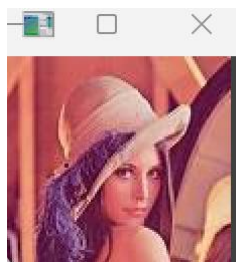
```
import cv2

# Charger les images
imageB = cv2.imread("LenaB.bmp") # Remplace avec le bon chemin
imageC = cv2.imread("LenaC.jpg")
imageT = cv2.imread("LenaT.tif")

# Afficher les images
cv2.imshow("Image LenaB - OpenCV", imageB)
cv2.imshow("Image LenaC - OpenCV", imageC)
cv2.imshow("Image LenaT - OpenCV", imageT)

# Attendre une touche pour fermer les fenêtres
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Résultat :





Comment OpenCv considère-t-elle les images?

- **OpenCV** considère une image comme un **tableau NumPy**.
- L'image est stockée en **BGR** au lieu de **RGB**.
- Une image en niveaux de gris n'a qu'une seule dimension.
- On peut facilement accéder et modifier les pixels

3. Lecture et affichage des 3 images en utilisant Pillow

Code python :

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

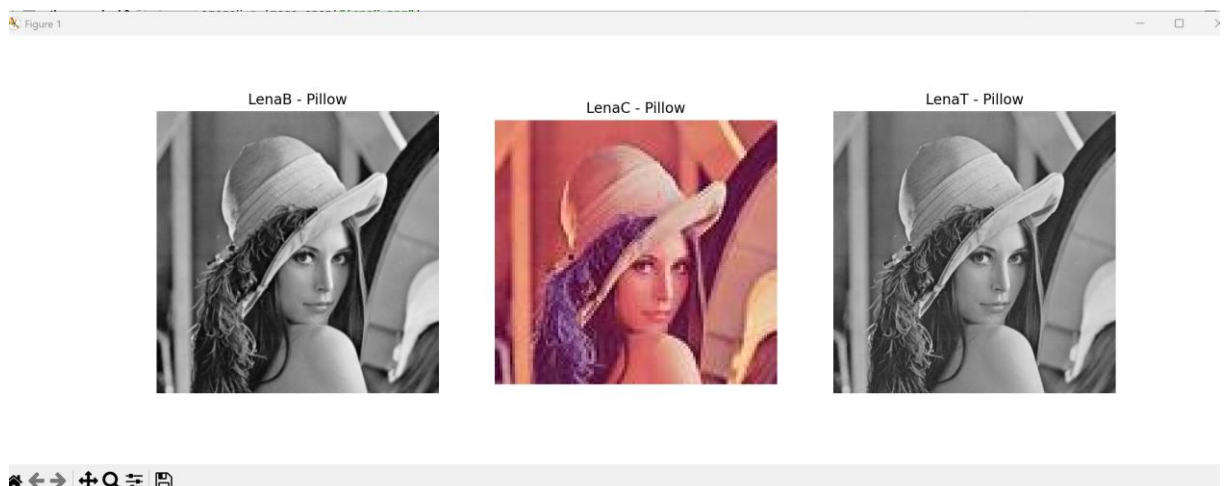
imageB = Image.open("lenaB.bmp").convert("L") # Convertir en niveaux de gris
imageC = Image.open("lenaC.jpg")
imageT = Image.open("lenaT.tif") # Déjà en niveaux de gris

# Afficher les images
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(imageB, cmap="gray") # Affichage correct en noir et blanc
axes[0].set_title("LenaB - Pillow")
axes[1].imshow(imageC)
axes[1].set_title("LenaC - Pillow")
axes[2].imshow(imageT, cmap="gray") # Affichage correct en noir et blanc
axes[2].set_title("LenaT - Pillow")

# Supprimer les axes
for ax in axes:
    ax.axis("off")

plt.show()
```

Résultat:





Comment Pillow considère-t-elle les images?

- Lorsque nous utilisons Pillow (PIL), nous avons besoin de matplotlib pour afficher correctement les images.
- **Pillow** permet de lire, modifier et sauvegarder des images, mais il n'a pas de fonction native pour afficher les images directement dans une fenêtre interactive comme **OpenCV**.
- Il fournit une méthode **.show()**, mais celle-ci ouvre une visionneuse d'images externe, ce qui n'est pas toujours pratique
- **matplotlib.pyplot.imshow()** est souvent utilisé pour afficher des images dans une interface graphique.
- Contrairement à OpenCV, matplotlib utilise le format **RGB** par défaut, qui est compatible avec **Pillow**
- **convert("L")** force lenaB.bmp à être en niveaux de gris (comme lenaT.tif).
- **cmap="gray"** s'assure que matplotlib n'applique pas une colormap par défaut

4. Affichage des informations relatives aux 3 images : nombre de lignes, de colonnes, nombre de canaux, format, nom :

Code python :

```
# Fonction pour afficher les infos d'une image
def info_image(image, nom):
    print(f"Informations pour {nom}:")
    print(f"- Format : {image.format}") # Format du fichier (BMP, JPEG, TIFF,
etc.)
    print(f"- Taille (Lignes x Colonnes) : {image.size[1]} x
{image.size[0]}") # (hauteur, largeur)
    print(f"- Mode : {image.mode}") # Mode couleur ("RGB", "L" pour niveaux
de gris, "P" pour palette)
    print(f"- Nombre de canaux : {len(image.getbands())}") # Nb de canaux (1
pour "L", 3 pour "RGB")
    print("-" * 40)

# Afficher les infos des 3 images
info_image(imageB, "lenaB.bmp")
info_image(imageC, "lenaC.jpg")
info_image(imageT, "lenaT.tif")
```

Résultat :



```
C:\Users\safaa\PycharmProjects\pythonmarrakech2\venv\Scripts\python.exe C:/Users/safaa/PycharmProjects/pythonmarrakech2/q1-4.py
Informations pour lena8.bmp:
- Format : BMP
- Taille (Lignes x Colonnes) : 256 x 256
- Mode : L
- Nombre de canaux : 1
-----
Informations pour lenaC.jpg:
- Format : JPEG
- Taille (Lignes x Colonnes) : 104 x 111
- Mode : RGB
- Nombre de canaux : 3
-----
Informations pour lenaT.tif:
- Format : TIFF
- Taille (Lignes x Colonnes) : 512 x 512
- Mode : P
- Nombre de canaux : 1
-----
Process finished with exit code 0
```

Quelle bibliothèque python avez-vous utilisé et pourquoi?

On a utiliser pillow car :

- **Facile à utiliser** : Permet de charger, manipuler et analyser les images en quelques lignes de code.
- **Compatible avec plusieurs formats** : BMP, JPEG, TIFF, PNG, etc.
- **Permet d'accéder aux métadonnées** : Taille, mode, format, etc.

5. Discuter la manière dont les canaux de l'image sont stockés

Les canaux d'une image représentent les différentes composantes de couleur. Leur stockage dépend du **mode** de l'image.

Stockage des Canaux en Fonction du Mode de l'Image

Mode "L" (Grayscale / Niveaux de gris)

- **Stocke un seul canal (1 canal).**
- Chaque pixel est représenté par une valeur d'intensité entre 0 (noir) et 255 (blanc).
- **Matrice 2D** de dimensions (hauteur x largeur)

Mode "RGB" (Couleur)

- **Stocke 3 canaux** : Rouge (R), Vert (G), Bleu (B).
- Chaque pixel est représenté par un triplet (R, G, B), où chaque valeur varie entre 0 et 255.
- **Stockage sous forme de 3 matrices 2D** (une pour chaque couleur).
- Représentation sous forme de tableau NumPy de dimensions (**hauteur x largeur x 3**).

Mode "RGBA" (Couleur avec transparence)



- **Stocke 4 canaux** : Rouge (R), Vert (G), Bleu (B), Alpha (A).
- Le canal Alpha (A) représente la transparence (0 = transparent, 255 = opaque).
- **Stockage sous forme de 4 matrices 2D.**

Mode "P" (Palette)

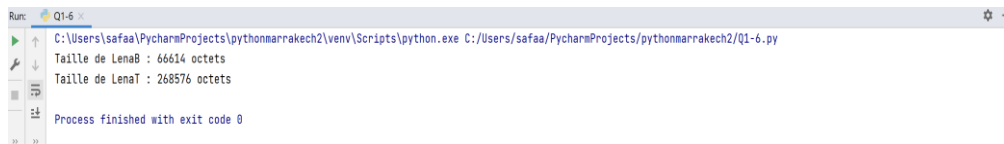
- Utilisé pour **les images indexées** comme certaines .bmp ou .gif.
- Chaque pixel stocke un **indice** vers une **palette de couleurs**.
- Doit être **converti en "RGB" ou "L"** pour une manipulation plus simple.

6. Comparaison du rendu visuel des 2 images niveaux de gris et la taille du fichier image

```
import os
taille_B = os.path.getsize("lenaB.bmp") # Taille en octets
taille_T = os.path.getsize("lenaT.tif") # Taille en octets

print(f"Taille de LenaB : {taille_B} octets")
print(f"Taille de LenaT : {taille_T} octets")
```

Résultat :



Quelle image fournit la meilleure qualité visuelle ?

Interprétation des tailles des fichiers :

- **LenaB.bmp** : 66 614 octets (**plus petit**).
- **LenaT.tif** : 268 576 octets (**plus grand**).

Le format **BMP** ne compresse pas l'image, donc il conserve tous les pixels **sans perte de qualité**.

Le format **TIFF** peut être **non compressé (qualité parfaite)** ou **compressé avec perte**.

2.2 Lecture d'une ligne, d'une colonne ou d'un pixel

Utiliser l'image LenaB, puis répéter pour l'image LenaC

Code python :



```
# Charger l'image en niveaux de gris (LenaB) et couleur (LenaC)
imageB = Image.open("lenaB.bmp").convert("L") # Convertir en niveaux de gris
imageC = Image.open("lenaC.jpg").convert("RGB") # Garder la couleur
```

```
# Convertir en tableau NumPy pour manipulation
arrayB = np.array(imageB)
arrayC = np.array(imageC)
```

#QUESTION 1

```
# Trouver l'index de la ligne centrale
ligne_centrale = arrayB.shape[0] // 2 # Hauteur // 2
ligne_centrale2 = arrayC.shape[0] // 2

# Extraire les valeurs de la ligne centrale
ligne_B = arrayB[ligne_centrale, :]
ligne_C = arrayC[ligne_centrale2, :]# Image en niveaux de gris

# Afficher les valeurs de la ligne centrale
print(f"Ligne centrale de LenaB (Niveaux de gris) : {ligne_B}")
print(f"Ligne centrale de LenaC (Niveaux de gris) : {ligne_C}")
```

#QUESTION 2

```
colonne_centrale_B = arrayB.shape[1] // 2 # Largeur // 2
colonne_centrale_C = arrayC.shape[1] // 2 # Largeur // 2

# Extraire les valeurs de la colonne centrale
colonne_B = arrayB[:, colonne_centrale_B] # Image en niveaux de gris
colonne_C = arrayC[:, colonne_centrale_C] # Image en couleur

# Afficher les valeurs de la colonne centrale
print(f"Colonne centrale de LenaB (Niveaux de gris) : {colonne_B}")
print(f"Colonne centrale de LenaC (RGB) : {colonne_C}")
```

#QUESTION 3

```
pixel_central_B = arrayB[arrayB.shape[0] // 2, arrayB.shape[1] // 2] # Pixel
central pour LenaB (niveaux de gris)
pixel_central_C = arrayC[arrayC.shape[0] // 2, arrayC.shape[1] // 2] # Pixel
central pour LenaC (RGB)

# Afficher la valeur du pixel central
print(f"Valeur du pixel central de LenaB (Niveaux de gris) :
{pixel_central_B}")
print(f"Valeur du pixel central de LenaC (RGB) : {pixel_central_C}")
```

- Sélection de la ligne centrale de l'image et affichage de ses valeurs.

```
Run: partie3
C:\Users\safaa\PycharmProjects\pythonmarakech2\venv\Scripts\python.exe C:/Users/safaa/PycharmProjects/pythonmarakech2/partie3.py
Ligne centrale de Lena8 (Niveaux de gris) : [100 101 101 104 102 101 99 98 100 94 92 85 85 98 121 134 148 156
164 172 176 176 174 176 176 176 175 168 162 153 141 116 94 76 74 80
84 86 95 96 98 100 101 103 100 98 100 99 99 98 94 98 101 100
104 110 120 135 140 145 153 156 154 152 155 172 160 165 162 118 76 68
63 48 67 57 66 85 114 86 55 82 76 109 62 32 42 80 81 42
50 89 44 39 38 52 86 102 46 43 44 43 46 46 42 39 45 46
34 112 145 105 162 183 168 172 183 196 201 195 182 172 182 197 159 86
88 90 85 92 69 66 66 61 48 60 78 56 50 57 84 118 100 127
121 119 120 125 137 148 156 159 157 169 180 192 192 196 194 192 190 185
160 139 124 116 84 70 57 52 54 62 62 64 71 66 72 50 44 37
47 75 66 168 162 114 96 54 49 73 60 65 60 53 50 48 48 50
49 53 49 41 56 97 134 141 135 128 139 153 161 160 160 158 157 154
155 155 158 157 156 154 153 155 156 156 156 156 156 156 154 153 152
152 152 154 156 152 155 154 154 156 154 153 150 147 146 147 148 144 144
143 138 138 134]
Ligne centrale de LenaC (Niveaux de gris) : [[168 67 71]
[172 68 75]
[171 65 79]
[172 64 80]
[174 61 81]
[168 58 71]
[184 80 81]
[217 121 109]
[229 139 113]
[236 152 116]
[236 154 117]
[236 154 120]]
```

- Sélection de la colonne centrale de l'image et affichage de ses valeurs.

```
C:\Users\safaa\PycharmProjects\pythonmarrakech2\venv\Scripts\python.exe C:/Users/safaa/PycharmProjects/pythonmarrakech2/partie3.py
Colonne centrale de LenaB (Niveaux de gris) : [140 140 137 134 131 128 130 128 132 132 133 128 130 131 130 133 133 132
132 132 130 128 123 120 117 122 176 192 194 190 194 186 187 189 186 194
190 192 192 193 192 185 170 192 211 210 200 203 203 198 200 196 195 180
190 198 198 196 200 194 196 192 186 193 176 170 181 180 183 186 185 176
174 176 176 175 176 162 167 168 162 169 146 162 164 161 160 114 125 108
60 61 107 84 101 77 69 84 102 62 39 43 54 92 102 87 83 86
95 148 173 163 164 165 166 176 180 186 187 184 182 186 194 195 190 138
116 110 85 53 51 48 77 100 101 102 100 102 131 144 149 148 158 171
172 173 173 174 174 173 166 166 166 159 162 161 153 151 150 151 150 151
150 149 146 145 145 142 140 143 142 138 138 138 142 143 147 144 144 140
137 135 134 131 133 133 131 132 126 126 120 99 70 102 132 130 128 126
124 126 126 127 126 125 117 121 136 132 128 130 133 133 136 134 134 138
137 137 137 137 138 139 137 139 140 138 138 141 142 140 142 142 143 141
140 140 138 138 140 139 140 138 139 139 138 136 140 141 142 139 138 138
141 140 134 135]
Colonne centrale de LenaC (RGB) : [[204 97 91]
[206 97 92]
[208 96 94]
[209 97 95]
[210 96 96]
[208 96 95]
[204 94 95]
[199 95 94]
[203 108 106]
[199 97 95]]
```

- Sélection de le pixel central et a affichage de sa valeur.

[227 114 107]]

Valeur du pixel central de LenaB (Niveaux de gris) : 85

Valeur du pixel central de LenaC (RGB) : [234 170 160]

3.Écriture d'images

3.1 Écriture d'une image entière



1. Écriture de l'image LenaB sous le format jpg et sous le format png

Code python :

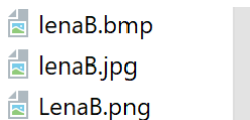
```
from PIL import Image

# Charger l'image LenaB
imageB = Image.open("LenaB.bmp")

# Enregistrer l'image en format JPG
imageB.save("LenaB.jpg", "JPEG")

# Enregistrer l'image en format PNG
imageB.save("LenaB.png", "PNG")
```

Résultat :



2. Affichage de LenaB, Lena.jpg et lena.png et comparer le rendu visuel et la taille des 3 fichiers de l'image, que constatez-vous?

Code python :

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

# Charger les trois images
image_B = Image.open("lenaB.bmp")
image_jpg = Image.open("lenaB.jpg")
image_png = Image.open("lenaB.png")

# Créer une figure avec des sous-graphes pour afficher les trois images côte à côte
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(imageB, cmap="gray") # Affichage correct en noir et blanc
axes[0].set_title("imageB - Pillow")
axes[0].axis('off')

axes[1].imshow(image_jpg, cmap="gray")
axes[1].set_title("image_jpg - Pillow")
axes[1].axis('off')

axes[2].imshow(image_png, cmap="gray") # Affichage correct en noir et blanc
```



```
axes[2].set_title("image_png - Pillow")
axes[2].axis('off')

# Vérifier la taille des fichiers
size_bmp = os.stat("LenaB.bmp").st_size
size_jpg = os.stat("LenaB.jpg").st_size
size_png = os.stat("LenaB.png").st_size

print(f"Taille de LenaB.bmp : {size_bmp} octets")
print(f"Taille de LenaB.jpg : {size_jpg} octets")
print(f"Taille de LenaB.png : {size_png} octets")
```

Résultat :



- Le **rendu visuel** des trois formats peut être similaire, mais le format **BMP** offrira la meilleure qualité

3.2 Écriture d'une partie de l'image

1. Extraire la sous-image correspondant à un quart de LenaT et l'enregistrer

Code python :

```
# Charger l'image LenaT
imageT = Image.open("lenaT.tif")

# Obtenir les dimensions de l'image
width_T, height_T = imageT.size
```



```
# Extraire un quart de l'image (coin supérieur gauche)
quarter_image_T = imageT.crop((0, 0, width_T // 2, height_T // 2))

# Enregistrer la sous-image extraite
quarter_image_T.save("LenaT_quarter.tif")
image_T = Image.open("LenaT_quarter.tif")
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(image_T, cmap="gray") # Affichage correct en noir et blanc
axes[0].set_title("quarter_lenaT")
axes[0].axis('off')
```

2. Même question pour LenaC :

#Q2 : la même chose pour lenaC

```
imageC = Image.open("lenaC.jpg")

# Obtenir les dimensions de l'image
width_C, height_C = imageC.size

# Extraire un quart de l'image (coin supérieur gauche)
quarter_image_C = imageC.crop((0, 0, width_C // 2, height_C // 2))

# Enregistrer la sous-image extraite
quarter_image_C.save("LenaC_quarter.jpg")
image_B = Image.open("LenaC_quarter.jpg")
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(image_B, cmap="gray") # Affichage correct en noir et blanc
axes[0].set_title("quarter_lenaC")
axes[0].axis('off')
```

Résultat :

Figure 2



Figure 1



3.3 Convertir en image niveaux de gris



1. Convertir l'image LenaC en image niveaux de gris, l'afficher et l'enregistrer en format jpg en utilisant Pillow :

Code python :

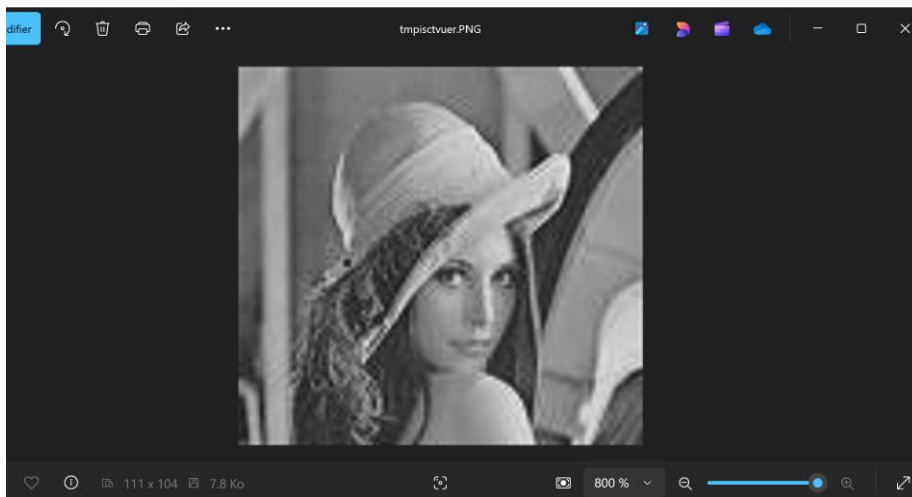
```
# Charger l'image LenaC
imageC = Image.open("lenaC.jpg")

# Convertir l'image en niveaux de gris
image_gray_C_pillow = imageC.convert("L")

# Afficher l'image en niveaux de gris
image_gray_C_pillow.show()

# Enregistrer l'image en niveaux de gris en format jpg
image_gray_C_pillow.save("LenaC_gray_pillow.jpg")
```

Résultat :



3. Même question en utilisant Opencv :

Code python :

```
# Charger l'image LenaC
imageC_opencv = cv2.imread("lenaC.jpg")

# Convertir l'image en niveaux de gris
image_gray_C_opencv = cv2.cvtColor(imageC_opencv, cv2.COLOR_BGR2GRAY)

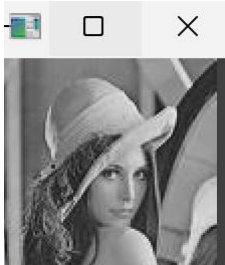
# Afficher l'image en niveaux de gris
cv2.imshow("lenaC in Gray - OpenCV", image_gray_C_opencv)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Enregistrer l'image en niveaux de gris en format jpg
```



```
cv2.imwrite("lenaC_gray_opencv.jpg", image_gray_C_opencv)
plt.show()
```

Résultat :



3. Est-il possible de rendre une image niveaux de gris en image couleurs?

- **Conversion en niveaux de gris** : C'est possible en utilisant à la fois **Pillow** et **OpenCV**, et les deux méthodes sont efficaces.
- **Restaurer l'image en couleurs** : Impossible sans information supplémentaire, mais vous pouvez appliquer des palettes de couleurs pour "coloriser" une image en niveaux de gris de manière artificielle.

4. Rotation d'images

Code python :

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

# Charger l'image
image = Image.open("lenaB.bmp") # Remplace avec LenaC.jpg si besoin

# Effectuer une rotation de 45°
image_rotated = image.rotate(45, expand=True) # expand=True pour éviter le recadrage

# Afficher les images originale et tournée
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
axes[0].imshow(image, cmap="gray")
axes[0].set_title("Image originale")
axes[1].imshow(image_rotated, cmap="gray")
axes[1].set_title("Image après rotation de 45°")

for ax in axes:
    ax.axis("off")

plt.show()
```

Résultat :



On a utilisé :

- `image.rotate(45)` : Applique une rotation de 45°.
- `expand=True` : Évite que l'image soit **recadrée** en agrandissant le canevas.
- Affichage avec `matplotlib` pour comparer **l'image originale et la version tournée**

On remarque que

- **La qualité est conservée**, car la rotation ne modifie pas les couleurs ni les valeurs des pixels
- **Des zones noires peuvent apparaître** autour de l'image après la rotation, car les pixels en dehors du cadre original sont remplis en noir.

5.Redimensionnement d'images :

1-Redimensionner d'une image Lena de votre choix en la divisant par deux dans chaque dimension (hauteur et largeur) avec Pillow

Code python :

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

# Charger l'image
image = Image.open("lenaB.bmp") # Remplace avec LenaC.jpg si besoin

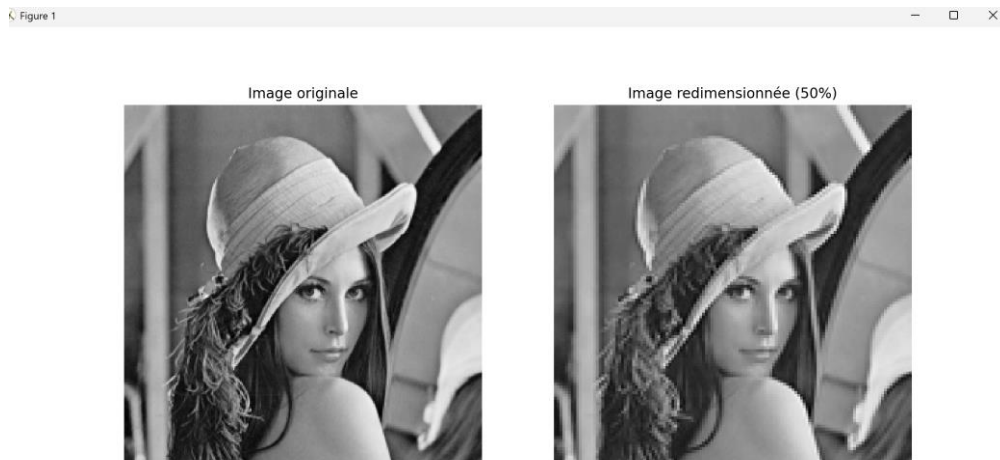
# Récupérer les dimensions actuelles
largeur, hauteur = image.size
print(f"Dimensions originales : {largeur}x{hauteur}")

# Calculer les nouvelles dimensions (division par 2)
nouvelle_largeur = largeur // 2
nouvelle_hauteur = hauteur // 2
```



```
print(f"Dimensions après redimensionnement :  
{nouvelle_largeur}x{nouvelle_hauteur}")  
  
# Appliquer le redimensionnement  
image_resized = image.resize((nouvelle_largeur, nouvelle_hauteur),  
Image.LANCZOS)  
  
# Afficher les images originale et redimensionnée  
fig, axes = plt.subplots(1, 2, figsize=(12, 6))  
axes[0].imshow(image, cmap="gray")  
axes[0].set_title("Image originale")  
  
axes[1].imshow(image_resized, cmap="gray")  
axes[1].set_title("Image redimensionnée (50%) par open CV")  
  
for ax in axes:  
    ax.axis("off")  
  
plt.show()
```

Résultat:



```
Run: section4 x section 5 x  
C:\Users\safaa\PycharmProjects\pythonmarrakech2\venv\Scripts\python.exe "C:/Users/safaa/PycharmProjects/pythonmarrakech2/section 5.py"  
Dimensions originales : 256x256  
Dimensions après redimensionnement : 128x128
```

2-Même question avec OpenCV

Code python :

```
import cv2  
# Charger l'image en niveaux de gris  
image = cv2.imread("lenaB.bmp", cv2.IMREAD_GRAYSCALE)
```



```
# Récupérer les dimensions actuelles
hauteur, largeur = image.shape
print(f"Dimensions originales : {largeur}x{hauteur}")

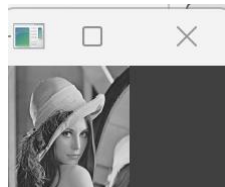
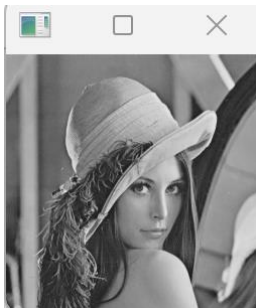
# Calculer les nouvelles dimensions (division par 2)
nouvelle_largeur = largeur // 2
nouvelle_hauteur = hauteur // 2
print(f"Dimensions après redimensionnement :
{nouvelle_largeur}x{nouvelle_hauteur}")

# Redimensionner l'image
image_resized = cv2.resize(image, (nouvelle_largeur, nouvelle_hauteur),
interpolation=cv2.INTER_AREA)

# Afficher les images originale et redimensionnée
cv2.imshow("Image originale", image)
cv2.imshow("Image redimensionnée (50%)", image_resized)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Résultat:



```
C:\Users\safaa\PycharmProjects\pythonmarrakech2\ven
Dimensions originales : 256x256
Dimensions après redimensionnement : 128x128
```

3-Comparaison des deux méthodes de redimensionnement ?

Critère	Pillow (PIL)	OpenCV (cv2)
Méthode utilisée	.resize()	cv2.resize()
Interpolation	Image.LANCZOS (haute qualité)	cv2.INTER_AREA (optimisé pour réduction)



Format des images	Garde le mode (RGB, L)	Par défaut, charge en BGR
Affichage	matplotlib	cv2.imshow()

6.Changement du pas de quantification

6.1 Réduction du nombre de niveaux de gris

- Une image en niveaux de gris est codée sur **8 bits** par défaut → **256 niveaux (0-255)**.
- En **4 bits**, il ne reste que **16 niveaux de gris** (0, 17, 34, ..., 255).
- En **2 bits**, il ne reste que **4 niveaux de gris** (0, 85, 170, 255).

Formule pour quantifier l'image à N bits :

- **image_quantifiee** = (image / 255) * (2**N - 1)
- **image_quantifiee** = np.round(image_quantifiee) * (255 / (2**N - 1))

1. Affichage de l'image originale LenaB et l'image quantifiée à 4 bits.

Code python :

```
# Charger l'image en niveaux de gris
image = cv2.imread("lenaB.bmp", cv2.IMREAD_GRAYSCALE)

# Fonction de quantification à N bits
def quantification(image, bits):
    niveaux = 2 ** bits # Nombre de niveaux (16 pour 4 bits, 4 pour 2 bits)
    image_quantiee = np.round(image / 255 * (niveaux - 1)) * (255 / (niveaux - 1))
    return image_quantiee.astype(np.uint8)

# Quantification à 4 bits (16 niveaux)
image_4bits = quantification(image, 4)

# Quantification à 2 bits (4 niveaux)
image_2bits = quantification(image, 2)

# Affichage des images
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(image, cmap="gray")
axes[0].set_title("Image originale ,")

axes[1].imshow(image_4bits, cmap="gray")
```



```
axes[1].set_title("Quantification 4 bits (16 niveaux)")
```

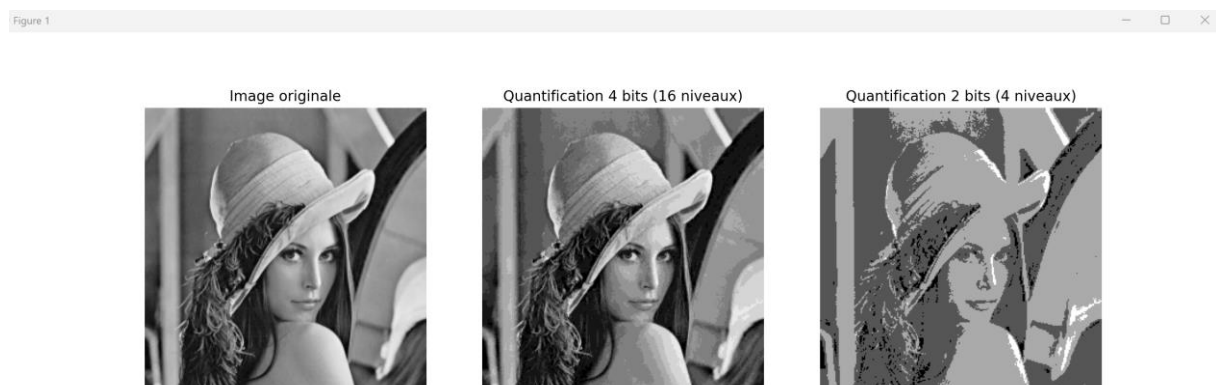
```
axes[2].imshow(image_2bits, cmap="gray")
```

```
axes[2].set_title("Quantification 2 bits (4 niveaux)")
```

```
for ax in axes:  
    ax.axis("off")
```

```
plt.show()
```

Résultat :



2. Quels changements visuels observez-vous? Que se passe-t-il si l'on réduit encore le nombre de bits à 2?

- **En 4 bits (16 niveaux de gris) :** L'image reste détaillée, mais certaines nuances subtiles disparaissent et l'effet "**posterization**" apparaît légèrement (zones uniformes).
- **En 2 bits (4 niveaux de gris) :** Perte massive des détails et l'effet de **seuil** est visible (seulement 4 teintes de gris).

Conclusion :

- Plus le nombre de bits est réduit, plus l'image devient "**simplifiée**".
- Avec 2 bits, l'image est très limitée en contraste.
- 4 bits est un bon compromis pour économiser de la mémoire tout en gardant une image compréhensible.

6.2 Réduction de la profondeur couleur

1. Réduire pour LenaC le nombre de bits à 4 pour chaque canal, puis à 2

Code python :



```
import cv2
import matplotlib.pyplot as plt

# Charger l'image en couleur (RGB)
image = cv2.imread("lenaC.jpg")
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convertir BGR → RGB pour
affichage correct

# Quantification à 4 bits (16 niveaux par canal)
image_4bits = quantification(image, 4)

# Quantification à 2 bits (4 niveaux par canal)
image_2bits = quantification(image, 2)

# Affichage des images
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(image)
axes[0].set_title("Image originale")

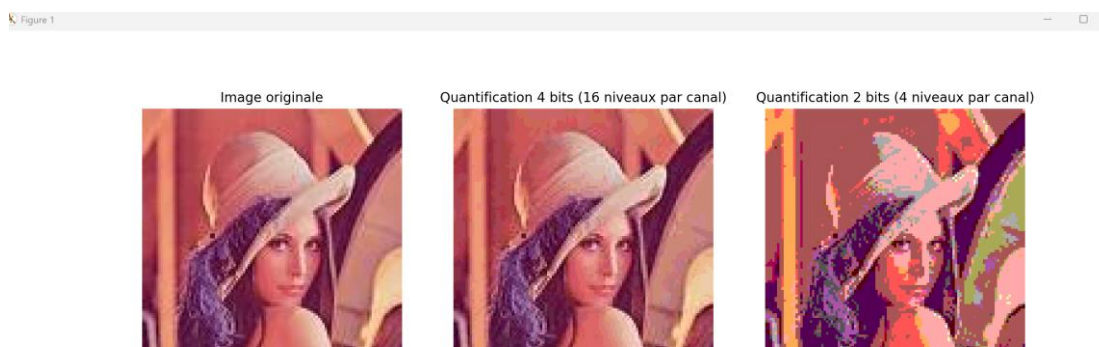
axes[1].imshow(image_4bits)
axes[1].set_title("Quantification 4 bits (16 niveaux par canal)")

axes[2].imshow(image_2bits)
axes[2].set_title("Quantification 2 bits (4 niveaux par canal)")

for ax in axes:
    ax.axis("off")

plt.show()
```

Résultat :



2. Comment la perte de quantification affecte la perception des couleurs?

Analyse des changements visuels



- **En 4 bits (16 niveaux par canal) :** Les couleurs restent assez fidèles, mais les transitions sont plus brutales.
- **En 2 bits (4 niveaux par canal) :** Forte réduction du nombre de couleurs (effet cartoon).et La perception des détails se détériore fortement

Conclusion :

- **4 bits garde encore une bonne fidélité visuelle, mais simplifie l'image.**
- **2 bits entraîne une forte perte de détails.**