



ENSA TETOUAN | 2024/2025

RAPPORT **TRAITEMENT** **D'IMAGES**

Encadré par:

Prof. Abdellatif Medouri

Réalisé par :

Barhrouj Saad

Ait Jaakike Mohamed Amine

Remerciements

Nous tenons tout d'abord à remercier le Professeur **M. Abdellatif Medouri** pour ses cours enrichissants en traitement d'images dans le cadre du module de vision artificielle. Ses enseignements et les ressources qu'il a mises à notre disposition ont été indispensables au bon déroulement de notre projet sur les images avec Python.

Nous souhaitons également exprimer notre gratitude envers nos camarades de classe pour leur collaboration et leur soutien tout au long de cette expérience d'apprentissage.

Enfin, nous remercions chaleureusement les membres de notre équipe, dont l'implication, qu'elle soit directe ou indirecte, a grandement contribué à la réussite de ce projet. Leur soutien, leurs idées et leur persévérance ont été des atouts majeurs pour mener à bien cette expérience.

Table des matières

➤ Introduction	1
➤ Méthodologie.....	2
➤ Développement et implémentation du projet	3
➤ Opérations d'images binaires et manipulation des Niveaux de gris	3
✓ Calcul de l'Image I(ad) (Addition)	6
✓ Calcul de l'Image I(s) (Soustraction)	8
✓ Calcul de l'Image I(p) (Multiplication par 2)	11
➤ Création de l'Histogramme d'une Image Couleur	14
➤ Conversion d'une Image en Niveaux de Gris ou Binaire	20
➤ Génération de l'Histogramme des Niveaux de Gris d'une Image	26
➤ Application du Filtre de Nagao sur une Image en Niveaux de Gris ...	32
➤ Conclusion	37

Introduction

Dans le cadre de notre quatrième année en tant qu'étudiants en Génie Informatique à l'ENSA de Tétouan, nous avons l'opportunité d'appliquer les connaissances acquises au cours du cours de Traitement d'Images, dans le cadre du module de Vision Artificielle. Ce projet vise à développer nos compétences en programmation Python en manipulant différentes images.

Nous réaliserons plusieurs opérations sur des images binaires et en niveaux de gris, tout en explorant les concepts fondamentaux du traitement d'images. Nous travaillerons avec des matrices numériques pour appliquer diverses techniques, telles que la multiplication d'images et la création d'histogrammes, afin d'obtenir des résultats visuels significatifs.

Ce rapport a pour objectif de présenter le travail effectué au cours de ces travaux pratiques, en documentant les étapes réalisées et en incluant des captures d'écran illustrant les résultats obtenus. Les différentes sections du rapport aborderont les tâches spécifiques réalisées, notamment la création d'images binaires, le calcul d'histogrammes et l'application de filtres sur des images en niveaux de gris.

Méthodologie

Le projet a été réalisé en étroite collaboration entre les membres de l'équipe, composée de **Saad Barhrouj** et **Mohamed Amine Ait Jaakike**.

Nous avons adopté une méthodologie de travail agile, répartissant les tâches de manière équilibrée et complémentaire afin d'assurer une progression harmonieuse et cohérente.

L'utilisation de GitHub a joué un rôle central dans notre collaboration. Cet outil nous a permis de versionner notre code, de partager nos développements en temps réel, et de gérer efficacement les contributions de chaque membre. Chaque étape du développement a été documentée, garantissant ainsi une transparence totale et facilitant les revues de code en équipe. Cela a également permis de maintenir une trace précise de l'évolution du projet, d'assurer une meilleure traçabilité des modifications, et d'éviter tout conflit dans les contributions.

Chacun des membres a pris en charge des tâches spécifiques, **réparties équitablement entre les deux membres du groupe**, tout en restant attentif à l'avancement global du projet. Cette approche collaborative a permis d'assurer une cohésion de groupe solide et d'atteindre nos objectifs dans les délais impartis. L'évaluation finale du projet portera non seulement sur les résultats obtenus mais aussi sur la qualité de notre travail en équipe, caractérisée par une communication continue, une entraide mutuelle, et une rigueur dans le développement et la validation des solutions proposées.

Développement et Implémentation Du projet

1. Opérations d'images binaires et manipulation des Niveaux de gris

i. Création des Matrices en Texte

Dans l'éditeur de texte **Visual Studio Code**, nous avons d'abord défini les matrices I1 et I2 en utilisant le format PBM, spécifiquement avec le type P1, adapté pour des images en noir et blanc (binaire). Voici les étapes suivies :

Format et Taille : Les matrices ont été configurées pour une taille de 9x9.
Structure des Données : Dans le fichier, nous avons représenté les pixels sous forme de valeurs binaires (0 et 1), organisés en une grille de 9x9 pour chaque image. Ces valeurs constituent les images I1 et I2.

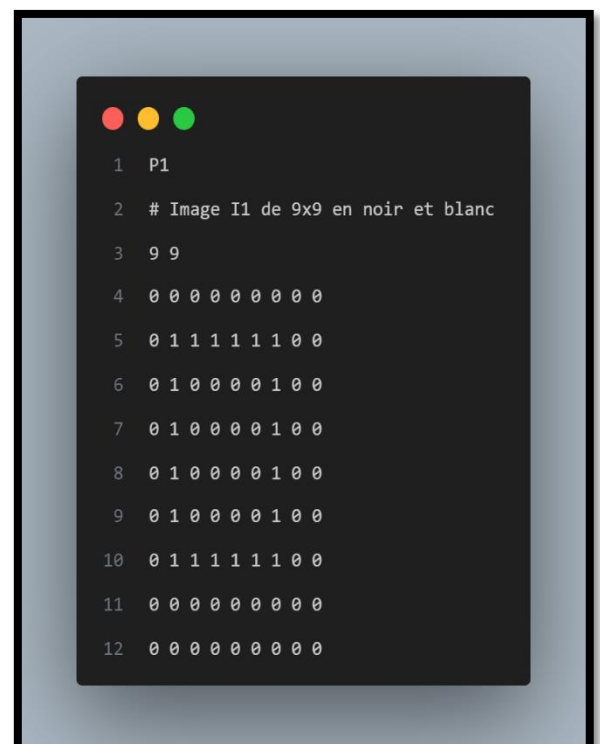
ii. Conversion en Format BMP

Pour permettre l'affichage des matrices comme des images :
Nous avons sauvegardé chaque fichier en changeant son extension en « .bmp » (BMP signifie Bitmap Image File), un format couramment utilisé pour stocker des images numériques.

La matrice I2 :

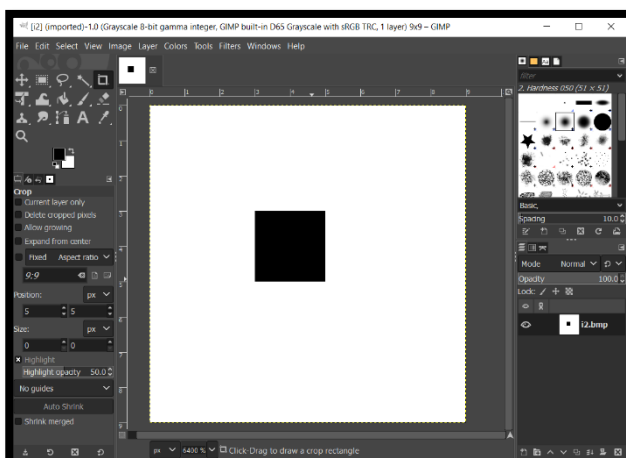


La matrice I1 :

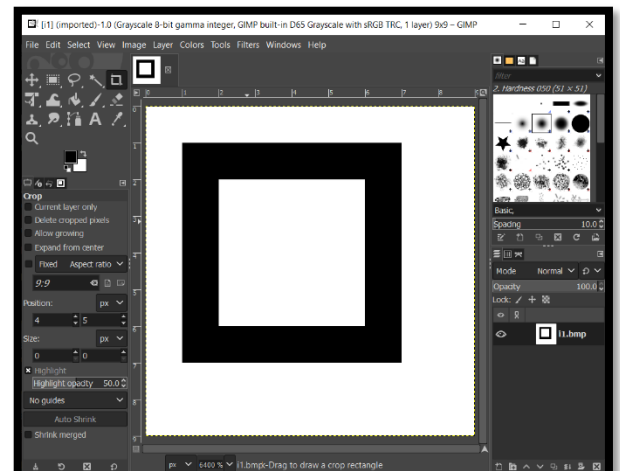


Ensuite, nous avons utilisé **GIMP** (GNU Image Manipulation Program), un logiciel de traitement d'image, pour lire les fichiers **BMP**, permettant ainsi de visualiser les images et de vérifier la structure binaire des matrices

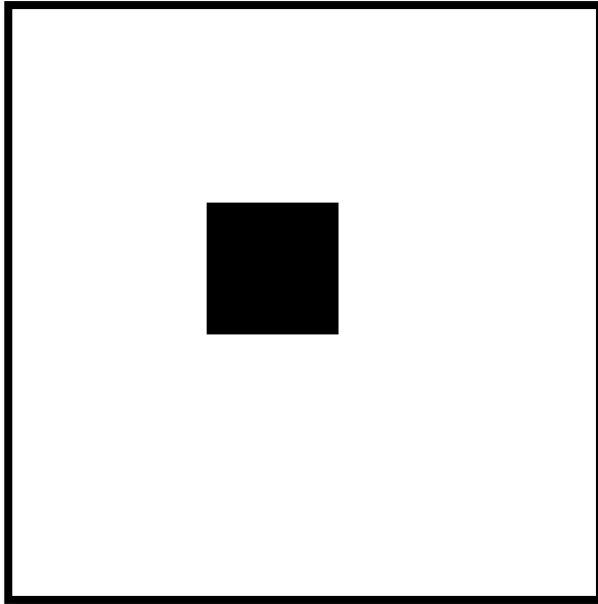
La matrice I2 :



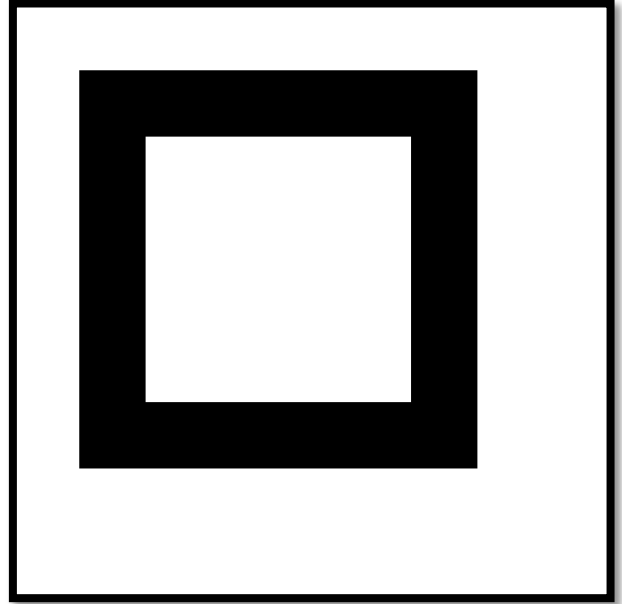
La matrice I1 :



La matrice I2 :



La matrice I1 :



iii. Bibliothèques nécessaires

Pour exécuter les différents scripts **Python** fournis (addition, soustraction, multiplication, lecture et écriture de fichiers BMP/PGM, etc.), la bibliothèque suivante est nécessaire :

- ❖ **NumPy** : Utilisée pour les opérations sur les matrices, telles que l'addition, la soustraction, la multiplication, et pour manipuler les données des images.

a) Calcul de l'Image I(ad) (Addition)

Méthode :

Nous avons utilisé ce programme Python pour effectuer l'addition des matrices I1 et I2. Pour chaque pixel des matrices, la valeur résultante a été calculée selon la relation suivante :

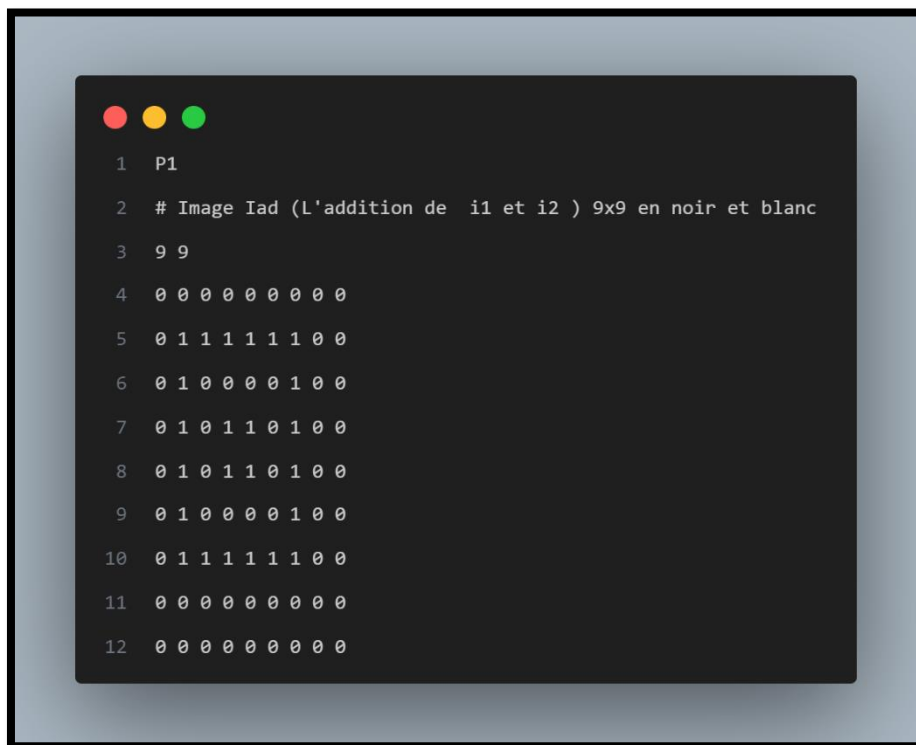
$$A(i, j) = \min(I_1(i, j) + I_2(i, j), 1)$$

```
1 import numpy as np
2
3 # Fonction pour lire la matrice depuis un fichier BMP
4 def lire_matrice_bmp(fichier):
5     with open(fichier, 'r') as f:
6         # Ignorer la ligne 'P1' et toute ligne de commentaire
7         while True:
8             ligne = f.readline().strip()
9             if ligne == 'P1':
10                 break
11             if ligne.startswith('#'): # Ignorer les commentaires
12                 continue
13
14         # Lire la taille de la matrice
15         dimensions = f.readline().strip().split()
16         lignes = int(dimensions[0])
17         colonnes = int(dimensions[1])
18
19         # Lire les éléments de la matrice
20         matrice = [list(map(int, ligne.strip().split())) for ligne in f.readlines()]
21
22     return np.array(matrice)
23
24 # Fonction pour écrire la matrice dans un fichier BMP
25 def ecrire_matrice_bmp(fichier, matrice):
26     with open(fichier, 'w') as f:
27         f.write('P1\n')
28         f.write(f"{matrice.shape[0]} {matrice.shape[1]}\n")
29         for ligne in matrice:
30             f.write(' '.join(map(str, ligne)) + '\n')
31
32 # Exemple d'utilisation
33 matrice1 = lire_matrice_bmp('i1.bmp')
34 matrice2 = lire_matrice_bmp('i2.bmp')
35
36 # Limite maximale de valeur (par exemple 15 pour des images 4 bits)
37 max_val = 15
38
39 # Effectuer l'addition des matrices et limiter les valeurs à max_val
40 resultat_addition = np.minimum(matrice1 + matrice2, max_val)
41
42 # Écrire le résultat dans un fichier BMP
43 ecrire_matrice_bmp('resultat_addition.bmp', resultat_addition)
44
```

Cela permet de garantir que les pixels restent en binaire, avec une valeur maximale de 1 (pour des images binaires). L'addition a été effectuée pixel par pixel, et pour chaque position, la somme des valeurs de I1 et I2 a été comparée à 1, la valeur finale étant la plus petite des deux.

Génération du fichier :

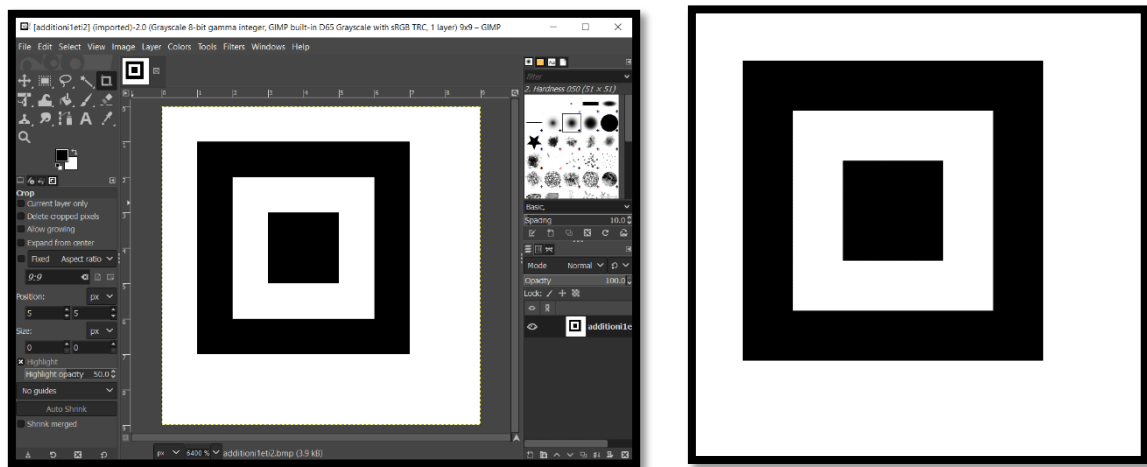
Le fichier BMP résultant a été généré directement par le programme Python, sans nécessiter de conversion intermédiaire en format texte. La matrice résultante a été enregistrée dans le format P1 (binaire), tout comme les images initiales I1 et I2. Le fichier BMP final contient ainsi l'image résultante de l'addition des deux matrices, en respectant la structure binaire définie par le format P1.



```
1 P1
2 # Image Iad (l\'addition de i1 et i2 ) 9x9 en noir et blanc
3 9 9
4 0 0 0 0 0 0 0 0 0
5 0 1 1 1 1 1 1 0 0
6 0 1 0 0 0 0 1 0 0
7 0 1 0 1 1 0 1 0 0
8 0 1 0 1 1 0 1 0 0
9 0 1 0 0 0 0 1 0 0
10 0 1 1 1 1 1 1 0 0
11 0 0 0 0 0 0 0 0 0
12 0 0 0 0 0 0 0 0 0
```

Vérification :

Le fichier a été ouvert avec GIMP pour observer si le résultat visuel correspondait à l'addition mathématique des pixels. Après avoir observé l'image résultante, il a été constaté que les résultats visuels étaient similaires à ceux obtenus par le calcul mathématique, et l'image affichée correspondait parfaitement à la transformation effectuée, validant ainsi la précision du processus de calcul.



b) Calcul de l'Image I(s) (Soustraction)

Méthode :

Nous avons utilisé ce programme Python pour effectuer la soustraction des matrices I_1 et I_2 . Pour chaque pixel des matrices, la valeur résultante a été calculée selon la relation suivante :

$$S(i, j) = \max(I_1(i, j) - I_2(i, j), 0)$$

```
1 import numpy as np
2
3 # Fonction pour lire la matrice depuis un fichier BMP
4 def lire_matrice_bmp(fichier):
5     with open(fichier, 'r') as f:
6         # Ignorer la ligne 'P1' et toute ligne de commentaire
7         while True:
8             ligne = f.readline().strip()
9             if ligne == 'P1':
10                 break
11             if ligne.startswith('#'): # Ignorer les commentaires
12                 continue
13
14         # Lire la taille de la matrice
15         dimensions = f.readline().strip().split()
16         lignes = int(dimensions[0])
17         colonnes = int(dimensions[1])
18
19         # Lire les éléments de la matrice
20         matrice = [list(map(int, ligne.strip().split())) for ligne in f.readlines()]
21
22     return np.array(matrice)
23
24 # Fonction pour écrire la matrice dans un fichier BMP
25 def ecrire_matrice_bmp(fichier, matrice):
26     with open(fichier, 'w') as f:
27         f.write('P1\n')
28         f.write(f"{matrice.shape[0]} {matrice.shape[1]}\n")
29         for ligne in matrice:
30             f.write(' '.join(map(str, ligne)) + '\n')
31
32 # Exemple d'utilisation
33 matrice1 = lire_matrice_bmp('i1.bmp')
34 matrice2 = lire_matrice_bmp('i2.bmp')
35
36
37
38 resultat_soustraction = np.maximum(matrice1 - matrice2, 0) # Limiter à 0 si négatif
39 ecrire_matrice_bmp('resultat_soustraction.bmp', resultat_soustraction)
40
41
42
```

Cela permet de garantir que les pixels ne deviennent pas négatifs, en plaçant une limite inférieure à 0 (pour des images binaires). La soustraction a été effectuée pixel par pixel, et pour chaque position, la différence entre les valeurs de I1 et I2 a été calculée, avec un minimum de 0 appliqué pour éviter les valeurs négatives.

Génération du fichier :

Le fichier BMP résultant a été généré directement par le programme Python, sans nécessiter de conversion intermédiaire en format texte. La matrice résultante a été enregistrée dans le format P1 (binaire), tout comme les images initiales I1 et I2. Le fichier BMP final contient ainsi l'image résultante de la soustraction des deux matrices, en respectant la structure binaire définie par le format P1.

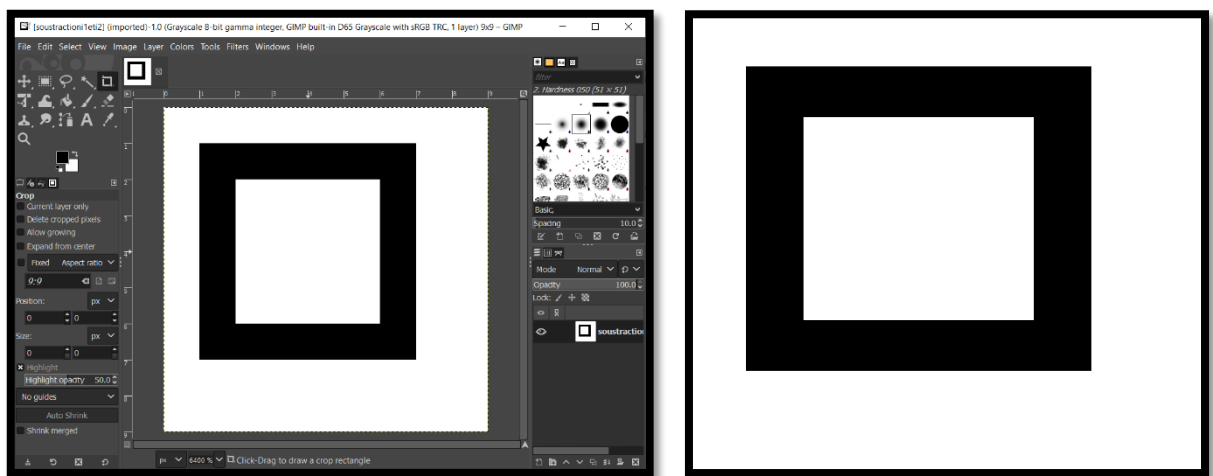
A screenshot of a terminal window with a dark background and light gray text. The terminal shows the content of a P1 format binary image file. The first line is '1 P1'. The second line is a comment: '2 # Image Is (La soustraction de i1 et i2) 9x9 en noir et blanc'. The third line is '3 9 9'. The following lines are a 9x9 matrix of 0s and 1s, representing the binary image data. The matrix is as follows:

```
4 0 0 0 0 0 0 0 0 0
5 0 1 1 1 1 1 1 0 0
6 0 1 0 0 0 0 1 0 0
7 0 1 0 0 0 0 1 0 0
8 0 1 0 0 0 0 1 0 0
9 0 1 0 0 0 0 1 0 0
10 0 1 1 1 1 1 1 0 0
11 0 0 0 0 0 0 0 0 0
12 0 0 0 0 0 0 0 0 0
```

Vérification :

Le fichier a été ouvert avec GIMP pour observer si le résultat visuel correspondait à la soustraction mathématique des pixels. Après avoir observé l'image résultante, il a été constaté que les résultats visuels étaient similaires à ceux obtenus par le calcul mathématique, et l'image affichée

correspondait parfaitement à la transformation effectuée, validant ainsi la précision du processus de calcul.



c) Calcul de l'Image I(p) (Multiplication par 2)

Méthode :

Nous avons utilisé ce programme Python pour effectuer la multiplication de chaque pixel de l'image d'origine, en suivant la formule suivante :

$$M(i, j) = \min(I(i, j) \times 2, L - 1)$$

```
1 import numpy as np
2
3 # Fonction pour lire la matrice depuis un fichier P2 (format PGM ASCII)
4 def lire_matrice_p2(fichier):
5     with open(fichier, 'r') as f:
6         # Lire l'en-tête P2 et ignorer les commentaires
7         while True:
8             ligne = f.readline().strip()
9             if ligne == 'P2':
10                 break
11             if ligne.startswith('#'): # Ignorer les commentaires
12                 continue
13
14         # Lire la taille de l'image (dimensions)
15         dimensions = f.readline().strip().split()
16         lignes = int(dimensions[0])
17         colonnes = int(dimensions[1])
18
19         # Lire la valeur maximale (souvent 255 pour des images en niveaux de gris)
20         max_val = int(f.readline().strip())
21
22         # Lire les éléments de la matrice (pixels)
23         matrice = [list(map(int, ligne.strip().split())) for ligne in f.readlines()]
24
25     return np.array(matrice), max_val
26
27 # Fonction pour écrire la matrice dans un fichier P2 (format PGM ASCII)
28 def ecrire_matrice_p2(fichier, matrice, max_val):
29     with open(fichier, 'w') as f:
30         f.write('P2\n')
31         f.write(f"{matrice.shape[1]} {matrice.shape[0]}\n") # dimensions : colonnes x lignes
32         f.write(f"{max_val}\n") # valeur maximale (par exemple 255)
33         for ligne in matrice:
34             f.write(' '.join(map(str, ligne)) + '\n')
35
36 # Exemple d'utilisation
37 matrice, max_val = lire_matrice_p2('i.bmp') # Remplacer 'i.bmp' par le nom de votre fichier image
38
39 # Valeur maximale pour limiter le résultat (par exemple, 15 pour une image 4 bits)
40 max_val = 15
41
42 # Effectuer la multiplication de la matrice par 2 (et limiter la valeur maximale à 15)
43 resultat_multiplication = np.minimum(matrice * 2, max_val) # Limiter à la valeur maximale (par exemple 15)
44
45 # Écrire le résultat dans un nouveau fichier PGM
46 ecrire_matrice_p2('resultat_multiplication.pgm', resultat_multiplication, max_val)
47
```

Avec $L-1=15$, cette formule multiplie chaque pixel de l'image par 2 tout en plafonnant les valeurs maximales à 15, garantissant que l'image reste dans la plage des niveaux de gris sur 4 bits (de 0 à 15). L'opération a été effectuée pixel par pixel sur la matrice de l'image.

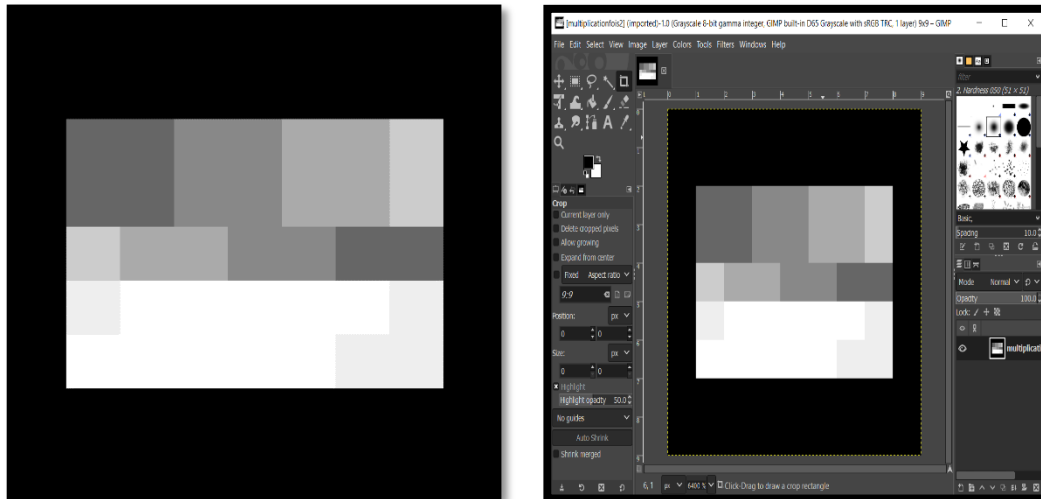
Génération du fichier :

Le fichier résultant a été généré directement par le programme Python. La matrice transformée a été enregistrée dans le format P2 (format PGM ASCII), où chaque pixel est exprimé sous forme numérique. Le fichier PGM final contient ainsi l'image résultante de la transformation, respectant la structure ASCII définie par le format P2.



```
1  P2
2  # image de multiplication fois 2 de matrice I
3  9 9
4  15
5  0 0 0 0 0 0 0 0 0
6  0 0 0 0 0 0 0 0 0
7  0 6 6 8 8 10 10 12 0
8  0 6 6 8 8 10 10 12 0
9  0 12 10 10 8 8 6 6 0
10 0 14 15 15 15 15 14 0
11 0 15 15 15 15 15 14 0
12 0 0 0 0 0 0 0 0
13 0 0 0 0 0 0 0 0
14
```


La matrice $I(P)$:



2. Création de l'Histogramme d'une Image Couleur

Pour cette tâche, nous avons choisi d'utiliser le langage de programmation **Python**, déjà utilisé dans la première question. Ce choix s'est imposé en raison de la simplicité et de l'efficacité de Python pour le traitement d'images, grâce aux bibliothèques spécifiques disponibles, telles que « OpenCV » et « Matplotlib ». Ces bibliothèques offrent des fonctions avancées de traitement d'image et de visualisation, permettant de générer des histogrammes de couleur en quelques lignes de code.

Installation des Bibliothèques :

Pour l'exécution de ce script, les bibliothèques **OpenCV** et **Matplotlib** ont été installées.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Med Amine Jaakike>pip install opencv-python matplotlib
Collecting opencv-python
  Downloading opencv_python-4.10.0.84-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting matplotlib
  Downloading matplotlib-3.9.2-cp312-cp312-win_amd64.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.21.2 in c:\python\python312\lib\site-packages (from opencv-python) (2.0.0)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.0-cp312-cp312-win_amd64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.54.1-cp312-cp312-win_amd64.whl.metadata (167 kB)
----- 167.0/167.0 kB 627.2 kB/s eta 0:00:00
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.7-cp312-cp312-win_amd64.whl.metadata (6.4 kB)
Requirement already satisfied: packaging>=20.0 in c:\python\python312\lib\site-packages (from matplotlib) (24.1)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-11.0.0-cp312-cp312-win_amd64.whl.metadata (9.3 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Downloading pyparsing-3.2.0-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in c:\python\python312\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\python\python312\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Downloading opencv_python-4.10.0.84-cp37-abi3-win_amd64.whl (38.8 MB)
----- 38.8/38.8 MB 1.6 MB/s eta 0:00:00
Downloading matplotlib-3.9.2-cp312-cp312-win_amd64.whl (7.8 MB)
----- 7.8/7.8 MB 2.3 MB/s eta 0:00:00
Downloading contourpy-1.3.0-cp312-cp312-win_amd64.whl (218 kB)
----- 218.3/218.3 kB 13.9 MB/s eta 0:00:00
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
----- 2.2/2.2 MB 3.3 MB/s eta 0:00:00
Downloading fonttools-4.54.1-cp312-cp312-win_amd64.whl (2.2 MB)
----- 55.9/55.9 kB 3.0 MB/s eta 0:00:00
Downloading kiwisolver-1.4.7-cp312-cp312-win_amd64.whl (55 kB)
----- 2.6/2.6 MB 2.3 MB/s eta 0:00:00
Downloading pillow-11.0.0-cp312-cp312-win_amd64.whl (2.6 MB)
----- 106.9/106.9 kB 3.0 MB/s eta 0:00:00
Installing collected packages: pyparsing, pillow, opencv-python, kiwisolver, fonttools, cycler, contourpy, matplotlib
Successfully installed contourpy-1.3.0 cycler-0.12.1 fonttools-4.54.1 kiwisolver-1.4.7 matplotlib-3.9.2 opencv-python-4.10.0.84 pillow-11.0.0 pyparsing-3.2.0

[notice] A new release of pip is available: 24.1.2 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Code Utilisé

Le code Python ci-dessous a été utilisé pour générer l'histogramme des trois canaux de couleurs (Bleu, Vert et Rouge) d'une image couleur :

```
1 import cv2 as cv
2 from matplotlib import pyplot as plt
3 from os import path
4
5 # Demander à l'utilisateur de saisir le chemin d'accès complet à l'image
6 print("Saisissez le chemin d'accès complet à l'image (par exemple, (/chemin/vers/image.jpg) : ")
7 imagePath = input()
8
9 # Vérifier si le fichier existe
10 if not path.exists(imagePath):
11     print("Image non trouvée !")
12     exit()
13
14 # Charger l'image
15 img = cv.imread(imagePath)
16
17 # Si l'image n'a pas été chargée correctement, sortir du programme
18 if img is None:
19     print("Échec du chargement de l'image.")
20     exit()
21
22 # Séparer les canaux de l'image (Bleu, Vert, Rouge)
23 b, g, r = cv.split(img)
24
25 # Afficher l'image dans une fenêtre
26 cv.imshow("Image", img)
27
28 # Tracer l'histogramme des canaux Bleu, Vert et Rouge
29 plt.figure()
30 plt.title("Histogramme des canaux de couleurs")
31 plt.xlabel("Valeur des pixels")
32 plt.ylabel("Fréquence")
33
34 # Tracer les histogrammes des canaux avec des couleurs correspondantes
35 plt.hist(b.ravel(), 256, [0, 256], color='blue', label='Canal Bleu')
36 plt.hist(g.ravel(), 256, [0, 256], color='green', label='Canal Vert')
37 plt.hist(r.ravel(), 256, [0, 256], color='red', label='Canal Rouge')
38
39 # Ajouter une légende pour chaque canal
40 plt.legend()
41
42 # Afficher le graphique
43 plt.show()
44
45 # Attendre une touche pour fermer la fenêtre d'affichage de l'image
46 cv.waitKey(0)
47 cv.destroyAllWindows()
48
```

Explication du Code

Chargement de l'image :

Le programme commence par demander à l'utilisateur de saisir le chemin d'accès complet à l'image. Cela permet à l'utilisateur de sélectionner une image située n'importe où sur le système de fichiers, et non uniquement dans le même dossier que le script. Le programme vérifie ensuite si le fichier existe à l'emplacement spécifié. S'il n'existe pas, un message d'erreur s'affiche et le programme se termine.

Ensuite, la fonction `path.exists(imageName)` vérifie l'existence du fichier. Si l'image n'est pas trouvée, le programme affiche un message d'erreur et se termine.

Une fois le fichier confirmé, `cv.imread(imageName)` charge l'image, et si le chargement échoue, un autre message d'erreur est affiché, et le programme se termine.

Séparation des Canaux de Couleurs :

Avec `cv.split(img)`, l'image est décomposée en trois canaux de couleur : Bleu (B), Vert (G), et Rouge (R). Ces canaux contiennent les valeurs d'intensité de chaque couleur pour chaque pixel, permettant de traiter chaque couleur indépendamment.

Cette séparation est essentielle pour pouvoir analyser chaque canal de couleur individuellement dans l'histogramme.

Affichage de l'Image :

La commande `cv.imshow("Image", img)` ouvre une fenêtre pour afficher l'image chargée, offrant une visualisation de celle-ci avant le traitement d'histogramme.

Cette étape permet également de vérifier visuellement que l'image a bien été chargée.

Génération de l'Histogramme :

En utilisant Matplotlib, le programme trace un histogramme pour chaque canal de couleur. `plt.hist(b.ravel(), 256, [0, 256], color='blue', label='Canal Bleu')` et les commandes similaires pour les canaux vert et rouge produisent les histogrammes en utilisant les valeurs de chaque canal. La fonction `ravel()` permet de "détendre" les matrices de couleur en un tableau linéaire de valeurs, ce qui facilite le comptage des occurrences de chaque valeur de pixel.

Les paramètres 256 et `[0, 256]` définissent la plage des valeurs de pixels de 0 à 255 (typique pour une image 8 bits par canal), et `color` applique une couleur correspondant à chaque canal (bleu, vert, rouge).

Enfin, `plt.legend()` ajoute une légende pour chaque couleur, clarifiant l'information.

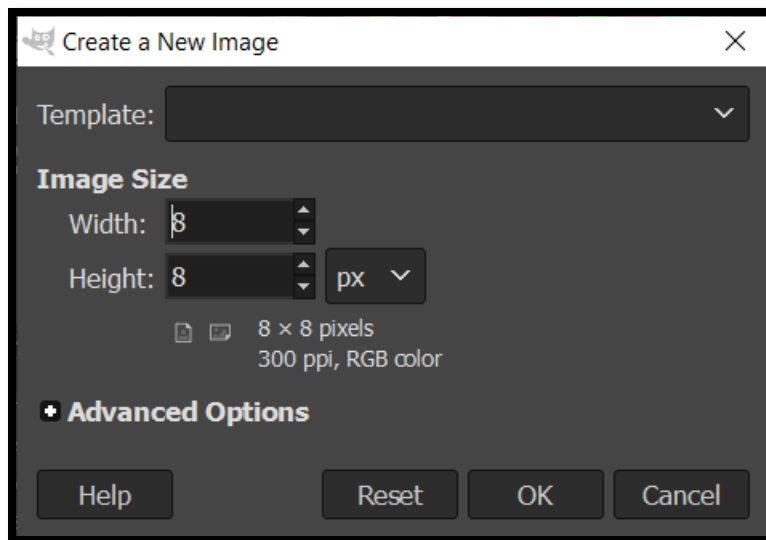
Affichage du Résultat :

`plt.show()` affiche le graphique de l'histogramme final dans une nouvelle fenêtre, permettant une visualisation des distributions de couleur.

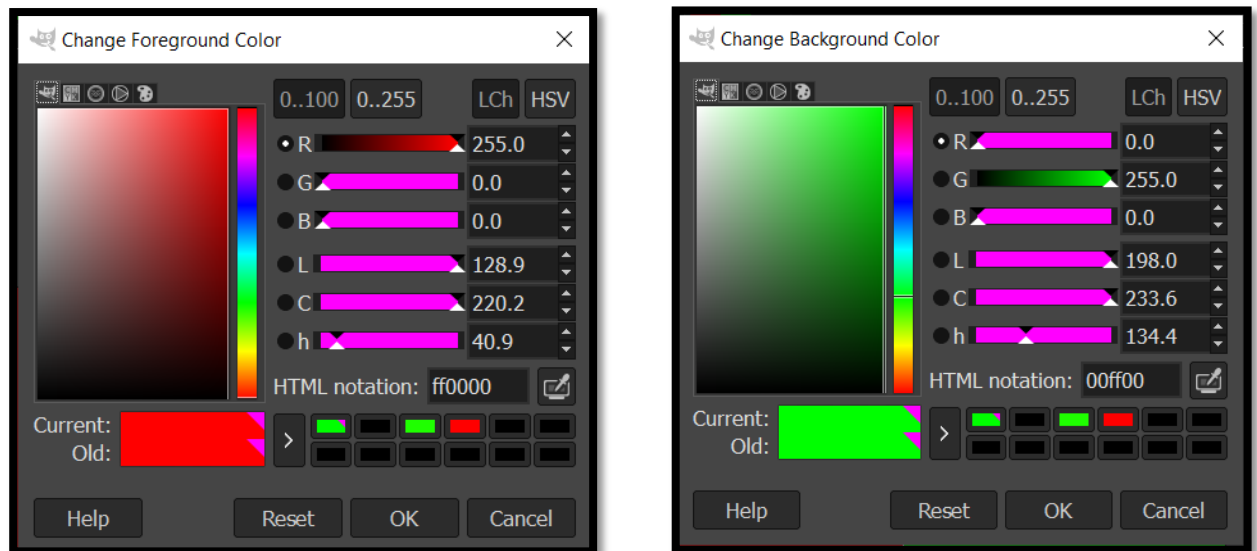
`cv.waitKey(0)` et `cv.destroyAllWindows()` permettent d'attendre une action de l'utilisateur pour fermer la fenêtre d'image et toute autre fenêtre ouverte.

Vérification:

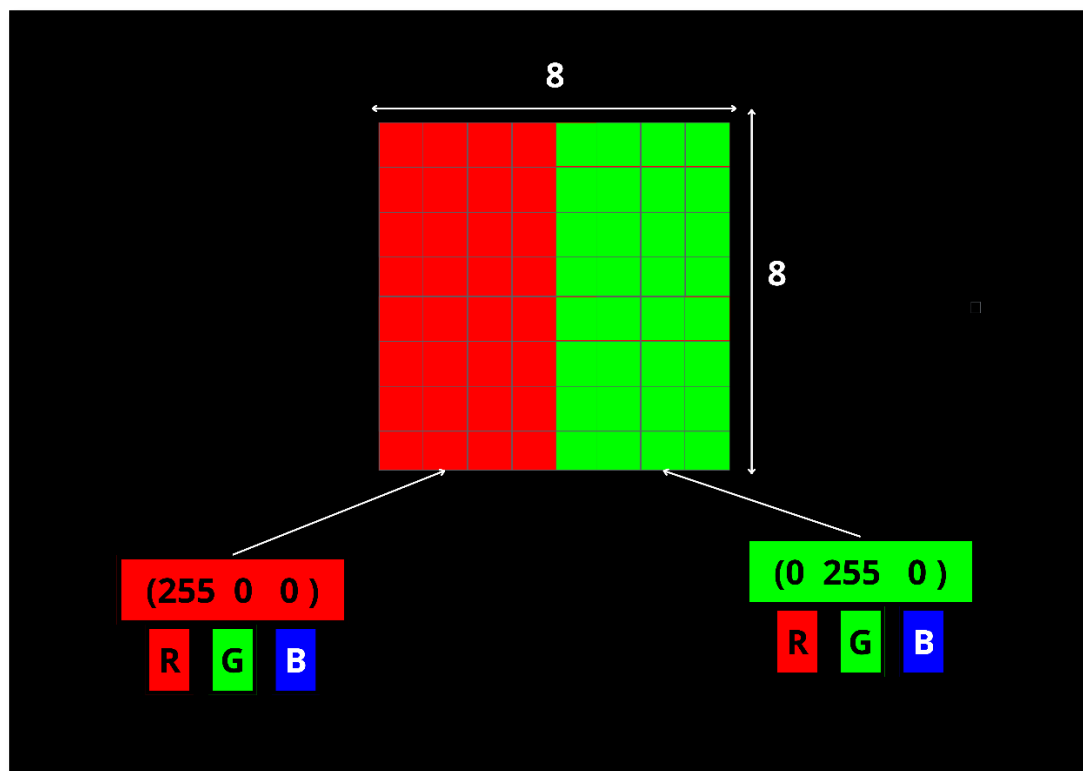
Pour vérifier que le code fonctionne correctement, nous avons créé une image dans GIMP avec une taille de 8x8 pixels :



L'image a été divisée en deux parties égales : les 32 premiers pixels sont rouges avec la valeur RGB (255, 0, 0), et les 32 pixels suivants sont verts avec la valeur RGB (0, 255, 0)



Voici un schéma qui illustre cette répartition :



La matrice correspondante de l'image est également présentée ci-dessous :

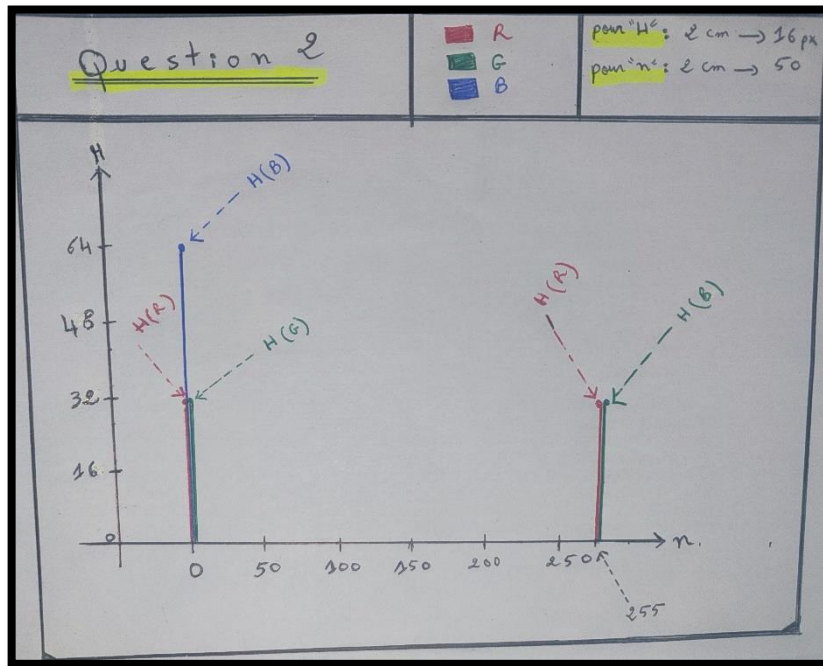
(255 0 0)	(255 0 0)	(255 0 0)	(255 0 0)	(0 255 0)	(0 255 0)	(0 255 0)	(0 255 0)
(255 0 0)	(255 0 0)	(255 0 0)	(255 0 0)	(0 255 0)	(0 255 0)	(0 255 0)	(0 255 0)
(255 0 0)	(255 0 0)	(255 0 0)	(255 0 0)	(0 255 0)	(0 255 0)	(0 255 0)	(0 255 0)
(255 0 0)	(255 0 0)	(255 0 0)	(255 0 0)	(0 255 0)	(0 255 0)	(0 255 0)	(0 255 0)
(255 0 0)	(255 0 0)	(255 0 0)	(255 0 0)	(0 255 0)	(0 255 0)	(0 255 0)	(0 255 0)
(255 0 0)	(255 0 0)	(255 0 0)	(255 0 0)	(0 255 0)	(0 255 0)	(0 255 0)	(0 255 0)
(255 0 0)	(255 0 0)	(255 0 0)	(255 0 0)	(0 255 0)	(0 255 0)	(0 255 0)	(0 255 0)
(255 0 0)	(255 0 0)	(255 0 0)	(255 0 0)	(0 255 0)	(0 255 0)	(0 255 0)	(0 255 0)

Ensuite, nous avons calculé manuellement le tableau de l'histogramme pour chaque composante de couleur RGB par ligne : $H_r(n)$, $H_g(n)$, et $H_b(n)$

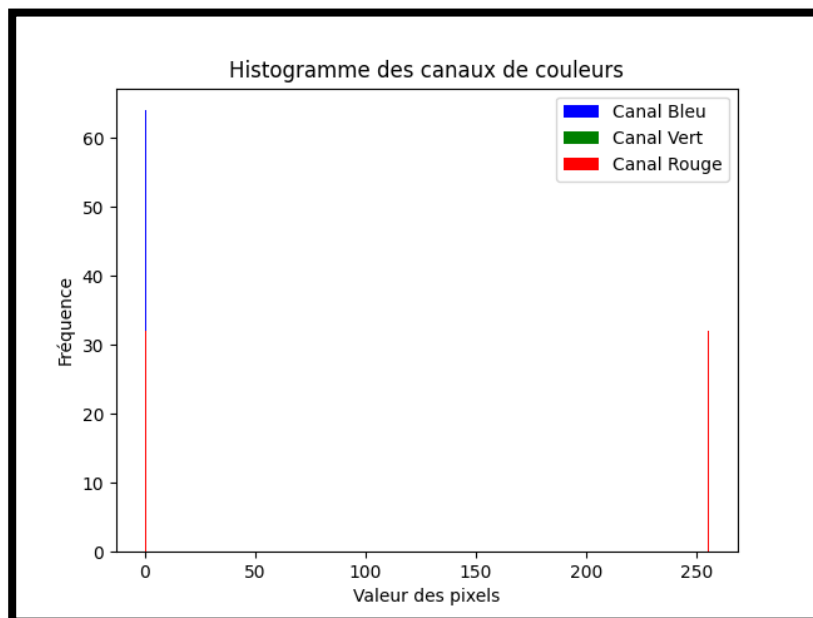
Le tableau obtenu est le suivant :

n	0	255
$H_r(n)$	32	32
$H_g(n)$	32	32
$H_b(n)$	64	0

Nous avons également dessiné manuellement l'histogramme des valeurs RGB, en nous basant sur les résultats du tableau :

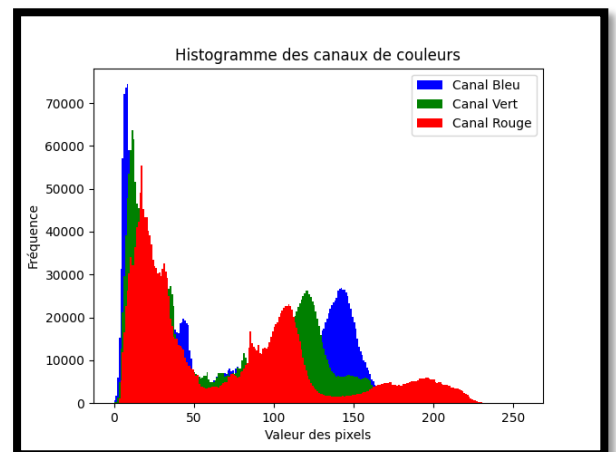


Nous avons ensuite soumis cette même image à notre code pour vérifier son exactitude. L'histogramme généré par le code est présenté ci-dessous :

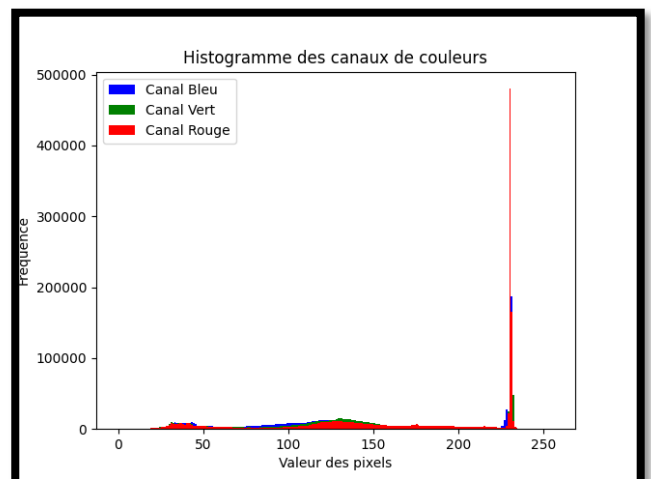


En comparant les résultats, nous constatons que le code fonctionne correctement et produit des résultats parfaits.

Exécution du Code avec l'Image 1 (notre portrait)



Exécution du Code avec l'Image 2 (notre portrait)



3. Conversion d'une Image en Niveaux de Gris ou Binaire

Dans cette tâche, nous avons écrit ce programme en Python permettant de convertir une image en niveaux de gris ou en binaire. Le programme offre à l'utilisateur la possibilité de choisir entre une image binaire (noir et blanc) ou une image en niveaux de gris. Si l'image est convertie en niveaux de gris, l'utilisateur peut aussi définir la profondeur de couleur en spécifiant le nombre de bits (de 1 à 8 bits). Ce programme est basé sur la bibliothèque Pillow, qui est une bibliothèque populaire pour la manipulation d'images en Python.

Installation des Bibliothèques

Le programme nécessite l'installation de la bibliothèque Pillow, qui est utilisée pour ouvrir, manipuler et sauvegarder les images.

```
C:\Users\Med Amine Jaakike>pip install Pillow
Requirement already satisfied: Pillow in c:\python\python312\lib\site-packages (11.0.0)

[notice] A new release of pip is available: 24.1.2 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Code Utilisé

Le code suivant permet de réaliser la conversion de l'image en niveaux de gris ou en binaire, selon le choix de l'utilisateur. Si le mode "niveaux de gris" est choisi, l'utilisateur peut spécifier le nombre de bits entre 1 et 8 pour définir la profondeur des niveaux de gris :

```
1  from PIL import Image
2  import numpy as np
3  from os import path
4
5  def convert_image(image_path, output_path, mode, bits=None):
6      # Vérifier si l'image existe
7      if not path.isfile(image_path):
8          print("Le chemin de l'image est invalide ou le fichier n'existe pas.")
9          return
10
11     # Ouvrir l'image
12     img = Image.open(image_path)
13
14     if mode == 'binaire':
15         # Convertir en image binaire
16         img = img.convert('1') # Mode '1' pour binaire
17     elif mode == 'niveau_de_gris':
18         # Convertir en niveaux de gris
19         img = img.convert('L') # Mode 'L' pour niveaux de gris
20
21         if bits is not None:
22             # Appliquer le nombre de bits
23             max_val = 2**bits - 1
24             img = img.point(lambda x: (x // (256 // (max_val + 1))) * (256 // (max_val + 1)))
25
26     # Sauvegarder l'image convertie
27     img.save(output_path)
28     print(f"L'image a été enregistrée sous : {output_path}")
29
30 def main():
31     image_path = input("Entrez le chemin complet de l'image à convertir (par exemple, /chemin/vers/image.jpg) : ")
32     output_path = input("Entrez le chemin complet de sortie de l'image convertie (par exemple, /chemin/vers/image_convertie.jpg) : ")
33     mode = input("Choisissez le mode (binaire/niveau_de_gris) : ").strip().lower()
34
35     if mode == 'niveau_de_gris':
36         bits = int(input("Entrez le nombre de bits (1-8) pour l'image en niveaux de gris : "))
37         if bits < 1 or bits > 8:
38             print("Le nombre de bits doit être entre 1 et 8.")
39             return
40     else:
41         bits = None
42
43     convert_image(image_path, output_path, mode, bits)
44
45 if __name__ == "__main__":
46     main()
47
```

Explication du Code

Ouverture de l'Image :

Le programme commence par ouvrir l'image spécifiée par l'utilisateur à l'aide de la fonction `Image.open()`. Cette fonction permet de lire l'image et de la charger dans un objet `Image` de la bibliothèque `Pillow`. Cela permet d'effectuer des transformations directement sur l'image.

Choix du Mode de Conversion :

Mode Binaire : Si l'utilisateur choisit le mode 'binaire', l'image est convertie en noir et blanc uniquement. Chaque pixel de l'image est soit complètement noir (valeur 0), soit complètement blanc (valeur 255). Cette conversion est effectuée avec la méthode `img.convert('1')`, où '1' est le mode binaire de `Pillow`, qui ne gère que les pixels blancs et noirs.

Mode Niveaux de Gris : Si l'utilisateur choisit le mode 'niveau_de_gris', l'image est convertie en une échelle de gris, où chaque pixel prend une valeur entre 0 (noir) et 255 (blanc), avec des niveaux intermédiaires pour représenter différentes intensités de gris. Cela se fait via `img.convert('L')`, où 'L' signifie "Luminance" et représente l'intensité lumineuse en niveaux de gris.

Application des Niveaux de Gris avec un Nombre de Bits Défini : Lorsque l'utilisateur choisit de travailler en niveaux de gris, il peut spécifier un nombre de bits. Le nombre de bits détermine la gamme de gris possible. Par exemple :

1 bit : Seulement deux niveaux de gris (noir et blanc).

8 bits : 256 niveaux de gris possibles, ce qui permet une transition plus fluide entre les couleurs.

Le code utilise la fonction `img.point()` pour ajuster les valeurs de pixels en fonction du nombre de bits. Il divise les pixels de l'image selon le nombre de niveaux de gris spécifiés et les réajuste pour qu'ils tombent dans cette gamme.

Sauvegarde de l'Image Convertie : Après avoir effectué la conversion, l'image est enregistrée à l'emplacement spécifié par l'utilisateur. Le programme utilise `img.save(output_path)` pour cela. L'image convertie est alors prête à être visualisée ou utilisée selon les besoins.

Interaction avec l'Utilisateur : Le programme interagit avec l'utilisateur via la fonction `main()`, où l'utilisateur est invité à fournir :

Le chemin de l'image à convertir.

Le mode de conversion (binaire ou niveaux de gris).

Si le mode est 'niveaux de gris', le programme demande également le nombre de bits (entre 1 et 8).

Vérification du Résultat

Après avoir exécuté ce programme avec une image de test, vous pouvez vérifier que la conversion a bien été réalisée en fonction des choix effectués. Par exemple, pour un mode binaire, l'image doit être réduite à seulement deux couleurs (noir et blanc). Pour un mode niveaux de gris, l'image aura une gamme de couleurs allant de 0 (noir) à 255 (blanc), avec une résolution définie par le nombre de bits que vous avez choisi.

Image2 en binaire

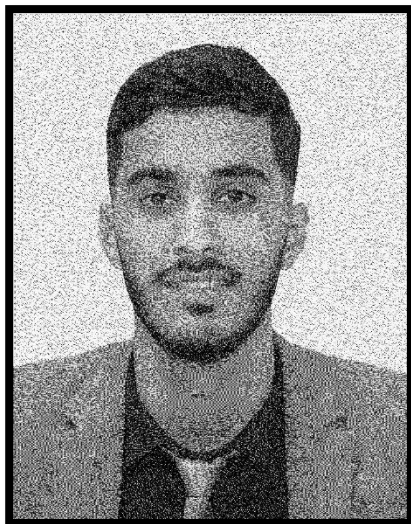


Image1 en binaire



Image en niveaux de gris codée sur 1 bits



Image en niveaux de gris codée sur 2 bits



Image en niveaux de gris codée sur 4 bits



Image en niveaux de gris codée sur 8 bits



4. Programme d'Ouverture et de Conversion d'Image en Matrice

Nous avons écrit un programme qui permet d'ouvrir une image, de la convertir en matrice et de l'enregistrer sous un format texte spécifique.

Nous avons choisi d'utiliser le même langage de programmation pour cette tâche en raison de ses puissantes bibliothèques de traitement d'images telles que Pillow et Matplotlib, qui facilitent la manipulation des images et la génération de matrice sous forme de fichier texte . NumPy est utilisé pour le traitement des données

Sous forme de matrices, ce qui permet d'effectuer des calculs rapides et efficaces sur les pixels de l'image.

Installation des Bibliothèques

```
C:\Users\Med Amine Jaakike>pip install numpy
Requirement already satisfied: numpy in c:\python\python312\lib\site-packages (2.0.0)

[notice] A new release of pip is available: 24.1.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Code Utilisé

Voici le code Python utilisé pour générer l'histogramme des niveaux de gris de l'image :

```
1 from PIL import Image
2 import numpy as np
3 from os import path
4
5 # Demander à l'utilisateur de saisir le chemin complet de l'image
6 print("Entrez le chemin d'accès complet de l'image (par exemple, /chemin/vers/image.jpg) : ")
7 nom_image = input()
8
9 # Vérifier si le fichier existe
10 if not path.exists(nom_image):
11     print("Image introuvable !")
12     exit()
13
14 # Ouvrir l'image
15 image = Image.open(nom_image)
16
17 # Vérifier si l'image est en mode RGB (couleur)
18 if image.mode != 'RGB':
19     image = image.convert('RGB')
20
21 # Récupérer les dimensions de l'image
22 largeur, hauteur = image.size
23
24 # Convertir l'image en une matrice NumPy
25 pixels = np.array(image)
26
27 # Déterminer la valeur maximale des pixels en fonction du type d'encodage
28 if pixels.dtype == np.uint8:
29     valeur_max = 255 # Codage sur 8 bits
30 elif pixels.dtype == np.uint16:
31     valeur_max = 65535 # Codage sur 16 bits
32 else:
33     valeur_max = pixels.max() # Autre codage (rare), on prend la valeur maximale trouvée
34
35 # Définir le chemin du fichier texte dans le même répertoire que l'image
36 repertoire_image = path.dirname(nom_image)
37 fichier_sortie = path.join(repertoire_image, "matrice_image.txt")
38
39 # Générer un fichier texte contenant le format PNM
40 with open(fichier_sortie, "w") as fichier:
41     # Écrire l'en-tête P3
42     fichier.write("P3\n")
43     # Écrire les dimensions et la valeur maximale
44     fichier.write(f"{largeur} {hauteur}\n{valeur_max}\n")
45
46     # Écrire la matrice sous forme de triplets [R, G, B]
47     for i in range(hauteur):
48         ligne = " ".join([f"{pixels[i][j][0]}, {pixels[i][j][1]}, {pixels[i][j][2]}"
49                             for j in range(largeur)])
50         fichier.write(ligne + "\n")
51
52 # Confirmer à l'utilisateur que le fichier a été généré
53 print(f"Fichier généré avec succès : {fichier_sortie}")
54 print(f"Dimensions : {largeur}x{hauteur}")
55 print(f"Valeur maximale des pixels : {valeur_max}")
56
```

Voici comment cela fonctionne :

Demande de l'image :

Le programme commence par demander à l'utilisateur de saisir le chemin d'accès complet de l'image qu'il souhaite analyser. Cela permet de charger l'image à partir de n'importe quel emplacement sur le système.

Vérification de l'existence de l'image :

Une fois le chemin saisi, le programme vérifie si le fichier spécifié existe réellement. Si l'image n'est pas trouvée, un message d'erreur est affiché, et le programme s'arrête.

Ouverture de l'image :

L'image spécifiée est ouverte à l'aide de la fonction `Image.open()` de la bibliothèque PIL (Python Imaging Library). Si l'image n'est pas déjà en mode RGB (format standard avec trois couleurs : rouge, vert et bleu), elle est convertie en RGB grâce à la méthode `convert('RGB')`.

Récupération des dimensions de l'image :

Une fois l'image ouverte et convertie, la méthode `size` permet de récupérer ses dimensions, c'est-à-dire la largeur et la hauteur de l'image. Ces dimensions sont nécessaires pour créer la matrice de l'image, où chaque

pixel sera représenté par un triplet de valeurs (R, G, B).

Conversion de l'image en matrice :

L'image est convertie en une matrice NumPy avec la fonction `np.array()`, où chaque élément de la matrice représente un pixel. Chaque pixel est une liste de trois valeurs : la composante rouge, la composante verte et la composante bleue (R, G, B).

Détermination de la valeur maximale des pixels :

Le programme vérifie ensuite le type de données des pixels grâce à `pixels.dtype`. Si le type est `np.uint8`, la valeur maximale est 255, correspondant à un encodage sur 8 bits. Si le type est `np.uint16`, la valeur maximale sera 65535, pour un encodage sur 16 bits. Pour d'autres types de codage plus rares, le programme utilise la fonction `pixels.max()` pour déterminer la valeur maximale des pixels.

Enregistrement de la matrice dans un fichier texte :

Le programme génère un fichier texte dans le même répertoire que l'image. Ce fichier est au format PNM (Portable Anymap), un format simple qui stocke les images en texte. Le fichier commence par l'en-tête "P3", suivi des dimensions de l'image et de la valeur maximale des pixels. Ensuite, chaque ligne de pixels est écrite sous la forme de triplets représentant les valeurs RGB. Le fichier est créé grâce à la fonction `open()` avec l'option d'écriture "w".

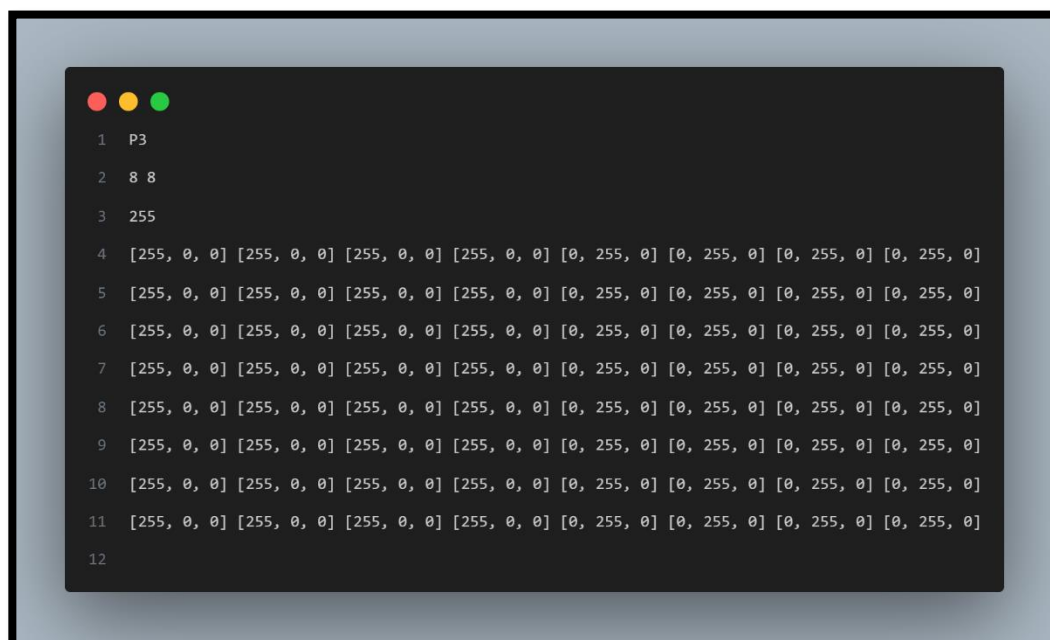
Confirmation de la génération du fichier :

Une fois que le fichier est généré, un message de confirmation est affiché indiquant le chemin du fichier généré, ainsi que les dimensions de l'image et la valeur maximale des pixels.

Vérification avec la méthode inverse :

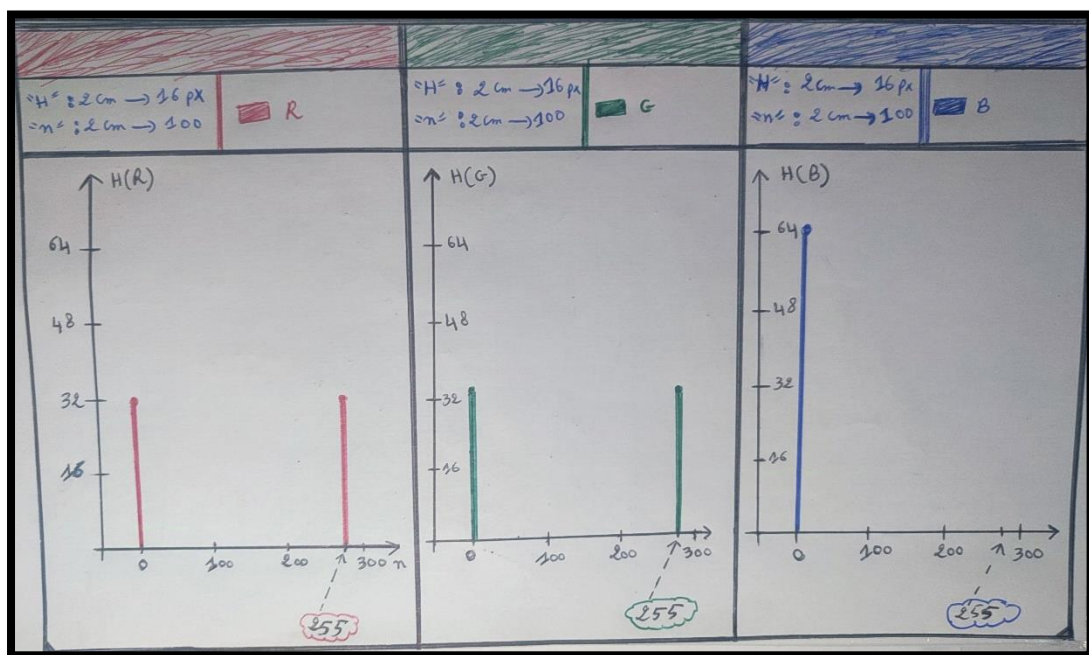
Pour vérifier que le programme fonctionne correctement, nous avons utilisé la même image que dans la question 2, dont nous connaissions déjà la matrice. Le processus de vérification s'est déroulé comme suit :

Le programme a généré la matrice de l'image et l'a enregistrée dans un fichier texte. La matrice obtenue représente chaque pixel sous la forme d'un triplet RGB. Cette matrice correspond exactement à celle de l'image initiale.



```
1 P3
2 8 8
3 255
4 [255, 0, 0] [255, 0, 0] [255, 0, 0] [255, 0, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0]
5 [255, 0, 0] [255, 0, 0] [255, 0, 0] [255, 0, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0]
6 [255, 0, 0] [255, 0, 0] [255, 0, 0] [255, 0, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0]
7 [255, 0, 0] [255, 0, 0] [255, 0, 0] [255, 0, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0]
8 [255, 0, 0] [255, 0, 0] [255, 0, 0] [255, 0, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0]
9 [255, 0, 0] [255, 0, 0] [255, 0, 0] [255, 0, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0]
10 [255, 0, 0] [255, 0, 0] [255, 0, 0] [255, 0, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0]
11 [255, 0, 0] [255, 0, 0] [255, 0, 0] [255, 0, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0] [0, 255, 0]
12
```

Nous avons ensuite isolé les trois couleurs principales de l'image : le rouge, le vert et le bleu. Pour chaque couleur, nous avons calculé l'histogramme, qui montre la répartition des valeurs des pixels pour chaque couleur. Ces histogrammes ont été dessinés pour visualiser la fréquence des différentes valeurs.



Les résultats obtenus lors de la vérification ont montré que le code fonctionnait comme prévu. Les histogrammes des couleurs rouge, vert et bleu étaient cohérents avec les données de l'image, et la recreation de l'image à partir de la matrice a confirmé la validité du programme.

5. Application du Filtre de Nagao sur une Image en Niveaux de Gris

Le filtre de **Nagao** est un algorithme de filtrage non linéaire qui sert à améliorer les images en atténuant les bruits tout en conservant les bords et les détails. Il utilise une fenêtre de taille 5x5 autour de chaque pixel et évalue les sous-fenêtres de taille 3x3 pour déterminer la meilleure représentation locale de l'image. Le pixel central de la fenêtre 5x5 est remplacé par la moyenne des pixels de la sous-fenêtre qui présente la variance la plus faible, ce qui permet de préserver les détails tout en réduisant les bruits.

Code Python pour Appliquer le Filtre de Nagao

Voici un programme Python qui applique le filtre de Nagao à une image en niveaux de gris. Le programme utilise Pillow pour le traitement de l'image, NumPy pour les calculs matriciels, et Matplotlib pour afficher les images avant et après filtrage.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from PIL import Image
4 from os import path
5
6 def nagao_filter(image):
7     m, n = image.shape
8     Imfin = np.zeros((m, n), dtype=np.uint8) # Image filtrée
9
10    # Boucles pour parcourir chaque pixel sauf les bords
11    for k in range(2, m-2):
12        for l in range(2, n-2):
13            A = image[(k-2):(k+3), (l-2):(l+3)] # Extraire une fenêtre 5x5 autour du pixel (k, l)
14            variances = []
15
16            # Nous allons analyser chaque sous-fenêtre 3x3 de la fenêtre 5x5
17            for i in range(3):
18                for j in range(3):
19                    # Extraire la sous-fenêtre 3x3 à partir de (i, j)
20                    sub_matrix = A[i:i+3, j:j+3]
21                    variances.append(np.var(sub_matrix)) # Calcul de la variance de cette sous-fenêtre
22
23            # Trouver l'indice de la sous-fenêtre avec la variance minimale
24            min_variance_idx = np.argmin(variances)
25
26            # Calculer la moyenne des pixels de la sous fenêtre avec la variance minimale
27            best_sub_matrix = A[min_variance_idx//3:min_variance_idx//3+3, min_variance_idx%3:min_variance_idx%3+3]
28            Imfin[k, l] = np.mean(best_sub_matrix)
29
30    return Imfin
31
32 # Demander à l'utilisateur d'entrer le chemin de l'image
33 image_path = input("Entrez le chemin de l'image (ex: suburb_g.ima ou suburb_g.png) : ")
34
35 if not path.exists(image_path):
36     print("Image non trouvée !")
37     exit()
38
39 # Lire l'image en niveaux de gris
40 image = Image.open(image_path).convert('L') # Convertir en niveaux de gris
41 image = np.array(image) # Convertir en tableau NumPy
42
43 # Appliquer le filtre de Nagao
44 filtered_image = nagao_filter(image)
45
46 # Afficher l'image originale et l'image filtrée
47 plt.figure(figsize=(12, 6))
48 plt.subplot(1, 2, 1)
49 plt.title("Image originale")
50 plt.imshow(image, cmap='gray')
51 plt.axis('off')
52
53 plt.subplot(1, 2, 2)
54 plt.title("Image filtrée (Filtre de Nagao)")
55 plt.imshow(filtered_image, cmap='gray')
56 plt.axis('off')
57
58 plt.show()
59
60 # Sauvegarder l'image filtrée au format PNG avec PIL
61 output_path = input("Entrez le chemin de sortie pour l'image filtrée (ex: suburb_gn.png) : ")
62 filtered_image_pil = Image.fromarray(filtered_image) # Convertir la matrice NumPy en image PIL
63 filtered_image_pil.save(output_path, format='PNG') # Sauvegarder l'image au format PNG
64
65 print(f"l'image filtrée a été sauvegardée sous {output_path}.")
66
```

Explication du Code

Fonction de filtrage de Nagao :

Paramètre : Une image en niveaux de gris (tableau NumPy).

La fonction applique le filtre de Nagao en parcourant chaque pixel de l'image (en évitant les bords).

Fenêtre 5x5 : Pour chaque pixel (k, l) , on extrait une fenêtre 5x5 autour de lui.

Sous-fenêtres 3x3 : Pour chaque fenêtre 5x5, on analyse plusieurs sous-fenêtres 3x3 et on calcule la variance de chaque sous-fenêtre.

Choix de la sous-fenêtre optimale : La sous-fenêtre avec la variance minimale est choisie, et la moyenne des pixels de cette sous-fenêtre est attribuée au pixel filtré correspondant dans l'image.

Retour : Une image filtrée où chaque pixel est remplacé par la moyenne des pixels de la sous-fenêtre choisie.

Chargement et traitement de l'image :

Lecture de l'image : L'image est lue avec Pillow et convertie en niveaux de gris (mode 'L').

Conversion en tableau NumPy : L'image est convertie en un tableau NumPy pour manipuler les pixels plus facilement.

Application du filtre :

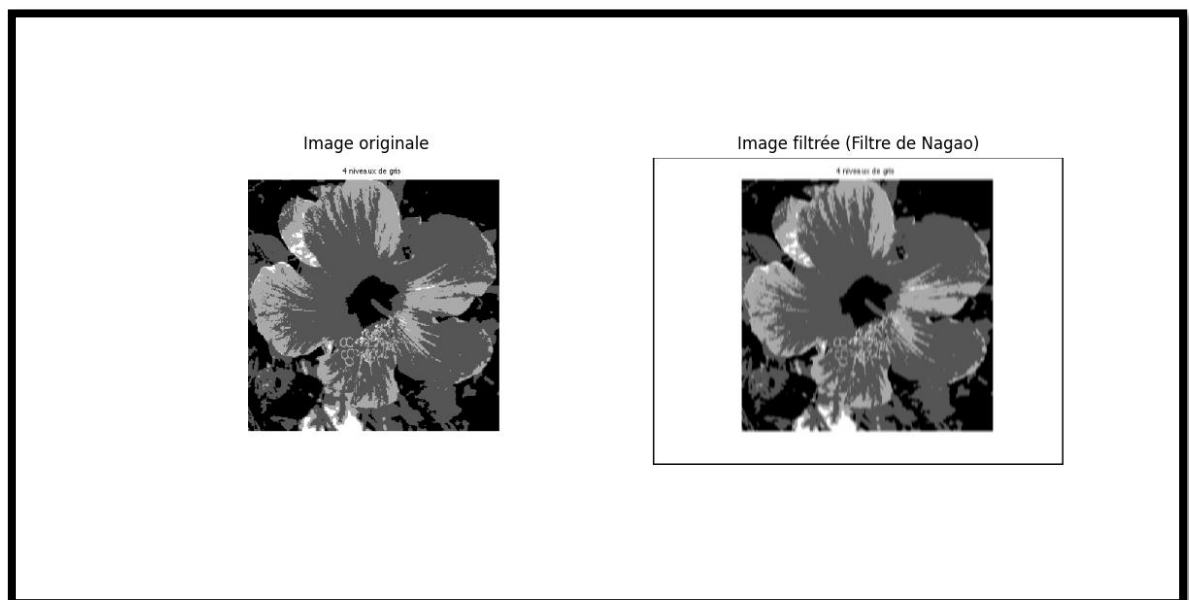
Filtrage : Le filtre de Nagao est appliqué à l'image en utilisant la fonction `nagao_filter()`.

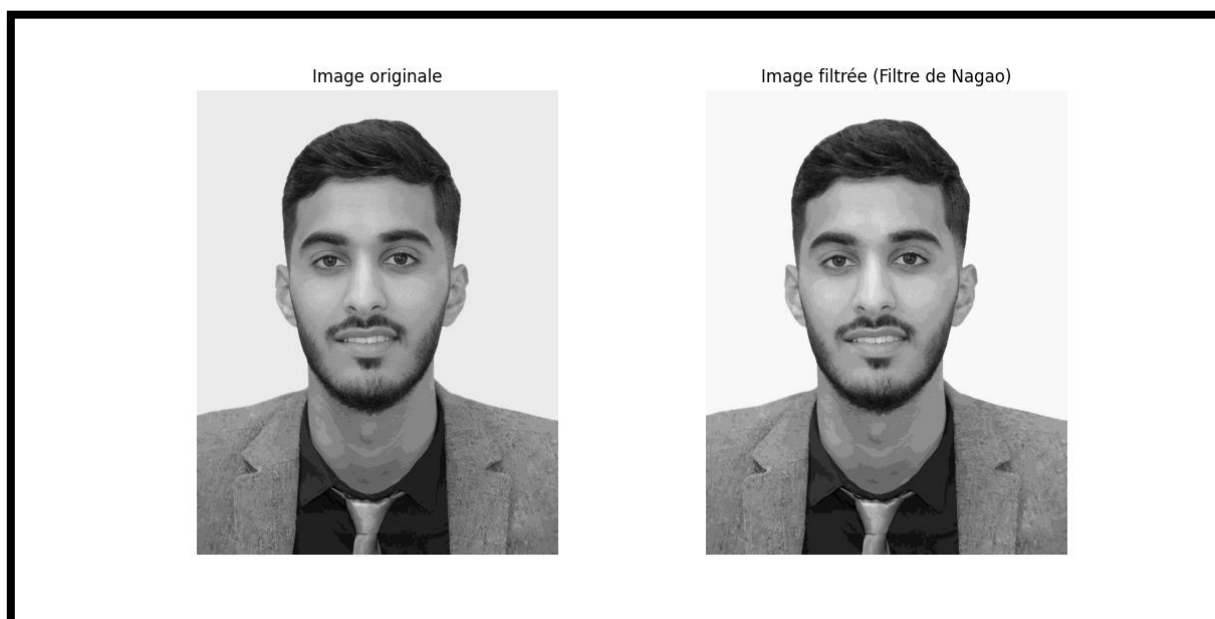
Affichage : Les images originale et filtrée sont affichées côte à côte pour comparaison avec Matplotlib.

Sauvegarde de l'image filtrée :

L'utilisateur peut spécifier un chemin de sortie pour enregistrer l'image filtrée au format PNG.

L'image filtrée est convertie en objet PIL avant d'être sauvegardée.





Conclusion

Ce projet a permis d'explorer plusieurs aspects essentiels du traitement d'images en utilisant le langage Python. À travers la construction et la manipulation d'images binaires, ainsi que l'application d'opérations mathématiques et de transformations sur des images en niveaux de gris, nous avons pu consolider notre compréhension des opérations fondamentales de traitement d'images.

Les résultats obtenus, notamment la visualisation des histogrammes et l'application de filtres, montrent l'importance des outils de calcul numérique tels que NumPy, qui facilitent les manipulations de données et optimisent les calculs d'images. De plus, ce projet met en lumière l'impact de la précision des opérations mathématiques et des choix de codage des niveaux de gris, éléments essentiels dans le domaine du traitement d'images.

En somme, ces travaux pratiques ont permis de renforcer des compétences en programmation, d'acquérir une méthodologie rigoureuse pour la manipulation d'images et de mieux comprendre l'importance des calculs matriciels dans l'analyse et le traitement visuel.

Pour une exploration plus approfondie et accéder à tout le code source de ce projet, veuillez consulter notre dépôt GitHub :

<https://github.com/SaadBarhrouj/traitement-d-images.git>

