Rapport mini-projet-traitement d'image

Introduction:

Notre monde d'aujourd'hui regorge de données et les images en constituent une partie importante. Cependant, pour pouvoir être utilisées, ces images doivent être traitées. Le **traitement d'images** consiste donc à analyser et à manipuler une image numérique principalement dans le but d'en améliorer la qualité ou d'en extraire des informations qui pourraient ensuite être utilisées. Les tâches courantes du traitement d'images incluent l'affichage des images, les manipulations basiques comme le recadrage, le retournement, la rotation, ou encore la segmentation, la classification et les extractions de caractéristiques, la restauration et la reconnaissance d'images. Python devient un choix judicieux pour de telles tâches de traitement d'images. Cela est dû à sa popularité croissante en tant que langage de programmation scientifique et à la disponibilité gratuite de nombreux outils de traitement d'images de pointe dans son écosystème.

I. Les opérations d'entrée et de sorties sur les images:

Dans cette partie de notre projet nous commencerons par définir des fonctions d'entrées et de sorties sur les images que nous utiliserons tout au long du programme.

- Fonction afficher une image:

  La fonction Afficher_Img(Img) ci-dessous prend en argument une image ensuite elle

```
#Partie 1:
#Q1
def AfficherImg(img):
    plt.axis("off")
    plt.imshow(img, cmap = "gray")#palette predefinie pour afficher une image
    plt.imsave("image1.png",img,cmap="gray")
    plt.show()
✓ 0.3s
```

  l'affiche sur l'écran

- Fonction ouvrir une image:

  La fonction ouvrir Image(Img) ci-dessous prend en argument une image Img, ensuite

```
#Q2
def ouvrirImage(chemin):
    img=plt.imread(chemin)
    return img
✓ 0.7s
```

  une retourne une matrice contenant le codage de cette matrice

- Fonction sauvegarder une image:

```
#Q3
def saveImage(img):
    plt.imsave("image1.png",img)
```

La fonction save Image(Img) ci-dessous prend en argument une image Img, ensuite elle enregistrer cette image dans le disque dur.

II. les images en noir et blanc:

Dans cette partie nous développons quelques fonctions pour manipuler les images en mode noir et blanc.

- Fonction créer une image noir:

  On cree une fonction qui génère une image en noir de hauteur et de largeur l . Pour ce faire, la fonction retourne une matrice contenant h lignes et l colonnes dont la valeur de chaque pixel est

```python
#Partie 2:
#Q4
def image_noire(h,l):
    return [[0]*h for i in range(l)]
```
✓ 0.4s

  initialisé par 0

- Fonction créer une image blanche:

  On cree une fonction qui génère une image en noir de hauteur et de largeur l . Pour ce faire, la fonction retourne une matrice contenant h lignes et l colonnes dont la valeur de chaque pixel est

```python
#Q5
def image_blanche(h,l):
  return np.ones((h,l,3))
```
✓ 0.2s

  initialisé par 1

- Fonction créer une image noir et blanc:

  nous avons créer une fonction qui retourne une image noir et blanc dont le contenu de chaque pixel (i,j) est donné par (i+j)%2

```python
#Q6
def creerImgBlancNoir(h,l):
    M=image_noire(h,l)
    for i in range(len(M)):
        for j in range(len(M[i])):
            if (i+j)%2==1:
                M[i][j]=1
    return M
```
✓ 0.2s

- Fonction mettre une image en négatif:

Nous avons créé une une fonction d'en tête def négatif(Img) qui construit le négatif de l'image représentée par une matrice Img. Pour cela, il nous a suffit d'inverser les valeurs de la matrice t (0 devient 1 et 1 devient 0)

- Fonction test:

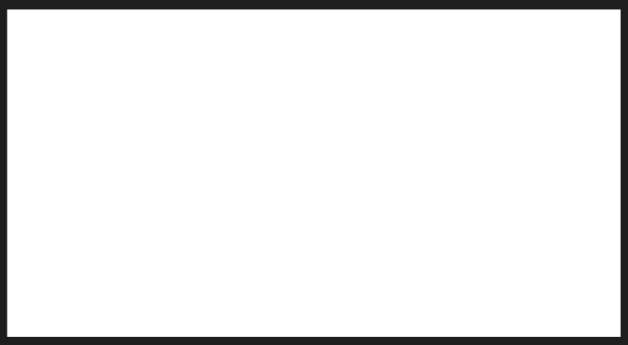
  C'est la fonction dans laquelle nous avons testé toutes les fonctions précédentes.

```
#Q8
print(image_noire(4,4))
AfficherImg(image_noire(4,4))
print(image_blanche(4,4))
AfficherImg(image_blanche(4,4))
print(creerImgBlancNoir(4,4))
AfficherImg(creerImgBlancNoir(4,4))
print(negatif(creerImgBlancNoir(4,4)))
AfficherImg(creerImgBlancNoir(4,4))
```

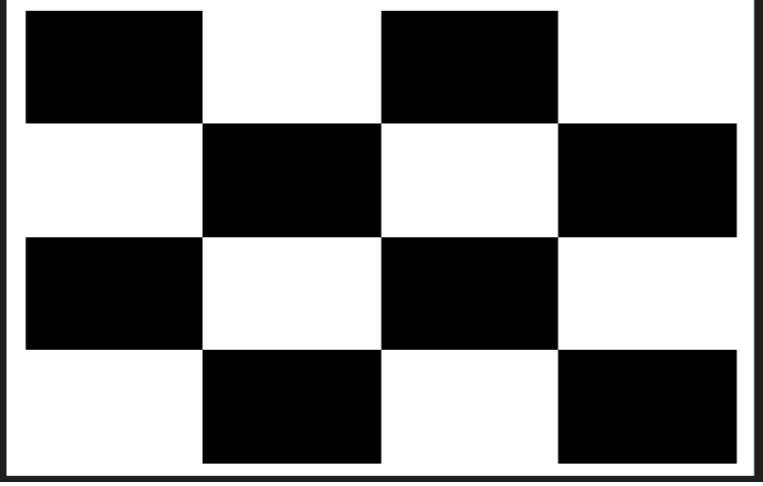
✓ 3.7s

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]]
```



```
[[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1], [1, 0, 1, 0]]
```



III. Les images en niveaux de gris:

Dans cette partie nous développons quelques fonctions pour manipuler les images en niveau de gris.

- Fonction luminances d'une image:

  La luminance (ou brillance) d'une image est définie comme la moyenne de tous les

  pixels de l'image. Nous avons créé une fonction d'entête def luminance(Img) qui retourne la

```
#Partie 3:
#Q9
def luminance(Img):
    s=0
    for i in Img:
        for j in i:
            s+=j[0]
    return s/(len(Img)*len(Img[0]))
print(luminance(ouvrirImage("lengris.png")))
```

```
[40]  ✓ 1.1s
...   0.7230342741230471
```

  luminance d'une image .

- Fonction contrast d'une image:

Le contraste peut être défini comme la variance des niveaux de gris (N nombre de pixels dans l'image). Nous avons créé une fonction d'en tête def contrast(Img) qui retourne le contraste d'une

```
#Q10
def constrast(Img):
    Moy=luminance(Img)
    N=len(Img)*len(Img[0])
    s=0
    for i in Img:
        for j in i:
            s+=(j[0]-Moy)**2
    return s/N
print(constrast(ouvrirImage("lengris.png")))
```

```
[25] ✓ 8.6s
... 0.08453961297943265
```

image .

- Fonction profondeur d'une image:

  Le contraste peut être défini comme la variance des niveaux de gris (N nombre de pixels dans l'image). Nous avons créé une fonction d'en tête def contrast(Img) qui retourne le contraste

```
#Q11
def profondeur(img):
    im=ouvrirImage(img)
    return np.max(im)
print(profondeur("lengris.png"))
```

```
] ✓ 0.7s
1.0
```

d'une image .

- Fonction ouvrir une image:

```
#Q12
def Ouvrir(Img):
    matG = cv2.imread(Img, 0)
    return matG     #erreur
print(Ouvrir("Lenna.png"))
```

```
] ✓ 0.3s
[[168 168 167 ... 171 177 154]
 [168 167 167 ... 174 182 158]
 [169 166 167 ... 150 132  99]
 ...
 [ 59  61  64 ...  82  98 102]
 [ 56  61  61 ...  95 110 111]
 [ 53  61  59 ... 106 116 118]]
```

On a créé une fonction qui retourne la matrice représentant l'image A.

IV. opérations élémentaires sur les images en niveaux de gris:

Dans cette partie nous développons des fonctions pour modifier la position et la mise en page des images que nous avons générées.

- Fonction inverser une image:

Nous avons coder une fonction inverser(img) qui renvoie l'image inverse de l'image img, c'est-à dire que le ton d'un pixel de la nouvelle image est p − v où v est le ton du pixel correspondant de

```
#Partie 4:
#Q13
def inverser(img):
    image = cv2.imread(img, 0) # charge le fichier dans une matrice de pixels gris

    image = 255 - image
    return image
AfficherImg(inverser("Lenna.png"))
```

1.7s

l'image d'origine.

- <u>Fonction flip-horizontale d'une image:</u>

  Nous avons écrit une fonction flipH(img) qui renvoie la transformée de l'image img par la

```
def flipH(img):
    image_flip=cv2.flip(img, 1)
    return image_flip
AfficherImg(flipH(Ouvrir("Lenna.png")))
```

[45] 1.5s

symétrie d'axe vertical passant par le milieu de l'image.

- <u>Fonction poser-verticale d'une image:</u>

  Nous avons écrit une fonction poser(img1,img2) qui prend en arguments deux images img1 et img2 de même largeur et profondeur, et qui renvoie la nouvelle image obtenue en posant img1 sur img2.

```python
#Q15
def poserV(img1,img2):
    I1=Ouvrir(img1)
    I2=Ouvrir(img2)
    v=np.vstack((I1,I2))
    return v
AfficherImg(poserV("Lenna.png","Lenna.png"))
```

✓ 1.2s

- <u>Fonction poser-horizontale d'une image:</u>

  Nous avons écrit une fonction poserH(img1, img2) qui prend en arguments deux images img1 et img2 de même hauteur et profondeur, et qui renvoie la nouvelle image obtenue en posant img2 à

```python
#Q16
def poserH(img1,img2):
    I1=Ouvrir(img1)
    I2=Ouvrir(img2)
    h=np.hstack((I1,I2))
    return h
AfficherImg(poserH("Lenna.png","Lenna.png"))
```

✓ 0.7s

droite de img1.

V. <u>les images en RGB</u>

Les images RGB en couleur sont constituées de trois couches (RGB). Les données de l'image sont stockées dans un tableau à trois dimensions : la première dimension correspond à la couleur (rouge, vert ou bleu, indice 0, 1 ou 2). Ainsi, les dimensions du tableau sont : 3 × n × m où n le nombre des lignes et m le nombre des colonnes. La valeur associée à chaque pixel est un entier compris entre 0 et 255.

- <u>Opérations d'affichages des expressions:</u>

```python
M=[[[210, 100, 255],[100, 50, 255],[90, 90, 255],[90, 90, 255],[90, 90, 255],[90, 80, 255]],
[[190, 255,89],[ 201, 255,29],[200, 255,100],[100, 255,90],[20, 255,200], [100, 255,80]],
[[255,0, 0],[ 255,0, 0],[255,0, 0],[255,0, 0],[255,0, 0], [255,0, 0]]]
#Q22
print(M[0][1][1])
print(M[1][0][1])
print(M[2][1][0])
```

✓ 0.8s

```
50
255
255
```

- Déterminations de la quantité de mémoire nécessaire en octets pour stocker le tableau représentant une image en RGB, avec justifications:

```
#Q23
#puisque la taille d'un entier est 4 octets
#alors la quantité de mémoire nécessaire en octets(8 bits) pour stocker le tableau représentant une image RGB est:
#4*la taille du tableau = 4*3*n*m = 12*n*m
```

- Fonction initialiser image RGB:

  Nous avons écrit une fonction def initImageRGB(imageRGB) permettant d'initialiser et de renvoyer le tableau image RGB à trois dimensions .

```
#Q24
import random
def initImageRGB(imageRGB):
    imageRGB=[[[random.randint(0,255)]*3 for i in range(3)] for j in range(3)]
    return imageRGB
AfficherImg(initImageRGB(ouvrirImage("Lenna.png")))
```

[35] ✓ 1.2s

- Fonction symétrie d'une image par rapport à l'axe horizontale:

  Nous avons écrit une fonction def_symetrie(img) qui retourne une image symétrique à l'image img par rapport à l'axe horizontal .

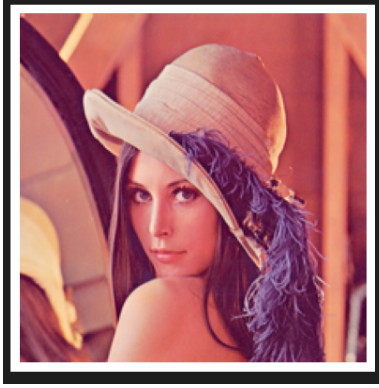
```
#Q25
def symetriehorizontal(img):
    return img[::-1]
AfficherImg(ouvrirImage("Lenna.png"))
AfficherImg(symetriehorizontal(ouvrirImage("Lenna.png")))
```

- Fonction Symétrie d’une image par rapport à l’axe verticale:

  Nous avons écrit une fonction def_symetrie(img) qui retourne une image symétrique à

```
def symetrievertical(img):
    M=ouvrirImage(img)
    T=[[[0]*3 for i in range(len(M[0]))] for j in range(len(M))]
    for i in range(len(M)):
        for j in range(len(M[i])):
            T[i][j]=M[i][-j-1]
    return AfficherImg(T)
symetrievertical("Lenna.png")
```

6.8s



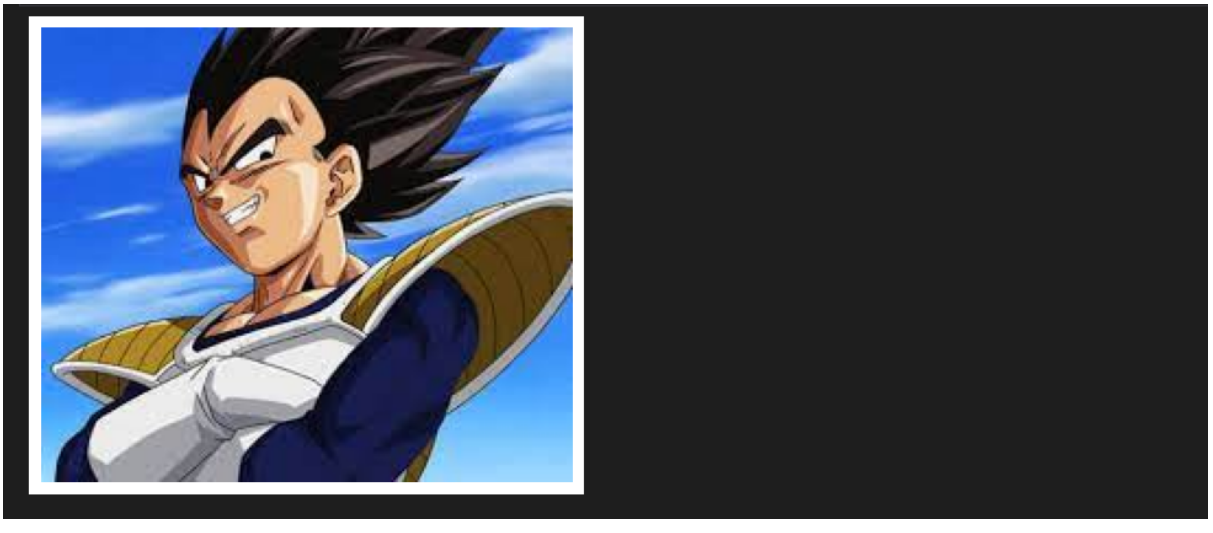
  l'image img par rapport à l'axe vertical .

- Fonction GRAYSCALE:

  Nous avons écrit une fonction def grayscale(image RGB) qui renvoie une image en niveaux de gris, sous forme de tableau à deux dimensions de taille n × m contenant des valeurs entières comprises entre 0 et 255.

```
#026
def grayscale(imageRGB):
    T=ouvrirImage(imageRGB)
    s,max,min=0,0,0
    M=[[0]*len(T[0]) for f in range(len(T))]
    for i in range(len(T)):
        for j in range(len(T[i])):
            max,min=T[i][j][0],T[i][j][0]
            for k in range(len(T[i][j])):
                if max<T[i][j][k]:
                    max=T[i][j][k]
                if T[i][j][k]<min:
                    min=T[i][j][k]
            s=(max+min)//2
            M[i][j]=s
    return M
AfficherImg(ouvrirImage("vege.jpeg"))
AfficherImg(grayscale("vege.jpeg"))
```

[37] ✓ 7.4s



Conclusion:

Nous sommes arrivés au terme de ce projet sur le traitement d'image python qui a fait recours a beaucoup de notion de base vu en cours mais qui également nous à pousser à la recherche pour approfondir nos connaissances dans un sens un plus pratique notamment par l'utilisation de nouvelles bibliothèques.

En somme le traitement d'image n'est qu'une discipline qui consiste à manipuler des images par l'intermédiaire de modifications matricielles, nous espérons très prochainement avoir à travailler sur un sujet dans ce cadre d'activité.