Programmation Python
Math Licence 2, IOB
Projet : traitement d'images

Dans ce projet, vous allez développer des fonctions pour du traitement d'images.

Préambule

Lire un fichier contenant une image

Pour manipuler les images, on a besoin des bibliothèques de programmes `numpy`, `matplotlib` et `scipy`.

Exemple pour lire une image en PYTHON :

```
import numpy as np
from scipy import misc

damier = misc.imread("damier-ng.jpeg")
misc.imsave("damier-ng.png", damier)
```

En exécutant l'instruction `damier.shape`, on vérifie que `damier` est un tableau numpy de taille 795×1024. Ce tableau contient pour chaque pixel un entier variant entre 0 et 255 qui mesure le niveau de gris. Les cases du tableau peuvent être accédées individuellement ou par tranche

`damier[i,j]`	pixel de coordonnées
`damier[i]`	ligne i
`pixels[i:k, j:l]`	sous-tableau formé des lignes de i à k-1 et des colonnes j à l-1.
`pixels[:k, j:]`	sous-tableau formé des lignes inférieures strictement à k et des colonnes supérieure ou égale à j.
`pixels[:, j]`	colonne j

Les images RGB (ie en couleur) sont des tableaux. Ils ont 3 dimensions : les deux dimension de la taille de l'image × 3.

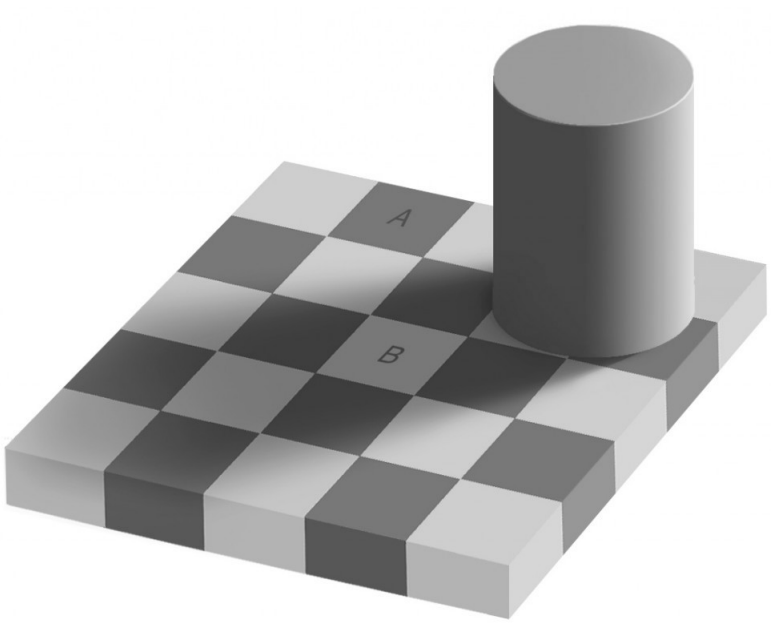
```
paysage = misc.imread("paysage.png")
paysage.shape

Out[48]: (682, 1024, 4)
```

La première "couche" `paysage[:,:,0]` contient les niveaux de rouge,
la seconde `paysage[:,:,1]` les niveaux de vert
et la troisième `paysage[:,:,2]` les niveaux de bleu.

Afficher une image

Pour afficher une image on utilise la fonction `plt.imshow`. Pour afficher l'image en niveau de gris, on doit spécifier l'échelle de couleur par l'argument `cmap`.

```
import matplotlib.pyplot as plt
plt.imshow(damier,cmap="gray")
```



Pour afficher une image en couleur, l'échelle de couleur est spécifiée par défaut.

```
paysage = misc.imread("paysage.png")
plt.imshow(paysage)
```

Important

— Dans les questions qui suivent, on vous demande d'écrire des fonctions pour manipuler des images. Les étudiants qui sont à l'aise en programmation Python sont invités à définir une (ou plusieurs) classes. Le travail est individuel.

— Vous devez rendre des codes commentés, incluant les fonctions et les scripts qui les appellent ainsi que les images utilisées. Les différents fichiers doivent être regroupés dans un unique fichier .zip portant votre nom (ex : monbet.zip).
Pour évaluer votre travail, je vais tester vos codes avec mes images. Il doivent donc s'exécuter quelque soit la taille de l'image.
Si vos fonctions sont définies dans des fichiers différents, vous devez fournir un script où elles sont appelées.

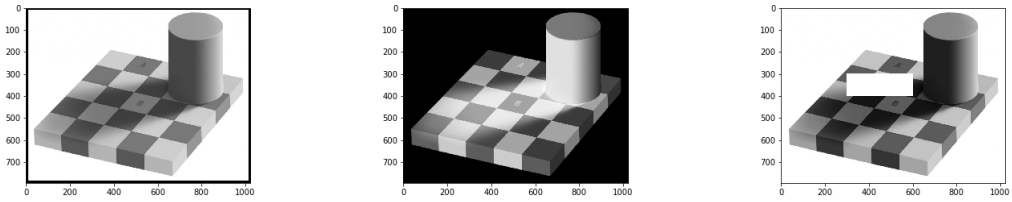
— Le travail sera envoyé par mail à l'adresse `valerie.monbet@univ-rennes1.fr` **au plus tard le 12 nov. 2017**. Aucun délai ne sera accordé. Le sujet du mail : "projet IOB".

1 Modifier les niveaux de gris

Écrire une fonction pour chaque action listée ci-dessous. Ces fonctions prendront en argument l'image et les paramètres nécessaires.

1. Modifier les pixels des bords de l'image de façon à créer un bord noir de k pixels de largeur.
2. Modifier l'image en inversant l'intensité des pixels. En particulier le blanc et le noir seront inversés.
3. Poser un masque blanc dans une fenêtre de taille $L \times \ell$ pixels dont le coin gauche du bas est positionné au pixel de coordonnées (x, y).

(1) (2) (3)

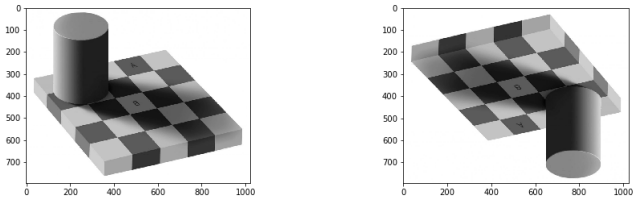


Tester ces fonctions en ajoutant à une image en niveau de gris de votre choix un bord noir de 10 pixels de large, un fenêtre blanche de taille 300×100 positionnée en $(300, 300)$. Vous pouvez utiliser l'image `damier`.

2 Effet mirroir

Écrire une fonction pour renverser l'image par effet miroir selon une ligne verticale ou horizontale.

(1) (2)



3 Floutage et extraction de contours

3.1 Floutage

Pour flouter une image, on lui applique une convolution. Ceci consiste à remplacer un pixel par une combinaison linéaire de ses pixels voisins.

On peut décrire la transformation par une matrice. Notons la K. Pour chaque pixel qui n'est pas au bord de l'image, on peut considérer qu'il est au centre d'un carrée de 5 par 5 pixels et on va remplacer ce pixel par la moyenne pondérée de ses 25 voisins.

Si la matrice

$$K = \frac{1}{100}\begin{pmatrix} 1 & 2 & 4 & 2 & 1 \\ 2 & 4 & 8 & 4 & 2 \\ 4 & 8 & 16 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{pmatrix}$$

les pixels proches du centre ont un poids plus fort que les pixels éloignés.

En pratique, on va parcourir l'image et calculer la somme (`np.sum`) du produit terme à terme des matrices K et la chaque sous image de taille 5×5 pixels.
Écrire la fonction `filtrer(im, K)` qui prend en entrée l'image à modifier et retourne l'image filtrée.
Appliquer une fois, puis plusieurs fois le filtre à une image de votre choix.

Image

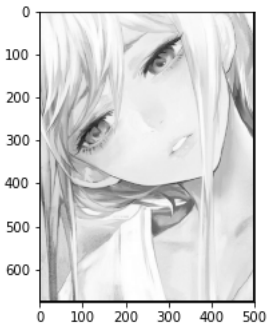
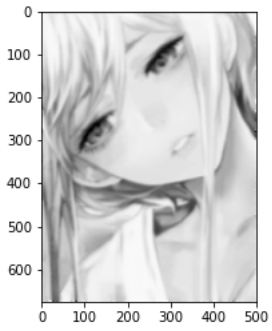


Image filtrée



On note que cette méthode sert aussi pour débruiter une image.

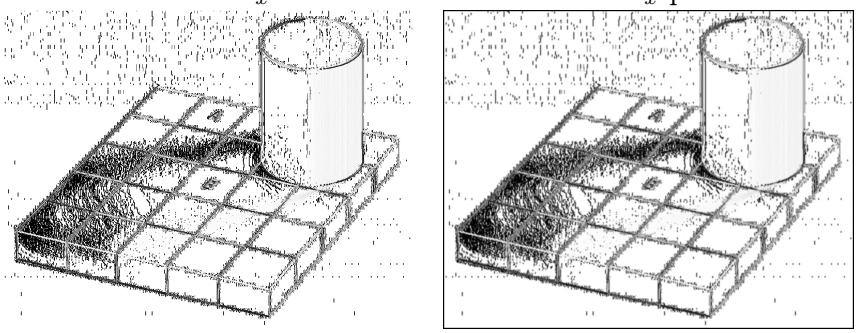
3.2 Extraction de contours

La même technique peut être utilisée pour extraire les contours des images. Il suffit de définir une autre matrice de filtrage. Par exemple

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \text{ou } G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Utiliser votre fonction `filtrer` pour extraire les contours de l'image choisie.

Filtre G_x Filtre G_x puis K



4 Photomaton

Les images en couleur sont codées sur 3 canaux (R=rouge, B=bleu, G=vert). Le tableau aura 3 dimensions et la troisième dimension correspond aux 3 couleurs. Il faut penser appliquer les transformations aux 3 couleurs.

```
starwars = misc.imread('starwars.jpg')
starwars[100, 105, 0]     # Canal 0 du pixels (100, 105)
Out[278]: 131
starwars[100, 105, :]
Out[279]: array([131,  64,   0], dtype=uint8)# 3 canaux du pixels (100, 105)
starwars[100, 105]
Out[280]: array([131,  64,   0], dtype=uint8)# 3 canaux du pixels (100, 105)
starwars[:, :, 2]                  # Canal 2 complet
Out[281]:
array([[255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       ...,
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255]], dtype=uint8)
plt.imshow(starwars) # on n'utilise plus l'argument cmap
```

L'image de droite est composée de 256×256 pixels. L'image du milieu a été obtenue à partir de l'image de droite en déplaçant les pixels :

— les pixels des lignes paires sont dans la partie haute et les pixels des lignes impaires dans la partie basse ;
— les pixels des colonnes paires sont dans la partie gauche et les pixels des colonnes impaires dans la partie droite.

Écrire une fonction `photomaton` pour construire l'image du milieu à partir de l'image de gauche. Appliquer la fonction à l'image du milieu pour obtenir l'image de droite à partir de l'image du milieu.

5 Stéganographie

La stéganographie est l'art de la dissimulation : son objet est de faire passer inaperçu un message dans un autre message. Elle se distingue de la cryptographie, « art du secret », qui cherche à rendre un message inintelligible à autre que qui-de-droit.

Chaque pixel d'une image RGB est codé sur 3 octects (un par canal). Au lieu d'utiliser les 8 bits pour coder un entier quelquonque entre 0 et 255, seuls les quatre bits de poids forts (4 premiers bits) ont été utilisés pour coder l'image apparente de `paysage_zen.png`. Les quatre bits de poids faibles (4 derniers bits) peuvent alors être utilisés pour l'image cachée.

1. Supposons que la valeur d'un pixel bleu de l'image d'origine soit A=105 dans l'image apparente originale et C=96 dans l'image cachée originale.
   (a) Écrivez les nombres A et C en binaire (voir di-cessous).
   (b) Si on met à 0 les 4 bits de poids faibles de A, quel nombre obtient-on ?
   (c) Si on remplace ces quatre bits de poids faibles par les quatre bits de poids forts de C, quel nombre obtient-on ?

   C'est ce nombre B qui remplace A dans le fichier `paysage_zen.png`.
2. Écrire une fonction `decode` pour faire apparaitre l'image cachée.

Ci dessous un exemple de conversion entier → binaire et binaire → entier

```
A = 105 # valeur d'un pixel dans un canal donné
Abin = bin(A)
Abin
Out[284]: '0b1101001'

C = 96
Cbin = bin(C)
Cbin
Out[285]: '0b1100000'

int( '0b1100000', 2)
Out[299]: 96
```

Ecrire une fonction `steganographie` qui prend en entrée l'image `paysage_zen.png` à décoder et retourne l'image cachée.