TP d'informatique n°11
Manipulation d'image

PCSI 2024 – 2025

I Objectifs :

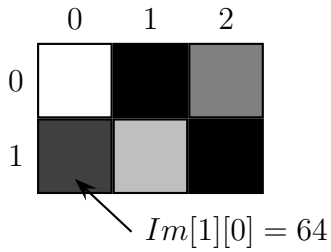
- manipuler des images en niveau de gris ;
- manipuler des images en couleur RGB ;
- réaliser des boucles imbriquées et comprendre l'ordre dans lequel les instructions sont exécutées ;
- évaluer la complexité des algorithmes et comprendre pourquoi le traitement d'image prend du temps.

II Introduction

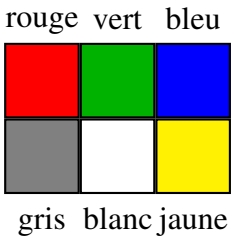
Une image numérique est généralement[1] représentée comme un tableau à deux dimensions de « pixels ». Dans le cas d'une image en niveau de gris, chaque pixel contient un nombre (généralement sur 8 bits, donc entre 0 et 255) représentant la luminosité du pixel : 0 pour noir, 255 pour blanc et les valeurs intermédiaires étant des gris plus ou moins sombres. Une autre convention fréquemment utilisée pour l'intensité d'un pixel est d'utiliser un nombre « réel[2] » : **float**. Le nombre est alors compres entre 0.0 (noir) et 1.0 (blanc).

En terme d'indexation, on a donc deux indices, le premier correspond en général au numéro de la ligne dans le tableau en partant du haut et le deuxième au numéro de la colonne en partant de la gauche (convention similaire aux matrices).

Dans une image en couleur, on représente fréquément[3] la couleur en trois composantes, **R**ouge, **V**ert, **B**leu (RVB en français, RGB en anglais). Chaque pixel est donc un tableau de trois nombres représentant l'intensité de chaque couleur. Ainsi si un pixel contient $[0,255,0]$, sa couleur sera verte, et s'il contient $[255,255,255]$ il sera blanc. On peut donc voir une image en couleur comme un tableau à trois dimension (ligne, colonne, couleur).



(1) image N&B



(2) image couleur


(3) icone de firefox

Exercice n°1 Pour les images (1) et (2) ci-dessus, écrire une listes de listes pouvant représenter l'image.

Pour charger et afficher des images, on utilisera le module `matplotlib` et en particulier les sous modules image et pyplot.

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
```

1. il y toujours des exceptions, on peut citer le cas des images vectorielles qui ne sont absolument pas pixelisées, quelque soit le zoom.
2. Nous verrons au second semestre la différence entre les nombres réels de votre cours de maths et les nombres généralement manipulés par les ordinateurs
3. Il y a parfois une composante supplémentaire pour la transparence et on utilise parfois d'autres « espaces colorimétrique » comme sRGB ou CMJN, XYZ…

```
im0 = # à vous de compléter avec l'exemple (1) ci dessus
plt.imshow(im0, cmap = "gray") #cmap = "gray" force l'affichage en gris
im1 = # à vous de compléter avec l'exemple (2) ci dessus
im1 = np.array(im1, dtype = np.uint8)  # conversion pour utiliser des
   entiers positifs sur 8 bits
plt.figure()      # crée une nouvelle fenêtre pour éviter que les
   images ne se superposent
plt.imshow(im1)  # affiche l'image sur la figure
plt.show() # force l'ouverture des figures créées
new_im = np.zeros( (100, 200))  # création d'une image noir et blanc de
    taille 100 lignes et 200 colonnes
new_im = np.zeros( (100, 200, 3), dtype = np.uint8)  # création d'une
   image couleur de taille 100 lignes et 200 colonnes
image = mpimg.imread("chaton.png")   # Charge l'image chaton.png si
   elle est dans le répertoire du programme
```

Exercice n°2 À l'aide des exemples ci dessus, affichez les deux images que vous avez créées dans l'exercice précédent.

Exercice n°3 Des images sont disponibles sur `http://grenard.dyndns.org/info.html`. Charger l'image $owl\_negatif.png$[4] et déterminer (dans python) les dimensions de l'image. Cela sera très utiles dans beaucoup de fonctions. Créer une fonction `shape_gris(im:"image")->(int,int)` qui prend en argument une image en niveau de gris (sous forme d'un tableau) et qui renvoie un tuple contenant dans l'ordre le nombre de ligne et le nombre de colonne. Cette fonction « pourra » être réutilisée par la suite.

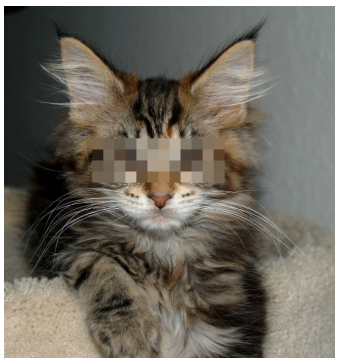
III Quelques traitements élémentaires

1. Anonymat

Exercice n°4 Modifier l'image fournie $lego\_chimiste\_Gris.png$ pour ajouter un bandeau noir devant les yeux du personnage. On pourra dans un premier temps afficher l'image avec `plt.imshow` pour repérer quels pixels doivent devenir noirs.

Exercice n°5 (Bonus si vous êtes rapide[5])

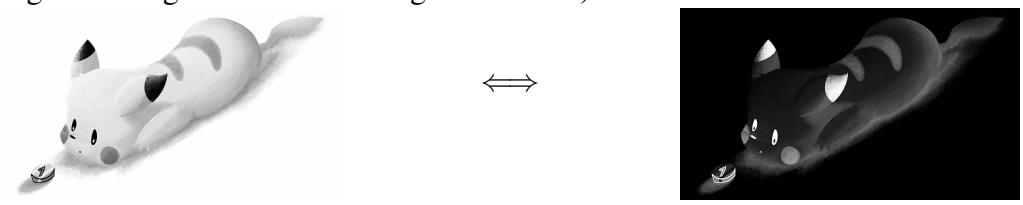
Il s'agit d'une amélioration de l'exercice précédent. Créer une fonction `pixelise(im:"image", ideb:int, ifin:int, jdeb:int, jfin:int, largeur:int)->"image"` qui prend en argument une image et des entiers et qui pour la zone comprise entre (ideb,jdeb) et (ifin, jfin) découpe la zone en carrés de côté $largeur$ et remplace la valeur de chaque pixel du carré par la moyenne sur le carré. Testez avec l'image précédente et $largeur = 10$. Exemple ci-contre.



4. Crédit image : Dommk
5. Crédit photo : John Nyberg

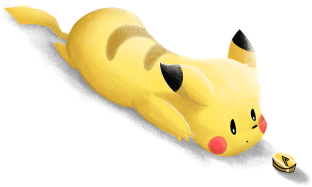
2. Négatif

Un passage en négatif revient à inverser les pixels sombres et lumineux. On peut voir un exemple[6] ci-dessous. (Le négatif du négatif est donc l'image elle même).



Exercice n°6 Écrire une fonction `negatif(im:"image")->"image"` qui prend en argument une image en noir et blanc (sous forme d'un tableau 2D) et renvoie son négatif (même forme).
Testez avec l'image *owl_negatif.png* qui est un négatif, vous devriez retrouver l'image d'origine. (Attention, matplotlib.image.imread interprète généralement les images comme un flottant (réel) entre 0 et 1).

image initiale



miroir horizontal

rotation 90° à gauche

3. Miroir horizontal

Exercice n°7 Écrire une fonction `miroir_horizontal(im:"image")->"image"` qui prend en argument une image et renvoie une autre image correspondant à une symétrie par rapport à un axe vertical passant par le milieu de l'image (les pixels à droite initialement vont à gauche et vice versa). Tester avec une des images fournies de votre choix (de préférence rectangulaire et non carrée).

4. Rotation

Exercice n°8 Écrire une fonction `rotation90gauche(im:"image")->"image"` qui prend en argument une image et renvoie une autre image correspondant à une rotation de 90° vers la gauche. Aide en note de bas de page si besoin[7]. Tester avec une des images fournies de votre choix (de préférence rectangulaire et non carrée).

IV De la convolution pour des traitements plus évolués

De nombreux traitement d'images sont à base de moyenne pondérée : on remplace chaque pixel par une moyenne pondérée sur les pixels avoisinant. On enregistre souvent les coefficients de pondération dans un tableau 2D appelé noyau ou kernel. En fonction du noyau choisi, on peut réaliser différentes opérations (atténuation du bruit, floutage, détection de contours).

6. Crédit image : Dommk

7. (1) faire un schéma (2) créer une image vide de la bonne taille, pensez à réutiliser shape (3) sur le schéma prendre un pixel quelconque (i,j) (avec i très différent de j et loin du centre) et regarder comment ses coordonnées se transforment lors de la rotation (4) faire une boucle sur tous les pixels pour appliquer la formule précédente.

Par exemple, si k est un noyau de dimension 3×3, le calcul d'un pixel de l'image de sortie s en fonction de l'image d'entrée e se ferait de la façon suivante :

$$\begin{aligned} s[i][j] \quad = \quad & e[i-1][j-1] \times k[0][0] + e[i-1][j] \times k[0][1] + e[i-1][j+1] \times k[0][2] + \\ & e[\quad i \quad][j-1] \times k[1][0] + e[\quad i \quad][j] \times k[1][1] + e[\quad i \quad][j+1] \times k[1][2] + \\ & e[i+1][j-1] \times k[2][0] + e[i+1][j] \times k[2][1] + e[i+1][j+1] \times k[2][2] \end{aligned}$$

On ne considérera que des noyaux de taille impaire à la fois en largeur et en hauteur. Le noyaux k est souvent choisi avec un somme des coefficients égale à 1 pour ne pas modifier l'intensité lumineuse moyenne.

Exercice n°9 Écrire une fonction `calcule_pixel(im:"image", noyau:"image", i:int, j:int)->np.uint8` qui, étant donné une image et un noyau de taille 3×3, renvoie la valeur du pixel de coordonée (i,j) après convolution entre l'image et le noyau. On supposera que le pixel est assez loin du bord pour qu'il n'y ait pas de problème.

1. Filtre gaussien

Exercice n°10 Écrire une fonction `filtre_image(im:"image", noyau:"image")->"image"` qui applique un noyau à toute une image. Pour les pixels des bords pour lesquels la formule ne peut pas s'appliquer simplement[8], on renverra juste 0. Tester avec les noyaux ci-dessous sur l'image "pika_bruite.png" sur laquelle on a volontairement rajouté du bruit[9].

```
moyenne = np.ones( (3,3))/9
gaussien = np.array([ [1,2,1], [2,4,2], [1,2,1] ])/16
```

2. Détection de contour

image initiale

image du contour

négatif du contour

Exercice n°11 On utilise les deux noyaux ci-dessous (filtre de Kirsch) qui sondent les variations verticales et horizontales dans l'image (la somme des coefficients vaut 0, ce qui fait que le filtre renvoie 0 dans les zones homogènes).

```
Kh = np.array([[-3, -3, 5],[-3,0,5],[-3, -3, 5]])/15
Kv = np.array([[-3, -3, -3],[-3, 0, -3], [5, 5, 5]])/15
```

Pour chaque pixel de coordonnés (i,j), on calcule le résultat par chacun des noyaux : x_h et x_v. On calcule ensuite $\sqrt{x_h^2 + x_v^2}$. L'ensemble des valeurs ainsi calculées pour chaque (i,j) forme une nouvelle image qui met en évidence les contours des objets présents sur l'image de départ.

8. on peut envisager des choses plus ou moins compliquées en fonction de nos besoin
9. Ce genre de bruit est fréquemment présent en photo, en particulier lorsque l'on travaille avec une forte amplification ISO due à de mauvaise condition de luminosité.

On peut ensuite éventuellement appliquer une fonction de seuillage : les pixels ayant une valeur inférieur à un seuil s sont ramenés à 0 et les autres à 255. Cela n'a pas été fait sur l'image ci-dessous, mais permettrait d'améliorer le contraste.

3. Temps de calcul

Exercice n°12 Si dans le paragraphe sur l'application d'un noyau à une image la taille de l'image est $L \times H$ et la taille du noyau est $a \times a$, quel est approximativement le nombre de calculs (addition et multiplication) à faire pour exécuter la fonction `filtre_image` ? Et si l'image est en couleur ?

Exercice n°13 Si l'on souhaite appliquer un filtre tel que précédemment à toutes les images d'un film couleur de 10 min à 60 fps (image par seconde), que la résolution est 4K (3840×2160) et qu'une opération élémentaire prend 10 ns à s'éxécuter, quel est le temps de calcul approximatif ?

V Des manipulations plus évoluées

Exercices disponibles à la demande : rotation d'un angle quelconque ; amélioration du contraste ; changement de taille d'un facteur 2 ; changement de taille d'un facteur autre que 2 ; augmentation/diminution de la saturation (« accentue » les couleurs, les rend plus vives) ; stéganographie ; cacher une image dans les bits de poids faibles d'une autre image.

VI Pour la prochaine fois

Exercice n°14 Faire une fonction `negatifCouleur(im:"image")->"image"` qui réalise le négatif pour une image couleur.

Exercice n°15 Faire une fonction `miroir_vertical(im:"image")->"image"` qui réalise la symétrie par rapport à une droite horizontale passant par le milieu de l'image (image en niveau de gris demandé, si vous souhaitez le faire pour une image couleur, vous pouvez).

Exercice n°16 Faire une fonction `rotation90droite(im:"image")->"image"` qui réalise la rotation à droite (image en niveau de gris demandé, si vous souhaitez le faire pour une image couleur, vous pouvez).

Exercice n°17 Faire une fonction `RVBvers_gris(im:"image couleur")->"image noir et blanc"` qui prend en argument une image en couleur et qui renvoie une image correspondante en noir et blanc. On pourra simplement faire la moyenne des différentes couleurs pour chaque pixel [10].

10. On fait normalement une moyenne pondérée pour tenir compte de la sensibilité de l'œil plus importante dans le vert que dans le rouge que dans le bleu.