TP 8 - Images

Le but de ce TP est de manipuler des images via des algorithmes très classiques afin de comprendre le type de structure de données associé ainsi que les systèmes de coordonnées utilisés.

1 Préparation de l'environnement

Pour travailler sur une image avec Python, il faut commencer par ouvrir cette image dans Pyzo.

1. Commencer par créer un dossier TP8-Images dans votre Espace Personnel.
2. Y placer les 3 fichiers : le script Python (py) et les deux images "dauphinNB.png" et "meduseSombre.png".
3. Afin de pouvoir ouvrir les images se trouvant dans le dossier du TP, il faut se placer dans le bon répertoire :
```
#On se place dans le bon répertoire.
import os
os.chdir("adresse/de/votre/dossier")
print(os.getcwd()) #Affiche le répertoire dans lequel on se trouve pour vérification.
```
4. Il faut aussi importer les modules `Image` et `numpy` afin de pouvoir effectivement ouvrir et traiter les images :
```
from PIL import Image
import numpy as np  #Tableaux à 3 dimensions hauteur x largeur x profondeur.
```
5. Pour ouvrir et afficher une image présente dans le répertoire, on utilise les commandes suivantes :
```
imageDauphin = Image.open("dauphinNB.png") #Lecture du fichier.
imageDauphin.show()  #Affiche l'image.
tabImageDauphin = np.array(imageDauphin)  #création d'un tableau contenant l'image.
```

En compilant ces premières lignes dans le script, l'écran doit afficher l'image d'un dauphin.

2 Implémentation d'une image

En Python, une image est représentée par une matrice. Chaque élément de cette matrice est un point de l'image correspondant à **un pixel**. Chaque pixel est repéré à l'aide de ses coordonnées (i, j) telles que :

- le pixel de coordonnées $(0, 0)$ se situe en haut à gauche de l'image ;
- i représente la ligne du pixel, numérotées de haut en bas
- et j représente la colonne du pixel, numérotées de gauche à droite.

La structure de données sous-jacente est celle de **tableaux numpy**. Un tableau numpy est un tableau à 3 dimensions : la hauteur (nombre de lignes), la largeur (nombre de colonnes) et la profondeur. En effet, il existe plusieurs méthodes de codages d'images. Par exemples

- le codage dit "rouge, vert, bleu" (RGB) dans lequel chaque pixel est représenté par sa couleur codée par une liste de taille 3 indiquant la proportion de "rouge", de "vert" et de "bleu" (entier compris entre 0 et 255) dans cette couleur ;
- le codage en noir et blanc, un pixel est alors représenté par sa nuance de gris soit un entier compris entre 0 (pixel noir) et 255 (pixel blanc).

L'accès à un élément du tableau numpy s'effectue de la même manière que sur un tableau "classique" : `tabImage[i][j]` représente le pixel de coordonnées (i, j). De même, la longueur du tableau s'obtient avec la fonction `len`.

Question 1. Quelles informations obtient-on avec les commandes `len(tabImageDauphin)` et `len(tabImageDauphin[0])` ?
Comparer avec la commande `tabImageDauphin.shape`.

Question 2. Extraire les valeurs de gris des pixels de chaque coin de l'image du dauphin.
Vous devez trouver : 114 pour le pixel en haut à gauche, 84 pour le pixel en bas à gauche, 140 pour le pixel en bas à droite et 109 pour le pixel en haut à droite.

3 Algorithmes classiques

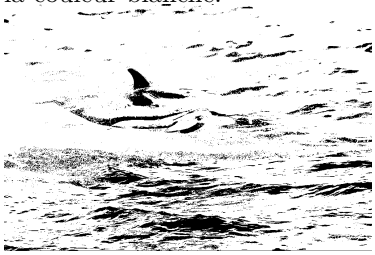
Les algorithmes qui suivent prennent tous en paramètre un élément de type `image`. Pour répondre, il faut utiliser la structure suivante :

```
def fonctionDemandee(image) :
    tabImage = np.array(image) #on transforme l'image en tableau
    ligne,colonne = tabImage.shape #on récupère les dimensions de l'image
    #Algorithme à réaliser
    return Image.fromarray(tabImage) #on transforme le tableau en image
```

Pour tester les fonctions, il suffira d'afficher l'image retournée par l'algorithme avec la commande `.show()` :
`nouvelleImage = fonctionDemandee(image)`
`nouvelleImage.show()`.

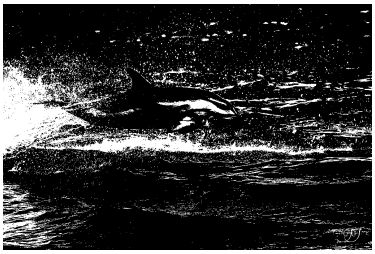
3.1 Binarisation

La **binarisation** d'une image consiste à modifier les valeurs des pixels en leurs affectant la couleur blanche ou noire selon un seuil. Si la couleur du pixel est sous le seuil, on lui affecte la couleur noire, sinon on lui affecte la couleur blanche.



← Image du dauphin binarisée avec un seuil de 75.

Image du dauphin binarisée avec un seuil de 150. →



Question 3. Ecrire une fonction `binariser()` prenant en paramètre une image `image` et un seuil `s`, et retournant la nouvelle image binarisée selon le seuil `s`.

3.2 Négatif

Le **négatif** d'une image est obtenu en "inversant" la valeur de gris de chaque pixel. Pour inverser la couleur d'un pixel, on applique la fonction f qui à tout entier $0 \leq n \leq 255$ associe $255 - f(n)$.

Négatif de l'image du dauphin. →

Question 4. Ecrire une fonction `negatif` prenant en paramètre une image `image` et retournant une nouvelle image représentant le négatif de l'image.

3.3 Retourner

On veut retourner une image selon un axe vertical (ce serait la même chose pour un axe horizontal). L'idée est d'inverser deux pixels sur la même ligne de taille n tel que le pixel de la colonne j est permuté avec celui de la colonne $n - 1 - j$.

Image du dauphin retournée verticalement. →

Question 5. Ecrire une fonction `retourner` prenant en paramètre une image `image` et retournant une nouvelle image représentant l'image retournée selon l'axe vertical.

3.4 Rotation

On veut effectuer une rotation horaire ou anti-horaire d'une image.

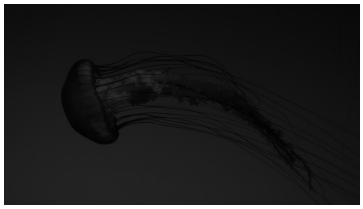
Image du dauphin après rotation horaire de 90°. →

Question 6. Ecrire une fonction `pivoter` prenant en paramètre une image `image` et retournant une nouvelle image représentant l'image pivotée de 90° dans le sens horaire.

Pour créer un nouveau tableau numpy de taille (n, m) contenant des 0, on utilise la commande `tabImage = np.zeros((n,m),dtype=np.uint8)`.

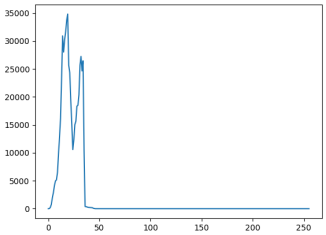
3.5 Augmentation du contraste

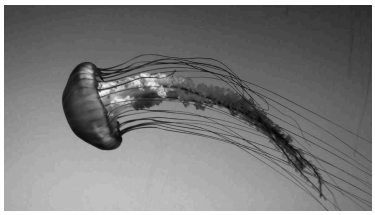
Normaliser une image c'est utiliser toute le spectre possible de nuances de gris. Pour visualiser les nuances utilisées dans l'image, on trace un histogramme des nuances de gris, obtenu en comptant les pixels correspondant à une même valeur de gris. Normaliser l'image consiste à augmenter le contraste en étirant l'histogramme à l'aide de la formule $NouvelleValeur = \lfloor \frac{ValeurActuelle - Min}{Max - Min}.255 \rfloor$ où Min et Max sont les valeurs minimale et maximale dans l'ensemble des pixels de l'image.



← Image de la méduse.

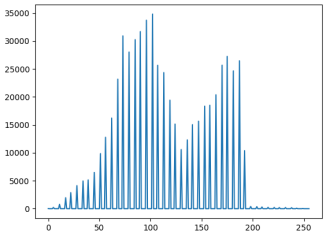
Histogramme pour l'image de la méduse. →





← Image de la méduse normalisée.

Histogramme pour l'image de la méduse normalisée. →



Question 7. Ecrire une fonction `CalculHistogramme` prenant en paramètre une image `image` et retournant une liste dont l'indice `i` contient le nombre de pixels ayant pour valeur `i` dans l'image.

Question 8. Ecrire une fonction `normaliserImage` prenant en paramètre une image `image` et retournant une nouvelle image normalisée.