Lycée Loges

Chapitre 8

Manipulations d'images TP

Simon Dauguet
simon.dauguet@gmail.com

13 mars 2025

Pour tester vos programmes, il faut utiliser les fonctions du fichiers **CPGE-traitementImage.py**. Vous commencerez donc par copier les codes du fichiers **CPGE-traitementImage.py** et les mettre au début de votre script.

Pour récupérer la matrice depuis une image **.bmp** ou créer une image **.bmp** depuis une matrice, vous pourrez vous inspirer des lignes suivantes :

```
mat = photo2matriceXX( "./image.bmp" )       # XX = NB ou C, ouverture de l'image
# traitement sur la matrice mat ...
matrice2photoXX( mat, "./newImage.bmp" )     # XX = NB ou C, sauvegarde de la matrice
afficher("newImage.bmp")                         # affiche l'image correspondant au fichier "newImage.bmp
```

Évidemment, il faudra commencer par importer dans **Capytale** les images en questions.

1 Manipulations matricielles

Exercice 1 :
On va commencer par coder quelques manipulations matricielles élémentaires :

1. Faire une fonction **Hadamard(mat1:list, mat2:list) -> list** qui effectue le produit de Hadamard (*i.e.* produit scalaire canonique) des deux matrices **mat1** et **mat2** de même taille.
2. Faire une fonction **prodMat(mat1:list, mat2:list) -> list** qui effectue le produit de matrice (le vrai) des deux matrices **mat1** et **mat2** ayant les bonnes tailles.

2 Manipulations d'images

Exercice 2 :
Proposer une fonction **inversionNB(mat: list) -> list**, qui renvoie une matrice où les proportions de blanc et de noir ont été interverties pour chaque pixel. exemple : si un pixel a $100/255$ de blanc, le nouveau pixel en aura $155/255$. Tester sur l'image **cpge-champigris.bmp**.

Exercice 3 :
On souhaite modifier la proportion de blanc et de noir d'une image selon une fonction $f : [\![0; 255]\!] \to [\![0; 255]\!]$.

1. Proposer une fonction `modificationNB(mat: list, f: 'function'= (lambda x: x)) -> list`, qui renvoie une nouvelle matrice où chaque pixel (i,j) est l'image du pixel (i,j) par la fonction f (par défaut l'identité).
2. Soit la fonction $f : x \mapsto \lfloor x^2/k \rfloor$. Quelle valeur de $k \in \mathbb{N}^*$ choisir pour que $f([\![0;255]\!]) = [\![0;255]\!]$? Tester la fonction `modificationNB` avec cette fonction f.

   Cette fonction entraîne-t-elle un éclaircissement ou un assombrissement de l'image ?
3. Définir la fonction $g : x \mapsto \lfloor k\sqrt{x} \rfloor$ en fixant $k \in \mathbb{R}_+^*$ tel que $g([\![0;255]\!]) = [\![0;255]\!]$. Quelle conséquence observe-t-on sur l'image ?
4. On peut également définir la fonction f définie par $f(x) = \begin{cases} x + 0,5(x-128) & 0 \leq x \leq 255 \\ 0 & x + 0,5(x-128) < 0 \\ 255 & x + 0,5(x-128) > 255 \end{cases}$.

   Tester et commenter.

Exercice 4 :
L'image `cpge-mystereDecale.bmp` semble de guingois après un décalage accidentel de -400 pixels horizontalement et verticalement. Proposer une fonction qui annule ce décalage.

Exercice 5 :
On considère le code suivant :

```
1  def mystereV(mat: list) -> list:
2      n, m = len(mat), len(mat[0])
3      nmat = [ [0]*m for i in range( n )  ]
4      for i in range( n ):
5          for j in range( m ):
6              nmat[i][j] = mat[i][-1-j]
7      return nmat
```

1. Que permet de faire ce code ? Vous pourrez l'appliquer à l'image `cpge-pereRenneNB.bmp` après transformations.
2. Proposer une suite d'instructions (définir une fonction `retourneAxeH(mat:list) -> list`) qui permette d'obtenir l'image de `cpge-pereRenneNB.bmp` retournée selon un axe horizontal.
3. Comment faire une symétrie selon la diagonale $y = x$?

Exercice 6 :
Pour restaurer une image contenant de petites aberrations, on peut flouter l'image en appliquant un filtre qui, pour chaque pixel, donne la valeur moyennes des neuf pixels attenants (lui-compris).

1. Quelle matrice de convolution permet cela ?
2. Proposer une procédure `recupImage(mat: list) -> list`, qui renvoie la matrice après floutage, où chaque pixel (i,j) par la valeur moyenne de ses voisins.
3. Restaurer ainsi l'image `cpge-chatelBruite.bmp`.

Exercice 7 :
L'écriture sympathique est une méthode simple pour cacher un message en écrivant avec une encre invisible, du moins proche d'une couleur proche de celle du fond. Un procédé similaire a été effectué sur l'image

`cpge-pointsCaches.bmp`, dans laquelle des points de couleur presque noire ont été ajoutés. On peut détecter ces points en effectuant une convolution à l'aide d'une matrice $\begin{pmatrix} -a & -a & -a \\ -a & 8a & -a \\ -a & -a & -a \end{pmatrix}$ avec $a \in \mathbb{N}^*$.

1. Appliquer, à la main, cette convolution au coefficient $(1,1)$, $(1,2)$ et $(1,3)$ (avec des numéros de lignes et colonnes "à la python") de la matrice $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$.

   Commenter le résultat.
2. Combien de points presque noirs sont contenus dans l'image `cpge-pointsCaches.bmp` ?
3. Quel est le message caché ?

Exercice 8 (Contours) :

On va ici, essayer de s'entraîne à faire apparaître les contours d'une image en noir et blanc.

1. Coder une fonction `transpo(mat:list) -> list` qui renvoie la transposée de la matrice `mat`.
2. Coder une fonction `prod2conv(mat1:list, mat2:list) -> list` effectuant le produit de convolution de `mat1` par `mat2` (la matrice `mat2` étant plus petite que `mat1`).
3. Faire une fonction `Grad(mat:list, seuil:int) -> list` qui effectue la détection de contour à la matrice `mat` d'une image par la méthode du gradient, en appliquant en plus un filtre seuil, et qui utilise la fonction `tranpso`.
4. Effectuer des tests sur la photo `cpge-champigris.bmp` avec différents seuils.
5. D'autres types de filtres sont possibles pour la détection de contour. Le masque de Prewitt consiste à reprendre la méthode du gradient, mais en appliquant la matrice $\frac{1}{3}\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$ selon x, et la transposée de cette matrice selon y. Faire une fonction `Prewitt(mat:list) -> list` qui effectue la détection de contour avec le masque de Prewitt sur la matrice `mat`. L'appliquer à l'image `cpge-champigris.bmp`.
6. Une autre version de la détection de contour peut être fait avec le masque de Sobel qui consiste à appliquer les trois produits de convolutions avec les matrices par les matrices

$$\frac{1}{4}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \qquad \frac{1}{4}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \qquad \frac{1}{4}\begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

   Faire une fonction `Sobel(mat:list) -> list` qui effectue, sur le même principe que le gradient, la détection de contour avec le masque de Sobel.