TP 8 : Redimensionnement

Mots clés : *matrices de pixels*

Image originale | Redimensionnement naïf | Redimensionnement "intelligent"

Table des matières

I Introduction

I.1 Objectifs

Dans ce TP, nous allons implémenter et tester différentes méthodes permettant de **réduire une image** dans la largeur.

I.2 Consignes

Vous travaillerez dans le fichier *TP8_ redimensionnement.py*[1]. Pensez à :

- bien lire les commentaires des fonctions déjà écrites,
- tester au fur et à mesure les fonctions. Trois images sont fournies pour cela : *flamantsNB.jpg*, *guepiersNB.jpg* et *rueNB.jpg*.

> Dans tout ce TP il n'y a RIEN à écrire dans la console.
> Pour tester les fonctions, il suffit d'enlever les # devant les lignes de commande et exécuter le fichier.

Les listes rencontrées sont, sauf mention du contraire, des listes non vides d'entiers.

Une image I est codée sous la forme d'une liste de listes d'entiers pouvant être vue comme un tableau à deux dimensions. On notera :

- `h` le nombre de lignes de l'image (pour `height`)

1. A télécharger sur http ://alicenolot.free.fr

- `w` le nombre de colonnes de l'image (pour width)

Par convention, le pixel d'indice $(0,0)$ correspond à celui situé en haut à gauche de l'image. Chaque pixel de coordonnées $(i,j) \in [\![0,h-1]\!] \times [\![0,w-1]\!]$ a une valeur entière $I_{i,j}$ comprise entre 0 et 255, correspondant à son niveau de gris (0 pour noir, 255 pour blanc).

Des fonctions permettant de charger et d'afficher une image vous sont présentées dans le fichier.

II Pour s'échauffer ...

❒ Q1. Une fois avoir placé le bon répertoire dans `os.chdir` où se trouvent les images, exécuter le script. Si l'image avec des oiseaux s'affichent alors c'est que tout va bien !

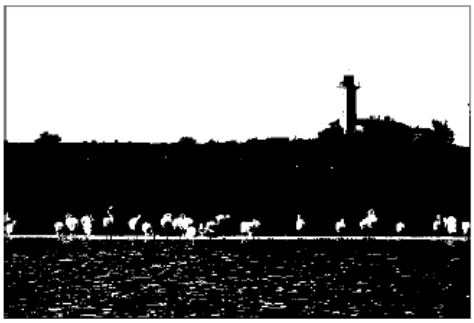
❒ Q2. Comprendre (faire un dessin d'un tableau) la fonction `dim(img)` qui prend en argument une image et renvoie le couple d'entiers `(w, h)`.
Dans la suite, on pourra utiliser `(w, h) = dim(img)` pour récupérer les dimensions de l'image `img`.

❒ Q3. En complétant les fonctions correspondantes, afficher, à partir de l'image *flamantsNB.jpg*, l'image

a) avec contraste inversé : le blanc devient noir et vice-versa (a),

b) avec seuil à la valeur 128 : les pixels de valeur inférieure stricte à 128 deviennent noir, les autres blancs (b),

(a)



(b)

❒ Q4. A quelle transformation sur l'image `img` s'attendre lorsque l'on exécute la fonction suivante ?

```
def mystere(img: list) -> list:
    w, h = dim(img)
    L = []
    for j in range(w):
        ligne = []
        for i in range(h):
            ligne.append(img[i][j])
        L.append(ligne)
    return L
```

III Réduction de largeur naïve

❒ Q5. Compléter la fonction `reduction_moitie_ligne(l)` qui prend en argument une liste `l` de longueur paire $2n$ et contenant des entiers $a_0, a_1, \ldots, a_{2n-1}$ et qui renvoie la liste de longueur n contenant : $(a_0+a_1)//2, (a_2+a_3)//2, \ldots, (a_{2n-2}+a_{2n-1})//2$.

```
>>> reduction_moitie_ligne([5, 4, 1, 3, 2, 8])
[4, 2, 5]
```

❒ Q6. Compléter la fonction `reduction_moitie_image(img)` qui prend en argument une image ayant une largeur paire et la modifie en appliquant à chaque ligne l'opération décrite dans la question précédente. L'image est modifiée par effet de bord et la fonction ne renvoie rien. Que constate-t-on ?

IV Réduction plus intelligente

IV.1 Energie d'un pixel

Pour $(i,j) \in [\![1, h-2]\!] \times [\![1, w-2]\!]$, l'énergie du pixel (i,j) intérieur à l'image est définie par :

$$e_{i,j} = \left|\frac{I_{i+1,j} - I_{i-1,j}}{2}\right| + \left|\frac{I_{i,j+1} - I_{i,j-1}}{2}\right|.$$

Cette quantité est d'autant plus petite que le pixel est dans une zone uniforme de l'image, d'autant plus grande lorsqu'il est dans une zone de forte variation de niveau de gris (par exemple au niveau d'un contour).

Lorsqu'on a affaire à un pixel du bord de l'image, cette formule doit être adaptée et une manière cohérente est la suivante :

- si $i = 0$ alors la partie $\left|\frac{I_{i+1,j}-I_{i-1,j}}{2}\right|$ est remplacée par $|I_{1,j} - I_{0,j}|$;
- si $i = h-1$ alors elle est remplacée par $|I_{h-1,j} - I_{h-2,j}|$;
- si $j = 0$ alors la partie $\left|\frac{I_{i,j+1}-I_{i,j-1}}{2}\right|$ est remplacée par $|I_{i,1} - I_{i,0}|$;
- si $j = w-1$ alors la partie $\left|\frac{I_{i,j+1}-I_{i,j-1}}{2}\right|$ est remplacée par $|I_{i,w-1} - I_{i,w-2}|$.

❒ Q7. Comprendre rapidement la fonction `energie(img)`, qui prend un argument une image et renvoie une nouvelle image correspondant à son énergie (les pixels n'auront plus nécessairement une valeur entière).

Image originale | Image des énergies

IV.2 Réduction ligne par ligne

Pour réduire la largeur d'une image, on peut supprimer un pixel d'énergie minimale dans chaque ligne.

❒ Q8. Compléter la fonction `enlever(L, i)` qui prend en argument une liste `L` et un indice `i` compris entre 0 et `len(L)-1` et renvoie une nouvelle liste de taille `len(L)-1` correspondant à `L` dont l'élément d'indice `i` a été supprimé. On pourra utiliser des extractions de tranches `:` et concaténations de listes.

❒ Q9. Compléter la fonction `indice_min(L)` qui prend en argument une liste `L` et renvoie un indice pour lequel apparaît la valeur minimale de la liste.

❒ Q10. Compléter la fonction `reduction_par_ligne(img)` qui prend en argument une image et la modifie en supprimant un pixel d'énergie minimale dans chacune de ses lignes.

❒ Q11. Compléter la fonction `itere_reduction_par_ligne(img,n)` qui prend en arguments une image et un entier $n > 0$, et applique n fois la procédure décrite dans la question précédente à `img`.

❒ Q12. Réduire la largeur de l'image *flamantsNB.jpg* de 200 pixels (ou 50 pixels si l'ordinateur met trop de temps). Expliquer pourquoi le résultat est visuellement peu satisfaisant.

IV.3 Réduction par colonne

Pour remédier au problème précédent, on souhaite lors d'une itération enlever uniquement des pixels situés sur une même colonne.

❒ Q13. Compléter la fonction `meilleure_colonne(e)` qui prend en argument une image d'énergies et renvoie un indice de colonne d'énergie minimale, où l'énergie d'une colonne est définie comme la somme des énergies de ses pixels.

❒ Q14. Compléter la fonction `reduction_meilleure_col(img)` qui prend en argument une image et qui en supprime une colonne d'énergie minimale.

❒ Q15. Compléter enfin la fonction `itere_reduction_meilleure_col(img, n)` qui applique la méthode ci-dessus pour réduire l'image `img` de `n` pixels dans sa largeur.

❒ Q16. Tester une réduction de 100 pixels et expliquer pourquoi les résultats obtenus sont corrects sur les images *guepiersNB.jpg* et *flamantsNB.jpg* mais pas sur *rueNB.jpg*.

V Aller plus loin : Seam carving

L'idée de l'algorithme de *Seam carving*, introduit en 2007 par S.Avidan et A.Shamir est d'assouplir un peu la contrainte de réduction colonne par colonne.

On définit un *chemin* de pixels comme une suite de pixels connectés (verticalement ou en diagonale) dont le premier appartient au bord haut de l'image, le dernier au bord bas, et contenant exactement un pixel par ligne de l'image.

L'énergie d'un chemin est la somme des énergies des pixels le constituant.

Voici par exemple un chemin d'énergie 6 dans une image des énergies e de 4×4 pixels.

1	**1**	0	3
4	1	**2**	4
1	2	**2**	1
4	**1**	1	0

Pour réduire la largeur d'une image d'un pixel, on souhaite trouver puis enlever un chemin d'énergie minimale.

On définit un *chemin partiel* comme un chemin, sans la contrainte que son dernier pixel atteigne le bord bas de l'image. Afin de calculer un chemin minimal, on va commencer par calculer, pour chaque pixel, l'énergie minimale E d'un chemin partiel terminant sur ce pixel.

Par exemple, pour notre image des énergies e, on obtient le tableau E suivant :

1	1	0	3
5	1	2	4
2	3	3	4
6	3	4	3

❒ Q17. Calculer à la main E pour l'image des énergies e suivante, puis trouver un chemin d'énergie minimale.

2	1	1	0
3	3	2	2
2	0	1	2

❒ Q18. Expliquer comment construire E à partir de e, en procédant ligne par ligne.

❒ Q19. Une fois E construit, comment en déduire un chemin d'énergie minimale ?

❒ Q20. Implémenter tout cela. Il sera judicieux d'écrire plusieurs fonctions, et vous pourrez utiliser la fonction `min` permettant de calculer le minimum des éléments d'une liste (de complexité linéaire en la longueur de la liste).
Enfin, compléter la fonction `seam_carving`, qui prend en arguments une image `img` et un entier `n`, et modifie l'image en enlevant successivement n chemins minimaux.
Testez !