ITC

Sauf mention du contraire, les fonctions et procédures de ce devoir n'effectuent pas d'effets de bord : les paramètres passés en arguments ne sont pas modifiés.

Tiré d'un travail de Mickaël Péchaud.

Présentation

FIGURE 1 – Image originale, redimensionnement naïf, et redimensionnement ńintelligentż

Nous implémentons différentes méthodes permettant de **réduire une image** en évitant les pertes d'information.

Une image I est codée sous la forme d'une liste de listes d'entiers, qui peut être vue comme un tableau à deux dimensions. On note h le nombre de lignes de l'image, et w son nombre de colonnes (respectivement pour **h**eight et **w**idth). Le pixel d'indice $(0, 0)$ correspond par convention au pixel 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).

Toute image en niveau de gris peut être vue comme une matrice d'entiers et, réciproquement, toute matrice d'entiers entre 0 et 255 peut s'afficher comme une image :

```
import matplotlib.pyplot as plt
# mat est une matrice d'entiers définie ailleurs
plt.imshow(mat,cmap='gray',clim=(0,255))
plt.plot()
```

Outre celle des guêpiers, on considère aussi la photo de flamants **flamantsNB.jpg** et celle de rue **rueNB.jpg** de la figure 2 comme images de référence dans ce qui suit. Pour information, si ce fichier est dans le répertoire courant, on le récupère sous forme de liste de listes d'entiers en écrivant :

```
import matplotlib.image as mpimg
flam = (mpimg.imread('./flamantsNB.jpg')).tolist()
rue = (mpimg.imread('./rueNB.jpg')).tolist()
```

Prise en main

Les arguments passés en paramètres des fonctions ne sont pas modifiés.

Q.1 Compléter la fonction `dim(img:list)->tuple`, qui prend en argument une image et renvoie le couple d'entiers (w, h), w étant la largeur de l'image, et h sa hauteur (en nombre de pixels).

Q.2 On effectue des changements sur l'image :

(a) Écrire la fonction `inv(img:list)->list` qui inverse les contrastes (0 devient 255 et 255, 0) (voir figure 3a).

(b) Écrire la fonction `seuil(img:list,s)->list` qui seuille les pixels : tout ce qui est plus grand que s est affiché en blanc, et ce qui est plus petit, en noir (voir figure 3b) .

FIGURE 2 – Deux photos : des flamants et une rue

(c) Écrire la fonction `symh(img:list)->list` qui inverse l'ordre des lignes de l'image (voir figure 3c) .

(d) Écrire la fonction `symv(img:list)->list` qui inverse l'ordre des colonnes de l'image (voir figure 3d).

(a)

(b)

(c)

(d)

Réduction de largeur naïve

Q.3 Écrire la fonction `reduction_moitie_ligne(l:list) -> list`, qui prend en argument une liste `l` de longueur paire $2n$ contenant des entiers $a_0, a_1, a_2, \dots a_{2n-1}$, et renvoie la liste de longueur n contenant $(a_0 + a_1)//2$, $(a_2 + a_3)//2$, etc.

Q.4 Écrire la fonction `reduction_moitie_image(img:list) -> None` 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 (la fonction modifie l'image par effet de bord, et ne renvoie rien).
Quelle est sa complexité en fonction du nombre de lignes h et du nombre de colonnes w de l'image ?

On observe figure 3 l'application de cette fonction à la matrice décrivant les flamants.

Q.5 Décrire en quelques lignes une méthode permettant de réduire la largeur d'une image d'un facteur autre que moitié (par exemple 1/3 pour fixer les idées). Il n'est pas demandé d'implémenter la méthode correspondante.

Nous allons maintenant développer des algorithmes plus ńintelligentsż, dans la mesure où ils prennent en compte le contenu de l'image. On commence par déterminer une mesure d'ńimportanceż des pixels - que nous appellons *énergie* dans la suite.

FIGURE 3 – Réduction de moitié

Calcul de l'énergie d'un pixel

Si l'on note $I_{i,j}$ le niveau de gris du pixel de coordonnées $(i,j) \in [\![h-1,\times]\!][\![0,w-1]\!]$, l'*énergie* $e_{i,j}$ du pixel i,j intérieur à l'image est définie comme la somme $\Delta_x + \Delta_y$ telle que :

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

Il faut adapter la formule ci-dessus à un pixel situé sur un bord de l'image : on remplace dans la formule le ou les voisins manquants par le pixel courant (i,j). Par exemple, si $i = 0$, $\Delta_x = \left|\frac{I_{1,j} - I_{0,j}}{2}\right|$.

Cette quantité $e_{i,j}$ est d'autant plus petite que le pixel est dans une zone uniforme de l'image, et d'autant plus grande qu'il est dans une zone de forte variation du niveau de gris (par exemple au niveau d'un contour) [1].

Q.6 Écrire la fonction `energie(img:list) -> list`, qui prend en argument une image, et renvoie une nouvelle image correspondant à son énergie (les pixels n'auront plus nécessairement une valeur entière). Quelle est la complexité de votre fonction ?

Image originale, image des énergies

Réduction ligne par ligne

Maintenant que nous disposons d'une image et de son énergie, pour réduire la largeur d'une image d'un pixel, nous allons simplement enlever un pixel d'énergie minimale dans chaque ligne.

Nous avons pour cela besoin de quelques fonctions élémentaires de manipulations de listes.

Q.7 Écrire la fonction `enlever(l:list,i:int)->list` qui prend en argument une liste ℓ et un indice `i` licite de ℓ. La fonctiont renvoie une nouvelle liste correspondant à ℓ dont l'élément d'indice `i` a été supprimé. La liste ℓ n'est pas modifiée.

1. Il s'agit en fait d'une version discrétisée de $\|\nabla I\|_1 = \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right|$, qui est une norme du gradient de l'image.

Q.8 Écrire la fonction `indice_min(l:list) -> int`, qui prend en argument une liste ℓ et renvoie une position de la valeur minimale de la liste. Si le minimum apparaît en plusieurs indices, la fonction renvoie le premier d'entre eux.

Q.9 Écrire la fonction `reduction_ligne_par_ligne(img:list)->None`, qui prend en arguments une image `img`, et la modifie en supprimant un pixel d'énergie minimale dans chacune de ses lignes (il y a des effets de bord).

Q.10 (a) Écrire la fonction `itere_reduction_ligne_par_ligne(img:list)->None`, qui prenant en arguments une image `img` et un entier $n > 0$, applique n fois la procédure décrite dans la question précédente à `img`.

(b) Quelle est sa complexité ?

(c) Sur la figure 4, on observe le résultat de cette fonction appliquée à la photo des flamants et $n = 200$: il est peu satisfaisant.
Expliquez qualitativement, de façon brève, les distorsions observées.

FIGURE 4 – réduction de 200 lignes

Réduction par colonne

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

Q.11 Écrire la fonction `meilleure_colonne(e:list)->int`, qui prend en argument une matrice d'énergies `e`, et renvoie un indice de colonne d'énergie minimale (l'énergie d'une colonne étant définie comme la somme des énergies de ses pixels).

Q.12 Écrire la fonction `reduction_meilleure_colonne(img:list)->None`, qui prend en argument une image, et en supprime une colonne d'énergie minimale. Elle réalise des effets de bord.

Q.13 En déduire une fonction `itere_reduction_meilleure_colonne(img:list, n:int)->None`, qui applique la méthode décrite ci dessus pour réduire l'image `img` de `n` pixels dans sa largeur.
Quelle est sa complexité ?

Q.14 On constate que les résultats obtenus sont ńbonsż sur les images **guepiersNB.jpg** et **flamantsNB.jpg**, mais pas sur **rueNB.jpg** (observer par exemple la voiture en bas de l'image figure 5 ci-dessous). Expliquez qualitativement pourquoi on observe la disparition de certains détails « intéressant » dans la photo de rue mais pas dans celle des guêpiers..

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.

Sur la figure 6, on observe un chemin d'énergie 6 dans une image des énergies (notée e) de 4×4 pixels.

FIGURE 5 – Deux photos : des guêpiers et une rue

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

FIGURE 6 – Image d'énergie notée e et chemin d'énergie 6

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 d'un chemin partiel terminant sur ce pixel.

Par exemple, pour notre image des énergies e de la figure 6, on obtient le *tableau des énergies minimales* E indiqué figure 7. Les tailles de e et E étant similaires, on a coloré les pixels que parcourerait un chemin d'enérgie minimale dans e.

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

FIGURE 7 – Tableau des énergies minimales et itinéraire emprunté par un chemin d'énergie minimale.

Q.15 Calculer à la main le tableau des énergies minimales E pour la matrice d'énergies e suivante, puis trouver un chemin d'énergie minimale.

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

Q.16 Expliquez comment construire E à partir de e, en procédant ligne par ligne.

Q.17 Une fois E construit, comment en déduire un chemin d'énergie minimale ?

Une fois pour toute dans ce qui suit, on suppose que `img` est de dimension $w \times h$ (donc `e` et `E` aussi).

Q.18 Écrire la fonction `ajouter_ligne(e:list,E:list)->list` qui prend en paramètre une matrice d'énergie `e` et le tableau non vide (mais pas encore complet) des énergies minimales `E`. La fonction renvoie une liste qui a vocation à être la prochaine ligne de E.

Le code suivant :

```
e = [[1,1,0,3],[4,1,2,4],[1,2,2,1],[4,1,1,0]]
E= [[1,1,0,3]]
print(ajouter_ligne(e,E))
```

produit l'affichage

```
[5, 1, 2, 4]
```

Donner la complexité.

Q.19 Écrire la fonction `construire_E(e:list)->list` qui renvoie le tableau des énergies minimale E à partir de la matrice des énergies e.

Le code suivant :

```
e = [[1,1,0,3],[4,1,2,4],[1,2,2,1],[4,1,1,0]]
 E=construire_E(e)
for l in E:
    print(l)
```

produit l'affichage

```
[1, 1, 0, 3]
[5, 1, 2, 4]
[2, 3, 3, 3]
[6, 3, 4, 3]
```

Donner la complexité de `construire_E`.

Q.20 Écrire la fonction `chemin_minimal(E:list)->list` qui calcule un chemin d'énergie minimale à partir du tableau des énergies minimales. Avec E de l'exemple précédent, `chemin_minimal(E)` renvoie `[(0, 2), (1, 1), (2, 0), (3, 1)]`. Le chemin est donné comme la liste des coordonnées des pixels en commençant par celui du haut.
Donner la complexité.

Q.21 Écrire la fonction `reduction_meilleure_energie(img:list)->None` qui prend en paramètre une image, calcule un chemin d'energie minimale dans cette image et retire dans l'image les pixels de ce chemin. La fonction réalise des effets de bord : `img` est modifiée.
Donner la complexité.

Q.22 Écrire la fonction `itere_reduction_meilleure_energie(img:list, n:int)->None` qui itère la fonction précédente n fois.
Donner la complexité.

L'image obtenue par cette fonction appliquée aux guêpiers avec $n = 100$ est la 3ème photo sur la droite tout en haut de ce devoir.