TP7 - Traitement d'images avec `numpy`

Il existe de nombreuses bibliothèques de traitement d'images (`opencv, pillow, PIL, ...`), Mais `numpy` et `matplotlib` suffisent bien souvent. On importera ces modules par les instructions usuelles :

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

Dans ce TP, nous nous intéresserons particulièrement à l'algorithme *Seam Carving*, qui permet de réduire « intelligement » des images.

Objectifs

- Réviser la manipulation des tableaux `numpy` et les slicings ;
- Lire et afficher des images ;
- Comprendre un algorithme et l'implémenter en Python ;

1 Manipulations de base

1.1 Lecture

Question 1. Récupérer une grande image en couleur avec un moteur de recherche, et la sauvegarder dans votre répertoire de travail. L'ouvrir avec `Paint` et la sauvegarder au format **PNG** : c'est le format reconnu par `matplotlib`. L'image devra être stockée dans le même dossier que le fichier Python contenant vos codes.

Question 2. Convertir l'image en un tableau `numpy` avec l'instruction

```
im = plt.imread(nom_fichier)
```

où `nom_fichier` désigne le nom du fichier (avec l'extension `.png`) contenant votre image (sous la forme d'une chaîne de caractères). N'oubliez pas de sélectionner sous Spyder le répertoire où se trouve votre image afin qu'il puisse la trouver (icône en forme de dossier en haut à droite de l'écran).

Visualiser enfin l'image avec l'instruction `plt.imshow(im)`.

Question 3. Vérifier son nombre de dimension (`im.ndim`), sa taille (`im.shape`), son nombre de couleurs, le type des éléments du tableau (`im.dtype`). Expliquer.

Question 4. Isoler les trois canaux rouge, vert, bleu dans trois tableaux nommés `imr,imv,imb` à l'aide de slicings. Les visualiser dans trois fenêtres graphiques différentes (créées avec `plt.figure()`). On peut fermer toutes les fenêtres ouvertes avec l'instruction `plt.close("all")`. Par défaut, les tableaux à deux dimensions sont affichés en fausses couleurs. On peut choisir la palette avec l'argument `cmap` : `cmap="Greens_r"` pour une palette de verts par exemple. Une liste des palettes de couleurs disponibles est fournie par `dir(plt.cm)` (par exemple, on dispose également des palettes `"Reds_r"` et `"Blues_r"`).

Certaines images stockent en plus des trois canaux rouge, vert et bleu, un canal de transparence que l'on supprimera pour continuer : si `im` désigne une telle image, `im[:,:,:3]` permet de conserver uniquement les trois canaux souhaités. On suppose donc dans la suite que `im.shape[2] = 3`.

1.2 Image en niveau de gris

Question 5. Écrire une fonction `ng(im)` qui génère une image en niveau de gris (tableau à 2 dimensions), en faisant la moyenne des 3 canaux. On ne fera pas de boucle, on utilisera plutôt la fonction `np.mean`, dont on consultera l'aide (raccourci Ctrl-I sous Spyder, ou `help(np.mean)` dans la console). On s'intéressera plus particulièrement à l'argument `axis` de la fonction. Visualiser le résultat avec la palette de couleurs `cmap="Greys_r"`.

1.3 Réduction

□ **Question 6.** Écrire une fonction `reduction(IM,h,l)` qui réduit une image `IM` **en niveaux de gris** de taille $H \times L$ en une image `im` de taille $h \times l$. Seuls les pixels de coordonnées $((i \times H)//h, (j \times L)//l)$ de l'image `IM` sont conservés, i et j variant dans le bon intervalle. On utilisera la fonction `np.zeros` pour créer l'image résultat.

1.4 Détection de contours

On peut détecter les contours des objets présents dans une image en niveaux de gris en calculant (par exemple) la norme au carré du gradient local, appelée *énergie* et notée E :

$$E[i,j] = \left(\frac{im\,[i+1,j] - im\,[i-1,j]}{2}\right)^2 + \left(\frac{im\,[i,j+1] - im\,[i,j-1]}{2}\right)^2$$

On prendra E infini sur les bords gauche et droit pour simplifier la suite.

□ **Question 7.** Que fait la fonction suivante ? Expliquer le type de chaque valeur et la façon dont `numpy` traite les tableaux. On consultera la documentation de `np.gradient` et on précisera la forme de `E`.

```
def energie(im_ng):
    gx,gy = np.gradient(im_ng)
    E = gx*gx + gy*gy
    E[:,0] = E[:,-1] = np.inf # gardes
    return E
```

□ **Question 8.** Visualiser l'image produite par la fonction `energie`.

2 Seam Carving

L'algorithme *Seam Carving* est un algorithme de réduction « intelligent » qui tient compte du contenu, en préservant les zones contrastées au détriment des zones plus homogènes.

Image originale, redimensionnement naïf, et redimensionnement « intelligent »

Pour préserver la structure des objets de l'image, on supprime les pixels « chemin par chemin ». Un chemin est défini comme un ensemble « continu » de pixels, joignant un pixel du haut de l'image ($i = 0$) à un pixel du bas de l'image. Un chemin optimal est représenté en rouge sur la troisième figure. Pour une image de hauteur H, un chemin sera représenté par une liste d'indices de colonnes $C = [j_0, j_1, \ldots, j_{H-1}]$, vérifiant :

$$\forall\, i \in [\![1, H]\!] \quad |j_i - j_{i-1}| \leqslant 1$$

À chaque chemin C on associe son score, défini par la somme des énergies des pixels le composant :

$$S(C) = \sum_{i=0}^{H-1} E[i, C[i]]$$

On définit enfin MS la matrice des meilleurs scores (plus basses énergies), tels que $MS[I,J]$ soit le meilleur score possible pour un chemin (partiel) joignant un point quelconque du haut de l'image au pixel (I,J) :

$$MS[I,J] = \min_{C,C[I]=J} \left(\sum_{i=0}^{I} E[i,C[i]] \right)$$

Question 9. On souhaite écrire une fonction calculant la matrice MS.

1. Que vaut $MS[0,j]$? Et $MS[i,0]$? Et $MS[i,L-1]$ (L désigne la largeur de l'image) ?
2. Si $i \geq 1$ et $j \in [\![1, L-2]\!]$, expliquer pourquoi l'égalité suivante est vérifiée :

$$MS[i,j] = E[i,j] + \min(MS[i-1,j-1], MS[i-1,j], MS[i-1,j+1])$$

3. Les deux questions précédentes permettent d'initialiser la matrice aux bords gauche, droit et haut, puis de calculer les éléments restants de proche en proche. Écrire alors une fonction `meilleur_score(E)` qui calcule et renvoie la matrice MS des meilleurs scores en fonction de la matrice `E`.

Question 10. Écrire une fonction `argmin(L)` prenant en argument une liste ou un tableau `numpy` à une dimension, et qui renvoie l'indice du plus petit élément de `L`.

Question 11. En déduire une fonction `chemin(MS)` qui renvoie un chemin C associé au score minimum, sous la forme d'une liste contenant les indices de colonnes successifs. Pour tester votre fonction, on pourra colorer en rouge le chemin trouvé (comme sur la figure précédente) et afficher le résultat à l'aide des instructions suivantes :

```
C = chemin(MS)
im[range(len(C)),C] = [1,0,0] # attention, im est l'image en 3 niveaux de couleurs
plt.figure()
plt.imshow(im)
```

Question 12. Écrire une fonction `supprime(im, C)` qui renvoie un nouveau tableau `numpy` correspondant à l'image initiale où les pixels du chemin C ont été supprimés. Par souci d'efficacité, on décalera les pixels de la ligne `i` avec une instruction de la forme `im[i,j:-1] = im[i,j+1:]`, et on renverra une image avec un pixel de moins en largeur.

Question 13. Écrire enfin une fonction `seam_carving(im, n)` qui réduit intelligemment une image de n pixels en largeur, en recalculant à chaque étape E, MS et C.

Attention la complexité de cette fonction est assez élevée : commencez par la tester avec des valeurs de n raisonnables (comme $n = 50$).

Question 14. Justement, quelle est la complexité de la fonction `seam_carving` en fonction des dimensions de l'image `im` et de n ?