TP : Traitements d'images

Objectifs : Le but de ce TP est de réaliser les différentes fonctions de traitement d'image vues en cours. L'énoncé est volontairement assez succinct. Vous penserez donc à utiliser la fonction `help`. Chaque image sera affichée dans une figure portant le nom de la transformation qui lui a été appliquée.

I Rappels sur les manipulations de tableau

Il est possible et même souhaitable de faire des calculs « vectorisés » sur les tableaux, c'est d'ailleurs ce qui fait l'efficacité de `numpy` (supposé ici importé sous le suffixe `np`). Mais le tableau final rendu sera du type correspondant au type le plus englobant nécessaire.
Par exemple, si `tableau` est un tableau dont les coefficients sont de type `np.uint8`, alors `256 - tableau` sera de type `uint16`.
Quelques rappels :
• On obtient les « tailles » d'un tableau par : `tableau.shape` ou `np.shape(tableau)` c'est un `tuple`.
• Création d'un tableau à trois dimensions $5 \times 2 \times 8$ remplis de 0 : `np.zeros((5, 2, 8))`. Noter qu'il faut passer un `tuple` comme paramètre.
Création d'un tableau à trois dimensions $5 \times 2 \times 8$ remplis de 1 : `np.ones((5, 2, 8))`.
• minimum et maximum **globaux** d'un tableau `np.amin(tableau)` et `np.amax(tableau)`
minimum **élément par élément** de deux tableaux : `np.fmin(tableau_1, tableau_2)`
maximum élément par élément de deux tableaux : `np.fmax(tableau_1, tableau_2)`

II Ouvrir et afficher une image

Il existe plusieurs modules permettant d'ouvrir une image, nous utiliserons le module `imageio`. On utilisera alors la fonction `imread()` afin d'ouvrir une image. Cette fonction prend comme argument une chaîne de caractère qui correspond à l'adresse de l'image à ouvrir (par exemple `imread('./dossier_perso/mes_images/mon_image.jpg')`) et renvoie un tableau de pixels de type `numpy.array` codé en `numpy.uint8` (entiers non signés sur 8 bits i.e. les entiers entre 0 et 255).

Pour afficher une image dans une figure, on pourra utiliser la fonction `imshow` du module `matplotlib.pyplot`. Attention un affichage correct n'a lieu que si l'on affiche un tableau de type `numpy.uint8`.

Une image en niveaux de gris est un tableau à deux dimensions d'entiers non signés sur 8 bits (type `numpy.uint8`). Pour l'afficher correctement, on utilise l'option `cmap=cm.gray` dans `imshow`. Pour cela, une fois le module `matplotlib.cm` importé sous l'alias `cm` et le module `matplotlib.pyplot` sous l'alias `plt`, on utilise `plt.imshow(image_gris, cmap=cm.gray)`.

Vous pouvez également enregistrer l'image grâce à la fonction `imsave`.

1 Mises en garde

Les manipulations de tableaux peuvent être délicates en raison de leur type (qui peut changer en cours de manipulation si on n'y prend pas garde). Pour un tableau (image initiale obtenue

via `imread`) codé en `uint8` tous les calculs se font modulo 256 ! Ainsi si on ajoute 10 à 250, on ne dépasse pas le seuil 255 mais la valeur prise en compte est $10 + 250 \mod 256$ i.e. 4 ce qui pose des problèmes de manipulation par exemple pour coder une augmentation de luminosité. En revanche, on doit absolument avoir un tableau codé en `uint8` pour obtenir un affichage conforme à ce qu'on attend.

2 Méthode de travail

Lorsque l'on est confronté à ce problèmes, on pourra procéder de la manière suivante :

— Ouvrir l'image `image`

— Obtenir une copie en tableau dont les coefficients sont de type `numpy.int16` : `image=image.astype(np.int16)` (la méthode `astype` fabrique une copie)

— Faire les traitements souhaités en prenant garde que les résultats finaux soit des entiers entre 0 et 255 (mais toujours de type `int16`)

— Transformer l'image modifiée en tableau de type `numpy.uint8` : `image.astype(numpy.uint8)` et l'afficher (ce sera l'objet de la procédure `affiche` (cf plus loin)

3 À vous

1. Importer **proprement** (avec un suffixe simple) toutes les bibliothèques utiles.
2. Télécharger sur hugoprépa l'image `vache.jpg` sur laquelle nous allons travailler et la placer dans votre espace personnel.
3. Écrire le script permettant de l'ouvrir et de l'afficher dans Pyzo.
4. Écrire une fonction `affiche` d'arguments un tableau image (de type `int`), une chaîne de caractère et un boolén (pour déterminer si l'image est en niveau de gris ou non), qui affiche proprement l'image correspondante.
   C'est l'utilisateur qui veillera à utiliser la fonction `affiche` dans ses conditions de bon fonctionnement : les valeurs des entiers seront tous entre 0 et 255. La fonction `affiche` pourra afficher selon le cas, une image en couleur ou une image en niveaux de gris.

III Quelques manipulations pour commencer

1. Quel est le code RGB du pixel dans le coin supérieur droit ?
2. Afficher séparément les trois composantes rouge, verte et bleue. On affichera des images en niveaux de rouge, bleu ou vert. Pour cela, on commencera par copier l'image (méthode `image.copy()`) pour ne pas la modifier et fixer dans la copie les deux composantes inutiles à 0. On privilégiera pour cela les affectations globales.
3. Tracer un trait noir horizontal de deux pixels de large au milieu de l'image.
4. Afficher l'image en négatif.

IV Opérations de base

1. Réaliser une fonction `change_luminosite(image, decalage)` qui prend comme arguments l'image à traiter et un entier compris en -255 et 255 désignant le montant dont on souhaite augmenter ou diminuer la luminosité des composantes RGB.
   On pourra d'abord écrire une fonction à l'aide de boucles puis essayer un traitement global (vectorisé). On appréciera le gain de temps réalisé.
2. Écrire une fonction `convertion_niveaux_gris(image)` convertissant l'image en niveaux de gris.
   Appliquer cette fonction à l'image de la vache et enregistrer l'image résultat sous le nom grisette.
3. Créer une fonction `seuil(image, I_seuil)` qui, à partir d'une image en niveau de gris, rend une image en noir et blanc. Les pixels dont l'intensité est plus faible de `I_seuil` seront noirs, les autres blancs.
4. Réaliser une fonction `augmente_contraste(image, min, max)` qui prend comme arguments l'image (en couleur) à traiter et deux entiers *min* et *max* : les pixels dont l'intensité est plus faible que *min* deviendront noir, les pixels dont l'intensité est plus grande que *max* deviendront blanc et les pixels entre *min* et *max* seront repartis entre 0 et 255.
5. Afin de bien choisir les valeurs des seuils *min* et *max*, on peut s'aider d'histogrammes. Créer une fonction `trace_histogrammes(image)` qui trace les trois histogrammes des intensités des pixels rouge vert et bleu. On utilisera pour cela la fonction `hist` du module `matplotlib.pyplot`. Choisir les valeurs de *min* et *max* optimaux pour l'image étudiée.

V Opérations avancées

Dans cette partie, on travaillera à partir de l'image en niveau de gris grisette..

1 Filtrage

1. On souhaite réaliser un lissage. Commencez par créer une fonction qui prend comme argument la taille d (un entier naturel impair) du masque de convolution et qui retourne un tableau de taille $d \times d$ dont les coefficients valent tous $\frac{1}{d^2}$.
2. Créer une fonction qui calcule le produit de convolution entre l'image à traiter et le masque de convolution.
3. Appliquer le lissage sur l'image pour différentes valeurs de d.
4. Écrire une fonction créant un masque de flou de gaussien de rayon σ et de taille d impaire.
5. Appliquer ce masque à l'image.

   Remarque : on pourra dans un premier temps travailler pixel par pixel, puis essayer de vectorialiser le problème (traitement plus global en utilisant au mieux la structure en tableau) et enfin utiliser la fonction `scipy.signal.convolve2d`. On appréciera le gain de temps de traitement ainsi réalisé.

2 Détection de contour

Pour gagner en rapidité, on pourra utiliser la fonction `scipy.signal.convolve2d`.

1. Créer les deux filtres de Sobel.
2. Créer une fonction qui calcule la norme du gradient de l'image. L'appeler sur l'image grisette et afficher le résultat.
3. Appliquer la fonction seuil à l'image obtenue précédemment pour obtenir uniquement les contours.
4. Améliorer la détection en appliquant préalablement un flou gaussien.