Informatique

7
Matrices de pixels et images

TD 7-2

Détection de contours

Convolution

A.I. Contexte

Il existe une multitude de contextes dans lesquels la détection de contours est mise en œuvre (effets artistiques, identification de routes et fleuves, tracé du réseau de vaisseaux sanguins du fond d'œil, imagerie médicale, lecture d'empreintes digitales, reconnaissance des bords d'un document pour scanning sur smartphone type Genius Scan, ajout de filtres sur les visages type Snapchat…).

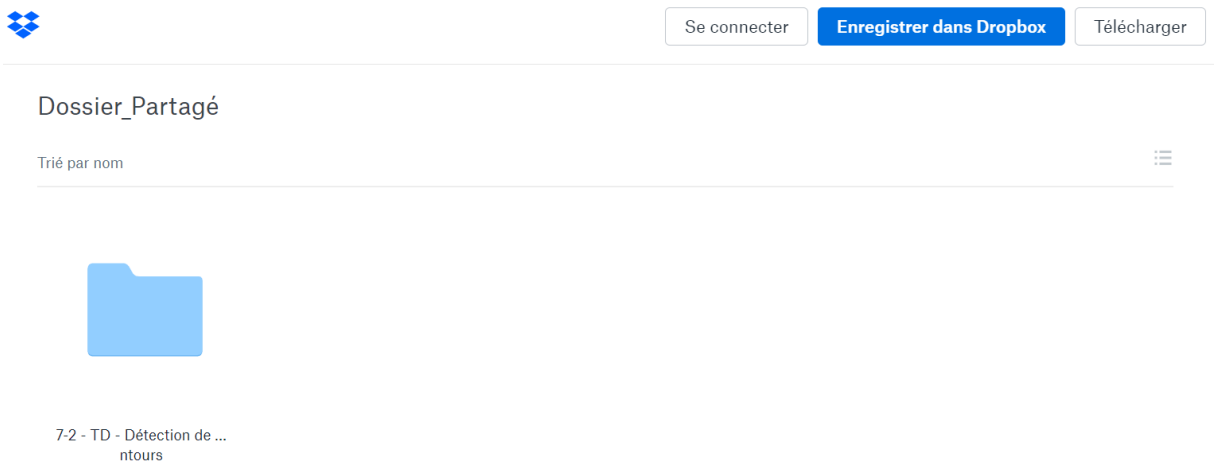
Aujourd'hui, intéressons-nous à une utilisation médicale. Les grains de beauté peuvent devenir des mélanomes (cancers). Le suivi régulier de leur couleur et de la forme de leur contour par un dermatologue permet d'en prévenir les risques. Nous souhaitons ici créer un algorithme capable de renvoyer l'image des contours des grains de beauté d'un individu. Il sera alors possible par la suite de comparer des images prises régulièrement de manière automatisée afin d'avertir l'utilisateur du danger imminent que représente l'un de ses grains de beauté.

Voici l'image que nous utiliserons, trouvée sur Google image :

Vous trouverez toutes les images dans le dossier élèves en lien plus bas.

Affichage de l'image

Afin d'assurer un fonctionnement rapide sur tous les ordinateurs, je vous mets à disposition un dossier à télécharger COMPLETEMENT, soit le dossier contenant tous les fichiers, et non les fichiers pris séparément.



Sans ouvrir le dossier, faite juste « Télécharger – Téléchargement direct » puis mettez ce dossier dans votre répertoire personnel.

LIEN

Si le téléchargement est sous forme de Rar, Zip… Pensez à dézipper l'archive afin d'avoir le dossier voulu !

Question 1: Télécharger et exécuter le code fourni qui affichera l'image fournie « Image » sous Python

Transformation en nuances de gris

Une couleur grise est simplement définie sous Python en mode RGB à l'aide d'un code tel que $R = G = B$.
Ainsi, définir $R = G = B = 0$ crée un pixel noir, $R = G = B = 255$ crée un pixel blanc, et toute valeur entre 0 et 255 donne une nuance de gris, par exemple $R = G = B = 120$.

On souhaite dans un premier temps transformer la photo couleur infrarouge en une photo en nuances de gris. Pour tout pixel, on propose d'associer la nuance suivante :

$$Nuance = k_R R + k_G G + k_B B \; avec \begin{cases} k_R = 0{,}33 \\ k_G = 0{,}33 \\ k_B = 0{,}34 \end{cases}$$

Ce choix permet de donner ne pas privilégier de couleur particulière.

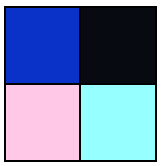
Eviter d'écrire $(R + G + B)/255$ qui peut faire apparaître des problèmes d'overflow.


Exemple d'overflow
Merci Arsène pour l'image
(MPSI 21/22)

Remarque : Pour la suite, je vous suggère lorsque vous modifiez une image de la copier d'abord avec $Image\_Copie = np.copy(Image)$. En effet, si vous faites uniquement $Image\_Copie = Image$, toute modification sur l'une modifiera l'autre, comme pour les listes…

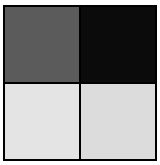
Question 2: Créer une fonction f_Nuances_de_gris(im) qui renvoie une image basée sur l'image im telle que chacun de ses pixels soit un triplet de 3 valeurs identiques dont le calcul est basé sur la formule de nuance proposée ci-dessus

Exemple : sur une image à 4 pixels



$$\begin{bmatrix} [10 & 50 & 200] & [7 & 10 & 17] \\ [255 & 200 & 230] & [150 & 255 & 255] \end{bmatrix}$$

L'algorithme renvoi un array tel que :



$$\begin{bmatrix} [91 & 91 & 91] & [11 & 11 & 11] \\ [228 & 228 & 228] & [220 & 220 & 220] \end{bmatrix}$$

Question 3: Utiliser cette fonction et visualiser l'image issue de celle-ci afin de vérifier que vous obtenez bien une image en nuances de gris

Identification des contours

Il existe diverses méthodes pour identifier les contours dans une image. L'idée de cette partie est d'étudier les variations des nuances de gris, entre des pixels adjacents. Nous nous limiterons à l'étude d'un pixel et des 8 autres qui lui sont adjacents. Soit donc un pixel de nuance Px, entouré de 8 pixels adjacents. On définit la matrice locale, carrée de côté 3, contenant les 9 valeurs entières des R=G=B des 1+8 pixels concernés :

$$Mat\_Loc(Px) = \begin{bmatrix} 230 & 229 & 219 \\ 210 & Px & 167 \\ 189 & 192 & 142 \end{bmatrix}$$

Soit la quantité suivante calculée sur tous les pixels hors des bords (les bords ne seront pas modifiés) :

$$Ecart = \frac{\sum_{i=0}^{2}\sum_{j=0}^{2}|Mat\_Loc[i,j] - Px|}{8}$$

Cet écart sera compris entre 0 et 255. Plus il est grand, plus il y a de différences entre la valeur de la nuance Px du pixel central et les valeurs des nuances des pixels adjacents. Cet écart est donc grand en présence de contraste, c'est-à-dire de contour 😊.

Remarque : si cela vous intéresse, vous pourrez ici essayer d'autres calculs d'écarts, distance euclidienne, valeur max etc

On définit donc un critère $Crit$ tel que : $\begin{cases} Ecart > Crit \Rightarrow Présence\ de\ contour \\ Ecart \leq Crit \Rightarrow Pas\ de\ contour \end{cases}$

On crée alors une nouvelle image telle que :

- Si l'on est sur un contour, le pixel est noir : R=G=B=0
- Sinon, le pixel est blanc : R=G=B=255

Question 4: Créer une fonction f_Ecart(M) qui prend en argument une matrice carrée d'entiers de côté impaire, et qui renvoie l'écart défini ci-dessus
Essayez :

```
>>> M = np.array([[100,100,100],[100,200,100],[100,100,100]],dtype="uint8")

>>> f_Ecart(M)
100.0
```

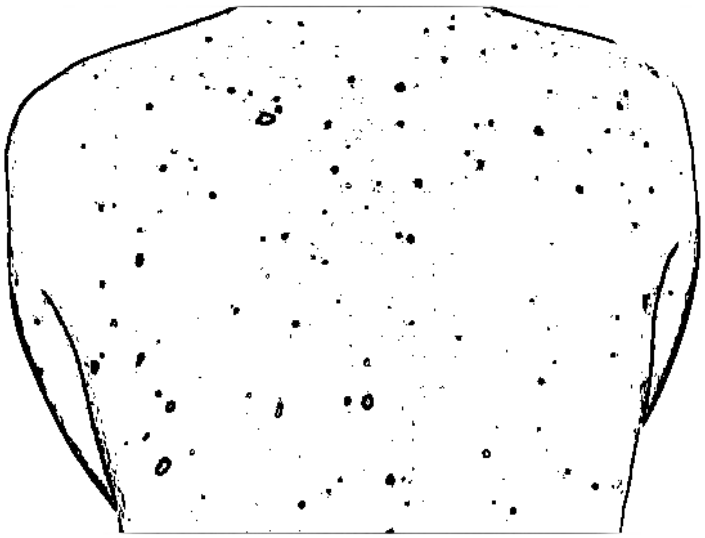
Vous obtenez 156 ? Vous rencontrez un problème d'overflow dans la différence M[l,c]-Pix. Pensez dans ce cas à une différence de flottants.

Question 5: Créer une fonction f_Contour(im,Crit) qui prend en argument l'image en nuances de gris im et le critère Crit, fait une copie de im, et renvoie une nouvelle image Im_Ctr composée de pixels noirs et blancs contenant les contours de l'image im (les bords ne seront pas modifiés)

Remarques :

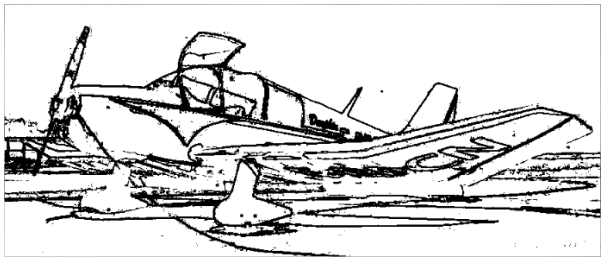
- Vous pouvez récupérer un array de tous les R d'une image im en écrivant « `im[:,:,0]` »
- Vous obtenez une image comme ci-dessous (le résultat dépend de la valeur décrit choisie) ? Cela vient de l'overflow de f_Ecart
- Erreur « index 1 is out of bounds for axis 0 with size 0 » ? J'ai bien précisé de ne pas traiter les bords… Ajoutons qu'écrire L[i-1 :2] avec i=0 donne L[-1 :2], slice à l'envers, ce qui renvoie un résultat vide !

Question 6: Créer les lignes de code permettant d'afficher les contours de l'image proposée dans ce sujet, et jouer sur le critère Crit afin d'obtenir un résultat satisfaisant



Traitement par convolution

Nous allons maintenant aller plus loin en utilisant la convolution. Je vous suggère d'utiliser l'image nommée « avion » dans le dossier élèves et d'en afficher les contours.
Vous remarquerez que selon la photo, du bruit peut apparaître :

Bien que cela ne soit pas forcément une bonne idée dans le cas où l'on souhaite trouver de petites formes (grains de beauté), une solution simple consiste à retravailler l'image avec la convolution en appliquant filtre moyenneur (floutage) avant d'en déterminer les contours avec l'algorithme précédent.

Question 7: Créer une fonction f_Normalisation(K) qui prend en argument une matrice noyau K et renvoie la matrice normalisée associée si la somme des termes est différente de 0, K sinon
Question 8: Créer une fonction f_Conv_Loc(M,K) qui prend en argument une matrice noyau K (array) composée d'entiers et une portion d'image M (array) de mêmes dimensions que K composée de triplets RGB, et qui renvoie le triplet (liste) d'entiers RGB résultat du produit de convolution Mat*K
Remarques :

- Le simple produit M*K ne donnera pas le résultat espéré
- Les valeurs de R, G et B de sortie peuvent sortir de l'intervalle [0,255] et il y risque d'overflow lors de l'application du résultat dans la fonction suivante sur une image en uint8. Pour pallier ce problème, il faudra donc limiter les valeurs de R, G et B de sortie en utilisant les fonctions min et max de python afin que
    - Toute valeur négative soit limitée à 0
    - Toute valeur supérieure à 255 soit limitée à 255

Exemple de ce que réalise cette fonction :

$$K = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \quad ; \quad M = \begin{bmatrix} [a_1,a_2,a_3] & [b_1,b_2,b_3] & [c_1,c_2,c_3] \\ [d_1,d_2,d_3] & [e_1,e_2,e_3] & [f_1,f_2,f_3] \\ [g_1,g_2,g_3] & [h_1,h_2,h_3] & [i_1,i_2,i_3] \end{bmatrix}$$

$$[Aa_1 + Bb_1 + Cc_1 + Dd_1 + Ee_1 + Ff_1 + Gg_1 + Hh_1 + Ii_1,$$
$$\Rightarrow Aa_2 + Bb_2 + Cc_2 + Dd_2 + Ee_2 + Ff_2 + Gg_2 + Hh_2 + Ii_2,$$
$$Aa_3 + Bb_3 + Cc_3 + Dd_3 + Ee_3 + Ff_3 + Gg_3 + Hh_3 + Ii_3,]$$

Vérifiez :

```
K = (1/9)*np.array([[1,1,1],[1,1,1],[1,1,1]])
```

```
M = np.array([[[221,10,120],[219,15,110],[216,14,115]],[[232,5,109],[225,21,108],[210,17,120]],[[217,12,116],[218,10,118],[220,9,125]]])
```

```
>>> f_Conv_Loc(M,K)
[219, 12, 115]
```

Question 9: Créer une fonction f_Convolution(Im,K) qui réalise la convolution de noyau K normalisé de l'image Im et renvoie l'image résultat

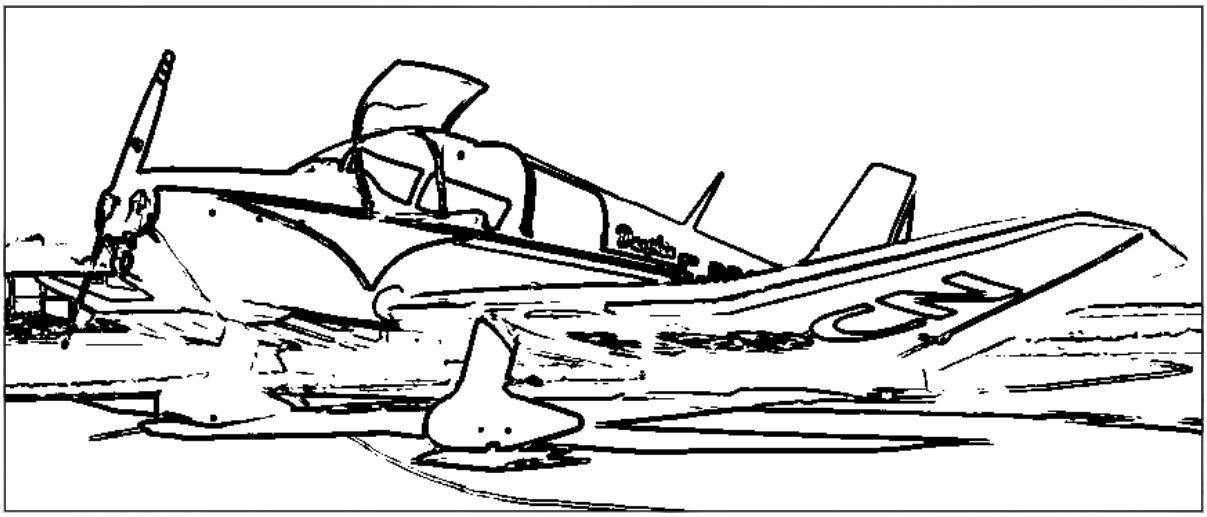
Remarques :

- La convolution ne sera pas appliquée sur les bords qui ne le permettent pas, les pixels des bords de l'image Im ne seront ainsi pas modifiés dans le résultat
- Le nombre de pixels non traités sur les bords dépend de la taille de la matrice K
- Si vous faites une copie de l'image initiale, et si vous appliquez une liste d'entiers (résultat de f_Conv_Loc) à la place de l'un de ses triplets, alors si des valeurs sont en dehors de [0,255], il y aura overflow sans en être averti ! Mais à priori, vous avez déjà réglé le problème avant

Question 10: Créer les lignes de code permettant de créer un floutage par convolution avec filtre moyenneur de l'image en noire et blanc

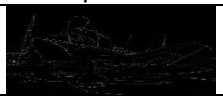
Vérifiez le résultat :


Question 11: Créer les lignes de code permettant d'afficher les contours de l'image proposée dans ce sujet après avoir été floutée en nuances de gris



Remarque : Il est possible d'appliquer le floutage à l'image en couleurs, puis de la transformer en nuances de gris pour ensuite en déterminer les contours, mais vous verrez que le résultat est très similaire

Question 12: Amusez-vous à essayer d'autres convolutions en prenant comme base l'image en couleurs

Repoussage	Laplacien	Réhausseur	Gaussien 3x3	Gaussien 5x5
				

Pour le Laplacien, appliquez une convolution qui multiplie par k chaque R, G et B de l'image sans normalisation pour amplifier le blanc :