Informatique

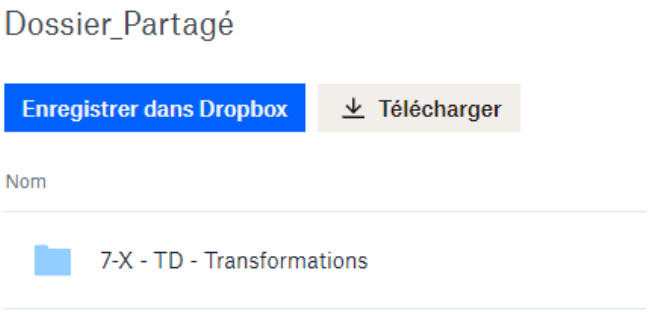
7
Matrices de pixels et images

TD7-3

Transformations

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

Rotation directe (1)

On souhaite programmer la rotation de l'image source d'un angle α de centre $C(lc, cc)$ par la méthode directe qui détermine l'image par rotation $R(C, \alpha)$ de tout pixel $P(l, c)$ en un pixel $P'(l', c')$, et lui applique le triplet RGB de l'image source s'il est contenu dans l'image cible.

Question 2: Proposer une fonction f_Prod_MV(M,V) qui renvoie le produit matrice (M) vecteur (V) M*V sous forme d'une liste de n termes, M étant une liste de n liste de n termes et V une liste de n termes

Vérifier :

```
>>> M = [[1,2],[3,4]]

>>> V = [5,6]

>>> f_Prod_MV(M,V)
[17, 39]
```

Question 3: Proposer une fonction f_Rotation(V,Alpha_d) prenant en argument une liste V représentant un vecteur, ainsi que l'angle de rotation Alpha_d degrés souhaité, et renvoyant la liste image du vecteur V par cette rotation

Vérifier :

```
>>> f_Rotation([1,1],45)
[0.0, 1.4142135623730951]
```

Question 4: Proposer une fonction f_Rot1(image,C,Alpha_d) qui crée et renvoie une image initialisée en blanc (array d'entiers 255 de type uint8) et remplace le blanc par la bonne couleur, quand c'est possible, par rotation directe de l'image source
Question 5: Mettre en place les quelques lignes de code affichant l'image tournée pour différents angles et centres et apprécier les résultats

Rotation indirecte (2)

On souhaite maintenant programmer la rotation de l'image source d'un angle α de centre $C(lc, cc)$ par la méthode indirecte qui détermine l'antécédent $P(l, c)$ par rotation $R(C, -\alpha)$ de tout pixel $P'(l', c')$, et applique à P' le triplet RGB de l'image source si l'antécédent est contenu dans l'image source.

Question 6: Proposer une fonction f_Rot2(image,C,Alpha_d) qui crée et renvoie une image initialisée en blanc (array d'entiers 255 de type uint8) et remplace le blanc par la bonne couleur, quand c'est possible, par rotation directe de l'image source
Question 7: Mettre en place les quelques lignes de code affichant l'image tournée pour différents angles et centres et apprécier les résultats

Réduction par suppression (1)

On souhaite programmer la réduction d'une image par suppression de lignes et colonnes. On propose de procéder en deux étapes, une suppression de lignes d'abord, en gardant la première ligne puis, une ligne toutes les nl lignes, puis en procédant de la même manière toutes les nc colonnes.

Question 8: Proposer une fonction f_Red1_L(image,nl) qui renvoie une nouvelle image remplie des lignes à garder dans « image »

Remarque : on pourra créer une liste de listes, puis la transformer à la fin en un array d'entiers en écrivant `Im_Red = np.array(Im_Red,dtype='uint8')`

Question 9: Proposer une fonction f_Red1_C(image,nc) qui renvoie une nouvelle image remplie des colonnes à garder dans « image »

Question 10: Mettre en place les quelques lignes de code affichant l'image après suppression des lignes uniquement, des colonnes uniquement, puis des lignes et des colonnes

Réduction par moyenne de paquets (2)

On souhaite programmer la réduction d'une image en remplaçant les pixels par blocs carrés de dimensions nxn (n impair) par la moyenne du bloc (on ne traitera pas les dernières lignes/colonnes qui resteraient du fait de dimensions de l'image non multiples de n).

On définit $k = \left\lfloor \frac{n}{2} \right\rfloor$. Les blocs auront donc une dimension de $2k+1$ par $2k+1$.

Question 11: Proposer une fonction f_Mat_LC(image,L,C,k) prenant en argument l'image source « image », la ligne L et la colonne C du pixel Pix au centre du bloc, k défini précédemment, et renvoyant sous forme de liste de listes, le bloc des pixels (array RGB) autour de Pix (L±k et c±k)

Vérifier :
```
>>> f_Mat_LC(Image,10,10,1)
[[array([182, 176, 174], dtype=uint8), array([157, 152, 149]
, dtype=uint8), array([180, 175, 173], dtype=uint8)], [array
([188, 183, 180], dtype=uint8), array([163, 158, 155], dtype
=uint8), array([186, 178, 176], dtype=uint8)], [array([187,
182, 179], dtype=uint8), array([166, 161, 158], dtype=uint8)
, array([176, 172, 168], dtype=uint8)]]
```

Question 12: Proposer une fonction f_Moyenne(M) prenant en argument un bloc carré M sous forme de liste de liste issu d'une image et renvoyant la liste des moyennes des valeurs de R, G et B du bloc

Vérifier :
```
>>> f_Moyenne(f_Mat_LC(Image,10,10,1))
[176.11111111111111, 170.77777777777777, 168.0]
```

Question 13: Proposer une fonction f_Red2(image,n) qui renvoie une nouvelle image, réduction de « image » par blocs de dimensions nxn

Question 14: Mettre en place les quelques lignes de code affichant l'image après réduction par paquets

Agrandissement

On souhaite réaliser l'agrandissent d'une image en ajoutant n lignes/colonnes entre chaque ligne/colonne de l'image originale. Ainsi, si l'image originale est de dimensions $L * C$, elle aura après transformation la dimension $(n(L-1)+1) * (n(C-1)+1)$.
Pour l'agrandissement par plus proches voisins, nous pourrions procéder par 2 étapes comme pour la réduction par suppression, par lignes puis colonnes. Toutefois, comme dans l'agrandissement par interpolation bilinéaire, nous utiliserons l'interpolation bilinéaire qui ajoutera les lignes et colonnes en même temps, j'ai choisi de procéder de la même manière pour les plus proches voisins.
Ainsi, nous allons réaliser les choses de la manière suivante :

- Mise en place d'une fonction f_AG(image,n,fonction) qui prendra en argument l'image source « image », le nombre de lignes/colonnes à ajouter n, et une fonction parmi les deux suivantes :
    - f1(image,l,c) : fonction qui renvoie le Pixel (array RGB) de l'image source « image » le plus proche du pixel P(l,c) avec priorité « Haut » ou « Droite », l et c des flottants
    - f2(image,l,c) : fonction renvoyant l'array RGB issu de l'interpolation bilinéaire en utilisant les 4 voisins de l'image source « image » autour de P(l,c), l et c des flottants

Cela aura l'avantage de rendre la fonction f_AG utilisable pour les deux méthodes d'agrandissement en appelant f1 ou f2.

Pour la fonction f_AG, cn créera une image blanche dans laquelle on remplacera chaque pixel P(l,c) par le résultat renvoyé par la fonction « fonction(image,ll,cc) » où ll et cc seront les « coordonnées » du pixel de l'image résultat dans l'image source. Autrement dit, ll et cc seront des flottants représentant des lignes et colonnes dans l'image source, des nouveaux pixels de l'image cible. Par exemple, si on ajoute 2 lignes entre chaque ligne d'une image de dimensions initiales (Nl, Nc) et de dimensions finales (Nl_New, Nc_New):

- Le pixel de la 1° ligne $(l = 0)$ de l'image cible aura $ll = 0$
- Le pixel de la 2° ligne $(l = 1)$ aura $ll \approx 0{,}33$
- Le pixel de la 3° ligne $(l = 2)$ aura $ll \approx 0{,}66$
- Le pixel de la 4° ligne $(l = 3)$ aura $ll = 1$
- …
- Le pixel de la dernière ligne $(l = Nl\_New - 1)$ aura $ll = Nl - 1$

Question 15: Donner l'expression de ll en fonction de l, Nl, Nl_New et de cc en fonction de c, Nc et Nc_New

Question 16: Proposer la fonction f_AG(image,n,fonction) renvoyant l'image agrandie, quel que soit la fonction « fonction » utilisée

Remarque : en réfléchissant bien, vous remarquerez qu'il est possible de mettre n'importe quelles valeurs à Nl_New et Nc_New dans cette fonction, l'image sera agrandie/réduite en conséquence 😊. En gardant nos formules, on ajoutera bien n lignes/colonnes entre les lignes et colonnes initiales.

Agrandissement par plus proches voisins (1)

Question 17: Mettre en place la fonction f1(image,l,c) renvoyant comme proposé le pixel dans l'image source « image » le plus proche de la ligne l (flottant) avec priorité à gauche et de la colonne c (flottant) avec priorité en haut

Vérifier :

```
>>> f1(Image,0,0)
array([132, 131, 137], dtype=uint8)

>>> f1(Image,1,1)
array([141, 137, 143], dtype=uint8)

>>> f1(Image,0.5,0.5)
array([132, 131, 137], dtype=uint8)

>>> f1(Image,0.51,0.51)
array([141, 137, 143], dtype=uint8)
```

Question 18: Mettre en place les quelques lignes de code affichant l'image après agrandissement par plus proches voisins

Agrandissement par interpolation bilinéaire (2)

Question 19: Mettre en place la fonction f2(image,l,c) renvoyant comme proposé le pixel résultat de l'interpolation bilinéaire utilisant les pixels environnants

Remarques

- Vous pourrez tirer parti des opérations possibles avec les arrays pour ne pas modifier la fonction Prod_MV et pour proposer une fonction Prod_VV du produit de deux arrays
- Il faudra traiter à part les cas particuliers suivants :
  - Pixel à la dernière ligne et colonne : renvoyer le pixel concerné
  - Dernière ligne : Renvoyer une interpolation linéaire uniquement entre les deux pixels voisins sur la ligne par interpolation linéaire simple
  - Dernière colonne : Renvoyer une interpolation linéaire uniquement entre les deux pixels voisins sur la colonne par interpolation linéaire simple
  - Autres cas : renvoyer le résultat de la formule du cours

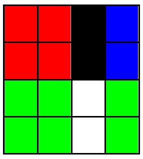
Vérifier :

```
>>> Nl,Nc = np.shape(Image)[:2]

>>> f2(Image,Nl-1,Nc-1)
[145, 113, 88]

>>> f2(Image,10,Nc-1)
[148, 135, 143]

>>> f2(Image,Nl-1,0)
[141, 122, 105]

>>> f2(Image,0.5,0.5)
[133, 130, 136]

>>> f2(Image,0,0)
[132, 131, 137]
```

Question 20: Mettre en place les quelques lignes de code affichant l'image après agrandissement par interpolation bilinéaire

Pour aller plus loin

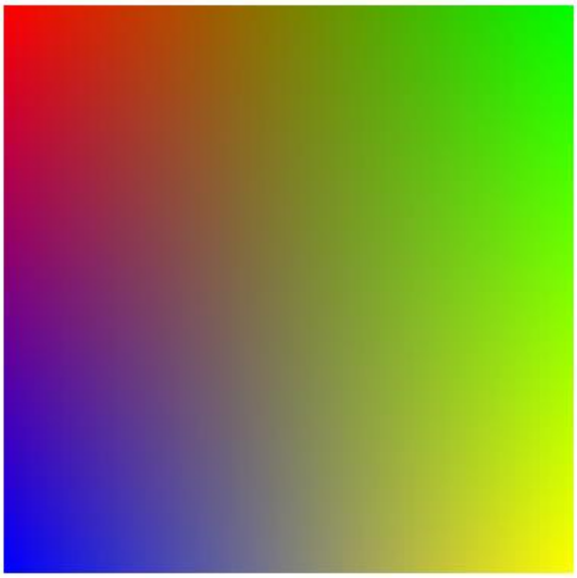
Rappel de l'exemple du cours (n=4) :



R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	**R=255 G=0 B=0**	R=191 G=0 B=0	R=127 G=0 B=0	R=63 G=0 B=0	**R=0 G=0 B=0**	R=0 G=0 B=63	R=0 G=0 B=127	R=0 G=0 B=191	**R=0 G=0 B=255**
R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=191 G=0 B=0	R=127 G=0 B=0	R=63 G=0 B=0	R=0 G=0 B=0	R=0 G=0 B=63	R=0 G=0 B=127	R=0 G=0 B=191	R=0 G=0 B=255
R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=191 G=0 B=0	R=127 G=0 B=0	R=63 G=0 B=0	R=0 G=0 B=0	R=0 G=0 B=63	R=0 G=0 B=127	R=0 G=0 B=191	R=0 G=0 B=255
R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=191 G=0 B=0	R=127 G=0 B=0	R=63 G=0 B=0	R=0 G=0 B=0	R=0 G=0 B=63	R=0 G=0 B=127	R=0 G=0 B=191	R=0 G=0 B=255
R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	R=255 G=0 B=0	**R=255 G=0 B=0**	R=191 G=0 B=0	R=127 G=0 B=0	R=63 G=0 B=0	**R=0 G=0 B=0**	R=0 G=0 B=63	R=0 G=0 B=127	R=0 G=0 B=191	**R=0 G=0 B=255**
R=191 G=63 B=0	R=191 G=63 B=0	R=191 G=63 B=0	R=191 G=63 B=0	R=191 G=63 B=0	R=159 G=63 B=15	R=127 G=63 B=31	R=95 G=63 B=47	R=63 G=63 B=63	R=47 G=63 B=95	R=31 G=63 B=127	R=15 G=63 B=159	R=0 G=63 B=191
R=127 G=127 B=0	R=127 G=127 B=0	R=127 G=127 B=0	R=127 G=127 B=0	R=127 G=127 B=0	R=127 G=127 B=31	R=127 G=127 B=63	R=127 G=127 B=95	R=127 G=127 B=127	R=95 G=127 B=127	R=63 G=127 B=127	R=31 G=127 B=127	R=0 G=127 B=127
R=63 G=191 B=0	R=63 G=191 B=0	R=63 G=191 B=0	R=63 G=191 B=0	R=63 G=191 B=0	R=95 G=191 B=47	R=127 G=191 B=95	R=159 G=191 B=143	R=191 G=191 B=191	R=143 G=191 B=159	R=95 G=191 B=127	R=47 G=191 B=95	R=0 G=191 B=63
R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	**R=0 G=255 B=0**	R=63 G=255 B=63	R=127 G=255 B=127	R=191 G=255 B=191	**R=255 G=255 B=255**	R=191 G=255 B=191	R=127 G=255 B=127	R=63 G=255 B=63	**R=0 G=255 B=0**
R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=63 G=255 B=63	R=127 G=255 B=127	R=191 G=255 B=191	R=255 G=255 B=255	R=191 G=255 B=191	R=127 G=255 B=127	R=63 G=255 B=63	R=0 G=255 B=0
R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=63 G=255 B=63	R=127 G=255 B=127	R=191 G=255 B=191	R=255 G=255 B=255	R=191 G=255 B=191	R=127 G=255 B=127	R=63 G=255 B=63	R=0 G=255 B=0
R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=63 G=255 B=63	R=127 G=255 B=127	R=191 G=255 B=191	R=255 G=255 B=255	R=191 G=255 B=191	R=127 G=255 B=127	R=63 G=255 B=63	R=0 G=255 B=0
R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	R=0 G=255 B=0	**R=0 G=255 B=0**	R=63 G=255 B=63	R=127 G=255 B=127	R=191 G=255 B=191	**R=255 G=255 B=255**	R=191 G=255 B=191	R=127 G=255 B=127	R=63 G=255 B=63	**R=0 G=255 B=0**

Question 21: Vérifier votre programme en créant l'image support du cours et en validant les valeurs des triplets RGB trouvées avec Python

Voici une belle image :



Question 22: Mettre en place les quelques lignes de code permettant d'en réaliser une similaire