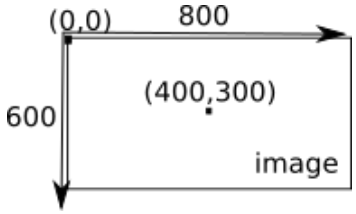
Séance n°5

Traitement d'Images avec Python

Nous allons utiliser le langage de programmation Python afin de directement travailler sur les pixels d'une image. Par travailler sur les pixels, j'entends déterminer la valeur du canal rouge, la valeur du canal et la valeur du canal bleu pour un pixel donné ou bien encore modifier carrément la couleur d'un pixel.

Avant de commencer à écrire un programme qui nous permettra de travailler sur les pixels d'une image, il est nécessaire de préciser que chaque pixel a des coordonnées x,y.



Comme vous pouvez le constater sur le schéma ci-dessus, le pixel de coordonnées (0,0) se trouve en haut à gauche de l'image. Si l'image fait 800 pixels de large et 600 pixels de haut, le pixel ayant pour coordonnées (400,300) sera au milieu de l'image.

Dans un premier temps nous allons utiliser l'image `andywarhol.jpg`. Cette image devra se trouver dans le même dossier que vos programmes Python.

Exercice 1 : Ouvrir le logiciel **EduPython**, puis saisir et tester le programme suivant :

```
from PIL import Image
im = Image.open("andywarhol.jpg")
r,v,b=im.getpixel((100,250))
print("canal rouge : ",r,"canal vert : ",v,"canal bleu : ",b)
```

Ce programme vous donne le canal rouge, le canal vert et le canal bleu du pixel de coordonnées (100,250) de l'image `andywarhol.jpg`

Voici une analyse ligne par ligne du programme ci-dessus :

- "from PIL import Image" : pour travailler sur les images nous avons besoin d'une extension de Python (appelé bibliothèque). Cette bibliothèque se nomme PIL.
- "im = Image.open("andywarhol.jpg")" c'est grâce à cette ligne que nous précisons que nous allons travailler avec l'image "andywarhol.jpg". Pour travailler avec une autre image, il suffit de remplacer "andywarhol.jpg" par un autre nom (attention, votre fichier image devra se trouver dans le même dossier que le ficher de votre programme Python).
- "r,v,b=im.getpixel((100,250))" cette ligne récupère les valeurs du canal rouge (r), du canal vert (v) et du canal bleu (b) du pixel de coordonnées (100,250). Dans la suite du programme, r correspondra à la valeur du canal rouge, v correspondra à la valeur du canal vert et b correspondra à la valeur du canal bleu
- "print("canal rouge : ",r,"canal vert : ",v,"canal bleu : ",b)" permet d'imprimer le résultat

Exercice 2 : Modifiez le programme de l'exercice 1 pour qu'il affiche les valeurs du canal rouge, du canal vert et du canal bleu du pixel de coordonnées (250,300), notez votre réponse.

Exercice 3 : Il est possible de modifier les canaux RVB d'un pixel. Saisissez et testez le programme suivant :

```
from PIL import Image
im = Image.open("andywarhol.jpg")
im.putpixel((250,250),(0,255,0))
im.show()
```

Regardez attentivement le centre de l'image, vous devriez voir un pixel vert.

Voici une analyse ligne par ligne du programme ci-dessus :

- "im.putpixel((250,250),(0,255,0))" permet de colorier le pixel de coordonnées (250,250) en rouge (0,255,0).
- "im.show()" permet d'afficher l'image modifiée

Exercice 4 : Modifiez le programme précédent afin de colorier le pixel de coordonnées (100,250) en bleu.

Modifiez un pixel c'est déjà bien, mais comment faire pour modifier plusieurs pixels ? La réponse est simple, nous allons utiliser des boucles "for". Le but ici n'est pas de détailler le fonctionnement des boucles "for" en Python, vous devez juste comprendre que grâce à ces boucles nous allons pouvoir balayer toute l'image et ne plus nous contenter de modifier les pixels un par un.

Exercice 5 : Saisissez et testez le programme suivant (ATTENTION : l'exécution de ce programme n'est pas très intéressante en soi, vous pouvez l'arrêter à tout moment en appuyant simultanément sur la touche Ctrl et sur la touche C):

```
from PIL import Image
im = Image.open("andywarhol.jpg")
largeur_image=1024
hauteur_image=1020
for y in range(hauteur_image):
    for x in range(largeur_image):
        r,v,b=im.getpixel((x,y))
        print("rouge : ",r,"vert : ",v,"bleu : ",b)
print("fin")
```

Quelques commentaires sur ce programme :

- Nous commençons par définir les variables "largeur_image" et "hauteur_image" ("largeur_image=1024" et "hauteur_image=1020"). Je pense que vous aurez compris que notre image "andywarhol.jpg" fait 1024 pixels de large et 1020 pixels de haut. Si vous désirez travailler avec une autre image, il faudra veiller à bien modifier la valeur de ces deux variables.
- Les 2 boucles "for" nous permettent de parcourir l'ensemble des pixels de l'image :

  ```
  for y in range(hauteur_image):
     for x in range(largeur_image):
                        ...
  ```

  Le plus important ici est de bien comprendre que dans la suite du programme, les variables x et y vont nous permettre de parcourir l'ensemble des pixels de l'image : nous allons commencer avec le pixel de coordonnées (0,0), puis le pixel de coordonnées (1,0), puis le pixel de coordonnées (2,0)...jusqu'au pixel de coordonnées (1023,0). Ensuite, nous allons changer de ligne avec le pixel de coordonnées (0,1), puis le pixel de coordonnées (1,1)...bref, le dernier pixel sera le pixel de coordonnées (1023,1019), tout cela grâce à la double boucle "for" !
- "r,v,b=im.getpixel((x,y))" cette ligne ne devrait pas poser de problème, nous avons juste remplacé les coordonnées des pixels par (x,y) afin de considérer l'ensemble des pixels de l'image.
- "print("rouge : ",r,"vert : ",v,"bleu : ",b)" nous imprimons les valeurs des canaux RVB pour chaque pixel de l'image.

Compliquons un peu la chose en modifiant tous les pixels de l'image.

Exercice 6 : Saisissez et testez le programme suivant :

```
from PIL import Image
im = Image.open("andywarhol.jpg")
largeur_image=1024
hauteur_image=1020
for y in range(hauteur_image):
    for x in range(largeur_image):
        r,v,b=im.getpixel((x,y))
        n_r=v
        n_v=b
        n_b=r
        im.putpixel((x,y),(n_r,n_v,n_b))
im.show()
```

Expliquez en quelques mots ce que fait ce programme.

Exercice 7 : En vous inspirant de ce qui a été fait dans l'exercice 6, écrivez un programme qui inverse les valeurs des canaux bleu et rouge sans changer la valeur du canal vert.

Exercice 8 : Après avoir fait quelques recherches sur le "négatif d'une image", écrivez un programme qui donne le négatif d'une image.

Exercice 9 : Testez le programme précédent avec une image de votre choix (attention aux variables "largeur_image" et "hauteur_image").

Pour l'instant nous avons modifié tous les pixels de l'image. Avec l'instruction "if", il est possible de modifier seulement certains pixels.

Exercice 10 : Saisissez et testez le programme suivant :

```
from PIL import Image
im = Image.open("andywarhol.jpg")
largeur_image=1024
hauteur_image=1020
for y in range(hauteur_image):
    for x in range(largeur_image):
        r,v,b=im.getpixel((x,y))
        if b<200:
            n_b=255-b
        else :
            n_b=b
        im.putpixel((x,y),(r,v,n_b))
im.show()
```

Expliquez en quelques mots ce que fait ce programme.

Pour aller plus loin : La détection de contours

Dans cette partie, nous allons utiliser la fonction mathématique racine carrée `sqrt` et comme nous travaillons sur les images, nous avons besoin de la bibliothèque PIL. Le programme doit donc débuter par ces deux lignes :

```
from PIL import Image

from math import*
```

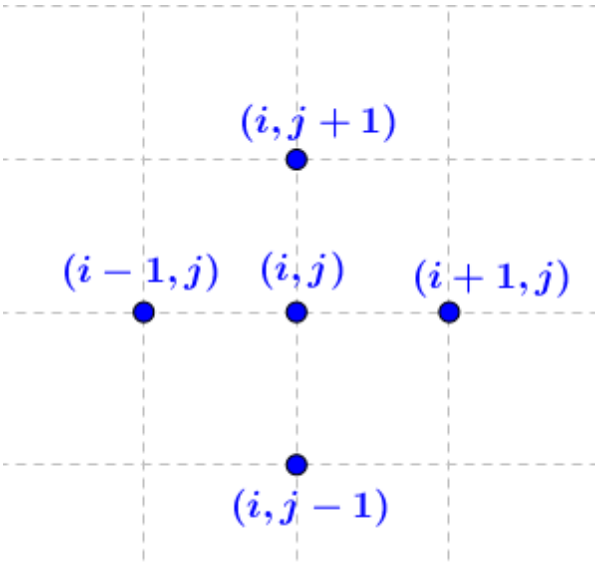
La reconnaissance de formes dans une image est une composante importante de l'analyse d'images. Elle se décompose en plusieurs étapes qui consistent à extraire les contours des objets dans l'image afin de les reconnaitre ou d'en détecter le mouvement. La première de ces étapes est la mise en évidence des contours des objets dans l'image.

Un contour définit la limite d'un objet dans une image. Cette limite est caractérisée par un changement dans l'image : un changement de couleur ou de contraste. Ce changement se traduit dans la valeur des pixels qui sont localisés de part et d'autre de la limite. Nous sommes donc à la recherche d'un moyen de détecter et de localiser un changement. Les mathématiques nous donnent ce moyen sous la forme de la différentiation.

Considérons un pixel p situé en (i , j) dans une image couleur. Ce pixel est-il semblable, de même couleur, que ses voisins ? Si non, quelle est la différence de couleur entre lui et ses voisins ? Est-elle grande, ce qui signifierait qu'il est situé à la limite d'un objet ? Que signifie une "grande" ou une "petite" différence ? Comment la mesurer pratiquement ? C'est ce que nous allons essayer de traduire en algorithme.

Le principe est de récupérer la valeur de chaque pixel avoisinant pour chaque pixel de l'image, ce que je fait dans cette boucle :

```
for i in range(1,colonne-1):
  for j in range(1,ligne-1):
    p1 = im.getpixel((i , j - 1))
    p2 = im.getpixel((i , j + 1))
    p3 = im.getpixel((i - 1 , j))
    p4 = im.getpixel((i + 1 , j))
```



Puis de mesurer la différence, la "distance", entre notre pixel de référence et ses voisins en utilisant une fonction de norme standard :

```
def norme(p1,p2,p3,p4):
  n = sqrt((p1-p2)*(p1-p2) + (p3-p4)*(p3-p4))
  return n
```

Je passe à ma fonction `norme()` les 4 pixels dont je veux évaluer la distance. Après avoir calculé la distance entre mon pixel courant et ses voisins, je décide si ce pixel est sur un contour ou pas à l'aide d'un seuillage. S'il est inférieur au seuil, c'est à dire pas très "distant" de ses voisins, je décide qu'il n'est pas élément d'un contour et je le trace en blanc, sinon, je le trace en noir, comme un contour. C'est le bout de code suivant qui fait ça :

```
if n < seuil:
  p = 255
else:
  p = 0
im.putpixel((i-1,j-1),p)
```

Voilà pour le principe.

Exercice 11 : Ecrire un programme permettant de détecter les contours de l'image `andywarhol2.jpg` avec un seuil égal à 25.