Travailler avec des images en Python

Méthode 2017-2019

Cette section a été mise à jour en fonction des méthodes actuellement utilisées pour manipuler et afficher des images.

Ici l'objectif sera toujours de transformer une image en tableau numpy, pour pouvoir ensuite la manipuler.

Lecture de l'image

PIL permet de lire une image enregistrée localement dans de nombreux formats (il n'est pas forcément disponible sur vos machines). `matplotlib.image` ne permet que de charger des `.png`. Mais vous avez directement un tableau numpy (pas besoin de transformation).

Avec ''matplotlib.image''

```
import matplotlib.image as mpimg
import numpy as np
img = mpimg.imread("monimage.png")
```

Le résultat, `img` est un tableau numpy. C'est parfois un tableau de float (entre 0 et 1) ou parfois un tableau d'octets. On peut être amenés à faire une transformation :

```
if img.dtype == np.float32: # Si le résultat n'est pas un tableau d'entiers
    img = (img * 255).astype(np.uint8)
```

Avec PIL

```
from PIL import Image
import numpy as np
imgpil = Image.open("monimage.png")
img = np.asarray(imgpil) # Transformation de l'image en tableau numpy
```

Affichage de l'image

Matplotlib permet d'afficher une image (si c'est un tableau numpy).

```
import matplotlib.pyplot as plt
plt.imshow(img)
plt.show()
```

L'affichage est assez spartiate, mais permet de se rendre compte de ce qu'on a.

Création d'image et manipulation des pixels

Pour certaines applications, on peut être amené à créer une image à partir de rien (image de pixels noirs). Pour créer un tableau numpy qui pourra contenir une image :

```
import numpy as np
image = np.zeros((100, 200, 3), dtype=np.uint8)
```

Souvent, plutôt que de créer une image vierge, on peut faire une copie d'une image existante :

```
import numpy as np
image2 = np.copy(image)
```

Dans une image courleur, les plans sont généralement au nombre de 3 : Rouge, Vert et Bleu. Il peut parfois y avoir un 4^{e} plan qui correspond à la transparence. Pour obtenir la taille le long de chaque dimension :

```
>>> img.shape
(291, 437, 3)
```

L'image précédente a donc 291 lignes, 437 colonnes et 3 plans. S'il y avait eu un quatrième plan, on aurait pu l'enlever ainsi :

```
img = img[:,:,:3]
```

On peut connaître la valeur d'un pixel en donnant la ligne et la colonne :

```
>>> img[100, 120]
array([39, 40, 34], dtype=uint8)
```

La valeur du rouge est donc 39, celle du vert 40 et celle du bleu 34.

On modifie un pixel ainsi :

```
img[100, 120] = (56, 120, 355)
```

Voici un exemple de fonction qui prend une image en paramètre, et renvoie une image modifiée (les composantes vertes et bleues sont annulées). Elle peut servir de base pour construire des filtres :

```
def filtre_rouge(img_orig):
    im = np.copy(img_orig) # On fait une copie de l'original
    for i in range(im.shape[0]):
        for j in range(im.shape[1]):
            r, v, b = im[i, j]
            im[i, j] = (r, 0,0)
    return im
```

Le parti pris, dans l'exemple qui précède, est de ne pas modifier l'image d'origine, mais de créer une nouvelle image qu'on renvoie.

Sauvegarde des images

Pour enregistrer des images, on utilise une méthode similaire au chargement. Matplotlib permet ainsi de sauvegarder directement un tableau numpy au format PNG uniquement. PIL permettra de sauvegarder dans n'importe quel format, pourvu qu'on ait transformé le tableau numpy en Image PIL.

Avec Matplotlib

```
import matplotlib.image as mpimg
mpimg.imsave("resultat.png", img)
```

Avec PIL

Avec PIL, il faut s'abord transformer le tableau numpy en image PIL.

```
from PIL import Image
imgpil = Image.fromarray(img) # Transformation du tableau en image PIL
imgpil.save("resultat.jpg")
```

Anciennes méthodes

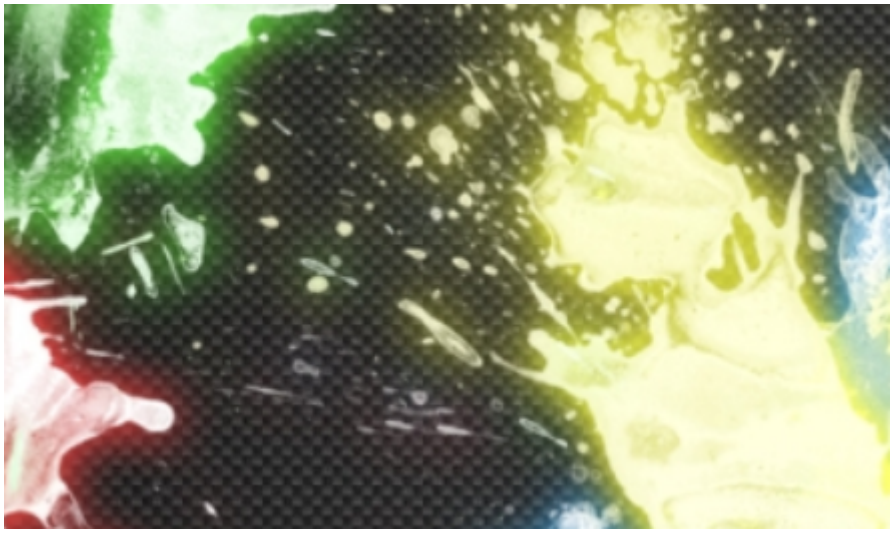
On détaille dans la suite d'autres méthodes et modules pour manipuler des images.

Nous allons ici nous focaliser sur 4 méthodes :

- imageio qui est fourni en standard avec Pyzo et qui permet très simplement de charger une image et d'accéder aux valeurs des pixels.
- `Imagewindow`, un module *maison* qui permet d'afficher des images et d'interagir avec l'utilisateur par le biais de la souris ou du clavier
- Pillow qui est la version Python3 du très célèbre module PIL
- scipy (numpy, scipy, matplotlib, le trio pour scientifiques...)
- pygame qui entre aux choses possèdes des fonctions pour manipuler les images

Le choix d'une méthode particulière dépend de ce qu'on veut obtenir à la fin, de ses propres habitudes etc... Demandez conseil à l'encadrant(e) si vous hésitez.

Pour faire fonctionner les exemples, téléchargez cette image :



Module imageio

Pour vérifier si imageio est installé,entrez simplement `import imageio` dans un shell Python.

- imageio est fourni avec Pyzo
- il peut être installé séparément : imageio sur Pypi
- le module imageio est documenté sur readthedocs

imageio utilise le module numpy qui permet entre autre de manipuler des tableaux homogènes très rapidement.

On dispose donc entre autres :

- d'une fonction pour lire un fichier image et récupérer un tableau numpy : `imageio.imread`
- d'une fonction pour écrire un tableau numpy (au bon format) dans un fichier image : `imageio.imsave`

Exemple de programme :

```
import imageio
im = imageio.imread("peint.png")
# La taille de l'image est accessible dans le triplet (largeur, hauteur, nb composantes) suivant :
print(im.shape)
# On peut alors connaître le triplet de couleur du pixel (0,0) (coin supérieur gauche)
print(tuple(im[0][0]))
# On peut chager la couleur d'un point en donnant 3 composante RGB :
im[0,0] = (255, 0, 0)
# On peut modifier simplement une composante :
im[0,1][0] = 0 # on enlève le rouge
# Puis on peut sauvegarder l'image résultante
imageio.imsave("img2.png", im)
```

Le module `imageio` n'est pas fait pour afficher les images. Cette étape peut être réalisée à part, sur le fichier. On peut toutefois faire appel à matplotlib, dans le cas où un affichage un peu plus *interactif* est nécessaire. Matplotlib permet en effet d'afficher les tableaux numpy tels qu'ils sont renvoyés pas `imageio`.

```
import imageio
import matplotlib.pyplot as plt
im = imageio.imread("peint.png")
plt.imshow(im)
plt.show()
```

Le module `ImageWindow` développé en interne permet aussi d'afficher les images provenant de `imageio`. (voir la section sur `ImageWindow`)

Module ImageWindow

Le module `ImageWindow` permet d'afficher et d'interagir avec des images. Ces images peuvent provenir de tableaux `numpy`, du module `imageio` ou de `Pillow`. Pour l'utiliser, il faut le télécharger ici : PYTHON/ImageWindow.py. La version actuelle (au moment de la rédaction de cette page) est la 1.6 (il se peut que vous ayez une version plus récente) :

```
>>> import ImageWindow
>>> print(ImageWindow.version())
1.6
```

Attention, il faut **au moins** la version 1.6 pour fonctionner avec PyQt5. Et c'est PyQT5 qui est actuellement (2016-2017) installé dans les salles info de l'école.

Voici les principales fonctions/méthodes disponibles :

- `win = ImageWindow.Visu()` : ouvre une fenêtre de visualisation
- `win.setImage(image)` : affiche l'image dans la fenêtre (image peut être une image lue par `imageio`, une image `Pillow` ou un tableau `numpy`
- `win.run()` : lance la boucle des événements. Cette ligne est nécessaire dans le cas d'un programme lancé avec l'interpréteur python, mais n'est pas recommandée si on utilise IEP (Pyzo), ce qu'on supposera dans la suite.

Pour interagir avec la fenêtre, on doit *enregistrer* des fonctions (appelées *callbacks*) qui seront appelées lors d'un clic souris ou de l'utilisation du clavier. Le prototype des fonctions callback (*la liste des paramètres*) est fixé et vous devez vous y tenir. Voici un exemple de programme qui réagit aux clics souris :

test_ImageWindow2.py

```
from ImageWindow import Visu
import imageio
```

```
def myclic(img,x,y,b):
    if b == 1:
        img[y,x] = (255, 0, 0)
    elif b == 3:
        img[y,x] = (0, 0, 255)


win = Visu() # Création de la fenêtre
img = imageio.imread("peint.png") # Chargement d'une image avec imageio
win.setImage(img) # Affiche l'image chargée dans la fenêtre
win.register("mouse",myclic) # Appelle myclic en cas de clic souris
win.setAutoupdate(True) # Met l'image à jour à chaque clic
# Pas nécessaire depuis un shell interactif comme IEP, mais
# il faut le mettre dans une appli indépendante.
#  win.run()
```

Certains images ont un canal alpha, c'est une troisième composante qui représente le niveau de transparence du pixel. Si c'est le cas, chaque pixel de l'image est un *tuple* de 4 valeurs et non 3. Pour savoir si une image a ou non un canal alpha, il suffit d'afficher la valeur d'un des pixels et de compter le nombre de valeurs renvoyées.

Si une image a un canal alpha, **tous les pixels** sont définis par 4 valeurs, et non 3. En conséquence, le programme qui précède ne marchera que pour une image sans canal alpha.

Il est possible d'enlever le canal alpha d'une image gérée par `imageio` en une seule instruction :

```
img_sans_alpha = img[:,:,:3]
```

L'exemple précédent permet d'interagir avec la souris sur l'image (on a enregistré une callback associée au clic souris). On peut faire de même pour interagir avec le clavier :

```
def kbd(t):
   print("Touche : ",t)

win.register('keyboard', kbd)
```

Lors de la réalisation d'un programme avec animation et tâche de fond (automate cellulaire par exemple), et si on ne fait pas appel à `run()`, il faut régulièrement appeler la méthode `win.flush()` pour traiter les événements en attente dans la fenêtre (comme le réaffichage par exemple).

`ImageWindow` est *compatible* avec PIL (qu'on peut utiliser en remplacement de `imageio`).

Le module ImageWindow offre d'autre possibilités (essayez `import ImageWindow; help(ImageWindow)`)

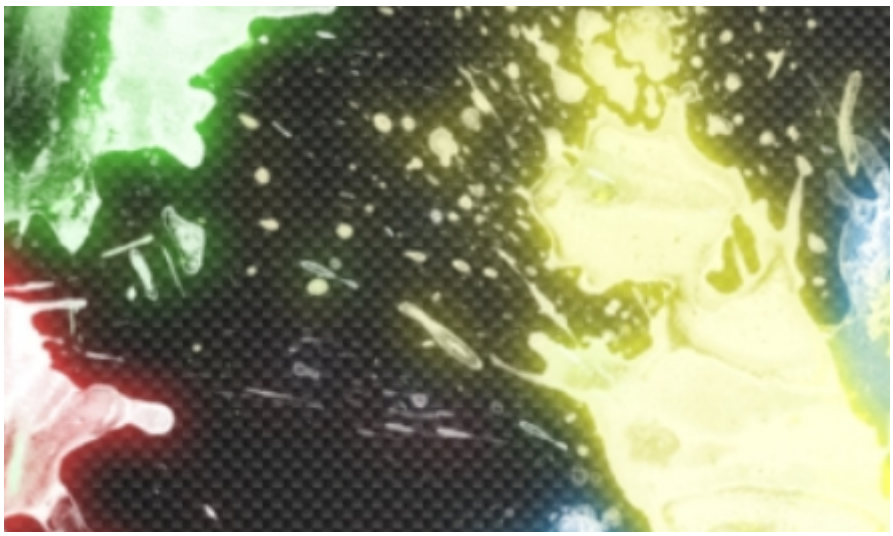
Pillow/PIL

Le module phare de traitement de l'image en Python, PIL (Python Image Library) n'a pas tout de suite été porté en Python 3. Comme c'est souvent le cas dans le monde du libre, un fork pour Python 3 est apparu : Pillow. Pillow et PIL s'utilisent donc pratiquement de la même façon.

La documentation complète de Pillow est accessible ici :
http://pillow.readthedocs.org/en/latest/guides.html

D'autres informations peuvent être trouvées dans la documentation de PIL :
http://effbot.org/imagingbook/

Dans les exemples suivants, on utilise cette image :



Désigner une image et l'afficher

```
>>> from PIL import Image
>>> im = Image.open("peint.png")
>>> im.show()
```

L'objet retourné, `im` peut être interrogé :

```
>>> print(im.size, im.format)
(442, 262) PNG
```

Lire et modifier un pixel

Un pixel de l'image (RGB) peut être modifié en donnant ses coordonnées et ses nouvelles composantes. La modification peut alors être visualisée avec `show()` :

```
>>> im.putpixel((100,100),(255,0,0))
```

Inversement, on peut connaître la valeur d'un pixel ainsi :

```
>>> im.getpixel((100,100))
(255,0,0)
```

Utilisation avec ImageWindows

On peut utilsier ImageWindow pour afficher (et modifier) les images chargées avec PIL :

test_ImageWindow.py

```
from ImageWindow import Visu
from PIL import Image

def myclic(img,x,y,b):
    if b == 1:
        img.putpixel((x,y),(255,0,0))
    elif b == 3:
        img.putpixel((x,y),(0,0,255))


win = Visu() # Création de la fenêtre
img = Image.open("peint.png") # Chargement d'une image avec PIL
win.setImage(img) # Affiche l'image chargée dans la fenêtre
win.register("mouse",myclic) # Appelle myclic en cas de clic souris
win.setAutoupdate(True) # Met l'image à jour à chaque clic
# Pas nécessaire depuis un shell interactif comme IEP, mais
# il faut le mettre dans une appli indépendante.
#  win.run()
```

Récupérer les valeurs de tous les pixels

La fonction `getdata` permet d'obtenir une séquence (linéaire) contenant toutes les valeurs des couleurs des pixels. Cette séquence peut être indicée comme une liste. Si d'autres opérations sont nécessaires, on peut aussi la *convertir* en liste (mais les opérations seront alors moins rapides).

```
>>> data = im.getdata()
>>> type(data)
ImagingCore
>>> data[0]
(100, 149, 87)
>>> l = list(data)
>>> l[0:5]
[(100, 149, 87), (115, 163, 103), (110, 159, 97), (94, 147, 81), (76, 133,
```

```
61)]
```

L'opération inverse est `putdata` et elle prend en paramètre une séquence (`list` ou `ImagingCore` par exemple) :

```
>>> l[0:200] = [(255,255,0)]*200
>>> im.putdata(l)
>>> im.show()
```

Fonctions de dessin avancées

Il est possible de *dessiner* sur une image. Pour cela, on commence par obtenir une instance de `ImageDraw`:

```
>>> from PIL import ImageDraw
>>> imdr = ImageDraw.Draw(im)
```

Puis on dessine sur ce nouvel objet (liste des fonctions disponibles : `help(ImageDraw)`):

```
>>> imdr.line([(0,0),(200,150)], (0,200,255), width=5)
>>> im.show()
```

Les modifications sont immédiatement répercutées sur l'image.

Lorsqu'un objet de type `ImageDraw` n'est plus nécessaire, on peut le supprimer de la mémoire :

```
>>> del imdr
```

Enregistrer une image

Pour enregistrer une image dans un fichier :

```
>>> im.save("resultat.png", format="png")
```

Si le paramètre format n'est pas mentionné, PIL le choisit automagiquement en fonction de l'extension donnée.

Créer une image à partir de rien

La commande suivante crée une image de taille 512×512, contenant 3 plans de couleurs, et initialement entièrement blanche :

```
>>>  im = Image.new("RGB", (512, 512), "white")
```

Convertir une image en tableau numpy

La conversion d'une image PIL en tableau numpy est immédiate :

```
import numpy as np
>>> pic = np.array(im)
```

L'object obtenu, `pic` est tridimensionnel. On accède à la composante 1 (vert) du pixel (100,200) par :

```
>>> pic[100,200,1]
```

La transformation inverse peut être faite ainsi :

```
>> im2 = Image.fromarray(pic)
```

Interagir avec l'image

L'affichage d'une image avec Pillow utilise des outils externe d'affichage d'image. On ne peut donc pas, par ce moyen, interagir avec l'image (en cliquant dessus par exemple).

Pour avoir un affichage interactif, on peut utiliser le module `ImageWindow` développé en interne et détaillé dans ...

Trio Numpy, Scipy, matplotlib

Scipy possède un module `ndimage` : Le tout est immédiat :

```
from scipy import ndimage
import matplotlib.pyplot as plt

im = ndimage.imread("peint.png")
plt.imshow(im)
plt.show()
```

L'objet `im` étant un tableau au sens de Numpy, on a un accès direct aux valeurs de pixels.

Différentes conversions

imageio vers QPixmap

Cette méthode n'est plus disponible.

D'un tableau 3D `numpy` ou d'une image type `imageio`, vers un `QPixmap` affichage par `PySide`, on

peut utiliser la méthode `imageio_to_pixmap` du module `ImageWindow`:

```
import ImageWindow
import imageio

im = imageio.imread('example.png')
pix = ImageWindow.imageio_to_pixmap(im) # On récupère un QPixmap
```

La fonction prend n'importe quel tableau 3D numpy d'octets :

```
import numpy
import ImageWindow

im = np.zeros((128,128,0), dtype=np.uint8
pix = ImageWindow.imageio_to_pixmap(im) # On récupère un QPixmap
```

Dans le cas où on ne souhaite pas utiliser ImageWindow, voici le code de la fonction qui réalise la transformation :

```
import imageio
from PySide import QtGui
import numpy as np

def imageio_to_pixmap(array):
    newarray = np.array(array,dtype='uint32')
    tailley, taillex = array.shape[0], array.shape[1]
    cr = newarray[:,:,0]
    cg = newarray[:,:,1]
    cb = newarray[:,:,2]

    b = (255 << 24 | cr << 16 | cg << 8 | cb).flatten()
    image = QtGui.QImage(b, taillex , tailley, QtGui.QImage.Format_ARGB32)
    pixmap = QtGui.QPixmap(image)
    return pixmap
```

pygame

A priori, nous n'utiliserons plus ce module en TP.

Pour vérifier si pygame est installé, entrez simplement `import pygame` dans un shell Python.

- pygame n'ets pas fourni en standard. On peut le télécharger sur le site officiel

pygame permet entre autres choses de charger des images à partir de fichiers, et de les afficher

(pygame fournit un système de fenêtres). Il est aussi possible d'accéder aux piels des images. Les fonctions nécessaire pour réaliser tout ça sont :

- `surf=pygame.image.load("img.png")` charge une image et renvoie une `Surface`.
- `screen.blit(surf,(0,0))` affiche la surface `surf` sur la fenêtre `screen` (dans le coin supérieur gauche) (penser à faire un `pygame.display.flip()` pour forcer l'affichage.
- `img=pygame.surfarray.array3d(surf)` renvoie un tableau tridimensionnel à partir d'une surface. Le pixel 10,30 ainsi obtenu est accessible en consultant la valeur `img[10][30]`, ou `img[10,30]`, qui est une liste des 3 composantes *rouge*, *vert*, *bleu*.
- `surf2=pygame.surfarray.make_surface(img)` permet, inversement, de créer une surface à partir d'un tableau tridimensionnel. Cette surface pourra ensuite être affichée par `screen.blit`

Voici les pages pertinentes dans la documentation de PyGame :

- SurfArray
- Image

pygame permet de gérer les événements dans la fenêtre d'affichage (clic souris, touches clavier). Pour l'utiliser, il est nécessaire d'écrire un boucle de gestion de ces événements (voir dans l'exemple). pygame permet de faire des choses plus compliquées et plus perfectionnées que simplement changer quelques pixels dans une image. La contrepartie de cette puissance est une utilisation un peu technique et délicate.

```
import pygame
import pygame.gfxdraw
import math
import os
import random


def modifie(screen) :
    # Ajoute du rouge à quelques points choisis aléatoirement
    pixels_orig = pygame.surfarray.array3d(screen)
    pixels_dest=pixels_orig.copy()
    for n in range(256):
        i = random.randint(pixels_dest.shape[0])
        j = random.randint(pixels_dest.shape[1])
        v = pixels_orig[i,j]
        v = [255, v[0], v[1]]
        pixels_dest[i,j]=v
    surface_dest=pygame.surfarray.make_surface(pixels_dest)
    screen.blit(surface_dest,(0,0))
    print("Fini")
    pygame.display.flip()

# ====================== PROGRAMME PRINCIPAL ====================
#  Initialisation de la bibliothèque
pygame.init()
#  Ouverture d'une fenêtre 512 x 512 (nommée screen)
screen=pygame.display.set_mode([512,512])
```

```
# Chargement de l'image à partir d'un fichier
surface_fichier = pygame.image.load('peint.png')
# Affichage de l'image dans la fenêtre
screen.blit(surface_fichier,(0,0))
pygame.display.flip()


#  Boucle des événements
#  Les événements gérés sont : appui sur m, sur s ou fermeture de la fenêtre
while True :
    event = pygame.event.poll()
    if event.type == pygame.QUIT: break
    if event.type == pygame.KEYDOWN :
        key=event.dict['unicode']
        if key == 'm' : modifie(screen)
        elif key == 's' : pygame.image.save(screen,"img_tmp.png")
        else : print('Commande non reconnue')

    pygame.quit()
```