



Ecole Polytechnique de l'Université François Rabelais de Tours

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

[polytech:univ-tours:fr](mailto:polytech:univ-tours:fr)



# Rapport – Analyse d'Images

2020-2021

LIU Yuanyuan

GUO Xiaoqing

07/12/2020

# Table des matières

Partie 0 : Prise en main de Python + OpenCV .....	1
Partie 1 : Type d'images et statistiques images .....	4
Partie 2 : Filtrage, convolution et détection de contours.....	7
Partie 3 : Images binaires et opérations entre images.....	16
Partie 4 : Image Tagging .....	26
1. Environment setup.....	26
2. Algorithmes .....	28
2.1 Reconnaissance des couleurs principales .....	28
2.2 Reconnaissance de scène (type d'environnement et scène spéciale) .....	29
2.3 Reconnaissance d'objet principal au contenu .....	31
2.4 Reconnaissance de la netteté de l'image .....	35
2.5 Reconnaissance de la luminosité de l'image.....	36
2.6 Reconnaissance de la complexité de l'image.....	38
3. Run et Résultat .....	38

## Partie 0 : Prise en main de Python + OpenCV

### 1. Image\_processing.py

Code Source avec commentaires :

```
# Afficher des info de Python
print("Python version")
print_(sys.version)
print("Version info.")
print_(sys.version_info)

# Afficher la version de opencv
print("OPENCV Version =", cv2._version__)

# Obtenir le path du fichier python
rep_cour = os.getcwd()
print(rep_cour)

# Lire une image et changer le couleur en gris au travers de cvtColor
img = cv2.imread('data/boats.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

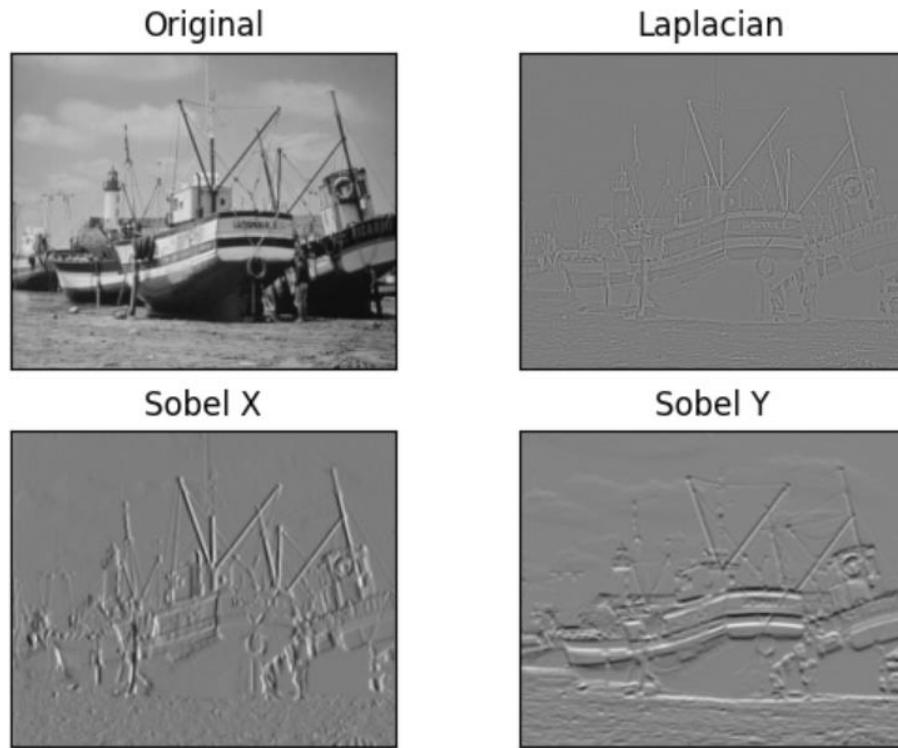
# L'opérateur Laplacian est utilisé pour la détection des contours d'une image
laplacian = cv2.Laplacian(img, cv2.CV_64F)

## Le gradient de Sobel est un algorithme pour détecter les contours d'une image avec un noyau
## ici la taille de noyau est 5*5
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)

# Visualiser les images dans un graphe
plt.subplot(2,2,1), plt.imshow(img, cmap='gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2), plt.imshow(laplacian, cmap='gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3), plt.imshow(sobelx, cmap='gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4), plt.imshow(sobely, cmap='gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```

Résultat :

```
D:\download\python\python-3.7.6\python.exe "D:/yuanyuan/2020-2021 S2/analyse d'images/Code Source/Images2020/Images2020/image_processing.py"
Python version
3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)]
Version info.
sys.version_info(major=3, minor=7, micro=6, releaselevel='final', serial=0)
OPENCV Version = 4.4.0
D:\yuanyuan\2020-2021 S2\analyse d'images\Code Source\Images2020\Images2020
```



## 2. video\_processing.py

Code Source avec commentaires :

```
# retourner une image
def frame_processing(imgc):
    return imgc

# Lire le vidéo et enregistrer dans une variable cap
cap = cv2.VideoCapture('data/jurassicworld.mp4')

# Ne pas sortir le boucle while jusqu'à terminer le programme
while (True):

    # Capturer, décoder et retourner la frame suivante
    # ret indique la frame suivante est bien capturée ou pas
    # frame est une matrice des pixels d'image
    ret, frame = cap.read()
    print(ret)
    print(frame)
```

```

# True, la frame est bien capturée
if ret == True:
    # Copier la frame et changer le couleur RGB en gris utilisant cvtColor
    img = frame.copy()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Obtenir la frame à afficher
    gray = frame_processing(gray)

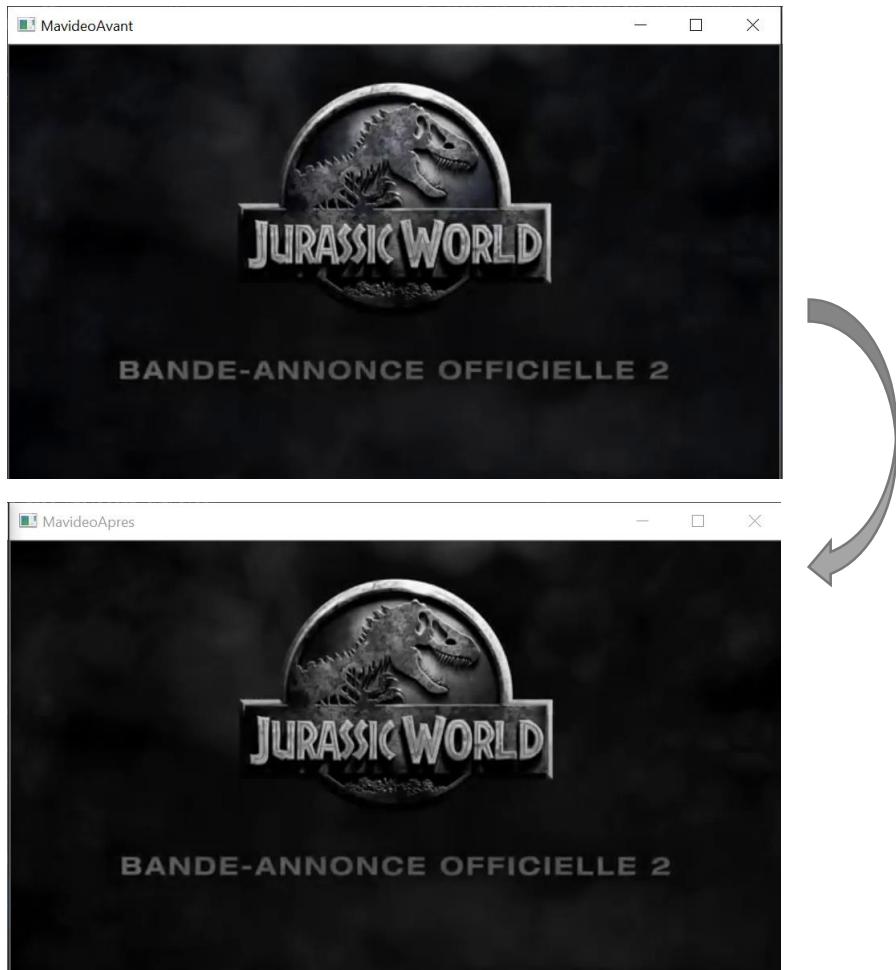
    # Affiche la frame originale et la frame gris dans la fenêtre
    cv2.imshow('MavideoAvant', frame)
    cv2.imshow('MavideoApres', gray)

else:
    print('video ended')
    break

if cv2.waitKey(1000) & 0xFF == ord('q'):
    break

```

Résultat :



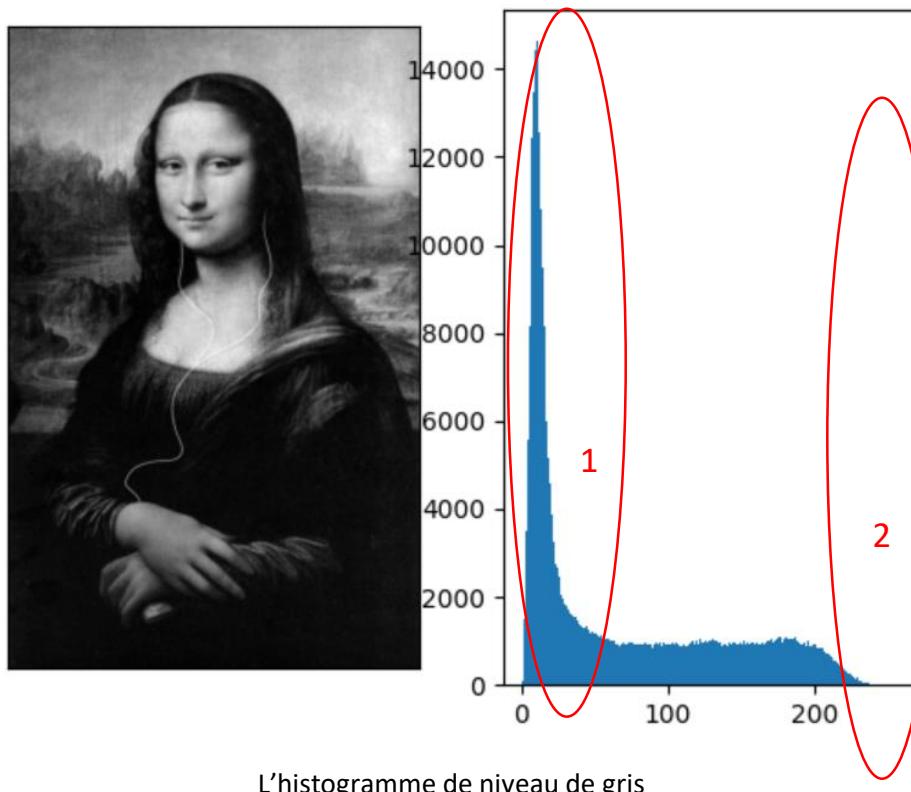
## Partie 1 : Type d'images et statistiques images

Q1. L'utilisation de la méthode `plt.hist` pour afficher le diagramme de histogramme.

```
# Calculer la graphe d'histogramme représentant la distribution d'intensité d'une image
def img_hist(img):
    plt.subplot(121)
    plt.imshow(img, 'gray')
    plt.xticks([])
    plt.yticks([])
    plt.subplot(122)
    plt.hist(img.ravel(), 256, [0, 256])
    plt.show()
```

Voici est la figure de l'histogramme de l'image *lisa.png* de niveau de gris :

- L'axe vertical représente le nombre de pixels et l'axe horizontal représente le niveau de gris entre 0 (noir) et 255 (blanc) ;
- On voit que la plupart des pixels sont concentrés près de 0 ;
- Donc les pixels de la région 1 sont sombres et ceux de la région 2 sont lumineux.



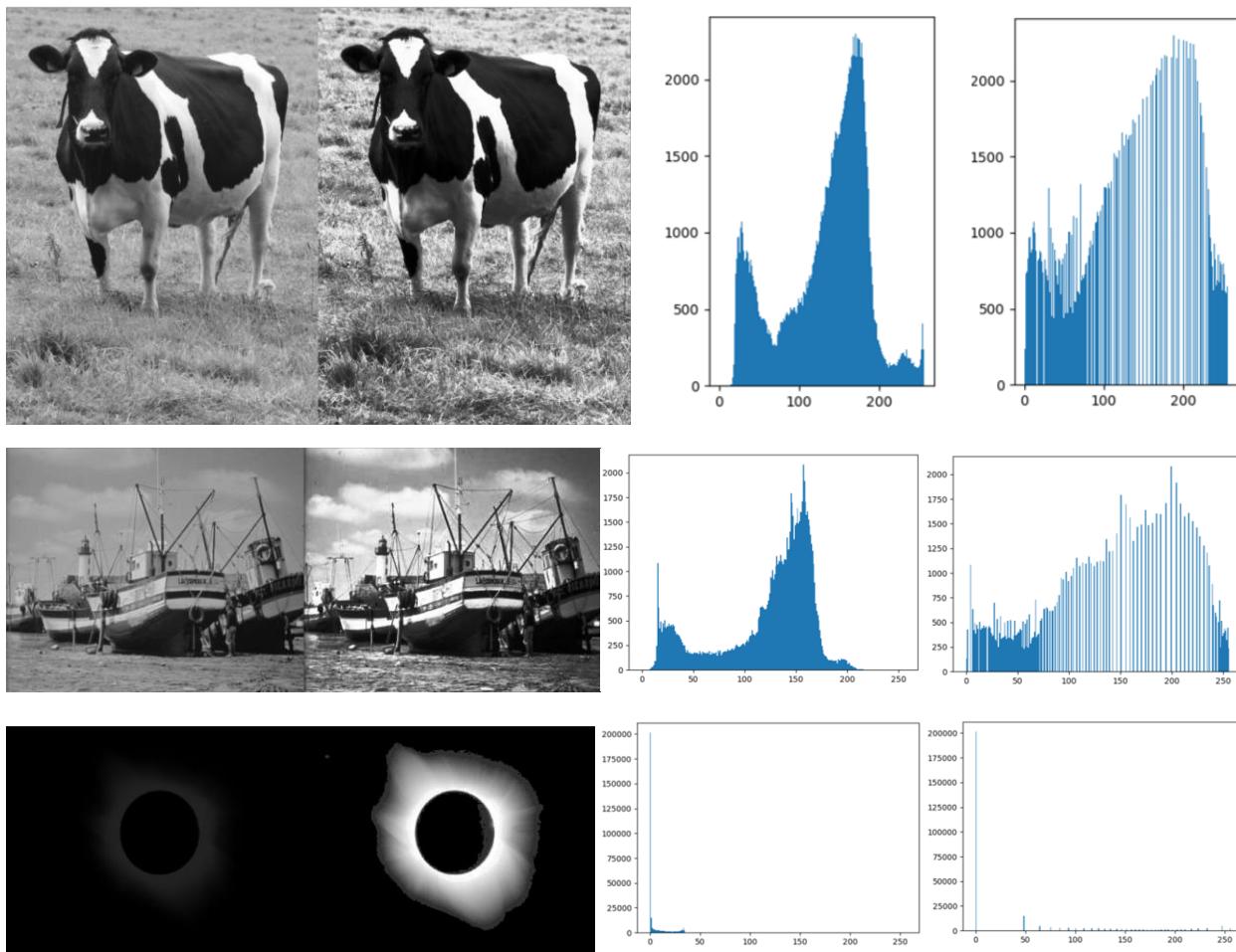
Q2. L'égalisation d'histogramme est une méthode d'ajustement du contraste d'une image numérique qui utilise l'histogramme.

```
# Afficher l'image d'égalisation
def img_equ(img):
    global equ
    equ = cv2.equalizeHist(img)
    res = np.hstack((img, equ))

    cv2.imshow('img', res)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

Ci-dessous est un exemple d'égaliser d'un histogramme de niveau de gris :

- D'abord, on affiche l'histogramme de niveau de gris pour les images de paysage et soleil, ici nous choisirons trois images de vache, bateau et soleil ;
- Puis on égalise l'histogramme et on compare les deux images avant et après l'égalisation d'histogramme ;
- Enfin on génère l'histogramme de niveaux de gris après égalisation.



Les deux images avant et après l'égalisation d'histogramme

L'histogramme de niveaux de gris avant et après égalisation

Selon les résultats que nous avons obtenus, nous pouvons savoir que les intensités peuvent être mieux réparties sur l'histogramme grâce à cet ajustement. Cela permet pour les zones à faible contraste local d'obtenir un contraste plus élevé.

<https://boowiki.info/art/traitement-d-image-numerique/egalisation-d-histogramme.html>

## Partie 2 : Filtrage, convolution et détection de contours

Q1: Ouvrez lisa.png (joconde) et convertissez-les en niveaux de gris. Appliquez une seule fois des filtres moyenneur de taille/rayon variable:

1. 5X5            2. 9X9            3. 15X15

Que donnerait l'application d'un filtre moyenneur dont la dimension serait égale à celle de l'image ?

[https://docs.opencv.org/3.4.2/d2/d96/tutorial\\_py\\_table\\_of\\_contents\\_imgproc.html](https://docs.opencv.org/3.4.2/d2/d96/tutorial_py_table_of_contents_imgproc.html)

Code : P2\_Q1\_filtre\_moyenneur.py

- Le filtre moyenneur est une opération de traitement d'images utilisée pour réduire le bruit dans une image et/ou flouter une image.
- On choisit `cv2.blur()` pour faire le filtre moyenneur

`cv2.blur(img, (i, i))` : fait une moyenne dans un voisinage  $i \times i$  (matrice de convolution avec tous les coefficients identiques et leur somme qui vaut  $i^2$  et renvoie l'image résultat).

```
img = cv2.imread('../data/lisa.png', 0)
img5 = cv2.blur(img, (5, 5))
img9 = cv2.blur(img, (9, 9))
img15 = cv2.blur(img, (15, 15))
```

```
print("image_shape: ", img.shape)
img_height = img.shape[0]
img_width = img.shape[1]
new_dim = min(img_width, img_height)
img_max = cv2.blur(img, (new_dim, new_dim))
```

```
plt.subplot(2, 3, 1), plt.imshow(img, cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(2, 3, 2), plt.imshow(img5, cmap = 'gray')
plt.title('5*5'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(2, 3, 3), plt.imshow(img9, cmap = 'gray')
plt.title('9*9'), plt.xticks([]), plt.yticks([])
```

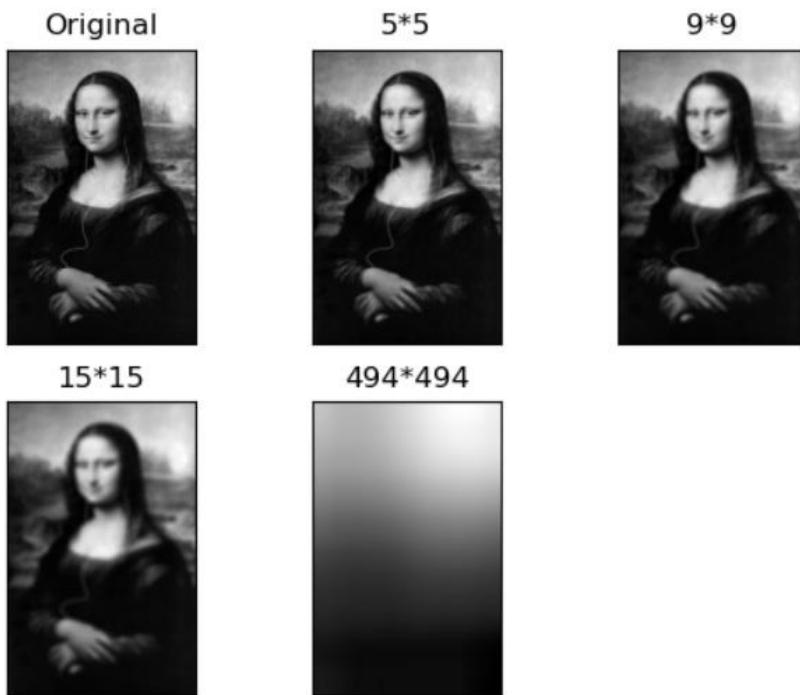
```
plt.subplot(2, 3, 4), plt.imshow(img15, cmap = 'gray')
plt.title('15*15'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(2, 3, 5), plt.imshow(img_max, cmap = 'gray')
plt.title(str(new_dim) + '*' + str(new_dim)), plt.xticks([]), plt.yticks([])
```

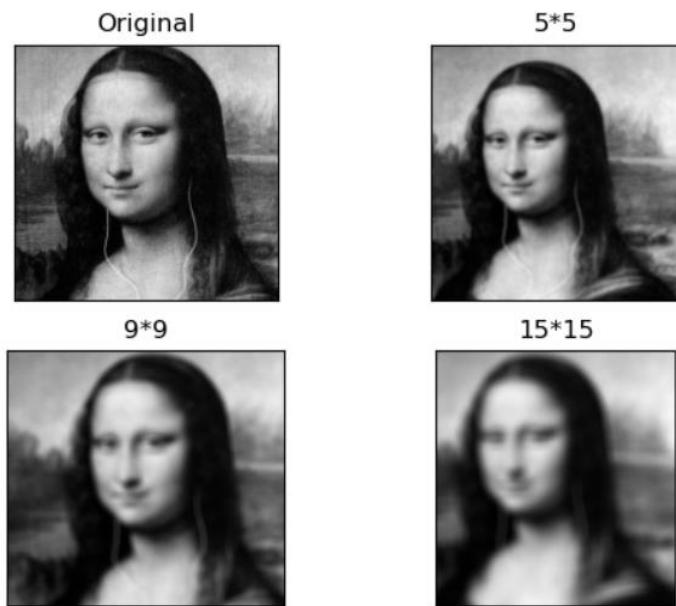
● `plt.show()`

Réultat :

●



- En zoomant, on peut voir en détail les effets du filtre; le bruit clairement visible dans le arrière-plan a bien été réduit mais les détails du visage sont floutés :



- Quand la dimension serait égale à celle de l'image :
- `image_shape: (768, 494)`

494\*494



Le fonctionnement est principale de reduire les détails presque indépendants dans l'image, c'est-à-dire la zone de pixel qui est plus petite que la taille de la taille(masque) de filtre.  
Donc, lorsque la taille du filtre est égale / supérieuse à la dimension de l'image, l'image est complètement floue (les niveaux de gris sont environ complètement concentrés).

Q2: Complétez le code fourni en partie 0 afin d'appliquer différents traitements de filtrage (flou gaussien, masque median, ...). Insérez certains résultats dans votre rapport et commentez-les afin de démontrer que vous avez compris le mécanisme de filtrage d'images

Code : P2\_Q2\_filtres.py

```
img = cv2.imread('..../data/yache.jpg',0)
source = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Filter box + Normalise
img_box_n = cv2.boxFilter(source, -1, (5, 5), normalize = 1)

# Filter box + not Normalise
img_box = cv2.boxFilter(source, -1, (5, 5), normalize = 0)

# Filter Gaussian
img_gaussian = cv2.GaussianBlur(source, (5, 5), 0)

# Filtre moyenneur
img_moyenneur = cv2.blur(source, (5,5))

# Filtre mmedian
img_mmedian = cv2.medianBlur(source, 5)
```

Résultat :





- Filtrage boîte :
  - Les noyaux de convolution du filtrage boîte et du filtrage moyen sont fondamentalement les mêmes, la différence est si la normalisation[`normalize = 1`] est nécessaire ou pas.
  - fonction du filtrage boîte :  
`def boxFilter(src, ddepth, ksize, dst=None, anchor=None, normalize=None, borderType=None)`
  - Lorsque "normalizez" est vrai[1], il a besoin de traiter avec valeur de moyenne.  
 Lorsque "normalizez" est faux[0], il n'a pas besoin de traiter avec valeur de moyenne. En fait, il s'agit de la somme des autres pixels, il est donc facile de se produire un dépassement. Quand il se passe, les parties d'images sont blanc et la valeur de pixel correspondante est de 255.
- Filtrage gaussien :
  - `cv2.GaussianBlur(img, (5, 5), 0)` : filtre gaussien de taille 5 x 5 et d'écart-type 0.
  - La différence avec les autres filtrages: le noyau de convolution est remplacé par un noyau gaussien.
  - On peut voir que l'image traitée n'est pas aussi floue que le traitement du filtre moyen.
- Filtrage mdeian :
  - `cv2.medianBlur(img, 5)` : utilise la médiane sur un voisinage 5 x 5 et renvoie l'image résultat.
  - Le filtrage médian est une méthode de traitement d'image non linéaire, qui peut préserver les informations de limite lors du débruitage.
  - Le noyau pour le filtrage médian est un nombre impair supérieur à 1 comme le noyau pour le filtrage gaussien.
  - On voit clairement que le filtre médian obtient la réduction la plus élevée après le débruitage de l'image d'origine.

Q3: Effectuez les actions suivantes et reportez dans votre rapport les coefficients de filtres utilisés et les images filtrées obtenues :

- Ouvrez l'image zebre.jpg et convolez la avec un filtre permettant de faire ressortir les traits horizontaux
- Ouvrez l'image suzan.jpg et convolez la avec un filtre permettant de faire ressortir les traits verticaux

Expliquez les masques utilisés

1. pour ressortir les traits horizontaux:

Noyau de convolution :

$$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

Code :

On utilise la fonction *filter2D()* dans opencv pour convoler l'image.

```
def trait_horizontaux():
```

```
img = cv2.imread('../data/zebre.jpg', 0)

# Noyau de convolution
con = np.array([[1, 1, 1],
                [0, 0, 0],
                [-1, -1, -1]])

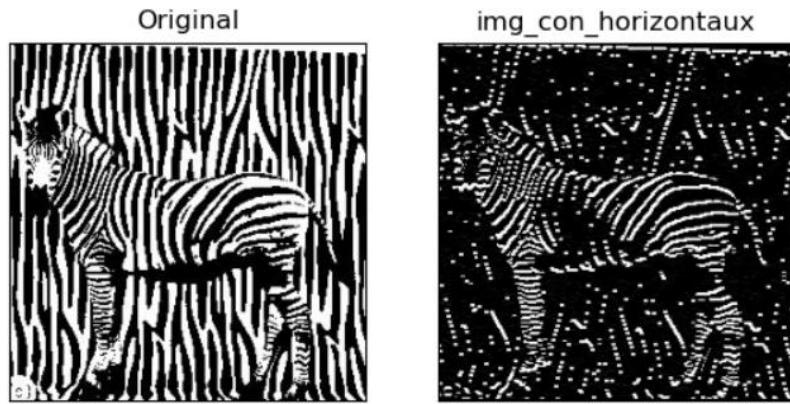
img_con = cv2.filter2D(img, -1, con)

plt.subplot(1, 2, 1), plt.imshow(img, cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(1, 2, 2), plt.imshow(img_con, cmap = 'gray')
plt.title('img_con_horizontaux'), plt.xticks([]), plt.yticks([])

plt.show()
```

Résultat :



2. Pour ressortir les traits verticaux:

Noyau de convolution :

```
1 0 -1
1 0 -1
1 0 -1
```

Code :

On utilise la fonction *filter2D()* dans opencv pour convoluer l'image.

```
def trait_verticaux():

    img = cv2.imread('../data/suzan.jpg', 0)

    # Noyau de convolution
    con = np.array([[1, 0, -1],
                   [1, 0, -1],
                   [1, 0, -1]])

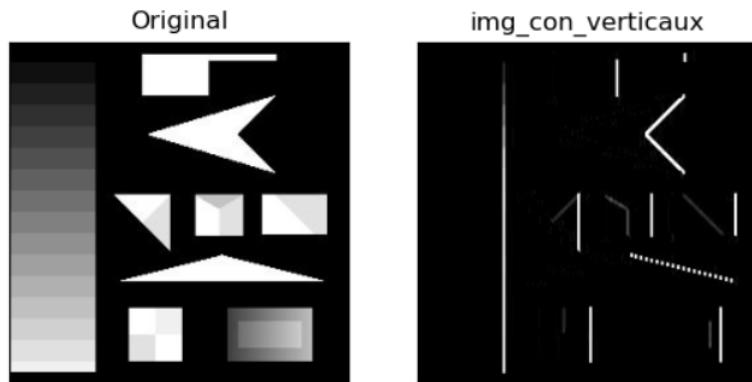
    img_con = cv2.filter2D(img, -1, con)

    plt.subplot(1, 2, 1), plt.imshow(img, cmap = 'gray')
    plt.title('Original'), plt.xticks([]), plt.yticks([])

    plt.subplot(1, 2, 2), plt.imshow(img_con, cmap = 'gray')
    plt.title('img_con_verticaux'), plt.xticks([]), plt.yticks([])

    plt.show()
```

Résultat :



Q4: Complétez le code fourni en partie 0 afin d'appliquer différents algorithmes de détection de contours (Sobel, Laplacien, Canny ...) à la frame courante avant affichage. Insérez certains résultats dans votre rapport et commentez-les afin de démontrer que vous avez compris le mécanisme de filtrage d'images.

Pour l'implémentation du code, voir la partie 0 de ce rapport.

**Fichier :** /TP\_2020/partie2\_3/P2\_Q4\_image\_processing.py

**Code :**

Test image : / TP\_2020/data/boats.jpg

```
# Lire une image et changer l'image en niveau de gris au travers de cvtColor
# img = cv2.imread(path)

img = cv2.imread('../data/boats.jpg',0)
# img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# L'opérateur Laplacian est utilisé pour la détection des contours d'une image
laplacian = cv2.Laplacian(img, cv2.CV_64F)

# Le gradient de Sobel est un algorithme pour détecter les contours d'une image avec un noyau
# ici la taille de noyau est 5*5
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)

# Canny Edge Detection
canny = cv2.Canny(img, 100, 200)
```

Algorithmes de détection de contours : Sobel, Laplacien, Canny

Description:

- Laplacien :

Le détecteur de bord laplacien n'utilise qu'un seul noyau. Il calcule les dérivées du second ordre en une seule passe. Un noyau utilisé dans cette détection laplacienne ressemble à ceci:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Sobel :

Le détecteur de bord Sobel est une méthode basée sur le gradient basée sur les dérivés du premier ordre. Il calcule les premières dérivées de l'image séparément pour les axes X et Y. Dans ce cas, l'opérateur utilise deux noyaux 5x5 qui sont convolus avec l'image d'origine pour calculer les approximations des dérivées - un pour les changements horizontaux et un pour les changements verticaux.

- Canny :

Canny Edge Detection est un algorithme de détection de bord populaire. C'est un algorithme en plusieurs étapes.

1. Réduction du bruit

Étant donné que la détection des contours est sensible au bruit dans l'image, la première étape consiste à supprimer le bruit dans l'image avec un filtre gaussien 5x5.

2. Recherche du dégradé d'intensité de l'image

L'image lissée est ensuite filtrée avec un noyau Sobel à la fois dans la direction horizontale et verticale pour obtenir la première dérivée dans la direction horizontale ( $G_x$ ) et la direction verticale ( $G_y$ ).

3. Suppression non maximale

Après avoir obtenu la magnitude et la direction du gradient, une analyse complète de l'image est effectuée pour supprimer tous les pixels indésirables qui peuvent ne pas constituer le bord. Pour cela, à chaque pixel, le pixel est vérifié s'il s'agit d'un maximum local à son voisinage dans le sens du gradient. (le résultat que nous obtiendrons est une image binaire avec des «bords fins».)

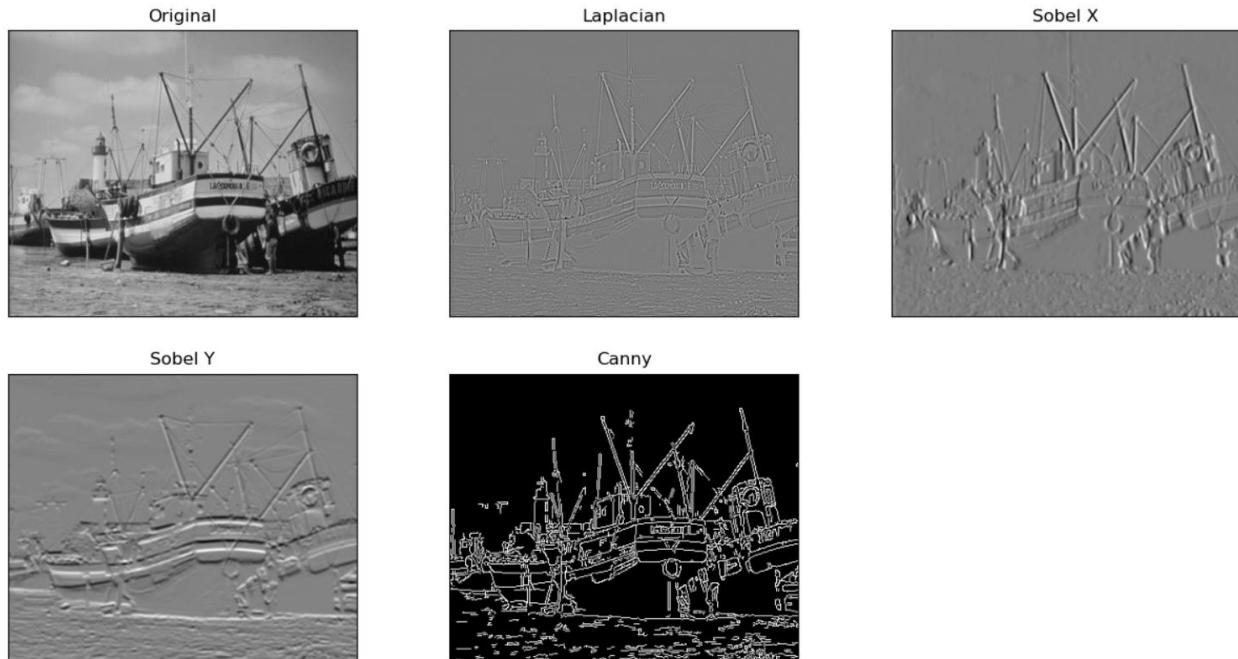
4. Seuil d'hystérésis

Cette étape décide quelles sont toutes les arêtes qui sont réellement des arêtes et lesquelles ne le sont pas. Pour cela, nous avons besoin de deux valeurs de seuil,  $\text{minVal}$  et  $\text{maxVal}$ . Dans ce cas, nous définissons  $\text{minVal} = 100$  et  $\text{maxVal} = 200$ . Toutes les arêtes avec un gradient d'intensité supérieur à  $\text{maxVal}$  sont sûres d'être des arêtes et celles inférieures à  $\text{minVal}$  sont sûrement des arêtes non-arêtes, donc rejetées. Ceux qui se situent entre ces deux seuils sont classés bords ou non-bords en fonction de leur connectivité. S'ils sont connectés à des pixels à «bord sûr», ils sont considérés comme faisant partie des bords. Sinon, ils sont également rejetés.

OpenCV met tout ce qui précède dans une seule fonction, `cv2.Canny()`. Le premier argument est notre image d'entrée. Les deuxième et troisième arguments sont respectivement nos  $\text{minVal}$  et  $\text{maxVal}$ . Le troisième argument est `aperture_size`. C'est la taille du noyau Sobel utilisé pour

trouver des dégradés d'image. Par défaut, il est 3. Le dernier argument est L2gradient qui spécifie l'équation pour trouver la magnitude du gradient. Par défaut, il est faux.

**Résultat :**



Q5: Selon vous, quels sont les points faibles de toutes ces méthodes de filtrage par convolution ?

Si le noyau de convolution n'est pas centré-symétrique, les opérations de convolution et de filtrage obtiendrez des résultats complètement différents. Les opérations de convolution peuvent faire en sorte que l'image devienne plus petite (arêtes d'image de perte).

## Partie 3 : Images binaires et opérations entre images

Q1: Complétez le code fourni afin d'appliquer différents algorithmes de seuillage / binarisation (seuillage fixe ou adaptatif) à différentes images fournies. Insérez certains résultats dans votre rapport et commentez-les afin de démontrer que vous avez compris le mécanisme de binarisation

[https://docs.opencv.org/3.4.2/d2/d96/tutorial\\_py\\_table\\_of\\_contents\\_imgproc.html](https://docs.opencv.org/3.4.2/d2/d96/tutorial_py_table_of_contents_imgproc.html)

1. les traits de seuillage fix:

Test\_image : suzan.jpg

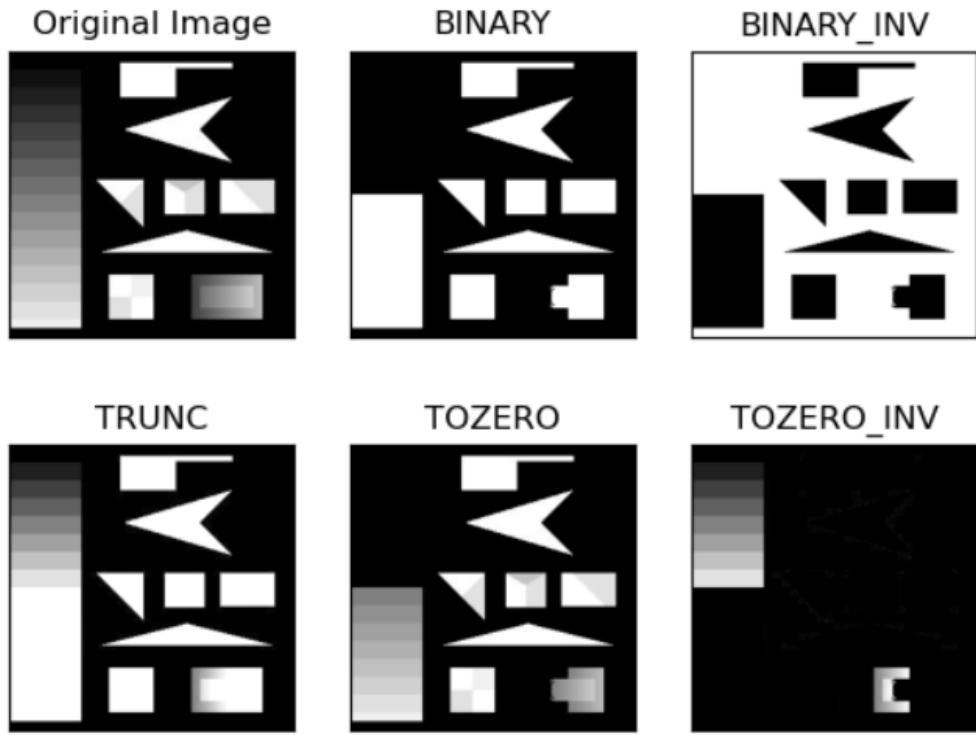
Code :

On utilise la fonction *threshold ()* dans opencv pour seuiller l'image.

Si la valeur du pixel est supérieure à une valeur de seuil, une valeur lui est attribuée (peut être blanche), sinon une autre valeur lui est attribuée (peut être noire). La fonction utilisée est `cv.threshold`. Le premier argument est l'image source, qui doit être une image en niveaux de gris. Le deuxième argument est la valeur de seuil utilisée pour classer les valeurs de pixel. Dans ce cas, nous définissons la valeur de seuil sur 127. Le troisième argument est la valeur `maxVal` qui représente la valeur à donner si la valeur du pixel est supérieure (parfois inférieure à) la valeur de seuil.

```
def trait_threshold():
    img = cv.imread('../data/suzan.jpg', 0)
    ret, thresh1 = cv.threshold(img, 127, 255, cv.THRESH_BINARY)
    ret, thresh2 = cv.threshold(img, 127, 255, cv.THRESH_BINARY_INV)
    ret, thresh3 = cv.threshold(img, 127, 255, cv.THRESH_TRUNC)
    ret, thresh4 = cv.threshold(img, 127, 255, cv.THRESH_TOZERO)
    ret, thresh5 = cv.threshold(img, 127, 255, cv.THRESH_TOZERO_INV)
    titles = ['Original Image', 'BINARY', 'BINARY_INV', 'TRUNC', 'TOZERO', 'TOZERO_INV']
    images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
    for i in range(6):
        plt.subplot(2, 3, i+1), plt.imshow(images[i], 'gray')
        plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
    plt.show()
```

Résultat :



## 2. les traits de seuillage adaptatif:

Test\_image : fields.jpg

Code :

On utilise la fonction *adaptiveThreshold ()* dans opencv pour seuiller l'image.

Dans le cas du seuillage adaptatif, l'algorithme calcule le seuil pour une petite région de l'image. Nous obtenons donc des seuils différents pour différentes régions de la même image et cela nous donne de meilleurs résultats pour les images avec un éclairage variable.

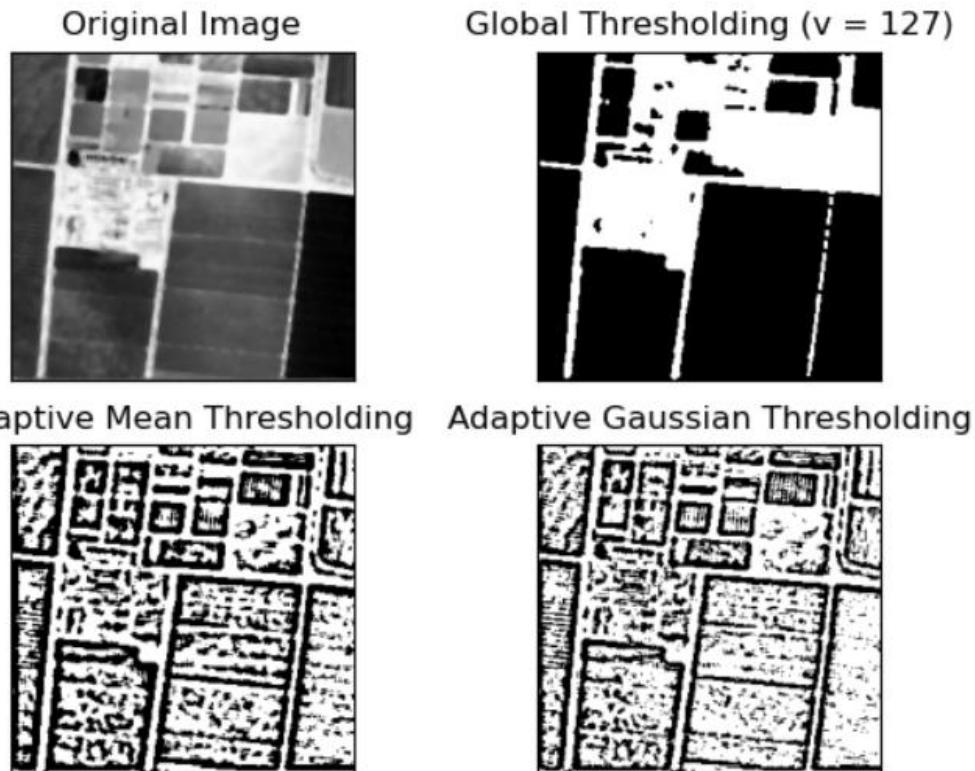
- Méthode adaptative - Elle décide de la manière dont la valeur de seuil est calculée.
  - cv.ADAPTIVE\_THRESH\_MEAN\_C*: la valeur de seuil est la moyenne de la zone de voisinage.
  - cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C*: la valeur de seuil est la somme pondérée des valeurs de voisinage où les poids sont une fenêtre gaussienne.
- Taille du bloc - Il décide de la taille du quartier.
- C - Il s'agit simplement d'une constante qui est soustraite de la moyenne ou moyenne pondérée calculée.

```

def trait_adaptive_threshold():
    img = cv.imread('../data/fields.jpg', 0)
    img = cv.medianBlur(img, 5)
    ret, th1 = cv.threshold(img, 127, 255, cv.THRESH_BINARY)
    th2 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 2)
    th3 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 11, 2)
    titles = ['Original Image', 'Global Thresholding (v = 127)',
              'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
    images = [img, th1, th2, th3]
    for i in range(4):
        plt.subplot(2, 2, i + 1), plt.imshow(images[i], 'gray')
        plt.title(titles[i])
        plt.xticks([]), plt.yticks([])
    plt.show()

```

Resultat :

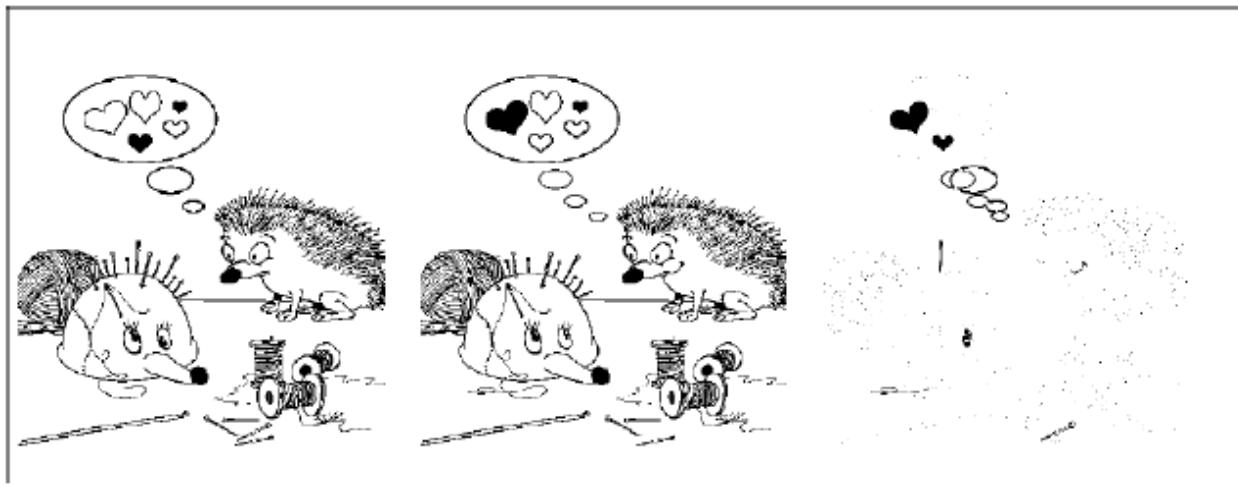


On compare le traitement à seuillage fixe et le traitement à seuillage adaptatif de la même image, on peut voir que les caractéristiques d'image du traitement de seuillage adaptatif avec éclairage variable sont plus claire.

Q2: Ouvrez les images jeu1 et jeu2.

A l'aide d'opérations arithmétiques entre ces deux images, mettez (comme sous l'image ci-dessous) en évidence les 7 différences existant entre ces deux images.

Proposez un algorithme capable de résoudre ce problème lorsque les images à comparer sont susceptibles d'être décalées de quelques pixels l'une par rapport à l'autre (CF jeu3).



L'algorithme : XOR ( $img1 = img1 \oplus img2$ ) ou Difference ( $img1 = |img1 - img2|$ )

Code :

On utilise la fonction `subtract()` dans opencv pour comparer les 2 images.

On réinitialise la taille de l'image pour que les 3 images aient la même taille, puis les comparons plus tard.

```

processing.py x P3_Q2_comparer.py x P3_Q4_counter.py x

import cv2
from matplotlib import pyplot as plt

img_1 = cv2.imread('../data/jeu1.jpg', 0)
img_2 = cv2.imread('../data/jeu2.jpg', 0)
img_3 = cv2.imread('../data/jeu3.jpg', 0)

sp_1 = img_1.shape
crop_size = (sp_1[1], sp_1[0])
img_2 = cv2.resize(img_2, crop_size, interpolation=cv2.INTER_CUBIC)
img_3 = cv2.resize(img_3, crop_size, interpolation=cv2.INTER_CUBIC)

img_subtract_1 = cv2.subtract(img_1, img_2)
img_subtract_2 = cv2.bitwise_xor(img_1, img_3)
img_subtract_3 = cv2.bitwise_xor(img_2, img_3)

ret_1, img_thresh_1 = cv2.threshold(img_subtract_1, 127, 255, cv2.THRESH_BINARY)
ret_2, img_thresh_2 = cv2.threshold(img_subtract_2, 127, 255, cv2.THRESH_BINARY)
ret_3, img_thresh_3 = cv2.threshold(img_subtract_3, 127, 255, cv2.THRESH_BINARY)

titles = ['jeu1 (380,442)', 'jeu2 (380,442)', 'jeu3 (383,442)', 'jeu1-jeu2 (Différence d'image)', 'jeu1-jeu3 (Décalage d'image)', 'jeu2-jeu3 (Décalage d'image)']
images = [img_1, img_2, img_3, img_thresh_1, img_thresh_2, img_thresh_3]

for i in range(6):
    plt.subplot(2, 3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()

```

Les images à comparer et les résultats:

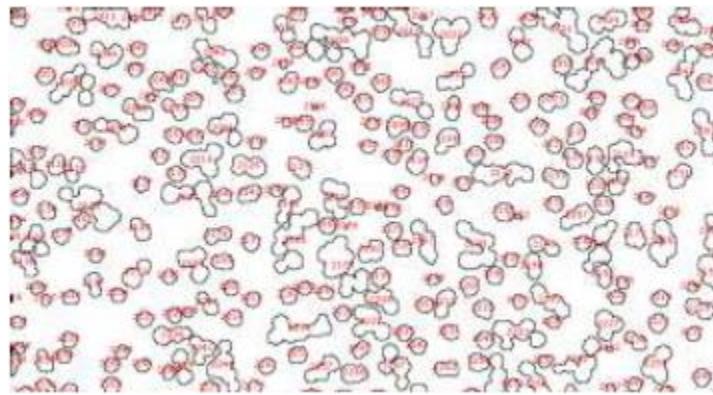


Q3: Complétez le code Video\_processing fourni afin d'appliquer une séquence d'opérations (soustraction entre 2 frames successives, binarisation, érosion/dilatation, ...) avant affichage du résultat. Expliquez l'intérêt de cette séquence (opérations entre frames successives) ?

Voir la partie Video\_processing de Partie 0.

Q4: Des biologistes aimeraient compter le nombre de cellules présentes dans des images. Proposez un algorithme permettant de compter les cellules automatiquement. Vérifiez la présence de 329 cellules dans l'image cellule.png.

[https://docs.opencv.org/3.4/d3/dc0/group\\_\\_imgproc\\_\\_shape.html#ga107a78bf7cd25dec05fb4dfc5c9e765f](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga107a78bf7cd25dec05fb4dfc5c9e765f)



Code :

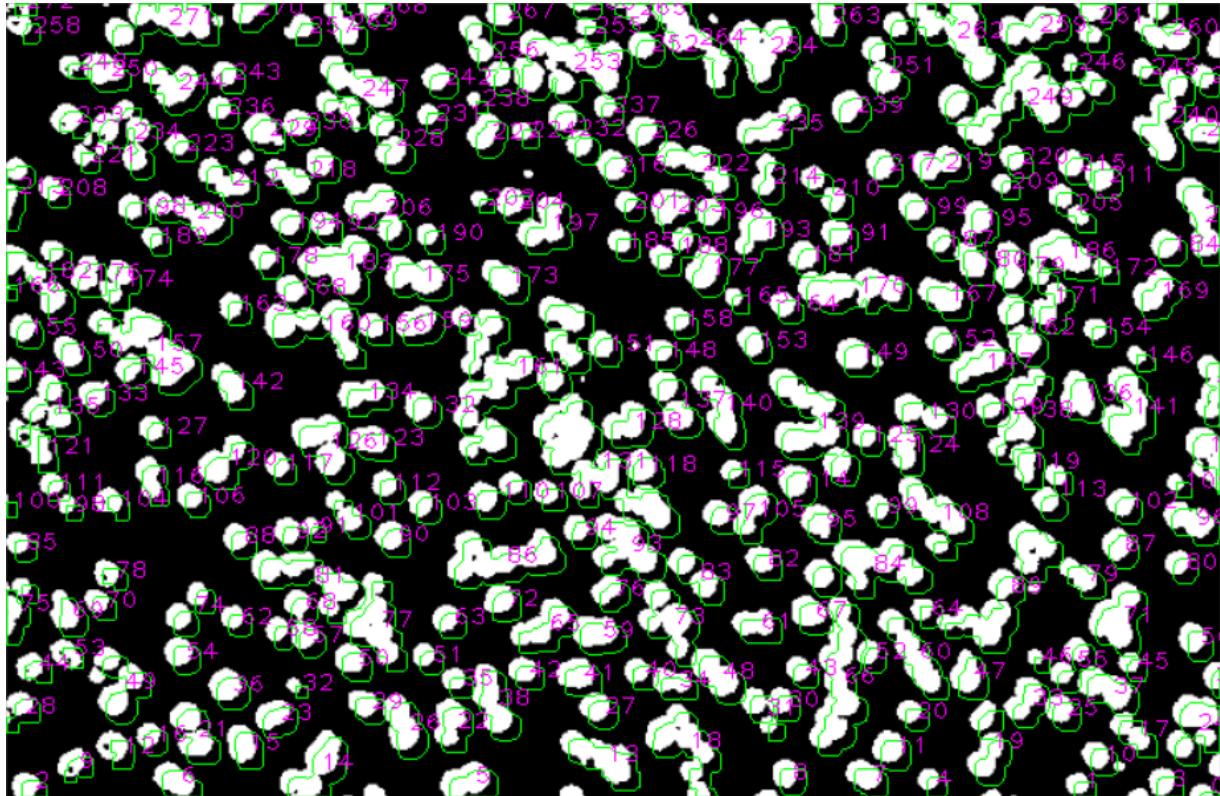
On utilise la fonction *findContours ()* dans opencv pour obtenir le nombre de cellule.

```

img = cv2.imread('../data/cellules.png', 1)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Transformez l'image en image en niveaux de gris
kernel=np.ones((2,2),np.uint8) #Opération d'expansion de corrosion
erosion=cv2.erode(gray,kernel,iterations=5) #expansion
dilation=cv2.dilate(erosion,kernel,iterations=5) #corrosion
ret, thresh = cv2.threshold(dilation, 150, 255, cv2.THRESH_BINARY) # Méthode binaire de traitement de seuil
thresh1 = cv2.GaussianBlur(thresh,(3,3),0) # Filtrage gaussien
contours,hierarchy=cv2.findContours(thresh1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) # Trouver un domaine connecté
#Comparez la zone des domaines connectés
area=[] #Créez un tableau vide et mettez la zone du domaine connecté
contours1=[] #Créez un tableau vide et mettez le nombre moins la plus petite zone
for i in contours:
    area.append(cv2.contourArea(i))
    #print(area)
    if cv2.contourArea(i)>30: # Calculer la zone Supprimer les domaines connectés avec une petite zone
        contours1.append(i)
print(len(contours1)-1) #Calculez le nombre de domaines connectés
draw=cv2.drawContours(img,contours1,-1,(0,255,0),1) #Delimiter les domaines connectés
#Trouvez le centre de gravité du domaine connecté et dessinez le nombre au point de coordonnées du centre de gravité
for i, j in zip(contours1,range(len(contours1))):
    M = cv2.moments(i)
    cX=int(M["m10"]/M["m00"])
    cY=int(M["m01"]/M["m00"])
    draw1=cv2.putText(draw, str(j), (cX, cY), 1, 1, (255, 0, 255), 1) #Dessinez des nombres sur le point de coordonnées central
cv2.imshow("draw", draw1)
cv2.imshow("thresh1", thresh1)
cv2.waitKey()
cv2.destroyAllWindows()

```

Résultat : 293



Q5 : Malheureusement, une vérification faite manuellement par un biologiste indique que l'algorithme s'est trompé et que le nombre réel de cellules avoisine plutôt les 370.

Une rapide analyse de l'image vous permet de trouver l'origine de cette erreur. En effet, une mauvaise utilisation du système d'acquisition a généré deux types de bruits artificiels :

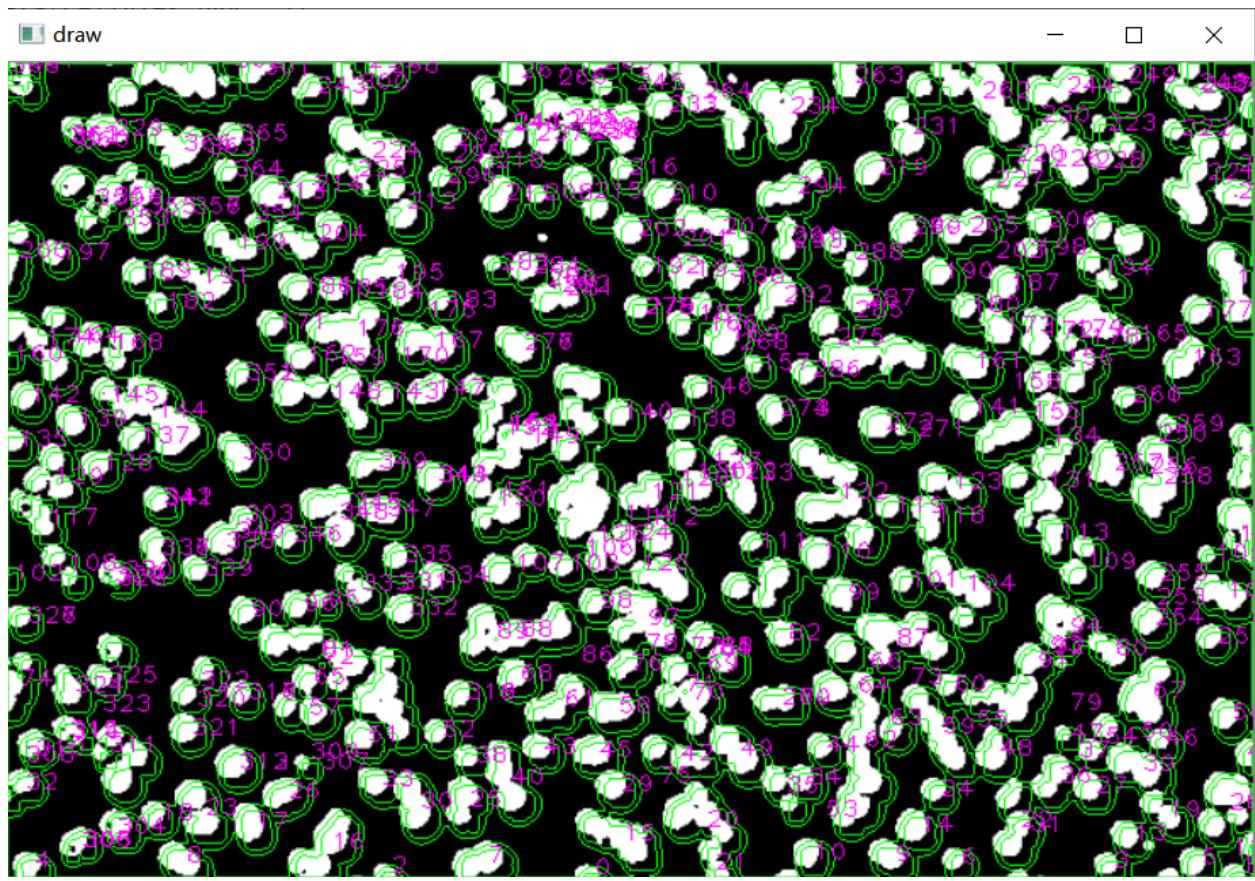
1. Des petites taches noires sont apparues un peu partout et sont interprétées comme étant des cellules (cf cas 1 de l'image ci-dessous)
2. Des cellules sont collées sur la photo alors qu'elles sont dissociées dans la réalité (cf cas 2 de l'image ci-dessous)

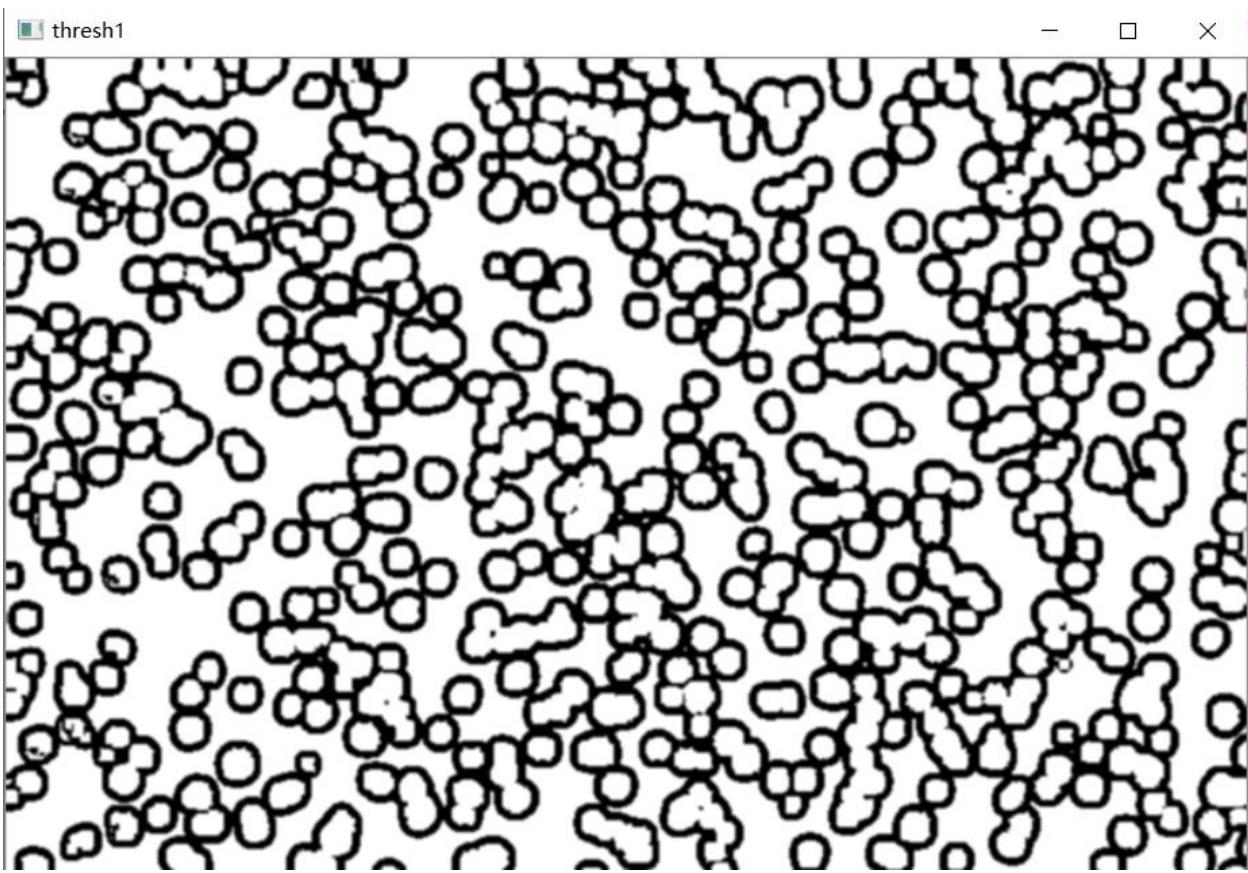


Trouvez la suite d'opérations morphologiques à réaliser afin que ces tâches disparaissent et que le moins de cellules possibles restent collées les unes aux autres. Un protocole "correct" devrait vous ramener vers une détection automatique de 370 cellules.

On transforme le seuillage fixe en un seuillage adaptatif.

```
# ret, thresh = cv2.threshold(dilation, 150, 255, cv2.THRESH_BINARY) # Méthode binaire de traitement de seuil
thresh = cv2.adaptiveThreshold(dilation, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
```





r:\Python\pyt

368

Resultat : 368

## Partie 4 : Image Tagging

A vous de jouer, durant les séances de TP restantes, en créant un programme associant des mot-clés décrivant le contenu (couleur, contenu, qualité ...) des images et montrant vos compétences en traitement d'images. Ce programme sera décrit de manière détaillé dans votre rapport.

*Base d'apprentissage disponible sur Celene. Ce programme devra générer des fichiers « .txt » au format similaire à ceux de la base d'apprentissage.*

*Une image = un fichier .txt portant le même nom que l'image ; seul l'extension change ; le fichier txt est créé dans le même dossier que le fichier image.*

### Liste des mot-clés à associer aux images :

( Note : Apportez les modifications correspondantes en fonction des mot-clés aux images du dossier /GT )

- **Color** : yellow, green, red, blue, brown, gray, white, black, orange
- **Scene** : Indoor, outdoor, countryside, city  
(Format d'affichage : inside, outside, nature, city)
- **Content inside** : human, face, animal, car, plane, van, byke  
(Objets supplémentaires : ski)
- **Content outside** : sky, sun, house, tree, sea, water  
(Objets supplémentaires : mountain, snow)
- **Quality** : fuzzy, sharp, dark, light
- **Complexity** : isolated, multiple

Fichier : /TP\_2020/partie4/IngTagging

### 1. Environment setup

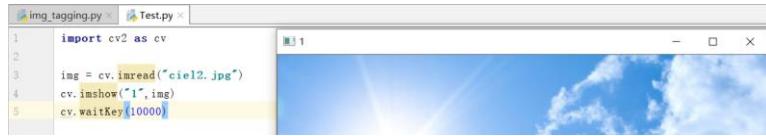
#### 1) Configuration du système :

- Système: *Windows 10*
- Pilote graphique: *NVIDIA GeForce 930MX , Inter® HD Graphics 620*
- Processeur: *Inter® Core™ i5-7200U CPU @ 2.50GHz 2.71 GHz*

#### 2) OpenCV ( *OpenCV-Python – cv2* )

*OpenCV* est une bibliothèque open source pour le traitement d'images, l'analyse et la vision industrielle.

- Dans la console win10, installez *OpenCV*: `>pip install opencv-python`.
- Testez si l'environnement d'*OpenCV-Python* est configuré avec succès.



L'image est correctement affichée, l'environnement est donc correctement configuré.

### 3) Tensorflow

Ma version du tensorflow : tensorflow 2.1.0

- Dans la console win10, installez tensorflow: `>pip install tensorflow`.
- Testez si l'installation est correctement configurée.

```
F:\Python\python\l\Scripts>python -c "import tensorflow as tf; print(tf.__version__)"
2020-03-09 08:11:34.967203: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library cudart64_101.dll; dlerror: cudart64_101.dll not found
2020-03-09 08:11:34.977122: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2.1.0
```

*TensorFlow* est une plate-forme de bout en bout qui facilite la création et le déploiement de modèles *ML*. Un système entier peut aider à résoudre des problèmes difficiles et réels avec l'apprentissage automatique.

### 4) Python [ *Deep learning research* ]

Ma version du python : Python 3.7.1

*(v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)] on win32*

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)] on win32
```

Develop and evaluate *deep learning models* in *Python*.

The platform for getting started in applied deep learning is *Python*.

### 5) Autres packages nécessaires

***collections*** : Ce module implémente des types de données de conteneurs spécialisés offrant des alternatives aux conteneurs, dict, liste, ensemble et tuple intégrés à usage général de Python.

***NumPy*** : NumPy ajoute la prise en charge de grands tableaux et matrices multidimensionnels, ainsi qu'une grande collection de fonctions mathématiques de haut niveau pour fonctionner sur ces tableaux.

***torch*** : Il fournit un tableau N-dimensionnel ou Tensor flexible, qui prend en charge les routines de base pour l'indexation, le découpage, la transposition, la conversion de type, le redimensionnement, le partage de stockage et le clonage.

***torchvision.transforms*** : *Transforms* sont des transformations d'image courantes

**os** : Le module OS de Python permet d'utiliser les fonctionnalités dépendant du système d'exploitation. Les fonctions fournies par le module OS vous permettent d'interfacer avec le système d'exploitation sous-jacent sur lequel Python s'exécute - que ce soit Windows, Mac ou Linux.

**PIL** : La bibliothèque d'imagerie Python (PIL) ajoute des capacités de traitement d'image à votre interpréteur Python. Cette bibliothèque prend en charge de nombreux formats de fichiers et offre de puissantes capacités de traitement d'images et de graphiques.

**sys** : Paramètres et fonctions spécifiques au système. Ce module permet d'accéder à certaines variables utilisées ou maintenues par l'interpréteur et à des fonctions qui interagissent fortement avec l'interpréteur.

**matplotlib.pyplot** : Matplotlib est une bibliothèque complète pour créer des visualisations statiques, animées et interactives en Python.

**tkinter** : Le package tkinter est une fine couche orientée objet au-dessus de Tcl / Tk. tkinter est un ensemble de wrappers qui implémentent les widgets Tk en tant que classes Python.

## 2. Algorithmes

### 2.1 Reconnaissance des couleurs principales

#### Méthode:

- Convertissez le mode de couleur de l'image de RVB à HSV
- Utilisez la fonction cv2.inRange() pour le filtrage des couleurs d'arrière-plan
- Binariser les couleurs filtrées
- Utilisez cv2.dilate() pour faire la dilatation et l'érosion
- Zone blanche des statistiques

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'Project'. The main editor window shows the 'Colors.py' file with the following code:

```
# Treat image + get color
def get_color(file_path):
    frame = cv2.imread(file_path)
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    color_list = []
    sum_ctns = 0
    color_eff = []
    sum_list = []
    num_color_all = 0
    color_dict = CreateColorList()
    for d in color_dict:
        num_color_all = num_color_all + 1
        mask = cv2.inRange(hsv, color_dict[d][0], color_dict[d][1])
        binary = cv2.threshold(mask, 127, 255, cv2.THRESH_BINARY)[1]
        binary = cv2.dilate(binary, None, iterations=2)
        cnts, hiera = cv2.findContours(binary.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        sum = 0
        for c in cnts:
            sum += cv2.contourArea(c)
        sum_ctns = sum + sum_ctns
        sum_list.append([d, sum])

        if sum > 0:
            color_eff.append([d, sum])

    # Get the second element of the list
    def takeSecond(elem):
        return elem[1]

    color_eff.sort(key=takeSecond, reverse=True)
    color_eff = color_eff[:2]
    return color_eff
```

**Explication:**

Il existe actuellement de nombreux types d'espaces colorimétriques dans le domaine de la vision par ordinateur. HSL et HSV sont les deux modèles de couleurs les plus courants représentés par des coordonnées cylindriques. Il remappe le modèle RVB afin qu'il puisse être plus intuitif visuellement que le modèle RVB. Par conséquent, le traitement efficace de l'image dans l'espace colorimétrique est généralement effectué dans l'espace HSV, puis une plage stricte doit être donnée pour le composant HSV correspondant dans la couleur de base. Voici la plage de flou calculée par l'expérience:

	black	gray	white	red		orange	yellow	green	cyan	blue	brown
hmin	0	0	0	0	156	11	26	35	78	100	0
hmax	180	180	180	10	180	25	34	77	99	124	10
smin	0	0	0	43		43	43	43	43	43	43
smax	255	43	30	255		255	255	255	255	255	255
vmin	0	46	221	46		46	46	46	46	46	46
vmax	46	220	255	255		255	255	255	255	255	255

Note: H: 0 - 180, S: 0 - 255, V: 0 - 255

## 2.2 Reconnaissance de scène (type d'environnement et scène spéciale)

**Méthode:**

- Appelez le modèle pré-entraîné PlacesCNN formé sur le jeu de données Places365.
- L'étiquette des attributs de scène est générée à partir du jeu de données SUNattribute.
- La carte thermique est générée à l'aide de la technique CAM.

```

115
116     def load_model():
117         # this model has a last conv feature map as 14x14
118
119         model_file = folder_path + '/wideresnet18_places365.pth.tar'
120         file_wideresnet = folder_path + '/wideresnet.py'
121
122         from New_Tags.resource import wideresnet
123         model = wideresnet.resnet18(num_classes=365)
124         checkpoint = torch.load(model_file, map_location=lambda storage, loc: storage)
125         state_dict = {str.replace(k, 'module.', ''): v for k, v in checkpoint['state_dict'].items()}
126         model.load_state_dict(state_dict)
127         model.eval()
128
129         # hook the feature extractor
130         features_names = ['layer4', 'avgpool'] # this is the last conv layer of the resnet
131         for name in features_names:
132             model._modules.get(name).register_forward_hook(hook_feature)
133
134         return model
135
136     def get_folder_path():
137         return folder_path
138
139
140     def test_S_ME(image_path):
141
142         global result_S_ME, features_blobs, model
143         result_S_ME = []

```

## Explication:

- Dataset and Model:**

Dans le domaine de la vision par ordinateur, la détection d'objets et la reconnaissance de scènes sont deux parties très importantes, et si nous nous attendons à ce que les algorithmes d'apprentissage automatique atteignent des performances de classification sémantique quasi humaine, nous avons besoin de plus en plus de données valides pour l'aider à s'améliorer. Nous recherchons donc de grands ensembles de données.

La base de données Places365 est un référentiel de 10 millions de photographies de scènes qui est étiqueté avec des catégories sémantiques de scènes, comprenant une liste vaste et diversifiée des types d'environnements rencontrés dans le monde.

Et PlacesCNN est un modèle pré-entraîné formé sur l'ensemble de données Places365, qui utilise les réseaux de neurones à convolution (CNN) de pointe, et peut générer une carte thermique à l'aide de la technique CAM, qui est une visualisation des CNN formés sur Places365 .



Fig. 1. Image samples from various categories of the Places Database (two samples per category). The dataset contains three macro-classes: Indoor, Nature, and Urban.

(pic. viennent du fichier pdf du site Web [Places](#))

## 2.3 Reconnaissance d'objet principal au contenu

**Méthode:** (Différentes étapes viennent du [learn\\_OpenCV](#))

- Téléchargez les modèles et essayez d'appeler le modèle pré-entraîné formé sur l'ensemble de données MSCOCO en utilisant Mask-RCNN et Tensorflow.
- Initialisez les paramètres.  
(L'algorithme Mask-RCNN produit les sorties de détection prédites en tant que boîtes englobantes. Nous définirons un score de confiance au début et chaque boîte englobante lui sera associée. Seules les boîtes au-dessus du paramètre de seuillage de confiance seront utilisées pour un traitement ultérieur.)
- "Chargez le modèle et les classes."  
Le fichier mscoco\_labels.names contient tous les objets (classes) que le mode peut traiter. Le fichier colors.txt contient toutes les couleurs que nous avons définies pour masquer les objets au début.  
Chargez le réseau:
  - frozen\_inference\_graph.pb*: Utilisez le fichier graphique gelé pour obtenir les poids de modèle pré-entraînés.
  - mask\_rcnn\_inception\_v2\_coco\_2018\_01\_28.pbtxt*: Le fichier de graphique de texte qui a été réglé par le groupe de support DNN d'OpenCV, afin que le réseau puisse être chargé à l'aide d'OpenCV.
- Lisez l'entrée de l'image en cours de traitement.  
(Dans cette étape, nous lisons l'image, le flux vidéo ou la webcam. De plus, nous ouvrons également le graveur vidéo pour enregistrer les images avec des cadres de délimitation de sortie détectés.)
- Traitez chaque image  
(L'image d'entrée d'un réseau neuronal doit être dans un certain format appelé blob.)

Nous lisons donc une image à partir de l'image d'entrée ou du flux vidéo (un flux vidéo est composé de nombreuses images) et la convertissons en un objet blob d'entrée pour le réseau neuronal à l'aide de la fonction `blobFromImage`. Le flux vidéo peut être un fichier vidéo ou un moniteur de caméra mis à jour en temps réel.

L'objet blob est ensuite transmis au réseau en tant qu'entrée et pour obtenir une liste des cadres de délimitation prédits et des masques d'objets à partir des couches de sortie.)

f) Post-traitement de la sortie du réseau → Supprimez les cases dont le score de confiance est inférieur au seuillage donné.

g) Dessinez les cases prévues

(Enfin, nous dessinons les boîtes et définissons leurs étiquettes d'objets détectés et leurs scores de confiance sur l'image d'entrée. Nous dessinons également les masques colorés dans les boîtes.)

```

151 def test_ME(image_path, folder_path):
152
153     global result_ME
154     result_ME = []
155
156     # initialize
157     initial(folder_path)
158
159     # Open the image file
160     if not os.path.isfile(image_path):
161         print("Input image file ", image_path, " doesn't exist")
162         sys.exit(1)
163
164     img = cv.imread(image_path)
165
166     # Create a 4D blob from a img.
167     blob = cv.dnn.blobFromImage(img, swapRB=True, crop=False)
168
169     # Set the input to the network
170     net.setInput(blob)
171
172     # Run the forward pass to get output from the output layers
173     boxes, masks = net.forward(['detection_out_final', 'detection_masks'])
174
175     # Extract the bounding box and mask for each of the detected objects
176     postprocess(boxes, masks, img)

```

### Explication:

- **Segmentation d'instance :**

Dans la segmentation d'instance, le but est de détecter des objets spécifiques dans une image et de créer un masque autour de l'objet d'intérêt.

La segmentation d'instance peut également être considérée comme une détection d'objet où la sortie est un masque au lieu d'une simple boîte englobante.

- **Proposition régionale basée :**

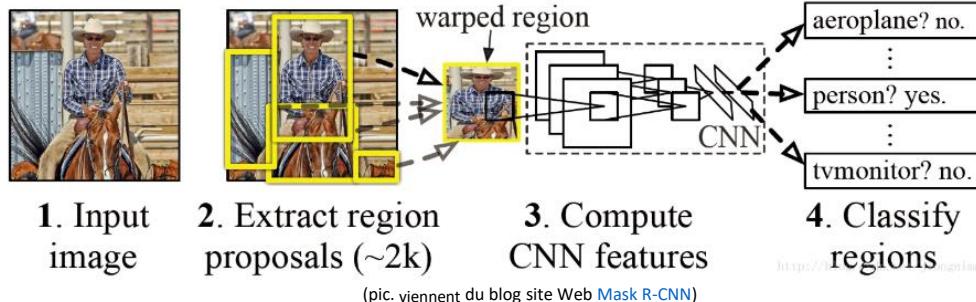
Sélection de la zone de détection de la cible: détectez l'espace colorimétrique et la matrice de similarité, puis détectez la zone à détecter en fonction de ceux-ci, puis classifiez et prédissez en fonction du résultat de la détection.

- Comment fonctionne Mask-RCNN?

Séquence de développement:

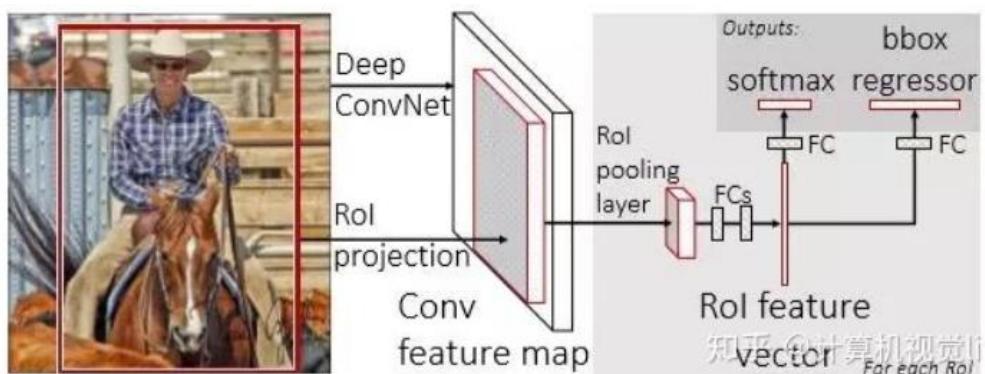
1. *R-CNN* (Longue durée, faible efficacité) --> Détection de cible uniquement

### R-CNN: Regions with CNN features

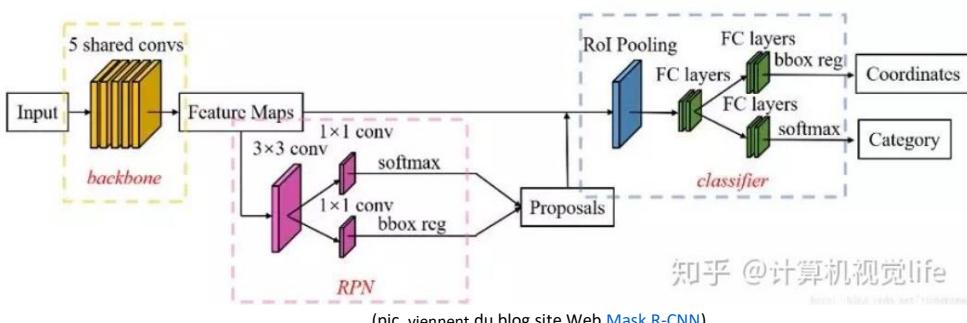


R-CNN génère des propositions de région basées sur une recherche sélective, puis traite chaque région proposée, puis à un moment donné pour une image, utilise des réseaux de convolution pour produire une objet label et sa bounding boîte.

2. *Fast R-CNN* [2015] --> Détection de cible uniquement



3. *Faster R-CNN* [2016] --> Détection de cible uniquement



**Process:**

- ① Utilisez d'abord RPN pour extraire la région de la boîte candidate
- ② Ensuite, utilisez directement un réseau de neurones pour effectuer une extraction de caractéristiques sur toute l'image;

- ③ Utilisez ensuite un ROI Pooling Layer pour obtenir les caractéristiques correspondant à chaque ROI sur la carte des caractéristiques de l'image complète,
- ④ Utilisez ensuite FC pour classer et corriger la boîte englobante.

4. *Mask R-CNN* [2017] --> détection de cible + classification de cible + segmentation au niveau des pixels

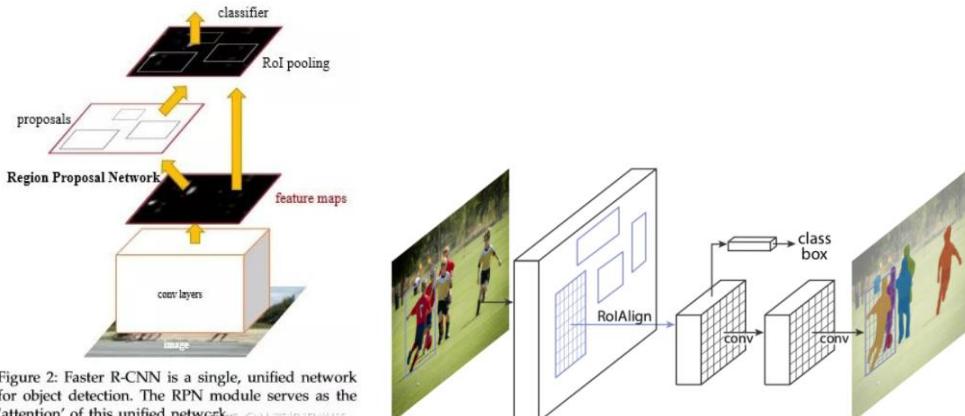


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.  
[译者注: @计算机视觉life]

(pic. viennent du blog site Web [Mask R-CNN](#))

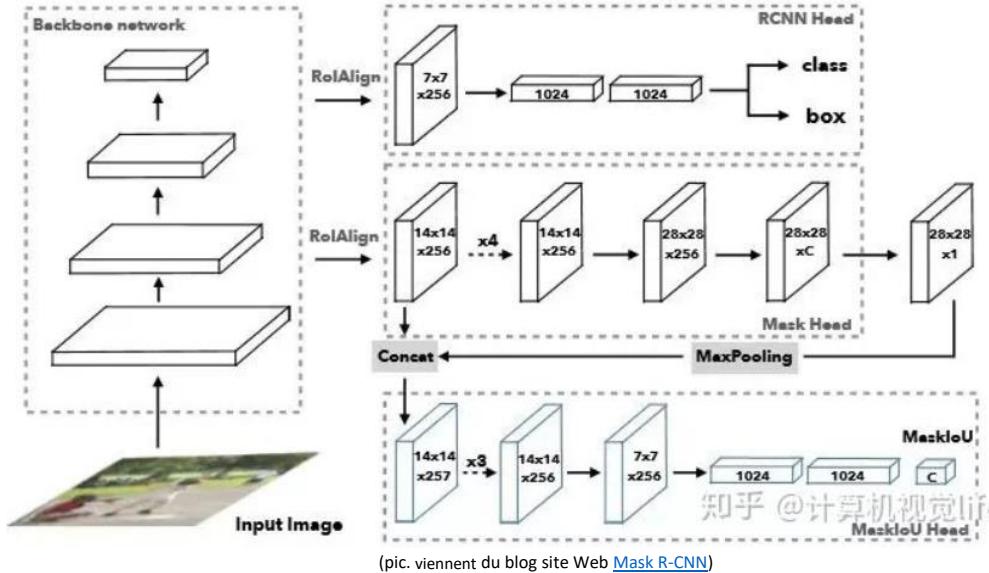
#### Améliorer:

- ① Introduisez la branche de prédiction de masque (Mask-Head) pour prédire le masque de segmentation de manière pixel à pixel.
- ② ROI Pooling est remplacé par ROI Align, et la quantification grossière de ROI Pooling est supprimée, de sorte que les caractéristiques extraites soient bien alignées avec l'entrée;

Dans la segmentation d'instance sur le framework Mask R-CNN, trois choses principales doivent être effectuées:

- 1) Détection de cible: les boîtes englobantes sont dessinées directement sur l'image résultante.
- 2) Classification des cibles: pour chaque cible, trouver la classe / étiquette correspondante pour distinguer s'il s'agit d'une personne, d'une voiture ou d'autres catégories.
- 3) Segmentation de la cible au niveau du pixel: pour chaque cible, pour distinguer au niveau du pixel, quel est le premier plan et quel est l'arrière-plan.

5. *Mask Scoring R-CNN* [2019] --> détection de cible + classification de cible + segmentation au niveau des pixels



(pic. viennent du blog site Web [Mask R-CNN](#))

## 2.4 Reconnaissance de la netteté de l'image

### Méthode:

- Effectuer une opération de convolution sur un canal de l'image (généralement en utilisant la valeur de gris) à travers un masque de Laplace [déttection de bord]
- Puis calculez l'écart-type / la variance,
- Si l'image a une variance plus élevée, alors elle a une plage de réponse en fréquence plus large, représentant une image normale et précisément focalisée, l'image est très claire.
- Mais si l'image a une petite variance, alors elle a une plage de réponse en fréquence étroite, ce qui signifie qu'il y a peu de bords dans l'image et que l'image est très floue.
- Définissez le seuillage approprié

```

8     # image sharpness recognition
9     def getImageVar(imgPath):
10
11         global sharpness
12         sharpness = []
13         image = cv2.imread(imgPath)
14
15         # Convert to grayscale image
16         img2gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
17
18         # Laplacian operator - edge detection
19         imageVar = cv2.Laplacian(img2gray, cv2.CV_64F).var() # var(): Calculate variance
20
21     return imageVar
22
23
24     # image sharpness recognition
25     def getSharpness(imageVar):
26         if imageVar <= 50:
27             sharpness.append("fuzzy")
28         elif imageVar >= 1000:
29             sharpness.append("sharp")
30
31     return sharpness

```

### Explication:

Le flou de l'image est causé par de nombreux facteurs tels que la défocalisation, le tremblement de l'appareil photo, le mouvement, etc.

La méthode la plus directe pour quantifier la qualité des images floues consiste à calculer la transformée de Fourier rapide de l'image, puis à visualiser la distribution des hautes et basses fréquences. Si l'image comporte une petite quantité de composants haute fréquence, l'image peut être considérée comme floue. Cependant, il est très difficile de distinguer la valeur de seuil spécifique de la quantité de haute fréquence, et une valeur de seuil inappropriée conduira à des résultats extrêmement médiocres.

## 2.5 Reconnaissance de la luminosité de l'image

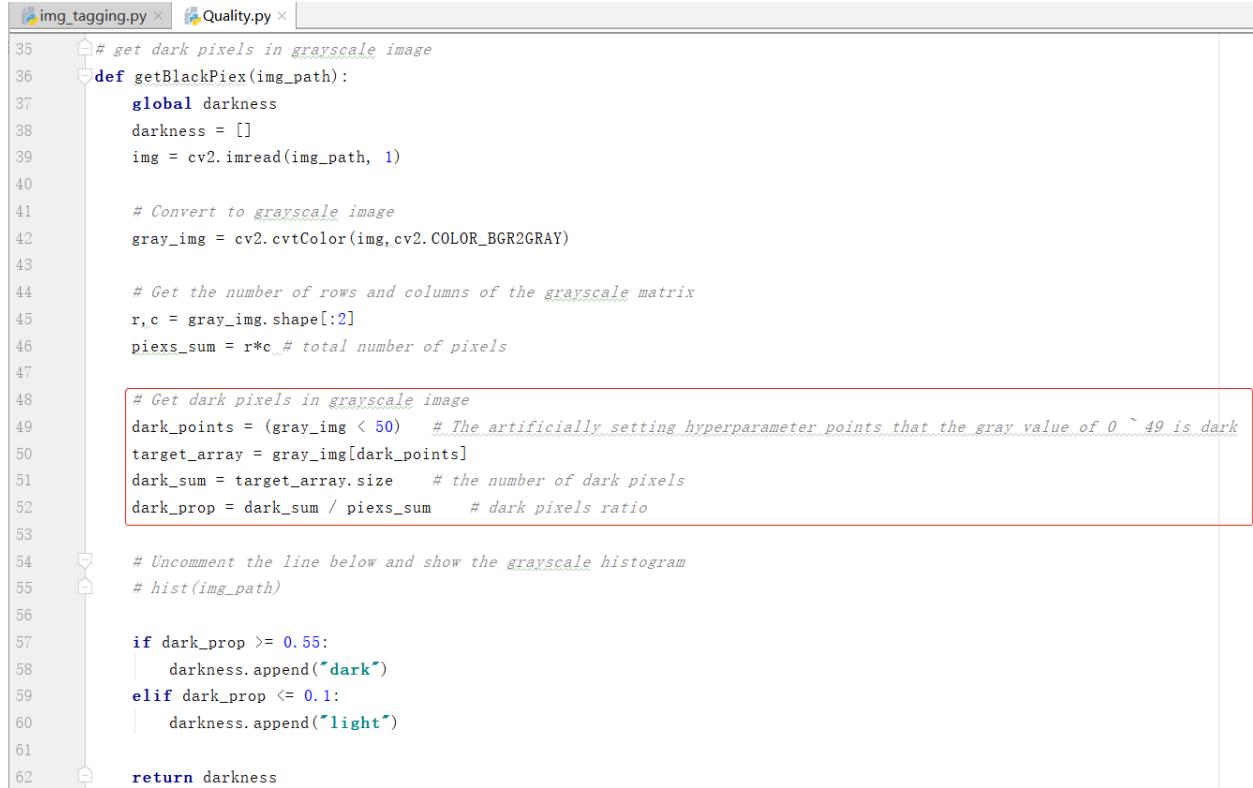
### Méthode:

- Obtenir l'histogramme en niveaux de gris de l'image → Juger en fonction de la distribution des

### valeurs de niveaux de gris

- Valeur de gris de l'image sombre: concentrée à l'avant, par exemple entre 0 et 30
- Valeur de gris de l'image lumineuse: concentrée à l'arrière
- Besoin de compter le nombre de pixels sombres, puis de diviser par le nombre total de pixels de l'image pour obtenir le pourcentage p
- Définir le seuillage de p

```



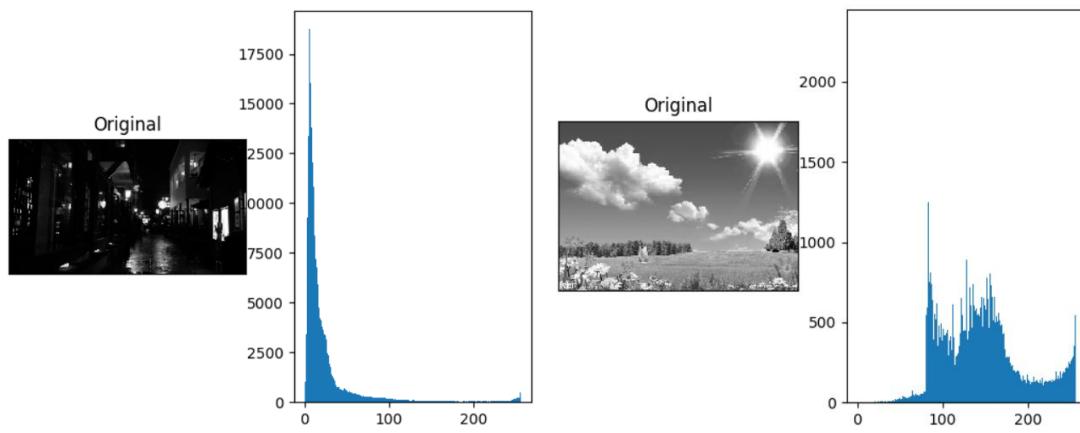
```

35     # get dark pixels in grayscale image
36     def getBlackPiex(img_path):
37         global darkness
38         darkness = []
39         img = cv2.imread(img_path, 1)
40
41         # Convert to grayscale image
42         gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
43
44         # Get the number of rows and columns of the grayscale matrix
45         r,c = gray_img.shape[:2]
46         piexs_sum = r*c # total number of pixels
47
48         # Get dark pixels in grayscale image
49         dark_points = (gray_img < 50) # The artificially setting hyperparameter points that the gray value of 0 ~ 49 is dark
50         target_array = gray_img[dark_points]
51         dark_sum = target_array.size # the number of dark pixels
52         dark_prop = dark_sum / piexs_sum # dark pixels ratio
53
54         # Uncomment the line below and show the grayscale histogram
55         # hist(img_path)
56
57         if dark_prop >= 0.55:
58             darkness.append("dark")
59         elif dark_prop <= 0.1:
60             darkness.append("light")
61
62     return darkness

```


```

### Explication:



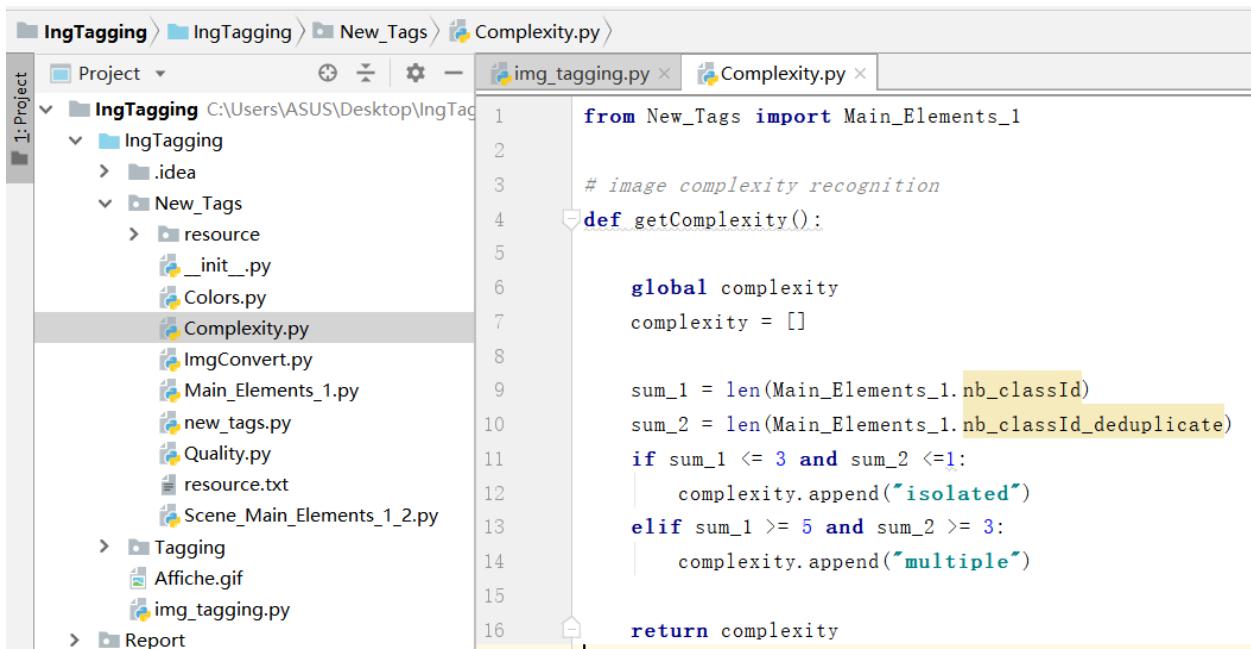
(pic. viennent du blog site Web [Brightness](#))

## 2.6 Reconnaissance de la complexité de l'image

### Méthode:

Comptez les objets identifiés dans le contenu et définissez un mécanisme de jugement:

- Si "Nombre total d'objets de détection initiale" <= 2 et "Nombre total après suppression des objets en double" <= 1, il est marqué comme "isolated",
- Si "Nombre total d'objets de détection initiale">> = 5 et "Nombre total après suppression des objets en double">> = 3, il sera marqué comme "multiple".



The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under '1:Project'. The 'IngTagging' project contains several sub-directories and files. In the 'New\_Tags' directory, the 'Complexity.py' file is selected and shown in the main editor window. The code in 'Complexity.py' is as follows:

```
from New_Tags import Main_Elements_1

# image complexity recognition
def getComplexity():

    global complexity
    complexity = []

    sum_1 = len(Main_Elements_1.nb_classId)
    sum_2 = len(Main_Elements_1.nb_classId_deduplicate)

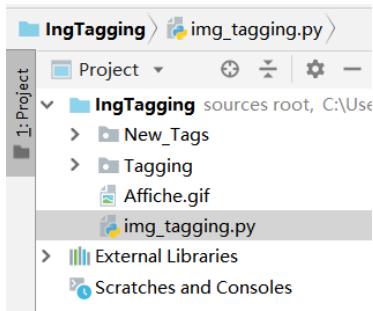
    if sum_1 <= 3 and sum_2 <= 1:
        complexity.append("isolated")
    elif sum_1 >= 5 and sum_2 >= 3:
        complexity.append("multiple")

    return complexity
```

## 3. Run et Résultat

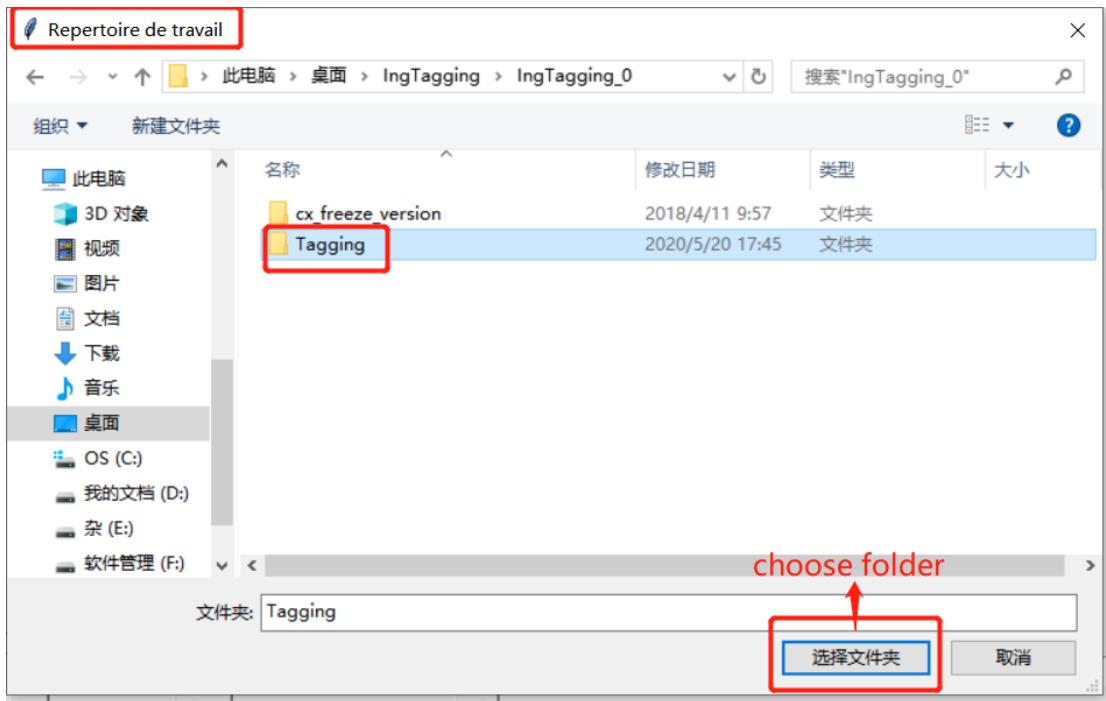
### Étape\_1: run

Ouvrez le projet *IngTagging* avec Python et exécutez “ *img\_tagging.py* ”.

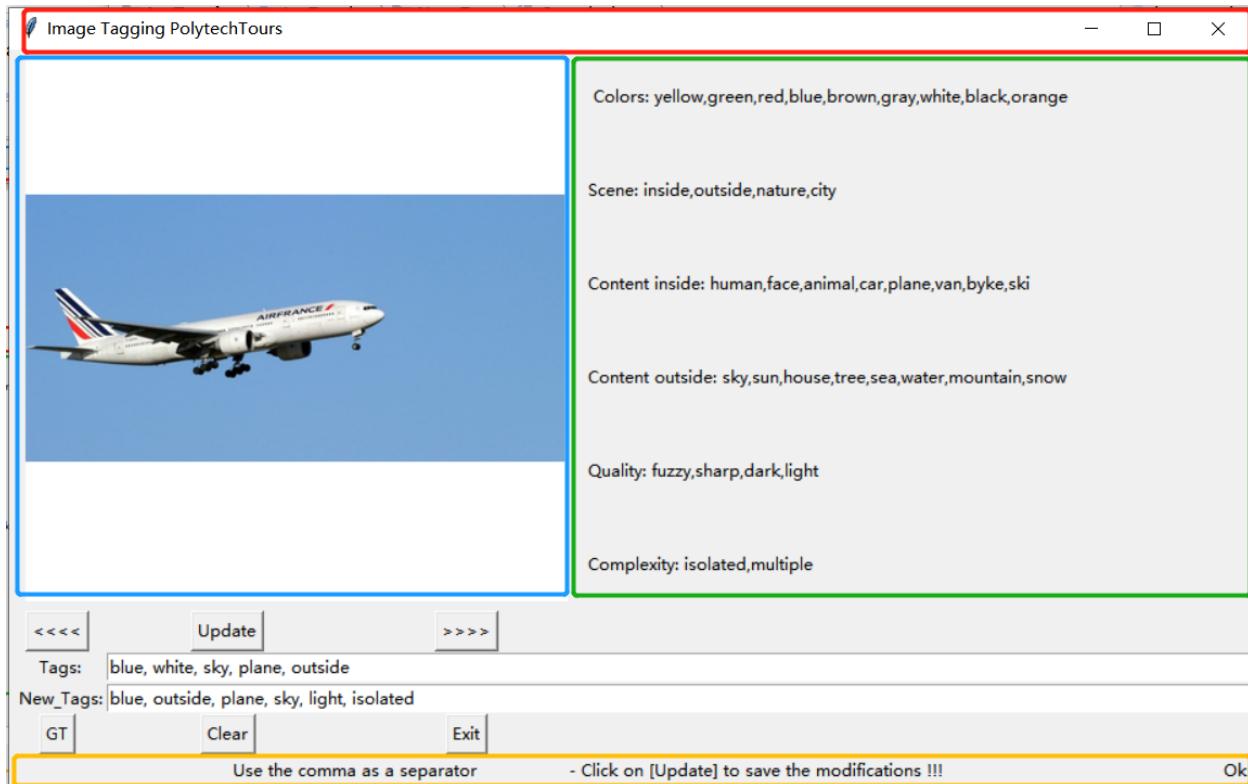


## Étape\_2: choisissez un dossier d'images à tester

Une fenêtre de sélection de dossier apparaîtra, puis sélectionnez un dossier d'images à tester (ici: le dossier “/Tagging” dans ce cas → Définir le répertoire de travail).



## Étape\_3: Affichage de l'interface du logiciel



### Description de la mise en page:

- **Cadre rouge:** titre du système logiciel.
- **Cadre bleu:** zone d'affichage des images sous le répertoire de travail des images.
- **Cadre vert:** les attributs d'image censés être détectés. Et il y a 5 parties:
  - Color : yellow, green, red, blue, brown, gray, white, black, orange
  - Scene : Indoor, outdoor, countryside, city  
(Format d'affichage : inside, outside, nature, city)
  - Content inside [1 - détection des objets principaux] : human, face, animal, car, plane, van, byke (Objets supplémentaires : ski)
  - Content outside [2 - prédition de la scène principale]: sky, sun, house, tree, sea, water (Objets supplémentaires : mountain, snow)
  - Quality: fuzzy , sharp, dark , light
  - Complexity : isolated, multiple
- **Cadre jaune:** Barre de note de nom d'opération / d'état / d'image.

### Description de la fonction du bouton:

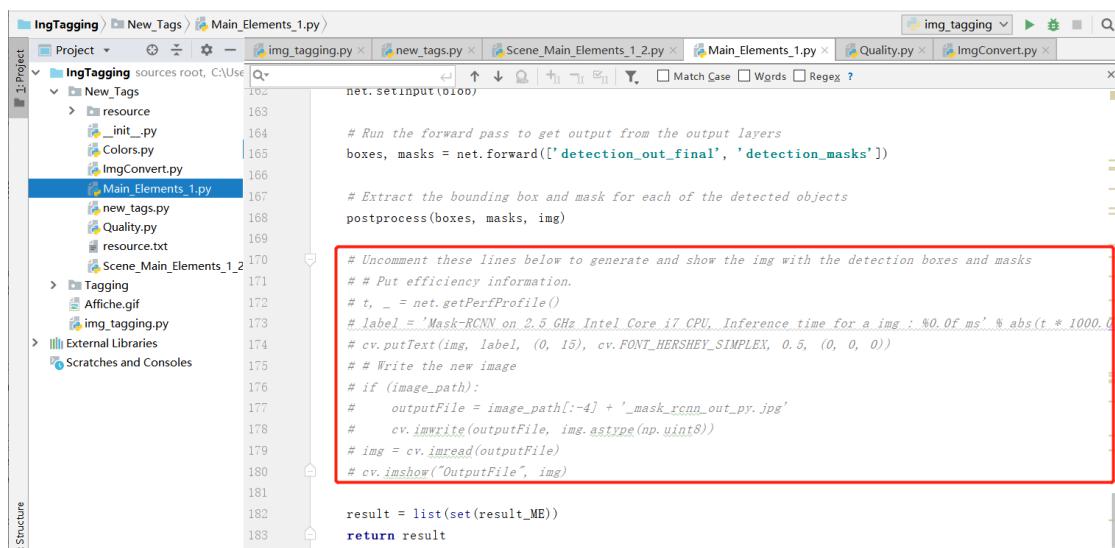
- **Bouton [ <<< ]:** Passez à l'image précédente.
- **Bouton [ Update ]:** Modifiez le contenu de la barre [Balises] et cliquez sur [Mettre à jour] pour enregistrer les modifications.
- **Bouton [ >>> ]:** Passez à l'image suivante.
- **Barre [ Tags ]:** Lisez le contenu du fichier txt avec le même nom que l'image dans le sous-dossier “/GT” et affichez-le dans la barre.

- **Barre [ New\_Tags ]:** Obtenez les étiquettes d'attributs d'image détectées par les algorithmes d'étiquetage automatique des images et écrivez les nouvelles étiquettes dans le fichier txt avec le même nom de l'image dans le sous-dossier “ /NT ”.
- **Bouton [ GT ]:** Comparez le contenu du fichier txt avec le même nom de l'image dans le sous-dossier “ /NT ” avec le contenu du fichier txt avec le même nom de l'image dans le sous-dossier “ /GT ”, et écrivez le résultat de la comparaison dans le Fichier **LOG.TXT** sous le répertoire de travail de l'image courante.
- **Bouton [ Clear ]:** Effacez le contenu de la barre [Tags].
- **Bouton [ Exit ]:** Quittez le système d'exploitation du logiciel.

## Étape\_4: Effets supplémentaires - Masques / rendus

### Main\_Elements [ 1 ] -La nouvelle image avec les cases de détection et les masques:

Fichier [ [/New\\_Tags/Main\\_Elements\\_1.py](#) ]: Décommentez la partie du cadre rouge pour générer et afficher la nouvelle image avec les cases de détection et les masques.

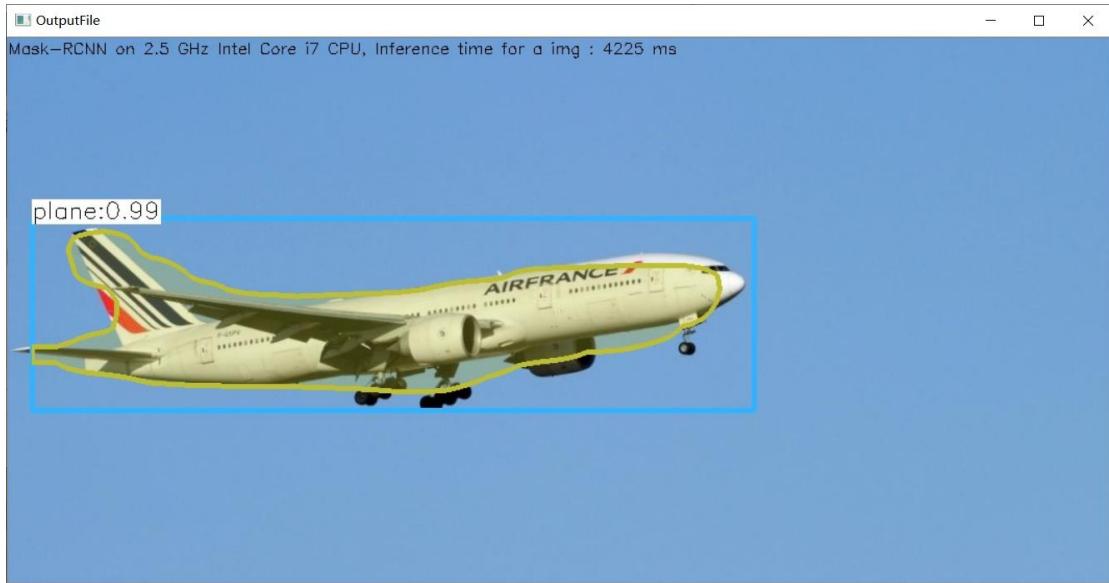


```

IngTagging > New_Tags > Main_Elements_1.py
Project  img_tagging.py new_tags.py Scene_Main_Elements_1_2.py Main_Elements_1.py Quality.py ImgConvert.py
1:Project IngTagging sources root, C:\Use
  New_Tags
    resource
      __init__.py
      Colors.py
      ImgConvert.py
    Main_Elements_1.py
      new_tags.py
      Quality.py
      resource.txt
    Scene_Main_Elements_1_2.py
      Tagging
        Affiche.gif
        img_tagging.py
      External Libraries
      Scratches and Consoles
  163 net.setInput(dI00)
  164
  165     # Run the forward pass to get output from the output layers
  166     boxes, masks = net.forward(['detection_out_final', 'detection_masks'])
  167
  168     # Extract the bounding box and mask for each of the detected objects
  169     postprocess(boxes, masks, img)
  170
  171     # Uncomment these lines below to generate and show the img with the detection boxes and masks
  172     # ## Put efficiency information.
  173     # t, _ = net.getPerfProfile()
  174     # _label = 'Mask-RCNN on 2.5 GHz Intel Core i7 CPU. Inference time for a img : %0.0f ms' % abs(t * 1000.0)
  175     # cv.putText(img, _label, (0, 15), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
  176
  177     # ## Write the new image
  178     # if (image_path):
  179     #     outputFile = image_path[:-4] + '_mask_rcnn_out_py.jpg'
  180     #     cv.imwrite(outputFile, img.astype(np.uint8))
  181     #     img = cv.imread(outputFile)
  182
  183     result = list(set(result_ME))
  return result

```

Ex [ [airfrance\\_4.jpg](#) ]:



## Main\_Elements [ 2 ] - Heatmap:

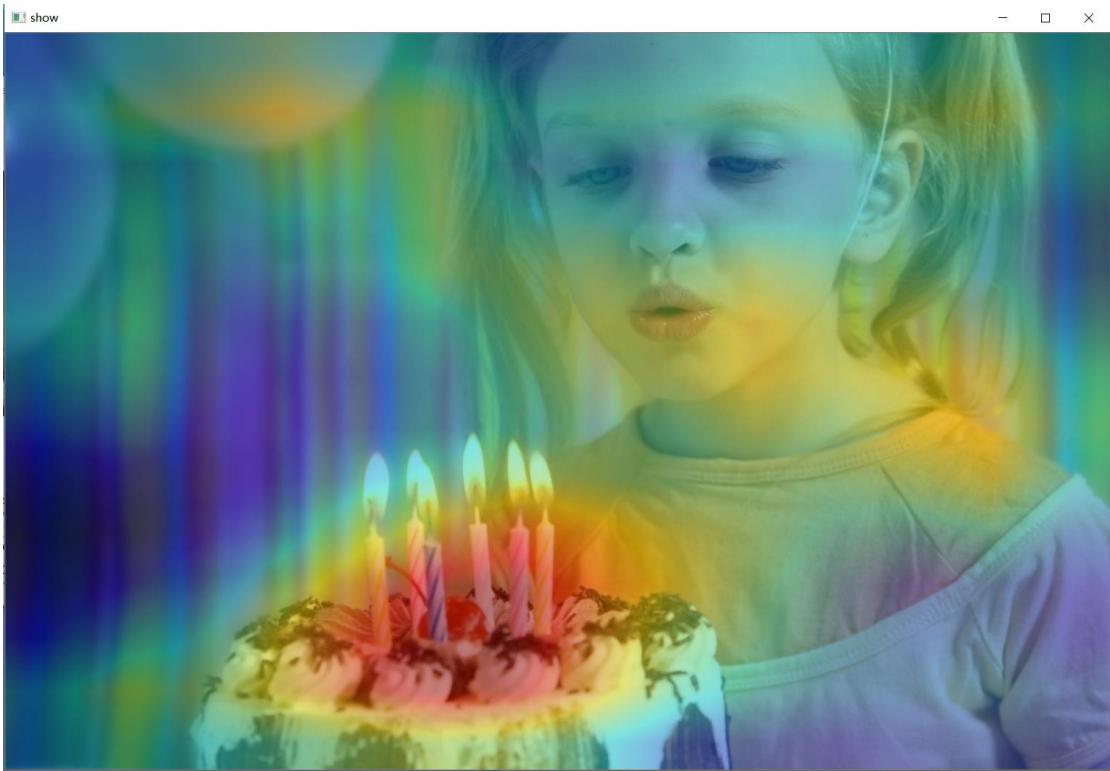
Fichier [ [/New\\_Tags/Scene\\_Main\\_Elements\\_1\\_2.py](#) ]: Décommentez la partie du cadre rouge pour générer et afficher l'image avec la région informative pour prédire la catégorie.

```



```

Ex [ [bougie.jpg](#) ]:



## Étape\_5: Affichage du fichier LOG.TXT

Évaluez l'efficacité de la détection des logiciels par score.

```
Log.TXT - 记事本  
文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)  
Log file :  
blue, outside, plane, sky, light, isolated >> blue, white, sky, plane, outside - Score:[1, 2]  
orange, inside, face, fuzzy, light >> yellow, orange, face, inside - Score:[1, 2]  
gray, green, inside, animal, isolated >> gray, animal - Score:[0, 3]  
black, orange, gray, nature, outside, human, sky, sea, isolated >> black, orange, gray, sun,  
outside, nature, sky - Score:[1, 3]  
orange, white, brown, outside, face, light, isolated >> white, face - Score:[0, 5]  
blue, white, nature, outside, sky, fuzzy, light, isolated >> blue, white, sky, sun, outside - Score:[1,  
4]  
white, gray, inside, human, light, multiple >> white, inside, human - Score:[0, 3]  
gray, black, inside, face >> gray, inside, human, face - Score:[1, 1]  
green, nature, outside, animal, light >> blue, green, gray, outside, animal, sky - Score:[3, 2]  
gray, outside, car >> red, blue, gray, outside, car, sky - Score:[3, 0]  
green, gray, outside, human, light >> green, white, outside, human - Score:[1, 2]  
blue, white, nature, outside, human, ski, snow, sky, light >> blue, white, red, ski, human, outside,  
sky - Score:[1, 3]  
brown, white, inside, face, sharp >> white, black, face - Score:[1, 3]
```

.....

```
>> blue, outside, sea, sky, city, dark - Score:[6, 0]
>> outside, city, dark - Score:[3, 0]
>> - Score:[0, 0]
>> red, outside, car, gray - Score:[4, 0]
>> green, outside, car, gray - Score:[4, 0]
Final score : [242, 33] on 66 images
```

### Explication:

- $s1 >> s2$  – Score:[ set1, set2]  
set1 =  $s1 - s2$  # mots-clés manquants  
set2 =  $s2 - s1$  # mots-clés supplémentaires
- Final score: [SET1, SET2]  
SET1: la somme de tous les set1  
SET2: la somme de tous les set2