

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

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](#_Toc58243615)

[Partie 1 : Type d’images et statistiques images 4](#_Toc58243616)

[Partie 2 : Filtrage, convolution et détection de contours 7](#_Toc58243617)

[Partie 3 : Images binaires et opérations entre images 9](#_Toc58243618)

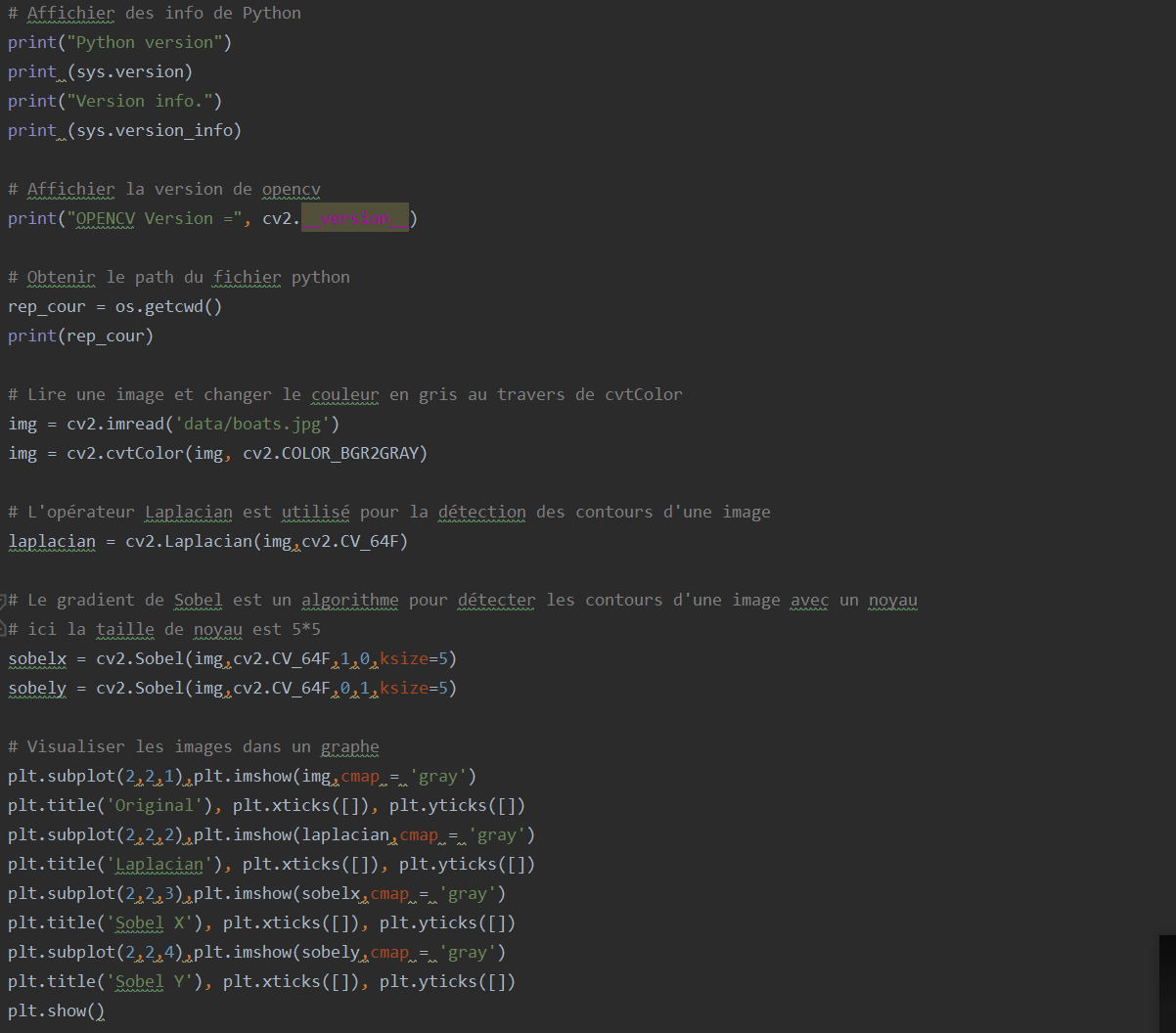
[Partie 4 : Image Tagging 12](#_Toc58243619)

[Partie 5 : Vidéos, Détection de changement de plan et résumé automatique de vidéo 13](#_Toc58243620)

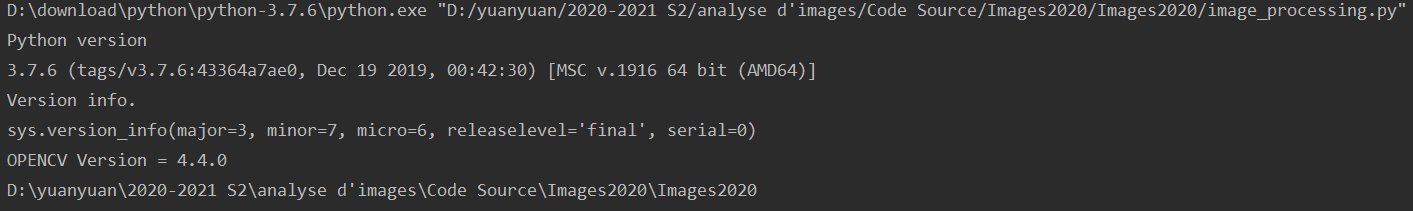
# Partie 0 : Prise en main de Python + OpenCV

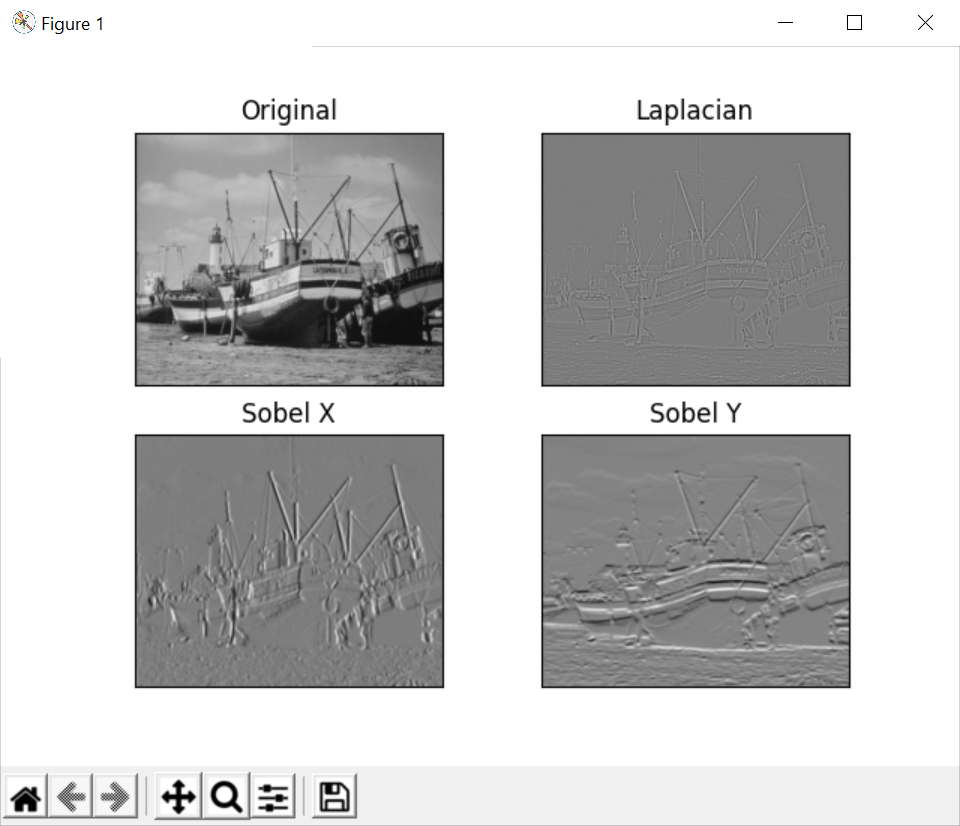
1. Image\_processing.py

Code Source avec commentaires :

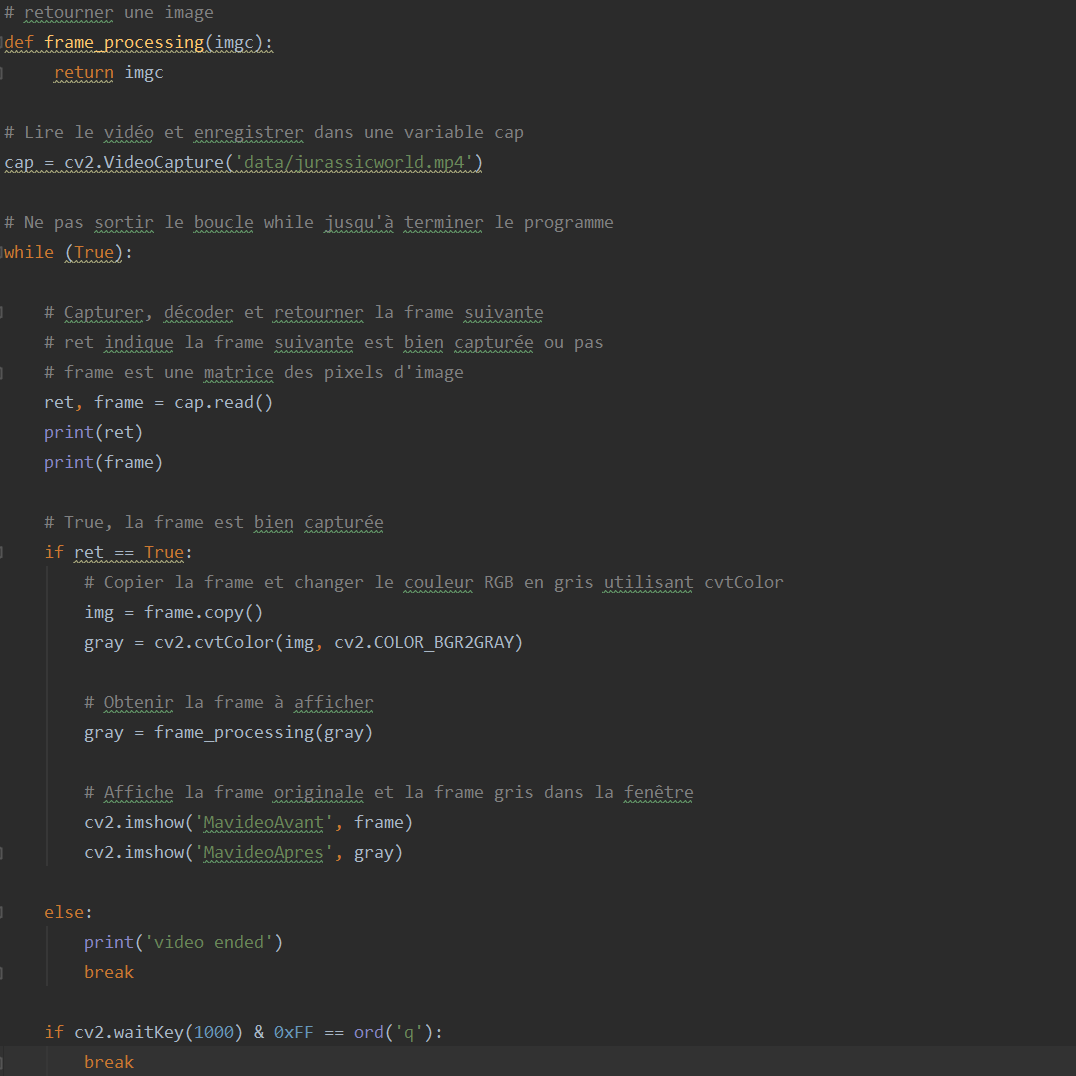


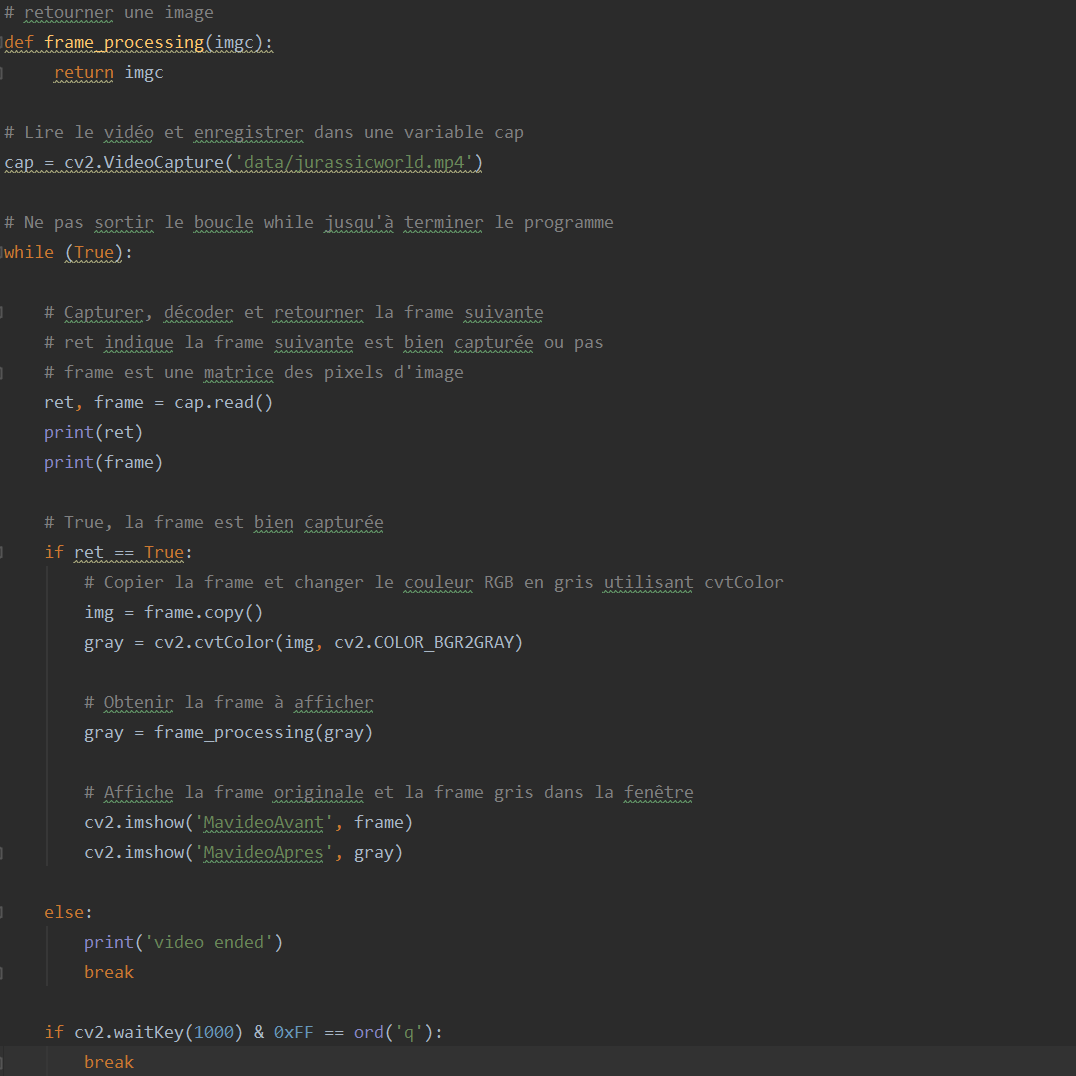
Résultat :



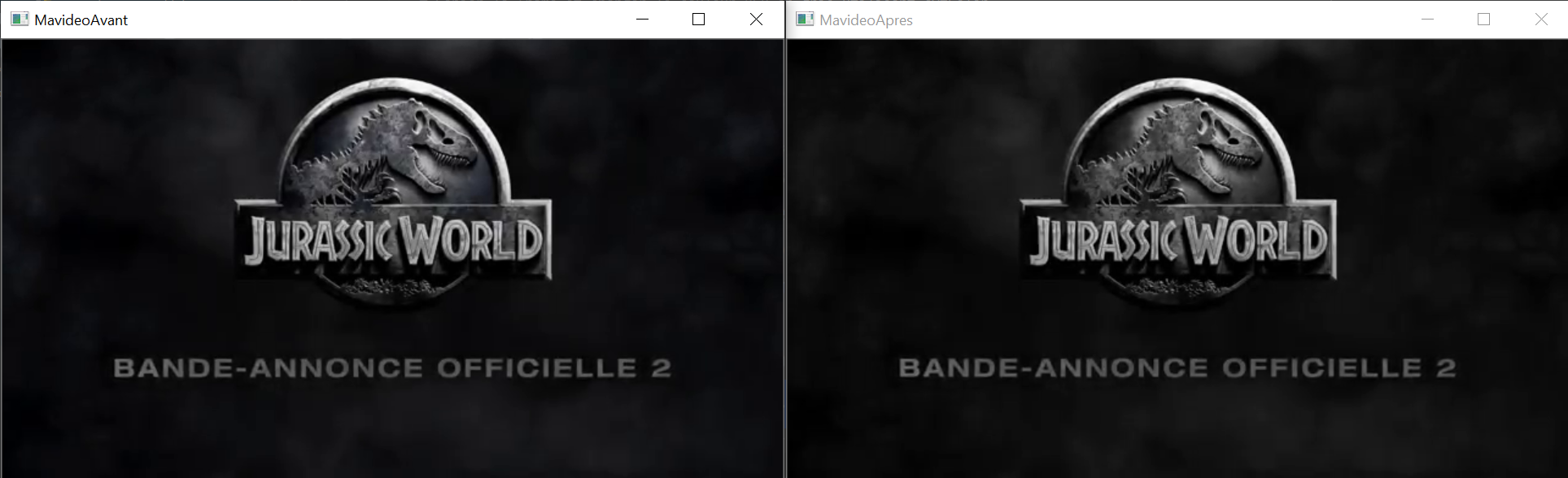


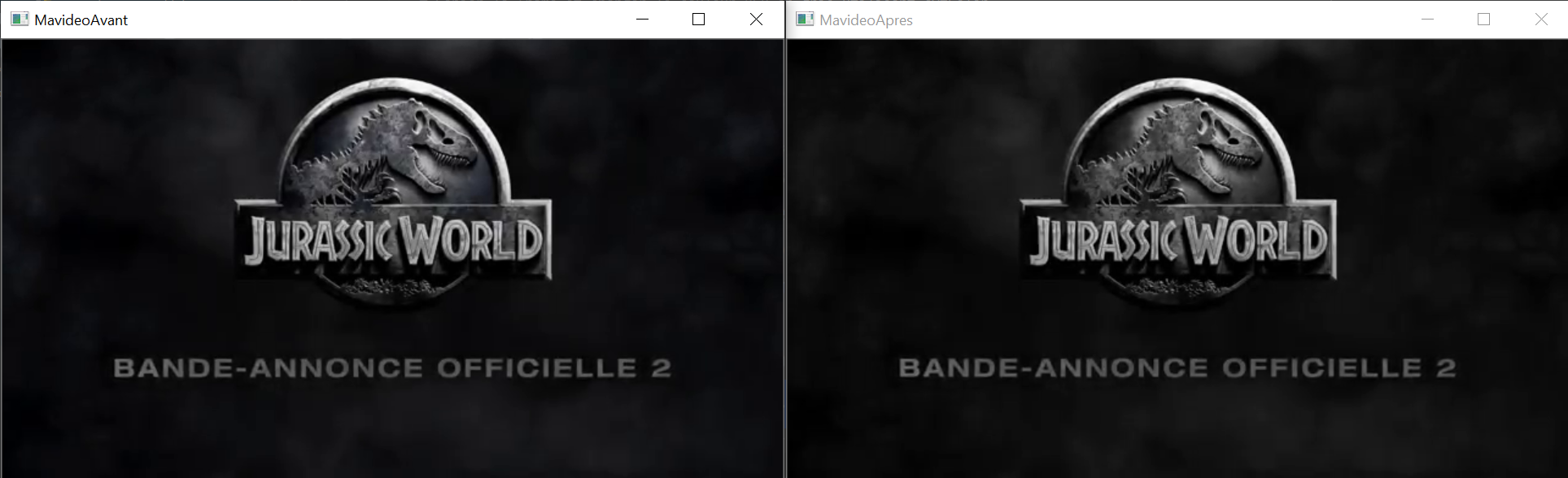
1. video\_processing.py

Code Source avec commentaires : 



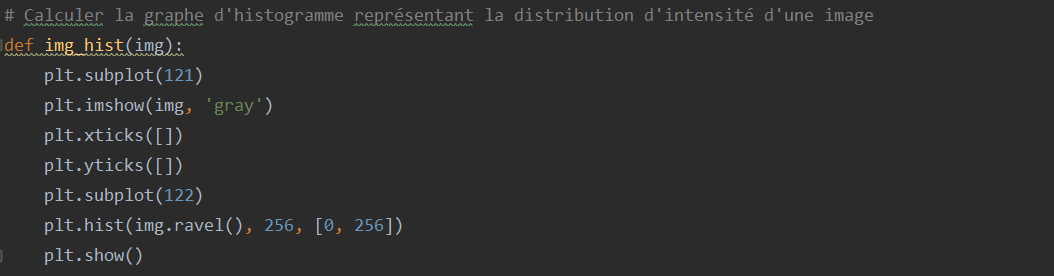
Résultat :





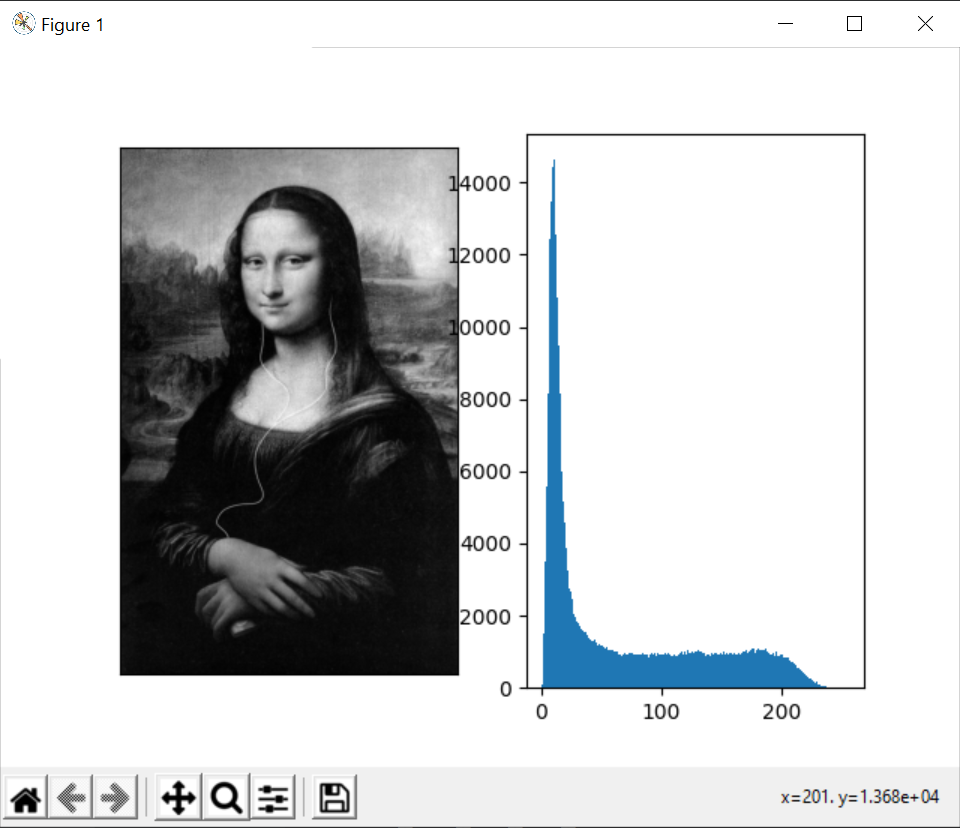
# Partie 1 : Type d’images et statistiques images

Q1. L’utilisation de la méthode *plt.histo* pour afficher le diagramme de histogramme.



Voici est la figure de l’histogramme de l’image *lisa.png*deniveau 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.



2

1

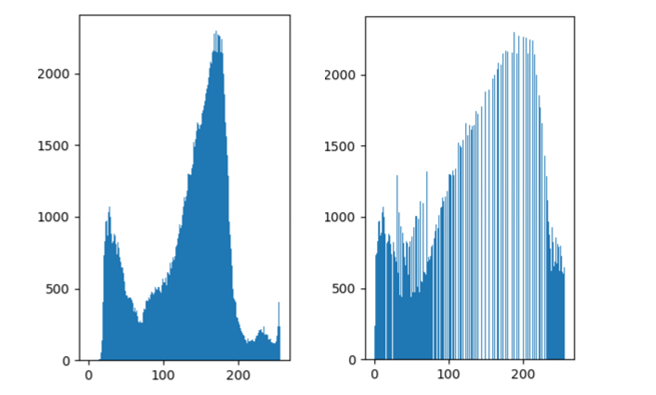
L’histogramme de niveau de gris

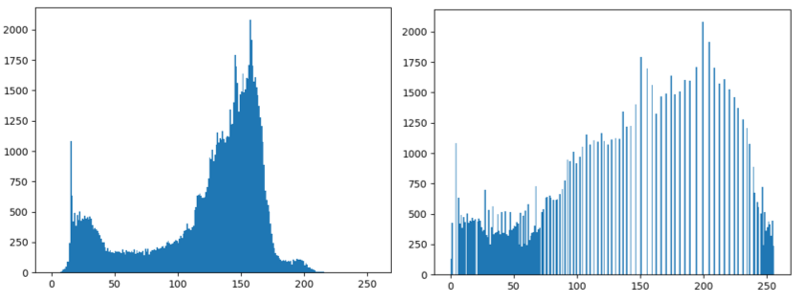
Q2. L’égalisation d’histogramme est une méthode d'ajustement du contraste d'une image numérique qui utilise l'histogramme.

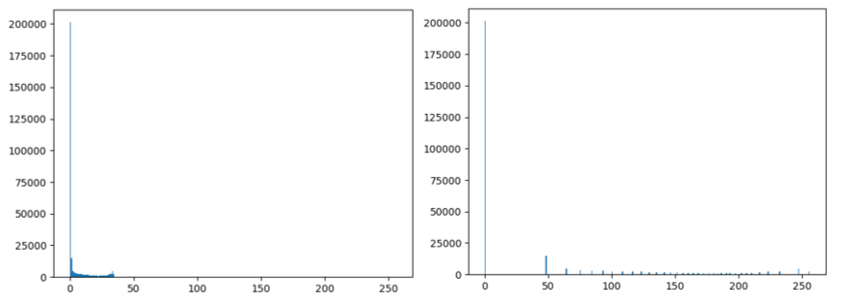
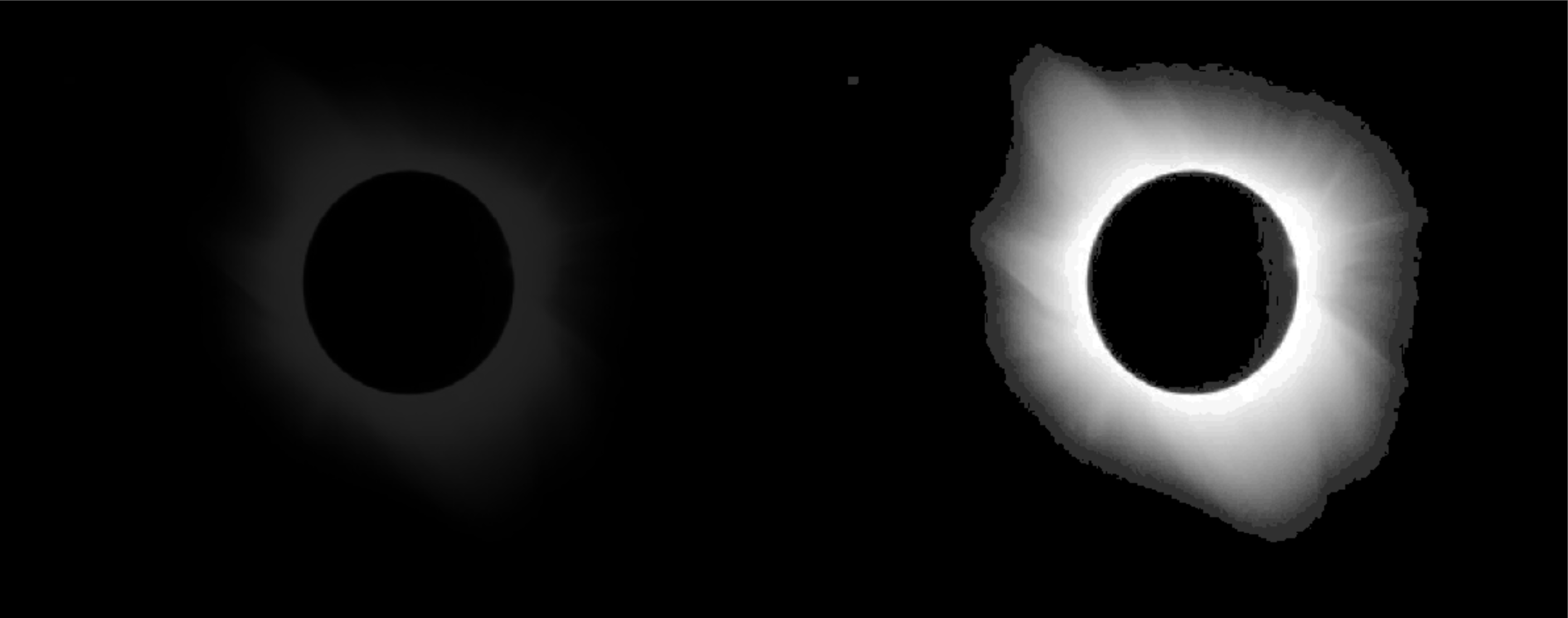


Ci-dessous est un exemple d’égaliser d’un histogramme de niveau de gris :

* 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

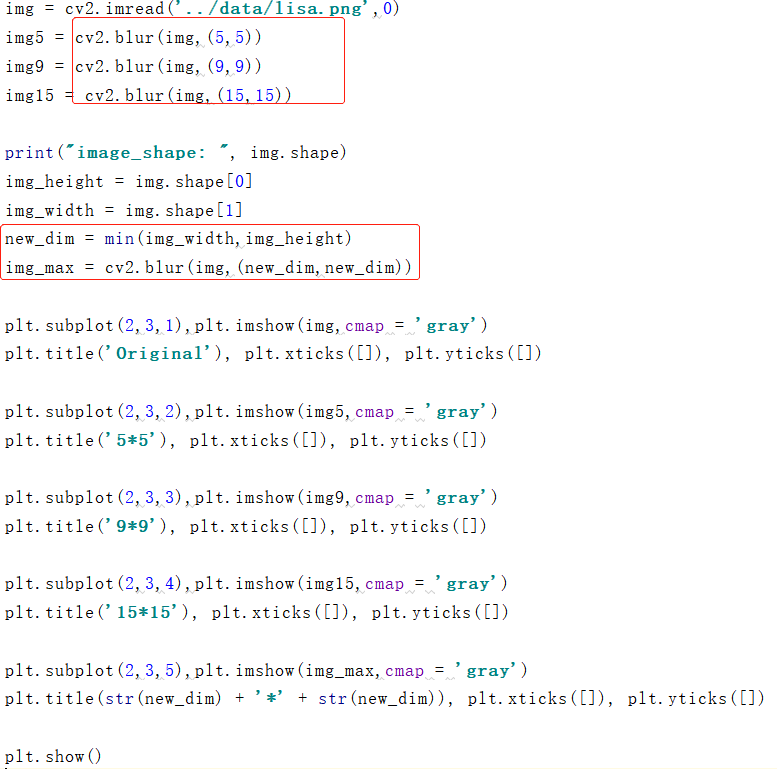
Que donnerait l’application d’un filtre moyenneur dont la dimension serait égale à celle de l’image ?

<https://docs.opencv.org/3.4.2/d2/d96/tutorial_py_table_of_contents_imgproc.html>

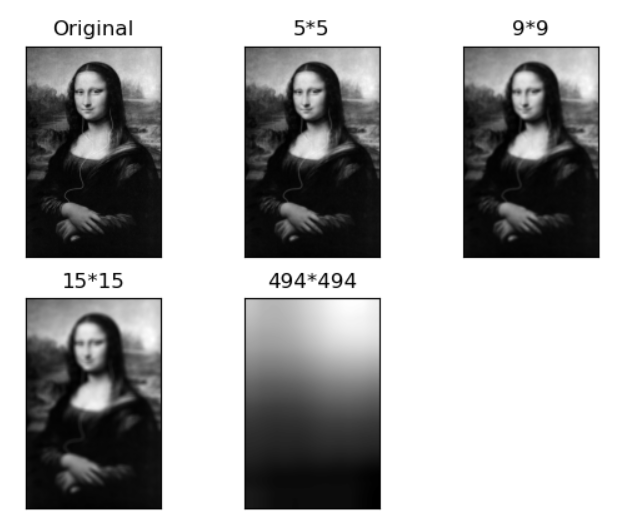
Code : *P2\_Q1\_filtre\_moyenneur.py*

* Le filtre moyenneur est une opération de traitement d’images utilisée pour réduire le bruit dans une image et/ou flouter une image.
* On choisi *cv2.blur()* pour faire lee filtre moyenneur

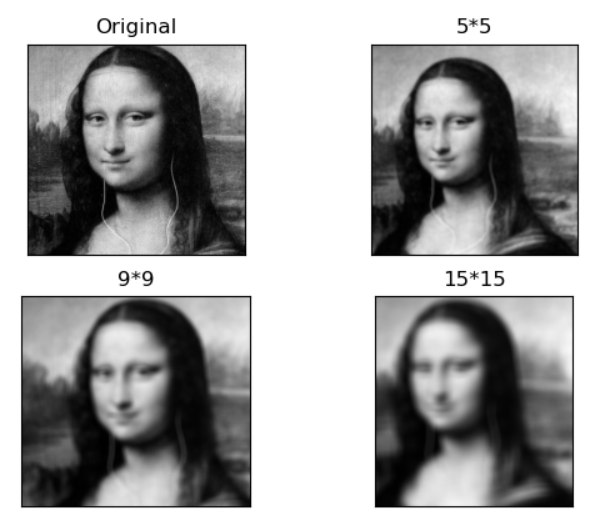
*cv2.blur(img, (i, i)) :* fait une moyenne dans un voisinage i x i (matrice de convolution avec tous les coefficients identiques et leur somme qui vaut et renvoie l'image résultat.

* 

Resultat :



* 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 :



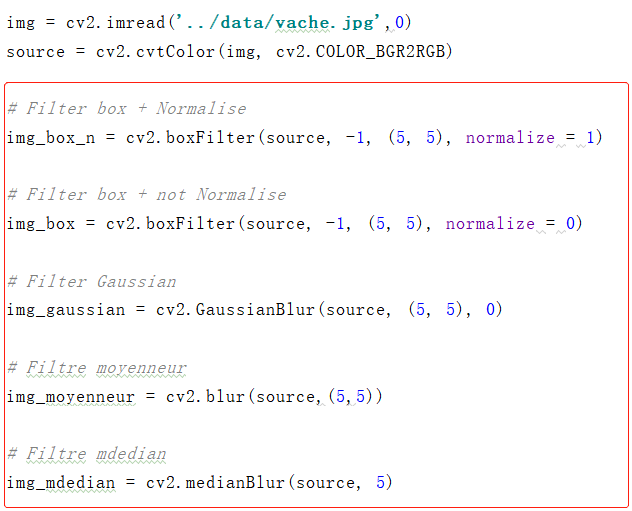


Le fonctionnement est principale de reduire les détails presque indépendants dans l’image, c’est-à-dire la zone de pixel qui est plus petite que la taille de la taille(masque) de filtre.

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



Resultat :





* 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 "normalisez" est vrai[1], il a besoin de traiter avec valeur de moyenne.

Lorsque "normalisez" est faux[0], il n'a pas besoin de traiter avec valeur de moyenne. En fait, il s'agit de la somme des autre pixels, il est donc facile de se produire un dépassement. Quand il se passe, les parties d'images est 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 moyenneur.
* Filtrage mdedian :
  + *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 convoluez la avec un filtre permettant de faire ressortir les traits horizontaux
* Ouvrez l’image suzan.jpg et convoluez la avec un filtre permettant de faire ressortir les traits verticaux

Expliquez les masques utilisés

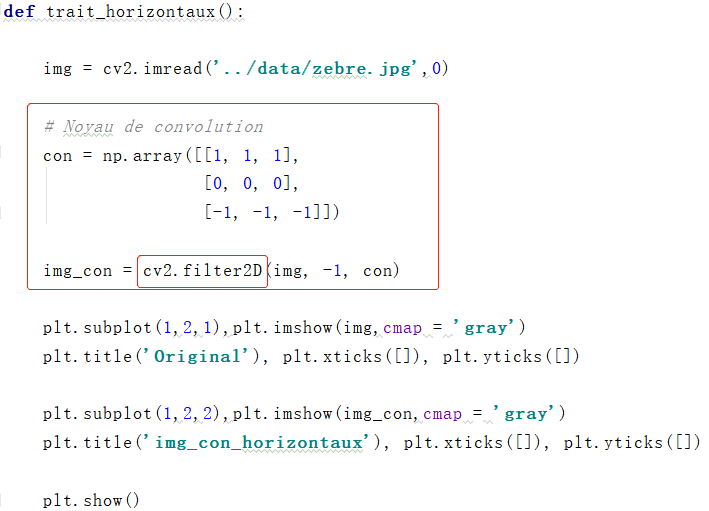
1. fore ressortir les traits horizontaux:

Noyau de convolution :

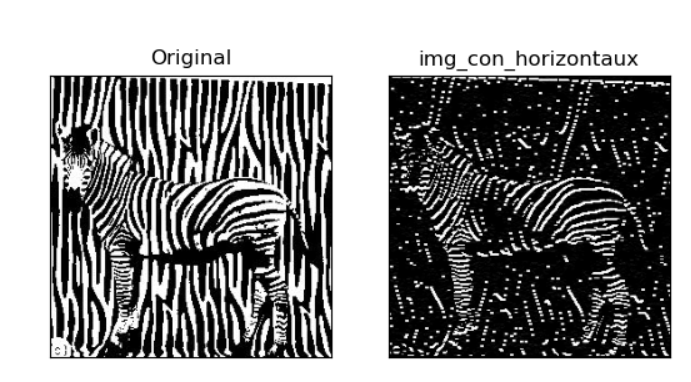


Code :

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



Resultat :



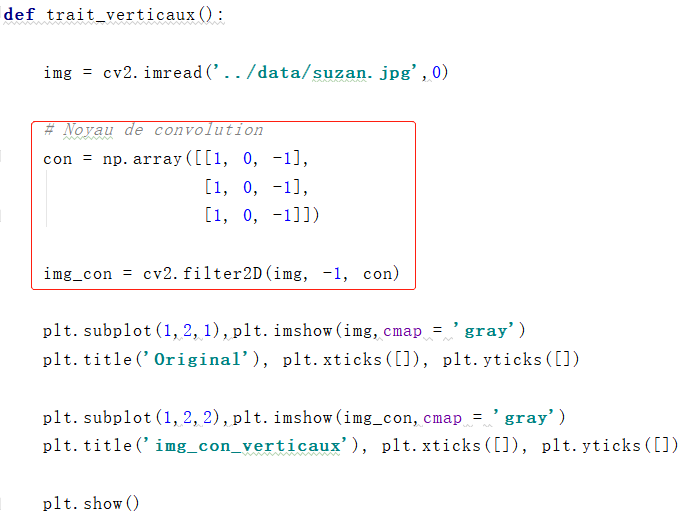
1. fore ressortir les traits verticaux:

Noyau de convolution :

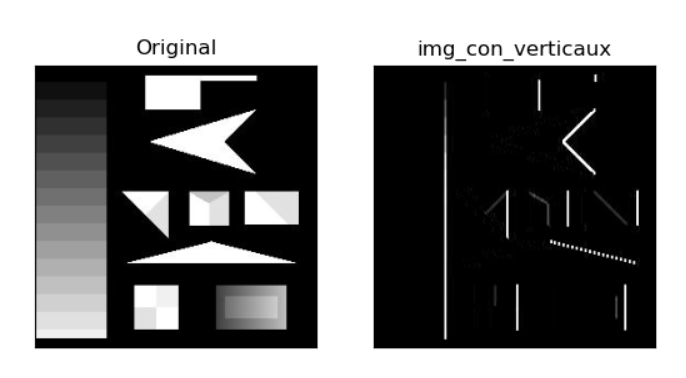


Code :

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



Resultat :



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. [在partie 0 补充算法/图片结果 讲解]

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

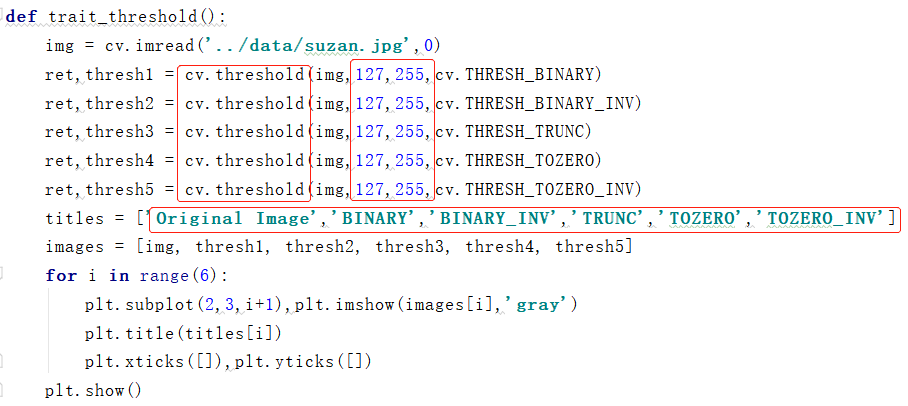
<https://docs.opencv.org/3.4.2/d2/d96/tutorial_py_table_of_contents_imgproc.html>

1. les traits de seuillage fix:

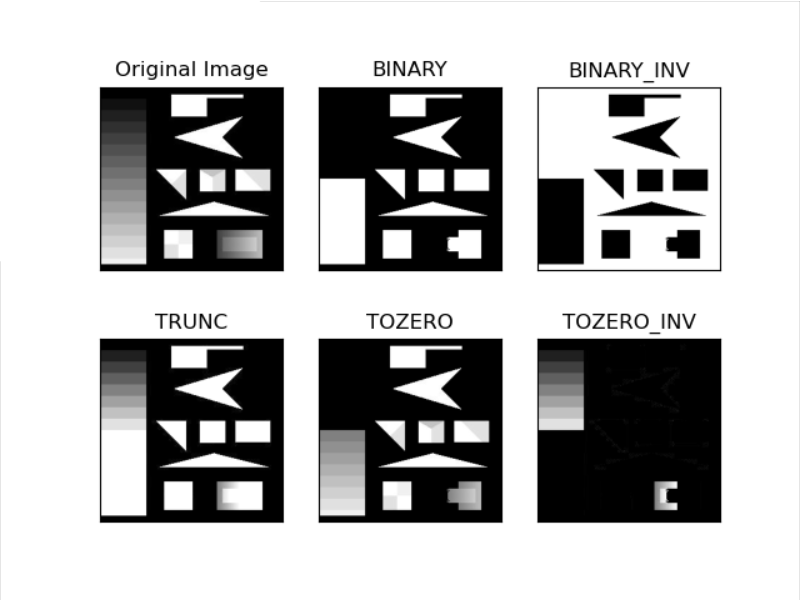
Test\_image : suzan.jpg

Code :

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



Resultat :

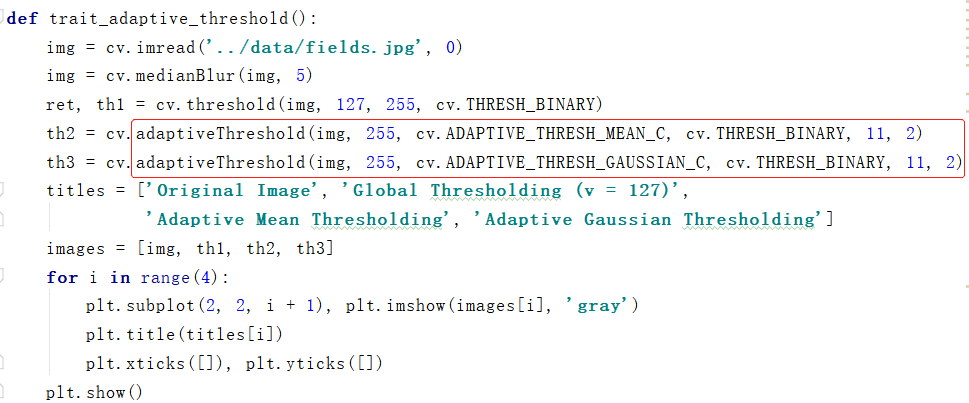


1. les traits de seuillage adaptatif:

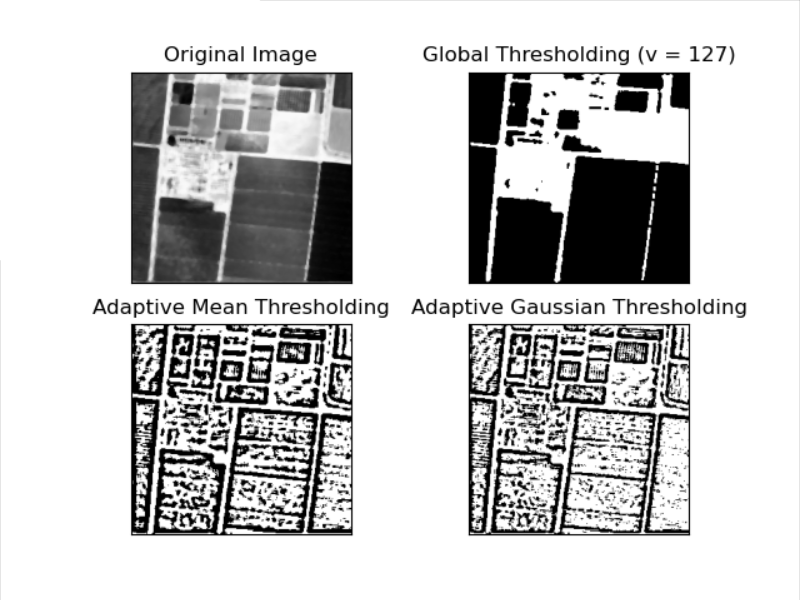
Test\_image : fields.jpg

Code :

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



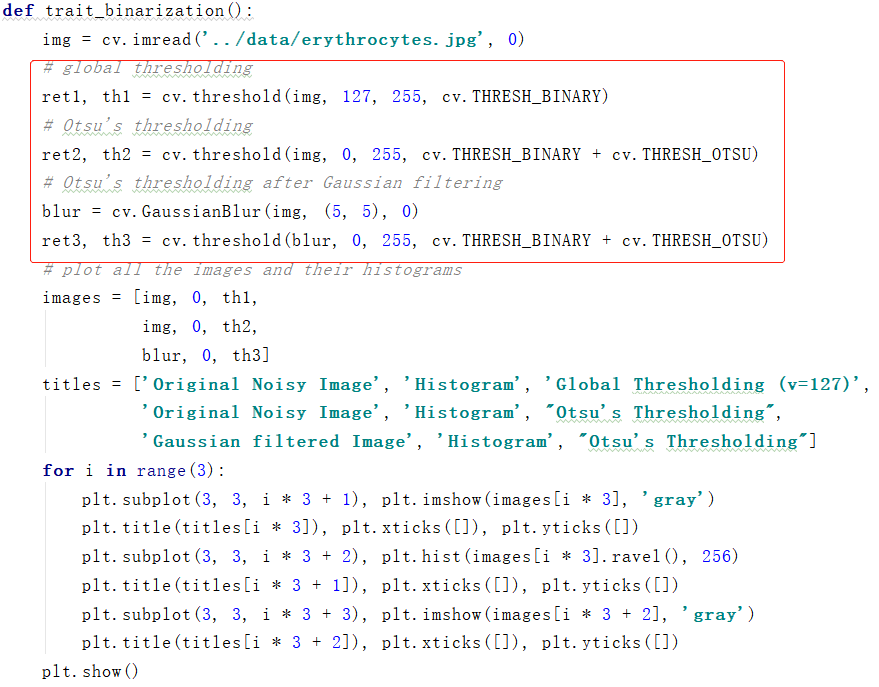
Resultat :



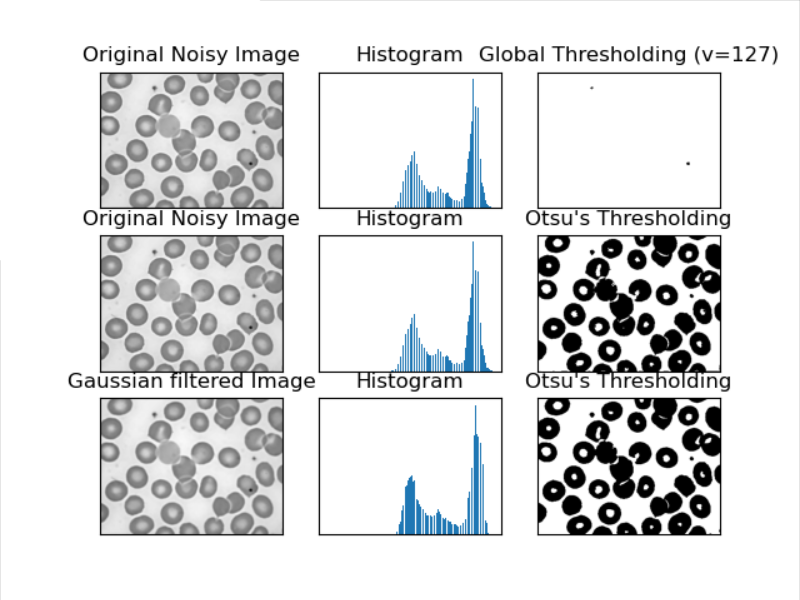
1. les traits de binarisation:

Test\_image : erythrocytes.jpg

Code :



Resultat :



[解释补充：<https://docs.opencv.org/3.4.2/d7/d4d/tutorial_py_thresholding.html> ]

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.

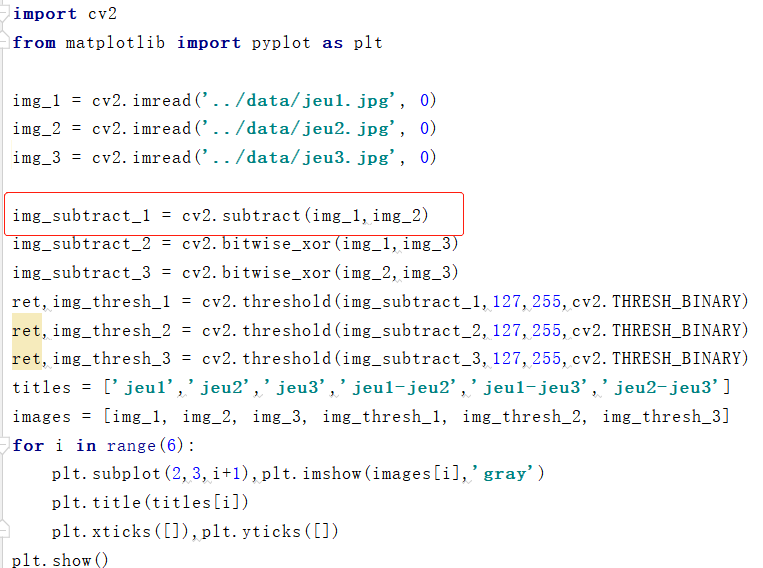
Proposez un algorithme capable de résoudre ce problème lorsque les images à comparer sont succeptibles d’etre décalées de quelques pixels l’une par rapport à l’autre (CF jeu3).



**L’algorithme :** XOR (*img*1 = *img*1⊕*img*2) ou Difference (*img*1 = |*img*1 − *img*2|)

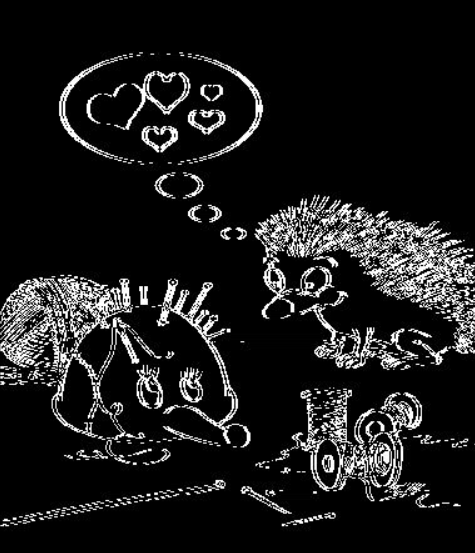
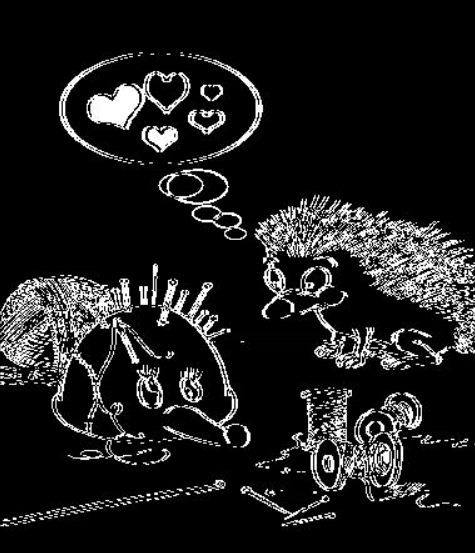
Code :

On utilise la fonction *subtract()* dans opencv pour comparerles 2 images.



Les images à comparer et les resultats:

jeu1 et jeu2 jeu1 et jeu3 jeu2 et jeu3

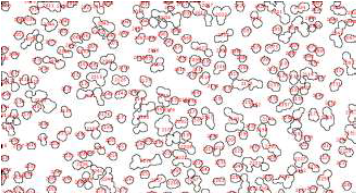


Q3: Complétez le code Video\_processing fourni afin d’appliquer une séquence d’opérations (soustraction entre 2 frames successives, binarisation, érosion/dilatation, …) avant affichage du résultat. Expliquez l’intérêt de cette séquence (opérations entre frames successives) ?

Voir la partie Video\_processing de Partie 0.

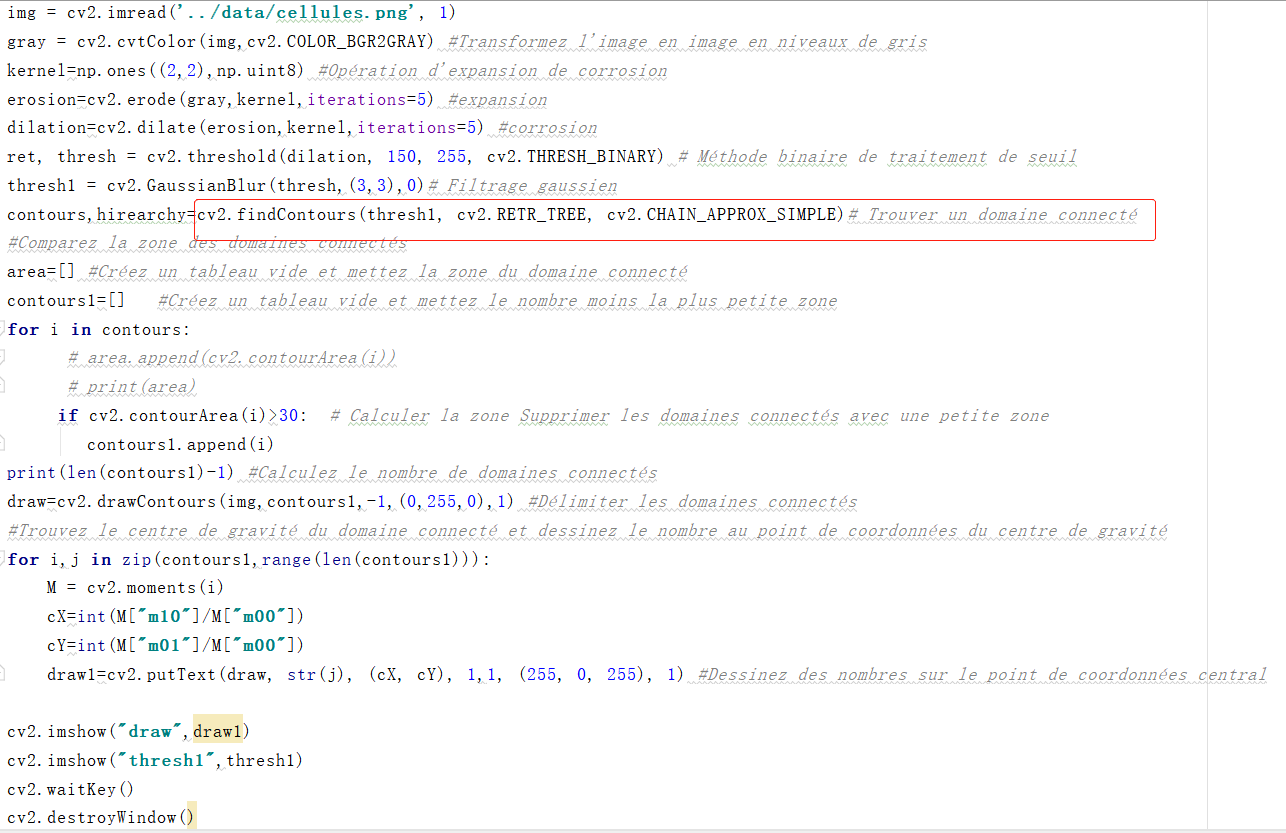
Q4: Des biologistes aimeraient compter le nombre de cellules présentes dans des images. Proposez un algorithme permettant de compter les cellules automatiquement. Vérifiez la présence de 329 cellules dans l’image cellule.png.

<https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga107a78bf7cd25dec05fb4dfc5c9e765f>

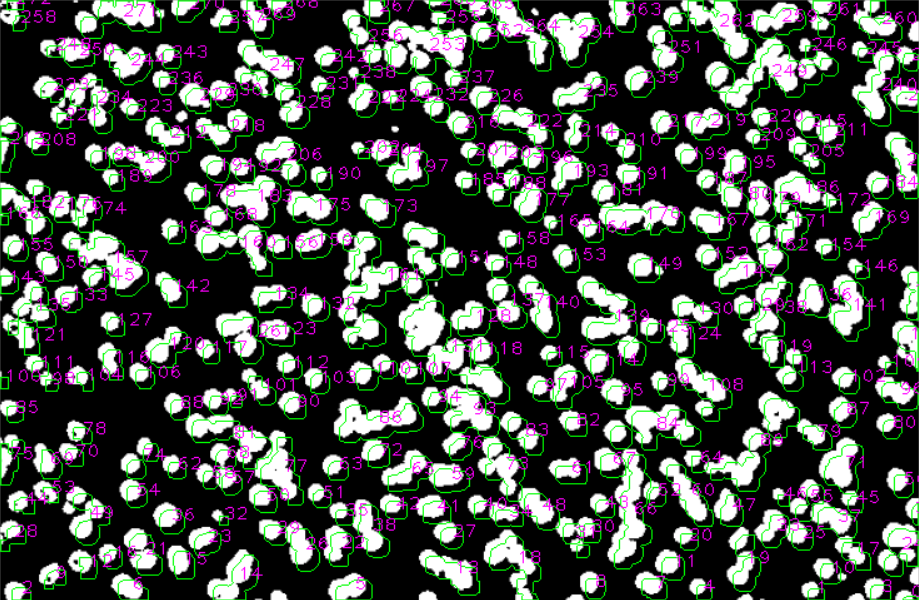


Code :

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



Resultat : 272



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.

# 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.*

# Partie 5 : Vidéos, Détection de changement de plan et résumé automatique de vidéo

Q1 : Changement de plan

Exploitez certaines des méthodes vues précédemment pour mettre en place un mécanisme de détection automatique de changement de plan dans les vidéos. Testez et commentez ses performances. Proposez sans le coder un mécanisme que vous jugeriez plus performant.

Q2 : Création d’Images résumés de plan

Exploiter l’ensemble des méthodes vues précédemment pour mettre en place un mécanisme de génération automatique de résumés de plans dans les vidéos. Testez et commentez ses performances. Proposez sans le coder un mécanisme que vous jugeriez plus performant.