
Traitement d'images

Cahier d'exercices en Python



Rudi Giot

1re édition 2021 - Dernière révision le mardi 7 septembre 2021

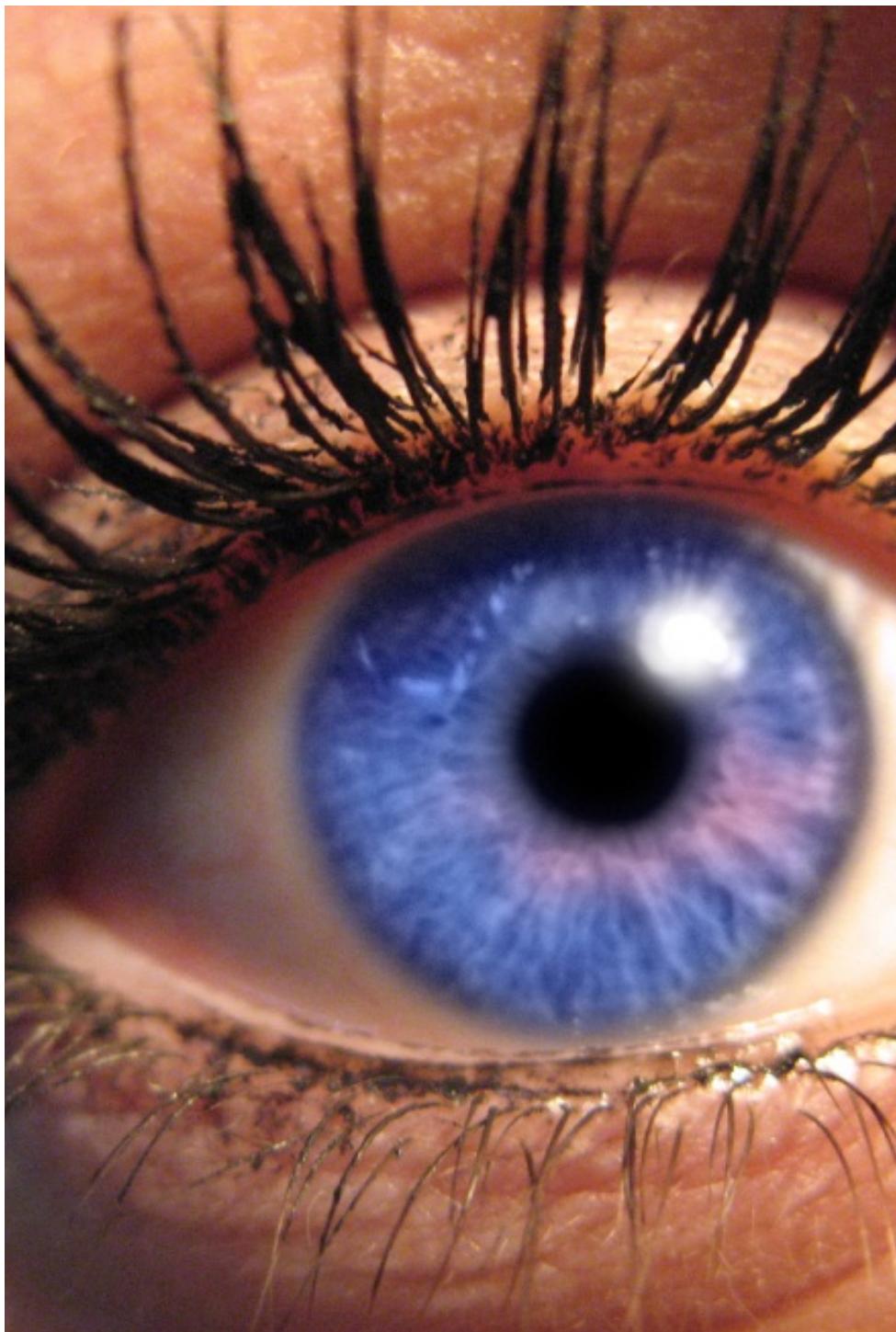
Introduction.....	4
<i>Installation</i>	<i>5</i>
<i>Affichage d'une image</i>	<i>6</i>
<i>Sauvegarde d'une image</i>	<i>6</i>
Les couleurs	7
<i>RGB</i>	<i>8</i>
<i>CMY / CMYK.....</i>	<i>9</i>
<i>HSL</i>	<i>10</i>
Transformations géométriques	12
<i>Redimensionnement.....</i>	<i>13</i>
<i>Extraction (crop).....</i>	<i>13</i>
<i>Inversion-Retournement (flip)</i>	<i>13</i>
<i>Rotation</i>	<i>14</i>
<i>Translation</i>	<i>14</i>
Histogramme et projection.....	15
<i>Histogramme</i>	<i>16</i>
<i>Projection.....</i>	<i>18</i>
Binarisation et LUT	19
<i>Binarisation</i>	<i>20</i>
<i>LUT</i>	<i>21</i>
<i>Seuillage adaptatif.....</i>	<i>23</i>
Opérations entre images	24
<i>Somme</i>	<i>25</i>
<i>Différence</i>	<i>25</i>
Opération de voisinage.....	26
<i>Convolution</i>	<i>27</i>

<i>Filtres non linéaires</i>	27
Opérations morphologiques	28
<i>Dilatation</i>	29
<i>Erosion</i>	29
<i>Ouverture</i>	29
<i>Fermeture</i>	30
Labellisation.....	31
Détection d'objets et OCR	33
<i>Correlation</i>	34
<i>Intelligence artificielle</i>	34
Bibliographie	35
<i>Livres</i>	35
<i>Liens Internet</i>	35
<i>Videos</i>	35

Chapitre 1

Introduction

Nous allons d'abord installer *Python3* et les bibliothèques nécessaires aux exercices, ensuite nous choisirons et préparerons un environnement de développement (*IDE*) et finalement nous réaliserons nos premiers programmes pour afficher et sauvegarder une image.



Installation

Tous les exercices seront réalisés en *Python* version 3, assurez-vous de bien avoir cette version installée sur votre ordinateur, en utilisant l'instruction :

```
Python3 --version
```

```
[MacdeRudi-2:~ RudiGiot$ Python3 --version
Python 3.6.2
```

Vous devez ensuite installer la librairie *OpenCV* qui est un standard en traitement d'images. Vous pouvez le faire simplement avec l'instruction :

```
pip install opencv-contrib-python
```

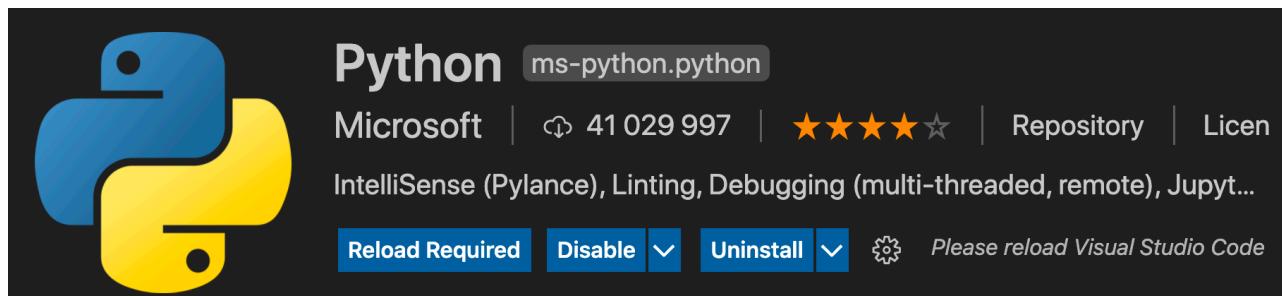
```
Python 3.6.2
[MacdeRudi-2:~ RudiGiot$ pip install opencv-contrib-python
Collecting opencv-contrib-python
  Downloading opencv_contrib_python-4.5.3.56-cp36-cp36m-macosx_10_15_x86_64.whl (51.4 MB)
    |████████| 51.4 MB 12.2 MB/s
Requirement already satisfied: numpy>=1.13.3 in /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages (from opencv-contrib-python) (1.16.3)
Installing collected packages: opencv-contrib-python
Successfully installed opencv-contrib-python-4.5.3.56
```

Vous remarquerez qu'une autre librairie (*numpy*) est requise pour cette installation, pour ma part elle était déjà installée :

```
Requirement already satisfied: numpy>=1.13.3 in ...
```

Si ce n'est pas le cas sur votre machine, installez cette librairie *numpy* comme vous l'avez fait pour la librairie *OpenCV* (avec l'instruction *pip*).

Vous devez ensuite choisir votre *IDE*, pour ma part, j'utilise *Visual Studio Code* avec l'extension *Python* de *Microsoft*. Mais vous pouvez utiliser *PyCharm* ou même un simple éditeur de texte, vous êtes libre de choisir votre environnement de travail.



Affichage d'une image

Une fois les installations terminées, nous pouvons commencer à programmer notre première application qui va juste afficher une image.

Remarque : les images utilisées dans les exercices vous seront fournies sur GIT ([mettre ici le lien](#)) ainsi que tous les codes de base et les solutions des exercices.

```
import cv2 as cv
img = cv.imread('path/testImages/Lena.png')
cv.imshow('Lena', img)
cv.waitKey(0)
```

Le code est très explicite, on charge d'abord la librairie, on lit ensuite l'image (faites attention de bien spécifier le chemin (*path*) vers cette image), ensuite on l'affiche et finalement la dernière instruction empêche le programme de se terminer trop rapidement avant d'avoir vu la fenêtre contenant l'image à afficher.

Sauvegarde d'une image

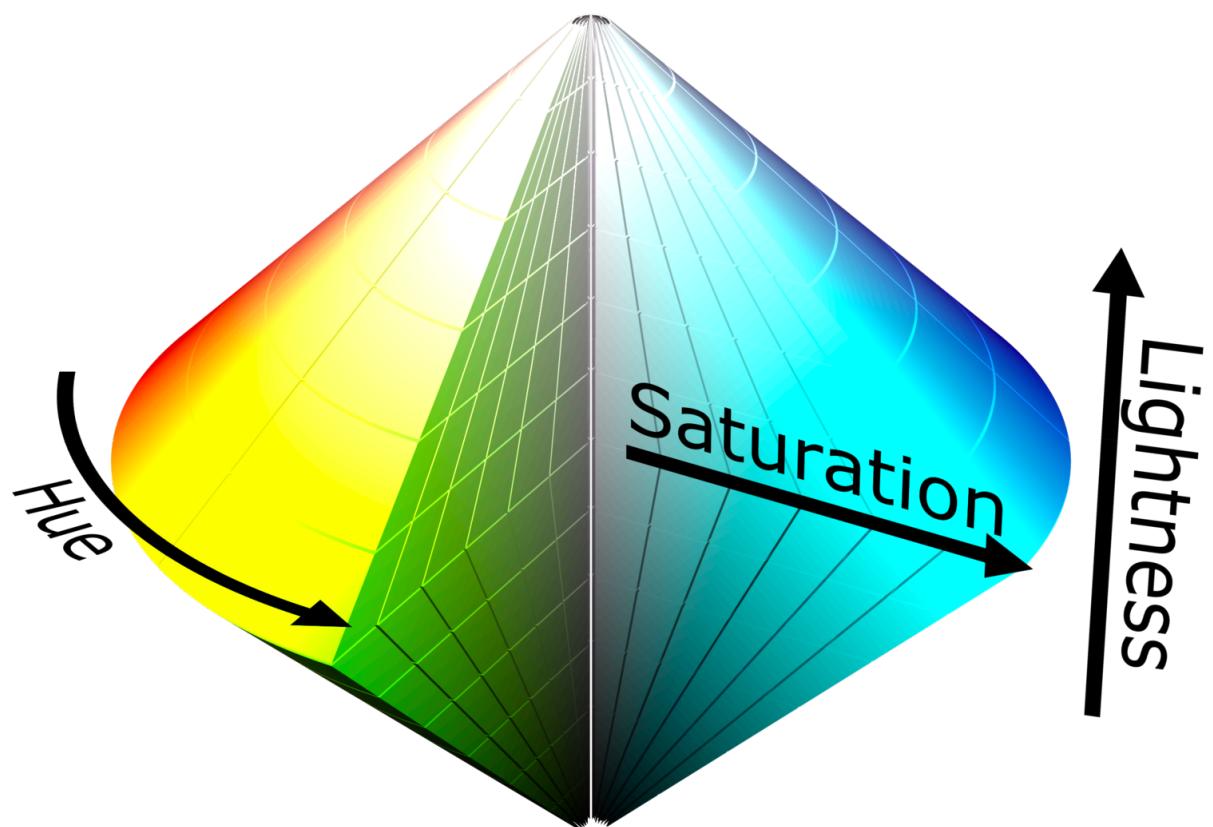
Il est parfois nécessaire de sauvegarder le résultat des différents traitements que nous aurons effectués sur l'image. Nous utiliserons alors la méthode *imwrite()* :

```
import cv2 as cv
img = cv.imread('path/testImages/lena.png')
cv.imwrite('path/testImages/lenaCopy.png', img)
```

Chapitre 2

Les couleurs

Nous allons dans ce chapitre aborder les différentes façons de coder des couleurs (*RGB*, *CMYK*, *HSL*, *Grayscale*) et surtout voir comment les exploiter judicieusement dans le cadre de traitements d'images.



RGB

Par défaut les images importées sont stockées en *RGB*. C'est à dire que chaque pixel comporte trois composantes, une rouge, une verte et une bleue. Ces trois composantes sont généralement codées sur 8 bits.

Exercices :

- Déterminez la place qu'occupe une image de 653 lignes sur 657 colonnes si elle est sauvegardée sans compression de données
- Vérifiez la taille trouvée avec la place occupée par l'image « *cat.bmp* »
- Expliquez pourquoi la taille théorique n'est pas exactement la taille sur le disque
- Calculez le nombre de couleurs que l'on peut coder sur 3x8bits
- Posez vous la question de savoir si ce nombre de couleurs est « suffisant »
- Recherchez la signification et l'utilité du *A* dans le *RGBA*

Pour accéder aux valeurs de chacun des pixels, on peut simplement faire :

```
print(img[100,20])
```

pour obtenir les trois composantes (*RGB*) du pixel qui retrouve sur la centième ligne et la vingtième colonne. Attention, les composantes sont données dans l'ordre *BGR* : l'indice 0 correspond au bleu, l'indice 1 correspond au vert et l'indice 2 correspond au rouge.

Nous pouvons également obtenir la largeur et la hauteur d'une image grâce aux propriétés *shape[0]* et *shape[1]* de l'image, par exemple :

```
print(img.shape[0])
print(img.shape[1])
```

permettent d'afficher respectivement la largeur et la hauteur de l'image, tandis que *shape[2]* donne le nombre d'octet par pixel (une valeur de 3 pour une image codée en *RGB* et une valeur de 1 pour des images en niveau de gris).

Exercices :

- Réalisez un programme pour extraire la composante rouge d'une image donnée et affichez-là
- Sachant qu'un niveau de gris correspond à des valeurs identiques pour les trois composantes RGB, calculez le nombre de niveau de gris que l'on peut coder
- En utilisant les formules du cours (voir syllabus) transformez l'image en niveau de gris
- Essayez ensuite la fonction `cvtColor(img, cv.COLOR_BGR2GRAY)` et comparez le résultat avec l'exercice précédent
- Testez d'autres constantes que `COLOR_BGR2GRAY`¹ pour transformer les codes couleur et commentez leur intérêt

CMY / CMYK

Ce type de codage est surtout utilisé en imprimerie, nous l'utiliserons rarement en traitement d'image.

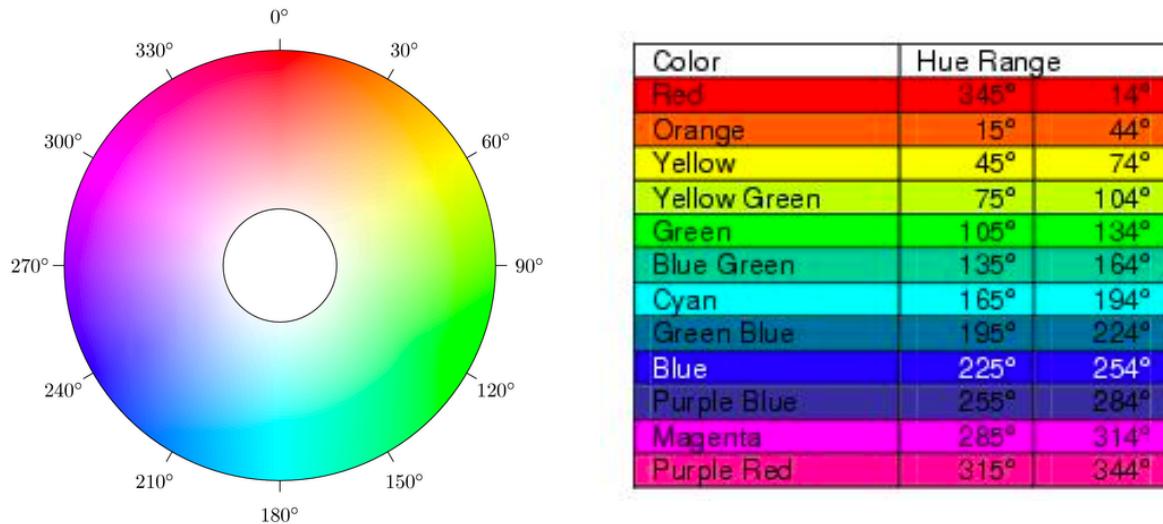
Exercices :

- En utilisant les formules du cours (voir syllabus), réalisez un programme pour extraire la composante *Cyan* d'une image donnée, affichez là
- En utilisant les formules du cours (voir syllabus), réalisez un programme pour extraire la composante *noire* (*K*) d'une image donnée, affichez là
- Trouvez la constante `cv.COLOR_BGR2...` adéquate pour extraire la composante *Cyan* et exploitez là pour refaire l'exercice précédent, comparez les résultats

¹ https://docs.opencv.org/3.4/d8/do1/group__imgproc__color__conversions.html

HSL

Ce type de codage est beaucoup plus intéressant car il permet d'extraire la teinte (*H*) d'un pixel. Cette teinte est généralement représentée par un angle sur le cercle des couleurs.



Le *L* (pour luminance) correspond au niveau de gris que l'on a déjà vu plus haut, tandis que la saturation (*S*) indique sa « pureté ». Une couleur très saturée est vive et intense tandis qu'une couleur peu saturée est plus terne (*pastel*) et devient grise à une saturation nulle.



Vous trouverez parfois des modèles, dans lesquels, on a à la place du *L* (*Lightness*) un *B* (pour *Brightness*) ou un *V* (pour *Value*).

Exercices :

- En utilisant les formules du cours (voir syllabus), réalisez un programme pour extraire la composante *S* (saturation) d'une image donnée et l'afficher
- Trouvez la constante *cv.COLOR_BGR2...* adéquate pour extraire la composante *Hue* (teinte) et affichez-là (attention aux valeurs qui dépassent 255)

Projet : Réalisez l'effet « Sin City » sur l'image « SinCity.jpg ». Vous devez pour cela transformer l'image en niveau de gris, sauf pour les portions « rouges » qui doivent devenir rouge vif. Regardez l'exemple suivant :



<https://www.lesinrocks.com/wp-content/uploads/2014/09/sin-city.jpg>

Chapitre 3

Transformations géométriques

Dans ce chapitre, nous allons voir comment redimensionner une image. En effet, souvent les calculs en traitement d'images peuvent être consommateurs de ressources processeur. Dans certain cas, on peut donc réduire la taille de l'image pour diminuer le nombre d'opération et accélérer le traitement. Nous verrons aussi comment extraire des morceaux d'une image (*cropping*). Nous aborderons aussi la translation et le retournement et nous terminerons ce chapitre avec les rotations, qui sont très utiles pour traiter, par exemple, des images numérisées « de travers ».



Redimensionnement

Pour changer la taille de l'image, on peut utiliser la fonction *resize()* qui prend comme paramètres : l'image à redimensionner, la nouvelle taille de l'image (en pixel) et la manière dont l'interpolation se fait.

```
scale = (200, 200)  
resizedImage = cv.resize(img, scale, interpolation=cv.INTER_AREA)
```

Exercices :

- Essayez différents type d'interpolation² (particulièrement le *CUBIC*) pour réduire, agrandir, écraser, étirer les images et observez les différences
- Faites une fonction qui redimensionne l'image en fonction d'un paramètre *scale* donné en pourcent. Pour rappel, on peut obtenir la largeur et la hauteur d'une image grâce aux propriétés *shape[0]* et *shape[1]*

Extraction (crop)

Pour extraire une partie d'image un extrait, il suffit, par exemple, de faire :

```
imgCropped = img[0:200,0:200]
```

pour obtenir une image carrée qui contient les 200 premières ligne et colonnes de l'image originale.

Exercice : Par essai/erreur extrayez la tête du chat (*cat.bmp*) et affichez la à l'écran

Inversion-Retournement (flip)

Inverser géométriquement (horizontalement ou verticalement) une image est très simple à réaliser. Vous pouvez utiliser la fonction :

```
cv.flip(img, 1)
```

Exercice:

- Testez la fonction précédente pour réaliser une inversion géométrique d'une image

² https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html

Rotation

La rotation consiste en une transformation affine de la matrice de pixels. Elle peut être réalisée simplement en utilisant les équations suivantes :

$$\begin{aligned}x' &= x * \cos(\theta) - y * \sin(\theta) \\y' &= y * \sin(\theta) + x * \cos(\theta)\end{aligned}$$

où (x, y) est la position actuelle du pixel dans l'image et (x', y') sa position après rotation d'un angle θ .

Exercices (en utilisant le système ci-dessus):

- Ecrivez une fonction qui fait tourner une image d'un angle donné
- Essayez ensuite différents angles, visualisez les résultats, discutez des problèmes qui apparaissent, imaginez comment résoudre ces problèmes
- Faites subir à une image une rotation suivie d'une autre rotation, observez les problèmes qui apparaissent, imaginez comment résoudre ces problèmes
- Faites une rotation d'un angle de 65 degrés et ensuite la rotation inverse de -65 degrés, observez les différences qui apparaissent entre l'original et l'image qui a tourné deux fois

Exercices:

- Il y a évidemment une fonction dans *OpenCV* qui permet de réaliser la rotation. Cherchez là dans la documentation et exploitez là pour refaire les exercices précédent.

Translation

Réaliser une translation consiste simplement à déplacer tous les pixels, d'un certain nombre de pixels vers le haut, le bas, la gauche et/ou la droite.

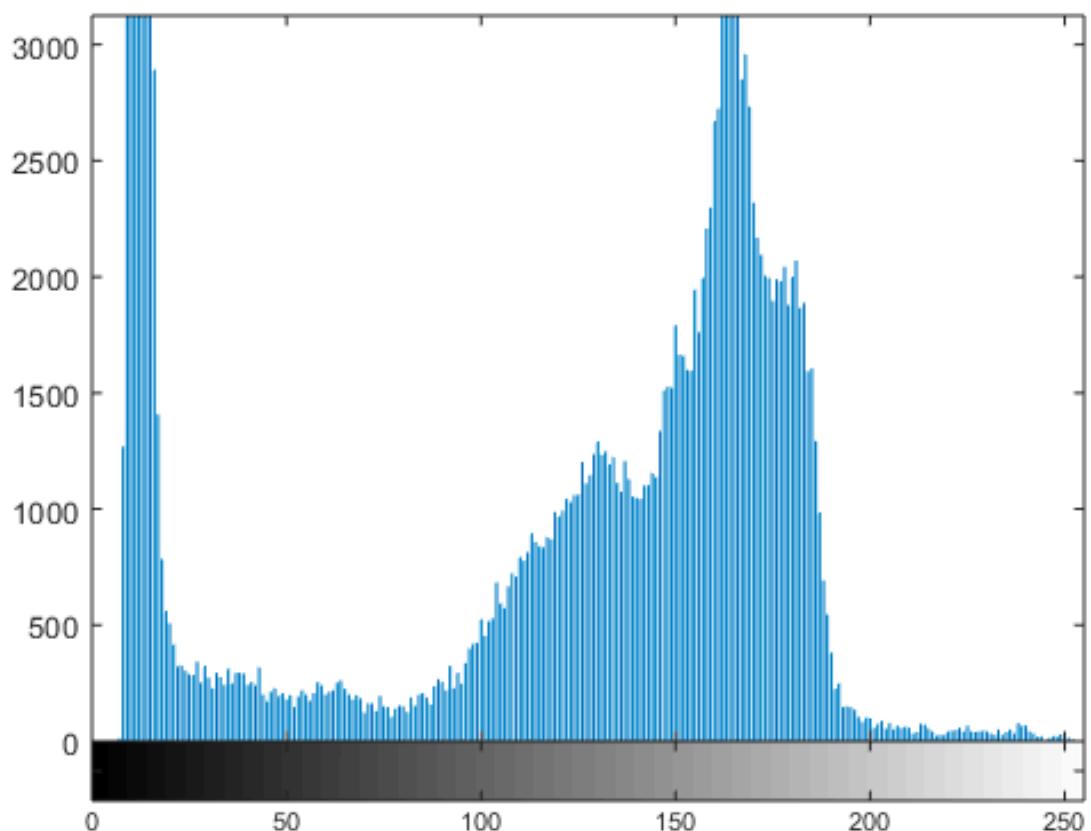
Exercice:

- Imaginez une fonction pour réaliser une translation sur une image

Chapitre 4

Histogramme et projection

Nous allons dans ce chapitre aborder les notions d'histogrammes et de projection qui sont des outils statistiques de base, essentiellement utilisés pour des estimations de paramètres nécessaires à d'autres traitement.



Histogramme

L'histogramme n'est pas vraiment un traitement d'image mais plutôt un outil statistique qui détermine la quantité de pixels ayant la même composante couleur (ou niveau de gris). Par exemple, pour une image en niveau de gris, on aura en abscisse les 256 valeurs possibles et en ordonnée le nombre de pixels ayant ces valeurs de gris (voir l'illustration de la page précédente).

Pour visualiser un histogramme, il faut donc créer une image. Pour ce faire, nous allons utiliser la librairie *numpy* pour créer des matrices qui seront utilisées comme images :

```
import numpy as np
import cv2 as cv

img = np.ones((255,300,1),np.uint8)*255
cv.imshow('image',img)
cv.waitKey(0)
```

qui dans ce cas crée une image en niveau de gris avec tous les pixels blancs.

Si vous voulez créer une image couleur, il faut trois composantes couleurs, le code devient par exemple :

```
img = np.zeros((255,300,3),dtype=np.uint8)
img[:] = 255, 0, 0
```

pour créer une image bleue.

Vous pouvez alors utiliser les instructions suivantes pour tracer des lignes, dessiner des rectangles, des cercles et écrire du texte :

```
cv.line(img,(0,0),(img.shape[1],img.shape[0]),(0,255,0),3)
cv.rectangle(img,(0,0),(250,350),(0,0,255),2)
cv.circle(img,(400,50),30,(255,255,0),5)
cv.putText(img, "Texte à insérer",
           (300,200),cv.FONT_HERSHEY_COMPLEX,1,(0,150,0),3)
```

Exercices :

- Dessinez un carré rempli de couleur rouge sur fond blanc, sans utiliser la fonction rectangle
- Comptez le nombre de pixels de chaque niveau de gris de l'image « *lena.png* » , construisez une nouvelle image dans laquelle vous allez « dessiner » l'histogramme et affichez la
- Appliquez cet algorithme à différentes images en imaginant au préalable à quoi il devrait ressembler

Plutôt que de compter les pixels avec des boucles, vous pouvez directement utiliser la fonction de *OpenCV* :

```
hist = cv.calcHist([img],[0],None,[256],[0,256])
```

où la variable *hist* est un tableau de 256 valeurs correspondant au nombre de pixels avec un niveau de gris correspondant à l'indice. Par exemple, *hist[134]* donne le nombre de pixel ayant comme niveau de gris 134.

Exercices :

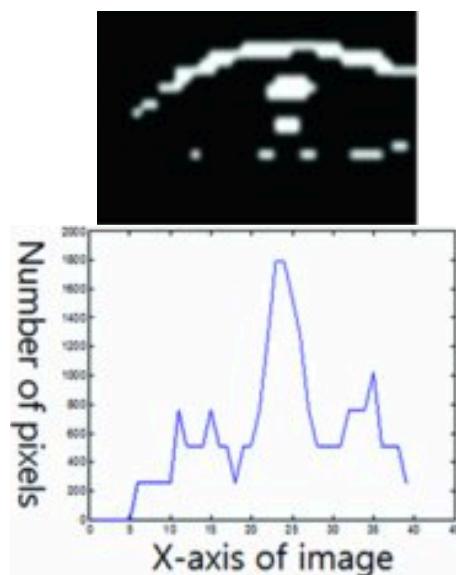
- Affichez l'histogramme sous forme d'image en utilisant la fonction ci-dessus
- Comparez cette visualisation avec celle des exercices précédents et estimez le temps de calcul pour voir laquelle est la plus rapide

Remarque : Il existe des librairies Python qui simplifie l'affichage de graph, dont les histogrammes, vous pouvez, par exemple, vous servir de *Matplotlib*. L'apprentissage de cette librairie ne fait pas partie de ce cours mais vous pouvez les utiliser si vous le souhaitez.

Projection

La projection n'est pas non plus un traitement à proprement parler. Il donne des indication pour déduire certains paramètres à appliquer sur des traitements. L'idée de la projection est de parcourir l'image verticalement (ou horizontalement) pour évaluer la moyenne du niveau de gris sur une bande verticale (ou horizontale) de un pixel de large (ou un pixel de haut). On peut ainsi obtenir une « forme d'onde » (*waveform*) de la projection verticale (ou horizontale).

Cette projection peut aussi être calculée sur une image binaire (en noir et blanc) en comptant simplement le nombre de pixel noir dans une bande verticale.



https://www.spiedigitallibrary.org/Content/Images/Journals/OPEGAR/50/12/127202/FigureImages/127202_1_13.jpg

Exercices :

- Calculez la projection verticale de l'image « *lena.png* » et visualisez la forme d'onde
- Refaites la même chose mais horizontalement
- Appliquez ces deux algorithmes à l'image « *SudokuGridEmpty.png* » qui est en noir et blanc

Projets :

- Utilisez la projection et la rotation pour redresser automatiquement une grille de Sudoku qui a été numérisée « de travers » dans l'image « *SudokuGridRotated.jpg* »
- Utilisez la projection et le *crop* pour extraire automatiquement toutes les cases (séparément) d'une grille de Sudoku dans l'image « *SudokuGrid.png* »

Chapitre 5

Binarisation et LUT

La *Look Up Table (LUT)* est un outil qui permet de transformer un niveau de gris (ou une composante couleur) en une autre valeur définie dans un tableau tandis que la binarisation est un traitement qui réduit le nombre de couleur dans une image à deux : le noir et le blanc. On peut obtenir une binarisation en utilisant une LUT particulière.



Binarisation

La binarisation est un traitement d'image de base, très simple à programmer. On l'applique généralement sur une image en niveau de gris (ou sur une composante de couleur). Il faut d'abord fixer un seuil (*threshold*) de niveau de gris en dessous duquel les pixels deviennent noirs et au dessus duquel ils deviennent blancs. L'image résultante ne présente dès lors plus que deux valeurs (couleurs) possible le noir ou le blanc.

En OpenCV il suffit de préciser sur quelle image on veut effectuer l'opération de binarisation (*thresholding*), le seuil (*threshold*) et le niveau de gris qui affichera le résultat (en général, on met 255 pour le blanc) :

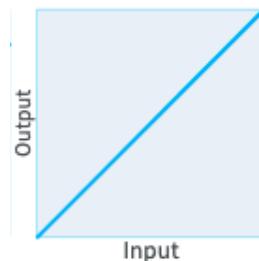
```
binary=cv.threshold(grayImage,150,255,cv.THRESH_BINARY)
```

Exercices :

- Testez cette instruction sur quelques images en niveau de gris, changez le seuil et observer les différences
- A partir d'une image couleur appliquez la binarisation sur la composante de saturation, changez le seuil et observer les différences

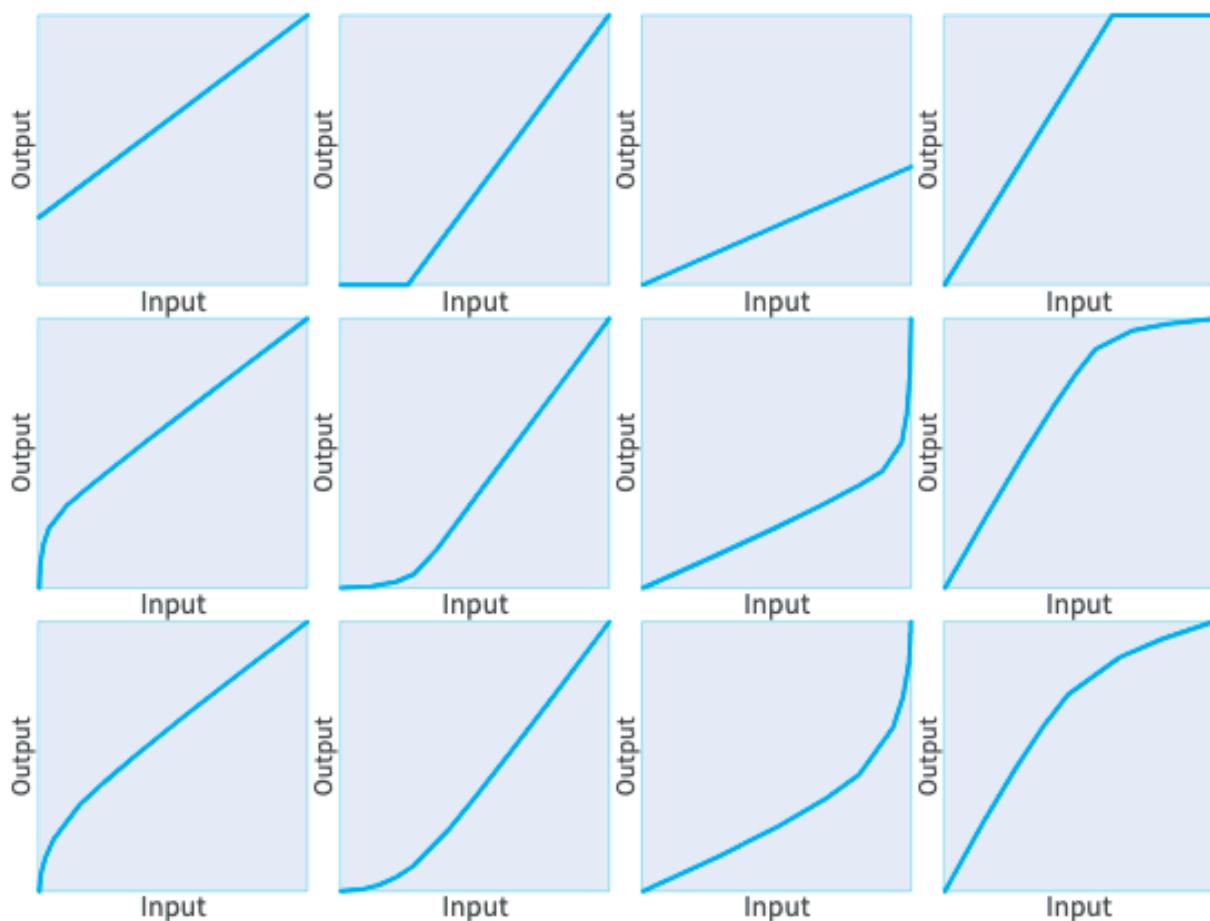
LUT

Les « Look Up Table » (LUT) sont des tableaux de 256 valeurs qui associent à chaque niveau de gris d'une image un nouveau niveau de gris. Ces LUT sont souvent représentées sous formes de graph, en abscisse on retrouve les valeurs originales des pixels et en ordonnées les nouvelles valeurs. Par exemple, la LUT suivante :



https://documentation.euresys.com/Products/Coaxlink/Coaxlink_10_4/Content/Resources/Drawings/coaxlink/lut-parameters/lut-parameters-brightness.svg

remplace les valeurs de niveaux de gris par leur même valeur et ne provoque sur l'image aucune différence. Les LUT prendront donc des formes différentes et plus complexes :



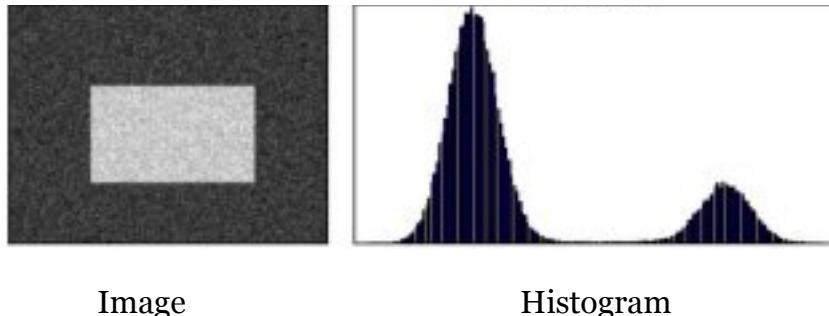
https://documentation.euresys.com/Products/Coaxlink/Coaxlink_10_4/Content/Resources/Drawings/coaxlink/lut-parameters/lut-parameters-visibility.svg

Exercices :

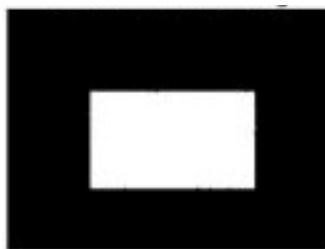
- Ecrivez une fonction qui applique une LUT simple (une droite, par exemple) sur une image
- Trouvez dans le syllabus ou la littérature différentes formes plus complexes, appliquez les, observez et commentez les résultats
- Dessinez une LUT qui, dans une image, diminue le nombre de niveaux de gris (128, 96, 64, 48, 32, 16, ...), déterminez ensuite à partir de quelle valeur (approximative) l'oeil commence à apercevoir des modifications (faux contours), réfléchissez à quoi correspond cette valeur
- Réalisez une LUT qui réduit le nombre de niveaux de gris à 2, commentez le résultat

Seuillage adaptatif

La binarisation automatique (ou seuillage adaptatif) consiste à trouver automatiquement la meilleure valeur du seuil pour dissocier efficacement les informations contenues dans une image en niveau de gris. Par exemple, si nous avons un histogramme qui ressemble à ceci :



on peut imaginer qu'on a deux parties dans l'image, il suffit donc de choisir le seuil entre les deux pics pour obtenir une image binaire qui dissocie ces deux parties :



<https://docs.opencv.org/4.5.1/otsu.jpg>

Exercices :

- Ecrivez une fonction qui évalue le meilleur point de seuillage et appliquer ce seuil dans une binarisation, analyser et commentez le résultat
- Lisez la littérature sur « Otsu’s Thresholding » (par exemple, <https://learnopencv.com/otsu-thresholding-with-opencv/>), cherchez les fonctions en Python qui implémentent cette méthode et utilisez-là dans un programme pour comparer les résultats avec ceux obtenus avec votre fonction de l'exercice précédent

Chapitre 6

Opérations entre images

On peut appliquer des opérations arithmétiques de base sur les images, en effet on va par exemple sommer deux images pour les « fusionner » ou soustraire une image d'une autre pour chercher des différences.



<https://www.cnp-polaris.be/uploads/images/layout/64938275.jpg>

Somme

La somme entre deux ou plusieurs images est un traitement de base, très simple à programmer. On l'applique souvent pour superposer des informations issues de plusieurs images afin de les combiner dans une seule. On peut, par exemple, superposer des données cartographiques avec des données météo satellitaires ou encore des données médicales venant de plusieurs appareils de mesure. Il faut évidemment partir d'images qui sont codées de la même manière (*grayscale*, par exemple) et qui ont la même résolution. Cependant parfois on « ajoutera » une image dans une autre plus grande. Attention si on se contente de faire la somme on va dépasser les valeurs autorisées (un niveau de gris de 138 additionné à un autre de 230, donne un résultat largement supérieur à 255). Le plus souvent on fera donc une moyenne (éventuellement pondérée) des pixels de chacune des images.

Exercices :

- Ecrivez une fonction qui somme de manière pondérée (paramètre de la fonction) deux images, analyser et commentez le résultat
- Imaginez une application à cette fonction, autre que celles décrites dans ce document

Difference

La différence entre deux images est souvent utilisée pour détecter des modifications entre « frames » d'une vidéo, ou encore pour appliquer des « masques ». Comme pour la somme, il faut partir d'images qui sont codées de la même manière (*grayscale*, par exemple) et qui ont la même résolution mais parfois, on « soustraira » une image d'une autre plus grande. Attention, si on retire d'une valeur, un niveau de gris de plus élevé, le résultat sera négatif, il faudra donc prévoir ces cas particulier.

Projet :

Ce cas d'utilisation propose de détecter une intrusion dans le champ d'une caméra. On peut imaginer une caméra de surveillance (pour l'exercice, vous pouvez utiliser votre Webcam) qui « surveille » une scène et qui détecte automatiquement un changement brusque dans le flux vidéo. L'alarme doit donc se déclencher si une personne arrive dans la scène mais ne doit pas provoquer d'alerte si la luminosité globale de l'image change (par exemple, à cause d'un nuage qui passe devant le soleil).

Chapitre 7

Opération de voisinage

Dans ces traitements, nous allons réaliser une opération sur un ensemble de pixels voisins les uns des autres (opération de voisinage). Ils sont généralement appliqués à des images en niveau de gris. Nous allons étudier un grand nombre de types de filtres linéaires (convolution) et d'autres, moins nombreux, non linéaires.



Convolution

La convolution s'applique sur une image en utilisant des masques qui sont des tableaux à deux dimensions d'ordre impair (en général 3×3 ou 5×5). Ils sont utilisés pour réaliser une opération linéaire sur le pixel traité et ses voisins afin de calculer une nouvelle valeur. Il existe un grand nombre de ces masques, définis dans la littérature, destinés à des traitements différents : mise en évidence des contours, lissage de l'image, élimination du bruit, ...

Exercices :

- En vous référant au syllabus, écrivez une fonction qui permet d'appliquer un masque 3×3 à une image, visualisez le résultats
- Posez-vous la question de quoi faire avec les pixels de la bordure de l'image, proposez des solutions
- Essayez les différents masques que vous trouvez dans le syllabus et la littérature, observez leurs effets et commentez
- Etendez votre fonction à des masques de 5×5 et 7×7 , mesurez les différences de performance des traitements quand on augmente la taille du masque

Filtres non linéaires

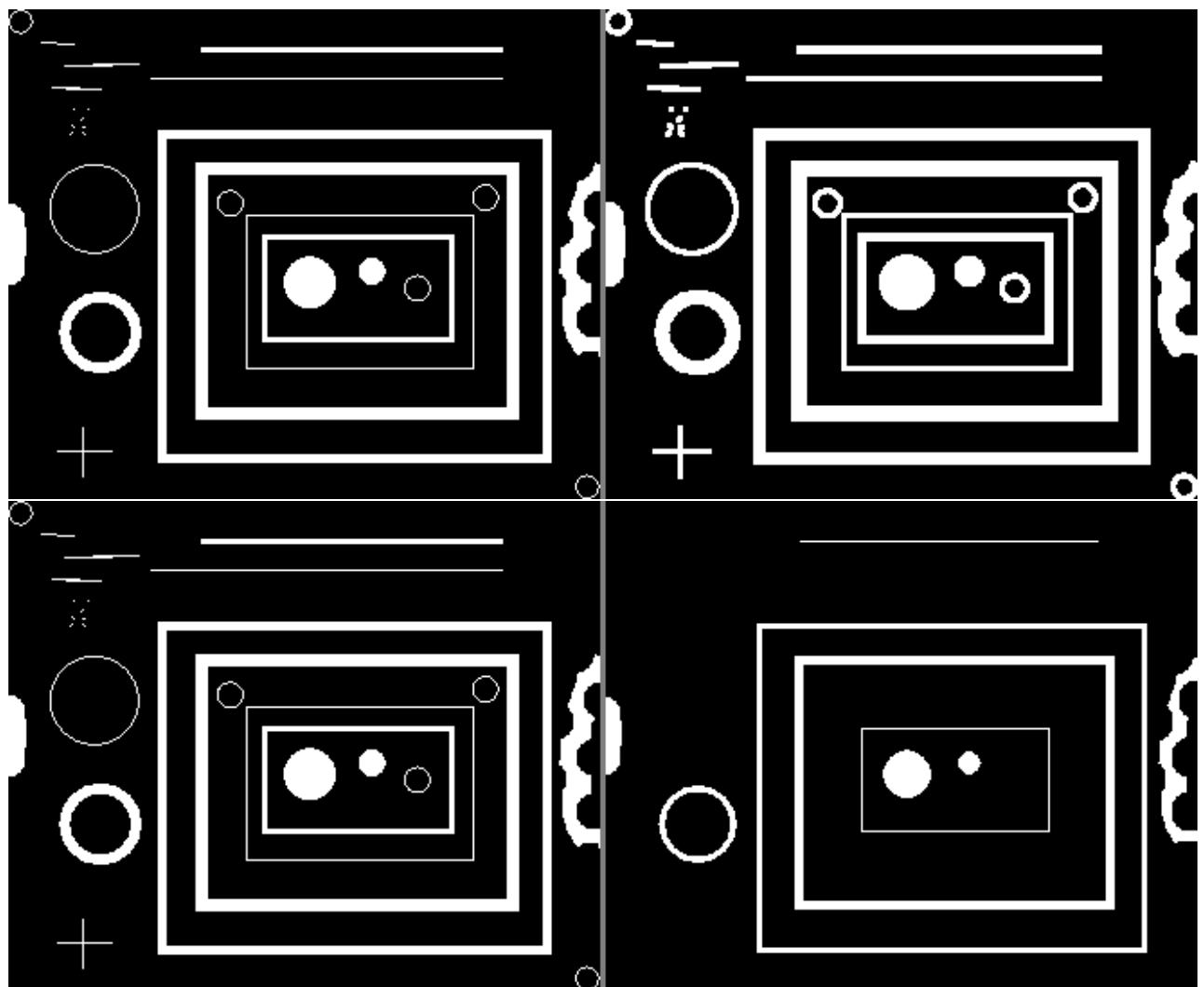
Il existe un autre type de filtre qui utilise également les techniques de masques mais qui réalise une opération non-linéaire. On trouve essentiellement le « *Local Binary Pattern* » (*LBP*) et le filtre médian (*Median Filter*), ils sont expliqués dans le syllabus.

Exercices :

- Ecrivez une fonction qui réalise l'opération décrite pour le filtre médian et imaginez dans quel contexte il pourrait être intéressant de l'appliquer
- Réalisez une fonction *LBP* pour détecter les coins des cases d'une grille
- Imaginez une autre application au filtre LBP, implémentez-là et commentez

Opérations morphologiques

Les opérations morphologiques sont des traitement appliqués sur des images binaires, elles agissent sur les pixels et ses voisins afin de renforcer ou « émincer » des contours ou des formes. Les deux opérations les plus utilisées sont l'érosion et la dilatation mais nous verrons aussi l'ouverture et la fermeture.



Dilatation

La dilatation est appliquée sur une image en noir et blanc pour « élargir » une forme ou un contour déjà mis en évidence généralement par une convolution. La dilatation peut être effectuée grâce à des masques dont les valeurs influenceront le type de dilation désiré.

Exercices :

- Ecrivez une fonction qui permet de dilater une image, appliquez-là sur différentes images, notez vos observations
- Tester plusieurs valeur de masques et commentez
- Essayez d'appliquer plusieurs fois d'affilée une dilatation sur une image, observez le résultat

Erosion

L'érosion est le traitement inverse de la dilatation, il va « désépaissir » une forme ou un contour, elle s'applique également sur des images en noir et blanc. L'érosion est calculée à l'aide de masques différents.

Exercices :

- Ecrivez une fonction qui permet d'éroder une image, appliquez-là sur différentes images, notez vos observations
- Tester plusieurs valeur de masques et commentez
- Essayez d'appliquer plusieurs fois d'affilée l'érosion sur une image, observez le résultat
-

Ouverture

L'ouverture consiste à réaliser une dilatation suivie d'une érosion, cet enchaînement de traitements va par exemple permettre de mettre en évidence les contour d'une forme « pleine ».

Exercice :

- Combinez l'utilisation de la dilatation (plusieurs passes éventuelles) suivi d'une (ou plusieurs) érosions afin de réaliser une ouverture, observez l'intérêt d'une telle combinaison

Fermeture

La combinaison inverse qui consiste à réaliser d'abord une érosion et ensuite une dilatation s'appelle une fermeture et permet, par exemple de « colorier » l'intérieur d'un contour pour ensuite labelliser (voir chapitres suivants) une image en noir et blanc.

Exercice :

- Combinez l'utilisation de l'érosion (plusieurs passes éventuelles) suivi d'une (ou plusieurs) dilatation afin de réaliser une fermeture, observez l'intérêt d'une telle combinaison

Chapitre 9

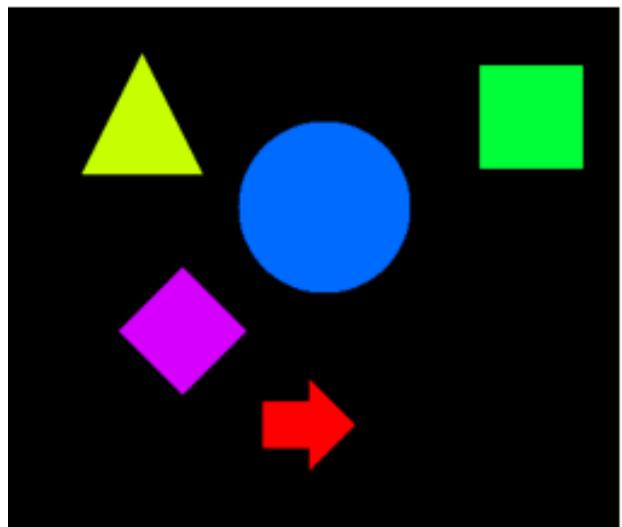
Labellisation

Le « *labelling* » consiste à retrouver dans une image (souvent en noir et blanc) les pixels connectés en eux, formant un tout qui pourrait être un sujet d'intérêt dans le cadre d'une reconnaissance d'image. C'est souvent une étape préalable à la reconnaissance d'objets. En effet, une fois les formes isolées on peut alors les « faire passer » dans des algorithmes d'intelligence artificielle qui vont tenter de reconnaître ces objets (*Object Detection*) ou ces caractères (*Optical Character Recognition*).

0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	
1	1	1	0	0	1	0	
1	1	0	0	0	1	0	
0	0	0	1	0	1	0	
0	0	1	1	0	0	0	
0	0	0	0	0	1	1	

0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	
1	1	1	0	0	2	0	
1	1	0	0	0	2	0	
0	0	0	3	0	2	0	
0	0	3	3	0	0	0	
0	0	0	0	0	4	4	

<http://k-science.blogspot.com/2017/06/object-counting-using-connected.html>



<https://iq.opengenus.org/connected-component-labeling/>

La labellisation (*labelling*) consiste à retrouver dans une image (souvent en noir et blanc) les pixels connectés entre eux, formant un tout et qui pourrait donc être un sujet d'intérêt dans le cadre d'une reconnaissance d'image.

Exercices :

- Labellisez dans l'image « *formes_geometriques.gif* » les différentes formes et sauvegardez-les automatiquement dans des fichiers séparés.
- En vous aidant du site https://en.wikipedia.org/wiki/Connected-component_labeling, dans une grille de « *Mot Mélés* » numérisée (*mots_meles.jpg*), faites disparaître la grille et donnez un label à chacune des lettres.

Projet :

Isolez dans l'image « *Planimeter.png* » les différentes cellules et calculez leur surface, compte tenu de l'étaillon carré en bas de page.

Chapitre 10

Détection d'objets et OCR

Les techniques « anciennes » de détection d'objets consiste à calculer la corrélation entre un pattern et chaque portion d'une image. Actuellement, l'intelligence artificielle remplace efficacement ces techniques de corrélation pour permettre en plus de la détection, la reconnaissance de multiples objets.



Correlation

Une technique de base pour la recherche d'objets consiste à « passer » sur l'image un masque qui représente le *pattern* à retrouver et de calculer pour chacun des pixels (et ses voisins) le niveau de corrélation entre cette portion d'image et le masque donné. Le résultat de cette corrélation est représenté par un tableau (de la taille de l'image) dans lequel les valeurs les plus faibles représenteront les endroits les plus « probables » pour la position de l'objet recherché.

Exercice :

- Appliquez les formules du syllabus pour écrire une fonction qui recherche la meilleure corrélation entre un objet et une image

Projets :

Ecrivez un programme qui recherche Waldo (voir exemple page précédente) dans une image et affichez sur cette image des petits cercles qui indiquent les positions les plus probables de l'individu.

Intelligence artificielle

Vu l'ampleur du sujet ces techniques ne seront pas abordées dans ce cours, d'autant plus, que d'autres cours spécifiques y sont consacrés. Cependant, il existe dans *OpenCV* des méthodes pour détecter rapidement des « blobs » et des bibliothèques Python qui permettent de reconnaître simplement des caractères. Deux projets vous sont donc proposés car ils ne nécessitent aucune connaissance approfondie dans les algorithmes d'intelligence artificielle.

Projets :

En vous aidant de la bibliothèque *Tesseract* pour *Python* (<https://pypi.org/project/pytesseract/>) et de différents sites (par exemple, <https://nanonets.com/blog/ocr-with-tesseract/>), réalisez un programme qui à partir d'un scan de document (simple, sans paragraphe) reconnaît les caractères et sauvegarder l'ensemble dans un fichier texte.

Réalisez un logiciel qui, à partir d'une grille numérisée de *Sudoku*, trouve et affiche la solution automatiquement.

<https://fr.eggs-iting.com/blog/prototypage/detection-des-oeufs-dans-un-nid-de-poule.html>

Bibliographie

Livres

Liens Internet

<http://webia.lip6.fr/~thomen/Teaching/BIMA/cours/amelioration.pdf>

https://www.kongakura.fr/article/OpenCV_Python_Tutoriel

Videos

<https://www.youtube.com/watch?v=oXlwWbU8l2o>

<https://www.youtube.com/watch?v=WQeoO7MioBs>