Analyse d'images

Compte rendu de TP n°1, 2 et 3

Mai 2017

Groupe [1]

[21506752] –Robin DERVIEUX]

[21506704] – [Myriam LE LIBOUX]

TP1 - Analyse d’image

Prise en main de ImageJ

Introduction :

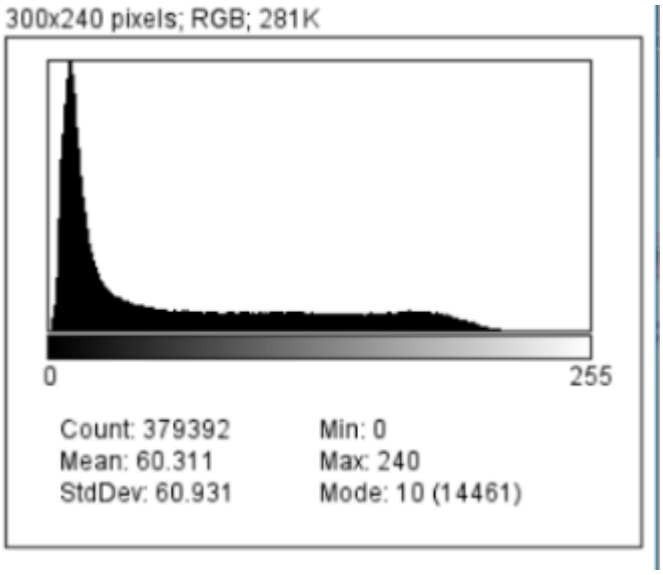
Le but de ce TP était de nous permettre de prendre en main le logiciel Imagej et de découvrir les différents procédés que ce logiciel propose. Dans un premier temps nous nous sommes intéressé à l’analyse d’'histogrammes de niveaux de gris avant et après différents traitements. Nous avons ensuite travaillé sur différents filtres de convolution tels que le filtre Gaussien, moyenneur ou médian.

Question 1 :

ImageJ permet de conversion d’image couleur en image en niveau de gris. Le logiciel permet de choisir la taille d’encodage de l’image.
Il est possible d’avoir son image en 8, 16 ou 32 bits. Plus l’encodage est grand, plus l’image aura de niveau de gris possible et ainsi une meilleure qualité.

Question 2 :

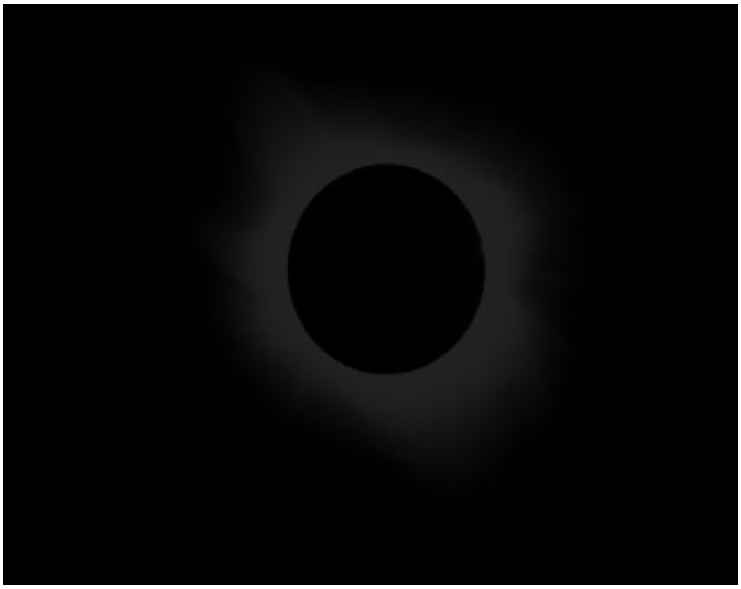
Après avoir converti l’image “lisa.png” on obtient l'histogramme suivant :

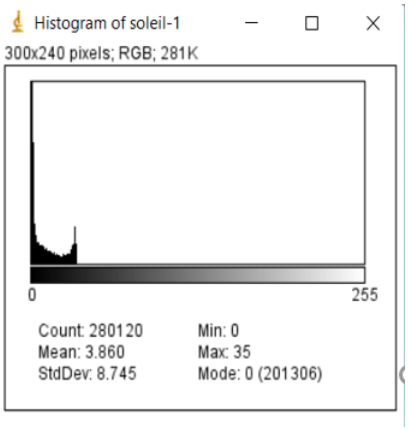


Le pic proche de zéro s’explique par le fait que l’image soit initialement sombre (Le pic est proche de zéro) et faiblement contrastée (le pic est haut). L’histogramme n’a plus de valeurs après 240 car l’image initiale ne contient pas d’éléments blanc ou très clairs.

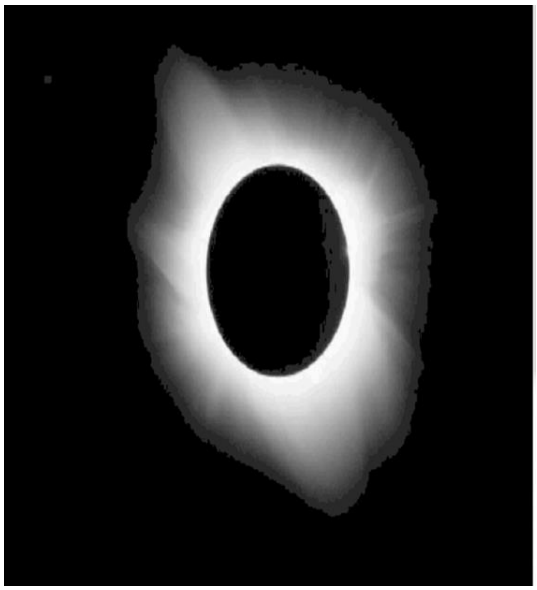
Question 3 :

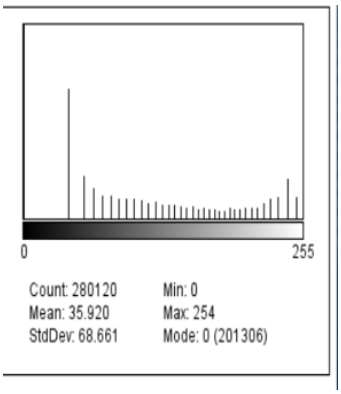
Lors de cette question nous avons égaliser les histogrammes de certaines images pour que le contraste soit compris dans la fourchette 0-255.



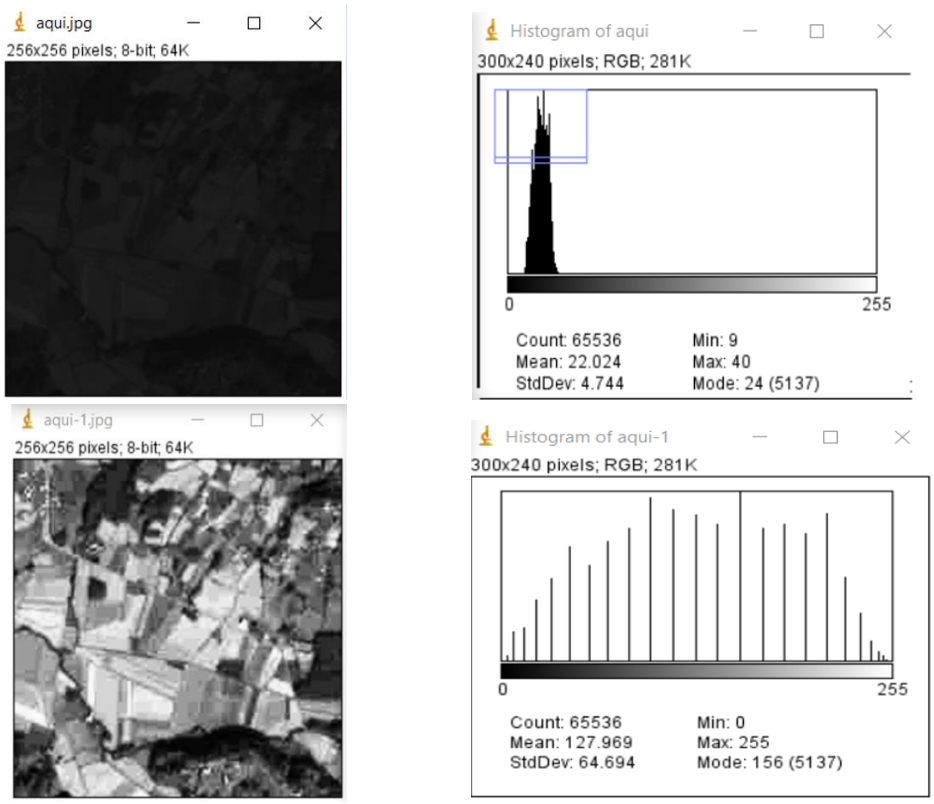


Soleil avant égalisation de l'histogramme





Soleil après égalisation de l'histogramme



Égalisation de l'histogramme

Comme on peut le voir, l'égalisation de l'histogramme permet de répartir l'histogramme sur toute l'échelle. Cela a pour effet d'augmenter le contraste de l'image.

Question 4 :

Le soleil est l'image qui semble le plus avoir changé car initialement son histogramme est l'histogramme le plus petit puisque celui-ci s'arrête à la valeur 35. Ainsi l'égalisation va permettre de répartir plus équitablement les valeurs dans l'histogramme. Cela a pour conséquence d'éclaircir l'image.

Question 5 :

Filtre moyenneur 5x5

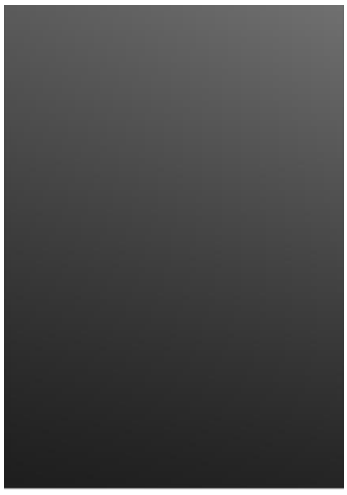
Filtre moyenneur 9x9

Filtre moyenneur 12x12

On peut remarquer que plus le rayon du filtre est important plus l'image est floutée. En effet le rayon représentant le nombre de voisins à considérer, il est normal que plus le nombre de voisins est grand plus l'image est floutée.

Question 6 :

Si l'on applique un filtre moyenneur ayant comme rayon la dimension de l'image on obtient l'image suivante :



Il s'agit d'un dégradé de gris ou le bas de l'image est plus foncé que le haut. Cela s'explique par le fait que le bas de l'image initiale soit en moyenne plus foncée que le haut. La

couleur du pixel situé au centre de cette image est la couleur moyenne de tous les autres pixels.

Question 7 :

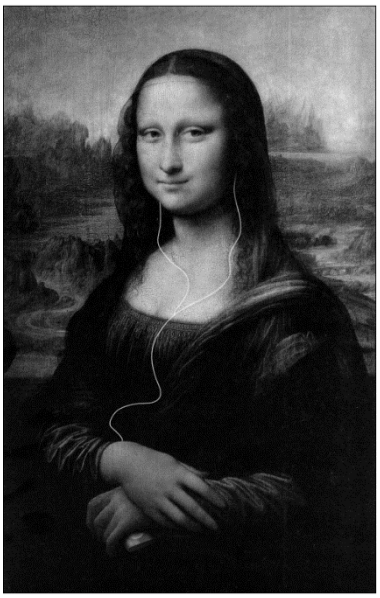


Image avec 0 convolution

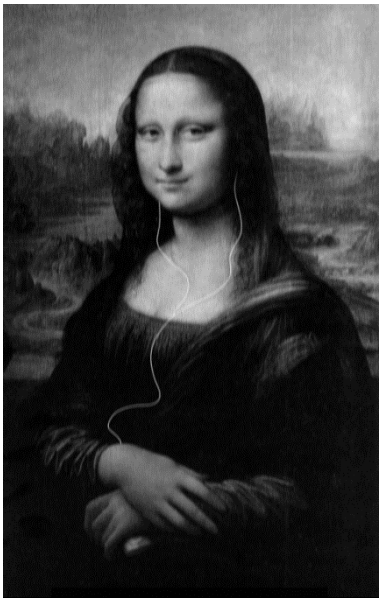


Image avec 1 convolution

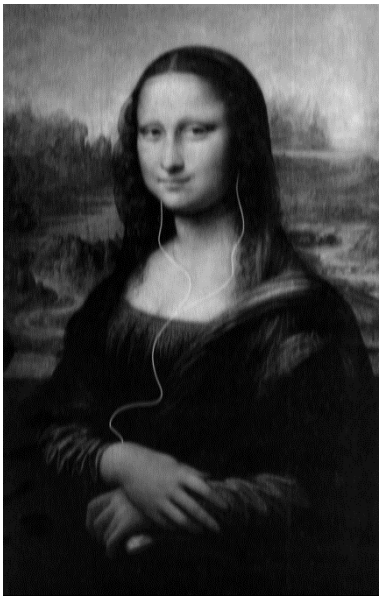
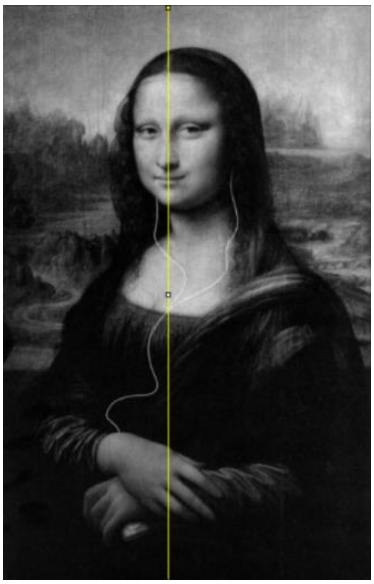
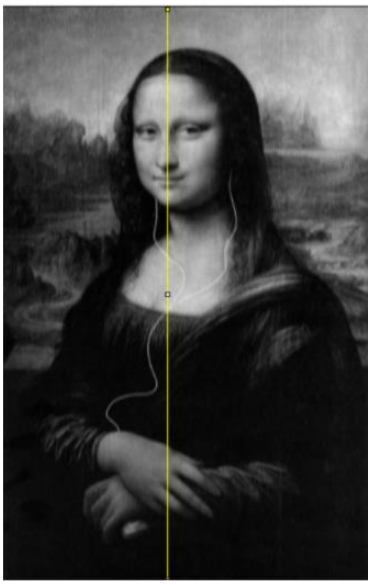
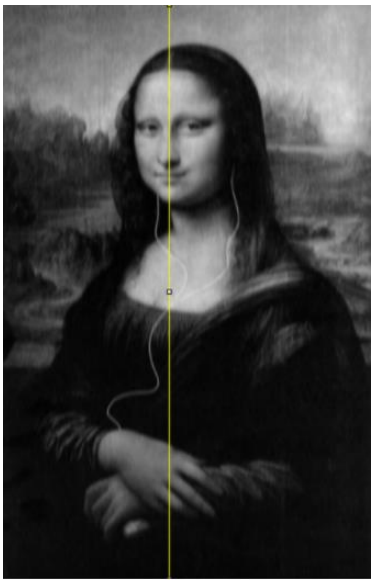
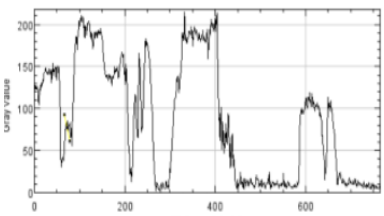
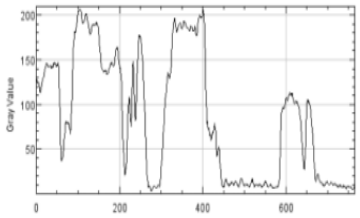


Image avec 2 convolutions

On peut remarquer que plus on réalise de convolutions plus l'image est floue. En effet ce filtre moyenneur qui est un filtre dit passe-bas ou de lissage va calculer la couleur d'un pixel selon la couleur de ces plus proche voisins (ici 5x5). Les contours de l'image sont donc plus difficiles à distinguer. Le filtre appliqué sur cette image est le même filtre que celui qui a été appliqué dans la question 5 avec un rayon de 5.

Question 8 :



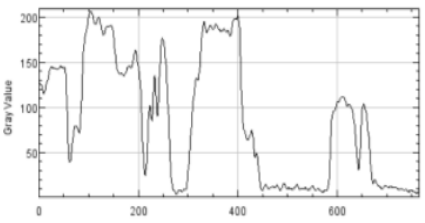


Image de base | 1ere convolution | 2eme convolution

Comme dit précédemment, les convolutions ont eu pour effet de lisser l'image avec un filtre passe-bas. Cela peut se voir sur l'histogramme, toutes les hautes fréquences de l'images disparaissent au fur et à mesure après chaque convolution. Le bruit sur l'histogramme s'estompe.

Question 9 :

Pour cette question nous avons volontairement dégradé une image avec 3 bruit différent : un bruit modéré (25), bruit fort (42.5), et un bruit de type Salt & Pepper. Puis nous avons essayé de restaurer cette image le mieux possible à l'aide de filtre pour retrouver l'original.

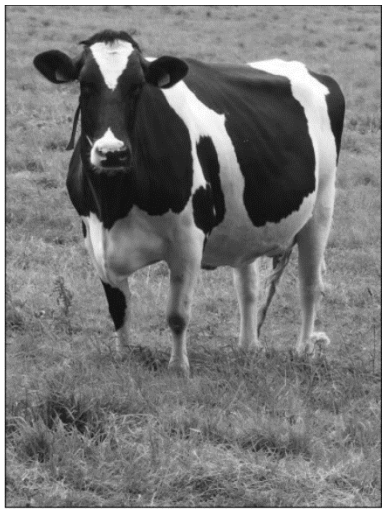
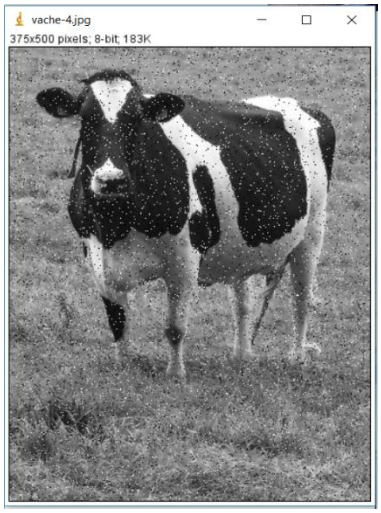
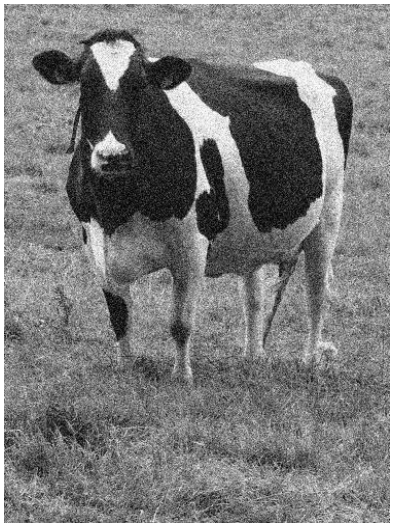


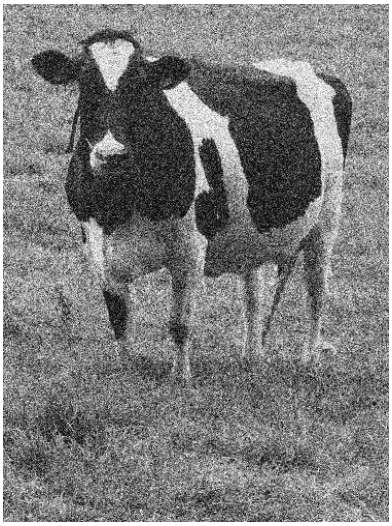
Image de base



Salt & Pepper



Bruit (25)



Bruit (42.5)

Pour la restauration du bruit des deux intensité (25 et 42.5), les résultats des filtres sont similaires : le filtre moyenneur fonctionne mieux quand on prend un rayon du filtre inférieur à 5, au-delà, le filtre médian rend un résultat plus proche, le filtre moyenneur devenant trop flou.

Pour le filtre Salt & Pepper, le filtre moyenneur conserve mieux les détails et reste plus fidèle à l'image de base, cependant le filtre médian fait mieux ressortir les contour et restore mieux le contraste.

Question 10 :

L'un des principaux problèmes de ces méthodes de filtrage par convolution est que bien qu'elles réduisent le bruit elles vont aussi beaucoup flouter l'image et ses contours et rendre ainsi la distinction entre les différents objets relativement compliquée. Une autre des difficultés est de déterminer le rayon des filtres et de choisir quel filtre utiliser.

Conclusion :

Ce TP nous a permis de nous familiariser avec différents filtres passe-bas et méthodes de convolution. Nous avions pris conscience de la complexité à choisir entre les différents filtres, dans certaines situations ces filtres donnant le même résultat. Nous avons aussi pu mettre en œuvre directement les différentes notions apprises en cours concernant l'analyse des histogrammes.

TP2 - Analyse d’image

Extraction de régions & Détection de contours

Introduction :

Le but de ce TP est de se familiariser avec la notion de détection de contours et l’extraction de régions et de comprendre comment celles-ci fonctionnent. Nous nous intéresserons dans un premier temps à comment sélectionner différentes régions d’une images (les régions différentes d’une image à une autre ou seulement les parties verticales ou horizontales). Dans un second temps nous nous intéressons à la détection de contours et comment le bruit peut influencer les méthodes utilisées. Finalement nous essaierons de mettre en œuvre cette détection de contours afin de compter un certain nombre d’éléments présents dans une image.

Question 1 :

Pour mettre en évidence les différences entre les deux images, nous avons appliqué un “ou exclusif” (XOR) entre les pixels de deux images.

b	a	X
0	0	0
0	1	1
1	0	1
1	1	0

Le ou exclusif mettant à 0 les pixels identiques,
et laissant à 1 les pixels différents, la nouvelle image générée est donc l’image des différences.

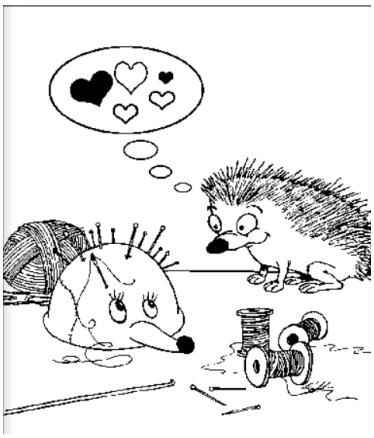


Image 1

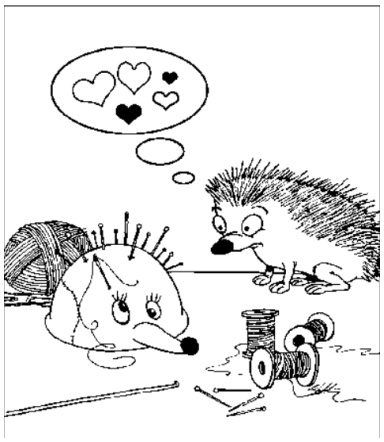


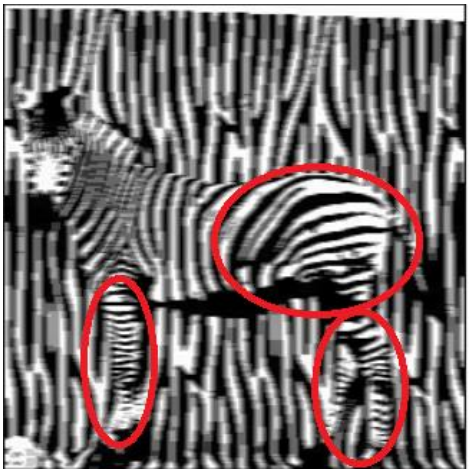
Image 2

Différences

Lorsque les images sont décalées, il pourrait être intéressant de mettre en place un algorithme permettant de calculer le décalage entre les deux images (en analysant un certain nombre de pixels), de décaler l'image et de chercher les différences. L'algorithme proposé est le suivant :

```
Image1 : int[][]
Image2 : int [][]
Image2Decalee : int [][]
x : int //décalage de l'image en x
y : int // décalage de l'image en y
rayon  : int//pour l'analyse des plus proches voisins
Pour i allant de 0 à un nombre de pixel faire
      Pour j allant de 0 à un nombre de pixel faire
            Pour h allant de 0 à offset
            a:int
            b:int
                        Si i+h<longueur de l'image AND b+h<hauteur
      de l'image
                              a=i+h
                              b=j+h
                              Si Image[i][j]==Image2[a][j]
                                    Alors x = (x+ (a-i))/2
                              Sinon Si Image[i][j]==Image2[i][b]
                                    Alors y = (y+ (b-i)) /2
                              Sinon Si Image[i][j]==Image2[a][b]
                                    Alors y = (y+ (b-i)) /2
                                    x = (x+ (a-i))/2
                              Fin si
                        Fin si
                  Fin Pour
            Fin Pour
      Pour i allant de 0 à un nombre de pixel faire
            Pour j allant de 0 à un nombre de pixel faire
                  Si i+x<longueur de l'image AND j+y < hauteur de
      l'image
                        Image2Decalee[i][j]=Image2[i+x][j+y]
                  Fin Si
            Fin pour
Fin pour
Difference : int[][]
Difference= Image1 XOR Image2Décalee
```

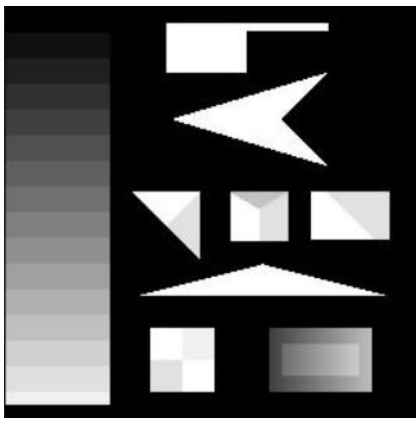
Question 2 :



```
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
150 150 150 150 150
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
```

Masque de convolution

A l'aide de ce masque, nous avons rendu tous les traits verticaux flous, alors que les traits horizontaux, eux, restes inchangés.
Nous avons appliqué la même méthode pour les traits verticaux.



```
0 0 150 0 0
0 0 150 0 0
0 0 150 0 0
0 0 150 0 0
0 0 150 0 0
```

Question 3 :

a. Le filtre de Deriche est assez similaire au filtre Gaussien, mais moins complexe à implémenter (seulement 2 multiplication par pixel). Cet algorithme a pour but de détecter les contours. Le filtre a pour équation *y[n]= a*x[n]+(1-a)*y[n-1]*. Ce filtre renvoie deux images, la première représente les contours détectés via des calculs avec des valeurs normées et la deuxième représente la détection des contours avec suppression des non-maxima locaux (la

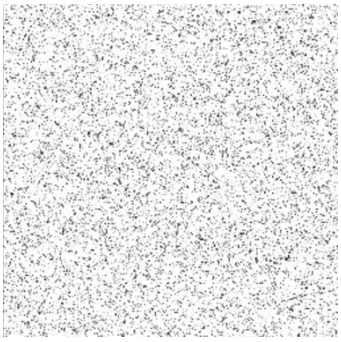
norme du gradient pour les pixels qui ne sont pas des maxima locaux est mise à 0). Cette deuxième méthode va nous permettre d'obtenir des contours ayant comme épaisseur 1 pixel.

b.

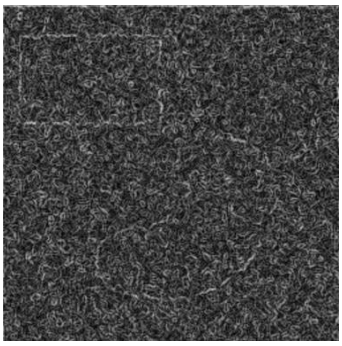
	Sobel (Find Edges)	Deriche
Aqui		X (suppr)
Cellules		X (suppr)
Delphin		X (suppr)
Fiels		X (suppr)
Noise		X (suppr)
Pisarro		X (suppr)

Nous avons remarqué que pour toutes les images demandées le filtre de Sobel est le plus mauvais. En effet celui-ci ne traite pas bien le bruit présent sur les images, la seule image ou la détection était quasi-identique est pour l'image "cellules". L'image "suppr" de la méthode de Deriche est toujours plus précise que la "norm" car celle-ci fournit un contour d'un pixel contrairement à l'autre qui peut parfois fournir des contours relativement épais.

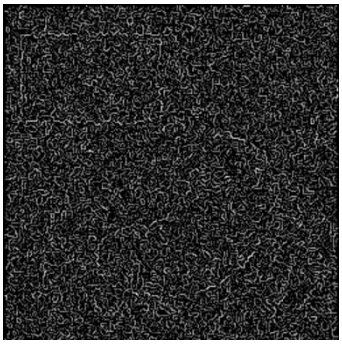
c.



Sobel

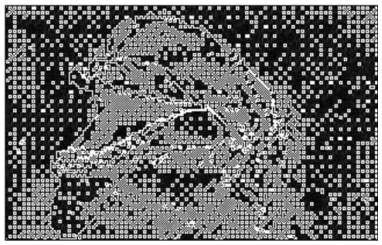


Deriche (norm)

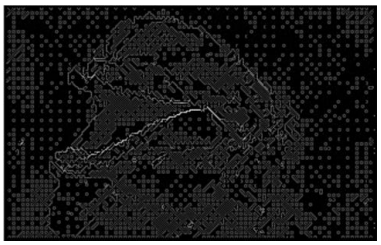


Deriche (Suppr)

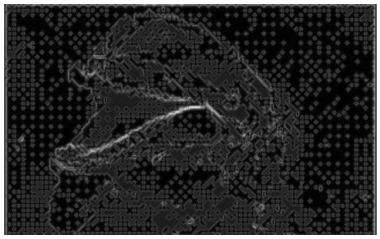
L'image "Noise" contient énormément de bruit ainsi lorsque le logiciel tente de détecter les contours il interprète le bruit comme des zones dont il détermine le contour. Toutes ces zones empêchent de détecter le contour des "vrais" objets. La méthode de Sobel n'arrivant pas à supprimer un minimum rend la distinction des contours des objets quasi-impossible. Pour la méthode de Deriche les contours sont visible mais pas nettement.



Sobel



Deriche (norm)



Deriche (Suppr)

Comme pour l'image "Noise" l'image Dolphin contient énormément de bruit. Ce bruit est détecté par le logiciel ce qui explique la présence des petits carrées autour du dauphin. Ces carrés empêchent la détection des contours spécialement pour le cou du dauphin.

L'image "Aqui" étant très sombre initialement la détection de contours était plus compliquée, les couleurs étant très proche les unes des autres. Cependant les trois méthodes fournissent un résultat que l'on pourrait considérer comme acceptable. Encore que les contours affichés par le filtre de Sobel sont assez compliqués à discerner.

Sobel

Deriche (norm)

Deriche (suppr)

Question 4 :

En utilisant "Analyse Particule" qui a pour but de compter le nombre de particules/ objets présents sur une image nous avons en effet constater que le logiciel permet d'en dénombrer 329. Nous nous sommes aussi rendu compte qu'en érodant ou dilatant les images le nombre de particules variaient. En effet en érodant la taille des particules diminuent, ce qui permet de séparer les particules qui peuvent être "soudées", on en compte alors 342. En dilatant l'image on augmente la taille des particules ce qui fait se rejoindre des particules distinctes. Le logiciel distingue donc moins de particules, il n'y en a plus que 247

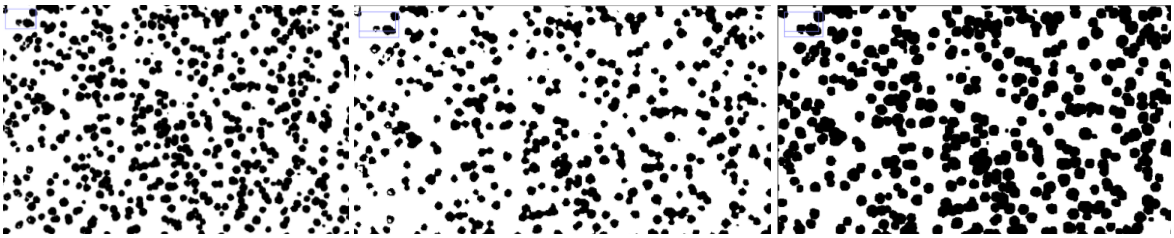
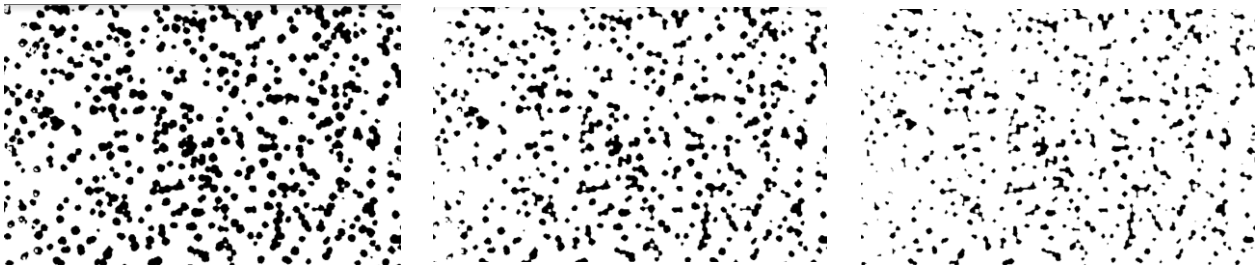


Image Binarisée Image érodée Image dilatée

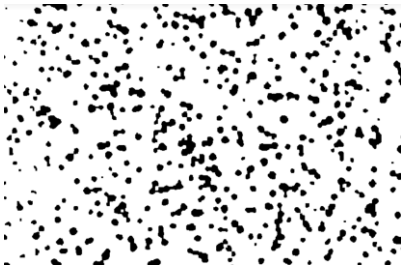
Question 5 :

```
cellules.png   329
cellules.png   342
cellules.png   361
cellules.png   379
cellules.png   369
```

Nombre de cellules durant les différentes étapes (la première étant situé en haut)



Erosion 1 Erosion 2 Erosion 3



Dilatation 1

Dans cette question, nous avons essayé d'améliorer le comptage des cellules dans une images. De base, certaines cellules étaient collées et faussaient la fonction de comptage. C'est pour cette raison que nous obtenons 329 cellules lors de la première itération.

Pour pallier à ce problème, nous appliquons successivement des actions d'érosion afin de mieux différencier chaque cellule. Après 3 érosions, nous appliquons une dilatation pour combler les défauts créés lors des dilatations. Nous obtenons alors le bon compte de cellules.

Conclusion :

Ce TP a permis de mettre en évidence les problèmes de détection des éléments dans une ou entre deux images, ainsi que de montrer les limites de détection liées à l'image d'origine. Il nous a aussi permis de découvrir comment améliorer une image en permettant de distinguer plus nettement différentes formes.

TP3 - Analyse d'images

Création de Plugins sous ImageJ

Introduction :

Le but de ce TP est de nous permettre de découvrir l'utilisation de plugins sous ImageJ. Codés en Java, ces plugins peuvent permettre de réaliser des actions non implémentées dans ImageJ ou encore d'en utiliser plusieurs à la suite les unes des autres. Dans un premier temps nous nous intéresserons à la compréhension du code de ces plugins avant d'en modifier certaines fonctionnalités, puis finalement créer notre propre plugin.

Question 2 :

Le tableau nommé pixel aurait la même taille que l'image contient de pixels. Les pixels seront stockés à la suite tels que l'explique le schéma suivant :

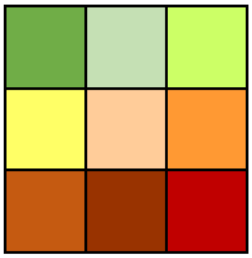


Tableau "pixel"

Image initiale

Chaque cellule de ce tableau contiendra la couleur du pixel correspondante dans l'image initiale.

Pour accéder à un pixel situé sur la ligne y à la position x on utilisera la formule :

*ndg[x][y]= pixel[x+y*width].*

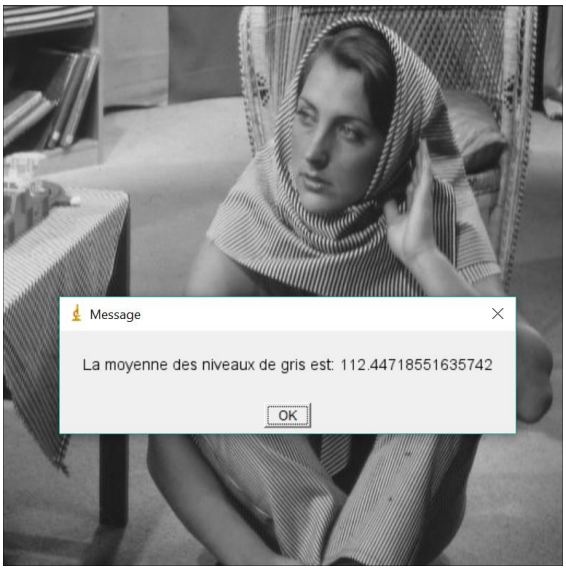
Le plugin proposé réalise un traitement sur chaque pixel. En effet il binarise l'image, si la couleur d'un pixel est supérieure à 120 ce pixel devient blanc, sinon il devient noir.

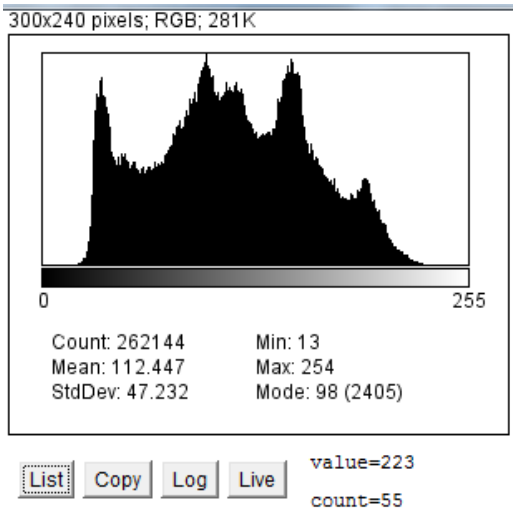
Question 3 :

Comme dit précédemment on peut remarquer que le script transforme une image en nuance de gris codée sur 8 bits en une image binaire.

Question 4 :

Afin de pouvoir calculer la moyenne des niveaux de gris nous avons dû réaliser quelques modifications dans le code. Nous avons ajouté un double, additionnant les couleurs de tous les pixels. Une fois l'image parcourue entièrement cette somme est divisée par le nombre de pixel afin d'obtenir la moyenne. Le résultat obtenu est le suivant :





Si l'on compare la moyenne obtenue à celle calculée via l'histogramme on peut remarquer quelles sont identiques.

Question 5 :

Afin de déterminer les ressemblances entre les images nous avons calculés pour chaque image présente dans le dossier, la moyenne de ces pixels. Une fois cela réalisé il ne nous restait plus qu'à comparer ces moyennes à celle de l'image sélectionnée. Cependant cette méthode peut être largement critiquée puisqu'elle ne tient pas compte des différentes formes présentes, du contraste, de la luminosité… Nous avons dans un premier temps travaillé avec des images présentes dans le fichier ainsi l'image la plus proche devait être elle-même puis dans un deuxième temps nous nous somme intéressé à des images non présentes dans "la base". La distance affichée représente, en fait la différence entre moyenne de l'image initiale avec la moyenne de l'image la plus proche (le code sera fourni en annexe).

Image présente dans "la base"

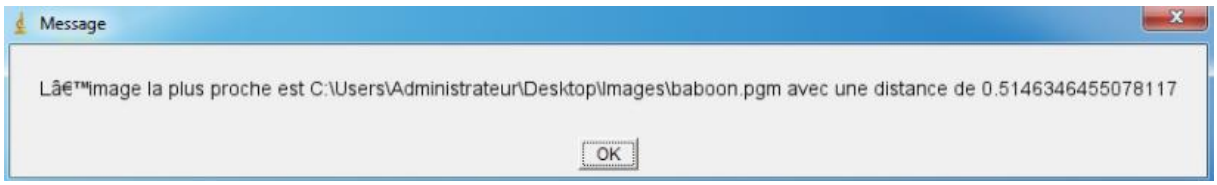


Image non présente dans la "base"

Question 6 :

Pour la réalisation de notre plugin nous avons choisi de réaliser une détection de contour puis de “colorer” la zone détectée en blanc alors que le reste de l’image sera mis en noir. Cette coloration contrairement à la recherche de contours fonctionne à peu près sur des images très simples tels qu’un cercle uni, mais ne produit pas de très bons résultats sur des images beaucoup plus complexes. Le résultat obtenu est le suivant :

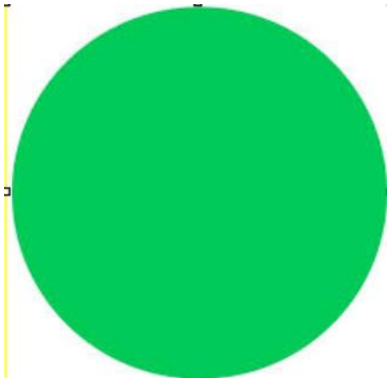


Image initiale

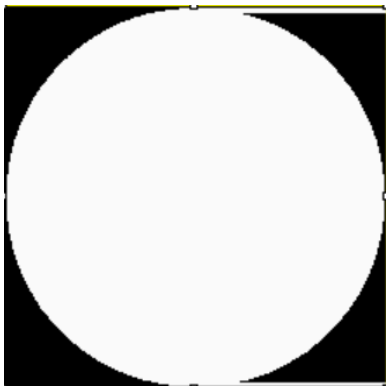


Image après traitement

Conclusion :

Ce TP était très intéressant puisqu’il nous a permis de pouvoir travailler sur la conception de notre propre plugin, le sujet étant libre nous avons pu choisir un procédé qui nous intéressait.

Annexes :

Question 3 :

```
import ij.*;
import ij.process.*;
import ij.plugin.filter.*;

public class Quest3 implements PlugInFilter {

public void run(ImageProcessor ip){
      double moy=0;
      byte [] pixels = ( byte []) ip.getPixels (); // Notez le cast en byte
()

      //Récupere la taille de l'image
      int width = ip.getWidth() ;
      int height = ip.getHeight() ;
      int ndg;
      //pour chaque pixel
      for (int y=0; y< height ; y++)
            for (int x=0; x< width ; x++) { // pas completement optimal
mais pedagogique…
                  ndg = pixels [ y*width + x ] & 0xff ;
                  //Calcul la somme de la couleur de chaque pixel
                  moy=moy+ndg;
            }
      //Calcul la moyenne des couleurs de ces pixels
      moy=moy/(width*height);
      //Affichage de la moyenne
      IJ.showMessage("La moyenne des niveaux de gris est: " +moy);
}

public int setup(String arg, ImagePlus imp){
      if (arg.equals("about")){
            IJ.showMessage("Traitement de l'image");
            return DONE;
      }
      return DOES_8G;
      }
}
```

Question 5 :

```
import ij.*;
import ij.process.*;
import ij.plugin.filter.*;
import java.io.File;

public class FindSimilar implements PlugInFilter{
	public void run ( ImageProcessor ip ) {
		String path ="C:/Users/Administrateur/Desktop/Images";

		File [] files = listFiles(path);
		if ( files.length != 0) {
			//Calcul la moyenne de l'image initiale
			double moyImage=AverageNdg(ip);
			int posImage=0;
			double gap=255;
			double moy=0;
			//Pour chaque image présente dans le fichier
			for (int i =0;i < files.length ;i++)
			{
				// creation d'une image temporaire
				ImagePlus tempImg = new ImagePlus (
files[i].getAbsolutePath() );
				new ImageConverter( tempImg ).convertToGray8 ();
				ImageProcessor ipTemp = tempImg.getProcessor() ;
				//Calcul la moyenne de cette image
				moy=AverageNdg(ipTemp);
				//Si l'image est la plus ressemblante trouvé pour
l'instant
				if(Math.abs(moy-moyImage)< gap)
				{
					//Recupere la position de l'image
					posImage=i;
					//Récupère la distance
					gap=Math.abs(moy-moyImage);
				}
			}
			IJ.showMessage("L image la plus proche est " +
files[posImage].getAbsolutePath( ) + " avec une distance de " + gap );
		}
		else{
			IJ.showMessage("0 images" );
		}
	}
	public File [] listFiles ( String directoryPath ) {
		File [] files = null ;
		File directoryToScan = new File ( directoryPath );
		files = directoryToScan.listFiles();
		return files ;
	}
// Retourne la moyenne des NdG d'une image en NdG
	public double AverageNdg ( ImageProcessor ip ) {
		double moy=0;
		byte [] pixels = ( byte []) ip.getPixels (); // Notez le cast
en byte ()
		int width = ip.getWidth() ;
		int height = ip.getHeight() ;
		int ndg=0;
		for (int y=0; y< height ; y++)
```

```
                  for (int x=0; x< width ; x++) { // pas complètement
optimal mais pedagogique…
                  ndg = pixels [ y*width + x ] & 0xff ;
                  moy=moy+ndg;
                  }
            moy=moy/(width*height);
            return moy;
      }

      public int setup(String arg, ImagePlus imp){
      if (arg.equals("about")){
            IJ.showMessage("Traitement de l'image");
            return DONE;
      }
      return DOES_8G;
      }
}
```