# *Analyse d'images*

May 28, 2018

Thomas COUCHOUD
thomas.couchoud@etu.univ-tours.fr
Victor COLEAU
victor.coleau@etu.univ-tours.fr

# Contents

# Chapter 1

# TP1

## 1.1 Question 1

- Image permet de gérer les caractéristiques propres à l'image en cours (type d'encodage, luminosité, contraste, ...).
- Process permet d'appliquer des opérations sur l'image telles qu'ajouter du bruit, des ombres, ...
- Analyze permet d'acquérir des informations sur l'image dans son état actuel (surface, min/max de couleurs, histogramme, ...).
- Plugins permet d'utiliser des plugins. Certains sont déjà fournis de base.

Afin de convertir une image en niveaux de gris nous utilisons le menu Image > Type puis avons le choix entre:

- 8 bit: Il y aura 256 niveaux de gris.
- 16 bit: Il y aura 65536 niveaux de gris.
- 32 bit: Il y aura 4294967296 niveaux de gris.

Ces valeurs ont un impact sur la manière dont est stockée l'image. Plus on utilise de bits, plus la taille en mémoire de l'image est importante.

! Dans la suite de ce TP, nous utiliserons le niveau de gris 8-bit.

## 1.2 Question 2

### 1.2.1 Zone 1

Nous pouvons observer dans la zone 1 un pic très important. Celui-ci s'explique par une forte présence de couleurs sombres dans l'image originelle transformées en gris sombre (a.k.a. noir). Ces pixels noirs ont une valeur comprise entre 3 et 28.

### 1.2.2 Zone 2

À l'inverse, l'image originelle contient très peu de pixels clairs. Cela est traduit par très peu ($\approx 250$) de gris pixels clairs (a.k.a. blancs).

## 1.3 Question 3

### 1.3.1 mystere.pgm

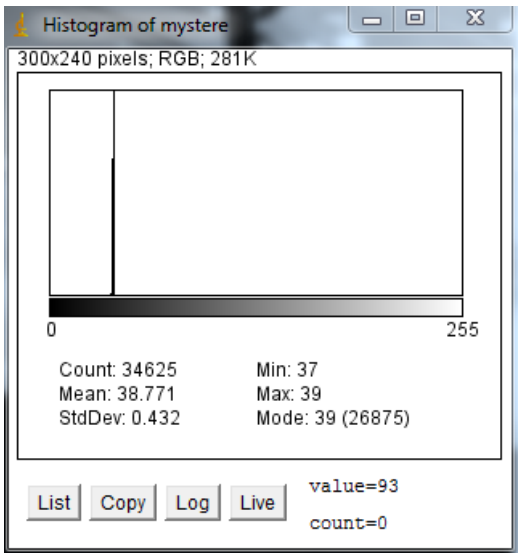


Figure 1.1 – Histogramme avant

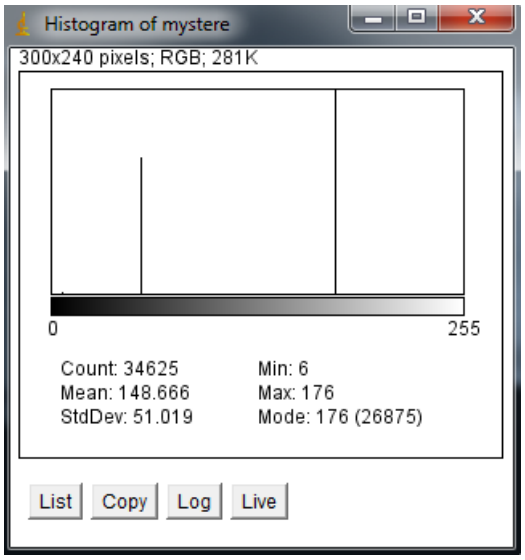


Figure 1.2 – Histogramme après

Figure 1.3 – Histogrammes

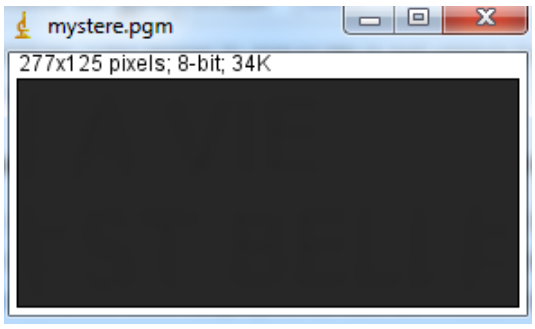


Figure 1.4 – Image avant


Figure 1.5 – Image après

Figure 1.6 – Images

### 1.3.2 mer.png

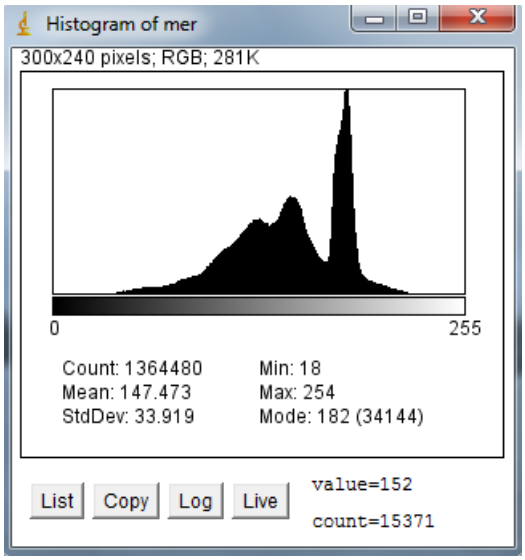


Figure 1.7 – Histogramme avant

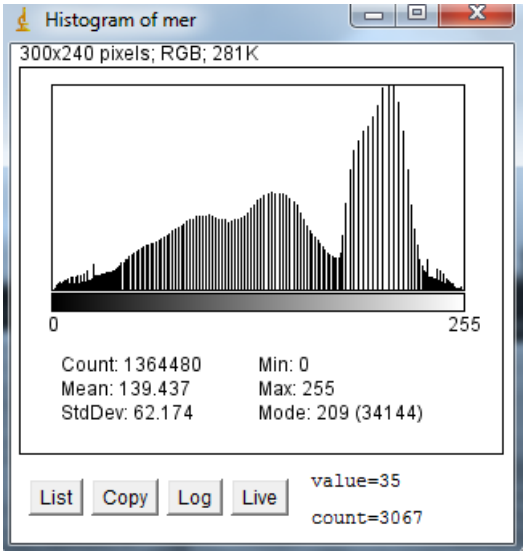


Figure 1.8 – Histogramme après

Figure 1.9 – Histogrammes

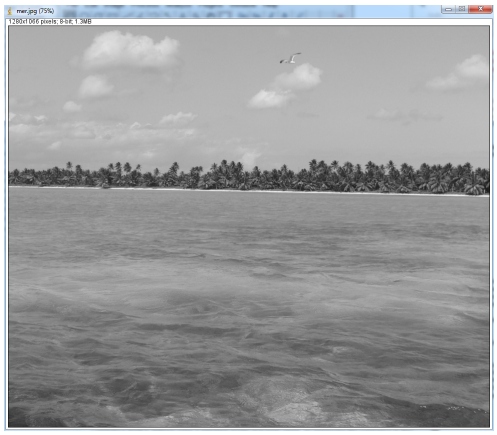

Figure 1.10 – Image avant

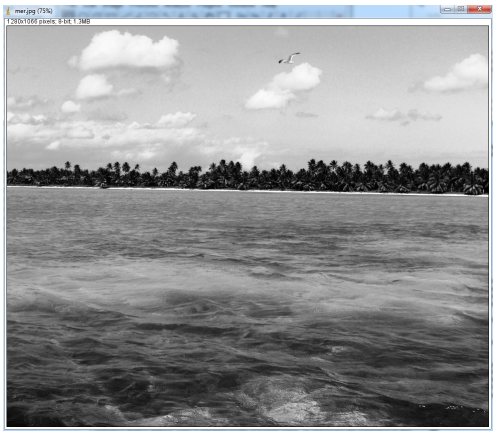

Figure 1.11 – Image après

Figure 1.12 – Images

## 1.4 Question 4

L'image soleil est celle ayant le plus changé car elle est à l'origine extrêmement sombre (très peu de contraste, c'est à dire très peu de niveaux de gris utilisés). L'égalisation du contraste fait donc beaucoup varier la couleur des pixels qui étaient à l'origine très proches.

## 1.5 Question 5

Les trois filtres rendent l'image floue. A cette étape, la taille du filtre n'influence que le niveau de flou (très ou peu flou). Cela permet de lisser l'image ainsi que de réduire son bruit.

## 1.6 Question 6

Un filtre moyenneur devrait en toute logique remplir l'image d'une couleur étant la moyenne de toutes les couleurs de l'image.

Cependant après test, nous observons un histogramme qui contient un pic centré à la couleur moyenne (mais d'autres valeurs sont présentes autour de ce pic).

## 1.7 Question 7

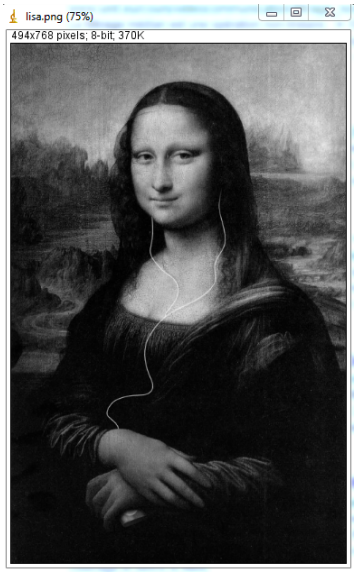


Figure 1.13 – Image avant


Figure 1.14 – Image après 1 convolution


Figure 1.15 – Image après 2 convolutions

Figure 1.16 – Images et convolution

Ce masque de convolution efface les détails présents dans l'image et réduit le bruit, et par conséquence la rend légèrement floue.

## 1.8 Question 8

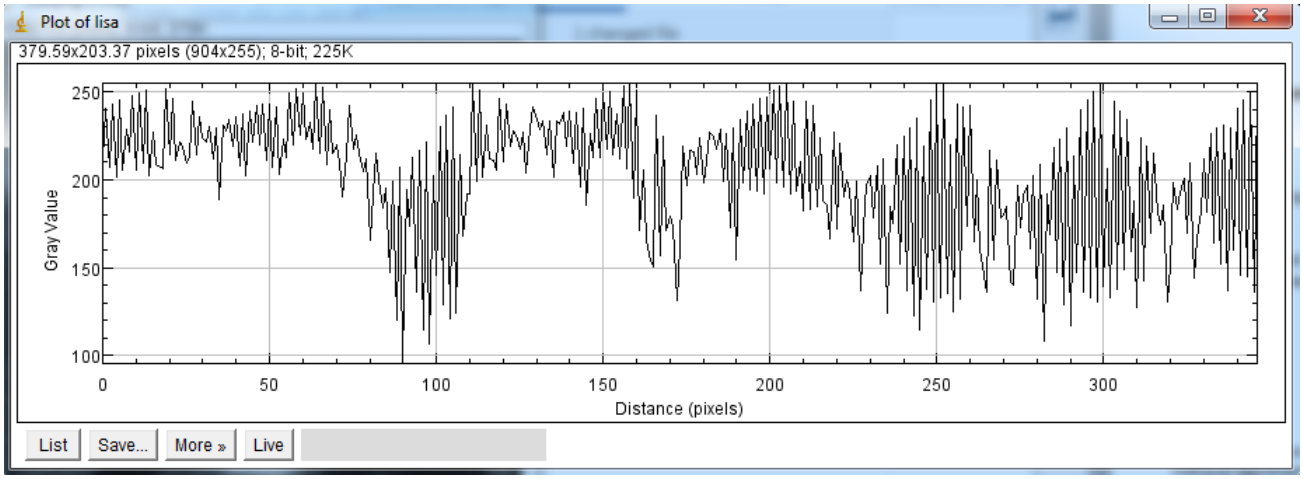


Figure 1.17 – Profile sur la droite avant

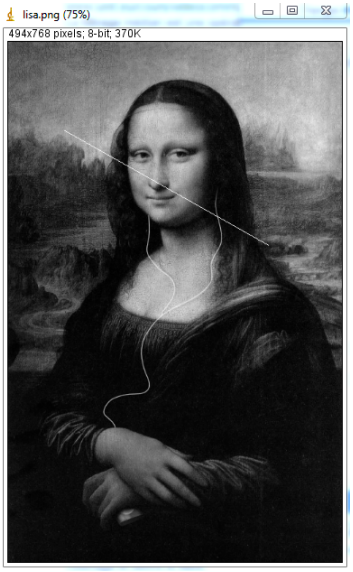


Figure 1.18 – Image avant

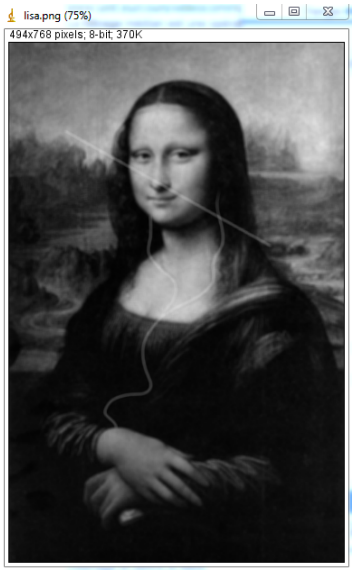


Figure 1.19 – Image après 1 convolution


Figure 1.20 – Image après 2 convolutions

Figure 1.21 – Images et convolution

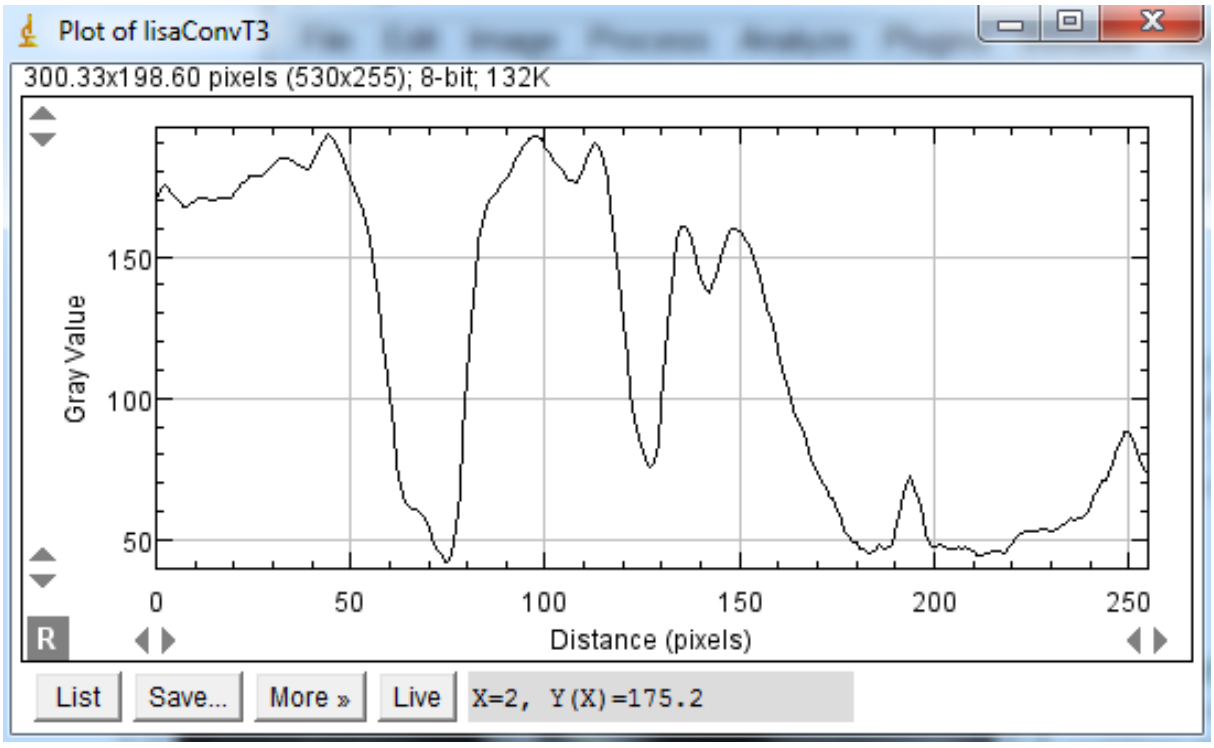


Figure 1.22 – Profile sur la droite après

On remarque que le diagramme est moins en dents de scie. Cela montre que le bruit a été atténué. De plus la courbe a des variations moins brutales et moins oscillantes. Cela traduit le lissage.

## 1.9 Question 10

Le point faible est que toute l'image devient floue alors que seules les parties bruitées "devraient" être affectées. De plus ce flou implique une perte de détails et de contraste.

# Chapter 2

# TP2

## 2.1 Question 1

### 2.1.1 Détection des contours

Nous avons choisit les images zebre.jpg et cellules.png. Le filtre en question applique des traits blancs sur les zones de contour. Sur les images choisies, nous remarquons que les rayures du zèbre et la membrane des cellules sont en blanc car ce sont de fortes zones de contour. A l'inverse l'intérieur/extérieur des cellules est entièrement noir car ce ne sont pas des zones de contour.

### 2.1.2 Deriche

Les étapes de Deriche sont:

- Applique un filtre pour lisser l'image
- Calcule des gradients
- Applique une suppression non-maximale pour éviter les fausses détections de contour.
- Détecte les bordures avec 2 valeurs bornes.

Ce filtre nous sort deux images résultat:

- La première est parfaitement nette. On y voit toute l'image en noir ainsi que les contours en blanc.
- La deuxième est similaire à la première à la différence que l'algorithme anti-bruit la rend floue.

Nous avons testé le filtre Deriche sur les 3 images suivantes:

- aqui.jpg : L'image d'origine est extrêmement sombre bien que l'on discerne tout de même des contours plus clairs. L'algorithme est donc très efficace car même si l'œil humain a du mal à distinguer les contours, les zones sont clairement distinctes en termes de couleurs. Voir Figure **??**.
- noise.jpg : Cette image est extrêmement bruitée. Bien que reconnaissant les contours de deux formes, l'algorithme considère aussi tous les "points bruit" et affiche donc un ensemble de contours qui ne devrait pas être reconnu. Voir Figure **??**.
- delphin.jpg : Cette image n'est composée que de points, l'algorithme ne peut donc pas y reconnaitre de contours. Les résultats obtenus sont donc quasiment entièrement noirs. Voir Figure **??**.

Figure 2.1 – Aqui avant

Figure 2.2 – Aqui après

Figure 2.3 – Aqui

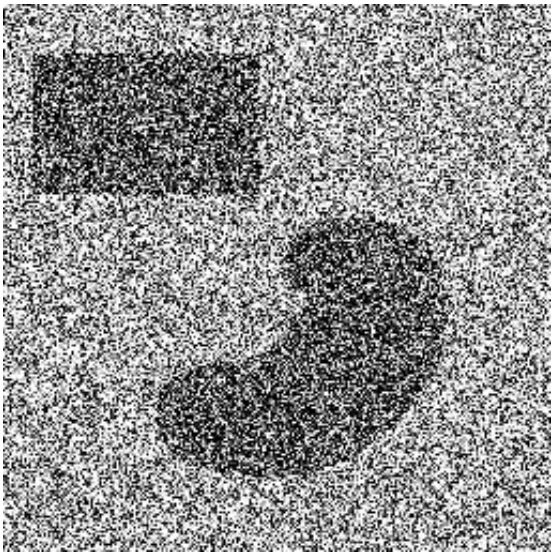

Figure 2.4 – Noise avant

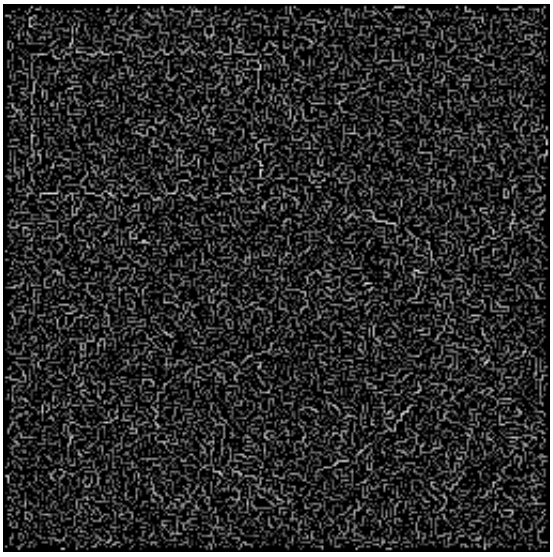

Figure 2.5 – Noise après

Figure 2.6 – Noise

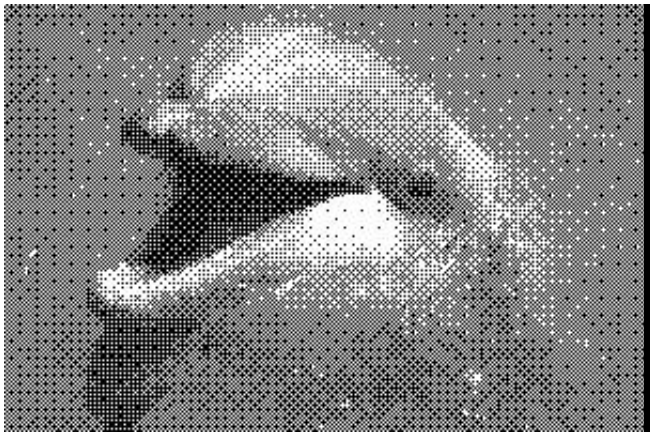

Figure 2.7 – Delphin avant

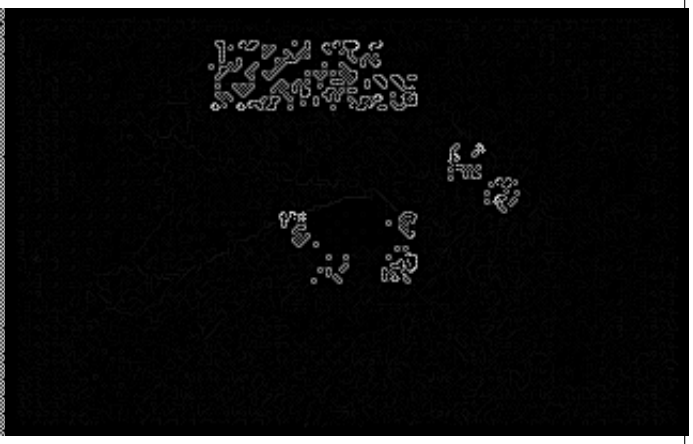

Figure 2.8 – Delphin après

Figure 2.9 – Delphin

## 2.2 Question 2

### 2.2.1 jeu1 & jeu2

Afin de mettre en évidence les différences entre les images, il faut appliquer l'opération XOR sur celles-ci. En effet cette opération permet de ne garder que les pixels étant strictement différents.

### 2.2.2 jeu3

Dans le fichier jeu3, on voit qu'une colonne de pixels blancs est présente tout à gauche de l'image, cela fausserai donc les calculs. Afin d'y remédier, nous proposons l'algorithme suivant: on commence par parcourir verticalement l'image. Toute colonne ne contenant que des pixels blancs sera éliminée, l'image sera donc décalée d'un pixel vers la gauche. Une fois que toutes les colonnes blanches ont étés supprimées, nous pouvons appliquer le XOR de l'opération précédente.

## 2.3 Question 3

### 2.3.1 Zèbre horizontal

Afin de ne faire apparaitre que les traits horizontaux de l'image zèbre.jpg, nous utilisons le masque de convolution suivant: $\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$.
En effet, ce filtre ne prend en compte que les pixels au dessus et en dessous du pixel courant. Dans le cas ou ceux-ci sont de même couleur, les valeurs après passage dans le masque s'annulent, on voit donc du noir. Si ceux-ci sont différents (bordure horizontale), les valeurs ne s'annulent pas, ce qui se traduit pas un pixel blanc.

Figure 2.10 – Zèbre horizontal

### 2.3.2 Suzan vertical

Afin de ne faire apparaitre que les traits verticaux de l'image suzan.jpg, nous utilisons le masque de convolution suivant: $\begin{pmatrix} -1 & 1 & 0 & 1 & -1 \end{pmatrix}$.
En effet, ce filtre le prend en compte que les pixels à gauche et à droite du pixel courant. Ce dernier est un mélange du filtre précédent mais détecte les bordure des deux cotés (transition blanc → noir et noir → blanc). En effet le filtre précédent ne détectait que la transition blanc → noir car lors de la transition noir → blanc la valeur obtenue était de -255 qui est arrondie à 0 (donc aucune détection).

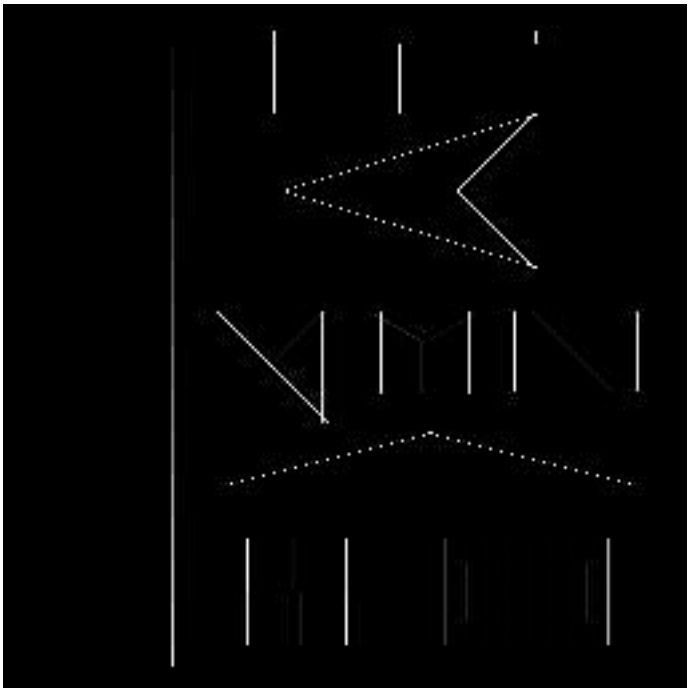

Figure 2.11 – Suzan vertical

## 2.4 Question 4

Après binarisation et analyse, nous trouvons bien 329 cellules.

## 2.5 Question 5

Afin d'obtenir un résultat ≈ 370 nous pouvons réaliser les étapes suivantes:

- Erode
- Open
- Erode
- Open
- Erode
- Open

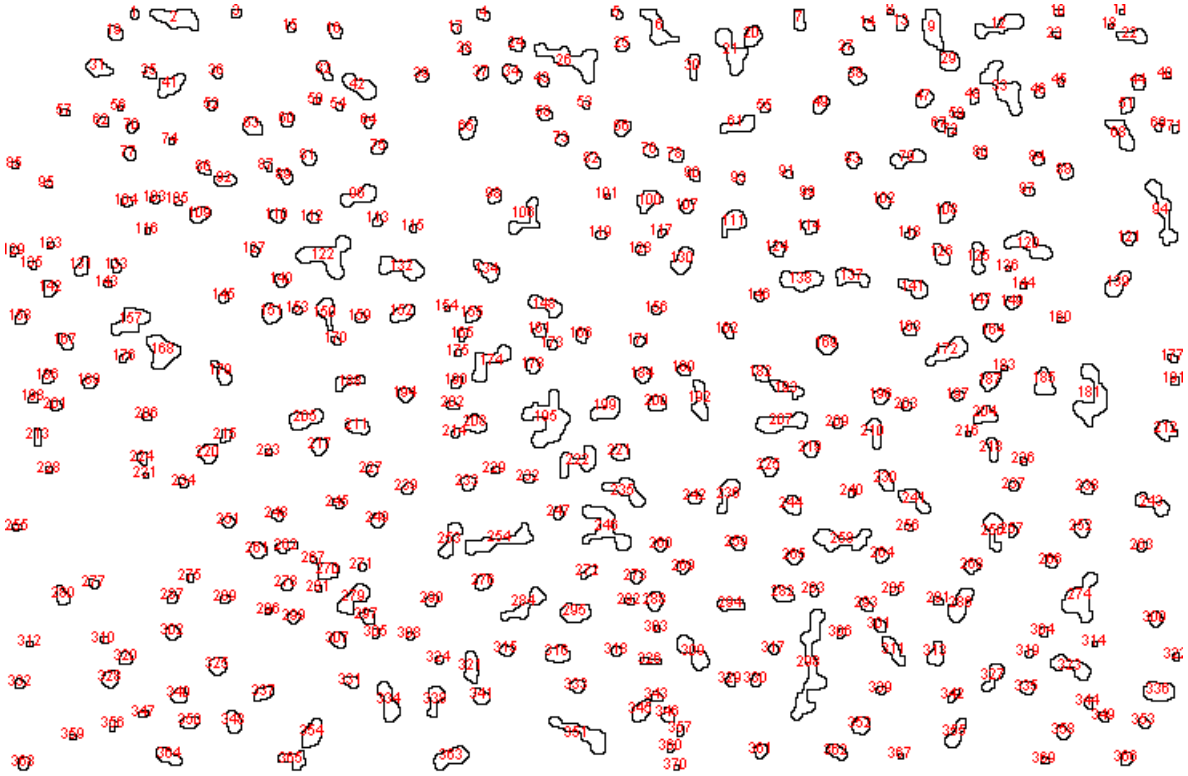

Figure 2.12 – Cell 370

De manière plus logique, nous pouvons réaliser les étapes suivantes:

- Fill holes
  - Permet de remplir les plus petits trous par la couleur environnante. Cela permet d'éviter les petites incohérences au sein des cellules.
- Watershed
  - Technique permettant de séparer les cellules qui étaient confondues / superposées / fusionnées.
- Erode
  - Réduit chaque cellule. Permet de supprimer les plus petites cellules qui n'en étaient pas en réalité (bruit).
- Open
  - Réagrandit les cellules.
- Erode
- Open
- Erode
- Open

Cependant, nous arrivons à 415 particules. Mais on remarque par exemple que l'ancienne 246 a été éclatée en plusieurs

cellules.

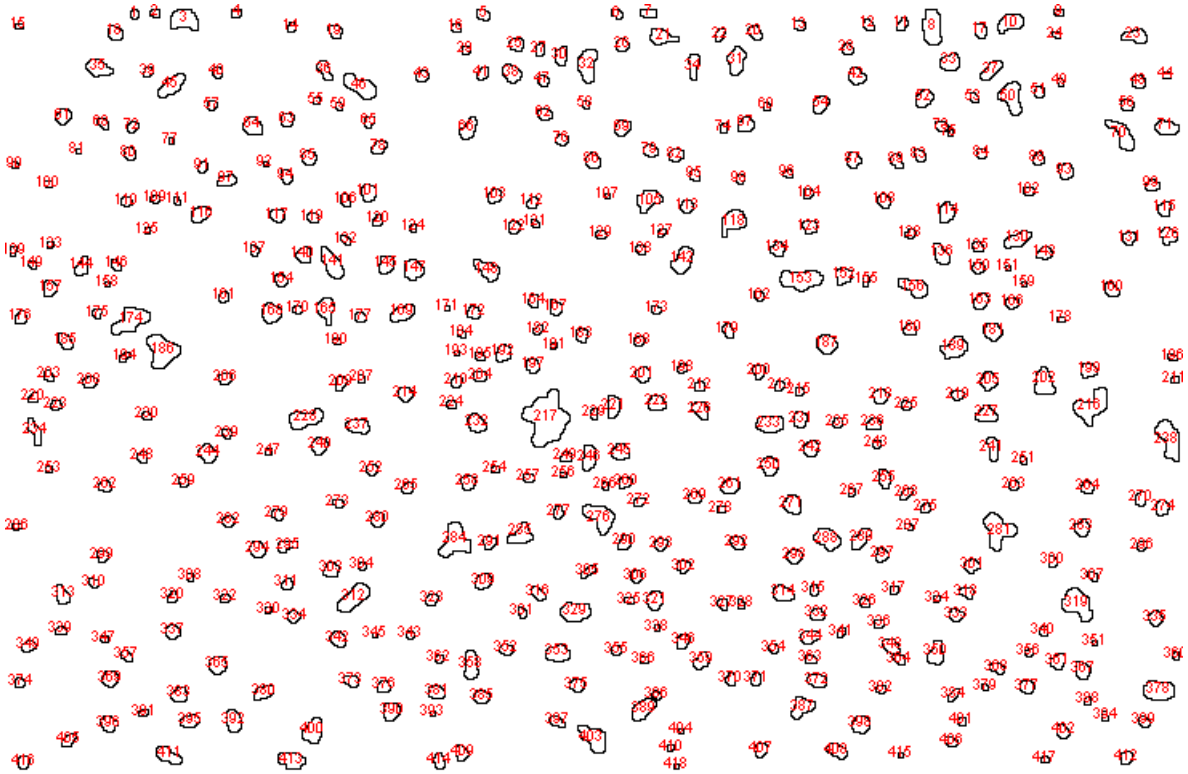

Figure 2.13 – Cell 415

# Chapter 3

# TP3

## 3.1 Question 2

La taille du tableau Pixels[] est égale au nombre de pixels total dans l'image analysée. De manière abstraite, elle vaut $width * height$.

Pour accéder à n'importe quel pixel d'une image, il fait utiliser la formule suivante : $ndg[x][y] = pixels[y * width + x]$.

## 3.2 Question 3

A partir d'une image donnée, le code remplace tous les pixels dont le niveau de gris est inférieur à 120 par un pixel noir, et tous ceux supérieur à 120 par un pixel blanc.

## 3.3 Question 4

Listing 3.1 – MonPlugin_.java

```
1  import ij.IJ;
2  import ij.ImagePlus;
3  import ij.plugin.filter.PlugInFilter;
4  import ij.process.ImageProcessor;
5
6  public class MonPlugin_ implements PlugInFilter
7  {
8     public void run(ImageProcessor ip)
9     {
10       byte[] pixels = (byte[]) ip.getPixels(); // Notez le cast en byte ()
11       int width = ip.getWidth();
12       int height = ip.getHeight();
13       int ndg;
14       double total = 0;
15       for(int y = 0; y < height; y++)
16          for(int x = 0; x < width; x++)
17          { // pas completement optimal mais pedagogique
18             ndg = pixels[y * width + x] & 0xff;
19             total += ndg;
20             if(ndg < 120)
21                pixels[y * width + x] = (byte) 0;
22             else
23                pixels[y * width + x] = (byte) 255;
24          }
25       IJ.showMessage(String.format("Niveau de gris moyen: %.2f", total / pixels.length));
26    }
27
28    public int setup(String arg, ImagePlus imp)
29    {
30       if(arg.equals("about"))
31       {
```

```
32          IJ.showMessage("Traitement de l'image");
33          return DONE;
34       }
35       return DOES_8G;
36    }
37 }
```

L'algorithme calcul le niveau de gris moyen de l'image. Plus celui-ci est élevé, plus l'image d'origine était claire, et inversement.

## 3.4 Question 5

Listing 3.2 – FindSimilar_.java

```
1  import ij.IJ;
2  import ij.ImagePlus;
3  import ij.plugin.filter.PlugInFilter;
4  import ij.process.ImageConverter;
5  import ij.process.ImageProcessor;
6  import java.io.File;
7  import java.util.*;
8
9  public class FindSimilar_ implements PlugInFilter
10 {
11    public void run(ImageProcessor ip)
12    {
13       String path = IJ.getDirectory("Selectionnez un dossier avec des images");
14       File[] files = new File(path == null ? "." : path).listFiles();
15       if(files != null && files.length != 0)
16       {
17          double avgReal = AverageNdg(ip);
18          Map<Double, List<File>> similarities = new HashMap<Double, List<File>>();
19          //initialization variables locales
20          for(File file : files)
21          {
22             // creation dune image temporaire
23             ImagePlus tempImg = new ImagePlus(file.getAbsolutePath());
24             new ImageConverter(tempImg).convertToGray8();
25             ImageProcessor ipTemp = tempImg.getProcessor();
26             double dst = Math.abs(AverageNdg(ipTemp) - avgReal);
27             if(!similarities.containsKey(dst))
28                similarities.put(dst, new ArrayList<File>());
29             similarities.get(dst).add(file);
30          }
31          double minDist = Collections.min(similarities.keySet());
32          IJ.showMessage("L'image la plus proche est " + getBeautifulFiles(similarities.get(minDist)) + "
                avec une distance de " + minDist);
33       }
34       else
35          IJ.showMessage("Merci de sélectionner un dossier avec des images àcomparer");
36    }
37
38    private String getBeautifulFiles(List<File> files)
39    {
40       StringBuilder builder = new StringBuilder();
41       for(File file : files)
42          builder.append(file.getAbsolutePath()).append(", ");
43       builder.delete(builder.length() - 2, builder.length());
44       return builder.toString();
45    }
46
47    public double AverageNdg(ImageProcessor ip)
48    {
49       byte[] pixels = (byte[]) ip.getPixels();
50       int width = ip.getWidth();
51       int height = ip.getHeight();
52       double total = 0;
```

```
53      for(int y = 0; y < height; y++)
54         for(int x = 0; x < width; x++)
55            total += pixels[y * width + x] & 0xff;
56      return total / pixels.length;
57   }
58
59   public int setup(String arg, ImagePlus imp)
60   {
61      if(arg.equals("about"))
62      {
63         IJ.showMessage("Traitement de l'image v2");
64         return DONE;
65      }
66      return DOES_8G;
67   }
68}
```

## 3.5 Question 6

Voir le fichier Java FindSimilar_.

## 3.6 Question 7

Listing 3.3 – PluginColor_.java

```
1  import ij.IJ;
2  import ij.ImagePlus;
3  import ij.WindowManager;
4  import ij.gui.ImageWindow;
5  import ij.plugin.filter.PlugInFilter;
6  import ij.process.ImageProcessor;
7  import java.awt.*;
8  import java.awt.event.WindowAdapter;
9  import java.awt.event.WindowEvent;
10 import java.awt.image.ColorModel;
11 import java.io.File;
12 import java.io.FileOutputStream;
13 import java.io.PrintWriter;
14 import java.util.HashMap;
15 import java.util.Map;
16
17 public class PluginColor_ implements PlugInFilter
18 {
19    private static HashMap<Color, String> baseColors;
20    private ImagePlus imp;
21
22    public void run(ImageProcessor ip)
23    {
24       //Color mappings
25       baseColors = new HashMap<Color, String>();
26       baseColors.put(new Color(255, 255, 0), "Jaune");
27
28       baseColors.put(new Color(0, 255, 0), "Vert");
29       baseColors.put(new Color(0, 189, 9), "Vert");
30       baseColors.put(new Color(105, 240, 112), "Vert");
31       baseColors.put(new Color(0, 118, 36), "Vert");
32       baseColors.put(new Color(72, 86, 33), "Vert");
33       baseColors.put(new Color(44, 58, 58), "Vert");
34
35       baseColors.put(new Color(255, 0, 0), "Rouge");
36       baseColors.put(new Color(255, 0, 66), "Rouge");
37       baseColors.put(new Color(199, 0, 39), "Rouge");
38       baseColors.put(new Color(129, 25, 0), "Rouge");
39
40       baseColors.put(new Color(0, 0, 255), "Bleu");
```

```
        baseColors.put(new Color(17, 31, 58), "Bleu");
        baseColors.put(new Color(119, 24, 255), "Bleu");
        baseColors.put(new Color(91, 33, 74), "Bleu");
        baseColors.put(new Color(117, 63, 121), "Bleu");
        baseColors.put(new Color(0, 255, 255), "Bleu");
        baseColors.put(new Color(0, 189, 195), "Bleu");
        baseColors.put(new Color(0, 174, 255), "Bleu");
        baseColors.put(new Color(134, 217, 255), "Bleu");
        baseColors.put(new Color(98, 140, 255), "Bleu");

        baseColors.put(new Color(102, 51, 0), "Marron");
        baseColors.put(new Color(113, 76, 43), "Marron");

        baseColors.put(new Color(128, 128, 128), "Gris");
        baseColors.put(new Color(192, 192, 192), "Gris");
        baseColors.put(new Color(64, 64, 64), "Gris");
        baseColors.put(new Color(64, 64, 64), "Gris");
        baseColors.put(new Color(30, 19, 17), "Gris");

        baseColors.put(new Color(255, 255, 255), "Blanc");
        baseColors.put(new Color(202, 212, 221), "Blanc");

        baseColors.put(new Color(0, 0, 0), "Noir");
        baseColors.put(new Color(2, 11, 12), "Noir");
        baseColors.put(new Color(7, 18, 19), "Noir");

        baseColors.put(new Color(255, 200, 0), "Orange");
        baseColors.put(new Color(220, 74, 1), "Orange");
        baseColors.put(new Color(234, 142, 119), "Orange");
        getColors(ip.duplicate().convertToRGB());
    }

    //Write into the output file
    private String getBeautifulColors(Map<String, Integer> colors, int count, double threshold)
    {
        StringBuilder builder = new StringBuilder();
        for(String color : colors.keySet())
            if(colors.get(color) > threshold * count)
                builder.append(color).append(":").append(String.format("%.2f%%", 100 * colors.get(color) /
                    (double) count)).append(", ");

        if(builder.length() > 1)
            builder.delete(builder.length() - 2, builder.length());

        File outFile = new File(IJ.getDirectory("current"), imp.getTitle() + "_tag" + ".txt");
        PrintWriter pw = null;
        try
        {
            pw = new PrintWriter(new FileOutputStream(outFile));
            try
            {
                pw.print("Quality: ");
                for(String color : colors.keySet())
                    if(colors.get(color) > threshold)
                        builder.append(color).append(" ");
                pw.println();
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            if(pw != null)
```

```
            pw.close();
        }

        return builder.toString();
    }

    private HashMap<String, Integer> getColors(ImageProcessor ip)
    {
        //Create a second processor, pass it in RGB & apply median filter to make it less sharp
        ImageProcessor ip2 = ip.createProcessor(ip.getWidth(), ip.getHeight());
        ip.medianFilter();
        ip.setColorModel(ColorModel.getRGBdefault());

        //For each pixel find its closest color in the mappings
        HashMap<String, Integer> colors = new HashMap<String, Integer>();
        for(int i = 0; i < ip.getWidth(); i++)
        {
            for(int j = 0; j < ip.getHeight(); j++)
            {
                Color c = getClosestColor(i, j, ip.getColorModel().getRGB(ip.getPixel(i, j)));
                String colorName = baseColors.get(c);
                if(!colors.containsKey(colorName))
                    colors.put(colorName, 0);
                colors.put(colorName, colors.get(colorName) + 1);
                ip2.putPixel(i, j, c.getRGB());
            }
        }

        //Keep only those that represent more than 10% of the pixels
        displayImage("Coleurs: " + getBeautifulColors(colors, ip.getWidth() * ip.getHeight(), 0.1), ip2);

        return colors;
    }

    private void displayImage(String title, ImageProcessor imageProcessor)
    {
        final ImageWindow iw = new ImageWindow(new ImagePlus(WindowManager.makeUniqueName(title),
            imageProcessor));
        iw.addWindowListener(new WindowAdapter(){
            @Override
            public void windowClosed(WindowEvent e)
            {
                super.windowClosed(e);
                WindowManager.removeWindow(iw);
            }
        });
        WindowManager.addWindow(iw);
    }


    private Color getClosestColor(int x, int y, int i)
    {
        double minDist = Double.MAX_VALUE;
        Color bestColor = null;

        Color c = new Color(i);
        float hsb1[] = new float[3];
        Color.RGBtoHSB(c.getRed(), c.getGreen(), c.getBlue(), hsb1);

        //Test each color and keep minimum distance
        for(Color c2 : baseColors.keySet())
        {
            float hsb2[] = new float[3];
            Color.RGBtoHSB(c2.getRed(), c2.getGreen(), c2.getBlue(), hsb2);
            double dist = getDistanceHSB(hsb1, hsb2);
            if(dist < minDist)
            {
                minDist = dist;
                bestColor = c2;
```

```
177         }
178      }
179
180      return bestColor;
181   }
182
183   //Get the distance in the HSB space where H is less powerful than the two others
184   private double getDistanceHSB(float[] hsb1, float[] hsb2)
185   {
186      return 0.24 * Math.sqrt(Math.pow(hsb1[0] - hsb2[0], 2)) + 0.38 * Math.sqrt(Math.pow(hsb1[1] -
           hsb2[1], 2)) + 0.38 * Math.sqrt(Math.pow(hsb1[2] - hsb2[2], 2));
187   }
188
189   public int setup(String arg, ImagePlus imp)
190   {
191      this.imp = imp;
192      if(arg.equals("about"))
193      {
194         IJ.showMessage("Traitement de l'image v2");
195         return DONE;
196      }
197      return DOES_ALL;
198   }
199}
```

Afin de récupérer les couleurs dominantes, nous parcourons chaque pixel. Nous comparons ces derniers à des couleurs de référence. Cette comparaison est basée sur le calcul d'une distance dans l'espace HSB, puis nous gardons la plus petite, qui correspond à la couleur la plus proche.

Ces résultats de comparaison sont stockées dans un tableau qui, pour chaque couleur de référence, compte le nombre de pixels associés. Nous considérons ensuite que les couleurs présentes sur plus de 10% des pixels de l'image sont notables. Nous les écrivons donc dans le fichier de sortie.

Afin d'assurer d'être le plus précis possible, nous avons enregistré des nuances pour chaque couleur. Par exemple le bleu pur RGB (0, 0, 255) est extrêmement foncé et fluo. Ainsi les bleus plus doux comme le ciel ou la mer ne sont pas reconnus comme bleu mais plutôt comme une couleur clair, gris ou blanc. L'ajout de ces nuances permet d'associer de manière précise chaque pixel à sa couleur de réfénrence la plus proche.

Listing 3.4 – PluginQuality_.java

```
1  import ij.IJ;
2  import ij.ImagePlus;
3  import ij.WindowManager;
4  import ij.gui.ImageWindow;
5  import ij.plugin.filter.PlugInFilter;
6  import ij.process.ImageConverter;
7  import ij.process.ImageProcessor;
8  import java.awt.*;
9  import java.awt.event.WindowAdapter;
10 import java.awt.event.WindowEvent;
11 import java.awt.image.ColorModel;
12 import java.io.File;
13 import java.io.FileOutputStream;
14 import java.io.PrintWriter;
15 import java.util.HashMap;
16
17 public class PluginQuality_ implements PlugInFilter
18 {
19    private ImagePlus imp;
20
21    public void run(ImageProcessor ip)
22    {
23       printOut(getIntensity(ip.duplicate()), getBlurrNess(ip.duplicate()));
24    }
25
26    private String getBlurrNess(ImageProcessor ip)
27    {
28       //To get the blurrness we apply a LoG
29       ImagePlus imagePlus = new ImagePlus("TESTT", ip);
30       ImageConverter imageConverter = new ImageConverter(imagePlus);
```

```
        imageConverter.convertToGray8();
        ImageProcessor imageProcessor = imagePlus.getProcessor().convertToFloatProcessor();
        /*imageProcessor.convolve(new float[]{ //LoG, sigma=1.4
                                        0,0,3,2,2,2,3,0,0,
                                        0,2,3,5,5,5,3,2,0,
                                        3,3,5,3,0,3,5,3,3,
                                        2,5,3,-12,-23,-12,3,5,2,
                                        2,5,0,-23,-40,-23,0,5,2,
                                        2,5,3,-12,-23,-12,3,5,2,
                                        3,3,5,3,0,3,5,3,3,
                                        0,2,3,5,5,5,3,2,0,
                                        0,0,3,2,2,2,3,0,0
        }, 9, 9);*/
        imageProcessor.convolve(new float[]{ //LoG, sigma=1.4
                                        0,
                                        1,
                                        1,
                                        2,
                                        2,
                                        2,
                                        1,
                                        1,
                                        0,
                                        1,
                                        2,
                                        4,
                                        5,
                                        5,
                                        5,
                                        4,
                                        2,
                                        1,
                                        1,
                                        4,
                                        5,
                                        3,
                                        0,
                                        3,
                                        5,
                                        4,
                                        1,
                                        2,
                                        5,
                                        3,
                                        -12,
                                        -24,
                                        -12,
                                        3,
                                        5,
                                        2,
                                        2,
                                        5,
                                        0,
                                        -24,
                                        -40,
                                        -24,
                                        0,
                                        5,
                                        2,
                                        2,
                                        5,
                                        3,
                                        -12,
                                        -24,
                                        -12,
                                        3,
                                        5,
                                        2,
                                        1,
```

```
                                    4,
                                    5,
                                    3,
                                    0,
                                    3,
                                    5,
                                    4,
                                    1,
                                    1,
                                    2,
                                    4,
                                    5,
                                    5,
                                    5,
                                    4,
                                    2,
                                    1,
                                    0,
                                    1,
                                    1,
                                    2,
                                    2,
                                    2,
                                    1,
                                    1,
                                    0,
                                    }, 9, 9);
    //imageProcessor.convolve3x3(new int[]{0, 1, 0, 1, -4, 1, 0, 1, 0});

    //We get the average value
    long average = 0;
    for(int i = 0; i < ip.getWidth(); i++)
       for(int j = 0; j < ip.getHeight(); j++)
          average += ip.getPixelValue(i, j);
    average /= ip.getWidth() * ip.getHeight();

    //Get the variance
    long variance = 0;
    for(int i = 0; i < ip.getWidth(); i++)
       for(int j = 0; j < ip.getHeight(); j++)
          variance += Math.pow(ip.getPixelValue(i, j) - average, 2);
    variance /= ip.getWidth() * ip.getHeight();

    //If the variance is high, we assume the image is blurred
    String valTxt = variance < 3000 ? "Net" : "Flou";
    displayImage("Log: " + valTxt + "(" + variance + ")", imageProcessor);
    return valTxt;
 }

 private void printOut(String intensity, String blurrness)
 {
    File outFile = new File(IJ.getDirectory("current"), imp.getTitle() + "_tag" + ".txt");
    PrintWriter pw = null;
    try
    {
       pw = new PrintWriter(new FileOutputStream(outFile));
       try
       {
          pw.println("Quality: " + intensity + " " + blurrness);
       }
       catch(Exception e)
       {
          e.printStackTrace();
       }
    }
    catch(Exception e)
    {
       e.printStackTrace();
    }
```

```
169      finally
170      {
171         if(pw != null)
172            pw.close();
173      }
174   }
175
176   private String getIntensity(ImageProcessor ip)
177   {
178      //Convert to RGB
179      ip = ip.convertToRGB();
180      ImageProcessor ip2 = ip.createProcessor(ip.getWidth(), ip.getHeight());
181      ip.medianFilter();
182      ip.setColorModel(ColorModel.getRGBdefault());
183      HashMap<String, Double> colors = new HashMap<String, Double>();
184
185      //For each pixel, get if it's dark or light or medium.
186      for(int i = 0; i < ip.getWidth(); i++)
187      {
188         for(int j = 0; j < ip.getHeight(); j++)
189         {
190            Color c = new Color(ip.getColorModel().getRGB(ip.getPixel(i, j)));
191            float hsb1[] = new float[3];
192            Color.RGBtoHSB(c.getRed(), c.getGreen(), c.getBlue(), hsb1);
193
194            //If the B in HSB is < 0.33 dar, > 0.66 light, otherwise medium
195            String colorName = hsb1[2] < 0.33 ? "Sombre" : (hsb1[2] > 0.66 ? "Clair" : "Normal");
196            if(!colors.containsKey(colorName))
197               colors.put(colorName, 0D);
198            colors.put(colorName, colors.get(colorName) + 1);
199            ip2.putPixel(i, j, c.getRGB());
200         }
201      }
202
203      //Get the relative percentage. For example if the image is half light and half dark, both should npw
           represent 100%.
204      String maxKey = null;
205      double max = 0;
206      for(String key : colors.keySet())
207      {
208         double count = colors.get(key);
209         max = Math.max(max, count);
210         if(max == count)
211            maxKey = key;
212      }
213      IJ.showMessage("Intensity", maxKey);
214      return maxKey;
215   }
216
217   private void displayImage(String title, ImageProcessor imageProcessor)
218   {
219      final ImageWindow iw = new ImageWindow(new ImagePlus(WindowManager.makeUniqueName(title),
           imageProcessor));
220      iw.addWindowListener(new WindowAdapter()
221      {
222         @Override
223         public void windowClosed(WindowEvent e)
224         {
225            super.windowClosed(e);
226            WindowManager.removeWindow(iw);
227         }
228      });
229      WindowManager.addWindow(iw);
230   }
231
232   public int setup(String arg, ImagePlus imp)
233   {
234      this.imp = imp;
235      if(arg.equals("about"))
```

```
236     {
237        IJ.showMessage("Traitement de l'image v2");
238        return DONE;
239     }
240     return DOES_ALL;
241   }
242 }
```

Ce plugin permet de calculer la netteté de l'image, ainsi que sa luminosité.

Pour la netteté, nous commençons par transformer l'image en nuances de gris 8bits. Puis nous calculons sa moyenne de niveau de gris. Enfin, nous calculons la somme des carrés des écarts à la moyenne pour chaque pixel (variance). Si ce résultat est inférieur à 3000, nous cnsiderons cette image nette, sinon, floue.

Pour la luminosité, nous comptons le nombre de pixel Clair, Sombre ou Moyen grâce à la valeur Brightness (dans le model HSB). Si cette valeur est inférieure à 1 tier, le pixel est considéré sombre, supérieure à 2 tiers, clair, et entre les deux, moyen. La catégorie regroupant le plus de pixels de l'image sera celle écrite dans le fichier de sortie.

Listing 3.5 – PluginEdges_.java

```
1  import ij.IJ;
2  import ij.ImagePlus;
3  import ij.WindowManager;
4  import ij.gui.ImageWindow;
5  import ij.plugin.filter.Convolver;
6  import ij.plugin.filter.PlugInFilter;
7  import ij.process.ImageConverter;
8  import ij.process.ImageProcessor;
9  import java.awt.event.WindowAdapter;
10 import java.awt.event.WindowEvent;
11 import java.util.Random;
12
13 public class PluginEdges_ implements PlugInFilter
14 {
15    public void run(ImageProcessor ip)
16    {
17       new ImageConverter(new ImagePlus("Title", ip.duplicate())).convertToGray8();
18       convolve(ip.convertToFloatProcessor());
19    }
20
21    private void convolve(ImageProcessor ip)
22    {
23       ip.smooth();
24       ImageProcessor ip2 = ip.duplicate();
25
26       //Apply 2 convolution filters for vertical and horizontal edges
27       Convolver cv = new Convolver();
28       cv.convolve(ip, new float[]{
29             -1, 0, 1,
30             -2, 0, 2,
31             -1, 0, 1
32       }, 3, 3);
33       cv.convolve(ip2, new float[]{
34             -1, -2, -1,
35             0, 0, 0,
36             1, 2, 1
37       }, 3, 3);
38
39       for(int i = 0; i < ip.getWidth(); i++)
40          for(int j = 0; j < ip.getHeight(); j++)
41             ip.putPixelValue(i, j, Math.sqrt(Math.pow(ip.getPixelValue(i, j), 2) +
                   Math.pow(ip2.getPixelValue(i, j), 2)));
42       displayImage("Contours", ip);
43    }
44
45    private void displayImage(String title, ImageProcessor imageProcessor)
46    {
47       final ImageWindow iw = new ImageWindow(new ImagePlus(WindowManager.makeUniqueName(title + new
             Random().nextInt()), imageProcessor));
48       iw.addWindowListener(new WindowAdapter(){
```

```
        @Override
        public void windowClosed(WindowEvent e)
        {
           super.windowClosed(e);
           WindowManager.removeWindow(iw);
        }
     });
     WindowManager.addWindow(iw);
  }

  public int setup(String arg, ImagePlus imp)
  {
     if(arg.equals("about"))
     {
        IJ.showMessage("Traitement de l'image v2");
        return DONE;
     }
     return DOES_ALL;
  }
}
```

Ce plugin permet de faire une détection de contour grace à des masques de convolutions.

Un premier permet de détecter les bordures verticales, tandis qu'un deuxième les bordures horizontales. Par la suite nous combinons ces résultats avec une distance euclidienne.

Nous obtenons ainsi à la fin une image où les bordures sont en blanc. Plus ce blanc est intense, plus la bordure est marquée.

Listing 3.6 – PluginKind_.java

```
import ij.IJ;
import ij.ImagePlus;
import ij.WindowManager;
import ij.gui.ImageWindow;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageConverter;
import ij.process.ImageProcessor;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.ColorModel;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.HashMap;

public class PluginKind_ implements PlugInFilter
{
  private static HashMap<Color, String> baseColors;
  private ImagePlus imp;

  //For the colors see the PluginColor_, this is the same
  public void run(ImageProcessor ip)
  {
     baseColors = new HashMap<Color, String>();
     baseColors.put(new Color(255, 255, 0), "Jaune");

     baseColors.put(new Color(0, 255, 0), "Vert");
     baseColors.put(new Color(0, 189, 9), "Vert");
     baseColors.put(new Color(105, 240, 112), "Vert");
     baseColors.put(new Color(0, 118, 36), "Vert");
     baseColors.put(new Color(72, 86, 33), "Vert");
     baseColors.put(new Color(44, 58, 58), "Vert");

     baseColors.put(new Color(255, 0, 0), "Rouge");
     baseColors.put(new Color(255, 0, 66), "Rouge");
     baseColors.put(new Color(199, 0, 39), "Rouge");
     baseColors.put(new Color(129, 25, 0), "Rouge");

```

```
        baseColors.put(new Color(0, 0, 255), "Bleu");
        baseColors.put(new Color(17, 31, 58), "Bleu");
        baseColors.put(new Color(119, 24, 255), "Bleu");
        baseColors.put(new Color(91, 33, 74), "Bleu");
        baseColors.put(new Color(117, 63, 121), "Bleu");
        baseColors.put(new Color(0, 255, 255), "Bleu");
        baseColors.put(new Color(0, 189, 195), "Bleu");
        baseColors.put(new Color(0, 174, 255), "Bleu");
        baseColors.put(new Color(134, 217, 255), "Bleu");
        baseColors.put(new Color(98, 140, 255), "Bleu");

        baseColors.put(new Color(102, 51, 0), "Marron");
        baseColors.put(new Color(113, 76, 43), "Marron");

        baseColors.put(new Color(128, 128, 128), "Gris");
        baseColors.put(new Color(192, 192, 192), "Gris");
        baseColors.put(new Color(64, 64, 64), "Gris");
        baseColors.put(new Color(64, 64, 64), "Gris");
        baseColors.put(new Color(30, 19, 17), "Gris");

        baseColors.put(new Color(255, 255, 255), "Blanc");
        baseColors.put(new Color(202, 212, 221), "Blanc");

        baseColors.put(new Color(0, 0, 0), "Noir");
        baseColors.put(new Color(2, 11, 12), "Noir");
        baseColors.put(new Color(7, 18, 19), "Noir");

        baseColors.put(new Color(255, 200, 0), "Orange");
        baseColors.put(new Color(220, 74, 1), "Orange");
        baseColors.put(new Color(234, 142, 119), "Orange");

        printOut(getKind(ip.duplicate()));
    }

    private String getKind(ImageProcessor imageProcessor)
    {
        // int cellsX = 10.D;
        // int cellsY = 10.D;
        // int sizeX = (int) Math.ceil(imageProcessor.getWidth() / (double)cellsX);
        // int sizeY = (int) Math.ceil(imageProcessor.getHeight() / (double)cellsY);

        //Create a region of 50x50px that will scan the image
        int sizeX = 50;
        int sizeY = 50;
        int cellsX = (int) Math.ceil(imageProcessor.getWidth() / (double) sizeX);
        int cellsY = (int) Math.ceil(imageProcessor.getHeight() / (double) sizeY);

        //For the colors
        ImageProcessor ip2 = imageProcessor.duplicate();
        ip2.medianFilter();
        ip2.setColorModel(ColorModel.getRGBdefault());

        //To find edges, pass in gray scale and find edges
        ImagePlus imagePlus = new ImagePlus("TESTT", imageProcessor);
        ImageConverter imageConverter = new ImageConverter(imagePlus);
        imageConverter.convertToGray8();
        ImageProcessor ip3 = imagePlus.getProcessor().convertToFloatProcessor();
        ip3.findEdges();
        ip3.setBinaryThreshold();

        //Output image
        ImageProcessor ip4 = ip2.duplicate();
        for(int i = 0; i < cellsX; i++)
            ip4.drawRect(i * sizeX, 0, 2, imageProcessor.getHeight());
        for(int i = 0; i < cellsY; i++)
            ip4.drawRect(0, i * sizeY, imageProcessor.getWidth(), 2);

        //Cout region of each kind, -1 is sky, 1 is sea
        HashMap<Integer, Double> counts = new HashMap<Integer, Double>();
```

```
109     counts.put(-1, 0D);
110     counts.put(1, 0D);
111 
112     //For each region
113     for(int i = 0; i < cellsX; i++)
114     {
115        for(int j = 0; j < cellsY; j++)
116        {
117           //Get what is inside, -1 sky, 1 sea, 0 it wasn't blue
118           float val = processPart(ip2, ip3, i * sizeX, Math.min((i + 1) * sizeX,
                  imageProcessor.getWidth()), j * sizeY, Math.min((j + 1) * sizeY,
                  imageProcessor.getHeight()));
119 
120           //If the edges are more than "400", then there's a lot of them and we assume it's the sea
121           int index = val == 0f ? 0 : (val >= 400 ? 1 : -1);
122           ip4.drawString(String.format("%s\n%.0f", index == 0 ? "Rien" : (index == -1 ? "Ciel" : "Mer"),
                  val), (int) ((i + 0.1) * sizeX), (int) ((j + 0.5) * sizeY));
123           if(index != 0)
124              counts.put(index, counts.get(index) + 1);
125        }
126     }
127     double tot = counts.get(-1) + counts.get(1);
128     counts.put(-1, 100 * counts.get(-1) / tot);
129     counts.put(1, 100 * counts.get(1) / tot);
130     displayImage(String.format("Parts: Ciel=%.2f%% %s/Mer=%.2f%% %s", counts.get(-1), counts.get(-1) >=
          30 ? "OK" : "NON", counts.get(1), counts.get(1) >= 30 ? "OK" : "NON"), ip4);
131 
132     return (counts.get(-1) >= 30 ? "Ciel " : "") + (counts.get(1) >= 30 ? "Mer" : "");
133  }
134 
135  private float processPart(ImageProcessor imageProcessor, ImageProcessor imageEdges, int startX, int
       endX, int startY, int endY)
136  {
137     //For each pixels in the region, get the closest color
138     HashMap<String, Integer> colors = new HashMap<String, Integer>();
139     for(int i = startX; i < endX; i++)
140     {
141        for(int j = startY; j < endY; j++)
142        {
143           Color c = getClosestColor(imageProcessor.getColorModel().getRGB(imageProcessor.getPixel(i, j)));
144           String colorName = baseColors.get(c);
145           if(!colors.containsKey(colorName))
146              colors.put(colorName, 0);
147           colors.put(colorName, colors.get(colorName) + 1);
148        }
149     }
150 
151     //If the most present color is blue, we continue the process
152     int maxCol = -1;
153     String col = "";
154     for(String color : colors.keySet())
155        if(colors.get(color) > maxCol)
156        {
157           maxCol = colors.get(color);
158           col = color;
159        }
160 
161     if(!col.equals("Bleu"))
162        return 0f;
163 
164     //Count the "quantity" of edges
165     float total = 0;
166     for(int i = startX; i < endX; i++)
167     {
168        for(int j = startY; j < endY; j++)
169           total += imageEdges.getPixelValue(i, j) >= 100 ? 1 : 0;
170     }
171 
172     return total;
```

```
173   }
174
175   //For the colors see the PluginColor_, this is the same
176   private Color getClosestColor(int i)
177   {
178      double minDist = Double.MAX_VALUE;
179      Color bestColor = null;
180
181      Color c = new Color(i);
182      float hsb1[] = new float[3];
183      Color.RGBtoHSB(c.getRed(), c.getGreen(), c.getBlue(), hsb1);
184
185      for(Color c2 : baseColors.keySet())
186      {
187         float hsb2[] = new float[3];
188         Color.RGBtoHSB(c2.getRed(), c2.getGreen(), c2.getBlue(), hsb2);
189         double dist = getDistanceHSB(hsb1, hsb2);
190         if(dist < minDist)
191         {
192            minDist = dist;
193            bestColor = c2;
194         }
195      }
196
197      return bestColor;
198   }
199
200   //For the colors see the PluginColor_, this is the same
201   private double getDistanceHSB(float[] hsb1, float[] hsb2)
202   {
203      return 0.24 * Math.sqrt(Math.pow(hsb1[0] - hsb2[0], 2)) + 0.38 * Math.sqrt(Math.pow(hsb1[1] -
          hsb2[1], 2)) + 0.38 * Math.sqrt(Math.pow(hsb1[2] - hsb2[2], 2));
204   }
205
206   private void printOut(String kind)
207   {
208      File outFile = new File(IJ.getDirectory("current"), imp.getTitle() + "_tag" + ".txt");
209      PrintWriter pw = null;
210      try
211      {
212         pw = new PrintWriter(new FileOutputStream(outFile));
213         try
214         {
215            pw.println("Type: " + kind);
216         }
217         catch(Exception e)
218         {
219            e.printStackTrace();
220         }
221      }
222      catch(Exception e)
223      {
224         e.printStackTrace();
225      }
226      finally
227      {
228         if(pw != null)
229            pw.close();
230      }
231   }
232
233   private void displayImage(String title, ImageProcessor imageProcessor)
234   {
235      final ImageWindow iw = new ImageWindow(new ImagePlus(WindowManager.makeUniqueName(title),
          imageProcessor));
236      iw.addWindowListener(new WindowAdapter()
237      {
238         @Override
239         public void windowClosed(WindowEvent e)
```

```
240         {
241            super.windowClosed(e);
242            WindowManager.removeWindow(iw);
243         }
244      });
245      WindowManager.addWindow(iw);
246   }
247
248   public int setup(String arg, ImagePlus imp)
249   {
250      this.imp = imp;
251      if(arg.equals("about"))
252      {
253         IJ.showMessage("Traitement de l'image v2");
254         return DONE;
255      }
256      return DOES_ALL;
257   }
258 }
```

Ce plugin permet de définir si une image représente la mer ou le ciel (ou les deux).

Pour ce faire, on commence par découper l'image en cases de 50x50 pixels. Puis nous appliquons le traitement suivant sur chaque case :

- Nous cherche la couleur dominante de la case. S'il ne s'agit pas du Bleu, on s'arrête, la mer ou le ciel étant forcment bleu.
- Si c'est le cas, on dessine l'image des contours (pixels clairs si un contour est marqué, sombre sinon) et, pour chaque pixel de la case, regarde sa valeur. Si la valeur est supérieure à 100 (soit un contour assez marqué), on le compte.
- Au final, s'il y a plus de 400 pixels dans la case correspondants au critère précédent, il s'agit d'une case de mer, sinon d'une case de ciel.