Analyse d'images

May 25, 2018

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

Contents

1 TP1 **2**
1.1 Question 1 . . . 2
1.2 Question 2 . . . 2
1.2.1 Zone 1 . . . 2
1.2.2 Zone 2 . . . 2
1.3 Question 3 . . . 3
1.3.1 mystere.pgm . . . 3
1.3.2 mer.png . . . 4
1.4 Question 4 . . . 4
1.5 Question 5 . . . 4
1.6 Question 6 . . . 5
1.7 Question 7 . . . 5
1.8 Question 8 . . . 5
1.9 Question 10 . . . 6

2 TP2 **7**
2.1 Question 1 . . . 7
2.1.1 Détection des contours . . . 7
2.1.2 Deriche . . . 7
2.2 Question 2 . . . 9
2.2.1 jeu1 & jeu2 . . . 9
2.2.2 jeu3 . . . 9
2.3 Question 3 . . . 9
2.3.1 Zèbre horizontal . . . 9
2.3.2 Suzan vertical . . . 10
2.4 Question 4 . . . 10
2.5 Question 5 . . . 11

3 TP3 **13**
3.1 Question 2 . . . 13
3.2 Question 3 . . . 13
3.3 Question 4 . . . 13
3.4 Question 5 . . . 14
3.5 Question 6 . . . 15
3.6 Question 7 . . . 15

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 (≈ 250) de gris pixels clairs (a.k.a. blancs).

1.3 Question 3

1.3.1 mystere.pgm

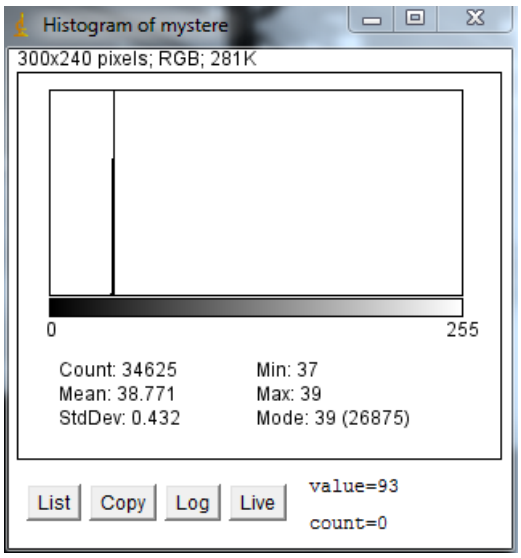

Figure 1.1 – Histogramme avant

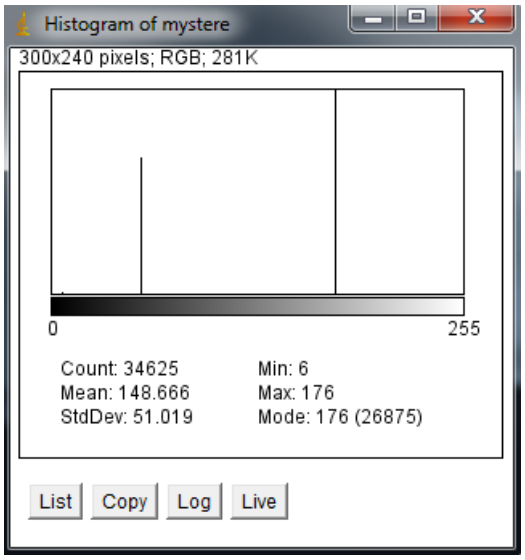

Figure 1.2 – Histogramme après

Figure 1.3 – Histogrammes

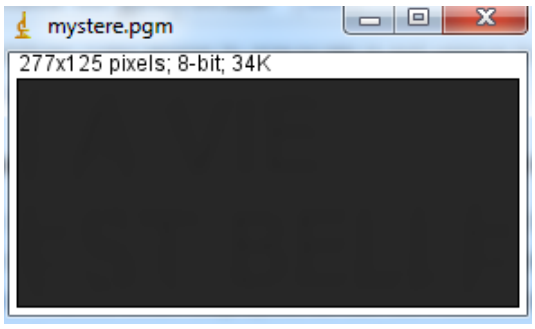

Figure 1.4 – Image avant

Figure 1.5 – Image après

Figure 1.6 – Images

1.3.2 mer.png

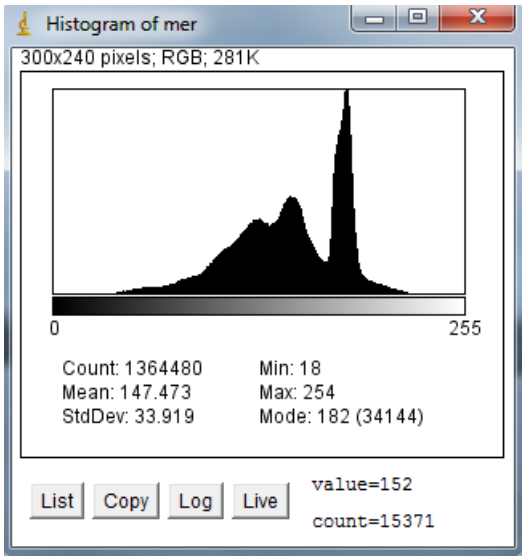

Figure 1.7 – Histogramme avant

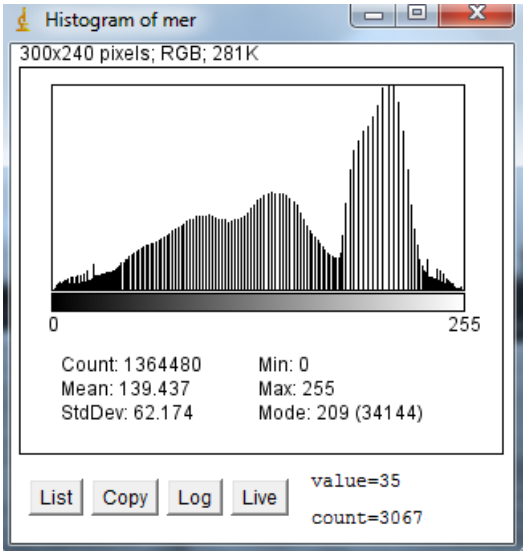

Figure 1.8 – Histogramme après

Figure 1.9 – Histogrammes

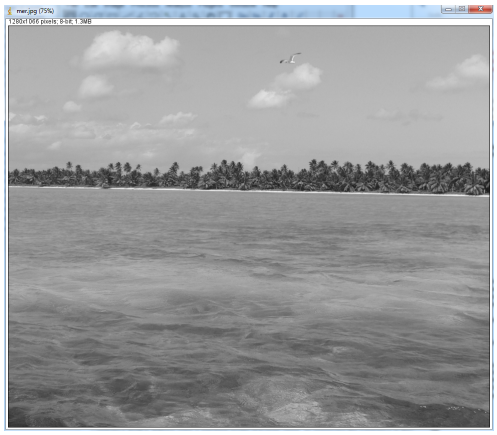

Figure 1.10 – Image avant

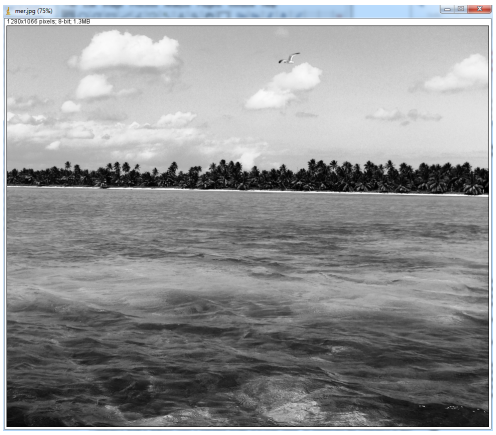

Figure 1.11 – Image après

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

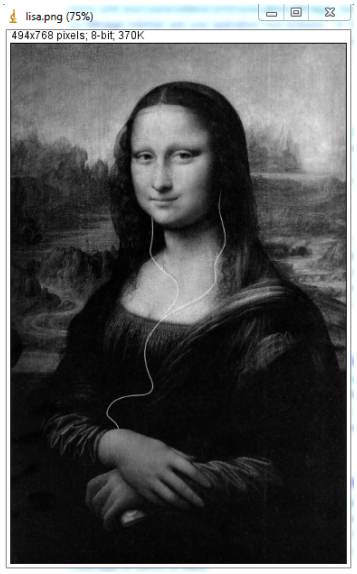

Figure 1.13 – Image avant

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

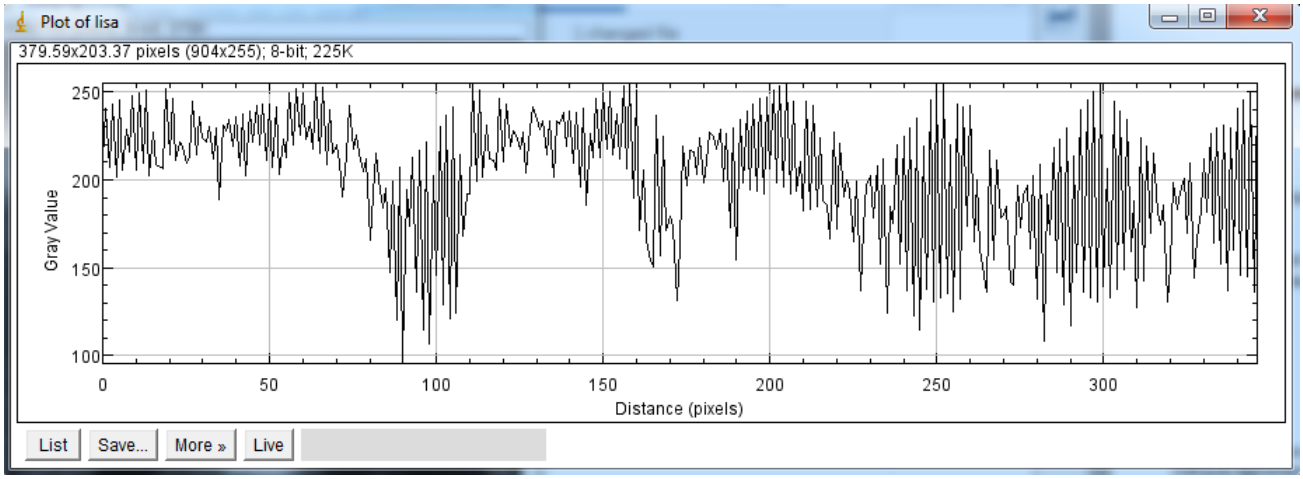

Figure 1.17 – Profile sur la droite avant

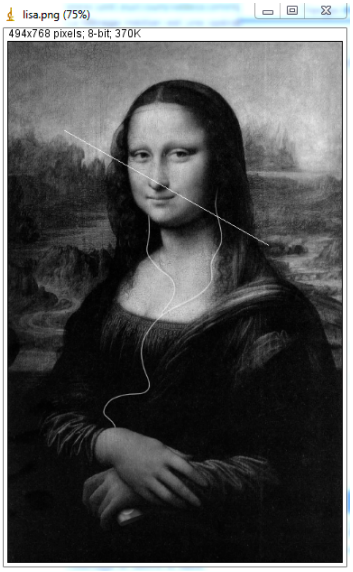

Figure 1.18 – Image avant

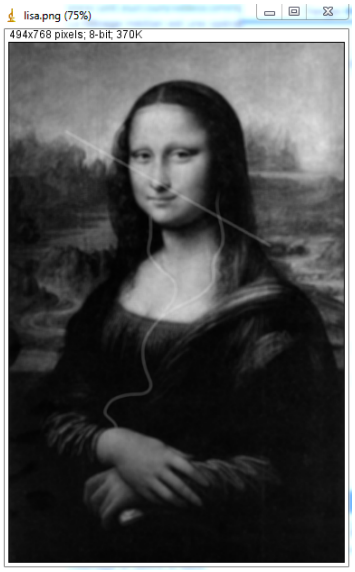

Figure 1.19 – Image après 1 convolution

Figure 1.20 – Image après 2 convolutions

Figure 1.21 – Images et convolution

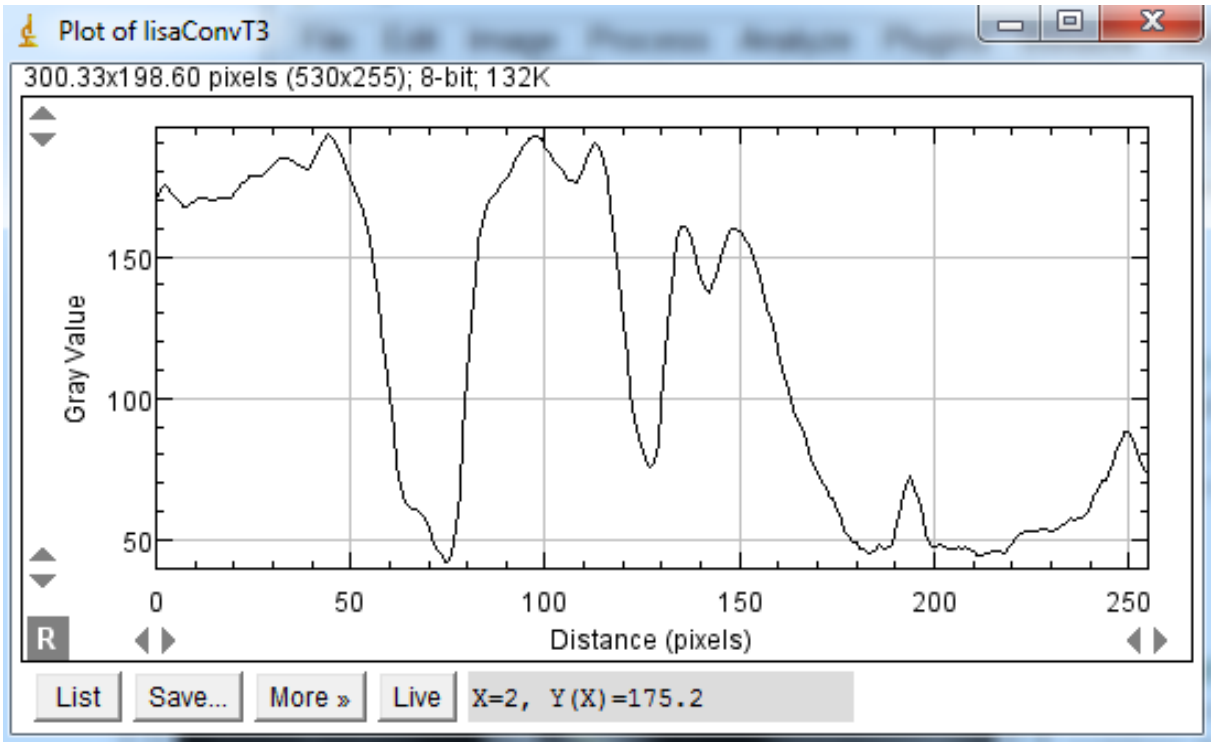

Figure 1.22 – Profile sur la droite après

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

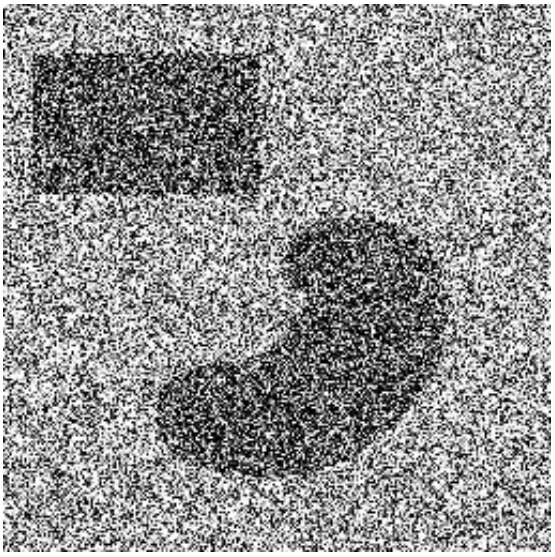

Figure 2.4 – Noise avant

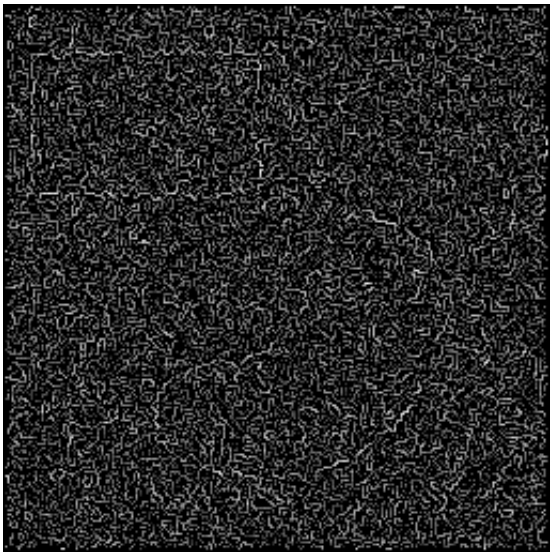

Figure 2.5 – Noise après

Figure 2.6 – Noise

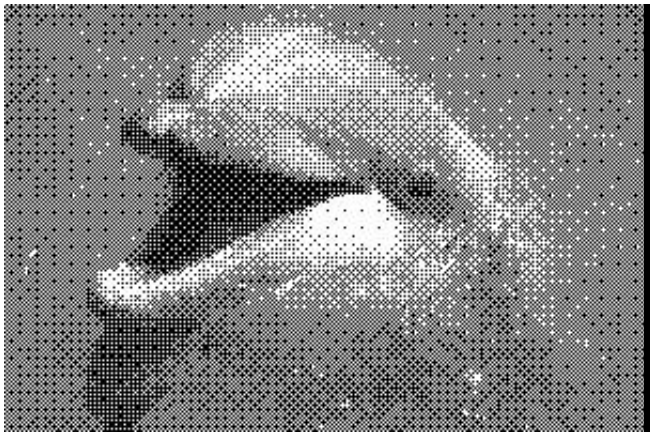

Figure 2.7 – Delphin avant

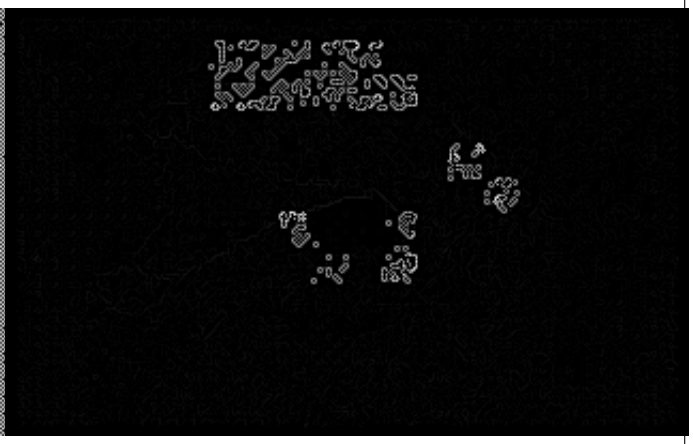

Figure 2.8 – Delphin après

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).

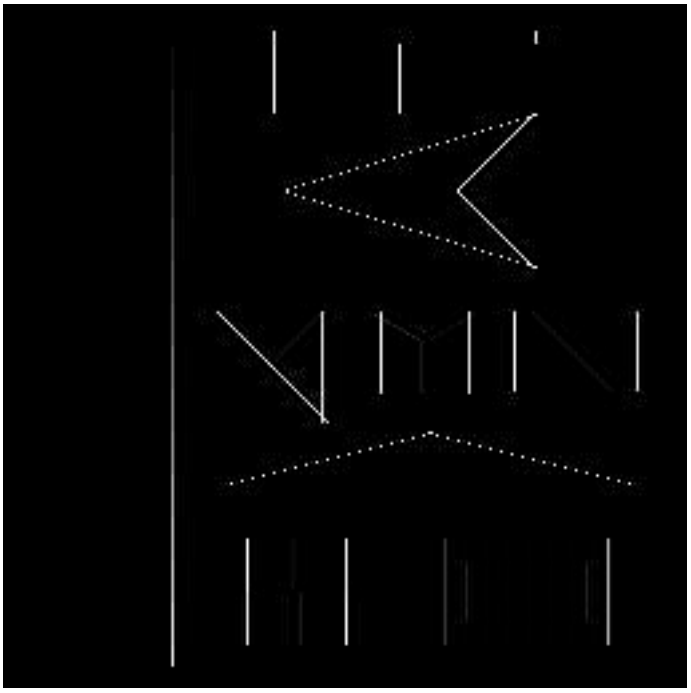

Figure 2.11 – Suzan vertical

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

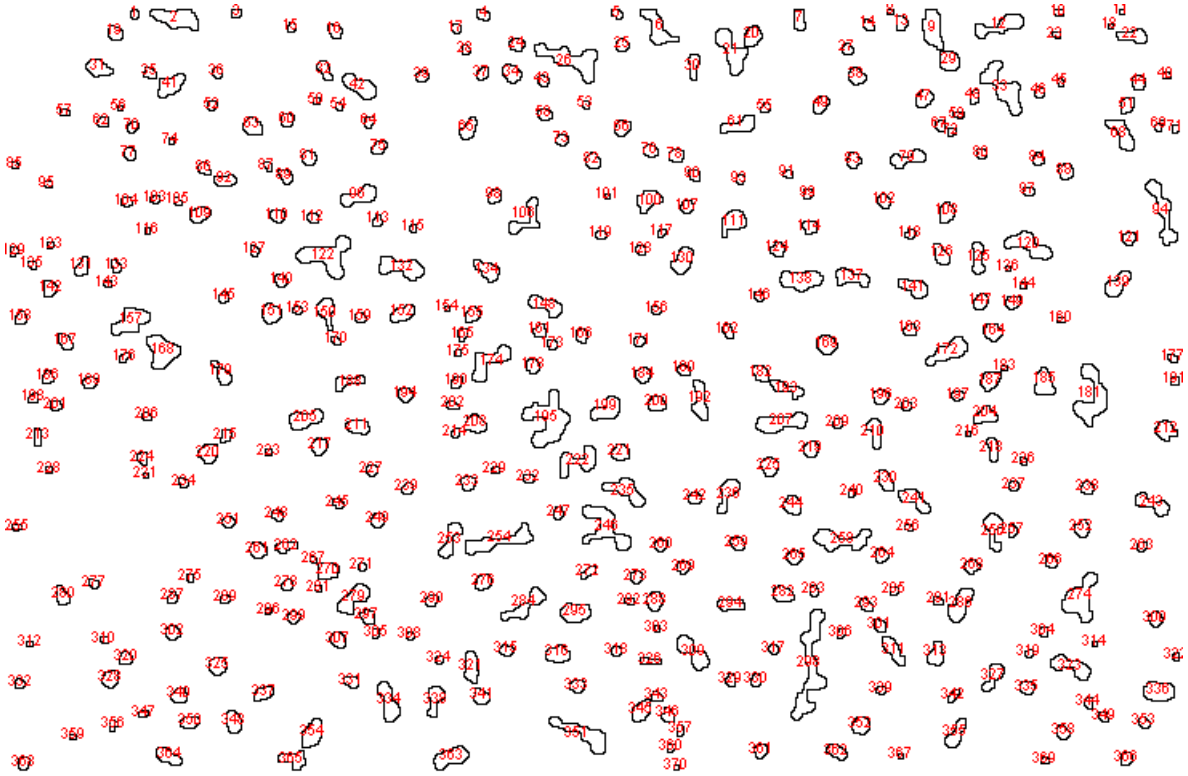

Figure 2.12 – Cell 370

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.

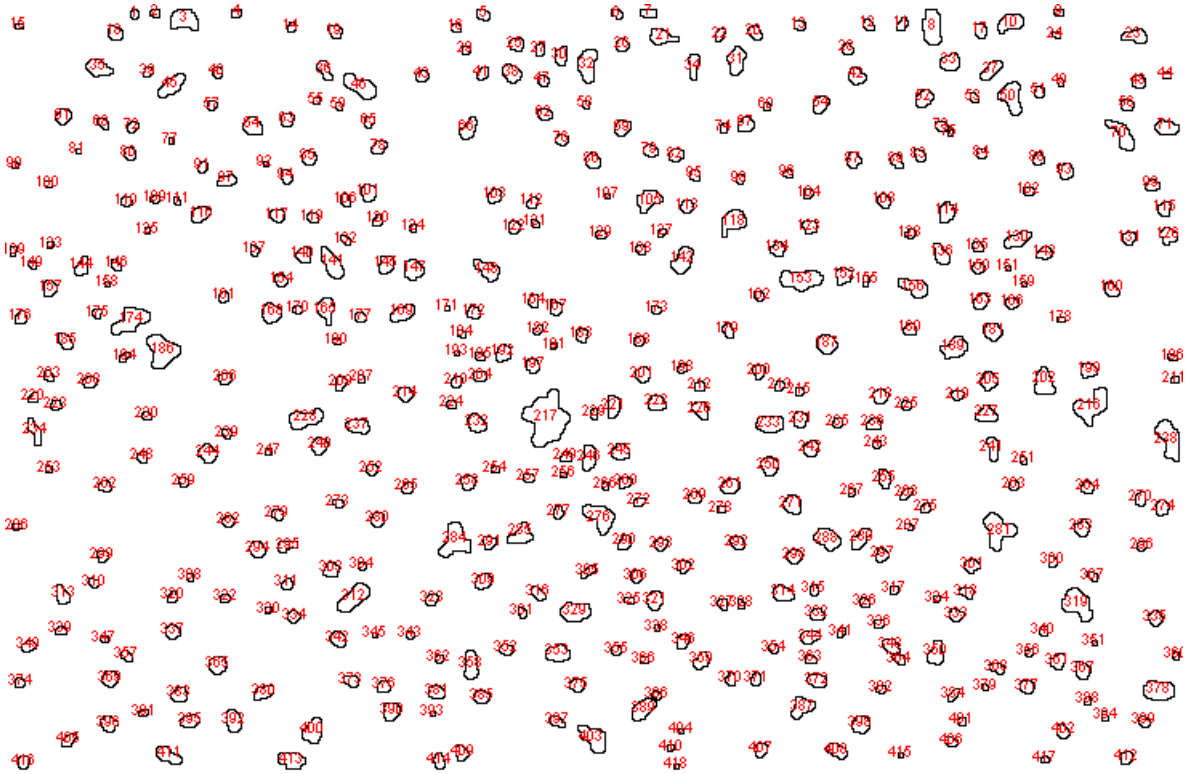

Figure 2.13 – Cell 415

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
21    public void run(ImageProcessor ip)
22    {
23       baseColors = new HashMap<Color, String>();
24       baseColors.put(new Color(255, 255, 0), "Jaune");
25
26       baseColors.put(new Color(0, 255, 0), "Vert");
27       baseColors.put(new Color(0, 189, 9), "Vert");
28       baseColors.put(new Color(105, 240, 112), "Vert");
29       baseColors.put(new Color(0, 118, 36), "Vert");
30       baseColors.put(new Color(72, 86, 33), "Vert");
31       baseColors.put(new Color(44, 58, 58), "Vert");
32
33       baseColors.put(new Color(255, 0, 0), "Rouge");
34       baseColors.put(new Color(255, 0, 66), "Rouge");
35       baseColors.put(new Color(199, 0, 39), "Rouge");
36       baseColors.put(new Color(129, 25, 0), "Rouge");
37
38       baseColors.put(new Color(0, 0, 255), "Bleu");
39       baseColors.put(new Color(17, 31, 58), "Bleu");
40       baseColors.put(new Color(119, 24, 255), "Bleu");
```

```
41      baseColors.put(new Color(91, 33, 74), "Bleu");
42      baseColors.put(new Color(117, 63, 121), "Bleu");
43      baseColors.put(new Color(0, 255, 255), "Bleu");
44      baseColors.put(new Color(0, 189, 195), "Bleu");
45      baseColors.put(new Color(0, 174, 255), "Bleu");
46      baseColors.put(new Color(134, 217, 255), "Bleu");
47      baseColors.put(new Color(98, 140, 255), "Bleu");
48
49      baseColors.put(new Color(102, 51, 0), "Marron");
50      baseColors.put(new Color(113, 76, 43), "Marron");
51
52      baseColors.put(new Color(128, 128, 128), "Gris");
53      baseColors.put(new Color(192, 192, 192), "Gris");
54      baseColors.put(new Color(64, 64, 64), "Gris");
55      baseColors.put(new Color(64, 64, 64), "Gris");
56      baseColors.put(new Color(30, 19, 17), "Gris");
57
58      baseColors.put(new Color(255, 255, 255), "Blanc");
59      baseColors.put(new Color(202, 212, 221), "Blanc");
60
61      baseColors.put(new Color(0, 0, 0), "Noir");
62      baseColors.put(new Color(2, 11, 12), "Noir");
63      baseColors.put(new Color(7, 18, 19), "Noir");
64
65      baseColors.put(new Color(255, 200, 0), "Orange");
66      baseColors.put(new Color(220, 74, 1), "Orange");
67      baseColors.put(new Color(234, 142, 119), "Orange");
68      getColors(ip.duplicate().convertToRGB());
69  }
70
71  private String getBeautifulColors(String title, Map<String, Integer> colors, int count, double
        threshold)
72  {
73      StringBuilder builder = new StringBuilder();
74      for(String color : colors.keySet())
75          if(colors.get(color) > threshold * count)
76              builder.append(color).append(":").append(String.format("%.2f%%", 100 * colors.get(color) /
                    (double) count)).append(", ");
77
78      if(builder.length() > 1)
79          builder.delete(builder.length() - 2, builder.length());
80
81      File outFile = new File(title + "_tag" + ".txt");
82      PrintWriter pw = null;
83      try
84      {
85          pw = new PrintWriter(new FileOutputStream(outFile));
86          try
87          {
88              pw.print("Quality: ");
89              for(String color : colors.keySet())
90                  if(colors.get(color) > threshold)
91                      builder.append(color).append(" ");
92              pw.println();
93          }
94          catch(Exception e)
95          {
96              e.printStackTrace();
97          }
98      }
99      catch(Exception e)
100     {
101         e.printStackTrace();
102     }
103     finally
104     {
105         if(pw != null)
106             pw.close();
107     }
```

```
108
109        return builder.toString();
110     }
111
112     private HashMap<String, Integer> getColors(ImageProcessor ip)
113     {
114        ImageProcessor ip2 = ip.createProcessor(ip.getWidth(), ip.getHeight());
115        ip.medianFilter();
116        ip.setColorModel(ColorModel.getRGBdefault());
117        HashMap<String, Integer> colors = new HashMap<String, Integer>();
118        for(int i = 0; i < ip.getWidth(); i++)
119        {
120           for(int j = 0; j < ip.getHeight(); j++)
121           {
122              Color c = getClosestColor(i, j, ip.getColorModel().getRGB(ip.getPixel(i, j)));
123              String colorName = baseColors.get(c);
124              if(!colors.containsKey(colorName))
125                 colors.put(colorName, 0);
126              colors.put(colorName, colors.get(colorName) + 1);
127              ip2.putPixel(i, j, c.getRGB());
128           }
129        }
130
131        displayImage("Coleurs: " + getBeautifulColors(WindowManager.getActiveWindow().getName(), colors,
               ip.getWidth() * ip.getHeight(), 0.1), ip2);
132
133        return colors;
134     }
135
136     private void displayImage(String title, ImageProcessor imageProcessor)
137     {
138        final ImageWindow iw = new ImageWindow(new ImagePlus(WindowManager.makeUniqueName(title),
               imageProcessor));
139        iw.addWindowListener(new WindowAdapter(){
140           @Override
141           public void windowClosed(WindowEvent e)
142           {
143              super.windowClosed(e);
144              WindowManager.removeWindow(iw);
145           }
146        });
147        WindowManager.addWindow(iw);
148     }
149
150
151     private Color getClosestColor(int x, int y, int i)
152     {
153        double minDist = Double.MAX_VALUE;
154        Color bestColor = null;
155
156        Color c = new Color(i);
157        float hsb1[] = new float[3];
158        Color.RGBtoHSB(c.getRed(), c.getGreen(), c.getBlue(), hsb1);
159
160        for(Color c2 : baseColors.keySet())
161        {
162           float hsb2[] = new float[3];
163           Color.RGBtoHSB(c2.getRed(), c2.getGreen(), c2.getBlue(), hsb2);
164           double dist = getDistanceHSB(hsb1, hsb2);
165           if(dist < minDist)
166           {
167              minDist = dist;
168              bestColor = c2;
169           }
170        }
171
172        return bestColor;
173     }
174
```

```
175    private double getDistanceHSB(float[] hsb1, float[] hsb2)
176    {
177       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));
178    }
179
180    public int setup(String arg, ImagePlus imp)
181    {
182       if(arg.equals("about"))
183       {
184          IJ.showMessage("Traitement de l'image v2");
185          return DONE;
186       }
187       return DOES_RGB + DOES_8G + DOES_16 + DOES_32;
188    }
189 }
```

Afin de récupérer les couleurs dominantes, nous parcourons chaque pixels. 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 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 donc pas reconnus comme bleu mais plutôt comme une couleur clair, gris ou blanc.

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.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 PluginQuality_ implements PlugInFilter
18 {
19    public void run(ImageProcessor ip)
20    {
21       getIntensity(ip.duplicate().convertToRGB());
22    }
23
24    private String getBeautifulIntensity(String title, Map<String, Double> colors, double threshold)
25    {
26       StringBuilder builder = new StringBuilder();
27       for(String color : colors.keySet())
28          if(colors.get(color) > threshold)
29             builder.append(color).append(":").append(String.format("%.2f%%", colors.get(color) *
                100)).append(", ");
30
31       if(builder.length() > 1)
32          builder.delete(builder.length() - 2, builder.length());
33
34       File outFile = new File(title + "_tag" + ".txt");
35       PrintWriter pw = null;
36       try
37       {
38          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, Double> getIntensity(ImageProcessor ip)
{
   ImageProcessor ip2 = ip.createProcessor(ip.getWidth(), ip.getHeight());
   ip.medianFilter();
   ip.setColorModel(ColorModel.getRGBdefault());
   HashMap<String, Double> colors = new HashMap<String, Double>();
   for(int i = 0; i < ip.getWidth(); i++)
   {
      for(int j = 0; j < ip.getHeight(); j++)
      {
         Color c = new Color(ip.getColorModel().getRGB(ip.getPixel(i, j)));
         float hsb1[] = new float[3];
         Color.RGBtoHSB(c.getRed(), c.getGreen(), c.getBlue(), hsb1);

         String colorName = hsb1[2] < 0.33 ? "Sombre" : (hsb1[2] > 0.66 ? "Clair" : "Normal");
         if(!colors.containsKey(colorName))
            colors.put(colorName, 0D);
         colors.put(colorName, colors.get(colorName) + 1);
         ip2.putPixel(i, j, c.getRGB());
      }
   }

   double max = 0;
   for(double count : colors.values())
      max = Math.max(max, count);
   for(String key : colors.keySet())
      colors.put(key, colors.get(key) / max);

   IJ.showMessage("Intensité: " + getBeautifulIntensity(WindowManager.getActiveWindow().getName(),
       colors, 0.6));

   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);
   }

   public int setup(String arg, ImagePlus imp)
   {
      if(arg.equals("about"))
      {
         IJ.showMessage("Traitement de l'image v2");
         return DONE;
      }
      return DOES_RGB + DOES_8G + DOES_16 + DOES_32;
   }
}
```

Listing 3.5 – PluginEdges_.java

```
import ij.IJ;
import ij.ImagePlus;
import ij.WindowManager;
import ij.gui.ImageWindow;
import ij.plugin.filter.Convolver;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageConverter;
import ij.process.ImageProcessor;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.Random;

public class PluginEdges_ implements PlugInFilter
{
   public void run(ImageProcessor ip)
   {
      new ImageConverter(new ImagePlus("Title", ip.duplicate())).convertToGray8();
      convolve(ip.convertToFloatProcessor());
   }

   private void convolve(ImageProcessor ip)
   {
      ip.smooth();
      ImageProcessor ip2 = ip.duplicate();
      Convolver cv = new Convolver();
      cv.convolve(ip, new float[]{
            -1, 0, 1,
            -2, 0, 2,
            -1, 0, 1
      }, 3, 3);
      cv.convolve(ip2, new float[]{
            -1, -2, -1,
            0, 0, 0,
            1, 2, 1
      }, 3, 3);

      for(int i = 0; i < ip.getWidth(); i++)
         for(int j = 0; j < ip.getHeight(); j++)
            ip.putPixelValue(i, j, Math.sqrt(Math.pow(ip.getPixelValue(i, j), 2) +
                Math.pow(ip2.getPixelValue(i, j), 2)));
      displayImage("Contours", ip);
   }

   private void displayImage(String title, ImageProcessor imageProcessor)
   {
      final ImageWindow iw = new ImageWindow(new ImagePlus(WindowManager.makeUniqueName(title + new
          Random().nextInt()), imageProcessor));
      iw.addWindowListener(new WindowAdapter(){
```

```
47          @Override
48          public void windowClosed(WindowEvent e)
49          {
50             super.windowClosed(e);
51             WindowManager.removeWindow(iw);
52          }
53       });
54       WindowManager.addWindow(iw);
55    }
56
57    public int setup(String arg, ImagePlus imp)
58    {
59       if(arg.equals("about"))
60       {
61          IJ.showMessage("Traitement de l'image v2");
62          return DONE;
63       }
64       return DOES_RGB + DOES_8G + DOES_16 + DOES_32;
65    }
66 }
```

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.